Exploring Large Convolutional Network Pruning

Yashika Batra, Spandan Das, Bradley Xu. April 17, 2024

1 Methodology

1.1 Magnitude Pruning. ***Learning Both Weights and Connections for Efficient Neural Networks. [1] [5]*** Network pruning is a three-step process that begins with initial training to identify significant connections between neurons based on larger weights. The second step, pruning, removes connections falling below a certain threshold, creating a sparser network with improved computation speed and memory usage. Finally, the sparse network is fine-tuned to recover any lost accuracy due to pruning. This iterative process of training, pruning, and retraining allows for increasingly aggressive pruning without substantial accuracy loss.

1.2 Pruning Filters. ***Pruning Filters for Efficient ConvNets. [2] [6]*** Filter pruning streamlines convolutional neural networks by removing less critical filters from convolutional layers, reducing the model's parameter count. The process involves measuring filter importance using the L1-norm of weights, where smaller L1-norms indicate less significant filters. A predetermined percentage of these less important filters are pruned from each layer, with more aggressive pruning in earlier layers of each block. After pruning, the model is retrained on the same dataset to recover lost accuracy, typically involving a single round of filter removal followed by a fixed number of training epochs to stabilize performance.

1.3 Network Slimming. ***Learning Efficient Convolutional Networks through Network Slimming. [3] [7]*** This method primarily works by inducing channel-level sparsity through a structured training process that involves applying L1 regularization on the scaling factors within batch normalization layers. During training, channels with small scaling factors are identified as less important and are subsequently pruned, resulting in a more compact network. Note that for the model provided, there were no batch normalization layers, so we instead took the L2 norm of each convolution layer channel after training. This approach is advantageous as it does not require any specialized hardware or software accelerators for the resulting models, can be integrated seamlessly into existing CNN architectures, and adds minimal overhead to the training process. The network slimming process involves a few key steps: initially training the network with sparsity-induced regularization on channel scaling factors to encourage reduction of less critical channel connections, pruning the identified insignificant channels based on their scaling factors, and fine-tuning the pruned network to restore or even enhance its performance.

1.4 Hybrid Method. ***Magnitude Pruning and Pruning Filters.*** In this method, we pulled from two separate papers (magnitude pruning and filter pruning) in order to create a pruning method that fits a small network well. Unlike magnitude pruning in section 1.1, which decides to include or remove particular connections, filter pruning in section 1.2 removes entire 3-dimensional filters at a time (channels * w_in * h_in), based on a threshold determined by all of a network's filters' L1 norms. While this is a good method on larger networks like ResNet and VGG, it caused large losses in accuracy on our (relatively) smaller network, which only had 4 convolutional layers with 3 filters in each. As such, we drew inspiration from the filter-by-filter scrutiny implemented in filter pruning, as well as the selection of individual weights from magnitude pruning, to create a combined method. In this method, we look at each of the 3 3-dimensional filters in our 4 convolutional layers, and prune all weights below a certain threshold in each of these filters, and then fine-tune. The results of this method can be seen in section 2 below.

Implementation Details: Fine Tuning. In order to allow our model to fine-tune to the new, sparser connections, we wrote a minibatch gradient descent training loop, ran for 20 epochs with a small learning rate (1e-6 with 1e-8 weight decay), that applied a mask re-enforcing the pruned connections at the end of every epoch. While this detracted from the computational speedup typically associated with pruning, we were able to fine tune our model and ensure that the weights we pruned remain pruned during retraining.

2 Comparison

2.1 Ease of Implementation. When it comes to ease of implementation, magnitude pruning was the winner, as there was only one threshold value that needed to be calculated at the very end of training, and that was used to generate all masks. Filter pruning and the hybrid method follow in ease of implementation, as they both required extracting convolutional layers and looping over filters. As such, the most difficult method to implement was network slimming, as it required not only extracting convolutional layers and delving into filters, but also delving into individual channels of each filter, and maintaining all of that information. Additionally, because our model did not have any batch normalization layers build into it, we also had to brainstorm how to implement this method such that it still applies to our model.

2.2 Performance. *Note: Larger versions of these figures can be found in the appendix.*

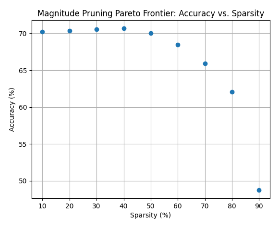


Fig 1: Magnitude Pruning. With L1-based magnitude pruning, accuracy remains nearly the same up until we hit 60% sparsity, at which point accuracy begins to decline. We may see this trend because magnitude pruning employs fine-tuning after connections have been pruned, allowing for the network, at least at low pruning rates, to largely recover its level of accuracy. However, once enough connections have been pruned, even fine tuning does not help the model regain its original accuracy levels.

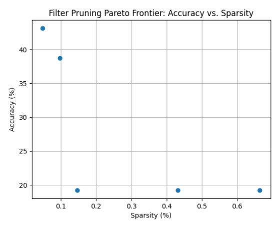


Fig 2: Pruning Filters. First off, we notice that all sparsity levels are below 1% with this method. This is because filter pruning only prunes parameters within the convolutional layers, maintaining full connection between all other layers. Additionally, we notice that initially accuracy is stable around 43% post fine-tuning; even with minor increases in sparsity (ex. pruning one filter), our network has suffered a large loss. As sparsity increases, albeit minimally, we see accuracy declining more and more, until accuracy stabilizes at around 19% past a certain sparsity threshold.

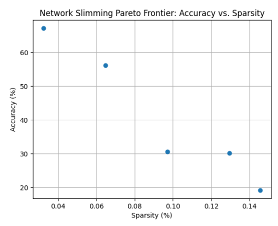


Fig 3: Network Slimming. Network slimming runs into similar issues to filter pruning when it comes to sparsity levels; because we are only removing some channels within some filters from our convolutional layers, it is difficult to achieve high sparsity on the given model. With the lowest levels of sparsity, we see that most accuracy ($\tilde{6}7$%) is retained. However, accuracy drops drastically with even the smallest decreases in sparsity, reaching around 19% with sparsity as low as 0.14%.

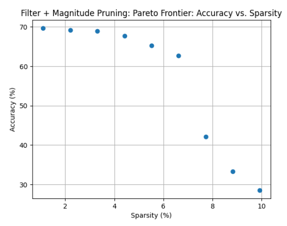


Fig 4: Hybrid Method Just like the filter pruning method, we notice that all our sparsity levels are quite low. However, because we prune individual weights within the filters (rather than entire filters) based on magnitude, our sparsity levels in general are higher than here the standalone filter pruning method (up to 10%). The hybrid method also maintains a higher accuracy than standalone filter pruning; similar to the trend with magnitude pruning, we see accuracy levels declining slowly up until 6% - 7% sparsity (or 60% - 70% prune rates for convolutional layers), and then declining sharply afterwards.

Easily, magnitude pruning was the best-performing method, as it investigated the relative significance of all weights, rather than just those in convolutional layers (like the other 3 methods). With an accuracy loss of below 2%, even when 50% of connections are pruned, magnitude pruning outperforms, by a large margin, every other method we tested.

We anticipate that methods like network slimming and pruning filters may be more effective on larger networks, where there are more filters, and thus more channels, to prune. However, because our model only had 4 convolutional layers, 3 filters per layer (total 12 3D filters), and 3 channels per filter (total 36 2D channels), pruning even 5-10% of the filters or channels was damaging to accuracy, even after fine-tuning. Simply put, on a model of this size, pruning entire filters and channels simply did too much damage to accuracy for these methods to be viable.

The hybrid method, while it performed better than filter pruning alone, due to only being able to prune the convolutional layers in the model, also did not achieve high sparsity rates in the model. In terms of accuracy, the hybrid model performs worse than magnitude pruning, even though it shows a similar pareto frontier trend. While magnitude pruning sets a pruning threshold based on the entire network, our hybrid method sets thresholds by filter. This means that every filter gets some percentage of weights pruned, even if those weights may have been significant in the global context of the model. While we had high hopes for this method, as hybrid pruning is often beneficial in larger models, where the different techniques can complement each other by removing redundant filters, low-magnitude weights, and less important channels, the size of this model, among other factors, may have hindered the performance of this method.

3 Reflection

For the project, we conducted filter pruning, magnitude pruning, network slimming pruning, and a hybrid of these techniques. The results were quite interesting and not entirely as expected. Magnitude pruning turned out to be the most effective, which aligns with common findings in pruning literature. It's known for its simplicity and often achieves good results by removing weights with the smallest absolute values.

While we didn't expect the simplest method to achieve the best results, we were also shocked by how far behind more complicated methods, such as filter pruning, network slimming, and our hybrid method trailed. This outcome can likely be attributed to the small number of convolutional layers in the model. In larger models, filter and channel pruning are more effective because there are more redundant filters and channels that can be safely removed without significantly impacting performance. However, in smaller models, each filter plays a more critical role, so removing even 1-2 can lead to significant performance drops. While the hybrid method avoided accuracy drops by removing individual weights within filters, it still trailed behind in sparsity and final score, because we were only looking at convolutional layers.

Overall, this project provided us valuable insights into the effectiveness of various pruning techniques, especially in the context of model size; it has highlighted to us the importance of model architecture when choosing a pruning strategy.

References

[1] Song Han, Jeff Pool, John Tran, William J. Dally. *Learning both Weights and Connections for Efficient Neural Networks*, 2015. arXiv:1506.02626 [cs.NE].

[2] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, Hans Peter Graf. *Pruning Filters for Efficient ConvNets*, 2017. arXiv:1608.08710 [cs.CV].

[3] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, Changshui Zhang. *Learning Efficient Convolutional Networks through Network Slimming*, 2017. arXiv:1708.06519 [cs.CV].

[4] jack-willtuner. *deep-compression(PyTorch)*, 2020. Available at: `https://github.com/jack-willturner/deep-compression`.
Note: We used this source as a reference on how to modularize our code for pruning, as well as what functionalities were available to use in PyTorch. However, since we wrote our implementation in Tensorflow, no code was pulled from this repository.

[5] ChloeeeYoo. *SimplePruning-PyTorch*, 2020. Available at: `https://github.com/ChloeeeYoo/SimplePruning-PyTorch`.
Note: We used this source as a guide for implementing our magnitude pruning method. However, since we wrote our implementation in TensorFlow and this was in PyTorch, no code was pulled from this method.

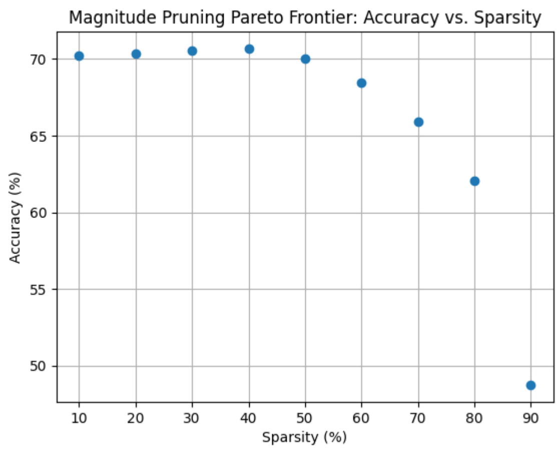
[6] tyui592. *Pruning Filters for Efficient ConvNets*, 2019. Available at: `https://github.com/tyui592/Pruning_filters_for_efficient_convnets`.
Note: We used this source as a guide for implementing our filter pruning method. However, since we wrote our implementation in TensorFlow and this was in PyTorch, no code was pulled from this method.

[7] Culturenotes. *Network-Slimming*, 2017. Available at: `https://github.com/Culturenotes/Network-Slimming/tree/master`.
Note: We used this source as a guide for implementing our Network Slimming method. However, since we wrote our implementation in TensorFlow and this was in PyTorch, no code was pulled from this method.

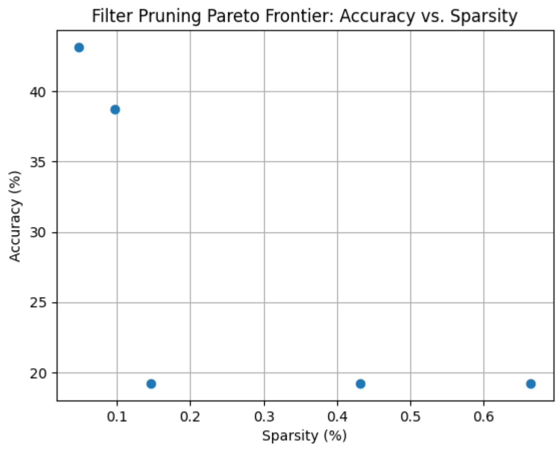
[8] Tensorflow Keras. *Writing a Training Loop From Scratch.* Available at: `https://www.tensorflow.org/guide/keras/writing_a_training_loop_from_scratch`.
Note: We heavily referenced this documentation in order to implement a training loop from scratch, and altered the code based on our needs (ex. applying masks, using different optimizers, etc.)

A Appendix

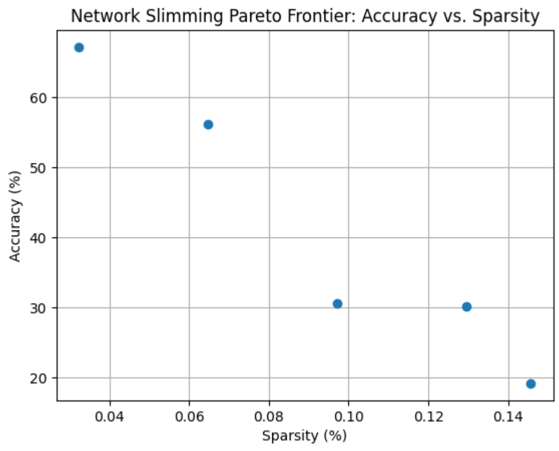
Magnitude Pruning



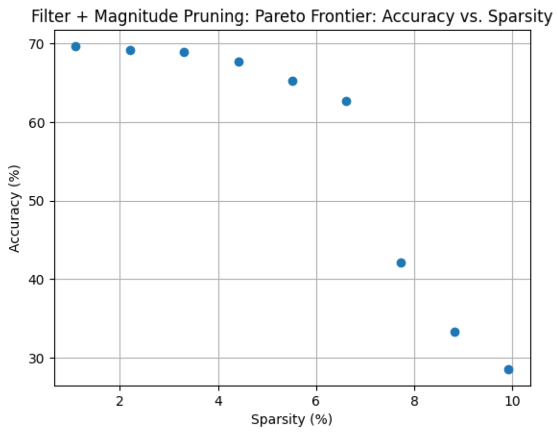
Filter Pruning



Network Slimming



Hybrid Pruning


Filter + Magnitude Pruning: Pareto Frontier: Accuracy vs. Sparsity
Accuracy (%)
Sparsity (%)