THE GENERALIZATION-STABILITY TRADEOFF IN NEURAL NETWORK PRUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Pruning neural network parameters is often viewed as a means to compress models, but pruning has also been motivated by the desire to prevent overfitting. This motivation is particularly relevant given the perhaps surprising observation that a wide variety of pruning approaches *increase* test accuracy despite sometimes massive reductions in parameter counts. To better understand this phenomenon, we analyze the behavior of pruning over the course of training, finding that pruning's effect on generalization relies more on the instability it generates (defined as the drops in test accuracy immediately following pruning) than on the final size of the pruned model. We demonstrate that even the pruning of unimportant parameters can lead to such instability, and show similarities between pruning and regularizing by injecting noise, suggesting a mechanism for pruning-based generalization improvements that is compatible with the strong generalization recently observed in over-parameterized networks.

1 INTRODUCTION

Pruning weights and/or convolutional filters from deep neural networks (DNNs) can substantially shrink parameter counts with minimal loss in accuracy (LeCun et al., 1990; Hassibi & Stork, 1993; Han et al., 2015a; Li et al., 2016; Molchanov et al., 2017; Louizos et al., 2017; Liu et al., 2017; Ye et al., 2018), enabling broader application of DNNs via reductions in memory-footprint and inference-FLOPs requirements. Moreover, many pruning methods have been found to actually improve generalization (measured by model accuracy on previously unobserved inputs) (Narang et al., 2017; Frankle & Carbin, 2018; You et al., 2019). Consistent with this, pruning was originally motivated as a means to prevent over-parameterized networks from overfitting to comparatively small datasets (LeCun et al., 1990).

Concern about over-parameterizing models has weakened, however, as many recent studies have found that *adding* parameters can actually *reduce* a DNN's generalization-gap (the drop in performance when moving from previously seen to previously unseen inputs), even though it has been shown that the same networks have enough parameters to fit large datasets of randomized data (Neyshabur et al., 2014; Zhang et al., 2016). Potential explanations for this unintuitive phenomenon have come via experiments (Keskar et al., 2016; Morcos et al., 2018; Yao et al., 2018; Belkin et al., 2018; Nagarajan & Kolter, 2019), and the derivation of bounds on DNN generalization-gaps that suggest less overfitting occurs as parameter counts increase (Neyshabur et al., 2018). This research has implications for neural network pruning, where a puzzling question has arisen: if larger parameter counts don't increase overfitting, how does pruning parameters improve generalization?

To address this question we first introduce the notion of pruning instability (Section 3), which, intuitively, is the size of the change in network outputs caused by a pruning iteration. We then empirically analyze the instability and generalization associated with various magnitude-pruning (Han et al., 2015b) algorithms in different settings, making the following contributions:

1. We find a tradeoff between the stability and potential generalization benefits of pruning, and show pruning's similarity to regularizing with noise—suggesting a mechanism unrelated to parameter counts through which pruning appears to affect generalization.
2. We characterize the properties of pruning algorithms which lead to instability and correspondingly higher generalization.

3. We derive a batch-normalized-parameter pruning algorithm to better control pruning stability.

2 Related Work

There are various approaches to pruning neural networks. Pruning may be performed post-hoc (LeCun et al., 1990; Hassibi & Stork, 1993; Han et al., 2015b; Liu et al., 2017), or iteratively throughout training, such that there are multiple pruning events as the model trains (Hochreiter & Schmidhuber, 1997; Narang et al., 2017; Zhu & Gupta, 2017). Most methods prune parameters that appear unimportant to the function computed by the neural network, though means of identifying importance vary. Magnitude pruning (Han et al., 2015b) uses small-magnitude to indicate unimportance and has been shown to perform competitively with more sophisticated approaches (Gale et al., 2019).

Many pruning studies have shown that the pruned model has heightened generalization (Narang et al., 2017; Frankle & Carbin, 2018; You et al., 2019), consistent with the fact that pruning may be framed as a regularization (rather than compression) approach. For example, variational Bayesian approaches to pruning via sparsity-inducing priors (Molchanov et al., 2017; Louizos et al., 2017) can describe weight removal as a process that reduces model description length, which in theory may help improve generalization (Rissanen, 1978). Similarly, the idea that models may be described more succinctly at flat minima has motivated pruning in service of flat minimum search (Hochreiter & Schmidhuber, 1997). Though Dinh et al. (2017) notes, however, that flatness can be arbitrarily modified by reparameterizing the function, and sharp minima can generalize well.

VC dimension (a measure of model capacity) has motivated the use of pruning to improve generalization as well (LeCun et al., 1990; Hassibi & Stork, 1993). Overfitting can be bounded above by an increasing function of VC dimension, which itself often increases with parameter counts, so fewer parameters can lead to a guarantee of less overfitting (Shalev-Shwartz & Ben-David, 2014). Unfortunately, such bounds can be so loose in practice that tightening them by reducing parameter counts need not translate to better generalization (Dziugaite & Roy, 2017).

Rather than support parameter-count-based arguments for generalization in DNNs, our results suggest pruning may improve generalization by creating various noisy versions of the internal representation of the data, which unpruned parameters try to fit to, as in noise-injection regularization (Srivastava et al., 2014; Poole et al., 2014). Dropout seems to create particularly similar noise, as it temporarily sets random subsets of layer outputs to zero (likely changing a given input's internal representation every epoch). Indeed, applying dropout-like zeroing noise to a subset of features during training can encourage robustness to a post-hoc pruning of that subset (Leclerc et al., 2018; Gomez et al., 2018). Pruning noise ultimately differs, however, as it is: applied less frequently, not temporary (except in algorithms that allow some weight re-entry), usually not random, and less well studied.

3 Approach

Given a neural network and set of test data, we define a pruning algorithm's *instability* on pruning iteration i in terms of the neural network's top-1 test accuracy before ($\mathrm{t_{pre}}$) and after ($\mathrm{t_{post}}$) pruning: $\text{instability}_i = |\mathrm{t_{pre}} - \mathrm{t_{post}}|$.

This measure is related to a weight's importance (sometimes referred to as "saliency"; LeCun et al. (1990); Hassibi & Stork (1993)) to the test accuracy, in that less stable pruning algorithms target more important sets of weights (all else equal). The stability of a pruning algorithm may be affected by many factors. Our experiments (Section 4) explore the effects of the following: pruning target, pruning schedule, iterative pruning rate, and model. The remainder of this section provides an overview of these factors and demonstrates a need for a novel pruning target, which we derive.

3.1 Pruning Target

In all of our experiments, we use iterative magnitude pruning (Han et al., 2015b), which removes weights according to some magnitude-based rule, retrains the resulting smaller network to recover from the pruning, and repeats until the desired size reduction is met. We denote pruning algorithms that target the smallest-magnitude parameters with an "S" subscript (e.g. prune_S), random parameters with an "R" subscript, and the largest-magnitude parameters with an "L" subscript. The usual

approach to pruning involves removing parameters that have the smallest magnitudes (Li et al., 2016; Gale et al., 2019), or, similarly, those parameters least important to the loss function as determined by some other metric (LeCun et al., 1990; Hassibi & Stork, 1993; Molchanov et al., 2016; 2017; Louizos et al., 2017; Ye et al., 2018; Yu et al., 2018; You et al., 2019).

3.1.1 IDENTIFYING IMPORTANT BATCH-NORMALIZED PARAMETERS

The correlation between parameter magnitude and importance weakens in the presence of batch normalization (BN) (Ioffe & Szegedy, 2015). Without batch normalization, a convolutional filter with weights W will produce feature map activations with half the magnitude of a filter with weights $2W$: filter magnitude clearly scales the output. With batch normalization, however, the feature maps are normalized to have zero mean and unit variance, and their ultimate magnitudes depend on the BN affine-transformation parameters γ and β. As a result, in batch normalized networks, filter magnitude does not scale the output, and equating small magnitude and unimportance may therefore be particularly flawed. This has motivated approaches to use the scale parameter γ's magnitude to find the convolutional filters that are important to the network's output (Ye et al., 2018; You et al., 2019).

In Appendix A.1, we derive a novel estimate of filter importance/magnitude that incorporates both γ and β. Aside from enabling a closer approximation to BN output magnitude, accounting for β is necessary to identify filters that produce meaningful, post-ReLU activations (i.e., a negative β could cause a large portion of the output to not pass through the ReLU nonlinearity).

We use the phrase "*E[BN] pruning*" to denote magnitude pruning that computes magnitude using this derived estimate. E[BN] pruning has two advantages. First, this approach avoids the problematic assumption that filter importance is tied to filter ℓ_2 norm in a batch-normalized network. Accordingly, we hypothesize that E[BN] pruning can grant better control of the stability of the neural network's output than pruning based on filters' ℓ_2 norms. Second, the complexity of the calculation is negligible as it requires (per filter) just a handful of arithmetic operations on scalars, and two PDF and CDF evaluations, which makes it cheaper than a data-driven approach (e.g. approximating the expected value via the sample mean of feature map activations for a batch of feature maps).

3.2 SUMMARY OF MODELS

We consider three basic model classes: a simple network with convolutions (2x32, pool, 2x64, pool) and fully connected layers (512, 10) that we denote Conv4, VGG11 (Simonyan & Zisserman, 2014) with its fully-connected layers replaced by a single fully-connected layer, and ResNet18 (He et al., 2016). All convolutions are 3x3. We trained these models using Adam (Kingma & Ba, 2014) with initial learning rate $lr = 0.001$, as we found Adam more helpful than SGD for recovering from unstable pruning (seemingly consistent with the observation in Zhu & Gupta (2017) that recovery from pruning is more difficult when learning rates are low).

3.3 ITERATIVE PRUNING RATE AND SCHEDULE

For Conv4, we apply pruning to its first linear layer (which contains 94% of Conv4's 1,250,858 parameters). For VGG/ResNet, pruning targets the final four convolutional layers (which contain 90% of VGG11's 9,231,114 parameters, and 74% of ResNet18's 11,173,962 parameters). Pruning focused on later layers partly because, as also found in Li et al. (2016); You et al. (2019), it allowed the network to recover more easily. Unless noted otherwise, the total pruning percentages were: 42% of VGG11, 46% of ResNet18, and 10% of Conv4.

The pruning algorithms we consider are iterative: we define a pruning schedule that describes the epochs on which pruning events occur, and set a corresponding (constant) iterative pruning rate that will ensure the total pruning percentage is met by the end of training (please see Appendix A.3 for rate and schedule details). Thus, throughout training, pruning steadily removes DNN parameters, with the iterative pruning rate determining the number pruned per event.

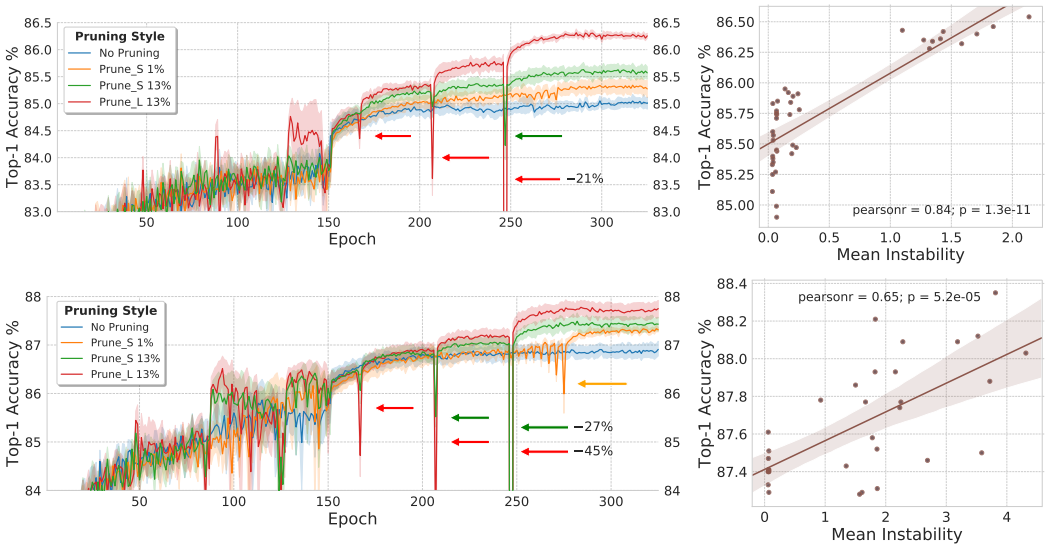

Figure 1: Pruning instability improves generalization of (Top) VGG11 and (Bottom) ResNet18 when training on CIFAR-10 (10 runs per configuration). (Left) Test accuracy during training of several models illustrates how adaptation to more unstable pruning leads to better generalization. (Right) Means reduce along the epoch dimension (creating one point per run-configuration combination).

4 Experiments

Pruning studies often aim to compress pre-trained models that generalize well, and consequently, much work has focused on metrics to identify parameter importance: if you can find the parameters that matter the least to the function computed by the DNN, then you can prune more parameters without significantly harming accuracy. As a bonus, such pruning methods can sometimes even *increase* generalization (Narang et al., 2017; Frankle & Carbin, 2018; You et al., 2019). However, the mechanism by which pruning induces higher generalization remains unknown. Here, rather than investigate how to best maintain accuracy when pruning the network, we instead focus on understanding the mechanisms underlying these generalization improvements.

4.1 The Generalization-Stability Tradeoff

Can improved generalization in pruned DNNs be explained by parameter-count reduction alone, or rather, do the properties of the pruning algorithm play an important role in generalization? As removing parameters from a DNN via pruning may make the DNN less capable of fitting to the noise in the training data, as originally suggested in LeCun et al. (1990); Hassibi & Stork (1993), we might expect that the generalization improvements observed in pruned DNNs are entirely explained by the number of parameters removed. In which case, methods that prune equal amounts of parameters would generalize similarly.

Alternatively, perhaps some aspect of the pruning algorithm itself is responsible for increased generalization. This seems plausible as the reported generalization benefits of pruning vary widely across studies. One possible explanation for this variability is differences in the pruning algorithms themselves. A key differentiator of these algorithms is their stability: more stable approaches may compute a very close approximation to the way the loss changes with respect to each parameter and prune a single parameter at a time (Hassibi & Stork, 1993), while less stable approaches may assume that parameter magnitude and importance are roughly similar and prune many weights all at once (Han et al., 2015b). Therefore, to the extent that differences in pruning algorithms explain differences in pruning-based generalization improvements, we might expect to observe a relationship between generalization and pruning stability.

To determine whether pruning algorithm stability affects generalization, we compared the instability and final top-1 test accuracy of several pruning algorithms with varying methods for estimating

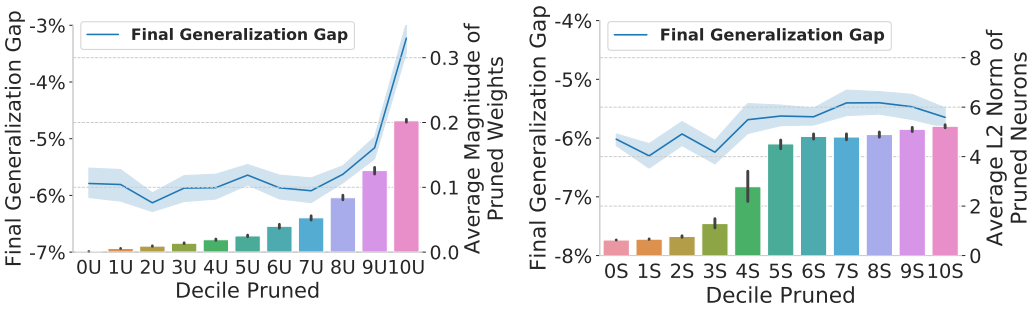

Figure 2: When pruning 10% of Conv4's largest dense layer, the final generalization gap depends on the magnitude of the weights that were pruned during training. This is particularly true when using unstructured pruning (left) rather than structured pruning (right).

importance and iterative pruning rates (Figure 1). Consistent with the nature of the pruning algorithm playing a role in generalization, we observed that more unstable pruning algorithms created *higher* final test accuracies than those which were stable (Figure 1, right; VGG11: Pearson's correlation $r = .84$, p-value $= 1.3\text{e}{-11}$; ResNet18: $r = .65$, p-value $= 5.2\text{e}{-5}$). While many pruning approaches have aimed to induce as little instability as possible, these results suggest that pruning techniques may actually facilitate better generalization when they induce *more* instability. Furthermore, these results suggest that parameter-count based arguments may not be sufficient to explain generalization in pruned DNNs, and suggest that the precise pruning method plays a critical role in this process.

Figure 1 also demonstrates that pruning events for prune_L with a high iterative pruning rate (red curve, pruning as much as 13% of a given convolutional layer per pruning iteration) are substantially more destabilizing than other pruning events, but despite the dramatic pruning-induced drops in performance, the network recovers to higher performance within a few epochs. Several of these pruning events are highlighted with red arrows. Please see Appendix A.2 for visualization of the epoch-wise instabilities of each method in VGG11.

Interestingly, we initially observed that ResNet18 adapted to pruning events more quickly than VGG11 (accuracy rebounded after pruning then flattened soon after instead of climbing steadily). Thinking that shortcut connections were allowing the network to adapt to pruning events too easily, we tried pruning a larger amount of the penultimate block's output layer: this reduced the number of shortcut connections to the final block's output layer, lengthened the adaptation period, and improved generalization. This simple improvement of pruning hyperparameters suggests a potential for further optimization of the results shown. Please see Appendix A.3.1 for all hyperparameters/details of these experiments.

4.2 The Role of Weight Magnitude in Pruning Regularization

We have demonstrated that, perhaps surprisingly, pruning larger magnitude weights via the E[BN] algorithm can result in *larger* test accuracy improvements (Figure 1). This suggests a positive correlation between pruning target magnitude and pruning's regularization effect. However, it's not clear whether this relationship holds more generally; i.e., perhaps it was caused by a feature of our E[BN] algorithm or the networks examined. Alternatively, this effect may be dependent on whether nodes/filters (structured pruning) or individual parameters (unstructured pruning) are pruned.

As such, we tested whether target weight magnitude correlates with pruning's regularizing effect when using both unstructured and structured magnitude pruning on the penultimate linear layer of a small network without batch normalization (Conv4). Specifically, we constructed a pruning target for each weight-magnitude decile (see Appendix A.3.2 for details), used each target to prune ten separate networks as they trained, and compared the generalization gaps (test-train accuracy) of the pruned networks to the target pruned (Figure 2).

For both unstructured and structured pruning (Figure 2 left and right, respectively), we found that pruning larger weights led to better generalization gaps, though, interestingly, this effect was much more dramatic in the context of unstructured pruning than structured pruning. One possible

explanation for this is that, in structured pruning, the ℓ_2 norm of pruned neurons did not vary dramatically past the fifth decile, whereas the unstructured deciles were approximately distributed exponentially. As a result, the top 50% of filters for the structured case were not clearly distinguished, making magnitude pruning much more susceptible to small sources of noise. These results suggest that, when weight magnitudes vary considerably, pruning large magnitude weights may lead to improved generalization.

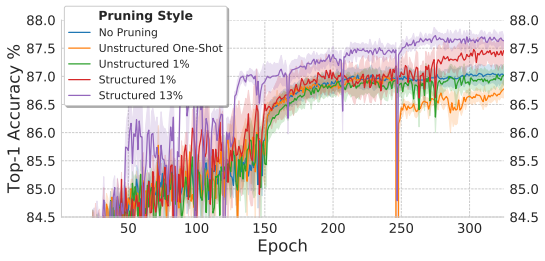

Figure 3: The top-1 test accuracy during training with multiple approaches to pruning ResNet18.

Interestingly, for ResNet18, we actually found that *structured* prune_L (red line) performed noticeably better than unstructured prune_L (green line) (Figure 3). The worse performance of unstructured prune_L may stem from its harming the helpful inductive bias provided by convolutional filters; i.e., perhaps removing the most important connections in all convolutional filters degrades performance more than pruning the same number of connections via removal of several entire filters. Note that these experiments are not directly comparable to Figure 1, as they employ a different set of layerwise pruning percentages (see Appendix A.3.3 for experimental setup information).

4.3 The Role of Iterative Pruning Rate in Pruning Instability

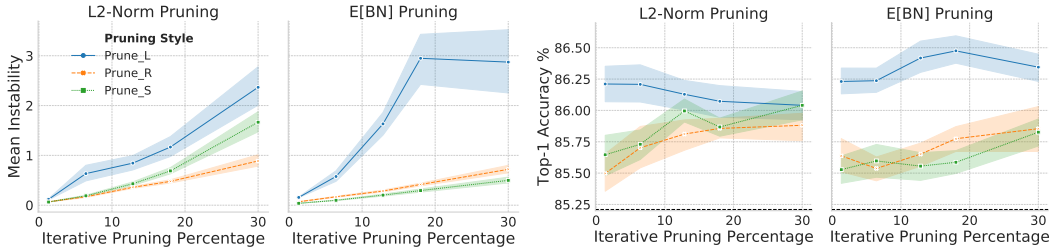

Figure 4: In VGG11, increasing the iterative pruning rate (and decreasing the number of pruning events in order to hold total pruning percentage constant) leads to more instability, and can allow methods that target less important parameters to generalize better. Additionally, E[BN] magnitude better approximates parameter importance than ℓ_2-norm magnitude (see Figure A2 for another example and discussion of this phenomenon). An unpruned baseline is shown on the dotted line.

While pruning large magnitude weights appears to play a role in pruning's ability to improve generalization, more commonly used pruning algorithms often see generalization improvements when targeting the smallest magnitude or least important parameters, suggesting that target magnitude/importance is not the only characteristic of pruning algorithms relevant to generalization. One possibility is that, given a pruning target, pruning more parameters per pruning iteration (while holding constant the total pruning percentage) may lead to greater instability. If this is the case, we would expect that increasing the iterative pruning rate should increase instability and, consequently, generalization performance. Alternatively, if the final pruning fraction is all that matters, we would expect that changing the iterative pruning rate while keeping the final pruning fraction fixed should have no effect on generalization.

To test this, we plotted mean instability and test accuracy as a function of different iterative pruning rates for both ℓ_2-norm and E[BN] pruning (Figure 4). Consistent with iterative pruning rate playing a role in instability, we find that (given a pruning target) more instability is induced by using larger iterative pruning rates (Figure 4 left). Moreover, pruning random or small magnitude parameters performs best at the largest iterative rate (30%), supporting the idea that these methods require a source of instability to boost generalization. This suggests that, when targeting less important weights, higher iterative pruning rates during training can be an effective way to induce additional instability and generalization.

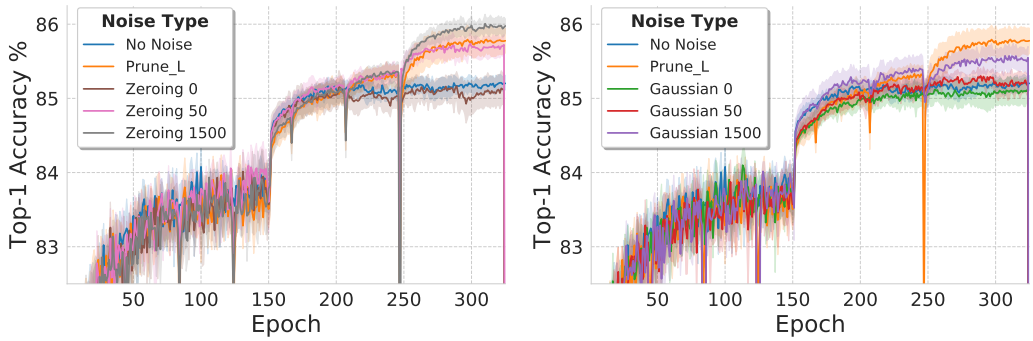

Figure 5: Generalization improvements from pruning bear resemblance to those obtained by using temporary (Left) multiplicative zeroing noise, and (Right) additive Gaussian noise, as long as the noise is applied for enough batches/steps.

We also found evidence that ℓ_2-norm pruning adds instability in networks with batch normalization at least, seemingly by failing to identify important weights properly. While E[BN] magnitude pruning led to the expected ordering of instabilities (e.g. pruning random weights is less stable than pruning small weights), standard ℓ_2-norm magnitude pruning led to small-weight pruning being *more* unstable than random-weight pruning, and almost as unstable as large-weight pruning. These results reinforce that the precise choice of pruning algorithm can have a dramatic effect on the resultant instability and test performance. (Algorithm and experiment details are available in Appendix A.3.4.)

Perhaps strangely, higher iterative pruning rates did not necessarily translate to improved generalization when targeting the largest magnitude weights (prune_L). The fact that prune_L does not generalize the best at the highest iterative pruning rate may be due to the reduction in pruning iterations required by the large iterative pruning rate (i.e., when the iterative rate is at 30%, the number of pruning events is capped at three, which removes 90% of a layer). Thus, while this rate is more unstable (Figure 4 left) per iteration, pruning affects the network less often. This suggests that the regularizing effect of pruning may be enhanced by pruning more often, providing a potential explanation for the observation that methods that prune iteratively can generalize better (Han et al., 2015b).

4.4 Pruning as Noise Injection

Our results thus far suggest that the generalization benefits from pruning result from pruning generated instability, which is affected by a number of related factors, including pruning target, iterative pruning rate, and the frequency of pruning. However, for all of our prior results, we also damaged model capacity simply by the nature of pruning, which decreases the number of model parameters. It therefore remains possible that the generalization benefits we see rely upon the reduction in capacity conferred by pruning. Here, we examine this critical question.

We first note that pruning can be viewed as noise injection (Srivastava et al., 2014; Poole et al., 2014), with the peculiarity that the noise is permanent. Because this permanence is what leads to the capacity of the network being diminished, removing this aspect from our pruning algorithms while holding everything else constant allows us to study pruning's role as a noise injection algorithm while maintaining model capacity.

As a baseline, we consider prune_L applied to VGG11, judging filter magnitude via the ℓ_2-norm (additional experimental details are in Appendix A.3.5). We then modify this algorithm such that, rather than permanently prune filters, it simply multiplies the filter weights by zero, allowing the zeroed weights to immediately resume training in the network ("Zeroing 0" in Figure 5 Left). However, by allowing pruned weights to immediately recover, this experiment also removes a key, potentially regularizing aspect of pruning noise: the requirement that the rest of the network adapts to fit the new representations generated by pruning. To encourage this potentially important facet of pruning noise, we also added variants that held weights to zero for 50 and 1500 consecutive batches. As a related experiment, we also measured the impact of adding Gaussian noise to the weights in Figure 5, right.

Noise was applied either once (Gaussian 0) or repeatedly over a series of training batches (Gaussian 50/1500).

If capacity effects are not necessary to explain pruning's role in generalization, then we would expect that the generalization behavior of these non-permanent noise injection algorithms could mimic the generalization behavior of prune_L. Alternatively, if capacity reduction is a necessary component of pruning-based generalization improvements, then we would not expect close similarities between the generalization phenomena of prune_L and non-permanent pruning noise injection.

Consistent with capacity not being necessary to explain generalization in pruned DNNs, applying zeroing noise for 50 batches to filters (rather than pruning them completely) generates strikingly similar accuracy to prune_L (Figure 5 Left). Specifically, the patterns in instability are qualitatively and quantitatively similar, as are the generalization levels throughout training.

Importantly, we found that applying zeroing noise once (Zeroing 0; brown line) was not sufficient to generate better performance, suggesting that the regularization induced by forcing weights to adapt to noised representations is critical to pruning's ability to improve generalization. Moreover, we found that, while applying Gaussian noise could increase generalization if applied for long enough (Gaussian 1500; purple line), it still did not match the performance of prune_L, suggesting that multiplicative zeroing noise is substantially more effective than additive Gaussian noise[1]. Together, these results demonstrate that pruning induced generalization benefits are not merely an effect of reduced capacity, but rather are dependent on the regularization conferred by forcing networks to adapt to noised representations over a sufficiently long period throughout training.

5 Discussion

In this study, we defined the notion of pruning algorithm instability, and applied several pruning approaches[2] to multiple neural networks, assessing the approaches' effects on instability and generalization. Throughout these experiments, we observed that pruning algorithms that generated more instability led to better generalization (as measured by test accuracy). For a given pruning target and total pruning percentage, instability and generalization could be fueled by raising iterative pruning rates (Figure 4, Section 4.3). Additionally, targeting more important weights, again holding total parameters pruned constant, led to more instability and generalization than targeting less important weights (Figure 1, Section 4.1).

These results support the idea that the generalization benefits of pruning cannot be explained solely by pruning's effect on parameter counts—the properties of the pruning algorithm must be taken into account. Our analysis also suggests that parameter-count effects may not even be *necessary* to explain how pruning improves generalization. Indeed, we provide an interpretation of pruning as noise injection, a popular approach to regularizing DNNs, and find that making pruning noise impermanent provides pruning-like generalization benefits without removing parameters from the model (Figure 5, Section 4.4).

5.1 Caveats and Future Work

While not emphasized in our discussion, pruning algorithm *stability* can be a desirable property, as recovery from pruning damage is not guaranteed. Indeed, pruning too many large/important weights can lead to worse final generalization (Li et al., 2016). Recovery appears to be a function of several factors, including: learning rate (Zhu & Gupta, 2017)); presence of an ongoing regularization effect (Figure 3, Section 4.2); preservation of helpful inductive biases (Figure 3, Section 4.2); and damage to network capacity (e.g., removing too much of an important layer could cause underfitting).

A better understanding of the factors which aid recovery from pruning instability could aid the design of novel pruning algorithms. For example, pruning methods that allow weights to re-enter the network (Narang et al., 2017) could perhaps prune *important* weights occasionally to enhance generalization improvements, without risking permanent damage to the pruned networks.

[1] At the scale we used.

[2] Including our own novel pruning approach, described in Section 3.1.1, which was designed to better control pruning stability.

In describing how pruning regularizes a model, we touched on similarities between pruning and noise injection. Our results, however, may also be consistent with other parameter-count-independent approaches to understanding generalization in neural networks, as pruning may reduce the information stored in the network's weights (Hinton & Van Camp, 1993), and make the network more distributed (Morcos et al., 2018; Dettmers & Zettlemoyer, 2019). This raises the possibility that pruning noise engenders helpful properties in DNNs, though it remains unclear whether such properties might be identical to those achieved with more common noise injection schemes (Srivastava et al., 2014; Poole et al., 2014). Further exploration will be necessary to better understand the relationship between these approaches.

One important caveat of our results is that they were generated with CIFAR-10, a relatively small dataset, so future work will be required to evaluate whether the presented phenomena hold in larger datasets. Relatedly, we only studied pruning's regularizing effect in isolation and did not include commonly used regularizers (e.g., weight decay) in our setups. In future work, it would be interesting to examine whether pruning complements the generalization improvements of other commonly used regularization techniques.

REFERENCES

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine learning and the bias-variance trade-off. *arXiv preprint arXiv:1812.11118*, 2018.

Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. *arXiv preprint arXiv:1703.04933*, 2017.

Gintare Karolina Dziugaite and Daniel M Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. *arXiv preprint arXiv:1703.11008*, 2017.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

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`.

Aidan N Gomez, Ivan Zhang, Kevin Swersky, Yarin Gal, and Geoffrey E Hinton. Targeted dropout. 2018.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 2015b.

Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pp. 164–171, 1993.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pp. 5–13. ACM, 1993.

Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Guillaume Leclerc, Manasi Vartak, Raul Castro Fernandez, Tim Kraska, and Samuel Madden. Smallify: Learning network size while training. *arXiv preprint arXiv:1806.03723*, 2018.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pp. 2755–2763. IEEE, 2017.

Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pp. 3290–3300, 2017.

Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. *arXiv preprint arXiv:1701.05369*, 2017.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient transfer learning. *arXiv preprint arXiv:1611.06440*, 2016.

Ari Morcos, David GT Barrett, Neil C Rabinowitz, and Matthew Botvinick. On the importance of single directions for generalization. In *Proceeding of the International Conference on Learning Representations*, 2018.

Vaishnavh Nagarajan and J Zico Kolter. Generalization in deep networks: The role of distance from initialization. *arXiv preprint arXiv:1901.01672*, 2019.

Sharan Narang, Gregory Diamos, Shubho Sengupta, and Erich Elsen. Exploring sparsity in recurrent neural networks. *arXiv preprint arXiv:1704.05119*, 2017.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. *arXiv preprint arXiv:1412.6614*, 2014.

Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. Towards understanding the role of over-parametrization in generalization of neural networks. *arXiv preprint arXiv:1805.12076*, 2018.

Ben Poole, Jascha Sohl-Dickstein, and Surya Ganguli. Analyzing noise in autoencoders and deep networks. *arXiv preprint arXiv:1406.1831*, 2014.

Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Zhewei Yao, Amir Gholami, Qi Lei, Kurt Keutzer, and Michael W Mahoney. Hessian-based analysis of large batch training and robustness to adversaries. In *Advances in Neural Information Processing Systems*, pp. 4949–4959, 2018.

Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. *arXiv preprint arXiv:1802.00124*, 2018.

Zhonghui You, Jinmian Yan, Kun; Ye, Meng Ma, and Ping Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2019.

Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9194–9203, 2018.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 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.

A APPENDIX

A.1 DERIVATION OF E[BN] MAGNITUDE (SECTION 3.1.1)

To approximate the expected value/magnitude of a batch-normalized, post-ReLU feature map activation, we start by defining the 2D feature map produced by convolution with BN:

$$M = \gamma \mathrm{BN}(W * x) + \beta.$$

We approximate the activations within this feature map as $M_{ij} \sim \mathcal{N}(\beta, \gamma)$. This approximation is justified if central limit theorem assumptions are met by the dot products in $W * x$, and we empirically show in Figure A.1 that this approximation is highly accurate early in training, though it becomes less accurate as training progresses. Given this approximation, the post-ReLU feature map

$$R = \max\{0, M\}$$

has elements R_{ij} that are either 0 or samples from a truncated normal distribution with left truncation point $l = 0$, right truncation point $r = \infty$, and mean μ where

$$\mu = \gamma \frac{\phi(\lambda) - \phi(\rho)}{Z} + \beta,$$
$$\lambda = \frac{l - \beta}{\gamma}, \rho = \frac{r - \beta}{\gamma}, Z = \Phi(\rho) - \Phi(\lambda),$$

and $\phi(x)$ and $\Phi(x)$ are the standard normal distribution's PDF and CDF (respectively) evaluated at x. Thus, an approximation to the expected value of R_{ij} is given by

$$\mathbb{E}[R_{ij}] \approx \Phi(\lambda)0 + (1 - \Phi(\lambda))\mu.$$

"*E[BN] pruning*" uses this approximation to $\mathbb{E}[R_{ij}]$ to calculate filter magnitude and is otherwise analogous to magnitude pruning.

The main drawback to the E[BN] approach is the sometimes poor approximation $M_{ij} \sim N(\beta, \gamma)$. For a VGG19 model, we found that the extent to which the approximation holds depends on the layer and training epoch (Figure A.1). A less serious drawback is that this approach does not account for the strength of connections to the post-BN feature map, which could have a large expected value but low importance if relatively small-magnitude weights connected it to the following layer.

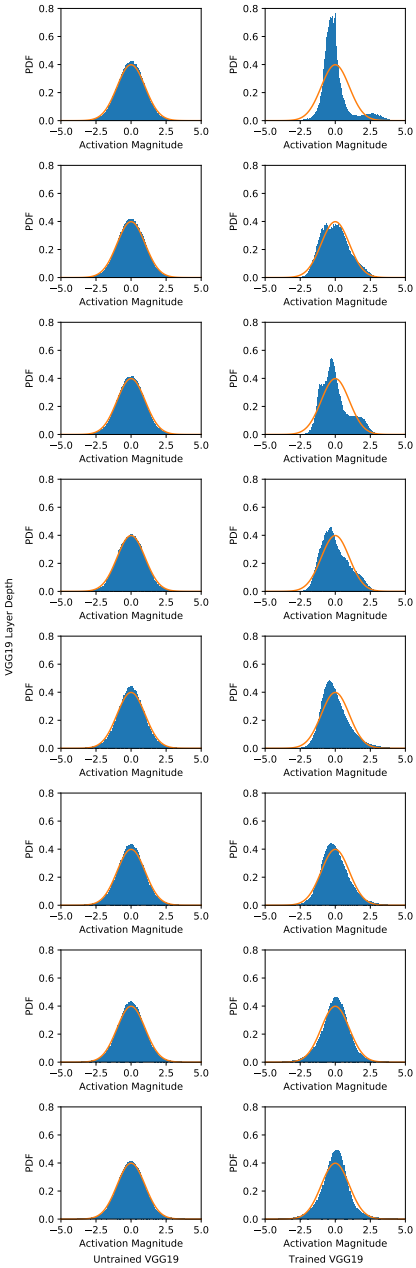

Figure A1: We examined the normalized activations (shown in blue histograms) of feature maps in the final eight convolutional layers of VGG19 before (left) and after (right) training to convergence. We found that the approximation to standard normality (shown in orange) of these activations is reasonable early on but degrades with training (particularly in layers near the output).

A.2 EPOCH-WISE INSTABILITIES

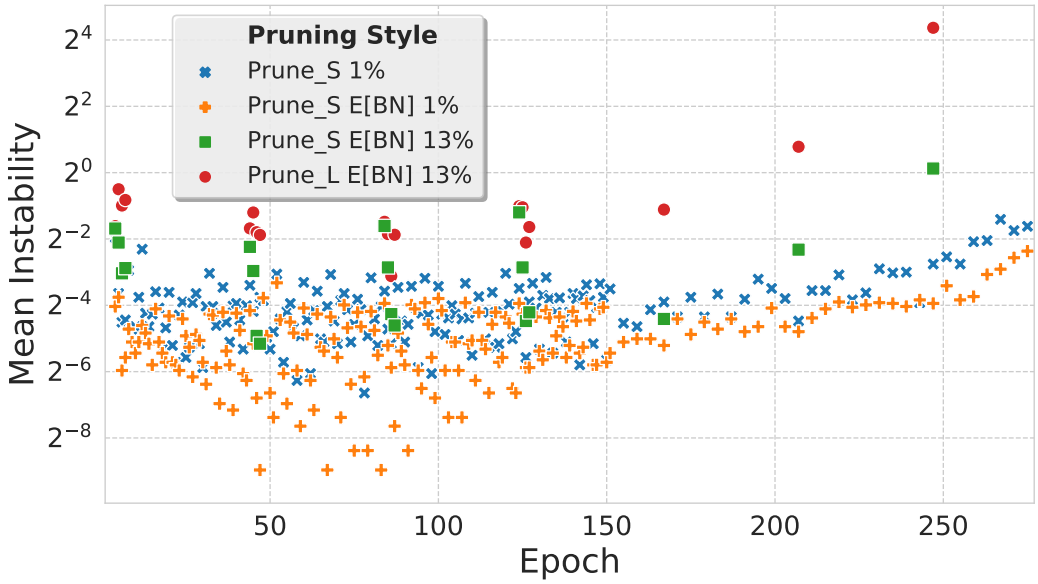

Figure A2: In VGG11, prune_S E[BN] is more stable than prune_S, which uses filter-ℓ_2-norm to compare parameter magnitudes. Methods with higher iterative pruning rates create more instability on a given iteration. Means reduce along the run dimension (10 runs per configuration).

Note that this graph uses a method (prune_S) that was included in Figure 1 right, but was not displayed in Figure 1 left due to its similarity to prune_S E[BN]. Additional experimental details are in Section A.3.1.

A.3 EXPERIMENTAL DETAILS

Unstructured magnitude pruning entails removing individual weights (subsets of filters/neurons), which are selected for pruning based on their magnitude. Our unstructured pruning approach does not allow previously pruned weights to reenter the network (Narang et al., 2017; Zhu & Gupta, 2017; Gale et al., 2019). Structured magnitude pruning removes entire filters/neurons, which are selected based on their ℓ_2-norms or via the E[BN] calculation. Except where noted, we use structured pruning for VGG11 and ResNet18.

We denote the pruning of n layers of a network by specifying a series of epochs at which pruning starts $s = (s_1, ..., s_n)$, a series of epochs at which pruning ends $e = (e_1, ..., e_n)$, a series of fractions of parameters to remove $p = (p_1, ..., p_n)$, and an inter-pruning-iteration retrain period $r \in \mathbb{N}$. For a given layer l, the retrain period r and fraction p_l jointly determine the iterative pruning percentage i_l. Our experiments prune the same number of parameters $i_l \times \text{size}(layer_l)$ per pruning iteration, ultimately removing $p_l \times 100\%$ of the parameters by the end of epoch e_l.

Our approach is designed to study the effects of changing factors such as the iterative pruning rate and lacks some practically helpful features, e.g. hyperparameters indicating how many parameters can be safely pruned (Liu et al., 2017; Molchanov et al., 2017). When layerwise iterative pruning percentages differ (i.e., when there exists an a and b such that i_a and i_b are unequal), our figures state the largest iterative pruning rate that was used in any of the layers.

For ResNet, our pruning algorithms did not account for the magnitude of incoming shortcut connections when judging filter magnitude/importance. Though we did prune the incoming and outgoing shortcut connections associated with any pruned feature maps.

We used only the CIFAR-10 dataset (Krizhevsky & Hinton, 2009) in our experiments, a limitation of our study. We used batch size 128, and only used data augmentation in the decile experiment (Figure 2). For some experiments, we give multi-step learning rate schedules $lr_s = (x, y)$, which means we shrink the learning rate by a factor of 10 at epochs x and y.

A.3.1 FIGURE 1

We used E[BN] pruning in all models that were pruned, except for one model that used ℓ_2-norm magnitude pruning, which was included in Figure 1 right but not displayed in Figure 1 left due to its qualitative similarity to prune$_S$ E[BN]. We leave out "E[BN]" in the legend of Figure 1 left, but all models nonetheless used E[BN] pruning.

The models were trained on CIFAR-10 with Adam for 325 epochs with $lr_s = (150, 300)$. The error bars are 95% confidence intervals for the mean, bootstrapped from 10 distinct runs of each experiment.

Since the layerwise pruning percentages varied, pruning required multiple iterative pruning percentages, the largest of which is denoted in the legend (rounded to the nearest integer).

VGG11 Pruning targeted the final four convolutional layers during training with (layerwise) starting epochs $s = (3, 4, 5, 6)$, ending epochs $e = (150, 150, 150, 275)$, and pruning fractions $p = (0.3, 0.3, 0.3, 0.9)$. To allow for the same amount of pruning among models with differing iterative pruning percentages, we adjusted the number of inter-pruning retraining epochs. The models with the maximum iterative pruning percentage of 1% had $r = 4$, while the models with the maximum iterative pruning percentage of 13% had $r = 40$. The model pruned with ℓ_2-norm magnitude pruning, which only appeared in Figure 1 right, had $r = 4$ as well.

ResNet18 Pruning targeted the final four convolutional layers of ResNet18 during training with (layerwise) starting epochs $s = (3, 4, 5, 6)$, ending epochs $e = (150, 150, 170, 275)$, and pruning fractions $p = (0.25, 0.4, 0.25, 0.95)$. As noted in Section 4.1, we increased the pruning rate of the output layer of the penultimate block to remove shortcut connections to the last layer, thinking that it should increase the duration of adaptation to pruning. The models with the maximum iterative pruning percentage of 1% had $r = 4$, while the models with the maximum iterative pruning percentage of 13% had $r = 40$.

A.3.2 FIGURE 2

Each experiment in Figure 2 targeted one of ten weight-magnitude deciles in the post-convolutional linear layer of the Conv4 network during training on CIFAR-10 with data augmentation.

While there are just ten deciles, the iterative nature of our pruning algorithms required the creation of eleven different pruning targets: ten methods pruned from the bottom of the decile upward (one experiment for each decile's starting point: 0th percentile, 10th percentile, etc.), and one (D10) pruned from the last decile's ending point downward (pruning the very largest collection of weights each iteration). In other words, D9 and D10 targeted the same decile (90th percentile to maximum value), but only D10 actually removed the largest weights on a given iteration (weights in the 100th-99th percentiles, for example). The D9 experiment would target weights starting from the 90th percentile (e.g. it may prune the 90th-91st percentiles on a particular iteration).

The training/pruning setup used the Adam optimizer, $s = (4)$, $e = (52)$, $p = (0.1)$, $r = 3$, and $lr_s = (30, 60)$. We calculated the generalization gap on epoch 54 and sampled average pruned magnitudes on epoch 35. We obtained qualitatively similar results regardless of whether we used fewer training epochs or data augmentation. The error bars are 95% confidence intervals for the means, bootstrapped from 10 distinct runs of each configuration.

A.3.3 FIGURE 3

In Figure 3, prune_L targeted the final four convolutional layers of ResNet18 during training with (layerwise) starting epochs $s = (3, 4, 5, 6)$, ending epochs $e = (150, 150, 150, 275)$, and pruning fractions $p = (0.1, 0.1, 0.1, 0.9)$. The models with the maximum iterative pruning percentage of 1% had $r = 4$, the models with the maximum iterative pruning percentage of 13% had $r = 40$, and the "One Shot" model pruned all its targeted parameters at once on epoch 246.

When performing unstructured pruning, we pruned individual weights from filters based on their magnitude. The structured pruning experiments used E[BN] pruning.

The models were trained on CIFAR-10 with Adam for 325 epochs with $lr_s = (150, 300)$. The error bars are 80% confidence intervals for the means, bootstrapped from 3+ distinct runs of each experiment.

A.3.4 FIGURE 4

In Figure 4, pruning targeted the final four convolutional layers of VGG11 during training with (layerwise) starting epochs $s = (3, 4, 5, 6)$, ending epochs $e = (150, 150, 150, 275)$, and pruning fractions $p = (0.3, 0.3, 0.3, 0.9)$. To create the different iterative pruning rates, we used models with inter-pruning retrain periods $r = 4$, $r = 20$,$r = 40$, and $r = 60$. Since the layerwise pruning percentages varied, pruning required multiple iterative pruning percentages, the largest of which is denoted on the horizontal axis. An unpruned baseline model average (10 runs) is plotted on the dotted line.

The models were trained on CIFAR-10 with Adam for 325 epochs with $lr_s = (150, 300)$. The error bars are 95% confidence intervals for the means, bootstrapped from 10 distinct runs of each experiment.

A.3.5 FIGURE 5

In Figure 5, pruning targeted the final four convolutional layers of VGG11 during training with (layerwise) starting epochs $s = (3, 4, 5, 6)$, ending epochs $e = (150, 150, 150, 275)$, pruning fractions $p = (0.3, 0.3, 0.3, 0.9)$, and inter-pruning-iteration retrain period $r = 40$. When injecting pruning noise, we used the same pruning schedule and percentages, but applied noise to the parameters instead of removing them. The Gaussian noise had mean 0 and standard deviation equal to the empirical standard deviation of a noiseless filter from the same layer. Prune_L used ℓ_2-norm pruning.

The models were trained on CIFAR-10 with Adam for 325 epochs with $lr_s = (150, 300)$. The error bars are 95% confidence intervals for the means, bootstrapped from 10 distinct runs of each experiment.