清華大學
Tsinghua University

交叉信息研究院
Institute for Interdisciplinary Information Sciences

Pruning Optimizations

马恺声

清华大学

Outline

- **Why We Need Pruning?**
- **Early Researches**
- **Optimization Solution: Structured Pruning**
- **Rethinking Works for Pruning**
- **Recent Works on Pruning**

Outline

- **Why We Need Pruning?**
- Early Researches
- Optimization Solution: Structured Pruning
- Rethinking Works for Pruning
- Recent Works on Pruning

Pruning Happens in Human Brain

50 Trillion Synapses → 1000 Trillion Synapses → 500 Trillion Synapses

Newborn

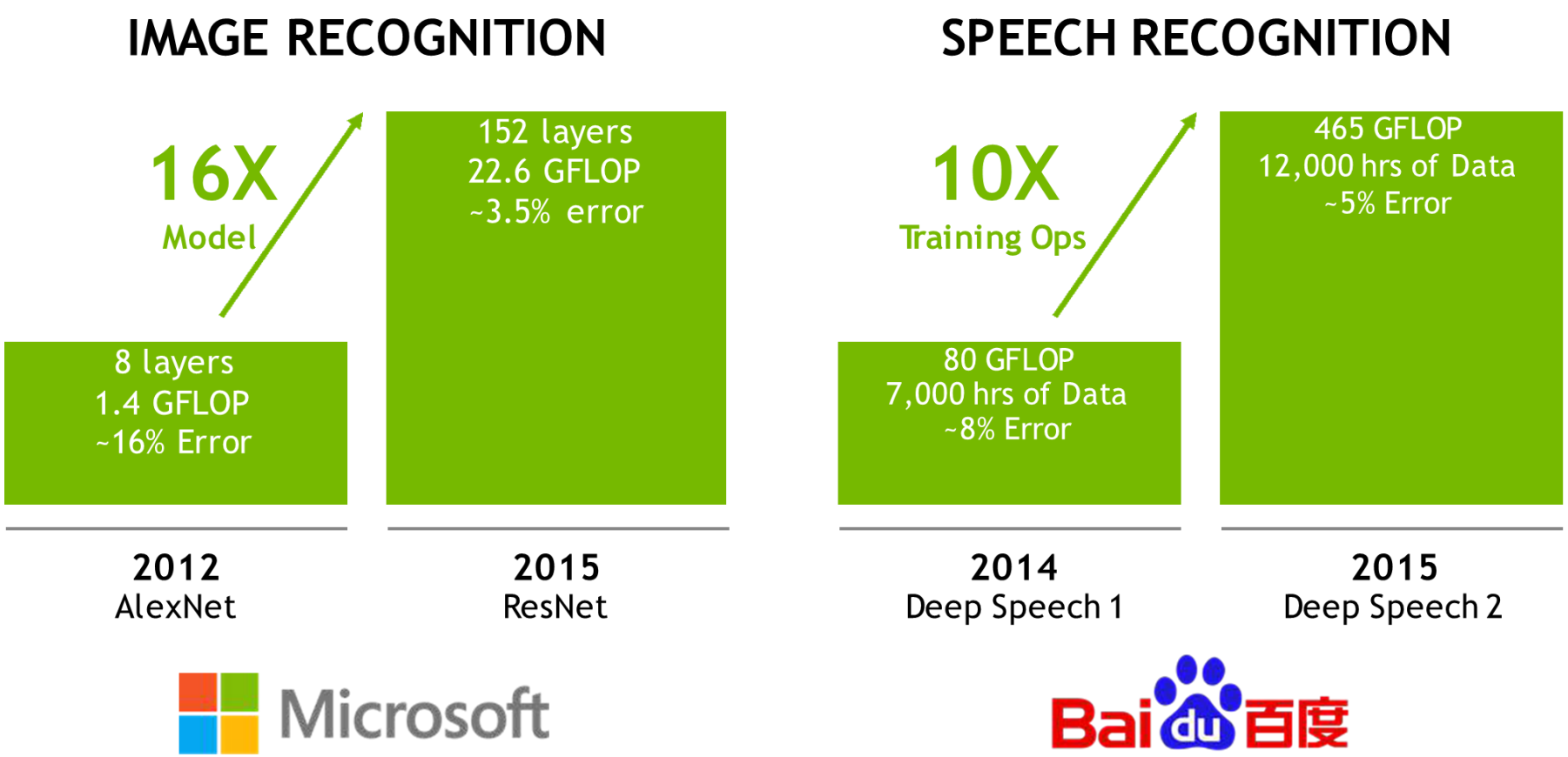
1 year old

Adolescent

Christopher A Walsh. Peter Huttenlocher (1931-2013). Nature, 502(7470):172–172, 2013.

Why We Need Pruning?

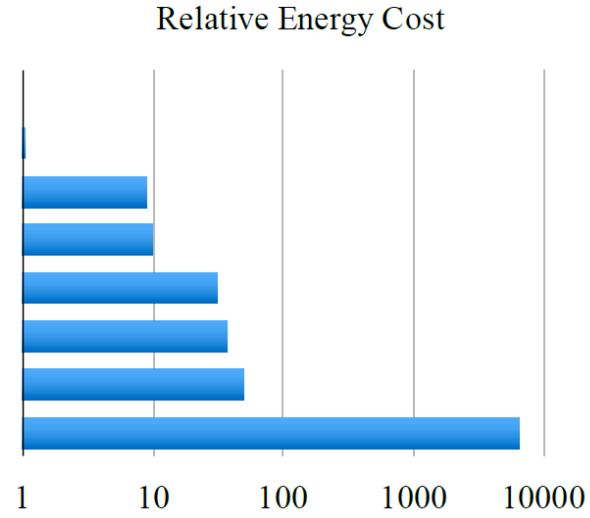
- **Models are Getting Larger**



Why We Need Pruning?

- **larger model => more memory access => more energy**

Operation	Energy [pJ]	Relative Cost
32 bit int ADD	0.1	1
32 bit float ADD	0.9	9
32 bit Register File	1	10
32 bit int MULT	3.1	31
32 bit float MULT	3.7	37
32 bit SRAM Cache	5	50
32 bit DRAM Memory	**640**	**6400**



how to make deep learning more efficient?

Mark Horowitz. Energy table for 45nm process, Stanford VLSI wiki

Outline

- Why We Need Pruning?
- **Early Researches**
- Optimization Solution: Structured Pruning
- Rethinking Works for Pruning
- Recent Works on Pruning

Early Approach to Pruning (1986~1993)

- **Magnitude based methods with weight decay (1980s)**

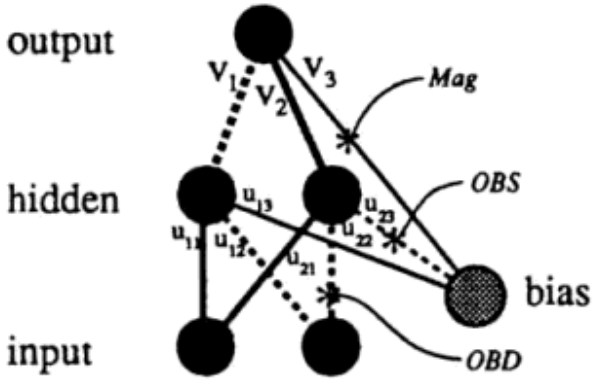
$$w_{ij}^{new} = (1-\varepsilon) w_{ij}^{old}$$

decay factor

$$E = E_0 + \frac{1}{2}\gamma \sum_{(ij)} w_{ij}^2$$

- Pros: feasible and very simple
- Cons: unfortunately often leads to the elimination of the wrong weights (small weights can be necessary for low error)

- **Optimal brain surgeon (1990s)**



Cost value for w_{kk} (higher means more important):

$$\frac{1}{2} h_{kk} \delta u_k^2 \qquad \left(h_{kk} = \sum_{(i,j)\in V_k} \frac{\partial^2 E}{\partial w_{ij}^2} \right)$$

- Pros: better than magnitude based method
- Cons: the second order derivative needs immensely additional computation

Pioneering Work (NIPS'15)

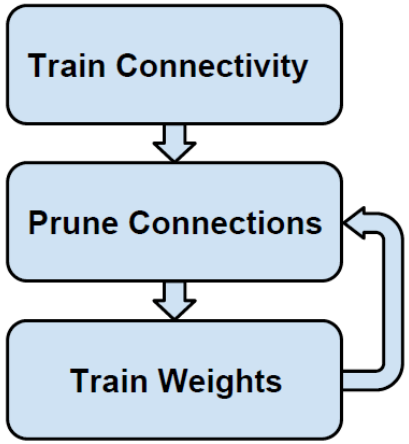


Figure 1: Three-Step Training Pipeline.

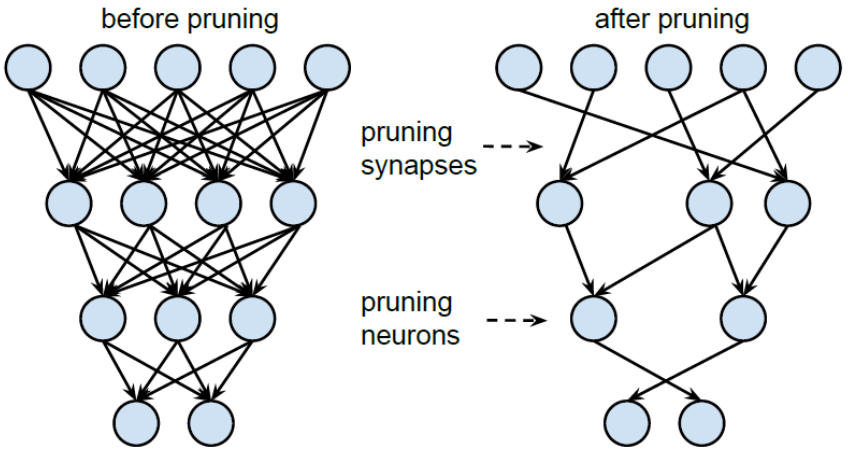


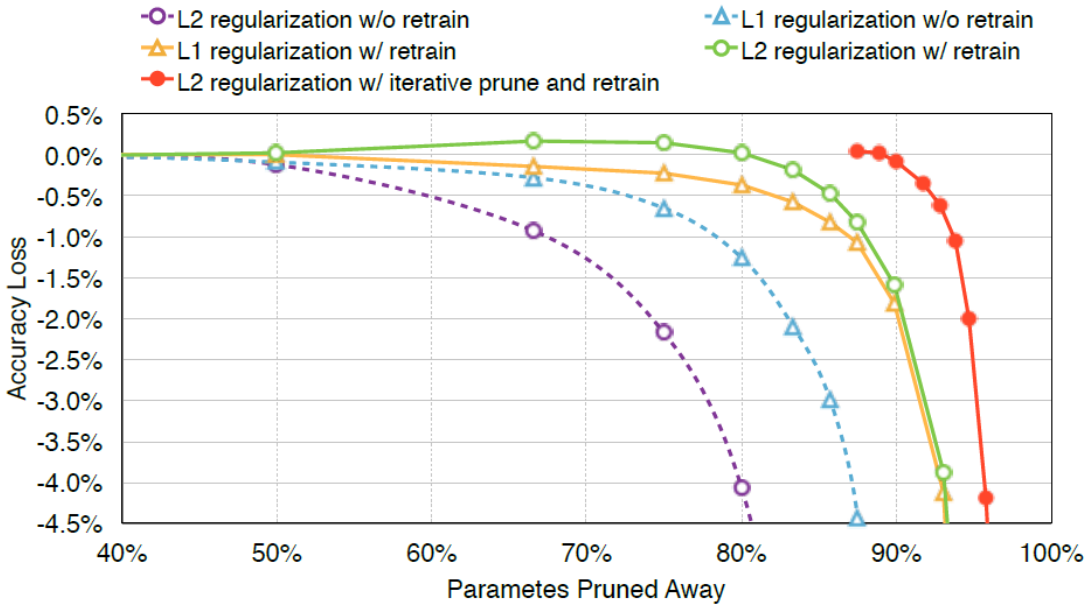
Figure 2: Synapses and neurons before and after pruning.

Step-1: Initial training
Step-2: Find the important connections based on a threshold (manually setting)
Step-3: Remove the dead neurons (those have no connections)
Step-4: Fine-tuning
Several iterations of step-2, step-3, and step-4

Tricks: L2-norm rather than L1; use drop-out ratio (lower in re-train stage); different thresholds across layers

Han, S., et al., "Learning both Weights and Connections for Efficient Neural Networks.". NIPS 2015

Pioneering Work (NIPS'15)



- Pros: excellent performance
- Cons: very long time to train a model (about one week for AlexNet); manual exploration for compression parameters

Network	Top-1 Error	Top-5 Error	Parameters	Compression Rate
Baseline Caffemodel [26]	42.78%	19.73%	61.0M	1×
Data-free pruning [28]	44.40%	-	39.6M	1.5×
Fastfood-32-AD [29]	41.93%	-	32.8M	2×
Fastfood-16-AD [29]	42.90%	-	16.4M	3.7×
Collins & Kohli [30]	44.40%	-	15.2M	4×
Naive Cut	47.18%	23.23%	13.8M	4.4×
SVD [12]	44.02%	20.56%	11.9M	5×
Network Pruning	**42.77%**	**19.67%**	**6.7M**	**9×**

Han, S., et al., "Learning both Weights and Connections for Efficient Neural Networks.". NIPS 2015

Improvement for Training Time (NIPS'16)

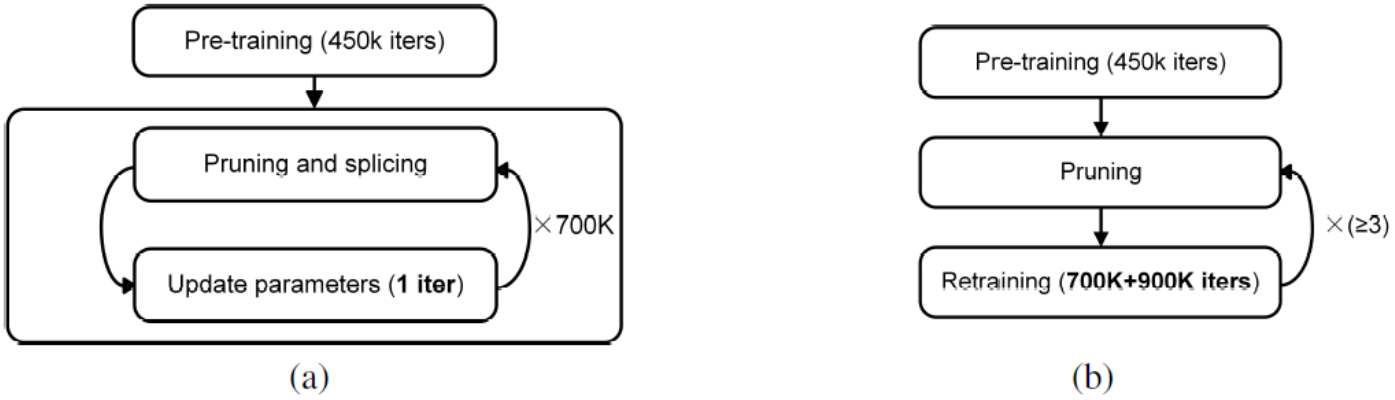
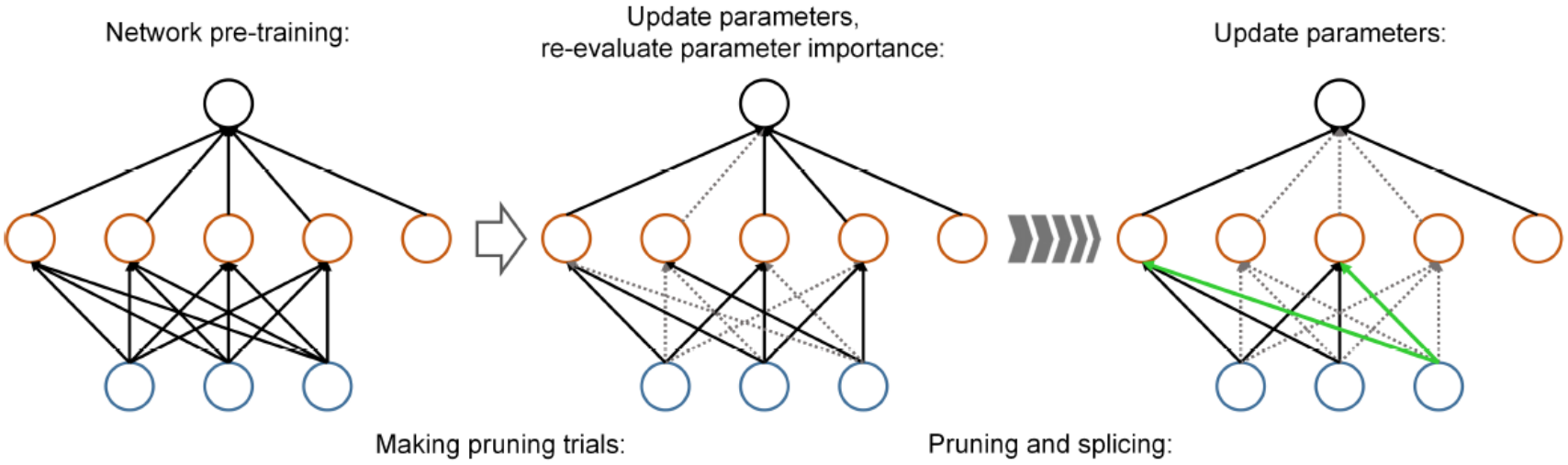
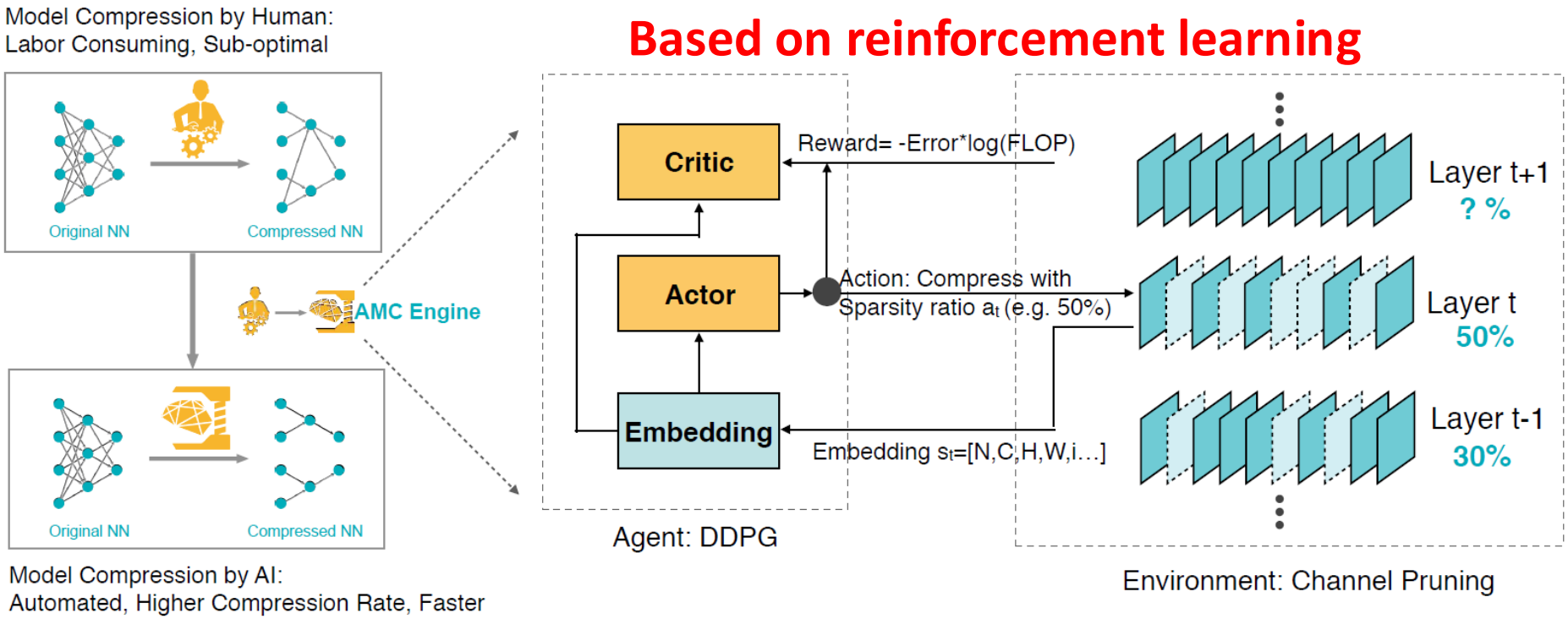


Figure 1: The pipeline of (a) our dynamic network surgery and (b) Han et al.'s method [9], using AlexNet as an example. [9] needs more than 4800K iterations to get a fair compression rate (9×), while our method runs only 700K iterations to yield a significantly better result (17.7×) with comparable prediction accuracy.



Improvement for Handcraft (ECCV'18)



	policy	FLOPs	Δacc %
VGG-16	FP (handcraft) [31]	20%	-14.6
	RNP (handcraft) [33]		-3.58
	SPP (handcraft) [49]		-2.3
	CP (handcraft) [22]		-1.7
	AMC (ours)		**-1.4**

MobileNet V1	uniform (0.75-224) [23]	56%	-2.5
	AMC (ours)	50%	**-0.4**
	uniform (0.75-192) [23]	41%	-3.7
	AMC (ours)	40%	**-1.7**
MobileNet V2	uniform (0.75-224) [44]	50%	-2.0
	AMC (ours)		**-1.0**

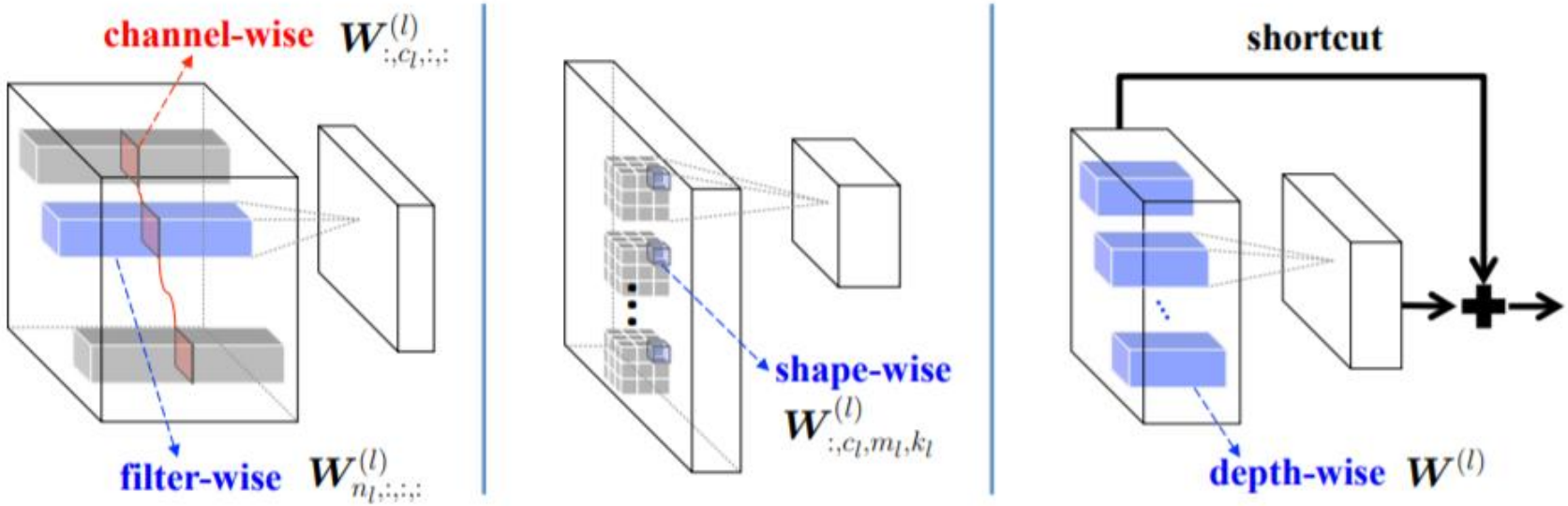
He, Y., et al., Amc: Automl for model compression and acceleration on mobile devices. ECCV 2018.

Outline

- Why We Need Pruning?
- Early Researches
- **Optimization Solution: Structured Pruning**
- Rethinking Works for Pruning
- Recent Works on Pruning

Optimize for the Development (NIPS'16)

People start to consider "structured pruning"

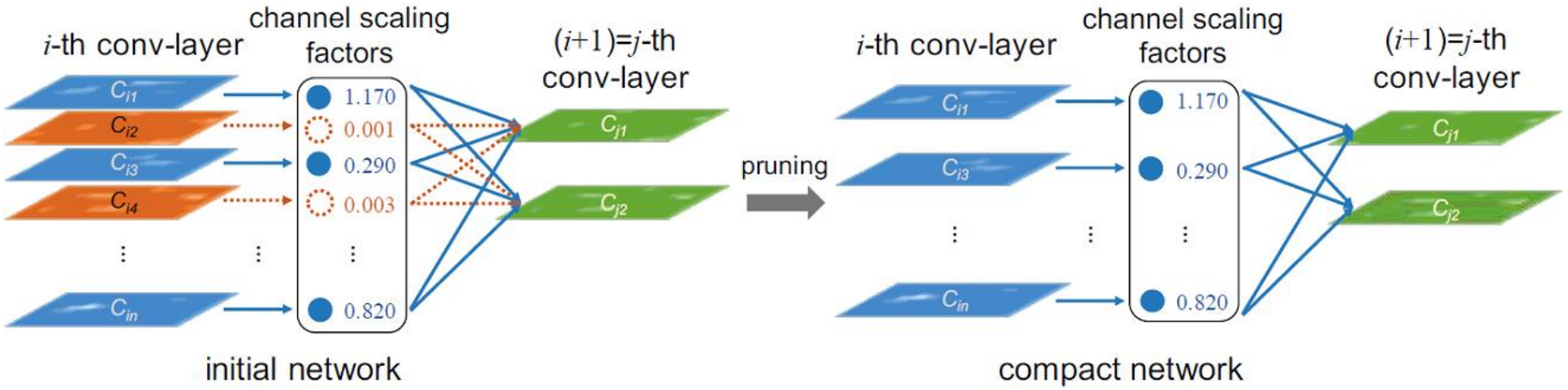


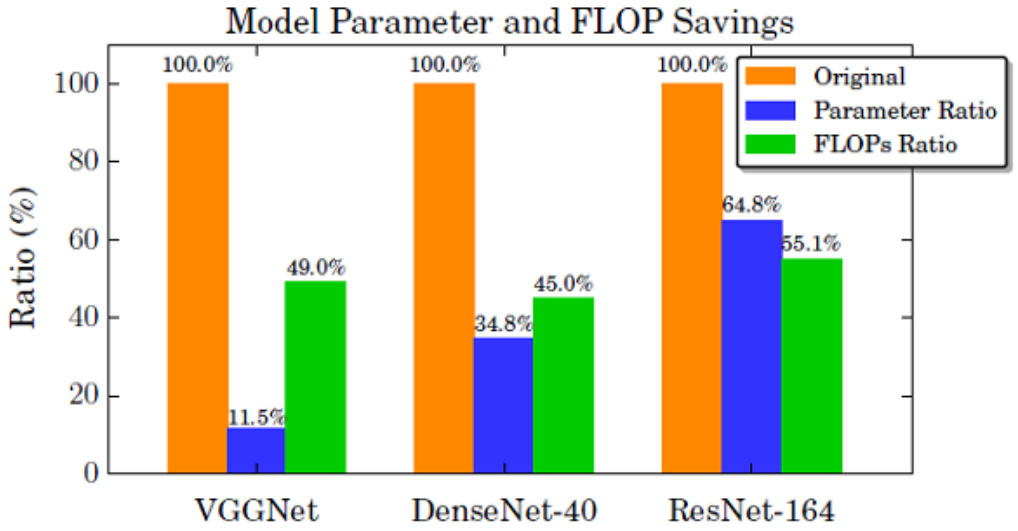
#	Method	Top1 err.	Statistics	conv1	conv2	conv3	conv4	conv5
1	ℓ_1	44.67%	sparsity	67.6%	92.4%	97.2%	96.6%	94.3%
			CPU ×	0.80	2.91	4.84	3.83	2.76
			GPU ×	0.25	0.52	1.38	1.04	1.36
2	SSL	44.66%	column sparsity	0.0%	63.2%	76.9%	84.7%	80.7%
			row sparsity	9.4%	12.9%	40.6%	46.9%	0.0%
			CPU ×	1.05	3.37	6.27	9.73	4.93
			GPU ×	1.00	2.37	4.94	4.03	3.05

Wen, W., et al., Learning Structured Sparsity in Deep Neural Networks. NIPS 2016.

Channel Pruning – 1 (ICCV'17)

Leverage BN scaling factor to decide "important channels"

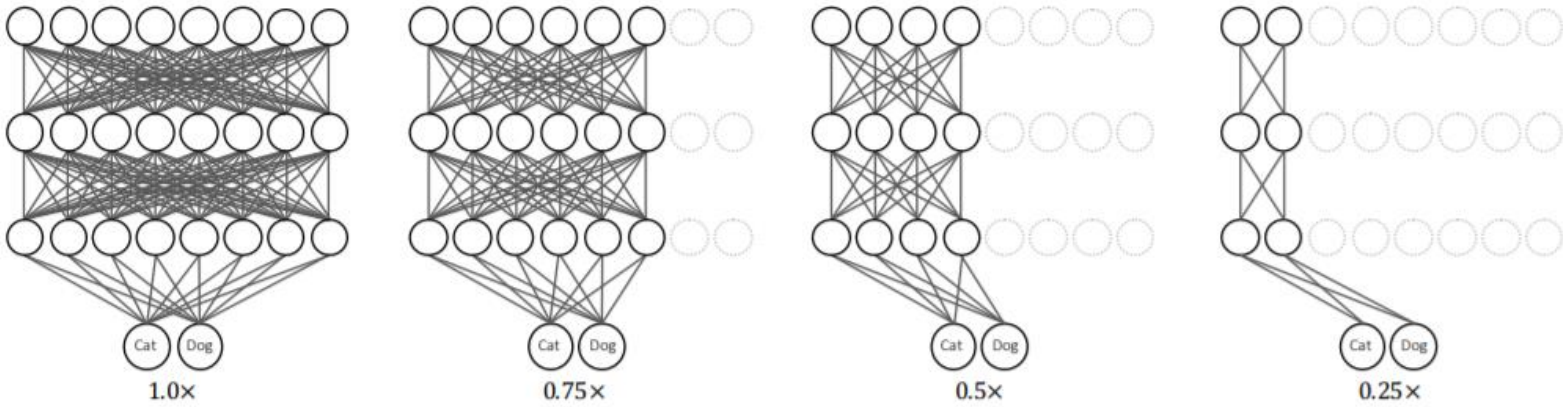




Liu, Z., et al., Learning Efficient Convolutional Networks through Network Slimming. ICCV 2017.

Channel Pruning – 2 (ICLR'19)

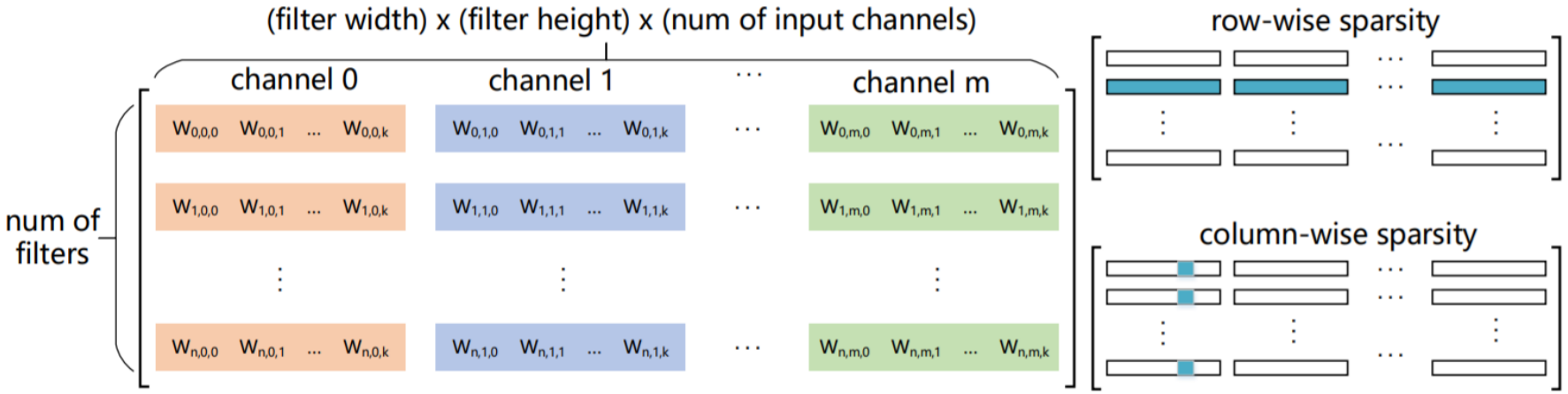
Dynamic channel pruning using switchable BN



Individual Networks			Slimmable Networks			FLOPs
Name	Params	Top-1 Err.	Name	Params	Top-1 Err.	
MobileNet v1 1.0×	4.2M	29.1	S-MobileNet v1 [0.25, 0.5, 0.75, 1.0]×	4.3M	28.5 (0.6)	569M
MobileNet v1 0.75×	2.6M	31.6			30.5 (1.1)	317M
MobileNet v1 0.5×	1.3M	36.7			35.2 (1.5)	150M
MobileNet v1 0.25×	0.5M	50.2			46.9 (3.3)	41M
MobileNet v2 1.0×	3.5M	28.2	S-MobileNet v2 [0.35, 0.5, 0.75, 1.0]×	3.6M	29.5 (-1.3)	301M
MobileNet v2 0.75×	2.6M	30.2			31.1 (-0.9)	209M
MobileNet v2 0.5×	2.0M	34.6			35.6 (-1.0)	97M
MobileNet v2 0.35×	1.7M	39.7			40.3 (-0.6)	59M
ShuffleNet 2.0×	5.4M	26.3	S-ShuffleNet [0.5, 1.0, 2.0]×	5.5M	28.7 (-2.4)	524M
ShuffleNet 1.0×	1.8M	32.6			34.5 (-0.9)	138M
ShuffleNet 0.5×	0.7M	43.2			42.7 (0.5)	38M

Yu, J., et al., Slimmable Neural Networks. ICLR 2019.

Put All Pruning Together w/ ADMM (ECCV'18)



$$\underset{\{\mathbf{W}_i\},\{\mathbf{b}_i\}}{\text{minimize}} \quad f\big(\{\mathbf{W}_i\},\{\mathbf{b}_i\}\big) + \sum_{i=1}^{N} g_i(\mathbf{W}_i),$$

where $g_i(\cdot)$ is the indicator function of $\mathbf{S}_i$, i.e.,

$$g_i(\mathbf{W}_i) = \begin{cases} 0 & \text{if } \mathrm{card}(\mathbf{W}_i) \le l_i, \\ +\infty & \text{otherwise.} \end{cases}$$

$$\underset{\{\mathbf{W}_i\},\{\mathbf{b}_i\}}{\text{minimize}} \quad f\big(\{\mathbf{W}_i\},\{\mathbf{b}_i\}\big) + \sum_{i=1}^{N} g_i(\mathbf{Z}_i),$$
$$\text{subject to} \quad \mathbf{W}_i = \mathbf{Z}_i,\ i = 1,\ldots,N.$$

The augmented Lagrangian [18] of the above optimization problem is given by

$$L_\rho\big(\{\mathbf{W}_i\},\{\mathbf{b}_i\},\{\mathbf{Z}_i\},\{\mathbf{\Lambda}_i\}\big) = f\big(\{\mathbf{W}_i\},\{\mathbf{b}_i\}\big) + \sum_{i=1}^{N} g_i(\mathbf{Z}_i) + \sum_{i=1}^{N} \mathrm{tr}[\mathbf{\Lambda}_i^T(\mathbf{W}_i - \mathbf{Z}_i)] + \sum_{i=1}^{N} \frac{\rho_i}{2}\|\mathbf{W}_i - \mathbf{Z}_i\|_F^2,$$

- Constraint sparsity of weight with $g_i(W_i)$.
- Note $g_i(W_i)$ is non-differentiable
- Leverage ADMM to solve the equation

Zhang, T., et al., A Systematic DNN Weight Pruning Framework using Alternating Direction Method of Multipliers, ECCV 2018
Zhang, T., et al., StructADMM: A Systematic, High-Efficiency Framework of Structured Weight Pruning for DNNs.

Put All Pruning Together w/ ADMM (ECCV'18)

Table 2: Comparison on column and row sparsity **with less than 2% accuracy loss** on AlexNet/CaffeNet using ImageNet ILSVRC-2012

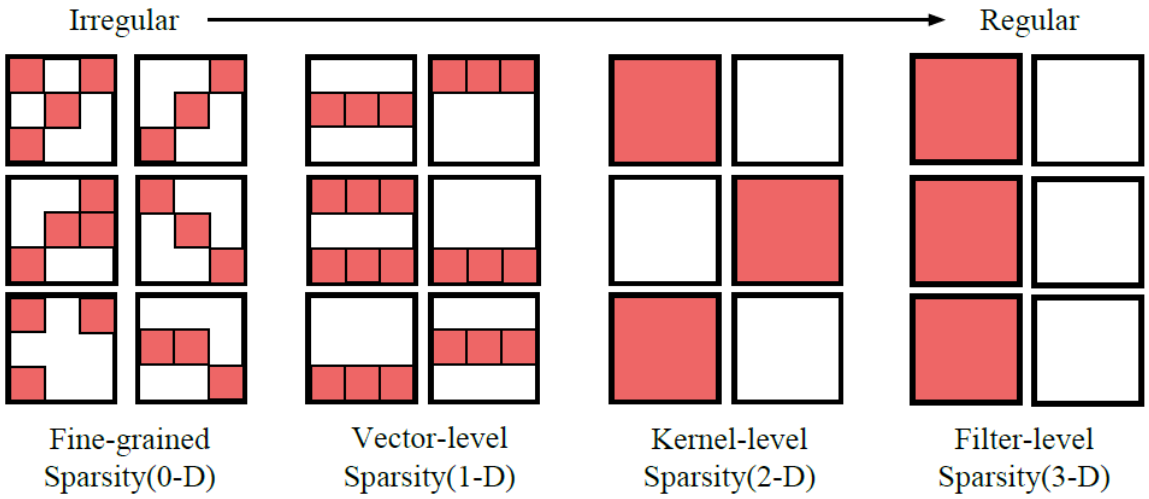
Method	Top1 Acc Loss	Statistics	conv1	conv2	conv3	conv4	conv5	conv2-5
SSL [Wen *et al.*, 2016]	2.0%	column sparsity	0.0%	63.2%	76.9%	84.7%	80.7%	84.4%§
		row sparsity	9.4%	12.9%	40.6%	46.9%	0.0%	
		pruning rate	1.1×	3.2×	7.7×	12.3×	5.2×	6.4×
our method	0.7%	column sparsity	0.0%	63.9%	78.1%	87.0%	84.9%	86.3%§
		row sparsity	9.4%	12.9%	40.6%	46.9%	0.0%	
		CPU×	1.05×	2.82×	6.63×	10.16×	5.00×	6.16×
		GPU1×	1.00×	1.28×	4.31×	1.75×	1.52×	2.29×
		GPU2×	1.00×	2.34×	6.85×	6.99×	4.15×	5.13×
		pruning rate	1.1×	3.1×	7.3×	14.5×	6.6×	7.3×
our method	2.0%	column sparsity	0.0%	87.5%	90.0%	90.5%	90.7%	93.7%§
		row sparsity	9.4%	12.9%	40.6%	46.9%	0.0%	
		CPU×	1.05×	8.00×	14.68×	14.22×	7.71×	11.93×
		GPU1×	1.00×	2.39×	5.34×	1.92×	2.04×	3.15×
		GPU2×	1.00×	4.92×	12.55×	8.39×	6.02×	8.52×
		pruning rate	1.1×	9.2×	16.8×	19.8×	8.4×	15.0×

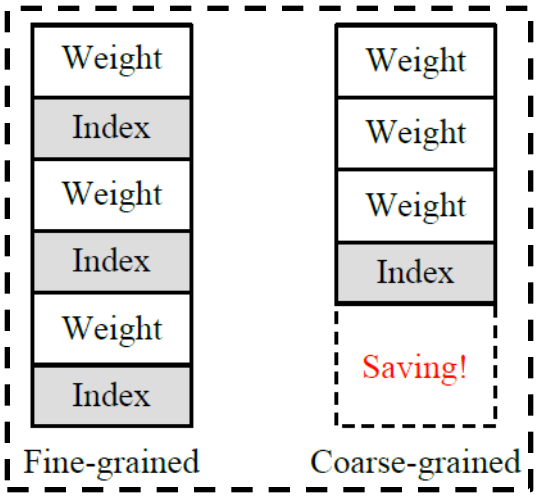
§ total sparsity accounting for both column and row sparsities.

Method	Top1 acc loss	Statistics	conv1	conv2	conv3	conv4	conv5	conv1-5§	conv2-5§
weight pruning [Han *et al.*, 2015]	0%	sparsity	16.0%	62.0%	65.0%	63.0%	63.0%	2.7×	2.7×
dynamic surgery [Guo *et al.*, 2016]	0.3%	sparsity	46.2%	59.4%	71.0%	67.7%	67.5%	3.1×	3.1×
NeST [Dai *et al.*, 2017]	0%	sparsity	51.1%	65.2%	81.5%	61.9%	59.3%	3.2×	3.3×
fine-grained pruning [Mao *et al.*, 2017]	−0.1% (Top5)	sparsity	17.0%	74.0%	77.0%	77.0%	77.0%	4.2×	4.3×
l_1 [Wen *et al.*, 2016]	−0.2%	sparsity	14.7%	76.2%	85.3%	81.5%	76.3%	5.0×	5.9×
our method	−0.2%	sparsity	0.0%	93.0%	94.3%	93.6%	93.5%	13.1×	16.1×

Zhang, T., et al., A Systematic DNN Weight Pruning Framework using Alternating Direction Method of Multipliers, ECCV 2018
Zhang, T., et al., StructADMM: A Systematic, High-Efficiency Framework of Structured Weight Pruning for DNNs.

Structured V.S. Non-Structured (NIPS'17)





Model	Top-5 Accuracy	Granularity	Density	Storage Ratio
AlexNet	80.3%	Kernel Pruning (2-D)	37.8%	39.7%
		Vector Pruning (1-D)	29.9%	34.5%
		Fine-grained Pruning (0-D)	22.1%	**33.0%**
VGG-16	90.6%	Kernel Pruning (2-D)	44.4%	46.9%
		Vector Pruning (1-D)	30.7%	**35.8%**
		Fine-grained Pruning (0-D)	27.0%	40.6%
GoogLeNet	89.0%	Kernel Pruning (2-D)	43.7%	51.6%
		Vector Pruning (1-D)	36.9%	**47.4%**
		Fine-grained Pruning (0-D)	32.3%	48.5%
ResNet-50	92.3%	Kernel Pruning (2-D)	61.3%	77.0%
		Vector Pruning (1-D)	40.0%	**52.7%**
		Fine-grained Pruning (0-D)	37.1%	55.7%

Mao, H., et al., Exploring the Regularity of Sparse Structure in Convolutional Neural Networks. NIPS Workshop 2017.

Outline

- Why We Need Pruning?
- Early Researches
- Optimization Solution: Structured Pruning
- **Rethinking Works for Pruning**
- Recent Works on Pruning

Some Rethinking Works – 1 (ISCA'18)

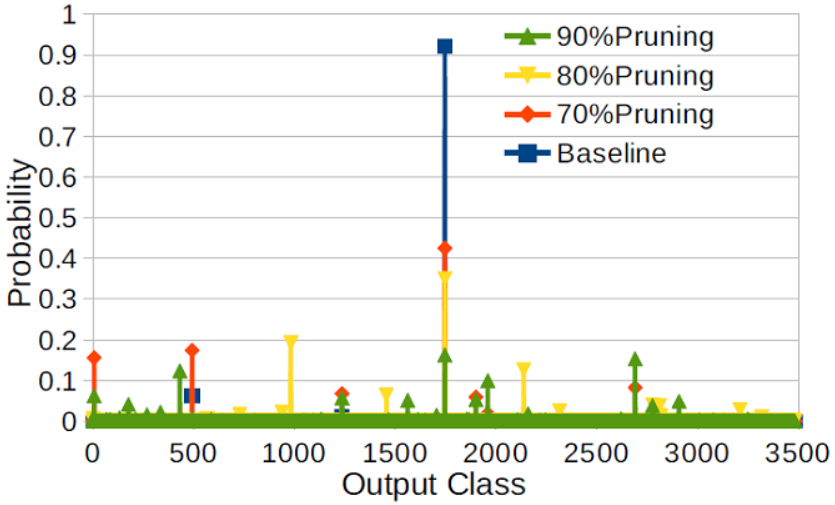


Fig. 1. Distribution of scores for a DNN for speech recognition and three pruned version at 70%, 80% and 90% of pruning. Although pruned models correctly identify top-1 class, the distribution of likelihoods is severely affected and confidence is largely reduced.

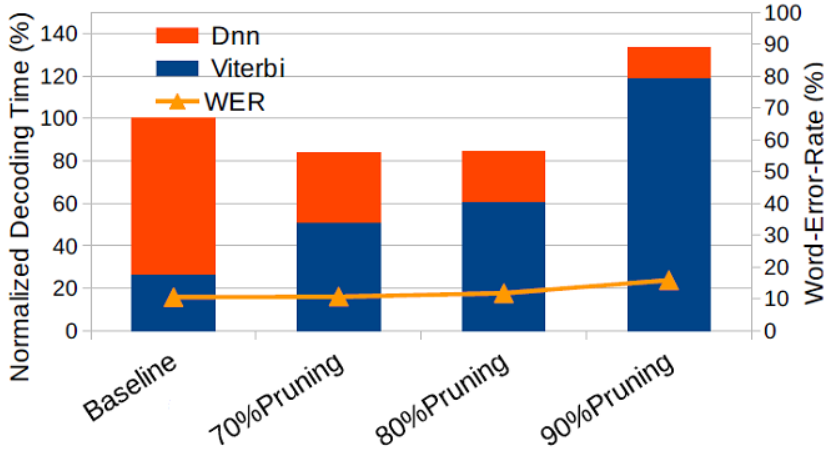


Fig. 2. Normalized execution time and Word Error Rate (WER) for a state-of-the-art ASR system, using a DNN with different degrees of pruning: 0% (Baseline), 70%, 80% and 90%. Pruning has a large impact on the execution time of the Viterbi beam search.

- Although top-1 accuracy may be maintained with DNN pruning, the likelihood of the class in the top-1 is significantly reduced when using the pruned models.
- For applications such as Automatic Speech Recognition (ASR), where the DNN scores are consumed by a successive stage, the workload of this stage can be dramatically increased due to the loss of confidence in the DNN.

Yazdani, R., et al., The Dark Side of DNN Pruning. ISCA 2018.

Some Rethinking Works – 2 (ICLR'19)

- Starting with a large model is not necessary and one could instead directly train the tiny model from scratch
- For these pruning algorithms, what matters is the obtained architecture, instead of the preserved weights

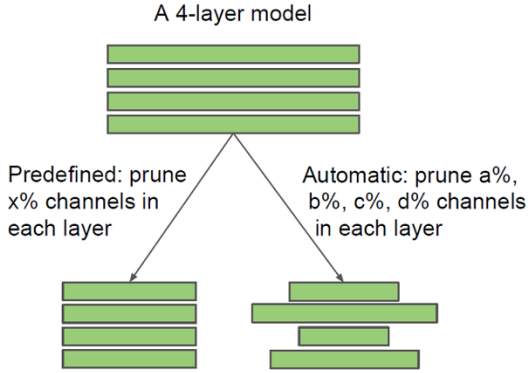


Figure 2: Difference between predefined and non-predefined (automatically discovered) target architectures. The sparsity x is user-specified, while a, b, c, d are determined by the pruning algorithm.

Dataset	Model	Unpruned	Pruned Model	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	VGG-16	93.63 (±0.16)	VGG-16-A	93.41 (±0.12)	93.62 (±0.11)	**93.78** (±0.15)
	ResNet-56	93.14 (±0.12)	ResNet-56-A	92.97 (±0.17)	92.96 (±0.26)	**93.09** (±0.14)
			ResNet-56-B	92.67 (±0.14)	92.54 (±0.19)	**93.05** (±0.18)
	ResNet-110	93.14 (±0.24)	ResNet-110-A	93.14 (±0.16)	**93.25** (±0.29)	93.22 (±0.22)
			ResNet-110-B	92.69 (±0.09)	92.89 (±0.43)	**93.60** (±0.25)
ImageNet	ResNet-34	73.31	ResNet-34-A	72.56	72.77	**73.03**
			ResNet-34-B	72.29	72.55	**72.91**

Table 1: Results (accuracy) for L_1-norm based channel pruning (Li et al., 2017). "Pruned Model" is the model pruned from the large model. Configurations of Model and Pruned Model are both from the original paper.

- **The value of automatic pruning algorithms may lie in identifying efficient structures and performing implicit architecture search, rather than selecting "important" weights.**

Liu, Z., et al., Rethinking the Value of Network Pruning. ICLR 2019.

Some Rethinking Works – 3 (ICLR'19)

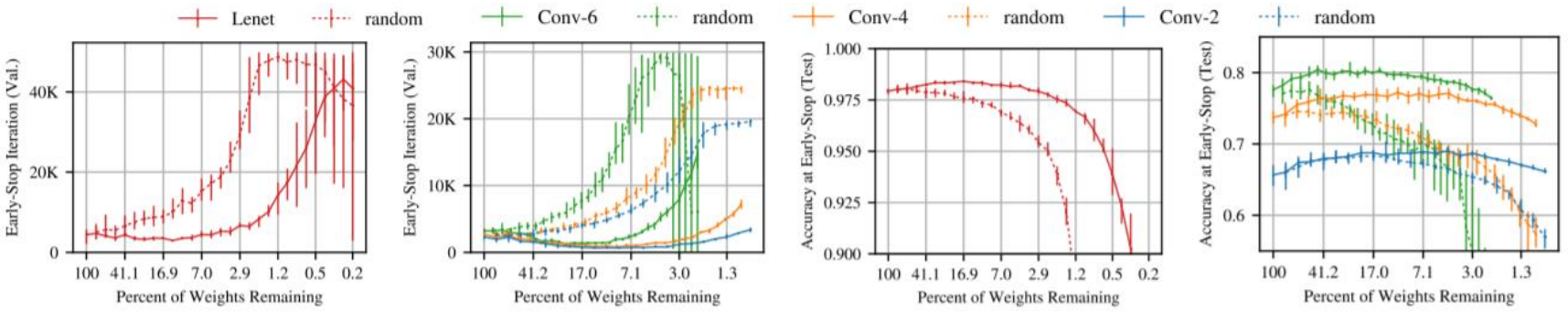


Figure 1: The iteration at which early-stopping would occur (left) and the test accuracy at that iteration (right) of the Lenet architecture for MNIST and the Conv-2, Conv-4, and Conv-6 architectures for CIFAR10 (see Figure 2) when trained starting at various sizes. Dashed lines are randomly sampled sparse networks (average of ten trials). Solid lines are winning tickets (average of five trials).

- the lottery ticket hypothesis: dense, randomly-initialized, feed-forward networks contain subnetworks (winning tickets) that—when trained in isolation—reach test accuracy comparable to the original network in a similar number of iterations.
- The winning tickets (solid lines in Fig.1) we find have won the initialization lottery: their connections have initial weights that make training particularly effective.

Outline

- Why We Need Pruning?
- Early Researches
- Optimization Solution: Structured Pruning
- Rethinking Works for Pruning
- **Recent Works on Pruning**

Recent Studies on Pruning – 1 (AAAI'20)

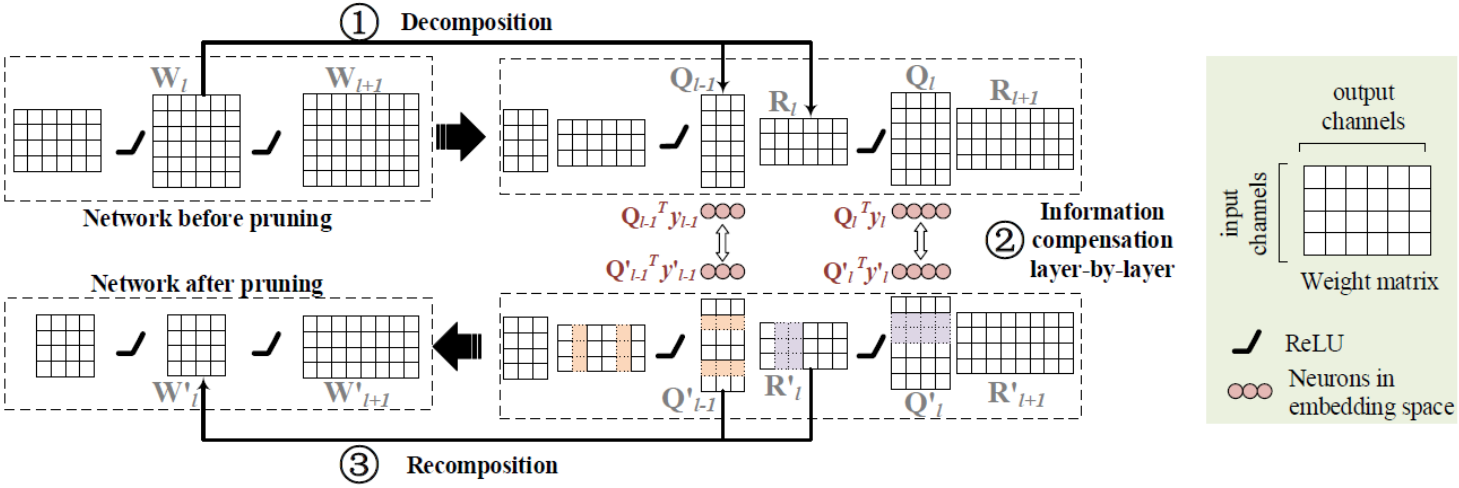
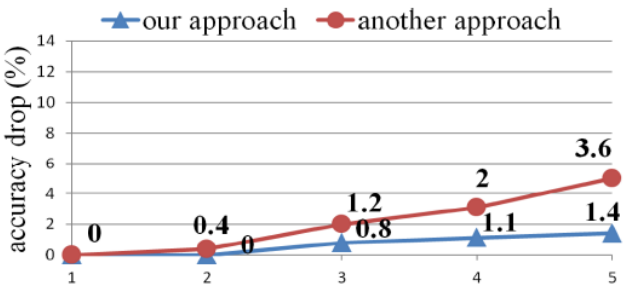
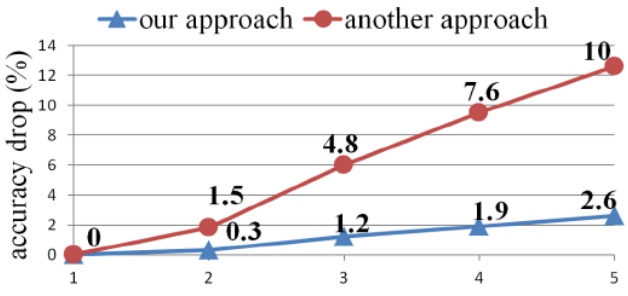


Figure 3: A Layer Decomposition-Recomposition Framework for neuron pruning. Layer decomposition is first applied to each layer to generate its embedding space. Then the redundant neurons are pruned, which is equivalent to prune the output channels of $\mathbf{R}'_l$ and the corresponding input channels of $\mathbf{Q}'_l$, while compensating the information loss of discarded neurons in each embedding space. It means one should optimize the pruned $\mathbf{R}'_l$ and $\mathbf{Q}'_l$ to force $\mathbf{Q}'^T_l \mathbf{y}'_l$ to approximate $\mathbf{Q}^T_l \mathbf{y}_l$. Ultimately, a slimmer network without depth increasing is returned through layer recomposition. (Best viewed in color)

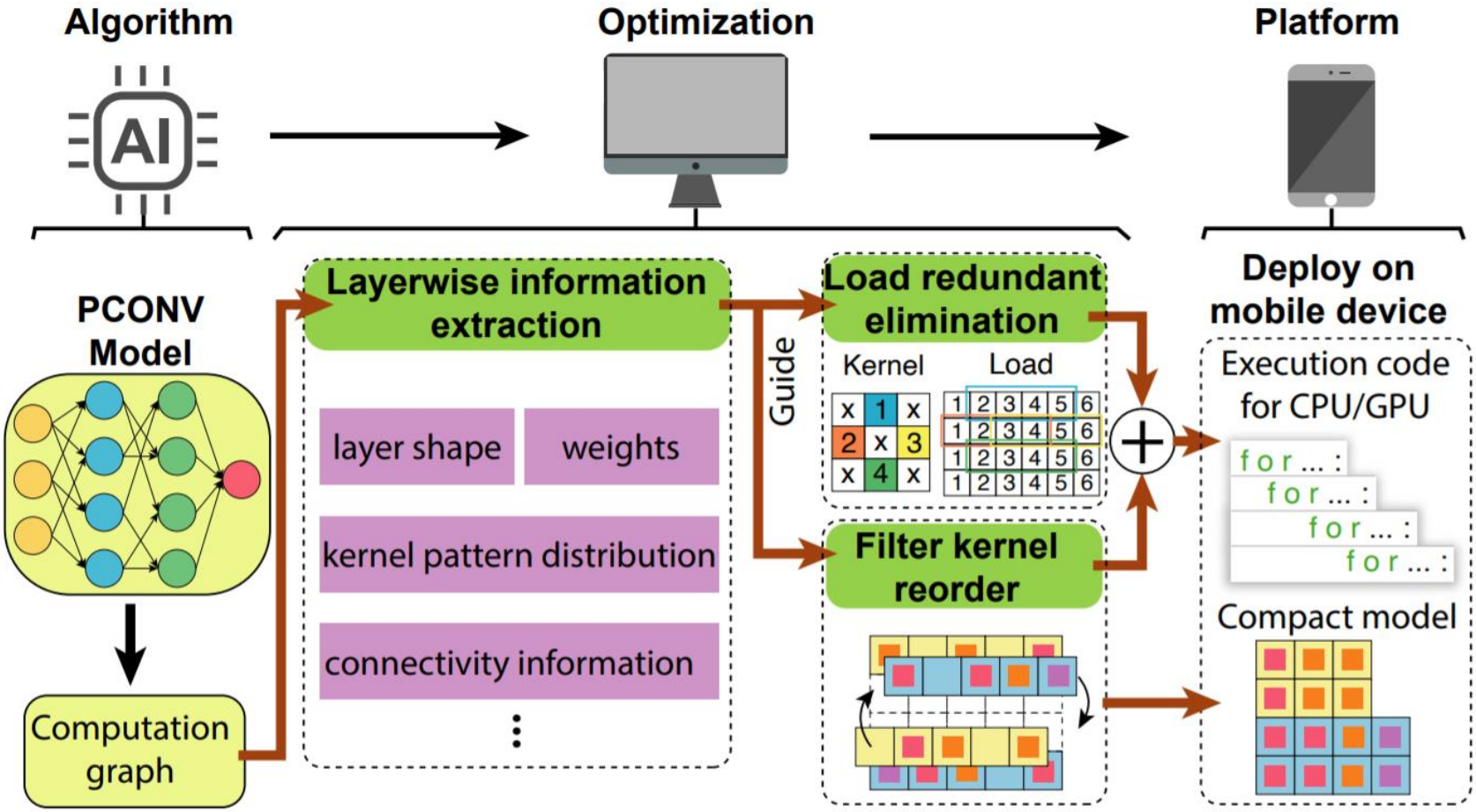


(b) after end-to-end fine-tuning



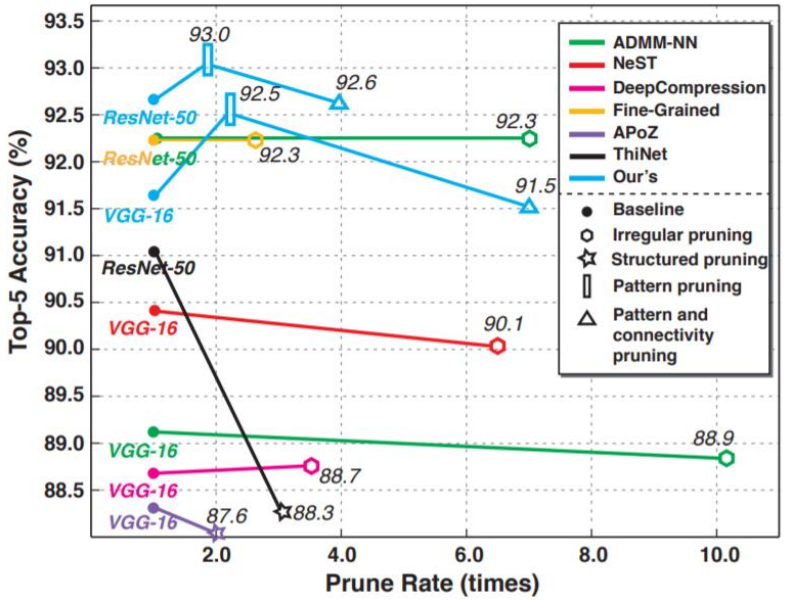
(a) before end-to-end fine-tuning

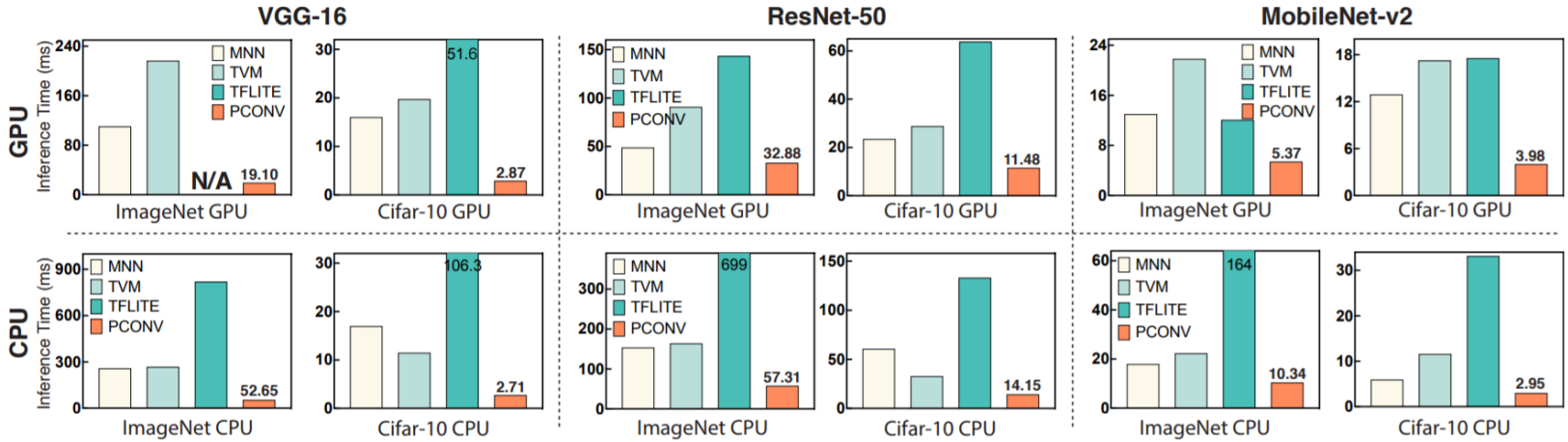
Recent Studies on Pruning – 2 (AAAI'20)



Ma, X., et al., PCONV: The Missing but Desirable Sparsity in DNN Weight Pruning for Real-time Execution on Mobile Devices. AAAI 2020.

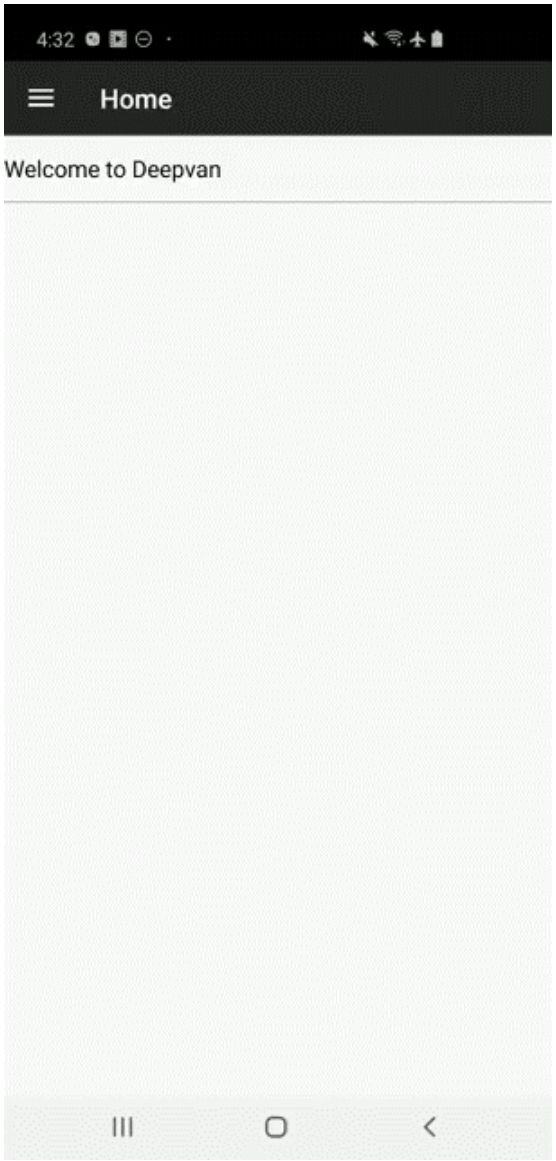
Recent Studies on Pruning – 2 (AAAI'20)





Ma, X., et al., PCONV: The Missing but Desirable Sparsity in DNN Weight Pruning for Real-time Execution on Mobile Devices. AAAI 2020.

Recent Studies on Pruning – 2 (AAAI'20)



Ma, X., et al., PCONV: The Missing but Desirable Sparsity in DNN Weight Pruning for Real-time Execution on Mobile Devices. AAAI 2020.

Recent Studies on Pruning – 3 (DAC'20)

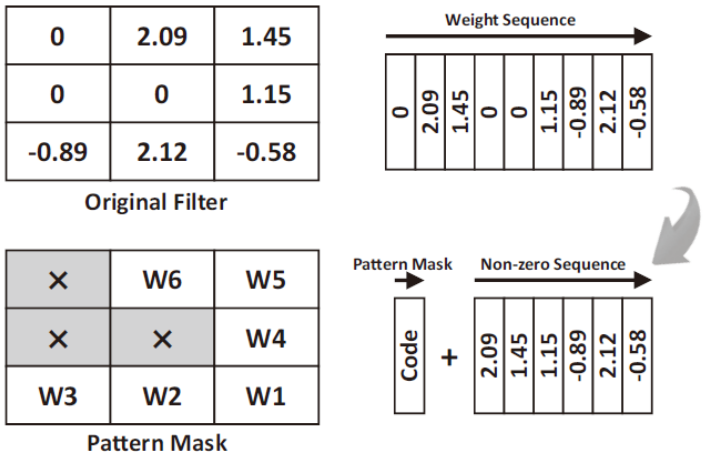


Fig. 1: Top: the original representation for a kernel. Bottom: pattern-based representation using an SPM index and the non-zero sequence.

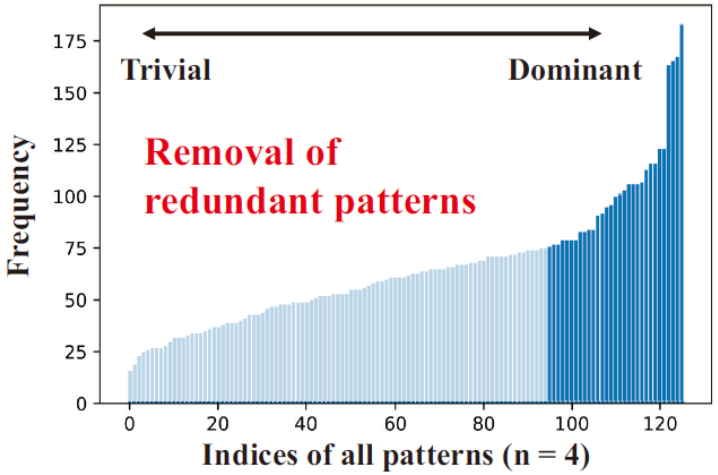


Fig. 2: Pattern distribution in CONV4 of VGG-16, where the number of non-zero is 4. Thus there are 126 patterns in total.

Benchmark	Top1 acc	Top1 acc Loss	CONV FLOPs	FLOPs Pruned	CONV Parameters	Compression (weight)	Compression (weight+idx)
VGG-16, Baseline	93.54%	-	3.13×10^8	-	1.47×10^7	-	-
VGG-16, n = 4	93.79%	+0.25%	1.39×10^8	56.5%	0.65×10^7	2.3×	2.2×
VGG-16, n = 3	93.58%	+0.04%	1.04×10^8	66.7%	0.49×10^7	3.0×	2.9×
VGG-16, n = 2	93.52%	-0.02%	0.30×10^8	77.8%	0.33×10^7	4.5×	4.1×
VGG-16, n = 1	93.07%	-0.21%	0.35×10^8	88.9%	0.16×10^7	9.0×	8.4×
VGG-16, Various setting[a]	93.33%	-0.21%	0.35×10^8	88.8%	0.16×10^7	9.0×	8.4×

[a] n in various layers: 2-1-1-1-1-1-1-1-1-1-1-1-1 with 32 patterns in n = 2 layers and 8 patterns in n = 1 layers.

TABLE I: Pruning rate and accuracy of different n for VGG-16 on CIFAR10.

Tan, Z., et al., PCNN: Pattern-based Fine-Grained Regular Pruning towards Optimizing CNN Accelerators. DAC 2020.

Recent Studies on Pruning – 4 (ICCV'19)

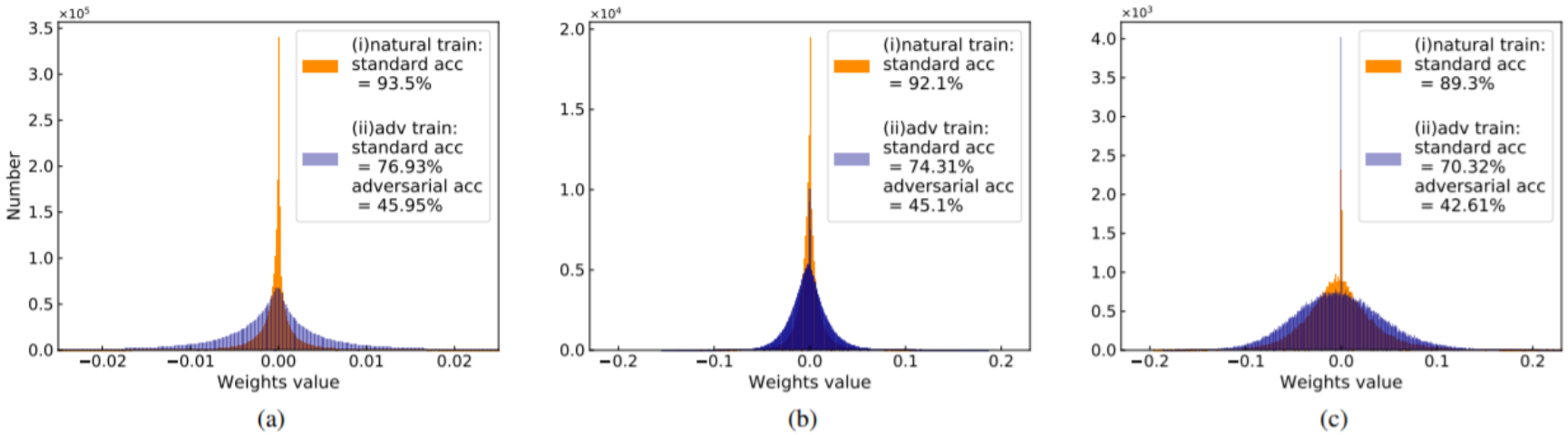


Figure 1: Weight distribution of VGG-16 network with (a) original size, (b) 1/2 size, and (c) 1/4 size on CIFAR dataset. For each size in one subfigure, the weights are characterized for (i) a naturally trained model and (ii) an adversarially trained model. The standard accuracy and adversarial accuracy are marked with the legend.

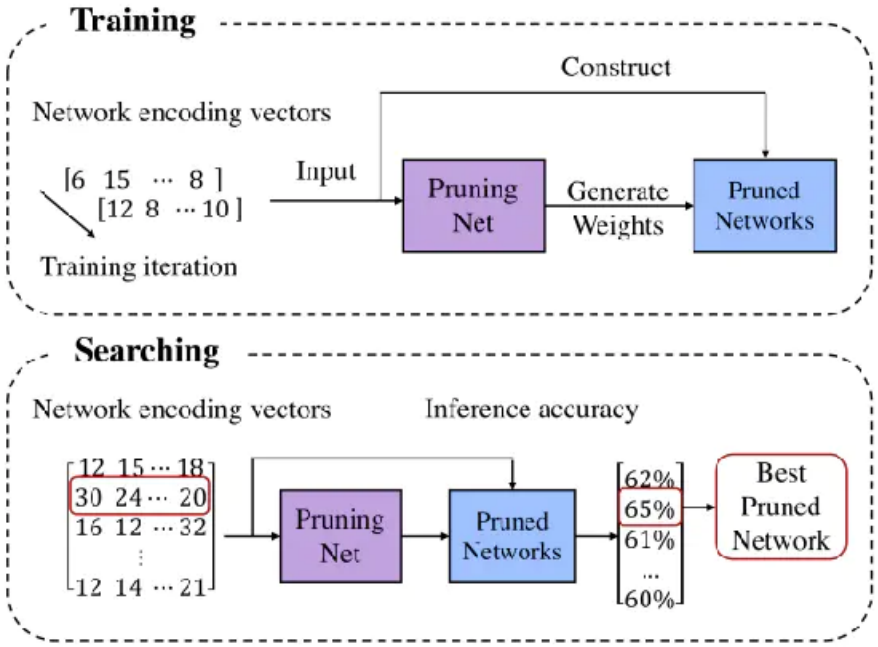
Table 4: **Natural test accuracy/adversarial test accuracy** (in %) on **CIFAR10 by ResNet** of [column ii] naturally trained model with different size w, [column iii] adversarially trained model with different size w, [columns iv–vii] concurrent adversarial training and weight pruning from a large size to a small size.

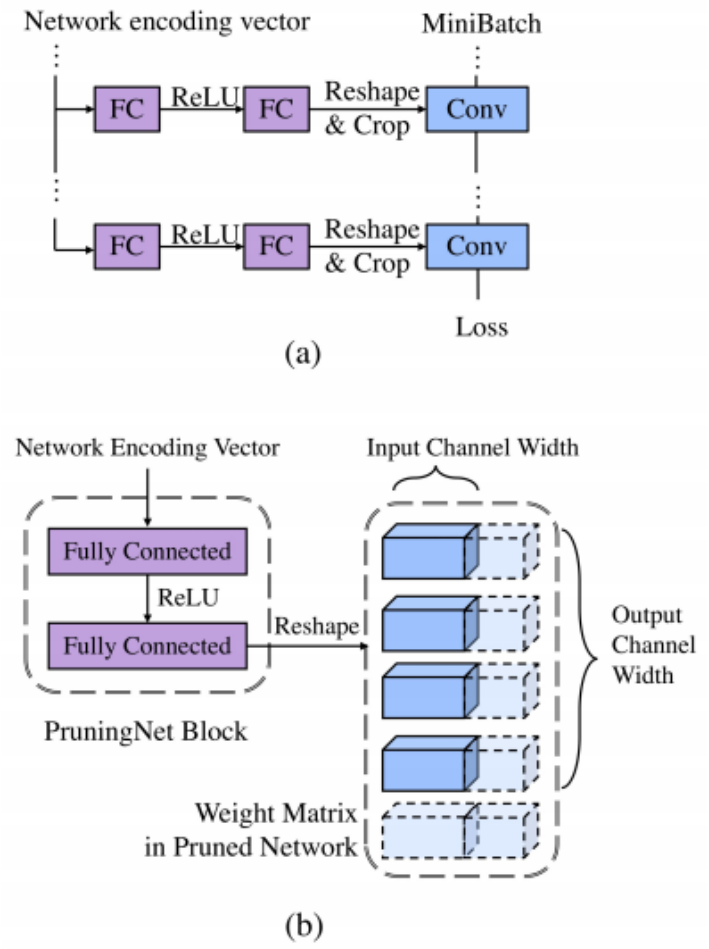
w	nat baseline	adv baseline	1	2	4	8
1	84.23/0.00	57.16/34.40	-	-	-	-
2	87.05/0.00	71.16/42.45	64.53/37.90	-	-	-
4	91.93/0.00	77.35/44.99	64.36/37.78	73.21/43.14	-	-
8	93.11/0.00	77.26/47.28	**64.52/38.01**	**73.36/43.17**	78.12/45.49	-
16	94.80/0.00	82.71/49.31	64.17/37.99	71.80/42.86	**78.85/47.19**	**81.83/48.00**

Ye, S., et al., Adversarial Robustness vs. Model Compression, or Both?. ICCV 2019.

Recent Studies on Pruning – 5(ICCV'19)

Learn a Meta Neural Network to Predict the Wegihts in CNN





First train a Meta NN to predict weights in CNN.

Then search the best architecture with weights predicted by Meta NN.

Recent DeepSeek NAS

NSA的A是attention的缩写。

attention是transformer网络中的核心计算模块。

原生attention的公式参见截图，可不需要深入理解：

- 先基于输入和模型权重，计算出query，key，value这3个矩阵。
- 对query，key，value进行矩阵乘和softmax运算，得到输出结果。

In practice, we compute the attention function on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V. We compute the matrix of outputs as:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \quad (1)$$

attention的计算量，以及中间结果的存储需求，均跟输入token数量的平方成正比。因此越长的输入序列，需要消耗高得多的算力，以及读写开销。

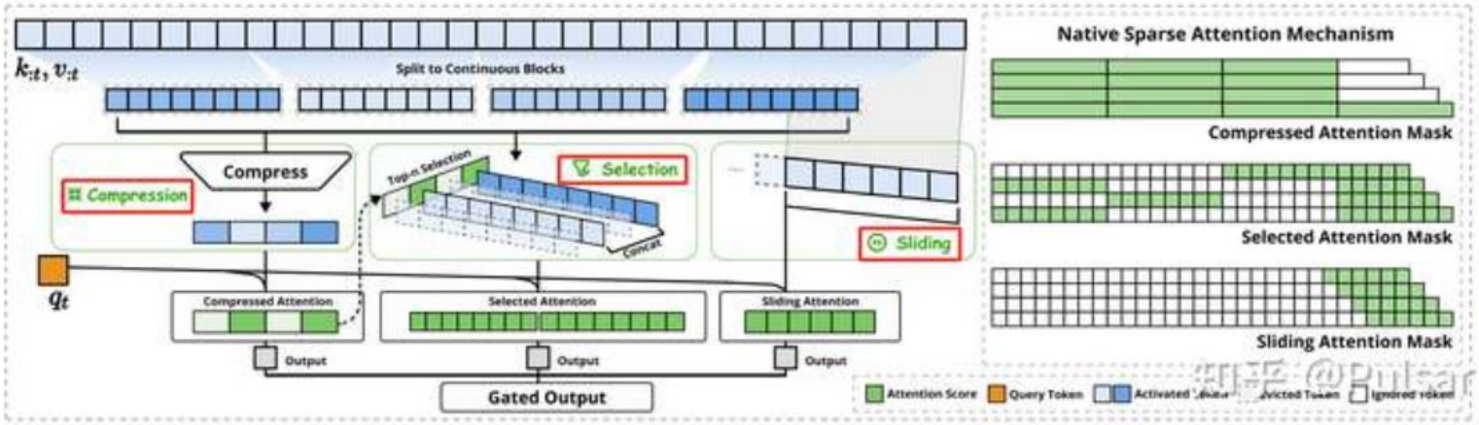
transformer的推理（inference）分为2个阶段：

- prefill：通常为计算瓶颈，硬件算力可以充分发挥。
- decode：通常算力密度低，瓶颈是内存读写，硬件算力无法充分发挥。

Source：Deepseek，知乎Pulsar

Recent DeepSeek NAS

- compress（压缩）
- selection（选择）
- sliding（滑窗）



原生attention和NSA的计算公式，如下。

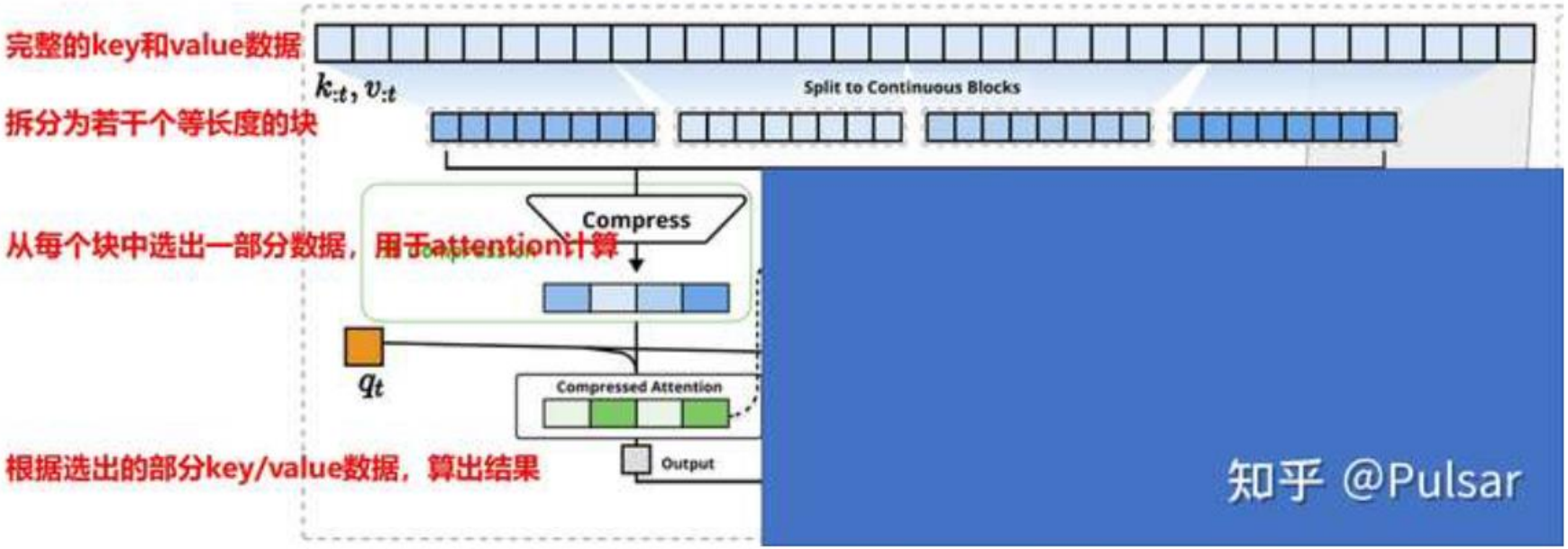
Attention Mechanism is widely used in language modeling where each query token $\mathbf{q}_t$ computes relevance scores against all preceding keys $\mathbf{k}_{:t}$ to generate a weighted sum of values $\mathbf{v}_{:t}$. Formally, for an input sequence of length t, the attention operation is defined as:

原生attention $$\mathbf{o}_t = \mathrm{Attn}\left(\mathbf{q}_t, \mathbf{k}_{:t}, \mathbf{v}_{:t}\right) \quad (1)$$

Source：Deepseek

Recent DeepSeek NAS

- 从完整的key/value数据中，选出部分数据，进行一些处理（矩阵乘）后，计算attention结果。



公式如下，非常简洁：

- 拆分出长度为 d 的块。
- 每个数据块，只挑选出前 l 个数据

selection

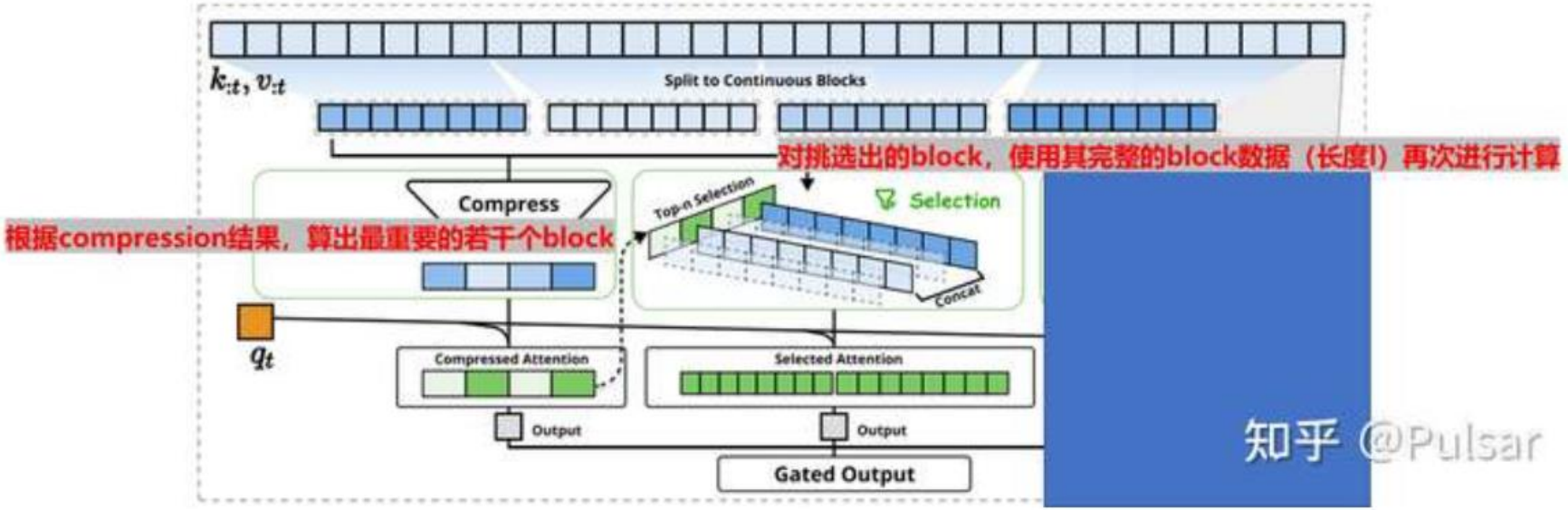
可以看出，compress就相当于对数据进行一个等间距的采样。

由于逻辑过于简单，难免会丢失很多细颗粒度的信息。

因此，NSA基于compression的计算结果，选出"最重要"的若干个block，然后载入其完整的block数据，进行计算。

Source：Deepseek

Recent DeepSeek NAS



先总结一下这两个步骤：

- **compression：使用所有块的数据，但每块数据不完整。**
- **selection：使用部分块的数据，但每块数据都是完整的。**

Source：Deepseek

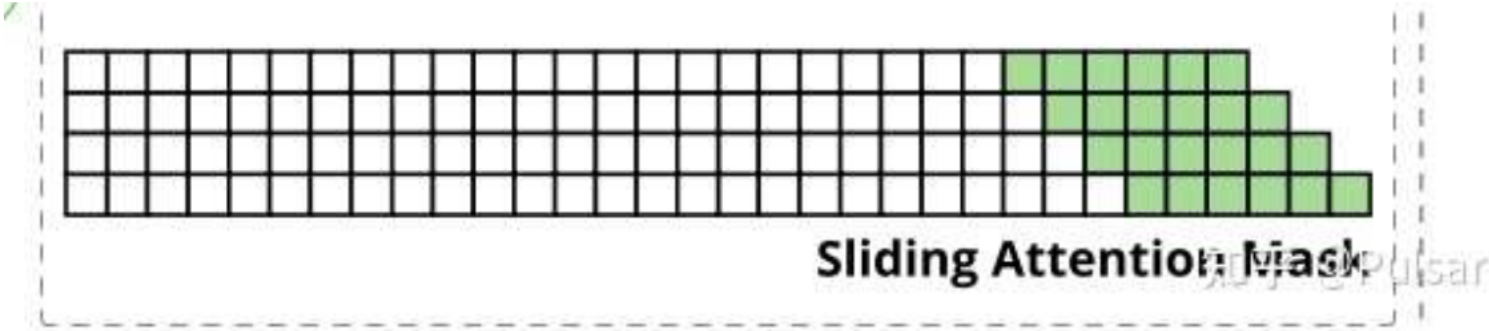
sliding

该步骤跟前2步compression和selection无关，可以并行执行。

论文给的例子就是，直接选出若干个最新生成的key和value。

on learning their respective features without being shortcutted by local patterns. Specifically, we maintain recent tokens $\tilde{K}_t^{\text{win}} = \mathbf{k}_{t-w:t}, \tilde{V}_t^{\text{win}} = \mathbf{v}_{t-w:t}$ in a window w, and isolate attention computations of different information sources (compression tokens, and selected tokens, sliding window) into separate branches. These branch outputs are then aggregated through a learned

如图所示，key和value是按照从左到右的顺序生成的，sliding步骤只选用最靠后生成的一部分数据。



结束3个步骤后，就可以根据公式(5)，将3步的计算结果加权求和，得到最终的输出。

$$\mathbf{o}_t^* = \sum_{c \in C} g_t^c \cdot \text{Attn}(\mathbf{q}_t, \tilde{K}_t^c, \tilde{V}_t^c). \tag{5}$$

As illustrated in Figure 2, NSA have three mapping strategies $C = \{\text{cmp}, \text{slc}, \text{win}\}$, representing

论文给出的结果非常惊艳，甚至于反直觉：

NSA相比于原生attention，不仅速度翻了数倍，而且**【性能还增加了】**。

由于性能的提升，显得有些不可思议，所以论文原话也有点收着，表示NSA【保持或许超过】了原生attention（NSA maintains or exceeds Full Attention models）。

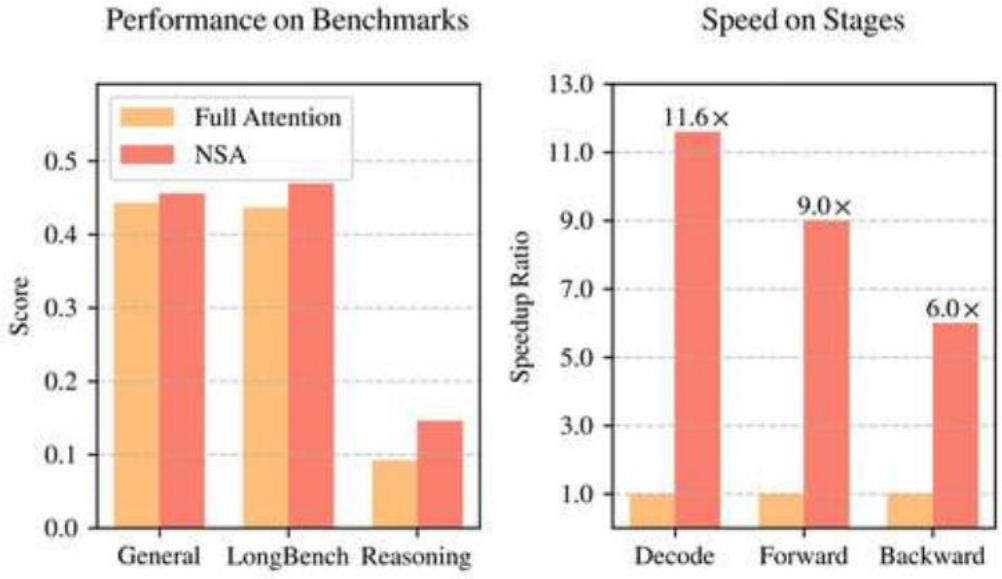


Figure 1 | Comparison of performance and efficiency between Full Attention model and our NSA. Left: Despite being sparse, NSA surpasses Full Attention baseline on average across general benchmarks, long-context tasks, and reasoning evaluation. Right: For 64k-length sequence processing, NSA achieves substantial computational speedup compared to Full Attention in all stages: decoding, forward propagation, and backward propagation.

Source：Deepseek

Recent DeepSeek NAS

Source: Deepseek

Recent DeepSeek NAS

Source: Deepseek

Recent DeepSeek NAS

Source: Deepseek

Recent DeepSeek NAS

Source: Deepseek