Network Pruning

Optimal Brain Damage (LeCun et al., 1990)
Rethinking the value of network pruning (Liu et al., 2019)

Carlos Quintero

Rice University

September 23, 2019

Outline

Basic pruning

Over-parameterization (deep learning) leads to:

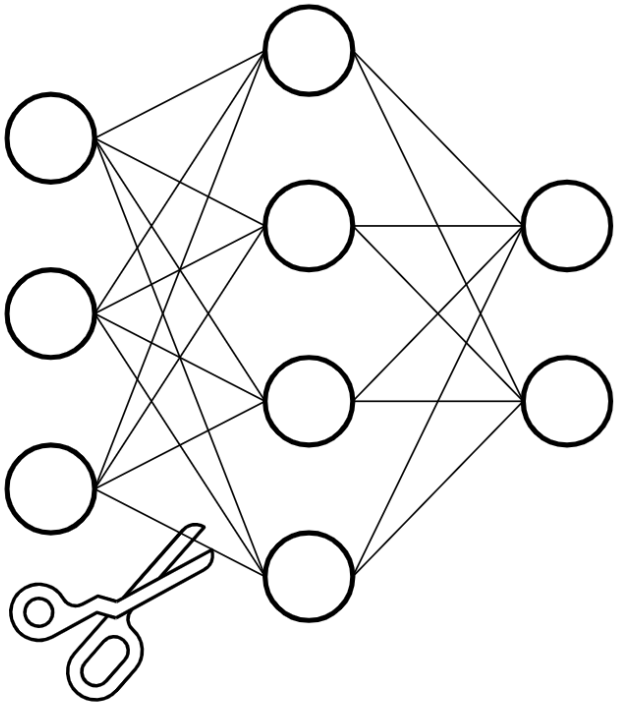
- High computational cost
- High memory footprint

For inference

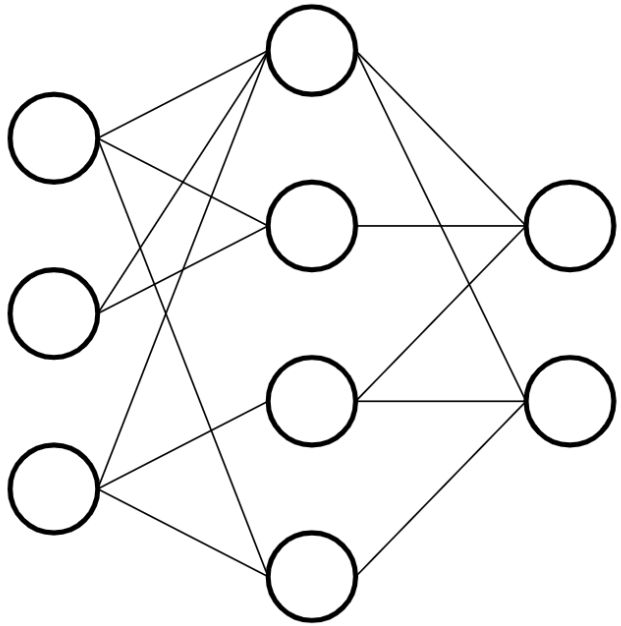
Pruning is the solution!!

To improve the efficiency of deep network for applications with limited computational budget

Basic pruning



Before pruning



After pruning

Basic pruning

Three-stage pipeline

1. Start with a large, over-parameterized network (Luo, et.al, Carreira-Perpinan & Idelbayev, 2018) to safely remove parameters
2. The pruned architecture and its weights are *essential* for obtaining the final efficient model
3. This has been reported that is better than training a network from scratch

Optimal Brain Damage: motivation

- OBD (Lecun, et al., 1990) aims at reducing the size of the network to avoid overfitting
- Weight deletion to control the network complexity
- After deletion, retrain
- A simple strategy is deleting weights whose deletion causes the least effect on the training error
- Define the saliency of a weight to be the change in the objective function when deleted

Optimal brain damage

- **Idea:** Construct a local model of the error and perturb it
- A perturbation δU of the parameter vector will cause:

$$\delta E = \sum_i g_i \delta u_i + \frac{1}{2} \sum_i h_{ii} \delta u_i^2 + \frac{1}{2} \sum_{i \neq j} h_{ij} \delta u_i \delta u_j + O(\|\delta U\|^3)$$

$$\delta E = \frac{1}{2} \sum_i h_{ii} \delta u_i^2$$

- Find parameters that will cause the least increase of E

Computing the second derivative

$$h_{kk} = \sum_{(i,j)\in V_k} \frac{\partial^2 E}{\partial w_{ij}^2}$$

$$\frac{\partial^2 E}{\partial w_{ij}^2} = \frac{\partial^2 E}{\partial a_i^2} x_j^2$$

$$\frac{\partial^2 E}{\partial a_i^2} = f'(a_i)^2 \sum_l w_{li}^2 \frac{\partial^2 E}{\partial a_l^2} + f''(a_i)\frac{\partial E}{\partial x_i}$$

$$\frac{\partial^2 E}{\partial a_i^2} = 2f'(a_i)^2 - 2(d_i - x_i)f''(a_i)$$

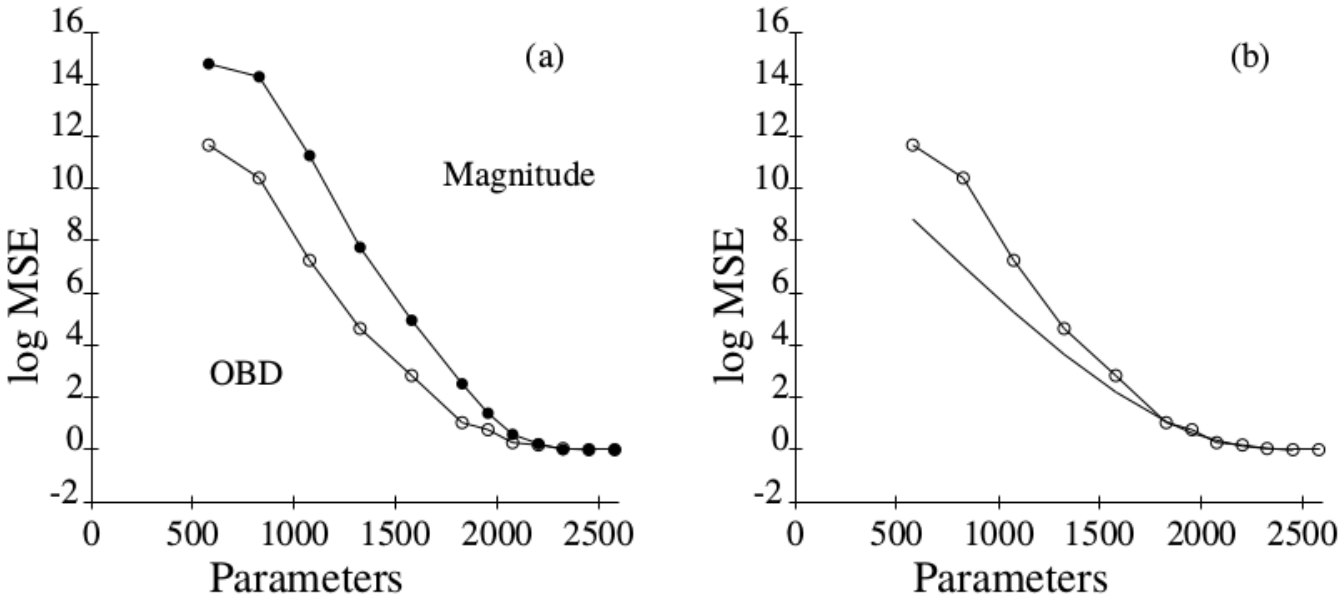
OBD

1. Choose a reasonable network architecture
2. Train the network until a reasonable solution is obtained
3. Compute the second derivatives h_{kk} for each parameter
4. Compute the saliencies for each parameter $s_k = h_{kk} u_k^2/2$
5. Sort the parameters by saliency and delete some low-saliency parameters
6. Iterate to step 2

Experiments

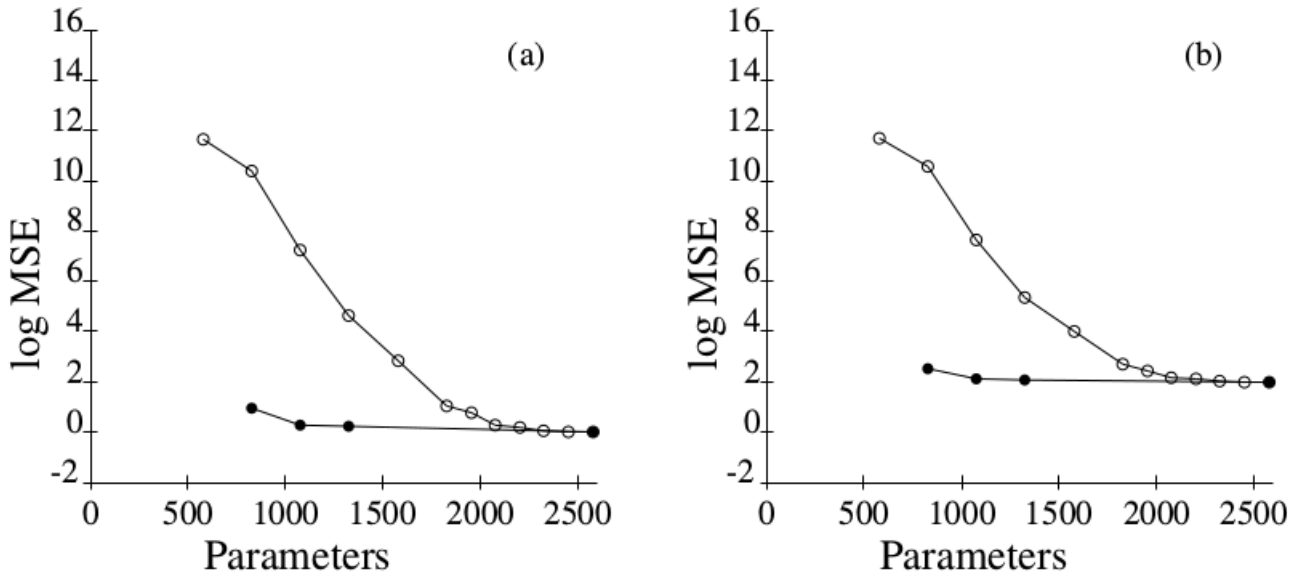
- Handwritten digit recognition
- Highly constrained and sparse network 10^5 connections controlled by 2578 free parameters
- 9300 training examples
- 3350 test examples

Experiments



- It works better than deleting parameters according to their magnitude
- The approximated value of the objective function is good to 800 deleted parameters: after that the off-diagonal terms are more important and also higher than quadratic terms

Experiments



Log-MSE on the training set (left) and on the test set (right) before (upper) and after (lower) retraining

OBD: conclusions

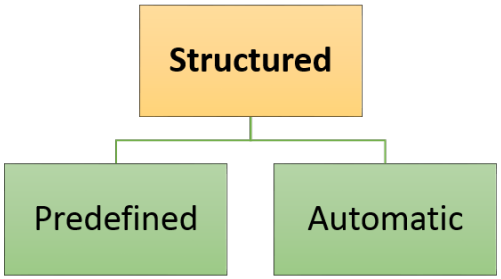
- Reduce the number of parameters in a practical network
- Network speed improved significantly
- Its recognition accuracy increased slightly
- A technique that improves the an already-good network is valuable

Basic pruning

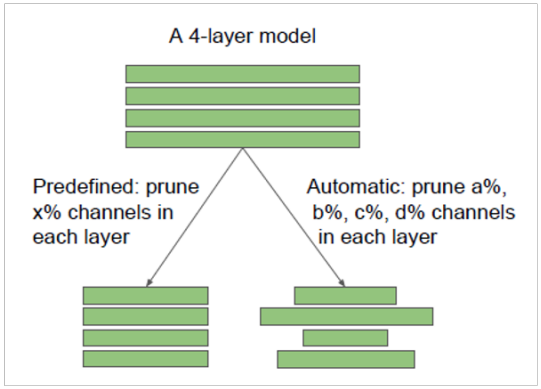
Three-stage pipeline

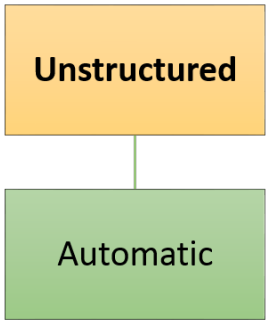
1. Start with a large, over-parameterized network (Luo, et.al, Carreira-Perpinan & Idelbayev, 2018) to safely remove parameters
2. The pruned architecture and its weights are *essential* for obtaining the final efficient model
3. This has been reported that is better than training a network from scratch

Types of Pruning



Prune at the levels of convolution chanels or larger (layers)





Prune at the levels of individual weights

Predefined pruning methods

A human determines the target architecture

- It's not important which specific channels are pruned. The pruned architecture remains the same
- The pruning algorithm prunes locally the least important channels in each layer
- The ratio in each layer is usually selected empirically

A common criterion is the ratio of channels to prune in each layer

Automatic pruning methods

A pruning algorithm determines the target architecture

- Usually by a pruning criterion that compares globally the structures across layers
- In unstructured pruning the weights are selected by the training process and pruning algorithm

Methods for pruning

Unstructured

- Optimal brain damage (LeCun et.al, 1990)
- Optimal brain surgeon (Hassibi & Stork, 1993)
- Weights with small magnitude (**Han et.al, 2015**)
- Remove redundant neurons (Srinivas & Babu, 2015)
- Variational dropout redundant weights (P. Kingma et.al, 2015)
- Learn sparse nets L0 –norm regularization (Louizos et.al, 2018)

Structured

- Filter weight norm (**Li et.al, 2017**)
- Group sparsity (Wen et.al, 2016, Alvarez & Salzman, 2016, Levedev & Lempitsky, 2016, Zhou et.al, 2016)
- Sparsity constraints on channel-wise scaling factors (**Liu et.al, 2017**, Ye et.al, 2018, **Huang & Wang, 2018**)
- Minimize next layer's feature reconstruction (**He et.al, 2017**, **Luo et.al, 2017**)
- Approximation of channel influence over los (Molchanov, 2016)
- Intrinsic correlation within layer (Suau et.al, 2018)
- Layer-wise compensate filter (Chin et.al, 2018)
- Prune based on current filter (Lin et.al, 2017, Wang et.al, 2017)
- Allow pruned filters to recover (**He et.al, 2018**)
- Random channel pruning (Mittal et.al, 2018)

Predefined tuning methods, automatic structured pruning, Unstructured pruning method

Conclusion 1

- For structured pruning methods (predefined): directly training the small target model from random initialization can achieve the same, if not better, performance, as the model from the three-stage pipeline (don't need large model)
- For structured pruning methods (automatic): training the pruned model from scratch can achieve comparable or even better performance than fine-tuning

For these methods what matters is the obtained architecture instead of the preserved weights

Conclusion 2

- For unstructured pruning methods: Training from scratch can mostly achieve comparable results with pruning and fine-tuning on smaller-scale datasets, but fails on the large-scale ImageNet benchmark

If a pretrained large model is already available, pruning and fine-tuning can save training time required to obtain the efficient model

Datasets

- CIFAR-10: 60000 colour images with 6000 images per class
- CIFAR-100: 60000 colour images with 600 images per class, 20 superclasses
- (Krizhevsky, 2009): 79M images
- ImageNet: Image dataset described by word phrases

Network architectures

- VGG: Very deep convolutional neural network
- ResNet: Deep residual learning
- DenseNet: Dense convolutional network

Methods for pruning

Unstructured

- Optimal brain damage (LeCun et.al, 1990)
- Optimal brain surgeon (Hassibi & Stork, 1993)
- Weights with small magnitude (**Han et.al, 2015**)
- Remove redundant neurons (Srinivas & Babu, 2015)
- Variational dropout redundant weights (P. Kingma et.al, 2015)
- Learn sparse nets L0 –norm regularization (Louizos et.al, 2018)

Structured

- Filter weight norm (**Li et.al, 2017**)
- Group sparsity (Wen et.al, 2016, Alvarez & Salzman, 2016, Levedev & Lempitsky, 2016, Zhou et.al, 2016)
- Sparsity constraints on channel-wise scaling factors (**Liu et.al, 2017**, Ye et.al, 2018, **Huang & Wang, 2018**)
- Minimize next layer's feature reconstruction (**He et.al, 2017**, **Luo et.al, 2017**)
- Approximation of channel influence over los (Molchanov, 2016)
- Intrinsic correlation within layer (Suau et.al, 2018)
- Layer-wise compensate filter (Chin et.al, 2018)
- Prune based on current filter (Lin et.al, 2017, Wang et.al, 2017)
- Allow pruned filters to recover (**He et.al, 2018**)
- Random channel pruning (Mittal et.al, 2018)

Predefined tuning methods, automatic structured pruning, Unstructured pruning method

Training budget

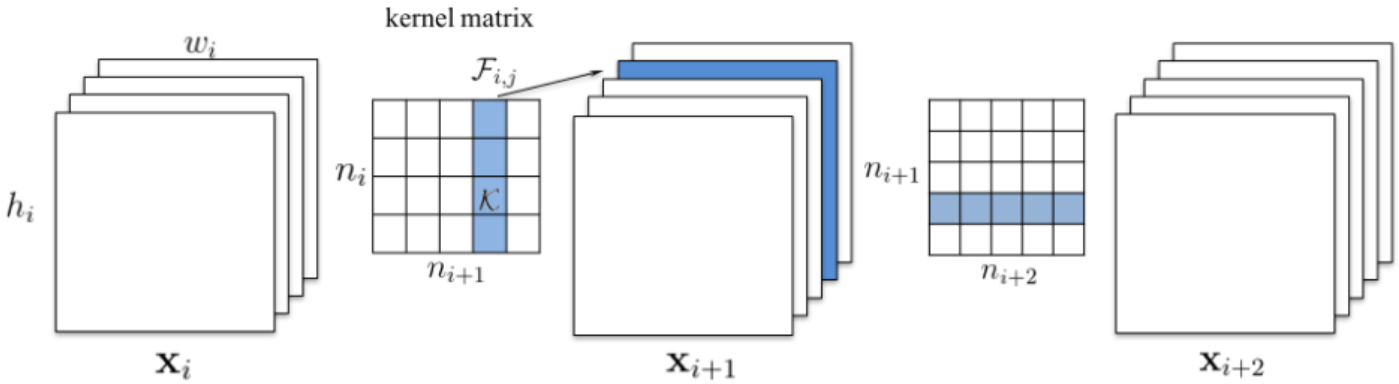
- **Scratch E**: Small pruned model is trained for the same number of epochs.
- **Scratch B**: Small pruned model is trained for the same amount of computation (FLOPS). When extending the number of epochs, they also extend the learning rate decay schedule

Smaller methods should be trained for fewer epochs? Increasing the epochs is not harmful

Implementation

1. Use the original implementation if available
2. Re-implement the three-stage pruning procedure and achieve similar results
3. When models are available but not the training setup, re-train both large and small target models from scratch (accuracy of re-trained large model is better than original papers). They report relative acccuracy drop from large model
4. Standard training hyper-parameters and data augmentation
5. Optimization is SGD with Nesterov momentum with stepwise decay learning rate schedule
6. Random weight initialization from (He et.al, 2015)
7. For fine-tuning from inherited weights, they use the release on the original papers or use the lowest learning rate when training the large model

Predefined structured:



- When a filter is pruned, its feature map in the next layer disappears along with its corresponding kernels

Experiments - Predefined structured:

L_1-norm based Filter Pruning (Li et al., 2017): A percentage of filters with smaller L_1-norm are pruned. A is 10% of filters skipping sensitive layers. B uses different pruning rates per layer (stage).

Dataset	Model	Unpruned	Pruned Model	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	VGG-16	93.63 (±0.16)	VGG-16-A	93.41 (±0.12)	93.62 (±0.11)	**93.78** (±0.15)
	ResNet-56	93.14 (±0.12)	ResNet-56-A	92.97 (±0.17)	92.96 (±0.26)	**93.09** (±0.14)
			ResNet-56-B	92.67 (±0.14)	92.54 (±0.19)	**93.05** (±0.18)
	ResNet-110	93.14 (±0.24)	ResNet-110-A	93.14 (±0.16)	**93.25** (±0.29)	93.22 (±0.22)
			ResNet-110-B	92.69 (±0.09)	92.89 (±0.43)	**93.60** (±0.25)
ImageNet	ResNet-34	73.31	ResNet-34-A	72.56	72.77	**73.03**
			ResNet-34-B	72.29	72.55	**72.91**

- Scratch-trained models achieve (at least) the same level of accuracy as fine-tuned

Experiments - Predefined structured:

ThiNet (Luo et al., 2017): Prune the channel with smallest effect in the next layer's activation values. VGG-Conv only the first 10 convolutional layers are pruned with compression rate 0.5. In VGG-GAP the FC layers are removed and replaced with global average pooling and fine-tuned in 12 epochs. VGG-Tiny is pruning with 0.25 compression rate. ResNet50 with different compression rates $(30\%, 50\%, 70\%)$

Dataset	Unpruned	Strategy	Pruned Model		
ImageNet	VGG-16		VGG-Conv	VGG-GAP	VGG-Tiny
	71.03	Fine-tuned	−1.23	−3.67	−11.61
	71.51	Scratch-E	−2.75	−4.66	−14.36
		Scratch-B	**+0.21**	**−2.85**	**−11.58**
	ResNet-50		ResNet50-30%	ResNet50-50%	ResNet50-70%
	75.15	Fine-tuned	−6.72	−4.13	−3.10
	76.13	Scratch-E	−5.21	−2.82	−1.71
		Scratch-B	**−4.56**	**−2.23**	**−1.01**

- Scratch models achieve almost always better performance

Experiments - Predefined structured

Regression based Feature Reconstruction (He et al., 2017b): Prune channels by minimizing feature map reconstruction error of the next layer (using LASSO)

Dataset	Unpruned	Strategy	Pruned Model
ImageNet	VGG-16		VGG-16-5x
	71.03	Fine-tuned	−2.67
	71.51	Scratch-E	−3.46
		Scratch-B	**−0.51**
	ResNet-50		ResNet-50-2x
	75.51	Fine-tuned	−3.25
	76.13	Scratch-E	−1.55
		Scratch-B	**−1.07**

Experiments - Automatic structured

Network Slimming (Liu et al., 2017): prunes channels with lower scaling factors imposing L_1-sparsity channel-wise from Batch Normalization layers

Dataset	Model	Unpruned	Prune Ratio	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	VGG-19	93.53 (±0.16)	70%	93.60 (±0.16)	93.30 (±0.11)	**93.81** (±0.14)
	PreResNet-164	95.04 (±0.16)	40%	94.77 (±0.12)	94.70 (±0.11)	**94.90** (±0.04)
			60%	94.23 (±0.21)	94.58 (±0.18)	**94.71** (±0.21)
	DenseNet-40	94.10 (±0.12)	40%	94.00 (±0.20)	93.68 (±0.18)	**94.06** (±0.12)
			60%	**93.87** (±0.13)	93.58 (±0.21)	93.85 (±0.25)
CIFAR-100	VGG-19	72.63 (±0.21)	50%	72.32 (±0.28)	71.94 (±0.17)	**73.08** (±0.22)
	PreResNet-164	76.80 (±0.19)	40%	76.22 (±0.20)	76.36 (±0.32)	**76.68** (±0.35)
			60%	74.17 (±0.33)	75.05 (± 0.08)	**75.73** (±0.29)
	DenseNet-40	73.82 (±0.34)	40%	**73.35** (±0.17)	73.24 (±0.29)	73.19 (±0.26)
			60%	72.46 (±0.22)	72.62 (±0.36)	**72.91** (±0.34)
ImageNet	VGG-11	70.84	50%	68.62	70.00	**71.18**

- Channel scaling factors are compared across layers (prune the lowest percentage of scaling factors), the architecture is automatically discovered

Experiments - Automatic structured

Sparse Structure Selection (Huang & Wang, 2018): also uses sparsified scaling factors. Other than channels, pruning can be on **residual blocks** or groups (ResNet-41, ResNet-32, ResNet-26). No fine-tuning

Dataset	Model	Unpruned	Pruned Model	Pruned	Scratch-E	Scratch-B
ImageNet	ResNet-50	76.12	ResNet-41	75.44	75.61	**76.17**
			ResNet-32	74.18	73.77	**74.67**
			ResNet-26	71.82	72.55	**73.41**

Experiments - Unstructured

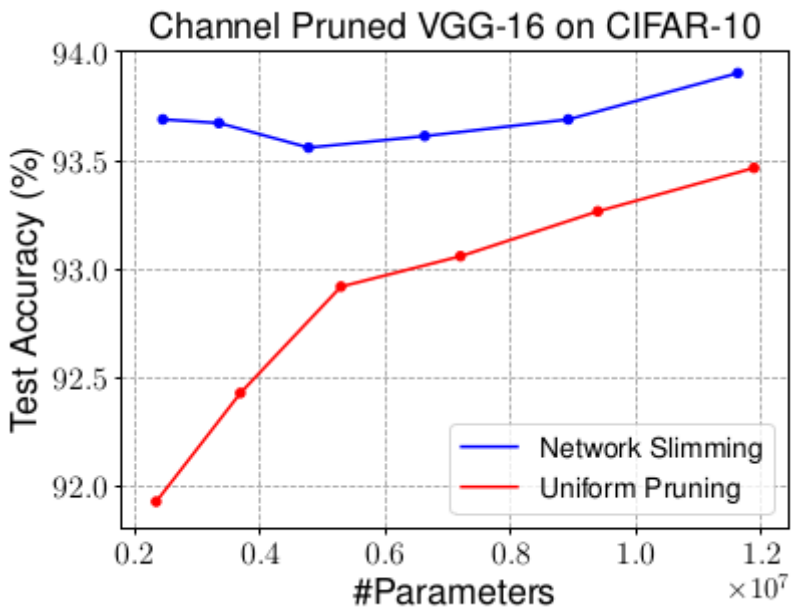
Unstructured magnitude-base weight pruning (Huang & Wang, 2018): automatic architecture discovery, since positions can not be determined before training

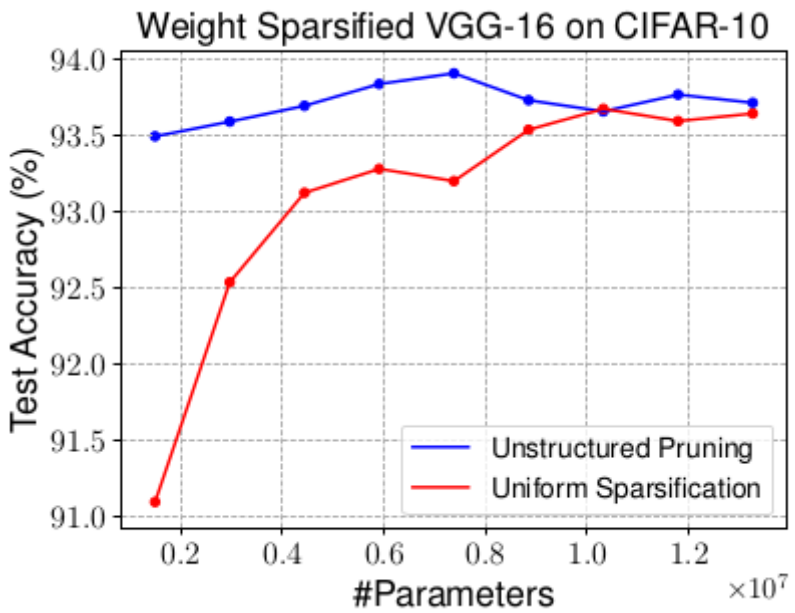
Only prune convolutional layers

Dataset	Model	Unpruned	Prune Ratio	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	VGG-19	93.50 (±0.11)	30%	93.51 (±0.05)	**93.71** (±0.09)	93.31 (±0.26)
			80%	93.52 (±0.10)	**93.71** (±0.08)	93.64 (±0.09)
			95%	93.34 (±0.13)	93.21 (±0.17)	**93.63** (±0.18)
	PreResNet-110	95.04 (±0.15)	30%	95.06 (±0.05)	94.84 (±0.07)	**95.11** (±0.09)
			80%	**94.55** (±0.11)	93.76 (±0.10)	94.52 (±0.13)
			95%	**92.35** (±0.20)	91.23 (±0.11)	91.55 (±0.34)
	DenseNet-BC-100	95.24 (±0.17)	30%	95.21 (±0.17)	95.22 (±0.18)	**95.23** (±0.14)
			80%	95.04 (±0.15)	94.42 (±0.12)	**95.12** (±0.04)
			95%	**94.19** (±0.15)	92.91 (±0.22)	93.44 (±0.19)
CIFAR-100	VGG-19	71.70 (±0.31)	30%	71.96 (±0.36)	72.81 (±0.31)	**73.30** (±0.25)
			50%	71.85 (±0.30)	73.12 (±0.36)	**73.77** (±0.23)
			95%	70.22 (±0.38)	70.88 (±0.35)	**72.08** (±0.15)
	PreResNet-110	76.96 (±0.34)	30%	76.88 (±0.31)	76.36 (±0.26)	**76.96** (±0.31)
			50%	**76.60** (±0.36)	75.45 (±0.23)	76.42 (±0.39)
			95%	68.55 (±0.51)	68.13 (±0.64)	**68.99** (±0.32)
	DenseNet-BC-100	77.59 (±0.19)	30%	77.23 (±0.05)	77.58 (±0.25)	**77.97** (±0.31)
			50%	77.41 (±0.14)	77.65 (±0.09)	**77.80** (±0.23)
			95%	**73.67** (±0.03)	71.47 (±0.46)	72.57 (±0.37)
ImageNet	VGG-16	73.37	30%	73.68	72.75	**74.02**
			60%	**73.63**	71.50	73.42
	ResNet-50	76.15	30%	**76.06**	74.77	75.70
			60%	**76.09**	73.69	74.91

Parameter Efficiency of Pruned Architectures

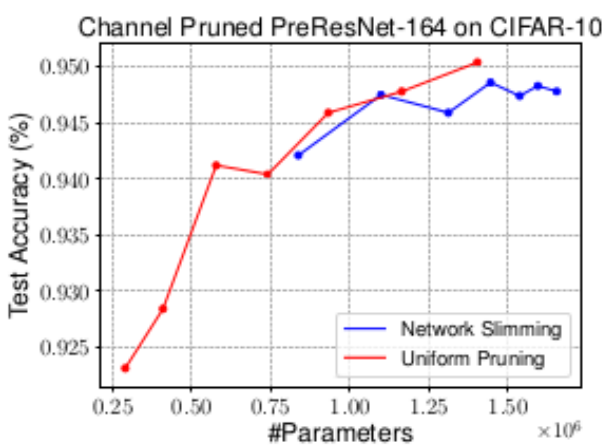
(Left) Compare automatic (structured) method (slimming, Liu et al., 2017) with one that uniformly prunes the same percentage of channels in each layer.
(Right) Compare automatic (unstructured) method (Han et al., 2015)

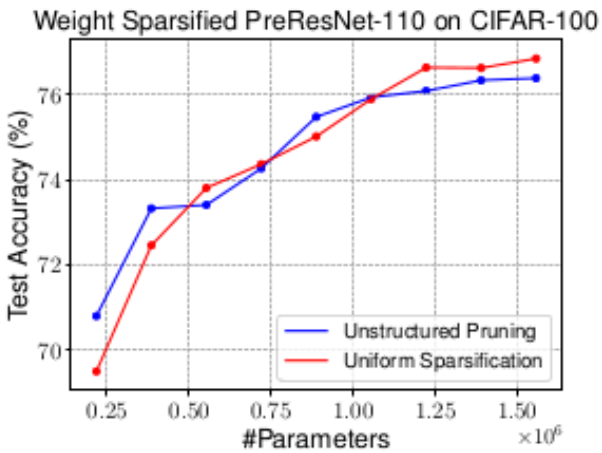


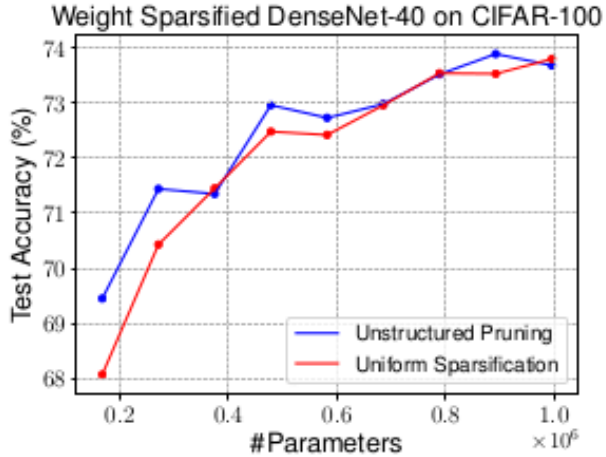


More Analysis

In some cases, architectures are not better than uniformly pruned ones.
Mostly in modern architectures (ResNets and DenseNets)
The sparsity pattern of those pruned architectures is close to uniform across stages. Not for VGG





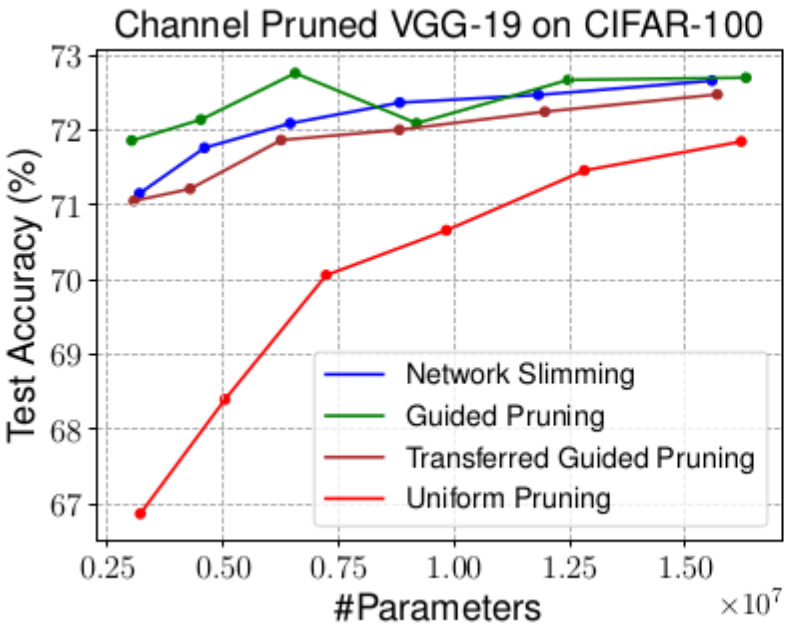


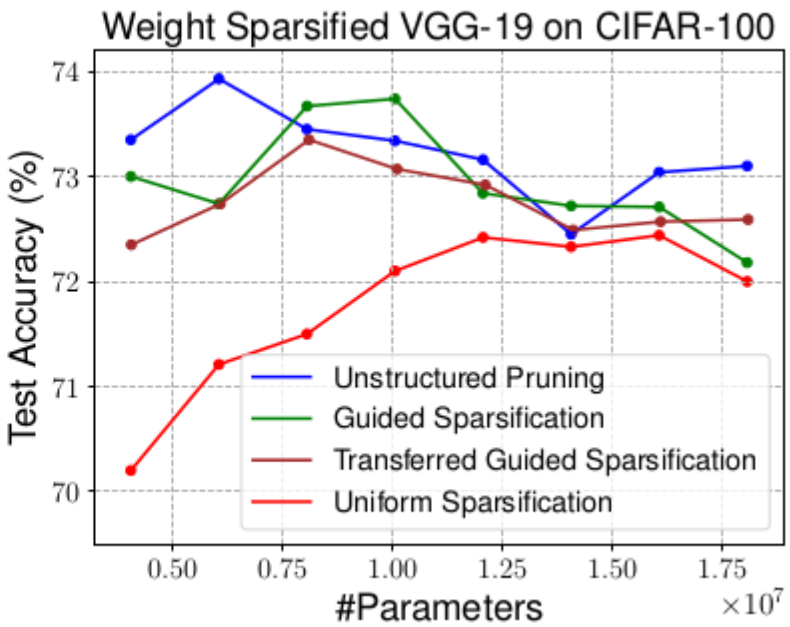
Redundancy is imbalanced across layer stages

More Analysis

Network Slimming: use the average number of channels in each layer stage from pruned architectures to construct new set of architectures **Guided Pruning**

Magnitud-based: Analyze sparsity patterns of pruned architectures and apply them to construct a new set of sparse models **Guided Sparsification**





Conclusions

- Predefined target models: training from scratch is good because:
  - The model is smaller (no need to train the large model)
  - There is no need to implement the pruning method (it may require fine-tuning layer by layer)
  - Avoid tuning additional parameters
- Automatic target models: They are capable of finding efficient architectures in some cases, but if they can be efficiently trained from scratch they must be compared to uniformly pruned baselines
- Pruning and fine-tuning:
  - Is good when pre-trained large models are already available and no training budget is available (method does not require modifications to the large model training).
  - Training a large model and prune by different ratios may help when one does not know the desirable size or need to obtain different size models

More Analysis

	10%	20%	30%	40%	50%	60%	70%
Stage 1	0.879	0.729	0.557	0.484	0.421	0.349	0.271
Stage 2	0.959	0.863	0.754	0.651	0.537	0.428	0.320
Stage 3	0.889	0.798	0.716	0.610	0.507	0.403	0.301

Table 18: Sparsity patterns of PreResNet-164 pruned on CIFAR-10 by Network Slimming shown in Figure 5 (left) under different prune ratio. The top row denotes the total prune ratio. The values denote the ratio of channels to be kept. We can observe that for a certain prune ratio, the sparsity patterns are close to uniform (across stages).

More Analysis

	10%			50%			90%		
Stage 1	0.905	0.905	0.909	0.530	0.561	0.538	0.129	0.171	0.133
	0.900	0.912	0.899	0.559	0.588	0.551	0.166	0.217	0.176
	0.903	0.913	0.902	0.532	0.563	0.547	0.142	0.172	0.163
Stage 2	0.906	0.911	0.906	0.485	0.523	0.503	0.073	0.102	0.085
	0.912	0.911	0.915	0.508	0.529	0.525	0.099	0.114	0.111
	0.911	0.916	0.912	0.502	0.529	0.519	0.080	0.113	0.096
Stage 3	0.901	0.904	0.900	0.454	0.475	0.454	0.043	0.059	0.048
	0.885	0.891	0.889	0.409	0.420	0.415	0.032	0.033	0.035
	0.898	0.903	0.902	0.450	0.468	0.458	0.042	0.055	0.046

Table 19: Average sparsity patterns of 3×3 kernels of PreResNet-110 pruned on CIFAR-100 by unstructured pruning shown in Figure 5 (middle) under different prune ratio. The top row denotes the total prune ratio. The values denote the ratio of weights to be kept. We can observe that for a certain prune ratio, the sparsity patterns are close to uniform (across stages).

More Analysis

	10%			50%			90%		
Stage 1	0.861	0.856	0.858	0.507	0.495	0.510	0.145	0.129	0.142
	0.843	0.844	0.851	0.484	0.486	0.479	0.123	0.115	0.126
	0.850	0.854	0.857	0.509	0.490	0.511	0.136	0.131	0.147
Stage 2	0.907	0.905	0.906	0.498	0.487	0.499	0.099	0.088	0.100
	0.892	0.888	0.892	0.442	0.427	0.444	0.064	0.043	0.065
	0.907	0.906	0.905	0.497	0.485	0.493	0.095	0.082	0.098
Stage 3	0.897	0.901	0.899	0.470	0.475	0.472	0.060	0.060	0.064
	0.888	0.890	0.889	0.433	0.437	0.435	0.040	0.040	0.042
	0.898	0.900	0.899	0.473	0.477	0.473	0.060	0.061	0.063

Table 20: Average sparsity patterns of 3×3 kernels of DenseNet-40 pruned on CIFAR-100 by unstructured pruning shown in Figure 5 (right) under different prune ratio. The top row denotes the total prune ratio. The values denote the ratio of weights to be kept. We can observe that for a certain prune ratio, the sparsity patterns are close to uniform (across stages).

More Analysis

	10%	20%	30%	40%	50%	60%
Stage 1	0.969	0.914	0.883	0.875	0.844	0.836
Stage 2	1.000	1.000	1.000	1.000	1.000	1.000
Stage 3	0.991	0.975	0.966	0.957	0.947	0.947
Stage 4	0.861	0.718	0.575	0.446	0.312	0.258
Stage 5	0.871	0.751	0.626	0.486	0.352	0.132

Table 21: Sparsity patterns of VGG-16 pruned on CIFAR-10 by Network Slimming shown in Figure 3 (left) under different prune ratio. The top row denotes the total prune ratio. The values denote the ratio of channels to be kept. For each prune ratio, the latter stages tend to have more redundancy than earlier stages.

Experiments - Predefined structured:

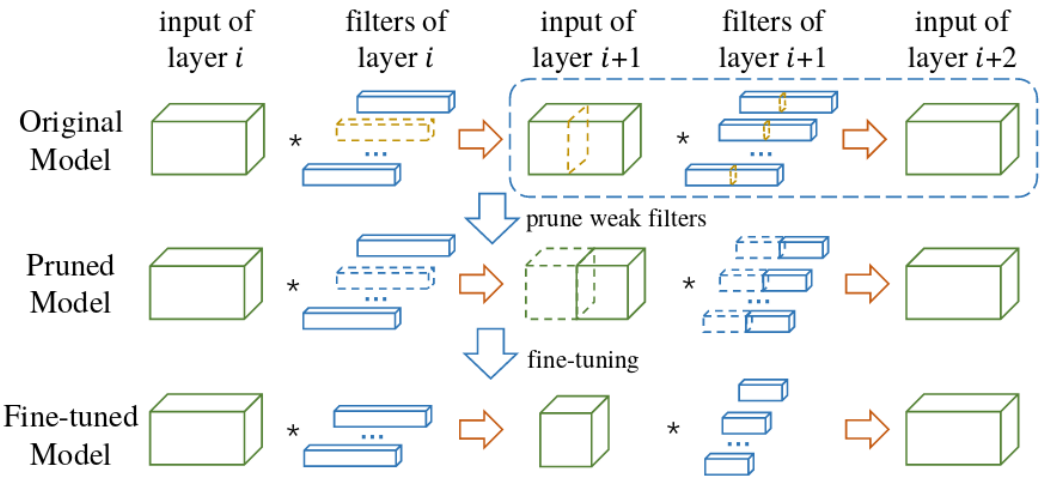
L_1-norm based Filter Pruning (Li et al., 2017): A percentage of filters with smaller L_1-norm are pruned. A is 10% of filters skipping sensitive layers. B uses different pruning rates per layer.

- For each filter calculate the sum of its absolute kernel weights
- Sort the filters
- Prune m filters with the smallest sum and their corresponding feature maps. Also remove the kernels corresponding to the removed feature maps
- A new kernel is created for both layers and the remaining weights are copied

Experiments - Predefined structured:

ThiNet (Luo et al., 2017)

- If we can use a subset of channels in layer $(i+1)$'s input to approximate the output in layer $i+1$, the other channels can be safely removed from the input of layer $(i+1)$



Experiments - Predefined structured:

Regression based Feature Reconstruction (He et al., 2017b):

- Figure out the most representative channels and prune redundant ones using LASSO regression
- Reconstruct the outputs with remaining channels with linear least squares

Experiments - Automatic structured

Network Slimming (Liu et al., 2017)

- Associate each channel with a scaling factor which is multiplied at the output of the channel
- Jointly train the network weights and the scaling factors with a sparsity regularization
- Prune channels with small scaling factors

$$L = \sum_{(x,y)} l(f(x, W), y) + \lambda \sum_{\gamma \in \Gamma} g(\gamma)$$

- L_1-norm is used as sparsity measure
- Insert BN layer after the convolutional layers is perhaps the most effective way we can learn meaningful scaling factors for channel pruning
- A pruning threshold for scaling factors is required. They select it by a percentile among all scaling factors e.g., 40% of channels are prunned.

Experiments - Automatic structured

Sparse Structure Selection (Liu et al., 2017)

- Very similar to Network Slimming (Liu et al., 2017), but a more general case
- Their method allows to add scaling factors to neurons, groups and blocks