arXiv:2301.12187v1 [cs.LG] 28 Jan 2023

Efficient Latency-Aware CNN Depth Compression via Two-Stage Dynamic Programming

Jinuk Kim [*1] **Yeonwoo Jeong** [*1] **Deokjae Lee** [1] **Hyun Oh Song** [1]

Abstract

Recent works on neural network pruning advocate that reducing the depth of the network is more effective in reducing run-time memory usage and accelerating inference latency than reducing the width of the network through channel pruning. In this regard, some recent works propose depth compression algorithms that merge convolution layers. However, the existing algorithms have a constricted search space and rely on human-engineered heuristics. In this paper, we propose a novel depth compression algorithm which targets general convolution operations. We propose a subset selection problem that replaces inefficient activation layers with identity functions and optimally merges consecutive convolution operations into shallow equivalent convolution operations for efficient end-to-end inference latency. Since the proposed subset selection problem is NP-hard, we formulate a surrogate optimization problem that can be solved exactly via two-stage dynamic programming within a few seconds. We evaluate our methods and baselines by TensorRT for a fair inference latency comparison. Our method outperforms the baseline method with higher accuracy and faster inference speed in MobileNetV2 on the ImageNet dataset. Specifically, we achieve $1.61\times$speed-up with only 0.62%p accuracy drop in MobileNetV2-1.4 on the ImageNet.

1. Introduction

Deep learning with Convolutional Neural Network (CNN) has achieved outstanding results in various fields such as image classification, object detection, image segmentation, and generation (Tan & Le, 2019; Wang et al., 2021; Isensee et al., 2021; Rombach et al., 2022). However, the success of CNNs in such fields is accompanied by the challenge of increased complexity and inference latency. For real-world applications, accelerating the inference latency of CNNs is of great practical importance, especially when deploying the models on edge devices with limited resources.

To this end, a line of research called channel pruning has been introduced to remove unnecessary channels in CNNs to accelerate the wall-clock time in the edge device while preserving the performance of the CNNs (Wen et al., 2016; Tiwari et al., 2021; Shen et al., 2022). However, with the advancement of hardware technology for parallel computation, channel pruning which reduces the width of neural networks has become less effective than removing entire layers in terms of latency (Jordao et al., 2020; Chen & Zhao, 2018; Xu et al., 2020; Fu et al., 2022).

In contrast, layer pruning, which prunes entire layers, has been proposed to reduce the depth of neural networks. Layer pruning also significantly reduces the run-time memory usage and achieves effective speed-up in many edge devices compared to channel pruning (Xu et al., 2020). However, layer pruning is more aggressive than channel pruning in terms of reducing the number of parameters and FLOPs, thereby resulting in a more severe accuracy drop compared to channel pruning methods. Instead of naively removing an entire layer, Fu et al. (2022) present a depth compression algorithm called DepthShrinker which integrates layers by replacing inefficient consecutive depth-wise convolution and point-wise convolution with an efficient dense convolution operation in MobileNetV2 (Sandler et al., 2018). This compression algorithm results in depth reduction with low run-time memory usage and fast inference latency similar to layer pruning. However, the depth compression algorithm does not suffer from a commensurate accuracy drop.

Although DepthShrinker has shown promising results in reducing the depth of the network while preserving the performance, the method is limited to constricted search space as it only considers merging within the Inverted Residual Block (Fu et al., 2022; Sandler et al., 2018). Furthermore, the method relies on human-engineered heuristics for layer merging which is unlikely to scale to other architectures. To this end, we introduce a novel optimization-based frame-

[*]Equal contribution [1] Department of Computer Science and Engineering, Seoul National University. Correspondence to: Hyun Oh Song <hyunoh.snu.ac.kr>.

Preprint.

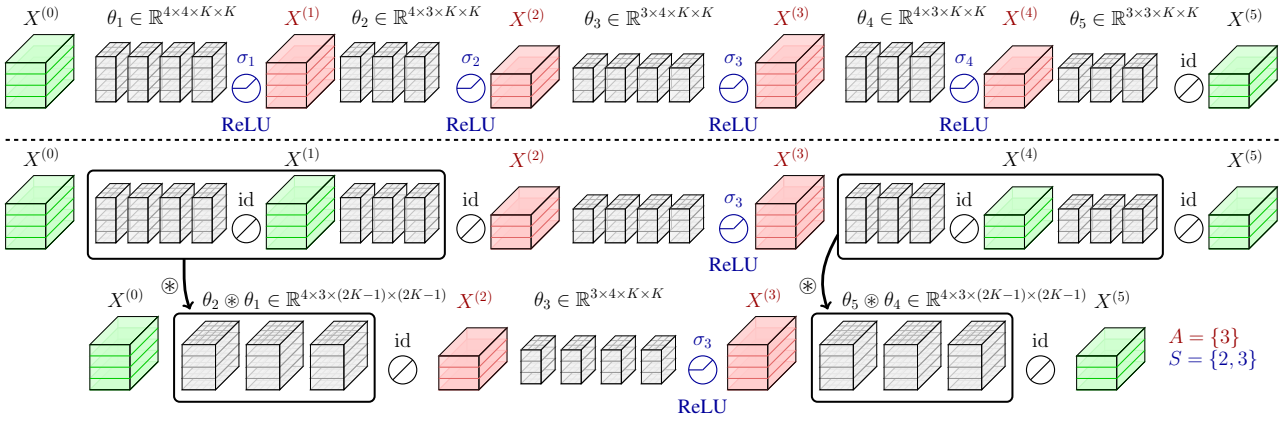

Figure 1. Illustration of depth compression for a five-layer CNN, $\bigcirc_{l=1}^{5} \sigma_l \circ f_{\theta_l}$. The original network is on the default setting when $A = S = \{1, 2, 3, 4\}$ (above). When $A = \{3\}$ and $S = \{2, 3\}$, the activation layer not in A is replaced with identity functions (middle). Then, the network is merged into the shallow network which functions identically (below).

work for general convolution merging framework that is not restricted to the design of the network and does not rely on manually designed heuristics. We formulate a depth compression optimization problem that replaces inefficient activation layers with identity functions and optimally merges consecutive convolution operations for optimal latency.

Our optimization problem is NP-Hard and its objective requires a prohibitively exhaustive training of the neural network. Thus, we formulate a surrogate optimization problem by approximating the objective as the linear sum of the accuracy change induced by each network block. The surrogate optimization problem can be exactly optimized via dynamic programming on a given network architecture with a given latency. Furthermore, we evaluate the latency of the network with TensorRT for a fair comparison (Vanholder, 2016). Our experiments show that the proposed method outperforms the baseline method in both the accuracy and the inference latency in MobileNetV2 on ImageNet dataset.

2. Related Works

Channel Pruning Channel pruning originally aims to reduce computation FLOPs by removing less important channels (Li et al., 2017; He et al., 2019; 2018a; Liu et al., 2019; He et al., 2018b; You et al., 2019; Hu et al., 2016). Specifically, Aflalo et al. (2020) formulate a knapsack problem for channel pruning with an explicit FLOPs constraint. For practitioners, however, end-to-end inference wall-clock time is the most important metric. In light of this, Shen et al. (2022) build a latency lookup table and proposes a knapsack problem for channel pruning with a latency constraint.

Network Morphism Our work is partially inspired by network morphism which morphs a trained parent network into a child network that functions identically (Chen et al., 2016; Wei et al., 2016). Here, the child network is larger than the parent network and is finetuned after morphing. Instead, we aim to find the parent network where some activation layers are removed, thereby morphing the parent network into the child network which has a faster inference time and functions almost identically to the parent network.

Depth Reduction There are two lines of research that reduce the depth of neural networks: layer-pruning and depth compression. In layer pruning, Jordao et al. (2020) and Chen & Zhao (2018) evaluate the importance of layers by the amount of discriminative information in each feature map. In depth compression, DepthShrinker points out the inefficiency of depth-wise convolutions during inference in the edge device and proposes a depth compression algorithm that replaces inefficient consecutive depth-wise convolution and point-wise convolution inside the Inverted Residual Block with an efficient dense convolution (Fu et al., 2022; Howard et al., 2017; Sandler et al., 2018). We generalize depth compression space to cover any general convolution operations. Also, while DepthShrinker requires full training of the network during the search phase to identify the unnecessary activations, our method employs importance evaluation which can be efficiently computed in an embarrasingly parallel fashion. Furthermore, we propose a novel two-stage dynamic programming algorithm which simultaneously finds the optimal set of selected activation layers and the optimal set of layers to be merged in a few seconds.

TensorRT Choosing the appropriate implementation of the network to measure the inference latency is crucial for a fair comparison. For instance, a batch normalization (BN)

module can be fused into the preceding convolution layer without compromising accuracy and accelerating the inference latency. In this regard, we utilize TensorRT to optimize trained network architectures with various techniques such as BN fusion, precision calibration and dynamic memory management (Vanholder, 2016).

3. Preliminary

Consider a L-layer CNN which consists of alternating convolution layer f_{θ_l} and activation layer σ_l with the layer index $l \in [L]$. Each convolution layer is parametrized by convolution kernel parameter $\theta_l \in \mathbb{R}^{C_{l-1} \times C_l \times K_l \times K_l}$, where C_{l-1}, C_l, K_l represent the number of input channels, the number of output channels, and the kernel size, respectively. The CNN can be represented as a composite function $\bigcirc_{l=1}^{L} \sigma_l \circ f_{\theta_l} : R^{H_0 \times W_0 \times K_0 \times K_0} \rightarrow \mathbb{R}^{H_L \times W_L \times K_L \times K_L}$, where H_l, W_l are the height and width of l-th feature map and $\bigcirc$ denotes an iterated function composition. We set the last activation layer (σ_L) to identity function (id).

Note, any consecutive convolution operations can be replaced by an equivalent convolution operation with a larger kernel due to the associative property. We denote this process as *merging*. For example, consider two consecutive convolutional layers, f_{θ_1} and f_{θ_2}, applied to an input image X (*i.e.* $f_{\theta_2}(f_{\theta_1}(X))$). This can also be computed using an equivalent *merged* convolutional layer $f_{\theta_2 \circledast \theta_1}$ where $\circledast$ denotes convolution with proper padding. Further merging details can be found in Appendix D.

4. Method

We first formulate an optimal subset selection problem for depth compression under a given latency constraint. Subsequently, we propose a surrogate objective for the objective in the optimal subset selection problem and formulate a corresponding surrogate optimization problem, which can be exactly solved via two-stage dynamic programming (DP).

4.1. Optimal Subset Selection Problem for Depth Compression

Any neighboring convolutional layers can be merged into an equivalent convolutional layer, often resulting in a latency speed-up from the depth compression of the CNN. In this regard, we aim to optimize the replacement of a subset of activation layers with id in order to reduce the latency of the resulting network while preserving its performance.

However, merging every consecutive series of convolutional layers into a single large layer may not be the optimal merge in terms of latency. In certain cases, it is possible that merging certain convolutional layers has a detrimental effect on the latency of a network. To illustrate, consider merging two consecutive 1×1 convolutional layers, with the first layer having 100 input channels and 1 output channel and the second layer having 1 input channel and 100 output channels. Then the merged convolutional layer results in a 1×1 convolution with 100 input channels and 100 output channels. This merge significantly increases the latency of a merged convolutional layer, thereby canceling out any benefits gained from the depth compression.

To address this, we propose two *ordered set* variables, A and S to be simultaneously optimized. A indicates the layer indices where the activation layer is kept intact and not replaced with an identity function, and S indicates the layer indices where we do not merge. It is important to note that S includes A, since the activation layers that are not id can not be merged. Figure 1 illustrates how network is merged according to S and A. Our goal is to optimize for the ordered set A and S in order to reduce the latency of the resulting network while preserving its performance.

Thus, our objective can be formulated as follows:

$$\underset{A \subseteq S \subseteq [L-1]}{\text{maximize}} \ \max_{\theta} \ \text{Acc}\left(\bigcirc_{l=1}^{L} (\mathbb{1}_A(l)\sigma_l + (1 - \mathbb{1}_A(l))\,\text{id}) \circ f_{\theta_l} \right) \tag{1a}$$

$$\text{subject to} \quad T\left(\bigcirc_{i=1}^{|S|+1} (\mathbb{1}_A(s_i)\sigma_{s_i} + (1 - \mathbb{1}_A(s_i))\,\text{id}) \circ f_{\hat{\theta}_i} \right) < T_0 \tag{1b}$$

$$\hat{\theta}_i = \bigcircledast_{l=s_{i-1}+1}^{s_i} \theta_l, \ \forall i \in [|S|+1] \ \text{ and } \ s_{|S|+1} = L, \ s_0 = 0,$$

where $(s_i)_{i=1}^{|S|}$ denotes the elements of S and $\text{Acc}(\cdot)$ and $T(\cdot)$ denote the accuracy and latency of the network, respectively. The objective in Equation (1a) describes the accuracy of the network where the activation layer is replaced by id if the layer index is not in A. The constraint in Equation (1b) describes the latency constraint of the network, which is merged according to S. Note, the networks in Equation (1a) and Equation (1b) function identically.

We can simplify the constraint in Equation (1b) by expressing the total latency of the network as the sum of the latency of each merged convolution layer as each layer is sequentially connected. Additionally, we ignore the latency from each activation layer since the latency incurred by activation layers is negligible[1]. Then, the total latency of the merged network in the constraint can be simplified as follows:

[1]Deactivating 50 ReLUs in MobileNetV2 results in less than a 1% change in end-to-end inference time on *RTX 2080 Ti*.

$$T\left(\bigcirc_{i=1}^{|S|+1}\left(\mathbb{1}_A(s_i)\sigma_l+(1-\mathbb{1}_A(s_i))\,\mathrm{id}\right)\circ f_{\hat{\theta}_i}\right)$$
$$\approx\sum_{i=1}^{|S|+1}T\left(\left(\mathbb{1}_A(s_i)\sigma_l+(1-\mathbb{1}_A(s_i))\,\mathrm{id}\right)\circ f_{\hat{\theta}_i}\right)\approx\sum_{i=1}^{|S|+1}T\left(f_{\hat{\theta}_i}\right).$$

As a shorthand, we denote $T(f_{\theta'})$ where $\theta'=\circledast_{l=i+1}^{j}\theta_l$ as $T[i,j]$. Note that $f_{\theta'}$ is the merged convolution operation equivalent to $\bigcirc_{l=i+1}^{j}f_{\theta_l}$. Thus, the latency constraint, Equation (1b), can be compactly expressed as $\sum_{s_{i-1},s_i\in\{0\}\cup S\cup\{L\}}T[s_{i-1},s_i]<T_0$.

4.2. Formulation with the Surrogate Objective

Directly optimizing Equation (1) requires training of the whole network for all combinations of A and S which is NP-hard. Therefore, we propose a surrogate objective for the objective in Equation (1a). Through this approach, Equation (1) can be reformulated into an optimal subset selection problem which can be exactly solved via DP.

The network in Equation (1a) can be equivalently represented as $\bigcirc_{j=1}^{|A|+1}\sigma_{a_j}\circ\left(\bigcirc_{l=a_{j-1}+1}^{a_j}f_{\theta_l}\right)$ where $a_0=0, a_{|A|+1}=L$, and $(a_j)_{j=1}^{|A|}$ denote the elements of A in the ascending order. We can observe that A partitions the network into contiguous network blocks, $\bigcirc_{l=a_{i-1}+1}^{a_i}f_{\theta_l}$. Therefore, Equation (1a) can be reformulated as the accuracy change from the original network as follows:

$$\underset{A\subseteq S\subseteq[L-1]}{\text{maximize}}\ \max_{\theta}\ \mathrm{Acc}\left(\bigcirc_{j=1}^{|A|+1}\sigma_{a_j}\circ\left(\bigcirc_{l=a_{j-1}+1}^{a_j}f_{\theta_l}\right)\right)-\max_{\theta}\mathrm{Acc}\left(\bigcirc_{l=1}^{L}\sigma_l\circ f_{\theta_l}\right),\tag{2}$$

where $\max_\theta \mathrm{Acc}\left(\bigcirc_{l=1}^{L}\sigma_l\circ f_{\theta_l}\right)$ is the accuracy of the original network which is a constant.

Each contiguous block results in an accuracy change from the original network. However, the exact estimation of the accuracy change resulting from all possible combinations of contiguous network blocks remains impractical due to the exponential number of possible combinations of contiguous network blocks and the requirement of training the neural network for each one. Therefore, we propose the sum of the accuracy change caused by each contiguous network block, $\bigcirc_{l=a_{i-1}+1}^{a_i}f_{\theta_l}$ as a proxy for the accuracy change resulting from contiguous network blocks in Equation (2).

We denote $I[i,j]$ as the accuracy change when the activation layers between the i+1-th and j-th layers in the original network are replaced with id. Concretely,

$$I[i,j]\coloneqq\max_{\theta}\mathrm{Acc}\left(\underbrace{\bigcirc_{l=j+1}^{L}\sigma_l\circ f_{\theta_l}}_{j+1\text{ to }L\text{ layers}}\circ\underbrace{\bigcirc_{l=i+1}^{j}f_{\theta_l}}_{i+1\text{ to }j\text{ layers}}\circ\underbrace{\bigcirc_{l=1}^{i}\sigma_l\circ f_{\theta_l}}_{1\text{ to }i\text{ layers}}\right)-\max_{\theta}\mathrm{Acc}\left(\bigcirc_{l=1}^{L}\sigma_l\circ f_{\theta_l}\right).\tag{3}$$

Note that computing $I[\cdot,\cdot]$ can be efficiently done in embarrassingly parallel fashion. We define the surrogate objective for Equation (1a) as $\sum_{a_{j-1},a_j\in\{0\}\cup A\cup\{L\}}I[a_{j-1},a_j]$. Then the optimization problem in Equation (1) becomes

$$\begin{aligned}&\underset{A\subseteq S\subseteq[L-1]}{\text{maximize}}\quad\sum_{a_{j-1},a_j\in\{0\}\cup A\cup\{L\}}I[a_{j-1},a_j]\\&\text{subject to}\quad\sum_{s_{i-1},s_i\in\{0\}\cup S\cup\{L\}}T[s_{i-1},s_i]<T_0.\end{aligned}\tag{4}$$

4.3. Optimization via Dynamic Programming

We first define an ordered set for the indices to be merged for the optimal inference time for the contiguous network block between k+1-th layer and l-th layer as $S_{\text{opt}}[k,l]$ and the optimal inference time as $T_{\text{opt}}[k,l]$. Concretely,

$$T_{\text{opt}}[k,l]\coloneqq\min_{S\subseteq\{k+1,\dots,l-1\}}\sum_{s_{i-1},s_i\in\{k\}\cup S\cup\{l\}}T[s_{i-1},s_i]\tag{5a}$$

$$S_{\text{opt}}[k,l]\coloneqq\operatorname*{argmin}_{S\subseteq\{k+1,\dots,l-1\}}\sum_{s_{i-1},s_i\in\{k\}\cup S\cup\{l\}}T[s_{i-1},s_i].\tag{5b}$$

For the base case, $T_{\text{opt}}[k,k]=0$ and $S_{\text{opt}}[k,k]=\emptyset$. Then, $T_{\text{opt}}[k,l]$ and $S_{\text{opt}}[k,l]$ can be computed via dynamic programming algorithm as described in Algorithm 1.

We formulate a sub-optimization problem of Equation (4) with respect to an intermediate layer index, $l\le L$ and a latency constraint, $t>T_{\text{opt}}[0,l]$:

$$\begin{aligned}&\underset{A\subseteq S\subseteq[l-1]}{\text{maximize}}\quad\sum_{a_{j-1},a_j\in\{0\}\cup A\cup\{l\}}I[a_{j-1},a_j]\\&\text{subject to}\quad\sum_{s_{i-1},s_i\in\{0\}\cup S\cup\{l\}}T[s_{i-1},s_i]<t.\end{aligned}\tag{6}$$

Then, we define the optimal ordered sets A and S in the sub-optimization problem as $A[l,t]$ and $S[l,t]$. Here, $A[l,t]$ indicates activation layers to keep until layer l given the latency budget t, and $S[l,t]$ indicates layers to merge until layer l given the latency budget t. Then, $A[L,T_0]$ and

Algorithm 1 Finding Optimal Latency with DP

input T, L
  Initialize $T_{\text{opt}}[k,l] \leftarrow 0, S_{\text{opt}}[k,l] \leftarrow \emptyset$ for $0 \le k \le l \le L$
  for $l = 1$ **to** L **do**
    for $k = 0$ **to** $l-1$ **do**
      $m \leftarrow \underset{k \le m' < l}{\text{argmin}} \left(T_{\text{opt}}[k,m'] + T[m',l]\right)$
      $T_{\text{opt}}[k,l] \leftarrow T_{\text{opt}}[k,m] + T[m,l]$
      if $m \notin \{k\}$ **then**
        $S_{\text{opt}}[k,l] \leftarrow S_{\text{opt}}[k,m] \cup \{m\}$
      end if
    end for
  end for
output $T_{\text{opt}}, S_{\text{opt}}$

Algorithm 2 Solving the Surrogate Objective with DP

input T_0, L, T, I
  Initialize $D[l,t] \leftarrow 0, A[l,t] \leftarrow \emptyset, S[l,t] \leftarrow \emptyset \quad \forall t, l$
  $T_{\text{opt}}, S_{\text{opt}} \leftarrow$ Algorithm 1 (T, L)
  for $l = 1$ **to** L **do**
    for $t = T_{\text{opt}}[0,l] + 1$ **to** T_0 **do**
      $k \leftarrow \underset{0 \le k' < l}{\text{argmax}} \left(D[k', t - T_{\text{opt}}[k',l]] + I(k',l)\right)$
        subject to $T_{\text{opt}}[0,k'] + T_{\text{opt}}[k',l] < t$
      $t_{\text{last}} \leftarrow T_{\text{opt}}[k,l]$
      $D[l,t] \leftarrow D[k, t - t_{\text{last}}] + I[k,l]$
      $A[l,t] \leftarrow A[k, t - t_{\text{last}}] \cup \{k : k > 0\}$
      $S[l,t] \leftarrow S[k, t - t_{\text{last}}] \cup \{k : k > 0\} \cup S_{\text{opt}}[k,l]$
    end for
  end for
output $A[L,T_0], S[L,T_0]$

$S[L,T_0]$ represent the optimal set A and S of the surrogate optimization problem, Equation (4), respectively.

For the base case, we set $A[0,t] = S[0,t] = \emptyset$. Then, we compute the ordered sets $A[l,t]$ and $S[l,t]$ according to the dynamic programming (DP) recurrence relation defined by

$$A[l,t] = A[k, t - T_{\text{opt}}[k,l]] \cup \{k : k > 0\} \tag{7a}$$
$$S[l,t] = S[k, t - T_{\text{opt}}[k,l]] \cup \{k : k > 0\} \cup S_{\text{opt}}[k,l], \tag{7b}$$
$$k = \underset{0 \le k' < l}{\text{argmax}} \sum_{a_{j-1}, a_j \in \{0\} \cup A[k', t - T_{\text{opt}}[k',l]] \cup \{k', l\}} I[a_{j-1}, a_j]$$
$$\text{subject to} \quad T_{\text{opt}}[0,k'] + T_{\text{opt}}[k',l] < t,$$

where k is the maximum element of $A[l,t]$. Therefore, $A[l,t]$ is an empty set when $k = 0$. Figure 2 illustrates DP computation example in detail.

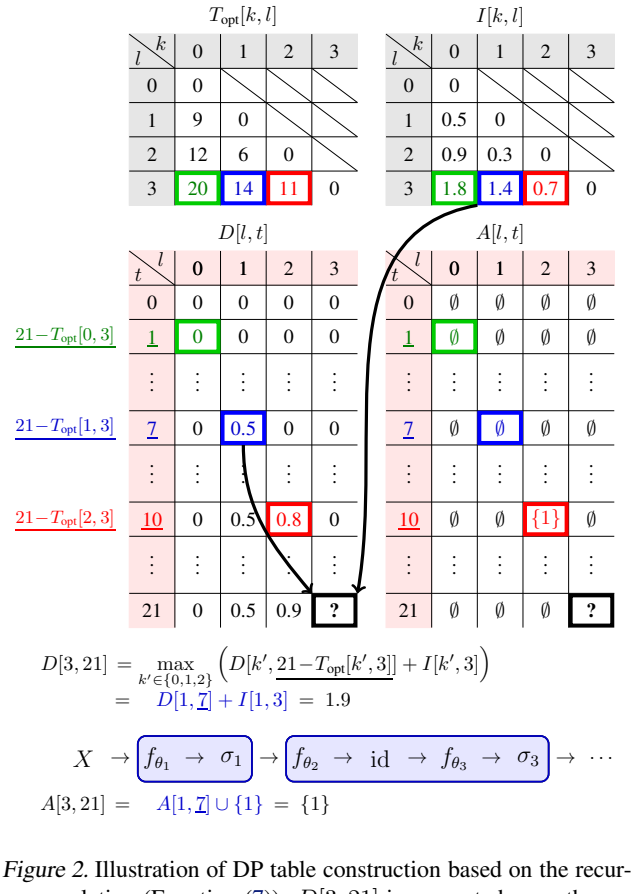

Figure 2. Illustration of DP table construction based on the recurrence relation (Equation (7)). $D[3,21]$ is computed upon the solutions of the previous sub-problems ($D[0,1]$, $D[1,7]$, $D[2,10]$).

4.4. Theoretical Analysis

Proposition 4.1 shows that Equation (7) exactly computes $A[l,t]$ and $S[l,t]$. The detailed procedure for implementing the DP recurrence relation can be found in Algorithm 2[2]. At the start of Algorithm 2, we compute the ordered set for the indices to be merged for the optimal inference time for the contiguous network block between k+1-th layer and l-th layer and the optimal inference time at Algorithm 1 where the time complexity for the DP recurrences is $\mathcal{O}(L^3)$. In Algorithm 2, the time complexity for the DP recurrences is $\mathcal{O}(L^2 T_0)$, thus the total time complexity is $\mathcal{O}(L^3 + L^2 T_0)$.

Proposition 4.1. *$A[l,t]$ and $S[l,t]$ computed from the DP recurrence relations, Equation (7) are the optimal sets A and S of Equation (6), respectively.*

Proof. Refer to Appendix A. □

For a given set of the optimal indices where activation layers are not replaced with identity functions, $A[l,t]$ and

[2]We denote $\sum_{a_{j-1}, a_j \in \{0\} \cup A[l,t] \cup \{l\}} I[a_{j-1}, a_j]$ as $D[l,t]$ in Algorithm 2 for brevity.

the network replaced according to $A[l,t]$, Proposition 4.2 shows that $S[l,t]$ is the optimal S which merges the network into the optimal structure in terms of latency.

Proposition 4.2. *$S[l,t]$ computed from the DP recurrence relations, Equation (7) is the optimal S which minimizes the latency of the network when $A[l,t]$ is fixed. Concretely, $S[l,t]$ is the optimal S of the optimization problem:*

$$\min_{A[l,t]\subseteq S\subseteq[l-1]} \sum_{s_{i-1},s_i\in\{0\}\cup S\cup\{l\}} T[s_{i-1},s_i]. \tag{8}$$

Proof. Refer to Appendix A. □

5. Experimental Results

We evaluate our method on various datasets and network architectures. We first introduce implementation details for the overall process in the experiments. Then, we present an evaluation of our method on various scales of networks and datasets to demonstrate its effectiveness. Furthermore, we conduct ablation studies on the search space of our proposed method and the ordered set S.

5.1. Implementation Details

Measurement We first evaluate the latency of each contiguous network block, $T[i,j]$, individually. The latency of the network is subject to the format which it is implemented on. We utilize TensorRT to convert the network into its optimal form and measure the latency for a fair comparison.

Then, we measure the change of the accuracy incurred by each contiguous network block, $I[i,j]$, for Equation (3). As the number of possible contiguous network blocks is of the order of N^2, where N is the number of activations, we need to train $\mathcal{O}(N^2)$ networks to obtain the accuracy change of every contiguous network block. For efficiency, we approximate the first term in Equation (3) using the accuracy of the network trained for a few epochs after replacing the activation layers between the i+1-th and j-th layers with identity functions. Specific details on the number of blocks for each architecture as well as the methodologies used to normalize the importance values can be found in the Appendix B.3.

Dynamic Programming Given the latency of each contiguous network block, $T[i,j]$, and the accuracy change caused by each contiguous network block, $I[i,j]$, we can solve Equation (4) for the time constraint T_0 with Algorithm 2. In Algorithm 2, we assume the time constraint T_0 and time index t to be integers. In practice, we multiply every occurrence of t and T_0 by a constant factor and round the multiplied values to integer.

Table 1. Accuracy and latency of compressed architectures applied to MobileNetV2-1.0 and MobileNetV2-1.4 on ImageNet-100 dataset. Compression methods use the latency information of *RTX 2080 Ti* and the latency is measured on the same *RTX 2080 Ti*.

Network	Acc (%)	TensorRT Lat. (ms)	*w/o* TensorRT Lat. (ms)
MBV2-1.0	87.58	19.55	40.68
DS-A-1.0	85.60	15.01	27.71
DS-B-1.0	85.16	12.59	23.09
Ours	**85.98**	**12.49**	**22.46**
DS-C-1.0	84.30	11.39	20.91
Ours	**84.36**	**10.50**	**18.39**
DS-D-1.0	83.18	10.65	18.91
Ours	**84.00**	**9.92**	**17.74**
MBV2-1.4	88.88	30.47	61.83
DS-A-1.4	85.58	19.92	35.28
DS-B-1.4	84.98	19.52	31.79
Ours	**86.16**	**18.74**	**31.22**
DS-C-1.4	84.00	17.76	29.87
Ours	**85.04**	**16.79**	**27.92**
DS-D-1.4	83.02	17.77	28.09
DS-E-1.4	81.80	15.93	26.14
Ours	**84.50**	**14.94**	**23.26**

Finetune and Merge After obtaining the optimal ordered sets A and S in Equation (4), we replace the activation layers that are not present in A with identity functions. Then, we apply the necessary zero padding required for the accurate merging in accordance with S and finetune the network until convergence. At the test time, we merge the finetuned network following S and evaluate the latency.

During finetuning, we follow the identical training protocol with the DepthShrinker for finetuning (Fu et al., 2022). In detail, we finetune the network for 180 epochs using cosine learning rate decay with the SGD optimizer. We further adopt the label smoothing and RandAugment following the Fu et al. (2022) when we finetune the MobileNetV2-1.4 on ImageNet dataset (Müller et al., 2019; Cubuk et al., 2020).

Evaluation We employ *RTX2080 Ti* GPU when evaluating the latency of each contiguous network block. Then, we evaluate the end-to-end inference latency of merged architectures on various GPUs including *TITAN Xp*, *RTX2080 Ti*, *RTX 3090*, and *Tesla V100*. Also, we evaluate the inference latency of the networks in two distinct formats: 1) TensorRT exported model (FP32) and 2) PyTorch model (Vanholder, 2016; Paszke et al., 2017). To ensure a fair

Table 2. Accuracy and latency of compressed architectures applied to MobileNetV2-1.0 on ImageNet dataset. Both compression methods use the latency information of *RTX 2080 Ti* and the latency is measured on the same *RTX 2080 Ti*. Accuracy of the baselines are brought from the paper (Fu et al., 2022).

Network	Acc (%)	TensorRT Lat. (ms)	*w/o* TensorRT Lat. (ms)
MBV2-1.0	72.30	19.65	40.84
DS-A-1.0	72.43	15.05	27.76
Ours	**72.67**	**13.94**	**25.25**
DS-B-1.0	71.54	12.60	23.11
Ours	**71.90**	**12.36**	**21.95**
DS-C-1.0	70.90	11.43	21.06
Ours	**71.29**	**11.10**	**20.67**
DS-D-1.0	69.40	10.72	19.03
Ours	**70.61**	**10.08**	**16.77**

comparison, we fuse the batch normalization (BN) modules with the previous convolution layers when we measure latency in the PyTorch format, as the depth compression algorithm results in a different number of BN modules.

5.2. Depth Compression Results

We apply our depth compression method to the MobileNetV2 architecture on ImageNet-100 and ImageNet dataset, starting from the public pretrained weight (Sandler et al., 2018; Tian et al., 2020; Russakovsky et al., 2015).

5.2.1. ImageNet-100

We first experiment our depth compression method on the ImageNet-100 dataset, which is a subset of ImageNet consisting of 100 classes. We bring the list of the subclasses from Tian et al. (2020). The size of the image is preprocessed to 224×224 and the dataset contains approximately 1200 images per class. We apply our depth compression method to both MobileNetV2-1.0 and MobilNetV2-1.4 starting from the pretrained weight and compare to the architectures proposed in DepthShrinker.

When implementing the DepthShrinker on the ImageNet-100 dataset, we bring the architectures in DepthShrinker and finetune from the pretrained weight after substituting the last classifier to match the number of classes (Fu et al., 2022). Then we measure the latency of the merged network.

Table 1 summarizes the depth compression results in MobileNetV2-1.0 and MobileNetV2-1.4. Our method consistently outperforms the baseline at every compression ratio in MobileNetV2-1.0 and MobileNetV2-1.4. In particular, we achieve $1.19\times$ speedup in TensorRT compiled format with 1.48%p higher accuracy compared to the baseline (comparing DS-D-1.4 to ours). Also, we acheive $1.20\times$ speedup with 0.38%p higher accuracy in TensorRT compiled format compared to the baseline (comparing DS-A-1.0 to ours).

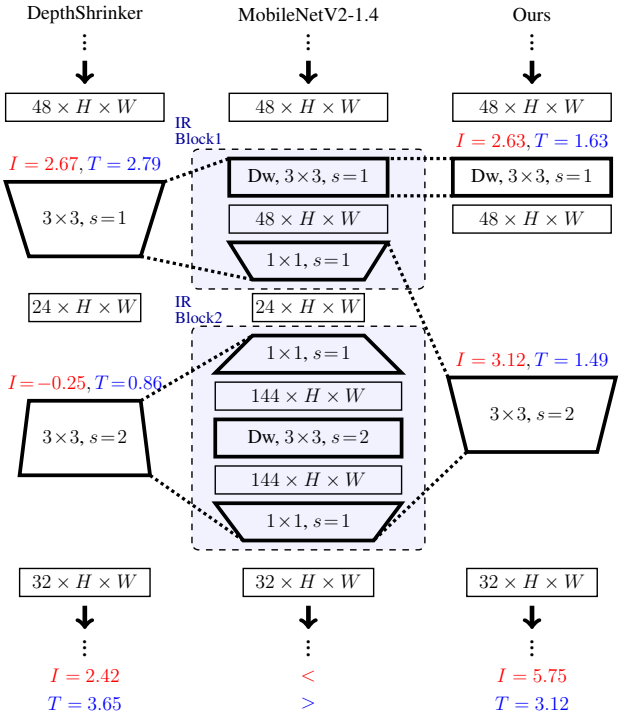

Figure 3. Example of our method finding the network structure that DepthShrinker is unable to find. Our method has a larger search space since it can merge across the blocks while DepthShrinker only considers merging within the Inverted Residual Block. $I[\cdot,\cdot]$ and $T[\cdot,\cdot]$ are evaluated for MobileNetV2-1.4 on ImageNet.

Additionally, we evaluate the wall-clock inference time on various GPU platforms other than *RTX 2080 Ti*. The comprehensive result of the latency on different GPUs can be found in Appendix C.1. Furthermore, we reproduce the full searching stage of DepthShrinker on top of the ImageNet-100 dataset and compare our method against the resulting architecture which we also provide the results in Appendix C.1.

5.2.2. ImageNet

We apply our depth compression method to MobileNetV2-1.0 and MobileNetV2-1.4 on the full ImageNet dataset (Russakovsky et al., 2015). Note that every method uses the latency information of the *RTX 2080 Ti* with TensorRT and is measured on different model formats and GPU plat-

Table 3. Accuracy and latency of compressed architectures applied to MobileNetV2-1.4 on top of the ImageNet dataset. Both compression methods use the latency information of *RTX 2080 Ti*. The latency of the compressed network architecture is measured on *TITAN Xp*, *RTX 2080 Ti*, *RTX 3090*, and *Tesla V100*. Accuracy of the baselines are brought from the paper (Fu et al., 2022).

		TensorRT Latency (ms)				*w/o* TensorRT (ms)
Network	Acc (%)	*TITAN Xp*	*RTX 2080 Ti*	*RTX 3090*	*Tesla V100*	*RTX 2080 Ti*
MobileNetV2-1.4	75.30	42.88	30.58	22.12	26.34	62.01
MBV2-1.4-DS-A	74.65	27.30	20.01	14.57	18.22	35.44
Ours	**74.68**	**26.27**	**18.96**	**14.00**	**17.62**	**32.79**
MBV2-1.4-DS-B	73.67	25.74	19.66	14.17	17.67	32.02
Ours	**73.76**	**23.81**	**17.30**	**12.64**	**16.21**	**30.20**
MBV2-1.4-DS-C	73.38	24.15	17.79	12.89	16.67	30.07
Ours	**73.47**	**22.81**	**16.67**	**12.06**	**15.38**	**27.90**
MBV2-1.4-DS-D	72.51	22.97	17.76	12.88	16.36	28.35
MBV2-1.4-DS-E	72.20	21.39	16.02	11.75	14.91	26.39
Ours	**72.56**	**20.88**	**15.29**	**11.21**	**14.72**	**26.10**

forms.

Table 2 demonstrates that our method consistently outperforms the baseline in MobileNetV2-1.0 architecture on the ImageNet dataset. In particular, our method attains $1.08\times$ speedup with 0.24%p higher accuracy compared to the baseline (comparing ours to the DS-A-1.0). We present the comprehensive table including the latency on different GPUs in Appendix C.2.

Table 3 shows the result of applying our method to MobileNetV2-1.4. The result demonstrates that our method outperforms the baseline method in every compression ratio and across all model formats and GPU platforms. In particular, our method achieves $1.14\times$ speedup in TensorRT compiled format with higher accuracy compared to the baseline DepthShrinker model (comparing MBV2-1.4-DS-B to ours). Compared to the vanilla network, our compressed network achieves $1.61\times$ speedup in TensorRT compiled format with 0.62%p accuracy drop when we compare to vanilla network. We achieve a bigger speedup factor ($1.89\times$ speedup) in PyTorch with the same accuracy.

5.3. An Illustration of the Larger Search Space

The scope of the DepthShrinker is restricted to the cases where merging operation occurs within the Inverted Residual Block (Fu et al., 2022). On the other hand, our merging algorithm can handle any series of convolution operations and is agnostic to any specific block structure. For instance, our method finds the architecture that merges across the blocks, which DepthShrinker cannot find as shown in Figure 3. Our method allows us to merge more general series of layers enabling us to discover a more diverse kind of efficient structure.

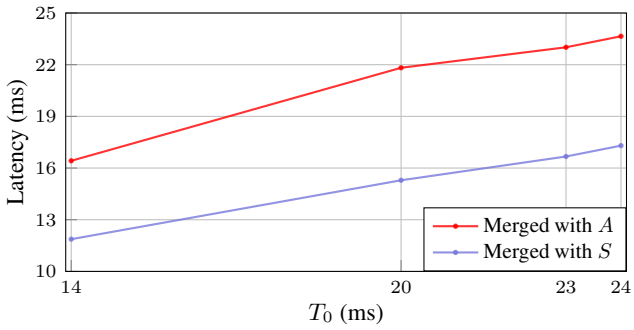

Figure 4. Latency comparison between the network that is merged according to A and the network that is merged according to S for different time constraint T_0. A and S are the optimal solutions of Equation (4) where $I[\cdot,\cdot]$ and $T[\cdot,\cdot]$ are evaluated for MobileNetV2-1.0 on ImageNet dataset.

5.4. Ablation Study on Ordered Set to be Merged

Recall the definition of A and S: A indicates locations where activation layer is not replaced with an identity function and S indicates indices where we do not merge. The set S always includes A since the activation layers that are not id cannot be merged. One could argue that we can merge the layers with respect to A, without separately computing the optimal merge pattern S. In this ablation study, we compare the inference time of merged network according to A and S. Figure 4 shows that the network merged according to S is about 30% faster than the network merged according to A. This demonstrates that jointly optimizing over A and S simultaneously is crucial for optimal depth compression.

6. Conclusion

We propose an efficient depth compression algorithm to reduce the depth of neural networks for the reduction in run-time memory usage and fast inference latency. Our compression target includes any general convolution operations, whereas existing methods are limited to consecutive depth-wise convolution and point-wise convolution within Inverted Residual Block. We propose a subset selection problem which replaces inefficient activation layers with identity functions and optimally merges consecutive convolution operations into shallow equivalent convolution operations for fast end-to-end inference latency. Since the optimal depth subset selection problem is NP-hard, we formulate a surrogate optimization problem which can be exactly solved via two-stage dynamic programming within a few seconds. We evaluate our methods and baselines by TensorRT for a fair inference latency comparison. Our method outperforms Depthshrinker with a higher accuracy and faster inference speed in MobileNetV2 on the ImageNet dataset. Specifically, we achieve $1.61\times$speed-up with only 0.62%p accuracy drop in MobileNetV2-1.4 on the ImageNet dataset.

References

Aflalo, Y., Noy, A., Lin, M., Friedman, I., and Zelnik, L. Knapsack pruning with inner distillation. In *arXiv:2002.08258*, 2020.

Chen, S. and Zhao, Q. Shallowing deep networks: Layer-wise pruning based on feature representations. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

Chen, T., Goodfellow, I. J., and Shlens, J. Net2net: Accelerating learning via knowledge transfer. In *ICLR*, 2016.

Cubuk, E. D., Zoph, B., Shlens, J., and Le, Q. V. Randaugment: Practical automated data augmentation with a reduced search space. In *CVPR Workshops*, 2020.

Ding, X., Zhang, X., Ma, N., Han, J., Ding, G., and Sun, J. Repvgg: Making vgg-style convnets great again. In *CVPR*, 2021.

Fu, Y., Yang, H., Yuan, J., Li, M., Wan, C., Krishnamoorthi, R., Chandra, V., and Lin, Y. Depthshrinker: A new compression paradigm towards boosting real-hardware efficiency of compact neural networks. In *ICML*, 2022.

He, Y., Kang, G., Dong, X., Fu, Y., and Yang, Y. Soft filter pruning for accelerating deep convolutional neural networks. In *IJCAI*, 2018a.

He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J., and Han, S. Amc: Automl for model compression and acceleration on mobile devices. In *ECCV*, 2018b.

He, Y., Liu, P., Wang, Z., Hu, Z., and Yang, Y. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *CVPR*, 2019.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. In *arXiv:1704.04861*, 2017.

Hu, H., Peng, R., Tai, Y.-W., and Tang, C.-K. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. In *arXiv:1607.03250*, 2016.

Isensee, F., Jaeger, P. F., Kohl, S. A., Petersen, J., and Maier-Hein, K. H. nnu-net: a self-configuring method for deep learning-based biomedical image segmentation. *Nature methods*, 2021.

Jordao, A., Lie, M., and Schwartz, W. R. Discriminative layer pruning for convolutional neural networks. *IEEE Journal of Selected Topics in Signal Processing*, 2020.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. In *ICLR*, 2017.

Liu, Z., Mu, H., Zhang, X., Guo, Z., Yang, X., Cheng, K., and Sun, J. Metapruning: Meta learning for automatic neural network channel pruning. In *ICCV*, 2019.

Müller, R., Kornblith, S., and Hinton, G. E. When does label smoothing help? In *NeurIPS*, 2019.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. In *IJCV*, 2015.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

Shen, M., Yin, H., Molchanov, P., Mao, L., Liu, J., and Alvarez, J. M. Structural pruning via latency-saliency knapsack. In *NeurIPS*, 2022.

Tan, M. and Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019.

Tian, Y., Krishnan, D., and Isola, P. Contrastive multiview coding. In *ECCV*, 2020.

Tiwari, R., Bamba, U., Chavan, A., and Gupta, D. K. Chipnet: Budget-aware pruning with heaviside continuous approximations. In *ICLR*, 2021.

Vanholder, H. Efficient inference with tensorrt. In *GTC*, 2016.

Wang, W., Xie, E., Li, X., Fan, D.-P., Song, K., Liang, D., Lu, T., Luo, P., and Shao, L. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *ICCV*, 2021.

Wei, T., Wang, C., Rui, Y., and Chen, C. W. Network morphism. In *ICML*, 2016.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. Learning structured sparsity in deep neural networks. In *NeurIPS*, 2016.

Xu, P., Cao, J., Shang, F., Sun, W., and Li, P. Layer pruning via fusible residual convolutional block for deep neural networks. *arXiv*, 2020.

You, Z., Yan, K., Ye, J., Ma, M., and Wang, P. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In *NeurIPS*, 2019.

A. Proof

Proposition A.1. *$A[l,t]$,and $S[l,t]$ computed from the DP recurrence relations, Equation (7) are the optimal sets A and S of Equation (6), respectively.*

Proof. For given (l_0, t_0), we suppose for all $l < l_0$ and $t < t_0$, $(A[l,t], S[l,t])$ computed from the DP recurrence, Equation (7) are the optimal (A, S) of Equation (6), respectively. When $(l,t) = (l_0, t_0)$,

$$\begin{aligned}
\sum_{s_{i-1},s_i \in \{0\}\cup S[l_0,t_0]\cup\{l_0\}} T[s_{i-1},s_i] &= \sum_{s_{i-1},s_i \in \{0\}\cup S[k_0,t_0-T_{\text{opt}}[k_0,l_0]]\cup\{k_0\}\cup S_{\text{opt}}[k_0,l_0]\cup\{l_0\}} T[s_{i-1},s_i] && \text{(by Equation (7b))}\\
&= \sum_{s_{i-1},s_i \in \{0\}\cup S[k_0,t_0-T_{\text{opt}}[k_0,l_0]]\cup\{k_0\}} T[s_{i-1},s_i] + \sum_{s_{i-1},s_i \in \{k_0\}\cup S_{\text{opt}}[k_0,l_0]\cup\{l_0\}} T[s_{i-1},s_i]\\
&= \sum_{s_{i-1},s_i \in \{0\}\cup S[k_0,t_0-T_{\text{opt}}[k_0,l_0]]\cup\{k_0\}} T[s_{i-1},s_i] + T_{\text{opt}}[k_0,l_0] && \text{(by Equation (5b))}\\
&< (t_0 - T_{\text{opt}}[k_0,l_0]) + T_{\text{opt}}[k_0,l_0] = t_0, \\
& && \text{(by the optimality assumption for } k_0 < l_0 \text{ and } t_0 - T_{\text{opt}}[k_0,,l_0] < t_0)
\end{aligned}$$

where

$$\begin{aligned}
k_0 = \underset{0\le k'<l}{\text{argmax}} \quad & \sum_{a_{j-1},a_j \in \{0\}\cup A[k',t_0-T_{\text{opt}}[k',l_0]]\cup\{k',l\}} I[a_{j-1},a_j] \\
\text{subject to} \quad & T_{\text{opt}}[0,k'] + T_{\text{opt}}[k',l] < t_0.
\end{aligned}$$

Assume that $(A[l_0,t_0], S[l_0,t_0])$ obtained using Equation (7) are not optimal (A,S) and (A^*, S^*) are the optimal (A,S) of Equation (6) when $(l,t) = (l_0,t_0)$. Then,

$$\sum_{a_{j-1},a_j \in \{0\}\cup A^*\cup\{l_0\}} I[a_{j-1},a_j] > \sum_{a_{j-1},a_j \in \{0\}\cup A[l_0,t_0]\cup\{l_0\}} I[a_{j-1},a_j] \tag{9a}$$

$$\sum_{s_{i-1},s_i \in \{0\}\cup S^*\cup\{l_0\}} T[s_{i-1},s_i] < t_0, \tag{9b}$$

where $A^* \subseteq S^* \subseteq [l_0 - 1]$.

A^* is not an empty set due to Equation (9a) and

$$\begin{aligned}
\sum_{a_{j-1},a_j \in \{0\}\cup A[l_0,t_0]\cup\{l_0\}} I[a_{j-1},a_j] &= \sum_{a_{j-1},a_j \in \{0\}\cup A[k_0,t_0-T_{\text{opt}}[k_0,l_0]]\cup\{k_0,l_0\}} I[a_{j-1},a_j] && \text{(by Equation (7a))}\\
&\ge \sum_{a_{j-1},a_j \in \{0\}\cup A[0,t_0-T_{\text{opt}}[0,l_0]]\cup\{0,l_0\}} I[a_{j-1},a_j] && \text{(by the definition of } k_0)\\
&= \sum_{a_{j-1},a_j \in \{0\}\cup\emptyset\cup\{l_0\}} I[a_{j-1},a_j]]. && \text{(by the base case condition)}
\end{aligned}$$

Then, let k^* be the maximum value of set A^*.

We define $A' = A^* \setminus \{k^*\}$, $S'_{<k^*} = \{s \in S^* \mid s < k^*\}$, and $S'_{>k^*} = \{s \in S^* \mid s > k^*\}$. The upper bound of $T(S'_{<k^*}, 0, k^*)$ is given as follows:

$$\begin{aligned}
\sum_{s_{i-1},s_i \in \{0\}\cup S'_{<k^*}\cup\{k^*\}} T[s_{i-1},s_i] &= \sum_{s_{i-1},s_i \in \{0\}\cup S^*\cup\{l_0\}} T[s_{i-1},s_i] - \sum_{s_{i-1},s_i \in \{k^*\}\cup S'_{>k^*}\cup\{l_0\}} T[s_{i-1},s_i]\\
&\le \sum_{s_{i-1},s_i \in \{0\}\cup S^*\cup\{l_0\}} T[s_{i-1},s_i] - T_{\text{opt}}[k^*,l_0] && \text{(by Equation (5a))}\\
&< t_0 - T_{\text{opt}}[k^*,l_0]. && \text{(by Equation (9b))}
\end{aligned}$$

Therefore, the optimality assumption of $A[k^*, t_0 - T_{\text{opt}}[k^*, l_0]]$ in Equation (6) leads to the inequality:

$$\sum_{a_{j-1}, a_j \in \{0\} \cup A[k^*, t_0 - T_{\text{opt}}[k^*, l_0]] \cup \{k^*\}} I[a_{j-1}, a_j] \geq \sum_{a_{j-1}, a_j \in \{0\} \cup A' \cup \{k^*\}} I[a_{j-1}, a_j]. \tag{10}$$

Thus,

$$\begin{aligned}
\sum_{a_{j-1}, a_j \in \{0\} \cup A[l_0, t_0] \cup \{l_0\}} I[a_{j-1}, a_j] &= \sum_{a_{j-1}, a_j \in \{0\} \cup A[k_0, t_0 - T_{\text{opt}}[k_0, l_0]] \cup \{k_0, l_0\}} I[a_{j-1}, a_j] && \text{(by Equation (7a))} \\
&\geq \sum_{a_{j-1}, a_j \in \{0\} \cup A[k^*, t_0 - T_{\text{opt}}[k^*, l_0]] \cup \{k^*, l_0\}} I[a_{j-1}, a_j] && \text{(by the definition of } k_0\text{)} \\
&= \sum_{a_{j-1}, a_j \in \{0\} \cup A[k^*, t_0 - T_{\text{opt}}[k^*, l_0]] \cup \{k^*\}} I[a_{j-1}, a_j] + I[k^*, l_0] \\
&\geq \sum_{a_{j-1}, a_j \in \{0\} \cup A' \cup \{k^*\}} I[a_{j-1}, a_j] + I[k^*, l_0] && \text{(by Equation (10))} \\
&= \sum_{a_{j-1}, a_j \in \{0\} \cup A' \cup \{k^*, l_0\}} I[a_{j-1}, a_j] = \sum_{a_{j-1}, a_j \in \{0\} \cup A^* \cup \{l_0\}} I[a_{j-1}, a_j].
\end{aligned}$$

This contradicts with Equation (9a). Therefore, our assumption that $(A[l_0, t_0], S[l_0, t_0])$ obtained using DP recurrence relation are not optimal (A, S) of Equation (6) is false. Thus, $(A[l, t], S[l, t])$ are optimal (A, S) of Equation (6). □

Proposition A.2. *$S[l, t]$ computed from the DP recurrence relations, Equation (7) is the optimal S which minimizes the latency of the network when $A[l, t]$ is fixed. Concretely, $S[l, t]$ is the optimal S of the optimization problem:*

$$\min_{A[l,t] \subseteq S \subseteq [l-1]} \sum_{s_{i-1}, s_i \in \{0\} \cup S \cup \{l\}} T[s_{i-1}, s_i]. \tag{11}$$

Proof. When $l = 1$, $S[l, t] = \emptyset$ which satisfies Equation (11) by Equation (7b). For given (l_0, t_0), we suppose for all $l < l_0$ and $t < t_0$, Equation (11) is satisfied. Then, we assume that $S[l_0, t_0]$ obtained using Equation (7b) is not optimal S and S^* are the optimal S of Equation (11) when $(l, t) = (l_0, t_0)$. Then, $A[l_0, t_0] \subseteq S^*$ and

$$\sum_{s_{i-1}, s_i \in \{0\} \cup S[l_0, t_0] \cup \{l_0\}} T[s_{i-1}, s_i] > \sum_{s_{i-1}, s_i \in \{0\} \cup S^* \cup \{l_0\}} T[s_{i-1}, s_i] \tag{12}$$

We can divide two cases whether $A[l_0, t_0]$ is an empty set or not.

Case1: $A[l_0, t_0]$ **is an empty set** $S[l_0, t_0] = S_{\text{opt}}[0, l_0]$ by Equation (7b). Then, $S_{\text{opt}}[0, l_0]$ is the optimal S of Equation (11) when $(l, t) = (l_0, t_0)$ which contradicts with our assumption that $S[l_0, t_0]$ is not optimal S of Equation (11) when $(l, t) = (l_0, t_0)$.

Case2: $A[l_0, t_0]$ **is not an empty set** Let k_0 be the maximum value of set $A[l_0, t_0]$. Then, we define $A' = A^* \setminus \{k_0\}$, $S'_{<k_0} = \{s \in S^* \mid s < k_0\}$, and $S'_{>k_0} = \{s \in S^* \mid s > k_0\}$. By the definition, $A[k_0, t_0 - T_{\text{opt}}[k_0, l_0]] \subseteq S'_{<k_0}$. Then, by the optimality assumption for $k_0 < l_0$ and $t_0 - T_{\text{opt}}[k_0, l_0] < t_0$,

$$\sum_{s_{i-1}, s_i \in \{0\} \cup S'_{<k_0} \cup \{k_0\}} T[s_{i-1}, s_i] \geq \sum_{s_{i-1}, s_i \in \{0\} \cup S[k_0, t_0 - T_{\text{opt}}[k_0, l_0]] \cup \{k_0\}} T[s_{i-1}, s_i]. \tag{13}$$

$$
\begin{aligned}
\sum_{s_{i-1},s_i\in\{0\}\cup S^*\cup\{l_0\}} T[s_{i-1},s_i] &= \sum_{s_{i-1},s_i\in\{0\}\cup S'_{<k_0}\cup\{k_0\}} T[s_{i-1},s_i] + \sum_{s_{i-1},s_i\in\{k_0\}\cup S'_{>k_0}\cup\{l_0\}} T[s_{i-1},s_i] \\
&\geq \sum_{s_{i-1},s_i\in\{0\}\cup S[k_0,t_0-T_{\text{opt}}[k_0,l_0]]\cup\{k_0\}} T[s_{i-1},s_i] + \sum_{s_{i-1},s_i\in\{k_0\}\cup S'_{>k_0}\cup\{l_0\}} T[s_{i-1},s_i] \\
&\qquad\qquad \text{(by Equation (13))} \\
&\geq \sum_{s_{i-1},s_i\in\{0\}\cup S[k_0,t_0-T_{\text{opt}}[k_0,l_0]]\cup\{k_0\}} T[s_{i-1},s_i] + \sum_{s_{i-1},s_i\in\{k_0\}\cup S_{\text{opt}}\cup\{l_0\}} T[s_{i-1},s_i] \\
&\qquad\qquad \text{(by Equation (5b))} \\
&\geq \sum_{s_{i-1},s_i\in\{0\}\cup S[k_0,t_0-T_{\text{opt}}[k_0,l_0]]\cup\{k_0\}\cup S_{\text{opt}}\cup\{l_0\}} T[s_{i-1},s_i] \\
&= \sum_{s_{i-1},s_i\in\{0\}\cup S[l_0,t_0]\cup\{l_0\}} T[s_{i-1},s_i]. \qquad \text{(by Equation (7b))}
\end{aligned}
$$

This contradicts with Equation (12). Therefore, our assumption that $S[l_0, t_0]$ obtained using DP recurrence relation is not optimal S of Equation (11) when $(l, t) = (l_0, t_0)$ is false. Thus, $S[l, t]$ is optimal S of Equation (11). □

B. Measuring the Importance

Algorithm 3 Finding Optimal Importance with DP

input I
  Initialize $I_{\text{opt}}[k,l] \leftarrow 0$, $B_{\text{opt}}[k,l] \leftarrow \emptyset$ for $0 \le k \le l \le L$
  for $l = 1$ **to** L **do**
    for $k = 0$ **to** $l-1$ **do**
      for (a,b) **in** $[(0,0),(0,1),(1,0),(1,1)]$ **do**
        $m \leftarrow \underset{k\le m'<l}{\text{argmax}} \left(I_{\text{opt}}[k,m',a,0] + I[m',l,0,b]\right)$
        $I_{\text{opt}}[k,l,a,b] \leftarrow I_{\text{opt}}[k,m',a,0] + I[m',l,0,b]$
        if $m \notin \{k\}$ **then**
          $B_{\text{opt}}[k,l] \leftarrow B_{\text{opt}}[k,m] \cup \{m\}$
        end if
      end for
    end for
  end for
output I_{opt}, B_{opt}

Algorithm 4 Solving the Extended Surrogate Objective

input T_0, L, T, I
  Initialize $D[l,t,a] \leftarrow 0$, $A[l,t] \leftarrow \emptyset$, $S[l,t] \leftarrow \emptyset \quad \forall t,l$
  $T_{\text{opt}}, S_{\text{opt}} \leftarrow$ Algorithm 1 (T, L)
  $I_{\text{opt}}, B_{\text{opt}} \leftarrow$ Algorithm 3 (I)
  for $l = 1$ **to** L **do**
    for $t = T_{\text{opt}}[0,l] + 1$ **to** T_0 **do**
      for $a = 0$ **to** 1 **do**
        $k, \alpha \leftarrow \underset{\substack{0\le k'<l \\ \alpha'\in\{0,1\}}}{\text{argmax}} \left(D\left[k',\alpha',t-T_{\text{opt}}[k',l]\right] + I_{\text{opt}}(k',l,\alpha',a)\right)$
          subject to $T_{\text{opt}}[0,k'] + T_{\text{opt}}[k',l] < t$
        $t_{\text{last}} \leftarrow T_{\text{opt}}[k,l]$
        $D[l,t,a] \leftarrow D[k,t-t_{\text{last}}] + I_{\text{opt}}[k,l,\alpha,a]$
        $A[l,t,a] \leftarrow A[k,t-t_{\text{last}},a] \cup \{k : k>0 \wedge \alpha = 1\}$
        $S[l,t] \leftarrow S[k,t-t_{\text{last}}] \cup \{k : k>0\} \cup S_{\text{opt}}[k,l]$
        $B[l,t] \leftarrow B[k,t-t_{\text{last}}] \cup \{k : k>0\} \cup B_{\text{opt}}[k,l]$
      end for
    end for
  end for
  $a_{\text{last}} \leftarrow \underset{a\in\{0,1\}}{\text{argmax}}(A[L,T_0,a])$
output $A[L,T_0,a_{\text{last}}]$, $S[L,T_0]$, $B[L,T_0]$

B.1. Extension of Importance and Improved Surrogate

Vanilla MobileNetV2 includes blocks which have the identity functions as activation layers at the end of the block. In light of this, we redefine the importance of contiguous network blocks to consider whether activation layers at the boundary are identity functions or not. To be more specific, we considered the importance of blocks only when activation layers are not identity functions at the boundary in the main paper. Here, we also consider the importance of contiguous network blocks when one of the activation layers at the boundary is the identity function.

For the contiguous network blocks from i+1-th layer to j-th layer, we define the discrete variables $a, b \in \{0, 1\}$ to indicate whether the first activation layer and the last activation layer of the contiguous network blocks are identity functions or not, respectively. Then, we redefine the importance of contiguous network blocks from $i+1$-th layer to j-th layer as

$I[i, j, a, b] \quad (a, b \in \{0, 1\})$. Concretely, we redefine the importance as follows:

$$I[l, k, a, b] \coloneqq \max_{\theta} \operatorname{Acc}\left(\underbrace{\bigcirc_{l=j+1}^{L} \sigma_l \circ f_{\theta_l}}_{j+1 \text{ to} L \text{ layers}} \circ (b\sigma + (1-b)\mathrm{id}) \circ \underbrace{\bigcirc_{l=i+1}^{j} f_{\theta_l}}_{i+1 \text{ to } j \text{ layers}} \circ (a\sigma + (1-a)\mathrm{id}) \circ \underbrace{f_{\theta_i} \circ \bigcirc_{l=1}^{i-1} \sigma_l \circ f_{\theta_l}}_{1 \text{ to } i \text{ layers}}\right) - \max_{\theta} \operatorname{Acc}\left(\bigcirc_{l=1}^{L} \sigma_l \circ f_{\theta_l}\right), \tag{14}$$

where σ is the activation layer that is not an identity function.

Due to the redefinition of importance, we propose an alternative surrogate for objective in Equation (1a) as follows:

$$\mathcal{I}(A, B) \coloneqq \sum_{b_{j-1}, b_j \in \{0\} \cup B \cup \{L\}} I[b_{j-1}, b_j, \mathbb{1}_A(b_{j-1}), \mathbb{1}_A(b_j)], \tag{15}$$

where $A \subseteq B \subseteq [L-1]$. A denotes the positions of activations and B denotes the boundary point of the contiguous network block for objective approximation. Then, the objective extends to

$$\begin{aligned} &\underset{A \subseteq B, S \subseteq [L-1]}{\text{maximize}} \sum_{b_{j-1}, b_j \in \{0\} \cup B \cup \{L\}} I[b_{j-1}, b_j, \mathbb{1}_A(b_{j-1}), \mathbb{1}_A(b_j)] \\ &\text{subject to} \sum_{s_{i-1}, s_i \in \{0\} \cup S \cup \{L\}} T[s_{i-1}, s_i] < T_0. \end{aligned} \tag{16}$$

Note, Equation (16) can be exactly solved with DP algorithm as described in Algorithm 4.

B.2. Possible Combinations of Blocks

It is unscalable to consider every possible configuration of blocks in Appendix B.1. Therefore in practice, we regularize the type of blocks we consider when we solve Equation (16) through Algorithm 4. In particular, when we implement our algorithm in MobileNetV2, we consider the blocks without the activation only if the corresponding boundary position doesn't have an activation layer in the original network. If both boundary positions do not have activation layer in the original network, we further do not consider the ones that don't have activation at the end.

Furthermore, we only take blocks that we can merge into a single layer, thus the skip-connections in MobileNetV2 considerably reduce the number of possible blocks. We also do not implement merging the series of layers that starts with stride 2 convolutional layer, because it results in significant increase in kernel size (Fu et al., 2022). In MobileNetV2, we have 171 different blocks to measure the latency ($T[l, k]$) and 315 different blocks to measure the importance ($I[l, k, a, b]$).

B.3. Calculating and Normalizing the Importance

We evaluate the importance by approximating the first term in Equation (14) with the accuracy attained after training the network for a few epochs starting from the pretrained weight. In MobileNetV2, we approximate the first term in Equation (14) by training it for a single epoch. If the block's size is 1 (*i.e.*, $k - l = 1$), we re-initialize the corresponding block and measure the accuracy drop after training from the pretrained weight.

When we approximate the performance of the network trained until convergence in Equation (14) with the accuracy attained after training it for a few epochs, we tend to calculate a lower importance value than the actual definition of the importance value. This effect is reflected independently for each block; thus, the more block we construct the network with, the more we underestimate the importance of the network. Therefore, it is crucial to normalize the importance values by adding an appropriate value to each block in order to address this issue. Specially, we compute the average of the importance of the contiguous network blocks and subtract the average from the importance of contiguous network blocks. Please refer to the attached code for details.

Table 4. Accuracy and latency of compressed architectures applied to MobileNetV2-1.0 and MobileNetV2-1.4 on ImageNet-100 dataset. The latency of the compressed network architecture is measured on *TITAN Xp*, *RTX 2080 Ti*, *RTX 3090*, and *Tesla V100*.

(a) MobileNetV2-1.0

Network	Acc (%)	TensorRT Latency (ms) *TITAN Xp*	*RTX 2080 Ti*	*RTX 3090*	*Tesla V100*	*w/o* TensorRT (ms) *RTX 2080 Ti*
MobileNetV2-1.0	87.58	27.58	19.55	14.38	17.17	40.68
MBV2-DS-A	85.60	20.14	15.01	10.51	13.38	27.71
MBV2-DS-B	85.16	18.73	12.59	9.53	11.57	23.09
Ours	**85.98**	**17.71**	**12.49**	**9.35**	11.58	**22.46**
MBV2-DS-C	84.30	15.96	11.39	8.51	10.36	20.91
Ours	**84.36**	**15.32**	**10.50**	**7.88**	**10.14**	**18.39**
MBV2-DS-D	83.18	14.68	10.65	7.89	10.00	18.91
Ours	**84.00**	**13.71**	**9.92**	**7.43**	**9.87**	**17.74**

(b) MobileNetV2-1.4

Network	Acc (%)	TensorRT Latency (ms) *TITAN Xp*	*RTX 2080 Ti*	*RTX 3090*	*Tesla V100*	*w/o* TensorRT (ms) *RTX 2080 Ti*
MobileNetV2-1.4	88.88	42.66	30.47	22.01	26.19	61.83
MBV2-1.4-DS-A	85.58	27.40	19.92	14.51	17.95	35.28
MBV2-1.4-DS-B	84.98	25.67	19.52	14.15	17.77	31.79
Ours	**86.16**	**25.44**	**18.74**	**13.44**	18.03	**31.22**
MBV2-1.4-DS-C	84.00	23.99	17.76	12.93	16.21	29.87
Ours	**85.04**	**23.32**	**16.79**	**11.95**	**15.58**	**27.92**
MBV2-1.4-DS-D	83.02	23.00	17.77	12.81	15.87	28.09
MBV2-1.4-DS-E	81.80	21.17	15.93	11.64	14.59	26.14
Ours	**84.50**	**19.78**	**14.94**	**10.20**	**13.86**	**23.26**

C. Additional Experiments

C.1. ImageNet-100

Reproducing Search Phase of DepthShrinker We implement the search phase of the baseline method on the ImageNet-100 dataset and compare it with our depth compression method. We reproduce the search phase of DepthShrinker on top of the ImageNet-100 dataset and search the patterns that match the compression ratio in the original paper (Fu et al., 2022).

In MobileNetV2-1.0, we sweep through the number of activated blocks among 12, 9, and 7 and denote them 'DS-AR-1.0', 'DS-BR-1.0', and 'DS-CR-1.0', respectively. In MobileNetV2-1.4, we sweep through the number of activated blocks among 11, 8, and 6 and name them 'DS-AR-1.4', 'DS-BR-1.4', and 'DS-CR-1.4', respectively. Table 6a and Table 6b summarize the results of comparing our method to the reproduced result of DepthShrinker for MobileNetV2-1.0 and MobileNetV2-1.4 on the ImageNet-100 dataset, respectively. Our method outperforms the baseline performance regardless of the type of network and compression ratio.

Inference Time Transfer Results on Different GPUs We further present the results of measuring the end-to-end inference time of discovered networks in MobileNetV2-1.0 and MobileNetV2-1.4 on different GPU platforms. We report the performance on *TITAN Xp*, *RTX 2080 Ti*, *RTX 3090*, and *Tesla V100*. Table 4a and Table 4b summarize the accuracy and the latency of the compressed networks. Our method outperforms the baseline in the majority of the GPUs.

Table 5. Accuracy and latency of compressed architectures applied to MobileNetV2-1.4 on top of the ImageNet dataset.The latency of the compressed network architecture is measured on *TITAN Xp*, *RTX 2080 Ti*, *RTX 3090*, and *Tesla V100*.

		TensorRT Latency (ms)				*w/o* TensorRT (ms)
Network	Acc (%)	*TITAN Xp*	*RTX 2080 Ti*	*RTX 3090*	*Tesla V100*	*RTX 2080 Ti*
MobileNetV2-1.0	72.30	27.68	19.65	14.28	17.23	40.84
MBV2-DS-A	72.43	20.19	15.05	10.54	13.46	27.76
Ours	**72.67**	**19.70**	**13.94**	**10.53**	**12.74**	**25.25**
MBV2-DS-B	71.54	18.63	12.60	9.50	11.67	23.11
Ours	**71.90**	**18.42**	**12.36**	**9.44**	**11.43**	**21.95**
MBV2-DS-C	70.90	16.03	11.43	8.52	10.38	21.06
Ours	**71.29**	**15.76**	**11.10**	**8.13**	**10.18**	**20.03**
MBV2-DS-D	69.40	14.95	10.72	7.94	9.89	19.03
Ours	**70.61**	**14.55**	**10.08**	**7.27**	9.94	**16.77**

Table 6. Accuracy and latency of compressed architectures applied to MobileNetV2-1.0 and MobileNetV2-1.4 on ImageNet-100 dataset. Both compression methods use the latency information of *RTX 2080 Ti* and the latency is measured on the same *RTX 2080 Ti*.

(a) MobileNetV2-1.0

Network	Acc (%)	TensorRT Lat. (ms)	*w/o* TensorRT Lat. (ms)
MBV2-1.0	87.58	19.55	40.68
DS-AR-1.0	85.46	13.32	24.48
Ours	**85.98**	**12.49**	**22.46**
DS-BR-1.0	83.92	11.86	20.96
Ours	**84.36**	**10.50**	**18.39**
DS-CR-1.0	83.24	10.82	17.95
Ours	**84.00**	**9.92**	**17.74**

(b) MobileNetV2-1.4

Network	Acc (%)	TensorRT Lat. (ms)	*w/o* TensorRT Lat. (ms)
MBV2-1.4	88.88	30.47	61.83
DS-AR-1.4	85.88	20.78	36.50
DS-BR-1.4	85.02	18.85	31.89
Ours	**86.16**	**18.74**	**31.22**
DS-CR-1.4	84.86	17.44	28.14
Ours	**85.04**	**16.79**	**27.92**

C.2. ImageNet

Inference Time Transfer Results on Different GPUs We present the results of the latency of the compressed network on ImageNet dataset across various GPU platforms including *TITAN Xp*, *RTX 2080 Ti*, *RTX 3090*, and *Tesla V100*. Table 5 demonstrates that our method outperforms the baseline in the majority of the settings, which is consistent with the previous results.

D. Merging Convolutional Layers in Modern CNN

We address the details to apply the merging for the convolution operations in modern CNNs with skip addition and padding. Consider a skip addition, $f(x) + x$ where $f(\cdot)$ is a network block and X is an input feature map. When $f(\cdot)$ is a single convolution operation, $f(x)+x$ can be replaced by an equivalent convolution operation (Ding et al., 2021). In light of this, our method fuses the skip addition into $f(\cdot)$ only if $f(x)$ is merged into a single convolution operation.

DepthShrinker's scope of merging convolution operations is restricted to cases where the kernel size of at least one of the convolution operations to be merged is 1 (Fu et al., 2022). To include more general cases of merging where the kernel size of both convolution operations is greater than 1, we need to address the details of padding. In this paper, we limit our considerations to zero padding for the exact merging and apply sufficient zero padding to prevent the computation disparities at the boundaries before and after merging.

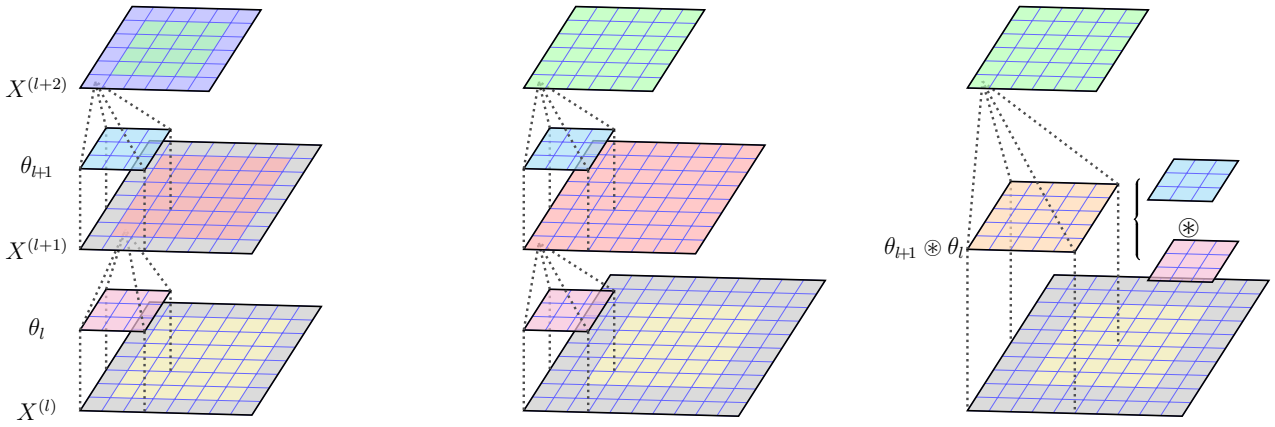

Figure 5. Illustration comparing the output from two different types of padding applied to two consecutive 3×3 convolution operations with the output from a merged 5×5 convolution operation. The boundary of the output feature map obtained from applying zero padding of size 1 before each 3×3 convolution is distinct from that of the output feature map obtained from the merged 5×5 convolution. Conversely, if zero padding of size 2 is applied to the first 3×3 convolution, the output feature map is equivalent to the output feature map obtained from the merged 5×5 convolution.

Consider a feature map $X^{(l)}$, upon which two consecutive 3×3 convolution operations, utilizing kernels θ_l and θ_{l+1}, are applied to produce an output feature map $X^{(l+2)}$. The output generated by the first convolution operation utilizing kernel θ_l is denoted as $X^{(l+1)}$. As shown in Figure 5, when zero padding of size 1 is applied prior to each of the 3×3 convolution operations, the boundary of the output resulting from the merged 5×5 convolution operation, utilizing kernel $\theta_{l+1} \circledast \theta_l$ differs from that of $X^{(l+2)}$. Insufficient zero padding results in a computation skip at the boundary of $X^{(l+1)}$, which in turn leads to a discrepancy between the computation at the boundary of $X^{(l+2)}$ and the output feature map of the merged 5×5 convolution operation. Conversely, when zero padding of size 2 is applied prior to the first 3×3 convolution operation, the output feature map of the two consecutive 3×3 convolution operations is equivalent to the output feature map of the merged 5×5 convolution operation.