COMPARING FINE-TUNING AND REWINDING IN NEURAL NETWORK PRUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural network pruning is a popular technique for reducing inference costs by removing connections, neurons, or other structure from the network. In the literature, pruning typically follows a standard procedure: train the network, remove unwanted structure (*pruning*), and train the resulting network further to recover accuracy (*fine-tuning*).

In this paper, we explore an alternative to fine-tuning: *rewinding*. Rather than continuing to train the resultant pruned network (fine-tuning), *rewind* the remaining weights to their values from earlier in training, and re-train the resultant network for the remainder of the original training process. We find that this procedure, which repurposes the strategy for finding lottery tickets presented by Frankle et al. (2019), makes it possible to prune networks further than is possible with fine-tuning for a given target accuracy, provided that the weights are rewound to a suitable point in training. We also find that there is a wide range of suitable rewind points that achieve higher accuracy than fine-tuning across all tested networks. Based on these results, we argue that practitioners should explore rewinding as an alternative to fine-tuning for neural network pruning.

1 INTRODUCTION

Pruning (Le Cun et al., 1990; Hassibi et al., 1993; Han et al., 2015; Li et al., 2017; Liu et al., 2019) is a popular set of techniques that reduce the memory or computational demands of a neural network by removing weights, filters, neurons, or other structures while maintaining the network's accuracy. In practice, pruning can reduce the parameter counts of large-scale networks for vision and NLP tasks by an order of magnitude while maintaining the accuracy of the original network (Han et al., 2016). Pruning allows for accelerating neural network inference: smaller neural networks can fit on smaller devices, and in some situations can use fewer FLOPS, reducing energy/power consumption and inference latency (Han, 2017).

Many state-of-the-art pruning methods follow a standard *fine-tuning* framework: (1) train the network, (2) prune a given percentage of network structures according to a heuristic, and (3) *fine-tune* the resulting network with additional training for several epochs to recover its accuracy. Instantiated with a simple heuristic that prunes weights with the lowest magnitude, this framework can reduce the size of common vision networks by an order of magnitude with no loss in accuracy (He et al., 2018; Liu et al., 2019; Gale et al., 2019).

Other techniques use variations of this framework, such as *iterative pruning* (Han et al., 2015), which repeats steps 2 and 3 iteratively, alternating between pruning and fine-tuning the network until reaching a desired sparsity or accuracy level. This is in contrast with *one-shot pruning*, which prunes to its target sparsity in a single step.

In practice, finding smaller or more accurate pruned networks requires more epochs of fine-tuning, thereby increasing pruning's computational cost. However, this investment may be justified if the network will be used extensively for inference, or needs to reach a particular size to run on certain platforms (e.g. mobile devices). Therefore, the process of pruning a neural network necessitates a careful tradeoff between three criteria:

ACCURACY.	The accuracy of the resultant pruned network.
INFERENCE COST.	The cost of performing inference with the pruned network.
SEARCH COST.	The cost of finding the pruned network (i.e. the amount of training required).

Many pruning techniques focus exclusively on finding the sparsest network that matches the accuracy of the original network (Han et al., 2015; Molchanov et al., 2017; He et al., 2018). However, other important regions exist in the tradeoff space – for instance, trading off ACCURACY for even more sparsity (decreasing INFERENCE COST), or constraining the cost of finding the pruned network (SEARCH COST) to be below some threshold.

In this paper, we demonstrate a technique, *rewinding*, that can find smaller networks (lower INFERENCE COST) than fine-tuning for any given target ACCURACY. Further, given the same target sparsity (INFERENCE COST) and total epochs of training (SEARCH COST), rewinding is able to produce networks that are at least as accurate as those produced by fine-tuning, provided a suitable point to rewind to.

Rewinding. The lottery ticket hypothesis (Frankle et al., 2019) demonstrates that typical neural networks contain far smaller subnetworks that emerge early in training (i.e., in the first 10% of the way through training), and can train in isolation to the same accuracy as the full network. To find these subnetworks, Frankle et al. propose a variation on the fine-tuning framework: (1) train the network to convergence at final epoch T, (2) prune structures according to a heuristic, (3) *rewind* the remaining weights to their values from an intermediate epoch t in training, and (4) re-train the pruned and rewound subnetwork for the final $T - t$ epochs of training, using the same training parameters and learning rate schedule as was used during the original training. Rewinding (which Frankle et al. use to find lottery tickets) offers an alternative framework to fine-tuning for neural network pruning.

Our contribution. In this paper, we demonstrate the value of rewinding as a general pruning framework and compare rewinding and fine-tuning on CIFAR-10 and ImageNet networks. We study rewinding to epochs all throughout training, and show that even rewinding to later in training (which Frankle et al. do not study) can result in more accurate networks than fine-tuning.

Rewinding expands the frontier of the tradeoff space between the pruning criteria: the standard network-agnostic rewinding framework we use is able to prune a ResNet-50 by 79% with no loss in accuracy, matching more complex and computationally expensive state-of-the-art results (He et al., 2018; Gale et al., 2019). Further, rewinding is able to find sparser neural networks for any given target accuracy than fine-tuning can find.

Rewinding involves picking a suitable rewind point. We show that across all tested networks, rewinding achieves higher accuracy than fine-tuning across a wide range of rewind points. For a fixed re-training budget, desired sparsity, and given a rewind point between 30% to 75% of the way through training, rewinding finds more accurate pruned networks than fine-tuning.

Based on these results, we argue that practitioners should explore rewinding as an alternative to fine-tuning for neural network pruning.

2 METHODOLOGY

We select a variety of standard vision networks, and train and prune them with standard hyperparameters. We collect 3 independent trials of all experiments.

2.1 EXPERIMENTS

Rewinding experiments. Rewinding introduces one hyperparameter: the epoch t that the network is rewound to. For example, consider a network that is normally trained for T epochs. We train the network for the original T epochs, prune, rewind the weights to their values at epoch t, and re-train the resultant network for the remaining $T - t$ epochs of training. When rewinding weights to a specific epoch, we also rewind the learning rate schedule to the same epoch. The rewinding algorithm is presented in Appendix A, Algorithm 1.

Fine-tuning experiments. There are several hyperparameters that must be chosen when pruning with fine-tuning, namely the number of epochs t to fine-tune for, and the learning rate schedule during that re-training. In our experiments, we explore many values of t, which controls the cost of obtaining the final, pruned network (SEARCH COST). We follow the standard practice of fine-tuning using the final learning rate used during the original training phase (Liu et al., 2019). The fine-tuning algorithm is presented in Appendix A, Algorithm 2.

Reinitialization experiments. We include a baseline of reinitializing and re-training the pruned network from scratch, for the same total number of epochs as the rewinding experiments and the fine-tuning experiments. This reinitialization baseline is from Liu et al. (2019). In the iterative case, it is not within our computation budget to reinitialize and re-train every distinct discovered network structure at every sparsity level: reinitialization re-trains a masked subnetwork, and iterative pruning generates a different mask for each different rewind point and sparsity level (as opposed to the singular mask per sparsity level generated by one-shot pruning). Instead, we use two approaches, and report the best performing of the two. First, we use the sparsity discovered by the best-performing network from the previous iterative pruning iteration, under the assumption that reinitializing from the best-performing network structure will result in the best accuracy. Second, we simply use a one-shot pruning and reinitialization from the original network. The reinitialization algorithm (following the choice of using one-shot pruning) is presented in Appendix A, Algorithm 3.

2.2 PRUNING FRAMEWORK

Pruning heuristic. We study a sparse pruning heuristic in which we remove individual weights with the lowest magnitudes from the network, following Han et al. (2016); Zhu & Gupta (2017); Gale et al. (2019); Liu et al. (2019); and Frankle & Carbin (2019). We prune all weights in the network *globally* (Lee et al., 2019; Frankle & Carbin, 2019), removing weights with the lowest magnitudes without consideration for the layer in which the weight resides.

Iterative pruning. Pruning frameworks can be applied iteratively, repeatedly pruning and fine-tuning or rewinding to gradually compress the network. In our iterative pruning experiments, we follow the strategy from Han et al. (2016) and Frankle & Carbin (2019) to remove 20% of weights per iteration, repeating this process as necessary to reach the desired level of sparsity.

2.3 NETWORKS AND HYPERPARAMETERS

Training hyperparameters. We study both CIFAR-10 (Krizhevsky & Hinton, 2009) and ImageNet (Russakovsky et al., 2015) networks. All networks are trained with SGD with Nesterov momentum with $\beta = 0.9$. We use the standard ResNet training schedule for CIFAR-10, which trains for 182 epochs with a batch size of 128, using an initial learning rate of 0.1, decaying by a factor of 10 at epochs 91 and 136 (He et al., 2016). On ImageNet, we use the schedule from Goyal et al. (2017) with a batch size of 1024, training for 90 epochs with an initial learning rate of 0 linearly scaling up to 0.4 over the first 5 epochs, then decaying by a factor of 10 at epochs 30, 60, and 80.

Networks. We use a standard implementation of a ResNet-20 (He et al., 2016) provided alongside Tensorflow (Abadi et al., 2016), available online[1]. With the CIFAR-10 training schedule, this implementation trains the ResNet-20 to $91.58\% \pm 0.20\%$ accuracy.

We use a slight modification of the VGG-16 network as described by Simonyan & Zisserman (2014). Our VGG-16 includes Batch Normalization (Ioffe & Szegedy, 2015) after each convolution and before each activation. Also, rather than multiple fully connected layers at the end, we use a single fully connected layer. This implementation trains to $93.24\% \pm 0.19\%$ accuracy with our CIFAR-10 training schedule.

We use a standard implementation of a ResNet-50 provided alongside Tensorflow, available online[2]. With the ImageNet training schedule, this implementation trains the ResNet-50 to $76.13\% \pm 0.13\%$ accuracy, in line with the baseline accuracy reported in the state-of-the-art pruning approaches (He et al., 2018; Gale et al., 2019).

[1] `https://github.com/tensorflow/models/tree/v1.13.0/official/resnet`

[2] `https://github.com/tensorflow/tpu/tree/98497e0b/models/official/resnet`

Network	Pruning Method	Finetune Sparsity	Rewinding Sparsity	Ratio
ResNet-20	One-shot	36.00%	**67.23%**	1.95×
ResNet-20	Iterative	67.23%	**83.22%**	1.95×
VGG-16	One-shot	94.50%	**96.48%**	1.56×
VGG-16	Iterative	94.50%	**98.85%**	4.77×
ResNet-50	One-shot	59.04%	**73.79%**	1.56×
ResNet-50	Iterative	48.80%	**79.03%**	2.44×

Table 1: Sparsest networks with no accuracy loss from the original network (max across all trials). The ratio column is the ratio between the *density* (inverse of sparsity) of the fine-tuning network and the density of the rewinding network.

2.4 Limitations of experiments

This paper studies only vision networks with a single pruning technique (unstructured magnitude pruning), comparing against a single variant of the fine-tuning framework. We do not consider any variations on this fine-tuning framework, such as techniques that gradually prune throughout training. We also do not consider structured pruning methods that produce structurally dense networks more easily amenable to acceleration.

Our rewinding framework also precludes some potentially useful techniques, such as disentangling the rewinding epoch from the number of re-training epochs, e.g. training for shorter or longer than the original training schedule or using a different learning rate. We believe that other tradeoffs between the pruning criteria can be made by exploring this space and tuning the re-training schedule, but do not explore these modified schedules in this paper in order to provide a fair comparison against fine-tuning.

3 Sparse Networks

3.1 Matching the Original Accuracy

Rewinding can find sparser networks (Inference Cost) that match the accuracy of the original network (Accuracy) than fine-tuning can. This metric follows standard approaches to pruning which allow for large budgets to produce the most accurate and efficient network (Han et al., 2015): pruning is often motivated as producing networks which are just as accurate as the original, unpruned network, allowing for an unbounded Search Cost.

Methodology. To find the sparsest network with no loss in accuracy achievable by a given technique, we ran a parameter sweep across 10 evenly distributed re-training epochs between 0 and the original training time T of the network (90 for ImageNet networks and 182 for CIFAR-10 networks) for rewinding, and between 0 and $2T$ for fine-tuning and reinitialization (allotting fine-tuning up to $2\times$ the re-training time as rewinding if it needs longer to train). We report the sparsity level at which at least one of the three trials for a given technique and number of re-training epochs matched the accuracy of the original unpruned network.

Results. Table 1 shows the sparsest network achievable with no loss in accuracy (i.e. a resultant accuracy equal to that of the original network). On all networks considered, rewinding finds a sparser network without loss in accuracy than fine-tuning finds. Beyond just outperforming fine-tuning, these results are competitive with state-of-the-art pruning approaches tailored specifically for ResNet-50: using a bespoke, highly tuned framework which prunes by specific layerwise rate at specific points throughout the regular training schedule, Gale et al. (2019) achieve 80% sparsity with a 0.17% accuracy drop on a ResNet-50. Using reinforcement learning to find layerwise pruning rates (rather than the simple global magnitude heuristic presented in this paper), He et al. (2018) achieve 80% sparsity with a 0.02% drop in accuracy on a ResNet-50. Our approach, using standard pruning hyperparameters (Han et al., 2015) and with just the addition of rewinding instead of fine-tuning, achieves 79.03% sparsity (7 iterations of pruning by 20%) with a 0.06% increase in accuracy.

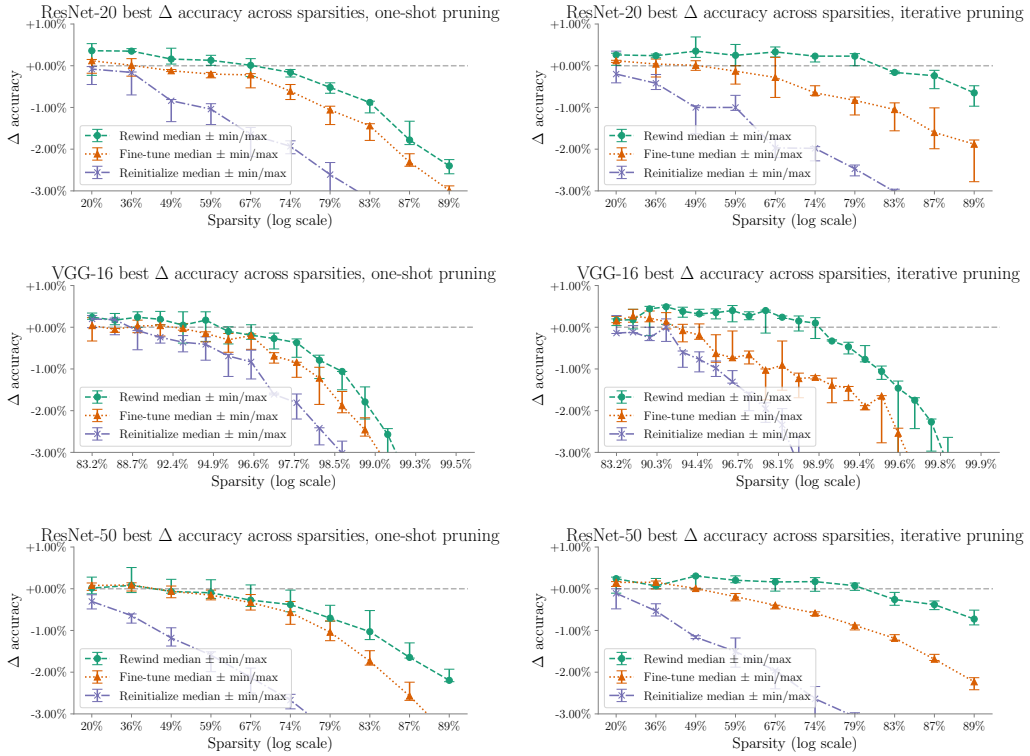


Figure 1: The most accurate networks achievable by different pruning and re-training methods across different target sparsity levels.

3.2 Pruning with an Accuracy Target

Beyond just producing sparser networks that match the original accuracy, rewinding can find sparser networks that match any accuracy threshold.

Methodology. Using the same experimental methodology as Section 3.1, we plot the most accurate achievable networks across a wide range of target sparsities in Figure 1. Accuracies reported in this section are the median accuracy of the best performing set of trials across all re-training epochs in the parameter sweep (e.g. the median of the accuracies for all 3 trials of re-training for 80 epochs, rather than the best 3 trials across all choices of re-training epochs).

As a baseline, we also consider re-initializing the discovered pruned network and re-training it by extending the original training schedule to the same total number of training epochs as fine-tuning trains for. Liu et al. (2019) found that for many pruning techniques, pruning and fine-tuning results in the same or worse performance as simply training the pruned network from scratch for an equivalent number of epochs. To address these concerns, we include comparisons against random reinitializations of networks with the discovered pruned structure, trained for the original T training epochs plus the extra t epochs that networks were re-trained for. The methodology for these experiments is detailed in Section 2.1.

Results. Rewinding always achieves either equivalent or higher accuracy than fine-tuning, with a particularly distinct advantage at higher sparsities. At lower sparsities, the techniques perform equivalently well, recovering the original accuracy of the unpruned network. At higher sparsities, the techniques result in different accuracies: rewinding consistently achieves higher accuracy than fine-tuning, which in turn outperforms reinitialization. These results are consistent across all tested networks and pruning methods.

4 PICKING THE REWIND EPOCH

To use rewinding when pruning a neural network, one must first select a point in training to rewind the network's weights to. While rewinding achieves higher accuracy than fine-tuning for many choices of rewind points, it does not do so for all.

Rewinding has regions of low performance. Rewinding to the very beginning of training results in significantly lower accuracy than rewinding to a small amount into training (Frankle et al., 2019). Similarly, rewinding to a point too late through training means that the full accuracy benefits of rewinding will not be realized: our rewinding framework does not allow adding on additional training time after the fact.

Further, the choice of rewind point is permanent: selecting a different rewind point to achieve better accuracy requires re-training from the new rewind point. On the other hand, one can continue fine-tuning to achieve its best available accuracy: it is easy to continue fine-tuning indefinitely until either a desired accuracy is met or learning plateaus. A given rewind point imposes a floor on re-training time in order to get the accuracy benefits, along with a ceiling on the accuracy achievable from that rewind point.

Methodology. To quantitatively determine where to rewind to, we consider two different metrics:

- *Safety*: a measure of the accuracy gap between rewinding to a given point and fine-tuning with an equivalent re-training budget. More formally, the safety of a rewind point t and a sparsity s is defined as the difference between the accuracy achievable by pruning to s sparsity, rewinding to epoch t in training, and re-training for exactly $T - t$ epochs (where T is the original training time of the network), and the best accuracy achievable by fine-tuning for at most $T - t$ epochs.
- *Dominance*: a measure of the accuracy gap between rewinding to a given point and fine-tuning given an unlimited budget. Formally, the dominance of a rewind point t and a sparsity s is defined as the accuracy achievable by pruning to s sparsity, rewinding to epoch t and re-training for $T - t$ epochs, and the maximum accuracy achievable by pruning to s sparsity and fine-tuning for any amount of time (up to $2\times$ the original training time).

Results. Figure 2 and Figure 3 show the safety and dominance respectively of rewinding across networks, pruning methods, and sparsity levels. In these plots, the y axis shows the rewind point as a percentage of the original training time of the network – i.e., 0% is rewinding to the beginning of training, and 100% is rewinding to the end. Red points mean that rewinding achieves higher accuracy for that sparsity and re-training budget; blue means that fine-tuning achieves higher accuracy.

An alternative presentation of these data, instead showing accuracy curves in a form similar to Figure 1, can be found in Appendix F.

Analysis. At medium to high sparsities, rewinding to anywhere between 30% to 75% of the way through training consistently outperforms fine-tuning for an equivalent number of re-training epochs (Figure 2). The cross-network dominant region extends from 30% to 60% of the way through training (Figure 3): essentially, rewinding to anywhere in that region and re-training results in higher accuracy than any amount of fine-tuning. Given a rewind point, higher accuracy can often (though not always) be attained by searching for better-performing rewind points closer to the beginning of training (i.e. moving down on Figure 3); re-training time can be reduced, at the cost of accuracy, by rewinding closer to the end of training and re-training for fewer epochs (i.e. moving up on Figure 3).

These regions vary across networks and datasets: for instance, ResNet-50 has a much broader safe and dominant region than VGG-16. The regions also vary across sparsity levels: rewinding performs better relative to fine-tuning on sparser networks, a result also seen in Section 3.2, Figure 1.

Edge cases. The top-most part of the plots of safety in Figure 2 have no strong tendency towards either fine-tuning or rewinding. Fine-tuning and rewinding have approximately the same behavior in the limit of few to no re-training epochs, since (especially in the last leg of the learning rate schedule, when weights are near their final values) they are trained from approximately the same starting point, using approximately the same learning rate.

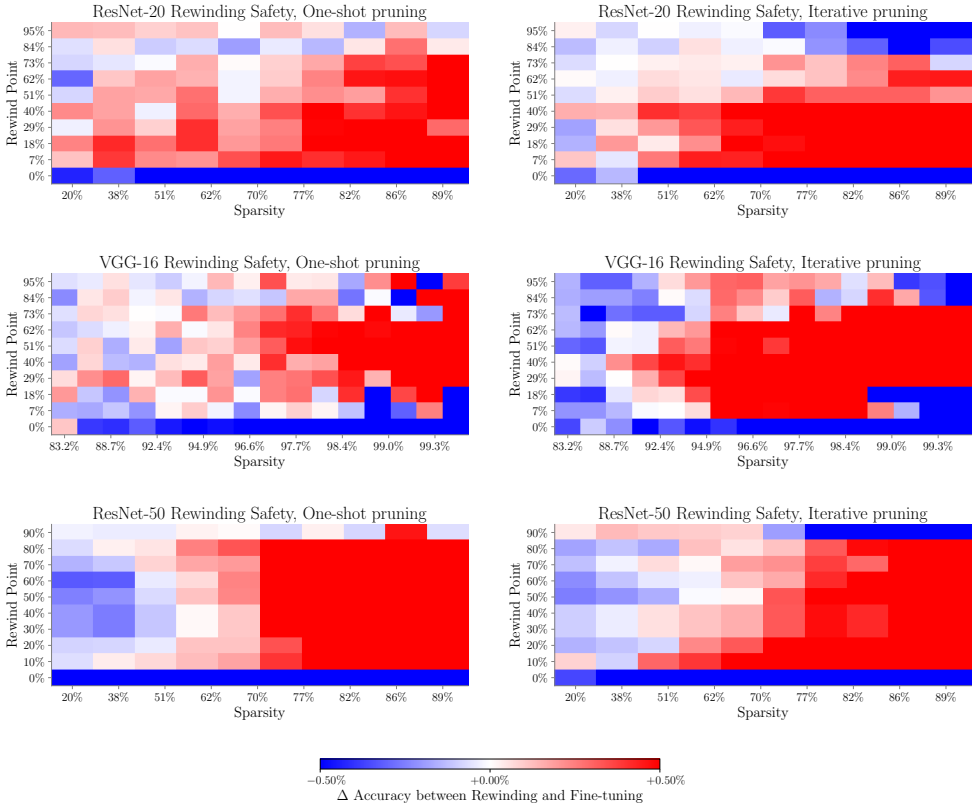


Figure 2: Safety across networks and pruning methods. Red points denote rewinding achieving higher accuracy than fine-tuning for any amount up through the same re-training budget; blue points denote the opposite.

The top region of Figure 3 is sharply blue, indicating that rewinding to the end of training and not re-training achieves significantly lower accuracy than fine-tuning reaches. Magnitude pruning of neural networks damages the function that the neural network learned. Without enough time spent re-training, the network is not capable recovering its original accuracy.

The bottom-most parts of all plots are also sharply blue, which means that rewinding loses its advantage as the weights are rewound to the very beginning of training. This behavior is predicted by Frankle et al. (2019), which demonstrates that networks rewound to the very beginning of training do not typically reach the same accuracy as the unpruned network, whereas networks rewound to a small amount of the way through training do reach that accuracy.

Dense networks. Fine-tuning typically outperforms rewinding in denser networks, as shown by the left side of the plots in Figure 2 and Figures 3 tending towards blue. This is likely due to these networks being trivially prunable, as discussed in Section 3.2. In these situations, fine-tuning is not just a method of recovering the original accuracy of the network: it also behaves like an additional phase of training. In essence, for dense networks fine-tuning can act as a crutch for a non-optimal training schedule, and thereby achieve higher accuracy than rewinding. This advantage falls away at higher sparsities: for deeply pruned networks, there is always a wide range of rewind points such that rewinding achieves higher accuracy than fine-tuning.

Necessary re-training time. These results show a floor on the amount of re-training time for rewinding to be a worthwhile technique: rewinding requires a re-training budget of at least 25% of the original training time to consistently achieve higher accuracy than an equivalent amount of fine-tuning. In situations where the re-training budget is strictly limited to less than 25% of the original training time, fine-tuning outperforms rewinding. However, standard approaches currently

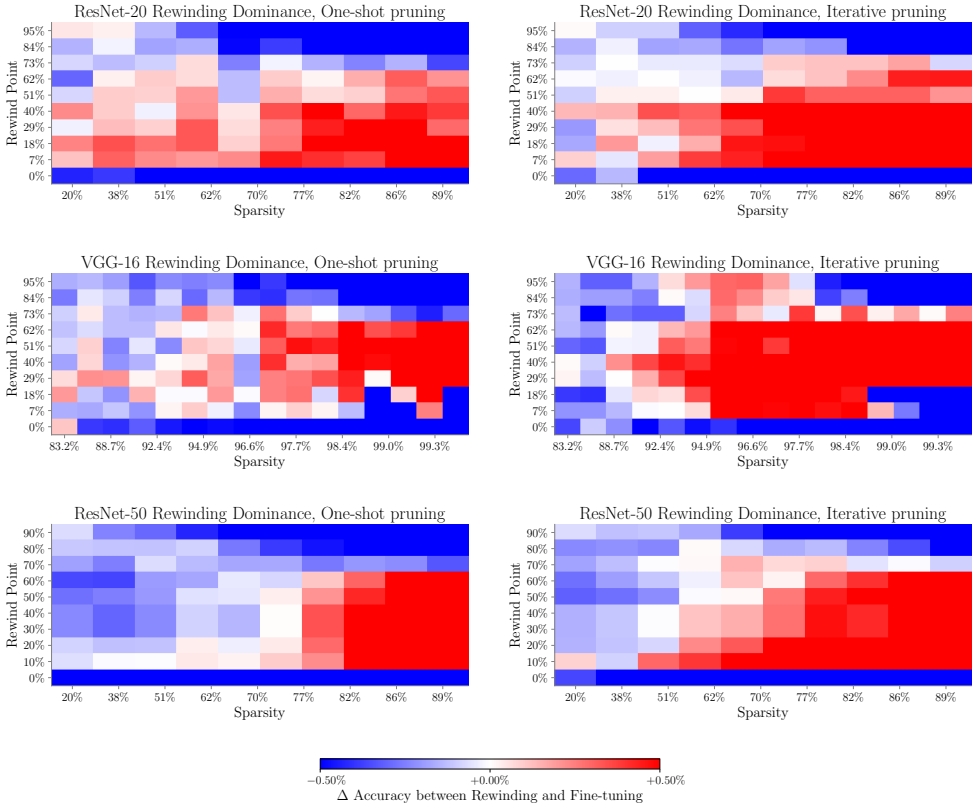


Figure 3: Dominance across networks and pruning methods. Red points denote rewinding achieving higher accuracy than fine-tuning for any amount; blue points denote the opposite.

fine-tune for at least a quarter of the original training time (Li et al., 2017; Liu et al., 2019), and often much longer (Han et al., 2015). Further, 25% of re-training time is often not enough for fine-tuning either: as can be seen by the extra accuracy accumulated by fine-tuning as more re-training time is added (i.e. by the difference between the safety and dominance plots), fine-tuning's accuracy does not always plateau with 25% re-training time. More visualizations can be found in Appendix E.

5 Conclusion

Rewinding expands the frontier in the tradeoff space between the three pruning criteria, Accuracy, Inference Cost, and Search Cost. For sparse magnitude pruning on a variety of vision networks, rewinding finds sparser networks than fine-tuning can for any given accuracy threshold. Using this technique, a ResNet-50 can be pruned by 79% with no loss in test accuracy, comparable with state-of-the-art techniques but without using any ResNet-50-specific hyperparameters. Using rewinding requires selecting a point to rewind to ahead-of-time, but we find that there is a wide range of suitable rewinding points that work across all tested networks, at moderate to high sparsities. Rewinding to anywhere between 30% to 75% of the original training time is a *safe* and efficient use of re-training time, achieving higher accuracy than an equivalent amount of fine-tuning. Rewinding to anywhere between 30% to 60% is *dominant*, outperforming any amount of fine-tuning.

Based on these results, we argue that practitioners should explore rewinding as an alternative to fine-tuning for neural network pruning, in order to find sparse and accurate neural networks.

REFERENCES

Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283, 2016. URL https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJl-b3RcF7.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. The lottery ticket hypothesis at scale. *CoRR*, abs/1903.01611, 2019. URL http://arxiv.org/abs/1903.01611.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *CoRR*, abs/1902.09574, 2019. URL http://arxiv.org/abs/1902.09574.

Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017. URL http://arxiv.org/abs/1706.02677.

Song Han. *Efficient Methods and Hardware for Deep Learning*. PhD thesis, Stanford University, 2017.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 28*, pp. 1135–1143. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5784-learning-both-weights-and-connections-for-efficient-neural-network.pdf.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1510.00149.

Babak Hassibi, David G. Stork, Gregory Wolff, and Takahiro Watanabe. Optimal brain surgeon: Extensions and performance comparisons. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, pp. 263–270, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc. URL http://dl.acm.org/citation.cfm?id=2987189.2987223.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, June 2016. doi: 10.1109/CVPR.2016.90.

Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *The European Conference on Computer Vision (ECCV)*, September 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 448–456. JMLR.org, 2015. URL http://dl.acm.org/citation.cfm?id=3045118.3045167.

A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

Yann Le Cun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, pp. 598–605. Morgan Kaufmann, 1990.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. SNIP: SINGLE-SHOT NETWORK PRUNING BASED ON CONNECTION SENSITIVITY. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=B1VZqjAcYX.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017. URL https://openreview.net/forum?id=rJqFGTslg.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJlnB3C5Ym.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1412–1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics. URL http://aclweb.org/anthology/D15-1166.

MLPerf. MLPerf v0.5 Training GNMT, 2019. See www.mlperf.org for more information.

Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 2498–2507. JMLR.org, 2017. URL http://dl.acm.org/citation.cfm?id=3305890.3305939.

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.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pp. 3104–3112, Cambridge, MA, USA, 2014. MIT Press. URL http://dl.acm.org/citation.cfm?id=2969033.2969173.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv*, art. arXiv:1609.08144, Sep 2016.

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

SUPPLEMENTARY MATERIAL: COMPARING FINE-TUNING AND REWINDING IN NEURAL NETWORK PRUNING

Anonymous authors
Paper under double-blind review

A PRUNING ALGORITHMS

The following 3 algorithms provide more formal descriptions of the three experiments (rewinding, fine-tuning, and reinitialization) described in Section 2.

Pruning Frameworks

A neural network is a function $f(x; W, mask)$ with weights $W \in \mathcal{R}^d$ and pruning mask $mask \in \{0,1\}^d$ such that the network parameters are the element-wise product $W \odot mask$.

RANDOM-INITIALIZATION() creates a random initialization of the weights in the network.

TRAIN$(W, mask, e, h)$ returns the weights resulting from training a masked network $W \odot mask$ for e epochs using the hyperparameters in h (e.g. learning rate, batch size, etc.).

PRUNE$(W, mask, h)$ removes weights from masked network $W \odot mask$ and returns an updated mask, using the hyperparameters in h (e.g. the pruning heuristic and percentage of weights to prune).

Algorithm 1 Prune with rewinding.

function REWIND($train_epochs$, $train_hparams$, $prune_iters$, $prune_hparams$, $retrain_epochs$)
  $W_0 \leftarrow$ RANDOM-INITIALIZATION()
  $mask \leftarrow 1^d$
  $rewind_epoch \leftarrow train_epochs - retrain_epochs$
  $W^* \leftarrow$ TRAIN$(W_0, mask, rewind_epoch, train_hparams)$
  $W \leftarrow$ TRAIN$(W^*, mask, retrain_epochs, train_hparams)$
  for $iter$ in $prune_iters$ **do**
    $mask \leftarrow$ PRUNE$(W, mask, prune_hparams)$
    $W \leftarrow$ TRAIN$(W^*, mask, retrain_epochs, train_hparams)$
  return W, $mask$

Algorithm 2 Prune with fine-tuning.

function FINE-TUNE($train_epochs$, $train_hparams$, $prune_iters$, $prune_hparams$, $retrain_epochs$, $finetune_hparams$)
  $W_0 \leftarrow$ RANDOM-INITIALIZATION()
  $mask \leftarrow 1^d$
  $W \leftarrow$ TRAIN$(mask, train_epochs, train_hparams)$
  for $iter$ in $prune_iters$ **do**
    $mask \leftarrow$ PRUNE$(W, mask, prune_hparams)$
    $W \leftarrow$ TRAIN$(W, mask, finetune_epochs, finetune_hparams)$
  return W, $mask$

Algorithm 3 Reinitialize.

function REINITIALIZE($train_epochs$, $train_hparams$, $prune_iters$, $prune_hparams$, $retrain_epochs$)
  $W_0 \leftarrow$ RANDOM-INITIALIZATION()
  $mask \leftarrow 1^d$
  $W \leftarrow$ TRAIN(W_0, $mask$, $train_epochs$, $train_hparams$)
  $mask \leftarrow$ PRUNE(W, $mask$, $prune_hparams$)
  $W'_0 \leftarrow$ RANDOMINITIALIZATION()
  $W \leftarrow$ TRAIN(W'_0, $mask$, $train_epochs + retrain_epochs \times prune_iters$, $train_hparams$)
  return W, $mask$

B STRUCTURED PRUNING

The pruning technique studied in the main body of the paper, unstructured global magnitude pruning, does not immediately result in inference speedups without the support of sparse linear algebra libraries or specialized hardware. To derive smaller networks with faster inference on commodity hardware, practitioners use *structured* pruning techniques, which prune entire neurons, filters, or other structure from the network, rather than the individual weights pruned by unstructured pruning. In this appendix, we compare fine-tuning and rewinding on a commonly used structured pruning technique, the filter pruning technique introduced by Li et al. (2017). This technique prunes entire convolutional filters (which are stacks of 2D kernels) from the network, according to a simple heuristic, the L1 norm of the filter. Given hand-crafted per-layer pruning rates (p_i, which for layer i denotes that $p_i\%$ of filters should be removed from that layer), this technique produces a reduced size network.

B.1 METHODOLOGY

We study a ResNet-56 and ResNet-110, both trained on CIFAR-10, using the same methodology described for the ResNet-20 in Section 2. We also study a ResNet-34, trained on ImageNet, using the same methodology described for the ResNet-50 in Section 2. We use the layerwise pruning percentages given by Li et al. (2017): specifically, the VGG-16-A, ResNet-56-{A,B}, ResNet-110-{A,B}, and ResNet-34-{A,B}. However, at these exact sparsity levels, fine-tuning and rewinding are indistinguishable. To this end, we apply these pruning rates iteratively to obtain sparser networks pruned with these rates. In the kth iteration of applying pruning technique T (denoted T^k), the effective pruning rate for layer i is $1 - (1 - p_i)^k$. For instance, a layer that was originally pruned 20% will be pruned $1 - (1 - 0.2)^3 = 48.8\%$ in T^3. We compare all pruning algorithms using one-shot pruning and re-training.

B.2 RESULTS

We show the results of the structured pruning experiments in the following plots.

Like the plots in Section 3.2, Figure 1 presents a comparison of the most accurate networks achievable by fine-tuning and rewinding across different networks and sparsity levels. Across all networks, pruning rates, and sparsity levels, rewinding can achieve more accurate networks than fine-tuning for this structured pruning technique.

Like the plots in Section 4, Figure 2 shows the *safety* of rewinding across different pruning techniques and networks, showing the point at which rewinding and re-training for a given number of epochs achieves higher accuracy than fine-tuning for up to that many epochs. Similar to the results in Section 4, rewinding is safer at higher sparsities, and is generally safe within the same range of rewind points, with two exceptions: in ResNet-56-A, the methods perform effectively identically, and in VGG-16-A, there are some highly sparse points where rewinding's accuracy plummets. The points at which fine-tuning outperform rewinding in VGG-16 are points at which the pruned network's accuracy occasionally drops to random, when an entire layer of the network is pruned away.

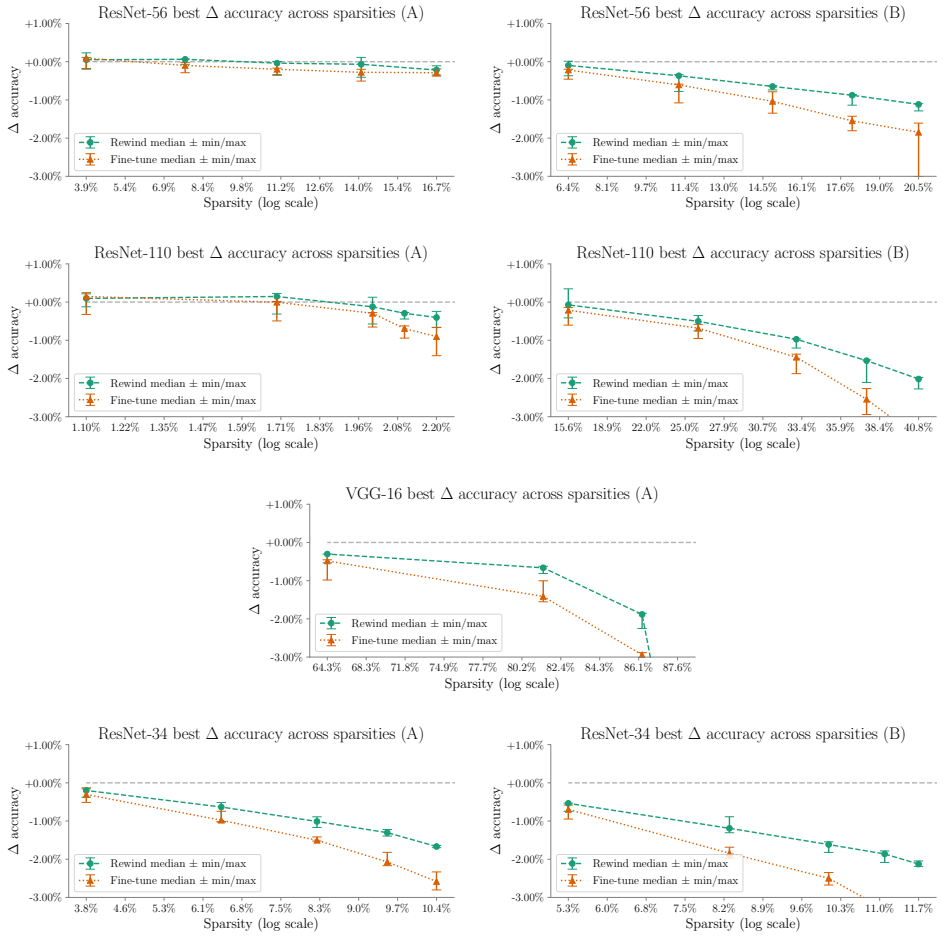


Figure 1: The most accurate networks achievable by different re-training methods across different networks and target sparsities, using structured pruning techniques.

This is a byproduct of our implementation adhering exactly to the specified pruning rates, and not a property of rewinding.

Figure 3 shows the *dominance* of rewinding in the structured setting. Again, rewinding is dominant with more sparsity and re-training iterations. The primary difference between these results and those in Section 4 is that rewinding does not seem to drop in performance when the network is rewound to the beginning of training. Frankle & Carbin (2019) and Frankle et al. (2019) do not study the structured pruning regime, so this result does not contradict any prior work.

Figures 4 through 10 compare fine-tuning and rewinding across networks and sparsity levels using the same number of re-training epochs for each, showing that epoch-for-epoch, rewinding outperforms fine-tuning in most situations. Plots of this style for unstructured pruning are shown in Appendix G. In these plots, the y-axis is the change in accuracy between the pruned network and the original network (higher is better), and the x-axis is the number of epochs that the pruned network was re-trained for according to Algorithms 1 and 2. For fine-tuning, the x-axis denotes the number of fine-tuning epochs. For rewinding, the x-axis denotes the epoch $T - x$ that the network was rewound to (where T is the original training time of the full network) and the subsequent x epochs of re-training.

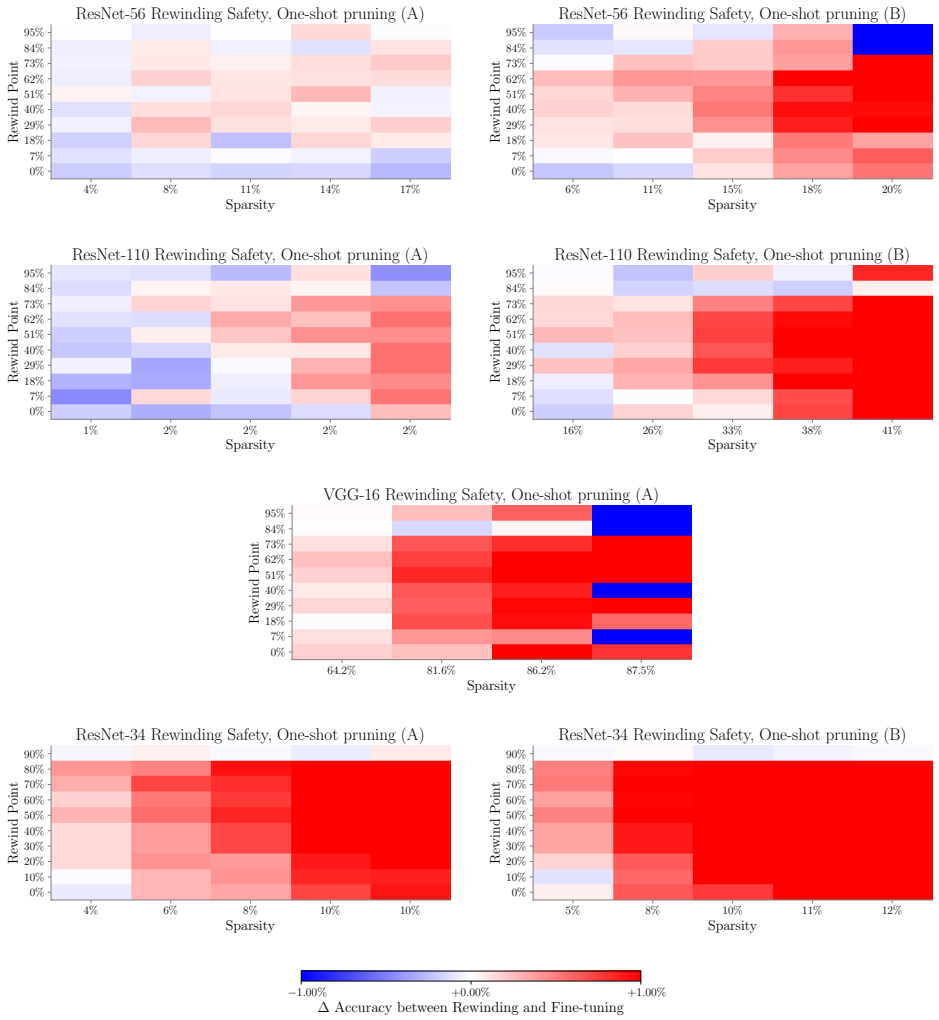


Figure 2: Safety across networks and pruning methods. Red points denote rewinding achieving higher accuracy than fine-tuning for any amount up through the same re-training budget; blue points denote the opposite.

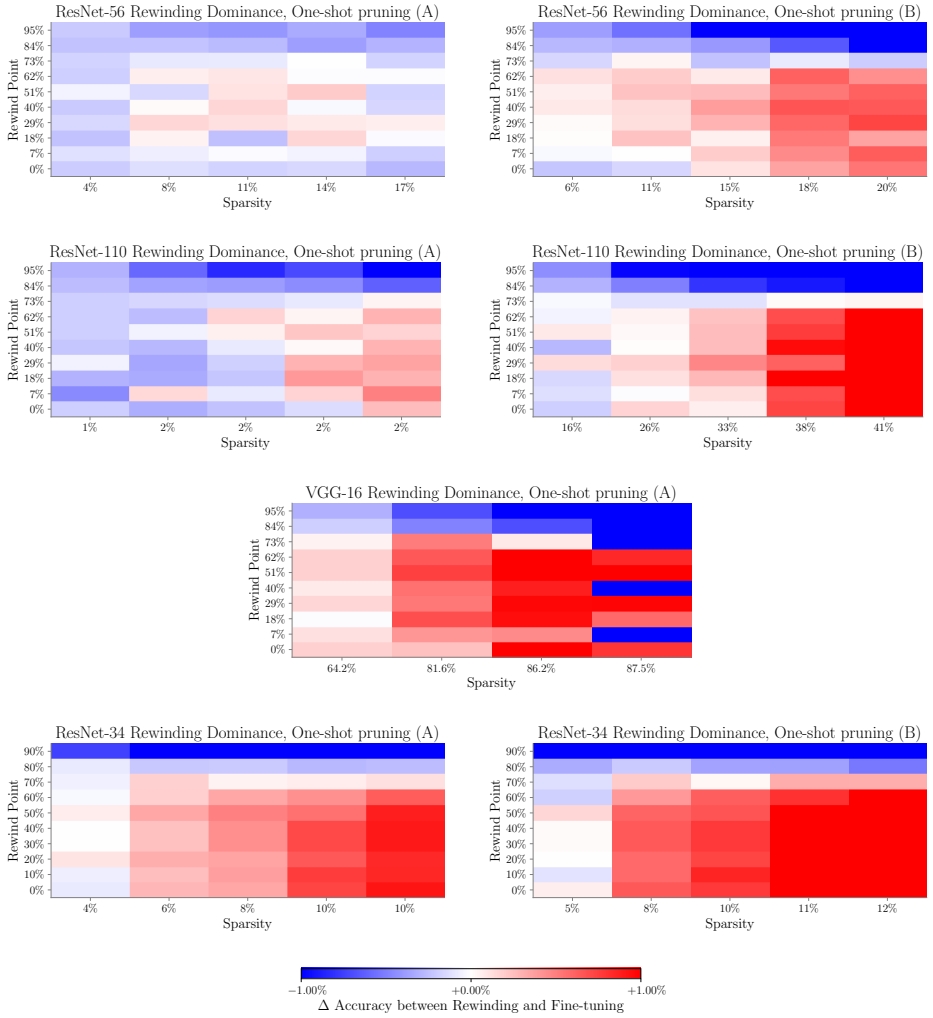


Figure 3: Dominance across networks and pruning methods. Red points denote rewinding achieving higher accuracy than fine-tuning for any amount; blue points denote the opposite.

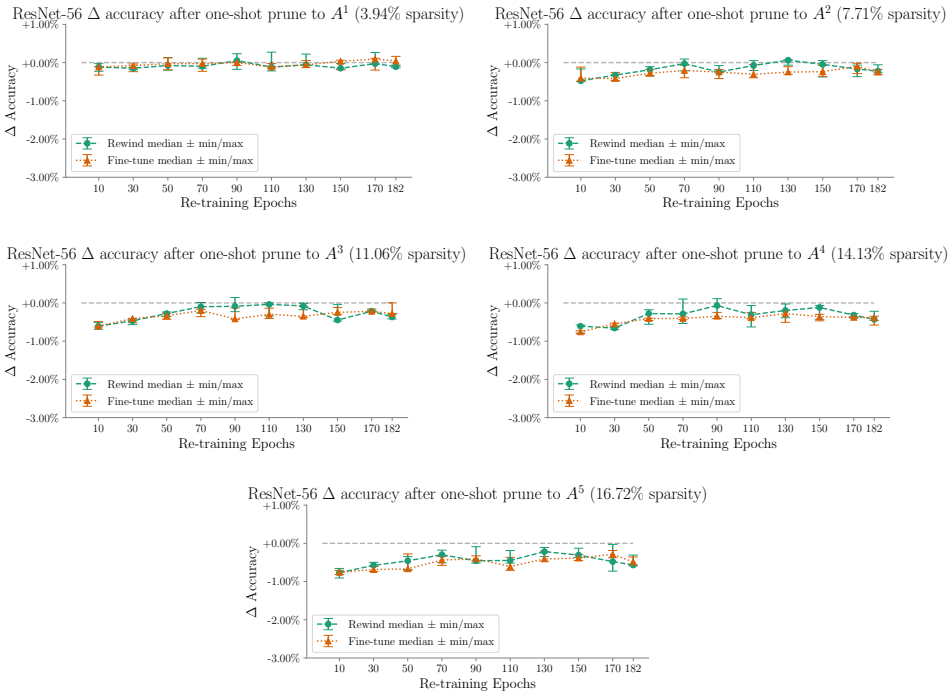


Figure 4: Comparison of equal budget fine-tuning and rewinding for ResNet-56-A.

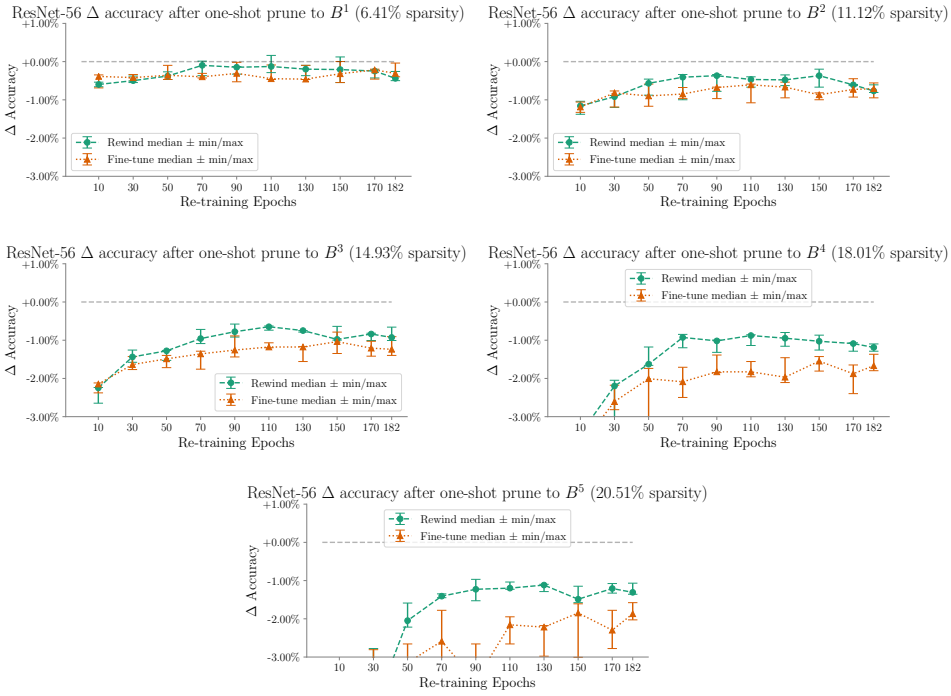


Figure 5: Comparison of equal budget fine-tuning and rewinding for ResNet-56-B.

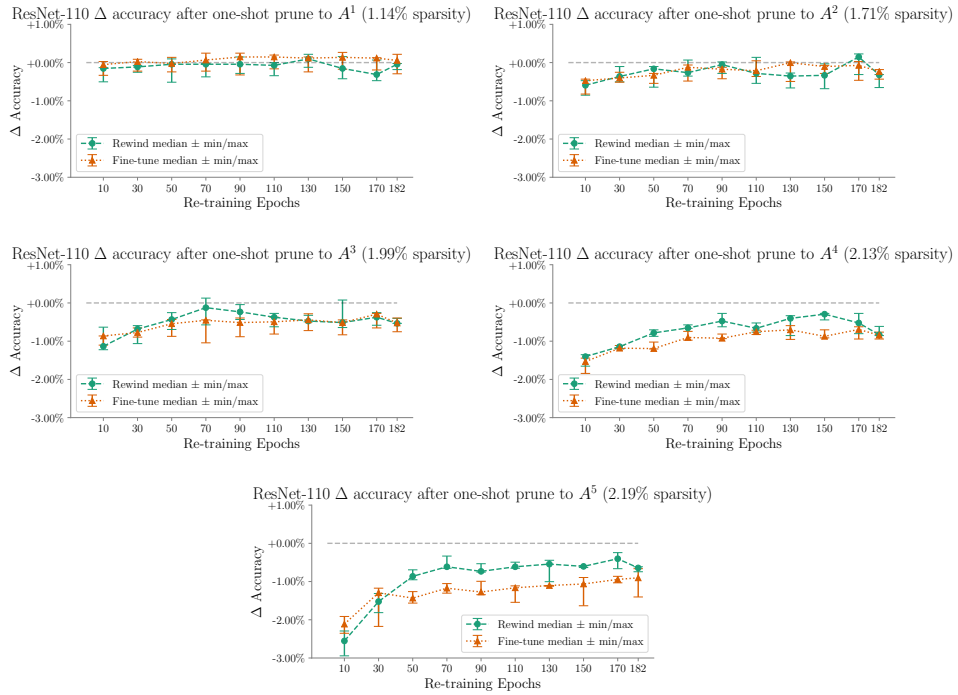


Figure 6: Comparison of equal budget fine-tuning and rewinding for ResNet-110-A.

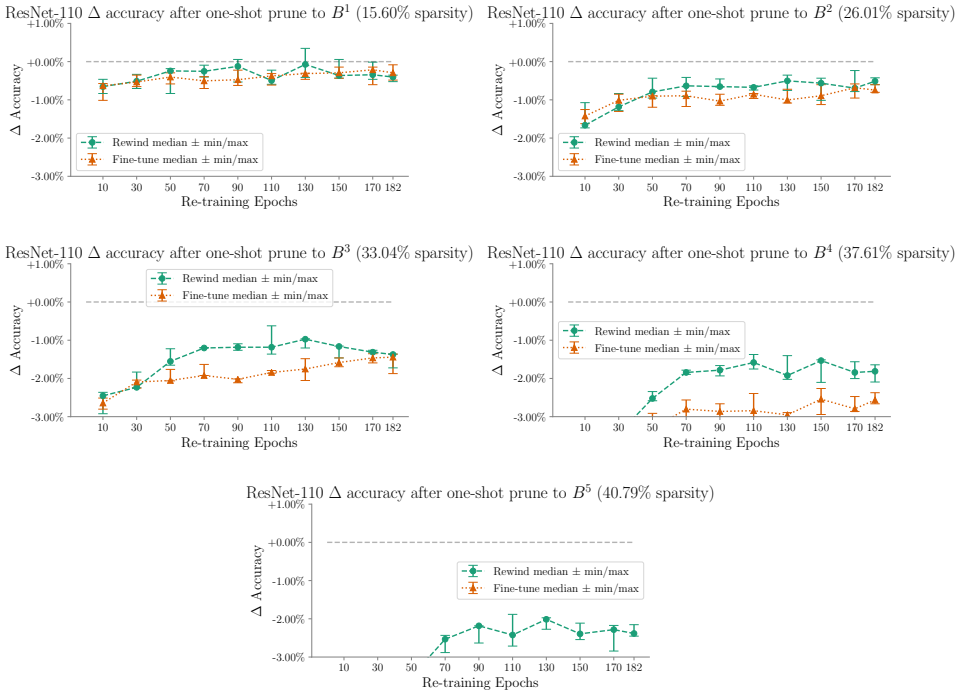


Figure 7: Comparison of equal budget fine-tuning and rewinding for ResNet-110-B.

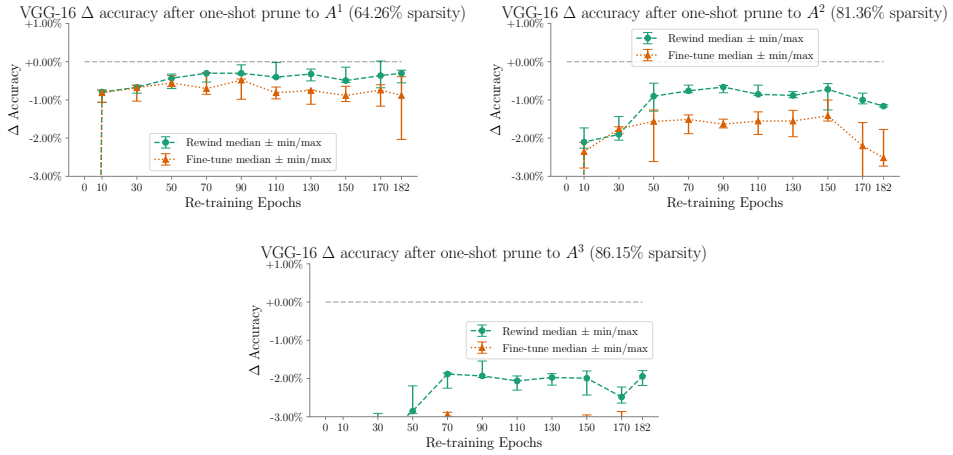


Figure 8: Comparison of equal budget fine-tuning and rewinding for VGG-16-A.

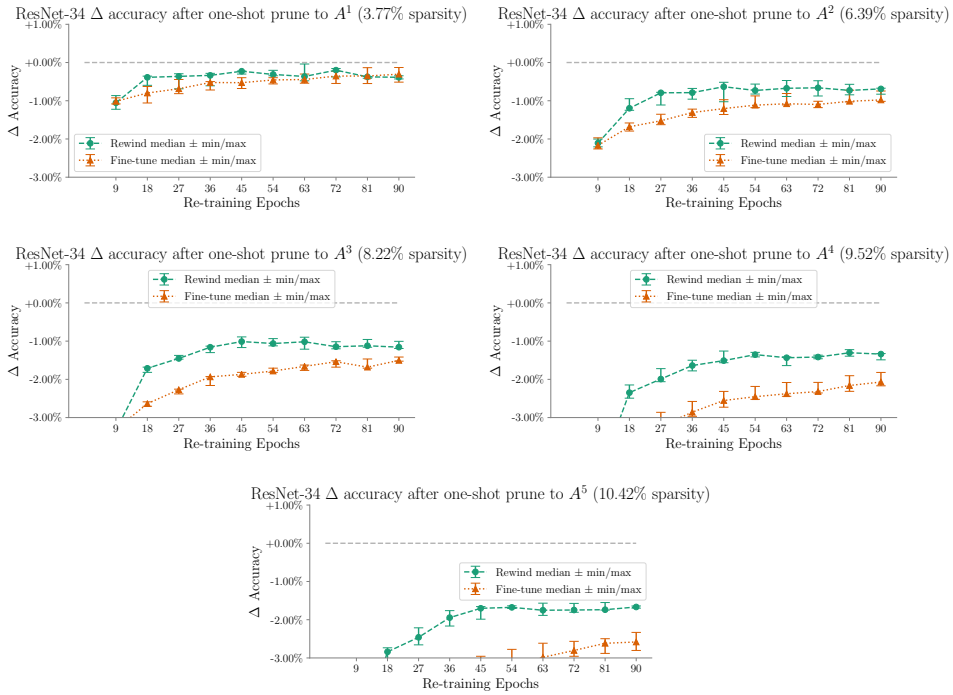


Figure 9: Comparison of equal budget fine-tuning and rewinding for ResNet-34-A.

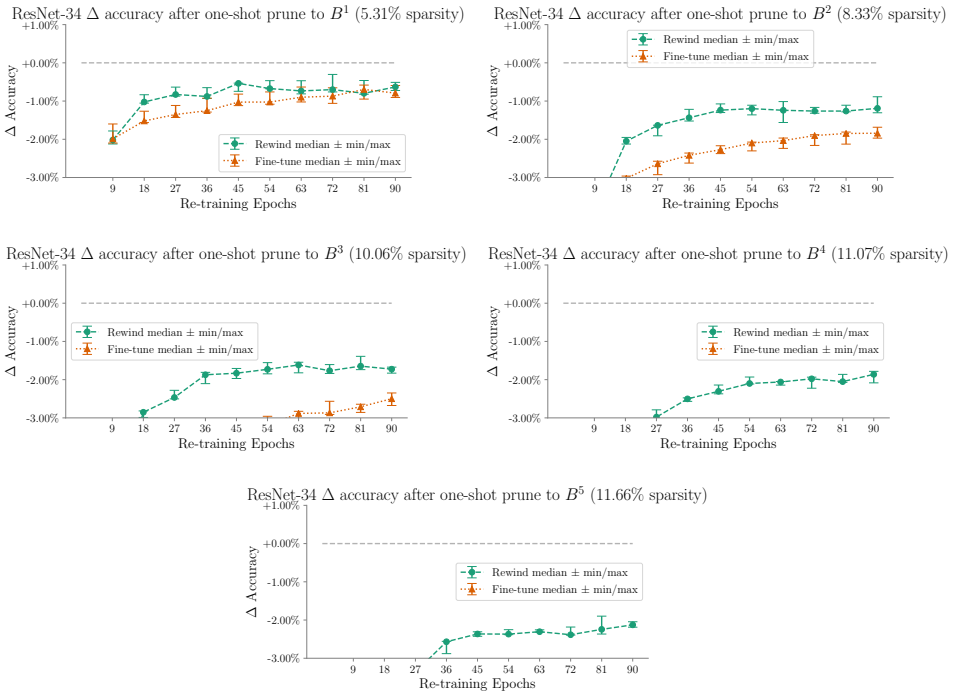


Figure 10: Comparison of equal budget fine-tuning and rewinding for ResNet-34-B.

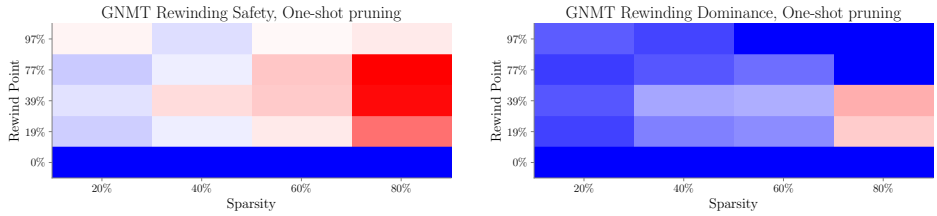


Figure 11: Safety and dominance for the GNMT model.

C Machine Translation

To evaluate whether fine-tuning and rewinding have the same performance and trade-offs on other datasets, we compared fine-tuning and rewinding on GNMT (Wu et al., 2016) in the MLPerf benchmark suite (MLPerf, 2019), which is a seq-to-seq (Sutskever et al., 2014) model based on stacked LSTMs (Hochreiter & Schmidhuber, 1997), trained on the WMT English-German dataset (Luong et al., 2015). We use a standard implementation provided by NVIDIA[1]. Due to hardware and time constraints, we use hidden layer sizes of 512 units rather than 1024 units. We prune all weights using global magnitude pruning. We compare across a small set of sparsities and re-training budgets.

Our results are consistent with the results for vision networks, showing that at high sparsities (80%), rewinding outperforms fine-tuning on this LSTM-based network. At lower sparsities, fine-tuning outperforms rewinding, and at medium sparsities, the two techniques behave approximately equivalently. These results match those seen for vision networks. Comparisons between fine-tuning and rewinding on the GNMT model are presented in Figures 11 through 13.

[1] `https://github.com/NVIDIA/DeepLearningExamples/tree/4e00153/TensorFlow/Translation/GNMT`

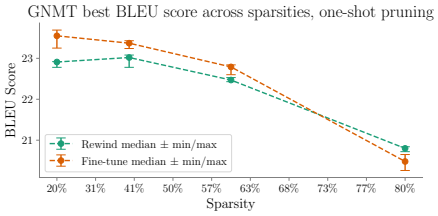


Figure 12: Max accuracy achievable across sparsities for the GNMT model.

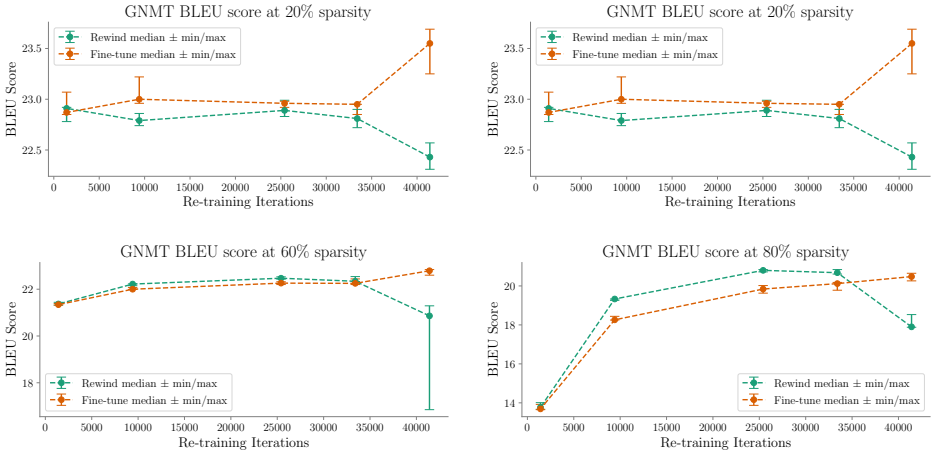


Figure 13: Equal budget re-training times across sparsities for the GNMT model.

D FLOPs vs. Sparsity Comparison

In the main body of the paper, we use the percent of pruned weights as the metric denoting the sparsity of a given neural network. However, the sparsity does not tell the full story: networks of different sparsities can require different amounts of floating point operations (FLOPs) to perform inference. For instance, pruning a weight in the convolutional first layer of a ResNet results in pruning more FLOPs than pruning a weight in the last convolutional layer. In this appendix, we analyze the number of FLOPs required to perform inference on pruned networks acquired through fine-tuning and rewinding.

Our methodology in Section 2.2 for one-shot pruning uses the same initial trained network for our comparisons between both methods, and prunes using the same pruning technique. This means that both networks have the exact same sparsity pattern, and therefore same number of FLOPs. For iterative pruning the networks differ, meaning the FLOPs also differ, since we use global pruning.

Figure 14 below shows the FLOPs required for iterative pruning by different re-training techniques on a ResNet-50. Each plot compares the techniques using iterative pruning with a fixed number of re-training epochs. The x-axis denotes the sparsity level, corresponding to the number of iterations of iterative pruning. The y-axis on the left (corresponding to the orange and blue lines) denotes the number of GFLOPs required to perform inference on the networks: the original network requires 3.9 GFLOPs to execute; the number of FLOPs required to execute decreases for sparser networks. Lower values on this axis means that the network executes faster. The y-axis on the right (corresponding to the red line) denotes the relative speedup of the network obtained by rewinding compared to the network obtained by fine-tuning; that is, the ratio of fine-tuning FLOPs to rewinding FLOPs. Higher values on this axis means that at this sparsity, the network obtained by rewinding requires fewer FLOPs than the network obtained by fine-tuning.

Iterative pruning with rewinding finds networks that require fewer FLOPs than those found by iterative pruning with fine-tuning. This advantage correlates with the accuracy advantage of the network

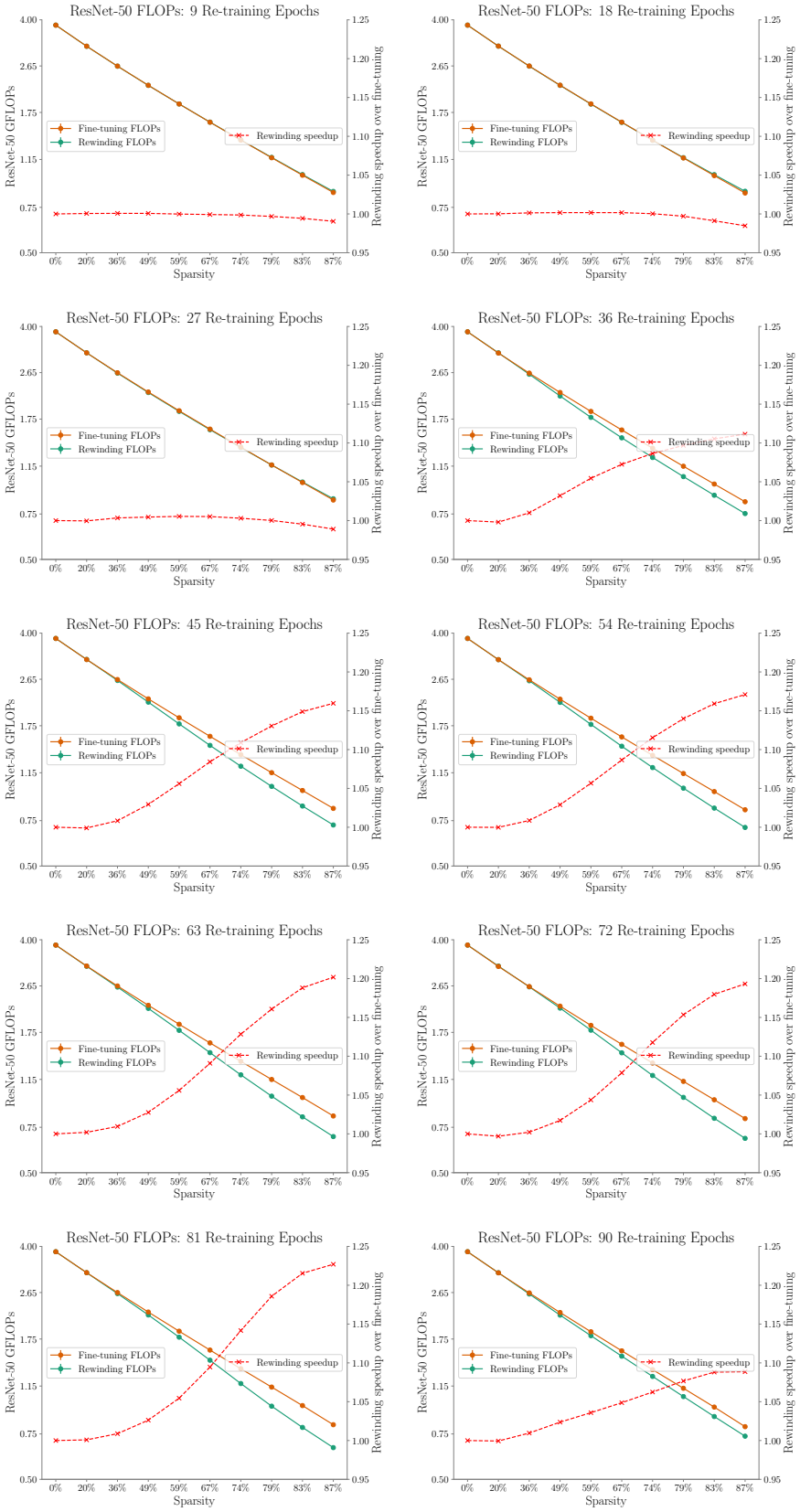


Figure 14: ResNet-50 FLOPs

obtained by rewinding: re-training for 81 epochs results in the most accurate network that rewinding can obtain, and results in the largest speedup over fine-tuning; re-training for 9 or 90 epochs results in a less accurate network, and results in less of a speedup. After 10 pruning iterations, corresponding to 87% sparsity, a ResNet-50 pruned with rewinding will have a $1.23\times$ speedup over a ResNet-50 pruned with fine-tuning to 87% sparsity. When pruning to match the accuracy of the original network (corresponding to the networks in Section 3.1), a ResNet-50 pruned to 79% sparsity using rewinding will require $2.3\times$ fewer FLOPs than an equivalently accurate network obtained by fine-tuning pruned to 48.8% sparsity.

E FINE-TUNING TIME

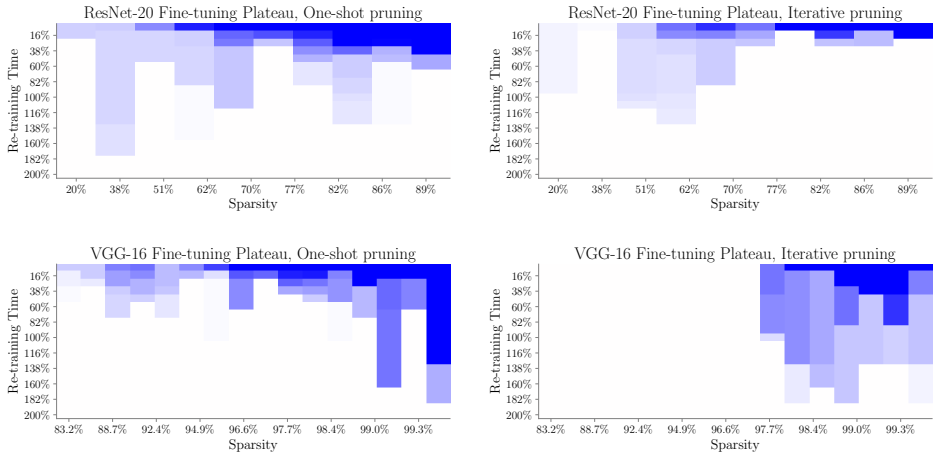
Background. Many standard approaches to pruning only fine-tune for up to 25% of the training time. For instance, Liu et al. (2019) fine-tune CIFAR-10 networks for 40 epochs against an original training schedule of 160 epochs (25%), and fine-tune Imagenet networks for 20 epochs against an original training schedule of 90 epochs (22%). Li et al. (2017) also fine-tune for 40 epochs against the original 160 on CIFAR-10 networks. This is not always the case though: Han et al. (2015) fine-tune for a total time up to $3\times$ the original training time of the network, although this is in the context of iterative pruning.

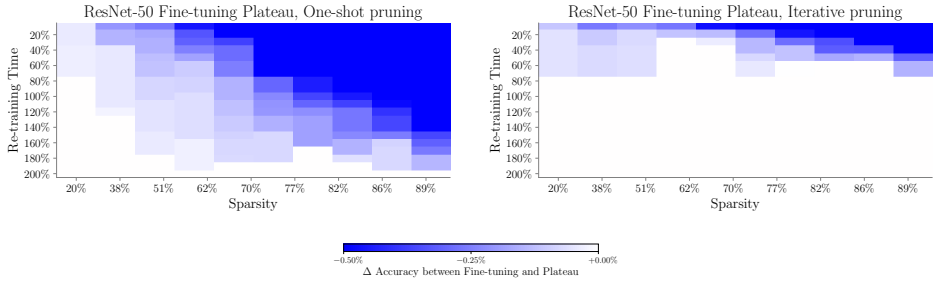
Data. Consider the plots below, which replicate the plots from Section 4, but instead considering fine-tuning's room for improvement. The y axis is re-training time: points along a given row represent the difference between fine-tuning for at most $y\%$ of the original training time of the work and fine-tuning for up to $2\times$ the original training time for the network. White points denote points where fine-tuning has plateaued; blue points denote points where more fine-tuning can help.

Analysis We find that in many cases, a quarter of the original re-training time is not enough for fine-tuning's accuracy to plateau. In ResNet-50 one-shot pruning, training does not seem to plateau

We also find that the required re-training time for accuracy to plateau is sparsity-dependent. This is unsurprising, as more pruning brings the network further away from the function it initially learned. However, standard approaches in the literature do not acknowledge this fact, and instead use a fixed fine-tuning schedule, rather than a sparsity-dependent one or a schedule that stops training upon accuracy plateau.

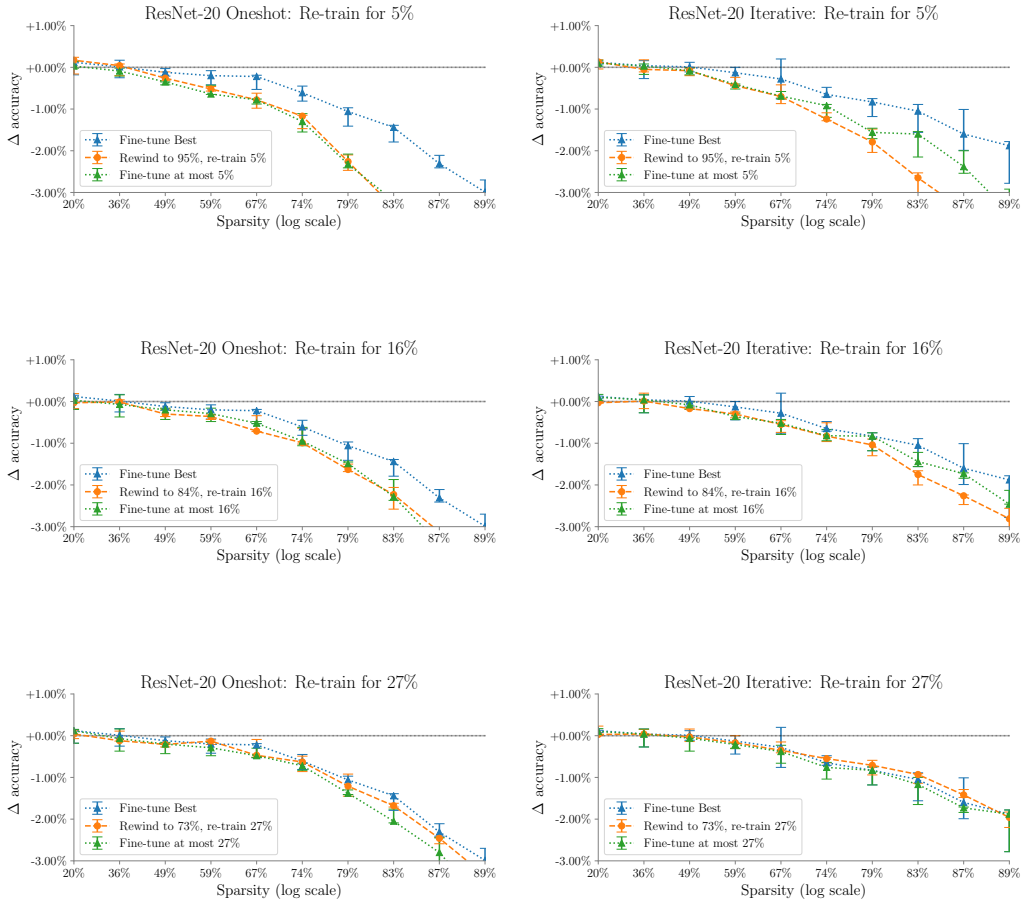
More views and analysis of the same underlying data can be found in Appendix F and Appendix G.

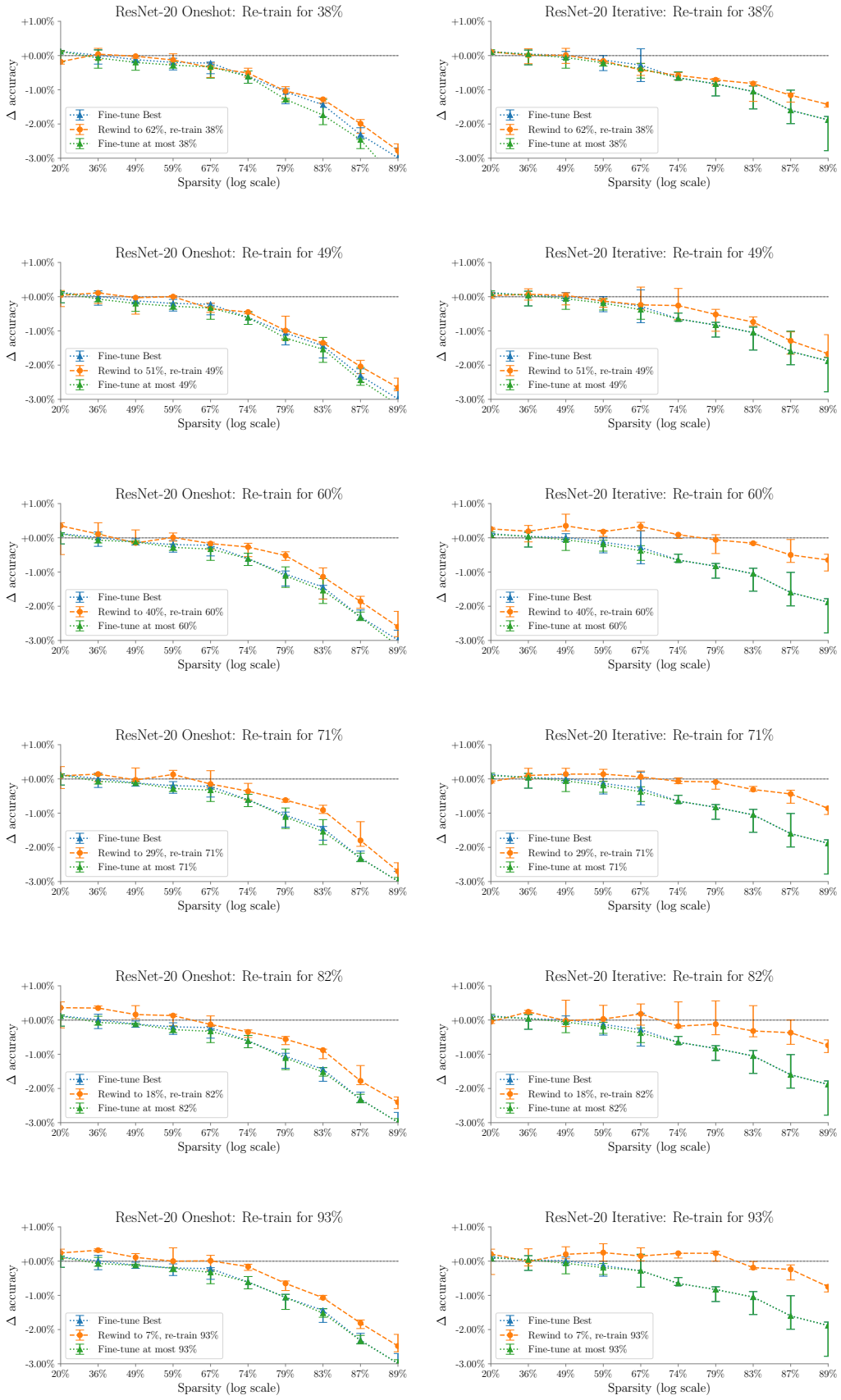


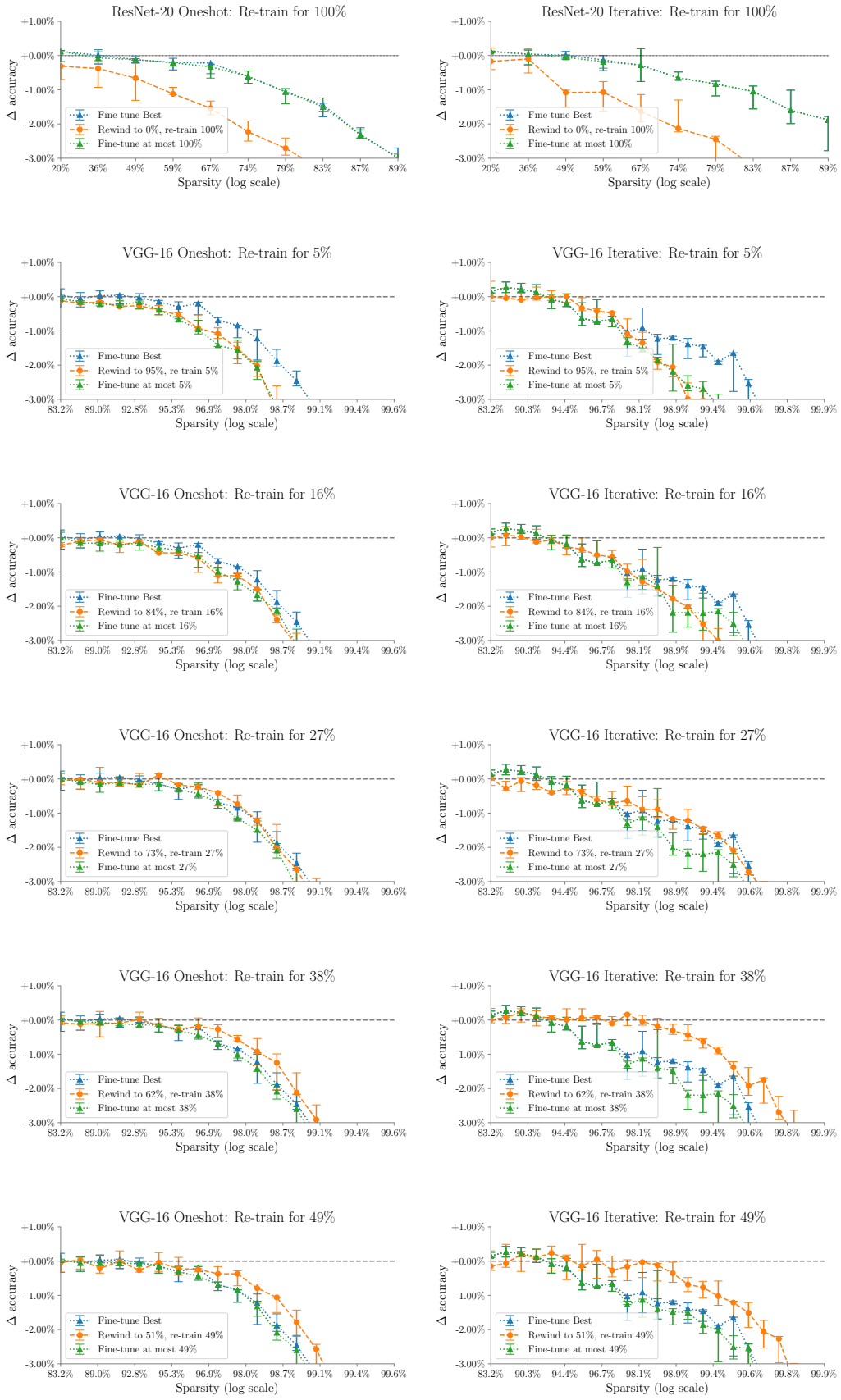


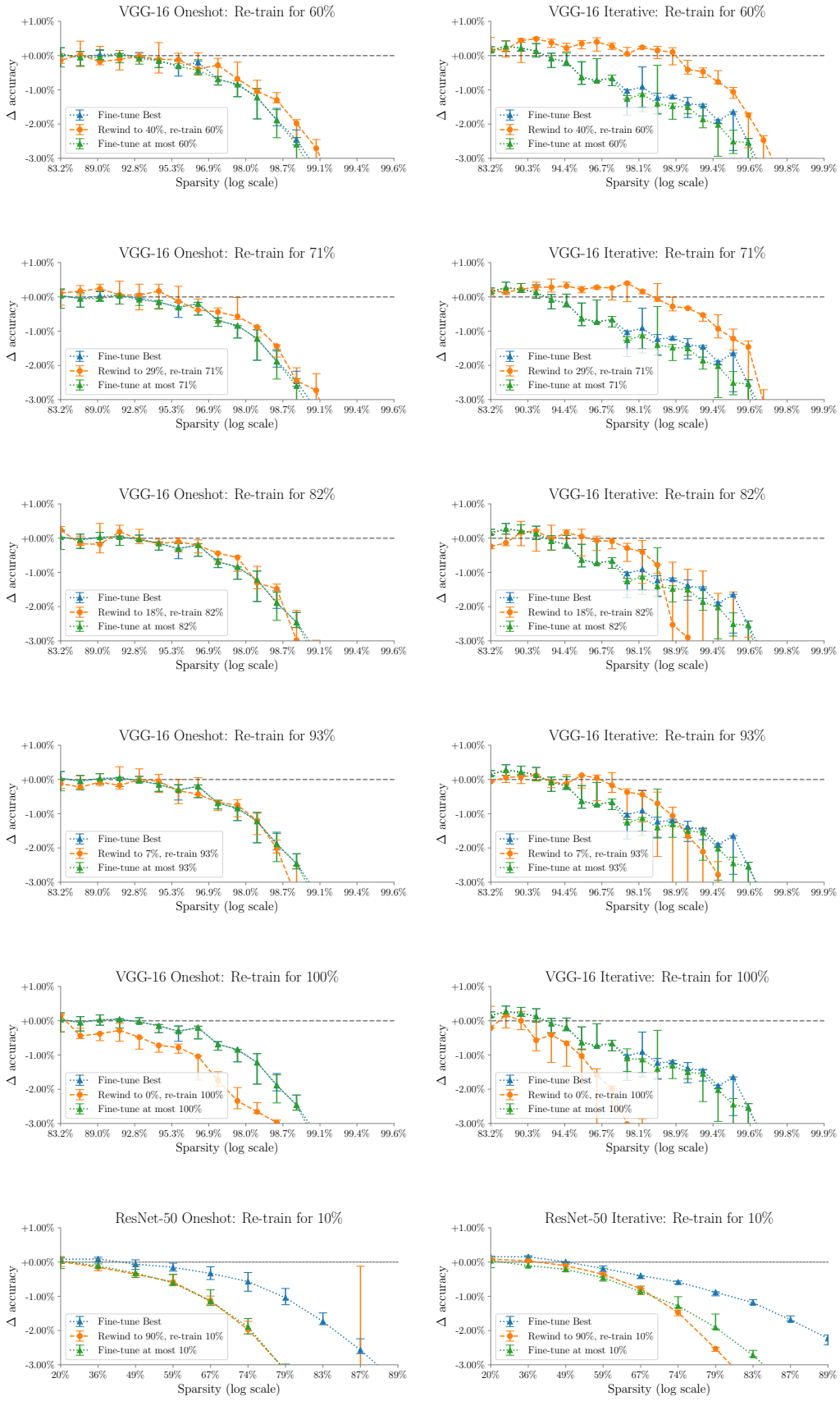
F BUDGET-LIMITED ACCURACIES

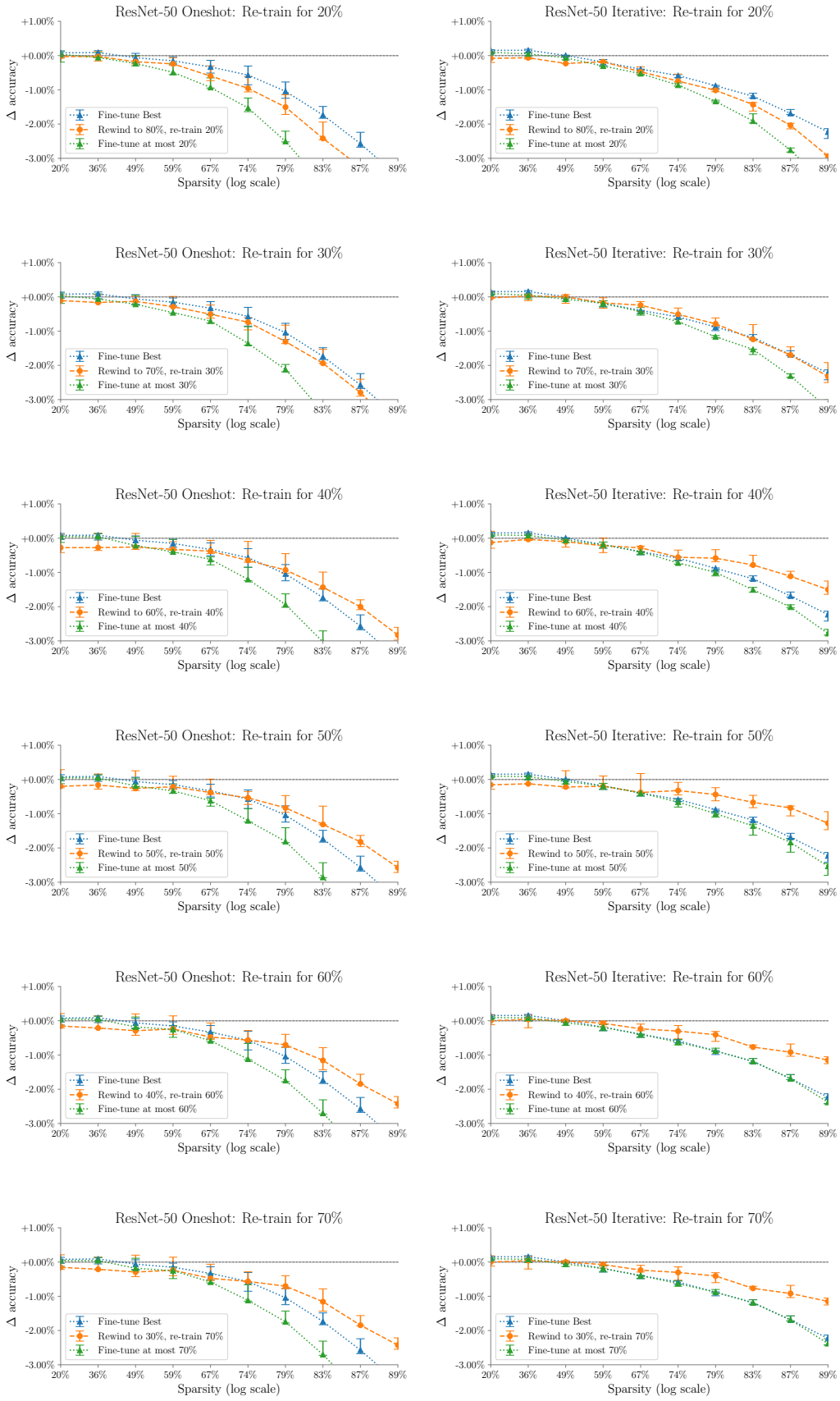
In this appendix, we consider the accuracy attained by rewinding to specific point in training, fine-tuning for any amount up to and including the number of epochs for which that rewound network is re-trained, and fine-tuning for any amount at all. These plots can be considered versions of the plots in Figure 1, but with rewinding to specific points and with budget-constrained fine-tuning. The intention of these plots is to show the absolute accuracy attainable by picking specific rewind points, and to provide another view of the safe and dominant regions as described in Section 4. These plots can be seen in the plots below.

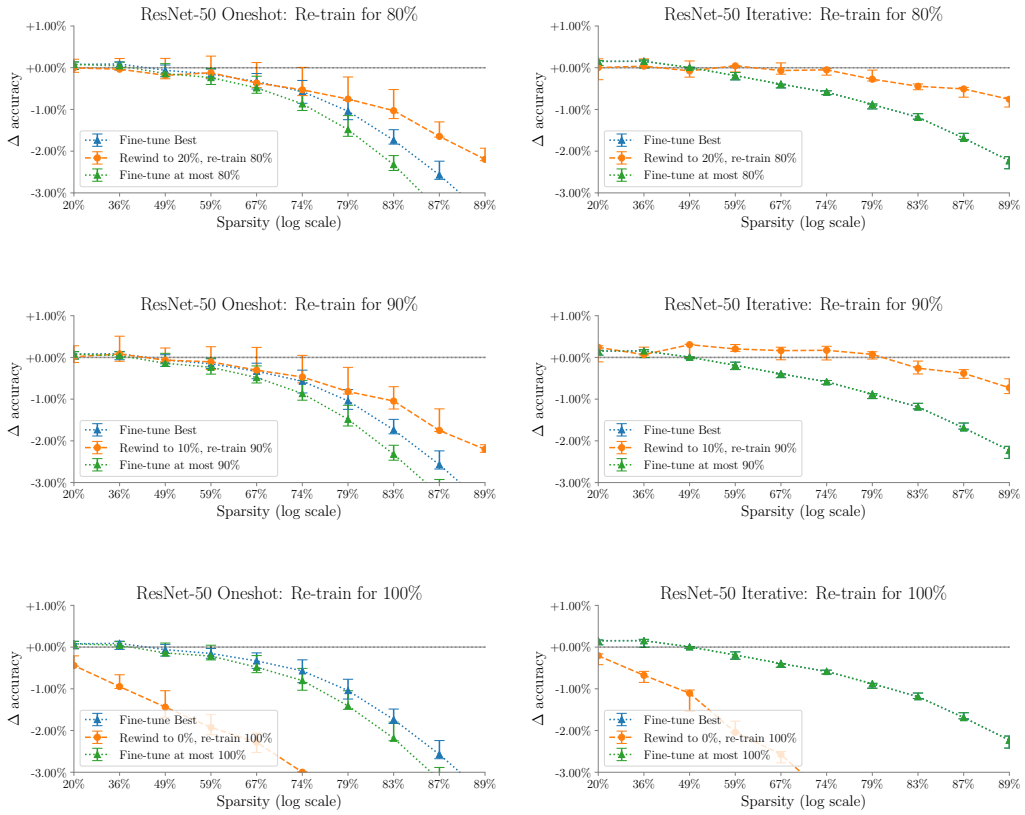



ResNet-20 Oneshot: Re-train for 38%
ResNet-20 Iterative: Re-train for 38%
Fine-tune Best
Rewind to 62%, re-train 38%
Fine-tune at most 38%
ResNet-20 Oneshot: Re-train for 49%
ResNet-20 Iterative: Re-train for 49%
Fine-tune Best
Rewind to 51%, re-train 49%
Fine-tune at most 49%
ResNet-20 Oneshot: Re-train for 60%
ResNet-20 Iterative: Re-train for 60%
Fine-tune Best
Rewind to 40%, re-train 60%
Fine-tune at most 60%
ResNet-20 Oneshot: Re-train for 71%
ResNet-20 Iterative: Re-train for 71%
Fine-tune Best
Rewind to 29%, re-train 71%
Fine-tune at most 71%
ResNet-20 Oneshot: Re-train for 82%
ResNet-20 Iterative: Re-train for 82%
Fine-tune Best
Rewind to 18%, re-train 82%
Fine-tune at most 82%
ResNet-20 Oneshot: Re-train for 93%
ResNet-20 Iterative: Re-train for 93%
Fine-tune Best
Rewind to 7%, re-train 93%
Fine-tune at most 93%
Δ accuracy
+1.00%
+0.00%
-1.00%
-2.00%
-3.00%
20% 36% 49% 59% 67% 74% 79% 83% 87% 89%
Sparsity (log scale)


ResNet-20 Oneshot: Re-train for 100%
ResNet-20 Iterative: Re-train for 100%
VGG-16 Oneshot: Re-train for 5%
VGG-16 Iterative: Re-train for 5%
VGG-16 Oneshot: Re-train for 16%
VGG-16 Iterative: Re-train for 16%
VGG-16 Oneshot: Re-train for 27%
VGG-16 Iterative: Re-train for 27%
VGG-16 Oneshot: Re-train for 38%
VGG-16 Iterative: Re-train for 38%
VGG-16 Oneshot: Re-train for 49%
VGG-16 Iterative: Re-train for 49%
Δ accuracy
+1.00%
+0.00%
-1.00%
-2.00%
-3.00%
Sparsity (log scale)
20% 36% 49% 59% 67% 74% 79% 83% 87% 89%
83.2% 89.0% 92.8% 95.3% 96.9% 98.0% 98.7% 99.1% 99.4% 99.6%
83.2% 90.3% 94.4% 96.7% 98.1% 98.9% 99.4% 99.6% 99.8% 99.9%
Fine-tune Best
Rewind to 0%, re-train 100%
Fine-tune at most 100%
Rewind to 95%, re-train 5%
Fine-tune at most 5%
Rewind to 84%, re-train 16%
Fine-tune at most 16%
Rewind to 73%, re-train 27%
Fine-tune at most 27%
Rewind to 62%, re-train 38%
Fine-tune at most 38%
Rewind to 51%, re-train 49%
Fine-tune at most 49%


VGG-16 Oneshot: Re-train for 60%
Fine-tune Best
Rewind to 40%, re-train 60%
Fine-tune at most 60%
VGG-16 Iterative: Re-train for 60%
Fine-tune Best
Rewind to 40%, re-train 60%
Fine-tune at most 60%
VGG-16 Oneshot: Re-train for 71%
Fine-tune Best
Rewind to 29%, re-train 71%
Fine-tune at most 71%
VGG-16 Iterative: Re-train for 71%
Fine-tune Best
Rewind to 29%, re-train 71%
Fine-tune at most 71%
VGG-16 Oneshot: Re-train for 82%
Fine-tune Best
Rewind to 18%, re-train 82%
Fine-tune at most 82%
VGG-16 Iterative: Re-train for 82%
Fine-tune Best
Rewind to 18%, re-train 82%
Fine-tune at most 82%
VGG-16 Oneshot: Re-train for 93%
Fine-tune Best
Rewind to 7%, re-train 93%
Fine-tune at most 93%
VGG-16 Iterative: Re-train for 93%
Fine-tune Best
Rewind to 7%, re-train 93%
Fine-tune at most 93%
VGG-16 Oneshot: Re-train for 100%
Fine-tune Best
Rewind to 0%, re-train 100%
Fine-tune at most 100%
VGG-16 Iterative: Re-train for 100%
Fine-tune Best
Rewind to 0%, re-train 100%
Fine-tune at most 100%
ResNet-50 Oneshot: Re-train for 10%
Fine-tune Best
Rewind to 90%, re-train 10%
Fine-tune at most 10%
ResNet-50 Iterative: Re-train for 10%
Fine-tune Best
Rewind to 90%, re-train 10%
Fine-tune at most 10%
Δ accuracy
Sparsity (log scale)


ResNet-50 Oneshot: Re-train for 20%
ResNet-50 Iterative: Re-train for 20%
ResNet-50 Oneshot: Re-train for 30%
ResNet-50 Iterative: Re-train for 30%
ResNet-50 Oneshot: Re-train for 40%
ResNet-50 Iterative: Re-train for 40%
ResNet-50 Oneshot: Re-train for 50%
ResNet-50 Iterative: Re-train for 50%
ResNet-50 Oneshot: Re-train for 60%
ResNet-50 Iterative: Re-train for 60%
ResNet-50 Oneshot: Re-train for 70%
ResNet-50 Iterative: Re-train for 70%
Δ accuracy
+1.00%
+0.00%
-1.00%
-2.00%
-3.00%
Sparsity (log scale)
20%
36%
49%
59%
67%
74%
79%
83%
87%
89%
Fine-tune Best
Rewind to 80%, re-train 20%
Fine-tune at most 20%
Rewind to 70%, re-train 30%
Fine-tune at most 30%
Rewind to 60%, re-train 40%
Fine-tune at most 40%
Rewind to 50%, re-train 50%
Fine-tune at most 50%
Rewind to 40%, re-train 60%
Fine-tune at most 60%
Rewind to 30%, re-train 70%
Fine-tune at most 70%



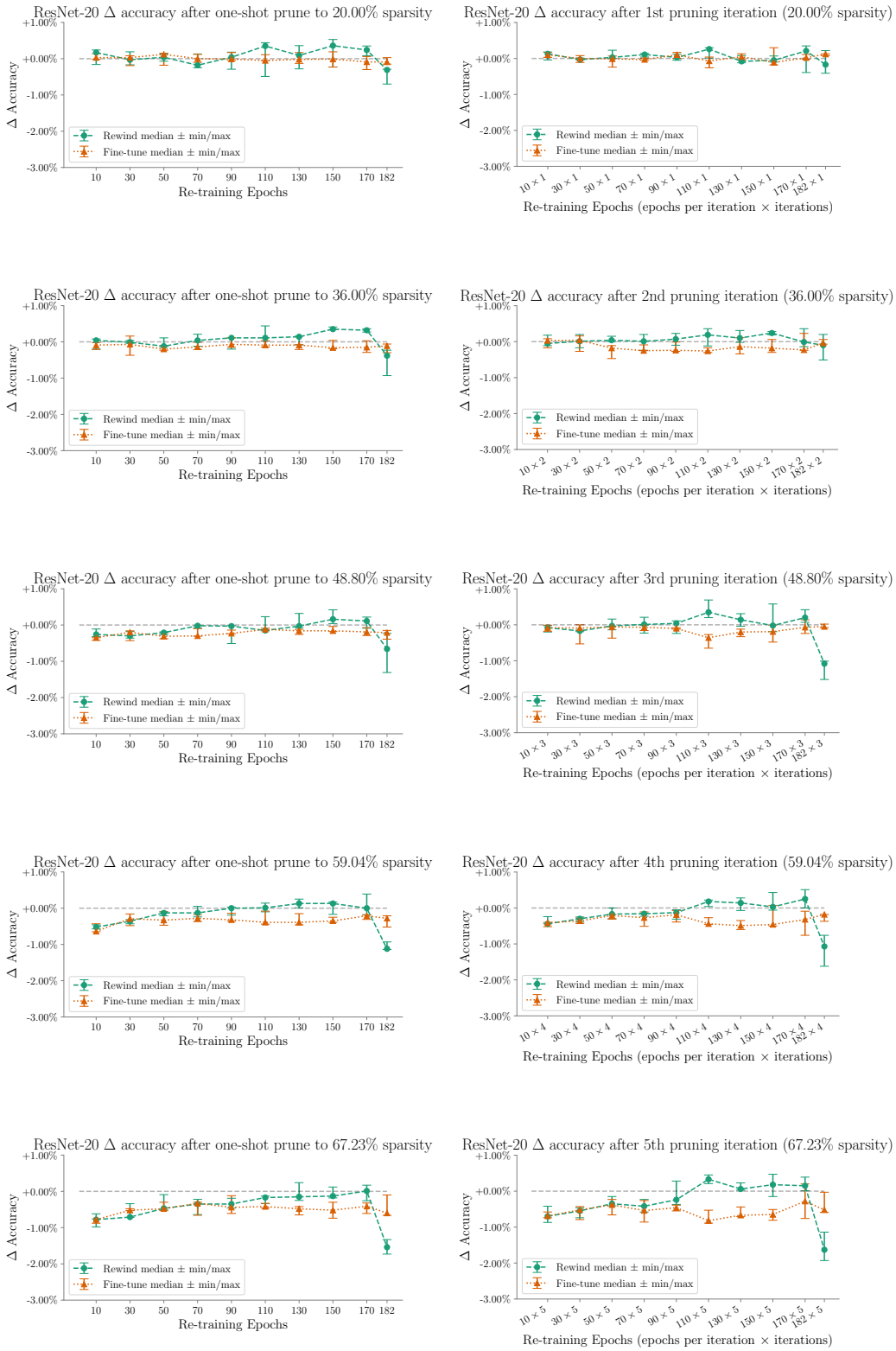
G EQUAL BUDGET RE-TRAINING

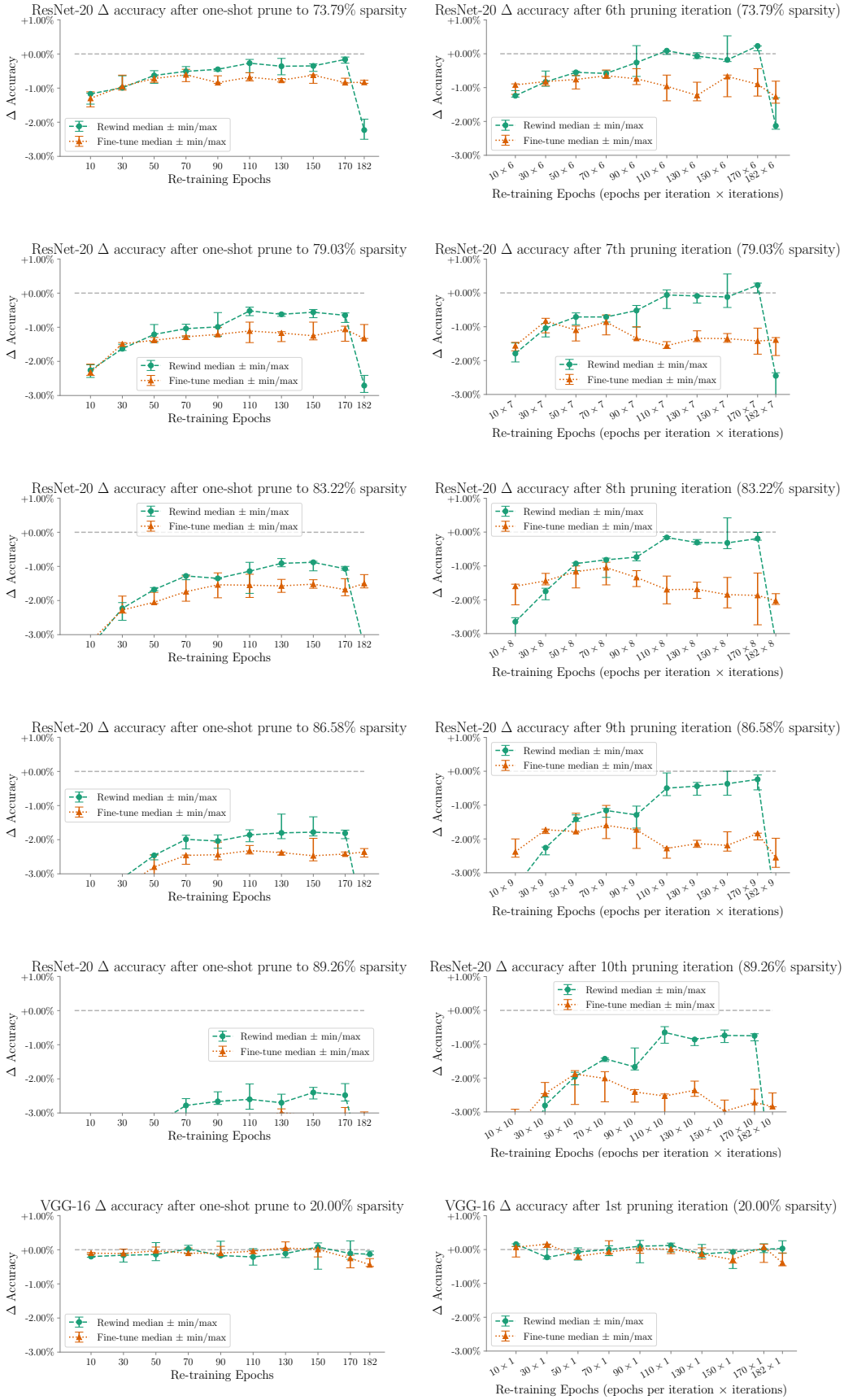
In this appendix, we compare rewinding and fine-tuning using the same number of re-training epochs for each, showing that epoch-for-epoch, rewinding outperforms fine-tuning in most situations, with the exception of iterative re-training for a small number of re-training epochs. This is similar to the concept of safety presented in Section 4, but with an exactly equal budget rather than comparing rewinding with the best-performing set of fine-tuning trials. We include these data for two reasons. First, it is informative to know how many epochs of fine-tuning are necessary for different networks and sparsity levels. Also, we include these data to show a more fine-grained comparison between rewinding and fine-tuning: although fine tuning can stop early, this is both not entirely true for iterative fine-tuning, and also is not representative of the real accuracy of fine-tuning since a max across trials is not an unbiased estimator (e.g. consider the case when learning from fine-tuning plateaus early: since each point is an independent set of 3 trials, the max of all medians picks the set of trials that happened to perform the best, rather than the true median performance).

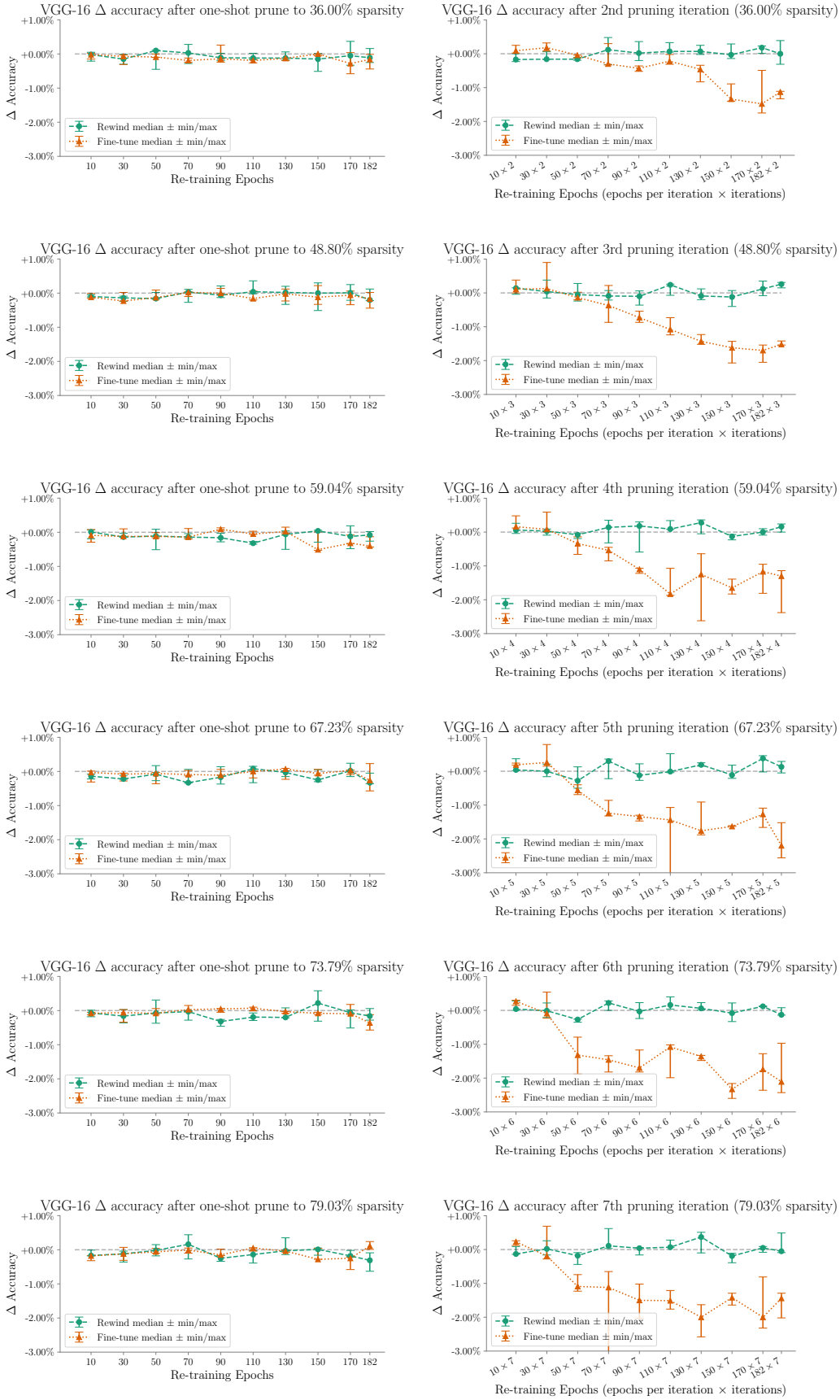
The plots below compare fine-tuning and rewinding across networks and sparsity levels. In these plots, the y-axis is the change in accuracy between the pruned network and the original network (higher is better), and the x-axis is the number of epochs that the pruned network was re-trained for according to Algorithms 1 and 2 (in iterative pruning, the x axis also reflects the iterative nature of iterative pruning – e.g., after 5 iterations of pruning with fine-tuning for 80 epochs, the total number of re-training epochs is 80×5.). For fine-tuning, the x-axis denotes the number of fine-tuning epochs. For rewinding, the x-axis denotes the epoch $T - x$ that the network was rewound to (where T is the original training time of the full network) and the subsequent x epochs of re-training.

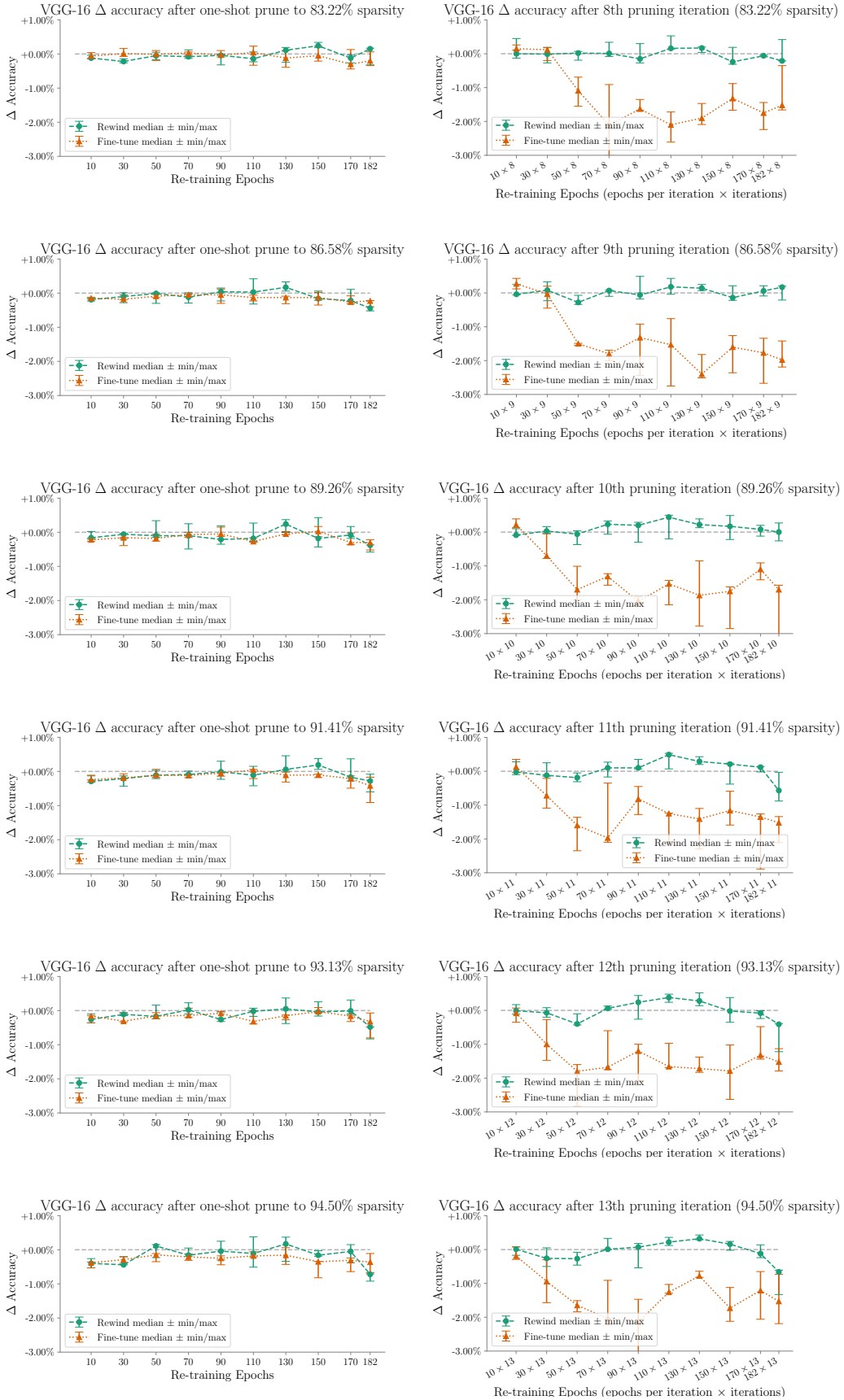
Iterative fine-tuning can overfit One interesting feature of the below graphs is that fine-tuning seems to overfit when ran for too long. Rewinding and replaying seems to be immune to overfitting from gradient descent: rather than accumulating gradients from hundreds or thousands of training epochs, networks pruned with the rewinding/replaying framework only receive gradient updates from a single full training run. While for one-shot pruning this is an easy problem to fix (simply

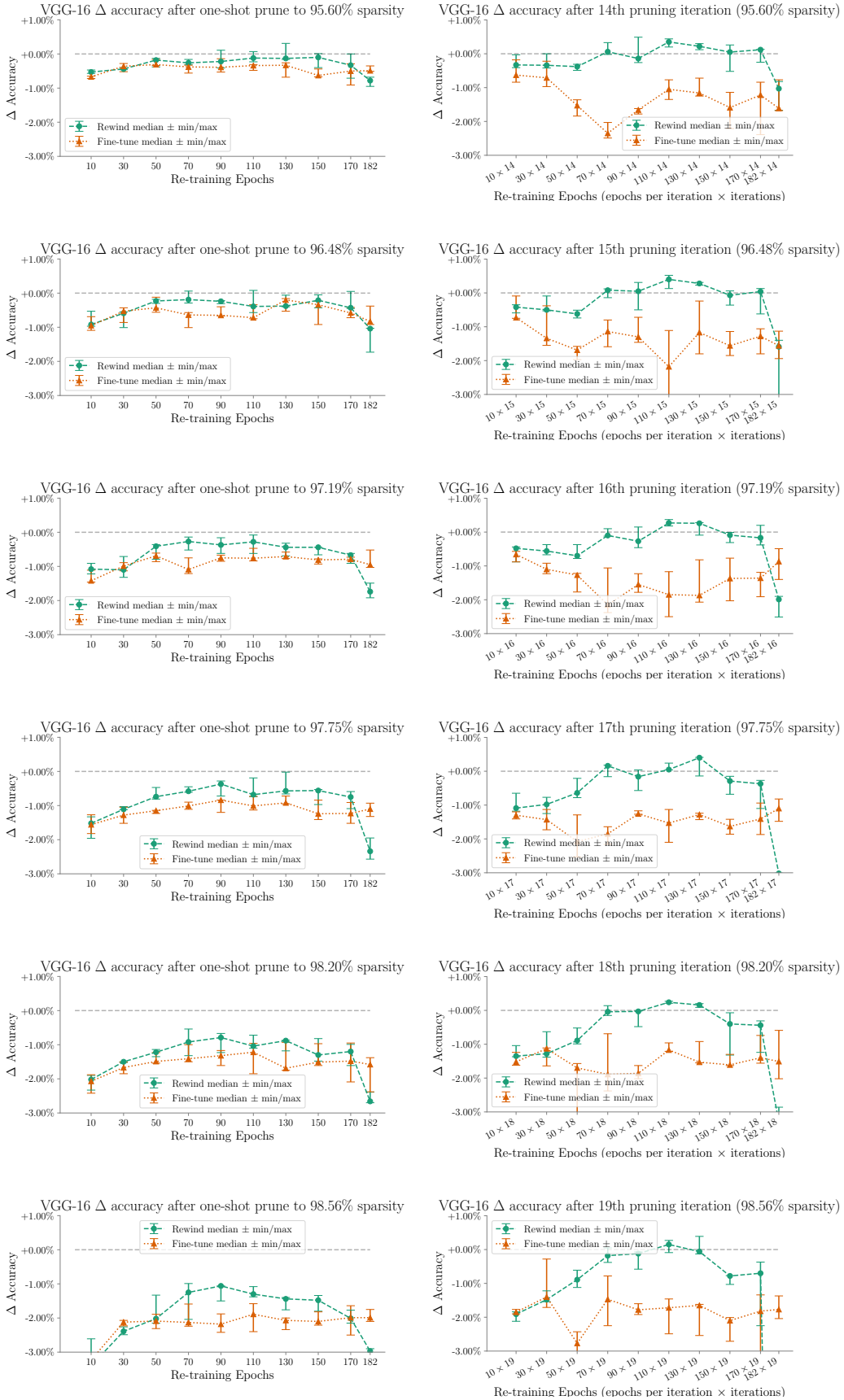
stopping early, or using a high accuracy checkpoint), this is a thornier problem for iterative pruning, since it is not possible to undo extra training iterations from previous iterations in the iterative pruning process by simply restoring from a checkpoint.

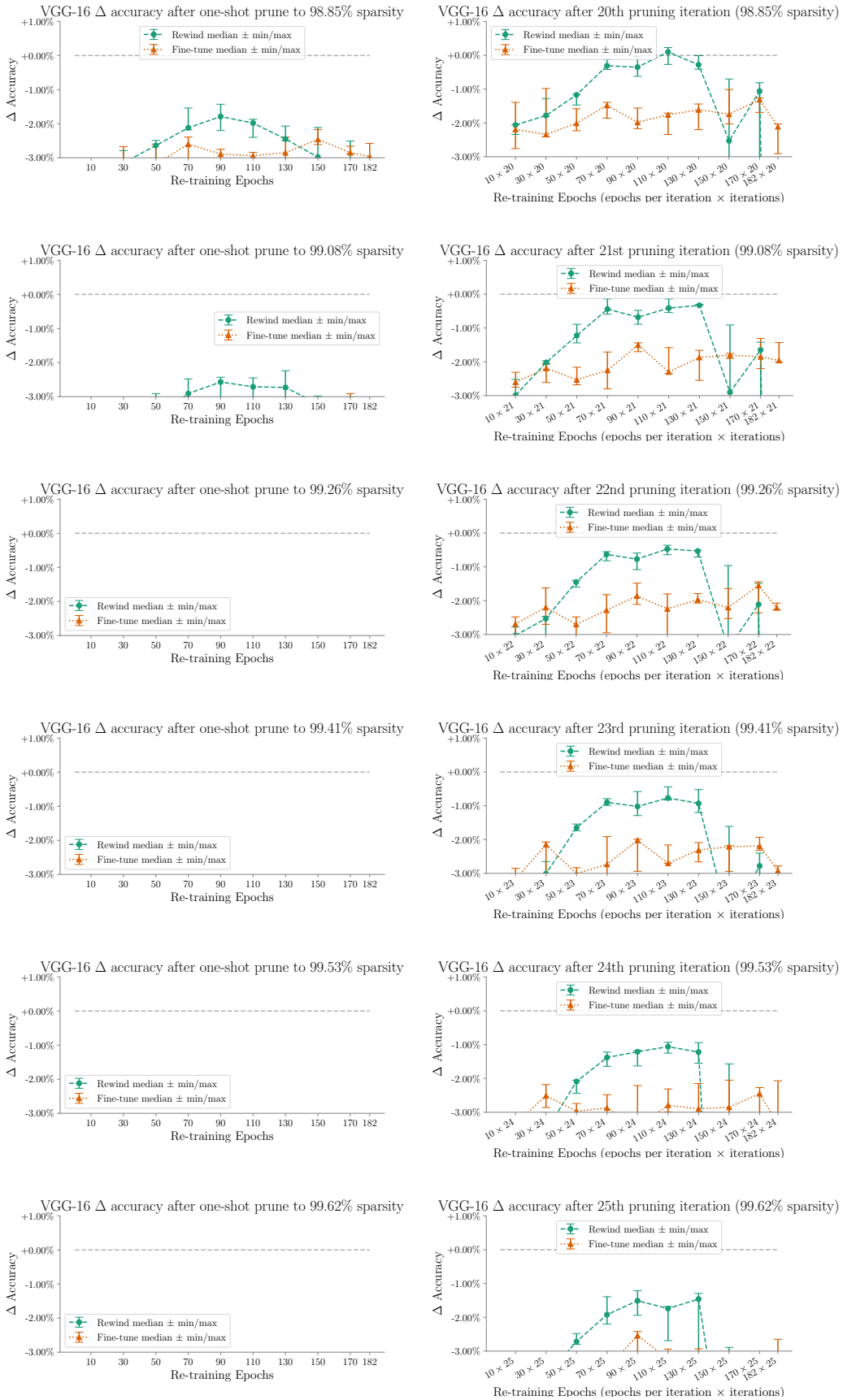


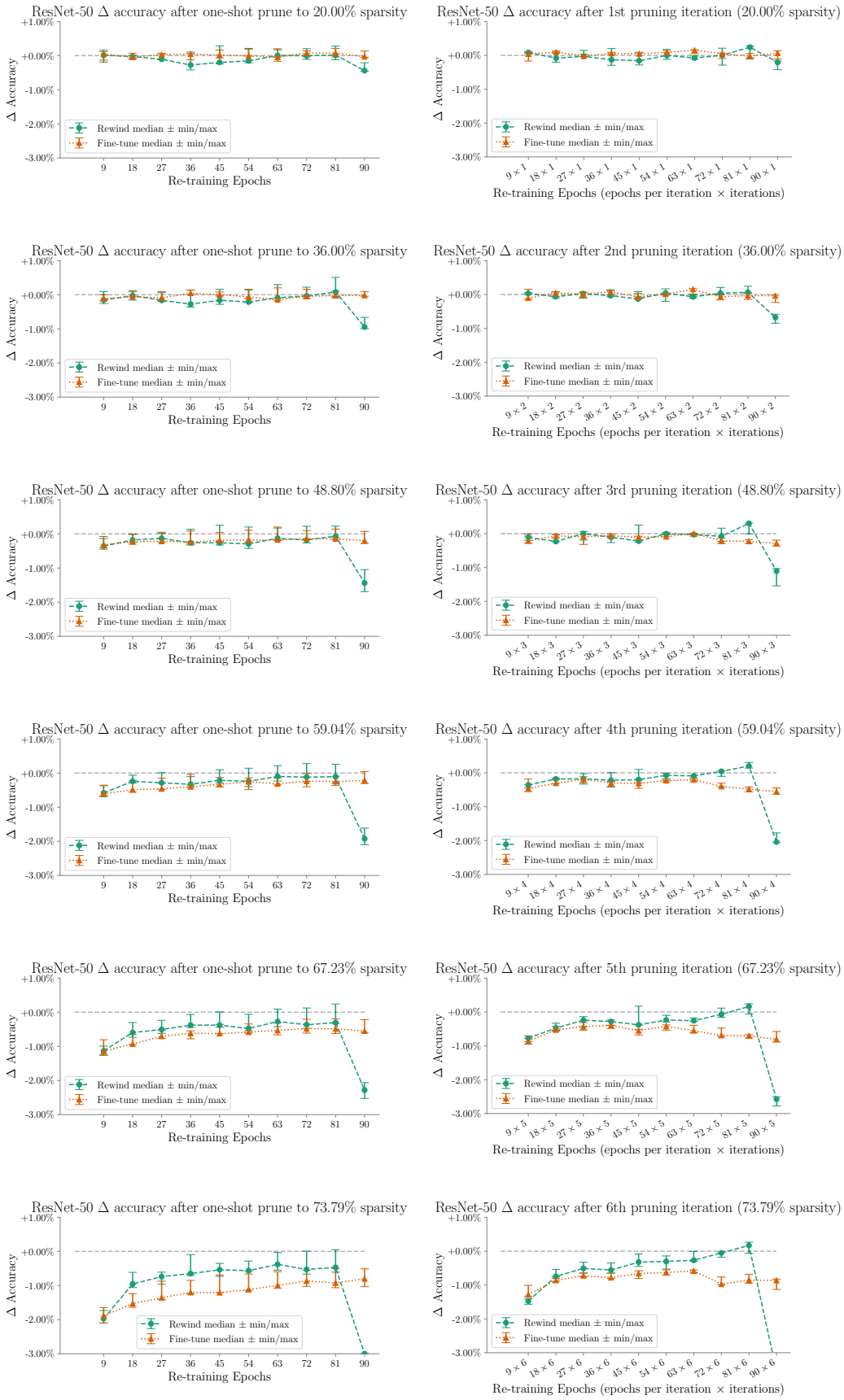

ResNet-20 Δ accuracy after one-shot prune to 73.79% sparsity
ResNet-20 Δ accuracy after 6th pruning iteration (73.79% sparsity)
ResNet-20 Δ accuracy after one-shot prune to 79.03% sparsity
ResNet-20 Δ accuracy after 7th pruning iteration (79.03% sparsity)
ResNet-20 Δ accuracy after one-shot prune to 83.22% sparsity
ResNet-20 Δ accuracy after 8th pruning iteration (83.22% sparsity)
ResNet-20 Δ accuracy after one-shot prune to 86.58% sparsity
ResNet-20 Δ accuracy after 9th pruning iteration (86.58% sparsity)
ResNet-20 Δ accuracy after one-shot prune to 89.26% sparsity
ResNet-20 Δ accuracy after 10th pruning iteration (89.26% sparsity)
VGG-16 Δ accuracy after one-shot prune to 20.00% sparsity
VGG-16 Δ accuracy after 1st pruning iteration (20.00% sparsity)
Δ Accuracy
+1.00%
+0.00%
-1.00%
-2.00%
-3.00%
Rewind median ± min/max
Fine-tune median ± min/max
10 30 50 70 90 110 130 150 170 182
Re-training Epochs
10 × 6 30 × 6 50 × 6 70 × 6 90 × 6 110 × 6 130 × 6 150 × 6 170 × 6 182 × 6
10 × 7 30 × 7 50 × 7 70 × 7 90 × 7 110 × 7 130 × 7 150 × 7 170 × 7 182 × 7
10 × 8 30 × 8 50 × 8 70 × 8 90 × 8 110 × 8 130 × 8 150 × 8 170 × 8 182 × 8
10 × 9 30 × 9 50 × 9 70 × 9 90 × 9 110 × 9 130 × 9 150 × 9 170 × 9 182 × 9
10 × 10 30 × 10 50 × 10 70 × 10 90 × 10 110 × 10 130 × 10 150 × 10 170 × 10 182 × 10
10 × 1 30 × 1 50 × 1 70 × 1 90 × 1 110 × 1 130 × 1 150 × 1 170 × 1 182 × 1
Re-training Epochs (epochs per iteration × iterations)

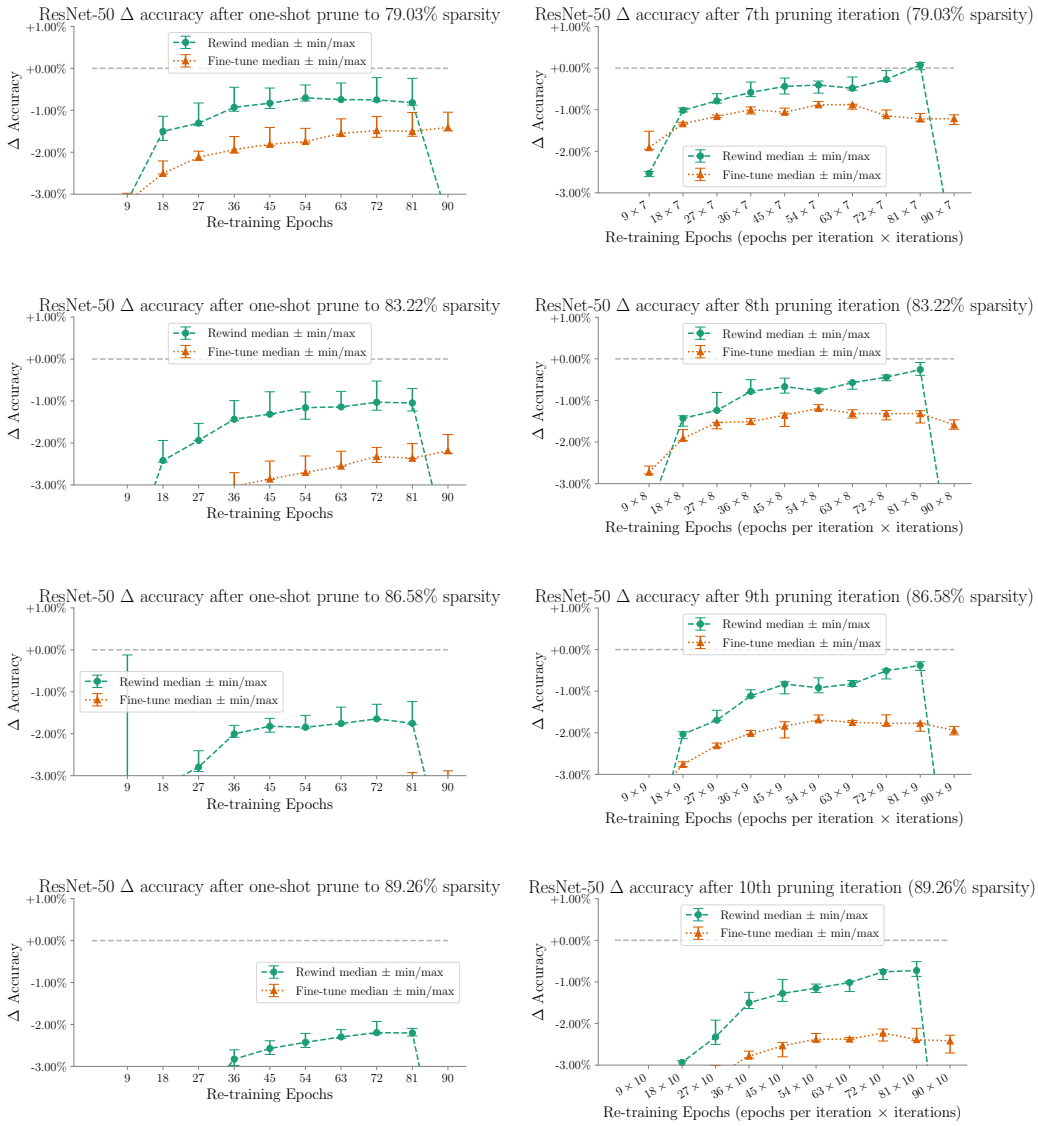

VGG-16 Δ accuracy after one-shot prune to 36.00% sparsity
VGG-16 Δ accuracy after 2nd pruning iteration (36.00% sparsity)
VGG-16 Δ accuracy after one-shot prune to 48.80% sparsity
VGG-16 Δ accuracy after 3rd pruning iteration (48.80% sparsity)
VGG-16 Δ accuracy after one-shot prune to 59.04% sparsity
VGG-16 Δ accuracy after 4th pruning iteration (59.04% sparsity)
VGG-16 Δ accuracy after one-shot prune to 67.23% sparsity
VGG-16 Δ accuracy after 5th pruning iteration (67.23% sparsity)
VGG-16 Δ accuracy after one-shot prune to 73.79% sparsity
VGG-16 Δ accuracy after 6th pruning iteration (73.79% sparsity)
VGG-16 Δ accuracy after one-shot prune to 79.03% sparsity
VGG-16 Δ accuracy after 7th pruning iteration (79.03% sparsity)
Δ Accuracy
+1.00%
+0.00%
-1.00%
-2.00%
-3.00%
Rewind median ± min/max
Fine-tune median ± min/max
10 30 50 70 90 110 130 150 170 182
Re-training Epochs
10 × 2 30 × 2 50 × 2 70 × 2 90 × 2 110 × 2 130 × 2 150 × 2 170 × 2 182 × 2
10 × 3 30 × 3 50 × 3 70 × 3 90 × 3 110 × 3 130 × 3 150 × 3 170 × 3 182 × 3
10 × 4 30 × 4 50 × 4 70 × 4 90 × 4 110 × 4 130 × 4 150 × 4 170 × 4 182 × 4
10 × 5 30 × 5 50 × 5 70 × 5 90 × 5 110 × 5 130 × 5 150 × 5 170 × 5 182 × 5
10 × 6 30 × 6 50 × 6 70 × 6 90 × 6 110 × 6 130 × 6 150 × 6 170 × 6 182 × 6
10 × 7 30 × 7 50 × 7 70 × 7 90 × 7 110 × 7 130 × 7 150 × 7 170 × 7 182 × 7
Re-training Epochs (epochs per iteration × iterations)


VGG-16 Δ accuracy after one-shot prune to 83.22% sparsity
VGG-16 Δ accuracy after 8th pruning iteration (83.22% sparsity)
VGG-16 Δ accuracy after one-shot prune to 86.58% sparsity
VGG-16 Δ accuracy after 9th pruning iteration (86.58% sparsity)
VGG-16 Δ accuracy after one-shot prune to 89.26% sparsity
VGG-16 Δ accuracy after 10th pruning iteration (89.26% sparsity)
VGG-16 Δ accuracy after one-shot prune to 91.41% sparsity
VGG-16 Δ accuracy after 11th pruning iteration (91.41% sparsity)
VGG-16 Δ accuracy after one-shot prune to 93.13% sparsity
VGG-16 Δ accuracy after 12th pruning iteration (93.13% sparsity)
VGG-16 Δ accuracy after one-shot prune to 94.50% sparsity
VGG-16 Δ accuracy after 13th pruning iteration (94.50% sparsity)
Δ Accuracy
+1.00%
+0.00%
-1.00%
-2.00%
-3.00%
Rewind median ± min/max
Fine-tune median ± min/max
10 30 50 70 90 110 130 150 170 182
Re-training Epochs
10 × 8 30 × 8 50 × 8 70 × 8 90 × 8 110 × 8 130 × 8 150 × 8 170 × 8 182 × 8
10 × 9 30 × 9 50 × 9 70 × 9 90 × 9 110 × 9 130 × 9 150 × 9 170 × 9 182 × 9
10 × 10 30 × 10 50 × 10 70 × 10 90 × 10 110 × 10 130 × 10 150 × 10 170 × 10 182 × 10
10 × 11 30 × 11 50 × 11 70 × 11 90 × 11 110 × 11 130 × 11 150 × 11 170 × 11 182 × 11
10 × 12 30 × 12 50 × 12 70 × 12 90 × 12 110 × 12 130 × 12 150 × 12 170 × 12 182 × 12
10 × 13 30 × 13 50 × 13 70 × 13 90 × 13 110 × 13 130 × 13 150 × 13 170 × 13 182 × 13
Re-training Epochs (epochs per iteration × iterations)


VGG-16 Δ accuracy after one-shot prune to 95.60% sparsity
VGG-16 Δ accuracy after 14th pruning iteration (95.60% sparsity)
VGG-16 Δ accuracy after one-shot prune to 96.48% sparsity
VGG-16 Δ accuracy after 15th pruning iteration (96.48% sparsity)
VGG-16 Δ accuracy after one-shot prune to 97.19% sparsity
VGG-16 Δ accuracy after 16th pruning iteration (97.19% sparsity)
VGG-16 Δ accuracy after one-shot prune to 97.75% sparsity
VGG-16 Δ accuracy after 17th pruning iteration (97.75% sparsity)
VGG-16 Δ accuracy after one-shot prune to 98.20% sparsity
VGG-16 Δ accuracy after 18th pruning iteration (98.20% sparsity)
VGG-16 Δ accuracy after one-shot prune to 98.56% sparsity
VGG-16 Δ accuracy after 19th pruning iteration (98.56% sparsity)
Δ Accuracy
+1.00%
+0.00%
-1.00%
-2.00%
-3.00%
Rewind median ± min/max
Fine-tune median ± min/max
10 30 50 70 90 110 130 150 170 182
Re-training Epochs
10 × 14 30 × 14 50 × 14 70 × 14 90 × 14 110 × 14 130 × 14 150 × 14 170 × 14 182 × 14
10 × 15 30 × 15 50 × 15 70 × 15 90 × 15 110 × 15 130 × 15 150 × 15 170 × 15 182 × 15
10 × 16 30 × 16 50 × 16 70 × 16 90 × 16 110 × 16 130 × 16 150 × 16 170 × 16 182 × 16
10 × 17 30 × 17 50 × 17 70 × 17 90 × 17 110 × 17 130 × 17 150 × 17 170 × 17 182 × 17
10 × 18 30 × 18 50 × 18 70 × 18 90 × 18 110 × 18 130 × 18 150 × 18 170 × 18 182 × 18
10 × 19 30 × 19 50 × 19 70 × 19 90 × 19 110 × 19 130 × 19 150 × 19 170 × 19 182 × 19
Re-training Epochs (epochs per iteration × iterations)


VGG-16 Δ accuracy after one-shot prune to 98.85% sparsity
VGG-16 Δ accuracy after 20th pruning iteration (98.85% sparsity)
VGG-16 Δ accuracy after one-shot prune to 99.08% sparsity
VGG-16 Δ accuracy after 21st pruning iteration (99.08% sparsity)
VGG-16 Δ accuracy after one-shot prune to 99.26% sparsity
VGG-16 Δ accuracy after 22nd pruning iteration (99.26% sparsity)
VGG-16 Δ accuracy after one-shot prune to 99.41% sparsity
VGG-16 Δ accuracy after 23rd pruning iteration (99.41% sparsity)
VGG-16 Δ accuracy after one-shot prune to 99.53% sparsity
VGG-16 Δ accuracy after 24th pruning iteration (99.53% sparsity)
VGG-16 Δ accuracy after one-shot prune to 99.62% sparsity
VGG-16 Δ accuracy after 25th pruning iteration (99.62% sparsity)
Rewind median ± min/max
Fine-tune median ± min/max
Δ Accuracy
+1.00%
+0.00%
-1.00%
-2.00%
-3.00%
10 30 50 70 90 110 130 150 170 182
Re-training Epochs
10 × 20 30 × 20 50 × 20 70 × 20 90 × 20 110 × 20 130 × 20 150 × 20 170 × 20 182 × 20
10 × 21 30 × 21 50 × 21 70 × 21 90 × 21 110 × 21 130 × 21 150 × 21 170 × 21 182 × 21
10 × 22 30 × 22 50 × 22 70 × 22 90 × 22 110 × 22 130 × 22 150 × 22 170 × 22 182 × 22
10 × 23 30 × 23 50 × 23 70 × 23 90 × 23 110 × 23 130 × 23 150 × 23 170 × 23 182 × 23
10 × 24 30 × 24 50 × 24 70 × 24 90 × 24 110 × 24 130 × 24 150 × 24 170 × 24 182 × 24
10 × 25 30 × 25 50 × 25 70 × 25 90 × 25 110 × 25 130 × 25 150 × 25 170 × 25 182 × 25
Re-training Epochs (epochs per iteration × iterations)


ResNet-50 Δ accuracy after one-shot prune to 20.00% sparsity
ResNet-50 Δ accuracy after 1st pruning iteration (20.00% sparsity)
ResNet-50 Δ accuracy after one-shot prune to 36.00% sparsity
ResNet-50 Δ accuracy after 2nd pruning iteration (36.00% sparsity)
ResNet-50 Δ accuracy after one-shot prune to 48.80% sparsity
ResNet-50 Δ accuracy after 3rd pruning iteration (48.80% sparsity)
ResNet-50 Δ accuracy after one-shot prune to 59.04% sparsity
ResNet-50 Δ accuracy after 4th pruning iteration (59.04% sparsity)
ResNet-50 Δ accuracy after one-shot prune to 67.23% sparsity
ResNet-50 Δ accuracy after 5th pruning iteration (67.23% sparsity)
ResNet-50 Δ accuracy after one-shot prune to 73.79% sparsity
ResNet-50 Δ accuracy after 6th pruning iteration (73.79% sparsity)
Δ Accuracy
+1.00%
+0.00%
-1.00%
-2.00%
-3.00%
Rewind median ± min/max
Fine-tune median ± min/max
9 18 27 36 45 54 63 72 81 90
Re-training Epochs
9 × 1 18 × 1 27 × 1 36 × 1 45 × 1 54 × 1 63 × 1 72 × 1 81 × 1 90 × 1
9 × 2 18 × 2 27 × 2 36 × 2 45 × 2 54 × 2 63 × 2 72 × 2 81 × 2 90 × 2
9 × 3 18 × 3 27 × 3 36 × 3 45 × 3 54 × 3 63 × 3 72 × 3 81 × 3 90 × 3
9 × 4 18 × 4 27 × 4 36 × 4 45 × 4 54 × 4 63 × 4 72 × 4 81 × 4 90 × 4
9 × 5 18 × 5 27 × 5 36 × 5 45 × 5 54 × 5 63 × 5 72 × 5 81 × 5 90 × 5
9 × 6 18 × 6 27 × 6 36 × 6 45 × 6 54 × 6 63 × 6 72 × 6 81 × 6 90 × 6
Re-training Epochs (epochs per iteration × iterations)


ResNet-50 Δ accuracy after one-shot prune to 79.03% sparsity
ResNet-50 Δ accuracy after 7th pruning iteration (79.03% sparsity)
ResNet-50 Δ accuracy after one-shot prune to 83.22% sparsity
ResNet-50 Δ accuracy after 8th pruning iteration (83.22% sparsity)
ResNet-50 Δ accuracy after one-shot prune to 86.58% sparsity
ResNet-50 Δ accuracy after 9th pruning iteration (86.58% sparsity)
ResNet-50 Δ accuracy after one-shot prune to 89.26% sparsity
ResNet-50 Δ accuracy after 10th pruning iteration (89.26% sparsity)
Rewind median ± min/max
Fine-tune median ± min/max
Δ Accuracy
+1.00%
+0.00%
-1.00%
-2.00%
-3.00%
Re-training Epochs
Re-training Epochs (epochs per iteration × iterations)