

Bayesian Compression for Deep Learning

Xin Yu, 2022-2-18

Paper lists

Channel-based pruning + variational inference

- **2019-CVPR-Variational Convolutional Neural Network Pruning**
- **2021-AISTAT-Dirichlet Pruning for Neural Network Compression**
- 2017-NeurIPS-Bayesian Compression for Deep Learning(NeurlPS17)

Pruning as sub-networks search:

- 2021-ECCV-Exploration and Estimation for Model Compression

Distillation:

- 2015-NIPS-[Bayesian dark knowledge](#)
- 2019-CVPR-[Variational Information Distillation for Knowledge Transfer](#)

Quantization:

- 2020-NIPS-[Bayesian Bits: Unifying Quantization and Pruning](#)
- 2018-ICLR-[Variational Network Quantization](#)

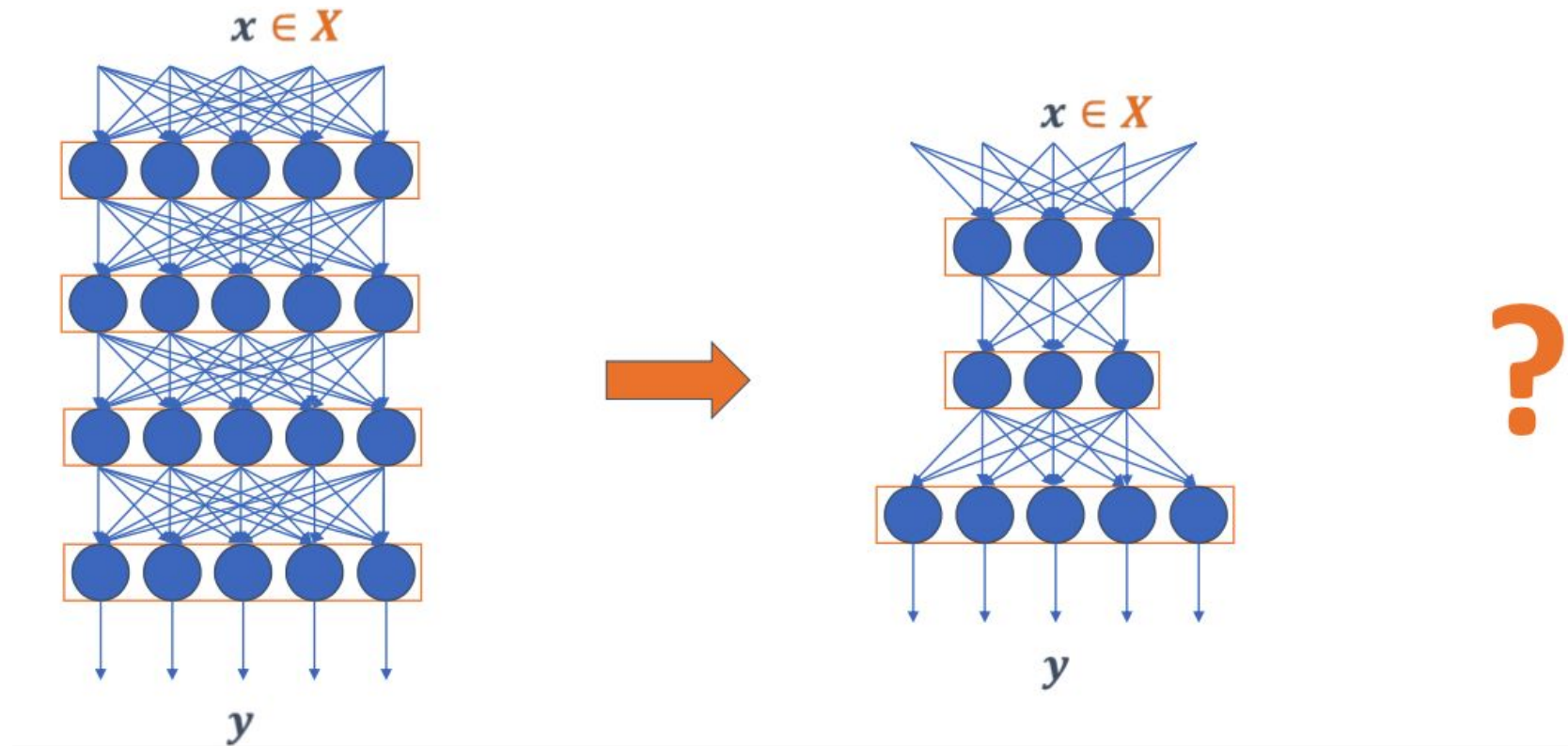
Dropout + compression:

- 2017-NIPS-[Structured Bayesian Pruning via Log-Normal Multiplicative Noise](#)
- 2017-ICML-[Variational dropout sparsifies deep neural networks](#)

Outline

- **Recall the overview of the network compression methods**
- Problem statement of Channel pruning
- Variational Inference
- Variational Convolutional Neural Network Pruning
 - Algorithm
 - Minimize KL divergence
 - Experiments
- Dirichlet Pruning for Neural Network Compression
 - Algorithm
 - Minimize KL divergence
 - Experiments

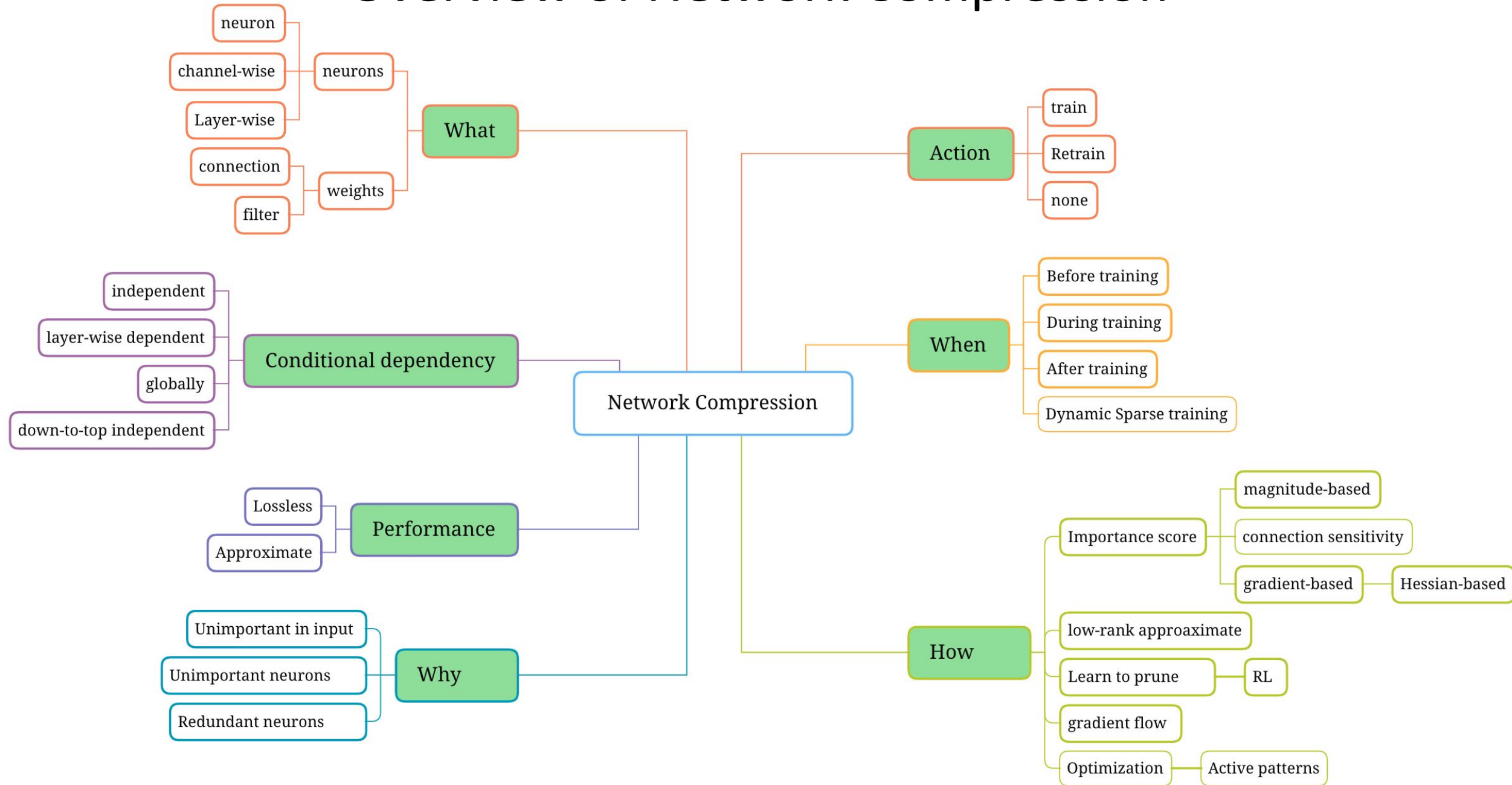
Network compression: find a smaller network as good as a large one



Background: why to prune networks

- Less resource consuming
 - Deploy to devices of limited resource, e.g., mobile phone, drone.
- Speedup inference
- Approximately same performance as the large network
- With quantization, it can be low-bit further
- Provide an insight into the presentation ability of a network
 - Which subnetwork contribute most to to task?

Overview of Network Compression



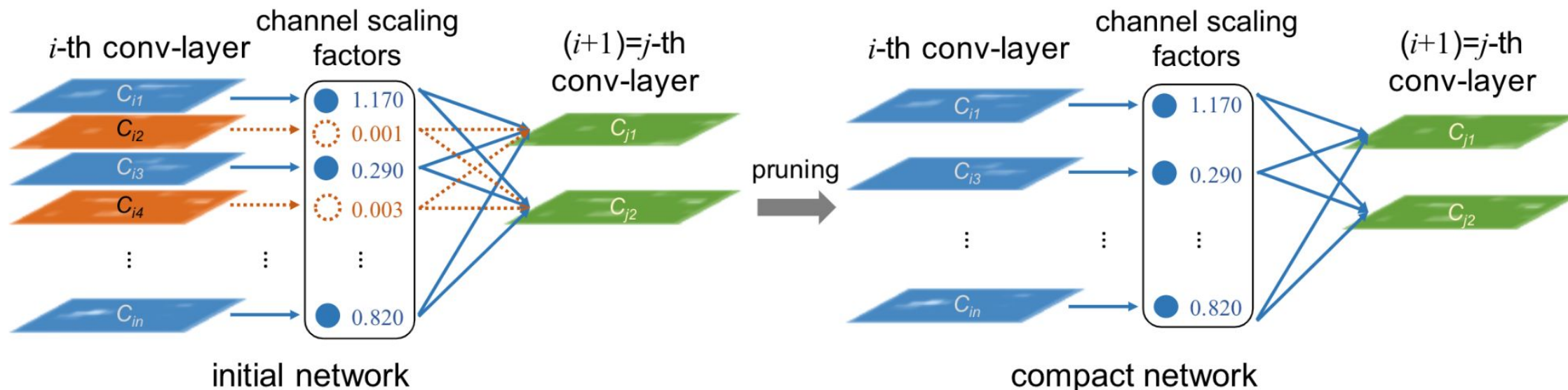
How to prune: branches of Methods

- Pruning
 - **After training**
 - OBD: Optimal brain damage
 - OBS: Optimal brain surgeon
 - WoodFisher: Efficient Second-Order Approximation for Neural Network Compression
 - Before training
 - SNIP: Single-shot network pruning based on connection sensitivity
 - GraSP: Picking winning tickets before training by preserving gradient flow
- Quantization
- Knowledge distillation
- Low-rank decomposition
- Compact architecture design (similar as architecture search)

Outline

- Recall the overview of the network compression methods
- **Problem statement of Channel pruning**
- Variational Inference
- Variational Convolutional Neural Network Pruning
 - Minimize KL divergence
 - Algorithm
 - Experiments
- Dirichlet Pruning for Neural Network Compression
 - Minimize KL divergence
 - Algorithm
 - Experiments

Problem statement of Channel pruning

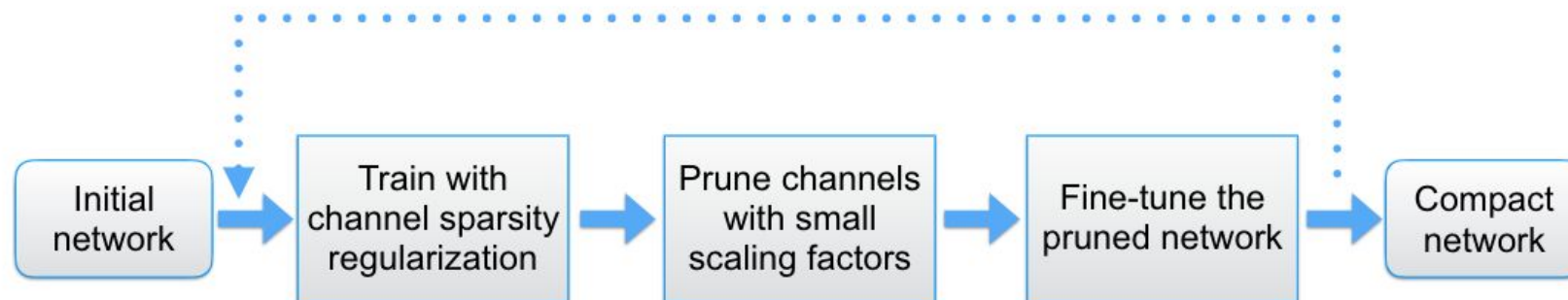


Optimization Objective: sparsity-induced penalty[1]

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

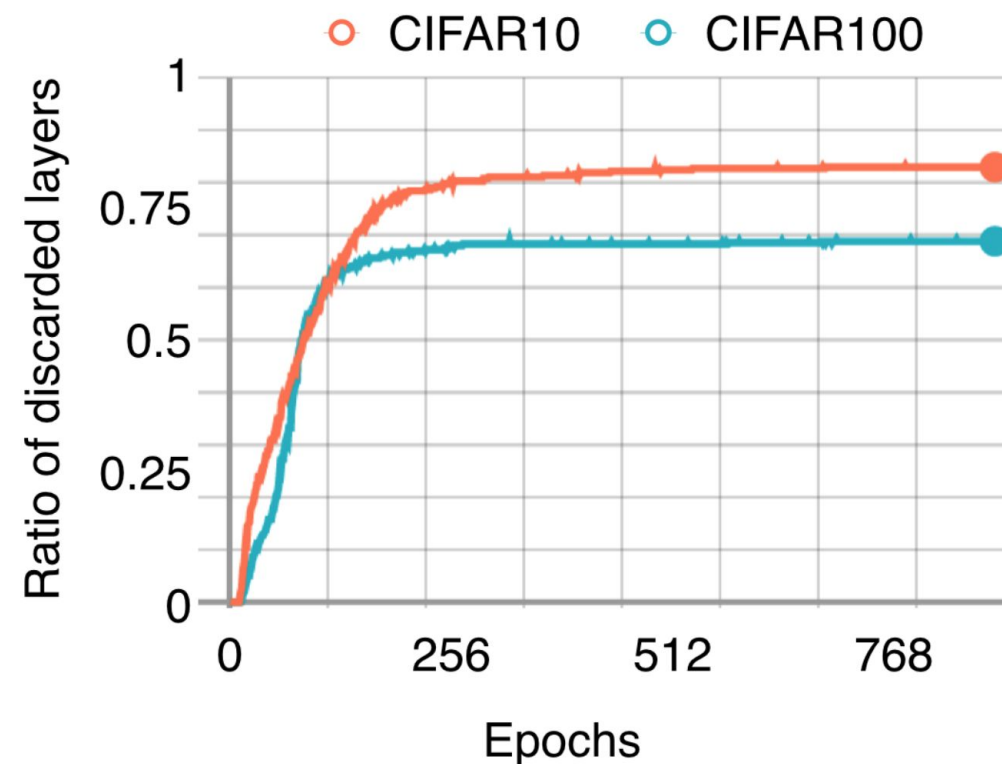
Problem statement of Channel pruning

Flow-chart of network slimming[1]:



Problems in network slimming[1]:

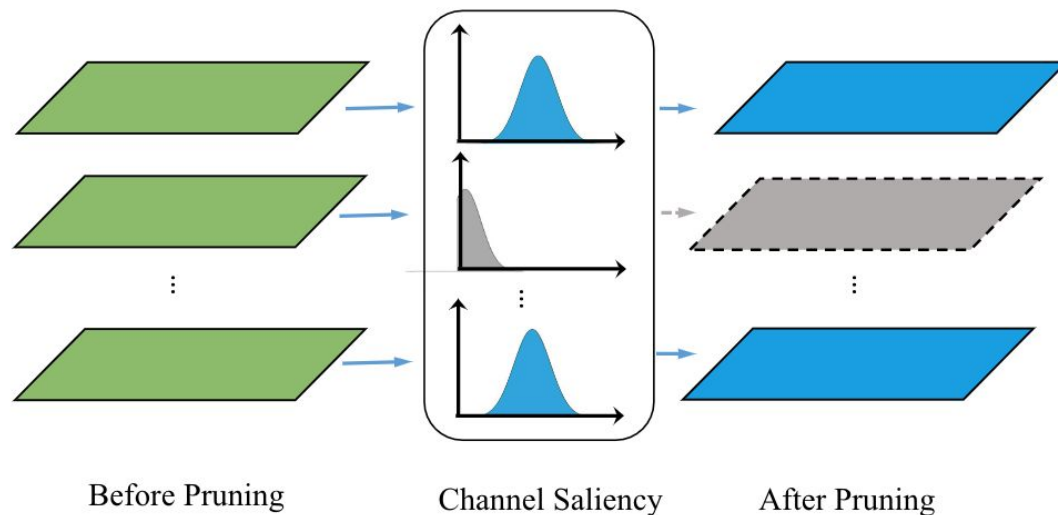
- deterministic value based pruning methods are inherently unstable
 - Scaling factor of hidden layers may change drastically during consecutive iteration
 - This results in arbitrary removal of some channels during optimization, which is NOT interpretable.



[1] Learning Efficient Convolutional Networks through Network Slimming (ICCV17)

[2] Learning Strict Identity Mappings in Deep Residual Networks (cvpr18)

Problem statement of Channel pruning



Variational Inference on Channel Importance[1]:

Problem statement:

- Consider a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$, x : input data, y : corresponding label.
- The goal is to learn a model with channel importance γ of the conditional probability $p(y|x, \gamma)$

We can learn the posterior distribution of γ with Bayes rule:

$$p(\gamma|\mathcal{D}) = p(\gamma)p(\mathcal{D}|\gamma)/p(\mathcal{D})$$

However, $p(\mathcal{D}) = \int p(\mathcal{D}, \gamma)d\gamma$ is a computation-intractable integral.

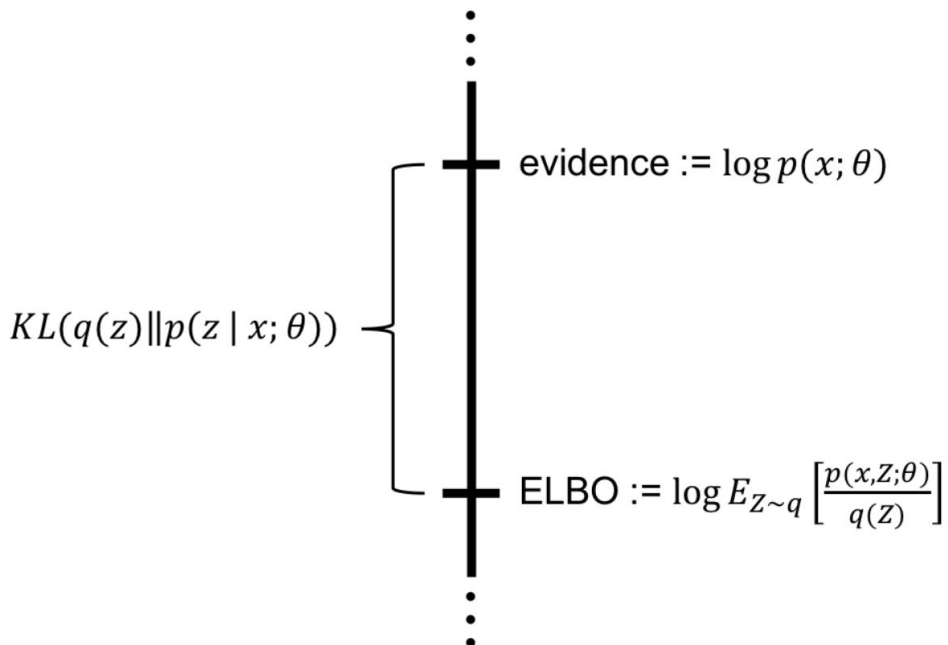
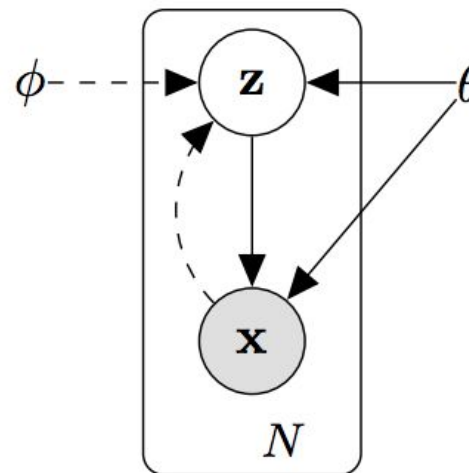
Variational Inference

Example: coin toss follows Bernoulli distribution $B(\theta)$

x : the observation of head or tail

z : the wind speed

The posterior $P(z|x)$ is intractable and we estimate it by $q(\phi)$ and will minimize their difference by KL divergence.



$$\begin{aligned}
 KL(q(z) \parallel p(z \mid x; \theta)) &:= E_{Z \sim q} \left[\log \frac{q(Z)}{p(Z \mid x; \theta)} \right] \\
 &= E_{Z \sim q} [\log q(Z)] - E_{Z \sim q} \left[\log \frac{p(x, Z; \theta)}{p(x; \theta)} \right] \\
 &= E_{Z \sim q} [\log q(Z)] - E_{Z \sim q} [\log p(x, Z; \theta)] + E_{Z \sim q} [\log p(x; \theta)] \\
 &= \log p(x; \theta) - E_{Z \sim q} \left[\log \frac{p(x, Z; \theta)}{q(z)} \right] \\
 &= \text{evidence} - \text{ELBO}
 \end{aligned}$$

Conclusion: minimizing KL divergence is equivalent to maximizing the evidence lower bound(ELBO)

Variational Inference on Channel Importance

To learn the posterior $p(\gamma|\mathcal{D}) = p(\gamma)p(\mathcal{D}|\gamma)/p(\mathcal{D})$

- Assume a parameterized distribution $q_\phi(\gamma)$ to estimate $p(\gamma|\mathcal{D})$.
- Minimize the KL divergence
- That is equivalently to maximize the ELBO as blow:

$$\mathcal{L}(\phi) = L_{\mathcal{D}}(\phi) - D_{KL}(q_\phi(\gamma)||p(\gamma)),$$

$$\text{where, } \mathcal{L}_{\mathcal{D}}(\phi) = \sum_{(x,y) \in \mathcal{D}} \mathbb{E}_{q_\phi(\gamma)} [\log p(y|x, \gamma)].$$

Outline

- Recall the overview of the network compression methods
- Problem statement of Channel pruning
- Variational Inference
- **Variational Convolutional Neural Network Pruning**
 - Minimize KL divergence
 - Algorithm
 - Experiments
- Dirichlet Pruning for Neural Network Compression
 - Minimize KL divergence
 - Algorithm
 - Experiments

Variational Convolutional Neural Network Pruning: Minimize KL divergence

ELBO as objective function:

$$\mathcal{L}(\phi) = L_{\mathcal{D}}(\phi) - D_{KL}(q_{\phi}(\gamma) || p(\gamma)),$$

$$\text{where, } \mathcal{L}_{\mathcal{D}}(\phi) = \sum_{(x,y) \in \mathcal{D}} \mathbb{E}_{q_{\phi}(\gamma)} [\log p(y|x, \gamma)].$$

To learn the log-likelihood term $L_{\mathcal{D}}(\phi)$:

- Maximize the probability of the model prediction is to minimize the prediction error.
- However, the expectation is not differentiable.
- A sampling method is used to estimate the expectation: minibatch-based Monte Carlo estimator[1]

$$\mathcal{L}_{\mathcal{D}}(\phi) \simeq \mathcal{L}_{\mathcal{D}}^A(\phi) = \frac{N}{M} \sum_{m=1}^M \log p(y_{im} | x_{im}, \gamma_{im} = f(\phi, \epsilon))$$

The reparameterization trick[1]: $\epsilon \sim \mathcal{N}(0, 1)$ $\phi = (\mu, \sigma)$ $f(\phi, \epsilon) = \mu + \sigma \cdot \epsilon$

Variational Convolutional Neural Network Pruning: Minimize KL divergence

$$\mathcal{L}(\phi) = L_{\mathcal{D}}(\phi) - D_{KL}(q_{\phi}(\gamma) || p(\gamma)),$$

To learn the KL divergence:

$$q_{\phi}(\gamma) = \prod_{i=1}^C q(\gamma_i), \quad \gamma_i \sim \mathcal{N}(\mu_i, \sigma_i).$$

$$p(\gamma) = \prod_{i=1}^C p(\gamma_i), \quad \gamma_i \sim \mathcal{N}(0, \sigma_i^*),$$

Assumptions:

- the prior encourage γ towards zero to have sparse network.
- Both distribution are Gaussian, which makes it possible to compute the KL.
- The variance of both distribution are identical

$$\begin{aligned} D_{KL}(q_{\phi}(\gamma) || p(\gamma)) &= \sum_i D_{KL}(q_{\phi}(\gamma_i) || p(\gamma_i)) \\ &= \sum_i \log \frac{\sigma_i^*}{\sigma_i} + \frac{\sigma_i^2 + \mu_i^2}{2(\sigma_i^*)^2} - \frac{1}{2}. \\ &= \sum_i k \mu_i^2 \end{aligned}$$

K: inversely proportional to variance

Variational Convolutional Neural Network Pruning: Algorithm

Algorithm 1 Variational CNN Pruning

Input: \mathcal{N} pairs data $\{(x_i, y_i)\}_{i=1}^N$, C channels $\{\gamma_i\}_{i=1}^C$

Output: ϕ, \mathbf{w}

```
1: for epoch= 1 to  $K$  do
2:    $q_\phi(\gamma) = \prod_{i=1}^C q(\gamma_i)$ 
3:    $\gamma_i \sim \mathcal{N}(\mu_i, \sigma_i)$ 
4:    $\mathcal{L}(\phi, \mathbf{w}) \simeq \mathcal{L}_D^A(\phi, \mathbf{w}) - D_{KL}(q_\phi(\gamma)||p(\gamma)).$ 
5:   Optimize :  $\mathcal{L}(\phi, \mathbf{w})$ 
6:   Update Parameters
7:   for  $i = 1$  to  $C$  do
8:     if  $u_i < \tau, \sigma_i < \theta$  then
9:       Pruning the  $i$ -channel
10:    end if
11:  end for
12: end for
```

Variational Convolutional Neural Network Pruning: Algorithm

NOTE:

- The algorithm is implemented in BN:

$$BN(x) = \frac{x_{in} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}; x_{out} = \gamma \cdot BN(x) + \beta$$

It extends the scale factor on shift term:

$$x_{out} = \gamma \cdot BN(x) + \tilde{\beta}, \text{ where, } \tilde{\beta} = \gamma \cdot \beta.$$

Thus, no extra parameters on the channel importances are required.

- No retraining is required.

Variational Convolutional Neural Network Pruning: Experiments

Model	Accuracy	Channels	Pruned	Parameters	Pruned	FLOPs	Pruned
VGG-16 Base	93.25%	4224	-	14.71M	-	313M	-
VGG-16 Pruned	93.18%	1599	62%	3.92M	73.34%	190M	39.10%
DenseNet-40 Base	94.11%	9360	-	1.04M	-	282M	-
DenseNet-40 Pruned	93.16%	3705	60%	0.42M	59.67%	156M	44.78%
ResNet-20 Base	92.01%	1808	-	0.21M	-	8.9M	-
<u>ResNet-20 Pruned</u>	91.66%	1114	38%	0.17M	20.41%	7.5M	16.47%
ResNet-56 Base	93.04%	4496	-	0.57M	-	22.3M	-
<u>ResNet-56 Pruned</u>	92.26%	2469	45%	0.46M	20.49%	17.8M	20.30%
ResNet-110 Base	93.21%	8528	-	1.12M	-	42.4M	-
<u>ResNet-110 Pruned</u>	92.96%	3121	63%	0.66M	41.27%	26.9M	36.44%
ResNet-164 Base	93.58%	12560	-	1.68M	-	62.4M	-
ResNet-164 Pruned	93.16%	3238	74%	0.73M	56.70%	31.8M	49.08%

Table 1. Accuracy and pruning ratio on CIFAR-10. We count pruned channels, parameters and FLOPs over different deep models, and the accuracy of pruned models are reported without retraining stage. We train these models from scratch without pruning as baseline in our experiments.

Variational Convolutional Neural Network Pruning: Experiments

Model	Accuracy	Channels	Pruned	Parameters	Pruned	FLOPs	Pruned
VGG-16 Base	73.26%	4224	-	14.71M	-	313M	-
VGG-16 Pruned	73.33%	2883	32%	9.14M	37.87%	256M	18.05%
DenseNet-40 Base	74.64%	9360	-	1.04M	-	282M	-
DenseNet-40 Pruned	72.19%	5851	37%	0.65M	37.73%	218M	22.67%
ResNet-164 Base	75.56%	12560	-	1.68M	-	62.4M	-
ResNet-164 Pruned	73.76%	6681	47%	1.38M	17.59%	45.4M	27.16%

Table 2. Accuracy and pruning ratio on CIFAR-100. We count pruned channels, parameters and FLOPs on VGG-16, DenseNet-40 and ResNet-164.

Model	Top-1	Top-5	Channels Pruned	
ResNet-50 Base	75.1%	92.8%	26560	-
ResNet-50 [29]	72.8%	91.1%	18592	30%
ResNet-50 Ours	75.2%	92.1%	15920	40%

Table 3. Performance and Comparison on ImageNet.

Variational Convolutional Neural Network Pruning: Experiments

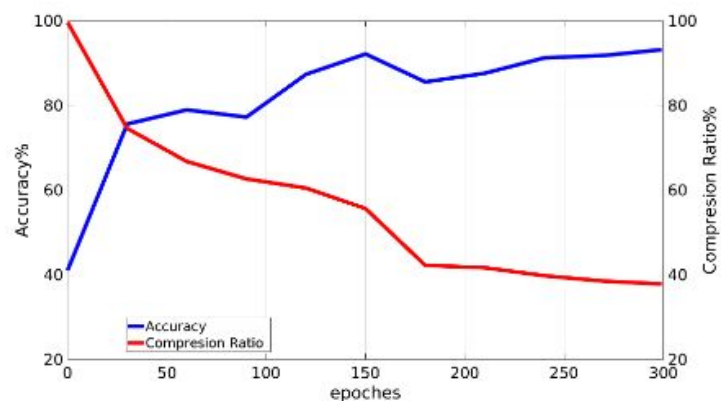
Compare to [1] without fine-tuning stage:

Dataset	Model	Accuracy	Channels	Params
CIFAR-10	Dense40* [27]	89.5%	60%	54%
	Denset40 Ours	93.1%	60%	59%
	Res164* [27]	47.7%	60%	34%
	Res164 Ours	93.1%	74%	56%
CIFAR-100	Dense40* [27]	67.7%	40%	35%
	Dense40 Ours	72.1%	37%	38%
	Res164* [27]	48.0%	40%	13%
	Res164 Ours	73.7%	47%	17%

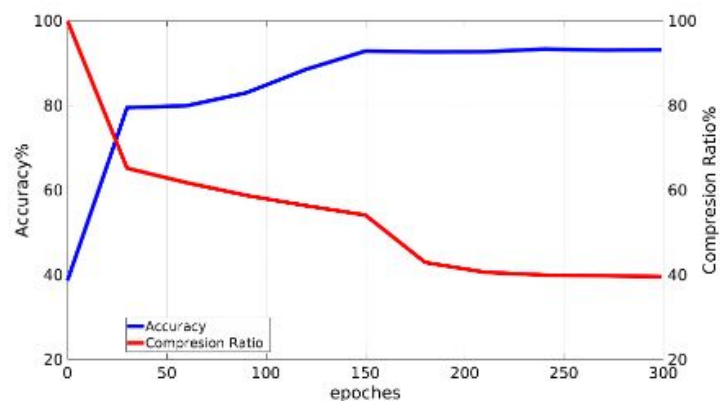
Table 4. Comparison with other method on CIFAR dataset. * denotes pruning without fine-tuning stage.

Variational Convolutional Neural Network Pruning: Experiments

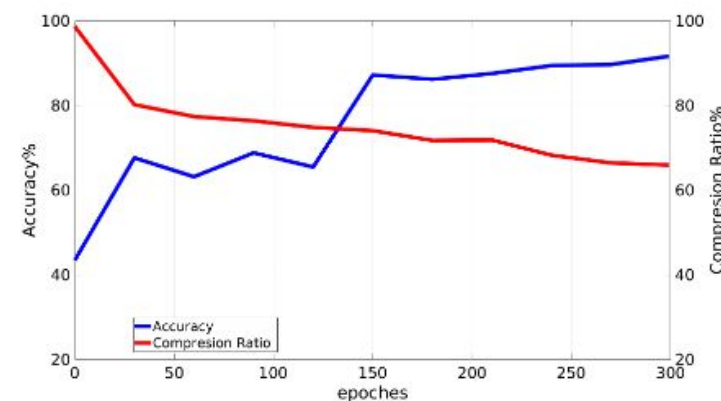
The stability of this method:



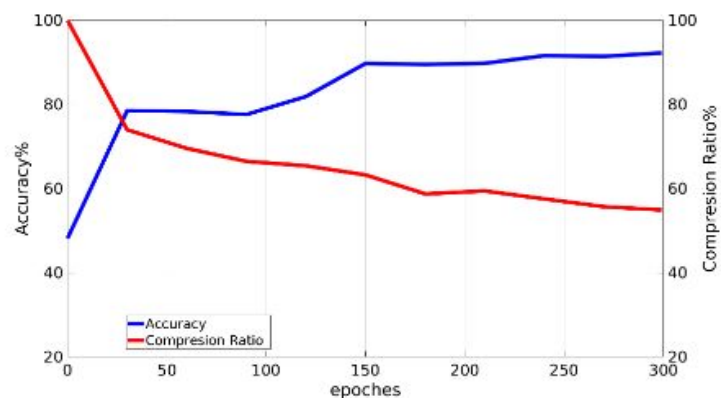
(a) VGG-16



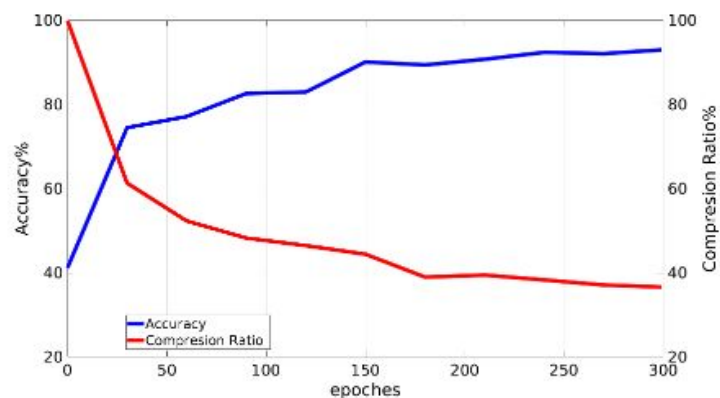
(b) DenseNet-40



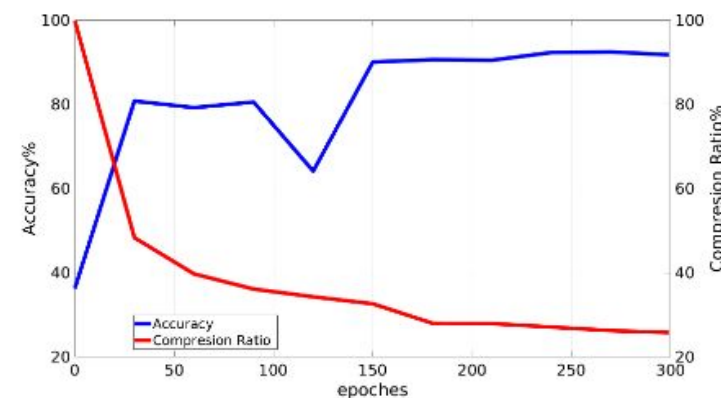
(c) ResNet-20



(c) ResNet-56



(d) ResNet-110



(e) ResNet-164

Figure 3. Compression and Accuracy Curves. We show the details of compression process about VGG-16, DenseNet-40 and ResNets. Best view in color.

Variational Convolutional Neural Network Pruning: Experiments

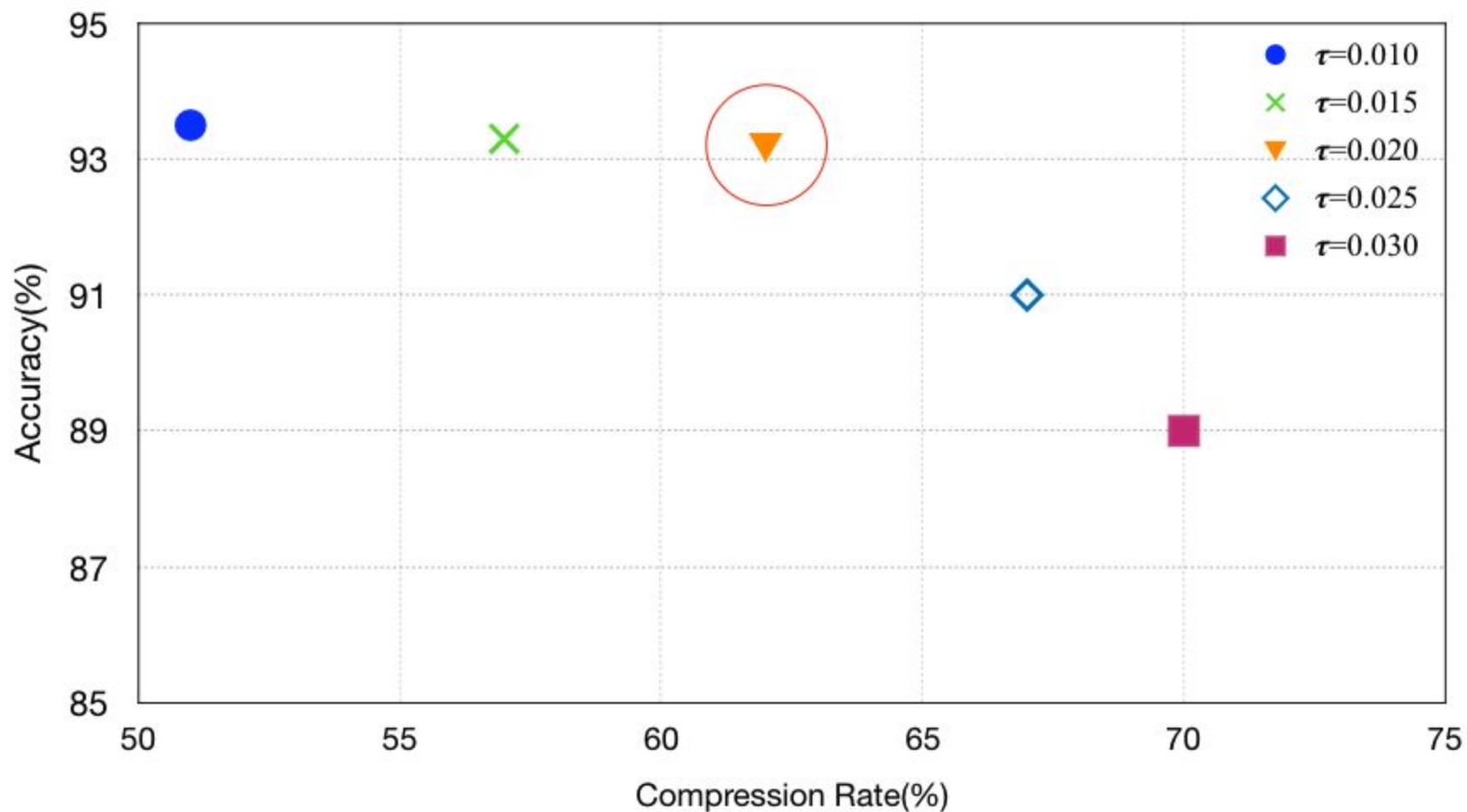


Figure 5. Sensitivity Analysis. We report compression rate and accuracy at different value of threshold τ . Best view in color.

Outline

- Recall the overview of the network compression methods
- Problem statement of Channel pruning
- Variational Inference
- Variational Convolutional Neural Network Pruning
 - Minimize KL divergence
 - Algorithm
 - Experiments
- **Dirichlet Pruning for Neural Network Compression**
 - Minimize KL divergence
 - Algorithm
 - Experiments

Dirichlet Pruning for Neural Network Compression: Minimize KL divergence

ElBO as objective function:

$$D_{KL}(q(\mathbf{s}_l) || (p(\mathbf{s}_l | \mathcal{D})))$$

which is equivalent to maximizing,

$$\int q(\mathbf{s}_l) \log p(\mathcal{D} | \mathbf{s}_l) d\mathbf{s}_l - D_{KL}[q(\mathbf{s}_l) || p(\mathbf{s}_l)],$$

Prior over importance score: $p(\mathbf{s}_l) = \text{Dir}(\mathbf{s}_l; \boldsymbol{\alpha}_0)$.

Posterior over the importance score: $q(\mathbf{s}_l) = \text{Dir}(\mathbf{s}_l; \boldsymbol{\phi}_l)$.

$$\begin{aligned} D_{kl}[q(\mathbf{s}_l | \boldsymbol{\phi}_l) || p(\mathbf{s}_l | \boldsymbol{\alpha}_0)] &= \log \Gamma\left(\sum_{j=1}^{D_l} \phi_{l,j}\right) - \\ &- \log \Gamma(D_l \alpha_0) - \sum_{j=1}^{D_l} \log \Gamma(\phi_{l,j}) + D_l \log \Gamma(\alpha_0) \\ &+ \sum_{j=1}^{D_l} (\phi_{l,j} - \alpha_0) \left[\psi(\phi_{l,j}) - \psi\left(\sum_{j=1}^{D_l} \phi_{l,j}\right) \right], \end{aligned}$$

Dirichlet Pruning for Neural Network Compression: Minimize KL divergence

ELBO as objective function:

$$D_{KL}(q(\mathbf{s}_l) || (p(\mathbf{s}_l | \mathcal{D})))$$

which is equivalent to maximizing,

$$\int q(\mathbf{s}_l) \log p(\mathcal{D} | \mathbf{s}_l) d\mathbf{s}_l - D_{KL}[q(\mathbf{s}_l) || p(\mathbf{s}_l)],$$

Prior over importance score: $p(\mathbf{s}_l) = \text{Dir}(\mathbf{s}_l; \boldsymbol{\alpha}_0).$

Posterior over the importance score: $q(\mathbf{s}_l) = \text{Dir}(\mathbf{s}_l; \boldsymbol{\phi}_l).$

Dirichlet Pruning for Neural Network Compression: Algorithm

Algorithm 1 Dirichlet Pruning

Require: A pre-trained model, \mathcal{M}_θ (parameters are denoted by θ).

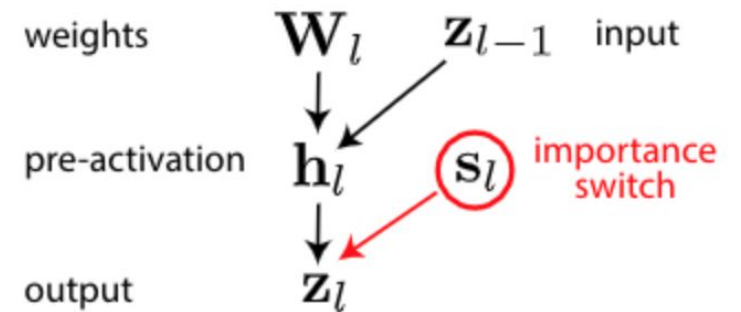
Ensure: Compressed model $\hat{\mathcal{M}}_{\hat{\theta}}$ (reduced parameters are denoted by $\hat{\theta}$).

Step 1. Add importance switches per layer to \mathcal{M}_θ .

Step 2. Learn the importance switches via optimizing eq. 7, with freezing θ .

Step 3. Remove unimportant channels according to the learned importance.

Step 4. Re-train $\hat{\mathcal{M}}_{\hat{\theta}}$ with remaining channels.



NOTE: this method requires re-training.

Dirichlet Pruning for Neural Network Compression: Experiments

Method	Error	FLOPs	Parameters
Dirichlet (ours)	8.48	38.0M	0.84M
Hrank [25]	8.77	73.7M	1.78M
BC-GNJ [27]	8.3	142M	1.0M
BC-GHS [27]	9.0	122M	0.8M
RDP [33]	8.7	172M	3.1M
GAL-0.05 [26]	7.97	189.5M	3.36M
SSS [17]	6.98	183.1M	3.93M
VP [43]	5.72	190M	3.92M

Table 2: **VGG-16** on CIFAR-10. Dirichlet pruning produces significantly smaller and faster models.

NOTE: VP is the method of “variational convolutional network pruning”

Dirichlet Pruning for Neural Network Compression: Experiments

Method	Error	FLOPs	Params
Dirichlet (150)	1.1	168K	6K
Dirichlet (mean)	1.1	140K	5.5K
Dirichlet (joint)	1.1	158K	5.5K
BC-GNJ [27]	1.0	288K	15K
BC-GHS [27]	1.0	159K	9K
RDP [33]	1.0	117K	16K
FDOO (100K) [36]	1.1	113K	63K
FDOO (200K) [36]	1.0	157K	76K
GL [39]	1.0	211K	112K
GD [35]	1.1	273K	29K
SBP [32]	0.9	226K	99K

Table 1: The structured pruning of **LeNet-5**. The

Method	Error	FLOPs	Parameters
Dirichlet (ours)	9.13	13.64M	0.24M
Hrank [25]	9.28	32.53M	0.27M
GAL-0.8 [26]	9.64	49.99M	0.29M
CP [44]	9.20	62M	-

Table 3: **ResNet-56** on CIFAR-10. Our method out-

Method	Error	Comp. Rate	Params
Dirichlet (ours)	4.5	52.2%	17.4M
L_0 ARM [24]	4.4	49.9%	18.3M
L_0 ARM [24]	4.3	49.6%	18.4M

Table 4: **WideResNet-28-10** on CIFAR-10. Com-

Dirichlet Pruning for Neural Network Compression: Experiments

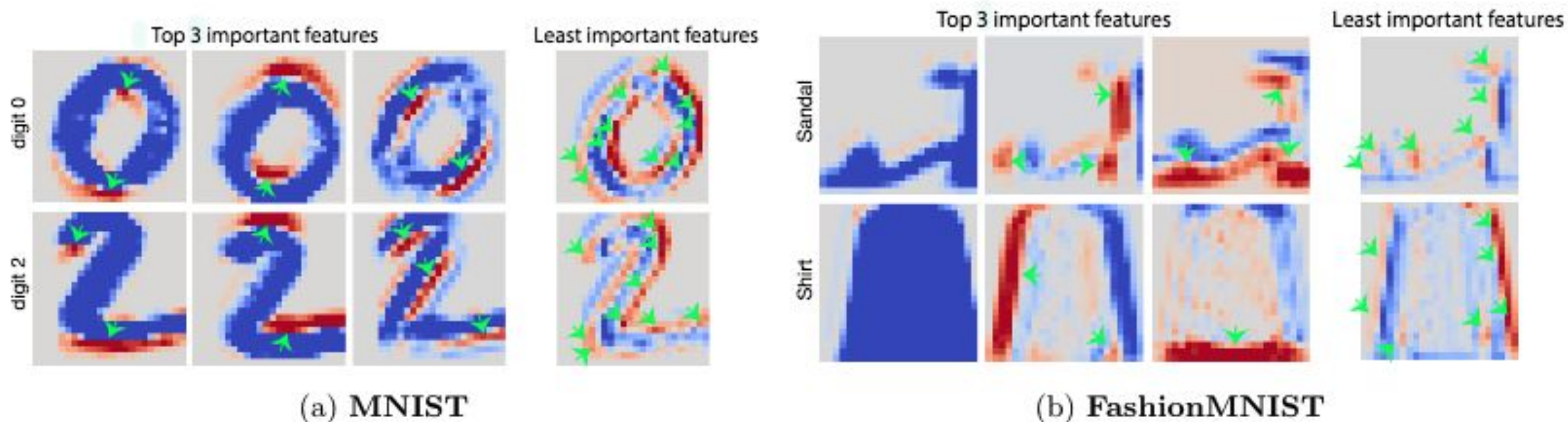


Figure 4: Visualization of learned features for two examples from MNIST and FashionMNIST data for top three (the most important) features and bottom one (the least important) feature. Green arrows indicate where high activations incur. The top, most significant features exhibit strong activations in only a few *class-distinguishing* places in the pixel space. Also, these features exhibit the complementary nature, i.e., the activated areas in the pixel space do not overlap among the top 3 important features. On the other hand, the bottom, least significant features are more fainter and more scattered.

Thanks! Q&A

	When	What	How	Action	Performance	Dependency	Why
Epsilon-ResNet[1]	During training	Layers	Magnitude of the feature map	prune-retrain	Approximate	-	Unimportant layers
ScalingUp[2]	After training	Neurons	MILP optimization	None	Lossless	Globally	Unimportant + redundant neurons
NISP[6]	After training	Neurons	Binary integer program	Finetune	Approximate	Down-to-top	
Surrogate[7]	After training	Weights	Hessian-based		Approximate	Independent	Weights which don't change Loss
LotteryTicket [3]	After training	Weights	Magnitude-based	Retrain	Approximate	-	Weight initialization + Data
SNIP[4]	Before training	Weights	Connection sensitivity	Train	Approximate	-	Data
PickingWinningTickets[5]	Before training	Weights	Preserve Gradient flow	Train	Approximate	Globally	Data