STRUCTURED PRUNING OF CNNS AT INITIALIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Pruning-at-initialization (PAI) methods can prune the individual weights of a convolutional neural network (CNN) before training, thus avoiding expensive fine-tuning or retraining of the pruned model. While PAI shows promising results in reducing model size, the pruned model still requires unstructured sparse matrix computation, making it difficult to achieve a real speedup. In this work, we show both theoretically and empirically that the accuracy of CNN models pruned by a PAI method depends on the *layer-wise density* (i.e., the fraction of the remaining parameters in each layer), irrespective of the granularity of pruning. We formulate PAI as a convex optimization problem based on an expectation-based proxy for model accuracy, which can produce the optimal allocation of the layer-wise densities with respect to the proxy model. Using our formulation, we further propose a structured and hardware-friendly PAI method, named PreCrop, to prune or reconfigure CNNs in the channel dimension. Our empirical results show that PreCrop achieves a higher accuracy than existing PAI methods on several popular CNN architectures, including ResNet, MobileNetV2, and EfficientNet, on both CIFAR-10 and ImageNet. Notably, PreCrop achieves an accuracy improvement of up to 2.7% over a state-of-the-art PAI algorithm when pruning MobileNetV2 on ImageNet. PreCrop also improves the accuracy of EfficientNetB0 by 0.3% on ImageNet with only 80% of the parameters and the same FLOPs.

1 INTRODUCTION

Convolutional neural networks (CNNs) have achieved state-of-the-art accuracy in a wide range of machine learning (ML) applications. However, the massive computational and memory requirements of CNNs remain a major barrier to more widespread deployment on resource-limited edge and mobile devices. This challenge has motivated a large and active body of research on CNN compression, which attempts to simplify the original model without significantly compromising the accuracy.

Weight pruning [15, 7, 17, 4, 8] has been extensively explored to reduce the computational and memory demands of CNNs. Existing methods create a sparse CNN model by iteratively removing ineffective weights/activations and training the resulting sparse model. Such an iterative pruning approach usually enjoys the least accuracy degradation but at the cost of a more computationally expensive training procedure. Moreover, training-based pruning methods introduce additional hyperparameters, such as the learning rate for fine-tuning and the number of epochs before rewinding [20], which make the pruning process even more complicated and less reproducible.

To minimize the cost of pruning, a new line of research proposes *pruning-at-initialization (PAI)* [16, 27, 24], which identifies and removes unimportant weights in a CNN before training. Similar to training-based pruning, PAI assigns an *importance score* to each individual weight and retains only a subset of them by maximizing the sum of the importance scores of all remaining weights. The compressed model is then trained using the same hyperparameters (e.g., learning rate and the number of epochs) as the baseline model. Thus, the pruning and training of CNNs are cleanly decoupled, greatly reducing the complexity of obtaining a pruned model. Currently, SynFlow [24] is considered the state-of-the-art PAI technique — it eliminates the need for data during pruning as required in prior arts [16, 27] and achieves a higher accuracy with the same compression ratio.

However, existing PAI methods mostly focus on fine-grained weight pruning, which removes individual weights from the CNN model without preserving any structures. As a result, both inference and training of the pruned model require sparse matrix computation, which is challenging to accelerate

on commercially-available ML hardware that is optimized for dense computation (e.g., GPUs and TPUs [14]). According to a recent study [6], even with the NVIDIA cuSPARSE library, one can only achieve a meaningful speedup for sparse matrix multiplications on GPUs when the sparsity is over 98%. In practice, it is difficult to compress modern CNNs by more than 50× without a drastic degradation in accuracy [2]. Therefore, structural pruning patterns (e.g., pruning weights for the entire output channel) are preferred to enable practical memory and computational saving by avoiding irregular sparse storage and computation.

In this work, we propose novel structured PAI techniques and demonstrate that they can achieve the same level of accuracy as the unstructured methods. We first introduce **synaptic expectation (SynExp)**, a new proxy metric for accuracy, which is defined to be the expected sum of the importance scores of all the individual weights in the network. SynExp is invariant to weight shuffling and reinitialization, thus addressing some of the deficiencies of the fine-grained PAI approaches found in recent studies [22, 5]. We also show that SynExp does not vary as long the layer-wise density remains the same, irrespective of the granularity of pruning. Based on this key observation, we formulate an optimization problem that maximizes SynExp to determine the layer-wise pruning ratios, subject to model size and/or FLOPs constraints. We then propose **PreCrop**, a structured PAI that prunes CNN models at the channel level in a way to achieve the target layer-wise density determined by the SynExp optimization. PreCrop can effectively reduce the model size and computational cost without loss of accuracy compared to existing fine-grained PAI methods. Besides channel-level pruning, we further propose **PreConfig**, which can *reconfigure* the width dimension of a CNN to achieve a better accuracy-complexity trade-off with almost zero computational cost. Our empirical results show that the model after PreConfig can achieve higher accuracy with fewer parameters and FLOPs than the baseline for a variety of modern CNN architectures.

We summarize our contributions as follows:

- We propose to use the SynExp as a proxy for accuracy and formulate PAI as an optimization problem that maximizes SynExp under model size and/or FLOPs constraints. We also show that the accuracy of the CNN model pruned by solving the constrained optimization is independent of the pruning granularity.
- We introduce PreCrop, a channel-level structured pruning technique that builds on the proposed SynExp optimization. Our experiments show that CNN models pruned by PreCrop achieve a similar or better accuracy compared to the state-of-the-art unstructured PAI approaches. Compared to SynFlow, PreCrop achieves 2.7% and 0.9% higher accuracy on MobileNetV2 and EfficientNet on ImageNet with fewer parameters and FLOPs.
- We show that PreConfig can be used to optimize the width of each layer in the network with almost zero computational cost (e.g., within one second on CPU). Notably, PreConfig can effectively optimize the structure of EfficientNet and MobileNetV2, increasing the accuracy by 0.3% on ImageNet while using 20% fewer parameters and the same FLOPs.

2 Related Work

Model Compression in General can reduce the computational cost of large networks to ease their deployment in resource-constrained devices. Besides pruning, quantization [3, 30, 13], neural architecture search (NAS) [31, 23], and distillation [12, 28] are also commonly used to improve the efficiency of the model.

Training-Based Pruning uses various heuristic criteria to prune unimportant weights. They typically employ an iterative training-prune-retrain process where the pruning stage is intertwined with the training stage, which may increase the overall training cost by several folds.

Existing training-based pruning methods can be either unstructured [7, 15] or structured [11, 18], depending on the granularity and regularity of the pruning scheme. Training-based unstructured pruning usually achieves better accuracy given the same model size budget, while structured pruning can achieve a more practical speedup and compression without special support from custom hardware.

(Unstructured) Pruning-at-Initialization (PAI) [16, 27, 24] methods provide a promising approach to mitigating the high cost of training-based pruning. They can identify and prune unimportant weights right after initialization and before the training starts. Related to these efforts, authors of [5]

and [22] independently find that for the existing PAI methods, randomly shuffling the weights within a layer or reinitializing the weights does not cause any accuracy degradation.

Neural Architecture Search (NAS) [31, 26] automatically explores a large space of candidate models to achieve a better accuracy-efficiency trade-off. The typical bases of the NAS search space include the width, depth, resolution, and choice of building blocks. However, existing approaches can only search in a small subset of the possible configurations due to the cost. For example, the search space of the channel width usually only contains a limited set of integer values. The cost for NAS is also orders of magnitude higher than training a single model. Some NAS algorithms [1, 29] use a cheap proxy instead of training the whole network, but an expensive reinforcement learning [1] or evolutionary algorithm [19] is still used to predict a good network.

3 PRUNING-AT-INITIALIZATION VIA SYNEXP OPTIMIZATION

In this section, we first review the preliminaries and deficiencies of existing PAI methods. To overcome the limitations, we introduce a new proxy for the accuracy of the PAI compressed model. We then propose a new formulation of PAI that maximizes the proxy metric using convex optimization.

3.1 PAI BACKGROUND

Preliminaries. PAI aims to prune a neural network after initialization but before training to avoid the time-consuming *training-pruning-retraining* process. Prior to training, PAI typically uses the magnitude of gradients (with respect to weights) to estimate the importance of individual weights. This requires both forward and backward propagation passes. PAI prunes the weights (W) with smaller importance scores by setting the corresponding entries in the binary weight mask (M) to zero. More concretely, to remove weights W, M is applied to W in an element-wise manner as $W \odot M$, where $\odot$ denotes the Hadamard product.

Popular PAI approaches, such as SNIP [16], GraSP [27], and SynFlow [24], employ different methods to estimate the importance of individual weights. Single-shot PAI algorithms, such as SNIP and GraSP, prune the model to the desired sparsity in a single pass. Alternatively, SynFlow, which represents the state-of-the-art PAI algorithm, repeats the process of pruning a small fraction of weights and re-evaluating the importance scores until the desired pruning rate is reached. Through the iterative process, the importance of each weight can be estimated more accurately.

Specifically, the importance score for a fully connected network used in SynFlow is defined as:

$$\mathcal{S}(W_{ij}^l) = \left[\mathbb{1}^T \prod_{k=l+1}^{N} \left| W^k \odot M^k \right| \right]_i \left| W_{ij}^l M_{ij}^l \right| \left[\prod_{k=1}^{l-1} \left| W^k \odot M^k \right| \mathbb{1} \right]_j , \tag{1}$$

where N is the number of layers, W^l and M^l are the weight and weight mask of the l-th layer, $\mathcal{S}(W_{ij}^l)$ is the SynFlow score for a single weight W_{ij}^l, $|\cdot|$ is element-wise absolute operation, and $\mathbb{1}$ is an all-one vector. Here no training data or labels are required to compute the importance score, thus making SynFlow a data-agnostic algorithm.

Deficiencies. Similar to training-based fine-grained pruning [7, 15], existing PAI methods also use the *sum* of importance scores of the remaining weights as a proxy for model accuracy. Specifically, PAI obtains a binary weight mask (i.e., the pruning decisions) by maximizing the following objective:

$$\text{maximize} \sum_{l=1}^{N} S^l \cdot M^l \quad \text{over } M^l \quad \text{subject to} \sum_{l=1}^{N} \|M^l\|_0 \leq B_{\text{params}} , \tag{2}$$

where S^l is the importance score matrix for the l-th layer, $\|\cdot\|_0$ is the number of nonzero entries in a matrix, and B_{params} is the target size of the compressed model.

Given the setup of this optimization, it is natural that a subset of the individual weights will be deemed more important than others. Moreover, existing methods for computing the importance scores all depend on the values of the weights, thus any updates to the weights (such as reinitialization) will easily result in a change to the accuracy metric (i.e., the sum of the individual importance scores).

However, recent studies in [22, 5] show that randomly shuffling the weight mask M^l or reinitializing the weights W^l does not affect the final accuracy of models compressed by any of the existing PAI methods. In addition, they show that the different pruned models have a similar accuracy as long as they have the same layer-wise density. This finding suggests that the aforementioned metric is not a good proxy for indicating the accuracy of the pruned model.

3.2 SYNEXP INVARIANCE THEOREM

In this section, we propose a new proxy metric called SynExp to address the deficiencies of the existing PAI approaches. We argue that a good accuracy proxy should enable PAI to achieve the following:

1. The pruning decision (i.e., weight mask M) can be made *before* the model is initialized.
2. Maximization of the proxy should output *layer-wise density* p_l as the result, as opposed to pruning decisions for individual weights.

For the ease of later discussion, we formalize the weight matrix W and weight mask matrix M as two random variables, given a fixed density p_l for each layer, for random pruning before initialization. If W^l contains α_l parameters, $A^l = \{M, M_i^l \in \{0,1\}\ \forall 1 \leq i \leq \alpha_l, \sum_i M_i^l = p_l \times \alpha_l\}$ is the set of all possible M^l with the same shape as the W^l that satisfies the layer-wise density (p_l) constraint. Then, the random weight mask M^l for layer l is sampled uniformly from A^l. Also, each individual weight W_i^l in layer l is independently sampled from a given distribution D^l.

The observations in Section 3.1 indicate that different values of these two random variables M and W result in similar final accuracy of the pruned model. However, different values do change the proxy value for the model accuracy in existing PAI methods. For example, the SynFlow score in Equation 2 may change under different instantiations of M and W. Therefore, we propose a new proxy that is invariant to the instantiation of M and W for the model accuracy in the context of PAI — the *expectation* of the sum of the importance scores of all unpruned (i.e., remaining) weights. The proposed proxy can be formulated as follows:

$$\text{maximize} \underset{M,W}{\mathbb{E}}[\mathcal{S}] = \underset{M,W}{\mathbb{E}}\left[\sum_{l=1}^{N} S^l \cdot M^l\right] \quad \text{over } p_l \quad \text{subject to} \sum_{l=1}^{N} \alpha_l \cdot p_l \leq B_{\text{params}}\ , \tag{3}$$

where $\mathbb{E}_{M,W}[\mathcal{S}]$ stands for the expectation of the importance score $\mathcal{S}$ over W and M. In this new formulation, p_l is optimized to maximize the proposed proxy for model accuracy. Since the expectation is computed over the W and M, the instantiations of these two random variables do not affect the expectation.

To evaluate the expectation before weight initialization, we adopt the importance metric proposed by SynFlow, i.e., plugging $\mathcal{S}$ in Equation 1 into Equation 3 As a result, we can compute the expectation analytically without forward or backward propagations. This new expectation-based proxy is referred to as *SynExp*, i.e., synaptic expectation.

We show SynExp is invariant to the granularity of pruning PAI in the SynExp Invariance Theorem, which is stated as follows.

Theorem 1. SynExp Invariance Theorem. Given a specific CNN architecture, the SynExp ($\mathbb{E}_{[M,W]}[\mathcal{S}_{\text{SF}}]$) of any randomly compressed model with the same layer-wise density p_l is a constant, independent of the pruning granularity. The constant SynExp equals:

$$\underset{M,W}{\mathbb{E}}[\mathcal{S}_{\text{SF}}] = NC_{N+1} \prod_{l=1}^{N} (p_l C_l \cdot \mathbb{E}_{x\sim\mathcal{D}}[|x|])\ , \tag{4}$$

where N is the number of layers in the network, $\mathbb{E}_{x\sim\mathcal{D}}[|x|]$ is the expectation of magnitude of distribution $\mathcal{D}$, C_l is the input channel size of layer l and is also the output channel size of $l-1$, and $p_l = \frac{1}{\alpha_l}\|M_l\|_0$ is the layer-wise density.

In Equation 4, N and C^l are all hyperparameters of the CNN architecture and can be considered constants. $\mathbb{E}_{|\mathcal{D}^l|}$ is also a constant under a particular distribution $\mathcal{D}^l$. The layer-wise density p_l is

the only variable in the equation. Thus, SynExp satisfies both of the aforementioned properties: 1) pruning is done prior to the weight initialization; 2) the layer-wise density can be directly optimized. Furthermore, Theorem 1 shows that the granularity of pruning has no impact on the proposed SynExp metric. In other words, the CNN model compressed using either unstructured or structured pruning method is expected to have a similar accuracy.

The detailed proof of SynExp Invariance Theorem can be found in Appendix A. We also empirically verify it by randomly pruning each layer of a CNN at three different granular levels but with the same layer-wise density. Specifically, we perform random pruning at (1) weight-, (2) filter-, and (3) channel-level to achieve the desired layer-wise pruning ratios obtained from solving Equation 3. For weight and filter pruning, randomly pruning each layer to match the layer-wise density p_l occasionally detaches some weights from the network, especially when the density is low. The detached weights do not contribute to the prediction but are counted as remaining parameters. Thus, we remove the detached weights for a fair comparison following the same approach described in [25]. For channel pruning, it is not trivial to achieve the target layer-wise density while satisfying the constraint that the number of output channels of the previous layer must equal the number of input channels of the next layer. Therefore, we employ PreCrop proposed in Section 4.2. As shown in Figure 1, random pruning with different granularity can obtain a similar accuracy compared to SynFlow, as long as the layer-wise density remains the same. The empirical results are consistent with SynExp Invariance Theorem and also demonstrate the efficacy of the proposed SynExp metric. We include additional empirical results using different CNN architectures and other importance scores (e.g, SNIP, GraSP) in Appendix C.

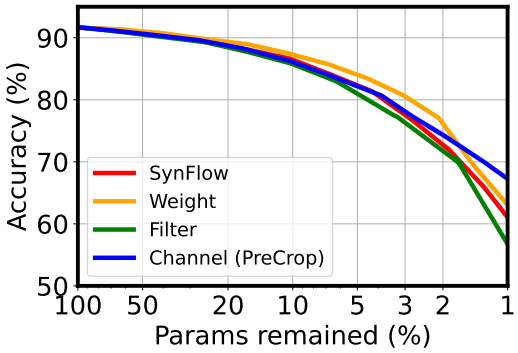


Figure 1: Comparison of the performance using different pruning granularities on ResNet20 using CIFAR-10.

3.3 Optimizing SynExp

As discussed in Section 3.2, the layer-wise density matters for our proposed SynExp approach. Here, we show how to obtain the layer-wise density in Equation 3 that maximizes SynExp under model size and/or FLOPs constraints.

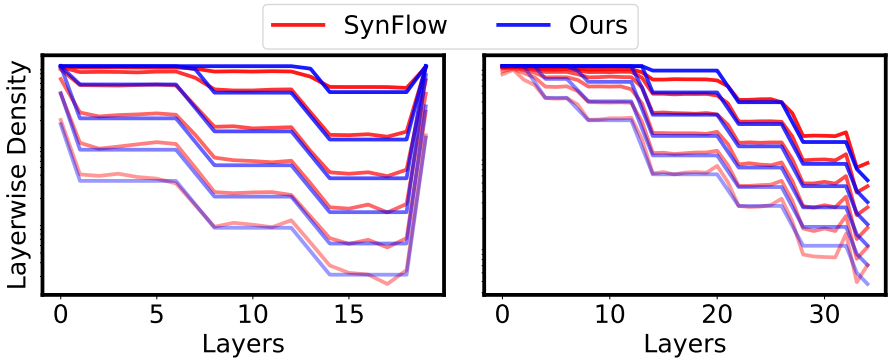
3.3.1 Optimizing SynExp with Parameter Count Constraint

Given that the goal of PAI is to reduce the size of the model, we need to add a constraint on the total number of parameters B_{params} (i.e., parameter count constraint), where B_{params} is typically greater than zero and less than the number of parameters in the original network. Since layer-wise density p_l is the only variable in Equation 3, we can simplify the equation by removing other constant terms, as follows:

$$\text{maximize} \sum_{l=1}^{N} \log p_l \quad \text{over } p_l \quad \text{subject to} \sum_{l=1}^{N} \alpha_l \cdot p_l \leq B_{\text{params}} \,, \quad 0 < p_l \leq 1, \forall 1 \leq l \leq N \,, \tag{5}$$

where α_l is the number of parameters in layer l.

Equation 5 is a convex optimization problem that can be solved analytically[1]. We compare the layer-wise density derived from solving Equation 5 with the density obtained using SynFlow. As shown in Figure 2, the layer-wise densities obtained by both approaches are nearly identical, where our new formulation eliminates the need for the iterative re-evaluation of the SynFlow scores as well as the pruning process in SynFlow. It is also worth noting that the proposed method can find the optimal layer-wise density even before the network is initialized.



(a) ResNet-20 (b) MobileNetV2

Figure 2: Comparison of the layer-wise densities obtained by SynExp optimization with parameter count constraint and SynFlow. Higher transparency means that the problem is constrained by a smaller parameter count.

3.3.2 Optimizing SynExp with Parameter Count and FLOPs Constraints

As discussed in Section 3.3.1, we can formulate PAI as a convex optimization problem with a constraint on the model size. However, the number of parameters does not necessarily reflect the performance (e.g., throughput) of the CNN model. In many cases, CNN models are compute-bound on commodity hardware [14, 9]. Therefore, we also introduce a FLOPs constraint in our formulation.

With the existing PAI algorithms, it is not straightforward to directly constrain the optimization using a bound on the FLOP count. The savings in FLOPs are instead the byproduct of the weight pruning as specified in Equation 2. In contrast, it is much easier to account for FLOPs in our formulation, which aims to determine the density of each layer as opposed to the inclusion of individual weights from different layers. Thus for each layer, we can easily derive the required FLOP count based on the density (p_l). After incorporating the constraint on FLOPs (B_{FLOPs}), the convex optimization problem becomes:

$$\text{maximize} \sum_{l=1}^{N} \log p_l \quad \text{over } p_l \quad \text{subject to} \sum_{l=1}^{N} \alpha_l \cdot p_l \leq B_{\text{params}}, \sum_{l=1}^{N} \beta_l \cdot p_l \leq B_{\text{FLOPs}}\ , \quad 0 < p_l \leq 1, \forall 1 \leq l \leq N, \tag{6}$$

where β_l is the number of FLOPs in the l^{th} layer.

Since the additional FLOPs constraint is linear, the optimization problem in Equation 6 remains convex and has an analytical solution[1]. By solving SynExp optimization with a fixed B_{params} but different B_{FLOPs}, we can obtain the layer-wise density for various models that have the same number of parameters but different FLOPs. Then, we perform random weight pruning on the CNN model to achieve the desired layer-wise density. We compare the proposed SynExp optimization (denoted as Ours) with other popular PAI methods. As depicted in Figure 3, given a fixed model size (1.5×10^4 in the figure), our method can be used to generate a Pareto Frontier that spans the spectrum of FLOPs, while other methods can only have a fixed FLOPs. Our method dominates all other methods in terms of both accuracy and FLOPs reduction.

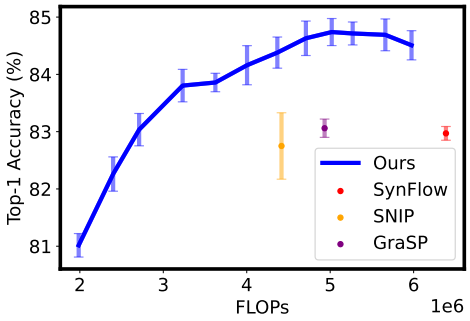


Figure 3: Comparison of our method with other PAI methods — we repeat the experiment using ResNet-20 on CIFAR-10 five times and report the mean and variance (error bar) of the accuracy. All the models in the figures have 1.5×10^4 parameters.

[1]We include analytical solutions for Equation 5 and Equation 6 in Appendix B for completeness.

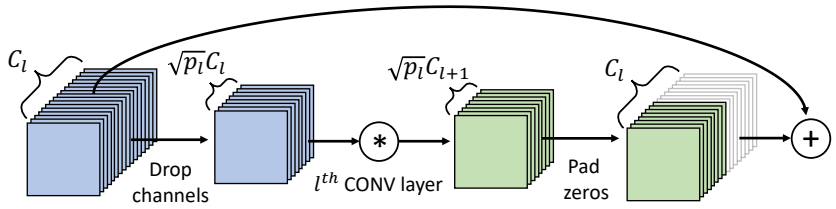


Figure 4: Illustration of PreCrop for layers with residual connections — C_l and C_{l+1} represent the number of input channels of layer l and $l+1$, respectively. p_l represents the density of layer l.

4 Structured Pruning-at-Initialization

The SynExp Invariance Theorem shows that the pruning granularity of PAI methods should not affect the accuracy of the pruned model. Channel pruning, which prunes the weights of the CNN at the output channel granularity, is considered the most coarse-grained and hardware-friendly pruning technique, Therefore, applying the proposed PAI method for channel pruning can avoid both complicated retraining/re-tuning procedures and irregular computations. In this section, we propose a structured PAI method for channel pruning, named PreCrop, to prune CNNs in the channel dimension. In addition, we propose a variety of PreCrop with relaxed density constraints to reconfigure the width of each layer in the CNN model, which is called PreConfig.

4.1 PreCrop

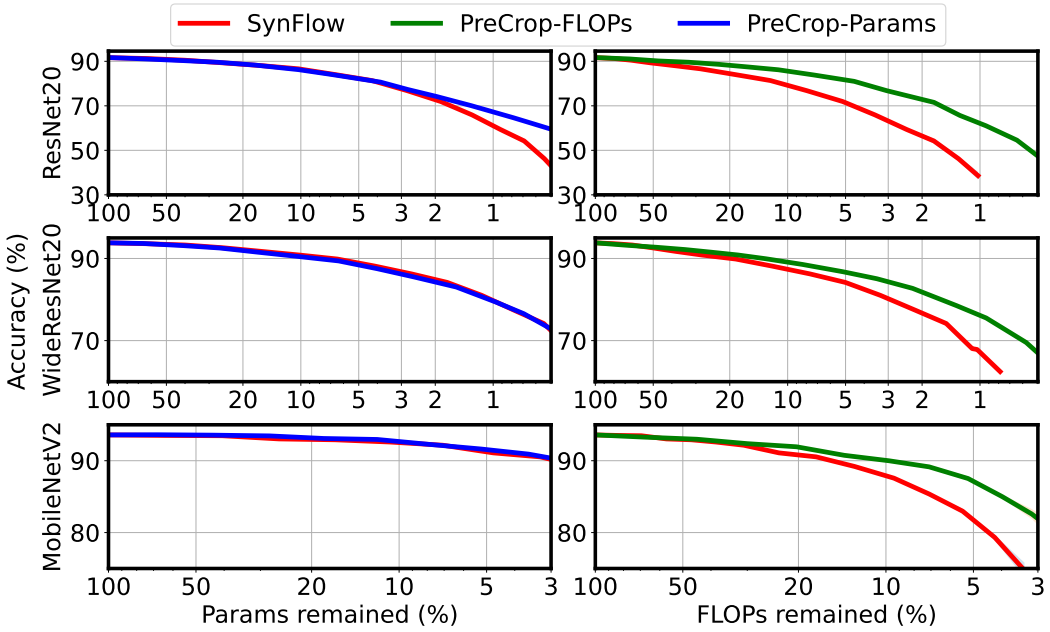
Applying the proposed PAI method to channel pruning requires a two-step procedure. First, the layer-wise density p_l is obtained by solving the optimization problem shown in Equation 5 or 6. Second, we need to decide how many output channels of each layer should be pruned to satisfy the layer-wise density. However, it is not straightforward to compress each layer to match a given layer-wise density due to the additional constraint that the number of output channels of the current layer must match the number of input channels of the next layer.

We introduce PreCrop, which compresses each layer to meet the desired layer-wise density. Let C_l and C_{l+1} be the number of input channels of layer l and $l+1$, respectively. C_{l+1} is also the number of output channels of layer l. For layers with no residual connections, the number of output channels of layer l is reduced to $\lfloor \sqrt{p_l} \cdot C_{l+1} \rfloor$. The number of input channels of layer $l+1$ needs to match the number of output channels of layer l, which is also reduced to $\lfloor \sqrt{p_l} \cdot C_{l+1} \rfloor$. Therefore, the actual density of layer l after PreCrop is $\sqrt{p_{l-1} \cdot p_l}$ instead of p_l. We empirically find that $\sqrt{p_{l-1} \cdot p_l}$ is close enough to p_l because the neighboring layers have similar layer-wise densities. Alternatively, one can obtain the exact layer-wise density p by only reducing the number of input or output channels of a layer. However, this approach leads to a significant drop in accuracy, because the number of the input and output channels can change dramatically (e.g., $p_l C_l \ll C_{l+1}$ or $C_l \gg p_l C_{l+1}$). This causes the shape of the feature map to change dramatically in adjacent layers, resulting in information loss.

For layers with residual connections, Figure 4 depicts an approach to circumvent the constraint on the number of channels of adjacent layers. We can reduce the number of input and output channels of layer l from C_l and C_{l+1} to $\sqrt{p_l}C_l$ and $\sqrt{p_l}C_{l+1}$, respectively. In this way, the density of each layer can match the given layer-wise density obtained from the proposed PAI method. Since the output of layer l needs to be added element-wisely with the original input to layer l, the output of layer l is padded with zero-valued channels to match the shape of the original input. In our implementation, we simply add the output of layer l to the first $\sqrt{p_l}C_{l+1}$ channels of the original input to layer l, thus requiring no extra memory or computation for zero padding. PreCrop eliminates the requirement for sparse computation in existing PAI methods and thus can be used to accelerate both training and inference of the pruned models.

4.2 PreConfig: PreCrop with Relaxed Density Constraint

PreCrop uses the layer-wise density obtained from solving the convex optimization problem, which is always less than 1 following the common setting for pruning (i.e., $p_l \leq 1$). However, this constraint on layer-wise density is not necessary for our method since we can increase the number of channels (i.e., expand the width of the layer) before initialization. By solving the problem in Equation 6 without the constraint $p_l \leq 1$, we can expand the layers with a density greater than 1 ($p_l > 1$) and prune the layers with a density less than 1 ($p_l < 1$). We call this variant of PreCrop as PreConfig



(a) PreCrop-Params vs. SynFlow.

(b) PreCrop-FLOPs vs. SynFlow.

Figure 5: Comparison of PreCrop-Params and PreCrop-FLOPs with SynFlow — we repeat the experiment using ResNet20 (top), WideResNet20 (middle), and MobileNetV2 (bottom) on CIFAR-10 three times and report the mean and variance (error bar) of the accuracy.

(PreCrop-Reconfigure). If we set B_{params} and B_{FLOPs} to be the same as the original network, we can essentially reconfigure the width of each layer of a given network architecture under certain constraints on model size and FLOPs.

The width of each layer in a CNN is usually designed manually, which often relies on extensive experience and intuition. Using PreConfig, we can automatically determine the width of each layer in the network to achieve a better cost-accuracy trade-off. PreConfig can also be used as (a part of) an ultra-fast NAS. Compared to conventional NAS, which typically searches on the width, depth, resolution, and choice of building blocks, PreConfig only changes the width. Nonetheless, PreConfig only requires a minimum amount of time and computation compared to NAS methods; it only needs to solve a relatively small convex optimization problem, which can finish within a second on a CPU.

5 Evaluation

In this section, we empirically evaluate PreCrop and PreConfig. We first demonstrate the effectiveness of PreCrop by comparing it with SynFlow. We then use PreConfig to tune the width of each layer and compare the accuracy of the model after PreConfig with the original model. We perform experiments using various modern CNN models, including ResNet [10], MobileNetV2 [21], and EfficientNet [23], on both CIFAR-10 and ImageNet. We set all hyperparameters used to train the models pruned by different PAI algorithms to be the same. See Appendix E for detailed experimental settings.

5.1 Evaluation of PreCrop

For CIFAR-10, we compare the accuracy of SynFlow (red line) and two variants of PreCrop: PreCrop-Params (blue line) and PreCrop-FLOPs (green line). PreCrop-Params adds the parameter count constraint whereas PreCrop-FLOPs imposes the FLOPs constraint into the convex optimization problem. As shown in Figure 5a, PreCrop-Params achieves similar or even better accuracy as SynFlow under a wide range of different model size constraints, thus validating that PreCrop-Params can be as effective as the fine-grained PAI method. Considering the benefits of structured pruning, PreCrop-Params should be favored over existing PAI methods. Figure 5b further shows that PreCrop-FLOPs consistently outperforms SynFlow by a large margin, especially when the reduction in FLOPs is large. The experimental results show that PreCrop-FLOPs should be adopted when the performance of the model is limited by the computational cost.

Table 1: **Comparison of PreCrop with SynFlow on ImageNet** — The dagger(†) implies that the numbers are theoretical without considering the overhead of sparse matrices in storing and computing.

NETWORK	METHODS	FLOPS (G)	PARAMS (M)	ACCURACY (%)
RESNET34	BASELINE	3.64	21.80	73.5
	SYNFLOW	2.78†	**10.91**†	**72.1 (-1.4)**
	PRECROP	**2.73**	11.09	71.5 (-2.0)
MOBILENETV2	BASELINE	0.33	3.51	69.6
	SYNFLOW	**0.26**†	2.44†	67.6 (-2.0)
	PRECROP	**0.26**	**2.33**	**68.8 (-1.8)**
	SYNFLOW	**0.21**†	1.91†	64.5 (-5.1)
	PRECROP	**0.21**	**1.85**	**67.2 (-2.4)**
EFFICIENTNETB0	BASELINE	0.40	5.29	73.0
	SYNFLOW	**0.30**†	3.72†	71.8 (-1.2)
	PRECROP	**0.30**	**3.67**	**72.7 (-0.3)**

Table 1 summarizes the comparison between PreCrop and SynFlow on ImageNet. For ResNet-34, PreCrop achieves 0.6% lower accuracy compared to SynFlow with a similar model size and FLOPs. For both MobileNetV2 and EfficientNetB0, PreCrop achieves 1.2% and 0.9% accuracy improvements compared to SynFlow with strictly fewer FLOPs and parameters, respectively. The experimental results on ImageNet further support SynExp Invariance Theorem that coarse-grained structured pruning (e.g., PreCrop) can perform as well as unstructured pruning. In conclusion, PreCrop achieves a favorable accuracy and model size/FLOPs tradeoff compared to the state-of-the-art PAI algorithm.

5.2 EVALUATION OF PRECONFIG

As discussed in Section 4.2, PreConfig can be viewed as an ultra-fast NAS technique, which adjusts the width of each layer in the model even before the weights are initialized.

Table 2: **PreConfig on ImageNet.**

NETWORK	METHODS	FLOPS (G)	PARAMS (M)	ACCURACY (%)
RESNET34	BASELINE	**3.64**	21.80	**73.5**
	PRECONFIG	**3.64**	**16.52 (75.8%)**	73.3 (-0.2)
MOBILENETV2	BASELINE	0.33	3.51	69.6
	PRECONFIG	**0.32 (97.0%)**	**2.83 (80.6%)**	**69.9 (+0.3)**
EFFICIENTNETB0	BASELINE	**0.40**	5.29	73.0
	PRECONFIG	**0.40**	**4.29 (81.1%)**	**73.3 (+0.3)**

Table 2 compares the accuracy of the reconfigured model with the original model under similar model size and FLOPs constraints. For ResNet34, with similar accuracy, we reduce the parameter count by 25%. For MobileNetV2, we achieve 0.3% higher accuracy than the baseline with 20% fewer parameters and 3% fewer FLOPs. For the EfficientNet, we can also achieve 0.3% higher accuracy than the baseline with only 80% of the parameters and the same FLOPs. Note that EfficientNet is identified by a NAS method. As PreConfig only changes the number of channels of the model before initialization, we believe it also applies to other compression techniques.

6 CONCLUSION

In this work, we theoretically and empirically show that the accuracy of the CNN models pruned using PAI methods depends on the layer-wise density. We formulate PAI as a simple convex SynExp optimization. Based on SynExp optimization, we further propose PreCrop and PreConfig to prune and reconfigure CNNs in the channel dimension. Our experimental results demonstrate that PreCrop can outperform existing fine-grained PAI methods on various networks and datasets.

REPRODUCIBILITY STATEMENT

The proof of SynExp Invariance Theorem is stated in the appendix with explanations. We provide the source code for the key experiments in the paper. We thoroughly checked the implementation and also verified empirically that the results in this paper are reproducible. The source code will be made available through GitHub.

REFERENCES

[1] Mohamed S Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak, and Nicholas D Lane. Zero-cost proxies for lightweight nas. *arXiv preprint arXiv:2101.08134*, 2021.

[2] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033*, 2020.

[3] Zhen Dong, Zhewei Yao, Yaohui Cai, Daiyaan Arfeen, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Hawq-v2: Hessian aware trace-weighted quantization of neural networks. *arXiv preprint arXiv:1911.03852*, 2019.

[4] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[5] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Pruning neural networks at initialization: Why are we missing the mark? *arXiv preprint arXiv:2009.08576*, 2020.

[6] Trevor Gale, Matei Zaharia, Cliff Young, and Erich Elsen. Sparse gpu kernels for deep learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press, 2020. ISBN 9781728199986.

[7] 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.

[8] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

[9] Pawan Harish and Petter J Narayanan. Accelerating large graph algorithms on the gpu using cuda. In *International conference on high-performance computing*, pp. 197–208. Springer, 2007.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[11] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 1389–1397, 2017.

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

[13] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2704–2713, 2018.

[14] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy,

James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. *SIGARCH Comput. Archit. News*, 45(2):1–12, jun 2017. ISSN 0163-5964. doi: 10.1145/3140659.3080246. URL https://doi.org/10.1145/3140659.3080246.

[15] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990.

[16] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

[17] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

[18] 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*, pp. 5058–5066, 2017.

[19] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, pp. 4780–4789, 2019.

[20] Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. *arXiv preprint arXiv:2003.02389*, 2020.

[21] 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*, pp. 4510–4520, 2018.

[22] Jingtong Su, Yihang Chen, Tianle Cai, Tianhao Wu, Ruiqi Gao, Liwei Wang, and Jason D Lee. Sanity-checking pruning methods: Random tickets can win the jackpot. *arXiv preprint arXiv:2009.11094*, 2020.

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

[24] Hidenori Tanaka, Daniel Kunin, Daniel LK Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *arXiv preprint arXiv:2006.05467*, 2020.

[25] Artem Vysogorets and Julia Kempe. Connectivity matters: Neural network pruning through the lens of effective sparsity. *arXiv preprint arXiv:2107.02306*, 2021.

[26] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, et al. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12965–12974, 2020.

[27] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*, 2020.

[28] 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*, pp. 8715–8724, 2020.

[29] Dongzhan Zhou, Xinchi Zhou, Wenwei Zhang, Chen Change Loy, Shuai Yi, Xuesen Zhang, and Wanli Ouyang. Econas: Finding proxies for economical neural architecture search. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pp. 11396–11404, 2020.

[30] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[31] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

A PROOF OF SYNEXP INVARIANCE THEOREM

Theorem 1. *Given a specific CNN architecture, the SynExp ($\mathbb{E}_{[M,W]}[\mathcal{S}_{SF}]$) of any randomly compressed model with the same layer-wise density p_l is a constant, independent of the pruning granularity. The constant SynExp equals:*

$$\mathop{\mathbb{E}}_{M,W}[\mathcal{S}_{SF}] = NC_{N+1}\prod_{l=1}^{N}(p_l C_l \cdot \mathbb{E}_{x\sim\mathcal{D}}[|x|]) \,, \tag{7}$$

where N is the number of layers in the network, $\mathbb{E}_{x\sim\mathcal{D}}[|x|]$ is the expectation of magnitude of distribution $\mathcal{D}$, C_l is the input channel size of layer l and is also the output channel size of $l-1$, and $p_l = \frac{1}{\alpha_l}\|C_l\|_0$ is the layer-wise density.

Proof. Assuming the network has N layers, weight matrix $W^l \in \mathbb{R}^{C_l\times C_{l+1}}$, mask matrix $M^l \in \{0,1\}^{C_l\times C_{l+1}}$. C_l and C_{l+1} are the input and output channel size of layer l. As the output channel size of any layer l equals to the input channel size of the next layer $l+1$, we have $C_{l+1} = C_{l+1}, \forall l < N$.

We first prove the Theorem 1 on fully-connected network, and we can extend it to CNNs easily. From Equation 1, in a fully-connected network, the Synaptic Flow score for any parameter W^l_{ij} with mask M^l_{ij} in layer l equals to:

$$\mathcal{S}_{\text{SF}}(W^l_{(i,j)}) = \left[\mathbb{1}^T \prod_{k=l+1}^{N}\left|W^k \odot M^k\right|\right]_i \left|W^l_{(i,j)}M^l_{(i,j)}\right| \left[\prod_{k=1}^{l-1}\left|W^k \odot M^k\right|\mathbb{1}\right]_j \tag{8}$$

We compute the SynExp of the layer l ($\mathbb{E}_{[M,W]}(\mathcal{S}_{\text{SF}})^{[l]}$), then the SynExp of the network is simply the sum of SynExp of all layers:

$$\mathbb{E}_{[M,W]}(\mathcal{S}_{\text{SF}}) = \sum_{l=1}^{N}\mathbb{E}_{[M,W]}(\mathcal{S}_{\text{SF}})^{[l]} \tag{9}$$

We define the expectation value for input channel i, output channel j, and the whole layer in layer l as $E^l_{(i,*)}$, $E^l_{(*,j)}$, and $E^l_{(*,*)}$:

$$\mathbb{E}^l_{(i,*)} = \frac{1}{C_{l+1}}\sum_x |W^l_{(i,x)}M^l_{(i,x)}| \tag{10}$$

$$\mathbb{E}^l_{(*,j)} = \frac{1}{C_l}\sum_x |W^l_{(x,j)}M^l_{(x,j)}| \tag{11}$$

$$\mathbb{E}^l_{(i,j)} = \mathbb{E}^l_{(*,*)} = \frac{1}{C_l C_{l+1}}\sum_{i,j}|W^l_{(i,j)}M^l_{(i,j)}| = \frac{1}{\alpha_l}\sum_{i,j}|W^l_{(i,j)}M^l_{(i,j)}| = p_l\mathbb{E}_{|\mathcal{D}^l|} \tag{12}$$

Here we use $\mathbb{E}_{|\mathcal{D}^l|}$ to denote $\mathbb{E}_{x\sim\mathcal{D}}[|x|]$.

As the weight in layer l is sampled from distribution $\mathcal{D}$, and the mask matrices are also randomly sampled, we have

$$\mathbb{E}^{[k]}_{(*,*)} = \mathbb{E}^k_{(i,*)} = \mathbb{E}^k_{(*,j)} = p_l\mathbb{E}_{|\mathcal{D}^l|} \tag{13}$$

With $E^k_{(i,*)}$, $E^k_{(*,j)}$, and $E^l_{(*,*)}$, we can rewrite Equation 8 to:

$$\mathbb{E}[\mathcal{S}_{\text{SF}}(W^l_{(i,j)})] = \left(\prod_{k=l+2}^{N} C_{k+1}\mathbb{E}^k_{(*,*)}\right)\cdot C_{l+2}\mathbb{E}^{l+1}_{(i,*)}\cdot\mathbb{E}^l_{(i,j)}\cdot C_{l-1}\mathbb{E}^{l-1}_{(*,j)}\cdot\left(\prod_{k=1}^{l-2}C_k E^k_{(*,*)}\right) \tag{14}$$

Combining Equation 3 and 14, because the instantiation of the weight matrices and mask matrices for each layer are independent:

$$\begin{aligned}
&\mathbb{E}_{[M,W]}(\mathcal{S}_{\text{SF}})^{[l]} = \mathbb{E}\left[\sum_{i=1}^{C_l}\sum_{j=1}^{C_{l+1}} \mathcal{S}_{\text{SF}}(W^l_{(i,j)})\right] = \sum_{i=1}^{C_l}\sum_{j=1}^{C_{l+1}} \mathbb{E}\left[\mathcal{S}_{\text{SF}}(W^l_{(i,j)})\right] \\
&= \left(\prod_{k=l+2}^{N} p_k C_{k+1}\mathbb{E}_{|\mathcal{D}^k|}\right)\sum_{i=1}^{C_l}\sum_{j=1}^{C_{l+1}}\left(p_{l+1}C_{l+2}\mathbb{E}^{l+1}_{(i,*)}\cdot p_l\mathbb{E}_{|\mathcal{D}^l|}\cdot p_{l-1}C_{l-1}\mathbb{E}^{l-1}_{(*,j)}\right)\left(\prod_{k=1}^{l-2} C_k\mathbb{E}_{|\mathcal{D}^k|}\right) \\
&= \left(\prod_{k=l+2}^{N} p_k C_{k+1}\mathbb{E}_{|\mathcal{D}^k|}\right) C_l C_{l+1}\left(p_{l+1}C_{l+2}\mathbb{E}_{|\mathcal{D}^{l+1}|}\cdot p_l\mathbb{E}_{|\mathcal{D}^l|}\cdot p_{l-1}C_{l-1}\mathbb{E}_{|\mathcal{D}^{l-1}|}\right)\left(\prod_{k=1}^{l-2} p_k C_k\mathbb{E}_{|\mathcal{D}^k|}\right) \\
&= C_{N+1}\prod_{l=1}^{N}(p_l C_l\mathbb{E}_{|\mathcal{D}^l|})
\end{aligned} \tag{15}$$

According to Equation 9,

$$\begin{aligned}
\mathbb{E}_{[M,W]}(\mathcal{S}_{\text{SF}}) &= \sum_{l=1}^{N} C_{N+1}\prod_{l=1}^{N}(C_l\mathbb{E}_{|\mathcal{D}^l|}) \\
&= NC_{N+1}\prod_{l=1}^{N}(p_l C_l\cdot\mathbb{E}_{x\sim\mathcal{D}}[|x|]),
\end{aligned} \tag{16}$$

SynExp Invariance Theorem can also be extended to CNNs, as it is obvious that SynExp of CNNs is proportional to that of fully connected networks. Thus the difference of SynExp between CNNs and fully connected networks for each layer is only a factor equal to $\frac{K^2}{C_{l+1}}$, where K is the kernel size of the convolutional layer. □

B Solution of the Optimization problem

For the convex optimization problem in Equation 5, Equation 6, or PreConfig, we can simply use Karush–Kuhn–Tucker (KKT) conditions to analytically solve it. We include the solutions as follows for completeness. In practice, we use convex solver to solve the problem to avoid the piecewise function.

B.1 Solution for Equation 5

$$\begin{aligned}
p_l &= \min(\frac{\mu}{\alpha}, 1) \\
\text{where } \mu \text{ satisfies: } &\sum_{l=1}^{N}\min(\alpha_l, \mu) = B_{\text{params}}
\end{aligned} \tag{17}$$

B.2 Solution for Equation 6

$$\begin{aligned}
p_l &= \min(\frac{1}{\mu_1\alpha_l + \mu_2\beta_l}, 1) \\
\text{where } \mu_1, \mu_2 \text{ satisfy: } &\sum_{l=1}^{N}\alpha_l\min(\frac{1}{\mu_1\alpha_l + \mu_2\beta_l}, 1) = B_{\text{params}} \\
&\sum_{l=1}^{N}\beta_l\min(\frac{1}{\mu_1\alpha_l + \mu_2\beta_l}, 1) = B_{\text{flops}}
\end{aligned} \tag{18}$$

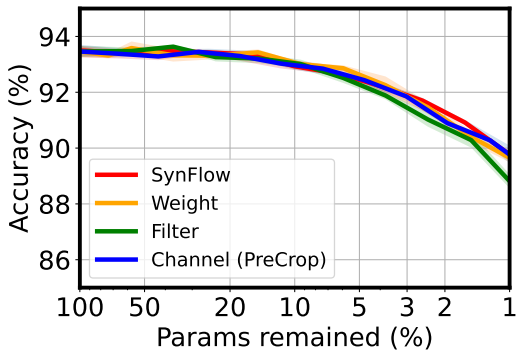


Figure 6: Comparison of the performance using different pruning granularities on VGG16 using CIFAR-10.

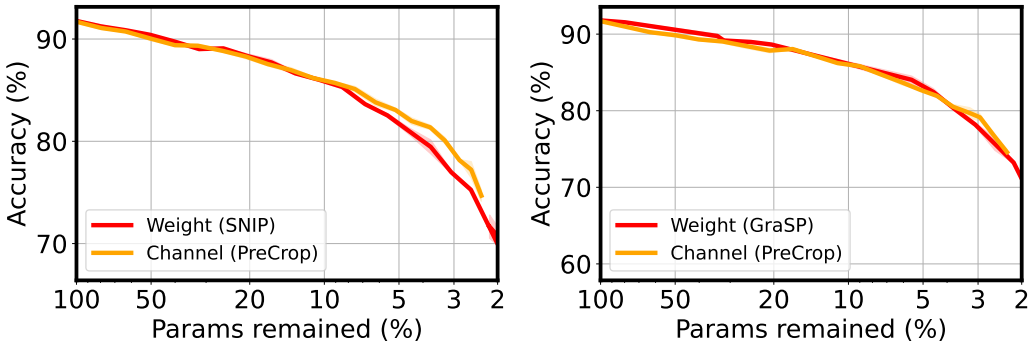


Figure 7: Comparison of the performance using different pruning granularities on ResNet20 using CIFAR-10. SNIP (left) and GraSP (right) importance scores are used.

B.3 Solution for PreConfig

$$\text{where } \mu_1, \mu_2 \text{ satisfy: } \begin{aligned} p_l &= \frac{1}{\mu_1 \alpha_l + \mu_2 \beta_l} \\ \sum_{l=1}^{N} \alpha_l \frac{1}{\mu_1 \alpha_l + \mu_2 \beta_l} &= B_{\text{params}} \\ \sum_{l=1}^{N} \beta_l \frac{1}{\mu_1 \alpha_l + \mu_2 \beta_l} &= B_{\text{flops}} \end{aligned} \tag{19}$$

In practice, to avoid solving the μ, we use a convex optimization solver, which can obtain the solution with a CPU within a second for such a small scale convex optimization.

C More Empirical Results on SynExp Invariance Theorem

We show more empirical results that validates SynExp Invariance Theorem. We first show the comparison of the performance using different pruning granulariteis on VGG16 using CIFAR-10. All the settings in this experiment is the same as in Figure 1, except this experiment is done on VGG16.

Then we also verify that SynExp Invariance Theorem not only holds for SynFlow, but also holds for other PAI algorithms. In this experiment, we first use other PAI (i.e., SNIP and GraSP) to obtain the layerwise density p_l. Then we use random pruning to match p_l in the channel level. The results are shown in Figure 7.

As shown in all the above experiments, as long as the layerwise density is the same, the pruning granularties do not affect the model accuracy.

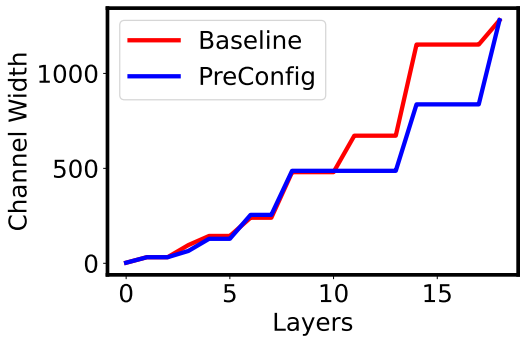


Figure 8: Comparison of the channel width of EfficientNetB0 before and after PreConfig.

D Channel Width Comparison

We also include a comparison of the channel width between the baseline EfficientNetB0 and PreConfig EfficientNetB0 in Figure 8.

E Experiment Details

E.1 Implementation

We adapt model implementations of ResNet, ShuffleNet, and MobileNetv2 from imgclsmob[2]. The implementations of SynFlow, SNIP, and GraSP are based on the codebase of SynFlow[3].

E.2 Hyperparameters

Here we provide the hyperparameters used in training all models in Table 3. No AutoAugment, Label Smoothing, or stochastic depth is used during training. All the CIFAR-10 models are trained with same hyperparameter setting.

Table 3: Hyperparameters used in training.

	CIFAR-10	ImageNet		
		MobileNet	ResNet	EfficientNet
Optimizer	momentum	momentum	momentum	momentum
Training Epochs	160	180	90	150
Batch Size	128	256	512	256
Initial Learning Rate	0.1	0.025	0.2	0.035
Learning Rate Schedule	linear	drop at each epoch	drop at 30, 60 epoch	drop at each epoch
Drop Rate	N.A.	0.98	0.1	0.99
Weight Decay	10^{-4}	4×10^{-5}	10^{-4}	4×10^{-5}

[2]https://github.com/osmr/imgclsmob
[3]https://github.com/ganguli-lab/Synaptic-Flow