The Generalization-Stability Tradeoff in Neural Network Pruning

Brian R. Bartoldson [1] **Ari S. Morcos** [2] **Adrian Barbu** [1] **Gordon Erlebacher** [1]

Abstract

Pruning neural network parameters to reduce model size is an area of much interest, but the original motivation for pruning was the prevention of overfitting rather than the improvement of computational efficiency. This motivation is particularly relevant given the perhaps surprising observation that a wide variety of pruning approaches confer increases in test accuracy, even when parameter counts are drastically reduced. 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 generated by pruning than the final size of the pruned model. We demonstrate that even pruning of seemingly unimportant parameters can lead to such instability, allowing our finding to account for the generalization benefits of modern pruning techniques. Our results ultimately suggest that, counter-intuitively, pruning regularizes through instability and mechanisms unrelated to parameter counts.

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 *increase* accuracy, even when parameter counts are reduced by a factor of 10 or more. Consistent with this, pruning was originally motivated as a means to prevent highly-parameterized networks from overfitting to finite datasets (LeCun et al., 1990).

[1]Florida State University [2]Facebook AI Research. Correspondence to: Brian R. Bartoldson <bbartoldson@fsu.edu>.

ICML 2019 Workshop on Understanding and Improving Generalization in Deep Learning, Long Beach, California, 2019.

However, the fear of potential overfitting has recently been replaced by surprise that modern DNNs (with parameter counts on the order of 10^7 and larger) generalize well despite their capacity to overfit (Neyshabur et al., 2014; Zhang et al., 2016). This finding has led to a flurry of studies attempting to explain DNN robustness from various perspectives, including empirical investigations (Neyshabur et al., 2014; Keskar et al., 2016; Morcos et al., 2018; Nagarajan & Kolter, 2019), as well as the derivation of generalization bounds that imply no additional (or perhaps even less) overfitting occurs as parameter counts increase (Neyshabur et al., 2017; Dziugaite & Roy, 2017; Neyshabur et al., 2018). These results raise a puzzling question: if large parameter counts don't result in overfitting, how can pruning increase performance?

To answer this question, we analyzed variants of magnitude pruning (Han et al., 2015b) over the course of training, finding that pruning large-magnitude weights rather than small-magnitude weights, an approach rarely taken in the literature, can actually lead to *better generalization*. We then demonstrate that this generalization benefit appears to be due to the instability generated by pruning rather than a property of large weights. Indeed, we found that pruning small weights can be tailored to generate as much instability as pruning large weights (especially in batch-normalized networks) and to confer the commensurate generalization benefit. This finding motivated our derivation of an approach to magnitude pruning of batch-normalized-parameters that accounts for the normalization process's ability to obscure parameter importance. The totality of our results suggests that parameter-count-dependent generalization bounds are unlikely to explain pruning's ability to improve test accuracy, while other approaches to understanding generalization such as minimum description length appear capable of explaining the effects of pruning.

2. Approach

Many factors affect how stable a neural network's output is in response to pruning. We restrict our experiments to the exploration of the following subset: pruning target, pruning schedule, pruning percentage, and model. In this section, we provide an overview of these factors and demonstrate a need for a novel pruning target, which we derive.

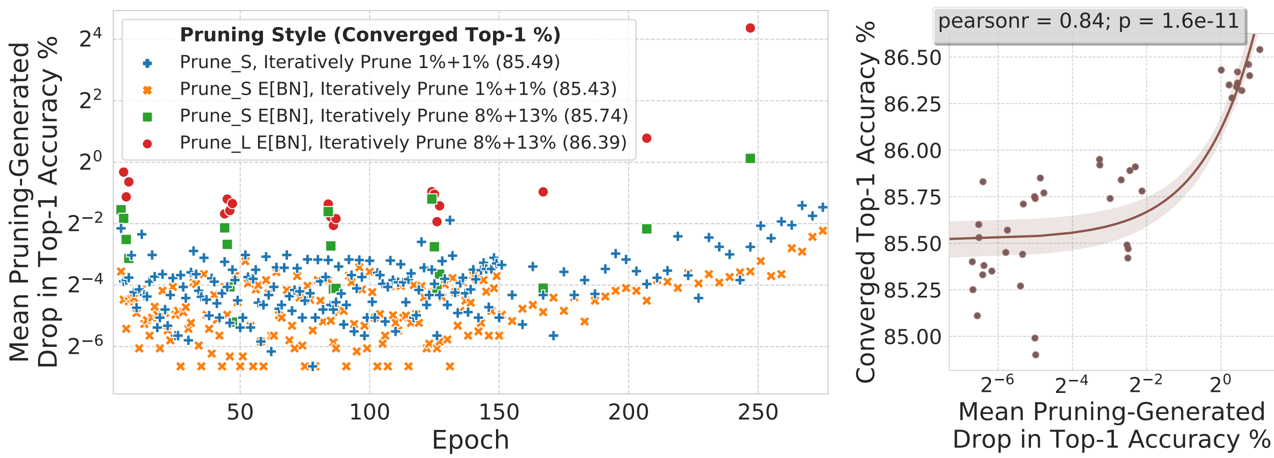

Figure 1. The instability levels generated by different approaches to pruning (10 runs per configuration). Pruning methods that generate more instability have higher top-1 accuracies. (Left) Means reduce along the run dimension and are computed from only positive drop values to aid visualization. (Right) Means reduce along the run and epoch dimensions and contain all drop values. Pruning targeted the final four convolutional layers of VGG11 during training on CIFAR-10 data with (layerwise) starting epochs $s = (3, 4, 5, 6)$, ending epochs $e = (150, 150, 150, 275)$, and pruning fractions $p = (0.3, 0.3, 0.3, 0.9)$. All models had 42% of their 9,231,114 parameters removed. Since the pruning disproportionately targeted the final layer, pruning required two separate iterative pruning percentages, denoted in the legend. 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 were trained with Adam until convergence at 325 epochs with $lr_s = (150, 300)$.

2.1. Pruning Target

In all of our experiments, we use magnitude pruning (Han et al., 2015b). We denote pruning algorithms that target small-magnitude parameters with an "S" subscript (e.g. prune$_{\mathrm{S}}$), random parameters with an "R" subscript, and large-magnitude parameters with an "L" subscript. The usual approach to pruning involves removing parameters that have a small magnitude (Li et al., 2016; Gale et al., 2019), or a small effect on the loss function (LeCun et al., 1990; Hassibi & Stork, 1993; Molchanov et al., 2016; 2017; Louizos et al., 2017; Ye et al., 2018; Yu et al., 2018). Despite the fact that small-magnitude weights are not necessarily the least important to the loss function (LeCun et al., 1990; Hassibi & Stork, 1993), our experiments and (Gale et al., 2019) suggest that magnitude can be an effective alternative to more sophisticated judgments of a parameter's importance to the model.

2.1.1. Identifying Important Batch-Normalized Parameters

The relationship between parameter magnitude and importance is particularly confusing in the context 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. This suggests that equating small magnitude with unimportance may be flawed for batch-normalized parameters, and 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). In A.1, we derive a novel approach to determining filter importance/magnitude that incorporates both γ and β. We denote this approach "E[BN] pruning".

2.2. Pruning Schedule and Percentage

We denote the pruning of n layers 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 a retrain period $r \in N$. 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).

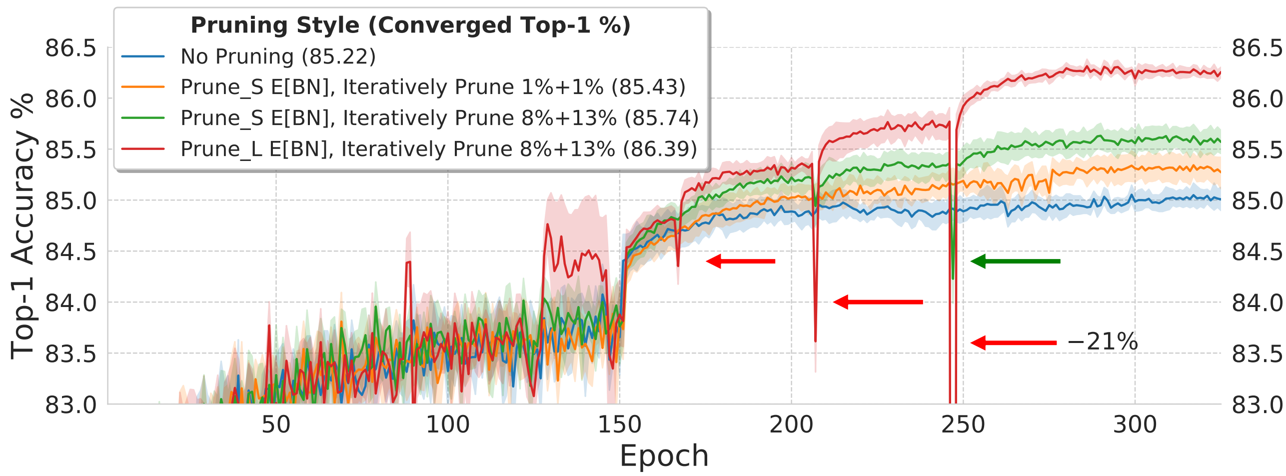

Figure 2. The top-1 test accuracy during training of VGG11 on CIFAR10 data. The first approach is an unpruned baseline. The other approaches use E[BN] pruning, but differ in their targets' magnitudes (small vs. large), and their iterative pruning percentages. Pruning was performed as described in Figure 1. The 95% confidence intervals are bootstrapped from 10 runs per configuration.

2.3. Summary of Models

The models considered are: a network with convolutions (2x32, pool, 2x64, pool) and fully connected layers (512, 10) that we denote Conv4, and VGG11 with its fully-connected layers replaced by a single fully-connected layer. All convolutions are 3x3. Our experiments applied these models to the CIFAR-10 dataset (Krizhevsky & Hinton, 2009) and employed data augmentation only where noted. Additional details on the training and pruning of these models are provided in A.2.

3. Experiments

3.1. Generalization from Instability

Modern generalization bounds make it difficult to simply attribute pruning-generated accuracy improvements to a reduction in parameter count. We therefore consider another original motivation of pruning: minimizing description length (MDL) (Rissanen, 1978; LeCun et al., 1990; Hassibi & Stork, 1993; Hochreiter & Schmidhuber, 1997). Pruning is capable of disrupting the network's computations by effectively adding noise to the internal representations of the input, which, when deployed throughout training, may encourage learned parameters to be less sensitive to noise and therefore able to be described more succinctly (Hochreiter & Schmidhuber, 1997; Srivastava et al., 2014; Poole et al., 2014). Alternatively, if the pruning process preserves the outputs of a network, then it likely failed to alter the representations of its inputs in a material way (or the network is already robust to pruning-induced noise), and the pruned model should not be expected to have a significantly different description length than the original model. The MDL principle therefore suggests that more disruptive approaches are more capable of improving a model's generalization than those which leave a model effectively unchanged.

To determine whether more disruptive pruning approaches generalize better, we compared the pruning-generated instability and final top-1 test accuracy of four unique pruning algorithms (Figure 1). We assessed the level of instability produced by the pruning procedures via the difference in test accuracy immediately before and after pruning.

Surprisingly, we observed that pruning algorithms that destabilized the network *more* over the course of training resulted in *higher* final test accuracies than those which were stable (Figure 1; correlation $= .84$, p-value $= 1.6\text{e}{-11}$). These results suggest that pruning techniques may facilitate better generalization when they induce *more* instability, consistent with the MDL principle. Furthermore, this result lends support to the idea that generalization benefits from pruning are due to the noise pruning adds rather than the parameter count reduction.

Figure 2 illustrates the test-accuracy dynamics of an unpruned baseline network and this same network under three of the pruning regimes from Figure 1. Pruning events for prune$_{\mathrm{L}}$ with a high iterative pruning rate (red curve, pruning either 13% of the final convolutional layer or 8% of one of the other 3 convolutional layers targeted per pruning iteration) are substantially more destabilizing than other pruning events, yet surprisingly, 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.

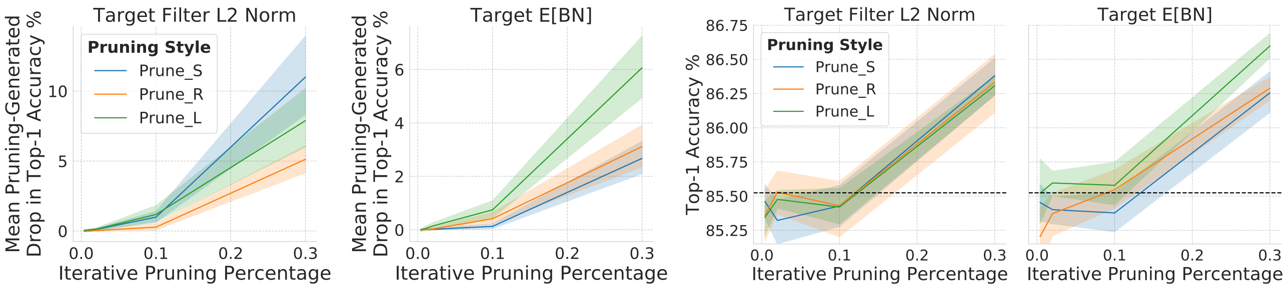

Figure 3. Here we seek to understand the effects of progressively larger pruning events. We pruned the final four convolutional layers of VGG11 during training on CIFAR-10 data with (layerwise) starting epochs $s = (3, 3, 3, 3)$, ending epochs $e = (50, 50, 50, 120)$, maximum prune fractions $p = (.3, .3, .3, .9)$, and inter-pruning retrain epochs $r = 40$. This experiment was unique in that we did not necessarily reach the total pruning percentages in p, rather we pruned every $r = 40$ epochs the same constant iterative pruning percentage $i = (x/2, x/2, x/2, x)$ for the x given in the figure's x-axis. The models were trained with Adam until convergence at 175 epochs with $lr_s = (50, 120)$.

3.2. Iterative Pruning Percentage

Is magnitude the only factor impacting pruning-induced instability? Another possibility is that removing more weights at each pruning event (while keeping the final fraction of pruned weights constant) may also increase instability, consistent with observations that post-hoc iterative pruning is often more effective (Han et al., 2015b). If this is the case, we would expect that increasing the iterative pruning rate should also increase instability and 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.

To test this, we plotted the average drop in accuracy immediately following a pruning event as well as the test accuracy as a function of different iterative pruning rates (Figure 3). Consistently, we observed that increasing the iterative pruning rate increased both the instability induced by pruning (Figure 3 left) and the overall test accuracy (Figure 3 right). Figure 3 holds r constant while allowing the final pruning percentage to vary, while Figure A2 holds the final pruning percentage constant and allows r to vary: the same relationships between instability, generalization, and iterative pruning percentage appear in each figure. These result suggests that using higher iterative pruning rates during training is an effective method to induce additional instability and generalization.

Interestingly, we found that while using standard magnitude pruning, there was little difference in test accuracy between pruning small, large, or random weights, and pruning small weights actually induced *more* instability than pruning large weights. In contrast, for E[BN] magnitude pruning, pruning large weights consistently resulted in greater instability and test performance. These results suggest that the precise pruning algorithm used can have a dramatic impact on the factors which introduce instability and induce better test performance.

In Figure 1, at a low iterative pruning percentage, E[BN] pruning led to a small but statistically significant 0.02 percentage point lower average disruption to test accuracy than filter-ℓ2-norm pruning (p-value 10e-10), and we find further support for this pattern in Figure 3. Although, we cannot entirely attribute this result to E[BN] pruning because our filter-ℓ2-norm pruning approach did not set the batch-normalization bias of pruned filters to zero, which creates additional instability when pruning (Morcos et al., 2018).

4. Conclusion

We applied several pruning approaches to multiple neural networks, assessing the approaches' effects on instability and generalization. Throughout, we observed that pruning algorithms that generated more instability led to better test accuracies. For instance, we found that utilizing high iterative pruning rates, rather than total parameters pruned, was particularly important to the creation of instability and generalization (see Figure 1, Section 3.1). This lends support to hypotheses stating that pruning regularizes through mechanisms unrelated to parameter counts, and supports the idea that the instability produced by pruning can induce networks to become robust to noisy internal representations, which leads to better generalization in the minimum description length framework.

In Appendix A.4, we demonstrate that the regularization benefit of pruning can be outweighed by the destruction of effective capacity. This problem was most noticeable when the model capacity was small relative to the number of examples per class. As such, an important caveat is that many of our results were generated with VGG11 and CIFAR-10, so future work will be required to evaluate whether the presented phenomena hold in large datasets and models.

References

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

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

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

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

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

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

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

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

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

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

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

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. Learning efficient convolutional networks through network slimming. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pp. 2755–2763. IEEE, 2017.

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

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

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. Pruning convolutional neural networks for resource efficient transfer learning. *arXiv preprint arXiv:1611.06440*, 2016.

Morcos, A., Barrett, D. G., Rabinowitz, N. C., and Botvinick, M. On the importance of single directions for generalization. In *Proceeding of the International Conference on Learning Representations*, 2018.

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

Narang, S., Diamos, G., Sengupta, S., and Elsen, E. Exploring sparsity in recurrent neural networks. *arXiv preprint arXiv:1704.05119*, 2017.

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

Neyshabur, B., Bhojanapalli, S., McAllester, D., and Srebro, N. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, pp. 5949–5958, 2017.

Neyshabur, B., Li, Z., Bhojanapalli, S., LeCun, Y., and Srebro, N. Towards understanding the role of overparametrization in generalization of neural networks. *arXiv preprint arXiv:1805.12076*, 2018.

Poole, B., Sohl-Dickstein, J., and Ganguli, S. Analyzing noise in autoencoders and deep networks. *arXiv preprint arXiv:1406.1831*, 2014.

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

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

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

Yu, R., Li, A., Chen, C.-F., Lai, J.-H., Morariu, V. I., Han, X., Gao, M., Lin, C.-Y., and Davis, L. S. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9194–9203, 2018.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

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

A. Appendix

A.1. Expected Value of Batch Normalized Parameters

To approximate the expected value/magnitude of a batch-normalized, post-ReLU feature map activation, we start by defining the 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

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

We use "E[BN] pruning" to refer to magnitude pruning with the approximation to $E[R_{ij}]$ as a target. This target has two advantages. First, this approach avoids the problematic assumption that filter importance is tied to filter magnitude 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 targeting small-magnitude filters. 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).

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. 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.

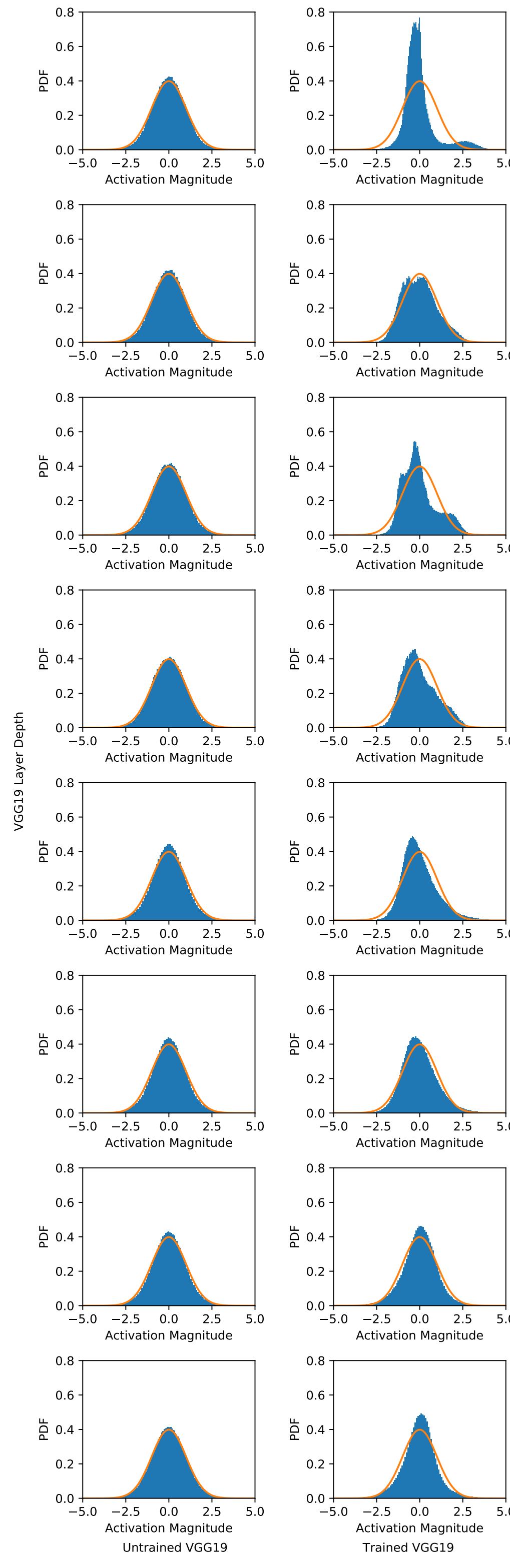

Figure A1. We examined the normalized activations (shown in blue histograms) of feature maps in the final eight convolutional layers of VGG19 before (left) and after (right) training to convergence. We found that the approximation to standard normality (shown in orange) of these activations is reasonable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

A.2. Additional Training/Pruning Details

Unstructured pruning of Conv4 is done via individual weight 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 pruning of VGG filters is done via the ℓ_2-norm of their weights or the E[BN] calculation. When pruning VGG models, we employ structured pruning of filters because **a)** most weights and FLOPs are created by their convolutional layers and **b)** speedups from pruning weights in these layers are most easily realized with modern algorithms/hardware by using structured pruning of entire filters (Molchanov et al., 2016). Pruning of Conv4 is always applied to its penultimate linear layer (which contains 94% of Conv4's 1,250,858 parameters), while pruning of VGG11 is always applied to its final four convolutional layers (which contain 90% of VGG11's 9,231,114 parameters).

We trained these models using Adam (Kingma & Ba, 2014) with initial learning rate $lr = 0.001$. We often found Adam more helpful than SGD for recovering from relatively destabilizing pruning events. We used batch size 128 except where noted. 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. Varying Iterative Pruning Percentage with a Constant Final Pruning Fraction

Here, we demonstrate the same patterns found in Figure 3 while holding final pruning percentage constant. Achieving this requires allowing r to vary: lower iterative pruning percentages require smaller values of r to reach a given final pruning percentage by the end of training. The combination of Figures 3 and A2 suggest that, given a final pruning percentage, iterative pruning percentage can fuel more instability and better generalization.

Future studies similar to Figure A2 could be conducted to explore a wider range of iterative pruning percentages. Achieving this while holding constant the final pruning fraction could be done by utilizing more total training epochs.

A.4. When More Instability Fails to Produce Better Generalization

Pruning approaches can be tailored to create instability that prevents network recovery (an extreme example being targeting all of the parameters of a given layer), but there are also more subtle cases in which the tradeoff presented here is less easily leveraged or may not even apply.

One of the best approaches to improving neural network generalization is expanding the size of the training dataset. If a model has low capacity relative to the number of training data instances per class, then overfitting becomes a worse strategy for minimizing the loss on the training dataset. Consequently, we would expect that pruning a model with low capacity (relative to dataset size) may mask or even overcome the regularization benefits of pruning.

Here, we seek to determine the extent to which the regularization effect of pruning can be outweighed by its effect on model capacity. In Conv4 experiments (A3), prune_S and prune_L prune stably and unstably, respectively. Stable pruning algorithms preserve the function computed by the network, enabling us to consider prune_S and prune_L as approaches that do and do not (respectively) preserve the model's effective capacity during training. As a model's effective capacity decreases, its ability to fit large datasets should also decrease. As a result, if the generalization-stability tradeoff can be masked by an unstable pruning approach's impact on effective capacity, then we would expect there to be some training set size at which performance would start to degrade. To test this, we train Conv4 on progressively larger subsets of CIFAR-10, seeking a point at which prune_L removes enough effective capacity to cause underfitting to the dataset and the consequent reduction in performance (Figure A3 left, middle).

Consistent with instability conferring generalization benefits as long as capacity is not affected by pruning, we see prune_L creating an initially substantial generalization benefit that declines with dataset size (Figure A3 left, middle). This effect is particularly prominent in the context of data augmentation (random crops and horizontal flips), which dramatically increases the effective dataset size.

Our batch size of 128 on the full dataset corresponds to 391 parameter updates per epoch, and we adjusted batch sizes for each data subset size such that approximately 391 updates occurred per epoch. For instance, the 128 examples/class experiment has a batch size of $(128 \times 10)//391 = 3$, which performs 427 updates/epoch. Since smaller batch sizes confer better generalization (Keskar et al., 2016), the apparent generalization benefit of prune_L may actually be muted; i.e., the baseline may already be regularized to some extent by the noisier gradient of the smaller batch size.

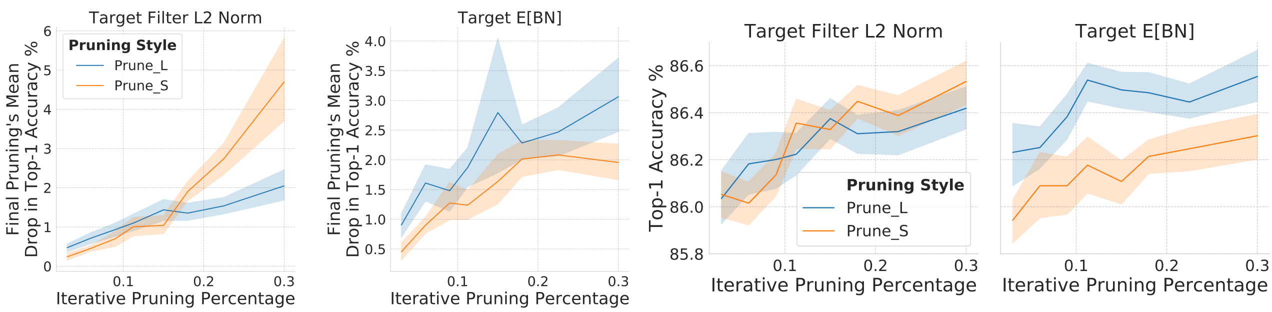

Figure A2. We pruned the final four convolutional layers of VGG11 during training on CIFAR-10 data with (layerwise) starting epochs $s = (3, 3, 3, 3)$, ending epochs $e = (50, 50, 50, 120)$, and prune fractions $p = (.1, .1, .1, .9)$. We allow retrain period r to vary such that the total prune fraction p will be met by the end of epoch e; specifically, the first iterative pruning percentage plotted corresponds to $r = 4$ and each of the following experiments add 4 to the prior experiment's r, ending in $r = 40$. The layerwise iterative pruning percentages each result from the combination of s_l, e_l, p_l, and r; they are roughly $i = (x/5, x/5, x/5, x)$ for the x given in the figure's x-axis. Unlike Figures 1 and 3, the mean post-pruning drops in accuracy were calculated using the top-1 test accuracy immediately after the final pruning event, and the top-1 test accuracy from the epoch before this pruning (the other figures made this calculation immediately before and immediately after pruning, and they used all pruning events); regardless, the same patterns exist in the pruning-generated accuracy drops here. The models were trained with Adam until convergence at 175 epochs with $lr_s = (50, 120)$.

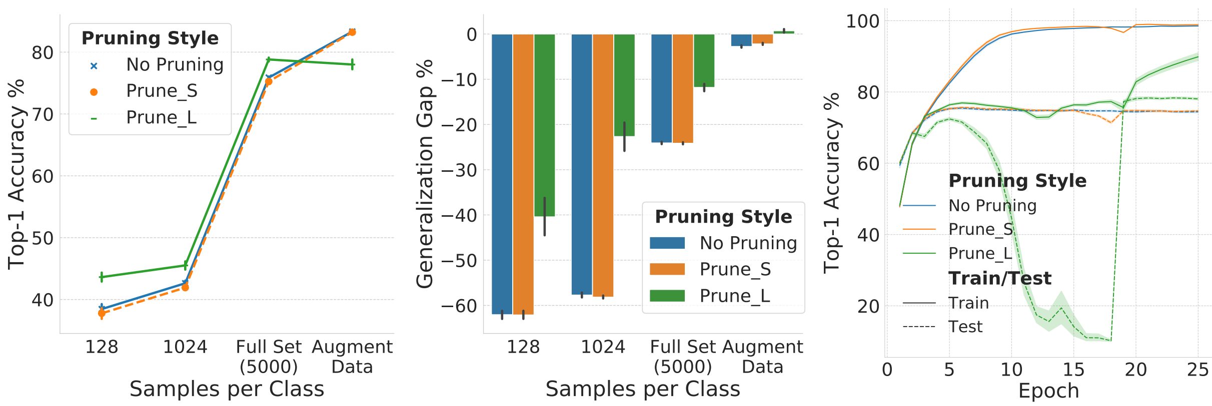

Figure A3. We trained Conv4 on CIFAR10 using 4 levels of data for 25 epochs. We found that, without data agumentation, the regularization effect of pruning can significantly improve the baseline's generalization (left) because the baseline's generalization gap (the difference between test and train accuracy) is large (middle). At all data levels, the generalization gap is better reduced by pruning the largest-magnitude weights (prune$_{\mathrm{L}}$) than by pruning the smallest-magnitude weights (prune$_{\mathrm{S}}$). (Right) The training accuracies and test accuracies (the latter were calculated immediately after pruning) illustrate how much each pruning algorithm disturbs the neural network's output during training on the full dataset. The pruning algorithms start on epoch $s = (3)$, end on epoch $e = (18)$, prune the percentage $p = (0.9)$, and prune every epoch via retrain period $r = 1$. The error bars in all three plots are 95% confidence bootstrapped from 20 distinct runs of each experiment.