ABCP: Automatic Block-wise and Channel-wise Network Pruning via Joint Search

Jiaqi Li, Haoran Li, *Member, IEEE*, Yaran Chen, *Member, IEEE*, Zixiang Ding, *Graduate Student Member, IEEE*, Nannan Li, Mingjun Ma, Zicheng Duan, and Dongbing Zhao, *Fellow, IEEE*

Abstract—**Currently, an increasing number of model pruning methods are proposed to resolve the contradictions between the computer powers required by the deep learning models and the resource-constrained devices. However, most of the traditional rule-based network pruning methods can not reach a sufficient compression ratio with low accuracy loss and are time-consuming as well as laborious. In this paper, we propose Automatic Block-wise and Channel-wise Network Pruning (ABCP[1]) to jointly search the block-wise and channel-wise pruning action with deep reinforcement learning. A joint sample algorithm is proposed to simultaneously generate the pruning choice of each residual block and the channel pruning ratio of each convolutional layer from the discrete and continuous search space respectively. The best pruning action taking both the accuracy and the complexity of the model into account is obtained finally. Compared with the traditional rule-based pruning method, this pipeline saves human labor and achieves a higher compression ratio with lower accuracy loss. Tested on the mobile robot detection dataset, the pruned YOLOv3 model saves 99.5% FLOPs, reduces 99.5% parameters, and achieves $37.3\times$ speed up with only 2.8% mAP loss. The results of the transfer task on the sim2real detection dataset also show that our pruned model has much better robustness performance.**

Index Terms—**Joint search, Pruning, Reinforcement learning, Model compression**

I. INTRODUCTION

IN recent years, the deep learning methods are widely applied to machine learning tasks such as speech recognition [1], image processing [2], and structured output [3]. However, the large computational costs of the deep learning models are unaffordable for many resource-constrained devices and make the inference very slow.

Jiaqi Li is with the State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences, Beijing, 100190, China, and also with the School of Mechanical Engineering, Beijing Institute of Technology, Beijing, 100081, China (email: xuer0324@gmail.com).

Haoran Li, Yaran Chen, Zixiang Ding, Nannan Li, Mingjun Ma and Dongbin Zhao are with the State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences, Beijing, 100190, China, and also with the University of Chinese Academy of Sciences, Beijing, China (email: lihaoran2015@ia.ac.cn, chenyaran2013@ia.ac.cn, dingzixiang2018@ia.ac.cn, linannan2017@ia.ac.cn, mamingjun2020@ia.ac.cn, dongbin.zhao@ia.ac.cn).

Zicheng Duan is with the College of Engineering and Computer Science, Australian National University, ACT, 2601, Australia (email: zicheng.duan@anu.edu.au).

[1]Our code will be released at https://github.com/DRL-CASIA/ABCP.

To tackle these problems, a large number of approaches have been proposed [4] [5]. Network pruning is a typical rule-based model compression method to reduce the redundant weights or structures in the network. There are two main types of network pruning: the non-structured pruning [6] and the structured pruning [7]. Since the structured pruning methods utilize structures as the pruning units, such as channels [8], blocks [9], groups [9] as well as both channels and blocks [10], so that the structured pruning is much more hardware-friendly than the non-structured pruning. Hence, most of the researchers tend to pay attention to structured model pruning currently, which is also the focus of this paper.

Nevertheless, the existing structured pruning methods still have some problems. Firstly, fine-tuning the hyperparameters like the pruning threshold increases the workload and it is hard to prove which threshold is optimal [8] [9]. Secondly, the traditional rule-based pruning method always cannot reach a sufficient compression ratio with low accuracy loss. Finally, the iterative pruning has been recommended [10][11], making the "sparse training – pruning – fine-tuning" pipeline be processed several times, which is time-consuming.

At present, the Neural Architecture Search (NAS) methods which can automatically search the network structure have gained ground [12] [13]. In this way, the networks with promising performance can be obtained efficiently with little human labor. To this end, we attempt to combine NAS and network pruning to achieve an automated process for the model pruning tasks. Some recent works have introduced the NAS methods into the model pruning procedure. Since AMC [14] utilized the deep reinforcement learning (DRL) to search each pruning ratio of each convolutional layer, several researchers realize the automated network pruning by DRL [15] or the evolutionary computation [16].

However, most of these works [14] only prune the channels and even can not prune the channels of the layers which belong to the residual blocks, making a very limited compression. The depth reduction of the network (e.g. residual block pruning) is also required to achieve high model pruning rates. Meanwhile, the discrete search space in some works [17] also results in the confined compression ratio. In addition, the layers in the model are so sensitive to be pruned by different ratios that it is more suitable to utilize continuous search space.

Therefore, we propose Automatic Block-wise and Channel-wise Network Pruning (ABCP) to jointly search the channel-wise and block-wise pruning action by DRL. The action of DRL is a list consisting of the pruning choice of each residual block and the channel pruning ratio of each convolutional

layer. The reward for DRL takes both the accuracy and complexity of the pruned model into account. Specifically, we propose a joint sample algorithm to generate the block-wise and channel-wise pruning action. We combine the discrete space and continuous space to sample the block pruning choice and the layer pruning ratio respectively. In addition, we also offer another choice to use the discrete space for sampling the layer pruning ratio. The joint sample algorithm is trained by the policy gradient method [18].

Extensive experiments suggest that ABCP has a very promising performance. We collect three datasets, two of them are proposed by ourselves for the fast and lightweights detection algorithm for the mobile robots, and the third is captured from the videos collected by University of California San Diego (UCSD). Based on these three datasets, we evaluate ABCP on YOLOv3 [19]. Results show that our method achieves better accuracy than the traditional rule-based pruning method with fewer floating point operations (FLOPs). Furthermore, the results also demonstrate that the pruned model via ABCP has much better robustness performance.

To summarize, our contributions are listed as the following:

- We propose ABCP, a pipeline to jointly search the block-wise and channel-wise network pruning action by DRL.
- We propose a joint sample algorithm to jointly sample the pruning choice of each residual block and the channel pruning ratio of each convolutional layer from the discrete and continuous space respectively.
- We test ABCP on YOLOv3 [19] based on three datasets. The results show that ABCP outperforms the traditional rule-based pruning methods. On the mobile robot detection dataset, the pruned model saves 99.5% FLOPs, reduces 99.5% parameters, and achieves $37.3\times$ speed up with only 2.8% mAP loss. On the sim2real detection dataset, the pruned model has better robustness performance, achieving 9.6% better mAP in the transfer task.

II. RELATED WORKS

A. Network Pruning

Network pruning can be divided into non-structured pruning and structured pruning. Because of the sparse weight matrix, the acceleration of non-structured pruning in the hardware implementation is very limited. While the model compressed by the structured pruning is easier to be deployed on the hardwares.

The structured pruning method takes the structures as the basic pruning unit, such as channels, residual blocks, and residual groups. Li et al. [7] firstly proposed the method to prune the unimportant filters of the convolutional layers, evaluating the importance of each filter via the absolute sum of the weights. Liu et al. [8] presented a "sparse learning – pruning – fine-tuning" pipeline to prune channels. He et al. [20] developed a soft channel pruning pipeline to remedy the problem of information loss in [8]. Then, a method for pruning residual blocks and residual groups was proposed by Huang and Wang [9]. Zhang, Zhong and Li [11] extended the pruning method to the detection tasks, which iteratively pruned YOLOv3 to accelerate the model on unmanned aerial

vehicles. Then, Li et al. [10] combined the block-wise pruning and channel-wise pruning of YOLOv3 by elaborately designed rules to reduce the cost of the model on the environment perception devices of vehicles. In this paper, we also aim to automatically prune blocks and channels of the YOLOv3 [19] model via joint search.

B. Automatic Network Pruning

Currently, Neural Architecture Search (NAS) methods have been proposed to automatically search network architectures to reduce the intensity of human labor [21] [22]. Then, several papers have presented the techniques to combine network pruning and NAS. AMC [14] proposed an Automatic Machine Learning (AutoML) engine to generate the pruning ratio of each layer with continuous search space. MetaPruning [16] utilized the evolutionary computation to search the pruned networks, and then the structure and weights can be efficiently sampled from a meta network without fine-tuning. DMCP [23] introduced a differentiable search method for channel pruning, with modeling the pruning procedure as a Markov process. CACP [24] also cast the channel pruning into a Markov decision procedure, and the pruned models with different compression ratios can be searched at the same time. ABCPruner [17] optimized the pruned structure by the artificial bee colony algorithm, but the search space of the pruning was discrete. AACP [25] presented an automatic channel pruning algorithm that can optimize the FLOPs, inference time, and model size simultaneously. Nevertheless, these methods only considered channel-wise pruning. AutoCompress [15] combined the channel pruning and the non-structured pruning and searched the best pruning action by DRL. In this paper, through DRL with both discrete and continuous search space, we aim to search for a block-wise and channel-wise pruning action that can reduce resource costs with almost no accuracy loss.

III. METHODOLOGY

In this section, we present our pruning action search method ABCP in Fig. 1. The method aims to automatically search the block-wise and channel-wise redundancy of the overall model by DRL. Inspired by the methods for neural network search [12] [13], for the network which has T layers to prune, a list $a_{1:T}$ consisting of the pruning choice of each block a_{bi} and the channel pruning ratio of each layer a_{li} can be considered as the action for DRL. After pruning and fine-tuning, the testing loss $\mathcal{L}_{test}$ and the FLOPs of the pruned network $\mathcal{F}$ can be used as the reward for DRL. Specifically, a joint sample algorithm which utilizes a stack long short-term memory (LSTM) [26] network has been proposed to the generate block-wise and channel-wise pruning action. The details of the framework are elaborated as follows:

a) Sampling the pruning action: The representation of the large pre-trained model is fed into the joint sample algorithm, and then the pruning action including the block pruning choice for each residual block and the channel pruning ratio for each convolutional layer is sampled. (see Sec.III.A)
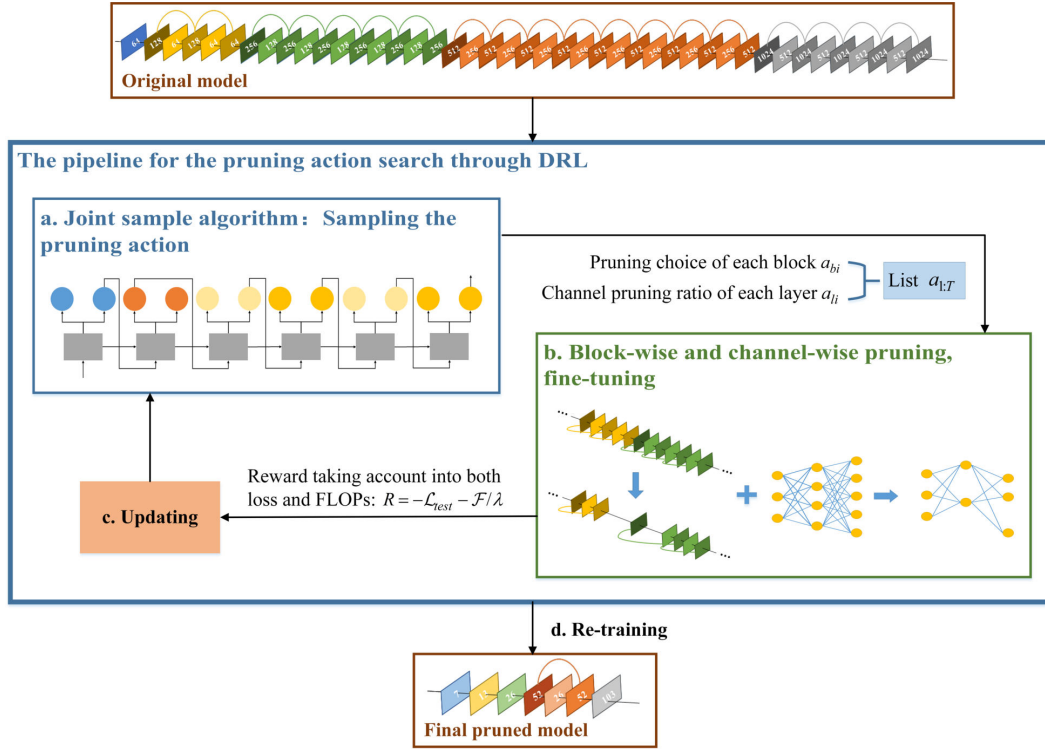
Fig. 1. The framework of Automatic Block-wise and Channel-wise Network Pruning (ABCP). The input of this pipeline is a large pre-trained residual network and the final pruned model has much fewer residual blocks and channels. In each episode, the first step is using the joint sample algorithm to sample the pruning action $a_{1:T}$ which includes the pruning choice of each residual block a_{bi} and the channel pruning ratio of each convolutional layer a_{li}, and then the model is pruned by setting the corresponding weights to zero according to the sampled pruning action. After pruning, several particular layers in the pruned model is fine-tuned on the training dataset with maintaining the values of the weights that have been set to zero in the previous step. In the final step, the loss on the testing dataset $\mathcal{L}_{test}$ is calculated and the FLOPs of the pruned model $\mathcal{F}$ is estimated to generate the reward R, then the weights of the joint sample algorithm are updated by the policy gradient method.

b) Pruning and fine-tuning: Once the pruning action is generated, the corresponding weights in the original models are set to zero. With maintaining the values of the weights which have been set to zero, several particular layers in the model is fine-tuned on the training dataset. (see Sec.III.B)

c) Updating: After finishing the fine-tuning, the loss of the pruned model is calculated on the testing dataset and the FLOPs of the pruned model is estimated. Then the reward that takes both accuracy and FLOPs into account is calculated, and the parameters of the joint sample algorithm are updated by the policy gradient method. (see Sec.III.C)

d) Re-training: After several episodes, the pruning action with the best reward is selected, and the final pruned model is re-trained from scratch. (see Sec.III.D)

A. Sampling the Pruning Action

1) The joint sample algorithm: As demonstrated in Fig. 2, to sample the block-wise and channel-wise pruning action, we propose a joint sample algorithm using the LSTM model. The structure of the residual network that would be pruned is also shown in Fig. 2, involving an ordinary convolutional layer and a residual group that consists of two residual blocks, each residual block consists of two convolutional layers. Each LSTM cell samples the block pruning choice a_{bi} or the layer pruning ratio a_{li} for each corresponding block or layer of the

residual network respectively, which constitutes the list $a_{1:T}$ as the pruning action of the residual network.

Fig.2 also illustrates that there are two branches connected with each LSTM cell to sample the block-wise and channel-wise pruning action. Each branch consists of one or two fully connected (FC) layers and the softmax operation. One branch is to sample the pruning choice of each residual block a_{bi}, and the other is to sample the channel pruning ratio of each convolutional layer a_{li}. In addition, following [13], the block pruning choice or the layer pruning ratio sampled in the previous cell are embedded in the next cell as the input e_i. In this way, a continuous distributed representation is created to capture the potential relationships between the current situation of the pruned network and the block pruning choices as well as the layer pruning ratios sampled by the previous LSTM cells.

Especially, there are four types of LSTM cells according to the positions of the layers they control: the LSTM cell for the ordinary convolutional layer, the LSTM cell for the 1st layer of a residual block, the LSTM cell for the 2nd layer of a residual block, and the LSTM cell for the 1st layer of a residual group:

• The LSTM cell for the ordinary convolutional layer: The LSTM cell for the ordinary convolutional layer takes the sampled layer pruning ratio as the output of this cell, then embeds and feeds it into the next cell along with the cell

(a) The model of the joint sample algorithm.

(b) The structure of the residual network that will be pruned.
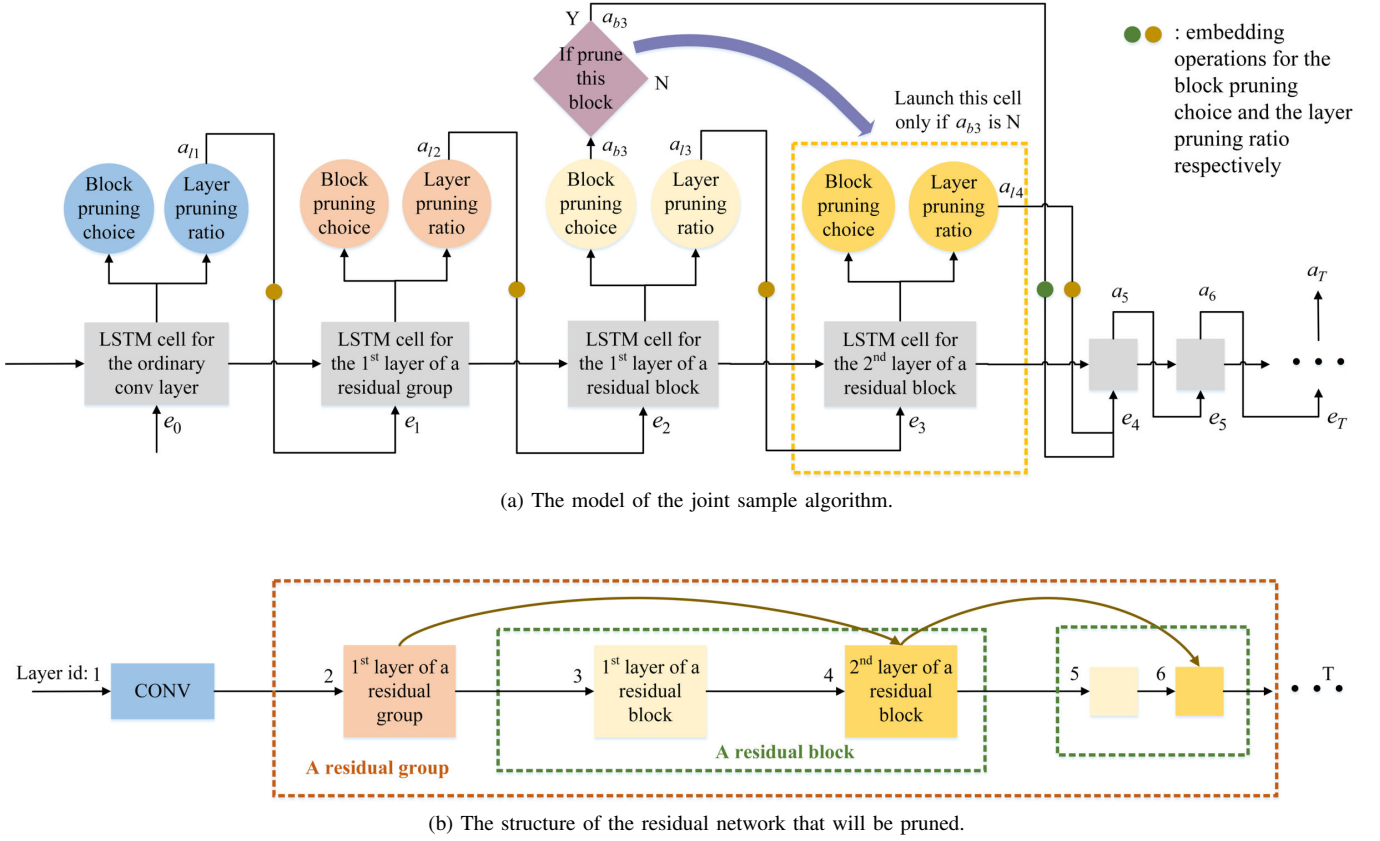
Fig. 2. The process of the pruning action sampling. (a) The model of the joint sample algorithm; (b) The structure of the residual network that will be pruned. The LSTM network is used for the joint sample algorithm where each cell is connected by two branches: one branch outputs the block pruning choice and the other branch outputs the layer pruning ratio. We denote four types of the layers in this residual network, so there are also four types of LSTM cells: the LSTM cell for the ordinary convolutional layer, the LSTM cell for the 1st layer of a residual block, the LSTM cell for the 2nd layer of a residual block, and the LSTM cell for the 1st layer of a residual group. During the pruning action sampling, each LSTM cell samples the corresponding block pruning choice a_{bi} or the layer pruning ratio a_{li} for each block and each layer in the residual network respectively, which constitutes the list $a_{1:T}$ as the pruning action for the whole residual network.

state and the recurrent information.

- The LSTM cell for the 1st layer of a residual block: The LSTM cell for the 1st layer of the residual block makes the block pruning choice whether pruning both the two layers involved in this block or not. When the choice is Yes, the block pruning choice is embedded and fed into the cell after the next cell. When the sampled block pruning choice is No, this cell outputs are the sampled results of the layer pruning ratio branch, which is embedded and fed into the next cell.
- The LSTM cell for the 2nd layer of a residual block: Whether to launch the LSTM cell for the 2nd layer of a residual block depends on the sampled block pruning choice of the previous cell.
- The LSTM cell for the 1st layer of a residual group: As for the LSTM cell for the 1st layer of a residual group, since the 1st layers of the residual groups usually include pooling operations, we treat these layers as the ordinary convolutional layers to only prune the channels for maintaining the accuracy performance.

2) Sampling the block pruning choice: For the block pruning choice search, the action space is discrete, including "pruning" and "no pruning", i.e. $a_{bi} \in \{1, 0\}$. The probabilities of "pruning" and "not pruning" are computed by the softmax

operation. Then the joint sample algorithm samples the block pruning choice from the probability distribution. The sampling process of the ith LSTM cell is denoted as:

$$\boldsymbol{B_i} = \mathcal{B}_i(e_i; \boldsymbol{\theta}_{bi}) \tag{1}$$

$$P(a_{bi} = \mathbb{B}_k) = \frac{\exp(B_i^k)}{\sum_{k'=1}^{K} \exp(B_i^{k'})} \tag{2}$$

$$a_{bi} \sim \pi(a_{bi} \in \mathbb{B}|s; \boldsymbol{\theta}_{bi}) \tag{3}$$

where $\mathbb{B}$ is the action space for the block pruning choice search, K is the cardinality of $\mathbb{B}$, i.e. $K = |\mathbb{B}|$, $K = 2$ here; $\mathbb{B}_k$ is the kth element in $\mathbb{B}$; $\mathcal{B}_i$ in (1) is the FC layer of the block pruning choice branch connected with the ith LSTM cell, e_i is the embedded result of the i-1th block pruning choice, and $\boldsymbol{\theta}_{bi}$ denotes the weights of $\mathcal{B}_i$; $\boldsymbol{B_i}$ is the output of $\mathcal{B}_i$ with K dimensions, and B_i^k is the kth element of $\boldsymbol{B_i}$; a_{bi} is the block pruning choice sampled by the ith LSTM cell, s is the current state. (2) illustrates the softmax calculating process, which generates the probability distribution of a_{bi}. Then, as denoted in (3), a_{bi} is sampled from the distribution $\pi(a_{bi} \in \mathbb{B}|s; \boldsymbol{\theta}_{bi})$.

After sampling, a embedding map is learned and then each block pruning choice in the discrete action space is mapped to

a tensor in the continuous vector space through the embedding layer.

3) Sampling the layer pruning ratio: For the layer pruning ratio search, the action space can be discrete or continuous. The discrete action space is a coarse-grained space, i.e. $a_{li} \in \{0, 0.225, 0.45, 0.675, 0.9\}$. The probability of each pruning ratio is calculated by the softmax operation. However, it is sensitive for the layers to be pruned by different pruning rates, so that the fine-grained search space may be more suitable for the layer pruning ratio search. Therefore, we also introduce the continuous action space to guide more fine-grained channel pruning, i.e. $a_{li} \in [0, 0.9]$.

For the discrete action space, the sampling process is similar to that of the block pruning choice:

$$
\begin{align}
\boldsymbol{D_i} &= \mathcal{D}_i(e_i; \boldsymbol{\theta}_{li}) \tag{4} \\
P(a_{li} = \mathbb{D}_m) &= \frac{\exp(D_i^m)}{\sum_{m'=1}^{M} \exp(D_i^{m'})} \tag{5} \\
a_{li} &\sim \pi(a_{li} \in \mathbb{D}|s; \boldsymbol{\theta}_{li}) \tag{6}
\end{align}
$$

where $\mathbb{D}$ is the discrete action space for the layer pruning ratio search, M is the cardinality of $\mathbb{D}$, i.e. $M = |\mathbb{D}|$, $M = 5$ here according to the action space mentioned above; $\mathbb{D}_m$ is the mth element of $\mathbb{D}$; $\mathcal{D}_i$ in (4) is the FC function of the layer pruning ratio branch connected with the ith LSTM cell, e_i is the embedded result, and $\boldsymbol{\theta}_{li}$ denotes the weights of $\mathcal{D}_i$; $\boldsymbol{D_i}$ is the output of $\mathcal{D}_i$ with M dimensions, and D_i^m is the mth element of $\boldsymbol{D_i}$; a_{li} is the layer pruning ratio sampled by the ith LSTM cell. (5) demonstrates the softmax calculating process, which can obtain the probability distribution of a_{li}. Finally, (6) shows that a_{li} is sampled from the distribution $\pi(a_{li} \in \mathbb{D}|s; \boldsymbol{\theta}_{li})$.

For the continuous action space, we use the Gaussian distribution to represent the distribution of the layer pruning ratio and sample the ratio with the distribution. The ith LSTM cell is connected with two FC layers to generate the mean and log variance of the Gaussian distribution respectively.

$$
\begin{align}
\hat{\mu}_i &= \boldsymbol{\mu}_i(e_i; \boldsymbol{\theta}_{li}^{\mu}) \tag{7} \\
\hat{\rho}_i &= \boldsymbol{\rho}_i(e_i; \boldsymbol{\theta}_{li}^{\rho}) \tag{8} \\
\hat{\sigma}_i^2 &= \exp(\hat{\rho}_i) \tag{9} \\
\pi(a_{li}|s; \boldsymbol{\theta}_{li}^{\mu}, \boldsymbol{\theta}_{li}^{\rho}) &\sim \mathcal{N}(\hat{\mu}_i, \hat{\sigma}_i^2) \tag{10} \\
a_{li} &\sim \pi(a_{li} \in \mathbb{C}|s; \boldsymbol{\theta}_{li}^{\mu}, \boldsymbol{\theta}_{li}^{\rho}) \tag{11}
\end{align}
$$

where $\boldsymbol{\mu}_i$ and $\boldsymbol{\rho}_i$ in (7) and (8) are the two FC layers in the ith LSTM cell that estimate the mean $\hat{\mu}_i$ and log variance $\hat{\rho}_i$ of the Gaussian distribution respectively, $\hat{\sigma}_i^2$ is the variance of the distribution, and $\boldsymbol{\theta}_{li}^{\mu}$ and $\boldsymbol{\theta}_{li}^{\rho}$ are the weights of $\boldsymbol{\mu}_i$ and $\boldsymbol{\rho}_i$. Experiments show that approximating the log variance has a better practice, so we set the output of the FC layer $\hat{\rho}_i$ as the log variance. Then the variance $\hat{\sigma}_i^2$ is generated with $\hat{\rho}_i$, as shown in (9). (10) illustrates that the distribution of the layer pruning ratio a_{li} can be the Gaussian distribution $\mathcal{N}(\hat{\mu}_i, \hat{\sigma}_i^2)$. (11) shows that a_{li} is sampled in the action space $\mathbb{C}$ (i.e. $[0, 0.9]$ here) from the distribution $\pi(a_{li} \in \mathbb{C}|s; \boldsymbol{\theta}_{li}^{\mu}, \boldsymbol{\theta}_{li}^{\rho})$.

After sampling, the layer pruning ratio sampled by each cell are embedded. For the discrete action space, each pruning ratio is mapped to a tensor in the continuous vector space

---

Algorithm 1: The joint sample algorithm

Input: the LSTM model $\mathcal{C}$ with T cells denoted as $\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_T$; the branch that outputs the block pruning choice of the ith each cell: $\mathcal{C}_{bi}$; the branch that outputs the layer pruning ratio of the ith each cell: $\mathcal{C}_{li}$; the cell state, recurrent information, and embedded result inputted in the ith cell: $c_{i-1}, h_{i-1}, e_{i-1}$; the set of the ids of the first layer in each residual block in the original network: $\mathcal{S}_{frb}$.

1   Initialize the pruning action as an empty list $a_{1:T}$.
     $a_{1:T} = [\quad]$;
2   Initialize c_0, h_0, e_0;
3   **for** $i = 1...T$ **do**
4   **if** $i \neq 1$ **then** Embed a_{i-1} as e_{i-1};
5   **if** i in $\mathcal{S}_{frb}$ **then**
6         $c_i, h_i \leftarrow \mathcal{C}_i(e_{i-1}, c_{i-1}, h_{i-1})$,
7   Generate the block pruning choice probability distribution by branch $\mathcal{C}_{bi}$: (2) $\leftarrow \mathcal{C}_{bi}$,
8   Sample a_{bi} with (3);
9   **if** $a_{bi}=0$ **then**
10   Generate the layer pruning ratio probability distribution by branch $\mathcal{C}_{li}$: (10) $\leftarrow \mathcal{C}_{li}$,
11   Sample a_{li} with (11), $a_i = a_{li}$;
12   **else** $a_i = a_{bi}$;
13   **else if** $i - 1$ in $\mathcal{S}_{frb}$ **then**
14   **if** $a_{bi-1}=0$ **then**
15            $c_i, h_i \leftarrow \mathcal{C}_i(e_{i-1}, c_{i-1}, h_{i-1})$,
16   (10) $\leftarrow \mathcal{C}_{li}$,
17   Sample a_{li} with (11), $a_i = a_{li}$;
18   **else** $a_i = a_{bi-1}$, $c_i = c_{i-1}$, $h_i = h_{i-1}$;
19   **else**
20         $c_i, h_i \leftarrow \mathcal{C}_i(e_{i-1}, c_{i-1}, h_{i-1})$,
21   (10) $\leftarrow \mathcal{C}_{li}$,
22   Sample a_{li} with (11), $a_i = a_{li}$;
23   Add a_i into $a_{1:T}$;
24 **end**

Output: the pruning action $a_{1:T}$.

---

through an embedding layer and fed into the next cell. For the continuous search space, the pruning ratio is first rounded down, and then is mapped to a tensor.

The joint sample algorithm is detailed in Algorithm 1.

B. Pruning and Fine-tuning

Once the pruning action is generated, the original pre-trained model should be pruned and fine-tuned. For the block pruning, we directly set the weights of the layers involved in the blocks to zero. Due to the existence of the shortcut operations, the block pruning does not influence the inference of the network. As for the channel pruning, we are supposed to select which channel to prune firstly.

As proposed in [8] and [27], the absolute value of the scale factor γ in the batch normal (BN) layer can represent the

importance of the channel. BN layer follows the convolutional layer in the network, which can be formulated as:

$$x_{out}^{p,q} = \gamma^{p,q} \frac{x_{in}^{p,q} - \phi_\Omega^{p,q}}{\sqrt{\delta_\Omega^{p,q} + \varepsilon}} + \beta^{p,q} \quad (12)$$

where the superscript p, q means the qth channel of the pth convolutional layer; $x_{in}^{p,q}$ and $x_{out}^{p,q}$ are the input and output of the BN layer; $\phi_\Omega^{p,q}$ and $\delta_\Omega^{p,q}$ are the mean and standard deviation of $x_{in}^{p,q}$ over the Ωth batch; $\gamma^{p,q}$ and $\beta^{p,q}$ are the scale and shift parameters, which are trainable.

Since $x_{in}^{p,q}$ is the output of the qth channel of the pth convolutional layer, $\gamma^{p,q}$ determines the output of the corresponding channel. In addition, $x_{in}^{p,q}$ has been normalized to multiply with $\gamma^{p,q}$. So the importance of the channel can be represented by the absolute value of $\gamma^{p,q}$. Hence, we sort the absolute values of γ in the original model and set the weights of the channels with smaller absolute γ to zero according to the layer pruning ratios.

It is worth noting that the numbers of the channels in the convolutional layers connected by the shortcut operations in each residual block must be equal. To this end, all of the pruning ratios of these layers are forced to be equal to the maximum pruning ratio in the residual groups.

After pruning, we fine-tune the pruned model for one epoch to recover the accuracy with maintaining the values of the weights that have been set to zero. Especially, to speedup the pruning action search, as mentioned in Sec.IV.B, we only fine-tune several particular layers.

C. Updating

The parameters of the joint sample algorithm are updated by the policy gradient method. Firstly, as shown in (13), we define a reward R that takes both the accuracy evaluated on the testing dataset and the cost of the pruned model into account for assessing the joint sample algorithm performance:

$$R = -\mathcal{L}_{test} - \mathcal{F}/\lambda \quad (13)$$

where $\mathcal{L}_{test}$ is the loss of the pruned and fine-tuned model, which is calculated on the testing dataset; $\mathcal{F}$ is the estimated total FLOPs of the pruned model; λ is a trade-off hyper-parameter to balance the accuracy and the complexity. During the joint sample algorithm updating, we compute the sum of the FLOPs of every convolutional layer $\mathcal{F}_{layer}$ to approximate the total FLOPs $\mathcal{F}$ of the pruned model. (14) demonstrates how to estimate $\mathcal{F}_{layer}$:

$$\mathcal{F}_{layer} = H \times W \times S \times S \times C_{in} \times C_{out} \quad (14)$$

where H and W are the height and width of the feature map input in the convolutional layer, S is the kernel size, C_{in} and C_{out} are the numbers of the input channels and the output channels of the convolutional layer respectively.

As shown in (15), the goal of the policy gradient method is to maximize the expected reward to find the best pruning action, represented by $J(\theta)$:

$$J(\theta) = \mathbb{E}_{\pi(a_{1:T};\boldsymbol{\theta})}[R(a_{1:T})] \quad (15)$$

Where the reward $R(a_{1:T})$ is computed with the pruning action $a_{1:T}$ for the model which have T layers to prune, and $a_{1:T}$ is sampled from the probability distribution $\pi(a_{1:T};\boldsymbol{\theta})$; $\boldsymbol{\theta}$ denotes the weights of the joint sample algorithm.

We employ the Adam optimizer [28] to optimize the parameters of the joint sample algorithm, and the gradient is computed by REINFORCE [18] as (16):

$$\nabla_\theta J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{\pi(a_{1:T};\boldsymbol{\theta})}[\nabla_\theta \log \pi(a_t|s;\boldsymbol{\theta})R(a_{1:T})] \quad (16)$$

where s is the current state, which can be denoted as $a_{(t-1):1}$ in this task.

Through the Monte Carlo estimate, an empirical approximation of (16) is shown as (17). And in order to reduce the high variance, a moving average baseline is employed in this estimate:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi(a_t|a_{(t-1):1};\boldsymbol{\theta})[R(a_{1:T,n}) - b] \quad (17)$$

where N is the number of different pruning actions $a_{1:T}$ that the joint sample algorithm samples in one episode; $R(a_{1:T,n})$ is the reward calculated with the nth pruning action; b is the moving average baseline. According to [13], although the Monte Carlo estimate can approximate $\nabla_\theta J(\theta)$ unbiasedly, this method would bring a high variance when only the joint sample algorithm is trained. Furthermore, [13] has found that $N = 1$ works well. So we let $N = 1$ and let the reward calculated with one pruning action sampled from $\pi(a_{1:T};\boldsymbol{\theta})$ be the expected reward.

D. Re-training

After finishing the joint sample algorithm training and searching, several candidate pruning actions are obtained. We only take the pruning action with the highest reward to get a new pruned architecture, then the pruned model is re-trained from scratch. Maybe it is better for us to prune and re-train the models pruned by all the sampled pruning actions and to choose the one with the best performance, but it is time-consuming.

IV. EXPERIMENTS

In the area of computer vision, object detection plays an essential role and is applied in increasing industrial areas. Among the existing detectors, YOLOv3 [19] is the most promising and classic model with excellent real-time performance and considerable accuracy. Extensive diversity of methods have been presented to improve YOLOv3 (e.g. YOLOv4 [29], etc.), while compared with these new modified networks, the operations in YOLOv3 are much more basic and simple, which are easier to be put into use. Currently, the YOLOv3 model is widely deployed on the embedded devices, hence the hope is to achieve better performance to meet the practical needs by compression.

To this end, YOLOv3 is adopted to illustrate the performance of our proposed ABCP framework in this section. We first introduce three datasets namely the UCSD dataset, the

TABLE I
THREE DATASETS FOR EXPERIMENTS

Datasets		resolution	frames		classes
			training set	testing set	
UCSD dataset		320×240	683	76	truck, bus, car
mobile robot detection dataset		1024×512, 640×480	13,914	5,969	red robot, red armor, blue robot, blue armor, dead robot
sim2real detection dataset	the simulation dataset	640×480	5,760	1,440	robot
	the real-world dataset	640×480	—	3,019	robot

(a)   (b)   (c)

Fig. 3. Examples of the UCSD dataset. (a) the sparse traffic; (b) the medium-density traffic; (c) the dense traffic.

(a)   (b)
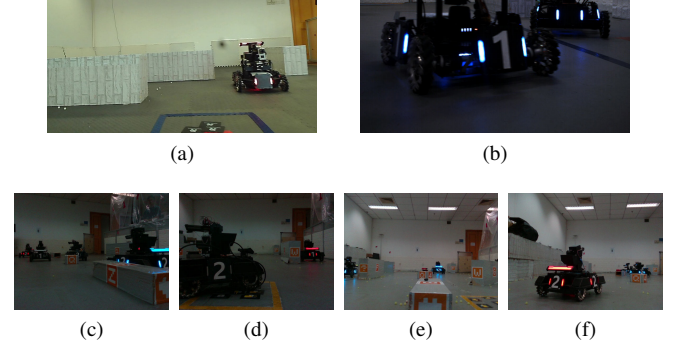
(c)   (d)   (e)   (f)

Fig. 4. Examples of the mobile robot detection dataset. Different exposures as well as various distances and angles of the robots are performed. (a) and (b) are the 1024×512 examples with different exposures, distances and angles; (c) - (f) are the 640×480 examples with different exposures, distances and angles.

mobile robot detection dataset, as well as the sim2real detection dataset respectively, and then present the implementation settings. Secondly, the ablation experiments on the UCSD dataset are conducted to learn the significance of the block-wise and channel-wise joint search and the continuous action space of the search for channel pruning. Next, the effectiveness of our pruned model is evaluated on the UCSD dataset and the mobile robot detection dataset. Finally, the robustness of our method is demonstrated on the sim2real detection dataset.

A. Dataset

The series of the YOLO models are designed for relatively complex detection tasks, such as the object detection tasks on the VOC dataset (20 classes) [30] and the COCO dataset (80 classes) [31]. However, in many practical applications, the abundant detection classes are not needed since the objects in the detection tasks are relatively single, while the real-time requirement is high. In this situation, the structure of the YOLOv3 network is always redundant, and ABCP is proposed for the network pruning in simple tasks. Therefore, we evaluate ABCP on three detection datasets [32] including vehicle detection and mobile robot detection, whose classes are relatively simple, shown in Table I.

1) UCSD dataset: The UCSD dataset is a small dataset captured from the freeway surveillance videos collected by UCSD [33]. As shown in Fig.3, this dataset involves three different traffic densities each making up about one-third: the sparse traffic, the medium-density traffic, and the dense traffic. We define three classes in this dataset: truck, car, and bus. The resolutions of the images are all 320×240. The training and testing sets contain 683 and 76 images respectively.

2) Mobile robot detection dataset: As shown in Fig.4, the mobile robot detection dataset is collected by the robot-mounted cameras to meet the requirements of the fast and lightweight detection algorithms for the mobile robots. There

are two kinds of ordinary color camera with different resolutions which are 1024×512 and 640×480 respectively. Five classes have been defined: red robot, red armor, blue robot, blue armor, dead robot. The training and testing sets contain 13,914 and 5,969 images respectively. During collecting, we change series of exposures as well as various distances and angles of the robots.

3) Sim2real detection dataset: The sim2real detection dataset is divided into two sub-datasets: the real-world dataset and the simulation dataset. We search and train the model on the simulation dataset and test it on the real-world dataset. Firstly, we collect the real-world dataset by the surveillance-view ordinary color cameras in the field. The field and the mobile robots are the same as those in the mobile robot detection dataset. Secondly, we leverage Gazebo to simulate the robots and the field from the surveillance view. Then we capture the images of the simulation environment to collect the simulation dataset. The resolutions of images in the sim2real dataset are all 640×480. There is only one object class in these two datasets: robot. The training and testing sets of the simulation dataset contain 5,760 and 1,440 images respectively, and the testing set of the real-world dataset contains 3,019 images. Examples in the two datasets are demonstrated in Fig.5.

B. Implementation Details of ABCP

We jointly search the block pruning choice with the discrete search space and the layer pruning ratio with the continuous search space in all of the experiments.

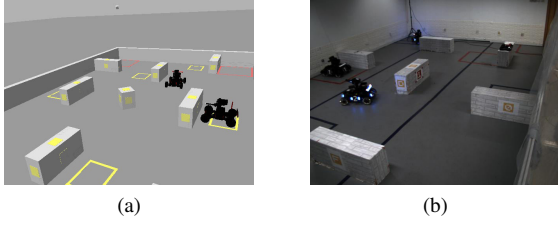(a)   (b)

Fig. 5. Examples of the sim2real detection dataset. (a) an example of the simulation dataset; (b) an example of the real-world dataset.

For pre-training the original YOLOv3 model, we firstly train the three prediction layers through the Adam optimizer for 20 epochs. The learning rate in the first stage is 10^{-3} for the UCSD dataset and is 10^{-4} for other datasets. Secondly, we train all of the weights in the model with Adam optimizer until the loss converges. The learning rate in the second stage is 10^{-4} for the UCSD dataset and is 10^{-6} for other datasets. The batch size is 64 and the weight decay is 5×10^{-4} in both two training stages. In addition, we perform the multi-scale image resizing and image augmenting to improve the accuracy.

For updating the parameters in the joint sample algorithm, we initialize the weights θ uniformly in $[-0.1, 0.1]$. For calculating the reward, λ in (13) is set to 5×10^5 for the UCSD dataset and is set to 10^6 for other datasets. The network of the joint sample algorithm is trained for 310 epochs with the Adam optimizer and the learning rate is 10^{-3}. Thus the sample of the pruning action is also processed for 310 times.

For fine-tuning the pruned model, we only train the three prediction layers in the pruned YOLOv3 for 1 epoch with the Adam optimizer, with maintaining the values of the weights that have been set to zero after pruning. The learning rate in this stage is the same as that in the first stage of the pre-training, and the other sets are also the same as those in the pre-training.

For re-training, the final pruned model are re-trained by Darknet [19]. The optimizer is the Stochastic Gradient Descent (SGD) [34]. The total batches are $30,000$ for the UCSD dataset and are $80,000$ for other datasets. The learning rate is set to 10^{-3} with no dropping. The batch size is set to 64, the weight decay is 5×10^{-4} and the momentum factor is 0.9.

C. Compared Algorithms and Evaluation Metrics

In the following experiments, we train the original YOLOv3 models [19] on the three datasets as the baseline and then prune the YOLOv3 models by ABCP. At present, YOLOv4 [29] develops a powerful method to improve YOLOv3 to get more efficient and more accurate. In addition, there is also a faster version of YOLOv3 named YOLO-tiny [19]. Hence, the YOLOv4 models and the YOLO-tiny models are also trained on the three datasets to compare with our pruned models. These models are all trained by Darknet with the SGD optimizer. During the training, following [19], we set the total batches to $50,200$ for the YOLOv3 and YOLOv4 models, to $500,200$ for the YOLO-tiny models. The initial learning rate

is 10^{-3}, which drops to 10^{-4} at 80% of the total batches and drops to 10^{-5} at 90% of the total batches.

We also run a rule-based block-wise and channel-wise pruning algorithm (RBCP) proposed in [10] to iteratively prune the YOLOv3 model. RBCP takes the YOLOv3 model trained by Darknet as the original model. During the iterative process, all the intermediate models and the finally pruned models are also trained by Darknet with the same hyper-parameters as those in the re-training of ABCP.

To evaluate the performance of the models, we use mean of average precision (mAP), FLOPs, the number of the parameters (Params) and the average inference time to represent the accuracy, the complexity, and the inference speed of the models. During the evaluating, images are resized to 416×416 before they are fed into the networks. The average inference time is tested on a NVIDIA MX250 GPU card, whose resource is very limited. In addition, there are three thresholds during the evaluation of the series of YOLO models [19]: the intersection over union (IOU) threshold is to calculate the IOUs between the predicted bounding boxes and the actual bounding boxes and to filter the predicted bounding boxes whose IOUs are smaller than the IOU threshold, which is set to 0.5; the confidence threshold is to filter the predicted bounding boxes whose confidences are smaller than the confidence threshold, which is set to 0.8; and the non-maximum suppression threshold [35] is set to 0.5.

D. Ablation Study

We ascribe the excellent performance of ABCP to two points: 1) ABCP prunes both residual blocks and channels via the joint search of the block-wise and channel-wise pruning action; 2) the search space for channel pruning is continuous. In the following experiments shown in Table II, we prove the contributions of these two points on the UCSD dataset.

1) Effects of the joint search: To explore the effectiveness of the block-wise and channel-wise joint search, as shown in Table II, the model pruned by ABCP is compared with two models pruned through the single-wise search: ABCP-w/o-C is the model pruned with the action generated by the block-wise pruning action search; ABCP-w/o-B is the model pruned with the action generated by the channel-wise pruning action search. The implementation details are the same as those of ABCP.

The results show that compared with the FLOPs of ABCP, ABCP-w/o-B can achieve a comparable compression ratio while ABCP-w/o-C is still much more resource-consuming. It is because that the block-wise pruning is coarse-grained while the channel-wise pruning can prune more fine-grained structures. Therefore, ABCP combines the coarse-grained and fine-grained pruning and can obtain an ultra-slim pruned model. As for the comparison of accuracy, ABCP reaches the highest mAP among these models. It is validated that the joint search can sample a better pruning action to accomplish a larger compression ratio with low accuracy loss.

2) Effects of the continuous search space: To check the effects of the continuous search space for the channel-wise pruning, as shown in Table II (ABCP-D), we prune the

TABLE II
RESULTS OF THE ABLATION STUDY ON THE UCSD DATASET

Models	mAP (%)	FLOPs (G)	Params (M)	Inference Time (s)
ABCP	**69.6**	**4.485**	4.685	0.016
ABCP-w/o-C	64.3	35.283	36.096	0.061
ABCP-w/o-B	58.1	4.943	9.249	0.020
ABCP-D	63.5	4.924	**4.340**	**0.015**

model with the action generated by the joint search, while the action space is discrete for the search of the channel-wise pruning, which is set to $\{0, 0.225, 0.45, 0.675, 0.9\}$. The implementation details are the same as those of ABCP.

The results show that ABCP achieves better accuracy with fewer FLOPs as well as the comparable Params and inference speed. It is verified by experiments that the continuous search space is more suitable for the pruning task since the models are sensitive for the layers to be pruned by different pruning ratios. In addition, the continuous search space contributes to getting a higher compression ratio.

E. Results and Analysis

According to the conclusion of the ablation study, we train and prune YOLOv3 models on three datasets by ABCP. Then the pruned models are compared with YOLOv4, YOLO-tiny, as well as the models pruned by RBCP.

1) Results on the UCSD dataset: The performances of the models on the UCSD dataset are demonstrated in Table III. The results show that the mAP of our pruned model surpasses the baseline YOLOv3 model by **8.2**% with **93.2**% FLOPs reduction, **92.4**% Params reduction, and **6.87**× speed up. Fig. 6 shows the great detection results of our pruned model, and the video of the detection results is demonstrated on the github. The pruned model also achieves 6.5% higher mAP than the YOLOv4 model with much fewer FLOPs and Params as well as much faster speed. The highest mAP of ABCP reflects the redundancy of the structures of YOLOv3 and YOLOv4, which are not suitable for the relatively simple detection tasks. Compared with the YOLO-tiny model, ABCP outperforms it by a large margin. It is due to in YOLO-tiny, the parameters are reduced by replacing the residual blocks with the ordinary convolutional layers and cutting a level of the feature pyramid structure, leading to irrecoverable accuracy loss.

As for the rule-based block-wise and channel-wise joint pruning method RBCP, since the pruning process of RBCP is iterative, we perform the pruning pipeline iteratively until the pruning causes a dramatic accuracy loss or the results of the sparse training indicates that there are almost no redundant structures to be pruned. During the pruning iteration of RBCP, the mAP of the 7th pruned model reaches 66.5%, but the FLOPs and Params are both much larger than ours. However, in the next iteration, the mAP of the 8th pruned model drops significantly, while the FLOPs reduction is little. In addition, after the 8th iteration, almost no parameter is close to zero after sparse learning, the iteration process is forced to terminate. Hence, we terminate the pruning iteration here and

Fig. 6. The detection results of the model pruned by ABCP on the UCSD dataset.

TABLE III
RESULTS OF THE MODELS ON THE UCSD DATASET

Models	mAP (%)	FLOPs (G)	Params (M)	Inference Time (s)
YOLOv3 [19]	61.4	65.496	61.535	0.110
YOLOv4 [29]	63.1	59.659	63.948	0.132
YOLO-tiny [19]	57.4	5.475	8.674	**0.014**
RBCP [10]	66.5	17.973	4.844	0.042
ABCP (Ours)	**69.6**	**4.485**	**4.685**	0.016

take the performance of the 7th pruned model as the result of RBCP in Table. III. Compared with the performance of the models pruned by ABCP shown in Table. III, it is verified by experiments that RBCP can not achieve the sufficient compression ratio.

Additionally, Fig. 7 demonstrates the pruning ratios of each layer of the models with the best performance pruned by RBCP and ABCP. The biggest difference between the two policies is the pruning ratios of the first 24 layers. For RBCP, the pruning ratio of each layer is the sum of the pruning ratios generated in all previous iterations. Hence, during the pruning iteration, the pruning ratios of the first 24 layers at each time are always confined in a small range, which results in the limited FLOPs reduction of RBCP. We suppose that it is another manifestation of the limited compression of RBCP. Therefore, it is easier for ABCP to find the best pruned network.

Furthermore, we attempt to use RBCP to prune the model pruned by ABCP. The block-wise pruning and the channel-wise pruning are processed iteratively in RBCP. After the block-wise pruning, three residual blocks are pruned and the FLOPs is reduced by 0.07G. Nevertheless, the inference speed is the same (0.016s) and the mAP is dropped by 0.04%. Sequentially, the channel-wise pruning is performed, but the results of the sparse training demonstrate that there are almost no redundant channels in the model. It is verified by experiments that the model pruned by ABCP can no longer be optimized by RBCP.

2) Results on the mobile robot detection dataset: The performances of the models on the mobile robot detection dataset are demonstrated in Table IV. The results show that compared with the baseline YOLOv3 model, our pruned model saves **99.5**% FLOPs, reduces **99.5**% Params, and achieves
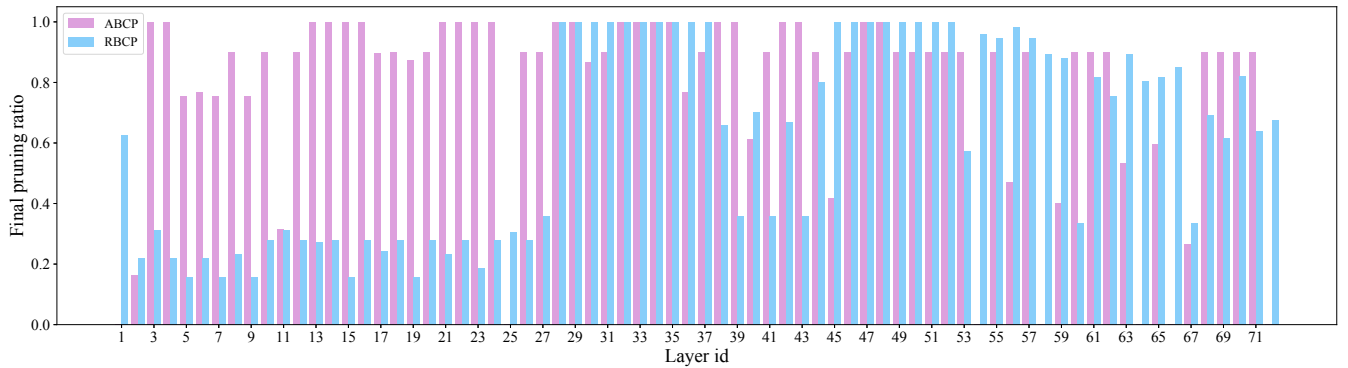
Fig. 7. The pruning ratios of each layer of the models with the best performance pruned by ABCP and RBCP. The vacancy of the column represents the pruning ratio is 0%.

TABLE IV
RESULTS OF THE MODELS ON THE MOBILE ROBOT DETECTION DATASET

Models	mAP (%)	FLOPs (G)	Params (M)	Inference Time (s)
YOLOv3 [19]	**94.9**	65.510	61.545	0.112
YOLOv4 [29]	92.1	59.673	63.959	0.141
YOLO-tiny [19]	85.3	5.478	8.679	0.014
RBCP [10]	89.9	2.842	1.879	0.012
ABCP (Ours)	92.1	**0.327**	**0.299**	**0.003**

Fig. 8. The detection results of the model pruned by ABCP on the mobile robot detection dataset.

37.3× speed up with only 2.8% accuracy loss. The accuracy of our pruned model is the same as that of YOLOv4. It is also demonstrated that reaching excellent detection accuracy on the simple task does not require such complex structures as YOLOv3 and YOLOv4. Fig. 8 shows the great detection results of our pruned model, and the video of the detection results is demonstrated on the github. Compared with YOLO-tiny and RBCP, ABCP outperforms them by a large margin with much lower accuracy loss and much higher compression ratio.

Compared with the UCSD dataset, the objects in the mobile robot detection dataset are sparser and more obvious, so that the detection task on this dataset is simpler. Hence, the results also illustrate that ABCP can work much better than RBCP on a simple dataset.

Moreover, we deploy the model pruned by ABCP on the NVIDIA® Jetson AGX Xavier™ platform mounted on the robot. After speeded up by NVIDIA® TensorRT™, the inference speed can reach approximate 300 frames per second, which manifests that the pruned model also has remarkable property on the embedded devices.

3) Results of the transfer task on the sim2real detection dataset: The transfer task on the sim2real dataset is to train the model on the simulation dataset and then transfer the model on the real-world dataset. In the following experiments, we search and train the model on the simulation dataset, and directly transfer the weights to the real-world dataset with no fine-tuning to evaluate the performance on the real-world dataset. Table V illustrates the results of the models on this task. Compared with the baseline YOLOv3 model, under **97.6%**
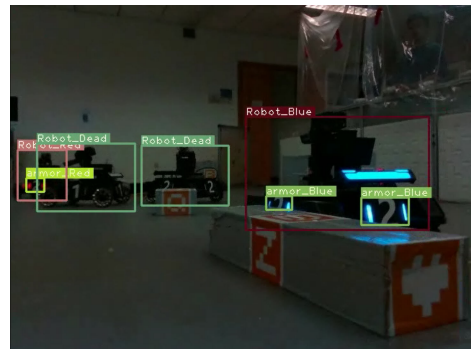
FLOPs reduction, **95.8%** Params reduction, and **14.6×** speed up, our pruned model achieves **2.4%** better accuracy on the simulation dataset and **9.6%** better accuracy on the real-world dataset. Compared with other models, the performances tested on the simulation dataset are similar, while the model pruned by ABCP achieves the best accuracy on the real-world dataset with the fewest FLOPs. Fig. 9 shows the visualization of the detection results comparison on the simulation dataset and the real-world dataset, and the video of the detection results of the model pruned by ABCP is demonstrated on the github. It can be seen that the model pruned by ABCP has better accuracy on the real-world dataset. It is verified by experiments that the model pruned by ABCP has better robustness performance.

Furthermore, it has been shown that YOLOv4 does not perform well on this transfer task, which may be caused by the overfitting problems as the YOLOv4 model has more redundant parameters. At the same time, YOLOv3 also has the overfitting problems. In addition, the accuracy of YOLO-tiny on the real-world dataset loses much more than ABCP, probably reflecting that the slim model generated by this method does not capture all of the available features of the objects.

V. CONCLUSION

In this paper, we propose ABCP, which jointly search the block-wise and channel-wise pruning action through DRL,
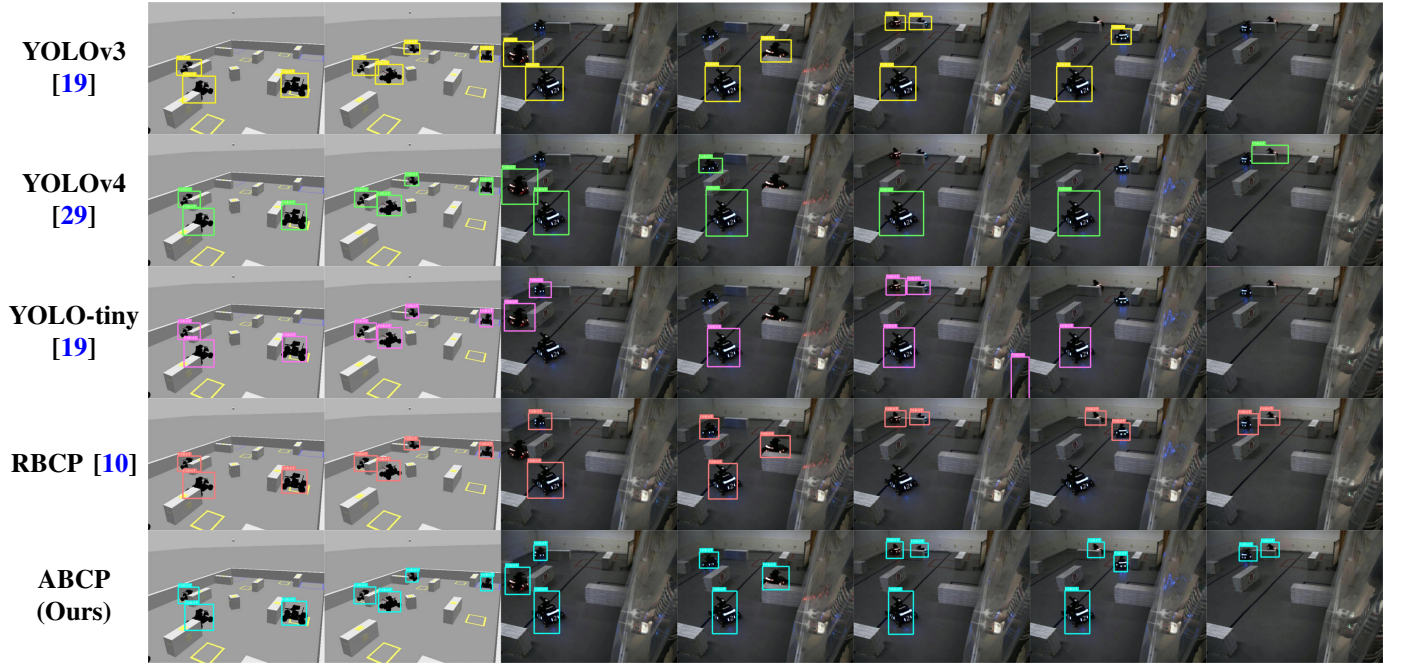
Fig. 9. The visualization of the detection results comparison on the simulation dataset and the real-world dataset. The first two columns are the detection results on the simulation dataset, and other columns are the detection results on the real-world dataset.

TABLE V
RESULTS OF THE TRANSFER TASK ON THE SIM2REAL DETECTION
DATASET

Models	mAP (%)		FLOPs (G)	Params (M)	Inference Time (s)
	sim dataset	real dataset			
YOLOv3 [19]	95.6	66.5	65.481	61.524	0.117
YOLOv4 [29]	**98.3**	28.8	59.644	63.938	0.141
YOLO-tiny [19]	**98.3**	42.3	5.472	8.670	0.014
RBCP [10]	97.9	71.2	2.321	**1.237**	0.009
ABCP (Ours)	98.0	**76.1**	**1.581**	2.545	**0.008**

pruning both residual blocks and channels automatically. A joint sample algorithm is proposed to generate the pruning choice of each residual block and the channel pruning ratio of each convolutional layer in the models. Evaluated on YOLOv3 with three datasets, the results indicate that our method outperforms the traditional rule-based pruning methods with better accuracy and higher compression ratio. Furthermore, since ABCP is only applied to the detection models in the experiments, we would like to apply the proposed method on more deep learning tasks, such as image classification and 3D object detection.

REFERENCES

[1] C.-C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, E. Gonina *et al.*, "State-of-the-art speech recognition with sequence-to-sequence models," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 4774–4778.

[2] H. Li, Y. Chen, Q. Zhang, and D. Zhao, "Bifnet: Bidirectional fusion network for road segmentation," *IEEE Transactions on Cybernetics*, 2021.

[3] Y.-H. Tsai, W.-C. Hung, S. Schulter, K. Sohn, M.-H. Yang, and M. Chandraker, "Learning to adapt structured output space for semantic segmentation," in *Proceedings of the IEEE/CVF CVPR*, 2018, pp. 7472–7481.

[4] N. Li, Y. Pan, Y. Chen, Z. Ding, D. Zhao, and Z. Xu, "Heuristic rank selection with progressively searching tensor ring network," *Complex & Intelligent Systems*, pp. 1–15, 2021.

[5] L. Zeng and X. Tian, "Accelerating convolutional neural networks by removing interspatial and interkernel redundancies," *IEEE Transactions on Cybernetics*, vol. 50, no. 2, pp. 452–464, 2018.

[6] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems*, 1990, pp. 598–605.

[7] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *ICLR*, 2016.

[8] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE ICCV*, 2017, pp. 2736–2744.

[9] Z. Huang and N. Wang, "Data-driven sparse structure selection for deep neural networks," in *Proceedings of the ECCV*. Springer, 2018, pp. 304–320.

[10] J. Li, Y. Zhao, L. Gao, and F. Cui, "Compression of yolov3 via block-wise and channel-wise pruning for real-time and complicated autonomous driving environment sensing applications," in *Proceedings of the 2020 25th ICPR*. IEEE, 2021, pp. 5107–5114.

[11] P. Zhang, Y. Zhong, and X. Li, "Slimyolov3: Narrower, faster and better for real-time uav applications," in *Proceedings of the IEEE/CVF ICCV Workshops*, 2019, pp. 0–0.

[12] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *ICLR*, 2016.

[13] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proceedings of the ICML*, 2018, pp. 4095–4104.

[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 *Proceedings of the ECCV*. Springer, 2018, pp. 784–800.

[15] N. Liu, X. Ma, Z. Xu, Y. Wang, J. Tang, and J. Ye, "Autocompress: An automatic dnn structured pruning framework for ultra-high compression rates," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 4876–4883.

[16] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, K.-T. Cheng, and J. Sun, "Metapruning: Meta learning for automatic neural network channel pruning," in *Proceedings of the IEEE/CVF ICCV*, 2019, pp. 3296–3305.

[17] M. Lin, R. Ji, Y. Zhang, B. Zhang, Y. Wu, and Y. Tian, "Channel pruning via automatic structure search," in *IJCAI*, 2020.

[18] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3, pp. 229–256, 1992.

[19] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[20] Y. He, X. Dong, G. Kang, Y. Fu, C. Yan, and Y. Yang, "Asymptotic soft filter pruning for deep convolutional neural networks," *IEEE Transactions on Cybernetics*, vol. 50, no. 8, pp. 3594–3604, 2019.

[21] Z. Ding, Y. Chen, N. Li, D. Zhao, Z. Sun, and C. P. Chen, "Bnas: Efficient neural architecture search using broad scalable architecture," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[22] Y. Chen, R. Gao, F. Liu, and D. Zhao, "Modulenet: Knowledge-inherited neural architecture search," *IEEE Transactions on Cybernetics*, 2021.

[23] S. Guo, Y. Wang, Q. Li, and J. Yan, "DMCP: Differentiable markov channel pruning for neural networks," in *Proceedings of the IEEE/CVF CVPR*, 2020, pp. 1539–1547.

[24] Y. Liu, Y. Guo, J. Guo, L. Jiang, and J. Chen, "Conditional automated channel pruning for deep neural networks," *IEEE Signal Processing Letters*, 2021.

[25] L. Lin, Y. Yang, and Z. Guo, "AACP: Model compression by accurate and automatic channel pruning," *arXiv preprint arXiv:2102.00390*, 2021.

[26] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[27] J. Ye, X. Lu, Z. Lin, and J. Z. Wang, "Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers," in *ICLR*, 2018.

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2014.

[29] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[30] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010.

[31] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Proceedings of the ECCV*. Springer, 2014, pp. 740–755.

[32] J. Li, "Detection datasets for ABCP," https://github.com/DRL-CASIA/Detection-Datasets-for-ABCP.

[33] A. B. Chan and N. Vasconcelos, "Modeling, clustering, and segmenting video with mixtures of dynamic tsconcelos, extures," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 5, pp. 909–926, 2008.

[34] J. M. Cherry, C. Adler, C. Ball, S. A. Chervitz, S. S. Dwight, E. T. Hester, Y. Jia, G. Juvik, T. Roe, M. Schroeder *et al.*, "SGD: Saccharomyces genome database," *Nucleic Acids Research*, vol. 26, no. 1, pp. 73–79, 1998.

[35] A. Neubeck and L. Van Gool, "Efficient non-maximum suppression," in *Proceedings of the 2006 18th ICPR*, vol. 3. IEEE, 2006, pp. 850–855.