BACK TO BASICS: EFFICIENT NETWORK COMPRESSION VIA IMP

Anonymous authors
Paper under double-blind review

ABSTRACT

Network pruning is a widely used technique for effectively compressing Deep Neural Networks with little to no degradation in performance during inference. Iterative Magnitude Pruning (IMP) (Han et al., 2015) is one of the most established approaches for network pruning, consisting of several iterative training and pruning steps, where a significant amount of the network's performance is lost after pruning and then recovered in the subsequent retraining phase. While commonly used as a benchmark reference, it is often argued that a) it reaches suboptimal states by not incorporating sparsification into the training phase, b) its global selection criterion fails to properly determine optimal layer-wise pruning rates and c) its iterative nature makes it slow and non-competitive. In light of recently proposed retraining techniques, we investigate these claims through rigorous and consistent experiments where we compare IMP to pruning-during-training algorithms, evaluate proposed modifications of its selection criterion and study the number of iterations and total training time actually required. We find that IMP with SLR (Le & Hua, 2021) for retraining can outperform state-of-the-art pruning-during-training approaches without or with only little computational overhead, that the global magnitude selection criterion is largely competitive with more complex approaches and that only few retraining epochs are needed in practice to achieve most of the sparsity-vs.-performance trade-off of IMP. Our goals are both to demonstrate that basic IMP can already provide state-of-the-art pruning results on par or outperforming more complex or heavily parameterized approaches and also to establish a more realistic yet easily realisable baseline for future research.

1 INTRODUCTION

Modern Neural Network architectures are commonly highly over-parameterized (Zhang et al., 2016), containing millions or even billions of parameters, resulting in both high memory requirements as well as computationally intensive and long training and inference times. It has been shown however (LeCun et al., 1989; Hassibi & Stork, 1992; Han et al., 2015; Gale et al., 2019; Lin et al., 2020; Blalock et al., 2020) that modern architectures can be compressed dramatically by *pruning*, i.e., removing redundant structures such as individual weights, entire neurons or convolutional filters. The resulting *sparse* models require only a fraction of storage and floating-point operations (FLOPs) for inference, while experiencing little to no degradation in predictive power compared to the dense model. It has also been observed that pruning can have a regularizing effect and be beneficial to the generalization capacities (Blalock et al., 2020; Hoefler et al., 2021).

There is of course an inherent trade-off between sparsity and model performance; a very heavily pruned model will normally be less performant than its dense (or moderately pruned) counterpart. To obtain optimal performance while benefiting from the advantages of a pruned network at inference, two different algorithmic paradigms have emerged. The first is characterized by *pruning after training* (Han et al., 2015; Li et al., 2016; Luo et al., 2017): the network is pruned after a standard training process, seemingly losing most of its predictive performance, and then retrained to compensate for that pruning-induced loss. This can be done either once (*One Shot*), or the process of pruning and retraining can be repeated iteratively until the desired level of sparsity is reached. This approach was most notably proposed by Han et al. (2015) in the form of ITERATIVE MAGNITUDE PRUNING (IMP).

The second paradigm is defined by its ability to find a well-performing pruned model *during the training procedure*. This can either be achieved by gradual pruning during training (Zhu & Gupta, 2017; Gale et al., 2019; Lin et al., 2020), i.e., extending the pruning mask dynamically, or by employing regularization- and constraint-optimization techniques (Louizos et al., 2017; Carreira-Perpinán & Idelbayev, 2018; Ding et al., 2019; Savarese et al., 2019; Kusupati et al., 2020) to learn an almost sparse structure throughout training such that the actual 'hard' pruning, i.e., the ultimate removal of certain parameters, induces almost no drop in accuracy. Methods of this class reach a sparse model at the end of training, ideally eliminating the need for further training.

Despite or probably because it arguably is the simplest pruning algorithm, IMP has been widely applied as a baseline comparison for approaches that learn the sparsity pattern throughout training (Carreira-Perpinán & Idelbayev, 2018; Ding et al., 2019; Savarese et al., 2019; Siegel et al., 2020; Hoefler et al., 2021). As such, it is often subject to criticism, with the most commonly made claims arguing against the efficacy of IMP being the following:

1. Throughout regular training IMP is designed to be agnostic of the desired sparsity, that is it does not have any impact on the solution the optimizer converges to. Many proposed improvements claim that *the sparsification should be part of the training* to reduce the impact of the actual 'hard' pruning, which results in a "failure to properly recover the pruned weights" (Liu et al., 2020). Especially not requiring additional retraining epochs is frequently proposed as an advantage, e.g., Ding et al. (2019) advertise that there is "no need for a time consuming re-training", but it is also argued that IMP achieves sub-optimal states since learning the pruning set throughout training "helps find a better subset and hence prune more weights with no or little loss degradation" (Carreira-Perpinán & Idelbayev, 2018).

2. IMP determines a single numerical threshold for pruning and applies it globally to every parameter, potentially resulting in very different levels of sparsity among the layers of the network. This behavior is often considered to be sub-optimal and it is argued that *pruning should be layer-dependent* (Liu et al., 2020), so more complex saliency criteria have been proposed (Gale et al., 2019; Lee et al., 2020). Kusupati et al. (2020) for example claim that "uniform or heuristic non-uniform sparsity budgets (...) have sub-optimal layer-wise parameter allocation resulting in a) lower prediction accuracy or b) higher inference cost (FLOPs)".

3. While only the iterative approach, that is repeatedly removing only a small fraction of the parameters followed by extensive retraining, is said to achieve results on the Pareto frontier (Renda et al., 2020), its iterative nature is also considered to be *computationally tedious, if not impractical*: "iterative pruning is computationally intensive, requiring training a network 15 or more times consecutively for multiple trials" (Frankle & Carbin, 2018), leading Liu et al. (2020) to trying to "avoid the expensive pruning and fine-tuning iterations".

Our interest lies in exploring these claimed disadvantages of IMP through rigorous and consistent computational experimentation with a focus on recent advancements concerning the retraining phase (Renda et al., 2020; Le & Hua, 2021). This comparative study is in fact intended to complement both of these works, which focused on improving the sparsity vs. performance of IMP through improved learning rate schemes during training, by putting an additional spotlight on the total computational cost of IMP in a direct comparison with methods that are commonly assumed to outperform IMP in that aspect by avoiding retraining.

Contributions. We empirically find that, using an appropriate learning rate scheme, only few retraining epochs are needed in practice to achieve most of the sparsity vs. performance trade-off of IMP. We also find that the global selection criterion not only finds sparsity distributions on par with but, somewhat surprisingly, often better than those of more sophisticated layer-dependent pruning criteria. Finally, we conclude that, using an appropriate learning rate scheme, IMP is on par with state-of-the-art approaches that incorporate sparsity into the training *without or with only little computational overhead*. That is, not only can IMP find some of the best performing architectures at any given sparsity level, but due to the compressed retraining time it does so without needing to leverage a longer running time.

Outline. Section 2 contains a complete overview over related works, including a brief summary of all pruning methods and approaches considered here. In Section 3 we then provide the computational results and their interpretation. We conclude with some discussion in Section 4 including aspects that might be of interest for future research.

2 Overview of Pruning methods and Methodology

While the sparsification of Neural Networks includes a wide variety of approaches, we will focus on *Model Pruning*, i.e., the removal of redundant structures in a Neural Network. More specifically, our results will be limited to *unstructured* pruning, that is the removal of individual weights, as opposed to its *structured* counterpart, where entire groups of elements, such as neurons or convolutional filters, are removed. We will also focus on approaches that start with a dense network and then either prune the network *during* training and or prune *after* training as already discussed in the introduction. Following Bartoldson et al. (2019), we will also refer to methods of the former category as *pruning stable*, since the final pruning should result in a negligible decrease in performance, where methods of the latter category are referred to as *unstable*.

Pruning unstable methods are exemplified by Iterative Magnitude Pruning (IMP) (Han et al., 2015). In its original form, it first employs standard network training, adding a common ℓ_2-regularization term on the objective, and then removes all weights from the network whose absolute values are below a certain threshold. The network at this point commonly loses some or even all of its learned predictive power, so it is then retrained for a fixed number of epochs. This prune-retrain cycle is usually repeated a number of times; the threshold at every pruning step is determined as the appropriate percentile such that, at the end of given number of iterations, a desired target sparsity is met.[1] In the following we will first discuss two particular details of IMP that have been the focus of recent research: the questions of (a) how to select the parameters to be pruned and (b) how to retrain. We will then conclude this section by briefly outlining the set of pruning stable methods we have chosen for this comparison and establish how to fairly compare them to IMP.

Retraining approaches. Let us first consider the learning rate scheme used during retraining. The original approach by Han et al. (2015) is commonly referred to as Fine Tuning (FT): suppose we train for T epochs using the learning rate schedule $(\eta_t)_{t \leq T}$ and retrain for T_{rt} epochs per prune-retrain-cycle, then FT retrains the pruned network for T_{rt} epochs using a fixed constant learning rate, most commonly η_T. It was first noticed by Renda et al. (2020) that the precise learning rate schedule during retraining can have a dramatic impact on the predictive performance of the pruned network. Motivated by Weight Rewinding (WR) (Frankle et al., 2019), they proposed Learning Rate Rewinding (LRW), where one retrains the pruned network for T_{rt} epochs using the last $T - T_{\text{rt}}$ learning rates $\eta_{T-T_{\text{rt}}+1}, \ldots, \eta_T$. Le & Hua (2021) argued that the reason behind the success of LRW is the usage of large learning rates and proposed Scaled Learning Rate Restarting (SLR), where the pruned network is retrained using a proportionally identical learning schedule, i.e., by compressing $(\eta_t)_{t \leq T}$ into the retraining time frame of T_{rt} epochs with a short warm-up phase. They also introduced Cyclic Learning Rate Restarting (CLR) based on the the 1-cycle learning rate schedule of Smith & Topin (2017).

While One Shot IMP, that is IMP with a single prune-retrain cycle, is a viable approach to model pruning, only the iterative approach (with multiple prune-retrain cycles) has been shown to achieve state-of-the-art accuracy-vs.-sparsity tradeoffs (Han et al., 2015; Renda et al., 2020). This iterative approach however commonly consists of a significant amount of prune-retrain cycles, each needing the full original training time, that is $T = T_{\text{rt}}$, resulting in several thousand epochs worth of total training time. Renda et al. (2020) for example suggested the following approach: train a network for T epochs and then iteratively prune 20% percent of the weights and retrain for $T_{\text{rt}} = T$ epochs using LRW, i.e., use the same learning rate scheme as during training, until the desired sparsity is reached. We note that for $T = T_{\text{rt}}$ the learning rate scheme of SLR becomes essentially identical to that of LRW. For a goal sparsity of 98% and $T = 200$ original training epochs, the algorithm would

[1] There is another way to view this relation: one can fix a given percentile to be pruned in every iteration and then simply repeat the prune-retrain cycle until either a desired level of sparsity is reached or the performance degradation exceeds a given threshold. This is in fact how it appears to be used by Han et al. (2015) and how it is for example presented by Renda et al. (2020). While this reframing may appear trivial, it in fact highlights a strength of IMP that we will further emphasize when contrasting it with pruning stable approaches.

therefore require 18 prune-retrain-cycles for a massive 3800 total retrain epochs. In Subsection 3.1 we will study the effect of the number of prune-retrain cycles, the number of retraining epochs and the learning rate scheme on the performance of the pruned network to establish whether IMP truly requires this massive amount of computational investment to develop its full potential.

Pruning approaches. IMP in its original form treats all trainable parameters as a single vector and computes a global threshold below which parameters are removed, independent of the layer they belong to. This simple approach, which we will refer to as GLOBAL, has been subject to criticism for not determining optimal layer-dependent pruning rates and for being inconsistent (Liu et al., 2020). Fully-connected layers for example have many more parameters than convolutional layers and are therefore much less sensitive to weight removal (Han et al., 2015; Carreira-Perpinán & Idelbayev, 2018). Further, it has been observed that the position of a layer can play a role in whether that layer is amenable to pruning: often first and last layers are claimed to be especially relevant for the classification performance (Gale et al., 2019). On the other hand, in which layers pruning takes place significantly impacts the sparsity-induced theoretical speedup (Blalock et al., 2020). Lastly, the non-negative homogeneity of modern ReLU-based neural network architectures (Neyshabur et al., 2015) would also seem to indicate a certain amount of arbitrariness to this heuristic selection rule, or at least a strong dependence on the network initialization rule and optimizer used, as weights can be rescaled to force it to fully remove all parameters of a layer, destroying the pruned network without having affected the output of the unpruned network.

Determining which weights to remove is hence crucial for successful pruning and several methods have been designed to address this fact. Zhu & Gupta (2017) introduced the UNIFORM allocation, in which a global sparsity level is enforced by pruning each layer to exactly this sparsity. Gale et al. (2019) extend this approach in the form of UNIFORM+ by (a) keeping the first convolutional layer dense and (b) pruning at most 80% of the connections in the last fully-connected layer. Evci et al. (2019) propose a reformulation of the ERDŐS-RÉNYI KERNEL (ERK) (Mocanu et al., 2018) to take the layer and kernel dimensions into account when determining the layerwise sparsity distribution. In particular, ERK allocates higher sparsity to layers with more parameters. Finally, Lee et al. (2020) propose LAYER-ADAPTIVE MAGNITUDE-BASED PRUNING (LAMP), an approach which takes an ℓ_2-distortion perspective by relaxing the problem of minimizing the output distortion at time of pruning with respect to the worst-case input. We note that we follow the advice of Evci et al. (2019) and Dettmers & Zettlemoyer (2019) and do not prune biases and batch-normalization parameters, since they only amount to a negligible fraction of the total weights, however keeping them has a very positive impact on the performance of the learned model.

We will compare these approaches in Subsection 3.2 with a focus on the impact of the retraining phase. Since Le & Hua (2021) found that SLR can be used to obtain strong results even when pruning convolutional filters randomly, i.e., by assigning random importance scores to the filters instead of using the magnitude criterion or others, we are interested in understanding the importance the retraining technique has on the sparsity distribution.

A fair comparison to pruning-stable methods. Pruning stable algorithms induce a strong implicit bias during training. For example, LEARNING COMPRESSION (LC) by Carreira-Perpinán & Idelbayev (2018) and GLOBAL SPARSE MOMENTUM (GSM) by Ding et al. (2019) both employ a modification of weight decay and force the k weights with the smallest score more rapidly towards zero, where k is the number of parameters that will eventually be pruned and the score is the parameter magnitude or its product with the loss gradient. Similarly, DISCOVERING NEURAL WIRINGS (DNW) by Wortsman et al. (2019) zeroes out the smallest k weights in the forward pass while still using a dense gradient. CONTINUOUS SPARSIFICATION (CS) by Savarese et al. (2019), SOFT THRESHOLD WEIGHT REPARAMETERIZATION (STR) by Kusupati et al. (2020) and DYNAMIC SPARSE TRAINING (DST) (Liu et al., 2020) all rely on the creation of additional trainable threshold parameters, which are applied to sparsify the model while being regularly trained alongside the usual weights. Here, the training objectives are modified via penalty terms to control the sparsification. GRADUAL MAGNITUDE PRUNING (GMP) (Zhu & Gupta, 2017) follows a tunable pruning schedule which sparsifies the network throughout training by dynamically extending and updating a pruning mask. Based on this idea, DYNAMIC PRUNING WITH FEEDBACK (DPF) by Lin et al. (2020) maintains a pruning mask which is extended using the pruning schedule of Zhu &

Table 1: Exact training configurations used throughout the experiments for IMP. For pruning stable methods we use the same settings, with the following exceptions: (1) we tune weight decay over 1e-4, 5e-4, 1e-3 and 5e-3 , (2) since momentum plays a crucial part in the justification of GSM, we tune it over 0.9, 0.99 and 0.995 and (3) any additional hyperparameters of the respective methods were tuned as indicated in Subsection A.2.

Dataset	Network (number of weights)	Epochs	Batch size	Momentum	Weight decay	Learning rate (t = training epoch)	Unpruned test accuracy
CIFAR-10	ResNet-56 (850 K) VGG-16 (138 Mio)	200	128	0.9	5e-4	$\eta_t = \begin{cases} 0.1 & t \in [1, 90], \\ 0.01 & t \in [91, 180], \\ 0.001 & t \in [181, 200] \end{cases}$	93.6% ±0.3% 93.8% ±0.2%
CIFAR-100	WRN28x10 (37 Mio)	200	128	0.9	1e-4	$\eta_t = \begin{cases} 0.1 & t \in [1, 60], \\ 0.02 & t \in [61, 120], \\ 0.004 & t \in [121, 160], \\ 0.0008 & t \in [161, 200] \end{cases}$	76.7% ±0.2%
ImageNet	ResNet-50 (26 Mio)	90	256	0.9	1e-4	$\eta_t = \begin{cases} 0.1\frac{t}{5} & t \in [1, 5], \\ 0.1 & t \in [5, 30], \\ 0.01 & t \in [31, 60], \\ 0.001 & t \in [61, 80], \\ 0.0001 & t \in [81, 90] \end{cases}$	76.17% ±0.03%

Gupta (2017), but allows for error compensation by modifying the update rule to use the (stochastic) gradient of the pruned model while updating the dense parameters.

The first claimed advantage of pruning stable methods is the lack of a need for retraining, as they produce a sparse and well-performing model at the end of regular training. This however comes at a price: the implicit biases in pruning stable algorithms result in computational overhead when compared to the usual network training, i.e., to find a sparse solution throughout training through extensive regularization, masking or other methods, the per-iteration computational cost is increased. An overview of the computational overhead of the previously introduced methods can be found in Table 2 in the appendix. Neither training epochs nor iterations are therefore an appropriate measure when comparing algorithms, since what matters is the total time required to obtain a sparse network. Gale et al. (2019), Evci et al. (2019) and Lin et al. (2020) consider this when presenting their results. The second claimed advantage of pruning stable methods is that they find a better pruning set than unstable methods that cannot recover mistakenly pruned weights (Liu et al., 2020) and whose heuristic selection mechanism is disadvantaged compared to methods that learn the pruning thresholds (Kusupati et al., 2020). While it seems reasonable to believe that the less performance lost at pruning, the better, Bartoldson et al. (2019) showed that this rationale might not be entirely sound: pruning seems to behave similar to noise injection and there exists a *generalization-stability-tradeoff* indicating that less stability can actually be beneficial to generalization.

Especially given the previously discussed recent advancements in better recovering pruning-induced losses during retraining, both the narrative that the need for retraining is a clear computational disadvantage and that pruning stable methods find better pruning sets stand to be reevaluated. We will do so in Subsection 3.3.

3 Experimental results

Let us outline the general methodological approach to computational experiments in this section, including datasets, architectures, metrics and optimization strategies used. Experiment-specific details are found in the respective subsections. We note that, given the surge of interest in pruning, Blalock et al. (2020) proposed experimental guidelines in the hope of standardizing the experimental setup. We aim to follow these guidelines whenever possible and encourage others to do so as well. All experiments performed throughout this computational study are based on the PyTorch framework (Paszke et al., 2019), using the original code of the methods whenever possible. All results and metrics were logged and analyzed using *Weights & Biases* (Biewald, 2020). We plan on publicly releasing our implementations and general setup for the sake of reproducibility.

We restrict ourselves to the task of image recognition on the publicly available *CIFAR-10* and *CIFAR-100* (Krizhevsky et al., 2009) datasets as well as *ImageNet* (Russakovsky et al., 2015). All experiments are performed using deep convolutional neural network architectures, in particular on *Residual Networks* (He et al., 2015), *Wide Residual Networks* (WRN) (Zagoruyko & Komodakis, 2016) and *VGG-16* (Simonyan & Zisserman, 2014). We use Stochastic Gradient Descent (SGD) with momentum as an optimizer throughout the experiments and exclusively use stepped learning rate schedules. We therefore also decided to forego CLR in favor SLR despite the recommendations of Le & Hua (2021). Exact parameters can be found in Table 1.[2]

The focus of our analysis will be the trade-off between the model sparsity and the final test accuracy. As a secondary measure, we will also consider the *theoretical speedup* (Blalock et al., 2020) induced by the sparsity, which is defined as the ratio of the FLOPs needed to evaluate the sparse over the dense model assuming a theoretical matrix multiplication algorithm capable of taking full advantage of even unstructured sparsity, see Subsection A.1 for full details. We decided to include it since it can help to give a more differentiated picture of the inherent trade-offs.

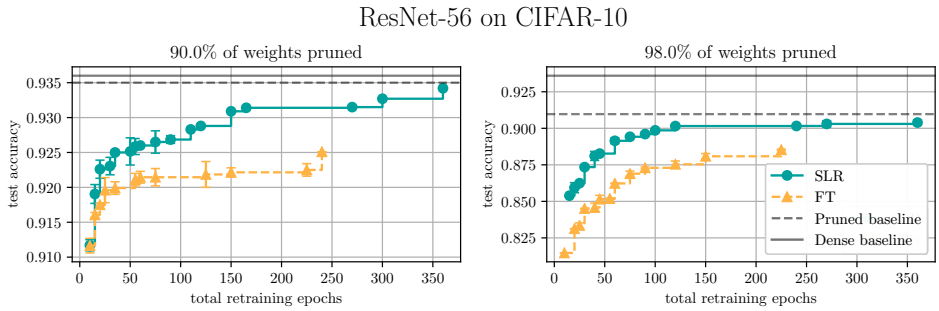

Figure 1: Test accuracy achieved by IMP in relation to the total number of epochs required for retraining using either FT or SLR. The pruned baselines were established by Renda et al. (2020). For each point on the x-axis, we show the highest mean accuracy achieved by any configuration of a grid search using up to this many total retraining epochs.

3.1 The Computational Cost of IMP

In this part we will treat the number of retrain epochs per prune-retrain cycle T_{rt} as well as the total number of prune-retrain cycles J as tunable hyperparameters for IMP and try to determine the trade-off between the predictive performance of the final pruned network and the total number of retrained epochs $J \cdot T_{\text{rt}}$. As a baseline performance for a pruned network, we will use the approach suggested by Renda et al. (2020). While it can be debated whether this approach reaches state-of-the-art results, it still serves as a good benchmark for the current potential of IMP. We will use the original global pruning criterion of Han et al. (2015) throughout this part.

In Figure 1 we present the results of our computations for ResNet-56 trained on CIFAR-10 with a moderate and high target sparsity of respectively 90% and 98%. The parameters for the retrain phase were optimized using a grid search over $T_{\text{rt}} \in \{10, 15, 20, ..., 60\}$ and $J \in \{1, 2, \ldots, 6\}$ and we consider both FT and SLR as learning rate schemes during retraining. For the pruned baseline we use the values reported by Renda et al. (2020) and note that they used 182 instead of 200 training epochs as well as a weight decay value of 2e-4 instead of 5e-4. All of our results are averaged over 2 seeds with max-min-bands indicated.

Summarizing the results, we find that SLR significantly outperforms FT at both levels of sparsity. Furthermore, SLR comes within half a percentage point of the pruned baseline within a mere 200 retraining epochs, while still obtaining strong results within 100 epochs. This stands in stark contrast to the full 1820 and 2912 retraining epochs required to establish the respective baseline. IMP therefore seems to achieve most of its respective potential with significantly less than the total number of retraining epochs usually budgeted for its full iterative form. Unlike commonly assumed, it

[2]Others have reported an accuracy of around 80% for WRN28x10 trained on CIFAR-100 that we were unable to replicate. The discrepancy is most likely due to an inconsistency in PyTorch's dropout implementation.

therefore seems at least competitive with other methods not just with respect to the predictive power of the pruned network it obtains but also with respect to its total computational cost.

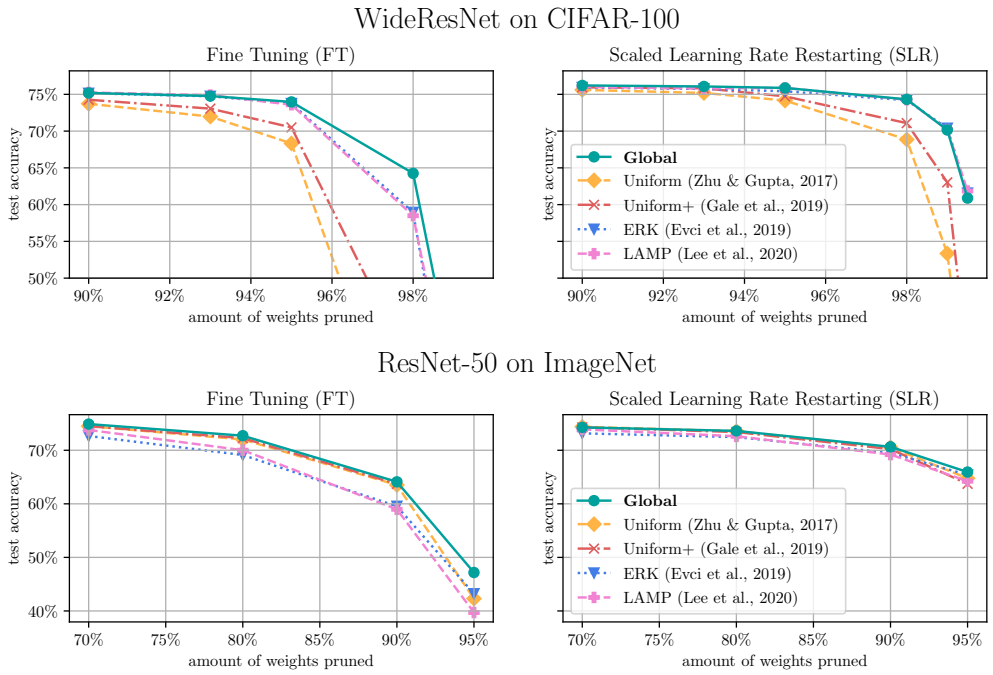

Figure 2: Performance-vs.-sparsity tradeoffs for the One Shot setting on CIFAR-100 (top) and ImageNet (bottom). We compare the sparsity allocation methods w.r.t. the different retraining techniques FT (left) and SLR (right).

3.2 The Importance of the Sparsity Distribution

We compare the original global pruning criterion of IMP to the previously introduced proposed alternatives. Figure 2 reports the test accuracy in relation to the level of sparsity in the One Shot setting for CIFAR-100, where the network is retrained for 30 epochs, and ImageNet, where it is retrained for 10 epochs. We tested both FT (Han et al., 2015) and SLR (Le & Hua, 2021) to see if the learning rate scheme during retraining has any impact on the performance of the pruning selection scheme. The CIFAR-100 results are averaged over three seeds. Plots including confidence bands and the theoretical speedup can be found in Subsection B.2 in the appendix. For ImageNet, the results are based on a single seed.

Surprisingly the simple global selection criterion performs at least on par with the best out of all tested methods at any sparsity level for every combination of dataset and architecture tested here. Using SLR during retraining compresses the results by equalizing performance, but otherwise does not change the overall picture. We note that the results on CIFAR-100 using FT largely track with those reported by Lee et al. (2020), with the exception of the strong performance of the global selection criterion. Apart from slightly different network architectures, we note that they used significantly more retraining epochs, e.g., 100 instead of 30, and that they use AdamW (Loshchilov & Hutter, 2017) instead of SGD. Comparing the impact different optimizers can have on the pruning selection schemes seems like a potentially interesting direction for future research.

The global selection criterion has previously been reported to suffer from a pruning-induced collapse at very high levels of sparsity in certain network architectures, a phenomenon that has been coined *layer-collapse* by Tanaka et al. (2020), who hypothesize that it can be avoided by using smaller pruning steps since gradient descent restabilizes the network after pruning by following a layer-wise magnitude conservation law. To verify this, we trained a VGG-16 network on CIFAR-10, as also done by Lee et al. (2020), both in the One Shot and in the iterative setting. In the One Shot setting the model is retrained for 30 epochs after pruning and the iterative setting consists of 3 prune-retrain

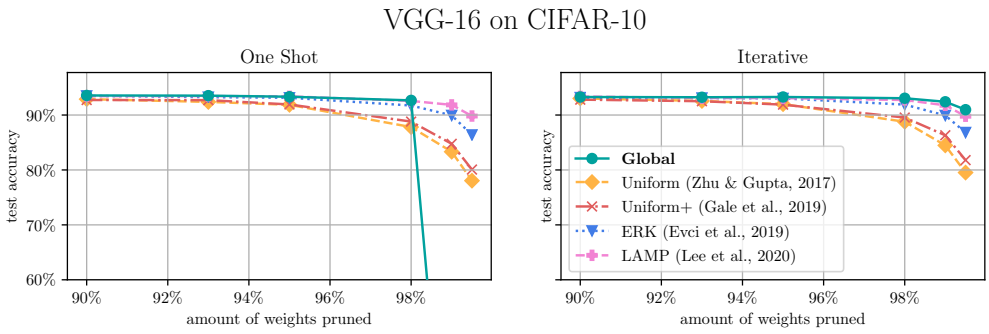

Figure 3: Performance-vs.-Sparsity tradeoffs for VGG-16 on CIFAR-10 for IMP in the One Shot (left) and iterative (right) setting using SLR. For One Shot we observe *layer-collapse* while the iterative splitting into less severe pruning steps avoids the problem.

cycles with 10 epochs each. The results using SLR are reported in Figure 3 and results using FT largely show the same behavior and can be found in Subsection B.2 in the appendix.

Interpreting the results, layer collapse is clearly occurring here for the global selection criterion at sparsity levels above 98% in the One Shot setting, but disappears entirely when pruning iteratively. Note that the total amount of retraining epochs between the two settings is identical here. This indicates that layer collapse, while a genuine potential issue, can be avoided even using the global selection criterion.

3.3 Comparing IMP to Pruning Stable Approaches

Using the lessons from the previous two sections, we finally compare IMP to recent pruning stable approaches. Given the high computational demand of tuning nine different methods, we limit ourselves to ResNet-56 networks trained on CIFAR-10. For IMP we employ the global selection criterion and retrain using SLR, where the number of prune-retrain-cycles J as well as the number of retraining epochs per cycle T_{rt} were tuned using a grid search over $T_{\text{rt}} \in \{10, 15, 20, 25\}$ and $J \in \{1, 2, 3, 4\}$. We will also consider a restricted version of IMP, where we impose the additional constraint $J \cdot T_{\text{rt}} \leq 30$ to obtain a method that will be largely comparable to pruning stable methods w.r.t. its total run time.

The hyperparameters of each of the pruning stable methods were likewise tuned using manually defined grid searches, resorting to the recommendations of the original publications whenever possible, see Subsection A.2 for exact details. Whenever possible, that is for GMP, GSM, DNW, DPF, and LC, we give the methods predetermined sparsity levels of 90%, 93%, 95% and 98%, the same as given to IMP. Tuning the remaining methods in a way that allows for a fair comparison however is significantly more difficult, since none of them allow to clearly specify a desired level of sparsity but instead require one to tune additional hyperparameters as part of the grid search.[3] Despite our best efforts, we were only able to cover part of the desired sparsity range using STR and could not obtain any usable results using DST. All results are averaged over 3 seeds.

One of the main intentions behind pruning stability is to eliminate the need for computationally expensive retraining. We start by verifying this assumption using the pruning stable approaches and allowing them to be retrained for further 30 epochs using FT. Figure 12 in the appendix shows the increase in accuracy of the pruned networks after retraining. We observe that LC is often unable to find an actually sparse model throughout training and hence greatly benefits from retraining. The remaining methods however mostly stay true to their claimed pruning stability, but nevertheless sometimes profit from retraining. For the sake of a fair comparison, we will therefore use the accuracy reached after retraining when it exceeds the original one for all pruning stable methods in

[3] In addition, we noticed that each of these method can have some variance in the level of sparsity achieved even for fixed hyperparameters. We thus group and average runs that differ only by .25 percent in the final sparsity obtained.

this section while only referring to the original train time when comparing run times for all methods except LC and of course IMP.

Figure 4 now reports the final test performance and the theoretical speedup of all eight methods. The results show that IMP with SLR and a tuned retrain phase is able to perform on par with the best pruning stable methods considered here. Only CS and GMP clearly exceed the test accuracy achieved by restricted IMP and are about matched by its 'unrestricted' version. Note that CS received significantly more attention during tuning as previously mentioned, which needs to be taken into consideration when viewing these results. Somewhat surprisingly, IMP is also able to get excellent theoretical speedups, performing on par with DPF while being favored with a slightly higher test accuracy. Most importantly, IMP manages to obtain these results without a strong computational overhead: considering the restricted version of IMP as a baseline, GMP, DNW and DPF obtain their pruned models slightly faster and respectively require 0.91, 0.9, and 0.89 times the total run time of IMP. LC (including retrain time), CS, STR and GSM however were slightly slower with respective factors of 1.14, 1.32, 1.17 and 1.01. The full version of IMP required 1.3 times the total computational time of the restricted version.

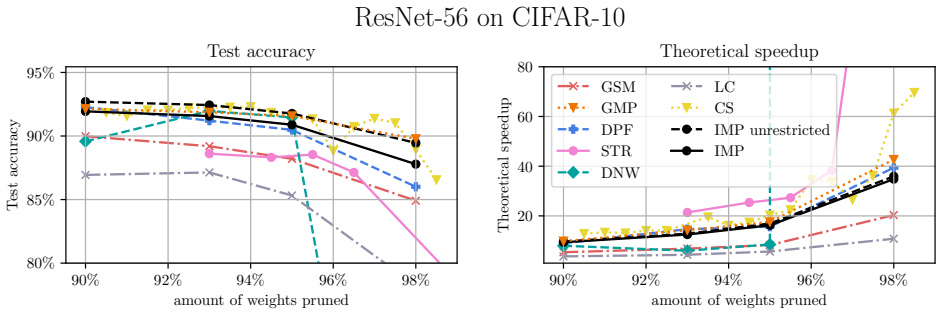

Figure 4: Test accuracy (left) and theoretical speedup (right) of IMP in comparison to different pruning stable methods when training a ResNet-56 network on CIFAR-10. All methods were trained to achieve sparsity levels of 90%, 93%, 95% and 98% with the exception of CS and STR, where additional hyperparameter searches were necessary to obtain the curves shown here.

Finally, we note that there are also some more subtle practical differences between IMP and the pruning stable methods considered here, that are difficult to explore in experiments. We have already noted that some pruning stable methods can be quite difficult to tune compared to the simplicity of IMP. Furthermore, the fact that IMP can be applied to an already trained model, allows one to quickly iterate over hyperparameter configurations, while pruning stable methods on the other hand need to start from scratch every time, resulting in much longer hyperparameter searches. Finally, IMP can in fact derive many different models along the sparsity-vs.-performance trade-off spectrum from just a single run, by simply storing the different models after each prune-retrain cycle. Pruning stable methods on the other hand need would need to fully train a network for every given level of sparsity. In practice, these aspects can make a massive computational difference, reversing the traditional narrative.

4 Discussion

The goal of this work is to demonstrate that IMP can provide state-of-the-art pruning results on par or outperforming more complex or heavily parameterized approaches without or with only little computational overhead. It is not self-evident that imposing an implicit bias during training has any benefits besides not requiring retraining, which is in and of itself neither an argument for nor against a method. While we believe that our results demonstrate this, we acknowledge a clear limitation of the study: our experiments were limited to image recognition tasks on particular network architectures. Extending the results to include, e.g., NLP as well as a wider variety of network architectures would help paint a more complete picture. Regardless, we think that IMP with SLR and a moderate retrain phase should serve as a realistic yet easily realisable baseline for future research.

5 REPRODUCIBILITY

Reproducibility is of utmost importance for any comparative computational study such as this. All experiments were based on the PyTorch framework and use publicly available datasets. The implementation of the ResNet-56 network architecture is based on `https://github.com/JJGO/shrinkbench/blob/master/models/cifar_resnet.py`, the implementation of the WideResNet network architecture is based on `https://github.com/meliketoy/wide-resnet.pytorch`, the implementation of the VGG-16 network architecture is based on `https://github.com/jaeho-lee/layer-adaptive-sparsity/blob/main/tools/models/vgg.py` and the implementation of the Resnet-50 network architecture is taken from `https://pytorch.org/vision/stable/index.html`. Regarding the pruning methods, the code was taken from the respective publications whenever possible. Regarding the different variants of magnitude pruning such as ERK or Uniform+, we closely followed the implementation of Lee et al. (2020) available at `https://github.com/jaeho-lee/layer-adaptive-sparsity/`. For metrics such as the theoretical speedup, we relied on the implementation in the *ShrinkBench*-framework of Blalock et al. (2020), see `https://github.com/JJGO/shrinkbench`. All parameters considered are specified in Section 3 or in the Appendix. We also plan on publicly releasing the code used for our implementations and general setup for the sake of reproducibility and will make it available during any submission of this work.

REFERENCES

Brian R. Bartoldson, Ari S. Morcos, Adrian Barbu, and Gordon Erlebacher. The generalization-stability tradeoff in neural network pruning. June 2019.

Lukas Biewald. Experiment tracking with weights and biases, 2020. URL `https://www.wandb.com/`. Software available from wandb.com.

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? March 2020.

Miguel A Carreira-Perpinán and Yerlan Idelbayev. "learning-compression" algorithms for neural net pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8532–8541, 2018.

Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. July 2019.

Xiaohan Ding, Guiguang Ding, Xiangxin Zhou, Yuchen Guo, Jungong Han, and Ji Liu. Global sparse momentum sgd for pruning very deep neural networks. September 2019.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. *Proceedings of the 37th International Conference on Machine Learning (2020) 471-481*, November 2019.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *ICLR 2019*, March 2018.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. Stabilizing the lottery ticket hypothesis. March 2019.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. June 2015.

Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In Stephen Jose Hanson, Jack D. Cowan, and C. Lee

Giles (eds.), *Advances in Neural Information Processing Systems 5, [NIPS Conference, Denver, Colorado, USA, November 30 - December 3, 1992]*, pp. 164–171. Morgan Kaufmann, 1992. URL http://papers.nips.cc/paper/647-second-order-derivatives-for-network-pruning-optimal-brain-surgeon.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. February 2015.

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. January 2021.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. February 2020.

Duong Hoang Le and Binh-Son Hua. Network pruning that matters: A case study on retraining variants. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Cb54AMqHQFP.

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In David S. Touretzky (ed.), *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*, pp. 598–605. Morgan Kaufmann, 1989. URL http://papers.nips.cc/paper/250-optimal-brain-damage.

Jaeho Lee, Sejun Park, Sangwoo Mo, Sungsoo Ahn, and Jinwoo Shin. Layer-adaptive sparsity for the magnitude-based pruning. October 2020.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. August 2016.

Tao Lin, Sebastian U. Stich, Luis Barba, Daniil Dmitriev, and Martin Jaggi. Dynamic model pruning with feedback. June 2020.

Junjie Liu, Zhe Xu, Runbin Shi, Ray C. C. Cheung, and Hayden K. H. So. Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. May 2020.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. November 2017.

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization. December 2017.

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pp. 5058–5066, 2017.

Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H. Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications*, 9(1), June 2018. doi: 10.1038/s41467-018-04316-3.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-based capacity control in neural networks. In *Conference on Learning Theory*, pp. 1376–1401, 2015.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. December 2019.

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. *arXiv preprint arXiv:2003.02389*, 2020.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Pedro Savarese, Hugo Silva, and Michael Maire. Winning the lottery with continuous sparsification. December 2019.

Jonathan W. Siegel, Jianhong Chen, and Jinchao Xu. Training sparse neural networks using compressed sensing. August 2020.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. September 2014.

Leslie N. Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. August 2017.

Hidenori Tanaka, Daniel Kunin, Daniel L. K. Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in Neural Information Processing Systems 2020*, June 2020.

Mitchell Wortsman, Ali Farhadi, and Mohammad Rastegari. Discovering neural wirings. June 2019.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. May 2016.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. November 2016.

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. October 2017.

A Technical details and Training Settings

A.1 Technical details

Let us define *pruning stability* as follows.

Definition A.1 (Bartoldson et al. (2019))**.** Let t_{pre} and t_{post} be the test accuracy before pruning and after pruning the trained model, respectively. We define the *pruning stability* of a method as

$$\Delta_{stability} \coloneqq 1 - \frac{t_{pre} - t_{post}}{t_{pre}} \in [0, 1].$$

Pruning stable methods are sparsification algorithms that learn a sparse solution throughout training such that $\Delta_{stability} \approx 1$. For example, methods that keep a pruned copy of the parameters which is used for the entire training procedure (Wortsman et al., 2019, e.g.), will have $\Delta_{stability} = 1$, since the 'hard' pruning step consists of an application of the present mask. Methods that actively drive certain parameter groups towards zero more rapidly (such as Carreira-Perpinán & Idelbayev, 2018; Ding et al., 2019) will have a pruning stability close to 1, since the orthogonal projection of (magnitude) pruning at the end of training will perturb the parameters only slightly.

Crucial to our analysis will be the trade-offs between the model sparsity and both the *final test accuracy* and the *theoretical speedup* (Blalock et al., 2020) induced by the sparsity. The latter is a metric measuring the ratio in FLOPs needed for inference comparing the dense and sparse model. More precisely, let F_d be the number of FLOPs the dense model needs for inference, and let similarly be F_s the same number for the pruned model, given some sparsity s.[4] The theoretical speedup is defined as F_d/F_s and depends solely on the position of the zero weights within the network and layers, not on the numerical values of non-zero parameters.

A.2 Training settings and hyperparameters

Table 2: Overview of sparsification methods. CS, STR and DST control the sparsity implicitly via additional hyperparameters. IMP is the only method that is pruning instable, i.e., it loses its performance right after the ultimate pruning. Further, IMP is the only method that is sparsity agnostic throughout the regular training; the sparsity does not play a role while training. All other methods require training an entire model when changing the goal sparsity. The computational overhead refers to the per-iteration overhead during regular training.

	Sparsity specifiable	Pruning stable	Sparsity agnostic training	Overhead during Training
IMP	✓	✗	✓	N/A
GMP	✓	✓	✗	$\mathcal{O}(n) + \mathcal{O}(n \log n)$ every time the mask is updated
GSM	✓	✓	✗	$\mathcal{O}(n + n \cdot \min(k, n-k))$
LC	✓	✓	✗	$\mathcal{O}(n \cdot \min(k, n-k))$
DPF	✓	✓	✗	$\mathcal{O}(n) + \mathcal{O}(n \log n)$ every 16 iterations
DNW	✓	✓	✗	$\mathcal{O}(n \cdot \min(k, n-k))$
CS	✗	✓	✗	$\mathcal{O}(n)$ + add. backprop
STR	✗	✓	✗	$\mathcal{O}(n)$ + add. backprop
DST	✗	✓	✗	$\mathcal{O}(n)$ + add. backprop

We list the hyperparameter grids used for each pruning stable method taking part in the comparative study. For each pruning stable method we tune weight decay over 1e-4, 5e-4, 1e-3 and 5e-3 for CIFAR-10 and keep momentum fixed at 0.9. Since momentum plays a crucial part in the justification of GSM, we tune it over 0.9, 0.99 and 0.995 for CIFAR-10. Otherwise, we used the following grids.

GMP
Equally distributed pruning steps: $\{20, 100\}$.

[4] To compute the number of FLOPs, we sample a single batch from the test set. The code to compute the theoretical speedup has been adapted from the repository of the *ShrinkBench* framework (Blalock et al., 2020).

GSM
Momentum: $\{0.9, 0.95, 0.99\}$.

LC
We only tune the weight decay and, similar to the recommendation by Carreira-Perpinán & Idelbayev (2018), increase it over time as $\lambda_0 \cdot 1.1^j$, where j is increased over time and λ_0 is the initial weight decay. For the retraining phase we deactivate weight decay.

DPF
As for GMP, we tune the number of pruning steps, i.e., $\{20, 100\}$, and the weight decay.

DNW
We only tune the weight decay, since there are no additional hyperparameters.

CS
As recommended by Savarese et al. (2019), we fix the temperature β at 300. Otherwise, we perform the following grid search. We set the mask initialization $s_0 \in \{-0.3, -0.25, -0.2, -0.1, -0.05, 0, 0.05, 0.1, 0.2, 0.25, 0.3\}$ and the ℓ_1 penalty λ to $\{$1e-8, 1e-7$\}$.

STR
We tune the initial threshold value $s_{\text{init}} \in \{-100, -50, -5, -2, -1, 0, 5, 50, 100\}$. In an extended grid search, we also used weight decays in $\{$5e-05, 1e-4$\}$ and varied $s_{\text{init}} \in \{-40, -30, -20, -10\}$.

DST
We tune the sparsity-controlling regularization parameter $\alpha \in \{$5e-6, 1e-5, 5e-5, 1e-4, 5e-4$\}$. In an extended grid search, we used weight decays in $\{$0, 1e-4$\}$ and tuned α over $\{$1e-7, 5e-7, 1e-6$\}$.

B ADDITIONAL PLOTS

B.1 THE COMPUTATIONAL COST OF IMP

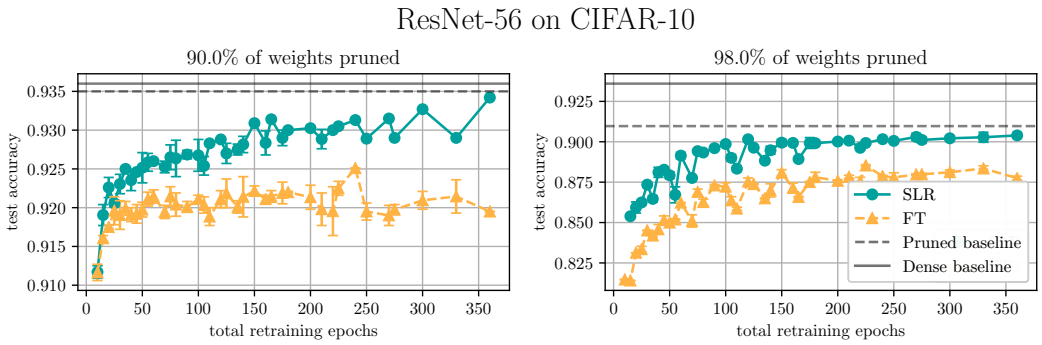

Figure 5: Test accuracy achieved by IMP in relation to the total number of epochs required for retraining using either FT or SLR. The pruned baselines were established by Renda et al. (2020).

B.2 THE IMPORTANCE OF THE SPARSITY DISTRIBUTION

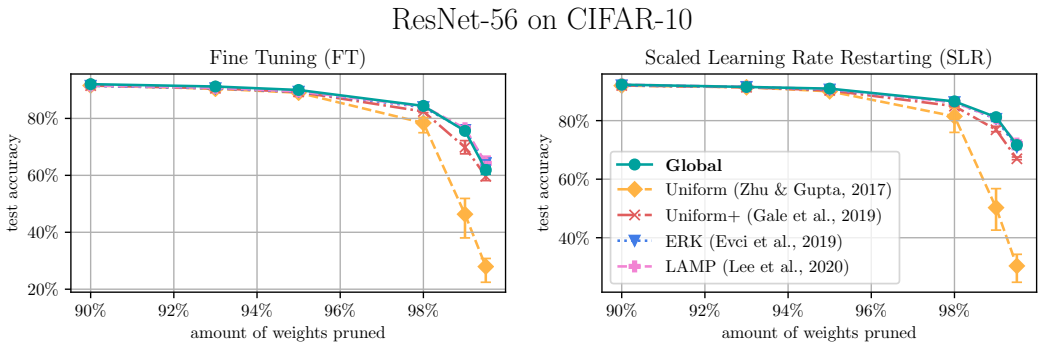

Figure 6: Performance vs. Sparsity tradeoffs for ResNet-56 on CIFAR-10 for IMP in the One Shot setting for FT (left) and SLR (right) as retraining methods. The plot includes max-min confidence intervals.

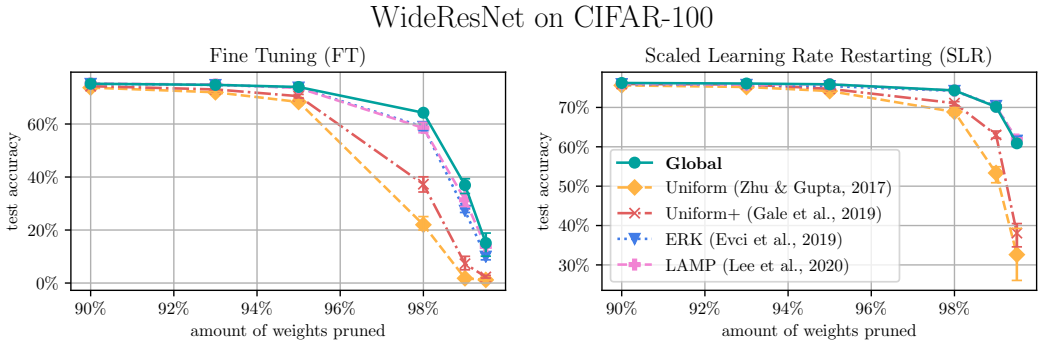

Figure 7: Performance vs. Sparsity tradeoffs for WideResNet on CIFAR-100 for IMP in the One Shot setting for FT (left) and SLR (right) as retraining methods. The plot includes max-min confidence intervals.

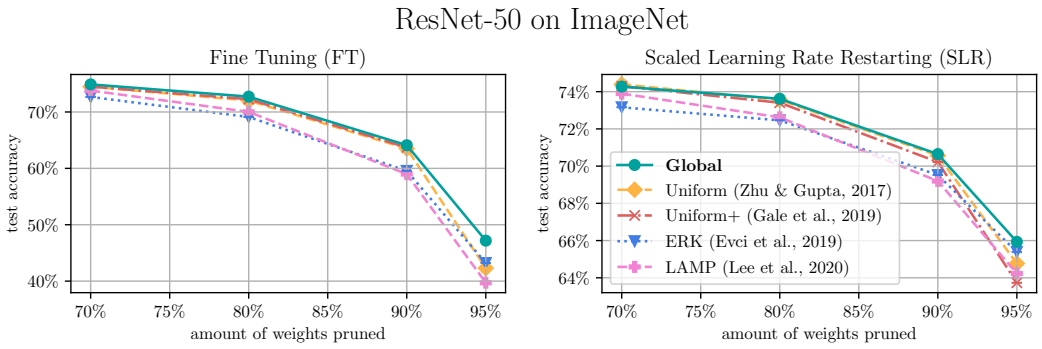

Figure 8: Performance vs. Sparsity tradeoffs for ResNet-50 on ImageNet for IMP in the One Shot setting for FT (left) and SLR (right) as retraining methods. The plot includes max-min confidence intervals.

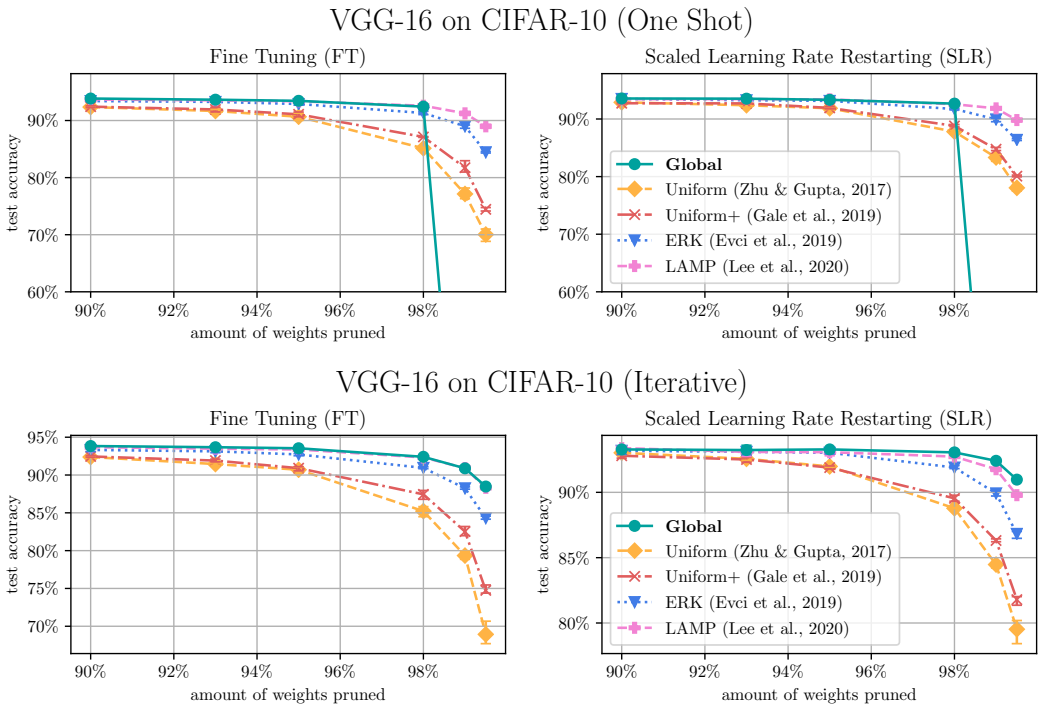

Figure 9: Performance-vs.-Sparsity tradeoffs for VGG-16 on CIFAR-10 for IMP in the One Shot (above) and Iterative (below) setting for FT (left) and SLR (right) as retraining methods.

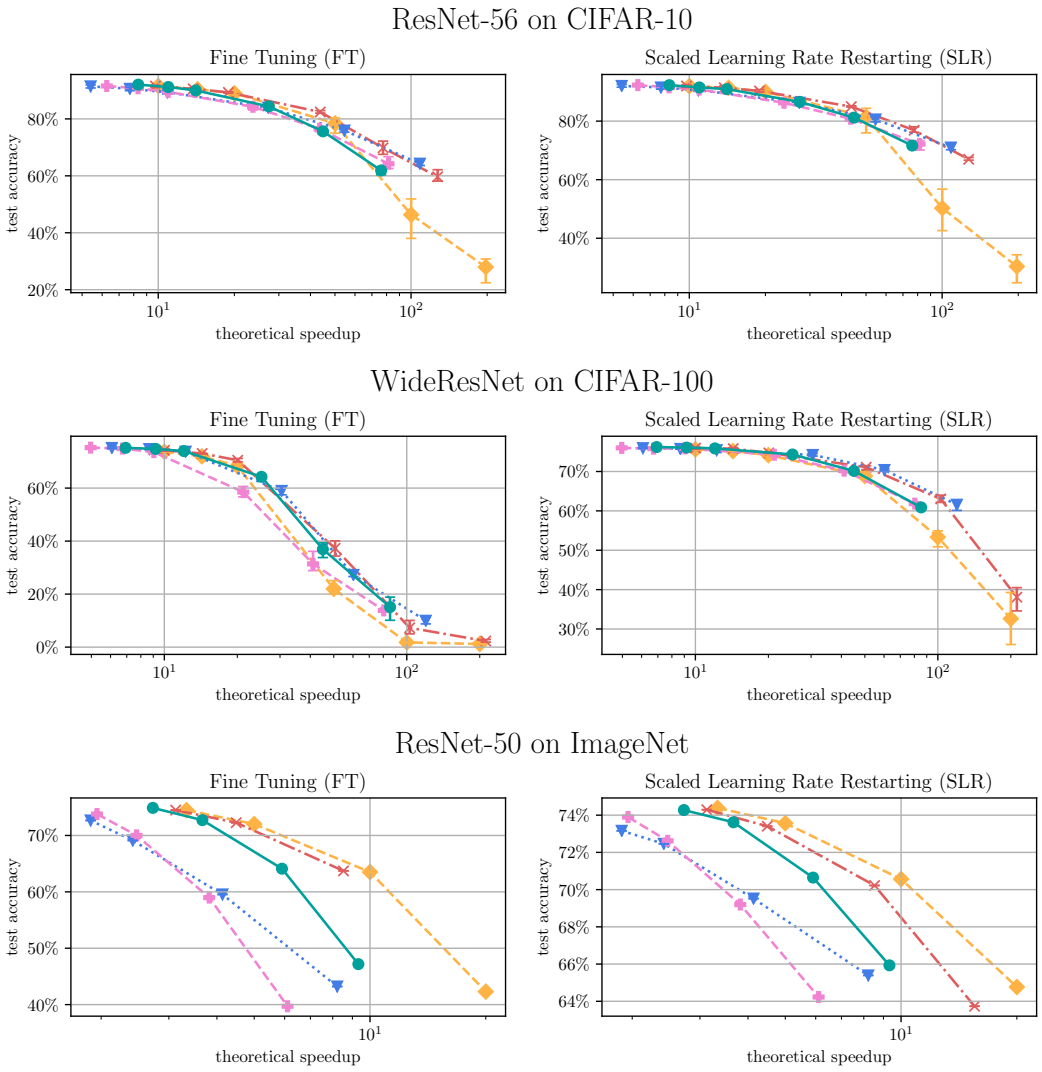

Figure 10: Performance-vs.-theoretical speedup trade-offs for ResNet-56, WideResNet and ResNet-50 on the respective datasets. All plots depict the one shot setting.

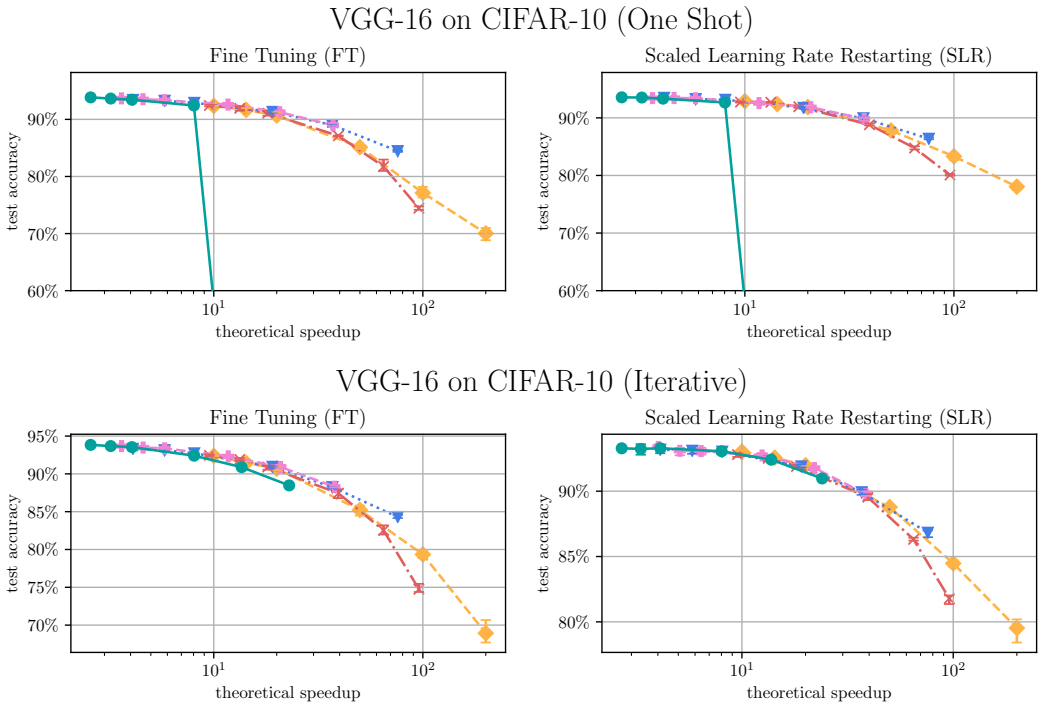

Figure 11: Performance-vs.-theoretical speedup trade-offs for VGG-16 on CIFAR-10 in both the one shot as well as iterative setting.

B.3 Comparing IMP to Pruning Stable Approaches

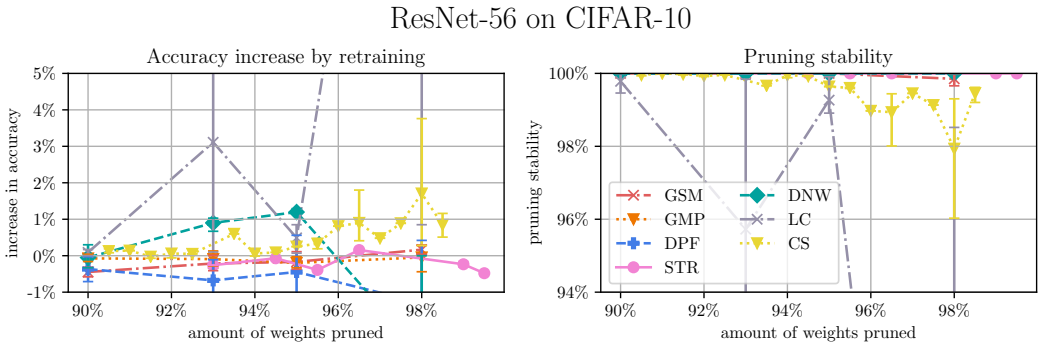

Figure 12: Increase in accuracy after retraining (left) as well as pruning stability (right) for ResNet-56 trained on CIFAR-10 using different pruning-stable-methods. Each method was retrained for 30 epochs.

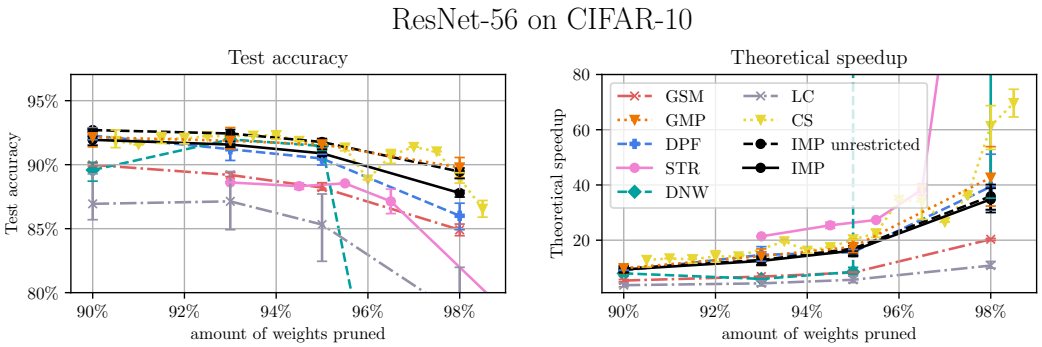

Figure 13: Test accuracy (left) and theoretical speedup (right) of IMP in comparison to different pruning stable methods when training a ResNet-56 network on CIFAR-10. All methods were trained to achieve sparsity levels of 90%, 93%, 95% and 98% with the exception of CS and STR, where additional hyperparameter searches were necessary to obtain the curves shown here. Vertical lines indicate min-max confidence intervals.