A Greedy Hierarchical Approach to Whole-Network Filter-Pruning in CNNs

Kiran Purohit, Anurag Parvathgari and Sourangshu Bhattacharya



Dept. of Computer Science & Engineering
IIT Kharagpur

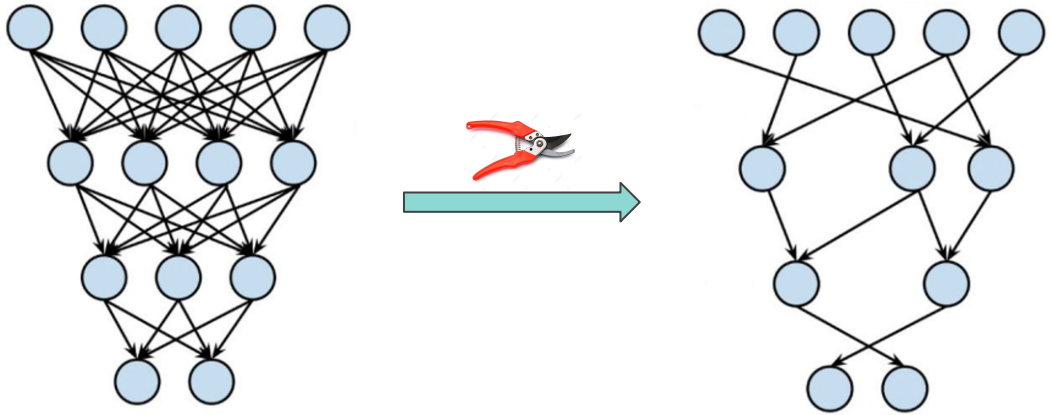
Introduction

Burden of CNNs ——ResNet-152	Filter Pruning ——Benefits
60.2 million parameters and 231MB storage spaces;	reduces the storage usage
380MB memory footprint	decreases the memory footprint
11.3 billion float point operations (FLOPs).	accelerates the inference

Introduction

Network Pruning

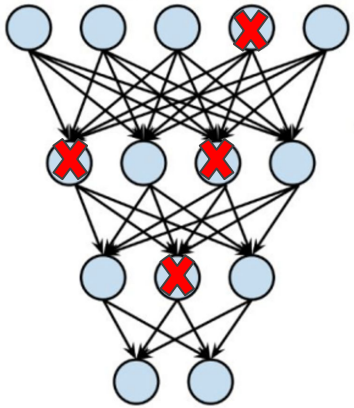
Given a pre-trained network Φ(.), the goal is to compress the network while maintaining the high performance as much as possible by removing the unnecessary parameters.



Pre-trained original network Φ(.)

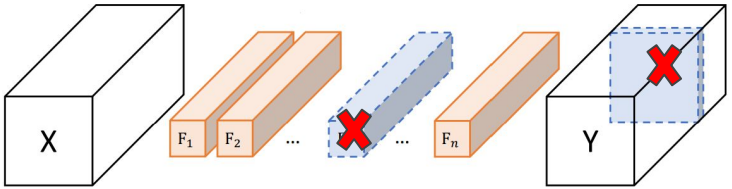
Final pruned network Φ'(.)

Network Pruning



Weights or Node Pruning

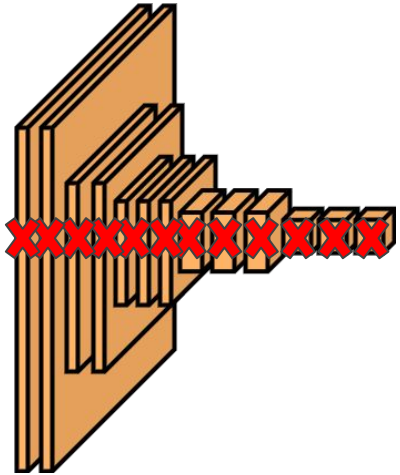
- Pruning applied to early DNN
- Check the importance of each weight or node
- Practical acceleration could not be achieved



Filter or Channel Pruning

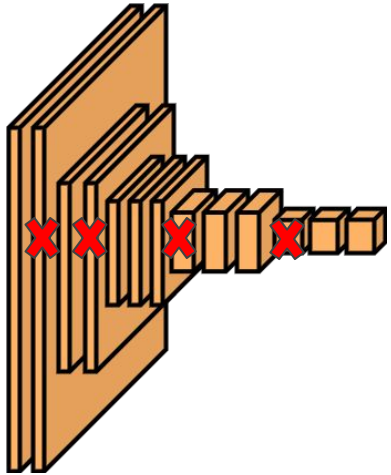
- Widely used for modern CNNs
- Remove the entire filter or channel at once
- Helps the practical acceleration of the network

Filter Pruning



Uniform Pruning

- Prune filters uniformly from each layer
- Process each layer independently and sequentially.



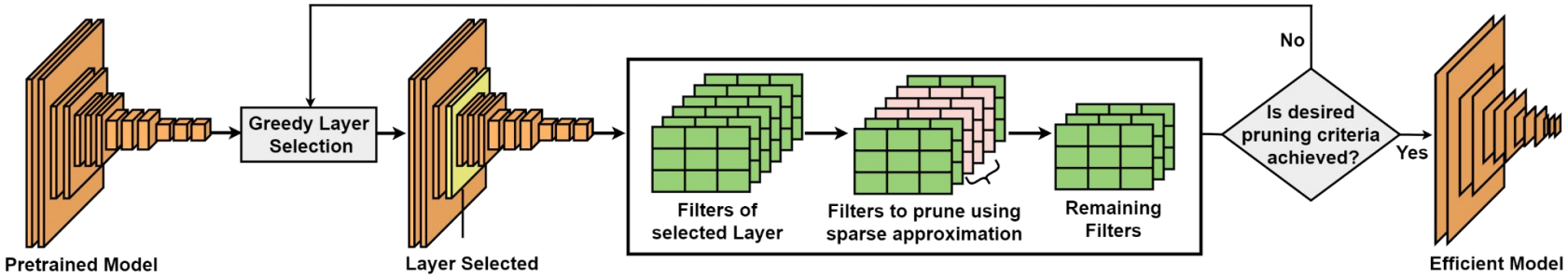
Non-Uniform Pruning

- Prune different fractions of filters from each layer
- All the layers in the network collectively make the final prediction

Contribution

- We developed faster **non-uniform pruning** methods.
- We used a hierarchical scheme with two-levels:
    - **filter pruning** - this step identifies the most appropriate filters to be pruned from each layer.
    - **layer selection** - this step selects the best layer to currently prune from.

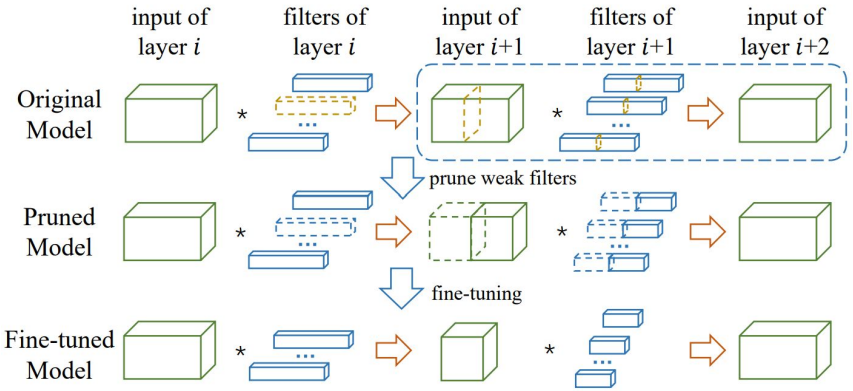
We apply these two steps iteratively to achieve a non-uniform pruning.



Related Work

Thinet

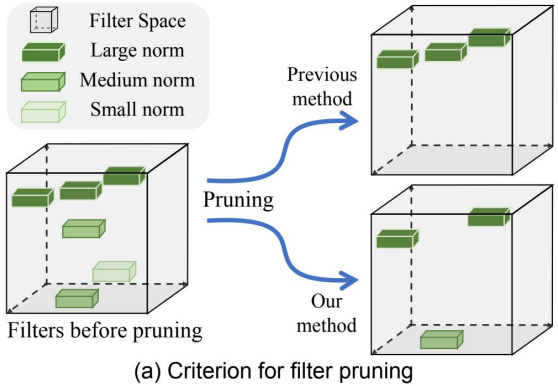
"ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression" ICCV 2017



- Removes each channel one-by-one, and record the output feature map difference at each step.
- Selects the channel with the lowest difference.
- Greedy way of selecting channel.

Related Work

FPGM **“Filter Pruning via Geometric Median for Deep Convolutional Neural Networks Acceleration” CVPR 2019**

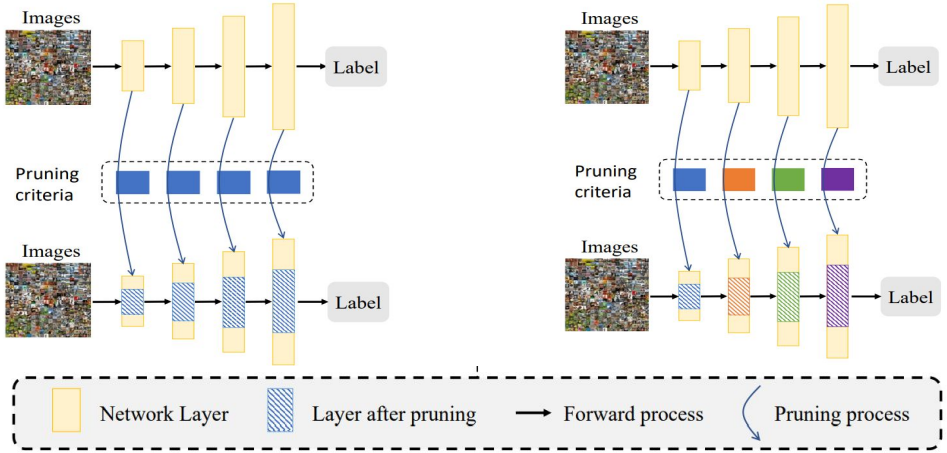


(a) Criterion for filter pruning

- Traditionally the filter weight, with small norm was regarded as less important filter.
- FPGM presents that filter with median norm is less important and can be removed

Related Work

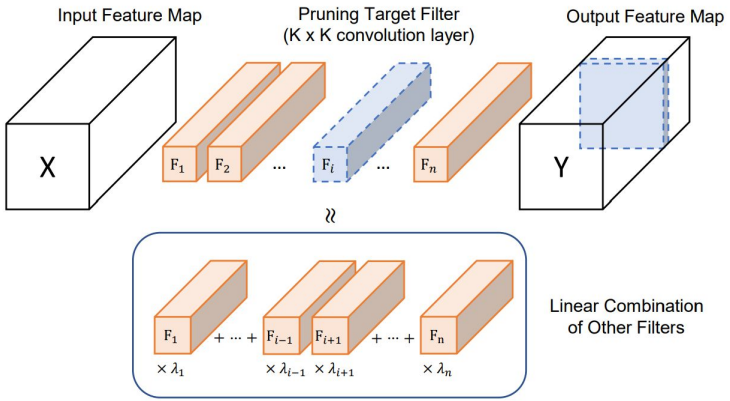
LFPC **“Learning Filter Pruning Criteria for Deep Convolutional Neural Networks Acceleration” CVPR 2020**



- Existing methods usually utilize pre-defined pruning criteria, such as ℓp-norm, to prune unimportant filters from each layer.
- LFPC adaptively select the appropriate pruning criteria for different layers.

Related Work

LRF **“Linearly Replaceable Filters for Deep Network Channel Pruning” AAAI 2021**



- LRF suggests that *we can replace the filter that can be approximated by the linear combination of other filters*

Related Work

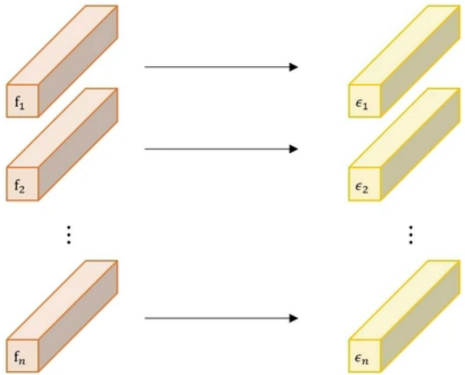
- In a layer, we can approximate each filter as a linear combination of the other filters

$$f_{:,j} = \sum_{l \neq j} \lambda_{j,l} f_{:,l} + \epsilon_j$$

Here, ϵ = approximation error and $\lambda_{j,l}$ = weight coefficient of the respective filters

- Each $\lambda_{j,l}$ can be found by solving following minimization problem

$$\min_{\lambda_{j,:}} ||f_{:,j} - \sum_{l \neq j} \lambda_{j,l} f_{:,l}||^2$$

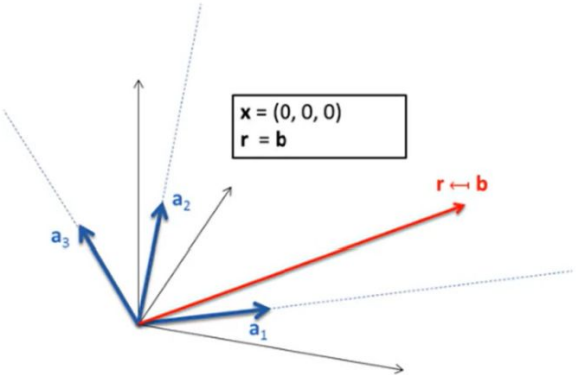


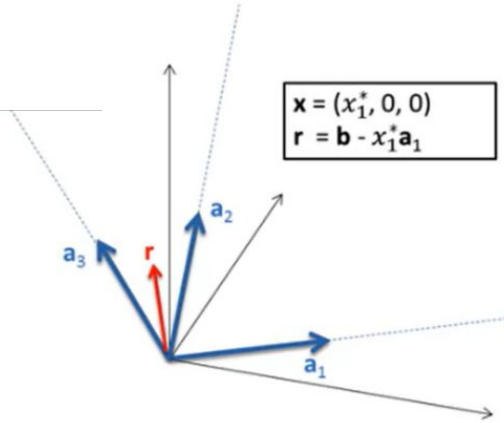
Remove the i^{th} filter with the smallest $||\epsilon_i||$

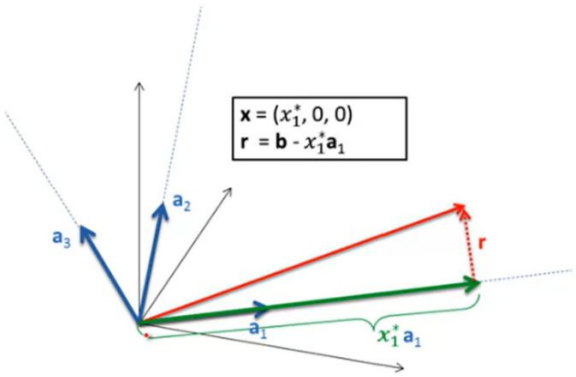
Orthogonal Matching Pursuit

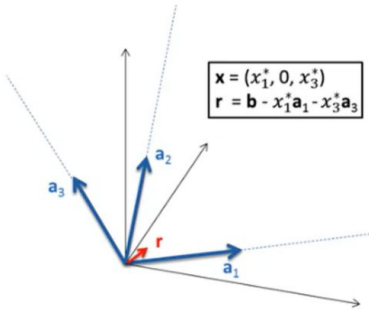
$$\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|_2 \quad \text{subject to } \|\mathbf{x}\|_0 \leq S$$

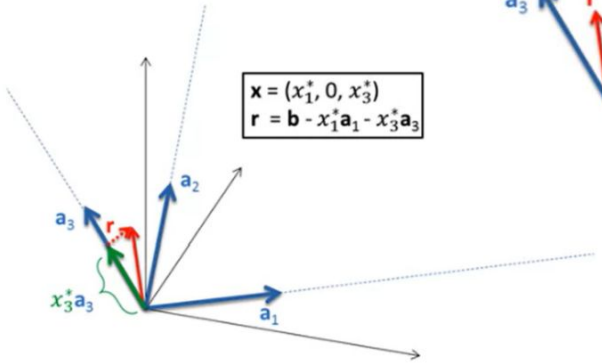
Input: A (with unit norm columns), $\mathbf{b}$, and S.
Initialize $\mathbf{r} = \mathbf{b}$ and $\Omega = \emptyset$.
While $\|\mathbf{x}\|_0 < S$
compute $x_j = \mathbf{a}_j^T \mathbf{r}$ for all $j \notin \Omega$
$i = \underset{j \notin \Omega}{argmax} |x_j|$
$\Omega \longleftarrow \Omega \cup \{i\}$
$\mathbf{x}_\Omega^* = \underset{\mathbf{x}}{argmin} \|A_\Omega \mathbf{x} - \mathbf{b}\|_2^2$
$\mathbf{r} \longleftarrow \mathbf{b} - A_\Omega \mathbf{x}_\Omega^*$











FP-OMP for Pruning Multiple Filters

We develop an Orthogonal Matching Pursuit (OMP) based algorithm for selecting retained filters of a layer into S. Hence filters that are to be pruned are {1,2,...n} \ S.

We can *approximate the pruned filters in terms of retained filters*.

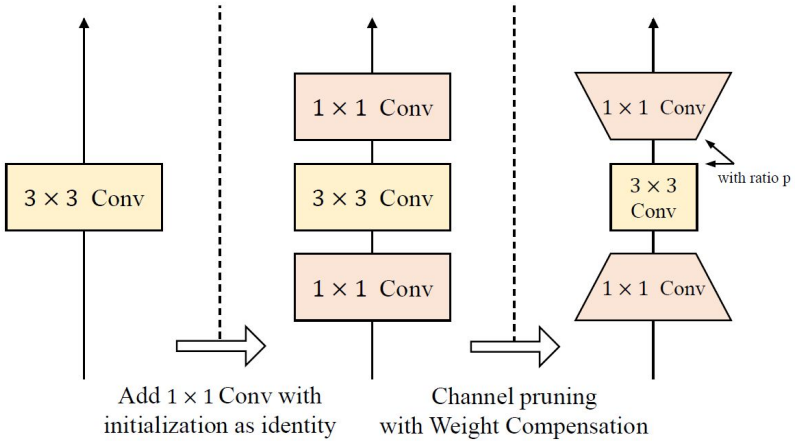
$$f_{:,j} = \sum_{l \in S} \lambda_{j,l} f_{:,l} + \epsilon_j, \forall j \notin S$$

We pose a sparse approximation problem for finding **S** and $\boldsymbol{\lambda}$

$$S^*, \lambda^* = \text{argmin}_{|S| \leq (1-\beta)n, \lambda} \sum_{j \in \{1,2,..,n\}} ||f_{:,j} - \sum_{l \in S} \lambda_{j,l} f_{:,l}||^2$$

where **S** is the set of the selected/retained filters in a layer, **n** is the total number of filter in that layer, and $\boldsymbol{\beta}$ is the pruning fraction

Weight Compensation



- Before we prune a layer, we add 1x1 convolutional layer at the top and bottom.
- When we prune a filter, we modify the weight value of 1x1 convolutional layer appropriately.
- Then the loss change can be further reduced.

Weight compensation for multiple filter pruning

LRF had proposed the weight compensation module, for a single filter pruning, for two purposes:

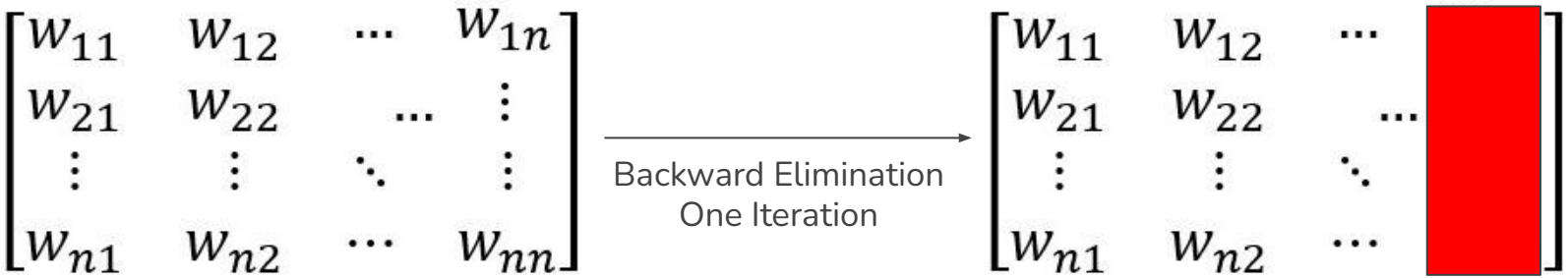
- The difference in output feature map of the pruned and unpruned layer will get adjusted by the updation of weights of the $1x1$ convolution.
- Usage of $1x1$ convolution enables the pruning of any network, regardless of its architecture.

We adopt this module and derive the compensated weights as per our framework for multiple channel pruning.

For the output channel pruning, $g'_{l,:} = g_{l,:} + \sum_{j \in S^c} \lambda_{j,l} * g_{j,:} \quad , \forall l \in S$

For the input channel pruning, $g'_{:,l} = g_{:,l} + \sum_{j \in S^c} \lambda_{j,l} * g_{:,j} \quad , \forall l \in S$

FP-Backward for Filter Pruning



For a given layer with **n** output channels, **m** input channels, and **K × K** filter size,
B: matrix of predicted filter weights.
A: matrix of retained filter weights.
B ≈ Aλ

$$\lambda^*_{:,j} = \text{argmin}_{\lambda_{:,j}} \sum_{j \in \{1,2,..,n\}} ||B_{:,j} - \sum_{l=1,...,n} A_{:,l}\lambda_{l,j}||^2 \quad = (A^T A)^{-1} A^T B_{:,j}, \qquad \forall j = 1, ..., n$$

E(A,B): the total least square error

$$E(A, B) = \sum_{j=1,...,n} \|B_{:,j} - A(A^T A)^{-1} A^T B_{:,j}\|^2 \quad = \sum_{j=1,...,n} (B^T_{:,j} B_{:,j} - B^T_{:,j} A(A^T A)^{-1} A^T B_{:,j})$$

Note that only the second term in E(A, B) is dependent on A

$$A = [A_{-k}\ a_k]\Pi^T$$

$\mathbf{A_{-k}}$: matrix A without the column a_k.
$\mathbf{\Pi_k}$: permutation matrix which permutes the columns of A so that k^{th} column is the last.

We compute $(A^TA)^{-1}$ and $(A^TA)^{-1}A^T$ for further purposes.

$$\begin{aligned}(A^TA)^{-1} &= \Pi \begin{bmatrix} G_k & g_k \\ g_k^T & \gamma_k \end{bmatrix} \Pi^T \\ (A^TA)^{-1}A^T &= \Pi \begin{bmatrix} D_k^T \\ d_k^T \end{bmatrix} = \Pi \begin{bmatrix} G_k A_{-k}^T + g_k a_k^T \\ g_k^T A_{-k}^T + \gamma_k a_k^T \end{bmatrix}\end{aligned}$$

$$\sum_j B_{:,j}^T A_{-k}(A_{-k}^T A_{-k})^{-1} A_{-k}^T B_{:,j} = \sum_j B_{:,j}^T A(A^TA)^{-1}A^T B_{:,j} - \sum_j \tfrac{1}{\gamma_k}|d_k^T B_{:,j}|^2$$

$$E(A_{-k}, B) = E(A, B) + \sum_{j=1,\dots,n} \tfrac{1}{\gamma_k}|d_k^T B_{:,j}|^2$$

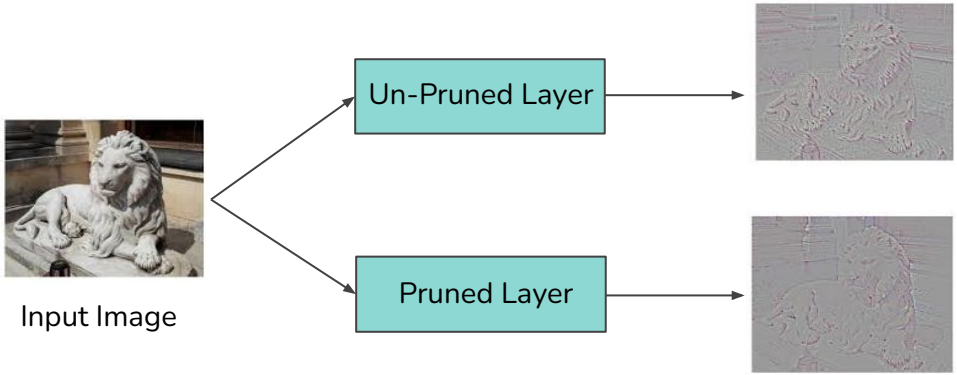
FP-Backward

Algorithm 3 Filter Pruning-Backward Elimination (`FP-Backward`)

Input: n: Number of filters, β: Pruning fraction,
$f_{:,j} \in \mathbb{R}^{K^2 m}$ $j = 1, ...n$: Filters
Initialize: $S = \{1, ..., n\}$ ▷ Set of currently retained filters
$B_{:,j} = [f_{:,j}]_{j=1,...,n}$ ▷ Matrix of predicted filter weights
$A = B$ ▷ Matrix of retained filter weights
while $|S| > (1 - \beta) * n$ **do**
  $G = [A^T A]^{-1} \in \mathbb{R}^{|S| \times |S|}$
  for $k = 1, ..., |S|$ **do**
    $g_k = G_{\{-k\},k}$
    $\gamma_k = G[k, k]$
    $d_k = A_{-k} g_k + a_k \gamma_k$
    $u_k = \frac{\sum_{j=1,...,n} |d_k^T B_{:,j}|^2}{\gamma_k}$
  end for
  $k^* = argmin_{k=1,...,|S|} u_k$
  $S \longleftarrow S \setminus \{S[k^*]\}$ ▷ remove original index corresponding to k^*
  $A = A_{:,\{-k^*\}}$ ▷ remove selected column
end while
Calculate λ using equation 3
Output: Set of selected filters-S, λ

HBGS for Layer Selection

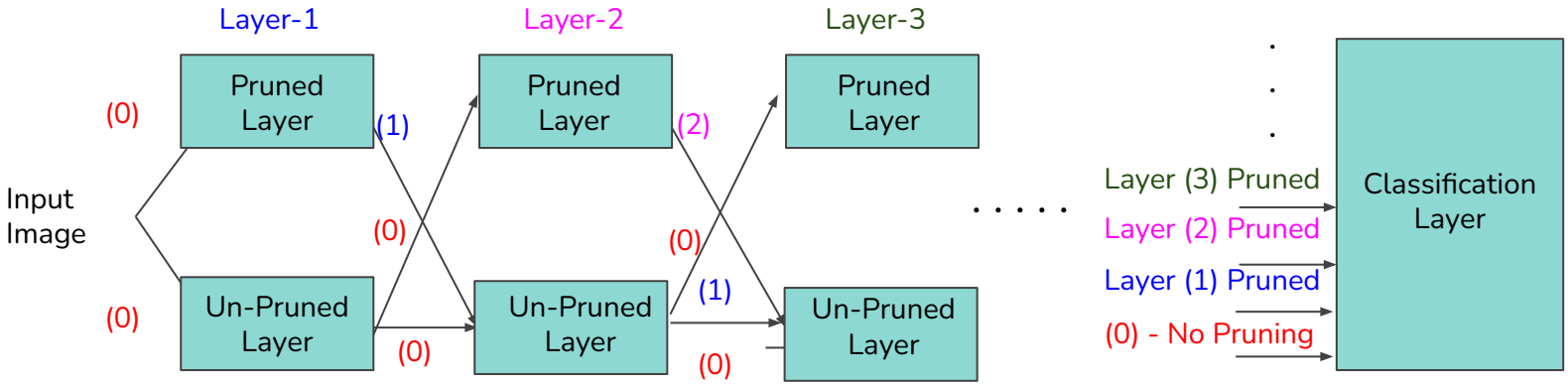
- We develop Hierarchical Backward Greedy Search (HBGS) for selecting the best layer to currently prune from.
- Key idea here is to calculate the relative reconstruction error between the pruned layer output and unpruned layer output
    - and then finally choose the layer with minimum error to currently prune from.



We want to minimise the difference b/w the output feature maps of pruned and unpruned layers

HBGTS for Layer Selection

- We develop Hierarchical Backward Greedy Tree Search (HBGTS) for selecting the best layer to currently prune from.
- Key idea here is to calculate the error in final layer output, if layer j ∈ {1, ..., C} is pruned
    - and then finally choose the layer with minimum error to currently prune from.



Experimental Results

Experimental Setting

Model Used

- ResNet-18, Resnet-32, Resnet-56, VGG-16
- Optimizer Used- SGD, batch size- 128, loss function- cross entropy loss, 1 epoch fine tune after pruning one layer and 300 after complete pruning
- Activation function- Relu

Performance Metric

Pruned Accuracy, Accuracy Drop, Parameters Drop, FLOPs Drop, Total Time taken for Pruning and Fine Tuning

Database Description

- CIFAR-10 - 10 Classes, 50k training images, 10k test images
- CIFAR-100 - 100 Classes, 50k training images, 10k test images
- Tiny Imagenet - 200 Classes, 0.1M images

Results and Analysis

Method	VGG16/CIFAR100 @ 98%				ResNet18/CIFAR10 @ 95%			
	Test Acc (%)	Acc ↓ (%)	Param ↓ (%)	FLOPs ↓ (%)	Test Acc (%)	Acc ↓ (%)	Param ↓ (%)	FLOPs ↓ (%)
Dense	67.1 ± 0.01	0 ± 0	-	-	94.5 ± 0.02	0 ± 0	-	-
Random	55.5 ± 0.16	11.6 ± 0.16	98.0	86.0	86.3 ± 0.06	8.2 ± 0.06	93.7	65.0
EarlyCroP-S (Rachwan et al., 2022)	62.8 ± 0.52	4.3 ± 0.52	97.9	88.0	91.0 ± 0.52	3.5 ± 0.52	95.1	65.8
DLRFC (He et al., 2022)	63.5 ± 0.09	3.56 ± 0.09	97.1	53.7	-	-	-	-
SAP (Diao et al., 2023)	-	-	-	-	91.4 ± 0.03	3.1 ± 0.03	94.9	64.9
PL (Chen et al., 2023)	63.5 ± 0.03	3.6 ± 0.03	97.3	87.9	-	-	-	-
LRF (Joo et al., 2021)	64.0 ± 0.31	3.1 ± 0.31	97.9	88.0	91.5 ± 0.37	3.0 ± 0.37	95.1	65.8
`FP-Backward`	66.2 ± 0.11	0.9 ± 0.11	97.9	88.0	92.8 ± 0.15	1.7 ± 0.15	95.1	65.8
`HBGS`	67.3 ± 0.17	-0.2 ± 0.17	98.3	89.6	93.9 ± 0.24	0.6 ± 0.24	95.3	66.2
`HBGS-B`	67.2 ± 0.15	-0.1 ± 0.15	98.1	89.4	93.7 ± 0.22	0.8 ± 0.22	95.2	66.0
`HBGTS`	**67.8** ± 0.23	**-0.7** ± 0.23	**98.5**	**89.8**	**94.7** ± 0.28	**-0.2** ± 0.28	**95.6**	**66.7**
`HBGTS-B`	$\underline{67.6} \pm 0.21$	$\underline{-0.5} \pm 0.21$	$\underline{98.4}$	$\underline{89.7}$	$\underline{94.6} \pm 0.24$	$\underline{-0.1} \pm 0.24$	$\underline{95.4}$	$\underline{66.5}$

Table: Performance comparison between different pruning methods on VGG16/CIFAR100 at 98% parameter reduction and ResNet18/CIFAR10 at 95% parameter reduction

- We can clearly see that our methods outperform other pruning algorithms.
- The drop in params and flops is equivalent or more compared to other methods

Results and Analysis (Cont.)

Method	Test Acc (%)	Acc ↓ (%)	Param ↓ (%)	FLOPs ↓ (%)	VRAM (GB)
Dense RN16	92.1	0	-	-	7.62
Dense RN8	91.8	0	-	-	3.91
FP-Backward	92.9	-0.8	98.5	89.9	1.59
HBGS-B	**93.0**	-0.9	98.7	92.1	1.55
HBGTS-B	**93.2**	-1.1	98.8	94.3	1.51

Table: Comparison of pruning methods for ResNext101 32x16d (RN16) and a similar sized dense ResNext101 32x8d (RN8) on CIFAR10 at 98% parameter reduction.

- Our backward method can be used for effectively pruning large models that exceed the capacity of commodity GPUs.
- ResNext101 32x16d has 193 M parameters and requires 7.62 GB of GPU memory for loading.
- We can efficiently deploy the pruned model on edge devices with GPU memory less than 2GB.

Results and Analysis (Cont.)

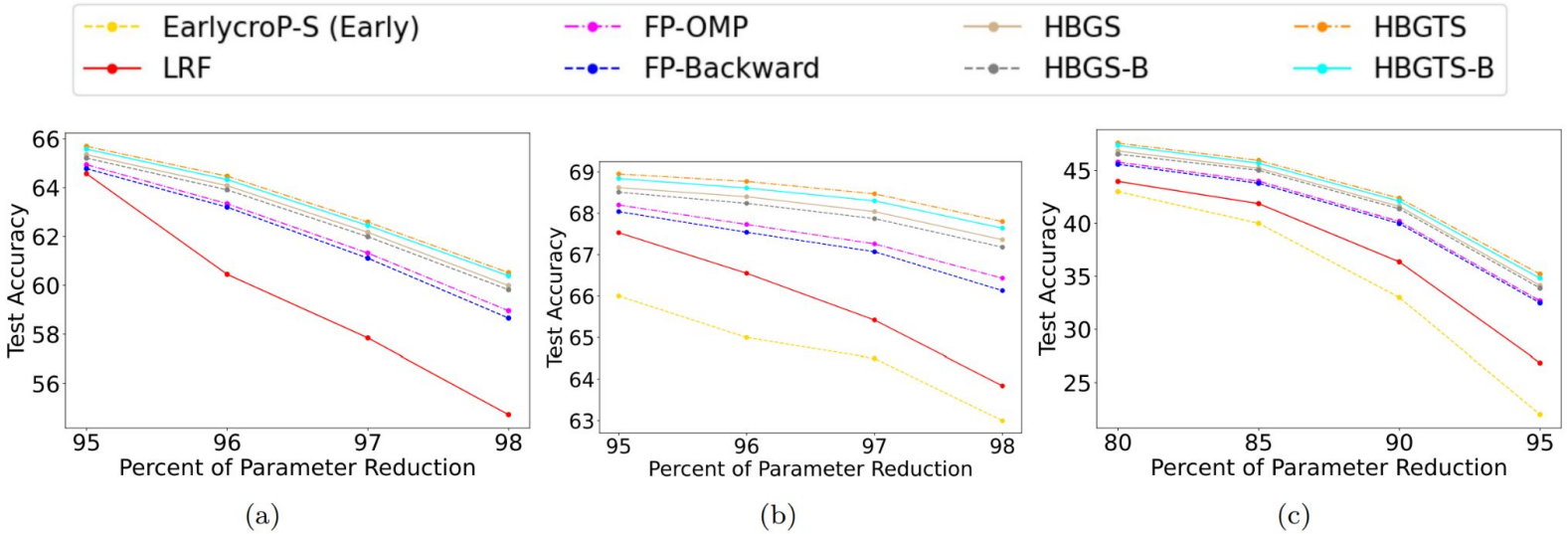


Figure: Test accuracy for (a) ResNet56/CIFAR100 (b) VGG16/CIFAR100 and (c) ResNet18/Tiny-Imagenet with increasing parameter reduction

- We can clearly see that our methods outperform other pruning algorithms.
- As the percentage of parameter reduction increases, the difference in test accuracy between our proposed methods and state-of-the-art methods also grows.

Results and Analysis (Cont.)

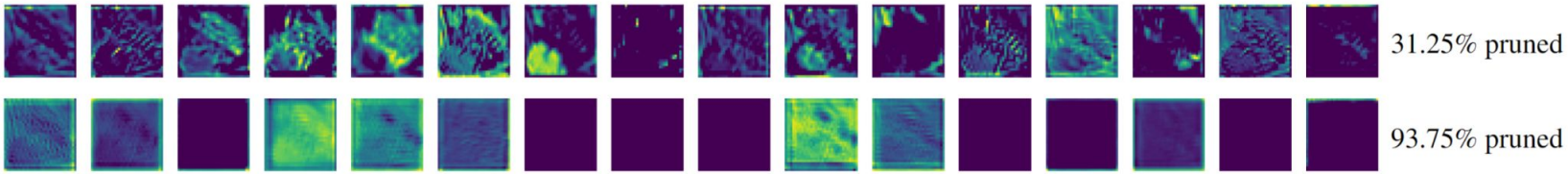


Figure: Visualisation of output feature map of ResNet32 2nd layer (top row) and 10th layer (bottom row) on CIFAR100

- Feature map of Layer 2 has a diverse set of filter outputs, indicates its usefulness in capturing different features of the inputs. Our HBGTS-B prunes only 31.25% of its filters.
- Feature map outputs of Layer 10 looks very similar, denoting its redundancy in filter outputs. 93.75% of its filters are removed by our HBGTS-B method.

Results and Analysis (Cont.)

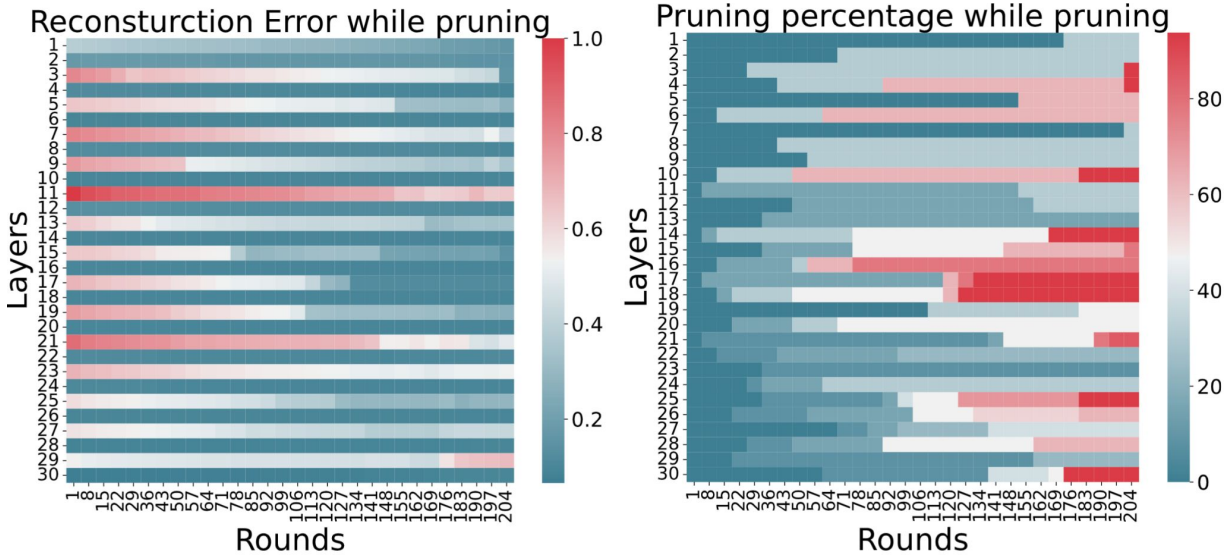


Figure: Heat map for relative reconstruction error and pruning percentage while pruning ResNet32 on CIFAR100 at 63% parameter reduction.

- Pruning percentage increases with each round, but not uniformly.
- Relative reconstruction error also decreases with pruning rounds but is not uniform across layers.
- Our method selects the layer with the least relative reconstruction error for pruning.

Results and Analysis (Cont.)

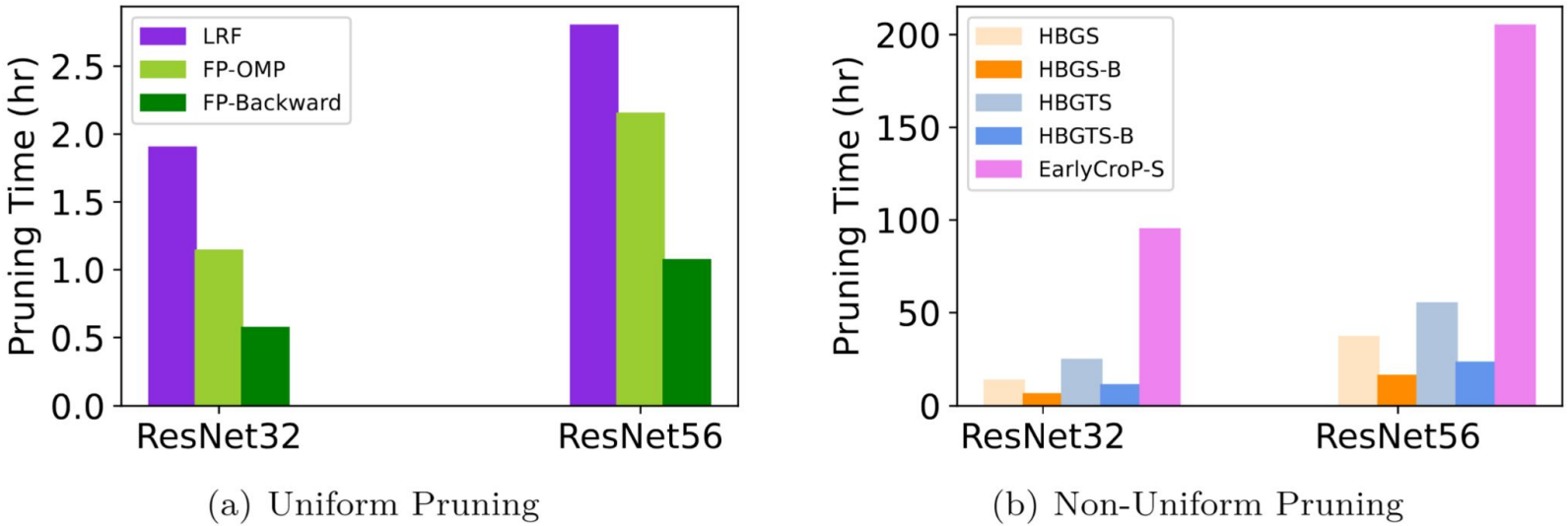


Figure : Time comparison on ResNet/CIFAR10 at 63% parameter reduction.

Conclusion

- We proposed a hierarchical scheme with two-levels for faster non-uniform pruning.
- FP-OMP and FP-Backward identifies the most appropriate filters to be pruned from each layer.
- HBGS and HBGTS algorithms selects the best layer to currently prune from.

References

1. Joo, Donggyu, et al. "*Linearly replaceable filters for deep network channel pruning*." Proceedings of the AAAI conference on artificial intelligence. Vol. 35. No. 9. 2021.
2. Tropp, Joel A., and Anna C. Gilbert. "*Signal recovery from random measurements via orthogonal matching pursuit*." IEEE Transactions on information theory 53.12 (2007): 4655-4666.
3. Purohit, Kiran, et al. "*Accurate and efficient channel pruning via orthogonal matching pursuit*." Proceedings of the Second International Conference on AI-ML Systems. 2022.
4. Reeves, Stanley J. "*An efficient implementation of the backward greedy algorithm for sparse signal reconstruction*." IEEE Signal Processing Letters 6.10 (1999): 266-268.

THANK YOU
FOR
YOUR ATTENTION!!!

https://github.com/kiranpurohit/

@kiranpurohit08