CUP: Cluster Pruning for Compressing Deep Neural Networks

Rahul Duggal
Georgia Institute of Technology
rahulduggal@gatech.edu

Cao Xiao
Amplitude
danica.xiao@amplitude.com

Richard Vuduc
Georgia Institute of Technology
richie@cc.gatech.edu

Duen Horng Chau
Georgia Institute of Technology
polo@gatech.edu

Jimeng Sun
University of Illinois at Urbana Champaign
jimeng@illinois.edu

***Abstract*—We propose CUP, a new method for compressing and accelerating deep neural networks. At its core, CUP achieves compression by clustering and pruning similar filters in each layer. For clustering, CUP uses hierarchical clustering which allows for an elegant parameterization of model capacity through a single hyper-parameter t. We observe that by increasing t, CUP can dynamically reduce model capacity through non-uniform layer-wise pruning leading to two advantages. First, CUP can effectively compress a model to within the desired compute budget through a simple line-search on t. Second, through a simple extension, CUP can obtain the pruned model in a single training pass leading to large savings in training time. On Imagenet, CUP leads to a $2.47\times$ FLOPS reduction on Resnet-50 with less than 1% drop in top-5 accuracy. Notably, in the retrain-free setting, CUP-RF saves over 10 hours of training time on 3 GPUs, in comparison to state-of-the-art methods. The code for CUP is open sourced[1].**

***Index Terms*—Pruning, Compact, Efficient, Neural Net**

I. Introduction

Neural network compression is a critical enabler for the deployment of powerful deep neural networks (DNNs) on the edge. There are several ways to compress a DNN, including pruning [1]–[3], low rank approximation [4], [5], knowledge distillation [6], [7] and quantization [8], [9]. In this paper, we focus on the problem of channel pruning which, in a nutshell, aims to delete the "unimportant" filters of a neural network.

A typical channel pruning pipeline consists of three steps [1]: (1) train the target DNN for some task, *e.g.*,image classification; (2) identify and delete unimportant filters based on an importance criterion; (3) retrain the pruned network to recover accuracy lost due to pruning. This pipeline presents two challenges: *C1–non-uniform pruning*: determining the optimal layerwise pruning amount in step 2 is a combinatorial problem, and intractable for modern DNNs with hundreds of layers; and *C2–long runtime*: the retraining performed in step 3 slows down the pruning pipeline. To tackle *C1*, a line of work [2], [10], [11] prunes the network uniformly across each layer (*e.g.*,delete 50% filters in each layer), but this has shown to be sub-optimal [12]. Other works use heuristics that are either computationally expensive [2], [13] or involve many hyper-parameters, which are difficult to tune [12]. For tackling *C2*, recent methods [14], [15] modify the pruning pipeline by interleaving the pruning and training steps (*i.e.*,merging steps 1 and 2), thereby eliminating the need for retraining—we refer to these as *retrain-free* methods.

With CUP, we enable layer-wise non-uniform pruning (addressing *C1*) whilst introducing only a single hyper-parameter t. At its core, CUP employs hierarchical clustering to cluster similar filters in each layer. Pruning is then achieved by replacing each cluster by a representative filter. A key advantage of our method is that the clustering strategy used offers a principled way to determine the appropriate number of clusters in each layer. Empirically, we find that t allows for a smooth parameterization of the pruning amount. We leverage this observation to extend CUP to the retrain-free setting (addressing *C2*). Essentially, by gradually increasing t during the initial training phase, CUP-RF (RF for retrain-free) can incrementally prune a target model within one training pass. This leads to large savings in training time, *e.g.*,saving over 14 hours while training a ResNet-50 on ImageNet.

To summarize, our contributions in this paper are 1) We propose CUP as a method for compressing deep neural networks with the benefit of enabling non-uniform pruning through a single hyper-parameter *t*. 2) We extend CUP to CUP-RF whereby filters are pruned in the initial training pass itself resulting in large savings of time cost during pruning. 3) We comprehensively compare our methods to the state-of-the-art methods on large datasets (*e.g.*,Imagenet).

II. Related Work

At a high level, pruning methods can be categorized based on whether they lead to *unstructured* [16]–[19] or *structured* [2], [3], [10], [11], [20] sparsity in the pruned network's weights. The latter reaps the benefits of pruning (*e.g.*,lower flops, faster inference) through a matrix reshaping operation completely avoiding the need of custom hardware as typically required by the former. Keeping this advantage in mind, we adopt structured pruning for CUP.

[1]https://github.com/duggalrahul/CUP_Public

978-1-6654-3902-2/21/$31.00 ©2021 IEEE

Within structured pruning, a promising research direction is *channel pruning* where the aim is to prune entire filters. Existing channel pruning algorithms primarily differ in the criterion used for identifying pruneable filters. Examples of such criterions include pruning filters; with smallest L1 norm of incoming weights [2]; with largest average percentage of zeros in their activation maps [10]; using structured regularization [11], [21]; or with least discriminative power [3]. A drawback of these methods [2], [20] is that the number of filters to prune in each layer is a hyper-parameter leading to a combinatorial search space.

Another aspect of comparison is based on the time cost of pruning. Traditionally, channel pruning employs a three-step regime which leads to a high time cost (ref. *C2* in introduction). Recent works [14], [15], [22], [23] address this issue by doing away with the retraining phase altogether and we refer to these as retrain-free methods. Compared to these works [14], [15], CUP-RF can further reduce the training time. This is due to the fact that CUP-RF *permanently* prunes some filters during training whereas previous methods continue to train the pruned filters. This distinction leads to an additional saving of up-to 10 hours for pruning a Resnet-50 on Imagenet.

III. The CUP framework

The CUP pruning algorithm is a three-step process and is outlined in Fig. 1. The first step computes features that characterize each filter. These features are specific to the layer type (fully connected or convolutional) and are computed from the incoming and outgoing weight connections. The second step clusters similar filters based on the features computed previously. The third and last step chooses a single representative filter from each cluster and prunes all others.

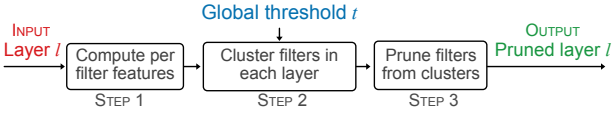


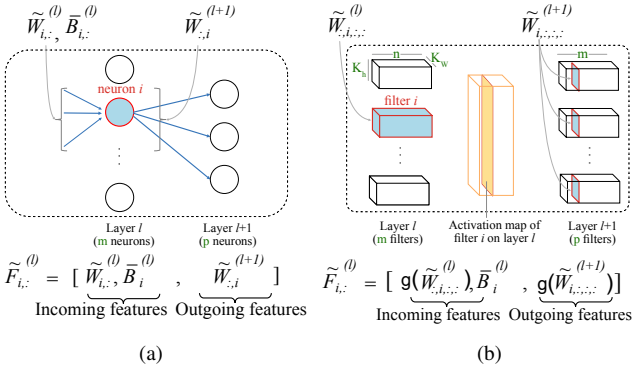
Fig. 1: Three steps of the Cluster Pruning (CUP) algorithm.

A. Compute per-filter features (step 1)

The first step of CUP computes features that characterize each filter. These features are computed using both the *incoming* and *outgoing* connections of a filter. For further discussion, we assume layers $l-1, l$ and $l+1$ of the neural network contain n, m and p filters respectively.

Fully Connected Layers (Fig. 2a). The l^{th} fully connected layer is parametrized by weights $\widetilde{W}^{(l)} \in \mathbb{R}^{m \times n}$ and bias $\overline{B}^{(l)} \in \mathbb{R}^{m}$. For neuron i within this layer, we define it's feature set $\widetilde{F}^{(l)}_{i,:} \in \mathbb{R}^{n+p+1}$ as

$$\widetilde{F}^{(l)}_{i,:} = \text{concat}(\underbrace{\widetilde{W}^{(l)}_{i,:}, \overline{B}^{(l)}_{i}}_{\text{Incoming features}} , \underbrace{\widetilde{W}^{(l+1)}_{:,i}}_{\text{Outgoing features}}), \qquad (1)$$



(a) (b)

Fig. 2: Computing features from the incoming and outgoing weights of (a) fully-connected layer, (b) convolution layer.

where concat concatenates two vectors into one.

Convolutional Layers (ref. Fig. 2b). The l^{th} convolutional layer is completely parameterized by the 4-D weight tensor $\widetilde{W}^{(l)} \in R^{n \times m \times k_h \times k_w}$ and the bias vector $\overline{B}^{(l)} \in \mathbb{R}^{m}$. The four dimensions of $\widetilde{W}^{(l)}$ correspond to - number of input channels (n), number of filters in layer l (m), height of filter (k_h) and width of filter (k_w). For filter i within this layer, we define its feature set $\widetilde{F}^{(l)}_{i,:} \in \mathbb{R}^{n+p+1}$, as

$$\widetilde{F}^{(l)}_{i,:} = \text{concat}(\underbrace{g(\widetilde{W}^{(l)}_{:,i,:,:}), b^{(l)}_{i}}_{\text{Incoming features}}, \underbrace{\widetilde{W}^{(l+1)}_{i,:,:,:}}_{\text{Outgoing features}}), \qquad (2)$$

$$\text{where } g(\widetilde{X}_{:,:,:}) = [\|\widetilde{X}_{1,:,:}\|_F, \ldots, \|\widetilde{X}_{C,:,:}\|_F]. \qquad (3)$$

Here $g : \mathbb{R}^c \times \mathbb{R}^d \times \mathbb{R}^e \rightarrow \mathbb{R}^c$ computes the channel-wise frobenius norm of any arbitrary 3D tensor $\widetilde{X}$.

B. Cluster filters in each layer (step 2)

Given feature vectors $\widetilde{F}^{(l)}_{i,:}$ for each filter i in layer l, step 2 clusters filters within a layer using agglomerative hierarchical clustering [24, Chapter 15]. This specific choice of clustering affords a **key benefit**: The number of clusters in each layer can be *jointly* controlled using a single hyper-parameter *t*. In contrast some recent works use other clustering techniques such as K-means++ [25] or spectral clustering [26]. These works face the combinatorial challenge of deciding the appropriate number of clusters in each layer and are limited to uniform pruning.

With hierarchical clustering, the clustering operation for layer l begins by building a weighted binary tree representation (dendrogram) for that layer. Assume that layer l contains m filters. The algorithm starts off with m clusters $\mathbb{C}^{(l)}_i \forall i \in [1, m]$ i.e. each filter is a separate cluster. Then it iteratively builds the tree by merging two closest clusters as per the *Wards variance minimization* criterion. The criterion specifies to merge two clusters $\mathbb{C}^{(l)}_p, \mathbb{C}^{(l)}_q$ that lead to the least decrease in intra-cluster variance over all possible pairings of clusters in $\mathbb{C}^{(l)}$. The output of this phase is a weighted binary tree, or dendrogram, whose each non-leaf node specifies a cluster of filters while the edge weights encode the distance or dissimilarity between its children.

Determining the number of clusters: After constructing the dendrograms for each layer, we use edge weights to jointly determine the number of clusters in that layer. Specifically, the dendrograms for all layers are chopped at the same height t which is a hyper-parameter. The higher the value of t, fewer the number of clusters. Since CUP replaces each cluster with a filter (presented in the next section), increasing t ultimately leads to fewer remaining filters, or higher compression. To summarize, the output of step 2 is a set of $n^{(l)}$ clusters $\mathbb{C}^{(l)}$ for each layer l, such that $|\mathbb{C}^{(l)}| = n^{(l)}$.

C. Prune filters from each cluster (step 3)

The third and last step chooses the representative filter from each cluster and prunes all others. Given the set of filter clusters $\mathbb{C}^{(l)}$ for layer l, we formulate pruning as a subset selection problem. The idea is to select the most representative subset of filters $\mathbb{S}_r^{(l)}$ from each filter cluster $\mathbb{C}_r^{(l)} \in \mathbb{C}^{(l)}$. Pruning then corresponds to replacing all filters in $\mathbb{C}_r^{(l)}$ by $\mathbb{S}_r^{(l)}$. Motivated by prior work, several subset selection criterion can be formulated as below.

- **Norm based** : [2] prune filters based on the $l1$ norm of incoming weights. This criterion amounts to selecting the top $k\%$ filters having the highest feature norm as the cluster representative.
- **Zero activation based** : [10] prune filters based on the average percentage of zeros (ApoZ) in their activation map when evaluated over a held-out set. This criterion amounts to choosing the top $k\%$ filters having least ApoZ as the cluster representative.
- **Activation reconstruction based** : [20] prunes filters based on its contribution towards the next layer's activation. This criterion amounts to choosing the top $k\%$ filters having maximum contribution.

In our work, we use a norm based criterion to select a subset $\mathbb{S}_r^{(l)}$ from a cluster of filters $\mathbb{C}_r^{(l)}$ with filter i having features $\widetilde{F}_{i,:}^{(l)}$. This criterion is described through the equation

$$\mathbb{S}_r^{(l)} = \underset{i \in \mathbb{C}_r^{(l)}}{\text{argmax}} \|\widetilde{F}_{i,:}^{(l)}\|_2, \tag{4}$$

where argmax implies that we select a single representative neuron from each cluster. Thus, post pruning, the number of clusters in layer l equals the number of remaining filters.

D. Extension to retrain free setting (CUP-RF)

Similar to previous methods, CUP achieves compression through a three step pipeline involving training, pruning and retraining. However, with a slight modification, CUP can completely avoid any fine-tuning whatsoever. The modified algorithm is termed CUP-RF for CUP "Retrain-Free". The idea is to gradually reduce the model capacity during the initial training phase. This is achieved by calling CUP at the beginning of each epoch, with a monotonically increasing schedule for t. We find that the following linear schedule for $t(e)$ (value of t at epoch e) suffices for good performance.

$$t(e) = k.e + b \tag{5}$$

Here k, b are hyper-parameters controlling the slope and offset of the linear pruning schedule and are determined through a linesearch.

IV. Experiments

A. Training details, base models & evaluation metrics

We evaluate our methods against prior art on CIFAR-10 and ImageNet. For all networks trained on CIFAR, we use the training hyper-parameter settings from [11]: a batch size of 64, a weight decay of 10^{-4}, total epochs of 160, and an initial learning rate of 0.1 which is divided by 10 at epochs 80 and 120. On ImageNet, we train with a batch size of 256 for 90 epochs. The initial learning is set to 0.1, which is divided by 10 at epochs 30 and 60. We use a weight decay of 10^{-4}. After compression, the pruned model is retrained with a tenth of the initial learning rate while other hyper-parameters remain the same. Our baseline models are:

- **VGG-16** [27]: trained up to 93.64% on CIFAR-10.
- **ResNet-56** [28]: trained up to 93.67% on CIFAR-10.
- **ResNet-{18,34,50}** [28]: We use the official PyTorch implementations where the models have 69.87%, 73.59% and 75.86% Top-1 accuracies on ImageNet respectively.

To measure compression, we use the standard flops reduction metric defined as $FR = \frac{F_{base}}{F_{compressed}}$ where F is number of multiply and adds to score one input.

B. Verifying two key benefits of pruning with CUP

Single hyper-parameter control. In Figure 3, we plot the number of remaining filters in each layer after pruning a VGG-16 on Cifar-10 with CUP ($t = 0.9$). Using a single hyper-parameter t, CUP accomplishes non-uniform pruning across the layers (see green bars). As t increases, CUP offers a desirable, largely monotonic effect on: (1) test accuracy reduction; (2) parameter reduction; (3) flops reduction; and (4) CPU wall-clock speedup, as show in Figures 4a–d.

Training time speedup. Table I shows our approach (bold font) achieves the shortest training time, best top-1 accuracy, and the most flop reduction, in both the *retrain-allowed* (marked with ✓) and the *retrain-free* settings (✗), for a ResNet-50 trained on ImageNet. In the *retrain-free* setting (Table I, bottom row), CUP-RF saves 14 hours when compared to that of the uncompressed model (51.6 v.s. 66 hours). All the

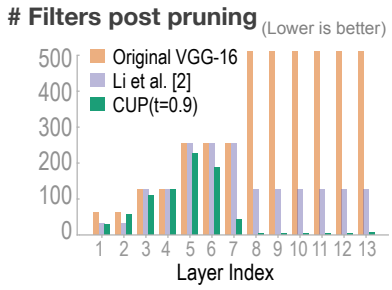


Fig. 3: The number of filters remaining in each layer post pruning a VGG-16 on CIFAR-10.

(a) (b) (c) (d)

Fig. 4: Compressing a VGG-16 on CIFAR-10 using CUP. We show the effect of pruning with a larger t on (a) test accuracy, (b) parameters, (c) flops, and (d) inference wall-clock speedup.

retrain-free methods generally save over 60 GPU hours compared to their traditional 3-step *retrain-allowed* counterparts. However, *retrain-allowed* models are generally more compact and also achieve a better Top-1 accuracy.

Method	Retrain?	Top-1 (%)	FR (×)	Training Time (GPU Hours)
Resnet-50	-	75.86	1.00	66.0
SFP [14]	✓	62.14	2.39	122.4
GM [15]	✓	74.83	3.81	122.7
CUP (ours)	✓	**75.07**	**3.86**	**116.8**
SFP [14]	✗	74.01	1.73	61.8
GM [15]	✗	74.13	2.15	62.2
CUP-RF (ours)	✗	**74.34**	**2.21**	**51.6**

TABLE I: For a ResNet-50 trained on ImageNet, our approach (bolded) achieves the shortest training time, best top-1 accuracy, and most flop reduction, in both the *retrain-allowed* (✓) and the *retrain-free* settings (✗). FR means *flops reduction*.

Method	Retrain?	ResNet-56		VGG-16	
		FR (×)	Acc (Δ%)	FR (×)	Acc (Δ%)
L1 [2]	✓	1.37	-0.02	1.51	-0.15
CP [20]	✓	2.00	-1.00	2.00	-0.32
GM [15]	✓	2.10	-0.33	-	-
GAL [22]	✓	2.45	-1.68	1.82	-0.54
NS [11]	✓	-	-	2.04	-0.32
CUP (ours)	✓	**2.77**	-0.40	**3.70**	-0.70
SFP [14]	✗	2.10	-1.33	-	-
GM [15]	✗	2.10	-0.70	-	-
VCNP [23]	✗	1.25	-0.78	1.64	-0.07
GAL [22]	✗	1.59	-0.28	1.82	-3.18
CUP-RF (ours)	✗	**2.12**	-0.31	**3.15**	-0.40

TABLE II: For ResNet-56 and VGG-16 models trained on CIFAR-10, our approach (bolded) leads to highest flops reduction (FR), in both the *retrain-allowed* (✓) and the *retrain-free* settings (✗). FR means *flops reduction*.

C. Comparison results on CIFAR-10 and ImageNet

Tables II and III present the results for compressing ResNet-56, VGG-16 models on CIFAR-10 and ResNet-{18,34,50} models on ImageNet, under the *retrain-allowed* and *retrain-free* settings.

Model	Method	Retrain?	FR (×)	Acc. (Δ%)	
				Top-1	Top-5
ResNet-18	GM [15]	✓	1.71	-1.87	-1.15
	COP [29]	✓	**1.75**	-2.48	-
	CUP (Our)	✓	**1.75**	**-1.00**	**-0.79**
	SFP [14]	✗	1.71	-3.18	-1.85
	GM [15]	✗	1.71	-2.47	-1.52
	CUP-RF (ours)	✗	**1.75**	**-2.37**	**-1.40**
ResNet-34	L1 [2]	✓	1.31	-1.06	-
	GM [15]	✓	1.69	-1.29	-0.54
	CUP (ours)	✓	**1.78**	**-0.86**	**-0.53**
	SFP [14]	✗	1.69	-2.09	-1.29
	GM [15]	✗	1.69	-2.13	-0.92
	CUP-RF (ours)	✗	**1.71**	**-1.61**	**-0.89**
ResNet-50	SFP [14]	✓	2.15	-14.0	-8.20
	MP [30]	✓	2.05	-1.20	-
	CUP (ours)	✓	**2.47**	**-1.17**	**-0.81**
	SFP [14]	✗	1.71	-1.54	**-0.81**
	GM [15]	✗	2.15	-2.02	-0.93
	CUP-RF (ours)	✗	**2.20**	**-1.47**	-0.88

TABLE III: For ResNet-18/34/50 models trained on ImageNet, our approach (bolded) leads to highest flops reduction (FR) with minimal accuracy drop, in both the *retrain-allowed* (✓) and the *retrain-free* settings (✗). FR means *flops reduction*.

Retraining-allowed (rows with ✓). Under this traditional three-stage pipeline, our approach consistently leads to highest flops reduction with a lower drop in accuracy.

Retrain-free (rows with ✗). Even when retraining is not allowed, our approach leads to highest flops reduction with a comparable, or in many cases, lower drop in accuracy.

V. Conclusion

We proposed a new channel pruning based method for model compression that prunes entire filters based on similarity. We showed how hierarchical clustering can be used to enable layer-wise *non-uniform* pruning whilst introducing only a single hyper-parameter. Using multiple models and datasets, we demonstrated that CUP achieves the highest flops reduction with the least drop in accuracy. Further, CUP-RF leads to large savings in training time with only a small drop in performance. A limitation of the current work is that we used simple yet effective *linear trajectory* for setting t in CUP-RF.

REFERENCES

[1] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Neurips*, 2015, pp. 1135–1143.

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

[3] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu, "Discrimination-aware channel pruning for deep neural networks," in *Neurips*, 2018, pp. 883–894.

[4] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *arXiv preprint arXiv:1405.3866*, 2014.

[5] W. Wang, Y. Sun, B. Eriksson, W. Wang, and V. Aggarwal, "Wide compression: Tensor ring nets," *learning*, vol. 14, no. 15, pp. 13–31, 2018.

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

[7] E. J. Crowley, G. Gray, and A. J. Storkey, "Moonshine: Distilling with cheap convolutions," in *Neurips*, 2018.

[8] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Neurips*, 2015, pp. 3123–3131.

[9] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *CVPR*, 2018, pp. 2704–2713.

[10] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv preprint arXiv:1607.03250*, 2016.

[11] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *ICCV*. IEEE, 2017, pp. 2755–2763.

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

[13] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," in *ECCV*, 2018, pp. 784–800.

[14] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," *arXiv preprint arXiv:1808.06866*, 2018.

[15] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *CVPR*, 2019, pp. 4340–4349.

[16] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," *Neurips*, vol. 2, pp. 598–605, 1989.

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

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

[19] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through l_0 regularization," *arXiv preprint arXiv:1712.01312*, 2017.

[20] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *ICCV*, 2017.

[21] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Neurips*, 2016, pp. 2074–2082.

[22] S. Lin, R. Ji, C. Yan, B. Zhang, L. Cao, Q. Ye, F. Huang, and D. Doermann, "Towards optimal structured cnn pruning via generative adversarial learning," in *CVPR*, 2019, pp. 2790–2799.

[23] C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, and Q. Tian, "Variational convolutional neural network pruning," in *CVPR*, 2019, pp. 2780–2789.

[24] D. S. Wilks, *Statistical methods in the atmospheric sciences*. Academic press, 2011, vol. 100.

[25] L. Li, Y. Xu, and J. Zhu, "Filter level pruning based on similar feature extraction for convolutional neural networks," *IEICE TRANSACTIONS on Information and Systems*, vol. 101, no. 4, pp. 1203–1206, 2018.

[26] L. Li, J. Zhu, and M.-T. Sun, "A spectral clustering based filter-level pruning method for convolutional neural networks," *IEICE TRANSACTIONS on Information and Systems*, vol. 102, no. 12, pp. 2624–2627, 2019.

[27] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[29] W. Wang, C. Fu, J. Guo, D. Cai, and X. He, "Cop: Customized deep model compression via regularized correlation-based filter-level pruning," *arXiv preprint arXiv:1906.10337*, 2019.

[30] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, T. K.-T. Cheng, and J. Sun, "Metapruning: Meta learning for automatic neural network channel pruning," *arXiv preprint arXiv:1903.10258*, 2019.