DATA SCIENCE (資料科學)
SPRING, 2023

LECTURE 2: MODEL COMPRESSION

PC: Midjourney

Shuai, Hong-Han (帥宏翰)

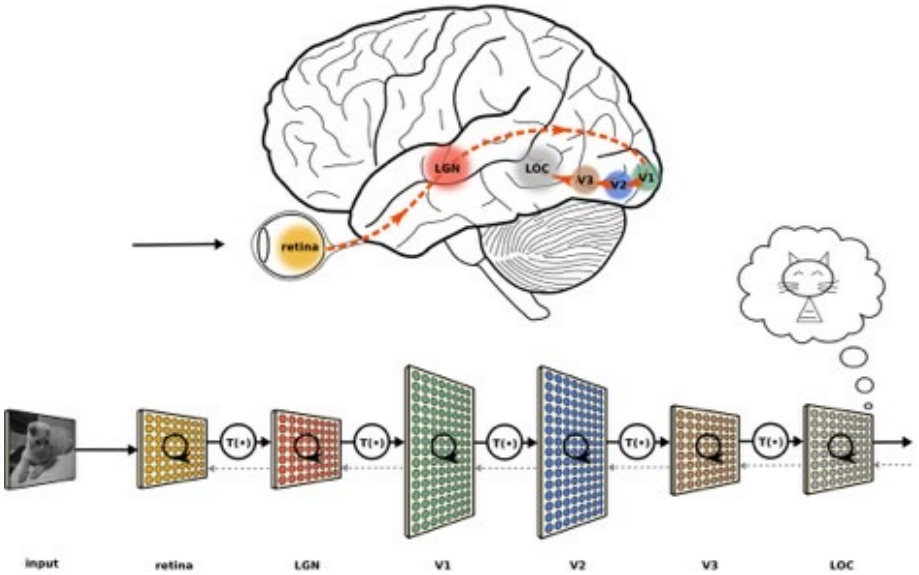
Associate Professor

Department of Electronics and Electrical Engineering

National Yang Ming Chiao Tung University

Thanks for the slides from Prof. Han Song in MIT and Prof. Hung-Yi Lee in NTU.

DEEP LEARNING:NEXT WAVE OF AI



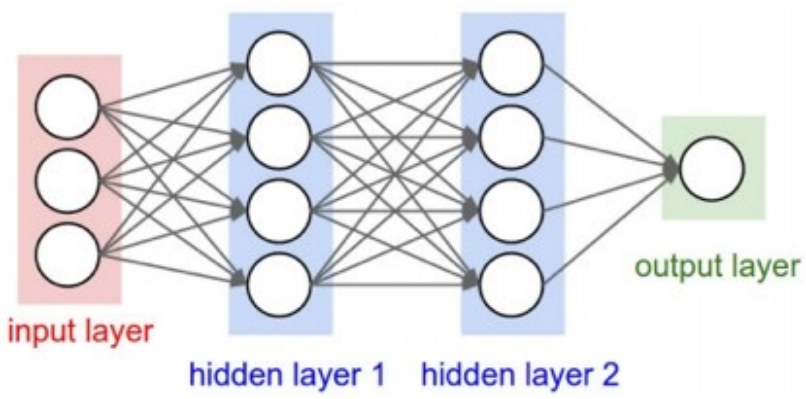
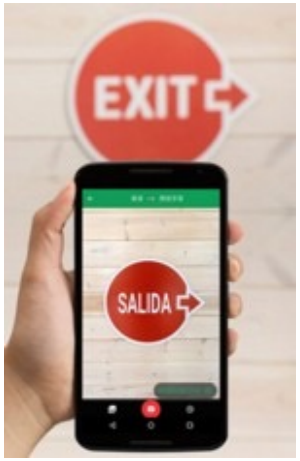


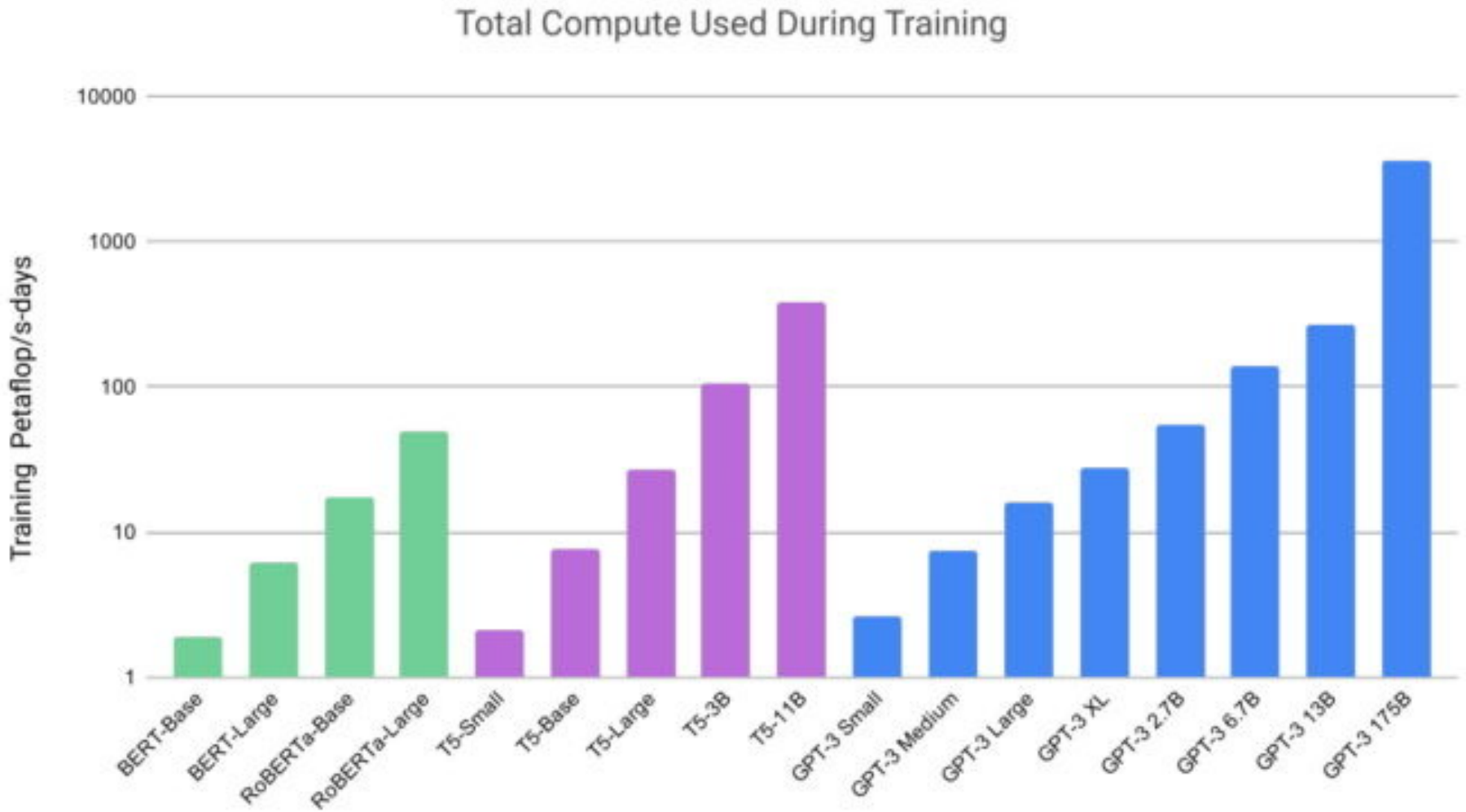
Image Recognition

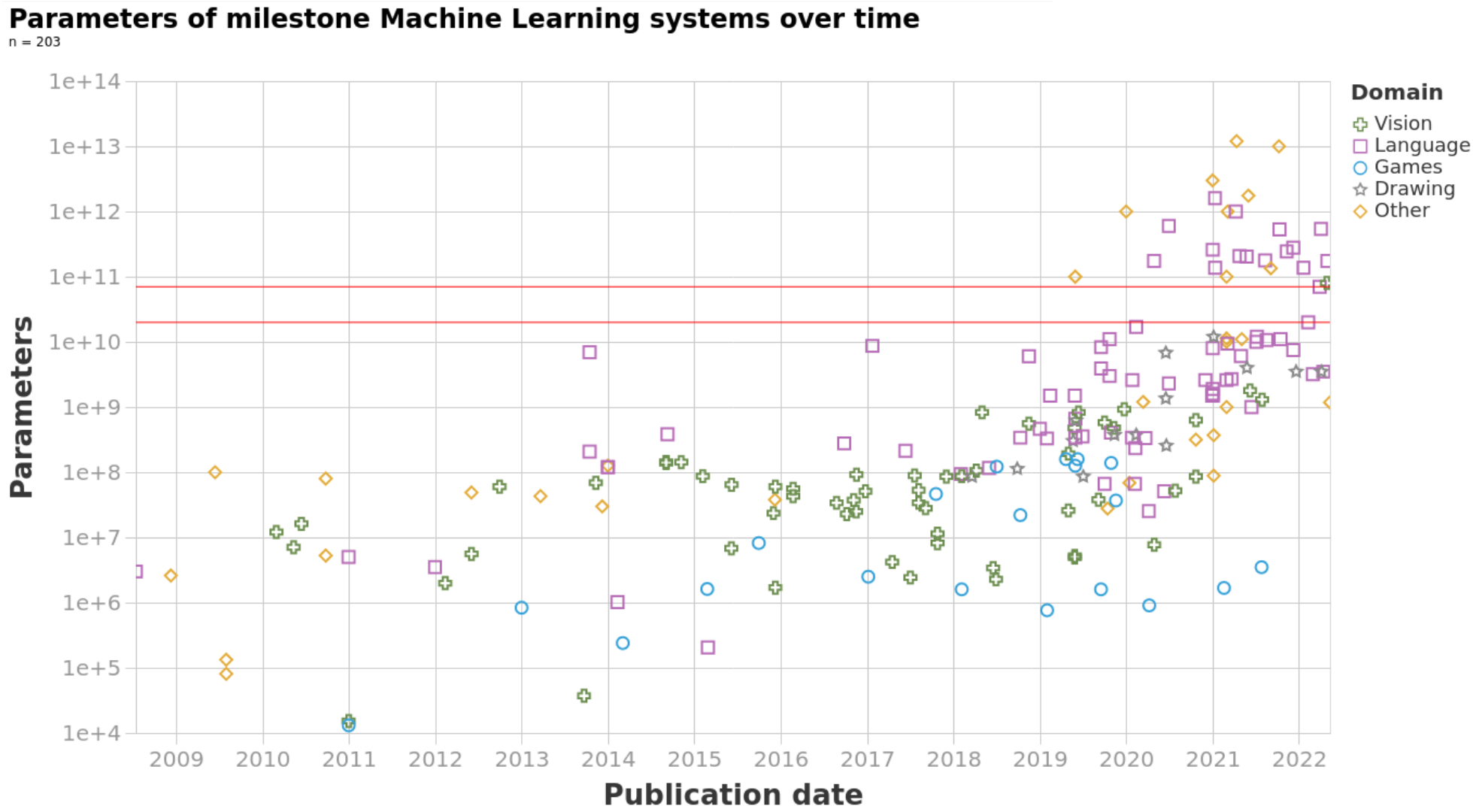
Speech Recognition

Natural Language Processing

APPLICATIONS



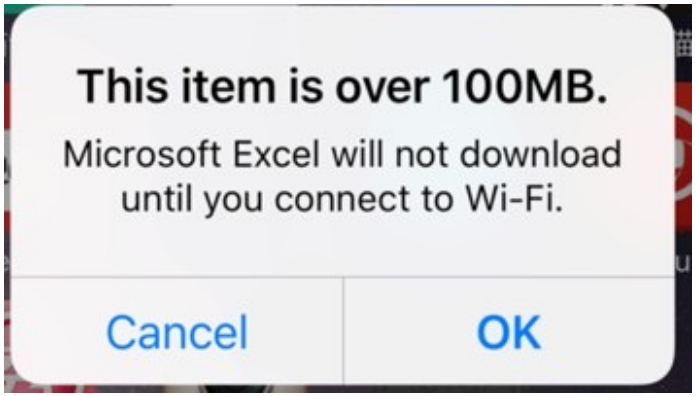

Total Compute Used During Training
Training Petaflop/s-days
10000
1000
100
10
1
BERT-Base
BERT-Large
RoBERTa-Base
RoBERTa-Large
T5-Small
T5-Base
T5-Large
T5-3B
T5-11B
GPT-3 Small
GPT-3 Medium
GPT-3 Large
GPT-3 XL
GPT-3 2.7B
GPT-3 6.7B
GPT-3 13B
GPT-3 175B


Parameters of milestone Machine Learning systems over time
n = 203
Parameters
1e+14
1e+13
1e+12
1e+11
1e+10
1e+9
1e+8
1e+7
1e+6
1e+5
1e+4
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
Publication date
Domain
Vision
Language
Games
Drawing
Other

THE PROBLEM:

IF RUNNING DNN ON MOBILE...

App developers suffers from the model size



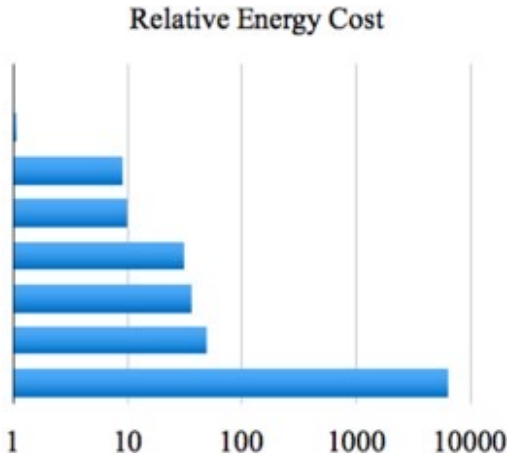
“At Baidu, our #1 motivation for compressing networks is to **bring down the size of the binary file**. As a mobile-first company, we frequently update various apps via different app stores. We've **very sensitive to the size of our binary files**, and a feature that increases the binary size by 100MB will receive much more scrutiny than one that increases it by 10MB.” —Andrew Ng

The Problem:

If Running DNN on Mobile…

Hardware engineer suffers from the model size (embedded system, limited resource)

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**



THE PROBLEM:

IF RUNNING DNN ON THE CLOUD...

Network Delay

Power Budget

User Privacy

Intelligent but Inefficient

AGENDA

Software

- Deep Compression (Pruning, Quantization, Huffman Coding)
- Architecture Design
- Knowledge Distillation
- Dynamic Computation
- New works

- EIE Accelerator

Hardware

DEEP COMPRESSION

Pruning, Quantization, Huffman Coding

GOAL

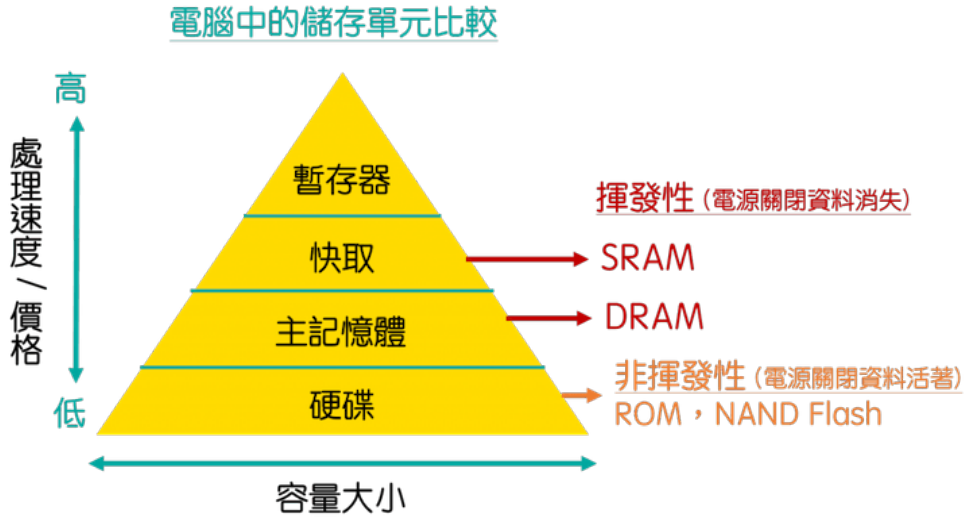
Deep Neural Network Model Compression

Smaller Size
Compress Mobile App Size by 35x-50x

Accuracy
no loss of accuracy
improved accuracy

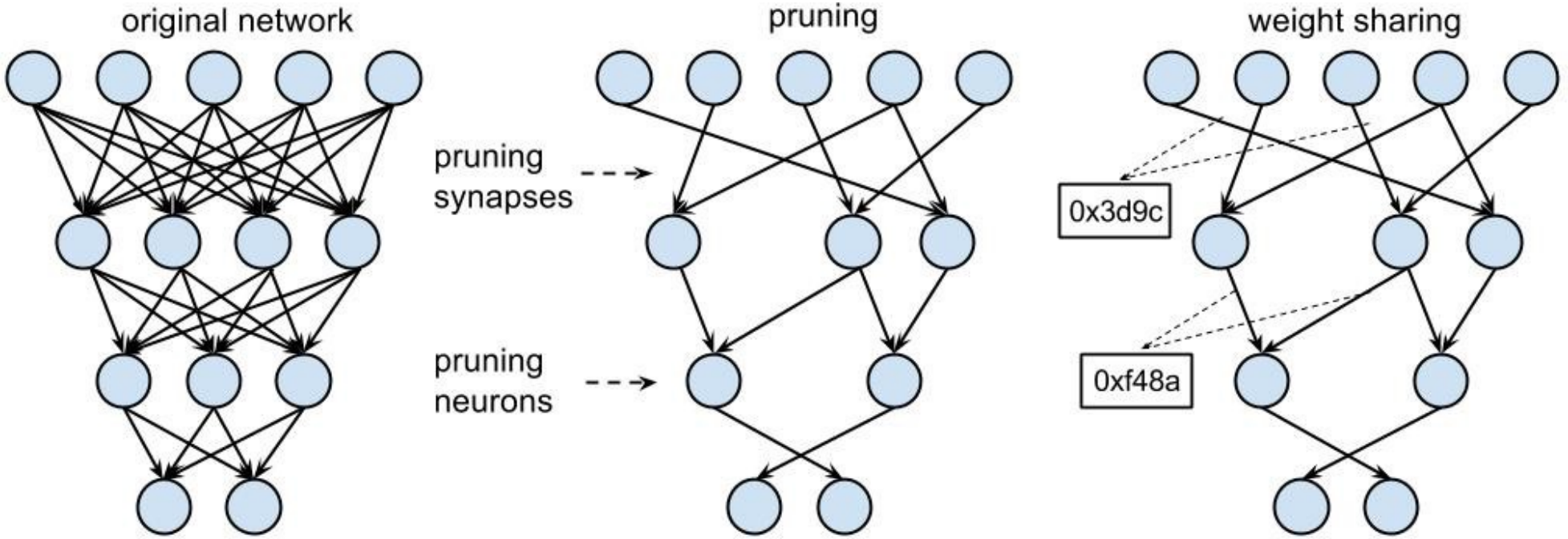
Speedup
make inference faster

DEEP COMPRESSION

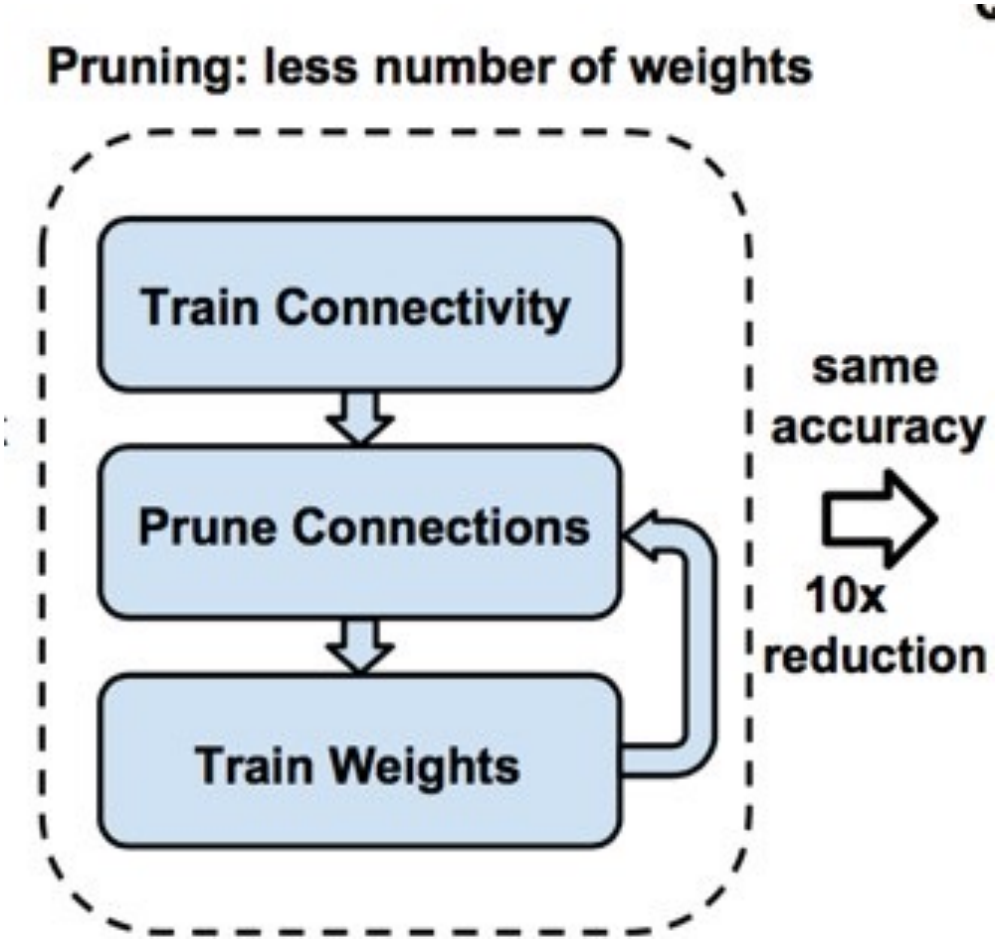


- AlexNet: 35×, 240MB => 6.9MB
- VGG16: 49×, 552MB => 11.3MB
- Both with no loss of accuracy on ImageNet12
- Weights fits on-chip SRAM, taking 120x less energy than DRAM

COMPRESSION PIPELINE: OVERVIEW



1. PRUNING


Pruning: less number of weights
Train Connectivity
Prune Connections
Train Weights
same accuracy
10x reduction

PRUNING: MOTIVATION

Age	Number of Connections	Stage
at birth	50 Trillion	newly formed
1 year old	1000 Trillion	peak
10 year old	500 Trillion	pruned and stabilized

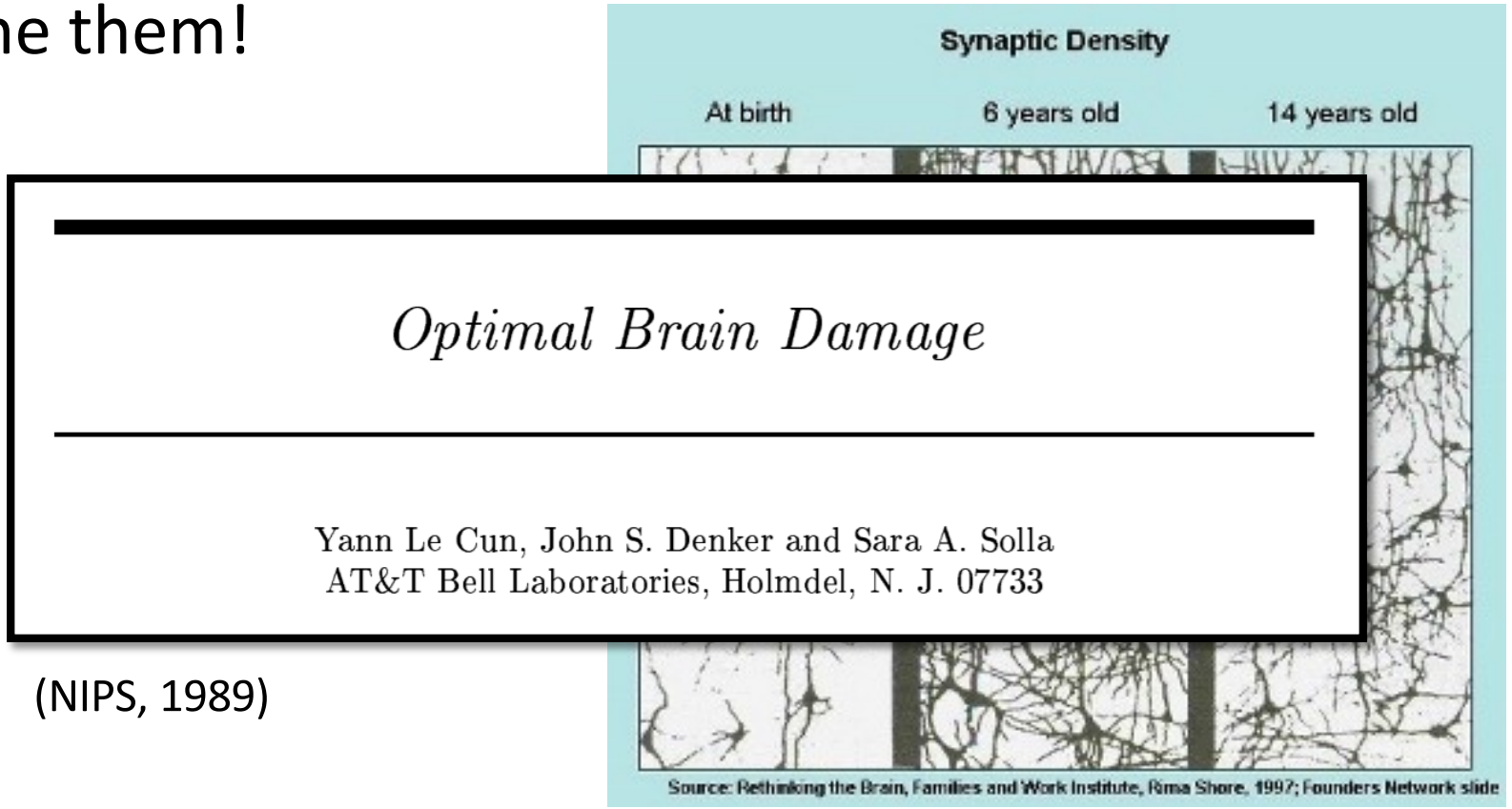
Table 1: The synapses pruning mechanism in human brain development

- Trillion of synapses are generated in the human brain during the first few months of birth.
- **1 year old**, peaked at **1000 trillion**
- Pruning begins to occur.
- **10 years old**, a child has nearly **500 trillion** synapses
- This ’pruning’ mechanism removes redundant connections in the brain.

[1] Christopher A Walsh. Peter huttenlocher (1931-2013). *Nature*, 502(7470):172–172, 2013.

Network can be pruned

- Networks are typically over-parameterized (there is significant redundant weights or neurons)
- Prune them!



(NIPS, 1989)

Network Pruning

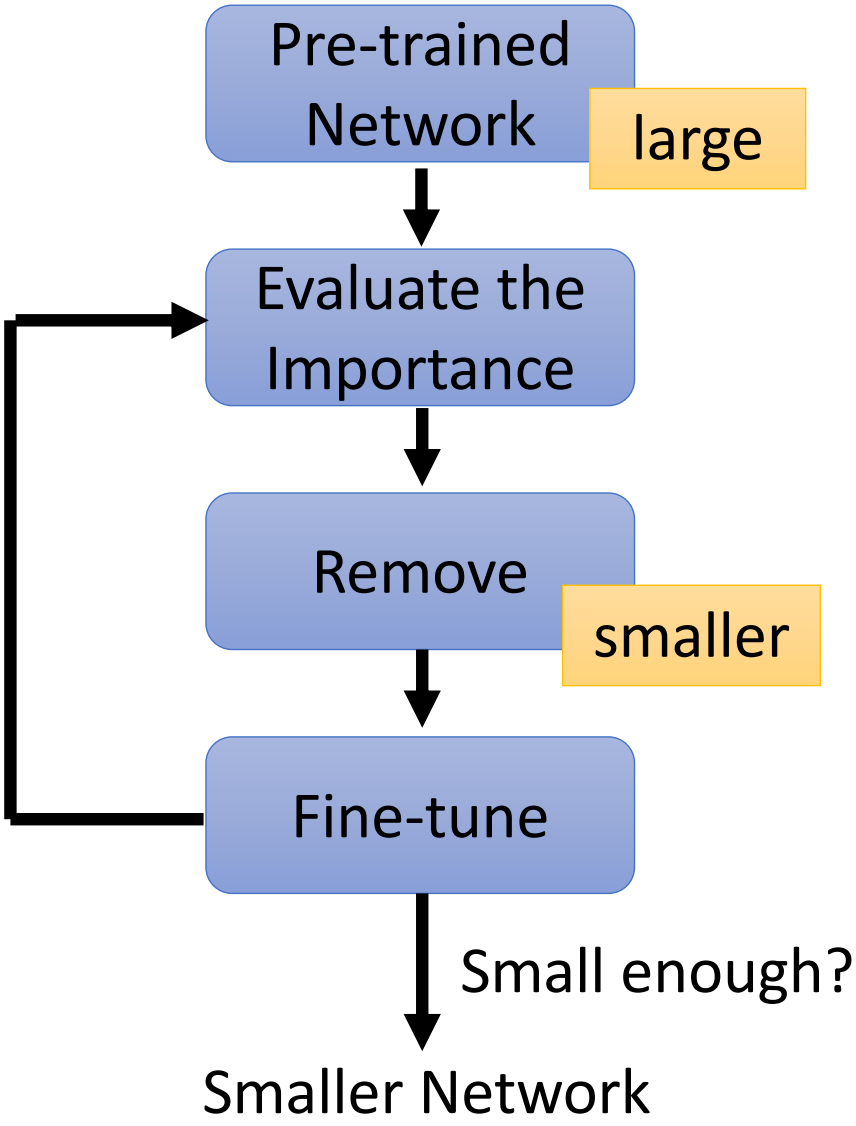
- Importance of a weight:

  absolute values, life long …

- Importance of a neuron:

  the number of times it wasn't zero on a given data set ……

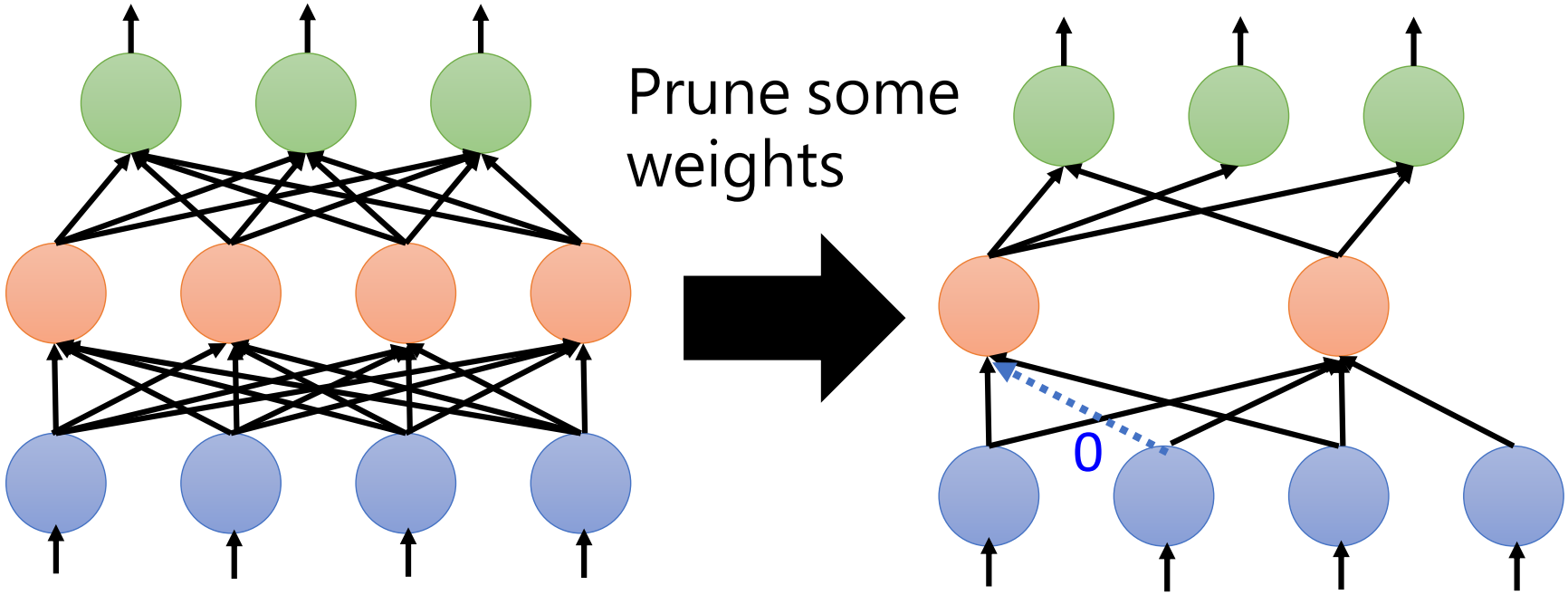
- After pruning, the accuracy will drop (hopefully not too much)
- Fine-tuning on training data for recover
- Don't prune too much at once, or the network won't recover.



Network Pruning - Practical Issue

- Weight pruning

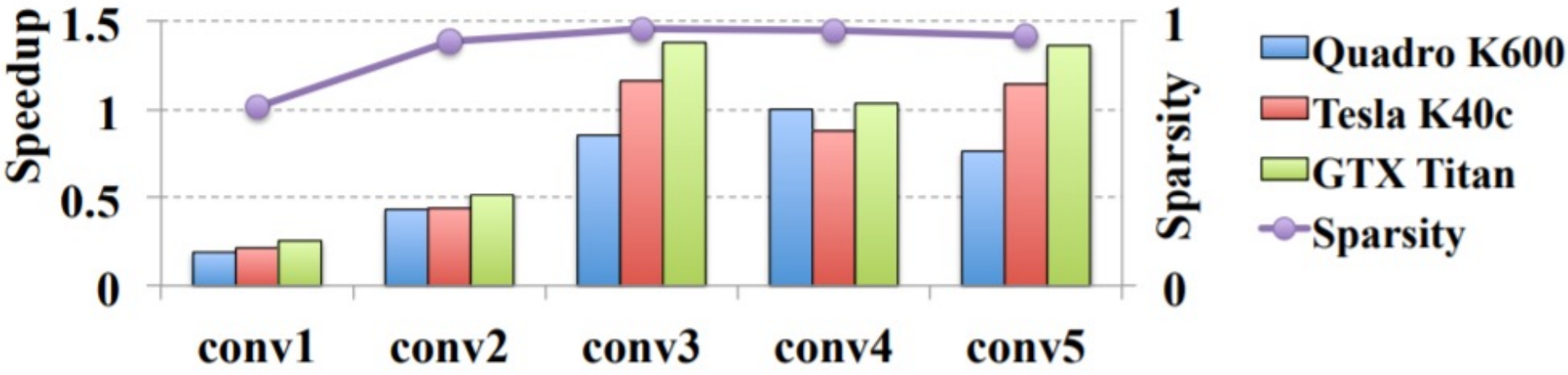
The network architecture becomes irregular.



Hard to implement, hard to speedup ……

Network Pruning - Practical Issue

- Weight pruning

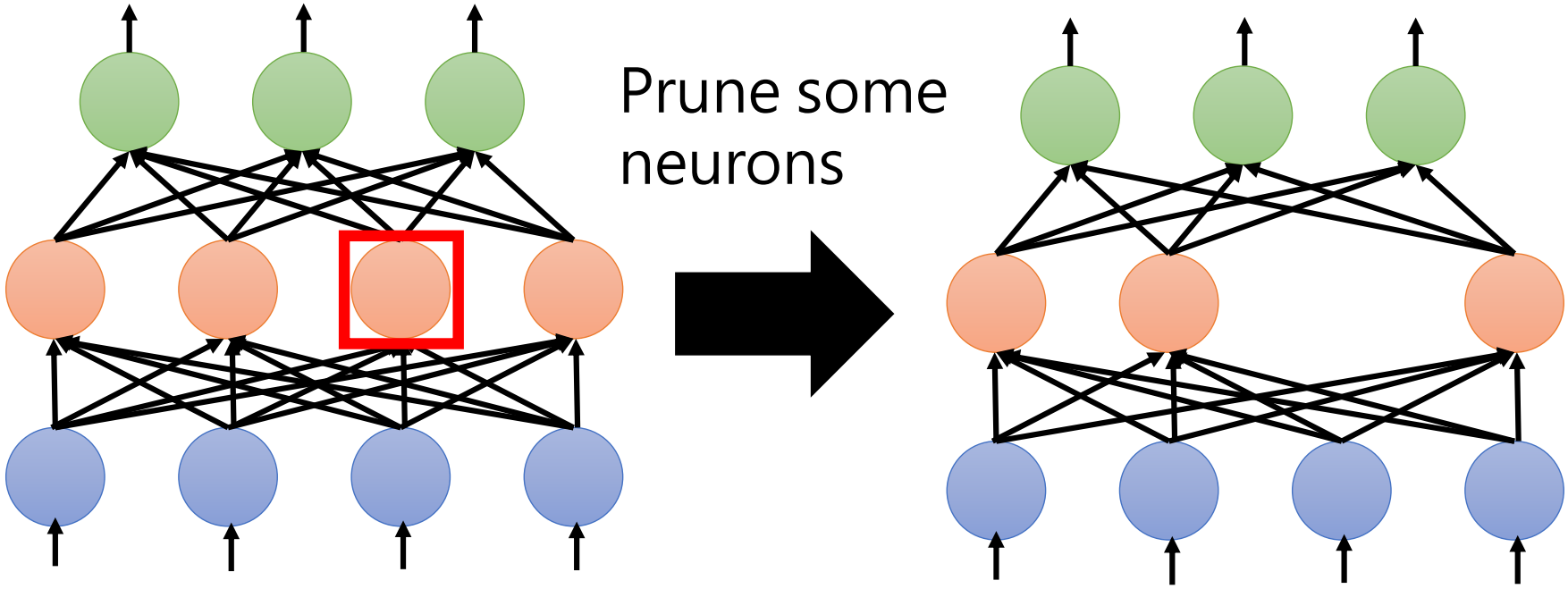


https://arxiv.org/pdf/1608.03665.pdf

Network Pruning - Practical Issue

- Neuron pruning

The network architecture is regular.



Easy to implement, easy to speedup ……

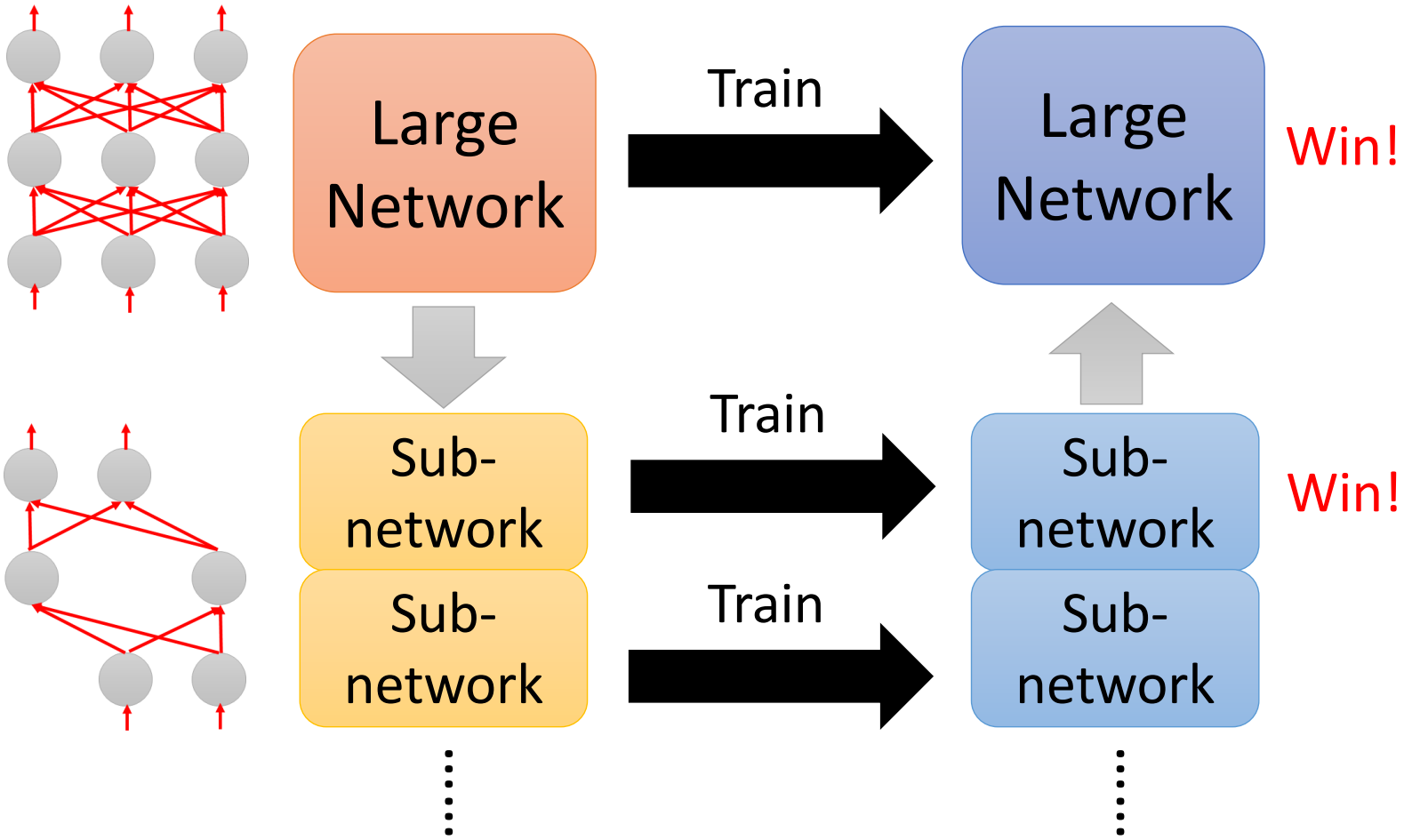
Why Pruning? (overparameterized)

- How about simply train a smaller network?
- It is widely known that smaller network is more difficult to learn successfully.
  - Larger network is easier to optimize?
  - Lottery Ticket Hypothesis

https://arxiv.org/abs/1803.03635

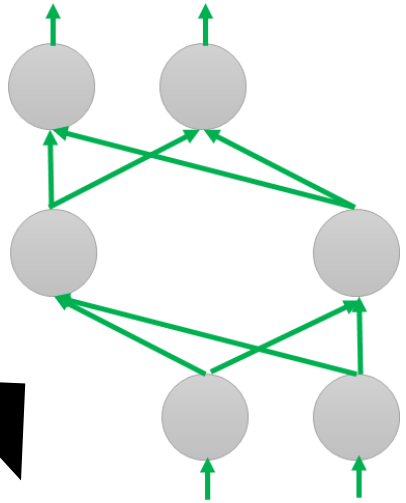
Why Pruning?

Lottery Ticket Hypothesis



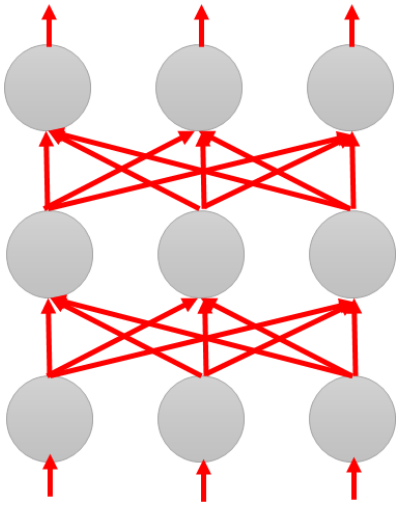
Why Pruning?

Lottery Ticket Hypothesis

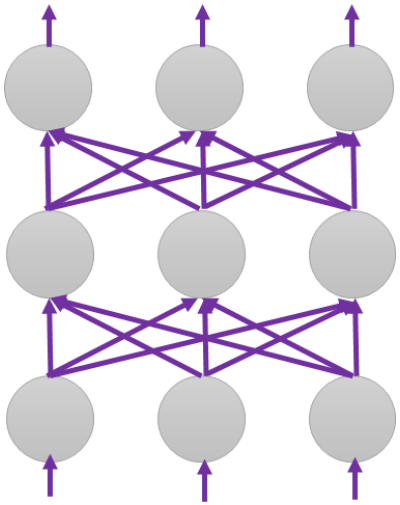


Random init

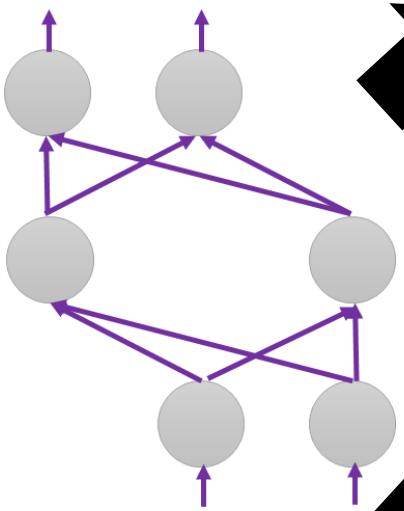
Again



Random init



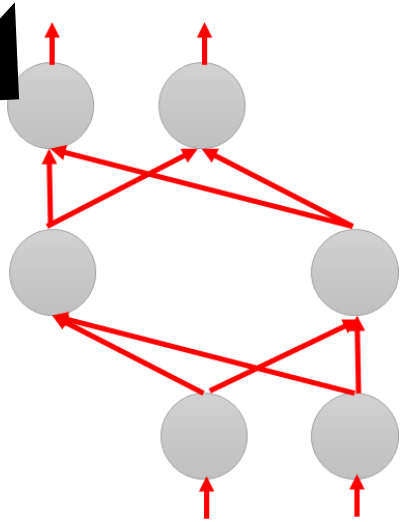
Trained



Pruned

Original

Random init



Random Init weights

Trained weight

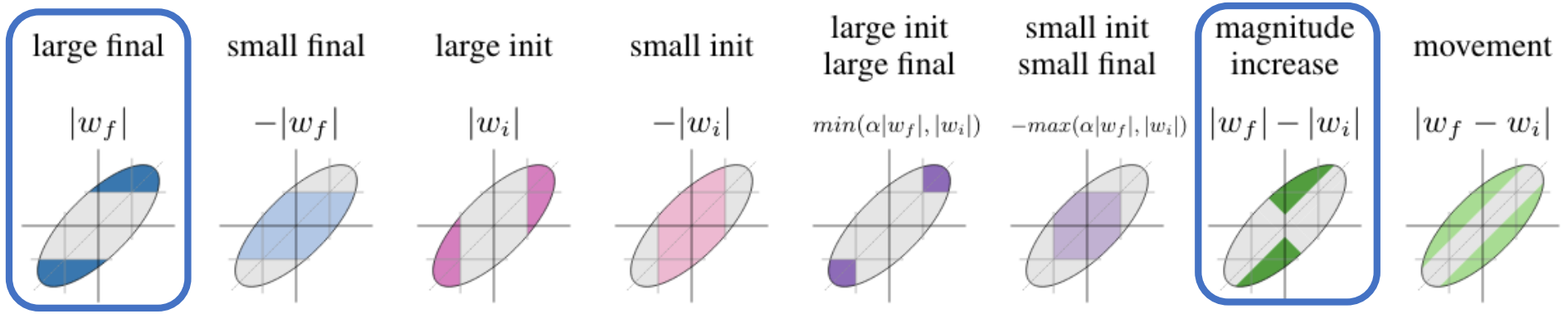
Another random Init weights

Deconstructing Lottery Tickets: Zeros, Signs, and the Supermask
https://arxiv.org/abs/1905.01067

Why Pruning?

Lottery Ticket Hypothesis

- Different pruning strategy



- "sign-ificance" of initial weights: Keeping the sign is critical

0.9, 3.1, -9.1, 8.5 ..... ➡ $+\alpha, +\alpha, -\alpha, +\alpha$ .....

- Pruning weights from a network with random weights

Weight Agnostic Neural Networks https://arxiv.org/abs/1906.04358

Why Pruning?

https://arxiv.org/abs/1810.05270

- Rethinking the Value of Network Pruning

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**

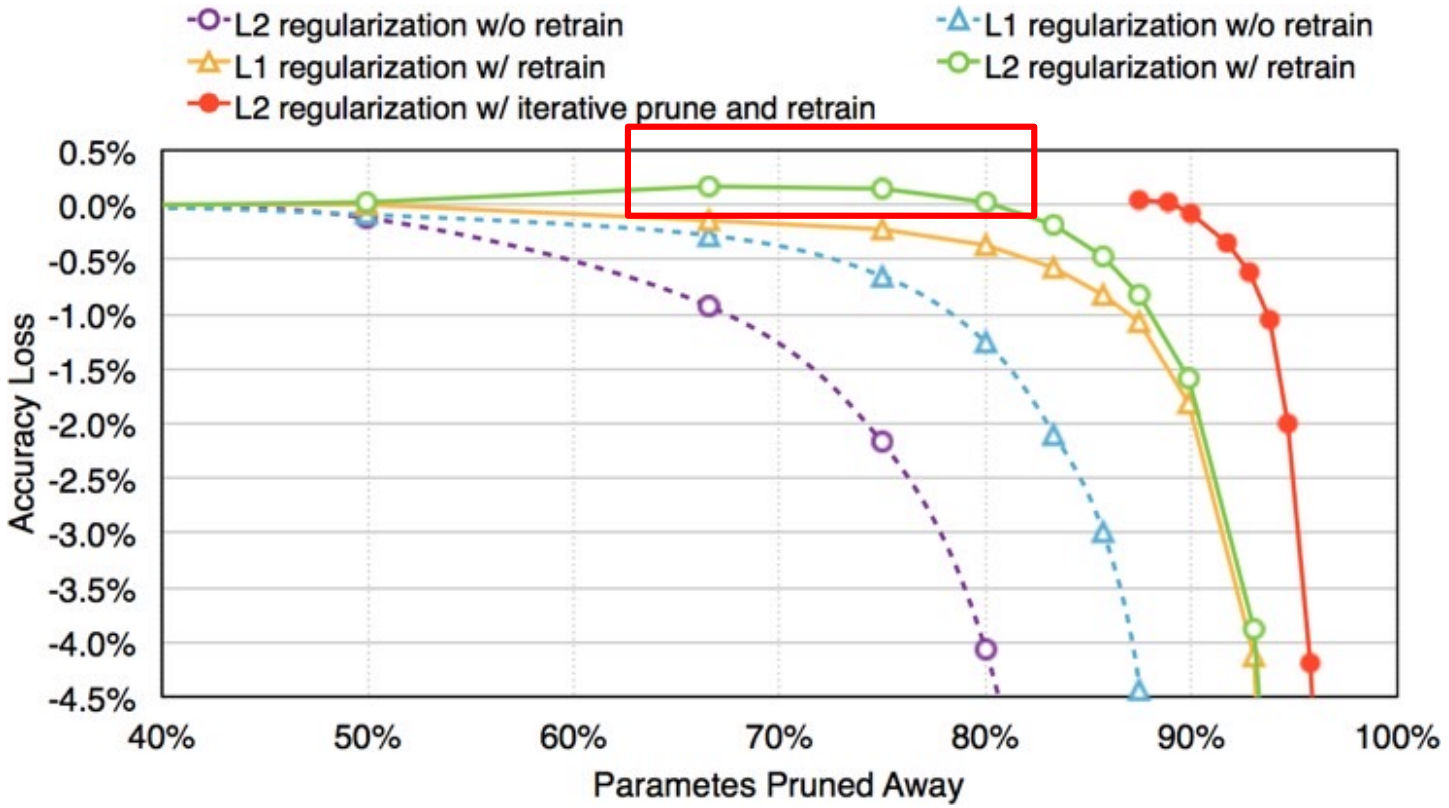
- **New** random initialization, not **original** random initialization in “Lottery Ticket Hypothesis”
- Limitation of “Lottery Ticket Hypothesis” (small lr, unstructured)

PRUNING: RESULT ON 4 COVNETS

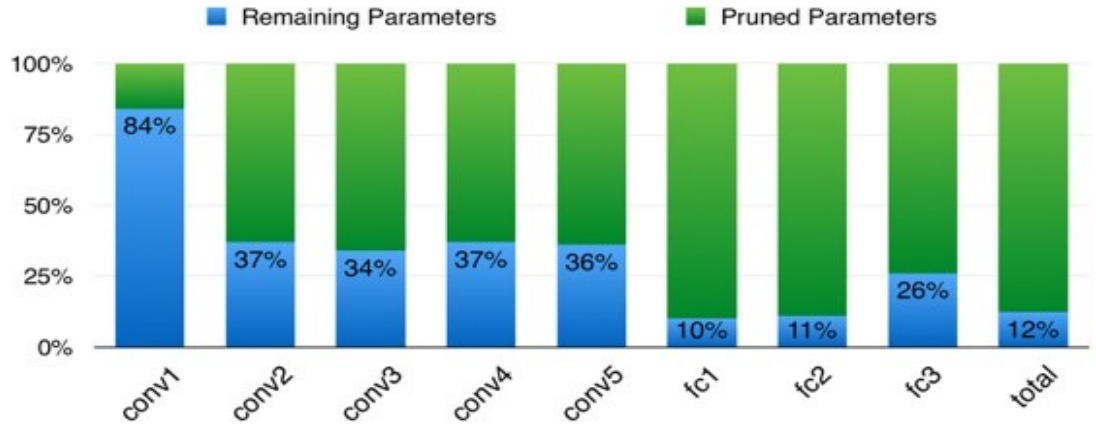
Network	Top-1 Error	Top-5 Error	Parameters	Compression Rate
LeNet-300-100 Ref	1.64%	-	267K	
LeNet-300-100 Pruned	1.59%	-	**22K**	**12**×
LeNet-5 Ref	0.80%	-	431K	
LeNet-5 Pruned	0.77%	-	**36K**	**12**×
AlexNet Ref	42.78%	19.73%	61M	
AlexNet Pruned	42.77%	19.67%	**6.7M**	**9**×
VGG16 Ref	31.50%	11.32%	138M	
VGG16 Pruned	31.34%	10.88%	**10.3M**	**13**×

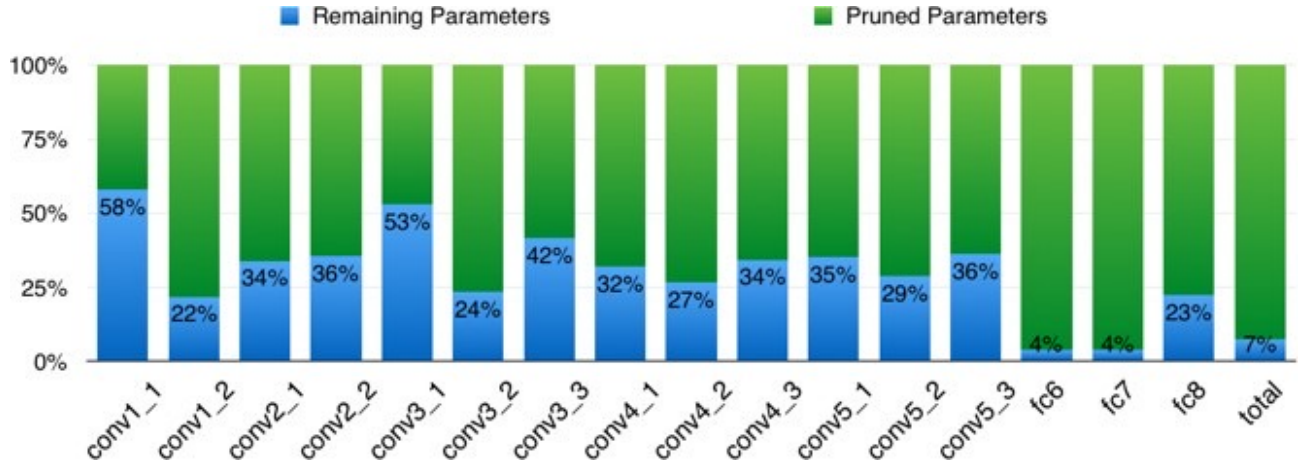
Table 1: Network pruning can save 9× to 13× parameters with no drop in predictive performance

PRUNING: ALEXNET

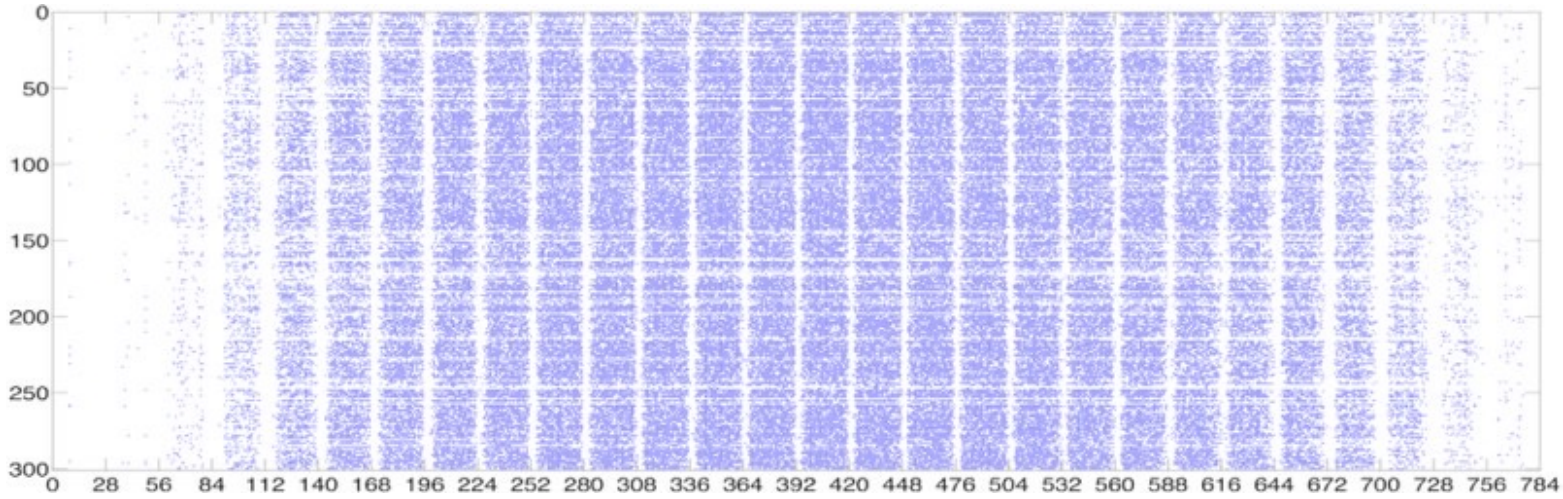


ALEXNET & VGGNET



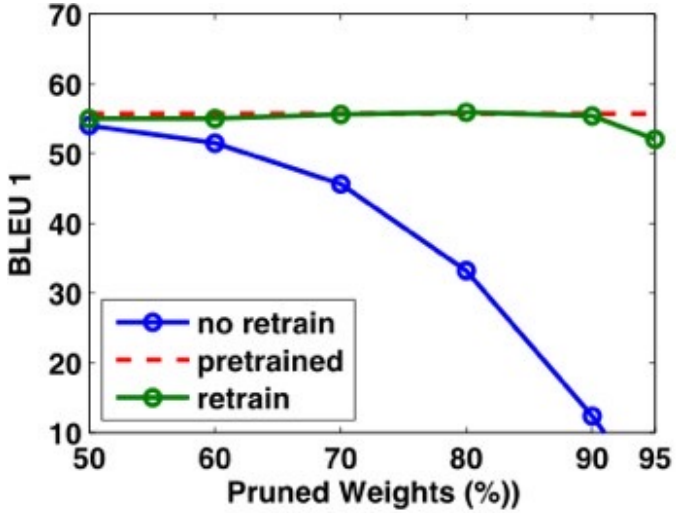


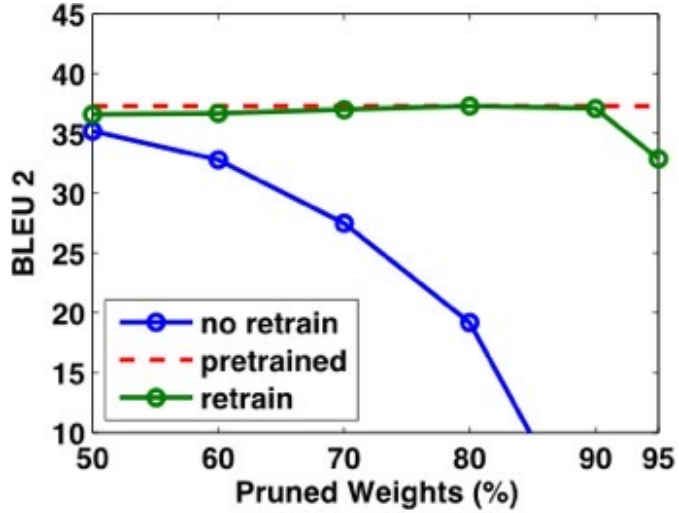
MASK VISUALIZATION

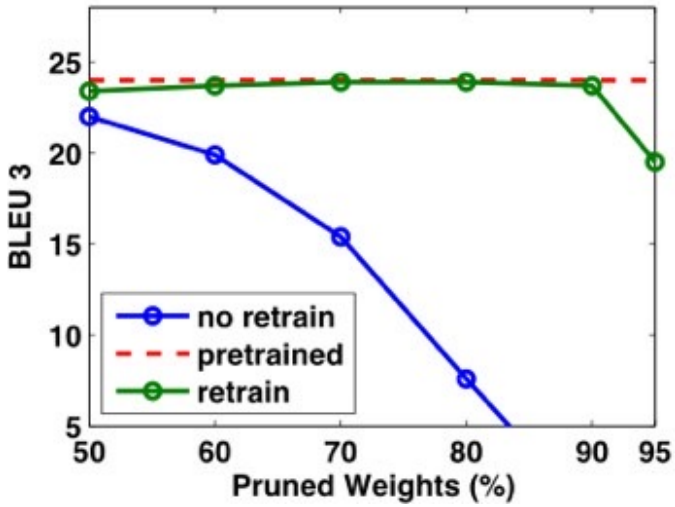


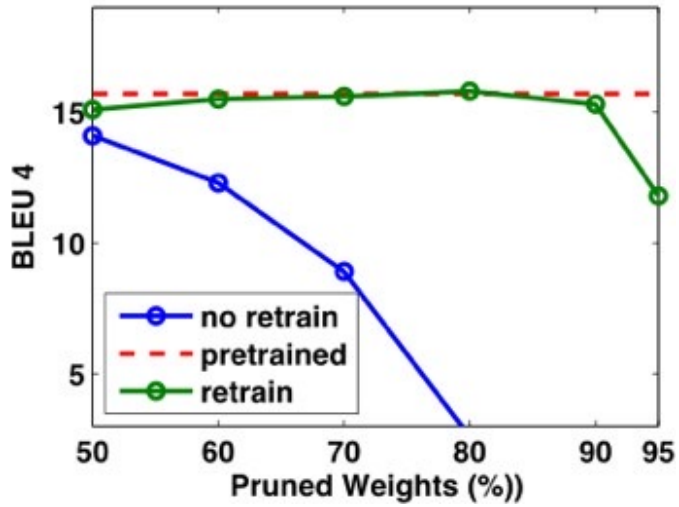
Visualization of the first FC layer's sparsity pattern of Lenet-300-100. It has a banded structure repeated 28 times, which correspond to the un-pruned parameters in the center of the images, since the digits are written in the center.

PRUNING ALSO WORKS WELL ON RNN+LSTM



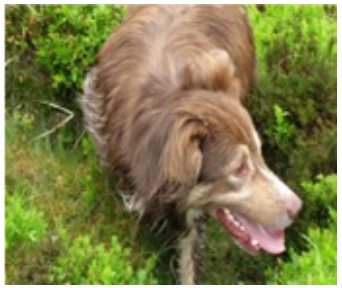






[1] Thanks Shijian Tang pruning Neural Talk

- **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

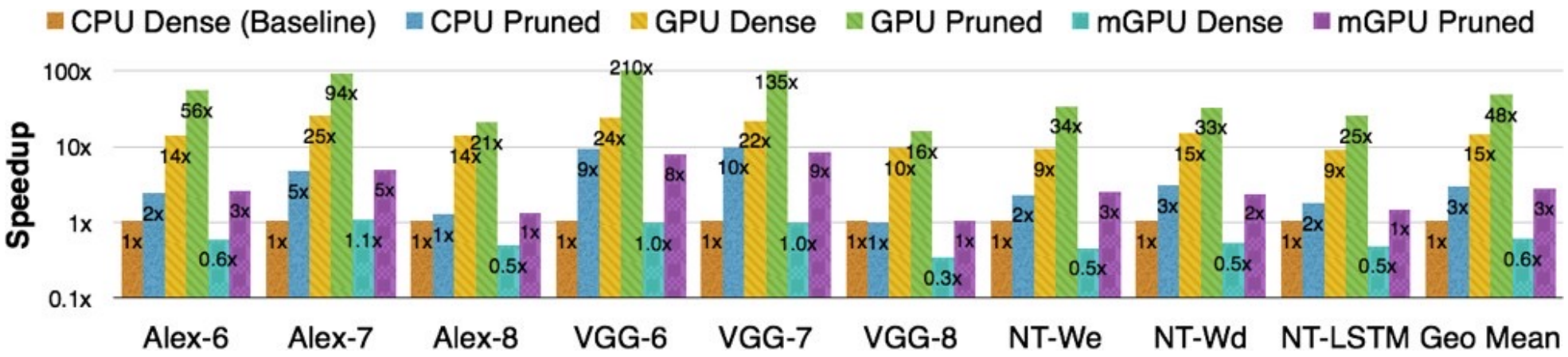


- **Original** : a brown dog is running through a grassy field
- **Pruned 90%**: a brown dog is running through a grassy area

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

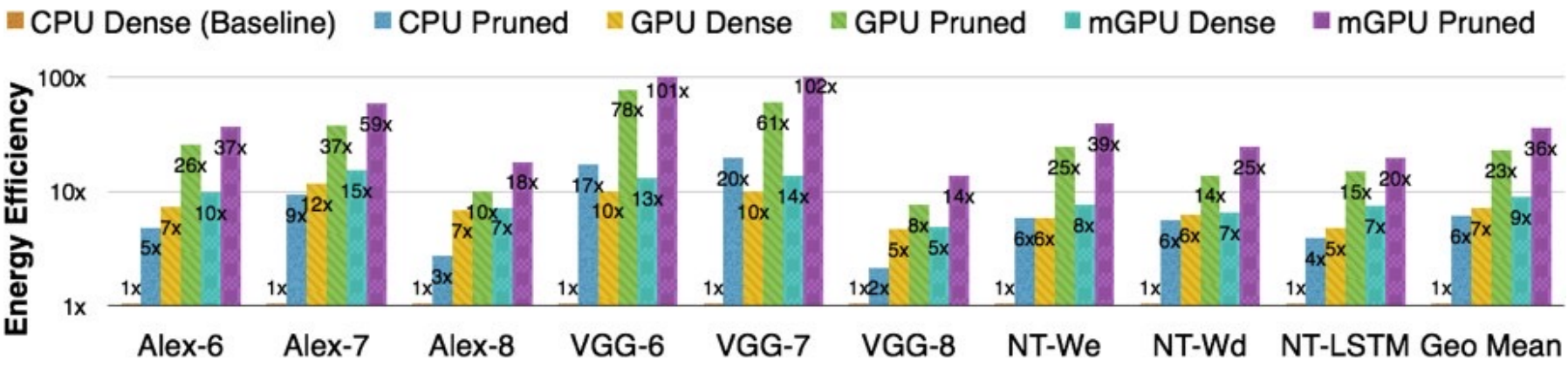
- **Original** : a soccer player in red is running in the field
- **Pruned <u>95%</u>**: a man in a red shirt and black and white black shirt is running through a field

SPEEDUP (FC LAYER)



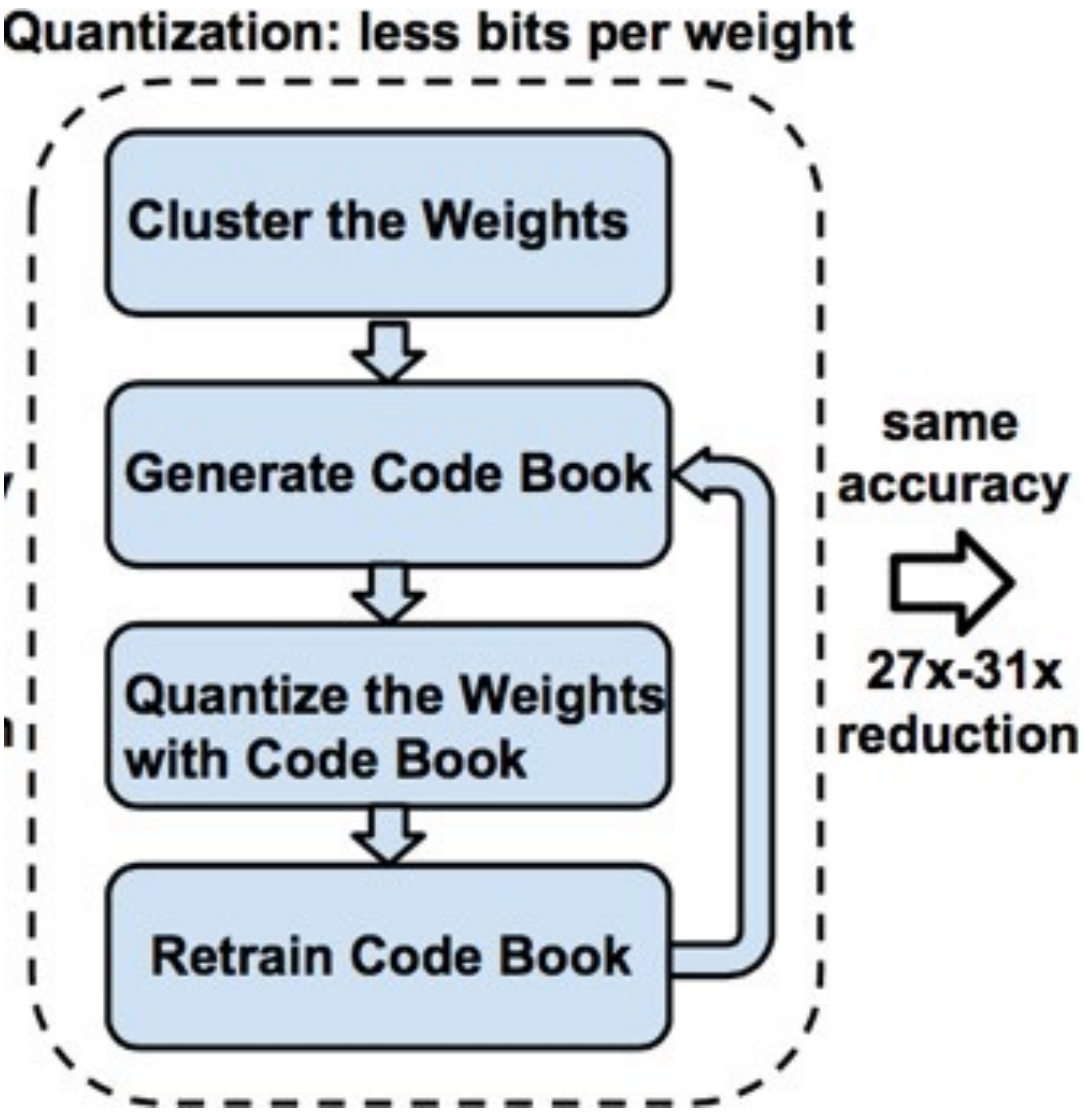
- Intel Core i7 5930K: MKL CBLAS GEMV, MKL SPBLAS CSRMV
- NVIDIA GeForce GTX Titan X: cuBLAS GEMV, cuSPARSE CSRMV
- NVIDIA Tegra K1: cuBLAS GEMV, cuSPARSE CSRMV

ENERGY EFFICIENCY (FC LAYER)



- Intel Core i7 5930K: CPU socket and DRAM power are reported by pcm-power utility
- NVIDIA GeForce GTX Titan X: reported by nvidia-smi utility
- NVIDIA Tegra K1: measured the total power consumption with a power-meter, 15% AC to DC conversion loss, 85% regulator efficiency and 15% power consumed by peripheral components => 60% AP+DRAM power

2. QUANTIZATION AND WEIGHT SHARING



WEIGHT SHARING: OVERVIEW

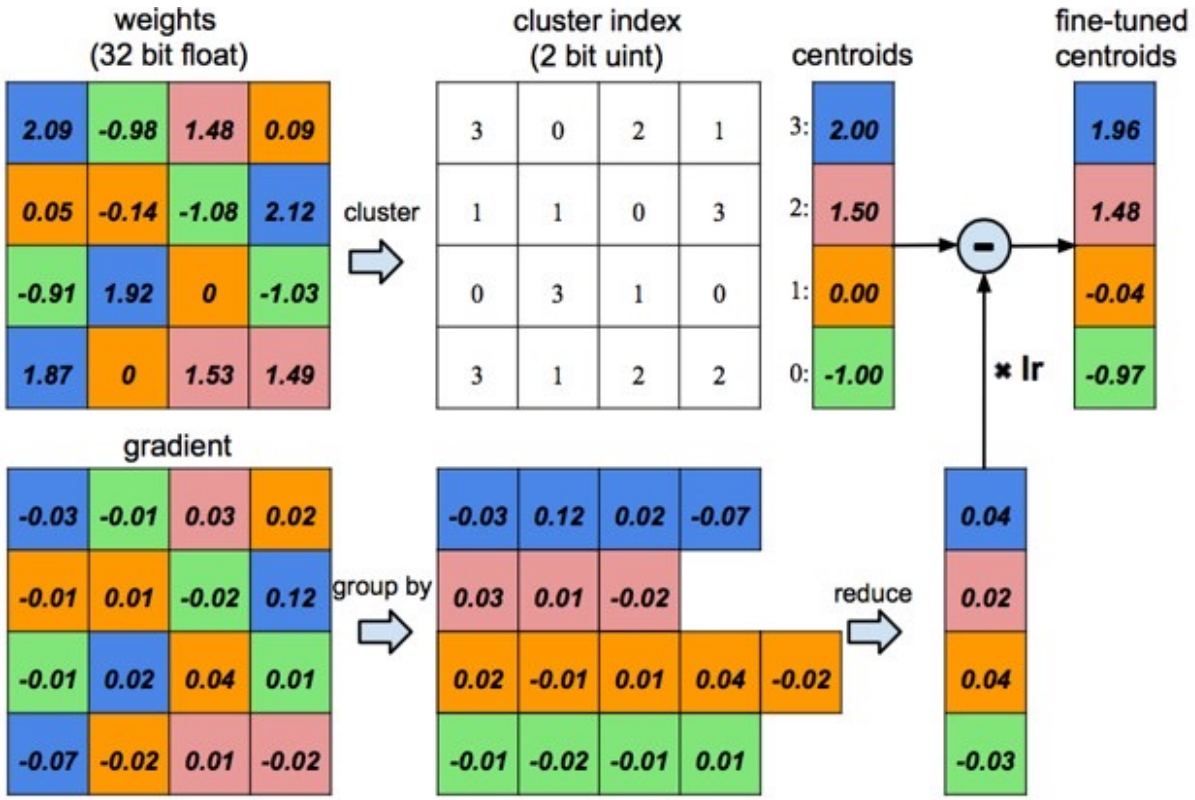
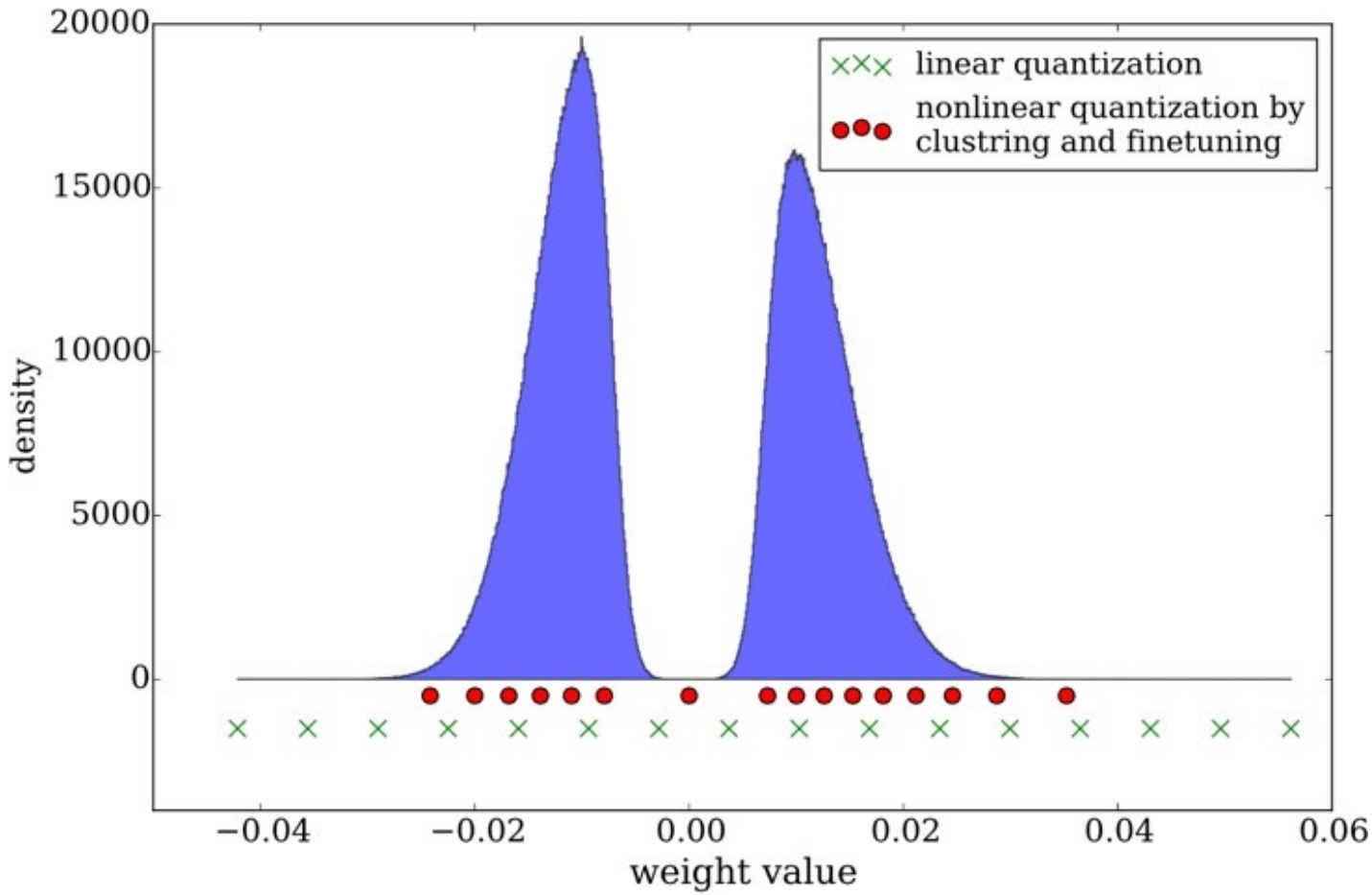


Figure 3: Weight sharing by scalar quantization (top) and centroids fine-tuning (bottom)

FINETUNE CENTROIDS



Parameter Quantization

- 1. Using less bits to represent a value
- 2. Weight clustering

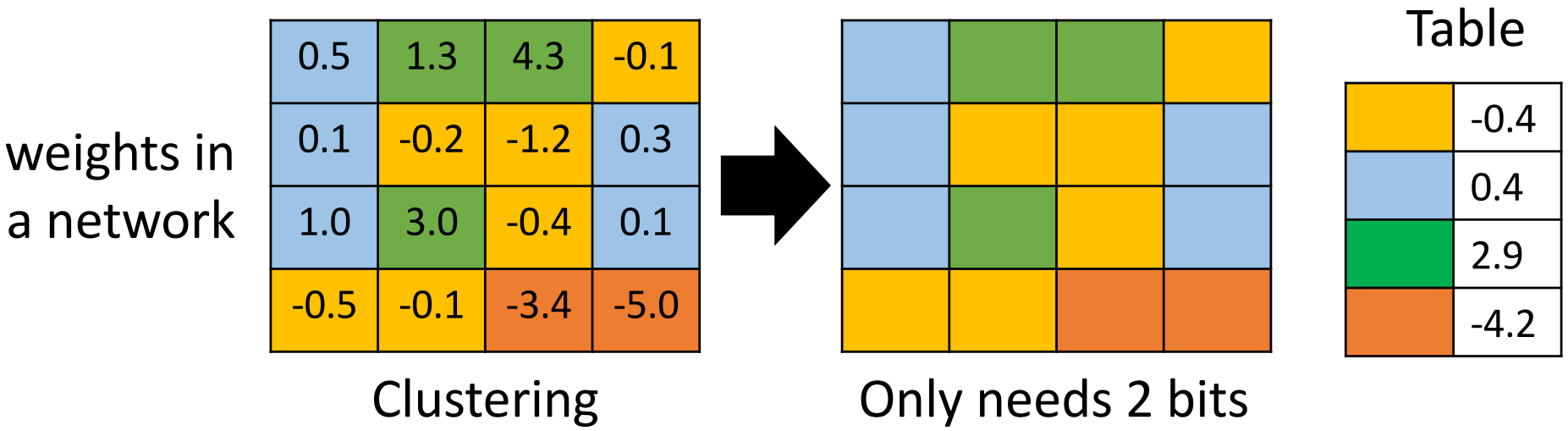
weights in a network

0.5	1.3	4.3	-0.1
0.1	-0.2	-1.2	0.3
1.0	3.0	-0.4	0.1
-0.5	-0.1	-3.4	-5.0

Clustering

Parameter Quantization

- 1. Using less bits to represent a value
- 2. Weight clustering



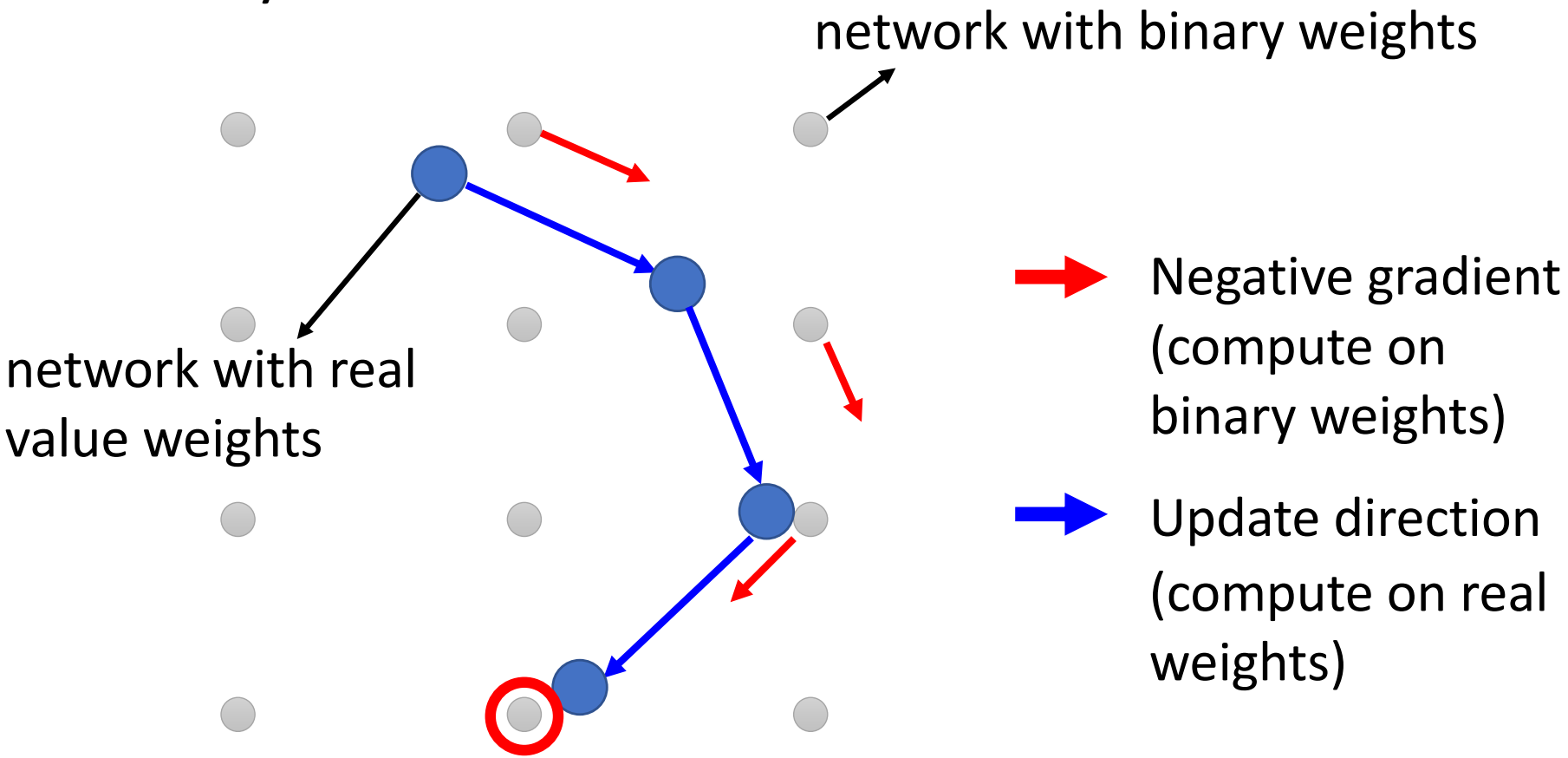
- 3. Represent frequent clusters by less bits, represent rare clusters by more bits
  - e.g. Huffman encoding

Binary Weights

Your weights are always +1 or -1

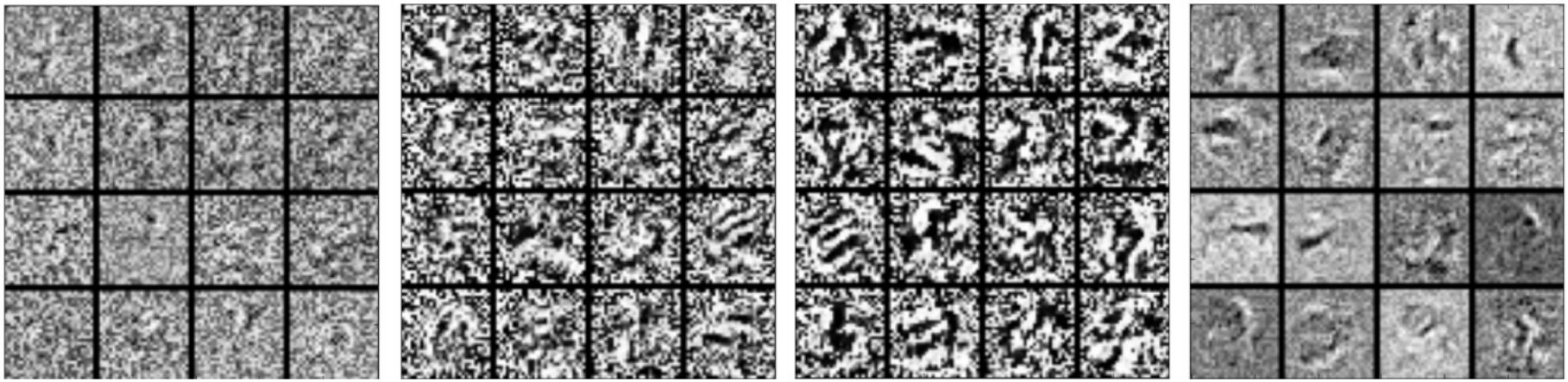
- Binary Connect

Binary Connect:
https://arxiv.org/abs/1511.00363
Binary Network:
https://arxiv.org/abs/1602.02830
XNOR-net:
https://arxiv.org/abs/1603.05279



Binary Weights

Method	MNIST	CIFAR-10	SVHN
No regularizer	$1.30 \pm 0.04\%$	10.64%	2.44%
BinaryConnect (det.)	$1.29 \pm 0.08\%$	9.90%	2.30%
BinaryConnect (stoch.)	$1.18 \pm 0.04\%$	**8.27%**	2.15%
50% Dropout	$1.01 \pm 0.04\%$		

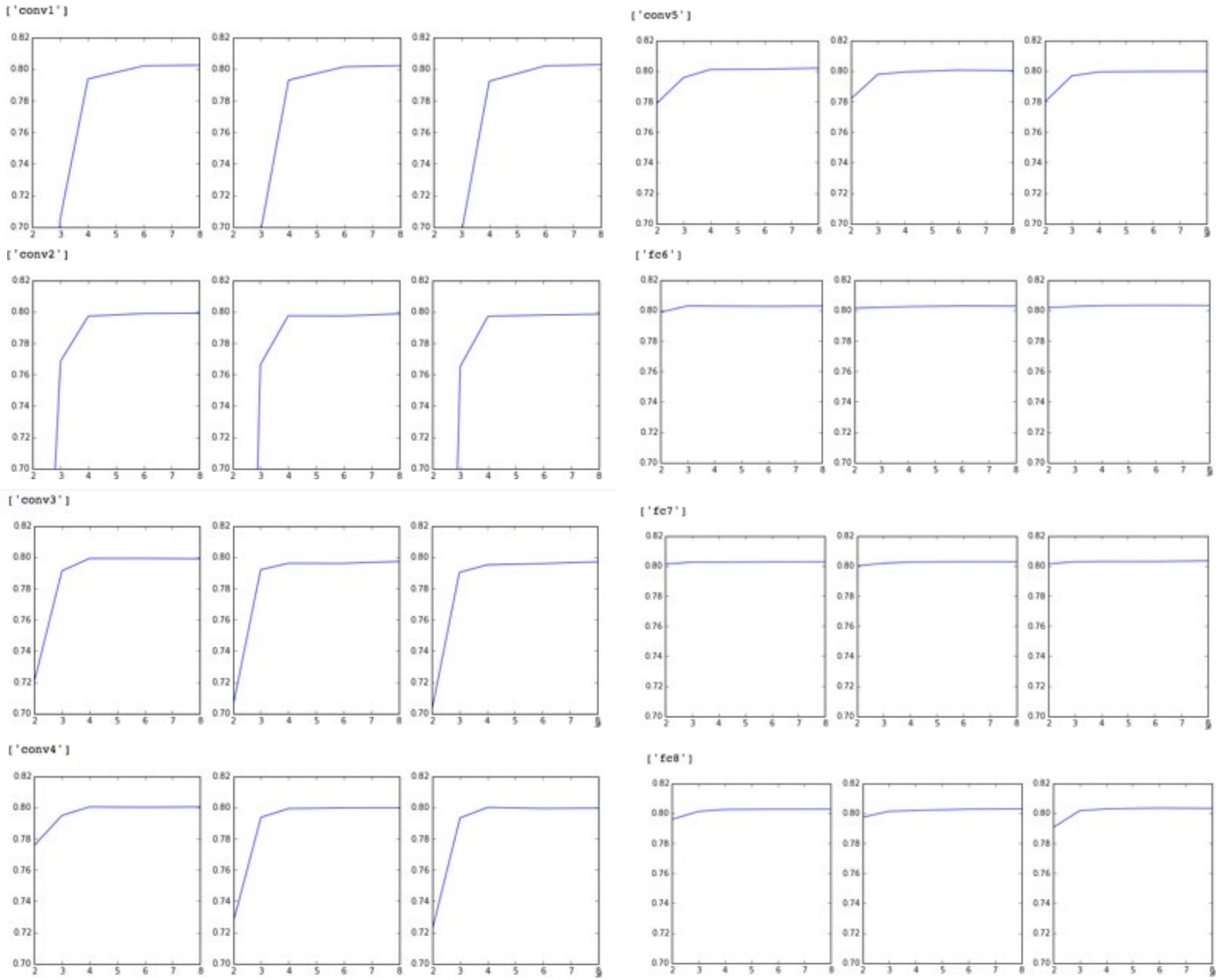


https://arxiv.org/abs/1511.00363

QUANTIZATION: RESULT

- 8/5 bit quantization results in **no** accuracy loss
- 8/4 bit quantization results in no top-5 accuracy loss, **0.1%** top-1 accuracy loss
- 4/2 bit quantization results in **-1.99%** top-1 accuracy loss, and **-2.60%** top-5 accuracy loss, not that bad-:

ACCURACY ~ #BITS ON 5 CONV LAYER + 3 FC LAYER



PRUNING AND QUANTIZATION WORK WELL TOGETHER

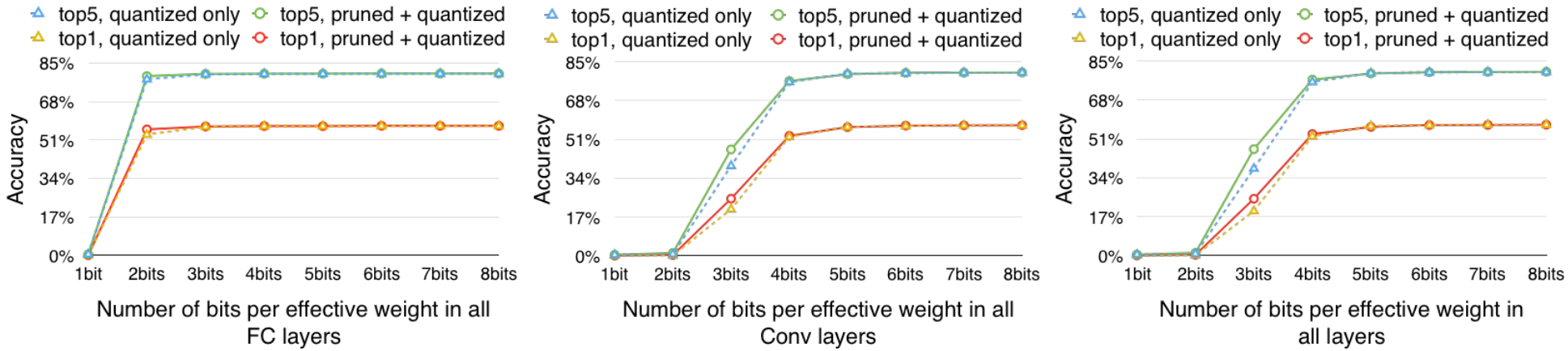
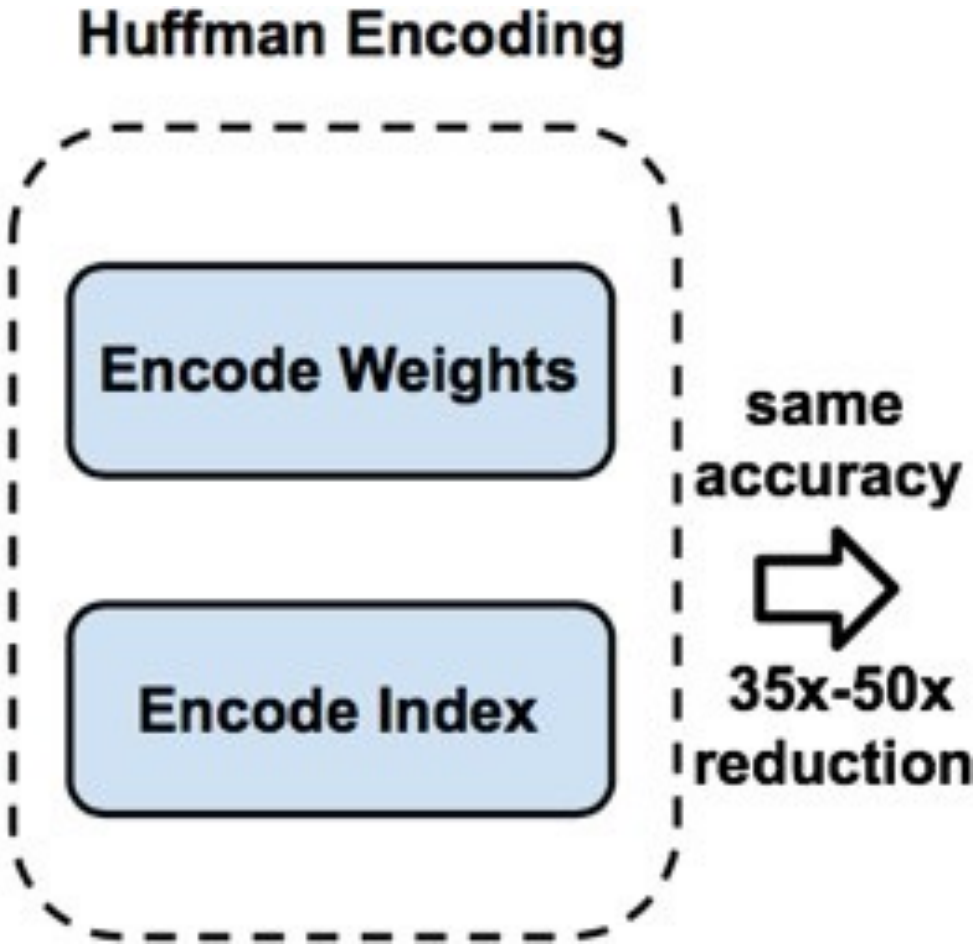


Figure 7: Pruning doesn't hurt quantization. Dashed: quantization on unpruned network. Solid: quantization on pruned network; Accuracy begins to drop at the same number of quantization bits whether or not the network has been pruned. Although pruning made the number of parameters less, quantization still works well, or even better(3 bits case on the left figure) as in the unpruned network.

3. HUFFMAN CODING



HUFFMAN CODING

Huffman code is a type of optimal prefix code that is commonly used for loss-less data compression. It produces a variable-length code table for encoding source symbol. The table is derived from the occurrence probability for each symbol. As in other entropy encoding methods, more common symbols are represented with fewer bits than less common symbols, thus save the total space.

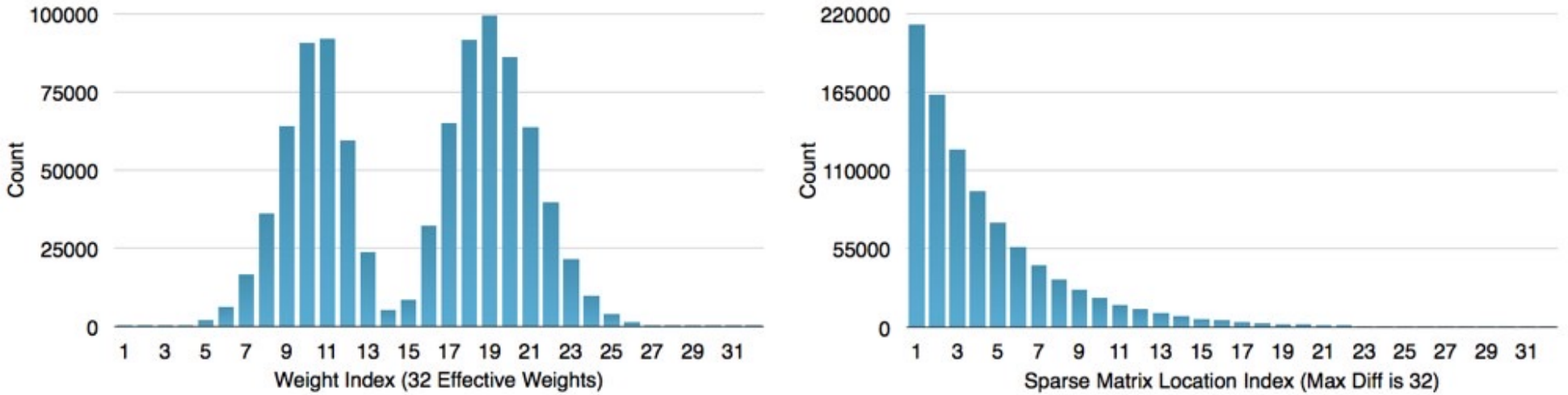


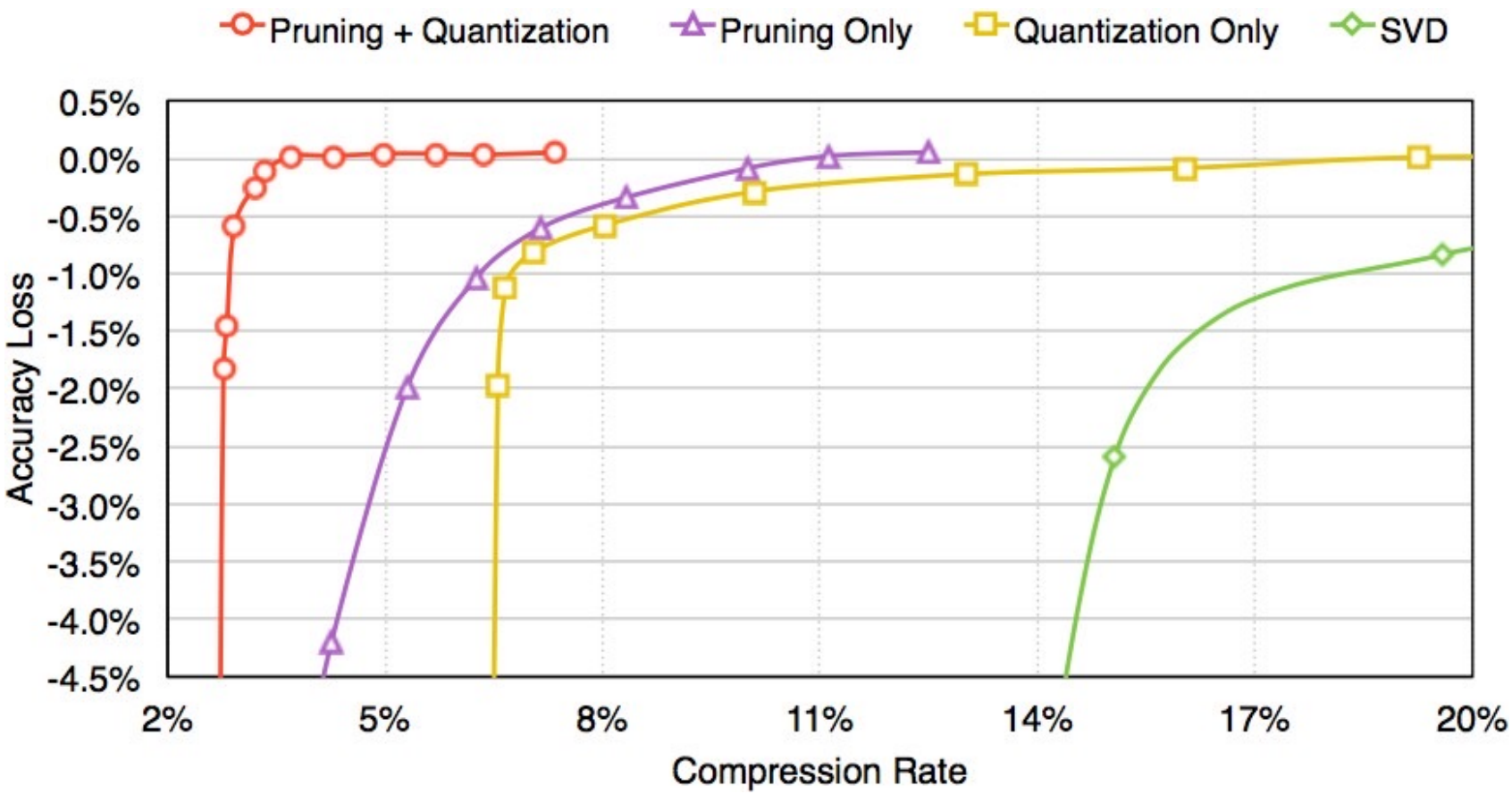
Figure 5: Distribution for weight (Left) and index (Right). The distribution is biased and can be compressed by Huffman encoding

DEEP COMPRESSION RESULT ON 4 CONVNETS

Network	Top-1 Error	Top-5 Error	Parameters	Compress Rate
LeNet-300-100 Ref	1.64%	-	1070 KB	
LeNet-300-100 Compressed	1.58%	-	**27 KB**	**40**×
LeNet-5 Ref	0.80%	-	1720 KB	
LeNet-5 Compressed	0.74%	-	**44 KB**	**39**×
AlexNet Ref	42.78%	19.73%	240 MB	
AlexNet Compressed	42.78%	19.70%	**6.9 MB**	**35**×
VGG16 Ref	31.50%	11.32%	552 MB	
VGG16 Compressed	31.17%	10.91%	**11.3 MB**	**49**×

Table 1: The compression pipeline can save 35× to 49× parameter storage with no drop in predictive performance

RESULT: ALEXNET



ALEXNET: BREAKDOWN

Layer	#Weights	Weights% (P)	Weigh bits (P+Q)	Weight bits (+H)	Index bits (P+Q)	Index bits (+H)	Compress rate (P+Q)	Compress rate (P+Q+H)
conv1	35K	84%	8	6.3	4	1.2	32.6%	20.53%
conv2	307K	38%	8	5.5	4	2.3	14.5%	9.43%
conv3	885K	35%	8	5.1	4	2.6	13.1%	8.44%
conv4	663K	37%	8	5.2	4	2.5	14.1%	9.11%
conv5	442K	37%	8	5.6	4	2.5	14.0%	9.43%
fc6	38M	9%	5	3.9	4	3.2	3.0%	2.39%
fc7	17M	9%	5	3.6	4	3.7	3.0%	2.46%
fc8	4M	25%	5	4	4	3.2	7.3%	5.85%
total	61M	11%	5.4	4	4	3.2	3.7%	2.88%

Table 4: Compression Statistics for Alexnet. P: pruning, Q:quantization, H:Huffman Encoding

COMPARISON WITH OTHER COMPRESSION METHODS

Network	Top-1 Error	Top-5 Error	Parameters	Compress Rate
Baseline Caffemodel [21]	42.78%	19.73%	240MB	1×
Fastfood-32-AD [22]	41.93%	-	131MB	2×
Fastfood-16-AD [22]	42.90%	-	64MB	3.7×
Collins & Kohli [23]	44.40%	-	61MB	4×
SVD [14]	44.02%	20.56%	55.2MB	5×
Pruning [6]	42.77%	19.67%	27MB	9×
Pruning+Quantization	42.78%	19.70%	8.9MB	27×
Pruning+Quantization+Huffman	**42.78%**	**19.70%**	**6.9MB**	**35×**

Table 6: Comparison with other model reduction methods on AlexNet. [23] reduced the parameters by 4× and with inferior accuracy. Deep Fried Convnets [22] worked on fully connected layers only and reduced the parameters by less than 4×. SVD save parameters but suffers from large accuracy loss as much as 2%. Network pruning [6] reduced the parameters by 9× without accuracy loss but the compression rate is only one third of this work. On other networks similar to AlexNet, [14] exploited linear structure of convnets and compressed the network by 2.4× to 13.4× layer wise, but had significant accuracy loss: as much as 0.9% even compressing a single layer. [15] experimented with vector quantization and compressed the network by 16 × −24×, but again incurred as much as 1% accuracy loss.

14EmilyLDenton,WojciechZaremba,JoanBruna,YannLeCun,andRobFergus.Exploitinglinearstructure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277, 2014.

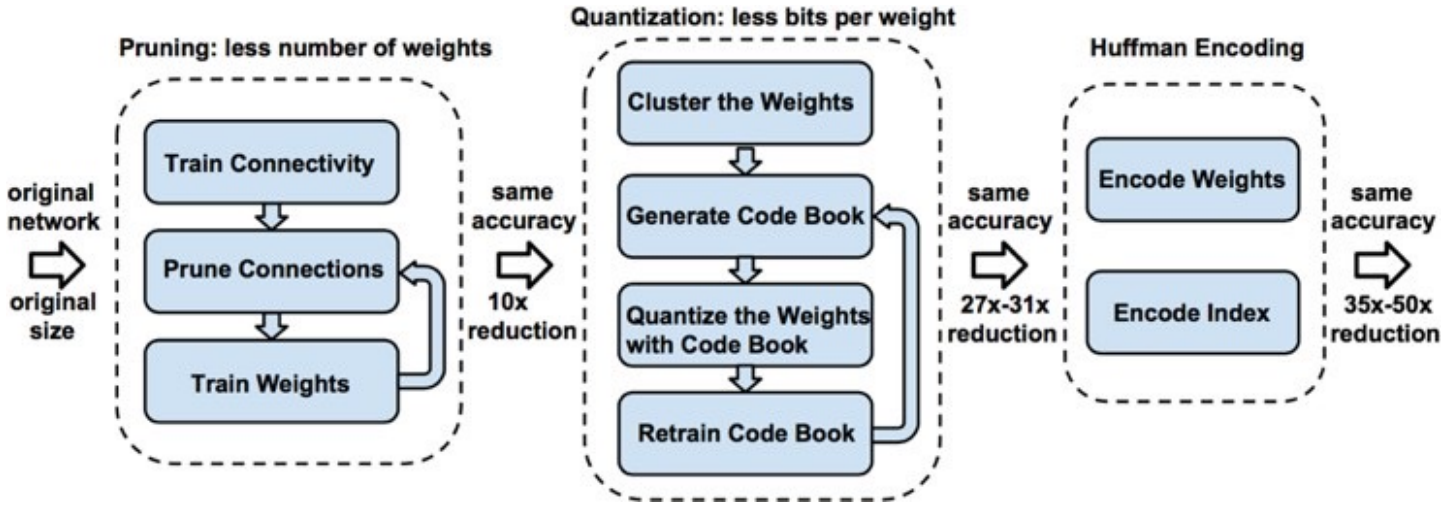
15 Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

21 Yangqing Jia. Bvlc caffe model zoo. ZichaoYang,MarcinMoczulski,MishaDenil,NandodeFreitas,AlexSmola,LeSong,andZiyuWang.

22 Deep fried convnets. *arXiv preprint arXiv:1412.7149*, 2014.

23 Maxwell D Collins and Pushmeet Kohli. Memory bounded deep convolutional networks. *arXiv preprint arXiv:1412.1442*, 2014.

SUMMARY



- We have presented a method to compress neural networks without affecting accuracy by finding the right connections and quantizing the weights.
- Pruning the unimportant connections => quantizing the network and enforce weight sharing => apply Huffman encoding.
- We highlight our experiments on ImageNet, and reduced the weight storage by 35×, VGG16 by 49×, without loss of accuracy.
- Now weights can fit in cache

PRODUCT: A MODEL COMPRESSION TOOL FOR DEEP LEARNING DEVELOPERS

- **Easy Version**:
  - ✓ No training needed
  - ✓ Fast
  - x 5x - 10x compression rate
  - x 1% loss of accuracy
- **Advanced Version:**
  - ✓ 35x - 50x compression rate
  - ✓ no loss of accuracy
  - x Training is needed
  - x Slow

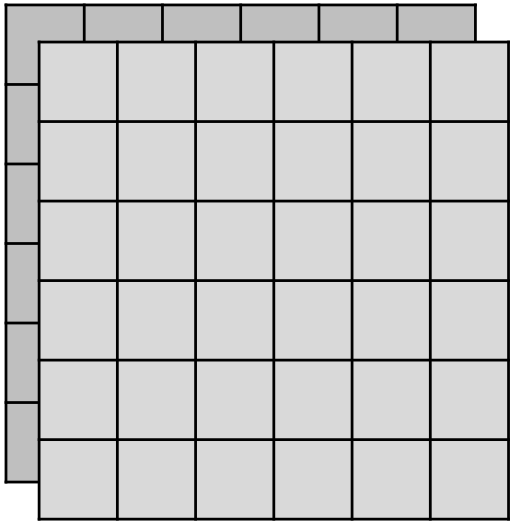
Demo:
Pocket AlexNet

ARCHITECTURE DESIGN

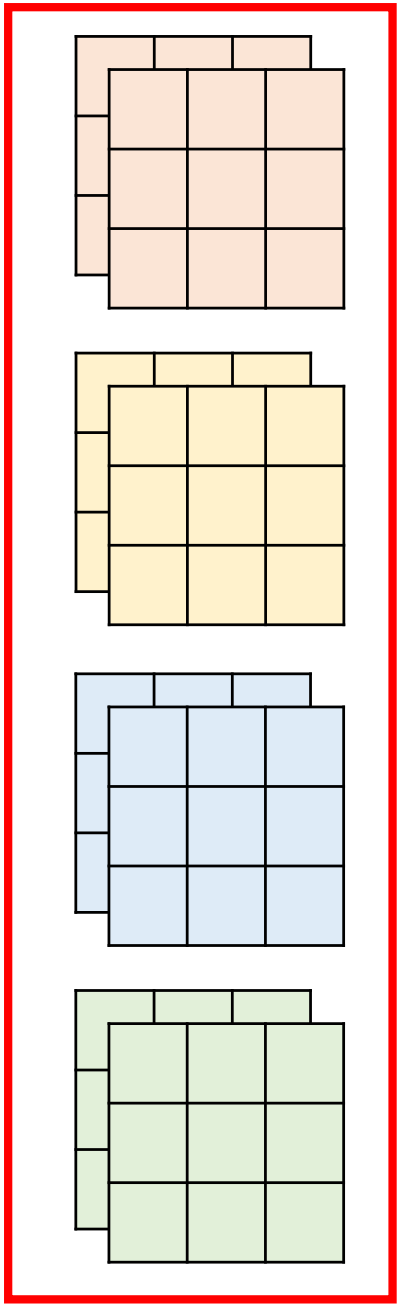
Depthwise Separable Convolution

Review: Standard CNN

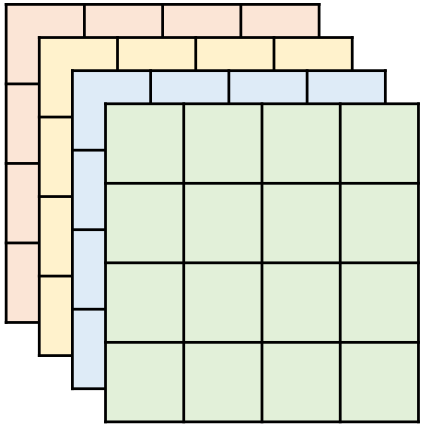
Input feature map



2 channels

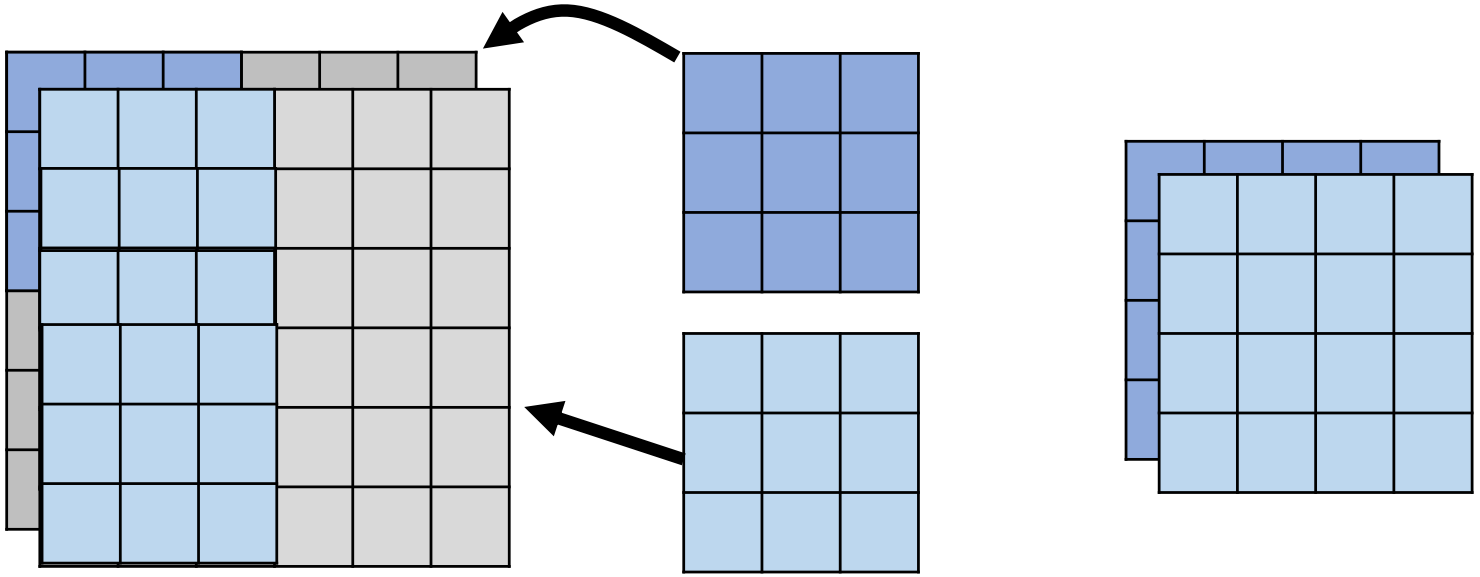


$3\times3\times2\times4 = 72$
parameters



Depthwise Separable Convolution

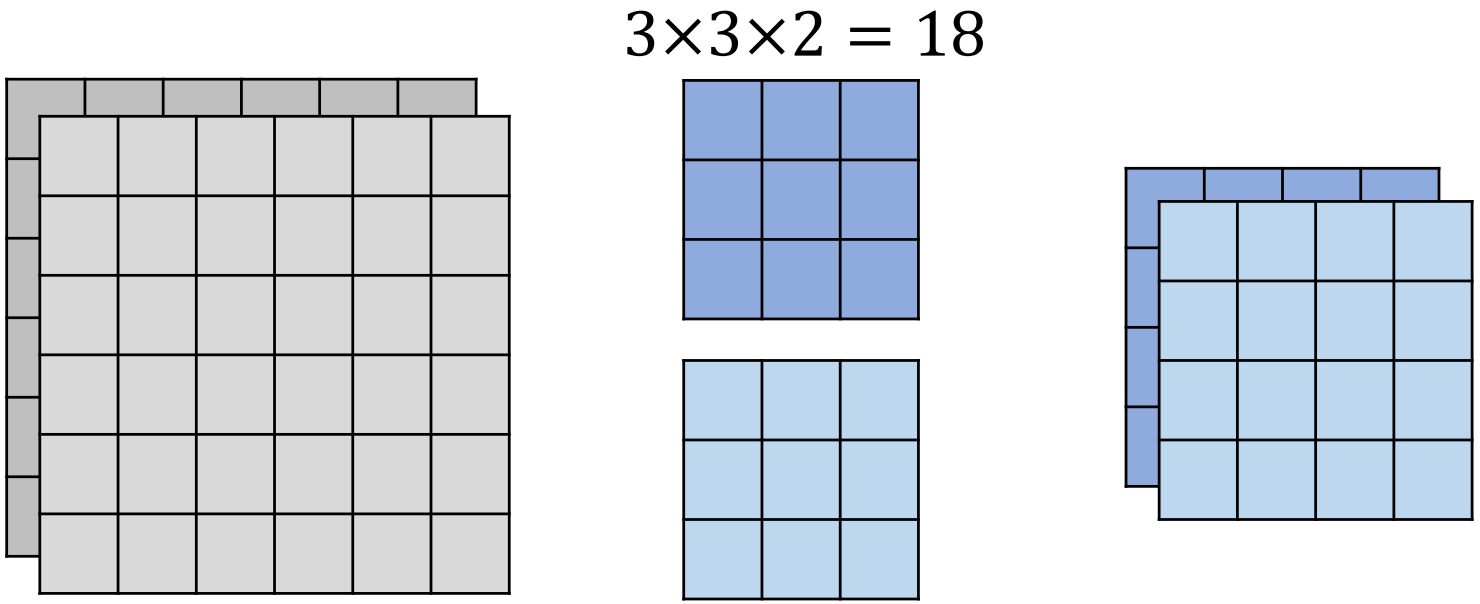
1. Depthwise Convolution



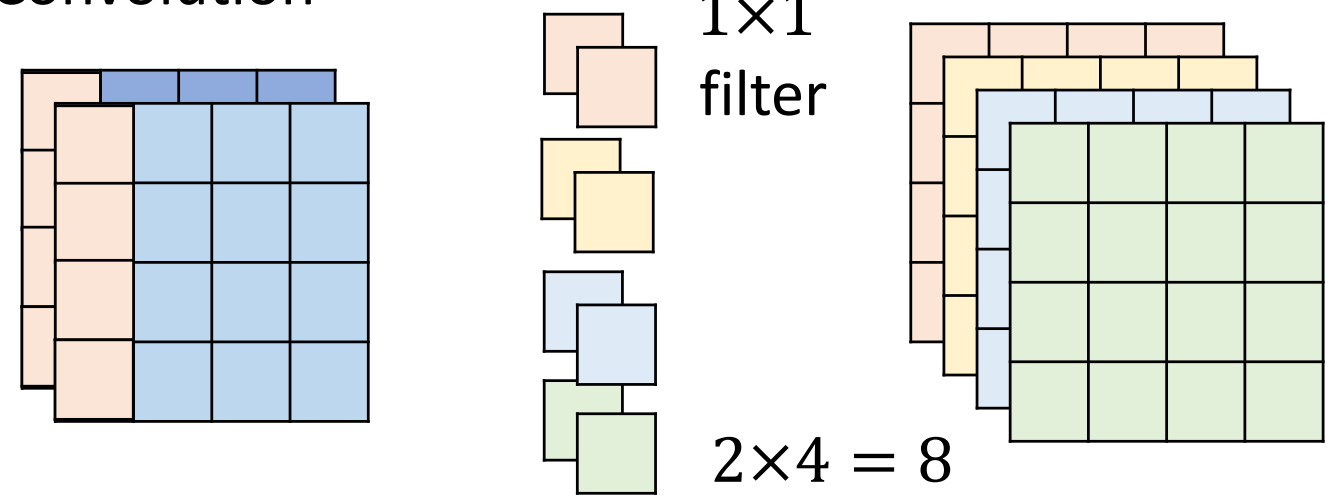
- Filter number = Input channel number
- Each filter only considers one channel.
- The filters are $k \times k$ matrices
- There is no interaction between channels.

Depthwise Separable Convolution

1. Depthwise Convolution



2. Pointwise Convolution



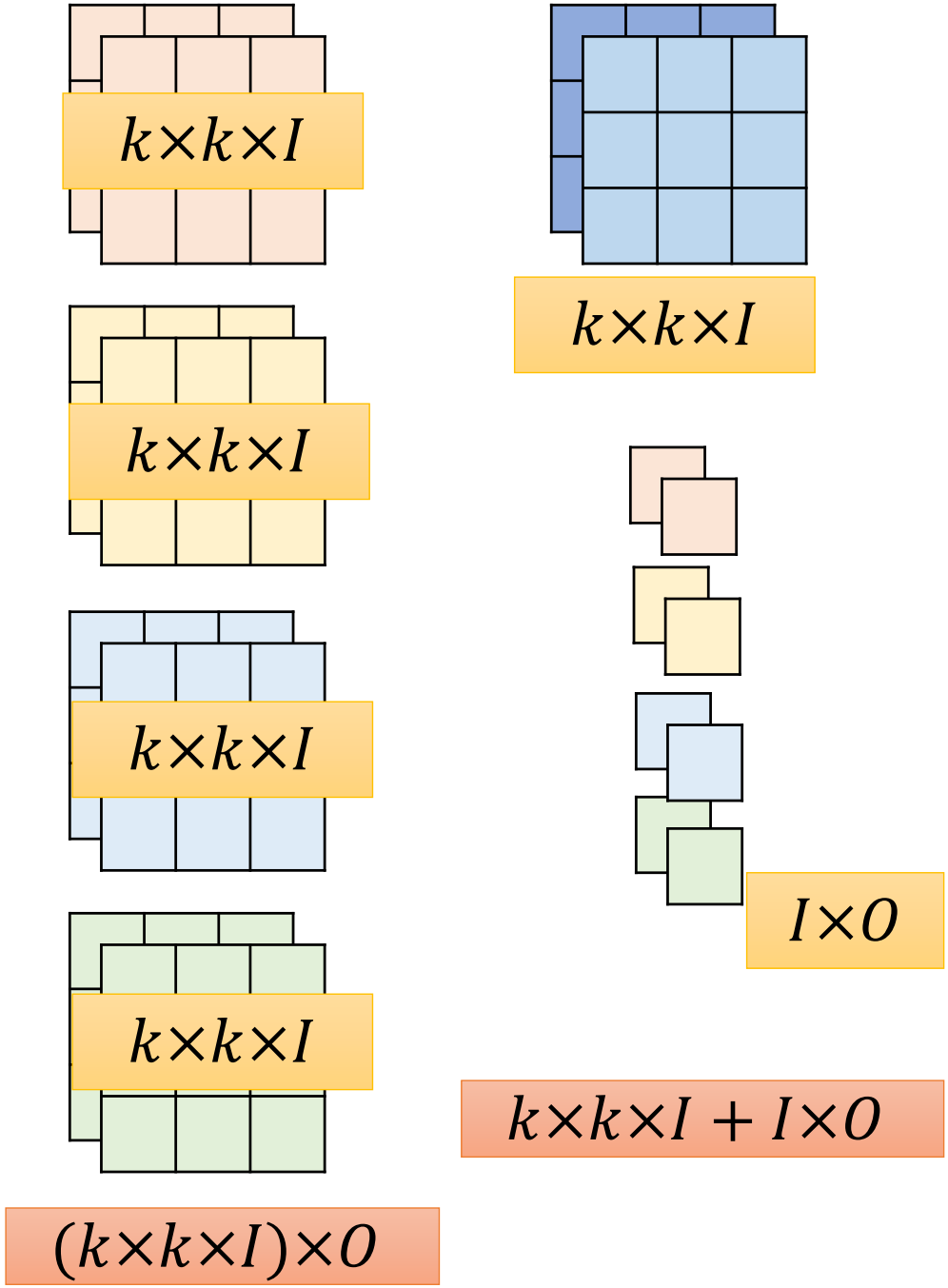
I: number of input channels

O: number of output channels

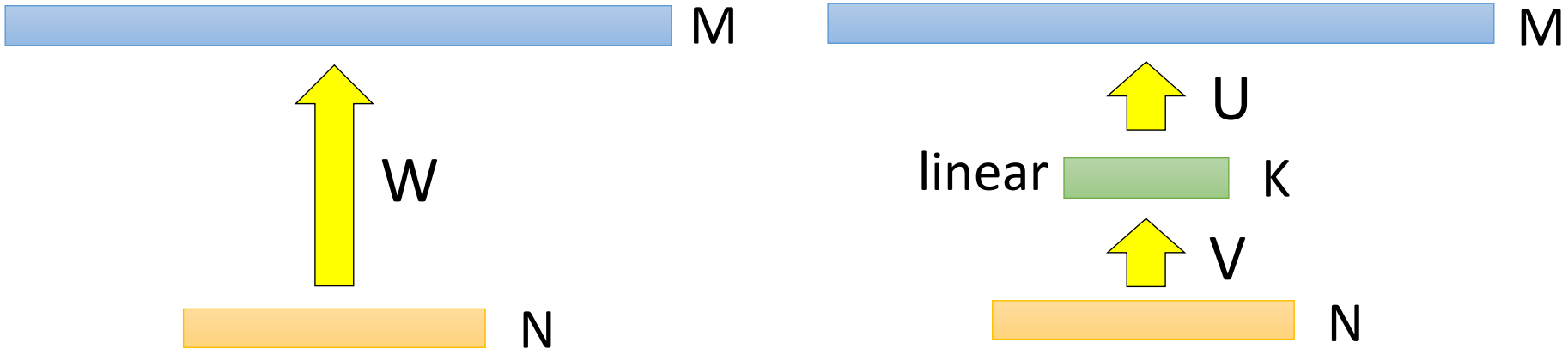
$k \times k$: kernel size

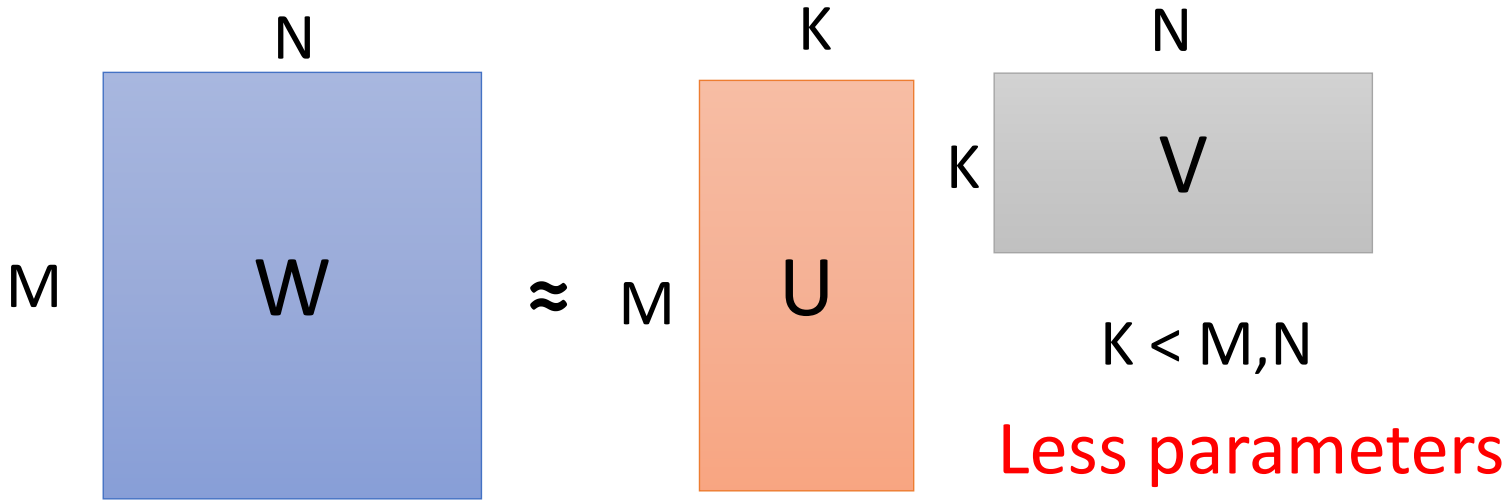
$$\frac{k \times k \times I + I \times O}{k \times k \times I \times O}$$

$$= \frac{1}{O} + \boxed{\frac{1}{k \times k}}$$



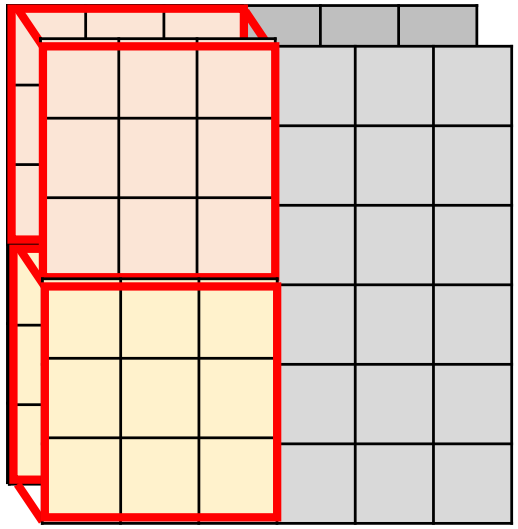
Low rank approximation

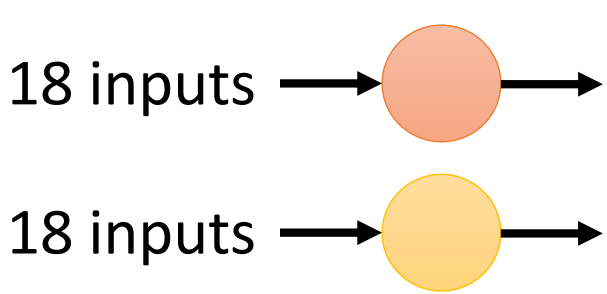


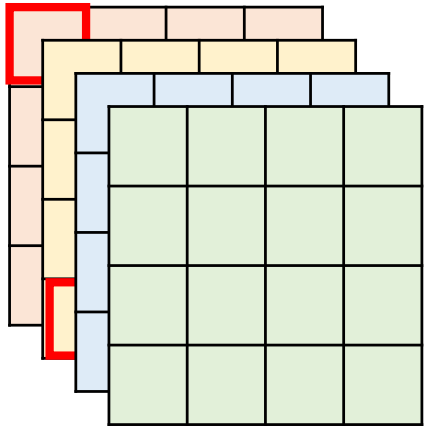


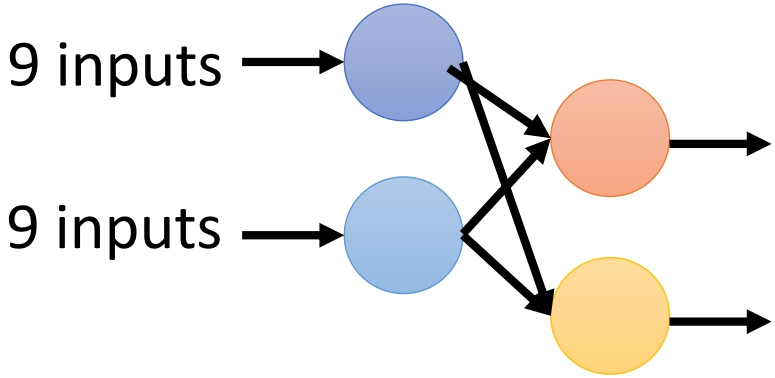
$K < M,N$

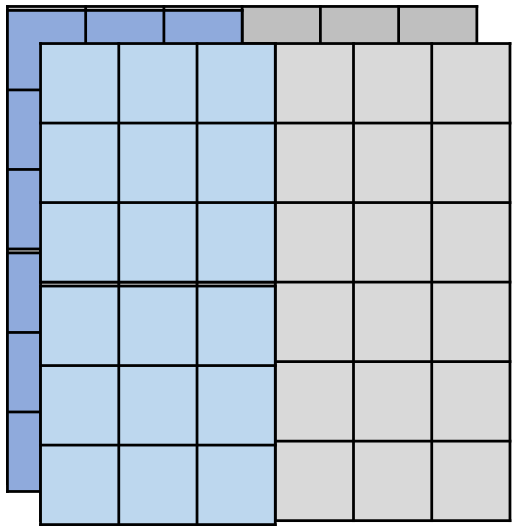
Less parameters

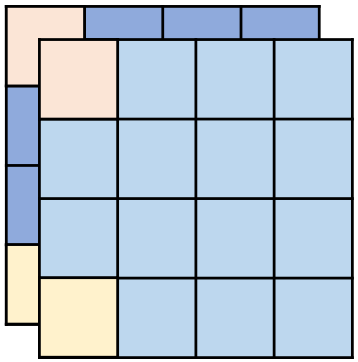


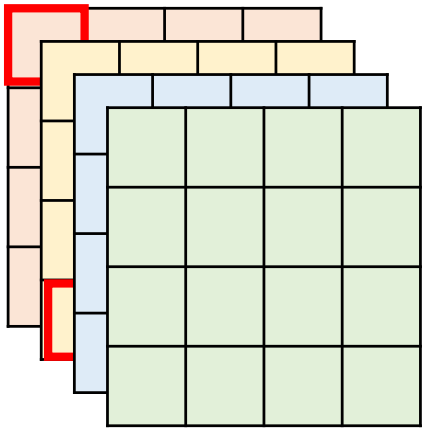

18 inputs
18 inputs




9 inputs
9 inputs







To learn more ……

- SqueezeNet
  - https://arxiv.org/abs/1602.07360
- MobileNet
  - https://arxiv.org/abs/1704.04861
- ShuffleNet
  - https://arxiv.org/abs/1707.01083
- Xception
  - https://arxiv.org/abs/1610.02357
- GhostNet
  - https://arxiv.org/abs/1911.11907

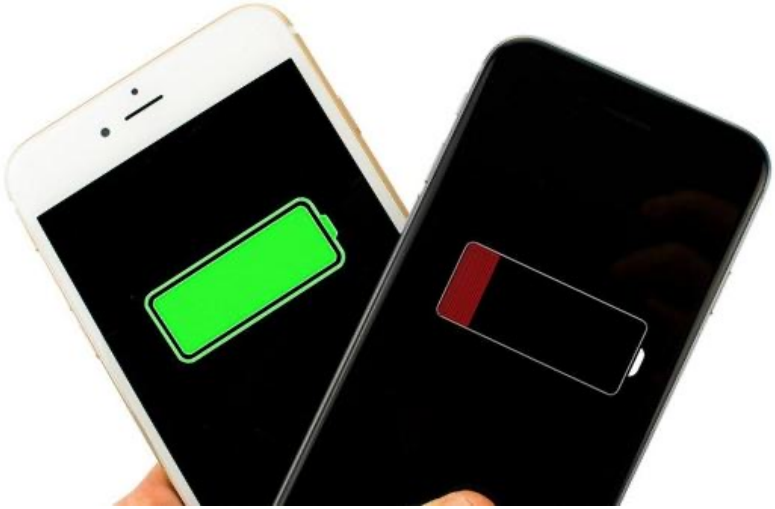
DYNAMIC COMPUTATION

Dynamic Computation

- The network adjusts the computation it need.

Different devices

high/low battery



- Why don’t we prepare a set of models?

Dynamic Depth

$$L = e_1 + e_2 + \cdots + e_L$$

high battery

Layer 1 → Layer 2 → … → Layer L → $y \dashleftarrow\dashrightarrow \hat{y}$ (e_L)

Extra Layer 1 → y' (e_1) → $\hat{y}$

Extra Layer 2 → y'' (e_2) → $\hat{y}$

low battery

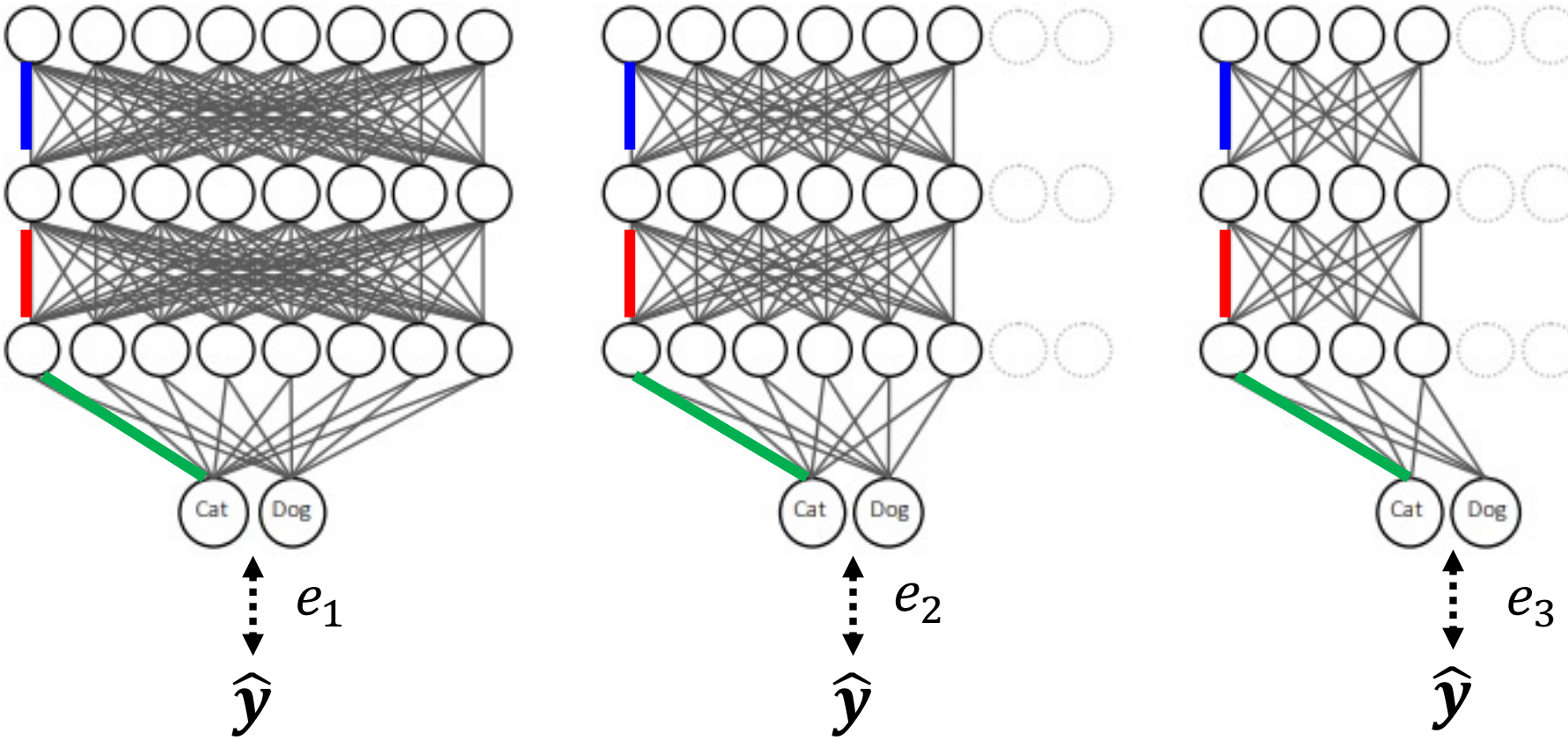
Does it work well?

Multi-Scale Dense Network (MSDNet)

https://arxiv.org/abs/1703.09844

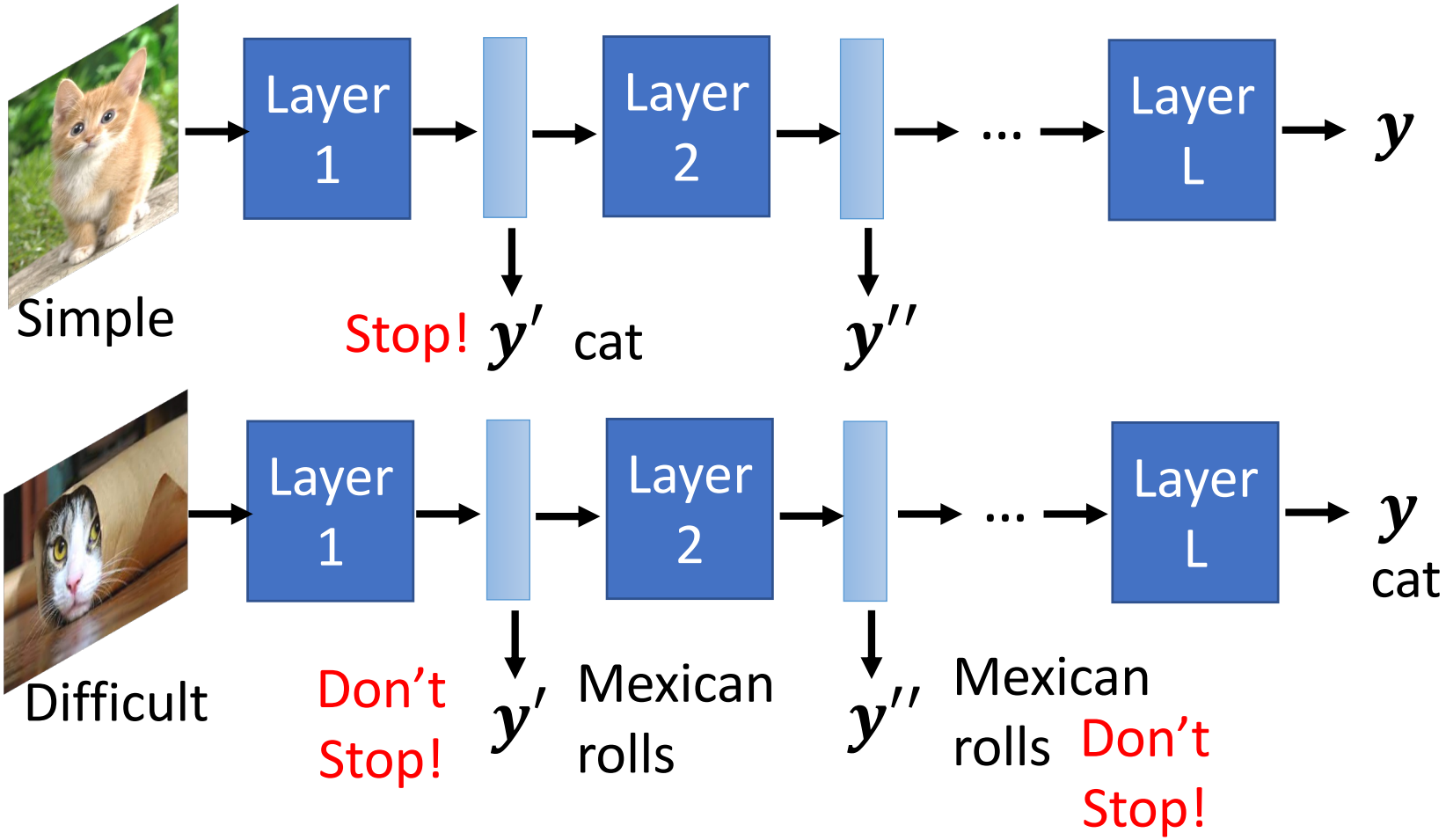
Dynamic Width

$$L = e_1 + e_2 + e_3$$



Slimmable Neural Networks
https://arxiv.org/abs/1812.08928

Computation based on Sample Difficulty

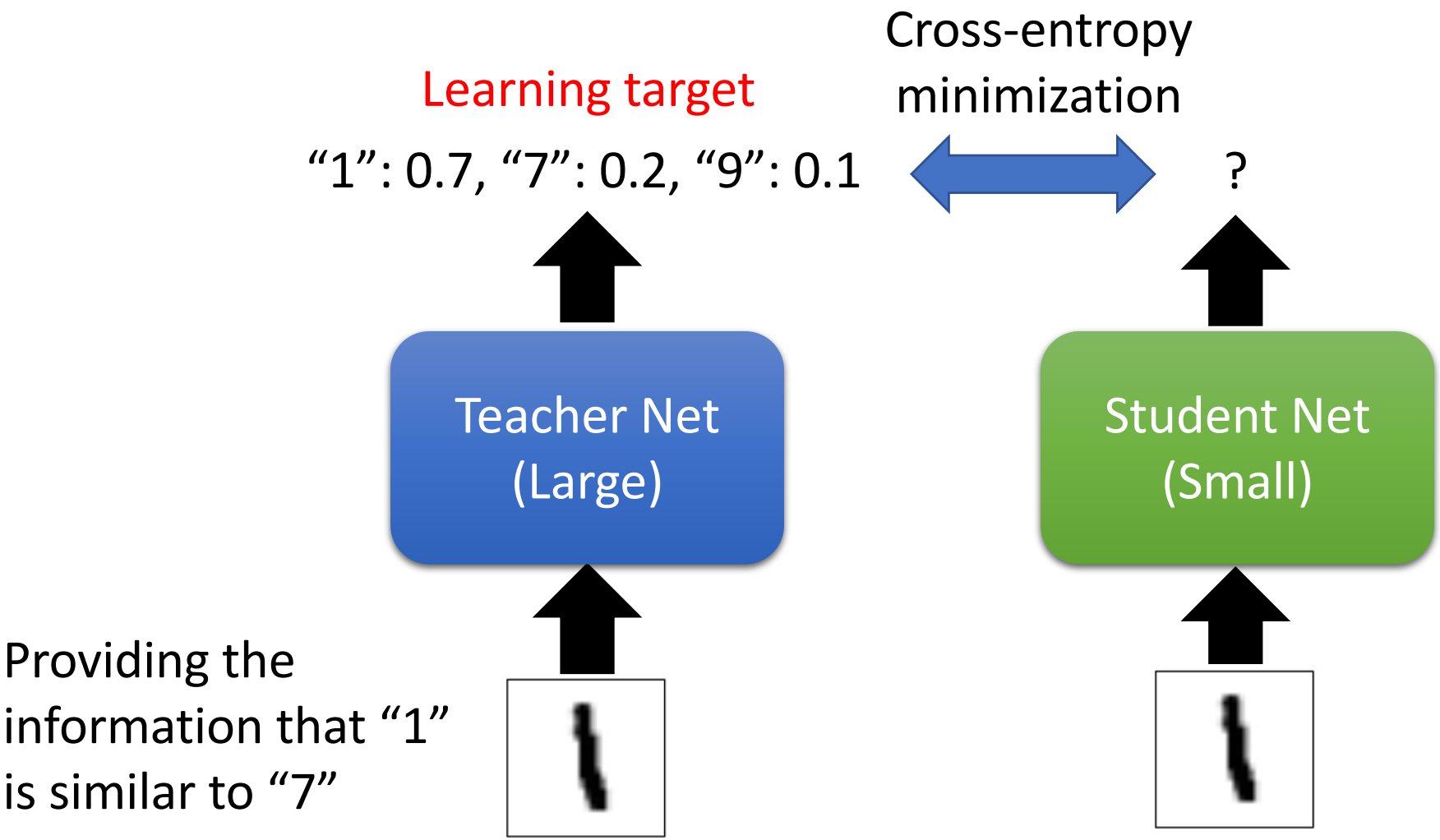


- SkipNet: Learning Dynamic Routing in Convolutional Networks
- Runtime Neural Pruning
- BlockDrop: Dynamic Inference Paths in Residual Networks

KNOWLEDGE DISTILLATION

Knowledge Distillation

Knowledge Distillation
https://arxiv.org/pdf/1503.02531.pdf
Do Deep Nets Really Need to be Deep?
https://arxiv.org/pdf/1312.6184.pdf

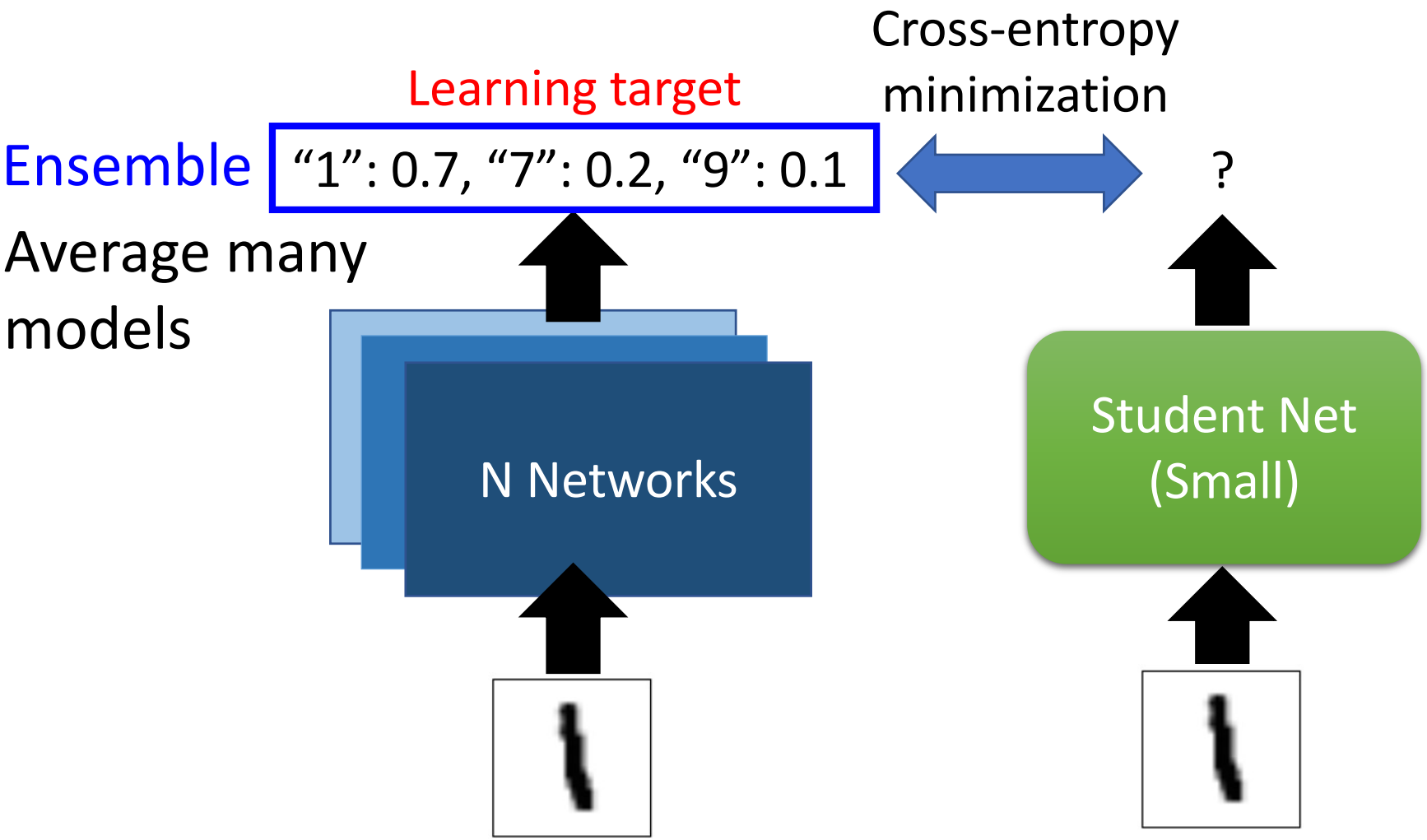


Hard vs. Soft

	cat	dog	car
hard label	0	1	0
soft label	0.09	0.9	0.01

Knowledge Distillation

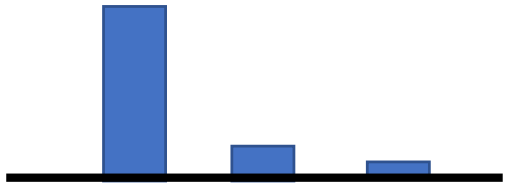
Knowledge Distillation
https://arxiv.org/pdf/1503.02531.pdf
Do Deep Nets Really Need to be Deep?
https://arxiv.org/pdf/1312.6184.pdf

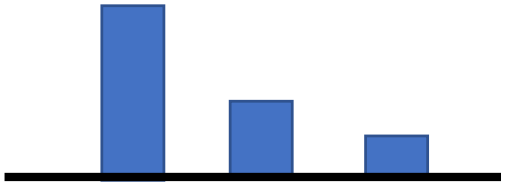


Knowledge Distillation

- Temperature for softmax

$$y_i' = \frac{exp(y_i)}{\sum_j exp(y_j)} \xrightarrow{T = 100} y_i' = \frac{exp(y_i/T)}{\sum_j exp(y_j/T)}$$





$y_1 = 100$ $y_1' = 1$

$y_2 = 10$ $y_2' \approx 0$

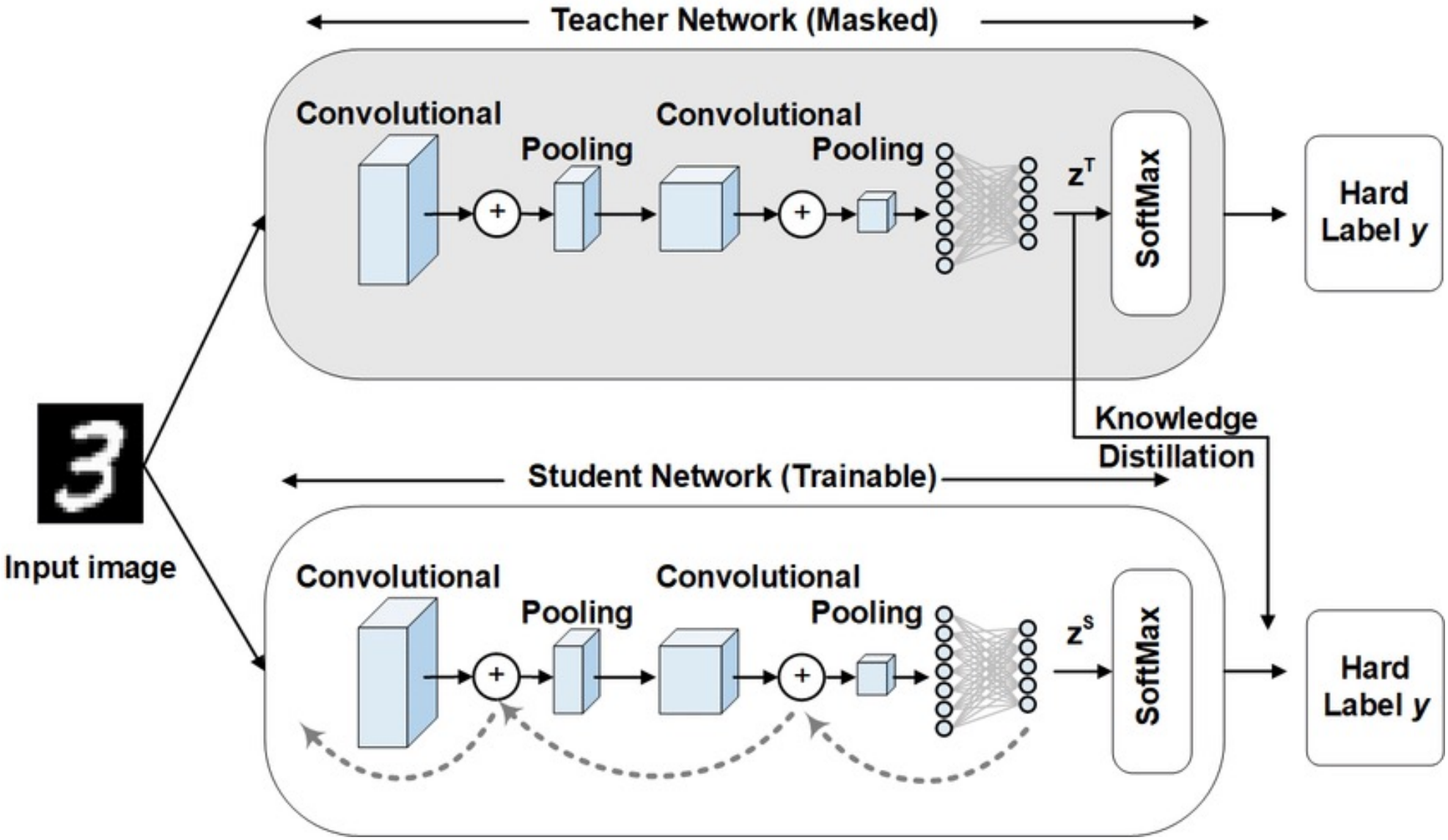
$y_3 = 1$ $y_3' \approx 0$

$y_1/T = 1$ $y_1' = 0.56$

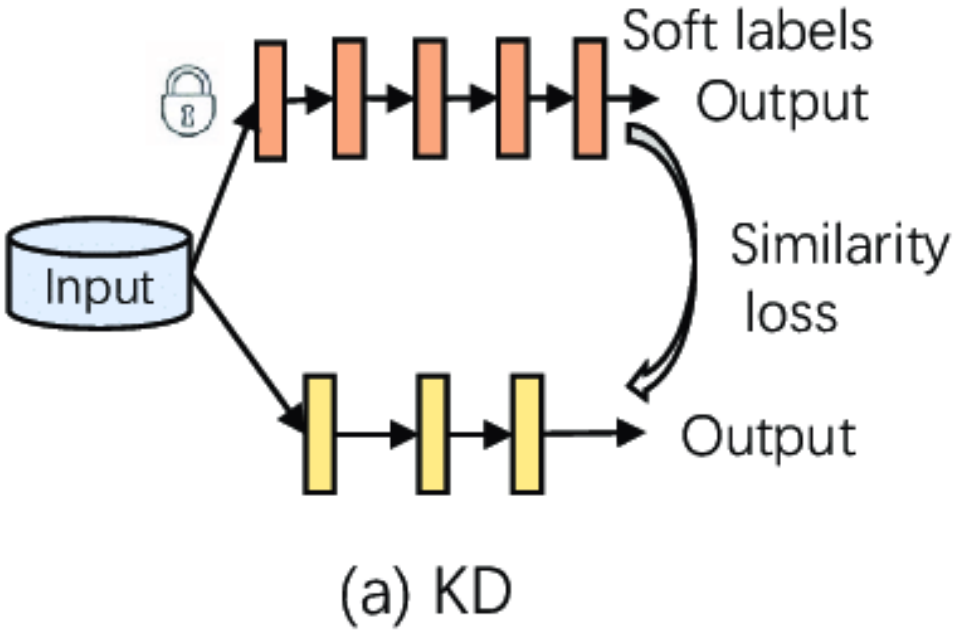
$y_2/T = 0.1$ $y_2' = 0.23$

$y_3/T = 0.01$ $y_3' = 0.21$

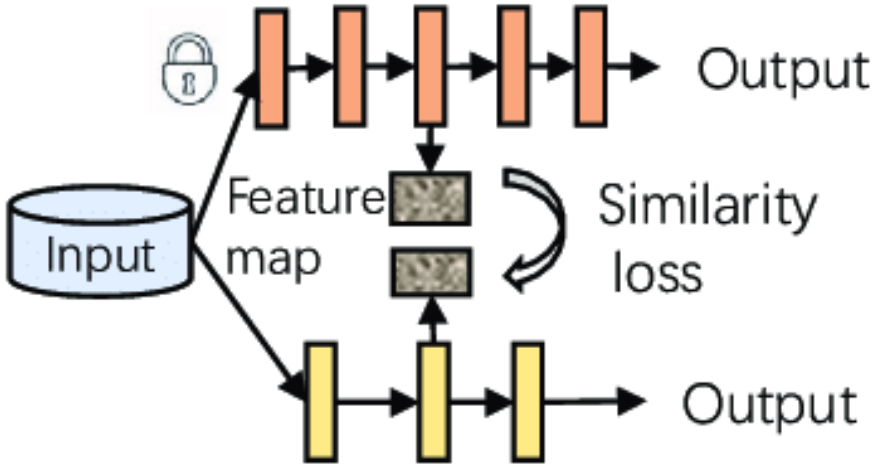
EXAMPLE


Teacher Network (Masked)
Convolutional
Pooling
Convolutional
Pooling
z^T
SoftMax
Hard Label *y*
Knowledge Distillation
Student Network (Trainable)
Input image
Convolutional
Pooling
Convolutional
Pooling
z^S
SoftMax
Hard Label *y*

LOGIT VS FEATURE

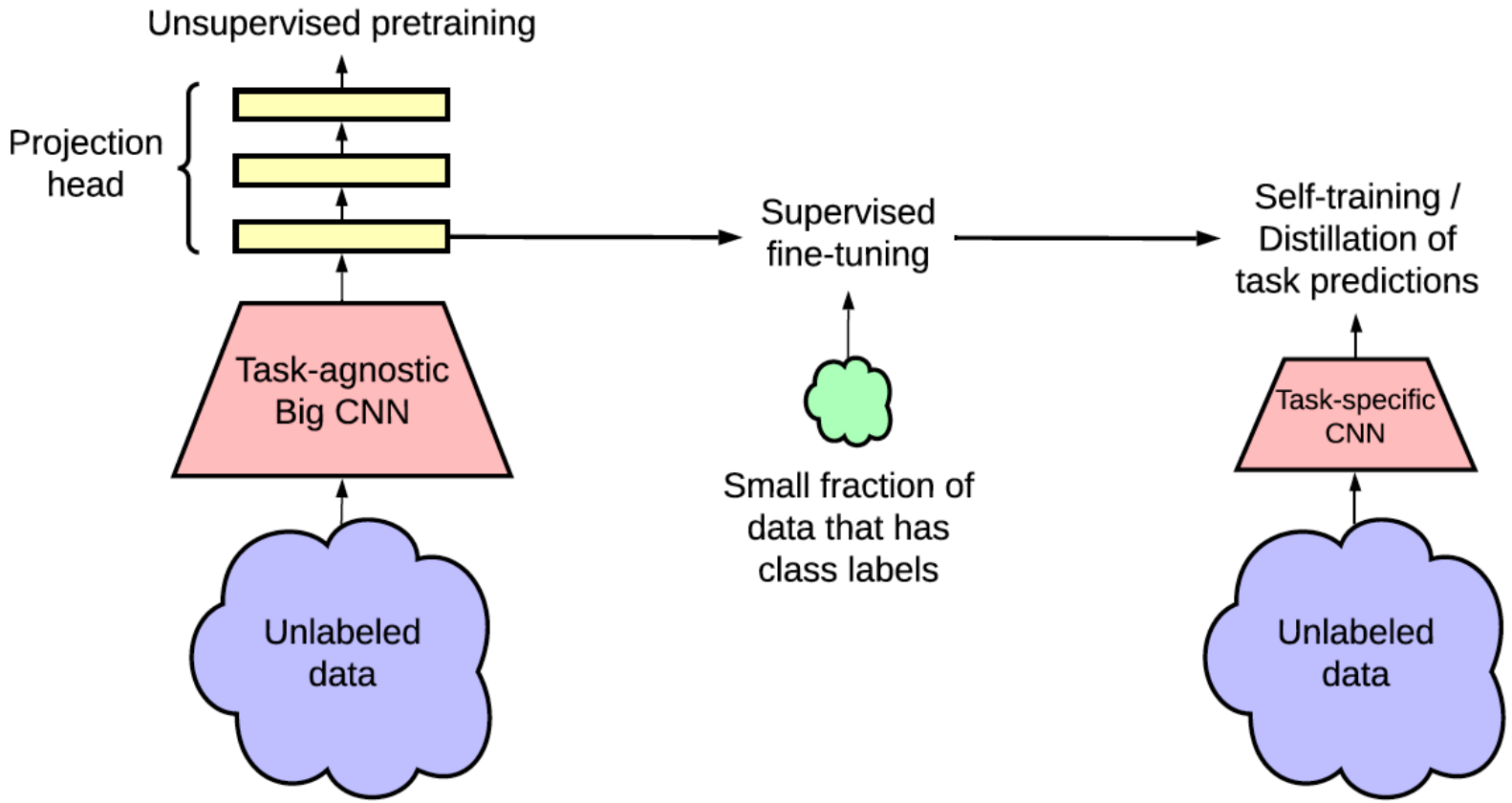


(a) KD



(b) FD

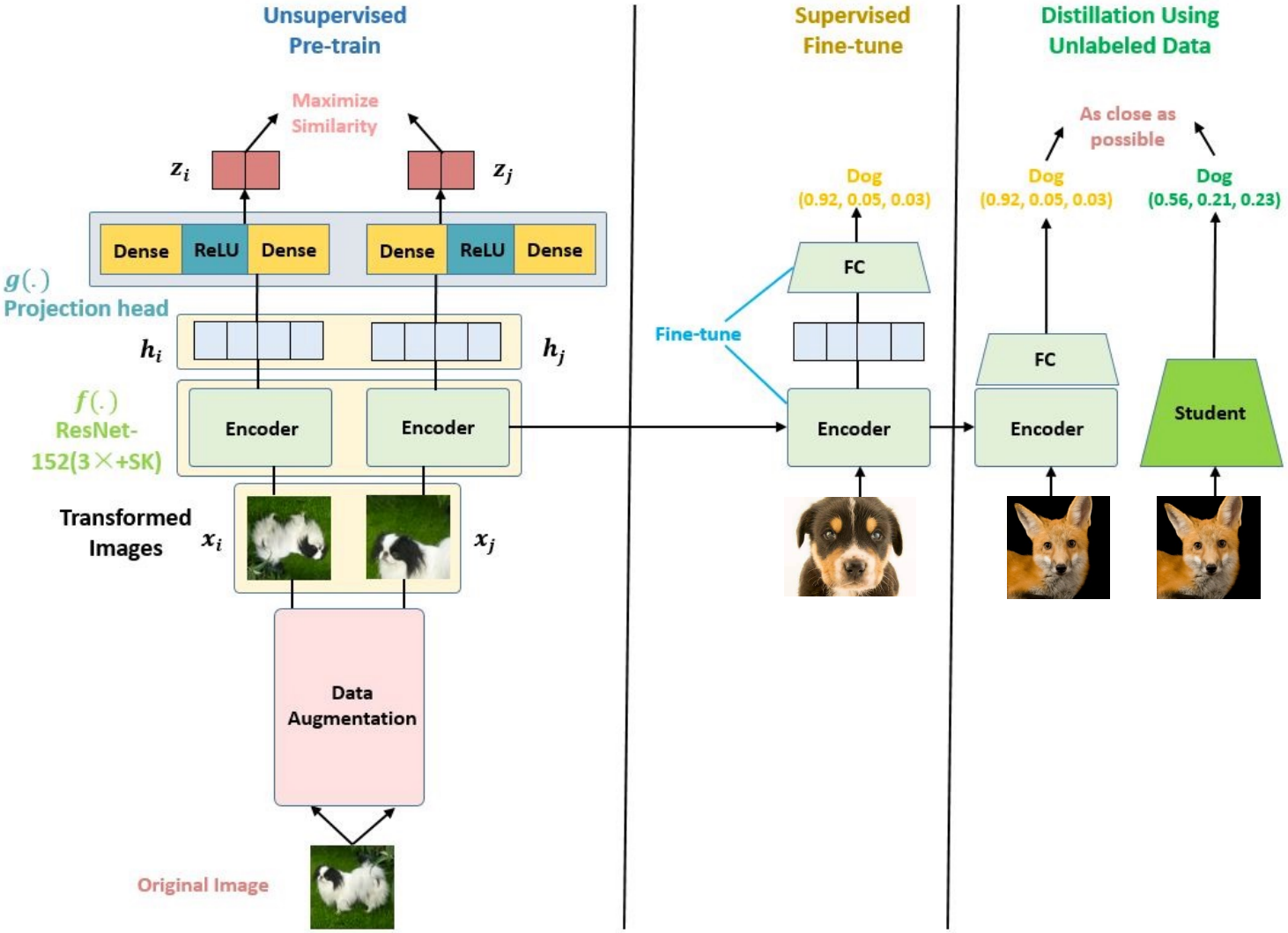
SELF-DISTILLATION



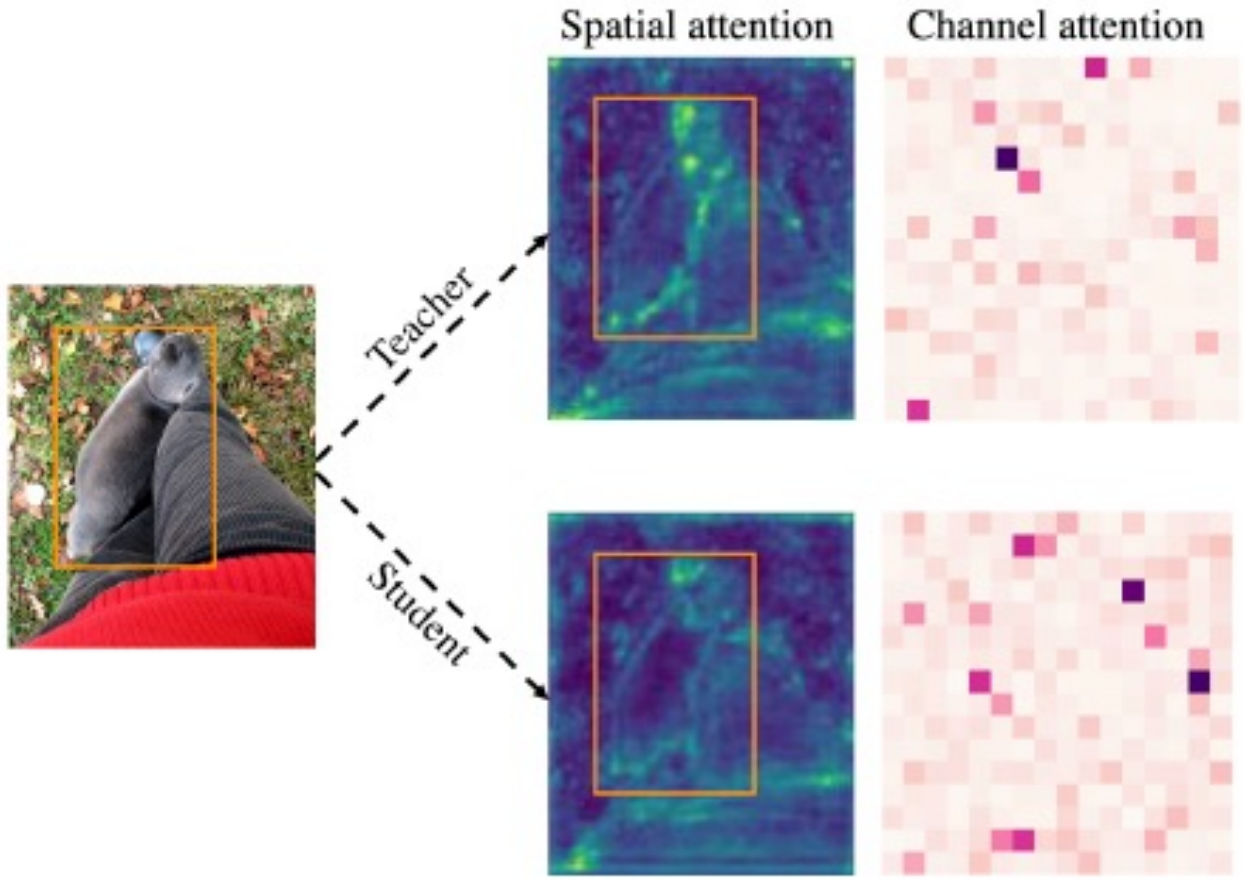
SimCLRv2: Big Self-Supervised Models are Strong Semi-Supervised Learners

https://arxiv.org/abs/2006.10029

SIMCLRV2

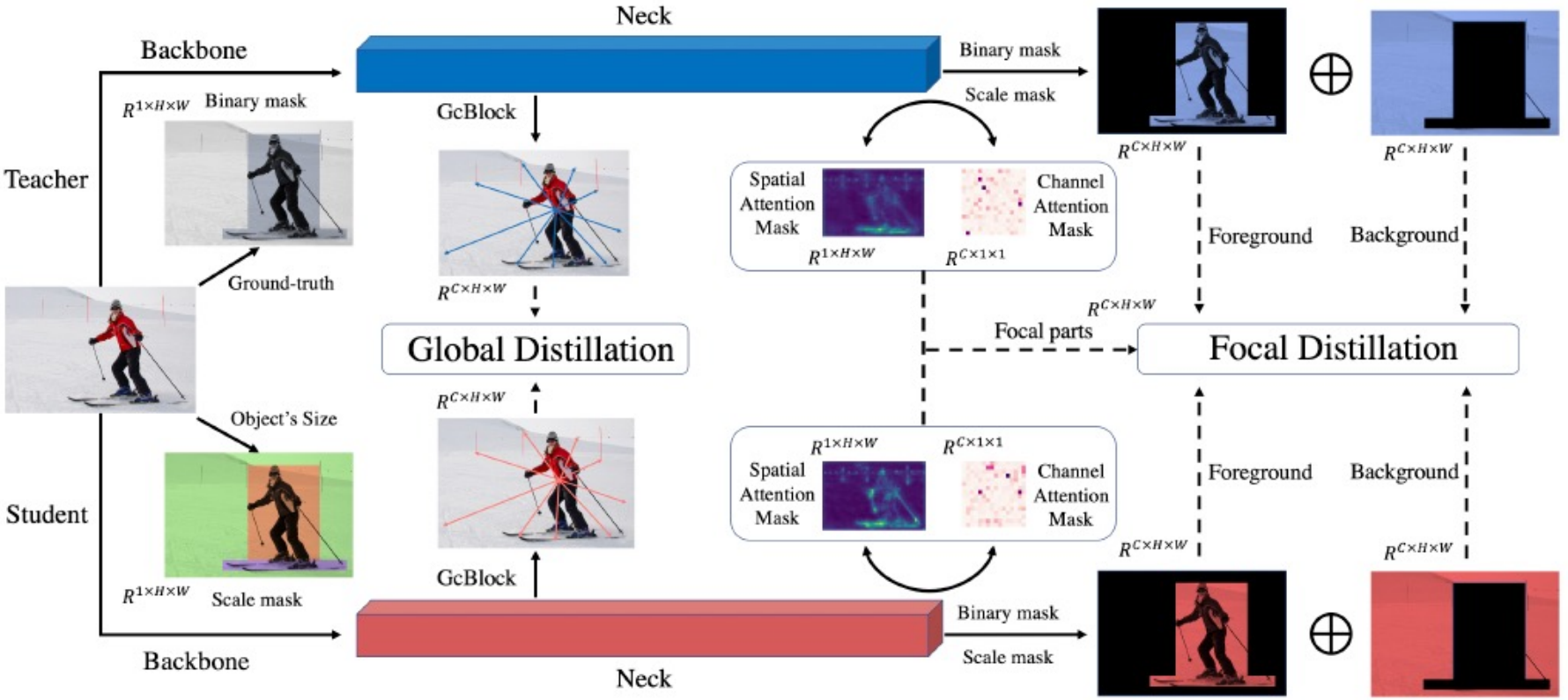

Unsupervised
Pre-train
Maximize
Similarity
z_i
z_j
Dense
ReLU
Dense
Dense
ReLU
Dense
$g(.)$
Projection head
h_i
h_j
$f(.)$
ResNet-
152(3×+SK)
Encoder
Encoder
Transformed
Images
x_i
x_j
Data
Augmentation
Original Image
Supervised
Fine-tune
Fine-tune
Dog
(0.92, 0.05, 0.03)
FC
Encoder
Distillation Using
Unlabeled Data
As close as
possible
Dog
(0.92, 0.05, 0.03)
Dog
(0.56, 0.21, 0.23)
FC
Encoder
Student

KD FOR OBJECT DETECTION



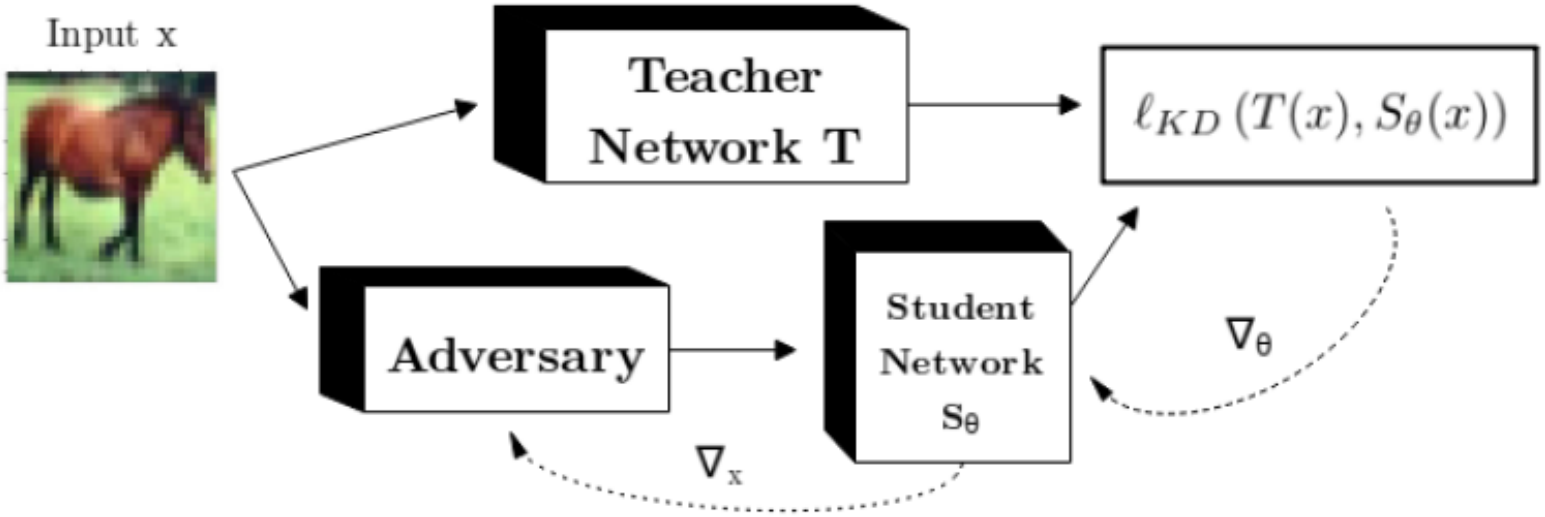
Visualization of the spatial and channel attention map from the teacher detector (RetinaNet-ResNeXt101) and the student detector (RetinaNet-ResNet50).

FGD



Focal and Global Knowledge Distillation for Detectors (CVPR'22)

ADVERSARIALLY ROBUST DISTILLATION (AAAI'20)



Model	Robust Accuracy ($\mathcal{A}_{adv}$)
AT ResNet18 teacher	44.46%
AT ResNet18 $\rightarrow$ MobileNetV2	38.21%
AT ResNet18 $\xrightarrow{\text{ARD}}$ MobileNetV2	**50.22%**

https://arxiv.org/abs/1905.09747

NEW WORKS

FOUR WORKS

- Deep Compression-based
- AdderNet
- Knowledge Distillation (2)

FOUR WORKS

- **Deep Compression-based**
- AdderNet
- Knowledge Distillation (2)

ON MINIMIZING DIAGONAL BLOCK-WISE DIFFERENCES FOR NEURAL NETWORK COMPRESSION

24th European Conference on Artificial Intelligence (ECAI 2020)

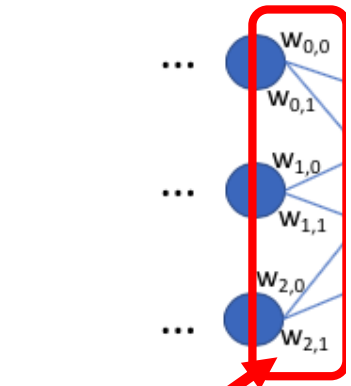
Yun-Jui Hsu, Yi-Ting Chang, Chih-Ya Shen,

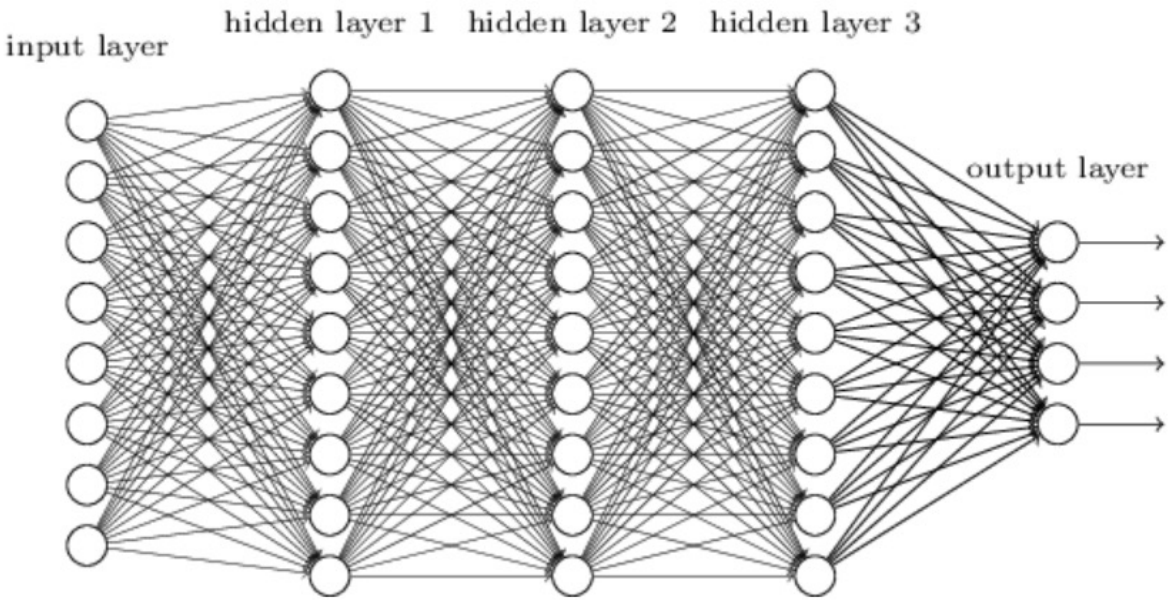
Hong-Han Shuai, Wei Lun Tseng, Chen Hsu Yang

https://ecai2020.eu/papers/11_paper.pdf

DEEP NEURAL NETWORKS

- Fully co
  matrices



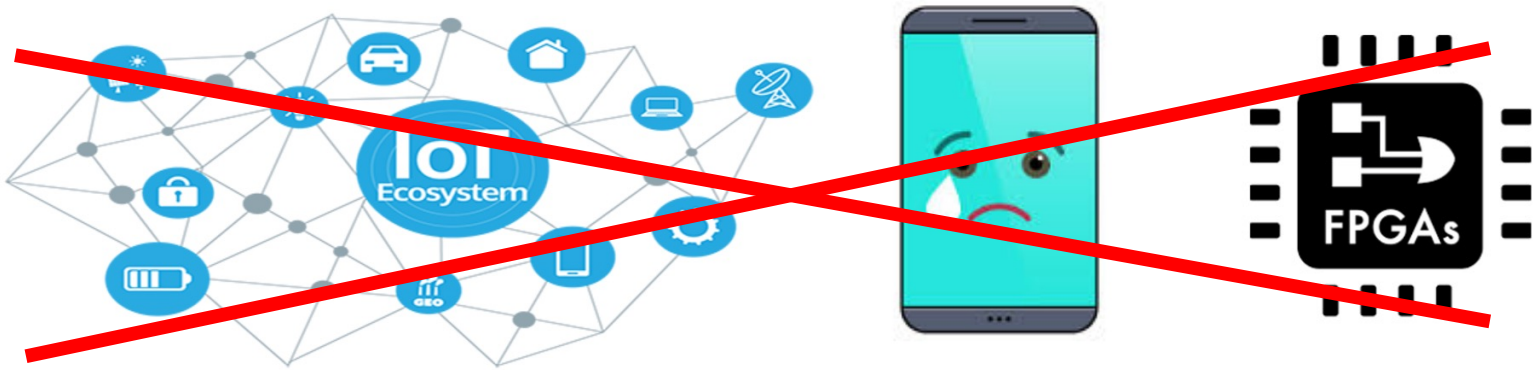


weights = $\begin{bmatrix} w_{(} & \\ w_{1,0} & w_{1,1} \\ w_{2,0} & w_{2,1} \end{bmatrix}$

In practice, we have much deeper neural networks than above

Deep Neural Networks

ResNet-152: **240 MB**
AlexNet: **248 MB**
VGG16 on ImageNet: **552 MB**



BACKGROUNDS

- **Quantization, Pruning,** and **Huffman Coding**
- **Deep Compression**: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding (DeepC)
- **Matrix Permutation Decomposition Algorithm for Deep Neural Network Compression** (MPDC)

PRUNING

- Keep only important weights after training
  - **Q:** How to determine the **importance** of each weight?
  - **A:** Usually, the greater the absolute value of weight, the more importance

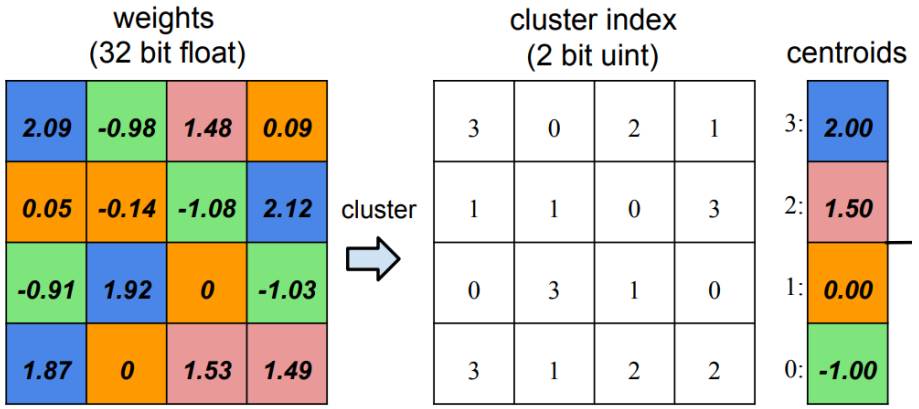
- **Pruning rate**: the ratio of weights we keep

0.3	0.6	-0.1
-0.5	0.2	1.0
-0.7	0.4	-0.9

If the pruning rate is 66.7%, we remove 33.3% of the weights
The weights in red are pruned

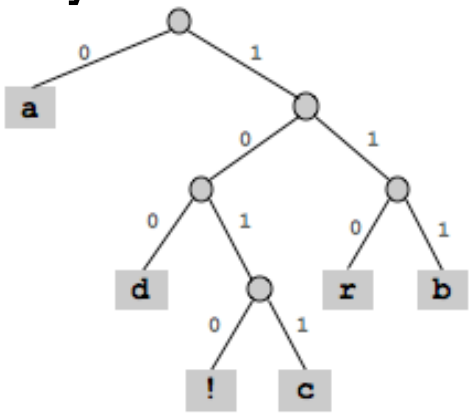
QUANTIZE

- Use fewer bits to represent all weights (with some loss of accuracy)
  1. Use clustering algorithms to cluster weights to 2^{bits} categories The example below uses 2 bits
  2. All weights of the same category are replaced by centroid of that category



HUFFMAN CODING

- For lossless data compression
- Shorter encoding for higher frequency terms; longer encoding for lower frequency terms
- E.g., frequency: a>d>r>b>!>c



char	encoding
a	0
b	111
c	1011
d	100
r	110
!	1010

MPDCompression

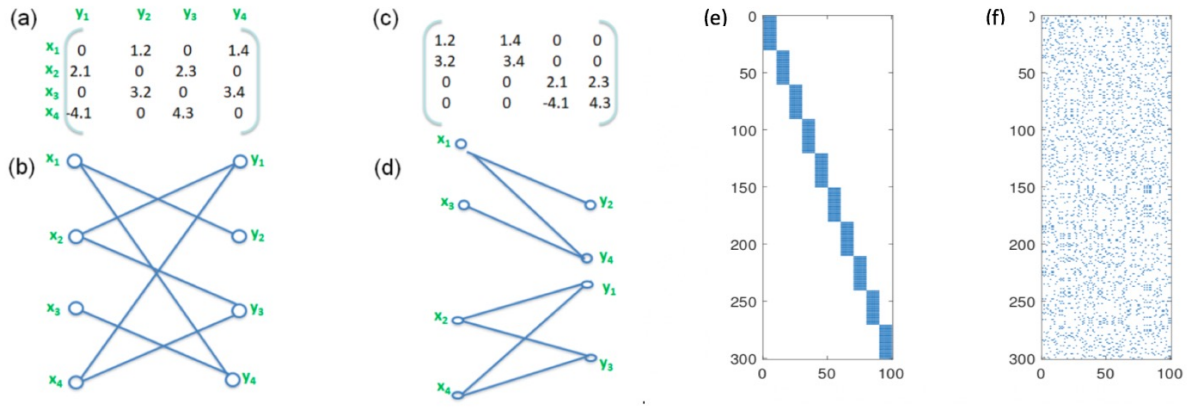


Figure 1: (a) Example of a 4×4 sparse matrix with irregular structure; (b) graph representation of the 4×4 matrix of Fig. 1(a); (c) block-diagonalized version of the 4×4 matrix of Fig. 1(a), achieved by row and column permutations; (d) graph representation of the block-diagonalized matrix of Fig. 1(c). (e) 300×100 block-diagonal matrix, $\mathbf{B}_1$; (f, right) 300×100 binary mask, $\mathbf{M}_1$ created by randomly permuting the rows and columns of $\mathbf{B}_1$.

- Mainly for speed-up
- Needs to compute the inverse of matrix (f) during inference mode (difference from MESA)

PROBLEMS WE OBSERVED

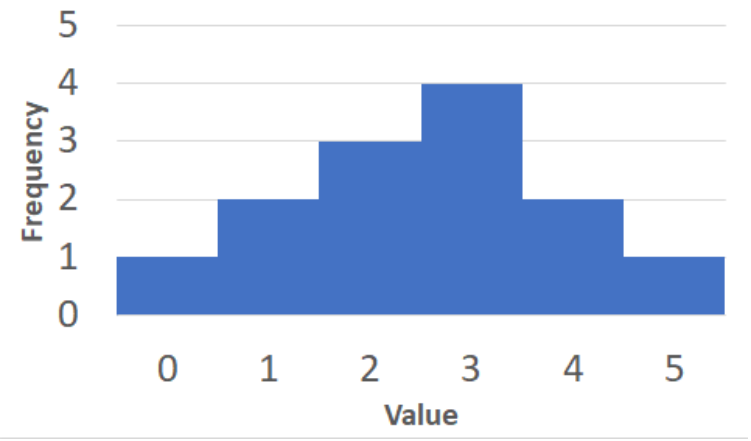
P1. Values scattered in the weight matrix

- Requiring additional indices for sparse matrix representation

P2. Various values (using Huffman Coding alone)

- Requiring longer bits to encode

0			1		
2			4		3
	5			3	
		3			
	1			2	3
		2		4	

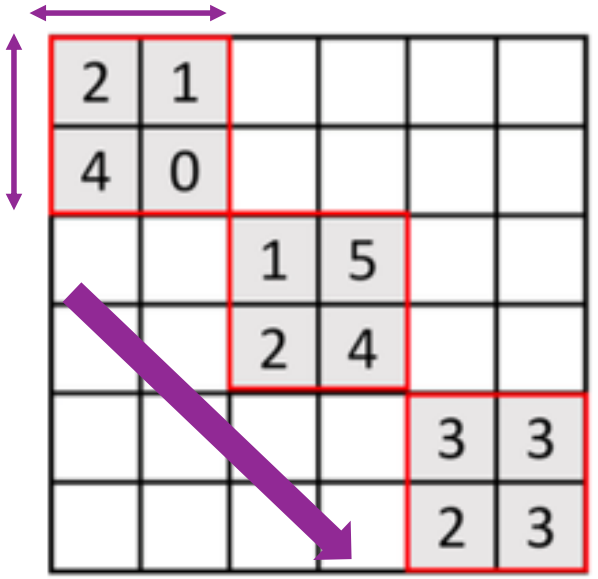


OUR IDEA TO TACKLE P1

P1. Values scattered in the weight matrix
- Requiring additional indices for sparse matrix representation

P2. Various values (using Huffman Coding alone)
- Requiring longer bits to encode

- Block-Diagonal Structure
  - Each block locates on the diagonal
  - Known block size
    -> block location can be calculated easily
    -> reduce the burden of storing indices

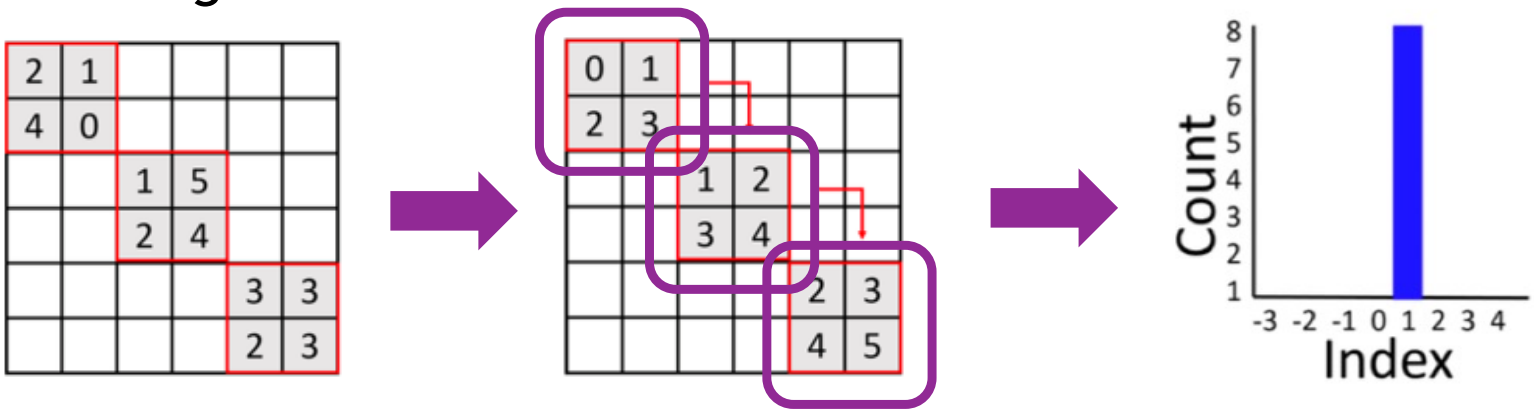


OUR IDEA TO TACKLE P2

P1. Values scattered in the weight matrix
- Requiring additional indices for sparse matrix representation

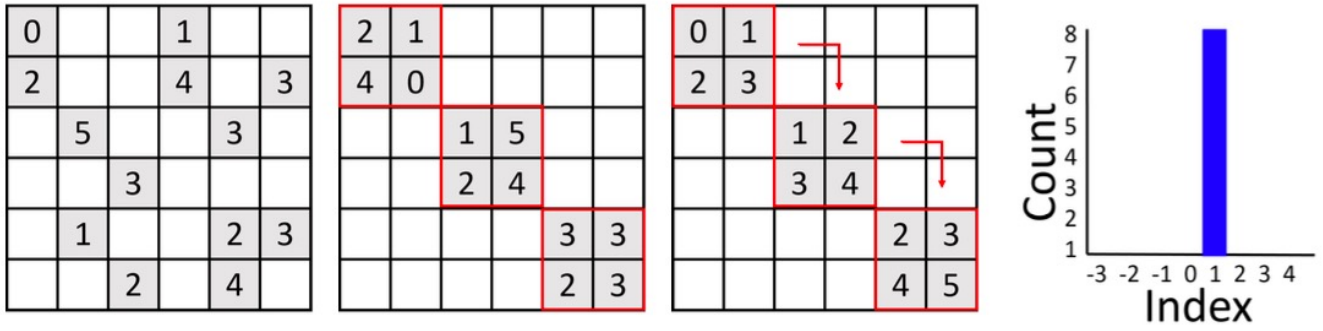
P2. Various values (using Huffman Coding alone)
- Requiring longer bits to encode

- Train to make blocks similar
- Employ **Delta Coding**
  - First block [0, 1, 2, 3]
  - Second block [1-0, 2-1, 3-2, 4-3] = [1, 1, 1, 1]
  - Third block [2-1, 3-2, 4-3, 5-4] = [1, 1, 1, 1]
- Then use Huffman Coding to encode – bits shortened for encoding



OVERVIEW AND REVIEW OF OUR IDEA

- Compress NN by converting the weight matrix to **block diagonal structure**
- Minimize the pair-wise differences
- **Delta Coding + Huffman Coding**



(a) Irregular structure (b) Block diagonal structure (c) Preferred structure (d) Distribution for Fig. 1(c)

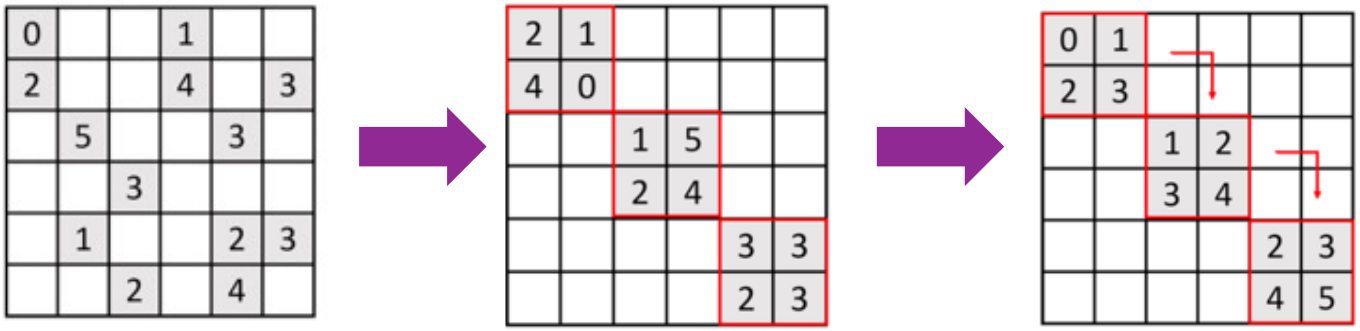
ACHIEVEMENTS

- Achieve 135× to 392× compression rates
  - **Tripling** the state-of-the-art approaches (Deep Compression)

- Inference speed-up 2.6× to 5.1×
  - Up to 44% to the state-of-the-art (Deep Compression)

Question

- *If the weight matrices of fully connected layers is pruned to block diagonal structure, will it affect the accuracy?*
  - Not much
  - MPDCompress shows that pruning the weights arbitrarily (regardless the weight importance) retains nearly the same performance of the model

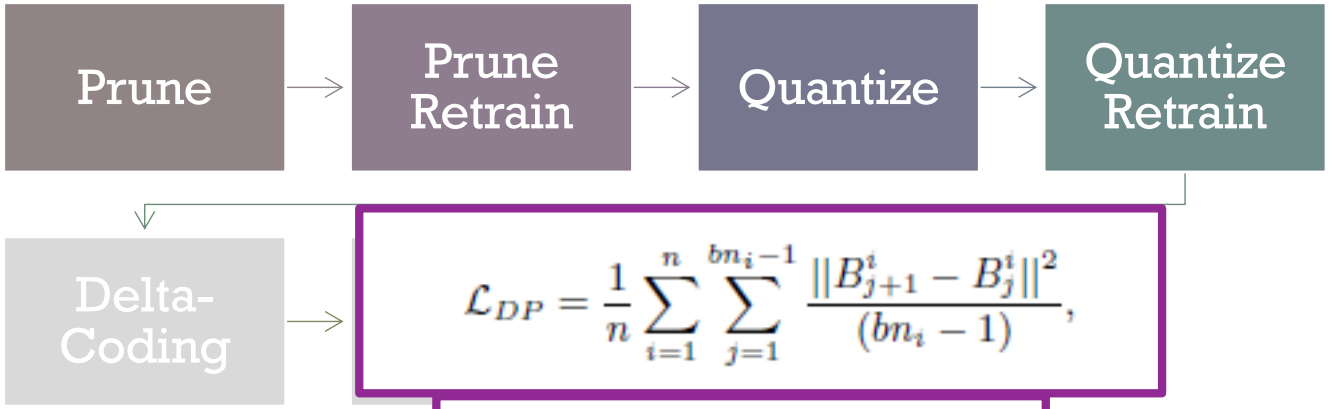


Question

- *If the weight matrices of fully connected layers is pruned to block diagonal structure, will it affect the accuracy?*
  - Validation: we employed DeepCompression to conduct a preliminary experiment (with fixed pruning rate, quantization bits)

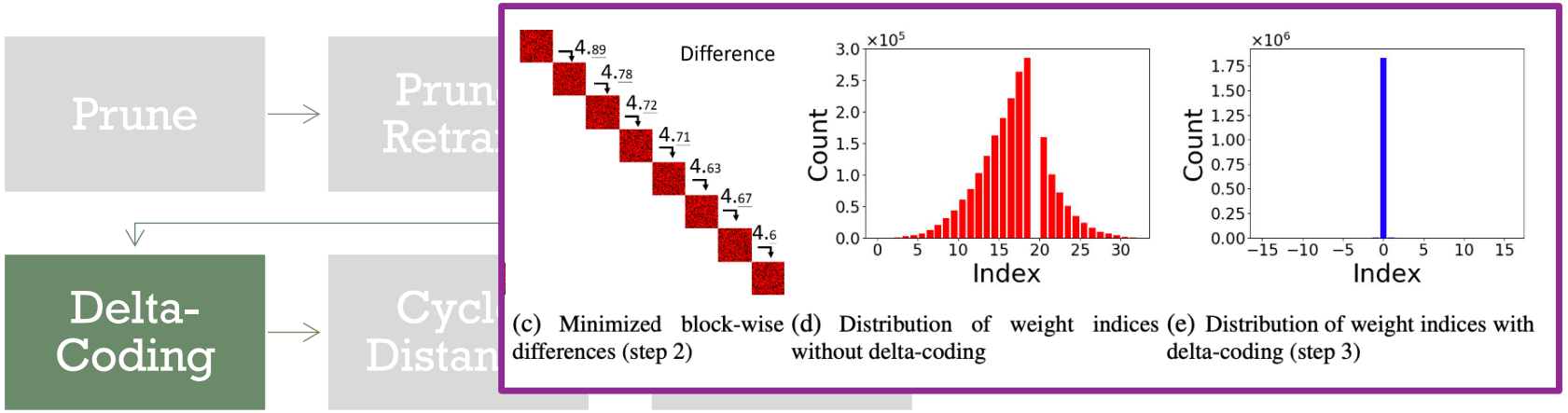
	Pruning rates	DeepC	Arb	Diag
LeNet-5	10%, 20%	99.35	99.03	99.11
VGG16 CIFAR-10	12.5%, 12.5%, 10%	91.38	91.1	91.44
VGG16 CIFAR-100	12.5%, 12.5%, 10%	72.33	72.35	72.12
AlexNet	10%, 10%, 25%	53.23	54.16	53.37

OUR APPROACH: MESA – OVERVIEW



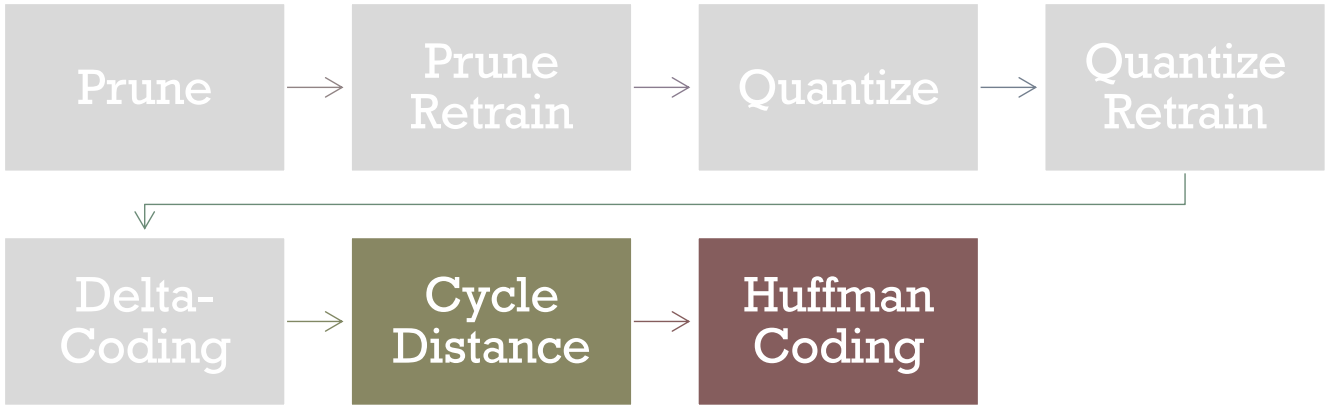
- **Prune:** prune fully … **liagonal structure**
- **Prune-Retrain:** add $\mathcal{L}_{DP}$ loss to enhance the performance of Huffman coding (with Delta coding)
- **Quantize, Quantize Retrain:** same as *Deep Compression*

MESA – OVERVIEW



- **Delta-Coding:** Because $\mathcal{L}_{DP}$ loss can learn to make the weights in the same relative position on the block be quantized into the same category, after using delta-coding, they can be concentrated at 0, which is beneficial to Huffman coding

MESA – OVERVIEW



- **Cycle-Distance:** For example, given 3-bit quantized blocks, [[0, 1], [7, 1]] and [[7, 1], [0, 1]], the range of number difference is [-7, 7], so we need 4 bits to represent these number differences. However, if we use cyclic-distance, only 3 bits are needed
- **Huffman Coding:** same as introduced earlier

MESA – OVERVIEW

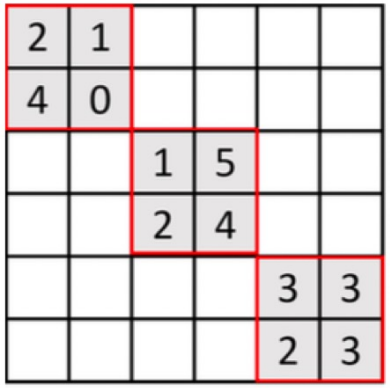
- **Step 1:** Pruning with Block Diagonal Mask (PM)
- **Step 2:** Difference Minimization for Neighboring Blocks (DM)
- **Step 3:** Post-Quantization Cyclic-distance Assignment (PQA)
- **Step 4:** Huffman Encoding with Organized Distribution (HE)

MESA – STEP 1

Algorithm 1 Pruning with Block Diagonal Mask (PM)

Input: Fully-connected weight matrix: W with size $(x \times y)$, pruning rate p
Output: Pruned fully-connected layer: W^M

1: Generate M^B by the size of W, with $\lfloor \frac{1}{p} \rfloor$ dense blocks assigned as binary True along the diagonal axis having the same size $\lfloor x \cdot p \rfloor \times \lfloor y \cdot p \rfloor$
2: $W^M \leftarrow W$ AND M^B
3: block-number: $bn \leftarrow \lfloor \frac{1}{p} \rfloor$
4: **return** W^M, bn



MESA – STEP 2

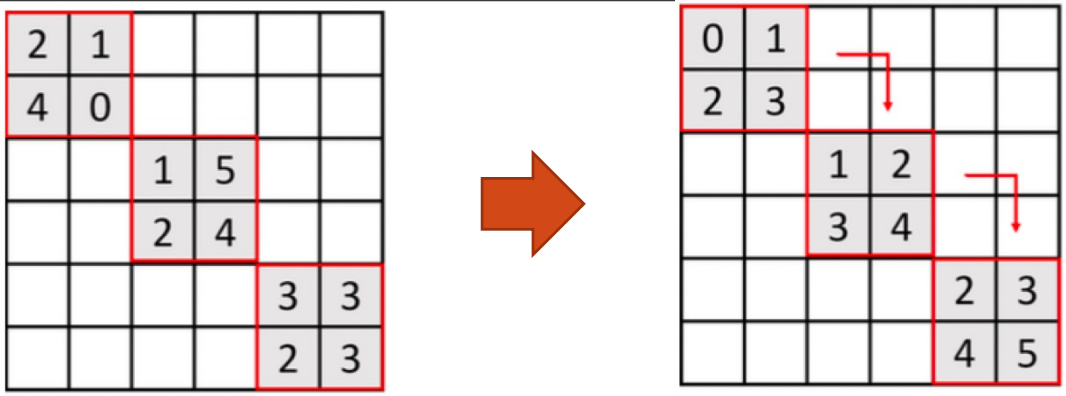
$$\mathcal{L}_{DP} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{bn_i-1}\frac{||B^i_{j+1} - B^i_j||^2}{(bn_i - 1)}$$

$$\mathcal{L}_{total} = \mathcal{L}_{acc} + \alpha\mathcal{L}_{DP}$$

Algorithm 2 Difference Minimization for Neighboring Blocks (DM)

Input: Masked weights for each layer: $W = \{W_1^M, \cdots, W_n^M\}$, block number for each layer: $BN = \{bn_1, \cdots, bn_n\}$
Output: Loss of Distance Penalty: ($\mathcal{L}_{\mathcal{DP}}$)

$i \leftarrow 1, j \leftarrow 1, \mathcal{L}_{\mathcal{DP}} \leftarrow 0$
for $i \leq n$ **do**
  for $j < bn_i$ **do**
    $\mathcal{L}_{\mathcal{DP}} \leftarrow \mathcal{L}_{\mathcal{DP}} + \frac{||B^i_{j+1} - B^i_j||^2}{b_{n_i} - 1}$
    $j \leftarrow j + 1$
return $\mathcal{L}_{\mathcal{DP}}$



MESA – STEP 3

- Quantize the weight matrix of each fully-connected layer

Algorithm 3 Post-Quantization Cyclic-distance Assignment (PQA)

Input: Quantized diagonal blocks: B^*, B, maximum weight index range: r
Output: Delta-encoded form of B^*: B^R

1: B^R= size(B^*)
2: **for** i, j in size of B^* **do**
3: $\quad B^R_{i,j} \leftarrow \min\{|B^*_{i,j} - B_{i,j}|, r - |B^*_{i,j} - B_{i,j}|\}$
4: **return** B^R

MESA – ALL STEPS

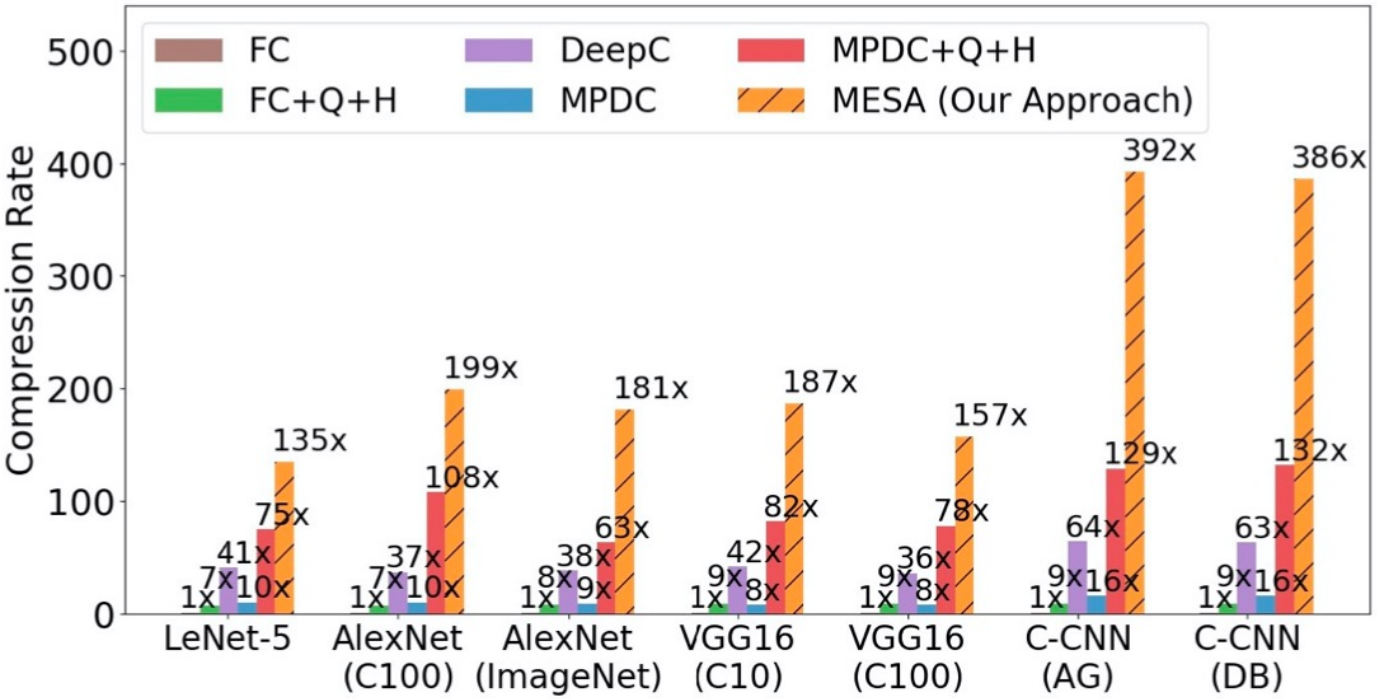
Algorithm 4 Memory-Efficient and Structure-Aware Compression (MESA)

Input: $n, W = \{W_1, \cdots, W_n\}, \mathbb{P} = \{p_1, \cdots, p_n\}, \mathbb{Q} = \{q_1, \cdots, q_n\}$
Output: Compressed NN weights after MESA is applied: W^C

for each NN layer W_i **do**
    W_i^M, $bn_i \leftarrow$ **PM**(W_i, p_i), initialize pruned weights W_i^M
For each layer, $BN \leftarrow$ all bn_i; $W^M \leftarrow$ all W_i^M
for each training epoch **do**
    Input batch data and calculate the MSE
    $Loss \leftarrow MSE$; $Loss \leftarrow Loss +$ **DM**(W^M,BN)
    Compute gradient, back propagation
for each Converged weight: W_i^M **do**
    $W_i^M \leftarrow$ Quantized W_i^M from 32 bits to q_i bits
for each Quantized weight matrix: W_i^M **do**
    range $\leftarrow 2_i^q$
    for j from 1 to $(bn_i - 1)$ **do**
        $B_{j+1}^i \leftarrow$ **PQA**(B_{j+1}^i, B_j^i, range)
    Compressed weight $W_i^C \leftarrow$ Huffman code B_1^i and $\{B_1^i, \cdots, B_m^i\}$ respectively from W_i^M
return Compressed NN weights W^C

Experiments – Compression Rate

Abbrev.	Full Name
FC	Fully Connected (no pruning)
FC+Q+H	FC + Quantization + Huffman Coding
DeepC	DeepCompression
MPDC	MPDCompression
MPDC+Q+H	MPDC + Quantization + Huffman Coding
MESA	Our approach



- The pruning rates for each models are same (exclude FC)
- All quantized to 5 bits (exclude FC, MPDC)
- Compressing the fully connected layers

EXPERIMENT – LAYER-WISE (LENET)

Table 3. Layer-wise average bits in LeNet-5

	Size	p	DeepC Avg. bits	M+Q+H Avg. bits	MESA Avg. bits	MESA CP Rate
fc1	1.52MB	10%	7.6	4.2	**2.3**	138×
fc2	19.53KB	10%	14.2	**6**	6.7	47×
total	1.54MB	10%	7.7	3.1	**2.3**	135×

- Average bits: the model size (in bits) divided by the total number of unpruned weights

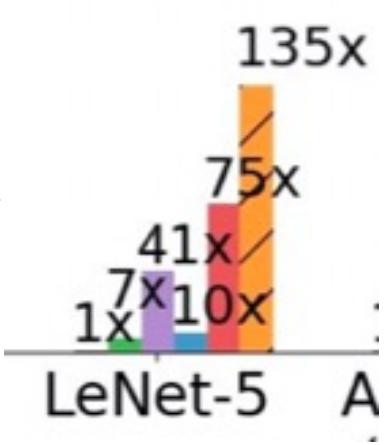


Table 2. Accuracy of different approaches

	FC	FC+Q+H	DeepC	MPDC	M+Q+H	*MESA*
LeNet-5	99.34	99.28	99.29	99.08	99.03	*99.04*
VGG16 (C10)	91.32	91.40	91.44	91.14	90.53	*91.14*
VGG16 (C100)	71.40	72.33	72.57	71.84	71.91	*71.46*
AlexNet	52.68	52.77	54.04	53.69	53.83	*53.47*
AlexNet (ImageNet)	77.12	76.51	77.58	76.38	76.27	*76.55*
C-CNN (AG)	86.61	86.68	86.76	86.51	87.22	*87.21*
C-CNN (DB)	97.84	97.90	97.94	97.92	98.05	*97.92*

EXPERIMENT – LAYER-WISE (C-CNN)

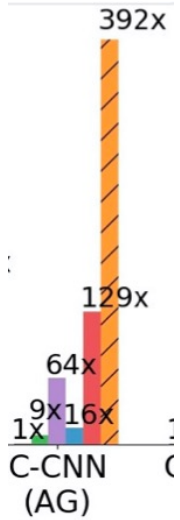


Table 4. Layer-wise average bits in C-CNN (AG)

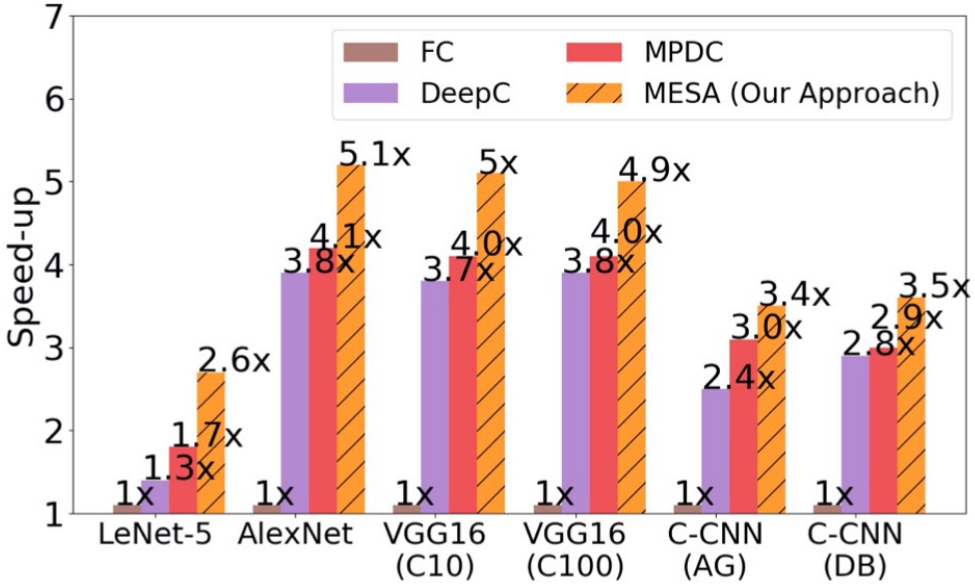
	Size	p	DeepC Avg. bits	M+Q+H Avg. bits	MESA Avg. bits	MESA CP Rate
fc1	34MB	6.25%	7.9	3.9	**1.3**	401×
fc2	4MB	6.25%	8.9	4.1	**1.5**	341×
fc3	16KB	25%	9.8	5.3	**4.2**	31×
total	38.01MB	6.28%	8	3.95	**1.3**	392×

- Average bits: the model size (in bits) divided by the total number of unpruned weights

Table 2. Accuracy of different approaches

	FC	FC+Q+H	DeepC	MPDC	M+Q+H	*MESA*
LeNet-5	99.34	99.28	99.29	99.08	99.03	*99.04*
VGG16 (C10)	91.32	91.40	91.44	91.14	90.53	*91.14*
VGG16 (C100)	71.40	72.33	72.57	71.84	71.91	*71.46*
AlexNet	52.68	52.77	54.04	53.69	53.83	*53.47*
AlexNet (ImageNet)	77.12	76.51	77.58	76.38	76.27	*76.55*
C-CNN (AG)	86.61	86.68	86.76	86.51	87.22	*87.21*
C-CNN (DB)	97.84	97.90	97.94	97.92	98.05	*97.92*

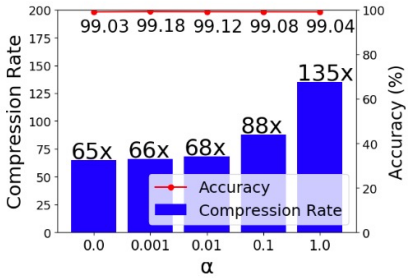
Experiment – Inference Speed-up



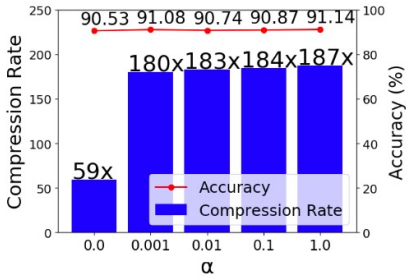
- Speed-up: the ratio of the inference time of the original uncompressed model (FC) to the inference time of the model compressed by a certain approach

EXPERIMENT – PENALTY TERM (α)

$$\mathcal{L}_{total} = \mathcal{L}_{acc} + \alpha \mathcal{L}_{DP}$$



(a) LeNet-5



(b) VGG16 (C10)

	LeNete-5				VGG16 (C10)			
	Orig. index	Delta-coded index			Orig. index	Delta-coded index		
		α=0.01	α=0.1	α=1.0		α=0.001	α=0.01	α=1.0
fc1	4.86	4.58	3.39	**2.00**	2.94	0.2	**0.07**	0.1
fc2	4.97	4.91	4.85	**3.42**	2.81	0.08	**0.03**	0.08
fc3	N/A	N/A	N/A	N/A	3.35	2.22	0.95	**0.21**

CONCLUSION

- MESA effectively prunes the weights into a block diagonal structure and minimizes the block-wise differences to boost the compression rate

- MESA achieves up to 392× compression rates, tripling those of the state-of-the-art approaches, while significantly improving the inference efficiency

FOUR WORKS

- Deep Compression-based
- **AdderNet**
- Knowledge Distillation

ADDERNET: DO WE REALLY NEED MULTIPLICATIONS IN DEEP LEARNING ? CVPR'20

- GPU is very fast and can accelerate the processing of a large number of multiplication operations in CNN.
- However, the GPU consumes too much energy. Also, mobile devices sometimes do not contain GPU.

MULTIPLICATION VS. ADD

- CNN

$$Y(m,n,t) = \sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=0}^{c_{in}} S\big(X(m+i,n+j,k), F(i,j,k,t)\big)$$

- AdderNet

$$Y(m,n,t) = -\sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=0}^{c_{in}} |X(m+i,n+j,k) - F(i,j,k,t)|.$$

EXPERIMENTAL RESULTS

Table 2. Classification results on the CIFAR-10 and CIFAR-100 datasets.

Model	Method	#Mul.	#Add.	XNOR	CIFAR-10	CIFAR-100
VGG-small	BNN	0	0.65G	0.65G	89.80%	65.41%
	AddNN	0	1.30G	0	93.72%	72.64%
	CNN	0.65G	0.65G	0	93.80%	72.73%
ResNet-20	BNN	0	41.17M	41.17M	84.87%	54.14%
	AddNN	0	82.34M	0	91.84%	67.60%
	CNN	41.17M	41.17M	0	92.25%	68.14%
ResNet-32	BNN	0	69.12M	69.12M	86.74%	56.21%
	AddNN	0	138.24M	0	93.01%	69.02%
	CNN	69.12M	69.12M	0	93.29%	69.74%

https://www.agner.org/optimize/instruction_tables.pdf

FOUR WORKS

- Deep Compression-based
- AdderNet
- **Knowledge Distillation (2)**

Weighted Distillation with Unlabeled Examples

GOOGLE RESEARCH

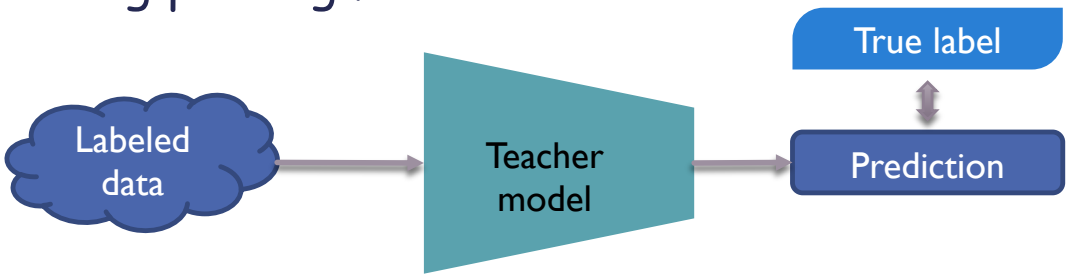
NIPS 2022

Slides credit: 羅婈

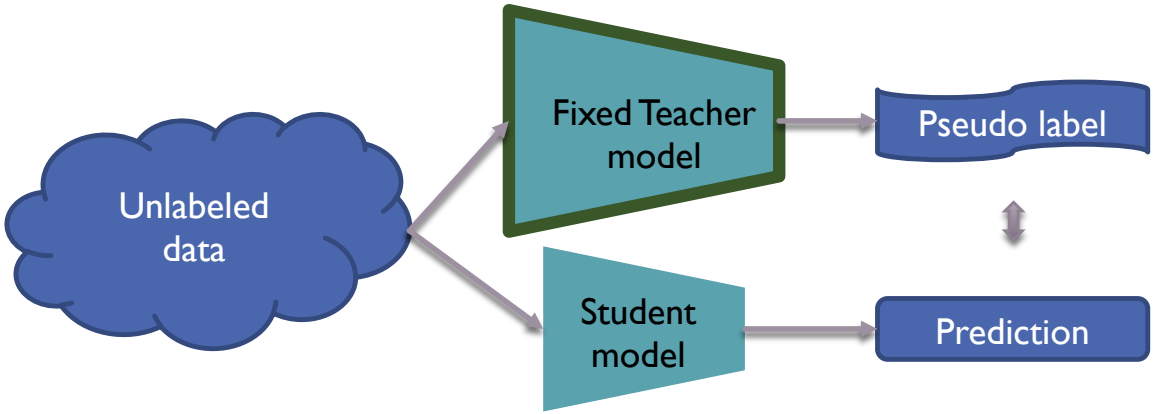
Distillation with Unlabeled Examples

- Two-stage training paradigm

Stage 1

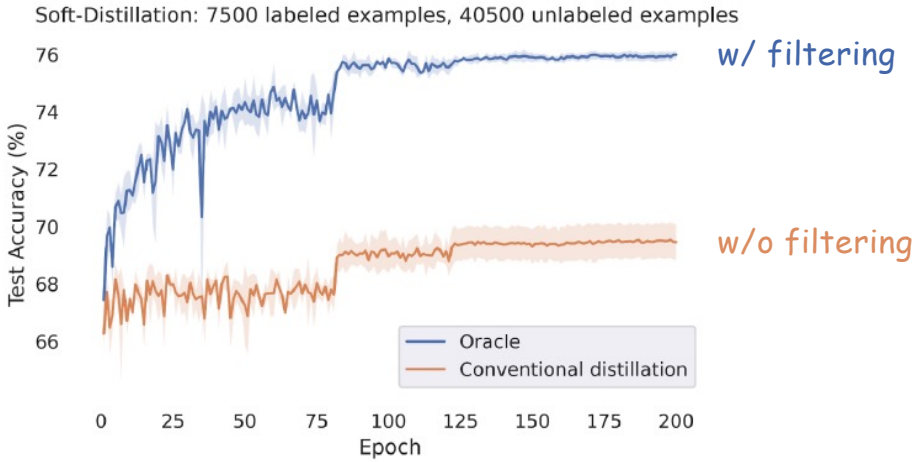


Stage 2



Distillation with Unlabeled Examples

- Issue – the quality of teacher label
  - Filtering teacher's noisy samples is beneficial for student model.

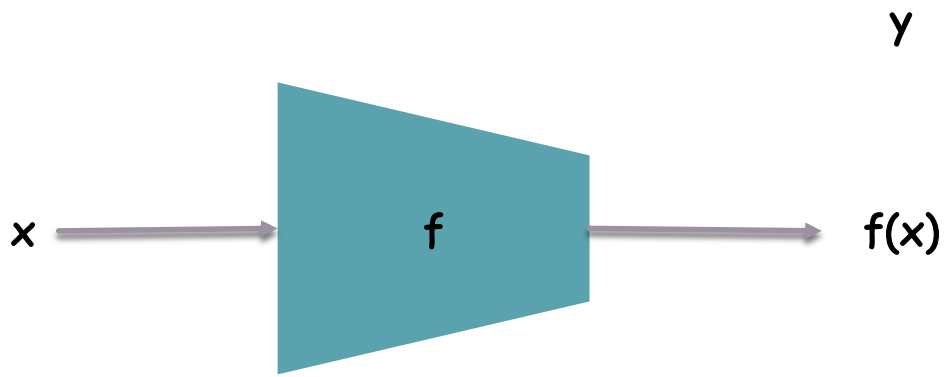


-> Noisy samples are NOT helpful during training.

-> Reweight the loss function for student training

Method

- Classification model

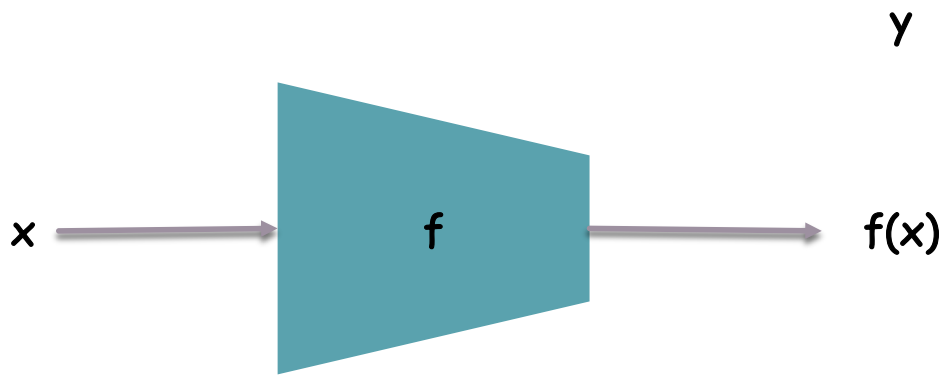


Empirical risk:

$$R(f) = \mathbb{E}[\ell(y, f(x))]$$

Method

- Classification model

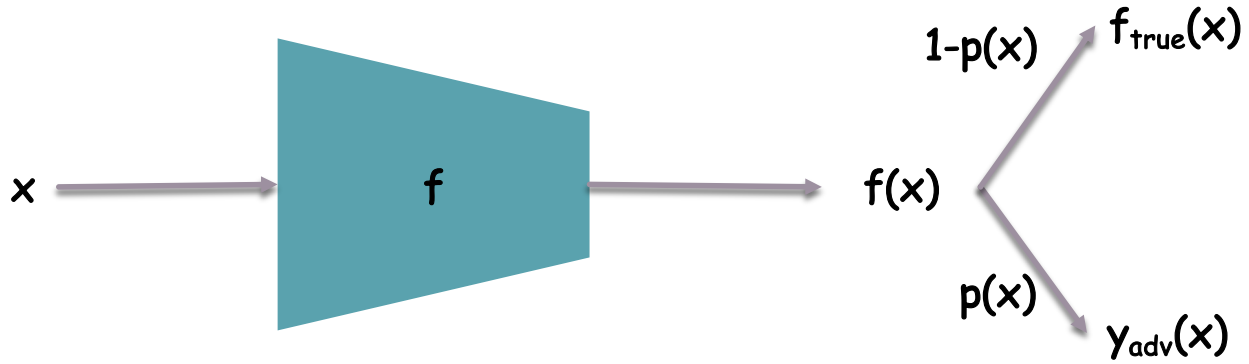


Empirical risk:

$$R_S(f) = \frac{1}{|S|} \sum_{(x,y)\in S} \ell(y, f(x)).$$

Method

- Debiasing weights
  - Noise modeling for teacher



Empirical risk:

$$R(f) = \mathbb{E}[\ell(y, f(x))]$$

$$R(f) = \mathbb{E}_{x \sim \mathbb{X}}[\ell(f_{\text{true}}(x), f(x))]$$

Method

- Debiasing weights
  - Noise modeling for teacher

$$
\begin{aligned}
\mathbb{E}[R_S(f)] &= \mathbb{E}_{S\sim\mathbb{D}^n}\left[\frac{1}{n}\sum_{i=1}^{n}\ell(y_i, f(x_i))\right] \\
&= \mathbb{E}_{(x,y)\sim\mathbb{D}}\left[\ell(y, f(x))\right] \\
&= \mathbb{E}_{x\sim\mathbb{X}}[\mathbb{E}_{y|x}[\ell(y, f(x))]] \\
&= \mathbb{E}_{x\sim\mathbb{X}}[p(x)\ell(y_{\text{adv}}(x), f(x)) + (1-p(x))\ell(f_{\text{true}}(x), f(x))] \\
&= \mathbb{E}_{x\sim\mathbb{X}}[\ell(f_{\text{true}}(x), f(x))] + \mathbb{E}_{x\sim\mathbb{X}}[p(x)\cdot(\ell(y_{\text{adv}}(x), f(x)) - \ell(f_{\text{true}}, f(x)))] \\
&= \mathbb{E}_{x\sim\mathbb{X}}[\ell(f_{\text{true}}(x), f(x))] + \mathbb{E}_{x\sim\mathbb{X}}\left[p(x)\cdot\left(\frac{\ell(y_{\text{adv}}(x), f(x))}{\ell(f_{\text{true}}(x), f(x))} - 1\right)\cdot\ell(f_{\text{true}}(x), f(x))\right]
\end{aligned}
$$

Unbiased risk — Bias(f)

Method

- Debiasing weights
  - Reweighting the loss

Original: $$R_S(f) = \frac{1}{|S|} \sum_{(x,y)\in S} \ell(y, f(x)).$$

Reweighted: $$R_S^w(f) = \frac{1}{|S|} \sum_{(x,y)\in S} w_f(x)\ell(y, f(x))$$

Method

- Debiasing weights
  - Reweighting the loss

$$w_f(x) = \frac{1}{1 + p(x)\,(\text{distortion}_f(x) - 1)}$$

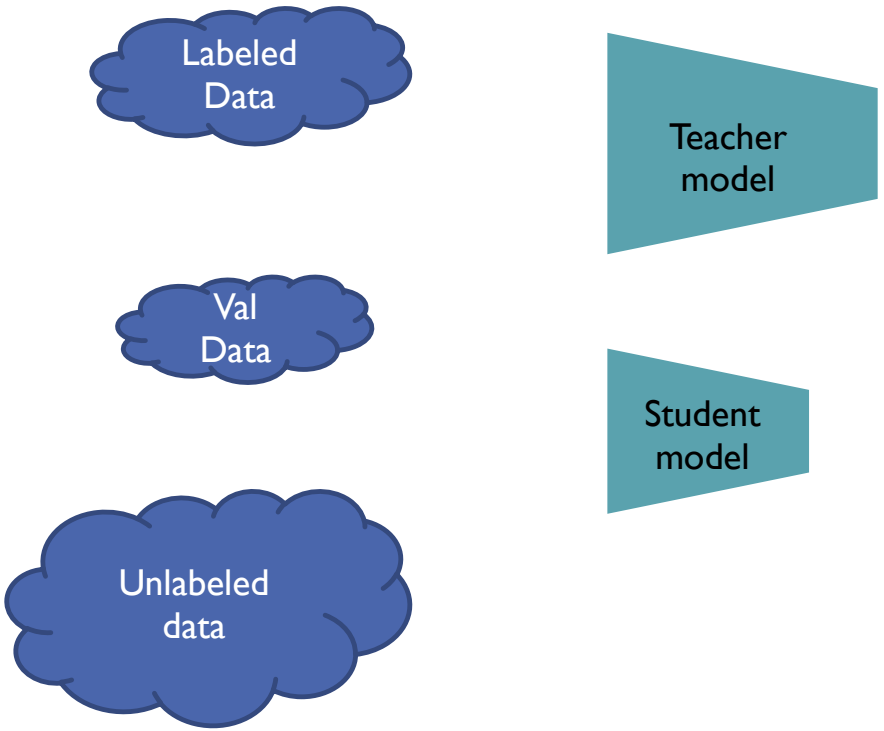
Probability of the bias corrupting the label

the distortion caused to the loss by corruption

$$\frac{\ell(\text{Teacher}(x), \text{Student}(x))}{\ell(y, \text{Student}(x))}$$

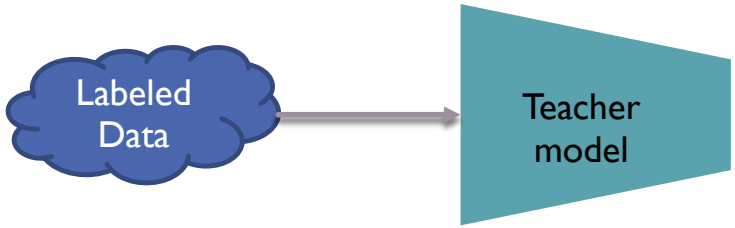
Method

- Training process



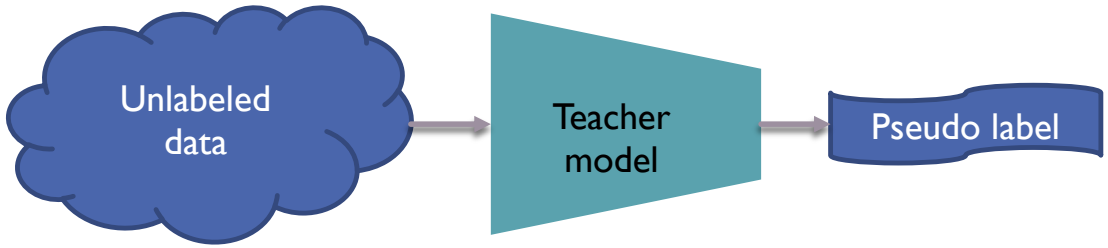
Method

- Training process
  - Training the teacher



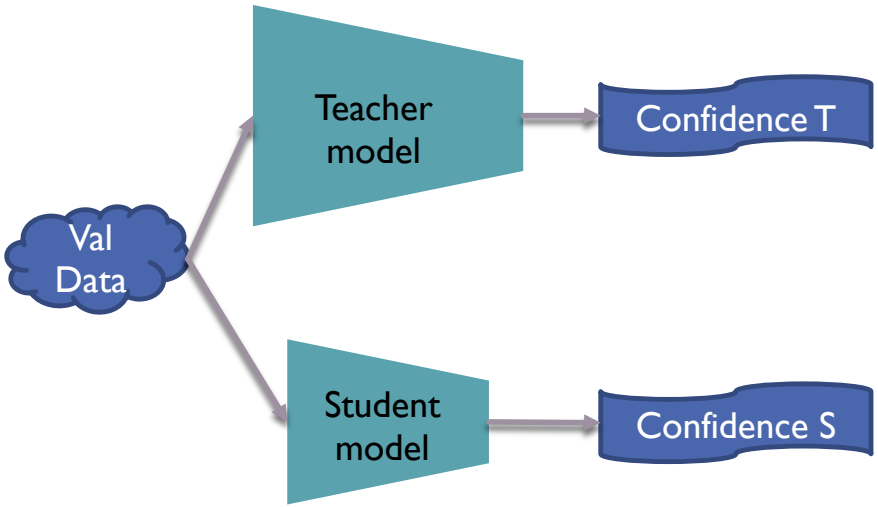
Method

- Training process
  - Generating pseudo label



Method

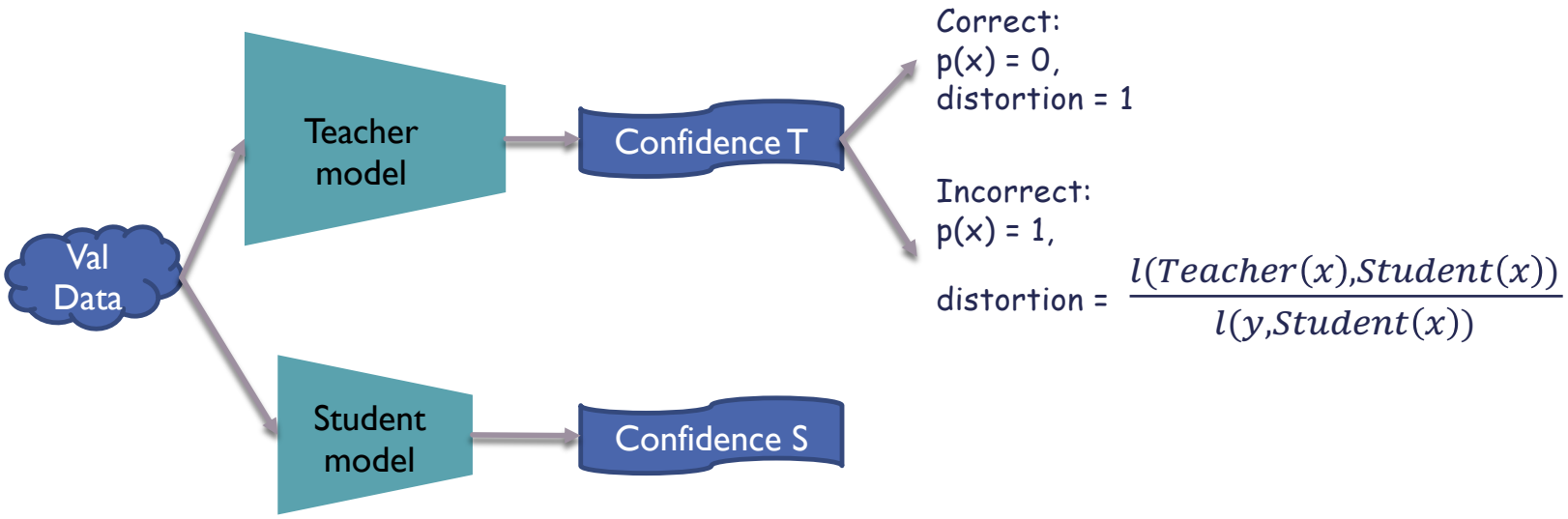
- Training process
  - Calculating p(x) and distortion of validation samples



Method

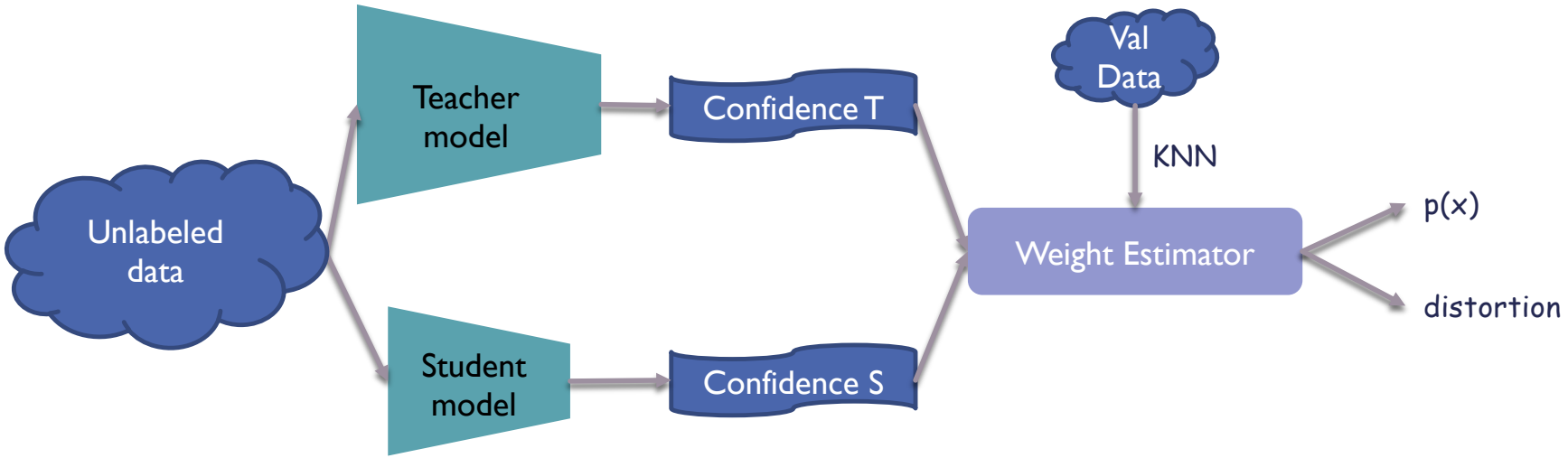
- Training process
  - Calculating p(x) and distortion of validation samples

$$w_f(x) = \frac{1}{1 + p(x)\,(\text{distortion}_f(x) - 1)}$$



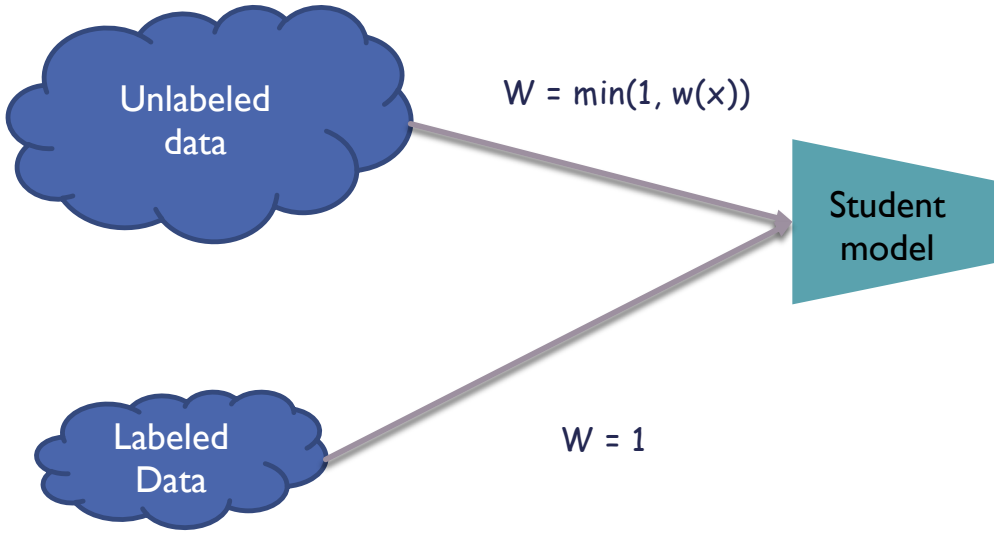
Method

- Training process
  - Prediction weights of unlabeled samples from validation samples



Method

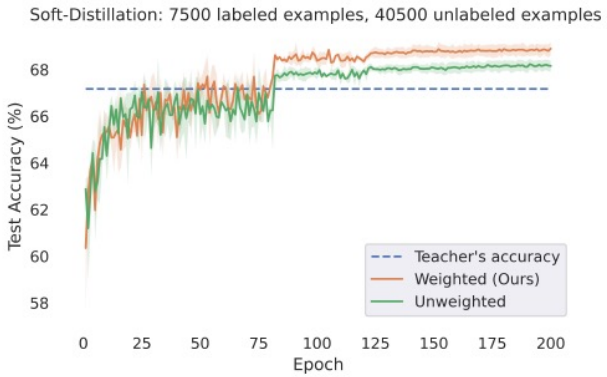
- Training process
  - Training the student with weighted loss



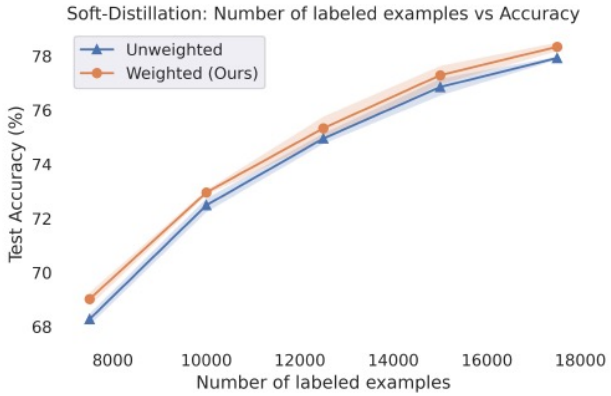
※ Clipping w for stable training:
-> w > 1 -> distortion < 1
-> similar predictions from both models but inaccurate

Experiments

- Converge Time on CIFAR-10



- Accuracy on CIFAR-10



Experiments

- Combining uncertainty-based weighting schemes
  - Accuracy

Labeled Examples	7500	10000	12500	15000	17500
Teacher (soft)	67.55%	72.85%	74.85%	77.63%	78.40%
Our Method	70.59 ± 0.05%	74.59 ± 0.07%	75.75 ± 0.32%	78.56 ± 0.14%	79.21 ± 0.17%
Fidelity-based weighting [12]	69.31 ± 0.41%	73.39 ± 0.44%	74.69 ± 0.31%	77.10 ± 0.16%	78.19 ± 0.19%
Composition	**71.20 ± 0.097%**	**75.54 ± 0.21%**	**76.80 ± 0.29%**	**79.55 ± 0.22%**	**80.28 ± 0.28%**

KNOWLEDGE DISTILLATION: A GOOD TEACHER IS PATIENT AND CONSISTENT

CVPR 2022

Various design choices for KD

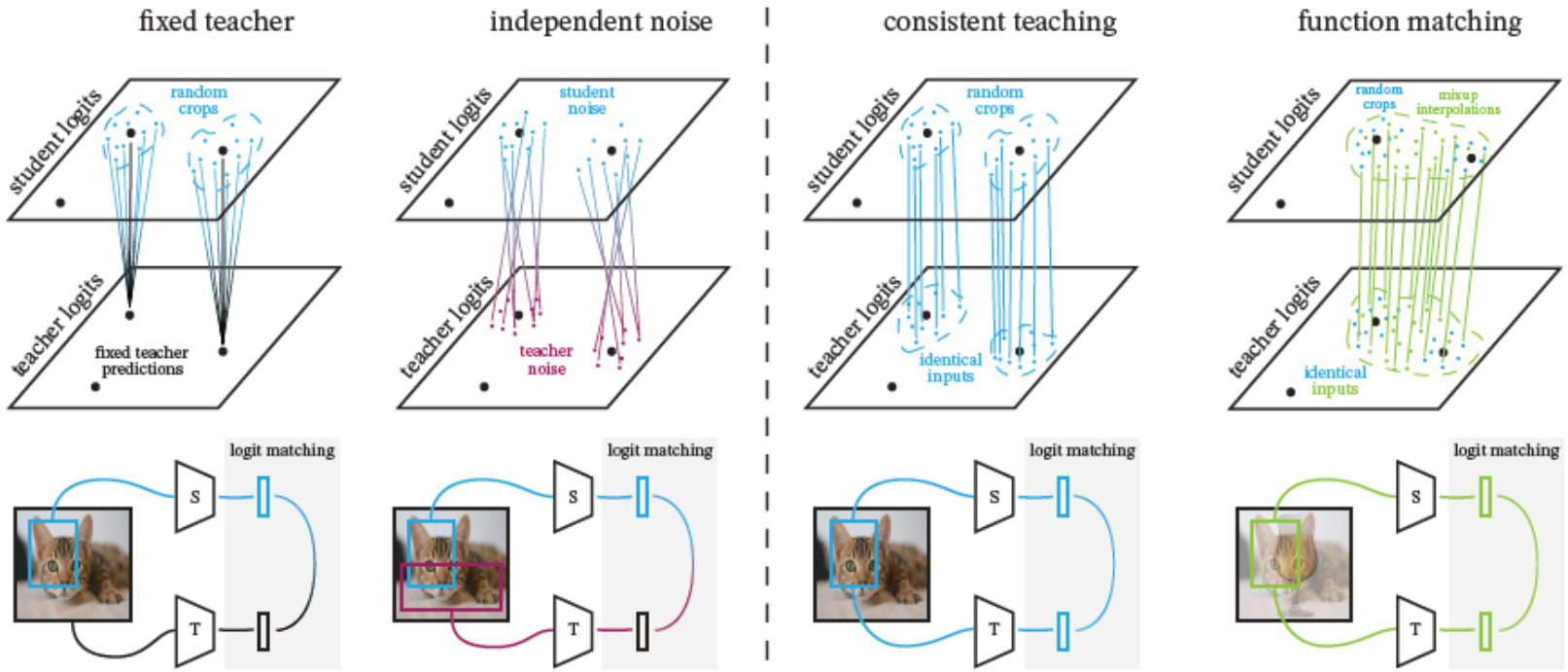
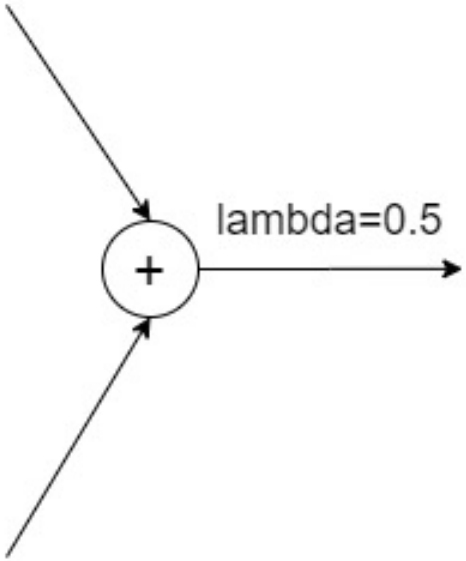


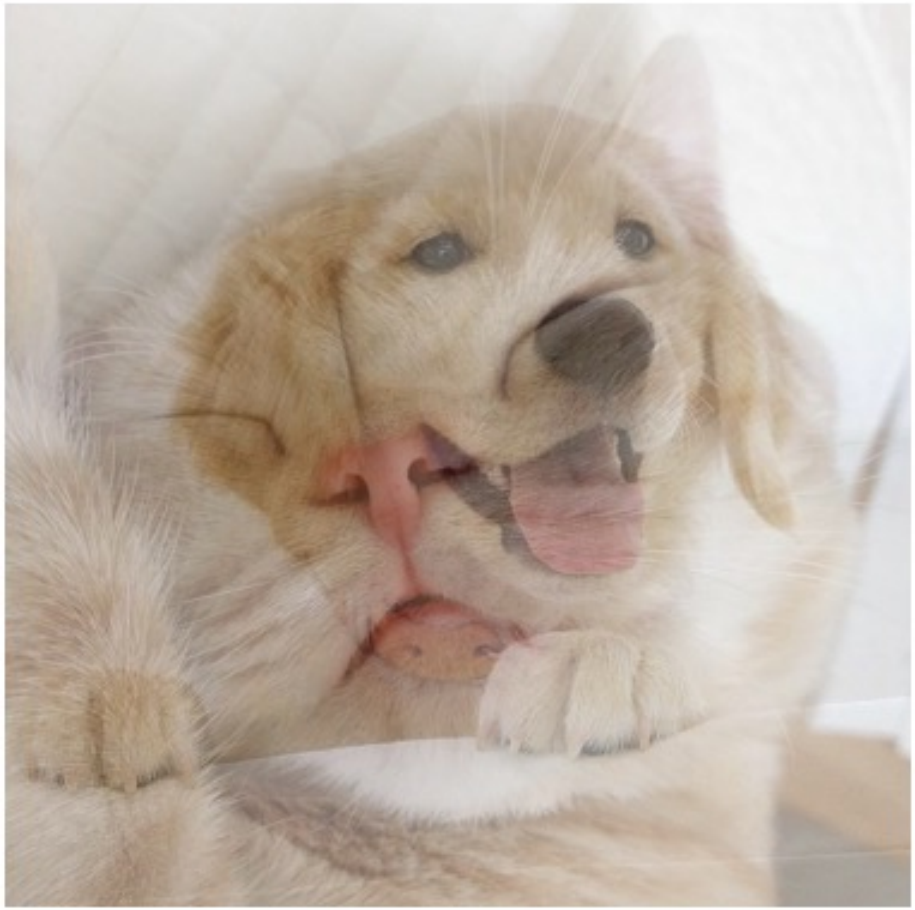
Figure 2. Schematic illustrations of various design choices when doing knowledge distillation. **Left**: *Teacher* receives a fixed image, while *student* receives a random augmentation. **Center-left**: *Teacher* and *student* receive independent image augmentations. **Center-right**: *Teacher* and *student* receive consistent image augmentations. **Right**: *Teacher* and *student* receive consistent image augmentations plus the input image manifold is extended by including linear segments between pairs of images (known as *mixup* [51] augmentation).

MIXUP AUGMENTATION

[1, 0]

[0, 1]





[0.5, 0.5]

TWO TAKEAWAY

- First, teacher and student should process the exact same input image views or, more specifically, same crop and augmentations.
- Second, we want the functions to match on a large number of support points to generalize well. Using an aggressive variant of mixup ,we can generate support points outside the original image manifold.

EXPERIMENTAL SETTING

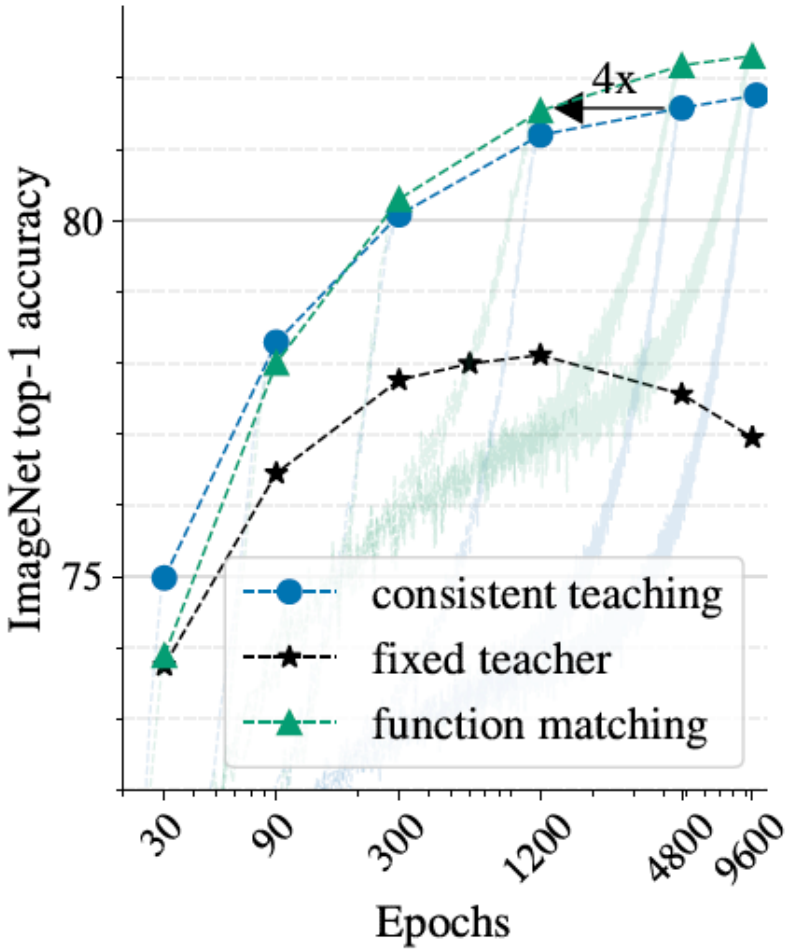
- Five dataset: flowers102 , pets , food101 , sun397 and ILSVRC-2012 (“ImageNet”)
- Teacher and student models: BiT-M-R152x2

EXPERIMENTAL SETTING

$$\mathrm{KL}(p_t||p_s) = \sum_{i \in \mathcal{C}} \left[-p_{t,i} \log p_{s,i} + p_{t,i} \log p_{t,i} \right], \qquad (1)$$

where $\mathcal{C}$ is a set of classes. Also, as in [12], we introduce a temperature parameter T, which is used to adjust the entropy of the predicted softmax-probability distributions before they are used in the loss computation: $p_s \propto \exp(\frac{\log p_s}{T})$ and $p_t \propto \exp(\frac{\log p_t}{T})$.

EXPERIMENTAL RESULTS



EXPERIMENTAL RESULTS

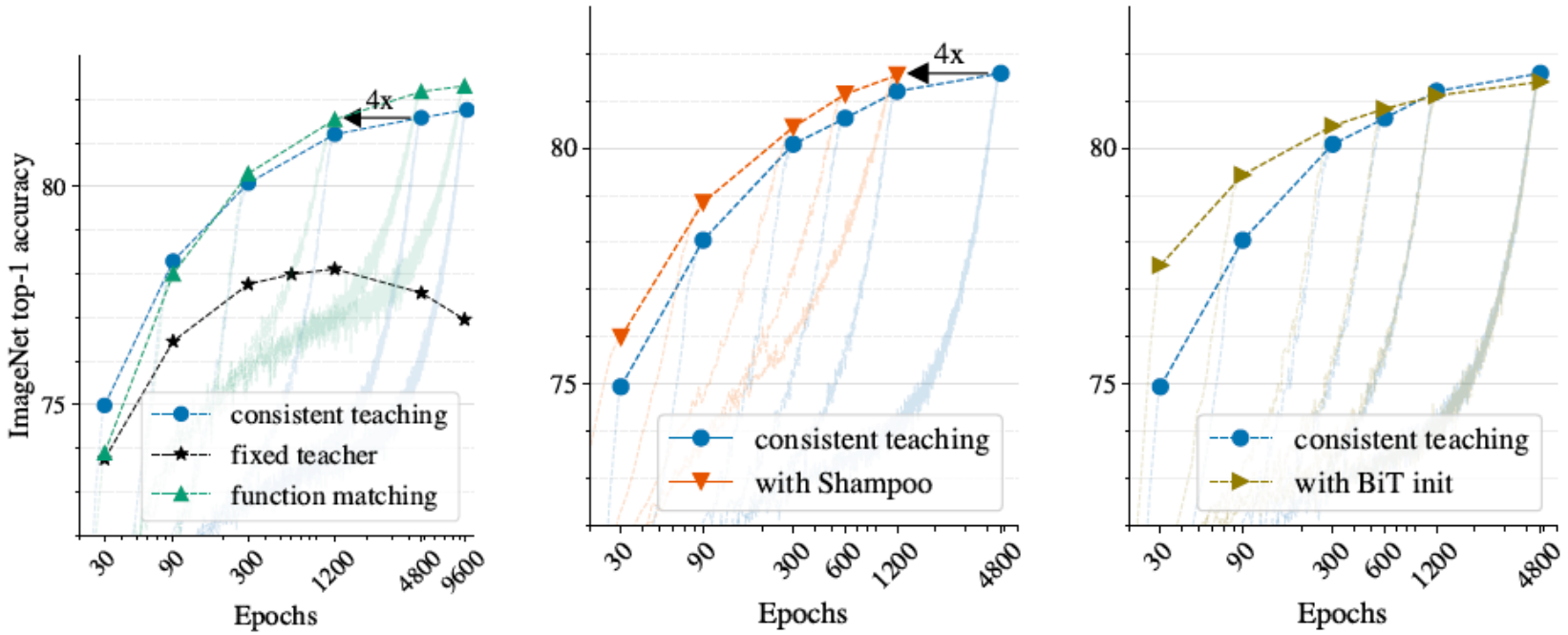
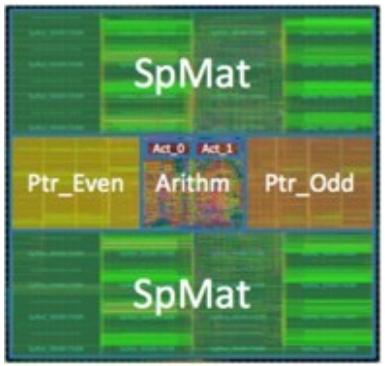


Figure 5. **Left**: Top-1 accuracy on ImageNet of three distillation setups: (1) fixed teacher; (2) consistent teaching; (3) function matching ("FunMatch"). Light color curves show accuracy throughout training, while the solid scatter plots are the final results. The student with a fixed teacher eventually saturates and overfits to it. Both consistent teaching and function matching do not exhibit overfitting or saturation. **Middle**: Reducing the optimization cost, via *Shampoo* preconditioning; with 1200 epochs, it is able to match the baseline trained for 4800 epochs. **Right**: Initializing *student* with pre-trained weights improves short training runs, but harms for the longest schedules.

EIE ACCELERATOR

EIE: Efficient Inference Engine on Compressed Deep Neural Network

ASIC ACCELERATOR THAT RUNS DNN ON MOBILE



Offline
No dependency on network connection

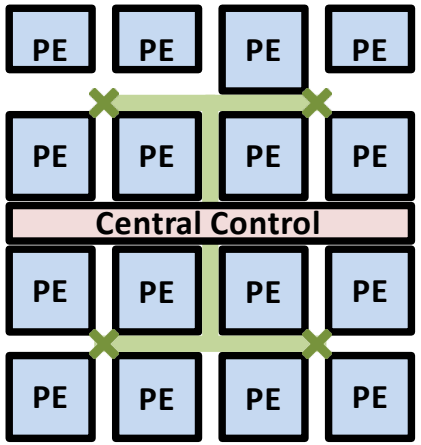
Real Time
No network delay high frame rate

Low Power
High energy efficiency that preserves battery

SOLUTION: EVERYTHING ON CHIP

- We present the **sparse, indirectly indexed, weight shared** MxV accelerator.
- Large DNN models fit on-chip SRAM, 120× energy savings.
- EIE exploits the sparsity of activations (30% non-zero).
- EIE works on compressed model (30x model reduction)
- Distributed both storage and computation across multiple PEs, which achieves load balance and good scalability.
- Evaluated EIE on a wide range of deep learning models, including CNN for object detection, LSTM for natural language processing and image captioning. We also compare EIE to CPUs, GPUs, and other accelerators.

DISTRIBUTE STORAGE AND PROCESSING



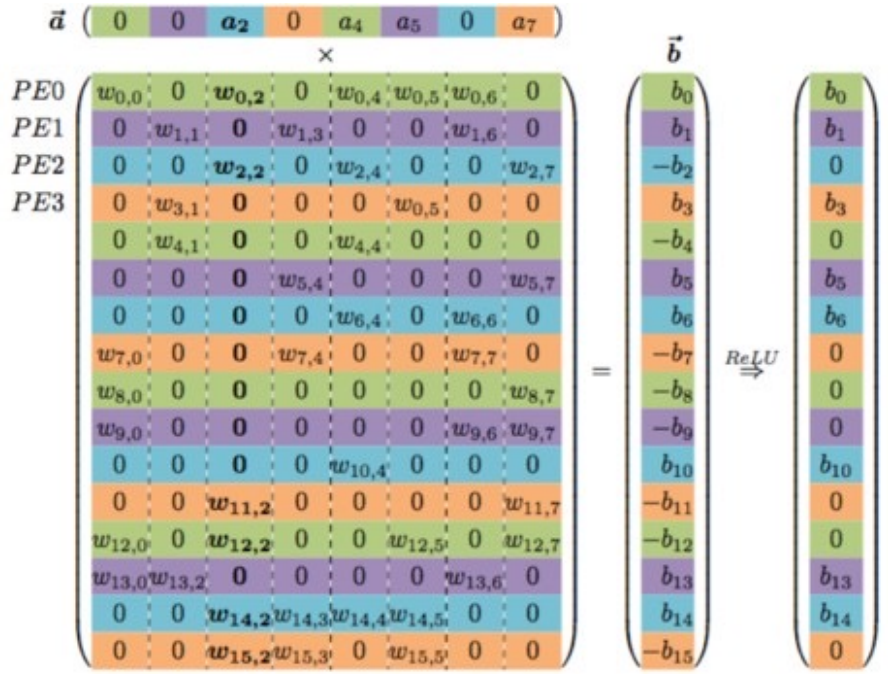
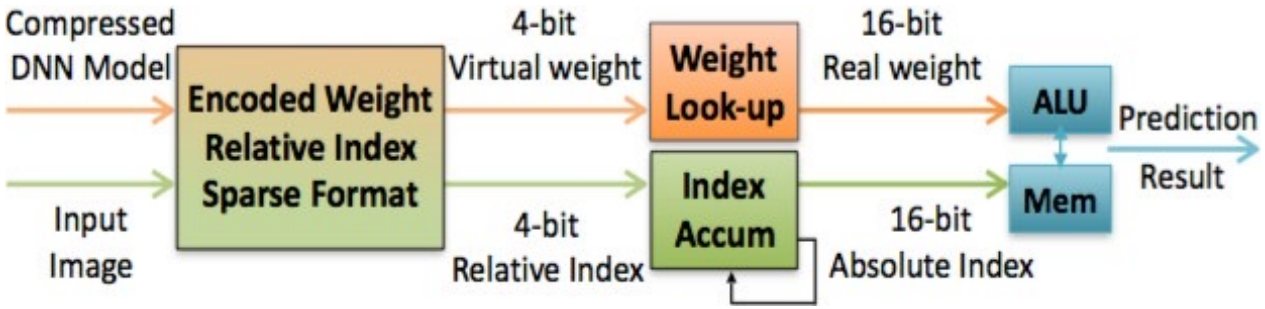


Figure 2: Matrix W and vectors a and b are interleaved over 4 PEs. Elements of the same color are stored in the same PE.

INSIDE EACH PE:



EVALUATION

1.Cycle-accurate C++ simulator. Two abstract methods: Propagate and Update. Used for DSE and verification.

2.RTL in Verilog, verified its output result with the golden model in Modelsim.

3.Synthesized EIE using the Synopsys Design Compiler (DC) under the TSMC 45nm GP standard VT library with worst case PVT corner.

4.Placed and routed the PE using the Synopsys IC compiler (ICC). We used Cacti to get SRAM area and energy numbers.

5.Annotated the toggle rate from the RTL simulation to the gate-level netlist, which was dumped to switching activity interchange format (SAIF), and estimated the power using Prime-Time PX.

BASELINE AND BENCHMARK

- CPU: Intel Core-i7 5930k
- GPU: NVIDIA TitanX GPU
- Mobile GPU: Jetson TK1 with NVIDIA

Table 3: Benchmark from state-of-the-art DNN models

Layer	Size	Weight%	Act%	FLOP%	Description
Alex-6	9216, 4096	9%	35.1%	3%	Compressed AlexNet [1] for large scale image classification
Alex-7	4096, 4096	9%	35.3%	3%	
Alex-8	4096, 1000	25%	37.5%	10%	
VGG-6	25088, 4096	4%	18.3%	1%	Compressed VGG-16 [3] for large scale image classification and object detection
VGG-7	4096, 4096	4%	37.5%	2%	
VGG-8	4096, 1000	23%	41.1%	9%	
NT-We	4096, 600	10%	100%	10%	Compressed NeuralTalk [7] with RNN and LSTM for automatic image captioning
NT-Wd	600, 8791	11%	100%	11%	
NTLSTM	1201, 2400	10%	100%	11%	

LAYOUT OF AN EIE PE

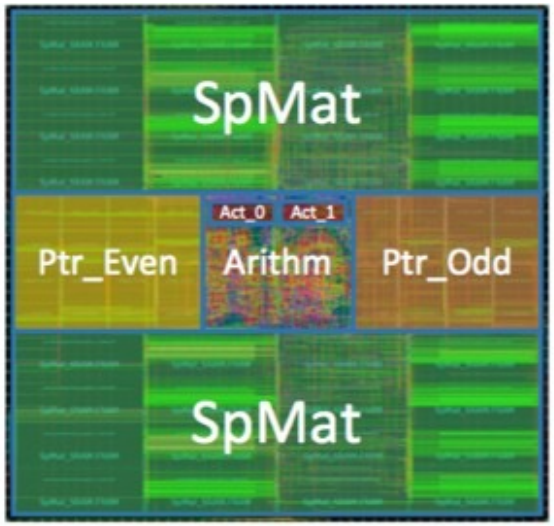
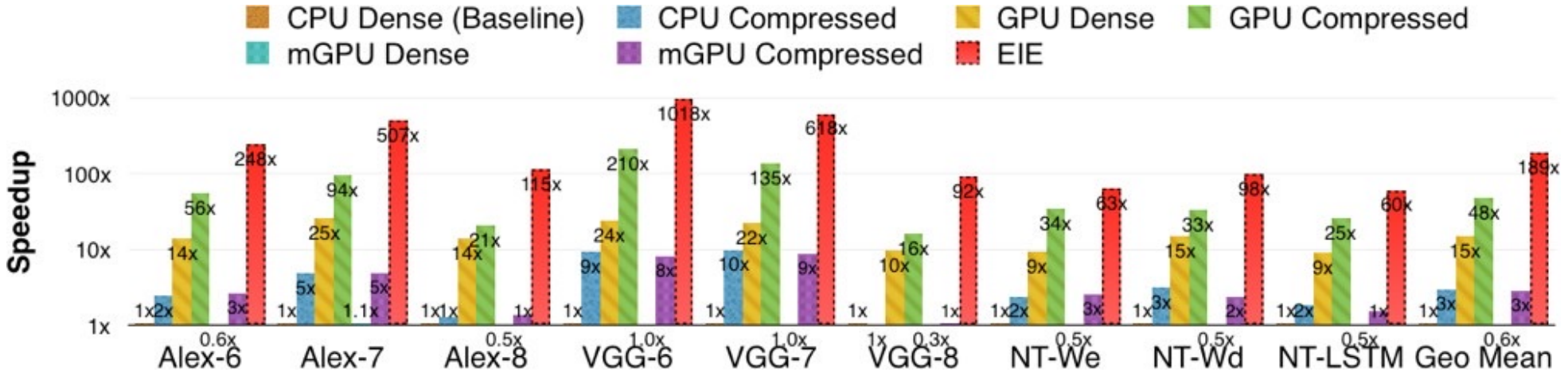


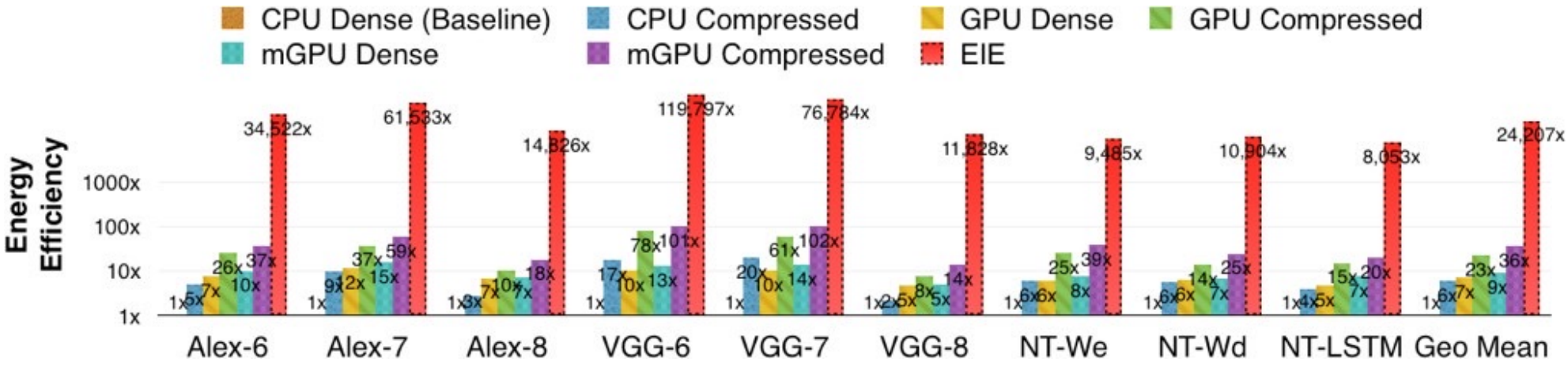
Figure 7: Layout of one PE in EIE under TSMC 45nm process.

	Power (mW)	(%)	Area (μm^2)	(%)
Total	9.157		638,024	
memory	5.416	(59.15%)	594,786	(93.22%)
clock network	1.874	(20.46%)	866	(0.14%)
register	1.026	(11.20%)	9,465	(1.48%)
combinational	0.841	(9.18%)	8,946	(1.40%)
filler cell			23,961	(3.76%)
Act_queue	0.112	(1.23%)	758	(0.12%)
PtrRead	1.807	(19.73%)	121,849	(19.10%)
SpmatRead	4.955	(54.11%)	469,412	(73.57%)
ArithmUnit	1.162	(12.68%)	3,110	(0.49%)
ActRW	1.122	(12.25%)	18,934	(2.97%)
filler cell			23,961	(3.76%)

Table 2: The implementation results of one PE in EIE and the breakdown by component type (line 3-7), by module (line 8-13). The critical path of EIE is 1.15ns

RESULT: SPEEDUP / ENERGY EFFICIENCY





RESULT: SPEEDUP

Table 4: Performance comparison between CPU, GPU, mobile GPU implementations and EIE.

Platform	Batch Size	Matrix Type	AlexNet			VGG16			NT-		
			FC6	FC7	FC8	FC6	FC7	FC8	We	Wd	LSTM
CPU (Core i7-5930k)	1	dense	7516.2	6187.1	1134.9	35022.8	5372.8	774.2	605.0	1361.4	470.5
		sparse	3066.5	1282.1	890.5	3774.3	545.1	777.3	261.2	437.4	260.0
	64	dense	318.4	188.9	45.8	1056.0	188.3	45.7	28.7	69.0	28.8
		sparse	1417.6	682.1	407.7	1780.3	274.9	363.1	117.7	176.4	107.4
GPU (Titan X)	1	dense	541.5	243.0	80.5	1467.8	243.0	80.5	65	90.1	51.9
		sparse	134.8	65.8	54.6	167.0	39.8	48.0	17.7	41.1	18.5
	64	dense	19.8	8.9	5.9	53.6	8.9	5.9	3.2	2.3	2.5
		sparse	94.6	51.5	23.2	121.5	24.4	22.0	10.9	11.0	9.0
mGPU (Tegra K1)	1	dense	12437.2	5765.0	2252.1	35427.0	5544.3	2243.1	1316	2565.5	956.9
		sparse	2879.3	1256.5	837.0	4377.2	626.3	745.1	240.6	570.6	315
	64	dense	1663.6	2056.8	298.0	2001.4	2050.7	483.9	87.8	956.3	95.2
		sparse	4003.9	1372.8	576.7	8024.8	660.2	544.1	236.3	187.7	186.5
EIE	**Theoretical Time**		**28.1**	**11.7**	**8.9**	**28.1**	**7.9**	**7.3**	**5.2**	**13.0**	**6.5**
	Actual Time		**30.3**	**12.2**	**9.9**	**34.4**	**8.7**	**8.4**	**8.0**	**13.9**	**7.5**

USEFUL COMPUTATION / LOAD BALANCE

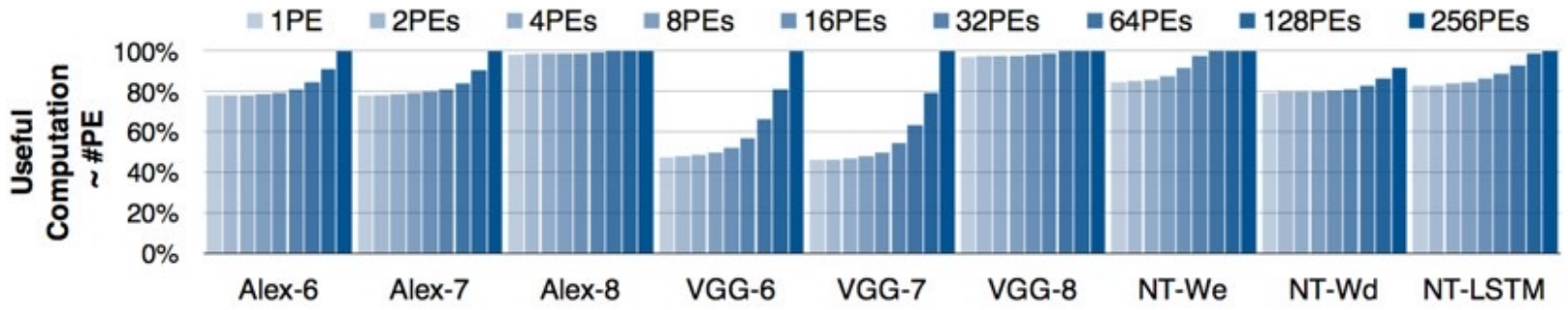


Figure 12: The number of padding zeros decreases as the number of PEs goes up, leading to less padding zeros and better compute efficiency.

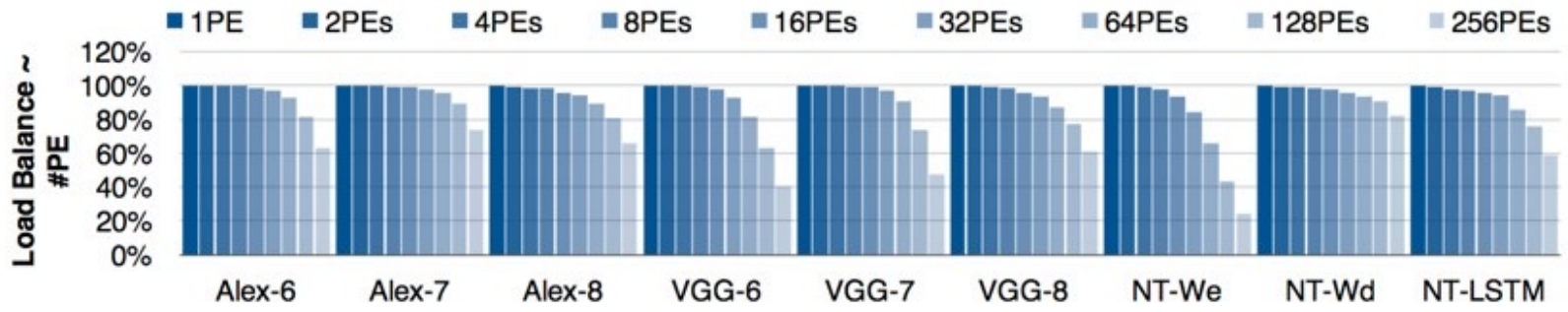


Figure 13: Load efficiency is measured by the ratio of stalled cycles over total cycles in ALU. More PEs lead to worse load imbalance accompanied with less load efficiency. This explains the sub-linear speedup at large number of PEs.

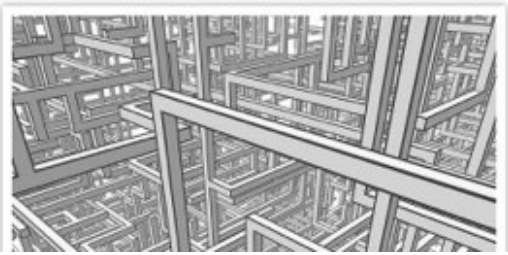
MEDIA COVERAGE

THENEXTPLATFORM

HOME COMPUTE STORE CONNECT CONTROL CODE ANALYZE HPC ENTERPRIS

EMERGENT CHIP VASTLY ACCELERATES DEEP NEURAL NETWORKS

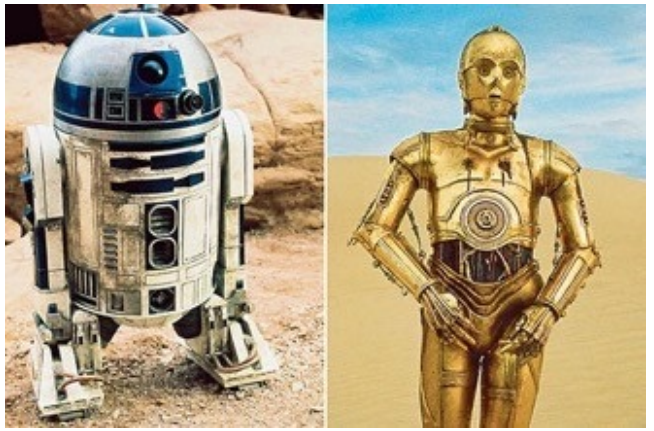
December 8, 2015 Nicole Hemsoth



Stanford University PhD candidate, Song Han, who works under advisor and networking pioneer, Dr. Bill Dally, responded in a most soft-spoken and thoughtful way to the question of whether the coupled software and hardware architecture he developed might change the world.

http://www.nextplatform.com/2015/12/08/emergent-chip-vastly-accelerates-deep-neural-networks/

HARDWARE FOR DEEP LEARNING



PC | **Mobile** | **Intelligent Mobile**

intel

?

Computation → Mobile Computation → Intelligent Mobile Computation

CONCLUSION

- We present EIE, an energy-efficient engine optimized to operate on compressed deep neural networks.

- By leveraging sparsity in both the activations and the weights, EIE reduces the energy needed to compute a typical FC layer by 3,000×.

- Three factors for energy saving:
  matrix is compressed by 35×;
  DRAM => SRAM: 120×;
  take advantage of sparse activation: 3×;

Questions