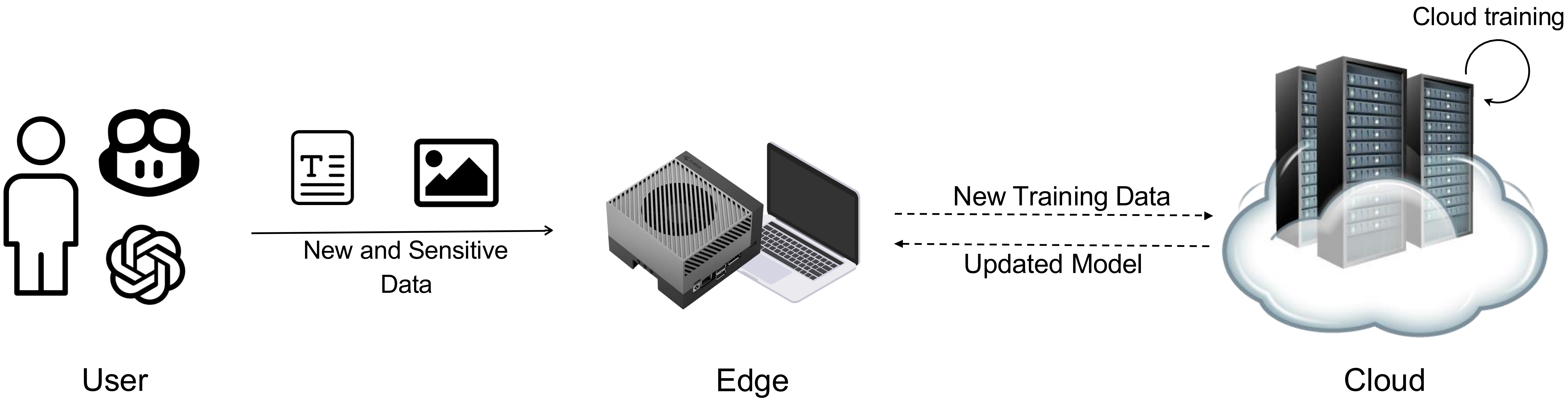
On-Device Training

Apr.24, 2025

Acknowledgment: Some slides and materials are adapted from EfficientML, courtesy of Prof. Song Han and the HAN Lab.

On-Device Learning

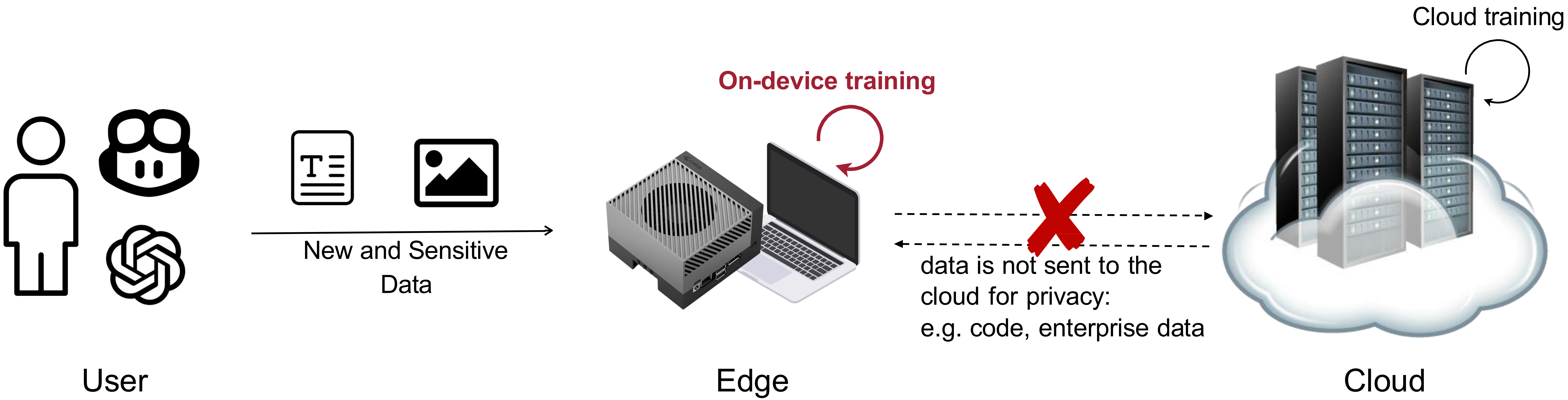
- Learning at the "edge", rather than cloud.



- Customization: AI systems need to continually adapt to new data collected from the sensors.

On-Device Learning

- Transfer learning at the "edge", rather than cloud.



- Customization: AI systems need to continually adapt to new data collected from the sensors.
- Privacy: Data does not leave the device.

Lecture Plan

Today we will discuss:

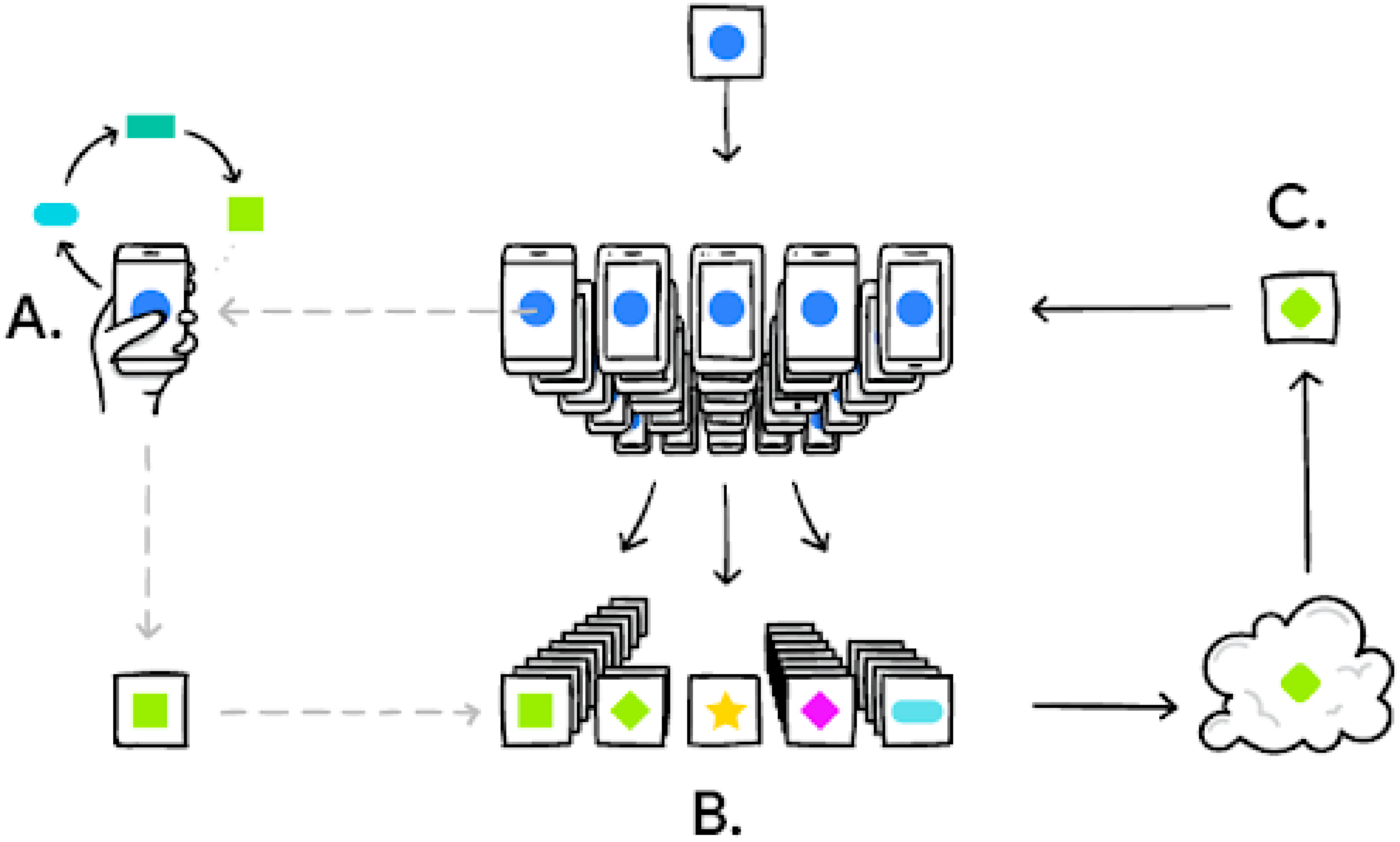
1. Federated learning and the deep leakage from gradients
2. Pruning, quantization and knowledge distillation
3. Memory bottleneck of on-device training
4. Tiny transfer learning (TinyTL)
5. Sparse back-propagation (SparseBP)

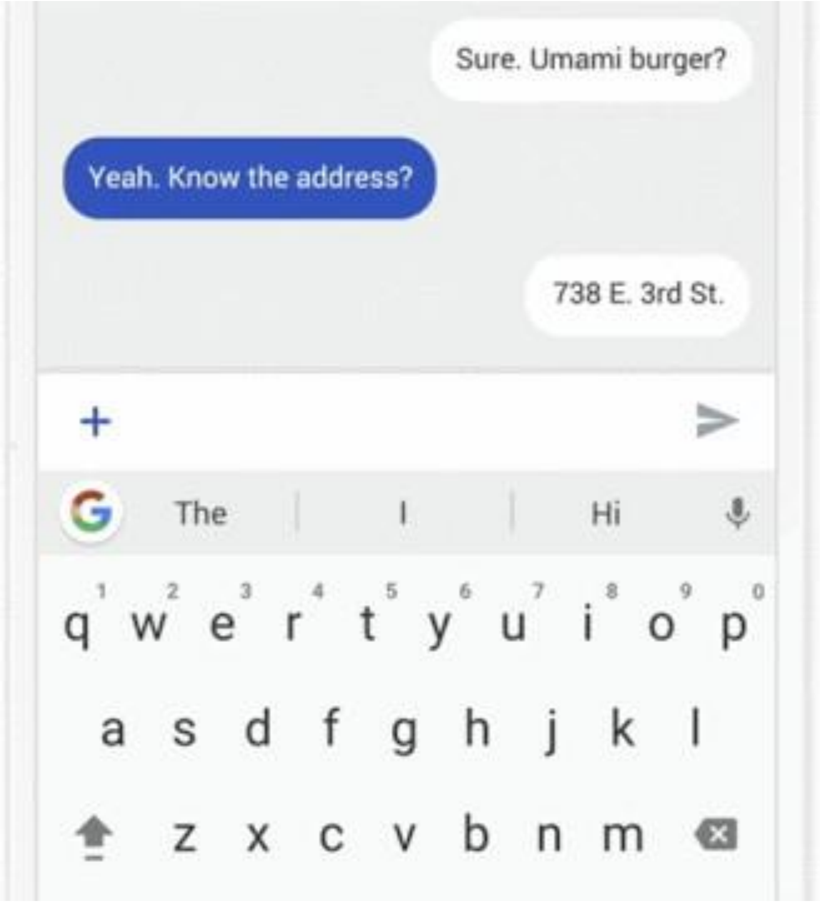
Lecture Plan

1. **Federated Learning and the deep leakage from gradients**
2. Pruning, quantization and knowledge distillation
3. Memory bottleneck of on-device training
4. Tiny transfer learning (TinyTL)
5. Sparse back-propagation (SparseBP)

On-Device Learning

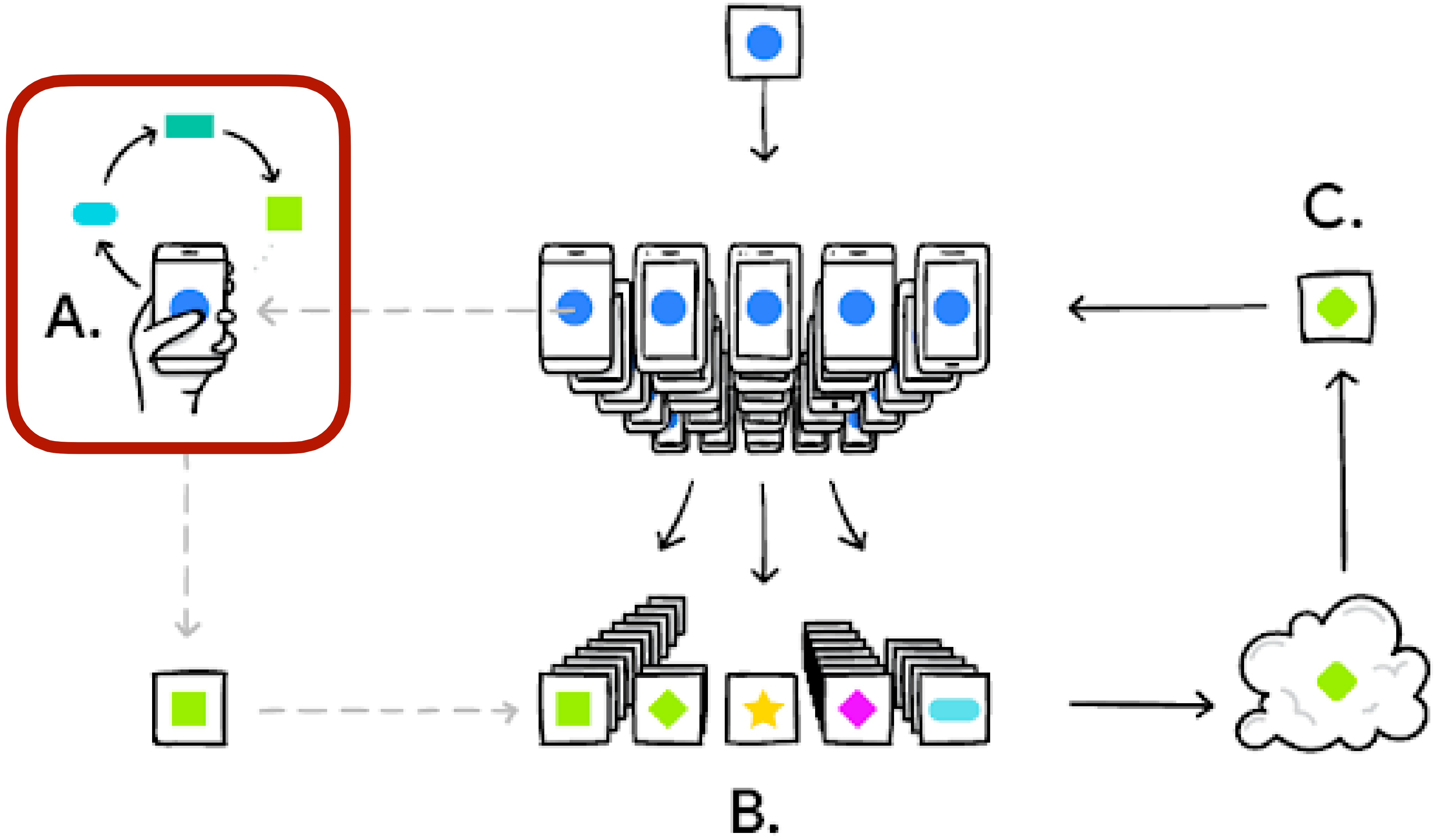
Federated learning: only share the gradients / weights, user data stays local.





Background of Federated Learning

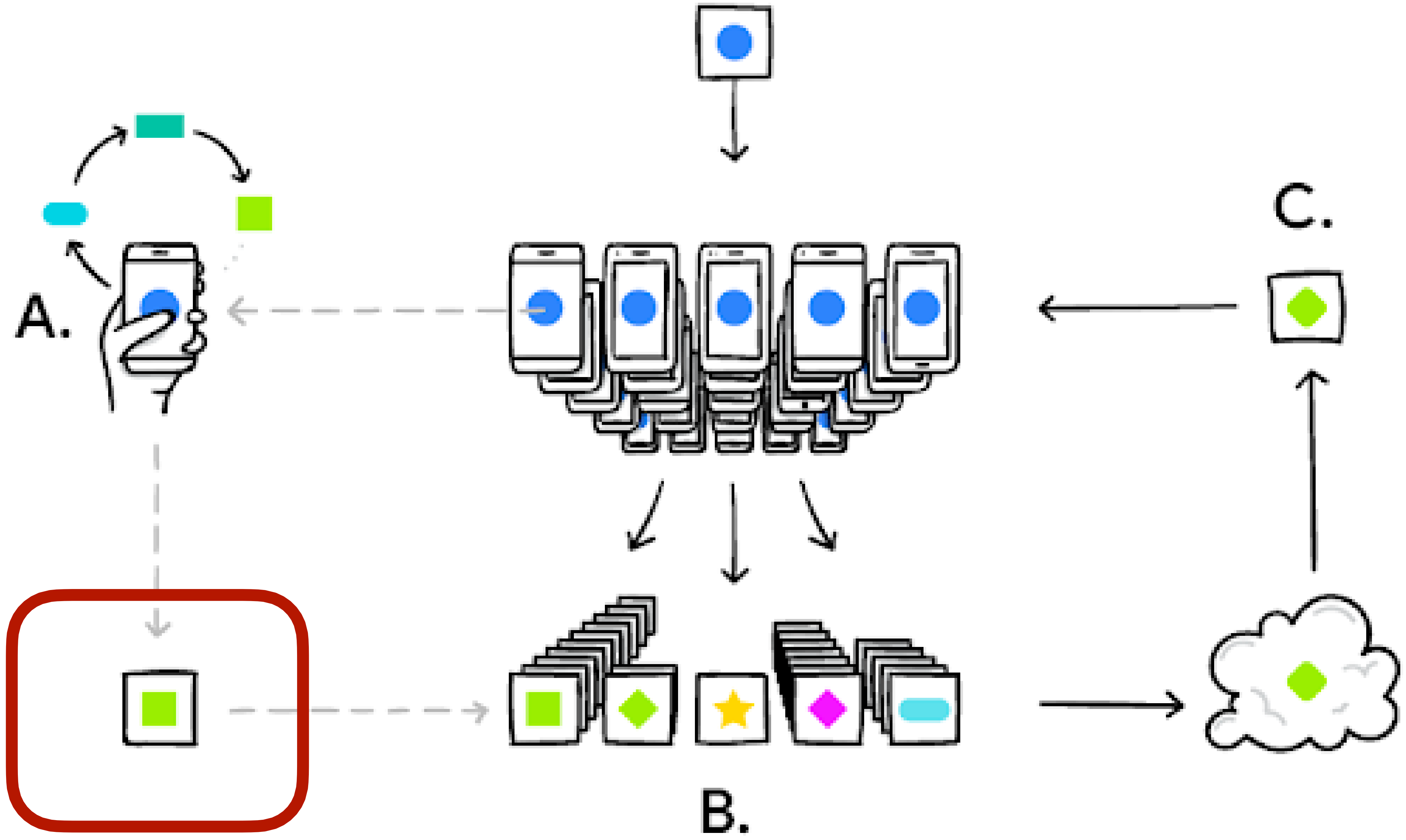
FedAvg Algorithm



1. Users generate personal data on device and perform local training.

Background of Federated Learning

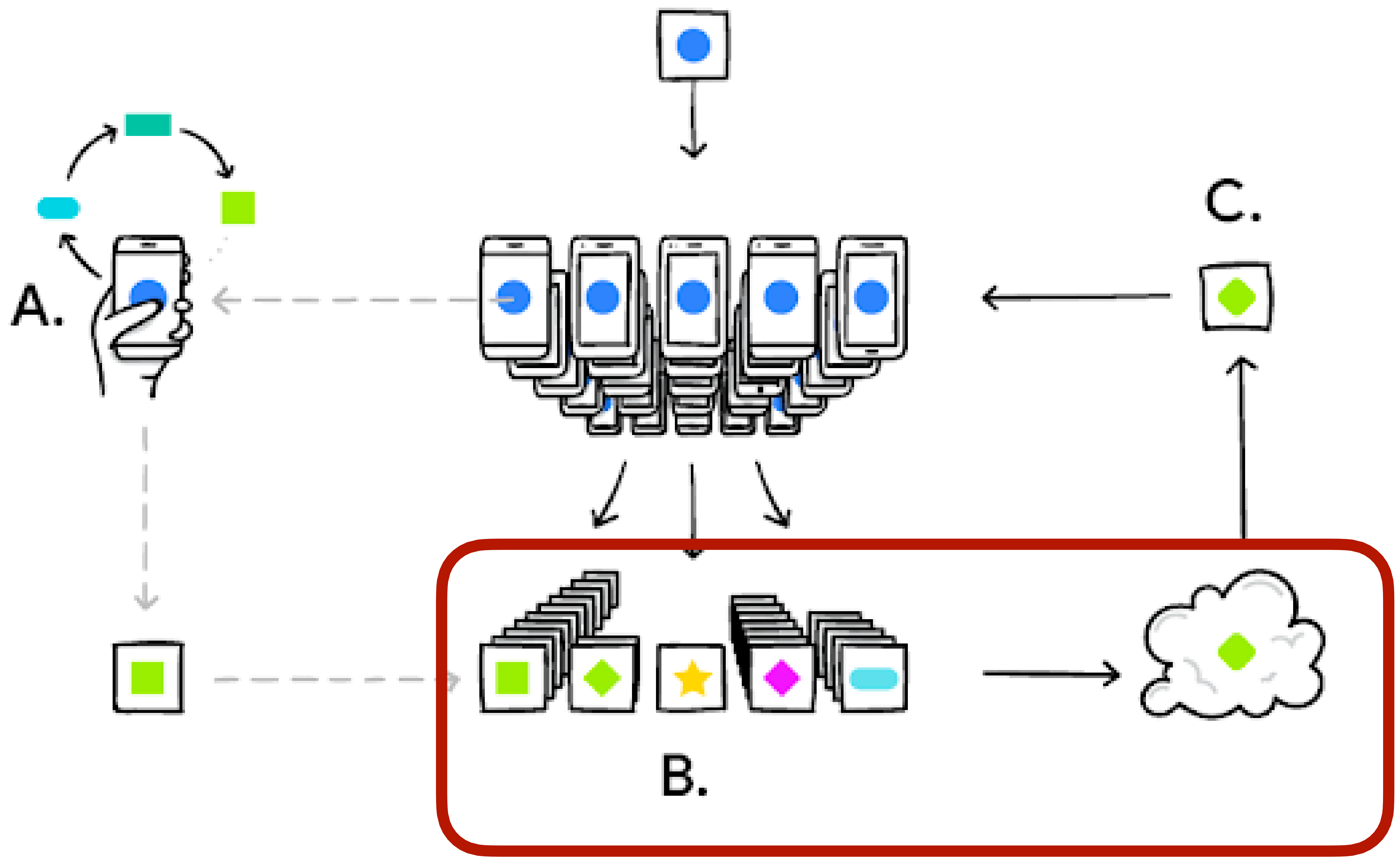
FedAvg Algorithm



1. Users generate personal data on device and perform local training.
2. Each device update its model using local data for N iterations.

Background of Federated Learning

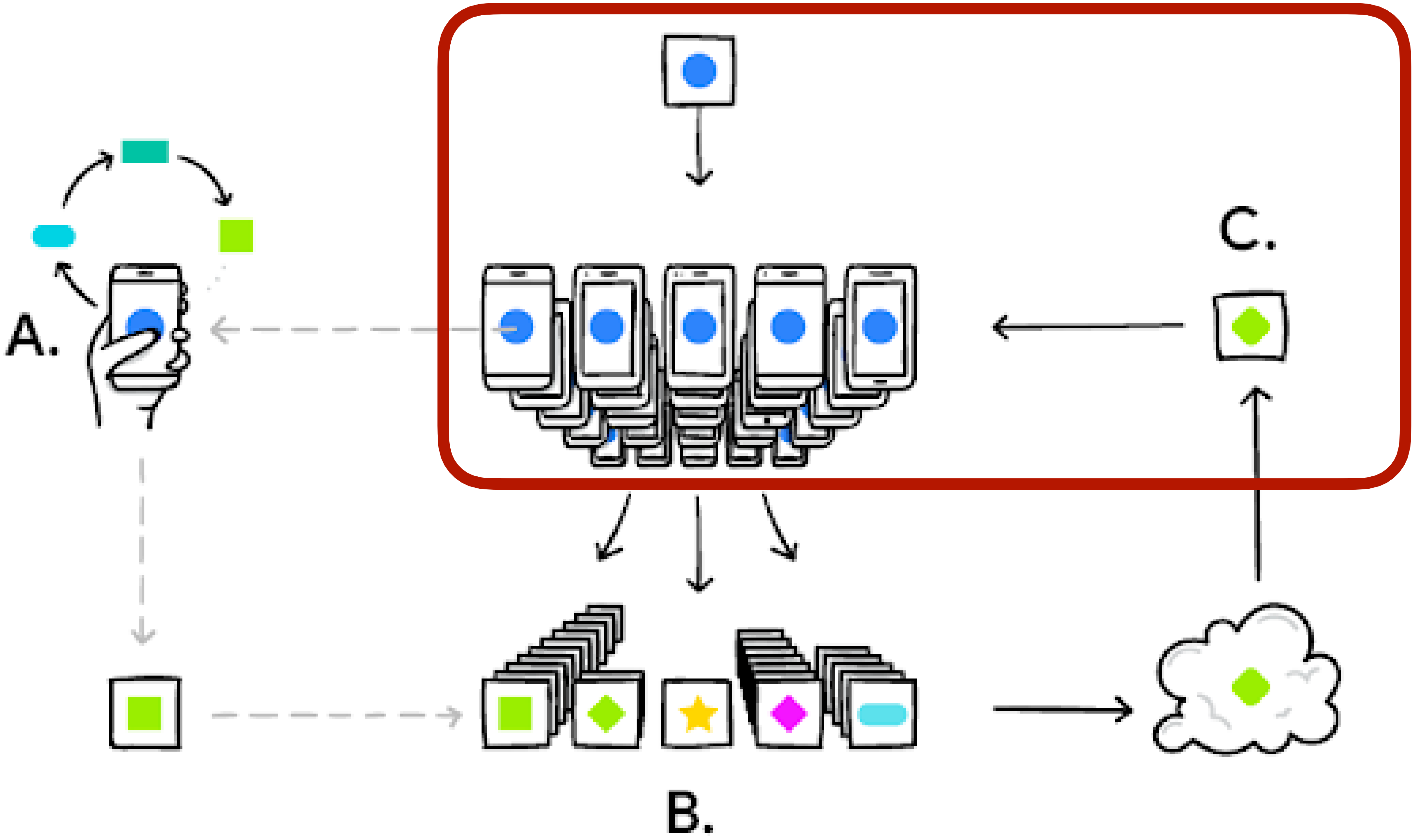
FedAvg Algorithm



1. Users generate personal data on device and perform local training.
2. Each device update its model using local data for N iterations.
3. Updated models are sent to the server.

Background of Federated Learning

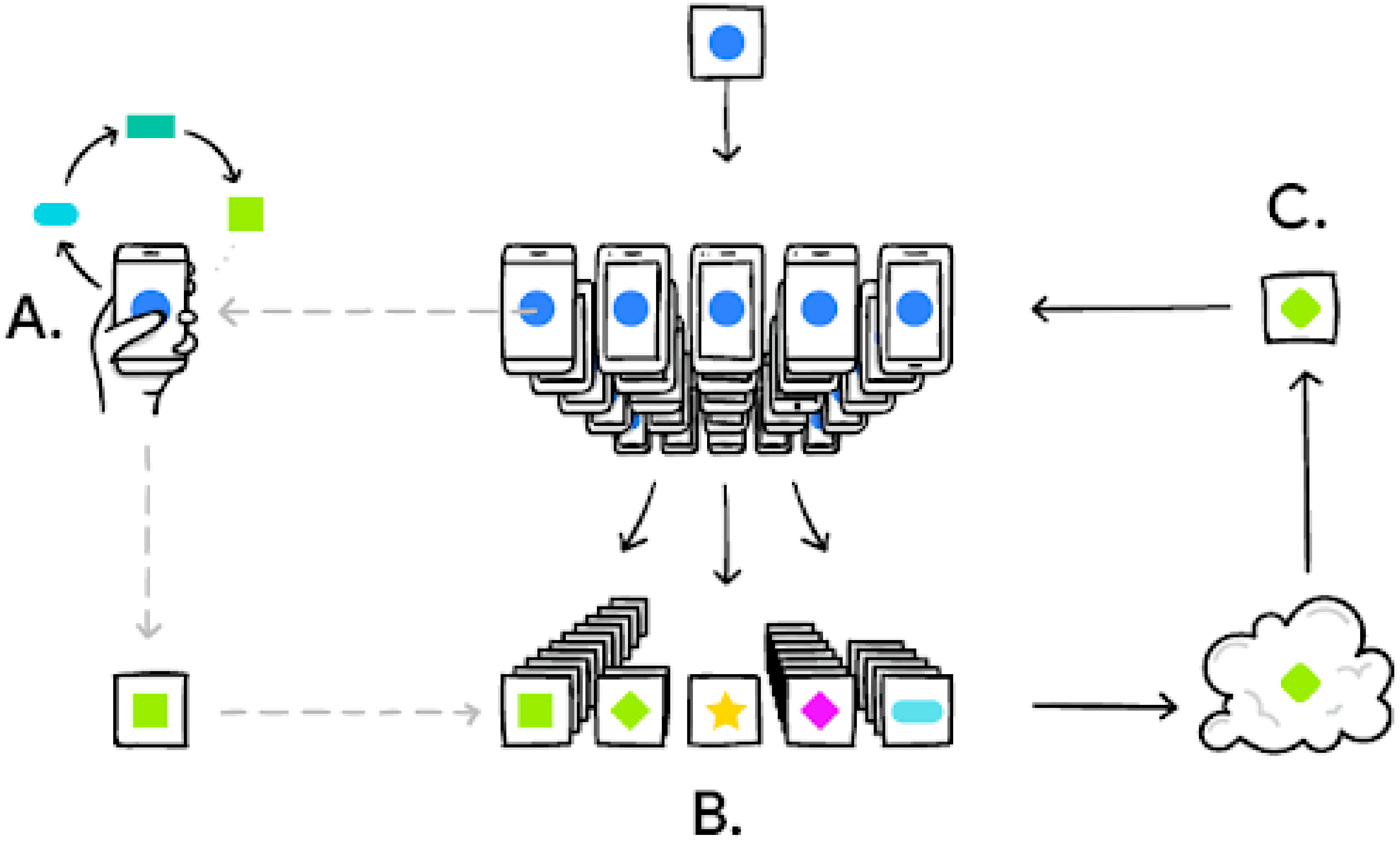
FedAvg Algorithm



1. Users generate personal data on device and perform local training.
2. Each device update its model using local data for N iterations.
3. Updated models are sent to the server.
4. Models are averaged on the server and sent back to devices.

Background of Federated Learning

FedAvg Algorithm

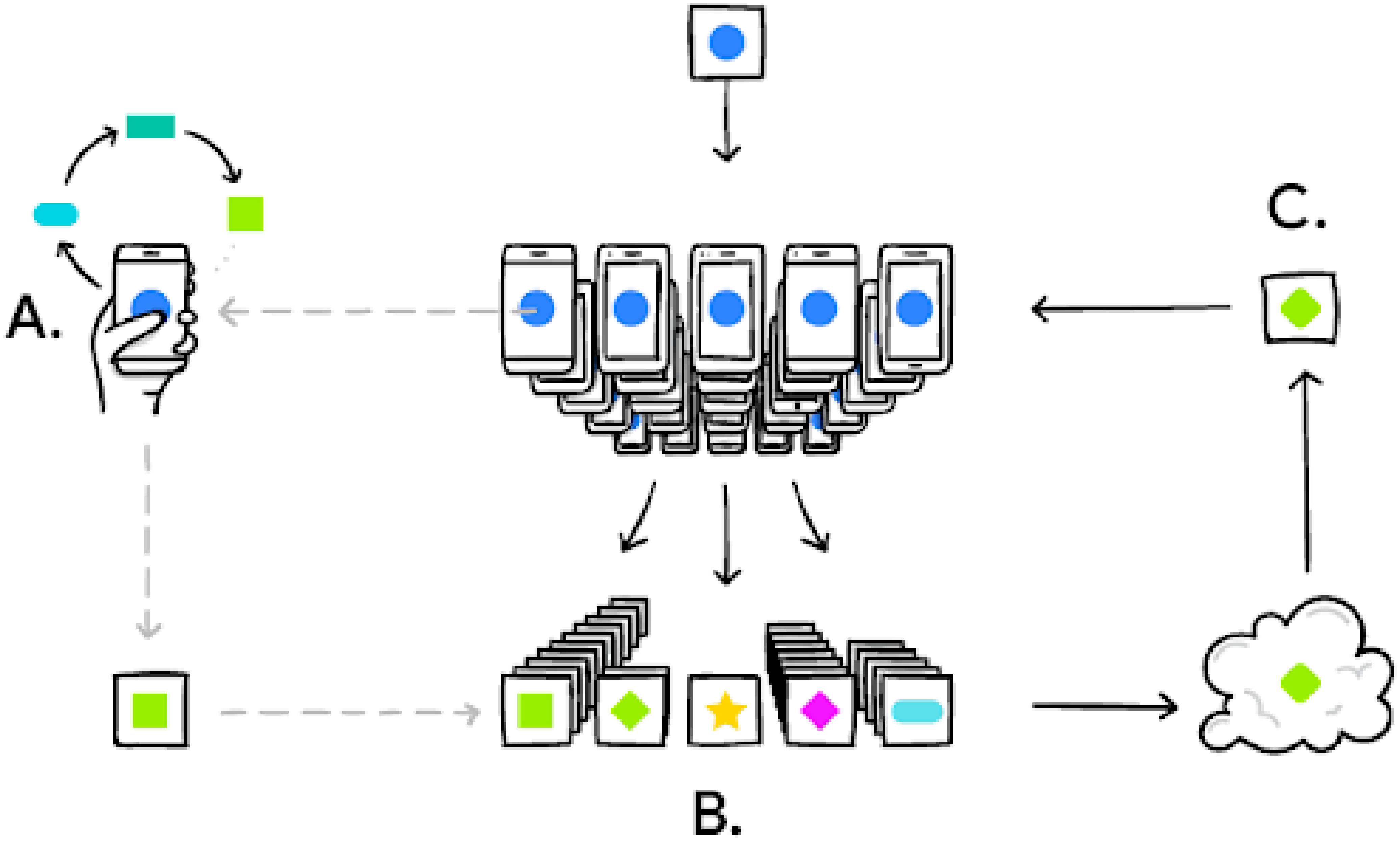


1. Users generate personal data on device and perform local training.
2. Each device update its model using local data for N iterations.
3. Updated models are sent to the server.
4. Models are averaged on the server and sent back to devices.

The important & private user data **NEVER** leaves local devices.

Background of Federated Learning

FedAvg Algorithm

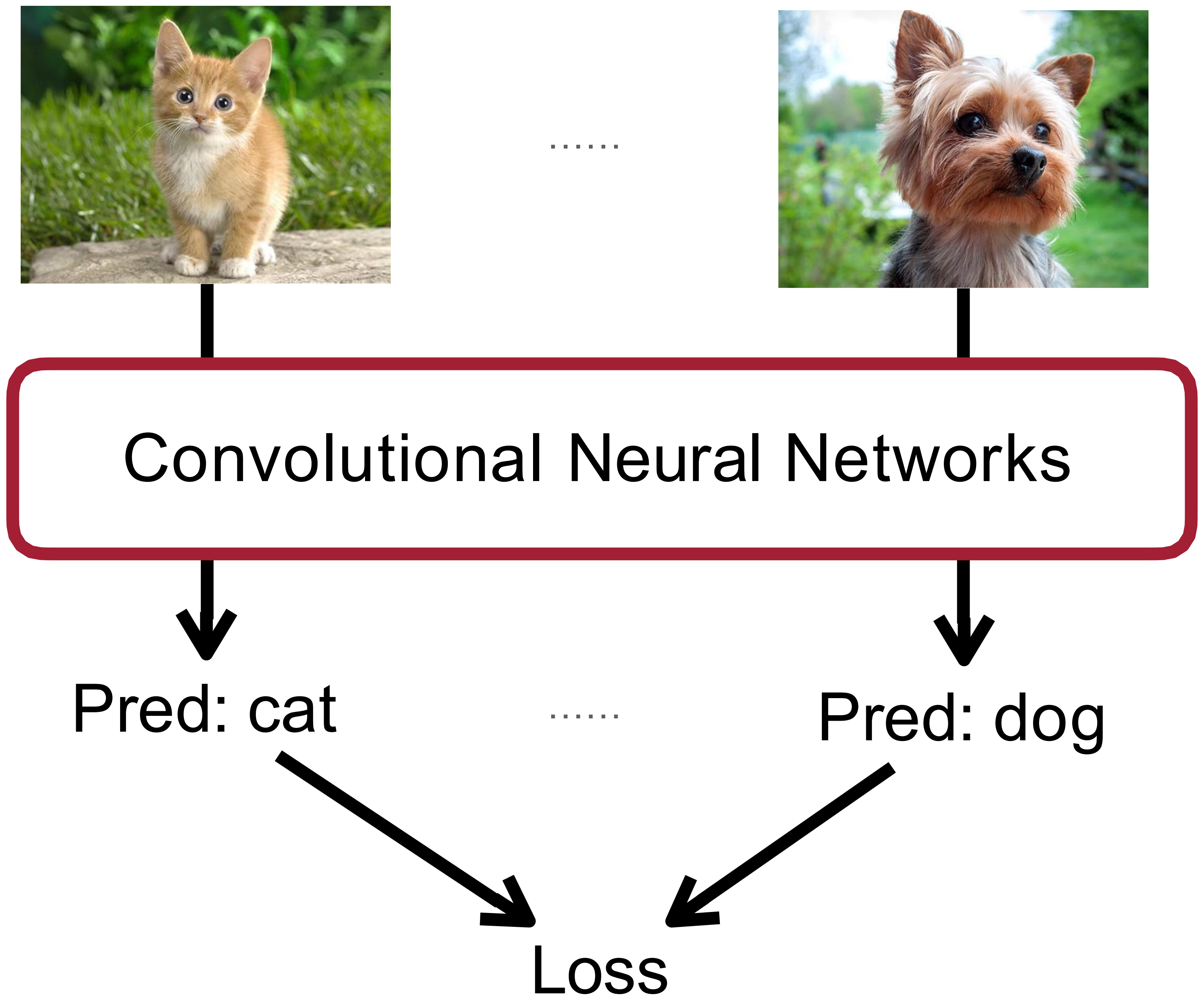


1. Users generate personal data on device and perform local training.
2. Each device update its model using local data for N iterations.
3. Updated models are sent to the server.
4. Models are averaged on the server and sent back to devices.

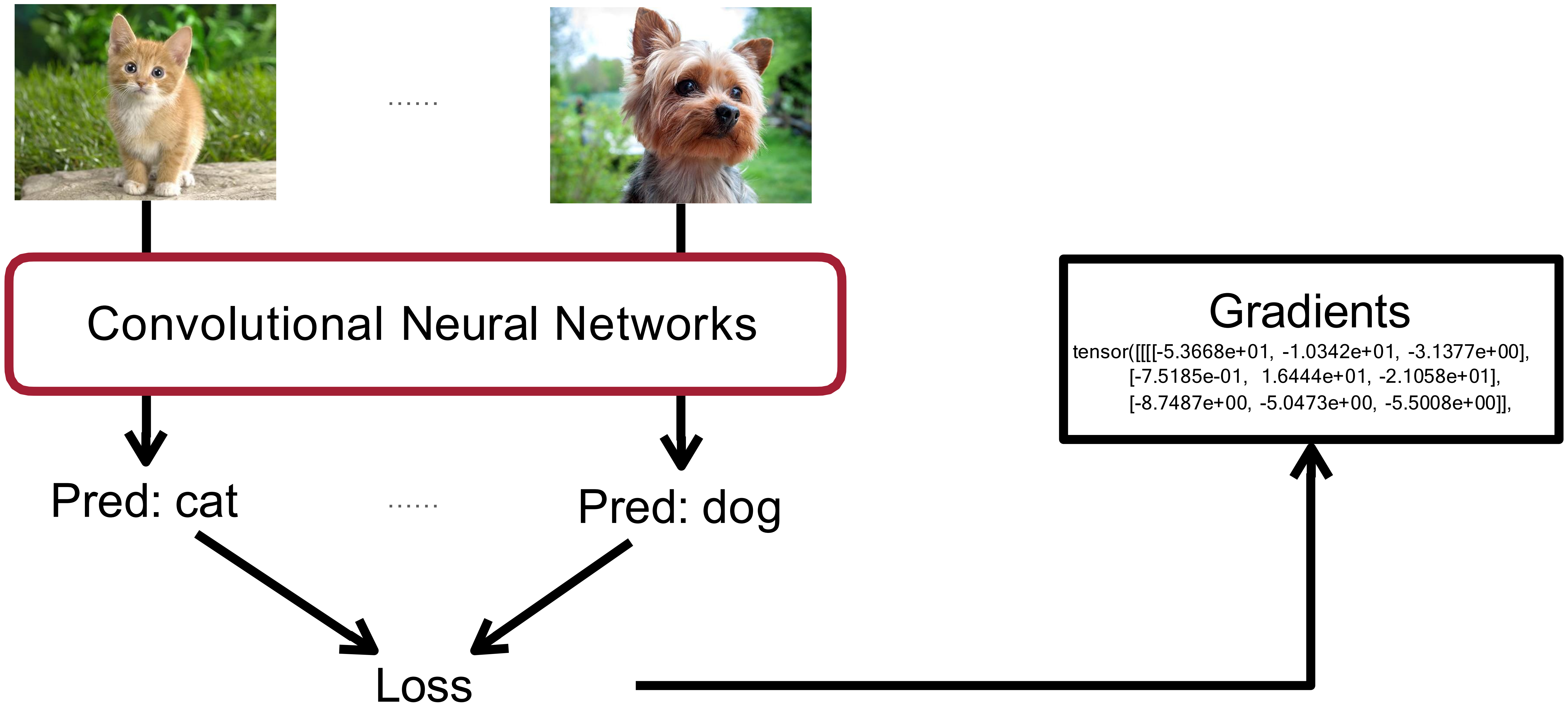
The important & private user data never leaves local devices.

Safe...?

Rethink the Safety of Gradients

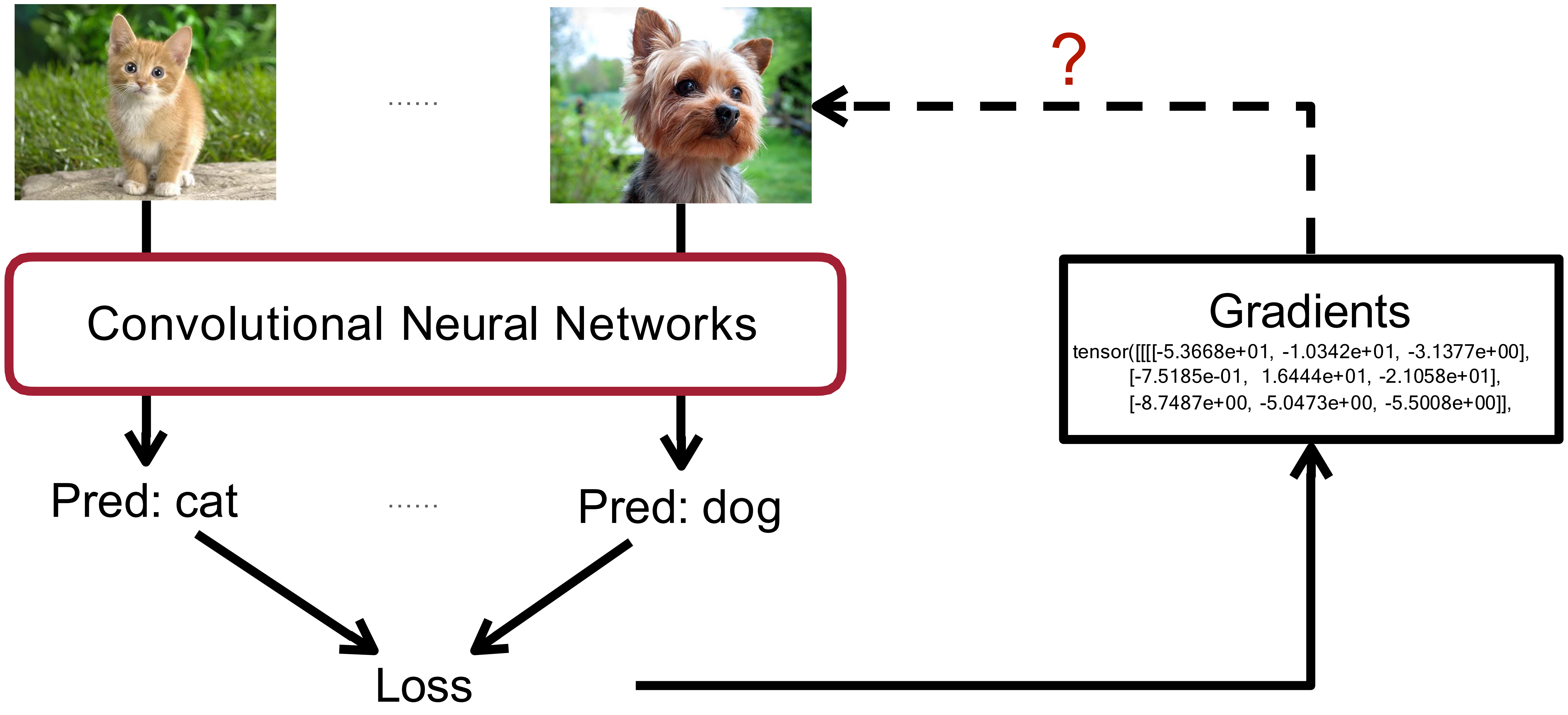


Rethink the Safety of Gradients



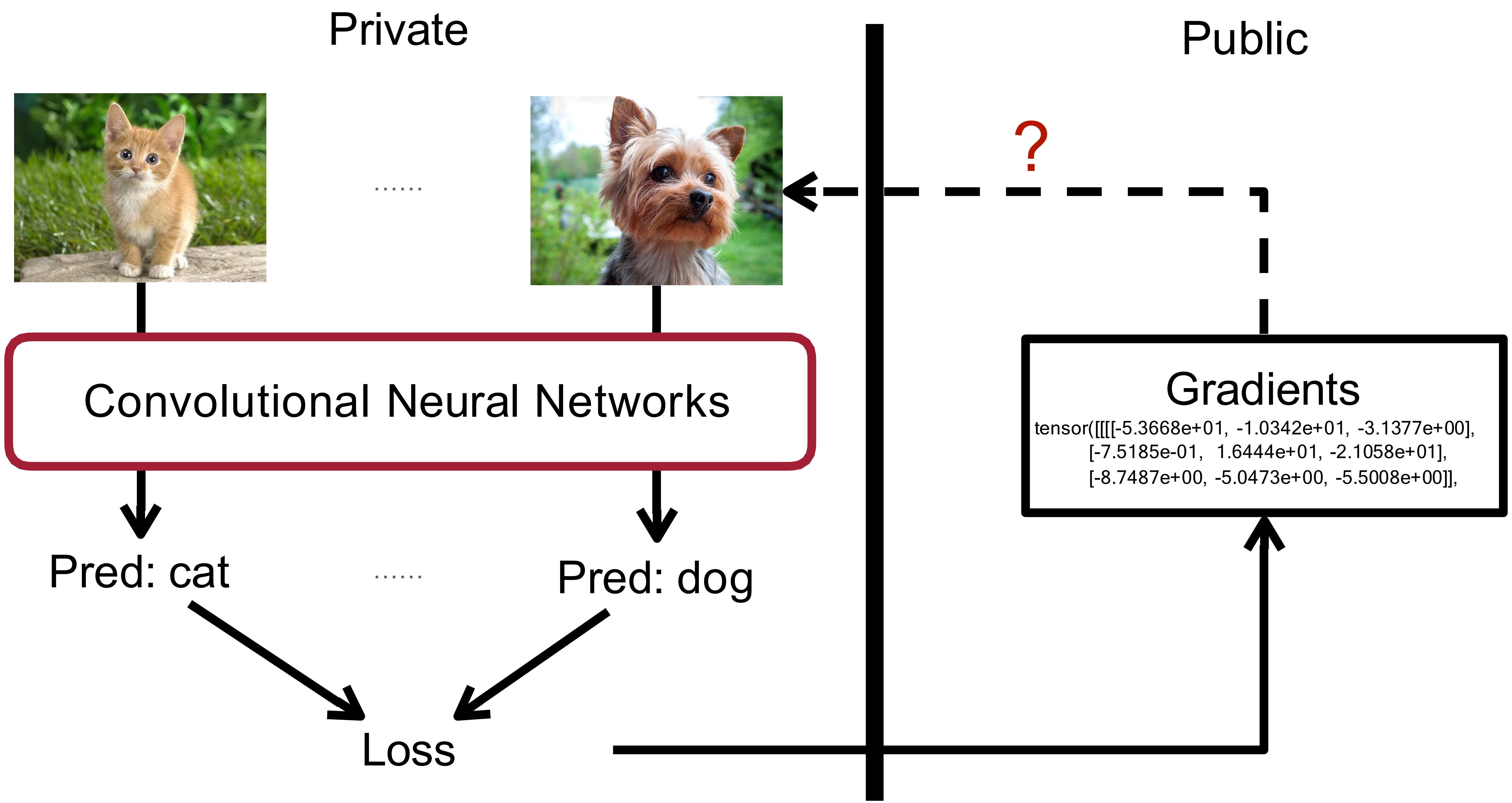
derive gradients from model and training data.

Rethink the Safety of Gradients



Can we derive the training data from gradients?

Rethink the Safety of Gradients



If that is possible, then sharing the gradient is not safe!

Rethink the Safety of Gradients

Existing Work of Gradient Inversion

Membership Inference [Shokri 2016]

- Given gradients, it’s possible to find whether a data point belongs to the batch.

Property Inference [Melis 2018]

- Given gradients, it’s possible to find whether a data point with certain property is in the batch.

Gradients

```
tensor([[[[-5.3668e+01, -1.0342e+01, -3.1377e+00],
          [-7.5185e-01,  1.6444e+01, -2.1058e+01],
          [-8.7487e+00, -5.0473e+00, -5.5008e+00]],
```

Membership Inference

Whether a record is used in the batch.

Property Inference

Whether a sample with certain property is in the batch.

Exploiting unintended feature leakage in collaborative learning. [Melis 2018]
Membership inference attacksagainst machine learning models. [Shokri 2016]

Rethink the Safety of Gradients

Existing Work of Gradient Inversion

Membership Inference [Shokri 2016]

- Given gradients, it's possible to find whether a data point belongs to the batch.

Property Inference [Melis 2018]

- Given gradients, it's possible to find whether a data point with certain property is in the batch.

Gradients contain certain information about the training data.

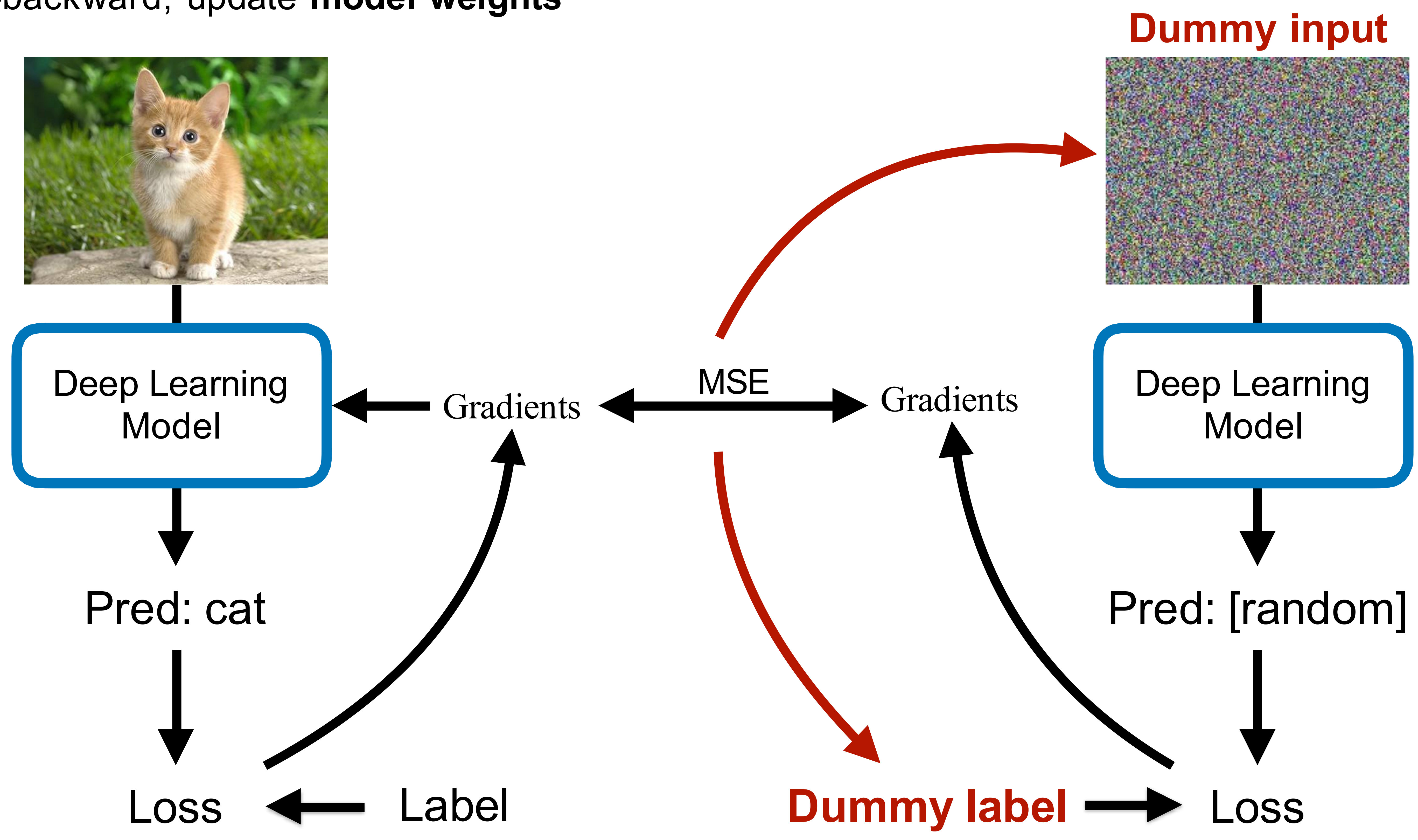
Can we obtain the **raw training data** from **gradient**?

Exploiting unintended feature leakage in collaborative learning. [Melis 2018]
Membership inference attacksagainst machine learning models. [Shokri 2016]

Deep Leakage from Gradients

Normal Training:
forward-backward, update **model weights**

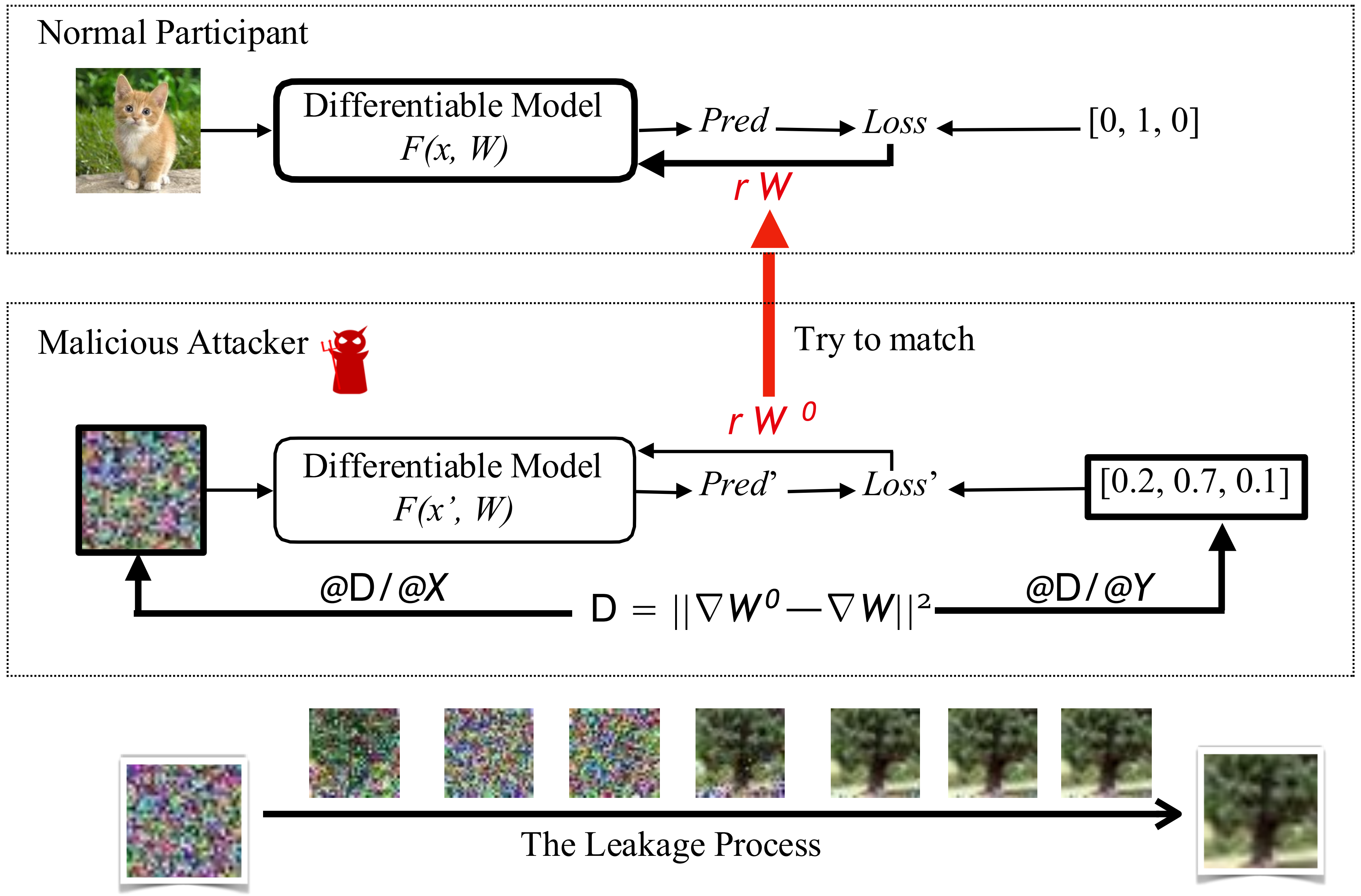
Deep Leakage Attack:
forward-backward, update **the dummy data**



Deep Leakage from Gradient. [Zhu et al, NeurIPS 2019]

Deep Leakage from Gradients

Deep Leakage Attack via Gradients Matching



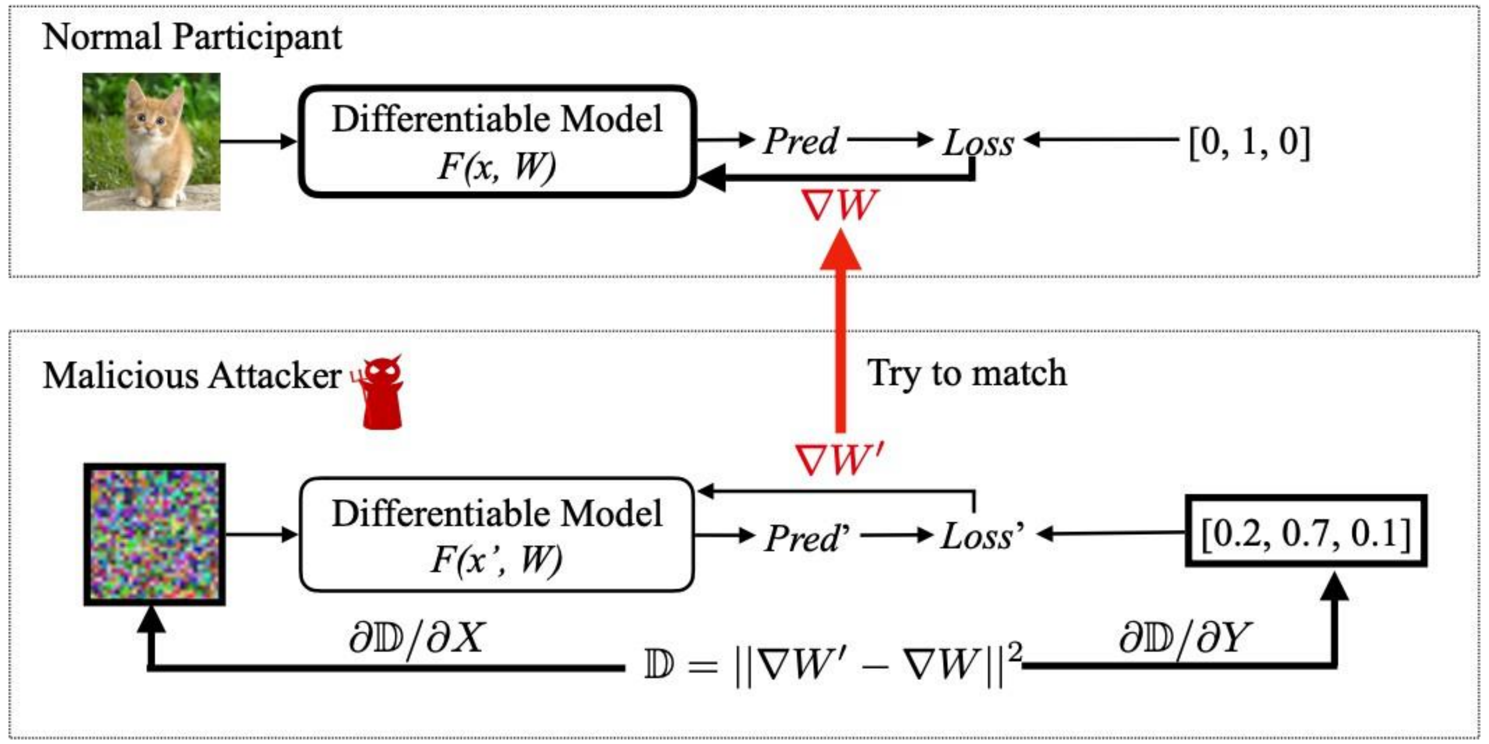
Only gradients are shared between malicious attacker and normal users.

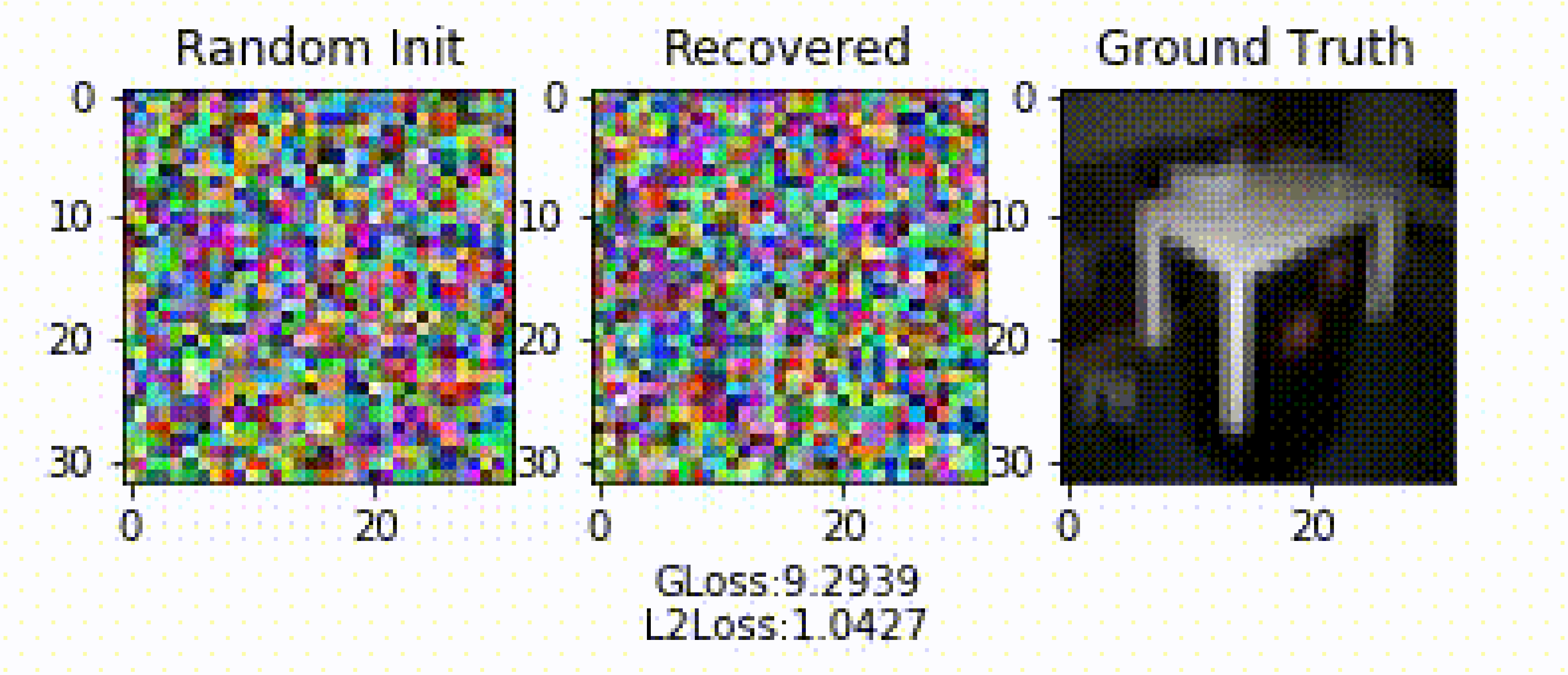
But, this action indeed **leaks the privacy**!

Deep Leakage from Gradient. [Zhu et al, NeurIPS 2019]

Deep Leakage Attack Results

Attack on Vision Model (bs=1)

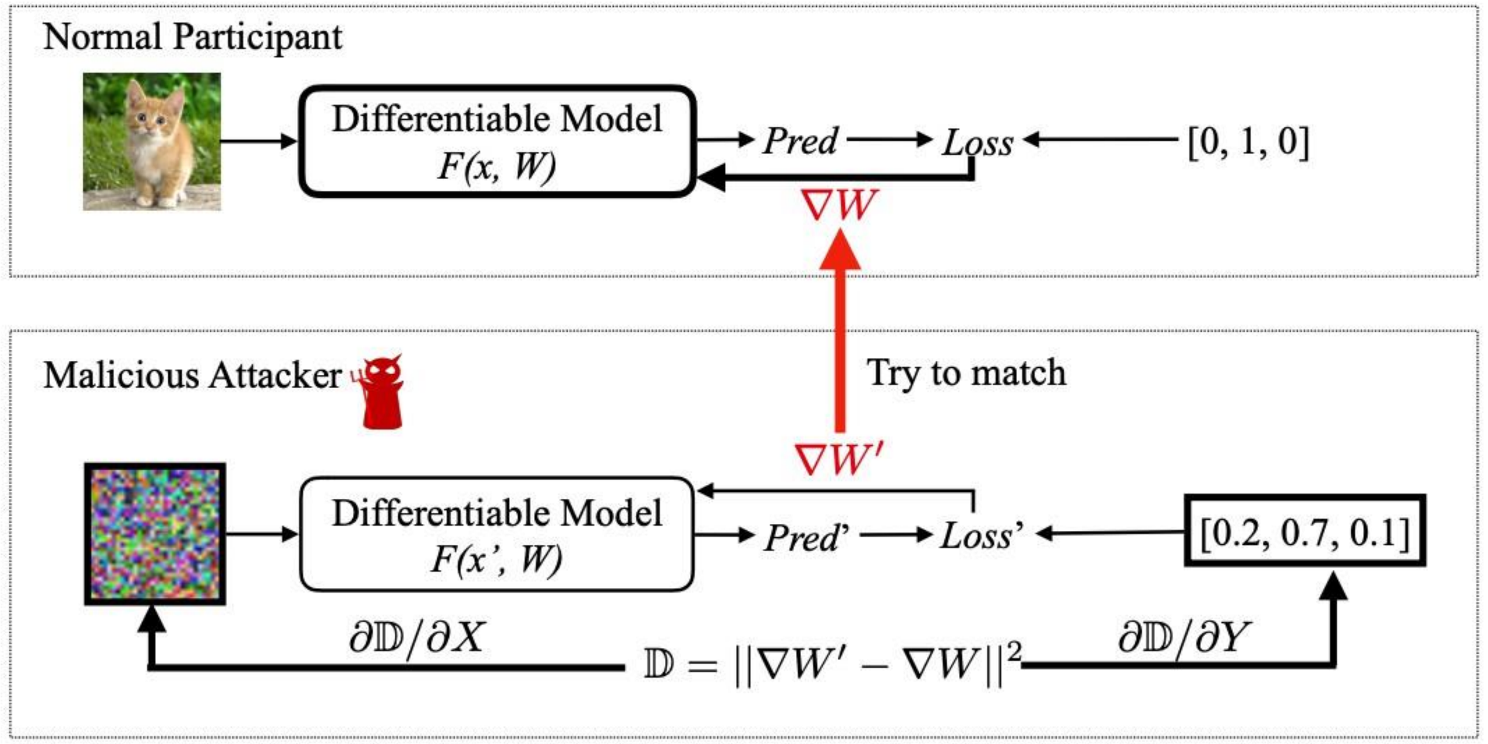


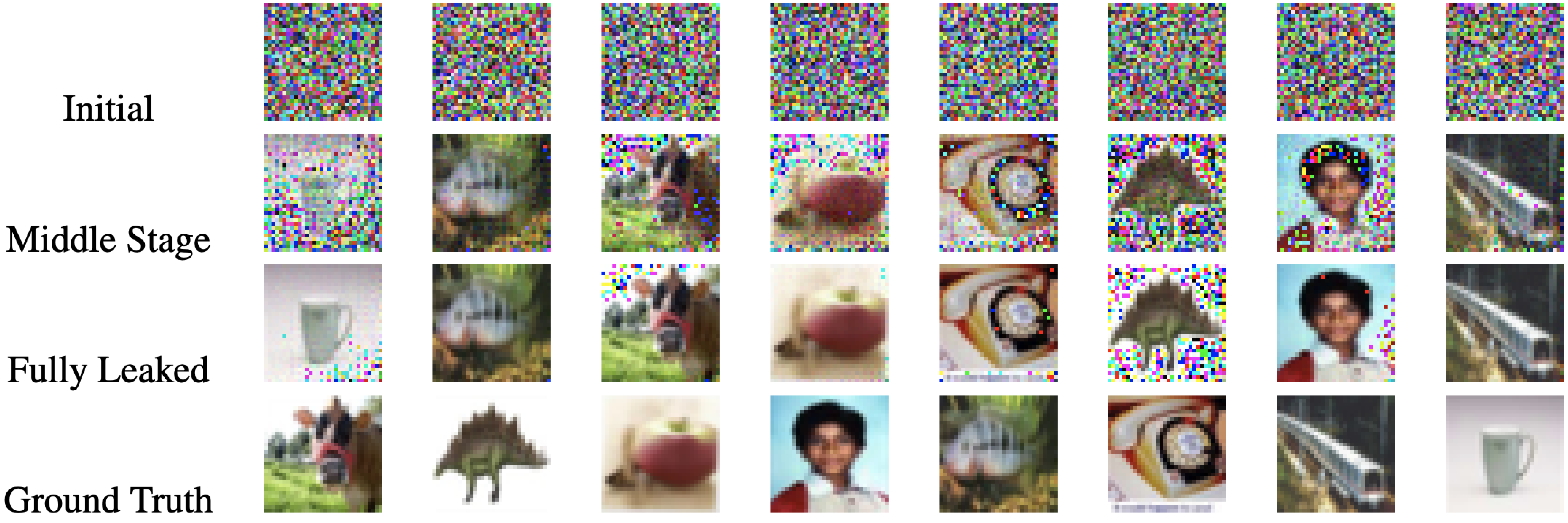


Deep Leakage from Gradient. [Zhu et al, NeurIPS 2019]

Deep Leakage Attack Results

Attack on Vision Model (bs=8)





Deep Leakage from Gradient. [Zhu et al, NeurIPS 2019]

Deep Leakage Attack Results

Attack on Language Model (BERT, Masked Language Model)

Iters=0: tilting fill given **less word **itude fine **nton overheard living vegas **vac **vation *f forte **dis cerambycidae ellison **don yards marne **kali

Iters=10: tilting fill given **less full solicitor other ligue shrill living vegas rider treatment carry played sculptures lifelong ellison net yards marne **kali

Iters=20: registration , volunteer applications , at student travel application open the ; week of played ; child care will be glare .

Iters=30: registration, volunteer applications, and student travel application open the first week of september . child care will be available

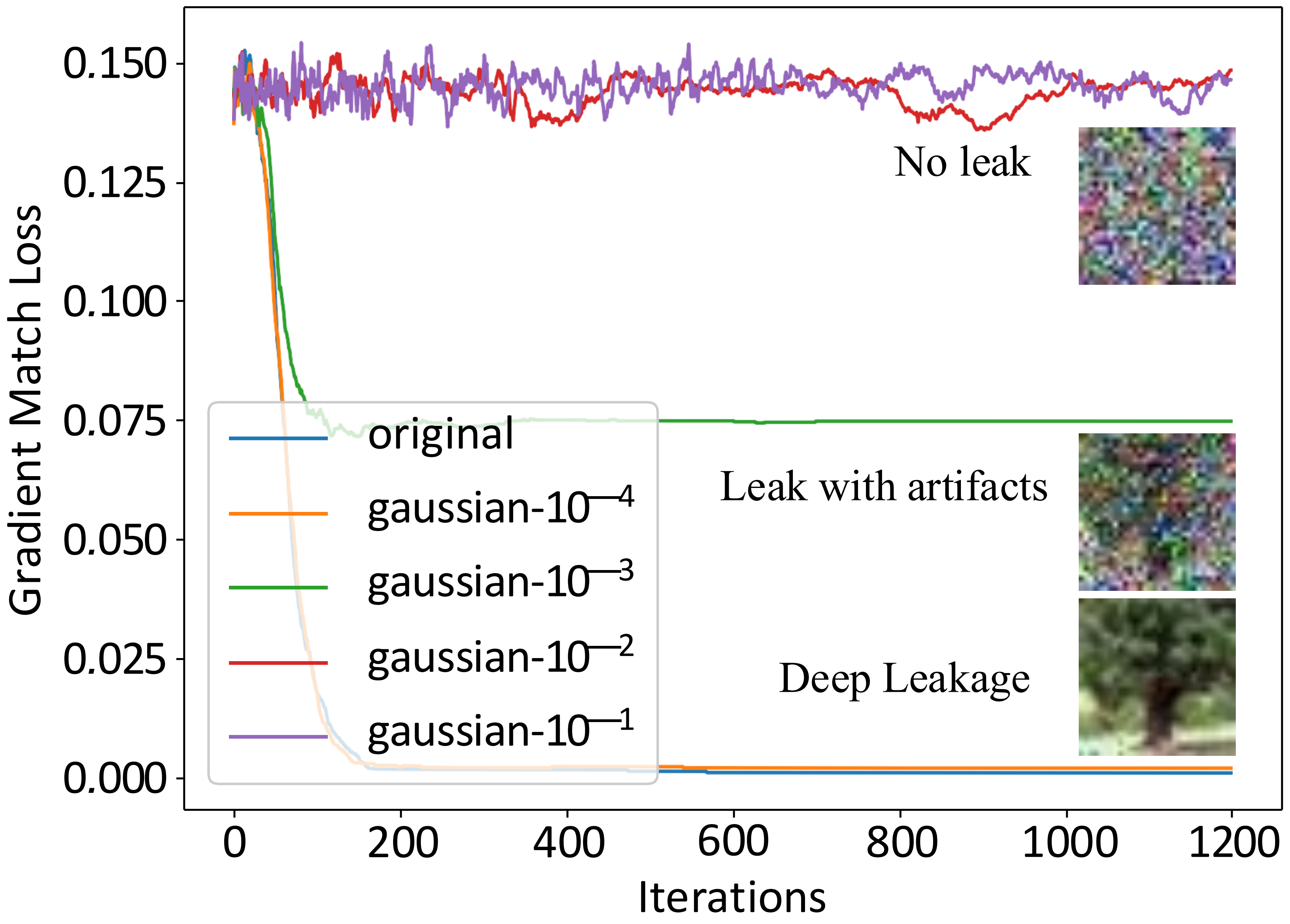
Original text: Registration, volunteer applications, and student travel application open the first week of September. Child care will be available.

Unmatched words are marked with red.

Deep Leakage from Gradient. [Zhu et al, NeurIPS 2019]

Defense Strategy for Deep Leakage

Gaussian and laplacian noise



	Original	G-10^{-4}	G-10^{-3}	G-10^{-2}	G-10^{-1}
Accuracy	76.3%	75.6%	73.3%	45.3%	≤1%
Defendability	–	✗	✗	✓	✓
		L-10^{-4}	L-10^{-3}	L-10^{-2}	L-10^{-1}
Accuracy	–	75.6%	73.4%	46.2%	≤1%
Defendability	–	✗	✗	✓	✓

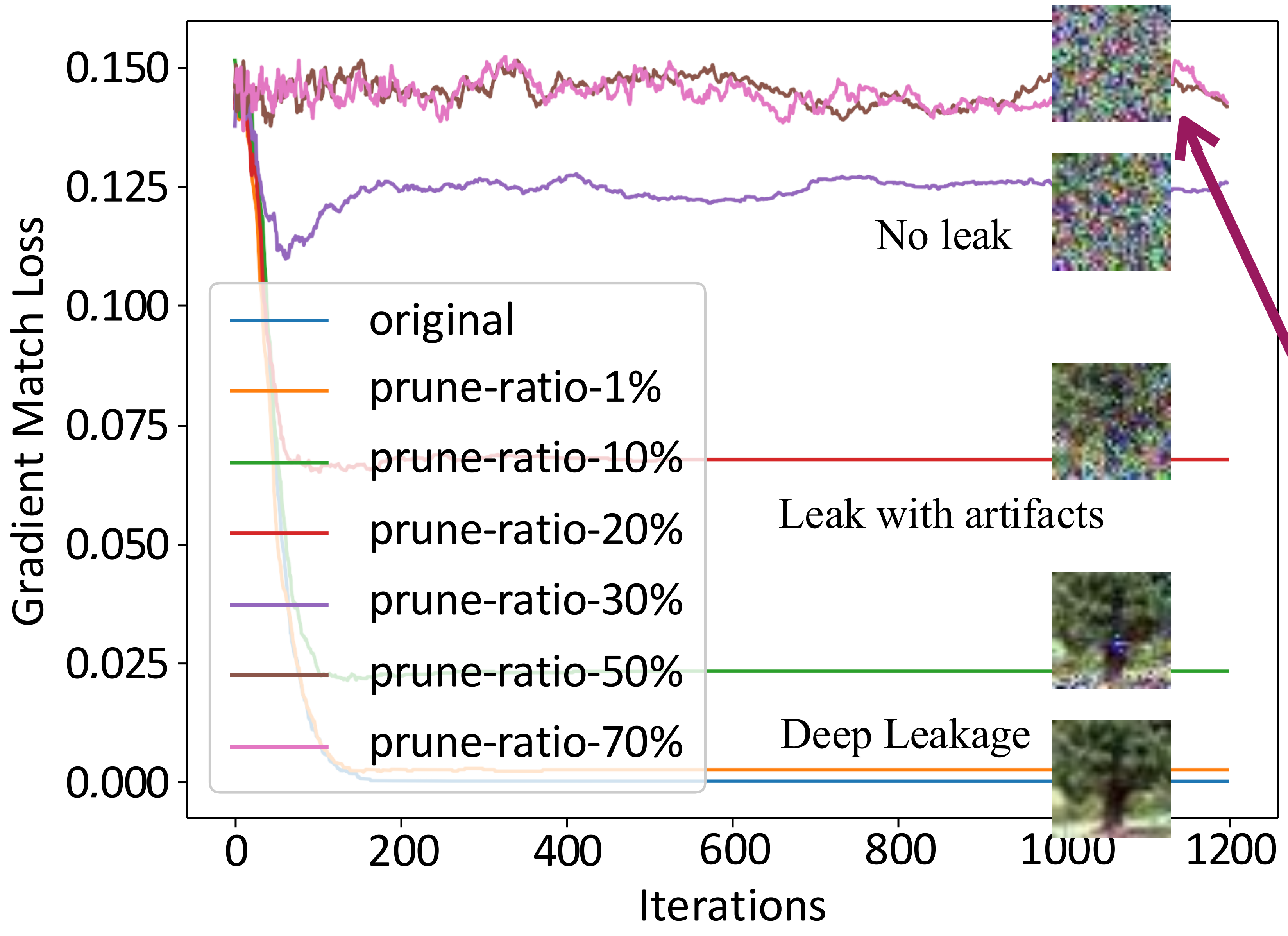
“G” denotes gaussian noise, “L” denotes laplacian noise

Simply applying noise cannot prevent deep leakage unless we allow significant accuracy drop (purple and red lines)

Deep Leakage from Gradient. [Zhu et al, NeurIPS 2019]

Defense Strategy for Deep Leakage

Gradient compression



ResNet50	Top-1	Defendablity
Prune-ratio:0%	75.96%	No
Prune-ratio:99% [2]	76.15% (+0.19%)	Yes

Gradient compression[2], **can effectively prevent** deep leakage while preserving accuracy.

Besides compression, DGC[2] further applies local accumulation to obfuscate gradients thus better protect users' privacy.

1 Deep Leakage from Gradient. [Zhu et al, NeurIPS 2019]
2 Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training [Lin et al, ICLR 2018]

Lecture Plan

1. Federated learning and the deep leakage from gradients
2. **Pruning, quantization and knowledge distillation**
3. Memory bottleneck of on-device training
4. Tiny transfer learning (TinyTL)
5. Sparse back-propagation (SparseBP)

Neural Network Pruning

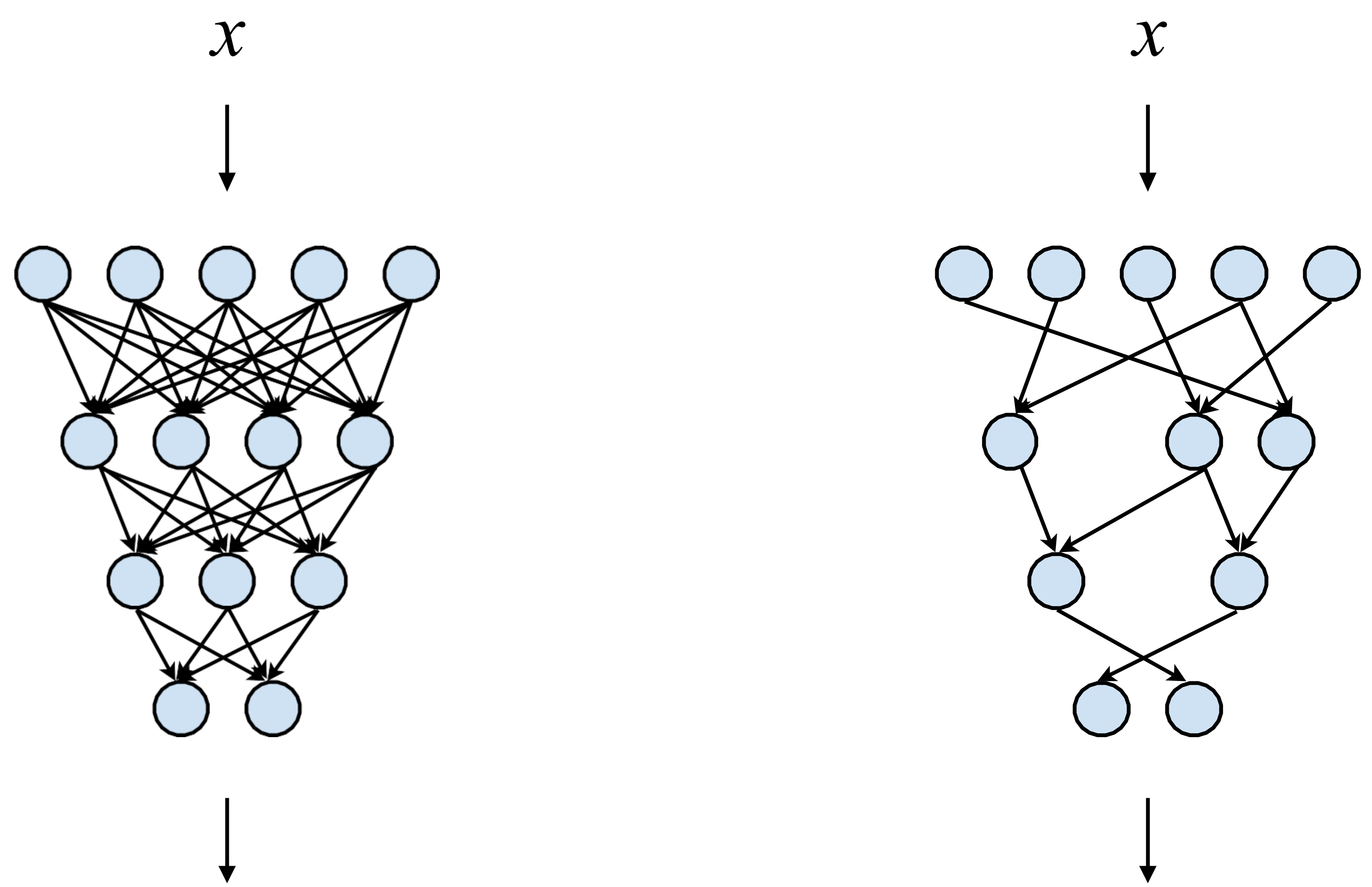
- In general, we could formulate the pruning as follows:

$$\arg\min_{\mathbf{W}_P} L(\mathbf{x}; \mathbf{W}_P)$$

subject to

$$\|\mathbf{W}_p\|_0 < N$$

- L represents the objective function for neural network training;
- $\mathbf{x}$ is input, $\mathbf{W}$ is original weights, $\mathbf{W}_P$ is pruned weights;
- $\|\mathbf{W}_p\|_0$ calculates the #nonzeros in W_P, and N is the target #nonzeros.

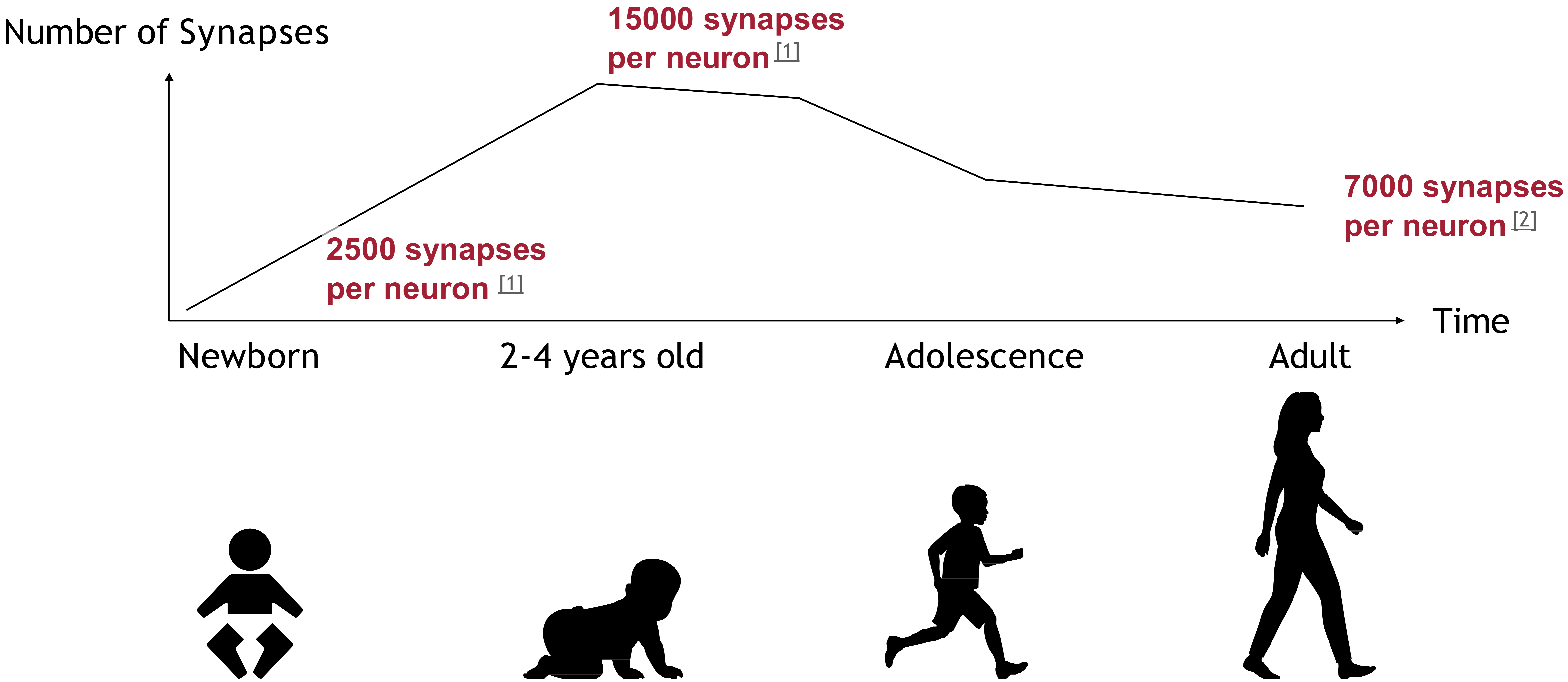


$$\arg\min_{\mathbf{W}} L(\mathbf{x}; \mathbf{W})$$

$$\arg\min_{\mathbf{W}_P} L(\mathbf{x}; \mathbf{W}_P)$$

$$s.t. \|\mathbf{W}_P\|_0 \leq N$$

Pruning Happens in Human Brain

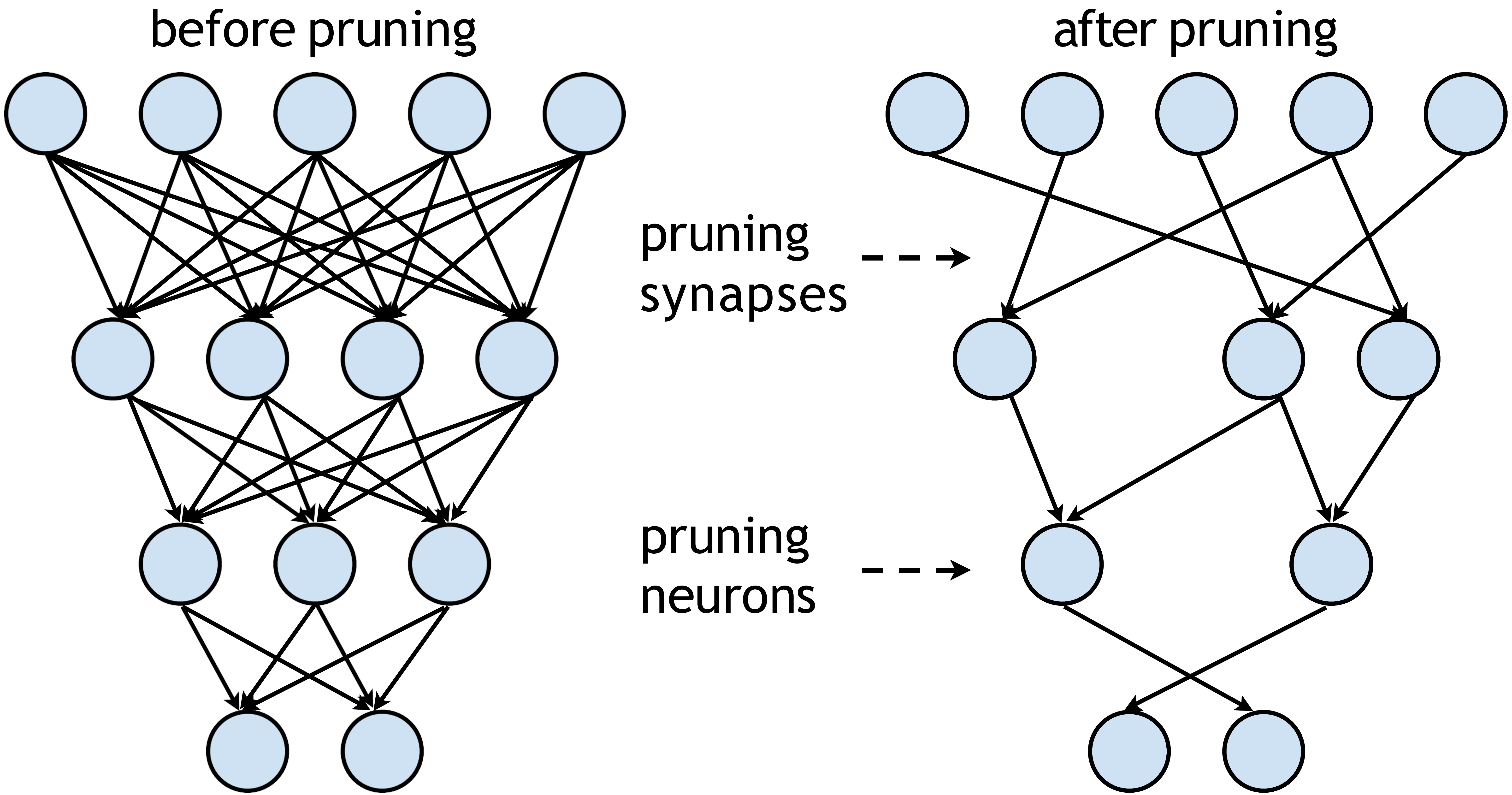


Do We Have Brain to Spare? [Drachman DA, Neurology 2004]
Peter Huttenlocher (1931–2013) [Walsh, C. A., Nature 2013]

Data Source: 1, 2
Slide Inspiration: Alila Medical Media

Neural Network Pruning

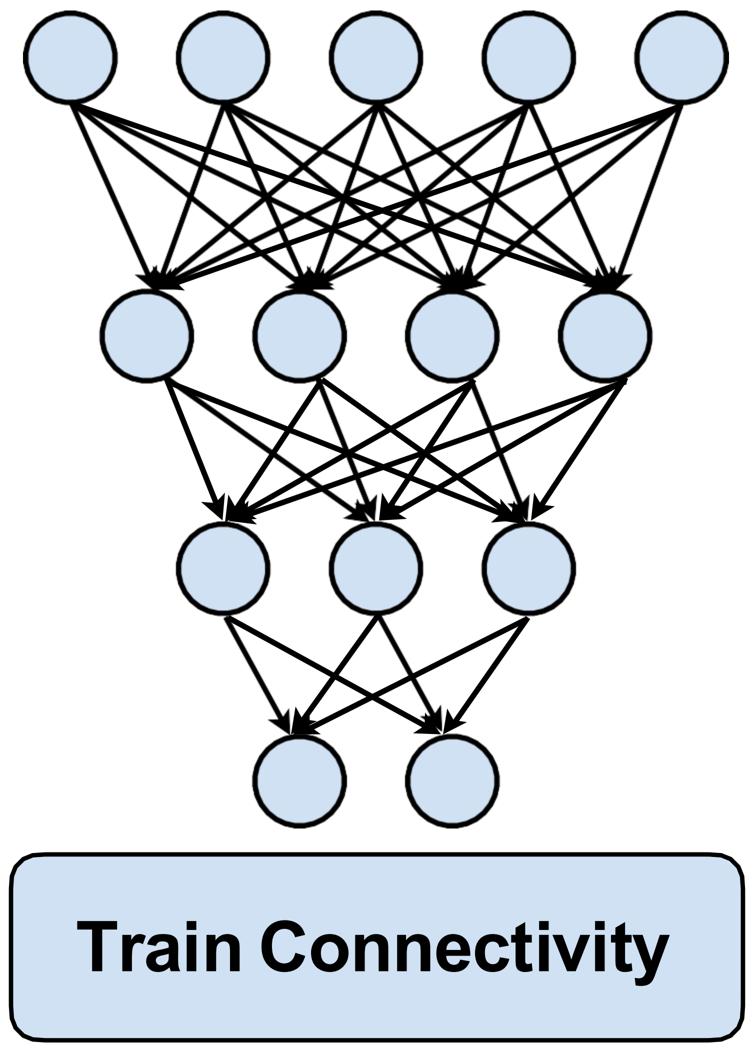
Make neural network smaller by removing synapses and neurons

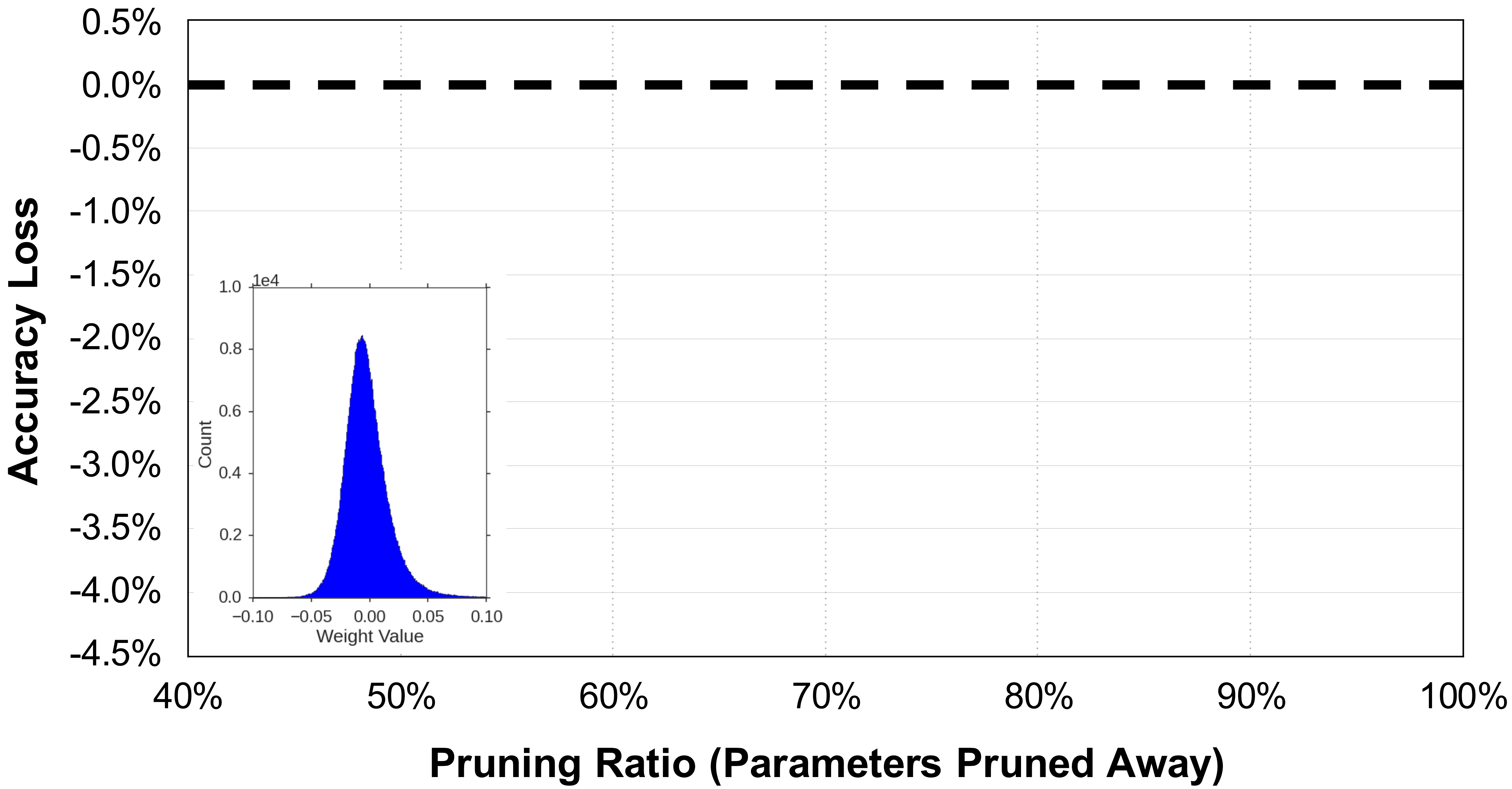


Optimal Brain Damage [LeCun *et al.*, NeurIPS 1989]
Learning Both Weights and Connections for Efficient Neural Network [Han *et al.*, NeurIPS 2015]

Neural Network Pruning

Make neural network smaller by removing synapses and neurons

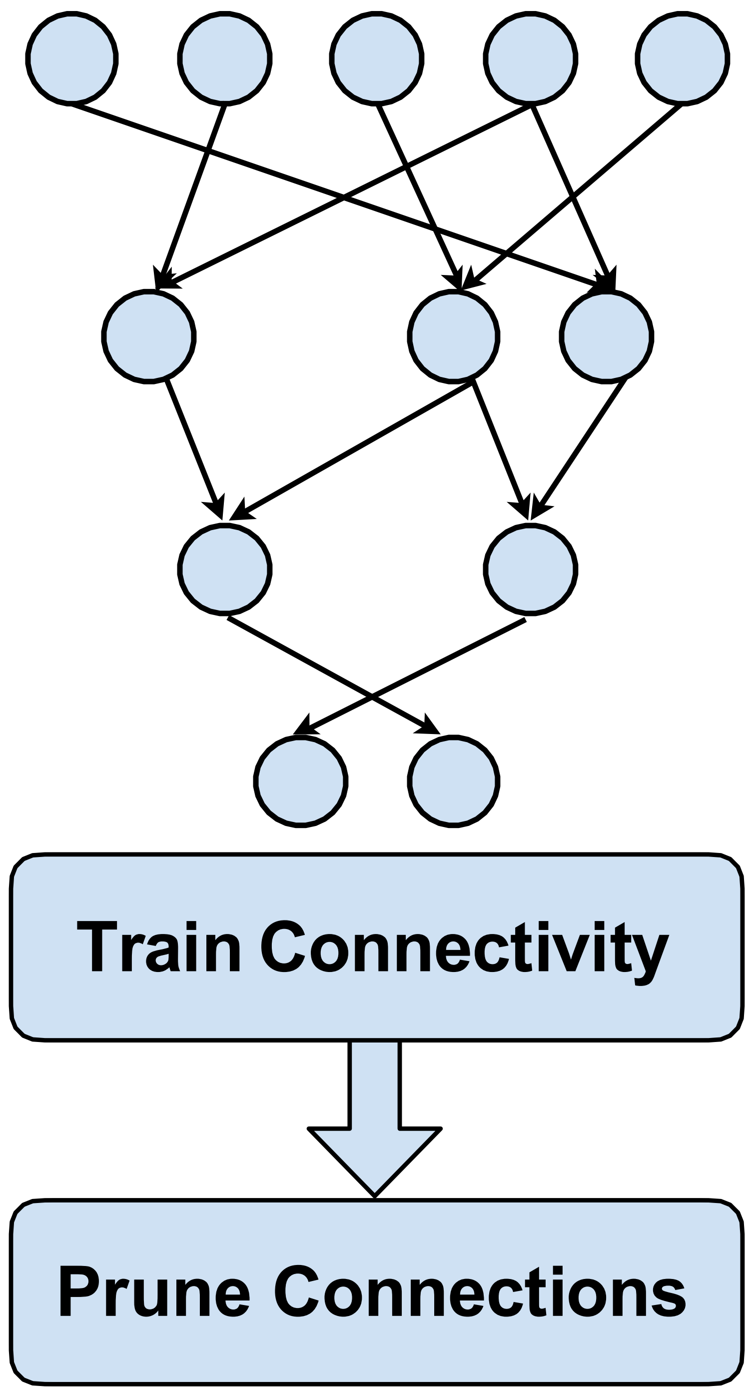


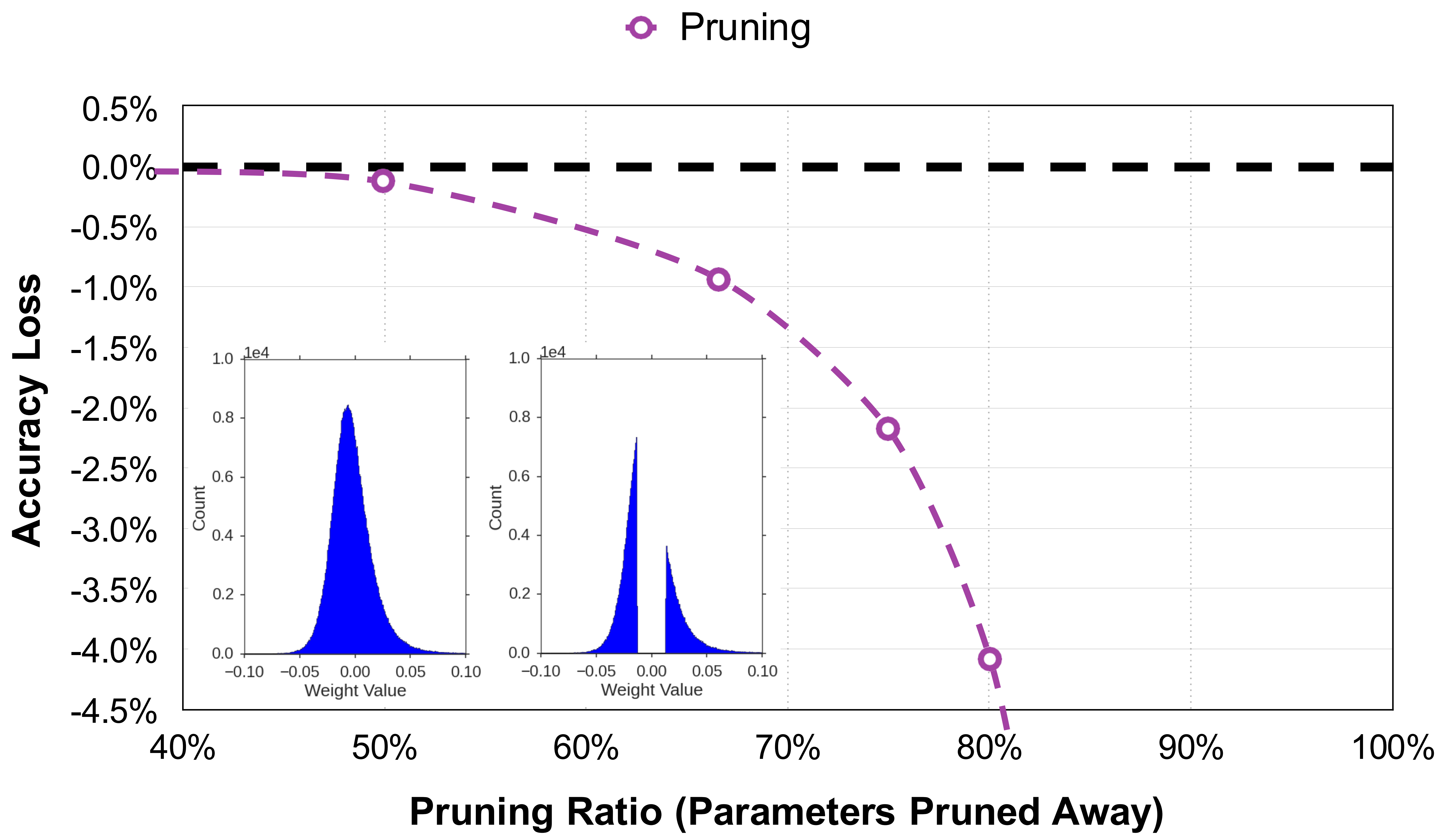


Learning Both Weights and Connections for Efficient Neural Network [Han *et al.*, NeurIPS 2015]

Neural Network Pruning

Make neural network smaller by removing synapses and neurons

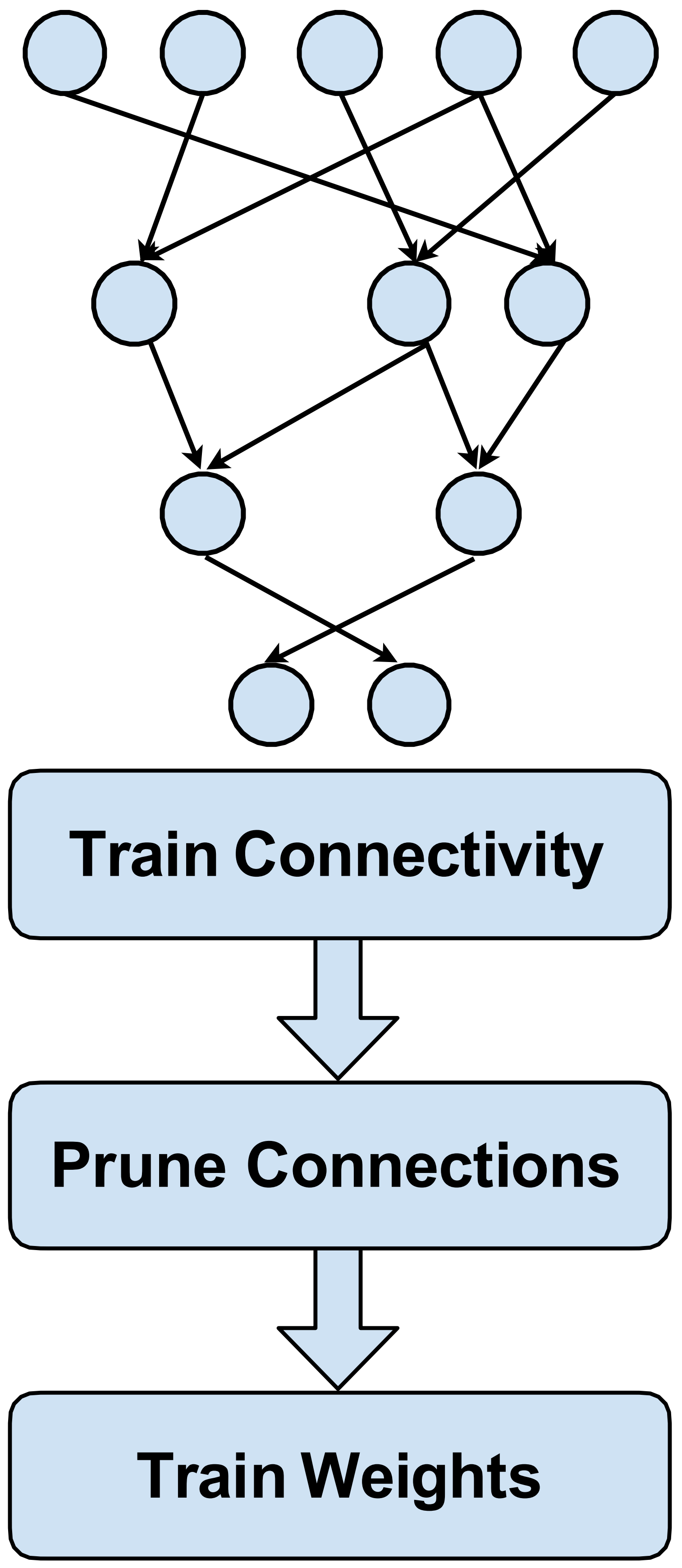


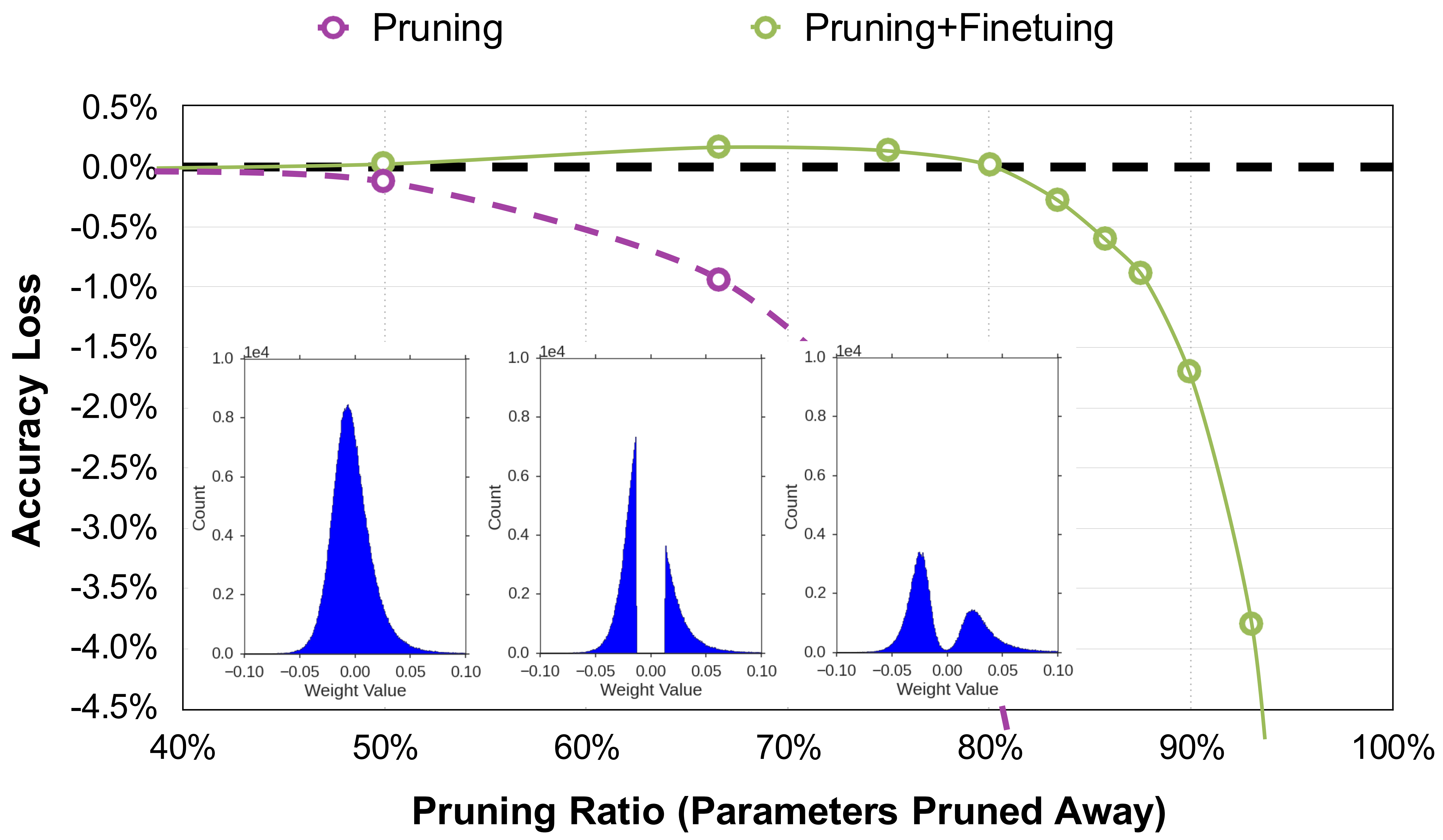


Learning Both Weights and Connections for Efficient Neural Network [Han *et al.*, NeurIPS 2015]

Neural Network Pruning

Make neural network smaller by removing synapses and neurons

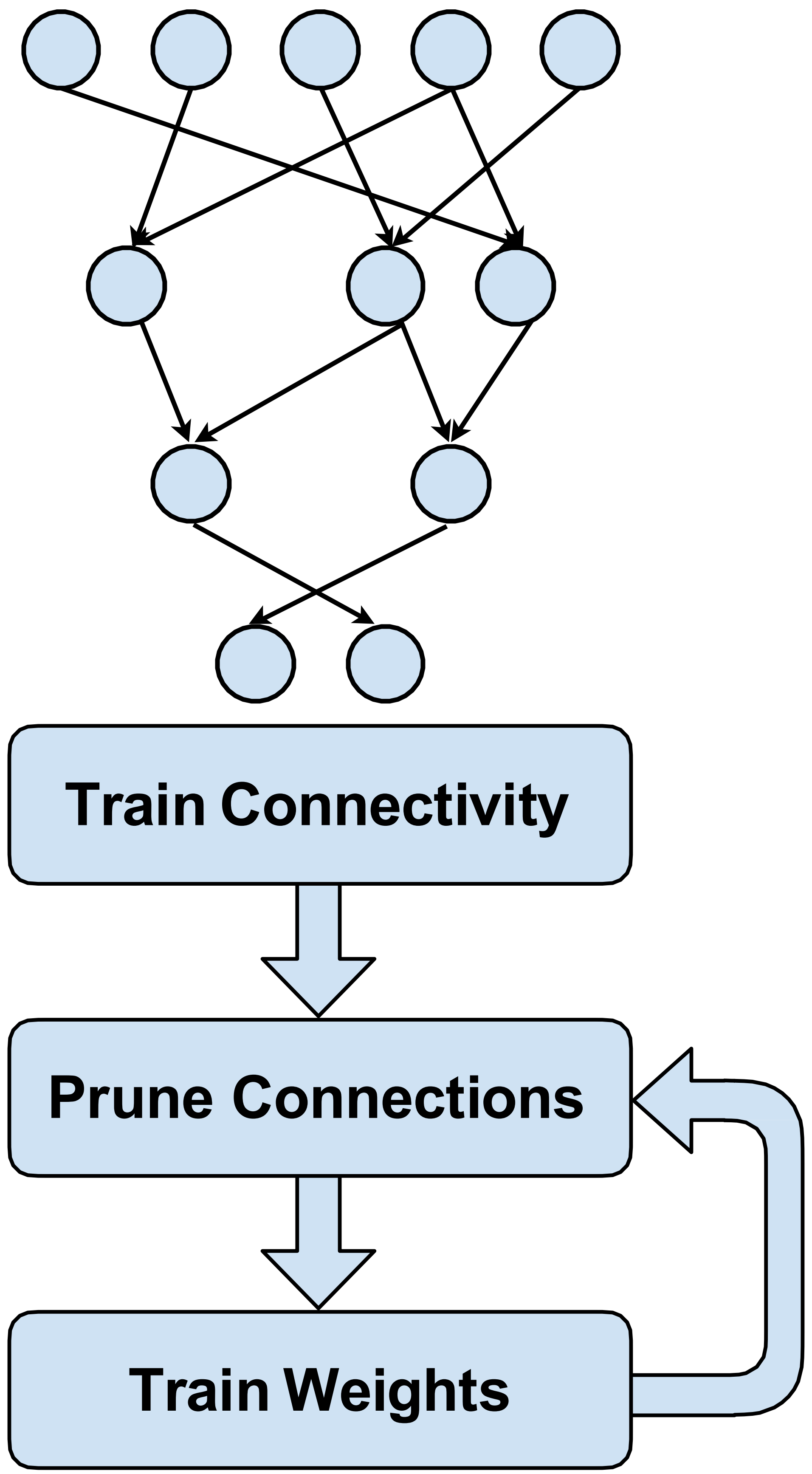


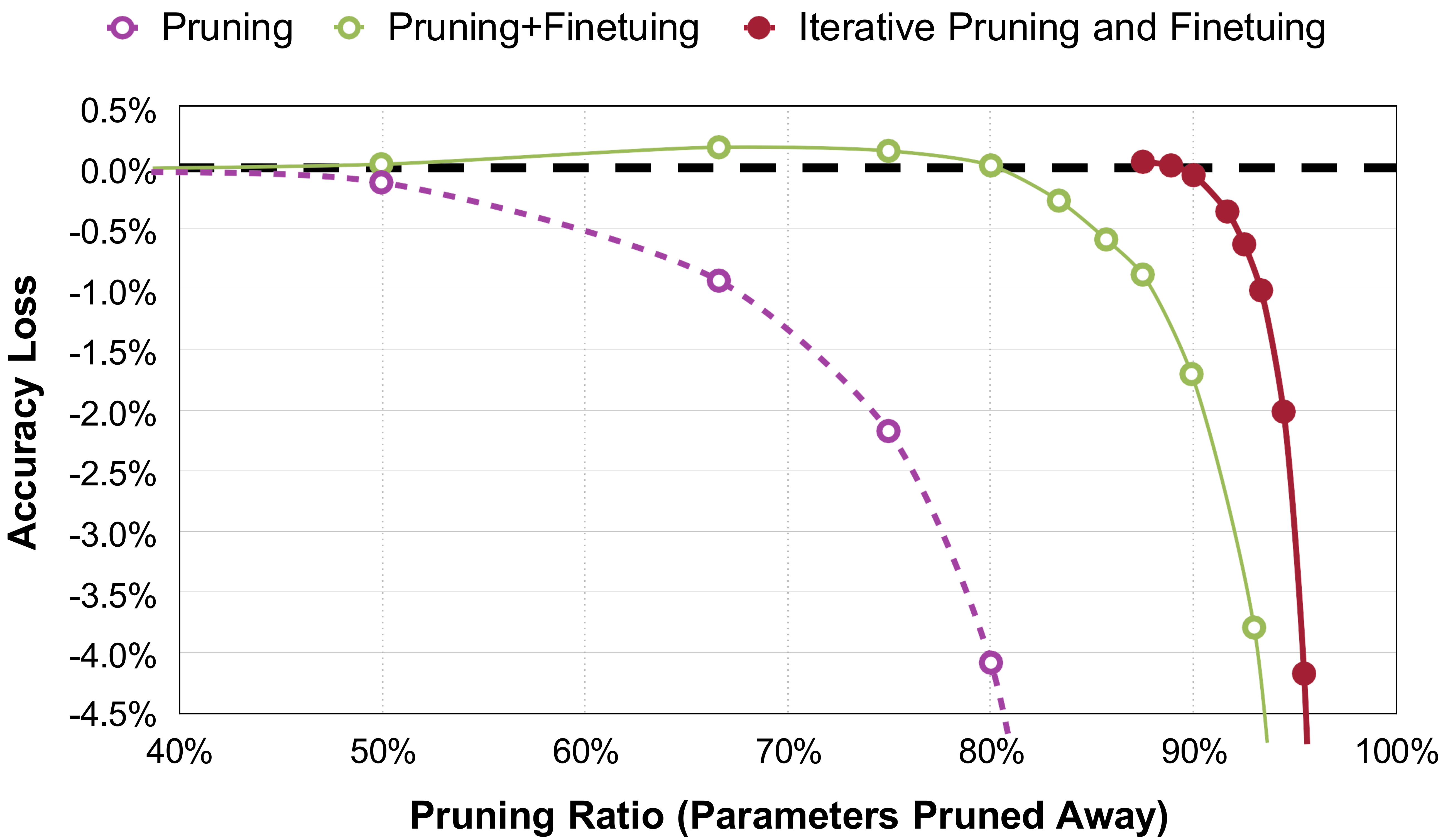


Learning Both Weights and Connections for Efficient Neural Network [Han *et al.*, NeurIPS 2015]

Neural Network Pruning

Make neural network smaller by removing synapses and neurons





Learning Both Weights and Connections for Efficient Neural Network [Han *et al.*, NeurIPS 2015]

Neural Network Pruning

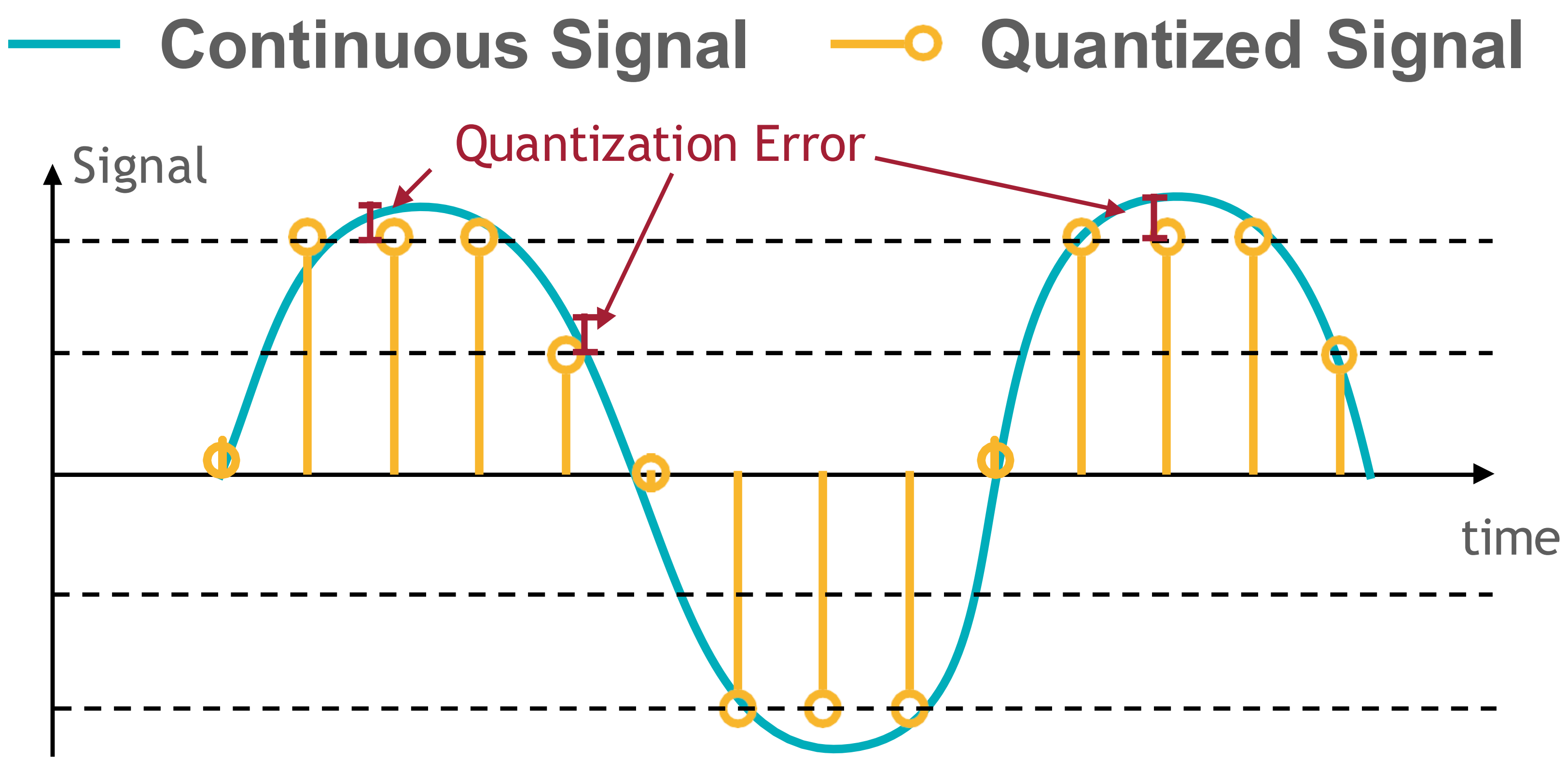
Make neural network smaller by removing synapses and neurons

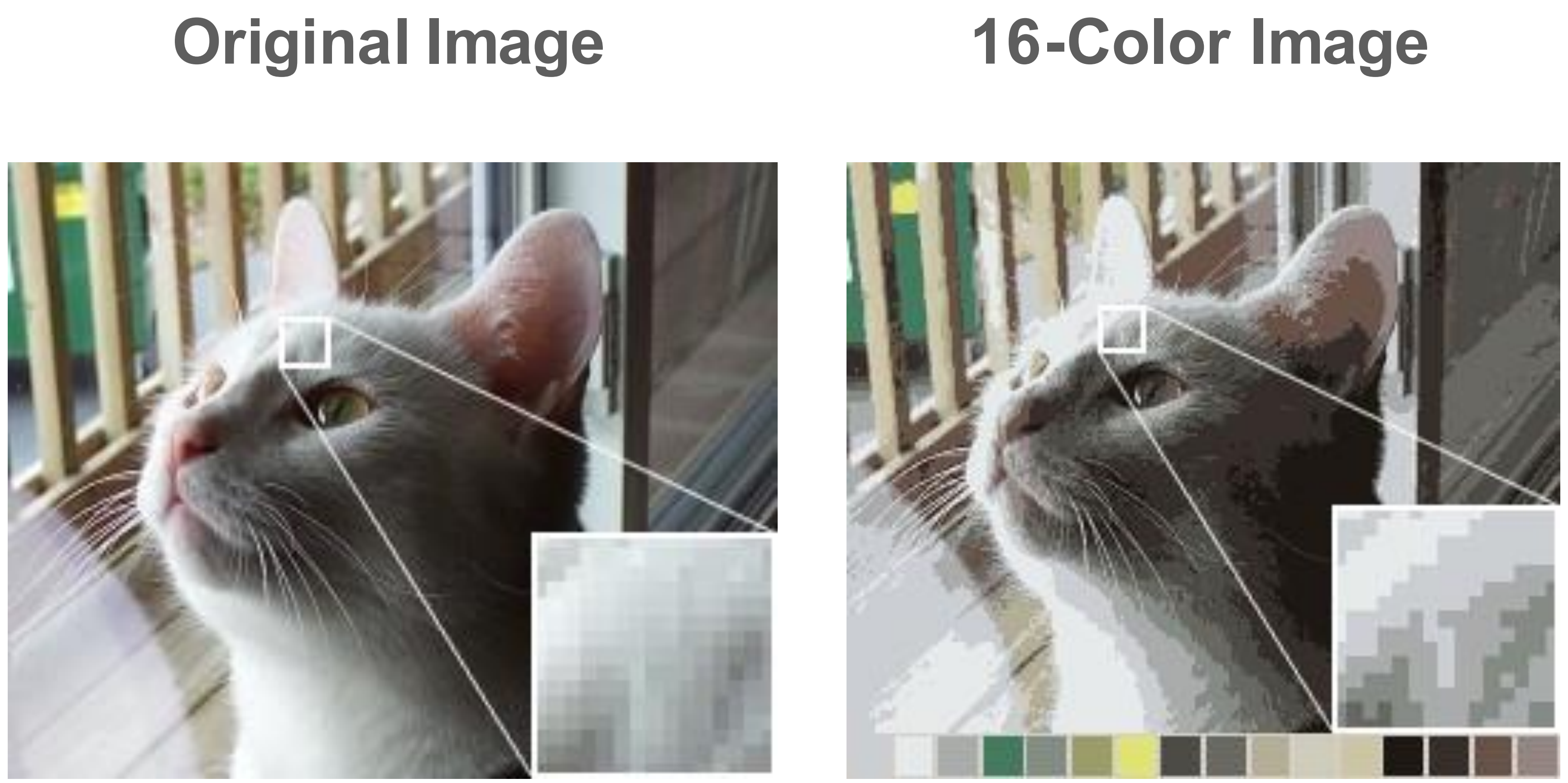
Neural Network	#Parameters		
	Before Pruning	After Pruning	Reduction
AlexNet	61 M	6.7 M	**9 ×**
VGG-16	138 M	10.3 M	**12 ×**
GoogleNet	7 M	2.0 M	**3.5 ×**
ResNet50	26 M	7.47 M	**3.4 ×**
SqueezeNet	1 M	0.38 M	**3.2 ×**

Efficient Methods and Hardware for Deep Learning [Han S., Stanford University]

What is Quantization?

Quantization is the process of constraining an input from a continuous or otherwise large set of values to a discrete set.





The difference between an input value and its quantized value is referred to as quantization error.

Quantization [Wikipedia]

Neural Network Quantization

Weight Quantization

weights
(32-bit float)

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

Neural Network Quantization

Weight Quantization

weights
(32-bit float)

2.09	*-0.98*	*1.48*	*0.09*
0.05	*-0.14*	*-1.08*	*2.12*
-0.91	*1.92*	*0*	*-1.03*
1.87	*0*	*1.53*	*1.49*

~~2.09, 2.12, 1.92, 1.87~~

2.0

K-Means-based Weight Quantization

weights
(32-bit float)

2.09	*-0.98*	*1.48*	*0.09*
0.05	*-0.14*	*-1.08*	*2.12*
-0.91	*1.92*	*0*	*-1.03*
1.87	*0*	*1.53*	*1.49*

Deep Compression [Han *et al.*, ICLR 2016]

K-Means-based Weight Quantization

weights (32-bit float)

2.09	*-0.98*	*1.48*	*0.09*
0.05	*-0.14*	*-1.08*	*2.12*
-0.91	*1.92*	*0*	*-1.03*
1.87	*0*	*1.53*	*1.49*

cluster ⇨

cluster index (2-bit int)

3	0	2	1
1	1	0	3
0	3	1	0
3	1	2	2

indexes

centroids

3:	*2.00*
2:	*1.50*
1:	*0.00*
0:	*-1.00*

codebook

reconstructed weights (32-bit float)

2.00	*-1.00*	*1.50*	*0.00*
0.00	*0.00*	*-1.00*	*2.00*
-1.00	*2.00*	*0.00*	*-1.00*
2.00	*0.00*	*1.50*	*1.50*

quantization error

0.09	*0.02*	*-0.02*	*0.09*
0.05	*-0.14*	*-0.08*	*0.12*
0.09	*-0.08*	*0*	*-0.03*
-0.13	*0*	*0.03*	*-0.01*

storage

32 bit × 16
= 512 bit = 64 B

2 bit × 16
= 32 bit = 4 B **+** 32 bit × 4
= 128 bit = 16 B **=** 20 B

3.2 × smaller

Assume N-bit quantization, and #parameters = M >> 2^N.

32 bit × M
= $32M$ bit

N bit × M
= NM bit

~~32 bit × 2^N~~
~~= 2^{N+5} bit~~

$32/N$ × smaller

Deep Compression [Han *et al.*, ICLR 2016]

K-Means-based Weight Quantization

Fine-tuning Quantized Weights

weights
(32-bit float)

2.09	-0.98	1.48	0.09
0.05	-0.14	-1.08	2.12
-0.91	1.92	0	-1.03
1.87	0	1.53	1.49

cluster

cluster index
(2-bit int)

3	0	2	1
1	1	0	3
0	3	1	0
3	1	2	2

centroids

3:	2.00
2:	1.50
1:	0.00
0:	-1.00

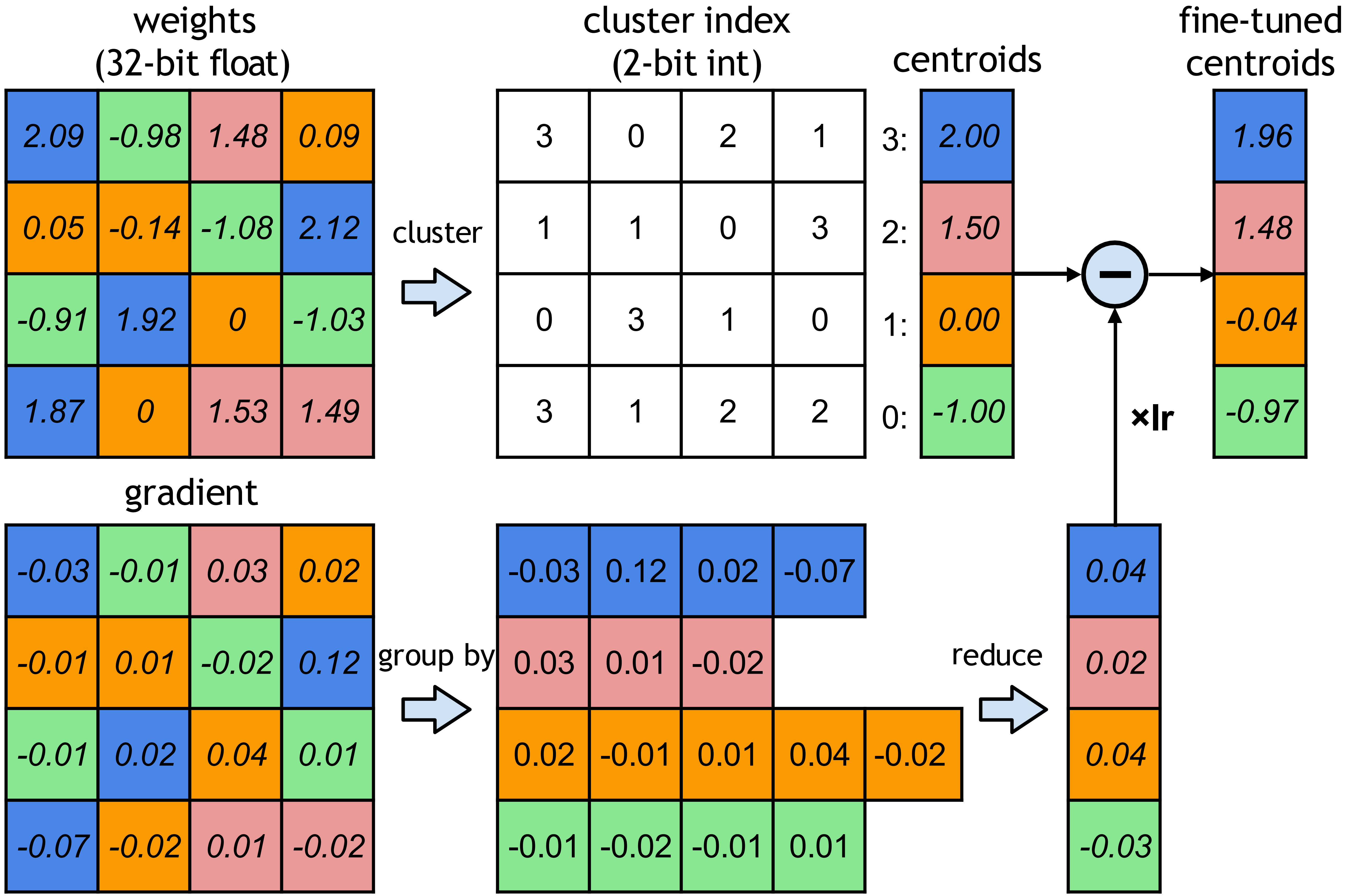
gradient

-0.03	-0.01	0.03	0.02
-0.01	0.01	-0.02	0.12
-0.01	0.02	0.04	0.01
-0.07	-0.02	0.01	-0.02

Deep Compression [Han *et al.*, ICLR 2016]

K-Means-based Weight Quantization

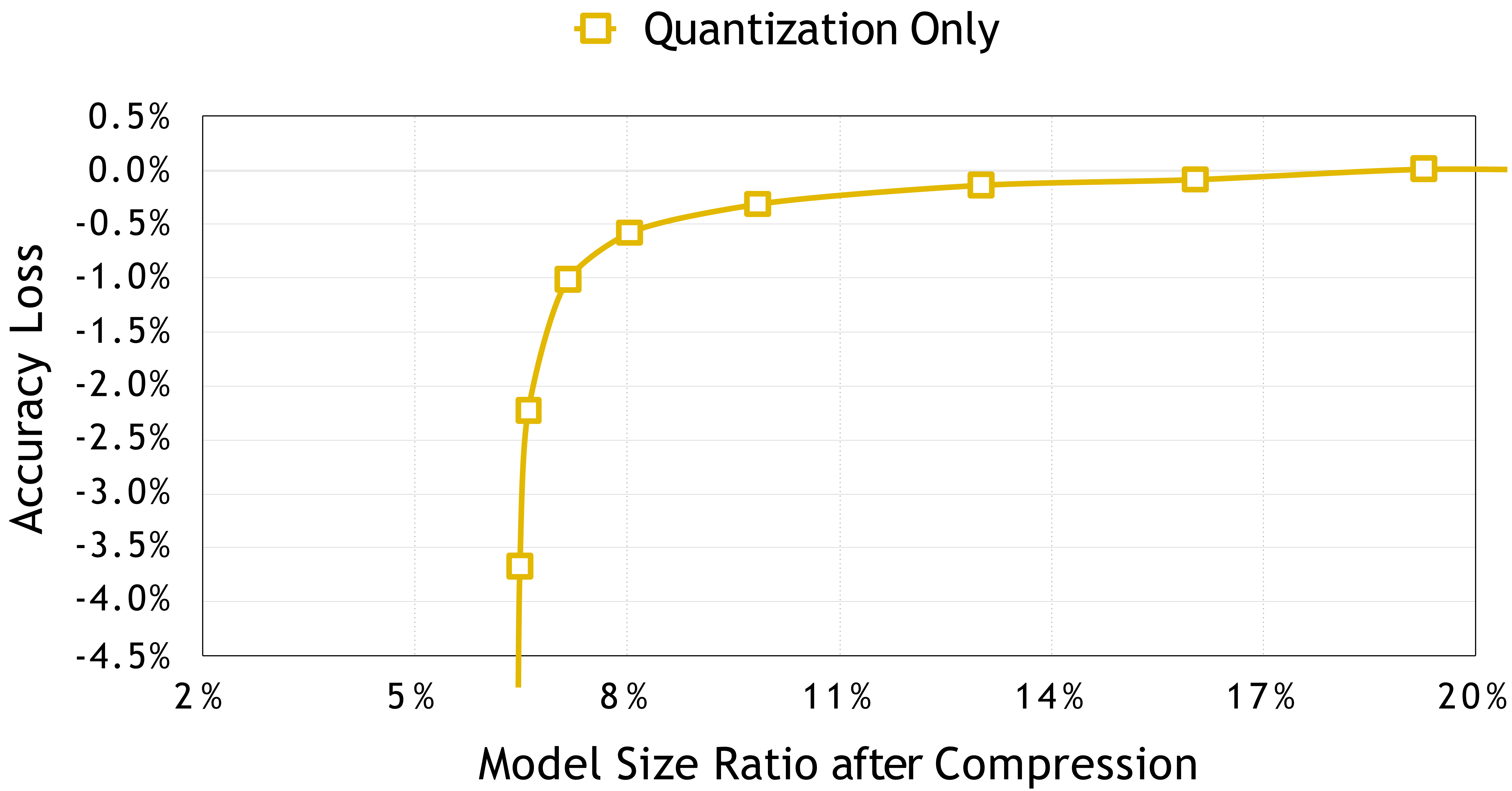
Fine-tuning Quantized Weights



Deep Compression [Han *et al.*, ICLR 2016]

K-Means-based Weight Quantization

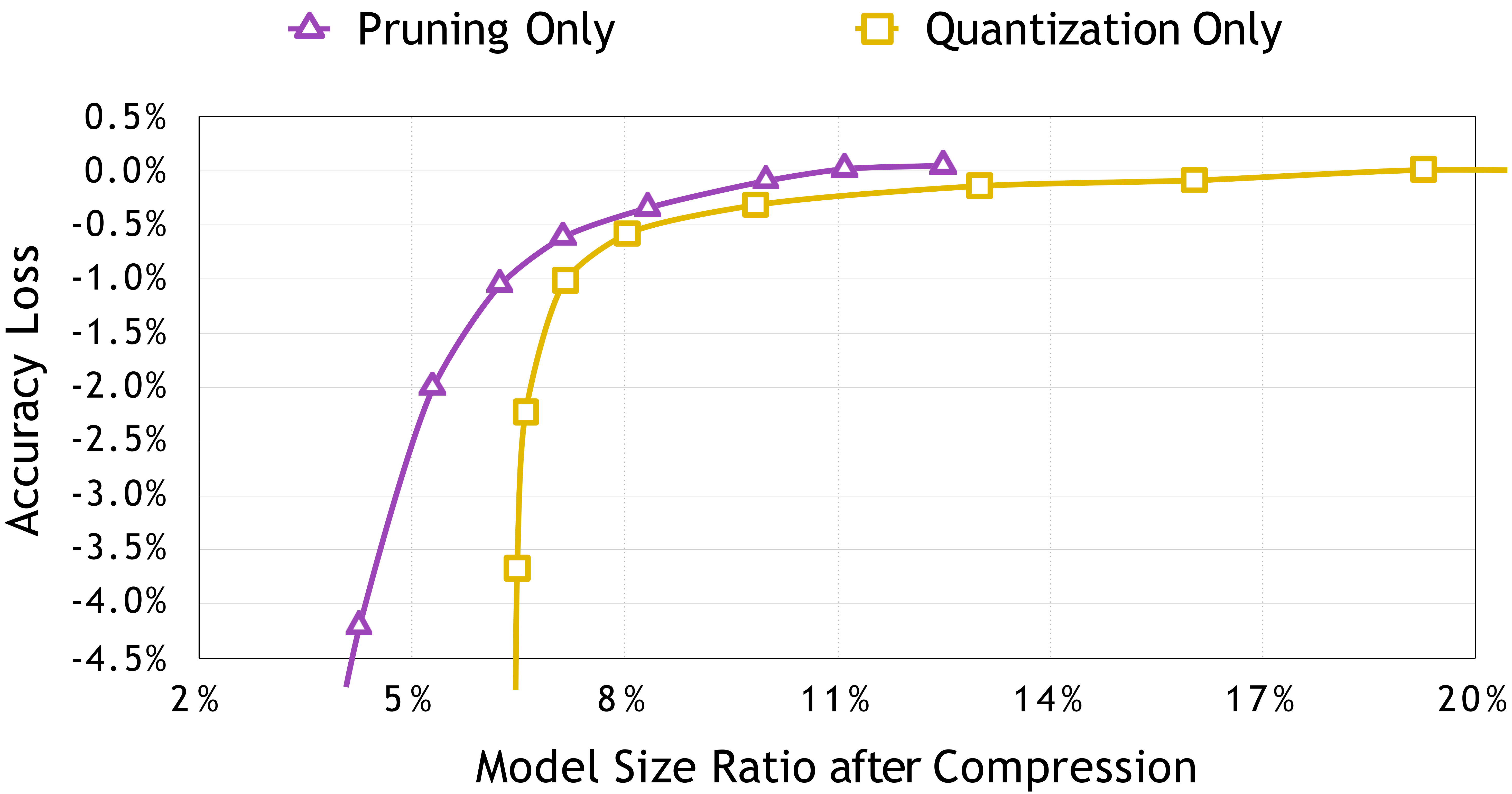
Accuracy vs. compression rate for AlexNet on ImageNet dataset



Deep Compression [Han *et al.*, ICLR 2016]

K-Means-based Weight Quantization

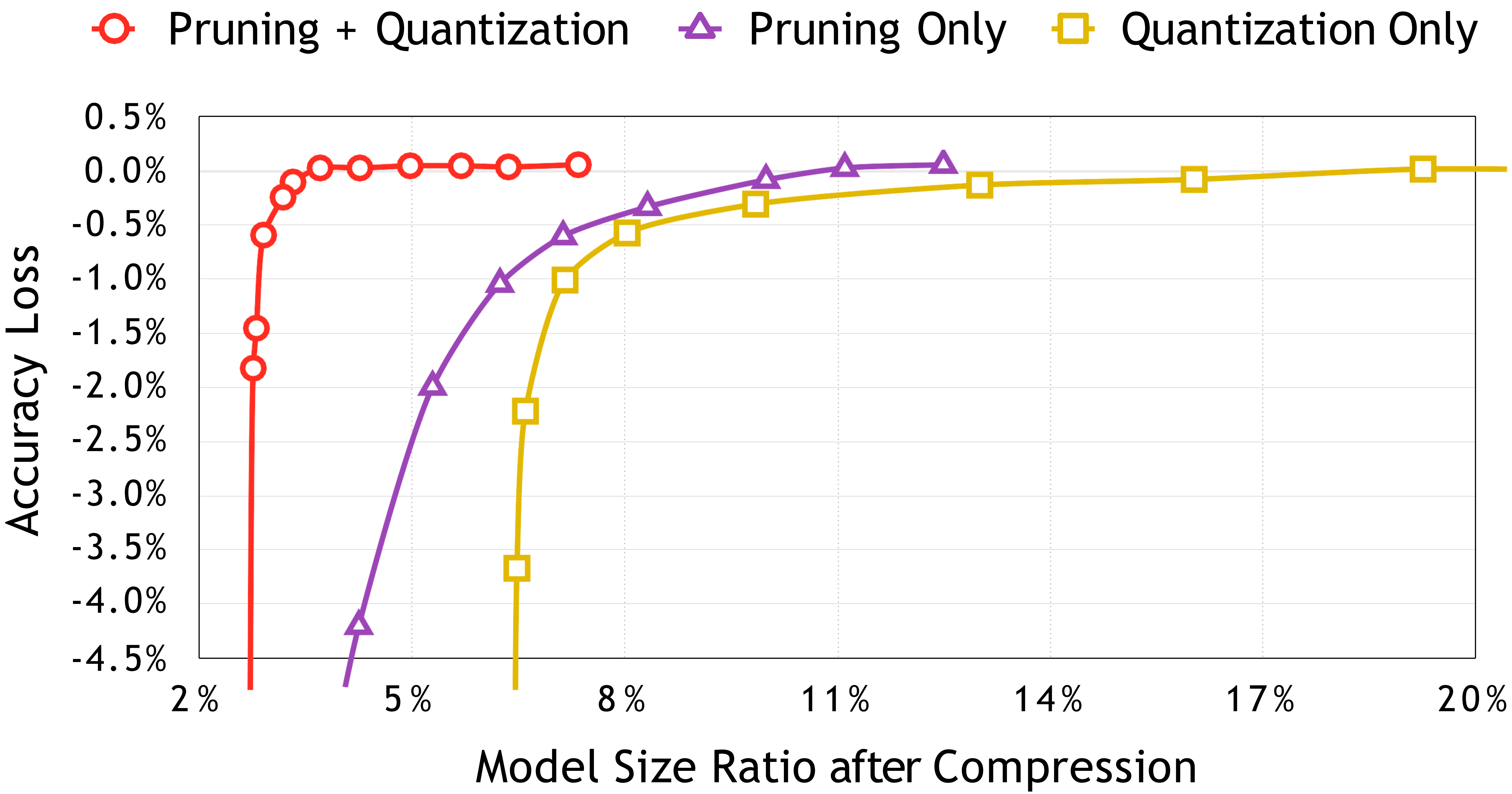
Accuracy vs. compression rate for AlexNet on ImageNet dataset



Deep Compression [Han *et al.*, ICLR 2016]

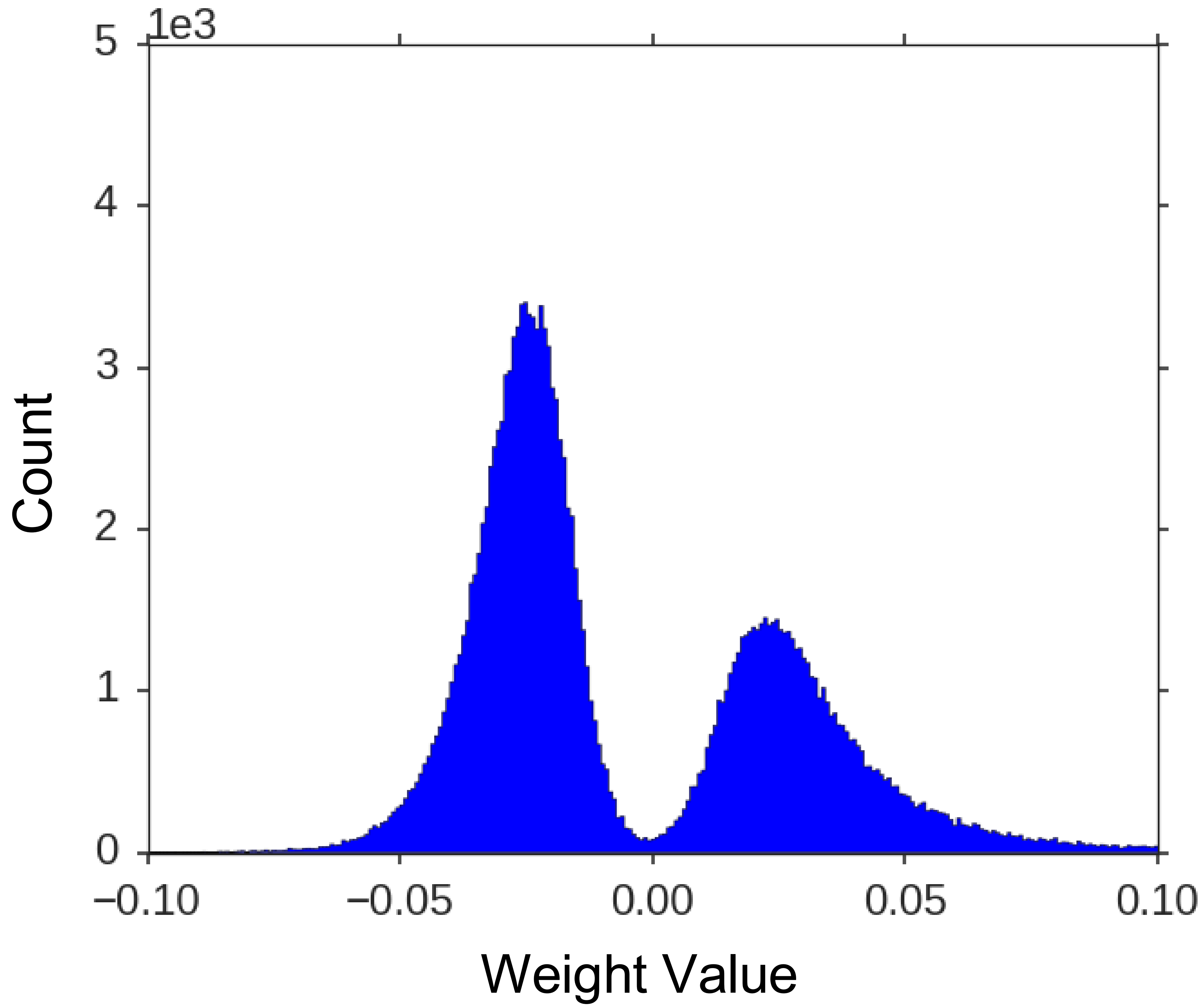
K-Means-based Weight Quantization

Accuracy vs. compression rate for AlexNet on ImageNet dataset



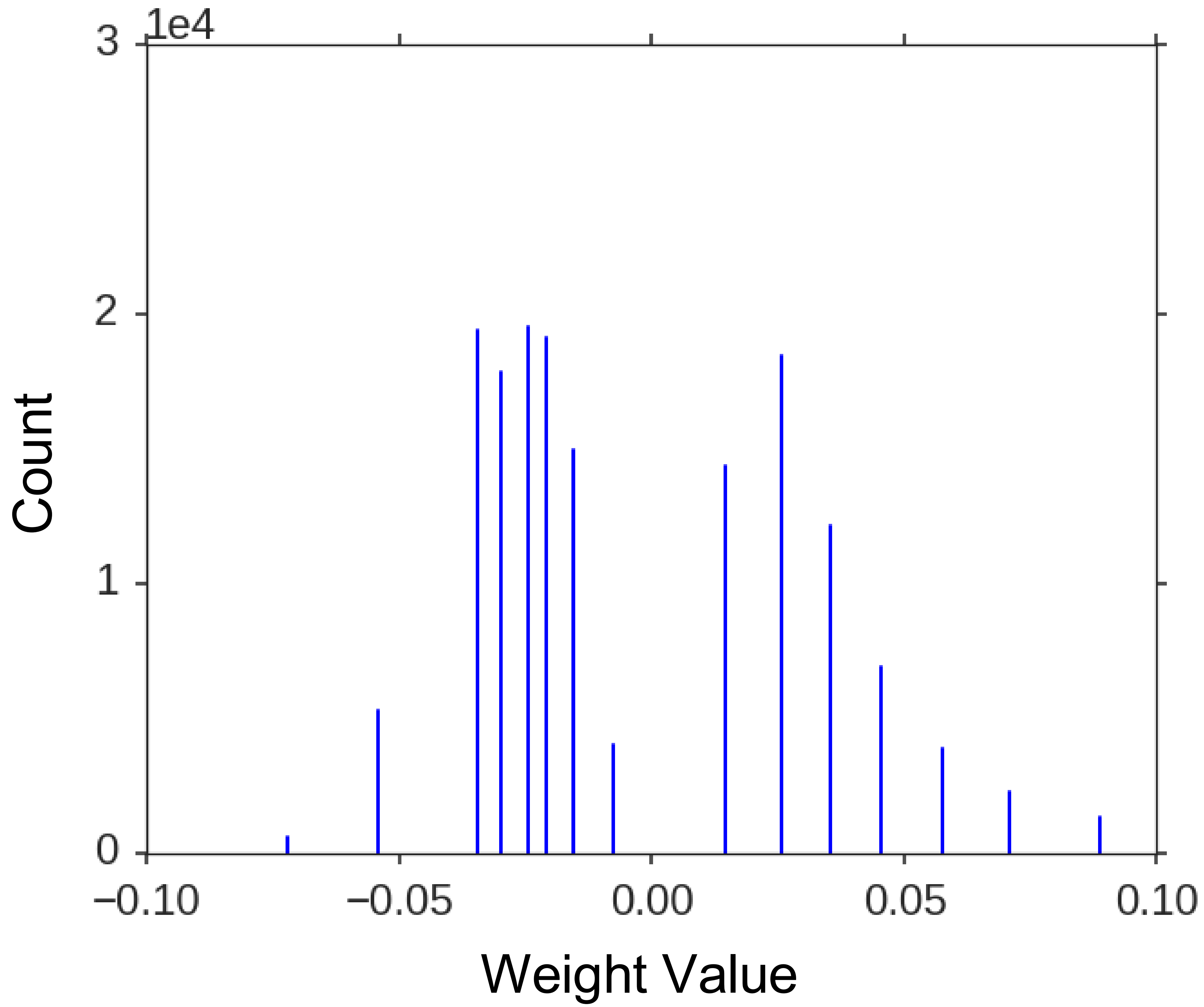
Deep Compression [Han *et al.*, ICLR 2016]

Before Quantization: Continuous Weight



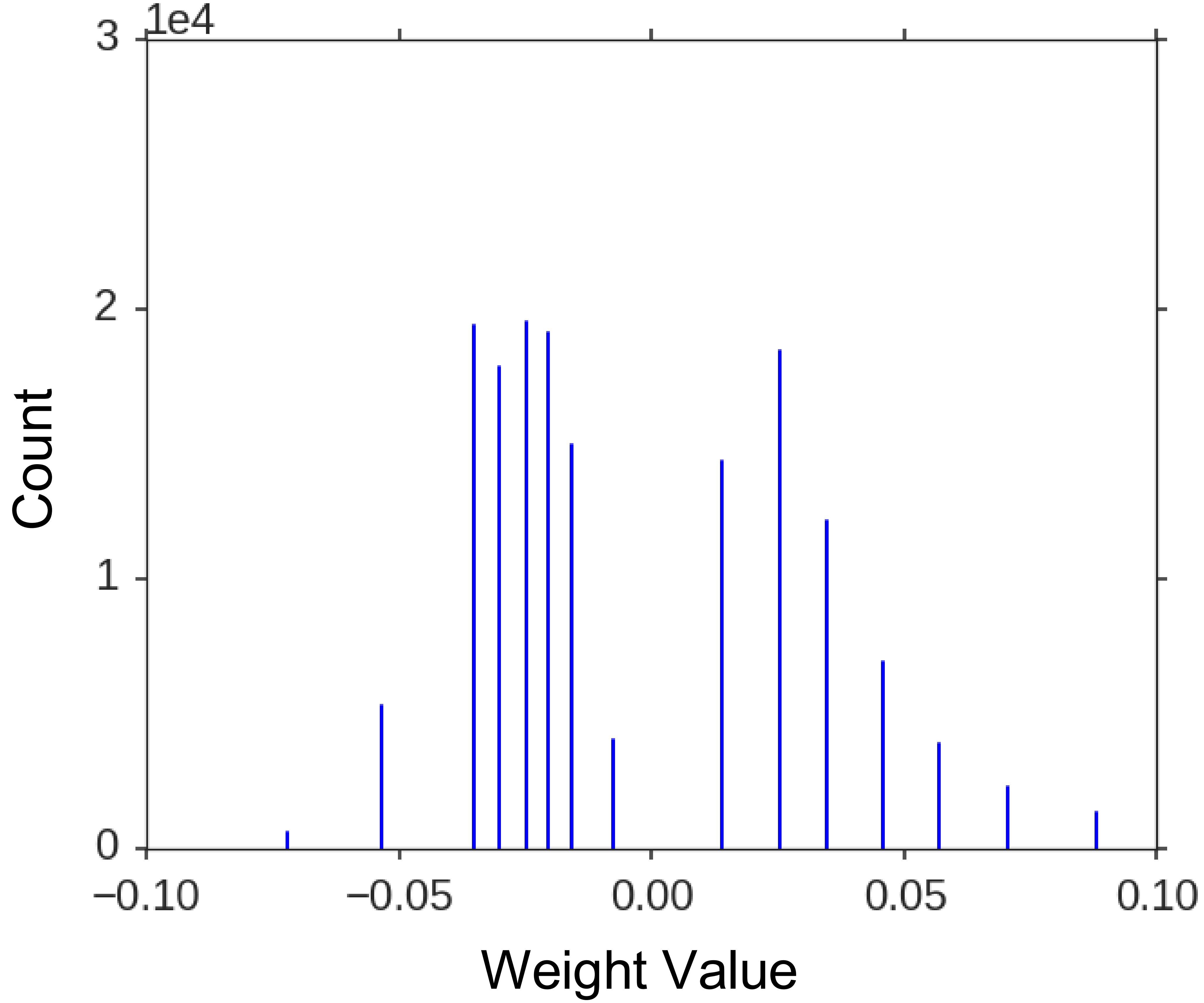
Deep Compression [Han *et al.*, ICLR 2016]

After Quantization: Discrete Weight



Deep Compression [Han *et al.*, ICLR 2016]

After Quantization: Discrete Weight after Retraining

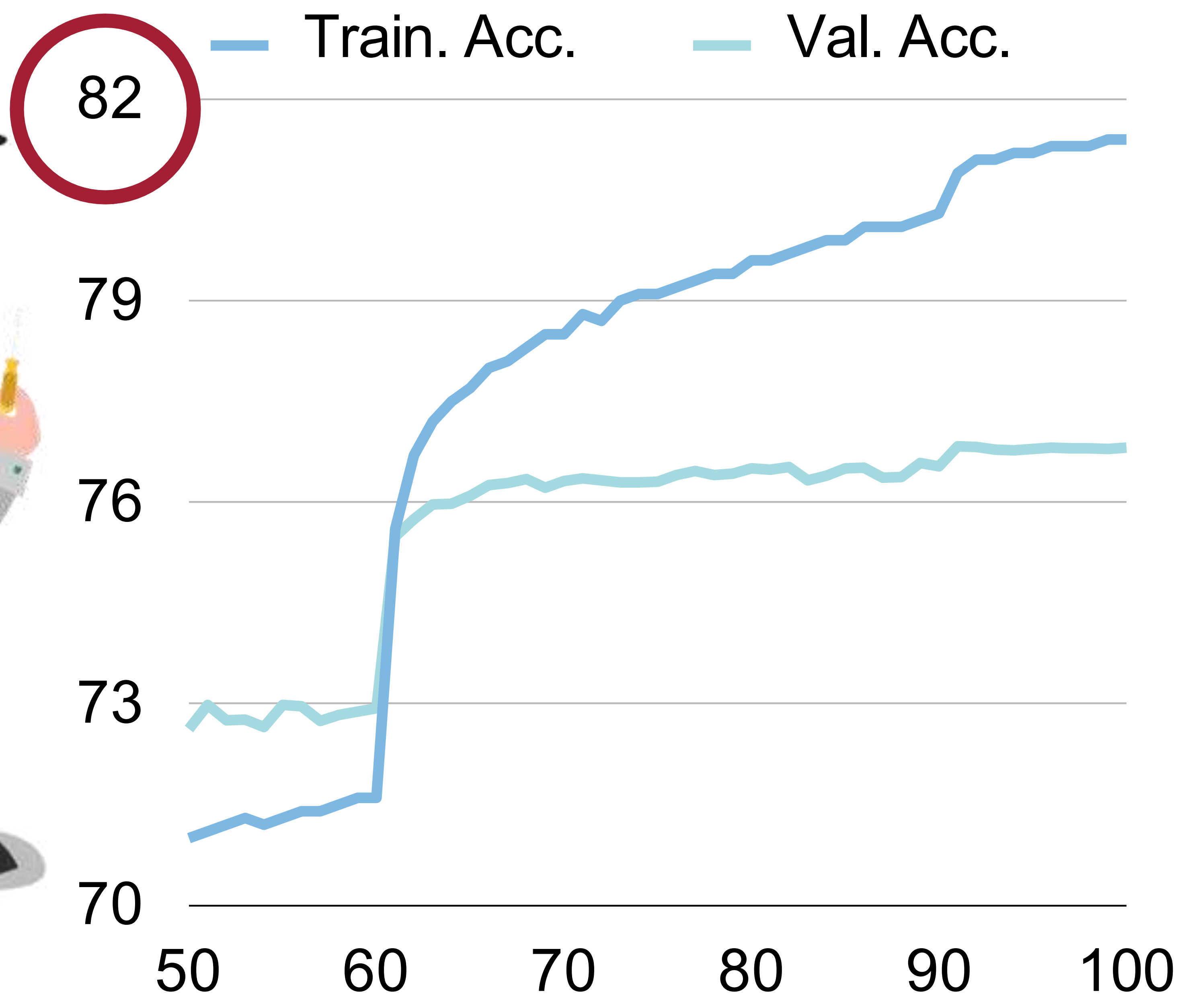


Deep Compression [Han *et al.*, ICLR 2016]

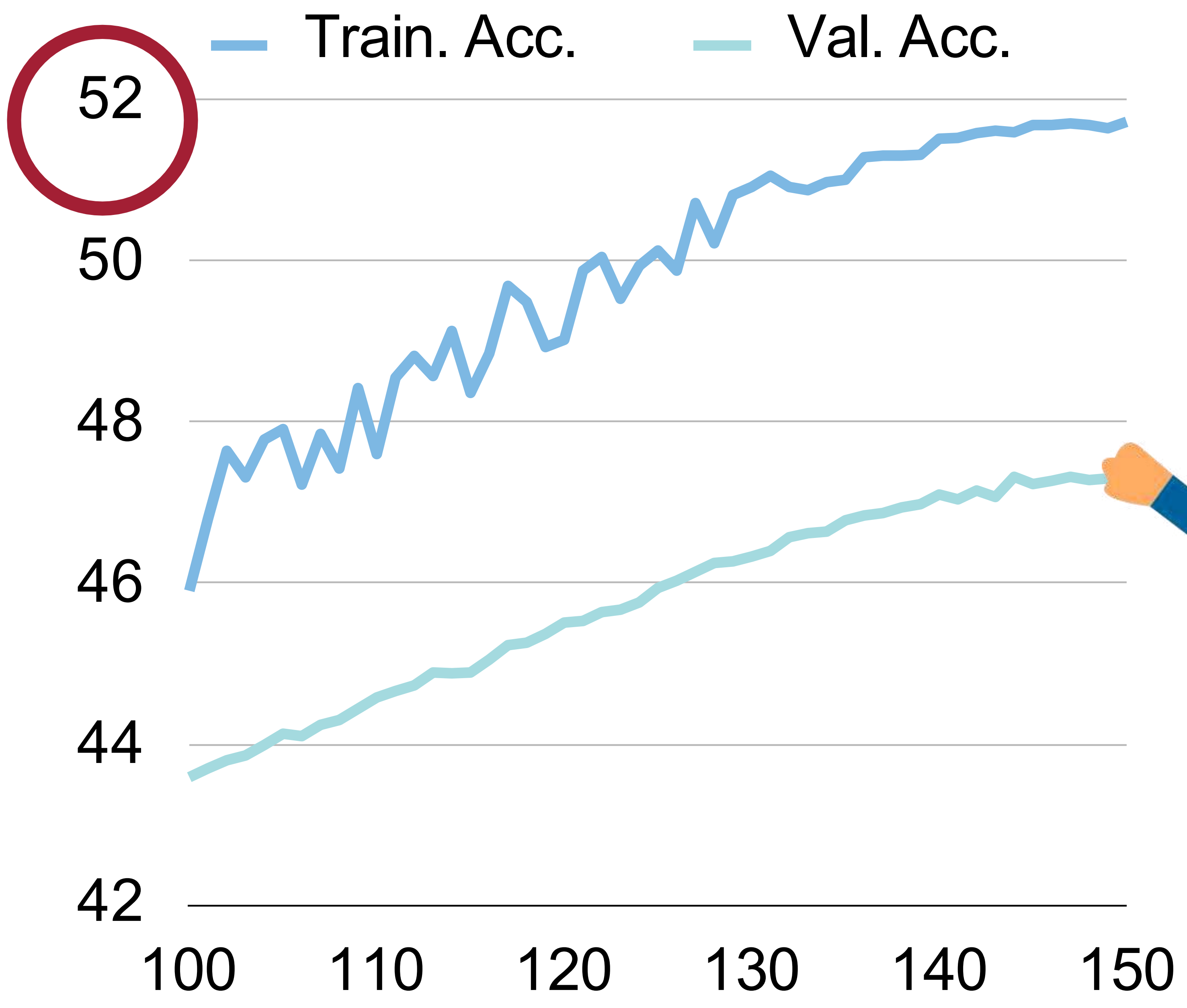
Distillation: Tiny models are hard to train

Tiny models underfit large datasets

Training curve for ResNet50



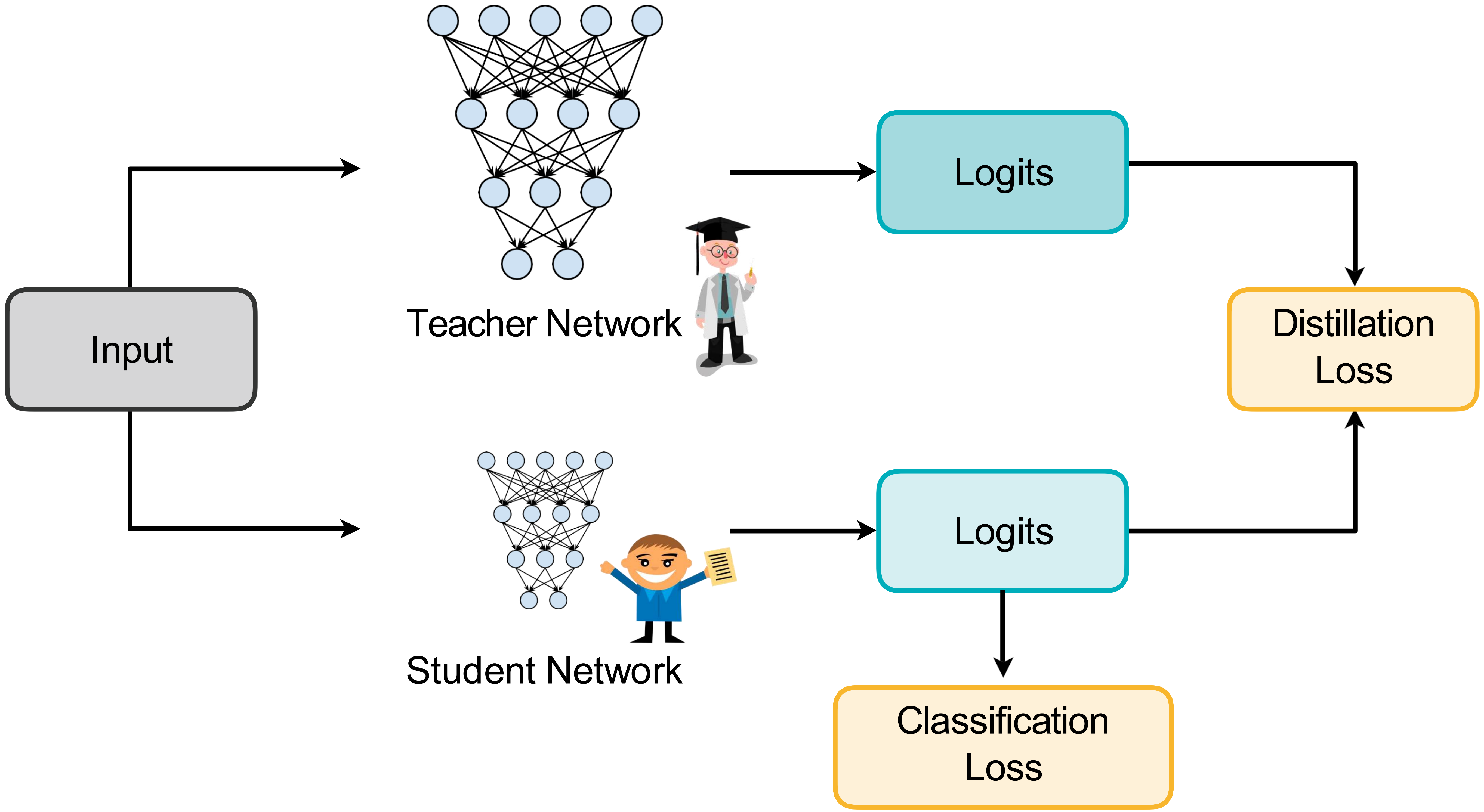
Training curve for MobileNetV2-Tiny



Question: Can we help the training of tiny models with large models?

Network Augmentation for Tiny Deep Learning [Cai *et al.*, ICLR 2022]

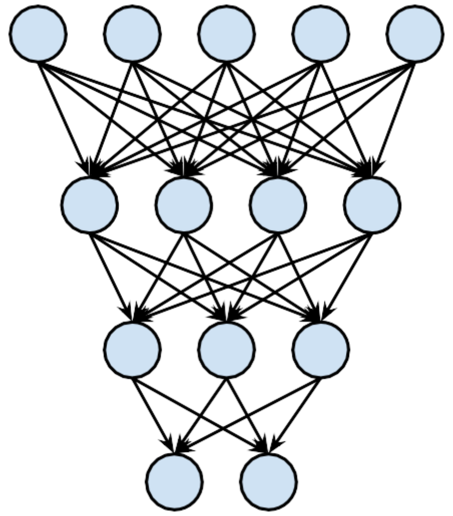
Illustration of knowledge distillation



Distilling the Knowledge in a Neural Network [Hinton *et al.*, NeurIPS Workshops 2014]

Intuition of knowledge distillation

Matching prediction probabilities between teacher and student

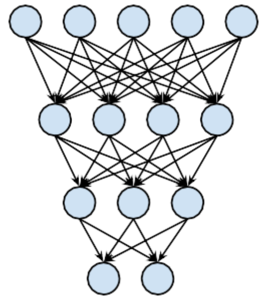


Teacher Network

	Logits	Probabilities
Cat	5	0.982
Dog	1	0.017

$$\frac{\exp(5)}{\exp(5) + \exp(1)}$$

$$\frac{\exp(1)}{\exp(5) + \exp(1)}$$



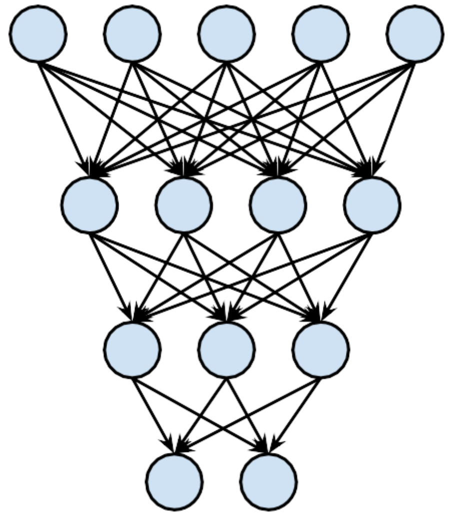
Student Network

	Logits	Probabilities
Cat	3	0.731
Dog	2	0.269

The student model is less confident

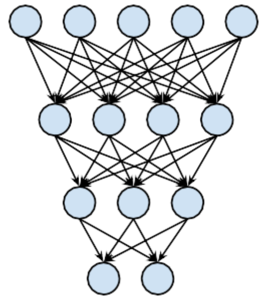
Intuition of knowledge distillation

Matching prediction probabilities between teacher and student



Teacher Network

	Logits	Probabilities
Cat	5	0.982
Dog	1	0.017

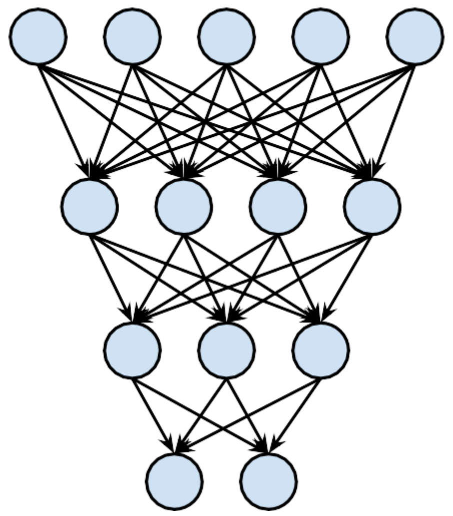


Student Network

	Logits	Probabilities
Cat	3	0.731
Dog	2	0.269

Intuition of knowledge distillation

Concept of temperature



Teacher Network

$$\frac{\exp(5/1)}{\exp(5/1) + \exp(1/1)}$$

	Logits	Probabilities (T=1)	Probabilities (T=10)
Cat	5	0.982	0.599
Dog	1	0.017	0.401

$$\frac{\exp(5/10)}{\exp(5/10) + \exp(1/10)}$$

A larger temperature smooths the output probability distribution.

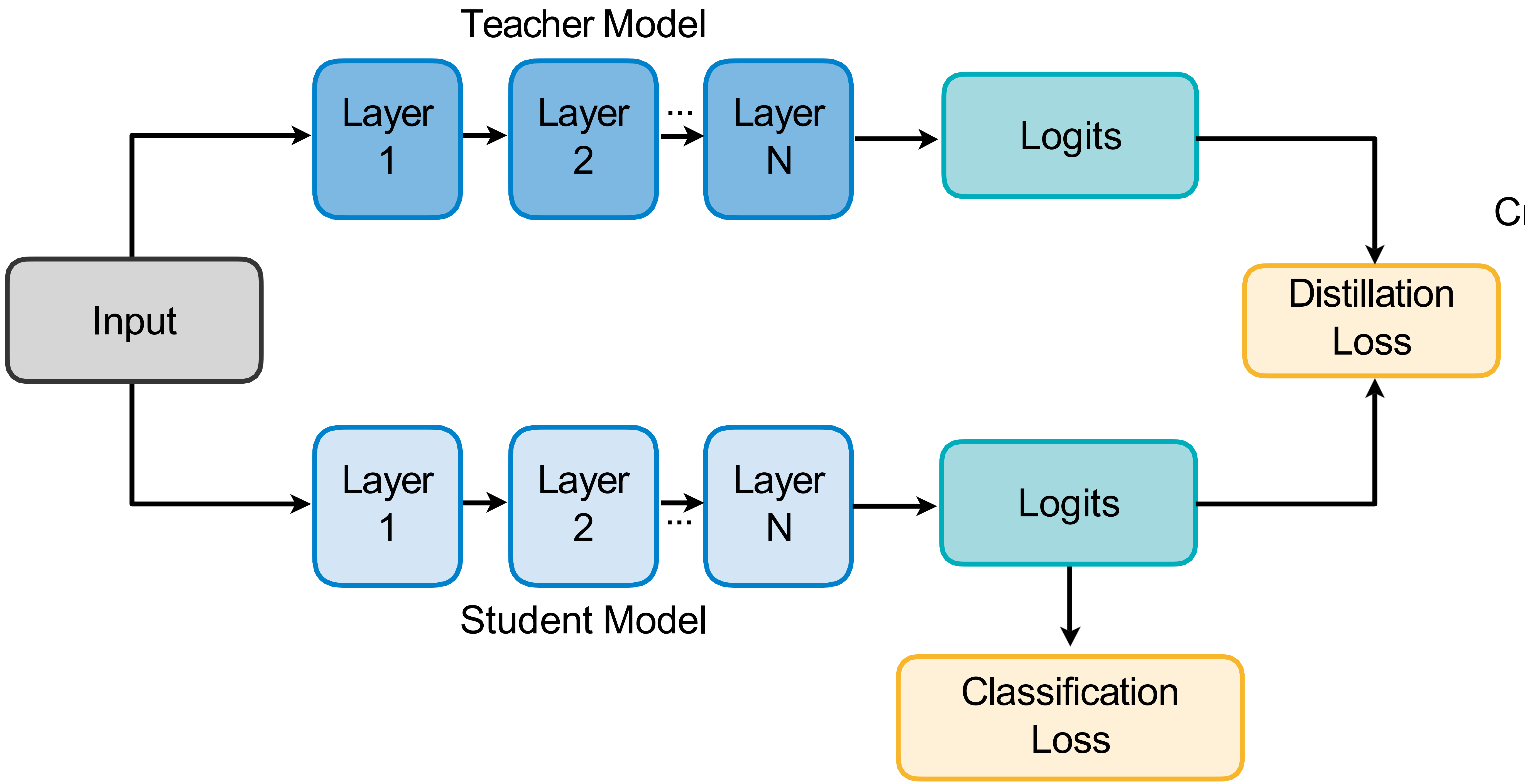
Formal Definition of KD

- Neural networks typically use a softmax function to generate the **logits** z_i to class **probabilities** $p(z_i, T) = \frac{\exp(z_i / T)}{\sum_j \exp(z_j / T)}$. Here, $i, j = 0,1,2,\ldots,C-1$, where C is the number of classes. T is the temperature, which is normally set to 1.

- The goal of knowledge distillation is to **align the class probability distributions from teacher and student networks**.

Distilling the Knowledge in a Neural Network [Hinton *et al.*, NeurIPS Workshops 2014]

Matching output logits



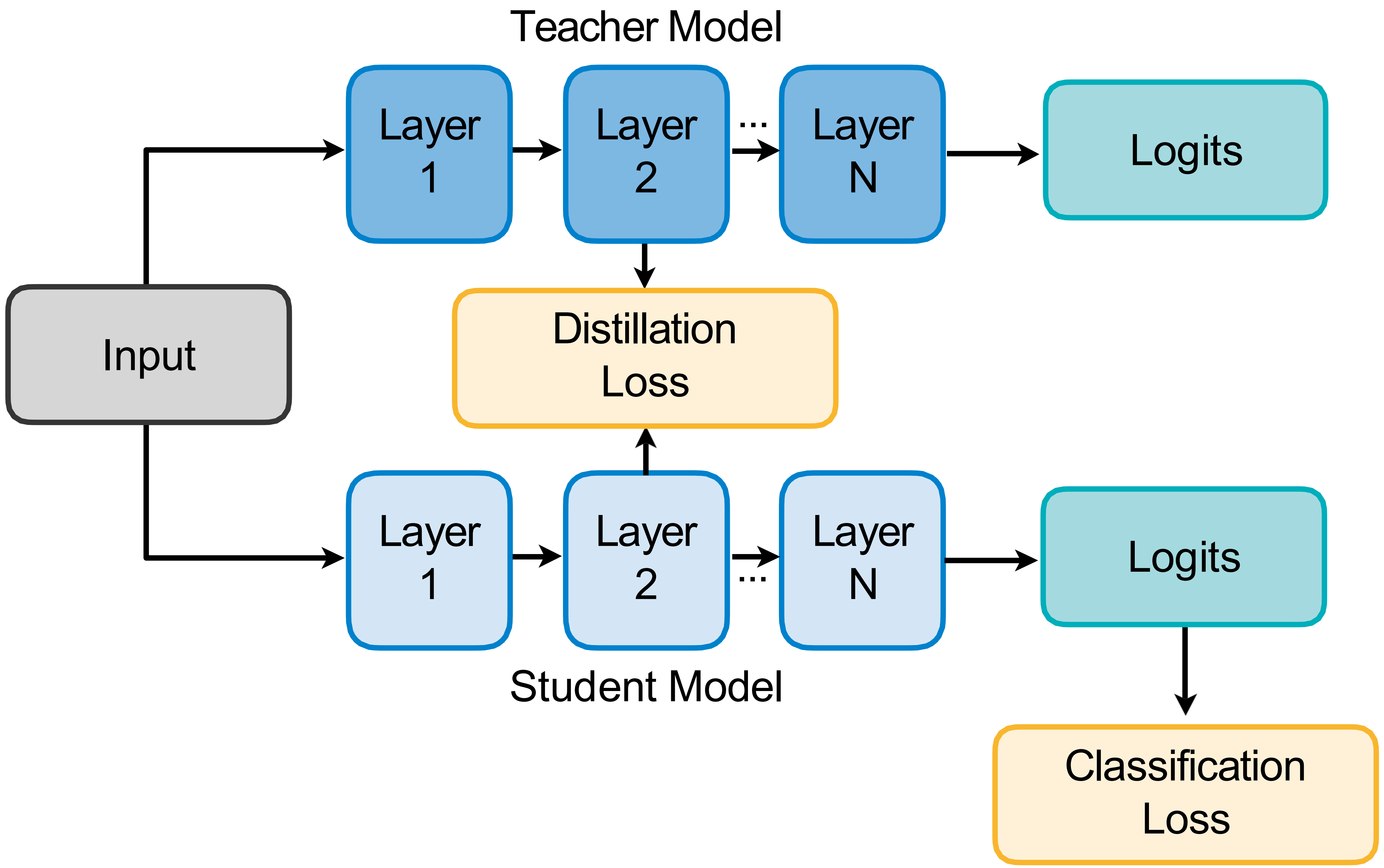
Cross entropy loss:
$\mathbb{E}(-p_t \log p_s)$;

L2 loss:
$E(\|p_t - p_s\|_2^2)$

Distilling the Knowledge in a Neural Network [Hinton *et al.*, NeurIPS Workshops 2014]
Do Deep Nets Really Need to be Deep? [Ba and Caruana, NeurIPS 2014]

What else to match other than output logits?

Matching intermediate weights

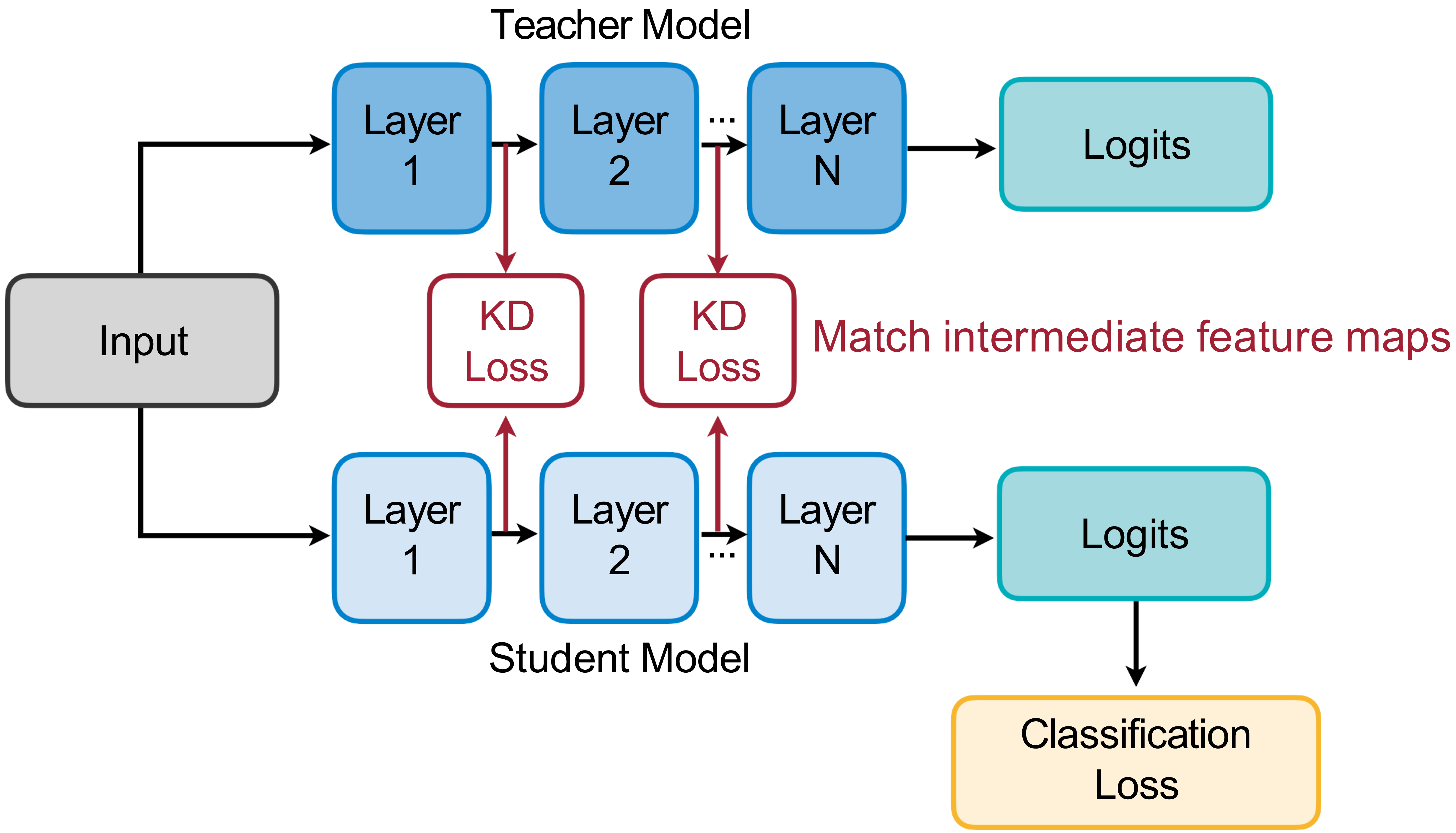


Knowledge Distillation: A Survey [Gou *et al.*, IJCV 2020]

Matching intermediate features

Minimizing maximum mean discrepancy between feature maps

- Intuition: teacher and student networks should have similar **feature** distributions, not just output probability distributions.



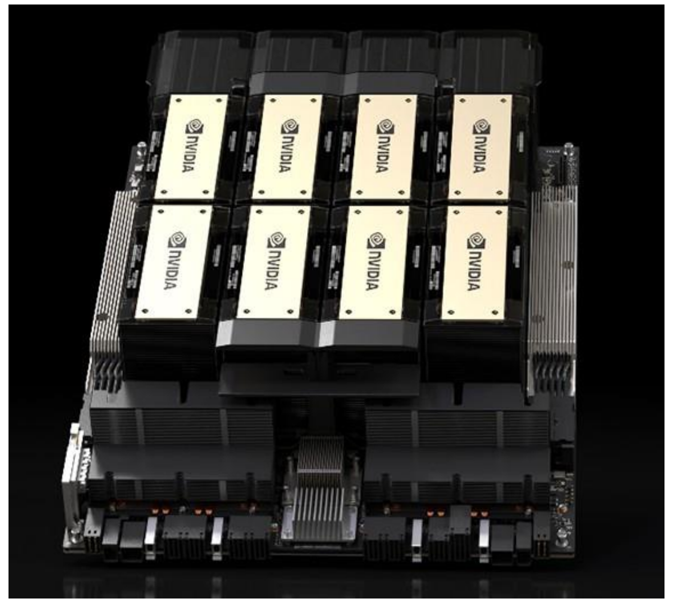
Like What You Like: Knowledge Distill via Neuron Selectivity Transfer [Huang and Wang, arXiv 2017]

Lecture Plan

1. Federated learning and the deep leakage from gradients
2. Pruning, quantization and knowledge distillation
3. **Memory bottleneck of on-device training**
4. Tiny transfer learning (TinyTL)
5. Sparse back-propagation (SparseBP)

On-Device Training is Challenging

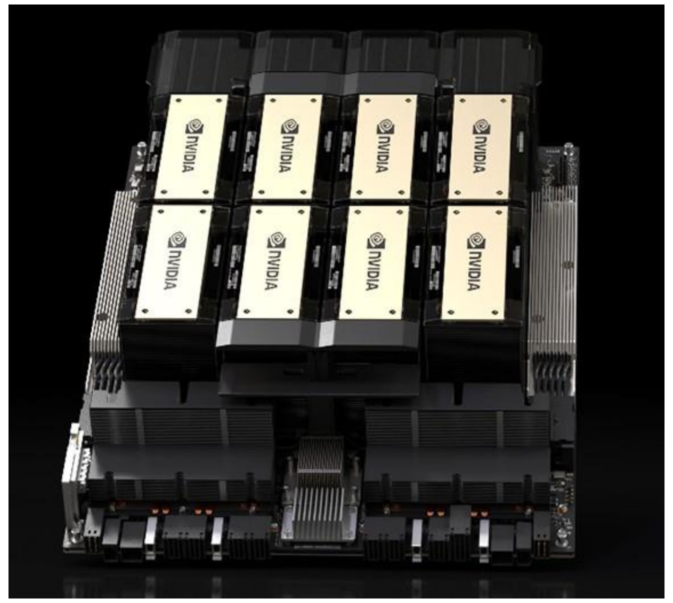
Memory size is too small to hold DNNs



	Cloud AI	**Mobile AI**
Memory (Activation)	141GB	4GB
Storage (Weights)	~TB/PB	256GB

On-Device Training is Challenging

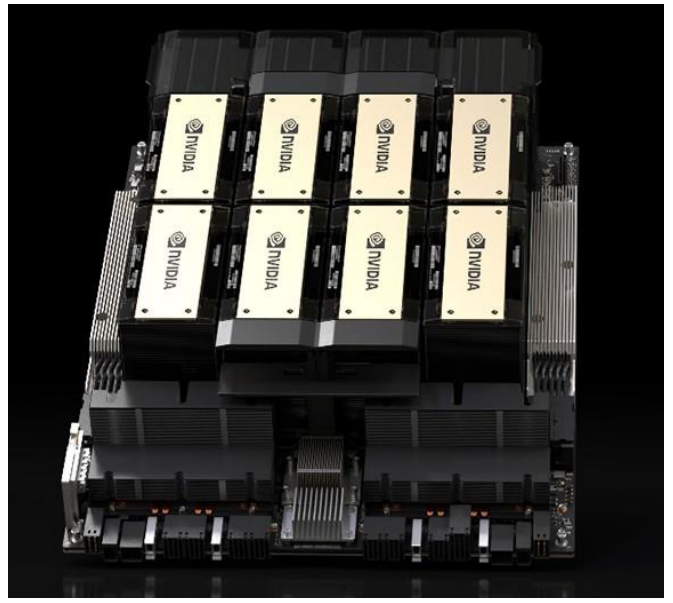
Memory size is too small to hold DNNs



	Cloud AI	Mobile AI	Tiny AI
Memory (Activation)	141GB	4GB	320kB
Storage (Weights)	~TB/PB	256GB	1MB

On-Device Training is Challenging

Memory size is too small to hold DNNs



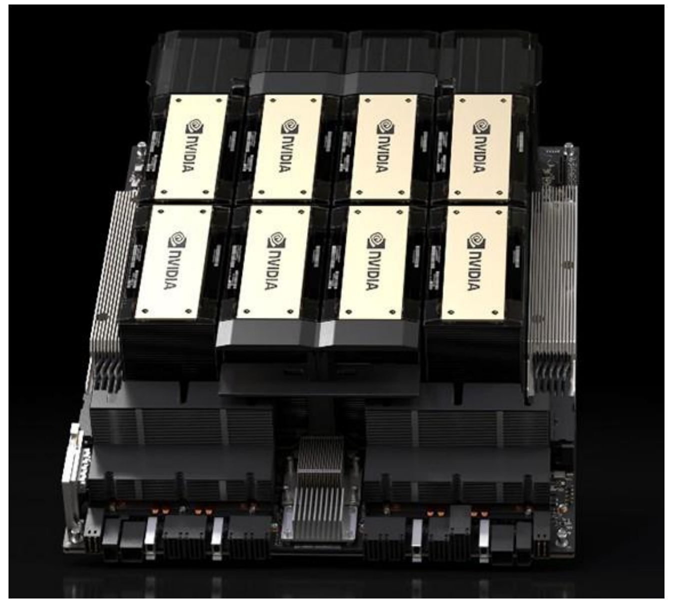
	Cloud AI	Mobile AI	Tiny AI
Memory (Activation)	141GB	4GB	320kB
Storage (Weights)	~TB/PB	256GB	1MB

13,000x smaller

1,000,000x smaller

On-Device Training is Challenging

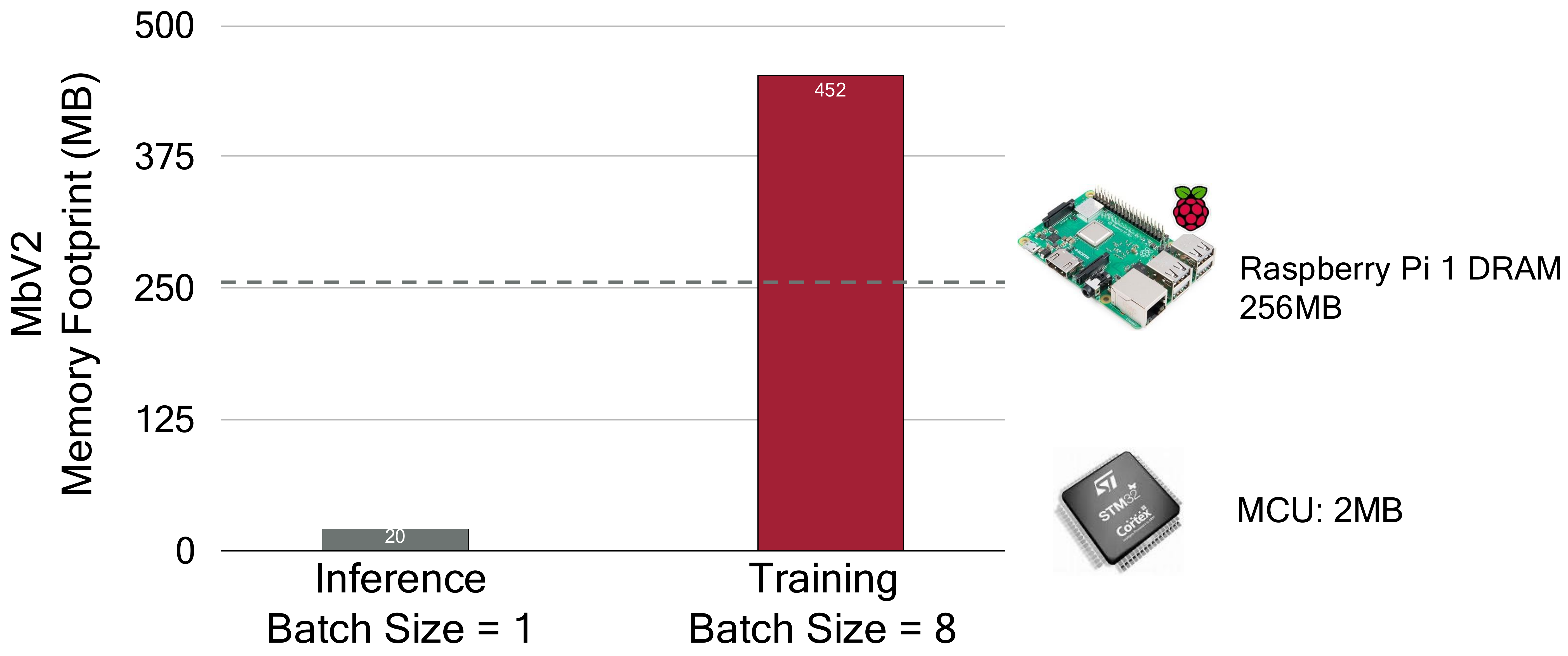
Memory size is too small to hold DNNs



	Cloud AI	Mobile AI	Tiny AI
Memory (Activation)	141GB	4GB	320kB
Storage (Weights)	~TB/PB	256GB	1MB

- We need to reduce both **weights** and **activation** to fit DNNs for On-Device Training

Training Memory is the Key Bottleneck



- Edge devices have tight memory constraints. The training memory footprint of neural networks can easily exceed the limit.

TinyTL: Reduce Activations, Not Trainable Parameters for Efficient On-Device Learning [Cai *et al.*, NeurIPS 2020]

Training Memory is the Key Bottleneck

Question: Why training memory is much larger than inference?

Answer: Because of intermediate **activations**

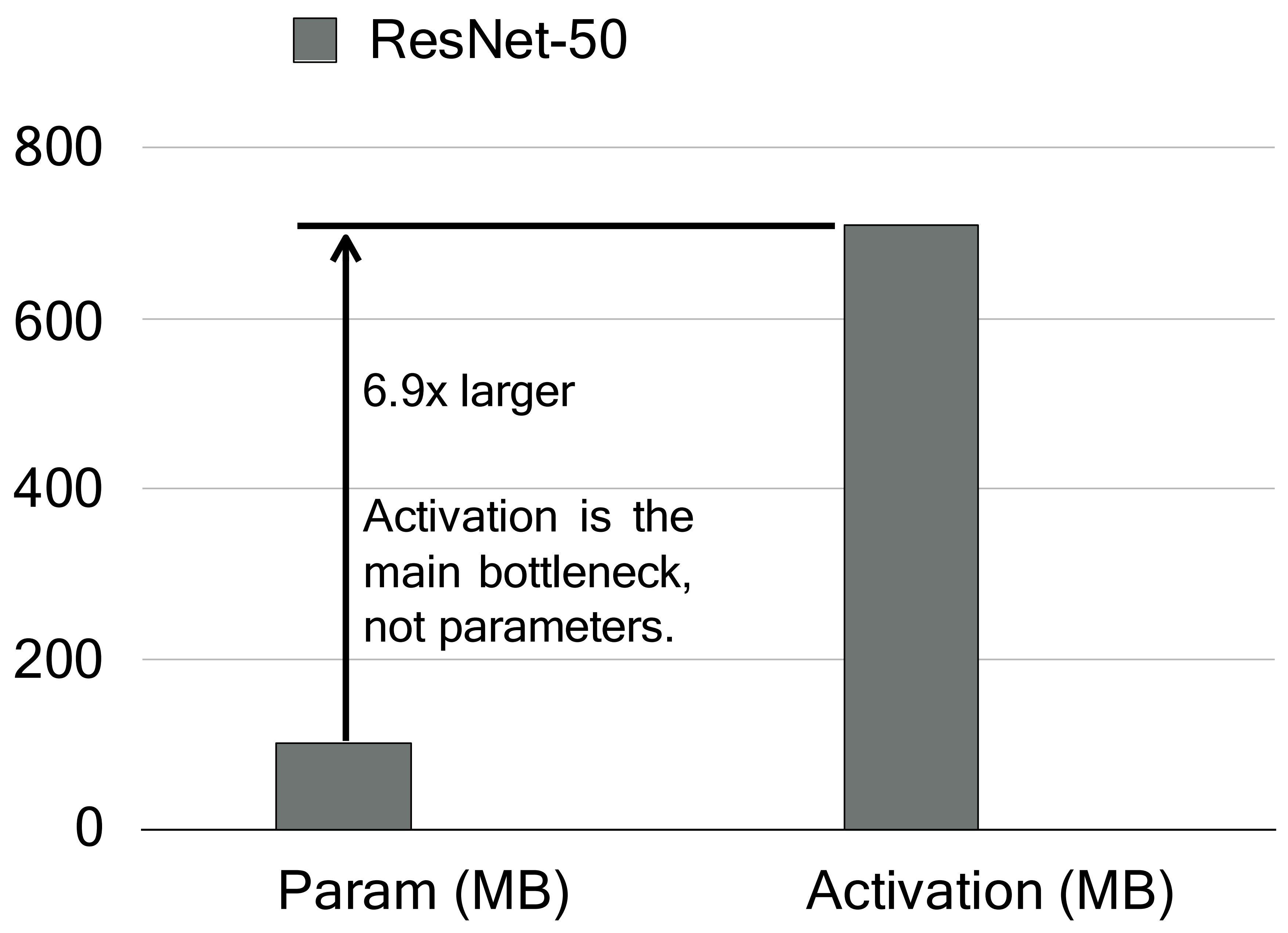
Forward: $\mathbf{a}_{i+1} = \mathbf{a}_i \mathbf{W}_i + \mathbf{b}_i$

Backward: $\dfrac{\partial L}{\partial \mathbf{W}_i} = \mathbf{a}_i^T \dfrac{\partial L}{\partial \mathbf{a}_{i+1}}$

- Inference does not need to store activations, training does.
- Activations grows linearly with batch size, which is always 1 for inference.
- Even with bs=1, activations are beyond memory limit of many edge devices.

TinyTL: Reduce Activations, Not Trainable Parameters for Efficient On-Device Learning [Cai *et al.*, NeurIPS 2020]

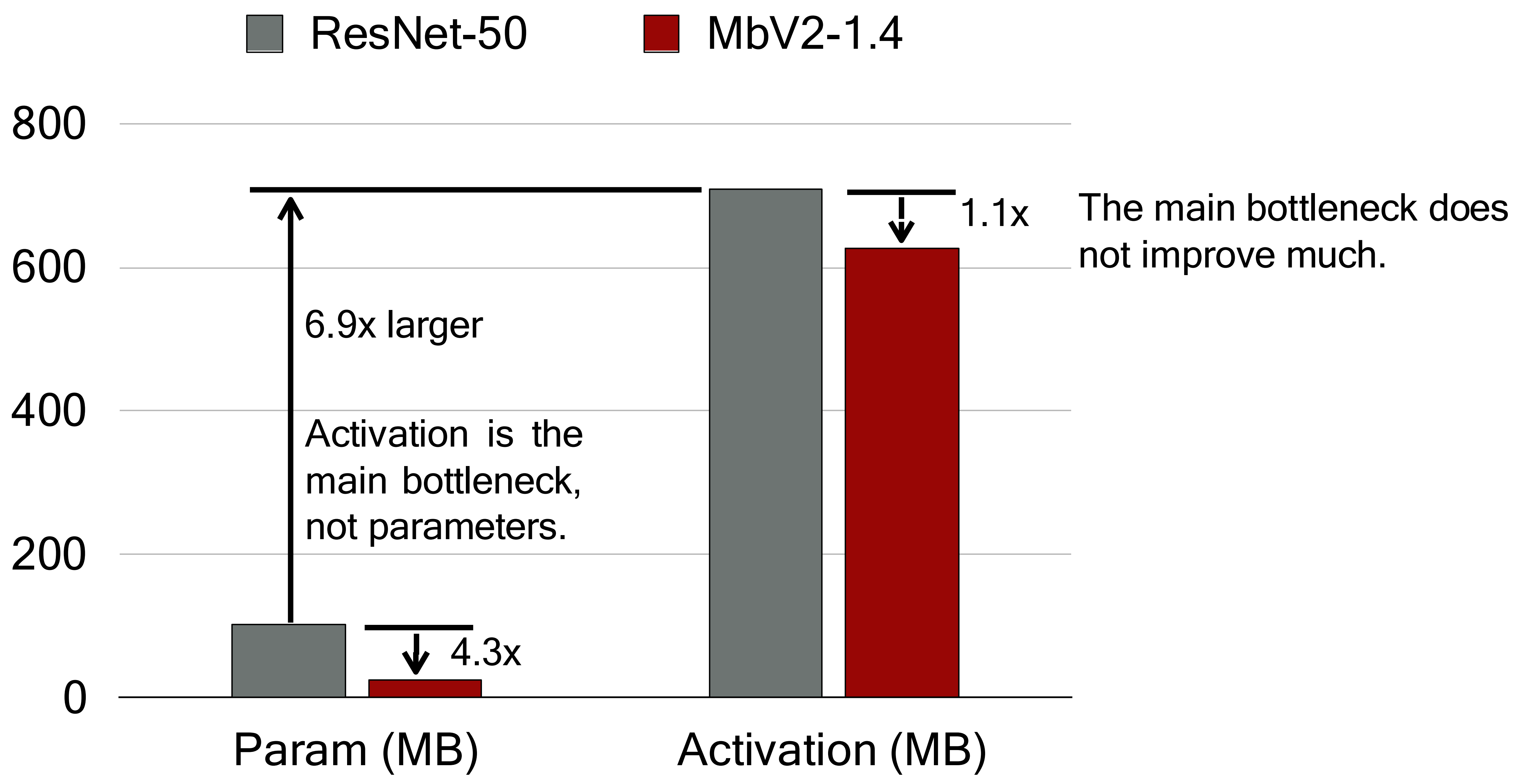
Activation is the Memory Bottleneck in CNNs



- Activation is the main bottleneck for CNN training

TinyTL: Reduce Activations, Not Trainable Parameters for Efficient On-Device Learning [Cai *et al.*, NeurIPS 2020]

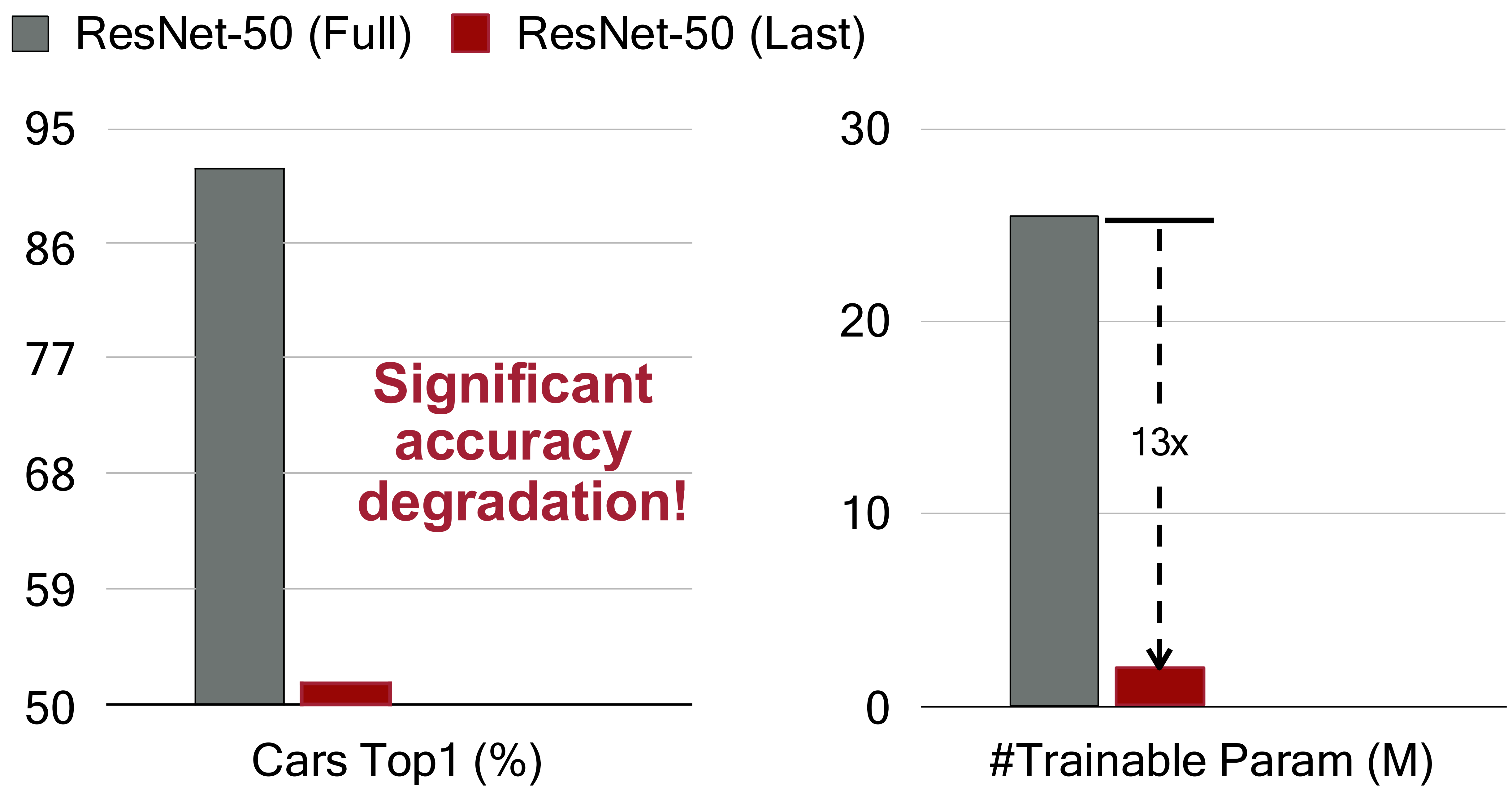
Activation is the Memory Bottleneck in CNNs



- Activation is the main bottleneck for CNN training.
- MobileNets focus on reducing the number of parameters or FLOPs, while the main bottleneck does not improve much.

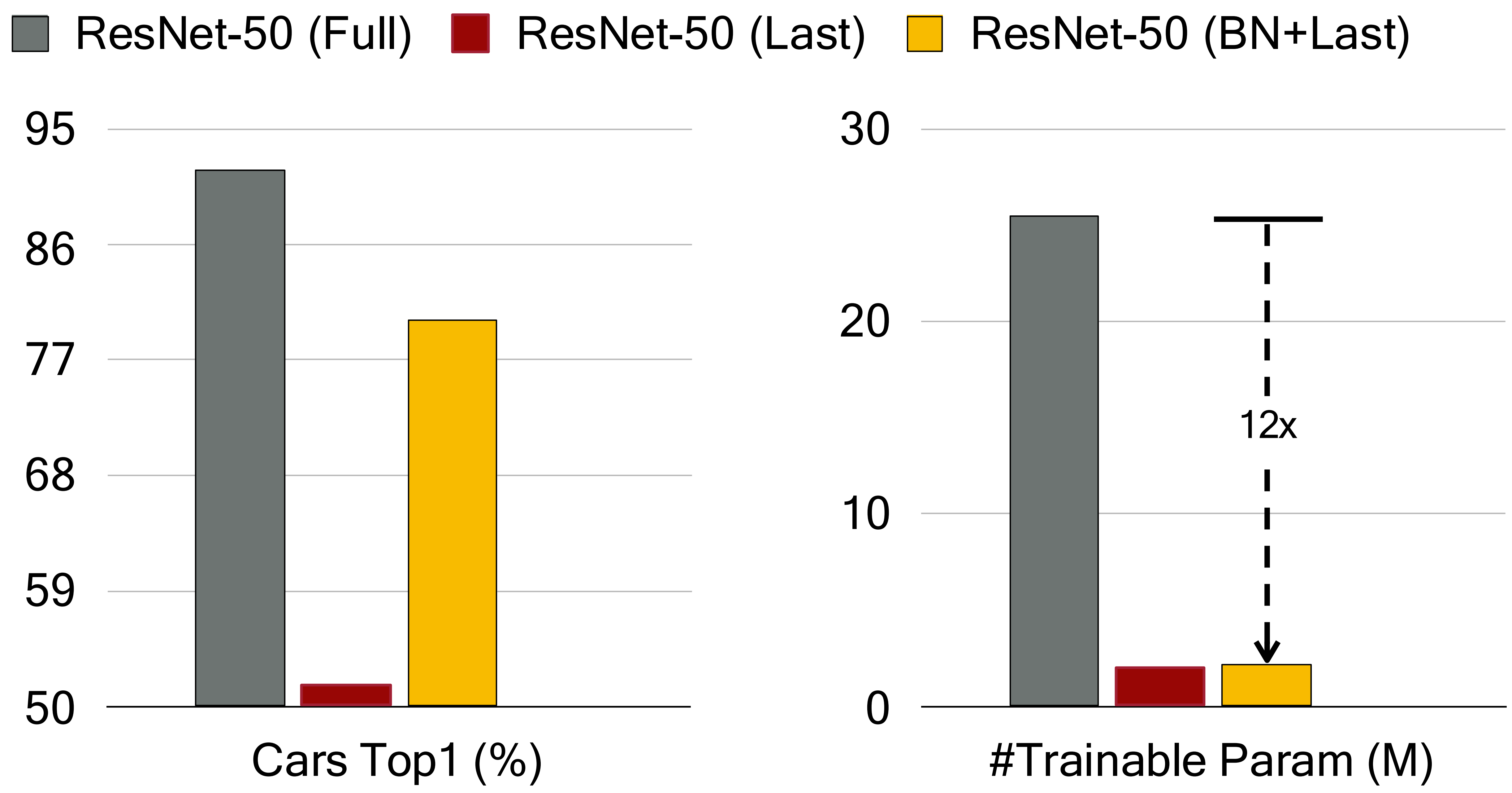
TinyTL: Reduce Activations, Not Trainable Parameters for Efficient On-Device Learning [Cai *et al.*, NeurIPS 2020]

Parameter-Efficient Transfer Learning in CNNs



- Full: Fine-tune the full network. Better accuracy but highly inefficient.
- Last: Only fine-tune the last classifier head. Efficient but the capacity is limited.

Parameter-Efficient Transfer Learning in CNNs

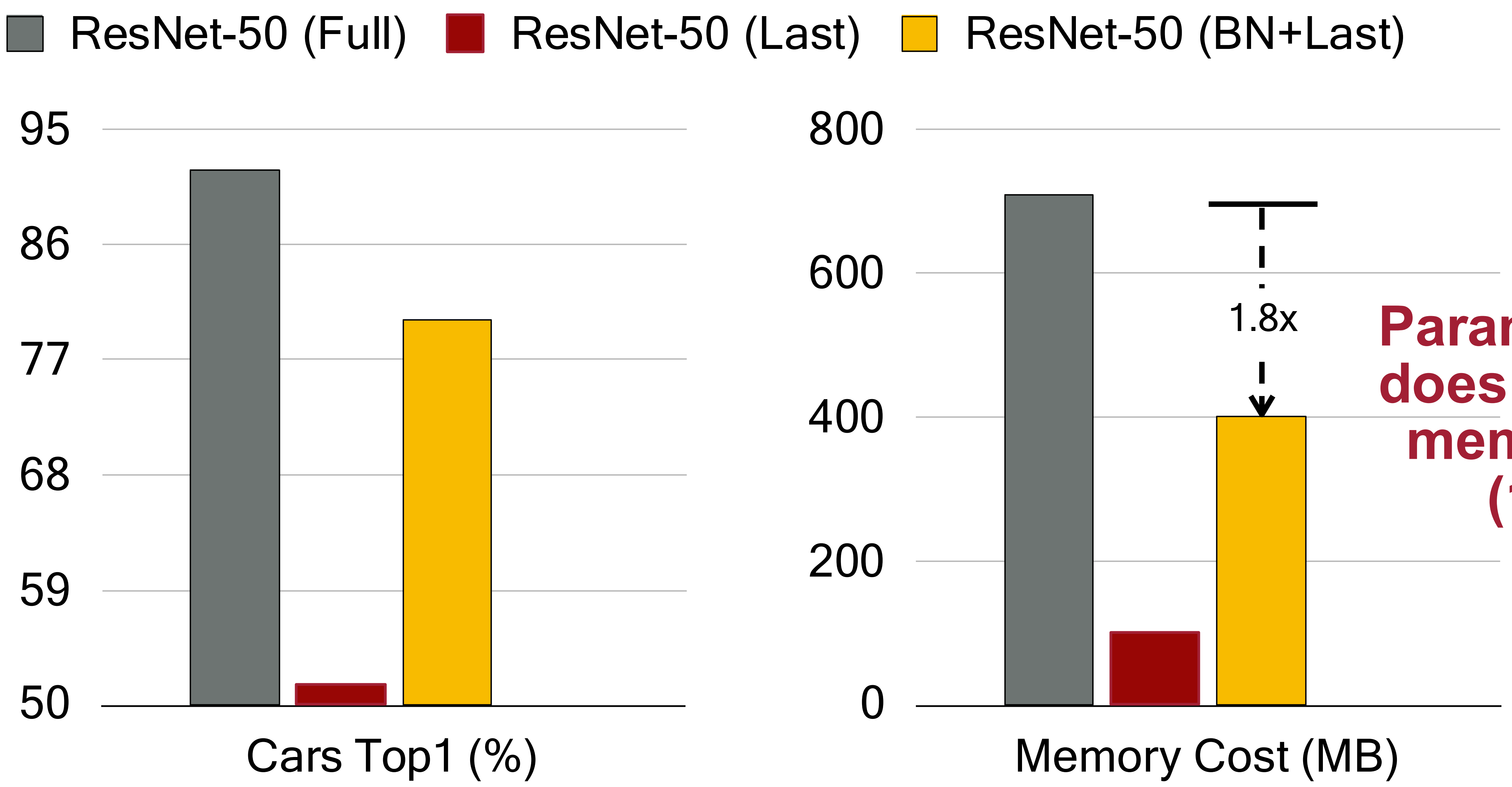


- Full: Fine-tune the full network. Better accuracy but highly inefficient.
- Last: Only fine-tune the last classifier head. Efficient but the capacity is limited.
- BN+Last: Fine-tune the BN layers and the last layer. Parameter-efficient.

Question: Is BN+Last update or Last-only update enough for on-device transfer learning?

K for the Price of 1: Parameter-efficient Multi-task and Transfer Learning [Mudrarkarta *et al.*, ICLR 2019]

Parameter-Efficient Transfer Learning in CNNs

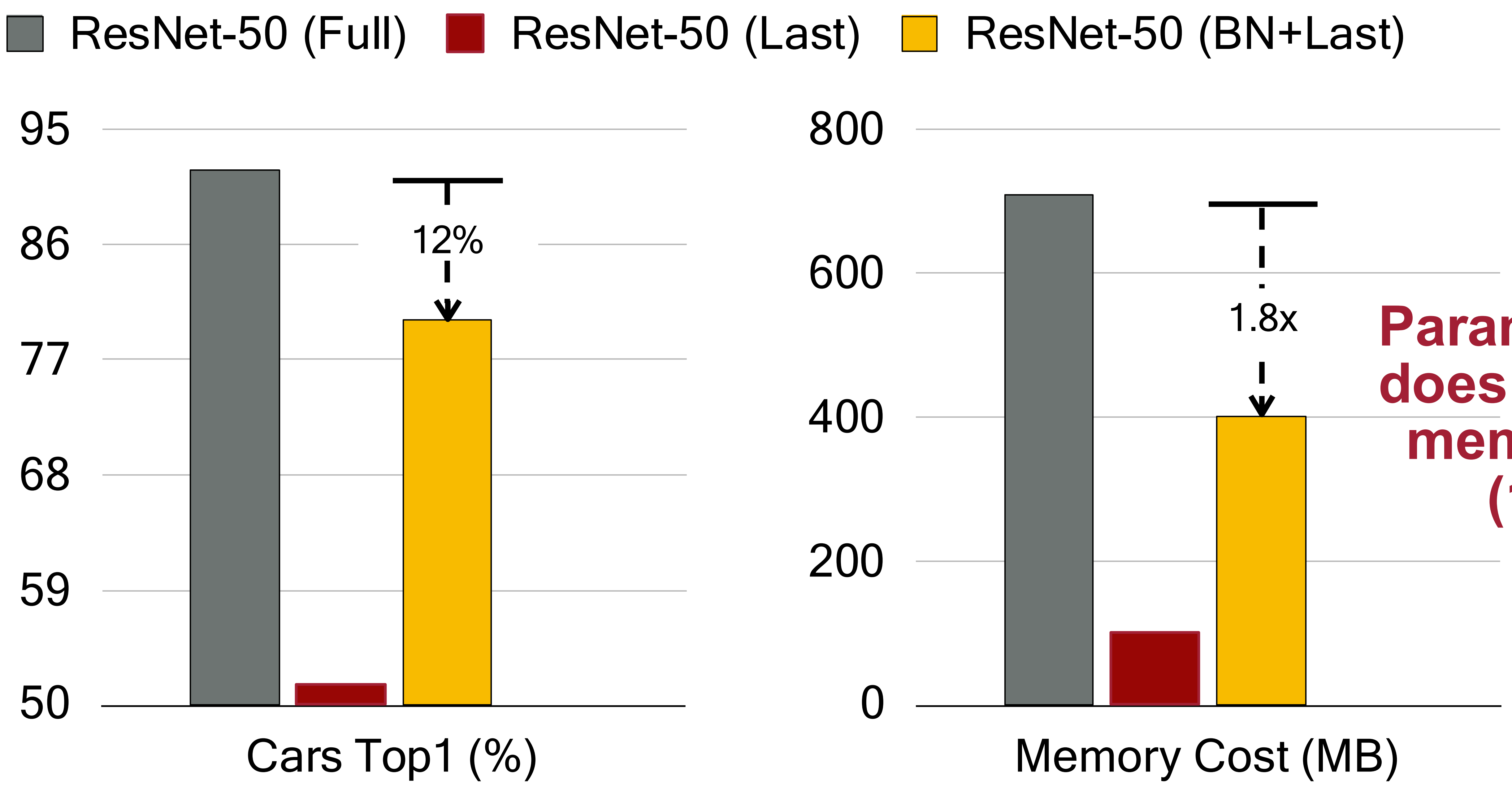


Parameter-efficiency does not translate to memory-efficiency (12x vs 1.8x)

- Full: Fine-tune the full network. Better accuracy but highly inefficient.
- Last: Only fine-tune the last classifier head. Efficient but the capacity is limited.
- BN+Last: Fine-tune the BN layers and the last layer. Parameter-efficient, **but the memory saving is limited.**

K for the Price of 1: Parameter-efficient Multi-task and Transfer Learning [Mudrarkarta *et al.*, ICLR 2019]

Parameter-Efficient Transfer Learning in CNNs



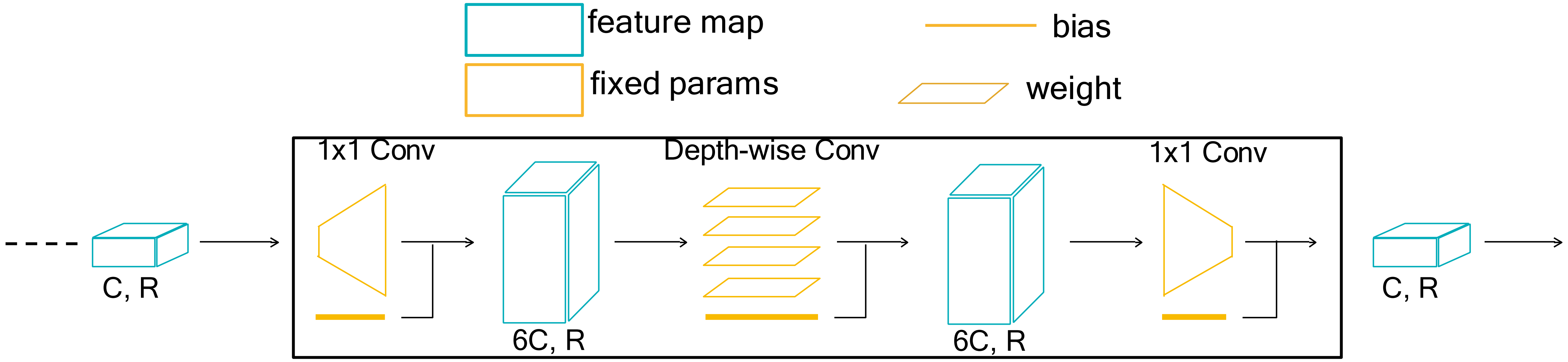
Parameter-efficiency does not translate to memory-efficiency (12x vs 1.8x)

- Full: Fine-tune the full network. Better accuracy but highly inefficient.
- Last: Only fine-tune the last classifier head. Efficient but the capacity is limited.
- BN+Last: Fine-tune the BN layers and the last layer. Parameter-efficient, **but the memory saving is limited. Significant accuracy loss.**

Lecture Plan

1. Federated Learning and the deep leakage from gradients
2. Pruning, quantization and knowledge distillation
3. Memory bottleneck of on-device training
4. **Tiny transfer learning (TinyTL)**
5. Sparse back-propagation (SparseBP)

Updating Weights is Memory-Expensive



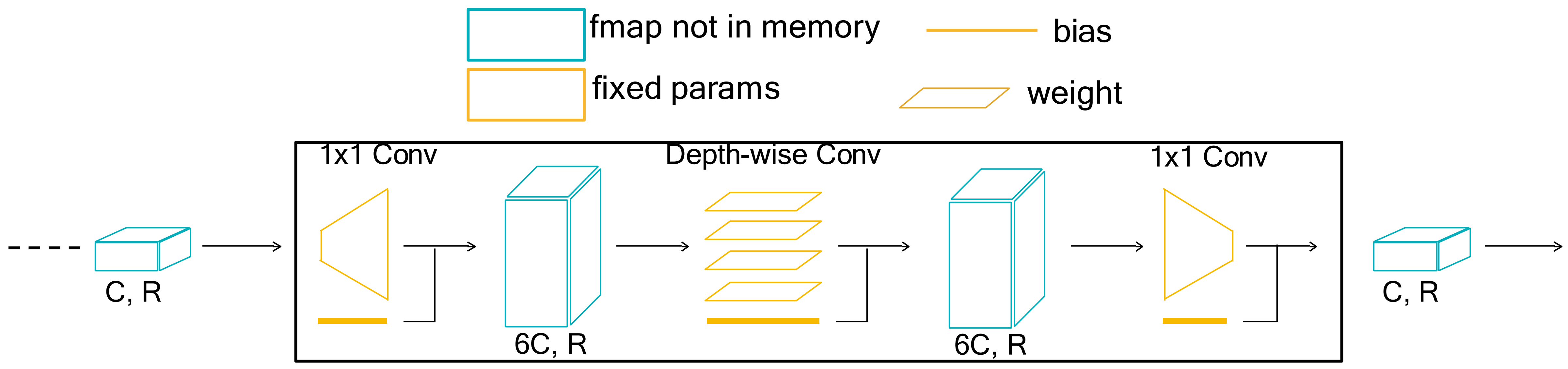
Forward: $\mathbf{a}_{i+1} = \mathbf{a}_i\mathbf{W}_i + \mathbf{b}_i$

Backward: $\frac{\partial L}{\partial \mathbf{W}_i} = \mathbf{a}_i^T \frac{\partial L}{\partial \mathbf{a}_{i+1}}, \quad \frac{\partial L}{\partial \mathbf{b}_i} = \frac{\partial L}{\partial \mathbf{a}_{i+1}} = \frac{\partial L}{\partial \mathbf{a}_{i+2}} \mathbf{W}_{i+1}^T$

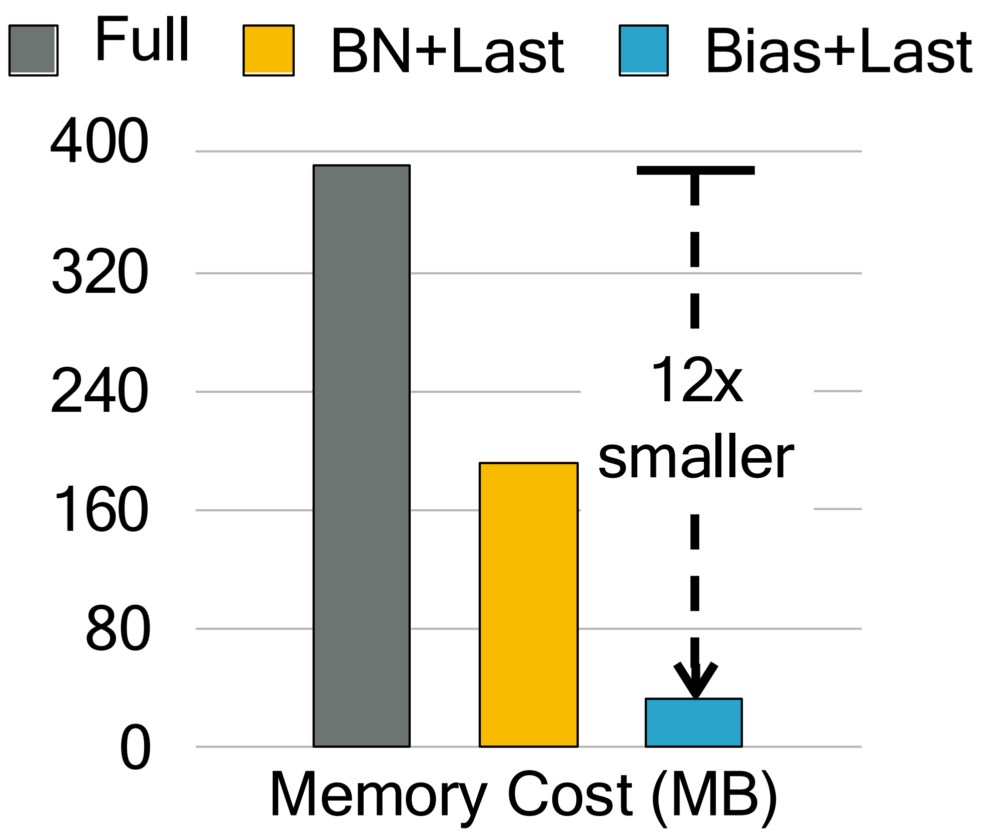
- Updating weights requires storing intermediate activations
- Updating biases does not, **is memory-efficient**

TinyTL: Reduce Activations, Not Trainable Parameters for Efficient On-Device Learning [Cai *et al.*, NeurIPS 2020]

TinyTL: Fine-tune Bias Only



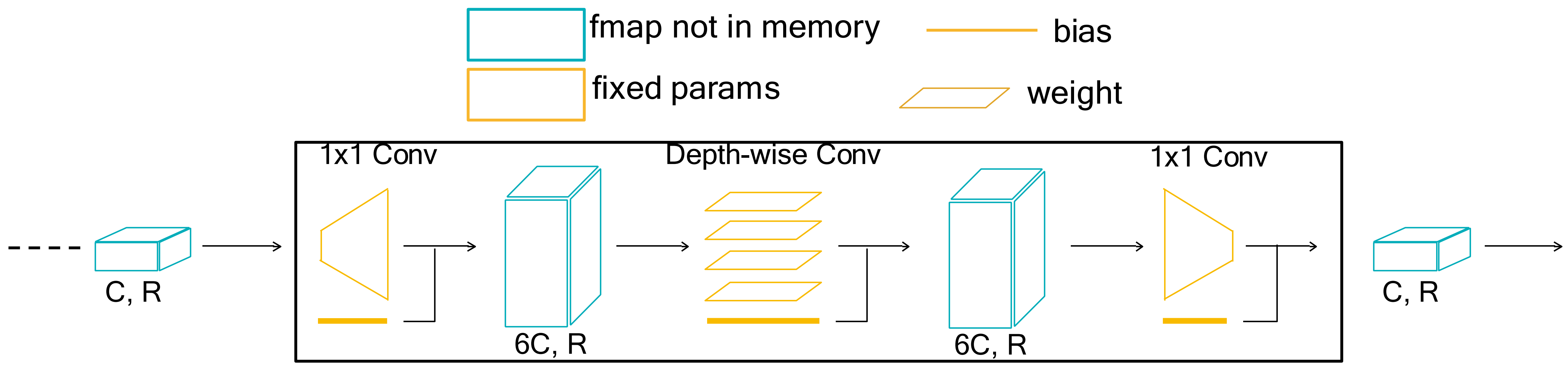
Fine-tune bias only



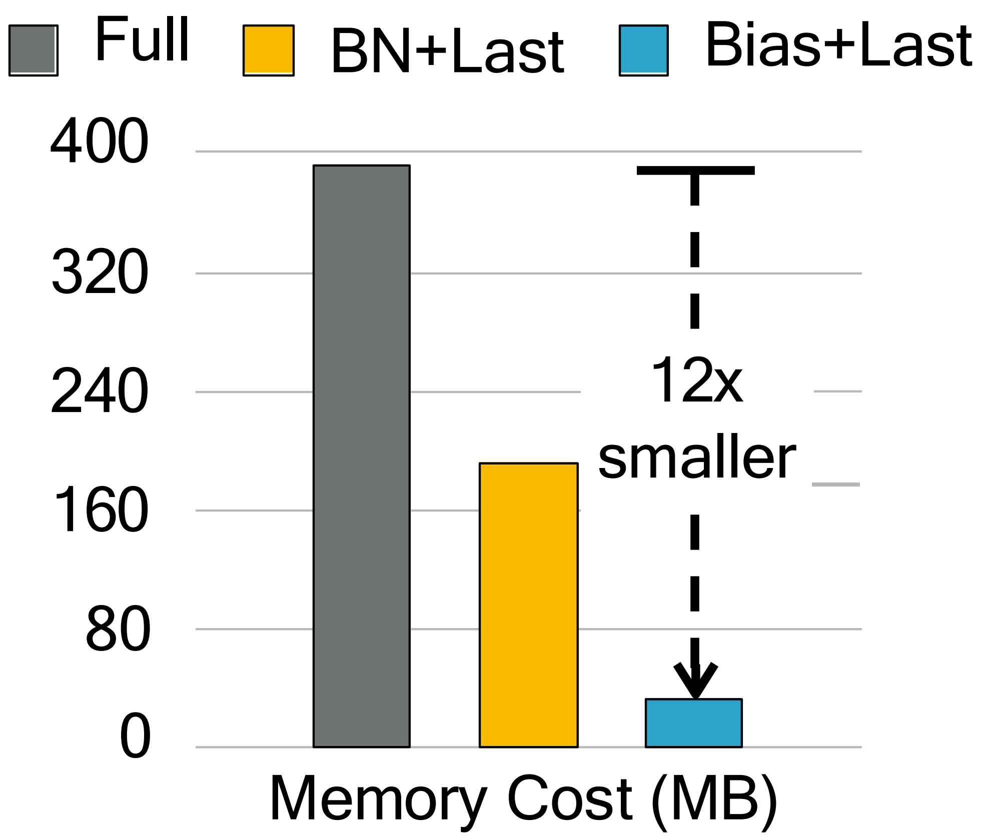
Freeze weights, only fine-tune biases
=> save 12x memory

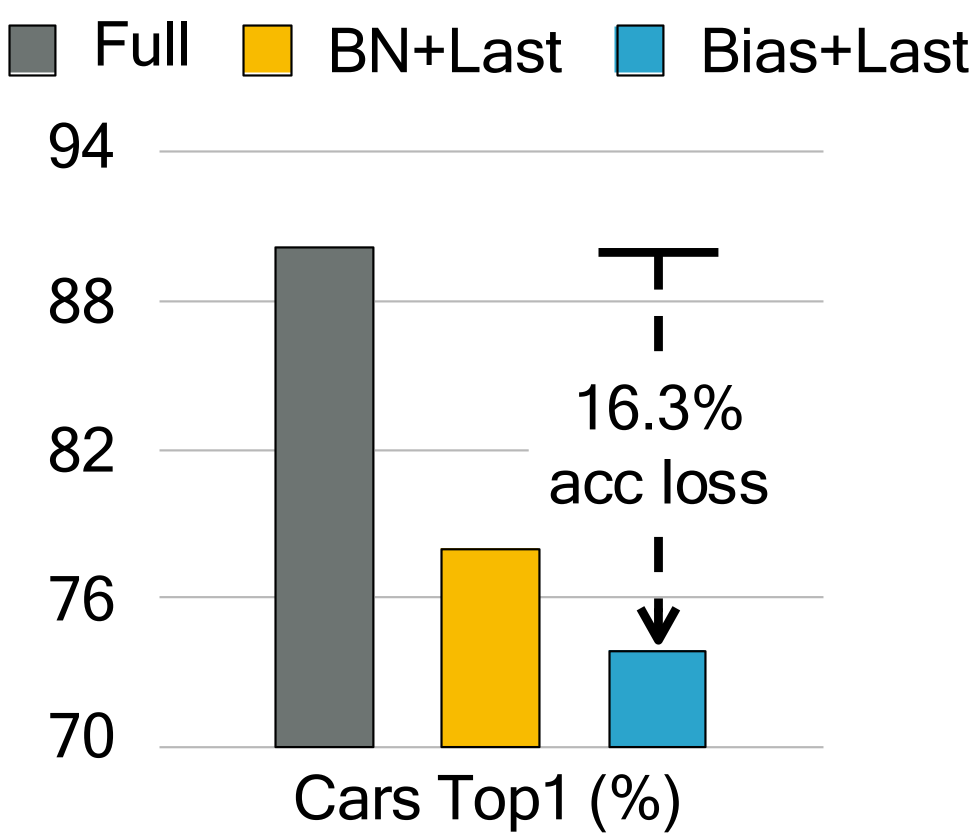
TinyTL: Reduce Activations, Not Trainable Parameters for Efficient On-Device Learning [Cai *et al.*, NeurIPS 2020]

TinyTL: Fine-tune Bias Only



Fine-tune bias only



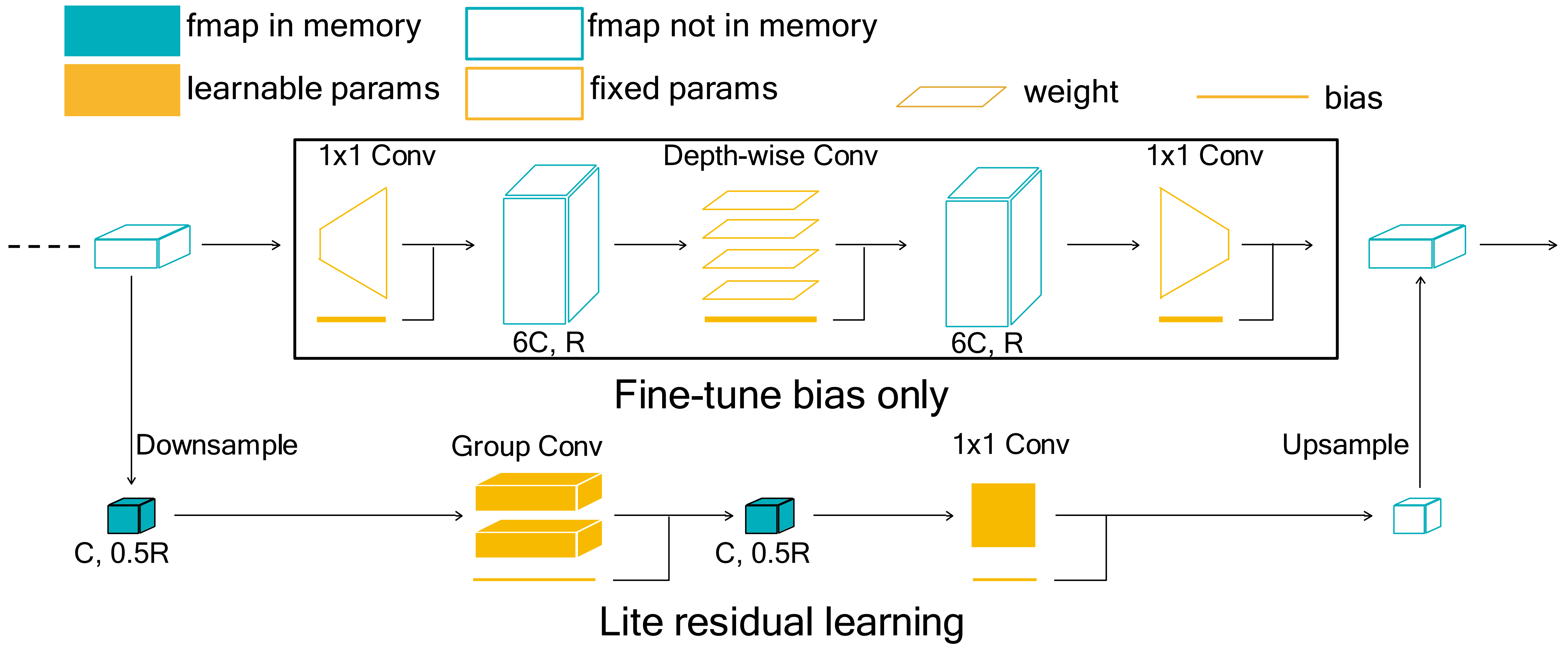


Freeze weights, only fine-tune biases

=> save 12x memory, but also hurt the accuracy

TinyTL: Reduce Activations, Not Trainable Parameters for Efficient On-Device Learning [Cai *et al.*, NeurIPS 2020]

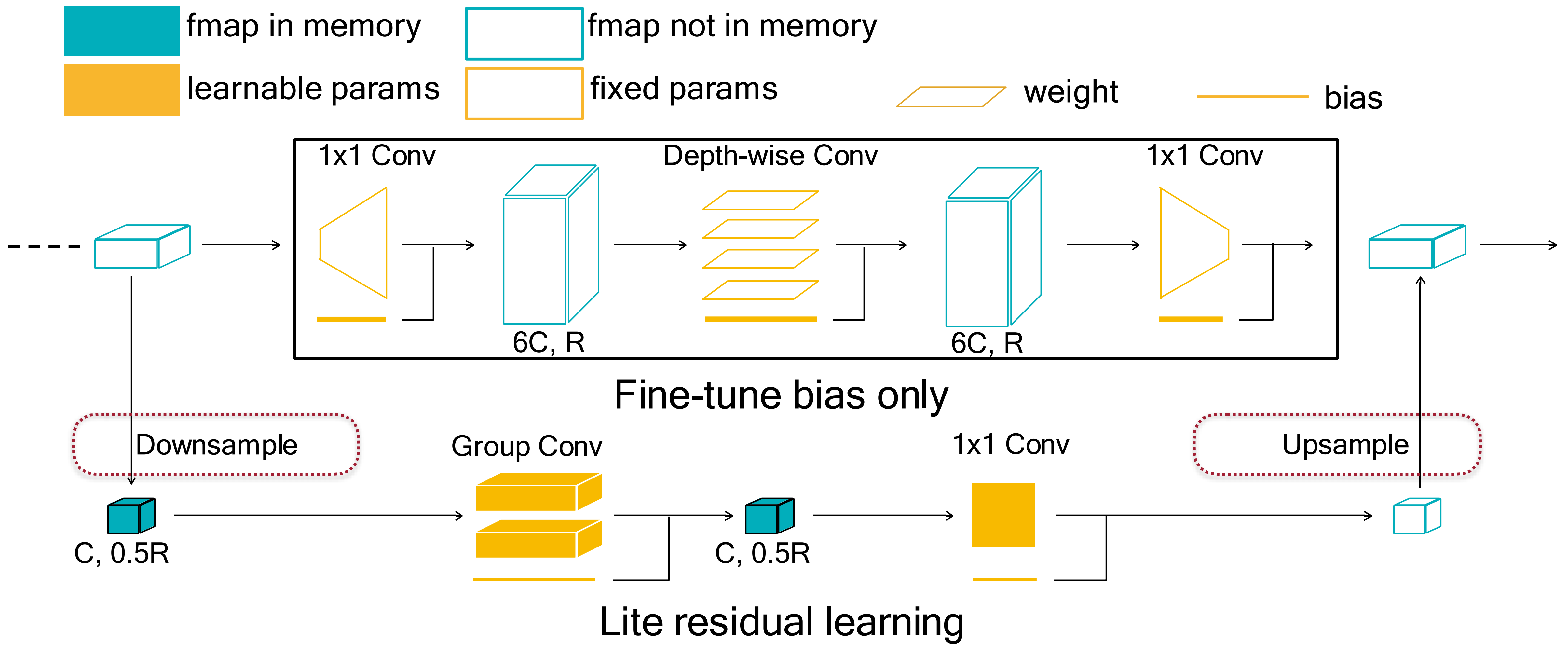
TinyTL: Lite Residual Learning



- Add lite residual modules to increase model capacity
- Key principle - keep activation size small

TinyTL: Reduce Activations, Not Trainable Parameters for Efficient On-Device Learning [Cai *et al.*, NeurIPS 2020]

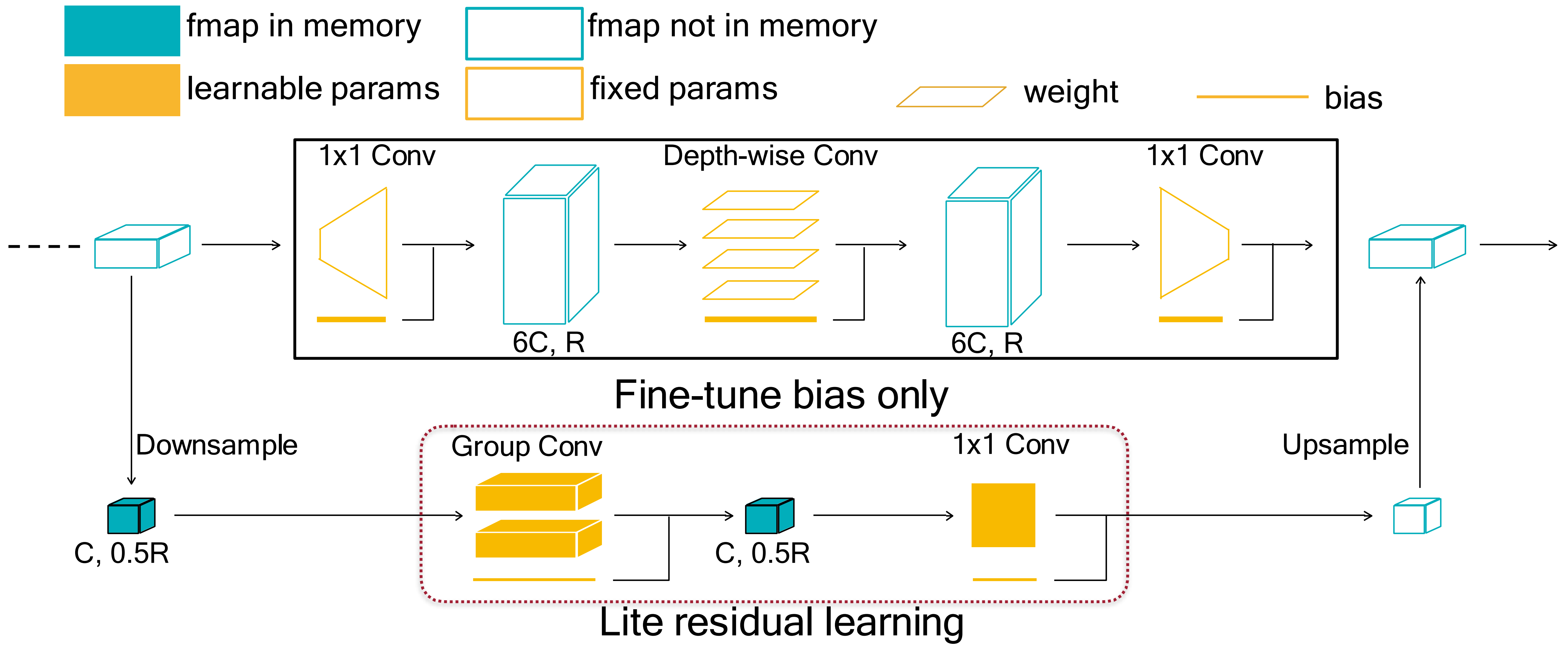
TinyTL: Lite Residual Learning



- Add lite residual modules to increase model capacity
- Key principle - keep activation size small
    1. Reduce the resolution

TinyTL: Reduce Activations, Not Trainable Parameters for Efficient On-Device Learning [Cai *et al.*, NeurIPS 2020]

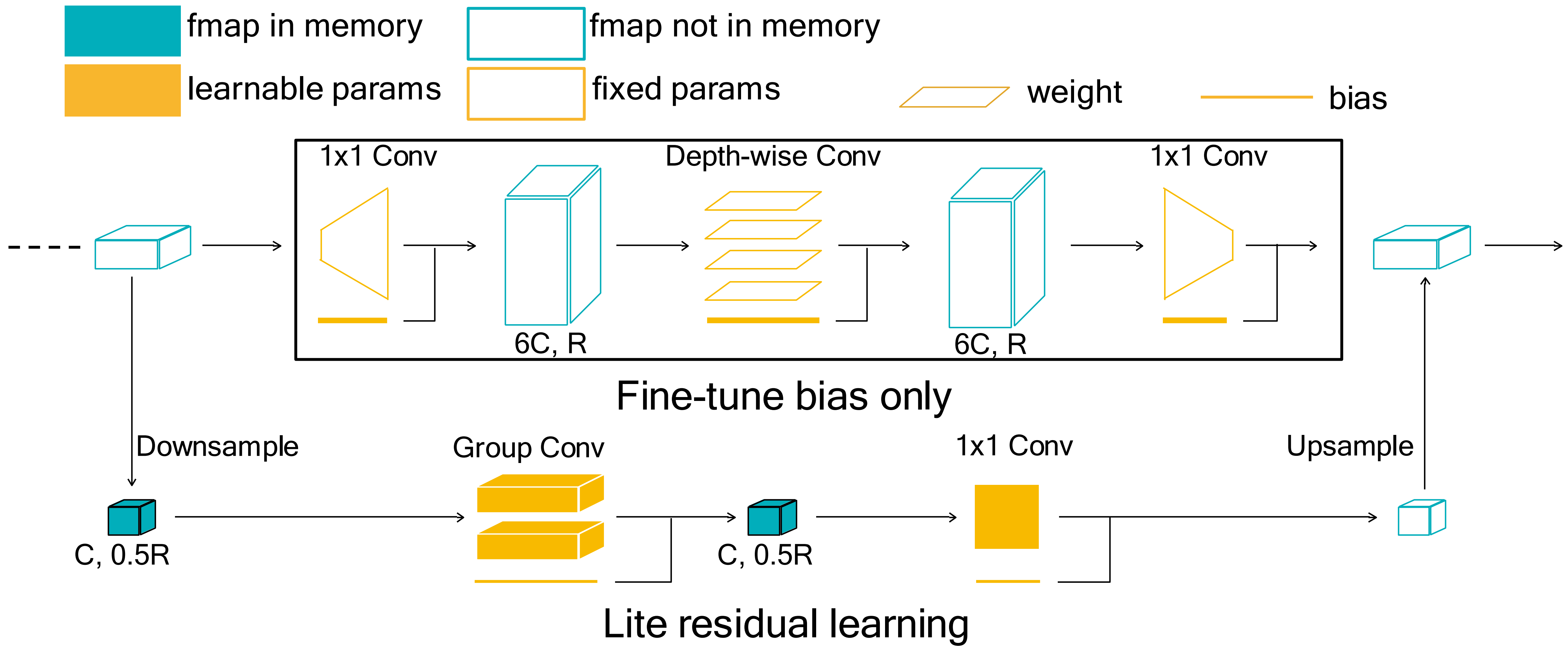
TinyTL: Lite Residual Learning



- Add lite residual modules to increase model capacity
- Key principle - keep activation size small
  1. Reduce the resolution
  2. Avoid inverted bottleneck

TinyTL: Reduce Activations, Not Trainable Parameters for Efficient On-Device Learning [Cai *et al.*, NeurIPS 2020]

TinyTL: Lite Residual Learning

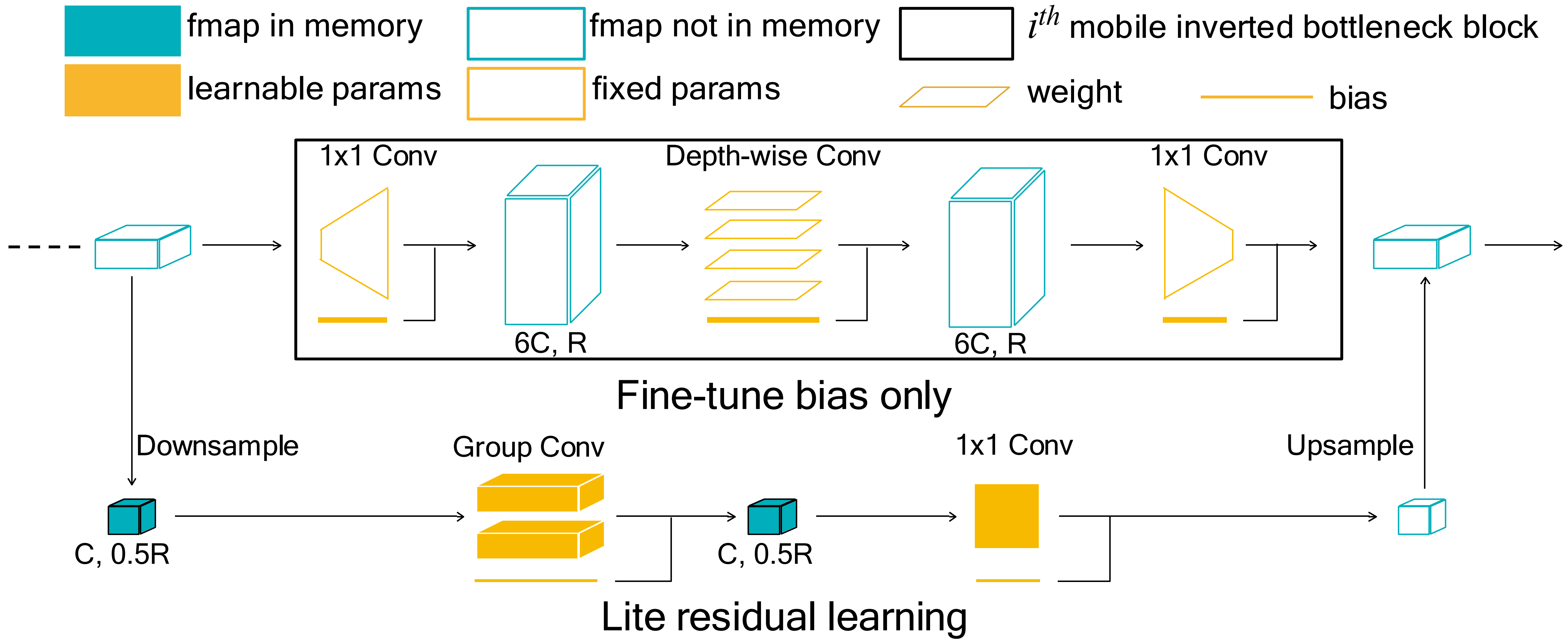


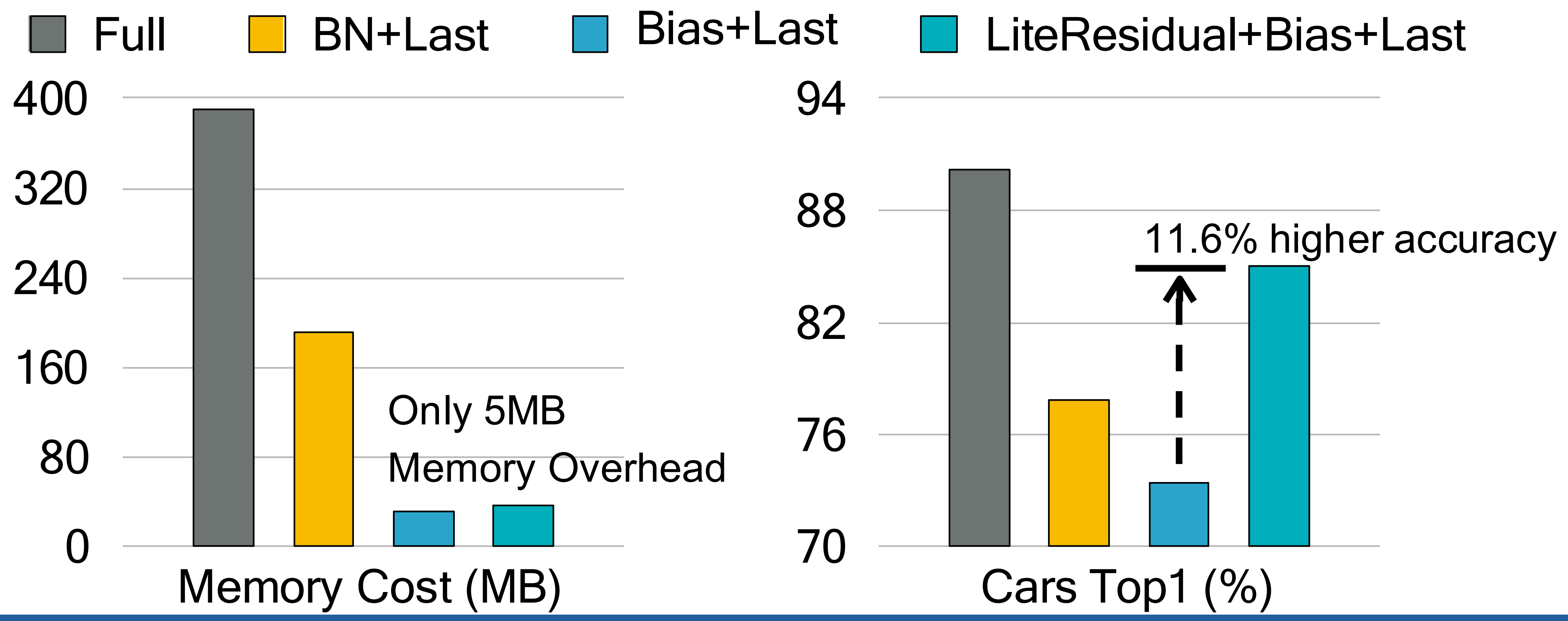
- Add lite residual modules to increase model capacity
- Key principle - keep activation size small
  1. Reduce the resolution
  2. Avoid inverted bottleneck

(1/6 channel, 1/2 resolution, 2/3 depth => ~4% activation size)

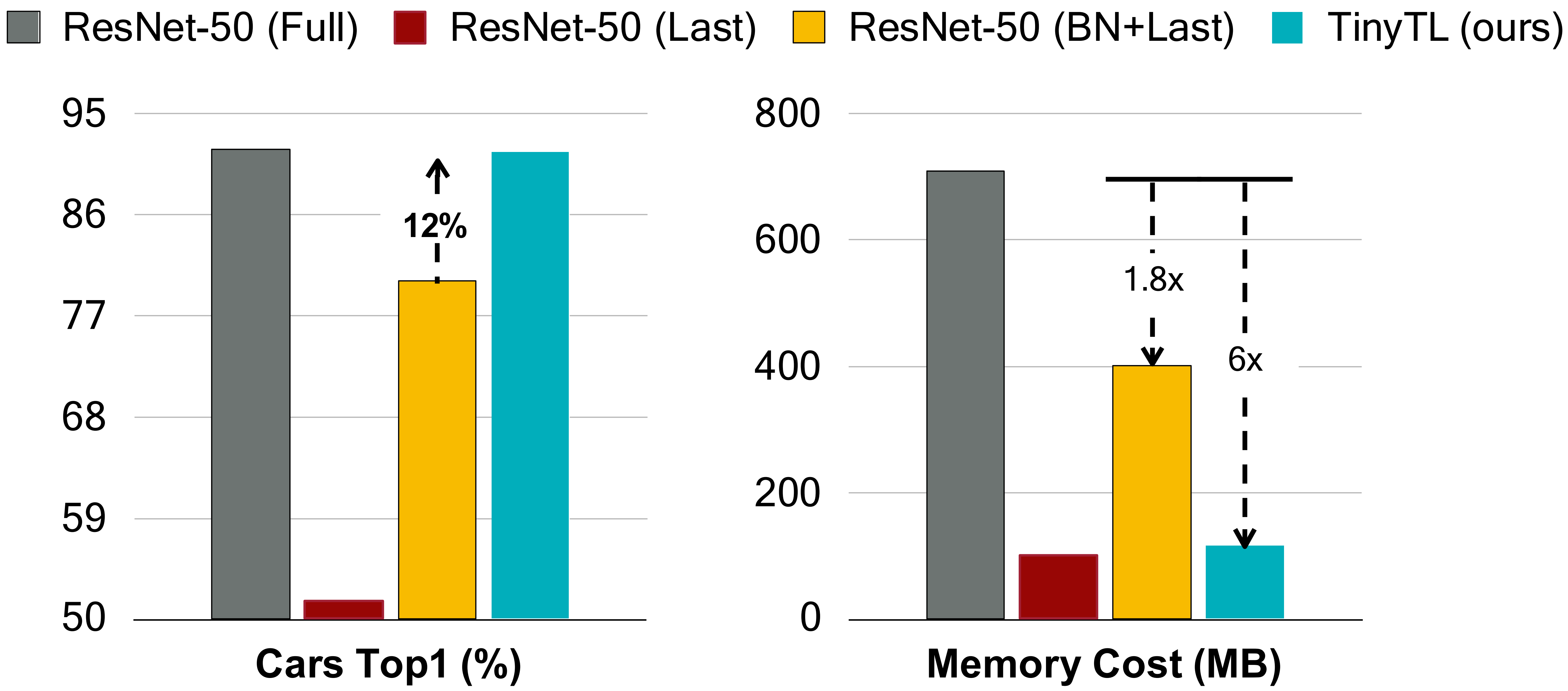
TinyTL: Reduce Activations, Not Trainable Parameters for Efficient On-Device Learning [Cai *et al.*, NeurIPS 2020]

TinyTL: Lite Residual Learning





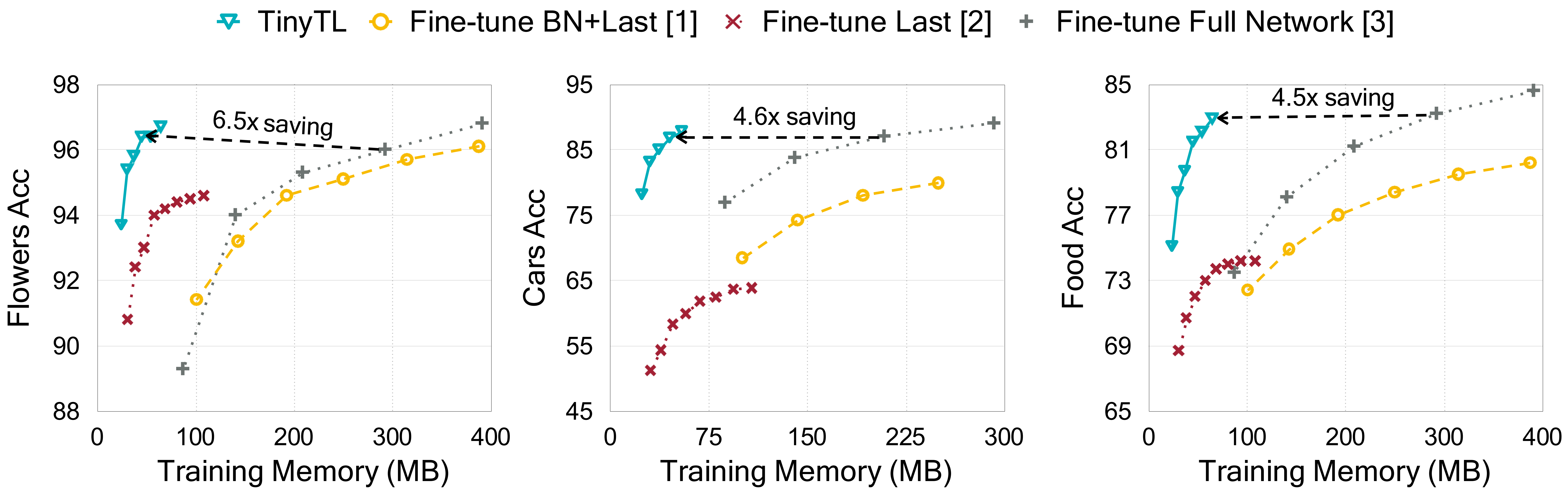
TinyTL: Memory-Efficient Transfer Learning



- Full: Fine-tune the full network. Better accuracy but highly inefficient.
- Last: Only fine-tune the last classifier head. Efficient but the capacity is limited.
- BN+Last: Fine-tune the BN layers and the last layer. Parameter-efficient, **but the memory saving is limited. Significant accuracy loss.**
- TinyTL: fine-tune bias only + lite residual learning: high accuracy, large memory saving

TinyTL: Reduce Activations, Not Trainable Parameters for Efficient On-Device Learning [Cai *et al.*, NeurIPS 2020]

TinyTL: Up to 6.5x Memory Saving



Backbone: ProxylessNAS-Mobile, Scanning over different resolutions

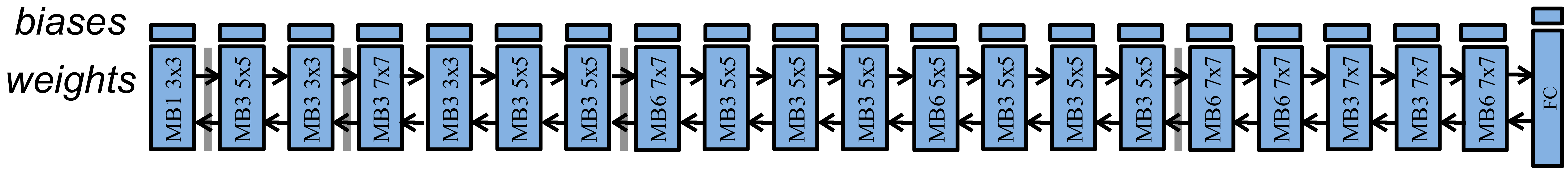
- TinyTL provides up to **6.5x** memory saving **without accuracy loss**.

TinyTL: Reduce Activations, Not Trainable Parameters for Efficient On-Device Learning [Cai *et al.*, NeurIPS 2020]

Lecture Plan

1. Federated learning and the deep leakage from gradients
2. Pruning, quantization and knowledge distillation
3. Memory bottleneck of on-device training
4. Tiny transfer learning (TinyTL)
5. **Sparse back-propagation (SparseBP)**

Dense, Full Back-Propagation



Model: ProxylessNAS-Mobile

Updating the whole model is **too expensive**:

- Need to save all intermediate activations (quite large)

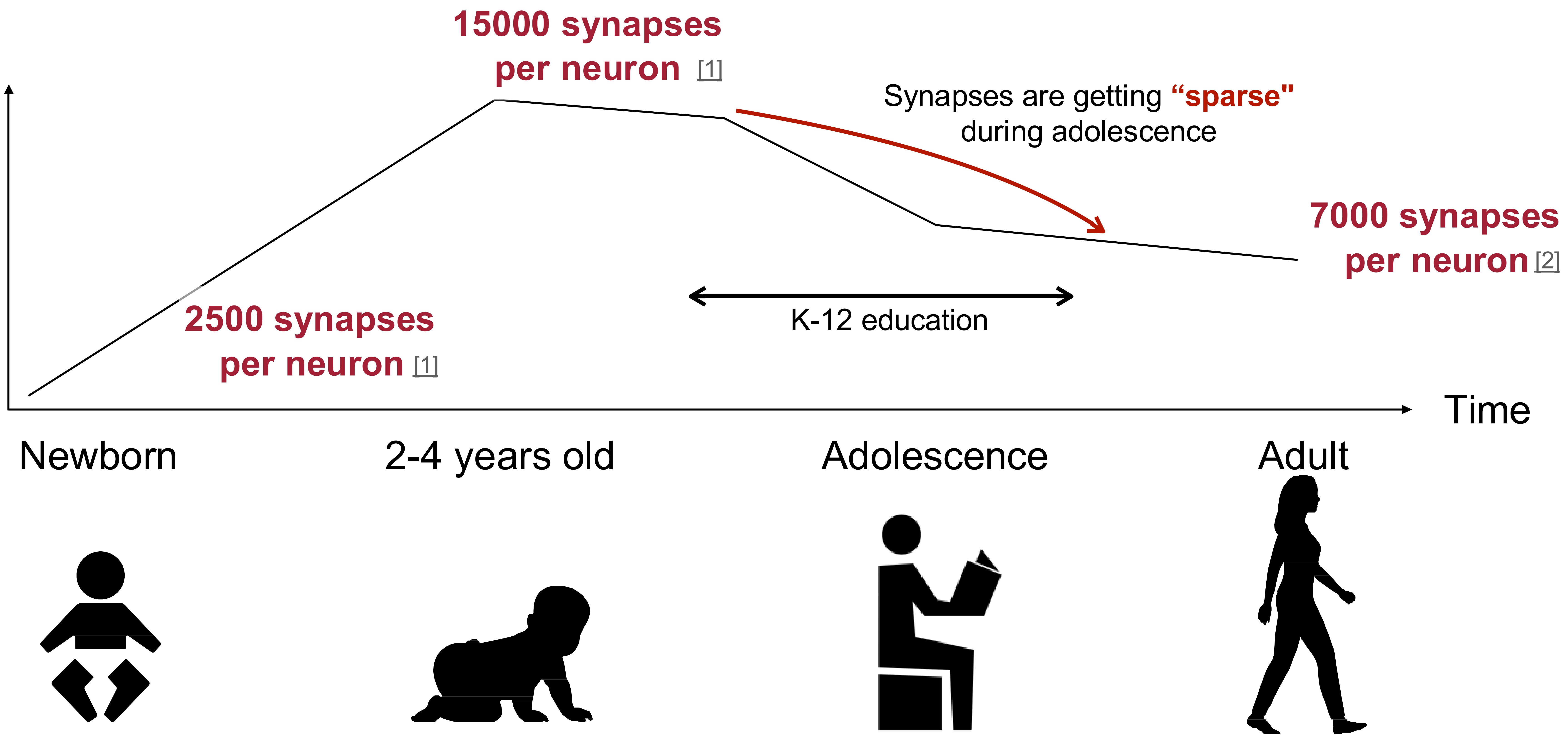
Forward: $\mathbf{a}_{i+1} = \mathbf{a}_i \mathbf{W}_i + \mathbf{b}_i$

Backward: $\dfrac{\partial L}{\partial \mathbf{W}_i} = \mathbf{a}_i^T \dfrac{\partial L}{\partial \mathbf{a}_{i+1}}, \quad \dfrac{\partial L}{\partial \mathbf{a}_i} = \dfrac{\partial L}{\partial \mathbf{a}_{i+1}} \mathbf{w}_i^T$

- Inference does not need to store activations, training does.
- Activations grows linearly with batch size.

TinyTL: Reduce Memory, Not Parameters for Efficient On-Device Learning [Cai et al, NeurIPS 2020]

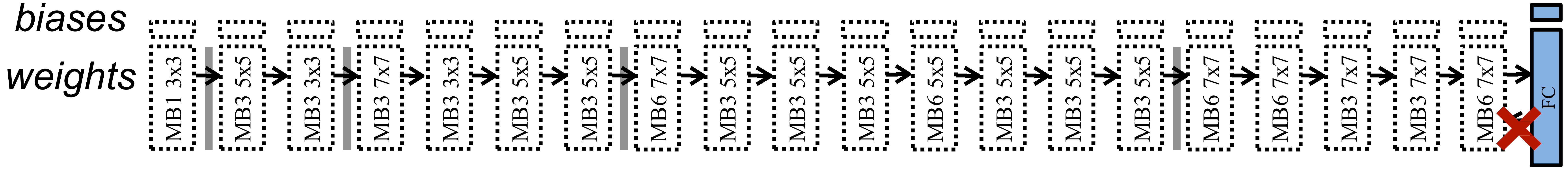
Sparse Learning



1 Do We Have Brain to Spare? [Drachman DA, Neurology 2004]
2 Peter Huttenlocher (1931–2013) [Walsh, C. A., Nature 2013]

Slide Inspiration: Alila Medical Media

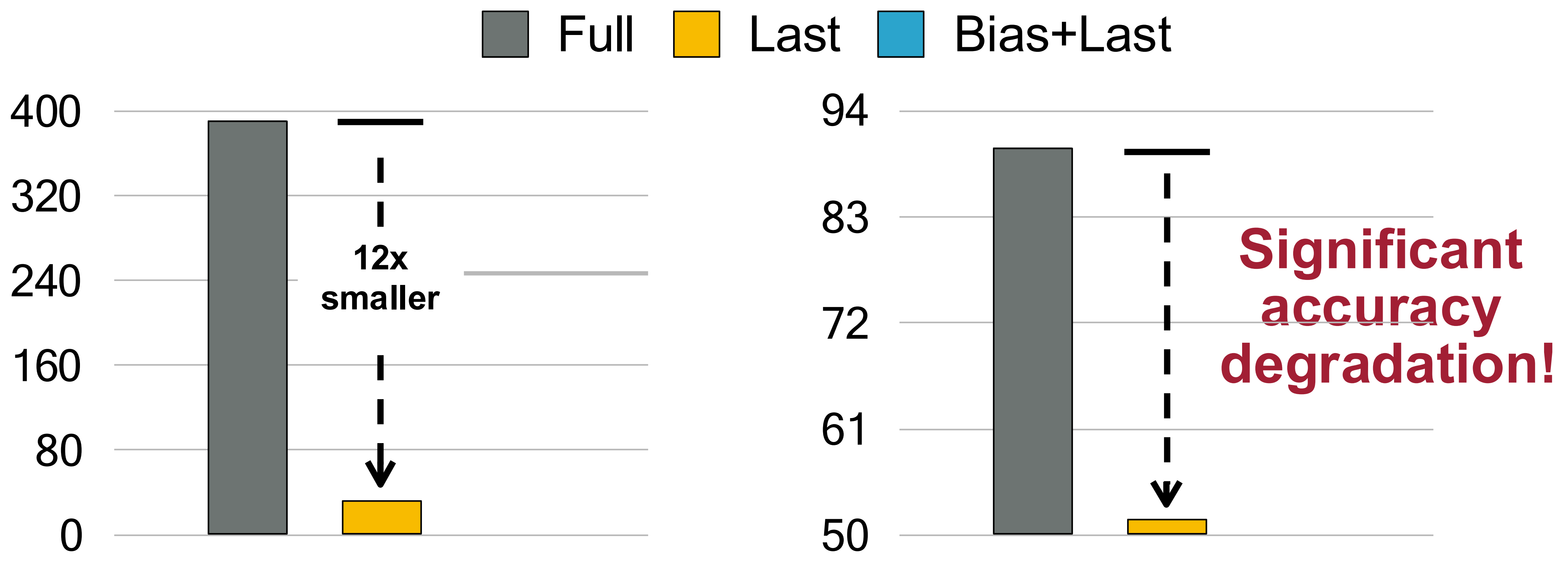
Last-Layer-Only Back-Propagation



Model: ProxylessNAS-Mobile

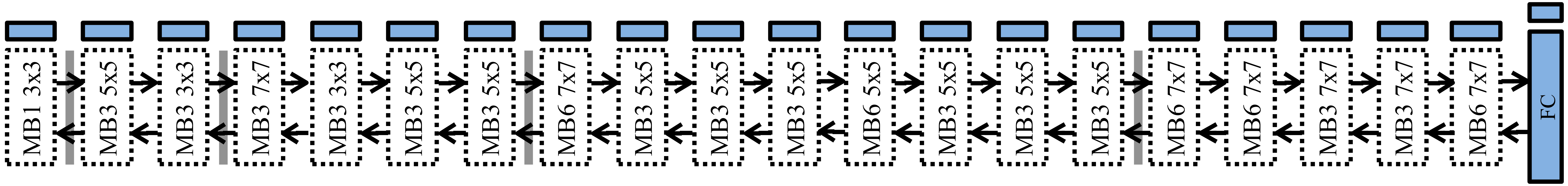
Updating only the last layer is cheap

- No need to back propagate to previous layers
- But, accuracy drops significantly



TinyTL: Reduce Memory, Not Parameters for Efficient On-Device Learning [Cai et al, NeurIPS 2020]

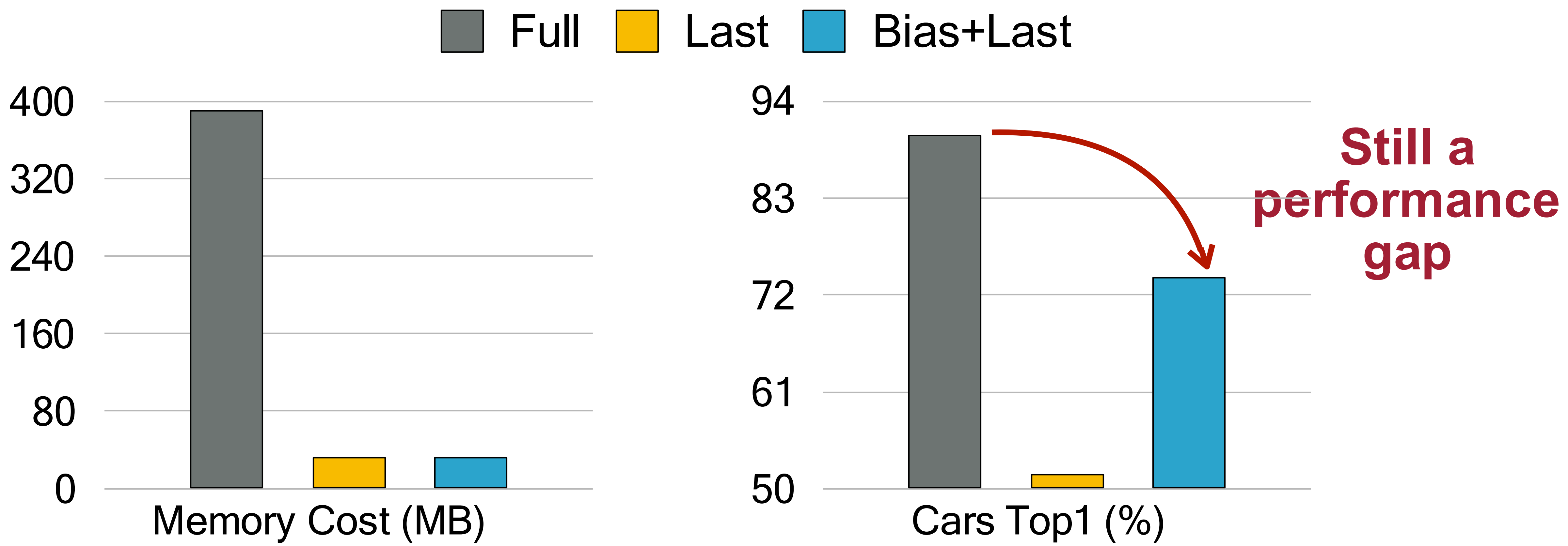
Bias-Only Back-Propagation



Model: ProxylessNAS-Mobile

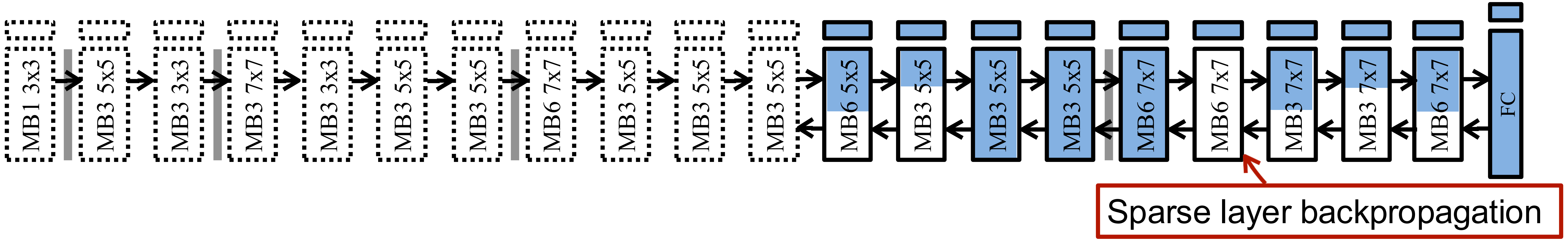
Updating the only the bias part
- No need to store the activations
- Back propagating to the first layer.

$$d\mathbf{W} = f(\mathbf{X}, d\mathbf{Y})$$
$$d\mathbf{b} = f(d\mathbf{Y})$$



TinyTL: Reduce Memory, Not Parameters for Efficient On-Device Learning [Cai et al, NeurIPS 2020]

Sparse Back-Propagation

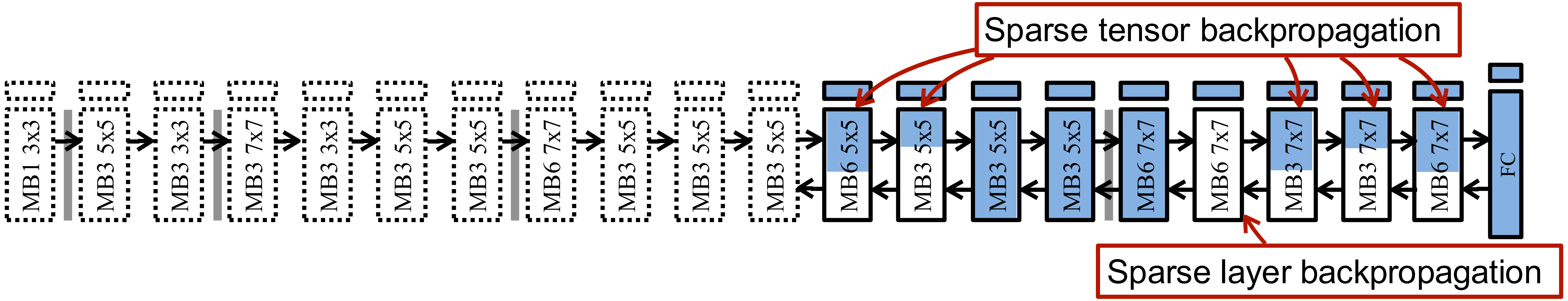


Model: ProxylessNAS-Mobile

Use sparse back-propagation to train the model

- Some **layers are not as important** as others

Sparse Back-Propagation

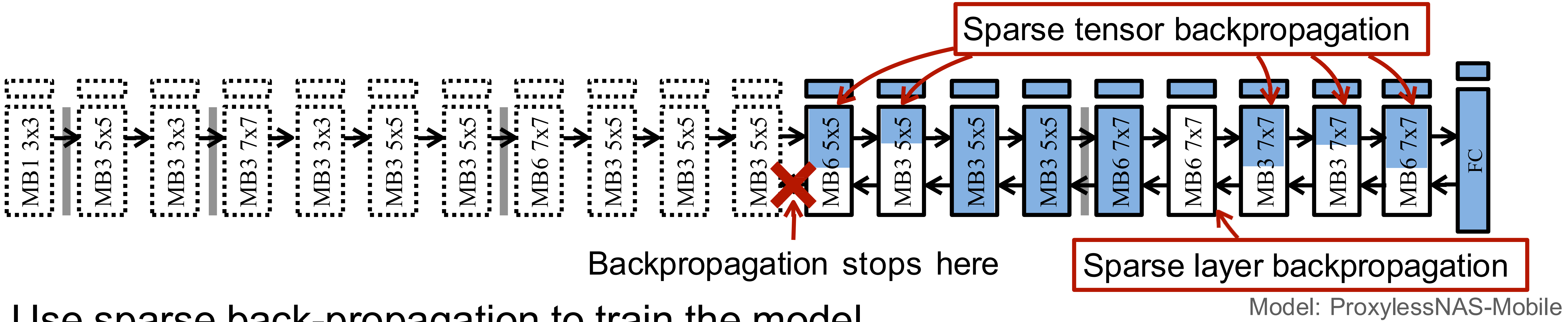


Model: ProxylessNAS-Mobile

Use sparse back-propagation to train the model

- Some layers are not as important as others
- Some **<u>channels</u> are not as important as** others

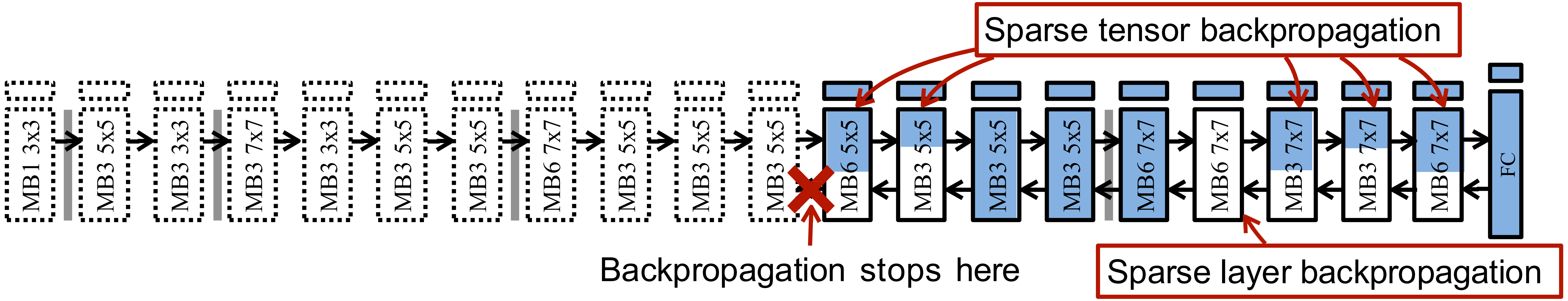
Sparse Back-Propagation



Use sparse back-propagation to train the model

- Some layers are not as important as others
- Some channels are not as important as others
- **No need to back-propagate to the early layers**

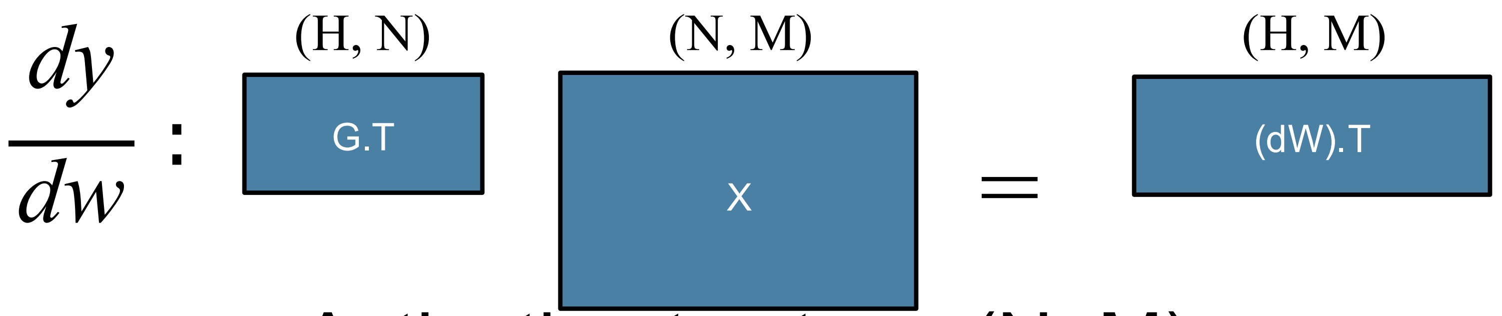
Sparse Back-Propagation

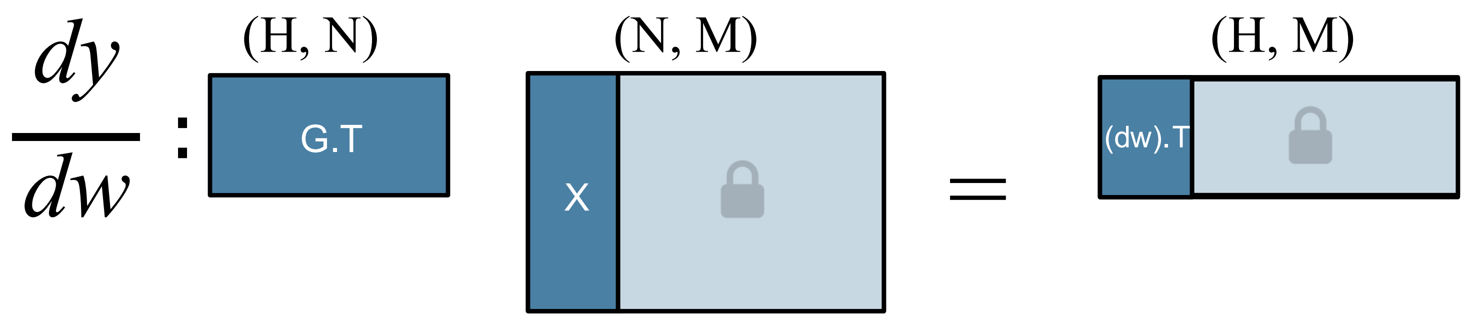


Model: ProxylessNAS-Mobile

Use sparse back-propagation to train the model

- Some layers are not as important as others
- Some channels are not as important as others
- No need to back-propagate to the early layers
- **Only need to store and compute on a subset of the activations.**



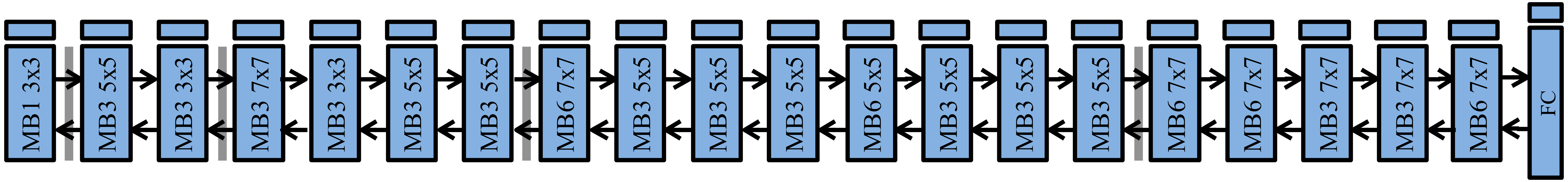


Activation to store: (N, M)
FLOPs: (M * H * N)

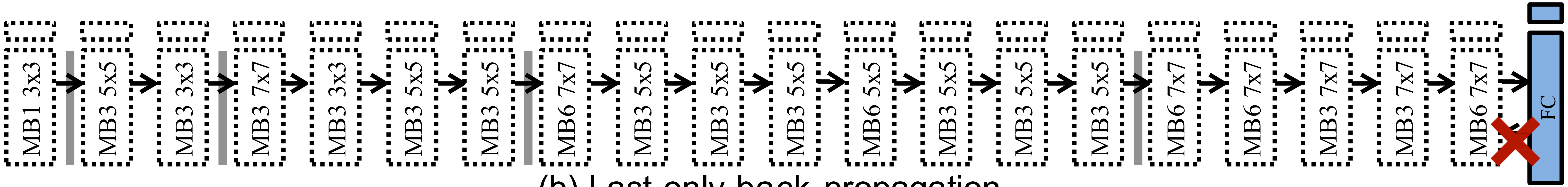
Reduce by 4x

Activation to store: (N, 0.25 * M)
FLOPs: (0.25 * M * H * N)

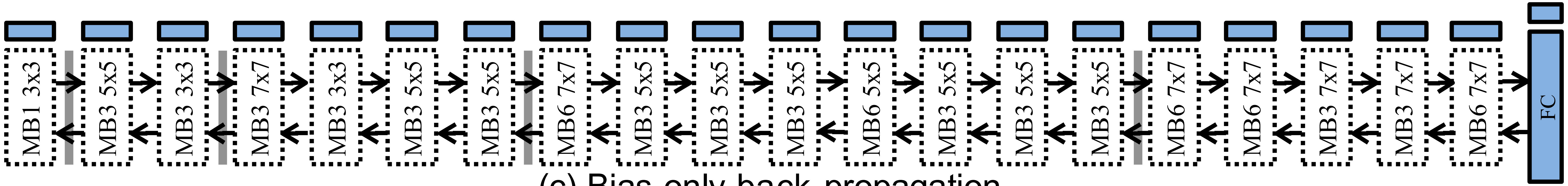
Comparison



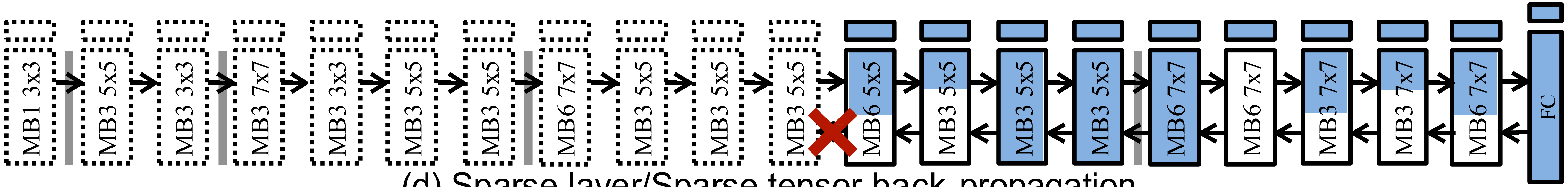
(a) Full back-propagation



(b) Last-only back-propagation



(c) Bias-only back-propagation

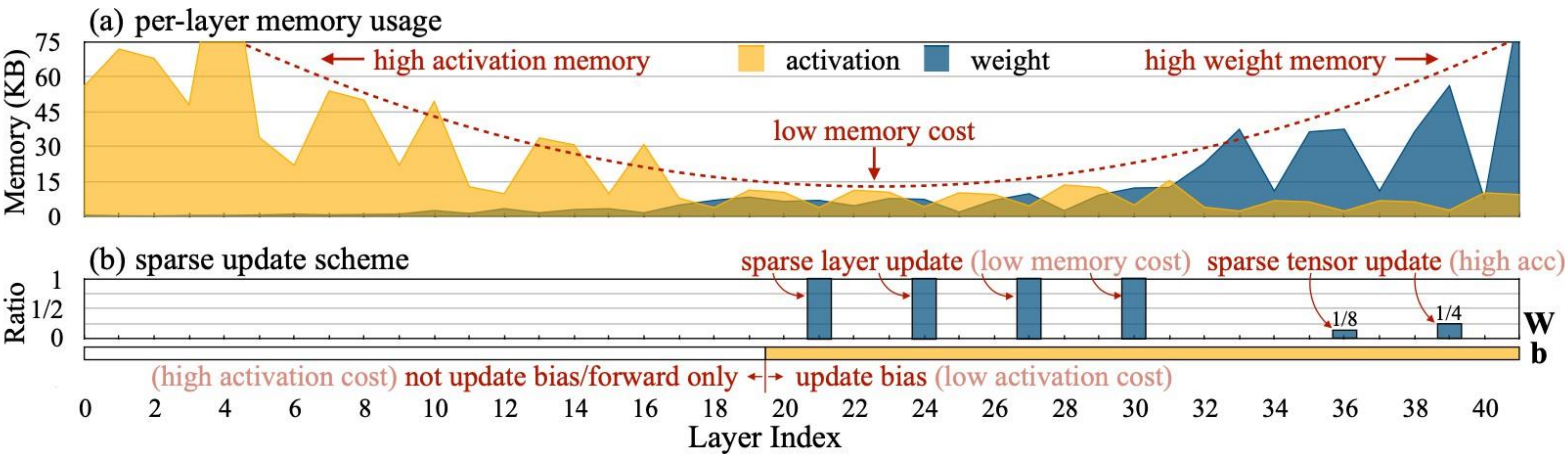


(d) Sparse layer/Sparse tensor back-propagation

Find Layers to Update by Contribution Analysis

Which layer to update?

- The **activation cost** is high for the starting layers; the **weight cost** is high for the later layers; the **overall memory** cost is low for the middle layers.
- We update biases for the later layers (related to activation only), and weights for the intermediate layers (related to activation and weights)

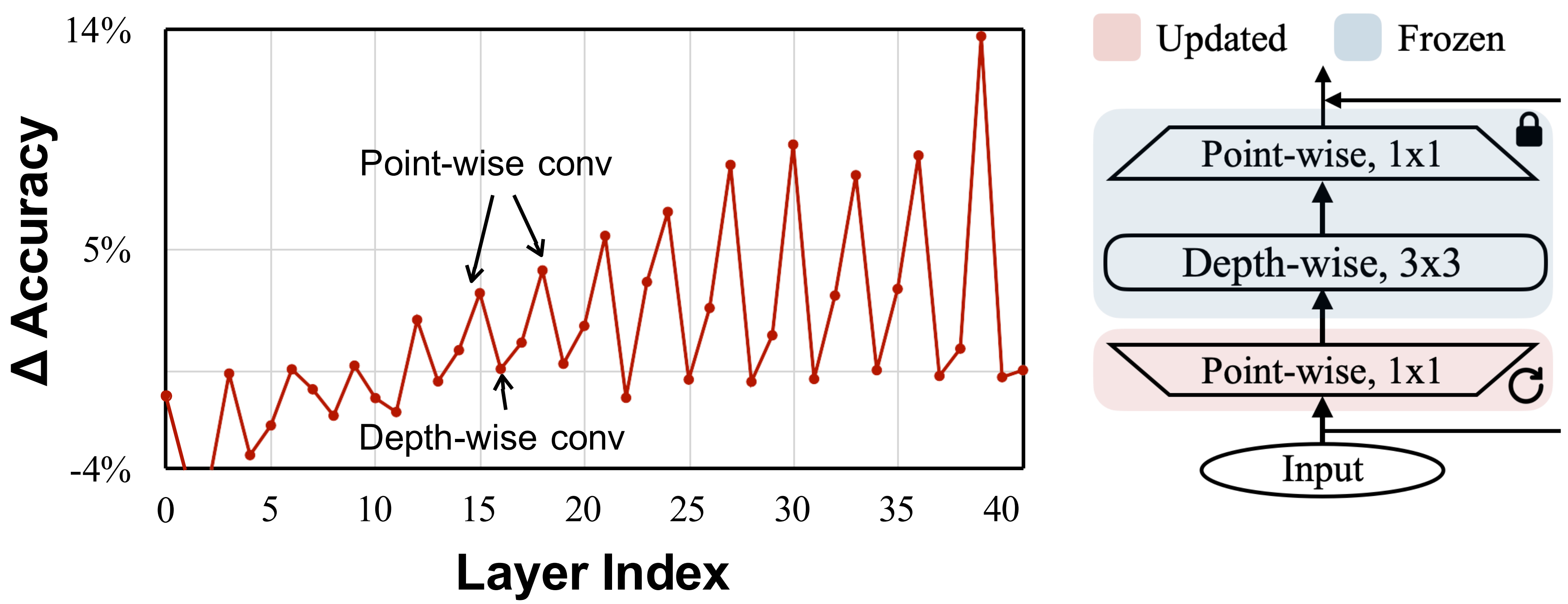


Contribution Analysis

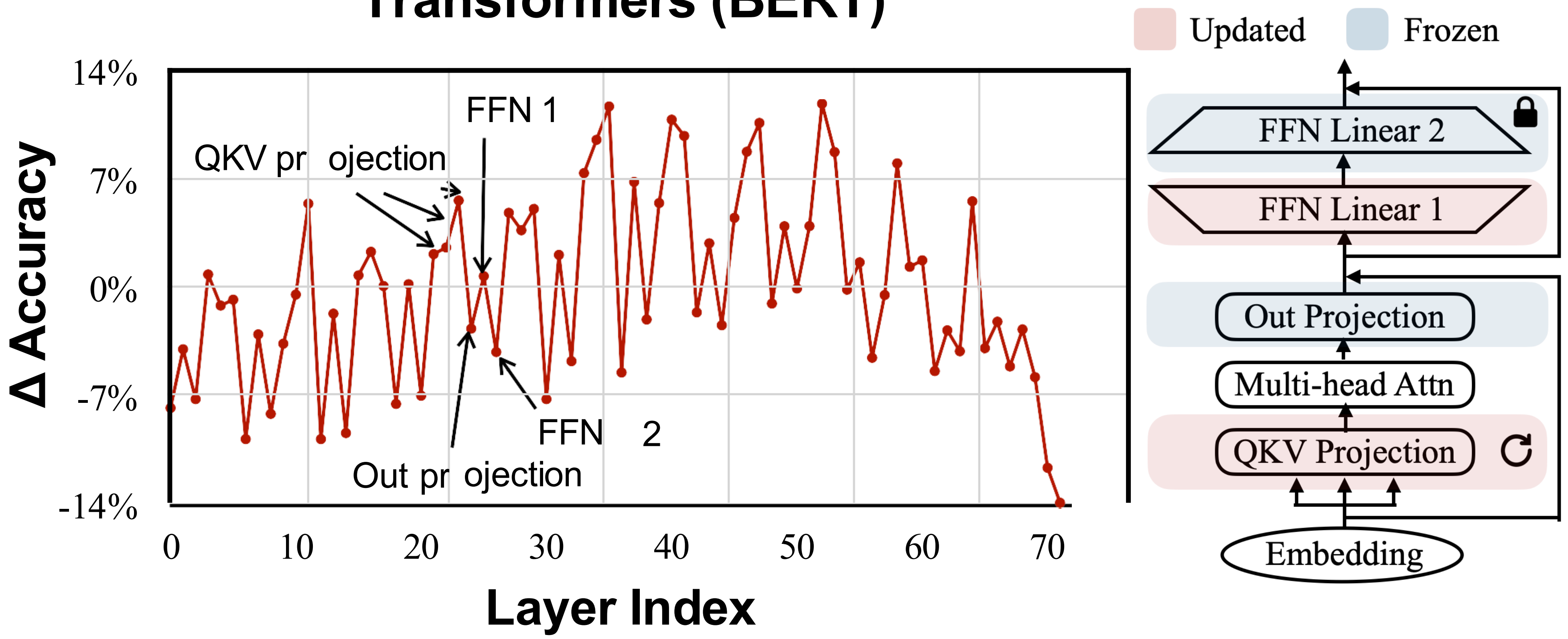
Which layer to update?

- Contribution Analysis: fine-tune only one layer on a downstream task to measure the accuracy improvement (Accuracy) as contributions.
- Only fine-tune the **layers with large Accuracy** (contributes more to performance)

CNN model (MobileNetV2)



Transformers (BERT)

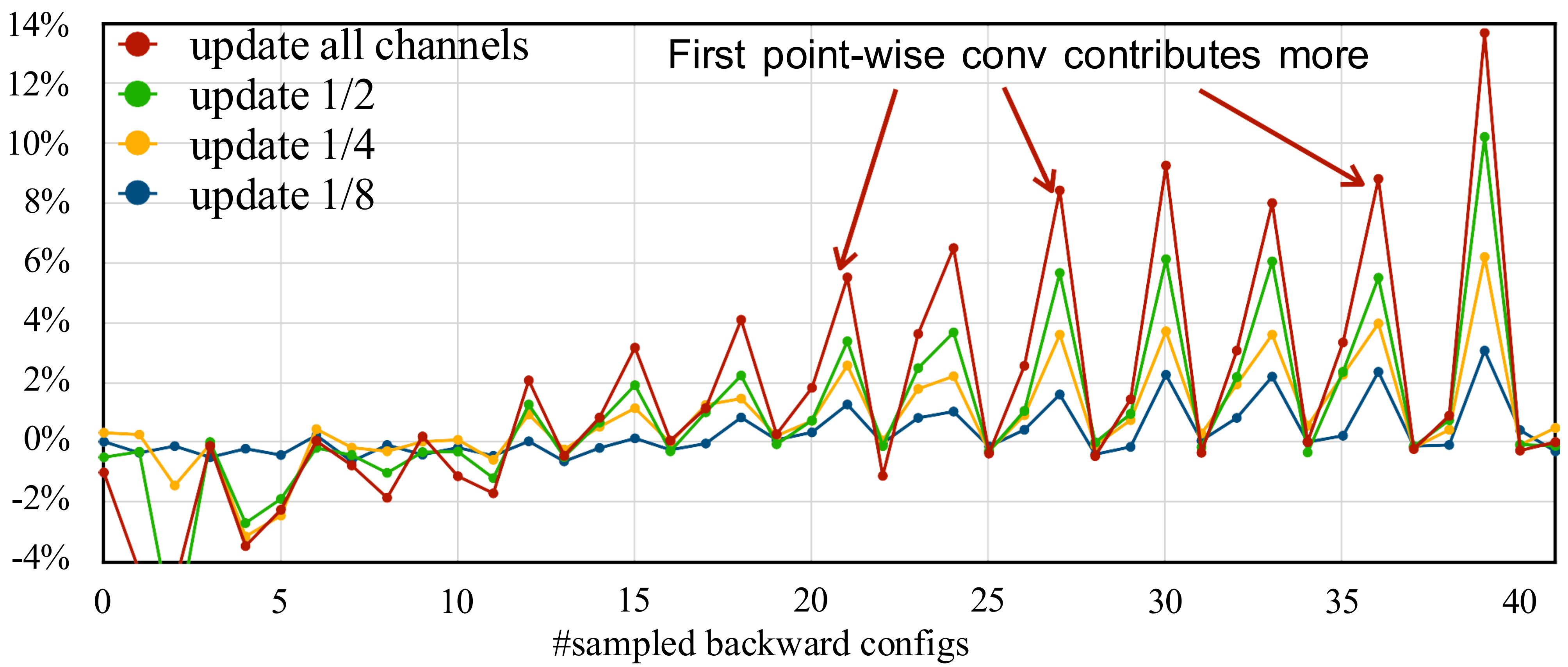


Different models prefer *different* layers for fine-tuning

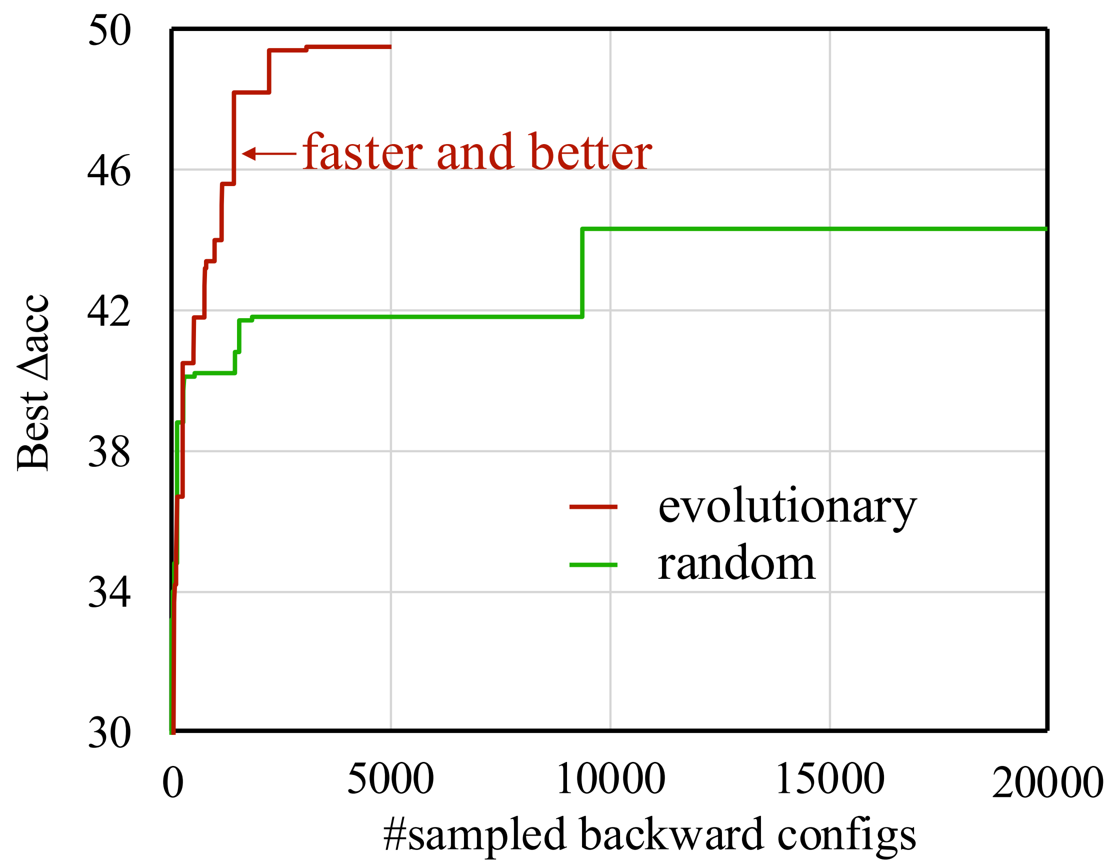
- MobilenetV2 prefers **first depth-wise conv**.
- BERT prefers **QKV projection** and **first FFN** layers.

Contribution Analysis

Which layer to update?



(a) Contribution analysis



(a) Evolution Search

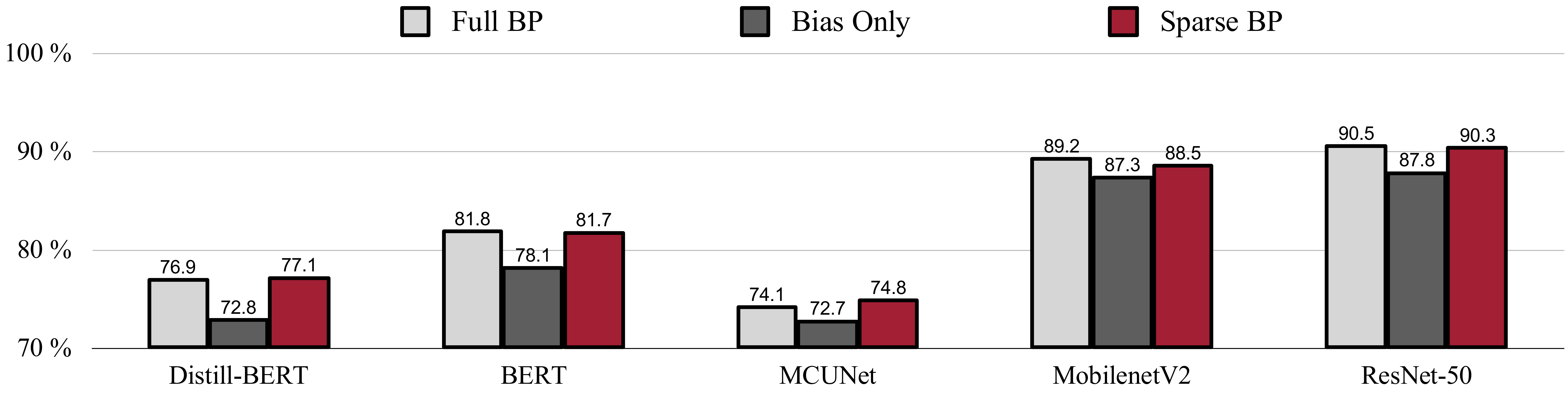
- Use **evolutionary search** to find the sparse back-propagation scheme.

$$k^*, \mathbf{i}^*, \mathbf{r}^* = \max_{k,\mathbf{i},\mathbf{r}} (\Delta\mathrm{acc}_{\mathbf{b}[:k]} + \sum_{i \in \mathbf{i}, r \in \mathbf{r}} \Delta\mathrm{acc}_{\mathbf{W}i,r}) \quad \text{s.t. Memory}(k, \mathbf{i}, \mathbf{r}) \leq \text{constraint}$$

- Thus we can train the model on the edge with low memory cost while achieving high accuracy.

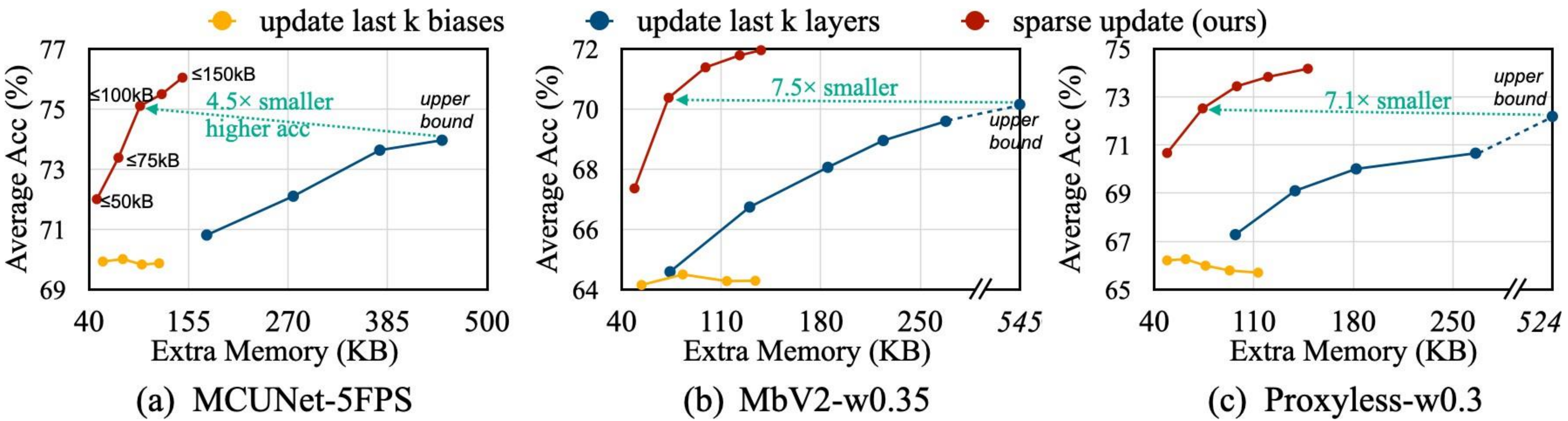
Accuracy of Sparse Back-Propagation

Well maintains the accuracy



- The accuracy on DistillBERT and BERT is average from GLUE Benchmark.
- The accuracy on MCUNet, MobilenetV2, ResNet-50 is average from TinyTL Benchmark.
- Sparse-BP demonstrates **on-par performance with Full-BP** on both vision and language tasks.

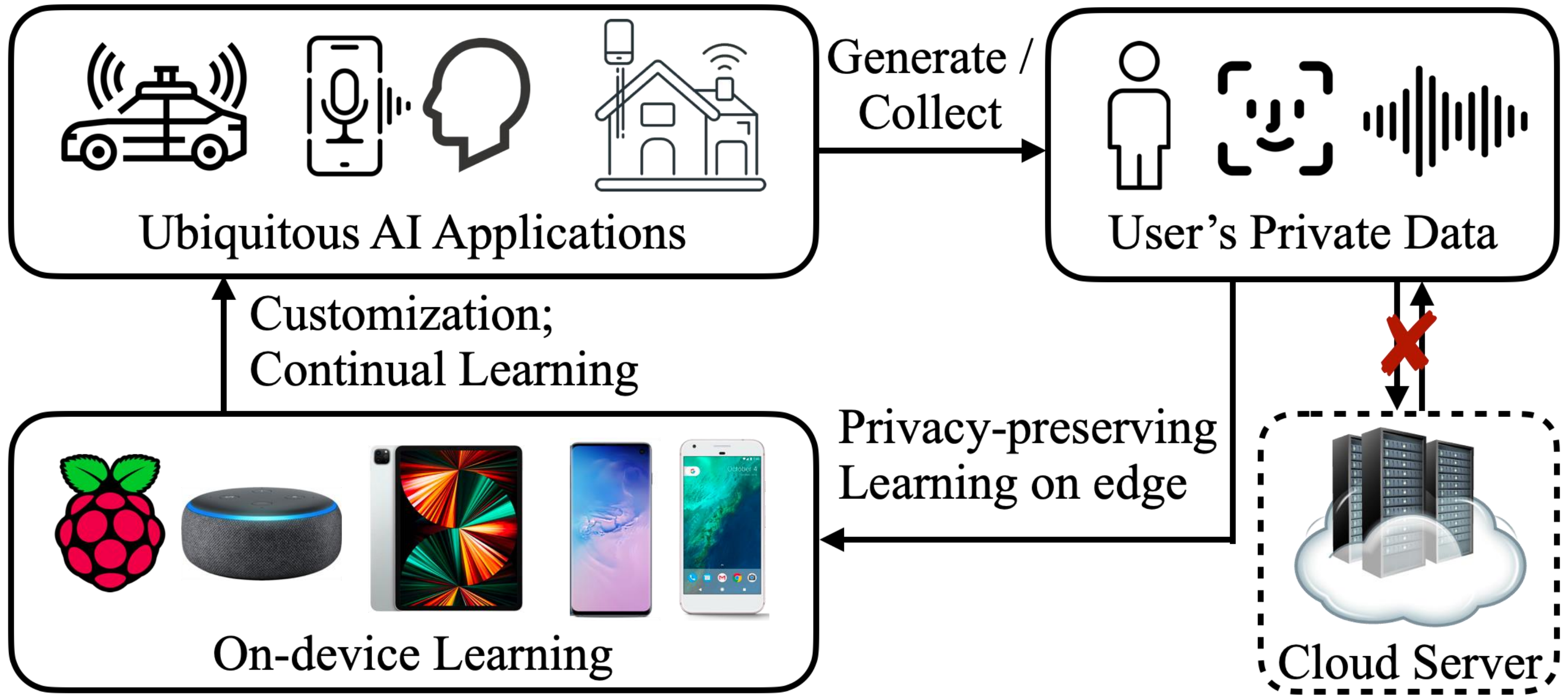
Sparse BP: Lower Memory, Higher Accuracy



Sparse back-propagation can achieve higher transfer learning accuracy using **4.5-7.5x** smaller extra memory.

Takeaways

1. Gradient is not safe to share. Staying local is important.
2. Three techniques to make model smaller: pruning, quantization and knowledge distillation.
3. CNN's training memory bottleneck is the activation.
4. Efficient transfer learning with bias-only and lite-residual.
5. Full-update is too expensive and using sparse back-propagation for on-device training.



References

- Return of the devil in the details: Delving deep into convolutional nets [Chatfield. 2014]
- Do better imagenet models transfer better? [Kornblith. 2019]
- TinyTL: Reduce Activations, Not Trainable Parameters for Efficient On-Device Learning [Cai et al. NeurIPS 2020]
- K for the Price of 1: Parameter-efficient Multi-task and Transfer Learning [Mudrarkarta et al., ICLR 2019]
- Do We Have Brain to Spare? [Drachman et al. 2004]
- Peter Huttenlocher (1931–2013) [Walsh. 2013]
- MCUNet: Tiny Deep Learning on IoT Devices [Lin et al 2020]