Accelerate CNNs from Three Dimensions: A Comprehensive Pruning Framework

Wenxiao Wang [1 2] **Minghao Chen** [1] **Shuai Zhao** [1] **Long Chen** [3 4] **Jinming Hu** [1] **Haifeng Liu** [1] **Deng Cai** [1]
Xiaofei He [1] **Wei Liu** [2]

Abstract

Most neural network pruning methods, such as filter-level and layer-level prunings, prune the network model along one dimension (*depth*, *width*, or *resolution*) solely to meet a computational budget. However, such a pruning policy often leads to excessive reduction of that dimension, thus inducing a huge accuracy loss. To alleviate this issue, we argue that pruning should be conducted along three dimensions comprehensively. For this purpose, our pruning framework formulates pruning as an optimization problem. Specifically, it first casts the relationships between a certain model's accuracy and *depth/width/resolution* into a polynomial regression and then maximizes the polynomial to acquire the optimal values for the three dimensions. Finally, the model is pruned along the three optimal dimensions accordingly. In this framework, since collecting too much data for training the regression is very time-costly, we propose two approaches to lower the cost: 1) specializing the polynomial to ensure an accurate regression even with less training data; 2) employing iterative pruning and fine-tuning to collect the data faster. Extensive experiments show that our proposed algorithm surpasses state-of-the-art pruning algorithms and even neural architecture search-based algorithms.

1. Introduction

To deploy pre-trained Convolutional Neural Networks (CNNs) (Simonyan & Zisserman, 2015; He et al., 2016; Huang et al., 2017; Tan & Le, 2019) on resource-constrained mobile devices, plenty of methods (Ba & Caruana, 2014; Hinton et al., 2015; Liu et al., 2017; He et al., 2019; Frankle & Carbin, 2019) have been proposed for model acceleration. Among them, neural network pruning, which prunes redundant components (e.g., filters) of CNNs to cater for a computational budget, is one of the most popular and is the focus of this paper.

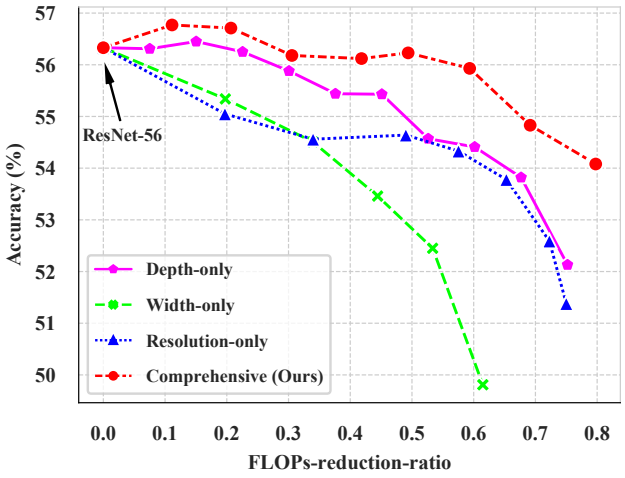

Figure 1. Accuracies of a base neural network model with different pruning policies on TinyImageNet. The base model is ResNet-56 (FLOPs-reduction-ratio = 0). $\mathcal{X}$-only means that the model is pruned only along $\mathcal{X}$ dimension, and "comprehensive" means that three dimensions are pruned comprehensively. A larger FLOPs-reduction-ratio implies a higher acceleration ratio.

Currently, the dominant pruning methods fall into three categories: (1) **layer-level pruning** (Wang et al., 2019b; Lin et al., 2019), which prunes redundant layers and reduces model's depth, (2) **filter-level pruning** (Liu et al., 2017; Li et al., 2017; Molchanov et al., 2017; He et al., 2019; Wang et al., 2019a; Luo et al., 2019; Kang & Han, 2020; Ye et al., 2020a), which prunes redundant filters and reduces model's width, and (3) **image-level pruning**[1] (Howard et al., 2017; Tan & Le, 2019; Han et al., 2020), which resizes images and reduces model's input resolution. These three kinds of pruning methods respectively focus on one single dimension (i.e., depth, width, or image resolution) that impacts on a model's computational cost.

[1]State Key Lab of CAD&CG, Zhejiang University, China [2]Tencent Data Platform, China [3]Columbia University, US [4]Tencent, China. This work was done when Long Chen was at Tencent. Correspondence to: Deng Cai <dengcai@gmail.com>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

[1]Though images are actually resized for acceleration, we will use term "pruning images" for simplicity in what follows.

Naturally, we raise an important question overlooked by much previous work: given a pre-trained neural network model, which dimension — depth, width, or resolution — should we prune to minimize the model's accuracy loss? In practice, users empirically choose a redundant dimension, which, however, often leads to a sub-optimal pruned model because of an inappropriate dimension choice. Even worse, excessive pruning of whichever dimension will cause an unacceptable loss, as shown in Figure 1. Instead, comprehensively pruning these three dimensions yields a much lower loss than solely pruning whichever dimension, demonstrated by Figure 1, therefore enabling model acceleration with much better quality.

In this paper, we propose a framework that prunes three dimensions comprehensively. Instead of solely pruning one dimension to reduce the computational cost, our framework first decides how much of each dimension should be pruned. To this end, we formulate model acceleration as an optimization problem. Precisely, given a pre-trained neural network model and a target computational cost, assuming that the pruned model's depth, width, and resolution are $d \times 100\%$, $w \times 100\%$, and $r \times 100\%$ of the original model, respectively, we seek the optimal (d, w, r) that maximizes the model's accuracy – a:

$$\max_{d,w,r} \; a := \mathcal{F}(d, w, r), \quad \text{s.t.} \;\; \mathcal{C}(d, w, r) = \tau, \tag{1}$$

where $\mathcal{F}(d, w, r)$ is a Model Accuracy Predictor (MAP). $\mathcal{C}(d, w, r)$ and τ represent the model's computational cost and its constraint, respectively. (Tan & Le, 2019) has designed a reasonable expression for $\mathcal{C}(d, w, r)$. However, designing a MAP manually is unachievable as its form can be arbitrarily complicated or even varies with the architecture (e.g., the MAPs for ResNet and MobileNet may be in different forms). Hence, we propose approximating the MAP via a polynomial regression, because polynomials can approximate arbitrary continuous functions according to Taylor's theorem. Specifically, we can formulate the MAP as a polynomial and collect a sufficient set of (d, w, r, a) as training data to estimate its parameters. Then, problem (1) can be solved with Lagrange's multiplier theorem, and the model is eventually pruned in terms of the optimized (d, w, r).

The main challenge that this framework encounters is that the polynomial regression requires tremendous training data (i.e., $\{(d, w, r, a)\}$), while the collection of the data is very costly because fetching each item of data, i.e., a (d, w, r, a), means training a new neural network model from scratch. To reduce both the collection time and model training cost, we improve the framework in two aspects: 1) A specialized polynomial is proposed whose weight tensor is replaced with its low-rank substitute. The low-rank weight tensor prevents the polynomial from overfitting and ensures an accurate regression even with limited training data. Further, as a bonus, the updated MAP owns a more concise form. 2) Given a pre-trained model, we prune and fine-tune it iteratively to acquire a series of new models and their corresponding $\{(d, w, r, a)\}$, which is much faster than training such new models from scratch.

Extensive experiments are conducted to show the superiority of our proposed pruning algorithm over the state-of-the-art pruning algorithms. Further, we compare against some algorithms that balance the size of three dimensions (depth, width, and resolution) from a Neural Architecture Search (NAS) perspective. The comparative results also show our advantages over them.

It is worth highlighting that the contributions of this work are three-fold:

- We propose to prune a model along three dimensions comprehensively and determine the optimal values for these dimensions by solving a polynomial regression and subsequently an optimization problem.
- To complete the regression process with an acceptable cost, we apply two approaches: 1) specializing a MAP adapting to the scenario of limited training data; 2) using iterative pruning and fine-tuning to collect data faster.
- We do extensive experiments to validate that our proposed algorithm outperforms state-of-the-art pruning and even NAS-based model acceleration algorithms.

2. Background and Related Work

Neural Network Pruning: In the early stage, neural network pruning is done at the weight-level (Han et al., 2016; Frankle & Carbin, 2019; Sehwag et al., 2020; Ye et al., 2020b; Frankle & Carbin, 2019; Lee et al., 2019). However, it needs specific libraries for sparse matrix calculation (e.g., cuSPARSE) to accelerate the inference, while these libraries' support on mobile devices is restricted. Nowadays, the most dominant pruning methods are at the filter-level, layer-level, or image-level, directly reducing the computational cost for all devices. Filter-level pruning (Liu et al., 2017; Li et al., 2017; Molchanov et al., 2017; He et al., 2018; 2019; Wang et al., 2019a; Kang & Han, 2020; Ye et al., 2020a; Li et al., 2020; Wang et al., 2020) compresses models by removing unimportant filters in CNNs, layer-level pruning (Wang et al., 2019b) does that by pruning redundant layers, and image-level pruning (Howard et al., 2017) saves computation by using small input images. They all receive great success in pruning CNNs. However, focusing on pruning one dimension solely also restricts their potentials.

Multi-Dimension Pruning: To the best of our knowledge, there are two methods (Wen et al., 2016; Lin et al., 2019) which prune models at both the filter- and layer-levels.

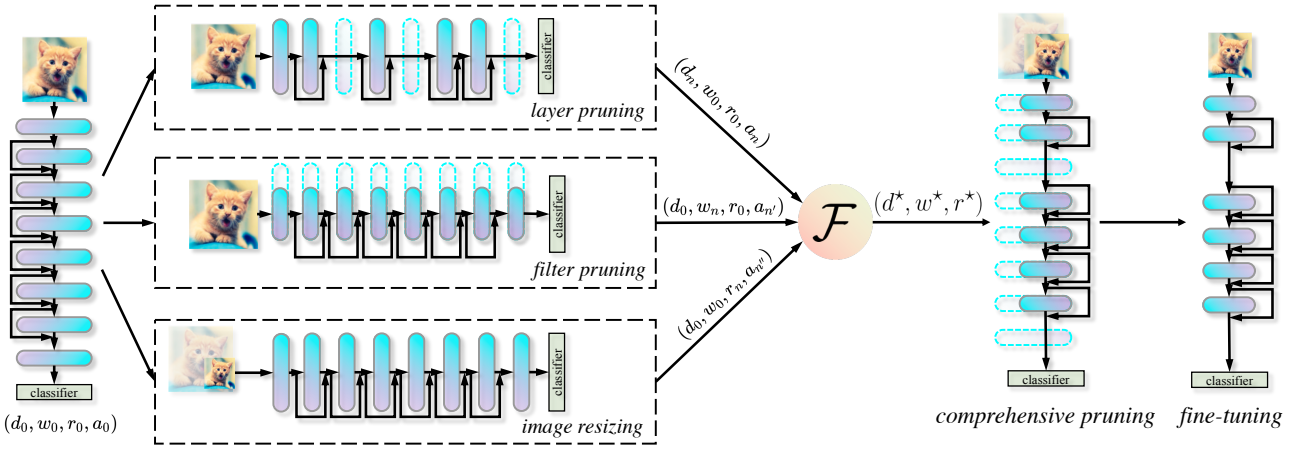

Figure 2. The pipeline of the proposed pruning framework. It first prunes a pre-trained model from three dimensions independently, yielding a set of (d_n, w_n, r_n, a_n) that is taken as training data. Then, the training data is used to fit our specialized MAP ($\mathcal{F}$) via a polynomial regression. The optimal $(d^\star, w^\star, r^\star)$ is then acquired by maximizing $\mathcal{F}$ subject to a computational cost constraint. Finally, the model will be pruned comprehensively in terms of $(d^\star, w^\star, r^\star)$.

Both of them train models with extra regularization terms and induce sparsity into the models. Then the filters or layers with much sparsity will be pruned with a slight loss incurred. However, the same method cannot be used for balancing image size because images do not contain trainable parameters, and there is no way to induce sparsity into the images. In contrast, our proposed framework can balance three dimensions comprehensively, yielding better model acceleration results than the above mentioned methods.

Pruning vs. NAS: Pruning and NAS (Pham et al., 2018; Gao et al., 2020; He et al., 2020; Tian et al., 2020; Liu et al., 2019; Tan & Le, 2019; Han et al., 2020; Howard et al., 2019; Huang et al., 2019; Zoph & Le, 2017) share the same goal, that is, maximizing a certain model's accuracy given a computational budget. However, their settings are very different: pruning shrinks the model from a pre-trained one, utilizing both pre-trained model's architecture and weights, while NAS searches the model(s) from scratch. Therefore, though several algorithms (Tan & Le, 2019; Han et al., 2020) also attempt to balance the three dimensions (i.e., depth, width, and resolution) of the model from a NAS perspective, they cannot be applied for pruning directly.

3. Proposed Framework

3.1. Preliminaries

For a model M, we define $\mathcal{D}(M)$, $\mathcal{W}(M, l)$, and $\mathcal{R}(M)$ as its depth, width, and input resolution. Specifically, $\mathcal{D}(M)$ represents the number of blocks[2] that M contains; $\mathcal{W}(M, l)$ denotes the number of filters of a certain layer l in the model M; $\mathcal{R}(M)$ is the side length of M's input image. Given a pre-trained model M_0, we also define d_n, w_n, and r_n of a pruned model M_n as:

$$d_n = \frac{\mathcal{D}(M_n)}{\mathcal{D}(M_0)},\ w_n = \frac{\mathcal{W}(M_n, l)}{\mathcal{W}(M_0, l)},\ r_n = \frac{\mathcal{R}(M_n)}{\mathcal{R}(M_0)}. \quad (2)$$

For filter pruning, following previous work (Luo et al., 2019; He et al., 2019; Lin et al., 2020), we prune all layers with the same ratio, so w_n of a model has no concern with the choice of layer l. Further, for a pruning task, it is easy to know: $d_n, w_n, r_n \in (0, 1]$ and $d_0 = w_0 = r_0 = 1$.

The pipeline of our proposed pruning framework is introduced in Figure 2. Unlike previous work that prunes one dimension solely, we first look for a pruning policy (i.e., how much of each dimension should be pruned) which aims to maximize the model's accuracy in Section 3.2. Then, we depict the process of pruning and fine-tuning a target model in terms of pruning policy in Section 3.3.

3.2. Model Acceleration as Optimization

3.2.1. Formulation

Given a CNN architecture, the model's depth, width, and image resolution are three key aspects that affect both the model's accuracy and its computational cost. Thus, model acceleration can be formulated as the following problem:

$$\begin{aligned} d^\star, w^\star, r^\star = &\arg\max_{d,w,r} \mathcal{F}(d, w, r; \Theta) \\ &\text{s.t. } \mathcal{C}(d, w, r) = T \times \mathcal{C}(d_0, w_0, r_0), \end{aligned} \quad (3)$$

where $\mathcal{F}(d, w, r; \Theta)$ is a **Model Accuracy Predictor (MAP)**, which predicts the model's accuracy given (d, w, r).

[2] E.g., Conv-BN-ReLu blocks and residual blocks.

Θ contains the parameters of the MAP. $\mathcal{C}(d, w, r)$ represents the computational cost (e.g., FLOPs) of a model. $T \in (0, 1)$ implies that the pruned model's computational cost is T proportion of the original model. Problem (3) can be solved using Lagrange's multiplier theorem once $\mathcal{F}(d, w, r)$ and $\mathcal{C}(d, w, r)$ are known. Following (Tan & Le, 2019) in which a model's computational cost is proportional to d, w^2, and r^2, we re-define $\mathcal{C}(d, w, r)$ as:

$$\mathcal{C}(d, w, r) = dw^2r^2. \tag{4}$$

However, designing a MAP manually is unachievable as its form can be arbitrarily complicated, and different architectures may own different forms. An intuitive idea is resorting to a polynomial regression because any continuous function can be approximated with polynomials according to Taylor's theorem. Specifically, we can train N models with different (d, w, r), attaining their accuracy a, and fit the MAP with a polynomial by using $\{(d_n, w_n, r_n, a_n)\}_{n=1}^{N}$ as training data. However, the regression process requires hundreds of data items (d_n, w_n, r_n, a_n) for training an accurate regression, whereas fetching each item of that data needs us to train a new model from scratch, which is very resource-inefficient and time-consuming. To overcome this obstacle, on the one hand, we specialize a MAP that ensures an accurate regression even with less training data in Section 3.2.2. On the other hand, we expedite acquiring each data item (d_n, w_n, r_n, a_n) by employing iterative pruning and fine-tuning in Section 3.2.3.

3.2.2. Specialized MAP

The polynomial-shaped MAP can be represented as:

$$\mathcal{F}(d, w, r; \Theta) = \sum_{i,j,k=0}^{\mathcal{K}} \theta_{ijk} d^i w^j r^k, \tag{5}$$

where $\Theta \in \mathbb{R}^{(\mathcal{K}+1)\times(\mathcal{K}+1)\times(\mathcal{K}+1)}$ is a tensor, and all θ_{ijk} are its elements[3]. Without any constraint on Θ, the polynomial can be highly flexible and expressive. However, high flexibility also makes it easy to overfit (Bishop, 2006), especially when the training data (i.e., $\{(d, w, r, a)\}$) is scarce. To avoid overfitting and ensure an accurate regression with limited training data, a relatively simple MAP with less flexibility and expressiveness is needed. We achieve this by restricting the rank[4] of Θ during the regression process, i.e., Θ in the MAP is replaced by its low-rank substitute. Formally, for Θ of rank $\widetilde{\mathcal{R}}$, its $\mathcal{R}$-rank substitute ($\mathcal{R} < \widetilde{\mathcal{R}}$) and elements are defined as (Kolda & Bader, 2009):

$$\Theta \approx \sum_{q=1}^{\mathcal{R}} \vec{s_q} \otimes \vec{u_q} \otimes \vec{v_q}, \quad \theta_{ijk} \approx \sum_{q=1}^{\mathcal{R}} s_{qi} u_{qj} v_{qk}, \tag{6}$$

[3]For the convenience of expression, we assume the same highest degree $\mathcal{K}$ for d, w, and r, though conclusions in this section still hold when they have different $\mathcal{K}$.

[4]The definition of tensor rank is the same as (Bourbaki, 2003).

Algorithm 1 Iterative Pruning and Fine-tuning

input pre-trained M_0, rounds rds, pruning setting T
initialize $train\_data = \{(d_0, w_0, r_0, a_0)\}$
function PruneAlong($dimension$, x_0, x_{min})
  for $n = 1$ **to** rds **do**
    $x_n = x_{n-1} - \frac{x_0 - x_{min}}{rds}$
    pruning M_{n-1} **along** $dimension$ **to** $x_n \rightarrow M_n$
    fine-tuning $M_n \rightarrow (d_n, w_n, r_n, a_n)$
    add (d_n, w_n, r_n, a_n) **to** $train\_data$
  end for
end function
$d_{min} = Td_0, w_{min} = \sqrt{T}w_0, r_{min} = \sqrt{T}r_0$
PruneAlong("depth", d_0, d_{min})
PruneAlong("width", w_0, w_{min})
PruneAlong("resolution", r_0, r_{min})
return $train\_data$

in which $\otimes$ represents outer product, and $\vec{s_q}, \vec{u_q}, \vec{v_q} \in \mathbb{R}^{\mathcal{K}+1}$ represent $(\mathcal{K}+1)$-dimensional vectors, e.g., $\vec{s_q} = [s_{q0}, s_{q1}, \cdots, s_{q\mathcal{K}}]^\top$. Then, replacing θ_{ijk} in Eq. (5) yields:

$$\begin{aligned}
\mathcal{F}(d, w, r; \Theta) &\approx \sum_{i,j,k=0}^{\mathcal{K}} \sum_{q=1}^{\mathcal{R}} s_{qi} u_{qj} v_{qk} d^i w^j r^k \\
&= \sum_{q=1}^{\mathcal{R}} \sum_{i,j,k=0}^{\mathcal{K}} (s_{qi} d^i)(u_{qj} w^j)(v_{qk} r^k) \\
&= \sum_{q=1}^{\mathcal{R}} \sum_{i=0}^{\mathcal{K}} s_{qi} d^i \sum_{j=0}^{\mathcal{K}} u_{qj} w^j \sum_{k=0}^{\mathcal{K}} v_{qk} r^k \\
&= \sum_{q=1}^{\mathcal{R}} \mathcal{H}(d; \vec{s_q}) \mathcal{H}(w; \vec{u_q}) \mathcal{H}(r; \vec{v_q}),
\end{aligned} \tag{7}$$

in which $\mathcal{H}$ represents univariate polynomial. In practice, we take Eq. (7) as our MAP and control its flexibility by adjusting $\mathcal{R}$. A smaller $\mathcal{R}$ indicates a simpler MAP. Empirically, we find that $\mathcal{R} = 1$ is enough for achieving an accurate regression in most cases, which provides our MAP with a highly succinct form. We also verify through experiments that $\mathcal{R} = 1$ makes sense because it accords with the prior of the MAP (Section 4.3).

3.2.3. Fast Data Collection

To collect data used for MAP's regression, instead of training many models with different (d, w, r) from scratch, we apply iterative pruning and fine-tuning to acquire the data.

Iterative Pruning and Fine-tuning: As shown in Algorithm 1, the pre-trained model M_0 is pruned along three dimensions independently. At each dimension, we iteratively apply pruning and fine-tuning on M_0 to generate many models, and the configurations $\{(d_n, w_n, r_n, a_n)\}$ of

these models are collected for the MAP's regression. d_{min} in Algorithm 1 indicates that if we reduce the model's depth to d_{min}, the computational cost constraint T can be fulfilled without pruning the model's width and input resolution. It is easy to deduce that the optimal $d^\star \geq d_{min}$. Likewise, w_{min} and r_{min} in Algorithm 1 are minimal possible values for w and r, respectively.

Compared with training models from scratch, our data collection strategy enjoys two advantages: 1) A pruned pre-trained model converges much faster than the one training from scratch, thus taking much less time to obtain a new model. 2) Besides the finally pruned model, iterative pruning yields several intermediate models as well as their configurations $\{(d_n, w_n, r_n, a_n)\}$, which can also be used for the MAP's regression.

3.2.4. Optimizing the MAP

With the collected data, we fit the MAP by using a regression algorithm. Then, the optimal $(d^\star, w^\star, r^\star)$ satisfies Eq. (8) according to Lagrange's multiplier theorem, where λ is the Lagrange multiplier.

$$\begin{cases} dw^2r^2 - T \times d_0 w_0^2 r_0^2 = 0 \\ \sum_{q=1}^{\mathcal{R}} \mathcal{H}'(d; \vec{s_q})\mathcal{H}(w; \vec{u_q})\mathcal{H}(r; \vec{v_q}) + \lambda w^2 r^2 = 0 \\ \sum_{q=1}^{\mathcal{R}} \mathcal{H}(d; \vec{s_q})\mathcal{H}'(w; \vec{u_q})\mathcal{H}(r; \vec{v_q}) + 2\lambda dwr^2 = 0 \\ \sum_{q=1}^{\mathcal{R}} \mathcal{H}(d; \vec{s_q})\mathcal{H}(w; \vec{u_q})\mathcal{H}'(r; \vec{v_q}) + 2\lambda dw^2 r = 0 \end{cases} \tag{8}$$

3.3. Comprehensive Pruning and Fine-tuning

Leveraging the optimal $(d^\star, w^\star, r^\star)$, filter-level pruning and layer-level pruning are applied to prune a pre-trained model M_0 to the target $d^\star$ and $w^\star$, and then the model is fine-tuned with images of size $r^\star$. During the entire pruning process, layer-pruning first and filter-pruning first are both viable and yield the same pruned model. Without loss of generality, we describe the pruning process by assuming layer-pruning first, and the concrete steps are as follows:

Pruning Layers: Following DBP (Wang et al., 2019b), we put a linear classifier after each layer of model M_0 and test its accuracy on the evaluation dataset. The accuracy of each linear classifier indicates the discrimination of its corresponding layer's features. Further, each layer's discrimination enhancement compared with its preceding layer is seen as the importance of the layer. With this importance metric, we pick out the least important $(1 - d^\star/d_0) \times 100\%$ layers and remove them from M_0, yielding M_{p_1}.

Pruning Filters: Filter-level pruning is performed over M_{p_1}. In particular, we use the scaling factor of BN layers as the importance metric, just like Slimming (Liu et al., 2017). However, different from Slimming that compares the importances of all filters globally, we only compare the importances of filters in the same layer, and the least important $(1 - w^\star/w_0) \times 100\%$ filters of each layer will be pruned. Through such a modification, the pruned ratios of all layers are kept the same. Assume the model after filter-pruning to be M_{p_2}.

Fine-tuning with Smaller Images: After pruning, the pruned model M_{p_2} is fine-tuned with images of size $r^\star$. The images are resized by bilinear down-sampling, which is the most common down-sampling scheme for images. The model will be fine-tuned with a small learning rate till convergence, leading to the finally pruned model M_p.

4. Experiments

4.1. Experimental Settings

Datasets: We take three popular datasets as testbeds of our algorithm: CIFAR-10 (Krizhevsky et al., 2009), Tiny-ImageNet (Wu et al., 2017), and ImageNet (Russakovsky et al., 2015). These three datasets differ in their image-resolutions (32×32 to 224×224), number of classes (10 to 1000), and scale of datasets (50K to 1000K images). For all the datasets, images are augmented by symmetric padding, random clipping, and randomly horizontal flip, all of which are common (He et al., 2016; Howard et al., 2017; Wang et al., 2019a) augmentation methods for these datasets.

Architectures: We test our algorithm on three popular network architectures: ResNet (He et al., 2016), DenseNet (Huang et al., 2017), and EfficientNet (Tan & Le, 2019). Their basic blocks vary from residual blocks to densely connected blocks and NAS-searched blocks, representing three of the most popular designs for deep CNNs.

Evaluation Protocol: Following the conventions of previous work (Li et al., 2020; Lin et al., 2020; Ye et al., 2020a), we take the accuracy, parameters-reduction-ratio (Prr), and FLOPs-reduction-ratio (Frr) as the evaluation protocol of our model acceleration algorithm. Prr and Frr are defined as Eq. (9), where M_0 and M_p represent the base model and the pruned model, respectively.

$$Prr = 1 - \frac{\textit{Params}(M_p)}{\textit{Params}(M_0)}, Frr = 1 - \frac{\textit{FLOPs}(M_p)}{\textit{FLOPs}(M_0)}. \tag{9}$$

Compared Algorithms: The compared algorithms fall into three categories: (1) Algorithms solely pruning the model along one dimension (i.e., depth, width, or resolution), including $\mathcal{R}$-only (Howard et al., 2017), $\mathcal{W}$-only (Liu et al., 2017), FPGM (He et al., 2019), DBP (Wang et al., 2019b), PScratch (Wang et al., 2020), DHP (Li et al., 2020),

and HRank (Lin et al., 2020); (2) Algorithms that prune along multi-dimensions, such as GAL (Lin et al., 2019); (3) NAS-based algorithms, including EfficientNet (Tan & Le, 2019) and TinyNet (Han et al., 2020), which balance the size of the three dimensions from the NAS perspective.

Training Settings: For base models trained on CIFAR-10, we set batch size to 64 for DenseNet and 128 for ResNet, respectively. Weight decay is set to 10^{-4}. The models are trained for 160 epochs with the learning rate starting from 0.1 and divided by 10 at epochs 80 and 120. These are all the most common training settings (He et al., 2016; Howard et al., 2017; Wang et al., 2019a) for models trained on CIFAR-10. For ResNet and DenseNet trained on TinyImageNet and ImageNet, batch size is set to 256, and weight decay is 10^{-4}. Models are trained for 100 epochs. The learning rate is set to 0.1 at the beginning and is multiplied by 0.1 at epochs 30, 60, and 90. For EfficientNet, we apply the same training policy as (Han et al., 2020), which is also the most common for EfficientNet implemented with PyTorch (Paszke et al., 2017).

Regression and Pruning Settings: The MAP's hyper-parameters are set to $\mathcal{R} = 1$ and $\mathcal{K} = 3$ in our pruning experiments. When collecting training data (i.e., $\{(d_n, w_n, r_n)\}_{n=1}^{N}$) for the polynomial regression, the model is pruned along each dimension for four times (i.e., $rds = 4$ in Algorithm 1). ResNet and DenseNet trained on CIFAR-10 are fine-tuned for 40 epochs at each round, and for 80 epochs after comprehensive pruning. Therefore, the data collection process consumes as much time as training 3 models (training one model from scratch costs 160 epochs). Similarly, models trained on TinyImageNet and ImageNet are fine-tuned for 30 epochs at each round of the iterative pruning process. Thus, it takes about the same time as training 3.6 models for the data collection process. The finally pruned models trained on TinyImageNet and ImageNet are fine-tuned for 60 epochs after comprehensive pruning.

4.2. Results and Analyses

Results on CIFAR-10 and TinyImageNet: The experimental results on CIFAR-10 and TinyImageNet are shown in Table 1. As we can see, $\mathcal{W}$-only induces greater loss than $\mathcal{D}$-only for ResNet-32 and ResNet-56, while for ResNet-101, the situation is opposite. In other words, the importance of different dimensions lies in the original size of *depth*, *width*, and *resolution*, and we cannot deduce it from a simple prior, which further shows the essentiality of our algorithm. We balance the size of the three dimensions dynamically and always achieve better results than pruning one or two dimensions. The most competitive opponent of our algorithm is DHP, which achieves similar accuracy and Frr to our algorithm for ResNet-56 trained on both datasets. However, we show higher accuracy than DHP for DenseNet-40 on CIFAR-10 (94.54% vs. 93.94%), for ResNet-101 (65.27% vs. 64.82%) on TinyImageNet, and for DenseNet-100 (60.22% vs. 59.40%) on TinyImageNet with similar Prr and Frr, which sheds light on the robustness of our algorithm across different architectures and datasets.

Results on ImageNet: Experiments with ImageNet are done on ResNet-50 and DenseNet-121. From Table 2, we can see that our algorithm achieves 0.45% higher accuracy on ResNet-50 than the state-of-the-art algorithms (DHP and PScratch) with the same Frr. The improvement on DenseNet-121 is marginal compared with $\mathcal{W}$-only, because our algorithm also prunes DenseNet-121 mainly along width dimension, which indicates that DenseNet-121's width is *large* and has much redundancy. By contrast, images do not need to be pruned. With a comprehensive consideration, our algorithm also deems that we should mainly prune filters of DenseNet-121 for acceleration. Therefore, it produces similar pruning results to filter-level pruning. However, the results do not imply that our algorithm is powerless. On the contrary, **a pruning policy with a comprehensive consideration is always better than an arbitrary one, though they may produce similar results sometimes**.

Comparison with NAS: Algorithms that balance the three dimensions (i.e., depth, width, and resolution) in a NAS manner are also compared, and the results are shown in Table 3. GPU-days is the most common metric to evaluate the search cost of NAS algorithms, which indicates natural days they spend if running with only one GPU. Both EfficientNet (Tan & Le, 2019) and TinyNet (Han et al., 2020) employ so many resources in searching the optimal $(d^\star, w^\star, r^\star)$, while we do not have enough GPUs to reproduce their searching process. Thus, the results of EfficientNet and TinyNet are both drawn from (Han et al., 2020), and their search costs are estimated through the number of models they trained. For example, training an EfficientNet for 300 epochs takes about 26 hours with 8×V100 GPUs, while TinyNet requires to train 60 EfficientNet models from scratch. Hence, its search cost is about 520 GPU days. Instead, our algorithm only spends about $\frac{1}{25}$ as much time as TinyNet on searching but achieves similar accuracy.

4.3. Ablation Study

Rank of Θ: In Section 3.2.2, our proposed MAP is

$$\mathcal{F}(d, w, r; \Theta) = \sum_{q=1}^{\mathcal{R}} \mathcal{H}(d; \vec{s_q}) \mathcal{H}(w; \vec{u_q}) \mathcal{H}(r; \vec{v_q}), \quad (10)$$

where the rank of Θ is less than or equal to $\mathcal{R}$. Experimentally, we find that $\mathcal{R} = 1$ works well in most cases. To further explore this interesting phenomenon, ResNets with different (d, w, r) are trained on CIFAR-10. The base model (i.e., $(d, w, r) = (1.0, 1.0, 1.0)$) is ResNet-32 with

Table 1. Pruning results on CIFAR-10 and TinyImageNet. $\mathcal{D}$, $\mathcal{W}$, and $\mathcal{R}$ indicate whether the model will be pruned along depth, width, and resolution dimension, respectively. "Acc. Drop" means the accuracy loss induced by pruning (smaller is better). Results with † are drawn from original papers, and the others are run with their published code with slight modifications. Our algorithm achieves **smaller accuracy losses than the others with similar Prr and Frr.**

Dataset	Architecture	Algorithm	$\mathcal{D}$	$\mathcal{W}$	$\mathcal{R}$	Baseline	Accuracy	Acc. Drop	Prr	Frr
CIFAR-10	ResNet-32	$\mathcal{R}$-only (Howard et al., 2017)			✓	93.18%	90.19%	2.99%	-	0.52
		$\mathcal{W}$-only (Liu et al., 2017)		✓		93.18%	92.16%	1.02%	0.47	0.47
		$\mathcal{D}$-only DBP (Wang et al., 2019b)	✓			93.18%	92.65%	0.53%	0.28	0.48
		GAL (Lin et al., 2019)	✓	✓		93.18%	91.72%	1.46%	0.39	0.50
		FPGM† (He et al., 2019)		✓		92.63%	92.31%	0.32%	-	0.42
		PScratch (Wang et al., 2020)		✓		93.18%	92.18%	1.00%	-	0.50
		Ours	✓	✓	✓	93.18%	**93.27%**	**-0.09%**	0.38	0.49
	ResNet-56	$\mathcal{R}$-only (Howard et al., 2017)			✓	93.69%	92.00%	1.69%	-	0.51
		$\mathcal{W}$-only (Liu et al., 2017)		✓		93.69%	92.97%	0.72%	0.50	0.50
		$\mathcal{D}$-only DBP (Wang et al., 2019b)	✓			93.69%	93.27%	0.42%	0.40	0.52
		GAL† (Lin et al., 2019)	✓	✓		93.26%	93.38%	**-0.12%**	0.12	0.38
		FPGM† (He et al., 2019)		✓		93.59%	93.26%	0.33%	-	0.52
		PScratch† (Wang et al., 2020)		✓		93.23%	93.05%	0.18%	-	0.50
		HRank† (Lin et al., 2020)		✓		93.26%	93.17%	0.09%	0.42	0.50
		DHP (Li et al., 2020)		✓		93.65%	93.58%	0.07%	0.42	0.49
		Ours	✓	✓	✓	93.69%	**93.76%**	**-0.07%**	0.40	0.50
	DenseNet-40	$\mathcal{R}$-only (Howard et al., 2017)			✓	94.59%	92.88%	1.71%	-	0.53
		$\mathcal{W}$-only (Liu et al., 2017)		✓		94.59%	94.26%	0.33%	0.65	0.65
		$\mathcal{D}$-only DBP (Wang et al., 2019b)	✓			94.59%	94.02%	0.57%	0.60	0.46
		GAL† (Lin et al., 2019)	✓	✓		94.81%	94.50%	0.31%	0.57	0.55
		HRank† (Lin et al., 2020)		✓		94.81%	93.68%	1.13%	0.54	0.61
		DHP† (Li et al., 2020)		✓		94.74%	93.94%	0.80%	0.36	0.62
		Ours	✓	✓	✓	94.59%	**94.54%**	**0.05%**	0.66	0.66
TinyImageNet	ResNet-56	$\mathcal{R}$-only (Howard et al., 2017)			✓	56.55%	54.64%	1.91%	-	0.49
		$\mathcal{W}$-only (Liu et al., 2017)		✓		56.55%	52.45%	4.10%	0.54	0.53
		$\mathcal{D}$-only DBP (Wang et al., 2019b)	✓			56.55%	55.57%	0.98%	0.25	0.53
		GAL (Lin et al., 2019)	✓	✓		56.55%	55.87%	0.68%	0.32	0.52
		DHP (Li et al., 2020)		✓		56.55%	55.82%	0.73%	0.46	0.55
		Ours	✓	✓	✓	56.55%	**56.04%**	**0.51%**	0.34	0.59
	ResNet-101	$\mathcal{R}$-only (Howard et al., 2017)			✓	64.83%	55.48%	9.35%	-	0.75
		$\mathcal{W}$-only (Liu et al., 2017)		✓		64.83%	63.47%	1.36%	0.75	0.75
		$\mathcal{D}$-only DBP (Wang et al., 2019b)	✓			64.83%	61.35%	3.48%	0.76	0.77
		GAL (Lin et al., 2019)	✓	✓		64.83%	64.33%	0.50%	0.45	0.76
		DHP (Li et al., 2020)		✓		64.83%	64.82%	0.01%	0.50	0.75
		Ours	✓	✓	✓	64.83%	**65.27%**	**-0.44%**	0.51	0.75
	DenseNet-100	$\mathcal{R}$-only (Howard et al., 2017)			✓	61.34%	56.97%	4.37%	-	0.75
		$\mathcal{W}$-only (Liu et al., 2017)		✓		61.34%	59.56%	1.78%	0.75	0.75
		$\mathcal{D}$-only DBP (Wang et al., 2019b)	✓			61.34%	58.44%	2.90%	0.65	0.78
		GAL (Lin et al., 2019)	✓	✓		61.34%	59.03%	2.31%	0.78	0.70
		DHP (Li et al., 2020)		✓		61.34%	59.40%	1.94%	0.73	0.73
		Ours	✓	✓	✓	61.34%	**60.22%**	**1.12%**	0.73	0.75

images of size 32, and the results are plotted in Figure 3. Observations from the first three sub-figures are shown in their titles. We can deduce from these observations[5]:

$$\mathcal{F}(d, w, r; \Theta) \approx \mathcal{H}(d; \vec{s_q})\mathcal{H}(w; \vec{u_q})\mathcal{H}(r; \vec{v_q}), \qquad (11)$$

which coincides with Eq. (10) once $\mathcal{R} = 1$. In other words, three variables (d, w, r) in the MAP can be approximately separated from each other. We also test the MAP with different $\mathcal{R}$, as shown in the 4*th* sub-figure of Figure 3. The MAP with larger $\mathcal{R}$ yields similar $(d^\star, w^\star, r^\star)$ to that of $\mathcal{R} = 1$, which also indicates that $\mathcal{R} = 1$ is enough for obtaining a well-performed MAP.

[5]More results and the proof are put in our supplementary material.

Other Methods of Avoiding Overfitting: Besides restricting the rank of Θ, we also try two extra methods of avoiding overfitting, i.e., decreasing the degree $\mathcal{K}$ of polynomials and applying regression with regularization terms. The results are reported in Table 4. Specifically, a set of 13 items (d_n, w_n, r_n, a_n) is used to fit the MAP, and a set of the other 80 items is used for evaluation. All data is collected with ResNet-56 trained on CIFAR-10. Training error and evaluation error are both reported. As we can see, normal polynomial regression induces severe overfitting and high evaluation loss, and ℓ_2-regularization has a limited effect on dealing with the overfitting issue. Still, lowering the degree of polynomials is not a wise choice because it makes the polynomials fail to converge even on training

Table 2. Pruning Results on ImageNet. The improvement on DenseNet-121 is marginal because of DenseNet-121's property. Detailed reasons are described in Section 4.2.

Algorithm	$\mathcal{D}$	$\mathcal{W}$	$\mathcal{R}$	Accuracy	Prr	Frr
ResNet-50 (76.15%)						
$\mathcal{R}$-only (Howard et al., 2017)			✓	71.56%	-	0.50
$\mathcal{W}$-only (Liu et al., 2017)		✓		74.52%	0.50	0.50
DBP (Wang et al., 2019b)	✓			73.92%	0.56	0.50
GAL† (Lin et al., 2019)	✓	✓		71.95%	0.17	0.43
FPGM† (He et al., 2019)		✓		74.83%	-	0.54
PScratch† (Wang et al., 2020)		✓		75.45%	0.64	0.50
HRank† (Wang et al., 2020)		✓		74.98%	0.37	0.44
DHP (Li et al., 2020)		✓		75.45%	0.54	0.50
Ours	✓	✓	✓	**75.90%**	0.53	0.50
DenseNet-121 (75.01%)						
$\mathcal{R}$-only (Howard et al., 2017)			✓	73.07%	-	0.51
$\mathcal{W}$-only (Liu et al., 2017)		✓		73.58%	0.51	0.51
DBP (Wang et al., 2019b)	✓			68.08%	0.66	0.37
Ours	✓	✓	✓	**73.68%**	0.48	0.51

Table 3. Comparison with NAS-based model acceleration algorithms. They all take EfficientNet-B0 as the baseline model. GPU-days is measured with NVIDIA V100. Note that training an EfficientNet from scratch costs about 8.7 GPU days.

Algorithm	Params	FLOPs	Top-1/Top-5 Acc.	Search Cost (GPU days)
EfficientNet-B0	5.3M	387M	76.7%/93.2%	-
TinyNet-A†	5.1M	339M	**76.8%/93.3%**	$\sim$ 520
Ours	5.1M	**314M**	**76.8%/93.3%**	26
EfficientNet-B^{-1}	3.6M	201M	74.7%/92.1%	-
TinyNet-B†	3.7M	202M	75.0%/92.2%	$\sim$ 520
Ours	3.6M	**198M**	**75.2%/92.7%**	24
EfficientNet-B^{-2}	3.0M	**98M**	70.5%/89.5%	-
TinyNet-C†	2.5M	100M	71.2%/89.7%	$\sim$ 520
Ours	3.1M	**98M**	**71.6%/89.9%**	21

data. Instead, our specialized MAP shows a lower error rate on both training data (0.08%) and evaluation data (0.25%).

Influence of Polynomials' Degree: Figure 4 shows the pruning results when adjusting the MAP's degree $\mathcal{K}$. Especially, the polynomial regression degrades to linear regression when $\mathcal{K} = 1$. It turns out that for polynomials with $\mathcal{K} \leq 2$, the predicted optimal $(d^\star, w^\star, r^\star)$ actually leads to a sub-optimal pruning policy, which indicates that the MAP is too simple to use. For polynomials with $\mathcal{K} \geq 3$, all MAPs generate similar predictions about the optimal (d, w, r), i.e., $(0.78, 0.82, 0.98)$ for ResNet-32 trained on CIFAR-10 and $(0.65, 1.0, 0.63)$ for ResNet-56 trained on TinyImageNet. These results corroborate that our algorithm is relatively robust with respect to different degrees $\mathcal{K}$, so practitioners do not need to choose the polynomial degree carefully.

4.4. Case Study

Visualization of Feature Maps for Different Pruning Policies: In order to further understand why pruning the

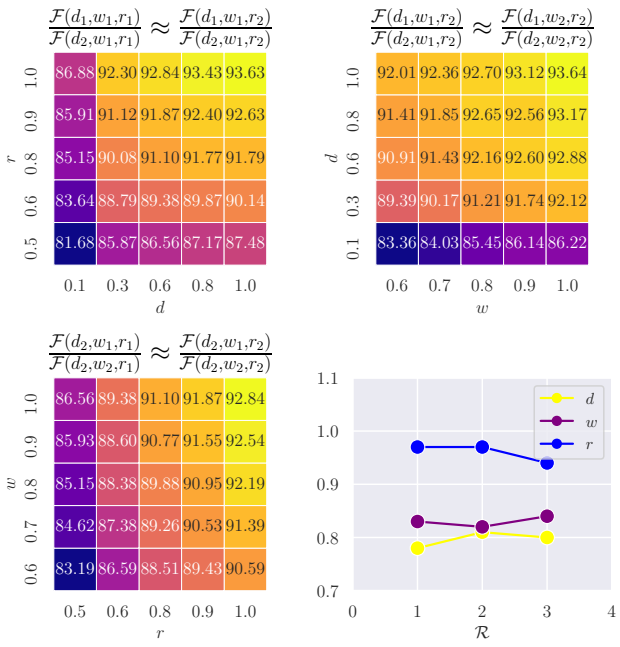

Figure 3. Accuracies of ResNets with different (d, w, r) trained on CIFAR-10 (the first three sub-figures) and the predicted optimal $(d^\star, w^\star, r^\star)$ with different $\mathcal{R}$ in Eq. (10).

Table 4. MAP's regression results with lower degree or regularization. $\mathcal{K}$ means the highest degree for each variable in (d, w, r). ℓ_2 is the coefficient of the regularization term, and 0 indicates no regularization.

Type	$\mathcal{K}$	ℓ_2	Train Err.	Eval Err.
Normal Polynomial	1	0	2.66%	2.97%
	2	0	1.62%	2.25%
	3	0	0.28%	1.28%
	5	0	0.02%	2.31%
	5	10^{-3}	0.02%	2.28%
	10	0	0.01%	2.58%
	10	10^{-3}	0.02%	2.32%
Ours	3	0	0.14%	0.33%
	5	0	0.08%	**0.25%**

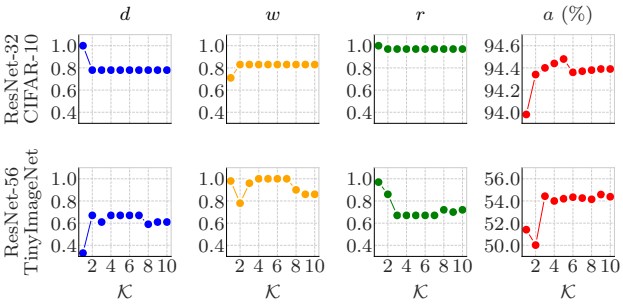

Figure 4. The predicted optimal $(d^\star, w^\star, r^\star)$ and corresponding pruning results with different $\mathcal{K}$. For all $\mathcal{K} \geq 3$, the MAP's predicted optimal $(d^\star, w^\star, r^\star)$ are very similar. Users do not need to bother to choose $\mathcal{K}$ carefully.

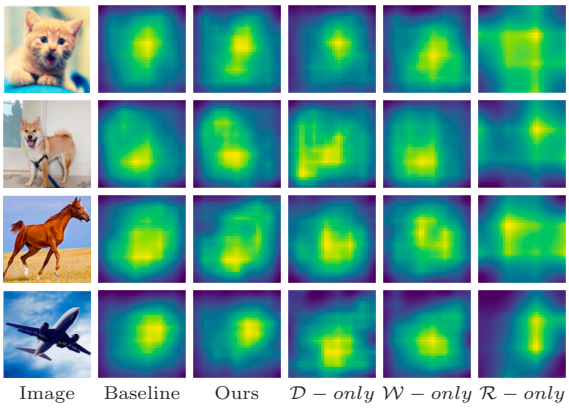

Figure 5. Visualization of last layer's feature maps from different models. The baseline model is a pre-trained ResNet-56. Four different pruning policies are tested, and our pruned model's feature maps look most like those of the baseline model.

three dimensions simultaneously yields better results than pruning only one, Figure 5 compares the feature maps for models with different pruning policies. Specifically, all models are pruned from the same baseline model — ResNet-56 pre-trained on CIFAR-10. Input images are randomly chosen from the Internet. For visualization, we extract the last convolutional layer's feature maps and compute their mean absolute values across channels. Figure 5 shows that the feature maps our pruned model outputs look most similar to those of the original model. This finding reveals that our algorithm preserves most information of the original model by pruning the three dimensions comprehensively.

5. Conclusion

In this paper, we proposed a novel pruning framework which prunes a pre-trained model along three dimensions, i.e., depth, width, and resolution, comprehensively. Remarkably, our framework can determine the optimal values for these three dimensions through modeling the relationships between the model's accuracy and depth/width/resolution into a polynomial regression and subsequently solving an optimization problem. The extensive experimental results demonstrate that the proposed pruning algorithm outperforms state-of-the-art pruning algorithms under a comparable computational budget. In contrast with NAS-based methods, we generated the pruned models that are superior to the NAS-searched models with a much reduced computational cost.

Acknowledgements

This work was supported in part by The National Key Research and Development Program of China (Grant No. 2018AAA0101400), in part by The National Nature Science Foundation of China (Grant Nos: 62036009, U1909203, 61936006), and in part by Innovation Capability Support Program of Shaanxi (Program No. 2021TD-05).

References

Ba, J. and Caruana, R. Do deep nets really need to be deep? In *NeurIPS*, 2014.

Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006.

Bourbaki, N. *Elements of mathematics: Algebra*. Springer, 2003.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019.

Gao, Y., Bai, H., Jie, Z., Ma, J., Jia, K., and Liu, W. MTL-NAS: task-agnostic neural architecture search towards general-purpose multi-task learning. In *CVPR*, pp. 11540–11549. IEEE, 2020.

Han, K., Wang, Y., Zhang, Q., Zhang, W., Xu, C., and Zhang, T. Model rubiks cube: Twisting resolution, depth and width for tinynets. *NeurIPS*, 2020.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

He, C., Ye, H., Shen, L., and Zhang, T. Milenas: Efficient neural architecture search via mixed-level reformulation. In *CVPR*, pp. 11990–11999. IEEE, 2020.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *CVPR*, 2016.

He, Y., Kang, G., Dong, X., Fu, Y., and Yang, Y. Soft filter pruning for accelerating deep convolutional neural networks. In *IJCAI*, 2018.

He, Y., Liu, P., Wang, Z., Hu, Z., and Yang, Y. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *CVPR*, 2019.

Hinton, G. E., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

Howard, A., Pang, R., Adam, H., Le, Q. V., Sandler, M., Chen, B., Wang, W., Chen, L., Tan, M., Chu, G., Vasudevan, V., and Zhu, Y. Searching for mobilenetv3. In *ICCV*, pp. 1314–1324. IEEE, 2019.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *CVPR*, 2017.

Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M. X., Lee, H., Ngiam, J., Le, Q. V., Wu, Y., and Chen, Z. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *NeurIPS*, 2019.

Kang, M. and Han, B. Operation-aware soft channel pruning using differentiable masks. In *ICML*, 2020.

Kolda, T. G. and Bader, B. W. Tensor decompositions and applications. *Society for Industrial and Applied Mathematics Review*, 2009.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Lee, N., Ajanthan, T., and Torr, P. H. S. Snip: single-shot network pruning based on connection sensitivity. In *ICLR*, 2019.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. In *ICLR*, 2017.

Li, Y., Gu, S., Zhang, K., Gool, L. V., and Timofte, R. DHP: differentiable meta pruning via hypernetworks. In *ECCV*, 2020.

Lin, M., Ji, R., Wang, Y., Zhang, Y., Zhang, B., Tian, Y., and Shao, L. Hrank: Filter pruning using high-rank feature map. In *CVPR*, 2020.

Lin, S., Ji, R., Yan, C., Zhang, B., Cao, L., Ye, Q., Huang, F., and Doermann, D. S. Towards optimal structured CNN pruning via generative adversarial learning. In *CVPR*, 2019.

Liu, H., Simonyan, K., and Yang, Y. DARTS: differentiable architecture search. In *ICLR*. OpenReview.net, 2019.

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017.

Luo, J., Zhang, H., Zhou, H., Xie, C., Wu, J., and Lin, W. Thinet: Pruning CNN filters for a thinner net. *TPAMI*, 2019.

Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. Pruning convolutional neural networks for resource efficient inference. In *ICLR*, 2017.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. 2017.

Pham, H., Guan, M. Y., Zoph, B., Le, Q. V., and Dean, J. Efficient neural architecture search via parameter sharing. In Dy, J. G. and Krause, A. (eds.), *ICML*, 2018.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. S., Berg, A. C., and Li, F. Imagenet large scale visual recognition challenge. *IJCV*, 2015.

Sehwag, V., Wang, S., Mittal, P., and Jana, S. HYDRA: pruning adversarially robust neural networks. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *NeurIPS*, 2020.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In Bengio, Y. and LeCun, Y. (eds.), *ICLR*, 2015.

Tan, M. and Le, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019.

Tian, Y., Shen, L., Shen, L., Su, G., Li, Z., and Liu, W. Alphagan: Fully differentiable architecture search for generative adversarial networks. *CoRR*, abs/2006.09134, 2020.

Wang, W., Fu, C., Guo, J., Cai, D., and He, X. COP: customized deep model compression via regularized correlation-based filter-level pruning. In *IJCAI*, 2019a.

Wang, W., Zhao, S., Chen, M., Hu, J., Cai, D., and Liu, H. DBP: discrimination based block-level pruning for deep model acceleration. *CoRR*, abs/1912.10178, 2019b.

Wang, Y., Zhang, X., Xie, L., Zhou, J., Su, H., Zhang, B., and Hu, X. Pruning from scratch. In *AAAI*, 2020.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. Learning structured sparsity in deep neural networks. In *NIPS*, 2016.

Wu, J., Zhang, Q., and Xu, G. Tiny imagenet visual recognition challenge. Technical report, 2017.

Ye, M., Gong, C., Nie, L., Zhou, D., Klivans, A., and Liu, Q. Good subnetworks provably exist: Pruning via greedy forward selection. In *ICML*, 2020a.

Ye, X., Dai, P., Luo, J., Guo, X., Qi, Y., Yang, J., and Chen, Y. Accelerating CNN training by pruning activation gradients. In Vedaldi, A., Bischof, H., Brox, T., and Frahm, J. (eds.), *ECCV*, 2020b.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *ICLR*, 2017.