

# Jointly Training and Pruning CNNs via Learnable Agent Guidance and Alignment

Alireza Ganjdanesh<sup>1\*</sup>Shangqian Gao<sup>2\*</sup>Heng Huang<sup>1†</sup><sup>1</sup> Department of Computer Science, University of Maryland College Park<sup>2</sup> Department of Electrical and Computer Engineering, University of Pittsburgh

{aliganj, heng}@umd.edu, shg84@pitt.edu

## Abstract

Structural model pruning is a prominent approach used for reducing the computational cost of Convolutional Neural Networks (CNNs) before their deployment on resource-constrained devices. Yet, the majority of proposed ideas require a pretrained model before pruning, which is costly to secure. In this paper, we propose a novel structural pruning approach to jointly learn the weights and structurally prune architectures of CNN models. The core element of our method is a Reinforcement Learning (RL) agent whose actions determine the pruning ratios of the CNN model's layers, and the resulting model's accuracy serves as its reward. We conduct the joint training and pruning by iteratively training the model's weights and the agent's policy, and we regularize the model's weights to align with the selected structure by the agent. The evolving model's weights result in a dynamic reward function for the agent, which prevents using prominent episodic RL methods with stationary environment assumption for our purpose. We address this challenge by designing a mechanism to model the complex changing dynamics of the reward function and provide a representation of it to the RL agent. To do so, we take a learnable embedding for each training epoch and employ a recurrent model to calculate a representation of the changing environment. We train the recurrent model and embeddings using a decoder model to reconstruct observed rewards. Such a design empowers our agent to effectively leverage episodic observations along with the environment representations to learn a proper policy to determine performant sub-networks of the CNN model. Our extensive experiments on CIFAR-10 and ImageNet using ResNets and MobileNets demonstrate the effectiveness of our method.

## 1. Introduction

Convolutional Neural Networks (CNNs) have enabled unprecedented achievements in the last decade in different domains [33, 69, 70, 81]. They have shown a trend for better performance when benefiting from deeper and wider architectures, larger dataset sizes, and longer training times with modern hardware [11, 58, 60, 88]. Despite their accomplishments, the tremendous memory and computational requirements of CNNs prohibit deploying them on edge devices with limited battery and compute resources, making CNN compression a crucial step before their deployment. The goal is to reduce the size and computational burden of CNNs while preserving their performance. Model pruning (removing weights [31] or structures [51] like channels and layers), weight quantization [68], knowledge distillation [40], Neural Architecture Search (NAS) [41, 98], and lightweight architecture designs [42, 71] are common categories of ideas for CNN compression.

Structural pruning which removes redundant channels of a CNN is the main focus of this paper. It is more practically plausible than weight pruning as it can effectively reduce the inference cost of a model on established hardware like GPUs without requiring special libraries [32] or post-processing steps. Further, it demands far less design efforts than NAS [30] and architecture design methods [62, 77]. The proposed structural pruning methods determine the importance of each channel using metrics such as resource loss [22], norm [51], and accuracy [59] and prune a model with techniques like greedy search [90] as well as evolutionary algorithms [8]. Thanks to the advances of Reinforcement Learning (RL) methods in complex decision making tasks [3, 16, 73], leveraging RL methods to determine proper sub-networks of a CNN given a desired budget has been explored in recent years [1, 36, 45, 89, 92, 95]. AMC [36] trains a DDPG agent [56] to prune layers of a pretrained CNN. LFP [45] trains an agent to get weights of a CNN's filters and determine keeping or pruning them. N2N [1] uses two agents to perform layer removal and

<sup>†</sup>This work was partially supported by NSF IIS 2347592, 2347604, 2348159, 2348169, DBI 2405416, CCF 2348306, CNS 2347617.

\*These authors contributed equally to this work.

layer shrinkage respectively. Finally, GNNRL [92] utilizes graph neural networks to identify CNN topologies and employ RL to find a proper compression policy. Despite their promising results, these models require a pretrained CNN model for training their RL agent for pruning as the prominent RL algorithms, like DDPG [56] used in AMC [36], cannot perform well in dynamic environments [48] if one trains model’s weights along with the agent’s policy.

We propose a novel model pruning method to jointly learn a CNN’s weights and structurally prune its architecture using an RL agent. As training the weights and pruning them cannot happen simultaneously, we apply a soft regularization term to the model’s weights during training to align with the sub-structure chosen by the best agent’s policy. We iteratively train the model’s weights for one epoch and perform several RL trajectories observations on the most recent model to update the policy of the RL agent. We design our RL agent so that in each of its episodic trajectories, its actions determine the pruning ratio for each layer of the model. After pruning all layers, we take the resulting model’s accuracy as our agent’s reward. However, as the model’s weights get updated in each epoch, the reward function of the RL agent changes dynamically between epochs. Accordingly, the training episodes of our agent are drawn from a non-stationary distribution. Therefore, one cannot simply employ prominent RL methods like DDPG [56] and Soft Actor-Critic (SAC) [29] in our framework to prune the model because their core assumption which is the environment being stationary [48] is not fulfilled. To overcome this challenge, we design a mechanism to model the evolving dynamics of the agent’s environment. We take an embedding for each epoch of the training and employ a recurrent model to determine a representation of the current state of the environment for the agent given the embeddings of the epochs so far. We train the recurrent model along with a decoder model in an unsupervised fashion to reconstruct the reward values observed in the agent’s trajectories. Finally, we augment each episodic trajectory of the agent with the representations of the state of the environment (provided by the recurrent model) at the time of the trajectory. By doing so, our RL agent has access to all information regarding the dynamic environment, and we employ SAC [29] to train the agent using the augmented trajectories. In addition, our soft regularization scheme for alignment of weights and the selected structure by the agent enables our pruned model to readily recover its performance in fine-tuning. We summarize our contributions as follows:

- We propose a novel channel pruning method that jointly learns the weights and prunes the architecture of a CNN model using an RL agent. In contrast with previous methods using RL for pruning, our method does not need a pretrained model before pruning.
- We perform joint training and pruning by iteratively train-

ing the model’s weights and the agent’s policy. We utilize a soft regularization technique to the model’s weights during training, encouraging them to align with the structure determined by the agent. By doing so, our method can identify a high-performing base model with weights that closely match the structure selected by the agent. Consequently, the pruned model can readily recover its high performance in fine-tuning.

- We design a mechanism to model the dynamics of our evolving pruning environment. To do so, we use a recurrent model that provides a representation of the state of the environment to the agent. We augment the trajectories observed by the agent using the provided representations to train the agent.

## 2. Related Work

**Model Pruning:** Model compression ideas [26, 63] can be categorized as structural pruning [20, 23–25, 51, 65, 66, 74, 86, 91, 97], weight quantization [6, 68, 83], weight pruning [18, 31, 94], Neural Architecture Search (NAS) [21, 87, 98], knowledge distillation [27], and lightweight architecture design [30, 77]. Structural pruning focusing on removing redundant channels (filters) of a CNN is the direction related to this paper. The proposed methods have approached this problem from various directions like pruning filters with smaller norms [51], applying group regularization [34] like LASSO [79] during training, ranking filters’ importance using low-rank decomposition [57], estimating influence of a filter on loss [64] using the Taylor decomposition, and meta-learning [59]. Recently, several ideas have employed Reinforcement Learning (RL) for pruning [1, 36, 45, 89, 92, 95]. LSEDN [95] trains an RL agent to perform layer-wise pruning on DenseNet [44]. N2N [1] proposed a two-stage process that employs two recurrent RL agents in which one agent removes layers of a pretrained CNN, and then, the other agent shrinks each remaining layer. LFP [45] introduces ‘try-and-learn’ scheme in which RL agents learn to take layers’ weights and predict binary masks for pruning or preserving layers’ filters. AMC [36] takes a pretrained CNN and trains a DDPG [56] to prune its convolution layers. Yet, these models can only prune pretrained models and cannot jointly train and prune the models. The main reason is that they employ off-the-shelf RL methods [29, 56] that assume stationary training environments, but if the model’s weights change during pruning, the environment will not be stationary. We develop a novel channel pruning method that jointly learns a CNN’s weights and prunes its architecture using an RL agent in an iterative manner. We design a procedure to model the changing dynamics of the reward function of our RL agent using a recurrent model that provides a representation of the current state of the environment. In addition, we regularize the model’s weights to align with the structure determined by

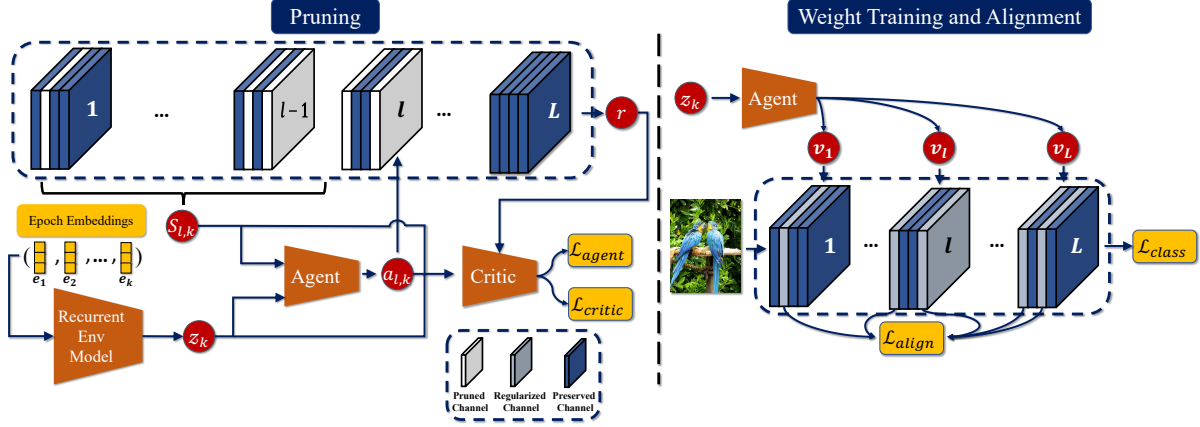


Figure 1. **Overview of our method.** We jointly train and prune a CNN model using an RL agent by iteratively training the agent’s policy and model’s weights. In each iteration, we train the model’s weights for one epoch and perform several episodic observations of the agent. **Left:** Each action of our agent prunes one layer of the model, and the procedure of pruning the  $l$ -th layer is shown. The agent’s actions on the previous layers and the remaining layers’ FLOPs determine its state, and we take the resulting model’s accuracy as its reward (Sec. 3.2). As the model’s weights change between iterations, the reward function also changes accordingly. Thus, we map each epoch to an embedding and employ a recurrent model to provide a state of the environment  $z$  to the agent. (Sec. 3.2.1) **Right:** Given a sub-network selected by the agent, we train the model’s weights while softly regularizing them to align with the selected structure (Sec. 3.2.3).

the agent. Thanks to such designs, our method can train a base model and select a performant sub-network of it that can easily recover its performance in fine-tuning.

**Reinforcement Learning:** RL methods [43, 84] have achieved outstanding results in complex tasks [3, 16, 73] using techniques like Q-learning [85] and policy optimization [76]. Yet, it has been shown that they cannot generalize to new variations of their primary task [2, 48]. Continual RL methods [48] are related to this paper as our agent’s environment is non-stationary. The proposed ideas address different non-stationarity conditions. For example, multi-task learning [93] and meta-learning methods [14, 17, 82] assume that sequential tasks presented to the agent have an unknown stationary distribution. Curriculum learning ideas [5, 67, 72] aim to learn the agent’s own curriculum in single or multi-agent dynamic environments. Finally, a group of methods [7, 12, 80] make assumptions on the variation budget of the reward function to improve learning of the agent. We refer to [48] for a comprehensive review on continual RL methods. Different from the mentioned ideas, we employ a recurrent model to model the changing dynamics of the environment of our agent and augment its observations with it to enable using episodic RL methods to train our agent.

### 3. Method

We propose a new channel pruning method for CNNs to jointly learn the weights of a CNN model and prune its filters using an RL agent. To do so, we iteratively train the model’s weights and the agent’s policy. We design the agent such that its actions determine the compression rate of the model’s layers, and we take the accuracy of the pruned

model as the reward function for the agent. Nevertheless, as we update the model’s weights iteratively, the reward associated with a certain action changes in consecutive iterations, which results in a non-stationary environment for the agent and prevents using episodic RL methods for our purpose as they assume the environment is stationary. We develop a mechanism in which a recurrent network models the changing dynamics of the reward function and provides a representation of the changing state of the environment to the agent to alleviate this challenge. Finally, as we cannot both train and prune the weights simultaneously during training, we propose a regularization to align the model’s weights with the selected sub-network by the agent. Fig. 1 shows the overall scheme of our method.

#### 3.1. Notations

We denote the number of layers in a CNN with  $L$  and the weights of its  $l$ -th convolution layer with  $\mathcal{W}_l \in \mathcal{R}^{C_{l+1} \times C_l \times W_l \times H_l}$ .  $C_{l+1}$ ,  $C_l$  represent the number of output and input channels of the layer, and  $W_l$  as well as  $H_l$  are the spatial dimensions of its kernel. We show the stride of the  $l$ -th layer with  $stride_l$ , and  $FLOPs[l]$  is its FLOPs value. Finally, we show the floor function with  $\lfloor \cdot \rfloor$ .

#### 3.2. Iterative Weight Training and Compression

We iteratively train a CNN’s weights and optimize the policy of an RL agent to jointly train and prune it. Specifically, in each iteration, we first train the model’s weights for one epoch on our training dataset while the agent is fixed. Then, we keep the model’s weights frozen and our agent observes several episodic trajectories by performing its actions on the CNN’s layers. To employ an RL agent to prune our model,

we need to define three main components for the agent: States that it visits in the environment, the actions that it can perform, and the reward function given states and actions. We describe our choices for each one in the following, and we denote the desired FLOPs budget for the pruned model with  $\text{FLOPs}_{\text{desire}}$ .

**States of the Agent:** We design our RL agent to perform actions that determine the pruning rate for consecutive layers of the CNN model. Thus, the agent’s states depend on the index of the layer that the agent is currently pruning; the layer’s characteristics like its kernel size and FLOPs; and the number of FLOPs that the agent has already pruned as well as the amount it has to prune from the remaining layers. Formally, given that the agent is currently pruning the  $l$ -th layer, we define the state of the environment as follows:

$$S_l = [l, C_l, C_{l+1}, \text{stride}_l, k_l, \text{FLOPs}[l], \text{FLOPs}_{1:l-1}, \text{FLOPs}_{l+1:L}, a_{l-1}] \quad (1)$$

where  $k_l$  is the layer’s kernel size.  $\text{FLOPs}_{1:l-1}$  denotes the number of previous layers’ FLOPs given the actions that the model has done so far on them.  $\text{FLOPs}_{l+1:L}$  shows the next layers’ FLOPs that are not pruned yet, and  $a_{l-1}$  is the agent’s action on the previous layer.

**Actions of the Agent:** Based on the state  $S_l$  for the  $l$ -th layer, our agent determines its pruning rate  $a_l$  such that  $a_l \in [0, 1)$ . Given the predicted pruning rate  $a_l$ , we remove  $\lfloor a_l \times c_l \rfloor$  channels of the layer. In addition, we calculate the minimum and maximum actual feasible pruning rates for the current layer based on  $\text{FLOPs}_{1:l-1}$ ,  $\text{FLOPs}[l]$ ,  $\text{FLOPs}_{l+1:L}$ , and the desired budget  $\text{FLOPs}_{\text{desire}}$ . Then, we bound the predicted action  $a_l$  to lie in the range  $[a_{l,\min}, a_{l,\max}]$ . We refer to supplementary materials for more details. In our experiments, we found that ranking the importance of filters using the norm criteria [51] and pruning the ones with the lowest rank works well in our framework, but one may employ more sophisticated approaches [57] as well.

**Reward Function:** We set our reward function to be the pruned model’s accuracy on a small held-out subset of the training dataset as a proxy for its final performance. As our agent prunes one layer of the model at a time, it will be extremely time-consuming to calculate the proxy value after each action of the agent. Thus, we take one pass of the agent on all layers of the model as one episodic trajectory for it. Then, we calculate the final pruned model’s accuracy on the subset at the end of the trajectory and take it as the reward value for all state-action pairs seen during the trajectory.

### 3.2.1 Modeling the Dynamic Nature of Rewards

As we iteratively train the model’s weights and the agent’s policy, the weights of the convolution layers that the agent performs its actions on them are not static in our framework. Thus, our reward function is dynamic in the course

of training, which prohibits directly applying prominent RL methods [29, 56] that leverage episodic trajectories to train the agent’s policy in our framework. The reason is that the optimization procedure of these models is biased to only optimize the agent’s policy *w.r.t* the current episode’s distribution and disregards the changing dynamics of the environment, resulting in a sub-optimal policy [48].

We design a new mechanism to overcome this challenge by providing a representation of the dynamic environment to the agent. To do so, first, we map the index of each epoch for training the model’s weights to an embedding. Then, we employ a recurrent GRU model [9] that takes a sequence of the embeddings corresponding to the epochs that have been passed so far and outputs a representation of the current state of the model’s weights. Formally, if we train the model’s weights for total  $T$  epochs, we denote the epoch embeddings corresponding to epoch indexes  $E = [e_1, e_2, \dots, e_T]$  with  $\Psi_{1:T} = [\psi_1, \psi_2, \dots, \psi_T]$  ( $\psi_1 = \text{Emb}(e_1)$ ,  $\text{Emb}$  is a learnable embedding layer). We calculate the representation of the state of the model’s weights at the epoch  $e_k$  as follows:

$$z_k = f_{\text{Env}}(\Psi_{1:k}, h_0; \theta_{\text{Env}}) \quad (2)$$

$f_{\text{Env}}$  denotes the recurrent model.  $\Psi_{1:k}$  are the embeddings of epochs until the epoch  $e_k$ .  $h_0$  is the initial hidden state of the GRU that we set it to a zero vector, and  $\theta_{\text{Env}}$  are the parameters of the GRU. We show in section 3.2.2 that we use the representations  $z$  provided by the GRU for training our RL agent.

We propose to train the recurrent model using another model that we call it ‘decoder.’ The decoder model takes 1) the state-action pairs  $(S, a)$  and 2) the representation  $z$  of the state of the environment when the agent observes  $(S, a)$  and predicts the agent’s reward  $r$ . Our intuition is that the representation  $z$  is informative of the state of the environment when the decoder can use it to accurately predict  $r$ . We train both the recurrent model’s weights and the ones for the decoder using the following objective:

$$\min_{\theta_{\text{Env}}, \theta_D} \mathcal{L}_{\text{recons}} = \mathbb{E}_{(s,a,r,e) \sim B} [(\hat{r} - r)^2] \quad (3)$$

$$\hat{r} = f_D(S, a, z; \theta_D) \quad (4)$$

$$z = f_{\text{Env}}(\Psi, h_0; \theta_{\text{Env}}) \quad (5)$$

In practice, we approximate the expectation in Eq. 3 using the agent’s episodic observations during training.  $f_D$  is our decoder model, and  $\theta_D$  represents its parameters.

### 3.2.2 RL Agent Training

We employ our recurrent model and the Soft Actor-Critic (SAC) [29] method to train our RL agent. We augment the states  $S$  with the representations of the environment’s state  $z$  and design our agent’s policy function so that it predicts a distribution over actions conditioned on both of them:



$$a \sim \pi(\cdot | S, z; \theta_A) \quad (6)$$

Similarly, we deploy the representations  $z$  when calculating the predicted Q-values by the critic networks in SAC. We train them using the mean squared Bellman error objective:

$$\mathcal{L}(\phi_i) = \mathbb{E}_{(s,a,r,s',d,e) \sim B} [(Q_{\phi_i}(s, a, z) - y(r, s', d, z))^2] \quad (7)$$

$$\begin{aligned} y(r, s', d, z) &= r + \gamma(1 - d) [\min_{j=1,2} Q_{\phi_{\text{tar},j}}(s', a', z) - \\ &\alpha \log(\pi(a' | s', z; \theta_A))]; \\ a' &\sim \pi(\cdot | s', z; \theta_A); z = f_{Env}(\Psi, h_0) \end{aligned} \quad (8)$$

$Q_{\phi_i}$  represents the critic models, and  $Q_{\phi_{\text{tar},i}}$  shows their target models obtained using Polyak averaging.  $B$  is a replay buffer containing previous episodic trajectories observed.  $(s, a)$  are state-action pairs from  $B$ .  $d$  indicates whether the state  $s$  is a terminal state.  $s'$  represents the state that the model gets in after taking the action  $a$  when being in the state  $s$ .  $a'$  is an action chosen using the most recent policy  $\pi(\cdot; \theta_A)$  conditioned on the state  $s'$  and environment representation  $z$ .  $\gamma$  is the discount factor for future rewards.  $\alpha$  determines the strength of the entropy regularization term, which is a hyperparameter. Finally, we train the agent's policy using the following objective:

$$\begin{aligned} \max_{\theta_A} \mathbb{E}_{(s,e) \sim B} [\min_{j=1,2} Q_{\phi_j}(s, a) - \alpha \log \pi(a | s, z; \theta_A)] \\ a \sim \pi(a | s, z; \theta_A) \end{aligned} \quad (9)$$

### 3.2.3 Soft Regularization of the Model's Weights

As mentioned in the section 3.2, we iteratively train and prune the model's weights in our framework. One approach to do so can be actually pruning the model's architecture by removing the redundant channels selected by the agent and only training the remaining ones in the weight training phase. However, doing so can make the training procedure unstable because it can significantly drop the model's accuracy. Accordingly, we propose an alternative approach to softly regularize the model's weights to align with the selected sub-network by the agent. Given binary architecture vectors  $[\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_L]$  denoting the channels selected by the current **best** agent for each layer, we use the following regularization term to train the model's weights:

$$\mathcal{L}_{\text{align}} = \sum_{l=1}^L \|(1 - \mathbf{v}_l) \odot \mathcal{W}_l\|_2 \quad (10)$$

Here,  $\odot$  means element-wise product and the proposed objective applies the Group Lasso regularization on the channels removed by the agent. Finally we combine the standard Cross Entropy Loss ( $\mathcal{L}_{\text{class}}$ ) with proposed  $\mathcal{L}_{\text{align}}$  to train the model's weights:

$$\mathcal{L}_w = \mathcal{L}_{\text{class}} + \beta \mathcal{L}_{\text{align}} \quad (11)$$

---

#### Algorithm 1: Joint Training and Pruning

---

**Input:** Training dataset  $\mathcal{D} = \{(x_i, y_i)\}$ ; replay buffer  $B$ ; CNN model  $f_c(\cdot; W)$  with  $L$  layers; Agent  $\pi(\cdot; \theta_A)$ ; Two Critics  $Q_{\phi_i}(\cdot)$  and their target models  $Q_{\phi_{\text{tar},i}}$  ( $i \in \{1, 2\}$ ); recurrent model  $f_{Env}$  and epoch embeddings  $\Psi$ ; regularization parameters  $\alpha, \beta$ ; discount factor  $\gamma$ ; number of iterations  $T$ ; number of pruning episodes per iteration  $P$ ; a subset  $\mathcal{D}_s$  of  $\mathcal{D}$  for calculating the agent's reward.

**for**  $t := 1$  **to**  $T$  **do**

    /\* Representation of Environment \*/  
    1. Calculate  $z_t$  using  $f_{Env}$  and embeddings  $\Psi$  in Eq. (2).  
    /\* RL Agent Exploration and Training \*/

**for**  $p := 1$  **to**  $P$  **do**

2. Prune the  $L$  layers of the CNN  $f_c$  one at a time by calculating states  $S_{l,p,t}$  and actions  $a_{l,p,t}$  using  $z_t$  and Eqs. (1,6).
3. Calculate the reward  $r_{p,k}$  using the final pruned model and  $\mathcal{D}_s$ .
4. Add the experiences  $(S_{l,p,t}, a_{l,p,t}, S_{l+1,p,t}, e_t, r_{p,k})$  to the replay buffer  $B$ .

**end**

5. Sample a batch of previous experiences  $\mathcal{B}$  from  $B$  and use them to calculate the loss value for the recurrent model, decoder, and epoch embeddings using Eq. (3). Then, update their parameters using the Adam optimizer.
6. Use the samples in  $\mathcal{B}$  to calculate the loss values of the critics  $Q_{\phi_i}(\cdot)$  and the agent  $\pi(\cdot; \theta_A)$  using Eqs. (7, 9).
7. Backpropagate the gradients of the calculated losses and update the parameters of the two critic models and the agent using the Adam optimizer.

    /\* Training the CNN's Weights \*/

8. Use the policy  $\pi(\cdot)$  with the highest reward so far to determine the binary architecture vectors  $[\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_L]$ .
9. Calculate the loss  $\mathcal{L}_w$  for the model's weights using Eqs. (10, 11). Backpropagate its gradients and update the model's parameters using SGD.

**end**

**Return:** Trained CNN model and agent.

---

In practice, we apply  $\mathcal{L}_{\text{align}}$  using the version of the policy with the highest reward until the current training iteration to make the training more stable. We summarize our training algorithm for training the model's weights and optimizing the agent's policy in Alg. 1.

Table 1. Comparison results on CIFAR-10 for pruning ResNet-56 and MobileNet-V2.

Model	Method	Baseline Acc	Pruned Acc	$\Delta$ -Acc	Pruned FLOPs
ResNet-56	DCP-Adapt [97]	93.80%	93.81%	+0.01%	47.0%
	SCP [47]	93.69%	93.23%	−0.46%	51.5%
	FPGM [37]	93.59%	92.93%	−0.66%	52.6%
	SFP [35]	93.59%	92.26%	−1.33%	52.6%
	AMC [36]	92.80%	91.90%	−0.90%	50.0%
	FPC [38]	93.59%	93.24%	−0.25%	52.9%
	HRank [57]	93.26%	92.17%	−0.09%	50.0%
	DTP [54]	93.36%	93.46%	+0.10%	50.0%
	RLAL (ours)	93.41%	<b>93.86%</b>	<b>+ 0.45%</b>	50.0%
MobileNet-V2	Uniform [97]	94.47%	94.17%	−0.30%	26.0%
	SCOP [78]	94.48%	94.24%	−0.24%	26.0%
	MDP [28]	95.02%	95.14%	+0.12%	28.7%
	DCP [97]	94.47%	94.69%	+0.22%	40.3%
	DDNP [23]	94.58%	94.81%	+ 0.23%	43.0%
	RLAL (ours)	94.48%	<b>94.85%</b>	<b>+ 0.37%</b>	<b>49.4%</b>

## 4. Experiments

We conduct experiments on ImageNet [10] and CIFAR-10 [50] to analyze the performance of our method. For all experiments, we use fully connected models with two hidden layers of size 300 for the architecture of the actor, two critics, and two target models of the critic models. We train the agent and critic models using the Adam optimizer [49] with learning rate of  $1e-4$  and  $1e-3$  respectively. We use exponential decay rates of  $(\beta_1, \beta_2) = (0.9, 0.999)$  for all of them. For our recurrent model, we use a GRU [9] model with the input size of 128. We also take embeddings of size 128 for all epochs (Sec. 3.2.1). We employ a fully connected model with two hidden layers of size 300 as our decoder model. We train the GRU model, epoch embeddings, and the decoder model using the Adam optimizer with learning rate of  $1e-3$  and decay parameters of  $(\beta_1, \beta_2) = (0.9, 0.999)$ . We set the entropy regularization coefficient  $\alpha$  to 0.1 and  $\beta$  for soft regularization to  $1e-4$  for all models. Finally, we choose the number of episodic observations per epoch for our agent to be  $P = 10$  (see Alg. 1). In all experiments, as we jointly train and prune our model by using **Reinforcement Learning** and softly **Aligning** the weights of the model with the selected sub-networks, we call our method **RLAL**. We refer to supplementary materials for more details of our experiments.

### 4.1. CIFAR-10 Results

Tab. 1 summarizes comparison results on the CIFAR-10 dataset. As can be seen, for ResNet-56, RLAL can achieve the best accuracy vs. computational efficiency trade-off compared to the baseline methods. On the one hand, it is able to prune FLOPs with a rate comparable to ( $< 3\%$  lower) FPC [38] while achieving +0.70 higher  $\Delta$ -Acc. On the other hand, only RLAL, DTP [54], and DCP-Adapt are able to outperform their baseline methods. RLAL can both prune 3% more FLOPs and accomplish 0.44% better  $\Delta$ -

Acc than DCP-Adapt. It also has 0.44% higher  $\Delta$ -Acc than DTP with the same FLOPs reduction ratio. Finally, with the same FLOPs pruning rate, RLAL significantly outperforms AMC [36] with 1.35% higher  $\Delta$ -Acc. For MobileNet-V2, RLAL can attain the highest  $\Delta$ -Acc while having the largest pruning rate at the same time. It remarkably prunes 6.4% more FLOPs than DDNP [23] while obtaining 0.14% higher  $\Delta$ -Acc. In summary, these results demonstrate the effectiveness of our method for finding efficient yet accurate models.

### 4.2. ImageNet Results

We present the experimental results on ImageNet in Tab. 2. For ResNet-18, RLAL shows the best  $\Delta$ -Acc Top-1/5 while showing a competitive pruning rate. It has a similar pruning rate (only 1% lower) to GNNRL [92] and achieves 0.30% higher  $\Delta$ -Acc Top-1. For pruning ResNet-34, RLAL is able to find a proper balance between accuracy and efficiency of the model. For instance, with a similar computation budget to GP [39] (only 1.1% FLOPs difference), RLAL’s pruned model has 1%/0.46% higher  $\Delta$ -Acc Top-1/5. Moreover, RLAL shows better final accuracies than ISP [19] while significantly pruning more FLOPs with a 6% margin. Pruning MobileNet-V2 is more challenging compared to ResNets because MobileNets [42, 71] are primarily designed for efficient inference. Accordingly, improvements in metrics are more difficult to secure than ResNet cases. We can observe that all methods have close FLOPs pruning rates in a relatively small range from 28.3% to 30.7%. RLAL can achieve 0.5% higher  $\Delta$ -Acc Top-1 while pruning only 0.6% lower FLOPs compared to AMC, and it has the best  $\Delta$ -Acc Top-1 among baselines. These results illustrate the capability of our method to effectively prune both large and small size models. Further, we highlight the advantages of our method compared to baseline RL-based pruning methods, GNNRL [92] and AMC [36], as it can obtain more accurate pruned models while not re-

Table 2. Comparison results on ImageNet for pruning ResNet-18/34 and MobileNet-V2.

Model	Method	Baseline Top-1 Acc	Baseline Top-5 Acc	$\Delta$ -Acc Top-1	$\Delta$ -Acc Top-5	Pruned FLOPs
ResNet-18	MIL [13]	70.28%	89.63%	-3.18%	-1.85%	41.8%
	SFP [35]	70.28%	89.63%	-3.18%	-1.85%	41.8%
	FPGM [37]	70.28%	89.63%	-1.87%	-1.15%	41.8%
	PFP [55]	69.74%	89.07%	-2.36%	-1.16%	29.3%
	SCOP [78]	69.76%	89.08%	-1.14%	-0.93%	45.0%
	GNNRL [92]	69.76%	-	-1.10%	-	51.0%
	GP [39]	70.28%	89.63%	-1.40%	-0.97%	43.9%
	PGMPF [4]	70.23%	89.51%	-3.56%	-2.15%	53.5%
	FTWT [15]	69.76%	-	-2.27%	-	51.5%
	EEMC [96]	70.28%	89.63%	-2.01%	-1.19%	46.6%
	RLAL (Ours)	69.80%	89.10%	<b>-0.80%</b>	<b>-0.42%</b>	<b>50.0%</b>
ResNet-34	SFP [35]	73.92%	91.62%	-2.09%	-1.29%	56.0%
	FPGM [37]	73.92%	91.62%	-1.29%	-0.54%	41.1%
	Taylor [65]	73.31%	-	-0.48%	-	24.2%
	SCOP [78]	73.31%	91.42%	-0.69%	-0.44%	44.8%
	GP [39]	73.92%	91.62%	-1.14%	-0.69%	51.1%
	DMC [22]	73.30%	91.42%	-0.73%	-0.31%	43.4%
	PGMPF [4]	73.27%	91.43%	-1.68%	-0.98%	52.7%
	FTWT [15]	73.30%	-	-1.59%	-	52.2%
	GFS [90]	73.31%	-	-0.40%	-	43.8%
	ISP [19]	73.31%	91.42%	-0.45%	-0.40%	44.0%
	RLAL (Ours)	73.45%	91.48%	<b>-0.14%</b>	<b>-0.23%</b>	<b>50.0%</b>
MobileNet-V2	Uniform [71]	71.80%	91.00%	-2.00%	-1.40%	30.0%
	AMC [36]	71.80%	-	-1.00%	-	30.0%
	Random [53]	71.88%	-	-0.98%	-	28.9%
	CC [52]	71.88%	-	-0.97%	-	28.3%
	MetaPruning [59]	72.00%	-	-0.80%	-	<b>30.7%</b>
	RLAL (ours)	71.82%	90.26%	<b>-0.50%</b>	<b>-0.33%</b>	29.4%

quiring a pretrained model for pruning.

### 4.3. Ablation Studies

We conduct ablation experiments to explore our method’s behavior by studying 1) the effect of changing the number of episodic observations of the agent in each epoch and 2) the advantage of using our soft regularization, epoch embeddings, and the recurrent environment model in our framework. We refer to supplementary materials for details of experimental settings.

**Changing the Number of Episodes:** We experiment using a ResNet-56 [33] model on CIFAR-10 with three different pruning rates in {35%, 50%, 65%}, and we set the number of episodic observations for our agent in each epoch from {5, 10, 15}. For each pruning ratio, we visualize the best reward that the agent achieves during the training vs. the epoch numbers. The results are shown in Fig. 2 (a-c). We can observe a common trend in all cases that increasing the number of episodes results in a higher final reward, especially, the higher number of episodes benefits more when the desired compression ratio is larger at the cost of longer training time. However, if the number of episodes is large enough, our method can attain a decent final reward value in a reasonable time.

**Benefit of the Recurrent Environment Model:** In our second experiment, we prune and finetune ResNet-56 and MobileNet-V2 [71] with three pruning rates in {35%, 50%, 65%} while using/dropping our mechanism to

Table 3. Ablation Results of our method for pruning ResNet-56 on the CIFAR-10 dataset. EE represents the Epoch Embeddings. SR represents the Soft Regularization in Eq. 10.

Setting	Baseline Acc	Pruned Acc	$\Delta$ -Acc	Pruned FLOPs
w/o EE	93.47%	93.44%	-0.03%	50%
w/o EE + w/o SR	93.33%	93.12%	-0.21%	
Ours	93.41%	93.86%	+0.45%	

provide a representation of the environment to the agent using the epoch embeddings and our recurrent model. We visualize the best reward of the agent in the course of training. The results for ResNet-56 and MobileNet-V2 are shown in Fig. 2 (d-f) and Fig. 2 (g-i), respectively. The cases using/not using our mechanism are shown with ‘w Emb’/‘w/o Emb.’ The results clearly demonstrate the benefit of our design that provides a representation of the environment to the agent. We can find that ‘w/o Emb’ cases commonly reach to a relatively high reward but cannot properly deal with the dynamic reward function for their agent to further improve their policy. In contrast, our method can consistently enhance its policy to reach higher reward values during training.

In our third experiment, we prune ResNet-56 on CIFAR-10 with two settings: 1) not using the recurrent model and epoch embeddings to provide representations of the environment to the agent 2) neither using the recurrent model and epoch embeddings nor the soft regularization. We

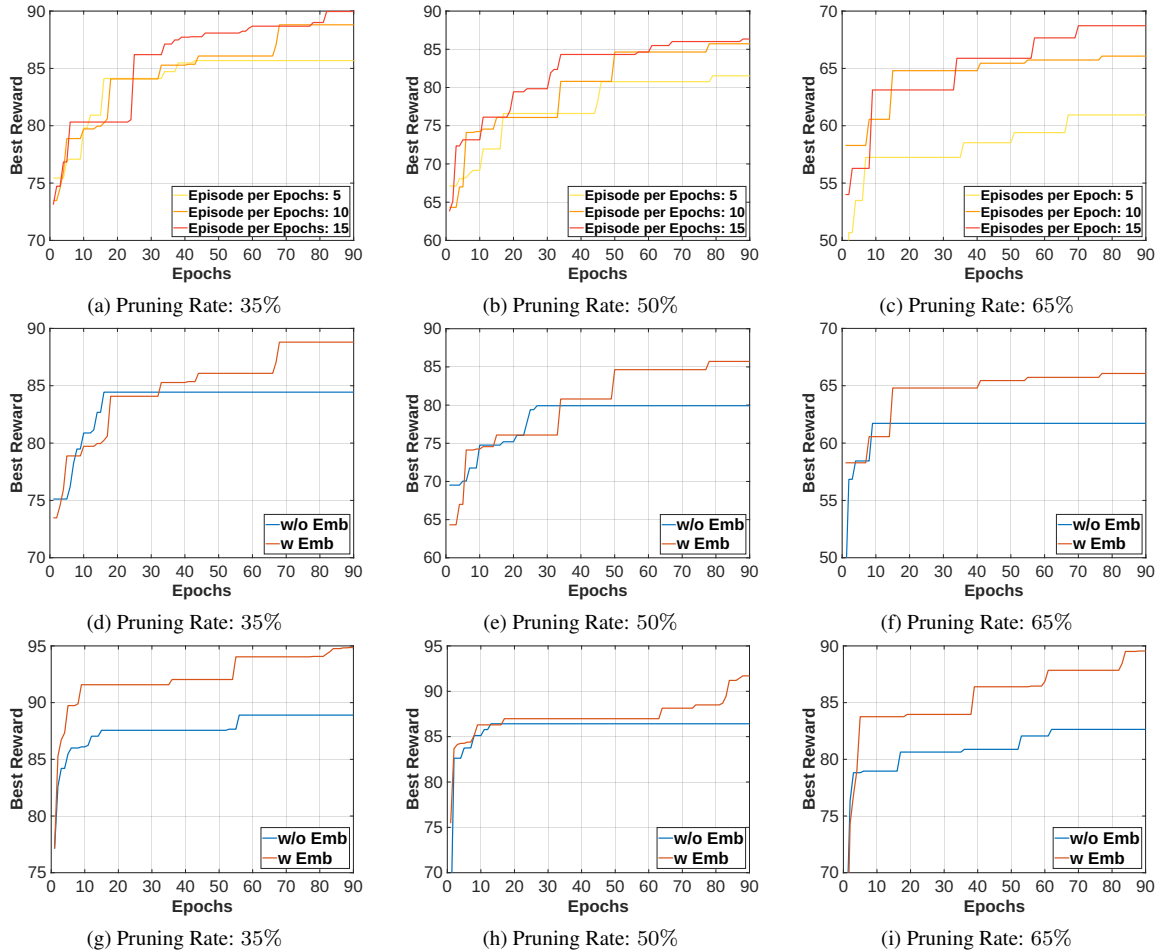


Figure 2. Results of ablation experiments on CIFAR-10. **(a-c)** Best reward of our agent when using a different number of episodes per epoch for three pruning rates when pruning ResNet-56. **(d-f)** Best reward with/without using our mechanism to provide representations of the environment to our agent during training for three pruning rates for ResNet-56. **(g-i)** Same results of **(d-f)** for MobileNet-V2.

present the results in Tab. 3. One can notice that removing each component of our method degrades its performance, especially not using the recurrent model and epoch embeddings severely degrades our method’s accuracy, which is inline with the results presented in Fig. 2 and discussed above. In summary, our ablation studies illustrate the effectiveness of our design choices in our method for jointly training and pruning a CNN model.

## 5. Conclusion

We proposed a method for structural pruning of a CNN model that jointly trains its weights and prunes its channels using an RL agent. Our method iteratively updates the model’s weights and allows the agent to observe several episodic pruning trajectories that it performs on the model to update its policy. Our agent’s actions determine the pruning ratios for the layers of the model, and we set the resulting model’s accuracy to be the agent’s reward. Such a design brings about a dynamic reward function for the agent. Thus,

we developed a mechanism to model the complex dynamics of the reward function and yield a representation of it to the agent. To do so, we mapped the index of each epoch of the training to an embedding. Then, we employed a recurrent model that takes the embeddings and provides a representation of the evolving environment’s state to the agent. We train the recurrent model and embeddings by utilizing a decoder model that predicts the agent’s rewards given observed states, actions, and environment representations predicted by the recurrent model. Finally, we regularized the model’s weights to align with the sub-network selected by the agent’s policy with the highest reward during training. Our designs enable the agent to effectively leverage the environment representations along with its episodic observations to learn a proper policy for pruning the model while interacting in our non-stationary pruning environment. Our experiments on ImageNet and CIFAR-10 demonstrate that our method can achieve competitive results with prior methods, especially the ones that use RL for pruning, while not requiring a pretrained model before pruning like them.



## References

- [1] Anubhav Ashok, Nicholas Rhinehart, Fares Beainy, and Kris M. Kitani. N2n learning: Network to network compression via policy gradient reinforcement learning. In *International Conference on Learning Representations*, 2018. 1, 2
- [2] Emmanuel Bengio, Joelle Pineau, and Doina Precup. Interference and generalization in temporal difference learning. In *International Conference on Machine Learning*, pages 767–777. PMLR, 2020. 3
- [3] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019. 1, 3
- [4] Linhang Cai, Zhulin An, Chuanguang Yang, Yangchun Yan, and Yongjun Xu. Prior gradient mask guided pruning-aware fine-tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 140–148, 2022. 7
- [5] Jiayu Chen, Yuanxin Zhang, Yuanfan Xu, Huimin Ma, Huazhong Yang, Jiaming Song, Yu Wang, and Yi Wu. Variational automatic curriculum learning for sparse-reward cooperative multi-agent problems. *Advances in Neural Information Processing Systems*, 34:9681–9693, 2021. 3
- [6] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International conference on machine learning*, pages 2285–2294, 2015. 2
- [7] Wang Chi Cheung, David Simchi-Levi, and Ruihao Zhu. Reinforcement learning for non-stationary markov decision processes: The blessing of (more) optimism. In *International Conference on Machine Learning*, pages 1843–1854. PMLR, 2020. 3
- [8] Ting-Wu Chin, Ruizhou Ding, Cha Zhang, and Diana Marculescu. Towards efficient model compression via learned global ranking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1518–1528, 2020. 1
- [9] Kyunghyun Cho, B van Merriënboer, Caglar Gulcehre, F Bougares, H Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, 2014. 4, 6
- [10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 6, 13
- [11] Piotr Dollár, Mannat Singh, and Ross Girshick. Fast and accurate model scaling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 924–932, 2021. 1
- [12] Omar Darwiche Domingues, Pierre Ménard, Matteo Pirotta, Emilie Kaufmann, and Michal Valko. A kernel-based approach to non-stationary reinforcement learning in metric spaces. In *International Conference on Artificial Intelligence and Statistics*, pages 3538–3546. PMLR, 2021. 3
- [13] Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. More is less: A more complicated network with less inference complexity. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5840–5848, 2017. 7
- [14] Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. RL<sup>2</sup>: Fast reinforcement learning via slow reinforcement learning, 2017. 3
- [15] Sara Elkerdawy, Mostafa Elhoushi, Hong Zhang, and Nilanjan Ray. Fire together wire together: A dynamic pruning approach with self-supervised mask prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12454–12463, 2022. 7
- [16] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatin, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610 (7930):47–53, 2022. 1, 3
- [17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017. 3
- [18] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. 2
- [19] Alireza Ganjdanesh, Shangqian Gao, and Heng Huang. Interpretations steered network pruning via amortized inferred saliency maps. In *Computer Vision—ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXI*, pages 278–296. Springer, 2022. 6, 7
- [20] Alireza Ganjdanesh, Shangqian Gao, Hiran Alipanah, and Heng Huang. Compressing image-to-image translation gans using local density structures on their learned manifold. *arXiv preprint arXiv:2312.14776*, 2023. 2
- [21] Alireza Ganjdanesh, Shangqian Gao, and Heng Huang. Eff-conv: efficient learning of kernel sizes for convolution layers of cnns. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 7604–7612, 2023. 2
- [22] Shangqian Gao, Feihu Huang, Jian Pei, and Heng Huang. Discrete model compression with resource constraint for deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1899–1908, 2020. 1, 7
- [23] Shangqian Gao, Feihu Huang, Yanfu Zhang, and Heng Huang. Disentangled differentiable network pruning. In *Computer Vision—ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XI*, pages 328–345. Springer, 2022. 2, 6
- [24] Shangqian Gao, Burak Uzkent, Yilin Shen, Heng Huang, and Hongxia Jin. Learning to jointly share and prune weights for grounding based vision and language models. In *The Eleventh International Conference on Learning Representations*, 2022.
- [25] Shangqian Gao, Zeyu Zhang, Yanfu Zhang, Feihu Huang, and Heng Huang. Structural alignment for network pruning through partial regularization. In *Proceedings of the*

- IEEE/CVF International Conference on Computer Vision*, pages 17402–17412, 2023. 2
- [26] Deepak Ghimire, Dayoung Kil, and Seong-heum Kim. A survey on efficient convolutional neural networks and hardware acceleration. *Electronics*, 11(6):945, 2022. 2
- [27] Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. Knowledge distillation: A survey. *Int. J. Comput. Vis.*, 129(6):1789–1819, 2021. 2
- [28] Jinyang Guo, Wanli Ouyang, and Dong Xu. Multi-dimensional pruning: A unified framework for model compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1508–1517, 2020. 6
- [29] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018. 2, 4
- [30] Kai Han, Yunhe Wang, Qiulin Zhang, Wei Zhang, Chunjing Xu, and Tong Zhang. Model rubik’s cube: Twisting resolution, depth and width for tinynets. *Advances in Neural Information Processing Systems*, 33:19353–19364, 2020. 1, 2
- [31] 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, 2
- [32] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News*, 44(3):243–254, 2016. 1
- [33] 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. 1, 7, 13
- [34] 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. 2
- [35] 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. 6, 7
- [36] 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. 1, 2, 6, 7
- [37] 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. 6, 7
- [38] Yang He, Yuhang Ding, Ping Liu, Linchao Zhu, Hanwang Zhang, and Yi Yang. Learning filter pruning criteria for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2009–2018, 2020. 6
- [39] Charles Herrmann, Richard Strong Bowen, and Ramin Zabih. Channel selection using gumbel softmax. In *Computer Vision—ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVII*, pages 241–257. Springer, 2020. 6, 7
- [40] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 1
- [41] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019. 1
- [42] 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, 6
- [43] Feihu Huang, Shangqian Gao, and Heng Huang. Bregman gradient policy optimization. In *International Conference on Learning Representations*, 2022. 3
- [44] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 2
- [45] Qiangui Huang, Kevin Zhou, Suya You, and Ulrich Neumann. Learning to prune filters in convolutional neural networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 709–718. IEEE, 2018. 1, 2
- [46] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015. 13
- [47] Minsoo Kang and Bohyung Han. Operation-aware soft channel pruning using differentiable masks. In *International Conference on Machine Learning*, pages 5122–5131. PMLR, 2020. 6
- [48] Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research*, 75:1401–1476, 2022. 2, 3, 4
- [49] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 6
- [50] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 6, 13
- [51] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *ICLR*, 2017. 1, 2, 4
- [52] Yuchao Li, Shaohui Lin, Jianzhuang Liu, Qixiang Ye, Mengdi Wang, Fei Chao, Fan Yang, Jincheng Ma, Qi Tian, and Rongrong Ji. Towards compact cnns via collaborative compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6438–6447, 2021. 7

- [53] Yawei Li, Kamil Adamczewski, Wen Li, Shuhang Gu, Radu Timofte, and Luc Van Gool. Revisiting random channel pruning for neural network compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 191–201, 2022. 7
- [54] Yunqiang Li, Jan C van Gemert, Torsten Hoefer, Bert Moons, Evangelos Eleftheriou, and Bram-Ernst Verhoef. Differentiable transportation pruning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16957–16967, 2023. 6
- [55] Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. In *International Conference on Learning Representations*, 2020. 7
- [56] 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. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. 1, 2, 4
- [57] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1529–1538, 2020. 2, 4, 6
- [58] Shiwei Liu, Tianlong Chen, Xiaohan Chen, Xuxi Chen, Qiao Xiao, Boqian Wu, Tommi Kärkkäinen, Mykola Pechenizkiy, Decebal Constantin Mocanu, and Zhangyang Wang. More convnets in the 2020s: Scaling up kernels beyond 51x51 using sparsity. In *International Conference on Learning Representations*, 2023. 1
- [59] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3296–3305, 2019. 1, 2, 7
- [60] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11976–11986, 2022. 1
- [61] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017. 13
- [62] 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. 1
- [63] Giosué Cataldo Marinó, Alessandro Petrini, Dario Malchiodi, and Marco Frasca. Deep neural networks compression: A comparative survey and choice recommendations. *Neurocomputing*, 520:152–170, 2023. 2
- [64] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations*, 2017. 2
- [65] 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. 2, 7
- [66] Hanyu Peng, Jiayang Wu, Shifeng Chen, and Junzhou Huang. Collaborative channel pruning for deep networks. In *International Conference on Machine Learning*, pages 5113–5122, 2019. 2
- [67] Rémy Portelas, Cédric Colas, Lilian Weng, Katja Hofmann, and Pierre-Yves Oudeyer. Automatic curriculum learning for deep rl: A short survey. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 4819–4825. International Joint Conferences on Artificial Intelligence Organization, 2020. Survey track. 3
- [68] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016. 1, 2
- [69] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 1
- [70] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015. 1
- [71] 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. 1, 6, 7, 13
- [72] Jürgen Schmidhuber. Powerplay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem. *Frontiers in psychology*, 4: 313, 2013. 3
- [73] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016. 1, 3
- [74] Yang Sui, Miao Yin, Yi Xie, Huy Phan, Saman Aliari Zonouz, and Bo Yuan. Chip: Channel independence-based pruning for compact neural networks. *Advances in Neural Information Processing Systems*, 34:24604–24616, 2021. 2
- [75] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013. 13
- [76] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999. 3
- [77] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019. 1, 2

- [78] Yehui Tang, Yunhe Wang, Yixing Xu, Dacheng Tao, Chun-jing Xu, Chao Xu, and Chang Xu. Scop: Scientific control for reliable neural network pruning. *Advances in Neural Information Processing Systems*, 33:10936–10947, 2020. 6, 7
- [79] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996. 2
- [80] Ahmed Touati and Pascal Vincent. Efficient learning in non-stationary linear markov decision processes. *arXiv preprint arXiv:2010.12870*, 2020. 3
- [81] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *arXiv preprint arXiv:2207.02696*, 2022. 1
- [82] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharmashan Kumaran, and Matt Botvinick. Learning to reinforce learn. *arXiv preprint arXiv:1611.05763*, 2016. 3
- [83] Longguang Wang, Xiaoyu Dong, Yingqian Wang, Li Liu, Wei An, and Yulan Guo. Learnable lookup table for neural network quantization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12423–12433, 2022. 2
- [84] Xu Wang, Sen Wang, Xingxing Liang, Dawei Zhao, Jincan Huang, Xin Xu, Bin Dai, and Qiguang Miao. Deep reinforcement learning: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–15, 2022. 3
- [85] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992. 3
- [86] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pages 2074–2082, 2016. 2
- [87] Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. Neural architecture search: Insights from 1000 papers. *arXiv preprint arXiv:2301.08727*, 2023. 2
- [88] Sanghyun Woo, Shoubhik Debnath, Ronghang Hu, Xinlei Chen, Zhuang Liu, In So Kweon, and Saining Xie. Convnext v2: Co-designing and scaling convnets with masked autoencoders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16133–16142, 2023. 1
- [89] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8817–8826, 2018. 1, 2
- [90] Mao Ye, Chengyue Gong, Lizhen Nie, Denny Zhou, Adam Klivans, and Qiang Liu. Good subnetworks provably exist: Pruning via greedy forward selection. In *International Conference on Machine Learning*, pages 10820–10830. PMLR, 2020. 1, 7
- [91] Miao Yin, Yang Sui, Wanzhao Yang, Xiao Zang, Yu Gong, and Bo Yuan. Hodec: Towards efficient high-order decomposed convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12299–12308, 2022. 2
- [92] Sixing Yu, Arya Mazaheri, and Ali Jannesari. Topology-aware network pruning using multi-stage graph embedding and reinforcement learning. In *International Conference on Machine Learning*, pages 25656–25667. PMLR, 2022. 1, 2, 6, 7
- [93] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33:5824–5836, 2020. 3
- [94] Xin Yu, Thiago Serra, Srikumar Ramalingam, and Shandian Zhe. The combinatorial brain surgeon: Pruning weights that cancel one another in neural networks. In *International Conference on Machine Learning*, pages 25668–25683. PMLR, 2022. 2
- [95] Xuanyang Zhang, Hao Liu, Zhanxing Zhu, and Zenglin Xu. Learning to search efficient densenet with layer-wise pruning. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020. 1, 2
- [96] Yanfu Zhang, Shangqian Gao, and Heng Huang. Exploration and estimation for model compression. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 487–496, 2021. 7
- [97] Zhuangwei Zhuang, Minghui 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. 2, 6
- [98] Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017. 1, 2



## A. Bounding our Agent’s Actions

As mentioned in Section 3.3 of our paper, we calculate the minimum ( $a_{l,min}$ ) and maximum ( $a_{l,max}$ ) feasible pruning rates for the  $l$ -th layer before pruning it to ensure that reaching the desired FLOPs budget,  $FLOPs_{desire}$ , is still possible after doing so. However, before formally introducing our scheme for calculating  $a_{l,min}$ ,  $a_{l,max}$ , we present how we implement our pruning actions in practice.

### A.1. Implementation of our Agent’s Actions

We describe our implementation for our agent’s actions for each architecture. For all models, we take each block of a CNN model as one ‘layer’ in our framework.

**ResNets:** for our experiments on ResNet [33] models (ResNet-56 on CIFAR-10 [50] and ResNet-18/34 on ImageNet [10]), we take each residual block as one layer. It contains a structure as Conv1-BN-ReLU-Conv2-BN where Conv1 and Conv2 are the convolution layers, BN represents Batch Normalization [46], and ReLU is the ReLU activation function. For each block, given the predicted action  $a_l$  for pruning it, we remove  $\lfloor a_l \times c \rfloor$  output channels of the Conv1 layer and the same number of input channels of the Conv2 layer where  $c$  is the number of output/input channels of Conv1/Conv2.

**MobileNet-V2:** for experiments using MobileNet-V2, we take each inverted residual block [71] as one layer for pruning. Each block has the structure with Conv1-BN-ReLU6-DW\_Conv-BN-ReLU6-Conv2-BN form where DW\_Conv is a depth-wise convolution layer. Given the predicted action  $a_l$ , we remove  $\lfloor a_l \times c \rfloor$  output channels of Conv1 and the same amount of channels of DW\_Conv and input channels of Conv2.

In summary, our pruning scheme changes the inner number of channels in each block of a CNN and preserves its number of input and output channels.

### A.2. Calculating Action Bounds

We calculate  $a_{l,min}$ ,  $a_{l,max}$  for the  $l$ -th layer based on the total model’s FLOPs that we denote with  $FLOPs_T$  the number of FLOPs for the previous pruned layers  $FLOPs_{1:l-1}$ ; the number of FLOPs for the next remaining layers  $FLOPs_{l+1:L}$ ;  $FLOPs[l]$ ; and  $FLOPs_{desire}$ . The formulations are as follows:

$$a_{l,min} = 1 - \frac{FLOPs_{desire} - FLOPs_{1:l-1}}{FLOPs[l]} \quad (12)$$

$$a_{l,max} = 1 - \frac{FLOPs_{desire} - FLOPs_{1:l-1} - FLOPs_{l+1:L}}{FLOPs[l]} \quad (13)$$

In these equations,  $a_{l,max}$  prevents very high pruning rates that even if all the next layers are kept intact, reaching  $FLOPs_{desire}$  get infeasible. Similarly,  $a_{l,min}$  provides

the minimum pruning rate for the current layer given all the next layers are pruned completely. We clip the predicted action  $a_l$  to lie in  $[a_{l,min}, a_{l,max}]$  when pruning the  $l$ -th layer.

## B. Experimental Settings

We provide more details of our experimental settings in the following.

**CIFAR-10:** For CIFAR-10 experiments, we evaluate our method on ResNet-56 [33] and MobileNet-V2 [71]. In our iterative pruning phase, we train both of the CNN models for 200 epochs with the batch size of 128 using SGD with momentum [75] of 0.9, weight decay of  $1e-4$ , and starting learning rate of 0.1. We decay the learning rate by 0.1 on epochs 100 and 150. We take 5000 samples of the training dataset as a subset for calculating the agent’s reward. For all cases, we start to train the RL agent after 10 warmup epochs of the model’s weights. Specifically, we collect initial data for the replay buffer of the RL agent from epochs 10 to 20. Then, for both ResNet-56 and MobileNet-V2, we update the agent from epoch 20 until the epoch 90, and we train only the model’s weights from epoch 90 to 200. After the iterative stage, we prune the model’s architecture and finetune it with the same settings for the base model.

**ImageNet:** We use ResNet-18, ResNet-34, and MobileNet-V2 for ImageNet experiments. For the iterative training stage of ResNets, we use SGD as the optimizer with the momentum of 0.9, weight decay of  $1e-4$ , and the start learning rate of 0.1. We train ResNet models for 90 epochs, and we decay the learning rate to 0.01 and 0.001 at epochs 30 and 60. For MobileNet-V2, we do so for 155 epochs with a batch size of 256. We train the model’s weights using SGD with the momentum of 0.9, weight decay of  $1e-4$ , and starting learning rate of 0.05 decayed using cosine scheduling [61]. For all cases, we use 50000 samples of the training dataset to evaluate rewards of the agent. Similar to CIFAR-10 experiments, we train the model’s weights for 10 warmup epochs followed by 10 epochs for filling the replay buffer of the RL agent. Then, for MobileNet-V2, we train the agent’s policy from epochs 20 to 90, and we only train the model’s weights from epoch 90 to 155. After the pruning stage, we fine-tune the pruned model with the same training parameters as the base model. For ResNet models, we train the agent’s policy from epochs 20 to 70, and the model’s weights are trained from epochs 70 to 90.

**Ablation Experiments:** We follow the same settings as mentioned above for our ablation experiments in Tab. 3 of the paper. For the visualizations in Fig. 2, we use the same settings except that we perform our iterative pruning scheme for 90 epochs.