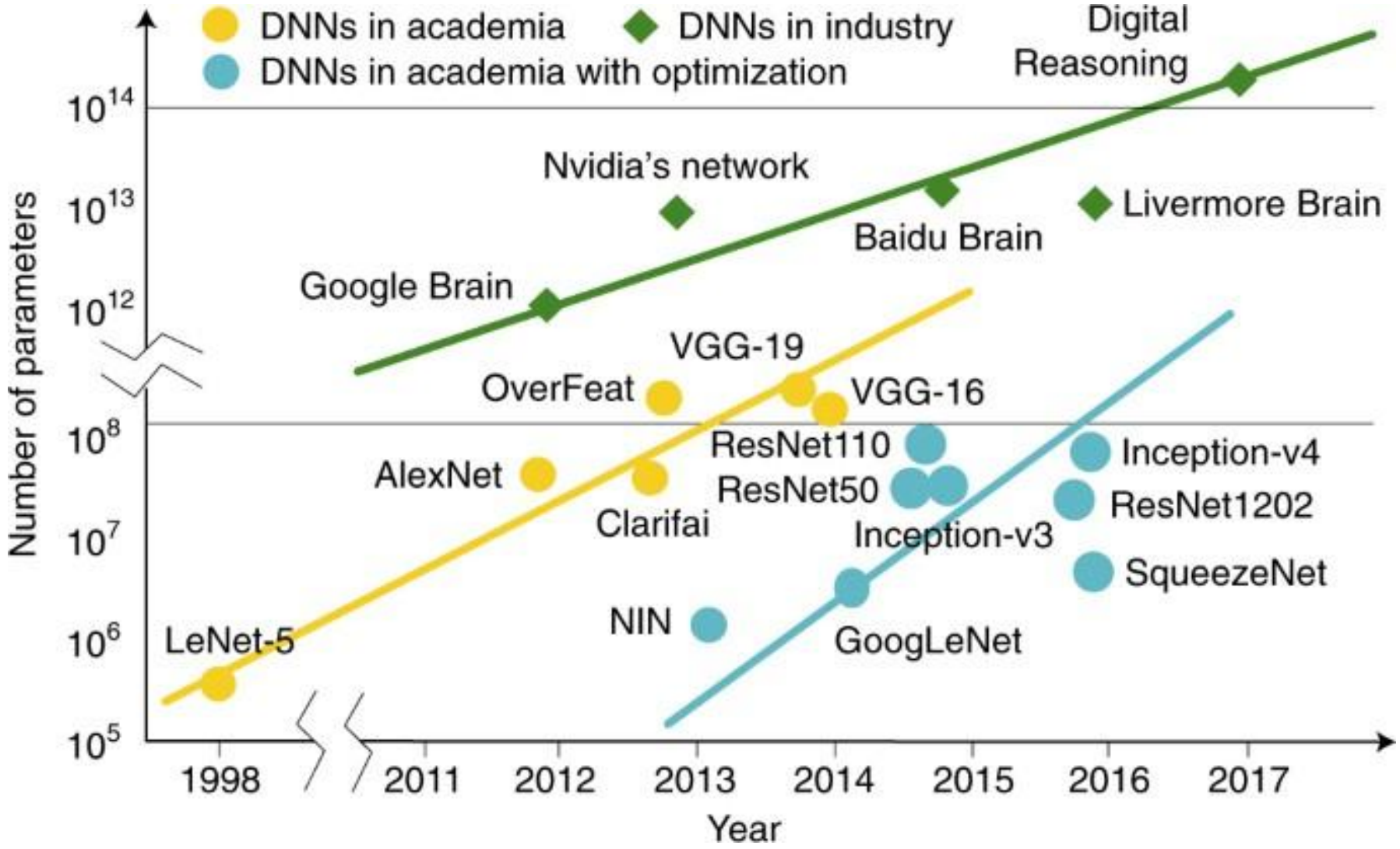
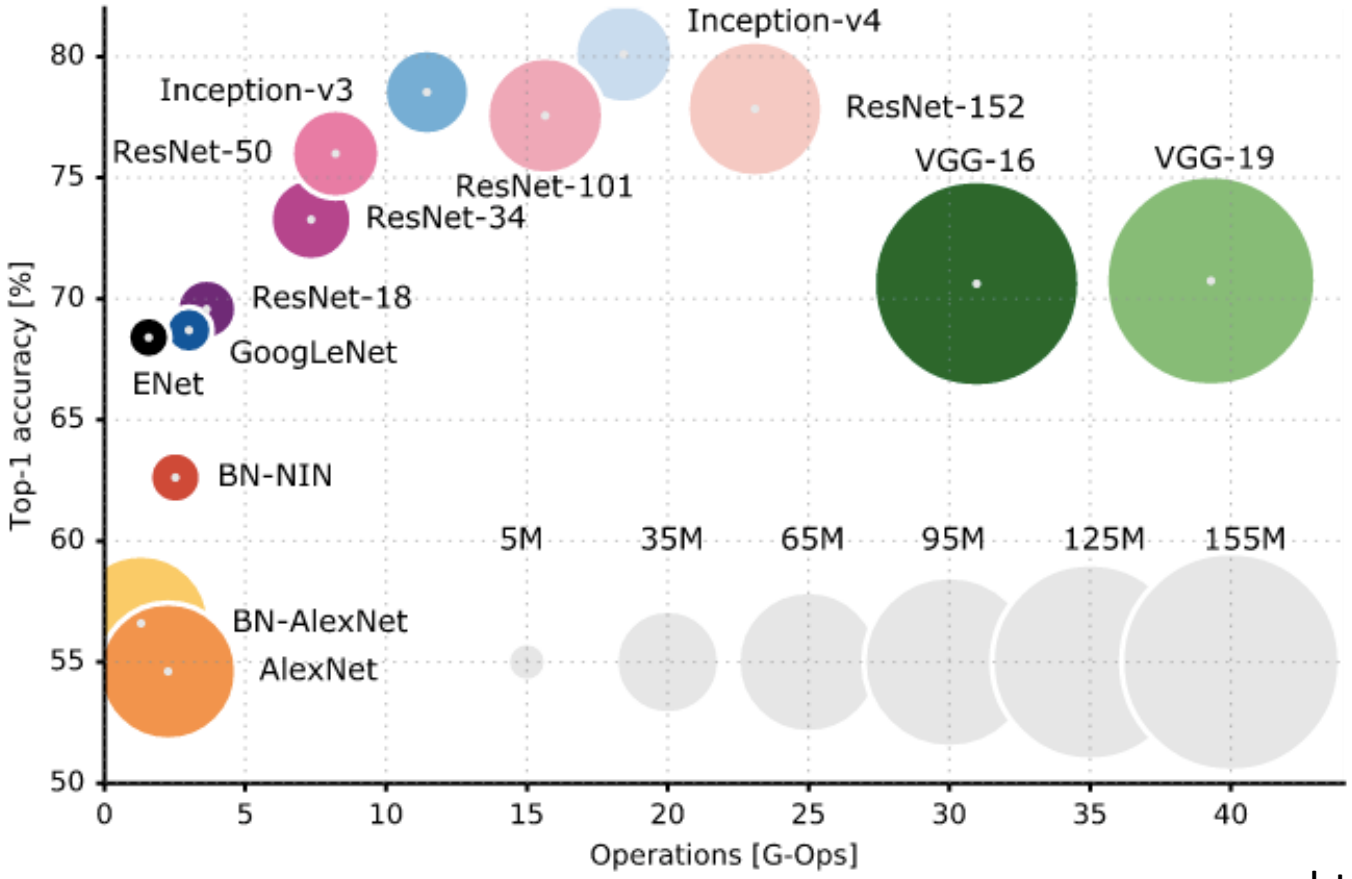
Lecture 7 Model Acceleration

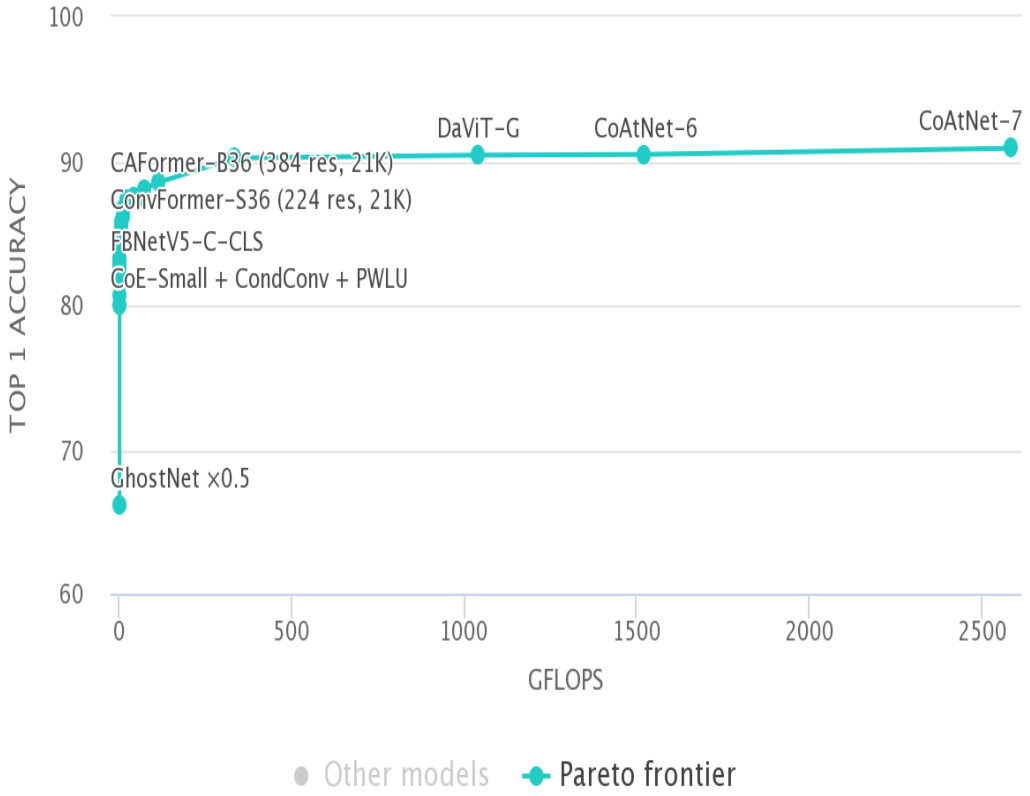
Tian Sheuan Chang

Why Compression?



DL Models Are Getting Larger and More Complex

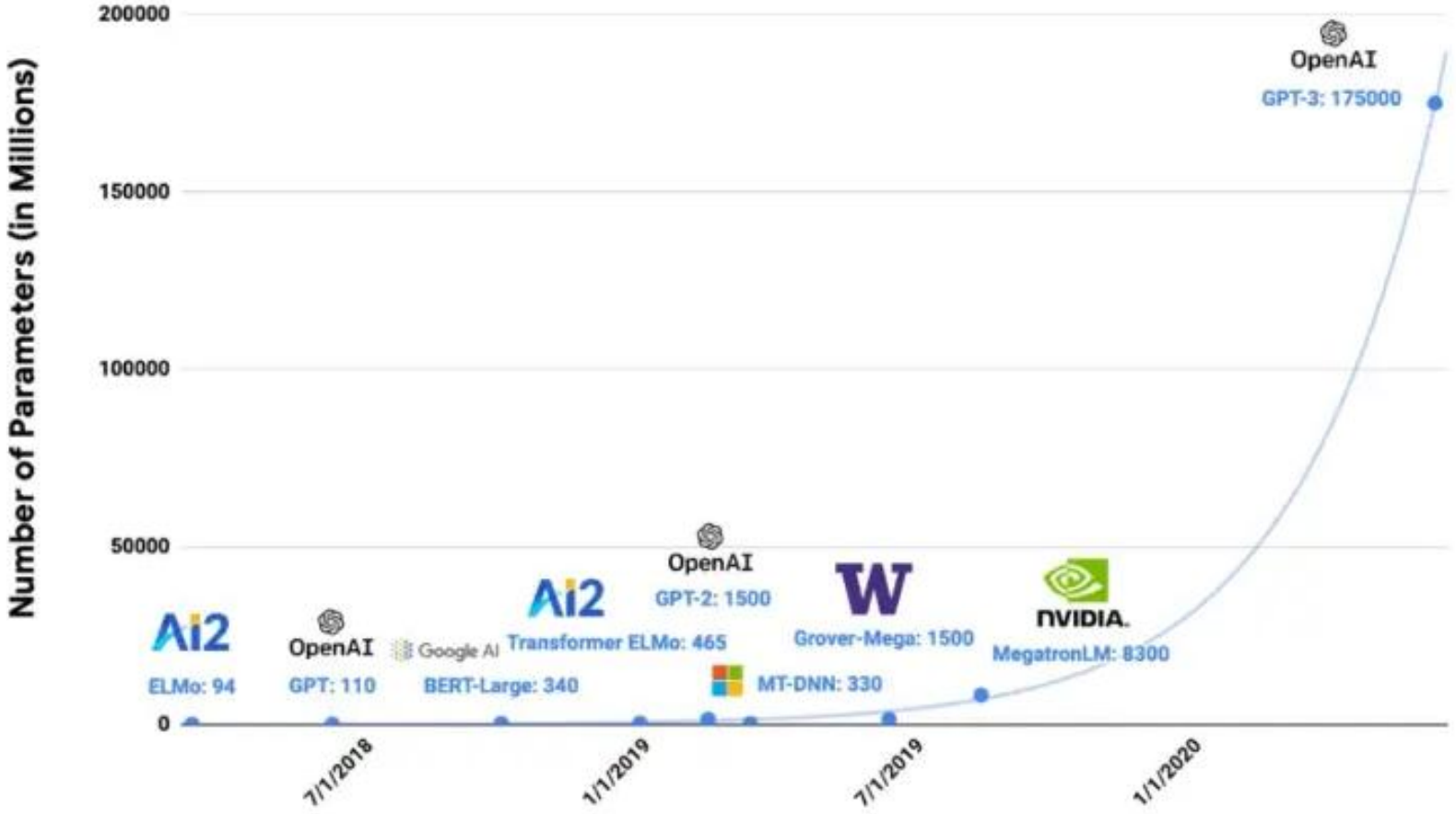




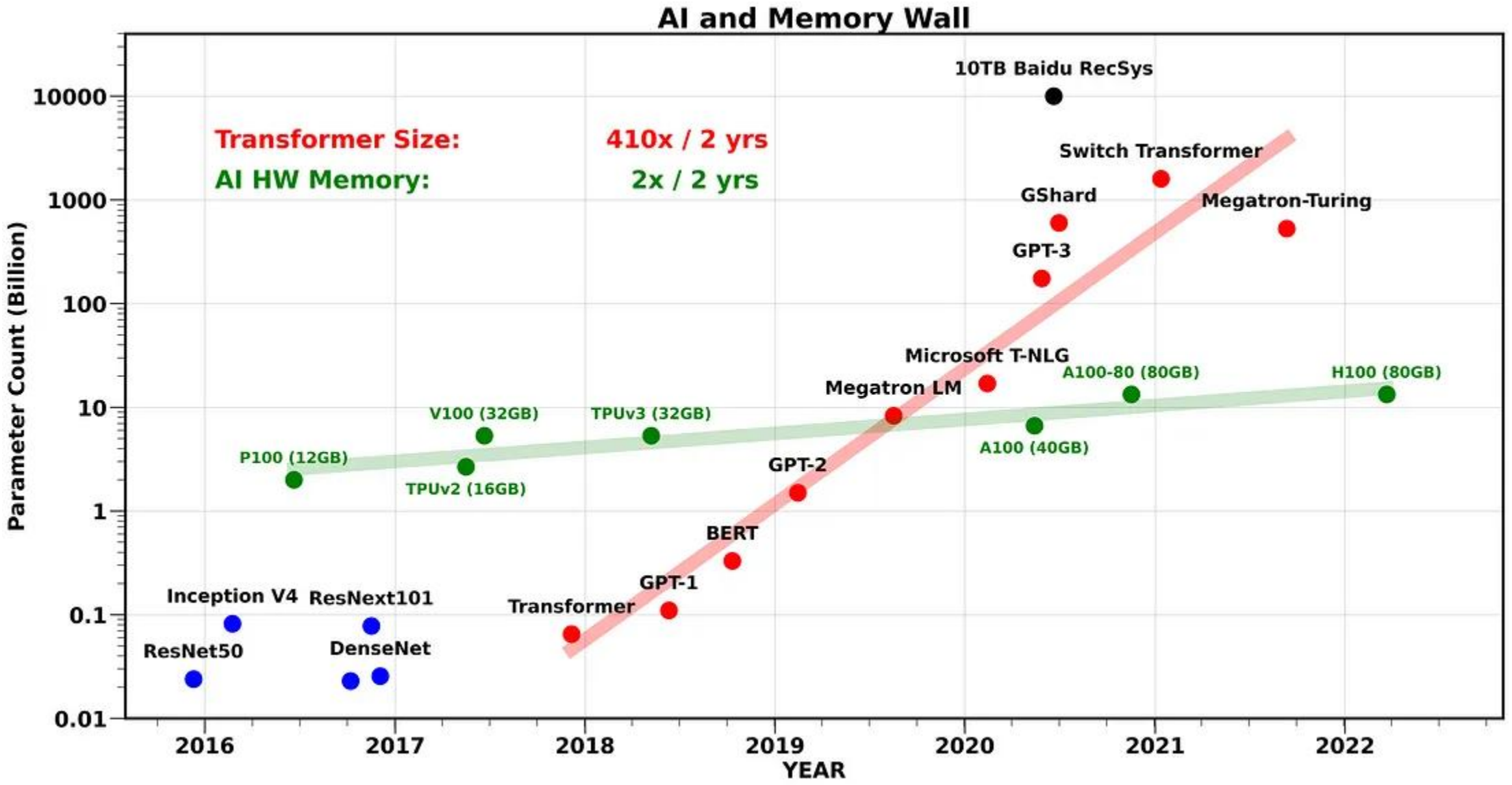
https://culurciello.github.io/tech/2016/06/04/nets.html

https://paperswithcode.com/sota/image-classification-on-imagenet?dimension=GFLOPs

NLP model size is increasing exponentially



source: EdX Fundamentals of TinyML


AI and Memory Wall
Transformer Size:
410x / 2 yrs
AI HW Memory:
2x / 2 yrs
Parameter Count (Billion)
10000
1000
100
10
1
0.1
0.01
2016
2017
2018
2019
2020
2021
2022
YEAR
10TB Baidu RecSys
Switch Transformer
GShard
Megatron-Turing
GPT-3
Microsoft T-NLG
Megatron LM
A100-80 (80GB)
H100 (80GB)
V100 (32GB)
TPUv3 (32GB)
P100 (12GB)
TPUv2 (16GB)
A100 (40GB)
GPT-2
BERT
GPT-1
Transformer
Inception V4
ResNext101
ResNet50
DenseNet

Challenges of Large Model Size

- Real time execution

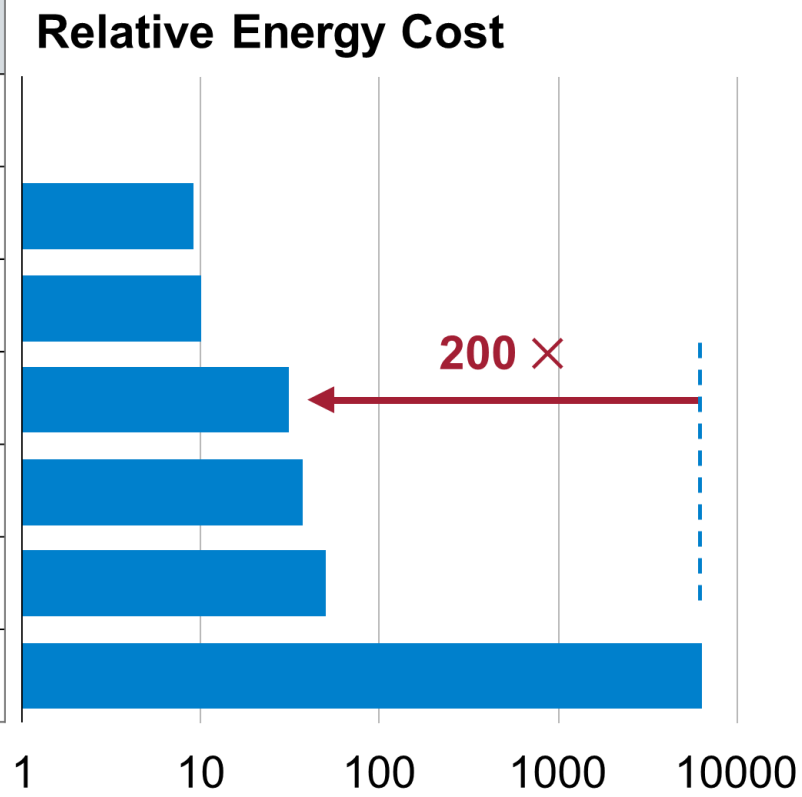
- Power consumption

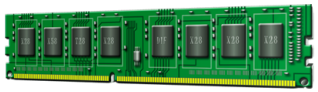
Challenge of Energy Efficiency

larger model => more memory reference => more energy

Operation	Energy [pJ]
32 bit int ADD	0.1
32 bit float ADD	0.9
32 bit Register File	1
32 bit int MULT	3.1
32 bit float MULT	3.7
32 bit SRAM Cache	5
32 bit DRAM Memory	640

Rough Energy Cost For Various Operations in 45nm 0.9V



1  = 200 ×+

This image is in the public domain

how to make deep learning more efficient?

Computing's Energy Problem (and What We Can Do About it) [Horowitz, M., IEEE ISSCC 2014]

Outline

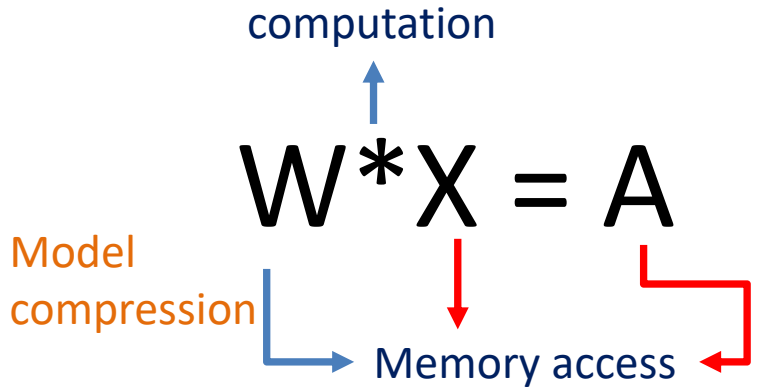
Static model acceleration	Smaller: fewer weights	Pruning: weights, vector, filter, channel
		unstructured vs structured
	Lower: quantization - from full precision to binary precision	Low precision, weight sharing
		Ternary and binary weights+activations
	Less computation	Low rank approximation
		Winograd transformation
	low complexity model	Mobilenet/EfficientNet
		MCUNet for MCU

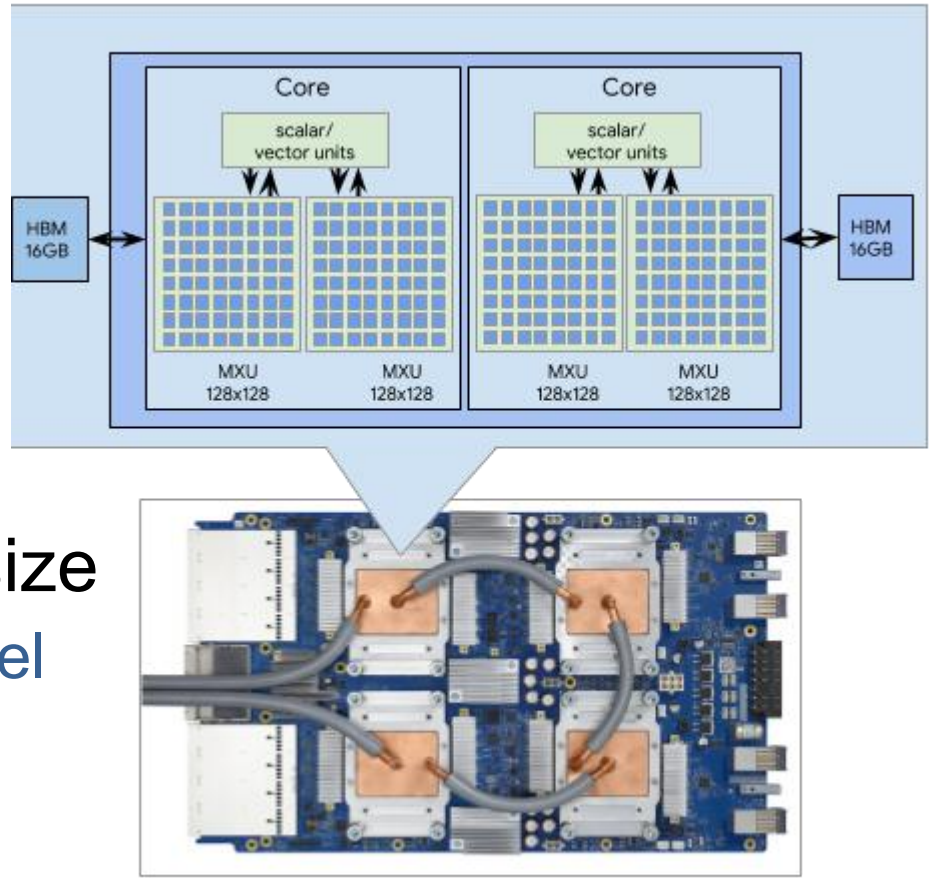
Outline

Dynamic execution based on input (run time)	SBNet	各種加速法與背後的的可行性原理 神經網路具有高容錯性與可塑性
	Dynamic Network Slimming	
Knowledge distillation	To improve or restore accuracy of compressed model	

From Model Compression to Acceleration

Model execution time





TPU v3 - 4 chips, 2 cores per chip

- Model compression gets smaller model size
  - But smaller model does not imply faster model execution
- Execution platform dependent impact
  - DRAM access
  - Computation parallelism

From Model Compression to Acceleration

Pruning

- Unstructured
  - +HW+SW
  - 不一定變快
  - Few vendors
  - Accuracy loss
- Structured
  - 高機率變快
  - Accuracy loss

Quantization

- 8/16 b
  - +HW/SW, Mainstream vendors
  - 高機率變快
  - Accuracy loss ~=0
- <8 b
  - +HW+SW
  - 不一定變快
  - Accuracy loss

Low complexity model

- Depthwise
  - +HW+SW
  - Low complexity/high memory access layer
  - 不一定變快
  - Accuracy loss
- Others
  - 高機率變快
  - Accuracy loss

Tools: Profiling

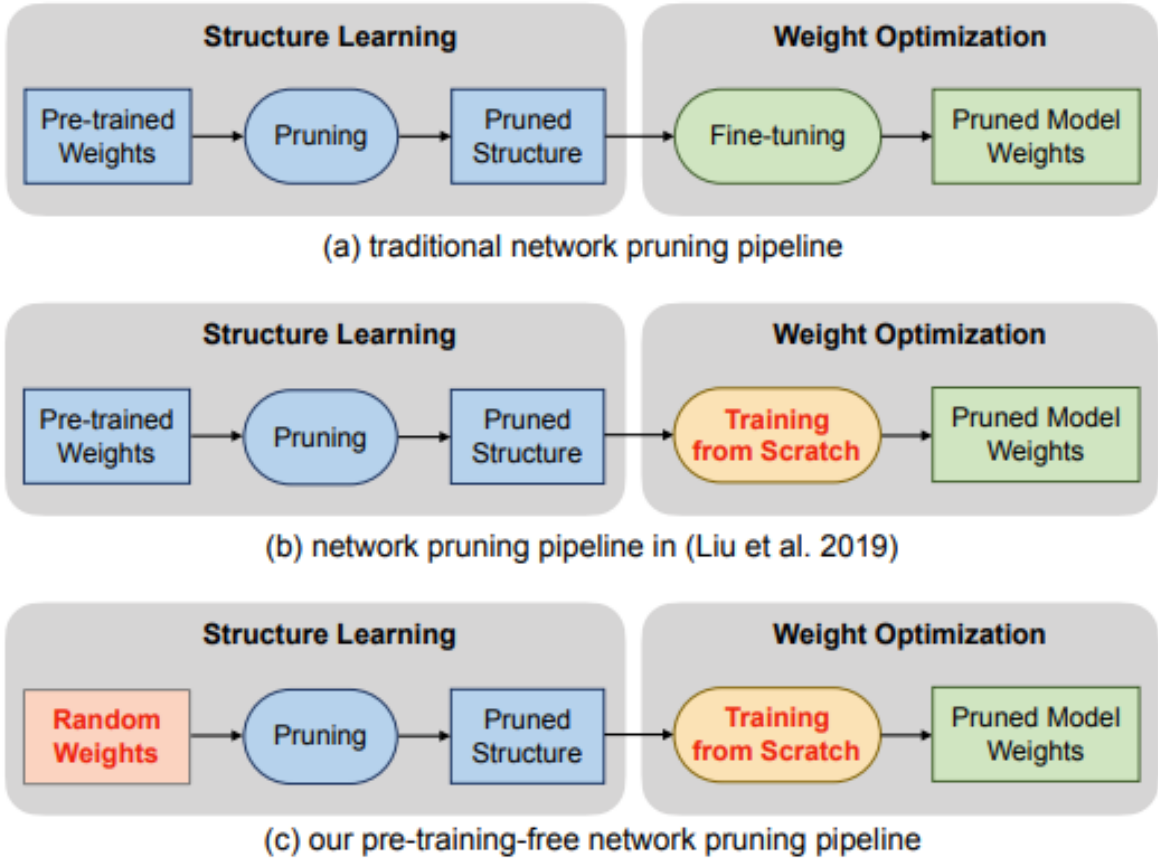
```
depth 6:
    params      - {'LinearActivation': '100.76 M', 'Linear': '100.69 M'}
    MACs        - {'LinearActivation': '1030.79 GMACs', 'Linear': '1030.79 GMACs'}
    fwd latency - {'BertSelfAttention': '16.29 ms', 'LinearActivation': '3.48 ms'}

------------------------------ Detailed Profile per GPU ------------------------------
Each module profile is listed after its name in the following order:
params, percentage of total params, MACs, percentage of total MACs, fwd latency, percentage of total fw

BertForPreTrainingPreLN(
  336.23 M, 100.00% Params, 3139.93 GMACs, 100.00% MACs, 76.39 ms, 100.00% latency, 82.21 TFLOPS,
  (bert): BertModel(
    335.15 M, 99.68% Params, 3092.96 GMACs, 98.50% MACs, 34.29 ms, 44.89% latency, 180.4 TFLOPS,
    (embeddings): BertEmbeddings(...)
    (encoder): BertEncoder(
      302.31 M, 89.91% Params, 3092.88 GMACs, 98.50% MACs, 33.45 ms, 43.79% latency, 184.93 TFLOPS,
      (FinalLayerNorm): FusedLayerNorm(...)
      (layer): ModuleList(
```

https://www.deepspeed.ai/tutorials/flops-profiler/

PRUNING: FEWER WEIGHTS
從既有MODEL開始，先把MODEL 縮小一點

Outline

- Unstructured pruning
- Structured pruning
- Pruning as a architecture search
- Resource constrained pruning
- Pruning from scratch

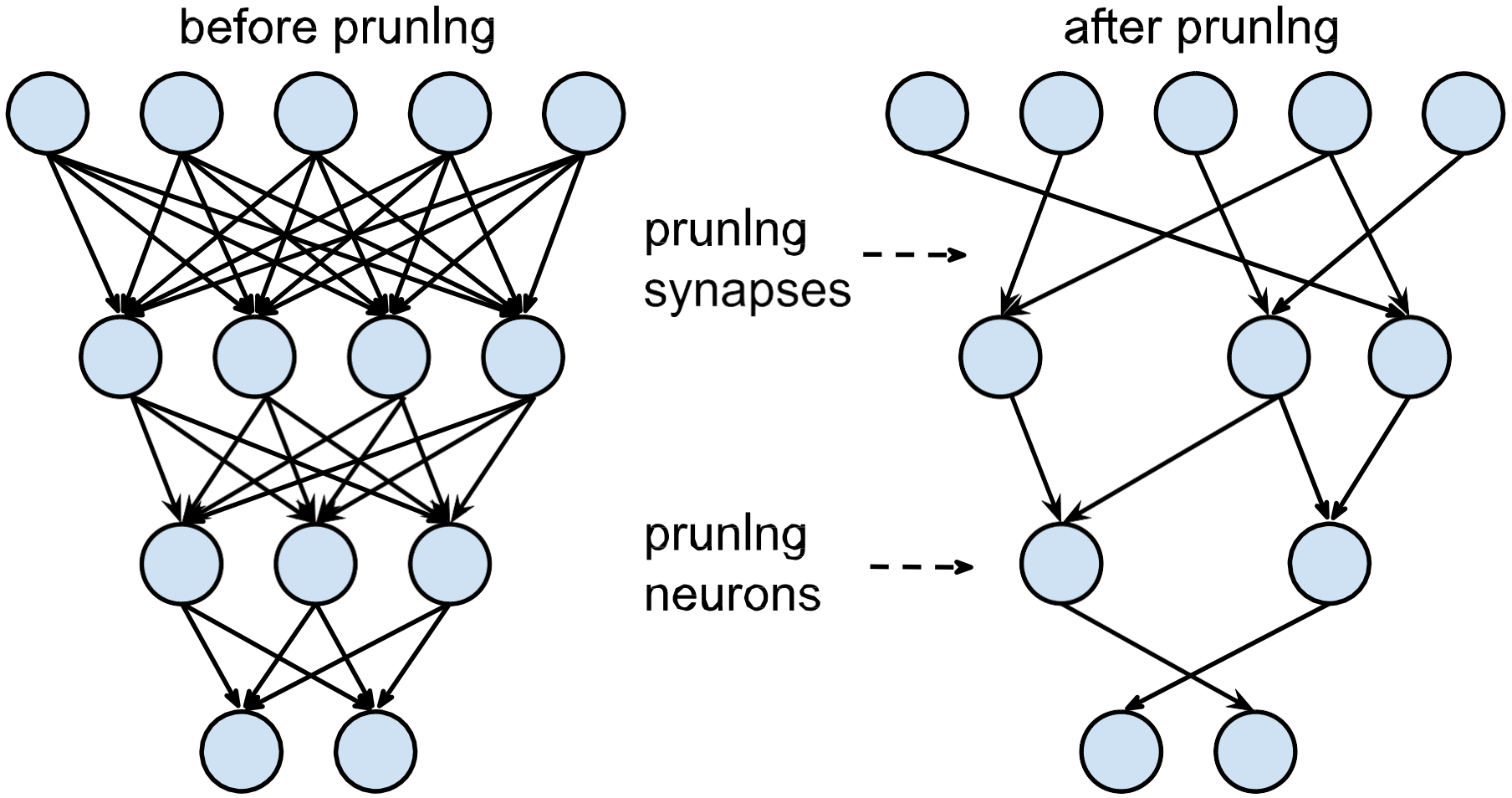


Reference

- https://github.com/he-y/Awesome-Pruning
- Structured Pruning for Deep Convolutional Neural Networks: A survey
- https://github.com/neuralmagic/sparsezoo
  - pruned and pruned-quantized models

Pruning

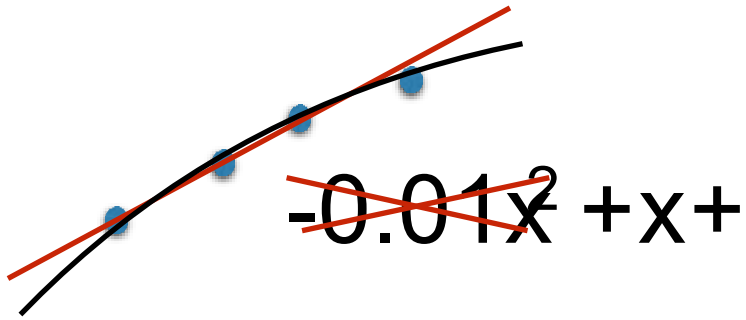
- Concept: Not all weights are created equal importance

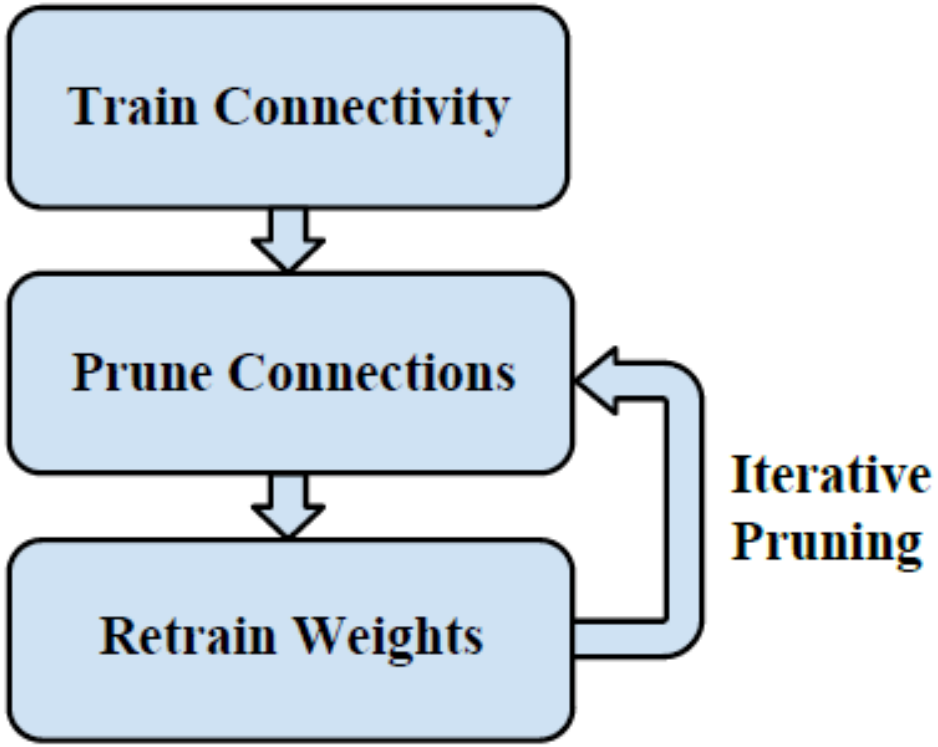


Han et al. Learning both Weights and Connections for Efficient Neural Networks, NIPS'15

Pruning

- Prune unimportance weight
  - Magnitude based method
  - If (w < threshold) w = 0;





Algorithm 1: Pruning Deep Neural Networks

Initialization: $W^{(0)}$ *with* $W^{(0)} \sim N(0, \Sigma)$, $iter = 0$.
Hyper-parameter: *threshold*, δ.
Output : $W^{(t)}$.

Train Connectivity

while *not converged* **do**
  $W^{(t)} = W^{(t-1)} - \eta^{(t)} \nabla f(W^{(t-1)}; x^{(t-1)})$;
  $t = t + 1$;
end

Prune Connections

// initialize the mask by thresholding the weights.
$Mask = \mathbb{1}(|W| > threshold)$;
$W = W \cdot Mask$;

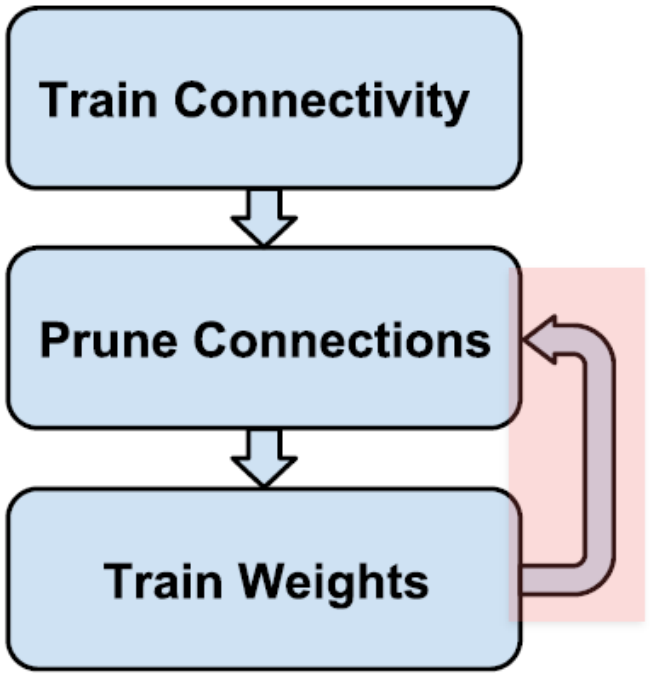
Retrain Weights

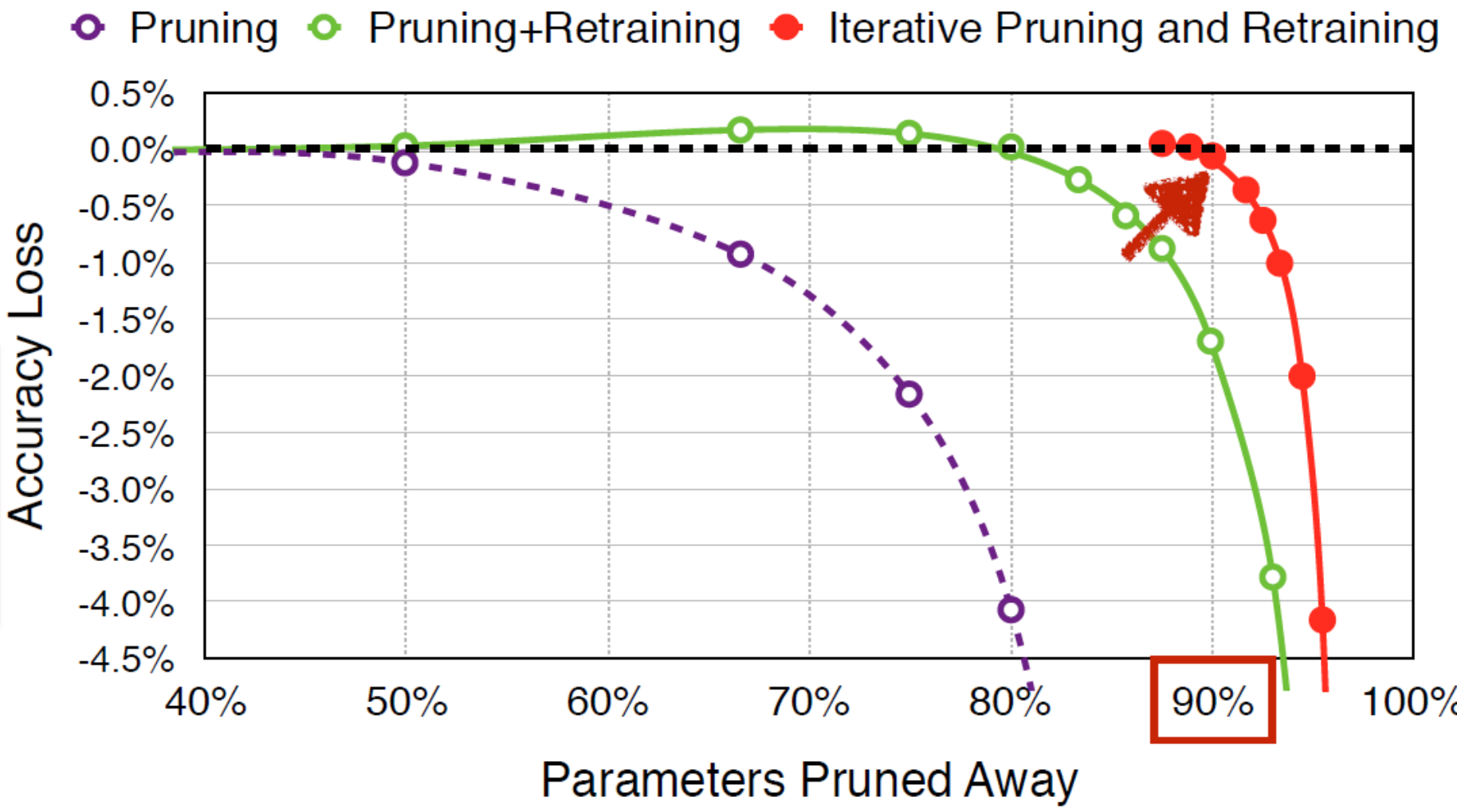
while *not converged* **do**
  $W^{(t)} = W^{(t-1)} - \eta^{(t)} \nabla f(W^{(t-1)}; x^{(t-1)})$;
  $W^{(t)} = W^{(t)} \cdot Mask$;
  $t = t + 1$;
end

Iterative Pruning

$threshold = threshold + \delta[iter++]$;
goto *Pruning Connections;*

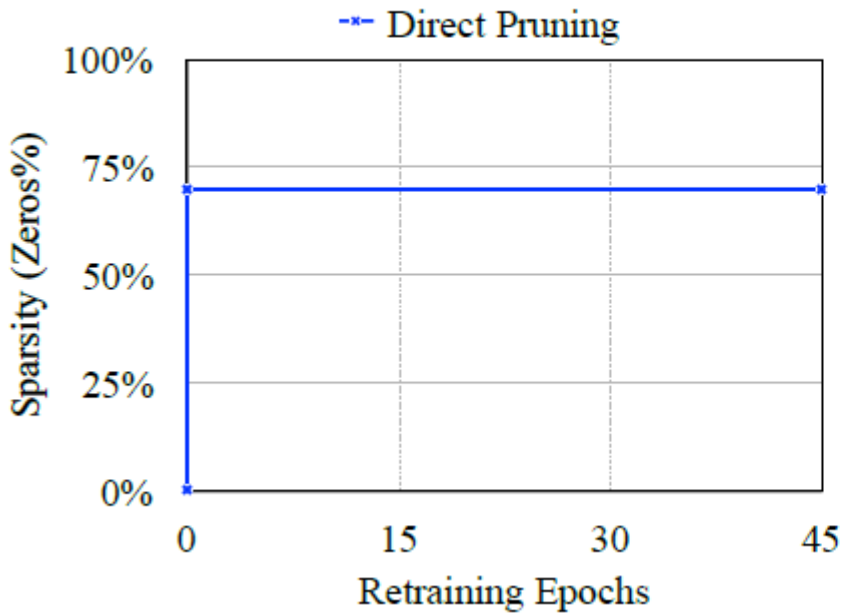
Iteratively Retrain to Recover Accuracy

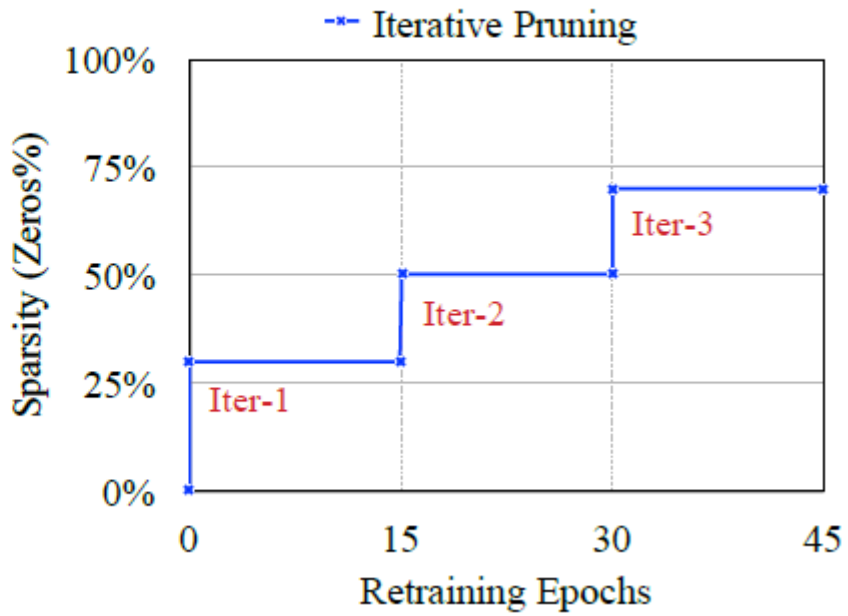




Pruning: pruning procedure

- Iterative pruning
  - Iterative pruning can achieve a better compression ratio than direct pruning
  - 5x to 9x pruning rate than direct pruning for AlexNet

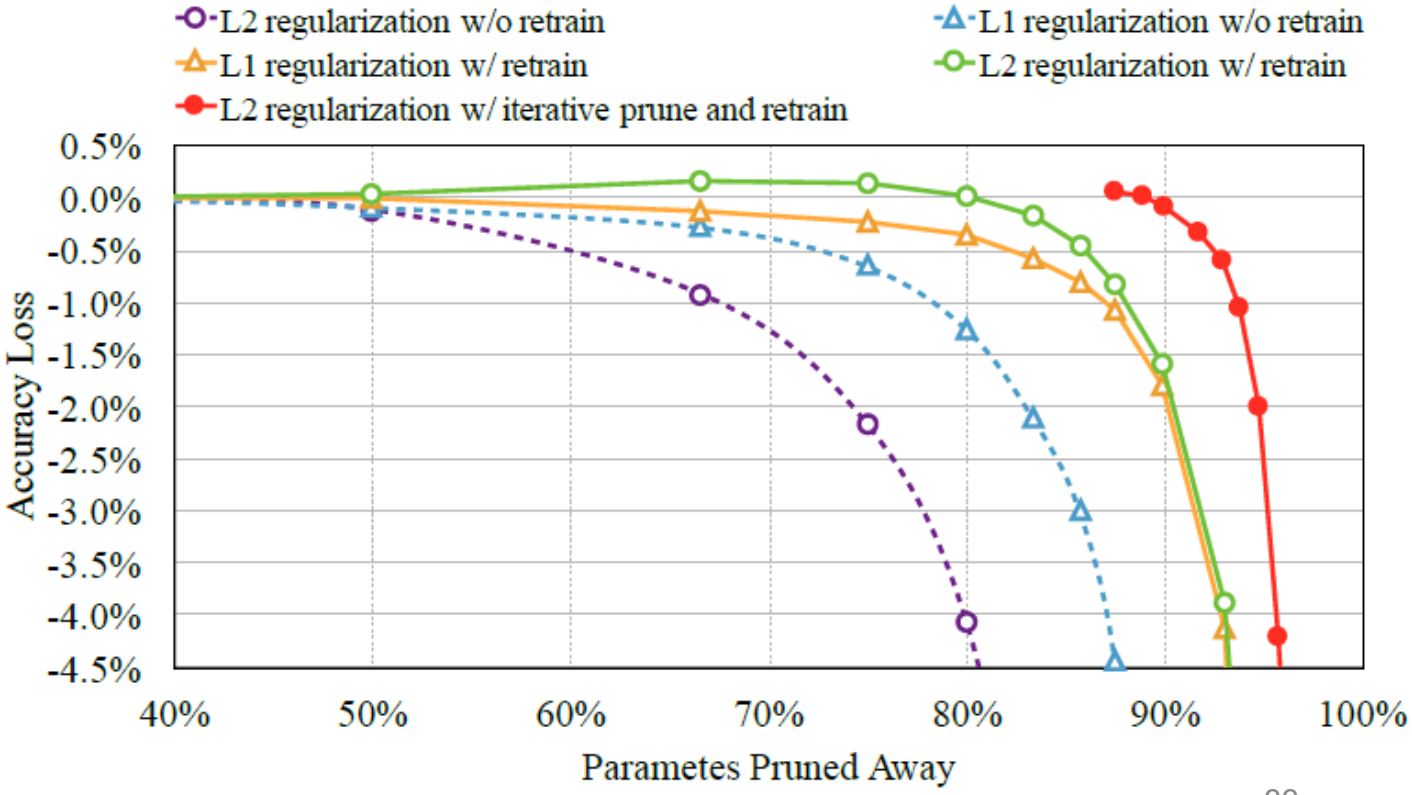




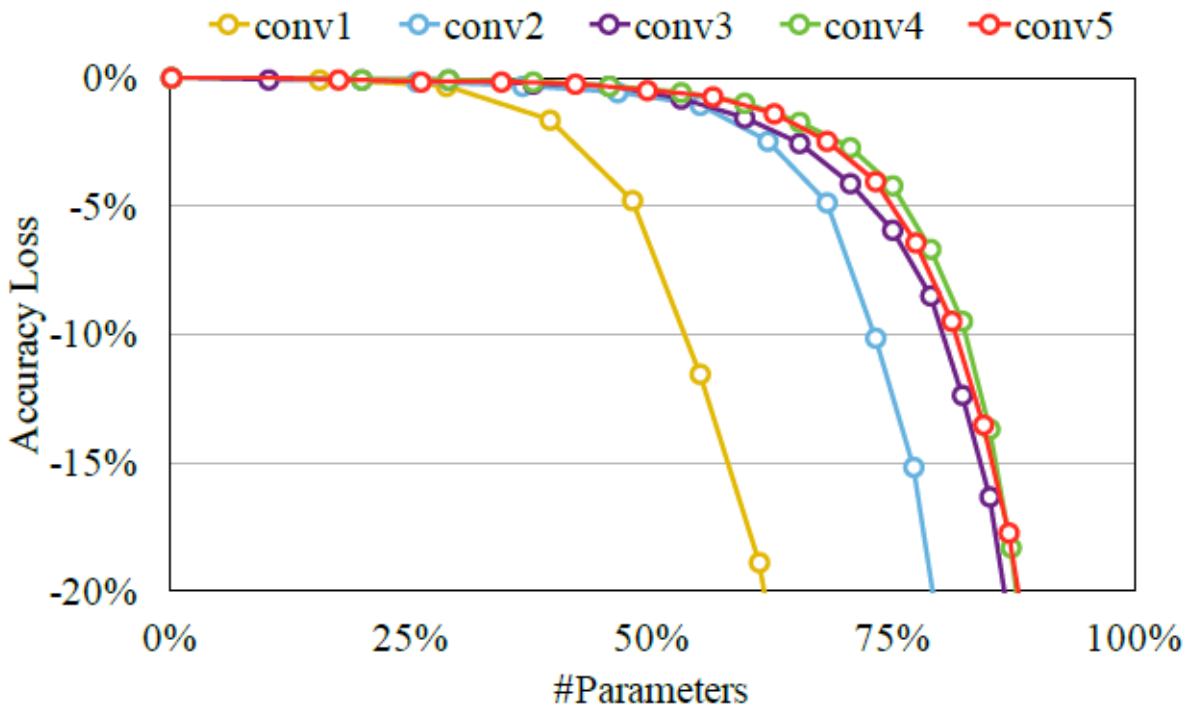
Pruning: pruning procedure

- Retraining (fine-tuning)
  - Free lunch: 2x wo retrain
  - 9x w retrain
- Training for pruning
  - L1 or L2 regularization
  - L1 results in more zeros
  - L2 + pruning+ retrain is better
    - No benefit to push more zero
  - Best: L2+ iterative prune + retrain

砍完，重新訓練才能恢復準確率



Pruning: Sensitivity



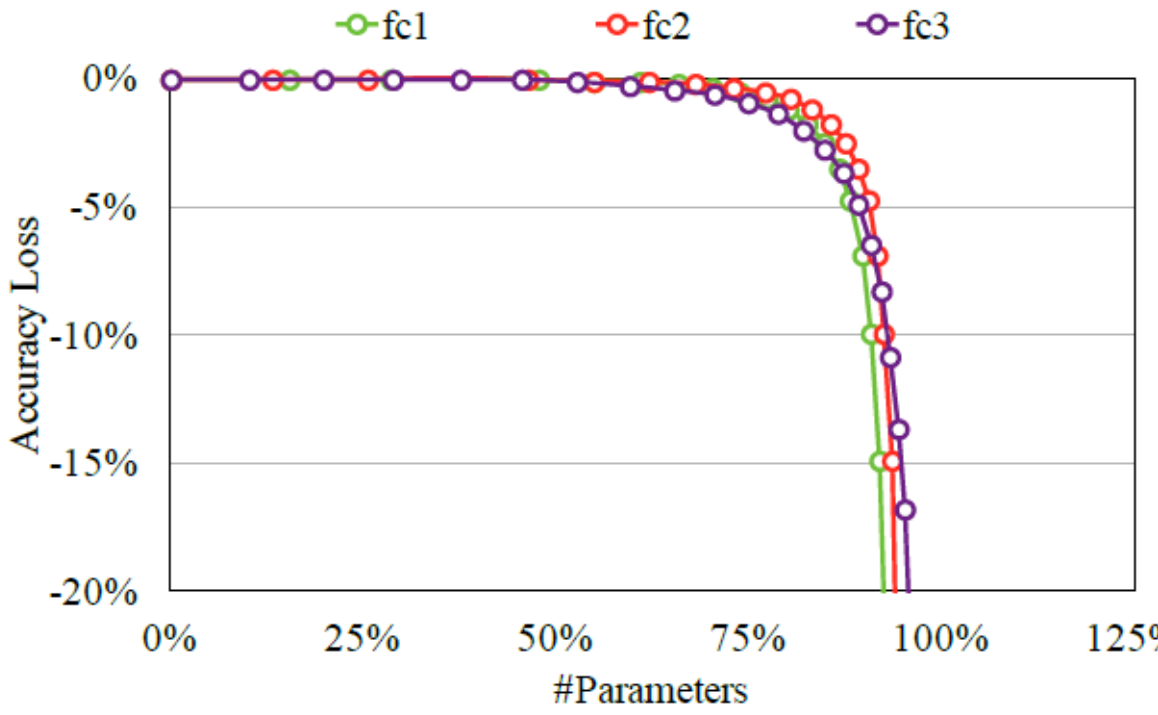
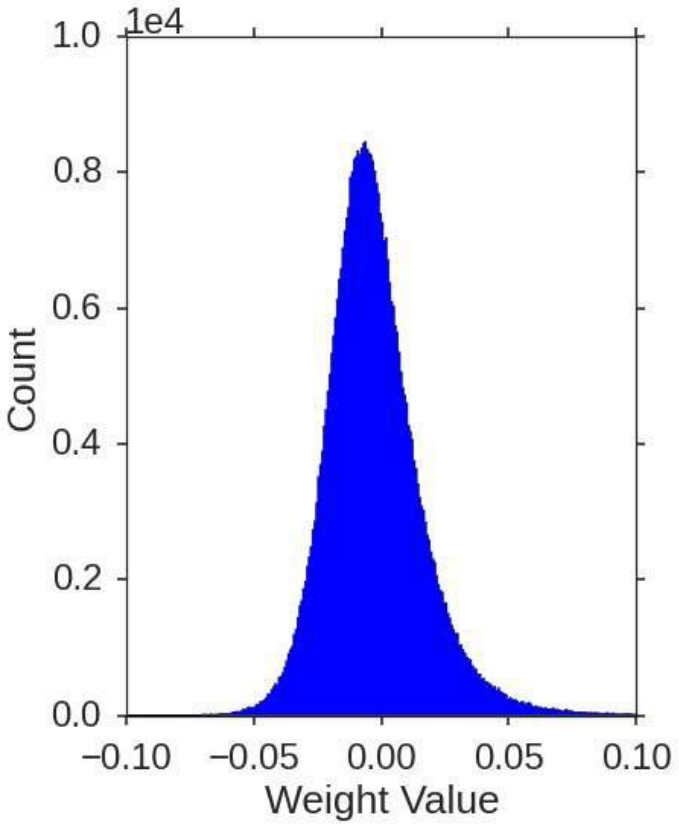


Figure 3.14: Pruning sensitivity for CONV layer (left) and FC layer (right) of AlexNet.

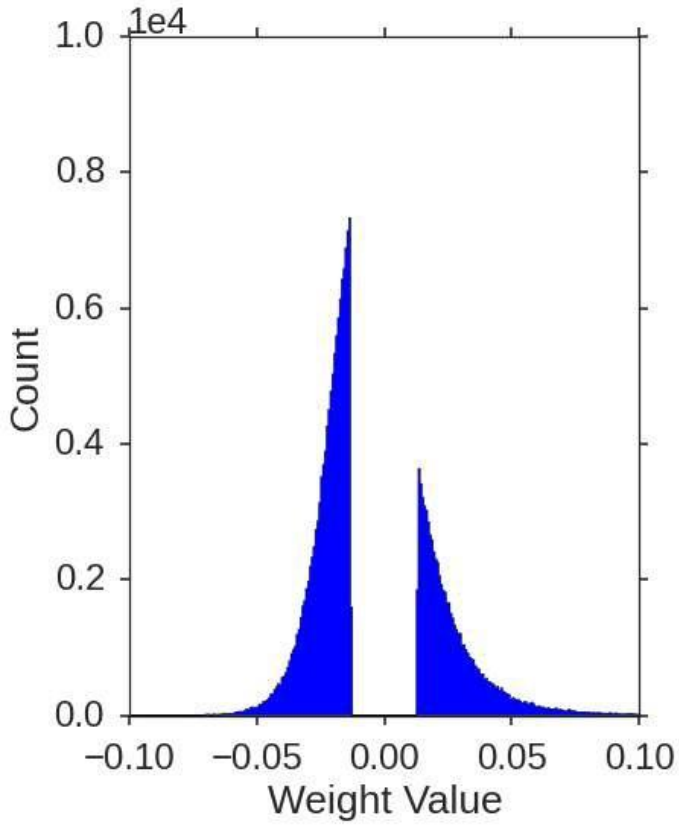
輸入層最敏感

Pruning Changes Weight Distribution

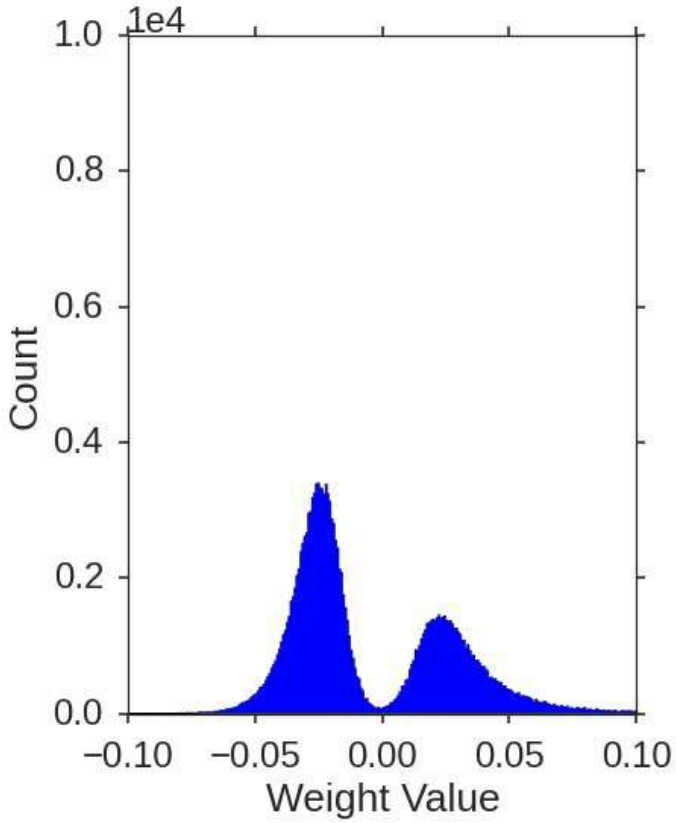
Before Pruning



After Pruning



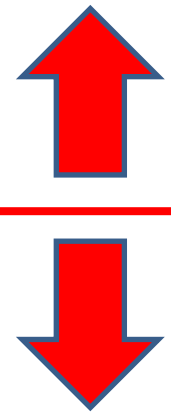
After Retraining



Conv5 layer of Alexnet. Representative for other network layers as well.

Pruning: Results

Network	Top-1 Error	Top-5 Error	Parameters	Pruning Rate
LeNet-300-100	1.64%	-	267K	
LeNet-300-100 Pruned	1.59%	-	**22K**	**12×**
LeNet-5	0.80%	-	431K	
LeNet-5 Pruned	0.77%	-	**36K**	**12×**
AlexNet	42.78%	19.73%	61M	
AlexNet Pruned	42.77%	19.67%	**6.7M**	**9×**
VGG-16	31.50%	11.32%	138M	
VGG-16 Pruned	31.34%	10.88%	**10.3M**	**13×**
GoogleNet	31.14%	10.96%	7.0M	
GoogleNet Pruned	31.04%	10.88%	**2.0M**	**3.5×**
SqueezeNet	42.56%	19.52%	1.2M	
SqueezeNet Pruned	42.26%	19.34%	**0.38M**	**3.2×**
ResNet-50	23.85%	7.13%	25.5M	
ResNet-50 Pruned	23.65%	6.85%	**7.47M**	**3.4×**



FC 層權重數量太多

Fully convolutional networ

FC 層權重數量少

Pruning AlexNet reduces the number of weights by 9× and computation by 3×.

Layer	Weights	FLOP	Activation%	Weight%	FLOP%
conv1	35K	211M	88%	84%	84%
conv2	307K	448M	52%	38%	33%
conv3	885K	299M	37%	35%	18%
conv4	663K	224M	40%	37%	14%
conv5	442K	150M	34%	37%	14%
fc1	38M	75M	36%	9%	3%
fc2	17M	34M	40%	9%	3%
fc3	4M	8M	100%	25%	10%
total	61M	1.5B	54%	**11%**	**30%**

ZERO WEIGHT/ACTIVATION RATIO OF ALEXNET [1]

Layer	Zero Weight [%]	Zero Activation [%]
conv1	15.7	0
conv2	62.1	50.9
conv3	65.4	76.3
conv4	62.8	61.8
conv5	63.1	59.0

Pruning VGG-16 reduces the number of weights by 12× and computation by 5×.

Layer	Weights	FLOP	Activation%	Weight%	FLOP%
conv1_1	2K	0.2B	53%	58%	58%
conv1_2	37K	3.7B	89%	22%	12%
conv2_1	74K	1.8B	80%	34%	30%
conv2_2	148K	3.7B	81%	36%	29%
conv3_1	295K	1.8B	68%	53%	43%
conv3_2	590K	3.7B	70%	24%	16%
conv3_3	590K	3.7B	64%	42%	29%
conv4_1	1M	1.8B	51%	32%	21%
conv4_2	2M	3.7B	45%	27%	14%
conv4_3	2M	3.7B	34%	34%	15%
conv5_1	2M	925M	32%	35%	12%
conv5_2	2M	925M	29%	29%	9%
conv5_3	2M	925M	19%	36%	11%
fc6	103M	206M	38%	4%	1%
fc7	17M	34M	42%	4%	2%
fc8	4M	8M	100%	23%	9%

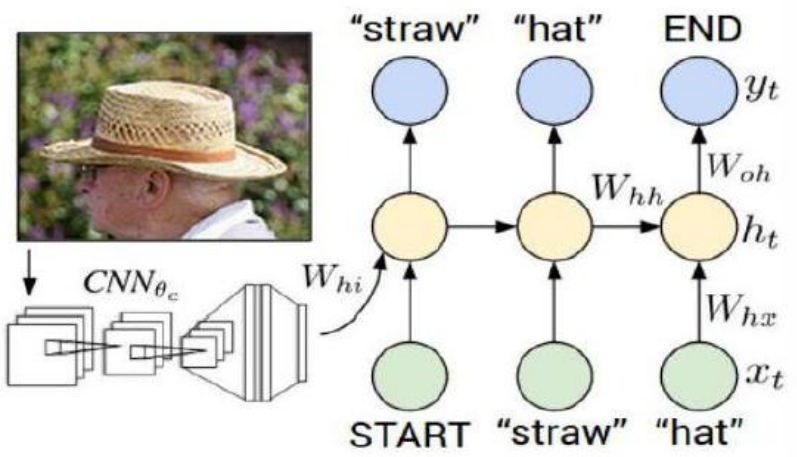
Weight in FC layer can be very sparse

Layer	Weights	FLOP	Activation%	Weight%	FLOP%
conv1	9K	236B	90%	50%	50%
layer1.0.conv1	4K	26B	58%	40%	36%
layer1.0.conv2	37K	231B	64%	30%	18%
layer1.0.conv3	16K	103B	59%	30%	19%
layer1.0.shortcut	16K	103B	59%	40%	36%
layer1.1.conv1	16K	103B	48%	30%	18%
layer1.1.conv2	37K	231B	51%	30%	14%
layer1.1.conv3	16K	103B	75%	30%	15%
layer1.2.conv1	16K	103B	49%	30%	22%
layer1.2.conv2	37K	231B	44%	30%	15%
layer1.2.conv3	16K	103B	80%	30%	13%
layer2.0.conv1	33K	206B	41%	40%	32%
layer2.0.conv2	147K	231B	58%	33%	14%
layer2.0.conv3	66K	103B	50%	30%	17%
layer2.0.shortcut	131K	206B	50%	30%	24%
layer2.1.conv1	66K	103B	61%	30%	15%
layer2.1.conv2	147K	231B	56%	30%	18%
layer2.1.conv3	66K	103B	66%	30%	17%
layer2.2.conv1	66K	103B	54%	30%	20%
layer2.2.conv2	147K	231B	55%	30%	16%
layer2.2.conv3	66K	103B	58%	30%	16%
layer2.3.conv1	66K	103B	49%	30%	17%
layer2.3.conv2	147K	231B	41%	30%	15%
layer2.3.conv3	66K	103B	59%	30%	12%
layer3.0.conv1	131K	206B	32%	40%	23%
layer3.0.conv2	590K	231B	62%	30%	10%
layer3.0.conv3	262K	103B	47%	30%	18%
layer3.0.shortcut	524K	206B	47%	30%	18%
layer3.1.conv1	262K	103B	48%	30%	14%
layer3.1.conv2	590K	231B	43%	30%	14%
layer3.1.conv3	262K	103B	52%	30%	13%
layer3.2.conv1	262K	103B	41%	30%	15%
layer3.2.conv2	590K	231B	40%	30%	12%
layer3.2.conv3	262K	103B	50%	30%	12%
layer3.3.conv1	262K	103B	36%	30%	15%
layer3.3.conv2	590K	231B	39%	30%	11%
layer3.3.conv3	262K	103B	48%	30%	12%
layer3.4.conv1	262K	103B	34%	30%	14%
layer3.4.conv2	590K	231B	35%	30%	10%
layer3.4.conv3	262K	103B	40%	30%	11%
layer3.5.conv1	262K	103B	31%	30%	12%
layer3.5.conv2	590K	231B	36%	30%	9%
layer3.5.conv3	262K	103B	32%	30%	11%
layer4.0.conv1	524K	206B	23%	30%	10%
layer4.0.conv2	2M	231B	38%	30%	7%
layer4.0.conv3	1M	103B	41%	30%	12%
layer4.0.shortcut	2M	206B	41%	30%	10%
layer4.1.conv1	1M	103B	27%	30%	12%
layer4.1.conv2	2M	231B	32%	30%	8%
layer4.1.conv3	1M	103B	56%	30%	10%
layer4.1.conv1	1M	103B	21%	30%	17%
layer4.1.conv2	2M	231B	37%	30%	6%
layer4.1.conv3	1M	103B	100%	30%	11%
fc	2M	4K	100%	20%	20%
total	25.5M	8G	56%	29%	16%

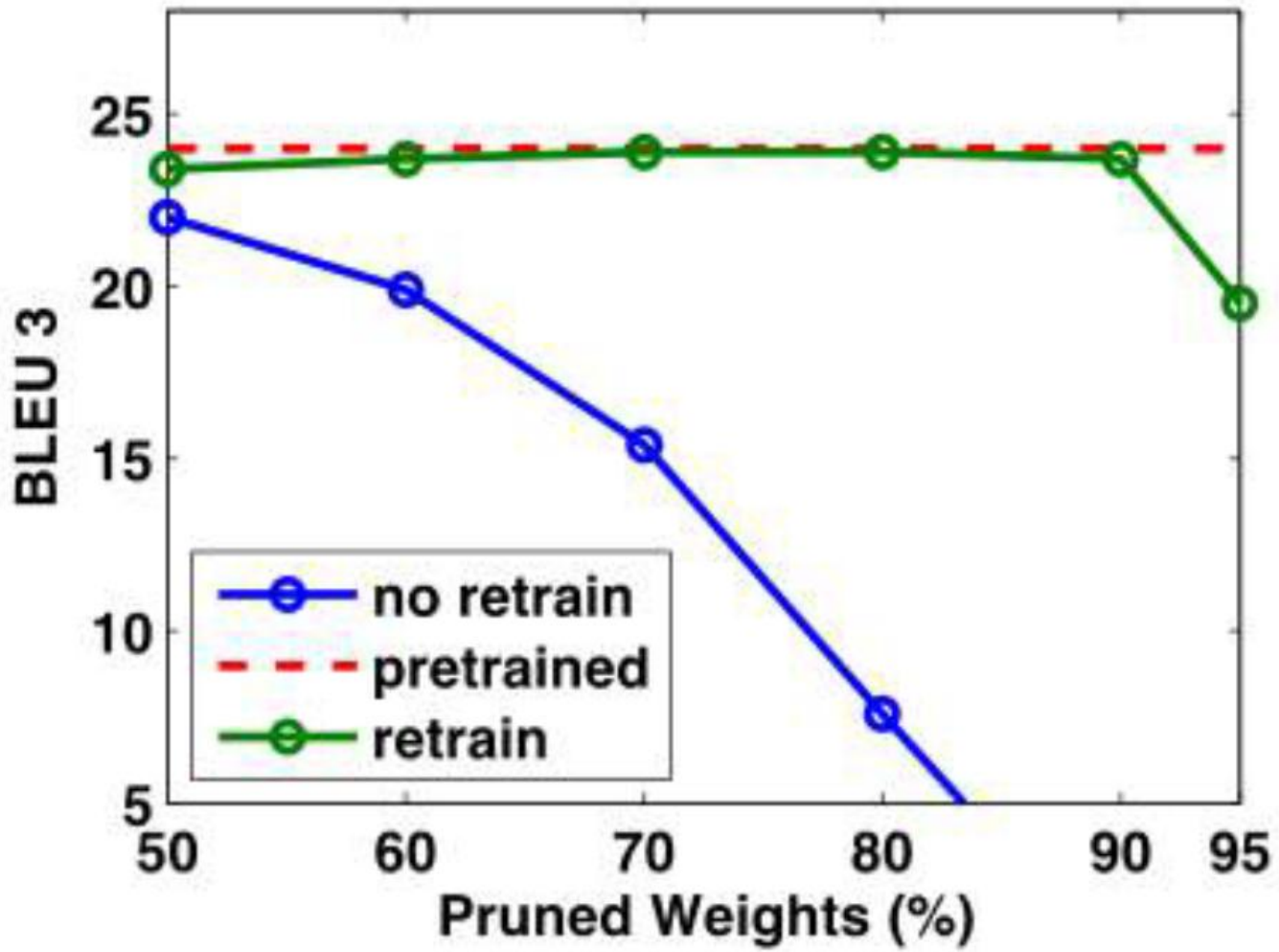
ResNet-50 reduces the number of weights by 3.4X and computation by 6.25X

1. Bypass changes the activation sparsity due to add operation
2. FC layer in ResNet is global average pooling, no reduction in activation

Pruning: RNN and LSTM



*Karpathy et al, "Deep Visual-Semantic Alignments for Generating Image Descriptions", 2015.



Pruning: Effect on RNN and LSTM

90%

- **Original**: a basketball player in a white uniform is playing with a ball
- **Pruned 90%**: a basketball player in a white uniform is playing with a basketball

90%

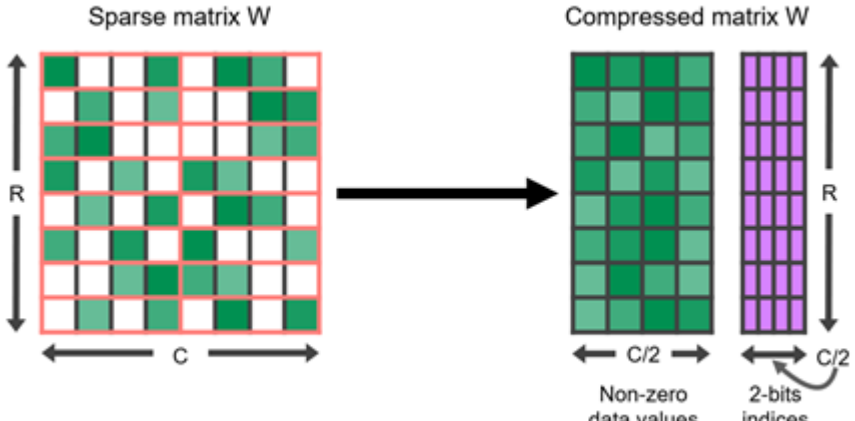
- **Original** : a man is riding a surfboard on a wave
- **Pruned 90%**: a man in a wetsuit is riding a wave on a beach

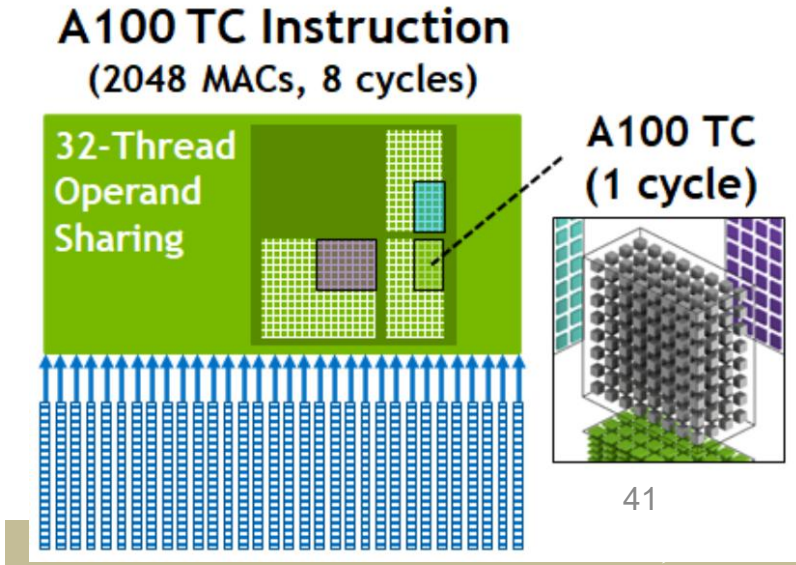
95%

- **Original** : a soccer player in red is running in the field
- **Pruned <u>95%</u>**: a man in a red shirt and black and white black shirt is running through a field

Computation with Accelerators

- Specialized matrix multiplication hardware is a disadvantage for sparsity
- Latest Nvidia A100 GPU with tensor cores
  - Sparse Matrix Multiply (SpMM) peak is 1/16 of dense matrix multiplication (GEMM) peak
    - To make it worse SpMM achieves 1/3 of peak, ($\mathbf{E}_{sparse} \leq 1/3$)
    - GEMM achieves 2/3 of peak ($\mathbf{E}_{dense}$)
  - 2:4 fine-grained structured sparsity
    - 50% reduction
    - 2X speedup





Pruning: Regular Sparsity for HW Load Balance

- Sort the L1 norm of that granularity
  - Discard if smaller
  - E.g. target 30% sparsity
  - Discard smallest 70%

$$S_i = \sum_{w \in G_i} |w|$$

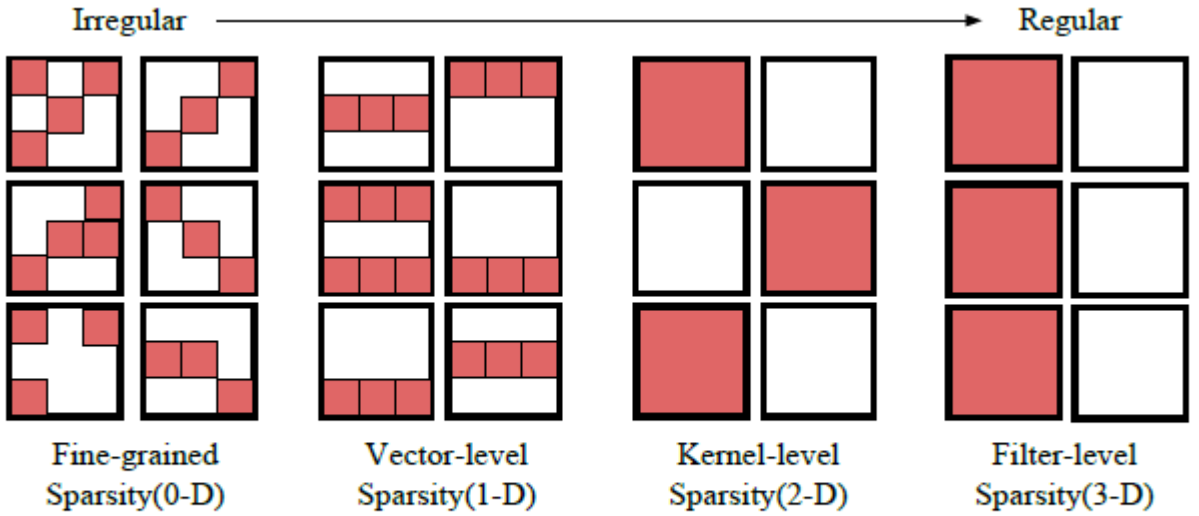


Figure 1: Different sparse structure in a 4-dimensional weight tensor. Regular sparsity makes hardware acceleration easier.

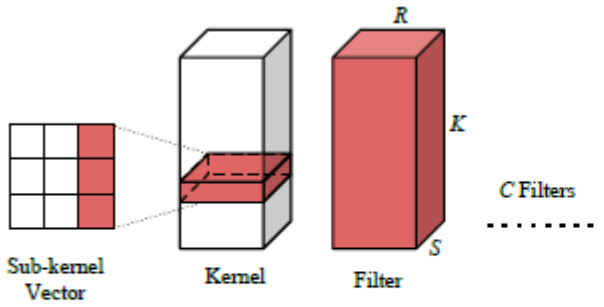


Figure 2: Example of Sub-kernel Vector, Filter and Kernel.

```
Weights = Array(C, K, R, S)

# Case: Dimension-level granularity
Filter(3-Dim) = Weights[c, :, :, :]
Kernel(2-Dim) = Weights[c, k, :, :]
Vector(1-Dim) = Weights[c, k, r, :]
Fine-grain(0-Dim) = Weights[c, k, r, s]
```

Pseudo code: different granularity levels

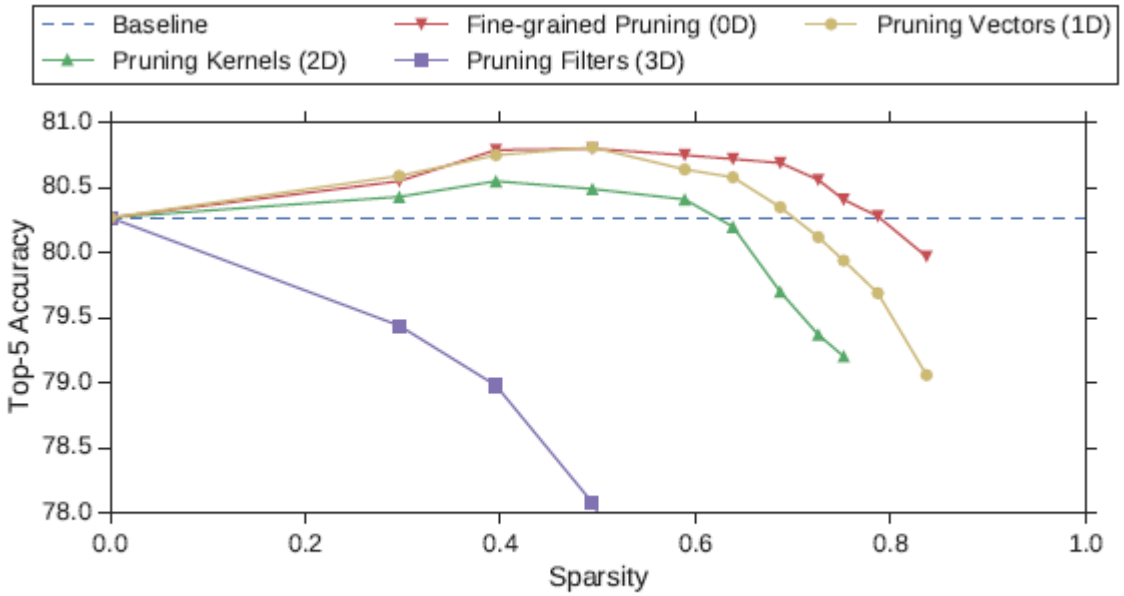


Figure 4: Accuracy-Sparsity Curve of AlexNet obtained by iterative pruning.

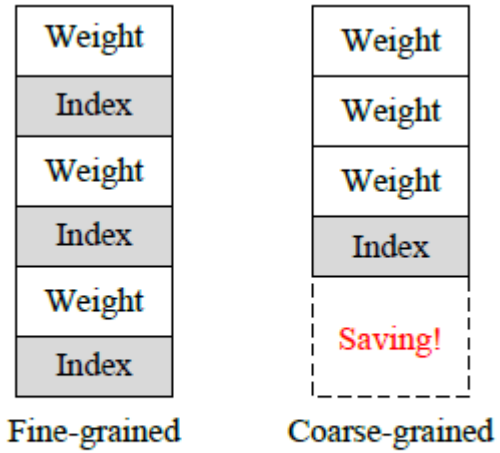


Figure 5: Illustration of index saving.

Table 1: Comparison of accuracies with the same density/sparsity.

Model	Density	Granularity	Top-5
AlexNet	24.8%	Kernel Pruning (2-D)	79.20%
		Vector Pruning (1-D)	79.94%
		Fine-grained Pruning (0-D)	**80.41%**
VGG-16	23.5%	Kernel Pruning (2-D)	89.70%
		Vector Pruning (1-D)	90.48%
		Fine-grained Pruning (0-D)	**90.56%**
GoogLeNet	38.4%	Kernel Pruning (2-D)	88.83%
		Vector Pruning (1-D)	89.11%
		Fine-grained Pruning (0-D)	**89.40%**
ResNet-50	40.0%	Kernel Pruning (2-D)	92.07%
		Vector Pruning (1-D)	92.26%
		Fine-grained Pruning (0-D)	**92.34%**
DenseNet-121	30.1%	Kernel Pruning (2-D)	91.56%
		Vector Pruning (1-D)	91.89%
		Fine-grained Pruning (0-D)	**92.21%**

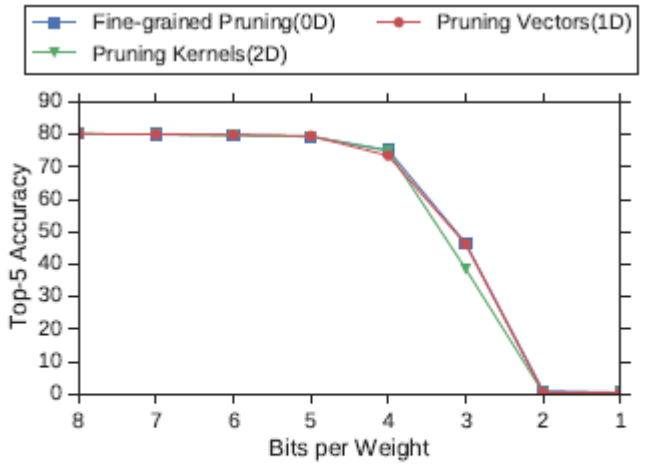
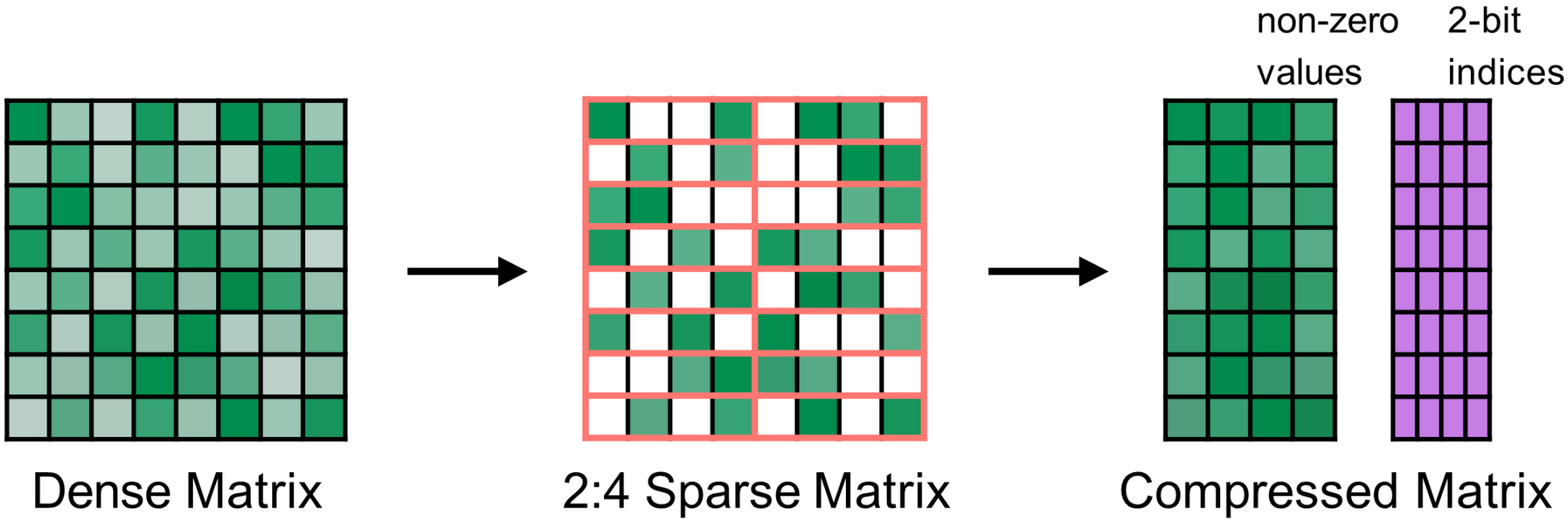


Figure 6: Three curves are almost identical, indicating sparsity structure does not impact quantization.

Pruning at Different Granularities

- Pattern-based Pruning: N:M sparsity
  - N:M sparsity means that in each contiguous M elements, N of them is pruned
  - A classic case is 2:4 sparsity (50% sparsity)
  - It is supported by NVIDIA's Ampere GPU Architecture, which delivers up to 2x speed up
  - Support by TensorRT to maintain accuracy



Sparsity vs Speedup

- Unit pruning
  - achieves nearly linear speedup
- block pruning
  - shows slightly sublinear performance.
- Weight pruning
  - shows negative speedup below approximately 85% sparsity.

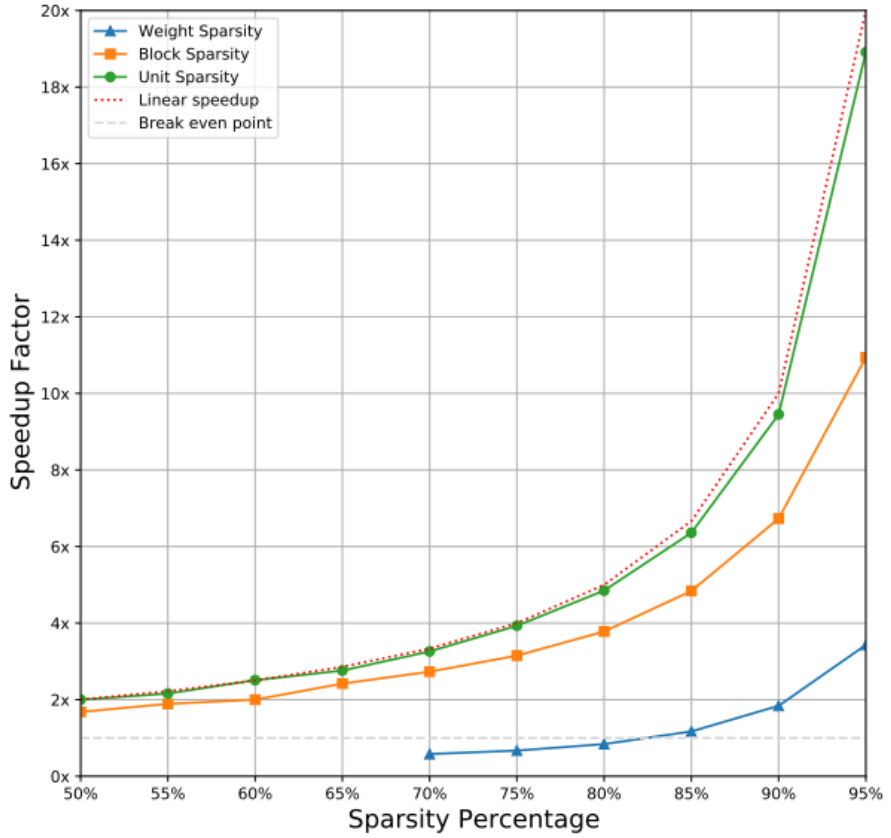


Figure 1: Plot of sparsity vs. computational speedup based on data aggregated from [33, 28] for the three sparsity structures we consider.

Learning Structured Sparsity in Deep Neural Networks

Fine grained 需要比較高的sparsity，才有加速效果

- With regularization
- Cost = data loss + structured regularizer + nonstructured regularizer

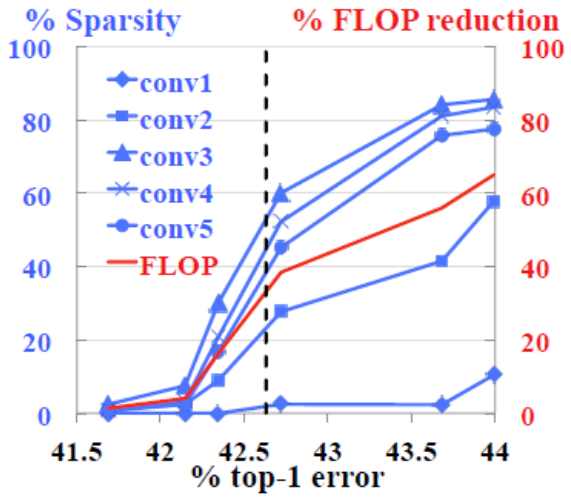


Table 4: Sparsity and speedup of *AlexNet* on ILSVRC 2012

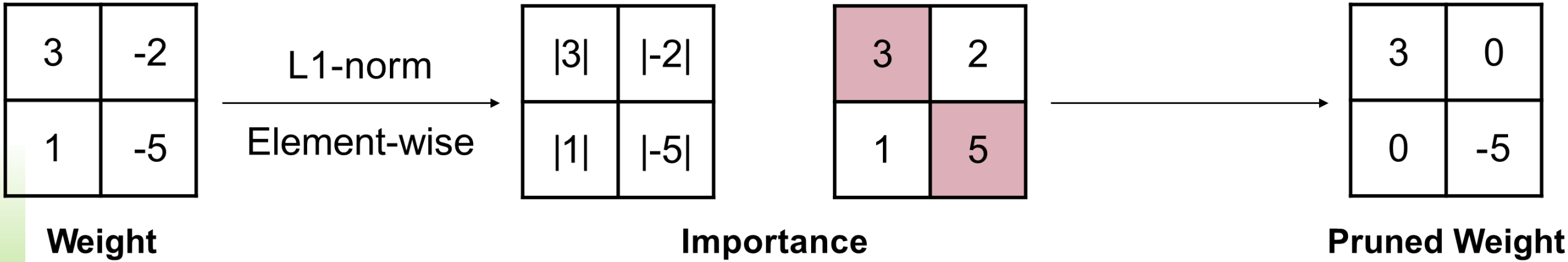
#	Method	Top1 err.	Statistics	conv1	conv2	conv3	conv4	conv5
1	ℓ_1	44.67%	sparsity	67.6%	92.4%	97.2%	96.6%	94.3%
			CPU ×	0.80	2.91	4.84	3.83	2.76
			GPU ×	0.25	0.52	1.38	1.04	1.36
2	SSL	44.66%	column sparsity	0.0%	63.2%	76.9%	84.7%	80.7%
			row sparsity	9.4%	12.9%	40.6%	46.9%	0.0%
			CPU ×	1.05	3.37	6.27	9.73	4.93
			GPU ×	1.00	2.37	4.94	4.03	3.05
3	pruning[7]	42.80%	sparsity	16.0%	62.0%	65.0%	63.0%	63.0%
4	ℓ_1	42.51%	sparsity	14.7%	76.2%	85.3%	81.5%	76.3%
			CPU ×	0.34	0.99	1.30	1.10	0.93
			GPU ×	0.08	0.17	0.42	0.30	0.32
5	SSL	42.53%	column sparsity	0.00%	20.9%	39.7%	39.7%	24.6%
			CPU ×	1.00	1.27	1.64	1.68	1.32
			GPU ×	1.00	1.25	1.63	1.72	1.36

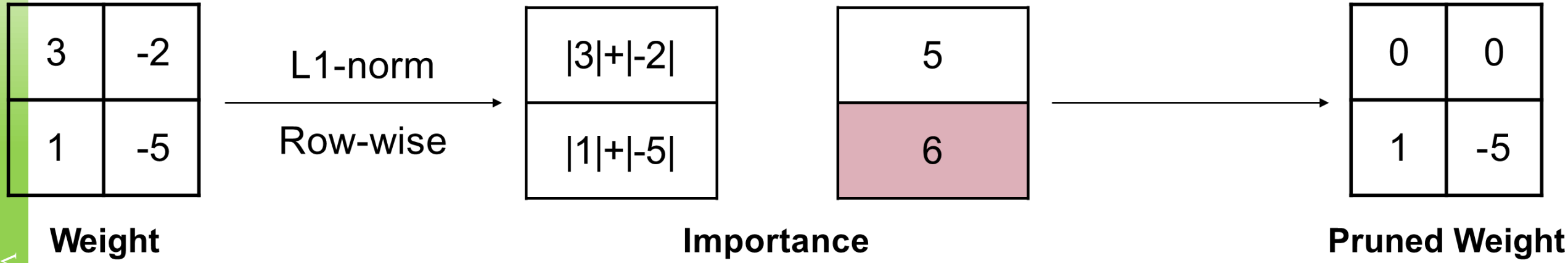
$$E(\boldsymbol{W}) = E_D(\boldsymbol{W}) + \lambda \cdot R(\boldsymbol{W}) + \lambda_g \cdot \sum^{L} R_g\left(\boldsymbol{W}^{(l)}\right)$$

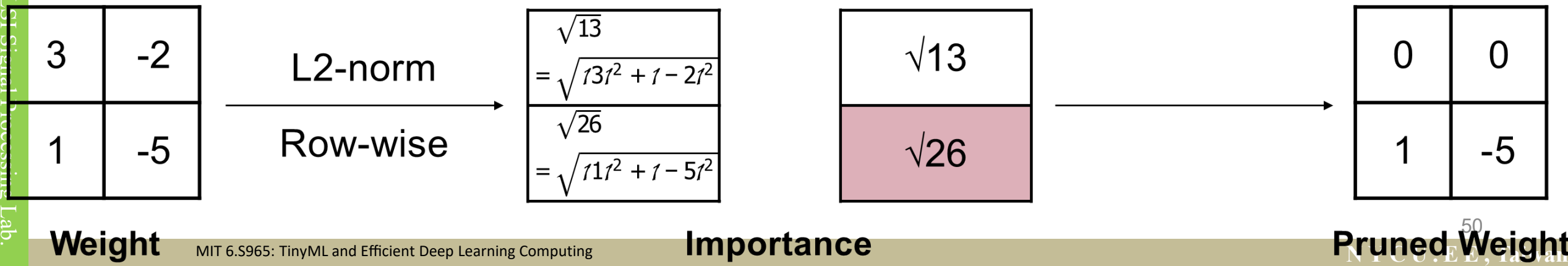
$$R_g(\boldsymbol{w}) = \sum_{g=1}^{G} ||\boldsymbol{w}^{(g)}||_g \quad ||\boldsymbol{w}^{(g)}||_g = \sqrt{\sum_{i=1}^{|\boldsymbol{w}^{(g)}|} \left(w_i^{(g)}\right)^2}$$

Pruning Criterions

- Remove the less important parameters
- Q: how to define the importance?
- Magnitude-based Pruning
  - Heuristic approach
  - Importance = $|w_i|$ or L1 norm $\sum |W_i|$ or L2 norm $\sum |W_i|^2$


3
-2
1
-5
L1-norm
Element-wise
|3|
|-2|
|1|
|-5|
3
2
1
5
3
0
0
-5
Weight
Importance
Pruned Weight


3
-2
1
-5
L1-norm
Row-wise
|3|+|-2|
|1|+|-5|
5
6
0
0
1
-5
Weight
Importance
Pruned Weight


3
-2
1
-5
L2-norm
Row-wise
$\sqrt{13}$
$= \sqrt{13^2 + 1 - 2^2}$
$\sqrt{26}$
$= \sqrt{11^2 + 1 - 5^2}$
√13
√26
0
0
1
-5
Weight
Importance
Pruned Weight

Network Slimming: BN Scaling Factors Channel Sparsity to Channel Pruning

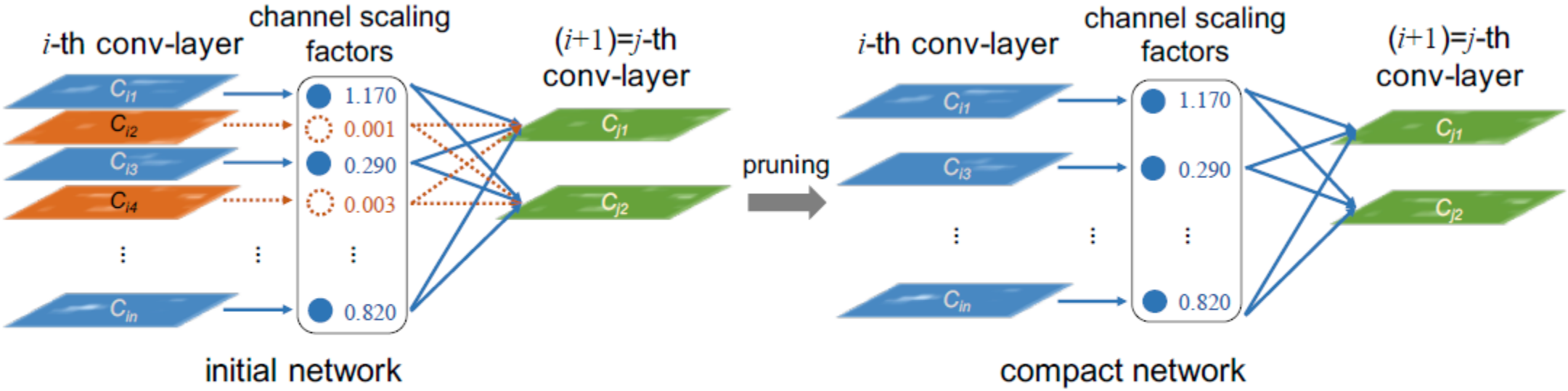


Figure 1: We associate a scaling factor (reused from a batch normalization layer) with each channel in convolutional layers. Sparsity regularization is imposed on these scaling factors during training to automatically identify unimportant channels. The channels with small scaling factor values (in orange color) will be pruned (left side). After pruning, we obtain compact models (right side), which are then fine-tuned to achieve comparable (or even higher) accuracy as normally trained full network.

Channel Sparsity to Channel Pruning

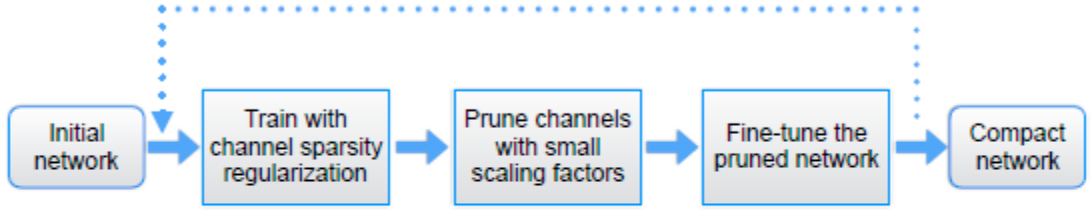


Figure 2: Flow-chart of network slimming procedure. The dotted-line is for the multi-pass/iterative scheme.

- L1 norm penalty for sparsity
- Scaling factor combined with BN to select pruned channels

$$\hat{z} = \frac{z_{in} - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}; \quad z_{out} = \gamma\hat{z} + \beta$$

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

VGG-A	Baseline	50% Pruned
Params	132.9M	23.2M
Params Pruned	-	82.5%
FLOPs	4.57×10^{10}	3.18×10^{10}
FLOPs Pruned	-	30.4%
Validation Error (%)	36.69	36.66

Table 2: Results on ImageNet.

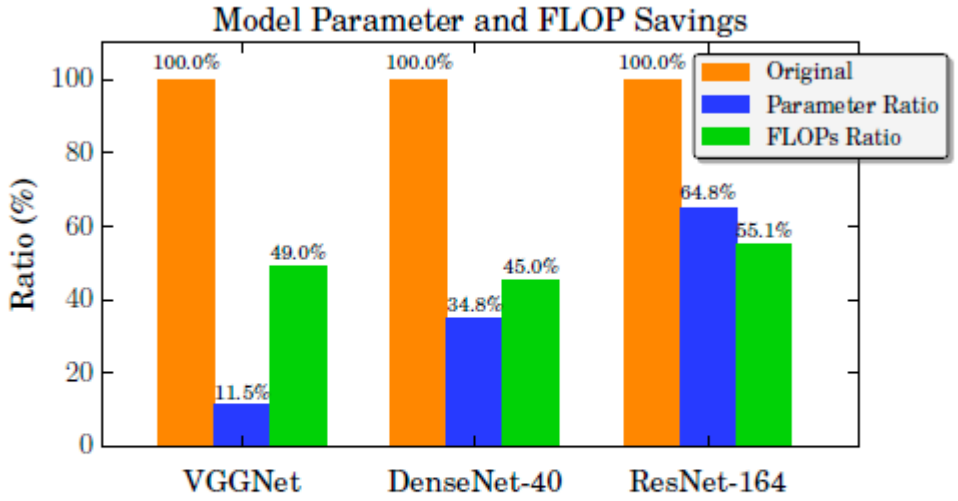


Figure 3: Comparison of pruned models with **lower** test errors on CIFAR-10 than the original models. The blue and green bars are parameter and FLOP ratios between pruned and original models.

Polarization Regularization

- Regularization term for r

$$\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} L(\boldsymbol{f}(\boldsymbol{x_i}; \boldsymbol{\theta}), \boldsymbol{y}_i) + R(\boldsymbol{\theta}) + \lambda R_s(\boldsymbol{\gamma})$$

$$R_s(\boldsymbol{\gamma}) = t\|\boldsymbol{\gamma}\|_1 - \|\boldsymbol{\gamma} - \bar{\gamma}\mathbf{1}_n\|_1$$

$$= \sum_{i=1}^{n} t|\gamma_i| - |\gamma_i - \bar{\gamma}|, (t \in \mathbb{R}, \gamma_i \in [0, a])$$

Minimum for half = 0, half = a

Push all factors to 0

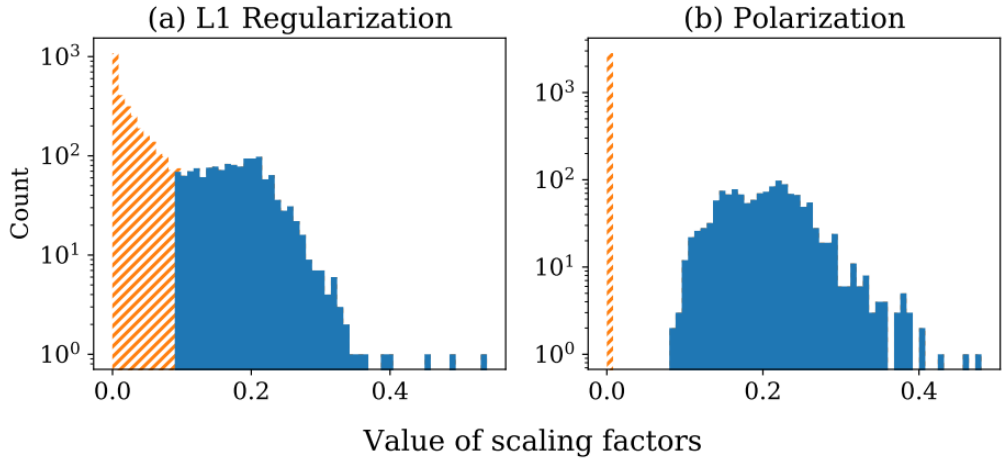


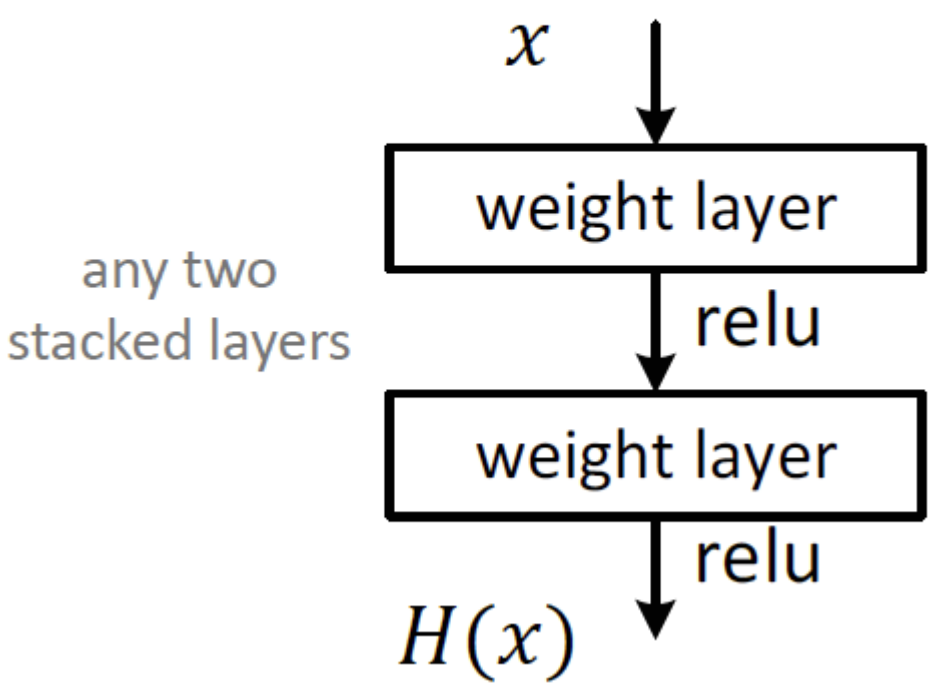
Figure 1: The distributions of scaling factors in VGG-16 trained on CIFAR-10 dataset, with L1 an polarization regularizers respectively. Under the same pruning ratio for both regularizers, the orang part are pruned.

t also controls the proportion of scaling factors that equal 0 under polarization regularizer.

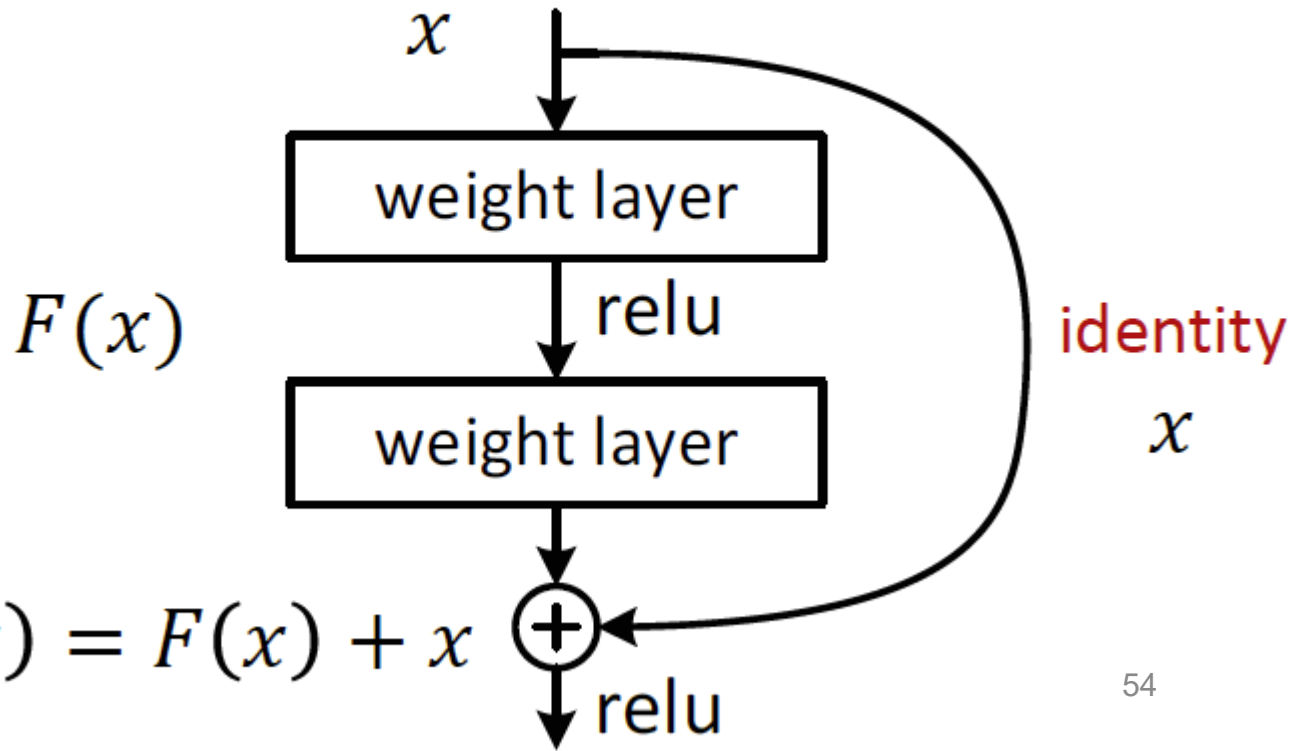
Layer Pruning

- Shortcuts to avoid cuting off the message propagation in the DNN

- Plaint net



- Residual net



Taylor Expansion Analysis on Pruning Error

- Evaluate pruning error induced by pruning synapses
- The induced error can be approximated by a Taylor series

$$\delta L = L(\mathbf{x}; \mathbf{W}) - L(\mathbf{x}; \mathbf{W}_P = \mathbf{W} - \delta\mathbf{W}) = \sum_i g_i \delta w_i + \frac{1}{2}\sum_i h_{ii}\delta w_i^2 + \frac{1}{2} h_{ij}\delta w_i \delta w_j + O(\|\delta\mathbf{W}\|^3)$$

where

$$g_i = \frac{\partial L}{\partial w_i}, \; h_{i,j} = \frac{\partial^2 L}{\partial w_i \partial w_j}$$

- There are two ways to simplify these approximation.

Second-Order-based Pruning

$$\delta L = L(\mathbf{x}; \mathbf{W}) - L(\mathbf{x}; \mathbf{W}_P = \mathbf{W} - \delta\mathbf{W}) = \sum_i g_i w_i + \frac{1}{2}\sum_i h_{ii}\delta w_i^2 + \frac{1}{2}h_{ij}\delta w_i \delta w_j + O(\|\delta\mathbf{W}\|^3)$$

where

$$g_i = \frac{\partial L}{\partial w_i},\ h_{i,j} = \frac{\partial^2 L}{\partial w_i \partial w_j}$$

- Optimal Brain Damage assumes that
  - The objective function L is nearly quadratic: the last term is neglected
  - The neural network training has converged: first-order terms are neglected
  - The error caused by deleting each parameter is independent: cross terms are neglected

$$\delta L_i = L(\mathbf{x}; \mathbf{W}) - L(\mathbf{x}; \mathbf{W}_P \,|\, w_i = 0) \approx \frac{1}{2} h_{ii} w_i^2$$

Optimal Brain Damage [LeCun et al., NeurIPS 1989]

$$\delta L_i = L(\mathbf{x}; \mathbf{W}) - L(\mathbf{x}; \mathbf{W}_P \mid w_i = 0) \approx \frac{1}{2} h_{ii} w_i^2, \quad \text{where} \quad h_{ii} = \frac{\partial^2 L}{\partial w_i \partial w_j}$$

- The synapses with smaller induced error $|\delta L_i|$ will be removed; that is to say,

$$importance_{w_i} = |\delta L_i| = \frac{1}{2} h_{ii} w_i^2$$

* h_{ii} is non-negative

Hessian Matrix *H* is difficult to compute.

First-Order-based Pruning

- Minimize the error on loss function introduced by pruning synapses

$$\delta L = L(\mathbf{x}; \mathbf{W}) - L(\mathbf{x}; \mathbf{W}_P = \mathbf{W} - \delta\mathbf{W}) = \sum_i g_i \delta w_i + \frac{1}{2}\sum_i h_{ii} w_i^2 + \frac{1}{2} h_{ij} \delta w_i w_j + O(\|\mathbf{W}\|^3)$$

where

$$g_i = \frac{\partial L}{\partial w_i}, \; h_{i,j} = \frac{\partial^2 L}{\partial w_i \partial w_j}$$

- If only first-order expansion is considered, under an i.i.d assumption

$$\delta L_i = L(\mathbf{x}; \mathbf{W}) - L(\mathbf{x}; \mathbf{W}_P \,|\, w_i = 0) \approx g_i w_i$$

Pruning Convolutional Filters with First Order Taylor Series Ranking [Wang M.]
Importance Estimation for Neural Network Pruning [Molchanov et al., CVPR 2019]

$$\delta L_i = L(\mathbf{x}; \mathbf{W}) - L(\mathbf{x}; \mathbf{W}_P \,|\, w_i = 0) \approx g_i w_i$$

- The synapses with smaller induced error will be removed; that is to say,

$$importance_{w_i} = |\delta L_i| = |g_i w_i|$$

- Or

$$importance_{w_i} = |\delta L_i|^2 = (g_i w_i)^2$$

- For coarse-grained pruning, we have,

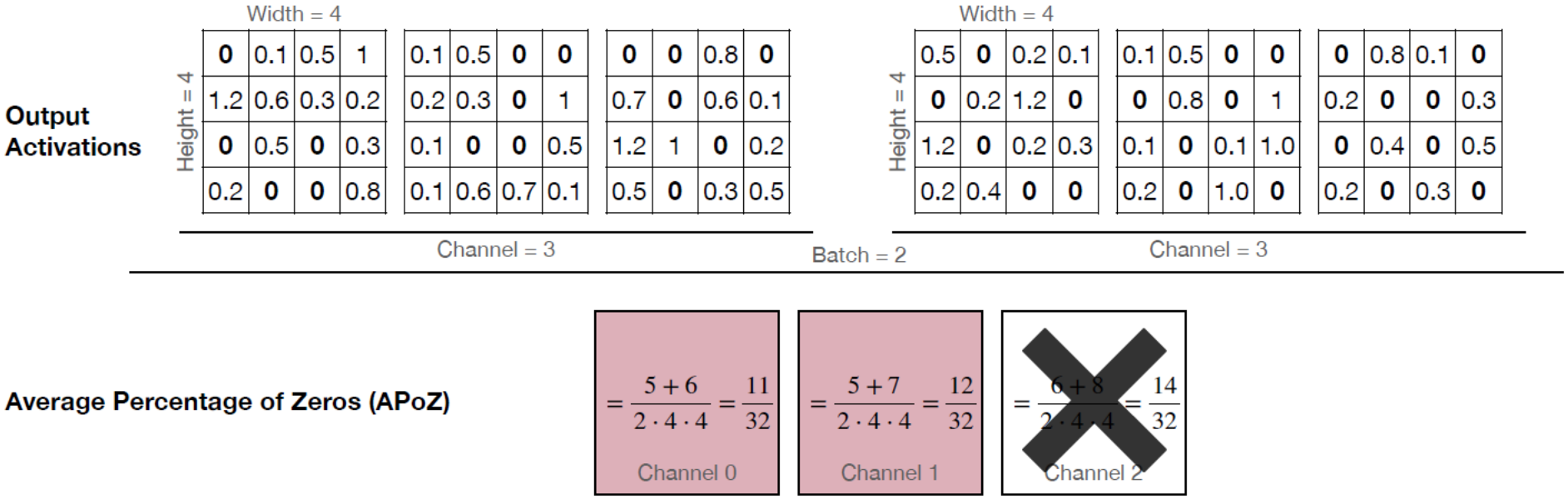
$$importance_{\mathbf{W}^{(S)}} = \sum_{i \in S} |\delta L_i|^2 = \sum_{i \in S} (g_i w_i)^2,$$

where $\mathbf{W}^{(S)}$ is the structural set of parameters

Pruning Method	GFLOPs	Params(10^7)	↓ Error, %
	ResNet-101		
Taylor-FO-BN-40% **(Ours)**	1.76	1.36	25.84
Taylor-FO-BN-50% **(Ours)**	**2.47**	1.78	**24.62**
BN-ISTA v2 [33]	3.69	**1.73**	25.44
Taylor-FO-BN-55% **(Ours)**	**2.85**	**2.07**	**24.05**
BN-ISTA v1 [33]	4.47	2.36	24.73
No pruning	7.80	4.47	**22.63**
Taylor-FO-BN-75% **(Ours)**	**4.70**	**3.12**	22.65
	ResNet-34		
No pruning	3.64	2.18	26.69
Taylor-FO-BN-82% **(Ours)**	2.83	**1.72**	**27.17**
Li *et al.* [22]	**2.76**	1.93	27.80
	VGG11-BN		
No pruning	7.61	13.29	**29.16**
Taylor-FO-BN-50% **(Ours)**	**6.93**	**3.18**	29.35
From scratch [24]	≈**6.93**	≈**3.18**	30.00
Slimming [23], from [24]	≈**6.93**	≈**3.18**	31.38
	DenseNet-201		
No pruning	4.29	2.20	**23.20**
Taylor-FO-BN-60% **(Ours)**	3.02	1.25	23.49
Taylor-FO-BN-36% **(Ours)**	**2.21**	0.90	24.72
No pruning	2.74	**0.76**	25.57

Selection of Neurons (Activations) to Prune

- Similar to weight pruning, remove the **less useful** neuron
  - Recall: Neuron pruning is coarse-grained pruning indeed
- Approach
  - Percentage-of-Zero-Based Pruning, (zero due to ReLU)
  - The smaller APoZ is, the more importance the neuron has



Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures [Hu et al., ArXiv 2017]

First-Order-based Pruning

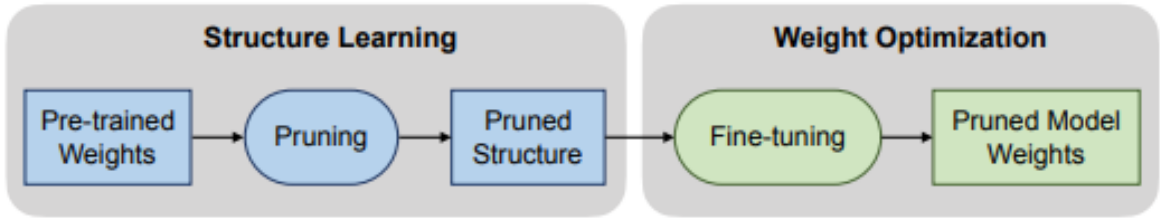
- Minimize the error on loss function introduced by pruning neurons
- Similar to previous Taylor expansion on weights, the induced error of the objective function can be approximated by Taylor series

$$\delta L_i = L(\mathbf{x}; \mathbf{W}) - L(\mathbf{x} \mid x_i = 0; \mathbf{W}) \approx \frac{\partial L}{\partial x_i} x_i$$

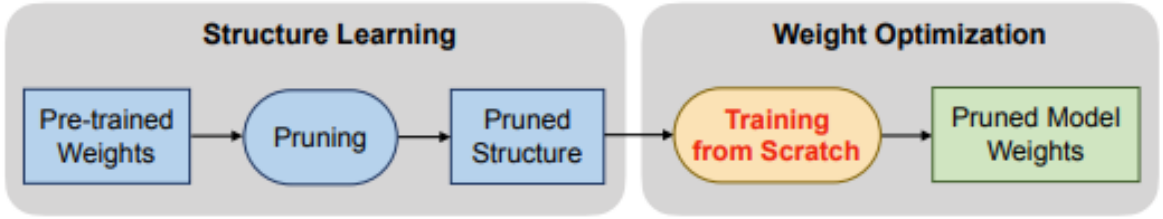
- For a structure set of neuron, (e.g. channel)

$$\left| \delta L_{\mathbf{x}^{(S)}} \right| = \left| \sum_{i \in S} \frac{\partial L}{\partial x_i} x_i \right|$$

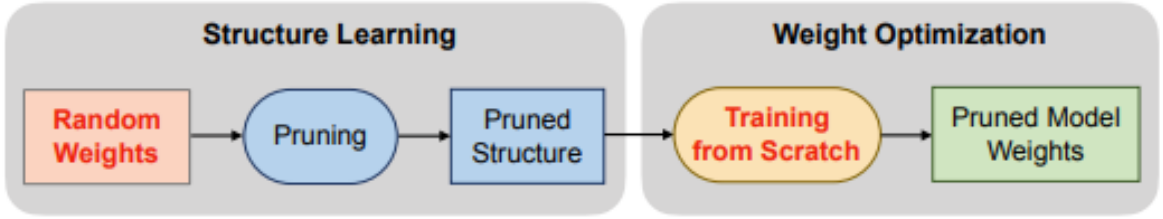
Pruning Convolutional Neural Networks for Resource Efficient Inference [Molchanov et al., ICLR 2017]



(a) traditional network pruning pipeline



(b) network pruning pipeline in (Liu et al. 2019)



(c) our pre-training-free network pruning pipeline

PRUNING AS ARCHITECTURE SEARCH

Questions on Pruning

- Training overhead
  - Do we need fine tuning?
  - No, if you use structure pruning and train from scratch
  - Yes, for unstructured pruning
- Do we need pre-trained model for better pruning result?
  - No, for structure pruning, architecture matters
  - Yes. for unstructured pruning
- 小而密集的網路（small-dense model）並不能達到和大而稀疏的網路（large-sparse model）一樣的準確度。體現了prune的重要性。
  - 小網路較難訓練，也較不容易從大的資料集獲益
  - Knowledge distillation could help

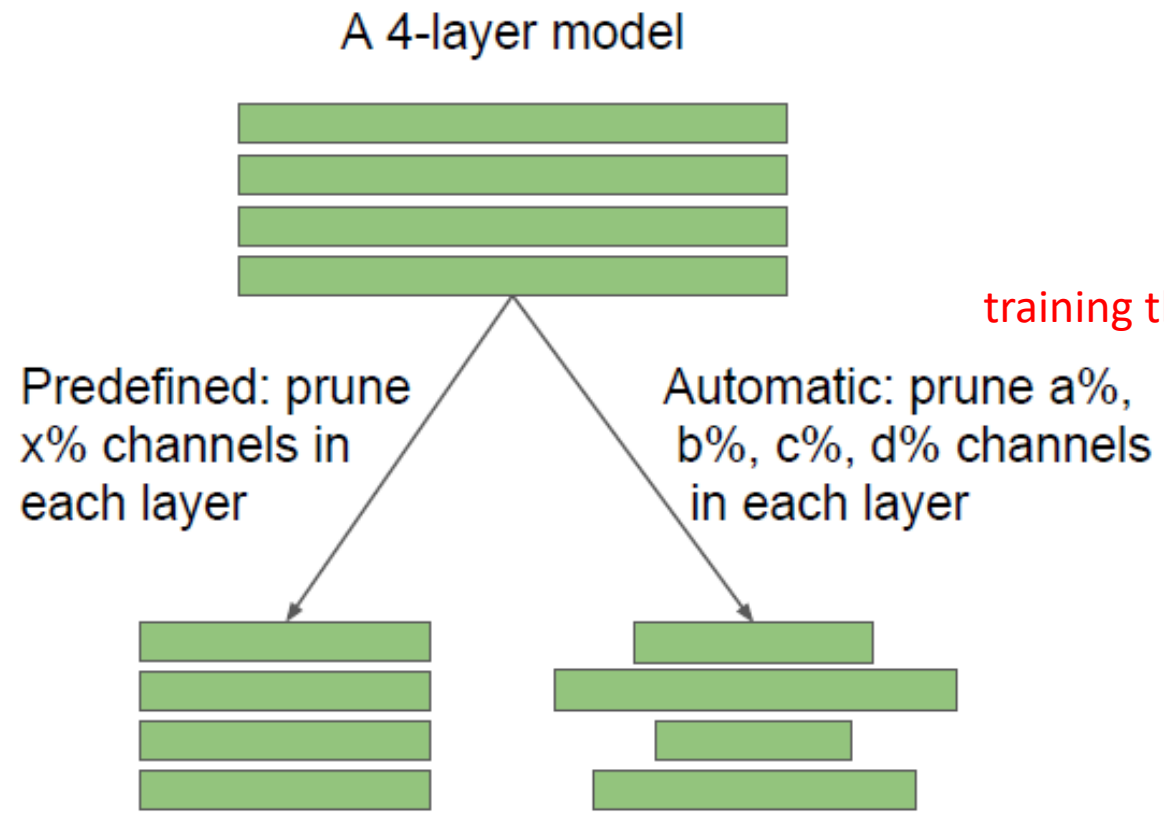
TinyViT: Fast Pretraining Distillation for Small Vision Transformers

Rethinking the Value of Network Pruning

- Train-prune-fine tuning
  - Only useful for unstructured pruning
  - Not necessary for structured pruning
- Structured pruning
  - With predefined target network, directly train the small target model from random initialization
  - With auto-discovered network, train the pruned model from scratch
- **Target architecture matters** for structured pruning
  - No need for pruning for predefined target network architecture
  - Directly train the target network from scratch
- 3 implications
  - large, over-parameterized model is not necessary for efficient final model
  - learned "important" weights of the large model are not necessarily useful for the small pruned model
  - Pruned architecture, instead of weights, is more important
  - (Pruning as a network architecture search)

Source: Rethinking the Value of Network Pruning

Structured pruning



directly training the small target model from random initialization

training the pruned model from scratch

L1-norm based channel pruning[Li, 2017]
ThiNet[Luo, 2017]
Regression based feature reconstruction[He, 2017]

Network slimming [Liu2017]
Sparse structure selection [Huang, 2018]
Non-structured weight pruning [Han, 2015]

Source: Rethinking the Value of Network Pruning

PREDEFINED STRUCTURED PRUNING

Fine Tuned v.s. Training from Scratch

- Scratch-B: longer training time (50% cut, 2X training time)
- Scratch-E: same training time
- The Pruned Model : the list of predefined target models

Model 直接縮小後，從頭訓練，時間加倍，效果比fine tuned好

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.

Source: Rethinking the Value of Network Pruning

AUTOMATIC STRUCTURED PRUNING

Fine Tuned v.s. Training from Scratch

(不同層用不同pruning rate的 方法，結果類似)

Dataset	Model	Unpruned	Prune Ratio	Fine-tuned	Scratch-E	Scratch-B
CIFAR-10	VGG-19	93.53 (±0.16)	70%	93.60 (±0.16)	93.30 (±0.11)	**93.81** (±0.14)
	PreResNet-164	95.04 (±0.16)	40%	94.77 (±0.12)	94.70 (±0.11)	**94.90** (±0.04)
			60%	94.23 (±0.21)	94.58 (±0.18)	**94.71** (±0.21)
	DenseNet-40	94.10 (±0.12)	40%	94.00 (±0.20)	93.68 (±0.18)	**94.06** (±0.12)
			60%	**93.87** (±0.13)	93.58 (±0.21)	93.85 (±0.25)
CIFAR-100	VGG-19	72.63 (±0.21)	50%	72.32 (±0.28)	71.94 (±0.17)	**73.08** (±0.22)
	PreResNet-164	76.80 (±0.19)	40%	76.22 (±0.20)	76.36 (±0.32)	**76.68** (±0.35)
			60%	74.17 (±0.33)	75.05 (± 0.08)	**75.73** (±0.29)
	DenseNet-40	73.82 (±0.34)	40%	**73.35** (±0.17)	73.24 (±0.29)	73.19 (±0.26)
			60%	72.46 (±0.22)	72.62 (±0.36)	**72.91** (±0.34)
ImageNet	VGG-11	70.84	50%	68.62	70.00	**71.18**

Table 4: Results (accuracy) for Network Slimming (Liu et al., 2017). "Prune ratio" stands for total percentage of channels that are pruned in the whole network. The same ratios for each model are used as the original paper.

Source: Rethinking the Value of Network Pruning

AUTOMATIC STRUCTURED PRUNING

Fine Tuned v.s. Training from Scratch

(不同層用不同pruning rate的 方法，結果類似)

Dataset	Model	Unpruned	Pruned Model	Pruned	Scratch-E	Scratch-B
ImageNet	ResNet-50	76.12	ResNet-41	75.44	75.61	**76.17**
			ResNet-32	74.18	73.77	**74.67**
			ResNet-26	71.82	72.55	**73.41**

Table 5: Results (accuracy) for residual block pruning using Sparse Structure Selection (Huang & Wang, 2018). In the original paper no fine-tuning is required so there is a "Pruned" column instead of "Fine-tuned" as before.

Pruning or Not Pruning

- Training predefined target models
  - Higher accuracy
  - Faster training
  - No need to implement pruning criteria and procedure
  - No need to tune hyper parameter in the pruning procedure
- Pruning works
  - When a pre-trained large model is already given and little or no training budget is available
  - need to obtain multiple models of different sizes, in this situation one can train a large model and then prune it by different ratios

Source: Rethinking the Value of Network Pruning

RETHINKING AGAIN THE VALUE OF NETWORK PRUNING - A DYNAMICAL ISOMETRY PERSPECTIVE

- 3 steps of Pruning
  - Pretraining, pruning, fine-tuning
  - Focus on fine-tuning
- weight removal in pruning breaks dynamical isometry,
  - which fundamentally answers for the performance gap between a large finetuning LR and a small one.
  - necessary to recover the dynamical isometry before finetuning with large LR
- Inherit weight in structured pruning is important
  - “no value of inheriting weights” value question due to improperly using a small finetuning LR
  - find simply using a larger finetuning LR (10^{-2} vs. 10^{-3} and decay it) can significantly improve the final performance
- 動力等距(dynamical isometry)
  - 如果input-output Jacobian矩陣的奇異值分佈滿足全部在1附近，那麼這個條件就叫做動力等距
  - 原則上Jacobian矩陣衡量的是輸出是如何隨著輸入的擾動而變化，如果在初始化的時候能夠滿足動力等距條件，則我們認為這個網絡有很好的性質，可以期待它的訓練效果非常好，容易訓練。

Table 1: Top-1 accuracy comparison of different implementations of the L_1-norm pruning (Li et al., 2017) on ImageNet. We adopt the torchvision models as unpruned models for fair comparison. ResNet-34-A speedup: 1.18×. ResNet-34-B speedup: 1.32×. The results of (Li et al., 2017) and (Liu et al., 2019) are directly cited from their papers. The best cases in our training from scratch and pruning are randomly repeated for 3 times (± indicates stddev) to prevent random variation.

Implementation	Unpruned (%)	Pruned model	Scratch (%)	Pruned-Finetuned (%)	Finetuning LR schedule
(Li et al., 2017)	73.23	ResNet-34-A	(Not reported)	72.56	20 epochs, initial 10^{-3}, fixed
		ResNet-34-B	(Not reported)	72.17	20 epochs, initial 10^{-3}, fixed
(Liu et al., 2019)	73.31	ResNet-34-A	**73.03**	72.56	20 epochs, initial 10^{-3}, fixed
		ResNet-34-B	**72.91**	72.29	20 epochs, initial 10^{-3}, fixed
Our rerun	73.31	ResNet-34-A	$73.51_{\pm 0.12}$	72.91	20 epochs, initial 10^{-3}, fixed
				72.94	90 epochs, initial 10^{-3}, fixed
				73.88	90 epochs, initial 10^{-3}, decay
				$73.92_{\pm 0.03}$	90 epochs, initial 10^{-2}, decay
Our rerun	73.31	ResNet-34-B	$73.16_{\pm 0.12}$	72.50	20 epochs, initial 10^{-3}, fixed
				72.58	90 epochs, initial 10^{-3}, fixed
				73.61	90 epochs, initial 10^{-3}, decay
				$73.62_{\pm 0.04}$	90 epochs, initial 10^{-2}, decay

Table 2: Top-1 accuracy comparison between scratch training ("Scratch") and L_1-norm pruning (Li et al., 2017) on ImageNet. "PR" means pruning ratio. †We adopt the official torchvision models as unpruned models. "Finetuned-1" and "Finetuned-2" refers to two finetuning LR schedules ("Finetuned-1": 90 epochs, initial 10^{-3}, decay 30/60/75; "Finetuned-2": 90 epochs, initial 10^{-2}, decay 45/68). Best results are in **bold**, second best underlined, each averaged by 3 random runs.

Network	PR	Params reduc. (%)	FLOPs reduc. (%)	Scratch (%)	Pruned-Finetuned-1 (%)	Pruned-Fintuned-2 (%)
ResNet-18	0	0	0	69.76†	/	/
	0.1	9.56	9.58	$70.15_{\pm 0.02}$	$70.43_{\pm 0.02}$	$\mathbf{70.48}_{\pm \mathbf{0.10}}$
	0.3	28.32	28.18	$68.90_{\pm 0.10}$	$69.29_{\pm 0.07}$	$\mathbf{69.54}_{\pm \mathbf{0.09}}$
	0.5	47.03	46.20	$67.03_{\pm 0.01}$	$67.36_{\pm 0.03}$	$\mathbf{67.71}_{\pm \mathbf{0.05}}$
	0.7	65.99	64.93	$64.21_{\pm 0.10}$	$63.72_{\pm 0.03}$	$\mathbf{64.45}_{\pm \mathbf{0.04}}$
	0.9	84.75	83.52	$\mathbf{56.70}_{\pm \mathbf{0.17}}$	$53.49_{\pm 0.10}$	$55.89_{\pm 0.11}$
	0.95	89.51	88.03	$\mathbf{51.83}_{\pm \mathbf{0.14}}$	$44.46_{\pm 0.15}$	$49.99_{\pm 0.10}$
ResNet-34	0	0	0	73.31†	/	/
	0.1	9.84	9.92	$73.53_{\pm 0.10}$	$73.86_{\pm 0.05}$	$\mathbf{74.04}_{\pm \mathbf{0.05}}$
	0.3	29.15	29.26	$72.50_{\pm 0.17}$	$73.11_{\pm 0.06}$	$\mathbf{73.31}_{\pm \mathbf{0.08}}$
	0.5	48.41	48.12	$71.27_{\pm 0.03}$	$71.71_{\pm 0.06}$	$\mathbf{71.85}_{\pm \mathbf{0.07}}$
	0.7	67.95	67.63	$68.69_{\pm 0.10}$	$68.90_{\pm 0.08}$	$\mathbf{69.33}_{\pm \mathbf{0.04}}$
	0.9	87.26	86.97	$62.08_{\pm 0.12}$	$60.34_{\pm 0.03}$	$\mathbf{62.26}_{\pm \mathbf{0.06}}$
	0.95	92.16	91.69	$\mathbf{57.21}_{\pm \mathbf{0.15}}$	$52.83_{\pm 0.11}$	$56.69_{\pm 0.23}$
VGG11_BN	0	0	0	70.37†	/	/
	0.1	9.19	17.78	$68.51_{\pm 0.04}$	$71.45_{\pm 0.07}$	$\mathbf{71.74}_{\pm \mathbf{0.04}}$
	0.3	26.80	47.63	$66.60_{\pm 0.11}$	$70.00_{\pm 0.03}$	$\mathbf{70.53}_{\pm \mathbf{0.05}}$
	0.5	43.86	70.56	$65.85_{\pm 0.13}$	$67.34_{\pm 0.05}$	$\mathbf{68.01}_{\pm \mathbf{0.10}}$
	0.7	60.54	87.03	$61.56_{\pm 0.06}$	$62.14_{\pm 0.09}$	$\mathbf{63.14}_{\pm \mathbf{0.08}}$
	0.9	76.50	96.49	$48.34_{\pm 0.12}$	$45.11_{\pm 0.07}$	$\mathbf{48.52}_{\pm \mathbf{0.06}}$
	0.95	80.49	97.76	$35.47_{\pm 0.09}$	$33.50_{\pm 0.10}$	$\mathbf{38.47}_{\pm \mathbf{0.13}}$

- Strong L2 regularization as a drop-in remedy
  - push the unimportant filters to rather close to zero first, before permanently removing them

Table 8: Test accuracies (%) of pruning ResNet56 on CIFAR10. Unpruned accuracy: 93.78%. Each setting is randomly run 3 times. "Acc. gain" means the accuracy gain of initial LR 10^{-2} over 10^{-3}.

Pruning ratio r	0.5	0.7	0.9	0.95
Sparsity/Speedup	49.82%/1.99×	70.57%/3.59×	90.39%/11.41×	95.19%/19.31×
	Finetuning LR schedule: 120 epochs, initial 10^{-2}, decay 60/90			
Train from scratch	$92.78_{\pm 0.23}$	$92.11_{\pm 0.12}$	$88.36_{\pm 0.20}$	$84.60_{\pm 0.14}$
L_1 (Li et al., 2017)	$93.51_{\pm 0.07}$	$92.26_{\pm 0.17}$	$88.71_{\pm 0.15}$	$84.63_{\pm 0.28}$
L_1 + OrthP	$93.36_{\pm 0.19}$	$91.96_{\pm 0.06}$	$86.01_{\pm 0.34}$	$82.62_{\pm 0.05}$
StrongReg	$\mathbf{93.55_{\pm 0.06}}$	$\mathbf{92.38_{\pm 0.09}}$	$\mathbf{89.24_{\pm 0.16}}$	$\mathbf{85.90_{\pm 0.19}}$
	Finetuning LR schedule: 120 epochs, initial 10^{-3}, decay 80			
L_1 (Li et al., 2017)	$93.12_{\pm 0.10}$	$91.77_{\pm 0.11}$	$87.57_{\pm 0.09}$	$83.10_{\pm 0.12}$
StrongReg	$\mathbf{93.44_{\pm 0.06}}$	$\mathbf{91.96_{\pm 0.11}}$	$\mathbf{88.69_{\pm 0.19}}$	$\mathbf{85.45_{\pm 0.25}}$
Acc. gain. (L_1)	0.39	0.49	1.14	1.53
Acc. gain. (StrongReg)	**0.11**	**0.42**	**0.55**	**0.45**

MorphNet: Constrained based Pruning

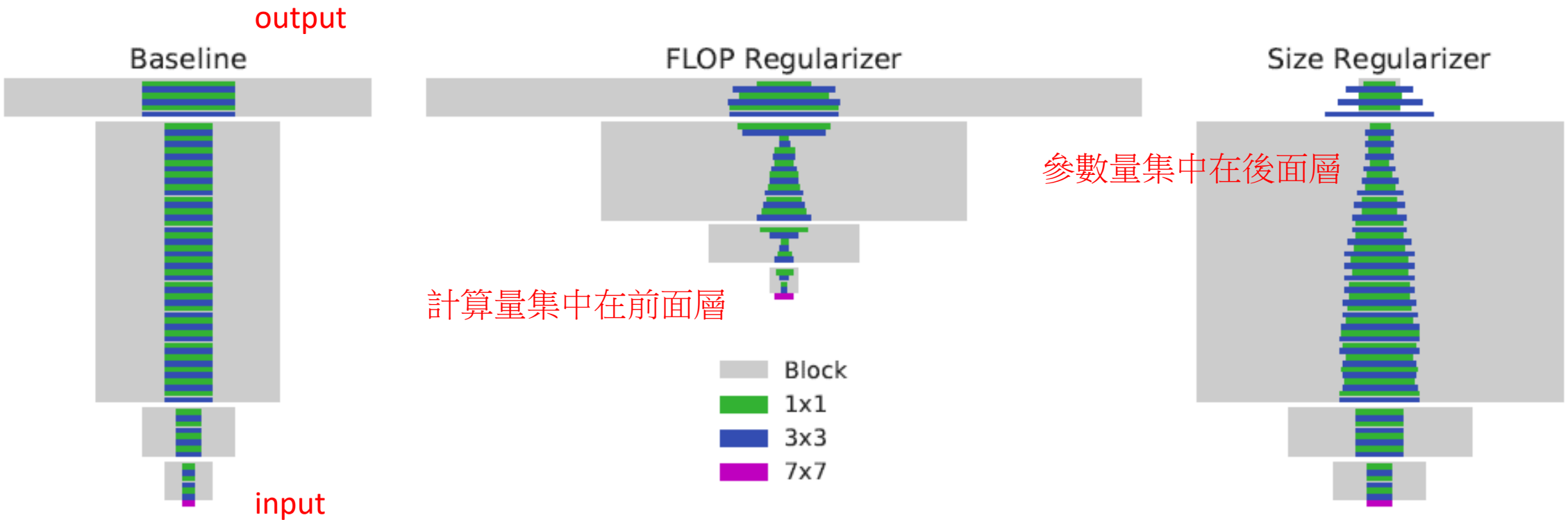


Figure 1. ResNet101 based models with similar performance (around 0.426 MAP on JFT, see Section 5). A structure obtained by shrinking ResNet101 uniformly by a $\omega = 0.5$ factor (left), and structures learned by MorphNet when targeting FLOPs (center) or model size (*i.e.*, number of parameters; right). Rectangle width is proportional to the number of channels in the layer and residual blocks are denoted in gray. 7×7, 3×3, and 1×1 convolutions are in purple, blue and green respectively. The purple bar at the bottom of each model is thus the input layer. Learned structures are markedly different from the human-designed model and from each other. The FLOP regularizer primarily prunes the early, compute-heavy layers. It notably learns to *remove whole layers* to further reduce computational burden. By contrast, the model size regularizer focuses on removal of 3×3 convolutions at the top layers as those are the most parameter-heavy.

MorphNet: only optimize over output widths

- iteratively alternating between a sparsifying regularizer and a uniform width multiplier

Algorithm 1 The MorphNet Algorithm

1: Train the network to find
$\theta^* = \underset{\theta}{\operatorname{argmin}}\{\mathcal{L}(\theta) + \lambda \mathcal{G}(\theta)\}$, for suitable λ.
2: Find the new widths $O'_{1:M}$ induced by θ^*.
3: Find the largests ω such that $\mathcal{F}(\omega \cdot O'_{1:M}) \leq \zeta$.
4: Repeat from Step 1 for as many times as desired, setting
$O^{\circ}_{1:M} = \omega \cdot O'_{1:M}$.
5: **return** $\omega \cdot O'_{1:M}$.

Shrinking:
a sparsifying regularizer
w < 1, uniform width multiplier

Expanind: w > 1
uniform width multiplier

網路直接乘上width factor去滿足條件

MorphNet: Constraints

- FLOPs and model size are bilinear in the number of inputs and outputs of that layer

$$\mathcal{F}(\text{layer } L) = C(w_L, x_L, y_L, z_L, f_L, g_L) \cdot I_L O_L$$

Input size, Output size, Filter size, I/O channel number

  - FLOPS $C(w, x, y, z, f, g) = 2yzfg$
  - Model size $C(w, x, y, z, f, g) = fg$
  - After pruning, indicate which neuron is zero

$$\mathcal{F}(\text{layer } L) = C \sum_{i=0}^{I_L - 1} A_{L,i} \sum_{j=0}^{O_L - 1} B_{L,j}$$

  - Total constrained quality

$$\mathcal{F}(O_{1:M}) = \sum_{L=1}^{M+1} \mathcal{F}(\text{layer } L)$$

A, B: number of alived channels in a layer

MorphNet: Regularization

- Similar to network slimming, use γ in BN for channel pruning

$$\mathcal{G}(\theta) = \sum_{L=1}^{M+1} \mathcal{G}(\theta, \text{layer } L).$$

$$\mathcal{G}(\theta, \text{layer } L) = C \sum_{i=0}^{I_L-1} |\gamma_{L-1,i}| \sum_{j=0}^{O_L-1} B_{L,j} + C \sum_{i=0}^{I_L-1} A_{L,i} \sum_{j=0}^{O_L-1} |\gamma_{L,j}|,$$

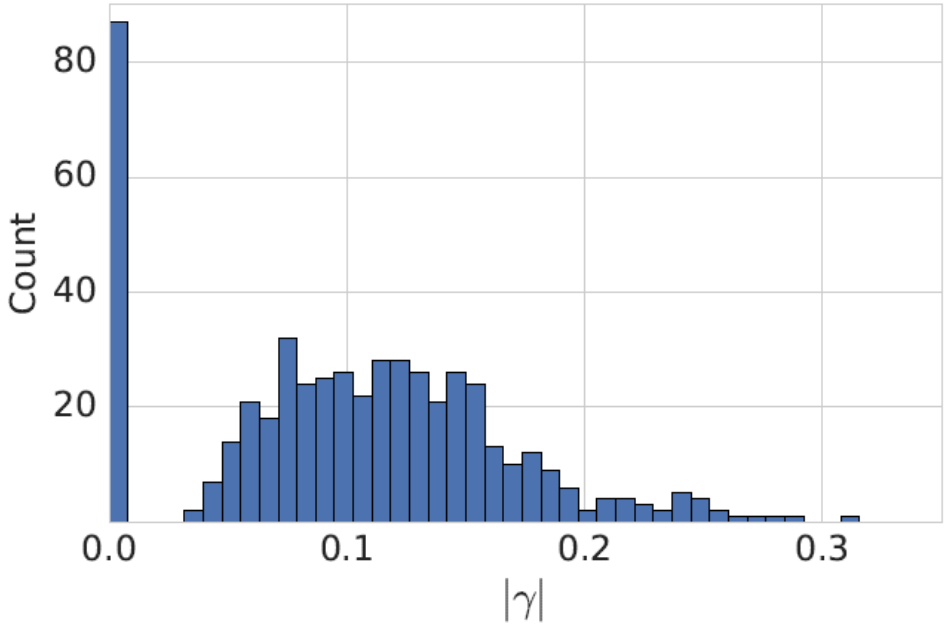


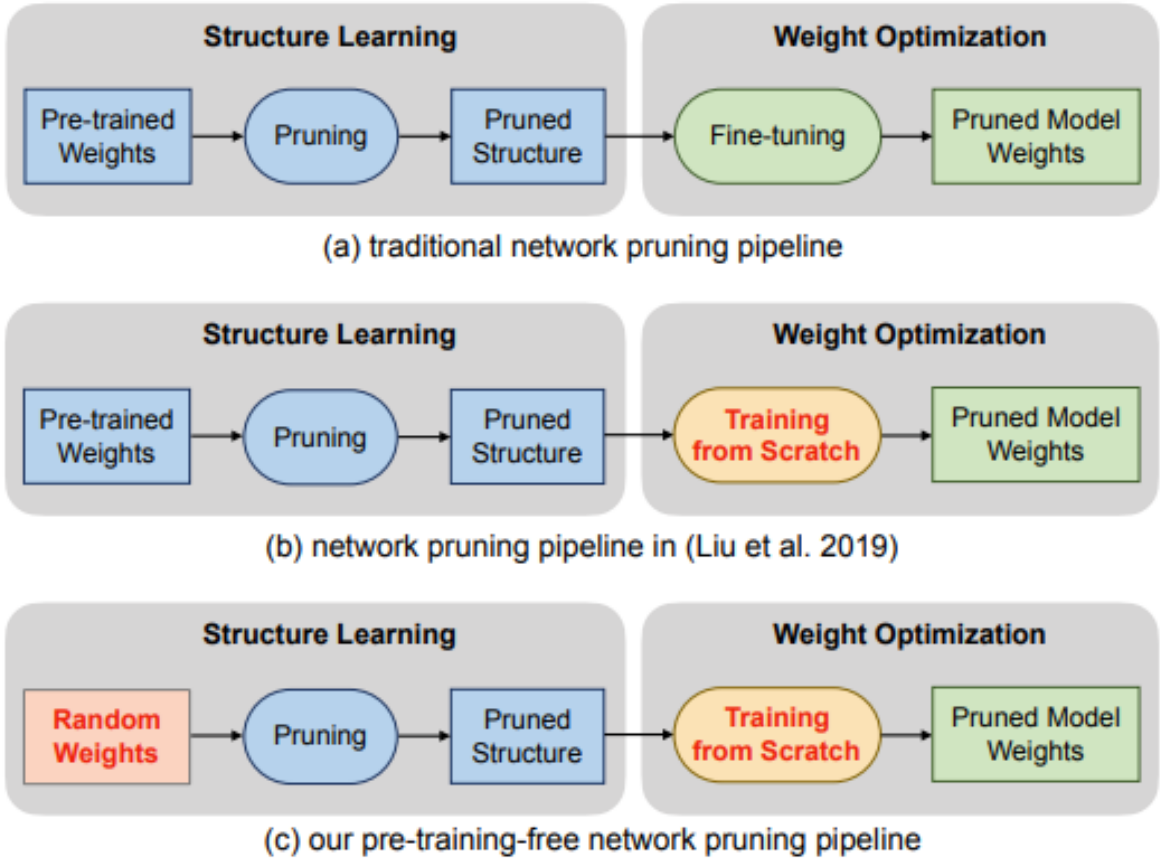
Figure 2. A histogram of γ for one of the ResNet101 bottleneck layers when trained with a FLOP regularizer. Some of the $|\gamma|$'s are zeroed out, and are separated by a clear gap from the nonzero $|\gamma|$'s.

MorphNet: Improved Performance at No Cost

Network	Baseline	MorphNet	Relative Gain
Inception V2	74.1	75.2	+1.5%
MobileNet 50%	57.1	58.1	+1.78%
MobileNet 25%	44.8	45.9	+2.58%
ResNet101	0.477	0.487	+2.1%
AudioResNet	0.182	0.186	+2.18%

Table 2. The result of applying MorphNet to a variety of datasets and model architectures while maintaining FLOP cost.

Pruning from Scratch



Learning structure is easier than learning weight

1. optimization objective, similar to Network slimming

$$\min_{\Lambda} \quad \sum_{i}^{N} \mathcal{L}(f(x_i; \boldsymbol{W}, \Lambda), y_i) + \gamma \sum_{j}^{K} |\boldsymbol{\lambda}_j|_1$$

$$s.t. \quad \mathbf{0} \preceq \boldsymbol{\lambda}_j \preceq \mathbf{1}, \quad \forall j = 1, 2, \cdots, K,$$

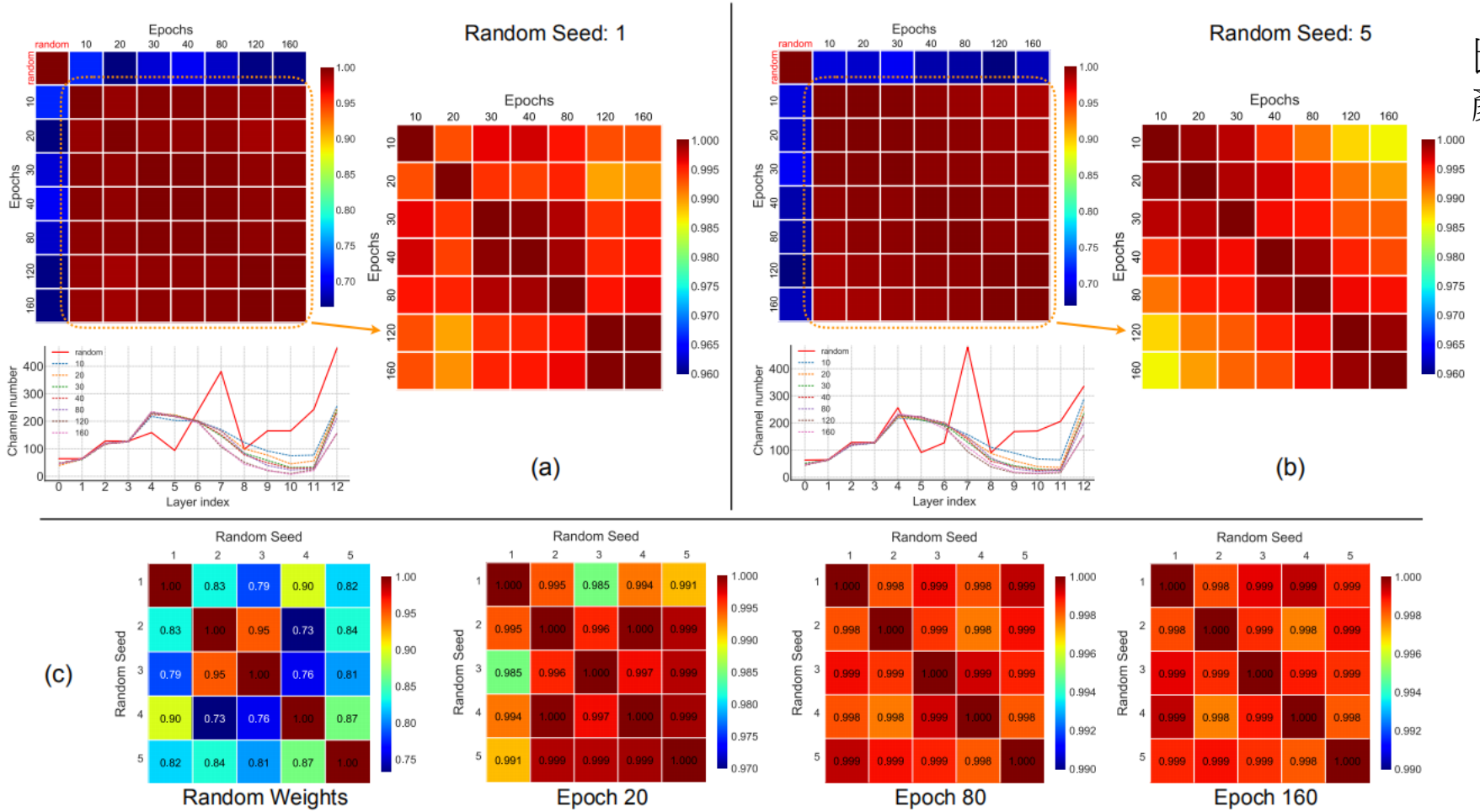
2. naive L1-normwill encourage the gates to be zeroes unconstrainedly

3. do not update the weights during channel importance learning; use randomly initialized weights without relying on pretraining

4. given a target sparsity ratio r, the regularization term

$$\Omega(\Lambda) = \left(\frac{\sum_j |\boldsymbol{\lambda}_j|_1}{\sum_j C_j} - r\right)^2$$

利用所有gate的平均值來近似其整體的稀疏比例，
用平方範數將稀疏度逼近為預先定義的比值r



比較了不同剪枝模型的結構相似度
顏色越紅關聯度越高。

預訓練階段隨著權重的更新，
剪枝結構的搜索空間逐步縮小，
可能會限制模型的潛在性能。

另一方面，隨機初始化的權重使得剪枝算法可以探索更多樣化的剪枝結構

在預訓練階段10個epochs 的權重更新之後，剪枝模型趨向於同質化

Figure 2: Exploring the effect of pre-trained weights on the pruned structures. All the pruned models are required to reduce 50% FLOPS of the original VGG16 on CIFAR10 dataset. (a) (Top-left) We display the correlation coefficient matrix of the pruned models directly learned from randomly initialized weights ("**random**") and other pruned models based on different checkpoints during pre-training ("**epochs**"). (Right) We display the correlation coefficient matrix of pruned structures from pre-trained weights on a finer scale. (Bottom-left) We show the channel numbers of each layer of different pruned structures. Red line denotes the structure from random weights. (b) Similar results from the experiment with a different random seed. (c) We display correlation coefficient matrices of all the pruned structures from five different random seeds. We mark the names of initialized weights used to get pruned structures below.

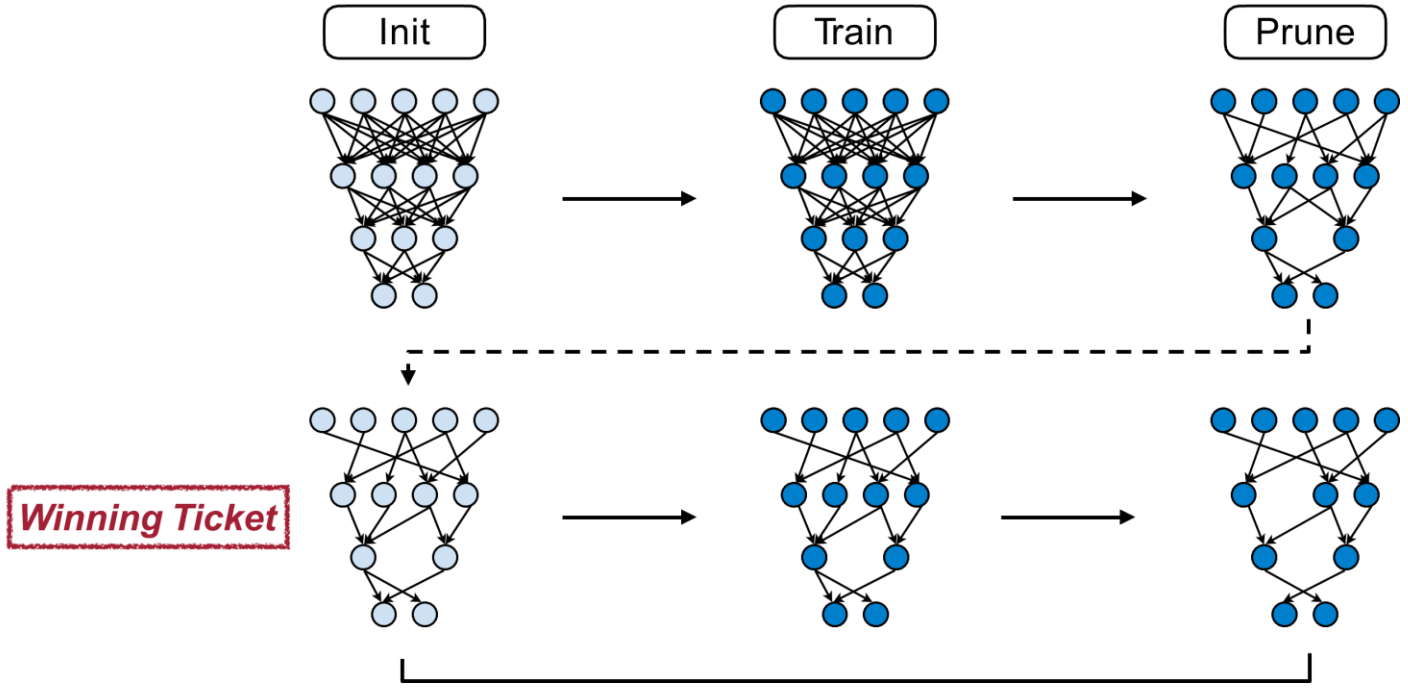
Table 3: Network pruning results on ImageNet dataset. For uniform channel expansion models, we expand the channels of each layer with a fixed ratio m, denoted as "$m\times$". "Baseline 1.0×" stands for the original full model. "Params" column summarizes the sizes of the total parameters of each pruned models.

	Model	Params	Latency	FLOPS	Top-1 Acc (%)
MobileNet-V1	Uniform 0.5×	1.3M	20ms	150M	63.3
	Uniform 0.75×	3.5M	23ms	325M	68.4
	Baseline 1.0×	4.2M	30ms	569M	70.9
	NetAdapt	–	–	285M	70.1
	AMC	2.4M	25ms	294M	70.5
	Ours 0.5×	1.0M	20ms	150M	**65.5**
	Ours 0.75×	1.9M	21ms	286M	**70.7**
	Ours 1.0×	4.0M	23ms	567M	**71.6**
t-V2	Uniform 0.75×	2.6M	39ms	209M	69.8
	Baseline 1.0×	3.5M	42ms	300M	71.8
	Uniform 1.3×	5.3M	43ms	509M	**74.4**
MobileNet-	AMC	2.3M	41ms	211M	70.8
	Ours 0.75×	2.6M	37ms	210M	**70.9**
	Ours 1.0×	3.5M	41ms	300M	**72.1**
	Ours 1.3×	4.5M	42ms	511M	74.1
ResNet50	Uniform 0.5×	6.8M	50ms	1.1G	72.1
	Uniform 0.75×	14.7M	61ms	2.3G	74.9
	Uniform 0.85×	18.9M	62ms	3.0G	75.9
	Baseline 1.0×	25.5M	76ms	4.1G	76.1
	ThiNet-30	–	–	1.2G	72.1
	ThiNet-50	–	–	2.1G	74.7
	ThiNet-70	–	–	2.9G	75.8
	SFP	–	–	2.9G	75.1
	CP	–	–	2.0G	73.3
	Ours 0.5×	4.6M	44ms	1.0G	**72.8**
	Ours 0.75×	9.2M	52ms	2.0G	**75.6**
	Ours 0.85×	17.9M	60ms	3.0G	**76.7**
	Ours 1.0×	21.5M	67ms	4.1G	**77.2**

Lottery Ticket Hypothesis

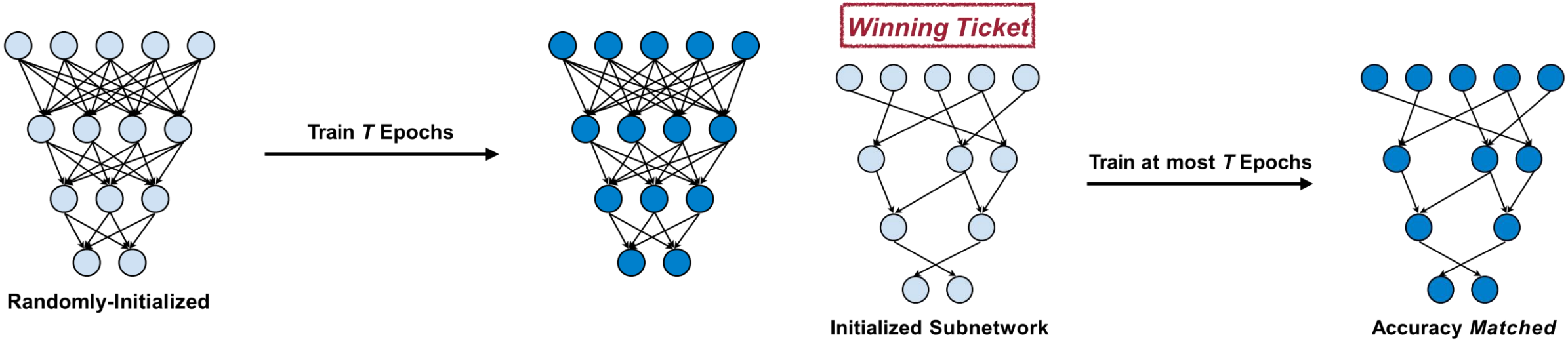
- Can we directly train this sparse neural network from scratch?
  - the architectures uncovered by pruning are harder to train from the start,
  - reaching lower accuracy than the original networks
- Lottery ticket hypothesis

Train dense, prune, rewind the rest, repeat this flow



A randomly-initialized, dense neural network contains ***a subnetwork*** *that is initialized such that—when* ***trained in isolation****—it can* ***match the test accuracy*** *of the original network after training for* ***at most the same number of iterations****.*

—The Lottery Ticket Hypothesis

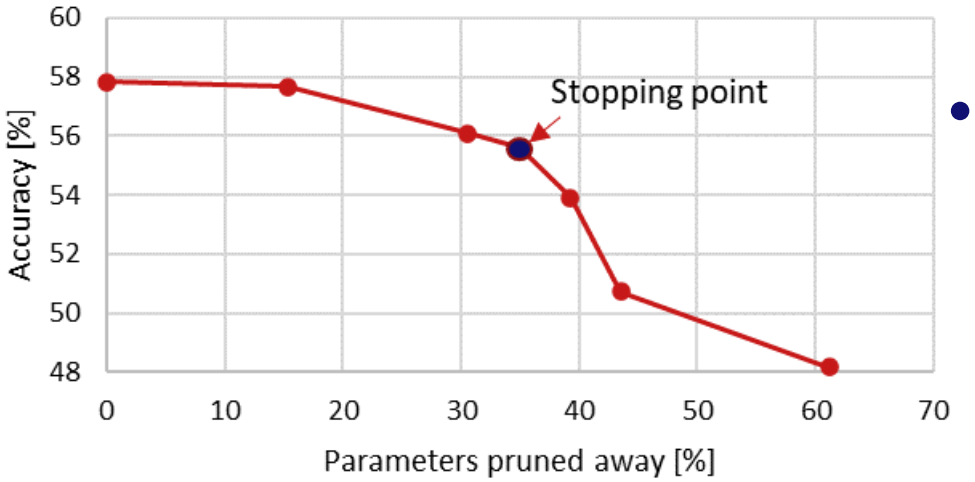


Train dense, prune, rewind the rest, repeat this flow

The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks [Frankle et al., ICLR 2019]

Scenario 1 You only have a model: How to prune? (aka. Data free pruning)

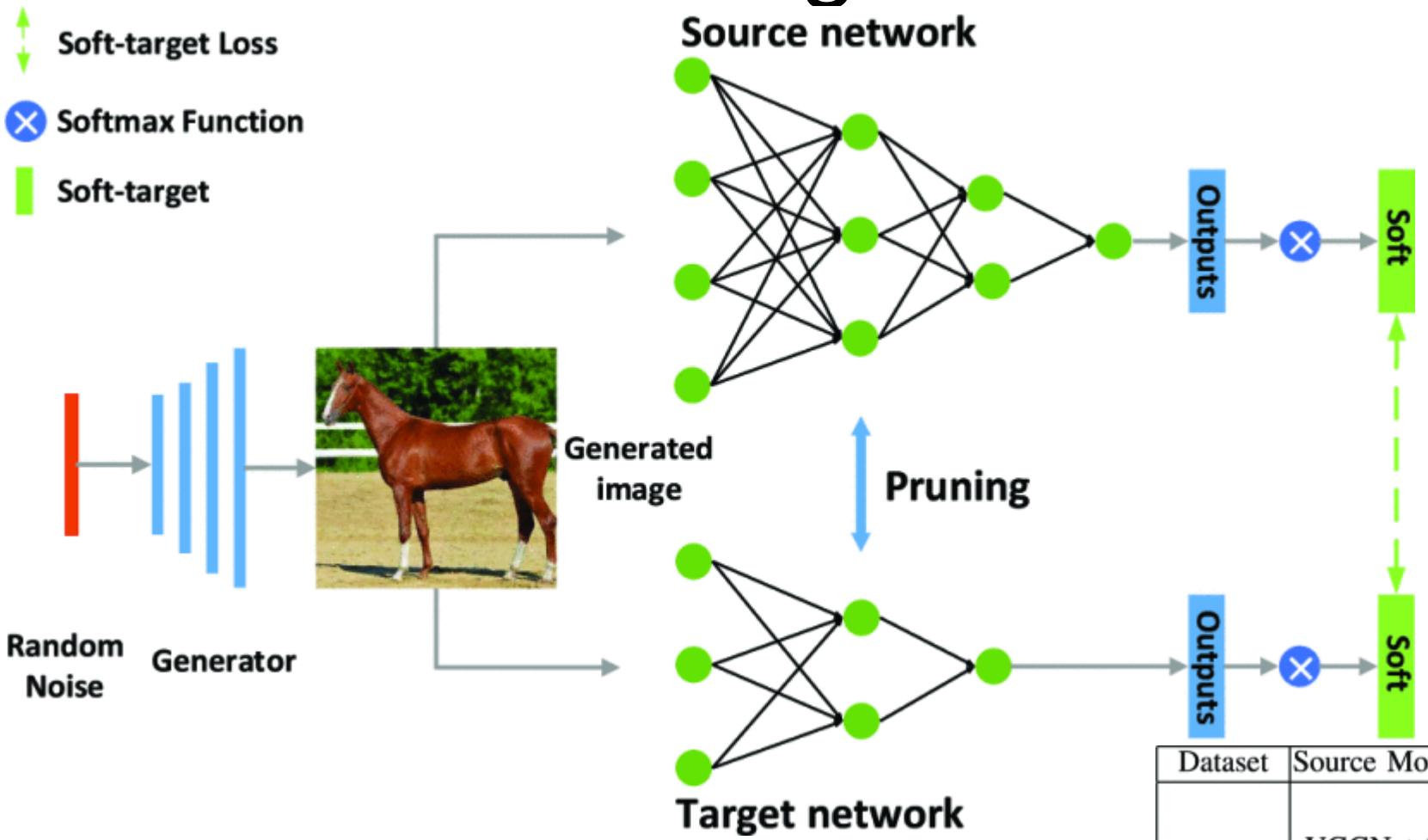
- Naïve pruning:
  - Remove weights based on magnitude, weights close to zero are removed
  - No well-founded theory, error increases rapidly
- Data-Free parameter pruning based upon weight similarity or sensitivity



AlexNet

- Data-Free pruning about 1.5x Compression
  - 61M => 39.6M parameters (151MB)
  - Accuracy reduces with 2.24%

Data Free Pruning: Create Data



Use GAN to generate pseudo data + Knowledge transfer
75% size reduction, 35% FLOPS reduction

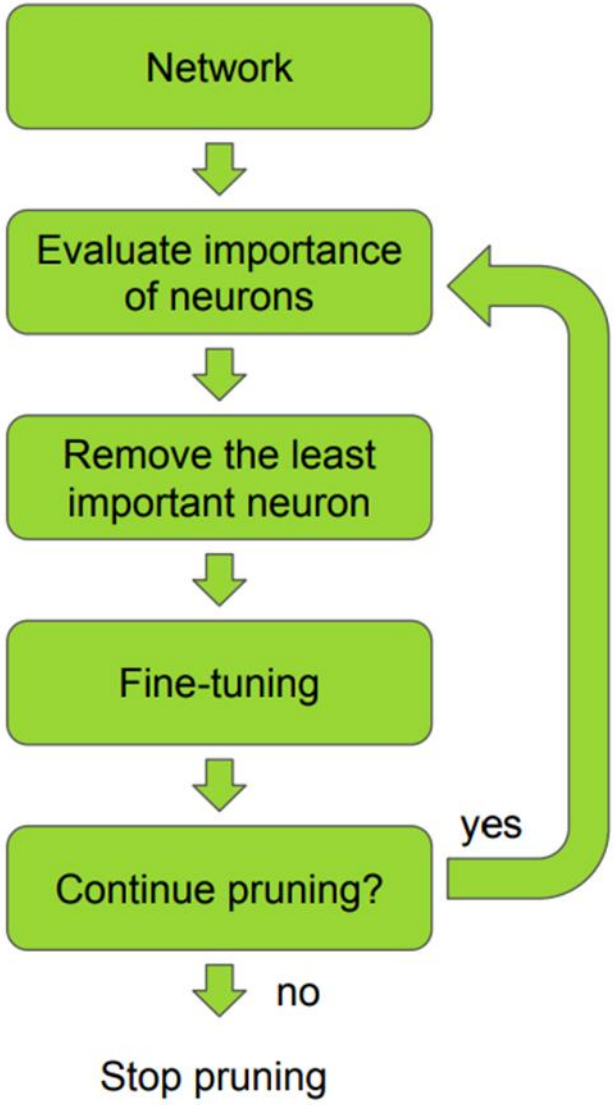
Dataset	Source Model	Algorithm	Target Model	Para	FLOPs	Accuracy
CIFAR-10	VGGNet-16	BP	VGGNet-16	14.72M	0.31G	93.17%
		DFL [13]	VGGNet-13	9.41M	0.23G	91.03%
		DFNP	VGGNet-16-A	3.13M	0.21G	92.16%
		DFNP	VGGNet-16-B	3.13M	0.21G	92.43%
	VGGNet-19	BP	VGGNet-19	20.04M	0.40G	93.34%
		DFL [13]	VGGNet-13	9.41M	0.23G	91.16%
		DFNP	VGGNet-19-A	4.73M	0.26G	92.55%
		DFNP	VGGNet-19-B	4.73M	0.26G	92.78%

Scenario 2 You have data: How to prune aggressively?

- With access to training data, you can do a lot more
  - (optional)
    - Train your network differently such that you have more zero weights
  - Prune
  - Retrain you network after pruning to fix the errors

How to Select Your Pruning Approach?

- Target: pruning granularity
  - Model size
    - Unstructured pruning (post-training pruning)
  - Execution speed
    - Structured pruning (pruning, then 2X training time from scratch)
- Training overhead
  - No training or fine-tuning
    - Post-training pruning (train then prune, + fine-tuning)
  - Medium training overhead
    - Pruning from scratch (prune then train)
  - High training overhead
    - Pruning during training (iterative training + pruning)



- Metrics for pruning (aka. How to define importance?)
  - L1, L2 norm
  - Threshold decision
    - Heuristic or optimization based
    - Soft or hard
    - Resource constrained or not
    - Static (training) or dynamic (runtime)
- Metrics target
  - Weight, activation
  - Gamma in BN
  - Add regularization term to help train these targets

On Network Pruning

- Pretrained models (看參數大小，直接砍，再反覆訓練)
  - Threshold on L1 or L2 norm of weights
- Trained with regularization (讓參數重新調整，更容易砍)
  - **Gamma from BN**, regularization on Gamma
  - L1 or L2 regularization on weights
- Pruning from scratch (不用訓練，直接砍)
  - **pruning before training**
  - Pruning and then train from scratch
- Constrained driven pruning
  - FLOPS
  - Model size

Summary

- Sparse weight pruning
  - 不像表面看起來壓縮率高，儲存與計算都有overhead
  - Recommend vector or filter pruning
- Pruning from scratch works as well as other methods
  - Longer epochs are enough
  - Structure pruning 對網路影響遠小於quantization，基本上是一種 architecture search
- Large model is robust for pruning
  - Small model is more sensitive for pruning
- Add fine-tuning to restore accuracy

APPENDIX

Train Large, Then Compress: Rethinking Model Size for Efficient Training and Inference of Transformers

- even though smaller Transformer models execute faster per iteration, wider and deeper models converge in significantly fewer steps

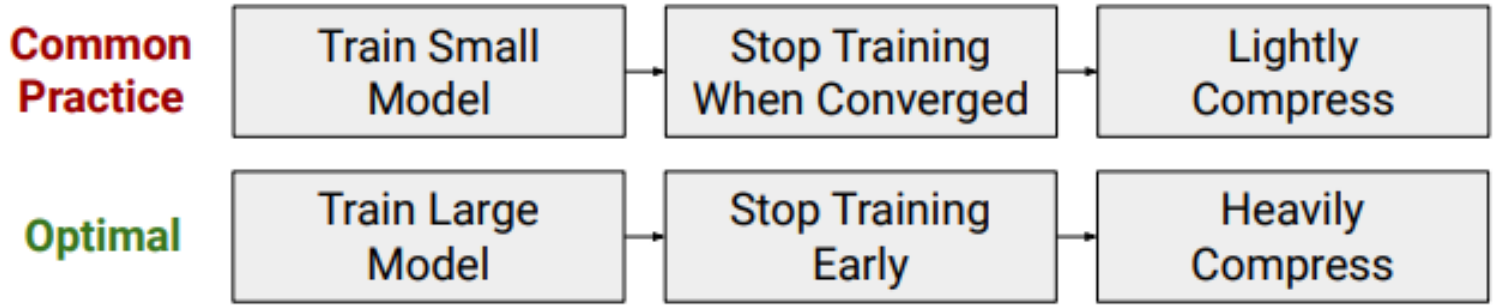
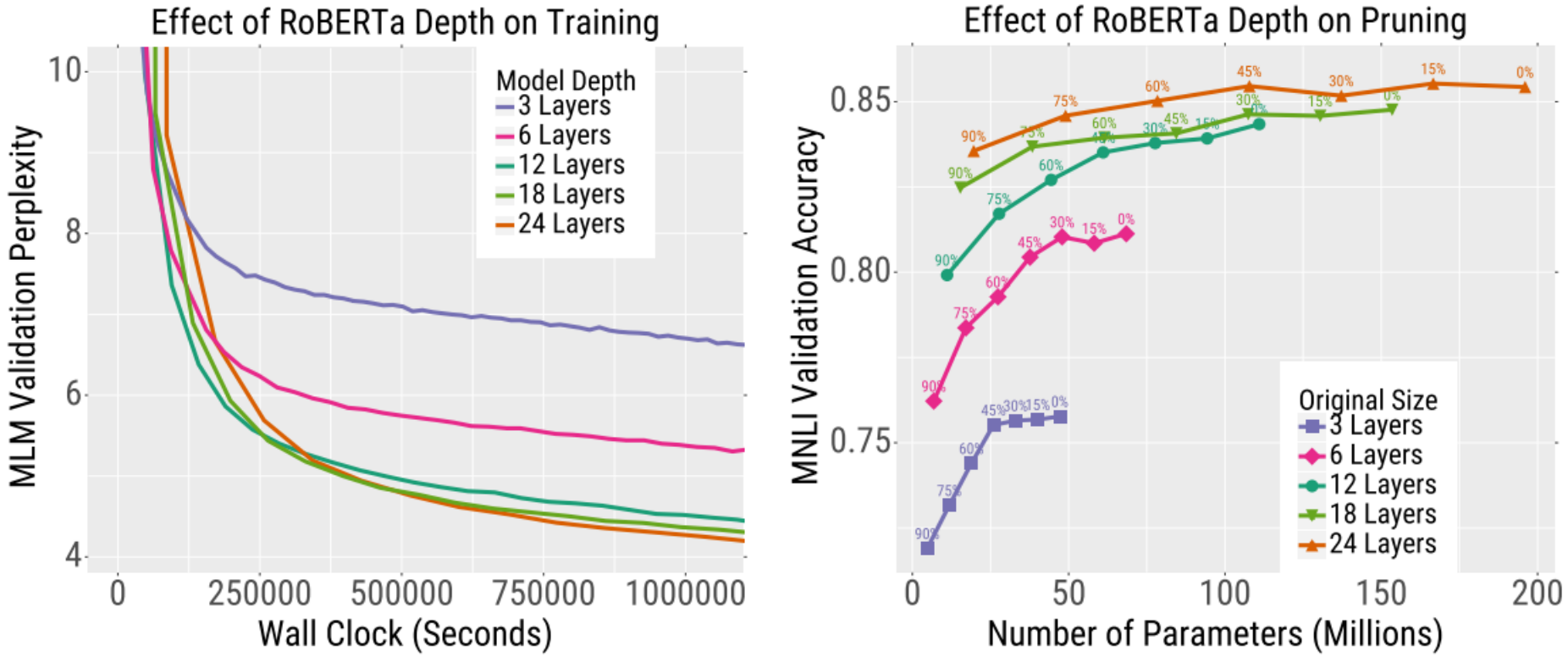


Figure 1. Under the usual presumption that models are trained to convergence, only small models that are fast-to-execute are feasible in resource-constrained settings. Our work shows that the most compute-efficient training scheme is instead to train very large models, stop them well short of convergence, and then heavily compress them to meet test-time constraints.



(a) (b)

Figure 2. Increasing Transformer model size results in lower validation error as a function of *wall-clock time* and better test-time accuracy for a given *inference budget*. **(a)** demonstrates the training speedup for RoBERTa models of different sizes on the masked language modeling pretraining task. In **(b)**, we take RoBERTa checkpoints that have been pretrained for the *same* amount of wall-clock time and finetune them on a downstream dataset (MNLI). We then iteratively prune model weights to zero and find that the best models for a given test-time memory budget are ones which are trained large and then heavily compressed.

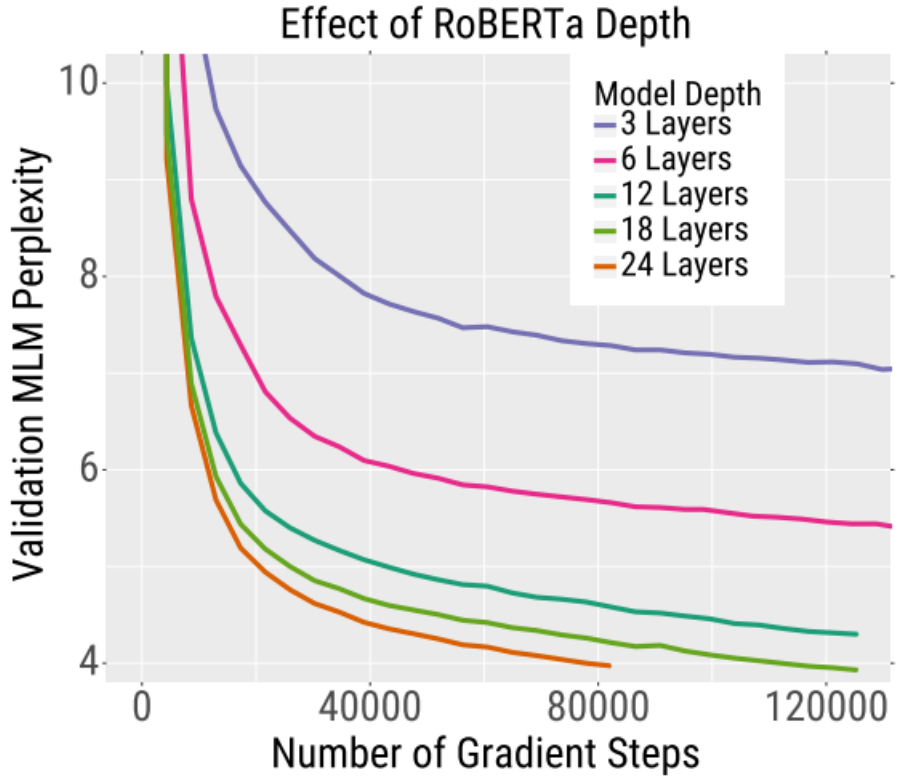


Figure 3. Deeper RoBERTa models converge faster than shallow models with respect to the gradient steps (wall-clock time shown in Figure 2, left).

Larger Models Train Faster

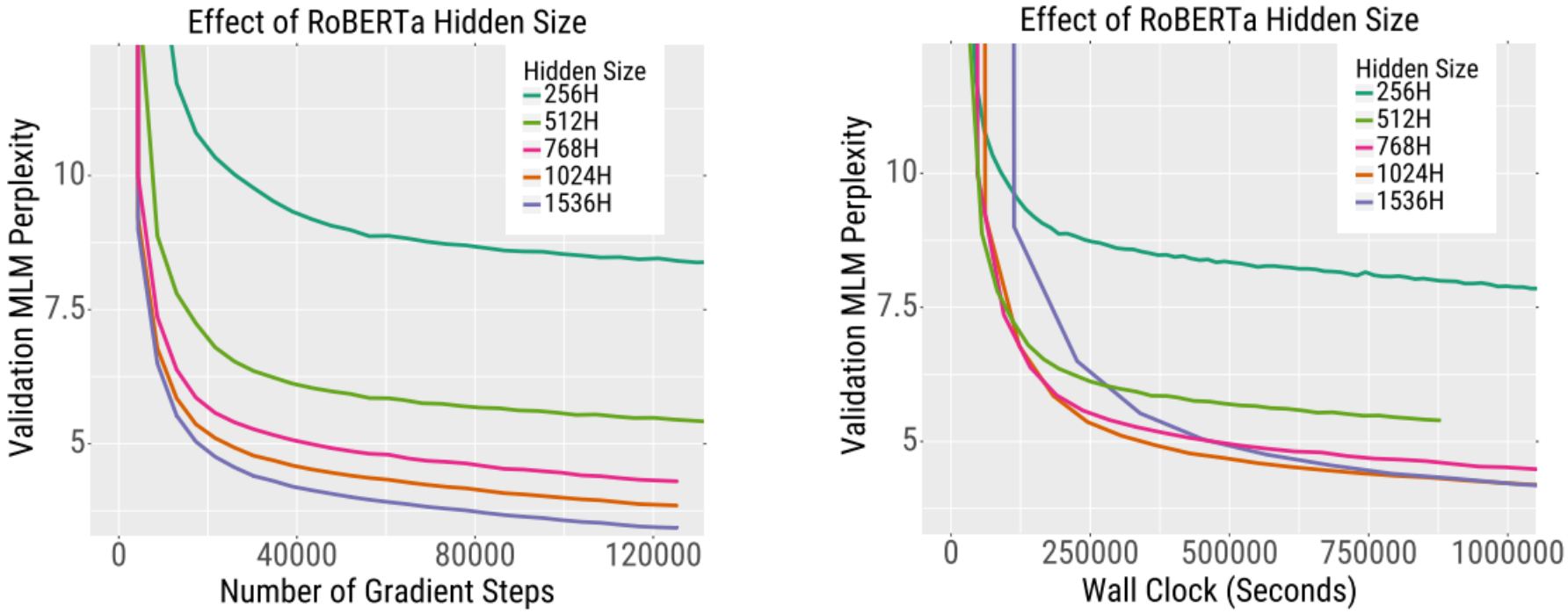


Figure 4. Wider models converge faster than narrower models as function of both gradient steps (left plot) and wall-clock time (right plot).

Increase Model Width and Sometimes Depth

Larger Models Are More Robust to Quantization and Pruning

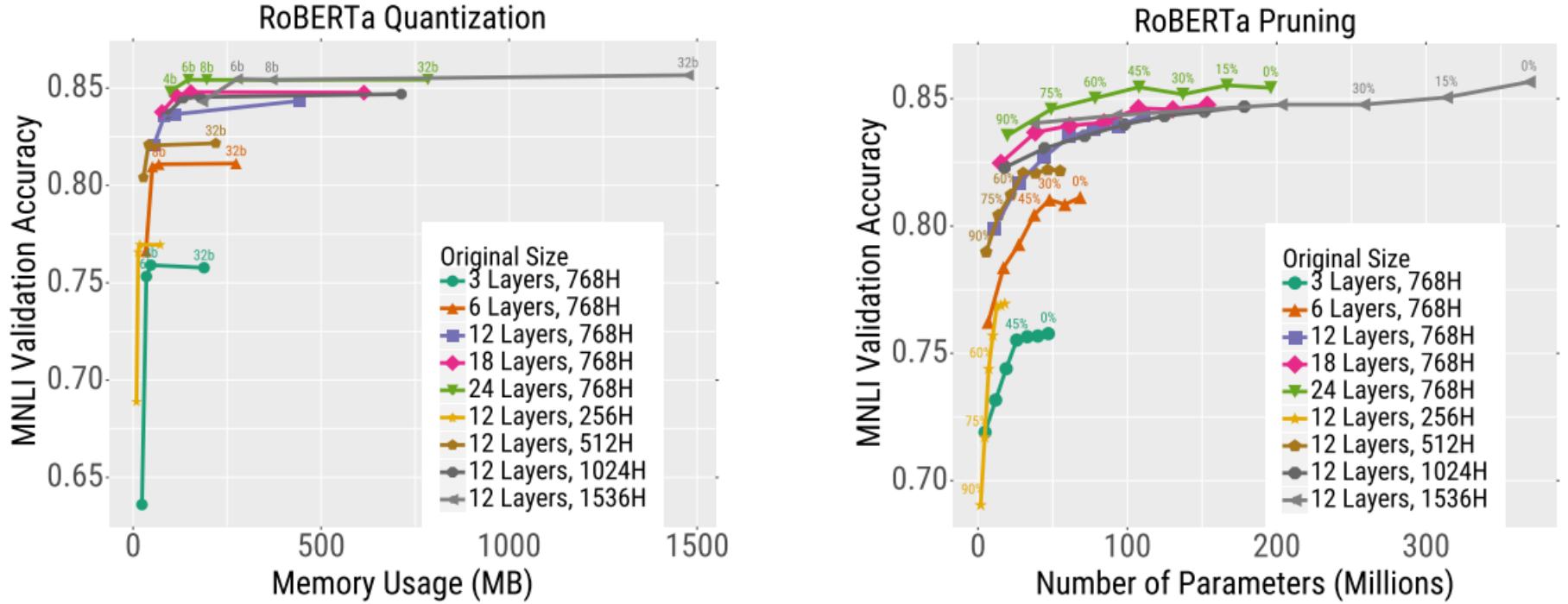


Figure 6. We first pretrain RoBERTa models of different sizes for the *same* total wall-clock time (larger models are trained for fewer steps). We then finetune each model on MNLI and compress them using quantization (left) and pruning (right). For most budgets (x-axis), the highest accuracy models are the ones which are trained large and then heavily compressed. The labels above each point indicate the compression amount (e.g., 4-bit quantization or 45% sparsity); we omit cluttered labels. SST-2 results are shown in Appendix D.

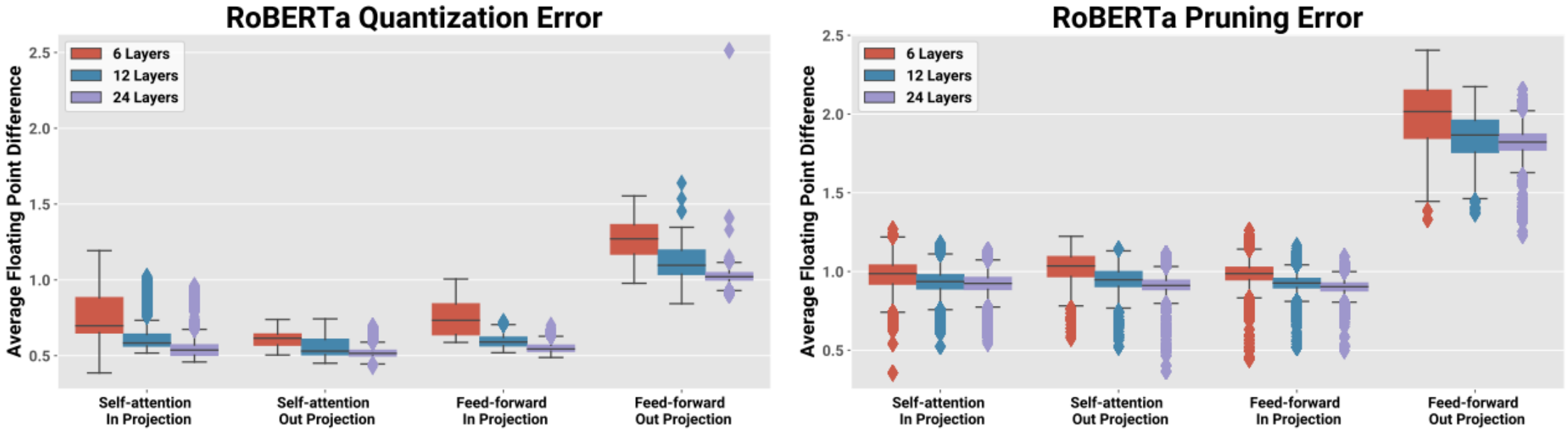


Figure 9. We finetune RoBERTa models of different sizes (6 layers, 12 layers, and 24 layers) on MNLI. We then quantize models to 4-bits or prune models to 60% sparsity. We plot the difference between the weights of the original and the quantized/pruned models averaged across different modules in the Transformer. The mean and variance of the weight difference after quantization (left) is consistently lower for the deeper models compared to the shallower models. The same holds for the difference after pruning (right). This shows that the larger model's weights are naturally easier to approximate with low-precision / sparse matrices than smaller models.

train large models and then heavily compress them

Is Complexity Required for Neural Network Pruning?

- Global Magnitude Pruning
  - ranks weights in order of their magnitudes and prunes the smallest one
  - Gradual is better than one-shot
    - selecting a pre-trained model in one-shot pruning, or an untrained model in gradual pruning
    - Minimum threshold: fixed number ofweights that are preserved in every layer of the neural network post pruning

https://arxiv.org/pdf/2209.14624

Method	Sparsity	WRN-28-8 Acc.	ResNet-32 Acc.
Baseline	0.0%	96.06%	93.83 ± 0.12 %
SNIP	90%	95.49 ± 0.21%	90.40 ± 0.26%
SM	90%	95.67 ± 0.14%	91.54 ± 0.18%
DSR	90%	95.81 ± 0.10%	91.41 ± 0.23%
DPF	90%	96.08 ± 0.15%	92.42 ± 0.18%
Global MP	90%	**96.30 ± 0.03%**	**92.67 ± 0.03%**
SNIP	95%	94.93 ± 0.13%	87.23 ± 0.29%
SM	95%	95.64 ± 0.07%	88.68 ± 0.22%
DSR	95%	95.55 ± 0.12%	84.12 ± 0.32%
DPF	95%	95.98 ± 0.10%	**90.94 ± 0.35%**
Global MP	95%	**96.16 ± 0.02%**	90.65 ± 0.13%

TABLE I: Results of SOTA pruning algorithms on WideResNet-28-8 and ResNet-32 on CIFAR-10. The bold font denotes algorithm with the best performance. Global MP outperforms or yields comparable performance to other algorithms.

Method	Top-1 Acc	Params.	Sparsity	FLOPs pruned
MobileNet-V1	71.95%	4.21M	0.00%	0.0%
GMP	67.70%	1.09M	74.11%	71.4%
STR	68.35%	1.04M	75.28%	82.2%
Global MP	**70.74%**	1.04M	**75.28%**	68.9%
GMP	61.80%	0.46M	89.03%	85.6%
STR	61.51%	0.44M	89.62%	93.0%
Global MP	59.49%	0.42M	90.00%	83.7%
Global MP with MT	**63.94%**	0.42M	**90.00%**	72.9%

TABLE III: Results of pruning algorithms on MobileNet-V1

GMP	70.59%	1.28M	95.00%	95.0%
DNW	68.30%	1.28M	95.00%	95.0%
RigL*	67.50%	1.28M	95.00%	92.2%
RigL + ERK	70.00%	1.28M	95.00%	85.3%
WoodFisher	72.12%	1.28M	95.00%	-
MFAC	**72.32%**	1.28M	**95.00%**	-
STR	70.40%	1.27M	95.03%	96.1%
Global MP (One-shot)	71.56%	1.20M	95.30%	89.3%
Global MP (Gradual)	**72.14%**	1.20M	**95.30%**	93.1%
GMP	57.90%	0.51M	98.00%	98.0%
DNW	58.20%	0.51M	98.00%	98.0%
STR	61.46%	0.50M	98.05%	98.2%
Global MP (One-shot)	61.80%	0.50M	98.05%	93.7%
Global MP (Gradual)	**66.57%**	0.50M	**98.05%**	96.2%

TABLE II: Results on ResNet-50 on ImageNet. Global MP outperforms SOTA pruning algorithms at all sparsity levels for the sparsity-accuracy trade-off and at high sparsity levels for the FLOPs-accuracy trade-off. Bold font denotes best performance for the sparsity-accuracy trade-off while underlined font denotes best performance for FLOPs-accuracy trade-off. * and # imply the first and the last layer are dense, respectively.