Improving Pruning Filters for Efficient ConvNets

Cenk Alkan

April 2024

1 Introduction

This report begins with a summary of *Pruning Filters for Efficient ConvNets* [1], followed by a presentation of my improvements to the program described in that paper.

Pruning Filters for Efficient ConvNets [1] was presented at *ICLR 2017* by Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf.

2 Original Paper

2.1 Research Question

The research question posed in [1] is: How can the computational and parameter storage costs of convolutional neural networks (CNNs) be reduced while preserving model performance as these networks grow deeper and more complex? This question is critical because reducing computational costs is essential for many CNN applications, including mobile and cloud services.

2.2 Challenge Analysis

Previous research has also focused on reducing computational costs by compressing models through various methods. Some approaches prune individual weights by selectively removing those that contribute the least to the output, often based on criteria such as the smallest magnitude. While this method is effective at reducing the parameter count in fully connected layers, it does not necessarily lead to significant reductions in computational time for convolutional layers, which dominate the computation in CNNs. Furthermore, this approach creates sparse matrices that require specialized computing libraries to process efficiently, and such libraries may not be universally available or fully optimized.

2.3 Philosophy

The philosophy of the paper was to implement pruning at the level of entire filters rather than individual weights. This approach aimed to achieve two key

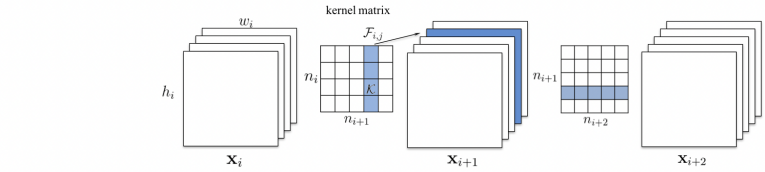


Figure 1: Filter pruning removes kernels not just in the next layer, but also in subsequent layers.

objectives: preserving existing connectivity patterns and significantly reducing computational costs. By maintaining existing connectivity patterns, the method offered a simpler solution with the potential to better preserve program accuracy. Reducing computational costs was particularly desirable as it decreased total running time, enhancing both speed and efficiency.

2.4 Solution

The proposed solution is to remove redundant filters and retrain the model to recover from any accuracy loss caused by the removal of filters. The key idea is to select and prune filters that have a small effect on output accuracy. The authors measure this effect by calculating the L1-norm of each filter. Filters are ranked based on their L1-norm, and a predetermined percentage of filters with the lowest values are pruned away. Layers with the same input sizes often have similar sensitivity to pruning, so the same pruning ratio is applied to them. For layers that are more sensitive, a smaller pruning rate is used or pruning is skipped entirely.

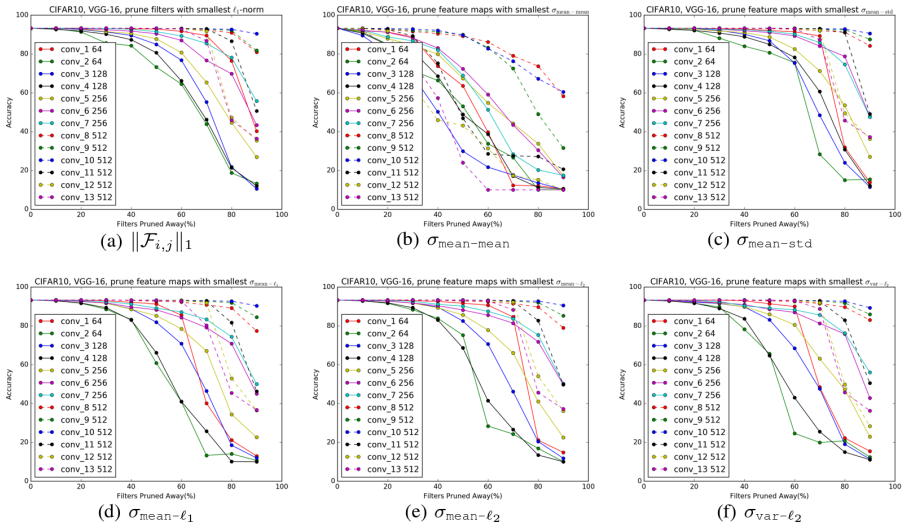
When a filter $F_{i,j}$ is pruned, its corresponding feature map $x_{i+1,j}$ is removed, reducing the number of operations required. The kernels applied to those removed feature maps in subsequent convolutional layers are also removed, saving additional operations. Pruning m filters of layer i reduces the computation cost for both layers i and $i+1$. This significantly reduces overall computation cost.

The authors adopted a one-shot pruning and retraining strategy, pruning multiple layers at once and retraining to quickly regain accuracy.

2.5 Experiments and Results

The authors conducted experiments across various CNN architectures and datasets, comparing their pruning method with alternatives such as random pruning or activation-based pruning. Results demonstrated that L1 norm is an effective indicator for determining which filters to prune.

When applying VGG-16 to CIFAR-10, pruning 50% of filters in layers 1 and 8–13 achieved a 34% reduction in FLOPs while slightly improving accuracy. ResNet-56 pruning yielded a 27.6% reduction with minimal accuracy loss, and ResNet-110 pruning achieved a 38.6% reduction. On ImageNet with ResNet-34,



(a) $\|\mathcal{F}_{i,j}\|_1$ (b) $\sigma_{\texttt{mean-mean}}$ (c) $\sigma_{\texttt{mean-std}}$

(d) $\sigma_{\texttt{mean-}\ell_1}$ (e) $\sigma_{\texttt{mean-}\ell_2}$ (f) $\sigma_{\texttt{var-}\ell_2}$

Figure 2: Comparison of activation-based feature map pruning.

Table 1: Overall results. The best test/validation accuracy during the retraining process is reported. Training a pruned model from scratch performs worse than retraining a pruned model, which may indicate the difficulty of training a network with a small capacity.

Model	Error(%)	FLOP	Pruned %	Parameters	Pruned %
VGG-16	6.75	3.13×10^8		1.5×10^7	
VGG-16-pruned-A	**6.60**	2.06×10^8	34.2%	5.4×10^6	64.0%
VGG-16-pruned-A scratch-train	6.88				
ResNet-56	6.96	1.25×10^8		8.5×10^5	
ResNet-56-pruned-A	6.90	1.12×10^8	10.4%	7.7×10^5	9.4%
ResNet-56-pruned-B	**6.94**	9.09×10^7	27.6%	7.3×10^5	13.7%
ResNet-56-pruned-B scratch-train	8.69				
ResNet-110	6.47	2.53×10^8		1.72×10^6	
ResNet-110-pruned-A	**6.45**	2.13×10^8	15.9%	1.68×10^6	2.3%
ResNet-110-pruned-B	6.70	1.55×10^8	38.6%	1.16×10^6	32.4%
ResNet-110-pruned-B scratch-train	7.06				
ResNet-34	26.77	3.64×10^9		2.16×10^7	
ResNet-34-pruned-A	27.44	3.08×10^9	15.5%	1.99×10^7	7.6%
ResNet-34-pruned-B	27.83	2.76×10^9	24.2%	1.93×10^7	10.8%
ResNet-34-pruned-C	27.52	3.37×10^9	7.5%	2.01×10^7	7.2%

Figure 3: Approximately 30% reduction in FLOPs for VGG-16 and ResNets without significant loss of accuracy.

pruning produced a 24.2% FLOP reduction with only about 1% accuracy loss.

Comparisons with other strategies (e.g., pruning the largest filters or activation-based pruning) showed that L1 pruning consistently outperformed alternatives, particularly at higher pruning ratios. Overall, L1-based pruning reduces computation and model size without degrading—and sometimes improving—accuracy.

3 My Improvements

3.1 Philosophy

My implementation builds upon *Pruning Filters for Efficient ConvNets* by introducing advancements that enhance efficiency while maintaining accuracy. Traditional pruning is static, removing filters post-training and requiring extensive retraining. My improvements center on dynamically clustering filters during training based on feature similarities, enabling real-time identification and pruning of redundant filters. This proactive methodology continuously refines the architecture as training progresses, minimizing computational overhead.

Additionally, the improvements incorporate multi-GPU training, which is well-suited to the dynamic approach. Clustering and pruning involve intensive tasks such as k-means clustering and cosine similarity calculations. Multi-GPU support distributes these tasks, accelerating the process and enabling scalability to larger models and datasets. Together, these enhancements maintain or improve predictive capability while significantly reducing computational cost.

3.2 Solution

My solution integrates dynamic online clustering and pruning into the training process, allowing for real-time architectural adjustments. Filters are clustered during training using cosine similarity metrics and k-means clustering, and those with minimal contribution to accuracy are removed. This reduces training computation and eliminates the need for extensive post-training pruning and retraining.

To implement this effectively, I optimized the training process with a learning rate schedule, starting at 0.02 and reducing it fivefold every 60 epochs. This schedule matches the evolving complexity of the network, ensuring stability as pruning modifies the architecture. Data augmentation (random cropping, horizontal flipping) and normalization further enhance generalization and robustness.

The methodology combines two strategies: `update_clusters()` and `cluster_and_prune()`. The first dynamically groups filters without altering network weights, particularly effective early in training. The second prunes filters at milestones once the network stabilizes. This combination continuously optimizes the structure, while multi-GPU training accelerates the process.

Trial	Top-1 Accuracy	Top-5 Accuracy	Inference Time (s)
1	89.24	99.41	19.14
2	89.44	99.48	19.79
3	90.12	99.58	19.05
4	88.27	99.39	18.98
5	89.44	99.53	19.67

Figure 4: Results for the original algorithm presented in [1], five trials.

Trial	Top-1 Accuracy	Top-5 Accuracy	Inference Time (s)	Clusters
1	88.53	99.45	4.74	5
2	88.34	99.52	5.30	5
3	88.71	99.53	5.06	5
4	89.03	99.55	4.50	5
5	88.76	99.61	4.58	5
...	...	...	...	...

Figure 5: Results for my algorithm across cluster sizes ($k = 5$ to $k = 30$).

3.3 Experiments

I compared my algorithm against the original, conducting five trials per cluster size ($k = 5$ to $k = 30$). I also ran five trials of the original algorithm. Metrics included top-1 and top-5 accuracy as well as inference time. All experiments used batch size 128, 100 epochs, and the Brandeis HPCC cluster with NVIDIA TitanX GPUs and Intel Xeon CPUs.

3.4 Results

The inference time for the original algorithm (Figure 4) was consistently about five times longer than for my algorithm (Figure 5). Top-1 and Top-5 accuracy were similar across both algorithms and across cluster sizes.

Statistical testing confirmed these results. For inference time, a Kruskal–Wallis test yielded $p \approx 0.033$, providing significant evidence of differences. Post-hoc Dunn tests (Holm correction) showed $p < 0.02$ for all comparisons between my algorithm and the original, confirming that my version ran faster across all cluster sizes.

For Top-1 accuracy, $p \approx 0.19$; for Top-5 accuracy, $p \approx 0.87$. Thus, performance was not significantly degraded. My version achieved an average fivefold speedup in inference while preserving accuracy, confirming the effectiveness of integrating clustering and pruning during training.

4 Conclusions

I demonstrated that a modified version of the original algorithm achieved significantly faster performance without sacrificing accuracy. Future work could expand trial counts, explore more datasets beyond CIFAR-10, and evaluate additional architectures such as ResNet. These directions would provide broader evidence for the generality of the approach.

References

[1] Hao Li et al. *Pruning Filters for Efficient ConvNets*. 2017. arXiv: 1608.08710 [cs.CV].