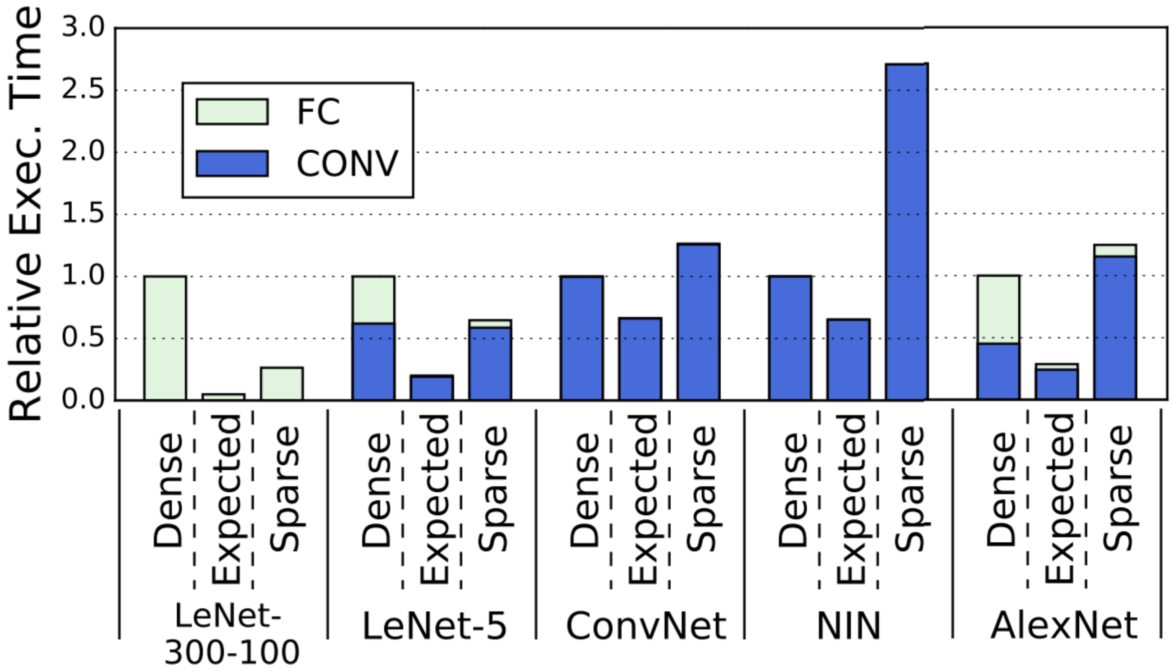
Scalpel: Customizing DNN Pruning to the Underlying Hardware Parallelism

Jiecao Yu, Andrew Lukefahr, David Palframan, Ganesh Dasika, Reetuparna Das , Scott Mahlke

2019-07-18

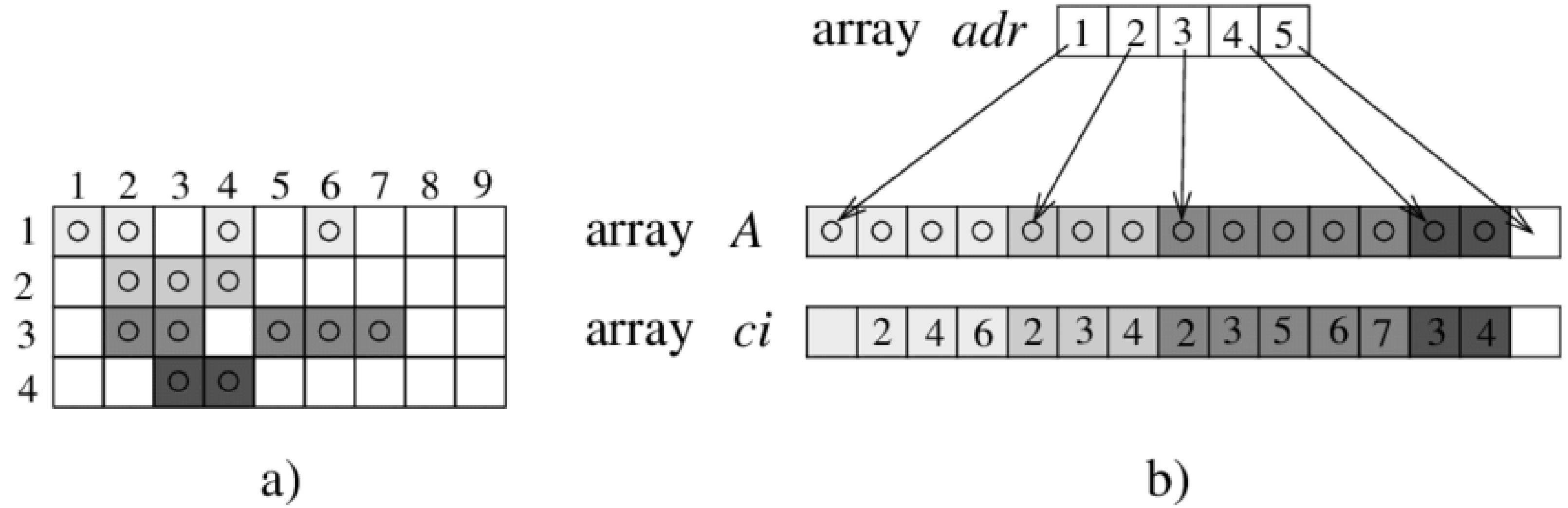
Park Sang Soo

Introduction: Sparse matrix



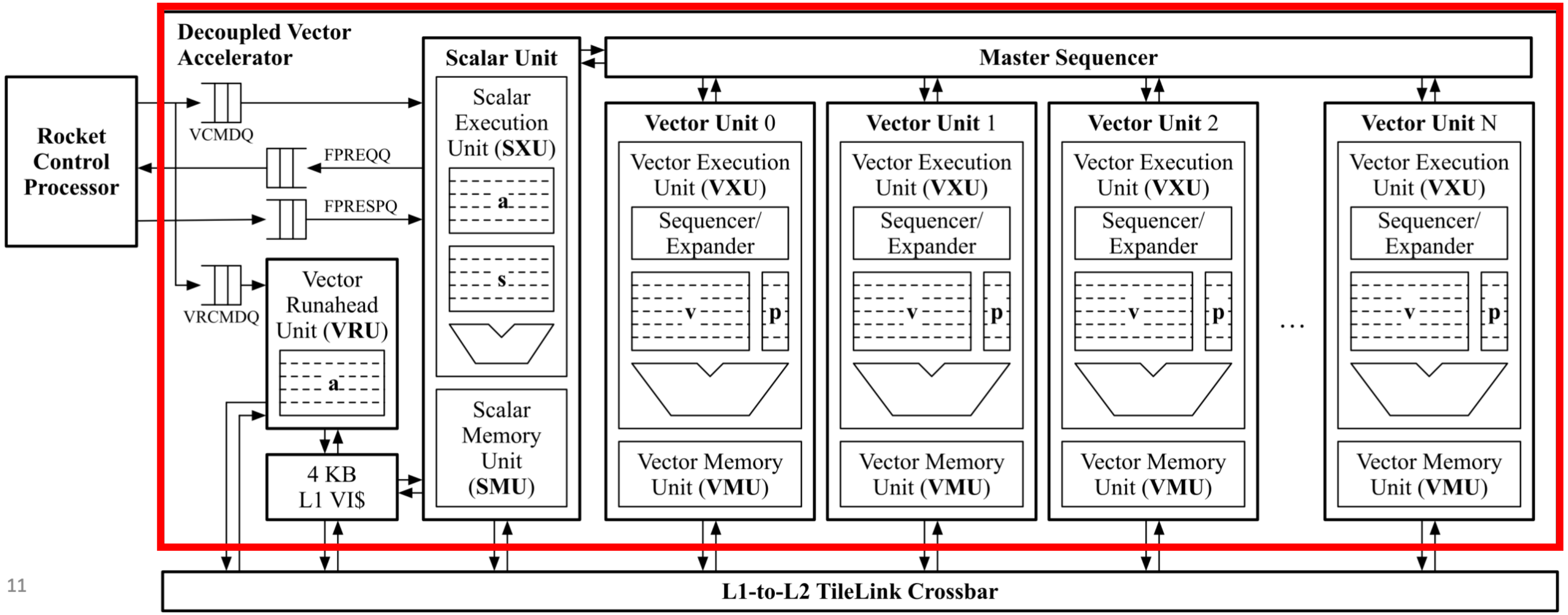
- Weight pruning 방법들은 pruning 후에 평균 80% 정도의 weight가 없어졌지만, 실제로 퍼포먼스는 떨어지는 경우가 많음

Introduction: Sparse matrix



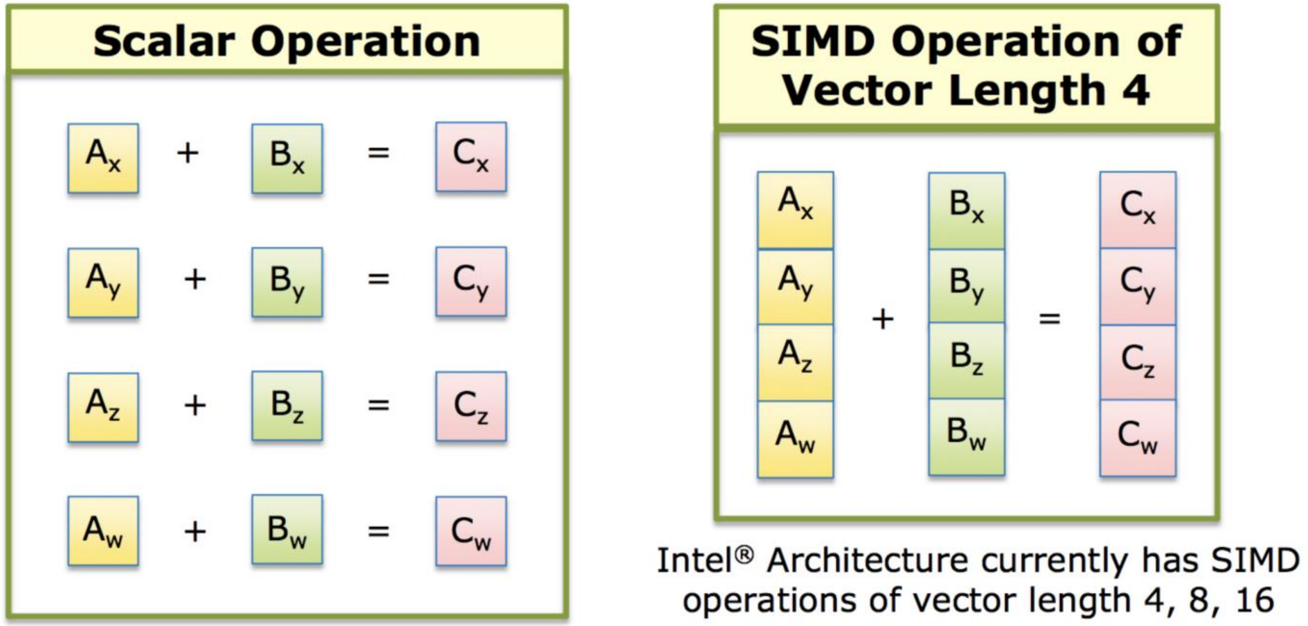
- Pruning된 네트워크의 행렬 곱은 sparse하게 되기 때문에 남은 weight들은 sparse matrix format으로 저장
- Sparse weight matrix는 기존의 dense matrix가 가지고 있던 일정한 구조를 잃게 되며, 곱을 행할 때 sparse format을 디코드하기 위한 추가적인 연산이 필요

Introduction: SIMD



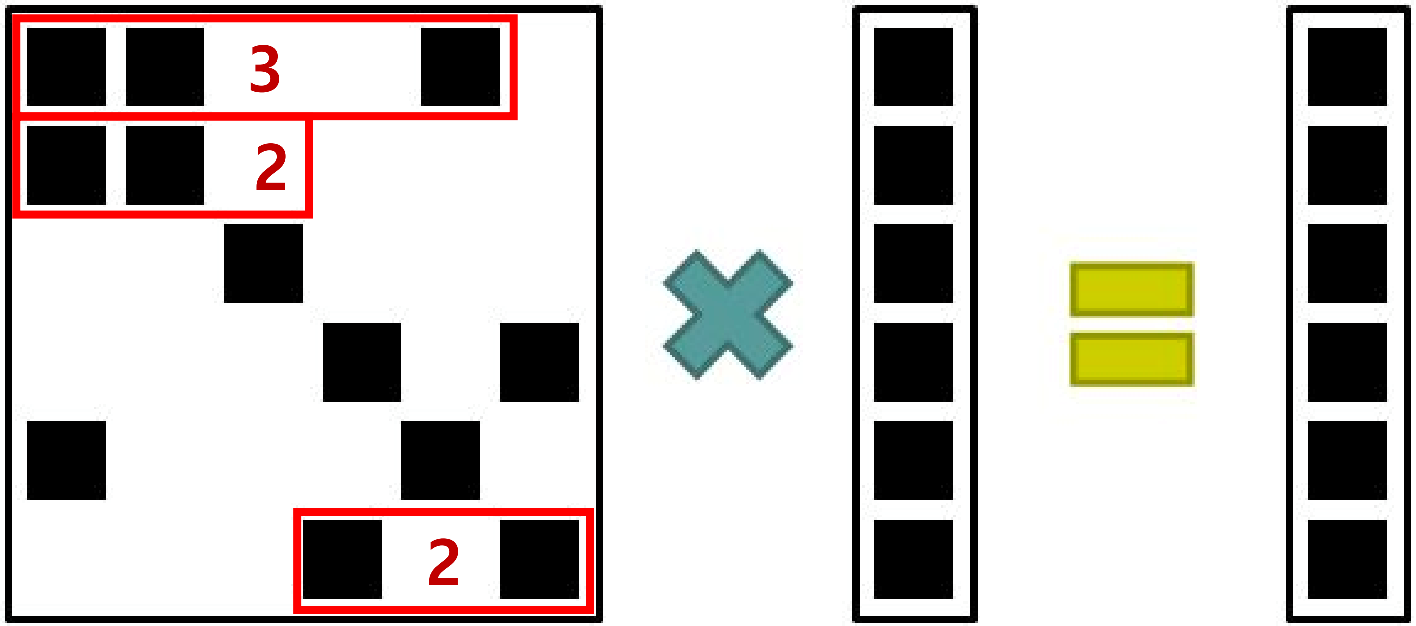
- 프로세서 내부에서는 벡터 연산을 위한 **Single Instruction Multiple Data (SIMD)** 연산기가 있음
- SIMD 연산기를 사용하여 딥러닝의 곱셈과 덧셈을 병렬화

Introduction: SIMD



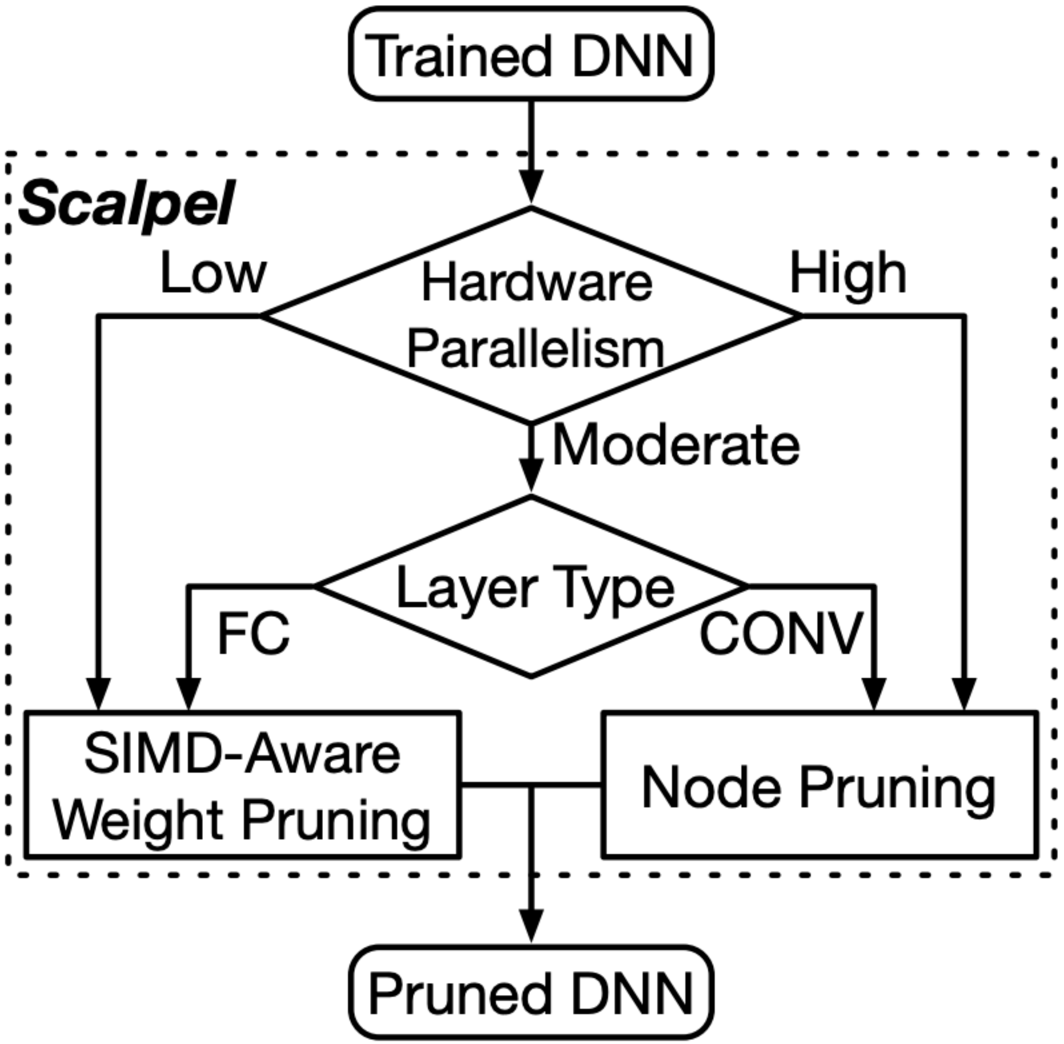
- SIMD 연산기를 사용하여 여러개의 곱셈/덧셈을 병렬로 계산
- 알고리즘 특성으로 인하여 연산기를 다 사용하지 못하는 경우 발생

Introduction: SIMD

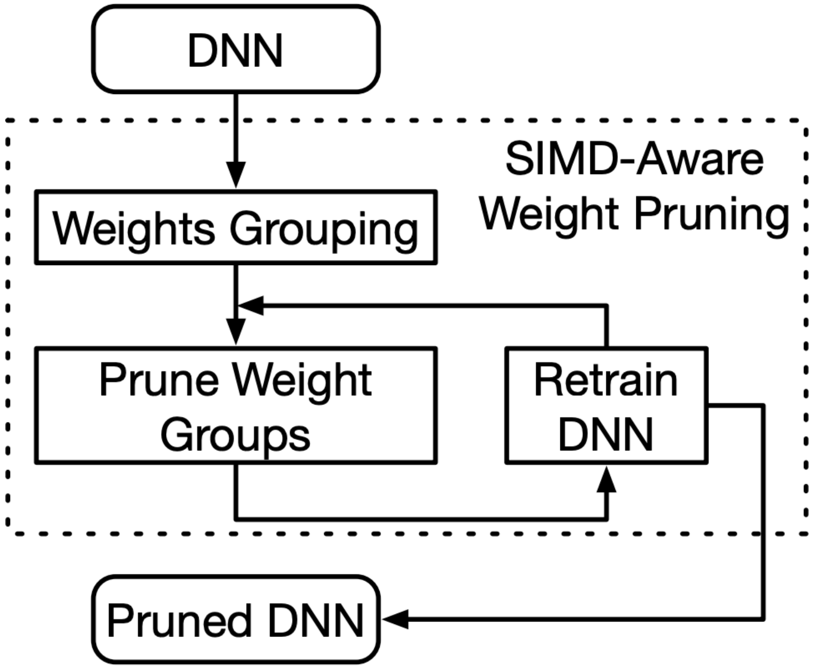


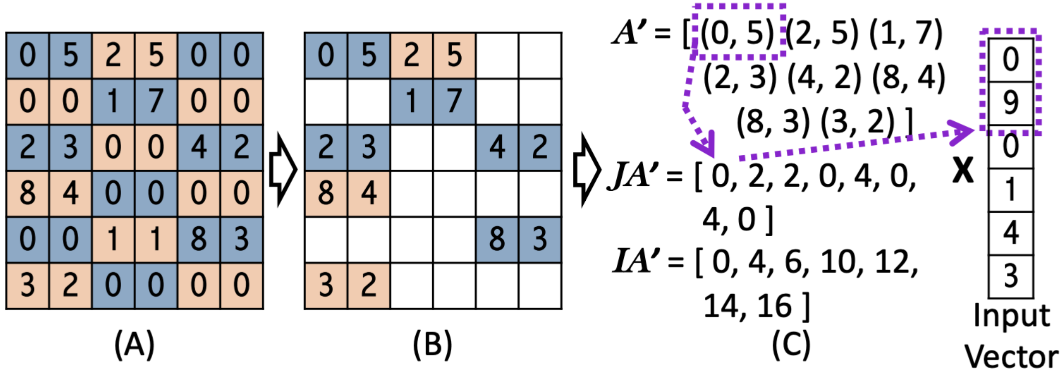
- 한번에 **최대 4개의 곱셉/덧셈을 계산**할 수 있는 SIMD 연산기
- 첫번째 경우 (3개 병렬 계산), 두번째, 마지막 경우 (2개 병렬)

Pruning technique



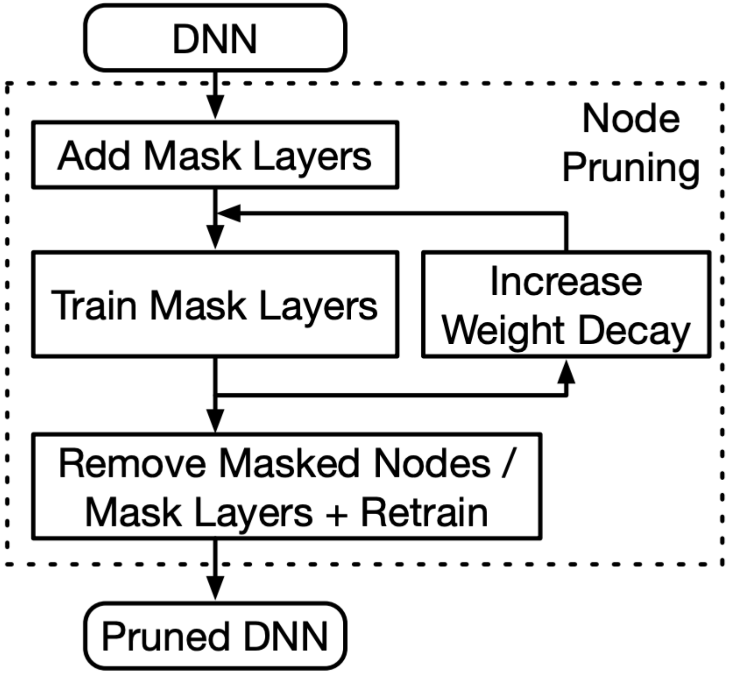
SIMD-Aware Pruning





- RMS를 사용하여 **연산 능력과 동일한 개수의 aligned 그룹** 생성
- Threshold보다 작은 그룹은 제거, Dropout을 사용하여 overfitting 방지 [Han]

Node Pruning (=Channel Pruning)



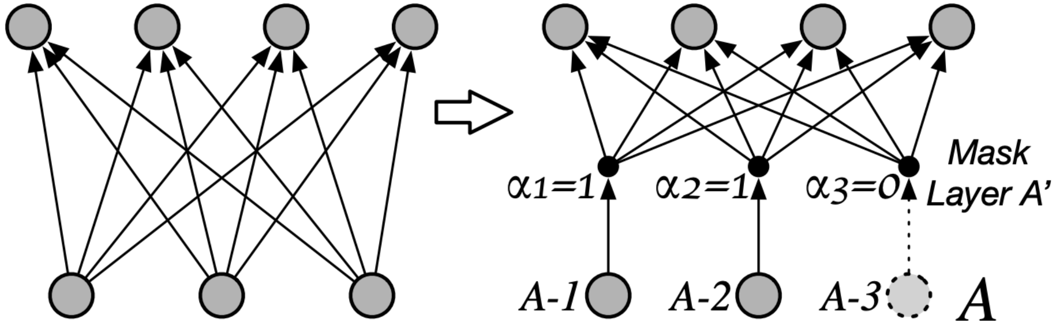


Figure 12: Mask layers. Node *A-3* with $\alpha_3 = 0$ can be removed. The whole mask layer A' will be removed after pruning all redundant nodes.

- Weight를 Pruning하는 것이 아니라 Node를 Pruning
- 마스크를 사용하여 중요하지 않는 노드를 찾고, 그 출력을 막음
- FC에서는 한 뉴런, Conv에서는 피쳐맵을 하나의 노드로 간주 (Dense 구조 유지)

Node Pruning (=Channel Pruning)

$$\alpha_i|_k = \begin{cases} 1, & T + \varepsilon \leqslant \beta_i|_k \\ \alpha_i|_{k-1}, & T \leqslant \beta_i|_k < T + \varepsilon \\ 0, & \beta_i|_k < T \end{cases}$$

$$R_{i,\,L1} = \lambda|\beta_i| = \lambda\beta_i$$

- 각 노드 마스크는 α (바이너리), β (0과 1사이의 실수 값) 두개의 파라미터
- □, □□는 레이어 □의 원래 출력, 마스크 □□에 의해 마스킹 된 출력
- $y'_i = \alpha_i * yi$□□노드 제거 (α=0), Dropout을 사용하여 overfitting 방지 [Han] □

Experimental Result: SIMD-Aware

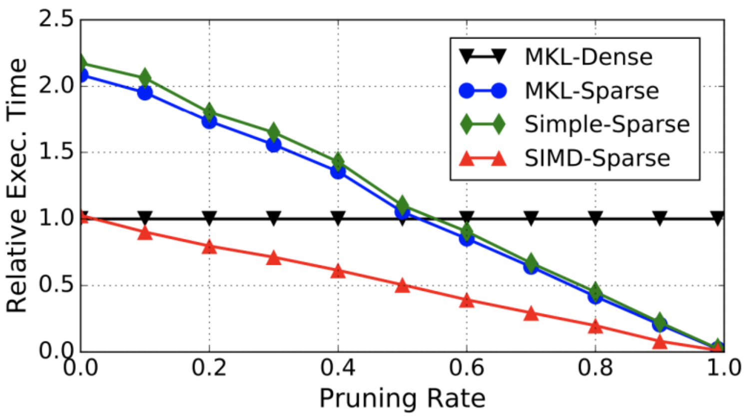


Figure 13: Relative execution time for sparse matrix-vector multiplication (FC layers) on Intel Core i7-6700. The matrix size is 4096 x 4096 and the vector size is 4096. MKL-Dense/Sparse show the results of dense and sparse weight matrix with the Intel MKL library.

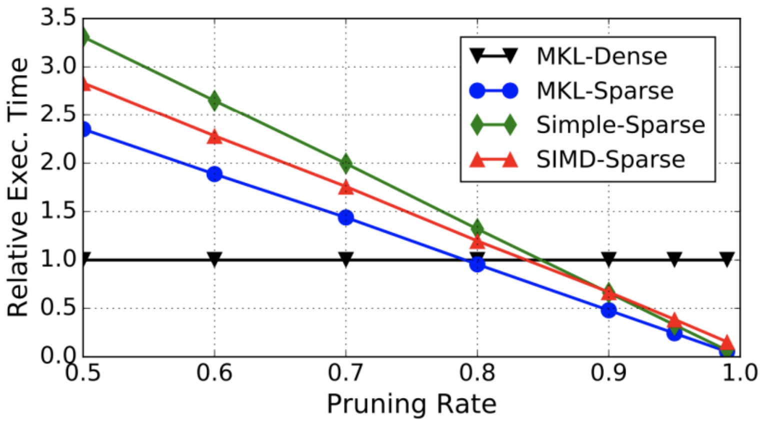


Figure 14: Relative execution time for sparse matrix-matrix multiplication (CONV layers) on Intel Core i7-6700. The weight matrix and input matrix have the size of 128 x 1200 and 1200 x 729, respectively.

- SIMD-Spare (SIMD-Aware), Simple-Sparse (CSR)
- Intel's BALS (Math Kernel Lib, MKL)

Experimental Result: All Pruning

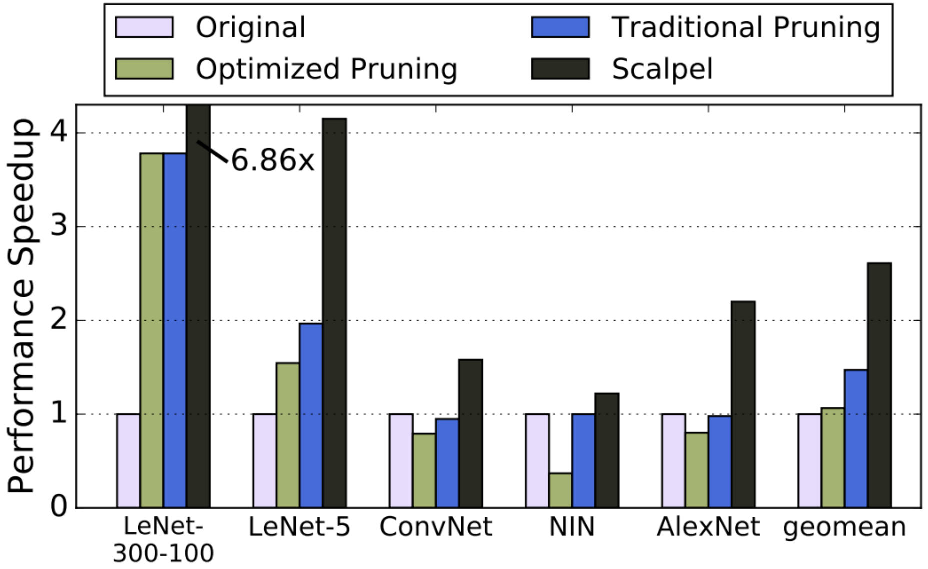


Figure 18: Relative performance speedups of the original models, traditional pruning, optimized pruning and Scalpel on Intel Core i7-6700 CPU.

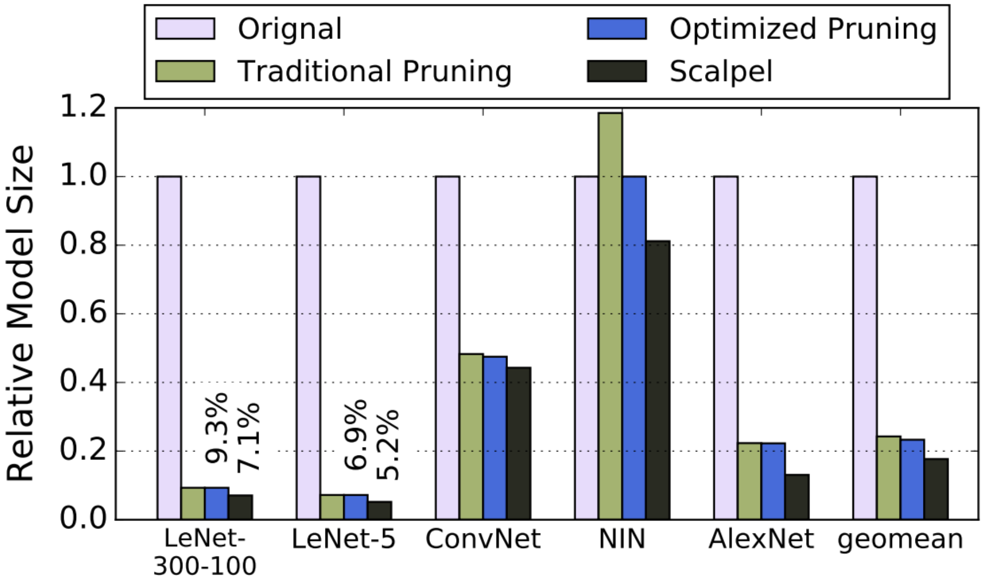


Figure 19: Relative model sizes of the original models, traditional pruning, optimized pruning and Scalpel for Intel Core i7-6700 CPU.

- Traditional (DeepCompression), Optimized (Channel Pruning)
- Scapel (SIMD-Aware + Node Pruning)

Experimental Result: Accuracy and Time

Networks	Num of Layers		Test Dataset	Error Rate
	CONV	FC		
LeNet-300-100	0	3	MNIST	1.50%
LeNet-5	2	2		0.68%
ConvNet	3	1	CIFAR-10	18.14%
NIN	9	0		10.43%
AlexNet	5	3	ImageNet	19.73% (top-5)

Hardware	DNNs	Speedup	Relative Size
Micor-controller	LeNet-300-100	9.17x	6.93%
	LeNet-5	3.51x	6.72%
	ConvNet	1.38x	40.95%
CPU	LeNet-300-100	6.86x	7.08%
	LeNet-5	4.15x	5.20%
	ConvNet	1.58x	44.28%
	NIN	1.22x	81.16%
	AlexNet	2.20x	13.06%
GPU	LeNet-300-100	1.08x	66.83%
	LeNet-5	1.59x	11.67%
	ConvNet	1.14x	45.40%
	NIN	1.17x	81.16%
	AlexNet	1.35x	76.52%

- AlexNet in DeepCompression: Top5 (19.846%), 11% (9x) compressed
- LeNet-300-100/LeNet-5: Top1 (1.58/0.74%), 3.1% (32x) compressed

Experimental Result: Conclusion

Table 4: Percentage of nodes removed by node pruning in each layer. Output layers are not included.

DNNs	Percentage of Nodes Removed in Each Layer
LeNet-300-100	31% (fc1)- 32% (fc2)
LeNet-5	50% (conv1)- 68% (conv2)- 65% (fc3)
ConvNet	28% (conv1)- 25% (conv2)- 49% (conv3)
NIN	28% (conv1)- 20% (cccp1)- 5% (cccp2)- 2% (conv2)- 14% (cccp3)- 8% (cccp4)- 22% (conv3)- 48% (cccp5)
AlexNet	3% (conv1)- 20% (conv2)- 24% (conv3)- 18%(conv4)-0%(conv5)-17%(fc6)-23%(fc7)

Zero Activation in DeepCompresison
78.83% (conv1)–30.80%(conv2)-71.05%(conv3)
–62.79%(conv4)–43.78%(conv5)–43.92%(fc6)–48.19%(fc7)

- Compression과 Execution time에서 효과적이긴 하지만
- Fine-grain한 방법에 비해 Compression 효과가 미비함