# Rare Gems: Finding Lottery Tickets at Initialization

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

Large neural networks can be pruned to a small fraction of their original size, with little loss in accuracy, by following a time-consuming "train, prune, re-train" approach. Frankle & Carbin [8] conjecture that we can avoid this by training *lottery tickets*, *i.e.*, special sparse subnetworks found *at initialization*, that can be trained to high accuracy. However, a subsequent line of work [10, 34] presents concrete evidence that current algorithms for finding trainable networks at initialization, fail simple baseline comparisons, *e.g.*, against training random sparse subnetworks. Finding lottery tickets that train to better accuracy compared to simple baselines remains an open problem. In this work, we resolve this open problem by proposing GEM-MINER which finds lottery tickets *at initialization* that beat current baselines. GEM-MINER finds lottery tickets trainable to accuracy competitive or better than Iterative Magnitude Pruning (IMP), and does so up to $19\times$ faster.

## 1 Introduction

A large body of research since the 1980s empirically observed that large neural networks can be compressed or sparsified to a small fraction of their original size while maintaining their predictive accuracy [13–15, 18, 20, 25, 38]. Although several pruning methods have been proposed during the past few decades, many of them follow the "*train, prune, re-train*" paradigm. Although the above methods result in very sparse, accurate models, they typically require several rounds of re-training, which is computationally intensive.

Frankle & Carbin [8] suggest that this computational burden may be avoidable. They conjecture that given a randomly initialized network, one can find *a sparse subnetwork that can be trained to accuracy comparable to that of its fully trained dense counterpart*. This trainable subnetwork found *at initialization* is referred to as a *lottery ticket*. The study above introduced iterative magnitude pruning (IMP) as a means of finding these lottery tickets. Their experimental findings laid the groundwork for what is now known as the *Lottery Ticket Hypothesis* (LTH).

Although Frankle & Carbin [8] establish that the LTH is true for tasks like image classification on MNIST, they were not able to get satisfactory results for more complex datasets like CIFAR-10 and ImageNet when using deeper networks, such as VGG and ResNets [9]. In fact, subsequent work brought the effectiveness of IMP into question. Su et al. [34] showed that even randomly sampled sparse subnetworks at initialization can beat lottery tickets found by IMP as long as the layerwise sparsities are chosen carefully. Gale et al. [11] showed that methods like IMP which train tickets from initialization cannot compete with the accuracy of a model trained with pruning as part of the optimization process.

Frankle et al. [9] explain the failures of IMP using the concept of *linear mode connectivity* which measures the stability of these subnetworks to SGD noise. Extensive follow-up studies propose several heuristics for finding trainable sparse subnetworks at initialization [21, 35, 36]. However, subsequent work by Frankle et al. [10], Su et al. [34] show experimentally that all of these methods

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

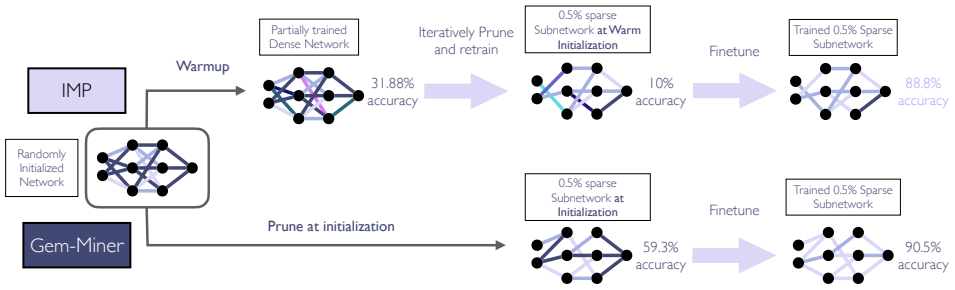


Figure 1: Conceptual visualization of GEM-MINER vs IMP with warmup. The accuracies listed are on a $0.5\%$ sparse VGG-16 trained on CIFAR-10. Given a randomly initialized network, both methods output a subnetwork which is then finetuned. IMP requires warmup *i.e.*, few epochs of training before it can find a sparse subnetwork. GEM-MINER finds a *rare gem*, a subnetwork *at initialization* that achieves high accuracy both before and after weight training.

fail simple sanity checks. Most methods seem to merely identify good sparsities at each layer, but given those, random sparse subnetworks can be trained to similar or better accuracy.

Frankle et al. [9] show that with a small modification, IMP can beat these sanity checks; the caveat is that it no longer finds these subnetworks at initialization, but after a few epochs of *warm-up* training. Since these subnetworks are found *after initialization*, **IMP with warmup does not find lottery tickets**.

As noted in the original work by Frankle & Carbin [8], the importance of finding trainable subnetworks at initialization is computational efficiency. It is far preferable to train a sparse model from scratch, rather than having to deal with a large dense model, even if that is for a few epochs (which is what IMP with warmpup does). To the best of our knowledge, the empirical validity of the *Lottery Ticket Hypothesis*, *i.e.*, the hunt for subnetworks at initialization trainable to SOTA accuracy, remains an open problem.

**Our Contributions.** We resolve this open problem by developing GEM-MINER, an algorithm that finds sparse subnetworks *at initialization*, trainable to accuracy comparable or better than IMP *with warm-up*. GEM-MINER does so by first discovering *rare gems*. Rare gems are subnetworks at initialization that attain accuracy far above random guessing, even before training. Rare gems can then be *refined* to achieve near state-of-the-art accuracy. Simply put, rare gems are lottery tickets that also have high accuracy at initialization.

High accuracy at initialization is not a requirement for a network to be defined as a lottery ticket. However, if our end goal is high accuracy after training, then having high accuracy at initialization likely helps.

Rare gems found by GEM-MINER are the first lottery tickets to beat all baselines in [10, 34]. In Fig. 1 we give a sketch of how our proposed algorithm GEM-MINER compares with IMP with warm start. GEM-MINER finds these subnetworks in exactly the same number of epochs that it takes to train them, and is up to $19\times$ faster than IMP *with warmup*.

## 2 Related Work

**Lottery ticket hypothesis.** Following the pioneering work of Frankle & Carbin [8], the search for lottery tickets has grown across several applications, such as language tasks, graph neural networks and federated learning [3, 4, 12, 22]. While the LTH itself has yet to be proven mathematically, the so-called strong LTH has been derived which shows that any target network can be approximated by pruning a randomly initialized network with minimal overparameterization [24, 26, 27]. Recently, it has been shown that for such approximation results it suffices to prune a random binary network with slightly larger overparameterization [6, 33].

**Pruning at initialization.** While network pruning has been studied since the 1980s, finding sparse subnetworks at initialization is a more recently explored approach. Lee et al. [21] propose SNIP, which prunes based on a heuristic that approximates the importance of a connection. Tanaka et al.

Table 1: We compare the different popular pruning methods in the literature on whether they prune at initialization, are finetunable and pass sanity checks. We also list the amount of computation they need to find a $1.4\%$ sparse subnetwork on ResNet-20, CIFAR-10. For consistency, we do not include the time required to finetune this subnetwork to full accuracy as it would be equal for all methods. For single-shot pruning method we list it as 1 epoch but this depends on the choice of batch-size. Learning Rate Rewinding which we label Renda et al. [29] is a pruning after training algorithm and just outputs a high accuracy subnetwork and hence the sanity checks do not apply to it.

| Pruning Method | Prunes at initialization | Finetunable | Passes sanity checks | Commputation to reach $1.4\%$ sparsity |
|---|---|---|---|---|
| IMP [8] | ✗ | ✓ | ✓ | 2850 epochs |
| SNIP [21] | ✓ | ✓ | ✗ | 1 epoch |
| GraSP [36] | ✓ | ✓ | ✗ | 1 epoch |
| SynFlow [35] | ✓ | ✓ | ✗ | 1 epoch |
| Edge-popup [28] | ✓ | ✗ | ✗ | 150 epochs |
| Smart Ratio [34] | ✓ | ✓ | – | $\mathcal{O}(1)$ |
| Learning Rate Rewinding [29] | ✗ | – | – | 3000 epochs |
| **GEM-MINER** | ✓ | ✓ | ✓ | **150 epochs** |

[35] propose SynFlow which prunes the network to a target sparsity without ever looking at the data. Wang et al. [36] propose GraSP which computes the importance of a weight based on the Hessian gradient product. The goal of these algorithms is to find a subnetwork that can be trained to high accuracy. Ramanujan et al. [28] propose Edge-Popup (EP) which finds a subnetwork at initialization that has high accuracy to begin with. Unfortunately, they also note that these subnetworks are not conducive to further finetuning.

The above algorithms are all based on the idea that one can assign a "score" to each weight to measure its importance. Once such a score is assigned, one simply keeps the top fraction of these scores based on the desired target sparsity. This may be done by sorting the scores layer-wise or globally across the network. Additionally, this can be done in *one-shot* (SNIP, GraSP) or *iteratively* (SynFlow). Note that IMP can also be fit into the above framework by defining the "score" to be the *magnitude* of the weights and then pruning globally across the network iteratively.

More recently, Alizadeh et al. [1] propose ProsPr which utilizes the idea of *meta-gradients* through the first few steps of optimization to determine which weights to prune. Their intuition is that this will lead to masks at initialization that are more amenable to training to high accuracy within a few steps. While it finds high accuracy subnetworks, we show in Section 4.2 that it fails to pass the sanity checks of Frankle et al. [10] and Su et al. [34].

**Sanity checks for lottery tickets.** A natural question that arises with pruning at initialization is whether these algorithms are truly finding interesting and nontrivial subnetworks, or if their performance after finetuning can be matched by simply training equally sparse, yet random subnetworks. Ma et al. [23] propose more rigorous definitions of winning tickets and study IMP under several settings with careful tuning of hyperparameters. Frankle et al. [10] and Su et al. [34] introduce several sanity checks (i) Random shuffling (ii) Weight reinitialization (iii) Score inversion and (iv) Random Tickets. Even at their best performance, they show that SNIP, GraSP and SynFlow merely find a good sparsity ratio in each layer and fail to surpass, in term of accuracy, fully trained randomly selected subnetworks, whose sparsity per layer is similarly tuned. Frankle et al. [10] show through extensive experiments that none of these methods show accuracy deterioration after random reshuffling. We explain the sanity checks in detail in Section 4 and use them as baselines to test our own algorithm.

**Pruning during/after training.** While the above algorithms prune at/near initialization, there exists a rich literature on algorithms which prune during/after training. Unlike IMP, algorithms in this category do not rewind the weights. They continue training and pruning iteratively. Frankle et al. [10] and Gale et al. [11] show that pruning at initialization cannot hope to compete with these algorithms. While they do not find lottery tickets, they do find high accuracy sparse networks. Zhu & Gupta [38] propose a gradual pruning schedule where the smallest fraction of weights are pruned at a predefined frequency. They show that this results in models up to 95% sparsity with negligible loss in performance on language as well as image processing tasks. Gale et al. [11] and Frankle et al. [10] also study this as a baseline under the name *magnitude pruning after training*. Renda et al. [29] show that rewinding the learning rate as opposed to weights(like in IMP) leads to the best performing sparse networks. However, it is important to remark that these are not Lottery Tickets, merely high accuracy sparse networks. We contrast these different methods in Table 1 in terms of

whether they prune at initialization, their finetunability, whether they pass sanity checks as well as their computational costs.

Finally, we note that identifying a good pruning mask can be thought of as training a binary network where the loss is computed over the element-wise product of the original network with the mask. This has been explored in the quantization during training literature [5, 17, 31].

## 3 GEM-MINER: Discovering Rare Gems

**Setting and notation.** Let $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ be a given training dataset for a $k$-classification problem, where $\boldsymbol{x}_i \in \mathbb{R}^{d_0}$ denotes a feature vector and label $y_i \in \{1, \dots, k\}$ denotes its label. Typically, we wish to train a neural network classifier $f(\boldsymbol{w}; \boldsymbol{x}) : \mathbb{R}^{d_0} \to \{1, \dots, k\}$, where $\boldsymbol{w} \in \mathbb{R}^d$ denotes the set of weight parameters of this neural network. The goal of a pruning algorithm is to extract a mask $\boldsymbol{m} = \{0, 1\}^d$, so that the pruned network is denoted by $f(\boldsymbol{w} \odot \boldsymbol{m}; \boldsymbol{x})$, where $\odot$ denotes the element-wise product. We define the *sparsity* of this network to be the fraction of non-zero weights $s = \|\boldsymbol{w} \odot \boldsymbol{m}\|_0 / d$. The loss of a classifier on a single sample $(\boldsymbol{x}, y)$ is denoted by $\ell(f(\boldsymbol{w} \odot \boldsymbol{m}; \boldsymbol{x}), y)$, which captures a measure of discrepancy between prediction and reality. In what follows, we will denote by $\boldsymbol{w}_0 \in \mathbb{R}^d$ to be the set of random initial weights. The type of randomness will be explicitly mentioned when necessary.

**On the path to rare gems; first stop: Maximize pre-training accuracy.** A rare gem needs to satisfy three conditions: (i) sparsity, (ii) non-trivial pre-training accuracy, and (iii) that it can be finetuned to achieve accuracy close to that of the fully trained dense network. This is not an easy task as we have two different objectives in terms of accuracy (pre-training and post-training), and it is unclear if a good subnetwork for one objective is also good for the other. However, since pre-training accuracy serves as a lower bound on the final performace, we focus on maximizing that first, and then attempt to further improve it by finetuning.

Our algorithm is inspired by Edge-Popup (EP) [28]. EP successfully finds subnetworks with high pre-training accuracy but it has two major limitations: (i) it does not work well in the high sparsity regime (*e.g.*, $< 5\%$), and (ii) most importantly, the subnetworks it finds are not conducive to further finetuning.

In the following, we take GEM-MINER apart and describe the components that allow it to surpass these issues.

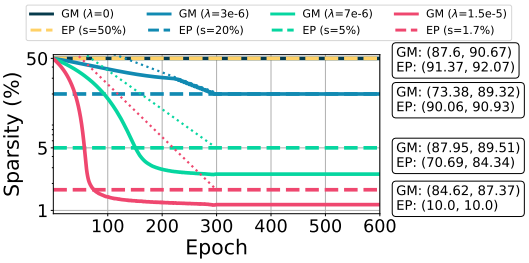


Figure 2: The sparsity of intermediate results, the accuracy of the final output, and the accuracy after finetuning on MobileNet-V2, CIFAR-10. For GEM-MINER, we also visualize the sparsity upper bounds as dotted lines. As $\lambda$ increases, note that the sparsity of GEM-MINER's output decreases. For$\lambda = 3 \cdot 10^{-6}$, the iterative freezing algorithm kicks in around epoch 220, regularizing the sparsity thereafter. The gem found by GEM-MINER($\lambda = 1.5 \cdot 10^{-5}$) achieves accuracy of $84.62\%$ before finetuning and $87.37\%$ after finetuning, while EP is unable to achieve non-trivial accuracy before or after finetuning.

**GEM-MINER without sparsity control.** Much like EP, GEM-MINER employs a form of backpropagation, and works as follows. Each of the random weights $[\mathbf{w}_0]_i$ in the original network is associated with a normalized score $p_i \in [0, 1]$. These normalized scores become our optimization variables and are responsible for computing the *supermask* $\mathbf{m}$, *i.e.*, the pruning pattern of the network at initialization.

For a given set of weights $\boldsymbol{w}$ and scores $\boldsymbol{p}$, GEM-MINER sets the effective weights as $\boldsymbol{w}_{\text{eff}} = \boldsymbol{w} \odot r(\boldsymbol{p})$, where $\mathrm{r}(\cdot)$ is an element-wise rounding function, and $\mathbf{m} = r(\boldsymbol{p})$ is the resulting supermask. The rounding function can be changed, *e.g.*, $r$ can perform randomized rounding, in which case $p_i$ would be the probability of keeping weight $w_i$ in $\mathbf{m}$. In our case, we found that simple deterministic rounding, *i.e.*, $r(p_i) = \mathbf{1}_{p_i \geq 0.5}$ works well.

At every iteration GEM-MINER samples a batch of training data and performs backpropagation on the loss of the effective weights, with respect to the scores $\boldsymbol{p}$, while projecting back to $[0, 1]$ when needed. During the forward pass, due to the rounding function, the effective network used is indeed a subnetwork of the given network. Here, since $r(\boldsymbol{p})$ is a non-differentiable operation we use the Straight Through Estimator (STE) [2] which backpropagates through the indicator function as though it were the identity function.

Note that this vanilla version of GEM-MINER is unable to exercise control over the final sparsity of the model. For reasons that will become evident in below, we will call this version of our algorithm GEM-MINER(0). There is already a stark difference from EP: GEM-MINER(0) will automatically find the optimal sparsity, while EP requires the target sparsity $s$ as an input parameter.

However, at the same time, this also significantly limits the applicability of GEM-MINER(0) as one cannot obtain a highly sparse gem. Shown as a dark blue curve in Fig. 2 is the sparsity of GEM-MINER(0). Here, we run GEM-MINER with a randomly initialized MobileNet-V2 network on CIFAR-10. Note that the sparsity stays around $50\%$ throughout the run, which is consistent with the observation by Ramanujan et al. [28] that accuracy of subnetworks at initialization is maximized at around $50\%$ sparsity.

**Algorithm 1:** GEM-MINER

**Input:** Dataset $D = \{(\boldsymbol{x}_i, y_i)\}$, learning rate $\eta$, rounding function $r(\cdot)$, number of epochs $E$, freezing period $T$, target sparsity $s \in [0, 1]$

**Output:** Mask $\boldsymbol{m} = r(\boldsymbol{p}) \odot \boldsymbol{q} \in \{0, 1\}^d$

$c \leftarrow \frac{\ln(1/s)}{E}$, $\boldsymbol{q} \leftarrow \mathbf{1}_d$

$\boldsymbol{w}, \boldsymbol{p} \leftarrow$ random vector in $\mathbb{R}^d$, random vector in $[0, 1]^d$

**for** $j$ *in* $1, 2, \ldots, E$ **do**

  **for** $(\boldsymbol{x}_i, y_i) \in D$ **do**

    $\boldsymbol{w}_{\text{eff}} \leftarrow (\boldsymbol{w} \odot \boldsymbol{q}) \odot r(\boldsymbol{p})$

    $\boldsymbol{p} \leftarrow \boldsymbol{p} - \eta \nabla_{\boldsymbol{p}} \ell(f(\boldsymbol{w}_{\text{eff}}; \boldsymbol{x}_i), y_i)$

    `/* STE */`

    $\boldsymbol{p} \leftarrow \text{proj}_{[0,1]^d} \boldsymbol{p}$

  **if** $\text{mod}(j, T) = 0$ **then**

    $I_1 \leftarrow \{i : q_i = 1\}$

    $\boldsymbol{p}_{sorted} \leftarrow \text{sort}(\boldsymbol{p}_{i \in I_1})$

    $\boldsymbol{p}_{bottom} \leftarrow$ Bottom-$(1 - e^{cT})$ fraction of $\boldsymbol{p}_{sorted}$

    $q \leftarrow q \odot \mathbb{1}_{p_i \notin \boldsymbol{p}_{bottom}}$

**Regularization and Iterative freezing.** GEM-MINER(0) is a good baseline algorithm for finding accurate subnetworks at initialization, but it cannot be used to find *rare gems*, which need to be sparse and trainable. To overcome this limitation, we apply a standard trick – we add a regularization term to encourage sparsity. Thus, in addition to the task loss computed with the effective weights, we also compute the $L_2$ or $L_1$ norm of the score vector $\boldsymbol{p}$ and optimize over the total regularized loss. More formally, we minimize $\ell := \ell_{\text{task}} + \lambda \ell_{\text{reg}}$, where $\lambda$ is the hyperparameter and $\ell_{\text{reg}}$ is either $L_2$ or $L_1$ norm of the score vector $\boldsymbol{p}$.

We call this variant GEM-MINER($\lambda$), where $\lambda$ denotes the regularization weight. This naming convention should explain why we called the initial version GEM-MINER(0).

The experimental results in Fig. 2 show that this simple modification indeed allows us to control the sparsity of the solution. We chose to use the $L_2$ regularizer, however preliminary experiments showed that $L_1$ performs almost identically. By varying $\lambda$ from $\lambda = 0$ to $\lambda = 7 \cdot 10^{-6}$ and $\lambda = 1.5 \cdot 10^{-5}$, the final sparsity of the gem found by GEM-MINER($\lambda$) becomes $2.5\%$ and $1.4\%$, respectively.

One drawback of this regularization approach is that it only indirectly controls the sparsity. If we have a target sparsity $s$, then there is no easy way of finding the appropriate value of $\lambda$ such that the resulting subnetwork is $s$-sparse. If we choose $\lambda$ to be too large, then it will give us a gem that is way too sparse; too small a $\lambda$ and we will end up with a denser gem than what is needed. As a simple heuristic, we employ *iterative freezing*, which is widely used in several existing pruning algorithms, including IMP [8, 11, 38]. More specifically, we can design an exponential function $\overline{s}(j) = e^{-cj}$ for some $c > 0$, which will serve as the upper bound on the sparsity. If the total number of epochs is $E$ and the target sparsity is $s$, we have $\overline{s}(E) = e^{-cE} = s$. Thus, we have $c = \ln(1/s)/E$.

Once this sparsity upper bound is designed, the iterative freezing mechanism regularly checks the current sparsity to see if the upper bound is violated or not. If the sparsity bound is violated, it finds the smallest scores, zeros them out, and freezes their values thereafter. By doing so, we can guarantee the final sparsity even when $\lambda$ was not sufficiently large. To see this freezing mechanism in action, refer the blue curve in Fig. 2. Here, the sparsity upper bounds (decreasing exponential functions) are visualized as dotted lines. Note that for the case of $\lambda = 3 \cdot 10^{-6}$, the sparsity of the network does not decay as fast as desired, so it touches the sparsity upper bound around epoch 220. The iterative freezing scheme kicks in here, and the sparsity decay is controlled by the upper bound thereafter, achieving the specified target sparsity at the end.

The full pseudocode of GEM-MINER is provided in Algorithm 1. There are two minor implementation details which differ from the explanation above: (i) we impose the iterative freezing every $T$ epochs, not every epoch and (ii) iterative freezing is imposed even when the sparsity bound is not violated.

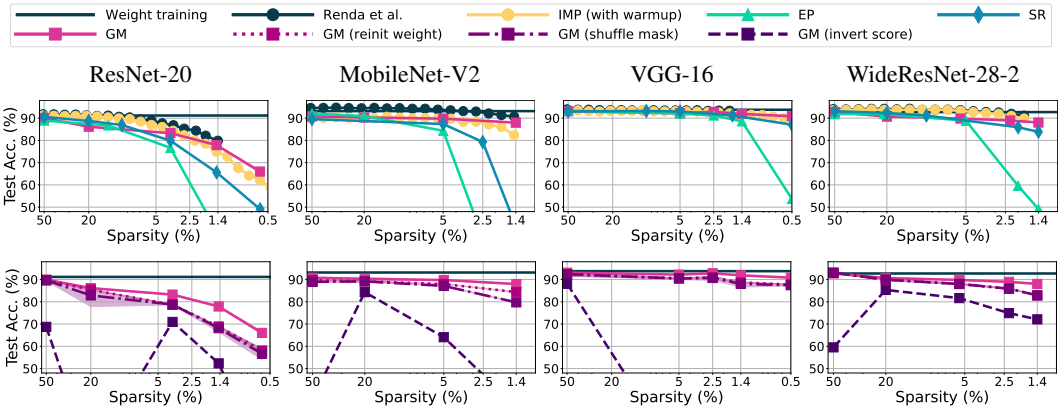


Figure 3: Performance of different pruning algorithms on CIFAR-10 for benchmark networks. Top: post-finetune accuracy; Bottom: sanity check methods suggested in Frankle et al. [10] applied on GEM-MINER (GM). GM achieves similar post-finetune accuracy as IMP, and typically outperforms it in the sparse regime. GM has higher post-finetune accuracy than EP and Smart Ratio (SR). GM also passes the sanity checks suggested in Frankle et al. [10]. Finally, GM (which prunes *at* init) nearly achieves the performance of Renda et al. (which is a pruning after training method) in the sparse regime, e.g., 1.4% sparsity in ResNet-20.

# 4 Experiments

In this section, we present the experimental results[1] for the performance of GEM-MINER across various tasks.

**Tasks.** We evaluate our algorithm on **(Task 1)** CIFAR-10 classification, on various networks including ResNet-20, MobileNet-V2, VGG-16, and WideResNet-28-2, **(Task 2)** TinyImageNet classification on ResNet-18 and ResNet-50, and **(Task 3)** Finetuning on the Caltech-101 [7] dataset using a ResNet-50 pretrained on ImageNet. The detailed description of the datasets, networks and hyperparameters can be found in Section A of the Appendix.

**Proposed scheme.** We run GEM-MINER with an $L_2$ regularizer. If a network reaches its best accuracy after $E$ epochs of weight training, then we run GEM-MINER for $E$ epochs to get a sparse subnetwork, and then run weight training on the sparse subnetwork for another $E$ epochs.

**Comparisons.** We tested our method against five baselines: dense weight training and four pruning algorithms: (i) IMP [9], (ii) Learning rate rewinding [29], denoted by Renda et al., (iii) Edge-Popup (EP) [28], and (iv) Smart-Ratio (SR) which is the random pruning method proposed by Su et al. [34].

We also ran the following sanity checks, proposed by Frankle et al. [10]: (i) (Random shuffling): To test if the algorithm prunes specific connections, we randomly shuffle the mask at every layer. (ii) (Weight reinitialization): To test if the final mask is specific to the weight initialization, we reinitialize the weights from the original distribution. (iii) (Score inversion): Since most pruning algorithms use a heuristic/score function as a proxy to measure the importance of different weights, we invert the scoring function to check whether it is a valid proxy. More precisely, this test involves pruning the weights which have the *smallest* scores rather than the largest. In all of the above tests, if the accuracy after finetuning the new subnetwork does not deteriorate significantly, then the algorithm is merely identifying optimal layerwise sparsities.

## 4.1 Rare gems obtained by GEM-MINER

**Task 1.** Fig. 3 shows the sparsity-accuracy tradeoff for various pruning methods trained on CIFAR-10 using ResNet-20, MobileNet-V2, VGG-16 and WideResNet-28-2. For each column (network), we compare IMP, IMP with learning rate rewinding (Renda et al.), GEM-MINER, EP, and SR in two performance metrics: the top row shows the accuracy of the subnetwork after weight training and bottom row shows the result of the sanity checks on GEM-MINER.

[1] Our codebase can be found at `https://anonymous.4open.science/r/pruning_is_enough-F0B0`.

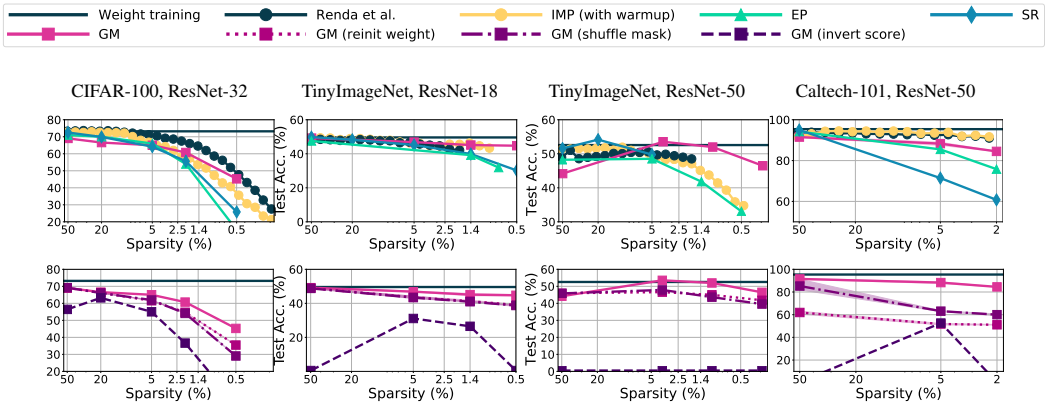


Figure 4: Accuracy on image classification tasks on TinyImageNet, ImageNet and Caltech-101. For Caltech-101, we pruned a pre-trained ImageNet model (ResNet-50). Top: post-finetune accuracy, bottom: sanity check methods suggested in Frankle et al. [10] applied on GEM-MINER.

As shown in the top row of Fig. 3, GEM-MINER finds a lottery ticket *at* initialization. It reaches accuracy similar to IMP after weight training. Moreover, for in the sparse regime (e.g., below 1.4% for ResNet-20 and MobileNet-V2), GEM-MINER outperforms IMP in terms of post-finetune accuracy. The bottom row of Fig. 3 shows that GEM-MINER passes the sanity check methods. For all networks, the performance in the sparse regime (1.4% sparsity or below) shows that the suggested GEM-MINER algorithm enjoys 3–10% accuracy gap with the best performance among variants. The results in the top row show that GEM-MINER far outperforms the random network with smart ratio (SR).

**Tasks 2–4.** Fig. 4 shows the sparsity-accuracy tradeoff for Tasks 2–4. Similar to Fig. 3, the top row reports the accuracy *after* weight training, and the bottom row contains the results of the sanity checks.

As shown in Fig. 4a and Fig. 4b, the results for **Task 2** show that (i) GEM-MINER achieves accuracy comparable to IMP as well as Renda et al. (IMP with learning rate rewinding) even in the sparse regime, (ii) GEM-MINER has non-trivial accuracy before finetuning (iii) GEM-MINER passes all the sanity checks, and (iv) GEM-MINER outperforms EP and SR. These results show that GEM-MINER successfully finds rare gems even in the sparse regime for **Task 2**.

Fig. 4c shows the result for **Task 3**. Unlike other tasks, GEM-MINER does not reach the post-finetune accuracy of IMP, but GEM-MINER enjoys over an 8% accuracy gap compared with EP and SR. Moreover, the bottom row shows that GEM-MINER has over 20% higher accuracy than the sanity checks below 5% sparsity showing that the subnetwork found by GEM-MINER is unique in this sparse regime.

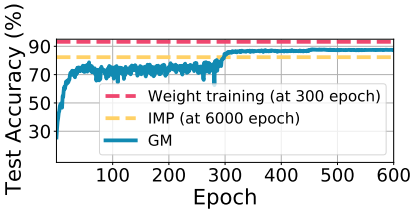


Figure 5: Convergence plot for CIFAR-10, MobileNet-V2 experiments, where we apply GEM-MINER for 300 epochs and then finetune the sparse model for another 300 epochs, to reach 1.16% sparse model. We added the accuracy of weight training (dense model) and IMP (1.4% sparse model) as a reference. Note that we compared with 300 epochs of weight training, and compared with IMP using 20 rounds of iterative pruning, i.e., 300 $\times$ 20 = 6000 epochs, to reach 1.4% sparsity. GEM-MINER achieves a higher accuracy than IMP despite its 19$\times$ shorter runtime to find a sparse subnetwork.

## 4.2 Comparison to ProsPr

Alizadeh et al. [1] recently proposed a pruning at init method called ProsPr which utilizes meta-gradients through the first few steps of optimization to determine which weights to prune, thereby accounting for the "trainability" of the resulting subnetwork. In Table 2 we compare it against GEM-MINER on ResNet-20, CIFAR-10 and also run the (i) Random shuffling and (ii) Weight reinitialization sanity checks from Frankle et al. [10]. We were unable to get ProsPr using their publicly available codebase to generate subnetworks at sparsity below 5% and therefore chose that sparsity. Note that GEM-MINER produces a subnetwork that is higher accuracy despite being more sparse. After finetuning for 150 epochs, our subnetwork reaches 83.4% accuracy while the subnetwork found by

Table 2: We compare ProsPr [1] vs GEM-MINER on ResNet-20, CIFAR-10 and run the random shuffling as well as the weight reinit sanity checks. Note that GEM-MINER produces a subnetwork that is higher accuracy despite being more sparse. Moreover, ProsPr does not show significant decay in performance after the sanity checks while GEM-MINER does. Therefore, it is likely that ProsPr is merely identifying good layerwise sparsity ratios.

| Algorithm | Sparsity | Accuracy after finetune | Accuracy after Random shuffling | Accuracy after Weight reinitialization |
|---|---|---|---|---|
| ProsPr | 5% | 82.67% | 82.15% | 81.64% |
| **GEM-MINER** | **3.72%** | **83.4%** | **78.73%** | **78.6%** |

ProsPr only reaches $82.67\%$ after training for 200 epochs. More importantly, ProsPr does not show significant decay in performance after the random reshuffling or weight reinitialization sanity checks. Therefore, as Frankle et al. [10] remark, it is likely that it is identifying good layerwise sparsity ratios, rather than a mask specific to the initialized weights.

### 4.3 Observations on GEM-MINER

**Convergence of accuracy and sparsity.** Fig. 5 shows how the accuracy of GEM-MINER improves as training progresses, for MobileNet-V2 on CIFAR-10 at sparsity $1.4\%$. This shows that GEM-MINER, reaches high accuracy even early in training, and can be finetuned to accuracy higher than that of IMP (which requires $19\times$ the runtime than our algorithm).

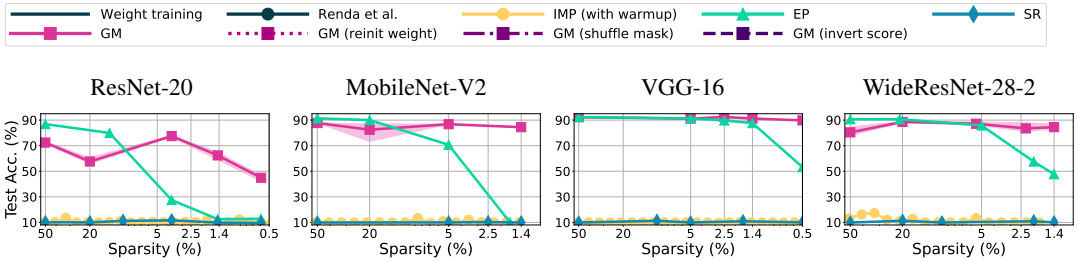


Figure 6: Performance of different pruning algorithms before finetuning on CIFAR-10 for benchmark networks. GEM-MINER finds subnetworks that already have reasonably high accuracy even before weight training. Note that, while IMP and SR have scarcely better than random guessing at initialization, subnetworks found by GEM-MINER typically perform even better than EP, especially in the sparse regime.

**High pre-finetune accuracy.** A shown in Fig. 6, GEM-MINER finds subnetworks at initialization that have a reasonably high accuracy even before the weight training, e.g., above 90% accuracy for 1.4% sparsity in VGG-16, and 85% accuracy for 1.4% sparsity in MobileNet-V2. Note that, in contrast, IMP and SR have accuracy scarcely better than random guessing at initialization. Clearly, GEM-MINER fulfills its objective in maximizing accuracy before finetuning and therefore finds rare gems – lottery tickets at initialization which already have high accuracy.

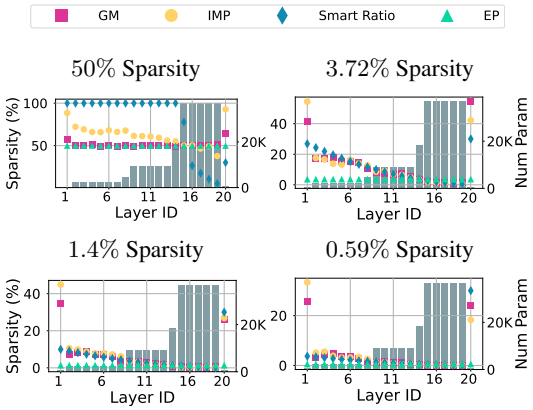


Figure 7: The layerwise sparsity for ResNet-20 pruned by GEM-MINER, IMP, Smart Ratio, and EP. The dark bar is the layerwise number of parameters. Both GEM-MINER and IMP save the most portion of parameter in the first layer and the last layer.

**Limitations of GEM-MINER.** We observed that in the dense regime (50% sparsity, 20% sparsity), GEM-MINER sometimes performs worse than IMP. While we believe that this can be resolved by appropriately tuning the hyperparameters, we chose to focus our attention on the sparse regime. We would also like to remark that GEM-MINER is fairly sensitive to the choice of hyperparameters and for some models, we had to choose different hyperparameters for each sparsity to ensure optimal performance. Though this occurs rarely, we also find that an extremely aggressive choice of $\lambda$ can lead to *layer-collapse* where one or more layers gets pruned completely. This happens when all the scores $\boldsymbol{p}$ of that layer drop below $0.5$.

Table 3: We construct different variants of EP and compare their performance with GEM-MINER, for ResNet-20, CIFAR-10, 0.59% sparsity. We establish that having a global score metric and gradually pruning is key to improved performance.

| Pruning Method | EP | Global EP | Global EP with Gradual Pruning | Global EP with Gradual Pruning and Regularization | GEM-MINER |
|---|---|---|---|---|---|
| **Pre-finetune acc (%)** | 19.57 | 22.22 | 31.56 | 19.67 | **45.30** |
| **Post-finetune acc (%)** | 24.47 | 34.42 | 63.54 | 63.72 | **66.15** |

**Layer-wise sparsity.** We compare the layer-wise sparsity pattern of different algorithms for ResNet-20 trained on CIFAR-10 in Fig. 7. Both GEM-MINER and IMP spend most of their sparsity budget on the first and last layers. By design, SR assigns $30\%$ sparsity to the last layer and the budget decays smoothly across the others. EP maintains the target sparsity ratio at each layer and therefore is always a horizontal line.

**How does GEM-MINER resolve EP's failings?** An open problem from Ramanujan et al. [28] is why the subnetworks found by EP are not fine-tunable. While GEM-MINER is significantly different from EP, it is reasonable to ask which modification allowed it to find lottery tickets without forgoing high accuracy at initialization. Table 3 shows how we can modify EP to improve the pre/post-finetune performance, for ResNet-20, CIFAR-10 at 0.59% sparsity. Here, we compare EP, GEM-MINER, as well as three EP variants that we construct. (i) *(Global EP)*: is a modification where the bottom-$k$ scores are pruned globally, not layer-wise. This allows the algorithm to trade-off sparsity in one layer for another. (ii) *(Gradual pruning)* reduces the parameter $k$ gradually as opposed to setting it to the target sparsity from the beginning. (iii) *(Regularization)*: we add an $L_2$ term on the score $p$ of the weights to encourage sparsity. The results indicate that *global pruning* and *gradual pruning* significantly improve both the pre and post-finetune accuracies of EP. Adding regularization does not improve the performance significantly. Finally, adding all three features to EP allows it to achieve 63.72% accuracy, while GEM-MINER reaches 66.15% accuracy. It is important to note that even with all three features, EP is inherently different from GEM-MINER in how it computes the *supermask* based on the scores. But we conjecture that aggressive, layerwise pruning is the key reason for EP's failings.

Table 4: Comparison of GEM-MINER and its longer version, for ResNet-20, CIFAR-10, 1.4% sparsity. LONG GEM-MINER, when given the same number of epochs improves post-finetune accuracy by 1.5%, rivaling the performance of Renda et al. [29]

| **Method** | GM (cold) | Long GM (cold) | IMP (warm) | Renda et al. (pruning after training) |
|---|---|---|---|---|
| **Number of Epochs** | 300 | 3000 | 3000 | 3000 |
| **Accuracy (%)** | 77.89 | 79.50 | 74.52 | 80.21 |

**Applying GEM-MINER for longer periods.** Recall that GEM-MINER uses $19\times$ fewer training epochs than iterative train-prune-retrain methods like IMP [9] and Learning rate rewinding (Renda et al. [29]), to find a subnetwork at $1.4\%$ sparsity which can then be trained to high accuracy. Here, we consider a long version of GEM-MINER to see if it can benefit if it is allowed to run for longer. Table 4 shows the comparison of post-finetune accuracy for GEM-MINER, LONG GEM-MINER, IMP and Renda et al. [29] tested on ResNet-20, CIFAR-10, sparsity=1.4% setting. Conventional GEM-MINER, applies iterative freezing every $5$ epochs to arrive at the target sparsity in $150$ epochs. LONG GEM-MINER instead prunes every $150$ epochs and therefore reaches the target sparsity in $3000$ epochs.

We find that applying GEM-MINER for longer periods improves the post-finetune accuracy in this regime by 1.5%. This shows that given equal number of epochs, GEM-MINER, which prunes at initialization, can close the gap to Learning rate rewinding [29] which is a prune-after-training method.

## 5 Conclusion

In this work, we resolve the open problem of pruning at initialization by proposing GEM-MINER that finds *rare gems* – lottery tickets *at initialization* that have non-trivial accuracy even before finetuning and accuracy rivaling prune-after-train methods after finetuning. Unlike other methods, subnetworks found by GEM-MINER pass all known sanity checks and baselines. Moreover, we show that GEM-MINER is competitive with IMP despite not using warmup and up to $19\times$ faster.

# References


[1] Alizadeh, M., Tailor, S. A., Zintgraf, L. M., van Amersfoort, J., Farquhar, S., Lane, N. D., and Gal, Y. Prospect pruning: Finding trainable weights at initialization using meta-gradients. In *International Conference on Learning Representations*, 2021.

[2] Bengio, Y., Léonard, N., and Courville, A. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[3] Chen, T., Frankle, J., Chang, S., Liu, S., Zhang, Y., Wang, Z., and Carbin, M. The lottery ticket hypothesis for pre-trained bert networks. *arXiv preprint arXiv:2007.12223*, 2020.

[4] Chen, T., Sui, Y., Chen, X., Zhang, A., and Wang, Z. A unified lottery ticket hypothesis for graph neural networks. In *International Conference on Machine Learning*, pp. 1695–1706. PMLR, 2021.

[5] Courbariaux, M., Bengio, Y., and David, J.-P. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pp. 3123–3131, 2015.

[6] Diffenderfer, J. and Kailkhura, B. Multi-prize lottery ticket hypothesis: Finding accurate binary neural networks by pruning a randomly weighted network. In *International Conference on Learning Representations*, 2020.

[7] Fei-Fei, L., Fergus, R., and Perona, P. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *2004 conference on computer vision and pattern recognition workshop*, pp. 178–178. IEEE, 2004.

[8] Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[9] Frankle, J., Dziugaite, G. K., Roy, D., and Carbin, M. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning*, pp. 3259–3269. PMLR, 2020.

[10] Frankle, J., Dziugaite, G. K., Roy, D. M., and Carbin, M. Pruning neural networks at initialization: Why are we missing the mark? *arXiv preprint arXiv:2009.08576*, 2020.

[11] Gale, T., Elsen, E., and Hooker, S. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

[12] Girish, S., Maiya, S. R., Gupta, K., Chen, H., Davis, L. S., and Shrivastava, A. The lottery ticket hypothesis for object recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 762–771, 2021.

[13] Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 2015.

[14] Han, S., Mao, H., and Dally, W. J. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv:1510.00149 [cs]*, February 2016. URL http://arxiv.org/abs/1510.00149.

[15] Hassibi, B. and Stork, D. G. Second order derivatives for network pruning: Optimal Brain Surgeon. In Hanson, S. J., Cowan, J. D., and Giles, C. L. (eds.), *Advances in Neural Information Processing Systems 5*, pp. 164–171. Morgan-Kaufmann, 1993.

[16] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[17] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks. In *Advances in neural information processing systems*, pp. 4107–4115, 2016.

[18] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[19] Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

[20] LeCun, Y., Denker, J. S., and Solla, S. A. Optimal Brain Damage. In Touretzky, D. S. (ed.), *Advances in Neural Information Processing Systems 2*, pp. 598–605. Morgan-Kaufmann, 1990. URL http://papers.nips.cc/paper/250-optimal-brain-damage.pdf.

[21] Lee, N., Ajanthan, T., and Torr, P. H. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

[22] Li, A., Sun, J., Wang, B., Duan, L., Li, S., Chen, Y., and Li, H. Lotteryfl: Personalized and communication-efficient federated learning with lottery ticket hypothesis on non-iid datasets. *arXiv preprint arXiv:2008.03371*, 2020.

[23] Ma, X., Yuan, G., Shen, X., Chen, T., Chen, X., Chen, X., Liu, N., Qin, M., Liu, S., Wang, Z., et al. Sanity checks for lottery tickets: Does your winning ticket really win the jackpot? *Advances in Neural Information Processing Systems*, 34, 2021.

[24] Malach, E., Yehudai, G., Shalev-Schwartz, S., and Shamir, O. Proving the lottery ticket hypothesis: Pruning is all you need. In *International Conference on Machine Learning*, pp. 6682–6691. PMLR, 2020.

[25] Mozer, M. C. and Smolensky, P. Skeletonization: A Technique for Trimming the Fat from a Network via Relevance Assessment. In Touretzky, D. S. (ed.), *Advances in Neural Information Processing Systems 1*, pp. 107–115. Morgan-Kaufmann, 1989.

[26] Orseau, L., Hutter, M., and Rivasplata, O. Logarithmic pruning is all you need. *Advances in Neural Information Processing Systems*, 33, 2020.

[27] Pensia, A., Rajput, S., Nagle, A., Vishwakarma, H., and Papailiopoulos, D. Optimal lottery tickets via subsetsum: Logarithmic over-parameterization is sufficient. *Advances in neural information processing systems*, 2020.

[28] Ramanujan, V., Wortsman, M., Kembhavi, A., Farhadi, A., and Rastegari, M. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11893–11902, 2020.

[29] Renda, A., Frankle, J., and Carbin, M. Comparing rewinding and fine-tuning in neural network pruning. *arXiv preprint arXiv:2003.02389*, 2020.

[30] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[31] Simons, T. and Lee, D.-J. A review of binarized neural networks. *Electronics*, 8(6):661, 2019.

[32] Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[33] Sreenivasan, K., Rajput, S., Sohn, J.-y., and Papailiopoulos, D. Finding everything within random binary networks. *arXiv preprint arXiv:2110.08996*, 2021.

[34] Su, J., Chen, Y., Cai, T., Wu, T., Gao, R., Wang, L., and Lee, J. D. Sanity-checking pruning methods: Random tickets can win the jackpot. *arXiv preprint arXiv:2009.11094*, 2020.

[35] Tanaka, H., Kunin, D., Yamins, D. L., and Ganguli, S. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in Neural Information Processing Systems*, 33, 2020.

[36] Wang, C., Zhang, G., and Grosse, R. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2019.

[37] Zagoruyko, S. and Komodakis, N. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[38] Zhu, M. and Gupta, S. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

# Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Refer Section 4.3
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Our codebase can be found at `https://anonymous.4open.science/r/pruning_is_enough-F0B0`
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Refer the appendix
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Wherever it was computationally feasible, we ran 3 trials and report the plots with an error bar
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Refer the appendix
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We cite the original papers that introduced ResNets, VGGs etc.
   (b) Did you mention the license of the assets? [Yes] We cite the original papers that introduced CIFAR-10, TinyImagenet etc.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]