Conditional Automated Channel Pruning for Deep Neural Networks

Yixin Liu*, Yong Guo*, Jiaxin Guo, Luoqian Jiang, Jian Chen†

***Abstract*—Channel pruning has become one of the predominant compression methods to deploy deep models on resource-constrained devices. Most channel pruning methods often use a fixed compression rate for all the layers of the model, which, however, may not be optimal. To address this issue, given a specific target compression rate, one can search for the optimal compression rate for each layer via some automated methods. Nevertheless, when we consider multiple compression rates, these methods have to repeat the channel pruning process multiple times, once for each rate, which can be unnecessary and inefficient. To tackle the problem, we propose a Conditional Automated Channel Pruning (CACP) method which simultaneously produces compressed models under different compression rates through a single channel pruning process. Specifically, CACP takes a set of compression rates and the original model as its input, and outputs the feasible compressed models that satisfy the considered compression rates. To learn CACP, we cast the layer-by-layer channel pruning process into a Markov decision process (MDP), in which we seek to solve a series of decision-making problems. Based on MDP, we develop a reinforcement learning (RL) framework with deep deterministic policy gradient (DDPG) to learn the optimal policy. To satisfy the constraint items in the optimization problem, we design a constraint-guaranteed method, which guides the agent to search for compressed models that satisfy the computational constraints by limiting the action space. Extensive experiments on CIFAR-10 and ImageNet datasets demonstrate the superiority of our method over existing methods.**

***Index Terms*—Channel Pruning, AutoML, Reinforcement learning, Model Compression**

I. Introduction

Deep Neural Networks (DNNs) have achieved great success in many tasks, *e.g.*, image caption [1], [2], image retrieval [3] and image denoising [4]. However, it is hard to apply deep learning methods on resource-constrained devices. To address this, model compression [5]–[7] has become an effective way to reduce redundancy of deep networks. Recently, channel pruning [8]–[10] has been one of the predominant approaches for deep model compression. Specifically, channel pruning aims to remove the redundant channels of the layers in a deep network. To obtain the models with desired compactness, most rule-based channel pruning methods apply a fixed compression rate to all the layers of the deep model [11]–[13]. However, the degree of redundancy normally differs among layers [14], [15]. Thus, pruning the same proportion of channels for all the layers may be sub-optimal. Since the problem of finding the optimal compression rate for each layer can be formalized as a combinatorial optimization problem (COP) [16], [17], some automated search methods [18]–[21] can be applied to obtain the solution, *e.g.*, heuristic search [22], reinforcement learning [14], [23]. In the common scenario of model deployment [24], the devices to be deployed have different resource constraints, and thus compressed models of different sizes are required. In this sense, the pruning problem with multiple compression rates should be considered. However, existing methods [14], [22], [23] only perform channel pruning for a specific compression rate. When considering multiple compression rates, they have to repeat the channel pruning process for each compression rate, which is very unnecessary and inefficient. Therefore, the research on how to simultaneously obtain different compressed models with only a single search is quite important.

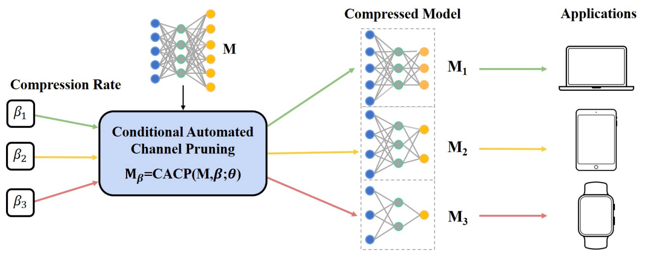

Fig. 1. The overview of what CACP does. Given multiple compression rates as input at once, instead of carrying out the channel pruning separately, our method can automatically conduct the channel pruning and obtain the corresponding compressed networks simultaneously.

In this paper, we propose a novel Conditional Automated Channel Pruning (CACP) method (Illustrated in Fig.1) to automatically conduct optimal pruning. Unlike existing methods, CACP takes several target compression rates as the condition and simultaneously outputs the compressed models satisfying computational constraints.

Our contributions are summarized as follows:

- We propose a Conditional Automated Channel Pruning (CACP) method to simultaneously obtain feasible models under multiple compression rates. Specifically, CACP takes a set of compression rates and the original model as its input, and outputs feasible compressed models that satisfy the considered compression rates.
- We formulate the conditional pruning problem as a constrained optimization problem and cast it into a Markov

Yixin Liu, Yong Guo and Luoqian Jiang are with the School of Software Engineering, South China University of Technology. Jiaxin Guo is with the School of Computer Science and Engineering, South China University of Technology. Jian Chen is also with Guangdong Key Laboratory of BigData Analysis and Processing. E-mail: {seyixinliu, guo.yong, cs_guojiaxin}@mail.scut.edu.cn, lqrosie728@163.com, ellachen@scut.edu.cn

* Authors contributed equally to this work.
† Corresponding author.

decision process (MDP), in which we seek to solve a series of decision-making problems. Based on MDP, we then solve the problem with deep deterministic policy gradient (DDPG). To satisfy the constraint items in the optimization problem, we limit the action space to guide the agent in the searching process.

- Extensive experiments on different network architectures show that our method significantly reduces the pruning time compared with the considered baseline methods. To be more specific, given three compression rates as targets on ResNet-50, our method is 3× to 800× faster than the considered baseline methods.

II. Conditional Automated Channel Pruning

In this paper, we propose a Conditional Automated Channel Pruning (CACP) method to simultaneously conduct automated pruning under multiple compression rates. To this end, we formulate the problem as a constrained optimization problem and cast it into a Markov decision process (MDP). Based on the MDP, we develop an RL framework with DDPG to learn the optimal policy. To satisfy the constraint items, we design a constraint-guaranteed method to guide agent's searching by limiting the action space.

A. Problem Definition

Given a T-layer CNN model M and its filter set W, we denote the channel structure of M by $C=(c_1,c_2,...,c_T)$, where c_k is the channel number of k-th layer. Given a specific compression rate β, channel pruning methods aim to search for the best channel pruning strategy, namely $C'=(c'_1,c'_2,...,c'_T)$. With the optimal pruning strategy C', the compressed network M'_β can achieve the best validation accuracy. We seek to obtain compressed models with different compression rates β through only a single search. To achieve this goal, we aim to learn a good pruner $M'_\beta=\text{CACP}\left(M,\beta;\theta\right)$ parameterized by θ, where we assume β follows some distribution $p(\cdot)$, *e.g.*, discrete uniform distribution. Let $Acc(M'_\beta)$ denote the validation accuracy of the pruned model M'_β. Following [25], we can instead learn a sampling policy $\pi_\theta\left(\cdot\right)$ that maximizes the expectation of performance metric $Acc(M'_\beta)$ over the distribution of β and M'_β under the constraint of computation cost $c(M'_\beta)\le\beta c(M)$, where $c(M)$ measures the computational cost (*e.g.*, floating-point operations (FLOPs)) of network M. Then, the optimization problem can be written as

$$\begin{aligned}&\max_\theta \mathbb{E}_{\beta\sim p(\cdot)}\left[\mathbb{E}_{M'_\beta\sim\pi_\theta(M,\beta)}\left[Acc\left(M'_\beta\right)\right]\right]\\ &\text{s.t.}\quad c\left(M'_\beta\right)\le\beta c\left(M\right)\end{aligned}. \tag{1}$$

To construct the policy function $\pi_\theta\left(\cdot\right)$, we formulate the pruning process as a Markov decision process (MDP), which we seek to decide the compression rate of each layer of M to map the original network M to the compressed network M'_β.

B. Markov Decision Process for CACP

A typical MDP [26] can be defined by a tuple $(\mathcal{S},\mathcal{A},P,r,p_0,\gamma)$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $P:\mathcal{S}\times\mathcal{A}\times\mathcal{S}\to\mathbb{R}$ is the state transition distribution, $r:\mathcal{S}\times\mathcal{A}\to\mathbb{R}$ is the reward function, $p_0:\mathcal{S}\to[0,1]$ is the distribution of initial state, and $\gamma\in[0,1]$ is a discount factor. Here, we define the compression rate a as action for each layer. Specifically, the action $a_t\in(0,1]$ for the t-th layer L_t results in $c'_t=\lceil a_t c_t\rceil$ channels being reserved. The discount factor γ is set to 1 following [14].

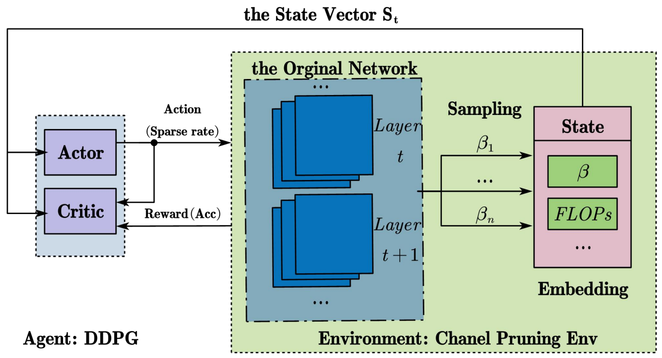

Fig. 2. The overview of how CACP works. We cast the channel pruning process as a Markov decision process (MDP), in which we seek to solve a series of decision-making problems. We define 12 features of each layer including FLOPs, compression target β *etc*. The action is the compression rate of each layer. And the final validation accuracy of the pruned network is taken as a variable of the reward function. Then we use the deep deterministic policy gradient (DDPG) algorithm to train an agent to interact with the pruning environment. By sampling different compression targets β, the agent learns to conduct the optimal pruning for different compression targets.

State space. For each layer L_t, we use 12 features to characterize its state s_t. 9 of them are static features, including FLOPs (floating point operations), the layer index, *etc*. The other three are dynamic features (*i.e.*, dynamically change according to the pruning operation of the previous layer generated by agent), including the action taken in layer L_{t-1} and the total reduced FLOPs in previous layers. A complete description of state features is given in the supplement.

Reward. Let $r_\text{t}=r\left(s_\text{t},a_\text{t}\right)$ be the intermediate reward observed for action a_t under state s_t and s_T be the final state (*i.e.*, the last layer). Here, r_T is the validation accuracy of the pruned network before fine-tuning. Intuitively, we cannot receive any reward signals except for the final layer since we can only perform network pruning after all actions are done. Ignoring these rewards in the iterative process [27], *i.e.*. setting them to zero, may cause the sparse reward problem [28] which leads to slow convergence in the beginning of learning. Following Block-QNN [29], we instead use the reshaped reward to promote the learning process, which is defined as:

$$r_\text{t}=\frac{r_\text{T}}{T}. \tag{2}$$

C. Policy Learning with DDPG

The sum of discounted future rewards is called the *return* from a state s_t and is defined as $G_\text{t}=\sum_{i=t}^{T}\gamma^{\text{i}-\text{t}}r\left(s_\text{i},a_\text{i}\right)$. Based on MDP, the optimization objective of Equation 1 can be transformed into the objective of the agent, which is to learn a policy π_θ that maximizes the expected return from the start distribution $J\left(\pi_\theta\right)=\mathbb{E}_{r_i,s_i\sim E,a_i\sim\pi_\theta}\left[G_1\right]$, where $r_i,s_i\sim E$ means that r_i,s_i are drawn from environment E. We define the discounted state visitation distribution

Algorithm 1 Training method for CACP.

```
Input: Number of total iterations $T$, the original network $M$, learning rate $\eta$, the compression rate distribution $p(\cdot)$
Output: The best pruned network set $M_{\mathrm{B}}^{*}$
 1: Randomly initialize the current network $Q$ and $\mu$
 2: Copy $Q$ & $\mu$ to create target network $Q'$ & $\mu'$
 3: for $T_i$=1 $\rightarrow T$ do
 4:     Sample $\beta \sim p(\cdot)$
 5:     for $t$=1 $\rightarrow L$ do
 6:         $a_t \leftarrow \mu(s_t;\theta^{\mu}) + \mathcal{N}$;
 7:         $s_{t+1} \leftarrow \text{pruneLayer}\left(L_t, a_t, M\right)$;
 8:         $\mathcal{R} \leftarrow \text{storeBuffer}\left(s_t, a_t, r_t, s_{t+1}\right)$;
 9:     end for
10:     Sample a random mini-batch $m$ from buffer
11:     // Update the critic network $Q$ by descending the gradient
12:     $\theta^{Q} \leftarrow \theta^{Q} - \eta\frac{1}{m}\sum_{i=1}^{m}\nabla_{\theta^{Q}}\left(y_t - Q\left(s_t, a_t;\theta^{Q}\right)\right)^2$;
13:     // Update the actor network $\mu$ by ascending the gradient
14:     $\theta^{\mu} \leftarrow \theta^{\mu} + \eta\frac{1}{m}\sum_{i=1}^{m}\nabla_{\mathrm{a}}Q\left(s,a\right)|_{\mathrm{a}=\mu(s_t)}\nabla_{\theta^{\mu}}\mu\left(s\right)|_{\mathrm{s}=\mathrm{s_t}}$;
15:     // Update the target networks by using soft weights copying
16:     $\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$; $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$;
17: end for
18: for $i$=1 $\rightarrow n$ do
19:     Sample $\beta_{\mathrm{i}} \sim p\left(\cdot\right)$, $M'_{\beta_{\mathrm{i}}} \sim \text{CACP}\left(M, \beta_{\mathrm{i}};\theta\right)$;
20:     $M^{*}_{\beta_{\mathrm{i}}} \leftarrow \text{finetunes}\left(M'_{\beta_{\mathrm{i}}}\right)$;
21: end for
22: $M_{\mathrm{B}}^{*} = \left\{M^{*}_{\beta_1}, M^{*}_{\beta_2}, \cdots, M^{*}_{\beta_{\mathrm{n}}}\right\}$, where $\beta_{\mathrm{i}} \sim p\left(\cdot\right)$.
```

as $\rho^{\pi_\theta}\left(s\right) := \int_{\mathcal{S}}\sum_{t=1}^{\infty}\gamma^{t-1}p_0(s')p\left(s' \rightarrow s, t, \pi_\theta\right)\mathrm{d}s$, where $p\left(s' \rightarrow s, t, \pi_\theta\right)$ is the probability at state s after transitioning for t time steps from state s' following policy π_θ. Then, the goal of agent can be rewritten as the expectation over all the state and action pairs [30]:

$$\begin{aligned} J\left(\pi_\theta\right) &= \int_{\mathcal{S}}\rho^{\pi_\theta}\left(s\right)\int_{\mathcal{A}}\pi_\theta(s,a)r(s,a)\mathrm{d}a\mathrm{d}s \\ &= \mathbb{E}_{s_t\sim\rho^{\pi_\theta}, a_t\sim\pi_\theta}[r(s_t,a_t)] \end{aligned}. \tag{3}$$

We denote the action-value function as $Q^{\pi_\theta}\left(s_{\mathrm{t}}, a_{\mathrm{t}}\right) = \mathbb{E}_{r_{i\geqslant t}, s_{i\geqslant t}\sim E, a_{i\geqslant t}\sim\pi_\theta}\left[G_{\mathrm{t}}|s_{\mathrm{t}}, a_{\mathrm{t}}\right]$, which describes the expected return after taking an action a_t in state s_t and following policy π_θ thereafter. Then the gradient of the objective function $J\left(\pi_\theta\right)$ can be calculated by the *policy gradient theorem* [31]:

$$\begin{aligned} \nabla_\theta J\left(\pi_\theta\right) &= \int_{\mathcal{S}}\rho^{\pi_\theta}(s)\int_{\mathcal{A}}\nabla_\theta\pi_\theta(s,a)Q^{\pi_\theta}(s,a)\mathrm{d}a\mathrm{d}s \\ &= \mathbb{E}_{s_t\sim\rho^{\pi_\theta}, a_t\sim\pi_\theta}\left[\nabla_\theta\log\pi_\theta(s_t,a_t)Q^{\pi_\theta}(s_t,a_t)\right] \end{aligned}. \tag{4}$$

To calculate the above gradient, one key issue we must address is how to estimate the action value function $Q^{\pi_\theta}\left(s,a\right)$. One simple approach is to use the return G_t to estimate the value of $Q^{\pi_\theta}\left(s,a\right)$, which leads to a variant of the REINFORCE algorithm [31]. However, such a simple approach suffers from the problem of high variance in the estimates [32]. To address this issue, the *actor-critic* algorithms [32] bring in a function approximator named *critic*. The critic $Q^{\pi_\theta}(s,a;\theta^{Q})$ parameterized by θ^{Q} aims to estimate the *true* action-value function $Q^{\pi_\theta}\left(s,a\right)$, namely $Q^{\pi_\theta}(s,a;\theta^{Q}) \approx Q^{\pi_\theta}\left(s,a\right)$.

Deep Neural Network (DNN) is a powerful function approximator for modeling the critic [33]. However, learning with such a highly non-linear function approximator is known to be unstable [32]. To address this problem, two effective techniques from DQN can be used: experience relay and target network. However, DQN is based on the Q-Learning algorithm which has difficulty in solving the problem with continuous action space [33] while the compression action at each layer in our problem is exactly continuous. DPG [30] introduces a deterministic actor $\mu : \mathcal{S} \rightarrow \mathcal{A}$ that directly maps a state to a specific action to solve the problem of continuous action space. However, DPG models the critic with a simple approximator, which lacks of representation ability when dealing with more difficult problems [33]. In this paper, we intend to introduce an actor-critic algorithm named *Deep Deterministic Policy Gradient* (DDPG) [33], which combines the advantages of both DQN [34] and DPG [30].

The *temporal-difference* (TD) learning [35] is used as the policy evaluation algorithm [36] to learn the critic. To present its idea, we first introduce the recursive form of the *true* action value function $Q^{\mu}(s,a)$ with a deterministic policy $\mu\left(s;\theta^{\mu}\right)$ as follows, which is also known as the Bellman Equation [37]:

$$Q^{\mu}\left(s_t, a_t\right) = \mathbb{E}_{r_t, s_{t+1}\sim E}\left[r\left(s_t, a_t\right) + \gamma Q^{\mu}\left(s_{t+1}, \mu\left(s_t;\theta^{\mu}\right)\right)\right]. \tag{5}$$

We use a critic network $Q^{\mu}\left(s,a;\theta^{Q}\right)$ to approximate the *true* action value function $Q^{\mu}\left(s,a\right)$. Then, we define the the left side of the Equation 5 above as *the current term* $Q^{\mu}\left(s_t, a_t;\theta^{Q}\right)$, the expectation on the right side as *the target term* y_t and their difference as the *TD error* δ_t:

$$\begin{aligned} \delta_t &= y_t - Q^{\mu}\left(s_t, a_t;\theta^{Q}\right), \\ \text{where}\quad y_t &= r_t + \gamma Q^{\mu}\left(s_{t+1}, \mu\left(s_t;\theta^{\mu}\right);\theta^{Q}\right). \end{aligned} \tag{6}$$

The core idea of TD learning is to minimize the TD error. The critic $Q^{\mu}\left(s,a;\theta^{Q}\right)$ parameterized by θ^{Q} is optimized by minimizing the following loss function:

$$\mathcal{L}\left(\theta^{Q}\right) = \mathbb{E}_{s_t\sim E, a_t\sim\mu}\left[\delta_t^2\left(\theta^{Q}\right)\right]. \tag{7}$$

With the feedback from critic, the deterministic actor $\mu\left(s;\theta^{\mu}\right)$ adjusts its parameters θ^{μ} by stochastic gradient ascent of Equation 4 to maximize its expected evaluation:

$$\mathcal{L}\left(\theta^{\mu}\right) = \mathbb{E}_{s_t\sim E}\left[Q^{\mu}\left(s_t, \mu\left(s_t;\theta^{\mu}\right);\theta^{Q}\right)\right]. \tag{8}$$

Two effective tricks are also used to enhance the learning stability in DDPG. To address the problem that the target term y_t computed by the network $Q^{\mu}\left(\cdot\right)$ and $\mu\left(\cdot\right)$ in Equation 6 changes greatly during training [34], two target networks μ' and Q', whose parameters $\theta^{\mu'}$ and $\theta^{Q'}$ respectively track θ^{μ} and θ^{Q} using exponential smoothing, are used instead to compute the target term y_t. In addition, a finite-size cache $\mathcal{R}$ called *replay buffer* is used to break the correlation between sequentially generated samples and thus agent can learn from more independently distributed past experiences. The training process is summarized in Algorithm 1.

D. Constraint-Guaranteed Search Strategy

Since the reward designed in Section II-B offers no incentive for model size reduction, to satisfy the constraints of computing cost in Equation 1, we limit the action space to guide the agent in the searching process. To this end, we intend to limit the pruning action when we find that the resultant model cannot reach the target compression rate even though we adopt the most aggressive pruning strategy (*e.g.*, remove nearly all channels of a layer) in the following layers. Specifically, we formalize this intuition as follows.

Let C_t be the total computational cost of the current layer L_t, and $D_t^{(\beta)}$ be the lower bound of the computational cost that should be reduced for the t-th layer to achieve the compression rate β. To ensure that we can obtain the model satisfying the overall compression rate, the compression rate $\alpha_t^{(\beta)}$ for the t-th layer becomes

$$\alpha_t^{(\beta)} = \max\left(\mu\left(s_{\mathrm{t}}; \theta^{\mu}\right), D_t^{(\beta)}/C_t\right). \tag{9}$$

We compute $D_t^{(\beta)}$ in Equation 9 based on the "duty" of Layer L_t in reducing computational cost. Let $\alpha_{\max}$ be the maximum possible compression rate for the following layers of Layer L_t, C_{all} be the overall cost of the whole model, C_{reduced} be the total amount of reduced cost in the previous layers, and C_{rest} be the amount of the remaining cost in the following layers. Thus, the minimum computation cost that should be reduced for layer L_t becomes

$$D_t^{(\beta)} = \beta \cdot C_{\text{all}} - \alpha_{\max} \cdot C_{\text{rest}} - C_{\text{reduced}}. \tag{10}$$

Based on Eqns. (9) and (10), we can guarantee that the compressed model would satisfy the target compression rate.

III. Experiments

In the experiments, we consider following network architectures for channel pruning, including VGGNet [38] and ResNet-18/34 [39] on CIFAR-10, and ResNet-50/101/152 [39] on ILSVRC-2012. The source code of CACP can be found at https://github.com/liuyixin-louis/CACPpruner.

A. Quantitative Results

To demonstrate the effectiveness of CACP, we applied CACP on the two benchmark datasets. On CIFAR-10, we chose three network architecture, *i.e.*, VGG-16, ResNet-18 and ResNet-34. From Table A, we pruned these networks with nearly no loss of accuracy under three different compression targets. Moreover, the pruned network outperforms the origin one in some cases (*e.g.*, 70% for VGG-16), which shows that the pruned model is less redundant and thus can prevent overfitting. Furthermore, on ILSVRC-2012, we apply CACP on three deeper ResNet's and obtain competitive compressed networks, with only 1% to 4% loss of Top-1 accuracy (See the supplement). To demonstrate the efficiency of CACP, we first conducted an ablation experiment. By setting the number of compression rates to 1 (*i.e.*, $|B| = 1$), which indicates that the model only deals with a specific compression rate at a time, the acceleration effect brought by our designed module can be clearly shown in Table B. Note that the reinforcement learning engine is not necessary to be DDPG. By discretizing the action space, we are able to investigate the effect of different reinforcement learning algorithms on the performance of our method, including Policy Gradient (PG) and DQN (See Table B). Moreover, we compared our method with several state-of-the-art automated channel pruning methods, including ThiNet [40], AMC [14] and ABCpruner [22]. The results are shown in the Table B. Our method can search under multiple compression rates simultaneously, which greatly improves the superiority of our method in search time. With compression target $\beta \in \{30\%, 50\%, 70\%\}$, CACP yields equally well-performed compressed networks, while the total searching time of CACP is 3× less than AMC [14], 50× less than ABCpruner [22], and 800× less than ThiNet [40]. Note that our method can be easily combined with other acceleration techniques to achieve higher acceleration ratios. For example, by leveraging the recent one-stage technique [41]–[44], which combines the process of searching architecture and repairing weight, the pruning process can be further accelerated.

TABLE A
COMPARISONS OF THE COMPRESSED MODELS(VGG-16, RESNET-18/34) ON CIFAR-10. WE COUNT THE TOP1 ACCURACY, FLOPS AND PARAMS FOR EACH PRUNED MODEL.

Model	Top1-acc(%)	#FLOPs(M)	#Params(M)
VGG-16 Baseline	93.02	314.59	14.73
VGG-16 CACP-30%	92.89	94.37	4.41
VGG-16 CACP-50%	93.08	157.29	7.36
VGG-16 CACP-70%	**93.53**	220.21	10.31
ResNet-18 Baseline	93.02	37.62	11.68
ResNet-18 CACP-30%	92.03	11.28	3.50
ResNet-18 CACP-50%	92.28	19.81	5.84
ResNet-18 CACP-70%	**93.54**	26.33	8.18
ResNet-34 Baseline	93.58	75.42	21.79
ResNet-34 CACP-30%	93.42	22.62	6.63
ResNet-34 CACP-50%	93.28	37.71	10.89
ResNet-34 CACP-70%	**93.72**	52.79	15.25

TABLE B
COMPARISON OF SEARCH TIME AND NETWORK PERFORMANCE WITH DIFFERENT METHODS FOR THE RESNET-50 ON IMAGENET. THE COMPRESSION TARGETS ARE 30%, 50% AND 70%.

Method	Top1-acc(%)	#FLOPs(G)	Total Search Time(min)
Baseline	76.01	4.11	-
ThiNet [40]	68.02/71.01/72.07	1.02/2.19/2.86	52,920
AMC [14]	72.04/72.33/74.08	1.17/2.02/2.75	172
ABCpruner [22]	72.58/73.47/75.77	1.2/2.02/2.99	3,240
CACP ($|B| = 1$)	72.88/73.73/74.28	1.18/2.01/2.73	167
CACP-DQN	72.78/73.75/74.89	1.25/1.97/2.83	52
CACP-PG	72.08/72.34/74.10	1.24/2.01/2.85	50
CACP-Ours	**72.98/73.84/75.88**	**1.22/1.93/2.80**	**54**

IV. Conclusion

In this paper, we have proposed a novel Conditional Automated Channel Pruning (CACP) method to obtain the compressed models under different compression rates through a single channel pruning process. To this end, we cast the problem into a Markov decision process (MDP) and develop a reinforcement learning framework to learn a policy. To show the effectiveness and efficiency of our method, we applied it to two famous network architectures (VGG and ResNet) and compared with several SOTA methods in the pruning time. Extensive experiments on CIFAR-10 and ImageNet datasets demonstrate the superiority of the proposed method.

Acknowledgement

This work is partially supported by the National Natural Science Foundation of China (62072186), the Guangdong Basic and Applied Basic Research Foundation (2019B1515130001), the Guangzhou Science and Technology Planning Project (201904010197).

References

[1] C. Yan, Y. Hao, L. Li, J. Yin, A. Liu, Z. Mao, Z. Chen, and X. Gao, "Task-adaptive attention for image captioning," *IEEE Transactions on Circuits and Systems for Video Technology*, 2021.

[2] C. Yan, B. Shao, H. Zhao, R. Ning, Y. Zhang, and F. Xu, "3d room layout estimation from a single rgb image," *IEEE Transactions on Multimedia*, vol. 22, no. 11, pp. 3014–3024, 2020.

[3] C. Yan, B. Gong, Y. Wei, and Y. Gao, "Deep multi-view enhancement hashing for image retrieval," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[4] C. Yan, Z. Li, Y. Zhang, Y. Liu, X. Ji, and Y. Zhang, "Depth image denoising using nuclear norm and learning graph model," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 16, no. 4, pp. 1–17, 2020.

[5] C. Buciluă, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006, pp. 535–541.

[6] R. Venkatesan, G. Swaminathan, X. Zhou, and A. Luo, "Out-of-the-box channel pruned networks," *CoRR*, vol. abs/2004.14584, 2020.

[7] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, "Netadapt: Platform-aware neural network adaptation for mobile applications," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 285–300.

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

[9] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis, "Nisp: Pruning networks using neuron importance score propagation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9194–9203.

[10] D. Yi, Z. Lei, S. Liao, and S. Z. Li, "Learning face representation from scratch," *CoRR*, vol. abs/1411.7923, 2014.

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

[12] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu, "Discrimination-aware channel pruning for deep neural networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 875–886.

[13] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *CVPR*, 2019, pp. 4340–4349.

[14] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," in *ECCV*, 2018.

[15] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[16] W. Cook, L. Lovász, P. D. Seymour *et al.*, *Combinatorial optimization: papers from the DIMACS Special Year*. American Mathematical Soc., 1995, vol. 20.

[17] B. H. Korte, J. Vygen, B. Korte, and J. Vygen, *Combinatorial optimization*. Springer, 2011, vol. 1.

[18] Z. Wang, C. Li, and X. Wang, "Convolutional neural network pruning with structural redundancy reduction," *arXiv preprint arXiv:2104.03438*, 2021.

[19] T. Bäck and H.-P. Schwefel, "An overview of evolutionary algorithms for parameter optimization," *Evolutionary computation*, vol. 1, no. 1, pp. 1–23, 1993.

[20] T. E. Stuart and J. M. Podolny, "Local search and the evolution of technological capabilities," *Strategic management journal*, vol. 17, no. S1, pp. 21–38, 1996.

[21] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[22] M. Lin, R. Ji, Y. Zhang, B. Zhang, Y. Wu, and Y. Tian, "Channel pruning via automatic structure search," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, C. Bessiere, Ed. ijcai.org, 2020, pp. 673–679.

[23] G. Ding, S. Zhang, Z. Jia, J. Zhong, and J. Han, "Where to prune: Using lstm to guide data-dependent soft pruning," *IEEE Transactions on Image Processing*, vol. 30, pp. 293–304, 2020.

[24] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," in *International Conference on Learning Representations*, 2020.

[25] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *International Conference on Learning Representations*, 2017.

[26] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

[27] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[28] M. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. Wiele, V. Mnih, N. Heess, and J. T. Springenberg, "Learning by playing solving sparse reward tasks from scratch," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4344–4353.

[29] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2423–2432.

[30] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International conference on machine learning*. PMLR, 2014, pp. 387–395.

[31] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour *et al.*, "Policy gradient methods for reinforcement learning with function approximation." in *NIPs*, vol. 99. Citeseer, 1999, pp. 1057–1063.

[32] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*. Citeseer, 2000, pp. 1008–1014.

[33] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. 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*, Y. Bengio and Y. LeCun, Eds., 2016.

[34] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[35] G. Tesauro, "Practical issues in temporal difference learning," *Machine learning*, vol. 8, no. 3, pp. 257–277, 1992.

[36] E. A. Hansen, "An improved policy iteration algorithm for partially observable mdps." *Advances in neural information processing systems*, pp. 1015–1021, 1998.

[37] R. Bellman, "Dynamic programming and lagrange multipliers," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 42, no. 10, p. 767, 1956.

[38] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[39] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[40] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5058–5066.

[41] G. Tian, J. Chen, X. Zeng, and Y. Liu, "Pruning by training: A novel deep neural network compression framework for image processing," *IEEE Signal Processing Letters*, vol. 28, pp. 344–348, 2021.

[42] X. Ding, G. Ding, Y. Guo, and J. Han, "Centripetal sgd for pruning very deep convolutional networks with complicated structure," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4943–4953.

[43] Q. Li, C. Li, and H. Chen, "Filter pruning via probabilistic model-based optimization for accelerating deep convolutional neural networks," in *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, 2021, pp. 653–661.

[44] X. Ding, G. Ding, X. Zhou, Y. Guo, J. Han, and J. Liu, "Global sparse momentum sgd for pruning very deep neural networks," *arXiv preprint arXiv:1909.12778*, 2019.

[45] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic algorithms and applications," *CoRR*, vol. abs/1812.05905, 2018.

[46] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *ICCV*, 2017.

[47] V. Steinbiss, B.-H. Tran, and H. Ney, "Improvements in beam search," in *Third International Conference on Spoken Language Processing*, 1994.

[48] S. J. Miller, "The method of least squares," *Mathematics Department Brown University*, vol. 8, pp. 1–7, 2006.

Supplementary Materials for "Conditional Automated Channel Pruning for Deep Neural Networks"

We organize our supplementary materials as follows. Firstly, we will present the results of compressed model performance on ImageNet and the design details of our state features in Section A. Secondly, we will describe the implementation details about our method in Section B.

A. Performance Results and Details of State Features

TABLE C
COMPARISONS OF THE COMPRESSED MODELS(RESNET-50/101/152) ON ILSVRC-2012. WE COUNT THE TOP1 ACCURACY, FLOPS AND PARAMS FOR EACH PRUNED MODEL.

Model	Top1-acc(%)	#FLOPs(G)	#Params(M)
ResNet-50 Baseline	76.01	4.11	25.55
ResNet-50 CACP-30%	72.55	1.23	7.66
ResNet-50 CACP-50%	73.84	2.05	12.77
ResNet-50 CACP-70%	**75.88**	2.87	17.88
ResNet-101 Baseline	77.38	7.88	44.54
ResNet-101 CACP-30%	73.37	2.35	13.36
ResNet-101 CACP-50%	75.88	3.91	22.27
ResNet-101 CACP-70%	**76.77**	5.48	31.18
ResNet-152 Baseline	78.21	11.55	60.19
ResNet-152 CACP-30%	74.45	3,46	18.03
ResNet-152 CACP-50%	76.55	5.77	30.09
ResNet-152 CACP-70%	**77.87**	8.09	42.13

TABLE D
THE STATE FEATURES OF LAYER L_t

Type	Symbol	Description
Static features	t	the layer index
	n, c, k	the dimension of the kernel is $n \times c \times k \times k$
	w, h	the shape of input channel is $c \times w \times h$
	stride	the stride in the convolution operation
	FLOPs $[t]$	the FLOPs of Layer L_t
	β	the compression rate that the entire pruned network needs to meet
Dynamic feature	Reduced $[t]$	the total number of reduced FLOPs in the previous layers
	Rest $[t]$	the nunber of remaining FLOPs in the following layers
	a_{t-1}	the action in the layer L_{t-1}

B. Implementation Details

Agent Details. We use the actor-critic [45] scheme as the reinforcement learning engine to train our CACP model. We build the model using a 3-layer neural actor network. Our model receives an embedding state S_l of Layer l as the inputs. We use three fully connected layers, each followed by a ReLU layer to extract the input layer state. Then, we exploit a Sigmoid layer to map the output to compression rate $\alpha_l \in (0, 1)$.

Training Details. We conducted experiments on a single GPU of the GeForce GTX 1650. During the training phase, the first 300 epochs are used for random exploration to fill the experience replay buffer. Then, the next 500 epochs are used for learning. In each epoch, we first repair the weight of pruned networks using the FM-Reconstruction [46] and then estimate its potential goodness without fine-tuning. During the inference phase, we add small noise to the well-learned deterministic policy and conduct sampling in the neighborhood using Beam Search [47].

FM-Reconstruction. Following [14], instead of fine-tuning the pruned network, we use the FM-Reconstruction method [46] to repair its weight, which greatly speeds up the searching process. For VGG-16, we chose CNN layers at regular intervals (from the second layer on) as the compressed layer. For each layer, given its compression action a_{t}, we use the Least Squares method [48] to fit W to reconstruct the output feature maps Y, *i.e.*, solve the following optimization problem for $\underset{\mathrm{W}'}{\arg\min} \left\| \mathrm{Y} - \mathrm{X}' \left(\mathrm{W}'\right)^{\top} \right\|_F^2$, where X' is the data sampled from the original input feature maps X. For ResNets with residual branch structures, we selected the 3×3 CNN layer inside each residual block as the compressed layer. Let the original output feature maps from the residual branch be Y_1, the feature maps passed through shortcut be Y_2, and the current feature maps after the previous layers are pruned be Y_1', and we fit the weight W by reconstructing $\mathrm{Y} = \mathrm{Y}_1 - \mathrm{Y}_1' + \mathrm{Y}_2$. More details can be found in [46].