# Soft Masking for Cost-Constrained Channel Pruning

Ryan Humble[1]*, Maying Shen[2], Jorge Albericio Latorre[2], Eric Darve[1], and Jose Alvarez[2]

[1] Stanford University, Stanford CA 94305, USA
{ryhumble,darve}@stanford.edu
[2] NVIDIA, Santa Clara CA 95051, USA
{mshen,jalbericiola,josea}@nvidia.com

**Abstract.** Structured channel pruning has been shown to significantly accelerate inference time for convolution neural networks (CNNs) on modern hardware, with a relatively minor loss of network accuracy. Recent works permanently zero these channels during training, which we observe to significantly hamper final accuracy, particularly as the fraction of the network being pruned increases. We propose Soft Masking for cost-constrained Channel Pruning (SMCP) to allow pruned channels to adaptively return to the network while simultaneously pruning towards a target cost constraint. By adding a soft mask re-parameterization of the weights and channel pruning from the perspective of removing input channels, we allow gradient updates to previously pruned channels and the opportunity for the channels to later return to the network. We then formulate input channel pruning as a global resource allocation problem. Our method outperforms prior works on both the ImageNet classification and PASCAL VOC detection datasets.

**Keywords:** Neural network pruning, Model compression

## 1 Introduction

Deep neural networks have rapidly developed over the last decade and come to dominate many traditional algorithms in a wide range of tasks. In particular, convolutional neural networks (CNNs) have shown state-of-the-art results on a range of computer vision tasks, including classification, detection, and segmentation. However, modern CNNs have grown in size, computation, energy requirement, and prediction latency, as researchers push for accuracy improvements. Unfortunately, these models can now easily exceed the capabilities of many edge computing devices and requirements of real-time inference tasks, such as those found in autonomous vehicle applications.

Since neural networks have been shown to be heavily over-parameterized [52], one popular method for reducing the computation and prediction latency is

* Work performed during a NVIDIA internship

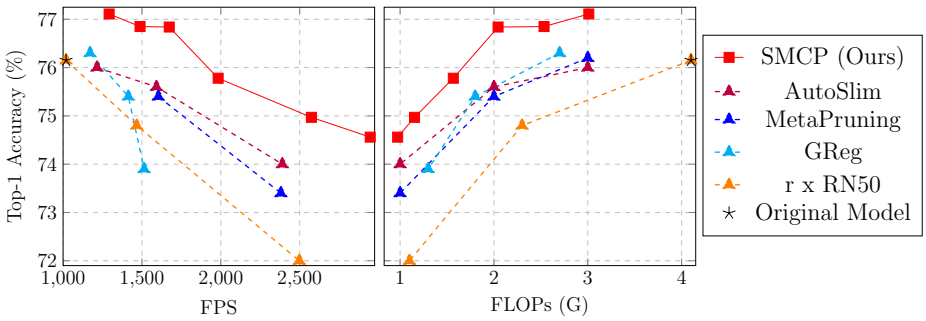


**Fig. 1.** Top-1 accuracy tradeoff curve for pruning ResNet50 on the ImageNet classification dataset using a latency cost constraint. Baseline is from PyTorch [34] model hub. Accuracy against FPS speed (left) and FLOPs (right) show the benefit of our method, particularly at high pruning ratios. For FPS, top-right is better. For FLOPs, top-left is better. FPS measured on an NVIDIA TITAN V GPU.

to prune (or remove) portions of the neural network, ultimately yielding a model with fewer parameters. Due to the strict requirements for many deployment applications, a large fraction of the parameters often must be removed; we focus on this regime, which we refer to as the high pruning ratio regime. Towards this aim, many pruning methods have been proposed to identify and remove those parameters that are least important for inference [1,15,27,20,29,47,51]. Since each layer of the network involves a different computation and associated computational burden, each parameter does not contribute equally to the final network inference cost, typically measured as FLOPs or latency, so more recent works have focused on pruning the network subject to explicit cost constraints. To maximize inference speedup on modern hardware (e.g., GPUs), these works largely focus on channel pruning [21,27,29,37,42,50].

However, in general, existing pruning works permanently remove the network parameters along these channels, zeroing the network weights and preventing the channel from being used during the rest of training. Particularly at high pruning ratios, where a significant fraction of the total channels in the network must be removed, the decisions on which channels to remove early during pruning are potentially myopic. Moreover, as a large number of channels are removed, the gradients to the remaining channels in each layer are significantly disrupted and can grow quite substantially due to the batch normalization layers ubiquitous in modern CNNs. This interferes with both network training and the identification of which further channels to remove.

In this work, we introduce a novel channel pruning approach for neural networks that is particularly suitable for large pruning ratios. The core of our approach relies on regularly rewiring the network sparsity, through soft masking of the network weights, to minimize the accuracy drop for large pruning ratios. The introduction of soft masking allows previously pruned channels to later be

restored to the network, instead of being permanently pruned. Additionally, to mitigate the effect of large gradient magnitudes caused by removing many channels, we incorporate a new batch normalization scaling approach. Lastly, we formulate channel pruning under a cost constraint as a resource allocation problem and show it can be efficiently solved. All together, we refer to this method as Soft Masking for cost-constrained Channel Pruning (SMCP).

Our main contributions are:

1. We demonstrate that a network's channel sparsity can be adaptively rewired, using a soft mask re-parameterization of the network weights, and that this requires channel pruning to be performed along input, instead of output, channels, see Sec. 3.1.
2. We propose a new scaling technique for the batch normalization weights to mitigate a gradient instability at high channel pruning ratios, see Sec. 3.2.
3. We perform channel pruning subject to a cost constraint by encoding it as a resource allocation problem, which automatically allocates cost across the network instead of relying on manual or heuristic-based layer-wise pruning ratios. We show this allocation problem is a variant of the classic 0-1 knapsack problem, called the multiple-choice knapsack problem [38], which can be efficiently solved for our experiments, see Sec. 3.3.
4. We analyze our method's accuracy and cost improvements for the ImageNet and PASCAL VOC datasets for ResNet, MobileNet, and SSD architectures. We outperform prior pruning approaches, as shown in Fig. 1 and more extensively in Sec. 4. In particular, at high pruning ratios for ResNet50/ResNet101 on ImageNet, SMCP can achieve up to an additional 20% speedup at the same Top-1 accuracy level or up to a 0.6% Top-1 accuracy improvement at the same FPS (frames per second). SMCP can also prune an SSD512 with a ResNet50 backbone to achieve a speedup of 2.12×, exceeding the FPS (frames per second) of the smaller SSD300-ResNet50 model by 12%, while simultaneously improving on the mAP of the baseline model.

## 2 Related Work

### 2.1 Soft pruning

Most pruning methods start with a dense pretrained network and prune iteratively over a schedule to obtain a final network with the desired cost, where at each pruning step parameters are permanently zeroed (or masked). This effectively limits the model capacity as pruning occurs. Stosic and Stosic [39] argue that preserving the larger model capacity is critical to sparse model training by forming new paths for optimization that are not available for permanently pruned networks; they suggest it is important to allow gradient flow to previously pruned parameters and to rewire the sparsity occasionally.

Along these lines, several works have proposed soft pruning methods where parameters can be pruned and later unpruned if desirable. He et al. [14] zero weights during pruning but allows gradients to update them in an effort to

maintain model capacity. Dettmers and Zettlemoyer [6], Evci et al. [8], Mostafa and Wang [31], and Wortsman et al. [43] allow previously pruned weights to be regrown. Kusupati et al. [19] used a soft thresholding operator to achieve state-of-the-art results for unstructured and low-rank structured pruning. Kang and Han [18] introduces soft channel pruning by adding a differentiable mask in the batch normalization layers; however, their approach is limited to an implicit cost constraint on the total number of neurons. Our approach though is most similar to Guo et al. [10], Lin et al. [23], De Jorge et al. [17], and Zhou et al. [53], which explicitly or implicitly use the Straight-through Estimator (STE) [2] to adaptively prune parameters during training. The first three target unstructured sparsity, and the last targets N:M structured sparsity. In our work, we extend the use of the STE to channel pruning, show this requires pruning to be formulated along input channels, and embed this soft masking into a general-purpose, explicit cost-constrained formulation.

### 2.2 Cost-constrained and structured pruning

The goal of most pruning methods is to maximize network accuracy subject to low memory, computation, and/or latency requirements. Although unstructured sparsity approaches have proven to be very successfully in removing upwards of 95% of weights without affecting network accuracy [12], modern hardware has poor support for unstructured sparsity and therefore this rarely translates to actual speedup. Therefore, it is common to choose a pruning sparsity structure that can actually be accelerated in hardware, typically channel pruning for CNNs. There is now some hardware support for other sparsity structures, such as the N:M structured sparsity of [28], but we limit our focus to channel pruning in this work. Both Li et al. [21] and Yang et al. [45,46] select the best constraint-abiding network from a large number of candidate networks, which can be prohibitively expensive. Yu and Huang et al. [50], Tan et al. [41], and Wu et al. [44] pose cost-constrained optimization problems but use a greedy selection or cost-aware importance score to approximately select the best channels to prune. Chen et al. [3] presents a Bayesian optimization approach to determine compression hyperparameters that satisfy a cost constraint while maximizing network accuracy. Liu et al. [26] linked network pruning to Neural Architecture Search (NAS), arguing that the resulting pruned architectures are the novel contribution instead of the trained weights themselves. However, most NAS methods, such as those in [5,7,40], remain more computationally expensive than network pruning approaches. Our approach is most similar to the concurrent work of Shen et al. [37], called HALP, which also poses a cost-constrained resource allocation problem. There are however several major differences. First, we reduce our allocation problem to the multiple choice knapsack problem [38] and solve it with a meet-in-the-middle algorithm, which provides both optimality guarantees and efficient ($< 1$ second) solutions for general cost-constraints. HALP solves their allocation problem with a custom augmented knapsack solver, which gives no optimality guarantees and requires significant extra computation (1+ minute for each pruning step on ResNet50 [13], even after a large GPU-specific neuron

grouping step). Second, our method uses soft input channel masking as opposed to the permanent output channel pruning of HALP; we show this change alone yields performance gains in Sec. 4.3. Lastly, we use a new batch normalization scaling technique to stabilize training at high pruning ratios.

### 2.3 Pruning impact on batch normalization layers

Channel pruning can have a significant impact on the batch normalization statistics, which therefore strongly affects the network gradients to the remaining channels. This effect is particularly pronounced at high pruning ratios, since a large number of channels are being removed from most layers. Several pruning methods note this phenomenon and describe mitigation strategies. Li et al. [21] demonstrated the need to update the batch normalization statistics after pruning, as they can be significantly impacted, before evaluating possible pruned candidate networks. This approach does not however alleviate the issue of large gradients. Instead of immediately removing pruned weights and incurring the disruption, Wang et al. [42] slowly regularized them away, noticing significant performance gains particularly at high pruning ratios. They do not connect this to a sudden change in batch normalization statistics and gradients caused by pruning. They also use a non-gradient based importance so the impact on the importance of the remaining parameters is somewhat subdued. Since we are adaptively adjusting the sparsity and want to preserve the ability for pruned weights to become later unpruned, we do not want to regularize away pruned weights. We instead adopt a scaling technique on the batch normalization weights to stabilize training at high pruning ratios.

### 2.4 Parameter importance scoring

In order to decide which parameters of the network can be pruned while least harming network accuracy, most pruning methods define an importance for each parameter (or set of parameters) that approximates the effect of removal on the network's loss. Many importance scores have been proposed, largely falling into three groups: (i) based on weight magnitude [1,11,22,25,47,51]; (ii) based on a reconstruction-based objective [15,27]; and (iii) based on network gradients [20,29,30]. We adopt the Taylor first-order importance [29] due to its computational simplicity and its strong correlation with the true impact on the network's loss.

## 3 Soft masking for cost-constrained channel pruning

We propose a novel input channel pruning approach targeted towards high pruning ratios. Our method is initialized with a pretrained CNN model, and the desired network cost function and target cost constraint. We first re-parameterize the network weights with input channel masking variables, as shown in Sec. 3.1, to enable adaptive channel pruning. Then, after a warmup period, we iteratively

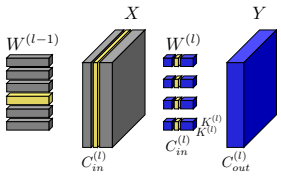


**Fig. 2.** Input channel pruning of a convolutional layer. Removing an input channel from weight $W^{(l)} \in \mathbb{R}^{C_{out}^{(l)} \times C_{in}^{(l)} \times K^{(l)} \times K^{(l)}}$ in layer $l$ removes the corresponding channel in the input feature map $X$ and the corresponding output channel in the previous weight $W^{(l-1)}$. The shape of the output feature map $Y$ is unaffected.

prune every $r$ minibatches by solving a resource allocation optimization problem, discussed in Sec. 3.3, to update the channel masks. After each mask update, we apply the batch normalization scaling described in Sec. 3.2, which stabilizes training at high pruning ratios. Finally, we fix the masks for a cooldown and fine-tuning period. We present the full algorithm and pseudocode in Sec. 3.4.

### 3.1 Soft input channel pruning

We specifically consider input channel pruning, as previously done in [15] and shown in Fig. 2, where we mask and later remove input channels to sparsify the CNN. As we will shortly show, channel pruning with a soft mask reparameterization requires it to be done along input channels, as this approach does not work when performing output channel pruning. This is a departure from the many output channel pruning approaches. From a global view of network sparsity, pruning one layer's input channel is equivalent to pruning the previous layer's output channel; however, the approaches are distinct when considering the effect on each individual layer.

For soft input channel masking, we consider a neural network with weights $W = \{W^{(l)}\}$, where $W^{(l)} \in \mathbb{R}^{C_{out}^{(l)} \times C_{in}^{(l)} \times K^{(l)} \times K^{(l)}}$ is the weight for layer $l$ of the network and has $C_{in}^{(l)}$ input channels and $C_{out}^{(l)}$ output channels. To allow input channels to be pruned and later unpruned, we introduce an input channel mask $m^{(l)} \in \{0,1\}^{C_{in}^{(l)}}$ for each layer $l$. Using these masks, we re-parameterize the weights so that the network's sparse weights are

$$\widetilde{W}^{(l)} = W^{(l)} \odot m^{(l)}. \tag{1}$$

where $m^{(l)}$ is broadcasted to match the shape of $W^{(l)}$. Instead of permanently zeroing a channel when pruning, the underlying network weights can be preserved and merely the masks set to zero. This has two distinct advantages. First, it helps preserve the full capacity of the original model while training towards a sparse model. Second, by allowing channels to be restored to their original values at a later time, poor early decisions on where to allocate the sparsity across the layers

can be undone. This is particularly important for high pruning ratios where a large portion of the network's channels must be removed.

As written though, our masking definition would define the gradient with respect to $W^{(l)}$ as $g_{W^{(l)}} = g_{\widetilde{W}^{(l)}} \odot m^{(l)}$. This masks the gradients as they flow back to the completely dense weights $W$, rendering masked weights unused in the forward pass and left untouched by the backward pass. Following the argument by Stosic and Stosic [39] that updating parameters not currently participating in the forward pass offers additional optimization paths that improve training of sparse networks, we adopt the Straight-through Estimator (STE) [2]. The STE has been successfully used in model quantization [35] and Ampere 2:4 structured pruning [53] for sparse parameter updates. The STE defines the gradient as

$$g_{W^{(l)}} = g_{\widetilde{W}^{(l)}}, \tag{2}$$

where gradients on the sparse weights pass straight through to the underlying, dense weights. Note that we still use the masks when computing the gradient with respect to the input feature map of the layer.

However, for this STE to have a useful impact in a modern CNN with the ubiquitious Conv-BN-ReLU pattern, it requires that channel pruning must be posed as input-oriented. Since $g_{\widetilde{W}^{(l)}}$ is defined by a matrix multiplication using the input feature map and the gradient of the output feature map, a masked input channel still receives non-zero gradients, except under a few edge cases. If we had instead masked output channels, the elements of $g_{W^{(l)}}$ would be either 0 or $\infty$, depending on the value of the batch normalization bias. Alternatively, if we instead tried to directly mask the batch normalization weight $\gamma^{(l)}$ and bias $\beta^{(l)}$ to emulate pruning the channel, we would get $g_{\gamma^{(l)}} = g_{\beta^{(l)}} = 0$ due to the ReLU. In either of these cases, the gradient $g_{W^{(l)}}$ is not useful.

Finally then, for input channel pruning with soft masking, we define the importance of each input channel, a proxy for the effect of removing this channel on the network's loss, according to the group first-order Taylor importance of [29]:

$$\mathcal{I}_i^{(l)} = \left| \sum_{o,r,s} W^{(l)}_{o,i,r,s} g_{W^{(l)}_{o,i,r,s}} \right| \tag{3}$$

where $\mathcal{I}_i^{(l)}$ is the importance of the $i$th input channel to layer $l$. Under certain conditions, this is in fact equivalent to the first-order batch normalization-based Taylor importance of [29], as shown in the supplementary materials.

### 3.2 Batch normalization scaling

When channel pruning at high ratios, there are many layers where a significant number of channels must be pruned. As a result of pruning these channels, either by zeroing them out or by applying masking, the subsequent gradient magnitudes to the remaining unpruned channels can be excessively large, which we show in the supplementary materials. We propose a batch normalization scaling technique that adjusts the batch normalization weight $\gamma^{(l)}$ of layer $l$ to mitigate

large gradients and stabilize the network sparsity and training. Specifically, we scale $\gamma^{(l)}$ according to the fraction of channels left unpruned by the current input channel mask $m^{(l)} \in \{0,1\}^{C_{in}^{(l)}}$

$$\gamma^{(l)} \leftarrow \gamma_{orig}^{(l)} \frac{\sum_i m_i^{(l)}}{C_{in}^{(l)}}. \tag{4}$$

In practice, we always treat $\gamma_{orig}^{(l)}$ as the parameter under optimization and vary a scaling variable $s^{(l)}$ to adjust the weight used by the network.

Moderating gradient magnitudes is particularly consequential since we employ the gradient-based importance score shown in Eq. (3). Even without soft masking and the STE, the large gradients cause importance accumulation in the remaining channels as pruning iteratively proceeds, artificially inhibiting additional channels in the layer from being pruned. When employing soft masking without this scaling technique, the large gradients cause large network sparsity thrashing. For example, if at one pruning iteration a large number of the channels are pruned, the importance to every channel, not only those left unpruned, is boosted by the resulting large gradient magnitudes. At the very next pruning iteration, those channels appear quite important and are restored to the network, causing other portions of the network to be pruned to still meet the cost constraint. This can oscillate, inhibiting network convergence and the final network accuracy. Moreover, for architectures in which pruning entire layers is possible, such as ResNet due to the skip connections, the infinite gradient magnitudes cause numerical overflow in updating the weights or even calculating the importance of channels. As shown in our experiments in Sec. 4, the proposed batch normalization scaling is crucial to overcome these training issues.

### 3.3 Cost-constrained channel pruning

At each pruning iteration, we seek to both minimize the impact on the network's loss as a result of pruning and sparsify the network towards the final cost constraint (e.g., latency constraints). We therefore formulate pruning as a cost-constrained importance maximization problem

$$\begin{aligned} \max_{m^{(2)},\ldots,m^{(L)}} \quad & \sum_{l=1}^{L} \sum_{i=1}^{C_{in}^{(l)}} \mathcal{I}_i^{(l)} m_i^{(l)} \\ \text{s.t.} \quad & \sum_{l=1}^{L} \mathcal{T}^{(l)} \left( \left\| m^{(l)} \right\|_1, \left\| m^{(l+1)} \right\|_1 \right) \leq \tau \\ & \left\| m^{(l)} \right\|_1 \in \mathcal{P}^{(l)}, \end{aligned} \tag{5}$$

where $L$ is the number of layers in the network, layer $l$ has $C_{in}^{(l)}$ input channels, $\mathcal{I}_i^{(l)}$ is the importance of input channel $i$ of layer $l$, $m^{(l)} \in \{0,1\}^{C_{in}^{(l)}}$ is the

input channel mask for layer $l$, $\mathcal{T}^{(l)}$ is the cost function for layer $l$, $\tau$ is the cost constraint, and $\mathcal{P}^{(l)}$ is the set of permitted values for the number of channels kept by mask $m^{(l)}$. By definition, $m_i^{(1)} = 1$ and $m_i^{(L+1)} = 1$ since those are the unprunable inputs and outputs of the network. A complete derivation of Eq. (5) can be found in the supplementary materials, as well as a discussion on how to handle skip connections in architectures like ResNet [13].

The final constraint, on the set of permitted values $\mathcal{P}^{(l)}$, is optional but useful in several situations. First, it can be used to disallow pruning the entire layer: by omitting 0 from $\mathcal{P}^{(l)}$ we prevent $m^{(l)} = 0$. As explained in the supplementary materials, layer pruning violates a key assumption of the derivation of Eq. (5). Second, it can be used to ensure the number of remaining channels is hardware-friendly, such as 8× multiples for GPU tensorcores [32] with $\mathcal{P}^{(l)} = \{0, 8, 16, \ldots, \lfloor C_{in}^{(l)}/8 \rfloor\}$.

We can further reduce this to an optimization over only the number of channels $p^{(l)}$, as the most important channels will always be kept in each layer:

$$\begin{aligned} \max_{p^{(2)},\ldots,p^{(L)}} \quad & \sum_{l=1}^{L} \sum_{i=1}^{p^{(l)}} \mathcal{I}_{(i)}^{(l)} \\ \text{s.t.} \quad & \sum_{l=1}^{L} \mathcal{T}^{(l)} \left( p^{(l)}, \overline{p^{(l+1)}} \right) \leq \tau \\ & p^{(l)} \in \mathcal{P}^{(l)} \end{aligned} \tag{6}$$

where $p^{(l)} = \left\| m^{(l)} \right\|_1$ and $\mathcal{I}_{(i)}^{(l)}$ is the $i$th largest value in $\mathcal{I}^{(l)}$. We also approximated the constraint using the current channel counts $\overline{p^{(l)}}$ to decouple the cost impact of masks in consecutive layers, which is required to pose this as an example of the following class of optimization problems.

**Multiple-choice knapsack problem** The optimization problem in Eq. (6) is an example of a generalization of the classic 0-1 knapsack problem called the multiple-choice knapsack (MCK) problem [38]. We show this connection explicitly in the supplementary materials. The MCK problem takes the form

$$\begin{aligned} \max_{x} \quad & \sum_{l=1}^{L} \sum_{i=1}^{n_l} v_{l,i} x_{l,i} \\ \text{s.t.} \quad & \sum_{l=1}^{L} \sum_{i=1}^{n_l} c_{l,i} x_{l,i} \leq C \\ & x_{l,i} \in \{0, 1\}, \quad \sum_{i=1}^{n_l} x_{l,i} = 1 \end{aligned} \tag{7}$$

where $L$ is the number of groups, group $l$ has size $n_l$, and the items have value $v_{l,i}$ and cost $c_{l,i} \geq 0$. The additional constraint relative to the classic 0-1 knapsack problem enforces that we select exactly one item from each group.

**Algorithm 1** Soft masking for cost-constrained channel pruning

**Inputs**: Pretrained network weights $W$, training set $\mathcal{D}$, total number of epochs $E$, pruning schedule $(r, K_w, K_t, K_c)$, target cost $\tau$

Initialize masks $m^{(l)} = 1$
Re-parameterize the weights (Eq. (1))
Train the network as usual for $K_w$ epochs
Calculate the pruning schedule $\{\tau_e\}$
**for** epoch $e \in [K_w, E - K_c)$ **do**
    **for** step $s$ in epoch $e$ **do**
        Perform the forward pass and backward pass, using Eqs. (1) and (2)
        Calculate and accumulate $\mathcal{I}_i^{(l)}$ (Eq. (3))
        **if** $s\%r = 0$ **then**
            Solve the optimization problem (Eq. (6)) using target cost $\tau_e$
            Update the masks $m^{(l)}$ accordingly
            Scale the BN weights $\gamma^{(l)}$ (Eq. (4))
            Reset the accumulated importance
        **end if**
    **end for**
**end for**
Train the network as usual for $K_c$ epochs
Apply the masks to the weights permanently
**return** Sparse network weights $W$

We solve Eq. (7) with a GPU-implemented meet-in-the-middle algorithm, presented in full in the supplementary materials. Our approach generalizes the standard meet-in-the-middle algorithm for the classic 0-1 knapsack problem, does not require integer costs, and very efficiently solves the MCK problem for our use cases. For example, for a ResNet50 [13], our approach solves the MCK problem in under 1 second. We present more complete timing details in the supplementary materials.

### 3.4 Overall method

We present our full method in Algorithm 1. We start with a pretrained network, layer-wise cost functions $\mathcal{T}^{(l)}$, and a global cost constraint $\tau$. We define our pruning schedule by: (i) $K_w$: the number of warmup epochs before starting pruning; (ii) $K_t$: the number of epochs after the warmup to reach the target cost $\tau$; (iii) $r$: the number of steps between recomputing the channel masks; and (iv) $K_c$: the number of cooldown epochs where the masks are kept fixed. During those $K_t$ epochs to reach the target cost, we define intermediate cost constraints $\{\tau_e\}$ using the exponential scheduler of [17]. Additionally, to stabilize the importance scores, which can be noisy due to the stochastic minibatches, we calculate and accumulate the importance score in Eq. (3) every minibatch between pruning iterations according to the exponential momentum approach of [29].

## 4 Results

We evaluate our method on both the ImageNet and PASCAL VOC benchmark datasets[3]. Full details on training settings and architectures can be found in the supplementary materials. We use a latency cost constraint, defined by a layer-wise lookup table (LUT) as previously described in [48,37,45]. We target and measure latency speed on a NVIDIA TITAN V GPU with cudNN V7.6.5 [4].

### 4.1 ImageNet results

We compare SMCP with several prior works on the ImageNet ILSVRC2012 [36] classification dataset. In Tab. 1, we compare the results of pruning ResNet50, ResNet101 [13], and MobileNet-V1 [16] at a number of pruning thresholds. We refer to SMCP-$X$% as retaining $X$% of the full model's original latency and calculate the frames per second (FPS) and speedup of the final network. For ResNet50, we show results for two different baseline models for a better comparison with prior works. The first baseline is from the PyTorch [34] model hub, with a Top-1 accuracy of 76.15%; the second baseline is the one used as a baseline for EagleEye [21] and has a Top-1 accuracy of 77.2%. We prune and fine-tune following the training setup of [33].

Our method performs comparably to prior works at low pruning ratios and outperforms them for large pruning ratios. For the PyTorch ResNet50 baseline model, we achieve a 0.3% higher Top-1 accuracy with a higher FPS at 2G and 1G FLOPs with an additional 0.04× and 0.19× speedup respectively. For the EagleEye [21] baseline, our method produces models near 1G FLOPs that have a 0.6% higher Top-1 accuracy for nearly the same FPS or a similar Top-1 accuracy while being 19% (or 0.5×) faster. The results are similar for ResNet101, which is based on the PyTorch model hub baseline model. At 2G FLOPs, we get a 0.3% higher Top-1 accuracy and an additional 0.03× speedup. On the already compact MobileNet-V1 model, where the desired pruning ratios are smaller, our method performs comparably to prior works; at the highest pruning ratio, we show a minor FPS improvement of 0.07× despite a higher FLOPs count, demonstrating the ability of the optimization problem in Sec. 3.3 to choose cost-constraint aware masks.

The benefits of our method, particularly at high pruning ratios, are possibly more easily seen when plotting the tradeoff curve for Top-1 accuracy versus FPS, as shown in Fig. 1 for the PyTorch baseline and Fig. 3 for the EagleEye baseline. For example in Fig. 3, at the 75% latency reduction level (or 3102 FPS), our method outperforms the nearest HALP [37] model with a 0.2% higher Top-1 accuracy and a 15% higher FPS; compared to EagleEye [21], we show a 0.23% higher Top-1 accuracy and a 26% higher FPS.

Moreover, our method can aggressively prune large, over-parameterized models to outperform smaller unpruned models. As shown in Tab. 1 and Fig. 3, a 50% pruned ResNet101 achieves a 1.6% Top-1 improvement over a baseline ResNet50,

[3] Our code can be accessed at https://github.com/NVlabs/SMCP.

**Table 1.** Pruning results on the ImageNet classification dataset considering two different ResNet50 baseline models as well as ResNet101 and MobileNetV1. We group results by those with similar FLOP counts, and refer to SMCP-$X$% as retaining $X$% of the full model's original latency. Results for prior works are as shown in [37].

| Method | FLOPs (G) | Top1 (%) | Top5 (%) | FPS (im/s) | Speedup |
|---|---|---|---|---|---|
| **ResNet50** | | | | | |
| No pruning | 4.1 | 76.2 | 92.87 | 1019 | 1× |
| ThiNet-70 [27] | 2.9 | 75.8 | 90.67 | - | - |
| AutoSlim [50] | 3.0 | 76.0 | - | 1215 | 1.14× |
| MetaPruning [26] | 3.0 | 76.2 | - | - | - |
| GReg-1 [42] | 2.7 | 76.3 | - | 1171 | 1.15× |
| HALP-80% [37] | 3.1 | **77.2** | **93.47** | 1256 | 1.23× |
| **SMCP-80% (Ours)** | 3.0 | 77.1 | 93.43 | **1292** | **1.27×** |
| 0.75× ResNet50 [13] | 2.3 | 74.8 | - | 1467 | 1.44× |
| ThiNet-50 [27] | 2.1 | 74.7 | 90.02 | - | - |
| AutoSlim [50] | 2.0 | 75.6 | - | 1592 | 1.56× |
| MetaPruning [26] | 2.0 | 75.4 | - | 1604 | 1.58× |
| GBN [49] | 2.4 | 76.2 | 92.83 | - | - |
| GReg-2 [42] | 1.8 | 75.4 | - | 1414 | 1.39× |
| HALP-55% [37] | 2.0 | 76.5 | 93.05 | 1630 | 1.60× |
| **SMCP-55% (Ours)** | 2.0 | **76.8** | **93.22** | **1673** | **1.64×** |
| 0.50× ResNet50 [13] | 1.1 | 72.0 | - | 2498 | 2.45× |
| ThiNet-30 [27] | 1.2 | 72.1 | 88.30 | - | - |
| AutoSlim [50] | 1.0 | 74.0 | - | 2390 | 2.45× |
| MetaPruning [26] | 1.0 | 73.4 | - | 2381 | 2.34× |
| GReg-2 [42] | 1.3 | 73.9 | - | 1514 | 1.49× |
| HALP-30% [37] | 1.0 | 74.3 | 91.81 | 2755 | 2.70× |
| **SMCP-30% (Ours)** | 1.0 | **74.6** | **92.00** | **2947** | **2.89×** |
| **ResNet50 - EagleEye [21] baseline** | | | | | |
| No pruning | 4.1 | 77.2 | 93.70 | 1019 | 1× |
| EagleEye-3G [21] | 3.0 | 77.1 | 93.37 | 1165 | 1.14× |
| HALP-80% [37] | 3.0 | 77.5 | 93.60 | 1203 | 1.18× |
| **SMCP-80% (Ours)** | 3.1 | **77.6** | **93.61** | **1263** | **1.23×** |
| EagleEye-2G [21] | 2.1 | 76.4 | 92.89 | 1471 | 1.44× |
| HALP-55% [37] | 2.1 | **76.6** | 93.16 | 1672 | **1.64×** |
| **SMCP-50% (Ours)** | 1.9 | **76.6** | **93.17** | **1706** | **1.67×** |
| EagleEye-1G [21] | 1.0 | 74.2 | 91.77 | 2429 | 2.38× |
| HALP-30% [37] | 1.2 | 74.5 | 91.87 | 2597 | 2.55× |
| **SMCP-30% (Ours)** | 1.1 | **75.1** | **92.29** | 2589 | 2.51× |
| **SMCP-25% (Ours)** | 0.9 | 74.4 | 91.98 | **3102** | **3.01×** |

| Method | FLOPs (G) | Top1 (%) | FPS (im/s) | Speedup |
|---|---|---|---|---|
| **ResNet101** | | | | |
| No pruning | 7.8 | 77.4 | 620 | 1× |
| Taylor-75% [29] | 4.7 | 77.4 | 750 | 1.21× |
| HALP-60% [37] | 4.3 | **78.3** | 847 | 1.37× |
| **SMCP-60% (Ours)** | 4.0 | 78.1 | 951 | 1.53× |
| HALP-50% [37] | **3.6** | 77.8 | 994 | 1.60× |
| **SMCP-50% (Ours)** | **3.6** | 77.8 | **1016** | **1.64×** |
| Taylor-55% [29] | 2.9 | 76.0 | 908 | 1.47× |
| HALP-40% [37] | 2.7 | 77.2 | 1180 | 1.90× |
| **SMCP-30% (Ours)** | 2.6 | **77.3** | 1273 | 2.05× |
| HALP-30% [37] | **2.0** | 76.5 | 1521 | 2.45× |
| **SMCP-25% (Ours)** | **2.0** | 76.8 | **1535** | **2.48×** |

| Method | FLOPs (M) | Top1 (%) | FPS (im/s) | Speedup |
|---|---|---|---|---|
| **MobileNet-V1** | | | | |
| No pruning | 569 | 72.6 | 3415 | 1× |
| 0.75× MobileNetV1 | 325 | 68.4 | 4678 | 1.37× |
| NetAdapt [45] | 284 | 69.1 | - | - |
| MetaPruning [26] | 316 | 70.9 | 4838 | 1.42× |
| EagleEye [21] | 284 | 70.9 | 5020 | 1.47× |
| HALP-60% [37] | 297 | **71.3** | 5754 | 1.68× |
| **SMCP-60% (Ours)** | 356 | 71.0 | **5870** | **1.72×** |
| MetaPruning [26] | 142 | 66.1 | 7050 | 2.06× |
| AutoSlim [50] | 150 | 67.9 | 7743 | 2.27× |
| HALP-42% [37] | 171 | **68.3** | 7940 | 2.32× |
| **SMCP-40% (Ours)** | 208 | **68.3** | **8163** | **2.39×** |

with no performance loss, and a 80% pruned ResNet50 achieves a similar Top-1 to an unpruned MobileNet-V1 while achieving a 10% FPS speedup.

Lastly, the accuracy and performance gains are in part due to the final network architecture chosen by our method. In particular, since we solve a global resource allocation problem during training, our method automatically determines the layer-wise pruning ratios for the given cost function and constraint. For example, on ResNet50, we find that SMCP is aggressive in pruning the early convolution layers and leaves the later layers better preserved; we provide additional analysis and figures in the supplementary materials.

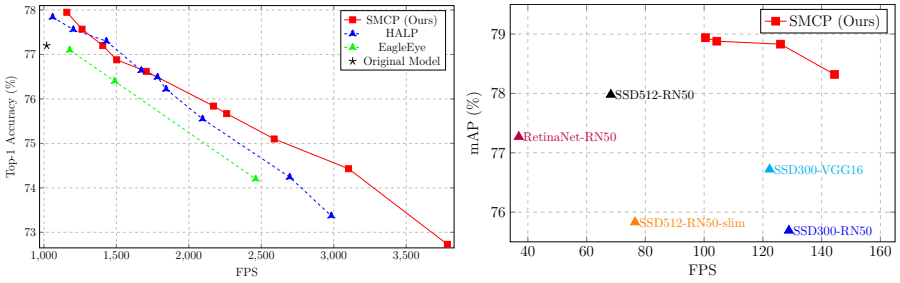


**Fig. 3.** (Left) Top-1 accuracy tradeoff curve for pruning ResNet50 on the ImageNet classification dataset using a latency cost constraint. Baseline model is from EagleEye [21]. (Right) mAP accuracy tradeoff curve for pruning SSD512-RN50 on the PASCAL VOC object detection dataset using a latency cost constraint. Top-right is better.

### 4.2 PASCAL VOC results

To analyze our method beyond image classification, we also analyze SMCP on the PASCAL VOC object detection dataset [9]. Specifically, we consider whether a large model, such as SSD512 [24] with a ResNet50 backbone, can be pruned at a high ratio to match the FPS of smaller models while retaining a superior mAP (mean average precision). We use the "07+12" train and test setup of [24] and prune both the backbone and feature layers.

As shown in Fig. 3, our method can prune an SSD512-RN50 to have a higher mAP than the pretrained model and a faster FPS than the much smaller SSD300-RN50 model, again showing the ability of our method to aggressively prune large over-parameterized models to outperform smaller models. In particular, our fastest pruned model has a 2.63 point higher mAP score while achieving 12% higher FPS. Critically, the latency reduction to achieve this is 75%, demonstrating the strength of our approach in the high pruning ratio regime. We also compare to and outperform a number of other common detector models.

### 4.3 Ablation study

We also study the effect of our contributions on the accuracy results shown above, specifically at high pruning ratios. We run our method again on the ImageNet classification dataset, starting from the ResNet50 EagleEye [21] baseline. We first remove the batch normalization scaling technique from Sec. 3.2 while keeping the soft input channel masking re-parameterization of Sec. 3.1. We then additionally remove the soft input channel masking, reverting to permanent pruning. We keep the solver and latency constraint in Sec. 3.3 unchanged. The ablation results are shown in Fig. 4. Removing the batch normalization scaling generally leads to marginally worse results, due to the training instability described in Sec. 3.2. Additionally removing the soft input masking, thereby using permanent channel pruning, degrades accuracy and performance further.

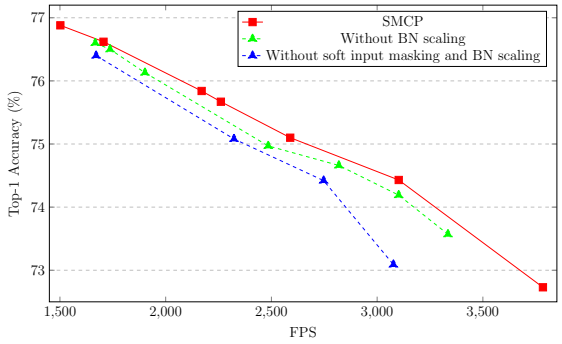


**Fig. 4.** Ablation study for SMCP at high pruning ratios on ResNet50 using the Eagle-Eye [21] baseline. We remove consecutively two major components of our method, soft input masking and batch normalization scaling, and observed worse Top-1 accuracy and FPS than the full SMCP method.

### 4.4 Choice of latency cost constraint

Although our cost-constrained formulation is general to any number of cost functions, the benefits of our approach are most pronounced under challenging, non-linear latency cost landscapes (i.e., latency cliffs for GPUs). Linear constraints (i.e., parameter/FLOP constraints) lessen the need for soft masking and an efficient and global resource allocation: removed channels are more likely to stay pruned once removed and the number of remaining channels in each layer tends to change slowly. Despite training against a latency constraint, Tab. 1 shows that SMCP is comparable to or even outperforms previous methods under low FLOP constraints.

## 5 Conclusion

By applying channel pruning, modern CNNs can be significantly accelerated, with a smaller memory footprint, computational cost, and inference time. In this work, we presented a novel structured input channel pruning approach, called SMCP, that combines soft masking of input channels, a batch normalization scaling technique, and the solution to a resource allocation problem to outperform prior works. We motivate the use of each component of our method and demonstrate their effectiveness on both the ImageNet and PASCAL VOC datasets. Although we only consider channel pruning in this work, our approach can be extended to jointly consider both channel and N:M structured pruning [28] to satisfy an explicit cost-constraint. This can be viewed as an extension of both this work and that of [53] and is left for a future work.

## References


1. Alvarez, J.M., Salzmann, M.: Learning the number of neurons in deep networks. In: NeurIPS. pp. 2262–2270 (2016) 2, 5
2. Bengio, Y., Léonard, N., Courville, A.C.: Estimating or propagating gradients through stochastic neurons for conditional computation. CoRR **abs/1308.3432** (2013) 4, 7
3. Chen, C., Tung, F., Vedula, N., Mori, G.: Constraint-aware deep neural network compression. In: ECCV. pp. 409–424 (2018) 4
4. Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., Shelhamer, E.: cudnn: Efficient primitives for deep learning. CoRR **abs/1410.0759** (2014) 11
5. Dai, X., Zhang, P., Wu, B., Yin, H., Sun, F., Wang, Y., Dukhan, M., Hu, Y., Wu, Y., Jia, Y., Vajda, P., Uyttendaele, M., Jha, N.K.: Chamnet: Towards efficient network design through platform-aware model adaptation. In: CVPR. pp. 11398–11407 (2019) 4
6. Dettmers, T., Zettlemoyer, L.: Sparse networks from scratch: Faster training without losing performance. CoRR **abs/1907.04840** (2019) 4
7. Dong, J., Cheng, A., Juan, D., Wei, W., Sun, M.: Dpp-net: Device-aware progressive search for pareto-optimal neural architectures. In: ECCV. pp. 540–555 (2018) 4
8. Evci, U., Gale, T., Menick, J., Castro, P.S., Elsen, E.: Rigging the lottery: Making all tickets winners. In: ICML. pp. 2943–2952 (2020) 4
9. Everingham, M., Gool, L.V., Williams, C.K.I., Winn, J.M., Zisserman, A.: The pascal visual object classes (VOC) challenge. Int. J. Comput. Vis. **88**(2), 303–338 (2010) 13
10. Guo, Y., Yao, A., Chen, Y.: Dynamic network surgery for efficient dnns. In: Lee, D.D., Sugiyama, M., von Luxburg, U., Guyon, I., Garnett, R. (eds.) NeurIPS. pp. 1379–1387 (2016) 4
11. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In: ICLR (2016) 5
12. Han, S., Pool, J., Tran, J., Dally, W.J.: Learning both weights and connections for efficient neural network. In: NeurIPS. pp. 1135–1143 (2015) 4
13. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR. pp. 770–778 (2016) 4, 9, 10, 11, 12
14. He, Y., Kang, G., Dong, X., Fu, Y., Yang, Y.: Soft filter pruning for accelerating deep convolutional neural networks. In: IJCAI. pp. 2234–2240 (2018) 3
15. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: ICCV. pp. 1398–1406 (2017) 2, 5, 6
16. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. CoRR **abs/1704.04861** (2017) 11
17. de Jorge, P., Sanyal, A., Behl, H.S., Torr, P.H.S., Rogez, G., Dokania, P.K.: Progressive skeletonization: Trimming more fat from a network at initialization. In: ICLR (2021) 4, 10
18. Kang, M., Han, B.: Operation-aware soft channel pruning using differentiable masks. In: ICML. pp. 5122–5131 (2020) 4
19. Kusupati, A., Ramanujan, V., Somani, R., Wortsman, M., Jain, P., Kakade, S.M., Farhadi, A.: Soft threshold weight reparameterization for learnable sparsity. In: ICML. pp. 5544–5555 (2020) 4

20. LeCun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: NeurIPS. pp. 598–605 (1989) 2, 5
21. Li, B., Wu, B., Su, J., Wang, G.: Eagleeye: Fast sub-net evaluation for efficient neural network pruning. In: ECCV. pp. 639–654 (2020) 2, 4, 5, 11, 12, 13, 14
22. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. In: ICLR (2017) 5
23. Lin, T., Stich, S.U., Barba, L., Dmitriev, D., Jaggi, M.: Dynamic model pruning with feedback. In: ICLR (2020) 4
24. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S.E., Fu, C., Berg, A.C.: SSD: single shot multibox detector. In: ECCV. vol. 9905, pp. 21–37 (2016) 13
25. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C.: Learning efficient convolutional networks through network slimming. In: ICCV. pp. 2755–2763 (2017) 5
26. Liu, Z., Sun, M., Zhou, T., Huang, G., Darrell, T.: Rethinking the value of network pruning. In: ICLR (2019) 4, 12
27. Luo, J., Wu, J., Lin, W.: Thinet: A filter level pruning method for deep neural network compression. In: ICCV. pp. 5068–5076 (2017) 2, 5, 12
28. Mishra, A.K., Latorre, J.A., Pool, J., Stosic, D., Stosic, D., Venkatesh, G., Yu, C., Micikevicius, P.: Accelerating sparse deep neural networks. CoRR **abs/2104.08378** (2021) 4, 14
29. Molchanov, P., Mallya, A., Tyree, S., Frosio, I., Kautz, J.: Importance estimation for neural network pruning. In: CVPR. pp. 11264–11272 (2019) 2, 5, 7, 10, 12
30. Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J.: Pruning convolutional neural networks for resource efficient inference. In: ICLR (2017) 5
31. Mostafa, H., Wang, X.: Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In: ICML. pp. 4646–4655 (2019) 4
32. NVIDIA Deep Learning Performance Guide: Convolutional Layers User Guide. https://docs.nvidia.com/deeplearning/performance/dl-performance-convolutional/index.html, accessed: 2021-11-15 9
33. NVIDIA Deep Learning Examples: ResNet50 v1.5 For Pytorch. https://github.com/NVIDIA/DeepLearningExamples/blob/master/PyTorch/Classification/ConvNets/resnet50v1.5/README.md, accessed: 2021-11-15 11
34. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E.Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) NeurIPS 2019. pp. 8024–8035 (2019) 2, 11
35. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: Xnor-net: Imagenet classification using binary convolutional neural networks. In: ECCV. pp. 525–542 (2016) 7
36. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M.S., Berg, A.C., Fei-Fei, L.: Imagenet large scale visual recognition challenge. Int. J. Comput. Vis. **115**(3), 211–252 (2015) 11
37. Shen, M., Yin, H., Molchanov, P., Mao, L., Liu, J., Alvarez, J.M.: HALP: hardware-aware latency pruning. CoRR **abs/2110.10811** (2021) 2, 4, 11, 12
38. Sinha, P., Zoltners, A.A.: The multiple-choice knapsack problem. Oper. Res. **27**(3), 503–515 (1979) 3, 4, 9

39. Stosic, D., Stosic, D.: Search spaces for neural model training. CoRR **abs/2105.12920** (2021) 3, 7
40. Su, X., You, S., Wang, F., Qian, C., Zhang, C., Xu, C.: Bcnet: Searching for network width with bilaterally coupled network. In: CVPR. pp. 2175–2184 (2021) 4
41. Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V.: Mnasnet: Platform-aware neural architecture search for mobile. In: CVPR. pp. 2820–2828 (2019) 4
42. Wang, H., Qin, C., Zhang, Y., Fu, Y.: Neural pruning via growing regularization. In: ICLR (2021) 2, 5, 12
43. Wortsman, M., Farhadi, A., Rastegari, M.: Discovering neural wirings. In: NeurIPS. pp. 2680–2690 (2019) 4
44. Wu, Y., Liu, C., Chen, B., Chien, S.: Constraint-aware importance estimation for global filter pruning under multiple resource constraints. In: CVPR. pp. 2935–2943 (2020) 4
45. Yang, T., Howard, A.G., Chen, B., Zhang, X., Go, A., Sandler, M., Sze, V., Adam, H.: Netadapt: Platform-aware neural network adaptation for mobile applications. In: ECCV. pp. 289–304 (2018) 4, 11, 12
46. Yang, T., Liao, Y., Sze, V.: Netadaptv2: Efficient neural architecture search with fast super-network training and architecture optimization. In: CVPR. pp. 2402–2411. Computer Vision Foundation / IEEE (2021) 4
47. Ye, J., Lu, X., Lin, Z., Wang, J.Z.: Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In: ICLR (2018) 2, 5
48. Yin, H., Molchanov, P., Alvarez, J.M., Li, Z., Mallya, A., Hoiem, D., Jha, N.K., Kautz, J.: Dreaming to distill: Data-free knowledge transfer via deepinversion. In: CVPR. pp. 8712–8721 (2020) 11
49. You, Z., Yan, K., Ye, J., Ma, M., Wang, P.: Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In: NeurIPS. pp. 2130–2141 (2019) 12
50. Yu, J., Huang, T.S.: Network slimming by slimmable networks: Towards one-shot architecture search for channel numbers. CoRR **abs/1903.11728** (2019) 2, 4, 12
51. Yu, R., Li, A., Chen, C., Lai, J., Morariu, V.I., Han, X., Gao, M., Lin, C., Davis, L.S.: NISP: pruning networks using neuron importance score propagation. In: CVPR. pp. 9194–9203 (2018) 2, 5
52. Zhang, C., Bengio, S., Hardt, M., Recht, B., Vinyals, O.: Understanding deep learning requires rethinking generalization. In: ICLR (2017) 1
53. Zhou, A., Ma, Y., Zhu, J., Liu, J., Zhang, Z., Yuan, K., Sun, W., Li, H.: Learning N: M fine-grained structured sparse neural networks from scratch. In: ICLR (2021) 4, 7, 14