# 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, USA

## Abstract

*The state of neural network pruning has been noticed to be unclear and even confusing for a while, largely due to "a lack of standardized benchmarks and metrics" [3]. To standardize benchmarks, first, we need to answer:* ***what kind of comparison setup is considered fair****? This basic yet crucial question has barely been clarified in the community, unfortunately. Meanwhile, we observe several papers have used (severely) sub-optimal hyper-parameters in pruning experiments, while the reason behind them is also elusive. These sub-optimal hyper-parameters further exacerbate the distorted benchmarks, rendering the state of neural network pruning even more obscure.*

*Two mysteries in pruning represent such a confusing status: the performance-boosting effect of a larger finetuning learning rate, and the no-value argument of inheriting pretrained weights in filter pruning.*

*In this work, we attempt to explain the confusing state of network pruning by demystifying the two mysteries. Specifically, (1) we first clarify the fairness principle in pruning experiments and summarize the widely-used comparison setups; (2) then we unveil the two pruning mysteries and point out the central role of* ***network trainability****, which has not been well recognized so far; (3) finally, we conclude the paper and give some concrete suggestions regarding how to calibrate the pruning benchmarks in the future. Code: https://github.com/mingsun-tse/why-the-state-of-pruning-so-confusing.*

## 1. Introduction

The past decade has witnessed the great success of deep learning, empowered by deep neural networks [38, 68]. The success comes at the cost of over-parameterization [5, 12, 26, 31, 32, 36, 61, 62, 66, 69, 75], causing prohibitive model footprint, slow inference/training speed, and rapid-growing energy consumption. Neural network pruning, an old model parameter reduction technique that used to be employed for improving generalization [2, 6, 64], now is mostly used for model size compression or/and speed acceleration [7, 8, 11, 20, 30, 73].

The prevailing pipeline of pruning comprises three steps: 1) **pretraining**: train a dense model; 2) **pruning**: prune the dense model based on certain rules; 3) **finetuning**: retrain the pruned model to regain performance. Most existing research focuses on the second step, seeking better criteria to remove unimportant weights so as to incur as less performance degradation as possible. This 3-step pipeline has been practiced for more than 30 years [2, 6, 39, 57] and is still extensively adopted in today's pruning methods [30, 73].

Despite the long history of network pruning, recent progress seems a little bit unclear. As we quote from the

Table 1. Top-1 accuracy (%) benchmark of filter pruning with **ResNet50** [26] on **ImageNet** [10]. Simply by using a better finetuning LR schedule, we manage to revive a *6-year-ago* baseline filter pruning method, *$L_1$-norm pruning* [42], 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 paper studies the reasons and lessons behind this pretty confounding benchmark situation in filter pruning.

| Method | Pruned acc. (%) | Speedup |
|---|---|---|
| SFP [28] IJCAI'18 | 74.61 | 1.72× |
| DCP [88] NeurIPS'18 | 74.95 | 2.25× |
| GAL-0.5 [47] CVPR'19 | 71.95 | 1.76× |
| Taylor-FO [55] CVPR'19 | 74.50 | 1.82× |
| CCP-AC [60] ICML'19 | 75.32 | 2.18× |
| ProvableFP [45] ICLR'20 | 75.21 | 1.43× |
| HRank [46] CVPR'20 | 74.98 | 1.78× |
| GReg-1 [79] ICLR'21 | 75.16 | 2.31× |
| GReg-2 [79] ICLR'21 | 75.36 | 2.31× |
| CC [44] CVPR'21 | **75.59** | 2.12× |
| $L_1$-norm [42] ICLR'17 **(our reimpl.)** | 75.24 | **2.31**× |
| GAL-1 [47] CVPR'19 | 69.88 | 2.59× |
| Factorized [43] CVPR'19 | 74.55 | 2.33× |
| LFPC [27] CVPR'20 | 74.46 | 2.55× |
| GReg-1 [79] ICLR'21 | 74.85 | 2.56× |
| GReg-2 [79] ICLR'21 | **74.93** | 2.56× |
| CC [44] CVPR'21 | 74.54 | **2.68**× |
| $L_1$-norm [42] ICLR'17 **(our reimpl.)** | 74.77 | 2.56× |

[†]Corresponding author: wang.huan@northeastern.edu

abstract of a recent paper [3], "*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*". We also sense this kind of confusion. Particularly, two mysteries in the area represent such confusion:

- **Mystery 1 (M1): The performance-boosting effect of a larger finetuning learning rate (LR)**. It *was* broadly believed that finetuning the pruned model should use a *small* LR, *e.g.*, in the famous $L_1$-norm pruning [42], finetuning LR 0.001 is used for the ImageNet experiments. However, many other papers choose a larger LR, *e.g.*, 0.01, which delivers significantly better performance than 0.001. For a pretty long time, pruning papers do not officially investigate this critical performance-boosting effect of a larger finetuning LR (although they may have *already used* it in their experiments; see Tab. 3), until this paper [37]. [37] formally studies how different finetuning LR schedules affect the final performance. They find random pruning and magnitude-pruning (the two most basic pruning methods) armed with a good finetuning LR schedule (CLR, or Cyclic Learning Rate Restarting) can counter-intuitively rival or even surpass other more sophisticated pruning algorithms. Unfortunately, they do not give explanations for this phenomenon except for calling upon comparing pruning algorithms in the same retraining configurations.

- **Mystery 2 (M2): The value of network pruning**. The value of network pruning seems unquestionable given the development history of over 30 years. However, two papers [9, 50] empirically find that training the pruned model from scratch can match pruning a pretrained model, thus radically challenging the necessity of pretraining a big model first in the conventional 3-step pruning pipeline.

Tab. 1 shows a concrete example that M1 makes the pruning benchmarks unclear. After using an improved finetuning LR schedule (see Tab. 3), we make $L_1$-norm pruning [42], which is broadly considered the *most basic baseline* approach, *match or beat* many top-performing pruning methods published in top-tier conferences in the past several years. Such a situation really bewilders us, especially for those not in this area trying to borrow the most advanced pruning methods in their projects – *"Has the area really developed in the past several years?"* they might question.

This paper is meant to unveil these mysteries. Over this process, we hope to offer some helpful thoughts about why we have run into the current chaotic benchmark situation and how we can avoid it in the future.

Specifically, when we try to unveil the mysteries, only to find there are various comparison setups enforced in the area, many of the conclusions actually *hinge on which comparison setup is used*. To decide which comparison setup is more trustworthy, we have to be clear with the *fairness principle* in pruning experiments first (*i.e.*, what kind of comparison setup is considered fair?). After we sort out the fairness principle and comparison setups, we empirically examine the two mysteries and find M2 reduces to M1. When examining M1, we find the role of *network trainability* in network pruning, through which we can easily explain M1.

Therefore, our investigation path and the contributions in this paper can be summarized as follows.

- We first clarify the fairness principle in pruning experiments and summarize the outstanding comparison setups (Sec. 3), which have been unclear in the literature for a long time.

- Then, we start to unveil M1 [37] and M2 [50]. As we shall show (Sec. 4), the conclusion of M2 actually varies up to which comparisons setup is used: If a larger finetuning LR is allowed to be used, the no-value-of-pruning argument cannot hold; otherwise, it mostly holds. Thus, to unveil M2, we have to unveil M1 first.

- Next, we focus on unveiling M1. We introduce the perspective of network trainability to diagnose pruning and clearly explain why the finetuning LR has a significant impact on the final performance (Sec. 5) – to our best knowledge, we are the *first* to clarify this mystery in the area.

- Finally, we summarize the major reasons that have led to the confusing benchmark situation of network pruning up to now, and give some concrete suggestions about how to avoid it in the future (Sec. 6).

## 2. Prerequisites

### 2.1. Taxonomy of Pruning and Related Work

A pruning algorithm *has and only has* five exclusive aspects to be specified: (1) Base model (when to prune): is pruning conducted on a pretrained model or a random model? (2) Sparsity granularity (what to prune): what is the smallest weight group in pruning? (3) Pruning ratio (how many to prune): how many weights are to be pruned? (4) Pruning criterion (by what to prune): what measure is used to select the important weights (*i.e.*, those to be kept) *vs.* the unimportant weights (*i.e.*, those to be pruned)? (5) Pruning schedule (how to schedule): How is the sparsity scheduled over the pruning process?

These five mutually orthogonal questions can shatter most (if not all) pruning papers in the literature. Researchers typically use (1), (2), and (4) to classify different pruning methods. We give the major backgrounds in these three axes.

**Pruning after training *vs*. pruning at initialization**. Pruning has been mostly conducted on a *pretrained* model over the past 30 years, which is thus called *pruning after training* or *post-training pruning*. This fashion has been the unquestioned norm until (at least) two papers in 2019, SNIP [41] and LTH [17]. They argue pruning can be conducted on *randomly initialized* models and can achieve promising performance (allegedly matching the dense counterpart), too. This new fashion of pruning is called *pruning at initialization* (PaI). Existing PaI approaches mainly include [19, 40, 41, 63, 77] and the series of lottery ticket hypothesis [17, 18]. PaI is not very relevant to this work because the benchmarking chaos and the mysteries are mostly discussed in the PaT context, so here we would not discuss in length the specific PaI techniques. Interested readers may refer to [80] for a comprehensive summary.

**Sparsity structure: structured pruning *vs*. unstructured pruning**. If the smallest weight group in pruning is a single weight element, this kind of pruning is called unstructured (or fine-grained) pruning [17, 22, 23], because the resulting zero-weight (*i.e*., pruned-weight) locations are typically irregular (if no extra regularization is enforced). If the smallest weight group in pruning presents some structure, this kind of pruning is called structured (or structural/coarse-grained) pruning [29, 42, 53, 82]. In the area, structured pruning typically narrowly refers to filter pruning or channel pruning if not explained otherwise. Structured pruning benefits more acceleration because the regular sparsity pattern is more hardware-friendly; meanwhile, the regularity imposes more constraints on the network, so given the same sparsity level, structured pruning typically underperforms unstructured pruning.

Note, the definition of "structured" sparsity is severely hardware-dependent and thus can vary as the hardware condition changes. *E.g*., the *N:M sparsity*[1] pioneered by NVIDIA Ampere architecture was considered as unstructured sparsity, but since NVIDIA has launched new library support to exploit such kind of sparsity for acceleration, now it can be considered as structured sparsity (called fine-grained structured sparsity [58, 86]), too.

This paper focuses on *filter pruning* for now because the two aforementioned mysteries (the effect of finetuning LR and the value of network pruning) are mainly discussed in this context.

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

**Pruning criterion: Importance-based *vs*. regularization based**. The former prunes weights based on some established importance criteria, such as magnitude (for unstructured pruning) [22, 23] or $L_1$-norm (for filter pruning) [42], saliency based on 2nd-order gradients (*e.g*., Hessian or Fisher) [24, 39, 70, 74, 76]. The latter adds a penalty term to the objective function, drives unimportant weights toward zero, then removes those with the smallest magnitude. Two notable points: (1) Even in a regularization-based pruning method, after the regularization process, the weights are still removed by a certain importance (typically magnitude). Namely, regularization-based pruning inherently embeds importance-based pruning. (2) The two paradigms are not exclusive; they can be employed simultaneously. *E.g*., [13, 79, 81] select unimportant weights by magnitude while also employing the regularization to penalize weights.

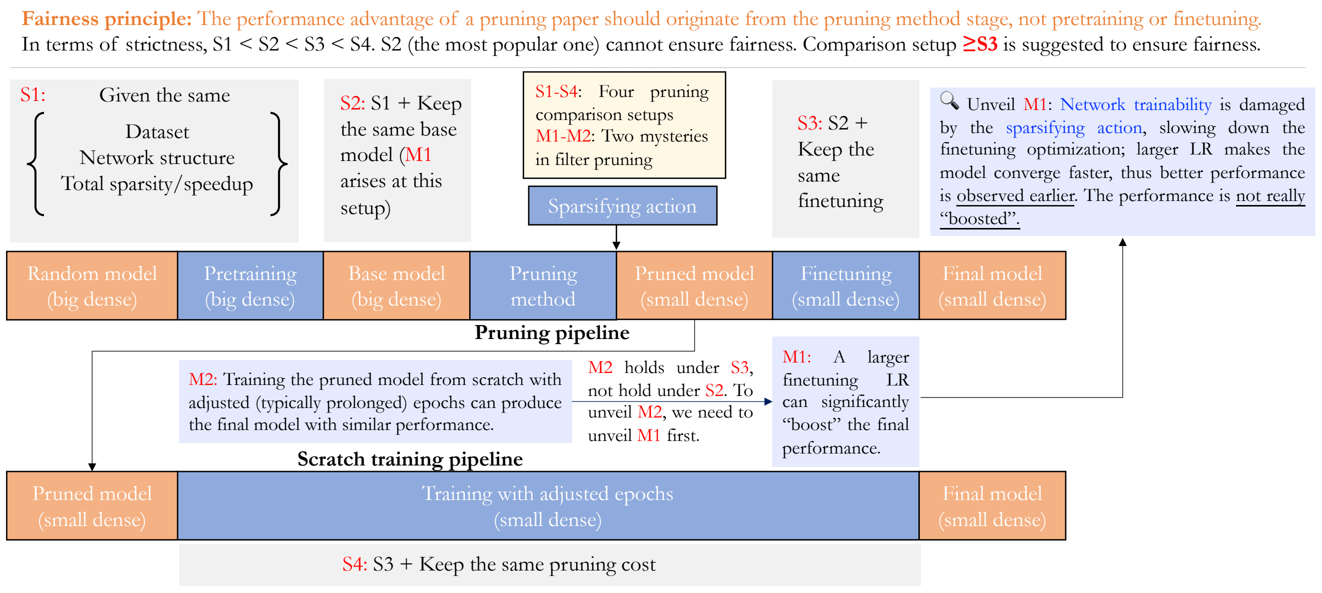
Finally, for more comprehensive literature on network pruning, we refer interested readers to several surveys: an early one [64], some recent surveys of pruning alone [3, 20, 30] or pruning as a sub-topic under the general umbrella of model compression and acceleration [7, 8, 11, 73].

**Most relevant papers**. The most relevant works to this paper are

- [3], which is the first to systematically report the frustrating state of network pruning benchmarks, identify some causes (such as the lack of standard benchmarks and metrics in pruning), and give concrete remedies. However, they do not go deeper and analyze why the standard benchmarks are hard to achieve. Our paper succeeds [3] and will elaborate more on this aspect and point out the central role of trainability within.

- [37], which officially reports the performance-boosting effect of a larger finetuning LR and calls upon comparing pruning algorithms under the same retraining configurations. [37] is actually motivated by [65], which proposes *LR rewinding vs*. the *weight rewinding* [18] proposed for finding winning ticket on large-scale networks and datasets. [65] virtually takes advantage of the performance-boosting effect of a larger finetuning LR, yet we do not know if they were aware at that point that the performance boosting is not because of the magic rewound LR schedule but simply because of a larger finetuning LR (*i.e*., any appropriately larger LR would do, even not a rewound one), as later clarified by [37]; so we tentatively consider [37] as the *first* work to systematically report the performance-boosting effect of a larger finetuning LR.

- [50], which brings forward the argument regarding the value of network filter pruning against scratch training. We will re-evaluate the major claim (scratch training



Fairness principle: The performance advantage of a pruning paper should originate from the pruning method stage, not pretraining or finetuning.
In terms of strictness, S1 < S2 < S3 < S4. S2 (the most popular one) cannot ensure fairness. Comparison setup ≥S3 is suggested to ensure fairness.
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

Figure 1. Overview of this paper. We are motivated by unveiling two mysteries (`M1`, `M2`) in filter pruning, which represent the confusing pruning benchmark situation. We first clarify the *fairness principle* and summarize outstanding *comparison setups* to lay down the discussion foundation (the notation "`S1` < `S2`" means `S1` is *less strict* than `S2`; others can be inferred likewise). Then we start to unveil `M1` and `M2`. `M2` will be shown to reduce to `M1`, actually. To unveil `M1`, we introduce *network trainability* as an effective perspective to demystify `M1`. The unawareness of the role of network trainability in pruning has actually led to several sub-optimal hyper-parameter settings, which exacerbates the chaotic benchmark status. We finally give some concrete suggestions to calibrate the pruning benchmarks.

can match filter pruning) of this paper under our more strictly controlled comparison setups.

## 2.2. Terminology Clarification

First, we make some critical concepts clear to lay down the common ground for discussion. Although they are pretty simple concepts, misinterpreting them will twist our discussions.

- ***Pruning pipeline* vs. *pruning method* vs. *sparsifying action*.** Some papers refer to *pruning* as the whole pruning algorithm, *i.e.*, the 2nd step in the pruning pipeline; while others may mean a pruning paper or the instant sparsifying action. We realize such a vague conception definition is one reason causing confusion, so we make them exact here. We use *pruning pipeline* to mean all the three steps in a pruning paper. We can consider *pruning pipeline* to be interchangeable with a *pruning paper*. Then, we use *pruning method* = *pruning algorithm* to mean the 2nd step of the pruning pipeline. Finally, the instant pruning action (*i.e.*, zeroing out weights or physically taking away weights from a network) is referred to as *sparsifying action*. To summarize, in terms of concept scope, pruning paper = pruning pipeline = pruning > pruning method = pruning algorithm > sparsifying action[2].

[2]The notation "pruning paper = pruning pipeline > pruning method" means, in this paper we consider *pruning paper* interchangeable with *pruning pipeline*, which includes *pruning method* as one part.

- ***Training from scratch*.** *Training from scratch* = *scratch training*, means to train a randomly initialized model to convergence. Scratch training of a *pruned model* means, we already know the network architecture of the pruned model; the weights are randomly initialized; train this network from scratch using *the same* training recipe as training the dense model. Notably, for filter pruning, when the architecture of the pruned model is known, the model should be implemented as a small-dense model, *not a large-sparse model* (with structural masks). The reason is, the widely-used parameter initialization schemes (*e.g.*, He initialization [25], the default initialization scheme for CONV and Linear layers in PyTorch [59]) depend on the parameter shape. The large-sparse implementation is *not* equivalent to (often underperforms) the small-dense implementation. For unstructured pruning, the standard implementation scheme is large-sparse weights (with unstructured masks) [50, 79].

- ***Finetuning*.** After the sparsifying action action, the subsequent training process is called *finetuning* or *retraining*. We notice the community seems to have different interpretations about these two terms. *E.g.*, in [37], finetuning is a *sub-concept* of retraining, specifically meaning retraining with the last (smallest) learning rate of original training[3]; while many

[3] [37] attributes this term usage to [23, 42, 48]. We double-checked these

more papers [50, 56, 76, 79, 82] consider finetuning *the same as* retraining, meaning the 3rd step of the pruning pipeline. In this paper, we take the more common stance: considering finetuning and retraining interchangeable, and in the end of this paper, we will show the term "finetuning" should be deprecated in favor of "retraining".

- ***Scratch-E, Scratch-B***. These two terms are from [50], denoting two scratch training schemes. "E" is short for epochs, "B" short for (computation) budget. In practice, [50] uses FLOPs as an approximation for the computation budget. In Scratch-E, the point is to maintain the same *total epochs* when comparing scratch training to pruning. In Scratch-B, the point is to maintain the same *total FLOPs* when comparing scratch training to pruning. Here is a concrete example of Scratch-B: a dense model has FLOPs $F_1$; pretraining the dense model takes $K_1$ epochs; it is pruned by $L_1$-norm pruning, giving a pruned model with FLOPs $F_2$; the finetuning takes another $K_2$ epochs; then the scratch training should take $(K_1F_1 + K_2F_2)/F_2 = K_1(F_1/F_2) + K_2$ epochs. The ratio $F_1/F_2$ is typically called *speedup* in network pruning.

- ***Value of network pruning***. This term comes from [50]. The conclusion of [50] is that scratch training can match the performance of the 3-step pruning pipeline if Scratch-B is adopted, for filter pruning. Therefore, they argue there is no value for filter pruning algorithms that *use predefined layerwise pruning ratios*. For the filter pruning algorithms that do not use predefined layerwise pruning ratios, their role is to decide the favorable network architectures, akin to NAS [15, 89]. As for unstructured pruning, [42] shows scratch training *cannot* match pruning. Therefore, rigorously, the argument about the value of network pruning means the value of inheriting pretrained weights in filter pruning with predefined layerwise pruning ratios. We will use the short notion, *value of network pruning*, without mentioning its much richer context.

## 3. Fairness and Comparison Setups in Pruning

### 3.1. *Fairness Principle* in Network Pruning

This section is prepared for the next section, Sec. 3.2, where we will summarize the major comparison setups in pruning. As we shall see in the experiments, a pruning algorithm `A` can be better than `B` under one comparison setup, while worse than or on par with `B` under another comparison setup. To decide which comparison is more trustworthy, we have to evaluate which comparison setup is *fairer*. Thence comes the necessity of a clear fairness principle in pruning.

***Fairness Principle***. *The performance advantage of a pruning paper should be attributed to the pruning stage, not finetuning or pretraining.* As aforementioned (Sec. 2), a pruning algorithm exclusively has five aspects [30, 80]: base model, sparsity granularity, pruning ratio, pruning criterion, and pruning schedule. Excluding the base model and sparsity granularity axes[4], therefore, **when we say a pruning method is better than another one, the performance advantage should be attributed to *at least one of the three aspects*; namely, a better pruning ratio scheme, or/and a better pruning criterion, or/and a better pruning schedule**. Otherwise, it means the performance advantage comes from some outside factors other than the pruning algorithm itself – in this case, attributing the performance credit to the pruning method would be an unjustified claim, potentially leading to an unfair comparison.

Per this fairness principle, clearly, we should keep the same base model and the finetuning process (*i.e.*, the 1st and 3rd steps) in the pruning pipeline. Next, we elaborate on the outstanding comparison setups in network pruning and examine their fairness.

### 3.2. Comparison Setups in Network Pruning

In the literature, we find there are at least the following four groups of comparison setups, summarized in Tab. 2. Note, in the following discussion, we consider different pruning algorithms (and scratch training) that remove or zero out unimportant weights *only once*. Namely, we do not consider *iterative pruning* for now, and we will discuss how the conclusions can carry over to iterative pruning.

**Historical contexts of Tab. 2.** In the following paragraphs, we briefly go through the historical context of the different comparison setups in Tab. 2.

**`S1`**. Obviously, the `S1` setup is the most basic one, and also the earliest one. It simply compares performance regardless of many factors, such as training epochs and even the base model. This setup is adopted by early pruning papers, especially those using Caffe [33], *e.g.*, [22, 23, 29, 82]. Such a comparison setup does not even demand the same base model, which we may consider problematic today; yet back in those days, they had their own reasons – before the deep learning (DL) community had mature DL platforms/tools/computation power as we have today, these papers usually *trained their own base models*. Consequently, due to different implementations, their base models do not have the same (or even close) accuracy, *e.g.*, the VGG16

three papers and found they do not evidently have the inclination to mean finetuning as one particular type of retraining.

[4]Most pruning works do not consider the performance gain due to a better base model or sparsity granularity as a valid advantage over other methods, because a pruning method can be easily applied to different base models and sparsity granularities (*e.g.*, SSL [82]) although the paper may only focus on one kind.

Table 2. Summary of popular comparison setups in pruning papers. It is helpful to review them along with the 3-step pruning pipeline: pretraining (output: base model) ⇒ pruning (output: pruned model) ⇒ finetuning (output: final small model). In terms of strictness, `S1` $<$ `S2` $<$ `S3.1` $<$ `S3.2` $<$ `S4.1` $<$ `S4.2` (the notation "`S1` $<$ `S2`" means `S1` is *less strict* than `S2`; others can be inferred likewise). Most existing pruning papers follow the `S2` comparison setup.

| **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<br>+Same finetuning epochs |
| `S3.2` | +Same base model<br>+Same finetuning LR schedule |
| `S4.1` | +Same base model<br>+Same finetuning LR schedule<br>+Same pruning epochs |
| `S4.2` | +Same base model<br>+Same finetuning LR schedule<br>+Same pruning LR schedule |
| `SX-A` | +Same epochs of "pretraining + pruning + finetuning" |
| `SX-B` | +Same FLOPs of "pretraining + pruning + finetuning" |

base model reported by ThiNet [52] has top-5 accuracy 88.44% while CP [29], a concurrent work with ThiNet, reported 89.9% (for those who are not familiar with these numbers, 1.5% top-5 accuracy is a *very significant* gap for ImageNet-1K classification).

As a remedy, to make the results comparable, many papers report the relative *performance drop*, namely, base model accuracy minus final model accuracy. Such an idea is still broadly practiced at present [50, 79], *esp.* when comparing methods that are implemented under quite different conditions.

**`S2`**. Later, as the DL community develops, more DL platforms *e.g.*, PyTorch [59] and TensorFlow [1] mature. There is usually a well-accepted model zoo (such as torchvision models[5]) for others to use. As a result, more and more pruning papers adopt them as the base models, such as [42, 79], which has become the mainstream practice at present. Thus, the `S2` comparison setup arises.

At this stage, few researchers have noticed the importance of finetuning. This makes sense since, in the pruning pipeline, only the pruning method part (*i.e.*, the 2nd step) is regarded as the central one. The finetuning process is often so *downplayed* that many papers do not even clearly report the hyper-parameters, as also noted by [3].

**`S3`**. Later, in the endless pursuit of higher and higher performance, there is a clear trend that the finetuning epochs become longer and longer (see Tab. 3 for an incomplete summary). This in effect renders the comparison more and more unfair. Besides, the finetuning LR has been noticed

[5]https://pytorch.org/vision/stable/models.html

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

to have a significant impact on the final performance, as formally studied by [37] (although [37] is the first one to formally study this phenomenon, a larger finetuning LR has been employed by many papers even before). Because of these, the finetuning process must be taken into account to maintain fairness.

The most exact way to rule out the impact of finetuning is to use exactly the same finetuning process – the same LR schedule (including the same epochs; we omit the hyper-parameters, like weight decay, momentum, *etc*., and assume they are maintained the same), *i.e*., the `S3.2` in Tab. 2. However, due to various objective or subjective reasons (*e.g*., prior papers may not release their finetuning details, making the follow-ups unable to reproduce the same finetuning), `S3.2` is often impractical. Thence comes a weaker setup `S3.1`, which only keeps the same epochs of finetuning. It is allowed to use different finetuning LR schedules (*e.g*., different initial LR) – *this is where* `M1`*, the mystery of the finetuning LR effect, arises.*

Several papers (such as [78, 79, 84, 85]) have ablative analysis experiments on small-scale datasets (*e.g*., CIFAR10 [35]) using the setup `S3.2`, while the main benchmark experiments (*e.g*., with ResNet-50 on ImageNet) using `S2`. The primary reason is that, following the setup `S3.2` means re-running the experiments for other comparison methods in the large-scale benchmarks, which is usually impractical (too costly) or even impossible (*e.g*., the comparison methods do not release usable code).

**`S4`**. The `S3.2` is still not the most strictly fair setup since it does not consider the cost (measured by the number of epochs) of the pruning method. For one-shot pruning (such as $L_1$-norm pruning [42]), the cost of pruning is zero; while for a regularization-based method (such as GReg [79]), it may take another few epochs for regularized training. Considering these cases, `S4.2` comes out: it builds upon `S3.2` and demands the same LR schedule for the pruning algorithm – *as far as we know, this is the most strict comparison setup*. In practice, again, for various reasons, we may not know the LR schedule of a pruning algorithm. Then, `S4.2` degrades to `S4.1`, which only demands the same epochs.

**`SX`**. In setups `S2` to `S4.2`, when comparing pruning to scratch training in obtaining the same pruned (small) model, the scratch training employs the same training recipe of obtaining the base (big) model. [50] challenges this practice. They argue, the scratch training scheme spends less cost than pruning, so the comparison is unfair. As a remedy, they propose to take into account the cost of the *pretraining* stage, which gives the `SX-A` and `SX-B` setups. About the *cost*, one way to measure it is to use the number epochs (hence the `SX-A`); another is to consider the same computation and they approximate computation with FLOPs (hence the `SX-B`).

It is hard to say if considering the cost of the pretraining stage is really necessary and practical. Advocates of the older practice may list reasons, *e.g.*, pretrained models often exist already (like those pretrained on ImageNet [10] and shared on HuggingFace[6]), so we do not need to consider the cost of pretraining. However, advocates of `SX` may argue that not all pretrained models are available; for many tasks, we still need to train the pretrained models first, so the cost of scratch training should be considered.

We have no inclination here regarding which one is more correct. We make two points that we are fairly certain about: (1) In the pruning literature, most papers still follow the *older* practice when reporting the scratch training results of the pruned model. (2) Given the recent rise of foundation models [4] (*e.g.*, Bert [12], GPT-3 [5], CLIP [61], diffusion models [66, 71]), common researchers barely have the resources to train a model from scratch, so pruning would inevitably be conducted on the pretrained model, probably, for those big models.

**What comparison setup is mostly used now?** Unfortunately, `S2` is the most prevailing comparison setup at present [3]. This setup ignores at least one important factor that, we now know [37], has a significant impact on the final performance: the finetuning LR schedule.

In the following sections, we start our empirical investigation of unveiling `M1` and `M2`. We study `M2` first and then `M1`, because the conclusion about `M2` actually depends on `M1`, as we are about to show.

## 4. Reexamining the Value of Pruning

The rethinking paper [50] presents many valuable thoughts regarding the value of the 3-step pruning pipeline against scratch training. However, there are a few potential concerns in their experiments that may shake the validity of their conclusion. *First*, they directly cite the results of a few pruning papers and compare the relative performance drop.

[6]https://huggingface.co/

Because of the stark differences between different DL platforms, such a comparison (*e.g.*, comparing methods that use different base models) may not be convincing enough for rigorous analysis. *Second*, when reproducing the $L_1$-norm pruning [42], they use fixed LR 0.001 and 20 epochs, following [42], for the finetuning stage, which is now known to be severely sub-optimal (see Tab. 4, a larger finetuning LR 0.01 can significantly boost performance).

It is thereby of interest whether the no-value-of-pruning argument would change if the comparison is conducted under a strictly controlled condition and a better finetuning LR is employed. This section attempts to answer this question. Three comparison setups (`SX-A`, `SX-B`, and `S4.2`) are considered since they are the *most strict* setups up to date.

**Pruning method**. We choose *$L_1$-norm pruning* [42] because it is the most representative pruning method and easy to control at a strict comparison setup. Specifically, $L_1$-norm pruning prunes the filters of a pretrained model with the smallest $L_1$-norms to a predefined pruning ratio. After pruning, the pruned model is finetuned for a few epochs to regain performance. Other pruning methods, such as regularization-based methods (*e.g.*, [49, 79, 82]), introduce many factors that are hard to control for rigorous analysis, so we do not adopt them here. We will discuss how the findings can generalize to those cases later.

**Networks and datasets**. The network used for analysis is ResNet34 [26], following [42]. For standard benchmarks (*e.g.*, Tab. 1), we use ResNet50 [26] because it is one of the most representative benchmark networks in filter pruning. The datasets are ImageNet100 and the full ImageNet [10]. ImageNet100 is a randomly drawn 100-class subset of ImageNet. We use it for *faster* analysis given our limited resource budget. The full ImageNet is used for benchmarks.

**Implementation details of pruning**. For analysis, pruning is conducted on the 1st CONV layer (the 2nd CONVs are not pruned, following $L_1$-norm pruning [42]) in all residual blocks of ResNet34. The first CONV and all FC layers are spared, also following the common practice [20, 79, 87]. Uniform layerwise pruning ratio is employed (which usually under-performs a tuned non-uniform layerwise pruning ratio scheme; but since this paper does not target the best performance but explanation, we adopt it for easy analysis). We conduct pruning at a wide sparsity spectrum (10% to 95%) in the hopes of comprehensive coverage.

**One table to show them all**. The results are presented in Tab. 4. Before we present the analyses, we introduce a notion, *pruning epoch*, which is defined as the epoch when the sparsifying action is physically enforced. *E.g.*, if a model is trained for 30 epochs and then the sparsifying action is enforced, the pruning epoch is 30. We observe:

**(1)** For the `S4.2` setup (rows marked by ●), we are not

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<br>FLOPs (G, speedup: $k\times$) | 10%<br>3.30 (1.11×) | 30%<br>2.59 (1.41×) | 50%<br>1.90 (1.93×) | 70%<br>1.19 (3.09×) | 90%<br>0.48 (7.68×) | 95%<br>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/$k$F90, 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/$k$F90, 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).

allowed to change the LR schedule. The only thing we can change is the pruning epoch. As seen, the best pruning epoch varies *w.r.t.* the sparsity level – at a small pruning ratio, different pruning epochs give a similar performance; while as the pruning ratio arises, the performance becomes more sensitive to the pruning epoch, *e.g.*, for pruning ratio 95%, `P90F30, 1e-4` severely underperforms `P30F90, 1e-2`. Notably, a clear trend is, when the pruning ratio is large (70% to 95%), it is better to have a smaller pruning epoch.

Under this setup, only at pruning ratios of 30%-70%, pruning surpasses scratch training by a statistically significant gap. Therefore, we can only say pruning has a marginal advantage over scratch training here.

**(2)** Then we look at the setup `SX-A` (rows marked by ●). Under this setup, we are allowed to adjust the finetuning LR as long as the total epochs are kept the same. We increase the initial finetuning LR. As seen, it significantly improves the accuracies, *e.g.*, (`P30F90, 1e-1`) improves the accuracy by nearly 2% at pruning ratio 95%, against (`P30F90, 1e-2`). This is the performance-boosting effect aforementioned [37]. We also apply the larger LR trick to another two settings `P60F60` and `P90F30`. In all of them, we see a larger finetuning LR improves performance by an obvious margin.

Now, the gap between pruning and scratch training becomes much more significant. Pruning is more surely valuable under this setup.

**(3)** Next, we use the comparison setup `SX-B` (rows marked by ●), which maintains the total FLOPs. We apply this scheme to the best pruning setup `P30F90, 1e-1` in `S4.1` in the hopes of better performances. Since the dense model is trained for 30 epochs, to compensate for the FLOPs, the pruning epoch should be squeezed by the speedup ratio $k$. *E.g.*, for pruning ratio 10%, the speedup ratio is 1.11, then the pruning epoch should be adjusted to $30/k \approx 27$.

As seen, the squeezing of the pruning epoch does close the gap between pruning and scratch training: At pruning ratios of 10% to 70%, pruning is still better; while for 90% and 95%, pruning only matches or underperforms scratch training – this is a concrete example that we do *not* have a once-for-all answer to questions like "*is pruning better than scratch training?*"

We also try a smaller finetuning LR in this setup, as shown in the row (`P30/kF90, 1e-2`). The LR effect also translates to this case – a smaller finetuning LR degrades the performance.

**Short summary**. As seen, the argument about the value of network pruning severely hinges on which comparison setup is employed and the pruning ratio. For the setup `SX-A`, where pruning outperforms scratch training obviously, the advantage comes from a better finetuning LR. Yet, we are not sure if such better LR schedules also exist for the scratch training; if so, scratch training can be further boosted, too – as such, this kind of "competition" can be *endless*. There are two kinds of attitudes toward this situation: *(1)* Do not consider the performance improvement from a better finetuning LR as a fair/valid performance advantage as it is *not* from the pruning algorithm. *(2)* Still consider it as a valid performance advantage but will meet the "endless competition" challenge we just mentioned. The community now is mostly using *(2)*. We suggest using *(1)*, following our fairness definition clarified in Sec. 3.

Despite many uncertainties, we are certain about one thing from Tab. 4: Whichever setup is favored, the finetuning LR holds a critical role in performance. Even for the comparisons setup `S4.2`, where the finetuning LR does not change, by changing the pruning epoch, implicitly, we change the finetuning LR, and it has been shown very pertinent to the final performance as well. In this sense, the two mysteries of pruning actually boil down to one (`M1`): *Why does finetuning LR have such a great impact on the performance?*

This is the next question we would like to answer. LR, (arguably) as the most influential hyper-parameter in training neural networks, has a significant impact on performance – this is definitely not surprising; what is really surprising might be, why the prior pruning works (*e.g.*, the original $L_1$-norm pruning [42] adopts LR 0.001 in finetuning for their ImageNet experiments) did not realize that such a simple "trick" is so important to performance? This question is also worth our thinking.

## 5. Trainability in Network Pruning

### 5.1. Background of Trainability

Trainability, by its name, means the ability (easiness) of training a neural network, *i.e.*, the optimization speed (note, speed is not equal to quality, so we may see a network with good trainability turns out to have a bad generalization ability eventually).

Notably, essentially, the role of a pruning method is to provide the initial weights for the later finetuning process, that is, pruning is essentially a kind of *initialization*. In stark contrast to the broad awareness that initialization is very critical to neural network training [21,25,34,54,72], the initialization role of pruning has received negligible research attention, however. Trainability is also mostly studied for random initialization [67, 83].

A few recent works marry it with network pruning in some other similar forms like signal propagation [40] and gradient flows [77] (a good signal propagation or strong gradient flow usually suggests a good trainability). These works are inspiring, while they mostly stay in the domain of pruning at initialization (PaI). Few attempts before, to our best knowledge, tried to utilize the notion of trainability to examine pruning after training (PaT), at least, for the two mysteries we study here. This paper is meant to bridge this gap. The major difference between PaI and PaT is whether using a pretrained model as base. Such a context is essential to this paper since the above two mysteries are both brought forward in the context of PaT.

**Trainability accuracy**. Literally, a bad trainability implies the training is hard and the training performance will arise slowly. Per this idea, there is a straightforward metric to measure trainability – we introduce *trainability accuracy*, the average of the first $N$ epochs,

$$T = \frac{1}{N}\sum_{i=1}^{N} \mathrm{Acc}_i. \tag{1}$$

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 | $79.57_{\pm0.15}$ | 88.00 |
| P30F90, 1e-2 | $77.64_{\pm0.39}$ | 77.45 |
| P30F90, 1e-2 (+30 epochs) | $79.12_{\pm0.19}$ | / |
| P30F90, 1e-2 (+60 epochs) | $\mathbf{79.59}_{\pm0.25}$ | / |
| P60F60, 1e-2 | $\mathbf{77.81}_{\pm0.25}$ | 87.39 |
| P60F60, 1e-3 | $70.53_{\pm0.37}$ | 68.19 |
| P60F60, 1e-3 (+60 epochs) | $75.71_{\pm0.09}$ | / |
| P60F60, 1e-3 (+120 epochs) | $77.17_{\pm0.13}$ | / |
| P60F60, 1e-3 (+180 epochs) | $77.33_{\pm0.09}$ | / |
| P90F30, 1e-2 | $75.20_{\pm0.23}$ | 84.83 |
| P90F30, 1e-4 | $29.89_{\pm0.26}$ | 37.93 |
| P90F30, 1e-4 (+60 epochs) | $60.69_{\pm0.17}$ | / |
| P90F30, 1e-4 (+270 epochs) | $70.78_{\pm0.16}$ | / |
| P90F30, 1e-4 (+1485 epochs) | **78.18** | / |

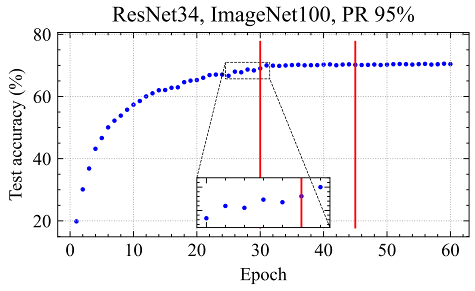
Since the optimization speed depends on the LR used, when we calculate trainability accuracy, we must ensure they are under the same LR schedule. In this paper, we choose $N$ as the number of the 1st LR stage, which characterizes the optimization speed in the early phase.

Next, we utilize trainability to explain the mysterious effect of the finetuning LR.

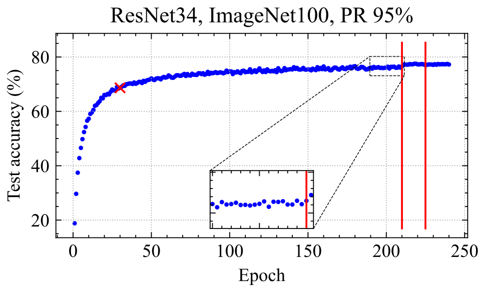
### 5.2. Examining the Effect of Finetuning LR

**Two facts as foundation**. We first lay down two facts as the common ground for the discussion of this section. We will show the mystery about the finetuning LR effect boils down to these two simple facts.

*First, pruning damages trainability*. This is an intuitively straightforward fact since pruning removes connections or neurons, which virtually makes the network harder to train. This fact holds for not only pruning a random network [40], but also for pruning a pretrained model here. Moreover, notably, more aggressive pruning leads to more damaged trainability. *Second, a model of worse trainability will need more effective updates to reach convergence*. More effective updates mean two cases: If LR is not changed, more epochs are needed; if the number of epochs does not change, a larger LR is needed. This is also easy to understand since trainability measures the easiness of optimization; a bad trainability implies harder optimization



(a) P60F60, 1e-3



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

Figure 2. Test accuracy *vs*. epoch during finetuning of the setting `P60F60, 1e-3` at pruning ratio 95% in Tab. 5. Red vertical lines mark the epoch of decaying LR by 0.1. Particularly note before the 1st LR decay, the accuracy keeps arising in (a), implying the 1st LR decay may be too early – this is confirmed in (b), where the red cross marker (×) indicates the time point of the 1st LR decay in (a). See more similar plots in our supplementary material.

literally, hence the more effective updates. Such observation has been made by some sparse training papers, *e.g*., RigL [16] notes that "*sparse training methods benefit significantly from increased training steps*".

When we observe that a larger LR improves the final test accuracy of the pruned model (*e.g*., Row `P30F90, 1e-1` *vs*. Row `P30F90, 1e-2` in Tab. 4), it is worthwhile to differentiate two subtle yet distinct possibilities:

- A larger LR helps the pruned model reach a solution that the smaller LR *cannot* reach, *i.e*., a larger LR help the model located into a better local minimum basin.

- The smaller LR can also help the model reach the solution as the larger LR does; just, the larger LR helps the model get there faster.

The former implies the performance-boosting effect of a larger LR is *fundamental*; while the latter implies there is no fundamental gap between the two solutions; it is only an issue of optimization speed.

Let's analyze a concrete example of `P60F60` in Tab. 5. For pruning ratio 95% (we use this for example because at larger sparsity, the performance boosting effect is most pronounced), using 1e-2 *vs*. 1e-3 improves the test accuracy from 70.53 to 77.81, a very significant jump. This improvement also translates to the rows of `P30F90` and `P90F30`.

However, in Tab. 5, we note the performance improvement coincides with trainability accuracy improvement. We were wondering if the performance improvement is actually due to a better trainability.

Fig. 2(a) plots the test accuracy during the finetuning of `P60F60, 1e-3`. We notice before the 1st LR decay at epoch 30, the accuracy keeps arising even at epoch 30. This triggers a question: usually, we decay LR when the accuracy *saturates*; now, when the accuracy is still steadily rising, *is the LR decayed too early?* This question matters because if the LR decays too early, the model is *forced* to stabilize due to the small step size and insufficient updates, not because it gets close to the local minimum, *i.e*., the model may *not converge at all*.

To verify this, we extend the epochs of the LR 0.001 phase by 60/120/180 epochs. See the results in Tab. 5 (note the rows `P60F60, 1e-3 (+60/120/180 epochs)`) and Fig. 2(b). Now, the model finetuned by LR 1e-3 can reach 77.33, very close to 77.81 reached by LR 1e-2. The test accuracy plot in Fig. 2(b) also confirms that the seeming underperformance of LR 1e-3 is due to insufficient epochs – namely, **the advantage of a larger LR 0.01 is not some magic fundamental advantage, but a simple consequence of faster optimization.**

We also verify this on other cases (`P30F90` and `P90F30`) that the smaller LR "underperforms" the larger LR. The results are also presented in Tab. 5. In *all* of these cases, given abundant epochs, the gap between the larger LR and the smaller LR can be closed. Especially for `P90F30`, the smaller LR 1e-4 can achieve a much better result than LR 1e-2 (78.18 *vs*. 75.20). This strongly demonstrates the smaller LR can also achieve what the larger LR can do.

To summarize, our results suggest **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*.** In practice, when researchers tune different LR's, they usually keep the total epochs fixed (for the sake of fairness). Given the same total epochs, the pruned model using the smaller finetuning LR does not fully converge, making the performance *appear* "worse".

**Further remarks.** It is worthwhile to note that such an experimenting trap is *so covert* if we are unaware of the damaged trainability issue in pruning. We may never realize that the epochs should be increased properly if a smaller finetuning LR is used. What's even trickier, we do not know how many more epochs is the so-called *proper* – Tab. 5 is a living example. For some cases (*e.g*., `P30F90`), 60 more epochs is enough, while for others (*e.g*., `P60F60`, `P90F30`), 180 epochs or more is not enough to bridge the performance gap. Clearly, there is still much work to be done here toward a more rigorous understanding of the influence of damaged trainability on pruning.

**Retrospective remarks and the lessons.** It is worthwhile to ponder why [42] employed a *seriously sub-optimal* finetuning LR scheme. This, we conceive, may originate from a long-standing *misunderstanding* in the area of network pruning – many have believed that because pruning is conducted onto a *converged* model, the retraining of the pruned model needs not to be long and the LR should be *small* to avoid destroying the knowledge the model has acquired, *e.g.*, in [65], the authors mentioned in their abstract "*The standard retraining technique, fine-tuning, trains the unpruned weights from their final trained values using a small fixed learning rate*", implying that such misconception spreads so widely that it is taken for "standard".[7]

However, the results in Tab. 4 suggest, such thought only holds for the cases of low pruning ratios. For a moderate or large pruning ratio, this thought *hardly* holds. What was neglected is that the sparsifying action damages network trainability, slowing down the optimization. As a compensation, it is supposed to use a larger LR to accelerate the optimization, not a smaller LR; similarly, more epochs are needed to compensate for the slow optimization. However, the original $L_1$-norm pruning [42] chose LR 0.001 and only 20 epochs for their ImageNet experiments, exactly the opposite of what is expected. This, we conceive, is the reason that $L_1$-norm pruning has been underrated for a long time. Its real performance is actually pretty strong even compared with recent top-performing approaches (see Tab. 1).

Similarly, based on what we just learned about the truth of `M1`, if we examine the other filter pruning methods, *e.g.*, GAL [47] (see Tab. 3), its reported results are probably also underrated, because it uses only 30 epochs for finetuning and the model may well not expose its full potential, as a result of the immature convergence. This implies a pretty disturbing concern – for many filter pruning papers, we have to calibrate their results for fairness. Directly citing the numbers may (well) not show the real performance comparison.

**How to make comparisons in a pruning paper?** As seen, different finetuning hyper-parameters can lead to very different conclusions, which do not reflect the true comparison of the pruning algorithm part. The ideal solution to this problem is to re-implement the existing pruning algorithms *under the same configuration*, which however is barely feasible in practice. We suggest a more practical solution here, highlighting two rules of thumbs: *(1)* cross-validating different hyper-parameters; *(2)* better performance still weighs more when fairness is hard to figure out.

For a concrete example, given two pruning algorithms `A` (with finetuning setup $FT_A$) and `B` (with finetuning setup $FT_B$), we can conduct two more experiments: `A` + $FT_B$ and `B` + $FT_A$; then compare `A` + $FT_A$ *vs.* `B` + $FT_A$ and `A` + $FT_B$ *vs.* `B` + $FT_B$.

- If one method consistently wins in both cases, we are more confident that the winner is really better due to its effective pruning design.

- If there is inconsistency between the two comparisons, *e.g.*, `A` + $FT_A >$ `B` + $FT_A$, while `A` + $FT_B <$ `B` + $FT_B$, namely, different pruning methods favors different finetuning recipes. This means synergistic interaction exists between the pruning and finetuning stage and we cannot separate them when evaluating the pruning algorithms. In this case, we suggest that **the case delivering higher performance weighs more** as it advances the state-of-the-art[8].

## 6. Conclusion and Discussion

This paper attempts to figure out the confounding benchmark situation in filter pruning. Two particular mysteries are explored: the performance-boosting effect of a larger finetuning LR, the no-value-of-pruning argument. We present a clear fairness principle and sort out four groups of popular comparison setups used by many pruning papers. Under a strictly controlled condition, we examine the two mysteries and find they both boil down to the issue of damaged network trainability. This issue was not well recognized by prior works, leading to (severely) sub-optimal hyper-parameters, which ultimately exacerbates the confounding benchmark situation in filter pruning now. We hope this paper helps the community towards a clearer understanding of pruning and more reliable benchmarks of it.

**Takeaways and suggestions from this paper**:

- *Why is the state of neural network pruning so confusing?* Non-standard comparison setups (and its fundamental reason: unclear fairness principle) and the unawareness of the role of trainability are the two major reasons. The latter further leads to sub-optimal hyper-parameter settings, inherited by many follow-up papers, exacerbating the messy benchmark situation.

- As the area of network pruning develops, various comparison setups have appeared (see Tab. 2). Each has its own historical context. Unfortunately, the most prevailing comparison setup now, setup `S2` in Tab. 2, **cannot** ensure fairness. **We suggest using the setup `S3.2` or higher**, *i.e.*, maintaining the same base

[7]Actually, the 3rd-step of the pruning pipeline is broadly referred to as *finetuning* – this term per se already implies the inclination of using a small LR. To rule out such conceptual bias, a more accurate way to phrase the 3rd step in the pruning pipeline may be *retraining* the pruned model.

[8]Exact fairness sometimes may be of little meaning for scientific progress. Think about the surge of deep learning – compared to the past SVM era, deep learning nowadays definitely takes (way) more resources than SVM, *i.e.*, unfair, while few has questioned the unprecedented value of deep learning as LLMs like ChatGPT surprise us on a daily basis.

model *and the same finetuning process* – Higher comparison setup means stricter experiment control, also means more resources and efforts; so there would be inevitably a *trade-off* between how fair we want to be and how much we can invest in[9].

- Reporting all the finetuning details (*esp.* the LR schedule) is rather necessary and should be standardized.
- Cross-validating finetuning setups is a practical suggestion to decide the true winner between two pruning algorithms with different finetuning recipes.
- Filter pruning can beat scratch training or not, up to the specific comparison setup and pruning ratio in consideration. Given the recent rise of large foundation models, pruning may still follow the conventional 3-step pipeline.
- The observation that a larger finetuning LR "improves" pruning performance is largely a *misinterpretation* – the performance is not "improved"; what really happens is that the good performance is observed earlier because the larger LR accelerates the optimization. The fundamental factor playing under the hood is the network trainability damaged by the sparsifying action (or zeroing out) action in pruning.
- The damaged network trainability was not well recognized by prior pruning works, resulting in severely sub-optimal hyper-parameters, rendering the potential of a baseline method, $L_1$-norm pruning [42], *underestimated* for a long time. This fact may spur us to *re-evaluate* the actual efficacy of (so) many sophisticated pruning methods against the simple $L_1$-norm pruning.
- The term *finetuning* (*i.e.*, the 3rd step in the pruning pipeline) is not suitable. It has the incorrect conceptual bias of using a small LR – we should firmly use *retraining* instead.

## References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: A system for large-scale machine learning. In *IEEE Symposium on Operating Systems Design and Implementation*, 2016. 6

[2] Eric Baum and David Haussler. What size net gives valid generalization? In *NeurIPS*, 1988. 1

[3] Davis Blalock, Jose Javier Gonzalez, Jonathan Frankle, and John V Guttag. What is the state of neural network pruning? In *MLSys*, 2020. 1, 2, 3, 6, 7

[9] If we look back at the historical contexts of different comparison setups (Sec. 3.2), sometimes, the unfairness is not because we *do not want*, but because we *cannot*.

[4] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021. 7

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *NeurIPS*, 2020. 1, 7

[6] Yves Chauvin. A back-propagation algorithm with optimal use of hidden units. In *NeurIPS*, 1988. 1

[7] Jian Cheng, Pei-song Wang, Gang Li, Qing-hao Hu, and Han-qing Lu. Recent advances in efficient computation of deep convolutional neural networks. *Frontiers of Information Technology & Electronic Engineering*, 19(1):64–77, 2018. 1, 3

[8] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine*, 35(1):126–136, 2018. 1, 3

[9] Elliot J Crowley, Jack Turner, Amos Storkey, and Michael O'Boyle. A closer look at structured pruning for neural network compression. *arXiv preprint arXiv:1810.04622*, 2018. 2

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 1, 6, 7, 14

[11] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532, 2020. 1, 3

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 1, 7

[13] Xiaohan Ding, Guiguang Ding, Jungong Han, and Sheng Tang. Auto-balanced filter pruning for efficient convolutional neural networks. In *AAAI*, 2018. 3

[14] Xiaohan Ding, Tianxiang Hao, Jianchao Tan, Ji Liu, Jungong Han, Yuchen Guo, and Guiguang Ding. Resrep: Lossless cnn pruning via decoupling remembering and forgetting. In *ICCV*, 2021. 6

[15] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *JMLR*, 20(55):1–21, 2019. 5

[16] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *ICML*, 2020. 10

[17] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019. 3

[18] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *ICML*, 2020. 3

[19] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Pruning neural networks at initialization: Why are we missing the mark? In *ICLR*, 2021. 3

[20] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019. 1, 3, 7

[21] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010. 9

[22] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, 2016. 3, 5

[23] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural network. In *NeurIPS*, 2015. 3, 4, 5

[24] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *NeurIPS*, 1993. 3

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *CVPR*, 2015. 4, 9

[26] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 6, 7, 15, 16

[27] Yang He, Yuhang Ding, Ping Liu, Linchao Zhu, Hanwang Zhang, and Yi Yang. Learning filter pruning criteria for deep convolutional neural networks acceleration. In *CVPR*, 2020. 1

[28] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *IJCAI*, 2018. 1

[29] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017. 3, 5, 6

[30] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *JMLR*, 22(241):1–124, 2021. 1, 3, 5

[31] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, 2018. 1

[32] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, 2017. 1

[33] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM MM*, 2014. 5

[34] Philipp Krähenbühl, Carl Doersch, Jeff Donahue, and Trevor Darrell. Data-dependent initializations of convolutional neural networks. In *ICLR*, 2016. 9

[35] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 6

[36] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012. 1

[37] Duong H Le and Binh-Son Hua. Network pruning that matters: A case study on retraining variants. In *ICLR*, 2021. 2, 3, 4, 6, 7, 8

[38] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015. 1

[39] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *NeurIPS*, 1990. 1, 3

[40] Namhoon Lee, Thalaiyasingam Ajanthan, Stephen Gould, and Philip HS Torr. A signal propagation perspective for pruning neural networks at initialization. In *ICLR*, 2020. 3, 9

[41] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. Snip: Single-shot network pruning based on connection sensitivity. In *ICLR*, 2019. 3

[42] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *ICLR*, 2017. 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 15, 16

[43] Tuanhui Li, Baoyuan Wu, Yujiu Yang, Yanbo Fan, Yong Zhang, and Wei Liu. Compressing convolutional neural networks via factorized convolutional filters. In *CVPR*, 2019. 1, 6

[44] Yuchao Li, Shaohui Lin, Jianzhuang Liu, Qixiang Ye, Mengdi Wang, Fei Chao, Fan Yang, Jincheng Ma, Qi Tian, and Rongrong Ji. Towards compact cnns via collaborative compression. In *CVPR*, 2021. 1

[45] Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. In *ICLR*, 2020. 1

[46] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *CVPR*, 2020. 1, 6

[47] Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. Towards optimal structured cnn pruning via generative adversarial learning. In *CVPR*, 2019. 1, 6, 11

[48] Yufan Liu, Jiajiong Cao, Bing Li, Chunfeng Yuan, Weiming Hu, Yangxi Li, and Yunqiang Duan. Knowledge distillation via instance relationship graph. In *CVPR*, 2019. 4

[49] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017. 7

[50] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *ICLR*, 2019. 2, 3, 4, 5, 6, 7, 15

[51] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *ICLR*, 2017. 16

[52] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, 2017. 6

[53] Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J Dally. Exploring the granularity of sparsity in convolutional neural networks. In *CVPR Workshop*, 2017. 3

[54] Dmytro Mishkin and Jiri Matas. All you need is a good init. In *ICLR*, 2016. 9

[55] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *CVPR*, 2019. 1, 6

[56] P. Molchanov, S. Tyree, and T. Karras. Pruning convolutional neural networks for resource efficient inference. In *ICLR*, 2017. 5

[57] Michael C Mozer and Paul Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *NeurIPS*, 1988. 1

[58] Junghun Oh, Heewon Kim, Seungjun Nah, Cheeun Hong, Jonghyun Choi, and Kyoung Mu Lee. Attentive fine-grained structured sparsity for image restoration. In *CVPR*, 2022. 3

[59] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019. 4, 6, 14

[60] Hanyu Peng, Jiaxiang Wu, Shifeng Chen, and Junzhou Huang. Collaborative channel pruning for deep networks. In *ICML*, 2019. 1, 6

[61] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *ICML*, 2021. 1, 7

[62] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. 1

[63] Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What's hidden in a randomly weighted neural network? In *CVPR*, 2020. 3

[64] R. Reed. Pruning algorithms – a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, 1993. 1, 3, 15

[65] Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *ICLR*, 2020. 3, 11

[66] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022. 1, 7

[67] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *ICLR*, 2014. 9

[68] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015. 1

[69] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 1

[70] Sidak Pal Singh and Dan Alistarh. Woodfisher: Efficient second-order approximations for model compression. In *NeurIPS*, 2020. 3

[71] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*, 2015. 7

[72] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, 2013. 9

[73] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017. 1, 3

[74] Lucas Theis, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszár. Faster gaze prediction with dense networks and fisher pruning. *arXiv preprint arXiv:1801.05787*, 2018. 3

[75] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 1

[76] Chaoqi Wang, Roger Grosse, Sanja Fidler, and Guodong Zhang. Eigendamage: Structured pruning in the kronecker-factored eigenbasis. In *ICML*, 2019. 3, 5

[77] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *ICLR*, 2020. 3, 9

[78] Huan Wang and Yun Fu. Trainability preserving neural structured pruning. *arXiv preprint arXiv:2207.12534*, 2022. 6

[79] Huan Wang, Can Qin, Yulun Zhang, and Yun Fu. Neural pruning via growing regularization. In *ICLR*, 2021. 1, 3, 4, 5, 6, 7, 15

[80] Huan Wang, Can Qin, Yulun Zhang, and Yun Fu. Recent advances on neural network pruning at initialization. In *IJCAI*, 2022. 3, 5

[81] Huan Wang, Qiming Zhang, Yuehai Wang, Lu Yu, and Haoji Hu. Structured pruning for efficient convnets via incremental regularization. In *IJCNN*, 2019. 3

[82] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *NeurIPS*, 2016. 3, 5, 6, 7, 15

[83] Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks. In *ICML*, 2018. 9

[84] Yulun Zhang, Huan Wang, Can Qin, and Yun Fu. Aligned structured sparsity learning for efficient image super-resolution. In *NeurIPS*, 2021. 6

[85] Yulun Zhang, Huan Wang, Can Qin, and Yun Fu. Learning efficient image super-resolution networks via structure-regularized pruning. In *ICLR*, 2022. 6

[86] Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. Learning n: M fine-grained structured sparse neural networks from scratch. In *ICLR*, 2021. 3

[87] Michael H Zhu and Suyog Gupta. To prune, or not to prune: Exploring the efficacy of pruning for model compression. In *ICLR Workshop*, 2018. 7

[88] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. In *NeurIPS*, 2018. 1, 6

[89] Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. 5

## A. Training setting summary

In the paper, we evaluate on two datasets ImageNet [10] and ImageNet100. The latter is a small version (100 classes) of ImageNet for faster experimenting. We use PyTorch [59] to implement all of our experiments. Therefore, we mainly refer to the official PyTorch ImageNet example[10]

[10]https://github.com/pytorch/examples/tree/main/imagenet

for hyper-parameters. The default LR schedule in PyTorch ImageNet example is: `0:0.1,30:0.01,60:0.001, total epochs:90`. This is adopted by $L_1$-norm pruning [42] (and inherited by [50]), so in their finetuning, they use the last-stage LR (*i.e.*, 0.001) as finetuning LR and never decay it.

**How we decide the LR schedule for scratch training?** Per our empirical observations, decaying the LR from 1e-3 to 1e-4 can still improve the model by around 0.5-0.8% top-1 accuracy. Namely, 1e-3 is *not* where the model finally converges. Since we want to compare models *at their best potential*, we add another 30 epochs and decay another two times to *make sure the model fully utilizes its potential* – this gives us the LR schedule for training a scratch model in Tab. 4: `0:0.1,30:0.01,60:0.001,90:1e-4,105:1e-5, total epochs:120`.

**How we decide the LR schedule for finetuning?** We abide by the following rules to decide the finetuning LR schedule:

- Rule 1: When comparing scratch training to pruning methods, **the final LR should be the same** – as we mentioned, decaying LR from 1e-3 to 1e-4 on ImageNet can see another 0.5-0.8% top-1 accuracy bump, thus it would be unfair to compare a pruning method whose model is finetuned to LR smaller than 1e-3 to a scratch-training model whose smallest LR is only 1e-3. The final LR for our ImageNet/ImageNet100 results, as mentioned above, is 1e-5.
- Rule 2: **Halving**. This means a kind of epoch splitting scheme: Given $N$ total epochs, split half of them (*i.e.*, $N/2$ epochs) to the 1st LR, then split the *half of the left epochs* (*i.e.*, $N/4$) to the 2nd LR, *etc*. To our best knowledge, this scheme is due to (no later than) the paper of ResNet [26] (see their CIFAR10 experiments).
- Rule 3: The epochs for each LR stage is **no more than 30**. This is mainly due to the design in the official PyTorch ImageNet example. We do not know why they chose 30 historically, but since this example is extensively followed as a baseline, we follow it too.

Based on these rules, given the total number of finetuning epochs, we can exactly derive the finetuning LR used in the paper. For example, in Tab. 4, for ● *P30F90, 1e-1*, the finetuning LR schedule is: `0:1e-1,30:1e-2,60:1e-3,75:1e-4,83:1e-5`; for ● *P60F60, 1e-2*, the finetuning LR schedule is: `0:1e-2,30:1e-3,45:1e-4,53:1e-5`.

**Layerwise pruning ratio for the experiments of ResNet50 on ImageNet**. For the results of ResNet50 on ImageNet we report in Tab. 1, its finetuning LR schedule is: `0:1e-2,30:1e-3,60:1e-4,75:1e-5, total epochs:90`. As seen, this is never some heavily tuned magic LR schedule; nevertheless, we use it to finetune the pruned model by $L_1$-norm pruning [42], only to find the final performance actually can *beat/match many top-performing methods after ICLR'17*. The layerwise pruning ratios are borrowed from GReg [79] (as they released their ratios; for many other papers, we only know the total speedup, not aware of the detailed layerwise pruning ratios) to keep a fair comparison with it – speedup 2.31×: [0, 0.60, 0.60, 0.60, 0.21, 0]; speedup 2.56×: [0, 0.74, 0.74, 0.60, 0.21, 0].

**Code references**. We mainly refer to the following code implementations in this work. They are all publicly available.

- Official PyTorch ImageNet example[11];
- GReg-1/GReg-2 [79][12];
- Rethinking the value of network pruning [50][13].

## B. Can the findings generalize to other pruning methods than $L_1$-norm pruning?

Pruning methods, according to their methodology, typically are classified into two groups, regularization-based (a.k.a. penalty-based) and importance-based (a.k.a. saliency-based), from a long time ago [64].

Despite the different categorization, *any* pruning method has a step to physically zero out the weights, *i.e.*, the *sparsifying action* step, per the terminology in this paper. Typically, this step is the magnitude pruning (or $L_1$-norm pruning when it comes to filter pruning). *E.g.*, SSL [82] and GReg [79] are two regularization-based pruning methods, with different penalty terms proposed, yet both of them have a step to physically zero out unimportant weights by sorting their magnitude before finetuning. In other words, regularization-based methods, although they are classified into a different group from magnitude pruning (which is importance-based), they essentially include magnitude pruning as a part.

We have analyzed in the paper, the fundamental reason that incurs damaged trainability is the sparsifying action action in magnitude pruning. Therefore, *any pruning method that employs magnitude pruning as a part is subject to the analyses of this paper* – this means the discoveries of this paper are *generic*. The attended broken trainability in these methods should also lead to *similar*[14] finetuning LR effect

[11] https://github.com/pytorch/examples/tree/master/imagenet
[12] https://github.com/MingSun-Tse/Regularization-Pruning
[13] https://github.com/Eric-mingjie/rethinking-network-pruning/tree/master/imagenet/l1-norm-pruning
[14] This said, the severity of the trainability issue may vary up to specific pruning methods. *E.g.*, we have observed the GReg method [79] is less bothered by such damaged trainability due to their growing regularization design.

to the $L_1$-norm pruning case.

## C. Can the finetuning LR effect generalize to other LR schedules than the traditional step decay?

In the paper, we explore the finetuning LR effect (a large LR *vs*. a small LR,*e.g*., 0.01 *vs*. 0.001) to the final performance using the conventional *step decay* LR schedule. It is of interest if the effect can translate to other more advanced LR schedules.

We consider *Cosine Annealing LR schedule* [51] here, referring to the official PyTorch Cosine LR implementation[15]. When we switch from Step LR schedule to Cosine, the initial LR and minimum LR are kept the same (namely, the start point and end point of LR are the same; the only difference is the scheduling in between). The scratch model is trained for 200 epochs, initial LR 0.1, step decayed at epoch 100 and 150 by multiplier 0.1 (referring to the original ResNet CIFAR10 training recipe in the ResNet paper [26]). For finetuning, the initial LR is 0.01 or 0.001, the minimum LR 0.0001, total epochs 120.

Table 6. Effect of LR schedule of ResNet56 on CIFAR10. Baseline accuracy 93.78%, Params: 0.85M, FLOPs: 0.25G.

| Pruning ratio | 0.3 | 0.5 | 0.7 | 0.9 |
|---|---|---|---|---|
| Sparsity/Speedup | 31.14%/1.45× | 49.82%/1.99× | 70.57%/3.59× | 90.39%/11.41× |
| Scratch (Step LR) | 93.16$_{\pm 0.16}$ | 92.78$_{\pm 0.23}$ | 92.11$_{\pm 0.12}$ | **88.36**$_{\pm 0.20}$ |
| Scratch (Cosine LR) | **93.84**$_{\pm 0.06}$ | **93.20**$_{\pm 0.31}$ | **92.15**$_{\pm 0.21}$ | 88.17$_{\pm 0.43}$ |
| $L_1$ [42] (Step LR 0.001) | 93.43$_{\pm 0.06}$ | 93.12$_{\pm 0.10}$ | 91.77$_{\pm 0.11}$ | **87.57**$_{\pm 0.09}$ |
| $L_1$ [42] (Step LR 0.01) | **93.79**$_{\pm 0.06}$ | **93.51**$_{\pm 0.07}$ | **92.26**$_{\pm 0.17}$ | 86.75$_{\pm 0.31}$ |
| $L_1$ [42] (Cosine LR 0.001) | 93.48$_{\pm 0.04}$ | 93.11$_{\pm 0.09}$ | 91.65$_{\pm 0.11}$ | **87.17**$_{\pm 0.14}$ |
| $L_1$ [42] (Cosine LR 0.01) | **93.82**$_{\pm 0.07}$ | **93.74**$_{\pm 0.06}$ | **92.27**$_{\pm 0.00}$ | 86.90$_{\pm 0.20}$ |

Table 7. Effect of LR schedule of VGG19 on CIFAR100. Baseline accuracy: 74.02%, Params: 20.08M, FLOPs: 0.80G.

| Pruning ratio | 0.3 | 0.5 | 0.7 | 0.9 |
|---|---|---|---|---|
| Sparsity/Speedup | 19.24%/1.23× | 51.01%/1.97× | 74.87%/3.60× | 90.98%/8.84× |
| Scratch (Step LR) | 72.84$_{\pm 0.25}$ | **71.88**$_{\pm 0.14}$ | **70.79**$_{\pm 0.08}$ | **66.52**$_{\pm 0.37}$ |
| Scratch (Cosine LR) | **73.54**$_{\pm 0.22}$ | 71.87$_{\pm 0.09}$ | 70.10$_{\pm 0.24}$ | 65.92$_{\pm 0.10}$ |
| $L_1$ [42] (Step LR 0.001) | 73.67$_{\pm 0.05}$ | 72.04$_{\pm 0.12}$ | 70.21$_{\pm 0.02}$ | 64.72$_{\pm 0.17}$ |
| $L_1$ [42] (Step LR 0.01) | **74.01**$_{\pm 0.18}$ | **73.01**$_{\pm 0.22}$ | **71.49**$_{\pm 0.14}$ | **66.05**$_{\pm 0.04}$ |
| $L_1$ [42] (Cosine LR 0.001) | 73.69$_{\pm 0.08}$ | 72.10$_{\pm 0.08}$ | 69.96$_{\pm 0.09}$ | 63.93$_{\pm 0.15}$ |
| $L_1$ [42] (Cosine LR 0.01) | **74.39**$_{\pm 0.07}$ | **73.51**$_{\pm 0.18}$ | **71.78**$_{\pm 0.21}$ | **65.70**$_{\pm 0.11}$ |

The results of ResNet56 (on CIFAR10) and VGG19 (on CIFAR100) are presented in Tab. 6 and Tab. 7. As seen, the advantage of initial LR 0.01 over 0.001 does not only appear with the Step LR schedule, but also appears with the Cosine LR schedule in most cases (*esp*. for VGG19). This implies the finetuning LR effect is *generic*, not limited to one particular LR schedule, which further highlights the importance of the topic we have been studying in the paper.

## D. Additional results

**1e-4 vs. 1e-2 for *P90F30***. In Tab. 5 of the paper, we mention the performance gap between a small LR and a large

Table 8. Top-1 accuracy (%) comparison of different setups of $L_1$-norm pruning [42] with **ResNet34** on **ImageNet100**. Pruning ratio: 95%. This table shows, the performance gap between a smaller LR and a larger LR is not fundamental. It can be closed (or squeezed) 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. (%) | Trainability acc. (%) |
|---|---|---|
| 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** | / |

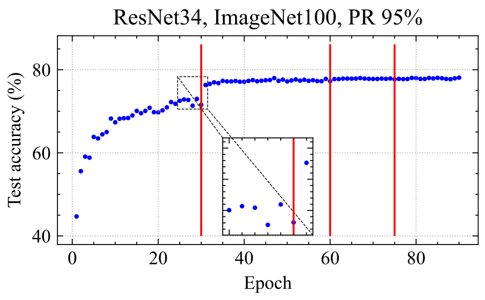
LR is not fundamental, but a simple consequence of convergence speed under different LRs. When we add more finetuning epochs, the performance gap can be closed fully or by a large part for *P30F90* and *P60F60*; while on *P90F30*, the gap is still obvious even after we add 270 epochs.

Here we add even more, 1485 epochs, so that the number of the 1st LR stage is now 1500 epochs, exactly 100 times of the 1st-LR-stage epochs (*i.e*., 15 epochs) when using 1e-2 as initial LR. As we see, now LR 1e-4 can achieve 78.18 top-1 accuracy, which is significantly better than 75.20 achieved by LR 1e-2. This is yet another strong piece of evidence to show that the seeming performance gap between a large LR and a small LR is *never* a gap that cannot be bridged, further confirming our opinion in the paper.

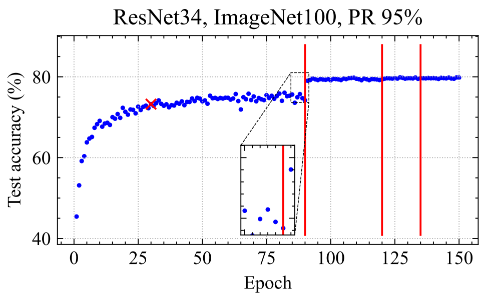
**More learning curves**. In Fig. 1 of the paper, we present the learning curves for *P60F60, 1e-3* without and with more finetuning epochs, to show the underperformance of a small LR is actually due to insufficient training. Here we present more plots for *P30F90, 1e-2* and *P90F30, 1e-4* – see Fig. 3.

In both cases (note the red crosses × in (1.b) and (2.b)), the performance can be boosted by adding more epochs to the 1st LR stage, especially for the case of *P90F30, 1e-4*, where the 1st LR decay is actually *way too early*. These plots further confirm our opinion in the paper – the seeming underperformance of a small finetuning LR is not something magic, but a simple consequence of *slow convergence* (caused by the broken trainability, *esp.* at large pruning ratios like 95%).

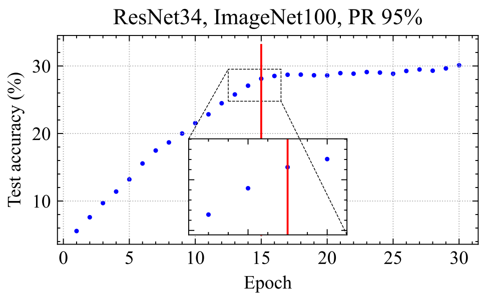
[15]https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.CosineAnnealingLR.html



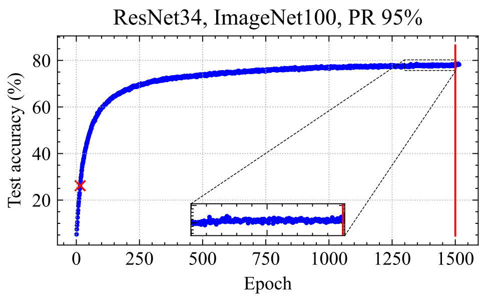
(1.a) P30F90, 1e-2



(1.b) P30F90, 1e-2 (+60 epochs)



(2.a) P90F30, 1e-4



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

Figure 3. Test accuracy *vs*. epoch during finetuning of the setting *P30F90, 1e-2* and *P90F30, 1e-4* at pruning ratio 95% in Tab. 5. Red vertical lines mark the epoch of decaying LR by 0.1. Particularly note before the *1st LR decay*, the accuracy keeps arising in (a), implying the 1st LR decay may be too early – this is confirmed in (b), where the red cross marker (×) indicates the time point of the 1st LR decay in (a).