LayerMerge: Neural Network Depth Compression through Layer Pruning and Merging

Jinuk Kim[1, 2], Marwa El Halabi[3], Mingi Ji[4], Hyun Oh Song[1, 2]

[1]Seoul National University [2]Neural Processing Research Center
[3]Samsung - SAIT AI Lab, Montreal [4]Google

ICML 2024

Problem: Depth Compression of CNN

Existing methods in reducing depth of the CNN usually follows one of two approaches:

- **Pruning Convolution Layers:** Eliminates less important convolution layers.
- **Pruning Activation Layers and Merging Layers:** Eliminates redundant activation layers and merges resulting consecutive convolution layers.

Problem: Depth Compression of CNN

- **Pruning Convolution Layers:** Eliminates less important convolution layers.

  $\rightarrow$ Aggressively removes parameters, risking loss of important information.

Problem: Depth Compression of CNN

- **Pruning Activation Layers and Merging Layers:** Eliminates redundant activation layers and merges resulting consecutive convolution layers.

  $\rightarrow$ Kernel size of the merged layer increases as layers are merged, negating speedup gains.

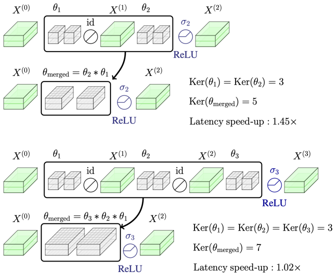
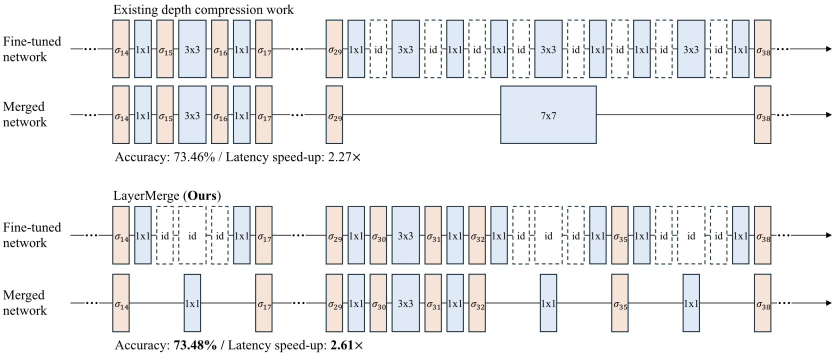


Figure: An illustration of the increase in kernel size undermining the latency reduction when merging layers in CNN.

Proposed Method: Goal

LayerMerge: Jointly prunes convolution and activation layers and merges resulting consecutive convolution layers at the test time.

$\rightarrow$ Have the best of both worlds!



Proposed Method: Goal

Optimize two sets of layer indices:

- A : Where we keep the original activation layers.
- C : Where we keep the original convolution layers.

$$\max_{A\subseteq[L-1],C\subseteq[L]} \max_{\theta} \operatorname{Perf}\left(\bigcirc_{l=1}^{L}(\sigma_{A,l}\circ f_{C,\theta_l,l})\right)$$

$$\text{subject to} \qquad R\subseteq C, \qquad \text{(irreducible conv)}$$

$$\sigma_{A,l} = (\mathbb{1}_A(l)\sigma_l + (1-\mathbb{1}_A(l))\,\mathrm{id}), \quad f_{C,\theta_l,l} = (\mathbb{1}_C(l)f_{\theta_l} + (1-\mathbb{1}_C(l))\,f_{\theta_{\mathrm{id}}}), \qquad \text{(replaced layers)}$$

$$\forall i\in[|A|+1] : \hat{\theta}_i = \circledast_{l=a_{i-1}+1}^{a_i} (\mathbb{1}_C(l)\theta_l + (1-\mathbb{1}_C(l))\,\theta_{\mathrm{id}}), \qquad \text{(merged parameters)}$$

$$T\left(\bigcirc_{i=1}^{|A|}(\sigma_{a_i}\circ f_{\hat{\theta}_i})\right) < T_0, \qquad \text{(latency constraint)}$$

Challenge: The selection problem is **NP-Hard!**

Proposed Method: Surrogate Optimization Problem

Simplify terms:

- $\max_{\theta} \mathrm{Perf}\left(\bigcirc_{l=1}^{L} (\sigma_{A,l} \circ f_{C,\theta_l,l}) \right) \approx$ **Sum of importance** values of merged layers.

- $T\left(\bigcirc_{i=1}^{|A|} (\sigma_{a_i} \circ f_{\widehat{\theta}_i}) \right) \approx$ **Sum of latency** values of merged layers.

Importance of the merged layers: **Change in performance** after replacing the corresponding part of the original network with the merged layer.

Proposed Method: Surrogate Optimization Problem

Key observation: C only affects the latency of a merged layer via the kernel size k.

Proposed approach: Construct look up tables with entries $I[i,j,k]$ and $T[i,j,k]$. Choose C with largest ℓ_1-norm among those resulting in the same merged kernel size k.

$$\begin{aligned} &\max_{A \subseteq [L-1], k_i} \sum_{i=1}^{|A|+1} I[a_{i-1}, a_i, k_i] \\ &\text{subject to} \sum_{i=1}^{|A|+1} T[a_{i-1}, a_i, k_i] < T_0, \quad k_i \in K_{a_{i-1} a_i}, \end{aligned}$$

where K_{ij} is the set of possible merged kernel sizes that can appear after merging from the $(i+1)$-th layer to the j-th layer.

Proposed Method: Surrogate Optimization Problem

Surrogate problem can be solved exactly using a **dynamic programming** algorithm in $O(L^2 K_0)$, where K_0 is the sum of the kernel sizes.

DP recurrence: The maximum objective over the first $\boldsymbol{l} \in [L]$ layers with latency budget $\boldsymbol{t} \in \{\frac{T_0}{P}, \frac{2T_0}{P} \ldots, T_0\}$ is given by

$$M[l,t] = \max_{0 \leqslant l' < l,\, k \in K_{l'l}} \left(\underbrace{M[l', t - T[l', l, k]]}_{\text{Optimal importance sum until } l'\text{-layer}} + \underbrace{I[l', l, k]}_{\text{Importance value of the last compressed layer}} \right).$$

Experimental Results

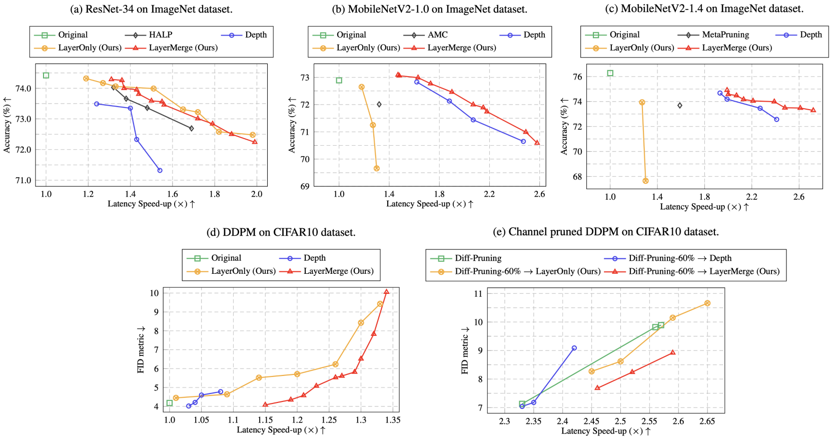


Figure: Pareto curve of each compression method applied to each network. Latency speed-up is measured on RTX2080 Ti GPU with batch size of 128.

Experimental Results

Input Image

MobileNetV2-1.0 (Pre-trained), Inference speedup = **1.00x**

```
# Top-5 prediction of previous image with pre-trained network
print_topk_predictions(pretrained_model, img_tensor)
```

```
Top 1 class | idx: 248 | name: Eskimo dog, husky | probability: 0.5287
Top 2 class | idx: 250 | name: Siberian husky | probability: 0.3235
Top 3 class | idx: 249 | name: malamute, malemute, Alaskan malamute | probability: 0.1283
```

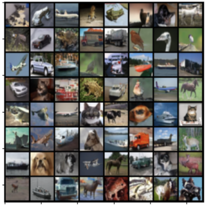
Compressed (LayerMerge-33%), Inference speedup = **2.50x**

```
# Predictions stay consistent after compression
print_topk_predictions(compressed_model, img_tensor)
```

```
Top 1 class | idx: 248 | name: Eskimo dog, husky | probability: 0.5620
Top 2 class | idx: 250 | name: Siberian husky | probability: 0.3250
Top 3 class | idx: 249 | name: malamute, malemute, Alaskan malamute | probability: 0.1128
```

DDPM (Pre-trained)

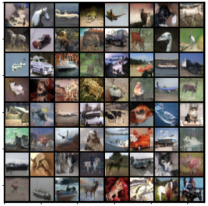
Inferece speedup =**1.00x**



Sampled Images

Compressed (LayerMerge-58%)

Inferece speedup =**1.27x**



Sampled Images

Figure: Qualitative results of applying LayerMerge to pre-trained MobileNetV2-1.0 network on ImageNet and DDPM network on CIFAR10.

Conclusion

- **LayerMerge** reduces **the depth** of CNN by **jointly pruning** convolution and activation layers to make the network more efficient while maintaining performance.

- Results show LayerMerge outperforms current methods for reducing network depth in tasks including image classification and generation.

- `https://github.com/snu-mllab/LayerMerge`