MIT HAN LAB

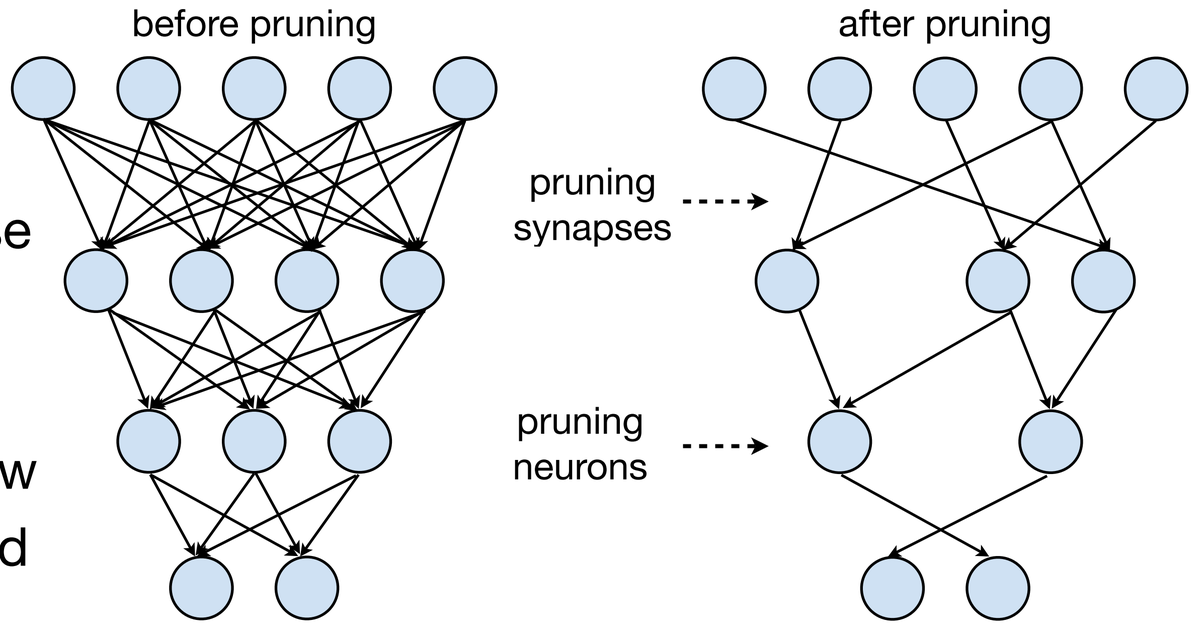
Lecture 04
Pruning and Sparsity
Part II

Song Han
songhan@mit.edu

Lecture Plan

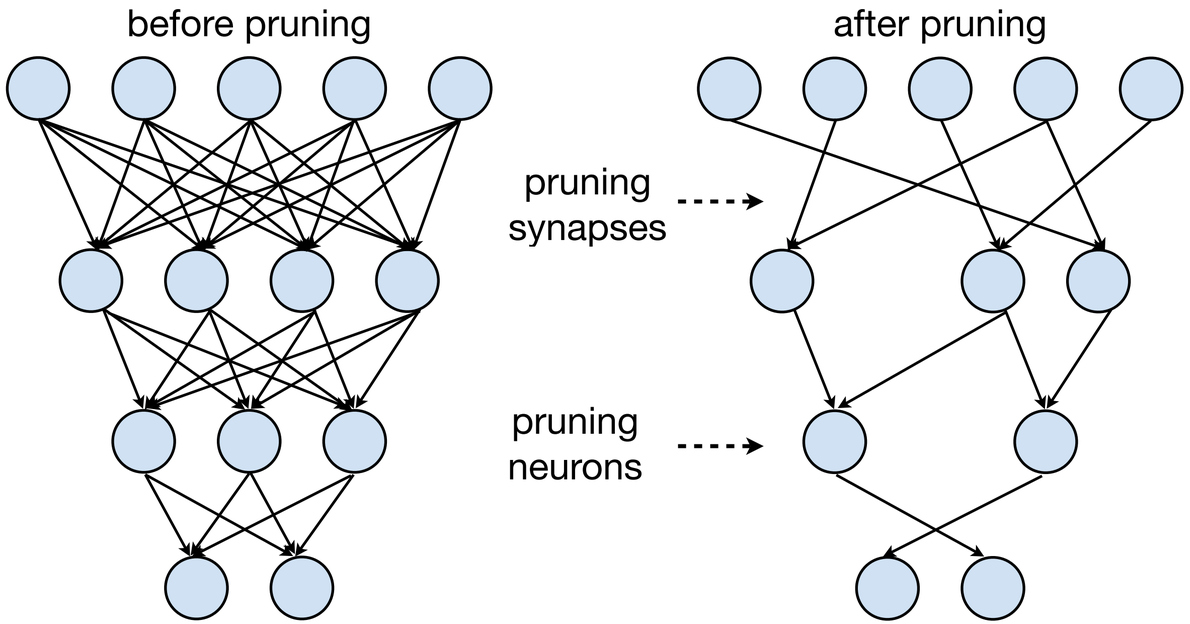
Today we will:

1. Go through all steps of pruning, and introduce how to select ***pruning ratio*** and how to ***fine-tune*** in neural network pruning.
2. Introduce ***Lottery Ticket Hypothesis*** in neural network pruning which shows that training a sparse neural network from scratch is possible.
3. Introduce the ***system support for the sparsity*** in the neural network after pruning, and elaborate how to translate the computation reduction to measured speedup on general-purpose and specialized hardware.



Neural Network Pruning

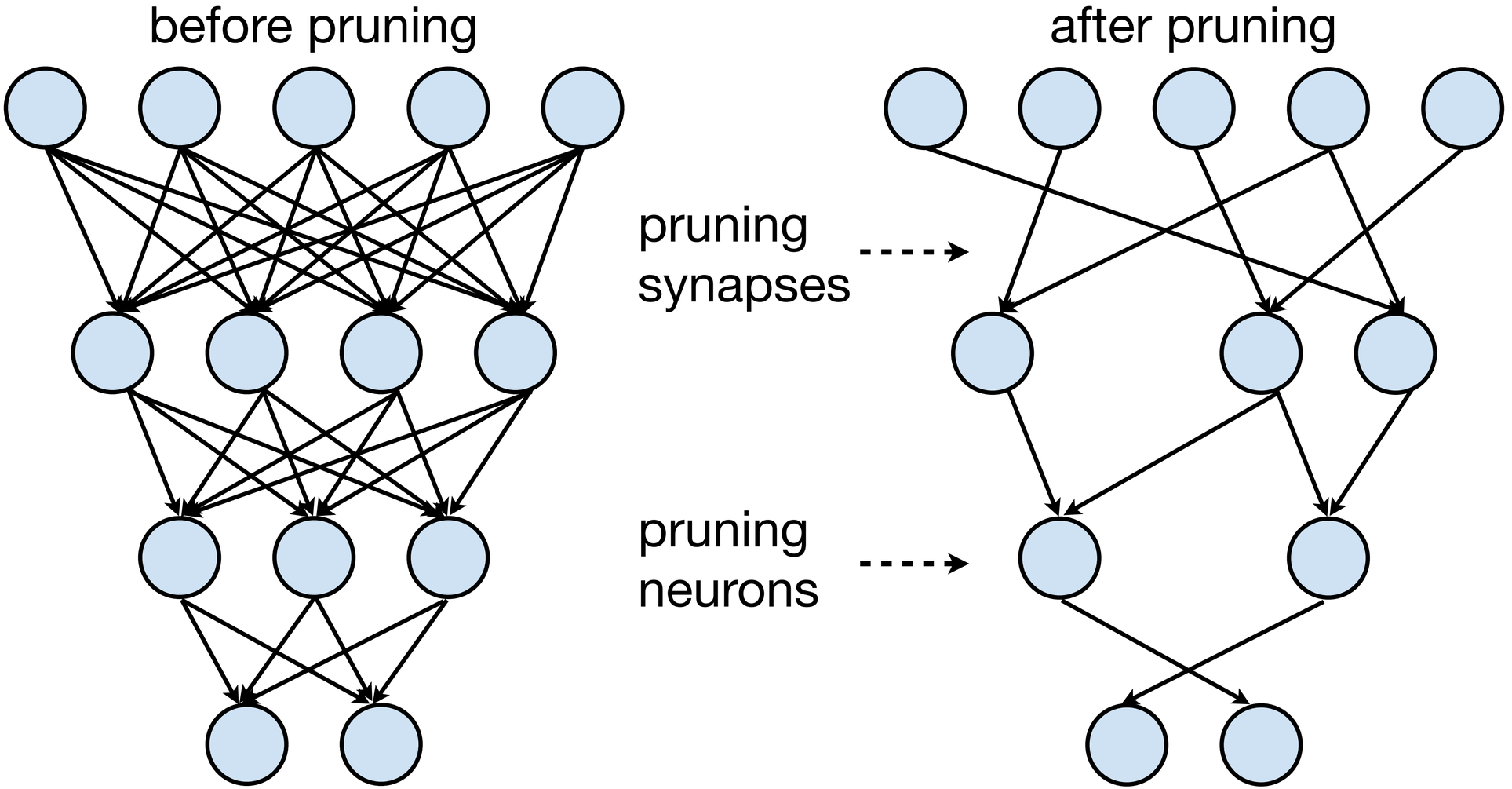
- **Introduction to Pruning**
  - What is pruning?
  - How should we formulate pruning?
- **Determine the Pruning Granularity**
  - In what pattern should we prune the neural network?
- **Determine the Pruning Criterion**
  - What synapses/neurons should we prune?
- **Determine the Pruning Ratio**
  - What should target sparsity be for each layer?
- **Fine-tune/Train Pruned Neural Network**
  - How should we improve performance of pruned models?



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

Neural Network Pruning

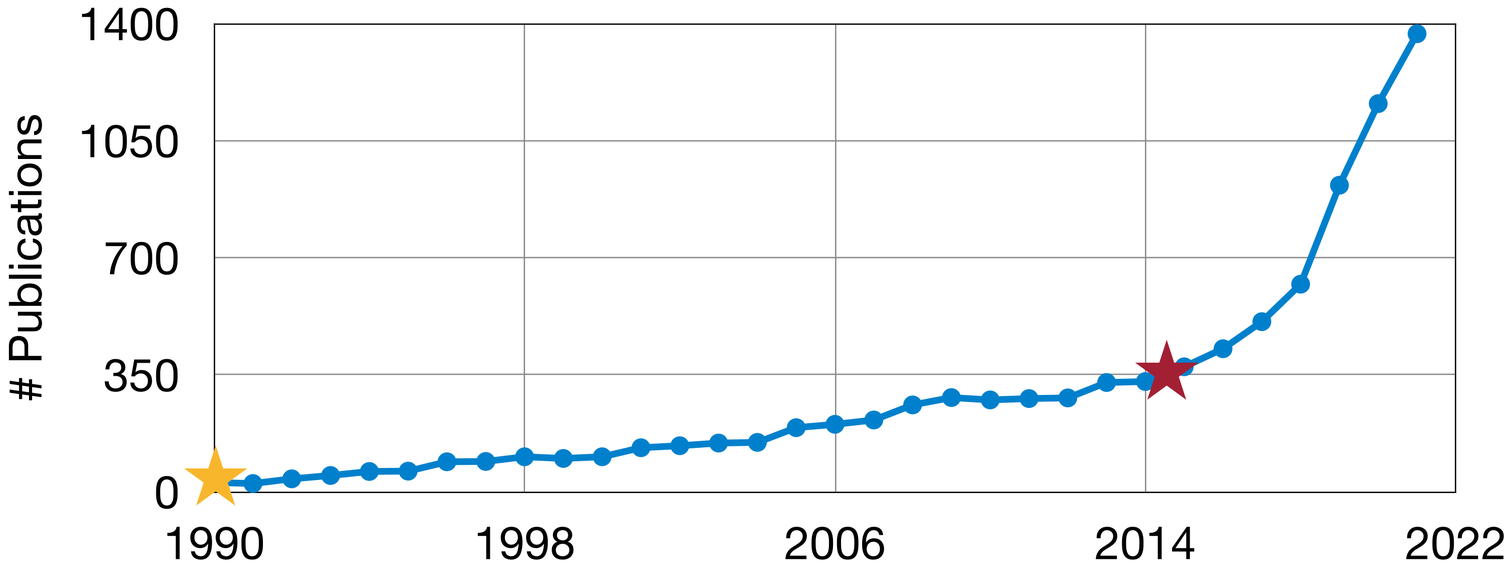
Make neural network smaller by removing synapses and neurons

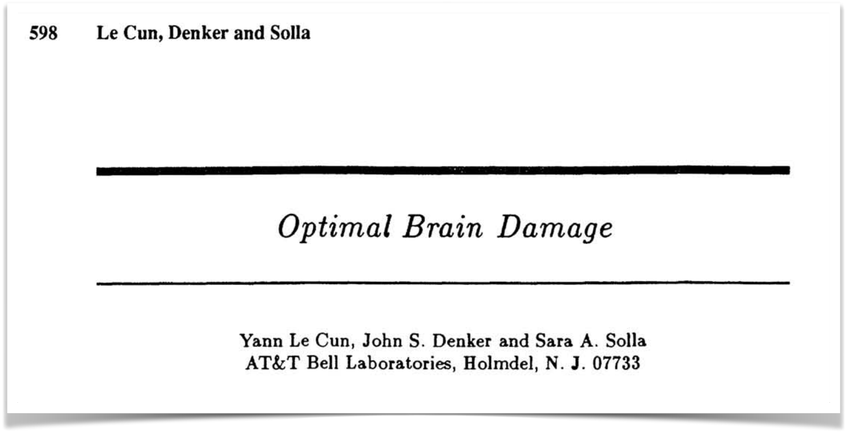


Optimal Brain Damage [LeCun *et al.,* NeurIPS 1989]
Learning Both Weights and Connections for Efficient Neural Network [Han *et al.*, NeurIPS 2015]

Neural Network Pruning

Make neural network smaller by removing synapses and neurons

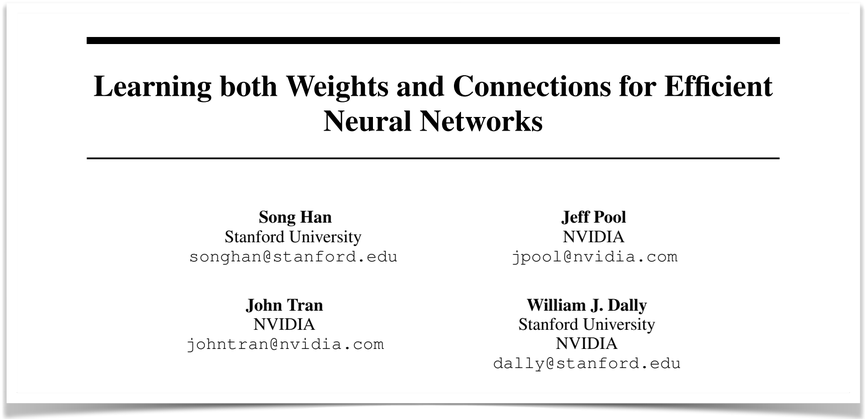




598 Le Cun, Denker and Solla

Optimal Brain Damage

Yann Le Cun, John S. Denker and Sara A. Solla
AT&T Bell Laboratories, Holmdel, N. J. 07733



Learning both Weights and Connections for Efficient Neural Networks

Song Han
Stanford University
songhan@stanford.edu

Jeff Pool
NVIDIA
jpool@nvidia.com

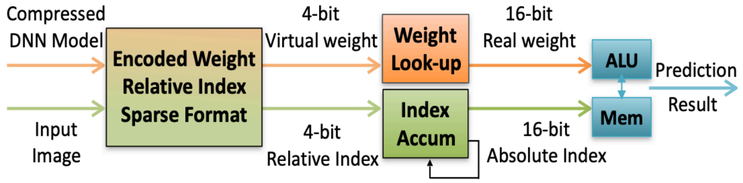
John Tran
NVIDIA
johntran@nvidia.com

William J. Dally
Stanford University
NVIDIA
dally@stanford.edu

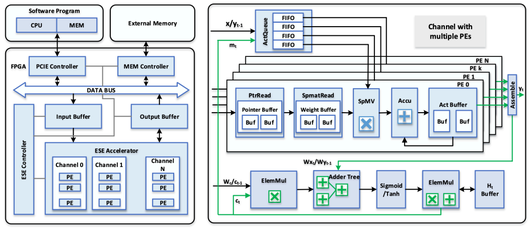
Souce: Dimension.ai

Pruning in the Industry

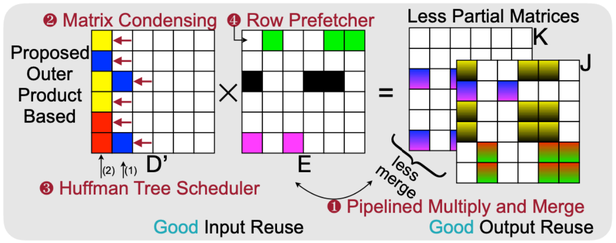
Hardware support for sparsity



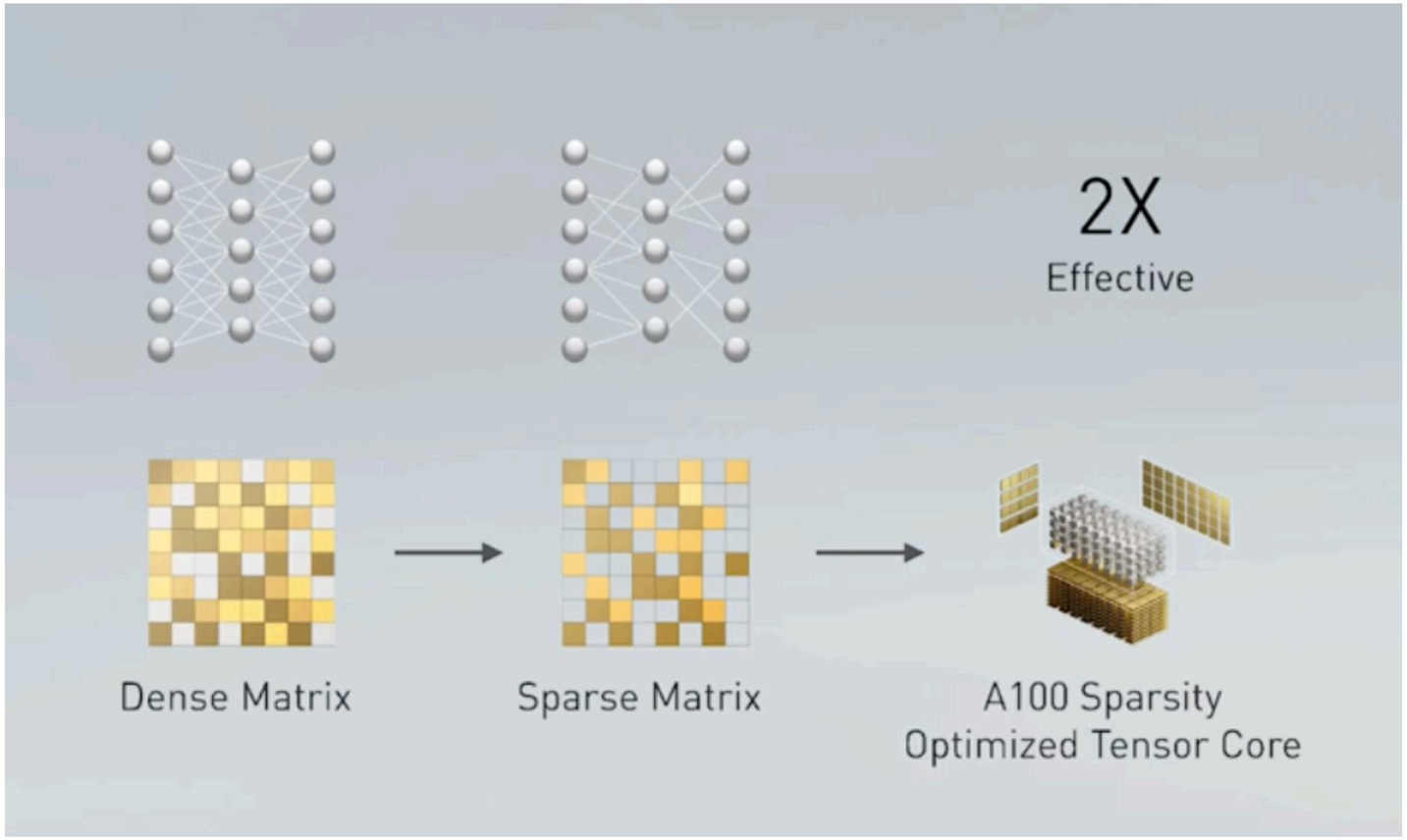
EIE [Han *et al.*, ISCA 2016]



ESE [Han *et al.*, FPGA 2017]



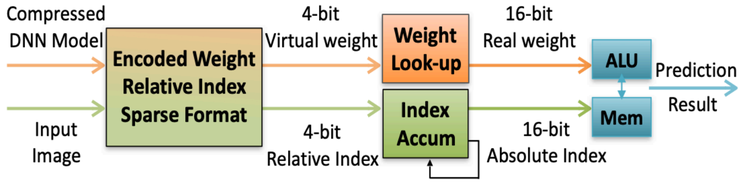
SpArch [Zhang *et al.*, HPCA 2020]
SpAtten [Wang *et al.*, HPCA 2021]



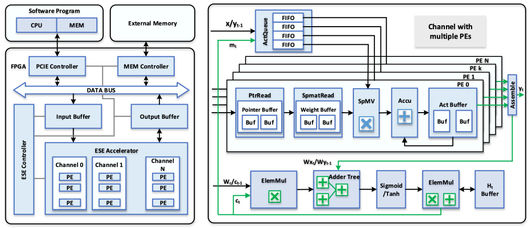
2:4 sparsity in A100 GPU
2X peak performance, 1.5X measured BERT speedup

Pruning in the Industry

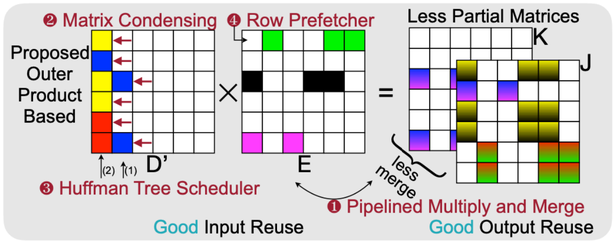
Hardware support for sparsitv



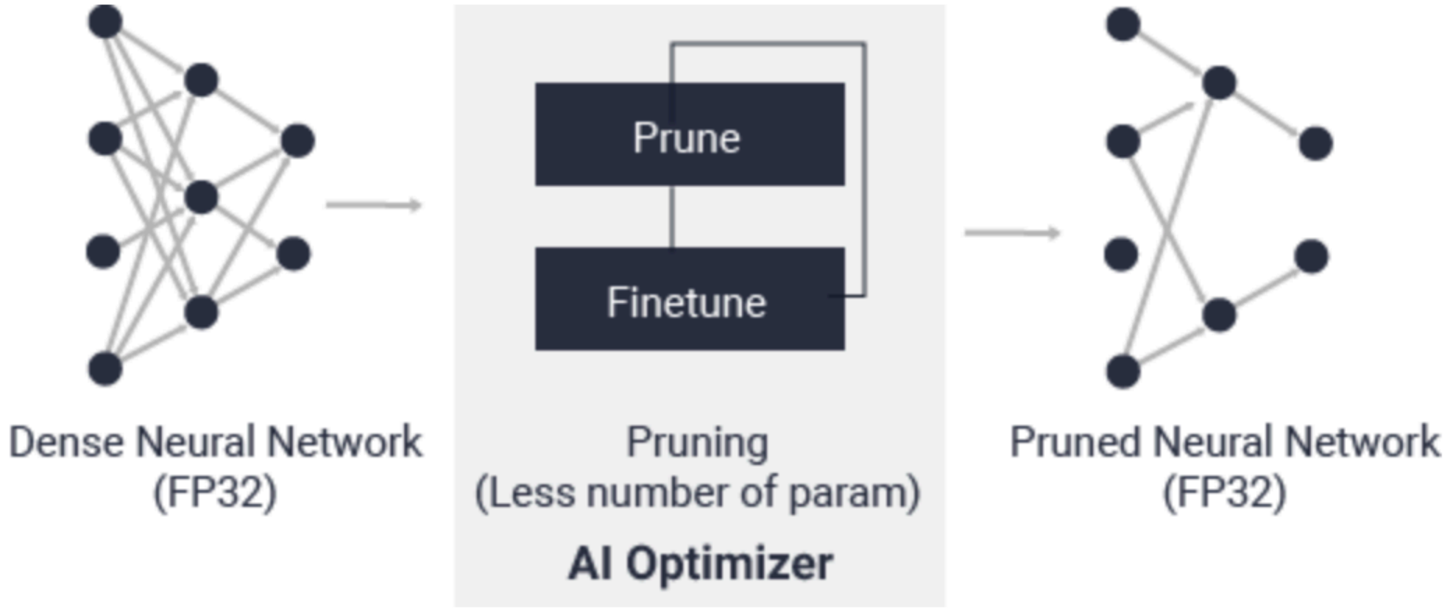
EIE [Han *et al.*, ISCA 2016]



ESE [Han *et al.*, FPGA 2017]



SpArch [Zhang *et al.*, HPCA 2020]
SpAtten [Wang *et al.*, HPCA 2021]



Reduce model complexity by 5x to 50x with minimal accuracy impact

Neural Network Pruning

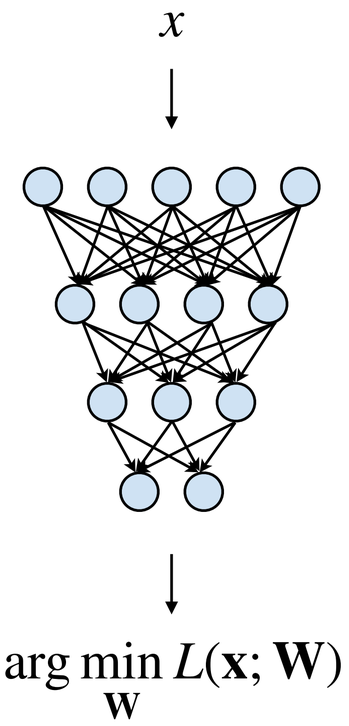
- In general, we could formulate the pruning as follows:

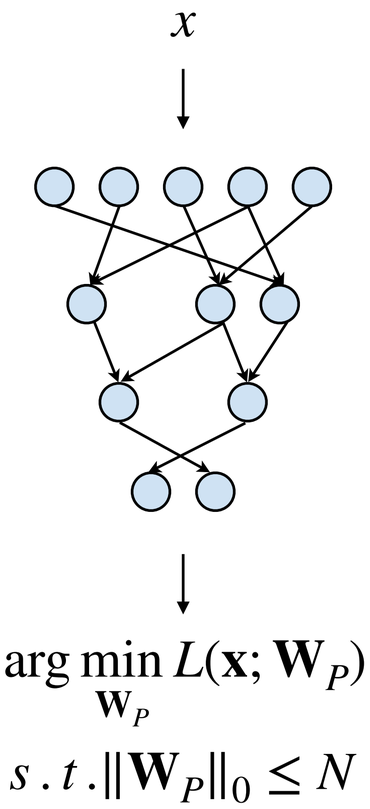
$$\arg\min_{\mathbf{W}_P} L(\mathbf{x}; \mathbf{W}_P)$$

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.

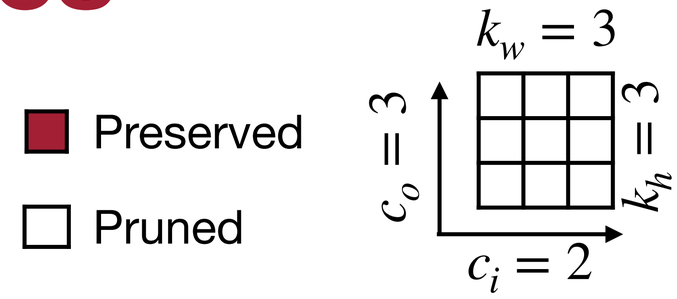


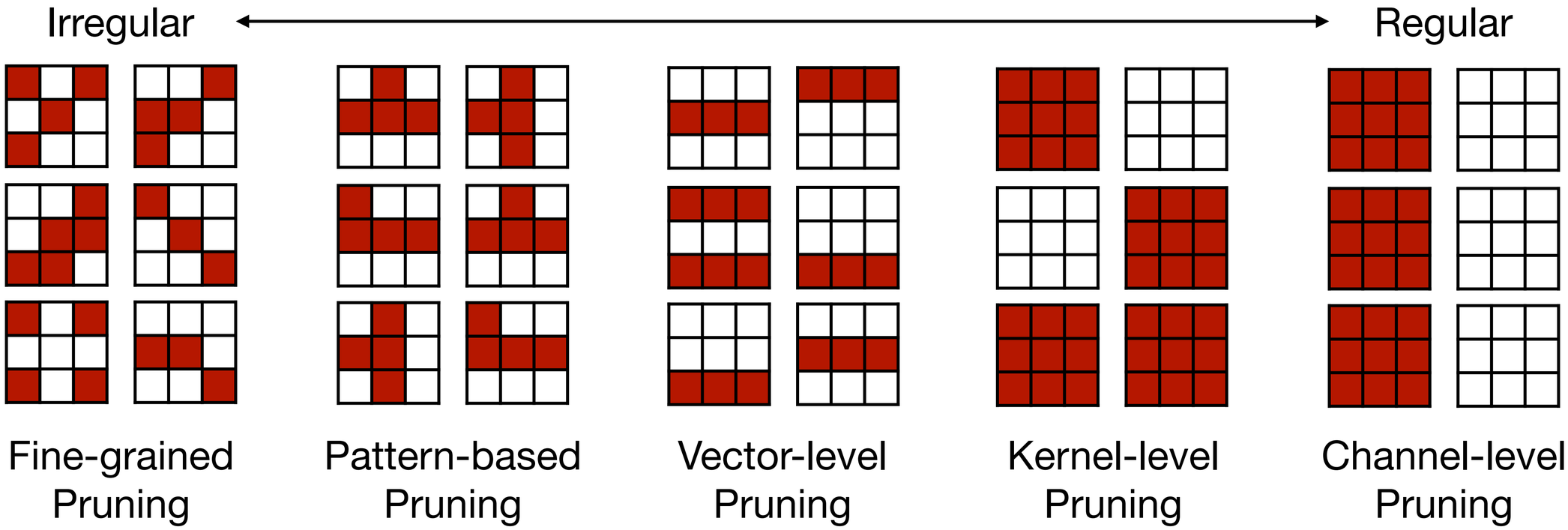


Pruning at Different Granularities

The case of convolutional layers

- Some of the commonly used pruning granularities





like Tetris :)

Exploring the granularity of sparsity in convolutional neural networks [Mao *et al.*, CVPR-W]

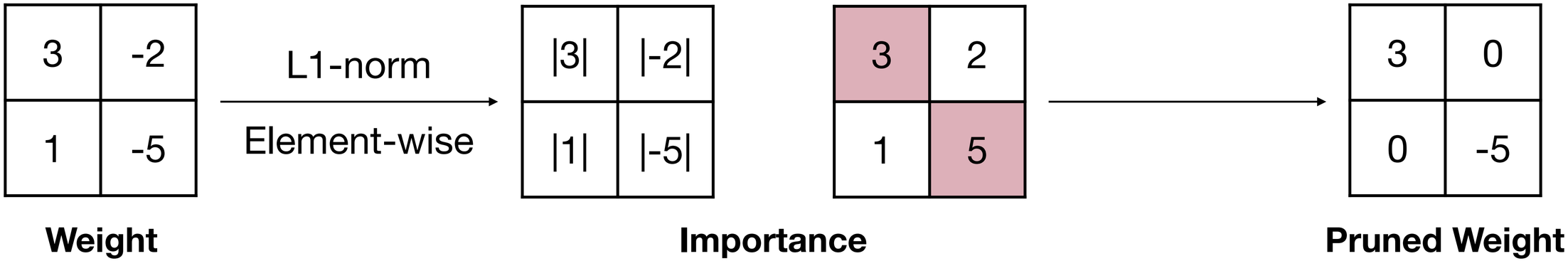
Selection of Synapses to Prune

- When removing parameters from a neural network model,
  - ***the less important*** the parameters being removed are,
  - the better the performance of pruned neural network is.

- **Magnitude-based pruning** considers weights with ***larger absolute values*** are more important than other weights.
  - For element-wise pruning,

$$Importance = |W|$$

  - **Example**

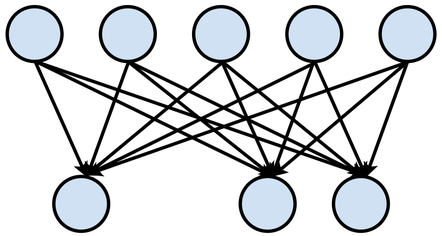


Selection of Neurons to Prune

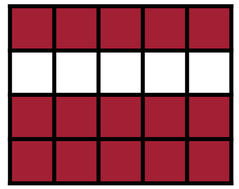
- When removing neurons from a neural network model,
  - ***the less useful*** the neurons being removed are,
  - the better the performance of pruned neural network is.

Recall: Neuron pruning is coarse-grained pruning indeed.

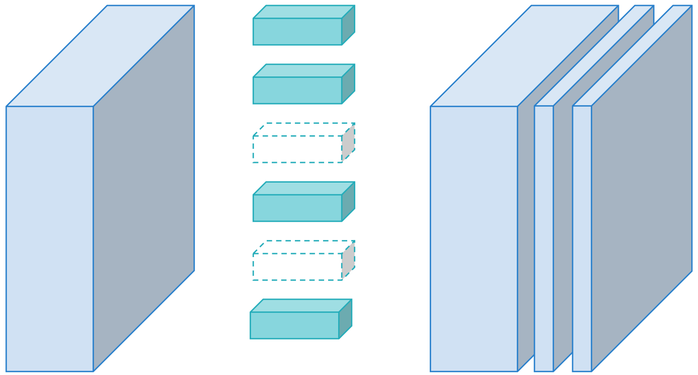
Neuron Pruning in Linear Layer

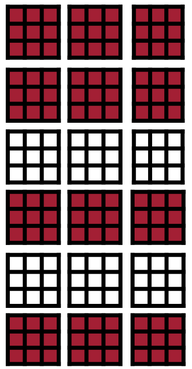


Weight Matrix



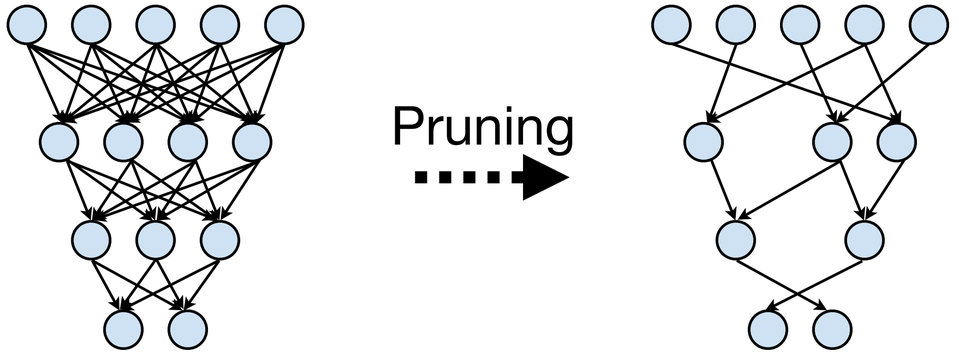
Channel Pruning in Convolution Layer





Neural Network Pruning

- **Introduction to Pruning**
  - What is pruning?
  - How should we formulate pruning?
- **Determine the Pruning Granularity**
  - In what pattern should we prune the neural network?
- **Determine the Pruning Criterion**
  - What synapses/neurons should we prune?
- **Determine the Pruning Ratio**
  - What should target sparsity be for each layer?
- **Fine-tune/Train Pruned Neural Network**
  - How should we improve performance of pruned models?



prune 30%?

prune 50%?

prune 70%?

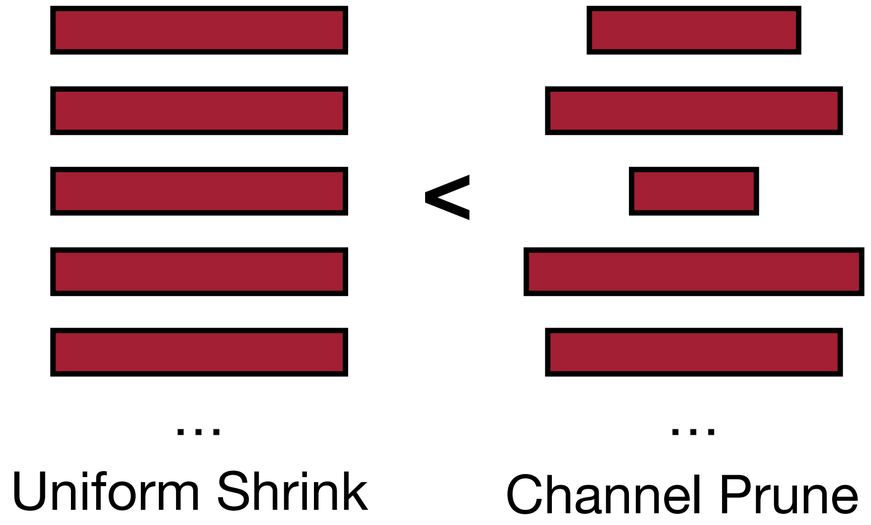
Learning Both Weights and Connections for Efficient Neural Network [Han *et al.*, NeurIPS 2015]

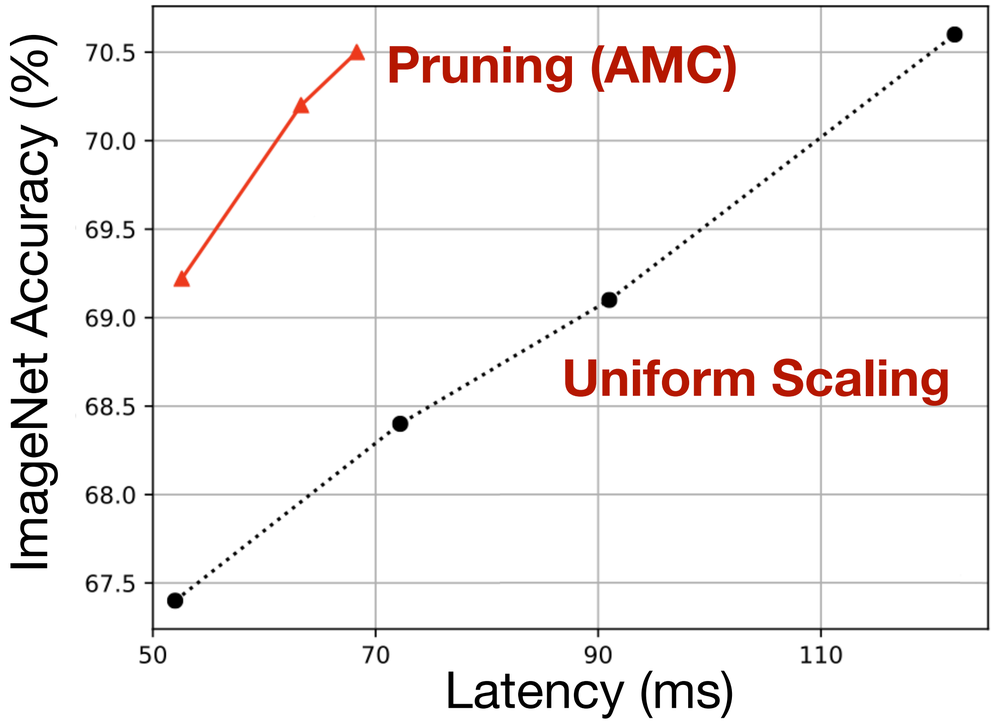
Section 1: Pruning Ratio

How should we find per-layer pruning ratios?

Recap

Non-uniform pruning is better than uniform shrinking

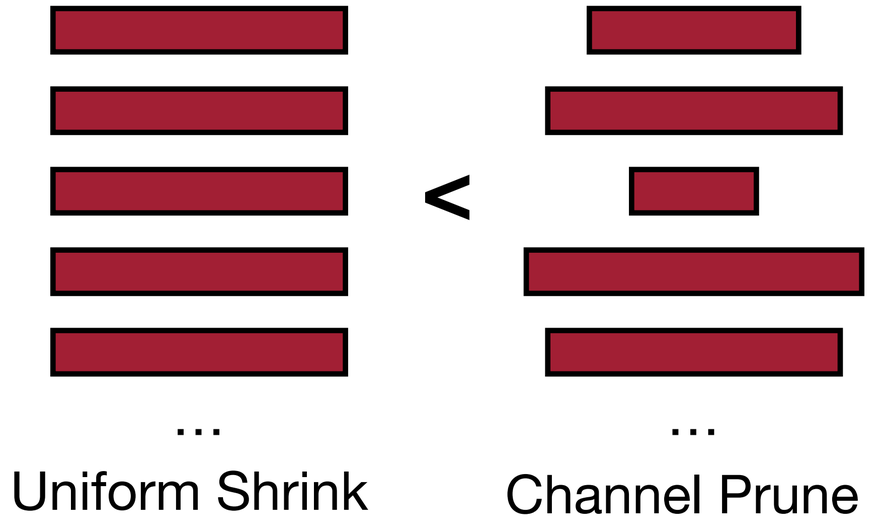


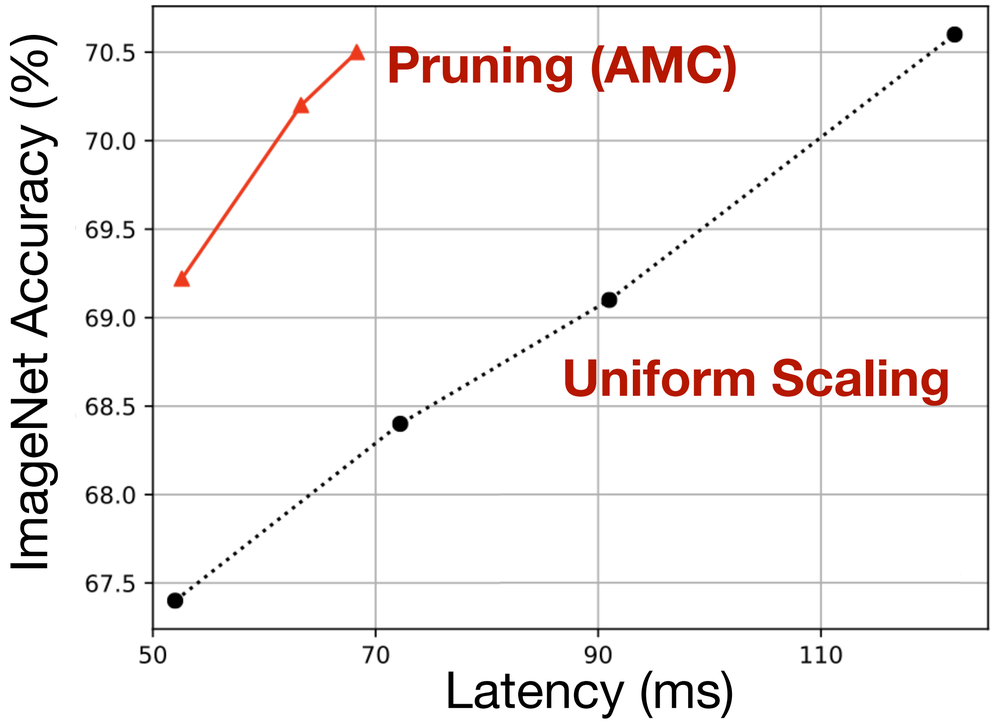


AMC: Automl for Model Compression and Acceleration on Mobile Devices [He *et al.*, ECCV 2018]

Recap

Non-uniform pruning is better than uniform shrinking





Question: how should we find ratios for each layer?

AMC: Automl for Model Compression and Acceleration on Mobile Devices [He *et al.*, ECCV 2018]

Finding Pruning Ratios

Analyze the sensitivity of each layer

- We need different pruning ratios for each layer since different layers have different **sensitivity**
  - Some layers are more sensitive (e.g., first layer)
  - Some layers are more redundant

Finding Pruning Ratios

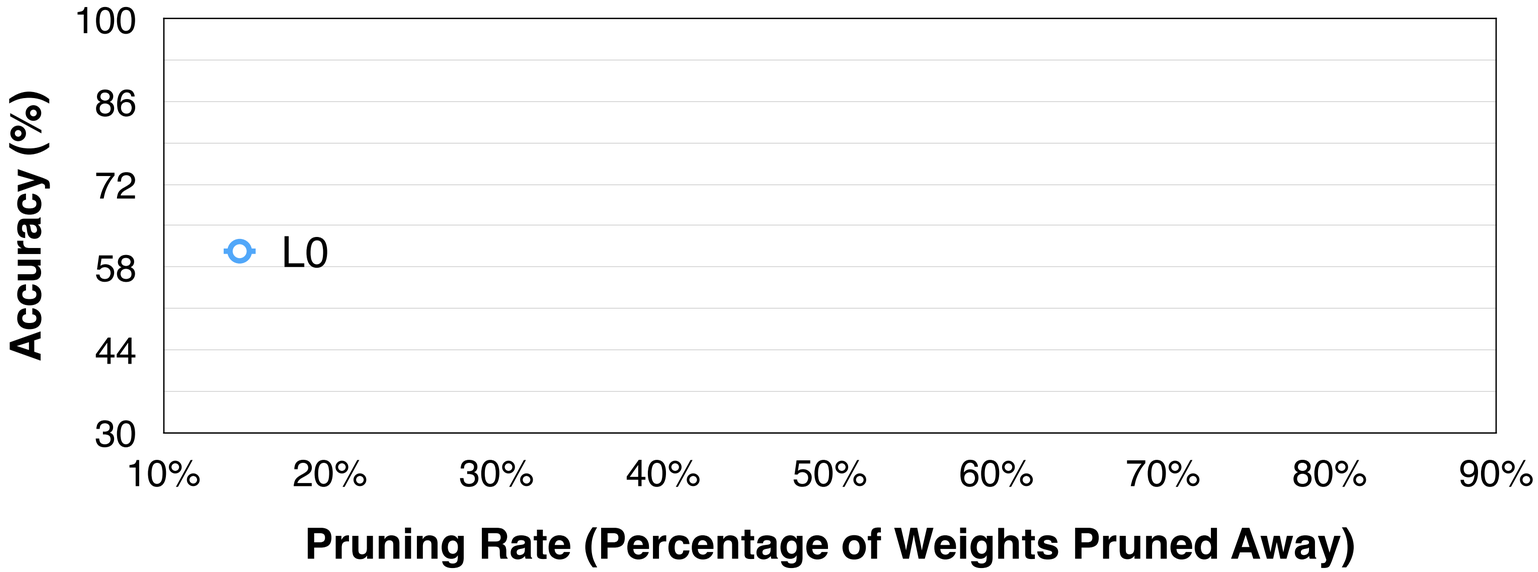
Analyze the sensitivity of each layer

- We need different pruning ratios for each layer since different layers have different **sensitivity**
  - Some layers are more sensitive (e.g., first layer)
  - Some layers are more redundant
- We can perform **sensitivity analysis** to determine the per-layer pruning ratio

Finding Pruning Ratios

Analyze the sensitivity of each layer

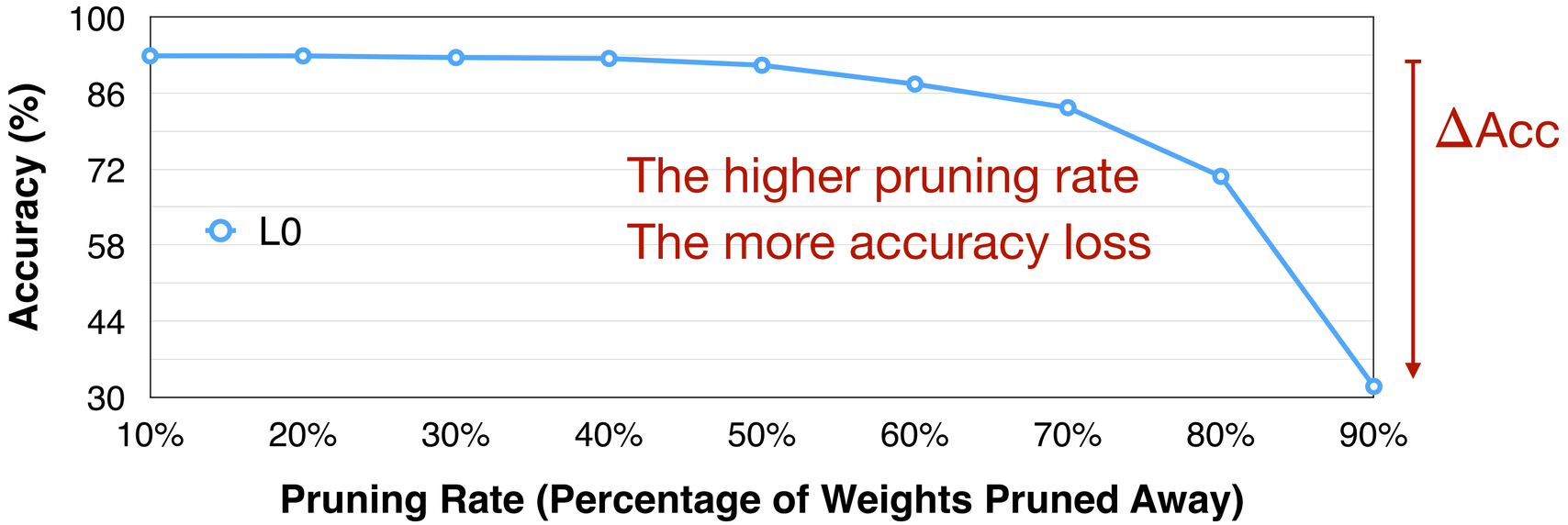
- The process of Sensitivity Analysis (* VGG-11 on CIFAR-10 dataset)
  - Pick a layer L_i in the model



Finding Pruning Ratios

Analyze the sensitivity of each layer

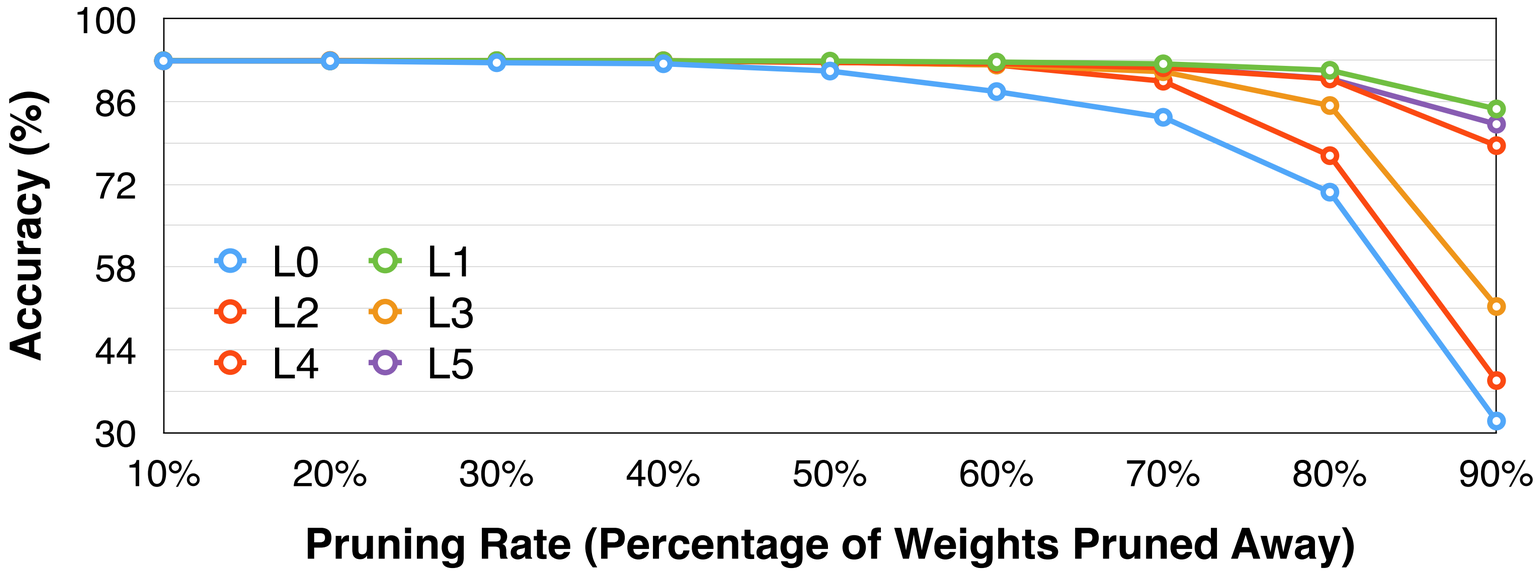
- The process of Sensitivity Analysis (* VGG-11 on CIFAR-10 dataset)
  - Pick a layer L_i in the model
    - Prune the layer L_i with pruning ratio $r \in \{0, 0.1, 0.2, ..., 0.9\}$ (or other strides)
    - Observe the accuracy degrade ΔAcc_r^i for each pruning ratio



Finding Pruning Ratios

Analyze the sensitivity of each layer

- The process of Sensitivity Analysis (* VGG-11 on CIFAR-10 dataset)
  - Pick a layer L_i in the model
    - Prune the layer L_i with pruning ratio $r \in \{0, 0.1, 0.2, ..., 0.9\}$ (or other strides)
    - Observe the accuracy degrade ΔAcc_r^i for each pruning ratio
  - Repeat the process for all layers

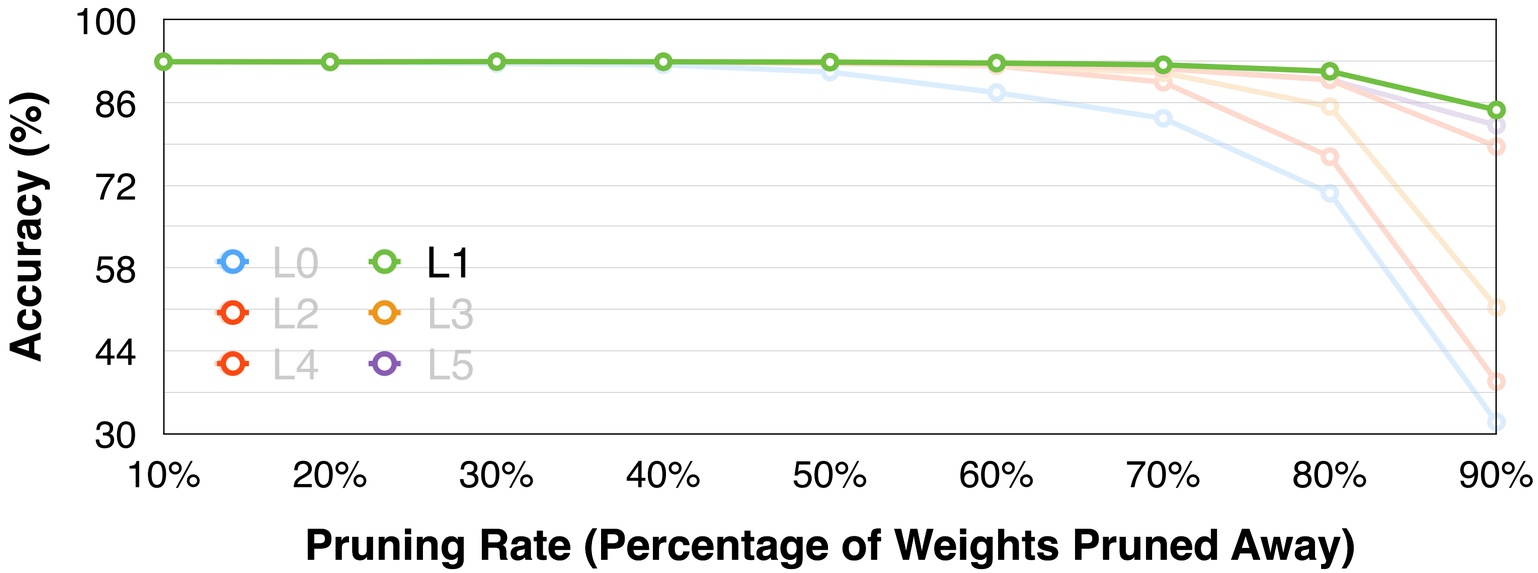


Finding Pruning Ratios

Analyze the sensitivity of each layer

- The process of Sensitivity Analysis (* VGG-11 on CIFAR-10 dataset)
  - Pick a layer L_i in the model
    - Prune the layer L_i with pruning ratio $r \in \{0, 0.1, 0.2, ..., 0.9\}$ (or other strides)
    - Observe the accuracy degrade ΔAcc_r^i for each pruning ratio
  - Repeat the process for all layers

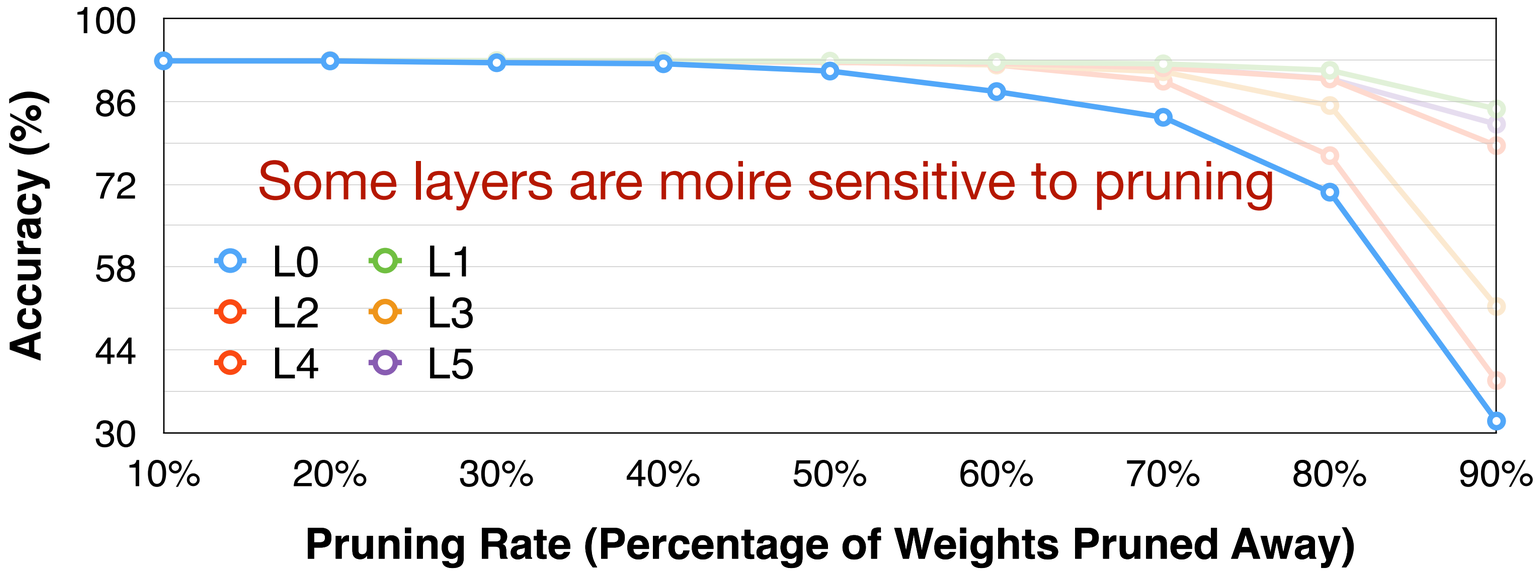
Some layers are less sensitive to pruning



Finding Pruning Ratios

Analyze the sensitivity of each layer

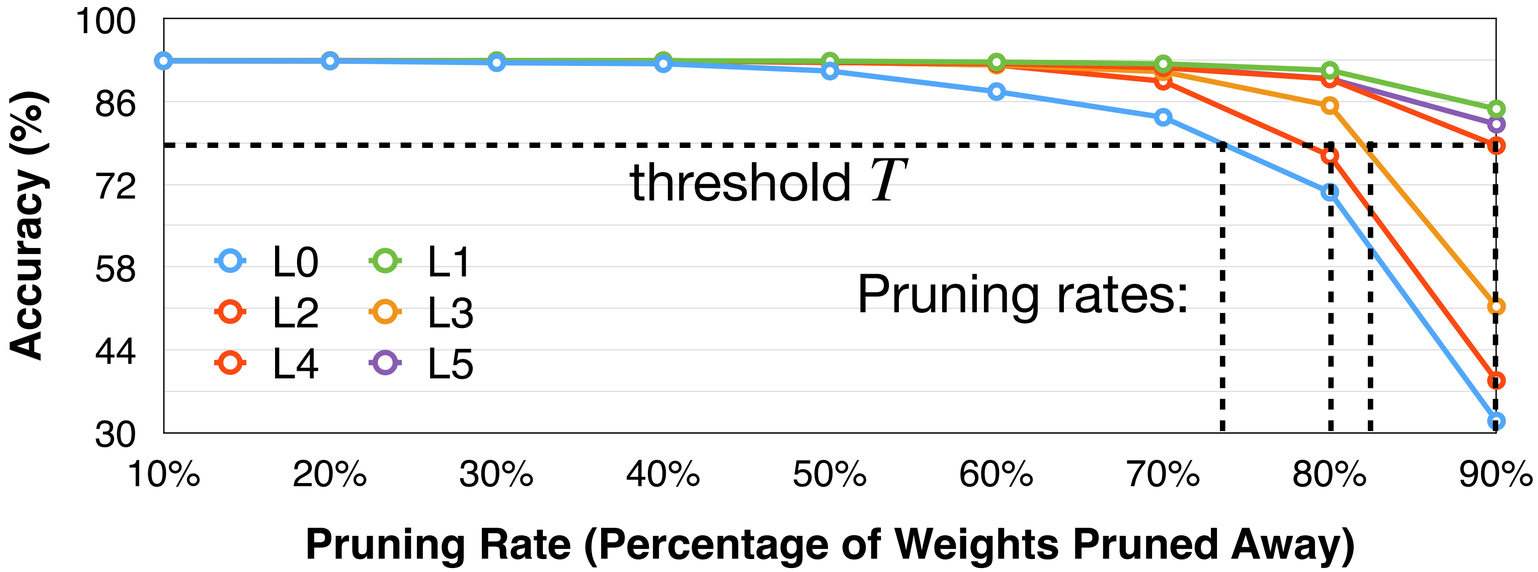
- The process of Sensitivity Analysis (* VGG-11 on CIFAR-10 dataset)
  - Pick a layer L_i in the model
    - Prune the layer L_i with pruning ratio $r \in \{0, 0.1, 0.2, ..., 0.9\}$ (or other strides)
    - Observe the accuracy degrade ΔAcc_r^i for each pruning ratio
  - Repeat the process for all layers



Finding Pruning Ratios

Analyze the sensitivity of each layer

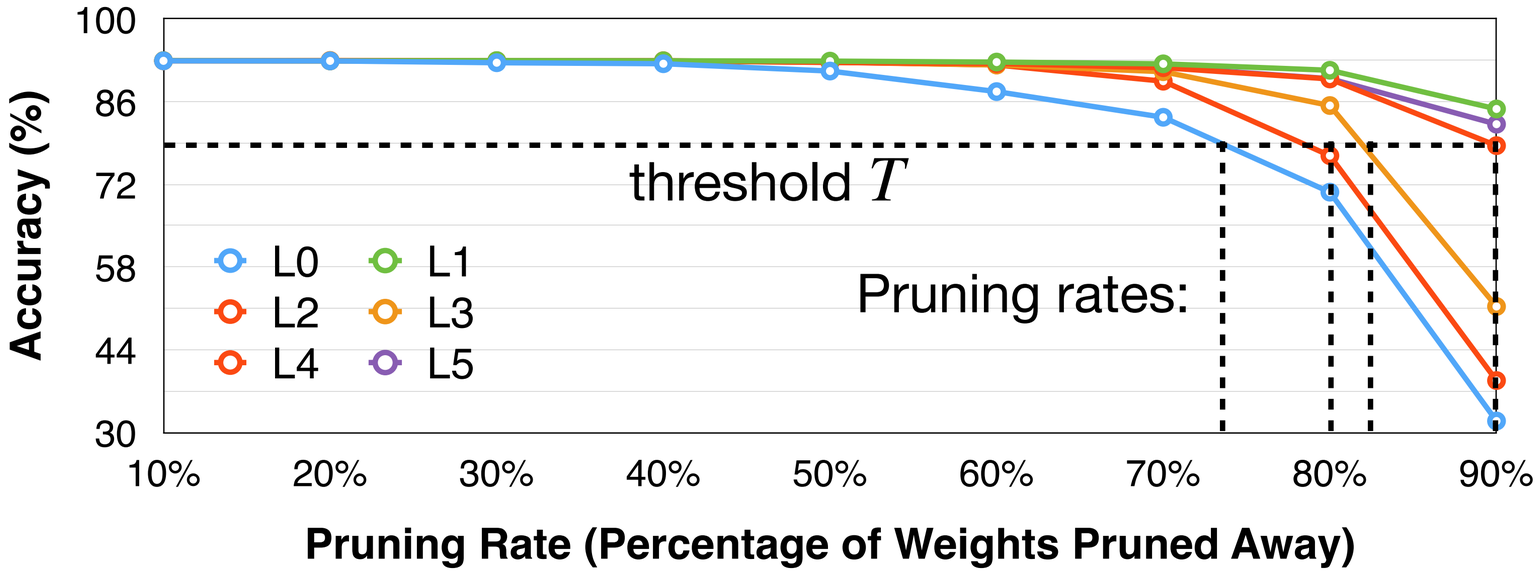
- The process of Sensitivity Analysis (* VGG-11 on CIFAR-10 dataset)
  - Pick a layer L_i in the model
    - Prune the layer L_i with pruning ratio $r \in \{0, 0.1, 0.2, ..., 0.9\}$ (or other strides)
    - Observe the accuracy degrade ΔAcc_r^i for each pruning ratio
  - Repeat the process for all layers
  - Pick a degradation threshold T such that the overall pruning rate is desired



Finding Pruning Ratios

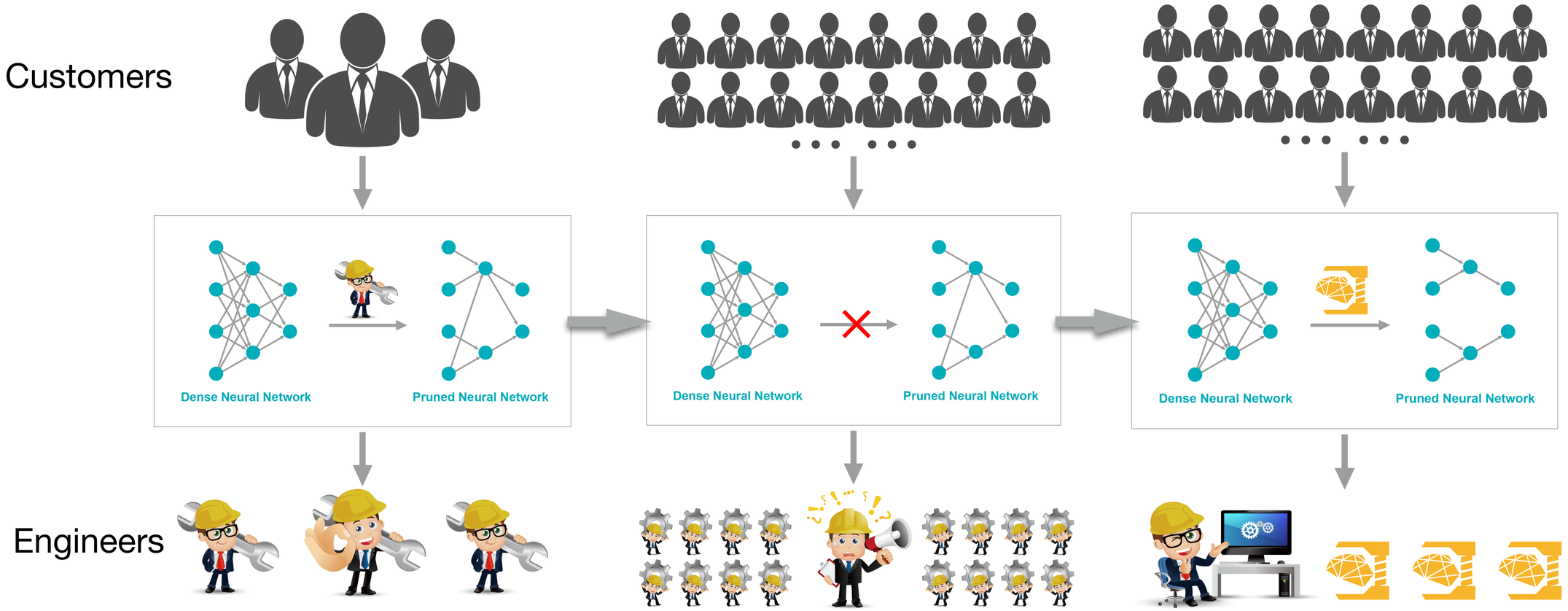
Analyze the sensitivity of each layer

- Is this optimal?
  - Maybe not. We do not consider the interaction between layers.
- Can we go beyond the heuristics?
  - Yes!



Automatic Pruning

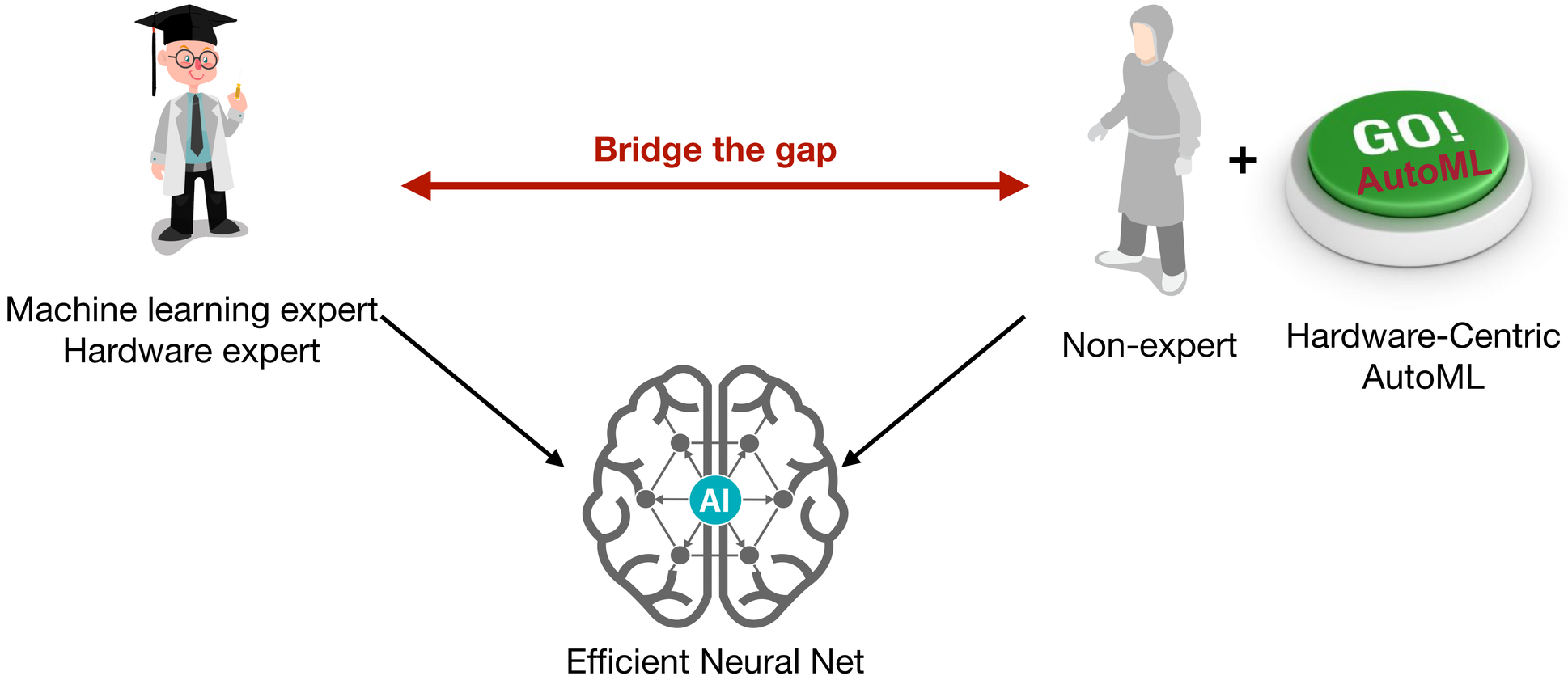
- Given an **overall** compression ratio, how do we choose **per-layer** pruning ratios?
  - Sensitivity analysis ignores the interaction between layers -> sub-optimal
- Conventionally, such process relies on human expertise and trials and errors



AMC: AutoML for Model Compression and Acceleration on Mobile Devices [He *et al.*, ECCV 2018]

Automatic Pruning

- Can we develop a push-the-button solution?

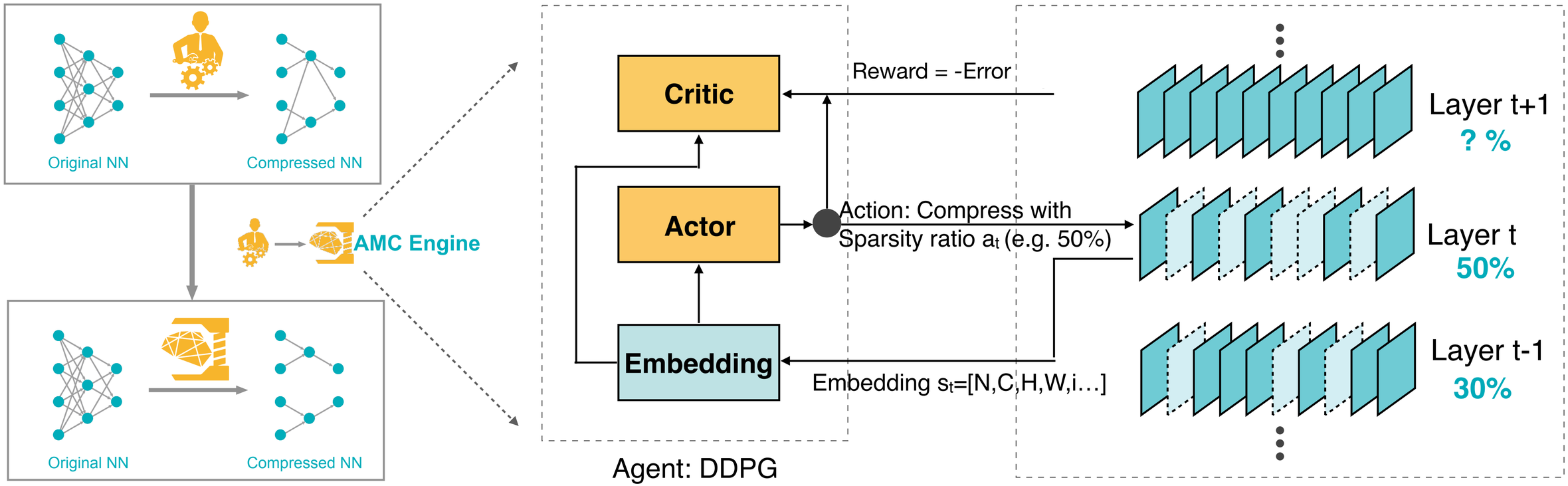


AMC: AutoML for Model Compression and Acceleration on Mobile Devices [He *et al.*, ECCV 2018]

AMC: AutoML for Model Compression

Pruning as a reinforcement learning problem

Model Compression by Human:
Labor Consuming, Sub-optimal



Model Compression by AI:
Automated, Higher Compression Rate, Faster

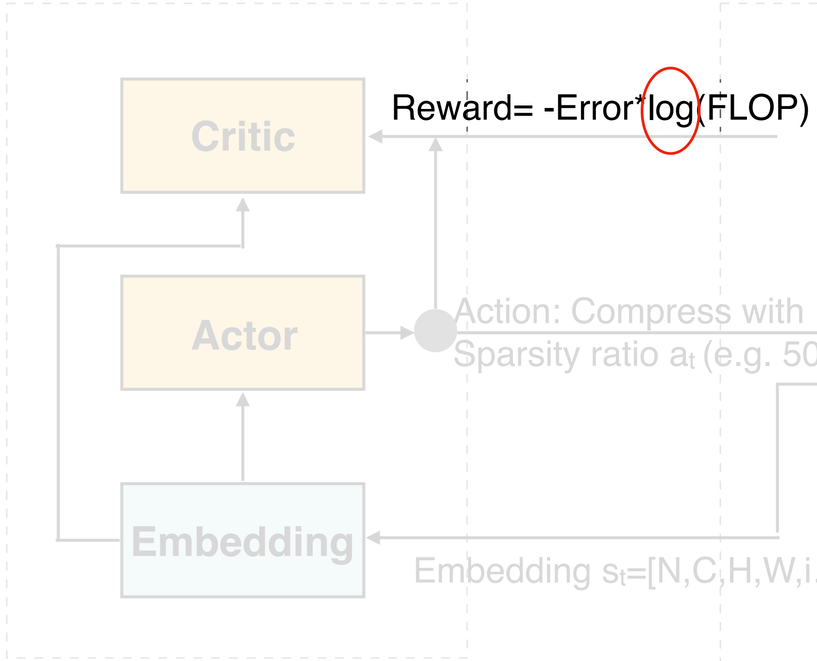
AMC: AutoML for Model Compression and Acceleration on Mobile Devices [He *et al.*, ECCV 2018]

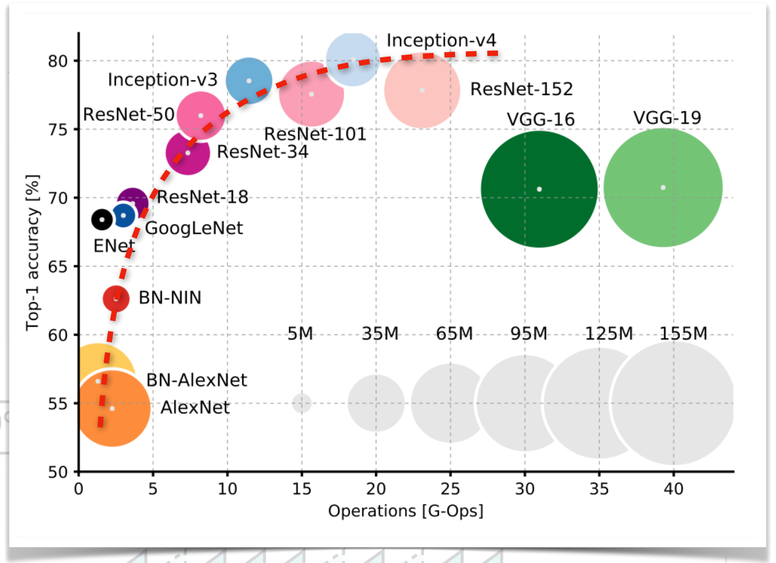
AMC: AutoML for Model Compression

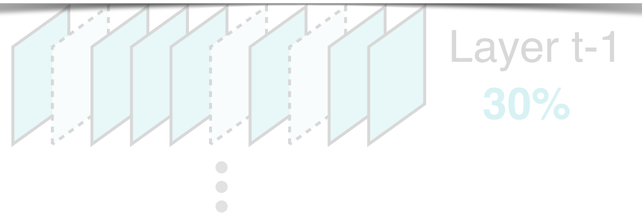
Pruning as a reinforcement learning problem

Model Compression by Human:
Labor Consuming, Sub-optimal

Model Compression by AI:
Automated, Higher Compression Rate, Faster







Environment: Channel Pruning

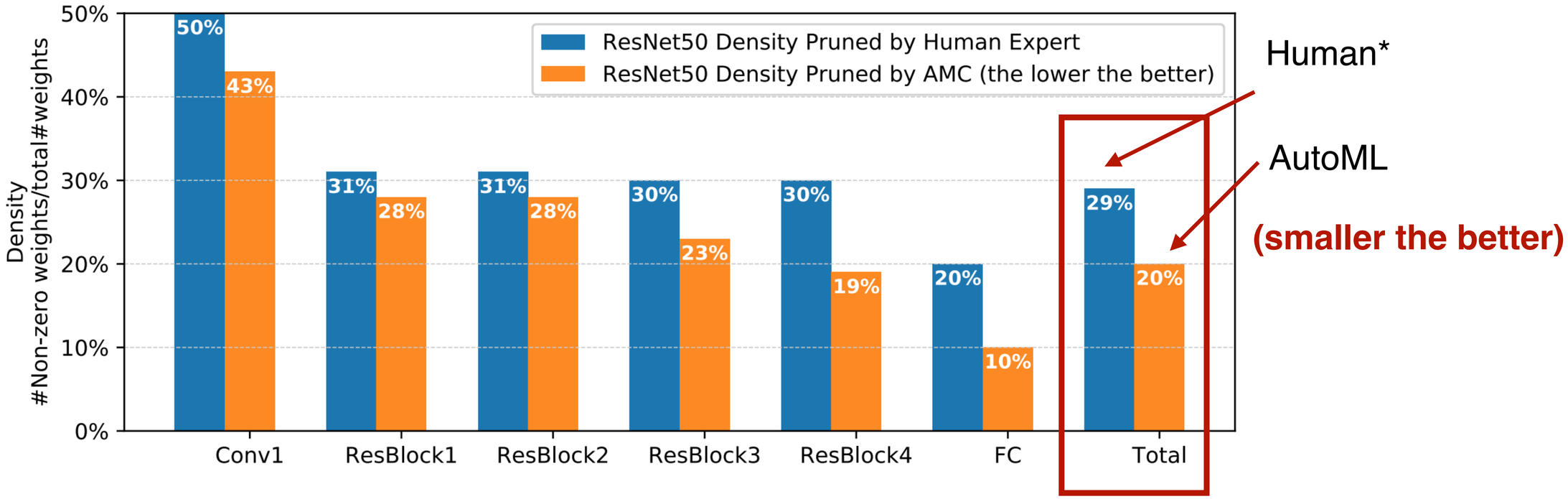
An Analysis of Deep Neural Network Models for Practical Applications [Canziani *et al.*, 2016]

AMC: AutoML for Model Compression

- AMC uses the following setups for the reinforcement learning problem
  - **State**: 11 features (including layer indices, channel numbers, kernel sizes, FLOPs, …)
  - **Action**: A continuous number (pruning ratio) $a \in [0,1)$
  - **Agent**: DDPG agent, since it supports continuous action output
  - **Reward**: $R = \begin{cases} -\text{Error}, & \text{if satisfies constrains} \\ -\infty, & \text{if not} \end{cases}$
    - We can also optimize latency constraints with a pre-built lookup table (LUT)

AMC: AutoML for Model Compression and Acceleration on Mobile Devices [He *et al.*, ECCV 2018]

AMC: AutoML for Model Compression



* Efficient Methods and Hardware for Deep Learning [Han, *thesis*]

AMC: AutoML for Model Compression

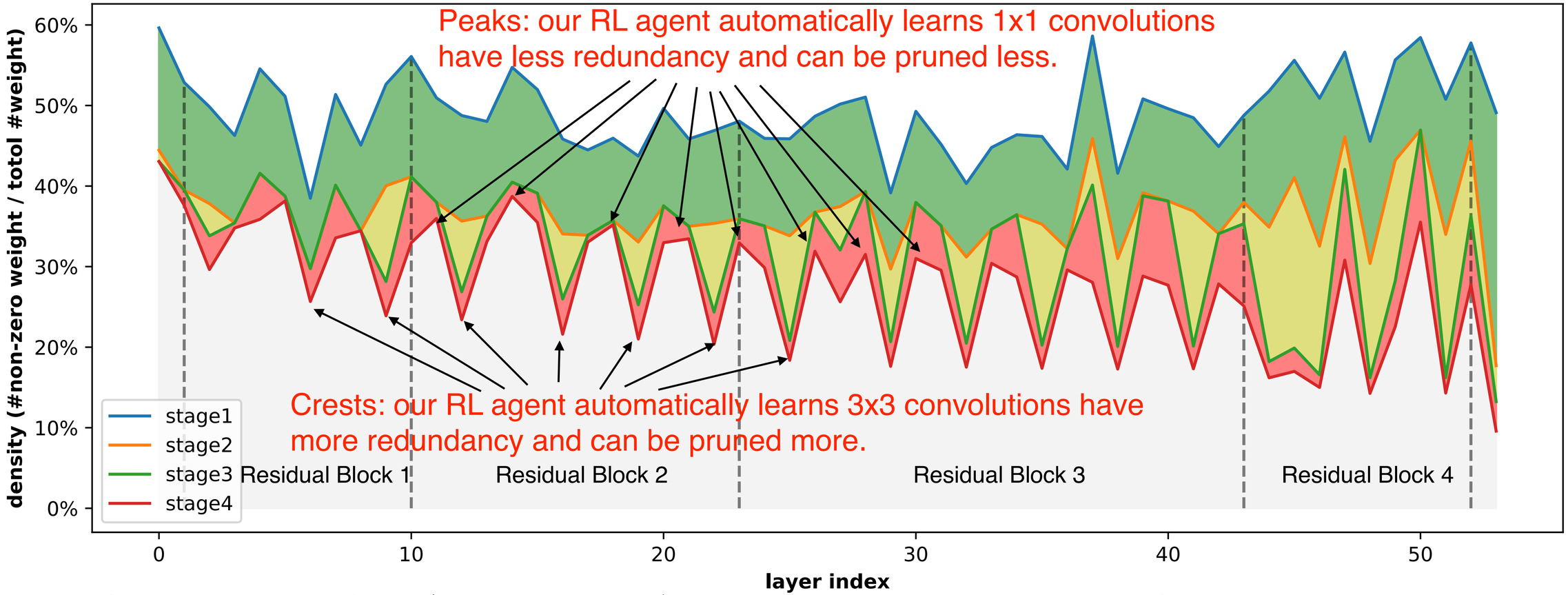


Figure 14: The pruning policy (sparsity ratio) given by our reinforcement learning agent for ResNet-50.

AMC: AutoML for Model Compression

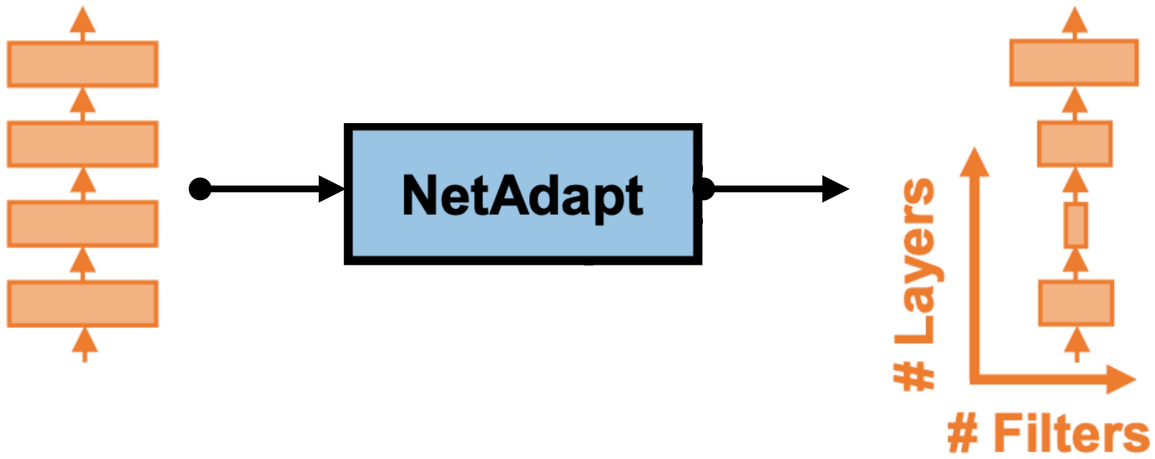
Model	MAC	Top-1	Latency*	Speedup	Memory
1.0 MobileNet	569M	70.6%	119.0ms	1x	20.1MB
AMC (50% **FLOPs**)	285M	**70.5%**	64.4ms	**1.8x**	14.3MB
AMC (50% **Time**)	272M	**70.2%**	59.7ms	**2.0x**	13.2MB
0.75 MobileNet	325M	68.4%	69.5ms	1.7x	14.8MB

* Measured with TF-Lite on Samsung Galaxy S7 Edge, which has Qualcomm Snapdragon SoC
Single core, Batch size = 1(mobile, latency oriented)

NetAdapt

A rule-based iterative/progressive method

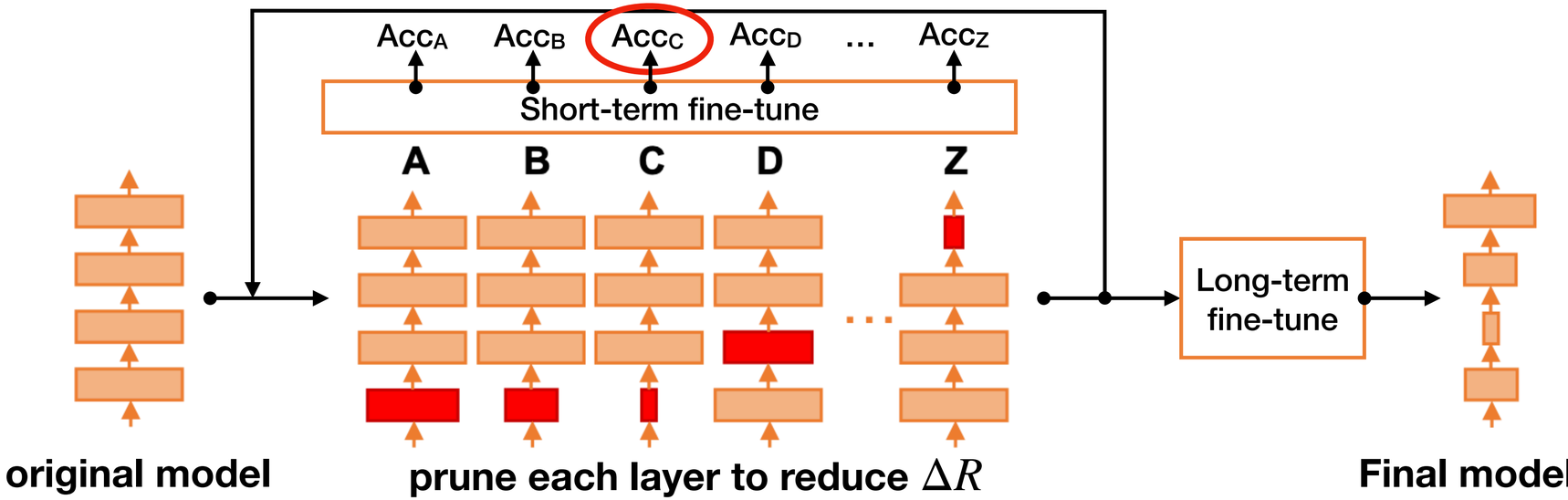
- The goal of NetAdapt is to find a per-layer pruning ratio to meet a global resource constraint (e.g., latency, energy, …)
- The process is done iteratively
- We take **latency** constraint as an example



NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications [Yang *et al.*, ECCV 2018]

NetAdapt

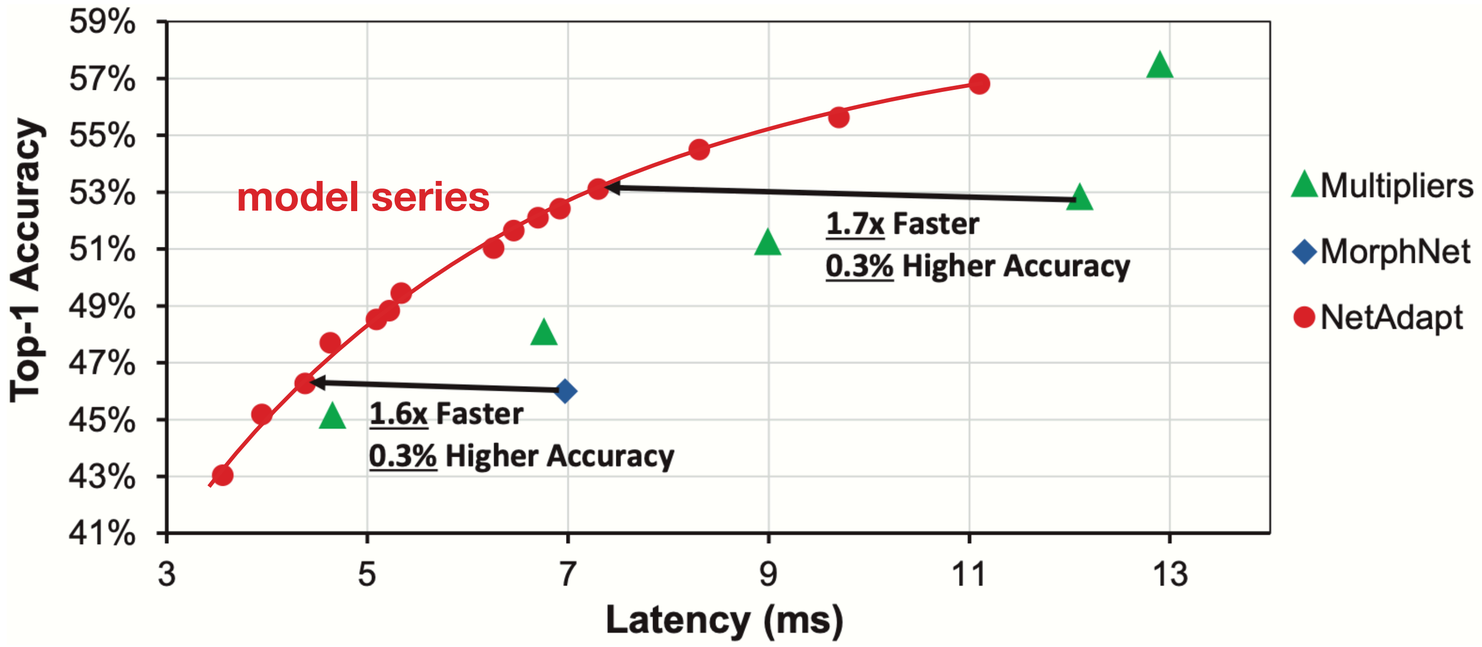
- For each iteration, we aim to reduce the latency by a certain amount ΔR (manually defined)
  - For each layer L_k (k in A-Z in the figure)
    - Prune the layer s.t. the latency reduction meets ΔR (based on a pre-built lookup table)
    - Short-term fine-tune model (10k iterations); measure accuracy after fine-tuning
  - Choose and prune the layer with the highest accuracy
- Repeat until the total latency reduction satisfies the constraint
- Long-term fine-tune to recover accuracy



NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications [Yang *et al.*, ECCV 2018]

NetAdapt

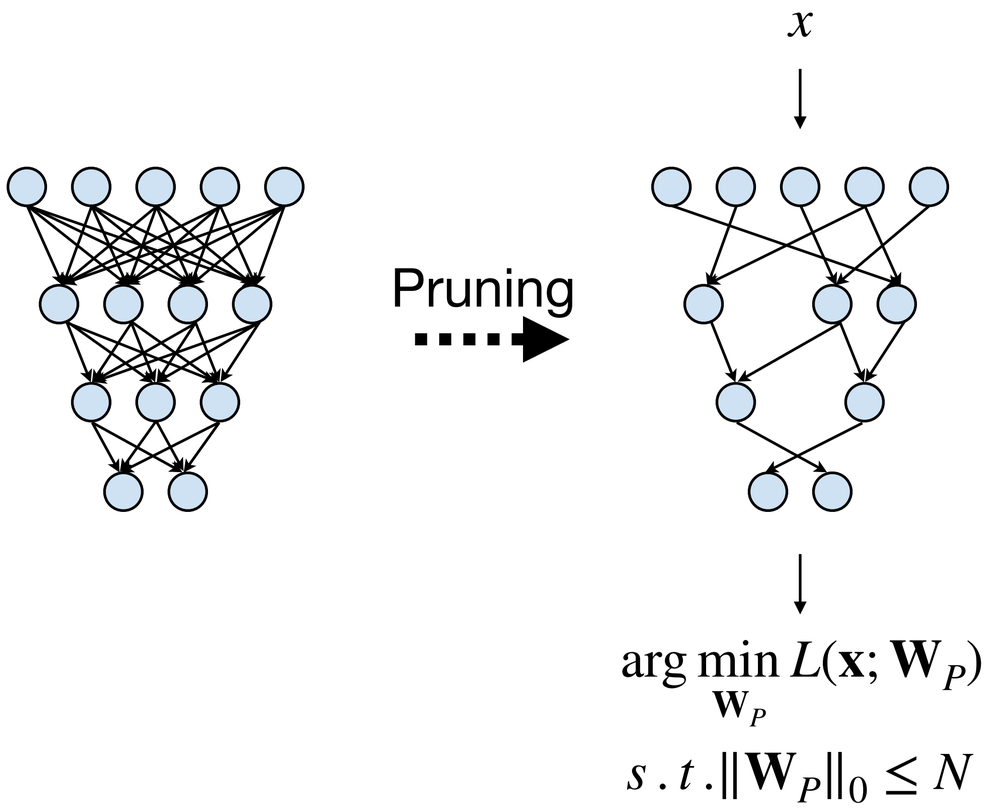
- The iterative nature allows us to obtain **a serial of** models with different costs
  - #models = #iterations



NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications [Yang *et al.*, ECCV 2018]

Neural Network Pruning

- **Introduction to Pruning**
  - What is pruning?
  - How should we formulate pruning?
- **Determine the Pruning Granularity**
  - In what pattern should we prune the neural network?
- **Determine the Pruning Criterion**
  - What synapses/neurons should we prune?
- **Determine the Pruning Ratio**
  - What should target sparsity be for each layer?
- **Fine-tune/Train Pruned Neural Network**
  - How should we improve performance of pruned models?



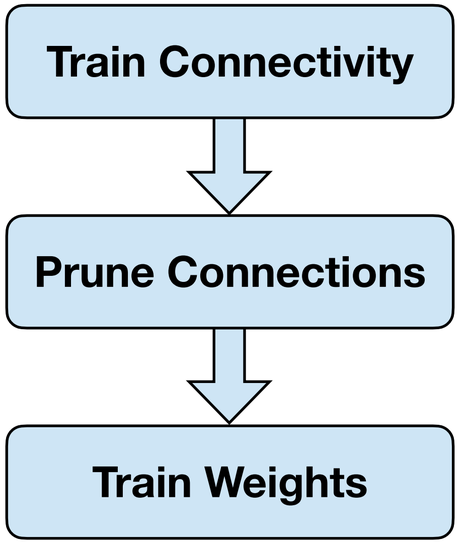
Learning Both Weights and Connections for Efficient Neural Network [Han *et al.*, NeurIPS 2015]

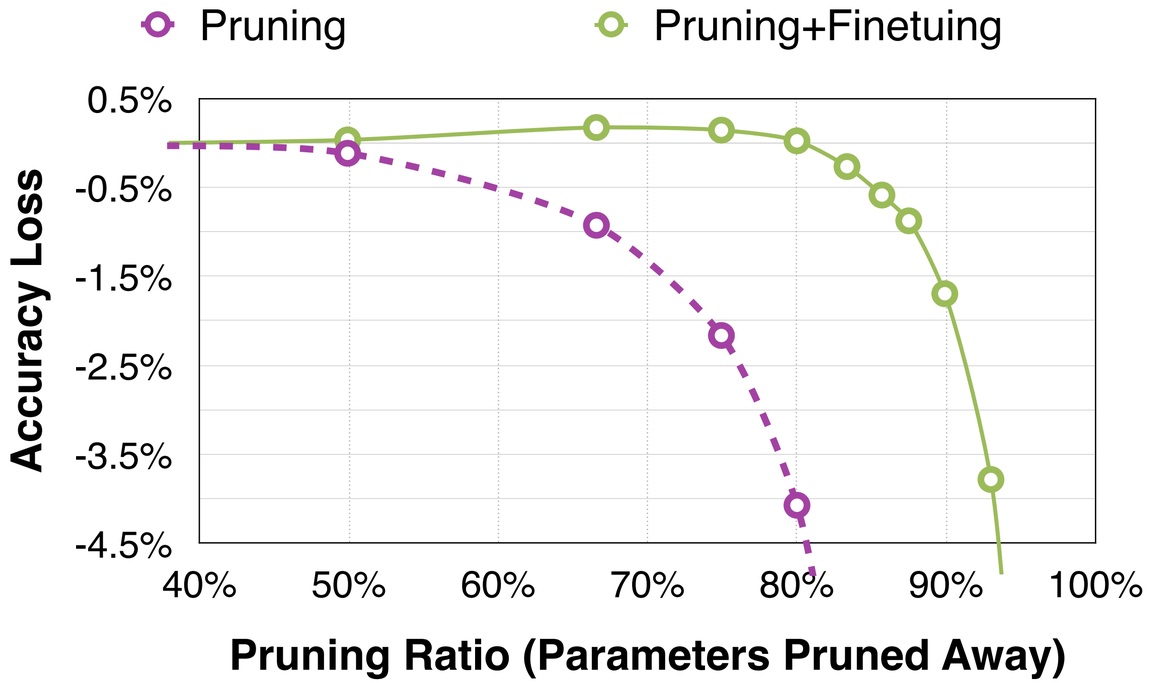
Section 2: Fine-tuning / Training

How should we improve performance of sparse models?

Finetuning Pruned Neural Networks

- After pruning, the model may decrease, especially for larger pruning ratio.
- Fine-tuning the pruned neural networks will help recover the accuracy and push the pruning ratio higher.
  - **Learning rate for fine-tuning is usually 1/100 or 1/10 of the original learning rate.**

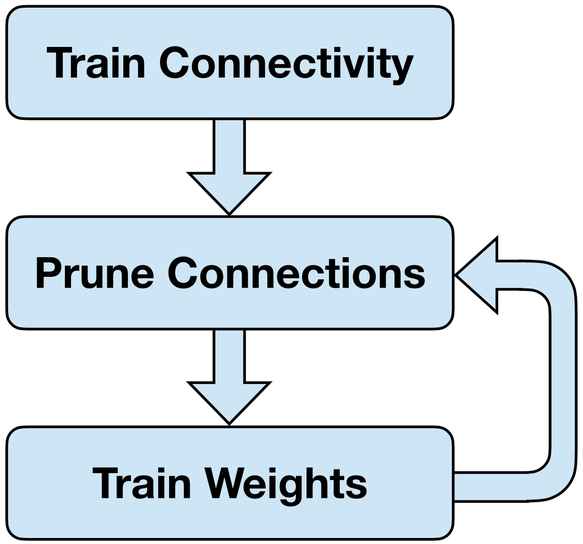


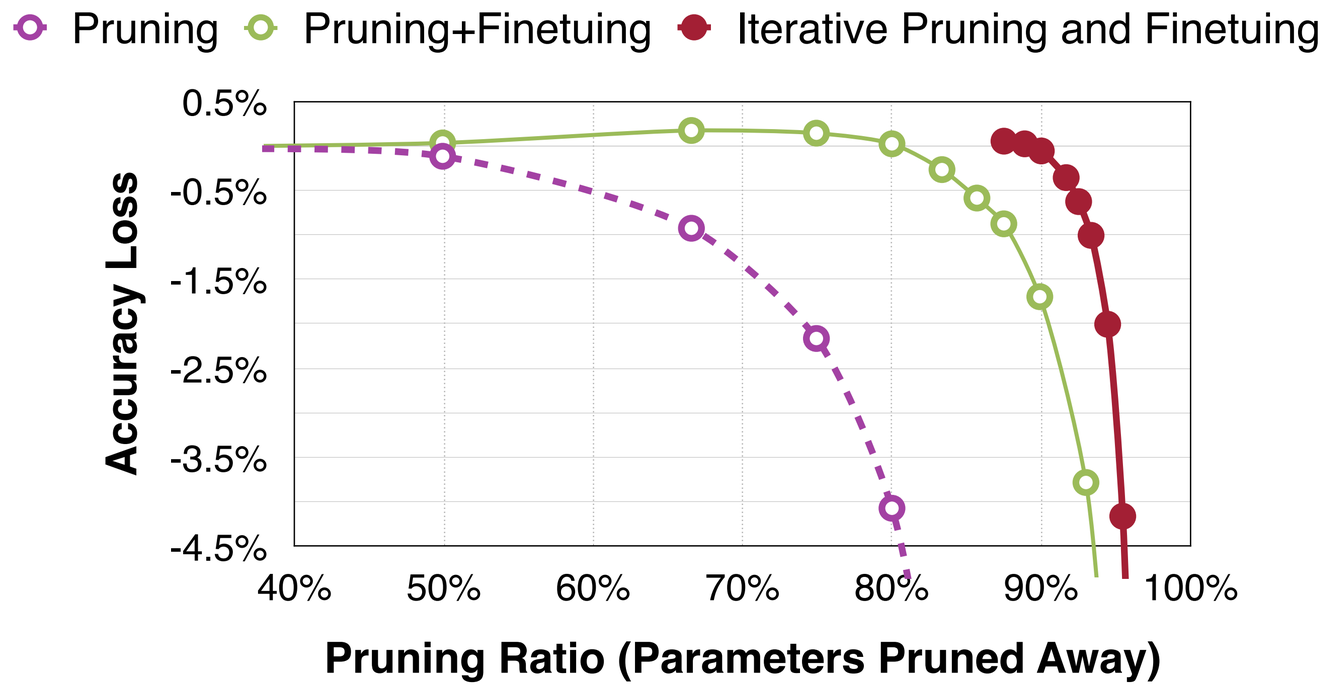


Learning Both Weights and Connections for Efficient Neural Network [Han *et al.*, NeurIPS 2015]

Iterative Pruning

- Consider pruning followed by a fine-tuning is one iteration.
- Iterative pruning gradually increases the target sparsity in each iteration.
  - boost pruning ratio from 5× to 9× on AlexNet compared to single-step aggressive pruning.





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

Regularization

- When training neural networks or fine-tuning quantized neural networks, regularization is added to the loss term to
  - ***penalize non-zero parameters***
  - ***encourage smaller parameters***
- The most common regularization for improving performance of pruning is L1/L2 regularization.
  - L1-Regularization

$$L' = L(\mathbf{x}; \mathbf{W}) + \lambda |\mathbf{W}|$$

  - L2-Regularization

$$L' = L(\mathbf{x}; \mathbf{W}) + \lambda ||\mathbf{W}||^2$$

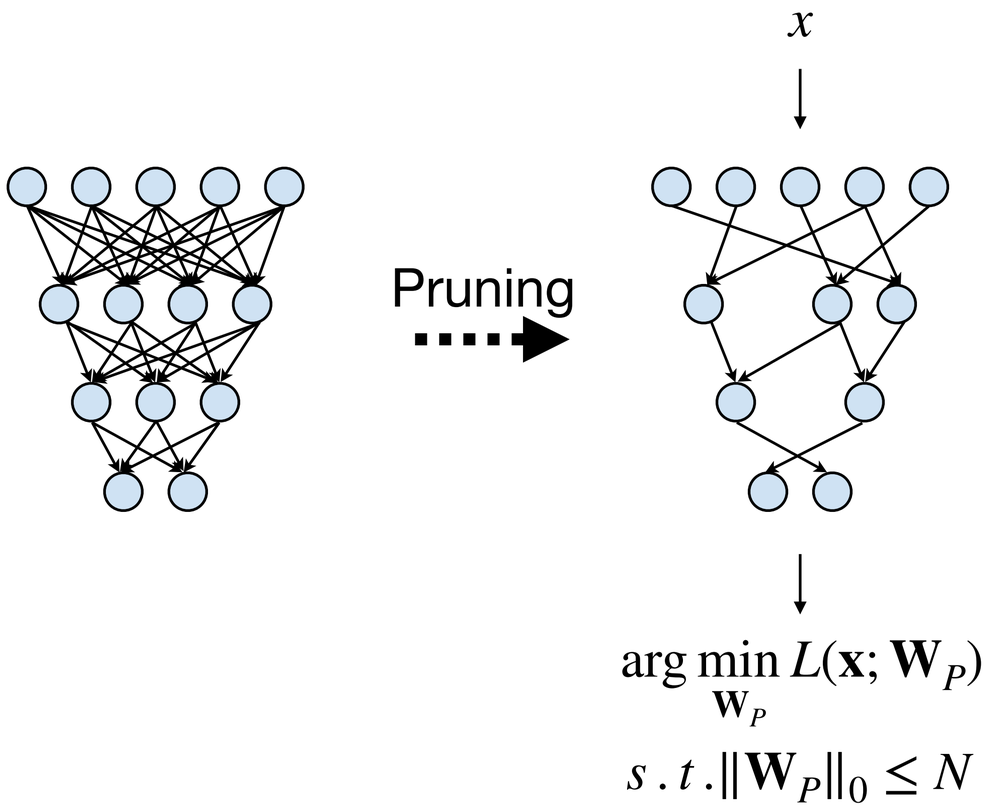
  - **Examples:**
    - Magnitude-based Fine-grained Pruning applies L2 regularization on weights
    - Network Slimming applies smooth-L1 regularization on channel scaling factors.

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

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

Neural Network Pruning

- **Introduction to Pruning**
  - What is pruning?
  - How should we formulate pruning?
- **Determine the Pruning Granularity**
  - In what pattern should we prune the neural network?
- **Determine the Pruning Criterion**
  - What synapses/neurons should we prune?
- **Determine the Pruning Ratio**
  - What should target sparsity be for each layer?
- **Fine-tune/Train Pruned Neural Network**
  - How should we improve performance of pruned models?



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

Neural Network Pruning

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

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

nonconvex problem

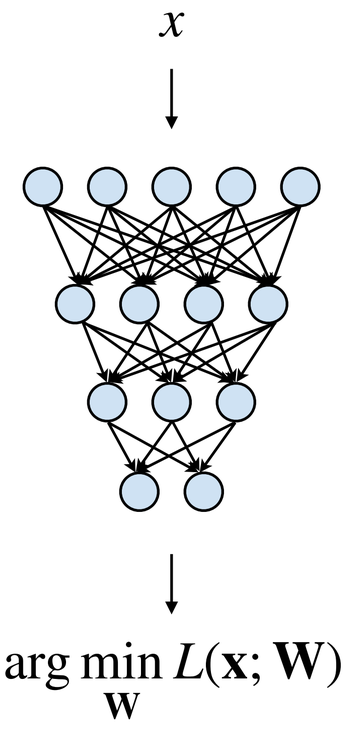
L is differentiable

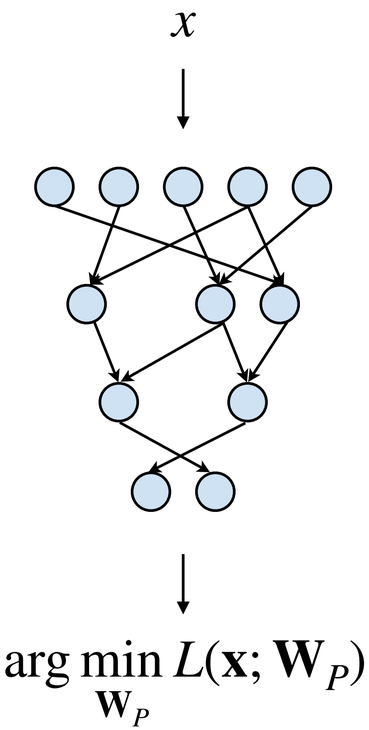
subject to

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

combinatorial constraint

  - 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.





ADMM

- Rewrite the pruning problem in a form without constraint as

$$\arg\min_{\mathbf{W}_P} L(\mathbf{x}; \mathbf{W}_P) + g(\mathbf{W}_P), \quad \text{where} \quad g(\mathbf{W}_P) = \begin{cases} 0, & \|\mathbf{W}_P\|_0 \leq N \\ \infty, & otherwise \end{cases}$$

- Split the variables and decompose the problem into simpler subproblems as

$$\arg\min_{\mathbf{W}_P} L(\mathbf{x}; \mathbf{W}_P) + g(\mathbf{Z}), \quad \text{subject to} \quad \mathbf{W_P} = \mathbf{Z}$$

- The **augmented Lagrangian** of the above problem is

$$\mathscr{L} = L\left(x; \mathbf{W}_P\right) + g(\mathbf{Z}) + tr\left[\Lambda^T(\mathbf{W}_P - \mathbf{Z})\right] + \rho\|\mathbf{W}_P - \mathbf{Z}\|^2$$

$$\mathscr{L} = L\left(x; \mathbf{W}_P\right) + g(\mathbf{Z}) + \frac{\rho}{2}\|\mathbf{W}_P - \mathbf{Z} + \mathbf{U}\|_2^2 - \frac{\rho}{2}\|\mathbf{U}\|_2^2$$

A Systematic DNN Weight Pruning Framework using Alternating Direction Method of Multipliers [Zhang *et al.*, ECCV 2018]

ADMM

- The augmented Lagrangian of the above problem is

$$\mathscr{L} = L\left(x; \mathbf{W}_P\right) + g(\mathbf{Z}) + tr\left[\Lambda^T(\mathbf{W}_P - \mathbf{Z})\right] + \rho\|\mathbf{W}_P - \mathbf{Z}\|^2$$

$$\mathscr{L} = L\left(x; \mathbf{W}_P\right) + g(\mathbf{Z}) + \frac{\rho}{2}\|\mathbf{W}_P - \mathbf{Z} + \mathbf{U}\|_2^2 - \frac{\rho}{2}\|\mathbf{U}\|_2^2$$

- The ADMM algorithm proceeds by repeating the following steps for $k = 0,1,\cdots,$ iterations

$$\mathbf{W}_P^{(k+1)} := \arg\min_{\mathbf{W}_P} \mathscr{L} = \arg\min_{\mathbf{W}_P} L\left(x; \mathbf{W}_P\right) + \frac{\rho}{2}\|\mathbf{W}_P - \mathbf{Z}^{(k)} + \mathbf{U}^{(k)}\|_2^2$$

$$\mathbf{Z}^{(k+1)} := \arg\min_{\mathbf{Z}} \mathscr{L} = \arg\min_{\mathbf{Z}} g(\mathbf{Z}) + \frac{\rho}{2}\|\mathbf{W}_P^{(k+1)} - \mathbf{Z} + \mathbf{U}^{(k)}\|_2^2$$

$$\mathbf{U}^{(k+1)} := \mathbf{W}_P^{(k+1)} - \mathbf{Z}^{(k+1)} + \mathbf{U}^{(k)}$$

A Systematic DNN Weight Pruning Framework using Alternating Direction Method of Multipliers [Zhang *et al.*, ECCV 2018]

ADMM

- The augmented Lagrangian of the above problem is

$$\mathscr{L} = L\left(x; \mathbf{W}_P\right) + g(\mathbf{Z}) + tr\left[\Lambda^T(\mathbf{W}_P - \mathbf{Z})\right] + \rho\|\mathbf{W}_P - \mathbf{Z}\|^2$$

$$\mathscr{L} = L\left(x; \mathbf{W}_P\right) + g(\mathbf{Z}) + \frac{\rho}{2}\|\mathbf{W}_P - \mathbf{Z} + \mathbf{U}\|_2^2 - \frac{\rho}{2}\|\mathbf{U}\|_2^2$$

- The ADMM algorithm proceeds by repeating the following steps for $k = 0,1,\cdots,$ iterations

$$\mathbf{W}_P^{(k+1)} := \arg\min_{\mathbf{W}_P} \mathscr{L} = \arg\min_{\mathbf{W}_P} L\left(x; \mathbf{W}_P\right) + \boxed{\frac{\rho}{2}\|\mathbf{W}_P - \mathbf{Z}^{(k)} + \mathbf{U}^{(k)}\|_2^2}$$

L2 regularization

Differentiable

Gradient Descent

$$\mathbf{Z}^{(k+1)} := \arg\min_{\mathbf{Z}} \mathscr{L} = \arg\min_{\mathbf{Z}} g(\mathbf{Z}) + \frac{\rho}{2}\|\mathbf{W}_P^{(k+1)} - \mathbf{Z} + \mathbf{U}^{(k)}\|_2^2$$

$$\mathbf{U}^{(k+1)} := \mathbf{W}_P^{(k+1)} - \mathbf{Z}^{(k+1)} + \mathbf{U}^{(k)}$$

A Systematic DNN Weight Pruning Framework using Alternating Direction Method of Multipliers [Zhang *et al.*, ECCV 2018]

ADMM

- The augmented Lagrangian of the above problem is

$$\mathscr{L} = L\left(x; \mathbf{W}_P\right) + g(\mathbf{Z}) + tr\left[\Lambda^T(\mathbf{W}_P - \mathbf{Z})\right] + \rho\|\mathbf{W}_P - \mathbf{Z}\|^2$$

$$\mathscr{L} = L\left(x; \mathbf{W}_P\right) + g(\mathbf{Z}) + \frac{\rho}{2}\|\mathbf{W}_P - \mathbf{Z} + \mathbf{U}\|_2^2 - \frac{\rho}{2}\|\mathbf{U}\|_2^2$$

- The ADMM algorithm proceeds by repeating the following steps for $k = 0,1,\cdots,$ iterations

$$\mathbf{W}_P^{(k+1)} := \arg\min_{\mathbf{W}_P} \mathscr{L} = \arg\min_{\mathbf{W}_P} L\left(x; \mathbf{W}_P\right) + \frac{\rho}{2}\|\mathbf{W}_P - \mathbf{Z}^{(k)} + \mathbf{U}^{(k)}\|_2^2$$

$$\mathbf{Z}^{(k+1)} := \arg\min_{\mathbf{Z}} \mathscr{L} = \arg\min_{\mathbf{Z}} g(\mathbf{Z}) + \frac{\rho}{2}\|\mathbf{W}_P^{(k+1)} - \mathbf{Z} + \mathbf{U}^{(k)}\|_2^2$$

Solve Analytically

$$\mathbf{U}^{(k+1)} := \mathbf{W}_P^{(k+1)} - \mathbf{Z}^{(k+1)} + \mathbf{U}^{(k)}$$

Keep N elements of $\mathbf{W}_P^{(k+1)} + \mathbf{U}^{(k)}$ with the largest magnitudes and set the reset to zero

$$\mathbf{Z}^{(k+1)} := \arg\min_{\mathbf{Z}} \|\mathbf{W}_P^{(k+1)} + \mathbf{U}^{(k)} - \mathbf{Z}\|_2^2, \quad s.t. \quad \|\mathbf{Z}\|_0 \leq N$$

A Systematic DNN Weight Pruning Framework using Alternating Direction Method of Multipliers [Zhang *et al.*, ECCV 2018]

Lottery Ticket Hypothesis

Can we train a sparse neural network from scratch?

Train Sparse Neural Network From Scratch

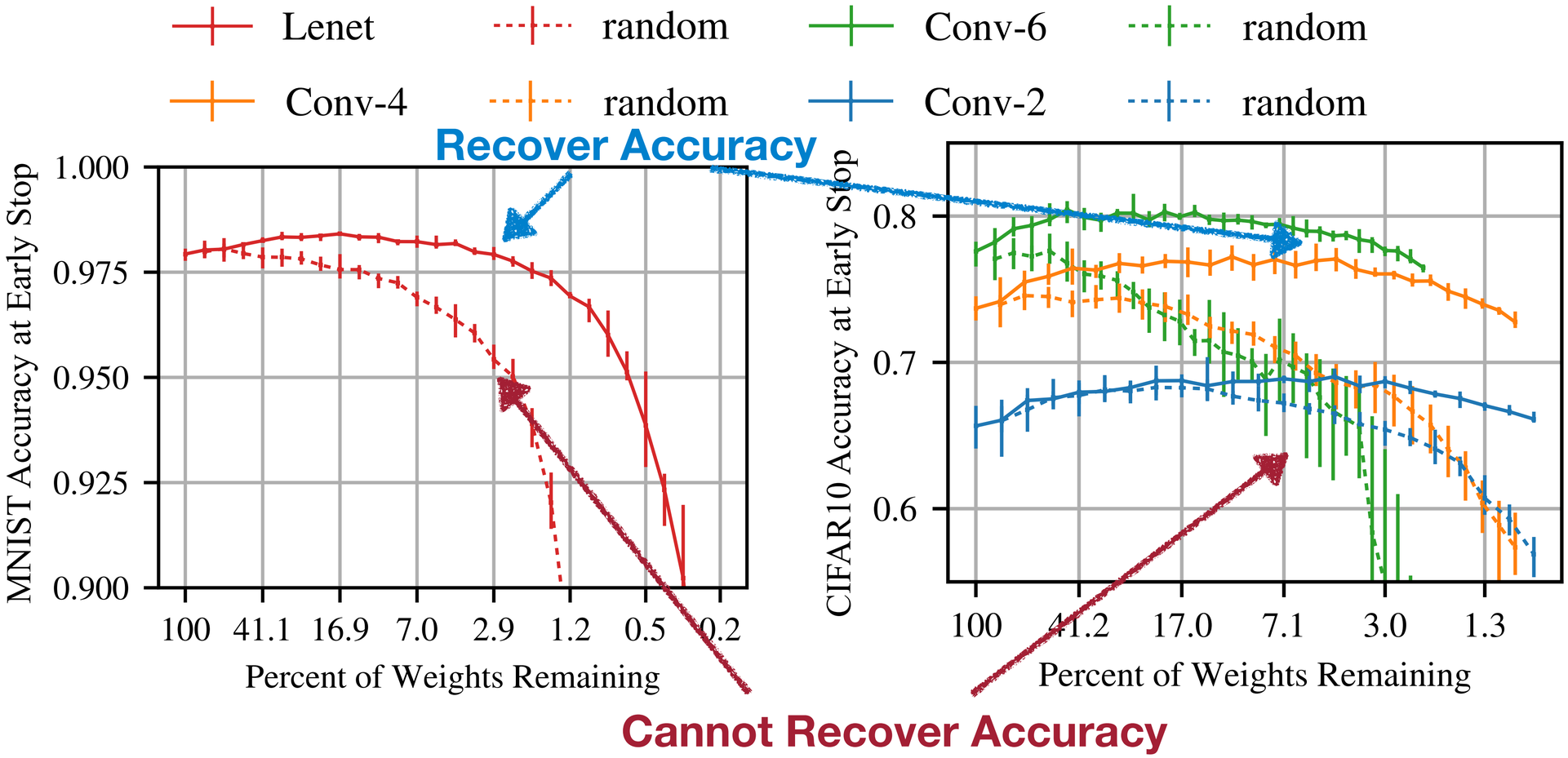
- Neural Network Pruning shows that
  - **a neural network can be reduced in size.**

- Question: **Can we directly train this sparse neural network from scratch?**

- Contemporary experience tells us that
  - **the architectures uncovered by pruning are harder to train from the start,**
  - **reaching lower accuracy than the original networks**

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

The Lottery Ticket Hypothesis



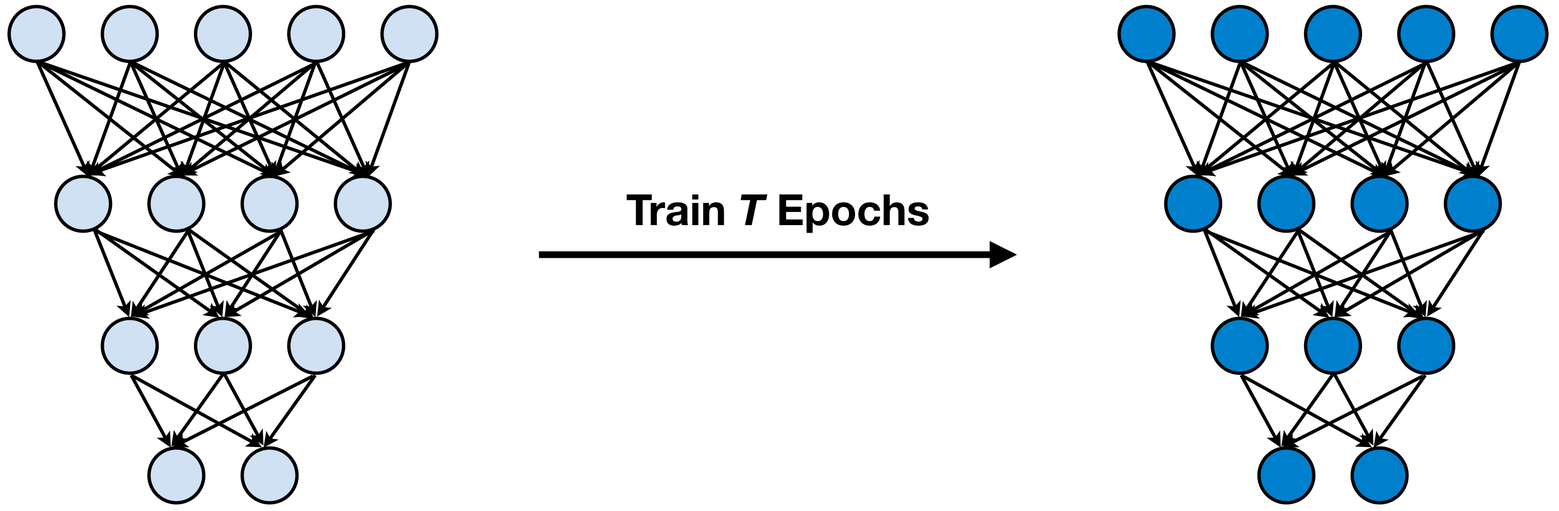
- Dashed Lines: randomly sampled sparse networks
- Solid Lines: correct sparsitiy mask (found by training) & same initialization.

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

The Lottery Ticket Hypothesis

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

—The Lottery Ticket Hypothesis

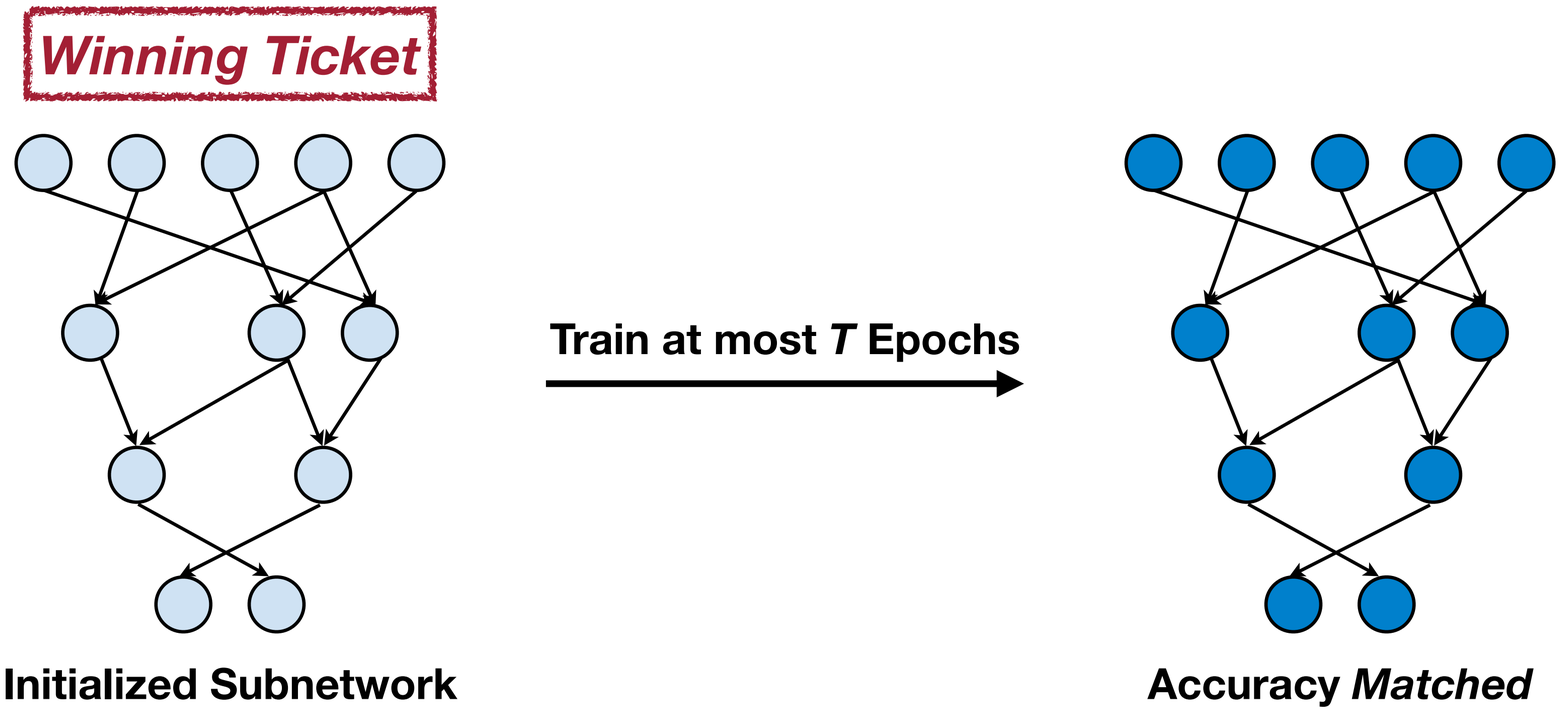


The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks [Frankle *et al.*, ICLR 2019]

The Lottery Ticket Hypothesis

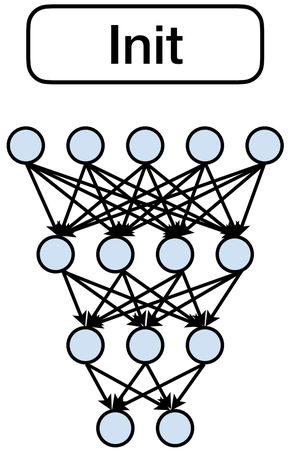
A randomly-initialized, dense neural network contains ***a subnetwork*** *that is initialized such that—when* ***trained in isolation*** *—it can* ***match the test accuracy*** *of the original network after training for* ***at most the same number of iterations****.*

—The Lottery Ticket Hypothesis



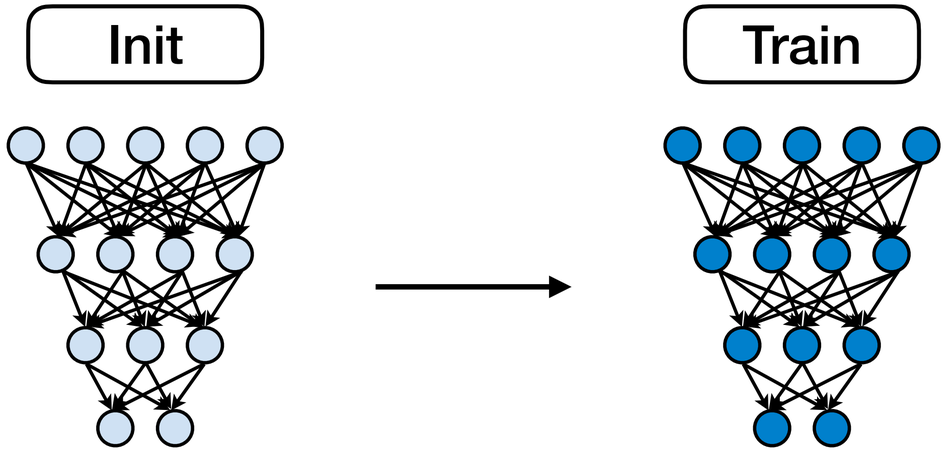
The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks [Frankle *et al.*, ICLR 2019]

Iterative Magnitude Pruning



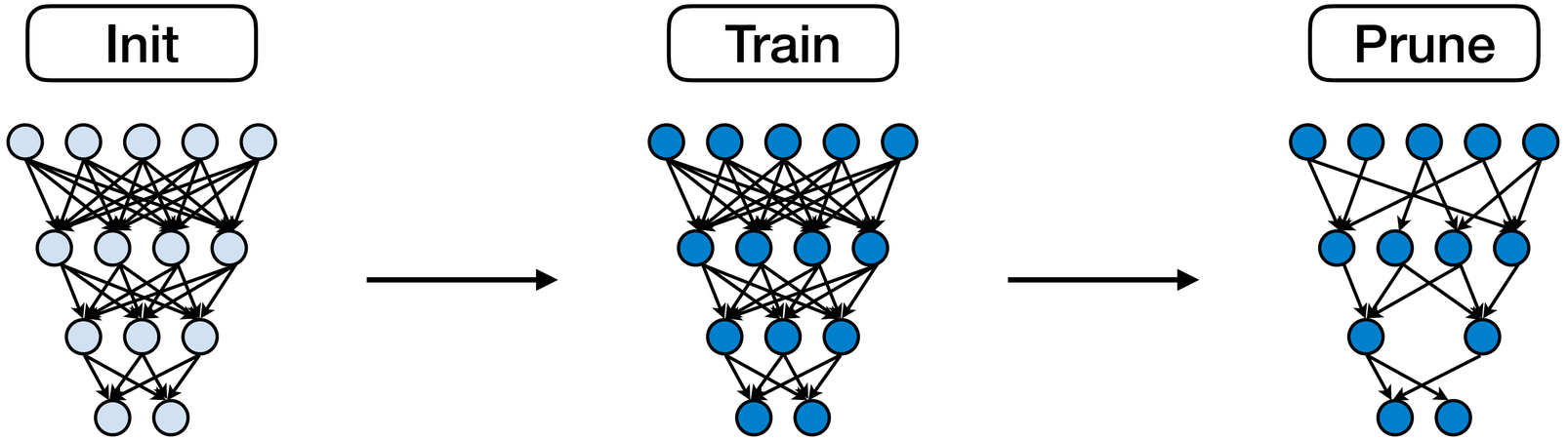
The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks [Frankle *et al.*, ICLR 2019]

Iterative Magnitude Pruning



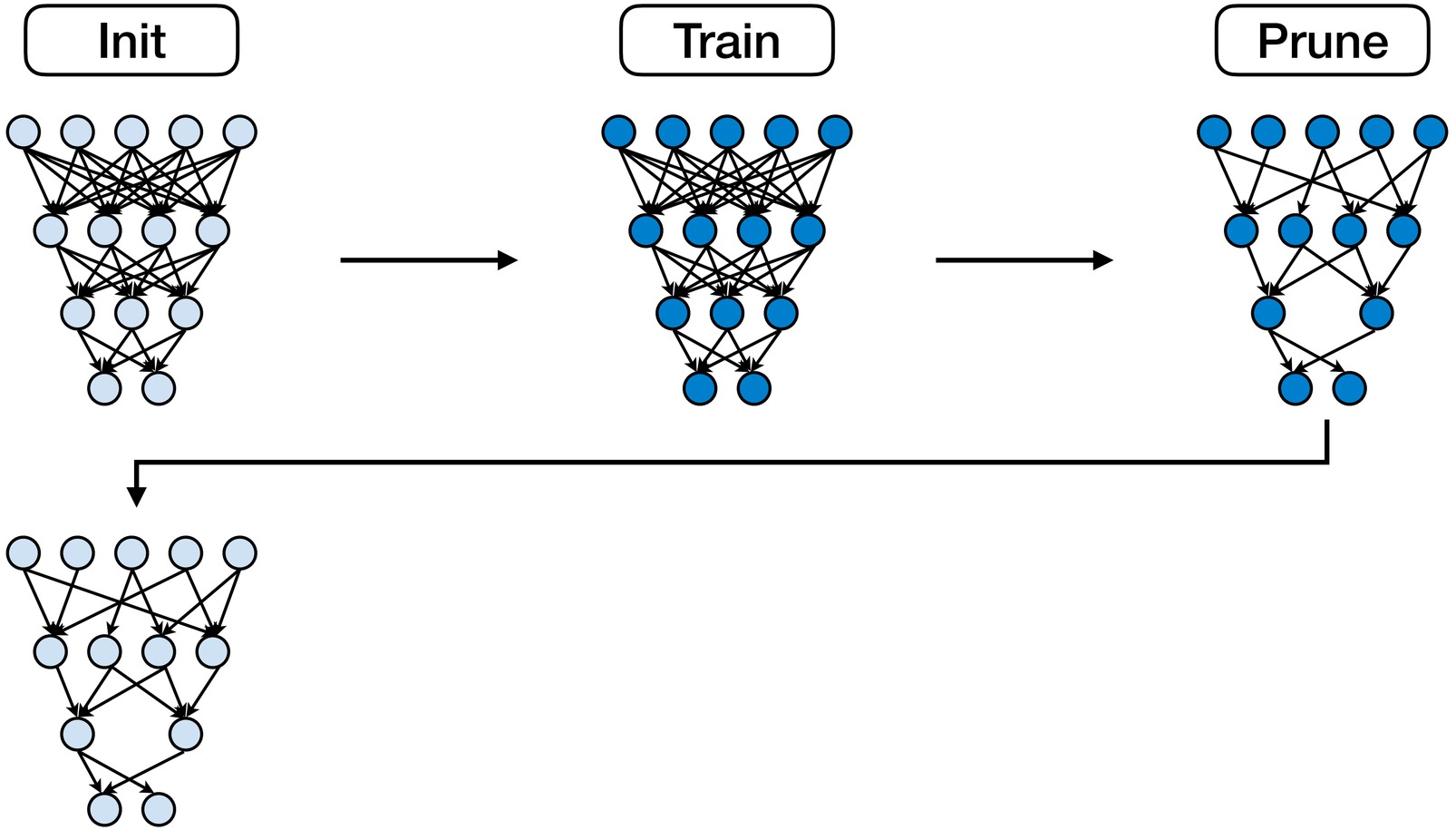
The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks [Frankle *et al.*, ICLR 2019]

Iterative Magnitude Pruning



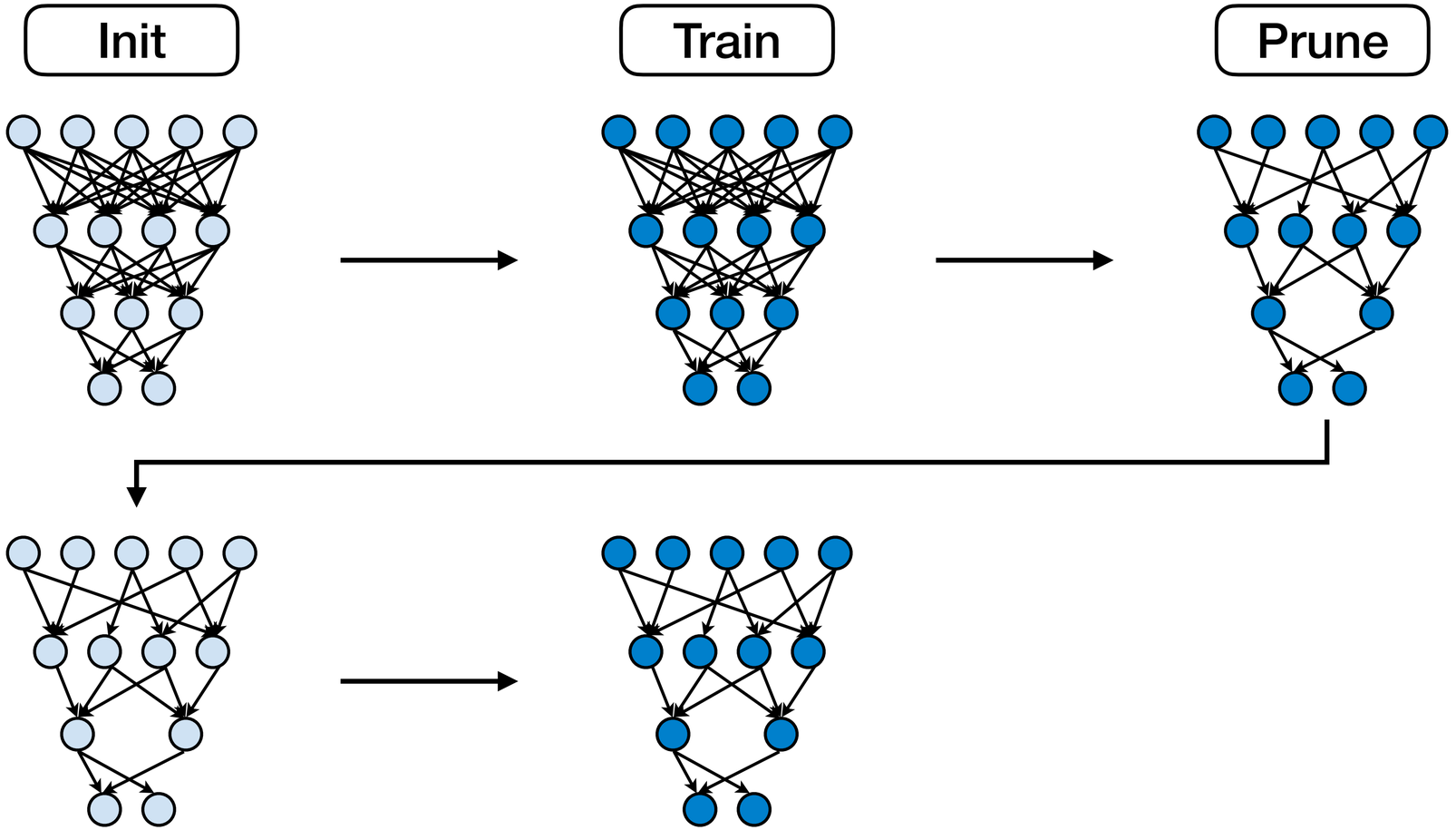
The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks [Frankle *et al.*, ICLR 2019]

Iterative Magnitude Pruning



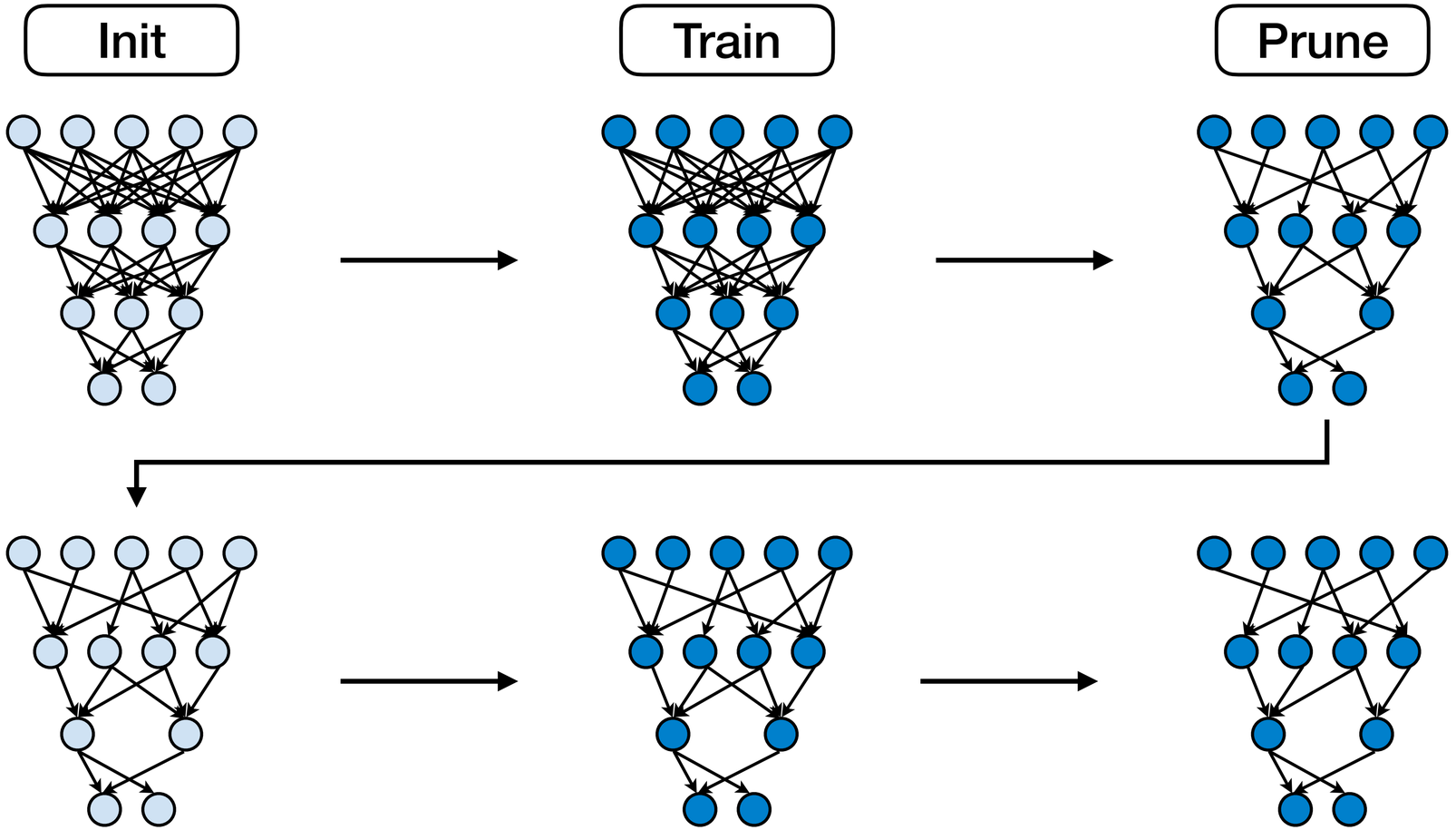
The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks [Frankle *et al.*, ICLR 2019]

Iterative Magnitude Pruning



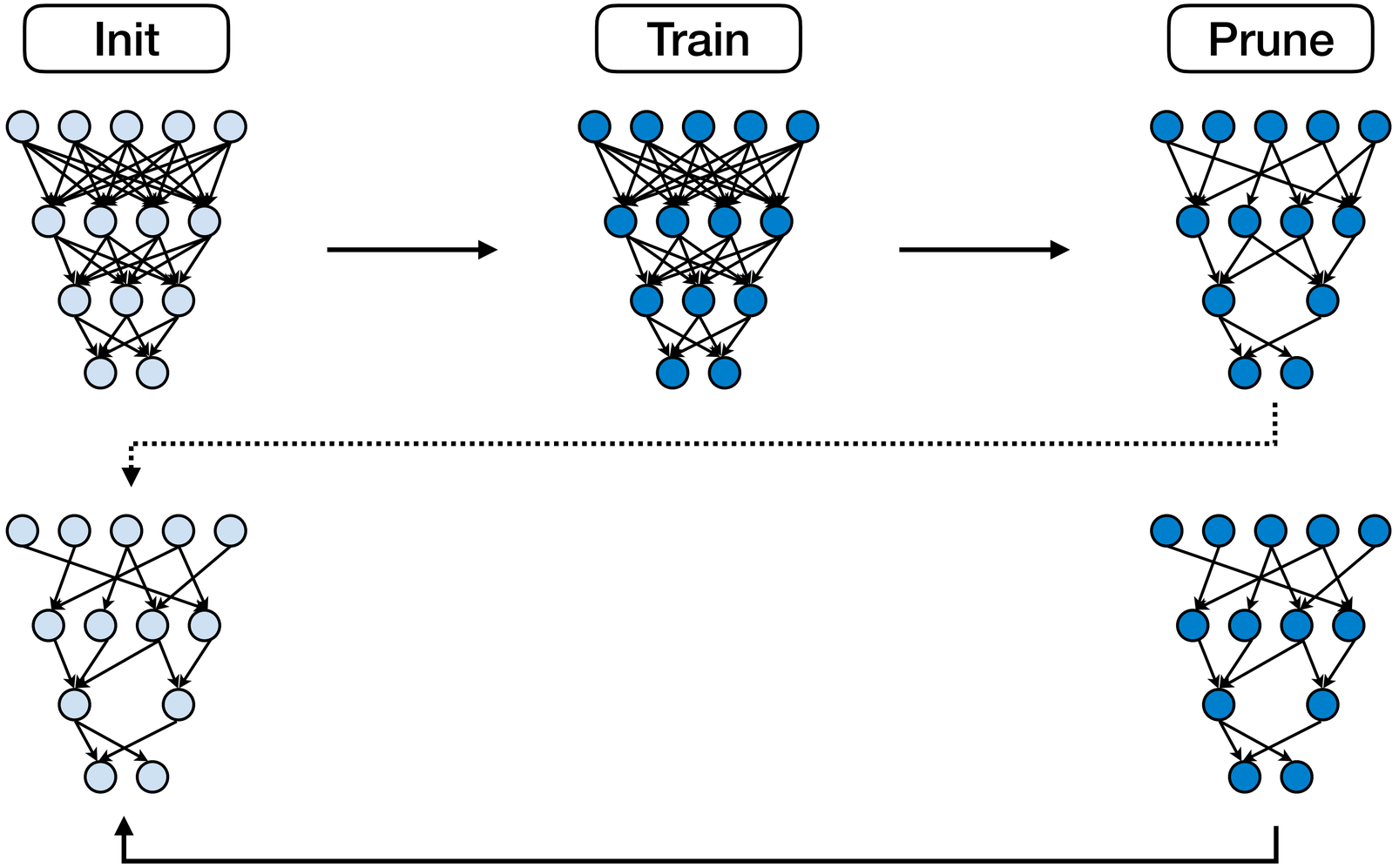
The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks [Frankle *et al.*, ICLR 2019]

Iterative Magnitude Pruning



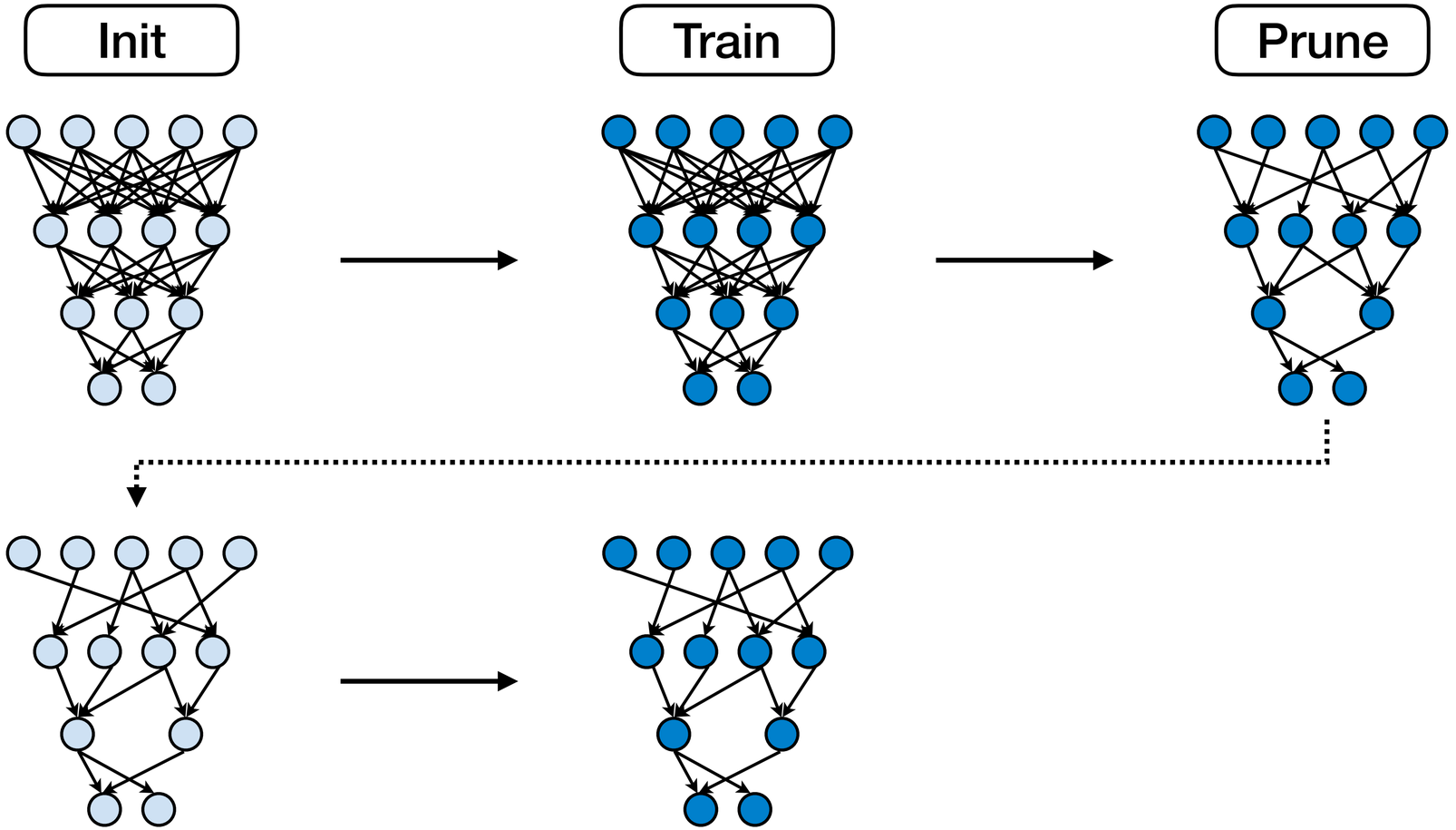
The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks [Frankle *et al.*, ICLR 2019]

Iterative Magnitude Pruning



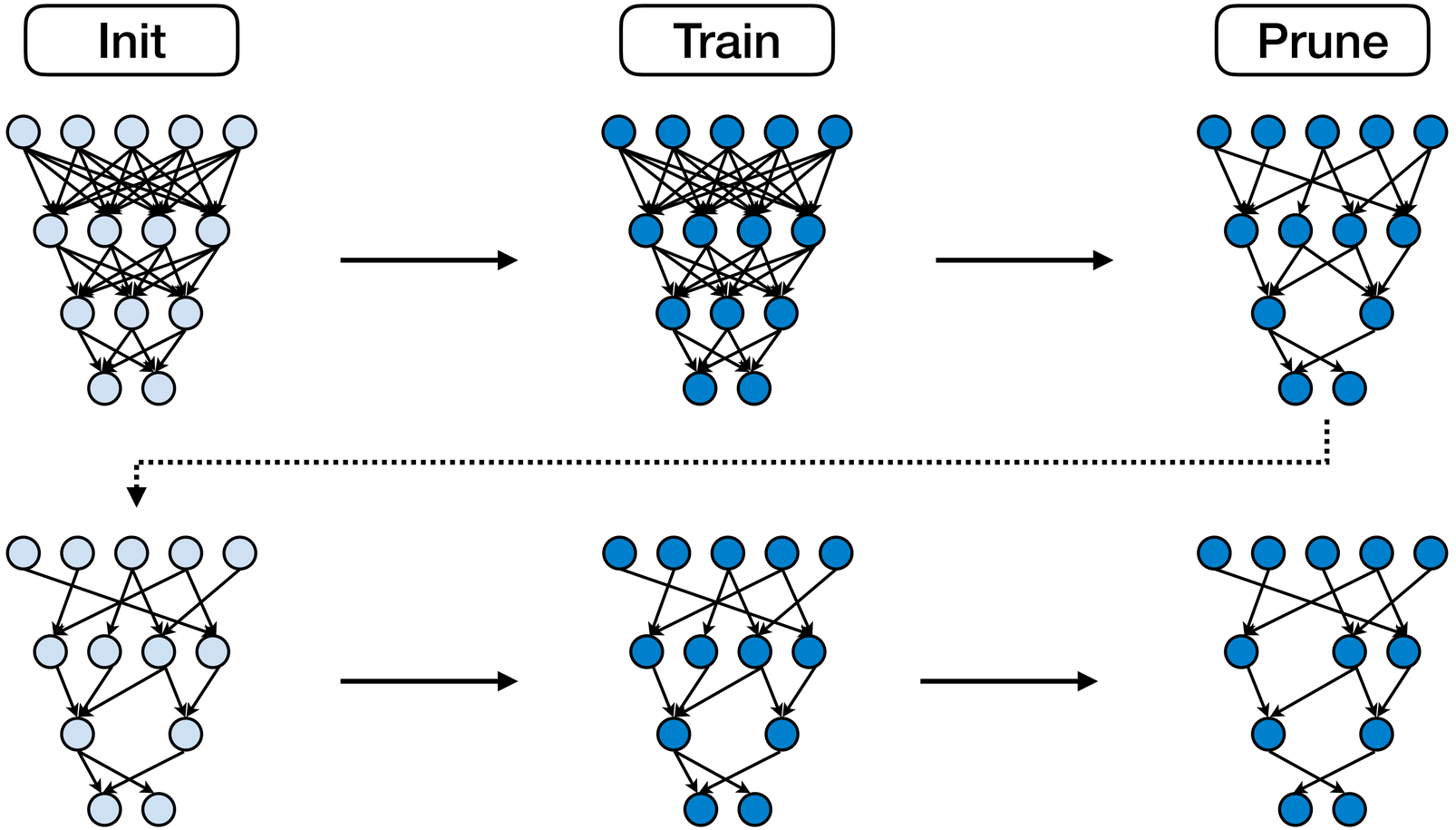
The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks [Frankle *et al.*, ICLR 2019]

Iterative Magnitude Pruning



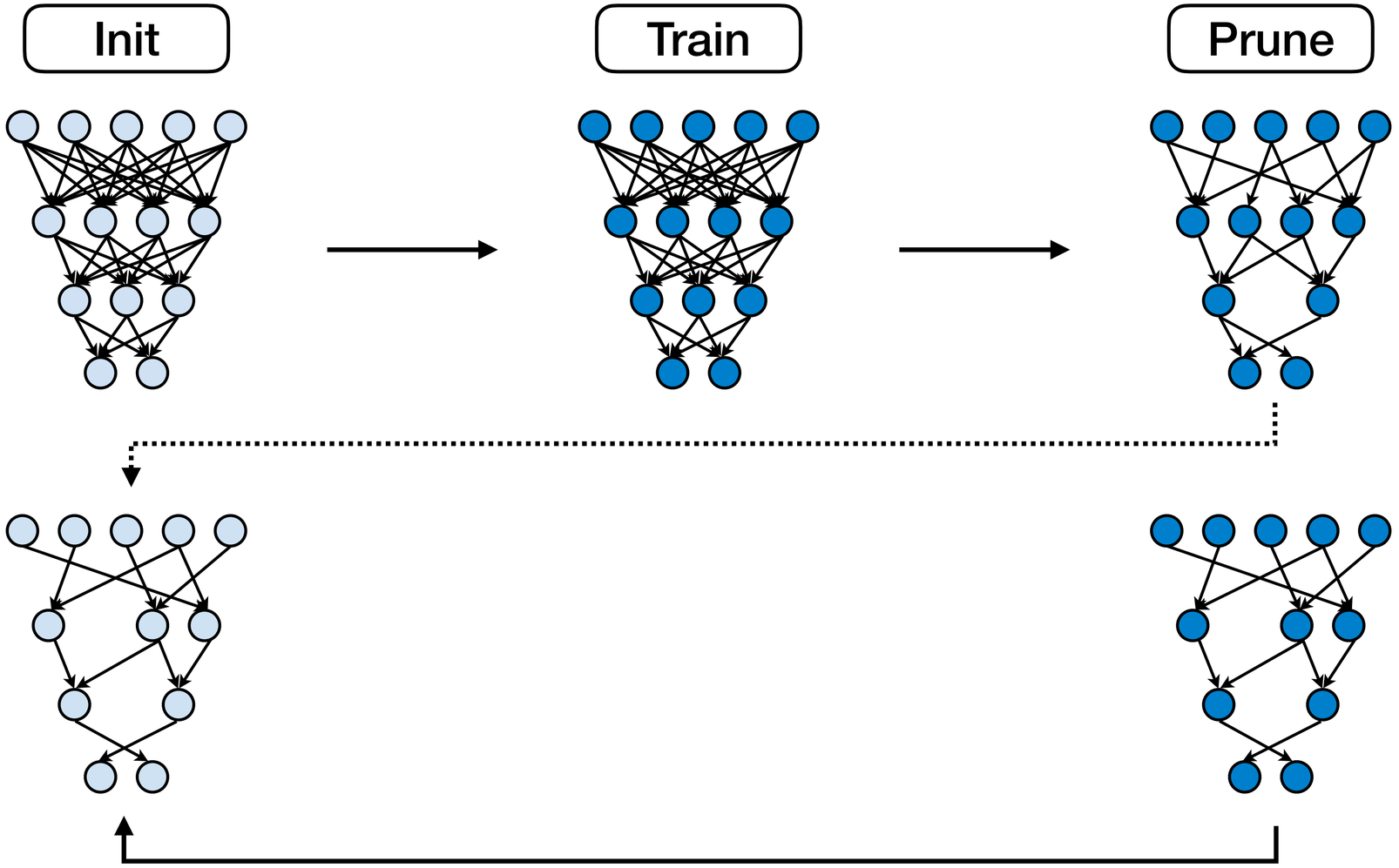
The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks [Frankle *et al.*, ICLR 2019]

Iterative Magnitude Pruning



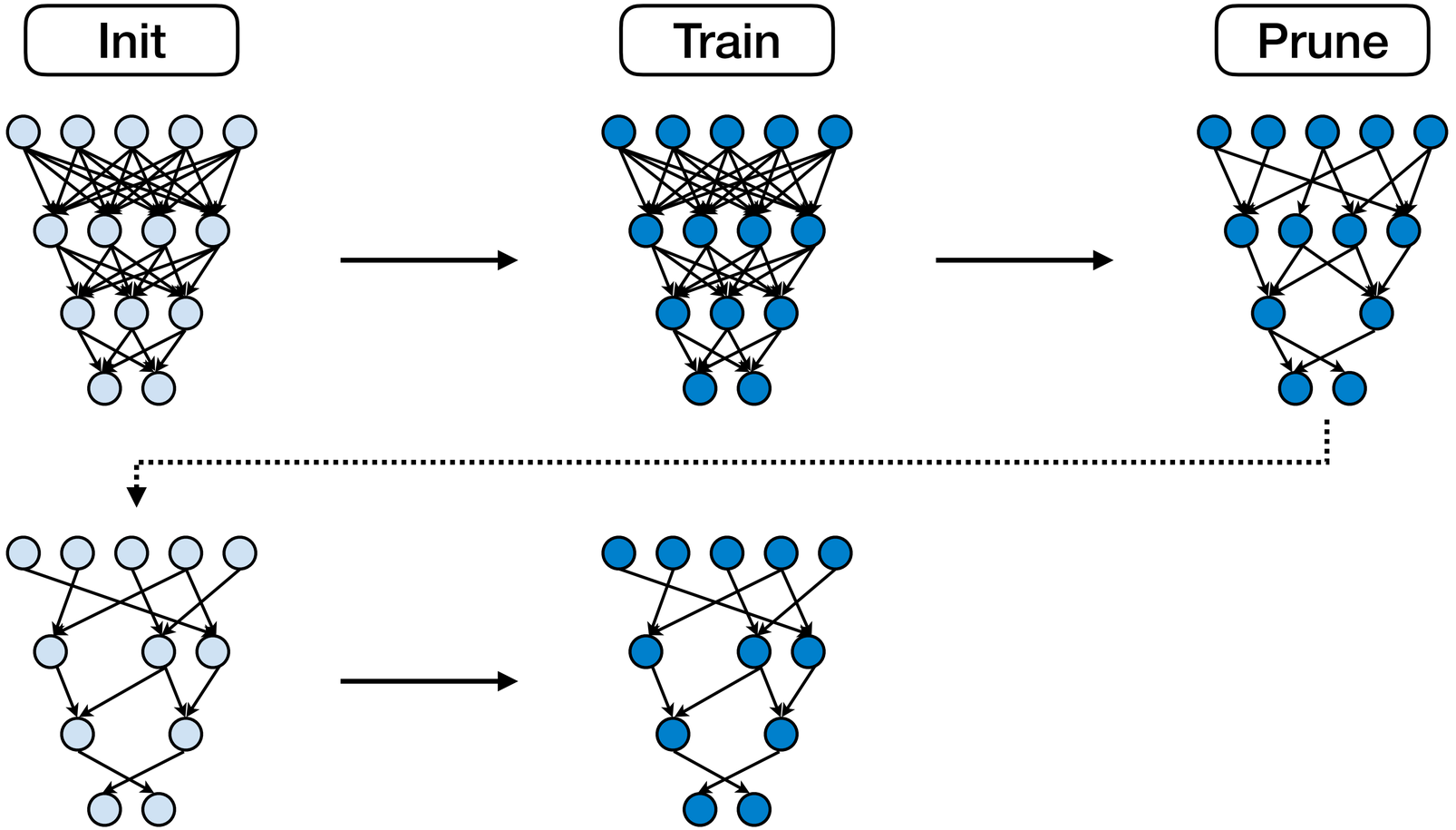
The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks [Frankle *et al.*, ICLR 2019]

Iterative Magnitude Pruning



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

Iterative Magnitude Pruning

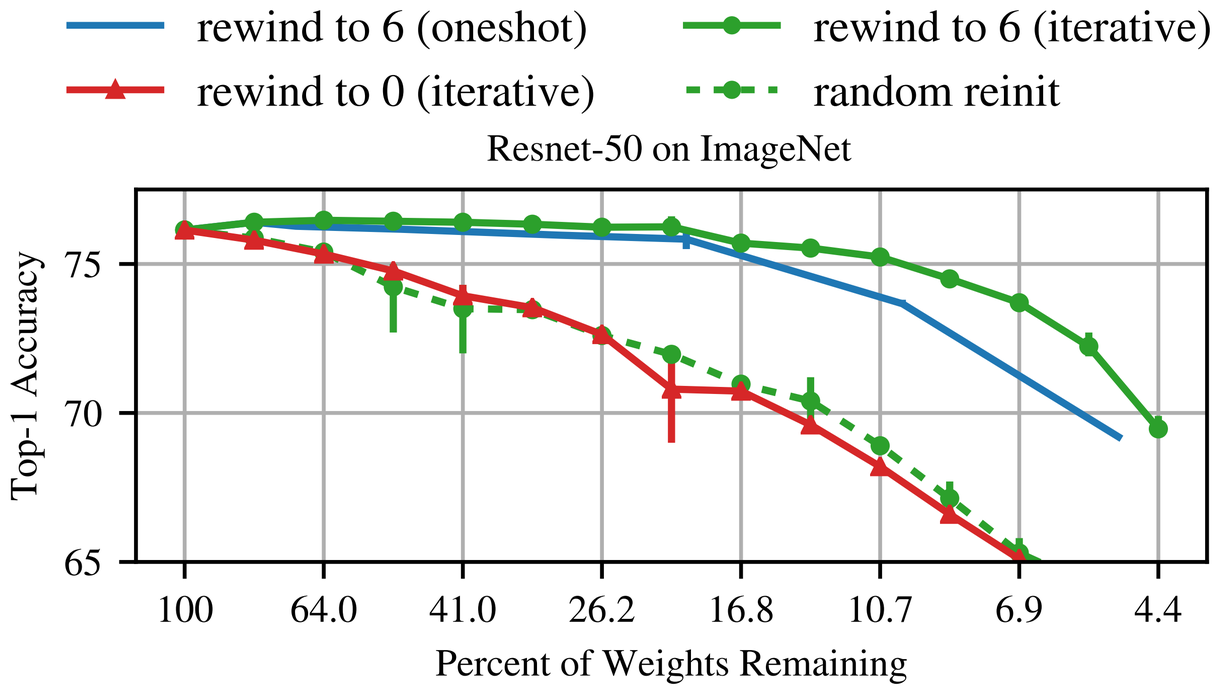


The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks [Frankle *et al.*, ICLR 2019]

Scaling Limitation

- Resetting the weights to the very initial value $\mathbf{W}_{t=0}$ works for small-scale tasks such as MNIST and CIFAR-10, and fails on deep networks.

- Instead, it is possible to robustly obtain pruned subnetworks by resetting the weights to the values after ***a small number of k training iterations***, that is $\mathbf{W}_{t=k}$.



Total Training Epochs: $T = 90$
Rewinding to: $k = 6$

Stabilizing the Lottery Ticket Hypothesis [Frankle *et al.*, arXiv 2019]

Scaling Limitation

- Resetting the weights to the very initial value $\mathbf{W}_{t=0}$ works for small-scale tasks such as MNIST and CIFAR-10, and fails on deep networks.

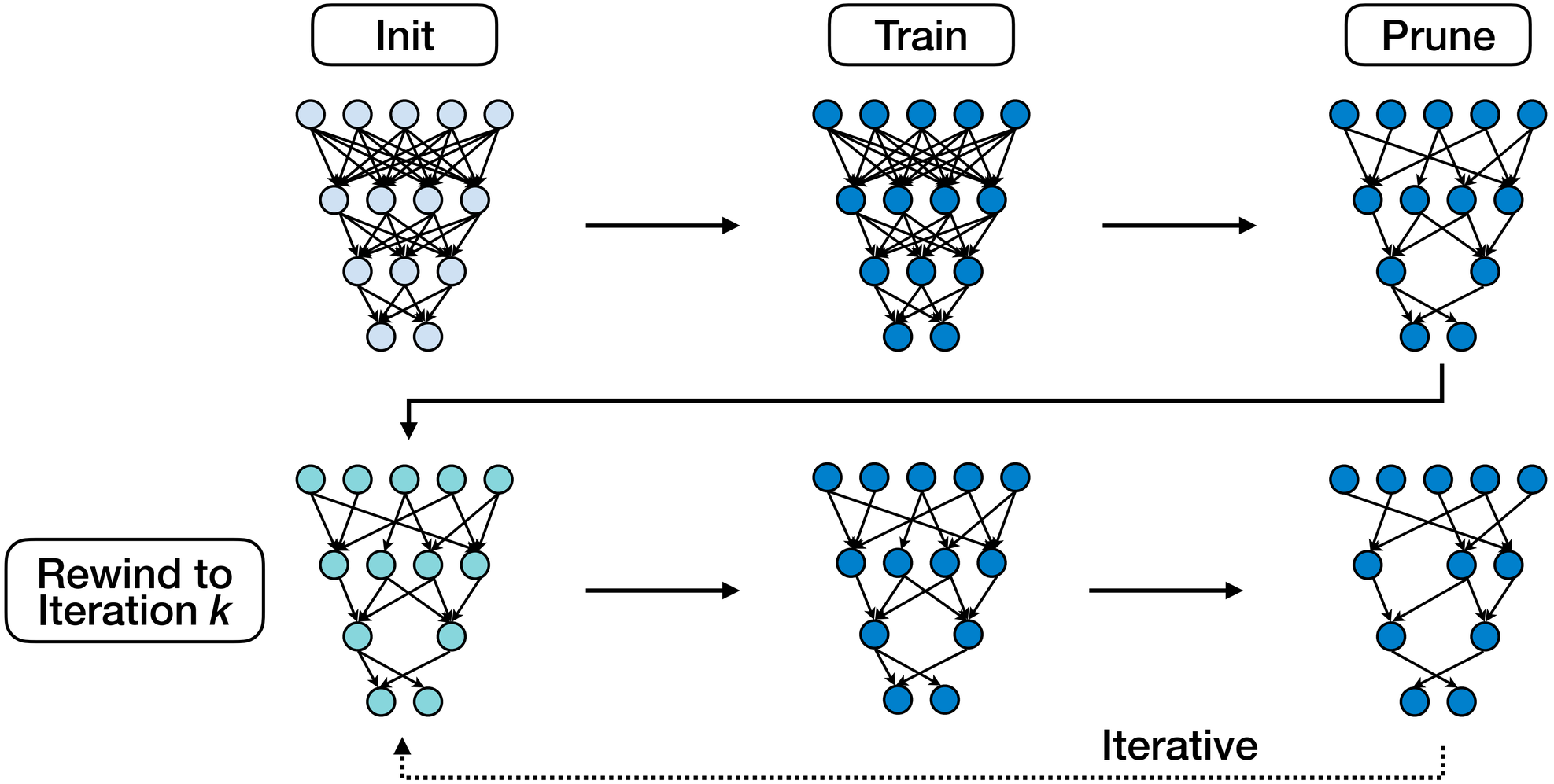
- Instead, it is possible to robustly obtain pruned subnetworks by resetting the weights to the values after ***a small number of k training iterations***, that is $\mathbf{W}_{t=k}$.

Consider a dense, randomly-initialized neural network $f\left(\mathbf{x}; \mathbf{W}_0\right)$ that trains to accuracy a^* in T^* iterations. Let $\mathbf{W}_t$ be the weights at iteration t of training. There exist an iteration $k \ll T^*$ and fixed pruning mask $\mathbf{m} \in \{0,1\}^{|\mathbf{W}|}$ (where $\|\mathbf{m}\|_1 \ll |\mathbf{W}|$) such that subnetwork $\mathbf{m} \odot \mathbf{W}_k$ trains to accuracy $a \geq a^*$ in $T \leq T^* - k$ iterations.

—The Lottery Ticket Hypothesis with Rewinding

Stabilizing the Lottery Ticket Hypothesis [Frankle *et al.*, arXiv 2019]

Iterative Magnitude Pruning with Rewinding

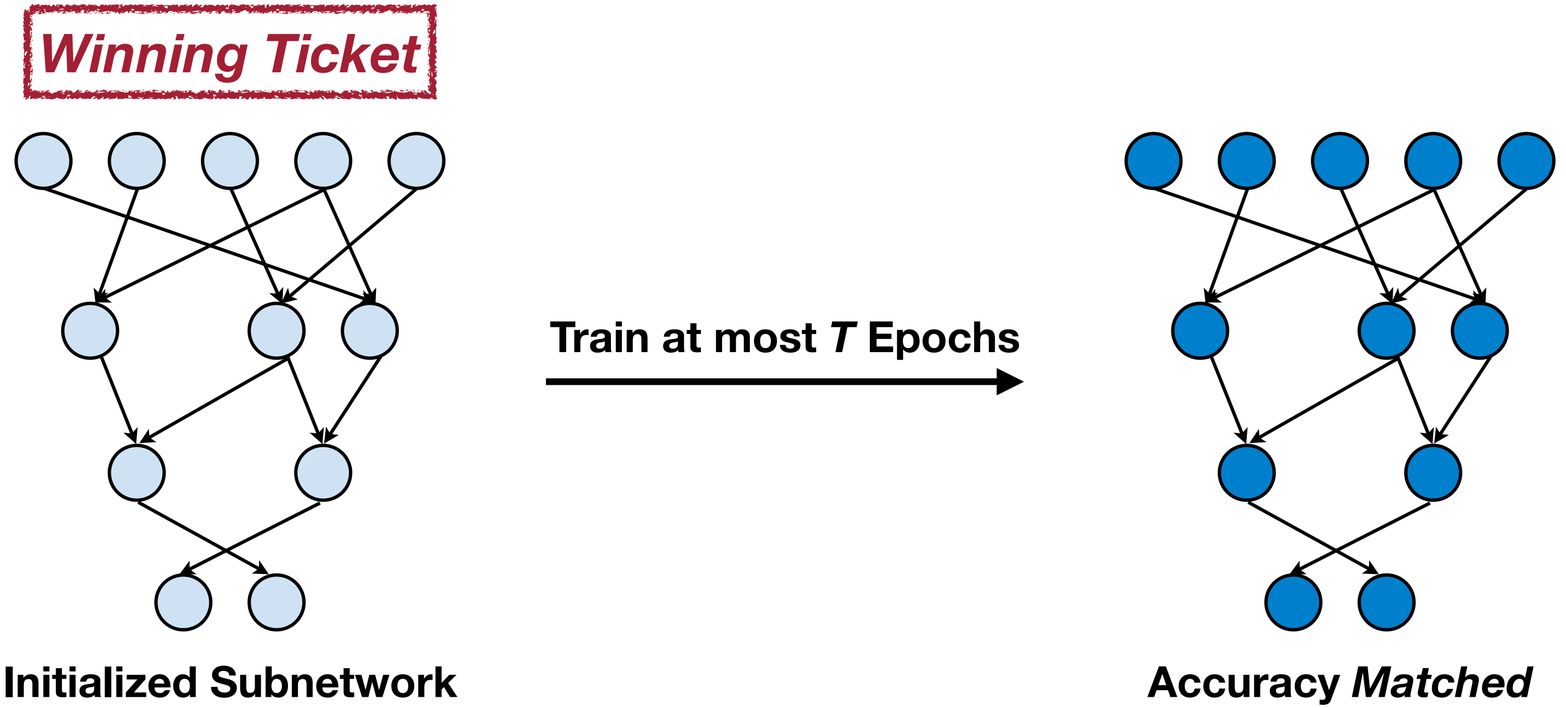


Stabilizing the Lottery Ticket Hypothesis [Frankle *et al.*, arXiv 2019]

The Lottery Ticket Hypothesis

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

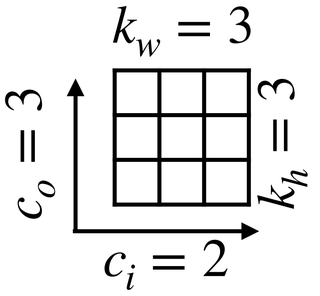
—The Lottery Ticket Hypothesis



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

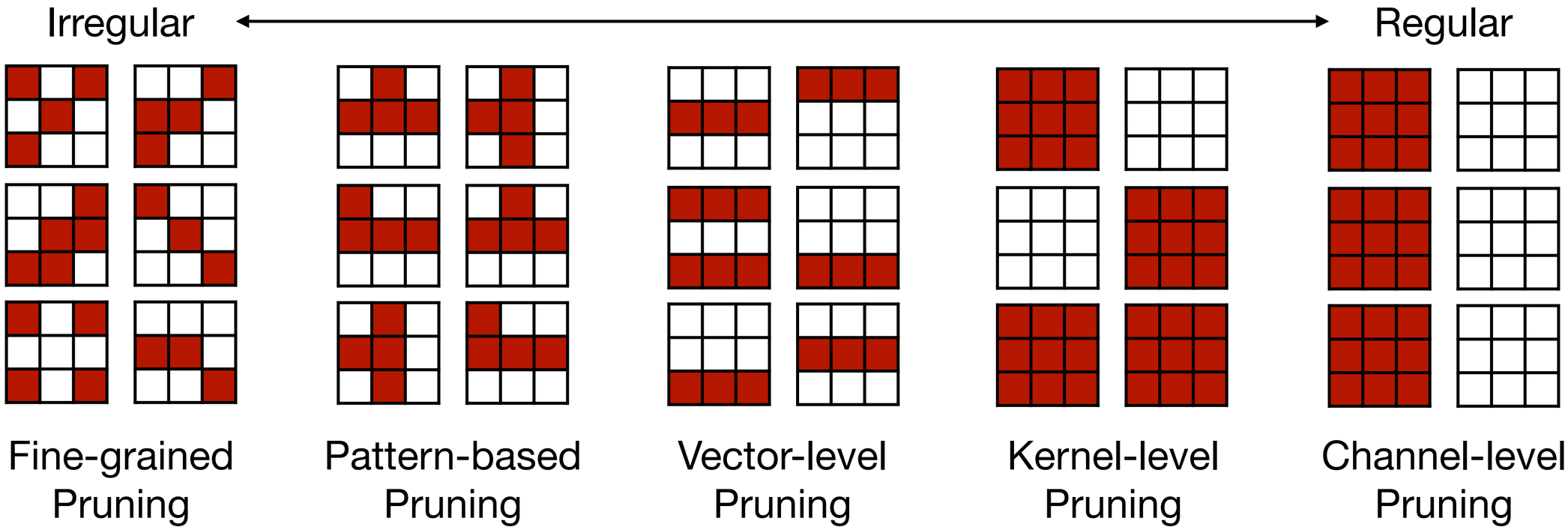
System Support for Sparsity

Recall: Different Pruning Granularity



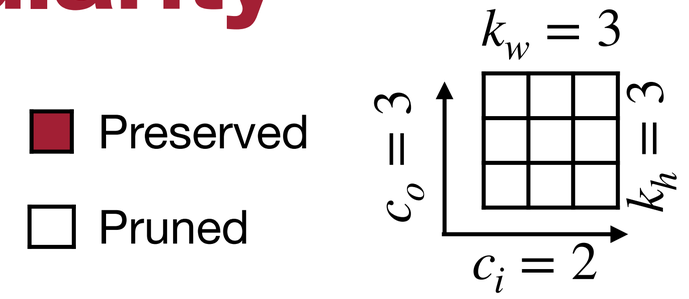
Preserved

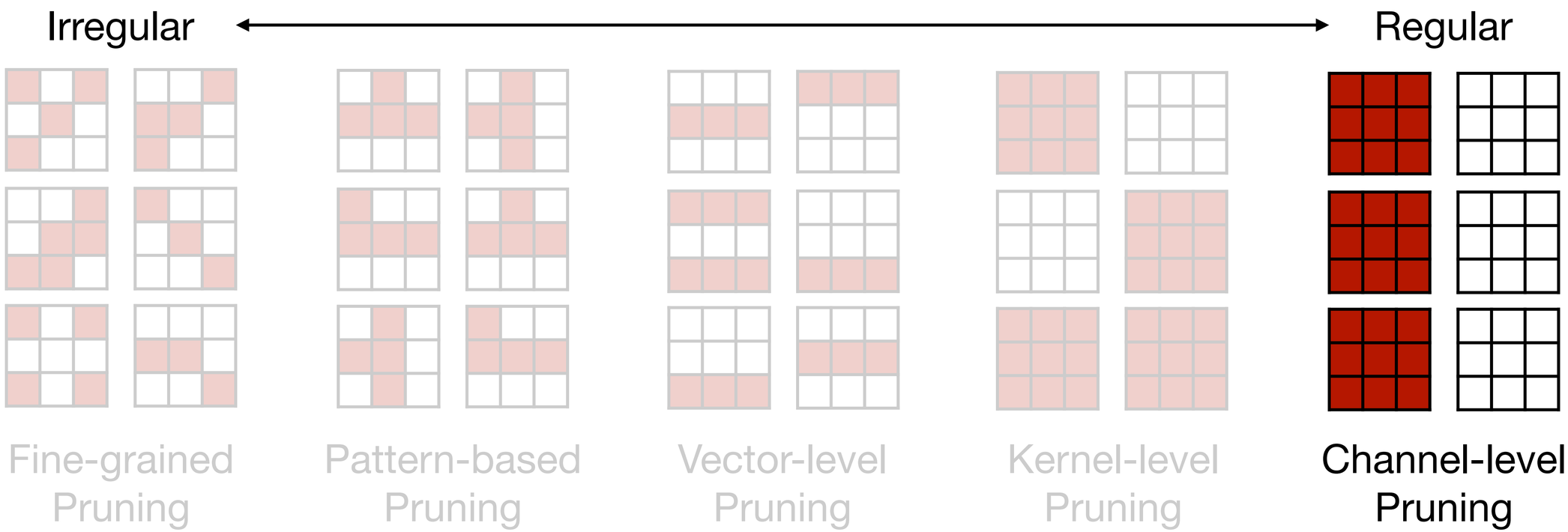
Pruned



Exploring the granularity of sparsity in convolutional neural networks [Mao *et al.*, CVPR-W]

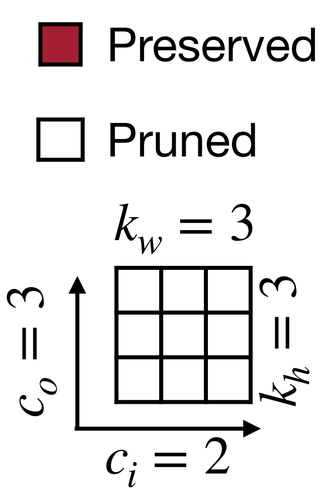
Recall: Different Pruning Granularity

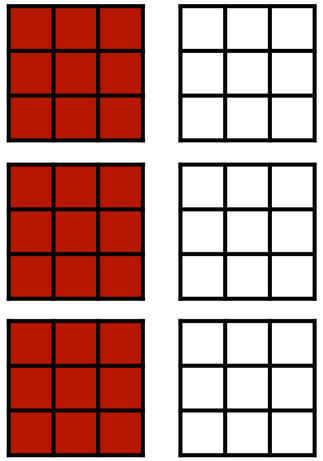




Exploring the granularity of sparsity in convolutional neural networks [Mao *et al.*, CVPR-W]

Channel-level Pruning





Channel-level Pruning

```
layer = nn.Conv2d(
  64,
  64,
  kernel_size=3,
  padding=1,
  stride=1
)
```

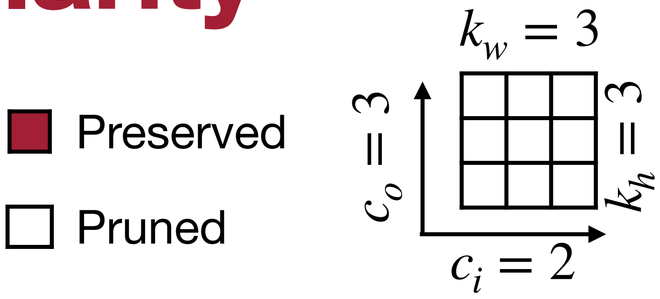
Original Implementation

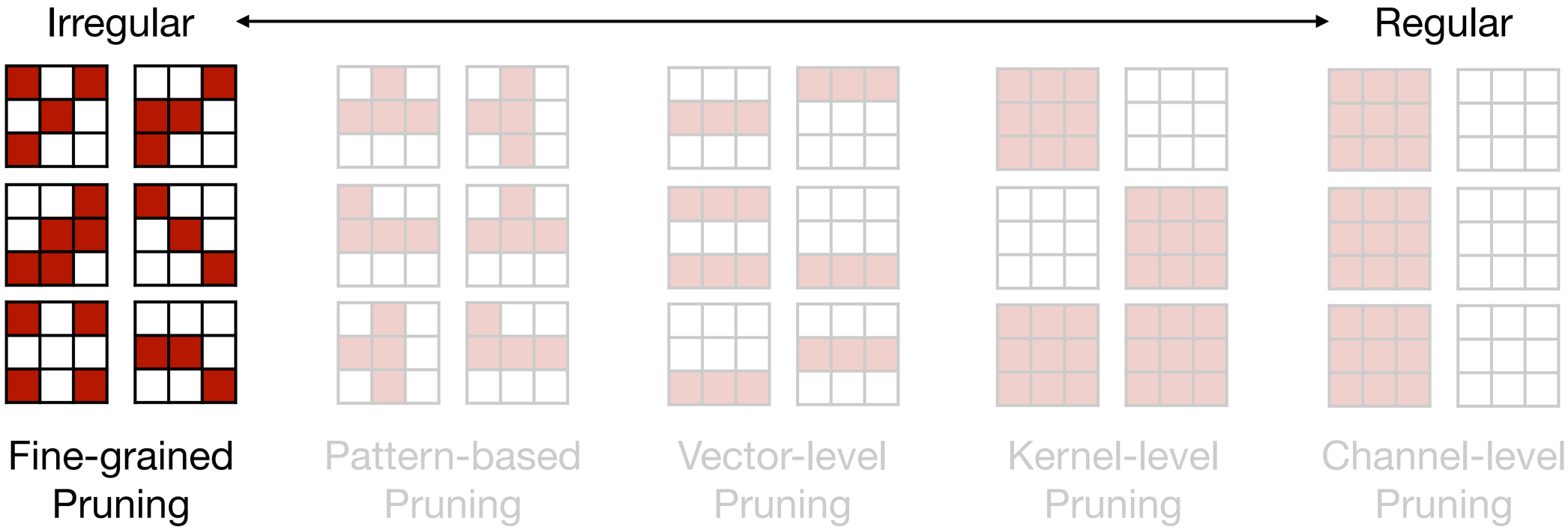
```
layer = nn.Conv2d(
  64,
  32,
  kernel_size=3,
  padding=1,
  stride=1
)
```

Pruned

No need for specialized system support!

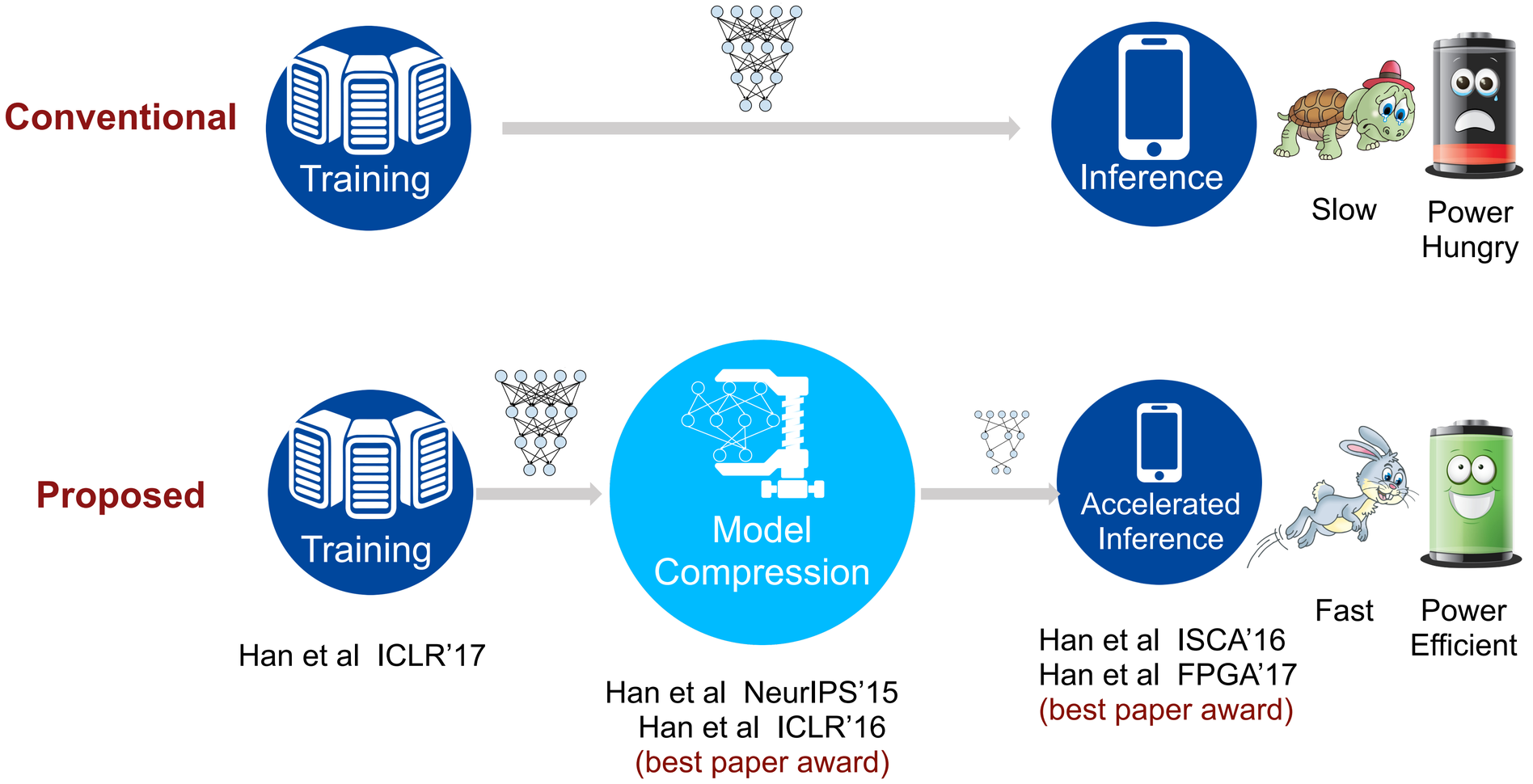
Recall: Different Pruning Granularity





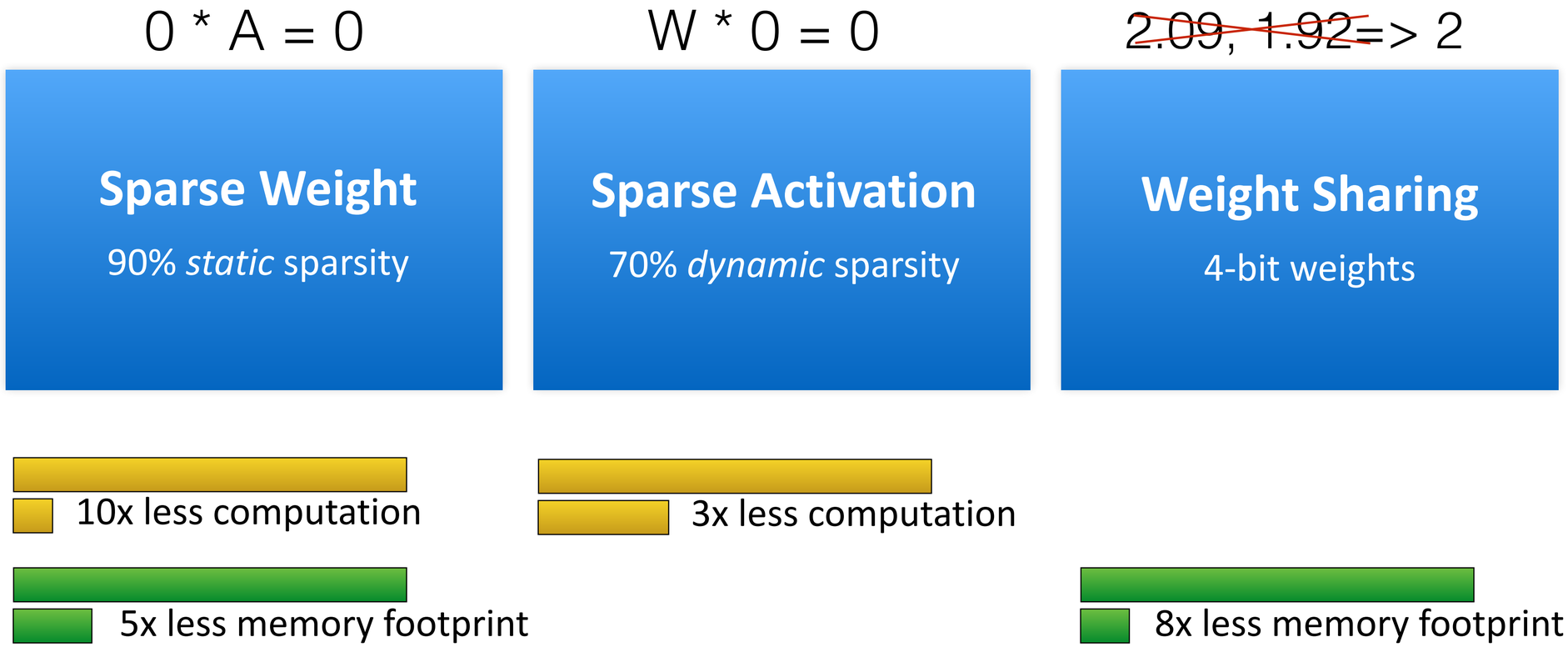
Exploring the granularity of sparsity in convolutional neural networks [Mao *et al.*, CVPR-W]

Proposed Paradigm



EIE: Efficient Inference Engine

The First DNN Accelerator for Sparse, Compressed Model



EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han *et al.,* ISCA 2016]

EIE: Parallelization on Sparsity

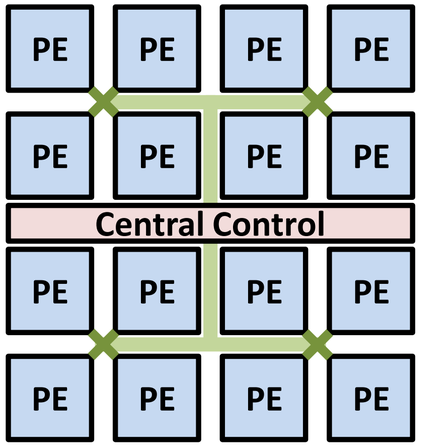
$$\vec{\boldsymbol{a}} \begin{pmatrix} 0 & \boldsymbol{a_1} & 0 & a_3 \end{pmatrix}$$

$$\times$$

$$\begin{pmatrix} w_{0,0} & \boldsymbol{w_{0,1}} & 0 & w_{0,3} \\ 0 & \mathbf{0} & w_{1,2} & 0 \\ 0 & \boldsymbol{w_{2,1}} & 0 & w_{2,3} \\ 0 & \mathbf{0} & 0 & 0 \\ 0 & \mathbf{0} & w_{4,2} & w_{4,3} \\ w_{5,0} & \mathbf{0} & 0 & 0 \\ 0 & \mathbf{0} & 0 & w_{6,3} \\ 0 & \boldsymbol{w_{7,1}} & 0 & 0 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ -b_2 \\ b_3 \\ -b_4 \\ b_5 \\ b_6 \\ -b_7 \end{pmatrix} \overset{ReLU}{\Rightarrow} \vec{\boldsymbol{b}} \begin{pmatrix} b_0 \\ b_1 \\ 0 \\ b_3 \\ 0 \\ b_5 \\ b_6 \\ 0 \end{pmatrix}$$

EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han *et al.,* ISCA 2016]

EIE: Parallelization on Sparsity



$$\vec{a} \begin{pmatrix} 0 & \boldsymbol{a_1} & 0 & a_3 \end{pmatrix}$$

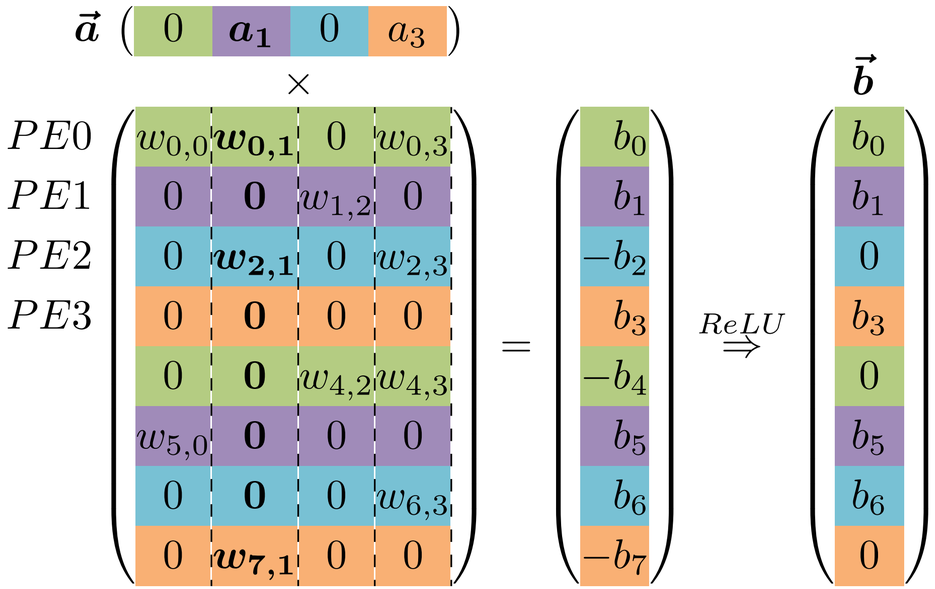
$$\times$$

$$\begin{array}{l} PE0 \\ PE1 \\ PE2 \\ PE3 \end{array} \begin{pmatrix} w_{0,0} & \boldsymbol{w_{0,1}} & 0 & w_{0,3} \\ 0 & \boldsymbol{0} & w_{1,2} & 0 \\ 0 & \boldsymbol{w_{2,1}} & 0 & w_{2,3} \\ 0 & \boldsymbol{0} & 0 & 0 \\ 0 & \boldsymbol{0} & w_{4,2} & w_{4,3} \\ w_{5,0} & \boldsymbol{0} & 0 & 0 \\ 0 & \boldsymbol{0} & 0 & w_{6,3} \\ 0 & \boldsymbol{w_{7,1}} & 0 & 0 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ -b_2 \\ b_3 \\ -b_4 \\ b_5 \\ b_6 \\ -b_7 \end{pmatrix} \overset{ReLU}{\Rightarrow} \vec{b} \begin{pmatrix} b_0 \\ b_1 \\ 0 \\ b_3 \\ 0 \\ b_5 \\ b_6 \\ 0 \end{pmatrix}$$

EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han *et al.,* ISCA 2016]

EIE: Parallelization on Sparsity

logically

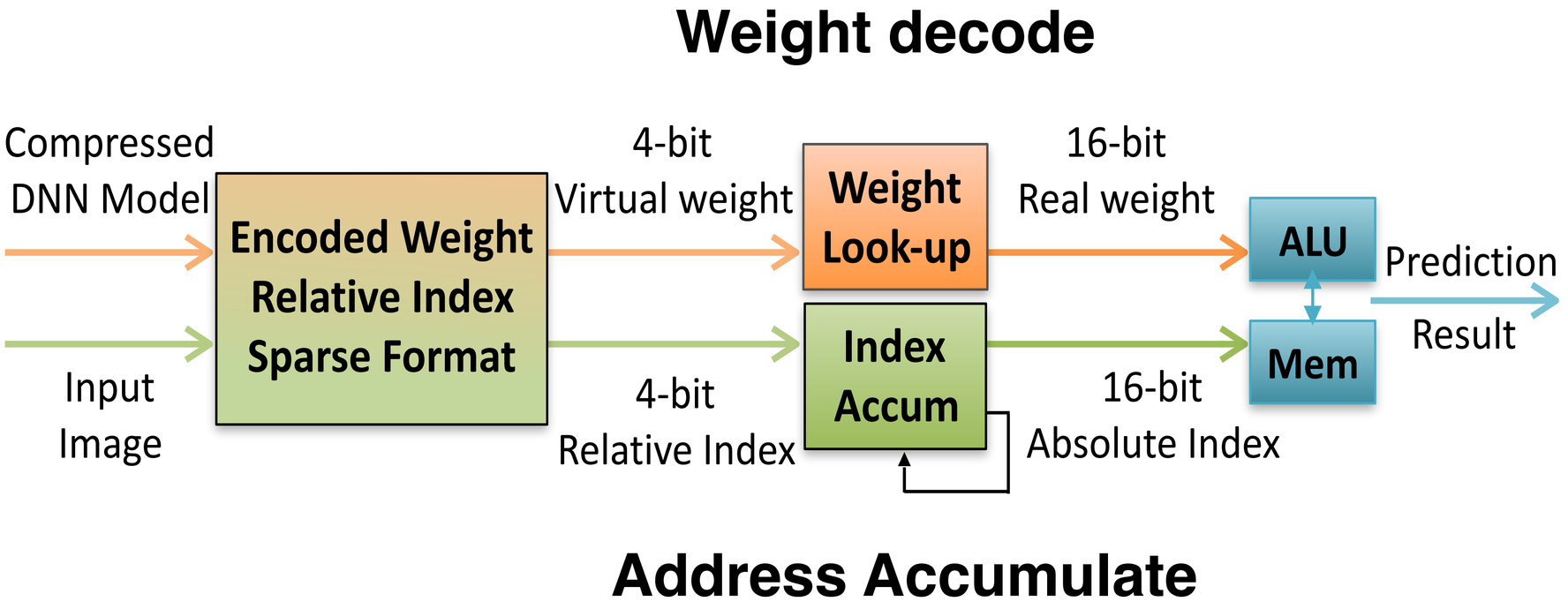


physically

Virtual Weight	$W_{0,0}$	$W_{0,1}$	$W_{4,2}$	$W_{0,3}$	$W_{4,3}$
Relative Index	0	1	2	0	0
Column Pointer	0	1	2	3	5

EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han *et al.,* ISCA 2016]

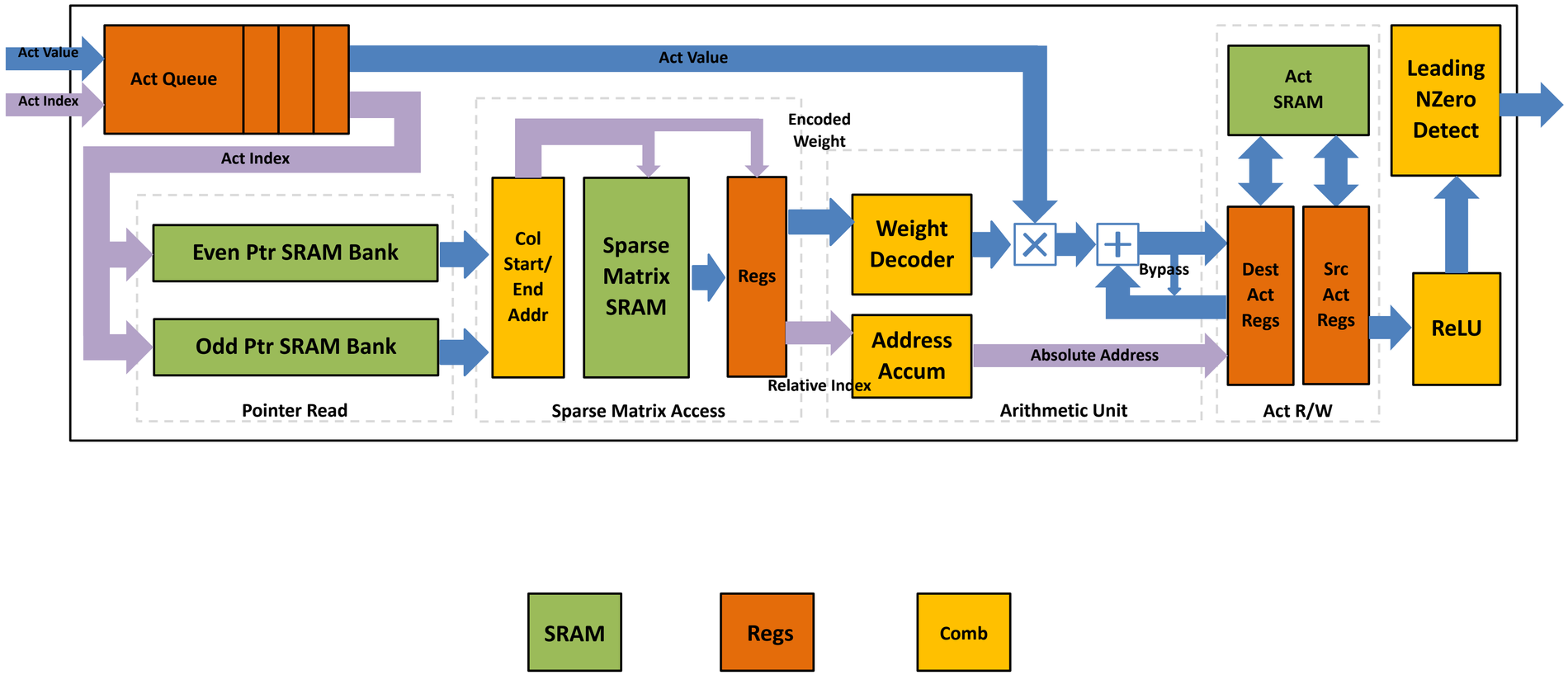
EIE Architecture



rule of thumb: 0 * A = 0 W * 0 = 0 ~~2.09, 1.92~~=> 2

EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han *et al.,* ISCA 2016]

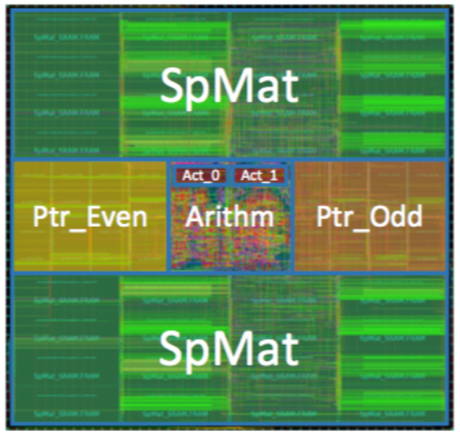
Micro Architecture for each PE



EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han *et al.,* ISCA 2016]

Post Layout Result of EIE

- ALU width: with accuracy
  - **32bit float**: no loss
  - **32 bit Int**: 0.3% loss
  - **16 bit Int**: 0.5% loss
  - **8 bit Int**: 27% loss
- FIFO queue depth
- number of PEs



Technology	40 nm
# PEs	64
on-chip SRAM	8 MB
Max Model Size	84 Million
Static Sparsity	10x
Dynamic Sparsity	3x
Quantization	4-bit
ALU Width	16-bit
Area	40.8 mm^2
MxV Throughput	81,967 layers/s
Power	586 mW

1. **Post layout result**
2. **Throughput measured on AlexNet FC-7**

EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han *et al.,* ISCA 2016]

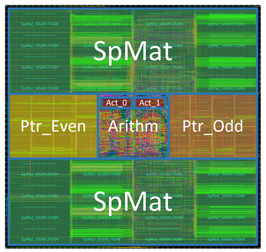
Benchmark

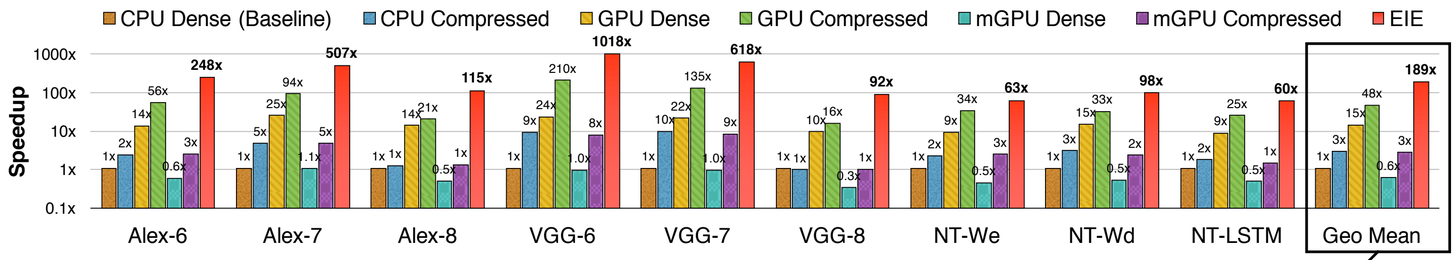
- CPU: Intel Core-i7 5930k
- GPU: NVIDIA TitanX
- Mobile GPU: NVIDIA Jetson TK1

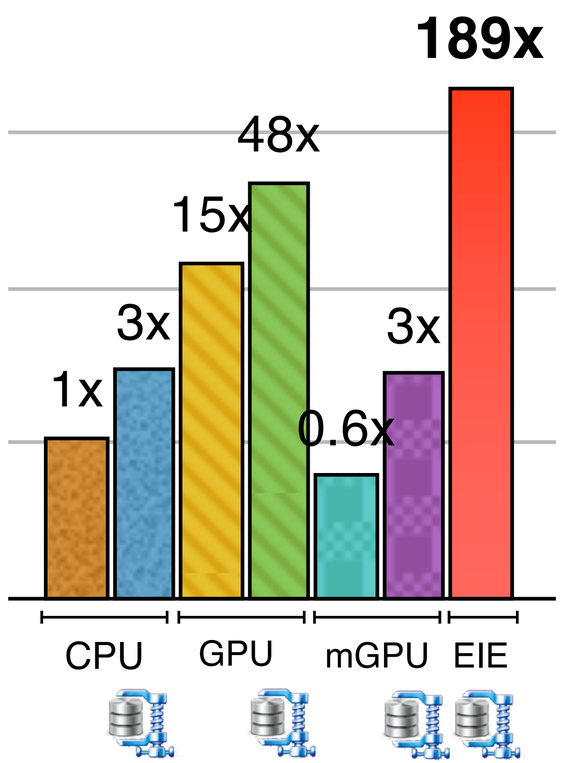
Layer	Size	Weight Density	Activation Density	FLOP Reduction	Description
AlexNet-6	4096 × 9216	9%	35%	**33x**	AlexNet for image classification
AlexNet-7	4096 × 4096	9%	35%	**33x**	
AlexNet-8	1000 × 4096	25%	38%	**10x**	
VGG-6	4096 × 25088	4%	18%	**100x**	VGG-16 for image classification
VGG-7	4096 × 4096	4%	37%	**50x**	
VGG-8	1000 × 4096	23%	41%	**10x**	
NeuralTalk-We	600 × 4096	10%	100%	**10x**	RNN and LSTM for image caption
NeuralTalk-Wd	8791 × 600	11%	100%	**10x**	
NeuralTalk-LSTM	2400 × 1201	10%	100%	**10x**	

EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han *et al.,* ISCA 2016]

Speedup on EIE



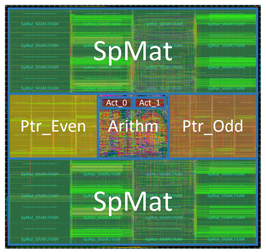


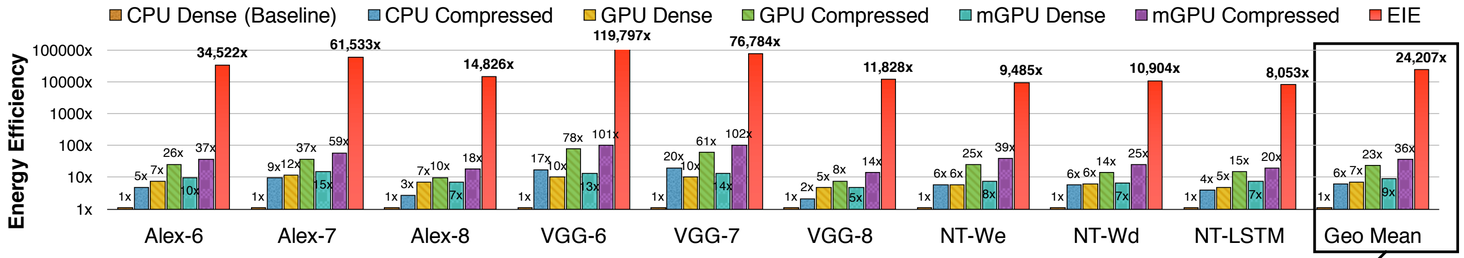


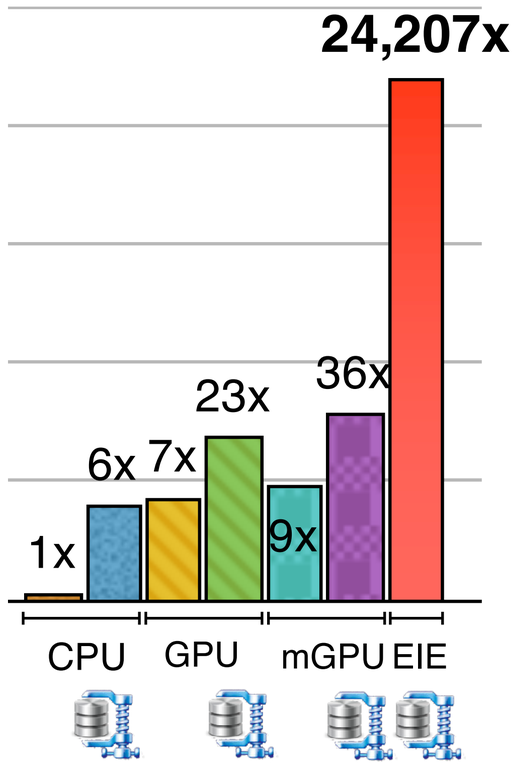
Geo Mean

EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han *et al.,* ISCA 2016]

Energy Efficiency on EIE



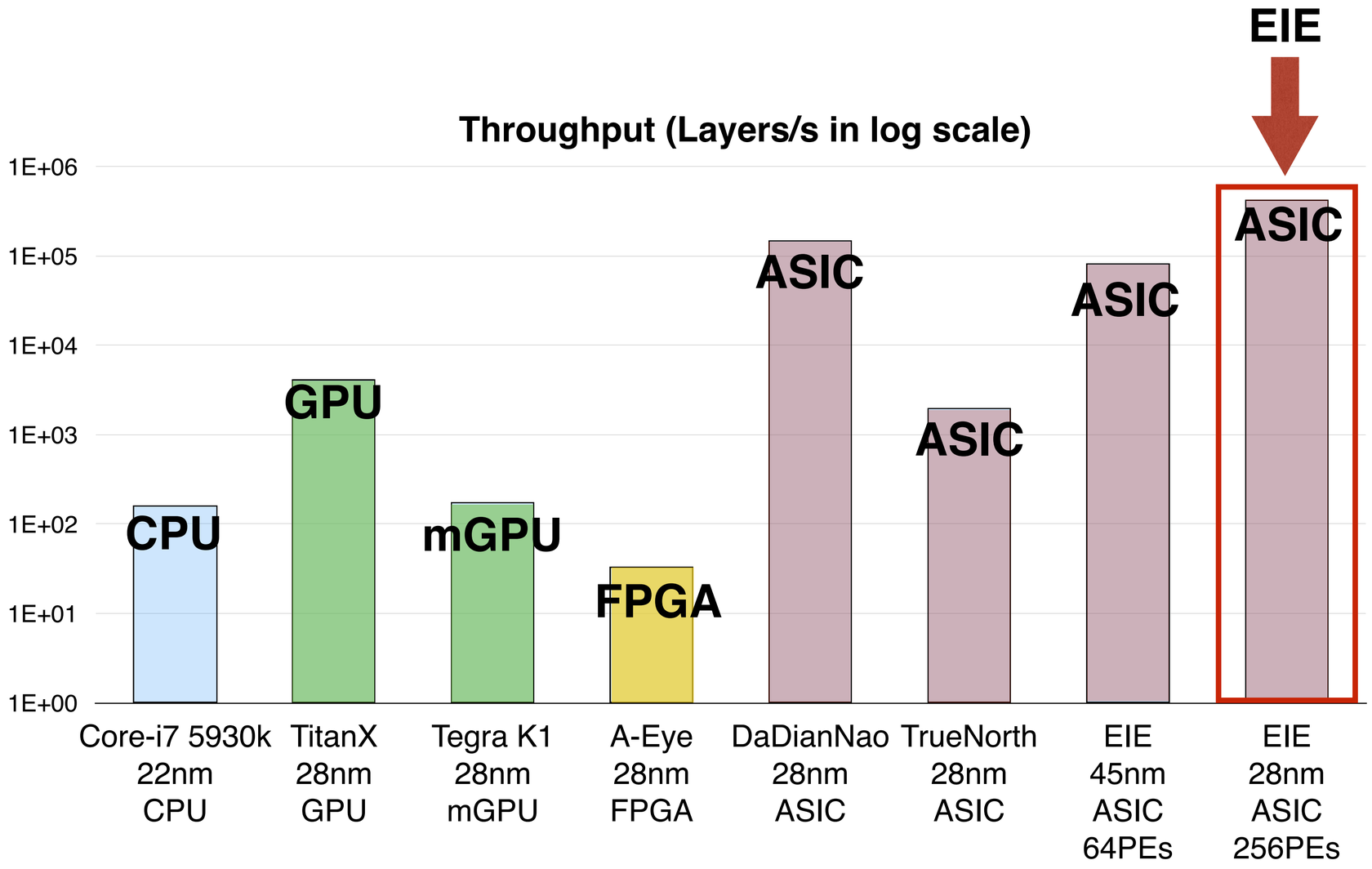




Geo Mean

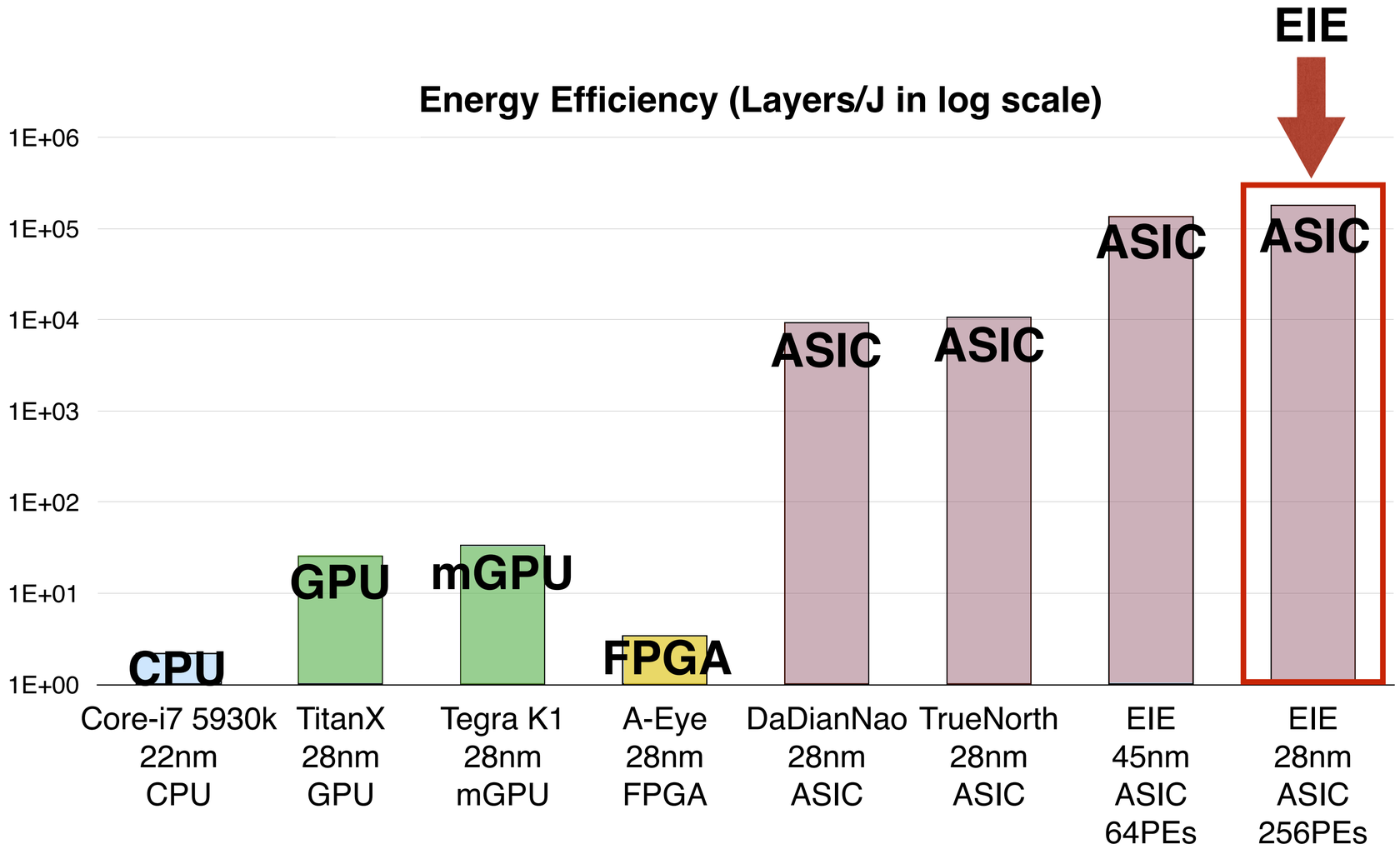
EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han *et al.,* ISCA 2016]

Comparison: Throughput



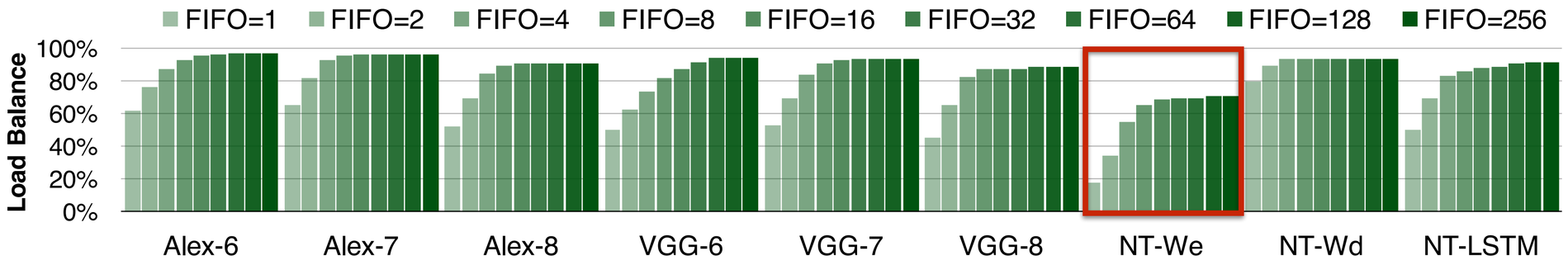
EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han *et al.,* ISCA 2016]

Comparison: Energy Efficiency



EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han *et al.,* ISCA 2016]

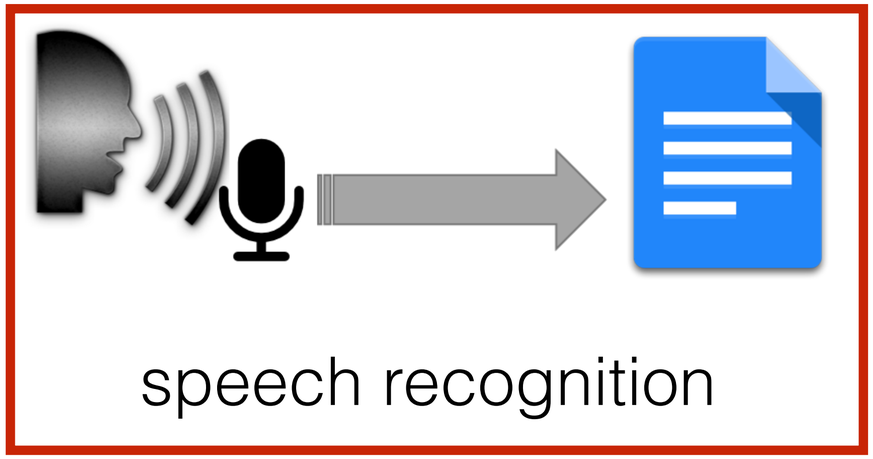
Load Balancing and Utilization



- Imbalanced non-zeros among PEs degrades system utilization.
- This load imbalance could be solved by FIFO.
- With FIFO depth=8, ALU utilization is > 80%.

EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han *et al.,* ISCA 2016]

Accelerating Recurrent Neural Networks



speech recognition

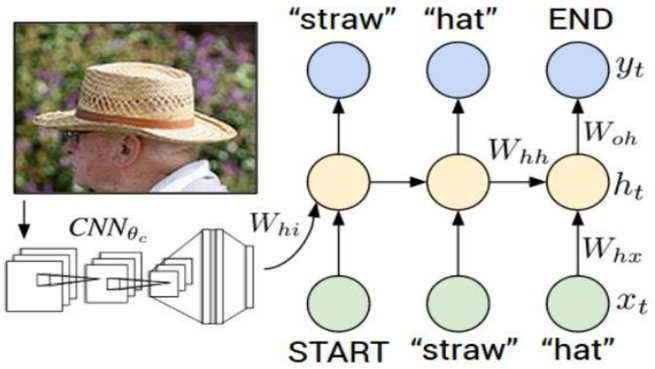
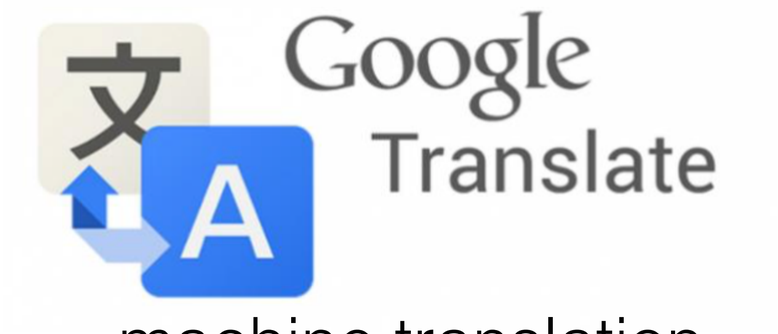
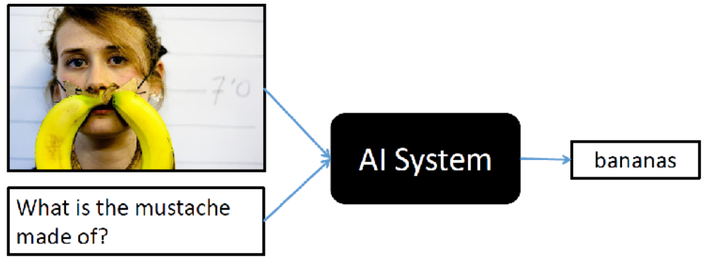


image caption



machine translation

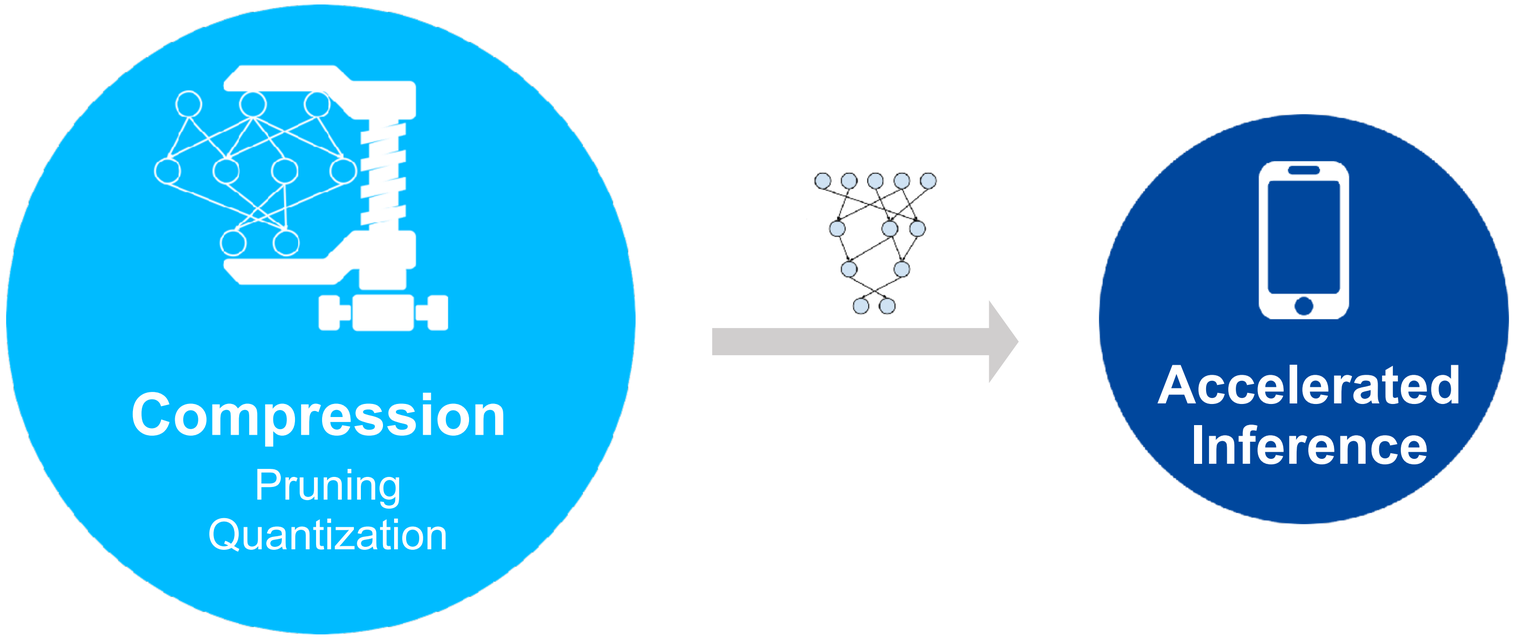


visual question answering

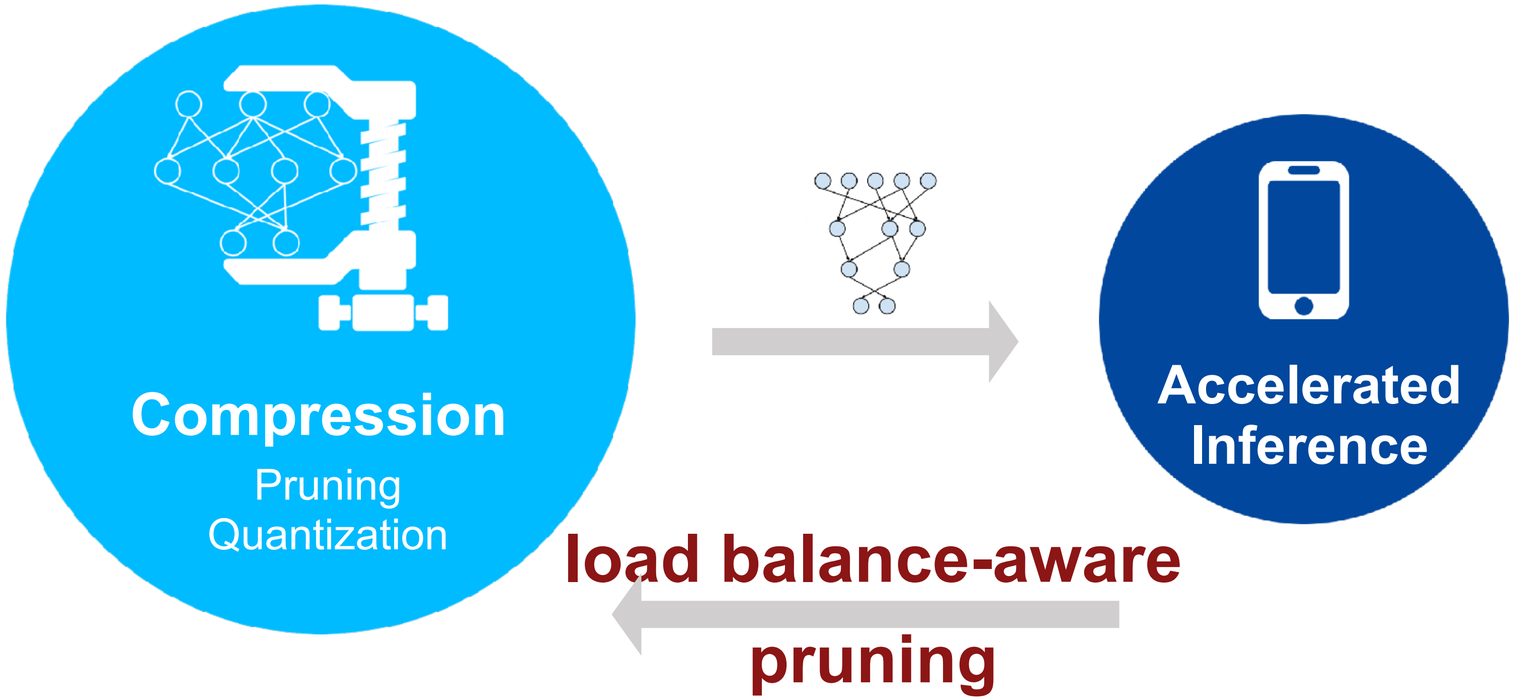
"If people use Google voice search for just **three minutes** a day and we ran deep neural nets for our **speech recognition** system on the processing units we were using, we would have had to **double the number of Google data centers!**"

—— Norm Jouppi

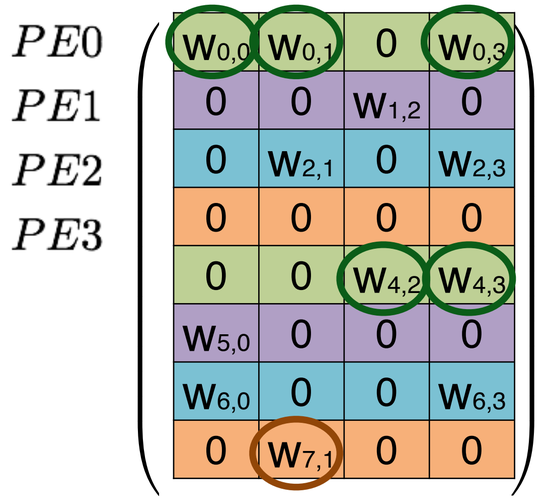
The Design Flow

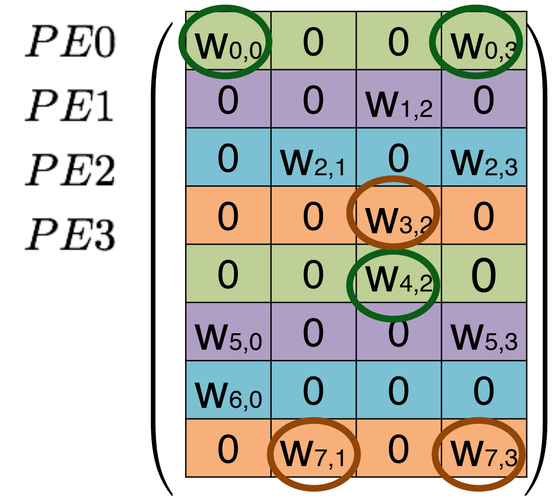


Rethink the Design Flow

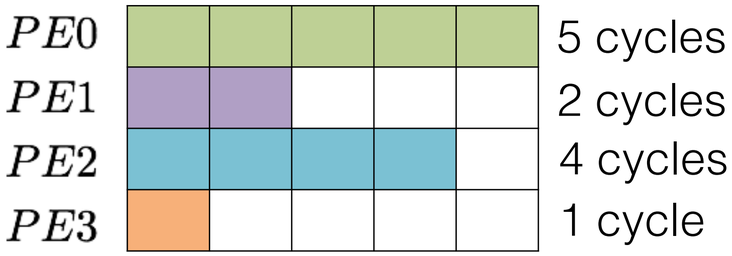


Load Balance Aware Pruning

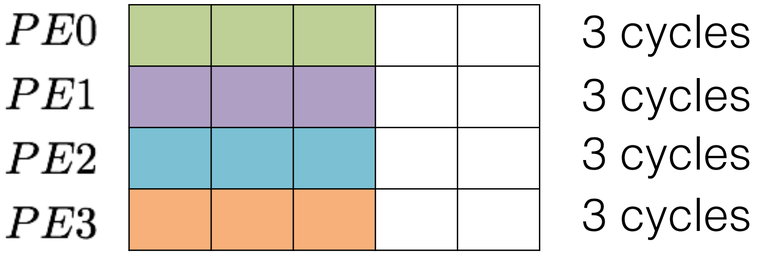




Unbalanced

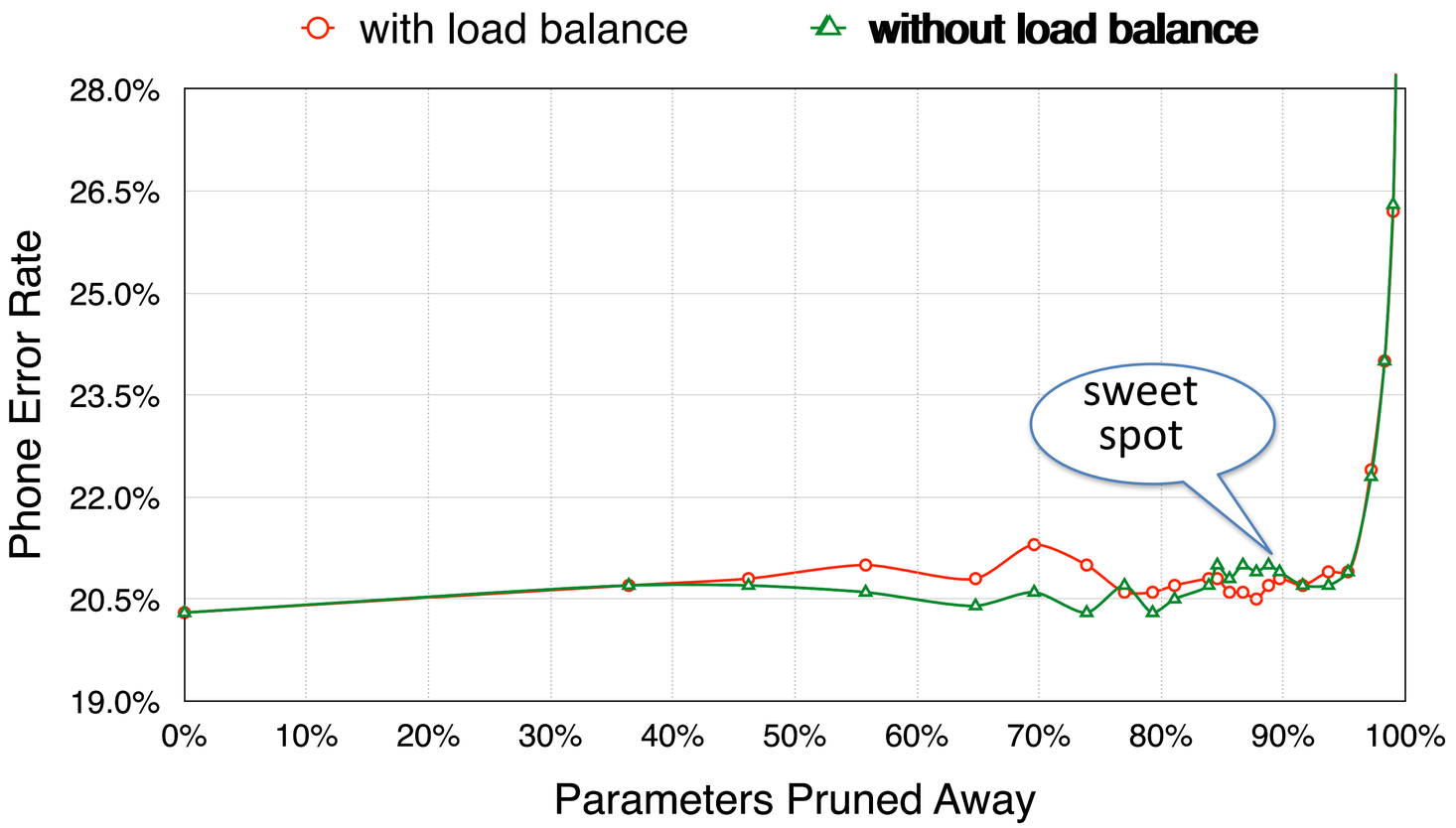


Balanced



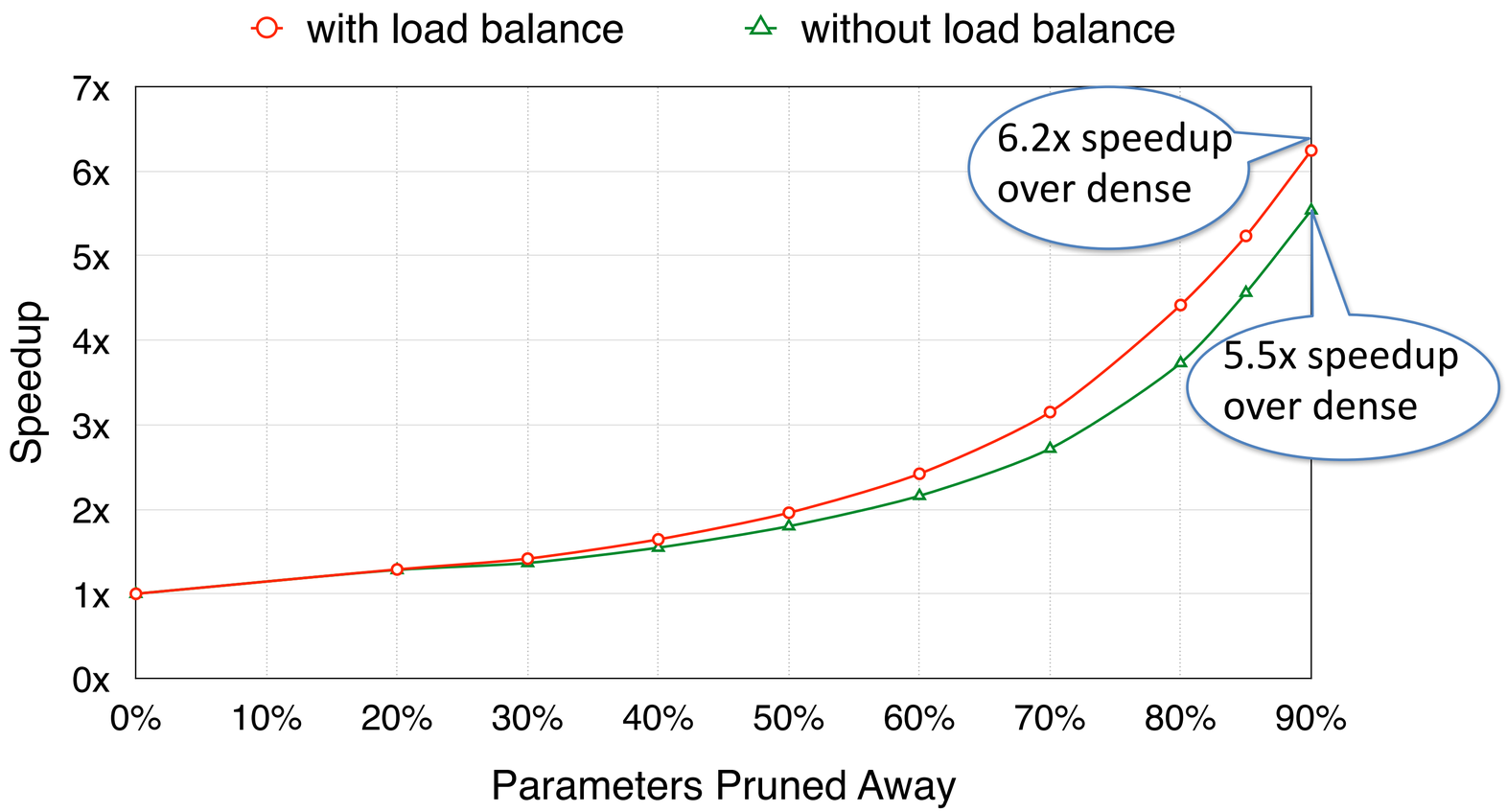
ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA [Han *et al.*, FPGA 2017]

Load Balance Aware Pruning: Same Accuracy



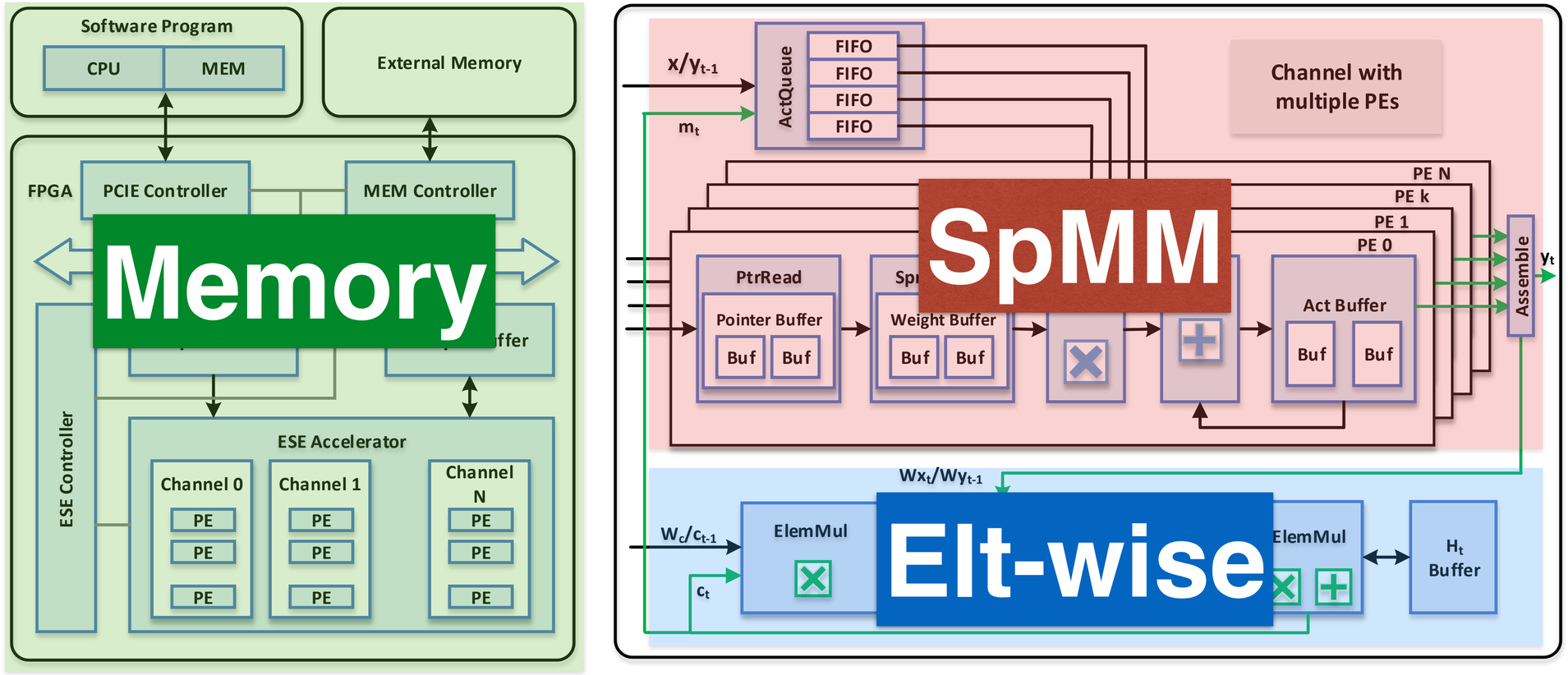
ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA [Han *et al.*, FPGA 2017]

Load Balance Aware Pruning: Better Speedup



ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA [Han *et al.*, FPGA 2017]

Hardware Architecture



ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA [Han *et al.*, FPGA 2017]

Speedup and Energy Efficiency

Network	GPU		CPU		ESE
	Dense	Sparse	Dense	Sparse	Sparse
Latency	240us	287us	6017us	3569us	82.7us
Power	202W	136W	111W	38W	41W
Speedup	1x	0.84x	0.04x	0.07x	3x
Energy Efficiency	1x	1.25x	0.07x	0.36x	14x

	LUT	LUTRAM[1]	FF	BRAM[1]	DSP
Avail.	331,680	146,880	663,360	1,080	2,760
Used	293,920	69,939	453,068	947	1,504
Utili.	88.6%	47.6%	68.3%	87.7%	54.5%

Resource Utilization on Xilinx KU060 FPGA @200MHz

ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA [Han *et al.*, FPGA 2017]

Speedup and Energy Efficiency

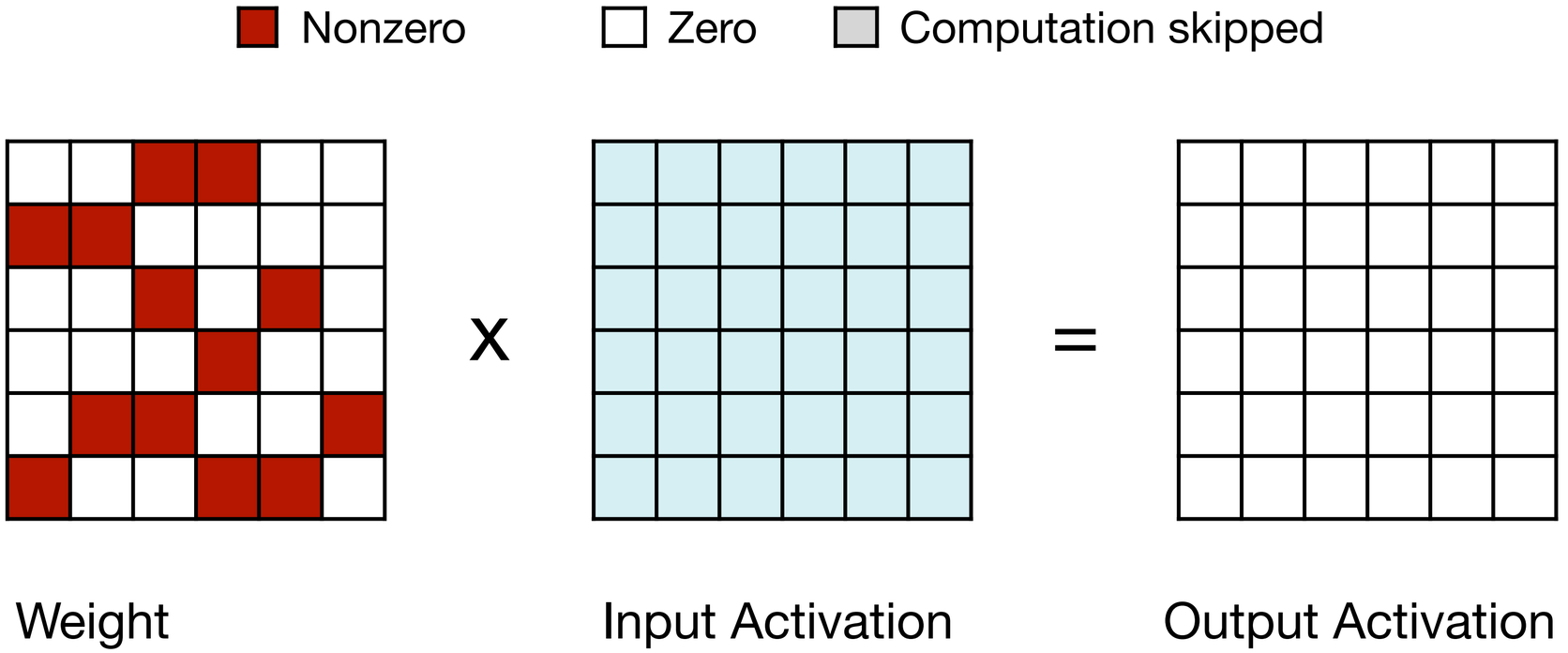
Platforms	GPU (Pascal TitanX)	CPU (Core i7-5930k)	ESE (Xilinx KU060)
Latency	240us	6017us	**82.7us**
Power	202W	111W	**41W**
Speedup	1x	0.04x	**3x**
Energy Efficiency	1x	0.07x	**14x**

	LUT	LUTRAM[1]	FF	BRAM[1]	DSP
Avail.	331,680	146,880	663,360	1,080	2,760
Used	293,920	69,939	453,068	947	1,504
Utili.	88.6%	47.6%	68.3%	87.7%	54.5%

Resource Utilization on Xilinx KU060 FPGA @200MHz

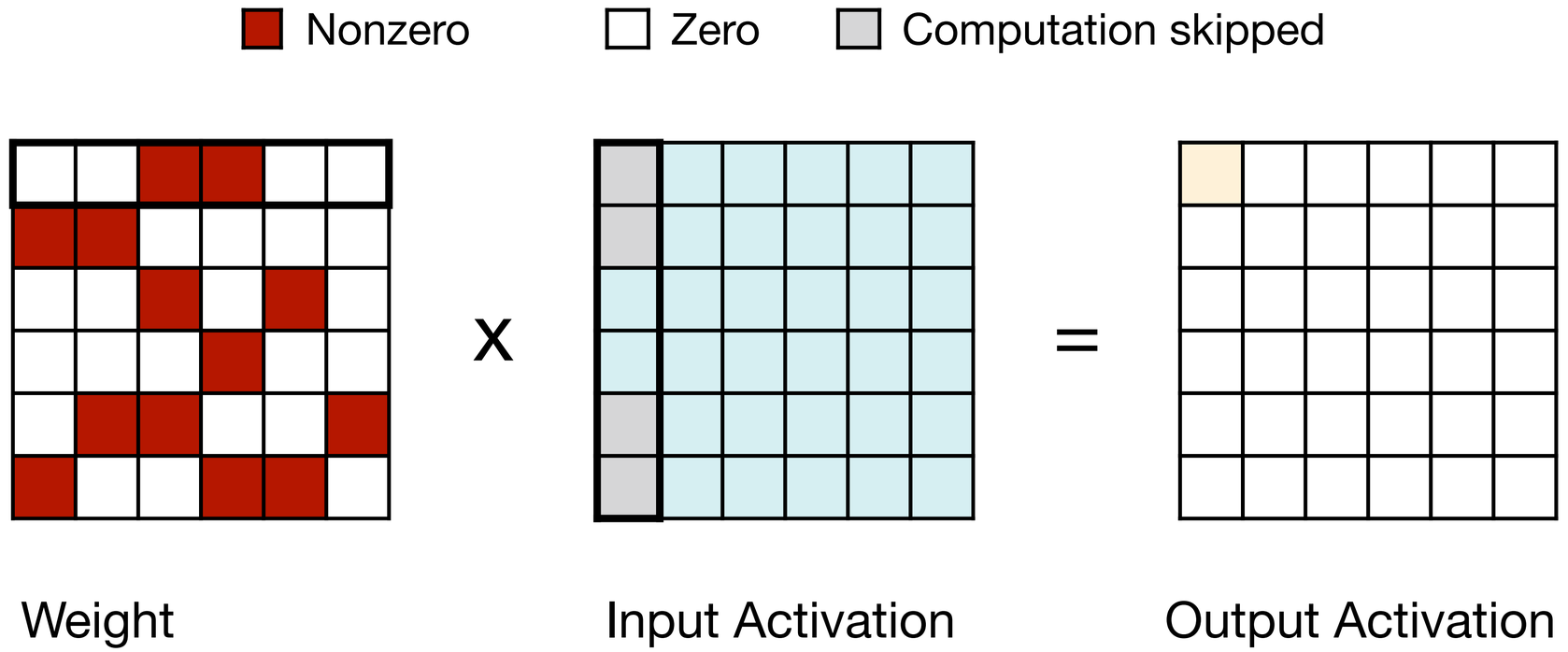
ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA [Han *et al.*, FPGA 2017]

Fine-grained Pruning



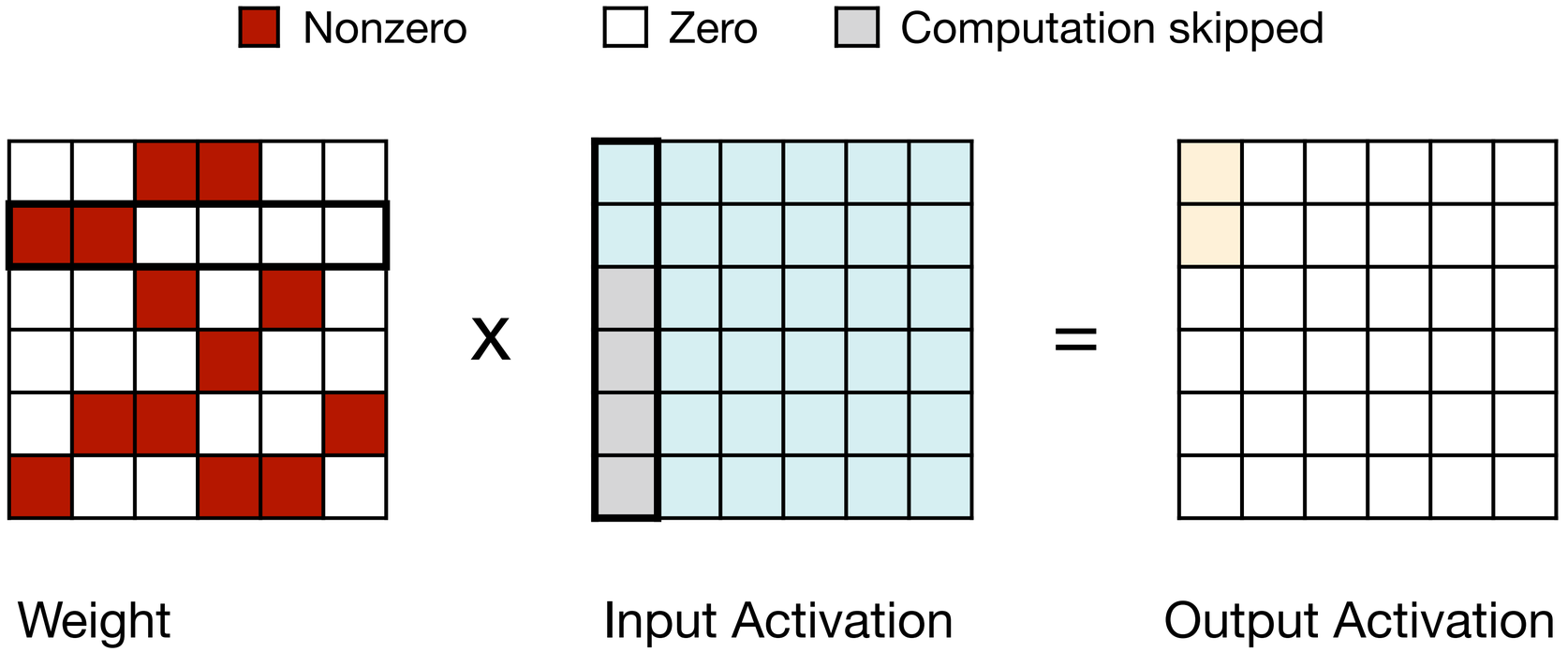
Zero x Any computation can be skipped!

Fine-grained Pruning



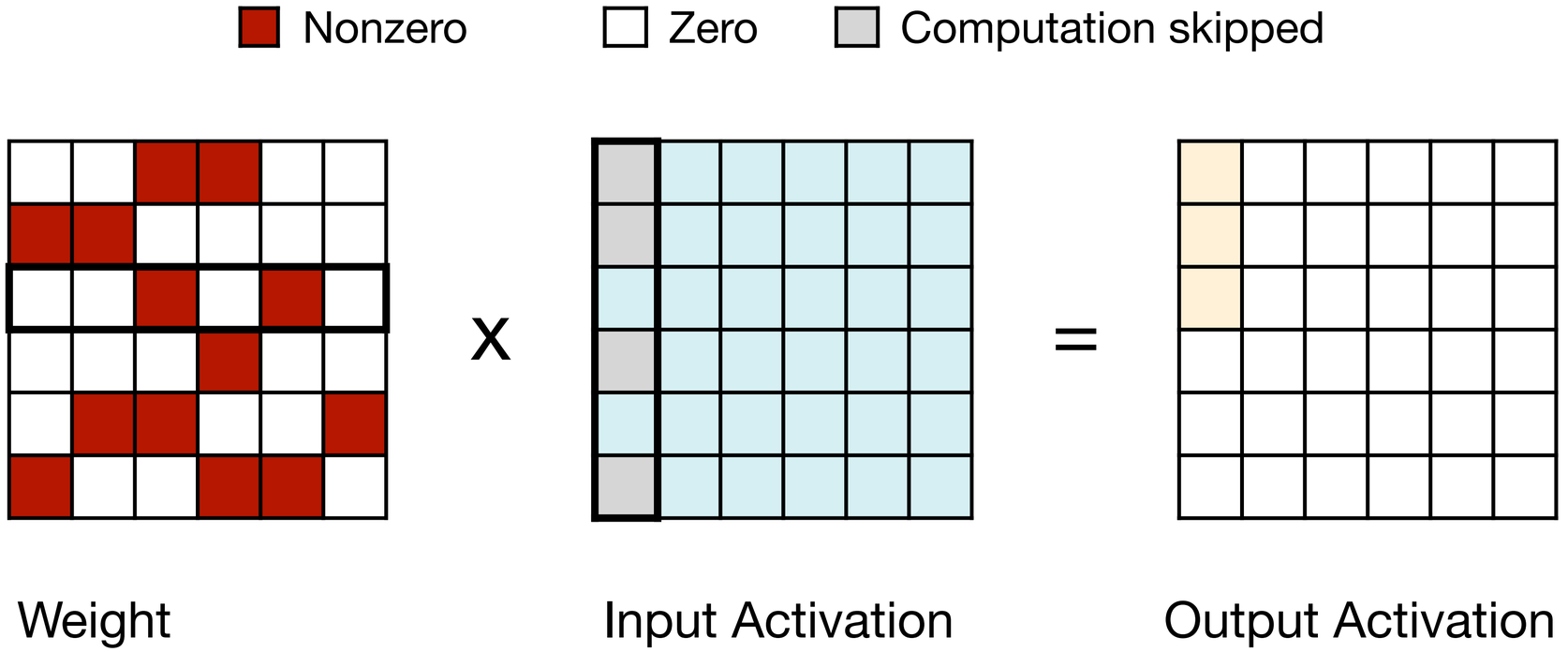
Zero x Any computation can be skipped!

Fine-grained Pruning



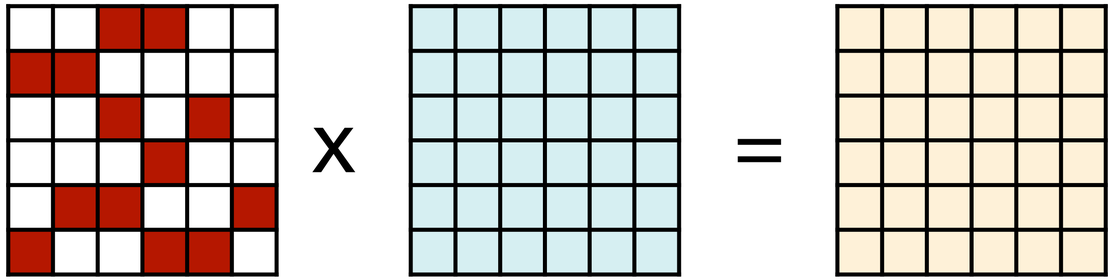
Zero x Any computation can be skipped!

Fine-grained Pruning



Zero x Any computation can be skipped!

Sparse Matrix-Matrix Multiplication (SpMM)




```
for m in range(M):
  for n in range(N):
    for k in range(A[m].size):
      C[m][n] = A[m][k] * B[k][n]
```

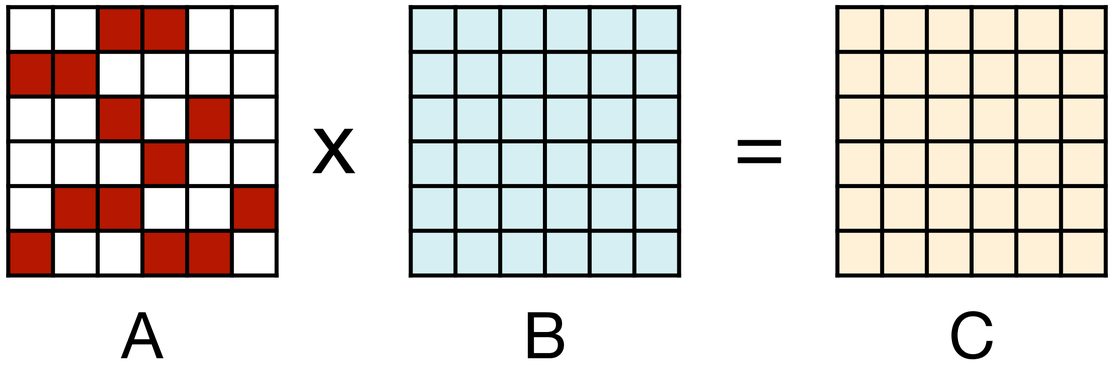
Dense Matrix Multiplication

```
for m in range(M):
  for n in range(N):
    for k in A[m].nonzeros:
      C[m][n] = A[m][k] * B[k][n]

    A[0].nonzeros = [2, 3]
    A[1].nonzeros = [0, 1]
    …
    A[5].nonzeros = [0, 3, 4]
```

Sparse Matrix Multiplication

Sparse Matrix-Matrix Multiplication (SpMM)




```
for m in range(M):
  for n in range(N):
    for k in range(A[m].size):
      C[m][n] = A[m][k] * B[k][n]
```

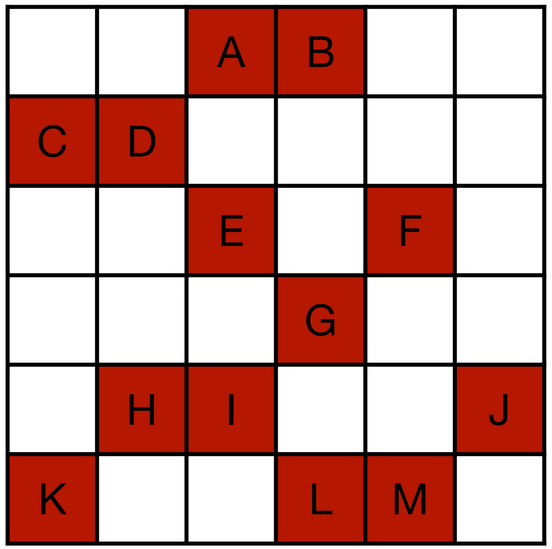
Dense Matrix Multiplication

```
for m in range(M):
  for n in range(N):
    for k in A[m].nonzeros:
      C[m][n] = A[m][k] * B[k][n]

    A[0].nonzeros = [2, 3]
    A[1].nonzeros = [0, 1]
    …
    A[5].nonzeros = [0, 3, 4]
```

Sparse Matrix Multiplication

CSR Format for Sparse Matrices



0	2	4	6	7	10	13

Row Pointer

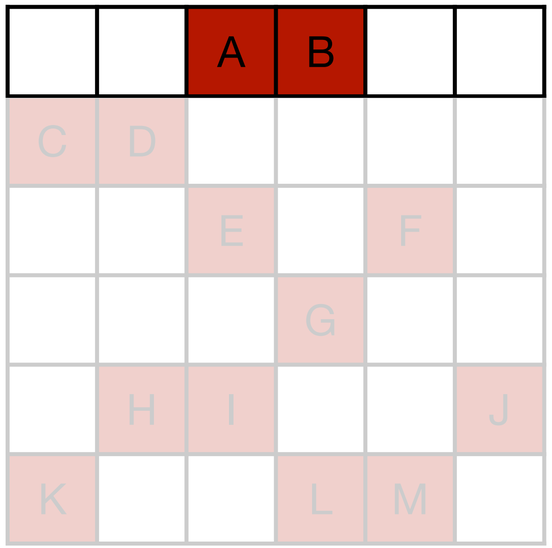
2	3	0	1	2	4	3	…

Column Indices

A	B	C	D	E	F	G	…

Values

CSR Format for Sparse Matrices



A[0].nonzeros = [2, 3]

Because there is no element before the current row.

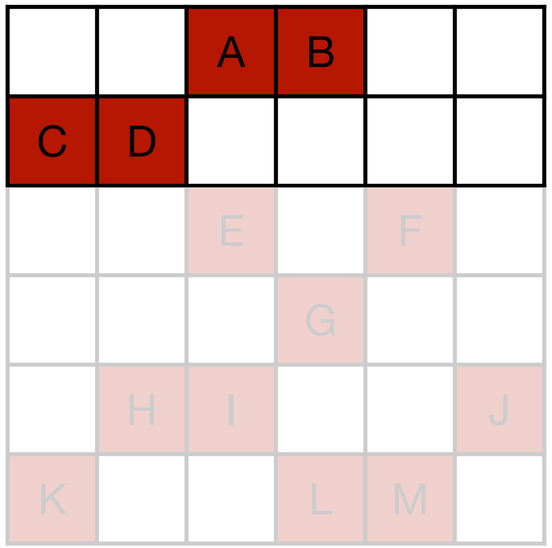
Row Pointer

Column Indices

Values

Nonzero values for row 0

CSR Format for Sparse Matrices




```
A[1].nonzeros
= [0, 1]
```

Because there are two elements before the current row.

0	2	4	6	7	10	13

Row Pointer

2	3	0	1	2	4	3	…

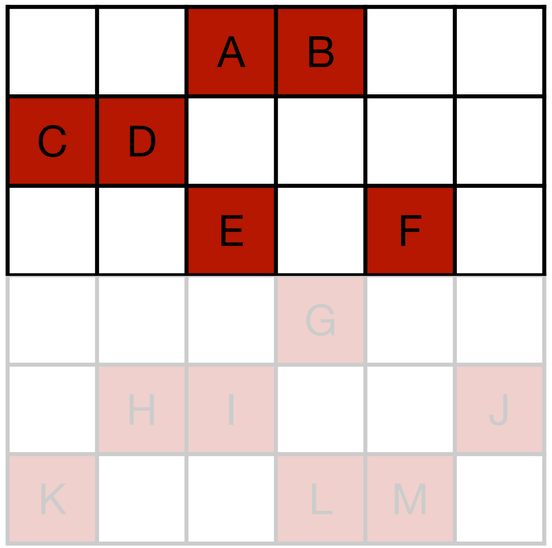
Column Indices

A	B	C	D	E	F	G	…

Values

Nonzero values for row 1

CSR Format for Sparse Matrices



Because there are four elements before the current row.

0	2	4	6	7	10	13

Row Pointer

2	3	0	1	2	4	3	…

Column Indices

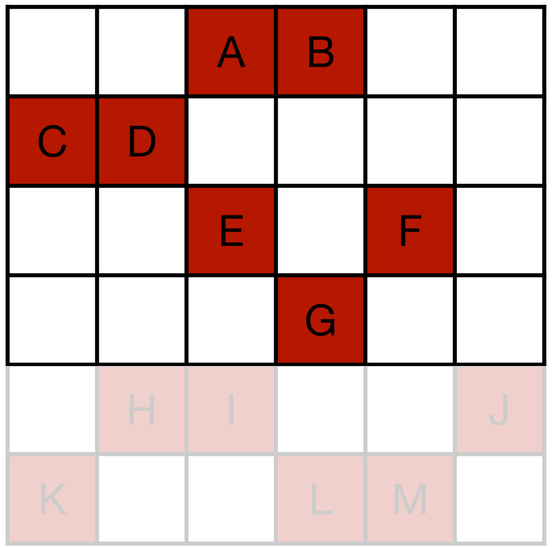
`A[2].nonzeros = [2, 4]`

A	B	C	D	E	F	G	…

Values

Nonzero values for row 2

CSR Format for Sparse Matrices



Because there are six elements before the current row.

0	2	4	6	7	10	13

Row Pointer

2	3	0	1	2	4	3	…

Column Indices

```
A[3].nonzeros
= [3]
```

A	B	C	D	E	F	G	…

Values

Nonzero values for row 3

GPU Support for SpMM

Deep Learning workloads typically have 40-90% sparsity

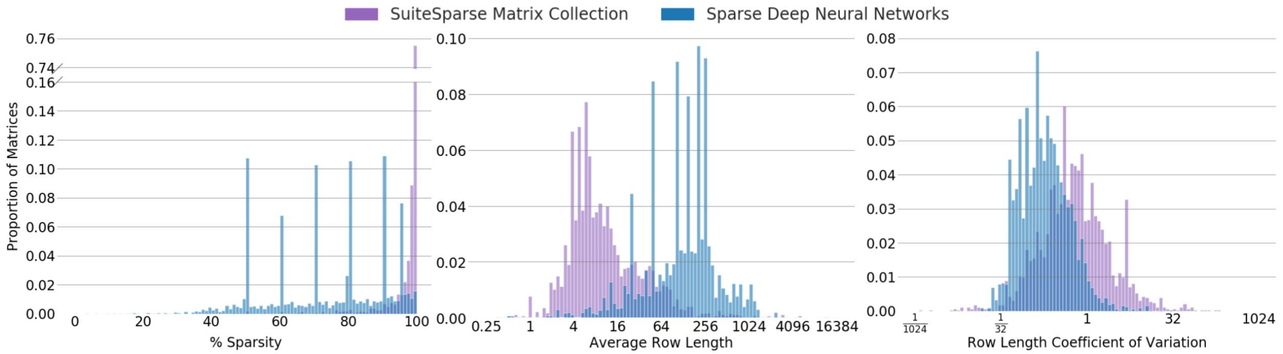


Fig. 2. **Properties of sparse matrices from scientific computing and deep learning applications.** Histograms are partially transparent to show overlapping regions. On average, deep learning matrices are 13.4× less sparse, have 2.3× longer rows, and have 25× less variation in row length within a matrix.

Deep learning workloads typically have lower sparsity than graphs. It is harder for libraries like cuSPARSE to accelerate them.

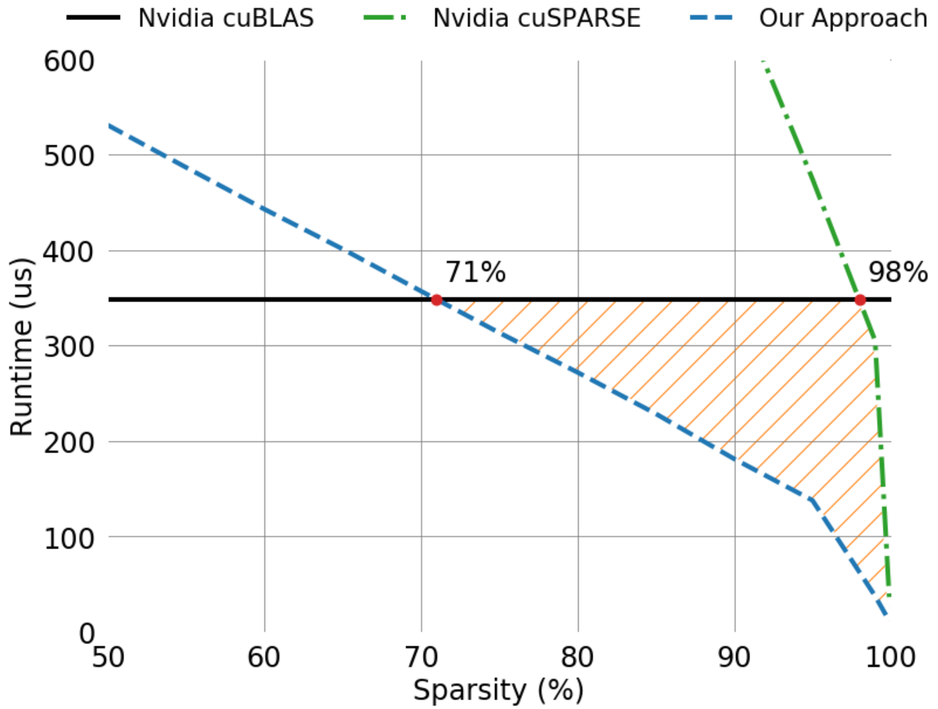
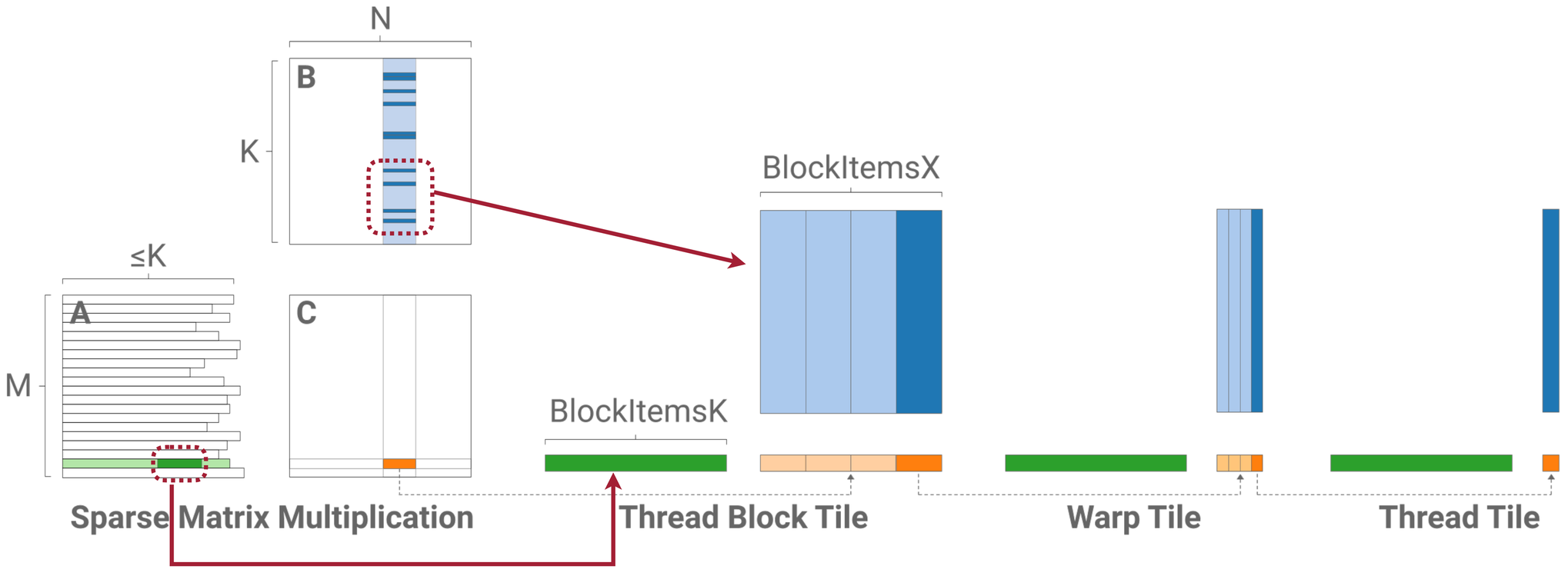


Fig. 1. **Sparse matrix–matrix multiplication runtime for a weight-sparse long short-term memory network problem**. Input size 8192, hidden size 2048, and batch size 128 in single-precision on an Nvidia V100 GPU with CUDA 10.1. Using our approach, sparse computation exceeds the performance of dense at as low as 71% sparsity. Existing vendor libraries require 14× fewer non-zeros to achieve the same performance. This work enables speedups for all problems in the highlighted region.

Sparse GPU Kernels for Deep Learning [Gale *et al.*, SC 2020]

GPU Support for SpMM

Hierarchical One-Dimensional Tiling

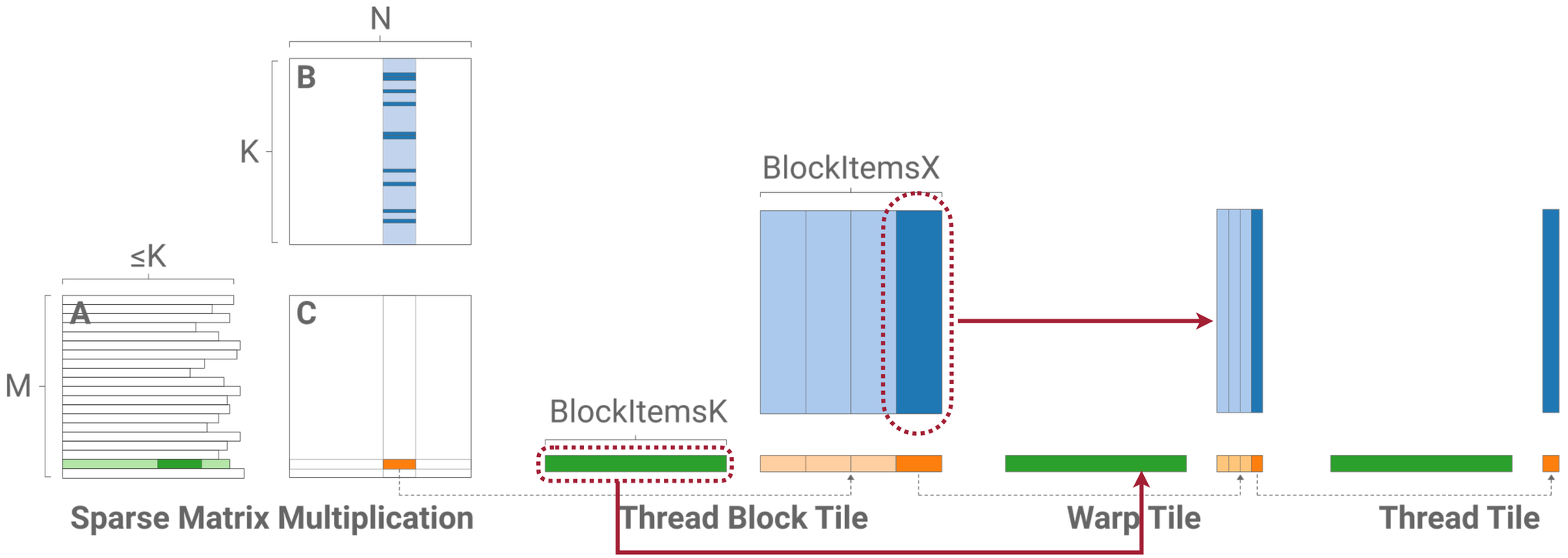


Dark color: calculated by the next level of tiling hierarchy.

Sparse GPU Kernels for Deep Learning [Gale *et al.*, SC 2020]

GPU Support for SpMM

Hierarchical One-Dimensional Tiling

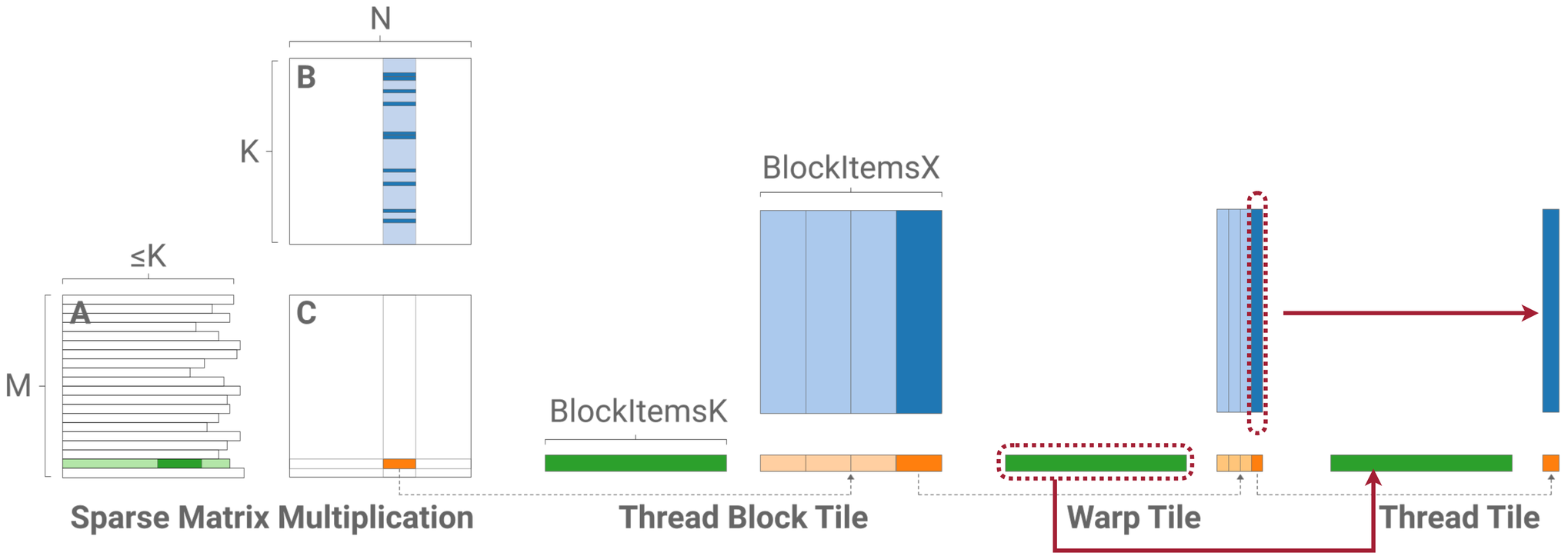


Dark color: calculated by the next level of tiling hierarchy.

Sparse GPU Kernels for Deep Learning [Gale *et al.*, SC 2020]

GPU Support for SpMM

Hierarchical One-Dimensional Tiling



Dark color: calculated by the next level of tiling hierarchy.

Sparse GPU Kernels for Deep Learning [Gale *et al.*, SC 2020]

GPU Support for SpMM

```
template <int kBlockItemsK, int kBlockItemsX>
__global__ void SpmmKernel(
  SparseMatrix a, Matrix b, Matrix c) {
  // Calculate tile indices.
  int m_idx = blockIdx.y;
  int n_idx = blockIdx.x * kBlockItemsX;

  // Calculate the row offset and the number
  // of nonzeros in this thread block's row.
  int m_off = a.row_offsets[m_idx];
  int nnz = a.row_offsets[m_idx+1] - m_off;

  // Main loop.
  Tile1D c_tile(/*init_to=*/0);
  for(; nnz > 0; nnz -= kBlockItemsK) {
    Tile1D a_tile = LoadTile(a);
    Tile2D b_tile = LoadTile(b);
    c_tile += a_tile * b_tile;
  }

  // Write output.
  StoreTile(c_tile, c);
}
```

Sparse GPU Kernels for Deep Learning [Gale *et al.*, SC 2020]

GPU Support for SpMM

Load balancing is crucial

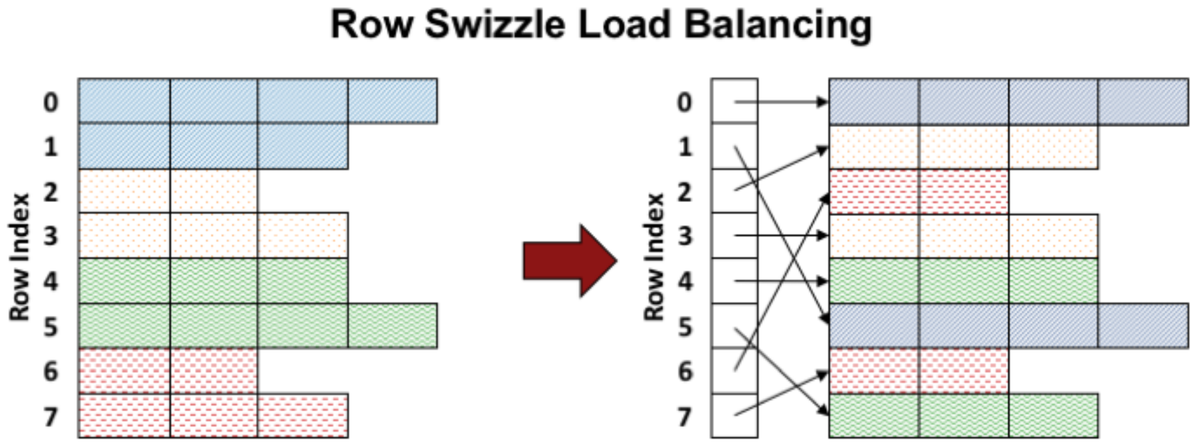
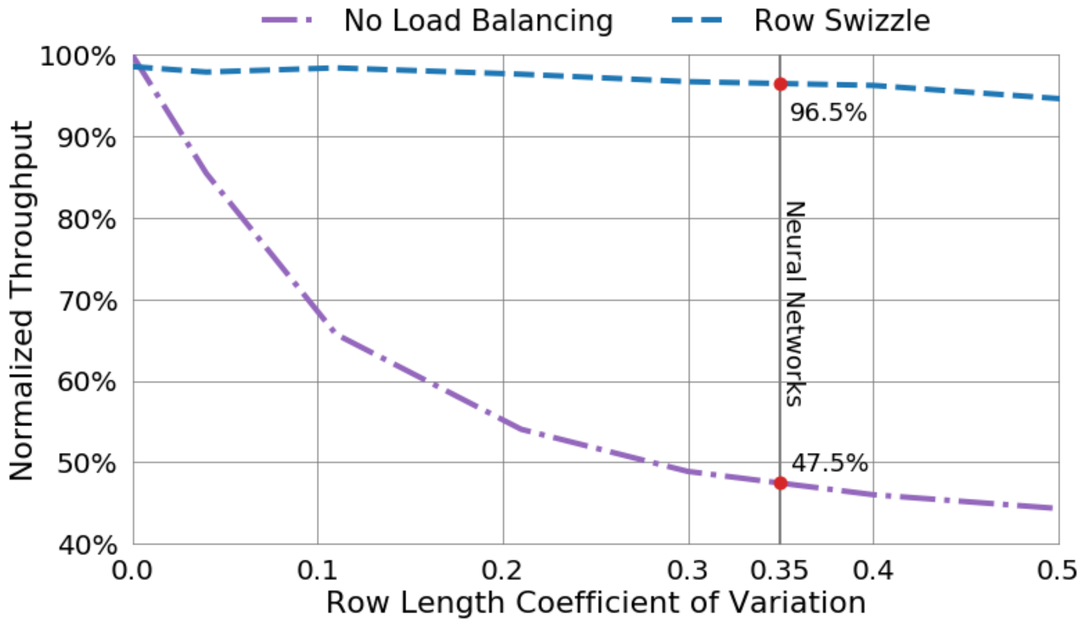


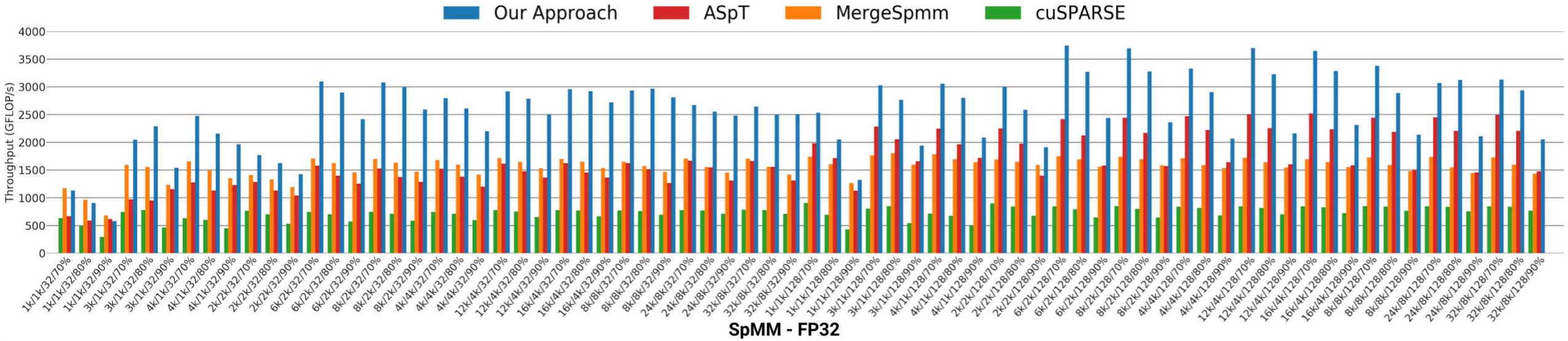
Fig. 6. **Row swizzle approach for load balancing sparse matrix computation**. Rows processed by the same warp are marked with the same pattern and color. We introduce a layer of indirection that re-orders when rows are processed. To balance work between threads in a warp, we group rows of similar length into *bundles*. To balance work between SMs, we process row bundles in decreasing order of size.



Load balancing: each thread in the warp (the same color) has **the same** workload size.

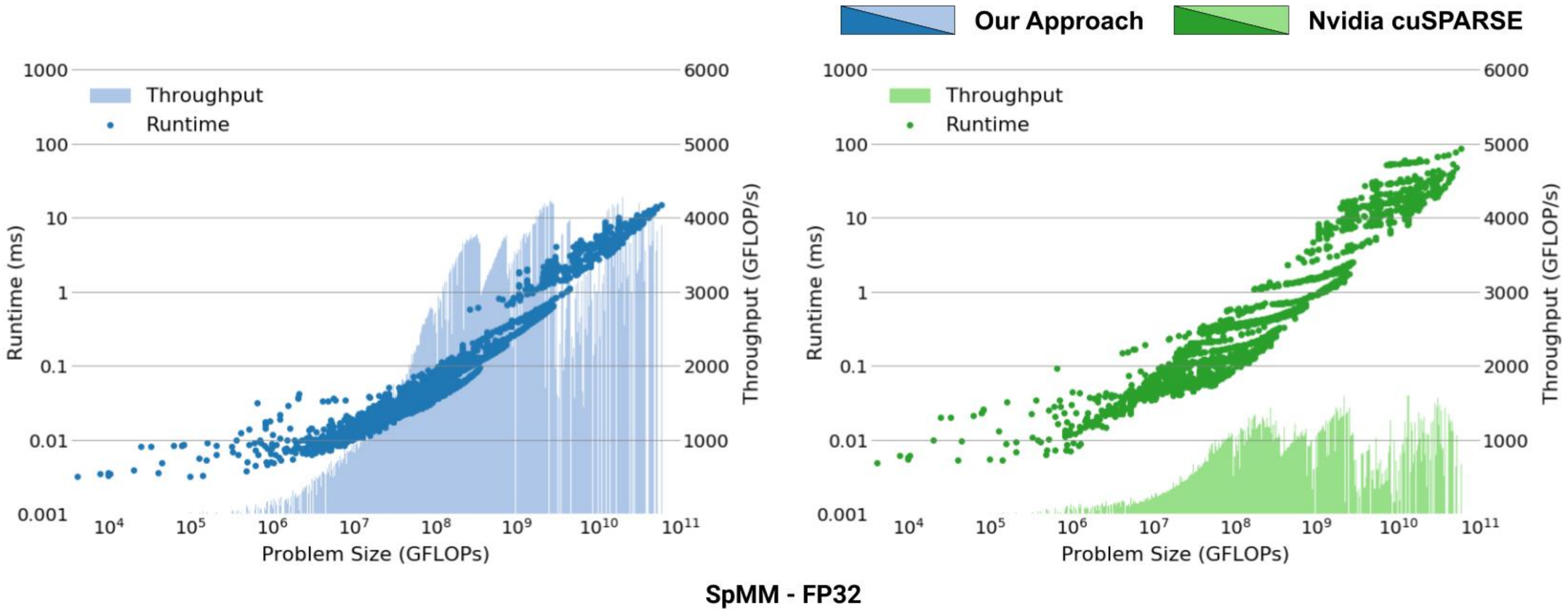
Sparse GPU Kernels for Deep Learning [Gale *et al.*, SC 2020]

GPU Support for SpMM



Sparse GPU Kernels for Deep Learning [Gale *et al.*, SC 2020]

GPU Support for SpMM



Sparse GPU Kernels for Deep Learning [Gale *et al.*, SC 2020]

GPU Support for SpMM

Results on MobileNetV1

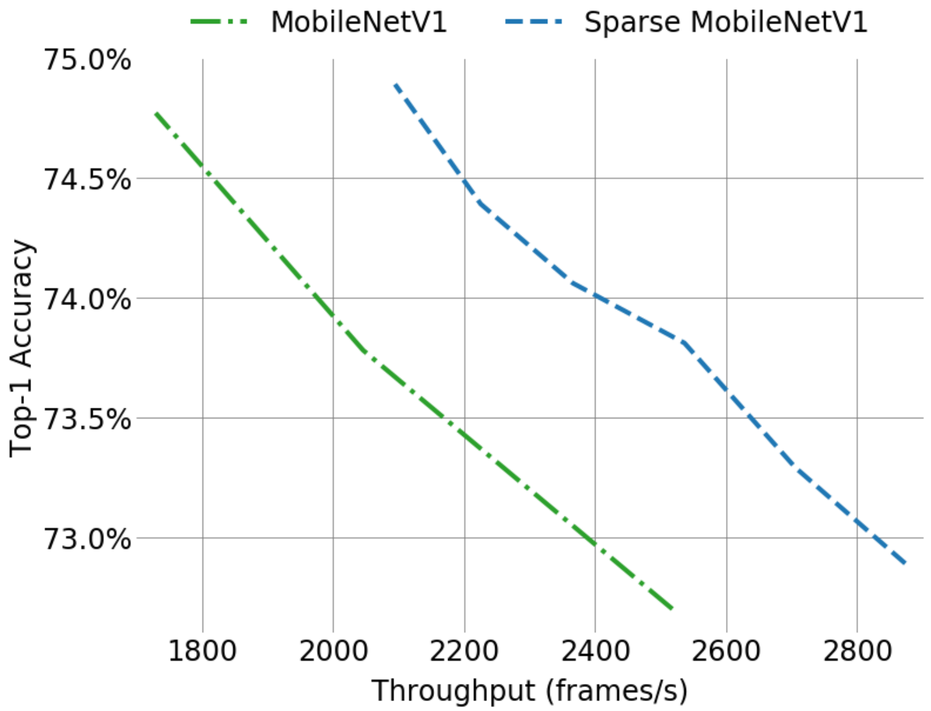


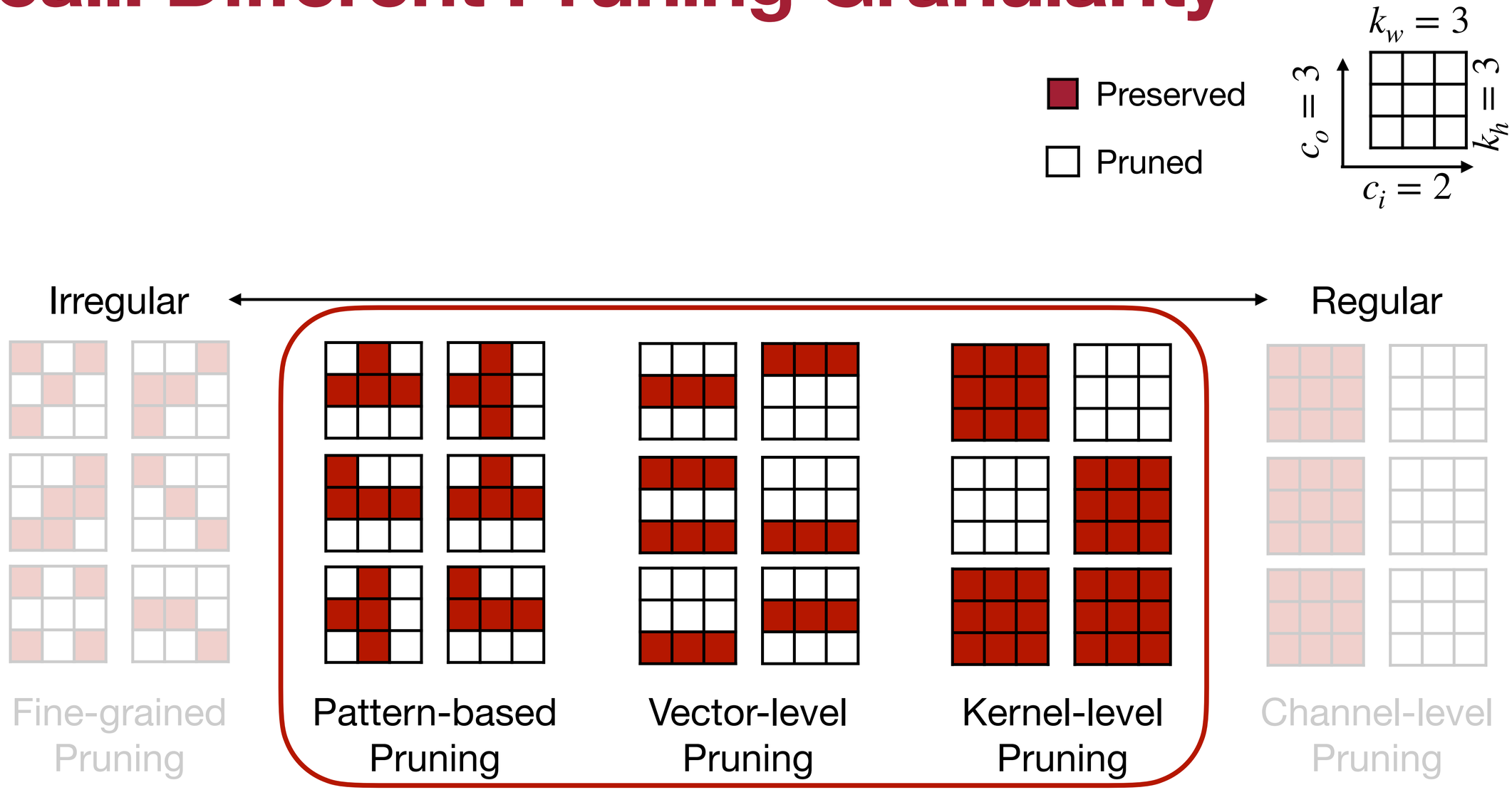
Fig. 12. **MobileNetV1 accuracy-runtime tradeoff curves.** Sparse modes are more efficient, achieving speedups of 21-24% for a given accuracy across all model sizes, or equivalently, ~1.1% higher accuracy for the same throughput.

TABLE IV
SPARSE MOBILENETV1 RESULTS

Model	Width	Accuracy	Throughput (frames/s)
Dense	1	72.7%	2,518
	1.2	73.8%	2,046
	1.4	74.8%	1,729
Sparse	1.3	72.9%	2,874
	1.4	73.3%	2,706
	1.5	73.8%	2,537
	1.6	74.1%	2,366
	1.7	74.4%	2,226
	1.8	74.9%	2,095

Sparse GPU Kernels for Deep Learning [Gale *et al.*, SC 2020]

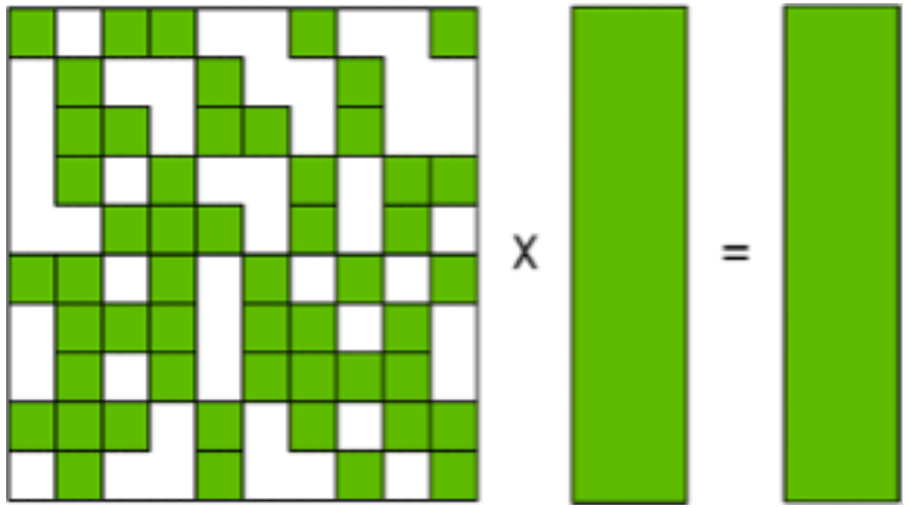
Recall: Different Pruning Granularity



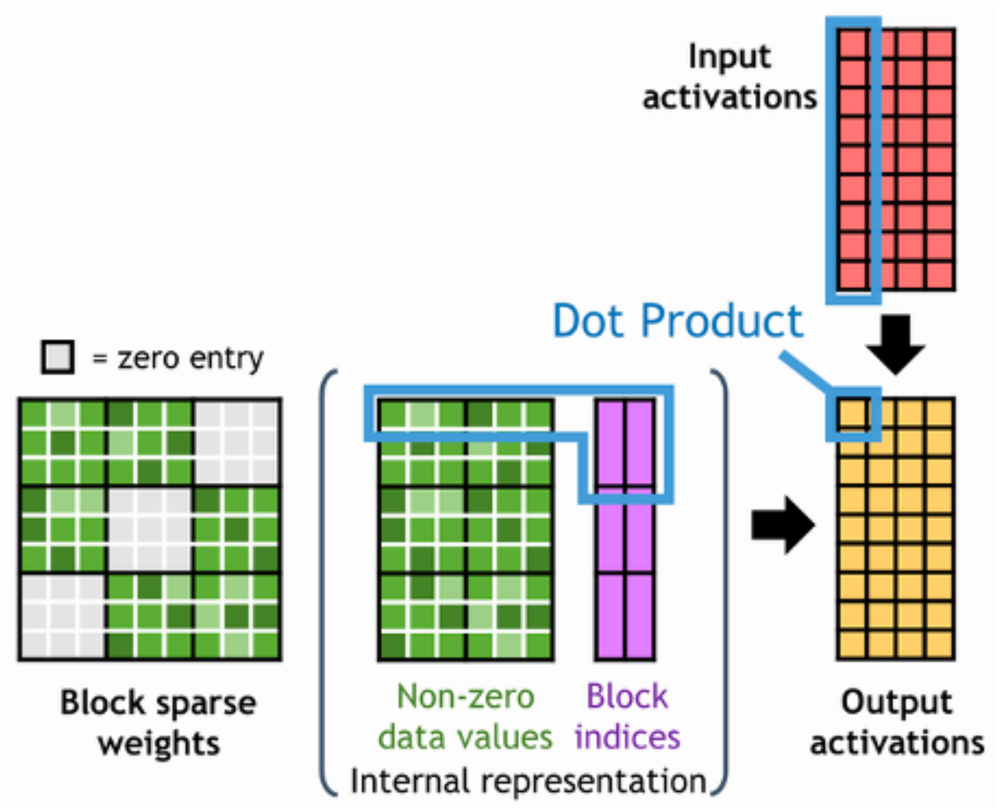
Exploring the granularity of sparsity in convolutional neural networks [Mao *et al.*, CVPR-W]

Block SpMM

Block Sparse Matrix-Matrix Multiplication



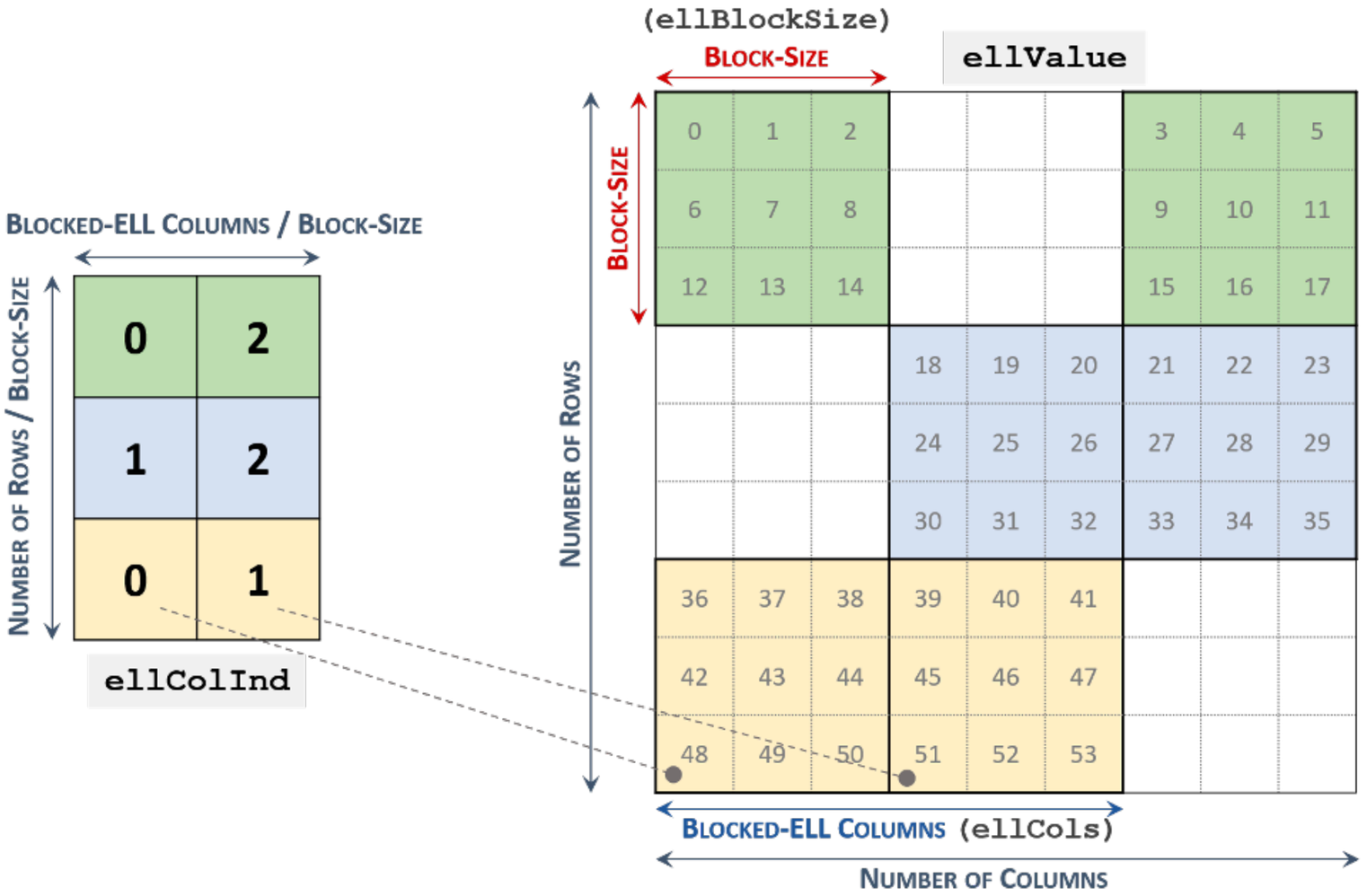
General SpMM: Irregular Sparsity



Block SpMM: **Structured** Sparsity

Block Sparse Format [NVIDIA, 2021]

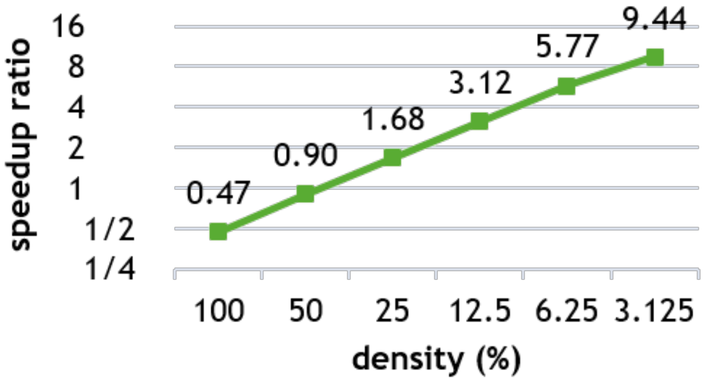
Blocked-Ellpack Format



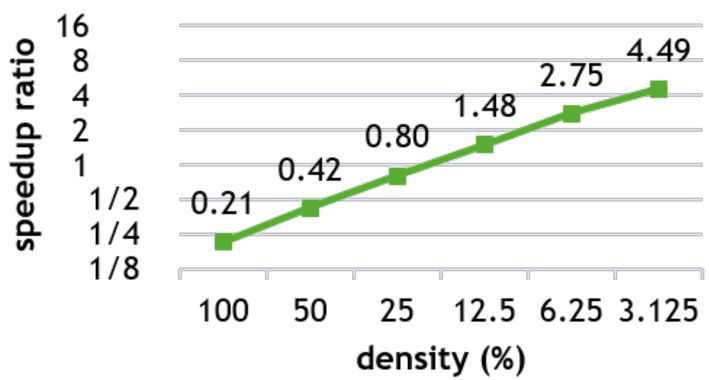
Recall the idea of CSR. There is an **index** array and a **value** array.

Block Sparse Format [NVIDIA, 2021]

Block SpMM

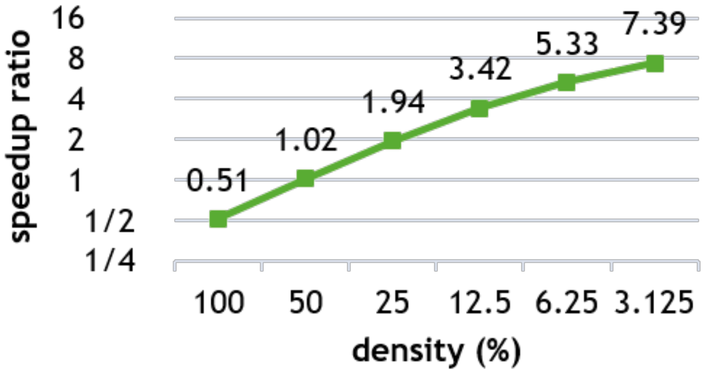


(a) block size = 32

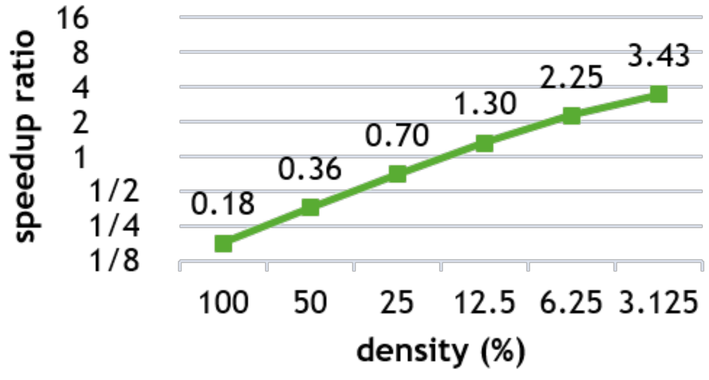


(b) block size = 16

Results on Tesla V100



(a) block size = 32

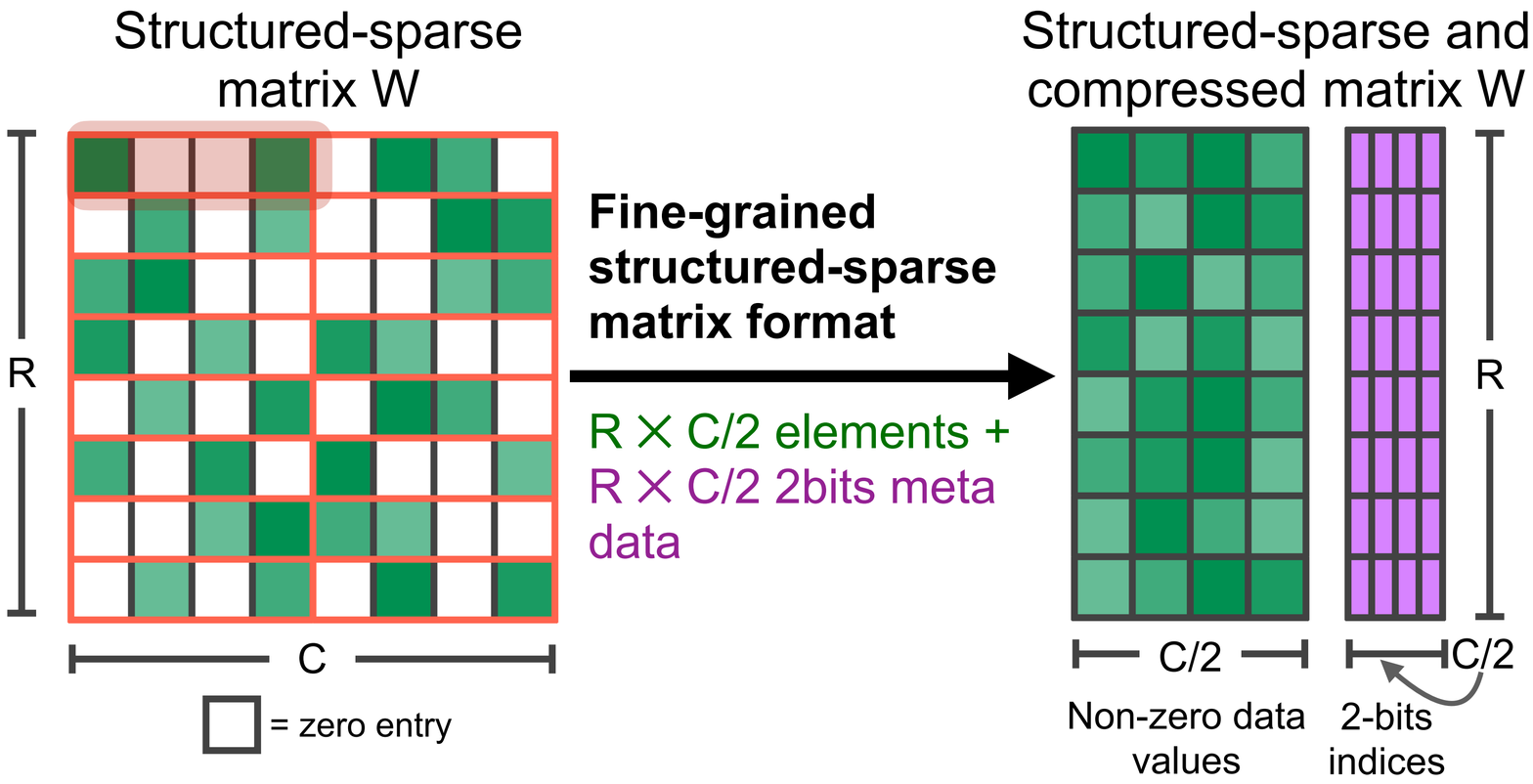


(b) block size = 16

Results on Tesla A100

Block Sparse Format [NVIDIA, 2021]

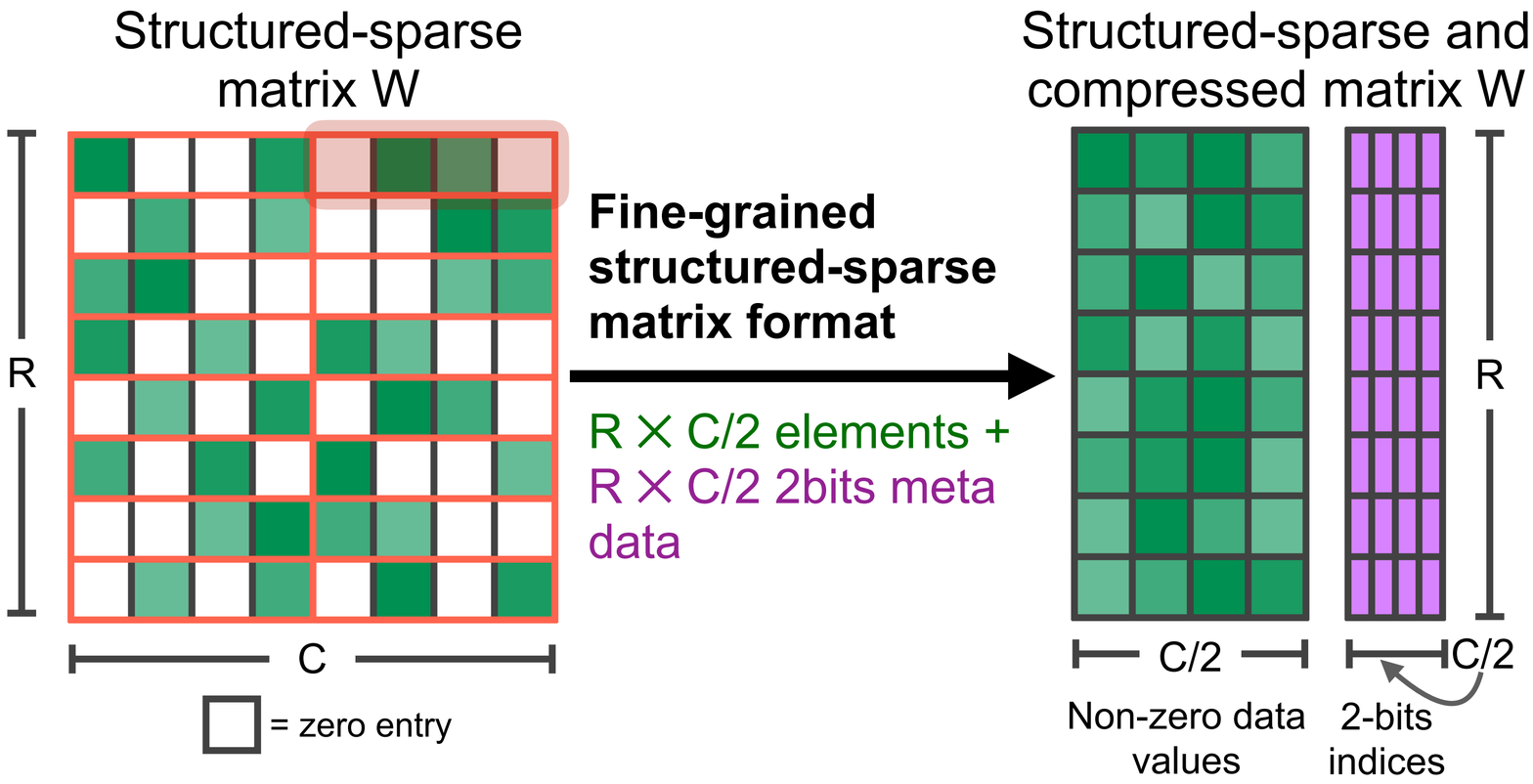
M:N Sparsity



Two weights are nonzero out of four consecutive weights (2:4 sparsity).

Accelerating Sparse Deep Neural Networks [Mishra *et al.*, arXiv 2021]

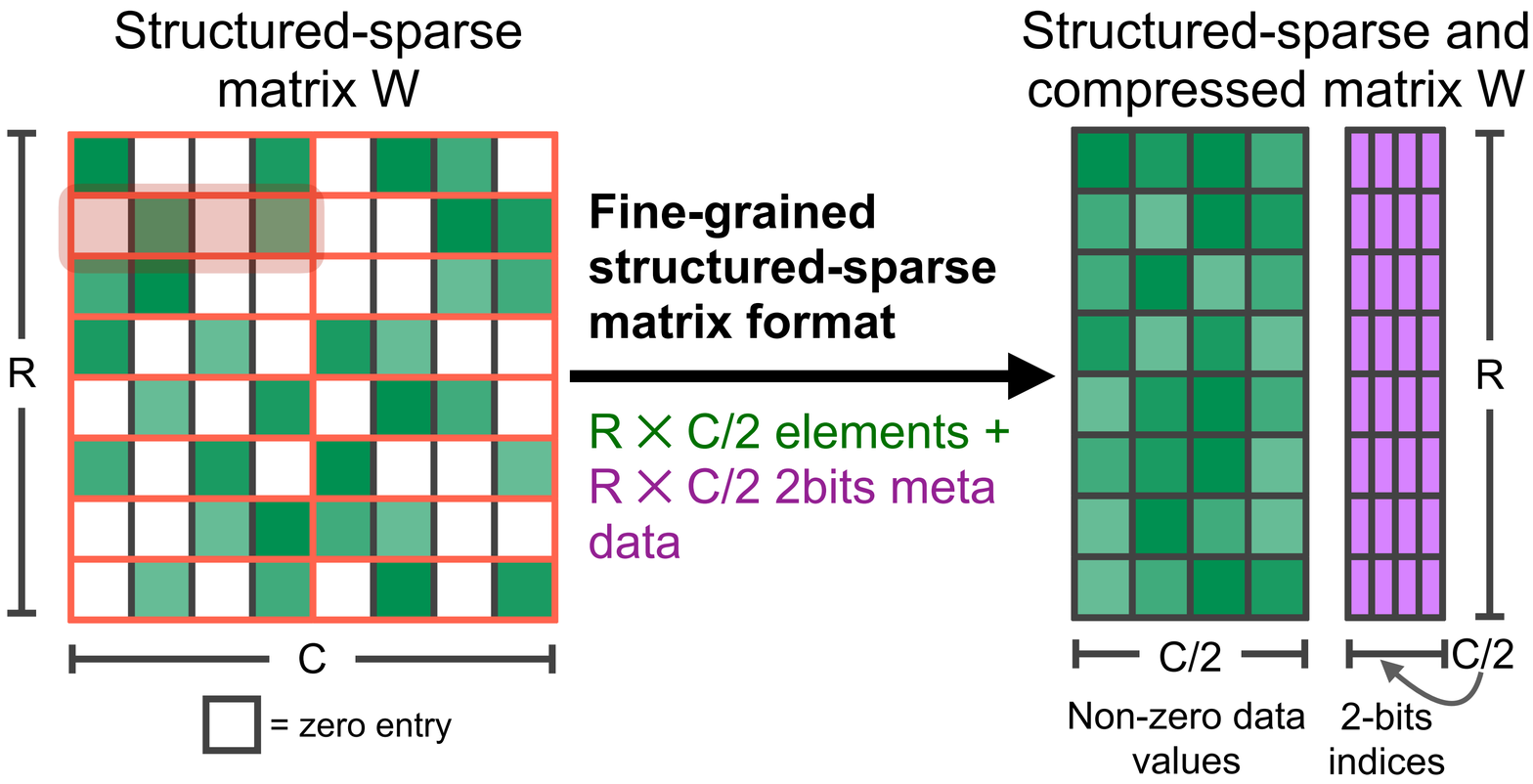
M:N Sparsity



Two weights are nonzero out of four consecutive weights (2:4 sparsity).

Accelerating Sparse Deep Neural Networks [Mishra *et al.*, arXiv 2021]

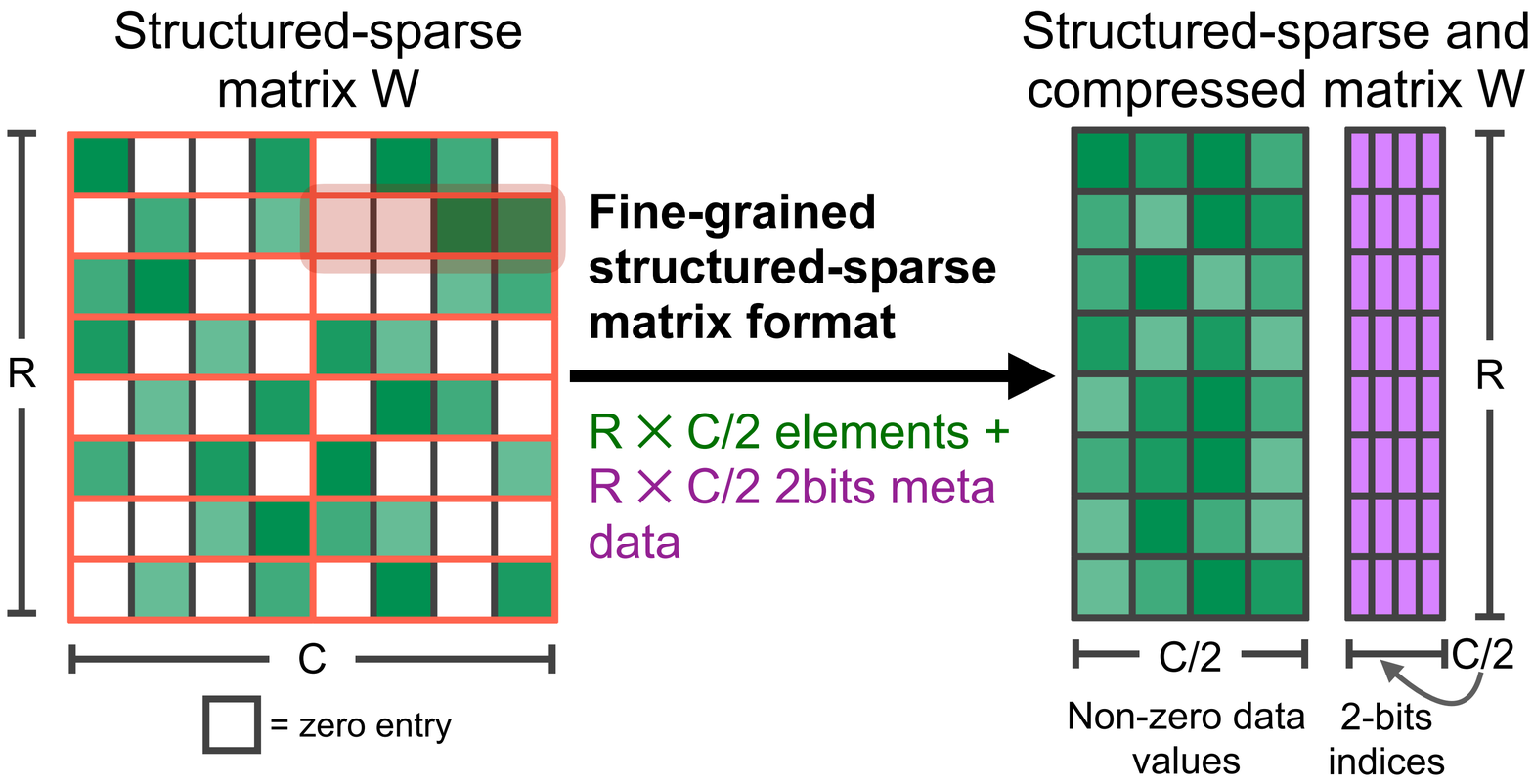
M:N Sparsity



Two weights are nonzero out of four consecutive weights (2:4 sparsity).

Accelerating Sparse Deep Neural Networks [Mishra *et al.*, arXiv 2021]

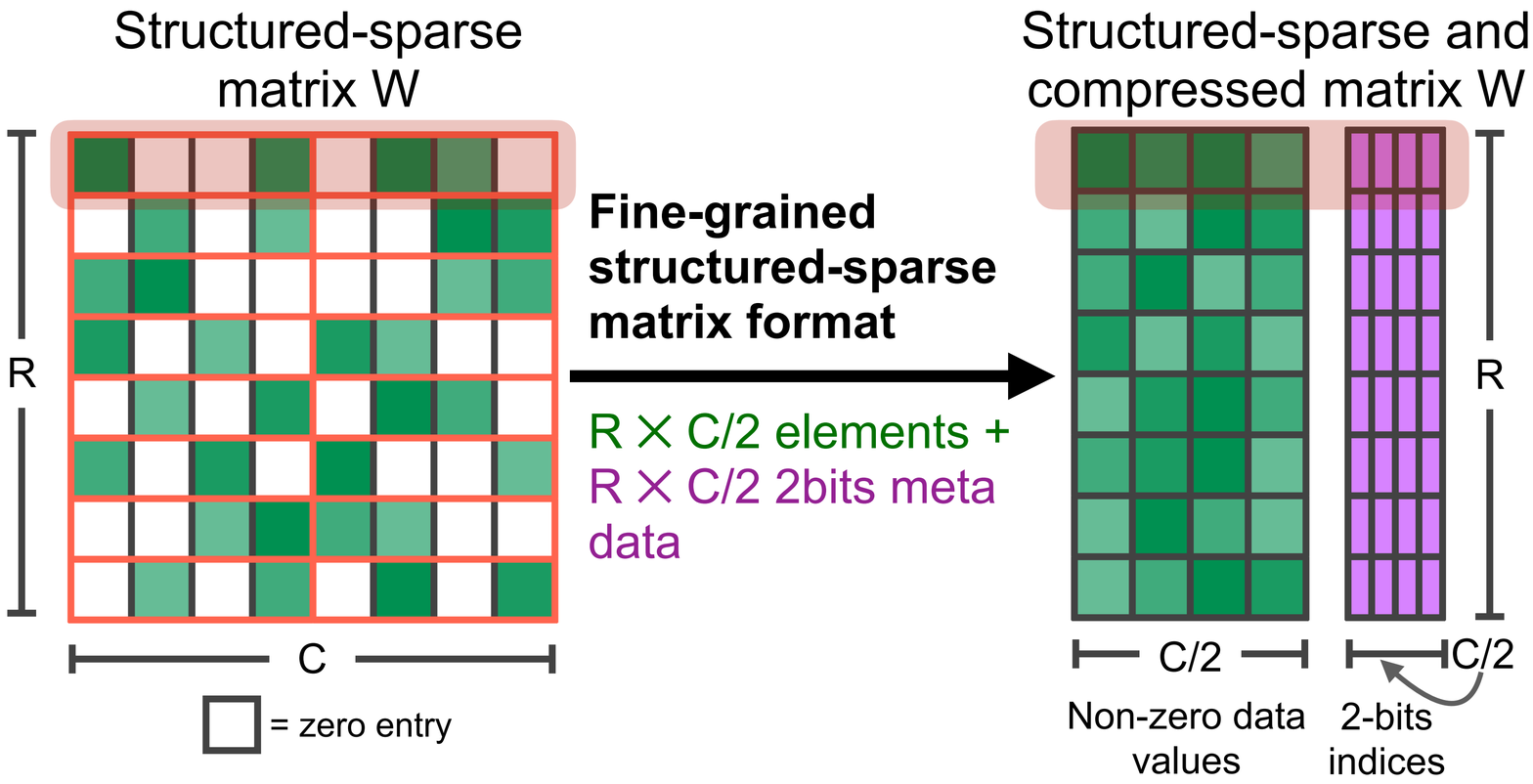
M:N Sparsity



Two weights are nonzero out of four consecutive weights (2:4 sparsity).

Accelerating Sparse Deep Neural Networks [Mishra *et al.*, arXiv 2021]

M:N Sparsity

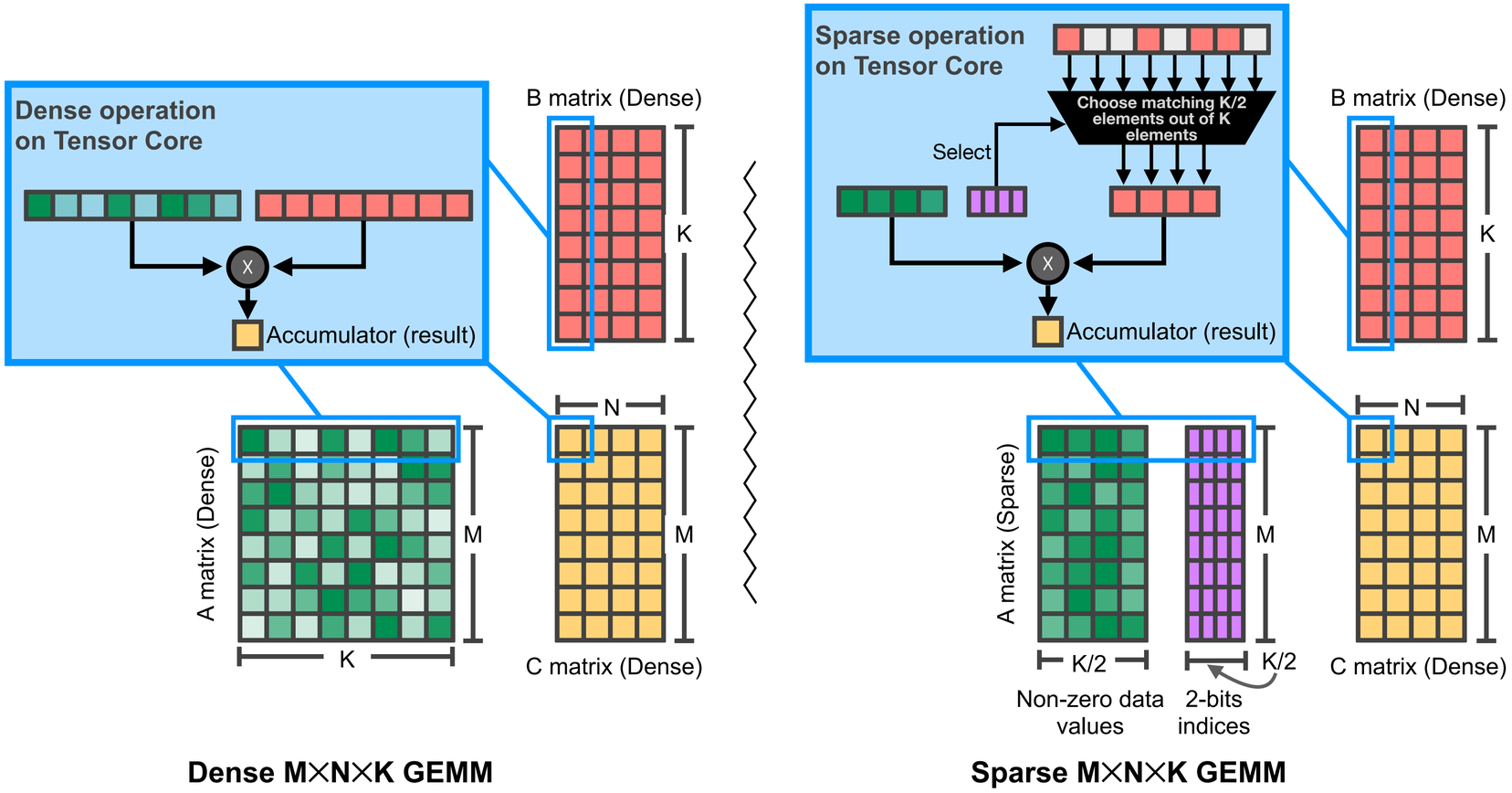


Push all the nonzero elements to the left in memory: save storage and computation.

Accelerating Sparse Deep Neural Networks [Mishra *et al.*, arXiv 2021]

System Support for M:N Sparsity

Mapping M:N sparse matrices onto NVIDIA tensor cores



The indices are used to mask out the inputs. Only 2 multiplications will be done out of four.

Accelerating Sparse Deep Neural Networks [Mishra *et al.*, arXiv 2021]

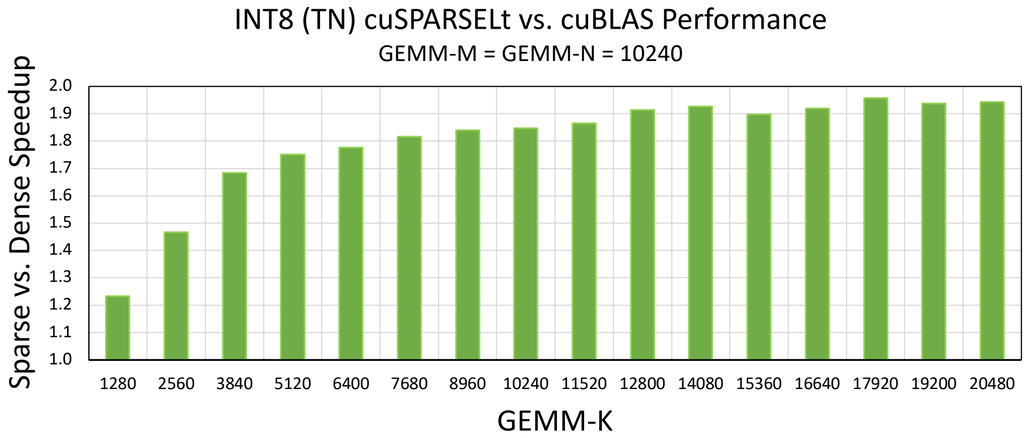


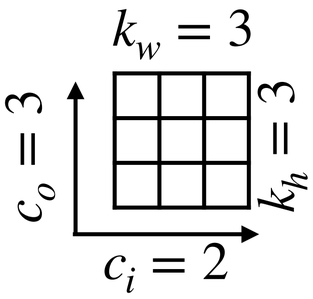
Fig. 3. Comparison of sparse and dense INT8 GEMMs on NVIDIA A100 Tensor Cores. Larger GEMMs achieve nearly a 2× speedup with Sparse Tensor Cores.

Network	Accuracy		
	Dense FP16	Sparse FP16	Sparse INT8
ResNet-34	73.7	73.9	73.7
ResNet-50	76.1	76.2	76.2
ResNet-50 (SWSL)	81.1	80.9	80.9
ResNet-101	77.7	78.0	77.9
ResNeXt-50-32x4	77.6	77.7	77.7
ResNeXt-101-32x16	79.7	79.9	79.9
ResNeXt-101-32x16 (WSL)	84.2	84.0	84.2
DenseNet-121	75.5	75.3	75.3
DenseNet-161	78.8	78.8	78.9
Wide ResNet-50	78.5	78.6	78.5
Wide ResNet-101	78.9	79.2	79.1
Inception v3	77.1	77.1	77.1
Xception	79.2	79.2	79.2
VGG-11	70.9	70.9	70.8
VGG-16	74.0	74.1	74.1
VGG-19	75.0	75.0	75.0
SUNet-128	75.6	76.0	75.4
SUNet-7-128	76.4	76.5	76.3
DRN26	75.2	75.3	75.3
DRN-105	79.4	79.5	79.4

Pruning CNNs with 2:4 sparsity will bring about large speedup for GEMM workloads and it will not incur performance drop for DNN models.

Accelerating Sparse Deep Neural Networks [Mishra *et al.*, arXiv 2021]

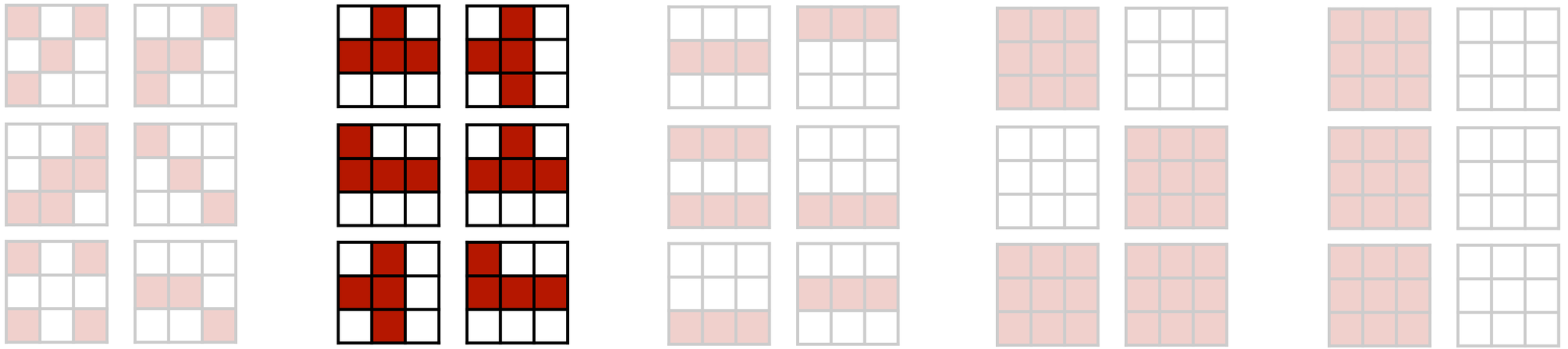
Recall: Different Pruning Granularity



Preserved

Pruned

Irregular → Regular



Fine-grained Pruning

Pattern-based Pruning

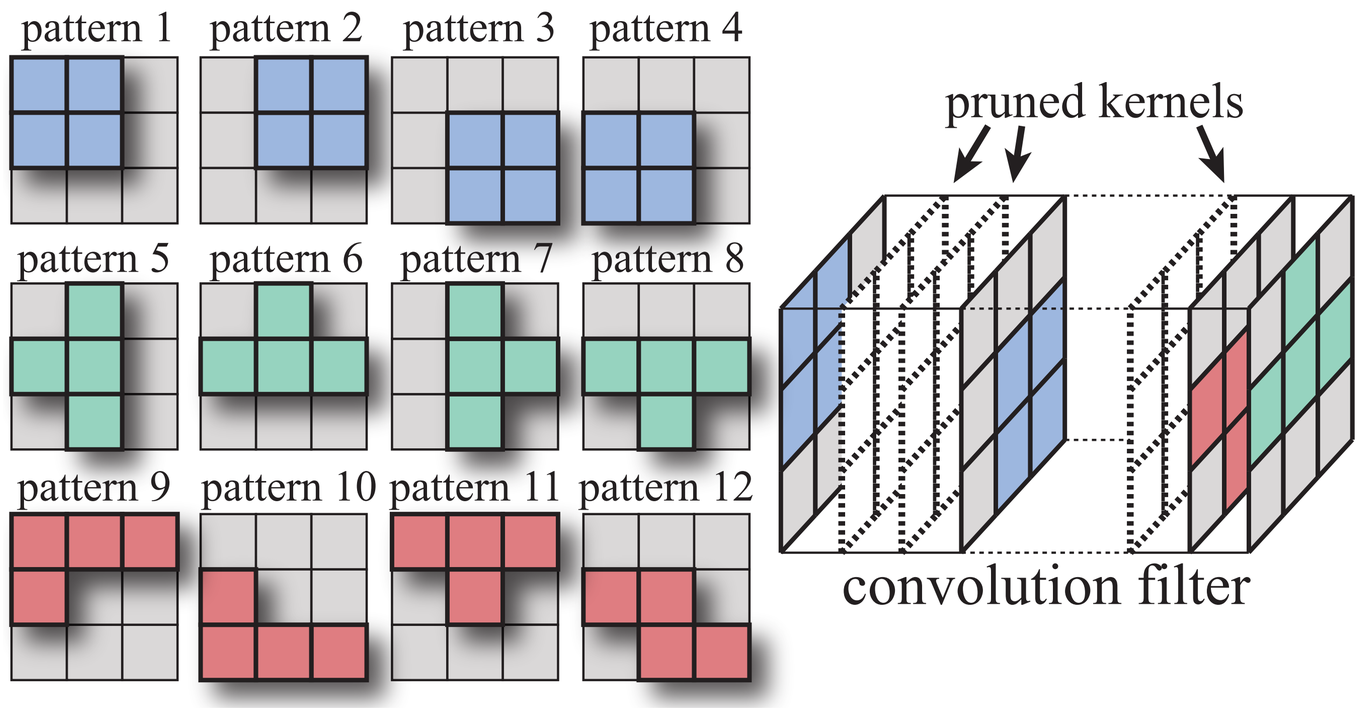
Vector-level Pruning

Kernel-level Pruning

Channel-level Pruning

Exploring the granularity of sparsity in convolutional neural networks [Mao *et al.*, CVPR-W]

Pattern-Based Sparsity

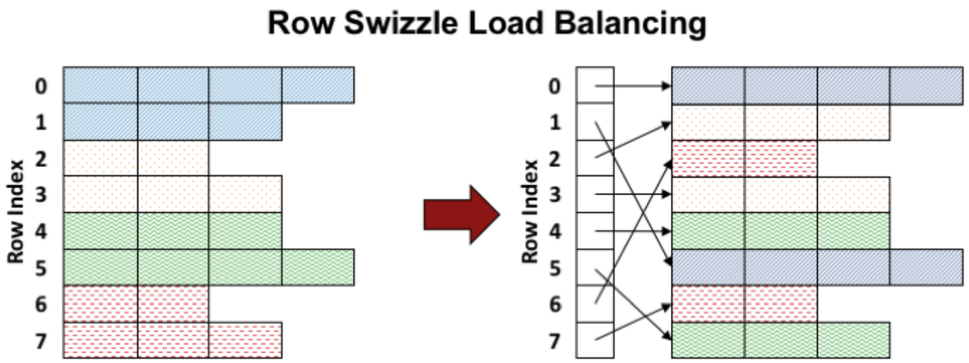


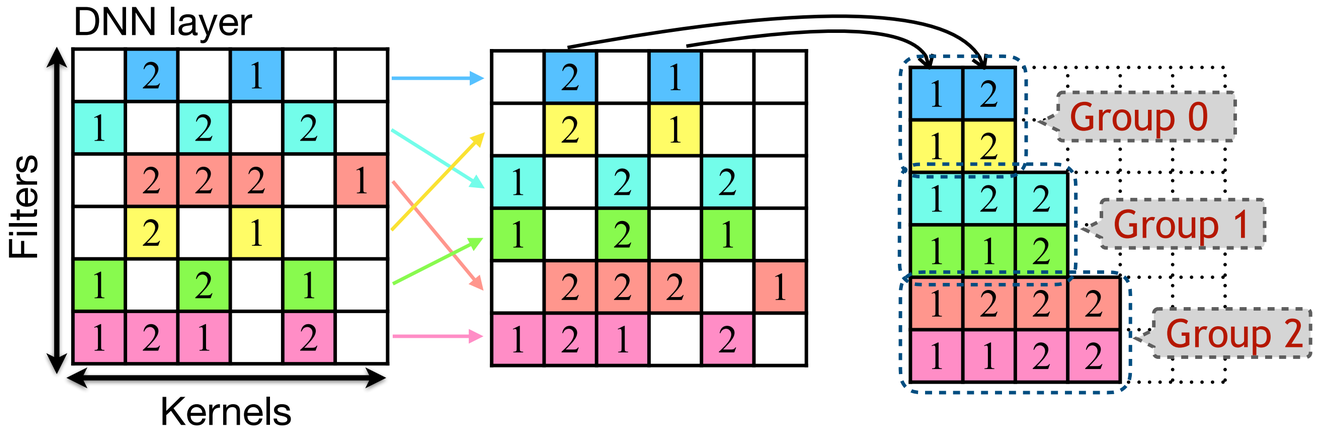
- For each output channel in the convolution filter, either **prune it away** or **select its sparsity pattern from a predefined set**.
- Problem: load imbalance between different channel groups (different number of pruned kernels).

An Efficient End-to-End Deep Learning Training Framework via Fine-Grained Pattern-Based Pruning [Zhang *et al.*, arXiv 2020]

GPU Support for Pattern-Based Sparsity

Filter Kernel Reorder





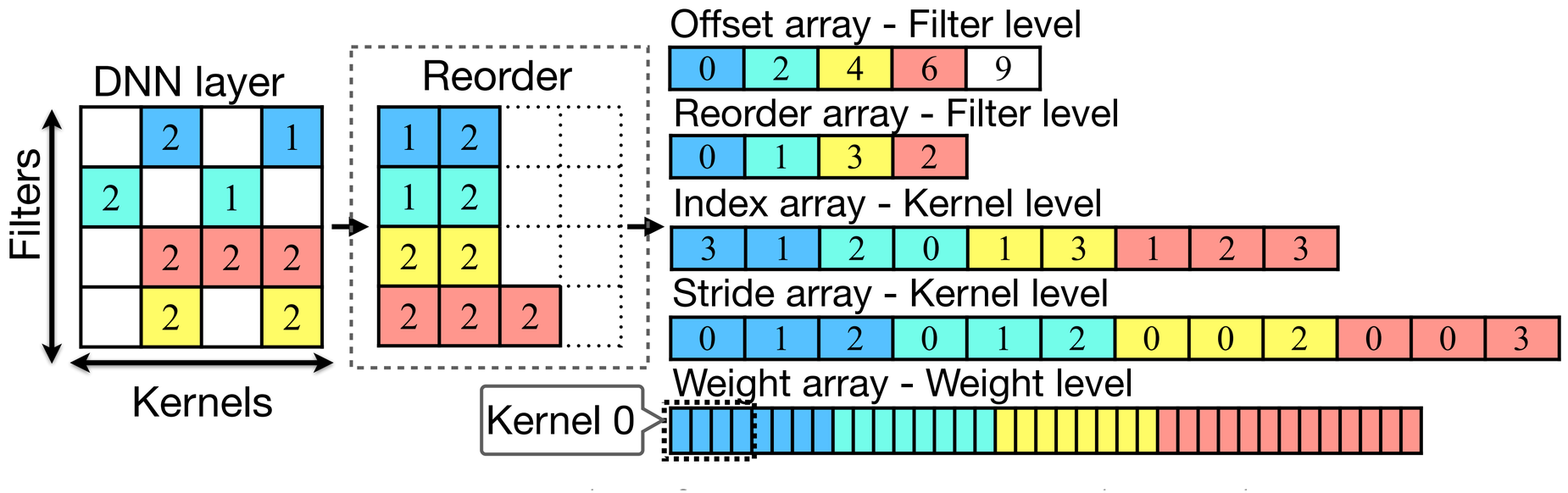
Recall: Row swizzling for SpMM → Filter Kernel Reorder for pruned CNNs

Within each group (warp), every thread has the same amount of work.
The number in the grid denotes different kernel **patterns**.

Sparse GPU Kernels for Deep Learning [Gale *et al.*, SC 2020]
PatDNN: Achieving Real-Time DNN Execution on Mobile Devices with Pattern-based Weight Pruning [Niu *et al.*, ASPLOS 2020]

GPU Support for Pattern-Based Sparsity

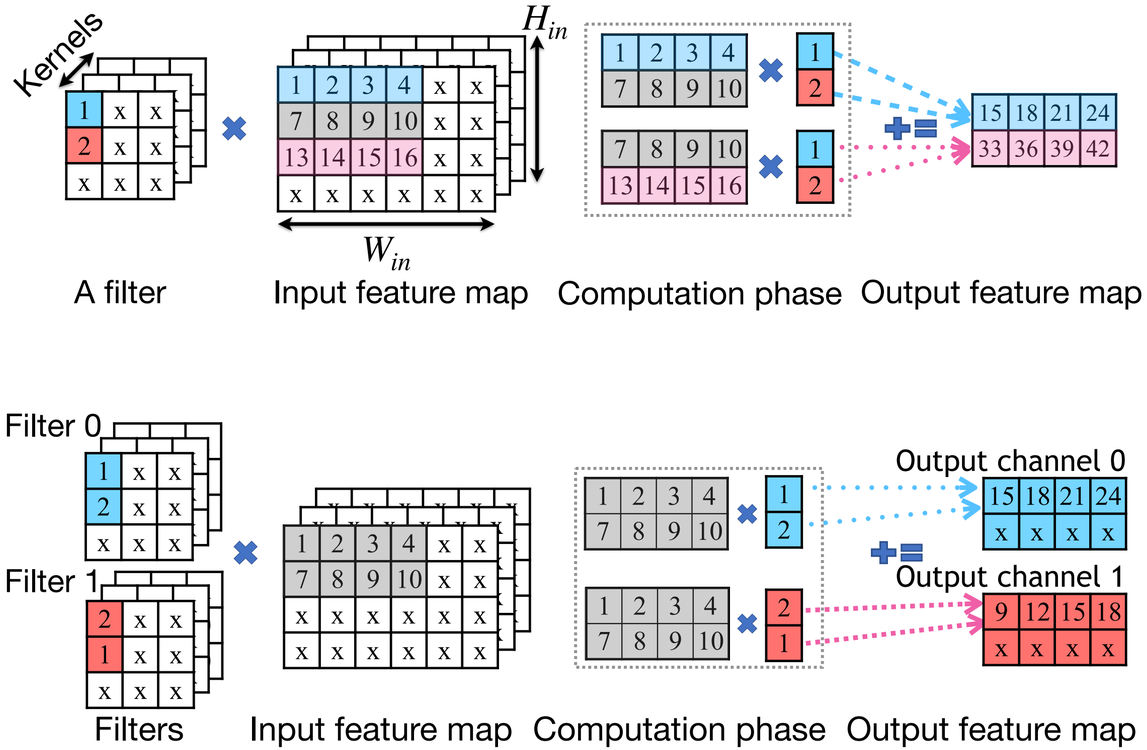
Filter-Kernel-Weight (FKW) Format



PatDNN: Achieving Real-Time DNN Execution on Mobile Devices with Pattern-based Weight Pruning [Niu *et al.*, ASPLOS 2020]

GPU Support for Pattern-Based Sparsity

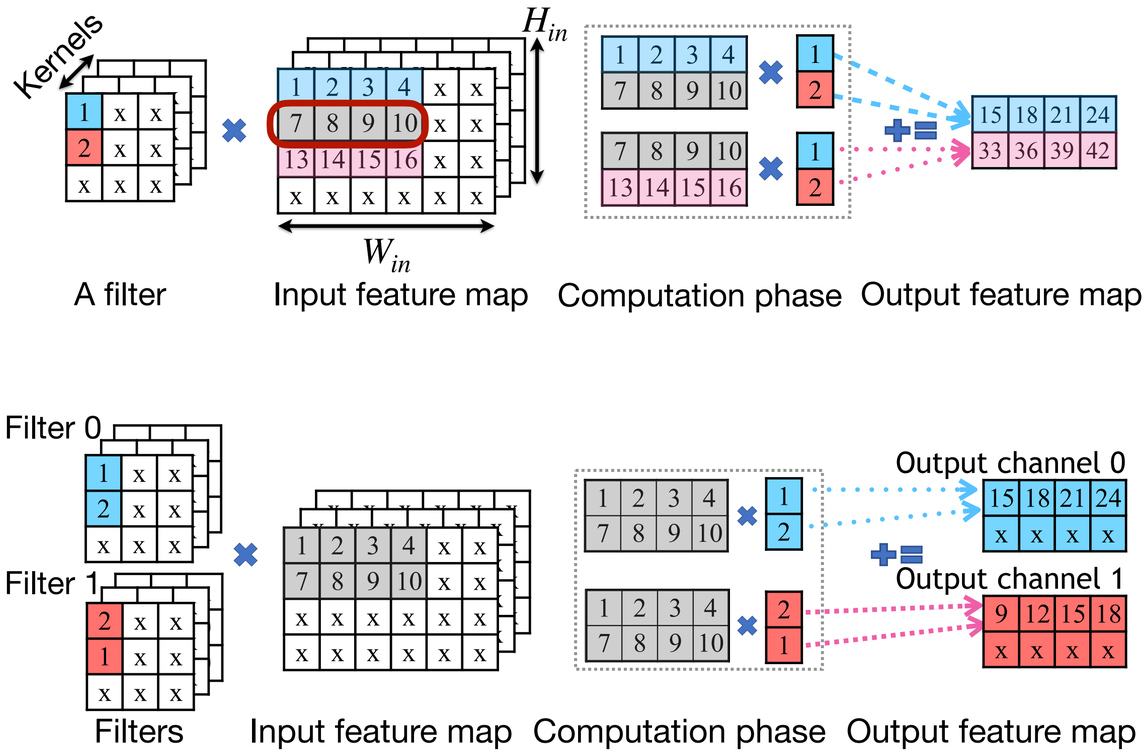
Load Redundancy Elimination



PatDNN: Achieving Real-Time DNN Execution on Mobile Devices with Pattern-based Weight Pruning [Niu *et al.*, ASPLOS 2020]

GPU Support for Pattern-Based Sparsity

Load Redundancy Elimination

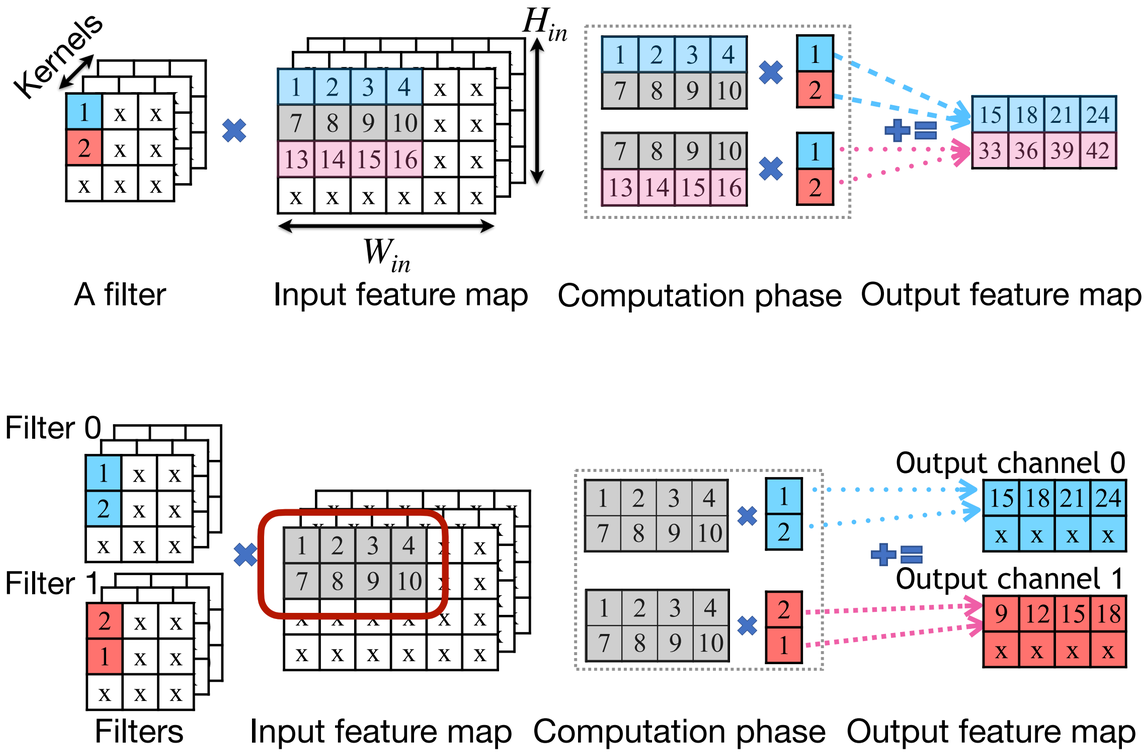


Kernel values 0 and 1 both require accessing the same row of input feature map.

PatDNN: Achieving Real-Time DNN Execution on Mobile Devices with Pattern-based Weight Pruning [Niu *et al.*, ASPLOS 2020]

GPU Support for Pattern-Based Sparsity

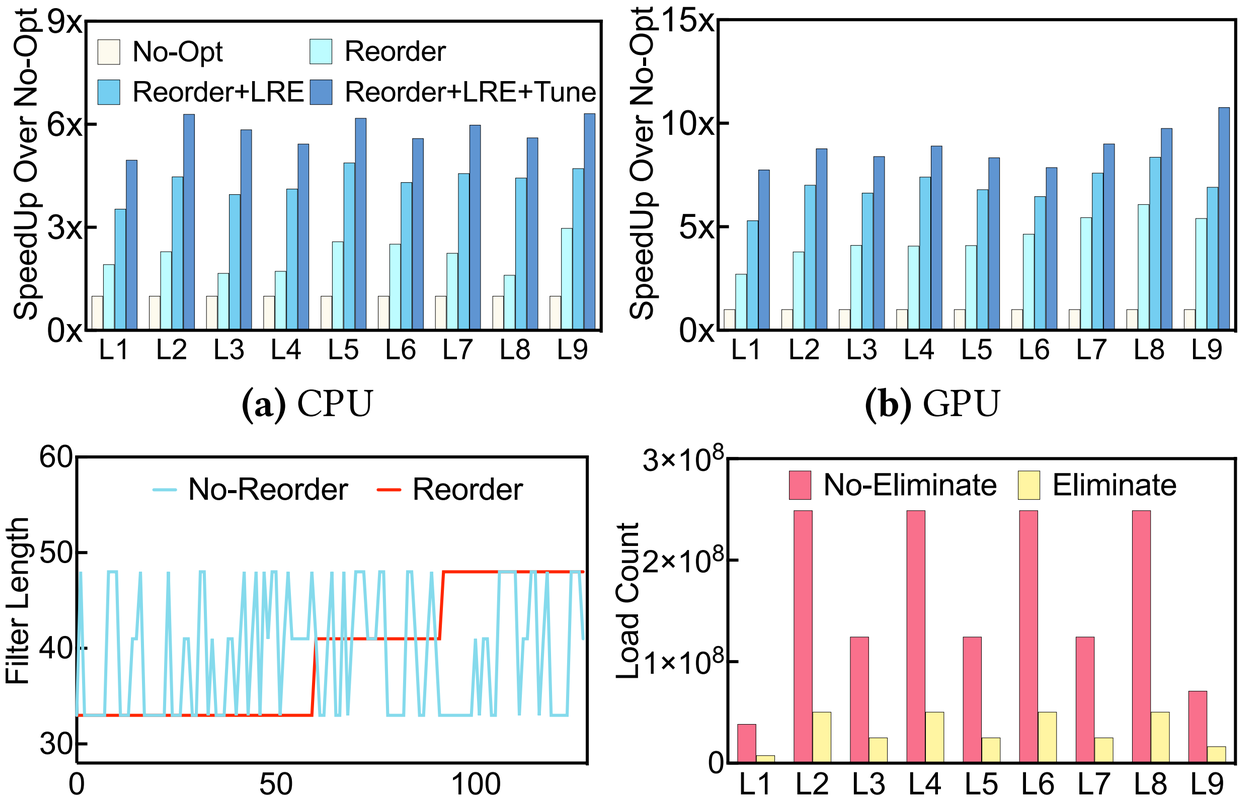
Load Redundancy Elimination



Filters 0 and 1 both require accessing the same trunk of input data

PatDNN: Achieving Real-Time DNN Execution on Mobile Devices with Pattern-based Weight Pruning [Niu *et al.*, ASPLOS 2020]

GPU Support for Pattern-Based Sparsity

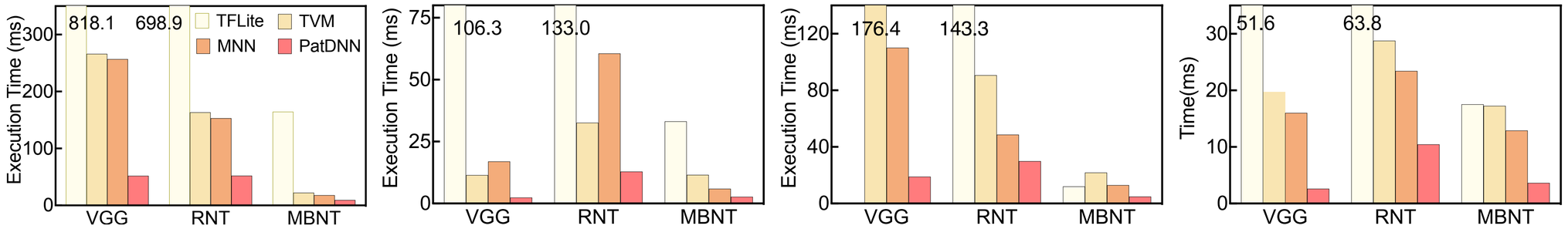


(a) Filter length distribution before and after filter kernel reorder for L4

(b) Register load counts before and after elimination

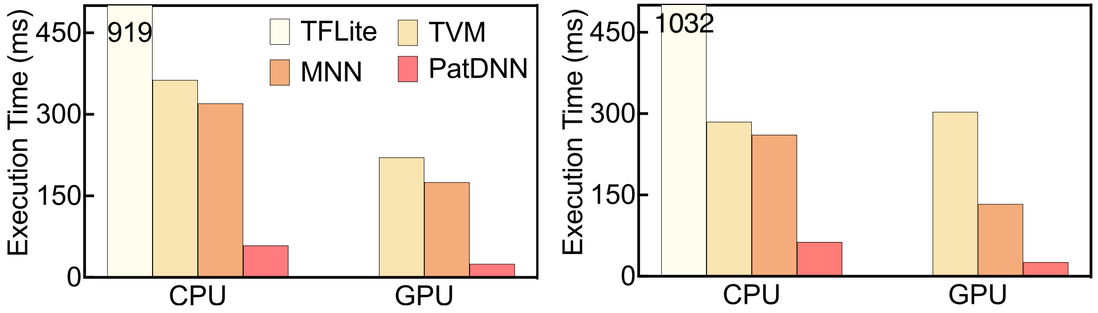
PatDNN: Achieving Real-Time DNN Execution on Mobile Devices with Pattern-based Weight Pruning [Niu *et al.*, ASPLOS 2020]

GPU Support for Pattern-Based Sparsity



(a) ImageNet-CPU **(b)** CIFAR-10-CPU **(c)** ImageNet-GPU **(d)** CIFAR-10-GPU

Figure 12. Overall performance: x-axis: different trained DNN models; y-axis: average DNN inference execution time on a single input.



(a) Kirin 980 **(b)** Snapdragon 845

Figure 18. Portability study: performance on two other platforms.

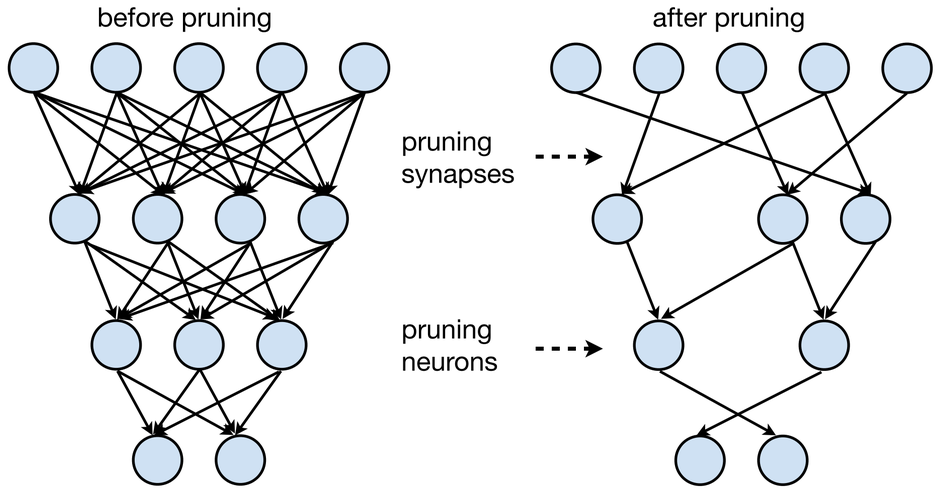
PatDNN: Achieving Real-Time DNN Execution on Mobile Devices with Pattern-based Weight Pruning [Niu *et al.*, ASPLOS 2020]

Summary of Today's Lecture

In this lecture, we introduced:

- Automated ways to find pruning ratios
- Lottery ticket hypothesis
- System support for different granularities

- **We will cover in the next lecture:**
  - Numeric data types in modern computer systems
  - Basic concept of neural network quantization
  - Common neural network quantization methods



References

1. Learning Both Weights and Connections for Efficient Neural Network [Han et al., NeurIPS 2015]
2. Exploring the granularity of sparsity in convolutional neural networks [Mao et al., CVPR-W]
3. Learning Structured Sparsity in Deep Neural Networks [Wen et al., NeurIPS 2016]
4. Learning Efficient Convolutional Networks through Network Slimming [Liu et al., ICCV 2017]
5. A Systematic DNN Weight Pruning Framework using Alternating Direction Method of Multipliers [Zhang et al., ECCV 2018]
6. AMC: Automl for Model Compression and Acceleration on Mobile Devices [He et al., ECCV 2018]
7. Accelerating Inference with Sparsity Using the NVIDIA Ampere Architecture and NVIDIA TensorRT
8. EIE: Efficient Inference Engine on Compressed Deep Neural Network [Han et al., ISCA 2016]
9. ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA [Han et al., FPGA 2017]
10. Block Sparse Format [NVIDIA, 2021]
11. Accelerating Sparse Deep Neural Networks [Mishra et al., arXiv 2021]