# Pruning-as-Search: Efficient Neural Architecture Search via Channel Pruning and Structural Reparameterization

**Yanyu Li**[1*], **Pu Zhao**[1], **Geng Yuan**[1], **Xue Lin**[1], **Yanzhi Wang**[1], **Xin Chen**[2*]
[1]Northeastern University
[2]Intel Corp.
{li.yanyu, zhao.pu, yuan.geng, xue.lin, yanz.wang}@northeastern.edu, xin.chen@intel.com

## Abstract

Neural architecture search (NAS) and network pruning are widely studied efficient AI techniques, but not yet perfect. NAS performs exhaustive candidate architecture search, incurring tremendous search cost. Though (structured) pruning can simply shrink model dimension, it remains unclear how to decide the per-layer sparsity automatically and optimally. In this work, we revisit the problem of layer-width optimization and propose Pruning-as-Search (PaS), an end-to-end channel pruning method to search out desired sub-network automatically and efficiently. Specifically, we add a depth-wise binary convolution to learn pruning policies directly through gradient descent. By combining the structural reparameterization and PaS, we successfully searched out a new family of VGG-like and lightweight networks, which enable the flexibility of arbitrary width with respect to each layer instead of each stage. Experimental results show that our proposed architecture outperforms prior arts by around $1.0\%$ top-1 accuracy under similar inference speed on ImageNet-1000 classification task. Furthermore, we demonstrate the effectiveness of our width search on complex tasks including instance segmentation and image translation. Code and models are released.

## 1 Introduction

Recently, Deep Neural Networks (DNNs) have achieved great success in various applications. However, their tremendous memory and computation consumption lead to difficulties for the wide deployment of DNNs especially on resource-limited edge devices. To mitigate this gap, *Neural Architecture Search* (NAS) and *network pruning* are proposed to lower DNN memory occupancy and improve inference speed.

Although NAS can automatically explore desired operator type, skip connections, as well as model depth and width, it suffers from significant searching overhead with exponentially growing search space. On the other hand, network pruning is designed to remove redundant weights from pre-defined network architectures. The fundamental challenge that impedes the pruning to serve as an efficient automatic width search method is that the layer-wise pruning ratios are hard to be determined automatically. It usually requires a trial-and-error process with domain expertise.

To overcome the disadvantages of NAS and pruning, we propose pruning-as-search (PaS), an efficient end-to-end channel pruning method to automatically search the desirable sub-networks with only a similar cost to one typical training process. Specifically, we introduce a **depth-wise binary convolutional (DBC)** layer as a pruning indicator after each convolutional (CONV) layer, as shown in Figure 1. We incorporate Straight Through Estimator (STE) technique to facilitate the trainability of DBC layers. Starting from a well-trained network, PaS only requires one fine-tuning process to learn layer-wise pruning policy and fine-tune the pruned DNN simultaneously. Moreover, another superiority of PaS is the ability to step out of the magnitude trap, which distinguishes PaS from prior arts that use the magnitude of weights or feature maps to select the pruning location (more details in Section 3). We demonstrate that PaS consistently outperforms prior arts by $0.3\%$ to $2.7\%$ on top-1 accuracy under the same computation constraint on ImageNet with ResNet-50.

Besides, we reveal a new dimension of flexibility in DNN width design inspired by structural reparameterization in [Ding *et al.*, 2021]. The design of DNN width is rather rigid in state-of-the-arts. To enable residual connection, most SOTA backbone networks [He *et al.*, 2016; Tan and Le, 2019] have to set block width identical within a stage, otherwise the dimension of output feature map mismatches those from identity path. However, do we really have to sacrifice the flexibility of layer width within a stage due to the *identical width by stage* design? In this work, we propose a feasible solution to this problem. By skipping only one convolution layer, identity path can be reparameterized into CONV layer during inference. We propose to search on a super-net with reparameterization design, and deploy compact sub-nets without residual connections, such that each layer can have arbitrary width and be free from the aforementioned constraint. We release a brand new family of VGG-like networks with stack of pure $3 \times 3$ CONVs as well as lightweight DNNs with depth-wise separable CONVs. Our searched network architecture can achieve $2.1\times$ inference speedup and $0.3\%$ higher accuracy compared with ResNet-50 on ImageNet.

We summarize our main contributions as following:

- We develop a PaS algorithm that directly learn pruning policy via DBC layers. Our method saves searching cost compared to brute search, enables exploration which is beyond the capabilities of traditional pruning, and is easily integratable to complex tasks.

[*]Work is done at Kwai Inc.

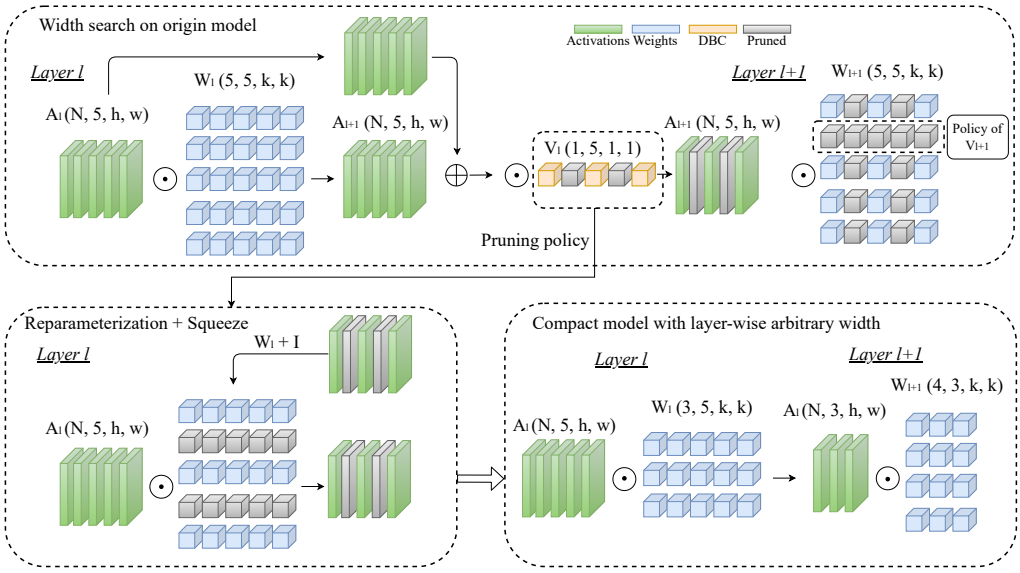


Figure 1: Illustration of depth-wise binary convolution (DBC) in our PaS and reparameterization-based deployment. Pruning is determined automatically by DBC parameters. DBC layer is post-block attached so that both channels from CONV output and channels from identity path can be removed simultaneously. Finally we perform structural reparameterization to merge the branch into mainstream CONV and get a plain and compact network. We have width 3 in layer $l$ and width 4 in layer $l+1$, which is not achievable if there are residual connections.

- We are the first to investigate the special role of structural reparameterazation in width optimization. Compared to SOTA backbone DNNs with residual connections, we introduce a new flexibility in channel configuration, enabling arbitrary width by layer instead of by stage.
- We release a new family of backbone networks based on our width search and reparameterization. We also validate the effectiveness of our PaS method on complex tasks including segmentation and generative models.

## 2 Related Work

**Neural Architecture Search (NAS).** NAS aims to design high-performance and efficient network architecture by leveraging powerful computation resources to reduce the intervention of humans. Prior works [Zoph and Le, 2017] incorporate RL into searching process. An agent such as an RNN is trained during the search process to generate desired candidate network architectures. In general, RL-based and evolution-based NAS methods require long searching time, which can even take hundreds to thousands of GPU days. Gradient-based methods [Liu *et al.*, 2018] relax the discrete architecture representation into a continuous and differentiable form, enabling more efficient architecture search by leveraging gradient descent. However, these methods either require conducting the search process on a proxy task (on the smaller dataset) or need a huge memory cost for a large number of candidate operators.

**Network Pruning.** DNN pruning removes redundant weights in DNNs, thus effectively reduces both storage and computation cost. According to the sparsity type, network pruning are generally categorized into (1) unstructured pruning [Han *et al.*, 2015] and (2) structured pruning [Li *et al.*, 2019]. Unstructured pruning usually achieves a high sparsity ratio, but it is hard to have a considerable acceleration due to its irregular sparsity. Structured pruning removes entire filters/channels of weights and has the potential to reconstruct the pruned model to a small dense model, enabling significant acceleration and storage reduction. Specifically, network pruning in this work refers to channel pruning [Hu *et al.*, 2016; He *et al.*, 2017; Liu *et al.*, 2017; Guo *et al.*, 2021; Su *et al.*, 2021] as we aim to find compact models with width shrinking.

**Structural Reparameterization.** As a multi-branch strategy, structural reparameterization is proposed either to improve accuracy of origin architecture [Marnerides *et al.*, 2018], or to deliver plain models for speedup purposes [Ding *et al.*, 2021] by merging branch operator into mainstream operator (CONV in general) in a mathematically equivalent way. In this work, we integrate structural reparameterization to decouple the model architecture at training-time and inference-time. We take advantage of residual connection during training but deploy plain architecture with arbitrary widths.

## 3 Challenges and Motivations

We discuss the key challenges in width optimization. Specifically, (C1) Tremendous searching cost for NAS, (C2) magnitude trap in pruning, and (C3) rigid width constraint.

### 3.1 Tremendous Search Cost for NAS

In NAS, the exponentially growing search space is a major problem. Specifically, the RL-based NAS methods [Zoph and Le, 2017] typically need to train each candidate architecture with multiple epochs, incurring large searching cost. For a model with $N_L$ layers, each layer has $N_C$ candidate width, (e.g. $\{0, 16, 32, 48, 64, ...\}$), and each candidate needs to be trained by $N_E$ (e.g. epochs) computation force. Then the computation complexity is $O((N_C)^{N_L} \cdot N_E)$. Besides, the differentiable NAS methods [Liu *et al.*, 2018] build a memory-intensive super-net to train multiple architectures simultane-

| Methods | CE | AP | non-MT | FW |
|---|---|---|---|---|
| NAS, Search to Prune | ✗ | ✓ | / | ✗ |
| Uniform, Handcrafted Pruning | ✓ | ✗ | ✗ | ✗ |
| Magnitude + One-Shot | ✓ | ✓ | ✗ | ✗ |
| Magnitude + Iterative Update | ✓ | ✓ | ✗ | ✗ |
| Equal Regularized | ✓ | ✓ | ✗ | ✗ |
| **Our PaS** | ✔ | ✔ | ✔ | ✔ |

Table 1: We summarize model width search methods by four metrics. (i) Computation efficient (CE), a practical method should not incorporate tremendous search cost. (ii) Automatic policy (AP), whether or not the layer width can be determined automatically without human intervention. (iii) Free from magnitude trap (non-MT), a desired method should be adaptive and update its policy dynamically instead of using a one-shot heuristic. (iv) Flexible per-layer width (FW), the per-layer width should not be forced to match the number of input and output channels in the block.

ously with memory complexity of $O(N_C N_L)$ for a super-net of $N_L$ layers and $N_C$ candidates each layer, leading to limited discrete search space up-bounded by the available memory.

### 3.2 Magnitude Trap for Pruning

The key challenge of pruning is to decide the per-layer pruning ratio and pruning positions. Following [Han *et al.*, 2016], magnitude-based method is widely employed which prunes channels smaller than a global threshold. It is based on the assumption that filters/channels with smaller magnitudes serve less important for final accuracy. However, this assumption is not necessarily true. As shown in Appendix A.1 , we demonstrate that simply penalizing small magnitude filters/channels leads to non-recoverable pruning which means that small channels do not have a chance to become large enough in contribution to the accuracy due to penalizing. It becomes pure exploitation without exploration, and we refer this as the *magnitude trap*, as illustrated in Tab. 1 We show the comparison of different types of methods including NAS [Zoph and Le, 2017; Zhong *et al.*, 2018] Search-to-Prune [Liu *et al.*, 2019; He *et al.*, 2018] Uniform pruning [Luo *et al.*, 2017], Handcrafted Pruning [Zhang *et al.*, 2018b] Magnitude plus One-Shot pruning [Han *et al.*, 2016], Magnitude plus Iterative Update [Zhang *et al.*, 2018a], Equally Regularized [Liu *et al.*, 2017; Guan *et al.*, 2022].

Till now, there are very few work recognizing this issue. [Liu *et al.*, 2017] and [Guan *et al.*, 2022] apply uniform regularization to all pruning indicators. All channels are forced to be close to zero, then the smallest are pruned. Though enabling policy updating thus exploration, the pruned channels are finally selected by magnitude, which falls into magnitude trap as well. Plus, this process usually destroys the performance of super-net, as shown in Appendix A.1.

### 3.3 Rigid Width Design

Inspired by ResNet, most state-of-the-art DNNs incorporate the residual connections in their building blocks. It requires that the input dimension matches the output dimension. This constraints the DNN design paradigm to divide DNNs into stages by downsampling positions. Within each stage, feature sizes are identical, and each block shares same width to ensure the residual connection can work without size mismatch issues, as demonstrated in Fig. 2 (a, b, c).

Current compression techniques, including NAS and pruning, still suffer from these constraints, significantly decreasing the design flexibility. In NAS, candidate blocks are still residual connected and should share the same width within a stage. Consequently, the freedom is to search expansion ratio by block, and width by stage, which is still at coarse level, as shown in Fig. 2 (b) and (c). Existing pruning methods struggle to satisfy the aforementioned design constraint as well. (i) [Wu *et al.*, 2019] remove the identity path or replace them with a $1 \times 1$ CONV to match the channel number, sacrificing the advantages of identity path with considerable performance degradation. (ii) [Guo *et al.*, 2020] incorporates constraints to force all blocks sharing the same output dimension within each stage. As observed in Fig. 3, this method is obviously non-optimal as the pruned channel width in a specific stage are not necessarily identical. (iii) As discussed in [Guan *et al.*, 2022], the last CONV within a stage can be pruned freely by incorporating a reshaping operation. The output of the narrow & dense CONV is inflated with zero channels to match the dimension of identity path. But the input of next block is fixed to be the original width, limiting the computation reduction.

### 3.4 Motivation

We summarize model search methods by four metrics in Table 1. To overcome the aforementioned challenges, we propose PaS to automatically search the model width. Besides, we propose a new pruning-deployment paradigm based on structural reparameterization, unleashing a new dimension of flexibility in per-layer width design.

## 4 Pruning as Search Algorithm

We first introduce our method and then discuss how our method can deal with the three challenges.

### 4.1 Depth-Wise Binary Convolution Layers

To automate channel pruning as width search, we formulate the per-layer pruning policy to be trainable along with regular network training, by creating a differentiable *depth-wise binary convolution* (DBC) layer for each pruned layer. Specifically, we insert a depth-wise $1 \times 1$ CONV layer following a CONV layer that is supposed to be pruned, as below,

$$\boldsymbol{a}_l = \boldsymbol{v}_l \odot (\boldsymbol{w}_l \odot \boldsymbol{a}_{l-1}) \tag{1}$$

where $\odot$ is the convolution operation. $\boldsymbol{w}_l \in R^{o \times i \times k \times k}$ is the weight parameters in $l$-th CONV layer, with $o$ output channels, $i$ input channels, and kernels of size $k \times k$. $\boldsymbol{a}_l \in R^{n \times o \times s \times s'}$ represents the output features of $l$-th layer (with the DBC layer), with $o$ channels and $s \times s'$ feature size. $n$ denotes batch size. $\boldsymbol{v}_l \in R^{o \times 1 \times 1 \times 1}$ is the DBC layer weights.

Each element in $\boldsymbol{v}_l$ corresponds to an output channel of $\boldsymbol{w}_l \odot \boldsymbol{a}_{l-1}$. Thus we use $\boldsymbol{v}_l$ as the pruning indicator and pruning is performed with reference to the magnitude of $\boldsymbol{v}_l$ elements. Then the problem of channel pruning is relaxed as binarization of $\boldsymbol{v}_l$. The binarization operations have zero derivatives, leading to difficulties for backpropagation. We propose to integrate Straight Through Estimator (STE) [Bengio *et al.*, 2013] as shown below to train DBC along with original network parameters.

$$\begin{aligned} &\textbf{Forward: } \boldsymbol{b}_l = \begin{cases} 1, \boldsymbol{v}_l > thres. \\ 0, \boldsymbol{v}_l \leq thres. \end{cases}, \\ &\textbf{Backward: } \frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_l} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_l} \end{aligned} \tag{2}$$

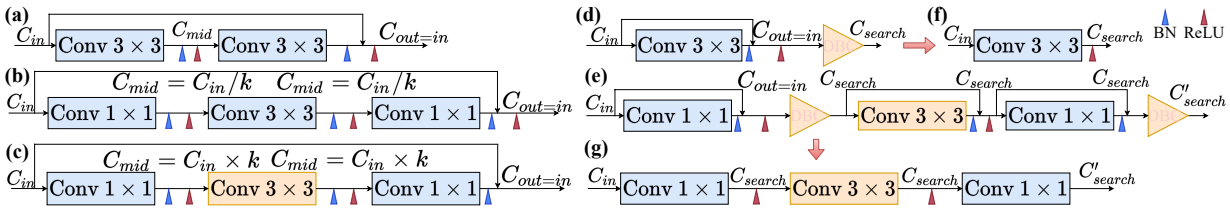


Figure 2: (a, b, c) are building blocks for ResNet18, ResNet50+ and MobileNet-V2. We can observe that only the width of middle layer is freely searchable, while the width of main path ($C_{out} = C_{in}$) are identical in the whole stage. On contrast in our reparameterization-based search, we construct building block with identity path to skip only one convolution, as in (d, e). During inference, identity path is merged so that arbitrary (smaller) widths can be set for each layer, as in (f, g).

where $\boldsymbol{b}_l \in \{0,1\}^{o\times1\times1\times1}$ is the binarized $\boldsymbol{v}_l$. The $thres$ is an adjustable threshold, and in our case it is simply set as $0.5$. With the DBC layers and the STE method, we can train the model parameters $\boldsymbol{W} = \{\boldsymbol{w}_l\}$ and the policy parameters $\boldsymbol{V} = \{\boldsymbol{v}_l\}$ simultaneously, and pruning policy is decoupled from weight or feature magnitudes.

STE method is originally proposed to avoid the non-differentiable problem in quantization task. We highlight that the benefits of integrating STE with DBC layers in pruning task is two-fold. First, we decouple the pruning policy from model parameter magnitudes. Second, the information in pruned channels is preserved since zeros in DBC layers block gradient flow. As a result, pruned channels are free to recover and contribute to accuracy as they originally did. While in most aforementioned pruning methods, weights in pruned layers are destroyed as they are forced to get close to zeros. Thus our DBC with STE outperforms other complicated strategies which relax the non-differentiable binary masks into sigmoid-like function, such as [Guo *et al.*, 2020; Guan *et al.*, 2022]. Our comprehensive evaluation demonstrates the effectiveness of the STE method on large-scale datasets with different architectures in various applications.

In order to deploy pruned sparse models, the next step is to convert sparse models to dense models by squeezing/-grouping the DBC layer based on the binary values. Let $\boldsymbol{b} = \{0\}^{o_0\times1\times1\times1} \oplus \{1\}^{o_1\times1\times1\times1}$, where $o_0$ and $o_1$ denote the number of zeros and ones, respectively, with $o_0 + o_1 = o$, and $\oplus$ refers to channel-wise concatenation. Thus we have

$$\begin{aligned}\boldsymbol{a}_l &= \boldsymbol{b}_l^{o\times1\times1\times1} \odot (\boldsymbol{w}_l^{o\times i\times k\times k} \odot \boldsymbol{a}_{l-1}) \\ &= (\{0\}^{o_0\times1\times1\times1} \cdot \boldsymbol{w}_l^{o_0\times i\times k\times k} \odot \boldsymbol{a}_{l-1}) \\ &\oplus (\{1\}^{o_1\times1\times1\times1} \cdot \boldsymbol{w}_l^{o_1\times i\times k\times k} \odot \boldsymbol{a}_{l-1}) \\ &= \boldsymbol{w}_l^{o_1\times i\times k\times k} \odot \boldsymbol{a}_{l-1}\end{aligned} \tag{3}$$

where zero channels are squeezed in the last equality.

## 4.2 Training Loss Function

With differentiable DBC layers, we can train and prune the model via SGD simultaneously with the loss function,

$$\min_{\boldsymbol{W},\boldsymbol{V}} \mathcal{L}(\boldsymbol{W},\boldsymbol{V}) + \beta \cdot \mathcal{L}_{reg}(\boldsymbol{V}) \tag{4}$$

where $\mathcal{L}_{reg}$ is the regularization term related to the computation complexity or on-chip latency. For simplicity, we take Multiply-Accumulate operations (MACs) as the constraint/-target rather than parameter number to estimate on-device execution cost more precisely. $\beta$ can weight the loss and stabilize training. $\mathcal{L}_{reg}$ can be simply defined as squared $\ell_2$ norm between current MACs and target MACs $\mathcal{C}$,

$$\mathcal{L}_{reg} = |\sum_l o'_l \times i_l \times s_l \times s'_l \times k^2 - \mathcal{C}|^2 \tag{5}$$

| Model | GMACs | Speed | Top1 |
|---|---|---|---|
| RepVGG-B1 | 11.8 | 826 | 78.37 |
| RepVGG-A2 | 5.1 | 1550 | 76.48 |
| ResNet-50 | 4.1 | 850 | 77.10 |
| RepVGG-B0 | 3.1 | 2041 | 75.14 |
| RepVGG-A1 | 2.4 | 2663 | 74.46 |
| RepVGG-A0 | 1.4 | 3677 | 72.41 |
| **PaS-A** | **4.3** | 1821 | **77.39** (±0.05) |
| **PaS-B** | **2.9** | 2313 | **75.86** (±0.19) |
| **PaS-C** | **2.4** | 2697 | **75.50** (±0.26) |
| MobileNet-V2×1.4 | 0.58 | 1704 | 74.09 |
| MobileNet-V2×1.0 | 0.30 | 2569 | 71.87 |
| MobileNet-V2×0.75 | 0.21 | 3052 | 69.95 |
| MNasNet 1.0 | 0.31 | 2542 | 73.46 |
| MNasNet 0.75 | 0.23 | 3435 | 71.71 |
| **PaS-light-A** | **0.88** | 2062 | **77.56** (±0.19) |
| **PaS-light-B** | **0.50** | 2817 | **74.90** (±0.18) |
| **PaS-light-C** | **0.34** | 3723 | **72.26** (±0.23) |

Table 2: PaS results on the proposed reparameterization-based design. All speed is measured with batch size 128, full precision, and 8 threads. The speed reported is in frame per second (FPS).

## 4.3 Simultaneous Pruning and Training for `C1`

By adopting DBC layers to introduce the pruning indicators and leveraging STE method to enable gradients back-propagation, our end-to-end channel pruning algorithm can train the model parameters and pruning indicators at the same time. The training is just like training a typical unpruned model until convergence. Compared with NAS to train each architecture with multiple epochs or other typical pruning methods to iteratively prune and train the model, our method can save tremendous training efforts.

## 4.4 DBC Layers as Indicators for `C2`

As discussed, we decouple pruning policy from parameter or feature magnitudes. With the DBC layers as pruning indicators, the benefits come two-fold. First, the indicators are trained with stochastic gradient descent which naturally enable exploration. Thus we jump out of one-shot magnitude-based decision and the pruning policy can be updated dynamically during training. Second, with STE, we zero out not only channels but also gradients during training. If a channel is determined to be pruned by DBC layer, its corresponding

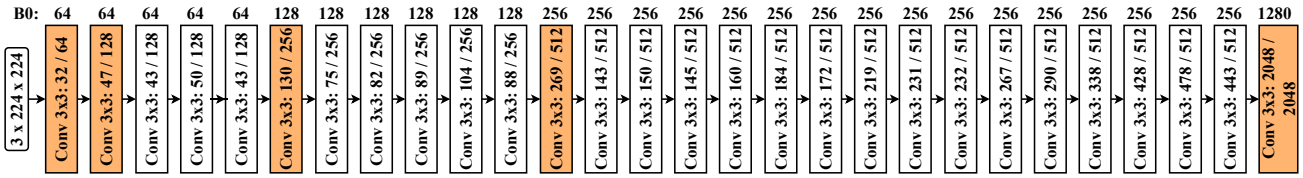


Figure 3: Architecture (width) of PaS-B, searching from RepVGG-B1. Orange block represents downsampling layers, e.g. CONV with stride= 2. We show remained and original channel numbers in each block (e.g., 47 channels remained within 128 channels for 47/128). Furthermore, we compare with RepVGG-B0 (the channel numbers on the top row) as they share same depth. More searched architecture on YoLACT for MS COCO, GANs for image generation are shown in Appendix A.2.

| Method | S.C. | GMACs | Top1 | Top5 | GMACs | Top1 | Top5 | GMACs | Top1 | Top5 |
|---|---|---|---|---|---|---|---|---|---|---|
| MetaPruning | / | 3.0 | 76.2 | / | 2.3 | 75.4 | / | 1.0 | 73.4 | / |
| AutoSlim | / | 3.0 | 76.0 | / | 2.0 | 75.6 | / | 1.0 | 74.0 | / |
| ThiNet | >>1750 | 2.9 | 75.8 | 90.7 | 2.1 | 74.7 | 90.0 | 1.2 | 72.1 | 88.3 |
| Uniform | / | 3.0 | 75.9 | 90.7 | 2.0 | 74.5 | 90.0 | 1.0 | 72.1 | 90.8 |
| EagleEye | 75 | 3.0 | 77.1 | 93.4 | 2.0 | 76.4 | 92.9 | 1.0 | 74.2 | 91.8 |
| DMCP | 120 | 2.8 | 76.7 | / | 2.2 | 76.2 | / | 1.1 | 74.4 | / |
| **Ours** | 60 | **3.0** | **77.6** ($\pm$0.07) | **93.4** | **2.0** | **76.7** ($\pm$0.05) | **93.1** | **1.0** | **74.8** ($\pm$0.12) | **92.0** |

Table 3: Accuracy of pruned ResNet50 on ImageNet, we provide results of top-1 and top-5 accuracy in percents (%). ResNet50 baseline network is 4.1 GMACs with 77.1% top-1 and 93.5% top-5 accuracy under our training configurations. S.C. refers to search cost and is measured by total GPU hours including candidate evaluation and training of the selected sub-network.

weights are not updated. As a result, the weights and performance of pruned layers are preserved thus are ready to recover anytime for re-evaluation. This distinguish our method from equal-penalty soft-mask ones which destroy origin weights.

### 4.5 Structural Reparameterization for `C3`

To deal with the residual constraints, RepVGG [Ding *et al.*, 2021] takes the identity path to skip only one convolution at a time, as a typical reparameterization instance. Thus, the identity path can be merged into the convolution during inference, outperforming same-level ResNet by a considerable margin in terms of inference speed, while preserving benefits of residual connections in gradient flow. Deriving from Eq. (1), we show the reparameterization as follows,

$$\begin{aligned} \boldsymbol{a}_l &= \boldsymbol{v}_l \odot (\boldsymbol{w}_l \odot \boldsymbol{a}_{l-1} + \boldsymbol{a}_{l-1}) \\ &= \boldsymbol{v}_l \odot ((\boldsymbol{w}_l + I) \odot \boldsymbol{a}_{l-1}) \\ &= \boldsymbol{v}_l \odot (\boldsymbol{m}_l \odot \boldsymbol{a}_{l-1}) \end{aligned} \tag{6}$$

where there is a residual connection besides $\boldsymbol{w}_l \odot \boldsymbol{a}_{l-1}$. We merge the identity path into CONV weights and obtain the merged weights $\boldsymbol{m}_l$. The DBC layer is added after the merged CONV $\boldsymbol{m}_l \odot \boldsymbol{a}_{l-1}$ rather than the original CONV $\boldsymbol{w}_l \odot \boldsymbol{a}_{l-1}$ to enable reparameterization, as shown in Fig. 1.

After training the pruning indicators and reparameterization, we zero out both pruned channel from CONV layer and its corresponding channel from identity path, so that the sparse block can be squeezed into a compact one for inference acceleration. The channel dimension of both the current layer's output channel and the next layer's input channel can be reduced, as shown in Fig. 1, Fig. 2 (d→f) and (e→g).

It is notable that these unique-width-per-layer architecture can not be created and trained directly. We demonstrate the searched architectures in section 5.1.

## 5 Experiment Results

All experiments are conducted on PyTorch 1.7 using NVIDIA RTX TITAN and GeForce RTX 2080Ti GPUs. To demonstrate the efficiency of *prune as search* method against brute NAS methods and sophisticated search to prune methods, we directly conducted experiments on large scale datasets or complex tasks including ImageNet for classification, MS COCO dataset for segmentation, Generative Adversarial Networks (GANs) for image generation.

### 5.1 PaS on ImageNet

ImageNet ILSVRC-2012 contains 1.2 million training images and 50k testing images. Note that all results are based on the standard resolution ($224 \times 224$) for fair comparison. Following standard data augmentation, we prune from pretrained model with weight decay setting to $3.05 \times 10^{-5}$, momentum as $0.875$. Learning rate is rewound to 0.4 for a total batch size of $1024$ synchronized on 8 GPUs. We search for 10 epochs, which is enough for per-layer pruning policy convergence as shown in Fig. 4. Then we freeze the policy (parameters in DBC) and anneal learning rate by cosine schedule for 50 epochs to achieve final accuracy. Thus we use a total of 60 epochs to deliver the compact well-trained model.

**PaS Networks.**

As demonstrated in Tab. 2, our searched networks outperform handcrafted design in terms of accuracy and inference speed by large margin under similar GMACs or speed. We provide 3 searched architectures of heavy model under different computation constraints. PaS-A outperforms RepVGG-A2 from the origin paper [Ding *et al.*, 2021] by $0.9\%$ higher top-1 accuracy with less computation cost (0.8 GMACs), because of the optimized layer widths. PaS-B targets on RepVGG-B0 which is a proportionally narrower version of RepVGG-B1. With 0.2 GMACs smaller computation cost, PaS-B outperforms RepVGG-B0 by $0.7\%$ top-1 accuracy. PaS-C achieves $1\%$ higher top-1 accuracy than RepVGG-A1 with the same computation complexity.

Besides accuracy, our proposed PaS-A, PaS-B and PaS-C achieve faster inference speed (in terms of frame per second).

| Yolact [Bolya *et al.*, 2019] | GMACs | Mask mAP | Box mAP | CycleGAN [Zhu *et al.*, 2017] | GMACs | FID ↓ |
|---|---|---|---|---|---|---|
| ResNet101-550 | 64.8 | 29.8 | 32.3 | Horse2zebra-ResNet | 56.8 | 61.5 |
| Uniform 1.7× | 37.8 | 22.1 | 23.6 | GC [Li *et al.*, 2020] | 2.67 | 65.1 |
| Magnitude 1.7× | 37.8 | 26.4 | 28.8 | CAT [Jin *et al.*, 2021] | 2.55 | 60.18 |
| PaS 1.7× | 37.8 | 29.9 | 32.3 | PaS-Large | 4.0 | 45.5 |
| PaS 4.0× | 16.2 | 23.2 | 25.4 | PaS-Small | 2.7 | 52.3 |

Table 4: Accuracy of pruned YoLACT on instance segmentation task, and CycleGAN with Mobile-ResNet backbone for image style transfer.

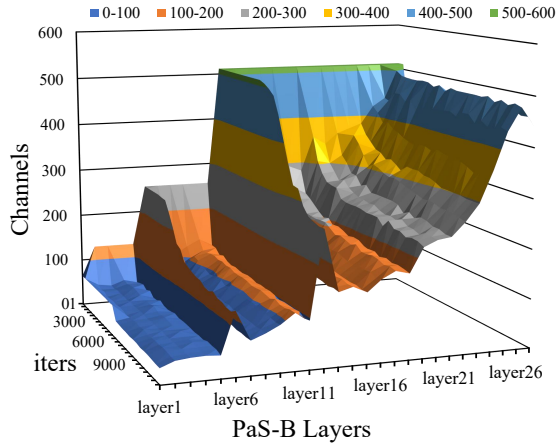


Figure 4: Convergence of pruning policy (searched width) in 10 training epochs on ImageNet. We take PaS-B as an example here.

Moreover, compared with ResNet-50 which has residual connections at inference time, PaS-A achieves $0.3\%$ higher top-1 accuracy and $114\%$ faster inference speed, demonstrating the potential of reparameterization-based backbones.

PaS-A, B, C exhibit high flexibility in layer width as shown in Fig.6. We also compared the searched architectures of PaS-B and RepVGG-B0 which share the same depth in Fig. 3. We can observe that the first few layers in PaS-B are narrower than those in RepVGG-B0, while the last ones are significantly wider. In classification task, doubling channel numbers when downsampling proves to be a good practice, as seen in the searched PaS-B. However, we should also gradually increase channel numbers within each stage, which is usually **not supported** in previous work due to residual constraints.

We also experiment over lightweight networks [Sandler *et al.*, 2018; Tan and Le, 2019; Tan *et al.*, 2019], which demonstrate significantly better parameter and computation efficiency and have become de facto choice on edge devices. We perform search on lightweight building blocks as shown in Tab. 2 and release a series of models in Appendix Fig.7, composed with $3 \times 3$ depth-wise convolutions and $1 \times 1$ regular convolutions. Our searched architecture exhibits high flexibility in layer width. Experiment results demonstrate that our searched network achieves best accuracy-complexity trade-off. Since we are investigating width optimization, in this work we do not employ sophisticated network elements including SE-block [Hu *et al.*, 2018], or activation functions [Ramachandran *et al.*, 2018]. Thus we only compare accuracy performance with plain-built baselines.

**Prune Traditional Models.**

Despite the newly proposed networks with arbitrary width, we also show the effectiveness of our PaS algorithm on common models. We validate our PaS on ResNet-50 [He *et al.*, 2016]. As shown in Tab. 3, PaS consistently outperforms prior arts by $0.3\%$ to $2.7\%$ top-1 accuracy under same computation constraint. Moreover, PaS just need to train once to deliver final accuracy, saving search cost (in GPU-hours) compared with other baselines with sophisticated searching process.

By comparing the pruned architecture of PaS and other baselines, we notice an interesting phenomenon that the first layer is pruned $50\%$ to $80\%$ with our method, which is significantly different from prior arts that claim it to be the most sensitive. On the contrary, the last layers are pruned less. We conclude this phenomenon as the last several layers extract high-level features and contribute more per-MACs-information.

### 5.2 PaS on Instance Segmentation and GAN

We evaluate the proposed method on instance segmentation with MS COCO dataset . We select the GPU real-time YoLACT [Bolya *et al.*, 2019] model as supernet. As shown in Tab. 4, the proposed PaS method can achieve $1.7\times$ total compression rate without degradation in mask mAP. Compared with other pruning methods such as uniform pruning or magnitude pruning, PaS can achieve better mask mAP and box mAP by a large margin under the same computation constraint. We also show the performance of PaS on unpaired image style transfer [Zhu *et al.*, 2017]. Specifically, we demonstrate horse2zebra dataset in Tab. 4. Though GAN suffers from unstable training, our PaS still achieves satisfactory compression results. Due to space limit, more detailed experiment settings and results on segmentation and GANs can be found in Appendix. Our PaS exhibits better generalization on complex tasks and datasets compared to prior arts.

### 5.3 Convergence of Pruning Policy

We demonstrate the robust convergence of PaS in Fig. 4. The pruning policy converges within a few epochs (less than 10). During PaS, we observe that pruned channels have the chance to recover and be re-evaluated, which proves our ability of exploration and exploitation. The policy converges to a stable point which proves the robustness of our PaS.

## 6 Conclusion

We propose *PaS* to optimize sub-network width with regular training, achieving high efficiency and robustness. Combining PaS with reparameterization, we generate a brand new series of reparameterization-based networks, which benefit from residual connections during training while enabling arbitrary width for each layer during inference, squeezing out an unrevealed dimension of width optimization.

## Acknowledgments

The research reported here was funded in whole or in part by the Army Research Office/Army Research Laboratory via grant W911-NF-20-1-0167 to Northeastern University. Any errors and opinions are not those of the Army Research Office or Department of Defense and are attributable solely to the author(s). This research is also partially supported by National Science Foundation CCF-1937500 and CCF-1901378.

## References

[Bengio *et al.*, 2013] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013.

[Bolya *et al.*, 2019] Daniel Bolya, Chong Zhou, et al. Yolact: Real-time instance segmentation. In *ICCV*, 2019.

[Ding *et al.*, 2021] Xiaohan Ding, Xiangyu Zhang, et al. Repvgg: Making vgg-style convnets great again. In *CVPR*, pages 13733–13742, 2021.

[Guan *et al.*, 2022] Yushuo Guan, Ning Liu, et al. Dais: Automatic channel pruning via differentiable annealing indicator search. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[Guo *et al.*, 2020] Shaopeng Guo, Yujie Wang, et al. Dmcp: Differentiable markov channel pruning for neural networks. In *CVPR*, 2020.

[Guo *et al.*, 2021] Yi Guo, Huan Yuan, et al. Gdp: Stabilized neural network pruning via gates with differentiable polarization. In *ICCV*, pages 5239–5250, October 2021.

[Han *et al.*, 2015] Song Han, Jeff Pool, et al. Learning both weights and connections for efficient neural network. In *NeurIPS*, 2015.

[Han *et al.*, 2016] Song Han, Huizi Mao, et al. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR*, 2016.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, et al. Deep residual learning for image recognition. In *CVPR*, 2016.

[He *et al.*, 2017] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017.

[He *et al.*, 2018] Yihui He, Ji Lin, et al. Amc: Automl for model compression and acceleration on mobile devices. In *ECCV*, 2018.

[Hu *et al.*, 2016] Hengyuan Hu, Rui Peng, et al. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint*, 2016.

[Hu *et al.*, 2018] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, 2018.

[Jin *et al.*, 2021] Qing Jin, Jian Ren, et al. Teachers do more than teach: Compressing image-to-image models. In *CVPR*, pages 13600–13611, 2021.

[Li *et al.*, 2019] Tuanhui Li, Baoyuan Wu, et al. Compressing convolutional neural networks via factorized convolutional filters. In *CVPR*, pages 3977–3986, 2019.

[Li *et al.*, 2020] Muyang Li, Ji Lin, et al. Gan compression: Efficient architectures for interactive conditional gans. In *CVPR*, pages 5284–5294, 2020.

[Liu *et al.*, 2017] Zhuang Liu, Jianguo Li, et al. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017.

[Liu *et al.*, 2018] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[Liu *et al.*, 2019] Zechun Liu, Haoyuan Mu, et al. Metapruning: Meta learning for automatic neural network channel pruning. In *ICCV*, 2019.

[Luo *et al.*, 2017] Jian-Hao Luo, Jianxin Wu, et al. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, 2017.

[Marnerides *et al.*, 2018] Demetris Marnerides, Thomas Bashford-Rogers, et al. Expandnet: A deep convolutional neural network for high dynamic range expansion from low dynamic range content. In *Computer Graphics Forum*, volume 37, pages 37–49. Wiley Online Library, 2018.

[Ramachandran *et al.*, 2018] Prajit Ramachandran, Barret Zoph, and Quoc Le. Searching for activation functions. In *ICLR*, 2018.

[Sandler *et al.*, 2018] Mark Sandler, Andrew Howard, et al. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

[Su *et al.*, 2021] Xiu Su, Shan You, et al. Bcnet: Searching for network width with bilaterally coupled network. *CoRR*, abs/2105.10533, 2021.

[Tan and Le, 2019] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, pages 6105–6114. PMLR, 2019.

[Tan *et al.*, 2019] Mingxing Tan, Bo Chen, et al. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, 2019.

[Wu *et al.*, 2019] Bichen Wu, Xiaoliang Dai, et al. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, 2019.

[Zhang *et al.*, 2018a] Tianyun Zhang, Shaokai Ye, et al. Structadmm: A systematic, high-efficiency framework of structured weight pruning for dnns. *arXiv preprint arXiv:1807.11091*, 2018.

[Zhang *et al.*, 2018b] Tianyun Zhang, Shaokai Ye, et al. Systematic weight pruning of dnns using alternating direction method of multipliers. *arXiv preprint arXiv:1802.05747*, 2018.

[Zhong *et al.*, 2018] Zhao Zhong, Junjie Yan, et al. Practical block-wise neural network architecture generation. In *CVPR*, 2018.

[Zhu *et al.*, 2017] Jun-Yan Zhu, Taesung Park, et al. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV*, pages 2223–2232, 2017.

[Zoph and Le, 2017] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.