Computation-Performance Optimization of Convolutional Neural Networks with Redundant Kernel Removal

Chih-Ting Liu, Yi-Heng Wu,Yu-Sheng Lin, and Shao-Yi Chien

Media IC and System Lab

Graduate Institute of Electronics Engineering

National Taiwan University,Taiwan

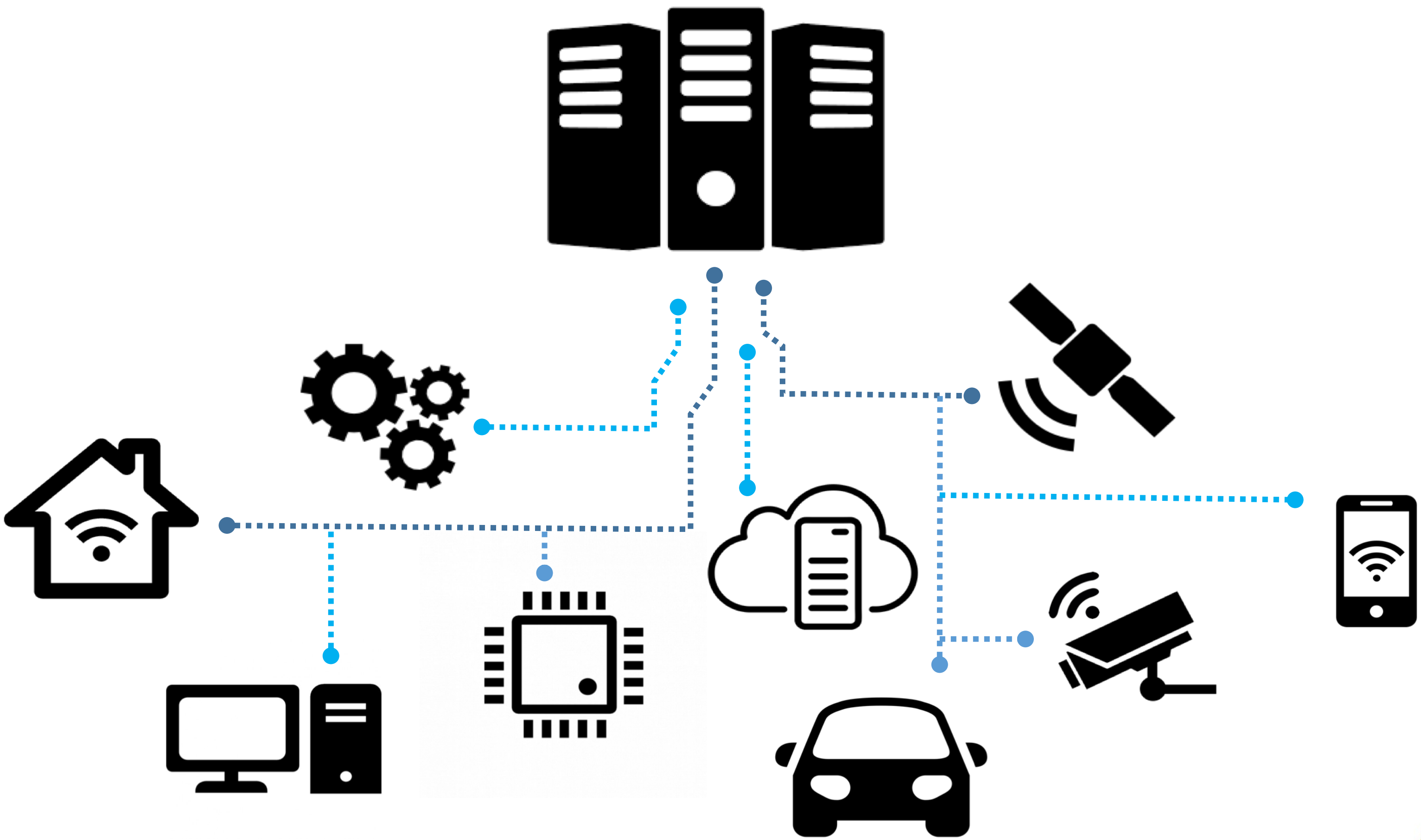
IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS & SYSTEMS
ISCAS·2018
FLORENCE, ITALY | MAY 27-30, 2018
THE ART OF CIRCUITS & SYSTEMS

國立臺灣大學
National Taiwan University

Outline

- Introduction
  - Challenge of IOT+AI
- Pruning
- Proposed Method
  - Layer-wise Kernel Removal
  - Computation-Performance Optimization
- Experimental Results
- Conclusion

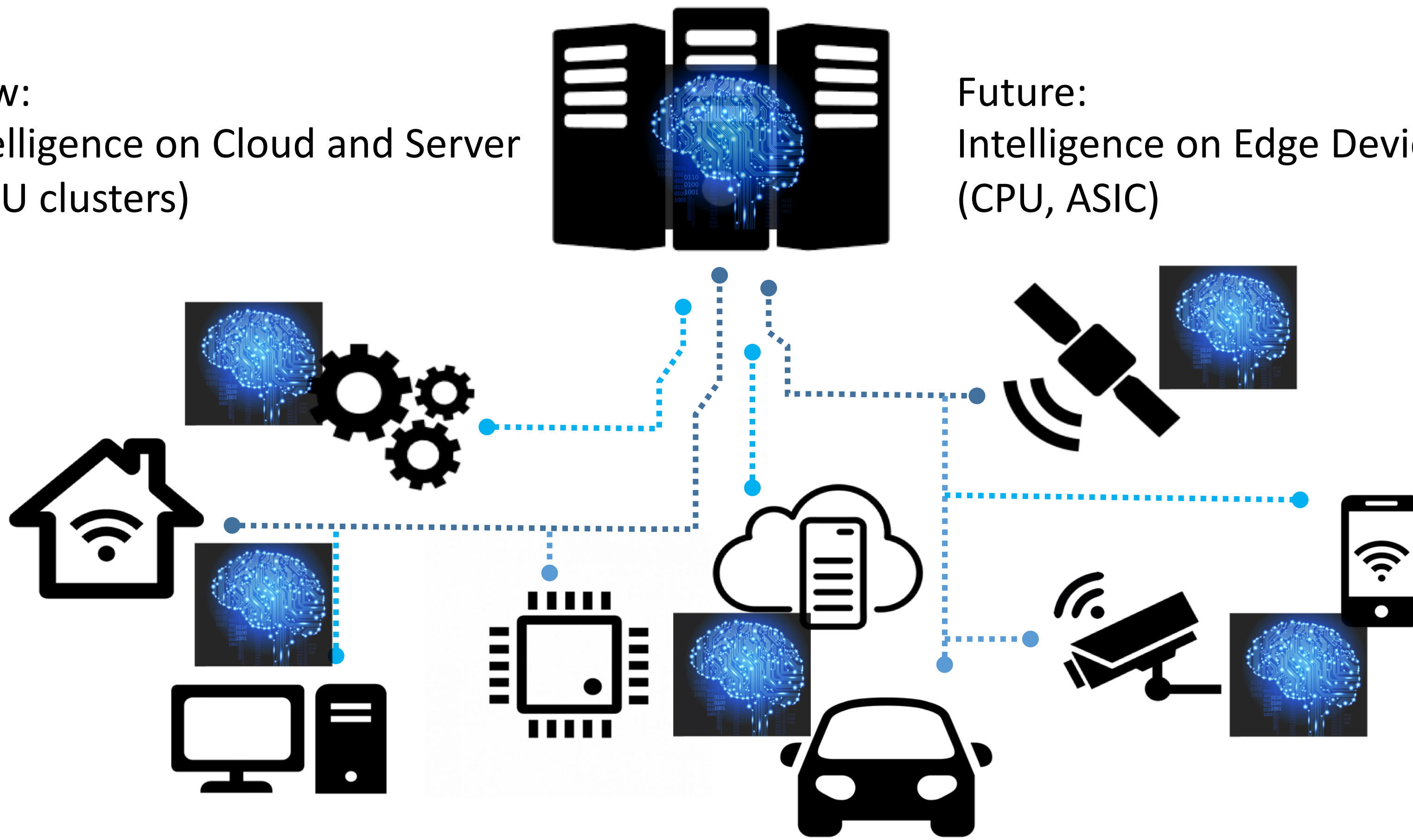
Internet of Things (IoT)



Internet of Things (IoT) + Artificial Intelligence (AI)

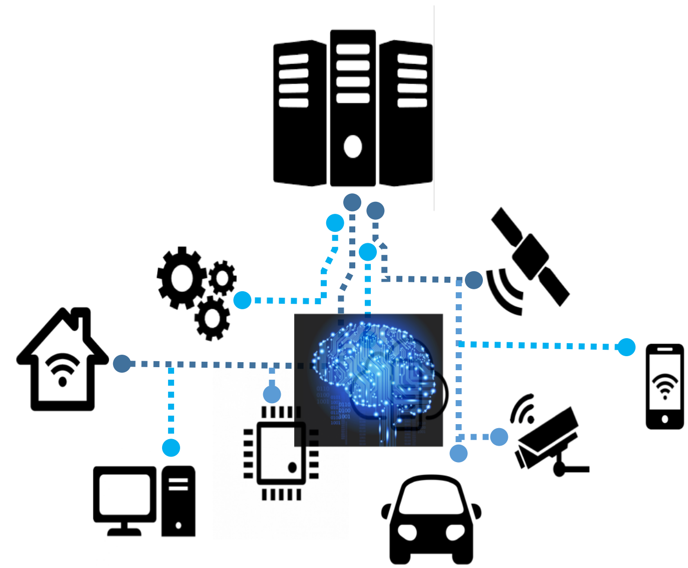
Now:
Intelligence on Cloud and Server
(GPU clusters)

Future:
Intelligence on Edge Devices
(CPU, ASIC)



IOT + Edge AI

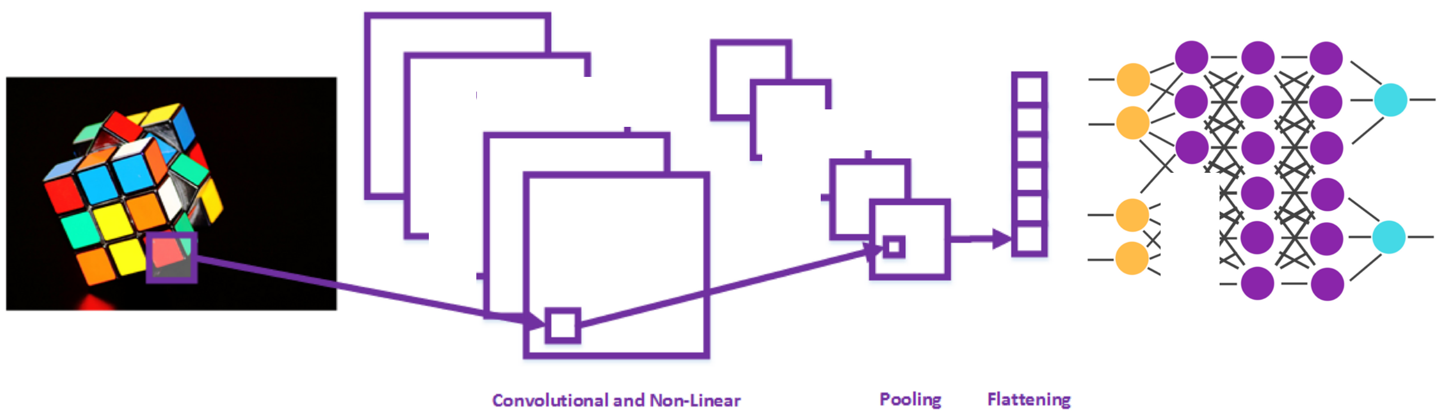
- For visual tasks, Convolutional Neural Networks (**CNN**) is one of the AI solutions.
- Apply the CNN inference procedure on devices.
- Challenges
  - # **Parameters** of deep CNN.
    (138M for VGG-16)
  - # **Operation** of deep CNN.
    (Convolution layers)



→ Hard to implement on CPU or ASIC
Cost huge (1) data access time and
(2) computation time

Goal

- Can we analyze and remove the possible redundancy of Deep CNN models?



Picture: https://goo.gl/images/oqYcfb

Pruning– Related Works

- Parameters-pruning based method [1]
  - Remove the parameters with small **magnitude**.
  - Cause sparse weight matrix with same size.

→ Reduce parameters. But can Not reduce operations!!

- Kernel-pruning based method [2]
  - Remove the entire kernel (filter) when sparsity $> threshold$.

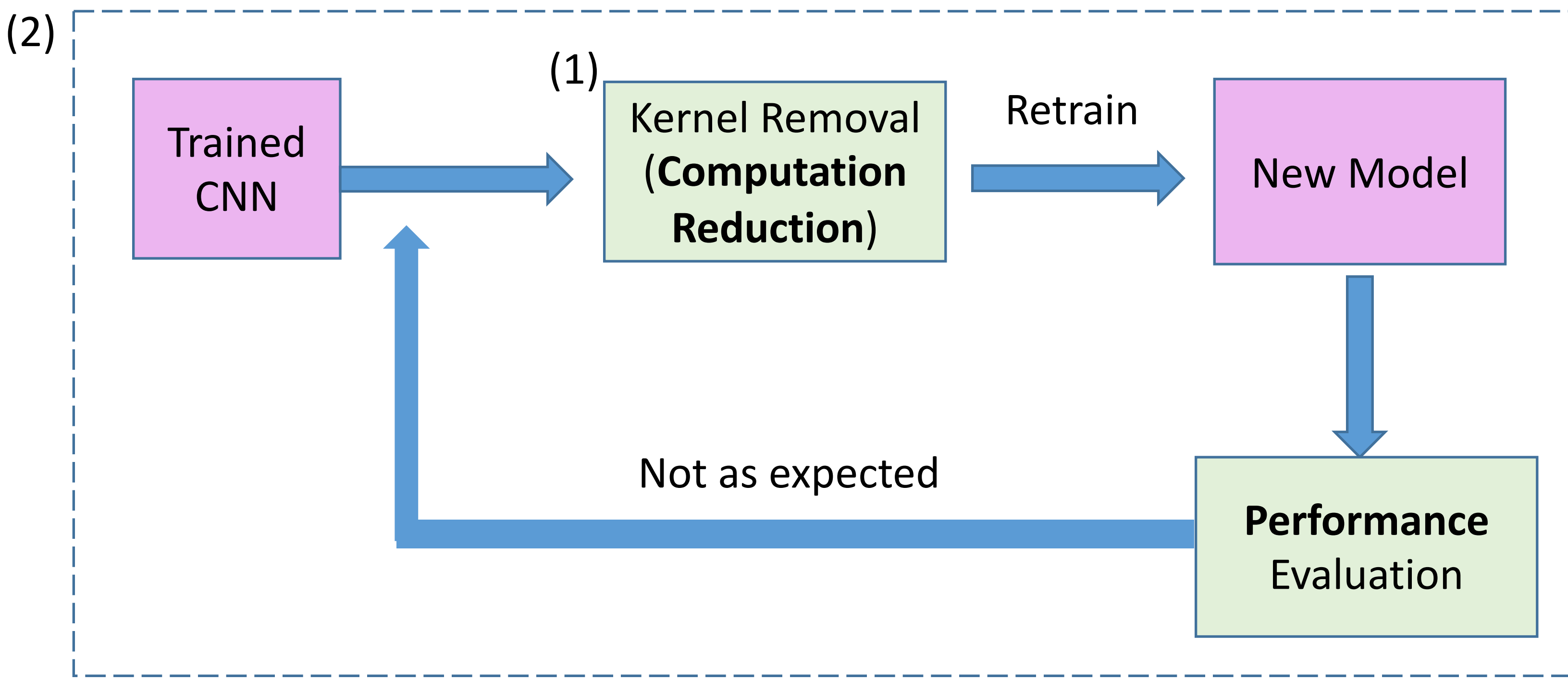
→ Can reduce operations! (reduction of convolution kernel)

[1] Han, Song, et al. "Learning both weights and connections for efficient neural network." Advances in neural information processing systems. 2015.

[2] C.-F. Chen, G. G. Lee, V. Sritapan, and C.-Y. Lin, “Deep convolutional neural network on iOS mobile devices,” in Proc. 2016 IEEE International Workshop on Signal Processing Systems (SiPS).

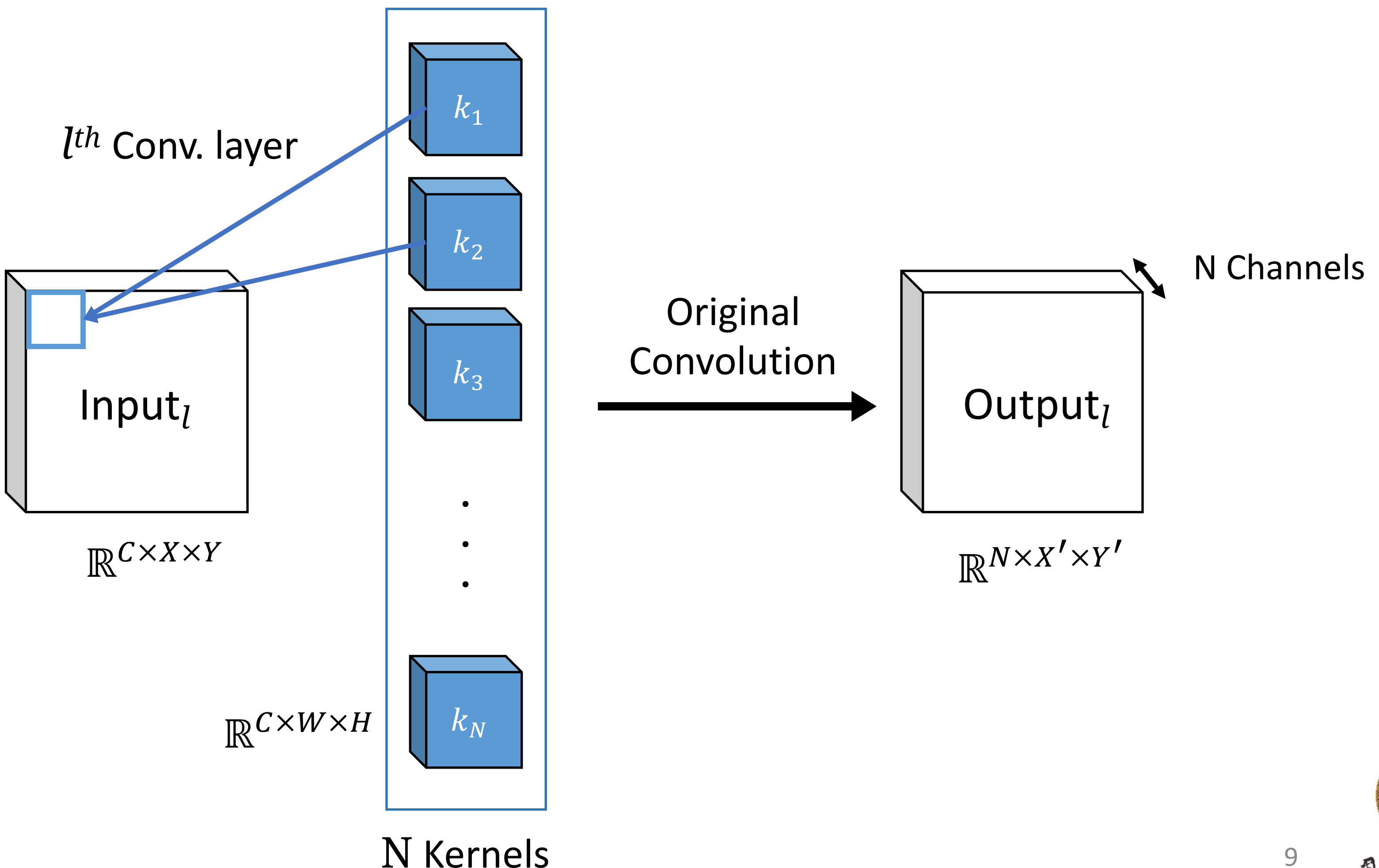
Proposed Method

- Based on [2] , we make up the disadvantage of manually determining the threshold when removing kernels.
- Propose (1) Layer-wise Kernel Removal (LKR)
  (2) Computation-Performance Optimization Flow (CPO)



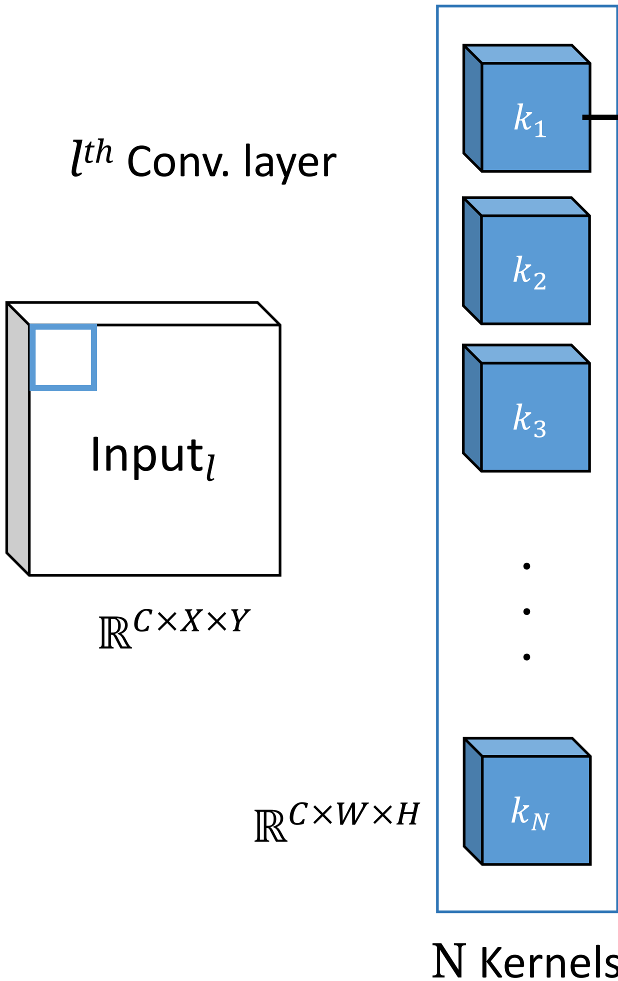
Layer-wise Kernel Removal

For a specific convolutional layer l,



Layer-wise Kernel Removal

For a specific convolutional layer l,



Sparsity per kernel :

$$S_l(n) = \frac{\sum_{c,h,w} \sigma(k_{l,nchw})}{C \times W \times H}$$

$\sigma(k_{l,nchw})$ ← # small values

$C \times W \times H$ ← # kernel values

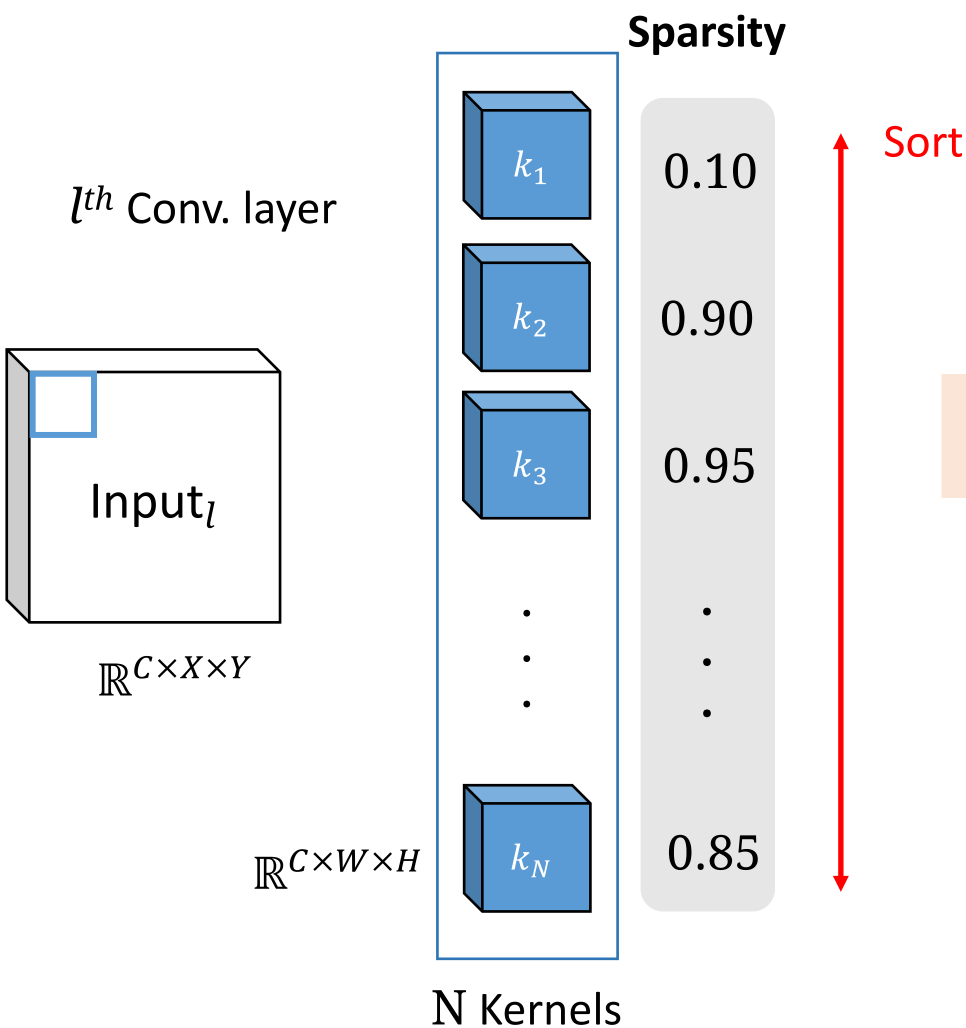
$$\sigma(x) = \begin{cases} 1, \text{if } |x| < M_l \\ 0, \text{otherwise} \end{cases}$$

M_l ← **mean value** of l^{th} Conv. layer

$$M_l = \frac{\sum_{n,c,h,w} |k_{l,nchw}|}{N \times C \times W \times H},$$

Layer-wise Kernel Removal

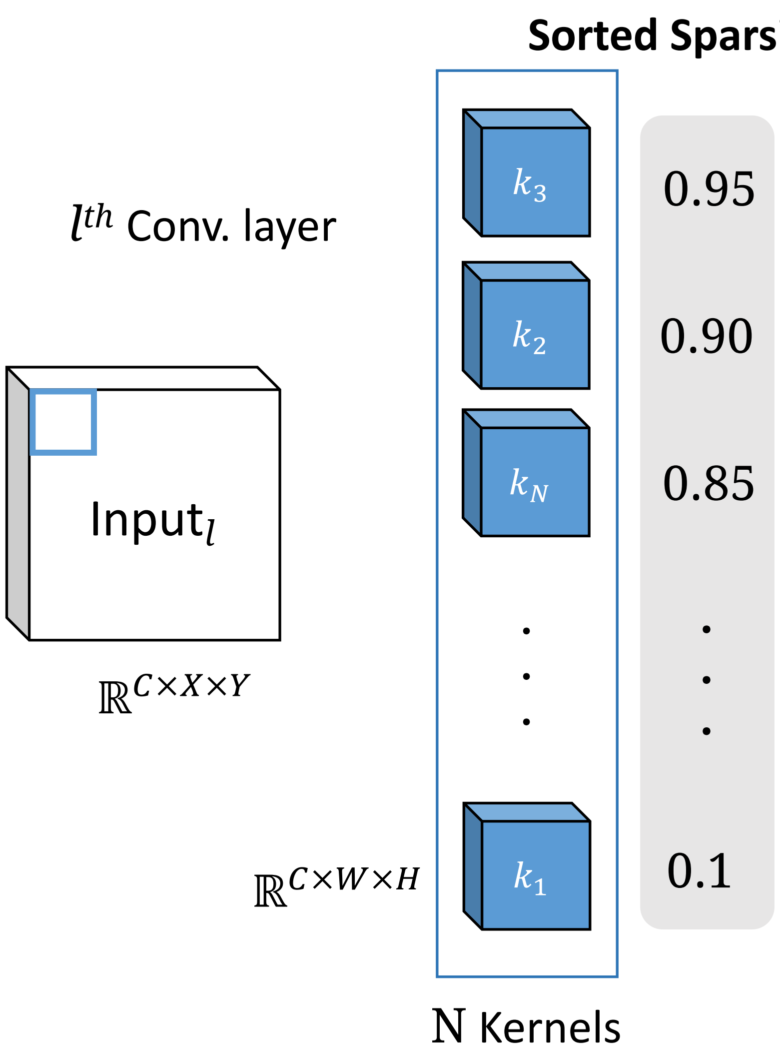
For a specific convolutional layer l,



Sparsity ↑, Redundancy ↑

Layer-wise Kernel Removal

For a specific convolutional layer l,

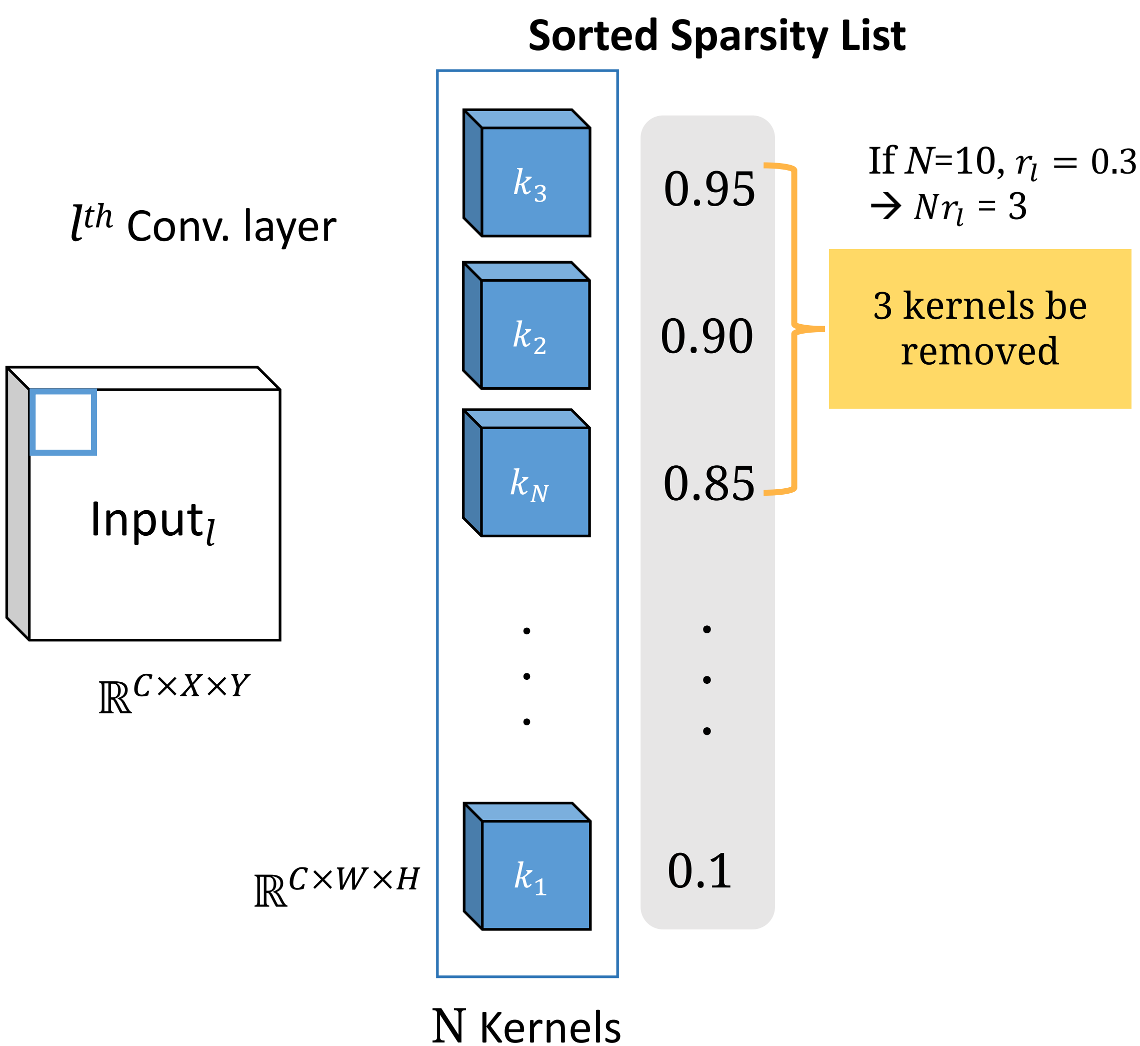


Define **reducing factor** r_l , $0 \leq r_l \leq 1$:
The proportion of removed kernel.

Given r_l, we will remove Nr_l kernels.

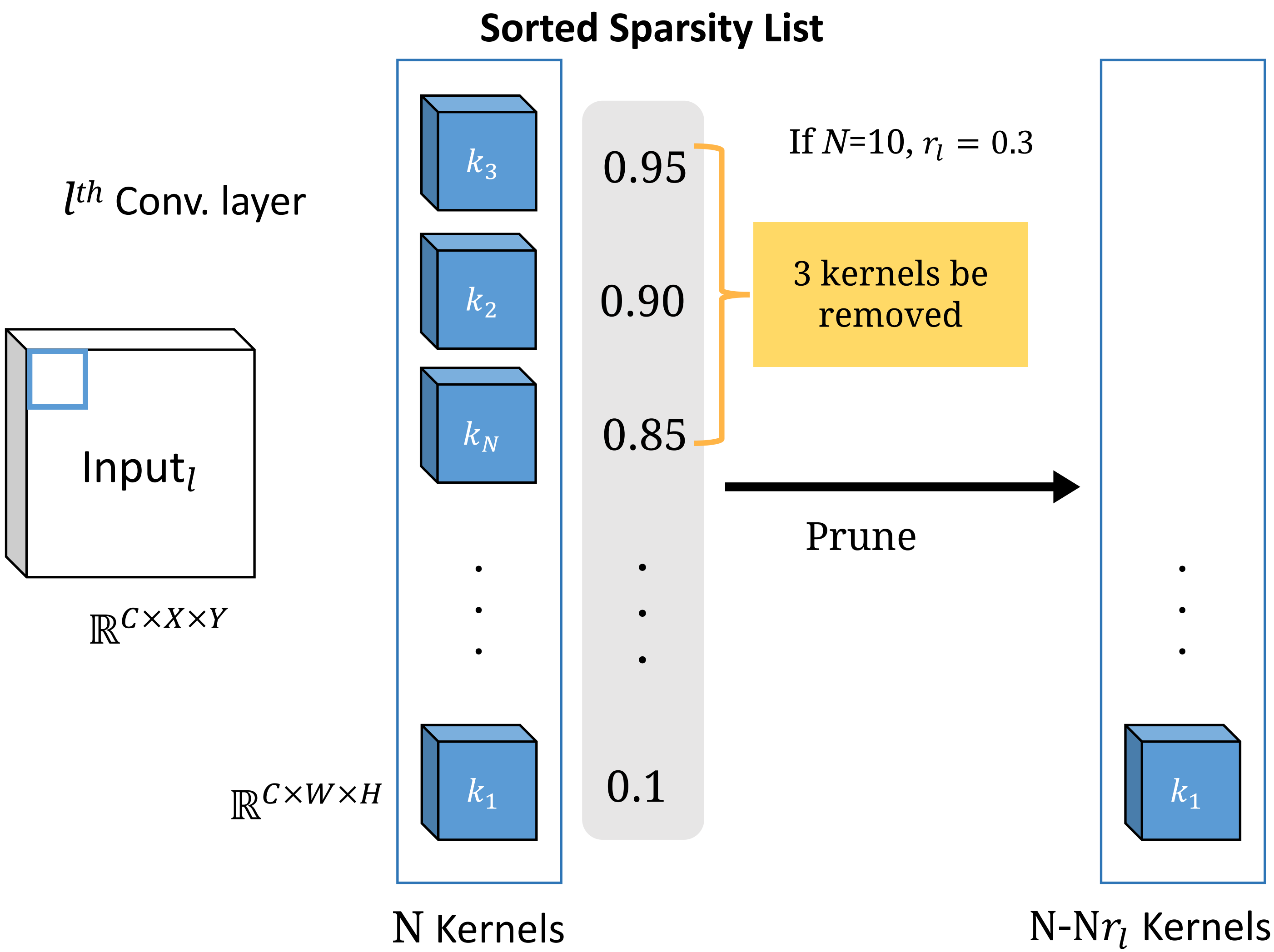
Layer-wise Kernel Removal

For a specific convolutional layer l,



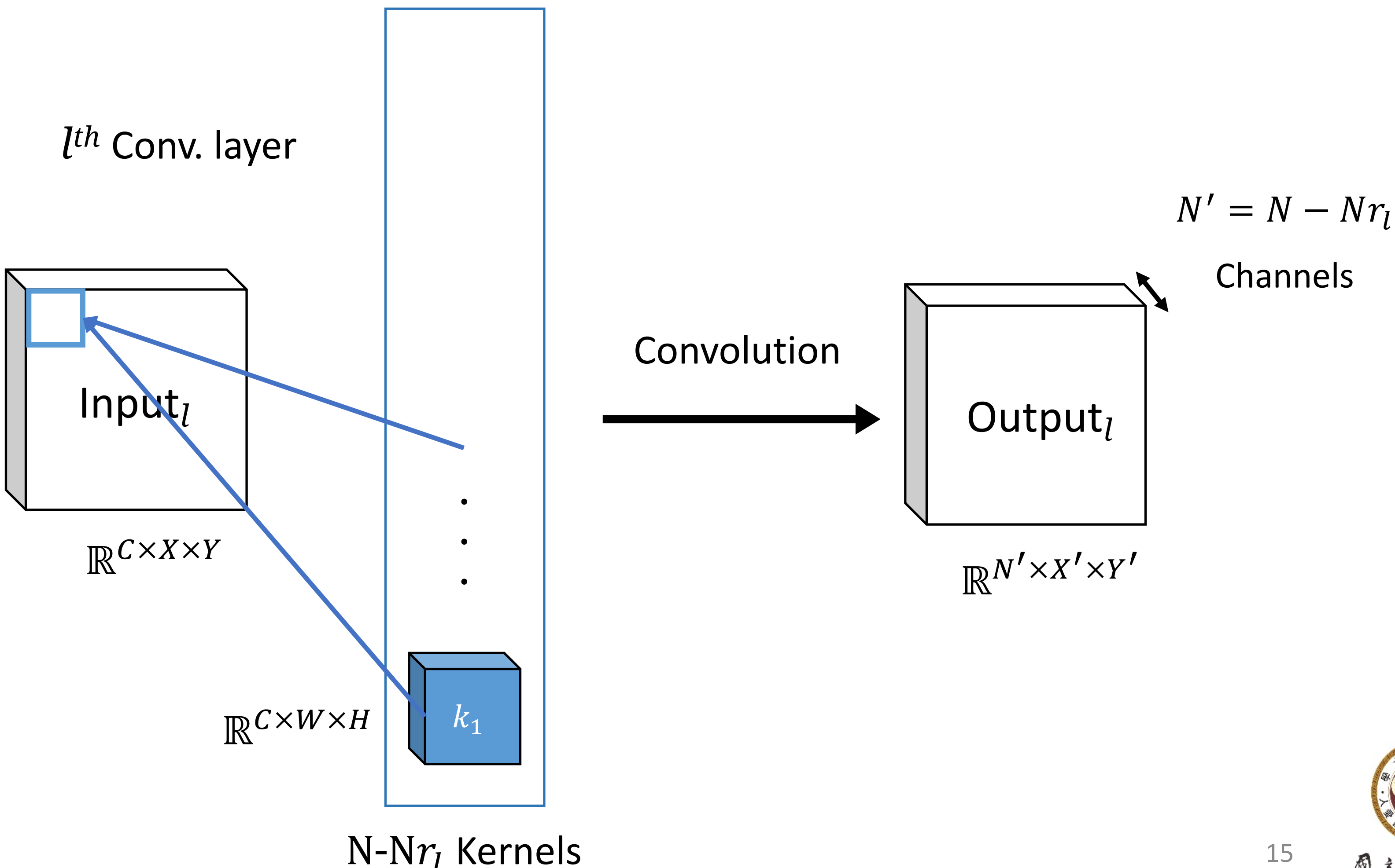
Layer-wise Kernel Removal

For a specific convolutional layer l,



Layer-wise Kernel Removal

For a specific convolutional layer l,

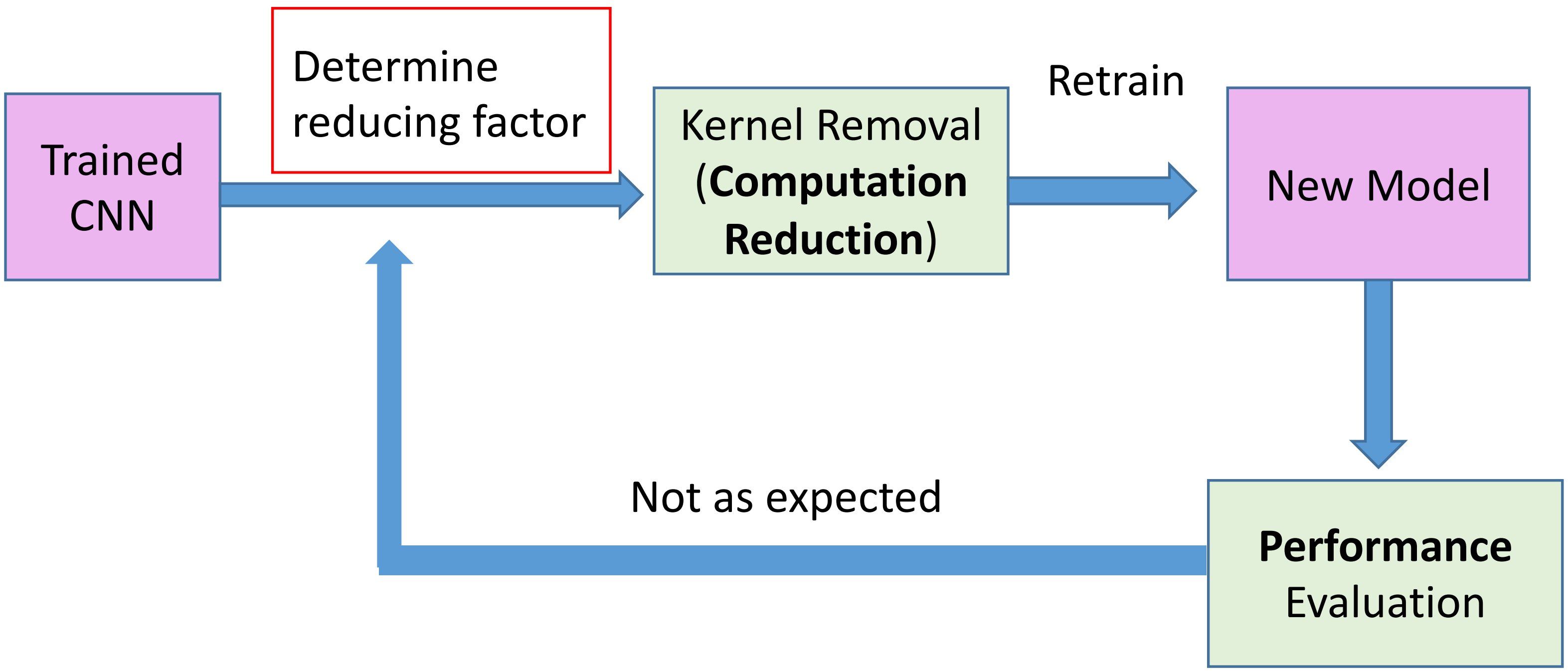


Summary of LKR

- Given a **reducing factor** $\boldsymbol{r_l}$ for each layer, we can prune the Conv. layers. (N-Nr_l kernels left)
- **Fast and Easy**.
- If we train from scratch with **smaller** network, not use pruning method, it may take time and may not train successfully.

Computation-Performance Optimization (CPO) Flow

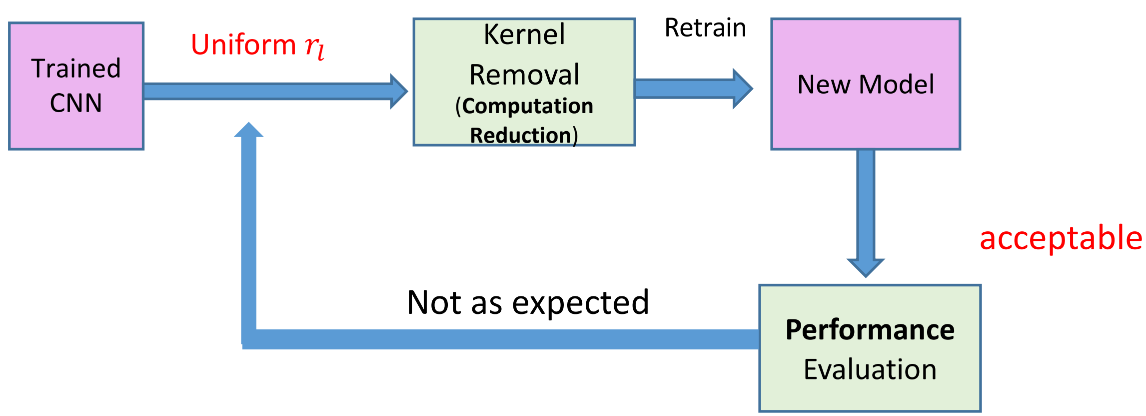
- Monitoring the Performance to adjust the Computation Reduction.



CPO Flow

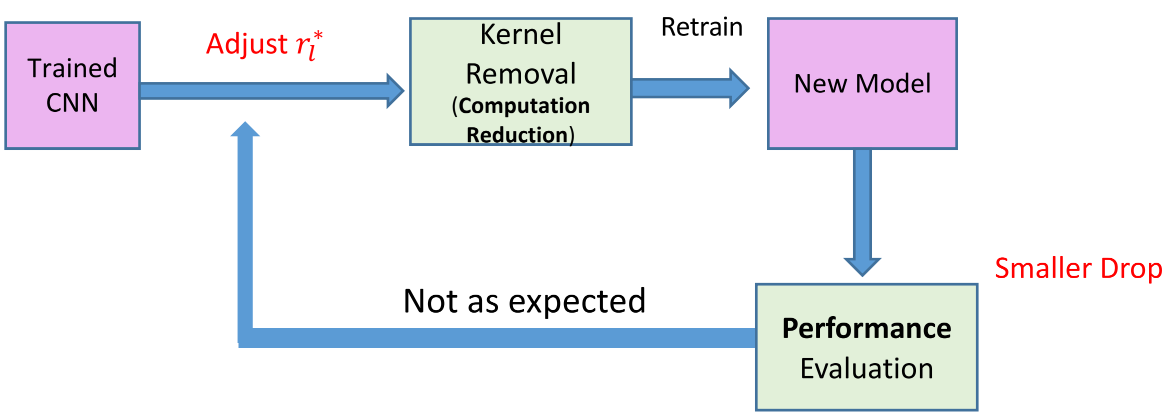
- Step 1 – Uniform Removal (UR)
  - Set **equal** r_l for every layer, obtain the performance drop.
  - Iteratively take experiments on **different** uniform r_l.
  - Pick an **acceptable** r_l^* among the experiments above.

    * "acceptable" depends on the performance requirement by user.
  - Go to step2 to further adjust r_l^* for every layer.



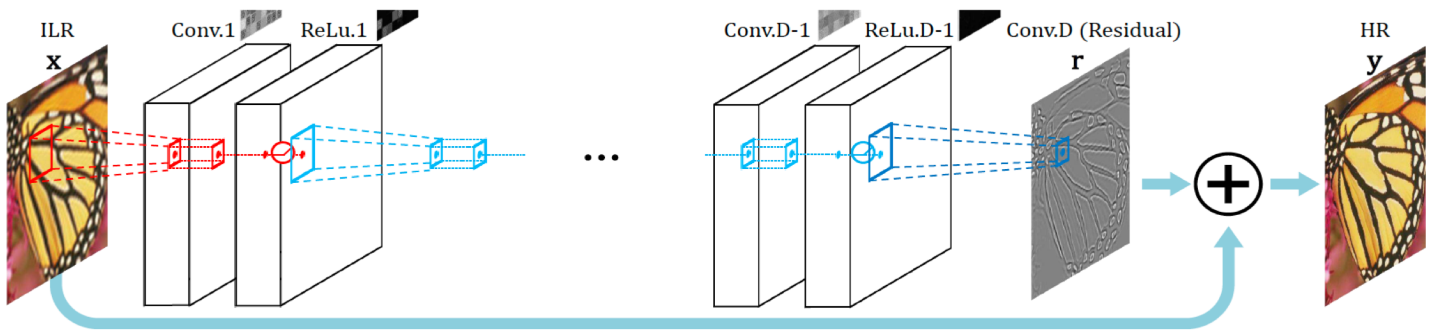
CPO Flow

- Step 2 – Probing Model Sensitivity
  - Split model into few parts (ex. front , middle ,end)
  - Under same parameters left, find which part is less sensitive → **Remove more** on the part with **lower sensitivity.**
  - Increase r_l^* for one part to probe sensitivity.
  - decrease r_l^* for other parts to maintain params left.
  - Observe the performance drop based on diff. probing.
  - Choose the new reducing factor with smaller performance drop!



Experimental Results

- CNN model: **VDSR** (**V**ery **D**eep **S**uper **R**esolution) model [3]
  - 20 layer fully convolutional Network→ Excluding the influence of FC layers.



ILR: Interpolated Low Resolution Image

HR: High Resolution Image

- Retrain Epoch: 8
- Evaluation Dataset: (1) Set5 X2 (2) Set14 X2
- Performance Evaluation : **PSNR (dB)**

[3] Kim et al. "Accurate image super-resolution using very deep convolutional networks." CVPR 2016.

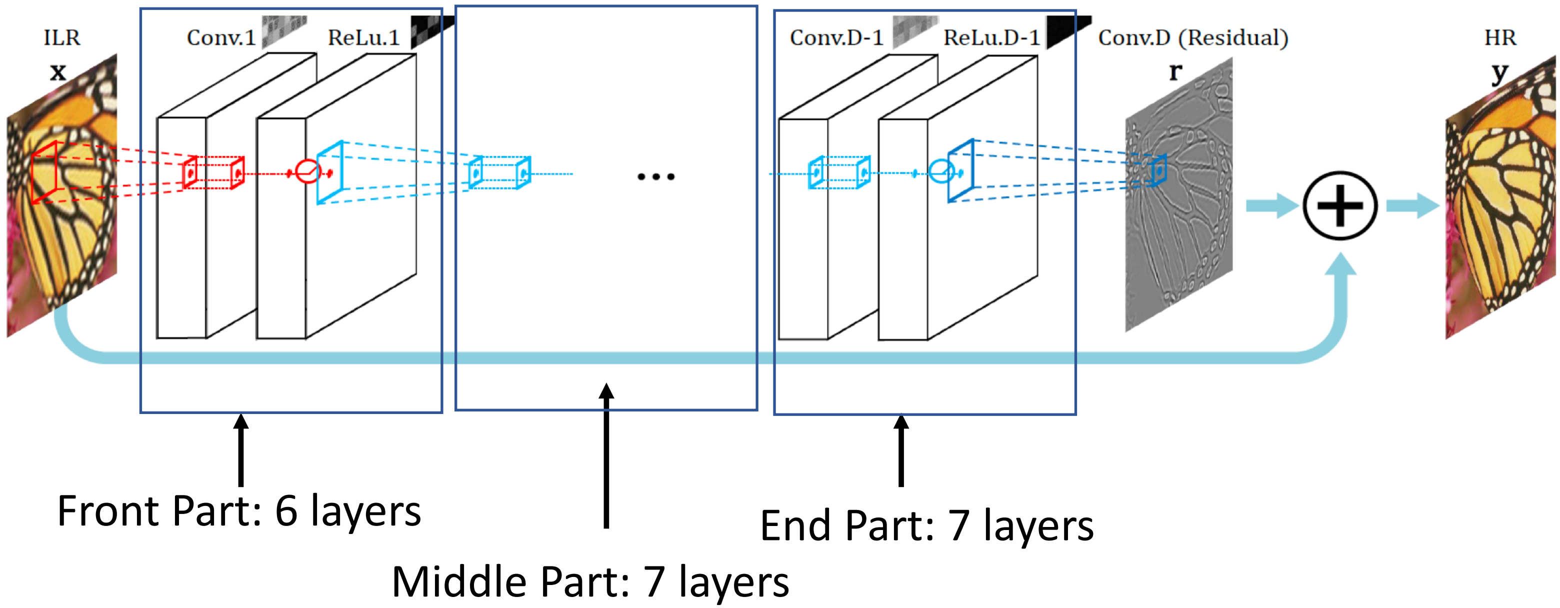
CPO step 1 Result – Uniform Removal(UR)

Model	Kernel per layer	Set5 PSNR	Set14 PSNR	Parameters	FLOP
Original	64	37.50 dB	33.08 dB	6.7×10^5	1.11×10^9

Reducing Factor (r)	Kernel per layer	Set5 PSNR Drop ↓ (dB)	Set14 PSNR Drop ↓ (dB)	Params Remained	FLOP Remained
0.00	64	0	0	100 (%)	100 (%)
0.12	56	0.13	0.19	76.60 (%)	76.58(%)
0.19	52	0.21	0.21	66.07 (%)	66.04(%)
0.25	48	0.19	0.25	56.32 (%)	56.28(%)
0.31	44	0.20	0.25	47.34 (%)	47.30(%)
0.38	40	0.36	0.36	39.15 (%)	39.10(%)
0.44	36	0.40	0.39	31.73 (%)	31.68(%)

CPO step 2 – Probe Model Sensitivity

- Split the model into three parts: **Front, Middle ,End**



- Start from $\boldsymbol{r^* = 0.25}$, and adjust the value for different parts.
- Try to **improve the performance** by pruning the **less sensitive part more** under almost same computation constraint.

CPO step2 Results

Reducing Factor (r) (F , M , E)	Kernel per parts (F, M, E)	Set5 PSNR Drop ↓ (dB)	Set14 PSNR Drop ↓ (dB)	Parameters Remained (%)
(0.25,0.25,0.25)	48, 48, 48	0.19	0.25	56.32 (UR)
(**0.44**,0.18,0.18)	36, 52, 52	0.19	0.23	55.39
(**0.44**,0.12,0.25)	36, 56, 48	0.18	0.20	56.36
(0.12,0.18,**0.44**)	56, 52, 36	0.24	0.29	58.60
(0.18,0.18,**0.38**)	52, 52, 40	0.30	0.26	57.54
(0.25,**0.44**,0.06)	48, 36, 60	0.27	0.30	55.94
(0.18,**0.44**,0.16)	52, 36, 56	0.26	0.30	55.51

→ The **Front part** of VDSR model is less sensitive; therefore, we can **increase** r_{front}.

→ Use (0.44,0.12,0.25) to prune VDSR can achieve 50% parameters removal and improve the UR performance. (PSNR drop 0.18/0.20)

Conclusion

- Layer-wise kernel removal can remove the redundancy of CNN and at the meantime reduce operations.
- Using Computation-Performance Optimization (CPO) Flow can remove more kernels in the model parts with more redundancy than others.

Extension

- CPO now is empirical, we've extended this work to develop a more **systematic CPO flow**.
- Have performed new CPO on VGG-19 for Cifar-10 classification.
- Compared with the state-of-the-art kernel removal method.
- Have submitted our extended work to **TCAS-1**.

Q & A

Thanks for Listening!

Backup

- Pruning on VGG-19 of Cifar-10

	Reducing Factor r	Drop(%)	Params Remained	FLOP Remained
Original	0	0% (Acc: 92.19%)	2.0×10^7 100%	4.0×10^8 100%
Uniform Removal (UR) [13]	0.125	0.10%	77.86%	76.87%
	0.250	0.54%	58.47%	56.78%
	0.375	1.36%	41.83%	39.72%
	0.500	3.23%	27.96%	25.70%
	0.625	6.42%	16.83%	**14.72%**

	D_{exp}	Final Drop	Params Remained	FLOP Remained
CPO (ours)	0.10%	0.06%	**26.89%**	**51.99%**
	0.54%	0.50%	**9.80%**	**35.02%**
	1.36%	1.35%	**5.16%**	**25.77%**
	3.23%	3.02%	**3.42%**	**20.96%**
	6.42%	6.30%	**1.76%**	16.76%

Backup

- VGG-16 on Cifar-10
- Compare with state-of-the-art [4]

Model	Error	FLOP	Pruned	Params	Pruned
VGG-16 [11]	6.75%	3.13×10^8		1.5×10^7	
Pruned [11]	6.60%	2.06×10^8	**34.2%**	5.4×10^6	64.00%
VGG-16	6.22%	3.13×10^8		1.5×10^7	
Pruned(CPO)	6.16%	2.09×10^8	33.4%	3.2×10^6	**78.76%**

[4] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets,"arXiv preprint arXiv:1608.08710, 2016