Pruning neural networks: is it time to nip it in the bud?

Elliot J. Crowley, Jack Turner, Amos Storkey, Michael O'Boyle
School of Informatics
University of Edinburgh
{elliot.j.crowley,jack.turner,a.storkey}@ed.ac.uk, mob@inf.ed.ac.uk

Abstract

Pruning is a popular technique for compressing a neural network: a large pre-trained network is fine-tuned while connections are successively removed. However, the value of pruning has largely evaded scrutiny. In this extended abstract, we examine residual networks obtained through Fisher-pruning and make two interesting observations. First, when time-constrained, it is better to train a simple, smaller network from scratch than prune a large network. Second, it is the architectures obtained through the pruning process — not the learnt weights — that prove valuable. Such architectures are powerful when trained from scratch. Furthermore, these architectures are easy to approximate without any further pruning: we can prune once and obtain a family of new, scalable network architectures for different memory requirements.

1 Introduction

Deep neural networks excel at a multitude of tasks (LeCun et al., 2015), but are typically cumbersome, and difficult to deploy on embedded devices. This can be rectified by compressing the network; specifically, reducing the number of parameters it uses in order to reduce its runtime memory. This also reduces the number of operations that have to be performed; making the network faster.

A popular means of doing this is through pruning. One trains a large network and fine-tunes it while removing connections in succession. This is an expensive procedure that often takes longer than simply training a smaller network from scratch. Does a pruned-and-tuned network actually outperform a simpler, smaller counterpart? If not, is there any benefit to pruning?

In this work, we show that for a given parameter budget, pruned-and-tuned networks are consistently beaten by networks with simpler structures (e.g. a linear rescaling of channel widths) trained from scratch. This indicates that there is little value in the weights learnt through pruning. However, when the architectures obtained through pruning are trained *from scratch* (i.e. when all weights are reinitialised at random and the network is trained anew) they surpass their fine-tuned equivalents *and* the simpler networks. Moreover, these architectures are easy to approximate; we can look at which connections remain after pruning and derive a family of copycat architectures that when trained from scratch display similar performance. This gives us a new set of compact, powerful architectures.

2 Related Work

Typically, a network is pruned by either setting unimportant weights to zero, resulting in a network with large, sparse weight matrices (Hanson & Pratt, 1989; LeCun et al., 1990; Han et al., 2015, 2016; Scardapane et al., 2017; Wen et al., 2016), or by severing channel connections to produce a smaller, dense network (Molchanov et al., 2017; Theis et al., 2018; He et al., 2017). In this work we use Fisher-pruning (Theis et al., 2018), a powerful channel pruning method, as recent work (Turner et al., 2018) has empirically demonstrated that on embedded devices this technique produces small, dense networks that both run efficiently, and perform well. These networks are therefore applicable to real-time applications where efficiency is of the essence.

Concurrent to this work, Liu et al. (2018) have demonstrated the benefits of training pruned models from scratch for a variety of other pruning techniques. They postulate that pruning may be seen as a form of architecture search. This is contrary to the work of Frankle & Carbin (2018) who hypothesise that the pruning process finds well-initialised weights. Very recently, Lee et al. (2018) proposed a single-shot pruning scheme to produce a reduced architecture that is then trained from scratch; the benchmarks of this work and our discovered architectures are given in Section 3.2.

3 Experiments

Here, we perform a set of experiments to answer our two questions: (i) Does a pruned network outperform a smaller simpler network? (ii) Is the pruned architecture itself useful? We evaluate the performance of our networks by their classification error rate on the test set of CIFAR-10 (Krizhevsky, 2009). Implementation details are given at the end of the section.

3.1 Pruned networks vs. Simpler Networks trained from scratch

For our base network, we use WRN-40-2 — a WideResNet (Zagoruyko & Komodakis, 2016) with 40 layers, and channel width multiplier 2. We chose this network as it features modular blocks and skip connections, and is therefore representative of a large number of commonly used networks. It is reasonably compact and doesn't have a large, redundant fully connected layer. It has 2.2M parameters in total, the bulk of which lie in 18 residual blocks, each containing two convolutional layers. Let us denote each block as $B(N_i, N_m, N_o)$ for variable N_i, N_m, N_o — the input has N_i channels and the first convolutional layer has a N_m channel output; this goes through the second layer which outputs N_o channels. By default, $N_m = N_o$.

The network is first trained from scratch, and is then Fisher-pruned (Theis et al., 2018). We prune the channels of the activations between the convolutions in each block; this has the effect of introducing a *bottleneck* (changing N_m), reducing the number of parameters used in each convolution. We alternate between fine-tuning the network and removing the channel that has the smallest estimated effect on the loss, as in Theis et al. (2018). Before each channel is removed, the test error and parameter count for the network is recorded. The resulting trajectory of Test Error versus Number of Parameters is represented by the red curve in Figure 1.

We compare this process to training smaller, simpler networks from scratch. We train WRN-40-k networks — the yellow curve in Figure 1 — treating k as an architectural hyperparameter which we can vary to control the total number of parameters. We also train WRN-40-2 networks where we apply a scalar multiplier z to the middle channel dimension in each block — $B(N_i, zN_o, N_o)$ — giving the green curve in Figure 1. We similarly vary z to control the total number of parameters. In both cases, we vary the relevant architectural hyperparameter (k or z) to produce networks that have between 200K and 2M parameters, in increments of roughly 100K. We can see that these simple networks trained from scratch consistently outperform the pruned-and-tuned networks. This difference is markedly more pronounced the smaller the networks get.

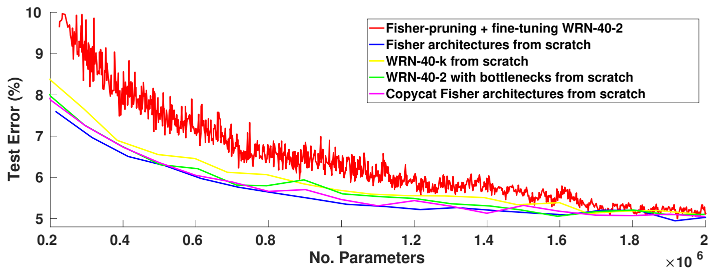

Figure 1: Test errors vs. Number of Parameters curves on CIFAR-10 for different families of networks. Curves for networks trained from scratch show the average error across 3 runs. The red curve corresponds to a WRN-40-2 undergoing Fisher-pruning and fine-tuning. The yellow curve represents WRN-40-k networks trained from scratch for various k whereas the green curve represents bottlenecked WRN-40-2 networks (also trained from scratch). Notice that the green and yellow curve are almost always below the red: it is preferable to train smaller architectures from scratch rather than prune. If we take Fisher-pruned architectures obtained along the red curve and train them from scratch we get the blue curve. Finally, if we profile the channel structure of a Fisher-pruned architecture, linearly scale it and train from scratch, we get the pink curve which closely follows the blue curve. Note that these blue and pink networks tend to outperform all others.

3.2 Examining the pruned architectures

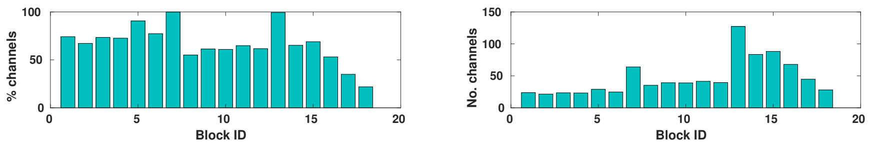

Figure 2: The channel profile for a Fisher-pruned network after 500 channels have been pruned. The left bar chart shows the percentage of remaining channels at the bottleneck in each residual block and the right bar chart shows the actual number. We use this profile to produce copycat networks.

It may seem thus far that pruning is a pointless endeavor. However, when we take the architectures produced along the pruning trajectory (at the same increments as above) and train them from scratch we obtain the blue curve in Figure 1. We can see that these networks outperform both (i) the fine-tuned versions of the same architectures and (ii) the simpler networks for most parameter budgets. This supports the notion that the true value of pruning is as a form of architecture search.

Figure 2 shows how many channels remain at the bottleneck of each block (the various N_m values) in the pruned-and-tuned WRN-40-2 after 500 channels have been Fisher-pruned (note that this profile is consistent with those obtained after further pruning). Notice that channels in later layers tend to be pruned; they are more expendable than those in earlier layers. It is intriguing that channels are rarely pruned in blocks 7 and 13; these blocks both contain a strided convolution that reduces the spatial resolution of the image representation. It is imperative that information capacity is retained for this process. These observations are consistent with those made in previous works (Veit et al., 2016; Huang et al., 2016; Jastrzębski et al., 2018).

To assess the merits of this particular structure, we train *copycat Fisher architectures* — architectures designed to mimic those obtained through Fisher-pruning. Specifically, we train a range of architectures from scratch where N_m in each block is proportional to that found in the Fisher-pruned architecture — block j can be represented by $B(N_i, \alpha N_{mj}, N_o)$: N_{mj} is a block-specific value corresponding to the height of each bar in Figure 2(right), and α scales the whole architecture. We vary α to produce networks with different parameter counts as before, which are then trained from scratch. These are represented by the pink curve in Figure 1. For most parameter budgets these networks perform similarly to those found through Fisher-pruning; this means we can simply prune a network once to find a powerful, scalable network architecture. Furthermore, the resulting networks are competitive. We compare them to the WideResNets produced by the pruning method of Lee et al. (2018): one of which gets 5.85% error on CIFAR-10 and has 858K parameters, and another gets 6.63% error with 548K parameters. Comparably, the copycat WideResNets in this work achieve (on average) 5.66% with 800K parameters, and 6.35% error with 500K parameters.

Implementation Details To train a network from scratch, we use SGD with momentum to minimise the cross-entropy loss for 200 epochs using mini-batches of size 128. Images were augmented using horizontal flips and random crops. The initial learning rate is 0.1 and is decayed by a factor of 0.2 every 60 epochs. We use weight decay of 5×10^{-4} and momentum of 0.9. For Fisher-pruning we fine-tune the learnt network with the lowest learning rate reached during training (8×10^{-4}) and momentum and weight decay as above. We measure the approximate effect on the change in loss (Theis et al., 2018) for each candidate channel in the network over 100 update steps and then remove the channel with the lowest value. We experimented with several different values for the FLOP penalty hyperparameter but found this made little difference. We therefore set it to zero.

4 Conclusion

Pruning is a very expensive procedure, and should be avoided when one is time-constrained. We show that under such constraints it is preferable to train a smaller, simpler network from scratch. Our work supports the view that pruning should be seen as a form of architecture search; it is the resulting structure — not the learnt weights — that is important. Pruned architectures trained from scratch perform well, and are easily emulated, as demonstrated by our copycat networks. These are a step towards a more efficient architecture for residual networks. Future work could entail expanding this analysis to other network types, datasets, and pruning schema. It would also be possible to use distillation techniques (Hinton et al., 2015; Zagoruyko & Komodakis, 2017) between our pruned architectures and the original architecture to further boost performance (Crowley et al., 2018).

References

Crowley, E. J., Gray, G., and Storkey, A. Moonshine: Distilling with cheap convolutions. In *Advances in Neural Information Processing Systems*, 2018.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding small, trainable neural networks. *arXiv:1803.03635*, 2018.

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*, 2015.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *International Conference on Learning Representations*, 2016.

Hanson, S. J. and Pratt, L. Y. Comparing biases for minimal network construction with back-propagation. In *Advances in Neural Information Processing Systems*, 1989.

He, Y., Zhang, X., and Sun, J. Channel pruning for accelerating very deep neural networks. In *International Conference on Computer Vision*, 2017.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv:1503.02531*, 2015.

Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Deep networks with stochastic depth. In *European Conference on Computer Vision*, 2016.

Jastrzębski, S., Arpit, D., Ballas, N., Verma, V., Che, T., and Bengio, Y. Residual connections encourage iterative inference. In *International Conference on Learning Representations*, 2018.

Krizhevsky, A. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, 2009.

LeCun, Y., Denker, J. S., and Solla, S. A. Optimal brain damage. In *Advances in Neural Information Processing Systems*, 1990.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature*, 521(7553):436–444, 2015.

Lee, N., Ajanthan, T., and Torr, P. H. S. SNIP: Single-shot Network Pruning based on Connection Sensitivity. *arXiv:1810.02340*, 2018.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. *arXiv:1810.05270*, 2018.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations*, 2017.

Scardapane, S., Comminiello, D., Hussain, A., and Uncini, A. Group sparse regularization for deep neural networks. *Neurocomputing*, 241:81–89, 2017.

Theis, L., Korshunova, I., Tejani, A., and Huszár, F. Faster gaze prediction with dense networks and fisher pruning. *arXiv:1801.05787*, 2018.

Turner, J., Cano, J., Radu, V., Crowley, E. J., O'Boyle, M., and Storkey, A. Characterising across stack optimisations for deep convolutional neural networks. In *IEEE International Symposium on Workload Characterization*, 2018.

Veit, A., Wilber, M. J., and Belongie, S. Residual networks behave like ensembles of relatively shallow networks. In *Advances in Neural Information Processing Systems*, 2016.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, 2016.

Zagoruyko, S. and Komodakis, N. Wide residual networks. In *British Machine Vision Conference*, 2016.

Zagoruyko, S. and Komodakis, N. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In *International Conference on Learning Representations*, 2017.