JUNE 18-22, 2023
CVPR
VANCOUVER, CANADA

DepGraph

Towards Any Structural Pruning

CVPR'23

Gongfan Fang, Gongfan Fang, Xinchao Wang, National University of Singapore (ECE lab); Mingli Song, Zhejiang University; Michael Bi Mi, Huawei Technologies Ltd.

Introduction

Model compression & acceleration

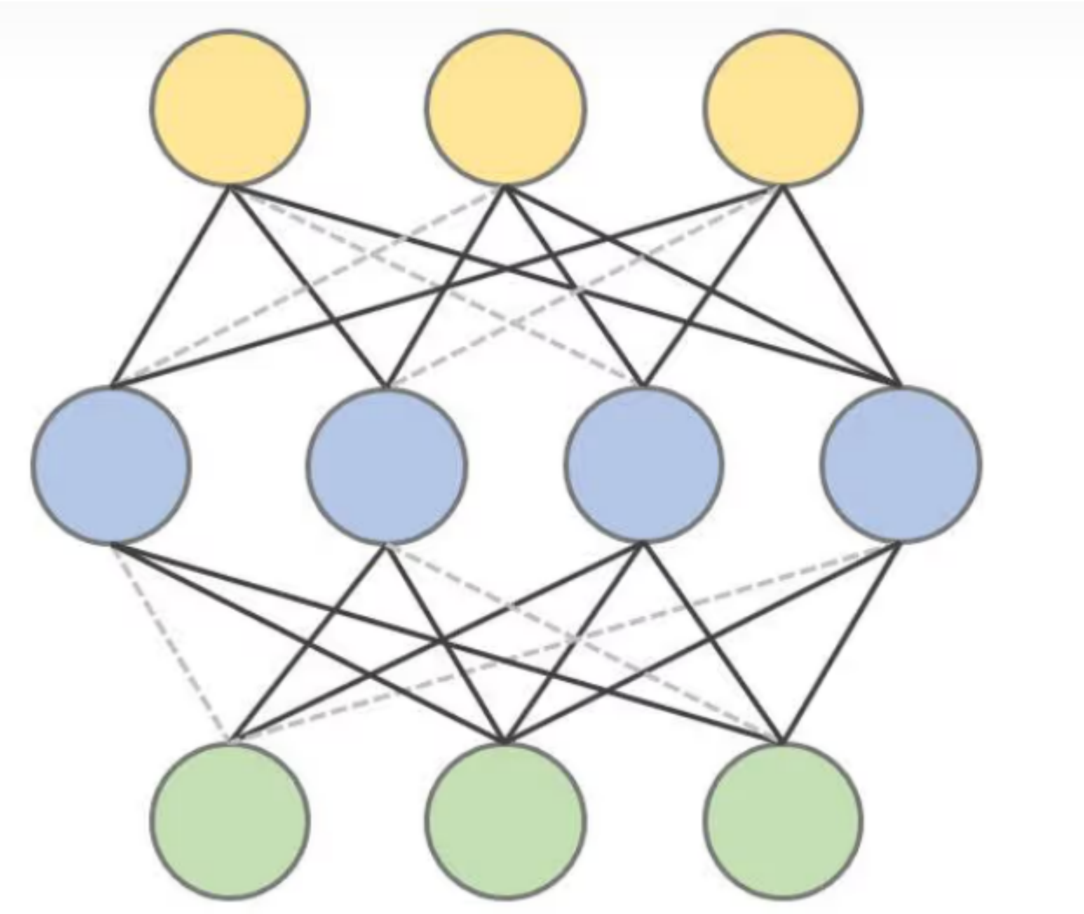
- Quantization
  - LUT-NN, Deep Compression
- Fast convolution
- Low rank approximation
- **Filter pruning**
  - MorphNet, NestDNN, LegoDNN (block-level), AdaptiveNet

Introduction

Two schemes of filter pruning

#1 Unstructural pruning

- Zero out the parameters using a **mask (not real)**
  - torch.nn.utils.prune (PyTorch official)
  - rely on **specific accelerators or software** to reduce memory consumption and computational costs

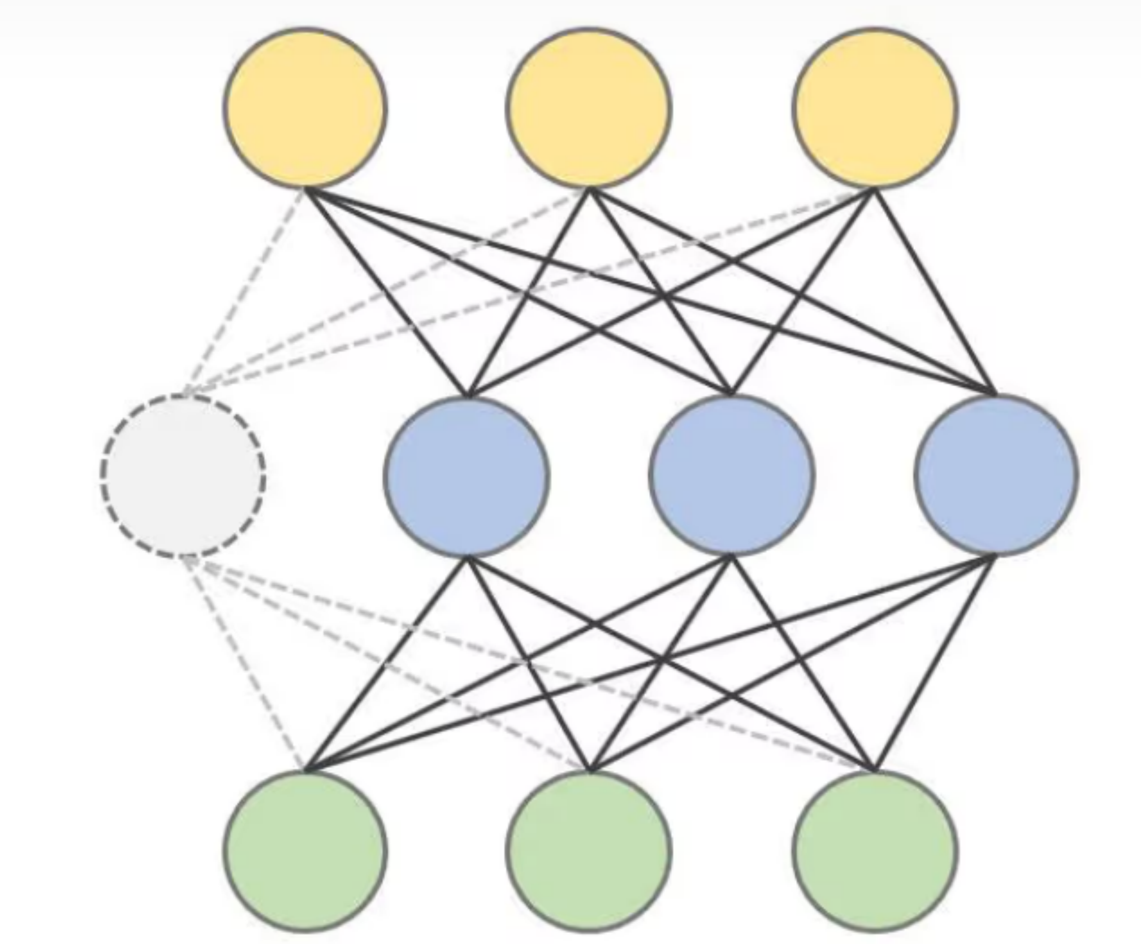


(a) Unstructured pruning

#2 Structural pruning

- **Change** the structure **indeed**
  - Wider domain of application, **spruning schemes, parameter selection**, **layer sparsity and training techniques**

Better ←

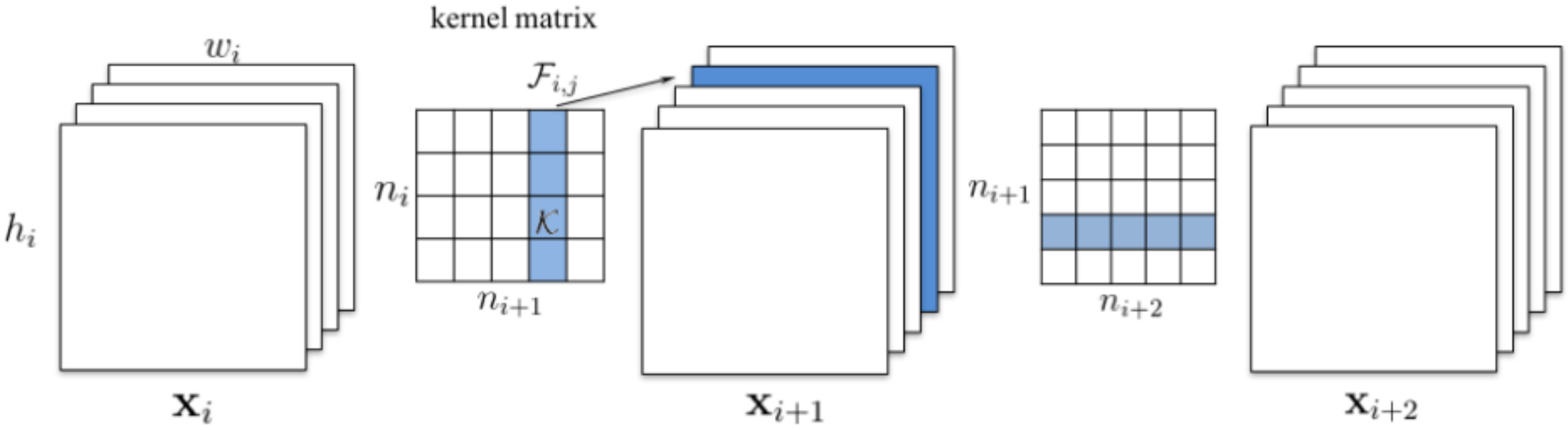


(b) Structured pruning

Introduction

Structural pruning — example

- Pruning **a filter** results in removal of its **corresponding feature map** and **related kernels** in the next layer



Pruning filters for efficient convnets ICLR'17

Challenges for structural filter pruning

Problems 1 : coupled params

- Parameters are **intrinsically coupled** through the **intricate connections.**

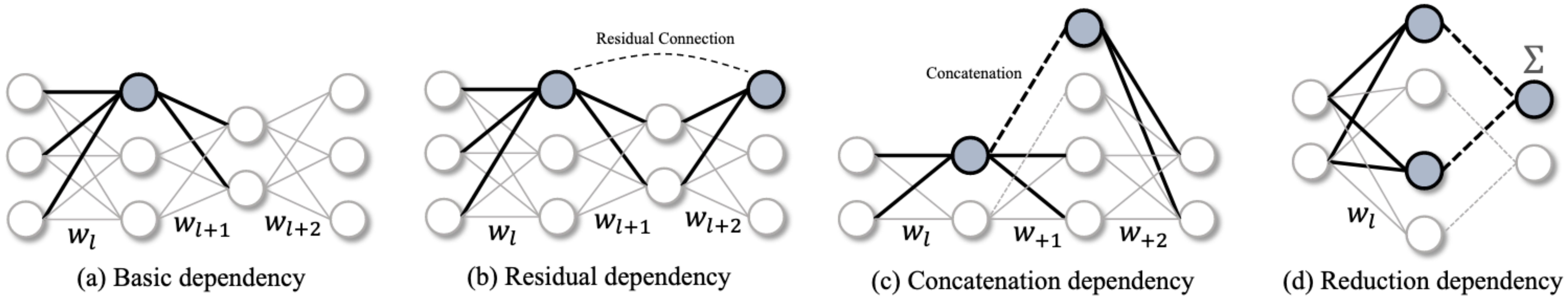
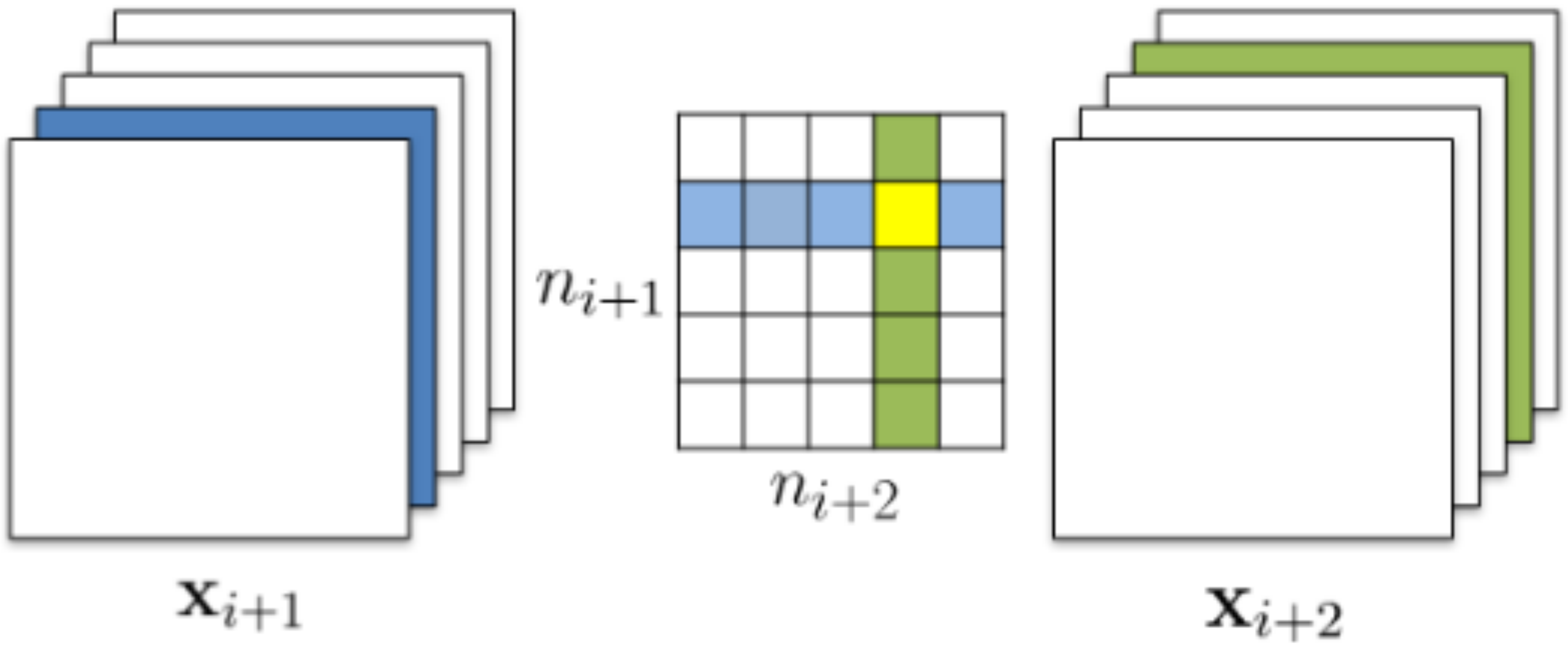


Figure 2. Grouped parameters with inter-dependency in different structures. All highlighted parameters must be pruned simultaneously.

Challenges for structural filter pruning

Problems 1 : coupled params

- Two types of pruning (**input/output channels**)

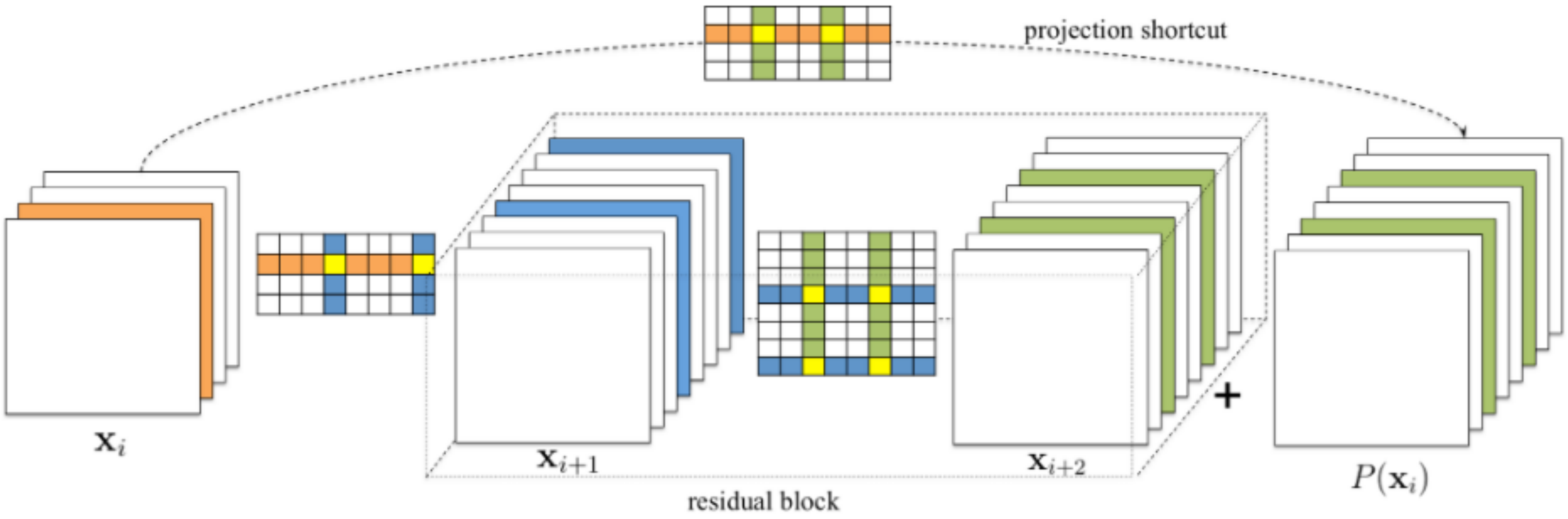


Pruning filters for efficient convnets ICLR'17

Challenges for structural filter pruning

Problems 1 : coupled params

- Example: residual blocks pruing (complex)



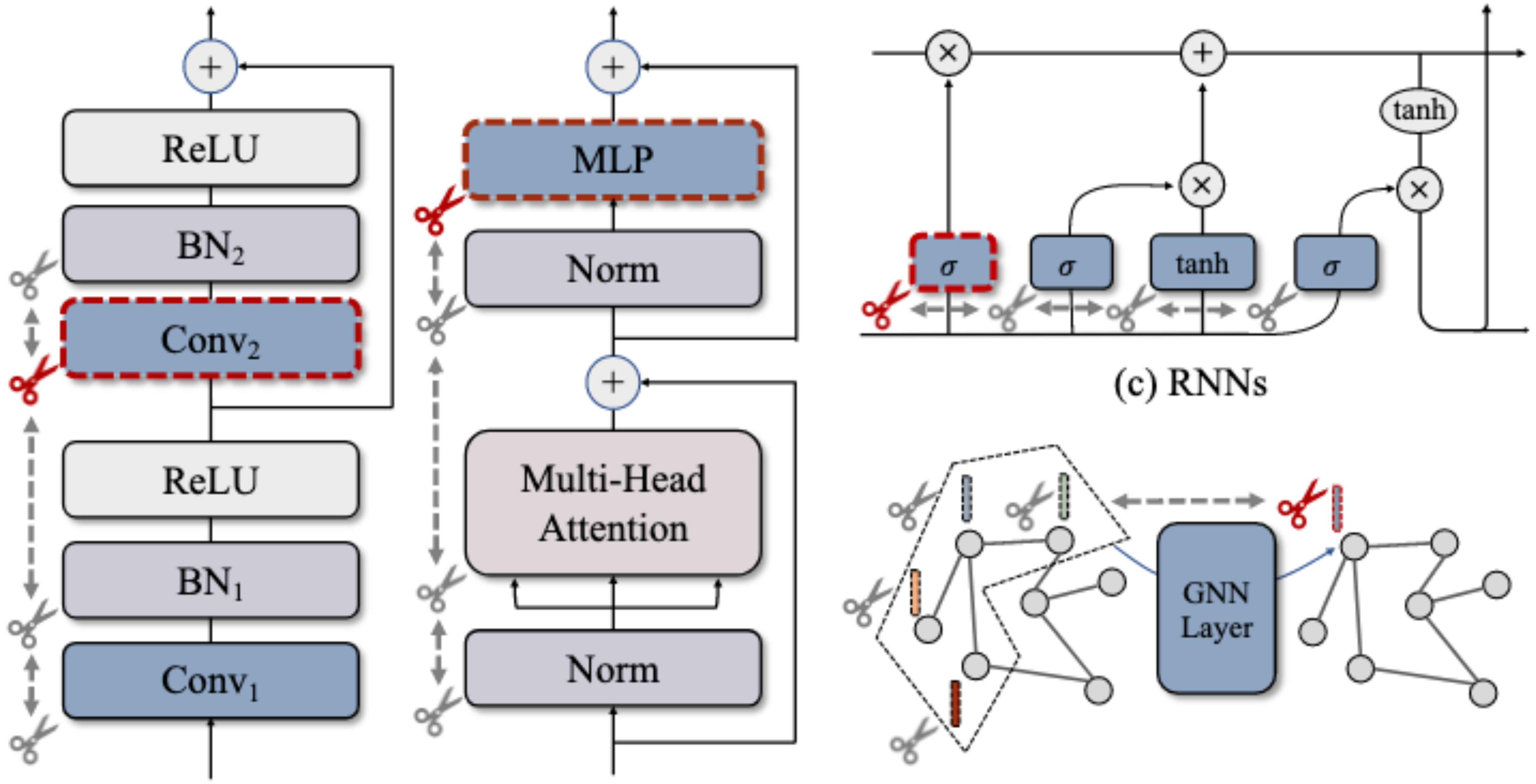
how to group params when dependency is complex ?

Pruning filters for efficient convnets ICLR'17

Challenges for structural filter pruning

Problems 2 : specialized pruning algorithm

- Algorithm implementation is **tightly coupled** with network architecture



(a) CNNs (b) Transformers (c) RNNs (d) GNNs

Need a generic scheme for all structures

Solution I — DepGraph Algorithm

DepGraph

#1 Couple the params

- By utilizing the **local dependency** relationships (**D** matrix) between **adjacent layers**, we **recursively** deduce the desired grouping matrix **G**.

$$g(i) = \{j | G_{ij} = 1\} \quad (1)$$

- G_{ij} is not only determined by the i-th and j-th layers but also affected by those **intermediate layers** between them
- The dependency between adjacent layers **can be recursively inferred.**

D -> G **path exist in D(i->j) <=>** $G_{ij} = 1$

Solution I — DepGraph Algorithm

DepGraph

#2 Modeling the dependency graph (D/G matrix)

- **Finer granularity** in ependency graph **D** (**layer->input/output**)

$$(f_1^-, \underbrace{f_1^+) \leftrightarrow (f_2^-}_{\text{Inter-layer Dep}}, f_2^+) \cdots \leftrightarrow \underbrace{(f_L^-, f_L^+)}_{\text{Intra-layer Dep}} \tag{2}$$

$$D(f_i^-, f_j^+) = \underbrace{\mathbb{1}\left[f_i^- \leftrightarrow f_j^+\right]}_{\text{Inter-layer Dep}} \vee \underbrace{\mathbb{1}\left[i = j \wedge sch(f_i^-) = sch(f_j^+)\right]}_{\text{Intra-layer Dep}} \tag{3}$$

- Two types of dependencies
  - **Inter-layer Dependency**: independent of layer types
  - **Intra-layer Dependency**: related to the type of layers
    - coupled input & output (e.g. add, bn)
    - independent input & output (e.g. conv2d)   w[k,:] (input) & w[:,k] (output)

Solution I — DepGraph Algorithm

DepGraph

#3 Algorithm

Algorithm 1: Dependency Graph

Input: A neural network $\mathcal{F}(x; w)$
Output: DepGraph $D(\mathcal{F}, E)$
$f^- = \{f_1^-, f_2^-, ..., f_L^-\}$ decomposed from the $\mathcal{F}$
$f^+ = \{f_1^+, f_2^+, ..., f_L^+\}$ decomposed from the $\mathcal{F}$
Initialize DepGraph $D = \mathbf{0}_{2L \times 2L}$
for $i = \{0, 1, .., L\}$ **do**
  for $j = \{0, 1, .., L\}$ **do**
    $D(f_i^-, f_j^+) = D(f_j^+, f_i^-) =$
    $\underbrace{\mathbb{1}\left[f_i^- \leftrightarrow f_j^+\right]}_{\text{Inter-layer Dep}} \vee \underbrace{\mathbb{1}\left[i = j \wedge sch(f_i^-) = sch(f_j^+)\right]}_{\text{Intra-layer Dep}}$
return D

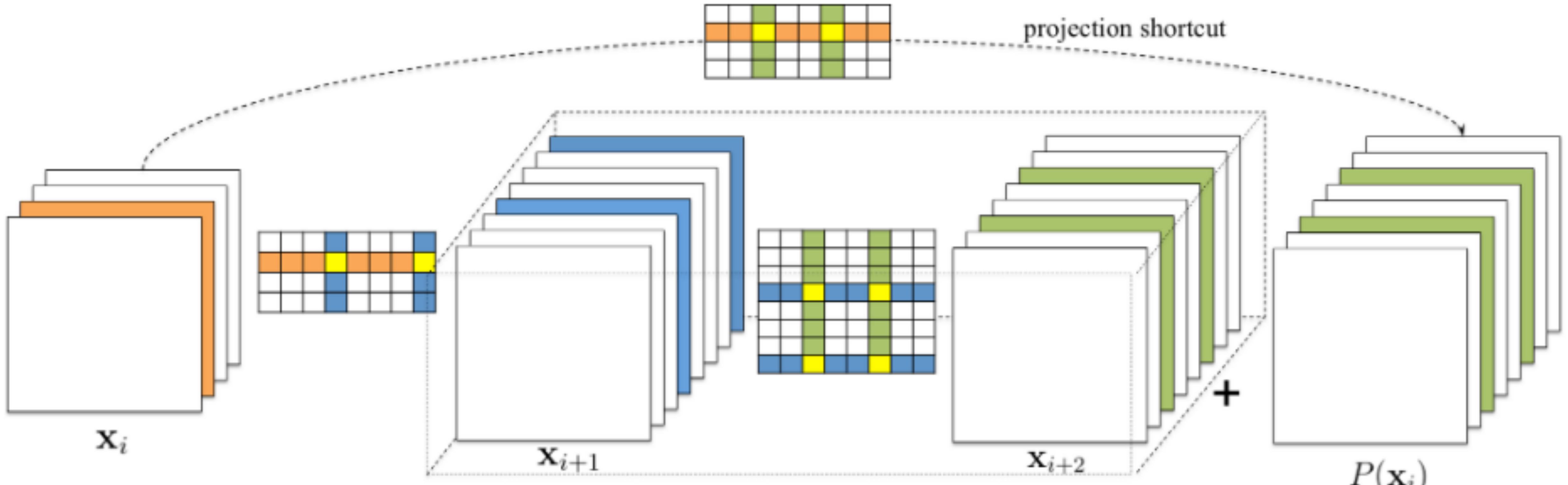
Algorithm 2: Grouping

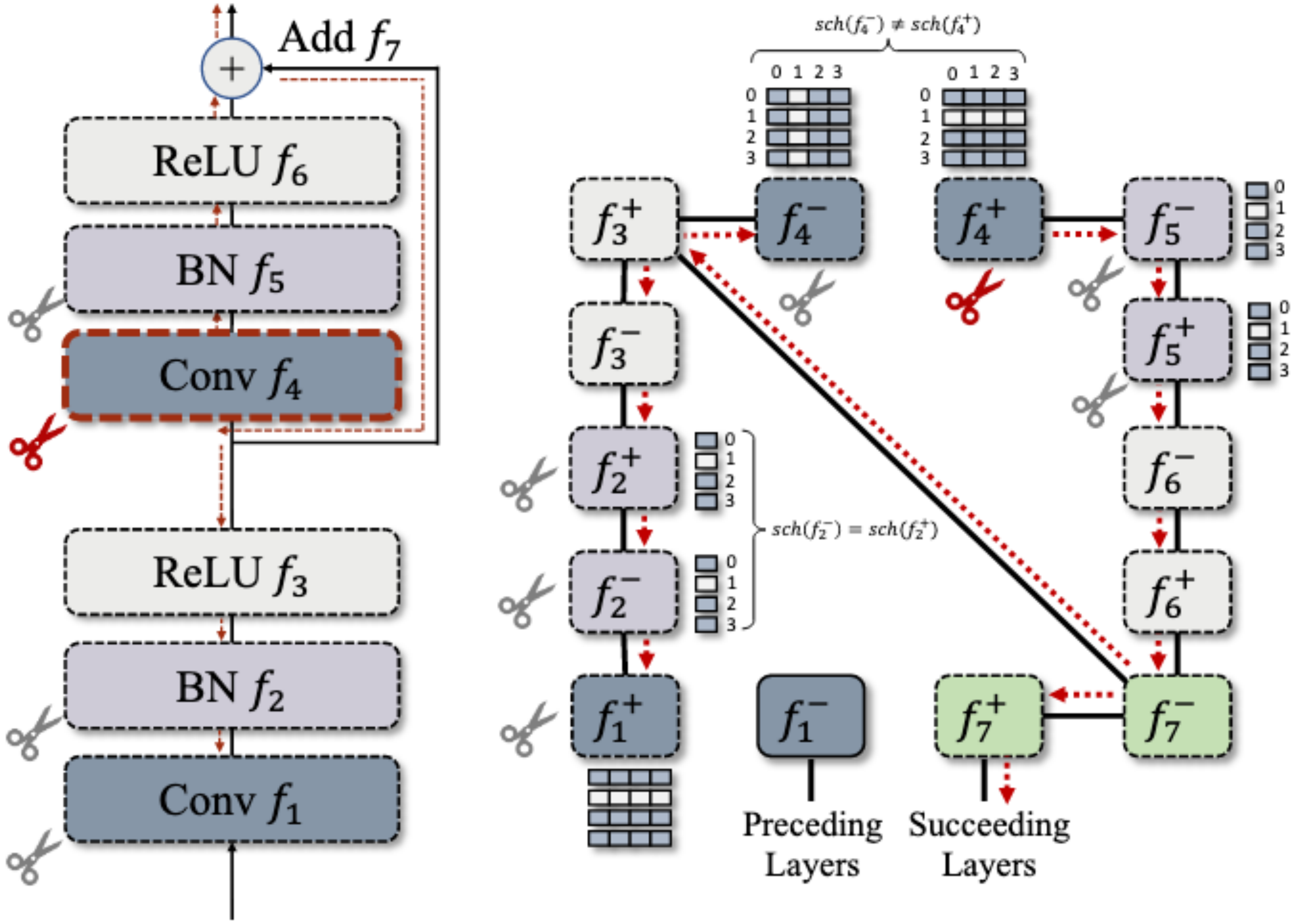
Input: DepGraph $D(\mathcal{F}, E)$
Output: Groups G
$G = \{\}$
for $i = \{1, 2, ..., 2 * \|\mathcal{F}\|\}$ **do**
  $g = \{i\}$
  repeat
    UNSEEN = $\{1, 2, ..., 2 * \|\mathcal{F}\|\} - g$
    $g' = \{j \in \text{UNSEEN} | \exists k \in g, D_{kj} = 1\}$
    $g = g \cup g'$
  until $g' = \emptyset$;
  $G = G \cup \{g\}$
return G

Solution I — DepGraph Algorithm

DepGraph

#4 Example — residual pruning





(a) CNNs

(b) Propagation on Dependency Graph

Figure 3. Layer grouping is achieved via a recursive propagation on DepGraph, starting from the f_4^+. In this example, there is no Intra-layer Dependency between convolutional input f_4^- and output f_4^+ due to the diverged pruning schemes illustrated above.

Solution 2 — Group-level pruning

Prune filter based on dependency graph

#1 Sparsity training

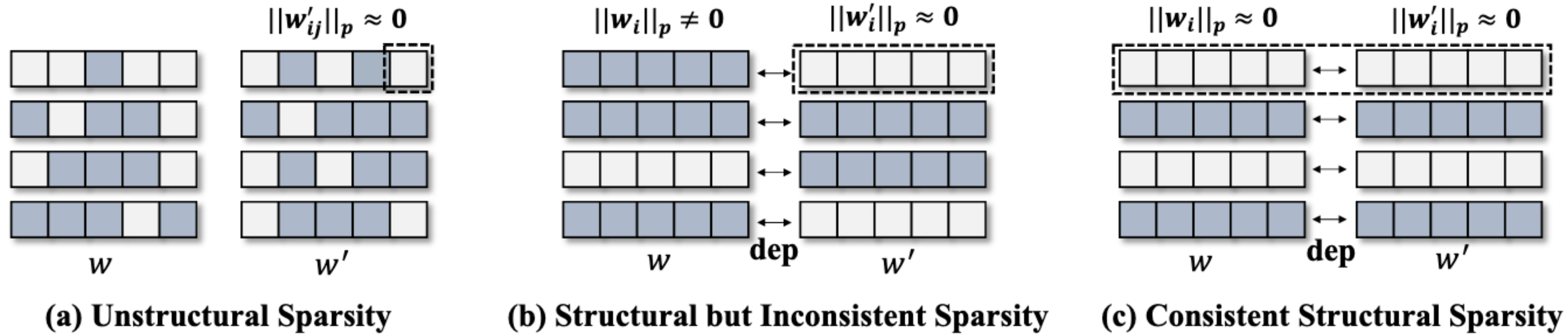


Figure 4. Learning different sparsity schemes to estimate the importance of grouped parameters. Method (a) is used in unstructural pruning which only focuses on the importance of single weight. Method (b) learns structurally sparse layers [35], but ignores coupled weights in other layers. Our method as shown in (c) learns group sparsity which forces all coupled parameters to zero, so that they can be easily distinguished by a simple magnitude method.

$$\mathcal{R}(g,k) = \sum_{k=1}^{K} \gamma_k \cdot I_{g,k} = \sum_{k=1}^{K} \sum_{w \in g} \gamma_k \|w[k]\|_2^2 \quad (4)$$

$$\gamma_k = 2^{\alpha(I_g^{\max} - I_{g,k})/(I_g^{\max} - I_g^{\min})} \quad (5)$$

Consistent sparsity

Solution 2 — Group-level pruning

Prune filter based on dependency graph

#2 Pruning

- Prune based on **relative score**

$$\mathcal{R}(g,k) = \sum_{k=1}^{K} \gamma_k \cdot I_{g,k} = \sum_{k=1}^{K} \sum_{w \in g} \gamma_k \|w[k]\|_2^2 \quad (4) \qquad \gamma_k = 2^{\alpha(I_g^{\max} - I_{g,k})/(I_g^{\max} - I_g^{\min})} \quad (5)$$

$$\hat{I}_{g,k} = N \cdot I_{g,k} / \sum \{\mathrm{TopN}(I_g)\}$$

Experiment — Benchmark

Benchmark on CIFAR & Distribution of Group Sparsity

Model / Data	Method	Base	Pruned	Δ Acc.	Speed Up
	NISP [74]	-	-	-0.03	1.76×
	Geometric [20]	93.59	93.26	-0.33	1.70×
	Polar [78]	93.80	93.83	+0.03	1.88×
	CP [29]	92.80	91.80	-1.00	2.00×
	AMC [19]	92.80	91.90	-0.90	2.00×
	HRank [31]	93.26	92.17	-0.09	2.00×
	SFP [18]	93.59	93.36	-0.23	2.11×
	ResRep [7]	93.71	93.71	+0.00	**2.12**×
ResNet56 CIFAR10	Ours w/o SL	93.53	93.46	-0.07	2.11×
	Ours	93.53	**93.77**	**+0.24**	2.11×
	GBN ([71])	93.10	92.77	-0.33	2.51×
	AFP ([6])	93.93	92.94	-0.99	2.56×
	C-SGD ([4])	93.39	93.44	+0.05	2.55×
	GReg-1 ([58])	93.36	93.18	-0.18	2.55×
	GReg-2 ([58])	93.36	93.36	-0.00	2.55×
	Ours w/o SL	93.53	93.36	-0.17	2.51×
	Ours	93.53	**93.64**	**+0.11**	**2.57**×
	OBD ([56])	73.34	60.70	-12.64	5.73×
	OBD ([56])	73.34	60.66	-12.68	6.09×
	EigenD ([56])	73.34	65.18	-8.16	8.80×
VGG19 CIFAR100	GReg-1 ([58])	74.02	67.55	-6.67	8.84×
	GReg-2 ([58])	74.02	67.75	-6.27	8.84×
	Ours w/o SL	73.50	67.60	-5.44	8.87×
	Ours	73.50	**70.39**	**-3.11**	**8.92**×

Table 1. Pruning results on CIFAR-10 and CIFAR-100.

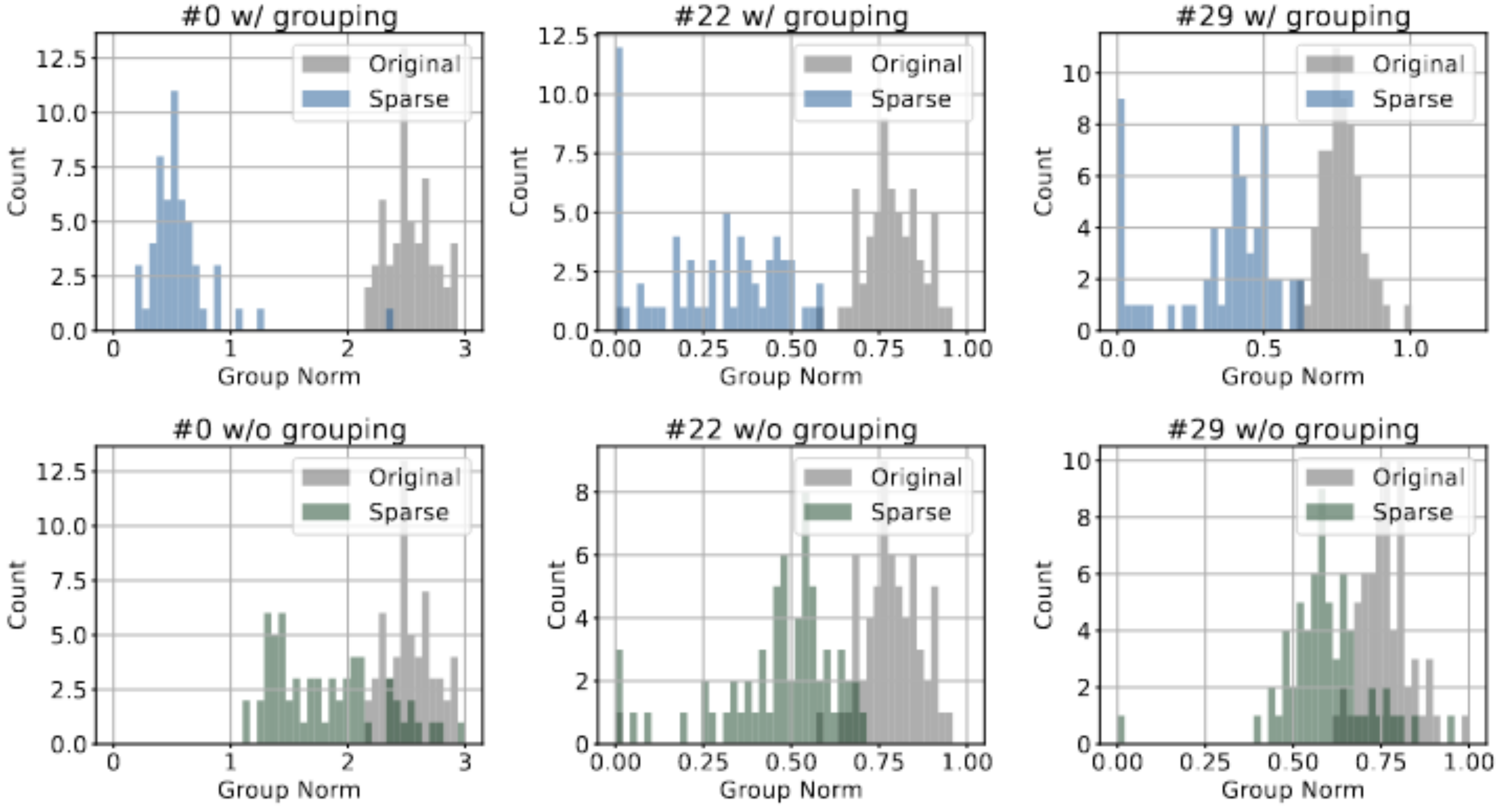


Figure 5. Histogram of group-level sparsity obtained by sparse learning w/ and w/o grouping, which respectively correspond to the strategy (c) and (b) in Figure.4.

Experiment — Ablation Study

- **Grouping** strategy (3 types)
- Learned **Sparsity** (uniform / learned
- **Generalizability** of DepGraph

Architecture	Strategy	Pruned Accuracy with Uniform / Learned Sparsity 1.5×	3.0×	6.0×	12×	Avg.
ResNet-56 (72.58)	Random	71.49 / 72.07	68.52 / 68.16	60.35 / 60.25	53.21 / 48.01	63.39 / 62.15
	No grouping	71.96 / 72.07	67.85 / 67.89	62.64 / 63.18	54.52 / 53.65	64.24 / 64.20
	Conv-only	71.64 / 71.94	68.30 / 69.07	62.44 / 62.63	53.89 / 54.94	64.07 / 64.65
	Full Grouping	71.68 / **72.57**	68.70 / **70.38**	63.72 / **65.33**	55.23 / **55.92**	64.83 / **66.09**
VGG-19 (73.50)	Random	72.63 / 72.77	71.27 / 70.83	68.97 / 69.16	62.45 / 63.42	63.83 / 69.05
	No Grouping	73.83 / 55.13	71.40 / 53.21	69.19 / 50.10	65.12 / †3.87	69.14 / 40.58
	Conv-Only	73.32 / 73.22	71.38 / 71.80	69.66 / 69.85	64.69 / 65.95	69.76 / 70.21
	Full Grouping	73.11 / **74.00**	71.57 / **72.46**	69.72 / **70.38**	65.74 / **66.20**	70.03 / **70.58**
DenseNet-121 (78.73)	Random	79.04 / 79.43	77.86 / 78.62	75.47 / 74.52	69.26 / 69.64	75.41 / 75.80
	No Grouping	79.31 / 78.91	78.08 / 78.62	**78.62** / 68.57	72.93 / 57.17	77.24 / 70.82
	Conv-Only	79.18 / **79.74**	77.98 / 78.85	76.61 / 77.22	73.30 / 73.95	76.77 / 77.44
	Full Grouping	79.34 / 79.74	77.97 / **79.19**	77.08 / 77.78	**74.77** / 75.29	77.29 / **77.77**
MobileNetv2 (70.80)	Random	70.90 / 70.69	67.75 / 67.54	61.32 / 62.26	53.41 / 53.97	63.35 / 63.62
	No Grouping	71.16 / 71.28	69.93 / 68.59	66.76 / 37.38	60.28 / 28.24	67.03 / 51.37
	Conv-Only	71.22 / 71.51	70.33 / 70.15	66.16 / 66.49	61.35 / 63.24	67.27 / 67.85
	Full Grouping	71.11 / **71.67**	70.06 / **70.81**	66.48 / **68.02**	60.32 / **63.37**	66.99 / **68.67**
GoogleNet (77.56)	Random	77.52 / 77.72	76.47 / 76.15	74.92 / 74.19	69.37 / 69.69	74.57 / 74.44
	No Grouping	77.44 / 77.23	76.84 / 74.95	75.60 / 63.78	71.92 / 63.72	75.45 / 69.92
	Conv-Only	77.33 / **77.62**	76.68 / 76.92	75.66 / 74.98	71.90 / 71.87	75.49 / 75.35
	Full Grouping	77.91 / 77.76	76.90 / **77.00**	75.42 / **75.44**	71.98 / **72.88**	75.53 / **75.57**

Table 2. Ablation study on CIFAR-100 for different grouping strategies and sparsity configurations. The proposed strategy, full grouping, takes all parameterized layers into account during sparse training, while other strategies only leverage partial layers. Accuracy (%) of pruned models with uniform layer sparsity or learned layer sparsity is reported. †:In some cases, our method over-prunes some dimension to 1, which severely damages the final accuracy.

Generic
rather than
SOTA solution

https://github.com/horseee/LLM-Pruner NerualPS'23

Pruning on DepGraph & PyTorch

Comparison and usage of two tools

#1 Platforms & Tools

- **Pytorch** ： torch.nn.utils.prune
- **DepGraph** : Torch-pruning

Pruning on DepGraph & PyTorch

Comparison and usage of two tools

#2 Exp on torch.nn.utils.prune

- **Model :** LeNet-5

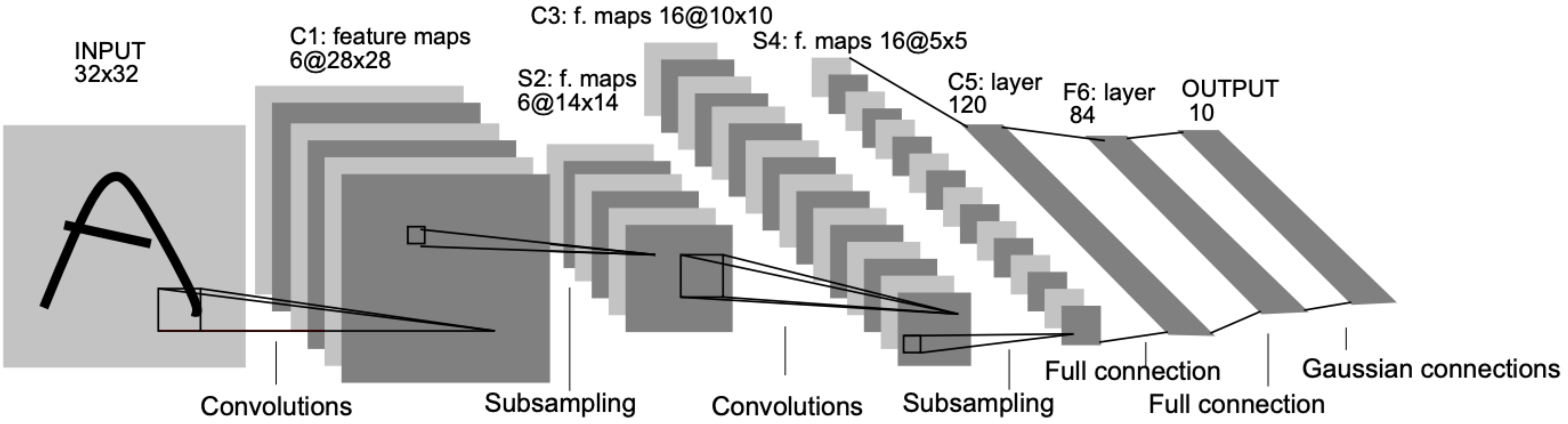


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Pruning on DepGraph & PyTorch

PyTorch

Comparison and usage of two tools

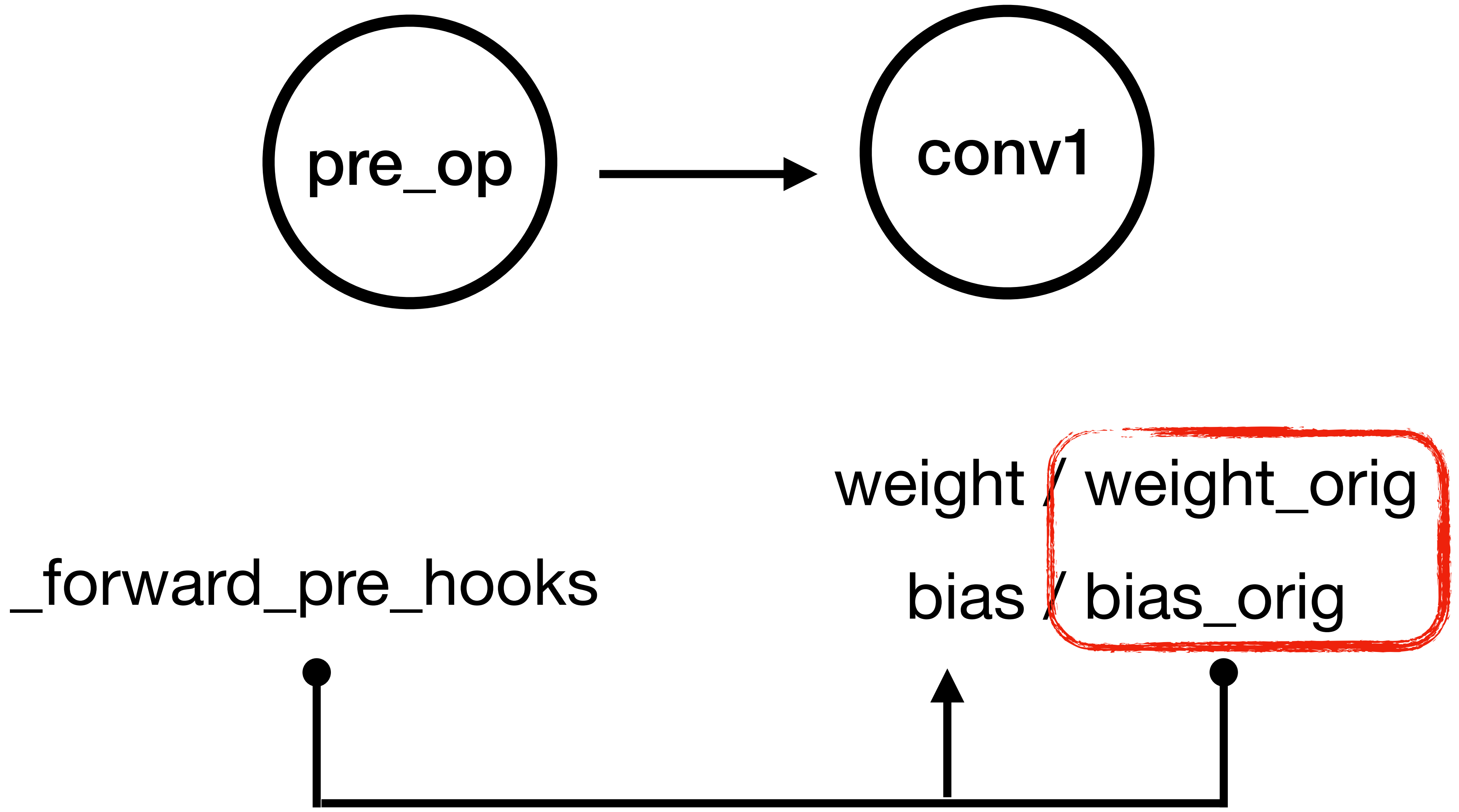
#2 Exp on torch.nn.utils.prune

- **Prunner :**
  - random_unstructured
  - l1_unstructured
  - ln_unstructured
- **Param :**
  - dim = 0 (filter)

Buffters (mask 0/1)

weight_mask

bias_mask



Layer pruning

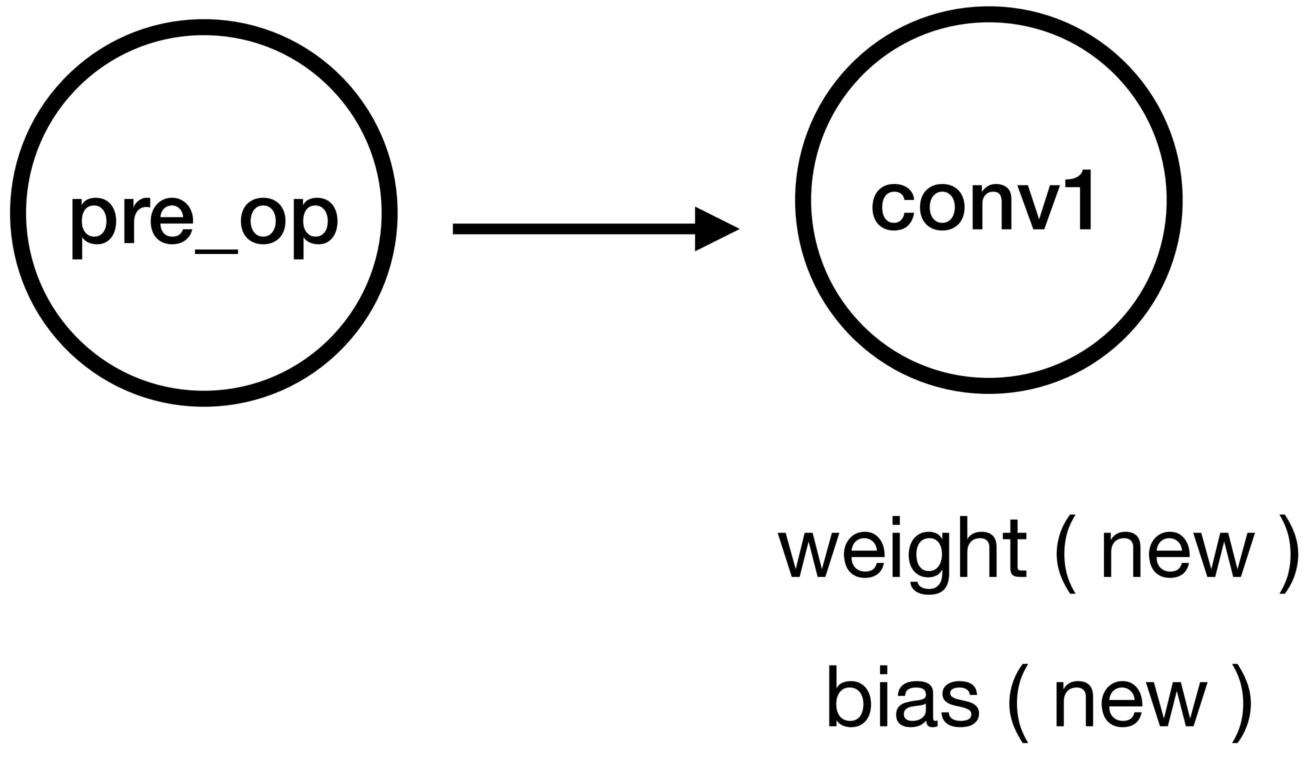
Pruning on DepGraph & PyTorch

PyTorch

Comparison and usage of two tools

#2 Exp on torch.nn.utils.prune

- **Remove pruning re-parametrization**



如剪 — fake structural pruning

Pruning on DepGraph & PyTorch

Comparison and usage of two tools

#3 Exp on Torch-Pruning

- **Model : ResNet18**
- **Low-level** pruning (specific filters)
- **High-level** pruning (prune based on groups)

Conclusion

- Introduced a **generic scheme**, termed as **Dependency Graph**, to explicitly account for such dependencies and execute the pruning of arbitrary architecture in a **fully automatic** manner
- There is room for improvement in **sparse training and importance evaluation.**
- **How to combine with resource constraints ?**

https://github.com/VainF/Torch-Pruning

Thanks

2023-10-24

Presented by Guangtong Li