Multi-Granularity Pruning for Model Acceleration on Mobile Devices

Tianli Zhao[1,2,3,4], Xi Sheryl Zhang[2,3], Wentao Zhu[5], Jiaxing Wang[6] Sen Yang[7], Ji Liu[8], and Jian Cheng[2,3]⋆

[1] School of Artificial Intelligence, University of Chinese Academy of Sciences
[2] Institute of Automation, Chinese Academy of Sciences
[3] AIRIA, [4] Maicro.ai, [5] Amazon Video Prime,
[6] JD.com, [7] Snap Inc., [8] Kwai Inc.

A Details of evolutionary search

A.1 Feature design

In this section, we introduce the feature design used during evolutionary search. Denote $C_{\max} = \{c_{\max}^{(l)}\}_{l=1}^{L}$ as the maximum number of channels for each layer. Then for architecture (C, S) with number of channels C (for channel pruning) and layer-wise weight sparsity (for weight pruning), the feature of (C, S) can be denoted by:

$$f(C,S) = (\frac{C}{C_{\max}}, S) \tag{1}$$

In this way, all the dimensions of features are normalized to $[0, 1]$. It is straight forward to re-generate architectures from features:

$$f^{(-1)}(\tilde{C}, S) = (\lfloor \tilde{C} \times C_{\max} \rceil, S) \tag{2}$$

A.2 Crossover and mutation

In this section, we introduce details about the crossover and mutation operations used in our method.

Crossover. For crossover operation, we use the popular Simulated Binary Crossover operation (SBX). We start with the crossover operation between two scalar numbers α, β. After the crossover operation, two new individuals are generated:

$$\begin{aligned} \alpha^{new} &= 0.5 \times [(1+u)\alpha + (1-u)\beta] \\ \beta^{new} &= 0.5 \times [(1-u)\alpha + (1+u)\beta] \end{aligned} \tag{3}$$

where:

$$u = \begin{cases} (2p)^{\frac{1}{\mu_c+1}} & p \leq 0.5 \\ (\frac{1}{2(1-p)})^{\frac{1}{\mu_c+c}} & otherwise \end{cases} \tag{4}$$

⋆ Corresponding author.

Algorithm 1 Crossover

```
Input: The population $P$ with size $n$.
Output: The population $Q$ with size $n$.
1: $Q = \emptyset$
2: while $|Q| < n$ do
3:     Randomly sample two individuals $p_1, p_2$ from $P$.
4:     $Q = Q \cup SBX(p_1, p_2)$
5: end while
6: return $Q$
```

where p is a random number sampled from $[0, 1]$, and μ_c is a hyper-parameter that makes a trade-off between exploration and exploitation. For two $n - dim$ vectors $\boldsymbol{\alpha} = \{\alpha_i\}_{i=1}^{n}$ and $\boldsymbol{\beta} = \{\beta_i\}_{i=1}^{(n)}$, the crossover operation between $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ can be implemented by applying the crossover operation on each dimension of the two vectors:

$$\begin{aligned} \boldsymbol{\alpha}^{new} &= \{\alpha_i^{new}\}_{i=1}^{n} \\ \boldsymbol{\beta}^{new} &= \{\beta_i^{new}\}_{i=1}^{n} \end{aligned} \tag{5}$$

where α_i^{new} and β_i^{new} are the output of crossover operations between α_i and β_i. For ease of explanation, we denote:

$$\{\boldsymbol{\alpha}^{new}, \boldsymbol{\beta}^{new}\} = SBX(\boldsymbol{\alpha}, \boldsymbol{\beta}) \tag{6}$$

With the above operation, the crossover operation on a population P can be implemented with Algorithm 1.

Mutation. For mutation operation, we use the popular PoLynomial Mutation operation (PLM) for MOO genetic algorithms. We denote u_i, l_i as the upper bound and lower bound of the i^{th} dimension of individuals. The algorithm for mutation are summarized in Algorithm 2, where μ_m is a hyper-parameter.

Crossover mutation in our method. Having introduced the crossover and mutation operations, we are able to introduce the final crossover-mutation operations used in our method. The algorithm is summarized in Algorithm 3. In all of our algorithm, we set $\mu_c = 0.5$, $\mu_m = 15$, and the probability for mutation to 0.1.

B Model retraining

After searching, we get the optimal layer-wise channel width as well as the weight sparsity. The generated models are retrained on the whole train set of ImageNet and validated on validation set. We follow the conventional pretrain + prune + fine tune process for model retraining. Specifically, for an architecture (C, S), we first train a dense network with number of channels C. We then conduct the weight pruning with ADMM. By introducing an auxiliary variable and using the

Algorithm 2 Mutation

```
Input: A population $P$ with size $n$, and feature dimension $d$ for each individual. The mutation probability $p$.
Output: A mutated population $Q$.
 1: $Q = \emptyset$
 2: for each individual $p \in P$ do
 3:     $q = p$
 4:     for each dimension $i \in 1 \rightarrow d$ do
 5:         $\mu$ = randomly sample from $[0, 1]$.
 6:         if $\mu < p$ then
 7:             $d_1 = \frac{p_i - l_i}{u_i - l_i}, d_2 = \frac{u_i - p_i}{u_i - l_i}$
 8:             $r$ = randomly sample from $[0, 1]$.
 9:             $q_i = \begin{cases} [2r + (1-2r)(1-d_1)^{\mu_m+1}]^{\frac{1}{\mu_m+1}} & r \leq 0.5 \\ 1 - [2(1-r) + 2(r-0.5)(1-d_2)^{\mu_m+1}]^{\frac{1}{\mu_m+1}} & \end{cases}$
10:         end if
11:     end for
12:     $Q = Q \cup \{q\}$.
13: end for
14: return $Q$
```

Algorithm 3 Crossover and mutation

```
Input: A set of architectures $P = \{(C_i, S_i)\}_{i=1}^{n}$ with population size $n$.
Output: A set of architectures $Q$ generated from $P$.
    $\tilde{P} = \{f(C_i, S_i)\}_{i=1}^{n}$
    $\tilde{Q}$ = crossover between individuals in $P$ with Algorithm 1.
    Clip the values of $\tilde{Q}$ to $[0, 1]$.
    $\tilde{Q}$ = mutate $\tilde{Q}$ with Algorithm 2.
    Clip the values of $\tilde{Q}$ to $[0, 1]$.
    $Q = \{f^{(-1)}(\tilde{C}_i, S_i)\}_{(\tilde{C}_i, S_i) \in \tilde{Q}}$
    return $Q$
```

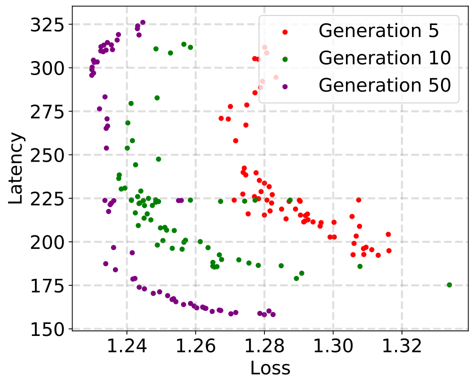

Fig. 1: Trends of frontier during evolution.

duality theorem, the primal parameters W, auxiliary variable U and the dual variable Z are updated alternatively:

$$\begin{aligned} W_{t+1} &= \arg\min_{W} \mathcal{L}(W) + \frac{\rho}{2}\|W - U_t + Z_t\|^2 \\ U_{t+1} &= \arg\min_{U} \|W_{t+1} - U + Z_t\|^2 \quad s.t. \quad \|U^{(l)}\|_0 = s^{(l)} \\ Z_{t+1} &= Z_t + (W_{t+1} - U_{t+1}) \end{aligned} \tag{7}$$

where $\mathcal{L}(W)$ is task specified loss function. U and Z are all of the same size as W, and $U^{(l)}$ is the auxiliary variable corresponding to parameters of the l^{th} layer. After ADMM steps, we fine tune the compressed model for 60 epochs. Detailed hyper parameters for model retraining are listed in Table 1, and the final algorithm of JCW is summarized in Algorithm 4.

C Linearity of latency w.r.t. sparsity

In figure 2 we plot the latency v.s. the number of input channels, the number of output channels and the weight sparsity for weight pruning. The data points are collected from 4 of the convolution layers of MobileNetV2. We can see that the latency of each layer is locally linear to the layer width and weight sparsity, this motivates us to approximate the latency of networks with tri-linear interpolation.
Derivation of tri-linear interpolation. We further derive the tri-linear interpolation of equation (9) in Section 2.5 of the paper.
1-d linear interpolation. We start from 1-d linear interpolation, which is illustrated in the left of Figure 3. Assume that we hope to approximate some function $f(\cdot)$ with an array of known data points $\{x_i, f_i\}_{i=0}^n$, where:

$$x_i = \frac{i}{n}x_n, \qquad f_i = f(x_i),$$

and x_n is the maximum value of x. Denote $\hat{f}(\cdot)$ to be the approximation to $f(\cdot)$ with linear interpolation in the sequence of known data points. Given any

Algorithm 4 JCW algorithm

```
Input: A supernet $\mathcal{N}$, the number of generations $N_G$, the population size $n$.
Output: A set of architectures $P = \{(C_1, S_1), \cdots, (C_n, S_n)\}$ with various latency and
    accuracy.
1: $P$ = Randomly sample $n$ architectures with various number of channels and weight
   sparsity for each layer.
2: for $g \in 1 \to N_G$ do
3:     $Q = crossover\_mutation(P)$ with Algorithm 3
4:     Reconstruct and reinitialize the super-net $\mathcal{N}$ to contain only models in $P \cup Q$.
5:     Train the super-net $\mathcal{N}$ with parameter sharing to estimate the accuracy rank of
       models in $P \cup Q$.
6:     Estimate the latency of models in $P \cup Q$ with trilinear interpolation as described
       in Section 2.5.
7:     Update $P$ with uniform non-dominated sorting selection as described in Section
       2.6.
8: end for
9: Train the models with architectural configurations in $P$ with ADMM.
```

Stage	Hyper parameter	Resnet18	MobileNetV1	MobileNetV2
Pretrain	batch size	512	512	512
	epochs	120	120	250
	lr	0.256	0.512	0.256
	lr annealing	cosine	cosine	cosine
	weight decay	1e-4	4e-5	4e-5
ADMM	batch size	512	512	512
	epochs	60	60	60
	lr	0.005	0.005	0.005
	lr annealing	constant	constant	constant
	weight decay	1e-4	4e-5	4e-5
	ρ	0.01	0.01	0.01
Fine tune	batch size	512	512	512
	epochs	60	60	60
	lr	0.005	0.005	0.005
	lr annealing	cosine	cosine	cosine
	weight decay	0	0	0

Table 1: Hyper-parameters for model re-training.

position x in the interval $[x_i, x_{i+1}]$, the approximated function $\hat{f}(x)$ is the straight line between the pair of data points (x_i, f_i) and (x_{i+1}, f_{i+1}), we then have:

$$\frac{f_{i+1} - f_i}{x_{i+1} - x_i} = \frac{\hat{f}(x) - f_i}{x - x_i}.$$

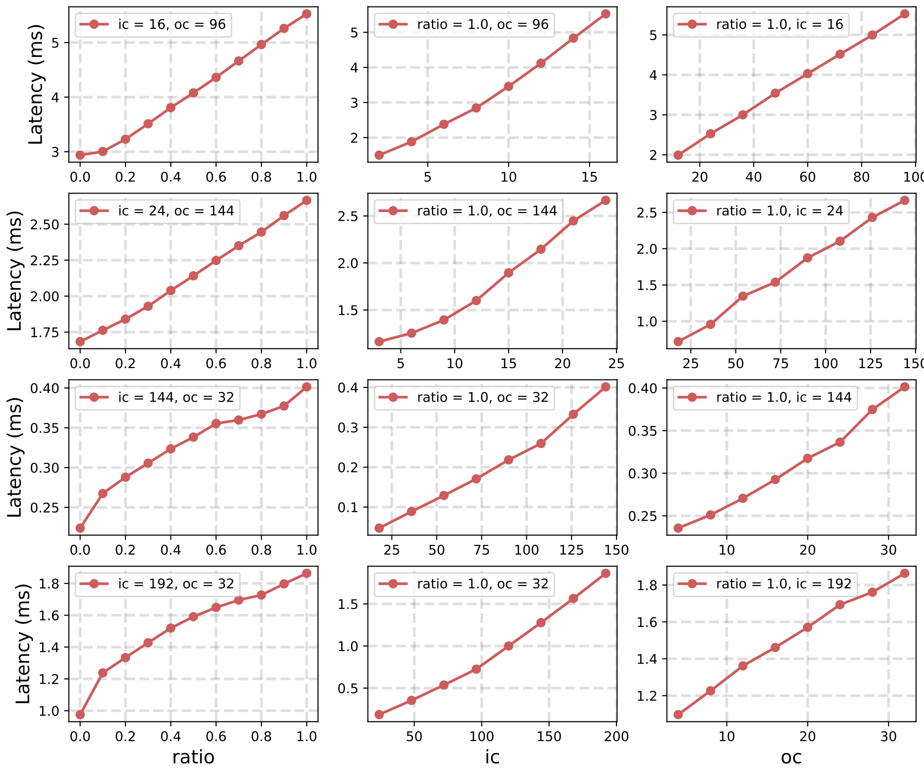

Fig. 2: Linearity of latency w.r.t. density ratio (left), input channels (middle) and output channels (right).

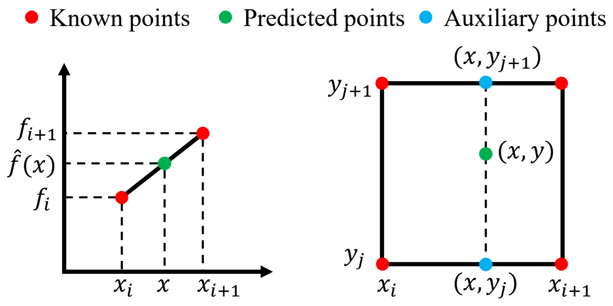

Fig. 3: An illustration of *left:* 1-d linear interpolation and *right:* 2-d bi-linear interpolation.

Solving the above linear equation, we have:

$$\begin{aligned}\hat{f}(x) = f_i \times \left(1 - \left|\frac{x - x_i}{\triangle_x}\right|\right) \\ + f_{i+1} \times \left(1 - \left|\frac{x - x_{i+1}}{\triangle_x}\right|\right),\end{aligned} \tag{8}$$

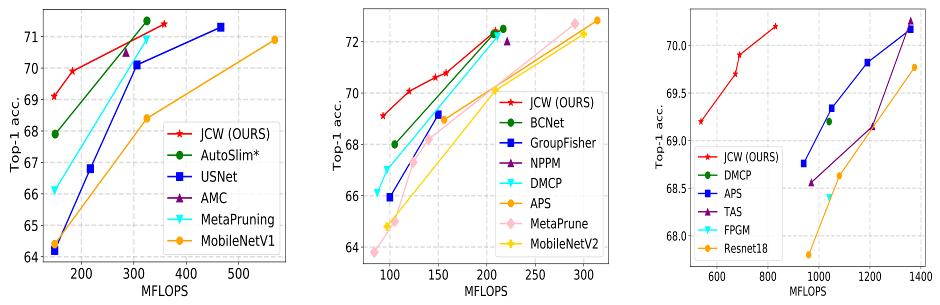

Fig. 4: FLOPS comparison for MobileNetV1 (left), MobileNetV2 (middle), and Resnet18 (right) on the ImageNet dataset. * denotes the results with knowledge distillation, which is not used in our JCW method. For BCNet [8] and DMCP [3], we report their results without knowledge distillation.

where:

$$\triangle_x = x_{i+1} - x_i = \frac{x_n}{n}.$$

Note that for any $j < i$, we have:

$$|x - x_j| = (x - x_i) + (x_i - x_j) \geq \triangle_x,$$

similarly, for any $j > i + 1$, we also have:

$$|x - x_j| \geq \triangle_x.$$

Thus, equation 8 can be further reorgnized by:

$$\begin{aligned} \hat{f}(x) &= \sum_{i=0}^{n} \tau(\frac{x - x_i}{\triangle_x}) f_i \\ &= \sum_{i=0}^{n} \tau(n\frac{x}{x_n} - i) f_i \end{aligned},$$

where:

$$\tau(x) = \max(0, 1 - |x|).$$

Multi dimensional linear interpolation. The linear interpolation in higher-dimensional spaces can be done by conducting linear interpolation along each dimension separately. In the right of Figure 3, we show a simple example for 2-d linear interpolation, or bi-linear interpolation. Specifically, in 2-d case, our goal is to approximate the values of some function $f : \mathbb{R}^2 \to \mathbb{R}$ with bi-linear interpolation given a grid of known data points $\{(x_i, y_j, f_{ij}); i = 0, 1 \cdots n, j = 0, 1 \cdots m\}$, where:

$$x_i = \frac{i}{n} x_n, \quad y_j = \frac{j}{m} y_m, \quad f_{ij} = f(x_i, y_j).$$

Given any point (x, y) such that:

$$x \in [x_i, x_{i+1}], \quad y \in [y_j, y_{j+1}],$$

the function value $f(x, y)$ can be then approximated in two steps. First, conduct the 1-d linear interpolation along the x-dimension, which gives:

$$\hat{f}(x, y_j) = \sum_i \tau(n\frac{x}{x_n} - i) f_{ij},$$

and then conduct the 1-d linear interpolation along the y-dimension, which further gives:

$$\begin{aligned}\hat{f}(x, y) &= \sum_j \tau(m\frac{y}{y_m} - j)\hat{f}(x, y_j) \\ &= \sum_{i,j} \tau(n\frac{x}{x_n} - i)\tau(m\frac{y}{y_m} - j) f_{ij}\end{aligned}.$$

The above derivation can be easily generalized to higher dimensional spaces. Particularly, in 3-d case, the tri-linear interpolation has the form as illustrated in equation 9 of our paper.

D Additional results

In our paper, we have provided extensive comparison of JCW with the very state of the art methods in terms of latency and accuracy. Here we provide more results of the comparison in terms of FLOPS and accuracy. The compared methods include BCNet [8], GroupFisher [6], NPPM [2], DMCP [3], APS [9], MetaPruning [7], AutoSlim [10], USNet [11], AMC [5], TAS [1] and FPGM [4]. Results are shown in Fig. 4.

From the figure, we see that although targeting latency, JCW still achieves overall better FLOPS-accuracy trade-off, especially in the low FLOPS region. This is reasonable because we include weight pruning in our framework, which is more flexible to achieve a high compression ratio.

Another observation in the figure is the advantage of JCW in terms of FLOPS and latency is more obvious on Resnet18. This is because that compared to compact models like MobileNetV1 and MobileNetV2, Resnet18 contains more parametric redundancies and thus is more insensitive to weight pruning. We believe that JCW will achieve much better FLOPS-accuracy trade-off on larger models, while *that is not our goal.* We focus on latency reduction in this paper.

References

1. Dong, X., Yang, Y.: Network pruning via transformable architecture search. In: Advances in Neural Information Processing Systems. pp. 760–771 (2019)
2. Gao, S., Huang, F., Cai, W., Huang, H.: Network pruning via performance maximization. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2021)
3. Guo, S., Wang, Y., Li, Q., Yan, J.: Dmcp: Differentiable markov channel pruning for neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1539–1547 (2020)
4. He, Y., Liu, P., Wang, Z., Hu, Z., Yang, Y.: Filter pruning via geometric median for deep convolutional neural networks acceleration. In: IEEE Conference on Computer Vision and Pattern Recognition, (CVPR) (2019)
5. He, Y., Lin, J., Liu, Z., Wang, H., Li, L.J., Han, S.: Amc: Automl for model compression and acceleration on mobile devices. In: in Proceedings of the European Conference on Computer Vision (ECCV) (September 2018)
6. Liu, L., Zhang, S., Kuang, Z., Zhou, A., Xue, J., Wang, X., Chen, Y., Yang, W., Liao, Q., Zhang, W.: Group fisher pruning for practical network compression. In: International Conference on Machine Learning, (ICML) (2021)
7. Liu, Z., Mu, H., Zhang, X., Guo, Z., Yang, X., Cheng, K.T., Sun, J.: Metapruning: Meta learning for automatic neural network channel pruning. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 3296–3305 (2019)
8. Su, X., You, S., Wang, F., Qian, C., Zhang, C., Xu, C.: Bcnet: Searching for network width with bilaterally coupled network. In: IEEE Conference on Computer Vision and Pattern Recognition, (CVPR) (2021)
9. Wang, J., Bai, H., Wu, J., Shi, X., Huang, J., King, I., Lyu, M., Cheng, J.: Revisiting parameter sharing for automatic neural channel number search. Advances in Neural Information Processing Systems **33** (2020)
10. Yu, J., Huang, T.S.: Autoslim: Towards one-shot architecture search for channel numbers. CoRR **abs/1903.11728** (2019), `http://arxiv.org/abs/1903.11728`
11. Yu, J., Huang, T.S.: Universally slimmable networks and improved training techniques. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 1803–1811 (2019)