JUNE 18-22, 2023
CVPR
VANCOUVER, CANADA

DepGraph: Towards Any Structural Pruning

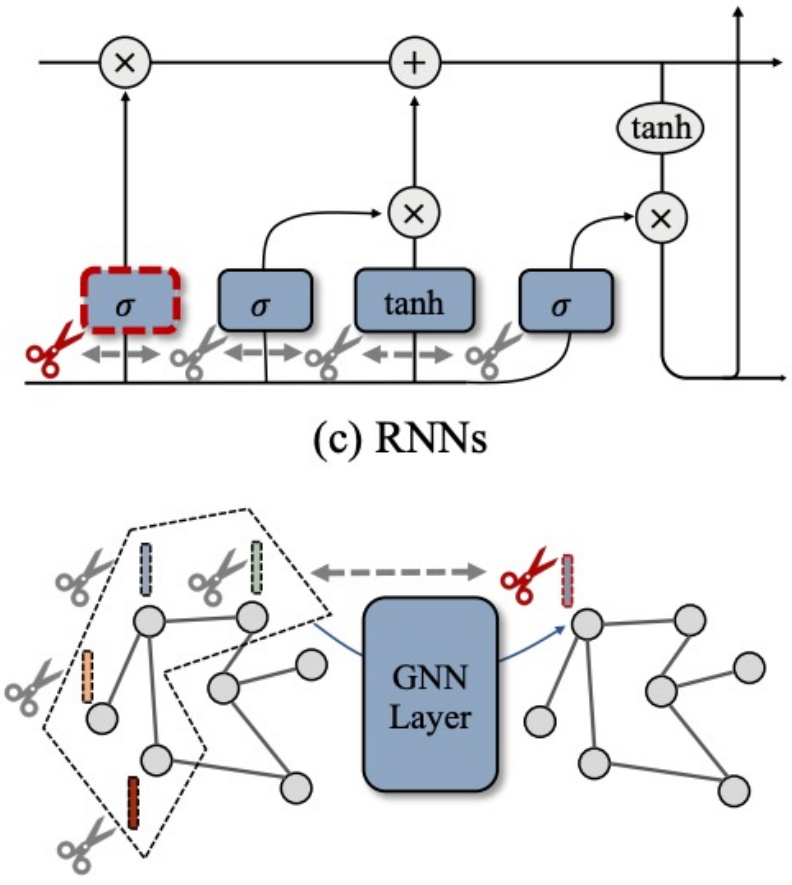
Gongfan Fang[1] Xinyin Ma[1] Mingli Song[3] Michael Bi Mi[2] Xinchao Wang[1]

[1]National University of Singapore [2]Huawei Technologies Ltd. [3]Zhejiang University

Poster ID: WED-PM-356 Paper ID: 1159

Background

Any Structural Pruning

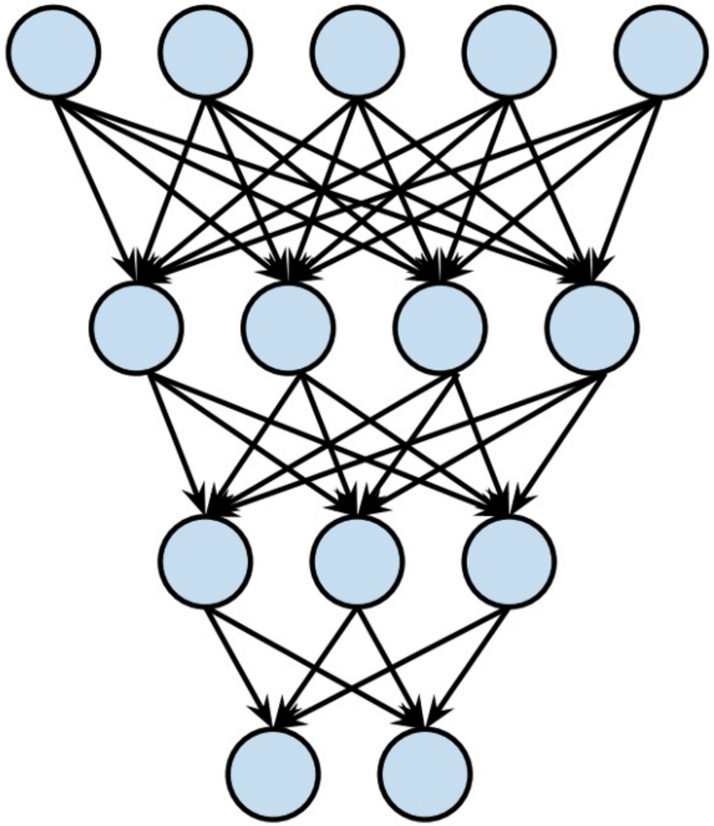


(a) CNNs (b) Transformers (c) RNNs (d) GNNs

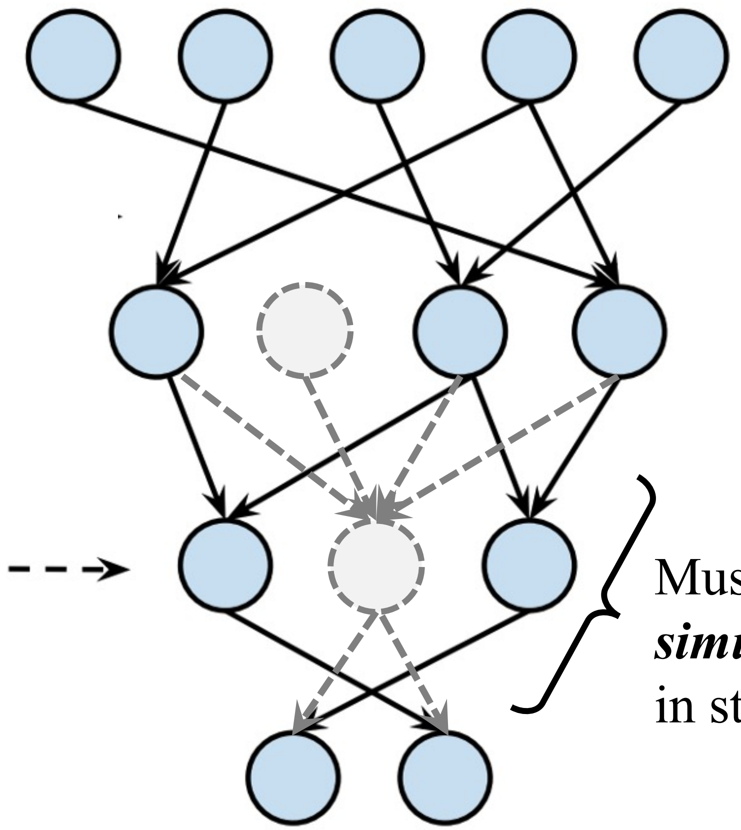
Background

Grouping: The challenge in Any Structural Pruning

Network → **Grouping** → Importance Evaluation → Remove the least important neuron → Fine-tuning



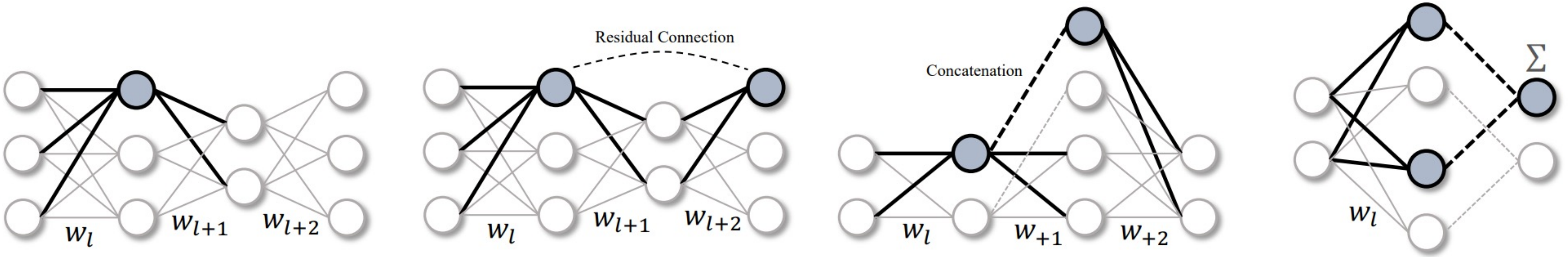
Pruning structures
(Structural)



Must be pruned
simultaneously
in structural pruning

Background

Challenge: The grouping patterns vary widely across different models



(a) Basic dependency (b) Residual dependency (c) Concatenation dependency (d) Reduction dependency

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

These dependencies make it difficult to prune different networks

Background

Introducing DepGraph

DepGraph: an automatic pipeline for structural pruning

1. Building DepGraph

2. Grouping

3. Pruning

```python
# 1. build dependency graph for resnet18
DG = tp.DependencyGraph().build_dependency(model, example_inputs=torch.randn(1,3,224,224))

# 2. grouping parameters
pruning_group = DG.get_pruning_group( model.conv1, tp.prune_conv_out_channels, idxs=[2, 6, 9] )

# 3. remove parameters
pruning_group.exec()
```

v.s.

Manual Pruning (ICCV):

```python
for layer_id in range(len(old_modules)):
    m0 = old_modules[layer_id]
    m1 = new_modules[layer_id]
    if isinstance(m0, nn.BatchNorm2d):
        idx1 = np.squeeze(np.argwhere(np.asarray(end_mask.cpu().numpy())))
        if idx1.size == 1:
            idx1 = np.resize(idx1,(1,))

        if isinstance(old_modules[layer_id + 1], channel_selection):
            # If the next layer is the channel selection layer, then the current batchnorm 2d layer won't be pruned.
            m1.weight.data = m0.weight.data.clone()
            m1.bias.data = m0.bias.data.clone()
            m1.running_mean = m0.running_mean.clone()
            m1.running_var = m0.running_var.clone()

            # We need to set the channel selection layer.
            m2 = new_modules[layer_id + 1]
            m2.indexes.data.zero_()
            m2.indexes.data[idx1.tolist()] = 1.0

            layer_id_in_cfg += 1
            start_mask = end_mask.clone()
            if layer_id_in_cfg < len(cfg_mask):
                end_mask = cfg_mask[layer_id_in_cfg]
        else:
            m1.weight.data = m0.weight.data[idx1.tolist()].clone()
            m1.bias.data = m0.bias.data[idx1.tolist()].clone()
            m1.running_mean = m0.running_mean[idx1.tolist()].clone()
            m1.running_var = m0.running_var[idx1.tolist()].clone()
            layer_id_in_cfg += 1
            start_mask = end_mask.clone()
            if layer_id_in_cfg < len(cfg_mask):  # do not change in Final FC
                end_mask = cfg_mask[layer_id_in_cfg]
    elif isinstance(m0, nn.Conv2d):
        if conv_count == 0:
            m1.weight.data = m0.weight.data.clone()
            conv_count += 1
            continue
        if isinstance(old_modules[layer_id-1], channel_selection) or isinstance(old_modules[layer_id-1], nn.BatchNorm2d):
            # This convers the convolutions in the residual block.
            # The convolutions are either after the channel selection layer or after the batch normalization layer.
            conv_count += 1
            idx0 = np.squeeze(np.argwhere(np.asarray(start_mask.cpu().numpy())))
            idx1 = np.squeeze(np.argwhere(np.asarray(end_mask.cpu().numpy())))
            print('In shape: {:d}, Out shape {:d}.'.format(idx0.size, idx1.size))
            if idx0.size == 1:
                idx0 = np.resize(idx0, (1,))
            if idx1.size == 1:
                idx1 = np.resize(idx1, (1,))
            w1 = m0.weight.data[:, idx0.tolist(), :, :].clone()

            # If the current convolution is not the last convolution in the residual block, then we can change the
            # number of output channels. Currently we use `conv_count` to detect whether it is such convolution.
            if conv_count % 3 != 1:
                w1 = w1[idx1.tolist(), :, :, :].clone()
            m1.weight.data = w1.clone()
            continue

        # We need to consider the case where there are downsampling convolutions.
        # For these convolutions, we just copy the weights.
        m1.weight.data = m0.weight.data.clone()
    elif isinstance(m0, nn.Linear):
        idx0 = np.squeeze(np.argwhere(np.asarray(start_mask.cpu().numpy())))
        if idx0.size == 1:
            idx0 = np.resize(idx0, (1,))

        m1.weight.data = m0.weight.data[:, idx0].clone()
        m1.bias.data = m0.bias.data.clone()
```

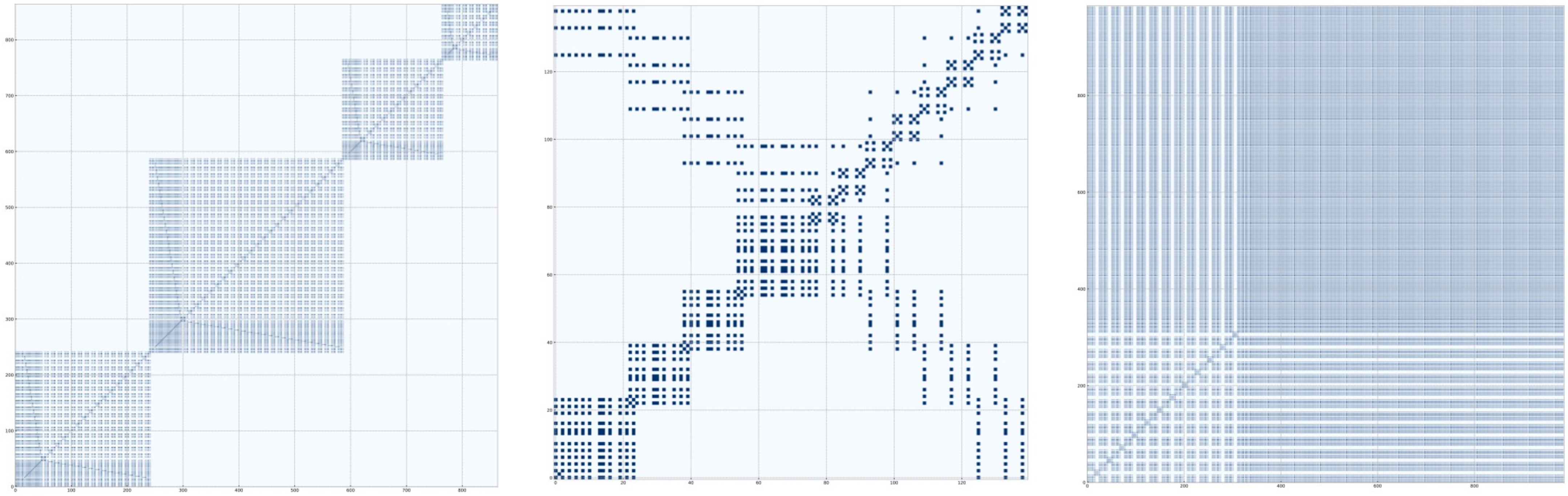
Methodology

❑ Formalizing the grouping step:

Finding a grouping graph G so that

$$G_{ij} = \begin{cases} 1, & \text{if layer i, j in the same group} \\ 0, & \text{otherwise} \end{cases}$$

Issue: no empirical and explicit rule for building this graph

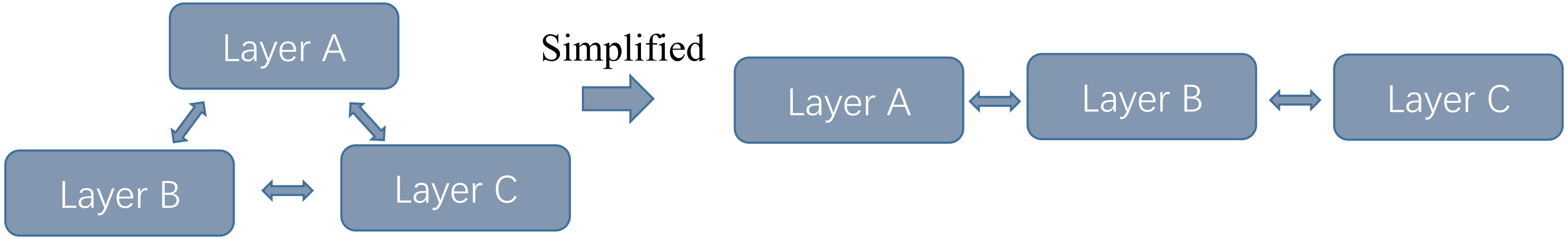


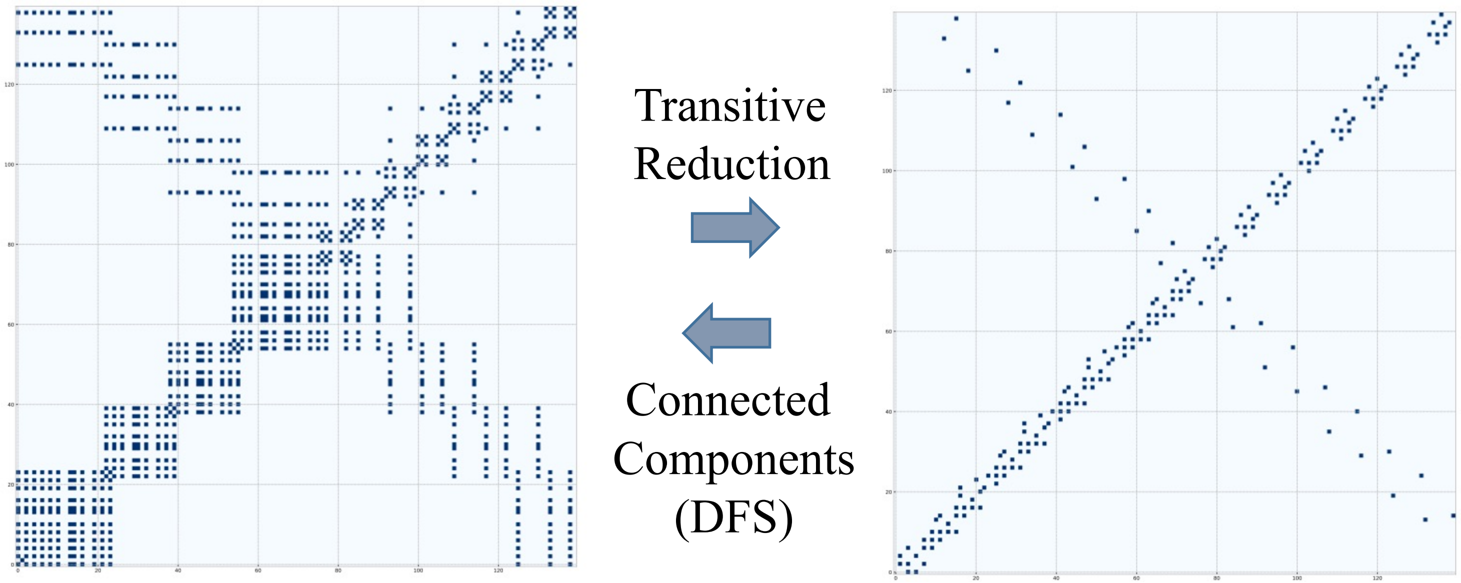
Grouping graph of DenseNet-121 | Grouping graph of ResNet-50 | Grouping graph of ViT-Base

Methodology

❑ Our solution: Dependency Graph

Leveraging the transitive property of dependency for simplification (**transitive reduction**):



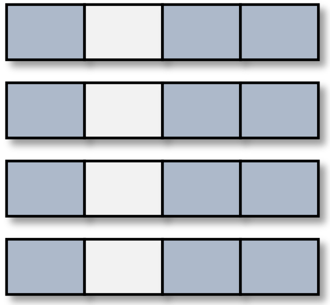


Grouping graph of ResNet-50 Dependency graph of connected layers

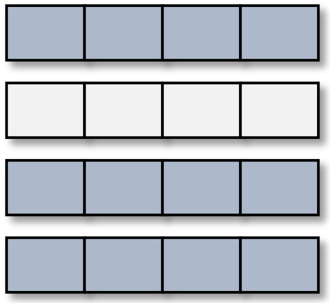
We only need to handle local relations

Methodology

❑ Different pruning schemes for inputs & outputs



Remove input channels



Remove output channels

❑ Network Decomposition

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

❑ Modeling local dependency

Connectivity: adjacent layers

Self-dependency: coupled inputs & outputs (e.g. ReLU, BN)

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

Methodology

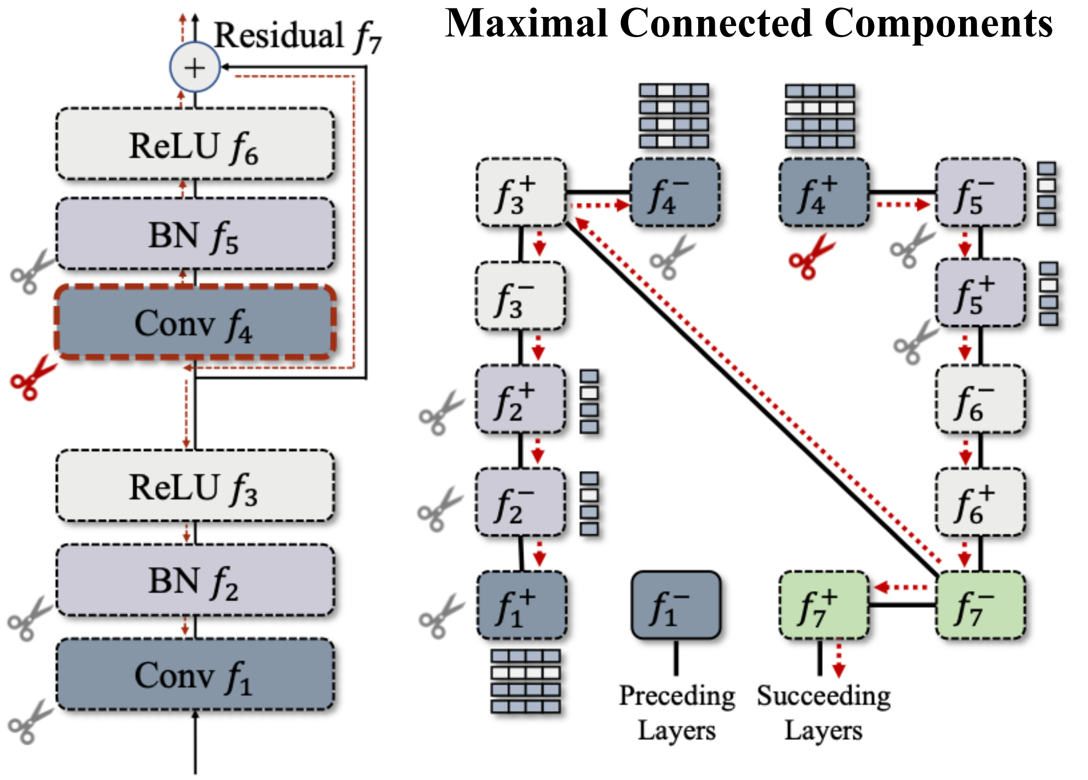
❑ Algorithms

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

Algorithm 2: Grouping

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

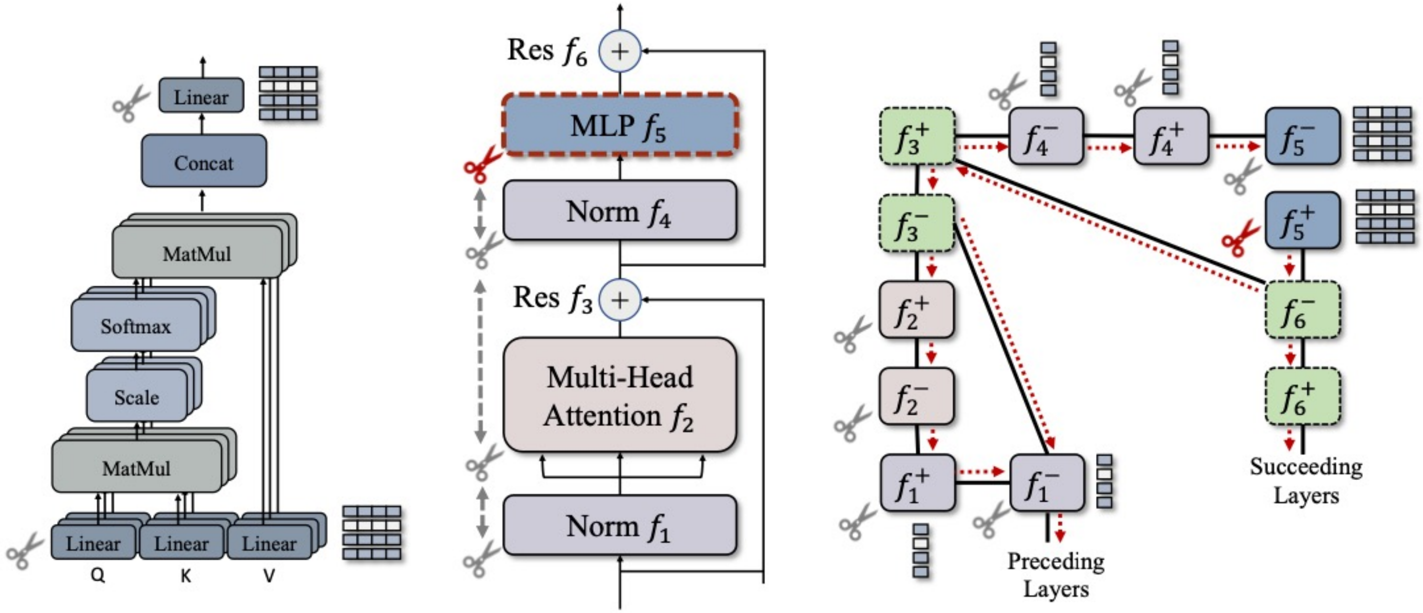


(a) CNNs (b) Propagation on Dependency Graph

Figure 3. Layer grouping is achieved via a recursive propagation on DepGraph, starting from the red scissor. DepGraph not only collects coupled layers, but also show coupling relation between layers as well as pruning schemes for different layers.

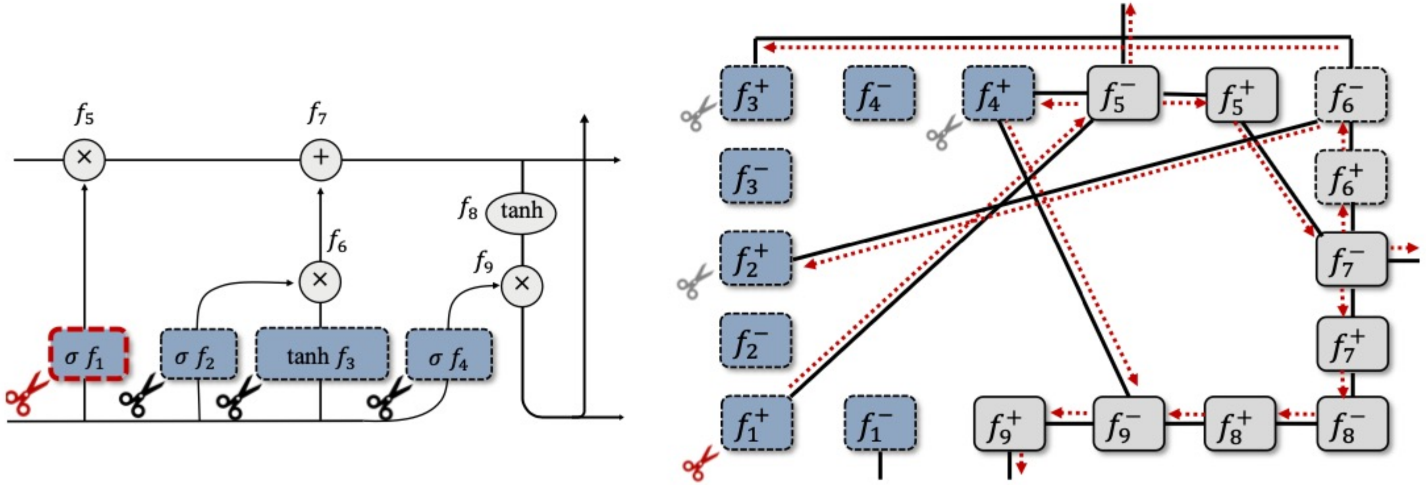
Methodology

- **Generalizing DepGraph to**
  - Vision Transformers
  - RNNs
  - YOLO v7 / YOLO v8
  - LLMs
  - HuggingFace Diffusers
  - TorchVision Models (95%)
  - …



(a) MHA Pruning (b) Transformer Block (b) Propagation on Dependency Graph

Figure 2. Pruning of Vision Transformers



(a) LSTM (b) Propagation on Dependency Graph

Figure 3. Pruning of LSTM. The input connections are eliminated for simplicity as they are not affected in this case.

Methodology

❑ Group Importance v.s. Layer Importance

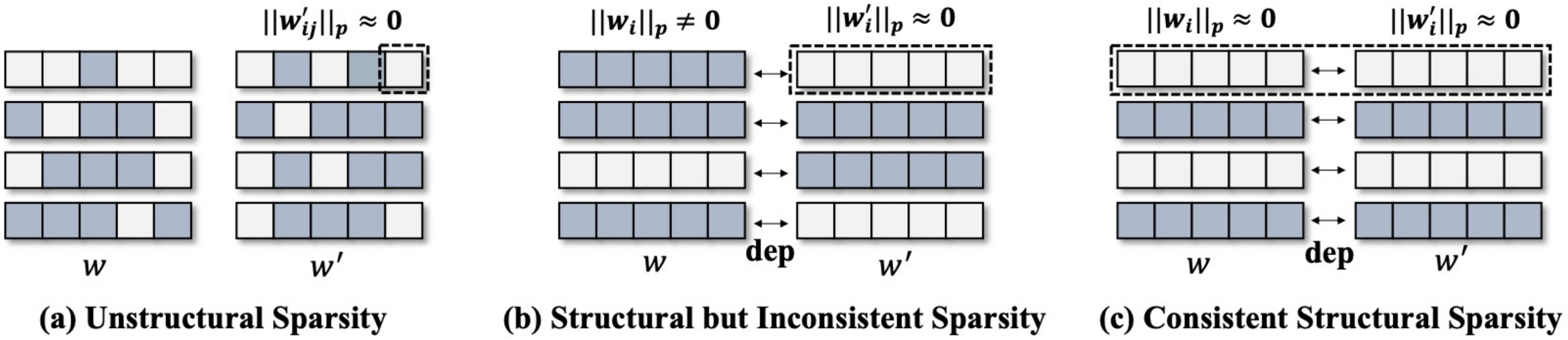


Figure 4. Learning different sparsity schemes to estimate the importance of grouped parameters. Method (a) is used in unstructural pruning which only focuses on the importance of single weight. Method (b) learns structurally sparse layers [31], but ignores coupled weights in other layers. Our method 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.

$$\mathcal{R}(g, k) = \sum_{w \in w_g[k]} \gamma_k \|w\|_2^2$$ Assigning different weights γ to different groups

Experiments

❑ Pruning results on CIFAR & ImageNet

Model / Data	Method	Base	Pruned	Δ Acc.	Speed Up
ResNet56 CIFAR10	NISP [64]	-	-	-0.03	1.76×
	Geometric [19]	93.59	93.26	-0.33	1.70×
	Polar [68]	93.80	93.83	+0.03	1.88×
	CP [27]	92.80	91.80	-1.00	2.00×
	AMC [18]	92.80	91.90	-0.90	2.00×
	HRank [29]	93.26	92.17	-0.09	2.00×
	SFP [17]	93.59	93.36	-0.23	2.11×
	ResRep [6]	93.71	93.71	+0.00	**2.12**×
	Ours w/o SL	93.53	93.46	-0.07	2.11×
	Ours	93.53	**93.77**	**+0.24**	2.11×
	GBN ([61])	93.10	92.77	-0.33	2.51×
	AFP ([5])	93.93	92.94	-0.99	2.56×
	C-SGD ([3])	93.39	93.44	+0.05	2.55×
	GReg-1 ([52])	93.36	93.18	-0.18	2.55×
	GReg-2 ([52])	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**×
VGG19 CIFAR100	OBD ([51])	73.34	60.70	-12.64	5.73×
	OBD ([51])	73.34	60.66	-12.68	6.09×
	EigenD ([51])	73.34	65.18	-8.16	8.80×
	GReg-1 ([52])	74.02	67.55	-6.67	8.84×
	GReg-2 ([52])	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.

Arch.	Method	Base	Pruned	Δ Acc.	MACs
ResNet-50	ResNet-50	76.15	-	-	4.13
	ThiNet [35]	72.88	72.04	-0.84	2.44
	SSS [23]	76.12	74.18	-1.94	2.82
	SFP [17]	76.15	74.61	-1.54	2.40
	AutoSlim [63]	76.10	75.60	-0.50	2.00
	FPGM [19]	76.15	75.50	-0.65	2.38
	Taylor [38]	76.18	74.50	-1.68	2.25
	Slimable [62]	76.10	74.90	-1.20	2.30
	CCP [41]	76.15	75.50	-0.65	2.11
	AOFP-C1 [4]	75.34	75.63	+0.29	2.58
	TAS [8]	77.46	76.20	-1.26	2.31
	GFP [30]	76.79	76.42	-0.37	2.04
	GReg-2 [52]	76.13	75.36	-0.77	2.77
	Ours	76.15	75.83	-0.32	1.99
DenseNet-121	DenseNet-121	74.44	-	-	2.86
	PSP-1.38G [45]	74.35	74.05	-0.30	1.38
	PSP-0.58G [45]	74.35	70.34	-4.01	0.58
	Ours-1.38G	74.44	73.98	-0.46	1.37
	Ours-0.58G	74.44	70.13	-4.31	0.57
Mob-v2	Mob-v2	71.87	-	-	0.33
	NetAdapt [58]	-	70.00	-	0.24
	Meta [32]	74.70	68.20	-6.50	0.14
	GFP [30]	75.74	69.16	-6.58	0.15
	Ours	71.87	68.46	-3.41	0.15
NeXt-50	ResNeXt-50	77.62	-	-	4.27
	SSS [23]	77.57	74.98	-2.59	2.43
	GFP [30]	77.97	77.53	-0.44	2.11
	Ours	77.62	76.48	-1.14	2.09
ViT-B/16	VIT-B/16	81.07	-	-	17.6
	CP-ViT [47]	77.91	77.36	-0.55	11.7
	Ours+EMA	81.07	79.58	-1.39	10.4
	Ours	81.07	79.17	-1.90	10.4

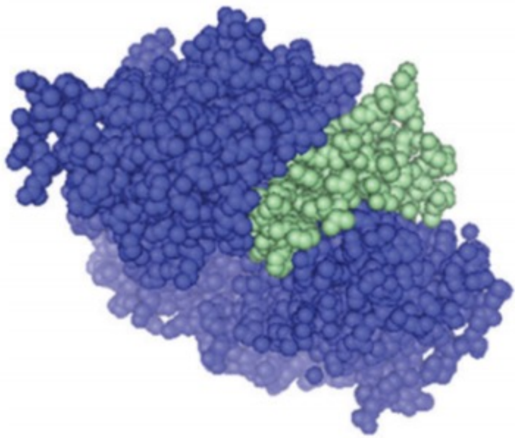
Table 3. Pruning results on ImageNet.

Experiments

❑ Text, 3D point cloud, graph and more

Most users think their computer is safe from adware and spyware--but they're wrong. A survey conducted by Internet service provider America Online found that 20 percent of home computers were infected by

Protein-protein interaction (PPI)



Arch. & Data	Method	Base	Pruned	Δ	Speedup
LSTM (AGNews)	DepGraph+Random	92.10	91.23	-0.87	16.28×
	DepGraph+CP [27]	92.10	91.50	-0.60	16.28×
	Ours w/o SL	92.10	91.53	-0.57	16.28×
	Ours	92.10	**91.75**	**-0.35**	16.28×
DGCNN (ModelNet40)	DepGraph+Random	92.10	91.05	-1.05	10.05×
	DepGraph+CP [27]	92.10	91.00	-1.10	10.05×
	DepGraph+Slim [31]	92.10	91.74	-0.36	10.35×
	Ours w/o SL	92.10	91.86	-0.24	11.46×
	Ours	92.10	**92.02**	**-0.08**	11.98×
GAT (PPI)	DepGraph+Random	0.986	0.951	-0.035	8.05×
	DepGraph+CP [27]	0.986	0.957	-0.029	8.05×
	Ours w/o SL	0.986	0.953	-0.033	8.26×
	Ours	0.986	**0.961**	**-0.025**	8.43×

Table 4. Pruning neural networks for non-image data, including AGNews (text), ModelNet (3D Point Cloud) and PPI (Graph). We report the classification accuracy (%) of pruned model for AG-News and ModelNet and micro-F1 score for PPI.

Experiments

DepGraph & the derived Grouping Graph

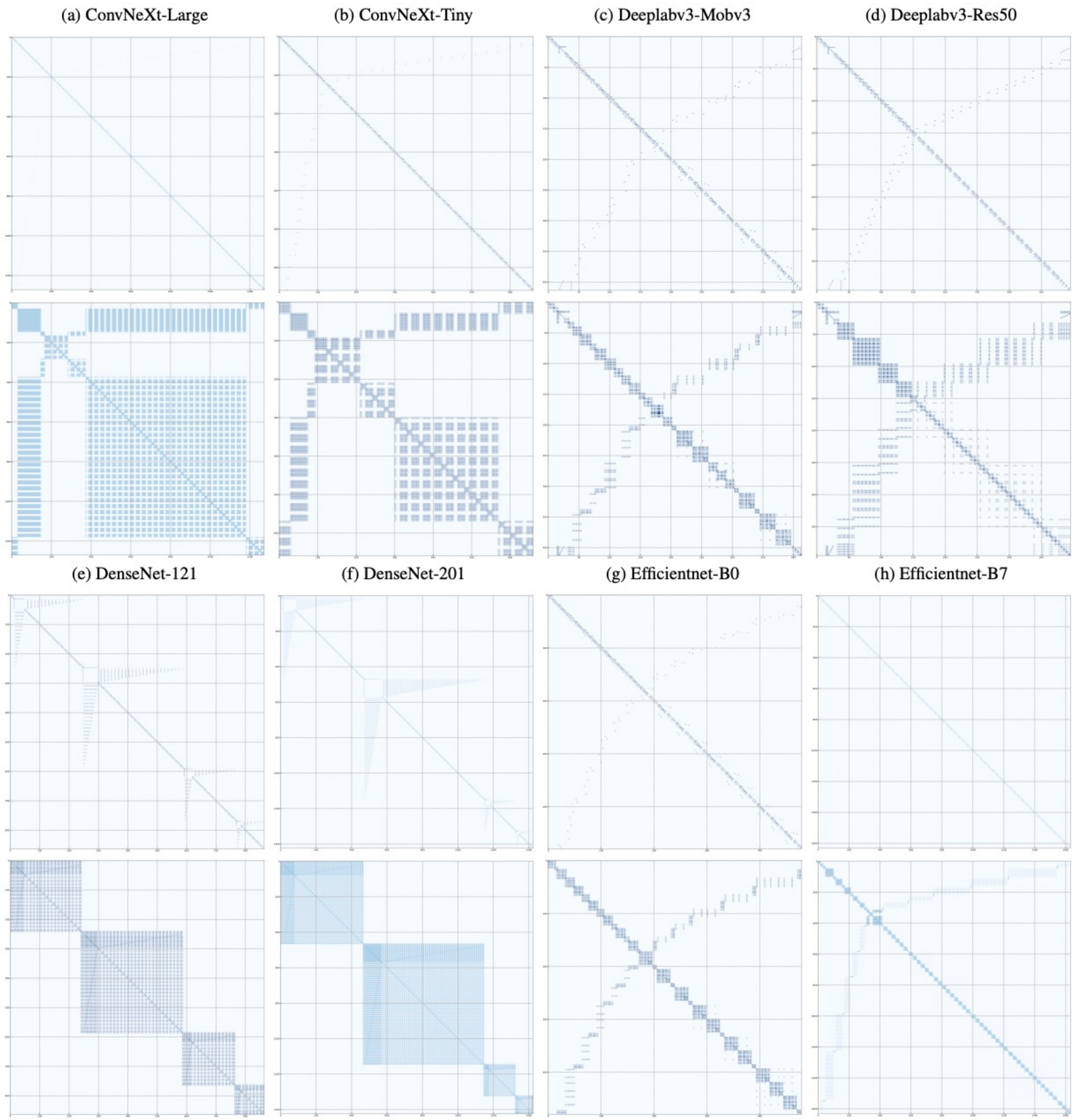


Figure 5. Dependency graphs (top) and the derived grouping schemes (bottom) for different neural networks.

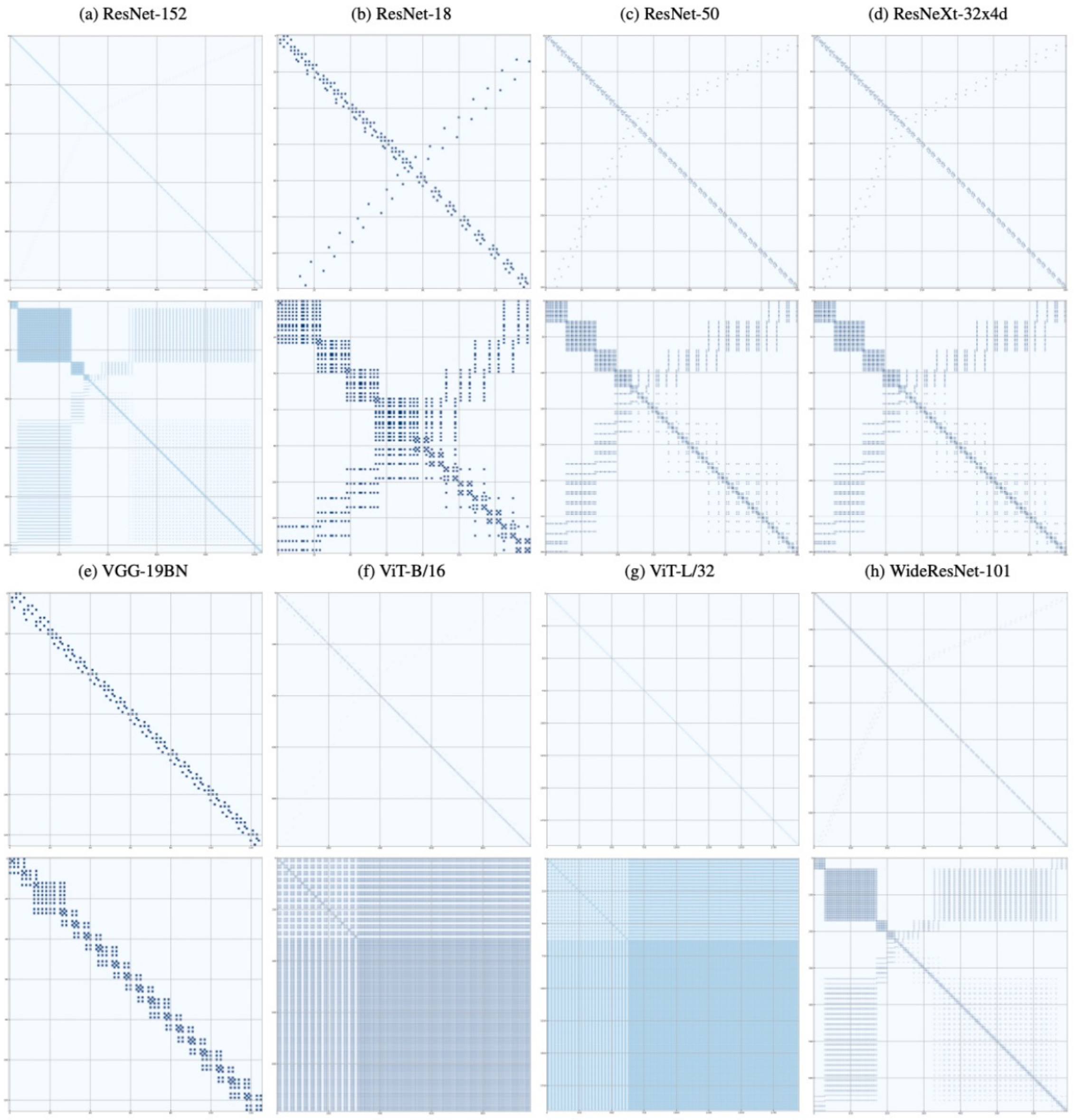


Figure 7. Dependency graphs (top) and the derived grouping schemes (bottom) for different neural networks

Projects

❑ Pruning in 3 lines

```python
# 1. build dependency graph for resnet18
DG = tp.DependencyGraph().build_dependency(model, example_inputs=torch.randn(1,3,224,224))

# 2. grouping parameters
pruning_group = DG.get_pruning_group( model.conv1, tp.prune_conv_out_channels, idxs=[2, 6, 9] )

# 3. remove parameters
pruning_group.exec()
```

❑ Grouping Example: pruning resnet18.conv1

```
-------------
[ <DEP: prune_conv => prune_conv on conv1 (Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), b
[ <DEP: prune_conv => prune_batchnorm on bn1 (BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_run
[ <DEP: prune_batchnorm => _prune_elementwise_op on _ElementWiseOp()>, Index=[2, 6, 9], NumPruned=0]
[ <DEP: _prune_elementwise_op => _prune_elementwise_op on _ElementWiseOp()>, Index=[2, 6, 9], NumPruned=0]
[ <DEP: _prune_elementwise_op => prune_related_conv on layer1.0.conv1 (Conv2d(64, 64, kernel_size=(3, 3), str
[ <DEP: _prune_elementwise_op => _prune_elementwise_op on _ElementWiseOp()>, Index=[2, 6, 9], NumPruned=0]
[ <DEP: _prune_elementwise_op => prune_batchnorm on layer1.0.bn2 (BatchNorm2d(64, eps=1e-05, momentum=0.1, af
[ <DEP: prune_batchnorm => prune_conv on layer1.0.conv2 (Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), pa
[ <DEP: _prune_elementwise_op => _prune_elementwise_op on _ElementWiseOp()>, Index=[2, 6, 9], NumPruned=0]
[ <DEP: _prune_elementwise_op => prune_related_conv on layer1.1.conv1 (Conv2d(64, 64, kernel_size=(3, 3), str
[ <DEP: _prune_elementwise_op => _prune_elementwise_op on _ElementWiseOp()>, Index=[2, 6, 9], NumPruned=0]
[ <DEP: _prune_elementwise_op => prune_batchnorm on layer1.1.bn2 (BatchNorm2d(64, eps=1e-05, momentum=0.1, af
[ <DEP: prune_batchnorm => prune_conv on layer1.1.conv2 (Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), pa
[ <DEP: _prune_elementwise_op => _prune_elementwise_op on _ElementWiseOp()>, Index=[2, 6, 9], NumPruned=0]
[ <DEP: _prune_elementwise_op => prune_related_conv on layer2.0.conv1 (Conv2d(64, 128, kernel_size=(3, 3), st
[ <DEP: _prune_elementwise_op => prune_related_conv on layer2.0.downsample.0 (Conv2d(64, 128, kernel_size=(1,
11211 parameters will be pruned
-------------
```

TORCH PRUNING

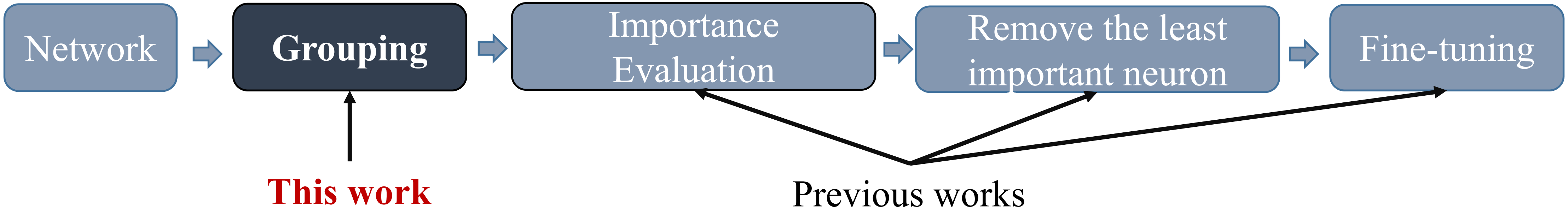
Torch-Pruning Public

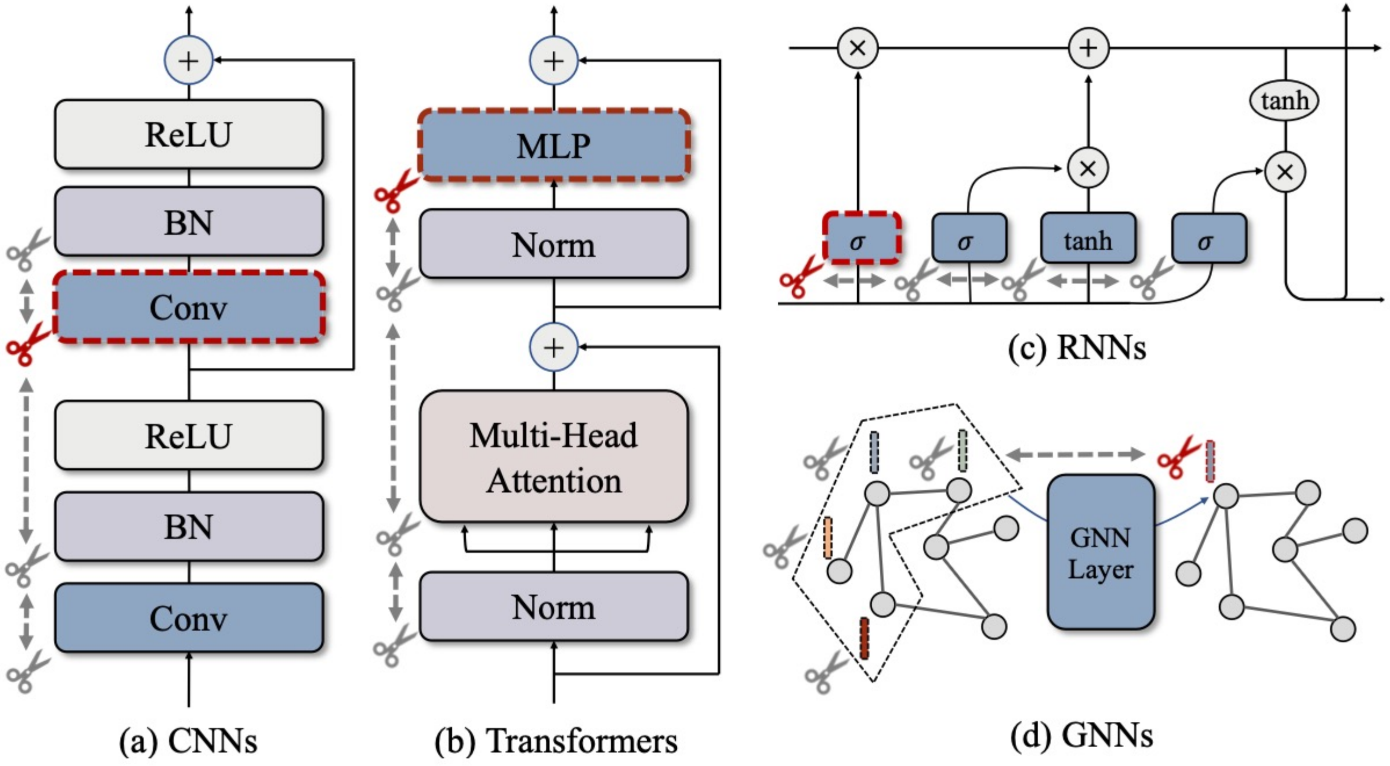
[CVPR-2023] Towards Any Structural Pruning; LLaMA / YOLOv8 / CNNs / Transformers

Python ☆ 1.1k 178

Conclusion

DepGraph: a simple way to prune any neural network





Thanks for Watching