Structural Pruning via Latency-Saliency Knapsack

Maying Shen[*], **Hongxu Yin**[*], **Pavlo Molchanov, Lei Mao, Jianna Liu, Jose M. Alvarez**

NVIDIA

{mshen,dannyy,pmolchanov,lmao,jiannal,josea}@nvidia.com

Abstract

Structural pruning can simplify network architecture and improve inference speed. We propose Hardware-Aware Latency Pruning (HALP) that formulates structural pruning as a global resource allocation optimization problem, aiming at maximizing the accuracy while constraining latency under a predefined budget on targeting device. For filter importance ranking, HALP leverages latency lookup table to track latency reduction potential and global saliency score to gauge accuracy drop. Both metrics can be evaluated very efficiently during pruning, allowing us to reformulate global structural pruning under a reward maximization problem given target constraint. This makes the problem solvable via our augmented knapsack solver, enabling HALP to surpass prior work in pruning efficacy and accuracy-efficiency trade-off. We examine HALP on both classification and detection tasks, over varying networks, on ImageNet and VOC datasets, on different platforms. In particular, for ResNet-50/-101 pruning on ImageNet, HALP improves network throughput by $1.60\times$/$1.90\times$ with $+0.3\%$/-0.2% top-1 accuracy changes, respectively. For SSD pruning on VOC, HALP improves throughput by $1.94\times$ with only a 0.56 mAP drop. HALP consistently outperforms prior art, sometimes by large margins. Project page at https://halp-neurips.github.io/

1 Introduction

Convolutional Neural Networks (CNNs) act as the central tenet behind the rapid development in computer vision tasks such as classification, detection, segmentation, image synthesis, among others. As performance boosts, so do model size, computation, and latency. With millions, sometimes billions of parameters (*e.g.*, GPT-3 [4]), modern neural networks face increasing challenges upon ubiquitous deployment, that mostly faces stringent constraints such as energy and latency [9, 46, 45, 55]. In certain cases like autonomous driving, a breach of real-time constraint not only undermines user experience, but also imposes critical safety concerns. Even for cloud service, speeding up the inference directly translates into higher throughput, allowing more clients and users to benefit from the service.

One effective and efficient method to reduce model complexity is network pruning. The primary goal of pruning is to remove the parameters, along with their computation, that are deemed least important for inference [2, 20, 37, 45]. Compatible with other compression streams of work such as quantization [5, 64, 73], dynamic inference [31, 67, 70], and distillation [23, 47, 69], pruning enables a flexible tuning of model complexity towards varying constraints, while requiring much less design efforts by neural architecture search [58, 61, 65] and architecture re-designs [24, 42, 59]. Thus, in this work, we study pruning, in particular structured pruning that reduces channels to benefit off-the-shelf platforms, *e.g.*, GPUs.

[*]Equal contribution.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

https://doi.org/10.52202/068431-0937

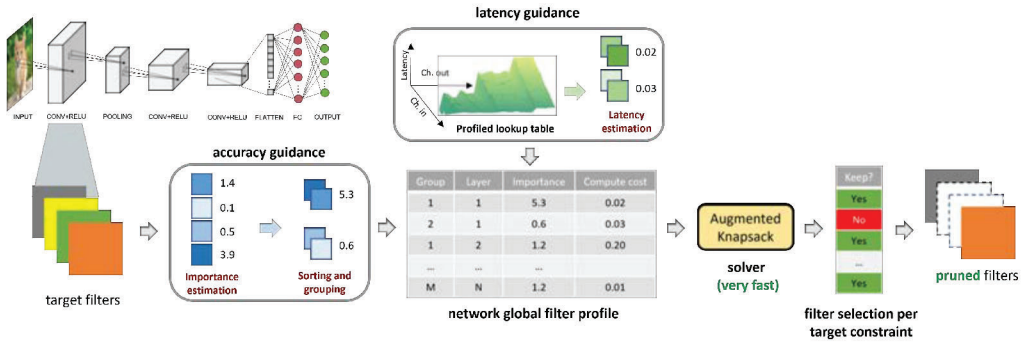


Figure 1: The proposed hardware-aware latency pruning (HALP) paradigm. Considering both performance and latency contributions, HALP formulates global structural pruning as a global resource allocation problem (Section 3.1), solvable using our augmented Knapsack algorithm (Section 3.2). Pruned architectures surpass prior work across varying latency constraints given changing network architectures for both classification and detection tasks (Section 4).

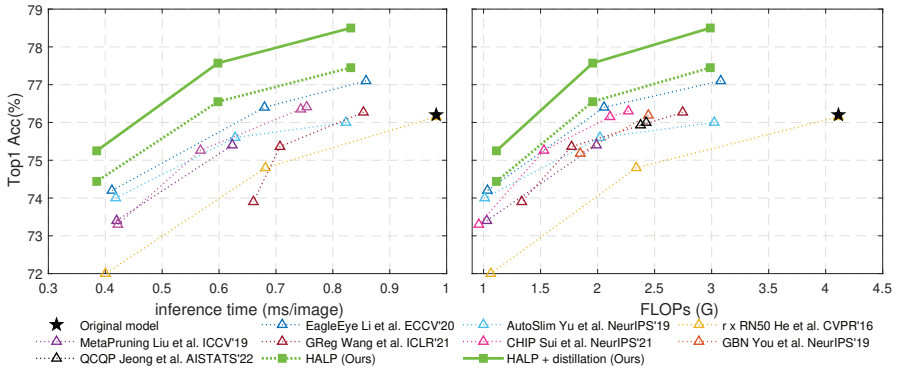


Figure 2: Pruning ResNet50 on the ImageNet dataset. The proposed HALP surpasses state-of-the-art structured pruning methods over accuracy, latency, and FLOPs metrics. Target hardware is NVIDIA Titan V GPU. **Top-left** is better.

As the pruning literature develops, the pruning criteria also evolve to better reflect final efficiency. The early phase of the field focuses on maximum parameter removal in seek for minimum representations of the pretrained model. This leads to a flourish of approaches that rank neurons effectively to measure their importance [46, 62]. As each neuron/filter possesses intrinsically different computation, following works explore proxy to enhance redundancy removal, FLOPs being one of the most widely adopted metrics [32, 66, 72] to reflect how many multiplication and addition computes needed for the model. However, for models with very similar FLOPs, their latency can vary significantly [58]. Recently, more and more works start directly working on reducing latency [6, 68, 55]. However, not much was done in the field of GPU friendly pruning methods due to non-trivial latency-architecture trade-off. For example, as recently observed in [51], GPU usually imposes staircase-shaped latency patterns for convolutional operators with varying channels, which inevitably occur per varying pruning rate, see the latency surface in Fig. 1. This imposes a constraint that pruning needs to be done in groups to achieve latency improvement. Moreover, getting the exact look-up table of layers under different pruning configurations will benefit maximizing performance while minimizing latency.

Pruning different layers in the deep neural network will result in different accuracy-latency trade-off. Typically, removing channels from the latter layers has smaller impact on accuracy and smaller impact on latency versus removing channels from the early layers. We ask the question, if it is better to remove *more* neurons from latter layer or *less* from early layer to achieve the same accuracy-latency trade-off. By nature, the problem is combinatorial and requires the appropriate solution.

In this paper, we propose hardware-aware latency pruning (HALP) that formulates pruning as a resource allocation optimization problem to maximize the accuracy while maintaining a latency budget on the targeting device. The overall workflow is shown in Fig. 1. For latency estimate per

https://doi.org/10.52202/068431-0937

pruned architecture, we pre-analyze the operator-level latency values by creating a look-up table for every layer of the model on the target hardware. Then we introduce an additional score for each neuron group to reflect and encourage latency reduction. To this end, we first rank the neurons according to their importance estimates, and then dynamically adjust their latency contributions. With neurons re-calibrated towards the hardware-aware latency curve, we now select remaining neurons to maximize the gradient-based importance estimates for accuracy, within the total latency constraint. This makes the entire neuron ranking solvable under the knapsack paradigm. To enforce the neuron selection order in a layer to be from the most important to the least, we have enhanced the knapsack solver so that the calculated latency contributions of the remaining neurons would hold. HALP surpasses prior art in pruning efficacy, see Fig. 2 and the more detailed analysis in Section 4. Our main contributions are summarized as follows:

- We propose a latency-driven structured pruning algorithm that exploits hardware latency traits to yield direct inference speedups.
- We orient the pruning process around a quick yet highly effective knapsack scheme that seeks for a combination of remaining neuron groups to maximize importance while constraining to the target latency.
- We introduce a group size adjustment scheme for knapsack solver amid varying latency contributions across layers, hence allowing full exploitation of the latency landscape of the underlying hardware.
- We compare to prior art when pruning ResNet, MobileNet, VGG architectures on ImageNet, CIFAR10, PASCAL VOC and demonstrate that our method yields consistent latency and accuracy improvements over state-of-the-art methods. Our ImageNet pruning results present a viable $1.6\times$ to $1.9\times$ speedup while preserving very similar original accuracy of the ResNets.

2 Related work

Pruning methods. Depending on when to perform pruning, current methods can generally be divided into three groups [14]: i) prune pretrained models [20, 40, 33, 22, 46, 45, 16], ii) prune at initialization [15, 30, 10], and iii) prune during training [2, 17, 41]. Despite notable progresses in the later two streams, pruning pretrained models remains as the most popular paradigm, with structural sparsity favored by off-the-shelf inference platforms such as GPU.

To improve inference efficiency, many early pruning methods trim down the neural network aiming to achieve a high channel pruning ratio while maintaining an acceptable accuracy. The estimation of neuron importance has been widely studied in literature [25, 40, 46]. For example, [45] proposes to use Taylor expansion to measure the importance of neurons and prunes a desired number of least-ranked neurons. However, a channel pruning ratio does not directly translate into computation reduction ratio, amid the fact that a neuron at different location leads to different computations.

There are recent methods that focus primarily on reducing FLOPs. Some of them take FLOPs into consideration when calculating the neuron importance to encourage penalizing neurons that induce high computations [66]. An alternative line of work propose to select the best pruned network from a set of candidates [32, 68]. However, it would take a long time for candidate selection due to the large amount of candidates. In addition, these methods use FLOPs as a proxy of latency, which is usually inaccurate as networks with similar FLOPs might have significantly different latencies [58].

Latency-aware compression. Emerging compression techniques shift attention to directly cut down on latency. One popular stream is Neural Architecture Search (NAS) methods [9, 12, 58, 65] that adaptively adjusts the architecture of the network for a given latency requirement. They incorporate the platform constraints into the optimization process in both the architecture and parameter space to jointly optimize the model size and accuracy. Despite remarkable insights, NAS methods remain computationally expensive in general compared to their pruning counterparts.

Latency-oriented pruning has also gained a growing amount of attention. [6] presents a framework for network compression under operational constraints, using Bayesian optimization to iteratively obtain compression hyperparameters that satisfy the constraints. Along the same line, NetAdapt [68] iteratively prunes neurons across layers under the guidance of the empirical latency measurements on the targeting platform. While these methods push the frontier of latency constrained pruning, the hardware-incurred latency surface in fact offers much more potential under our enhanced pruning policy - as we show later, large rooms for improvements remain unexploited and realizable.

https://doi.org/10.52202/068431-0937

3 Method

In this section, we first formulate the pruning process as an optimization process, before diving deep into the importance estimation for accuracy and latency. Then, we elaborate on how to solve the optimization via knapsack regime, augmented by dynamic grouping of neurons. We finalize the method by combining these key steps under one realm of HALP.

3.1 Objective Function

Consider a neural network that consists of L layers performing linear operations on their inputs, together with non-linear activation layers and potentially pooling layers. Suppose there are N_l neurons (output channels) in the l_{th} layer and each neuron is encoded by parameters $\mathbf{W}_l^n \in \mathbb{R}^{C_l^{in} \times K_l \times K_l}$, where C^{in} is the number of input channels and K is the kernel size. By putting all the neurons across the network together, we get the neuron parameter set $\mathbf{W} = \{\{\mathbf{W}_l^n\}_{n=1}^{N_l}\}_{l=1}^{L}$, where $N = \sum_{l=1}^{L} N_l$ is the total number of neurons in the network.

Given a training set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{M}$, the problem of network pruning with a given constraint C can be generally formulated as the following optimization problem:

$$\arg\min_{\hat{\mathbf{W}}} \mathcal{L}(\hat{\mathbf{W}}, \mathcal{D}) \quad \text{s.t.} \quad \Phi\left(f(\hat{\mathbf{W}}, x_i)\right) \leq C \tag{1}$$

where $\hat{\mathbf{W}} \subset \mathbf{W}$ is the remaining parameters after pruning and $\mathcal{L}$ is the loss of the task. $f(\cdot)$ encodes the network function, and $\Phi(\cdot)$ maps the network to the constraint C, such as latency, FLOPs, or memory. We primarily focus on latency in this work while the method easily scales to other constraints.

The key to solving the aforementioned problem relies on identifying the portion of the network that satisfies the constraint while incurring minimum performance disruption:

$$\arg\max_{p_1,\cdots,p_l} \sum_{l=1}^{L} I_l(p_l), \text{ s.t. } \sum_{l=1}^{L} T_l(p_{l-1}, p_l) \leq C, \ \forall l \ \ 0 \leq p_l \leq N_l \tag{2}$$

where p_l denotes the number of kept neurons at layer l, $I_l(p_l)$ signals the maximum importance to the final accuracy with p_l neurons, and $T_l(p_{l-1}, p_l)$ checks on the associated latency contribution of layer l with p_{l-1} input channels and p_l output channels. p_0 denotes a fixed input channel number for the first convolutional block, *e.g.*, 3 for RGB images. We next elaborate on $I(\cdot)$ and $T(\cdot)$ in detail.

Importance score. To get the importance score of a layer to final accuracy, namely $I_l(p_l)$ in Eq. 2, we take it as the accumulated score from individual neurons $\sum_{j=1}^{p_l} \mathcal{I}_l^j$. We first approximate the importance of neurons using the Taylor expansion of the loss change [45]. Specifically, we prune on batch normalization layers and the importance of the n-th neuron in the l-th layer is calculated as

$$\mathcal{I}_l^n = \left| g_{\gamma_l^n} \gamma_l^n + g_{\beta_l^n} \beta_l^n \right|, \tag{3}$$

where g denotes the gradient of the weight, γ_l^n and β_l^n are the corresponding weight and bias from the batch normalization layer, respectively. Unlike a squared loss in [45], we use absolute difference as we observe slight improvements.

In order to maximize the total importance, we keep the most important neurons at a higher priority. To this end, we rank the neurons in the l_{th} layer according to their importance score in a descending order and denote the importance score of the j_{th}-ranked neuron as $\mathcal{I}_l^j$, thus we have

$$I_l(p_l) = \sum_{j=1}^{p_l} \mathcal{I}_l^j, \quad 0 \leq p_l \leq N_l, \ \mathcal{I}_l^1 \geq \cdots \geq \mathcal{I}_l^{N_l}. \tag{4}$$

Latency contribution. We empirically obtain the layer latency $T_l(p_{l-1}, p_l)$ in Eq. 2 by pre-building a layer-wise look-up table with pre-measured latencies. This layer latency corresponds to the aggregation of the neuron latency contribution of each neuron in the layer, c_l^j:

$$T_l(p_{l-1}, p_l) = \sum_{j=1}^{p_l} c_l^j, \quad 0 \leq p_l \leq N_l. \tag{5}$$

The latency contribution of the j-th neuron in the l-th layer can also be computed using the entries in the look up table as:

$$c_l^j = T_l(p_{l-1}, j) - T_l(p_{l-1}, j-1), \quad 1 \leq j \leq p_l. \tag{6}$$

https://doi.org/10.52202/068431-0937

Algorithm 1 Augmented Knapsack Solver

Input: Importance score $\{\mathcal{I}_l \in \mathbb{R}^{N_l}\}_{l=1}^{L}$ where $\mathcal{I}_l$ is sorted descendingly; Neuron latency contribution $\{c_l \in \mathbb{R}^{N_l}\}_{l=1}^{L}$; Latency constraint C.

1: maxV $\in \mathbb{R}^{(C+1)}$, keep $\in \mathbb{R}^{L\times(C+1)}$ ▷ maxV[c]: max importance under constraint c; keep[l, c]: # neurons to keep in layer l to achieve maxI[c]
2: **for** $l = 1, \ldots, L$ **do**
3: **for** $j = 1, \ldots, N_l$ **do**
4: **for** $c = 1, \ldots, C$ **do**
5: $v_{\text{keep}} = \mathcal{I}_l^j + \text{maxV}[c - c_l^j],\ v_{\text{prune}} = \text{maxV}[c]$ ▷ total importance can achieve under constraint c with object n being kept or not
6: **if** $v_{\text{keep}} > v_{\text{prune}}$ **and** keep$[l, c - c_l^j] == j - 1$ **then** ▷ check if it leads to higher score and if more important neurons in layer are kept
7: keep$[l, c] = j,\ \text{update_maxV}[c] = v_{\text{keep}}$
8: **else**
9: keep$[l, c] = \text{keep}[l, c-1],\ \text{update_maxV}[c] = v_{\text{prune}}$
10: **end if**
11: **end for**
12: maxV $\leftarrow$ update_maxV
13: **end for**
14: **end for**
15:
16: keep_n $=$ to save the kept neurons in model
17: **for** $l = L, \ldots, 1$ **do** ▷ retrieve the set of kept neurons
18: $p_l = \text{keep}[l, C]$
19: keep_n $\leftarrow$ keep_n $\cup$ $\{p_l$ top ranked neurons in layer $l\}$
20: $C \leftarrow C - \sum_{j=1}^{p_l} c_l^j$
21: **end for**

Output: Kept important neurons (keep_n).

In practice, we first rank globally neurons by importance and then consider their latency contribution. Thus, we can draw the following properties. If we remove the least important neuron in layer l, then the number of neurons will change from p_l to $p_l - 1$, leading to a latency reduction $c_l^{p_l}$ as this neuron's latency contribution score. We assign the potential latency reduction to neurons in the layer by the importance order. The most important neuron in that layer would always have a latency contribution c_l^1. At this stage, finding the right combination of neurons to keep imposes a combinatorial problem, and in the next section we tackle it via reward maximization considering latency and accuracy traits.

3.2 Augmented Knapsack Solver

Given both importance and latency estimates, we now aim at solving Eq. 2. By plugging back in the layer importance Eq. 4 and layer latency Eq. 5, we come to

$$\max \sum_{l=1}^{L}\sum_{j=1}^{p_l} \mathcal{I}_l^j, \quad \text{s.t.} \quad \sum_{l=1}^{L}\sum_{j=1}^{p_l} c_l^j \le C, \quad 0 \le p_l \le N_l, \quad \mathcal{I}_l^1 \ge \mathcal{I}_l^2 \ge \ldots \mathcal{I}_l^{N_l}. \tag{7}$$

This simplifies the overall pruning process into a knapsack problem only with additional preceding constraints. The preceding enforcement originates from the fact that for a neuron with rank j in the l_{th} layer, the neuron latency contribution only holds when all the neurons with rank $r = 1, \ldots, j-1$ are kept in the l_{th} layer and the rest of the neurons with rank $r = j+1, j+2, \cdots, N_l$ are removed. Yet the problem is solvable by specifying each item with a list of preceding items that need to be selected before its inclusion.

We augment the knapsack solver to consider the reordered neurons with descending importance score so that all the preceding neurons will be processed before it. A description of the pseudo code of the augmented knapsack solver is provided in Algo. 1 (a detailed explanation is provided in Appendix A). The augmented solver is required to make sure that the latency cost is correct.

3.3 Neuron Grouping

Considering each neuron individually results in burdensome computation during pruning. We next explore grouping neurons so that a number of them can be jointly considered and removed enabling faster pruning [69]. Neuron grouping helps exploit hardware-incurred channel granularity guided by the latency, speeds up knapsack solving of Eq. 7, and yields structures that maximize the GPU utilization, keeping as many parameters as possible under the similar latency. In addition, neuron grouping simplifies the knapsack problem that scales linearly with the number of candidates under consideration (see Line 3 of Alg. 1), thus speedups the solver as we'll discuss later.

We refer to the difference of neuron counts between two latency cliffs of the staircase-patterned latency as the latency step size. In our method, we group s channels in a layer as an entirety, where the value of s is equal to the latency step size. The neurons are grouped by the order of importance.

Then we aggregate the importance score and latency contribution for the grouped entity. For skip connections in ResNet and group convolutions in MobileNet, we not only group neurons within a layer, we also group the neurons sharing the same channel index from the connected layers [11, 39]. For cross-layer grouping, as the latency step size for different layers might be different, we use the largest group size among the layers. Latency-aware grouping enables additional performance benefits when compared to a heuristic universal grouping, as we will later show in the experiments.

3.4 Final HALP Regime

With all aforementioned steps, we formulate the final HALP as follows. The pruning process takes a trained network as input and prunes it iteratively to satisfy the requirement of a given latency budget C. We perform one pruning every r minibatches and repeat it k pruning steps in total. In particular, we set k milestones gradually decreasing the total latency to reach the goal via exponential scheduler [10], with $C^1 > C^2 > \cdots > C^k$, $C^k = C$. The algorithm gradually trims down neurons using steps below:

- **Step 1**. For each minibatch, we get the gradients of the weights and update the weights as during the normal training. We also calculate each neuron's importance score as Eq. 3.
- **Step 2**. Over multiple minibatches we calculate the average importance score for each neuron and rank them accordingly. Then we count the number of neurons remaining in each layer and dynamically adjust the latency contribution as in Eq. 6.
- **Step 3**. We group neurons as described in Sec. 3.3 and calculate group's importance and latency reduction. Then we get the nearest preceding group for each layer.
- **Step 4**. We execute the Algo. 1 to select the neurons being remained with current latency milestone. Repeat starting from the Step 1 until k milestones are reached.

Once pruning finishes we fine-tune the network to recover accuracy.

4 Experiments

We demonstrate the efficacy and feasibility of the proposed HALP method in this section. We use ImageNet ILSVRC2021 [53] for classification. We first study five architectures (ResNet50, ResNet101, VGG16 and MobileNet-V1/V2) on different platforms and compare our pruning results with the state-of-the-art methods on classification task. All the main results are obtained with our latency-aware grouping. We then study the impact of grouping size s on the pruned top-1 accuracy and the inference time to show the effectiveness of our grouping scheme. Finally, we further show the generalization ability of our algorithm by testing with object detection task. We introduce the details of experimental setting in appendix Sec. B and provide pruning results on CIFAR10 [29] in appendix Sec. C. We apply HALP targeting latency reduction on multiple platforms to show the scalability of our method: NVIDIA TITAN V GPU, Jetson TX2, Jetson Xavier and Intel CPU. All the latencies are measured with batched images to take advantage of computation capacities. We include experimental results for batch size 1 inference in the appendix, which also show the efficacy of our algorithm.

4.1 Results on ImageNet

ResNets. We start by pruning ResNet50 and ResNet101 and compare our results with state-of-the-art methods in Tab. 1 on TITAN V. In order to have a fair comparison of the latency, for all the other methods, we recreate pruned networks according to the pruned structures they published and measure the latency. Those methods showing '-' in the table do not have pruned structures published so we are unable to measure the latency. For our method, by setting the percentage of latency to remain after pruning to be X, we get the final pruned model and refer to it as HALP-$X\%$. We report FPS (frames per second) in the table and calculate the speedup of a pruned network as the ratio of FPS between pruned and unpruned models.

From the results comparison in Table. 1 we can find that for pruned networks with similar FLOPs using different methods, our method achieves the highest accuracy and also the fastest inference speed. This also shows that FLOPs do not correspond 1:1 to the latency. Among these methods for ResNet50 comparison, EagleEye [32] yields the closest accuracy to ours, but the speedup is lower than ours. In Table. 1, for the pruned ResNet50 network with 3G FLOPs remaining, our method

https://doi.org/10.52202/068431-0937

Table 1: ImageNet structural pruning results. We compare HALP for ResNet50 with two different dense baselines (left), ResNet101 and VGG16 (right up), MobileNet-V1 and MobileNet-V2 (right bottom) pruning experiments, with detailed comparison to state-of-the-art pruning methods over varying performance metrics. More comparisons and CIFAR10 experiments can be found in Appendix C.

Method	FLOPs (G)	Top1 (%)	Top5 (%)	FPS (im/s)	Speedup
ResNet50					
No pruning	4.1	76.2	92.87	1019	1×
ThiNet-70 [40]	2.9	75.8	90.67	-	-
AutoSlim [72]	3.0	76.0	-	1215	1.14×
MetaPruning [36]	3.0	76.2	-	-	-
GReg-1 [63]	2.7	76.3	-	1171	1.15×
HALP-80% (Ours)	3.1	**77.2**	**93.47**	**1256**	**1.23×**
0.75× ResNet50 [21]	2.3	74.8	-	1467	1.44×
ThiNet-50 [40]	2.1	74.7	90.02	-	-
AutoSlim [72]	2.0	75.6	-	1592	1.56×
MetaPruning [36]	2.0	75.4	-	1604	1.58×
GBN [71]	2.4	76.2	92.83	-	-
CAIE [66]	2.2	75.6	-	-	-
LEGR [8]	2.4	75.6	92.70	-	-
GReg-2 [63]	1.8	75.4	-	1414	1.39×
CHIP [57]	2.1	76.2	92.91	-	-
CHIP [57]	2.2	76.4	93.05	1345	1.32×
HALP-55% (Ours)	2.0	**76.5**	**93.05**	**1630**	**1.60×**
0.50× ResNet50 [21]	1.1	72.0	-	2498	2.45×
ThiNet-30 [40]	1.2	72.1	88.30	-	-
AutoSlim [72]	1.0	74.0	-	2390	2.45×
MetaPruning [36]	1.0	73.4	-	2381	2.34×
CAIE [66]	1.3	73.9	-	-	-
GReg-2 [63]	1.3	73.9	-	1514	1.49×
CHIP [57]	1.0	73.3	91.48	2369	2.32×
HALP-30% (Ours)	1.0	**74.3**	**91.81**	**2755**	**2.70×**
ResNet50 - EagleEye [32] baseline					
No pruning	4.1	77.2	93.70	1019	1×
EagleEye-3G [32]	3.0	77.1	93.37	1165	1.14×
HALP-80% (Ours)	3.0	**77.5**	**93.60**	**1203**	**1.18×**
EagleEye-2G [32]	2.1	76.4	92.89	1471	1.44×
HALP-55% (Ours)	2.1	**76.6**	**93.16**	**1672**	**1.64×**
EagleEye-1G [32]	1.0	74.2	91.77	2429	2.38×
HALP-30% (Ours)	1.2	**74.5**	**91.87**	**2597**	**2.55×**

Method	FLOPs (G)	Top1 (%)	FPS (im/s)	Speedup
ResNet101				
No pruning	7.8	77.4	620	1×
Taylor-75% [45]	4.7	77.4	750	1.21×
HALP-60% (Ours)	4.3	**78.3**	847	1.37×
HALP-50% (Ours)	**3.6**	77.8	**994**	**1.60×**
Taylor-55% [45]	2.9	76.0	908	1.47×
HALP-40% (Ours)	2.7	**77.2**	1180	1.90×
HALP-30% (Ours)	**2.0**	76.5	**1521**	**2.45×**
VGG-16				
No pruning	15.5	71.6	766	1×
FBS-3× [18]	5.1	71.2	-	-
HALP-30% (Ours)	**4.6**	**72.3**	1498	2.42×
FBS-5× [18]	3.0	70.5	-	-
HALP-20% (Ours)	**2.8**	**70.8**	1958	5.49×

Method	FLOPs (M)	Top1 (%)	FPS (im/s)	Speedup
MobileNet-V1				
No pruning	569	72.6	3415	1×
MetaPruning [36]	142	66.1	7050	2.06×
AutoSlim [72]	150	67.9	7743	2.27×
HALP-42% (Ours)	171	**68.3**	**7940**	**2.32×**
0.75× MobileNetV1	325	68.4	4678	1.37×
AMC [22]	285	70.5	4857	1.42×
NetAdapt [68]	284	69.1	-	-
MetaPruning [36]	316	70.9	4838	1.42×
EagleEye [32]	284	70.9	5020	1.47×
GDP [19]	287	71.3	-	-
HALP-60% (Ours)	297	**71.3**	**5754**	**1.68×**
MobileNet-V2				
No pruning	301	72.1	3080	1×
HALP-60% (Ours)	183	70.4	5668	1.84×
HALP-75% (Ours)	249	72.2	4110	1.33×

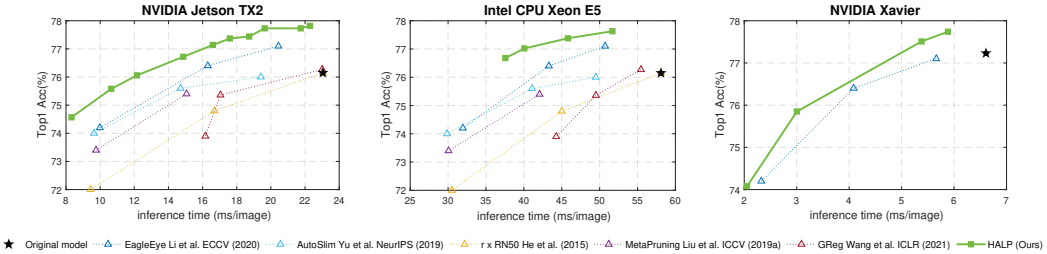


Figure 3: Pruning ResNet50 on the ImageNet dataset with NVIDIA Jetson TX2 (left), Intel CPU Xeon E5 (middle) and NVIDIA Xavier (right). The latency on Jetson TX2 and CPU is measured using PyTorch; on Xavier is measured using TensorRT FP32. Top-left is better.

achieves a .4% higher top1 accuracy and slightly (.04×) faster inference. It is expected that the advantage of our method for accelerating the inference is more obvious when it comes to a more compact pruned network, which is 14% (or .20×) additionally faster for a 2G-FLOPs network while increasing accuracy by .2%, and .17× faster with .3% higher accuracy compared to EagleEye-1G. We analyze the pruned network structure in detail in the supplementary material (Appendix Sec. Q). We plot the results comparison in Fig. 2, where we also add the results of training our pruned network with a teacher model RegNetY-16GF (top1 82.9%) [50]. With knowledge distillation, our model is 2.70× faster than the original model at 1% accuracy drop.

Scalability to other networks. We next experiment with three other models: VGG [56], MobileNetV1 [24] and MobileNetV2 [54]. Same as pruning on ResNets, in Tab. 1, we perform pruning with different latency constraints and compare with prior art. As shown, among these methods, the proposed HALP performs significantly better with higher accuracy and larger inference speedup.

Scalability to other platforms. Our approach is not limited to a single platform. In this section, we conduct the same ResNet50 experiments on three new platforms: NVIDIA Jetson TX2, NVIDIA

https://doi.org/10.52202/068431-0937

Table 2: HALP acceleration of ResNet50 on GPUs with TensorRT (version 7.2.1.6)

Model	Acc Drop	TITAN V GPU		RTX3080 GPU		
		FP32	FP16	FP32	FP16	INT8 (Acc Drop)
EagleEye-3G	−0.90%	1.14×	4.26×	1.06×	3.09×	6.31× (−0.84%)
HALP-80% (Ours)	−1.25%	**1.24×**	**4.70×**	**1.18×**	**3.32×**	**6.40×** (−1.02%)
EagleEye-2G	−0.18%	1.54×	5.10×	1.35×	3.68×	7.46× (0.27%)
HALP-55% (Ours)	−0.35%	**1.80×**	**6.36×**	**1.68×**	**4.45×**	**9.14×** (0.13%)
EagleEye-1G	2.02%	2.73×	7.81×	2.29×	5.61×	12.29× (2.55%)
HALP-30% (Ours)	1.76%	**2.91×**	**9.61×**	**2.56×**	**6.44×**	**14.12×** (2.38%)

Xavier, and Intel CPU Xeon E5, and compare the results in Fig 3. As shown, our approach consistently outperforms the other methods with more speedup and higher accuracy, as HALP leverages the latency characteristics of the platform to achieve a better accuracy-efficiency trade-off. Specifically, on these platforms, HALP yields up to relative $1.27\times$ speedup with slightly increment in top-1 accuracy compared to EagleEye [32], and $1.72\times$ faster than the original model. From these results, we can conclude that our method generalizes well to different platforms.

4.2 HALP Acceleration on GPUs with TensorRT

To make it closer to the real application in production, we also export the models into onnx format and test the inference speed with TensorRT. We run the inference of the model with FP32, FP16 and also INT8. For INT8, we quantize the model using entropy calibration with 2560 randomly selected ImageNet training images. Since the INT8 TensorCore speedup is not supported in TITAN V GPU, we only report the quantized results on RTX3080 GPU. The accelerations and the corresponding top1 accuracy drop (compared to PyTorch baseline model) are listed in Tab. 2. We include more pruning results specifically for **INT8 quantization** in Appendix N.

4.3 Design Effort for Pruning

In addition to noticeable performance boosts, HALP in fact requires less design effort compared to prior art, as summarized in Tab. 3 (details in Appendix O). NetAdapt [68] and AutoSlim [72] generate many proposals during iterative pruning. Then evaluations of the proposals are needed to select the best candidate for the next pruning iteration. EagleEye [32] pre-obtains 1000 candidates before pruning and evaluates all of them in order to get the best one. Such pruning candidate selection is intuitive but causes a lot of additional time costs. The computation cost for MetaPruning [36] and AMC [22] can be even higher because they need to train auxiliary network to generate the pruned structure.

Table 3: Comparison of extra computation required by pruning methods on ImageNet. Our approach is around *4.3*× *faster* than the next best method. Sub-network selection timing is approximated as running on same device (a NVIDIA V100).

Method	Evaluate proposals?	Auxiliary net training?	Sub-network selection (ResNet50)
NetAdapt [68]	Y	N	∼ 195h (GPU)
ThiNet [40]	Y	N	∼ 210h (GPU)
EagleEye [32]	Y	N	30h (GPU)
AutoSlim [72]	Y	Y	–
MetaPruning [36]	Y	Y	–
AMC [22]	N	Y	–
HALP (Ours)	**N**	**N**	6.5**h (GPU)** + 0.5**h (CPU)**

Compared to these methods, our method does not require auxiliary network training nor sub-network evaluation. The latency contribution in our method can be quickly obtained during pruning by the pre-generated latency lookup table. Although creating the table for the target platform might cost time, we only do it once for all pruning ratios. Solving the augmented knapsack problem brings extra computation, however, after neuron grouping, it only takes around additional 30 minutes of CPU time in total for ResNet50 pruning and less than 1 minute for MobileNetV1, which is negligible compared to the fine-tuning process or training additional models. Moreover, this is significantly lower than other methods, for example the fastest of them EagleEye [32] requires 30 GPU hours.

4.4 Efficacy of Neuron Grouping

We then show the benefits of latency-aware neuron grouping and compare the performance under different group size settings.

https://doi.org/10.52202/068431-0937

Performance comparison. As described in Sec. 3.3, we group s neurons in a layer as an entirety so that they are removed or kept together. Choosing different group sizes leads to different performances, and also different computation cost on the augmented knapsack problem solving. In our method, we set an individual group size for each layer according to each layer's latency step size in the look-up table. We name the grouping in our method as latency-aware grouping (LG). For instance, for a ResNet50, using this approach we set the individual group size of 23 layers to 32, of 20 layers to 64, and 10 layers to 128. Layers closer to the input tend to use a smaller group size. Another option for neuron grouping is to heuristically set a fixed group size for all layers as literature does [69].

Fig. 4 shows the performance of our grouping approach compared to various fixed group sizes for a ResNet50 pruned with different constraints.

As shown, using small group sizes yields the worst performance independently of the latency constraint. At the same time, a very large group such as 256 do also harm the final performance. Intuitively, a large group size averages the contribution of many neurons and therefore is not discriminative enough to select the most important ones. Besides, large groups might promote pruning the entire layer in a single pruning step, leading to performance drop. On the other hand, small group sizes such as 2 promote removing unimportant groups of neurons. These groups do not significantly improve the latency, but can contribute to the final performance. In contrast, our latency-aware grouping performs the best, showing the efficacy of our grouping scheme.

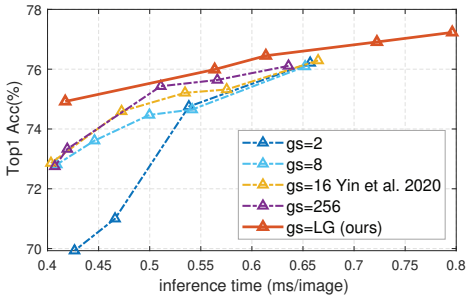
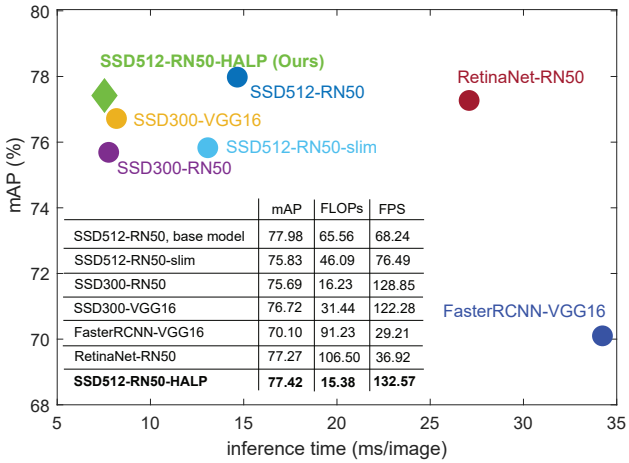


Figure 4: Performance comparison of different group size settings for ResNet50 pruning on ImageNet. We compare to heuristic-based group selection studied by [69]. LG denotes our proposed latency-aware grouping in HALP that yields consistent latency benefits per accuracy.

Algorithm efficiency improvement. Setting the group size according to the latency step size not only improves the performance, but also reduces computation cost on knapsack problem solving for neuron selection since it reduces the total number of object N to a smaller value. In our ResNet50 experiment, except for the first convolution layer, the group size of other layers varies from 32 to 128. By neuron grouping, the value of N can be reduced to 215, which takes around one minute on average at each pruning step to solve the knapsack problem on CPU. We have 30 pruning steps in total in our experiments, thus the time spent on neuron selection is around 30 minutes in total, which can be negligible compared to training time.

4.5 Generalization to Object Detection

To show the generalization ability of our proposed HALP algorithm, we also apply the algorithm to the object detection task. In this experiment we take the popular architecture Single Shot Detector (SSD) [35] on the PASCAL VOC dataset [13]. Following the "07+12" setting in [35], we use the union of VOC2007 and VOC2012 trainval as our training set and use the VOC2007 test as test set.We pretrain a SSD512 detector with ResNet50 as the backbone. The details of the SSD structure are elaborated in the appendix. Same to classification task, we prune the trained detector and finetune afterwards. We only prune the backbone network in the detector. The results



	mAP	FLOPs	FPS
SSD512-RN50, base model	77.98	65.56	68.24
SSD512-RN50-slim	75.83	46.09	76.49
SSD300-RN50	75.69	16.23	128.85
SSD300-VGG16	76.72	31.44	122.28
FasterRCNN-VGG16	70.10	91.23	29.21
RetinaNet-RN50	77.27	106.50	36.92
SSD512-RN50-HALP	**77.42**	**15.38**	**132.57**

Table 4: HALP for object detection on the PASCAL VOC dataset. Detailed numbers in the Appendix H.

https://doi.org/10.52202/068431-0937

in Fig. 4 show that the pruned detector maintains the similar final mAP but reduce the FLOPs and improve the inference speed greatly, with 77% FLOPs reduction and around $1.94\times$ speedup at the cost of only 0.56% mAP drop. We compare the pruned detector to some other commonly-used detectors in the table. The results show that pruning a detector using HALP improves performance in almost all aspects.

5 Conclusion

We proposed hardware-aware latency pruning (HALP) that focuses on pruning for underlying hardware towards latency budgets. We formulated pruning as a resource allocation optimization problem to achieve maximum accuracy within a given latency budget. We further proposed a latency-aware neuron grouping scheme to improve latency reduction. Over multiple neural network architectures, classification and detection tasks, and changing datasets, we have shown the efficiency and efficacy of HALP by showing consistent improvements over state-of-the-art methods. Our work effectively reduces the latency while maintaining the accuracy, which could significantly impact applications in resource-constrained environments, such as autonomous vehicles or other mobile devices.

References

[1] Yonathan Aflalo, Asaf Noy, Ming Lin, Itamar Friedman, and Lihi Zelnik. Knapsack pruning with inner distillation. *arXiv preprint arXiv:2002.08258*, 2020.

[2] Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pages 2270–2278, 2016.

[3] Rumen Andonov, Vincent Poirriez, and Sanjay Rajopadhye. Unbounded knapsack problem: Dynamic programming revisited. *European Journal of Operational Research*, 123(2):394–407, 2000.

[4] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[5] Yaohui Cai, Zhewei Yao, Zhen Dong, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Zeroq: A novel zero shot quantization framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13169–13178, 2020.

[6] Changan Chen, Frederick Tung, Naveen Vedula, and Greg Mori. Constraint-aware deep neural network compression. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 400–415, 2018.

[7] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[8] Ting-Wu Chin, Ruizhou Ding, Cha Zhang, and Diana Marculescu. Towards efficient model compression via learned global ranking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1518–1528, 2020.

[9] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, P Vajda, M Uyttendaele, and Niraj K Jha. ChamNet: Towards efficient network design through platform-aware model adaptation. In *CVPR*, 2019.

[10] Pau de Jorge, Amartya Sanyal, Harkirat S Behl, Philip HS Torr, Gregory Rogez, and Puneet K Dokania. Progressive skeletonization: Trimming more fat from a network at initialization. *arXiv preprint arXiv:2006.09081*, 2020.

[11] Xiaohan Ding, Guiguang Ding, Yuchen Guo, and Jungong Han. Centripetal sgd for pruning very deep convolutional networks with complicated structure. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4943–4953, 2019.

https://doi.org/10.52202/068431-0937

[12] Jin-Dong Dong, An-Chieh Cheng, Da-Cheng Juan, Wei Wei, and Min Sun. Dpp-net: Device-aware progressive search for pareto-optimal neural architectures. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 517–531, 2018.

[13] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[14] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019.

[15] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *International Conference on Learning Representations (ICLR)*, 2019.

[16] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

[17] Susan Gao, Xin Liu, Lung-Sheng Chien, William Zhang, and Jose M Alvarez. Vacl: Variance-aware cross-layer regularization for pruning deep residual networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[18] Xitong Gao, Yiren Zhao, Łukasz Dudziak, Robert Mullins, and Cheng-zhong Xu. Dynamic channel pruning: Feature boosting and suppression. *arXiv preprint arXiv:1810.05331*, 2018.

[19] Yi Guo, Huan Yuan, Jianchao Tan, Zhangyang Wang, Sen Yang, and Ji Liu. Gdp: Stabilized neural network pruning via gates with differentiable polarization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5239–5250, 2021.

[20] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. corr abs/1512.03385 (2015), 2015.

[22] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[23] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[24] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[25] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

[26] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[27] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311, 2017.

[28] Yeonwoo Jeong, Deokjae Lee, Gaon An, Changyong Son, and Hyun Oh Song. Optimal channel selection with discrete qcqp. In *International Conference on Artificial Intelligence and Statistics*, pages 7915–7941. PMLR, 2022.

[29] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[30] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. SNIP: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations*, 2019.

[31] Sam Leroux, Pavlo Molchanov, Pieter Simoens, Bart Dhoedt, Thomas Breuel, and Jan Kautz. Iamnn: Iterative and adaptive mobile neural network for efficient image classification. *arXiv preprint arXiv:1804.10123*, 2018.

[32] Bailin Li, Bowen Wu, Jiang Su, and Guangrun Wang. Eagleeye: Fast sub-net evaluation for efficient neural network pruning. In *European Conference on Computer Vision*, pages 639–654. Springer, 2020.

[33] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *International Conference on Learning Representations (ICLR)*, 2017.

[34] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[35] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[36] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3296–3305, 2019.

[37] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019.

[38] Ilya Loshchilov and Frank Hutter. Theoretical analysis of self-training with deep networks on unlabeled data. In *International Conference on Learning Representations*, 2017.

[39] Jian-Hao Luo and Jianxin Wu. Neural network pruning with residual-connections and limited-data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1458–1467, 2020.

[40] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[41] Sangkug Lym, Esha Choukse, Siavash Zangeneh, Wei Wen, Sujay Sanghavi, and Mattan Erez. Prunetrain: fast neural network training by dynamic sparse model reconfiguration. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13, 2019.

[42] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018.

[43] Silvano Martello, David Pisinger, and Paolo Toth. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management science*, 45(3):414–424, 1999.

[44] Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.

[45] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019.

[46] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

 https://doi.org/10.52202/068431-0937

[47] Ravi Teja Mullapudi, Steven Chen, Keyi Zhang, Deva Ramanan, and Kayvon Fatahalian. Online model distillation for efficient video inference. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3573–3582, 2019.

[48] Nvidia. Convolutional networks for image classification in pytorch. https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/Classification/ConvNets#convolutional-networks-for-image-classification-in-pytorch, 2020.

[49] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[50] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10428–10436, 2020.

[51] Valentin Radu, Kuba Kaszyk, Yuan Wen, Jack Turner, José Cano, Elliot J Crowley, Björn Franke, Amos Storkey, and Michael O'Boyle. Performance aware convolutional neural network channel pruning for embedded gpus. In *2019 IEEE International Symposium on Workload Characterization (IISWC)*, pages 24–34. IEEE, 2019.

[52] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28:91–99, 2015.

[53] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vision*, 115(3):211–252, December 2015.

[54] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[55] Maying Shen, Pavlo Molchanov, Hongxu Yin, and Jose M Alvarez. When to prune? a policy towards early structural pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12247–12256, 2022.

[56] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations (ICLR)*, 2015.

[57] Yang Sui, Miao Yin, Yi Xie, Huy Phan, Saman Aliari Zonouz, and Bo Yuan. Chip: Channel independence-based pruning for compact neural networks. *Advances in Neural Information Processing Systems*, 34:24604–24616, 2021.

[58] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

[59] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[60] Rishabh Tiwari, Udbhav Bamba, Arnav Chavan, and Deepak Gupta. Chipnet: Budget-aware pruning with heaviside continuous approximations. In *International Conference on Learning Representations*, 2021.

[61] Arash Vahdat, Arun Mallya, Ming-Yu Liu, and Jan Kautz. Unas: Differentiable architecture search meets reinforcement learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11266–11275, 2020.

[62] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2020.

[63] Huan Wang, Can Qin, Yulun Zhang, and Yun Fu. Neural pruning via growing regularization. In *International Conference on Learning Representations (ICLR)*, 2021.

[64] Tianzhe Wang, Kuan Wang, Han Cai, Ji Lin, Zhijian Liu, Hanrui Wang, Yujun Lin, and Song Han. Apq: Joint search for network architecture, pruning and quantization policy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2078–2087, 2020.

[65] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019.

[66] Yu-Cheng Wu, Chih-Ting Liu, Bo-Ying Chen, and Shao-Yi Chien. Constraint-aware importance estimation for global filter pruning under multiple resource constraints. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 686–687, 2020.

[67] Wenhan Xia, Hongxu Yin, Xiaoliang Dai, and Niraj K Jha. Fully dynamic inference with deep neural networks. *IEEE Transactions on Emerging Topics in Computing*, 2021.

[68] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 285–300, 2018.

[69] Hongxu Yin, Pavlo Molchanov, Jose M Alvarez, Zhizhong Li, Arun Mallya, Derek Hoiem, Niraj K Jha, and Jan Kautz. Dreaming to distill: Data-free knowledge transfer via DeepInversion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8715–8724, 2020.

[70] Hongxu Yin, Arash Vahdat, Jose M Alvarez, Arun Mallya, Jan Kautz, and Pavlo Molchanov. A-ViT: Adaptive tokens for efficient vision transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10809–10818, 2022.

[71] Zhonghui You, Kun Yan, Jinmian Ye, Meng Ma, and Ping Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1909.08174*, 2019.

[72] Jiahui Yu and Thomas Huang. Autoslim: Towards one-shot architecture search for channel numbers. *arXiv preprint arXiv:1903.11728*, 2019.

[73] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Abstract.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

https://doi.org/10.52202/068431-0937

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Please see the Section M about the discussion of latency lookup table.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Please see Section 4 of main paper and Section I of supplementary material. The link to the code is available in the abstract.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Please see Section 4 and more details in appendix Sec. B.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] We observed quite stable final performance of the pruning.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Please see Section 4 and more details in appendix Sec. B.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Please see Section 4.
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Code will be released. The link to the code is provided in the abstract.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] The ImageNet [53] and PASCAL VOC [13] dataset are open source and available for non-commercial academic research.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] The ImageNet [53] and PASCAL VOC [13] dataset are open source and available for non-commercial academic research.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

https://doi.org/10.52202/068431-0937