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(NeurIPS17)

Pruning as sub-networks search:

- 2021-ECCV-Exploration and Estimation for Model Compression

Distillation:

- 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

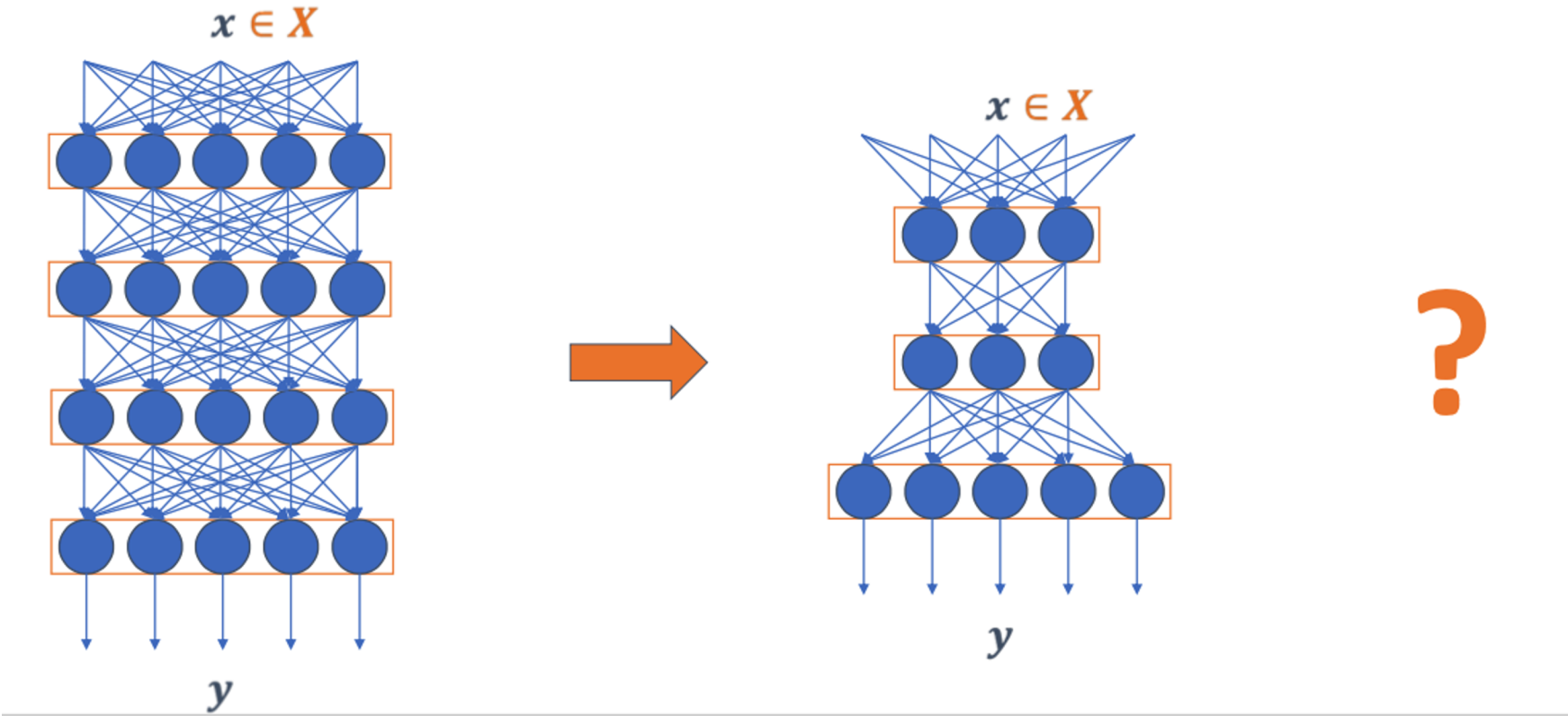
Outline

- **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

Background: why to prune networks

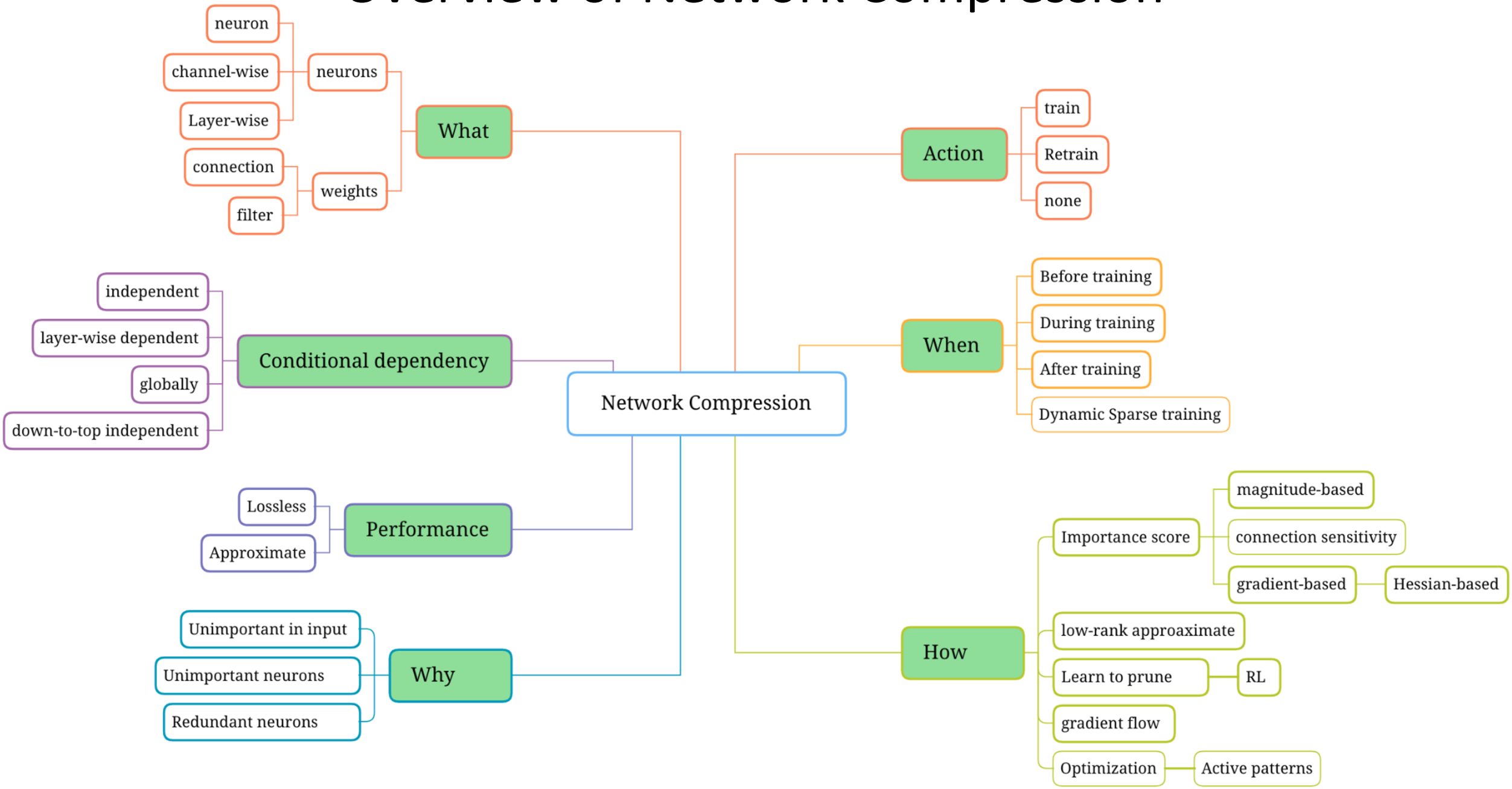
- Less resource consuming
  - Deploy to devices of limited resource, e.g., mobile phone, drone.
- Speedup inference
- Approximately same performance as the large network
- With quantization, it can be low-bit further
- Provide an insight into the presentation ability of a network
  - Which subnetwork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

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



[1] Serra, T., Kumar, A., Xin Y. and Ramalingam, S., 2021. Scaling Up Exact Neural Network Compression by ReLU Stability. *arXiv preprint arXiv:2102.07804*.

Overview of Network Compression



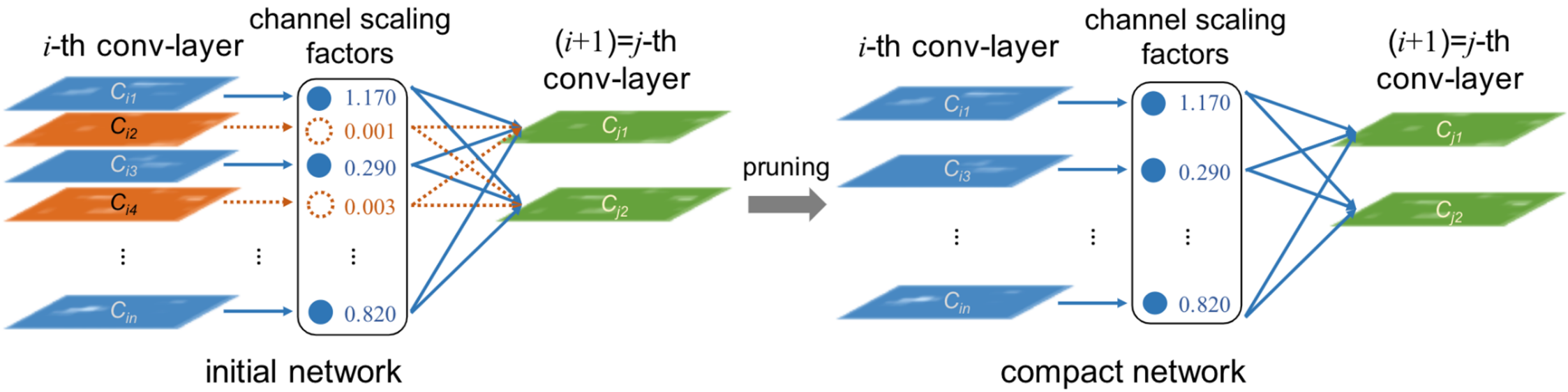
How to prune: branches of Methods

- 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



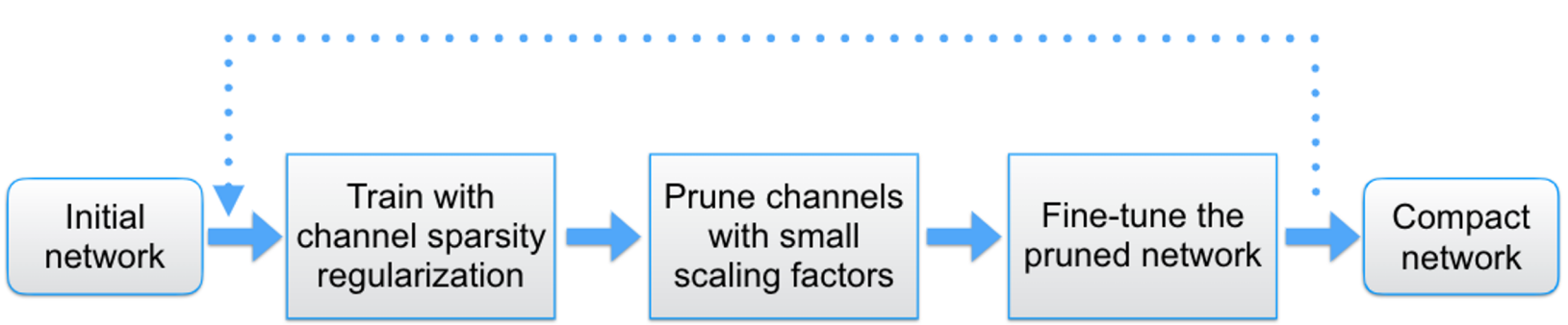
Optimization Objective: sparsity-induced penalty[1]

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

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

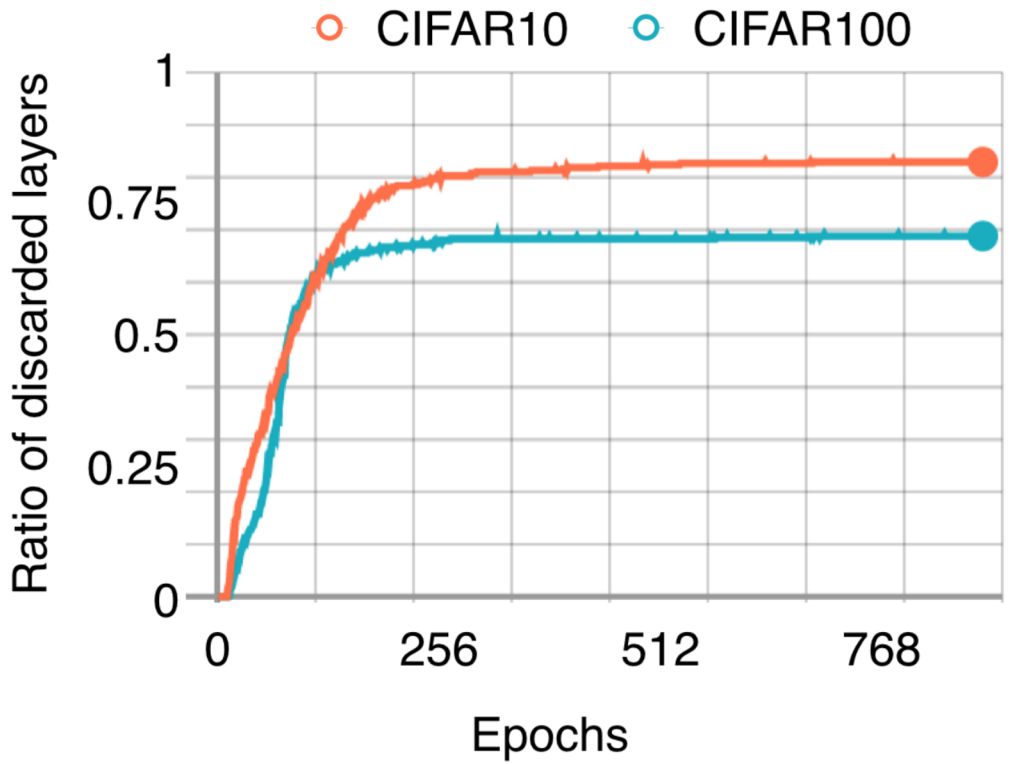
Problem statement of Channel pruning

Flow-chart of network slimming[1]:



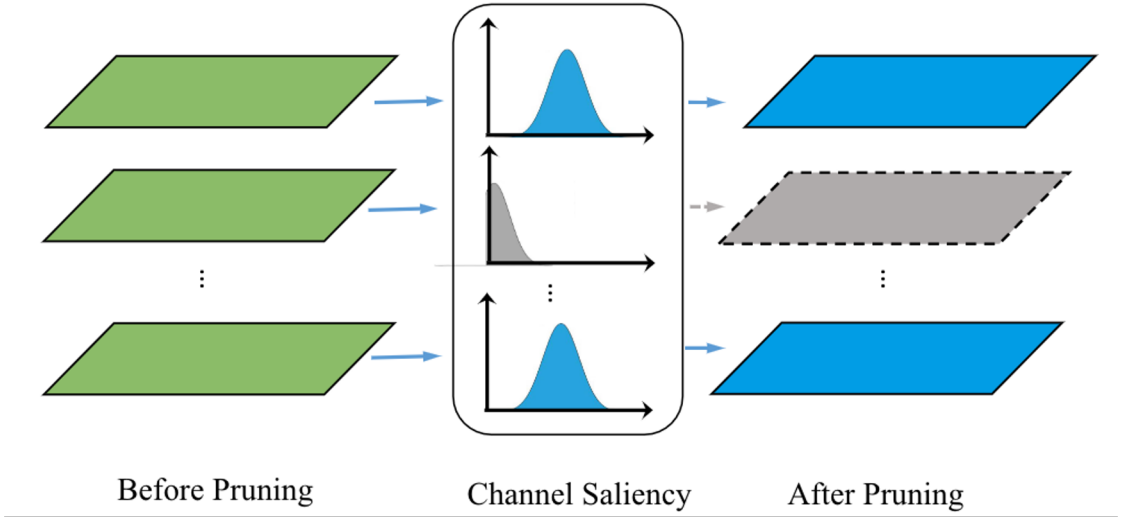
Problems in network slimming[1]:

- 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.

[1] Variational Convolutional Neural Network Pruning(cvpr19)

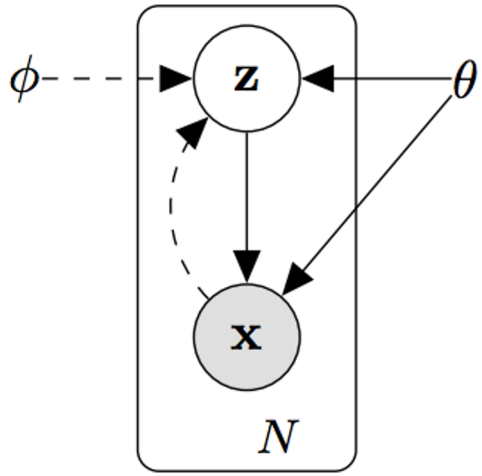
Variational Inference

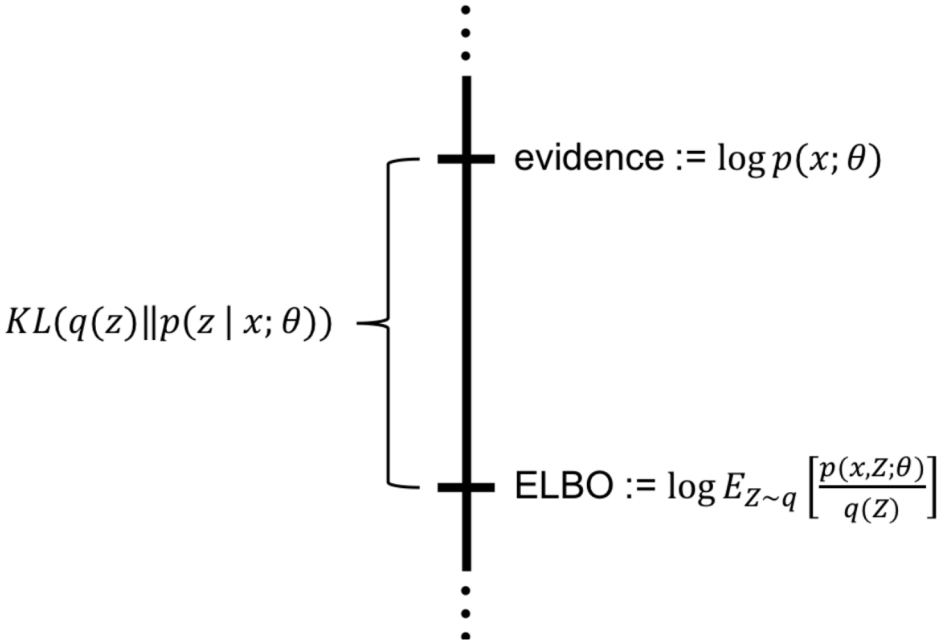
Example: coin toss follows Bernoulli distribution B(θ)

x: the observation of head or tail

z: the wind speed

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





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

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

[1] https://mbernste.github.io/posts/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:

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

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

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]

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

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

[1] Auto-Encoding Variational Bayes

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

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

To learn the KL divergence:

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

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

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

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

K: inversely proportional to variance

[1] Auto-Encoding Variational Bayes

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: $\phi, \mathbf{w}$

for epoch$= 1$ to K **do**
  $q_\phi(\gamma) = \prod_{i=1}^{C} q(\gamma_i)$
  $\gamma_i \sim \mathcal{N}(\mu_i, \sigma_i)$
  $\mathcal{L}(\phi, \mathbf{w}) \simeq \mathcal{L}_{\mathcal{D}}^{\mathcal{A}}(\phi, \mathbf{w}) - D_{KL}(q_\phi(\gamma)||p(\gamma))$.
  Optimize : $\mathcal{L}(\phi, \mathbf{w})$
  Update Parameters
  for $i = 1$ to C **do**
    if $u_i < \tau, \sigma_i < \theta$ **then**
      Pruning the i-channel
    end if
  end for
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:

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

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	-
ResNet-20 Pruned	91.66%	1114	38%	0.17M	20.41%	7.5M	16.47%
ResNet-56 Base	93.04%	4496	-	0.57M	-	22.3M	-
ResNet-56 Pruned	92.26%	2469	45%	0.46M	20.49%	17.8M	20.30%
ResNet-110 Base	93.21%	8528	-	1.12M	-	42.4M	-
ResNet-110 Pruned	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

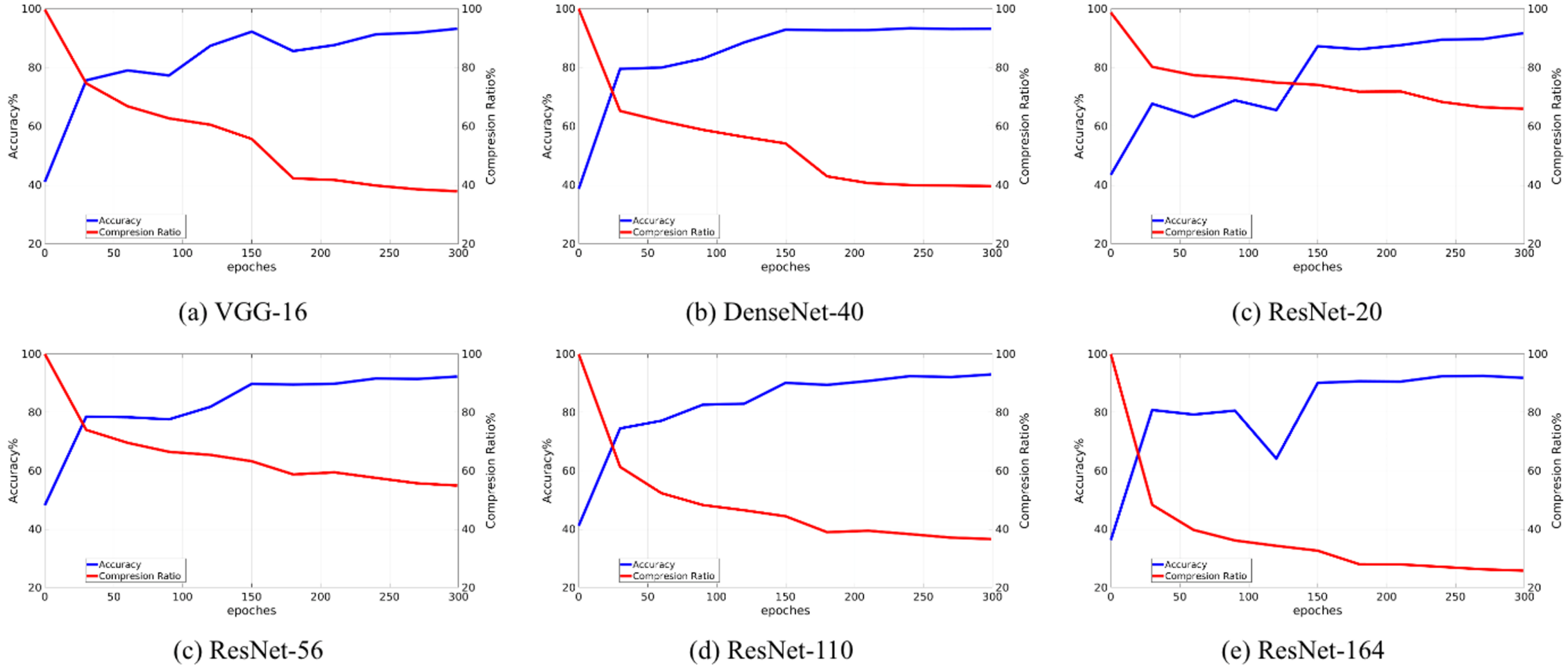
Comparse to [1] without fine-tuning stage:

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

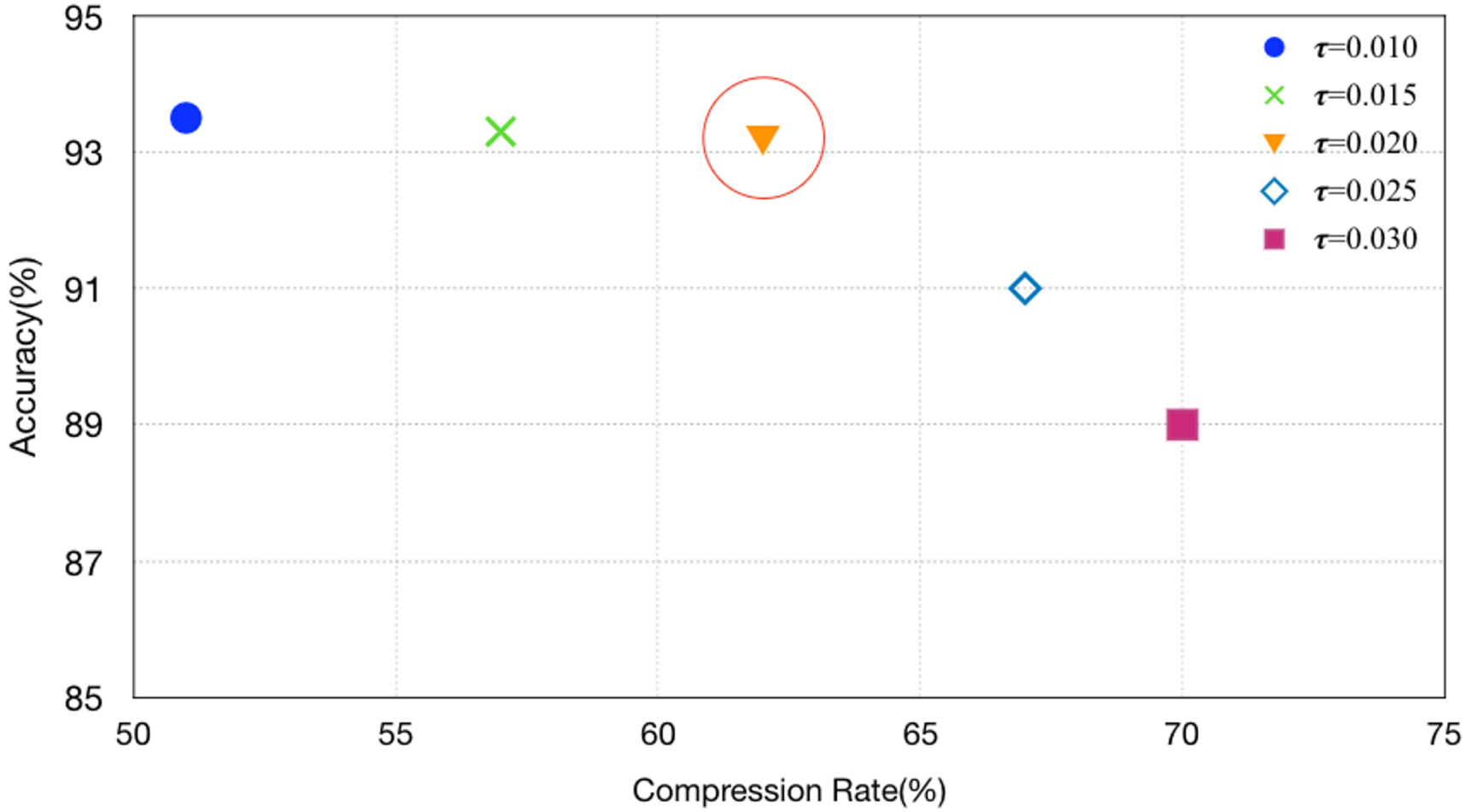


Figure 5. Sensitivity Analysis. We report compression rate and accuracy at different value of threshold τ. Best view in color.

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,

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

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

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

Why choose Dirichlet:

1. As a sample from this Dirichlet distribution sums to 1, each element of the sample can encode the importance of each channel in that layer.
2. it allows to learn the relative importance and thus we can rank the channels.

if set $\alpha_0 < 1$ this can induce sparse probability vector; else, all components become similar to each other.

Dirichlet Pruning for Neural Network Compression: Minimize KL divergence

ElBO as objective function:

$$\mathrm{D}_{kl}[q(\mathbf{s}_l|\boldsymbol{\phi}_l)||p(\mathbf{s}_l|\boldsymbol{\alpha}_0)] = \log\Gamma(\sum_{j=1}^{D_l}\boldsymbol{\phi}_{l,j}) -$$

$$-\log\Gamma(D_l\alpha_0) - \sum_{j=1}^{D_l}\log\Gamma(\boldsymbol{\phi}_{l,j}) + D_l\log\Gamma(\alpha_0)$$

$$+\sum_{j=1}^{D_l}(\boldsymbol{\phi}_{l,j}-\alpha_0)\left[\psi(\boldsymbol{\phi}_j) - \psi(\sum_{j=1}^{D_l}\boldsymbol{\phi}_{l,j})\right],$$

Reparametrization tricks:

1. Implicit gradient computation: compute gradient of the inverse CDF of the gamma distribution.
2. Analytic mean of Dirichlet random variable: calculate the gradient of the quantity without sampling from the posterior

$$\int q_{\boldsymbol{\phi}_l}(\mathbf{s}_l)\log p(\mathcal{D}|\mathbf{s}_l)d\mathbf{s}_l \approx \log p(\mathcal{D}|\tilde{\mathbf{s}}_l), \text{ where } \tilde{\mathbf{s}}_{l,j} = \boldsymbol{\phi}_{l,j}/\sum_{j'=1}^{D_l}\boldsymbol{\phi}_{l,j'}$$

Dirichlet Pruning for Neural Network Compression: Algorithm

Algorithm 1 Dirichlet Pruning

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

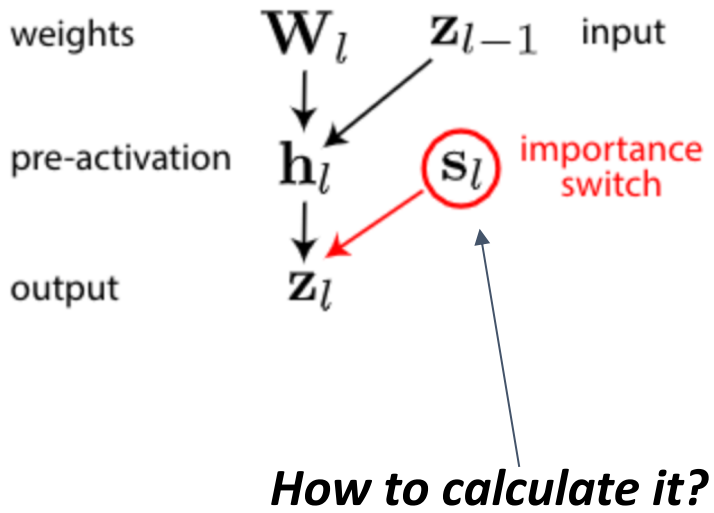
Ensure: Compressed model $\hat{\mathcal{M}}_{\hat{\theta}}$ (reduced parameters are denoted by $\hat{\theta}$).

Step 1. Add importance switches per layer to $\mathcal{M}_\theta$.

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 [14]	9.20	62M	-

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

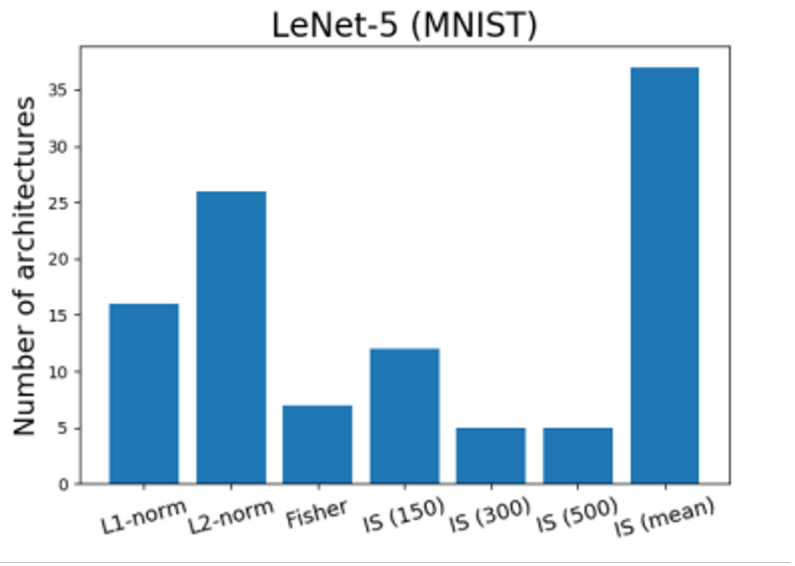
Method	Error	Comp. Rate	Params
Dirichlet (ours)	4.5	52.2%	**17.4M**
L_0 ARM [24]	4.4	49.9%	18.3M
L_0 ARM [24]	**4.3**	49.6%	18.4M

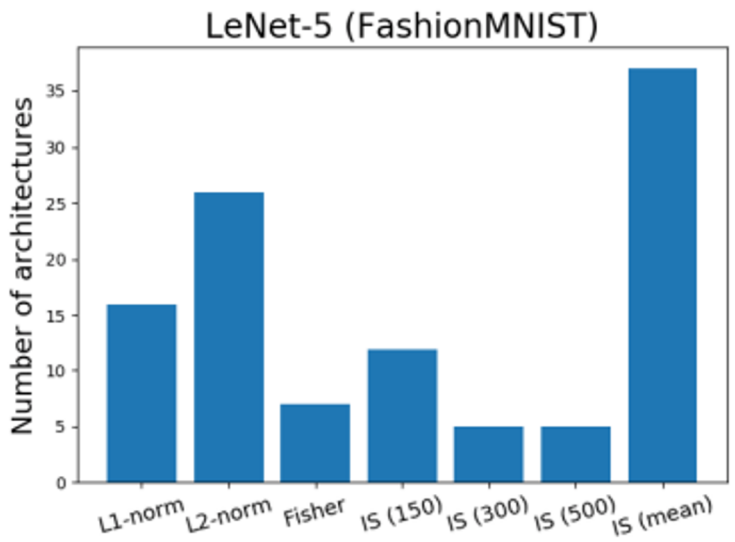
Table 4: **WideResNet-28-10** on CIFAR-10. Com-

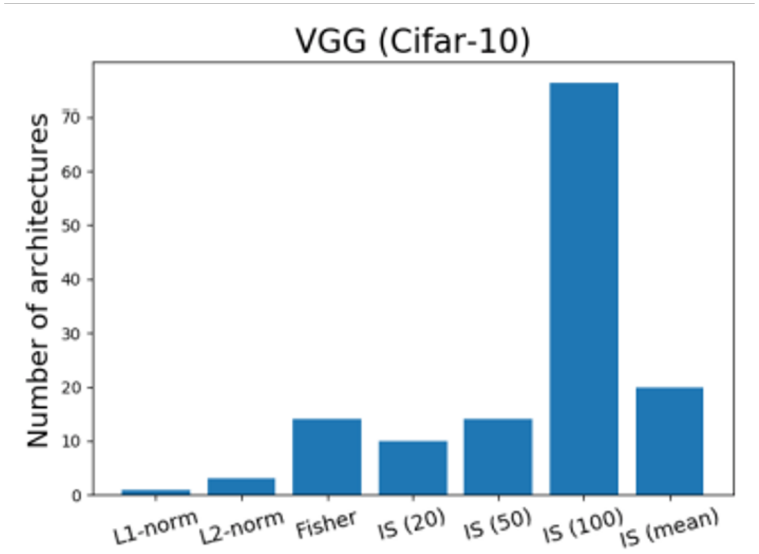
Dirichlet Pruning for Neural Network Compression: Experiments

Search over sub-architectures:

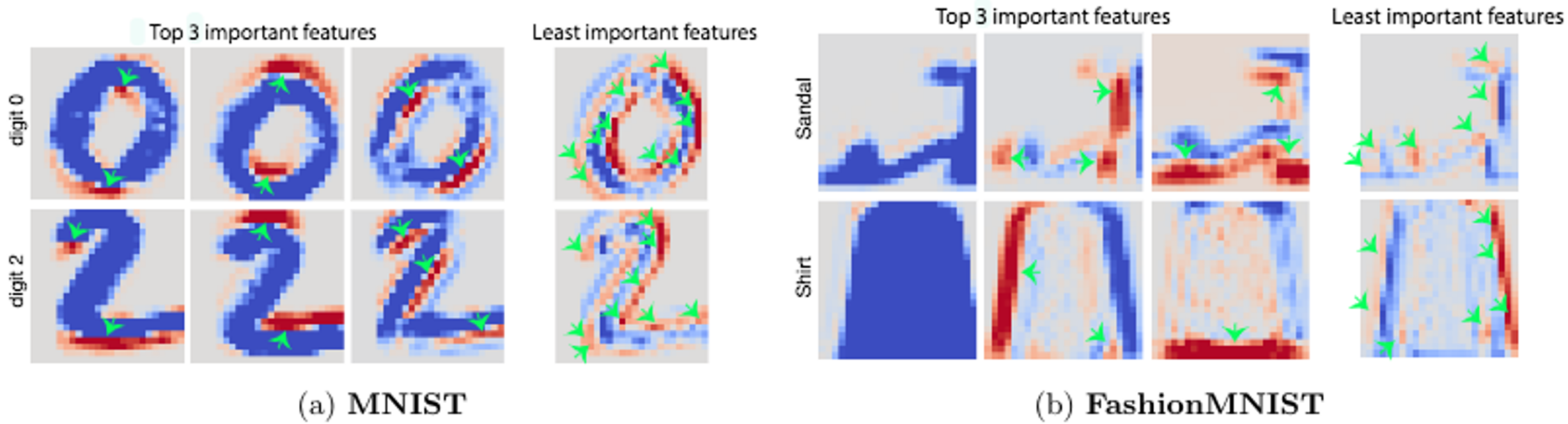
1. Tables for frequencies of best sub-architectures selected by each method:
   a. the number of times each method archives superior results to the others after pruning it to a given sub-architecture.
2. design a pool of sub-architectures with a compression rate ranging 20-60%:
   a. 108 sub-architectures for LeNet-5
   b. 128 sub-architectures for VGG.







Dirichlet Pruning for Neural Network Compression: Experiments



(a) **MNIST** (b) **FashionMNIST**

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.

Knowledge distillation:

Definition: a small model is trained to mimic a pre-trained, larger model (or ensemble of models)[1]

Motivation: the conflict constraints of training and test[2]

- During **training**, a model does not have to operate in real time and does not necessarily face restrictions on computational resources, as its primary goal is simply to extract as much structure from the given data as possible.

- But latency and resource consumption do become of concern if it is to be deployed for **inference**.

So what? we must develop ways to compress model for inference.

[1]Distilling knowledge in the neural network (NeurIPS Deep Learning Workshop, 2014)
[2] https://harvard-iacs.github.io/2020F-AC295/lectures/lecture9/presentation/lecture9.pdf

Knowledge distillation: idea[1]

Idea:

- In 2006, Buciluǎ et al. showed that it was possible to transfer knowledge from a large trained model (or ensemble of models) to a smaller model for deployment by **training it to mimic the larger model's output.**
- In 2014 Hinton et al generalized the process and gave the name **Distillation.**

Main idea of distillation is that **training and inference are 2 different tasks;** thus **a different model should be used**.

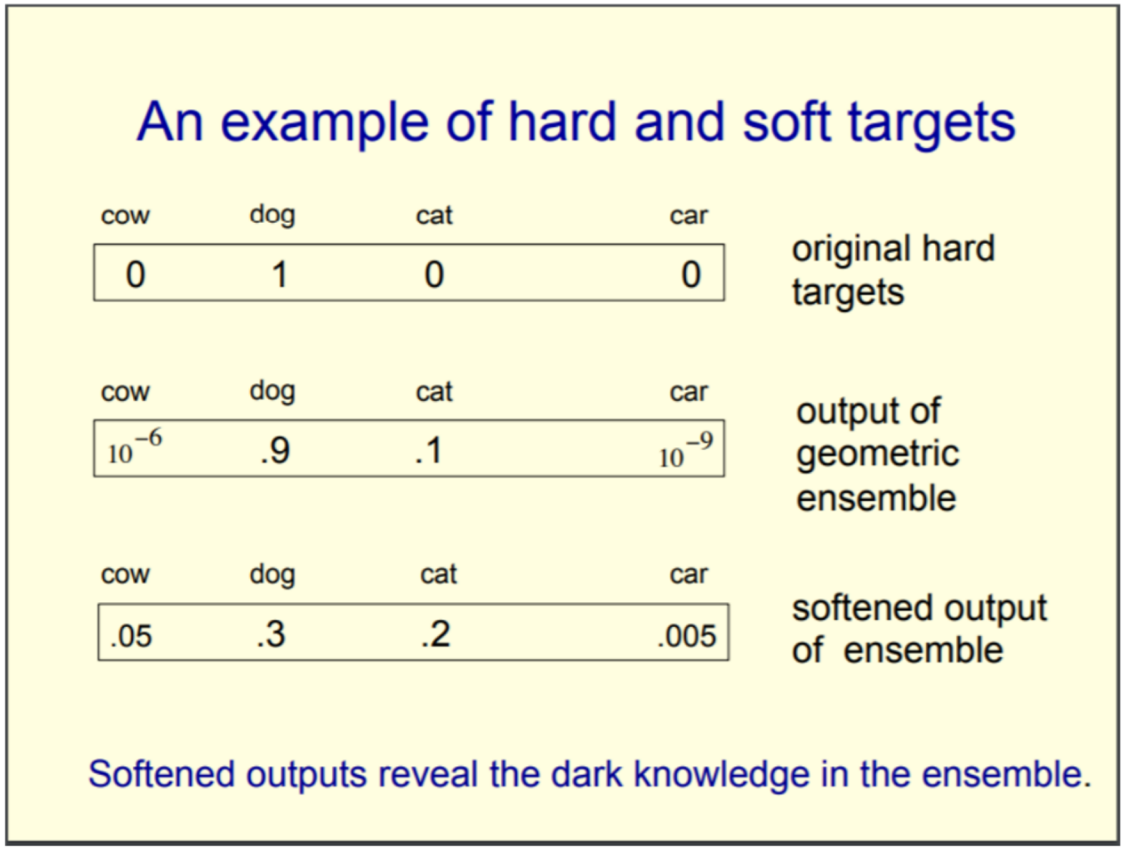
[1] https://harvard-iacs.github.io/2020F-AC295/lectures/lecture9/presentation/lecture9.pdf

Knowledge distillation: idea[1]

Modified softmax function with Temperature:

$$qi = \frac{\exp\left(\frac{z_i}{T}\right)}{\sum_j \exp\left(\frac{z_j}{T}\right)}$$

q_i : resulting probability
z_i : logit of a class
z_j : other logits
T: temperature (T=1, "hard output")

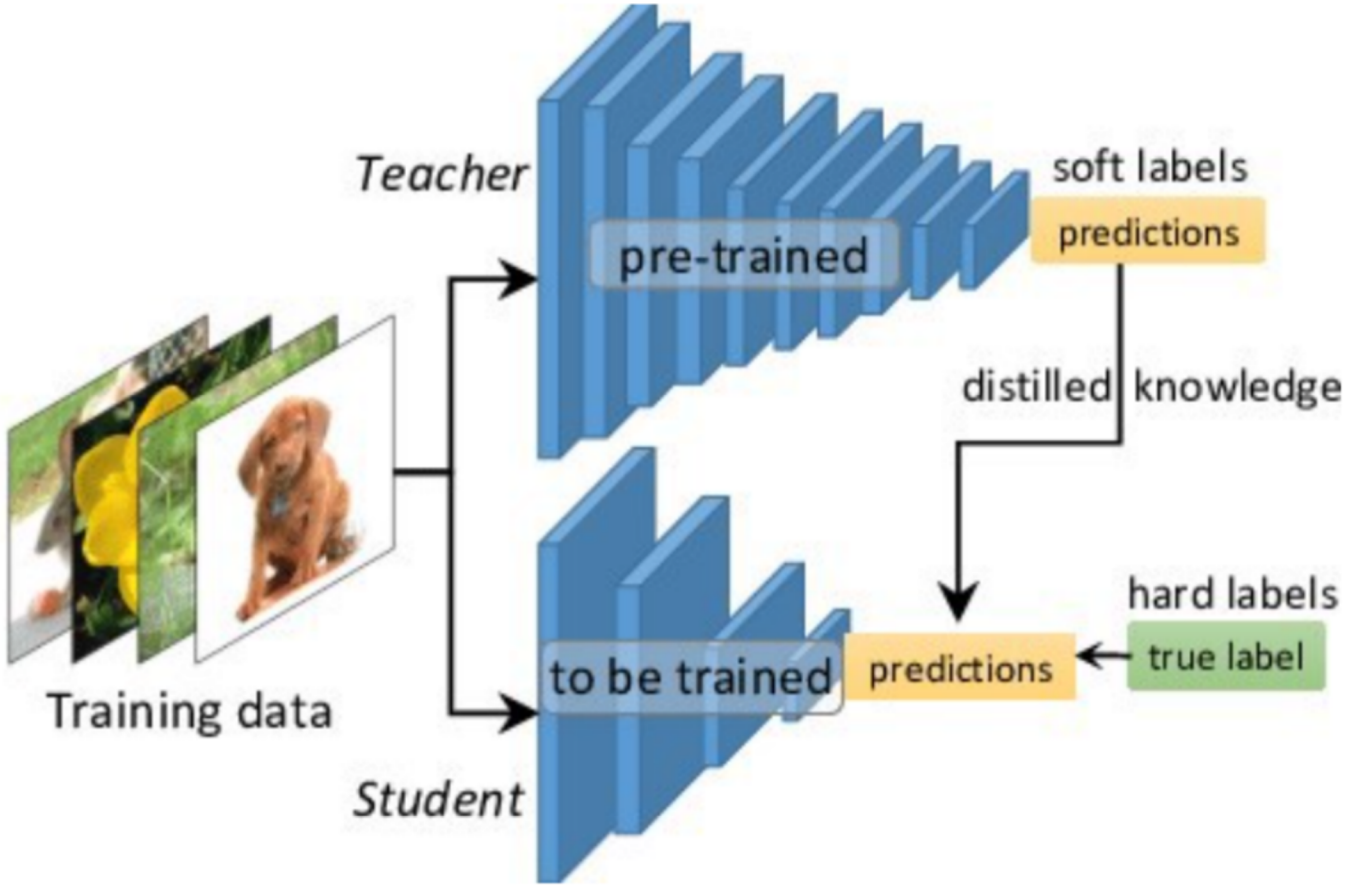


[1] https://harvard-iacs.github.io/2020F-AC295/lectures/lecture9/presentation/lecture9.pdf

Knowledge distillation: losses[1]

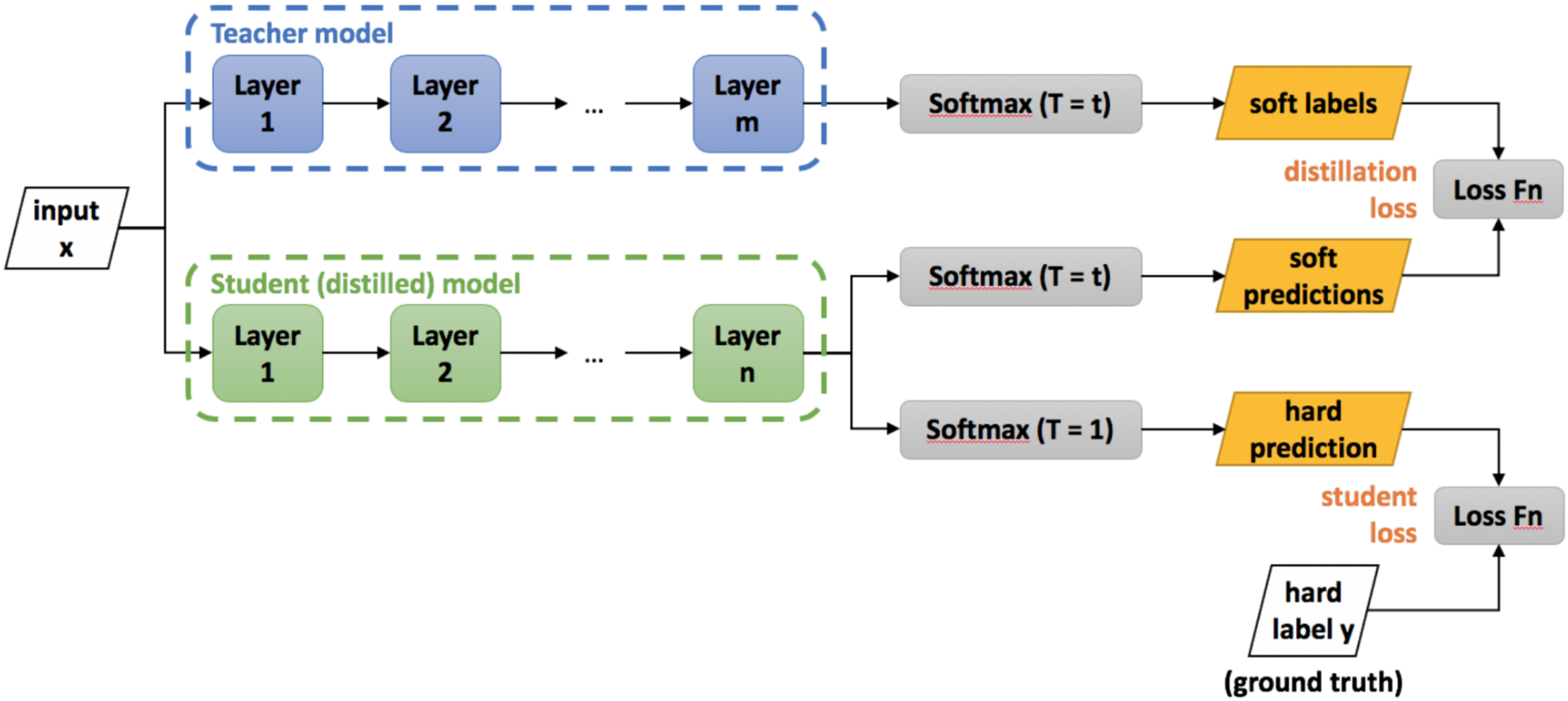
Trained to minimize the sum of two different cross entropy functions:

- one involving the original hard labels obtained using a softmax with T=1
- one involving the softened targets, T>1



[1] https://harvard-iacs.github.io/2020F-AC295/lectures/lecture9/presentation/lecture9.pdf

Knowledge distillation



[1] https://intellabs.github.io/distiller/knowledge_distillation.html

Knowledge distillation: experiments

System	Test Frame Accuracy	WER
Baseline	58.9%	10.9%
10xEnsemble	61.1%	10.7%
Distilled Single model	60.8%	10.7%

Table 1: Frame classification accuracy and WER showing that the distilled single model performs about as well as the averaged predictions of 10 models that were used to create the soft targets.

[1] https://intellabs.github.io/distiller/knowledge_distillation.html

Knowledge distillation: what's next

1: Multiple teacher (i.e. converting an ensemble into a single network).

2: Introducing a teaching assistant (the teacher first teaches the TA, who then in turn teaches the student) etc.

3: Quite young field

A **drawback** of knowledge distillation as a compression technique, therefore, is that there are many decisions that must be made up-front by the user to implement it (student network doesn't even need to have a similar structure to the teacher).

[1] https://intellabs.github.io/distiller/knowledge_distillation.html

Thanks! Q&A

Knowledge distillation: Bayesian dark knowledge

Motivation: distill a Monte Carlo approximation to the posterior predictive density into a single deep neural network, where we may have little data, and/ or where we need accurate posterior predictive densities p(y|x, D)

Inference in DNN:

1. MAP: $p(\theta|\mathcal{D}_N) \propto p(\theta) \prod_{i=1}^{N} p(y_i|x_i, \theta)$

1. Bayesian inference: $\hat{\theta} = \operatorname{argmax} p(\theta|\mathcal{D}_N) \quad p(y|x, \mathcal{D}_N) \approx p(y|x, \hat{\theta})$

Goal: train a student neural network (SNN) to approximate the Bayesian predictive distribution of the teacher, which is a Monte Carlo ensemble of teacher neural networks (TNN)

[1]Distilling knowledge in the neural network (NeurIPS Deep Learning Workshop, 2014)

	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
LotterayTicket[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	Preseve Gradient flow	Train	Approximate	Globally	Data