Scalpel: Customizing DNN Pruning to the Underlying Hardware Parallelism

Jiecao Yu et al.

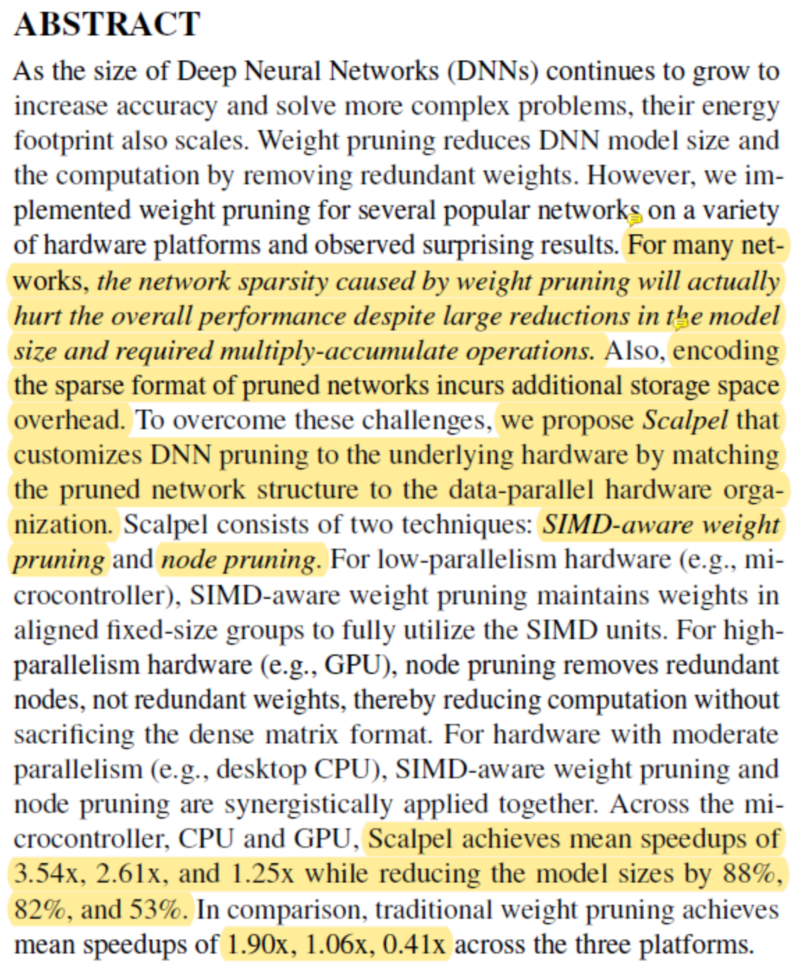
2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)

Presenter : Dawoon kim

March 15, 2020

Contents

Abstraction



ABSTRACT

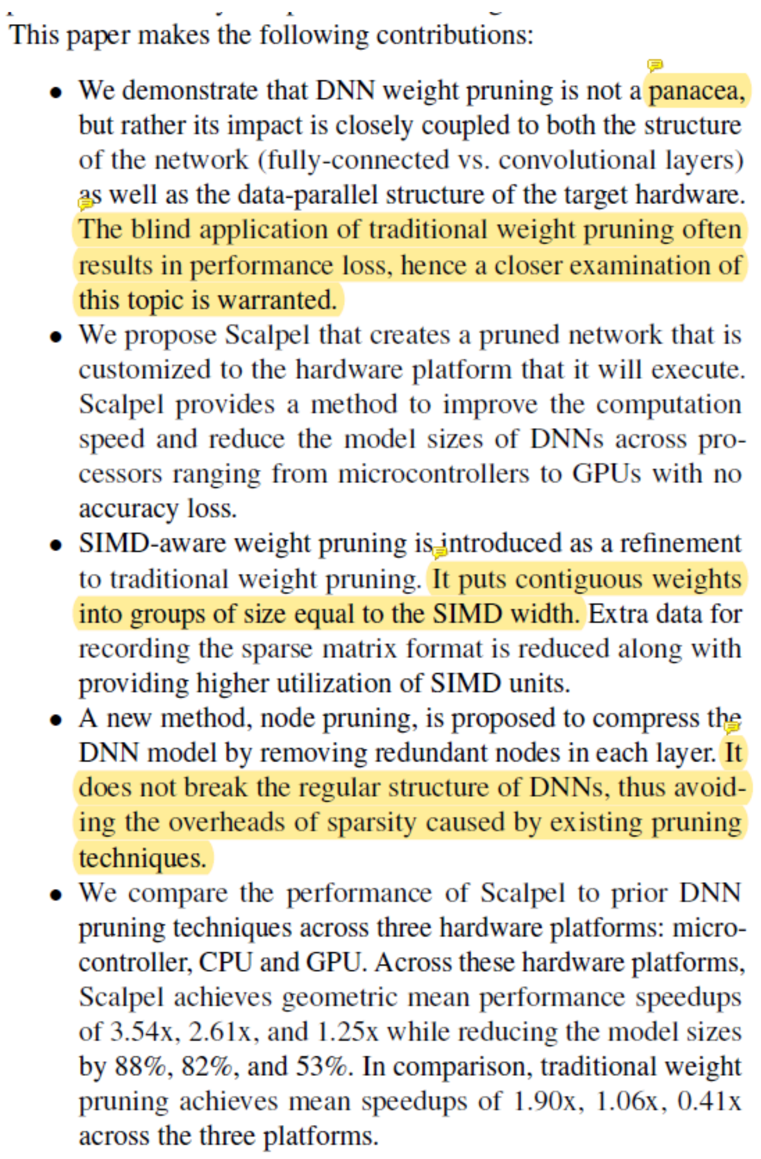
As the size of Deep Neural Networks (DNNs) continues to grow to increase accuracy and solve more complex problems, their energy footprint also scales. Weight pruning reduces DNN model size and the computation by removing redundant weights. However, we implemented weight pruning for several popular networks on a variety of hardware platforms and observed surprising results. For many networks, *the network sparsity caused by weight pruning will actually hurt the overall performance despite large reductions in the model size and required multiply-accumulate operations.* Also, encoding the sparse format of pruned networks incurs additional storage space overhead. To overcome these challenges, we propose *Scalpel* that customizes DNN pruning to the underlying hardware by matching the pruned network structure to the data-parallel hardware organization. Scalpel consists of two techniques: *SIMD-aware weight pruning* and *node pruning*. For low-parallelism hardware (e.g., microcontroller), SIMD-aware weight pruning maintains weights in aligned fixed-size groups to fully utilize the SIMD units. For high-parallelism hardware (e.g., GPU), node pruning removes redundant nodes, not redundant weights, thereby reducing computation without sacrificing the dense matrix format. For hardware with moderate parallelism (e.g., desktop CPU), SIMD-aware weight pruning and node pruning are synergistically applied together. Across the microcontroller, CPU and GPU, Scalpel achieves mean speedups of 3.54x, 2.61x, and 1.25x while reducing the model sizes by 88%, 82%, and 53%. In comparison, traditional weight pruning achieves mean speedups of 1.90x, 1.06x, 0.41x across the three platforms.

✓ Traditional Pruning problems
- Model size, MAC(multiply-accumulate) operations ↓
- 그러나 overall performance는 ↓
- Sparse format을 위한 추가적인 memory 필요

✓ Scalpel
- Data-parallel hardware에 따른 맞춤형 DNN pruning
- SIMD-aware weight pruning
- Node pruning

✓ Performance
- Speedup : 3.54x (micom), 2.61x (CPU), 1.25x (GPU)
  - ❖ Traditional pruning : 1.90x (micom), 1.06x (CPU), 0.41x (GPU)
- Reducing the model size : 88% (micom), 82% (CPU), 53% (GPU)

Introduction



This paper makes the following contributions:

- We demonstrate that DNN weight pruning is not a panacea, but rather its impact is closely coupled to both the structure of the network (fully-connected vs. convolutional layers) as well as the data-parallel structure of the target hardware. The blind application of traditional weight pruning often results in performance loss, hence a closer examination of this topic is warranted.
- We propose Scalpel that creates a pruned network that is customized to the hardware platform that it will execute. Scalpel provides a method to improve the computation speed and reduce the model sizes of DNNs across processors ranging from microcontrollers to GPUs with no accuracy loss.
- SIMD-aware weight pruning is introduced as a refinement to traditional weight pruning. It puts contiguous weights into groups of size equal to the SIMD width. Extra data for recording the sparse matrix format is reduced along with providing higher utilization of SIMD units.
- A new method, node pruning, is proposed to compress the DNN model by removing redundant nodes in each layer. It does not break the regular structure of DNNs, thus avoiding the overheads of sparsity caused by existing pruning techniques.
- We compare the performance of Scalpel to prior DNN pruning techniques across three hardware platforms: microcontroller, CPU and GPU. Across these hardware platforms, Scalpel achieves geometric mean performance speedups of 3.54x, 2.61x, and 1.25x while reducing the model sizes by 88%, 82%, and 53%. In comparison, traditional weight pruning achieves mean speedups of 1.90x, 1.06x, 0.41x across the three platforms.

✓ DNN weight pruning이 만능 X
  - Target HW의 병렬화 구조 뿐 아니라 FC인지 Conv. Layer에 따른 영향이 있다.

✓ Scalpel 제안
  - Micom, CPU, GPU에서 정확도를 잃지 않고 연산속도, 모델 사이즈를 줄일 수 있음

✓ SIMD-aware weight pruning
  - 전통적인 weight pruning을 가공한 방법

✓ Node pruning
  - Weight 대신 node를 제거하는 방법으로 중복 요소를 제거

Background and Motivation

- DNN Weight Pruning

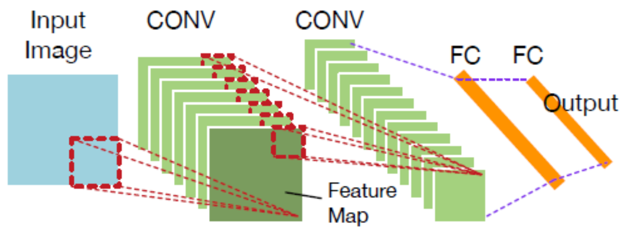


Figure 1: Deep Neural Networks (DNNs) structure. DNNs integrate convolutional layers (CONV) and fully-connected layers (FC) into an end-to-end structure.

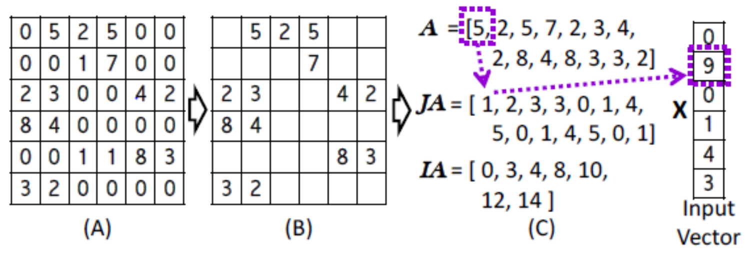

A = [5, 2, 5, 7, 2, 3, 4, 2, 8, 4, 8, 3, 3, 2]
JA = [1, 2, 3, 3, 0, 1, 4, 5, 0, 1, 4, 5, 0, 1]
IA = [0, 3, 4, 8, 10, 12, 14]
X
Input Vector
(A)
(B)
(C)

Figure 2: (A) Dense weight matrix; (B) Sparse weight matrix; (C) Compressed Sparse Rows (CSR) format for sparse matrices.

✓ Sparse matrix -> CSR(Compressed Sparse Rows) 저장 시 메모리 사용↑
- Non-zero value in matrix
- Index of the first non-zero element in each Row
- Column index of the non-zero

✓ Computation performance ↓
- FC/ Conv Layer에 따른 영향

Background and Motivation

- DNN Weight Pruning

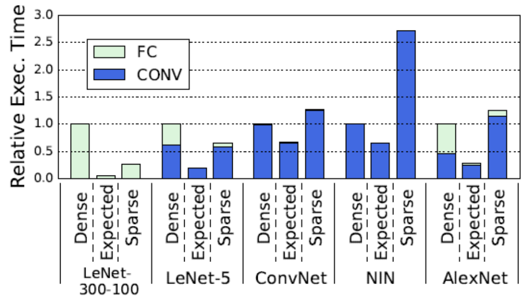


Figure 4: Execution time breakdown of the original networks (Dense) and the pruned networks (Sparse). The expected execution time of pruned networks (Expected) shows the relative MAC operations remaining. The second CONV layer (*conv2*) in AlexNet is tested.

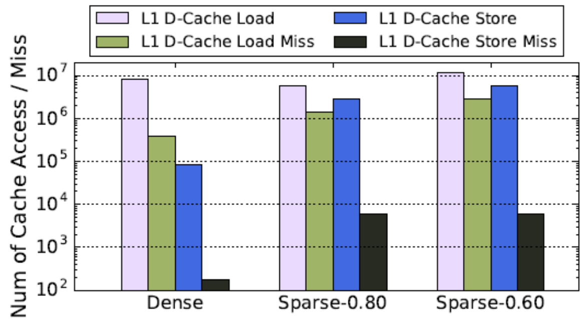


Figure 5: Numbers of cache access and cache miss in the computation of the second CONV layer (*conv2*) in AlexNet on Intel Core i7-6700 CPU. The original dense layer (Dense), the sparse layer with 80% (Sparse-0.80) and 60% (Sparse-0.60) of weights removed are tested. The matrices are randomly generated.

✓ DNN Weight Pruning Execution Time

✓ Alex Net의 second Conv. Layer의 cache acess, miss 분석
- sparsity에 따라 Access, miss ↑
- 80%까지 올려도 dense한 경우보다 좋지 않음

Scalpel

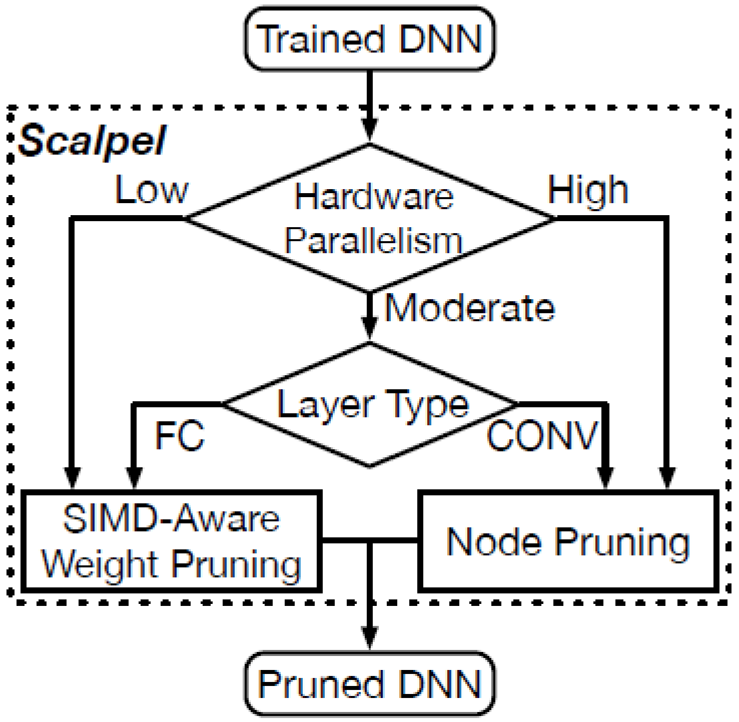


Figure 6: Overview of Scalpel.

✓ Scalpel 구조

- HW parallelism
- Layer Type

❖ SIMD : Single Instruction Multiple Data)

Scalpel

- SIMD-Aware Weight Pruning

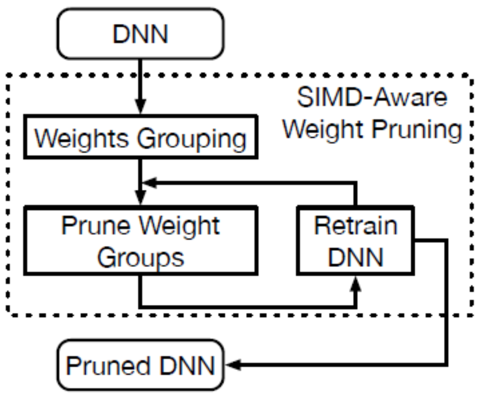
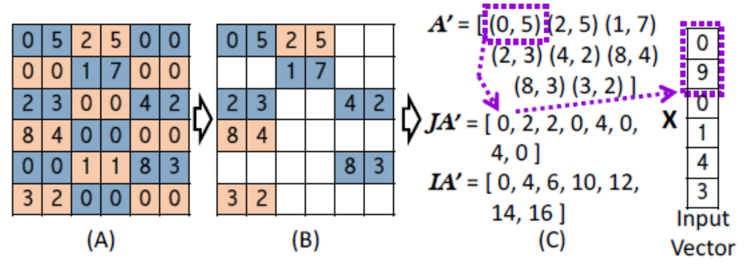


Figure 7: Main steps of SIMD-aware weight pruning.



A′ = [(0, 5) (2, 5) (1, 7) (2, 3) (4, 2) (8, 4) (8, 3) (3, 2)]
JA′ = [0, 2, 2, 0, 4, 0, 4, 0]
IA′ = [0, 4, 6, 10, 12, 14, 16]
X
Input Vector
(A)
(B)
(C)

Figure 8: (A) Weights grouping; (B) Sparse weight matrix after pruning weight groups; (C) Modified CSR format for SIMD-aware weight pruning.

✓ SIMD-Aware Weight Pruning

1. 모든 Weight → SIMD와 동일한 크기의 Align Group으로 나눔
2. 각 Group의 중요도를 계산한다.
3. 중요도가 Threshold 내려간 Group 제거
4. Pruned Weight 행렬 학습

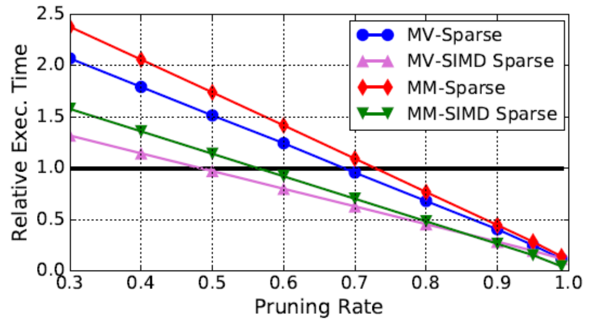


Figure 9: Relative execution time of sparse matrix multiplication on ARM Cortex-M4 with respect to the original dense matrix-vector or matrix-matrix multiplication. MV/MM-Sparse show the results for sparse Matrix-Vector (MV) and Matrix-Matrix (MM) multiplication, respectively. MV/MM-SIMD Sparse shows the corresponding performance with nonzero elements grouped and aligned as SIMD-aware weight pruning does. All matrices have the size of 100 x 100 and are randomly generated.

❖ SIMD : Single Instruction Multiple Data)

Scalpel

- Node Pruning

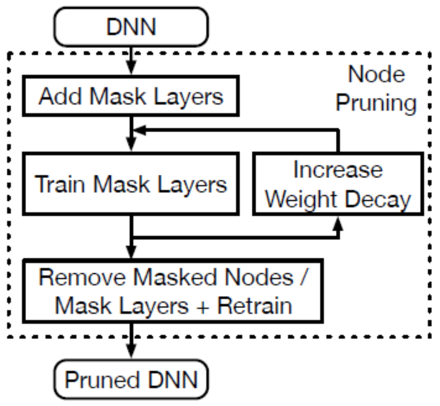


Figure 11: Main steps of node pruning.

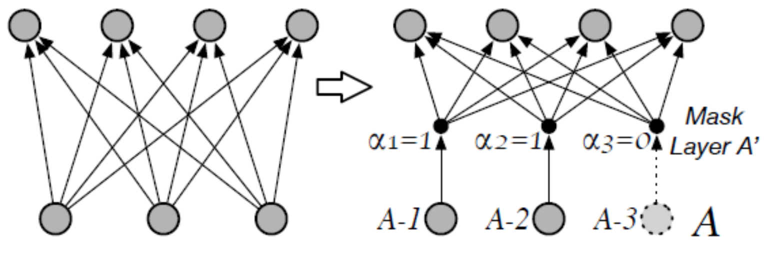


Figure 12: Mask layers. Node *A-3* with $\alpha_3 = 0$ can be removed. The whole mask layer A' will be removed after pruning all redundant nodes.

❖ SIMD : Single Instruction Multiple Data)

✓ Node Pruning

1. 개별 weight 대신 node Pruning
   - FC : neuron
   - Conv. Layer : feature map

2. Node mask

mask layer, α_i and β_i are both initialized to 1:

$$\alpha_i|_0 = 1, \quad \beta_i|_0 = 1.0$$

In training iteration $k \geqslant 1$, α_i is calculated as

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

Evaluation Results

Micom

CPU

GPU

Speed

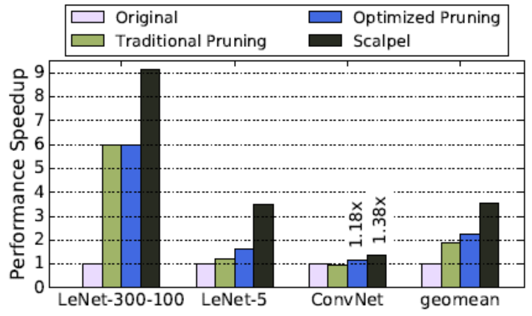


Figure 15: Relative performance speedups of the original models (original), traditional pruning, optimized pruning and Scalpel on ARM Cortex-M4 microcontroller. NIN and AlexNet are not tested due to the limited storage size of the microcontroller.

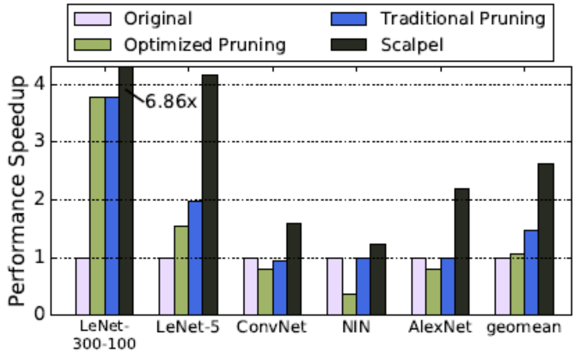


Figure 18: Relative performance speedups of the original models, traditional pruning, optimized pruning and Scalpel on Intel Core i7-6700 CPU.

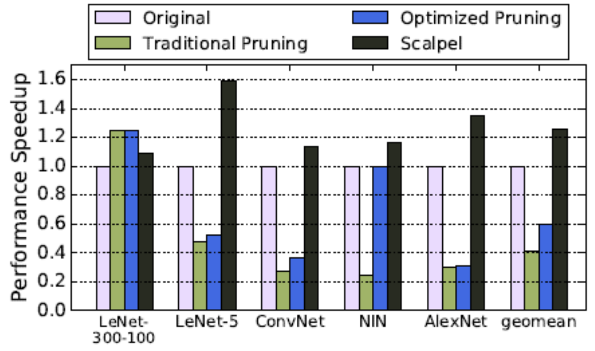


Figure 20: Relative performance speedups of the original models, traditional pruning, optimized pruning and Scalpel on NVIDIA GTX Titan X GPU.

Size

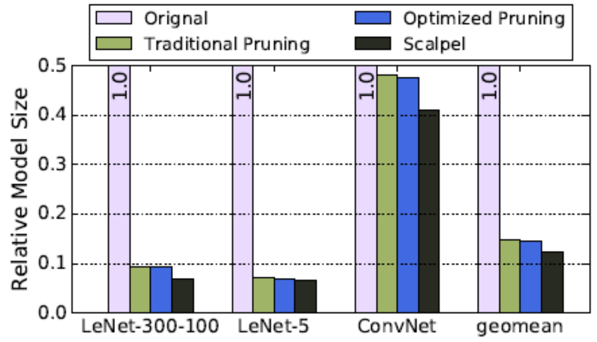


Figure 16: Relative model sizes of the original models, traditional pruning, optimized pruning and Scalpel for ARM Cortex-M4 microcontroller.

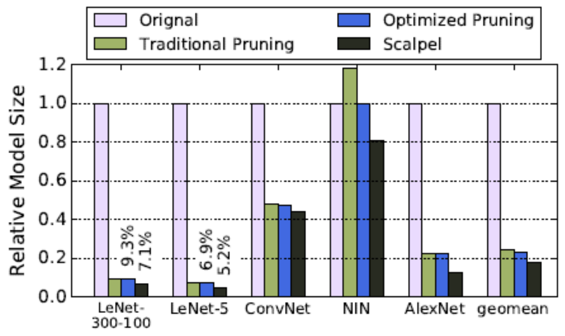


Figure 19: Relative model sizes of the original models, traditional pruning, optimized pruning and Scalpel for Intel Core i7-6700 CPU.

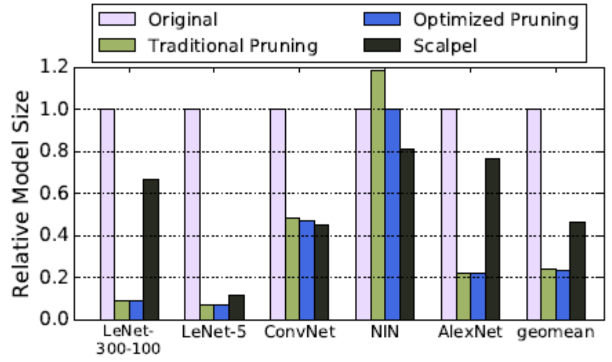


Figure 21: Relative model sizes of the original models, traditional pruning, optimized pruning and Scalpel for NVIDIA GTX Titan X GPU.

Thank you