ABCP: Automatic Blockwise and Channelwise 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, Graduate Student Member, IEEE, Zicheng Duan, and Dongbin 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, for simple tasks like robotic detection, most of the traditional rule-based network pruning methods cannot reach a sufficient compression ratio with low accuracy loss and are time consuming as well as laborious. In this article, we propose automatic blockwise and channelwise network pruning (ABCP) to jointly search the blockwise and channelwise pruning action for robotic detection by 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 data set, the pruned YOLOv3 model saves 99.5% floating-point operations, reduces 99.5% parameters, and achieves 37.3x speed up with only 2.8% mean of average precision (mAP) loss. On the sim2real detection data set for robotic detection task, the pruned YOLOv3 model achieves 9.6% better mAP than the baseline model, showing better robustness performance.

Index Terms—Joint search, model compression, pruning, reinforcement learning.

I. INTRODUCTION

In recent years, the deep learning methods are widely applied to machine learning tasks, such as speech recognition [1], image segmentation [2], object classification [3], decision making [4], and trajectory prediction [5]. However, the large computational costs of the deep learning models are unaffordable for many resource-constrained devices and make the inference very slow.

To tackle these problems, a large number of approaches have been proposed [6], [7], [8]. 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: 1) the nonstructured pruning [9] and 2) the structured pruning [10]. Since the structured pruning methods utilize structures as the pruning units, such as channels [11], blocks [12], groups [12], as well as both channels and blocks [13], the structured pruning is much more hardware-friendly than the nonstructured pruning. Hence, most researchers tend to pay attention to structured model pruning currently, which is also the focus of this article.

Nevertheless, the existing structured pruning methods still have some problems. First, fine-tuning the hyperparameters like the pruning threshold increases the workload and it is hard to prove which threshold is optimal [11], [12]. Second, the traditional rule-based pruning method always cannot reach a sufficient compression ratio with low accuracy loss, especially for simple tasks like robotic detection. Finally, the iterative pruning has been recommended [13], [14], 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 [15], [16]. 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 [17] utilized the deep reinforcement learning (DRL) to search each pruning ratio of each convolutional layer, several researchers realize the automated network pruning by DRL [18] or the evolutionary computation [19].

However, most of these works [17] only prune the channels and even cannot 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 [20] 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 blockwise and channelwise network pruning (ABCP)\(^1\) to jointly search the channelwise and blockwise pruning action for robotic detection tasks 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 blockwise and channelwise 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 [21].

Extensive experiments suggest that ABCP has a very promising performance. We collect three data sets, two of them are proposed by ourselves for the fast and lightweight detection algorithm for the mobile robots, and the third is captured from the videos collected by the University of California San Diego (UCSD). Based on these three data sets, we evaluate ABCP on YOLOv3 [22]. 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 follows.

1) We propose ABCP a pipeline to jointly search the channelwise and blockwise network pruning action for robotic detection tasks by DRL.

2) 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.

3) We test ABCP on YOLOv3 [22] based on three data sets. The results show that ABCP outperforms the traditional rule-based pruning methods and has competitive search efficiency. On the mobile robot detection data set, the pruned model saves 99.5% FLOPs, reduces 99.5% parameters, and achieves 37.3 x speed up with only 2.8% mean of average precision (mAP) loss. The structure transfer performance on the robot detection data set is also great. On the sim2real detection data set, achieving 9.6% better mAP than the baseline model, the pruned YOLOv3 model shows better robustness performance.

II. RELATED WORKS

A. Network Pruning

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

The structured pruning method takes structures as basic pruning units, such as channels, residual blocks, and residual groups. Li et al. [10] first 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. [11] presented a “sparse learning pruning fine-tuning” pipeline to prune channels. He et al. [23] developed a soft channel pruning pipeline to remedy the problem of information loss in [11]. Then, a method for pruning residual blocks and residual groups was proposed by Huang and Wang [12]. Zhang et al. [14] extended the pruning method to the detection tasks, which iteratively pruned YOLOv3 to accelerate the model on unmanned aerial vehicles. Then, Li et al. [13] combined the blockwise pruning and channelwise pruning of YOLOv3 by elaborately designed rules to reduce the costs of the model on the environment perception devices of vehicles. In this article, we also aim to automatically prune blocks and channels of the YOLOv3 [22] model via joint search.

B. Automatic Network Pruning

Currently, NAS methods have been proposed to automatically search network architectures to reduce the intensity of human labor [24], [25], [26]. Then, several papers have presented the techniques to combine network pruning and NAS. AMC [17] proposed an automatic machine learning (AutoML) engine to generate the pruning ratio of each layer with continuous search space. MetaPruning [19] 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 [27] introduced a differentiable search method for channel pruning, with modeling the pruning procedure as a Markov process. CACP [28] 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 [20] optimized the pruned structure by the artificial bee colony algorithm, but the search space of the pruning was discrete. AACP [29] presented an automatic channel pruning algorithm that can optimize the FLOPs, inference time, and model size simultaneously. Nevertheless, these methods only considered channelwise pruning. AutoCompress [18] combined the channel pruning and the nonstructured pruning and searched for the best pruning action by DRL. Moreover, EagleEye [30] proposed the adaptive batch normalization technique to establish a stronger relationship between the pruning structures and their accuracies, then to solve the unfair evaluation problem in many automatic pruning methods like AMC. In this article, through DRL with both discrete and continuous search spaces, we aim to search for a blockwise and channelwise pruning action that can reduce resource costs with almost no accuracy loss.

\(^1\)Our code is released at https://github.com/DRL-CASIA/ABCP.
III. METHODOLOGY

In this section, we present our pruning action search method ABCP in Fig. 1. The input of ABCP is a pretrained residual model and the output is a final pruned model with fewer residual blocks and fewer convolutional channels. The method aims to automatically search the blockwise and channelwise redundancy of the overall model by DRL. Inspired by the methods for neural network search [15], [16], 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 $L_{\text{test}}$ and the FLOPs of the pruned network $F$ can be used as the reward for DRL. Specifically, a joint sample algorithm that utilizes a stack long short-term memory (LSTM) [31] network has been proposed to generate blockwise and channelwise pruning action. The details of the framework are elaborated as follows.

1) **Sampling the Pruning Action:** The representation of the large pretrained 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 Section III-A).

2) **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 are fine-tuned on the training data set (see Section III-B).

3) **Updating:** After finishing the fine-tuning, the loss of the pruned model is calculated on the testing data set 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 Section III-C).

4) **Retraining:** After several episodes, the pruning action with the best reward is selected, and the final pruned model is retrained from scratch (see Section III-D).

A. Sampling the Pruning Action

1) **Joint Sample Algorithm:** Fig. 2 illustrates the details of the structure of a pretrained residual network. The residual network includes several residual groups and ordinary convolutional layers (the layer that is not in any residual group). Each residual group consists of several residual blocks, and there are two convolutional layers in each residual block. Different convolutional layers have different numbers of channels, like 32, 64, 128, etc. The blockwise pruning of ABCP is to reduce the number of residual blocks. The channelswise pruning of ABCP is to reduce the number of channels in each layer. In Fig. 3, we take the first ordinary convolutional layer and the first residual group in the pretrained residual network as examples to explain how ABCP samples the pruning action.
Fig. 2. Structure of the pretrained residual network. A residual group consists of several residual blocks, each residual block includes two convolutional layers. Different convolutional layers have different numbers of channels, like 32, 64, 128, etc. ABCP aims to reduce the number of residual blocks and the number of channels of each convolutional layer.

Fig. 3. First ordinary convolutional layer and the first residual group in the pretrained residual network are taken as examples to explain the process of the pruning action sampling. (a) Model of the joint sample algorithm. (b) 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 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 block pruning choice or the layer pruning ratio 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.

As demonstrated in Fig. 3, to sample the blockwise and channelwise pruning action, we propose a joint sample algorithm using the LSTM model, because the LSTM model was proposed for the time series prediction, and inference process of the neural network from input to output can be regarded as a time sequence. The example of the 871 structure of the residual network that would be pruned is also shown in Fig. 3, involving an ordinary convolutional layer and a residual group that consists of two residual blocks, each residual block consisting of two convolutional layers. Each LSTM cell samples the block pruning choice $a_{ib}$ or the layer pruning ratio $a_{il}$ 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. 3 also illustrates that there are two branches connected with each LSTM cell to sample the blockwise and channelwise 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 $a_{ib}$, and the other is to sample the channel pruning ratio of each convolutional layer $a_{il}$. In addition, following [16], the block pruning choice or the layer pruning ratio sampled in the previous cell is embedded in the next cell as the input $e_i$. In this way, a continuously 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: 1) the LSTM cell for the ordinary convolutional layer; 2) the LSTM cell for the 1st layer of a residual block; 3) the LSTM cell for the 2nd layer of a residual block; and 4) the LSTM cell for the 1st layer of a residual group.

1) 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 state and the recurrent information.

2) 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.

3) 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.

4) 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 \( i \)th LSTM cell is denoted as

\[
B_i = B_i(e_i; \theta_{bi})
\]

\[
P(a_{bi} = B_k) = \frac{\exp(b_{ki}^s)}{\sum_{k=1}^{K} \exp(b_{ki}^s)} \quad (2)
\]

\[
a_{bi} \sim \pi(a_{bi} \in \mathbb{B}|s; \theta_{bi}) \quad (3)
\]

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 \( k \)th element in \( \mathbb{B} \); \( B_i \) in (1) is the FC layer of the block pruning choice branch connected with the \( i \)th LSTM cell, and \( e_i \) is the embedded result of the \( i \)-th block pruning choice, and \( \theta_{bi} \) denotes the weights of \( B_i \); \( B_i \) is the output of \( B_i \) with \( K \) dimensions, and \( b_{ki}^s \) is the \( k \)th element of \( B_i \); and \( a_{bi} \) is the block pruning choice sampled by the \( i \)th LSTM cell, and \( s \) is the current state. Equation (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 \( \pi(a_{bi} \in \mathbb{B}|s; \theta_{bi}) \).

After sampling, an 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., \( a_i \in \{0, 0.225, 0.45, 0.675, 0.9\} \). 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 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., \( a_i \in [0, 0.9] \).

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

\[
D_i = D_i(e_i; \theta_{li}) \quad (4)
\]

\[
P(a_{li} = D_m) = \frac{\exp(D_m^i)}{\sum_{m=1}^{M} \exp(D_m^i)} \quad (5)
\]

\[
a_{li} \sim \pi(a_{li} \in \mathbb{D}|s; \theta_{li}) \quad (6)
\]

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 \( m \)th element of \( \mathbb{D} \); \( D_i \) in (4) is the FC function of the layer pruning ratio branch connected with the \( i \)th LSTM cell, \( e_i \) is the embedded result, and \( \theta_{li} \) denotes the weights of \( D_i \); \( D_i \) is the output of \( D_i \) with \( M \) dimensions, and \( D_m^i \) is the \( m \)th element of \( D_i \); and \( a_{li} \) is the layer pruning ratio sampled by the \( i \)th LSTM cell. Equation (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 \( \pi(a_{li} \in \mathbb{D}|s; \theta_{li}) \).

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 \( i \)th LSTM cell is connected with two FC layers to generate the mean and log variance of the Gaussian distribution, respectively

\[
\hat{\mu}_i = \mu_i(e_i; \theta_{li}^\mu) \quad (7)
\]

\[
\hat{\rho}_i = \rho_i(e_i; \theta_{li}^\rho) \quad (8)
\]

\[
\hat{\sigma}_i^2 = \exp(\hat{\rho}_i) \quad (9)
\]

\[
\pi(a_{li}|s; \theta_{li}^\mu, \theta_{li}^\rho) \sim \mathcal{N}(\hat{\mu}_i, \hat{\sigma}_i^2) \quad (10)
\]

\[
a_{li} \sim \pi(a_{li} \in \mathbb{C}|s; \theta_{li}^\mu, \theta_{li}^\rho) \quad (11)
\]

where \( \mu_i \) and \( \rho_i \) in (7) and (8) are the two FC layers in the \( i \)th 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 \( \theta_{li}^\mu \) and \( \theta_{li}^\rho \) are the weights of \( \mu_i \) and \( \rho_i \). Experiments show that approximating the log variance has 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). Equation (10) illustrates that the distribution of the layer pruning ratio \( a_{li} \) can be the Gaussian distribution \( \mathcal{N}(\hat{\mu}_i, \hat{\sigma}_i^2) \).

Equation (11) shows that \( a_{li} \) is sampled in the action space \( \mathbb{C} \) (i.e., \([0, 0.9] \)) here from the distribution \( \pi(a_{li} \in \mathbb{C}|s; \theta_{li}^\mu, \theta_{li}^\rho) \).
Algorithm 1: Joint Sample Algorithm

**Input:** the LSTM model $C$ with $T$ cells denoted as $C_1, C_2, \ldots, C_T$; the branch that outputs the block pruning choice of the $i$th cell: $C_{bi}$; the branch that outputs the layer pruning ratio of the $i$th each cell: $C_{li}$; the cell state, recurrent information, and embedded input resulted in the $i$th 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: $S_{frb}$.

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

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

After sampling, the layer pruning ratio sampled by each cell is embedded. For the discrete action space, each pruning ratio is mapped to a tensor in the continuous vector space through an embedding layer and fed into the next cell. For the continuous search space, the pruning ratio is first rounded down, and then 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 pretrained 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, block pruning does not influence the inference of the network. As for the channel pruning, we are supposed to select which channel to prune first.

As proposed in [11] and [32], the absolute value of the scale factor $\gamma$ in the batch normal (BN) layer can represent the importance of the channel. The BN layer follows the convolutional layer in the network, which can be formulated as:

$$x_{out}^{p,q} = \gamma^{p,q} \frac{x_{in}^{p,q} - \mu_{\Omega}^{p,q}}{\sigma_{\Omega}^{p,q} + \epsilon} + \beta^{p,q}$$

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

Since $x_{in}^{p,q}$ is the output of the $q$th channel of the $p$th 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 $\gamma$ in the original model and set the weights of the channels with smaller absolute $\gamma$ 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 speed up the pruning action search, as mentioned in Section 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. First, as shown in (13), we define a reward $R$ that takes both the accuracy evaluated on the testing data set and the cost of the pruned model into account for assessing the joint sample algorithm performance

$$R = -\mathcal{L}_{test} + \mathcal{F}/\lambda$$

where $\mathcal{L}_{test}$ is the loss of the pruned and fine-tuned model, which is calculated on the testing data set; $\mathcal{F}$ is the estimated total FLOPs of the pruned model; $\lambda$ is a tradeoff hyper-parameter to balance the accuracy and the complexity, the bigger $\lambda$ will make the algorithm to search for more accurate models, and the smaller $\lambda$ will make the algorithm to search for the models which have fewer FLOPs. 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. Equation (14) demonstrates how to estimate $\mathcal{F}_{layer}$

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

where $H$ and $W$ are the height and width of the feature map input in the convolutional layer, $S$ is the kernel size, and $C_{in}$
and \( C_{\text{out}} \) 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 \( J(\theta) \)

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

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 \( \pi(a_{1:T}; \theta) \); and \( \theta \) denotes the weights of the joint sample algorithm.

We employ the Adam optimizer [33] to optimize the parameters of the joint sample algorithm, and the gradient is computed by REINFORCE [21] as

\[
\nabla_{\theta} J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{\pi(a_{1:T}; \theta)} \left[ \nabla_{\theta} \log \pi(a_{t}; \theta) R(a_{1:T}) \right] \tag{16}
\]

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). In order to reduce the high variance, a moving average baseline is employed in this estimate

\[
\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} \mathbb{E}_{\theta} \left[ \log \pi(a_{t}; \theta) \right] \left[ R(a_{1:T,n}) - b \right] \tag{17}
\]

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 \( n \)th pruning action; and \( b \) is the moving average baseline. According to [16], 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, [16] has found that \( N = 1 \) works well. So we let \( N = 1 \) and let the reward calculated with one pruning action sampled from \( \pi(a_{1:T}; \theta) \) be the expected reward.

D. Retraining

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 retrained from scratch. Maybe it is better for us to prune and retrain 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 [22] 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 [34], 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 embedded devices; hence, the hope is to achieve better performance to meet the practical needs through compression.

To this end, YOLOv3 is adopted to illustrate the performance of our proposed ABCP framework in this section. We first introduce three data sets, namely, the UCSD data set, the mobile robot detection data set, as well as the sim2real detection data set, respectively, and then present the implementation settings. Second, the ablation experiments on the UCSD data set are conducted to learn the significance of the blockwise and channelwise joint search and the continuous action space of the search for channel pruning. Next, the search efficiency of ABCP and the effectiveness of our pruned model are evaluated on the UCSD data set and the mobile robot detection data set. The performance of the transferred structure of the mobile robot detection data set is also evaluated. Finally, the robustness of our method is demonstrated on the sim2real detection data set.

A. Data Set

The series of the YOLO models are designed for relatively complex detection tasks, such as the object detection tasks on the VOC data set (20 classes) [35] and the COCO data set (80 classes) [36]. However, in many practical applications like in robotic vision areas [37], 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 more suitable for these relatively simple tasks with a few classes of objects like robotic detection tasks. Therefore, we evaluate ABCP on three detection data sets [38] including vehicle detection and robotic detection, whose classes are relatively simple, shown in Table I.

1) UCSD Data Set: The UCSD data set is a small data set captured from the freeway surveillance videos collected by UCSD [39]. As shown in Fig. 4, this data set 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 data set: 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 Data Set: As shown in Fig. 5, the mobile robot detection data set is collected by the robot-mounted cameras to meet the requirements of the fast and lightweight detection algorithms for the mobile robots. There

![Fig. 4. Examples of the UCSD data set. (a) Sparse traffic. (b) Medium-density traffic. (c) Dense traffic.](image)
TABLE I
THREE DATA SETS FOR EXPERIMENTS

| Datasets                    | resolution | frames | classes                                      |
|-----------------------------|------------|--------|----------------------------------------------|
| UCSD dataset                | 320×240    | 683    | truck, bus, car                              |
| mobile robot detection data | 1024×512, 640×480 | 13,914, 5,969 | red robot, red armor, blue robot, blue armor, dead robot |
| sim2real detection data    | 640×480    |        | robot                                        |
| simulation dataset          | 640×480    | 5,760  |                                              |
| real-world dataset          | 640×480    | 1,440  |                                              |

For the pretraining of the original YOLOv3 model, we first 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 data set and is $10^{-4}$ for other data sets. Second, 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 data set and is $10^{-5}$ for other data sets. The batch size is 64 and the weight decay is $5 \times 10^{-4}$ in both two training stages. In addition, we perform multiscale image resizing and image augmenting to improve accuracy.

During the search process, for the UCSD data set and the sim2real detection data set, we use all of the frames in the training set. For the mobile robot detection data set, we sample about 33% frames from the training set to save the search time. For updating the parameters in the joint sample algorithm, we initialize the weights $\theta$ uniformly in $[-0.1, 0.1]$. For calculating the reward, $\lambda$ in (13) is set to $5 \times 10^5$ for the UCSD data set and is set to $10^8$ for other data sets. According to the complexity of the task, the network of the joint sample algorithm is trained for 310 epochs for the UCSD data set and 710 epochs for other data sets with the Adam optimizer, and the learning rate is $10^{-3}$. Thus, the sample of the pruning action is also processed 310 times or 710 times.

After each sample of pruning action, the corresponding weights of the pruned structures will be set to zero. For fine-tuning the pruned model, we only train the three final prediction layers in the pruned YOLOv3 for just 1 epoch with the Adam optimizer. The weights of other layers are frozen. Especially, the moving means and the moving variances of BN layers in the whole model are still being reset and updated without forward-propagation during fine-tuning. It is because the moving means and moving variances of the original large model are out-dated for the pruned model, which easily leads to the unfair evaluation of the pruned models. The learning rate in this stage is the same as that in the first stage of the
pretraining, and the other sets are also the same as those in the pretraining.

For retraining, the final pruned model is retrained by Darknet [22]. The optimizer is the stochastic gradient descent (SGD) [40]. The total batches are 30,000 for the UCSD data set and 80,000 for other data sets. The learning rate is set to $10^{-3}$ with no dropping. The batch size is set to 64, the weight decay is $5 \times 10^{-4}$ and the momentum factor is 0.9.

C. Compared Algorithms and Evaluation Metrics

In the following experiments, we train the original YOLOv3 models [22] on the three data sets as the baseline and then prune the YOLOv3 models by ABCP. At present, YOLOv4 [34] 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 [22]. Hence, the YOLOv4 models and the YOLO-tiny models are also trained on the three data sets to compare with our pruned models. These models are all trained by Darknet with the SGD optimizer. During the training, following [22], 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 blockwise and channelwise pruning algorithm (RBCP) proposed in [13] 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 retraining of ABCP.

To evaluate the performance of the models, we use the 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 evaluation, images are resized to $416 \times 416$ before they are fed into the networks. The average inference time is tested on an NVIDIA MX250 GPU card, whose resource is very limited. In addition, there are three thresholds during the evaluation of the series of YOLO models [22]: 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 0.5 for the UCSD data set and 0.8 for other data sets; and the nonmaximum suppression threshold [41] is set to 0.5.

To evaluate the search efficiency of ABCP, we run ABCP and RBCP on a single NVIDIA GeForce RTX 2080Ti GPU card and calculate the GPU hours required to get the final pruned structure on the UCSD data set. Similarly, to further compare and verify the performance and efficiency of ABCP, we also conduct a random search on the UCSD data set. During the random search, instead of the joint sample algorithm, we sample the block pruning choices and layer pruning ratios uniformly 310 times. Then, we also use (13) to evaluate the performance of the pruned structures and select the best pruned structure as the final pruned model of random search. The steps of pretraining, pruning, and retraining and their implementation details are the same as those of ABCP.

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 blockwise and channelwise pruning action and 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 data set.

1) Effects of the Joint Search: To explore the effectiveness of the blockwise and channelwise joint search, as shown in Table II, the model pruned by ABCP is compared with two models pruned through the singlewise search: 1) ABCP-w/o-C is the model pruned with the action generated by the blockwise pruning action search and 2) ABCP-w/o-B is the model pruned with the action generated by the channelwise pruning action search. The implementation details are the same as those of ABCP.

The results show that compared with the FLOPs of the model pruned by 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 blockwise pruning is coarse grained while channelwise pruning can prune more fine-grained structures. Therefore, ABCP combines coarse-grained and fine-grained pruning and can obtain an ultrasmall 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 channelwise pruning, as shown in Table II (ABCP-D), we prune the model with the action generated by the joint search, while the action space is discrete for the search of the channelwise 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 to the layers to be pruned by different pruning ratios. In addition, the continuous search space contributes to getting a higher compression ratio.
significantly, while the FLOPs reduction is little. In addition, in the next iteration, the mAP of the 8th pruned model drops. However, the mAP of the 7th pruned model reaches 66.5%, but the structures to be pruned. During the pruning iteration of RBCP, sparse training indicate that there are almost no redundant pruning causes a dramatic accuracy loss or the results of the iterative, we perform the pruning pipeline iteratively until the termination. Hence, we terminate the pruning iteration here and 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 cannot achieve a sufficient compression ratio.

Additionally, Fig. 8 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 to 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 blockwise pruning and the channelwise pruning are processed iteratively in RBCP. After the blockwise pruning, three residual blocks are pruned and the FLOPs is reduced by 0.07G. Nevertheless, the inference speed is the same (0.016 s) and the mAP is dropped by 0.04%. Sequentially, the channelwise 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.

For the efficiency of the algorithm, as shown in Table III, on the UCSD data set, ABCP finds a pruned structure in about 0.017 GPU hours and finishes the search process in about 5.24 GPU hours. As for RBCP, during its pipeline, the process including sparse training, pruning, and retraining should be performed iteratively, which leads to its much longer GPU hours. The two reasons are as follows: 1) multiple sparse training and retraining is a very time-consuming process and 2) during its pruning step, the pruning threshold should be manually defined and people need to judge whether the model needs to continue to be pruned after each iteration, so its time cost largely depends on the experience and practice of people’s trial and error. Hence, we can only estimate an approximate time for RBCP, which is absolutely much longer than ABCP.

Compared with random search, ABCP is slightly slower. The random search finds a pruned structure in about 0.015 GPU hours and finishes the search process in about 4.65 GPU hours. However, compared with ABCP, the random pruned model has lower accuracy, larger FLOPs and a larger number of parameters after the same number of searches. Especially, the FLOPs and the number of parameters of the random pruned model are 10.747G and 10.342M, respectively, which are far larger than the model pruned by ABCP. Also, the inference time of the random pruned model is 0.028 s, which is two times longer than ABCP.

### 1) Results on the UCSD Data Set

The performances of the models on the UCSD data set 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. 7 shows the great detection results of our pruned model, and the video of the detection results is demonstrated on 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 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 blockwise and channelwise 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 indicate 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 take the performance of the 7th pruned model as the result of RBCP in Table III.

### 2) Results on the Mobile Robot Detection Data Set

The performances of the models on the mobile robot detection data set are compared with YOLOv4, YOLO-tiny, and the pruned models are compared with YOLOv4, YOLO-tiny, and prune YOLOv3 models on three data sets by ABCP. Then, the pruned models are compared with YOLOv4, YOLO-tiny, as well as the models pruned by RBCP. Additionally, we also verify the search efficiency of ABCP on the UCSD data set.

#### TABLE III

| Models       | mAP (%) | FLOPs (G) | Params (M) | Inference Time (s) | GPU Hours (h) |
|--------------|---------|-----------|------------|--------------------|---------------|
| YOLOv3 [22] | 61.4    | 65.496    | 61.535     | 0.110              | –             |
| YOLOv4 [34] | 63.1    | 59.659    | 63.948     | 0.132              | –             |
| YOLO-tiny [22] | 57.4    | 5.475     | 8.674      | **0.014**          | –             |
| RBCP [13]   | 66.5    | 17.973    | 4.844      | 0.042              | ~168          |
| ABCP (Ours) | **69.6**| **4.485** | **4.685**  | 0.016              | 5.24          |
Fig. 8. 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 DATA SET

| Models         | mAP (%) | FLOPs (G) | Params (M) | Inference Time (s) |
|----------------|---------|-----------|------------|--------------------|
| YOLOv3 [22]    | 94.9    | 65.510    | 61.545     | 0.112              |
| YOLOv4 [34]    | 92.1    | 59.673    | 63.959     | 0.141              |
| YOLO-tiny [22] | 85.3    | 5.478     | 8.679      | 0.014              |
| RBCP [13]      | 89.9    | 2.842     | 1.879      | 0.012              |
| ABCP (Ours)    | 92.1    | 0.327     | 0.299      | 0.003              |

Fig. 9. Detection results of the model pruned by ABCP on the mobile robot detection data set.

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 300 frames per second approximately, which manifests that the pruned model also has remarkable properties on the embedded devices.

Furthermore, we also transfer the pruned structure on the mobile robot detection data set to another robot detection data set to study the structure transfer performance. As shown in Fig. 10, the transfer robot detection data set is also collected by the robot-mounted cameras, with the main difference being the robots in the data set. The field of the transfer robot detection data set is the same as that of the mobile robot detection data set. For the robot, the robots in the mobile detection data set are still in the field, but the main task of the transfer robot detection data set is to detect new robots, ignoring the robots of the mobile robot detection data set. The resolutions of images in the transfer robot detection data set are all 640 × 360. There is only one class: robot. The training and testing sets contain 1540 and 660 images, respectively.

We directly use the pruned structure of the mobile robot detection data set to retrain and test its accuracy on the transfer robot detection data set. The training and testing details are the same as those of the mobile robot detection data set. As demonstrated in Table V, the accuracy of the transferred pruned model is even 0.4% higher than the baseline YOLOv3 model, which can verify the good structure transfer performance of ABCP on the relatively simple robot data set. Fig. 10 illustrates the detection results of the transferred pruned model. It can be shown that the robot in the transfer robot detection data set can be detected correctly.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
3) Results on the sim2real Detection Data Set: The task on the sim2real data set is to train the model on the simulation data set and then transfer both the structure and weights to the real-world data set. In the following experiments, we search and train the model on the simulation data set, and directly transfer the weights to the real-world data set with no fine-tuning to evaluate the performance on the real-world data set. Table VI 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 14.6x speed up, our pruned model achieves 2.4% better accuracy on the simulation data set and 9.6% better accuracy on the real-world data set. Compared with other models, the performances tested on the simulation data set are similar, while the model pruned by ABCP achieves the best accuracy on the real-world data set with the fewest FLOPs. Fig. 11 shows the visualization of the detection results comparison on the simulation data set and the real-world data set, and the video of the detection results of the model pruned by ABCP is demonstrated on GitHub. It can be seen that the model pruned by ABCP has better accuracy on the real-world data set. 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 task, which may be caused by the overfitting problems as the YOLOv4 model has more redundant parameters. At the same time, YOLOv3 also has overfitting problems. In addition, the accuracy of YOLO-tiny on the real-world data set 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. DISCUSSION AND CONCLUSION

A. Discussion

Inspired by the one-shot neural network architecture search [42] and pruning [43], there may be some potential improvement for ABCP. In the one-shot network architecture search method, a super network will be defined and trained first, then through weight sharing, the sampled subnetwork will directly inherit the weights of the super network.
way, the performance of each subnetwork will be evaluated quickly. Hence, ABCP can be regarded as a kind of one-shot network search method: the pretrained original network that will be pruned is a super network, then each pruned structure can be seen as a subnetwork. However, the search space in the one-shot method is discrete, which means the width and depth of the subnetwork are chosen in the predefined candidates. In ABCP, we use continuous search space to pursue a more suitable subnetwork. Compared with the one-shot methods, during the iterative process of ABCP, fine-tuning each pruned structure to evaluate its performance may cost more GPU hours, so we would like to find a quicker performance evaluation algorithm.

In addition, now ABCP is more suitable the relatively simple tasks, where the original large network structure is very redundant for these tasks. This situation constrains the application scope of ABCP. It would be a research point to improve its performance on large data sets, such as VOC and COCO, which have more classes and more complex objects.

Furthermore, since ABCP is only applied to the detection models in the experiments, it is worth researching how to apply the proposed method to more deep learning tasks in the future, such as image classification and 3-D object detection.

B. Conclusion

In this article, we proposed ABCP, which jointly searches the blockwise and channelwise pruning action for robotic detection tasks through DRL, pruning both residual blocks and channels automatically. A joint sample algorithm was 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 data sets, the results indicated that our method outperforms the traditional rule-based pruning methods with better accuracy and a higher compression ratio.

REFERENCES

[1] P. Song, W. Zheng, Y. Yu, and S. Ou, “Speech emotion recognition based on robust discriminative sparse regression,” IEEE Trans. Cogn. Dev. Syst., vol. 13, no. 2, pp. 343–353, Jun. 2021.
[2] Y. Lu, Y. Chen, D. Zhao, B. Liu, Z. Lai, and J. Chen, “CNN-G: Convolutional neural network combined with graph for image segmentation with theoretical analysis,” IEEE Trans. Cogn. Dev. Syst., vol. 13, no. 3, pp. 631–644, Sep. 2021.
[3] D. Zhao, Y. Chen, and L. Lv, “Deep reinforcement learning with visual attention for vehicle classification,” IEEE Trans. Cogn. Dev. Syst., vol. 9, no. 4, pp. 356–367, Dec. 2017.
[4] J. Wang, Q. Zhang, and D. Zhao, “Highway lane change decision-making via attention-based deep reinforcement learning,” IEEE/CACJ A. Automatica Sinica, vol. 9, no. 3, pp. 567–569, Mar. 2022.
[5] X. Zhao, Y. Chen, J. Guo, and D. Zhao, “A spatial-temporal attention model for human trajectory prediction,” IEEE/CACJ A. Automatica Sinica, vol. 7, no. 4, pp. 965–974, Jul. 2020.
[6] Y. Wang, Y. Xu, R. Yan, and H. Tang, “Deep spiking neural networks with binary weights for object recognition,” IEEE Trans. Cogn. Dev. Syst., vol. 13, no. 3, pp. 514–523, Sep. 2021.
[7] E. Lv, Y. Cheng, X. Wang, and C. L. P. Chen, “Deep convolutional network based on interleaved fusion group,” IEEE Trans. Cogn. Dev. Syst., vol. 13, no. 3, pp. 555–565, Sep. 2021.
[8] S. Liu, L. Liu, and Y. Yi, “Quantized reservoir computing for spectrum sensing with knowledge distillation,” IEEE Trans. Cogn. Dev. Syst., early access, Jan. 31, 2022, doi: 10.1109/TCDS.2021.3147789.
[9] Y. LeCun, J. S. Denker, and S. A. Solla, “Optimal brain damage,” in Proc. Adv. Neural Inf. Process. Syst., 1990, pp. 598–605.
[10] H. Li, A. Kadav, I. Durdaranovic, H. Samet, and H. P. Graf, “Pruning filters for efficient convnets,” in Proc. ICLR, 2016, pp. 1–13.
[11] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, “Learning efficient convolutional networks through network slimming,” in Proc. IEEE/CVF ICCV, 2017, pp. 2736–2744.
[12] Z. Huang and N. Wang, “Data-driven sparse structure selection for deep neural networks,” in Proc. ECCV, 2018, pp. 304–320.
[13] 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 Proc. Proc. 25th ICPR, 2021, pp. 5107–5114.
[14] P. Zhang, Y. Zhong, and X. Li, “SlimYOLOv3: Narrower, faster and better for real-time UAV applications,” in Proc. IEEE/CVF ICCV Workshops, 2019, pp. 37–45.
[15] B. Baker, O. Gupta, N. Naik, and R. Raskar, “Designing neural network architectures using reinforcement learning,” in Proc. ICLR, 2016, pp. 1–18.
[16] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, “Efficient neural architecture search via parameters sharing,” in Proc. ICML, 2018, pp. 4095–4104.
[17] 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 Proc. ECCV, 2018, pp. 784–800.
[18] 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 Proc. AAAI Conf. Artif. Intell., vol. 34, 2020, pp. 4887–4883.
[19] Z. Liu et al., “MetaPruning: Meta learning for automatic neural network pruning channel pruning,” in Proc. IEEE/CVF ICCV, 2019, pp. 3396–3405.
[20] M. Lin, R. Ji, Y. Zhang, B. Zhang, Y. Wu, and Y. Tian, “Channel pruning via automatic structure search,” in Proc. IJCAI, 2020, pp. 673–679.
[21] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” Mach. Learn., vol. 8, no. 3, pp. 229–256, 1992.
[22] J. Redmon and A. Farhadi, “YOLOv3: An incremental improvement,” 2018, arXiv:1804.02767.
[23] Y. He, X. Dong, G. Kang, Y. Fu, C. Yan, and Y. Yang, “Asymmetric soft filter pruning for deep convolutional neural networks,” IEEE Trans. Cybern., vol. 50, no. 8, pp. 3594–3604, Aug. 2020.
[24] Z. Ding, Y. Chen, N. Li, D. Zhao, Z. Sun, and C. L. P. Chen, “BNAS: Efficient neural architecture search using broad scalable architecture,” IEEE Trans. Neural. Netw. Learn. Syst., vol. 33, no. 9, pp. 5004–5018, Sep. 2022.
[25] Y. Chen, R. Gao, F. Liu, and D. Zhao, “ModuleNet: Knowledge-inherited neural architecture search,” IEEE Trans. Cybern., vol. 52, no. 11, pp. 11661–11671, Nov. 2022.
[26] Z. Ding, Y. Chen, N. Li, and D. Zhao, “BNAS-v2: Memory-efficient and performance-collapse-prevented broad neural architecture search,” IEEE Trans. Syst., Man, Cybern., Syst., vol. 52, no. 10, pp. 6259–6272, Oct. 2022.
[27] S. Guo, Y. Wang, Q. Li, and J. Yan, “DMCP: Differentiable markov channel pruning for neural networks,” in Proc. IEEE/CVF CVPR, 2020, pp. 1539–1547.
[28] Y. Liu, Y. Guo, J. Guo, L. Jiang, and J. Chen, “Conditional automated channel pruning for deep neural networks,” IEEE Signal Process. Lett., vol. 28, pp. 1275–1279, Jun. 2021.
[29] L. Lin, Y. Yang, and Z. Guo, “AACP: Model compression by accurate and automatic channel pruning,” 2021, arXiv:2102.00390.
[30] B. Li, B. Wu, J. Su, and G. Wang, “EagleEye: Fast sub-net evaluation for efficient neural network pruning,” in Proc. ECCV, 2020, pp. 639–654.
[31] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural Comput., vol. 9, no. 8, pp. 1735–1780, 1997.
[32] J. Ye, X. Lu, Z. Lin, and J. Z. Wang, “Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers,” in Proc. ICLR, 2018, pp. 1–11.
[33] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization.”
[34] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal speed and accuracy of object detection,” 2020, arXiv:2004.10934.
[35] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The PASCAL visual object classes (VOC) challenge,” Int. J. Comput. Vis., vol. 88, no. 2, pp. 303–338, 2010.
[36] T.-Y. Lin et al., “Microsoft COCO: Common objects in context,” in Proc. ICCV, 2014, pp. 740–755.
[37] H. Li, Z. Duan, J. Li, M. Ma, Y. Chen, and D. Zhao, “Neurons perception dataset for RoboMaster AI challenge,” in Proc. IJCNN, 2022, pp. 1–8.
[38] J. Li. “Detection-datasets-for-ABCP.” 2021. [Online]. Available: https://github.com/DRL-CASIA/Detection-Datasets-for-ABCP
[39] A. B. Chan and N. Vasconcelos, “Modeling, clustering, and segmenting video with mixtures of dynamic textures,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 30, no. 5, pp. 909–926, May 2008.
[40] J. M. Cherry et al., “SGD: Saccharomyces genome database,” Nucleic Acids Res., vol. 26, no. 1, pp. 73–79, 1998.
[41] A. Neubeck and L. Van Gool, “Efficient non-maximum suppression,” in Proc. 18th ICPR, vol. 3, 2006, pp. 850–855.
[42] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, “Once for all: Train one network and specialize it for efficient deployment,” in Proc. ICLR, 2020, pp. 1-15.
[43] J. Yu, L. Yang, N. Xu, J. Yang, and T. Huang, “Slimmable neural networks,” in Proc. ICLR, 2019, pp. 1–12.

Jiaqi Li received the master’s degree from the School of Mechanical Engineering, Beijing Institute of Technology, Beijing, China, in 2022. She has been an Intern with the Institute of Automation, Chinese Academy of Sciences, Beijing, China. Her research interests include deep learning, computer vision, model compression, and reinforcement learning.

Haoran Li (Member, IEEE) received the B.S. degree in detection, guidance, and control technology from Central South University, Changsha, China, in 2015, and the Ph.D. degree in control theory and control engineering from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2020. He is currently working as an Associate Professor with the Institute of Automation, Chinese Academy of Sciences, and also with the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing. His research interests include deep learning, deep reinforcement learning, and autonomous driving.

Yaran Chen (Member, IEEE) received the Ph.D. degree from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2018. She is currently an Associate Professor with the Institute of Automation, Chinese Academy of Sciences, and also with the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing. Her research interests include deep learning, neural architecture search, deep reinforcement learning, and autonomous driving.

Zixiang Ding (Graduate Student Member, IEEE) received the Ph.D. degree in computer applications from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, and the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing, China, in 2021. His research interests include knowledge distillation, neural architecture search, and deep reinforcement learning.

Nannan Li received the bachelor’s degree from the School of Automation Engineering, University of Electronic Science and Technology of China, Chengdu, Sichuan, China, in 2017. She is currently pursuing the Ph.D. degree with the Institute of Automation, Chinese Academy of Sciences, Beijing, China, and the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing. Her research interests include computer vision and neural architecture search.

Mingjun Ma (Graduate Student Member, IEEE) received the B.E. degree in information management and information system from the School of Information Resources Management, Renmin University, Beijing, China, in 2018. He is currently pursuing the master’s degree with the Institute of Automation, Chinese Academy of Sciences, Beijing. His main research interests include robot perception and control, and deep reinforcement learning.

Zicheng Duan received the B.Sc. degree from the University of Nottingham Ningbo China, Ningbo, China, in 2020. He is currently pursuing the master’s degree with the College of Engineering and Computer Science, The Australian National University, Canberra, ACT, Australia. He is currently focusing on multiview detection.

Dongbin Zhao (Fellow, IEEE) received the B.S., M.S., and Ph.D. degrees from Harbin Institute of Technology, Harbin, China, in 1994, 1996, and 2000, respectively. He is currently a Professor with the Institute of Automation, Chinese Academy of Sciences, Beijing, China, and also with the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing. He has published six books and over 100 international journal papers. His current research interests are in the area of deep reinforcement learning, computational intelligence, autonomous driving, game artificial intelligence, and robotics. Dr. Zhao serves as the Associate Editor of the IEEE TRANSACTIONS ON CYBERNETICS and IEEE TRANSACTIONS ON ARTIFICIAL INTELLIGENCE. He works as a guest editor of renowned international journals. He is involved in organizing many international conferences.