

## A VISUALIZATION OF PRUNED ARCHITECTURES

In Fig 6, we visualize the pruned architecture for ResNet-50 and MobileNetV2.

## B BIAS OF FLOPS REGULARIZATION

We briefly discuss two types of FLOPs regularization used in our paper and trainable gate (TG) (Kim et al., 2020). First, we provide the specific definition of  $T(v_i)$  (FLOPs of  $i$ th layer):

$$T(v_i) = K_i^2 \frac{\mathbf{1}^T v_{i-1}}{\mathcal{G}_i} \mathbf{1}^T v_i W_i H_i, \quad (7)$$

where  $\mathcal{G}_i$  is the number of groups in a convolution layer,  $K_i$  is the kernel size,  $\mathbf{1}$  is a all one vector, and  $\mathbf{1}^T v_i$  is the number of perversed channels in  $i$ th layer. With  $T(v_i)$ ,  $T(\mathbf{v}) = \sum_{i=1}^L T(v_i)$ . In TG, they simply use mean square error (MSE) as the regularization term, and in their paper  $\mathcal{R}_{\text{MSE}}(T(\mathbf{v}), pT_{\text{total}}) = (T(\mathbf{v}) - pT_{\text{total}})^2$ . The gradients w.r.t  $v_i$  is:

$$\frac{\partial \mathcal{R}_{\text{MSE}}}{\partial v_i} = 2(T(\mathbf{v}) - pT_{\text{total}}) \frac{\partial T(v_i)}{\partial v_i}, \quad (8)$$

For the regularization used in our method:  $\mathcal{R}(T(\mathbf{v}), pT_{\text{total}}) = \log(|T(\mathbf{v}) - pT_{\text{total}}| + 1)$ , the gradients w.r.t  $v_i$  is:

$$\frac{\partial \mathcal{R}}{\partial v_i} = \frac{1}{|T(\mathbf{v}) - pT_{\text{total}}| + 1} \frac{T(\mathbf{v}) - pT_{\text{total}}}{|T(\mathbf{v}) - pT_{\text{total}}|} \frac{\partial T(v_i)}{\partial v_i}. \quad (9)$$

For both regularization functions, the ratio between the gradients w.r.t  $v_i$  of two layers  $k, j$  is

$\frac{\partial T(v_k)}{\partial v_k} / \frac{\partial T(v_j)}{\partial v_j} = \frac{K_k^2 \frac{\mathbf{1}^T v_{k-1}}{\mathcal{G}_k} W_k H_k}{K_j^2 \frac{\mathbf{1}^T v_{j-1}}{\mathcal{G}_j} W_j H_j}$ . Take ResNet-50 as an example, let  $j, k$  be the middle layers of a bottleneck block, and we random initialize HSN. If  $j$  is in the first block, and  $k$  is in the last block, then  $K_k = K_j = 3$ ,  $W_j = H_j = 56$ ,  $W_k = H_k = 7$ ,  $\mathbf{1}^T v_{j-1} \approx 0.5 \times 64$  (due to random initialization),  $\mathbf{1}^T v_{k-1} \approx 0.5 \times 512$ , finally,  $\frac{\partial T(v_k)}{\partial v_k} / \frac{\partial T(v_j)}{\partial v_j} \approx \frac{3 \times 3 \times 256 \times 7 \times 7}{3 \times 3 \times 32 \times 56 \times 56} \approx \frac{1}{8}$ , which is not trivial.

When calculating the gradients w.r.t  $\theta_i$ , we have  $\frac{\partial \mathcal{R}}{\partial \theta_i} = c_{\mathcal{R}} \frac{\partial T(v_i)}{\partial v_i} \frac{\partial v_i}{\partial \theta_i}$ , all  $\theta_i$  share the same  $c_{\mathcal{R}}$  decided by the regularization function. Without loss of generality, we assume the magnitude of  $\frac{\partial v_i}{\partial \theta_i}$  is similar given different layers. The assumption is based on the following derivation (to simplify derivation, we omit weight norm in dense layers):

$$\begin{aligned} \frac{\partial v_i}{\partial \theta_i} &= \frac{\partial z_i}{\partial \theta_i}, \\ &= \frac{\partial z_i}{\partial o_i} \frac{\partial o_i}{\partial \theta_i}, \\ &= \frac{1}{\tau} \text{sigmoid}((o_i + g)/\tau) (1 - \text{sigmoid}((o_i + g)/\tau)) \frac{\partial o_i}{\partial \theta_i} \leq \frac{1}{4\tau} b_i^T. \end{aligned}$$

where  $\text{sigmoid}(x)(1 - \text{sigmoid}(x)) \leq \frac{1}{4}$ , and  $b_i$  is the input to  $i$ th dense layer, which is also the outputs of GRU. Since all  $b_i$  have the same shape, and weights in GRU are normalized, we can assume all  $b_i$  have similar magnitude. Since  $\frac{1}{4\tau} b_i^T$  is a upper bound of  $\frac{\partial v_i}{\partial \theta_i}$ , similar assumptions can be made.

Following this assumption, the relative magnitude of gradients w.r.t  $\theta_j$  and  $\theta_k$  for layers  $j, k$  can be roughly represented by  $\frac{\partial T(v_k)}{\partial v_k} / \frac{\partial T(v_j)}{\partial v_j}$ . After training for a while, the ratio might be smaller, however, it only indicates that early layers are more aggressively pruned. Thus, when applying FLOPs regularization, it penalizes early layers much heavier compared to latter layers.

One should also note that this is a general problem when using gradient based model compression methods with the FLOPs regularization. It's quite hard to circumvent calculating  $\frac{\partial T(v_i)}{\partial v_i}$  as in TG (Kim et al., 2020) and our paper.

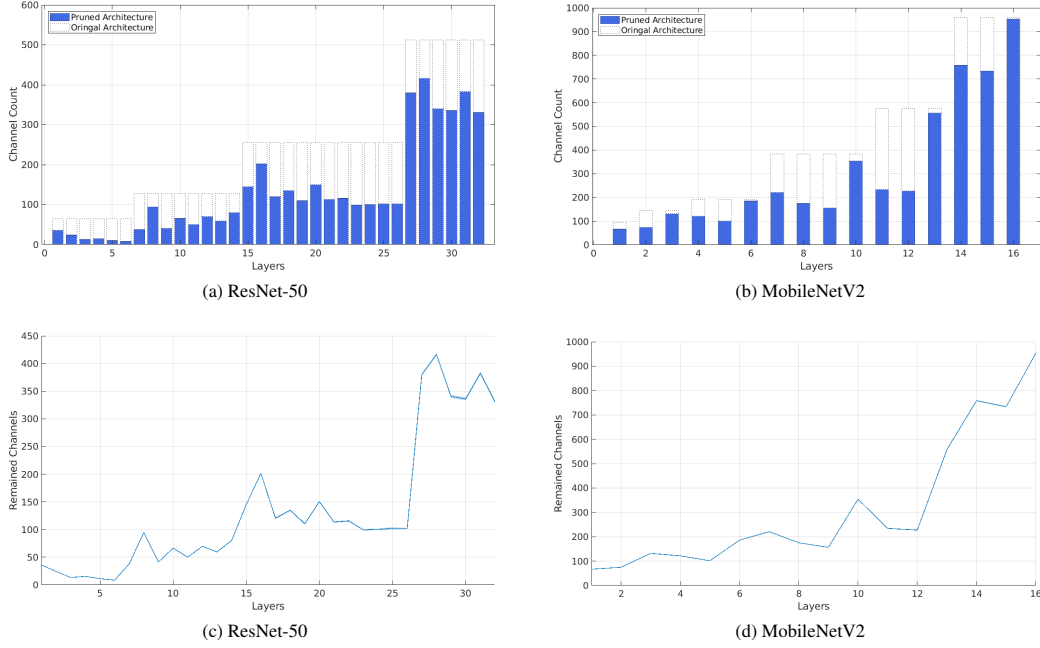


Figure 6: (a,b): visualization of pruned architectures for ResNet-50 and MobileNetV2. (c,d): mean and variance of 20 generated sub-networks for pruning.

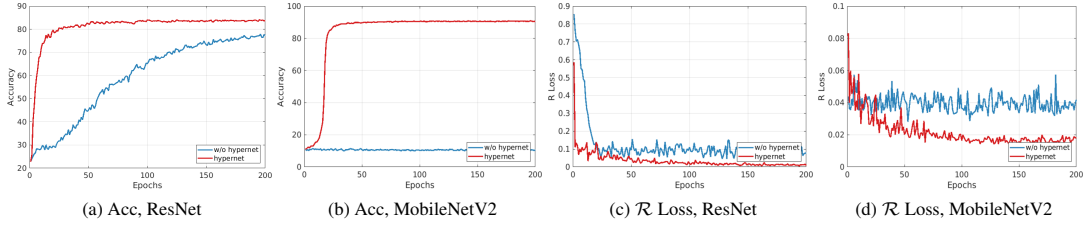


Figure 7: (a,b): Performance of sub-networks when using HSN or not using HSN (the setting in Eq. 11). (c,d): Regularization loss for the same settings.

## C DETAILED SETUP OF HYPER-STRUCTURE NETWORK

In Tab. 3, we present the architecture of HSN. The forward calculation is:

$$\begin{aligned} b_i, h_i &= \text{GRU}(a_i, h_{i-1}) \\ o_i &= \text{dense}_i(b_i) \end{aligned} \quad (10)$$

where  $h_i$  and  $b_i$  are hidden states and outputs of GRU at step  $i$ ,  $o_i$  is the final output of HSN. GRU also requires hidden layer input at time-step 0  $h_0$ . In the experiment, the  $h_0$  is a all zero tensor. As mentioned in Tab. 3, the dimension of  $a_i$  is 64. Since  $a_i$  is a single input instead of a mini-batch, we cannot apply batchnorm. To make the training more stable, we use weight norm (Salimans & Kingma, 2016) on both GRU and dense layers.

Initially, we tried to use a huge dense layer (input size 64, output size  $C_1 + C_2 + \dots + C_L$ ) as HSN. However, we find that the huge dense layer is hard to optimize and also parameter heavy.

To verify the strength of the proposed HSN, we can instead use a simplified setting to prune neural networks, which is shown as follows:

$$\begin{aligned} \hat{z}_i &= \text{sigmoid}((\hat{\theta}_i + g)/\tau), \\ \hat{v}_i &= \text{round}(\hat{z}_i), \text{ and } \hat{v}_i \in \{0, 1\}^{C_i}, \end{aligned} \quad (11)$$

Inputs $a_i, i=1, \dots, L$
GRU(64,128), WeightNorm, Relu dense $_i(128, C'_i)$ , WeightNorm, $i=1, \dots, L$
Outputs $o_i, i=1, \dots, L$

Table 3: The structure of HSN used in our method.

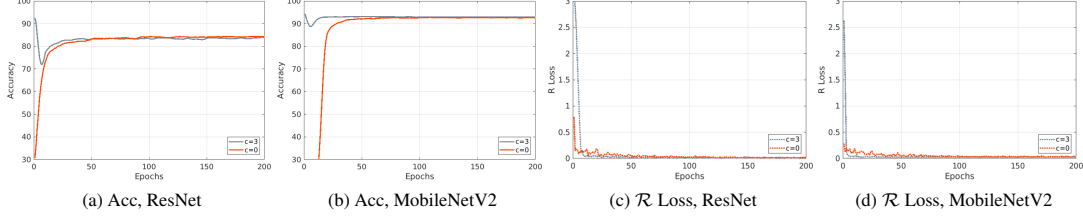


Figure 8: (a,b): Performance of sub-networks when training HSN given forward ( $c=0$ ) and backward pruning ( $c=3$ ). (c,d): Regularization loss of sub-networks when training HSN given forward ( $c=0$ ) and backward pruning ( $c=3$ ). All experiments are done on CIFAR-10.

where the architecture vector is parameterized by  $\hat{\theta}_i$ . Under this setting, the parameter for each channel does not have relationships. We use this setting to prune ResNet-56 and MobileNetV2 on CIFAR-10, the results are shown in Fig. 7. From the figure, we can see that the performance and convergence speed of using HSN is much better. Under high dimensional setting, like MobileNetV2, the simplified setting shown in Eq. 11 can not learn efficiently, which demonstrate that capturing inter-channel and inter-layer relationships are crucial for pruning deep neural networks.

## D FORWARD AND BACKWARD PRUNING

Here, we refer forward pruning as start pruning from a random sub-network, and refer backward pruning as start pruning from the original large model. Many model compression methods use backward pruning. We also provide a simple way to extend our method to backward pruning. When we binarize the output of HSN, we can add a constant  $c$ :

$$\begin{aligned} z_i &= \text{sigmoid}((o_i + (g + c))/\tau), \\ v_i &= \text{round}(z_i), \text{ and } v_i \in \{0, 1\}^{C_i}, \end{aligned} \quad (12)$$

where  $g \sim \text{Gumbel}(0, 1)$ , and the Gumbel(0, 1) distribution can be sampled using inverse transform sampling by drawing  $u \sim \mathcal{U}(0, 1)$  and computing  $g = -\log(-\log(u))$ . When the constant  $c$  is big enough, it will make  $v_i$  become an all one vector, thus the sub-network produced by HSN will start from the original large CNN. If we set  $c$  to 0, then it will start from a random sub-network. In Fig. 8, we show the results of forward and backward pruning. It can be seen that they can achieve similar sub-network performance, but the changes in regularization loss various dramatically.

## E DERIVATIVE OF HYPER-GRADIENT WITH ADAM OPTIMIZER

The update rule of ADAM for  $\theta_i$  is shown in Alg. 2, and it is:

$$u(\theta_i^{t-1}, \alpha_i^t) = \theta_i^{t-1} - \eta \hat{m}_t / (\sqrt{\hat{n}_t} + \epsilon), \quad (13)$$

**Algorithm 2:** ADAM optimizer for  $\theta_i$ **Input:**  $\eta, \beta_1, \beta_2 \in [0, 1]$ : learning rate and decay rate for ADAM.Initialize  $m_0, n_0, t = 0$ **Update rule at step  $t$ :**

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \left( \alpha_i^t \frac{\partial \mathcal{L}}{\partial \theta_i^{t-1}} + \lambda \frac{\partial \mathcal{R}}{\partial \theta_i^{t-1}} \right)$$

$$n_t = \beta_2 n_{t-1} + (1 - \beta_2) \left( \alpha_i^t \frac{\partial \mathcal{L}}{\partial \theta_i^{t-1}} + \lambda \frac{\partial \mathcal{R}}{\partial \theta_i^{t-1}} \right)^2$$

$$\hat{m}_t = m_t / (1 - \beta_1^t)$$

$$\hat{n}_t = n_t / (1 - \beta_2^t)$$

$$\theta_i^t = u(\theta_i^{t-1}, \alpha_i^t) = \theta_i^{t-1} - \eta \hat{m}_t / (\sqrt{\hat{n}_t} + \epsilon)$$

Then the derivation of  $\frac{\partial u(\theta_i^{t-1}, \alpha_i^t)}{\partial \alpha_i}$  is:

$$\begin{aligned} \frac{\partial u(\theta_i^{t-1}, \alpha_i^t)}{\partial \alpha_i} &= -\eta \frac{\partial (\hat{m}_t / (\sqrt{\hat{n}_t} + \epsilon))}{\partial \alpha_i} \\ &= -\eta \frac{-\frac{\partial \sqrt{\hat{n}_t} + \epsilon}{\partial \alpha_i} \hat{m}_t + \frac{\partial \hat{m}_t}{\partial \alpha_i} (\sqrt{\hat{n}_t} + \epsilon)}{(\sqrt{\hat{n}_t} + \epsilon)^2} \\ &= -\eta \left( \frac{\frac{\partial \hat{m}_t}{\partial \alpha_i}}{\sqrt{\hat{n}_t} + \epsilon} - \frac{\frac{\partial \hat{n}_t}{\partial \alpha_i} \hat{m}_t}{2\sqrt{\hat{n}_t}(\sqrt{\hat{n}_t} + \epsilon)^2} \right) \\ &= -\eta \left\{ \frac{(1 - \beta_1) \frac{\partial \mathcal{L}}{\partial \theta_i^{t-1}}}{(1 - \beta_1^t)(\sqrt{\hat{n}_t} + \epsilon)} - \frac{(1 - \beta_2) \left( \alpha_i^t \left( \frac{\partial \mathcal{L}}{\partial \theta_i^{t-1}} \right)^2 + \lambda \frac{\partial \mathcal{L}}{\partial \theta_i^{t-1}} \frac{\partial \mathcal{R}}{\partial \theta_i^{t-1}} \right) \hat{m}_t}{\sqrt{\hat{n}_t}(\sqrt{\hat{n}_t} + \epsilon)^2 (1 - \beta_2^t)} \right\}, \end{aligned}$$

where  $\frac{\partial \hat{m}_t}{\partial \alpha_i} = \frac{(1 - \beta_1) \frac{\partial \mathcal{L}}{\partial \theta_i^{t-1}}}{1 - \beta_1^t}$  and  $\frac{\partial \hat{n}_t}{\partial \alpha_i} = \frac{2(1 - \beta_2) \left( \alpha_i^t \left( \frac{\partial \mathcal{L}}{\partial \theta_i^{t-1}} \right)^2 + \lambda \frac{\partial \mathcal{L}}{\partial \theta_i^{t-1}} \frac{\partial \mathcal{R}}{\partial \theta_i^{t-1}} \right)}{1 - \beta_2^t}$ . Recall that when updating  $\alpha_i^{t-1}$  to  $\alpha_i^t$ , we have to compute:

$$\alpha_i^t = \alpha_i^{t-1} - \beta \frac{\partial J(u(\theta_i^{t-2}, \alpha_i^{t-1}))}{\partial \alpha_i} = \alpha_i^{t-1} - \beta \left( \frac{\partial \mathcal{J}}{\partial \theta_i^{t-1}} \right)^T \frac{\partial u(\theta_i^{t-2}, \alpha_i^{t-1})}{\partial \alpha_i}. \quad (14)$$

We need  $\alpha_i^t$  when updating  $\theta_i^{t-1}$  to  $\theta_i^t$ . Thus at each update step, it requires an extra copy of  $\theta_i^{t-2}$  and parameters of ADAM to compute  $\frac{\partial u(\theta_i^{t-2}, \alpha_i^{t-1})}{\partial \alpha_i}$ . The cost of the extra storage is trivial.

**F CHOICE OF  $p$  GIVEN DIFFERENT DATASETS AND ARCHITECTURES.**

Dataset	CIFAR-10		ImageNet			
Architecture	ResNet-56	MobileNetV2	ResNet-34	ResNet-50	ResNet-101	MobileNetV2
p	0.50	0.60	0.55	0.38	0.42	0.64

Table 4: Choice of  $p$  for different models.  $p$  is the **remained** FLOPs divided by the total FLOPs

In Tab. 4, we list the choices of  $p$  for different models and datasets used in our experiments.

**REFERENCES**

- Jaedeok Kim, Chiyoun Park, Hyun-Joo Jung, and Yoonsuck Choe. Plug-in, trainable gate for streamlining arbitrary neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.
- Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in neural information processing systems*, pp. 901–909, 2016.