TinyML and Efficient Deep Learning Computing

Pruning and Sparsity

발표자 신호주

Contents

MLPerf (the Olympic Game for AI Computing)

- 특정 정확도를 유지하며 Throughput & latency 성능 측정
- **Closed Division**: 네트워크 구조를 변경하지 않고, quantization만 사용
- **Open Division**: 네트워크 구조를 자유롭게 변경할 수 있으며, pruning 기술 사용 가능

	Closed Division	Open Division	Speedup
Offline samples/sec	4488 →	11189	2.5x

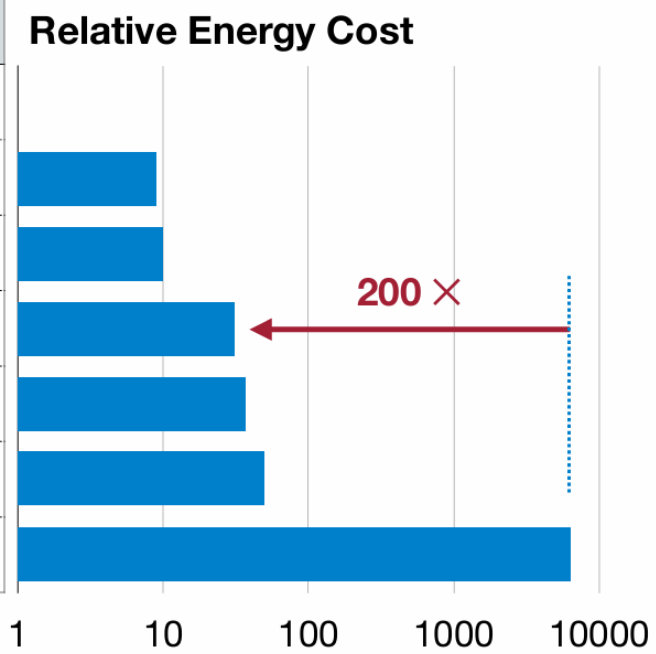
Llama 2 70B performance metrics for both closed division and open division.
Measured on a single NVIDIA H200 GPU.

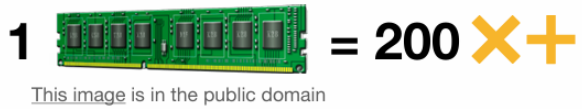
Memory is Expensive

- 메모리 이동은 computation 보다 훨씬 비용이 비쌈 (e.g., 0.9 → 640 pJ)
- **메모리 사용** / **모델 사이즈** / **활성화 함수**의 크기를 줄여야 함

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

Rough Energy Cost For Various Operations in 45nm 0.9V

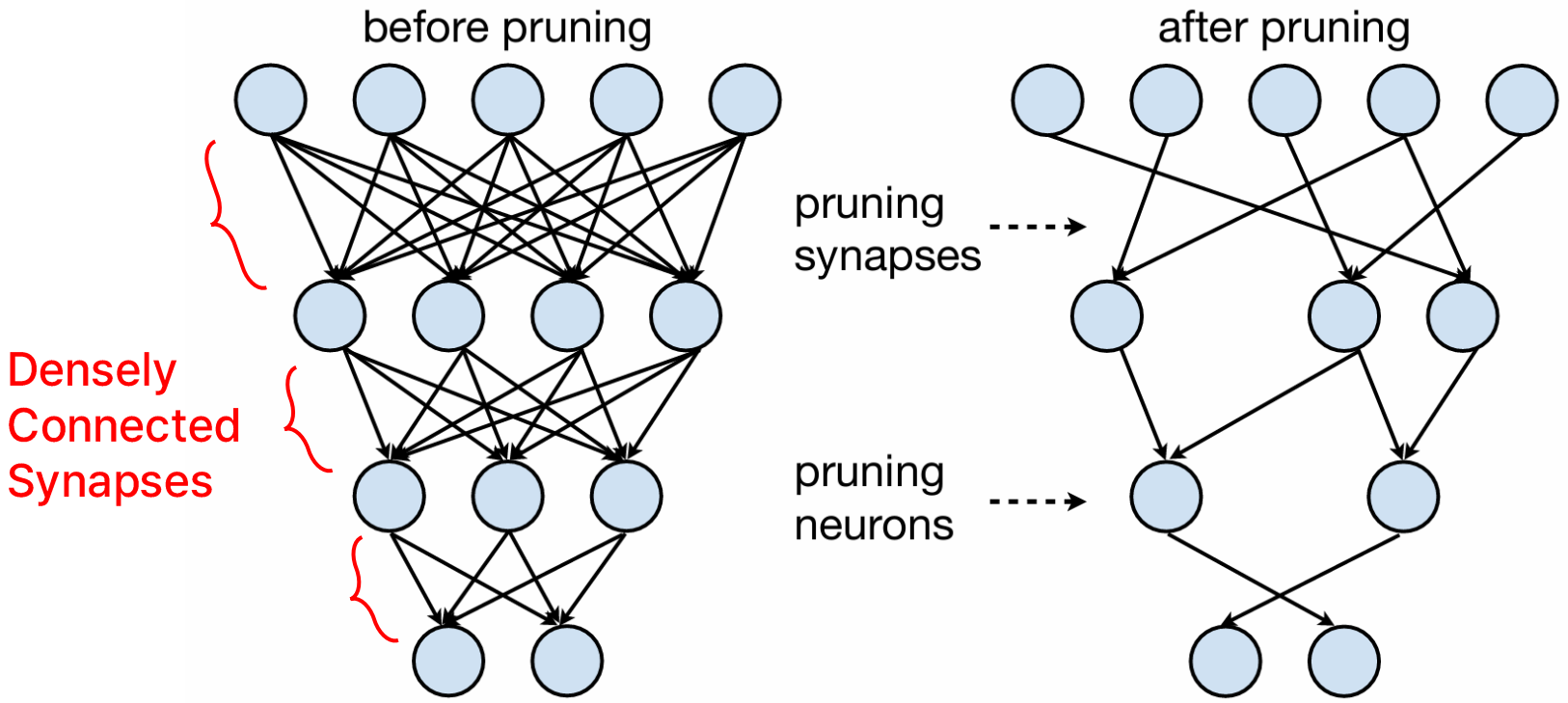




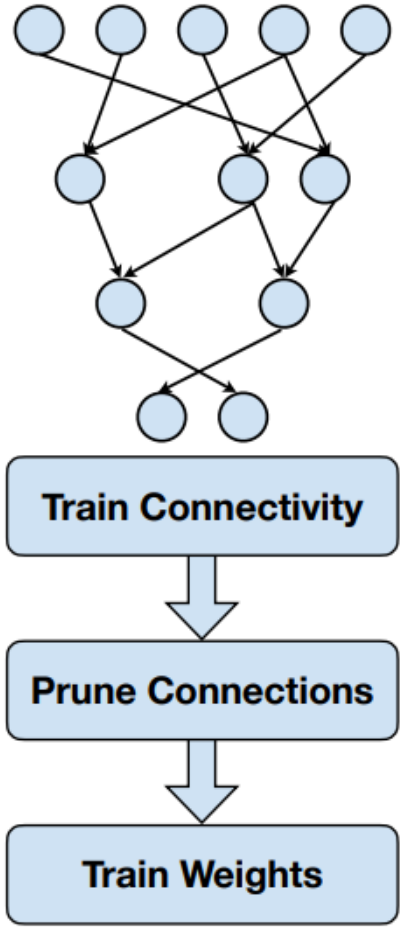
This image is in the public domain

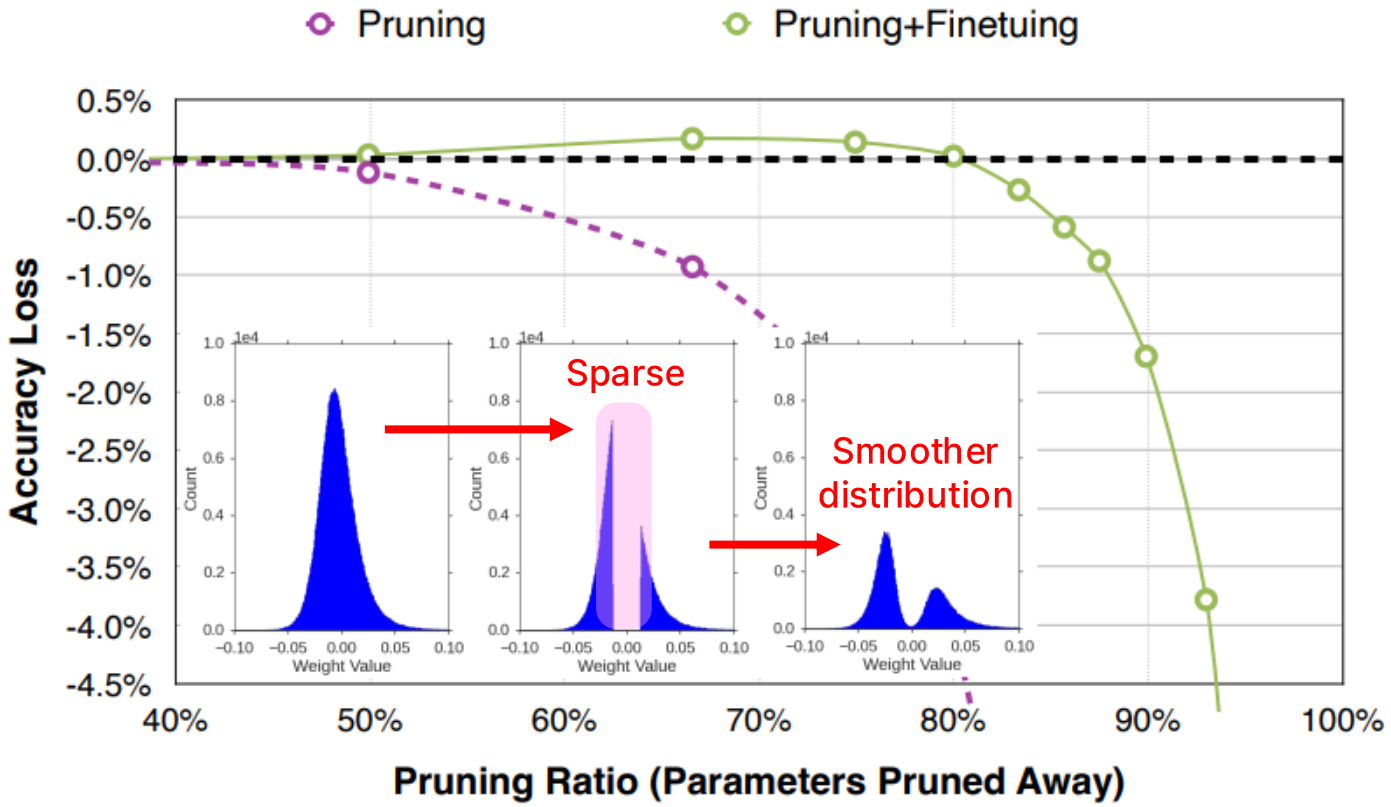
Neural Network Pruning

- Neurons & synapses 제거하여 네트워크를 작게 만드는 기술
- 네트워크의 중복된 (redundant) 부분을 제거하여 성능 저하를 최소화하면서도 연산 효율성 높임



Neural Network Pruning

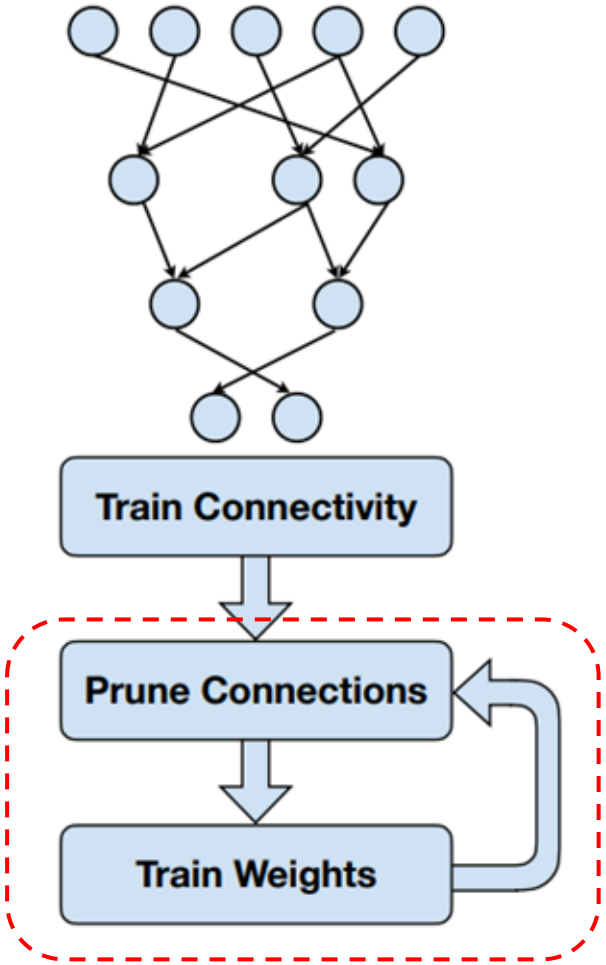




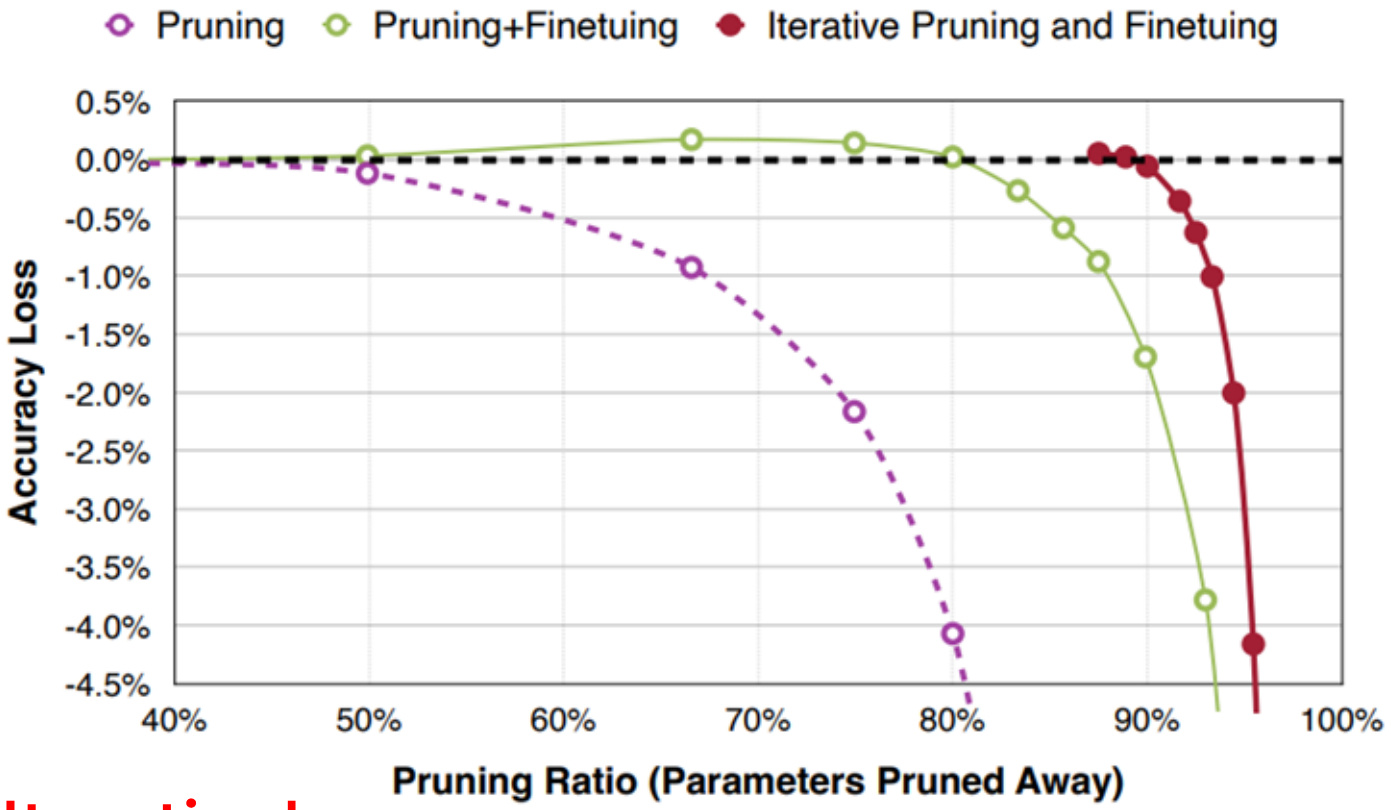
Magnitude 기준으로 작은 weight 제거

Fine-tuning pruned network

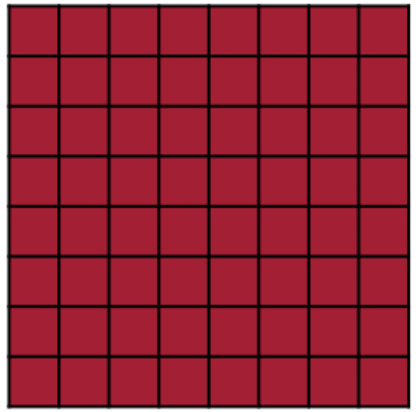
Neural Network Pruning



Iteratively

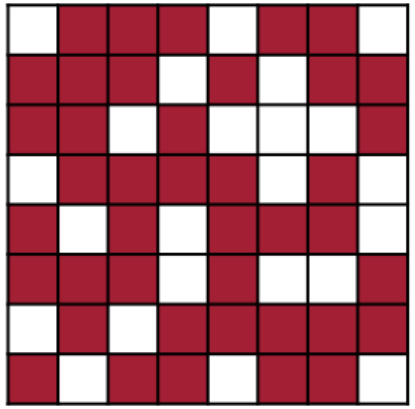


Pruning at Different Granularities



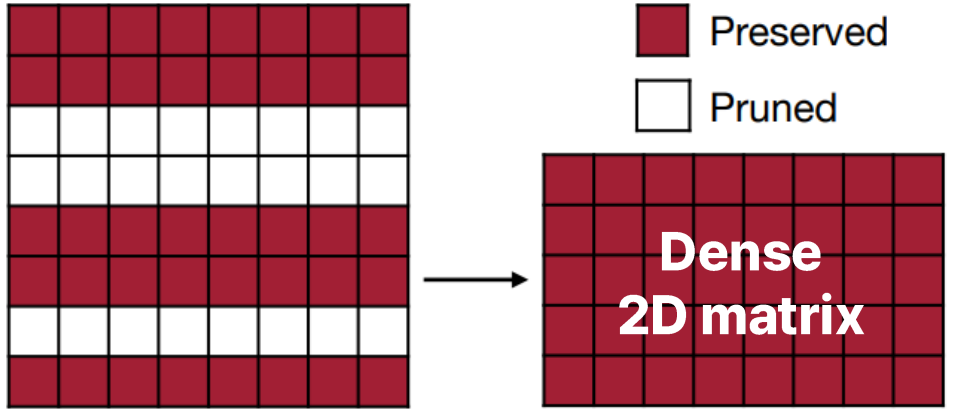
Dense 2D Weight Matrix

Inputs & outputs are densely connected



Fine-grained

- 위치 상관없이 pruning 가능
- 하드웨어 가속 어려움 (irregularity)

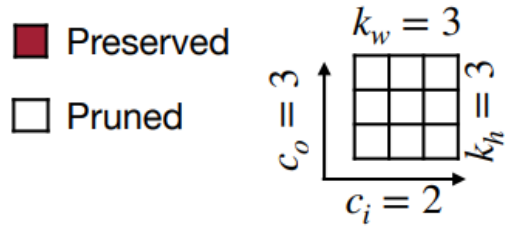


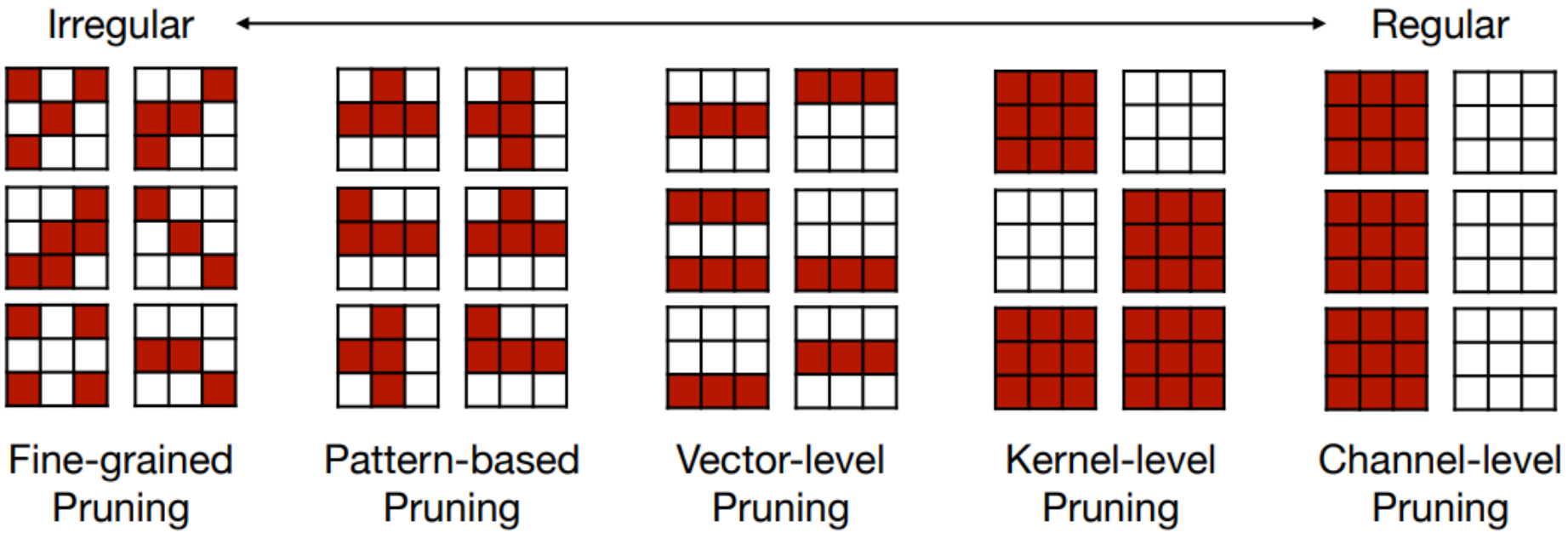
Coarse-grained

- Pruning 자유도 낮음
- 일반적인 하드웨어에서도 가속 쉬움 (smaller matrix 형태)

Pruning at Different Granularities

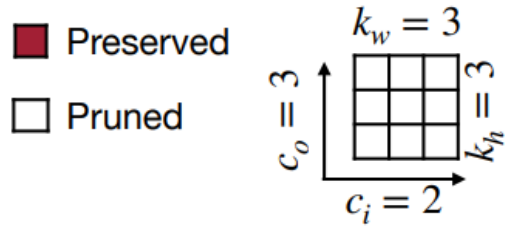
- Convolution Layer: $[C_o, C_i, k_h, k_w]$ 4개의 차원으로 이루어짐
- 더 많은 pruning granularity 가짐

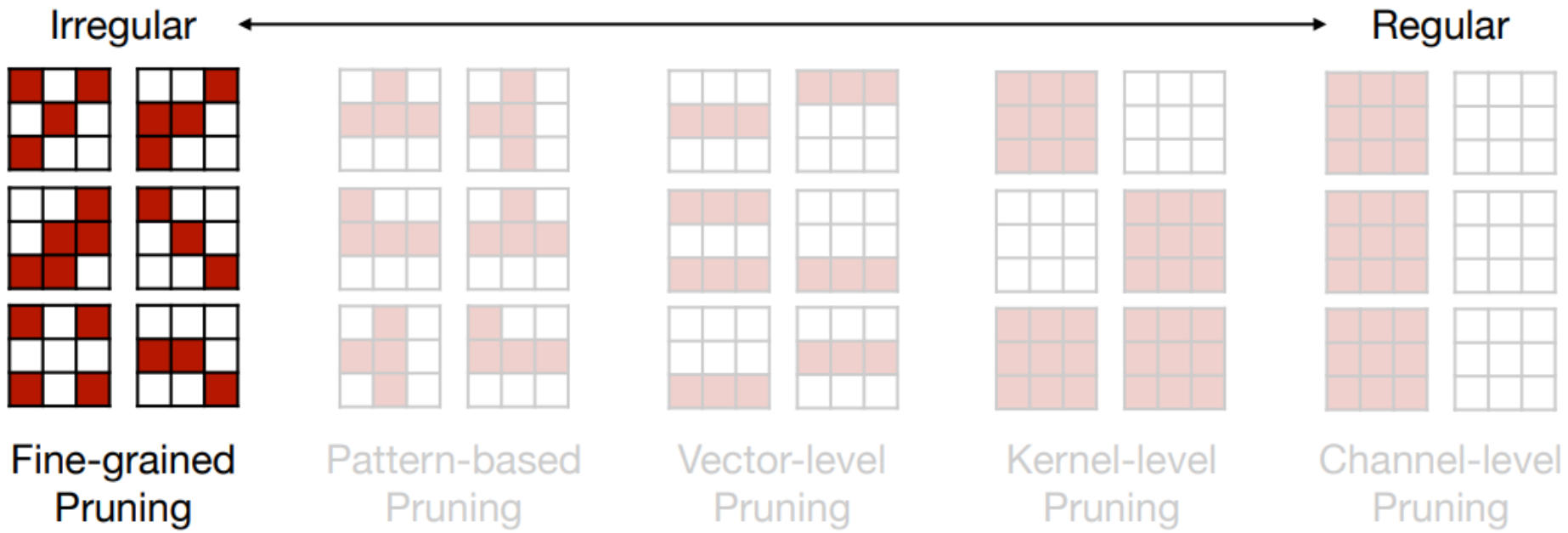




Pruning at Different Granularities

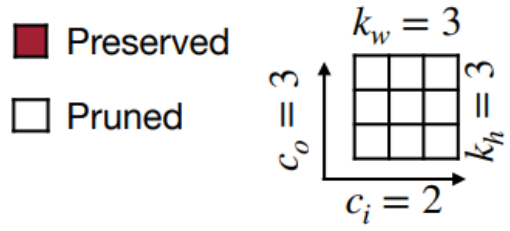
- **Fine-grained Pruning**
  - Redundant weight를 자유롭게 찾을 수 있기 때문에 일반적으로 더 높은 압축 비율을 가짐
  - 커스텀 하드웨어 (e.g., Efficient Inference Engine) 에서 가속 가능

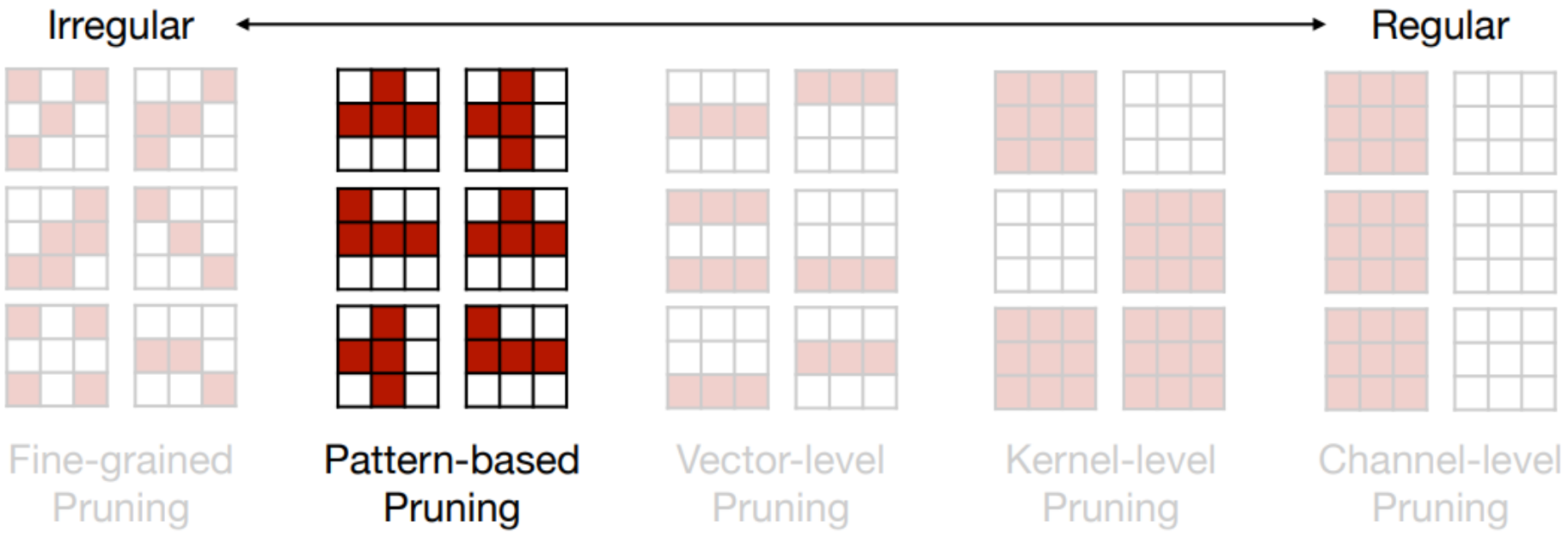




Pruning at Different Granularities

- **Pattern-based Pruning:** N:M sparsity
  - M개의 요소가 있을 때, N개의 요소를 Pruning 하는 기법
  - 일반적으로, 2:4 sparsity (50% sparsity) 사용
  - NVIDIA Ampere GPU에서 2× speed up

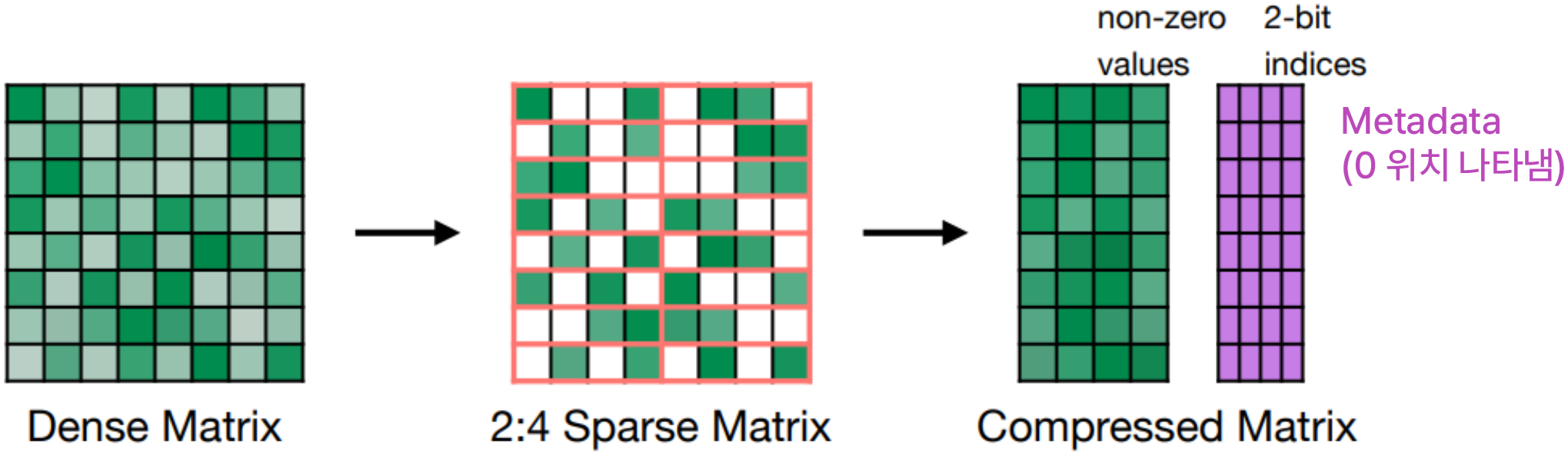




Pruning at Different Granularities

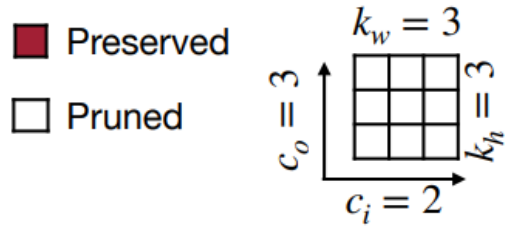
- **Pattern-based Pruning:** N:M sparsity
  - M개의 요소가 있을 때, N개의 요소를 Pruning 하는 기법
  - 일반적으로, 2:4 sparsity (50% sparsity) 사용
  - NVIDIA Ampere GPU에서 2× speed up

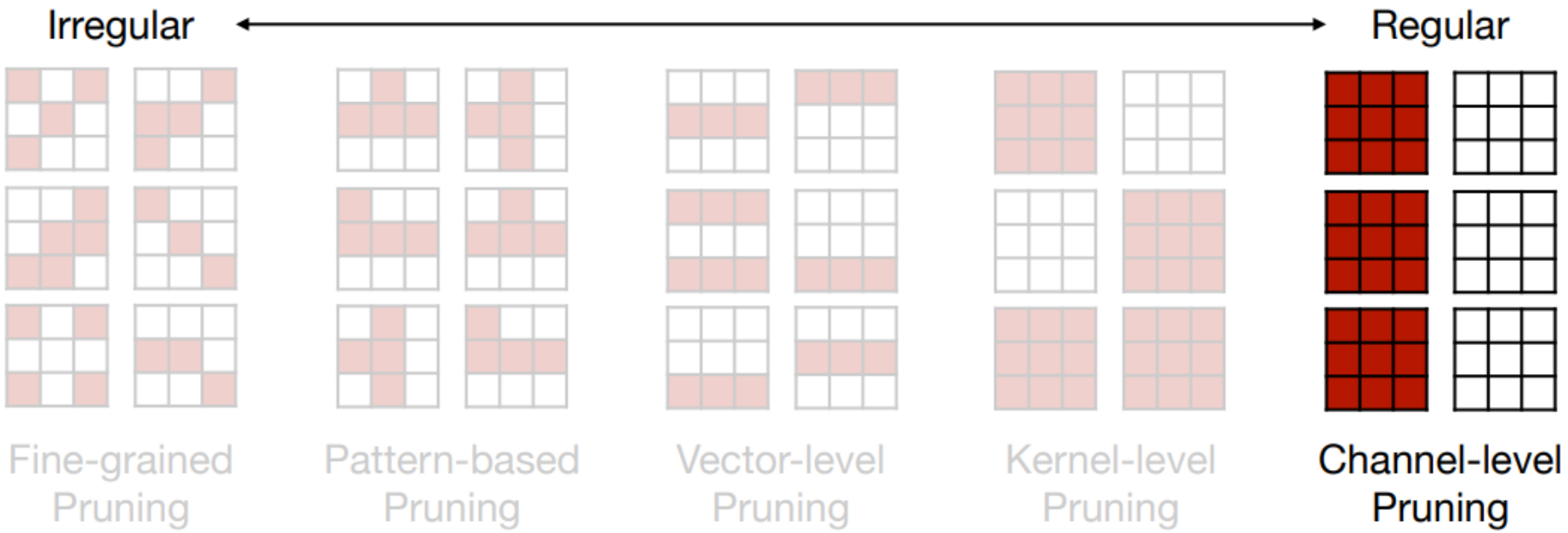
    e.g., RTX GeForce 30 series, A6000 ...



Pruning at Different Granularities

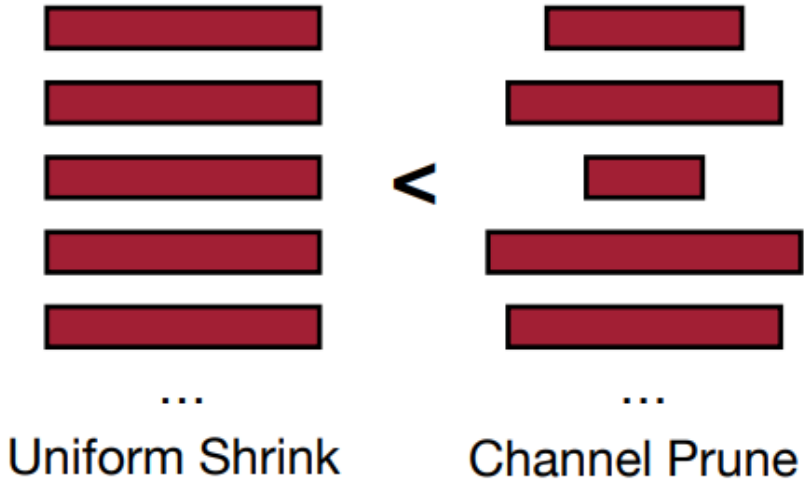
- **Channel-level Pruning:**
  - **Pro**: 네트워크의 채널 수를 줄이기 때문에 즉시 속도를 증가시킬 수 있음
    일반적인 CPU, GPU 에서 가속 쉬움
  - **Con**: 압축 비율이 작음

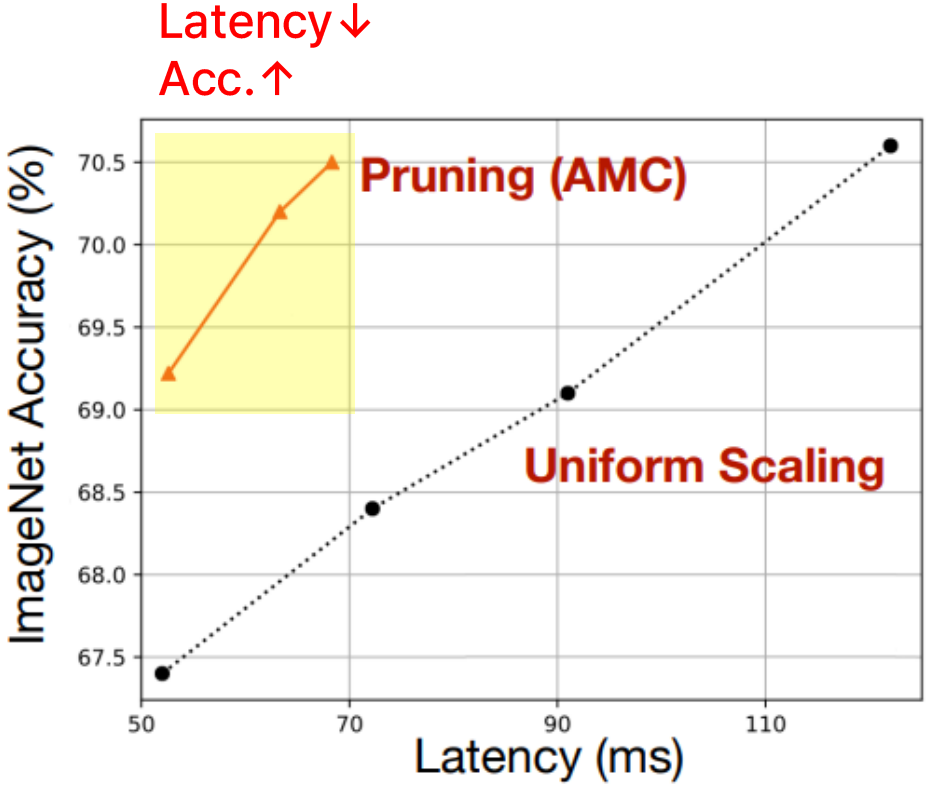




Pruning at Different Granularities

- **Channel-level Pruning:**
  - **Pro**: 네트워크의 채널 수를 줄이기 때문에 즉시 속도를 증가시킬 수 있음
    일반적인 CPU, GPU 에서 가속 쉬움
  - **Con**: 압축 비율이 작음

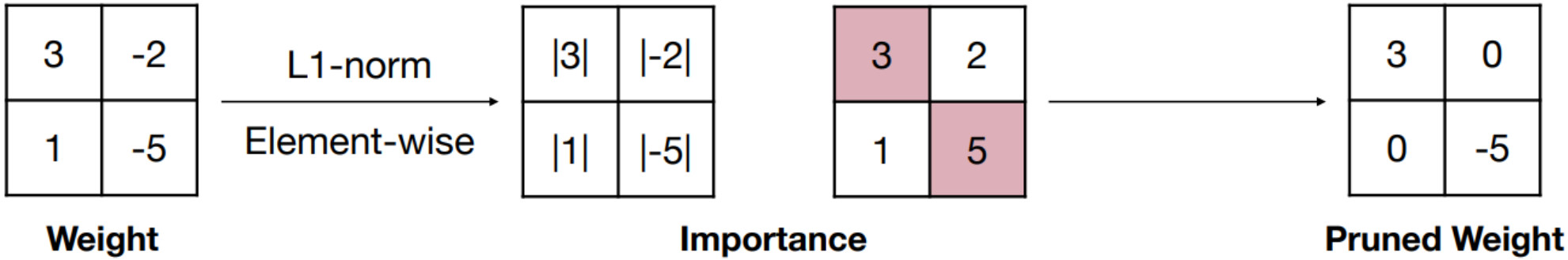




Magnitude-based Pruning

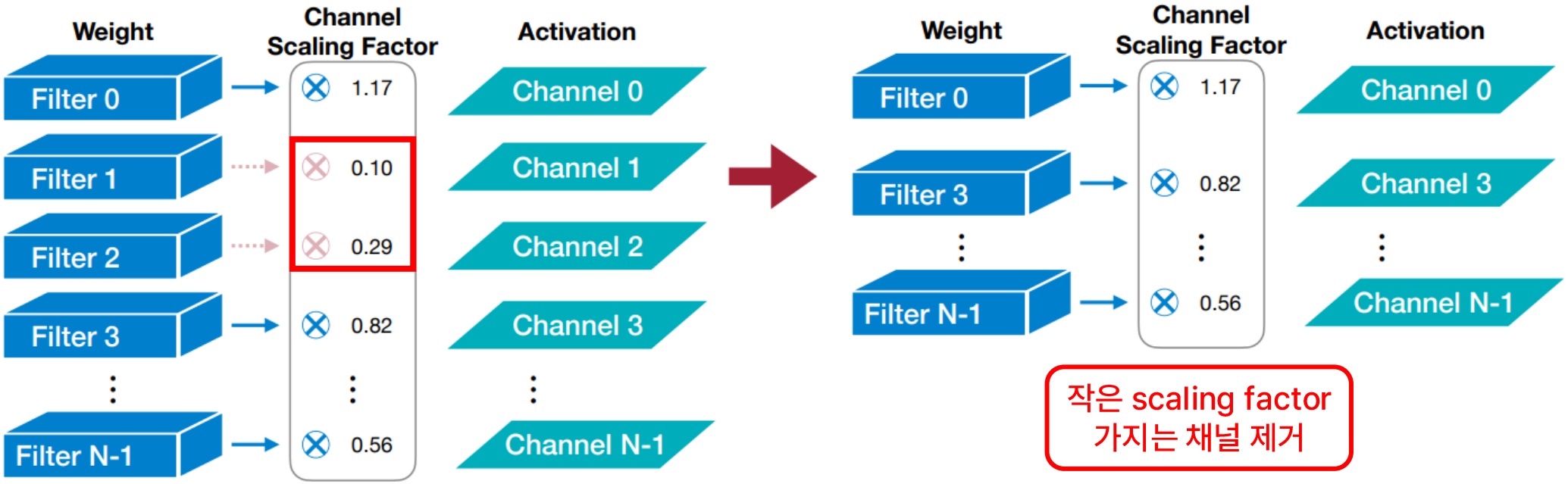
- **A heuristic pruning criterion**
  - **더 큰 절대값**을 가지는 파라미터를 **더 중요하다**고 가정함
  - 매우 간단한 방식이지만, 학계와 산업에서 잘 작동하는 방식
  - Element-wise, Row-wise 등 다양하게 적용 가능하며, 중요도는 L1, L2 norm으로 측정

$$\text{Importance} = |W|$$



Scaling-based Pruning

- **Pruning criterion for filter pruning**
  - Convolution layer의 필터 (i.e., output channel) 마다 scaling factor를 둠
  - Scaling factor는 학습 가능한 파라미터
  - BN 레이어의 scaling factor (γ) 재사용 가능

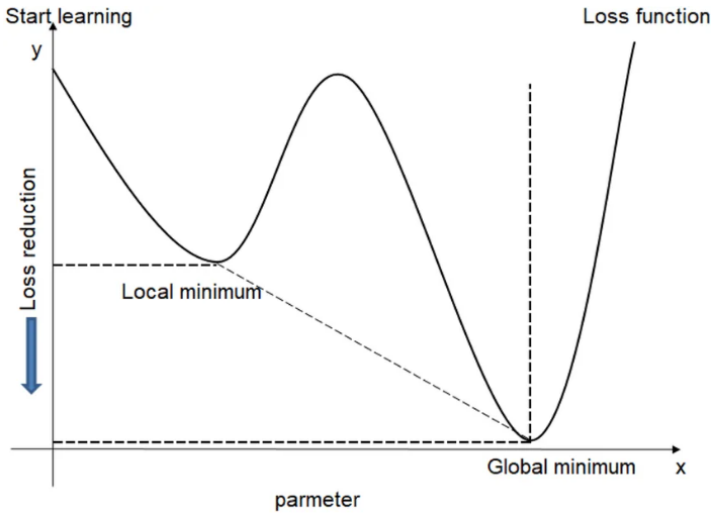


Second-Order-based Pruning

- **Minimize the error on loss function introduced by pruning synapses**
  - "Optimal Brain Damage" 논문에서는 loss가 quadratic하다고 가정, 마지막 항 무시함
  - 모델 학습이 수렴했기 때문에 first-order 사라짐
  - 각각의 파라미터들이 독립적이라고 가정하고, cross term 무시함

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

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



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

* h_{ii} is non-negative

Percentage-of-Zero-based Pruning

- Activation 값을 이용하여 pruning하는 방식 (weight 자체의 값을 이용하지 않음)
- Average Percentage of Zero activations (APoZ)를 뉴런의 중요도로 사용함
- APoZ 가 작을수록, 더 중요한 뉴런

Output Activations

Batch = 2

Channel = 3 (Width = 4, Height = 4)

0	0.1	0.5	1
1.2	0.6	0.3	0.2
0	0.5	**0**	0.3
0.2	**0**	**0**	0.8

0.1	0.5	**0**	**0**
0.2	0.3	**0**	1
0.1	**0**	**0**	0.5
0.1	0.6	0.7	0.1

0	**0**	0.8	**0**
0.7	**0**	0.6	0.1
1.2	1	**0**	0.2
0.5	**0**	0.3	0.5

Channel = 3 (Width = 4, Height = 4)

0.5	**0**	0.2	0.1
0	0.2	1.2	**0**
1.2	**0**	0.2	0.3
0.2	0.4	**0**	**0**

0.1	0.5	**0**	**0**
0	0.8	**0**	1
0.1	**0**	0.1	1.0
0.2	**0**	1.0	**0**

0	0.8	0.1	**0**
0.2	**0**	**0**	0.3
0	0.4	**0**	0.5
0.2	**0**	0.3	**0**

Average Percentage of Zeros (APoZ)

$$= \frac{5+6}{2 \cdot 4 \cdot 4} = \frac{11}{32}$$ Channel 0

$$= \frac{5+7}{2 \cdot 4 \cdot 4} = \frac{12}{32}$$ Channel 1

$$= \frac{6}{2 \cdot 4 \cdot 4} = \frac{14}{32}$$ Channel 2

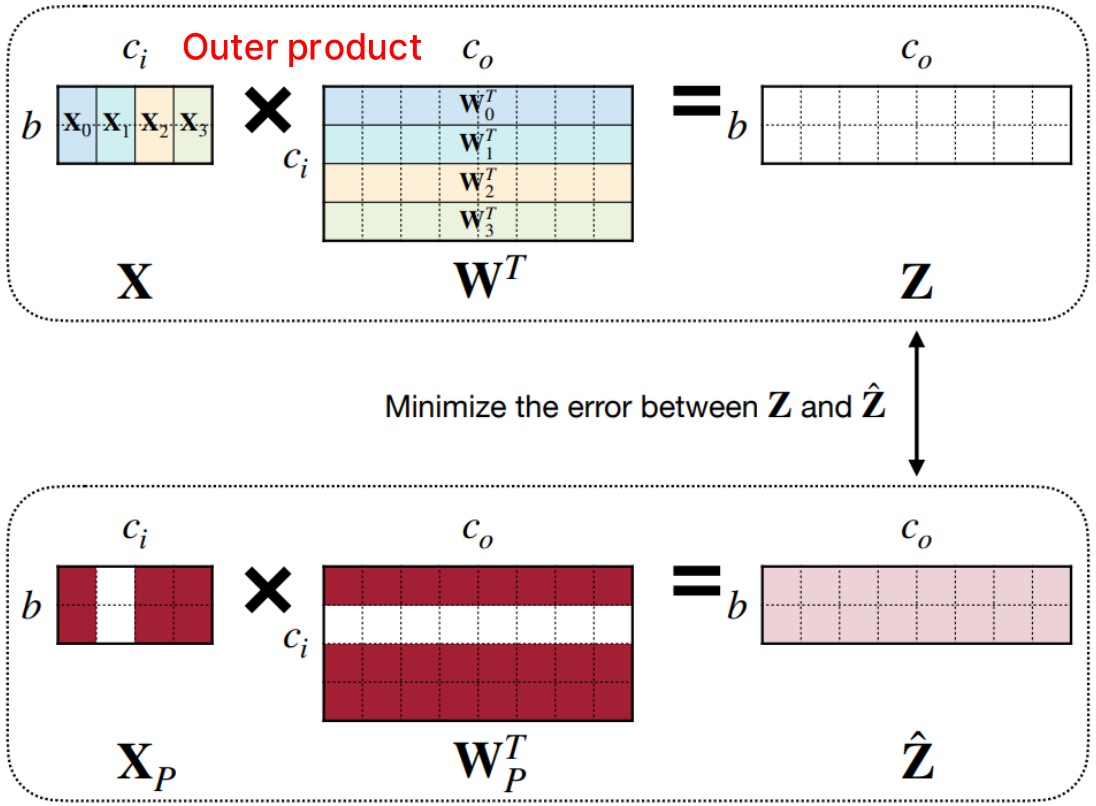
Regression-based Pruning

- **Minimize reconstruction error of the corresponding layer's outputs**
  - 모델 전체의 error를 pruning에 사용하지 않고, 각 레이어의 reconstruction error를 사용함

$$\arg\min_{\mathbf{W},\,\beta} \|\mathbf{Z} - \hat{\mathbf{Z}}\|_F^2 = \|\mathbf{Z} - \sum_{c=0}^{c_i-1} \beta_c \mathbf{X}_c \mathbf{W}_c^T\|_F^2$$

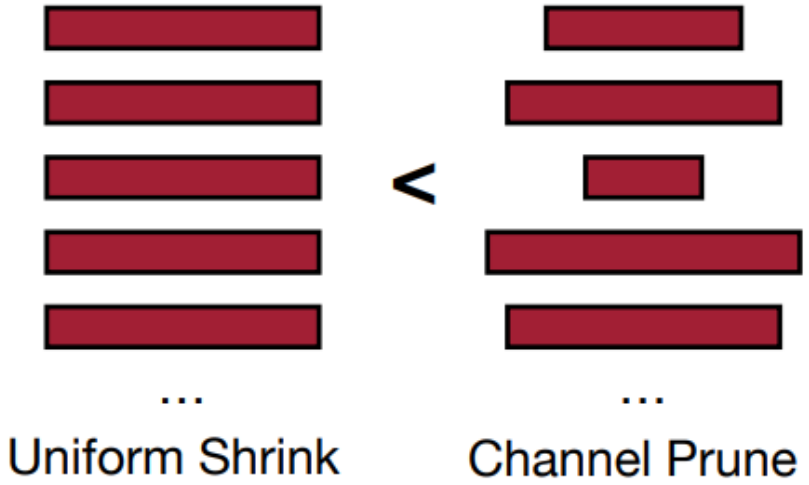
$$\text{subject to} \quad \|\beta\|_0 \le N_c$$

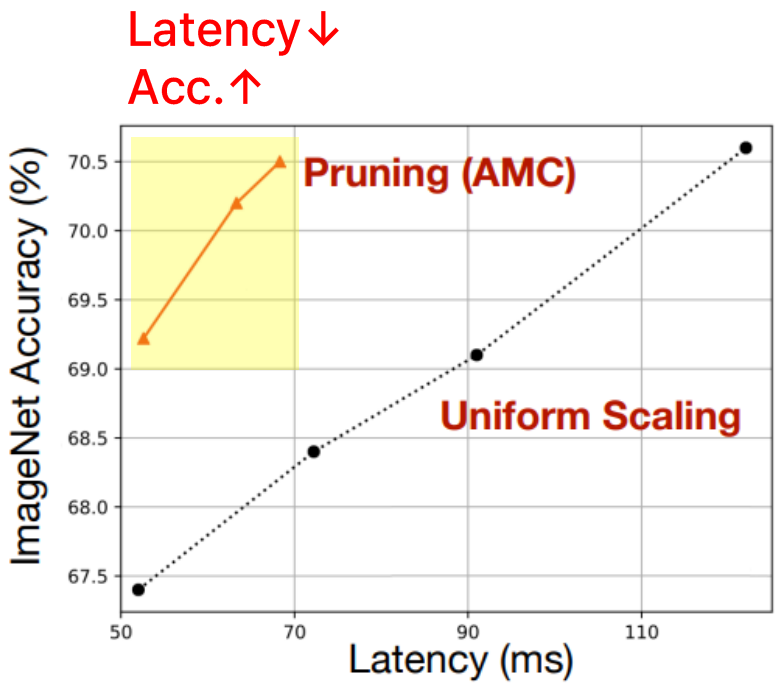
Iterate
- Fix $\mathbf{W}$, solve β for channel selection
- Fix β, solve $\mathbf{W}$ to minimize reconstruction error



Finding Pruning Ratios

- **Analyze the sensitivity of each layer**
  - 모델 전체에 같은 pruning ratio를 사용하는 것보다 각 레이어마다 지정해주는 것이 더 성능 좋음
  - 각 레이어는 pruning에 각기 다른 민감도를 가지고 있음
  - Per-layer pruning을 위해 sensitivity 분석

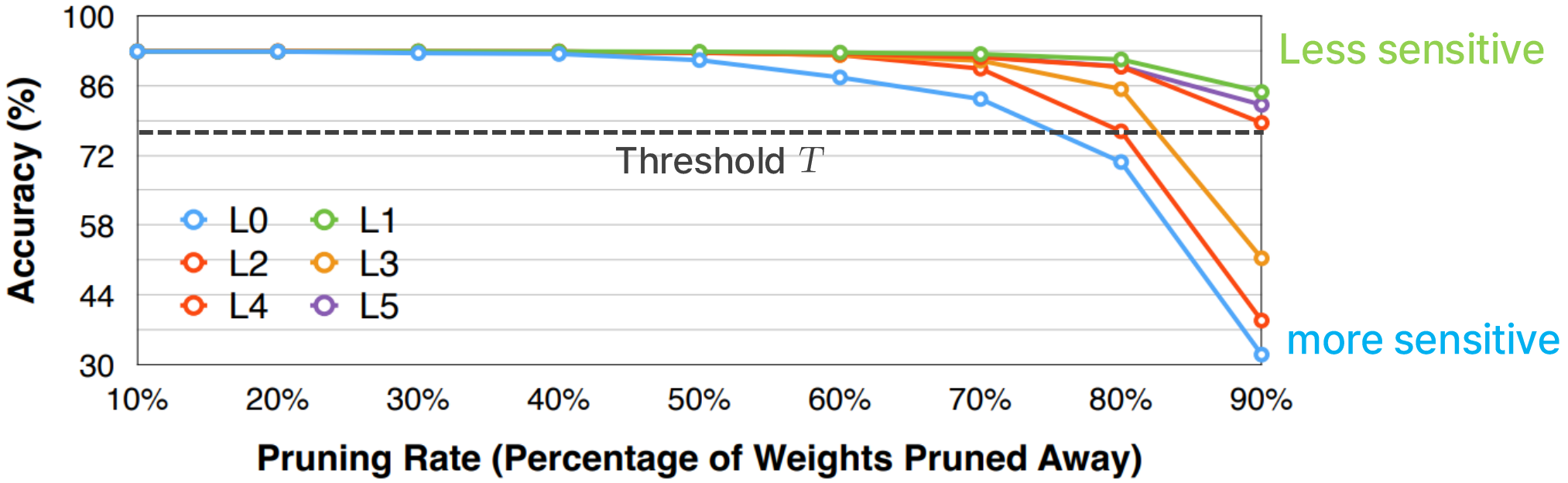




Finding Pruning Ratios

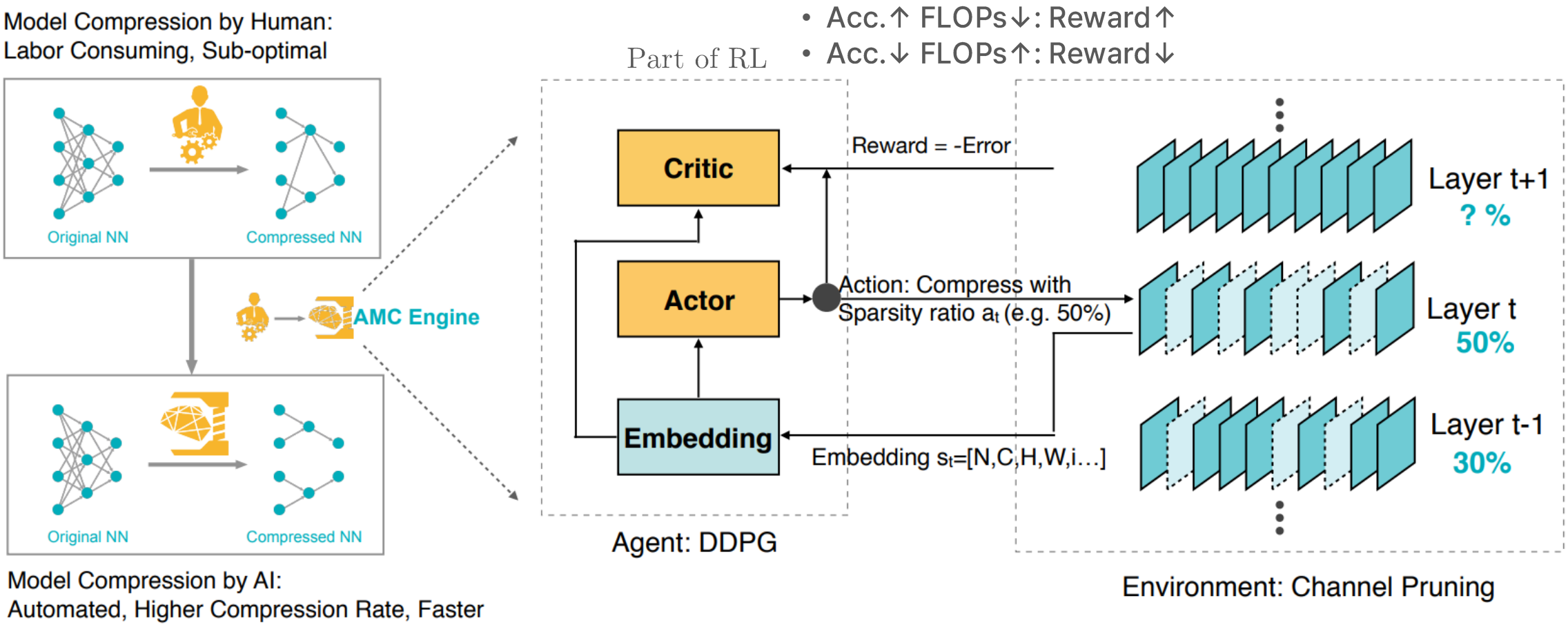
- **Analyze the sensitivity of each layer**
  - 모델의 전체 레이어에 대해 pruning rate를 조정하며 accuracy가 떨어지는 정도를 분석
  - Threshold T 를 정하여, 각 레이어에 대한 pruning rate 구함

➡ 최적의 값 아님. 레이어 간의 상호작용을 고려하지 않음



AMC: AutoML for Model Compression

- **Pruning as a reinforcement learning problem**



AMC: AutoML for Model Compression

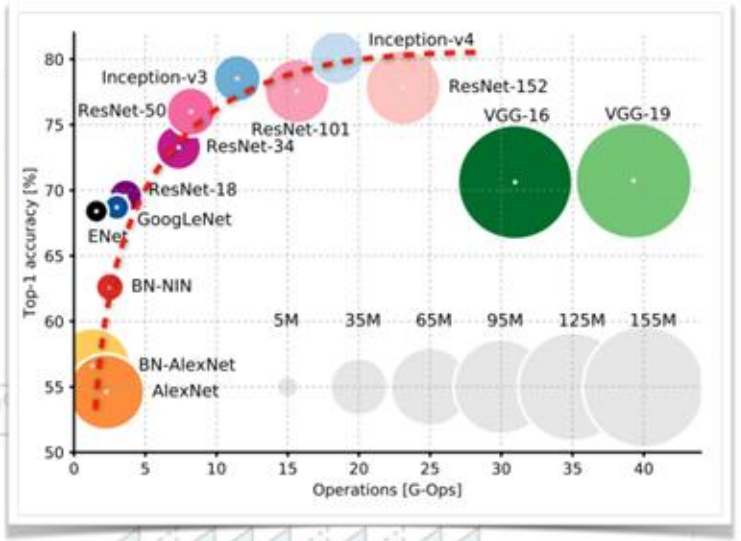
- **Pruning as a reinforcement learning problem**

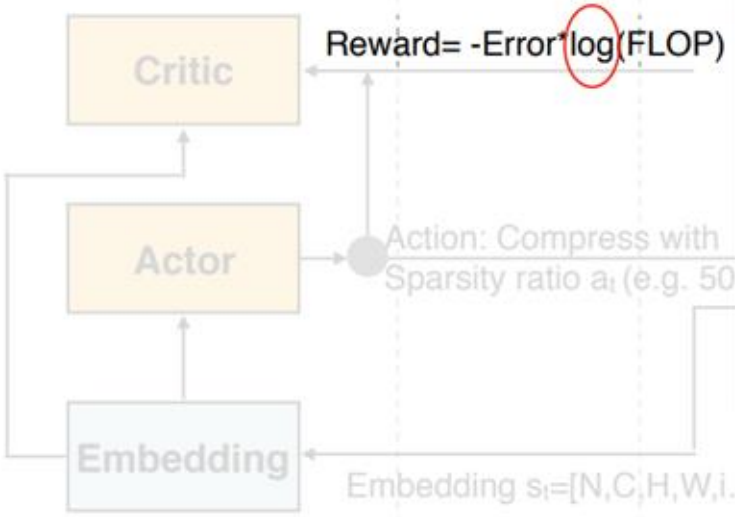
Model Compression by Human:
Labor Consuming, Sub-optimal

AMC Engine

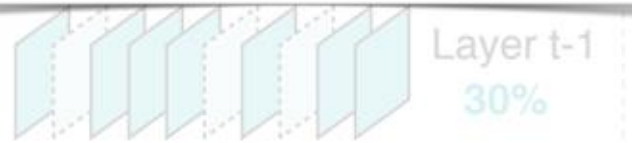
Model Compression by AI:
Automated, Higher Compression Rate, Faster

Acc.-FLOPs curve
≒ Log function





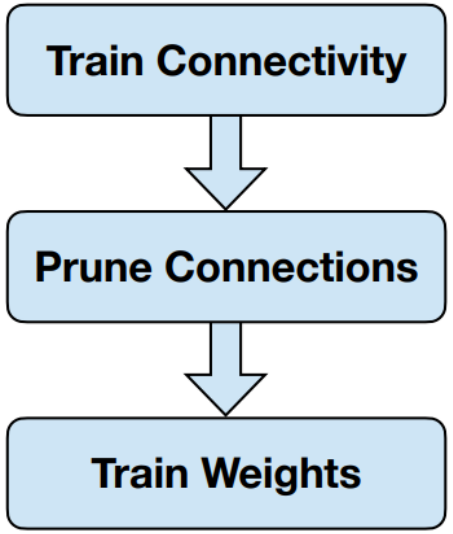
Agent: DDPG

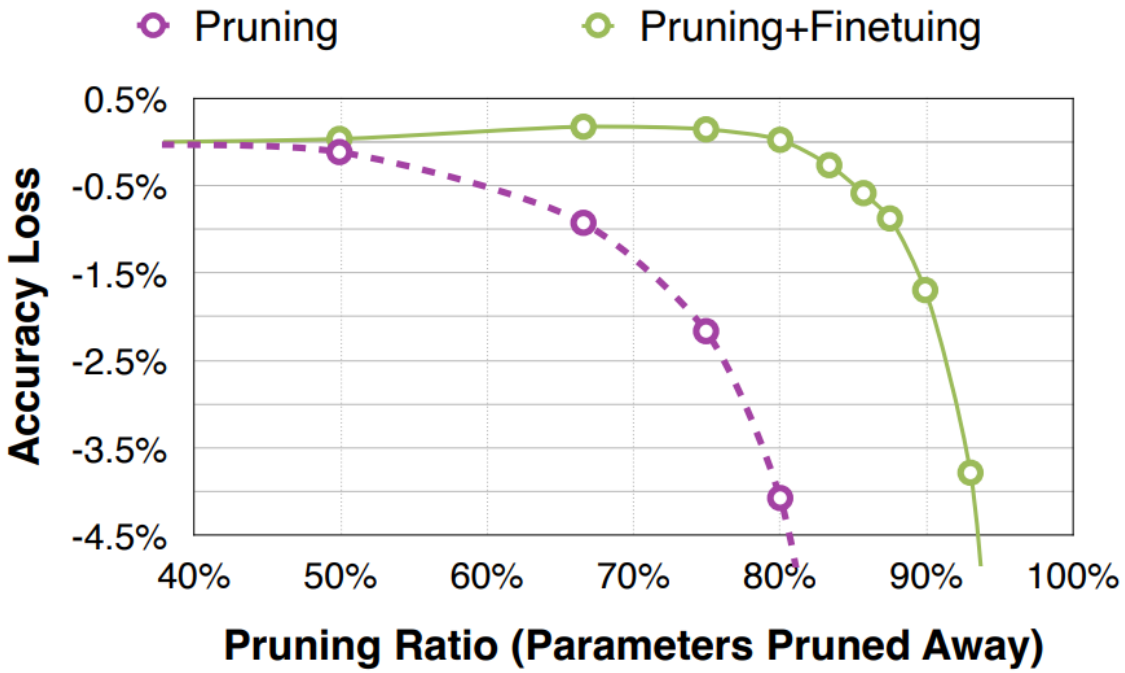


Environment: Channel Pruning

Fine-tuning Pruned Neural Networks

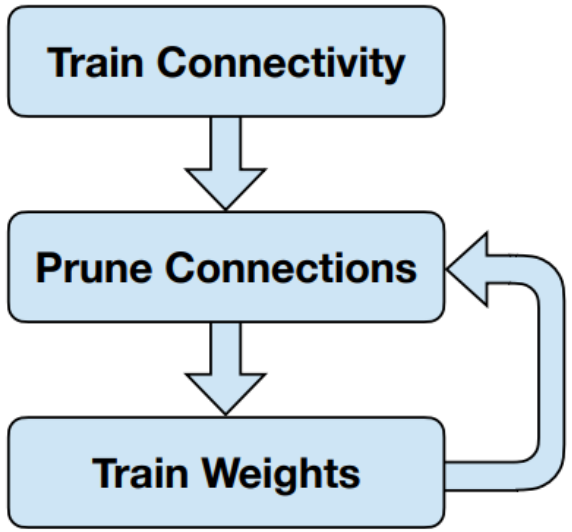
- Pruning 이후, 모델 크기가 줄어듦
- Fine-tuning을 통해 accuracy 성능을 회복시키고, pruning ratio를 더 높일 수 있음
- Learning rate 는 기존의 값보다 1/10 or 1/100 수준으로 설정

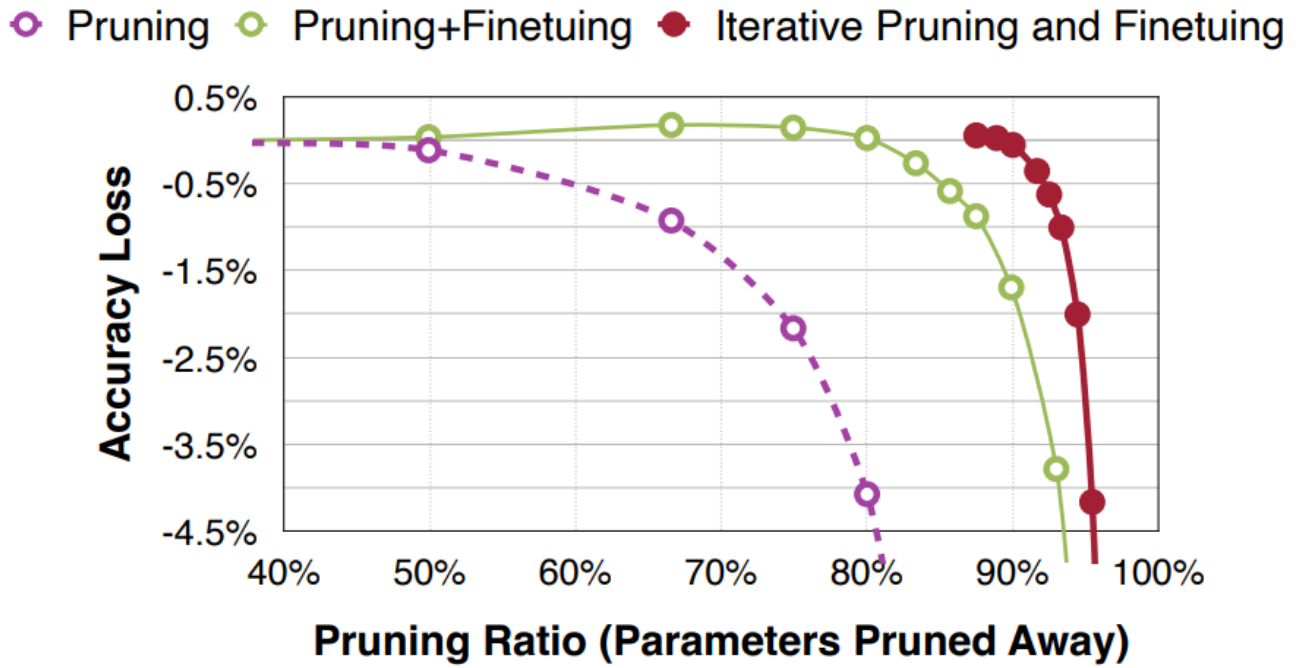




Iterative Pruning

- 한 번에 큰 pruning ratio를 적용하는 것보다, **pruning-training** 반복하는 것이 △accuracy를 작게 유지하면서도 더 높은 pruning ratio를 적용할 수 있음

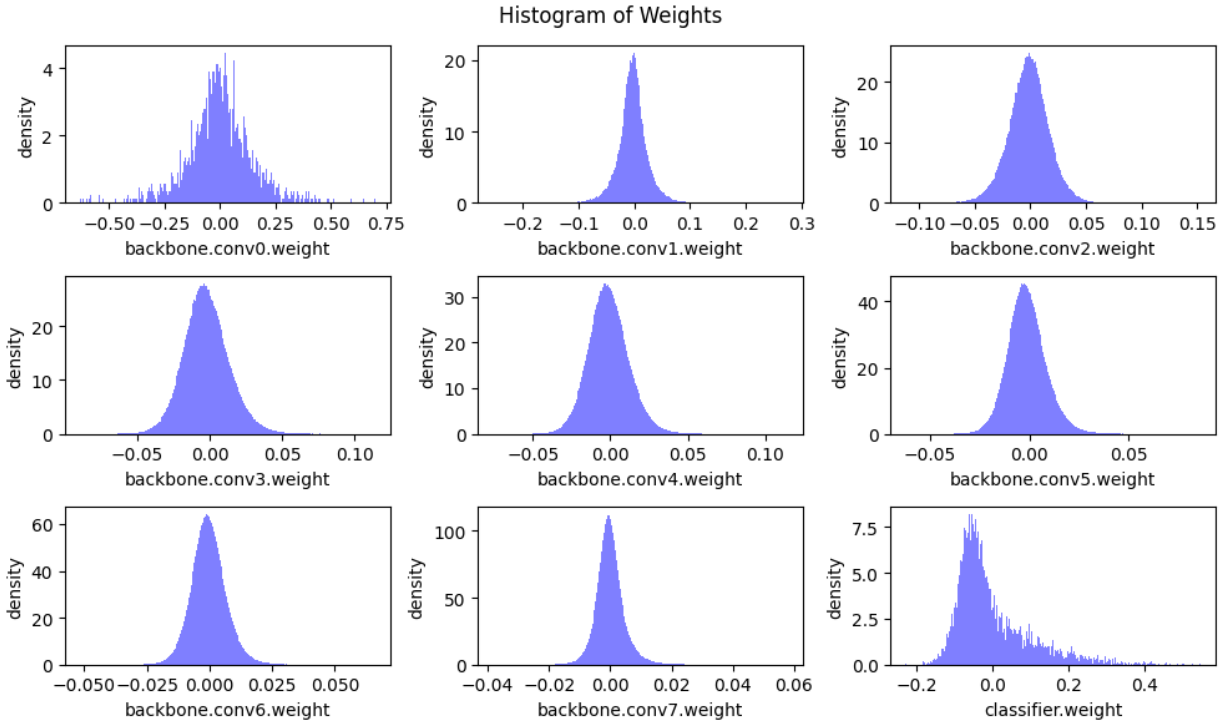




Magnitude-based Pruning with PyTorch

- Dataset: **CIFAR10** | Model: **VGGNet**

Dense Model

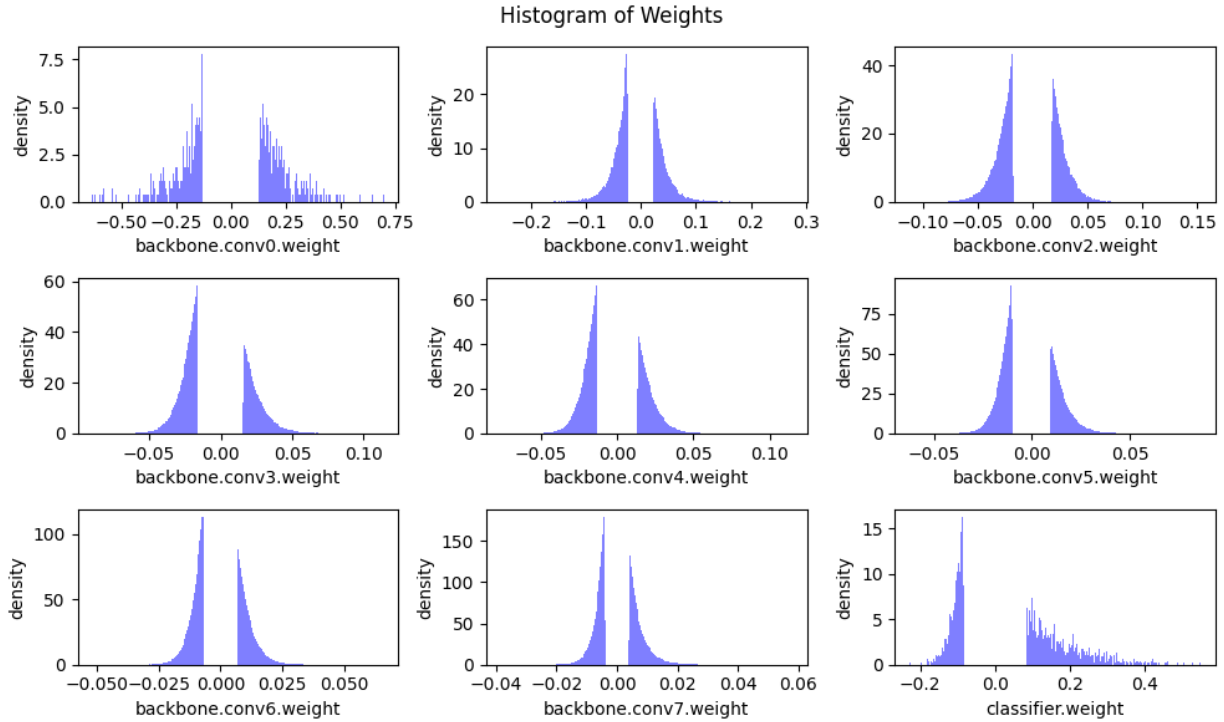


Top-1 Accuracy: **92.31%**
Model size: **35.20**MB

Magnitude-based Pruning with PyTorch

- Dataset: **CIFAR10** | Model: **VGGNet**

70% Sparse Model

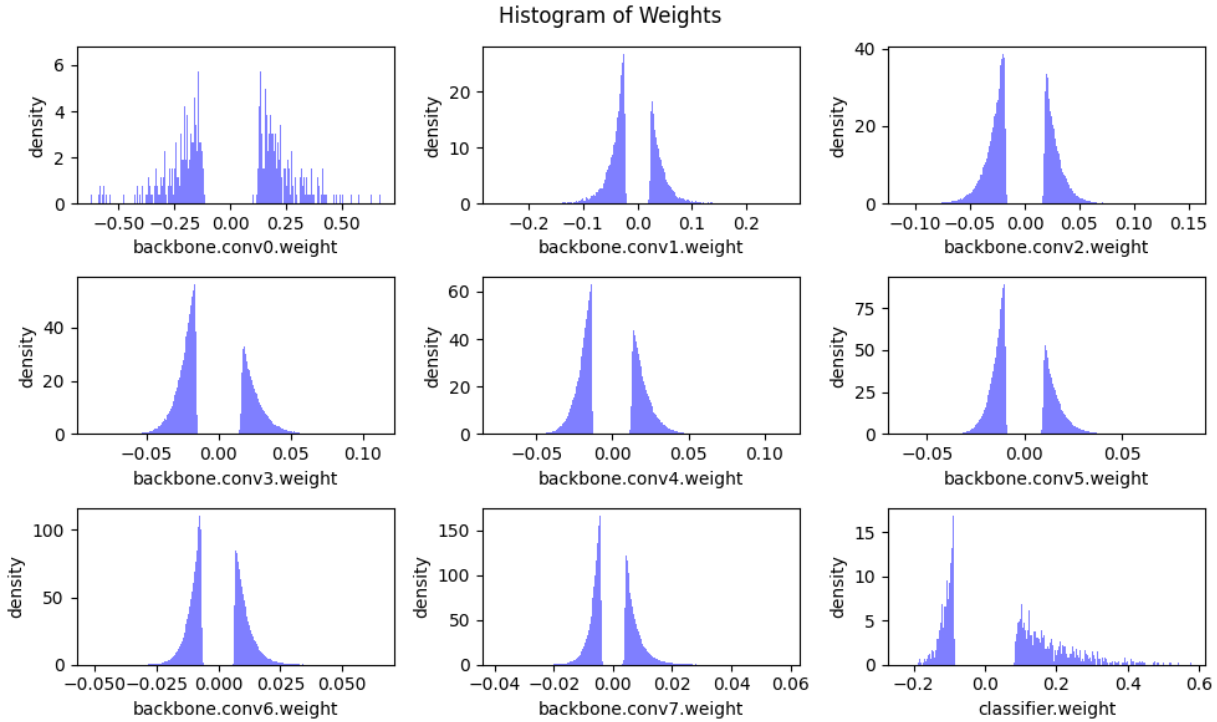


Top-1 Accuracy: **75.47%** ⇨ -16.84%
Model size: **10.58**MB ⇨ 3.33× smaller

Magnitude-based Pruning with PyTorch

- Dataset: **CIFAR10** | Model: **VGGNet**

Fine-tuned Pruned Model

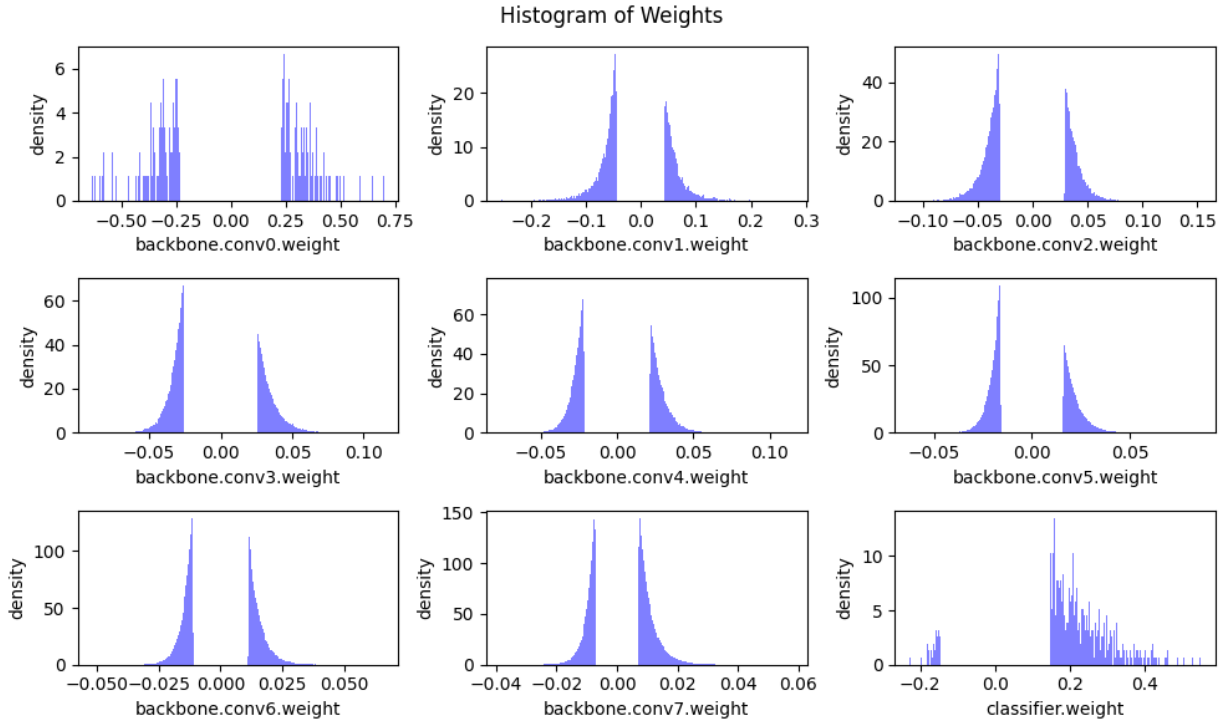


Top-1 Accuracy: **92.79%** ⇨ +0.48%
Model size: **10.58**MB ⇨ 3.33× smaller

Magnitude-based Pruning with PyTorch

- Dataset: **CIFAR10** | Model: **VGGNet**

90% Sparse Model

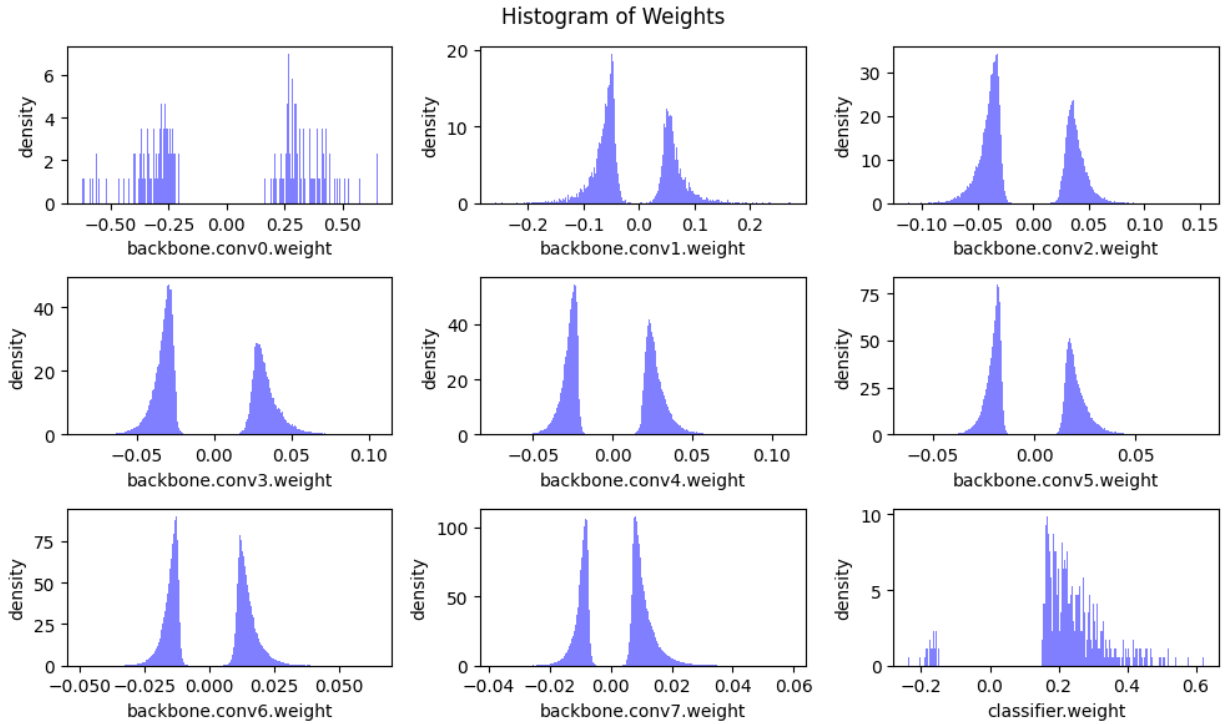


Top-1 Accuracy: **15.56**% ⇨ -76.75%
Model size: **3.54**MB ⇨ 9.95× smaller

Magnitude-based Pruning with PyTorch

- Dataset: **CIFAR10** | Model: **VGGNet**

Fine-tuned Pruned Model



Top-1 Accuracy: **91.53%** ⇨ -0.78%
Model size: **3.54**MB ⇨ 9.95× smaller

Thank You