Joint Multi-Dimension Pruning

Zechun Liu[*]
HKUST
zliubq@connect.ust.hk

Xiangyu Zhang
MEGVII Technology
zhangxiangyu@megvii.com

Zhiqiang Shen[†]
CMU
zhiqians@andrew.cmu.edu

Zhe Li **Yichen Wei**
MEGVII Technology
{lizhe,weiyichen}@megvii.com

Kwang-Ting Cheng
HKUST
timcheng@ust.hk

Jian Sun
MEGVII Technology
sunjian@megvii.com

Abstract

We present joint multi-dimension pruning (named as JointPruning), a new perspective of pruning a network on three crucial aspects: spatial, depth and channel simultaneously. The joint strategy enables to search a better status than previous studies that focused on individual dimension solely, as our method is optimized collaboratively across the three dimensions in a single end-to-end training. Moreover, each dimension that we consider can promote to get better performance through colluding with the other two. Our method is realized by the adapted stochastic gradient estimation. Extensive experiments on large-scale ImageNet dataset across a variety of network architectures MobileNet V1&V2 and ResNet demonstrate the effectiveness of our proposed method. For instance, we achieve significant margins of 2.5% and 2.6% improvement over the state-of-the-art approach on the already compact MobileNet V1&V2 under an extremely large compression ratio.

1 Introduction

Network pruning has been acknowledged as one of the most effective model compression methods for adapting heavy models to the resource-limited devices [21, 56, 30]. The pruning methods evolved from unstructured weight pruning [12, 15] to structured channel pruning [10, 62] with the purpose of more hardware-friendly implementation.

With the increasing demand of highly compressed models, merely pruning the channel dimension becomes insufficient to strike a good computation-accuracy trade-off. Some previous studies begin to seek compression schemes besides the channel dimension. For example, OctaveConv [6] proposed to manually reduce the spatial redundancy in feature maps. MobileNet [22] scales down the input image's spatial size for reducing the computational overhead. Besides the spatial dimension, depth is also a dimension worth exploring [8]. These three dimensions (i.e., channel, spatial, depth) are inter-related: A reduction in the network depth can enable using larger feature maps or more channels under the same computational constraint; An alteration in the spatial size will affect the optimal channel pruning scheme. To this end, manually set the spatial size and network depth while only adjust the layer-wise channel pruning strategy leads to sub-optimal solutions. It is of high significance to automatically allocate computational resource among three dimensions with a unified consideration.

However, jointly pruning channels together with the spatial and depth dimension is challenging and some difficulties are always underestimated. The first challenge is that most of the pruning algorithms are developed based on the unique property of the channel dimension, for instance, removing channels with small weight L1 norms [30, 54]. Such approaches are inflexible and can hardly generalize to other dimensions, especially the spatial dimension. The second challenge stems from the overladen

[*]This work is done when Zechun Liu interns at MEGVII Technology. [†]Corresponding author.

search space within the three integrated dimensions, since the potential choices of these dimensions are numerically consecutive integers. Therefore, the previous enumerate-based methods are always computationally prohibitive or under-optimized.

To resolve the aforementioned limitations and derive a decent solution, we propose the JointPruning that considers three dimensions (channel, spatial, depth) simultaneously. In contrast to previous pruning methods that mainly focus on exploring the correlation between channels (e.g., the redundancy), we formulate this problem by defining channel numbers associating with the spatial and depth into an integrated space through a configuration vector. Within this joint space, we can not only mine the relationships inside a single dimension, but also find relatively optimal options across three dimensions by optimizing this configuration vector. That is to say, we consider the global reciprocity between the three aspects and model joint pruning with the defined pruning configuration vector specifying layer-wise channel numbers, spatial size, and the network depth. This circumvents the algorithmic dependency on the exclusive properties in the channel dimension.

As there is no explicit function between the loss function of the pruned network and its configuration vector (*i.e.*, it is non-differentiable to the pruning configuration), also considering the efficiency of optimization, we propose to use stochastic gradient estimation to approximate the gradient *w.r.t.* configuration vectors. We first convert the loss function to an expectation of the loss of configuration vectors sampled from a Gaussian distribution around the original vector. Then we utilize the *log likelihood trick* [46] to obtain the gradient of expectation as an approximation to the underlying gradient. We further design weight sharing mechanism for evaluating the loss of configuration vectors without training each corresponding pruned network from scratch. With alternatively updating weights and configuration vector, the proposed JointPruning can automatically learn the global pruning configuration across three dimensions.

We show the effectiveness of JointPruning on MobileNet V1/V2 and ResNet structures. It achieves up to $9.1\%/6.3\%$ higher accuracy than MobileNet V1/V2 baselines under FLOPs constraint. Moreover, JointPruning is friendly to tackle channels in shortcuts and also supports multiple resource constraints. It can automatically adapt the pruning scheme according to the underlying hardware specialty and different circumstances due to its larger potential options. Under CPU latency constraint, JointPruning achieves comparable or higher results with less searching time than advanced adaptation based ChamNet [9]. When targeting at GPU latency, it surpasses MobileNet V1/V2 baselines with more than 3.9% and 2.4% improvement under an extremely large compression ratio.

Contributions:

• We reveal the perspective that joint pruning neural network in three dimensions (*i.e.*, channel, spatial and depth) is crucial for striking a good accuracy-computation trade-off. We define pruning configuration as a vector specifying the spatial size, network depth and layer-wise channel numbers, and formulate joint pruning as optimizing the configuration vector.

• We propose to use stochastic gradient estimation *w.r.t.* the configuration vector for efficient optimization. This strategy inherits the high efficiency advantage of the gradient descent algorithm meanwhile overcomes the non-differentiability in the configuration vectors.

• We design weight sharing strategy to avoid training individual pruned network from scratch, and an alternative updating strategy to jointly optimize weights and configuration vectors, which further facilitates optimization.

• The proposed JointPruning can flexibly incorporate different resource constraints including FLOPs and hardware latency. It outperforms traditional pruning algorithms with less human efforts and achieved higher results than state-of-the-art model adaptation methods with less optimization time.

2 Related Work

Model compression is recognized as an effective approach for efficient deep learning [14, 50]. The categories of model compression expands from pruning [21, 7, 23], quantization [60, 61], to compact network architecture design [34, 57, 22, 42]. Our approach is most related to pruning.

Pruning and Model Adaptation Early works [24, 16, 12] prune individual redundant weights and recent studies focus on removing the entire kernel [30, 33] to produce structured pruned networks. While traditional pruning involves massive human participation in determining the pruning ratio, AutoML-based methods with reinforcement learning [19], a feedback loop [54] or a meta network [31] can automatically decide the best pruning ratio. In those pruning methods, the optimization space is confined to channels only. Intuitively, a higher compression ratio and better trade-off can be obtained through jointly reducing channel, spatial and depth dimensions. So far, only a few works address the joint adaptation across three dimensions. EfficientNet [48] focuses on scales all dimensions of depth/width/resolution with a single compound coefficient. The coefficient is obtained by grid search. ChamNet [9] uses the Gaussian process to predict the accuracy of compression configurations in three dimensions. Despite their high accuracy, grid search or Gaussian process requires training hundreds of networks from scratch, which are with high computational cost. Instead, we propose to model the joint pruning as optimizing the numerical values of architecture configurations (i.e., channel, depth and spatial), which enables to use a *stochastic gradient estimation w.r.t.* the configuration vector in optimization, and further improves the efficiency greatly.

Gradient Estimation Gradient estimation is initially proposed for optimizing non-differentiable objectives [36]. Reinforcement learning (RL) algorithms often utilize gradient estimation to update the policy [49, 37, 11, 26, 3, 41, 1]. We customize a relaxation of the objective function and use the *log likelihood trick* to calculate the estimated gradient [44]. Different from these algorithms designed for the typical RL tasks, such as Atari and MuJoCo, our approach focuses on optimizing the network configuration and assumes the policy function to be a Gaussian distribution, which can better approximate the actual gradient and is free of hyper-parameters. Furthermore, we utilize the Lipchitz continuous property of the neural networks *w.r.t.* the configuration vectors. Such that we adopt a progressively shrinking Gaussian window for obtaining more accurate gradient estimations, and use an alternative paradigm for updating configuration vectors and the weight parameters.

Neural Architecture Search and Parameter Sharing Different from pruning tasks, which aims to find a compressed model through adjusting the numerical values in architecture configurations, NAS are targeting at choosing the best options [63, 38, 47, 53] and/or connections [52]. Although a few NAS studies include two or three channel number choices in their search space [51, 5], they enumerate these choice as independent nodes. The gradient-based neural architecture search (NAS) adopts Straight Through Estimation [5] or probabilistic relaxation [29], which models choices as independent and is essentially unsuitable for continuous numerical choices in the pruning configuration. While proposed JointPruning tailored for this task utilizes the consecutiveness in the numerical values, and is thus more effective. Parameter sharing is studied in NAS. Our parameter-sharing scheme is inspired by the practice in one-shot architecture search [13, 45, 2, 4]. Different from the operation-level sharing mechanism designed for independent operations, proposed weight sharing adopts a matrix-level sharing, which is specialized for a more efficient sharing among the consecutive channel/spatial/depth choices. The details of our algorithm are described in Sec. 3.

3 Methodology

To solve the pruning problem in three dimensions, we define the pruned networks with the corresponding unique architecture configuration vectors:

$$v_{conf} = \{c, s, d\}, \tag{1}$$

where c denotes the number of channels in each layer, c = $\{c_1, c_2, ..., c_n\}$, n is the total number of layers, s denotes the spatial size of the input image to the network and d denotes the network depth.

We formulate the JointPruning as a constrained optimization problem over the configuration vector. The objective is to minimize the loss under a given resource constraint:

$$\text{minimize } \mathcal{L}(v, w) \tag{2}$$

$$\text{subject to } \ \mathcal{C}(v) < constraint \tag{3}$$

where $\mathcal{L}$ refers to the loss function of the neural network, w is the weight parameters, $\mathcal{C}$ is the computational cost given the network configuration v and $constraint$ denotes the target resource constraint (i.e., FLOPs or latency). We merge the resource constraint as a regularization term in the error function $\mathcal{E}$ as follows:

$$\mathcal{E}(v, w) = \mathcal{L}(v, w) + \rho||\mathcal{C}(v) - constraint||^2, \tag{4}$$

where ρ is the positive regularization coefficient. Consequently, the goal is to find the architecture configuration vector v that minimizes the error function $\mathcal{E}$:

$$v = \underset{v}{argmin}(\mathcal{E}) \tag{5}$$

However, it is computationally prohibitive to obtain the error $\mathcal{E}$ through enumerating the configuration vectors and training each corresponding pruned network from scratch. To tackle this, we propose the gradient estimation method to approximate the steepest gradient descent direction of the configuration vector that minimizes the error, and also customize a weight sharing mechanism to further enhance the optimization efficiency. We detail our gradient estimation method and the weight sharing mechanism in Section 3.1 and Section 3.2, respectively.

3.1 Stochastic Gradient Estimation for Configuration Vectors

Since the error function $\mathcal{E}$ is naturally non-differentiable with respect to the configuration vectors, we propose gradient estimation to approximate the underlying gradients. Inspired by [39, 55, 43], we conduct the gradient estimation by relaxing the objective error function $\mathcal{E}$ to the expectation of $\mathcal{E}$ with configuration vector v obeying a distribution $p_\theta(v)$.

$$\mathcal{E}(v) \approx \mathbb{E}_{v \sim p_\theta(v)} \mathcal{E}(v), \tag{6}$$

where the distribution $p_\theta(v)$ is parameterized by θ. To simplify the expression, we omit the w in the term $\mathcal{E}(v, w)$. In previous works, $p_\theta(v)$ is chosen to be a function of v. Instead, and considering the relaxation purpose is for obtaining precise gradient estimation *w.r.t.* the configuration vector v, we simplify the probability distribution $p_\theta(v)$ to be an isotropic multivariate gaussian distribution:

$$p_\mu(v) \sim \mathcal{N}(\mu, \sigma), \tag{7}$$

with the mean μ being the current configuration vector and the deviation set to σ. Consequently, the approximation to the objective function can be written as:

$$\mathcal{E}(\mu) \approx \mathbb{E}_{v \sim \mathcal{N}(\mu,\sigma)} \mathcal{E}(v) = \mathbb{E}_{n \sim \mathcal{N}(0,\sigma)} \mathcal{E}(\mu + n), \tag{8}$$

where n denotes the random multidimensional Gaussian noise added to the current configuration vector. This approximation holds because the neural network is *Lipschitz* continuous with respect to v, i.e., there exists a positive real constant K such that:

$$||\mathcal{E}(\mu + n) - \mathcal{E}(\mu)|| < K||n||. \tag{9}$$

In configuration vector optimization, n is confined to small variation values, i.e., $||n|| < \epsilon$. Therefore, the alteration in the error function is bounded within $K\epsilon$. In experiments, we observe the constant K to be diminutive within a local region around the current configuration vector and thus the variation is tolerable.

With this approximation to the error function, we can derive the estimated gradient following the commonly adopted *log likelihood trick* in reinforcement learning:

$$\begin{aligned}
\nabla_\mu \mathbb{E}_{v \sim \mathcal{N}(\mu,\sigma)} \mathcal{E}(v) &= \mathbb{E}_{v \sim \mathcal{N}(\mu,\sigma)} [\mathcal{E}(v) \nabla_\mu \log(p(v; \mu))] & (10)\\
&= \mathbb{E}_{v \sim \mathcal{N}(\mu,\sigma)} [\mathcal{E}(v) \nabla_\mu (\log(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{v-\mu}{\sigma})^2}))] & (11)\\
&= \mathbb{E}_{v \sim \mathcal{N}(\mu,\sigma)} [\mathcal{E}(v) \frac{v-\mu}{\sigma^2}] & (12)\\
&= \frac{1}{\sigma^2} \mathbb{E}_{n \sim \mathcal{N}(0,\sigma)} [\mathcal{E}(\mu + n) n]. & (13)
\end{aligned}$$

Thus, the gradient of $\mathcal{E}(v)$ can be estimated by the configuration vector v stochastically sampled around the current vector μ under a gaussian distribution $n \sim \mathcal{N}(0, \sigma)$:

$$\nabla_\mu \mathbb{E}_{v \sim \mathcal{N}(\mu,\sigma)} \mathcal{E}(v) \approx \frac{1}{\sigma^2} \sum_{i=1}^{M} \mathcal{E}(\mu + n_i) n_i, \tag{14}$$

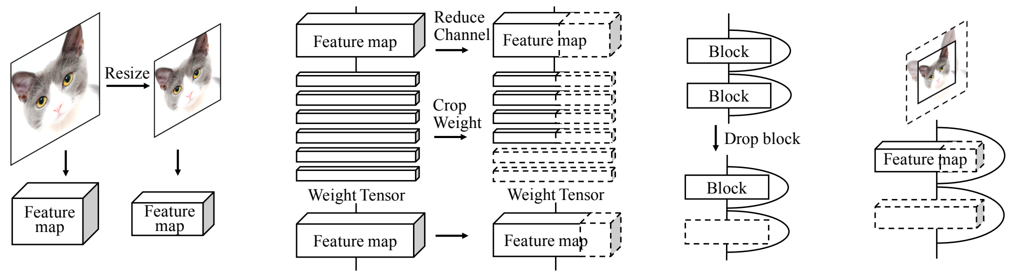
where M is the total number of samples. An intuitive interpretation of this gradient estimation approach is: by weighting the variation directions with the corresponding error, the gradient direction can be approximated with the variation direction that has the lower expected error. Then the configuration vector is updated with the estimated gradient:

$$\mu' = \mu - \alpha \nabla_\mu \mathcal{E}(v), \tag{15}$$

where α denotes the update rate, and the geometric meaning of μ is the center configuration vector for adding Gaussian variations. The resulting algorithm iteratively executes the following two steps: (1) stochastic sampling of the architecture configuration vectors and evaluate the corresponding errors; (2) integrating the evaluations to estimate the gradient and update the configuration vector.

3.2 Weight Sharing

In the aforementioned stochastic gradient estimation paradigm, one crucial issue that remains unsolved is how to obtain the error evaluation of the configuration vector. Recall that the error $\mathcal{E}$ is a function of v and w: $\mathcal{E} = \mathcal{E}(v, w)$. To evaluate the configuration vector(v), the weights(w) in the corresponding pruned network need to be trained. However, it would be too computationally prohibitive to train each pruned network from scratch. To address this, we customize a parameter-sharing mechanism to share the weights for different architectures varying in the channel, spatial and depth configurations.



(a) Spatial (b) Channel (c) Depth (d) Joint weight sharing

Figure 1: The weight-sharing mechanism.

The parameter-sharing mechanism is illustrated in Figure 1. For spatial dimension, decreasing the input image resolution will reduce the feature map size in the network, as Figure 1(a). This does not require any modification in the weight kernels and thus all the network weights are shared when adjusting the spatial resolution. To prune the channels, we follow [31] to keep the first c channels of feature maps in the original network and crop the weight tensors correspondingly, as shown in Figure 1(b). To reduce depth, we adopt block dropping. That is keeping the first d blocks and skipping the rest of the blocks, as Figure 1(c). Combining these three dimensions, we come up with the joint parameter-sharing mechanism as illustrated in Figure 1(d). With this mechanism, we only need to train one set of weights in the weight-shared network and evaluate the error of different pruning configuration vectors with corresponding weights cropped from this network.

3.3 Alternatively Update Weights and Configuration Vectors

For training the weights in the shared network, the optimization goal is set to minimize the error concerning the expectation of a bunch of configuration vectors. The objective is defined as:

$$\min_{w} \mathbb{E}_{v \sim \mathcal{N}(\mu,\sigma)} \mathcal{E}(v, w) \tag{16}$$

To optimize this objective function, we inject noise to the configuration vectors during weight training. That is, in each weight training iteration, the configuration vectors v is randomly sampled from $\mathcal{N}(\mu, \sigma)$.

After training the weights for one epoch, we use the stochastic gradient estimation to estimate the gradient *w.r.t.* the μ and take a step in the μ variable space to decline the error:

$$\min_{\mu} \mathbb{E}_{v \sim \mathcal{N}(\mu,\sigma)} \mathcal{E}(v, w) \tag{17}$$

We alternatively optimize the weights and the architecture configuration vectors and meanwhile decrease the deviation σ till convergence. With the error function $\mathcal{E}$ well approximated by its smoothed version $\mathbb{E}_{v \sim \mathcal{N}(\mu,\sigma)} \mathcal{E}(v, w)$, we can obtain the optimized configuration vector to be the mean of the configuration vector distributions (i.e., μ) of the final model. The optimization pipeline is detailed in Algorithm 1.

Algorithm 1: Joint Multi-Dimension Pruning

Input: Learning rate α, standard deviation of gaussian distribution σ, initial mean μ_0, configuration vector v, initial weight: w, error function: $\mathcal{E}$, number of total iterations: K, number of weight update iterations: N, number of configuration samples: M.

Output: Optimized configuration μ^*

```
for t = 0 : K do
    for i = 0 : N do
        min_w E_{v~N(μ,σ)} E(v, w)
    end for
    for j = 0 : M do
        E_j = E(μ_t + n_j)
    end for
    μ_{t+1} = μ_t − α Σ_{i=1}^{M} E_j n_j
end for
return μ*
```

4 Experiments

In this section, we begin by introducing our experiment settings in Sec. 4.1. Then, we explain the configuration vector settings in Sec. 4.2. Thirdly, we show the comparison between our method and the state-of-the-arts on various architectures and under different resource constraint metrics in Sec. 4.3. Lastly, we visualize two of the pruned networks obtained by our algorithm in Sec. 4.4.

4.1 Experiment Settings

Dataset All experiments are conducted on the ImageNet 2012 classification dataset [40]. For optimizing the configuration vector, we randomly split the original training set into two subsets: 50,000 images for validation (50 images for each class) and use the rest as the training set. After the architecture hyperparameters are optimized, we train the corresponding architecture from scratch using the original training/validation dataset split, following the practice in [5, 51].

Training We alternatively optimize the configuration vector and weights with an outer loop of 75 iterations. In each inner loop, we train the weights for 2000 iterations for MobileNet V1/V2 and 1000 iterations for ResNet with a batch size of 256. We use the SGD optimizer with momentum for weight training. Then we update the configuration vector for 20 iterations with the estimated gradients. In gradient estimation, the number of samples M is set to 100 for each iteration, the deviation σ and the update rate α are initialized with 1.25 and 5 respectively, and linearly decay to 0.25 and 0, respectively, with the outer loop.

Constraints The experiments are carried out under the FLOPs, GPU latency as well as the CPU latency constraints. For the latency constraints, we follow the practice in FBNet [51] to build a latency look-up-table for a layer with different configurations and obtain the total latency of the network by summing up the latency of all layers. In our experiments, we estimate GPU latency on the GTX 1080Ti with a batch size of 256. For CPU latency, we use the look-up-table released by ChamNet [9] for a fair comparison. As ChamNet has a very sparse look-up-table, we use the Gaussian process to predict the missing latency values as [9] did for energy estimation.

4.2 Details in configuration vector settings

For MobileNet V1, we set the optimization vector(v) as $v = \{c_1, c_2, \dots c_n, s\}$, where c_i are the numbers of output channels in layer i, and n denotes total number of layers. s is the spatial size of the input image. As MobileNet V1 is a network without shortcuts, reducing the depth will result in channel mismatch. With this structure restriction, we only optimize the channel and spatial dimensions. We initiate the configuration vector as the configuration of the original MobileNet V1. After obtaining the optimal configuration, the pruned network is constructed through pruning each layer's input channel equaling to previous layer's output channel, specified by c_i, and then crop the weight kernels correspondingly as shown in Figure 1(b).

Similar to MobileNet V1, the optimization vector(v) for MobileNet V2 is defined as $v = \{c_1, c_2, \dots c_n, s, d\}$. Since MobileNet V2 contains the shortcut structure, we confine the layers connected with identity shortcuts to have the same number of output channels. For pruning depth dimension, we drop the last block in the stage that contains the most number of blocks.

For ResNet, we adopt similar settings of MobileNet V2, and confine the configuration vectors to layer-wise channel numbers only, i.e. $v = \{c_1, c_2, \dots c_n\}$ for a direct comparison to the traditional channel pruning methods.

4.3 Comparisons with state-of-the-arts

To verify the effectiveness of proposed JointPruning, we compare our method with the uniform baselines, traditional pruning methods as well as the AutoML-based model adaptation methods. Our experiments are conducted on MobileNet V1, MobileNet V2 [22, 42] and ResNet [17] backbones.

MobileNet V1&V2. MobileNets [22, 42] are already compact networks, many recent pruning algorithms focus on these structures for the practical significance in pruned models.

The baseline group in Table 1 shows that rescaling channel and spatial dimension (C+S) by the same ratio can achieve higher accuracy than merely rescale channel (C) or spatial (S), suggesting the

Table 1: Comparison between the proposed method with the uniform baselines and the state-of-the-art AutoML-based pruning methods on MobileNet V1&V2 [22, 42], under the FLOPs constraint. The baseline is obtained by uniformly rescale channel (C) / spatial size of the input image (S) / channel + spatial (C+S) by a fixed ratio. C, S, D denotes channel, spatial and depth, respectively.

		MobileNet V1						**MobileNet V2**					
		45M		150M		330M		45M		150M		200M	
		Acc (%)	FLOPs (M)	Acc (%)	FLOPs (M)	Acc (%)	FLOPs (M)	Acc (%)	FLOPs (M)	Acc (%)	FLOPs (M)	Acc (%)	FLOPs (M)
Uniform Baseline	C	50.6	41	63.7	149	68.4	325	54.6	43	67.2	140	70.0	203
	S	54.0	43	65.7	149	70.6	343	59.1	42	68.3	138	70.8	220
	C + S	58.8	46	67.1	143	70.8	325	59.3	47	68.5	142	70.6	212
State-of -the-arts	NetAd [54]	–	–	–	–	69.1	284	–	–	–	–	–	–
	AMC [19]	–	–	–	–	70.5	281	–	–	–	–	70.8	220
	MetaP [31]	57.2	41	66.4	149	70.9	324	58.3	43	68.2	140	71.2	217
Ours	C	58.6	48	66.5	145	71.0	310	58.1	46	68.4	141	70.7	205
	C + S	**59.7**	43	**67.6**	139	**71.2**	309	60.1	42	68.8	139	**71.4**	206
	C + S + D	–	–	–	–	–	–	**60.9**	51	**69.1**	145	70.7	206

Table 2: Comparison between the proposed method and uniform channel rescaling baseline on MobileNet V1&V2 [22, 42] under GPU latency constraints. The latency is measured on the GTX 1080Ti with a batch-size of 256. Architecture obtained by jointly pruning layer-wise channel numbers and spatial dimension (C+S) outperforms the baseline of uniform rescaling channel numbers (C). Further including Depth (D) enables a more flexible pruning space and generates even higher accuracy.

		MobileNet V1						**MobileNet V2**					
		0.25×		0.5×		0.75×		0.35×		0.65×		0.8×	
		Acc (%)	Latency (ms)	Acc (%)	Latency (ms)	Acc (%)	Latency (ms)	Acc (%)	Latency (ms)	Acc (%)	Latency (ms)	Acc (%)	Latency (ms)
Baseline	C	50.6	2.266	63.7	3.998	68.4	5.620	54.6	4.22	67.2	5.97	70.0	7.36
Ours	C + S	**66.8**	2.231	**70.6**	3.972	**72.3**	5.617	67.7	4.10	71.2	5.93	72.3	7.34
	C + S + D	–	–	–	–	–	–	**68.7**	3.98	**71.7**	5.90	**72.4**	7.17

importance of jointly dealing with multi-dimensions. Then, in this joint pruning space, the proposed JointPruning can automatically capture the delicate linkage between spatial pruning and channel-wise pruning ratio adjustments, which outputs optimized pruned networks with higher accuracy than state-of-the-art pruning methods focusing on channels only. When further considering the depth dimension, JointPruning obtains even better accuracy, especially under the extreme small FLOPs constraint (45M), which demonstrates the superiority to others.

Besides the FLOPs constraint, we extensively study the behavior of JointPruning under direct metrics like GPU latency and CPU latency. Given various latency constraints, JointPruning consistently improves the accuracy by a remarkable margin especially for highly paralleled GPU devices. As shown in Table 2, on MobileNet V1 JointPruning with the channel and spatial dimension (C+S) achieves more than 3.9% accuracy enhancements. On MobileNet V2, JointPruning comprehensively considers channel, spatial and depth (C+S+D) dimensions, and automatically capture and utilize the underlying hardware specialty, which further boosts the accuracy. Compared to the state-of-the-art model adaptation method ChamNet building atop the Gaussian process, JointPurning with stochastic gradient estimation enjoys high optimization efficiency. As shown in Table 3, JointPruning achieves comparable or higher accuracy compared to ChamNet, but with much lower optimization time cost.

ResNet. Despite that JointPruning is designed for multi-dimensional pruning, we show that it is also effective when targeting at channel dimension only. In comparison to the traditional channel pruning methods, JointPruning achieves better performance with less human participation. The accuracy enhancement mainly comes from JointPruning's ability in pruning shortcuts. For traditional pruning with a layer-by-layer scheme, shortcuts will affect more than one layer which is tough to deal with. While JointPruning can effortlessly prune the shortcut by modeling the shortcut pruning as updating the numerical channel numbers.

Random search Compared to the random search which determines the configuration in three dimensions randomly, Table 4 shows that JointPurning is able to end up at an optimized minimum with high accuracy, and significantly outperforms random search scheme.

Table 3: Accuracy and optimization time comparison between proposed method and state-of-the-art model adaptation method ChamNet [9] on MobileNet V2 [42].

		ChamNet	Ours
0.25×	Acc(%)	64.1	**66.4**
	Latency	6.1ms	**6.0ms**
0.5×	Acc(%)	69.0	**69.0**
	Latency	10.0ms	**9.9ms**
0.75×	Acc(%)	**71.9**	71.8
	Latency	15.0ms	**14.8ms**
Optimization Time (GPU days)		~ 5760	$\mathbf{3 \times 40}$

Table 4: JointPruning vs. Random search on MobileNet V2 structure.

	Random search	Ours
Acc(%)	65.2 ± 2.5	**69.1**
FLOPs	$146.5\text{M} \pm 1.5\text{M}$	**145M**
Acc(%)	68.2 ± 2.9	**71.7**
Latency	$5.985\text{ms} \pm 0.015\text{ms}$	**5.90ms**

Table 5: Comparison between proposed method on channel dimension and state-of-the-art traditional channel pruning methods on ResNet50 [17] backbone.

	1G		2G		3G	
	Acc (%)	FLOPs (G)	Acc (%)	FLOPs (G)	Acc (%)	FLOPs (G)
Uniform	72.0	1.1	74.8	2.3	76.0	3.2
AP [32]	72.0	1.1	74.8	2.3	-	-
ThiNet [33]	72.1	1.2	74.7	2.1	75.8	2.9
CP [21]	-	-	73.3	2.0	-	-
SFP [18]	-	-	-	-	75.1	2.9
PFEC [25]	-	-	-	-	72.9	3.1
C-SGD [10]	-	-	74.5	2.1	75.3	2.9
GDP [27]	70.9	1.6	72.6	2.2	-	-
IENNP [35]	71.7	1.3	74.5	2.3	75.5	2.7
FPGM [20]	-	-	75.6	2.4	-	-
VCNNP [58]	-	-	75.2	2.4	-	-
GAL-0.5 [28]	-	-	72.0	2.3	-	-
RRBP [59]	73.0	1.9	-	-	-	-
Ours	**73.4**	1.0	**75.6**	2.0	**76.2**	3.0

4.4 Optimization results visualization

In this section, we visualize two sets of pruning configurations for MobileNetV2 focusing on optimizing CPU and GPU latency, respectively. By incorporating the latency into the error function $\mathcal{E}$, JointPruning is able to take advantage of hardware characteristics without knowing the implementation details. The final network optimized under the CPU latency constraint is deep with smaller spatial resolution. While for the GPU, the corresponding architecture discovered by JointPruning adopts large spatial resolution, more channels with fewer layers to fully utilize the parallel computing capability of GPUs.

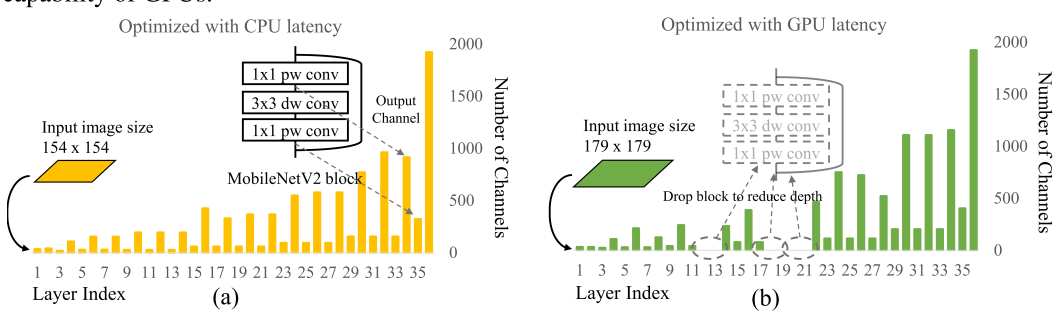


Figure 2: This figure shows the architecture hyper-parameters (input image size, layer-wise channel numbers and depth) of MobileNet V2 network structure optimized under (a) CPU latency constraint with latency = 10ms and (b) GPU latency constraint with latency = 5.62ms.

5 Conclusion

In this work, we focused on the joint multi-dimension pruning which naturally has broader pruning space on spatial, depth and channel for digging out better configurations than the isolated single dimension solution. We have proposed to use stochastic gradient estimation to optimize this problem and further introduced a weight sharing strategy to avoid repeatedly training multiple models from scratch. Our results on large-scale ImageNet dataset with MobileNet V1&V2 and ResNet outperformed previous state-of-the-art pruning methods with significant margins.

In the future, we will conduct more deep analyses about how multi-dimension pruning helps to find better optimum. It will also be interesting to apply our method to other tasks like unsupervised pruning to explore the upper limit of the proposed method.

6 Acknowledgement

The authors would like to acknowledge National Key R&D Program of China (No. 2017YFA0700800), Beijing Academy of Artificial Intelligence (BAAI) and HKSAR RGC's funding support under grant GRF-16203918.

References

[1] Thomas Back, Frank Hoffmeister, and Hans-Paul Schwefel. A survey of evolution strategies. In *Proceedings of the fourth international conference on genetic algorithms*, volume 2. Morgan Kaufmann Publishers San Mateo, CA, 1991. 3

[2] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, pages 549–558, 2018. 3

[3] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies–a comprehensive introduction. *Natural computing*, 1(1):3–52, 2002. 3

[4] Han Cai, Chuang Gan, and Song Han. Once for all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019. 3

[5] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018. 3, 6

[6] Yunpeng Chen, Haoqi Fan, Bing Xu, Zhicheng Yan, Yannis Kalantidis, Marcus Rohrbach, Shuicheng Yan, and Jiashi Feng. Drop an octave: Reducing spatial redundancy in convolutional neural networks with octave convolution. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3435–3444, 2019. 1

[7] Ting-Wu Chin, Ruizhou Ding, Cha Zhang, and Diana Marculescu. Legr: Filter pruning via learned global ranking. *arXiv preprint arXiv:1904.12368*, 2019. 2

[8] Elliot J Crowley, Jack Turner, Amos Storkey, and Michael O'Boyle. A closer look at structured pruning for neural network compression. *arXiv preprint arXiv:1810.04622*, 2018. 1

[9] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, et al. Chamnet: Towards efficient network design through platform-aware model adaptation. *arXiv preprint arXiv:1812.08934*, 2018. 2, 3, 6, 8

[10] Xiaohan Ding, Guiguang Ding, Yuchen Guo, and Jungong Han. Centripetal sgd for pruning very deep convolutional networks with complicated structure. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4943–4953, 2019. 1, 8

[11] Shixiang Shane Gu, Timothy Lillicrap, Richard E Turner, Zoubin Ghahramani, Bernhard Schölkopf, and Sergey Levine. Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. In *Advances in neural information processing systems*, pages 3846–3855, 2017. 3

[12] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances in neural information processing systems*, pages 1379–1387, 2016. 1, 2

[13] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. *arXiv preprint arXiv:1904.00420*, 2019. 3

[14] 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. 2

[15] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015. 1

[16] Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pages 293–299. IEEE, 1993. 2

[17] 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*, pages 770–778, 2016. 6, 8

[18] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*, 2018. 8

[19] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018. 3, 7

[20] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2019. 8

[21] 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*, pages 1389–1397, 2017. 1, 2, 8

[22] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 1, 2, 6, 7

[23] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 304–320, 2018. 2

[24] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990. 2

[25] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016. 8

[26] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. 3

[27] Shaohui Lin, Rongrong Ji, Yuchao Li, Yongjian Wu, Feiyue Huang, and Baochang Zhang. Accelerating convolutional networks via global & dynamic filter pruning. In *IJCAI*, pages 2425–2432, 2018. 8

[28] Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. Towards optimal structured cnn pruning via generative adversarial learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2790–2799, 2019. 8

[29] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 3

[30] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2736–2744, 2017. 1, 2

[31] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Tim Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. *arXiv preprint arXiv:1903.10258*, 2019. 3, 5, 7

[32] Jian-Hao Luo and Jianxin Wu. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. *arXiv preprint arXiv:1805.08941*, 2018. 8

[33] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017. 2, 8

[34] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 116–131, 2018. 2

[35] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019. 8

[36] Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566, 2017. 3

[37] Brendan O'Donoghue, Remi Munos, Koray Kavukcuoglu, and Volodymyr Mnih. Combining policy gradient and q-learning. *arXiv preprint arXiv:1611.01626*, 2016. 3

[38] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018. 3

[39] Ingo Rechenberg. *Evolutionsstrategie'94*. frommann-holzboog, 1994. 4

[40] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015. 6

[41] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017. 3

[42] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. 2, 6, 7, 8

[43] Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559, 2010. 4

[44] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. 2014. 3

[45] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. *arXiv preprint arXiv:1904.02877*, 2019. 3

[46] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000. 2

[47] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. *arXiv preprint arXiv:1807.11626*, 2018. 3

[48] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019. 3

[49] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992. 3

[50] Bichen Wu. Efficient deep neural networks. *arXiv preprint arXiv:1908.08926*, 2019. 2

[51] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. *arXiv preprint arXiv:1812.03443*, 2018. 3, 6

[52] Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. Exploring randomly wired neural networks for image recognition. *arXiv preprint arXiv:1904.01569*, 2019. 3
[53] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018. 3
[54] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 285–300, 2018. 1, 3, 7
[55] Sun Yi, Daan Wierstra, Tom Schaul, and Jürgen Schmidhuber. Stochastic search using the natural gradient. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1161–1168, 2009. 4
[56] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018. 1
[57] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018. 2
[58] Chenglong Zhao, Bingbing Ni, Jian Zhang, Qiwei Zhao, Wenjun Zhang, and Qi Tian. Variational convolutional neural network pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2780–2789, 2019. 8
[59] Yuefu Zhou, Ya Zhang, Yanfeng Wang, and Qi Tian. Accelerate cnn via recursive bayesian pruning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3306–3315, 2019. 8
[60] Bohan Zhuang, Jing Liu, Mingkui Tan, Lingqiao Liu, Ian Reid, and Chunhua Shen. Effective training of convolutional neural networks with low-bitwidth weights and activations. *arXiv preprint arXiv:1908.04680*, 2019. 2
[61] Bohan Zhuang, Lingqiao Liu, Mingkui Tan, Chunhua Shen, and Ian Reid. Training quantized network with auxiliary gradient module. *arXiv preprint arXiv:1903.11236*, 2019. 2
[62] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems*, pages 875–886, 2018. 1
[63] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018. 3