TOWARDS DYNAMIC SPARSIFICATION BY ITERATIVE PRUNE-GROW LOOKAHEADS

Anonymous authors
Paper under double-blind review

ABSTRACT

Model sparsification aims to accelerate and compress models by removing redundant connections. Most methods relied on estimating parameter importance on pretrained models, which is computationally intensive to train. Other approaches try to reduce training cost through sparse training either by producing sparsified models early in training or randomly and greedily explore new sparse architecture during training by prune and grow. The early weak feature representation and greediness of the unreliable exploration strategy limit the quality of such methods. In this work, we propose a simple, novel, and effective model sparsification method called *LookAhead* and directly address the shortcomings of previous methods by enforcing an iterative exploitation-exploration process into dynamic sparsity pursuit through training. The exploitation phase assumes stability of current sparse architecture and trains it to maximize performance. Whereas the exploration phase challenges the assumption by reactivating pruned parameters, quickly updating them while freezing existing ones, and updating the sparse architecture. We demonstrate the effectiveness and efficiency of *LookAhead* with extensive experiments covering both unstructured and structured latency sparsity on ImageNet and CIFAR-10 for classification and PASCAL VOC for object detection across multiple configurations. Not only do we observe state-of-art accuracy of *LookAhead* surpassing multiple latest weight and channel pruning methods but also tremendous training cost saving. We also design metrics to study the effectiveness of sparsity architecture exploration strategy.

1 INTRODUCTION

Convolutional Neural Networks (CNNs) (LeCun et al., 1998) have constituted the modern cornerstone for fundamental computer vision tasks such as image classification, object detection, and segmentation. As the literature progresses for improved performance, so do the model size, computation, and latency. Modern networks may contain billions of parameters (Brown et al., 2020) for competitive advantages but the associated huge costs hinder deployment to applications that suffer stringent resources, *e.g.,* edge device applications. Therefore, effectively compressing CNNs through removing redundant parameters for efficient storage and computation has been a very active research area. Despite the diversity in the compression schemes proposed in recent years, chasing sparsity via *pruning*, either in a structured (removing channels) or unstructured manner (removing weights), has been one of the central topics with initial solutions dated back to (Hassibi & Stork, 1992; LeCun et al., 1990). A plethora of pruning methods has yielded profound progresses in model compression (LeCun et al., 1990; Han et al., 2015; Louizos et al., 2018; Gale et al., 2019; Zhou et al., 2021), providing practical speedup and memory benefits (Li et al., 2017; Molchanov et al., 2019; He et al., 2020; Lin et al., 2020a; Yuan et al., 2021) through hardware-friendly sparsity (Wu et al., 2020; Yang et al., 2018; Li et al., 2020; Shen et al., 2022b) that trades off accuracy and performance via FLOPs/latency. Despite noticeable progresses, the majority of literature focuses on pruning pretrained models, causing unnecessary initial training costs when (i) knowingly trains a full model that is redundant for final inference, (ii) but, seemingly important, with sufficient connections and high enough degree-of-freedoms to support exploration with optimizer.

The recent trend of exploring model sparsity without dense model pretraining (Frankle & Carbin, 2018; Lee et al., 2018; Wang et al., 2019; Tanaka et al., 2020; Wimmer et al., 2020; Liu et al.,

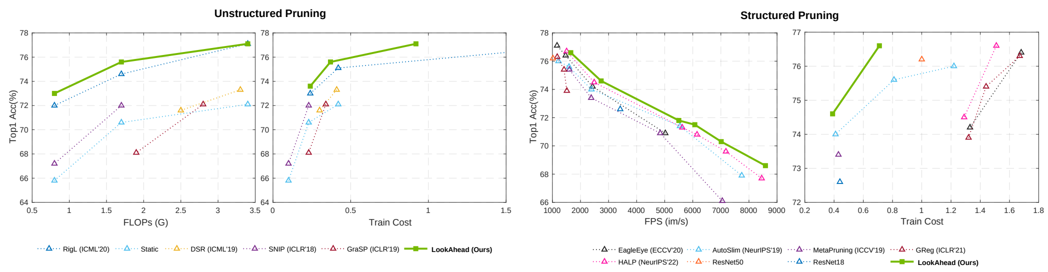

Figure 1: LookAhead unstructured and structured pruning results on ImageNet. **Left:** Unstructured weight sparsity with different pruning ratios as a function of FLOPs and train cost, the top-left is better; **Right:** Structured pruning targeting various latency constraints, as a function of frame per second during inference where the top-right is better, and train cost the top-left is better.

2021; Alvarez & Salzmann, 2016; Shen et al., 2022a) has gained attention for promising efficient sparse training paradigms. Such methods generate a sparse model right at the initialization or early in the training stage. Yet one noticeable challenge for the trend remains - a sharp performance drop caused by (i) a limited number of data samples processed before pruning, and (ii) the inability of network capacity to recover once pruned. The former causes difficulty in model exploration and potential lacks proper guidance as the rich feature manifold is not learned yet, whereas the latter prevents the network from regaining the capacity to learn the needed richer manifold. Addressing these challenges for early pruning methods will make them more compelling.

We solve (i) and (ii) simultaneously by formulating model sparsification as an iterative exploitation-exploration process to pursue dynamic sparsity during training. *The exploitation phase* assumes that the sparse model at hand has the final structure thus performs training on it for better exploitation of the convergence space. *The exploration phase* challenges the quality of the current sparse architecture and explores the space of the previously removed weights for a fresh and higher-quality sparse structure via growth-and-prune. Several previous works (Mocanu et al., 2018; Bellec et al., 2018; Dettmers & Zettlemoyer, 2019; Dai et al., 2019; Wortsman et al., 2019; Evci et al., 2020; Ma et al., 2021) including the popular RigL (Evci et al., 2020), mostly referred to as *dynamic sparse training*, have considered the idea of such growth-and-prune scheme to search better sparsity structure through training. Though yielding promising results, these methods are based on random or greedy exploration to search new sparsity architecture, leading to limited coverage of sparse patterns and potentially sub-optimal model quality. For more effective exploration to directly address those issues, we draw some insights from *Optimistic Initialization* approaches (Machado et al., 2015; Lobel et al., 2022), which are usually used to deal with the random nature of greedy exploration in tasks like Reinforcement Learning. We show that the resulting solution, named as LookAhead, effectively overcome the shortcomings of the previous solutions. It can be easily embedded into training in a naturally iterative fashion, bringing the dynamic adjustment capability throughout the entire training and constituting a new pruning efficacy Pareto front.

We conduct extensive experiments and ablation studies to demonstrate the effectiveness and efficiency of LookAhead. Our results demonstrate state-of-the-art in a wide range of settings, covering (i) both structured and unstructured sparsity on (ii) CIFAR-10 (Krizhevsky et al., 2009) and ImageNet (Deng et al., 2009) for classification and PASCAL VOC (Everingham et al., 2010) for object detection with (iii) four widely-used model architectures: ResNet50 (He et al., 2016), WideResNet (Zagoruyko & Komodakis, 2016), MobileNet-V1 (Howard et al., 2017), and SSD(Liu et al., 2016) across pruning targets including (iv) weight sparsity and latency. Figure 1 offers a glimpse of our comprehensive experiments. We do not only observe superior state-of-the-art results of LookAhead, but also tremendous training cost saving. Moreover, to validate LookAhead indeed improves upon previous random and greedy methods, we design metrics to directly evaluate exploration effectiveness of LookAhead and explicitly compare with them. Our contributions can be summarized as follows:

- We propose LookAhead, a simple, novel, and effective dynamic network sparsification method with an iterative exploitation-exploration scheme.
- We design new metrics to evaluate effectiveness of sparsity structure exploration strategy and empirically demonstrate ours is more performant than previous solutions.

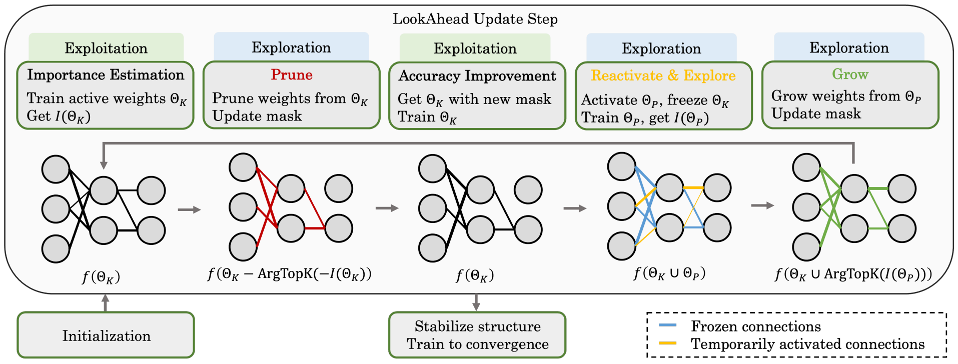
LookAhead Update Step
Exploitation
Importance Estimation
Train active weights Θ_K
Get $I(\Theta_K)$
Exploration
Prune
Prune weights from Θ_K
Update mask
Exploitation
Accuracy Improvement
Get Θ_K with new mask
Train Θ_K
Exploration
Reactivate & Explore
Activate Θ_P, freeze Θ_K
Train Θ_P, get $I(\Theta_P)$
Exploration
Grow
Grow weights from Θ_P
Update mask
$f(\Theta_K)$
$f(\Theta_K - \text{ArgTopK}(-I(\Theta_K))$
$f(\Theta_K)$
$f(\Theta_K \cup \Theta_P)$
$f(\Theta_K \cup \text{ArgTopK}(I(\Theta_P)))$
Initialization
Stabilize structure
Train to convergence
Frozen connections
Temporarily activated connections

Figure 2: An overview of LookAhead. In each LookAhead update step, we first train the kept weights Θ_K for H steps then prune a number of connections in the existing architecture. We later train the weights just selected for J steps for better exploiting the current architecture. For exploration of a potentially better sparse architecture, we temporarily activate all of the removed weights Θ_P and train them for K steps while freezing Θ_K. We then evaluate the importance scores of the activated Θ_P to grow the top-ranked weights. This completes one full LookAhead update step, and it is repeated until the update period ends.

- We demonstrate that the method outperforms previous state-of-the-art sparsification methods by large margin across multiple datasets, model architectures, target compactness, pruning criterion, for both structured and unstructured sparsity.
- We further instantiate the method with a hardware-aware compression scheme taking latency into account for hardware-friendly sparsity, setting the new state-of-the-art for latency pruning.

2 Related Works

In general, our work can be categorized into the field of model compression, or model sparsification and pruning in particular. We will provide recap of three popular schemes below. LookAhead is closest to the family of *Dynamic Sparse Training* works but differs from them in that we explicitly address the randomness or greedy nature of the exploration to discover enhanced sparse structure more solidly and effectively during the training.

Pruning from Pretrained Networks The majority of pruning methods evaluate parameters weight magnitude as their importance scores on well-performed pretrained dense weights and remove the lower-ranked ones in an either one-shot (Thimm & Fiesler, 1995; Ström, 1997) or iterative (Han et al., 2015; Narang et al., 2017; Zhu & Gupta, 2017; Gale et al., 2019) fashion, followed by an additional finetuning session to recover the accuracy. In addition to the magnitude-based indicator, there have been other importance criterion proposed such as Hessian-based scores (LeCun et al., 1990; Hassibi & Stork, 1992) and probability-based scores (Zhou et al., 2021; Srinivas et al., 2017; Molchanov et al., 2017). In this paper, we focus on weight magnitude metric as the saliency score for unstructured sparsity.

Some works, targetting structured sparsity, aim to prune convolutional filters (Li et al., 2017) or attention heads (Michel et al., 2019), thus enjoy immediate memory and latency benefit without specialized hardware and library support (Han et al., 2016). Exemplary channel importance criterion relied on metrics like weight norm (Li et al., 2017; Chin et al., 2020; He et al., 2020; 2018a; Yang et al., 2018), Taylor expansion (Lin et al., 2018; Molchanov et al., 2019; You et al., 2019), geometric median (He et al., 2019), and feature maps rank (Lin et al., 2020a). Other works (Chen et al., 2018; Shen et al., 2022b) including the very recent HALP consider channel pruning under a latency or FLOPs constraint, aiming for more hardware-friendly structured sparse architecture with practical speed-up. In our latency-constrained structured sparsity experiment, we build upon the pruning scheme developed in HALP, exploring better sparse structure in terms of both accuracy and inference latency.

Static Sparse Training Albeit the decent performance of pruning on pretrained models, the dense model pretraining is usually computationally demanding and redundant. To address this, a group of works (Frankle & Carbin, 2018; Lee et al., 2018; Wang et al., 2019; Tanaka et al., 2020; Wimmer et al., 2020; Liu et al., 2021; van Amersfoort et al., 2020; Verdenius et al., 2020; Shen et al., 2022a) trained sparse networks generated from scratch or early in training. This is also referred to as *zero-shot pruning* or *pruning at early stage*. Despite the reduced computation, the performance of these methods are notably worse than pruning from the pretrained since they compress models based on a limited number of data samples which usually can't represent the rich features in large-scale datasets.

Dynamic Sparse Training Another group of works (Mocanu et al., 2018; Dettmers & Zettlemoyer, 2019; Mostafa & Wang, 2019; Kusupati et al., 2020; Wortsman et al., 2019; Dai et al., 2019; Evci et al., 2020; Lin et al., 2020b; Ma et al., 2021; Yuan et al., 2021) have considered the idea of repeated alternating prune-grow sessions to dynamically configure the sparsity structure through training from scratch, giving the model more flexibility. This scheme is also referred to as *iterative prune-grow* or *neural rewiring*. SET (Mocanu et al., 2018) prunes weights according to the standard magnitude criterion then adds weights back at random. DeepR (Bellec et al., 2018) augments Stochastic Gradient Descent (SGD) with a random walk in parameter space. SNFS (Dettmers & Zettlemoyer, 2019) uses momentum of each parameter as the growing criterion. NeST (Dai et al., 2019) and RigL (Evci et al., 2020) employ magnitude-based pruning and grow greedily based on instantenious gradients on a small data-batch, achieving promising results. PC-GAP (Ma et al., 2021) iteratively prune and grow in a schedule by breaking the model into several partitions. Though decent in accuracy the method needs 1.2K training epochs on ImageNet. DPF (Lin et al., 2020b) and DCIL (Kim et al., 2021) dynamically train the model with soft masking techniques estimating gradients on sparse parameters. Very few dynamic sparse training works consider structured sparsity mainly due to the difficultly of estimating pruned channel gradients. SCS (Yuan et al., 2021) formulates the model growing into the optimization problem and utilizes continuous structured sparsification.

3 ITERATIVE PRUNE-GROW LOOKAHEAD(S)

We now introduce the details of our approach. Without loss of generalizability, we focus on unstructured pruning to describe the algorithm and leave the structured sparsity formulation for the Supplemental material.

We iteratively run LookAhead step during training to update the sparse architecture on-the-fly until reaching the total number of LookAhead update steps needed. The new sparse architecture is defined concretely by a prune stage which removes a fraction of parameters based on the saliency metric and a grow stage which activates a selected portion of pruned parameters. The novelty of LookAhead lies in how we select the pruned weights to grow back and how we schedule each prune and grow session with interleaved training stages to enforce exploitation-exploration. As mentioned, we explicitly improve upon the random and greedy heuristic in previous solutions (Mocanu et al., 2018; Dai et al., 2019; Evci et al., 2020) with insights drawn from *Optimistic Initialization* approaches(Machado et al., 2015; Lobel et al., 2022). With this strategy, all the actions are first deemed to be optimal and then are explored at least multiple times to challenge the assumption. We perform this by first temporarily activating all of the weights to be explored then training those connections quickly for a few iterations while ***freezing*** the currently selected architecture to look ahead the performance if growing those parameters back to the currently selected sparse weights. We evaluate importance scores of the temporarily activated weights which will unveil what parameters are worth continued exploration and grow the top-ranked ones. We discuss next the details of the approach.

Notation. Let us consider a neural network with weights $\Theta = \{\Theta^{(l)}\}_1^L$, where L is the number of layers in the network, $\Theta^{(l)}$ is the set of parameters for the layer l and m^l is the number of parameters of that layer, with $m = \sum_{l=1}^{L} m^l$ the total number of parameters in the network. We also define a binary mask $\mathcal{B} = \{\mathcal{B}_i\}_1^m, \mathcal{B}_i \in \{0, 1\}$ to identify the parameters of the network that are meant to be kept $\Theta_K = \{\Theta_i; \mathcal{B}_i = 1\}$, and those to be removed $\Theta_P = \{\Theta_i; \mathcal{B}_i = 0\}$, such that $\Theta = \Theta_K \cup \Theta_P$. Also suppose our target sparsity is S. Given a training set $\mathcal{D}$ consisting of N input-output pairs $\{(x_i, y_i)\}_{i=1}^N$, we can formulate the learning and sparsification of the network as solving an optimization problem of the form

$$\min_{\Theta,\mathcal{B}} \frac{1}{N} \sum_{i=1}^{\frac{N}{|\mathbf{B}|}} \ell(f(\Theta \odot \mathcal{B}; \mathbf{x}^i), \mathbf{y}^i), \qquad (1)$$
$$\text{s.t.} ||\mathcal{B}||_0 \leq (1 - S) \cdot m,$$

where $\odot$ is the element-wise multiplication, $\|\cdot\|_0$ is the L_0-norm, and $\ell(\cdot)$ is the training loss over a batch $(\mathbf{x}^i, \mathbf{y}^i) = \{(x_j, y_j)\}_{i\cdot|B|}^{(i+1)\cdot|B|}$ of size $|B|$ sampled from the training set $\mathcal{D}$. We next describe the LookAhead steps, the core components of the algorithm.

3.1 LookAhead

LookAhead is an iterative approach where each step consists of five stages: *Importance Estimation*, *Prune*, *Accuracy Improvement*, *Reactivate & Explore*, and *Grow*. The overview of the method is shown in Figure 2 and the algorithmic description in Algorithm 1.

Importance Estimation. The process starts with an initial training stage to exploit the current selected architecture Θ_K that has the target sparisty of S. In short, we train the kept weights Θ_K for H iterations and accumulate the pruning importance score while updating the network weights. In this paper, without loss of generality, we consider magnitude importance (Han et al., 2015) for unstructured sparsity and Taylor importance (Molchanov et al., 2019) for structured sparsity to compute Importance$(\cdot)$.

Prune. In the second stage, we remove a fraction of currently active parameters Θ_K. For each layer l, we prune the parameters given by ArgTopK$(-\mathcal{I}(\Theta_K^l), n^l)$, where ArgTopK$(\cdot, k)$ gives the indices of the top-k elements of its input. After pruning, we set the mask elements for the pruned parameters in $\mathcal{B}$ to 0, and Θ_K, Θ_P are also updated accordingly. Notice that we do *not* zero out the actual weight values, but rather *keep* them and mask weights during forward pass as $f(\Theta \odot \mathcal{B}; x^i)$. To determine n^l, the number of parameters to be selected each time, we follow the prior work (Evci et al., 2020; Ma et al., 2021) and use a cosine decay function f_{decay} (Evci et al., 2020), such that the number of neurons to be pruned is defined as:

Algorithm 1 LookAhead Pseudocode

```
Input: $\Theta$, $\mathcal{D}$, $S$, $f_{\text{decay}}$, $\alpha$, $T$, $H$, $J$, $K$
1:  $\Theta_K, \Theta_P, \mathcal{B} \leftarrow$ Sparsify $\Theta$ using $S$
2:  $\Delta T \leftarrow H + J + K$
3:  $t, flag \leftarrow 0, 0$
4:  for $i \leftarrow 1$ to $|\mathcal{D}|$ do
5:      if $(i + J + K) \mod \Delta T = 0$ and $t < T$ then
6:          for $l \leftarrow 1$ to $L$ do
7:              $\mathcal{I}_K^l \leftarrow$ Importance$(\Theta_K^l)$
8:              $n^l \leftarrow f_{\text{decay}}(t; \alpha, T) \cdot m^l \cdot (1 - S^l)$
9:              $\mathbb{I}_p \leftarrow$ ArgTopK$(-I_K^l, n^l)$
10:             //Prune Connections
11:             Update $\Theta_K, \Theta_P, \mathcal{B}$ with $\mathbb{I}_p$
12:         end for
13:     else if $(i + K) \mod \Delta T = 0$ and $t < T$ then
14:         $flag \leftarrow 1$
15:     else if $i \mod \Delta T = 0$ and $t < T$ then
16:         for $l \leftarrow 1$ to $L$ do
17:             $\mathcal{I}_P^l \leftarrow$ Importance$(\Theta_P^l)$
18:             $\mathbb{I}_g \leftarrow$ ArgTopK$(I_P^l, n^l)$
19:             //Grow Connections
20:             Update $\Theta_K, \Theta_P, \mathcal{B}$ with $\mathbb{I}_g$
21:         end for
22:         $t \leftarrow t + 1$
23:         $flag \leftarrow 0$
24:     end if
25:     if $flag$ then
26:         //Reactivate&Explore
27:         $\ell_i \leftarrow \ell(f(\Theta_P \cup \Theta_K; x_i), y_i)$
28:         $\Theta_P \leftarrow lr \cdot \nabla_{\Theta_P} l_i$
29:     else
30:         //Importance Estimation
31:         //Accuracy Improvement
32:         $\ell_i \leftarrow \ell(f(\Theta \odot \mathcal{B}; x_i), y_i)$
33:         $\Theta_K \leftarrow lr \cdot \nabla_{\Theta_K} l_i$
34:     end if
35: end for
```

$$n^l = f_{\text{decay}}(t; \alpha, T) m^l (1 - S^l), \tag{2}$$

where T is the total number of LookAhead update steps, m^l is the number of parameters in layer l, and α is the initial update ratio.

Accuracy Improvement. With the newly selected set of Θ_K, we carry out another training stage for J iterations on Θ_K with the goal to stabilize and fully exploit the architecture just selected to improve its performance. We show in ablation that this further exploitation step is crucial to the performance of LookAhead.

Reactivate & Explore. We now explore the pruned weights Θ_P for an updated better sparse architecture. In *reactivate*, we temporarily activate all of the pruned connections Θ_P by setting all elements in $\mathcal{B}$ to 1 then quickly update them for K iterations in *Reactivate & Explore* for a solid exploration. Notice that in the previous *Prune* stage, we do not zero out the weight values Θ but only the mask $\mathcal{B}$. When reactivated, the previously pruned connections Θ_P inherit their MRU (*i.e.,* Most Recently Used values) before they were turned off. Importantly, we ***freeze*** the previously selected architecture Θ_K and only update Θ_P here. We later show in ablation that this freezing is curcial to the performance of LookAhead by preserving the current selected architecture for a stable

Method	Sparsity	Total	Sparsity Ratio 80%			Sparsity Ratio 90%		
	Distribution	Epochs	Top-1 Acc(%)↑	FLOPs($\times e^9$)↓	Train FLOPs($\times e^{18}$)↓	Top-1 Acc(%)↑	FLOPs($\times e^9$)↓	Train FLOPs($\times e^{18}$)↓
Dense (Li et al., 2020)			77.2	8.2	×1(w.r.t.3.2)	77.2	8.2	×1(w.r.t.3.2)
Static	Uniform	100	70.6	1.7	×0.23	65.8	0.8	×0.10
SNIP (Lee et al., 2018)	Uniform	100	72.0	1.7	×0.23	67.2	0.8	×0.10
SET (Mocanu et al., 2018)	Uniform	100	72.9	1.7	×0.23	69.6	0.8	×0.10
RigL (Evci et al., 2020)	Uniform	100	74.6	1.7	×0.23	72.0	0.8	×0.10
LookAhead(Ours)	Uniform	100	**75.6**	**1.7**	×0.26	**73.0**	**0.8**	×0.16
Static	ERK	100	72.1	3.4	×0.42	67.7	2.0	×0.24
SNIP (Lee et al., 2018)	Non-Uniform	100	69.7	**2.8**	×0.34	61.9	**1.9**	×0.23
GraSP (Wang et al., 2019)	Non-Uniform	100	72.1	**2.8**	×0.34	68.1	**1.9**	×0.23
DSR (Mostafa & Wang, 2019)	Non-Uniform	100	73.3	3.3	×0.41	71.6	2.5	×0.30
RigL (Evci et al., 2020)	ERK	100	75.1	3.4	×0.42	73.0	2.0	×0.24
SNFS (Dettmers & Zettlemoyer, 2019)	ERK	100	75.2	3.4	×0.61	72.9	2.0	×0.50
LookAhead(Ours)	ERK	80	75.6	3.4	×0.37	73.6	2.0	×0.24
LookAhead(Ours)	ERK	100	**76.2**	3.4	×0.45	**74.3**	2.0	×0.30
IP-FT (Wimmer et al., 2022)	Non-Uniform	200	77.2	–	×1.55*	75.8	–	×1.38*
DCIL (Kim et al., 2021)	Non-Uniform	100	76.2	–	×1.80*	75.3	–	×1.75*
LookAhead(Ours)	ERK	200	77.1	**3.4**	× 0.92	75.7	**2.0**	×0.60
RigL (Evci et al., 2020)	ERK	500	77.1	**3.4**	×2.10	76.4	**2.0**	×1.23
LookAhead(Ours)	ERK	500	**77.8**	**3.4**	×2.30	**76.8**	**2.0**	×1.50

Table 1: ImageNet1K unstructured sparsity results using ResNet50 for 80% and 90% sparsity. Averaged results over two runs. $*$ sign indicates estimated cost with an ERK sparse forward pass.

exploration. This training stage with Θ_K frozen can be formulated as:

$$\min_{\Theta_P} \frac{1}{K} \sum_{i=1}^{K} \ell(f(\Theta_P \cup \Theta_K; \mathbf{x}^i), \mathbf{y}^i). \tag{3}$$

Grow. During the previous stage, *Reactivate & Explore*, the importance scores for Θ_P have been computed. These solid importance scores are used to choose which connections in Θ_P are worth exploring. For each layer l, we grow the parameters given by ArgTopK$(\mathcal{I}(\Theta^l_P), n^l)$ and set the corresponding mask elements in $\mathcal{B}$ to 1. The sets Θ_K and Θ_P are also updated accordingly. Also notice that n^l is the same as Eqn.2 here so model sparsity remains the same after one prune-grow. We then cycle back to *Importance Estimation* for another LookAhead step.

4 Experiments

We next demonstrate the effectiveness of our approach across a comprehensive set of scenarios. We first focus on unstructured sparsity and compare our results to other approaches in the literature on ResNet50 (He et al., 2016) and WideResNet22-2 (Zagoruyko & Komodakis, 2016) for ImageNet (Deng et al., 2009) and CIFAR10 (Krizhevsky et al., 2009). Then, we focus on structured sparsity using MobileNet-V1 (Howard et al., 2017) and ResNet50 on ImageNet. In order to show the generability of LookAhead, we also include object detection results on PASCAl VOC (Everingham et al., 2010). Finally, we ablate our method and analyze the effectiveness of LookAhead growing strategy. We run the experiments on 8 Nvidia Tesla V100 GPUs for ImageNet and 1 GPU for CIFAR-10. We include more details in supplementary material.

4.1 Unstructured Sparsity

We first compare our results with several state-of-the-art methods on unstructured sparsity on ImageNet and CIFAR-10 datasets. We show results using different sparsity levels and distributions for models trained with a different number of epochs. In our experiments, we aim for a specific final FLOP budget given a sparsity level. The sparsity level of every layers is predefined and set to be either *Uniform* or *ERK*, and we *avoid redistributing* the sparsity across layers throughout training for a fair comparison with methods like RigL (Evci et al., 2020). In the former, the sparsity of each layer S^l is equal to the total sparsity S, i.e., $S = S^l$. In the latter, we use the *Erdős-Rényi-Kernel* (ERK) formulation (Mocanu et al., 2018; Evci et al., 2020) to set sparsity for each layer, which means higher sparsity is assigned to those layers with more parameters i.e., $S^{l_i} > S^{l_j}$ if $m^{l_i} > m^{l_j}$.

Results on ImageNet. For this experiment, we used ResNet50 (He et al., 2016) and set the update period of our approach as ($H = J = K = 150$). Table 1 compares the top-1 accuracy and the performance of LookAhead with prior works in the literature using different sparsity distributions, training lengths, and sparsity levels. As shown, our approach consistently outperforms all the other methods by a significant margin. For instance, compared to RigL (Evci et al., 2020) under 100 training epochs, our approach yields an improvement of 1.0% top-1 accuracy at both 80% and 90% uniform sparsity levels. The improvement is even more significant when using a non-uniform ERK sparsity distribution: 1.1% and 1.3% at 80% and 90% sparsity levels, respectively. Importantly, our approach at 80% sparsity level trained for 500 epochs yields a 0.6% top-1 accuracy improvement compared to the dense ResNet50 baseline. Compared with the latest work IP-FT (Wimmer et al.,

Method	Sparsity Distribution	Total Epochs	Sparsity Ratio 80% Top-1 Acc(%)↑	Sparsity Ratio 80% FLOPs($\times e^8$)↓	Sparsity Ratio 90% Top-1 Acc(%)↑	Sparsity Ratio 90% FLOPs($\times e^8$)↓
Dense (Zagoruyko & Komodakis, 2016)			94.6	3.2	94.6	3.2
Pruning (Gale et al., 2019)	Non-Uniform	250	93.5	0.5	93.3	1.1
Static	ERK	250	92.9	0.5	91.6	1.1
RigL (Evci et al., 2020)	ERK	250	93.5	0.5	92.9	1.1
LookAhead (Ours)	ERK	250	**93.8**	0.5	**93.6**	1.1
Static	ERK	500	93.2	0.5	91.8	1.1
RigL(Evci et al., 2020)	ERK	500	93.7	0.5	93.3	1.1
LookAhead (Ours)	ERK	500	**94.5**	0.5	**93.8**	1.1

Table 2: CIFAR-10 unstructured sparsity results using WideResNet22-2 for 80% and 90% sparsity. Averaged results over three runs.

Method	Top-1 Acc(%)↑	Top-5 Acc(%)↑	FLOPs($\times e^9$)↓	FPS(im/s)↑	EPOCHS	Train FLOPs($\times e^{18}$)↓	Pretrained
Dense (Li et al., 2020)	77.2	92.9	4.1	1019	90	$\times 1$ (w.r.t.1.6)	–
EagleEye-2G(Li et al., 2020)	76.4	92.9	2.1	1471	$90 + 120$	$\times 1.68$	✓
GReg-2(Wang et al., 2021)	75.4	–	1.8	1414	$90 + 90$	$\times$**1.44**	✓
SCOP(Tang et al., 2020)	76.0		2.2	–	$90 + 140$	$\times 1.83$	✓
GBN(You et al., 2019)	76.2	92.8	2.4	–	$90 + 260$	$\times 2.69$	✓
HALP-55%(Shen et al., 2022b)	76.6	93.2	2.1	**1672**	$90 + 90$	$\times 1.51$	✓
LookAhead-55%-Pretrained	**77.0**	**93.2**	2.0	1554	$90 + 130$	$\times 1.71$	✓
DSA(Ning et al., 2020)	74.7	92.1	2.0	–	120	$\times 1.11$	✗
SCS(Yuan et al., 2021)	75.2	–	2.1	–	120	$\times 1.12$	✗
TAS(Dong & Yang, 2019)	76.2	93.1	2.3	–	240	$\times 1.50^*$	✗
LookAhead-55% (Ours)	**76.6**	**93.1**	2.1	**1654**	130	$\times$**0.71**	✗
EagleEye-1G(Li et al., 2020)	74.2	91.8	1.0	2429	$90 + 120$	$\times$**1.33**	✓
GReg-2(Wang et al., 2021)	73.9	–	1.3	1514	$90 + 90$	$\times 1.32$	✓
DSNet(Li et al., 2021)	74.6	–	1.2	–	$90 + 150$	$\times 1.49$	✓
Polarize(Zhuang et al., 2020)	74.2	–	1.2	–	$90 + 158$	$\times 1.51$	✓
HALP-30%(Shen et al., 2022b)	74.5	91.8	1.2	2597	$90 + 90$	$\times 1.29$	✓
LookAhead-30%-Pretrained	**74.8**	**92.2**	1.1	**2621**	$90 + 130$	$\times 1.39$	✓
MetaPruning(Liu et al., 2019)	73.4	–	1.0	2381	160	$\times 0.43^*$	✗
DMCP(Guo et al., 2020)	74.1	–	1.1	–	150	$\times 0.45^*$	✗
LookAhead-30% (Ours)	**74.6**	**92.1**	1.0	**2736**	130	$\times$**0.39**	✗

Table 3: ImageNet1K structured sparsity results using ResNet-50 for different pruning ratios. We use LookAhead-X% to refer to the percentage of parameters remaining in the model at the end of training. Averaged results over two runs. Detailed training flops calculation provided in supplementary material. "$*$" sign indicates lower-bound train cost for sparse training only for NAS-based methods.

2022) and DCIL (Kim et al., 2021), LookAhead achieves better or comparable accuracy with much less training cost needed as also demonstrated in Figure 1 (left).

Results on CIFAR10. We additionally evaluate our approach using WideResNet22-2 on CIFAR-10. In this case, we set the update periods to 65 and 150 for 250 and 500 training epochs, respectively. We detail the rest of the experimental settings in the supplementary material. As shown in Table 2, our approach outperforms the existing approaches while using only half the training time. In addition, as also happened for ImageNet, our approach with 80% sparsity and 500 training epochs achieves competitive performance compared to the dense model baseline. The results show the efficacy of LookAhead extends to small dataset as well, performing much better than other random exploration strategies like RigL (Evci et al., 2020).

4.2 Structured Sparsity

We next move to the more challenging structured sparsity task, where we focus on latency-aware structured sparsity (Shen et al., 2022b). To this end, we modify the LookAhead *Prune-Grow* stages to consider network channels rather than single parameters. We follow HALP (Shen et al., 2022b) but impose the dynamic regime of LookAhead into the knapsack process. First, we formulate the pruning step as a global cost-constraint importance maximization problem, where we take into account the latency benefits incurred every time we remove a channel from one of the layers of the network. Similarly, the growing step can also be formulated as a cost-constraint importance maximization problem where, in this case, we consider the latency increase for every additional channel added to a layer of the network. We provide the detailed formulation, and the experimental setup used in our experiments in the supplementary material.

Most structured sparsity approaches in the literature start from a pretrained model. Whereas in our approach, we start training the model from scratch and use an exponential scheduler to gradually increase the pruning ratio until we reach the target constraint. This leads to a significant reduction in the training costs. For additional insights and comprehensive comparison, we also consider the case where we start from a pretrained model in our experiments. In addition, we compare with prior art on resource or hardware-aware pruning covering a wider range of algorithms like (Li et al., 2020; Liu et al., 2019; Shen et al., 2022b), standard channel pruning methods like (Tang et al., 2020; Zhuang et al., 2020), the structured dynamic sparse training method SCS (Yuan et al., 2021), and

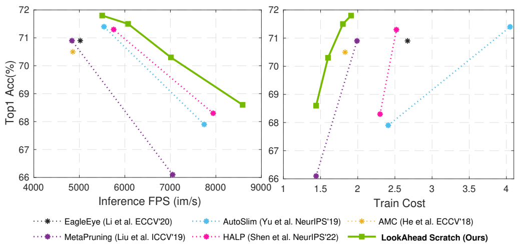

Figure 3: MobileNet-V1 on ImageNet1K structured sparsity results as a function of frames per second (left, top-right is better) and train cost (right, top-left is better).

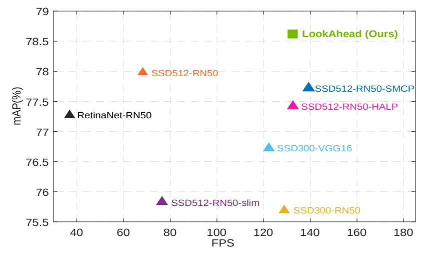

Figure 4: The mAP and inference FPS trade-off for pruning SSD512-RN50 on PASCAL VOC dataset.

results for AutoML and NAS-based methods such as AutoSlim (Yu & Huang, 2019) and AMC (He et al., 2018b).

Results on ImageNet. We show results for two different architectures: ResNet50 and MobileNet-V1 on ImageNet dataset. Results for these two experiments are shown in Table 3 and Figure 3. Our approach consistently outperforms all the referenced methods, including the very recently proposed state-of-the-art latency pruning method HALP (Shen et al., 2022b). For ResNet50, compared to other dynamic sparse training methods such as SCS (Yuan et al., 2021) our approach yields up to 1.4% (76.7% v.s. 75.2%) accuracy improvement with a $2.1G$-FLOPs model. We can observe similar patterns for the more efficient architecture MobileNet-V1, with LookAhead obtaining much more superior latency-accuracy tradeoff as clearly shown in Figure 3. Moreover, our approach trains networks from scratch yielding significant training cost saving compared with HALP which performs on a dense pretrained model, as shown in Figure 1 (right). We included detailed training FLOPs cost and epochs in the table to demonstrate the computation saving by LookAhead. We show the high efficacy of a dynamic sparsification method *in-training* is now made viable to offer similar or enhanced compact models compared to the literature.

Generalization to Object Detection We further demonstrate the generalizability of our approach to object detection task. The experimental settings and hyperparameters for this experiment are detailed in the supplemental material. We report our results of pruning a SSD512 (Liu et al., 2016) using a ResNet50 backbone on the popular PASCAL VOC dataset (Everingham et al., 2010) in Figure. 4. As shown, LookAhead clearly outperforms other methods in the literature with higher accuracy and/or a faster FPS, and even surpass the dense model while reducing the latency in half.

4.3 Ablation Studies

In this section, we perform ablations and conduct additional analysis to validate our design choices and provide observations for (i) the update period, (ii) freezing parameters (see *Reactivate & Explore* in Section3.1), (iii) the effect of the *Accuracy Improvement* stage, (iv) growing criterion of the neurons, and (v) initialization of the grown neurons. For these experiments, we use a ResNet50 with 90% ERK sparsity trained for 100 training epochs from scratch on ImageNet.

Sensitivity to the update period. We first study the effect of the update period H, J, K, which controls the balance between exploration and exploitation. A longer update period leads to more exploitation of the current selected sparse structure but fewer explorations. As shown in Table 5, we observe intuitive

Grow Criterion	Init	Freeze	*Accuracy Improvement*	Update Period H=J=K	Sparsity 90% ERK Top1 Acc(%)↑
Magnitude	Trained	✓	✓	100	73.9
Magnitude	Trained	✓	✓	150	74.3
Magnitude	Trained	✓	✓	200	74.0
Magnitude	Trained	✗	✓	150	73.6
Magnitude	Trained	✓	✗	150	73.4
Random	Trained	✓	✓	150	NaN
Magnitude	ZeroInit	✓	✓	150	74.0

Figure 5: Performance of our approach as a function of the update period, grown initialization, weight freezing, and inclusion of *Accuracy Improvement* stage. Results obtained using unstructured weight sparsity on Imagenet using ResNet50 with 90% ERK weight sparsity trained for 100 epochs.

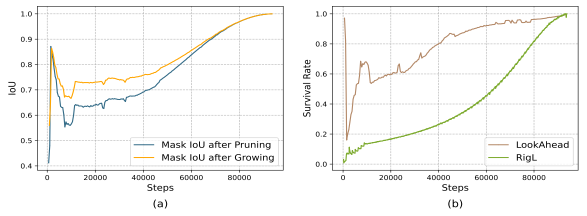

Figure 6: (a) Architecture convergence using mask IoU after pruning and growing; (b) Grown Neurons Survival Rate for our approach and RigL (Evci et al., 2020).

degradation in performance given
emphasis towards either end and observe 150 batches as a reliable amount.

Freezing Θ_K in *Reactivate & Explore* and inclusion of the *Accuracy Improvement* stage. We mentioned in Sec.3.1 that the inclusion of *Accuracy Improvement* stage and freezing currently selected architecture Θ_K in *Reactivate & Explore* are crucial to the performance with more thorough exploitation and stable exploration. The ablation results are demonstrated in Table 5. We observe a significant drop in performance when we do not enable these features. This demonstrates that explicitly forcing exploitation and exploration stages offers noticeable benefits.

Growing criterion and initialization. We further study the sensitivity of our algorithm to the growing criterion and the initialization of the grown neurons. As shown in Table 5, random growing the connections as in SET (Mocanu et al., 2018)(v.s. growing by magnitude criterion after *Reactivate & Explore*) leads to NaN with overflowed gradients after a few epochs. If we use ZeroInit, that is, initializing the grown neurons to 0 as in RigL (Evci et al., 2020), the performance also drops. We achieve the best results when we grow neurons using the weight magnitude criterion and initialize the grown ones using the trained values obtained after the *Reactivate & Explore* stage.

4.4 Discussions

Architecture convergence. We also analyze the convergence of the sparse architecture of our proposed method. To this end we report the mask IoU of the model mask $\mathcal{B}$ between two consecutive LookAhead update steps as training progresses. In particular we express the binary mask at the t–th LookAhead update step as $\mathcal{B}^t$ and measure the mask IoU between the t–th and $(t+1)$–th LookAhead step as IoU $= \frac{|\mathcal{B}^t \cap \mathcal{B}^{t+1}|}{|\mathcal{B}^t \cup \mathcal{B}^{t+1}|}$. $\mathcal{B}^t$ can either be the binary mask immediately after pruning, denoted as $\mathcal{B}^t_p$, or immediately after growing, denoted as $\mathcal{B}^t_g$.

Figure 6(a) shows the evolution of the mask IoU as the training progresses as a proxy for architecture convergence. As we can see, for both pruning and growing mask IoU there is a clear convergence towards IoU $= 1$, indicating that the discovered sparse architecture becomes more stable towards the end of the training process.

Effectiveness of the exploration strategy. To demonstrate the effectiveness of LookAhead exploration strategy compared to the previous random and greedy exploration approaches, RigL(Evci et al., 2020) as the representative, we propose a new metric named as the neuron growth ***survival rate***. The main idea is to gauge the fraction of newly grown neurons that is still active after the subsequent pruning step, indicating its reliability and usability once grown, joint with a side benefit to hint on architectural stability. This quantifiable metric is formally defined as $\frac{|(\mathcal{B}^t_g - \mathcal{B}^t_p) \cap \mathcal{B}^{t+1}_p|}{|\mathcal{B}^t_g - \mathcal{B}^t_p|}$. Intuitively, a high survival rate suggests an effective growth step as those newly grown parameters persist in the architecture for future training. As shown in Figure 6(b), the survival rate of our approach is significantly higher than RigL (Evci et al., 2020) for all exploration steps, demonstrating the effectiveness of our solid growing strategy.

Limitations As demonstrated in Sec.3, LookAhead requires additional hyperparameters like H, J, K which may need tuning effort for different datasets. Furthermore, our exploration step *Reactivate&Explore* introduce additional small amount of overhead as demonstrated in our tables and explained in the supplementary material.

5 Conclusions

In this paper, we re-formulate network sparsification as an iterative exploitation-exploration process to dynamically configure the network sparse structure through training. We introduce a novel *LookAhead* prune-grow scheme to balance both kept and removed parameters to discover enhanced sparse architecture while directly addressing the random and greedy nature of previous exploration solutions. We conduct extensive experiments with two different datasets and three model architectures on both unstructrued and structrued sparsity and demonstrate our method beats prior arts by a clear margin in various configurations. We also conduct a rigorous ablation study and design metrics to directly evaluate the effectiveness of our growing criterion.

REFERENCES

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

Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. *ICLR*, 2018.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *NeurIPS*, 33:1877–1901, 2020.

Changan Chen, Frederick Tung, Naveen Vedula, and Greg Mori. Constraint-aware deep neural network compression. In *ECCV*, pp. 400–415, 2018.

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.

Ting-Wu Chin, Ruizhou Ding, Cha Zhang, and Diana Marculescu. Towards efficient model compression via learned global ranking. In *CVPR*, pp. 1518–1528, 2020.

Xiaoliang Dai, Hongxu Yin, and Niraj K Jha. Nest: A neural network synthesis tool based on a grow-and-prune paradigm. *IEEE Transactions on Computers*, 68(10):1487–1497, 2019.

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. *ICLR*, 2021.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pp. 248–255. Ieee, 2009.

Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

Xuanyi Dong and Yi Yang. Network pruning via transformable architecture search. *NeurIPS*, 32, 2019.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *ICML*, pp. 2943–2952. PMLR, 2020.

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.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2018.

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

Shaopeng Guo, Yujie Wang, Quanquan Li, and Junjie Yan. Dmcp: Differentiable markov channel pruning for neural networks. In *CVPR*, pp. 1539–1547, 2020.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. 2015.

Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: Efficient inference engine on compressed deep neural network. *SIGARCH*, 44(3): 243–254, 2016.

Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. *NeurIPS*, 5, 1992.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pp. 770–778, 2016.

Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *IJCAI*, 2018a.

Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *CVPR*, pp. 4340–4349, 2019.

Yang He, Yuhang Ding, Ping Liu, Linchao Zhu, Hanwang Zhang, and Yi Yang. Learning filter pruning criteria for deep convolutional neural networks acceleration. In *CVPR*, pp. 2009–2018, 2020.

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 *ECCV*, pp. 784–800, 2018b.

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.

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*, pp. 7310–7311, 2017.

Ryan Humble, Maying Shen, Jorge Albericio Latorre, Eric Darve, and Jose Alvarez. Soft masking for cost-constrained channel pruning. In *European Conference on Computer Vision*, pp. 641–657. Springer, 2022.

Jangho Kim, Jayeon Yoo, Yeji Song, KiYoon Yoo, and Nojun Kwak. Dynamic collective intelligence learning: Finding efficient sparse model via refined gradients for pruned weights. *arXiv preprint arXiv:2109.04660*, 2021.

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

Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *ICML*, pp. 5544–5555. PMLR, 2020.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *NeurIPS*, pp. 598–605, 1990.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. Snip: Single-shot network pruning based on connection sensitivity. In *ICLR*, 2018.

Bailin Li, Bowen Wu, Jiang Su, and Guangrun Wang. Eagleeye: Fast sub-net evaluation for efficient neural network pruning. In *ECCV*, pp. 639–654, 2020.

Changlin Li, Guangrun Wang, Bing Wang, Xiaodan Liang, Zhihui Li, and Xiaojun Chang. Dynamic slimmable network. In *CVPR*, pp. 8607–8617, 2021.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *ICLR*, 2017.

Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *CVPR*, pp. 1529–1538, 2020a.

Shaohui Lin, Rongrong Ji, Yuchao Li, Yongjian Wu, Feiyue Huang, and Baochang Zhang. Accelerating convolutional networks via global & dynamic filter pruning. In *IJCAI*, volume 2, pp. 8. Stockholm, 2018.

Tao Lin, Sebastian U Stich, Luis Barba, Daniil Dmitriev, and Martin Jaggi. Dynamic model pruning with feedback. *ICLR*, 2020b.

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*, pp. 2980–2988, 2017.

Ning Liu, Geng Yuan, Zhengping Che, Xuan Shen, Xiaolong Ma, Qing Jin, Jian Ren, Jian Tang, Sijia Liu, and Yanzhi Wang. Lottery ticket preserves weight correlation: Is it desirable or not? In *ICML*, pp. 7011–7020. PMLR, 2021.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *ECCV*, pp. 21–37. Springer, 2016.

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 *ICCV*, pp. 3296–3305, 2019.

Sam Lobel, Omer Gottesman, Cameron Allen, Akhil Bagaria, and George Konidaris. Optimistic initialization for exploration in continuous control. In *AAAI*, volume 36, pp. 7612–7619, 2022.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. In *ICLR*, 2018.

Xiaolong Ma, Minghai Qin, Fei Sun, Zejiang Hou, Kun Yuan, Yi Xu, Yanzhi Wang, Yen-Kuang Chen, Rong Jin, and Yuan Xie. Effective model sparsification by scheduled grow-and-prune methods. In *ICLR*, 2021.

Marlos C Machado, Sriram Srinivasan, and Michael Bowling. Domain-independent optimistic initialization for reinforcement learning. In *AAAI*, 2015.

Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? *NeurIPS*, 32:14014–14024, 2019.

Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):1–12, 2018.

Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *ICML*, pp. 2498–2507. PMLR, 2017.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *CVPR*, pp. 11264–11272, 2019.

Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *ICML*, pp. 4646–4655. PMLR, 2019.

Sharan Narang, Erich Elsen, Gregory Diamos, and Shubho Sengupta. Exploring sparsity in recurrent neural networks. *ICLR*, 2017.

Xuefei Ning, Tianchen Zhao, Wenshuo Li, Peng Lei, Yu Wang, and Huazhong Yang. Dsa: More efficient budgeted pruning via differentiable sparsity allocation. In *ECCV*, pp. 592–607. Springer, 2020.

NVIDIA. Nvidia. convolutional networks for image classification in pytorch.

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. *NeurIPS Workshop*, 2017.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015.

Maying Shen, Hongxu Yin, Pavlo Molchanov, and Jose M Alvarez. When to prune? a policy towards early structural pruning. *CVPR*, 2022a.

Maying Shen, Hongxu Yin, Pavlo Molchanov, Lei Mao, Jianna Liu, and Jose Alvarez. Structural pruning via latency-saliency knapsack. In *Advances in Neural Information Processing Systems*, 2022b.

Suraj Srinivas, Akshayvarun Subramanya, and R Venkatesh Babu. Training sparse neural networks. In *CVPR workshops*, pp. 138–145, 2017.

Nikko Ström. Sparse connection and pruning in large dynamic artificial neural networks. In *Fifth European Conference on Speech Communication and Technology*. Citeseer, 1997.

Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *NeurIPS*, 33:6377–6389, 2020.

Yehui Tang, Yunhe Wang, Yixing Xu, Dacheng Tao, Chunjing Xu, Chao Xu, and Chang Xu. Scop: Scientific control for reliable neural network pruning. *NeurIPS*, 33:10936–10947, 2020.

Georg Thimm and Emile Fiesler. Evaluating pruning methods. In *ICANN*, pp. 20–25. Citeseer, 1995.

Joost van Amersfoort, Milad Alizadeh, Sebastian Farquhar, Nicholas Lane, and Yarin Gal. Single shot structured pruning before training. *arXiv preprint arXiv:2007.00389*, 2020.

Stijn Verdenius, Maarten Stol, and Patrick Forré. Pruning via iterative ranking of sensitivity statistics. *arXiv preprint arXiv:2006.00896*, 2020.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *ICLR*, 2019.

Huan Wang, Can Qin, Yulun Zhang, and Yun Fu. Neural pruning via growing regularization. In *ICLR*, 2021.

Paul Wimmer, Jens Mehnert, and Alexandru Condurache. Freezenet: Full performance by reduced storage costs. In *ACCV*, 2020.

Paul Wimmer, Jens Mehnert, and Alexandru Condurache. Interspace pruning: Using adaptive filter representations to improve training of sparse cnns. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12527–12537, 2022.

Mitchell Wortsman, Ali Farhadi, and Mohammad Rastegari. Discovering neural wirings. *NeurIPS*, 32, 2019.

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 *CVPR Workshops*, pp. 686–687, 2020.

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 *ECCV*, pp. 285–300, 2018.

Zhonghui You, Kun Yan, Jinmian Ye, Meng Ma, and Ping Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. *NeurIPS*, 32, 2019.

Jiahui Yu and Thomas Huang. Autoslim: Towards one-shot architecture search for channel numbers. *NeurIPS Workshop*, 2019.

Xin Yuan, Pedro Savarese, and Michael Maire. Growing efficient deep networks by structured continuous sparsification. *ICLR*, 2021.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*. British Machine Vision Association, 2016.

Xiao Zhou, Weizhong Zhang, Hang Xu, and Tong Zhang. Effective sparsification of neural networks with global sparsity constraint. In *CVPR*, pp. 3599–3608, 2021.

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

Tao Zhuang, Zhixuan Zhang, Yuheng Huang, Xiaoyi Zeng, Kai Shuang, and Xiang Li. Neuron-level structured pruning using polarization regularizer. *NeurIPS*, 33:9865–9877, 2020.

A APPENDIX

A.1 FORMULATION OF LOOKAHEAD WITH LATENCY-CONSTRAINED STRUCTURED SPARSITY

We now present LookAhead with latency-constrained structured sparsity setting. Specifically, we will highlight the different parts from our formulation with unstructured sparsity setting presented in Section 3.

A.1.1 NOTATION

For the neural network with L layers in total, we represent its parameters as $\Theta = \bigcup_{l=1}^{L} \Theta^l$, s.t. $\Theta^l \in \mathbb{R}^{C_{out}^l \times C_{in}^l \times K^l \times K^l}$. For the binary mask $\mathcal{B}$ indicating pruned and kept parameters, we now have $\mathcal{B} = \bigcup_{l=1}^{L} \mathcal{B}^l, \mathcal{B}^l \in \mathbb{R}^{C_{out}^l}, \mathcal{B}^l \in \{0,1\}^{C_{out}^l}$. , where C_{out}^l represents the number of output channels of layer l. We briefly denote C_{out}^l as m^l for simplicity. Unlike the element-wise multiplication of model parameters and corresponding binary mask, we now represent the model masking producing sparse model weights as follows:

$$\Theta \circledcirc \mathcal{B} = \bigcup_{l=1}^{L} \Theta^l \odot diag(\mathcal{B}^l), \tag{4}$$

where $diag$ is a diagonalization operator broadcasting $\mathcal{B}^l$ to the same shape as Θ^l. In PyTorch, this would simply be:

$$\Theta \circledcirc \mathcal{B} = \bigcup_{l=1}^{L} \Theta^l \odot \mathcal{B}^l.view(-1,1,1,1), \tag{5}$$

Moreover, Θ_K represents kept channels, expressed as $\Theta_K = \{\bigcup_{l=1}^{L} \bigcup_{i=1}^{m^l} \Theta_i^l; \mathcal{B}_i^l = 1\}$, and Θ_P represents pruned channels, expressed as $\Theta_P = \{\bigcup_{l=1}^{L} \bigcup_{i=1}^{m^l} \Theta_i^l; \mathcal{B}_i^l = 0\}$. Finally, we use p^l to define the active number of channels at layer l, expressed as $p^l = \|\mathcal{B}^l\|_0$.

A.1.2 RECAP OF HALP AND LATENCY-CONSTRAINED PRUNING

For our latency-constrained structured sparsification, we follow the latest resource-constrained pruning method HALP (Shen et al., 2022b) but impose the dynamic regime of LookAhead. Same as HALP, we formulate the pruning step as a global cost-constraint importance maximization problem, where we take into account the latency benefits incurred every time we remove a channel from one of the layers of the network. Similarly, we also formulate our unique growing part as a cost-constraint importance maximization problem. In this section, we will provide a brief recap of HALP and how it's used for the pruning step in our LookAhead iterative prune-and-grow setup. Given a global resource constraint C defining the maximum amount of resource we could use, HALP aims to find a sets of channels defining a sub-network achieving the best performance under the constraint C. In this case, C represents the inference latency for a target hardware platform. With the structured latency constraint, learning of the network sparsification (Eqn.1) now becomes:

$$\min_{\Theta,\mathcal{B}} \frac{1}{N} \sum_{i=1}^{\frac{N}{|\mathbf{B}|}} \ell(f(\Theta \circledcirc \mathcal{B}; \mathbf{x}^i), \mathbf{y}^i), \tag{6}$$
$$\text{s.t.} \sum_{l=1}^{L} \mathcal{T}^l(p^{l-1}, p^l) \leq C,$$

where $\mathcal{T}^l(p^{l-1}, p^l)$ defines the layer latency at layer l with p^{l-1} active input channels and p^l active output channels. In order to obtain the layer latency $T^l(p^{l-1}, p^l)$, HALP uses a pre-built layer-wise look-up table recording the latency at certain channel number and kernel dimension configuration. With this latency look-up table, HALP associates a potential latency reduction value R_j^l to each jth channel of layer l, computed as follows:

$$R_j^l = T^l(p^{l-1}, j) - T^l(p^{l-1}, j-1), 1 \leq j \leq p^l \tag{7}$$

R_j^l estimates the potential latency saving if we prune the corresponding channel. Now, in order to estimate the performance of the selected sub-newtwork, HALP measures the importance score $\mathcal{I}_j^l$ for each jth channel of layer l. The importance score metric adopted here is Taylor importance (Molchanov et al., 2019), which is evaluated as follows:

$$\mathcal{I}_j^l = |g_{\gamma_j^l} \gamma_j^l + g_{\beta_j^l} \beta_j^l|, \tag{8}$$

where γ and β are the BatchNorm layer's weight and bias. With R and $\mathcal{I}$ calculated, HALP formulates the channel pruning as a Knapsack problem where we try to maximize the total importance but under the latency constraint C:

$$\max \sum_{l=1}^{L} \sum_{j=1}^{p^l} I_j^l, \ \text{ s.t. } \sum_{l=1}^{L} \sum_{j=1}^{p^l} R_j^l \leq C, \ 0 \leq p^l \leq m^l, I_1^l \geq I_2^l \geq \dots I_{p^l}^l \tag{9}$$

Notice here the ranking of channels by the importance, in practice, HALP rank channels globally by importance and then consider their latency contribution. Concretely, if we prune the least important channel at layer l, the number of active channels will change from p^l to $p^l - 1$, leading to a latency reduction $R_{p^l}^l$ assigned as this channel's importance score. For solving Eqn.9, HALP developed an augmented Knapsack solver $Knapsack(V, W, C)$, where V and W are lists of values and weights for each item and C is the global resource contraint. $Knapsack(V, W, C)$ returns the items achieving maximum value while the accumulated weight is below the global constraint C.

A.1.3 Latency-Constrained Growing

Each LookAhead update step consists of alternative prune-and-grow to fully explore the sparse architecture. With a structured latency-constrained setting, during growing, we also want to take the model latency into account to prevent some latency-costly channels getting added back. We now present a latency-constrained growing step based on the Knapsack scheme developed in HALP (Shen et al., 2022b) and summarized in Sec.A.1.2. Similarly, we first use the latency look-up table to associate a potential latency addition value A_j^l to each jth channel of layer l, computed as follows:

$$A_j^l = T^l(p^{l-1}, j) - T^l(p^{l-1}, j-1), (p^l + 1) \leq j \leq m^l \tag{10}$$

A_j^l estimates the potential latency increase if we grow the corresponding channel. We then estimate the importance of the grown channel $\mathcal{I}$ similarly using a Taylor importance metric. With A and $\mathcal{I}$ calculated, we also treat the channel growing as a Knapsack problem to maximize the regrown importance but under the assigned growing latency budget G:

$$\max \sum_{l=1}^{L} \sum_{j=p^l+1}^{p^l+g^l} I_j^l, \ \text{ s.t. } \sum_{l=1}^{L} \sum_{j=p^l+1}^{p^l+g^l} A_j^l \leq G, \ 0 \leq p^l + g^l \leq m^l, I_{p^l+1}^l \geq I_{p^l+2}^l \geq \dots I_{p^l+g^l}^l$$

Here, g^l would be the number we choose to grow back for layer l. Similarly, we impose a ranking on the channels based on the importance for channel latency assignment. During growing, if we grow the most important channel from Θ_P, the number of active channels will change from p^l to $p^l + 1$, leading to a latency addition $A^l_{p^l}$ assigned as this channel's importance score. The augmented Knapsack solver $Knapsack(V, W, C)$ is also used here to solve this constrained optimization problem.

A.1.4 Update Schedule

Given C as our final targeted latency, we gradually decrease the total latency of the model using exponential scheduler (de Jorge et al., 2021) similar to HALP (Shen et al., 2022b). Suppose the total number of update steps is T, we have the latency target at each step t as $C^1 > C^2 > \cdots > C^T = C$. We also assign a latency budget to the model at each update step to grow an amount of connections. The grown latency budget at update step t could also be determined by an exponential scheduler or cosine annealing scheduler. However, we notice that a latency budget given by $G^t = \alpha \cdot (C^t - C^{t-1})$ yields good performance. In practice, we set $\alpha = 0.75$.

A.1.5 Procedure

To recap, in the prune step, we apply the Knapsack latency-constrained pruning as described in Sec.A.1.2 on the kept parameters Θ_K; and in the grow step, we apply the Knapsack latency-constrained growing as described in Sec.A.1.3 on the removed parameters Θ_P. We use the scheduled $\{C^1, \ldots, C^T\}$ and $\{G^1, \ldots, G^T\}$ to control the amount we prune and grow latency-wise at each LookAhead step. The other parts and the overall procedure is the same as the scheme we present in Sec.3 for the unstructured sparsity setting.

We provide an algorithmic description of the latency-constrained LookAhead dynamic sparse training scheme in Algo.2 for completeness.

A.2 Detailed Experiment Hyperparameter and Optimization Settings

The large-scale image classification dataset ImageNet (Deng et al., 2009) is of version ILSVRC2012 (Russakovsky et al., 2015), which consists of $1.3M$ images of 1000 classes. We run all experiments on ImageNet and PASCAL VOC with eight NVIDIA Tesla V100 GPUs. Experiments on CIFAR10 (Krizhevsky et al., 2009) are conducted with a single NVIDIA Tesla V100 GPU. All experiments are conducted with PyTorch (Paszke et al., 2017) V1.4.0. For experiments that require pretrained model weights, we take the ones provided by the official PyTorch model zoo.

A.2.1 Unstructured Weight Sparsity on ResNet50-ImageNet

We use an individual batch size of 128 per GPU and follow NVIDIA's recipe (NVIDIA) with mixed precision and Distributed Data Parallel training. The learning rate is warmed up linearly in the first 8 epochs reaching its highest learning rate then follows a cosine decay (Loshchilov & Hutter, 2016) over the remaining epochs. While reproducing the results of RigL (Evci et al., 2020), similar to PC-GAP (Ma et al., 2021), we found that the original optimization settings provided by RigL does not yield stable results particularly in the longer training with 500 epochs. However, with our hyperparameters and optimization setting, we're able to reproduce RigL (Evci et al., 2020) and even obtain better results than theirs, ensuring a fair comparison. Moreover, in main paper, we mentioned that for unstructured weight sparsity experiments, we leveraged a cosine decay function expressed as $n^l = f_{\text{decay}}(t; \alpha, T) m^l (1 - S^l)$ to determine the number of neurons to update at each step. In practice, we set α as 0.3 and choose T to be the $3/4$ of the entire training duration. For example, if we train the model for 100 epochs, the LookAhead update step will repeat until the 75th epoch.

A.2.2 Unstructured Weight Sparsity on WideResNet22-2-CIFAR10

In our experiments section, we also include results of WideResNet22-2, which is Wide Residual Network (Zagoruyko & Komodakis, 2016) with 22 layers using a width multiplier of 2. We use an individual batch size of 128, an initial learning rate of 0.1 decaying by a factor of 5 every 30000 iterations, an L2 regularization coefficient of $5e-4$, and a SGD momentum of 0.9. Similarly, results of RigL are reproduced using the same hyperparameters and optimization settings as ours, ensuring a fair comparison. Choices of α and T are similar to the above ImageNet settings.

A.2.3 Latency Constrained Structured Sparsity

ImageNet We follow HALP (Shen et al., 2022b) for setting the hyperparameters and optimization settings of experiments on latency constrained structured sparsity with ResNet50 and MobileNet-V1. They are also similar to the recipe described in A.2.1. We set T, the total number of updates, as 45 and H, J, K as 50. Total number of training epochs is set to 110. We also follow HALP (Shen et al., 2022b) for constructing the latency lookup table, which is pre-generated targetting the NVIDIA TITAN V GPU inference by iteratively reducing the number of channels in a layer and characterize the corresponding latency with NVIDIA cuDNN (Chetlur et al., 2014) V7.6.5. The latency measurement is conducted 100 times to avoid randomness. We also refer to HALP for some special implementation detail such as how to deal with the group convolution in MobileNet-V1, negative latency contribution, pruning of the first model layer, which are all described in detail in HALP.

PASCAL VOC We follow the "07 + 12" setting as in (Liu et al., 2016) and use the union of VOC2007 and VOC2012 trainval as our training set and VOC2007 test as test set. Our SSD model, similar to HALP (Shen et al., 2022b), is based on (Liu et al., 2016). Following (Huang et al., 2017), for efficiency, we remove the last stage of convolution layers, last avgpool, and fc layers from the original ResNet50 classification structure. Also, all strides in the third stage of ResNet50 layer are set to 1×1. We train our models for 900 epochs with SGD optimizer and learning rate schedule same as (Shen et al., 2022b) with an initial learning rate of $8e-3$ which warms up in the first 50 epochs then decays by $3/8, 1/3, 2/5, 1/10$ at the $700, 800, 840, 870$th epoch

Algorithm 2 LookAhead Pseudocode with Latency-Constrained Structured Sparsity

```
Input: $\Theta$, $\mathcal{D}$, $T$, $\{C^1, C^2, \ldots, C^T\}$, $\{G^1, G^2, \ldots, G^T\}$, $H$, $J$, $K$
1:  $\Theta_K, \Theta_P, \mathcal{B} \leftarrow \Theta, \{\}, \mathbb{1}$
2:  $\Delta T \leftarrow H + J + K$
3:  $t, flag \leftarrow 0, 0$
4:  for $i \leftarrow 1$ to $|\mathcal{D}|$ do
5:      if $(i + J + K) \mod \Delta T = 0$ and $t < T$ then
6:          $\mathcal{I}_K \leftarrow= \{\bigcup_{l=1}^{L} \bigcup_{i=1}^{m^l} \mathcal{I}_i^l; \Theta_i^l \in \Theta_K\}$
7:          $R \leftarrow= \{\bigcup_{l=1}^{L} \bigcup_{i=0}^{p^l} R_i^l\}$
8:          $\mathbb{I}_p \leftarrow Knapsack(I_K, R, C^t)$
9:          //Prune Connections
10:         Update $\Theta_K, \Theta_P, \mathcal{B}$ with $\mathbb{I}_p$
11:     else if $(i + K) \mod \Delta T = 0$ and $t < T$ then
12:         $flag \leftarrow 1$
13:     else if $i \mod \Delta T = 0$ and $t < T$ then
14:         $\mathcal{I}_P \leftarrow= \{\bigcup_{l=1}^{L} \bigcup_{i=1}^{m^l} \mathcal{I}_i^l; \Theta_i^l \in \Theta_P\}$
15:         $A \leftarrow= \{\bigcup_{l=1}^{L} \bigcup_{i=p^l+1}^{m^l} A_i^l\}$
16:         $\mathbb{I}_p \leftarrow Knapsack(I_P, A, G^t)$
17:         //Grow Connections
18:         Update $\Theta_K, \Theta_P, \mathcal{B}$ with $\mathbb{I}_g$
19:         $t \leftarrow t + 1$
20:         $flag \leftarrow 0$
21:     end if
22:     if flag then
23:         //Reactivate & Explore
24:         $\ell_i \leftarrow \ell(f(\Theta_P \cup \Theta_K; x_i), y_i)$
25:         $\Theta_P \leftarrow lr \cdot \nabla_{\Theta_P} l_i$
26:     else
27:         //Importance Estimation
28:         //Accuracy Improvement
29:         $\ell_i \leftarrow \ell(f(\Theta \odot \mathcal{B}; x_i), y_i)$
30:         $\Theta_K \leftarrow lr \cdot \nabla_{\Theta_K} l_i$
31:     end if
32: end for
```

A.3 Magnitude Rank Variation

Within one LookAhead step, we included three training stages, namely *Importance Estimation, Accuracy Improvement, Reactivate Explore*. To further analyze and understand LookAhead, we track and visualize the magnitude rank variation of the removed neurons in the prune step across one LookAhead update step. We conduct this study with the ResNet50-ImageNet 90% ERK unstructured weight sparsity setting. Concretely, same as 4.4, we denote the binary mask immediately after pruning at the tth update step as $\mathcal{B}_p^t$ and the binary mask immediately after growing at the tth update step as $\mathcal{B}_g^t$. We express the active neurons at the tth update step before pruning as:

$$\Theta_\alpha^t = \{\Theta_i; \mathcal{B}_{gi}^{t-1} = 1\} \tag{11}$$

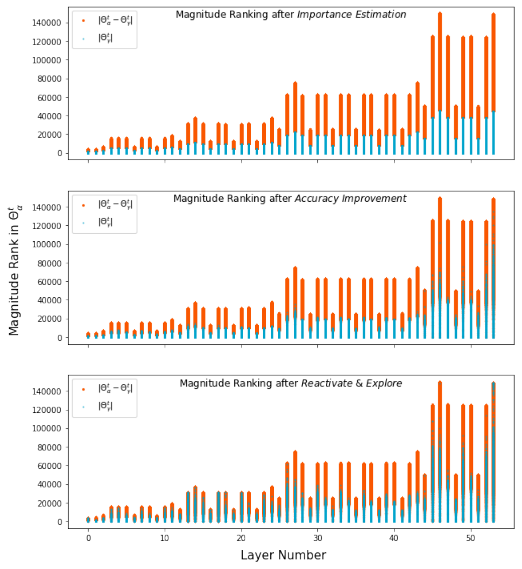

Figure 7: Magnitude rank of $|\Theta_\gamma^t|$ and $|\Theta_\alpha^t - \Theta_\gamma^t|$ among $|\Theta_\alpha^t|$ after *Importance Estimation*, *Accuracy Improvement*, and *Reactivate & Explore* respectively. Some of the neurons originally lower ranked in magnitude and pruned get updated significantly and higher ranked through one LookAhead update step.

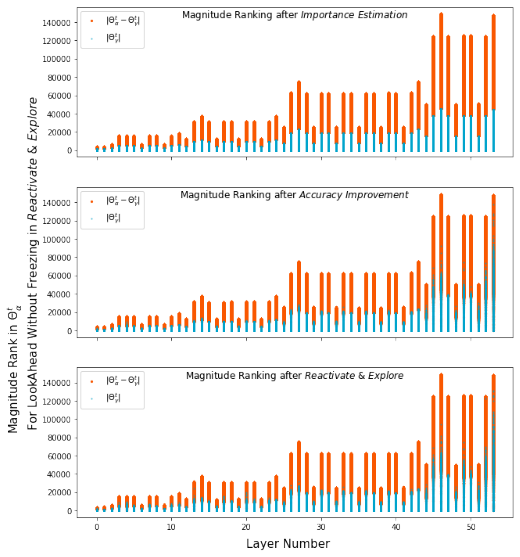

Figure 8: LookAhead without freezing in *Reactivate & Explore*. Magnitude rank of $|\Theta_\gamma^t|$ and $|\Theta_\alpha^t - \Theta_\gamma^t|$ among $|\Theta_\alpha^t|$ after *Importance Estimation*, *Accuracy Improvement*, and *Reactivate & Explore* respectively. Compared to Fig.7, magnitude of pruned neurons get much less updated through one LookAhead update step, illustraing the effectiveness of our freezing Θ_K.

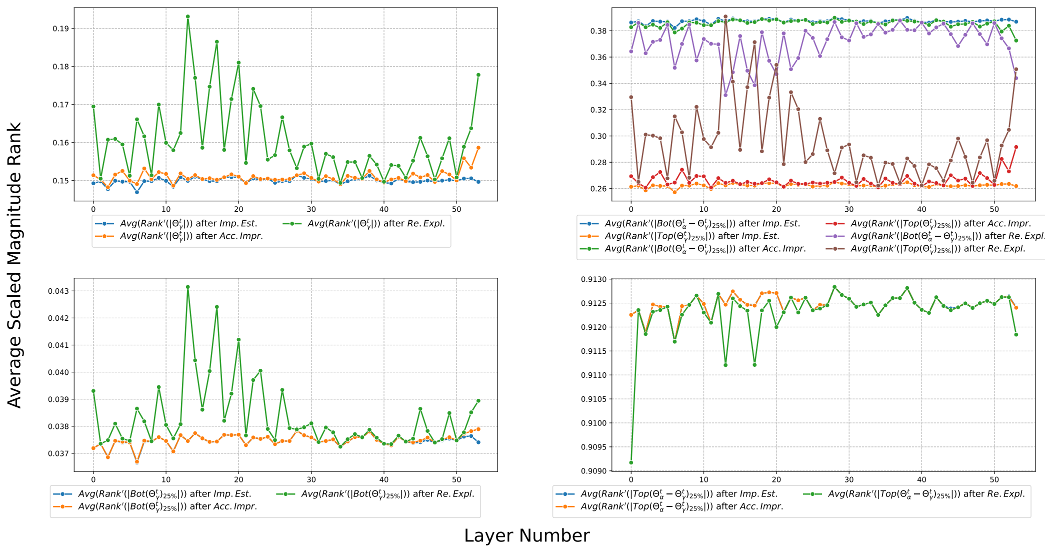

Figure 9: Variation of average magnitude ranking of $|\Theta_\gamma^t|$ and $|\Theta_\alpha^t - \Theta_\gamma^t|$ among $|\Theta_\alpha^t|$ after *Importance Estimation*(*Imp.Est.*), *Accuracy Improvement*(*Acc.Impr.*), and *Reactivate & Explore*(*Re.Expl.*) for different portions respectively.

Moreover, the newly pruned neurons at the tth update step Θ_γ^t can be expressed as:

$$\mathcal{B}_\gamma^t = \mathcal{B}_g^{t-1} - \mathcal{B}_p^t \tag{12}$$

$$\Theta_\gamma^t = \{\Theta_i; \mathcal{B}_{\gamma i}^t = 1\} \tag{13}$$

Obviously, we have $\Theta_\gamma^t \subset \Theta_\alpha^t$. The set of neurons that survive the *Prune* stage of the tth LookAhead update step can thus be expressed as $\Theta_\alpha^t - \Theta_\gamma^t$. Within one LookAhead update step which consists of

Importance Estimation, Prune, *Accuracy Improvement*, *Reactivate & Explore*, and Grow, we track the magnitude of the pruned $|\Theta_\gamma^t|$ and the survived portions $|\Theta_\alpha^t - \Theta_\gamma^t|$ respectively after each one of the three train sessions (*Importance Estimation*, *Accuracy Improvement*, *Reactivate & Explore*). In other words, we sorted and ranked all neurons in Θ_α^t and inspected how those newly pruned neurons Θ_γ^t change in magnitude by *Accuracy Improvement* and *Reactivate & Explore* compared with those survived neurons $\Theta_\alpha^t - \Theta_\gamma^t$. Visualization is provided in Fig. 7. As expected, we see Θ_γ^t get ranked in the bottom after *Importance Estimation* since our pruning importance score is directly the magnitude of neurons. Then, after *Accuracy Improvement* when only the kept neurons $\Theta_\alpha^t - \Theta_\gamma^t$ get trained, the magnitude rank of $|\Theta_\gamma^t|$ and $|\Theta_\alpha^t - \Theta_\gamma^t|$ get barely altered as visualized in the middle subplot. Finally, after *Reactivate & Explore* when we freeze $\Theta_\alpha^t - \Theta_\gamma^t$ and only train the pruned sub-network containing Θ_γ^t for exploration, we see quite a few neurons in Θ_γ^t get much higher ranked and even surpass those in $\Theta_\alpha^t - \Theta_\gamma^t$. The Grow step following *Reactivate & Explore* will then bring those neurons back to the model. To further understand the influence of freezing we conduct in *Reactivate & Explore*, we also show the magnitude rank variation diagram of LookAhead without freezing in Fig. 8. We can see that the magnitude rank of Θ_γ^t get barely changed, and Θ_γ^t remains ranked in the bottom after *Reactivate & Explore*.

To gain more understanding of which portion of Θ_γ^t and $\Theta_\alpha^t - \Theta_\gamma^t$ get updated significantly by LookAhead, we calculated and visualize the average magnitude rank of Θ_γ^t in different portions. First, we use $Top(\Theta_\gamma^t)_{25\%}$ to denote the subset of neurons originally ranked in the top 25% of Θ_γ^t **after *Importance Estimation***. Similarly, we use $Bot(\Theta_\gamma^t)_{25\%}$ to denote the bottom 25% of Θ_γ^t. $Top(\Theta_\alpha^t - \Theta_\gamma^t)_{25\%}$ and $Bot(\Theta_\alpha^t - \Theta_\gamma^t)_{25\%}$ thus denote the top and bottom 25% portion of $\Theta_\alpha^t - \Theta_\gamma^t$ after *Importance Estimation*. Moreover, we use $Avg(Rank(|\Theta_\gamma^t|))$ to denote the average magnitude rank of Θ_γ^t. Since the number of neurons differs a lot for each layer, we scale the rank by the inverse of number of active parameters of each layer. Specifically, for each layer l, we compute $Rank'(|\Theta_\gamma^{tl}|) = Rank(|\Theta_\gamma^{tl}|)/\|\Theta_\alpha^{tl}\|_0$. The visualization is provided in Fig. 9. From the figure, we could see that the magnitude ranks that vary the most within one LookAhead update step are mostly in $Top(\Theta_\gamma^t)_{25\%}$ and $Bot(\Theta_\alpha^t - \Theta_\gamma^t)_{25\%}$. At some layer, *Reactivate & Explore* increase the average scaled magnitude rank of $Top(\Theta_\gamma^t)_{25\%}$ by 13% from 26% to 39%. On the other hand, the neurons which originally already rank pretty high $Top(\Theta_\alpha^t - \Theta_\gamma^t)_{25\%}$ or low in magnitude $Bot(\Theta_\gamma^t)_{25\%}$ after *Importance Estimation* barely get their values changed much as visualized. The maximum of increase in the average scaled magnitude rank of $Bot(\Theta_\gamma^t)_{25\%}$ by *Reactivate & Explore* is around 0.005, much smaller compared to the change in $Top(\Theta_\gamma^t)_{25\%}$.

A.4 Training FLOPs Cost Computation

In tables presented in the paper, we demonstrate the training cost of LookAhead as well as other methods. FLOPs needed for a single forward pass inference of sparse model is computed by counting the total number of multiplications and additions. However, during training, the FLOPs computation would be slightly different due to different usage of the back-propagation gradients. In summary, training a neural network consists of 2 main steps which are *forward pass* and the *backward pass*. During the *forward pass*, we calculate the loss of the given batch of data using the current set of model parameters. Activations of each layer are stored in memory for the following backward pass. During the *backward pass*, we use the loss value as the initial error signal and back-propagate the error signal to calculate the gradients of parameters. We calculate respectively the gradient of the activations of the previous layer and the gradient of its parameters. Roughly, the FLOPs needed for backward pass will be ***twice*** the FLOPs needed for forward pass. Suppose a given dense architecture has forward pass FLOPs represented as ζ_D and its pruned or sparsified model has FLOPs ζ_P. Training a sample with dense model can be expressed as $3 \cdot \zeta_D$.

LookAhead Each LookAhead step consists of three training stages, namely: *Importance Estimation*, *Accuracy Improvement*, and *Reactivate & Explore*. For each *Importance Estimation* and *Accuracy Improvement*, we need $3 \times \zeta_P$ FLOPs for both sparse forward and backward pass. For *Reactivate & Explore*, since we are training with temporarily reactivated Θ_P, we need $2 \times \zeta_P + \zeta_D$ FLOPs to take care of the dense forward pass. We still use sparse gradients for updating due to the frozen Θ_K. After the entire update period, the FLOPs needed would simply be $3 \times \zeta_P$. Since the update period ends at $3/4$ of the entire training epochs, the average training cost can be calculated as:

$$\frac{3}{4} \cdot \frac{(H+J) \cdot 3 \cdot \zeta_P + K \cdot (2 \cdot \zeta_P + \zeta_D)}{H+J+K} + \frac{1}{4} \cdot 3 \cdot \zeta_P$$

With $H = J = K$, the cost would be:

$$\frac{11 \cdot \zeta_P + \zeta_D}{4}$$

This would be slightly higher than completely training a sparse model from scratch which is $3 \cdot \zeta_P$ but still substantially lower than dense model training cost ($3 \cdot \zeta_D$).

Also notice that, according to our above description of LookAhead with structured sparsity, we follow the exponential scheduler of HALP (Shen et al., 2022b), and the update period ends much earlier than $3/4$ of the total training epochs. The update with LookAhead for latency-constrained structured sparsity will instead end at $45 \times (50 \times 3) = 6750$, which roughly corresponds to the 5th epoch. The average training cost of LookAhead will also be much lower. With 130 training epochs in total, according to the calculation we provide above, it will instead be:

$$\frac{5}{130} \cdot \frac{(H+J) \cdot 3 \cdot \zeta_P + K \cdot (2 \cdot \zeta_P + \zeta_D)}{H+J+K} + \frac{125}{130} \cdot 3 \cdot \zeta_P$$

With $H = J = K$, the cost would approximately be:

$$\frac{388.3 \cdot \zeta_P + 1.7 \cdot \zeta_D}{130}$$

SOFT MASKING Now for the family of soft masking methods like SNFS (Dettmers & Zettlemoyer, 2019), DPF (Lin et al., 2020b), and DCIL (Kim et al., 2021), training cost vary based on different methods. Since these methods typically maintain dense gradients during backpropagation, training cost would usually be noticeably higher than typical sparse training approaches. For SNFS (Dettmers & Zettlemoyer, 2019), the total number of training FLOPs scales with $2 \cdot \zeta_P + \zeta_D$. For DCIL (Kim et al., 2021), the work requires two forward and backward passes each time to measure two sets of gradients(one with dense weight and one with sparse weight) for weights update, and the total number of training FLOPs scales with $5 \cdot \zeta_D + \zeta_P$, which is nearly doubled dense model training cost ($6 \cdot \zeta_D$).

ZERO-SHOT PRUNING For the family of static sparse training or zero-shot pruning, the cost can be expressed as $3 \cdot \zeta_P$.

PRUNING FROM PRETRAINED Most of the pruning from pretrained methods nowadays employed iterative pruning. For simplicity here, we estimate a *very loose theoretical lowerbound* with one-shot pruning and no further gradients calculation on the pruned parameters during finetuning. The training cost of pretrained dense model scales with $3 \cdot \zeta_D$ as discussed. In the later finetuning stage, the cost would scale with $3 \cdot \zeta_P$ since the model deals with a sparse model now.

RIGL (EVCI ET AL., 2020) For the representative state-of-the-art dynamic sparse training work RigL, iterations with no connections updates need $3 \cdot \zeta_P$ FLOPs. At every ΔT iteration, RigL calculates the dense gradients. The averaged FLOPs for RigL is given by $\frac{3 \cdot \zeta_P + 2 \cdot \zeta_P + \zeta_D}{\Delta T + 1}$.

INTERSPACE PRUNING For the very latest interspace pruning work (Wimmer et al., 2022), authors use FB convolution layers which introduce additional forward and backward overhead. Given the information provided in the paper, for a particular convolution layer with size $c_{out} \times c_{in} \times K \times K$, the relative increase of forward pass would be K^2/c_{out} times the dense forward pass. Notice that this is a constant overhead independent of the pruning rate and sparsity of the model. Similarly, the authors provide that the backward pass would introduce an additional constant overhead of K^2/c_{in} times the dense computation of gradients. Since authors provide no exact FLOPs of the model, we also estimate a lower bound of K^2/c_{out} and K^2/c_{in} as $3^2/128 \approx 0.07$ for ResNet-50. This is a lower bound since as identified in many works before the spatial size is the largest in the early layers with a large K and small c_{out} and c_{in} processing large-sized feature maps and dominating the overall FLOPs of the model. Now we could calculate the FLOPs needed to train a single example as $\zeta_P + 0.07 \cdot \zeta_D + 2 \cdot (\zeta_P + 0.07 \cdot \zeta_D)$ which is approximately $3 \cdot \zeta_P + 0.21 \cdot \zeta_D$.

NAS-BASED METHODS We also demonstrate the results of some NAS-based methods (Liu et al., 2019; Dong & Yang, 2019; Guo et al., 2020) in the main paper for comparison. Since the searching involved is very hard to quantify the training cost estimation, we only report the estimated training cost of the discovered pruned model ($3 \cdot \zeta_P$). Now notice that this is a very loose lower bound, and the actual cost could be much higher with the architecture search.

A.5 ABLATION ON CIFAR-10

METHOD	MAP↑	FLOPS($\times e^9$)↓	TRAIN FLOPS($\times e^{18}$)↓	FPS(BS=1)↑	FPS(BS=32)↑
SSD512-RN50, BASE MODEL	77.98	65.56	×1.00(w.r.t.2.60)	68.24	103.48
SSD512-RN50-SLIM	75.83	46.09	×0.70	76.49	114.80
SSD300-RN50	75.69	16.23	×0.25	128.85	309.32
SSD300-VGG16 Liu et al. (2016)	76.72	31.44	×0.48	122.28	262.93
RETINANET-RN50 Lin et al. (2017)	77.27	106.50	×1.62	36.92	-
SSD512-RN50-HALP Shen et al. (2022b)	77.42	15.38	×1.23	132.57	323.36
SSD512-RN50-SMCP Humble et al. (2022)	77.72	10.02	×1.75	139.30	-
LOOKAHEAD (OURS)	**78.62**	16.41	×0.29	132.46	318.46

Table 5: PASCAL VOC structured sparsity results on SSD512-RN50.

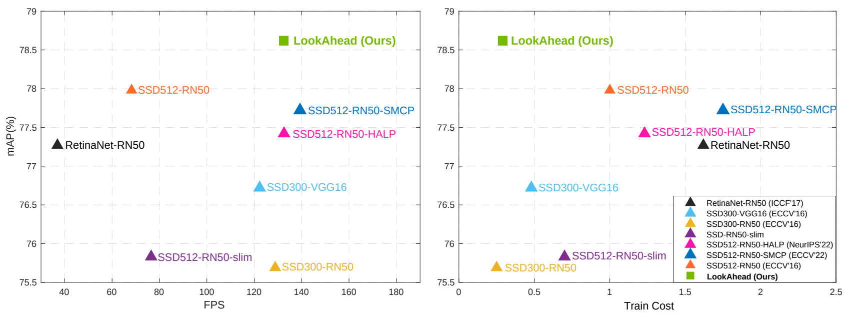

Figure 10: PASCAL VOC structured sparsity results on SSD512-RN50 as a function of frames per second (left figure, top-right is better) and train cost (right figure, top-left is better).

Update period is an important factor in our algorithm as it controls how much we exploit and explore. As mentioned in the main paper, the update period we use for WideResNet22-2 training on CIFAR-10 dataset is different for 250 and 500 epochs training. We demonstrate the ablation results on the update period ($H = J = K$) in Table 4. The results suggest that LookAhead with more training epochs enjoy longer update period.

Total Epochs	Update Period (H=J=K)	Top1 Acc(%)
250	50	93.1 ± 0.03
250	65	93.6 ± 0.02
250	80	93.2 ± 0.03
500	135	93.4 ± 0.03
500	150	93.8 ± 0.01
500	165	93.4 ± 0.02

Table 4: CIFAR-10 unstructured sparsity results using WideResNet-22 for 90% sparsity ratios. Ablation on LookAhead update period. Averaged results over three runs.

A.6 DETAILED RESULTS OF OBJECT DETECTION

As shown in Table 5, we demonstrate the detailed results of object detection on the PASCAL VOC Everingham et al. (2010) dataset compared with several strong baselines. Concretely, we compared with the base SSD model Liu et al. (2016) with different backbone architecture including our adopted ResNet-50 and VGG-16 without any pruning, RetinaNet Lin et al. (2017) for improved detection performance without pruning, and the same SSD model latency-pruned by HALP Shen et al. (2022b). The superiority of the proposed LookAhead method is clearly observed. We significantly surpassed the mAP achieved by the latest HALP pruned model (78.62 v.s.77.42) with comparable latency. Moreover, LookAhead is much better than the other methods in terms of training cost, indicating that LookAhead can achieve superior performance while also saving tremendous computation cost. The comparison of training cost is also presented in detail in Figure 10.