Optimal channel selection with discrete QCQP

Yeonwoo Jeong[*1] **Deokjae Lee**[*1] **Gaon An** [1] **Changyong Son** [2] **Hyun Oh Song** [†1]
[1] Department of Computer Science and Engineering, Seoul National University
[2]Samsung Advanced Institute of Technology

Abstract

Reducing the high computational cost of large convolutional neural networks is crucial when deploying the networks to resource-constrained environments. We first show the greedy approach of recent channel pruning methods ignores the inherent quadratic coupling between channels in the neighboring layers and cannot safely remove inactive weights during the pruning procedure. Furthermore, due to these inactive weights, the greedy methods cannot guarantee to satisfy the given resource constraints and deviate with the true objective. In this regard, we propose a novel channel selection method that optimally selects channels via discrete QCQP, which provably prevents any inactive weights and guarantees to meet the resource constraints *tightly* in terms of FLOPs, memory usage, and network size. We also propose a quadratic model that accurately estimates the *actual inference time* of the pruned network, which allows us to adopt inference time as a resource constraint option. Furthermore, we generalize our method to extend the selection granularity beyond channels and handle non-sequential connections. Our experiments on CIFAR-10 and ImageNet show our proposed pruning method outperforms other fixed-importance channel pruning methods on various network architectures.

1 Introduction

Deep neural networks are the bedrock of artificial intelligence tasks such as object detection, speech recognition, and natural language processing (Redmon and Farhadi, 2018; Chorowski et al., 2015; Devlin et al., 2019). While modern networks have hundreds of millions to billions of parameters to train, recent works show that these parameters are highly redundant and can be pruned without significant loss in accuracy (Han et al., 2015; Guo et al., 2016). This discovery has led practitioners to desire training and running the models on resource-constrained mobile devices, provoking a large body of research on network pruning.

Unstructured pruning, however, does not directly lead to any practical acceleration or memory footprint reduction due to poor data locality (Wen et al., 2016), and this motivated research on structured pruning to achieve practical usage under limited resource budgets. To this end, a line of research on channel pruning considers completely pruning the convolution filters along the input and output channel dimensions, where the resulting pruned model becomes a smaller dense network suited for practical acceleration and memory footprint reduction (Li et al., 2017; Luo et al., 2017; He et al., 2019; Wen et al., 2016; He et al., 2018a).

While many channel pruning methods select channels greedily, which is easy to model and optimize, they cannot safely remove inactive weights during the pruning procedure. As a result, these greedy approaches suffer from discrepancies with the true objective and cannot strictly satisfy the required resource constraints during the pruning process.

The ability to specify hard target resource constraints into the pruning optimization process is important since this allows the user to run the pruning and optional finetuning process only once. When the pruning process ignores the target specifications, the users may need to apply multiple rounds of pruning and finetuning until the specifications are eventually met, resulting in an extra computation overhead (Han et al., 2015; He et al., 2018a; Liu et al., 2017).

Our contributions can be summarized as follows:

- We propose an optimal channel selection method

Proceedings of the 25[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2022, Valencia, Spain. PMLR: Volume 151. * Equal contribution. † Corresponding author.

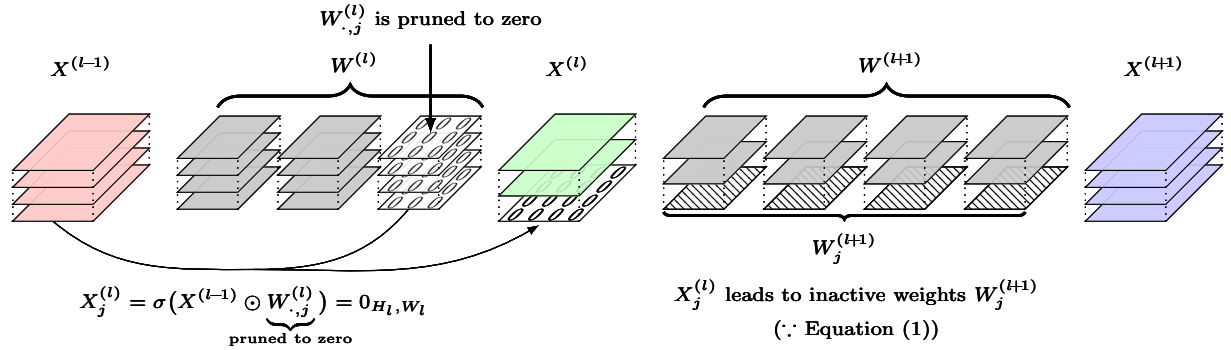


Figure 1: Illustration of a channel pruning procedure that leads to inactive weights. When j-th output channel of l-th convolution weights $W^{(l)}_{\cdot,j}$ is pruned, *i.e.* $W^{(l)}_{\cdot,j} = 0_{C_{l-1},K_l,K_l}$, then the j-th feature map of l-th layer $X^{(l)}_j$ should also be 0. Consequently, $X^{(l)}_j$ yields inactive weights $W^{(l+1)}_j$. Note that we use $W^{(l)}_{\cdot,j}$ to denote the tensor $W^{(l)}_{\cdot,j,\cdot,\cdot}$, following the indexing rules of NumPy (Van Der Walt et al., 2011).

which satisfies the user-specified constraints *tightly* in terms of FLOPs, memory usage, and network size, and directly maximizes the importance of neurons in the pruned network.

- We propose a new quadratic model that accurately estimates the *inference time* of a pruned network *without direct deployment*.
- We extend our method to increase the pruning granularity beyond channels and simultaneously prune channels and spatial patterns in the individual 2D convolution filters.
- We generalize our method to handle nonsequential connections (skip additions and skip concatenations).

Our experiments on CIFAR-10 and ImageNet datasets show the state of the art results compared to other fixed-importance channel pruning methods.

2 Motivation

In this section, we first discuss the motivation of our method more concretely. The weights of a sequential CNN can be expressed as a sequence of 4-D tensors, $W^{(l)} \in \mathbb{R}^{C_{l-1}\times C_l\times K_l\times K_l}$ $\forall l \in [L]$, where C_{l-1}, C_l, and K_l represent the number of input channels, the number of output channels, and the filter size of the l-th convolution weight tensor, respectively. We denote the feature map after the l-th convolution as $X^{(l)} \in \mathbb{R}^{C_l\times H_l\times W_l}$. Concretely, $X^{(l)}_j = \sigma(X^{(l-1)} \odot W^{(l)}_{\cdot,j}) = \sigma(\sum_{i=1}^{C_{l-1}} X^{(l-1)}_i * W^{(l)}_{i,j})$ for $j \in [C_l]$, where σ is the activation function, $*$ denotes 2-D convolution operation, and $\odot$ denotes the sum of channel-wise 2-D convolutions. Now consider pruning these weights in channel-wise direction. We show that with naive channel-wise pruning methods, we cannot exactly specify the target resource constraints due to unpruned inactive weights and deviate away from the true objective by ignoring quadratic coupling between channels in the neighboring layers.

2.1 Inactive weights

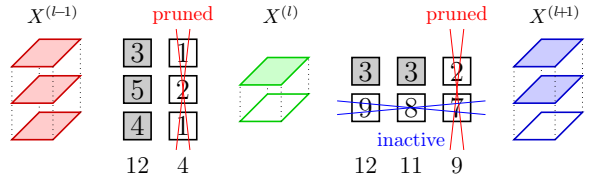
According to Han et al. (2015), network pruning produces dead neurons with zero input or output connections. These dead neurons cause *inactive* weights[1], which do not affect the final output activations of the pruned network. These inactive weights are not excluded automatically through the standard pruning procedure and require additional post-processing which relies on ad-hoc heuristics. For example, Figure 1 shows a standard channel pruning procedure that deletes weights across the output channel direction but fails to prune the inactive weights. Concretely, deletion of weights on j-th output channel of l-th convolution layer leads to $W^{(l)}_{\cdot,j} = 0_{C_{l-1},K_l,K_l}$. Then, $X^{(l)}_j$ becomes a dead neuron since $X^{(l)}_j = \sigma(X^{(l-1)} \odot W^{(l)}_{\cdot,j}) = \sigma(\sum_{i=1}^{C_{l-1}} X^{(l-1)}_i * W^{(l)}_{i,j}) = 0_{H_l,W_l}$.

The convolution operation on the dead neuron results in a trivially zero output as below:

$$X^{(l+1)}_p = \sigma\left(\sum_{i=1}^{C_l} X^{(l)}_i * W^{(l+1)}_{i,p}\right) = \sigma\left(\sum_{i=1}^{C_l} \mathbb{1}_{i\neq j} X^{(l)}_i * W^{(l+1)}_{i,p} + \underbrace{\underbrace{X^{(l)}_j}_{\text{dead}} * \underbrace{W^{(l+1)}_{j,p}}_{\text{inactive}}}_{=0_{H_{l+1},W_{l+1}}}\right). \quad (1)$$

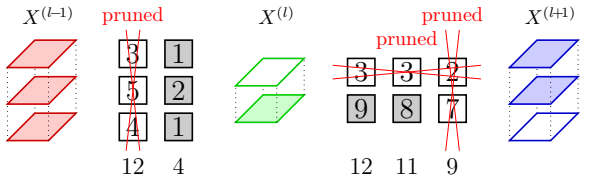
[1]Rigorous mathematical definition of inactive weights is provided in Supplementary material C.

Equation (1) shows that the dead neuron $X_j^{(l)}$ causes weights $W_{j,p}^{(l+1)}, \forall p \in [C_{l+1}]$ to be inactive. Such inactive weights do not account for the actual resource usage, even when they remain in the pruned network, which prevents the exact modeling of the user-specified resource constraints (FLOPs, memory usage, or network size). Furthermore, inactive weights unpruned during the pruning procedure becomes a bigger problem for nonsequential convolutional networks due to their skip connections, which is discussed in Section 3.4. To address thess problems, we introduce a quadratic optimization-based algorithm that provably eliminates all the inactive weights during the pruning procedure.



Objective value : $(3+5+4)+(3+3)=18$

(a) Greedy



Objective value : $(1+2+1)+(9+8)=21$

(b) Ours

Figure 2: A comparison of the greedy channel selection method and our selection method. Parallelograms represent feature maps and squares represent 2-D filters of convolution weights. Gray squares are filters which account for the objective. The numbers on each squares represent the absolute sum of weights in the filter.

2.2 Quadratic coupling

Most of the existing channel pruning methods select channels with regard to their importance. However, measuring a channel's contribution to the network should also take into account the channels in the neighboring layers, as illustrated in Figure 2. In the example, we define the importance of a channel as the absolute sum of the weights in the channel, as in Li et al. (2017), and assume the objective is to maximize the absolute sum of the weights in the whole pruned network, excluding the inactive weights. We compare two different channel selection methods: (a) a standard greedy channel selection method which greedily selects each channel in layerwise manner, and (b) our selection method that optimally considers the effect of the channels in neighboring layers, which will be described in Section 3. As a result of running each selection algorithms, (a) will prune the second output channel of the first convolution and the third output channel of the second convolution, and (b) will prune the first output channel of the first convolution, the third output channel of the second convolution, and the first input channel of the second convolution. The objective values for each pruned networks are (a) 18 and (b) 21, respectively.

This shows that the coupling effect of the channels in neighboring layers directly affects the objective values, and results in a performance gap between (a) and (b). We call this coupling relationship as the *quadratic coupling* between the neighboring layers and formulate the contributions to the objective by quadratic terms of neighboring channel activations. To address this quadratic coupling, we propose a channel selection method based on the QCQP (Quadratic Constrained Quadratic Program) with importance evaluation respecting both the input and the output channels.

3 Method

We first propose our discrete QCQP formulation of channel pruning for sequential convolutional neural networks (CNNs). Then, we present two extended versions of our formulation which can handle 1) joint channel and shape pruning of 2D convolution filters and 2) nonsequential connections, respectively.

3.1 Channel pruning for sequential CNNs

To capture the importance of weights in $W^{(l)}$, we define the *importance tensor* as $I^{(l)} \in \mathbb{R}_+^{C_{l-1} \times C_l \times K_l \times K_l}$. Following the protocol of Han et al. (2015); Guo et al. (2016), we set $I^{(l)} = \gamma_l |W^{(l)}|$ where γ_l is the ℓ_2 normalizing factor in l-th layer or $\|\mathrm{vec}(W^{(l)})\|^{-1}$. Then, we define the binary pruning mask as $A^{(l)} \in \{0,1\}^{C_{l-1} \times C_l \times K_l \times K_l}$. For channel pruning in sequential CNNs, we define *channel activation* $r^{(l)} \in \{0,1\}^{C_l}$ to indicate which indices of channels remain in the l-th layer of the pruned network. Then, the weights in $W_{i,j}^{(l)}$ are active if and only if $r_i^{(l-1)} r_j^{(l)} = 1$, which leads to $A_{i,j}^{(l)} = r_i^{(l-1)} r_j^{(l)} J_{K_l}$. For example, in Figure 2b, $r^{(l-1)} = [1,1,1]^\intercal$, $r^{(l)} = [0,1]^\intercal$, and $r^{(l+1)} = [1,1,0]^\intercal$, therefore, $A^{(l)} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \otimes J_{K_l}$ and $A^{(l+1)} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} \otimes J_{K_{l+1}}$ [2].

Our goal is to directly maximize the sum of the importance of active weights after the pruning procedure

[2] $\otimes$ denotes the outer product of tensors and J_n is a n-by-n matrix of ones.

under given resource constraints such as FLOPs, memory usage, network size, or inference time. Concretely, our optimization problem forms as

$$\begin{aligned} \underset{r^{(0:L)}}{\text{maximize}} \quad & \sum_{l=1}^{L} \left\langle I^{(l)}, A^{(l)} \right\rangle \\ \text{subject to} \quad & \sum_{l=0}^{L} a_l \left\| r^{(l)} \right\|_1 + \sum_{l=1}^{L} b_l \left\| A^{(l)} \right\|_1 \leq M \\ & A^{(l)} = r^{(l-1)} r^{(l)\intercal} \otimes J_{K_l} \quad \forall l \in [L]. \end{aligned} \tag{2}$$

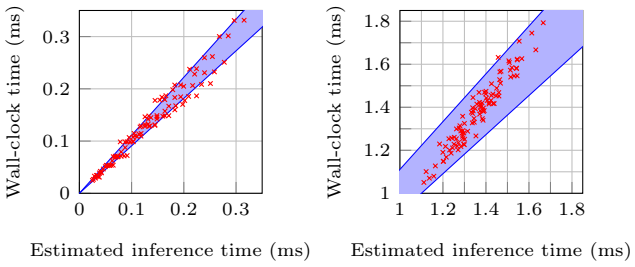
In our formulation, we can exactly compute the actual FLOPS, memory usage, and network size of the pruned network. Furthermore, we can estimate the inference time of the pruned network by modeling it with the number of channels ($= \|r^{(l)}\|_1$) and the pruning mask sparsity ($= \|A^{(l)}\|_1$) in each layer, which will be explained in Section 3.2.

The left hand side of the inequality in the first constraint of Equation (3) indicates the actual (or estimated) resource usage. Table 1 shows the a_l and b_l terms used for computing the usage of each resource. For the l-th convolution layer, its network size is equal to the number of parameters in the layer, which is $\|A^{(l)}\|_1$ ($a_l = 0,\ b_l = 1$). Memory resource implies the memory used during inference, which is the sum of the memory required for the input feature map, $H_{l-1}W_{l-1}\|r^{(l-1)}\|_1$, and the number of parameters, $\|A^{(l)}\|_1$ ($a_{l-1}{=}H_{l-1}W_{l-1}$, $b_l{=}1$). FLOPs indicates the sum of the number of multiplications for each parameter. Since each parameter requires H_lW_l multiplications, FLOPs equals to $H_lW_l\|A^{(l)}\|_1$ ($a_l{=}0$, $b_l{=}H_lW_l$).

The optimization problem Equation (3) is a discrete nonconvex QCQP of the channel activations $[r^{(0)}, \ldots, r^{(L)}]$, where the objective, which is the same with the objective in Section 2.2, respects the quadratic coupling of channel activations ($= r^{(l)}$). Please refer to Supplementary material B for the details on the standard QCQP form of Equation (3).

Table 1: Resource constraints with the corresponding a_l and b_l values. β_l and δ_l are device-specific, and determined via least square regression in Section 3.2.

Resource constraint (M)		a_l	b_l
Network size	exact	0	1
Memory	exact	H_lW_l	1
FLOPs	exact	0	H_lW_l
Inference time	approx.	β_{l+1}	$\frac{\delta_l}{K_l^2}$



(a) Conv (kernel=3, stride=1) and input (56×56)

(b) ResNet50 and input (224×224)

Figure 3: Estimated inference time vs. actual wall-clock time of a single convolution operation (a) and a ResNet-50 (b) while varying the number of channels. The blue area indicates the area where the error of the estimated value is under 10% with respect to the actual wall-clock time. The wall-clock time is measured on a machine with Intel Xeon E5-2650 CPU and Titan XP GPU. Additional experiment results are provided in Supplementary material D.

3.2 Inference time constraint

Unlike other resource constraints, inference time on the edge-device can only be measured by running each pruned network on the real device. However, deploying every candidate networks on the device during pruning can be prohibitive. Therefore, we propose an approximate method that accurately predicts the inference time of the pruned networks efficiently.

We utilize the fact that the computation cost of a pruned network is highly dependent to the compution cost of convolution operations with the pruned channels. We first build a quadratic model with respect to the number of unpruned channels, $\|r^{(l)}\|_1$, to estimate the inference time of each convolution operation. Then, we estimate the inference time of the pruned network by integrating these estimate values.

Concretely, we model the inference time of the l-th layer convolution operation with respect to the number of its input channels ($= \|r^{(l-1)}\|_1$) and the output channels ($= \|r^{(l)}\|_1$). Using these variables, we aim to model three major factors of the convolution operation that affect the inference time: 1) FLOPs (computation cost), 2) MAC (memory access cost), and 3) bias overhead. The resulting estimation model for the l-th convolutional operation is $\alpha_l + \beta_l\|r^{(l-1)}\|_1 + \delta_l\|r^{(l-1)}\|_1\|r^{(l)}\|_1$, where each terms represent the contribution of bias overhead, MAC, and FLOPs to the inference time. More discussions for choosing this model and analysis of the coefficients α_l, β_l, and δ_l are provided in Supplementary material D. Note that α_l, β_l, and δ_l are dependent on the edge-device. Therefore, we find the best α_l,

β_l, and δ_l values via least square regression on a few samples of $(\|r^{(l-1)}\|_1, \|r^{(l)}\|_1, \text{WallClock}^{(l)}(r^{(l-1)}, r^{(l)}))$, where $\text{WallClock}^{(l)}$ denotes the wall-clock time of the l-th convolution operation with $\|r^{(l-1)}\|_1$ input channels and $\|r^{(l)}\|_1$ output channels. Finally, we plug the learned quadratic model into the left hand side of the inequality in Equation (3) [3]. In this paper, we measure the actual wall-clock time using Nimble framework (Kwon et al., 2020) on CUBLAS backend.

Figure 7 shows the prediction performance of our quadratic model. The red points represent the estimated inference time versus the actual inference time of a single convolution operation (a) and a ResNet-50 network (b) while varying the number of channels. Our model successfully estimates the actual inference time of single convolution operations with 8% error rate[4] on average. Also, we estimate the inference time of the whole network by summing all of the estimated inference time of its convolution operations, and achieve 3% error rate on average. Note that as we gather the inference time of the convolution operations, the variance of the inference time decreases by the law of large number and the error rate simply decreases.

3.3 Joint channel and spatial pruning

For further efficiency, we increase the pruning granularity and jointly perform spatial pruning to 2-D convolution filters. Concretely, we prune by each weight tensor across the input channel direction additionally to perform channel and spatial pruning processes simultaneously.

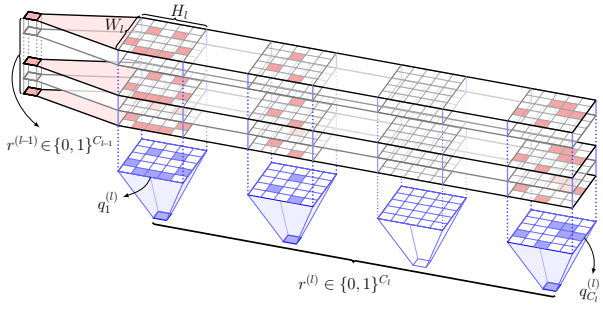


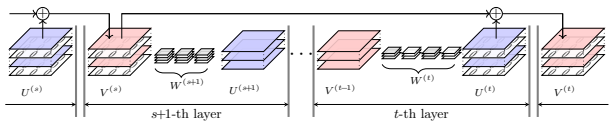
Figure 4: Input channel activation ($= r^{(l-1)}$), shape column activation ($= q^{(l)}$), and the corresponding mask ($= A^{(l)}$) for l-th convolution layer, where $A^{(l)} = r^{(l-1)} \otimes q^{(l)}$.

First, we define the *shape column* $W^{(l)}_{\cdot,j,a,b}$ by the vector of weights at spatial position (a, b) of a 2-D convolution filter along the j-th output channel dimension. Then, we define *shape column activation* $q^{(l)} \in \{0,1\}^{C_l \times K_l \times K_l}$ to indicate which shape columns in the l-th convolution layer remain in the pruned network. Figure 4 shows the illustration of each variables.

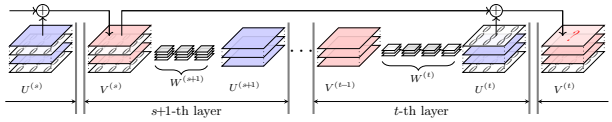
Note that this definition induces constraints on the channel activation variables. In detail, the j-th output channel activation in l-th layer is set if and only if at least one shape column activation in the j-th output channel is set. Concretely, the new formulation should include the constraints $r_j^{(l)} \leq \sum_{a,b} q_{j,a,b}^{(l)}$ and $q_{j,a,b}^{(l)} \leq r_j^{(l)}\ \forall a, b$. Our optimization problem with these shape column activation variables are introduced in Supplementary material A.

We can also formulate this optimization problem as a discrete nonconvex QCQP. The details on the standard QCQP form are provided in Supplementary material B. Furthermore, we show that the constraints in our optimization problems provably eliminate any unpruned inactive weights and accurately model the resource usage as well as the objective of the pruned network. The proof of this statement is given in Supplementary material C.

3.4 Handling nonsequential connections



(a) GBN [(You et al., 2019)], $v^{(s)} = [1, 0, 1, 0]$, $u^{(t)} = [1, 0, 1, 0]$, and $v^{(t)} = [1, 0, 1, 0]$.



(b) Ours, $v^{(s)} = [1, 0, 1, 0]$, $u^{(t)} = [0, 1, 1, 0]$, and $v^{(t)} = [X, 1, 1, 0]$, where X can be both 0 or 1.

Figure 5: Comparison of the two channel pruning constraints on a network with nonsequential connection (skip addition). Note that our method does not require $v^{(s)} = u^{(t)}$, and also has more flexibility with regard to $v^{(t)}$. Concretely, in this example, our method allows both $v^{(t)} = [0, 1, 1, 0]$ and $v^{(t)} = [1, 1, 1, 0]$.

Nonsequential connections, such as skip additions (He et al., 2016; Sandler et al., 2018; Tan and Le, 2019) or skip concatenations (Huang et al., 2017), are an essential part of modern neural networks. However, previous pruning works focus on dealing with only sequential connections and often resort to simple heuristics (Liu et al., 2017; He et al., 2018a, 2019; Molchanov et al., 2017,

[3] Concretely, we set a_l and b_l of Equation (3) to β_{l+1} and δ_l / K_l^2. K_l^2 in the denominator comes from $A^{(l)} = K_l^2 \|r^{(l-1)}\|_1 \|r^{(l)}\|_1$.

[4] We evaluate the mean percent error by $\frac{1}{n} \sum_{i=1}^{n} \frac{|E_i - A_i|}{A_i}$ where E_i and A_i are the estimated and the actual value, respectively.

2019). In this subsection, we show that our method can be naturally generalized to handle nonsequential connections, providing a much larger optimization search space compared to the previous works. As an example, we compare our method with the heuristic adopted by GBN (You et al., 2019) in Appendix A.

Before going into the details, we first define some new notations in Table 2, as an output feature map does not directly correspond to the subsequent layer's input feature map when nonsequential connections exist. First, we denote the input feature map and output feature map of the t-th convolution layer as $V^{(t-1)} \in \mathbb{R}^{C_{t-1} \times H_{t-1} \times W_{t-1}}$ and $U^{(t)} \in \mathbb{R}^{C_t \times H_t \times W_t}$, respectively. Then, we define input channel activation $v^{(t-1)} \in \{0,1\}^{C_{t-1}}$ and output channel activation $u^{(t)} \in \{0,1\}^{C_t}$ to indicate the remaining output channels of the $V^{(t-1)}$ and $U^{(t)}$ after pruning.

Table 2: Notation change of feature map and channel activation from sequential CNNs to nonsequential CNNs.

CNN type	Sequential	Nonsequential	
		input	output
Feature map	$X^{(l)}$	$V^{(l)}$	$U^{(l)}$
Channel activation	$r^{(l)}$	$v^{(l)}$	$u^{(l)}$

Assume layer s and t are nonsequentially connected via skip addition, as illustrated in Appendix A. Since $v^{(s)}$ and $u^{(t)}$ both affect $v^{(t)}$, we need a protocol to resolve $v^{(t)}$ when $v^{(s)}$ and $u^{(t)}$ are set differently. To handle this problem, GBN simply imposes the constraint $v^{(s)} = u^{(t)} = v^{(t)}$ to avoid any conflicts between the output channel activations. On the other hand, our method allows $v^{(s)}$ and $v^{(t)}$ to differ, and resolve the conflicts algorithmically according to the status of each activation masks. As a result, our method is able to adopt a more flexible constraint: $u^{(t)} \preceq v^{(t)} \preceq u^{(t)} + v^{(s)}$. More details of this procedure and the resulting optimization form are provided in Supplementary material A. When $v^{(s)}$, $u^{(t)}$, and $v^{(t)}$ are of dimension n, the number of possible $(v^{(s)}, u^{(t)}, v^{(t)})$ combinations without any constraints is 8^n. The constraint of GBN reduces this number to 2^n, while our constraint reduces the number to only 5^n. This shows our method provides a much larger optimization search space compared to the previous method.

We generalize Equation (3) to other types of nonsequential connections such as skip concatenations or skip additions with different dimensions in Supplementary material A. Note that the constraints in the generalized Equation (3) also eliminate any unpruned inactive weights in nonsequential networks with skip additions. The proof of this statement is also given in Supplementary material C.

3.5 Objective optimization

Equation (3) and generalized optimization problems of Equation (3) fall into the category of binary Mixed Integer Quadratic Constraint Quadratic Programming (MIQCQP). We solve these discrete QCQP problems with the CPLEX library (INC, 1993), which provides MIQCQP solvers based on the branch and cut technique. However, the branch and cut algorithm can lead to exponential search time (Mitchell, 2002) on large problems. Therefore, we provide a practical alternative utilizing a block coordinate descent style optimization, described in Supplementary material E.

4 Related works

Importance of channels Most of the channel pruning methods prune away the least important channels with a simple greedy approach, and the evaluation method for the importance of channels has been the main research problem (Molchanov et al., 2017, 2019; Liu et al., 2019). Channel pruning is divided into two major branches according to the method of evaluating the importance of channels: the *trainable-importance* method, which constantly evaluates the importance of channels while training the whole network from scratch, and the *fixed-importance* method, which directly evaluates the importance of channels on the pretrained network. Trainable-importance channel pruning methods include regularizer-based methods with group sparsity regularizers (Wen et al., 2016; Alvarez and Salzmann, 2016; Yang et al., 2019; Liu et al., 2017; Louizos et al., 2018; Gordon et al., 2018) and data-driven channel pruning methods (Kang and Han, 2020; You et al., 2019). Fixed-importance channel pruning methods first prune away most of the weights and then finetune the significantly smaller pruned network (Molchanov et al., 2017, 2019; Hu et al., 2016; He et al., 2018a; Li et al., 2017; He et al., 2019; Luo et al., 2017; Peng et al., 2019). As a result, fixed-importance methods are much more efficient than the trainable-importance methods in terms of computational cost and memory usage as trainable-importance methods require training the whole unpruned network. Our framework is on the line of fixed-importance channel pruning works. For example, when pruning a ResNet for CIFAR-10 dataset, Liu et al. (2017) trains the entire network with a sparsity regularizer for 160 epochs, prunes the trained network, and then finetunes it for another 160 epochs. Meanwhile, our method only requires pruning the pretrained network and finetuning the the pruned network for 200 epochs.

Quadratic coupling CCP (Peng et al., 2019) formulates a QP (quadratic formulation) to consider the quadratic coupling between channels in the same layer under predefined layer-wise constraints on the maximum number of channels. On the other hand, our formulation considers the quadratic coupling between channels in the neighboring layers under the target resource constraints.

Spatial pruning Spatial pruning methods aim to prune convolution filters along the channel dimension for inference efficiency. Spatial pruning methods manually define the spatial patterns of filters (Lebedev and Lempitsky, 2016; Anwar et al., 2017) or optimize spatial patterns of filters with group sparse regularizers (Wen et al., 2016; Lebedev and Lempitsky, 2016). Among these works, Lebedev and Lempitsky (2016) empirically demonstrates that enforcing sparse spatial patterns in 2-D filters along the input channel leads to great speed-up during inference time using group sparse convolution operations (Chellapilla et al., 2006). Our proposed method enforces the spatial patterns in 2-D filters as in Lebedev and Lempitsky (2016) for speed-up in inference.

Inference time constraint Even though inference time is a metric that many machine learning practitioners are interested in, previous pruning methods resort to a well-known but inaccurate proxy, FLOPs, due to the difficulty of measuring and modeling the inference time. NetAdapt and AMC (Yang et al., 2018; He et al., 2018b) directly measure the inference time of proposed networks by deploying each of them on the edge device. However, this approach incurs a high cost since we have to place, run, and remove every candidate network. On the other hand, we propose a quadratic model to estimate the inference time of the whole pruned network without direct deployment.

5 Experiments

We compare the classification accuracy of the pruned network against several pruning baselines on CIFAR-10 and ImageNet datasets using DenseNet-40 (Huang et al., 2017), VGG-16 (Simonyan and Zisserman, 2015), EfficientNet (Tan and Le, 2019), and various versions of ResNet (He et al., 2016). Note that most pruning baselines apply an iterative pruning procedure, which repeatedly alternates between network pruning and finetuning until the target resource constraints are satisfied (Han et al., 2015; He et al., 2018a; Liu et al., 2017; Yang et al., 2018). In contrast, our methods explicitly combine the target resource constraint to the optimization framework and only need one round of pruning and finetuning.

5.1 Experimental details

We follow the 'smaller-norm-less-important' criterion (Ye et al., 2018; Liu et al., 2017), which evaluates the importance of weights with the absolute value of the weight (Han et al., 2015; Guo et al., 2016). We assume FLOPs reduction are linearly proportional to the sparsity in shape column activations, as empirically shown in Lebedev and Lempitsky (2016). In the experiment tables, FLOPs of the pruned network are computed according to the resource specifications in Equation (3). The 'IC' column indicates whether each method is a trainable-importance method (T) or a fixed-importance method (F). 'ours-c' and 'ours-cs' each refers to our method with only channel pruning and with both the channel and spatial pruning, respectively. The experiment results on network size constraints are provided in Supplementary material F.

Table 3: Pruned accuracy and accuracy drop from the baseline network at given FLOPs on various network architectures (ResNet-20,32,56 and DenseNet-40) at CIFAR-10. Asterisk (*) indicates that the method does not use finetuning.

Method	IC	Baseline acc	Pruned acc↑	Acc drop↓	FLOPs(%)↓
Network: ResNet-20					
SFP (He et al., 2018a)	F	92.20 (0.18)	90.83 (0.31)	1.37	**57.8**
FPGM (He et al., 2019)	F	92.21 (0.18)	91.72 (0.20)	0.49	**57.8**
ours-c	F	92.21 (0.18)	91.74 (0.20)	0.47	58.3
ours-cs	F	92.21 (0.18)	**92.26** (0.10)	**-0.05**	**57.8**
Network: ResNet-32					
SFP (He et al., 2018a)	F	92.63 (0.70)	92.08 (0.08)	0.55	58.5
FPGM (He et al., 2019)	F	92.88 (0.86)	92.51 (0.90)	0.37	58.5
ours-c	F	92.88 (0.86)	92.52 (0.46)	0.36	**57.2**
ours-cs	F	92.88 (0.86)	**92.80** (0.61)	**0.08**	57.9
Network: ResNet-56					
SFP (He et al., 2018a)	F	93.59 (0.58)	92.26 (0.31)	1.33	47.5
FPGM (He et al., 2019)	F	93.59 (0.58)	93.49 (0.13)	0.10	47.5
CCP (Peng et al., 2019)	F	93.50	93.42	0.08	**47.4**
SCP (Kang and Han, 2020)	T*	93.69	93.23	0.46	48.5
ours-c	F	93.59 (0.58)	93.36 (0.68)	0.23	**47.4**
ours-cs	F	93.59 (0.58)	**93.59** (0.36)	**0.00**	**47.4**
Network: DenseNet-40					
SCP (Kang and Han, 2020)	T*	94.39	93.77	**0.62**	**29.2**
ours-c	F	95.01	93.80	1.21	**29.2**
ours-cs	F	95.01	**94.25**	0.76	**29.2**
slimming (Liu et al., 2017)	T	93.89	94.35	**-0.46**	**45.0**
ours-c	F	95.01	94.38	0.63	**45.0**
ours-cs	F	95.01	**94.85**	0.16	**45.0**
slimming (Liu et al., 2017)	T	93.89	94.81	**-0.92**	71.6
ours-c	F	95.01	94.82	0.19	**71.0**
ours-cs	F	95.01	**95.02**	-0.01	**71.0**

5.2 CIFAR-10

CIFAR-10 dataset has 10 different classes with $5k$ training images and $1k$ test images per each class Krizhevsky et al. (2009). In CIFAR-10 experiments, we evaluate our methods on four network architectures: ResNet-20, 32, 56, and DenseNet-40. Implementation details of our experiments are listed in Supplementary material F. We show the experiment results of pruning under FLOPs constraints in Table 3.

We find that 'ours-c' shows comparable results against FPGM, which is the previous state of the art method, on ResNet-20, 32, and 56. Moreover, 'ours-cs' significantly outperforms both 'ours-c' and FPGM on the same architectures, showing a state of the art performance. Also, 'ours-c' shows comparable results against slimming (Liu et al., 2017) and SCP (Kang and Han, 2020), which are trainable-importance methods, while 'ours-cs' outperforms the baselines by a large margin on DenseNet-40. These results show simultaneous channel and spatial pruning produces more computationally efficient networks with better performance compared to other channel pruning methods on CIFAR-10.

Table 4: Top1,5 pruned accuracy and accuracy drop from the baseline network at given FLOPs on various network architectures (ResNet-18, 50, VGG-16, and EfficientNet-B0) at ImageNet. † denotes the methods which report their results to the first decimal place.

Method	IC	Top1 Pruned Acc↑	Top1 Acc drop↓	FLOPs(%)↓
Network: ResNet-18				
SFP (He et al., 2018a)	F	67.10	3.18	**58.2**
FPGM (He et al., 2019)	F	68.41	1.87	**58.2**
ours-c	F	67.48	2.28	60.9
ours-cs	F	**69.59**	**0.17**	**58.2**
Network: ResNet-50				
SFP (He et al., 2018a)	F	74.61	1.54	58.3
FPGM (He et al., 2019)	F	75.50	0.65	**57.8**
ours-c	F	75.78	0.37	**57.8**
ours-cs	F	**75.93**	**0.22**	**57.8**
GBN (You et al., 2019)	T	**76.19**	**-0.31**	59.5
ours-c	F	75.89	0.26	61.5
ours-cs	F	76.00	0.15	**59.0**
Network: EfficientNet-B0				
uniform MP	F	75.06	2.57	**76.4**
ours-c	F	**75.71**	**1.92**	**76.4**
uniform MP	F	69.08	8.55	53.8
ours-c	F	**73.27**	**4.36**	**53.7**
Method	IC	Top5 Pruned Acc↑	Top5 Acc drop↓	FLOPs(%)↓
Network: VGG-16				
Molchanov et al. (2017)†	F	84.5	5.9	**51.7**
ours-c	F	87.20	3.18	**51.7**
ours-cs	F	**87.36**	**3.02**	**51.7**

5.3 ImageNet

ILSVRC-2012 (Russakovsky et al., 2015) is a large-scale dataset with 1000 classes that comes with $1.28M$ training images and $50k$ validation images. We conduct our methods under the fixed FLOPs constraint on ResNet-18, 50, EfficientNet-B0, and VGG-16. For more implementation details of the ImageNet experiments, refer to Supplementary material F. We show ImageNet experiment results in Table 4. In ResNet-50, 'ours-c' and 'ours-cs' achieve results comparable to GBN, a trainable-importance channel pruning method which is the previous state of the art, even though our method is a fixed-importance channel pruning method. In particular, top1 pruned accuracy in 'ours-cs' exceeds SFP by 1.32% using a similar number of FLOPs. Both 'ours-cs' and 'ours-c' clearly outperform FPGM in ResNet-50. In EfficientNet-B0, we compare 'ours-c' with 'uniform MP', where 'uniform MP' denotes a magnitude-based channel pruning method which greedily prunes filters with small weight norms uniformly among layers. 'ours-c' again outperforms 'uniform MP' in various FLOPs constraints. Also, 'ours-c' and 'ours-cs' show significantly better performance compared to Molchanov et al. (2017) on VGG-16. More experiments on FLOPs constraints with MobileNetV2 and image segmentation tasks are provided in Supplementary F.

Table 5: Top1 pruned accuracy and accuracy drop from the baseline network at given inference time on ResNet-50 architecture at ImageNet.

Method	IC	Top1 Pruned Acc↑	Top1 Acc drop↓	Inference time (ms)↓
FPGM	F	75.50	0.65	1.68 (1.51×)
ours-c	F	**75.83**	**0.32**	**1.66** (**1.52**×)

Table 5 shows the pruning result under inference time constraints. We observe that 'ours-c' outperforms FPGM on ResNet-50 in both top1 pruned accuracy and the inference time, reducing the accuracy drop to half compared to FPGM. Estimating the inference time of the pruned network using the quadratic model, our method successfully finds a network with faster inference than the baseline without direct deployment.

6 Conclusion

We propose an optimal channel selection method with a discrete QCQP based optimization framework. Greedy channel selection methods ignore the inherent quadratic coupling between channels in the neighboring layers and fail to eliminate inactive weights during pruning. To this end, our selection method models the quadratic coupling explicitly and prevents any inactive weights during the pruning procedure. Our selection method allows exact modeling of the user-specified resource constraints in terms of FLOPS, memory usage, and network size, which enables the direct optimization of the true objective on the pruned network. In addition, we propose a new quadratic model that accurately estimates the inference time of a pruned network without direct deployment, which allows us to adopt inference time as a resource constraint option. We also extend our method to also select individual 2D convolution filters simultaneously and handle nonsequential operations in modern neural networks more flexibly. Extensive experiments show our proposed method significantly outperforms other fixed-importance channel pruning methods, finding smaller and faster networks with the least drop in accuracy.

Acknowledgement

This research was supported in part by Samsung Advanced Institute of Technology, Samsung Electronics Co., Ltd, Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2020-0-00882, (SW STAR LAB) Development of deployable learning intelligence via self-sustainable and trustworthy machine learning), and Basic Science Research Program through the National Research Foundation of Korea (NRF) (2020R1A2B5B03095585). Yeonwoo Jeong was supported by NRF(National Research Foundation of Korea) Grant funded by the Korean Government(NRF-2019-Global Ph.D. Fellowship Program). Hyun Oh Song is the corresponding author.

References

Alvarez, J. M. and Salzmann, M. (2016). Learning the number of neurons in deep networks. In *NeurIPS*.

Anwar, S., Hwang, K., and Sung, W. (2017). Structured pruning of deep convolutional neural networks. In *JETC*.

Chellapilla, K., Puri, S., and Simard, P. (2006). High performance convolutional neural networks for document processing. In *IWFHR*.

Chorowski, J. K., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. (2015). Attention-based models for speech recognition. In *NeurIPS*.

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

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*.

Gordon, A., Eban, E., Nachum, O., Chen, B., Wu, H., Yang, T.-J., and Choi, E. (2018). Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *CVPR*.

Guo, Y., Yao, A., and Chen, Y. (2016). Dynamic network surgery for efficient dnns. In *NeurIPS*.

Han, S., Pool, J., Tran, J., and Dally, W. (2015). Learning both weights and connections for efficient neural network. In *NeurIPS*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *CVPR*.

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

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

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

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

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *CVPR*.

Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. (2016). Deep networks with stochastic depth. In *ECCV*.

INC, C. O. (1993). Using the cplex callable library and cplex mixed integer library. In *Incline Village, NV*.

Kang, M. and Han, B. (2020). Operation-aware soft channel pruning using differentiable masks. In *ICML*.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images. In *Tech Report*.

Kwon, W., Yu, G.-I., Jeong, E., and Chun, B.-G. (2020). Nimble: Lightweight and parallel gpu task scheduling for deep learning. In *NeurIPS*.

Lebedev, V. and Lempitsky, V. (2016). Fast convnets using group-wise brain damage. In *CVPR*.

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

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *ICLR*.

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. (2017). Learning efficient convolutional networks through network slimming. In *ICCV*.

Liu, Z., Mu, H., Zhang, X., Guo, Z., Yang, X., Cheng, K., and Sun, J. (2019). Metapruning: Meta learning for automatic neural network channel pruning. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. (2019). Rethinking the value of network pruning. In *ICLR*.

Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Louizos, C., Welling, M., and Kingma, D. P. (2018). Learning sparse neural networks through l_0 regularization. In *ICLR*.

Luo, J.-H., Wu, J., and Lin, W. (2017). Thinet: A filter level pruning method for deep neural network compression. In *ICCV*.

Mitchell, J. E. (2002). Branch-and-cut algorithms for combinatorial optimization problems. In *Handbook of applied optimization*.

Molchanov, P., Mallya, A., Tyree, S., Frosio, I., and Kautz, J. (2019). Importance estimation for neural network pruning. In *CVPR*.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. (2017). Pruning convolutional neural networks for resource efficient inference. In *ICLR*.

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

Peng, H., Wu, J., Chen, S., and Huang, J. (2019). Collaborative channel pruning for deep networks. In *ICML*.

Redmon, J. and Farhadi, A. (2018). Yolov3: An incremental improvement. In *arXiv:1804.02767*.

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

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

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *ICLR*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *JMLR*.

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

Van Der Walt, S., Colbert, S. C., and Varoquaux, G. (2011). The numpy array: a structure for efficient numerical computation. In *CISE*.

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

Yang, H., Wen, W., and Li, H. (2019). Deephoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures. In *ICLR*.

Yang, T.-J., Howard, A., Chen, B., Zhang, X., Go, A., Sandler, M., Sze, V., and Adam, H. (2018). Netadapt: Platform-aware neural network adaptation for mobile applications. In *ECCV*.

Ye, J., Lu, X., Lin, Z., and Wang, J. Z. (2018). Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *ICLR*.

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

Supplementary Material

A QCQP formulation on Nonsequential connections

We first formulate the optimization problem for channel and spatial pruning in Section 3.3. Then, we generalize the optimization problem to cover nonsequential connections.

To recap, the original formulation of our optimization problem is as follows:

$$\begin{aligned} \underset{r^{(0:L)}}{\text{maximize}} \quad & \sum_{l=1}^{L} \left\langle I^{(l)}, A^{(l)} \right\rangle \\ \text{subject to} \quad & \sum_{l=0}^{L} a_l \left\| r^{(l)} \right\|_1 + \sum_{l=1}^{L} b_l \left\| A^{(l)} \right\|_1 \leq M \\ & A^{(l)} = r^{(l-1)} r^{(l)\intercal} \otimes J_{K_l} \quad \forall l \in [L]. \end{aligned} \tag{3}$$

Concretely, we handle two prevalent types of nonsequential connections: skip addition (He et al., 2016; Sandler et al., 2018; Tan and Le, 2019) and skip concatenation (Huang et al., 2017).

A.1 Joint channel and spatial pruning

We first recap the definition of shape column activation. The shape column activations $q^{(l)} \in \{0,1\}^{C_l \times K_l \times K_l}$ indicate which shape columns in the l-th convolution layer remain in the pruned network. Then, the new formulation include the constraints $r_j^{(l)} \leq \sum_{a,b} q_{j,a,b}^{(l)}$ and $q_{j,a,b}^{(l)} \leq r_j^{(l)}$ $\forall a, b$.

We aim to maximize the sum of the importance of active weights after pruning under the given resource constraints. Then, our optimization problem for joint channel and spatial pruning becomes

$$\begin{aligned} \underset{r^{(0:L)}, q^{(1:L)}}{\text{maximize}} \quad & \sum_{l=1}^{L} \left\langle I^{(l)}, A^{(l)} \right\rangle \\ \text{subject to} \quad & \sum_{l=0}^{L} a_l \left\| r^{(l)} \right\|_1 + \sum_{l=1}^{L} b_l \left\| A^{(l)} \right\|_1 \leq M \\ & r_j^{(l)} \leq \sum_{a,b} q_{j,a,b}^{(l)} \quad q_{j,a,b}^{(l)} \leq r_j^{(l)} \quad \forall l, j, a, b \\ & A^{(l)} = r^{(l-1)} \otimes q^{(l)} \quad \forall l \\ & r^{(l)} \in \{0,1\}^{C_l} \quad q^{(l)} \in \{0,1\}^{C_l \times K_l \times K_l} \quad \forall l \in [L]. \end{aligned} \tag{4}$$

A.2 Skip addition

We first introduce two notations, *output channel activation* $u^{(t)} \in \{0,1\}^{C_t}$ and *input channel activation* $v^{(t)} \in \{0,1\}^{C_t}$, along with the corresponding new constraints, in Table 6. As described in Section 3.4, the definition of $u^{(t)}$ induces new constraints between the shape column activations $\left(= q^{(t)}\right)$ and output channel activations $\left(= u^{(t)}\right)$. Concretely, the constraints are given as $u_j^{(t)} \leq \sum_{a,b} q_{j,a,b}^{(t)}$ and $q_{j,a,b}^{(t)} \leq u_j^{(t)}$ $\forall a, b$.

We now discuss the constraints between the input and output channel activation variables under four possible scenarios depending on the architectural implementations of skip additions. Here, we denote the set of layer index pairs $\{(s,t)\}$ which have skip additions as $\mathcal{P}$. Concretely, $(s,t) \in \mathcal{P}$ if and only if the input feature map of the $s{+}1$-th convolution layer is added to the output feature map of the t-th layer, forming the input feature map of the $t{+}1$-th layer. Also, let $T = \{t \mid (s,t) \in \mathcal{P}\}$. For a layer t, we formulate the channel activation constraints for each possible connection scenarios separately:

(i) If there is no skip addition incoming to the t-th layer ($t \notin T$), then we force $u^{(t)} = v^{(t)}$.

(ii) For a skip addition pair $(s,t) \in \mathcal{P}$ with matching channel dimensions ($C_s = C_t$), the input feature map of the $s{+}1$-th convolution layer is directly added to the output feature map from the t-th layer as illustrated in Figure 6a. In this case, we can formulate the constraints as $u^{(t)} \preceq v^{(t)} \preceq u^{(t)} + v^{(s)}$.

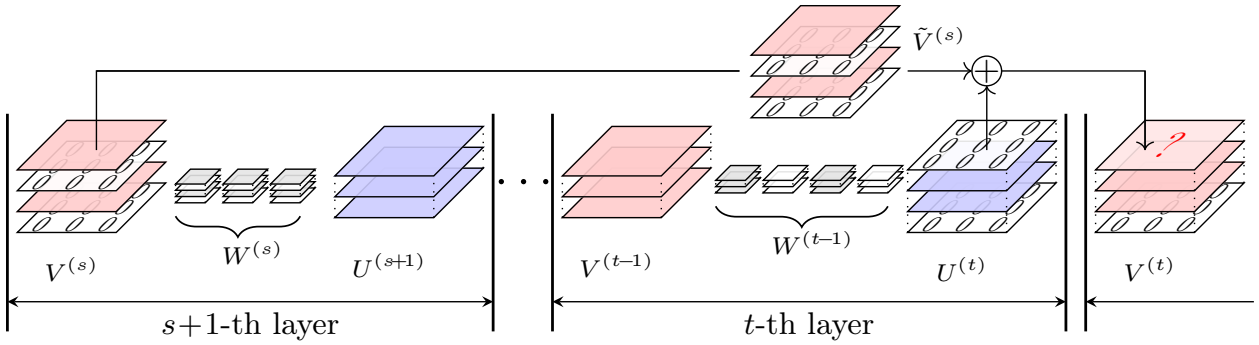
(iii) For a skip addition pair $(s,t) \in \mathcal{P}$ with mismatching channel dimensions ($C_s < C_t$), the skip addition can utilize zero padding $(iii-a)$ or 1×1 convolutions $(iii-b)$ to resolve the mismatch (He et al., 2016). We define the *augmented feature map* $\tilde{V}^{(s)}$ after the zero padding or $1\times$ convolution and corresponding *augmented channel activation* $\tilde{v}^{(s)} \in \{0,1\}^{C_t}$. Then, we formulate the constraints for both cases as below. Not that similar with the constraint in (ii), the constraints for both cases are formulated as $u^{(t)} \preceq v^{(t)} \preceq u^{(t)} + \tilde{v}^{(s)}$.

$(iii-a)$ A $(C_t - C_s)$-dimensional zero-valued feature map is padded to the end of the $s{+}1$-th convolution layer's input feature map. We define $\tilde{v}^{(s)} = \left[v^{(s)}, 0_{C_t - C_s}\right]$, as illustrated in Figure 6b. Therefore, for all $j \leq C_s$, $u_j^{(t)} \leq v_j^{(t)} \leq u_j^{(t)} + v_j^{(s)}$ and for all $j > C_s$, $u_j^{(t)} \leq v_j^{(t)} \leq u_j^{(t)} + 0$.

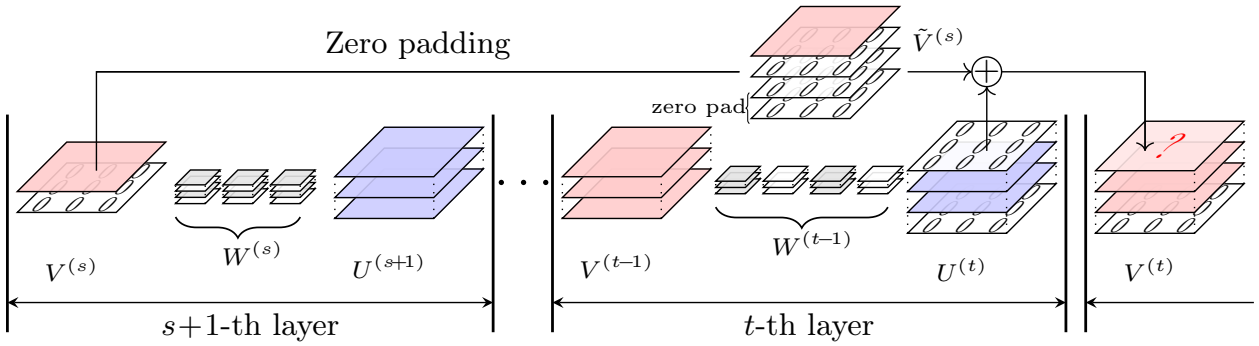
$(iii-b)$ 1×1 convolution is applied to the $s{+}1$-th layer's input feature map to match the larger channel dimension ($= C_t$). Since the number of FLOPs and weights in a 1×1 convolution is negligible compared to the total number of FLOPs and weights, we assume all of the output channels of a 1×1 convolution are activated and define $\tilde{v}^{(s)} = 1_{C_t}$ as illustrated in Figure 6c. Therefore, $u^{(t)} \preceq v^{(t)} \preceq u^{(t)} + 1_{C_t}$.

Table 6: Notation and constraints change of feature map and channel activation from sequential CNNs to nonsequential CNNs with skip addition.

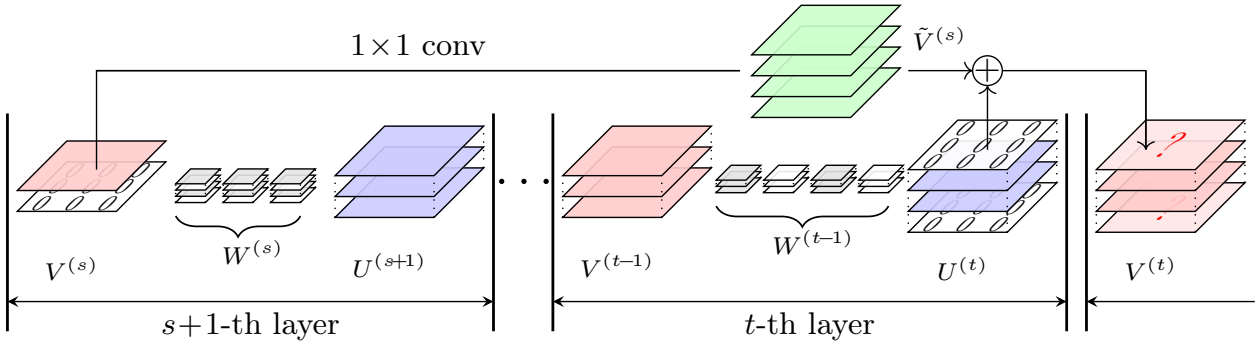
CNN type	Sequential	Nonsequential	
		input	output
Notation			
Feature map	$X^{(l)}$	$V^{(l)}$	$U^{(l)}$
Channel activation	$r^{(l)}$	$v^{(l)}$	$u^{(l)}$
Constraint			
Binary pruning mask	$A^{(l)} = r^{(l-1)} \otimes q^{(l)}$	$A^{(l)} = v^{(l-1)} \otimes q^{(l)}$	
Shape column activation	$r_j^{(l)} \leq \sum_{a,b} q_{j,a,b}^{(l)}$ $q_{j,a,b}^{(l)} \leq r_j^{(l)}$	$u_j^{(l)} \leq \sum_{a,b} q_{j,a,b}^{(l)}$ $q_{j,a,b}^{(l)} \leq u_j^{(l)}$	



(a) $v^{(s)} = [1, 0, 1, 0]$, $u^{(t)} = [0, 1, 1, 0]$, $v^{(t)} = [X, 1, 1, 0]$.



(b) $v^{(s)} = [1, 0]$, $\tilde{v}^{(s)} = [1, 0, 0, 0]$, $u^{(t)} = [0, 1, 1, 0]$, $v^{(t)} = [X, 1, 1, 0]$.



(c) $v^{(s)} = [1, 0]$, $\tilde{v}^{(s)} = [1, 1, 1, 1]$, $u^{(t)} = [0, 1, 1, 0]$, $v^{(t)} = [X, 1, 1, X]$.

Figure 6: Illustration of the skip addition procedure for each skip addition scenarios. X can be both 0 or 1.

We now summarize the constraints for the four cases discussed above to Equation (5).

$$
\begin{aligned}
&(i)\quad v^{(t)} = u^{(t)} \quad \forall t \notin T \\
&(ii)\quad u^{(t)} \preceq v^{(t)} \preceq u^{(t)} + v^{(s)} \quad \forall (s,t) \in \mathcal{P} \text{ and } C_t = C_s \\
&(iii)\quad \forall (s,t) \in \mathcal{P} \text{ and } C_t > C_s, \\
&\qquad (iii-a)\quad u_j^{(t)} \le v_j^{(t)} \le u_j^{(t)} + v_j^{(s)} \quad \forall j \le C_s \\
&\qquad\qquad \text{and } v_j^{(t)} = u_j^{(t)} \quad \forall j > C_s \\
&\qquad (iii-b)\quad u^{(t)} \preceq v^{(t)}.
\end{aligned} \tag{5}
$$

In Supplementary material C, we prove that the constraints of Equation (5) prevent inactive weights from remaining in the pruned network with skip additions.

We now formulate the network channel and spatial pruning optimization problem that handles nonsequential connections using the input channel, output channel, and shape column activation variables:

$$\underset{u^{(0:L)},v^{(0:L)},q^{(1:L)}}{\text{maximize}} \quad \sum_{t=1}^{L}\left\langle I^{(t)}, A^{(t)}\right\rangle \tag{6}$$

subject to

$$\sum_{t=0}^{L} a_t \left\|u^{(t)}\right\|_1 + \sum_{t\in T} a_t \left\|v^{(t)}\right\|_1 + \sum_{t=1}^{L} b_t \left\|A^{(t)}\right\|_1 \le M \qquad (i)\ \ v^{(t)} = u^{(t)} \quad \forall t \notin T$$

$$u_j^{(t)} \le \sum_{a,b} q_{j,a,b}^{(t)} \quad \text{and} \quad q_{j,a,b}^{(t)} \le u_j^{(t)} \quad \forall t,j,a,b \qquad (ii)\ \ u^{(t)} \preceq v^{(t)} \preceq u^{(t)} + v^{(s)} \quad \forall (s,t)\in\mathcal{P} \text{ and } C_t = C_s$$

$$A^{(t)} = v^{(t-1)} \otimes q^{(t)} \quad \forall t \qquad (iii)\ \ \forall (s,t)\in\mathcal{P} \text{ and } C_t > C_s,$$

$$u^{(t)}, v^{(t)} \in \{0,1\}^{C_t} \text{ and } q^{(t)} \in \{0,1\}^{C_t\times K_t\times K_t} \quad \forall t\in[L] \qquad (iii-a)\ \ u_j^{(t)} \le v_j^{(t)} \le u_j^{(t)} + v_j^{(s)} \quad \forall j \le C_s$$

$$v_j^{(t)} = u_j^{(t)} \quad \forall j > C_s$$

$$(iii-b)\ \ u^{(t)} \preceq v^{(t)}.$$

Concretely, Equation (6) reduces to the optimization problem for sequential convolution networks when $u^{(t)}=v^{(t)}$, $q_{j,a,b}^{(t)}=u_j^{(t)}$, and $\mathcal{P}=\emptyset\ \forall t,j,a,b$.

A.3 Skip concatenation

Skip concatenation, which is a crucial feature of the well-known DenseNet (Huang et al., 2017), requires different techniques from skip addition. Concretely, the skip concatenation of a layer pair (p,q) means the p-th layer's feature map is concatenated with the q-th layer's feature map before the $q+1$-th convolution. To handle the possible skip concatenations, we utilize the fact that when (p,q) is the skip concatenation pair, $q+1$-th convolution operation on the q-th layer can be thought as separate convolution operations on the p-th layer and the q-th layer, respectively. In this regard, we first assume there are convolution operations between every pair of layers. Then, we define $W^{(p,q)}\in\mathbb{R}^{C_p\times C_q\times K_{p,q}\times K_{p,q}}$ as the convolution weights between p-th layer and q-th layer where $p<q$. If there is no skip concatenation from p-th layer to $q-1$-th layer, we regard there is no convolution operation between p-th layer and q-th layer and set $W^{(p,q)}=0_{K_{p,q},K_{p,q}}$. Also, we introduce the corresponding shape column activation variables, $q^{(p,q)}\in\{0,1\}^{C_q\times K_{p,q}\times K_{p,q}}$, for the convolution operation from p-th layer to q-th layer. Then, we extend the optimization problem for skip concatenation as

$$\underset{r^{(0:L)},q^{(1:L,1:L)}}{\text{maximize}} \quad \sum_{p=1}^{L}\sum_{q=1}^{L}\left\langle I^{(p,q)}, A^{(p,q)}\right\rangle \tag{7}$$

subject to

$$\sum_{p=0}^{L} a_p \left\|r^{(p)}\right\|_1 + \sum_{p=1}^{L}\sum_{q=1}^{L} b_{p,q}\left\|A^{(p,q)}\right\|_1 \le M$$

$$r_j^{(q)} \le \sum_{a,b} q_{j,a,b}^{(p,q)} \quad \text{and} \quad q_{j,a,b}^{(p,q)} \le r_j^{(q)} \quad \forall p,q,j,a,b$$

$$A^{(p,q)} = r^{(p)} \otimes q^{(p,q)} \quad \forall p,q$$

$$r^{(p)}\in\{0,1\}^{C_p}, \quad q^{(p,q)}\in\{0,1\}^{C_q\times K_{p,q}\times K_{p,q}} \quad \forall p,q\in[L].$$

B Standard QCQP form

Proposition 3. *Equation* (3) *is a QCQP problem.*

Proof. We define the *importance of 2-D filter*, which is the sum of the importance of weights in the filter as $F^{(l)}\in\mathbb{R}_+^{C_{l-1}\times C_l}\quad\forall l$. Concretely, $F_{i,j}^{(l)}=\sum_{a,b} I_{i,j,a,b}^{(l)}\quad\forall i,j$. We wish to express the objective function and constraints in

Equation (3) with respect to $r^{(0:L)}$. Note that $\left\|A^{(l)}\right\|_1 = K_l^2 \left\|r^{(l-1)}\right\|_1 \left\|r^{(l)}\right\|_1$ and $\left\langle I^{(l)}, A^{(l)} \right\rangle = r^{(l-1)\intercal} F^{(l)} r^{(l)}$. To express Equation (3) in a standard QCQP form, we denote $\left[r^{(0)}, r^{(1)}, \ldots, r^{(L)}\right]$ as $\mathbf{r} \in \{0,1\}^N$ where $N = \sum_{l=0}^{L} C_l$. Standard QCQP form of Equation (3) is

$$\begin{aligned} &\underset{\mathbf{r}\in\{0,1\}^N}{\text{maximize}} \ \frac{1}{2}\mathbf{r}^\intercal P_0 \mathbf{r} \\ &\text{subject to} \\ &\qquad \frac{1}{2}\mathbf{r}^\intercal P_1 \mathbf{r} + q_1^\intercal \mathbf{r} \le M, \end{aligned}$$

where

$$P_0 = \begin{pmatrix} 0 & F^{(1)} & 0 & \cdots & 0 & 0 \\ F^{(1)\intercal} & 0 & F^{(2)} & \cdots & 0 & 0 \\ 0 & F^{(2)\intercal} & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & F^{(L)} \\ 0 & 0 & 0 & \cdots & F^{(L)\intercal} & 0 \end{pmatrix},$$

$$P_1 = \begin{pmatrix} 0 & Q_1 & 0 & \cdots & 0 & 0 \\ Q_1 & 0 & Q_2 & \cdots & 0 & 0 \\ 0 & Q_2 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & Q_L \\ 0 & 0 & 0 & \cdots & Q_L & 0 \end{pmatrix}$$

[5] , and $q_1 = [\underbrace{a_0, \ldots, a_0}_{C_0}, \underbrace{a_1, \ldots, a_1}_{C_1}, \cdots, \underbrace{a_L, \ldots, a_L}_{C_L}]$. □

Proposition 4. *Equation* (4) *is a QCQP problem.*

Proof. To prove Equation (4) is a QCQP problem, we show that objective function $\sum_{l=1}^{L} \left\langle I^{(l)}, A^{(l)} \right\rangle$, and the constraint $\sum_{l=0}^{L} a_l \left\|r^{(l)}\right\|_1 + \sum_{l=1}^{L} b_l \left\|A^{(l)}\right\|_1$, are sum of quadratic and linear terms of $r^{(0:L)}$ and $q^{(1:L)}$. Note that

$$\begin{aligned} \left\langle I^{(l)}, A^{(l)} \right\rangle &= \sum_{i=1}^{C_{l-1}} \sum_{j=1}^{C_l} \sum_{a=1}^{K_l} \sum_{b=1}^{K_l} I^{(l)}_{i,j,a,b} r_i^{(l-1)} q^{(l)}_{j,a,b} \\ \left\|r^{(l)}\right\|_1 &= \sum_{i=1}^{C_l} r_i^{(l)} \\ \left\|A^{(l)}\right\|_1 &= \sum_{i=1}^{C_{l-1}} \sum_{j=1}^{C_l} \sum_{a=1}^{K_l} \sum_{b=1}^{K_l} r_i^{(l-1)} q^{(l)}_{j,a,b}. \end{aligned}$$

Clearly, the objective function and all the constraints in Equation (4) can be expressed as the sum of quadratic and linear terms of $r^{(0:L)}$ and $q^{(1:L)}$. Therefore, Equation (4) is a QCQP problem with discrete variables, $r^{(0:L)}$ and $q^{(1:L)}$. □

Proposition 5. *Equation* (6) *is a QCQP problem.*

Proof. To prove Equation (6) is a QCQP problem, we show that $\sum_{t=1}^{L} \left\langle I^{(t)}, A^{(t)} \right\rangle$ and $\sum_{t=0}^{L} a_t \left\|u^{(t)}\right\|_1 +$

[5] $Q_l = b_l K_l^2 J_{C_l}, \quad \forall l \in [L]$

$\sum_{t\in T} a_t \left\|v^{(t)}\right\|_1 + \sum_{t=1}^{L} b_t \left\|A^{(t)}\right\|_1$ are sum of quadratic and linear terms of $u^{(0:L)}, v^{(0:L)}$ and $q^{(1:L)}$. Note that

$$\left\langle I^{(t)}, A^{(t)} \right\rangle = \sum_{i=1}^{C_{t-1}} \sum_{j=1}^{C_t} \sum_{a=1}^{K_t} \sum_{b=1}^{K_t} I^{(t)}_{i,j,a,b} v_i^{(t-1)} q^{(t)}_{j,a,b}$$

$$\left\|u^{(t)}\right\|_1 = \sum_{i=1}^{C_t} u_i^{(t)}$$

$$\left\|v^{(t)}\right\|_1 = \sum_{i=1}^{C_t} v_i^{(t)}$$

$$\left\|A^{(t)}\right\|_1 = \sum_{i=1}^{C_{t-1}} \sum_{j=1}^{C_t} \sum_{a=1}^{K_t} \sum_{b=1}^{K_t} v_i^{(t-1)} q^{(t)}_{j,a,b}.$$

Clearly, the objective function and all the constraints in Equation (6) can expressed by the sum of quadratic and linear terms of $u^{(0:L)}$, $v^{(0:L)}$ and $q^{(1:L)}$. Therefore, Equation (6) is a QCQP problem with discrete variables, $u^{(0:L)}$, $v^{(0:L)}$, and $q^{(1:L)}$. □

C Pruning consistency

Pruning operation that removes weights through output channel direction leads to *inactive* weights during the pruning procedure and prevent the exact modeling of the hard resource constraints (FLOPs and network size). In previous channel pruning methods based on the greedy approach, the pruned network requires post-pruning procedures to eliminate the remaining inactive weights. However, our formulation guarantees the exclusion of inactive weights from the pruned network.

C.1 Preliminary

We assume each pruning methods outputs a pruning mask $A^{(t)} \in \{0,1\}^{C_{t-1}\times C_t \times K_l \times K_l}$. Then, we denote the pruned weights as $\hat{W}^{(t)} = W^{(t)} \odot A^{(t)}$. In the pruned network with pruned weights $\hat{W}^{(1:L)}$, we denote the input feature map of the $t+1$-th convolution as $V^{(t)} \in \mathbb{R}^{C_t \times H_t \times W_t}$. Also, we denote the output feature map of the t-th convolution as $U^{(t)} \in \mathbb{R}^{C_t \times H_t \times W_t}$. To avoid notation clutter, we ignore batch normalization and nonlinear activation function in this section. Then, $U^{(t)} = g^{(t)}(V^{(t-1)}; \hat{W}^{(t)})$, where $g^{(t)} : \mathbb{R}^{C_{t-1}\times H_{t-1}\times W_{t-1}} \to \mathbb{R}^{C_t \times H_t \times W_t}$, represents the convolution operation. In particular,

$$U_j^{(t)} = \sum_{i=1}^{C_{t-1}} g^{(t)}_{i,j}\left(V_i^{(t-1)}; \hat{W}^{(t)}_{i,j}\right), \tag{8}$$

where $g^{(t)}_{i,j} : \mathbb{R}^{H_{t-1}\times W_{t-1}} \to \mathbb{R}^{H_t \times W_t}$ is a 2-D convolution operation with $\hat{W}^{(t)}_{i,j}$. Also, for ResNet, we formulate the relationship between the output feature map of a layer and the input feature map of the subsequent layer as

$$\begin{aligned}
&(i)\quad V^{(t)} = U^{(t)} \quad \forall t \notin T \\
&(ii)\quad V^{(t)} = U^{(t)} + V^{(s)} \quad \forall (s,t) \in \mathcal{P} \text{ and } C_t = C_s \\
&(iii)\quad \forall (s,t) \in \mathcal{P} \text{ and } C_t > C_s, \\
&\quad (iii-a)\quad V_j^{(t)} = U_j^{(t)} + V_j^{(s)} \quad \forall j \le C_s \text{ and} \\
&\qquad\qquad V_j^{(t)} = U_j^{(t)} \quad \forall j > C_s \\
&\quad (iii-b)\quad V^{(t)} = U^{(t)} + \tilde{V}^{(s)} \quad \text{where} \\
&\qquad\qquad \tilde{V}^{(s)} \text{ is } V^{(s)} \text{ after } 1\times 1 \text{ convolution.}
\end{aligned} \tag{9}$$

C.2 Inactive weights

Before we specify inactive weights, we first define two important terms (*trivially zero* and *meaningless*). A feature map is *trivially zero* if the feature map is zero for any input, $V^{(0)}$. A feature map is *meaningless* if the values in

the feature map do not have any effect on the final layer output feature map, $U^{(L)}$. Concretely, we state the definitions of trivially zero and meaningless in a cascading fashion.

Trivially zero We define trivially zero in the ascending order of t. Concretely, we define trivially zero in the following order $V^{(0)} \to U^{(1)} \to V^{(1)} \to U^{(2)} \to \cdots \to V^{(L-1)} \to U^{(L)}$.

1. $V_j^{(0)}$ in the input feature map is not trivially zero for all j.

2. $U_j^{(t)}$ is trivially zero if and only if $A_{i,j}^{(t)} = 0_{K_t,K_t}$ or $V_i^{(t-1)}$ is trivially zero for all $i \in [C_{t-1}]$ due to Equation (8).

3. In case of $V_j^{(t)}$, we divide the cases according to Equation (9).
   (i) $V_j^{(t)}$ is trivially zero if and only if $U_j^{(t)}$ is trivially zero.
   (ii) $V_j^{(t)}$ is trivially zero if and only if $U_j^{(t)}$ is trivially zero and $V_j^{(s)}$ is trivially zero.
   $(iii-a)$ If $j \leq C_s$, the condition is the same with (ii). Otherwise, the condition is the same with (i).
   $(iii-b)$ We suppose the output feature map of the 1×1 convolution, $\tilde{V}_j^{(s)}$, is not trivially zero. Therefore, $V_j^{(t)}$ is not trivially zero.

Meaningless We define meaningless in descending order of t. Concretely, we define meaningless in the following order $U^{(L)} \to V^{(L-1)} \to U^{(L-1)} \to V^{(L-2)} \to \cdots \to U^{(1)} \to V^{(0)}$.

1. $U_j^{(L)}$ in the final feature map is not meaningless for all j.

2. $V_i^{(t)}$ is meaningless if $A_{i,j}^{(t+1)} = 0_{K_{t+1},K_{t+1}}$ or $U_j^{(t+1)}$ is meaningless for all $j \in [C_{t+1}]$ due to Equation (8).

3. $U_i^{(t)}$ is meaningless if and only if $V_i^{(t)}$ is meaningless.

We now move on define *active weight* and *inactive weight* in Definition 1 with trivially zero and meaningless.

Definition 1 (Active weight, inactive weight)**.** *A weight $W_{i,j,a,b}^{(t)}$ is an **inactive weight** if 1) the weight is pruned ($A_{i,j,a,b}^{(t)} = 0$) or 2) the corresponding input channel feature map ($V_i^{(t-1)}$) is trivially zero or 3) the corresponding output channel feature map ($U_j^{(t)}$) is meaningless. Conversely, a weight $W_{i,j,a,b}^{(t)}$ is an **active weight** if 1) the weight is not pruned ($A_{i,j,a,b}^{(t)} = 1$) and 2) the corresponding input channel feature map ($V_i^{(t-1)}$) is not trivially zero and 3) the corresponding output channel feature map ($U_j^{(t)}$) is not meaningless.*

Note that only active weights should account for computation of the resource usage and the sum of the importance of weights. In this next subsection, we show that the inactive weights are provably excluded from the network pruned with our formulation.

C.3 Pruning consistency in our formulation

In our method, discrete variables $u^{(0:L)}$, $v^{(0:L)}$, and $q^{(1:L)}$ satisfy the constraints in Equation (10). We assume at least one of channel activation is set for each layer. Concretely, $\left\|u^{(t)}\right\|_1 \geq 1$ and $\left\|v^{(t)}\right\|_1 \geq 1 \quad \forall t$.

$$\left\|u^{(t)}\right\|_1 \geq 1 \text{ and } \left\|v^{(t)}\right\|_1 \geq 1 \quad \forall t \tag{10a}$$

$$u_j^{(t)} \leq \sum_{a,b} q_{j,a,b}^{(t)} \quad \text{and} \quad q_{j,a,b}^{(t)} \leq u_j^{(t)} \quad \forall t, j, a, b \tag{10b}$$

$$\begin{aligned}
&(i)\ \ v^{(t)} = u^{(t)} \quad \forall t \notin T \\
&(ii)\ \ u^{(t)} \preceq v^{(t)} \preceq u^{(t)} + v^{(s)} \quad \forall (s,t) \in \mathcal{P} \text{ and } C_t = C_s \\
&(iii)\ \ \forall (s,t) \in \mathcal{P} \text{ and } C_t > C_s, \\
&\qquad (iii-a)\ \ u_j^{(t)} \leq v_j^{(t)} \leq u_j^{(t)} + v_j^{(s)} \quad \forall j \leq C_s \text{ and} \\
&\qquad\qquad v_j^{(t)} = u_j^{(t)} \quad \forall j > C_s \\
&\qquad (iii-b)\ \ u^{(t)} \preceq v^{(t)}
\end{aligned} \tag{10c}$$

Lemma 1. *For $l \in [L]$, if $v_j^{(l-1)} = 1$, then $V_j^{(l-1)}$ is not trivially zero.*

Proof. We prove by mathematical induction with respect to l.

1. When $l = 1$, the statement is true since input data is not trivially zero.
2. Suppose the statement is true for $l = 1, \ldots, t$.
3. For j such that $v_j^{(t)} = 1$, we can think of three possible cases according to Equation (10c)

   (i) If $t \notin T$, $u_j^{(t)} = v_j^{(t)} = 1$. First, $\exists i, \quad v_i^{(t-1)} = 1$ since $\|v^{(t-1)}\|_1 \geq 1$ from Equation (10a). By the induction hypothesis, $V_i^{(t-1)}$ is not trivially zero. On the other hand, $\exists a, b \quad q_{j,a,b}^{(t)} = 1$ since $u_j^{(t)} \leq \sum_{a,b} q_{j,a,b}^{(t)}$ from Equation (10b). Then, $A_{i,j,a,b}^{(t)} = v_i^{(t-1)} q_{j,a,b}^{(t)} = 1$. By the second condition of trivially zero (2), $U_j^{(t)}$ is not trivially zero since $V_i^{(t-1)}$ is not trivially zero and $A_{i,j,a,b}^{(t)} = 1$. Also, by the third definition of trivially zero $(3-(i))$, $V_j^{(t)}$ is not trivially zero since $U_i^{(t)}$ is not trivially zero.
   (ii) If $\forall (s,t) \in \mathcal{P}$ and $C_t = C_s$, $u_j^{(t)} + v_j^{(s)} \geq v_j^{(t)} = 1$. Then, $u_j^{(t)} = 1$ or $v_j^{(s)} = 1$. If $u_j^{(t)} = 1, \quad U_j^{(t)}$ is not trivially zero as in (i). If $v_j^{(s)} = 1, \quad V_j^{(s)}$ is not trivially zero by the induction hypothesis. By the definition of trivially zero $(3-(ii))$, $V_j^{(t)}$ is not trivially zero since $V_j^{(s)}$ or $U_j^{(t)}$ is not trivially zero.
   (iii) $\forall (s,t) \in \mathcal{P}$ and $C_t > C_s$,
   $(iii-a)$ If $j \leq C_s$, the proof is the same with (ii). Otherwise, the proof is the same with (i).
   $(iii-b)$ By the definition of trivially zero $(3-(iii-b))$, $V_j^{(t)}$ is not trivially zero. In every possible cases, $V_j^{(t)}$ is not trivially zero and the statement is true for $l = t+1$.

By mathematical induction, for $l \in [L]$, if $v_j^{(l-1)} = 1$, then $V_j^{(l-1)}$ is not trivially zero. □

Lemma 2. *For $l \in [L]$, if $u_i^{(l)} = 1$, then $U_i^{(l)}$ is not meaningless.*

Proof. We prove by mathematical induction with respect to l.

1. When $l = L$, the statement is true since $U^{(L)}$ is not meaning less.
2. Suppose the statement is true for $l = t+1, \ldots, L$.
3. For i such that $u_i^{(t)} = 1$, $v_i^{(t)} = 1$ since $v_i^{(t)} \geq u_i^{(t)}$. Then, $\exists j \quad u_j^{(t+1)} = 1$ since $\|u^{(t+1)}\|_1 \geq 1$ from Equation (10a). By the induction hypothesis, $U_j^{(t+1)}$ is not meaningless. On the other hand, $\exists a, b \quad q_{j,a,b}^{(t+1)} = 1$

since $u_j^{(t+1)} \leq \sum_{a,b} q_{j,a,b}^{(t+1)}$ from Equation (10b). Then, $A_{i,j,a,b}^{(t+1)} = v_i^t q_{j,a,b}^{(t+1)} = 1$. By the definition of meaningless (2), $V_i^{(t)}$ is not meaningless. By the definition of meaningless (3), $U_i^{(t)}$ is not meaningless. The statement is true for $l = t$.

By mathematical induction, for $l \in [L]$, if $u_i^{(l)} = 1$, then $U_i^{(l)}$ is not meaningless. □

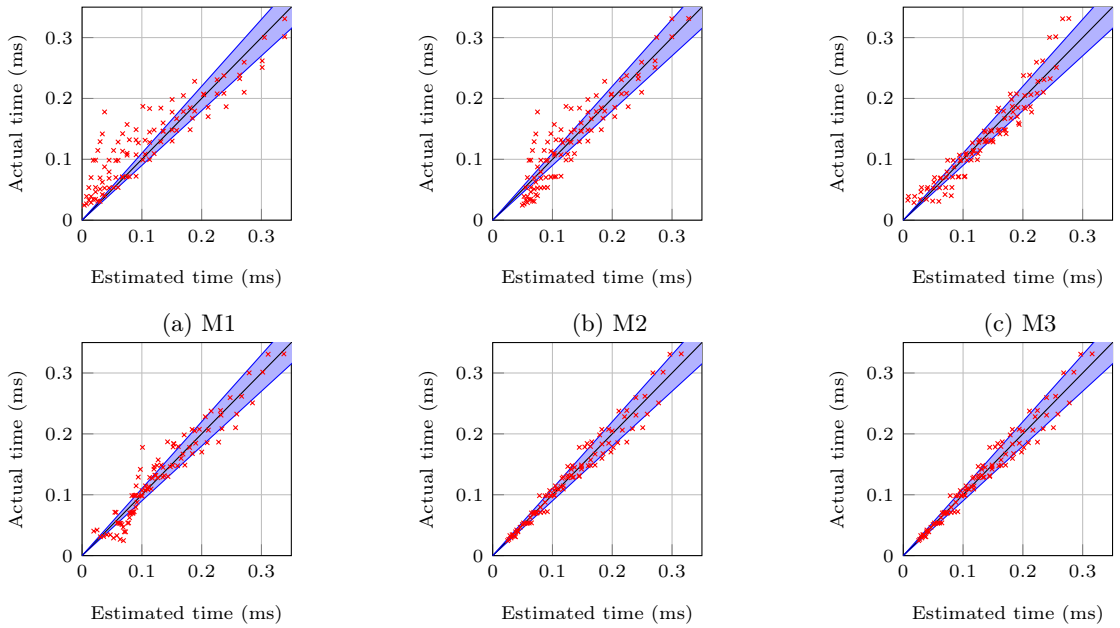
Proposition 2. *Optimizing over the input and output channel activation variables $u^{(0:L)}, v^{(0:L)}$ and shape column activation variables $q^{(1:L)}$ under the constraints in Equation (10) prevents the existence of any inactive weights in the pruned network guaranteeing exact computation of 1) resource usage and 2) the sum of the importance of active weights in the pruned network.*

Proof. Weight $W_{i,j,a,b}^{(l)}$ is not pruned if $A_{i,j,a,b}^{(l)} = 1$. If $A_{i,j,a,b}^{(l)} = 1$, then $u_j^{(l)} = 1$ and $v_i^{(l-1)} = 1$. Then, by Lemma 1 and Lemma 2, $V_i^{(l-1)}$ is not trivially zero and $U_j^{(l)}$ is not meaningless. By Definition 1, the weight $W_{i,j,a,b}^{(l)}$ is active. All the remaining weights in the network pruned with our method are active, which guarantees the exact specification of resource usage and sum of the importance of active weights in Equation (6). □

Proposition 1. *Optimizing over the input and output channel activation variables $r^{(0:L)}$ and shape column activation variables $q^{(1:L)}$ under the constraints in Equation (4) prevents the existence of any inactive weights in the pruned network guaranteeing exact computation of 1) resource usage and 2) the sum of the importance of active weights in the pruned network.*

Proof. Proposition 1 is the special case of Proposition 2 when $u^{(t)} = v^{(t)} \ \left(:= r^{(t)}\right)$, $q_{j,a,b}^{(t)} = u_j^{(t)}$, and $\mathcal{P} = \emptyset \ \ \forall t, j, a, b$. □

D Selecting a model for inference time estimation



(a) M1 (b) M2 (c) M3

(d) M4 (e) M5 (f) M6

Figure 7: Estimated inference time vs. actual wall-clock inference time of convolution operations using each prediction models. The blue area indicates the error of the estimated value is under 10% with respect to the actual value.

In this section, we aim to find a quadratic model that accurately predicts the inference time of convolution operations. We then analyze the coefficients of the proposed model and provide some intuitive meanings behind each term. Finally, we verify the proposed model is able to accurately predict the inference time of the whole

pruned network as well, using ResNet-50 as the baseline network. We use a machine with Intel Xeon E5-2650 CPU and Titan XP GPU as default. In Appendix D.3, we also experiment on another machine with Xeon Gold 5220R CPU and Geforce RTX 2080 Ti GPU.

Before we start, we define some notations to describe the convolution operation. We denote the kernel size, the stride, and the padding of a convolution operation as k, s and p, respectively. Also, we denote the number of the input channels and the output channels as n_{in} and n_{out}. Finally, we denote the shape of the input and the output feature map as $(n_{\text{in}}, H_{\text{in}}, W_{\text{in}})$ and $(n_{\text{out}}, H_{\text{out}}, W_{\text{out}})$, respectively.

D.1 Model selection

We first build a candidate set of quadratic models that estimate the inference time of a convolution operation and denote the models from M1 to M6, as described in Table 7. Then, we measure the actual wall-clock inference times of the convolution operations without bias terms, where $(H_{\text{in}}, W_{\text{in}}) = (56, 56)$, $k = 3$, $s = 1$, $p = 1$, and $(n_{\text{in}}, n_{\text{out}}) \in \{(16x, 16y) \,|\, x, y \in [10]\}$. Concretely, the inference time samples can be represented as $\{(n_{\text{in}}^{(i)}, n_{\text{out}}^{(i)}, \text{WallClock}^{(i)})\}_{i=1}^{100}$. Using these samples as the dataset, we find the best coefficients for each candidate models via least square regression. For example, in the case of M6, we solve the following optimization problem to find α^*, β^* , γ^*, and δ^* that best fits the inference time samples:

$$\min_{\alpha,\beta,\gamma,\delta} \sum_{i=1}^{100} \left(\text{WallClock}^{(i)} - \alpha - \beta n_{\text{in}}^{(i)} - \gamma n_{\text{out}}^{(i)} - \delta n_{\text{in}}^{(i)} n_{\text{out}}^{(i)} \right)^2 .$$

Figure 7 shows the estimated inference time versus the wall-clock inference time when using each quadratic model with its best coefficients. We observe that M5 and M6 show the most successful estimation performance. More concretely, the mean percent error (MPE) and the R^2 value in Table 7 verifies that M5 and M6 are the most accurate models. That said, since M5 is a simpler model compared to M6 and the performance gap between the two models is negligible, we select M5 as our prediction model to estimate the inference time of convolution operations and analyze the coefficients of this model.

Table 7: Description of the candidate quadratic models (M1-M6) for estimating the inference time of a convolution operation and their Mean Percent Error (%) and R^2 values after regression over the inference time samples.

Notation	Model	Mean Percent Error (%)	R^2
M1	$\delta n_{\text{in}} n_{\text{out}}$	28.9	0.687
M2	$\alpha + \delta n_{\text{in}} n_{\text{out}}$	26.8	0.846
M3	$\alpha + \beta n_{\text{in}} + \gamma n_{\text{out}}$	19.9	0.912
M4	$\alpha + \gamma n_{\text{out}} + \delta n_{\text{in}} n_{\text{out}}$	19.3	0.918
M5	$\alpha + \beta n_{\text{in}} + \delta n_{\text{in}} n_{\text{out}}$	7.68	**0.960**
M6	$\alpha + \beta n_{\text{in}} + \gamma n_{\text{out}} + \delta n_{\text{in}} n_{\text{out}}$	**7.67**	**0.960**

D.2 Analysis of the coefficients

We conduct additional experiments to analyze the coefficients β and δ of M5 by varying H_{in}, W_{in}, k, and s. We consider 96 hyperparameter configurations of $(k, s, p, H_{\text{in}}, W_{\text{in}})$, sampled from $\{(2x-1, y, x-1, 52+4z, 52+4z) \,|\, (x, y, z) \in [4] \times [2] \times [12]\}$. For each sampled configuration, we measure the wall-clock inference times of 25 different convolution operations with $(n_{\text{in}}, n_{\text{out}}) \in \{(16x, 16y) \mid x, y \in [5]\}$ and fit the parameters α, β, and δ. As a result, we get 96 pairs of β and δ values each corresponding to the 96 configurations. Examining the results, we observe a strong linear relationship between (β, δ) and $k^2 \frac{H_{\text{in}} W_{\text{in}}}{s^2}$, as illustrated in Figure 8. Combining this with the fact that the FLOPs of a convolution operation is equal to $k^2 \frac{H_{\text{in}} W_{\text{in}}}{s^2} n_{\text{in}} n_{\text{out}}$, we can hypothesize that $\delta n_{\text{in}} n_{\text{out}}$ in M5 represents the contribution of the computation cost (FLOPs) to the inference time. Also, the term βn_{in} in M5 is proportional to $k^2 \frac{H_{\text{in}} W_{\text{in}}}{s^2} n_{\text{in}}$. We assume that βn_{in} represents the contribution of the memory access cost (MAC), as storing $\frac{H_{\text{in}} W_{\text{in}}}{s^2}$ patches of the input feature map, each of size $k^2 n_{\text{in}}$, accounts for the major part of the MAC. As more detailed calculation of the MAC may differ depending on the software and the hardware used, we leave further analysis as a future work.

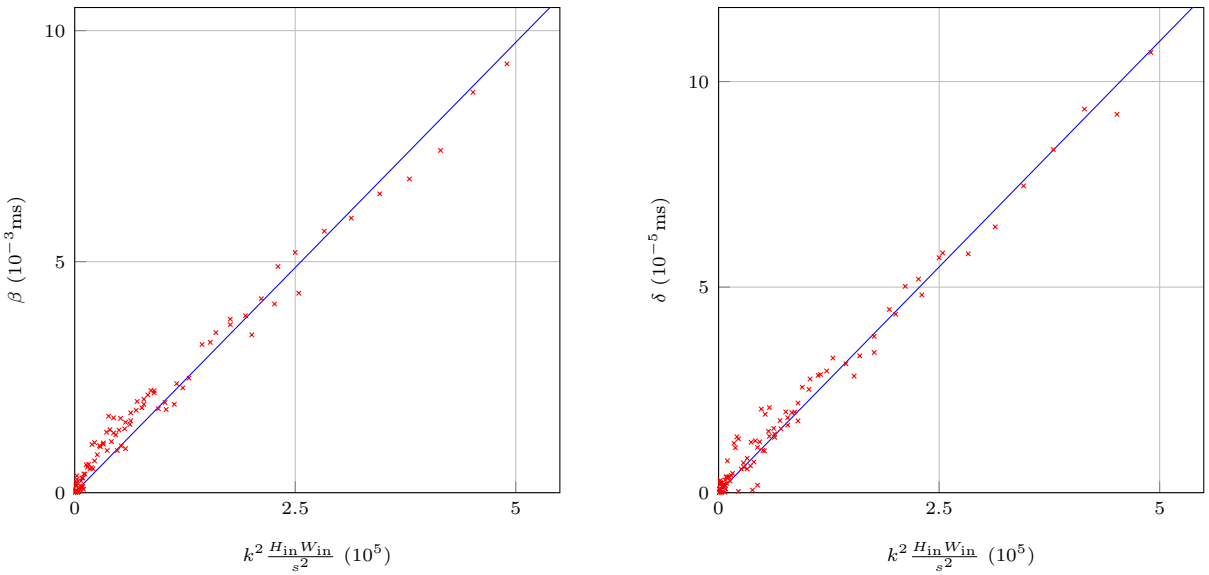


Figure 8: Plot of $k^2 \frac{H_{\text{in}} W_{\text{in}}}{s^2}$ vs. β (left) and δ (right). The R^2 values without bias terms are 0.971 (left) and 0.979 (right). The blue lines are the linear lines without bias terms which best fit these points.

D.3 Inference time estimation on ResNet-50

We show our quadratic model can accurately predict the inference time of the whole ResNet-50 network as well as its convolution modules. Here, we also predict the inference time using M6. This is to take into account the batch normalization and the activation function that come after the convolution operation since these operations are only dependent on n_{out}. Considering these two functions with M6 enables a more accurate prediction of the wall-clock inference time of the whole network.

As a first step, we estimate the inference time of each convolution operation in ResNet-50. In particular, on M5, we estimate the inference time of only the convolution operation. On M6, we estimate the inference time of the convolution operation along with its subsequent operations (batch normalization and activation function). The results in Table 8 show that M5 and M6 successfully estimate the actual inference time on two different machines. Concretely, the average MPE value of M5 and M6 at a Titan XP machine are 6.48% and 7.98%, respectively. Also, the average MPE value of M5 and M6 at a RTX 2080 Ti machine are 11.66% and 10.66%, respectively.

Next, we estimate the wall-clock inference time of the pruned ResNet-50 with M6 by summing all of the estimated inference time of the convolution modules, including the batch normalization and activation function. Figure 9 shows the estimated inference time and the corresponding wall-clock inference time of ResNet-50 while varying the number of channels. Our proposed quadratic model M6 achieves 3.1% and 3.5% error rates on average when estimating the inference time of ResNet-50 from Titan XP and RTX 2080 Ti, respectively.

E Coordinate descent style optimization

In this section, we provide a block coordinate descent-style optimization algorithm for solving Equation (6) in Algorithm 1. Note that Equation (6) is the generalized version of Equation (3) in Section 3.2.

We first set all shape column activation variables to ($q^{(t)}_{j,a,b} = u^{(t)}_j \ \forall t, j, a, b$). Then, we optimize over the input and output channel activation variables ($u^{(0:L)}$, $v^{(0:L)}$) in a block coordinate descent fashion with the resource constraint M/γ where γ is the average spatial sparsity smaller than 1. Then, we optimize over the shape column activation variables $q^{(1:L)}$, fixing the input and output channel activation variables. In all experiments using Algorithm 1, γ is decreased from 'M/(Resource requirment of the initial network)' to 1.0 with a step size of 0.1. After the pruning procedure, we employ one round of finetuning on the pruned network. Note that in Algorithm 1, we denote the objective function in Equation (6) as $f(\cdot)$ when the shape column activations are all forced to match the output channel activations ($q^{(t)}_{j,a,b} = u^{(t)}_j \ \forall t, j, a, b$). z denotes the concatenated variables of input and

Table 8: Mean Percent Error (MPE) and R^2 values for estimating the inference time of each convolution layer in ResNet-50 architecture on a machine with Intel Xeon E5-2650 CPU and Titan XP GPU ('Titan XP'), and a machine with Xeon Gold 5220R CPU and Geforce RTX 2080 Ti GPU ('RTX 2080 Ti'). On M5, we estimate the inference time of only the convolution operation, while on M6 we also take into account the inference of the subsequent batch normalization and activation function (ReLU). 'Avg' represents the averaged MPE and R^2 value of all convolutional layers in ResNet-50.

(a) Convolution, M5

								Titan XP		RTX 2080 Ti	
	Layer	H_{in}	W_{in}	n_{in}	n_{out}	k	s	MPE (%)	R^2	MPE (%)	R^2
conv1	1	224	224	3	64	7	2	5.07	0.85	4.65	0.10
Block1	2	56	56	64	64	1	1	1.21	0.52	0.95	0.42
	3	56	56	64	64	3	1	6.81	0.97	9.03	0.93
	4	56	56	64	256	1	1	3.46	0.40	10.30	0.73
	5	56	56	256	64	1	1	12.86	0.80	12.84	0.87
Block2	11	56	56	256	128	1	1	13.25	0.83	13.83	0.90
	12	28	28	128	128	3	2	6.93	0.59	7.49	0.90
	13	28	28	128	512	1	1	1.03	0.33	11.20	0.56
	14	28	28	512	128	1	1	7.60	0.55	14.19	0.78
	15	28	28	128	128	3	1	15.42	0.87	10.83	0.86
Block3	23	28	28	512	256	1	1	11.57	0.78	11.80	0.93
	24	14	14	256	256	3	2	6.45	0.95	6.69	0.82
	25	14	14	256	1024	1	1	1.93	0.51	13.13	0.63
	26	14	14	1024	256	1	1	2.77	0.49	11.64	0.86
	27	14	14	256	256	3	1	7.93	0.95	17.63	0.76
Block4	41	14	14	1024	512	1	1	13.17	0.66	12.75	0.90
	42	7	7	512	512	3	2	10.70	0.97	11.35	0.91
	43	7	7	512	2048	1	1	6.32	0.58	9.84	0.57
	44	7	7	2048	512	1	1	11.75	0.83	9.84	0.72
	45	7	7	512	512	3	1	8.72	0.97	11.64	0.92
Avg								6.48	0.67	11.66	0.74

(b) Convolution, batch normalization, and activation function, M6

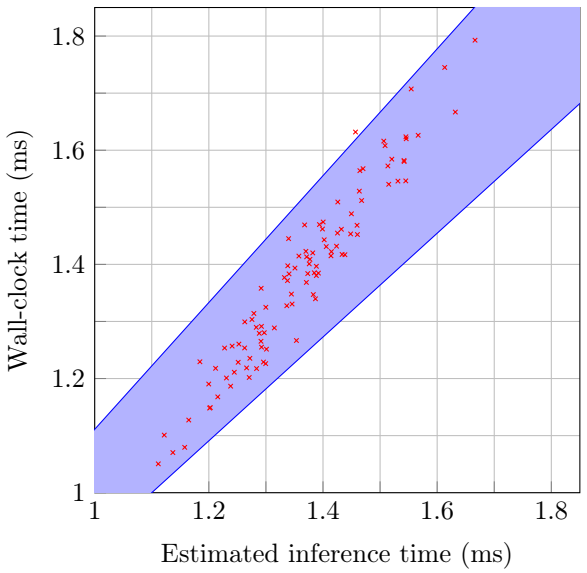
								Titan XP		RTX 2080 Ti	
	Layer	H_{in}	W_{in}	n_{in}	n_{out}	k	s	MPE (%)	R^2	MPE (%)	R^2
conv1	1	224	224	3	64	7	2	4.79	0.79	10.20	0.06
Block1	2	56	56	64	64	1	1	0.61	0.33	0.36	0.60
	3	56	56	64	64	3	1	5.06	0.97	4.99	0.96
	4	56	56	64	256	1	1	10.44	0.84	14.67	0.91
	5	56	56	256	64	1	1	10.30	0.91	10.00	0.95
Block2	11	56	56	256	128	1	1	9.67	0.92	9.59	0.95
	12	28	28	128	128	3	2	11.90	0.65	8.34	0.90
	13	28	28	128	512	1	1	5.87	0.82	9.83	0.82
	14	28	28	512	128	1	1	10.60	0.67	12.28	0.88
	15	28	28	128	128	3	1	14.35	0.91	10.28	0.86
Block3	23	28	28	512	256	1	1	10.43	0.89	9.55	0.95
	24	14	14	256	256	3	2	5.84	0.96	9.46	0.82
	25	14	14	256	1024	1	1	6.27	0.66	11.24	0.83
	26	14	14	1024	256	1	1	6.89	0.67	10.24	0.90
	27	14	14	256	256	3	1	5.40	0.98	15.93	0.76
Block4	41	14	14	1024	512	1	1	13.38	0.78	11.55	0.91
	42	7	7	512	512	3	2	10.21	0.98	14.79	0.86
	43	7	7	512	2048	1	1	9.72	0.69	9.45	0.86
	44	7	7	2048	512	1	1	10.07	0.91	7.23	0.92
	45	7	7	512	512	3	1	5.63	0.98	10.90	0.93
Avg								7.98	0.81	10.66	0.85

Algorithm 1 Succinct channel and spatial pruning optimization via QCQP

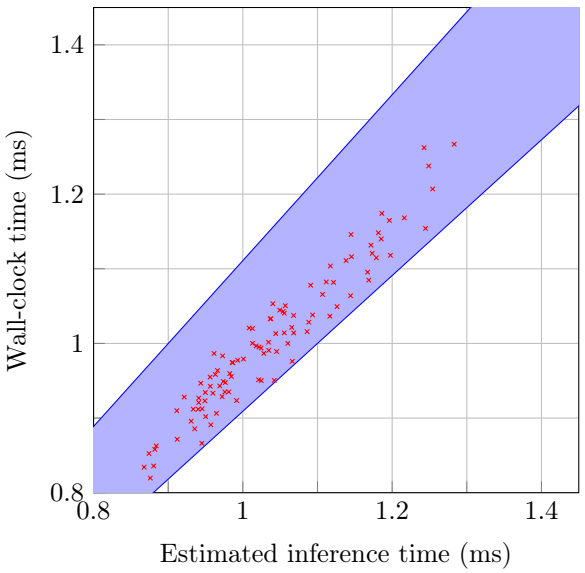
Input : B, M, γ, $a_l, a'_l, b'_l, I^{(l)}, \quad \forall l$
Initialize $u^{(0:L)}, v^{(0:L)}$.
$M \coloneqq M/\gamma$.
for $n = 1, \ldots,$MAXITER **do**
  for $i = 0, \ldots, L - B + 1$ **do**
    $z = \left[u^{(i:i+B-1)}, v^{(i:i+B-1)}\right]$
    $\tilde{f}(z) = f(z;$ rest of the variables in u, v fixed$)$.
    Optimize $\tilde{f}(z)$ with respect to z under the constraints in Equation (6).
  end for
end for
$M \coloneqq \gamma M$
Optimize Equation (6) with respect to $q^{(1:L)}$ while fixing $u^{(0:L)}$ and $v^{(0:L)}$.
Output : $u^{(0:L)}, v^{(0:L)}, q^{(1:L)}$

output channel activation in the target block.

We additionally conducted an experiment to check the CPLEX performance of Algorithm 1 compared to direct optimization on Equation (6), which we denote as Algorithm 0. However, Algorithm 0 is not scalable even in ResNet-20 on CIFAR-10. Therefore we compare Algorithm 0 and 1 for the first eight layers of ResNet-56. We set $B = 2$ and adjust γ with a step size of 0.1. Algorithm 1 succeeds in increasing the objective value to 100.16 in 12 minutes, while Algorithm 0 reaches 106.27 in 1 hour. Also, Algorithm 1 requires only 3 hours and 4GB memory for pruning ResNet-50 on ImageNet.



(a) ResNet50 and input (224×224) with Titan XP GPU



(b) ResNet50 and input (224×224) with Geforce RTX2080 Ti GPU

Figure 9: Estimated inference time vs. actual wall-clock time of ResNet-50 architecture while varying the number of channels at a machine with Intel Xeon E5-2650 CPU and Titan XP GPU (a), and a machine with Xeon Gold 5220R CPU and Geforce RTX 2080 Ti GPU (b). The blue area indicates the error of the estimated value is under 10% with respect to the actual wall-clock time.

F Implementation details and Additional experiment results

F.1 Implementation details

For ResNet experiments, we mostly follow the implementation from FPGM (He et al., 2019). We apply batch normalization and remove bias weight in every convolution layer. Zero padding and 1×1 convolution are used as the downsampling technique in CIFAR-10 (Krizhevsky et al., 2009) and ImageNet (Russakovsky et al., 2015), respectively.

For CIFAR-10 experiments on ResNet (He et al., 2016) architectures, we finetune the pruned model from the pretrained network given in He et al. (2019) and follow the protocol of He et al. (2019) for fair comparison. We finetune the pruned network for 200 epochs with batch size 128 and initial learning rate 0.01. Then, we adjust the learning rate at 60, 120, and 160 epoch by multiplying 0.2 each time. We use SGD optimizer with momentum 0.9, weight decay 5×10^{-4}, and Nesterov momentum. For CIFAR-10 experiments on the DenseNet-40 architecture, we finetune the pruned network for 300 epochs with batch size 128 and initial learning rate 0.1. We adjust the learning rate at 150 and 225 epoch by multiplying 0.1. Here, we also use SGD optimizer with momentum 0.9, weight decay 5×10^{-4}, and Nesterov momentum.

For ImageNet experiments on ResNet architectures, we follow the protocol of FPGM and start from the pretrained network provided by PyTorch (Paszke et al., 2017). We finetune the pruned network for 80 epochs on ImageNet with batch size 384 and the initial learning rate of 0.015. Then, we adjust the learning rate at 30 and 60 epoch by multiplying 0.1. Here, we use SGD optimizer with momentum 0.9 and weight decay 10^{-4}.

For ImageNet experiments on EfficientNet-B0 (Tan and Le, 2019), we use RMSProp optimizer with weight decay 0.9, momentum 0.9, and weight decay 1×10^{-5}. We train the baseline network for 250 epochs using batch size 256 and initial learning rate 0.008 that decays by 0.97 every 2.4 epochs with 3 warmup epochs. Then, we prune the baseline network and train the pruned networks using the same schedule with the baseline network except for using the initial learning rate 0.0008. For the regularization of both the baseline network and the pruned network, we use RandAugment (Cubuk et al., 2020), color jitters of factor 0.4, stochastic depth (Huang et al., 2016) with survival probability 0.8, and dropout (Srivastava et al., 2014) of rate 0.2.

Table 9: Pruned accuracy and accuracy drop from the baseline network at given pruning ratios on various ResNet architectures (ResNet-20,32,56) at CIFAR-10.

Method	IC	Baseline acc	Pruned acc↑	Acc drop↓	Pruning ratio(%)↑
Network: Resnet-20					
FPGM (He et al., 2019)	F	92.21 (0.18)	91.26 (0.24)	0.95	54.0
ours-c	F	92.21 (0.18)	91.26 (0.18)	0.95	**54.1**
ours-cs	F	92.21 (0.18)	**92.02**(0.10)	**0.19**	54.0
SFP (He et al., 2018a)	F	92.20 (0.18)	90.83 (0.31)	1.37	42.2
FPGM (He et al., 2019)	F	92.21 (0.18)	91.72 (0.20)	0.49	42.2
ours-c	F	92.21 (0.18)	92.27 (0.17).	-0.06	**42.3**
ours-cs	F	92.21 (0.18)	**92.35** (0.10)	**-0.14**	42.2
Network: Resnet-32					
FPGM (He et al., 2019)	F	92.88 (0.86)	91.96 (0.76)	0.92	**53.2**
ours-c	F	92.88 (0.86)	92.22 (1.02)	0.66	**53.2**
ours-cs	F	92.88 (0.86)	**92.78** (0.97)	**0.10**	**53.2**
SFP (He et al., 2018a)	F	92.63 (0.70)	92.08 (0.08)	0.55	41.5
FPGM (He et al., 2019)	F	92.88 (0.86)	92.51 (0.90)	0.37	41.5
ours-c	F	92.88 (0.86)	92.42 (0.77)	0.46	**42.7**
ours-cs	F	92.88 (0.86)	**92.83** (0.83)	**0.05**	**42.7**
Network: Resnet-56					
SFP (He et al., 2018a)	F	93.59 (0.58)	92.26 (0.31)	1.33	52.6
FPGM (He et al., 2019)	F	93.59 (0.58)	93.49 (0.13)	0.10	52.6
SCP (Kang and Han, 2020)	T*	93.69	93.23	0.46	51.5
ours-c	F	93.59 (0.58)	93.37 (0.96)	0.22	**52.7**
ours-cs	F	93.59 (0.58)	**93.69** (0.69)	**-0.10**	52.6

For ImageNet experiments on VGG-16 (Simonyan and Zisserman, 2015), we follow the protocol of Molchanov et al. (2017) for network training. We start from the pretrained network from Pytorch. We finetune the pruned network using SGD optimizer with a constant learning rate 10^{-4} and batch size 32 for 5 epochs.

F.2 CIFAR-10 network size constraint

We conduct the experiments of another resource constraint, network size on ResNet architectures. In the experiment tables Table 9, network size of the pruned network is computed according to the resource specifications in Equations (3) and (4). 'Pruning ratio' in the tables denotes the ratio of pruned weights among the total weights in baseline networks. Also, we ignore the extra memory overhead for storing the shape column activations due to its negligible size compared to the total network size.

F.3 MobileNetV2

We additionally apply our pruning method on MobileNetV2 (Sandler et al., 2018). We start from the pretrained network with accuracy 71.88 provided by Pytorch. Then, we finetune the pruned network for 150 epochs using SGD optimizer with weight decay 0.00004, momentum 0.9, and batch size 256. For the training schedule, we apply a learning rate warm-up for the initial five epochs, which steps up from 0 to 0.05. Then, we use the cosine learning rate decay for the remaining epochs.

We show our experiment results with three recent pruning baselines NetAdapt (Yang et al., 2018), AMC (He et al., 2018b), and MetaPruning (Liu et al., 2019) in Table 10. '(tuned)' indicates that the normalizing factor, γ_l, is tuned with grid search. A fixed value is used otherwise. Our method shows performance competitive to the other baselines, NetAdapt, AMC and MetaPruning. However, we note that our method is much more efficient than

Table 10: Top1 pruned accuracy and accuracy drop from the baseline network at given FLOPs on MobileNetV2 architecture at ImageNet

Method	Top1 Pruned Acc↑	Top1 Acc drop↓	FLOPs(%)↓
NetAdapt (Yang et al., 2018)	70.9	0.9	70
AMC (He et al., 2018b)	70.8	1.0	70
MetaPruning (Liu et al., 2019)	**71.2**	**0.6**	69
ours-c	70.8	1.0	**67**
ours-cs	70.2	1.6	**67**
ours-c (tuned)	71.0	0.8	**67**
ours-cs (tuned)	70.9	0.9	**67**

NetAdapt, AMC, and MetaPruning since NetAdapt requires repetitive finetuning steps for the proposed networks, AMC requires repetitive trial and error steps to train DDPG (Lillicrap et al., 2016) agent, and MetaPruning trains PruningNet of which network size is at least 30 times bigger than that of original model.

F.4 FCN-32s for segmentation

We apply our pruning method on FCN-32s (Long et al., 2015) for segmentation on PASCAL Visual Object Classes Challenge 2011 dataset. Then, we evaluate the segmentation performance with a widely-used measure, mean Intersection over Union (mIoU).

We use SGD optimizer with weight decay 0.0005, momentum 0.99, and batch size 1. For the original network to be pruned, we train FCN-32s for 25 epochs with a constant learning rate 8×10^{-11}. Then, we prune the original network of which mIoU 63.57 (%) and finetune the pruned network for 25 epochs with a constant learning rate 8×10^{-11}.

We show our experiment results in Figure 10. The pruned network reduces the FLOPs by 27% with 0.15 (%) mIoU drop for 'ours-c' and 0.09 (%) mIoU drop for 'ours-cs'.

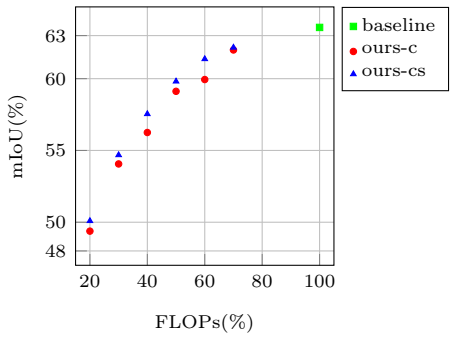


Figure 10: The plot of mIoU (%) versus FLOPs (%) on FCN-32s.

G Spatial pattern appearing in pruned network

'ours-cs' discovers diverse spatial patterns in convolution weights, as illustrated in Figure 11.

H Ablation study

H.1 Experiments on various FLOPs constraint

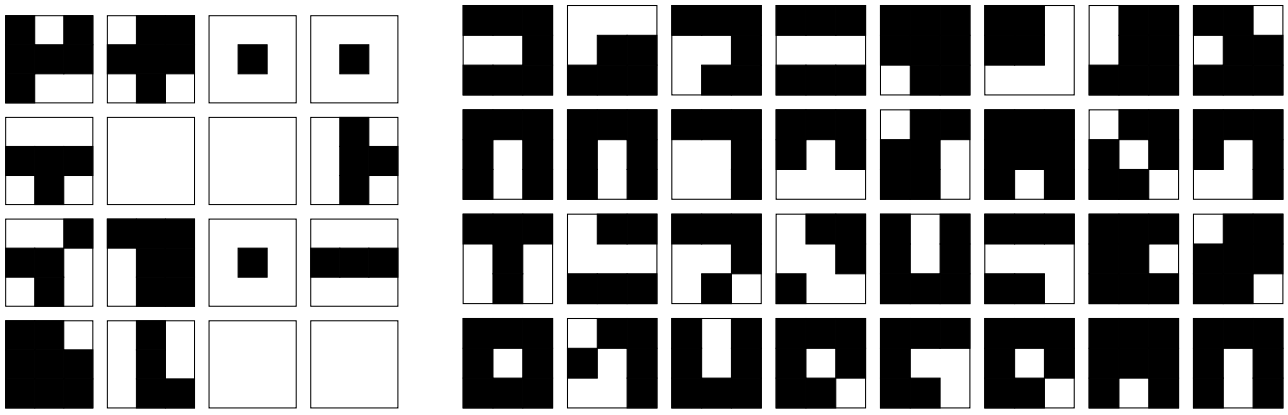
In Figure 12, we prune and finetune the ResNet architectures under various FLOPs constraints with 'ours-c', 'ours-cs', and FPGM. 'ours-cs' outperforms 'ours-c' and FPGM under almost all FLOPs constraints.

H.2 Experiments on various network size constraint

In Figure 13, we prune and finetune the ResNet architectures under various network size constraints with 'ours-c', 'ours-cs', and FPGM. 'ours-cs' outperforms 'ours-c' and FPGM on almost all network size constraints.

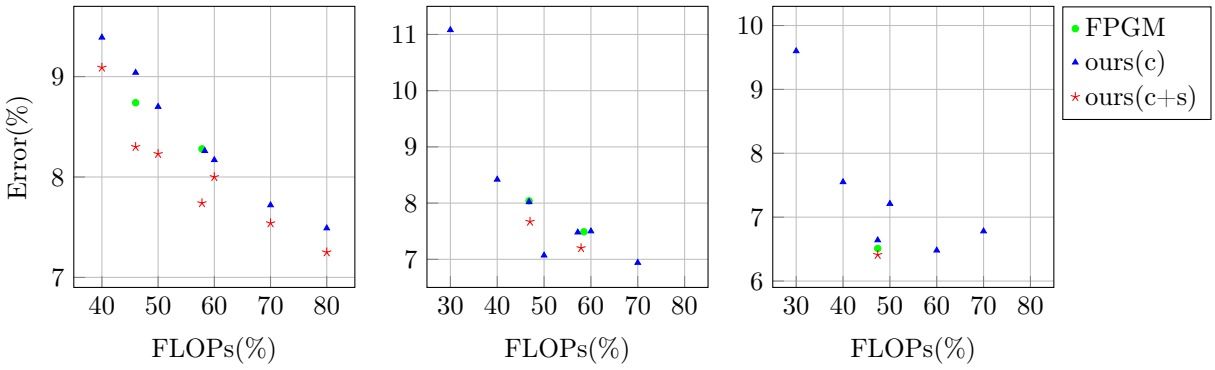
H.3 Ablation study on the possible pairs of resource usage (FLOPs and pruning ratio)

In the real-world, the resource constraint for network pruning may vary significantly in terms of how much each resource is available. In some cases, we may allow high FLOPs but strictly limit the network size, while in other cases, low FLOPs are much more important. However, when we prune the channels greedily, possible pairs of



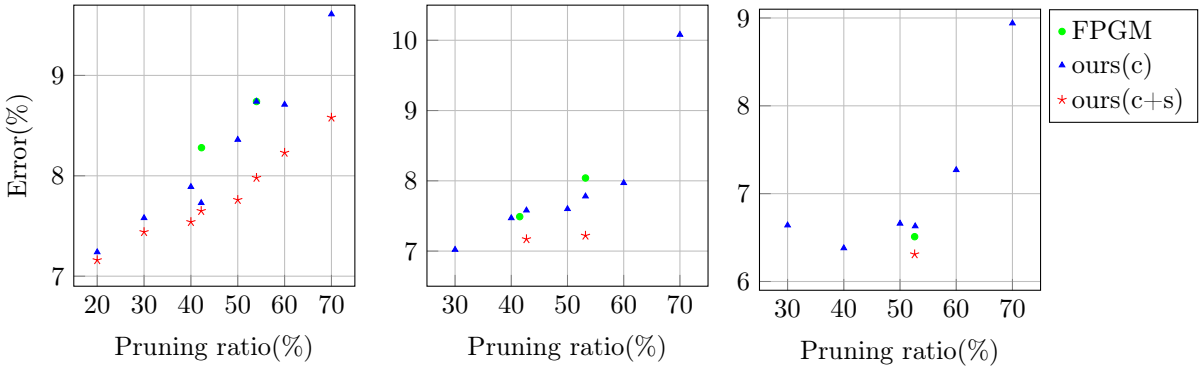
(a) $q^{(1)}$ (b) $q^{(8)}$

Figure 11: Visualization of shape column activations in the first convolution layer with $q^{(1)} \in \{0,1\}^{16\times3\times3}$ (left) and shape column activations in the eighth convolution layer with $q^{(8)} \in \{0,1\}^{32\times3\times3}$ (right) in ResNet-20 after pruned by 'ours (c+s)'. Black area indicates that the shape column activation is set.



(a) ResNet-20 (b) ResNet-32 (c) ResNet-56

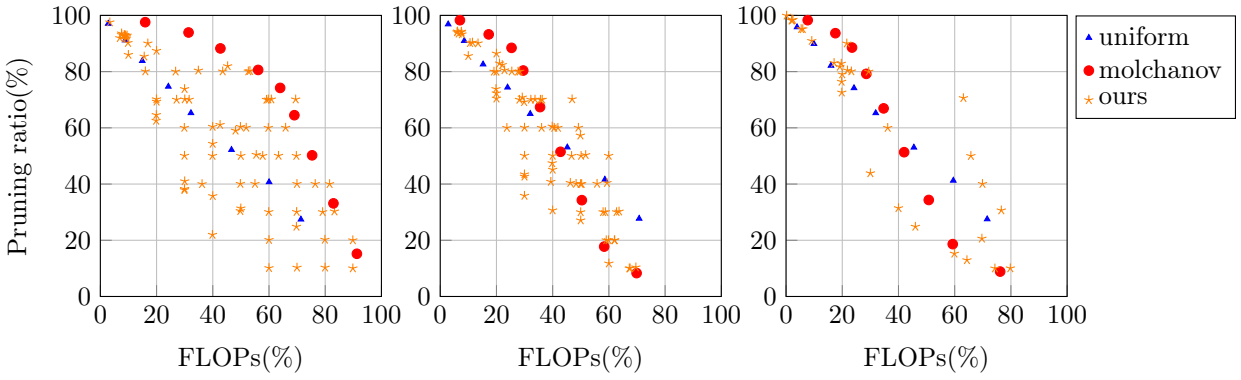
Figure 12: The plots of classification error (%) versus FLOPs (%) on various ResNet architectures.



(a) ResNet-20 (b) ResNet-32 (c) ResNet-56

Figure 13: The plots of classification error (%) versus pruning ratio (%) on various ResNet architectures.

resource usages are limited and nonadjustable. In contrast, our method can target any resource budget pairs. Figure 14 shows the possible pairs (FLOPS and pruning ratio) from three different pruning methods: 'uniform', which greedily prunes channels layer-wise, 'molchanov', which greedily prunes the channels from all layers, and



(a) ResNet-20 (b) ResNet-32 (c) ResNet-56

Figure 14: The possible pairs of resource usage (FLOPs and pruning ratio) from three different pruning criteria : (1) 'uniform' which greedily prunes channels layer-wise, (2) 'molchanov' which greedily prunes the channels from all layers, and (3) 'ours' which prunes via QCQP.

'ours', which prunes via QCQP. 'ours' results in diverse pairs of target resources which cover the pairs of 'uniform' and 'molchanov'.