Supplemental: 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

1 Experimental settings

We define our networks and pruning operations using PyTorch [8] and run our experiments across 8 NVIDIA Tesla V100 GPUs using automatic mixed precision and DDP (distributed data parallel) training.

1.1 ImageNet experiments

For all of the ImageNet experiments, across ResNet50, ResNet101, and MobileNet-v1, we follow NVIDIA's ResNet50 training setup [7] with a batch size of 256 per GPU, a linear learning rate warmup period of 8 epochs, a cosine decay learning rate schedule [5], and a 90 epoch training time.

We start pruning after $K_w = 10$ epochs, reach the target $K_t = 30$ epochs later at epoch 40, and allow the masks to continue to refine until fixing the masks for the final $K_c = 45$ epochs. We recompute the masks every $r = 80$ steps. (We conduct a small sensitivity study on these hyperparameters later in the supplementary materials). To ensure a GPU-friendly setting of the masks, we set $\mathcal{P}^{(l)} = \{i : i\%8 = 0, i \leq C_{in}^{(l)}\}$ for every layer but the first two. We prohibit any pruning of the first convolution by setting $\mathcal{P}^{(1)} = \{3\}, \mathcal{P}^{(2)} = \{C_{in}^{(2)}\}$ and prohibit any layer pruning by ensuring $0 \notin \mathcal{P}^{(l)}$.

We build the lookup table for ResNet50 and ResNet101 with a batch size of 256 and MobileNet-V1 with a batch size of 512; we assess the latency of the final pruned networks under these settings as well.

1.2 PASCAL VOC experiments

We use the SSD512 model described in [4], swapping the VGG16 backbone for a ResNet50 backbone. As in [2], we keep only the first three stages of the convolutions and change the strides in the third to stage to 1×1. We then add 6 pairs of feature detection layers as in [4] and the localization and confidence

* Work performed during a NVIDIA internship

heads to generate the boxes and their scores. We train for 800 epochs with a batch size of 16 per GPU, using PyTorch's *SyncBatchNorm* to synchronize the batch normalization statistics. We use a learning rate of 8e-3 for the total batch size 128, a linear warmup to that rate over 50 epochs, and reduce the rate by a multiple of $3/8, 1/3, 2/5, 1/10$ at $600, 700, 740, 770$ epochs respectively. For network biases, we double the learning rate. We also set the weight decay to 2e-3 except for the batch normalization parameters. We use the SGD optimizer with a momentum of 0.9

We start pruning after $K_w = 60$ epochs, reach the target $K_t = 250$ epochs later at epoch 310, and allow the masks to continue to refine until fixing the masks for the final $K_c = 350$ epochs. We recompute the masks every $r = 100$ steps. We use the same rules for $\mathcal{P}^{(l)}$ as with the ImageNet experiments.

We assess the latency of the final pruned SSD512-ResNet50 at a batch size of 1 for comparison with other detectors.

2 ResNet50 layer-wise pruning ratios

Since we solve a global resource allocation problem, there are no preset layer-wise pruning ratios. We therefore analyze the final pruning ratios for each layer to derive insights into our method. Fig. 1 plots the fraction of channels remaining in each layer relative to the original unpruned model. Generally, we find that SMCP prunes heavily in the early layers of the network and preserves more channels in the later layers of the network and in the second convolution layer in each residual block throughout (i.e., the conv2 layers).

We also compare our pruning ratios to the EagleEye [3] models of comparable FLOPs in Fig. 2. The general pattern of pruning heavily early and lighter later still holds. In particular, at the high pruning ratios, SMCP is able to keep a much larger number of channels in the later layers of the network, often twice as many as the EagleEye model, which seems to convey a Top-1 accuracy improvement as shown in Tab. 1 in the main paper.

3 Pruning schedule hyperparameters

Our algorithm only introduces new additional hyperparameters for the pruning schedule: the number of steps between recomputing the masks r and the target schedule defined by (K_w, K_t, K_c). To determine if our results are sensitive to these choices, we train a ResNet50 model at a 70% reduction while varying r for a fixed schedule and varying the schedule for fixed r; the results are shown in Fig. 3. We find that varying the target schedule, from the original schedule of $(10, 30, 45)$ to $(10, 35, 45), (10, 40, 35), (5, 20, 60), (5, 30, 50)$, has little impact on the final accuracy and FPS, with Top-1 by at most 0.07% with a FPS difference at most 46 FPS. Changing r has a much bigger impact. To understand why, recall that in order to define the knapsack-like optimization problem in Eq. (6) in the main paper we had to make an approximation $\mathcal{T}^{(l)}\left(p^{(l)}, p^{(l-1)}\right) \approx$

$\mathcal{T}^{(l)}\left(p^{(l)}, \overline{p^{(l-1)}}\right)$. Therefore solving the optimization problem less often results in a worse approximation of the loss landscape and a final model cost that can vary more from the desired cost. Nonetheless the resulting pruned model is quite accurate for its cost and lives on a Pareto frontier of possible models of different costs.

4 Equivalence of Taylor FO importance scores

Theorem 1. *For a network composed of Conv-BN-ReLU blocks and without a mask re-parameterization, the first-order Taylor importance on the batch normalization weight and bias (Taylor-FO-BN [6]) of layer l is equivalent to a first-order Taylor importance on the weights of the downstream input channel in layer $l+1$.*

$$\gamma_i^{(l)} g_{\gamma_i^{(l)}} + \beta_i^{(l)} g_{\beta_i^{(l)}} = \sum_{o,r,s} W_{o,i,r,s}^{(l+1)} g_{W_{o,i,r,s}^{(l+1)}} \tag{1}$$

Proof. Let the output BN layer be parameterized by γ and β and the following convolution layer be parameterized by W. Suppose y is the input to the BN layer, z is the output of the BN layer and $x = ReLU(y)$ is the input to the convolution layer. From the definition of a batch normalization layer, it follows that the gradient of the batch normalization weight and bias are given by

$$g_{\gamma_i} = \sum_{m,n} g_{z_{i,m,n}} \hat{y}_{i,m,n} \tag{2}$$

$$g_{\beta_i} = \sum_{m,n} g_{z_{i,m,n}} \tag{3}$$

where $\hat{y}$ is the value of y after normalization. Then,

$$\gamma_i g_{\gamma_i} + \beta_i g_{\beta_i} = \sum_{m,n} \left(\gamma_i \hat{y}_{i,m,n} + \beta_i\right) g_{z_{i,m,n}} \tag{4}$$

$$= \sum_{m,n} z_{i,m,n} g_{z_{i,m,n}} \tag{5}$$

$$= \sum_{m,n} x_{i,m,n} g_{x_{i,m,n}} \tag{6}$$

$$= \sum_{o,r,s} W_{o,i,r,s} g_{W_{o,i,r,s}} \tag{7}$$

where the last step proceeds from the definition of the convolution.

Corollary 1. *This holds even in the presence of skip connections in architectures like ResNet.*

Proof. Suppose for example that there are k BN layers whose outputs $z^{(k)}$ are added to create z, applied through a ReLU nonlinearity to create x, and then

distributed as $x^{(j)} = x$ to j different convolution layers with weights $W^{(j)}$. Using results from the proof of Theorem 1, we have

$$\sum_k \gamma_i^{(k)} g_{\gamma_i^{(k)}} + \beta_i^{(k)} g_{\beta_i^{(k)}} = \sum_{k,m,n} z_{i,m,n}^{(k)} g_{z_{i,m,n}^{(k)}} \tag{8}$$

$$= \sum_{k,m,n} z_{i,m,n}^{(k)} g_{z_{i,m,n}} \tag{9}$$

$$= \sum_{m,n} z_{i,m,n} g_{z_{i,m,n}} \tag{10}$$

$$= \sum_{m,n} x_{i,m,n} g_{x_{i,m,n}} \tag{11}$$

$$= \sum_{j,m,n} x_{i,m,n}^{(j)} g_{x_{i,m,n}^{(j)}} \tag{12}$$

$$= \sum_{j,o,r,s} W_{o,i,r,s}^{(j)} g_{W_{o,i,r,s}^{(j)}} \tag{13}$$

where we use gradient rules for both the addition and ReLU operations to simplify.

Corollary 2. *Theorem 1 also holds under hard masking (masking without the STE) and for unpruned channels under soft masking. For pruned channels under soft masking, $\gamma_i g_{\gamma_i} + \beta_i g_{\beta_i} = 0$ since it is the sparse weights that determine the gradient of input feature map but the dense weights receive a dense gradient update.*

5 Exploding gradients without batch normalization scaling

Let's consider the popular Conv-BN pattern, with weights W, γ, β and statistics $\mu^{(t)}, \sigma^{(t)}$ at step t. Suppose that a fraction $1 - \alpha$ of the input channels are pruned at the end of one training step. During the next training step, there are fewer non-zero weights, which can intuitively cause the batch variance to shrink: $\sigma^{(t+1)} < \sigma^{(t)}$. This does not affect the forward pass significantly, in that the output of the BN layer is still roughly $\mathcal{N}(\beta, \gamma^2)$, but it affects the gradients quite noticeably. In fact, this causes the gradients flowing to the remaining, unpruned channels to be boosted quite significantly as α nears 1.

Concretely, let the Conv-BN pair be characterized by $z = W * x, \hat{z} = \frac{z - \mu}{\sigma}, y = \gamma \hat{z} + \beta$. The gradient to the intermediate feature map z is equal to

$$g_{z_{o,h,w}} = \frac{\gamma_o}{\sigma_o} \left(g_{y_{o,h,w}} - \frac{1}{S} g_{\beta_o} - \frac{1}{S} \hat{z}_{o,h,w} g_{\gamma_o} \right) \tag{14}$$

where S is the size of each channel in the feature map y. When σ_o shrinks, the gradient magnitude is inversely increased, causing $g_{z_{o,h,w}}$ at time $t + 1$ to be

roughly a factor of $s_t = \sigma^{(t)}/\sigma^{(t+1)}$ larger than it was at time t. In the extreme case with $\alpha \to 1$, we can get exploding gradients, with a degeneracy at $\alpha = 1$ which corresponds to layer pruning.

6 Derivation of the cost-constrained optimization problem

We start from the idea cost-constrained network objective for neural network $f : X \to Y$

$$\arg\min_{W} \mathcal{L}\left(W, \mathcal{D}\right) \text{ s.t. } \mathcal{T}\left(f(W, x_i)\right) \leq \tau \tag{15}$$

where $\mathcal{L}$ is the network loss function, $W = \{W^{(l)}\}$ are the network's weights, $\mathcal{D} = \{(x_i, y_i)\}$ is the training set, $\mathcal{T}$ is the network's cost function, and τ is the cost constraint. Using the input channel masks $M = \{m^{(l)}\}$ and sparse weights $\widetilde{W}^{(l)} = W^{(l)} \odot m^{(l)}$, as described in Sec. 3.1 in the main paper, we get a joint optimization problem over the weights and masks

$$\arg\min_{W,M} \mathcal{L}\left(\widetilde{W}, \mathcal{D}\right) \text{ s.t. } \mathcal{T}(M) \leq \tau \tag{16}$$

where the cost constraint is now only a function of the masks. This is a combinatorially hard discrete optimization problem over the masks, so to make it tractable, we make several changes. First, we replace the loss minimization objective with an importance maximization objective and linearize it with a per-channel importance score $\mathcal{I}_i^{(l)}$ defined in Eq. (3) in the main paper. This assumes, that despite the nonlinearities of f, the importance score $\mathcal{I}_i^{(l)}$ is a good approximation of the effect of removing that channel from the network. This makes the objective linear in the masking variables:

$$\arg\max_{M} \sum_{l=1}^{L} \sum_{i=1}^{C_{in}^{(l)}} \mathcal{I}_i^{(l)} m_i^{(l)} \text{ s.t. } \mathcal{T}(M) \leq \tau \tag{17}$$

Second, we assume the cost function $\mathcal{T}$ is layer-wise separable into constituent cost functions $\mathcal{T}^{(l)}$ that depend only on the number of input and output channel masks for that layer. The output channel mask is defined by the input channel mask of the downstream layer. This yields

$$\begin{aligned} &\arg\max_{M} \sum_{l=1}^{L} \sum_{i=1}^{C_{in}^{(l)}} \mathcal{I}_i^{(l)} m_i^{(l)} \\ &\text{s.t.} \quad \sum_{l=1}^{L} \mathcal{T}^{(l)} \left(\left\| m^{(l)} \right\|_1, \left\| m^{(l+1)}) \right\|_1 \right) \leq \tau \end{aligned} \tag{18}$$

Lastly, we add the additional constraint on the allowable values for the number of input channels, $\left\| m^{(l)} \right\|_1 \in \mathcal{P}^{(l)}$, to get Eq. (5) in the main paper.

6.1 Skip connections

For architectures that have skip connections or other structural branching features, the optimization problem in Eq. (5) in the main paper needs one additional constraint. Specifically, all layers that share the same input channels must prune identically to one another. For example, in a ResNet bottleneck block that performs downsampling, both the downsample convolution and the first convolution in the branch share the same input channels. Therefore, their masks $m^{(down)}$ and $m^{(conv1)}$ must be equal to each. More formally, let g_k be a group of layers that share input channels. Then, we must add the constraint

$$m^{(l)} = m^{(g_k)} \quad \forall l \in g_k \tag{19}$$

for every group g_k of layers in the network. For $G = \{g_k\}$, the optimization problem reduces to choice of masks over each group instead of each layer

$$\begin{aligned} \arg\max_{M} & \sum_{k=1}^{|G|} \sum_{i=1}^{C_{in}^{(g_k)}} \left(\sum_{l \in g_k} \mathcal{I}_i^{(l)} \right) m_i^{(g_k)} \\ \text{s.t.} \quad & \sum_{k=1}^{|G|} \sum_{l \in g_k} \mathcal{T}^{(l)} \left(\left\| m^{(g_k)} \right\|_1, \left\| \overline{m^{(l+1)}} \right\|_1 \right) \leq \tau \\ & \left\| m^{(g_k)} \right\|_1 \in \bigcup_{l \in g_k} \mathcal{P}^{(l)} \\ & m^{(l)} = m^{(g_k)} \quad \forall l \in g_k \end{aligned} \tag{20}$$

where we decouple the cost impacts of masks in consecutive layers as in Eq. (6) in the main paper. This is a simply a generalization of Eq. (5) in the main paper where we consider groups of layers together instead of each layer individually.

7 Cost-constrained channel pruning as multiple-choice knapsack

Our cost-constrained resource allocation problem in Eq. (6) in the main paper is an example of a general class called the multiple-choice knapsack problem of [9]. We now show this connection explicitly. We start with Eq. (6) in the main paper, reproduced here for readability,

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

Now, we define

$$v_{l,j} = \sum_{i=1}^{j} \mathcal{I}_{(i)}^{(l)} \tag{21}$$

$$c_{l,j} = \mathcal{T}^{(l)}(j, \overline{p^{(l+1)}}) \tag{22}$$

$$x_{l,j} = \mathbf{1}\left(j = p^{(l)}\right) \tag{23}$$

where $\mathbf{1}(\cdot)$ is the indicator function. This yields the the equivalent optimization problem

$$\begin{aligned} \max_{x} \quad & \sum_{l=1}^{L} \sum_{j=1}^{n_l} v_{l,j} x_{l,j} \\ \text{s.t.} \quad & \sum_{l=1}^{L} \sum_{j=1}^{n_l} c_{l,j} x_{l,j} \leq \tau \\ & x_{l,j} \in \{0,1\}, \quad \sum_{j=1}^{n_l} x_{l,j} = 1 \\ & \sum_{j \in \mathcal{P}^{(l)}} x_{l,j} = 1 \end{aligned} \tag{24}$$

where $n_l = C_{in}^{(l)}$ is the maximum number of input channels for layer l. By trimming the problem to only the values $v_{l,j}$ and costs $c_{l,j}$ where $j \in \mathcal{P}^{(l)}$, we recover the general form of the multiple choice knapsack problem shown in Eq. (7) in the main paper.

8 Solving the multiple-choice knapsack problem

The standard method for solving the classic 0-1 knapsack problem is a dynamic programming algorithm. For the multiple-choice knapsack problem [9], Dudziński and Walukiewicz [1] define a similar dynamic programming solution. It requires integer costs $c_{l,i}, C \in \mathbb{Z}_{\geq 0}$ and has solution runtime complexity $O(nC)$ where $n = \sum_g n_l$ and space complexity $O(GC)$ (in order to recover the items used) when $c_{l,i}, C \in \mathbb{Z}_{\geq 0}$. When the costs are not integer or are incredibly large integers, this becomes intractable unless a scaling and rounding step is performed. Even still, for problems with very sparse values $v_{l,i}$ and costs $c_{l,i}$, the dynamic programming approach is inefficient. We instead solve the MCK problem using a GPU-implemented generalization of the meet-in-the-middle algorithm used for the classic 0-1 knapsack problem. The full algorithm is defined in Algorithm 1. The benefit of this approach is we only store feasible values and costs and aggressively reject suboptimal solutions with the condense step. However, the runtime and space complexities are asymptotically much worse in general, as shown in Theorem 2.

Theorem 2. *The meet-in-the-middle algorithm in Algorithm 1 has worst case runtime complexity* $O\left(LB^L \log(B)\right)$ *and space complexity* $O\left(B^L\right)$ *where* $B = \max_l n_l$ *and we assume* $B \gg L$.

Proof. We start by deriving the complexity of the merge function for a group of size M *and another of size* N.

$$T_{merge}(M,N) = O(MN) + T_{condense}(MN) \tag{25}$$
$$= O(MN\log(MN)) \tag{26}$$

since the condense function requires and is dominated by a sort of the values.

The runtime complexity of the full multiple-choice knapsack solver can be bounded by assuming every group takes the maximum size B. *Without loss of generality, we also assume* $L = 2^j$ *for some* j. *Then, for a multiple-choice knapsack (MCK) problem with* L *groups of size* B, *we have*

$$T_{mck}(L,B) = T_{mck}\left(\frac{L}{2}, B^2\right) + \frac{L}{2}T_{merge}(B,B) \tag{27}$$
$$\leq T_{mck}\left(1, B^{2^j}\right) \tag{28}$$
$$+ \sum_{i=1}^{j} \frac{L}{2^i} T_{merge}\left(B^{2^{i-1}}, B^{2^{i-1}}\right) \tag{29}$$
$$\leq O\left(B^{2^j}\right) \tag{30}$$
$$+ \sum_{i=1}^{j} \frac{L}{2^i} 2^i O\left(B^{2^i}\log(B)\right) \tag{31}$$
$$\leq O\left(LB^L \log(B)\right) \tag{32}$$

since the merge can create a new item for every combination of items in the two merging groups (squaring the size of the group), $T_{mck}(1,B) = O(B)$, *and the runtime is dominated by the final merge of groups.*

The space complexity proceeds similarly, except for the log *factor which is due to the sort in the condense step.*

Corollary 3. *The runtime and space complexity of Algorithm 1 can be improved to* $O\left(LB^{L/2}\log(B)\right)$ *and* $O\left(B^{L/2}\right)$ *respectively by replacing the final merge with a* $O\left(B^{L/2}\log\left(B^{L/2}\right)\right)$ *sort and sweep over the last two groups.*

Since the multiple-choice knapsack problem reverts to the original knapsack problem under $B = 2, v_{l,1} = 0, c_{l,1} = 0$, Algorithm 1 recovers the runtime and space complexities of the standard meet-in-the-middle knapsack solver, which are $O\left(L2^{L/2}\right)$ and $O\left(2^{L/2}\right)$ respectively, using the sort and sweep improvement of Corollary 3.

Algorithm 1 MCK meet-in-the-middle solver

Input: Number of groups L, group value vectors v_l, group cost vectors $c_l \leq C$, and capacity C

Output: Best value v_{best}, best cost c_{best}, and used items $k_l = i$ s.t. $x_{l,i} = 1$.

function MCK(v, c, C)
  $L \leftarrow \text{len}(v)$
  $k_l \leftarrow 0 \quad \forall l \in [L]$
  if $L == 1$ **then**
    $k_1 \leftarrow \arg\max_i v_{1,i}$
    return v_{1,k_1}, c_{1,k_1}, k
  end if
  for $l \in \text{range}(1, \lfloor L/2 \rfloor)$ **do**
    v_l, c_l, u_l, u_{L-l+1}
      $\leftarrow \text{MERGE}\left(v_l, c_l, v_{L-l+1}, c_{L-l+1}, C\right)$
  end for
  $v \leftarrow \{v_1, \ldots, v_{\lceil L/2 \rceil}\}$, $c \leftarrow \{c_1, \ldots, c_{\lceil L/2 \rceil}\}$
  $v_{best}, c_{best}, k_{rest} \leftarrow \text{MCK}(v, c, C)$
  for $l \in \text{range}(1, \lfloor L/2 \rfloor)$ **do**
    $i \leftarrow k_l$
    $k_l \leftarrow u_{l,i}$
    $k_{L-l+1} \leftarrow u_{L-l+1,i}$
  end for
  return v_{best}, c_{best}, k
end function
function MERGE(v_1, c_1, v_2, c_2, C)
  $v_{new,N(i-1)+j} \leftarrow v_{1,i} + v_{2,j} \quad \forall i \in [M], j \in [N]$
  $c_{new,N(i-1)+j} \leftarrow c_{1,i} + c_{2,j}$
  $v_{new}, c_{new}, u \leftarrow \text{CONDENSE}(v_{new}, c_{new}, C)$
  $u_1 \leftarrow \lfloor \frac{u}{N} \rfloor$
  $u_2 \leftarrow u \% N$
  return $v_{new}, c_{new}, u_1, u_2$
end function
function CONDENSE(v, c, C)
  $u \leftarrow \{i : v_i > v_j \, \forall j \neq i \text{ s.t. } c_j \leq c_i\}$
    $\cup \{i : v_i \geq v_j \, \forall j \neq i \text{ s.t. } c_j = c_i\}$
  $u \leftarrow u \cap \{i : c_i \leq C\}$
  return $\{v_i : i \in u\}, \{c_i : i \in u\}, u$
end function

Method	Accuracy	Solve time (s)
DP ($s = 10$)	1 decimal	< 1 second
DP ($s = 100$)	2 decimal	~ 6 second
DP ($s = 500$)	$2 - 3$ decimals	~ 30 second
DP ($s = 1000$)	3 decimals	~ 70 second
Algorithm 1	machine roundoff	< 1 second

Table 1. Representative time to solve Eq. (6) in the main paper under different settings. DP is our implementation of the algorithm in [1].

9 Multiple-choice knapsack solve effort

We present representative timings for the solve effort of the meet-in-the-middle multiple-choice knapsack solver in Algorithm 1. When pruning a ResNet50, there are 38 pruning groups (see Sec. 6.1 for more details). There are a total of $22,531$ input channels, including the 3 image channels of the first convolution layer. The largest layer has 2048 input channels. Restricting the possible masks to $8x$ for GPU tensorcores, we test the solution time for both the meet-in-the-middle solver in Algorithm 1 and an implementation of the dynamic programming (DP) approach [1]. We use a representative latency capacity of 255.4, in units of milliseconds, and randomly choose the current mask settings of the network. To use the DP approach, we define a scaling factor s and convert all costs to integers according to $\lfloor c_{l,i} s \rfloor$. For $s = 10^d$, the DP solution will be correct to d digits; Algorithm 1 is correct up to machine roundoff. The solution times are shown in Tab. 1.

10 Allowing layer pruning

In deriving our optimization problem in Eq. (5) in the main paper, as shown in Sec. 6, we had to assume that the importance and cost functions were layer-wise separable, meaning the input mask $m^{(l)}$ for layer l only affects layer l and the layer(s) immediately upstream. This assumption is obviously broken when we allow layer pruning, $m^{(l)} = 0$, as pruning the entire layer effectively prunes all layers in that branch of the network.

Nonetheless, we ran several experiments where we allowed layer pruning to occur, if so chosen by the knapsack solver, when pruning the ResNet50 from the EagleEye [3] baseline. The results are shown in Fig. 4. Breaking the theoretical layer-wise separability assumption yields poor experimental results.

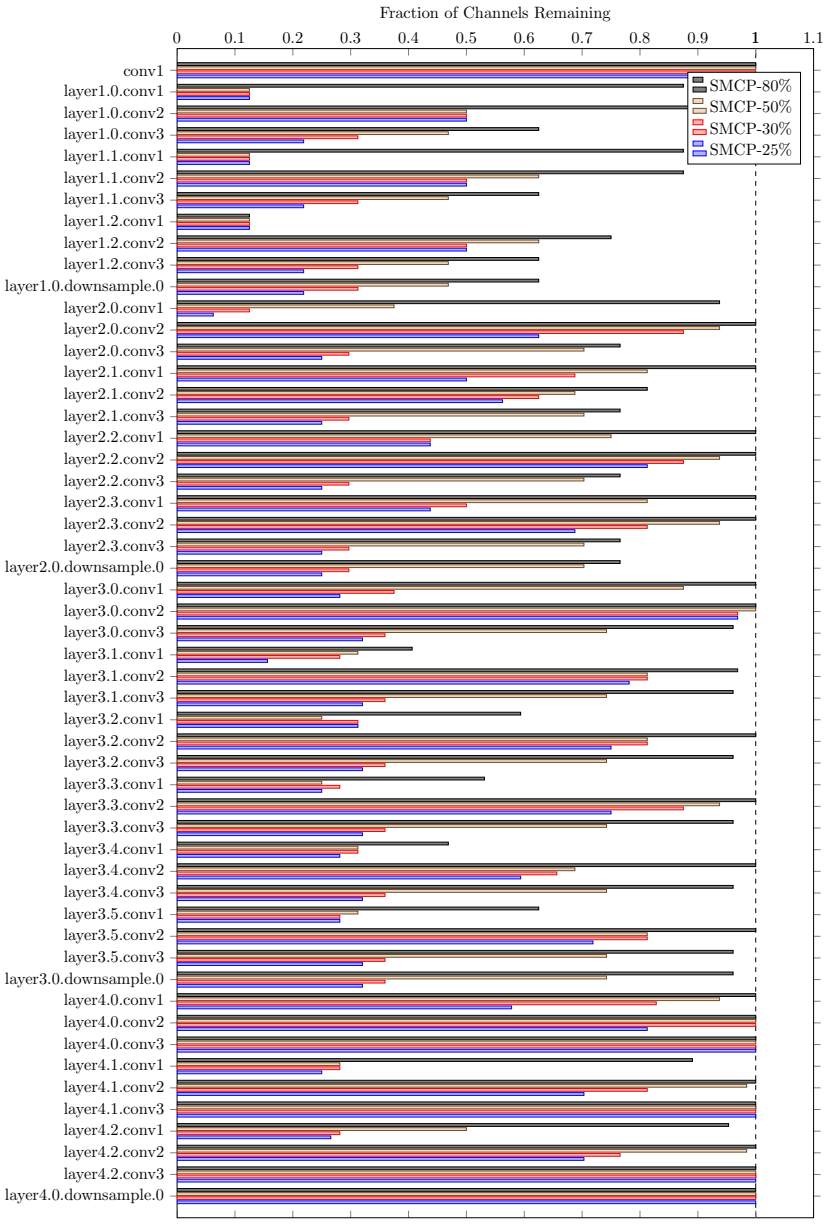

Fig. 1. Fraction of remaining channels per layer for SMCP models on the ImageNet classification dataset.

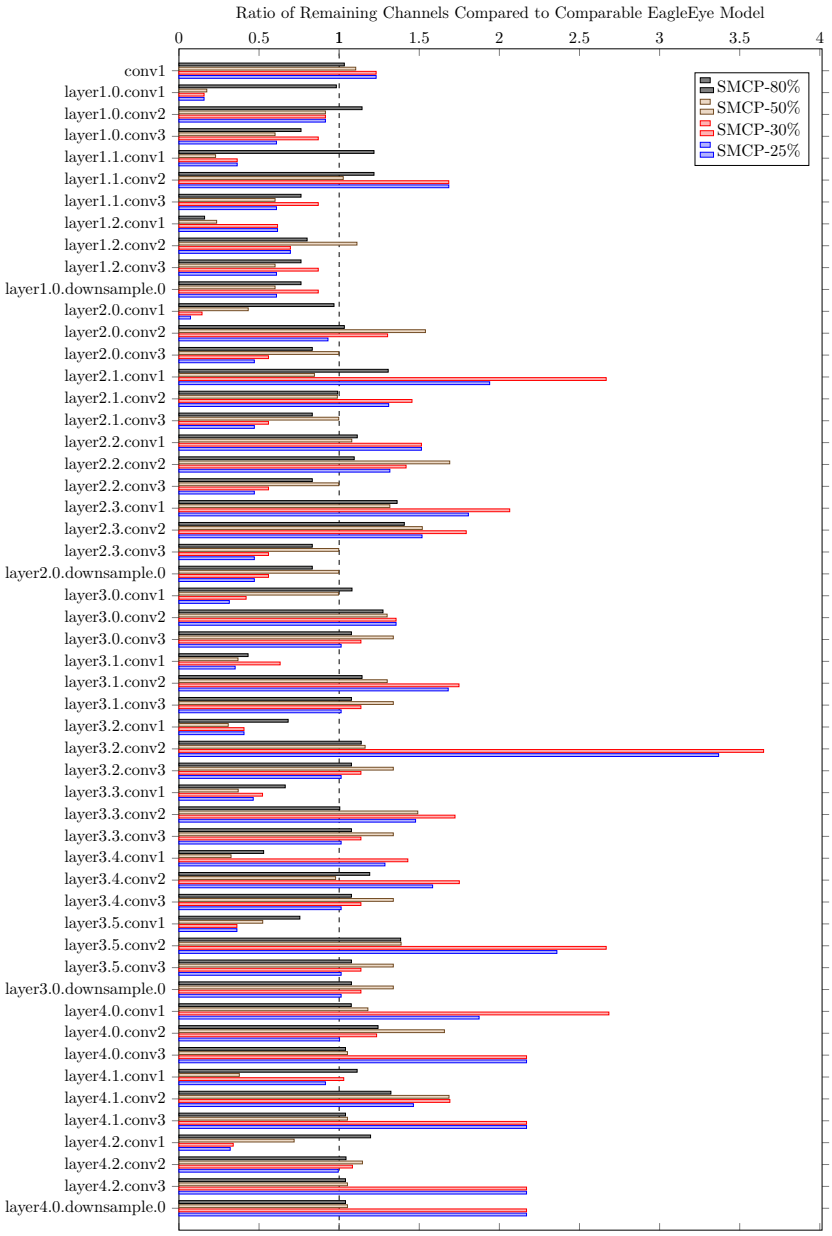

Fig. 2. Relative comparison of number of remaining channels per layer for SMCP and EagleEye ResNet50 models on the ImageNet classification dataset. Each SMCP model is compared to the EagleEye model of comparable FLOPs.

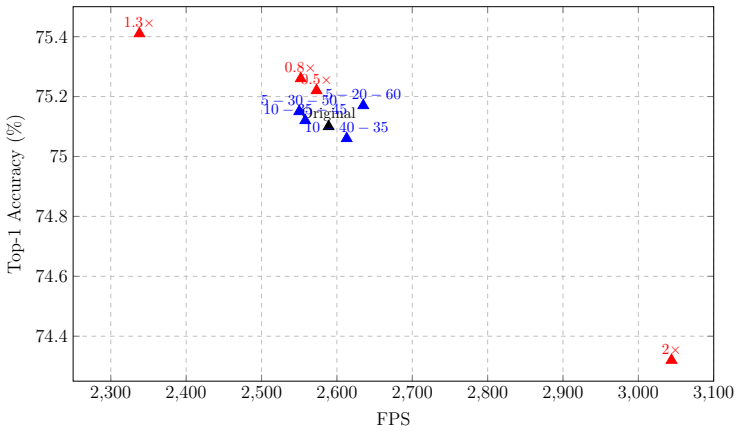

Fig. 3. Ablation study for SMCP's pruning schedule hyperparameters. Baseline model is EagleEye's ResNet50 model [3]. Multiple denotes changing the rewiring frequency by the given multiple. Triples denote changing the target schedule hyperparameters. FPS measured on an NVIDIA TITAN V GPU.

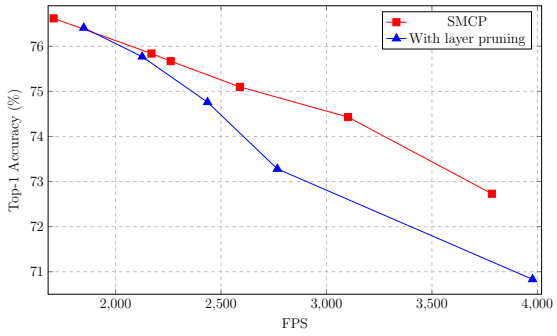

Fig. 4. Comparison of allowing versus disallowing layer pruning at high pruning ratios for ResNet50 on the ImageNet classification dataset using a latency cost constraint. Baseline model is from EagleEye [3]. Accuracy against FPS speed shows the disadvantage of allowing layer pruning. Top-right is better. FPS measured on an NVIDIA TITAN V GPU.

References

1. Dudziński, K., Walukiewicz, S.: Exact methods for the knapsack problem and its generalizations. European Journal of Operational Research **28**(1), 3–21 (1987) 7, 10
2. Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., Murphy, K.: Speed/accuracy trade-offs for modern convolutional object detectors. In: CVPR. pp. 3296–3297 (2017) 1
3. 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, 10, 13
4. 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) 1
5. Loshchilov, I., Hutter, F.: SGDR: stochastic gradient descent with warm restarts. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net (2017) 1
6. Molchanov, P., Mallya, A., Tyree, S., Frosio, I., Kautz, J.: Importance estimation for neural network pruning. In: CVPR. pp. 11264–11272 (2019) 3
7. 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 1
8. 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) 1
9. Sinha, P., Zoltners, A.A.: The multiple-choice knapsack problem. Oper. Res. **27**(3), 503–515 (1979) 6, 7