Deep Reinforcement Learning Based Multi-Task Automated Channel Pruning for DNNs

Xiaodong Ma
Beijing Key Lab of Traffic Data Analysis and Mining
School of Computer and Information Technology
Beijing Jiaotong University
Beijing, China
20120396@bjtu.edu.cn

Weiwei Fang
Beijing Key Lab of Traffic Data Analysis and Mining
School of Computer and Information Technology
Beijing Jiaotong University
Beijing, China
fangww@bjtu.edu.cn

***Abstract*—Model compression is a key technique that enables deploying Deep Neural Networks (DNNs) on Internet-of-Things (IoT) devices with constrained computing resources and limited power budgets. Channel pruning has become one of the representative compression approaches, but how to determine the compression ratio for different layers of a model still remains as a challenging task. Current automated pruning solutions address this issue by searching for an optimal strategy according to the target compression ratio. Nevertheless, when given a series of tasks with multiple compression ratios and different training datasets, these approaches have to carry out the pruning process repeatedly, which is inefficient and time-consuming. In this paper, we propose a Multi-Task Automated Channel Pruning (MTACP) framework, which can simultaneously generate a number of feasible compressed models satisfying different task demands for a target DNN model. To learn MTACP, the layer-by-layer multi-task channel pruning process is transformed into a Markov Decision Process (MDP), which seeks to solve a series of decision-making problems. Based on this MDP, we propose an actor-critic-based multi-task Reinforcement Learning (RL) algorithm to learn the optimal policy, working based on the IMPortance weighted Actor-Learner Architectures (IMPALA). IMPALA is known as a distributed RL architecture, in which the learner can learn from a set of actors that continuously generate trajectories of experience in their own environments. Extensive experiments on CIFAR10/100 and FLOWER102 datasets for MTACP demonstrate its unique capability for multi-task settings, as well as its superior performance over state-of-the-art solutions.**

***Index Terms*—Model Compression, Deep Neural Network, Deep Reinforcement Learning, Automated Pruning.**

I. Introduction

Deep Neural Networks (DNNs) have recently made remarkable achievements in many Deep Learning (DL) based intelligent tasks, e.g., face recognition, security surveillance and natural language processing [1]. However, the extraordinary performance of DNNs comes at a cost of high computational resources [2]. On the one hand, it usually takes a long time to train a task-specific DNN model from scratch. For example, the GPT-3 model has 175 billion parameters in total, and it takes one month to fully train it with 1024 NVIDIA A100 GPUs [3]. On the other hand, the increased model parameters, computational overhead and memory footprint make the DNN model difficult to be deployed on resource-constrained devices, e.g., Internet-of-Things (IoT). Taking Raspberry Pi 3B as an example, the time taken to perform for one inference ranges from several hundreds of milliseconds to tens of seconds, depending on the complexity of underlying model [4]. Thus, it is essential to develop innovative techniques that can tackle the contradiction between the increasingly complex DNN models and the stringently constrained computational resources.

To address these problems, model compression has become a promising technique to reduce DNN complexity and accelerate DNN inference [5]. Network pruning, as one of the representative compression approaches, aims to reduce computational cost and memory footprint by removing redundant structures and parameters from the DNN model. It can be classified as two categories, namely, unstructured-based and structured-based [2], [6]. Unstructured pruning allows to granularly prune the weights below a given threshold, which often achieves high compression rate with limited impact on inference accuracy [7]. However, it results in irregular sparsity in the weight matrix, which is difficult to be efficiently processed on common hardware [6]. In contrast, structured pruning removes groups of consecutive parameters, i.e., the entire channels/filters/sub-blocks of DNN weights, to form structured sparse patterns. It is more hardware friendly than unstructured pruning, and can achieve high acceleration performance by leveraging hardware parallelism. However, it is difficult to determine the specific compression ratio for different layers of a given DNN model, and this problem has been revealed as a combinatorial optimization problem [8]. By far, researchers have proposed some automated pruning approaches based on heuristic search [9] or Reinforcement Learning (RL) [10], to find the optimal compression ratio for each layer under a target compression ratio for the model.

However, there are limitations in the current solutions. Most of them only perform pruning for a specific compression ratio on a specific task dataset. In the real scenarios, the user may hope to have a tool to prune the model automatically for IoT devices with different resource constraints and training datasets with different contents. Taking the smart factory scenario as an example, a classic DNN model, e.g., VGG or MobileNet, can be used for detect defects for different types of products, such as printed fabric [11], wafer structure [12] and steel surface [13]. Meanwhile, the model may be

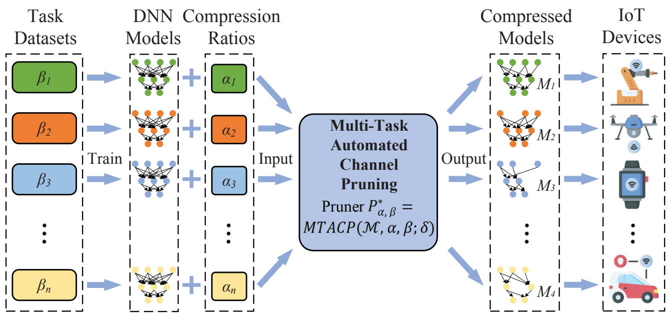


Fig. 1. Overview of the proposed multi-task automated channel pruning (MTACP) framework.

required to adapt to various kinds of computing hardware on site [4], [14], including Single Board Computer (SBC), Field Programmable Gate Arrays (FPGA), Micro Controller Unit (MCU), etc. In such cases, we usually have to repeat the pruning process according to each unique requirement, which is very inefficient and time-consuming [6], [7]. Thus, it is imperative to train an agent that can automatically and simultaneously generate multiple compressed models that correspond to different requirements on resource and task. However, to date, no research has conducted on pruning from this perspective.

In this paper, we propose MTACP, a new framework on multi-task automated channel pruning for resource-constrained IoT devices, as illustrated in Fig. 1. Given a DNN model, MTACP takes both target compression ratios and DL task datasets as the input conditions, and automatically and simultaneously outputs different pruned models that meet the resource constraints.

Specifically, MTACP incorporates the following contributions and innovations:

1) We present a MTACP, a new RL-based multi-task automated channel pruning framework, which can simultaneously output a number of feasible compressed models satisfying different demands on compression ratios and task datasets.
2) We formulate the multi-task channel pruning problem as a constrained optimization problem, which is then transformed into a Markov Decision Process (MDP). We propose a solution based on the IMPortance weighted Actor-Learner Architectures (IMPALA) [15], and optimize the whole searching process with some improvements.
3) We conduct extensive experiments with popular DNN models, including VGG, MobileNet, and MobileNetV2, on CIFAR10/100 and FLOWER102 datasets. The performance of our technique is compared with two state-of-the-art frameworks, namely AMC [10] and CACP [8], to demonstrate its superiority.

The rest of this paper is organized as follows. Section II briefly introduces relevant work. In Section III, we formulate the problem, transform it into a MDP, and develop a RL framework MTACP to learn the optimal policy. An extensive evaluation of our MTACP framework is given in Section IV. Finally, Section V concludes this paper, and discusses future works.

II. Related Work

A. DNN Pruning

The superior performance of DNNs is obtained at the cost of high consumption of computational resources. Model compression, which aims to reduce the model size and accelerate the computation at inference, has thus become an active research topic in recent years [2], [5]. One of the important model compression techniques is network pruning.

Early studies on pruning focus on exploiting the redundancy in the number of weights [7]. Nevertheless, such unstructured pruning results in irregular, sparse weight matrices that are not compatible with common computing hardware, e.g., GPUs and multi-core CPUs [6]. It has been reveal in [16] that the weight pruning has a very limited compression ratio for convolutional layers, and even leads to speed degradation on a commodity GPU. To address the limitations of unstructured pruning, three types of structured pruning have been introduced in [17], i.e., channel pruning, filter pruning, and filter shape pruning. They remove a number of channels [18], filters [19], or the weights at the same locations of all filters in a layer [17], respectively. Note that channel pruning and filter pruning are generally equivalent in essence. They not only have the superlinear effect on resource reduction [16], but also achieve high acceleration performance on CPU/GPU platforms [6], [19].

B. Deep Reinforcement Learning

Reinforcement Learning (RL) is an area of machine learning for experience-driven autonomous learning [20], and Deep Reinforcement Learning (DRL) combines RL with DL to handle high-dimensional problems [21]. Generally, there are three approaches to solve RL problems, based on value function, policy search and actor-critic, respectively.

Deep Q-Network (DQN) [22] is the most popular DRL model that receives high-dimensional inputs, and outputs a value function on the future rewards. Double DQN [23] is designed to use double estimators other than a single estimator in DQN to reduce overestimations and improve performance. Instead of maintaining a value function model, policy search approaches directly search for the optimal policy. The idea of updating policies with an estimator of the gradient of expected reward can date back to the REINFORCE algorithm [24]. TRPO [25] and PPO [26] are two important policy gradient algorithms. To estimate the right step size for policy updating, TRPO imposes a trust region constraint to the objective function to control KL divergence between consecutive policies, while PPO reduces the complexity of TPRO by using a first-order optimizer other than imposing this hard constraint.

The actor-critic method combines the benefits of the above two approaches together. The actor (policy) proposes a set of possible actions based on a state, while the critic (value function) evaluates these actions based on the given policy

[21]. DDPG [27] extends DQN to the continuous space with the actor-critic design while learning a deterministic policy. It inherits the shortcomings of DQN that risks overestimating the Q values in critic network, which finally results in inefficient learning or slow convergence when the estimated errors accumulate over time. Considering the drawbacks of experience replay in traditional algorithms like DDPG, A3C [28] proposes a parallel learning paradigm, in which multiple actors are trained asynchronously in parallel in their own environment and updated with global parameters from the centralized leaner whenever necessary. Unlike A3C, IMPALA [15] decouples acting from learning, by allowing the actors communicate trajectories of experience (i.e., sequences of states/actions/rewards) rather than gradients to a centralized learner. It proposes an off-policy actor-critic algorithm called V-trace, which allows actors to continue with trajectories when the learner asynchronously updates the policy. Furthermore, it is one of the few multi-task DRL solutions that can learn a set of closely related tasks concurrently [29].

C. AutoML

Popular DNN models are generally manually designed by a trail-and-error process, which requires substantial resource and time. Automated Machine Learning (AutoML) has emerged as a promising solution to automatically build DL systems without human intervention [30]. Two main techniques can server for such a purpose, namely Neural Architecture Search (NAS) and Automated Model Compression. As far as we know, RL has also played an important role in AutoML.

The two pioneer works in NAS are NAS-RL [31] and MetaQNN [32], both of which propose RL-based search strategies to reach the state-of-the-art accuracy on image classification tasks. NAS-RL [31] designs a RNN-based controller to generate a variable-length string that represents the DNN architecture, and then utilizes RL to optimize this controller. MetaQNN [32] models the selection of layer structure of a DNN as a MDP, and adjusts the selection behavior of agent by a Q-learning based search strategy. To reduce the size of search space, NASNet [33] only searches for the optimal structures for a few small cells (i.e., the small repeating structures in a DNN), and then stacks such multiple cells to form the final neural architecture. The aforementioned RL-based strategies treat NAS as a black-box optimization problem over a discrete search space, thus incurring high computational cost [34] (e.g., 2000 GPU days for NASNet [33]). Therefore, the gradient descent-based methods, e.g., DARTS [35], have recently been proposed to search for optimal architectures over a continuous search space in several GPU days. A comprehensive literature survey can be referred to [34] for more recent progress on NAS.

AMC [10] is the first work that introduces AutoML into model compression. It proposes a DDPG-based agent to search over a continuous pruning action space, penalize accuracy degradation and encourage model speedup. MetaPruning [36] proposes to train a PruningNet that can generate all candidate pruned models, and use evolutionary search to find optimal ones that satisfy resource constraints. However, both AMC and MetaPruning can only pruning for a specific compression ratio at a time. To improve the efficiency, CACP [8] is proposed to obtain a number of pruned models under different compression ratios in a single channel pruning process.

III. Multi-Task Automated Channel Pruning

In this section, we formulate the Multi-task channel pruning problem as a constrained optimization problem. Then, we transform this problem into a MDP. Based on this MDP, we propose a RL-based framework MTACP based on IMPALA to learn the optimal policy.

A. Problem Formulation

In this paper, we study the channel-based structured pruning that reduces the number of channels of each convolutional and fully connected layer. Given a L-layer DNN model $\mathcal{M}$, its channel structure can be denoted as $\mathcal{C} = (c_1, c_2, \ldots, c_L)$, where c_l ($l \in [1, L]$) is the total number of channels at layer l. Given a specific inference task represented by a compression ratio α and a dataset type β, the goal of channel pruning is to obtain the optimal pruning strategy $\mathcal{C}^* = (c_1^*, c_2^*, \ldots, c_L^*)$, and the pruned model $\mathcal{M}^*_{\alpha,\beta}$ that provides the highest inference accuracy $Acc(\mathcal{M}^*_{\alpha,\beta})$. Note that $\alpha \in (0, 100\%]$, and $\beta \in \mathcal{B}$, where $\mathcal{B}$ denotes a set of task datasets, e.g., CIFAR10, CIFAR100 and FLOWER102. We hope to obtain an automated pruner $P^*_{\alpha,\beta} = MTACP(\mathcal{M}, \alpha, \beta; \delta)$ that can satisfy different task demands on compression ratios α and dataset types β through a single search, where α and β follow some discrete uniform distribution $p(\cdot)$ and $q(\cdot)$, respectively. Beside, δ denotes a set of policy parameters. Following CACP [8], we can instead try to learn an optimal sampling policy $\pi_\delta(\cdot)$, which maximizes the long-term expectation of inference performance $Acc(\mathcal{M}^*_{\alpha,\beta})$ under the computational constraint $F(\mathcal{M}^*_{\alpha,\beta}) \leq \alpha F(\mathcal{M}_\beta)$, where $F(\mathcal{M}_\beta)$ denotes the computational cost of $\mathcal{M}_\beta$ measured in FLOPs [2], [8] on a given dataset β. Based on these models, we can formulate the optimization problem as follows:

$$\begin{aligned} \max_{\delta} \quad & \mathbb{E}_{\alpha \sim p(\cdot), \beta \sim q(\cdot)} \left[\mathbb{E}_{\mathcal{M}^*_{\alpha,\beta} \sim \pi_\delta(\mathcal{M},\alpha,\beta)} \left[Acc(\mathcal{M}^*_{\alpha,\beta}) \right] \right], \\ \text{s.t.} \quad & F(\mathcal{M}^*_{\alpha,\beta}) \leq \alpha F(\mathcal{M}_\beta), \alpha \in (0, 100\%], \beta \in \mathcal{B}. \end{aligned} \tag{1}$$

To solve this problem, the pruning process can be formally formulated as a MDP, which determines the compression ratio of each layer of $\mathcal{M}$ to prune the given model $\mathcal{M}$ to the target pruned model $\mathcal{M}^*_{\alpha,\beta}$.

B. Markov Decision Process for MTACP

As introduced in [20], a MDP can be defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{K}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ denotes a finite set of states, $\mathcal{A}$ denotes a finite set of actions, $\mathcal{K} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ denotes the state transition probability matrix, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denotes the reward function, and $\gamma \in [0, 1]$ denotes a discount factor. In order to obtain the optimal policy, it is necessary to identify the actions, states, and reward functions in the problem, which will be described as follows.

1) State space: To distinguish one convolutional layer from another, we carefully choose totally 13 important features for each layer l to characterize the state $s_l \in \mathcal{S}$,

$$(l, c, n, h, w, stride, k, \alpha, \beta, FLOPs[l], Reduced, Rest, a_{l-1}), \tag{2}$$

where for layer l, c and n denote the input and output channels, h and w denote the height and width of input feature map, $stride$ denotes the stride in convolution operation, k denotes the kernel size, $FLOPs[l]$ denotes the FLOPs of layer l, $Reduced$ and $Rest$ denote the reduced FLOPs for the layers $1 \sim (l-1)$ and the remaining FLOPs for the layers $(l+1) \sim L$, a_{l-1} denotes the action taken in layer $l-1$, respectively [2], [5]. Note that these state values should be normalized within $[0, 1]$ in the implementation [10].

2) Action space: We can choose either a coarse-grained discrete space or a fine-grained continuous space. If the former is selected, we may face a challenging problem that the valid action spaces of different states (i.e., the total number of channels at each layer) often have different sizes (e.g., 32, 64, 512, 1024). To avoid repeated sampling invalid actions due to such variations, recent RL algorithms generally integrate a technique called "invalid action masking", which masks out invalid actions while just sampling from those valid actions [37]. However, it is difficult to determine a proper number of discrete actions for efficient exploration [10], and may slow down the learning speed of the agent. Therefore, we select to use a continuous pruning action space $a_l \in (0, 100\%]$ for layer l, resulting in $c_l^* = \lceil a_l c_l \rceil$ channels after pruning.

3) Reward function: According to our formulation of problem (1), the goal is to maximize the accuracy of pruned model $\mathcal{M}_{\alpha,\beta}$. Since we can only conduct network pruning after L actions are all determined, we will receive the reward $r_L = Acc(\mathcal{M}_{\alpha,\beta})$ at the last layer, but no intermediate reward signal for other layers. This would incur the sparse reward problem [38], due to which the learning process would be slow down or even get stuck into some point [39]. We propose two solutions to this problem, i.e., by adding a Long Short-Term Memory (LSTM) network into the agent network, or by accumulating the rewards obtained by multiple consecutive samplings as a group and increasing the value of γ [40]. For the former, the memory-improved agent is capable to learn long-term dependencies for multiple decisions. For the latter, the agent is manually adjusted to pay more attention to long-term rewards in consecutive decisions.

However, the accuracy-based reward offers no incentive to the constraint on FLOPs reduction. Alternatively, we reduce the computational cost by limiting the action space [10]. That is, the arbitrary action output by the agent is allowed for the initial few layers, as long as the budget of FLOPs is sufficient. Once the budget can't hold even after the following layers are all compressed to the most extent, we turn to limit the action to prune the current layer more aggressively.

C. Policy Learning with Actor-Critic

Based on the modeled MDP, we can transform the problem in (1) into a problem of discounted infinite-horizon RL, in which an agent sequentially selects actions over a sequence of time steps T based on the stochastic policy π_δ, to maximize a cumulative reward. From the interaction between the policy and the MDP, we can obtain a trajectory of states, actions and rewards $s_1, a_1, r_1, \ldots, s_T, a_T, r_T$ over $\mathcal{S} \times \mathcal{A} \times \mathcal{R}$. The return (i.e., the sum of discounted future rewards) for state s_t is defined as $G_t = \sum_{i=t}^{T} \gamma^{i-t} r_i(s_i, a_i)$. Then, the state-value function and action-value function for policy π_δ can be defined as $V^{\pi_\delta}(s) = \mathbb{E}_{\pi_\delta}[G_1 | S_1 = s]$, and $Q^{\pi_\delta}(s, a) = \mathbb{E}_{\pi_\delta}[G_1 | S_1 = s, A_1 = a]$, respectively. The objective of agent is to learn and determine an optimal policy that maximizes G_t from the start state s_1, denoted by $U(\pi_\delta) = \mathbb{E}_{\pi_\delta}[G_1]$. Let ρ_{π_δ} denote the discounted state visitation distribution of policy π_δ [27], then this objective can be further expressed as an expectation [41]:

$$\begin{aligned} U(\pi_\delta) &= \int_{\mathcal{S}} \rho_{\pi_\delta}(s) \int_{\mathcal{A}} \pi_\delta(s, a) r(s, a) \mathrm{d}s\mathrm{d}a \\ &= \mathbb{E}_{s \sim \rho_{\pi_\delta}, a \sim \pi_\delta}[r(s_t, a_t)], \end{aligned} \tag{3}$$

To solve our MDP problem with a continuous pruning action space, the most suitable algorithm in RL is based on policy gradient that update policy parameters δ by taking steps in the direction of the gradient $\nabla_\delta U(\pi_\delta)$, which can be obtained by the policy gradient theorem [41]:

$$\begin{aligned} \nabla_\delta U(\pi_\delta) &= \int_{\mathcal{S}} \rho_{\pi_\delta}(s) \int_{\mathcal{A}} \nabla_\delta \pi_\delta(a|s) Q^{\pi_\delta}(s, a) \mathrm{d}s\mathrm{d}a \\ &= \mathbb{E}_{s_t \sim \rho_{\pi_\delta}, a_t \sim \pi_\delta}[\nabla_\delta \log \pi_\delta(a_t|s_t) Q^{\pi_\delta}(s_t, a_t)], \end{aligned} \tag{4}$$

The key issue is how to estimate the action-value function $Q^{\pi_\delta}(s, a)$. A straightforward way is to use Monte Carlo sampling, and treat the return G_t as unbiased sample of $Q^{\pi_\delta}(s, a)$ [24]. Although very simple, this approach suffers from high variance in the gradient estimates. To reduce the variance, the actor-critic architecture [21] is proposed as an efficient solution, which basically consists of two complementary modules. The actor adjusts the parameters δ of the stochastic policy π_δ by stochastic ascent or descent with estimates from the critic. The critic estimates a value function approximator $Q_\theta(s, a)$ (e.g., DDPG) or $V_\theta(s, a)$ (e.g., IMPALA) parameterized by θ, and use it instead of the unknown true $Q^{\pi_\delta}(s, a)$ or $V^{\pi_\delta}(s, a)$. The temporal-difference learning technique [20], which combines the advantages of Monte Carlo and Dynamic Programming, is often used to learn the critic.

However, the efficiency of actor-critic is constrained by its on-policy setting, i.e., the policy gradient is estimated from a trajectory of samples generated by only the current policy π_δ. When the policy is updated from ω to a new one π_δ, the old experience from ω will be thrown away. The training efficiency is severely affected for collecting enough qualified samples, especially in the cumbersome DNN pruning tasks. To address this issue, we introduce to use off-policy samples by the technique of importance sampling [26]. Because of the relative stability of gradient ascent in policy update, samples collected by one policy can be reused to perform updates for another policy with importance weighted return estimates [25]. Importance sampling allows us to avoid frequent sampling of fresh samples for policy update.

Given the trajectory of samples generated by the old policy ω, the policy gradient from (4) is modified to be:

$$\nabla_\delta U(\pi_\delta) = \mathbb{E}_{s_t \sim \rho_{\omega}, a_t \sim \omega}\left[\frac{\pi_\delta(a_t|s_t)}{\omega(a_t|s_t)} Q^\omega(s_t, a_t) \nabla_\delta \log \pi_\delta(a_t|s_t)\right], \quad (5)$$

where the importance weight $\frac{\pi_\delta(a_t|s_t)}{\omega(a_t|s_t)}$ is usually truncated by some constant to reduce variance [42].

D. IMPALA-based Multi-Task Learning

MTACP aims to solve a large number of pruning tasks using a single DRL agent with a single set of parameters (defined in Section III-B), which demands high throughput in training. The basic actor-critic architecture is merely able to learn one task at a time, which is much less productive and will incur considerable training efforts [29]. Thus, we choose to build our MTACP based on a distributed agent architecture named IMPortance weighted Actor-Learner Architectures (IMPALA) [15].

As shown in Fig. 2, the IMPALA-based architecture consists of a number of actors that continuously generates trajectories of experience, and one or more learners that use the experiences collected from actors to learn the target policy π_δ. At the start of each trajectory, an actor updates its local policy ω to the latest policy π_δ from the learner, and will execute policy ω for n steps in its own environment. After that, the actor sends the trajectory $s_1, a_1, r_1, \ldots, s_n, a_n, r_n$ and the underlying policy distributions $\omega(a_t|s_t)$ to a shared queue of the learner. Based on batches of trajectories collected from these actors, the learner keeps on updating its policy π_δ. In this way, a set of closely related pruning tasks can be learned concurrently and distributedly by individual actors with the coordination of the learner. Such an actor-leaner architecture can not only improve the exploration efficiency of actors in a cooperative way, but also reduce the communication overhead between actors and learners as compared to gradient-sharing [15].

However, due to the off-policy nature of data consumed by the learner, the actor's local policy ω might lag behind the learner's policy π_δ, by several updates at the time of gradient updating. Such policy-lag would lead to inefficiency or even instabilities in the training. We can use importance sampling to correct this discrepancy. For the trajectory $(s_k, a_k, r_k)_{k=t}^{k=t+n}$ generated by an actor with some policy ω, the n-step return at state s_t is calculated as $G_t = V^\omega(s_t) + \sum_{k=t}^{t+n-1} \gamma^{k-t}\epsilon_k$, where $\epsilon_k = r_k + \gamma V^\omega(s_{k+1}) - V^\omega(s_k)$ denotes a temporal difference (i.e., TD-error) for V^ω. As stated in Section III-C, we apply the truncated importance sampling corrections to the TD-error, leading to the V-trace return as follows:

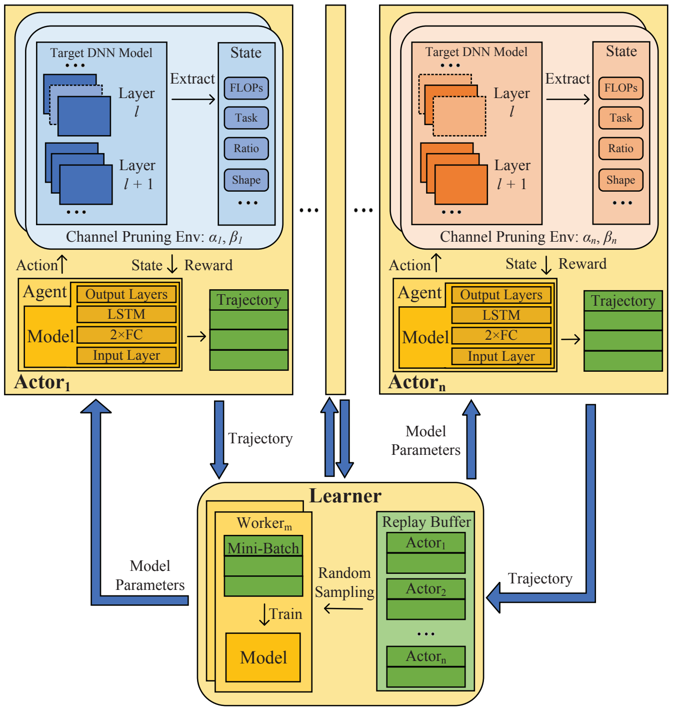


Fig. 2. The architecture of proposed DRL implementation for multi-task channel pruning.

$$G_t^{V-trace} = V^\omega(s_t) + \sum_{k=t}^{t+n-1} \gamma^{k-t} \left(\prod_{i=t}^{k-1} c_i\right) \psi_k \epsilon_k, \quad (6)$$

where $c_i = \min(\bar{c}, \frac{\pi_\delta(a_i|s_i)}{\omega(a_i|s_i)})$, and $\psi_k = \min(\bar{\psi}, \frac{\pi_\delta(a_k|s_k)}{\omega(a_k|s_k)})$ are truncated importance sampling weights, and $0 < \bar{c} \leq \bar{\psi} < \infty$. Note that $\bar{\psi}$ affects the value function the algorithm converge to, and $\bar{c}$ affects the convergence speed. From (6), we can compute the V-trace recursively by $G_t^{V-trace} = V^\omega(s_t) + \psi_t\epsilon_t + \gamma c_t(G_{t+1}^{V-trace} - V^\omega(s_{t+1}))$, which can be used to estimate the value of $Q^\omega(s_t, a_t)$ in (5), as $Q^\omega(s_t, a_t) \approx r_t + \gamma G_{t+1}^{V-trace}$.

At training time t, we update the critic with value parameters θ by gradient descent, in the direction of:

$$(G_t^{V-trace} - V_\theta(s_t))\nabla_\theta V_\theta(s_t), \quad (7)$$

where θ are adjusted to reduce the difference between $V_\theta(s_t)$ and $G_t^{V-trace}$. Meanwhile, we update the actor with policy parameters δ by policy gradient, in the direction of:

$$\psi_t \nabla_\delta \log \pi_\delta(a_t|s_t)(r_t + \gamma G_{t+1}^{V-trace} - V_\theta(s_t)), \quad (8)$$

where $V_\theta(s_t)$ is subtracted to reduce the variance of estimate in policy gradient [15]. The whole training process for the actors and the learner is shown in Algorithm 1 and 2.

E. Implementation Details

Given a state s, the agent determines its actions according to a continuous probability distribution over the action space

Algorithm 1 Training of an actor i

```
1: while $t <$ total training time slots for all tasks do
2:   for $l = 1, 2, ..., L$ do
3:     Get a state $s_{i,l}$ from environment $E_i$;
4:     Sample action $a_{i,l}$ from the local policy $\omega$;
5:     Execute action $a_{i,l}$ in $E_i$, and get reward $r_{i,l}$;
6:     if accumulate experience of $n$ steps then
7:       Send the trajectory $\{s_{i,\tau}, a_{i,\tau}, r_{i,\tau}\}_{\tau=t}^{\tau=t+n}$ with
         $\{\omega_\tau\}_{\tau=t}^{\tau=t+n}$ to the mini-batch queue of learner;
8:     end if
9:     $t \leftarrow t + 1$;
10:  end for
11: end while
```

Algorithm 2 Training of the learner

```
1: Receive a mini-batch of trajectories from actors, e.g., $i$
   with $\{s_{i,\tau}, a_{i,\tau}, r_{i,\tau}\}_{\tau=t}^{\tau=t+n}$ and local policy $\{\omega_\tau\}_{\tau=t}^{\tau=t+n}$);
2: Compute V-trace return $\{G_{i,\tau}^{V-trace}\}_{\tau=t}^{\tau=t+n}$ by (6);
3: Update the critic network $V_\theta$ in the direction of (7);
4: Update the policy network $\pi_\delta$ in the direction of (8);
5: Remote call to update each actor's local policy $\omega$ with $\delta$.
```

$\pi(a|s)$. The policy is modeled as a Gaussian distribution $\pi(\cdot|s) = \mathcal{N}(\mu, \sigma^2)$, where $\mu \in [0,1]$ and $\sigma^2 \in \mathbb{R}_+$ are determined by the state s. To achieve this, the agent model is designed to contain three main parts, as shown in Fig. 3. First, three fully connected layers (with 100, 400 and 300 neurons respectively), each of which is followed by a ReLU layer, are stacked sequentially to extract features from the input state s_l [8]. Then, the extracted feature is concatenated with the action a_{l-1} and the reword r_{l-1} of last layer, and sent to a LSTM recurrent layer with a memory cell size of 302. Finally, the output of LSTM is processed by a "multi-headed" architecture with three output layers [43]. Specifically, a linear layer with a sigmoid activation is used to obtain $\mu_l \in [0,1]$, a linear layer with a softplus activation is used to obtain $\sigma_l^2 \in (0, +\infty)$, and a linear layer is used to obtain the state value $V_l \in \mathbb{R}$ for model update. With the μ_l and σ_l^2, we randomly sample the action a_l from the policy $\pi_l(\cdot|s_l)$, and then clip it to $(0, 100\%]$ using the tanh activation as $\frac{\tanh(a_l)+1}{2}$. The optimal pruned models are finally obtained when the compression ratios converge to the target ones.

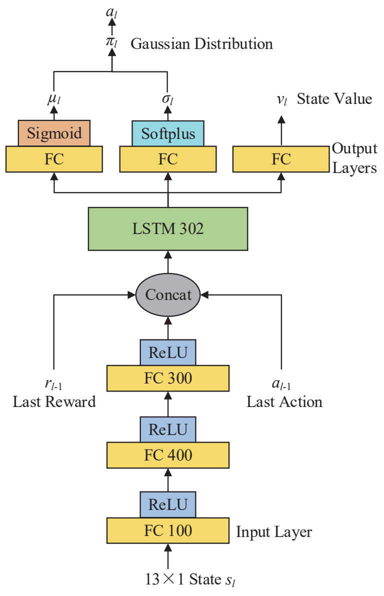


Fig. 3. The agent model architecture.

IV. Performance Evaluation

A. Experiment Settings

1) Models: In the experiments, we select three representative DNN models, i.e., VGG16 (VGG), MobileNet (MN), and MobileNetV2 (MN2), with different computation complexity, parameter size, and inference accuracy for image classification [2]. VGG16 [44] is a typical over-parameterized DNN model with a stack of regular convolutional layers, and often chosen by existing works, e.g., [6]–[8], to verify the performance of pruning algorithms. Both MobileNet [45] and MobileNetV2 [46] are specifically designed for resource-constrained devices (e.g., IoT), by replacing the regular convolution with the so-called depthwise separable convolution. MobileNetV2 is similar to the original MobileNet, except that it adopts inverted residual blocks with bottlenecking features to achieve less computation and fewer parameters.

2) Datasets: We perform experiments on three popular image datasets, namely CIFAR10 (C10), CIFAR100 (C100) and FLOWER102 (F102), which are image classification datasets widely used in the computer vision community [8], [47]. The CIFAR10 dataset has 60K 32×32 color images (50K for training and 10K for testing) in 10 classes, with 6K images per class. The CIFAR100 dataset has 20 super-classes and 100 classes, and each class contains 600 images (500 for training and 100 for for testing). The Flower-102 dataset has 102 flower categories, each of which consists of $40 \sim 258$ images. These flower images have large variations in light, pose and scale.

3) Hyperparameters: The experiments are conducted on a computing server equipped with GeForce RTX 2080Ti GPU using PyTorch. The total number of learners is 2, and the total number of actors is same as that of tasks. We set the total number of time slots to train all tasks as 3×10^4, batch size as 4, sequence length as 60, learning rate as 4.8×10^{-4}. The optimizer RMSProp is used for gradient optimization and model training, with decay as 0.99, epsilon as 0.01, momentum as 0.5, and gradient clipping as 40 (to avoid gradient explosion). Other hyper-parameters are set as $\lambda = 0.99$, $\overline{c} = \overline{\psi} = 1$, and $a_l \in [20\%, 100\%]$.

B. Experimental Results

1) Effectiveness of MTACP: In this set of experiments, we verify the effectiveness of MTACP by pruning each of the three DNN models with four typical task settings. The results are shown in Table I. In all cases, MTACP always guarantees the convergence to optimum subject to the FLOPs constraints in

TABLE I
THE PRUNING RESULTS OF MTACP FOR MULTI-TASKS.

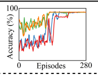
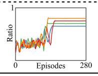
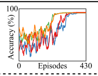
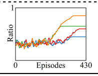
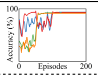
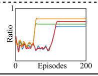
Model	β	α	**Acc. (%)**	**FLOPs (M)**	**Params. (M)**	**Process**
VGG	C10	1.00	92.72	313.20	14.72	
		0.70	92.44	210.70	3.91	
		0.80	92.74	240.50	5.48	
	F102	1.00	93.05	313.25	14.77	
		0.65	92.95	195.66	2.68	
		0.75	92.99	225.60	4.85	
MN	C10	1.00	92.32	11.60	3.29	
		0.65	92.74	7.47	1.09	
		0.85	92.30	9.69	2.48	
	F102	1.00	93.79	567.82	3.29	
		0.45	93.06	252.69	1.23	
		0.60	93.78	331.39	1.91	
MN2	C10	1.00	92.48	6.12	2.20	
		0.70	92.62	4.34	0.80	
		0.80	91.64	4.95	1.17	
	F102	1.00	93.52	299.62	2.04	
		0.65	93.83	195.37	1.41	
		0.75	93.38	225.28	1.54	

TABLE II
THE COMPARISON RESULTS BETWEEN AUTOMATED PRUNING METHODS.

Model	**Method**	**Metrics**	β				**Overall Time (min)**	
			C10		**F102**		**Serial**	**Parallel**
			α					
			0.60	0.70	0.60	0.70		
VGG	AMC	**Acc. (%)**	92.34	92.56	93.36	93.00	╲	╲
		Time (min)	201.36	206.35	210.70	212.23	**830.64**	**212.23**
	CACP	**Acc. (%)**	92.78	92.04	93.55	93.10	╲	╲
		Time (min)	170.51	182.94	180.14	175.93	**363.08**	**182.94**
	MTACP	**Acc. (%)**	92.36	92.54	93.05	93.20	╲	╲
		Time (min)	67.82	70.34	72.00	75.43	-	**75.43**
MN	AMC	**Acc. (%)**	92.12	92.90	93.45	93.25	╲	╲
		Time (min)	70.37	79.36	79.55	80.13	**309.41**	**80.13**
	CACP	**Acc. (%)**	92.56	92.28	93.08	93.12	╲	╲
		Time (min)	103.76	101.23	106.41	102.55	**210.17**	**106.41**
	MTACP	**Acc. (%)**	92.74	92.45	93.86	93.78	╲	╲
		Time (min)	46.25	45.37	47.08	42.38	-	**47.08**
MN2	AMC	**Acc. (%)**	91.54	92.52	93.39	93.24	╲	╲
		Time (min)	153.33	151.87	145.32	147.23	**597.75**	**153.33**
	CACP	**Acc. (%)**	92.38	92.53	93.22	93.27	╲	╲
		Time (min)	114.76	104.74	116.23	107.75	**230.99**	**116.23**
	MTACP	**Acc. (%)**	92.43	92.86	93.37	93.40	╲	╲
		Time (min)	55.43	59.80	50.45	50.61	-	**59.80**

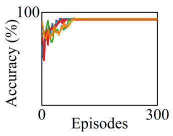
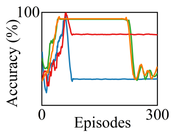
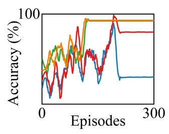
a few hundreds of running episodes. The inference accuracy can be well maintained (with the loss less than 1%), while the model redundancy is removed. It is worth mentioning that in some cases channel pruning can even improve the accuracy, as the risk of model overfitting is partially mitigated by removing unimportant channels [18]. Furthermore, since the fully connected layers that dominate the model size are also pruned in MTCAP like the convolutional layers, we can notice the sharp shrinkage of parameter size for all models after pruning. This will facilitate the deployment of these models on storage-limited IoT devices.

2) Comparison with Automated Pruning Methods: In this set of experiments, we compare MTACP with two state-of-the-art automated channel pruning approaches, i.e., AMC [10] and CACP [8]. The results are shown in Table II, from which we can make three main observations. First, these three schemes offer comparable inference accuracy, as the accuracy differences are generally small or even neglectable. Thus, all of them are qualified for automated pruning. Second, AMC and CACP are not designed for multi-task settings. In one search, AMC can only prune according to a given pair of (α, β), while CACP can prune for different α at the same time according to the given β. Thus, it generally takes longer searching time for these two schemes. Third, even multiple pruning tasks can be executed in parallel, we can still observe that MTACP has the shortest searching time among the three. Generally, the total searching time of MTACP is about $1.70\times \sim 2.81\times$ less than AMC, and about $1.94\times \sim 2.43\times$ less than CACP. In all, these results clearly demonstrate the high efficiency of our proposed MTACP, and its advantage over AMC and CACP.

3) Comparison with Multi-task DRL Algorithms: In this set of experiments, we compare MTACP with two basic DRL algorithms, i.e., PPO [26] and IMPALA [15], for multiple pruning tasks on MobileNet. As shown in Table III, neither PPO nor IMPALA can converge to the optimum, thus incurring severe accuracy degradation. As compared to PPO, IMPALA and MTACP use the V-trace to handle the policy-lag problem led by task diversity, making them more suitable for the multi-task learning. However, the classic IMPALA implementation still can't converge to a satisfactory accuracy level. To address this issue, MTACP has made several improvements, including the solving of sparse reward problem, the limitation of action space and the specialized designing of agent model. The results in Table III verify the necessity of these designs.

4) Impact of task settings to MTACP: In this set of experiments, we investigate the performance variation of MTACP under different number of tasks and different data diversity. To this end, we use the CIFAR100 dataset instead, and divide it into 20 subsets according to the pre-given super-classes. Then, we conduct experiments on MobileNet using 3, 6, 9, 12, and 15 subsets among them, with different compression ratios (i.e., 0.3, 0.4, 0.5, 0.6 and 0.7), respectively. We also carry out the 15 pruning tasks individually to provide a comparative reference. The results are shown in Fig. 4 - Fig. 9. In these figures, each task is represented by a colored shape, and each shape denotes a pre-set compression ratio. The horizontal red dash line denotes the baseline accuracy of 93.65%, and the vertical red dash line denotes the average searching time.

TABLE III
THE PRUNING RESULTS OF AMC AND CACP FOR MULTI-TASKS.

Algorithm	β	α	**Acc. (%)**	**FLOPs (M)**	**Process**
MTACP	C10	1.00	93.24	11.60	
		0.70	**92.46**	8.12	
		0.80	**93.18**	9.28	
	F102	1.00	93.90	567.82	
		0.65	**93.10**	370.11	
		0.75	**92.00**	425.96	
PPO	C10	1.00	92.32	11.60	
		0.70	**29.28**	8.02	
		0.80	**41.18**	9.14	
	F102	1.00	92.76	555.94	
		0.65	**28.23**	365.95	
		0.75	**75.53**	420.67	
IMPALA	C10	1.00	93.02	11.60	
		0.70	92.38	8.02	
		0.80	92.26	9.12	
	F102	1.00	93.79	555.94	
		0.65	**30.43**	365.95	
		0.75	**80.23**	420.67	

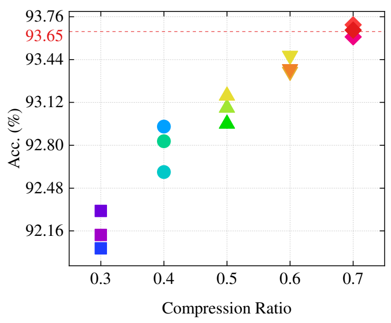


Fig. 4. Fifteen individual pruning tasks with 5 compression ratios.

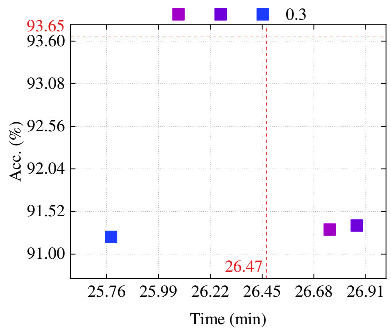


Fig. 5. Three pruning tasks with the compression ratio of 0.3.

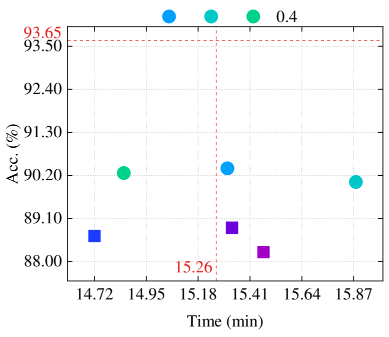


Fig. 6. Six pruning tasks with the compression ratios of 0.3 and 0.4.

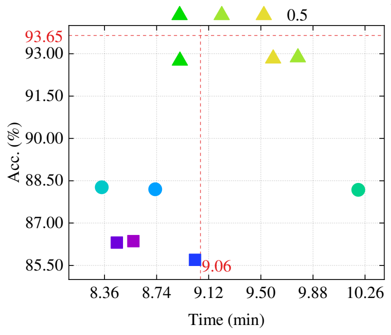


Fig. 7. Nine pruning tasks with the compression ratios of 0.3, 0.4 and 0.5.

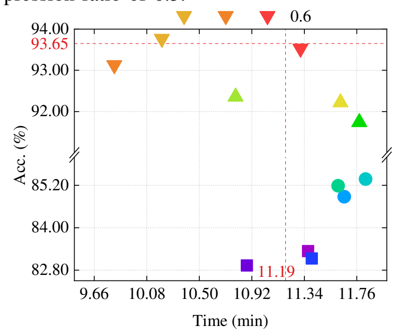


Fig. 8. Twelve pruning tasks with the compression ratios of 0.3, 0.4, 0.5 and 0.6.

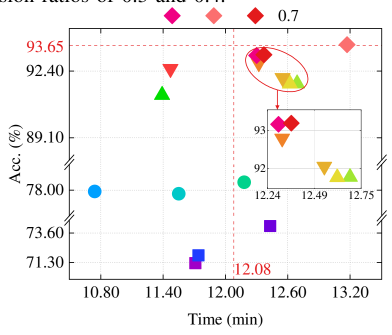


Fig. 9. Fifteen pruning tasks with the compression ratios of 0.3, 0.4, 0.5, 0.6 and 0.7.

From the perspective of time cost, we can observe that the average searching time first decreases from 26.47 to 9.06 minutes, and then increases to 12.08 minutes. That's because the accumulation of experiences in DRL is generally slow when there are insufficient number of tasks. After that, the duration for searching will be more affected by the total number of tasks. From the perspective of inference accuracy, it can be find that the accuracy gradually decreases with the addition of new tasks, especially for the tasks with high compression ratios (e.g., $\alpha = 0.3$). The results reveal that MTACP is sensitive to both the number of pruning tasks and the diversity of task data. Note that this is not just the limitation of MTACP, since even human expertise or current automated pruning solutions [8], [10] still face similar problems in reality.

V. Conclusion and Future Work

Current pruning-based DNN compression techniques generally rely on expertise and hand-crafted features to explore a large optimization space, which are often sub-optimal and laborious. In this paper, we have proposed a new multi-task automated channel pruning framework, MTACP, to obtain the pruned models under multiple compression ratios and different training datasets in one shot. The problem is transformed into a MDP, and then solved by a RL algorithm based on IMPALA. To demonstrate the efficacy and efficiency of MTACP, we applied it to several widely-used DNN models, and compare it with some SOTA solutions on automated pruning.

In the future, we plan to evaluate MTACP with more typical datasets (e.g., ImageNet [10]) and DNN models with more complex architectures (e.g., GoogLeNet and DenseNet [2]). Another important issue is to make it more scale as effectively to a large number of diverse tasks with the help of new RL mechanisms, e.g., PopArt [29].

Acknowledgment

This work was supported in part by the National Science Foundation of China (NSFC) under Grant 62172031, the Beijing Municipal Natural Science Foundation under Grant L191019. Weiwei Fang is the corresponding author.

References

[1] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen, and S. S. Iyengar, "A survey on deep learning: Algorithms, techniques, and applications," *ACM Computing Surveys (CSUR)*, vol. 51, no. 5, pp. 1–36, 2018.

[2] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec 2017.

[3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[4] R. Hadidi, J. Cao, Y. Xie, B. Asgari, T. Krishna, and H. Kim, "Characterizing the deployment of deep neural networks on commercial edge devices," in *2019 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2019, pp. 35–48.

[5] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[6] X. Ma, F.-M. Guo, W. Niu, X. Lin, J. Tang, K. Ma, B. Ren, and Y. Wang, "Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, pp. 5117–5124, Apr. 2020.

[7] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *arXiv preprint arXiv:1510.00149*, 2015.

[8] Y. Liu, Y. Guo, J. Guo, L. Jiang, and J. Chen, “Conditional automated channel pruning for deep neural networks,” *IEEE Signal Processing Letters*, vol. 28, pp. 1275–1279, 2021.

[9] M. Lin, R. Ji, Y. Zhang, B. Zhang, Y. Wu, and Y. Tian, “Channel pruning via automatic structure search,” in *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, 2021, pp. 673–679.

[10] 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 European conference on computer vision (ECCV)*, 2018, pp. 784–800.

[11] S. Chakraborty, M. Moore, and L. Parrillo-Chapman, “Automatic printed fabric defect detection based on image classification using modified vgg network,” in *International Conference on Applied Human Factors and Ergonomics*. Springer, 2021, pp. 384–393.

[12] S.-K. S. Fan, C.-Y. Hsu, C.-H. Jen, K.-L. Chen, and L.-T. Juan, “Defective wafer detection using a denoising autoencoder for semiconductor manufacturing processes,” *Advanced Engineering Informatics*, vol. 46, p. 101166, 2020.

[13] Y. He, K. Song, Q. Meng, and Y. Yan, “An end-to-end steel surface defect detection approach via fusing multiple hierarchical features,” *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 4, pp. 1493–1504, 2019.

[14] T. Sipola, J. Alatalo, T. Kokkonen, and M. Rantonen, “Artificial intelligence in the iot era: A review of edge ai hardware and software,” in *2022 31st Conference of Open Innovations Association (FRUCT)*. IEEE, 2022, pp. 320–331.

[15] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning *et al.*, “Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures,” in *International Conference on Machine Learning*. PMLR, 2018, pp. 1407–1416.

[16] X. Ma, S. Lin, S. Ye, Z. He, L. Zhang, G. Yuan, S. H. Tan, Z. Li, D. Fan, X. Qian *et al.*, “Non-structured dnn weight pruning–is it beneficial in any platform?” *arXiv preprint arXiv:1907.02124*, 2019.

[17] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, “Learning structured sparsity in deep neural networks,” *Advances in neural information processing systems*, vol. 29, 2016.

[18] M. Liu, W. Fang, X. Ma, W. Xu, N. Xiong, and Y. Ding, “Channel pruning guided by spatial and channel attention for dnns in intelligent edge computing,” *Applied Soft Computing*, vol. 110, p. 107636, 2021.

[19] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient convnets,” in *Conference Track Proceedings on 5th International Conference on Learning Representations, ICLR 2017*, 2017.

[20] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[21] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, “Deep reinforcement learning: A brief survey,” *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[23] H. v. Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016, pp. 2094–2100.

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

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

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.

[27] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.

[28] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

[29] N. Vithayathil Varghese and Q. H. Mahmoud, “A survey of multi-task deep reinforcement learning,” *Electronics*, vol. 9, no. 9, 2020.

[30] S. K. Karmaker, M. M. Hassan, M. J. Smith, L. Xu, C. Zhai, and K. Veeramachaneni, “Automl to date and beyond: Challenges and opportunities,” *ACM Computing Surveys (CSUR)*, vol. 54, no. 8, pp. 1–36, 2021.

[31] B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning,” *arXiv preprint arXiv:1611.01578*, 2016.

[32] B. Baker, O. Gupta, N. Naik, and R. Raskar, “Designing neural network architectures using reinforcement learning,” *arXiv preprint arXiv:1611.02167*, 2016.

[33] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning transferable architectures for scalable image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[34] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, “A comprehensive survey of neural architecture search: Challenges and solutions,” *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, pp. 1–34, 2021.

[35] H. Liu, K. Simonyan, and Y. Yang, “Darts: Differentiable architecture search,” *arXiv preprint arXiv:1806.09055*, 2018.

[36] 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 international conference on computer vision*, 2019, pp. 3296–3305.

[37] D. Ye, Z. Liu, M. Sun, B. Shi, P. Zhao, H. Wu, H. Yu, S. Yang, X. Wu, Q. Guo *et al.*, “Mastering complex control in moba games with deep reinforcement learning,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 6672–6679.

[38] M. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. Wiele, V. Mnih, N. Heess, and J. T. Springenberg, “Learning by playing solving sparse reward tasks from scratch,” in *International conference on machine learning*. PMLR, 2018, pp. 4344–4353.

[39] G. Matheron, N. Perrin, and O. Sigaud, “The problem with ddpg: understanding failures in deterministic environments with sparse rewards,” *arXiv preprint arXiv:1911.11679*, 2019.

[40] K. Zhu and T. Zhang, “Deep reinforcement learning based mobile robot navigation: A review,” *Tsinghua Science and Technology*, vol. 26, no. 5, pp. 674–691, 2021.

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

[42] P. Wawrzyński, “Real-time reinforcement learning by sequential actor–critics and experience replay,” *Neural networks*, vol. 22, no. 10, pp. 1484–1497, 2009.

[43] G. Schamberg, M. Badgeley, B. Meschede-Krasa, O. Kwon, and E. N. Brown, “Continuous action deep reinforcement learning for propofol dosing during general anesthesia,” *Artificial Intelligence in Medicine*, vol. 123, p. 102227, 2022.

[44] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[45] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *arXiv preprint arXiv:1704.04861*, 2017.

[46] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[47] D. Lu, Y. Zhai, J. Wu, and J. Shen, “Treenet: A hierarchical deep learning model to facilitate edge intelligence for resource-constrained devices,” in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2021, pp. 525–534.