Boosting the Convergence of Reinforcement Learning-Based Auto-Pruning Using Historical Data

Jiandong Mu, Mengdi Wang, Feiwen Zhu, Jun Yang, Wei Lin, and Wei Zhang, Member, IEEE

Abstract—Recently, neural network compression schemes like channel pruning have been widely used to reduce the model size and computational complexity of deep neural networks (DNNs) for applications in power-constrained scenarios, such as embedded systems. Reinforcement learning (RL)-based auto-pruning has been further proposed to automate the DNN pruning process to avoid expensive hand-crafted work. However, the RL-based pruner involves a time-consuming training process, and pruning and evaluating each network comes at high-computational expense. These problems have greatly restricted the real-world application of RL-based auto-pruning. Thus, we propose an efficient auto-pruning framework that solves this problem by taking advantage of the historical data from the previous auto-pruning process. In our framework, we first boost the convergence of the RL-pruner by transfer learning. Then, an augmented transfer learning scheme is proposed to further speed up the training process by improving the transferability. Finally, an assistant learning process is proposed to improve the sample efficiency of the RL agent. The experiments show that our framework can accelerate the auto-pruning process by 1.5×–2.5× for ResNet20, and 1.81×–2.375× for other neural networks, such as ResNet56, ResNet18, and MobileNet v1.

Index Terms—Auto-pruning, deep neural network (DNN), reinforcement learning (RL).

I. INTRODUCTION

NOWADAYS, deep neural networks (DNNs) have become one of the most popular algorithms for their impressive performance in applications ranging from object detection and image classification to speech recognition. However, this high performance is at the expense of the large model size and huge computing complexity, which have prevented DNN from broader usage. As one of the most successful solutions, network pruning [1], has been proposed to slim DNN models to obtain a good tradeoff between accuracy and model size, making them feasible for power-hungry devices, such as mobile phones. A variety of methods [2], [3], [4], [5], [6] have been proposed to prune DNNs using different granularities and metrics. Channel pruning, which prunes the featuremap and weights in the channel granularity, is widely used due to its high efficiency in hardware implementation. In this work, we mainly focus on channel pruning; however, our framework can be easily extended to other pruning schemes.

In order to avoid the extra hand-crafted work introduced by the pruning process, and to explore a larger network pruning space, auto-pruning is proposed to compress the network by automatically generating the optimal pruning policy, i.e., the pruning/preservation ratio of each layer of the input DNN model, using a trainable agent. During the auto-pruning process, the agent is trained by the existing pruning policy and corresponding accuracy. With sufficient data and training time, the agent can converge and generate the optimal pruning policy for the input network. Liu et al. [7] presented a detailed comparison between auto channel pruning and uniform pruning. The experiments demonstrate that auto channel pruning is significantly more parameter-efficient, achieving the same accuracy with 5 times fewer parameters. It is difficult to make a fair comparison between pruning and network architecture search (NAS) since their design spaces and search methodologies are distinct. However, it should be noted that network pruning, as one of the NAS approaches, has a restricted search space of subnets within a larger network, which NAS does not have.

Among all the auto-pruning agents, the reinforcement learning (RL) [8]-based agent [9] has attracted great attention from researchers and developers due to its mature theoretical study, universality, and high performance. However, the training process of the RL-based pruner is time consuming for several reasons.

1) The sample efficiency is low since the RL agent is randomly initiated and may not exploit its knowledge of the environment to improve its performance until enough interaction data with the environment have been collected.

2) The computation time of each sample is high as it requires inferences for thousands of images to measure the accuracy and latency.

The high-time cost has greatly restricted the usage of RL-based auto-pruning. For example, it takes over one day to prune the ResNet18 automatically on four Nvidia 1080Ti. It will take much longer for more complex and practical networks like ResNet50, which makes auto-pruning prohibitive for industry, where thousands of pruning requests are received per day and time-to-market is critical.

In this work, we propose a comprehensive learning framework to boost the convergence of the RL agent using historical data. First, we resort to transfer learning to resolve the random initialization problem. The core idea of transfer learning, which was proposed in [10], is that experience gained in learning to perform one task can benefit the learning performance in related but different tasks. By transferring the knowledge from a source task to a target task, instead of learning from
scratch, the training time can be significantly reduced. In this work, transfer learning between different pruning ratios, DNN models, and datasets is investigated in detail to accelerate the auto pruning process for the first time. Furthermore, we realize that one obstacle of transfer learning between different pruning scenarios is the differences in the preservation ratio, network model, etc. Therefore, we propose network and data augmentation (DA) to enhance the transferability so that the converging time of the RL agent can be further reduced compared to vanilla transfer learning. Finally, we propose a novel assistant learning (AL) process to improve the data efficiency of the RL agent at the beginning of the learning process by generating the training samples according to the pruning history. In this way, the negative effects of the low performance in the initial training process can be minimized.

In summary, we make the following contributions.

1) We propose to speed up the auto-pruning process with transfer learning. Transfer learning across different pruning ratios, models, and datasets is discussed in a comprehensive manner.

2) We propose a novel augmented transfer learning scheme to enhance the transferability between different pruning scenarios, thereby further reducing the training time.

3) We propose a novel AL process to improve the data efficiency of the RL agent in the initial training stage.

4) Comprehensive experiments are conducted for the proposed pruning framework. The experiments show that the auto-pruning time can be reduced by $1.5 \times \sim 2.5 \times$ for ResNet20, and $1.81 \times \sim 2.375 \times$ for other networks like ResNet56, ResNet18, and MobileNet v1.

5) We also consider both accuracy and latency as the reward signal to show that our framework can be easily applied to other metrics. A novel flow to measure the latency accurately is also proposed.

The remainder of this article is organized as follows, Section II states the background and related works. Section III introduces the overall framework of this work. Sections IV and V present the proposed augmented transfer learning scheme and AL scheme, respectively, to find the optimal pruning policy. Section VII shows the experiments to validate our proposed learning framework, and Section VIII concludes this article.

II. BACKGROUND AND RELATED WORKS

In this section, we show the background and related works from the following three perspectives.

A. Channel Pruning

There has been a significant number of works on neural network compression to slim DNNs so that they can be computed efficiently without losing much accuracy. As one of the most important pruning methods, channel pruning reduces the computing complexity by removing the redundant channels on the featuremap. Many channel pruning schemes [2], [3], [11], [12] have been proposed to identify the redundant channels efficiently.

Zhuang et al. [3] pruned the channels by keeping those channels that really contribute to discriminative power. To achieve this, the author introduces additional losses into the network to increase the discriminative power of the intermediate layers, and then selects the most discriminative channels for each layer by considering the additional loss and the reconstruction error. A greedy algorithm to conduct channel selection and parameter optimization is also proposed. Liu et al. [11] considered the channel pruning as a design space exploration problem, and searches the optimal channel pruning network by an evolutionary procedure. The proposed DSE is efficient since accuracy of the pruned network can be achieved by generating the weights for the pruned network using the proposed pretrained model. In [12], an artificial bee colony (ABC) algorithm [13] is proposed to search for the optimal preservation ratios for each layer in channel pruning.

In our work, auto channel pruning is achieved by predicting the numbers of the channels preserved in each layer based on the RL agent. Then, the redundant channels are identified and pruned by minimizing the reconstruction error [2]. An $l_1$ regularization is applied to push the weights of the abandoned channels to zero, and an iterative algorithm is proposed to solve the corresponding Lasso problem [14] efficiently.

Although most of the channel pruning works put the focus on the pruning criteria, there are still a few of them pay the attention to accelerating channel pruning process. Fan et al. [15] used the clustering algorithm to reduce the hyperparameters in the channel pruning process to reduce the pruning time significantly. The author report that they have an acceleration of $4 \times$ on ResNet56. However, this article simplified the channel pruning problem by reducing the hyper parameters which may miss the optimal channel pruning policy. Wang et al. [16] proposed a retraining free pruning framework, which prune the network during the training process by considering the training as an optimization problem and solve it with the constraint of sparse norm. The author report that extensive experiments demonstrate that such a retraining-free pruning method can maintain the accuracy as the methods with retraining while achieving $3 \times$ speed up, however, limited to a compression ratio of 43.0%–88.4%.

B. Reinforcement-Learning-Based Auto-Pruning

RL-based auto-pruning was first proposed in [9]. In that work, a deep deterministic policy gradient (DDPG) agent [17], which is one of the most popular RL algorithms that targets the continuous action space, is utilized to automatically generate the action of each layer of the DNN model. The action refers to the preservation ratio; however, its pruning ratio counterpart can be processed in a similar way. Currently, the RL agent takes a long time to converge since it usually takes hundreds of trials to train the DDPG agent, and the reward, such as accuracy and latency, requires inferences of thousands of images in each trial. Therefore, it is of great importance to improve the convergence speed of the RL agent.

Several works have proposed to address this problem. In [18], previously collected offline data is employed to aid the online learning process by constraining the current policy to stay close to the policy in the previous data. However, this work relies on the assumption that the offline data has the same distribution as the online data, which may not hold true for auto-pruning across different pruning scenarios. Therefore, it cannot be applied in our case. Pertsch et al. [19] accelerated the RL agent by integrating the actions into skills, and learns a prior over the skills from the offline data. However, this work also suffers from the offline data consistency problem. Wen et al. [20] trained a predictor to accelerate the NAS process by predicting the accuracy of the network in architecture space. This approach can also be applied to auto
pruning by predicting the accuracy of different pruning strategies. Gupta et al. [21] proposed using a dense reward signal instead of the previous sparse reward signal to speed up the RL process by training the agent more intensively. However, these approaches still train the RL agent from scratch without properly utilizing the historical data.

In our work, we show that our proposed framework can solve this problem by bridging the source task and target task via network augmentation (NA) and DA. As a result, the auto-pruning process can be accelerated with historical data from other pruning scenarios.

C. Transfer Learning

As we have mentioned before, transfer learning can expedite the learning process of DNN models by taking advantage of the knowledge learned from other task and has been widely deployed in many works [22], [23], [24], [25], [26]. Recent works also use transfer learning in RL [27], [28] to accelerate the learning process by leveraging external expertise. However, to the best of our knowledge, there still lacks investigations into transfer learning in auto-pruning. Furthermore, vanilla transfer learning may suffer from significant performance degradation in several scenarios and become less valuable for practical usage due to the difference between the source task and the target task. In this work, we present for the first time a framework to speed up RL-based auto-pruning via transfer learning. Then, we further boost the performance of the transfer learning by network and DA.

III. FRAMEWORK

In this section, we will show the overview of the proposed framework, illustrated in Fig. 1, followed by a detailed introduction of each component.

As mentioned previously, transfer learning has been applied to RL [27], [28] to speed up the learning process. In this work, we apply transfer learning to the RL-based auto-pruning to boost the auto-pruning process for the first time. Furthermore, we aim to accelerate the transfer learning process by improving the transferability between different pruning scenarios and a novel AL process.

To achieve this goal, the pruning specifications (e.g., the network to be compressed, the target preservation ratio, etc.) and historical data are first imported to the source model selection module. Then, the source model of the transfer learning process will be selected according to the rules and the performance of the vanilla transfer learning process. If the time constraints are already satisfied, the RL agent can be directly accelerated by vanilla transfer learning. Otherwise, an augmented transfer learning process will be conducted to boost the transferability of the source model and data so that the converging time of the transfer learning can be further reduced.

Our proposed augmented transfer learning consists of DA and NA, which increases the transfer learning efficiency by augmenting the data in the replay buffer and the architecture of the actor neural network, respectively. Note that the corresponding historical model will also be updated to ease the later transfer learning process as the augmented model has a better transferability. As a result, the overhead of the model selection and augmentation process will become negligible as the historical data becomes comprehensive.

Then, AL is employed to improve the sampling efficiency in the RL process. This is based on the rationale that the RL agent suffers from low-sampling efficiency because the interactions with the environment in the initial phase of the training process are random [29]. Therefore, we can improve the sampling efficiency and boost the convergence of the RL algorithm by taking advantage of the historical data from other pruning scenarios.

Finally, channel pruning is conducted according to the pruning policy generated by the DDPG agent. The reward signal, which is the evaluation of the pruned network, will be fed back to the actor and critic network to guide the later training process until the optimal pruning policy can be obtained.

In this work, we assume that there is an underlying principle for the auto pruning process that the RL agent can learn and transfer to similar pruning scenarios. However, we recognize that this assumption may be weak since we have found that the optimal pruning policy for similar networks often shares many similarities, as mentioned by other works, such as [10] and [28]. The frequently used symbols are summarized in Table I.

IV. AUGMENTED TRANSFER LEARNING

Similar to previous works [9], [30], we prune the channels in a layer-by-layer manner in which each layer is considered as a state, and the corresponding preservation ratio is considered as the action. The current layer pruning only depends on its state information, i.e., the layer index, height, width, input channels, output channels, stride, and the pruning budget feed from the previous layer. Both accuracy and latency are investigated as rewards in this work and will be discussed in detail in Section VI. The DDPG-based RL agent [17], which consists of an actor and a critic, is employed to automatically predict the preservation ratio of each layer. The actor, which is usually a light-weight CNN model, exploits the information of the
states and generates the action. The critic, which is also a lightweight CNN model, is used to evaluate the action to suggest the optimizing direction of the actor. During the training process of the RL agent, the weight of the actor network and of the critic network is updated by maximizing the output of the critic network and by balancing the Bellman equation [31], respectively. As the actor and critic converge, the optimal pruned model can be obtained by measuring the preservation ratio of each layer according to the inference of the actor network given the input state information.

However, the actor network and critic network take a long time to converge due to the huge number of parameters in their networks and the randomness of the network initialization. Observing that the training process of a DNN can be accelerated by transfer learning, which reuses the weights of the source task as the starting point of the target task, we find transfer learning promising to boost the convergence of RL-based auto-pruning, since there are many common features between auto-pruning agents in different scenarios. For example, it has been observed that the optimal pruning policies for different preservation ratios have a similar pattern, which depends on the importance of each layer. Hence, we proposed to embed the transfer learning framework into the RL-based auto-pruner so that the weights trained in other scenarios can be reused to speedup the convergence of the current pruning agent.

Vanilla transfer learning may not work perfectly across different pruning ratios, models, and datasets. The experiments for transfer learning from ResNet20 to ResNet18 are shown in Fig. 2. It is obvious that transfer learning from ResNet20 with a preservation ratio of 0.5 and 0.6 have nonoptimal performance. This is caused by the inconsistency of the source preservation ratio and of the target preservation ratio. The high-preservation ratio of the source model may lead to high action in the target model. However, when the weights of the actor are transferred to a pruning scenario, which has a lower-preservation ratio, they tend to mislead the target model, and the predicted action for the initial layers becomes higher than expected. Given the constraints of the overall preservation ratio requirement, the action for the later layers will be suppressed and become lower than expected. This inconsistency will significantly harm the performance of the RL-based auto-pruning agent. In order to solve this problem, we propose augmented transfer learning, which consists of an automatic source selection scheme, DA, and NA, to efficiently transfer the knowledge between different pruning scenarios. The framework of the augmented transfer learning is illustrated in Fig. 3.

In our augmented transfer learning framework, we first select the source model candidates according to the rules given by the theoretical analysis. More specifically, in this work, we show a theoretical study of the transfer learning within the same models while the corresponding study for different models still lacks solid foundation. As a result, same source and target model are preferred in the transfer learning process which also coincide with the intuition. We also notice that our theoretical study is based on Taylor expansion, which indicates a constrain of low-pruning ratio (preservation ratio) so that higher-order items of the expansion can be ignored. For more details, please see Section IV-B.

Then, the source models can be further filtered by vanilla transfer learning from multiple starting points in a parallel manner. The optimal source model can be selected according to the source selection scheme in Section IV-A. If the convergence speed meets the time requirement specified by the user, the auto-pruning policy can be obtained by the inference of the source actor network. Otherwise, data and NA will be applied to enhance the transferability. Next, the augmented data and weights will be transferred (DT and WT) to the target buffer and augmented target actor, respectively. The actor will be trained according to the samples from the replay buffer and the optimization direction given by the critic network. As the augmented target actor converges, the pruning policy can be obtained by the inference of it. Finally, the augmented data and actor will be used to update the library to ease the later auto-pruning process. The source selection, DA, and NA are explained in the following sections.

A. Source Selection

As Fig. 2 shows, it is critical to select the starting point of the target network. A good source model may benefit the training process, while an inappropriate source model may harm the performance. However, it will be time consuming to finish the transfer learning process for all of the source models in the library. Hence, we exploit an early stopping scheme to efficiently search for the optimal source model.

In this scheme, transfer learning from multiple pretrained models starts at the same time, and a moving average-based smoothing process, which has a window size of 21 based on our experiments, is conducted for each learning curve, as shown in Fig. 2. The mean (µ) of the points inside the window is considered to be the value of the center points, while the variance (σ) of the points inside the window is a good approximation of the corresponding variance. Then, the
optimal transfer learning starting point can be selected according to the priority of different source models, which is defined as the following.

Def: We consider source model “A” to be significantly superior to source model “B” if there exists a trial $x$ in which $\mu_A(x) - \mu_B(x) > \sigma_A(x) + \sigma_B(x)$.

Thus, the transfer learning process from the nonoptimal source model can be stopped early to save resources, as illustrated by the green line and red line in Fig. 2. Note that there exists the case that one model may not be significantly superior to the other if the performance difference of the two source models is smaller than the corresponding variance. In this case, we choose the source model that has the better-inference accuracy at the maximum trial specified by the user.

B. Data Augmentation

Another problem of the RL agent comes from the buffer filling process at the beginning of the training stage. Noting that the adjacent states are strongly dependent, a replay buffer is employed to store and shuffle the training samples, e.g., states, information, actions, and corresponding rewards, to resolve the dependence. Though the replay buffer can achieve a decent performance improvement, the filling process is time consuming and exacerbates the training costs due to the high expense of each sample. However, this also opens up an opportunity for accelerating the RL-based auto-pruning. To solve this problem, we propose to transfer training data to provide more information between the source model and the target model in addition to the weights transfer. As a result, the transfer efficiency can be further improved. However, directly applying the weights transfer. As a result, the transfer efficiency can be further improved. However, directly applying the training data of the source pruning agent may cause bias of the target agent, since the pruning scheme of the source agent and target agent may be different. In this work, we propose the following DA to reduce the data bias between the source and target models.

First, we consider DA for transfer learning across the same model with different preservation ratios. We propose to take advantage of the relationship of the actions between different pruning scenarios to reduce the action bias caused by the inconsistency of the preservation ratios. Without losing the generality, we consider the pruning ratio $\tilde{p} = 1 - a$ of layer $k$ as a function of the layer and target pruning ratio $p = 1 - p_t$, which can be indicated by $\tilde{a}_k = \mathcal{P}(k, p)$, This function can be expanded into the Taylor series, as illustrated by the following:

$$\tilde{a}_k = \mathcal{P}|_{p=0} + \frac{\partial \mathcal{P}}{\partial p}|_{p=0} p + \mathcal{O}(p^2).$$

Here, we assume that the preservation is high, therefore $\tilde{p}$ is small and it can be easy to expand at the point $\tilde{p} = 0$. The other case, that $\tilde{p}$ is large, will be considered later. The constant term, $\mathcal{P}|_{p=0}$ is 0 as the fact that the pruning ratio for layer $k$ becomes 0 when the target pruning ratio is 0. However, $|\frac{\partial \mathcal{P}}{\partial p}|_{p=0}$ turns out to be hard to estimate. To eliminate this term, we consider two pruning scenarios, which are the source and target of the transfer learning, respectively, and we have

$$\tilde{a}_k = \mathcal{P}|_{p=0} + \frac{\partial \mathcal{P}}{\partial p}|_{p=0} p + \mathcal{O}(p^2).$$

Then, the first order derivative can be easily eliminated by ignoring the infinitesimal of the second order and combining these two equations, which gives

$$\tilde{a}_k = \frac{\partial \mathcal{P}}{\partial p}|_{p=0} p + \mathcal{O}(p^2).$$

Then, the augmentation equation for the action can be obtained by replacing $\tilde{a}$ and $\tilde{p}$ with $1 - a$ and $1 - p_t$, respectively

$$d_k^a = 1 - \frac{1 - p_t}{1 - p} (1 - a_k).$$

We also observe that this DA can effectively protect the critical layers from pruning. For example, when layer $k$ is critical and $a_k^a$ is close to 1, the second term will be close to zero. As a result, $d_k^a$ will be close to 1 as well.

More accurate data augmentations could be obtained by expanding the Taylor series to a higher order and solve the corresponding equations by applying the series to more scenarios, as follows:

$$\tilde{a}_k = \mathcal{P}|_{p=0} + \frac{\partial \mathcal{P}}{\partial p}|_{p=0} p + \frac{1}{2} \frac{\partial^2 \mathcal{P}}{\partial p^2} p^2 + \mathcal{O}(p^3).$$

where $c$ can be $s_1, s_2, t$, which are two source and one target scenarios in the transfer learning process. Therefore, the action in the target scenario can be inferred by the actions in the source scenarios. We will skip the tedious computing process and directly show the analytical solution, as follows:

$$d_k^a = \frac{(p_t - p^2)}{(p_t^1 - p^2) p^1} a_k^1 + \frac{(p_t - p^1)}{(p_t^2 - p^2) p^2} a_k^2.$$

It can be organized into the following format:

$$d_k^a = \alpha \frac{p_t}{p^1} a_k^1 + (1 - \alpha) \frac{p_t}{p^2} a_k^2,$$

where $\alpha = (p_t - p^2) / (p_t^2 - p^2)$. Clearly, by keeping the second term of the Taylor expansion, the action of the target pruning scenario can be estimated by a linear combination of the actions in two different source pruning scenarios. This naturally leads to multisource transfer learning. The parameter $\alpha$ indicates the contribution of each source model. When $p_t$ is close to $p^2$, $\alpha$ will be small and thus the contribution will be dominated by $s^2$, which conforms to our common sense. Finally, the representation for $d_k^a$ can be obtained according to $\tilde{a} = 1-a$ and $p = 1-p_t$, which is not shown here for simplicity.

In this work, we mainly discuss the first order approximation, as we find that it can already achieve satisfying performance. However, our framework can be extended to the second order approximation in future work.

Note that the conclusions we have reached so far are based on Taylor expansion at $p = 0$; therefore, it works perfectly for low-pruning ratio scenarios, while it is less satisfying when the pruning ratio increases. However, the formula for the high-pruning ratio scenarios can be obtained by expanding $\mathcal{P}(k, p)$ at $p = 0$ in a similar way. In this case, the first order of the action augmentation is

$$d_k^a = \frac{p_t}{p^1} a_k^1,$$

and the second order of the action augmentation is

$$d_k^a = \frac{p_t}{p^1} \frac{p_t - p^2}{p^1} a_k^1 + \frac{p_t}{p^2} \frac{p_t - p^1}{p^2} a_k^2.$$
It is not suggested to transfer between scenarios of high and low-pruning ratios as there is no analytical action augmentation rule found so far. However, experiments show that there is still a considerable accuracy and converging speed gain by directly applying the rules from the scenarios of either high or low-pruning ratios. In our experiments, we will show a comprehensive study of the transfer learning with the source and target pruning ratios uniformly distributed in \([0, 1]\).

Next, we consider the DA for transfer learning across different models and datasets. In this case, the samples are fine-tuned according to the prior information of the source and target model. We illustrate this using the widely used ResNet model. It can be observed that the shortcut layer and the top layer inside the ResNet block have fewer parameters than other layers. As a result, the preservation ratios for these layers are usually high since pruning these layers may lead to a significant accuracy loss. Therefore, it becomes natural to adjust the preservation ratio according to the importance of the layers for the transfer learning between different ResNet models. In our experiments, we set the preservation ratio to 1 for the layers that are critical in the target model. The preservation ratios for layers that are critical in the source model, while not critical in the target model, are uniformly reduced to ensure the overall preservation ratio is unchanged. Then, (4) and (8) can be used to reduce the action bias if the inconsistency of the preservation ratio exists.

Finally, these data are randomly sampled to train the auto-pruner, which is similar to the data sampling in the vanilla transfer learning.

C. Network Augmentation

The actor network and critic network inside the RL agent aim to predict the action for the given input states and to evaluate its corresponding performance. However, the transferability is not considered in the actor network design, which may lead to suboptimal transfer learning performance. We show that by modifying the network, the transferability can be significantly increased without causing extra computational costs.

To achieve this goal, we modify the actor network to generate invariants across different scenarios instead of the pruning ratio of the given layer, which may vary significantly for different pruning ratios and models. As a result, the weights of the actor network for different pruning scenarios can be easily reused with little retraining.

By observing the formula in (8), it is obvious that in the low-preservation scenario, the invariant for layer \(k\) between different pruning scenarios is

\[
I_{k, low} = \frac{\phi^c_k}{p^c} \tag{10}
\]

while in the high-preservation ratio scenarios the invariant is

\[
I_{k, high} = \frac{1 - \alpha_k}{1 - p^c} \tag{11}
\]

where \(\alpha_k\) indicates the action, i.e., the preservation ratio of layer \(k\), and \(p\) is the targeted preservation ratio. We will first introduce the NA according to (10).

The architecture of the currently widely used actor is shown in Fig. 4, which consists of several convolutional layers (or fully connected layers) for feature extraction and a sigmoid layer for action generation. Considering the fact that the sigmoid layer will always be the last layer to regulate the action to the range of \([0, 1]\), we propose to add a new layer \(\Phi\) to improve the transferability of the actor network, and the explicit format of \(\Phi\) can be obtained according to relationship of the invariant and the action. More specifically, we note the output of the convolutional layer as \(f_{conv}\). After passing the proposed \(\phi_{low}\) layer and sigmoid layer we have

\[
sigmoid\left[\phi_{low}(f_{conv})\right] = \alpha_k^c. \tag{12}\]

In order to bypass the sigmoid function, we approximate the sigmoid function using its first order expansion at 0, which is \(y = (1/4)x + (1/2)\), and it is illustrated in Fig. 4. Then, (12) becomes

\[
\frac{1}{4}\phi_{low}(f_{conv}) + \frac{1}{2} = \alpha_k^c. \tag{13}\]

Finally, we approximate the invariant using the convolutional layers inside the RL agent, which indicates \(I_{k, low} = f_{conv}\). As a result, we have

\[
\frac{1}{4}\phi_{low}\left(\frac{\alpha_k^c}{p^c}\right) + \frac{1}{2} = \alpha_k^c. \tag{14}\]

and \(\phi_{low}\) can be obtained

\[
\phi_{low}(x) = 4\left(p^c x - \frac{1}{2}\right). \tag{15}\]

Then, we will discuss the NA in cases where the preservation ratios are high. In this case, we will use the convolutional layers to predict the \(I_{k, high}\), which indicates \(I_{k, high} = (1 - \alpha_k^c)/(1 - p^c)\). In order to ease the later processing, we switch the preservation ratio into a pruning ratio prediction by adding a scalar and an adder after the sigmoid layer. Similar to (12) and (13), we have

\[
1 - \text{sigmoid}\left[\phi_{high}\left(f_{conv}\right)\right] = \alpha_k^c
\]

\[
\frac{1}{4}\phi_{high}\left(\frac{1 - \alpha_k^c}{1 - p^c}\right) + \frac{1}{2} = 1 - \alpha_k^c. \tag{16}\]

By replacing the convolutional layer with invariant \(I_{k, high}\), we have

\[
\Phi_{high}\left(\frac{1 - \alpha_k^c}{1 - p^c}\right) = 4\left[1 - \frac{\alpha_k^c}{1 - p^c}(1 - p^c) - \frac{1}{2}\right]. \tag{17}\]

Fig. 4. NA for low-target preservation ratio (low \(p^c\)) and high-target preservation ratio (high \(p^c\)).
It is clear that $\phi^{\text{high}}$ has the following format:

$$\phi^{\text{high}}(x) = 4 \left[ x(1 - p^2) - \frac{1}{2} \right].$$

(18)

By modifying the actor network, we will see that the transferability will significantly improve with very little computing overhead.

V. ASSISTANT LEARNING

In Section IV, we reported taking advantage of the previous well-trained model and data to improve the learning speed of the target model. However, the learning time is still intolerable in many time-critical scenarios. This is caused by the low-sampling efficiency of the RL algorithm, since the actor network is not well-tuned in the initial trials and the action generated by the actor network may be useless.

Inspired by the high-data efficiency of Bayesian optimization, where each sample is generated according to the historical samples and a well-defined acquisition function [32], we propose an AL process to generate the next training samples at the initial stage according to the training history. As a result, the suboptimal samples generated by the actor network at the initial stages can be avoided and the corresponding data efficiency can be significantly improved.

The transfer-learning-based RL algorithm, which trains the network according to the action generated by the actor, can be resumed when the performance of the RL-based auto-pruner outperforms the history-based counterpart.

The details of our proposed AL algorithm are shown in Fig. 5. The proposed algorithm contains two separate dataflows, as indicated by the red and green flow, separately. The green color refers to the RL-based auto-pruning process, which receives the action by the inference of the actor neural network. The dataflow highlighted in red represents the history-data-based action prediction flow to improve the data efficiency of the RL agent in initial stages. A switch is used to select the action from the two data paths and output the action to interact with the environment, which is network pruning in our case.

It is nontrivial to select the historical data according to the states of the layer to be pruned due to the large pruning space and numerous historical records. In order to exploit the pruning history efficiently, both the similarity ($K$) and reward ($R$) are considered. First, the samples in the pruning history $\{S, a, R\}_{i=1:B}$, where $B$ is the buffer size, are iterated and the scores for similarity are evaluated for each historical record. Then, historical data with a higher similarity and reward are selected according to a comprehensive performance metric ($M$) for later action generation.

The definition of the similarity $K_{ij}$ for current state $S_i$ and historical state $S_j$ is based on the widely used RBF kernel [33], which can be illustrated by the following:

$$K_{ij} = \exp\left(-\gamma \|S_i - S_j\|^2\right)$$

(19)

where $\gamma$ is the hyperparameter to adjust the distance measurements. This is based on the assumption that the design space of the pruning network is continuous and the samples with similar input state information should have a similar action. This is a rational and mild assumption because the accuracy changes continuously as the pruning ratio and the convolution size change. Then, the historical data can be selected according to both the similarity and the reward function (which will be discussed in detail in the next section), as illustrated by the following:

$$M_{ij} = K_{ij}^{\omega} + R_i$$

(20)

where $M_{ij}$ is the performance metric of historical record $\{S_i, a, R_i\}$ with regard to the input state $S_i$, and $\omega$ is the weight parameter to balance the tradeoff between the similarity and the reward signal. In our experiment, we let $\omega$ equals to two according to the experimental performance.

We also observe that both the performance metric and the diversity are critical to the training process and therefore we employ a dynamic selection module and a noise module to get a good tradeoff between the performance and diversity. For our dynamic selection unit, A FIFO with depth $D$ is used to store the sampled data and prevent it from later selection so that the diversity of the samples can be greatly increased. In our experiments, $D$ is equal to the length of the replay buffer, which is sufficient to remove the dependency between the samples inside the replay buffer. Then, the action of the input state can be given according to the selected historical record and noise, which also aims to widen the exploration space of the RL process. Note that it is critical to select appropriate noise to get a good tradeoff between exploration and exploitation. Both uniform noise and Gaussian noise are investigated in our experiments, and we find that a linearly decreasing uniform noise is more effective for the AL process.

Finally, the new interactions with the environment are fed back to the replay buffer to update the data in the buffer. In contrast with the RL agent, which directly accepts all the data and refreshes the buffer, we adopt a probability-based updating algorithm to get rid of the low-performance trials in the AL phase. This is because the RL agent focuses on the whole training phase, while our AL algorithm only focuses on the initial training phase where low-quality samples are common. Therefore, accepting all the samples will harm the convergence of the actor, and a probability-based accepting rule becomes necessary. In our case, the top one third samples in accuracy will be accepted and update the buffer without rejection. The other samples will be accepted with an exponential decreased probability.

We also investigated model-based action predictors to take advantage of the history knowledge and to predict the action. XGBoost-[34] and matrix-factorization [35]-based models were tried. Both the simulated annealing-based solver and Adam [36], which is embedded in Tensorflow, were employed to solve the maximum point of the model. However, we found that this is nonpractical due to the huge data requirement to
learn the model and the low generalizability of the model across different pruning scenarios.

VI. LEARNING REWARD DISCUSSION

In this section, we will discuss the reward function module as shown in Fig. 1. We mainly consider the accuracy and latency of the pruned neural network as the reward signal and try to optimize them by considering the constraints of the preservation ratio. Other performance metrics, such as power, energy efficiency, etc., can be processed in a similar way. However, it might not be nontrivial to evaluate the reward signals, especially for the latency, due to the randomness of the execution environment. To obtain accurate measurements while being efficient, we employ several tricks to preprocess the reward signal, as illustrated in Fig. 6.

A. Accuracy

In our experiments, the number of samples in the test dataset varies for different benchmarks. For example, we have 10,000 samples for Cifar10 [37], which is limited by the size of the open-source dataset, while for ImageNet, the sample size is 50,000 as there are more images available in the test dataset and more accurate measurements can be obtained. Being aware of the fact that the accuracy is naturally distributed in the range of $[0, 1]$, there is no need to normalize the accuracy and it can be directly forwarded to the DDPG agent and serves as the reward signal to indicate the quality of the pruned model.

As we aim to maximize the accuracy within the constraint of the preservation ratio, the budget of the remaining flops for the current layer is also calculated and serves as a feature to train the actor network. Thus, the actor can generate the action without violating the restriction of the target preservation ratio.

B. Latency

In contrast to the accuracy-oriented auto-pruning, evaluating the reward function of the latency-oriented auto pruning turns out to be nontrivial. We illustrate our latency measurements from the following perspectives.

1) Rebuild Model: Note that our channel pruner “prunes” the channels by masking out the corresponding weights with zeros. As a result, the size of the model remains unchanged and the auto-tuning agent might be misguided. In order to avoid the misleading measurements, it becomes necessary to rebuild the pruned model according to the corresponding pruning policy so that the actual latency can be measured precisely. To achieve this, first the pseudo pruned model and the corresponding masks are stored during the auto pruning process.

Then, the convolutional operations can be extracted from the checkpoint file and its pruned counterpart can be obtained by chopping the weights with the corresponding mask. Finally, the actual pruned model can be obtained by replacing the edge and node in the graph with the pruned version using getter, which is the interface to overwrite the internal variable provided by TensorFlow official.

2) Traceline-Based Time Measuring: Having rebuilt the pruned model, latency measuring still turns out to be nontrivial. An intuitive method could be measuring the end-to-end time of the graph executing session and repeating this process multiple times for higher confidence. However, the variance of the measurements is too large due to the randomness of the executing environment, such as the utilization of CPU, GPU, and memory consumption. The uncertainty of the operation scheduling algorithms will further aggravate this problem. As a result, the latency is hidden by the noise and the auto pruner can hardly obtain any useful information from the inaccurate measurements. To overcome this problem, we propose to measure the latency of the convolutional operations according to the traceline, which is provided by TensorFlow and contains the real-time information of the graph executing session. This is effective as we only prune the convolutional layers in the model. More specifically, we first parse the traceline and extract the real-time information of the convolutional operations. Then, we dismiss the redundant operations by removing the repeated records of the same operation in the traceline file. Finally, the running time of the pruned model can be obtained by summing up the time slices of the convolutional operations.

3) Kalman-Filter-Based Time Estimation: To maintain the stability of the measurements, we allocate our program in the execution environment, such as the utilization of CPU, GPU, and memory consumption. The uncertainty of the operation scheduling algorithms will further aggravate this problem. As a result, the latency is hidden by the noise and the auto pruner can hardly obtain any useful information from the inaccurate measurements. To overcome this problem, we propose to measure the latency of the convolutional operations according to the traceline, which is provided by TensorFlow and contains the real-time information of the graph executing session. This is effective as we only prune the convolutional layers in the model. More specifically, we first parse the traceline and extract the real-time information of the convolutional operations. Then, we dismiss the redundant operations by removing the repeated records of the same operation in the traceline file. Finally, the running time of the pruned model can be obtained by summing up the time slices of the convolutional operations.
first compare the current pruning policy with the pruning policies in the historical dataset to get the match set, which is the union of the records in \( \{(P, t, \sigma)\}_{i=1:N} \) that has the same pruning policy as the current one. If the match set is empty, the measured latency \( t_{\text{eval}} \) will be considered as the output after appending the current pruning record to the historical dataset. Otherwise, we will consider both the historical-data-based time estimation and the measurements-based time estimation as two independent Gaussian distributions, and the final time estimation of the input pruning record can be obtained by maximizing the joint Gaussian distribution. The details are illustrated in Algorithm 1.

4) Linear Normalization: We also observe that directly forwarding the latency to the DDPG agent may not guide the auto-pruning agent efficiently due to the wide distribution of the latency. Recall that the accuracy for the pruned network is naturally normalized to \([0, 1] \), which does not hold for latency-based auto-pruning. Different hardware may have a distinct latency for the same model, and this may significantly harm the transferability of the auto-pruning agent. For example, in our experiments, ResNet20 has an inference latency of sub-millisecond on a GPU, while it takes tens of milliseconds to run on a CPU. On the contrary, the difference of the latency measurements on the same platform might be small, and the agent can hardly learn useful knowledge from it. In order to solve this problem, we employ a linear normalization unit to preprocess the latency estimation. In our normalization unit, the standard latencies for different platforms and models are summarized for calibration purposes, and the latency estimated in the previous section is then scaled into the range \([0, 1]\) accordingly.

5) Constraint of Preservation Ratio: To optimize the latency with the constraint of the preservation ratio, it is possible to apply a similar constraining scheme with the accuracy counterpart. However, in this work, we apply a regularization-based latency constraint instead to show that both constraints work for our learning platform. Another reason for the regularization-based constraint is that it can be easily extended to other pruning scenarios, such as accuracy-latency co-optimization-based pruning. In this work, we adopt the following regularization, which is inspired by the work in [39]

\[
R(t, p) = t \ast \left( \min\left(\frac{p}{p^*}, 1\right) \right)^v
\]

(21)

where \( v \) is used to balance the tradeoff between the latency and the pruning ratio, and it is set to two according to the experiments. When \( p \) is larger than the target preservation ratio \( p^* \), the regularization term becomes 1 and the reward signal degrades to the latency \( t \), and therefore no regularization is enforced. When \( p \) becomes smaller than \( p^* \), a regularization factor will be applied to prevent the auto pruner from over-pruning and to get a good tradeoff between the latency and the preservation ratio. Finally, \( R(t, p) \) can be forwarded to the DDPG agent and serves as the reward signal to guide the training process of the actor and critic network.

VII. EXPERIMENTS

Our framework is run on an Intel Core i7-5820K CPU @3.30 GHz with a 32-GB DDR memory. Tensorflow 1.12 is employed for the auto-pruning process. The experiments are conducted on a Nvidia GeForce GTX TITAN X, which has 3072 cores and a boost frequency of 1089 MHz, leading to a peak throughput of 6691 GFLOPS. The GPU cards are connected with the host machine via a PCI-e 3.0 interface, which offers a maximum bandwidth of 8 GT/s. The widely used CUDA-10.1 is used to efficiently program the DNN applications on the GPU.

The vanilla RL agent-based auto-pruner conducted on the auto-pruning platform PocketFlow [30] is employed as the baseline. Both the actor and critic are 3-layer MLPs with a width of 64. An Adam optimizer with a learning rate of 0.001 is employed to train the pruner. The batch size is 64. Note that the accuracy in the following experiments refers to the inference accuracy of the pruned model without the fine-tuning process. The learning curve is smoothed using the exponential moving average, which is also built into the widely used TensorBoard platform, with a weight factor of 0.5. Please note that, in this work, the dataset for ResNet20 and ResNet56 is Cifar10, and the dataset for ResNet18, ResNet34 and MobileNet is ImageNet.

A. Time Analysis

In this section, we show that our proposed learning framework has little overhead in wall clock time compared to the naive auto-pruning and transfer-learning-based auto-pruning. To have a comprehensive comparison, we conduct experiments for different scenarios, which include different preservation ratios, networks, datasets, reward functions, etc. The results are shown in Figs. 7 and 8.

In Fig. 7, we provide the time comparison for training the baseline actor network and the augmented version. Note that “res20 \( p=0.4 \) accuracy” refers to the accuracy-oriented auto-pruning process in PocketFlow [30], which serves as the baseline in our experiments, with a preservation ratio of \( p = 0.4 \). The legend for other figures can be interpreted in a similar manner. It can be seen that in all cases the time consumption of training the original actor network is very close to that of augmented network, which indicates that our NA has
very little overhead in wallclock time. We also observe that
the pruning time of ResNet20 with a preservation ratio of 0.4
is close to that of the ResNet20 $p = 0.6$ counterpart. The time
consumption of ResNet18 and MobileNet is much longer as
they have a larger model size. The latency-oriented pruning
is also significantly more time consuming than its accuracy-
oriented counterpart as we measure the latency multiple times
for more accurate measurements.

In Fig. 8, we provide the time comparison between the
vanilla transfer learning and our AL. “res20 0.4 to 0.6 accu-
"rate” indicates transferring the knowledge learned in the
source pruning scenario, which has a preservation ratio of 0.4,
to the target pruning scenario, with a preservation ratio of 0.6.
Again, we observe that the time consumption of our frame-
work is close to that of vanilla transfer learning, with only a
slight overhead (less than 5\%) in all cases.

B. Vanilla Transfer Learning

In this section, comprehensive experiments are conducted
to show the performance of the vanilla transfer learning. The
results are illustrated in Figs. 9 and 10.

First, we show the performance of the transfer learning
across different preservation ratios. To have a comprehensive
and fair comparison, we provide experiments for all the trans-
fer learning cases within the ratio list (0.4, 0.5, 0.6, 0.7), as
illustrated in Fig. 10. In Fig. 10(a), we show the RL-based
auto compression for ResNet20 with a target preservation ratio
of 0.4. It can be observed that transfer learning can significantly
increase the convergence speed. The baseline RL agent con-
verges in around 100 trials, while the transfer learning-based
RL agent can achieve the same pruning accuracy within 60
trials, and an acceleration of $1.67\times$ is achieved. The trans-
fer learning process can also benefit the final accuracy of the
auto-pruner.

In Fig. 9(b), we show the transfer learning for ResNet18
with different source and target preservation ratios, i.e., “res18
0.5 to res18 0.4.” In contrast to the previous example, although
the transfer learning-based auto-pruner in this scenario has
better-pruning accuracy at the initial trials, it fails to outper-
form the baseline when the training process converges. This
indicates that vanilla transfer learning may not always benefit
the learning process, and may lead to degraded performance.
However, in later experiments, we show that this problem
may even poison the convergence of the auto-pruning
process. This is because of the inconsistency of the
preservation ratios, as we mentioned in Section IV.

2) Source models with lower-preservation ratios are
preferred. For example, in (b), “res20 0.4 to res20 0.5”
has a higher performance than “res20 0.6 to res20 0.5.”
In precis, although the transfer learning for (c) and (d) is
promising, the performances in (a) and (b) are far from
optimal, which motivates us to develop a novel framework
to boost the vanilla transfer learning process.

We further extend our transfer learning algorithm to
different datasets and DNN models, as shown in Fig. 9.
Fig. 9(a)-(c) shows the transfer learning-based auto-pruning
for ResNet18. In Fig. 9(a), the source model has a preservation
ratio of 0.4, while the target model has a preservation ratio of 0.5.
It can be observed that transfer learning can significantly
increase the convergence speed. The baseline RL agent con-
verges in around 100 trials, while the transfer learning-based
RL agent can achieve the same pruning accuracy within 60
trials, and an acceleration of $1.67\times$ is achieved. The trans-
fer learning process can also benefit the final accuracy of the
auto-pruner.

In Fig. 9(b), we show the transfer learning for ResNet18
with different source and target preservation ratios, i.e., “res18
0.5 to res18 0.4.” In contrast to the previous example, although
the transfer learning-based auto-pruner in this scenario has
better-pruning accuracy at the initial trials, it fails to outper-
form the baseline when the training process converges. This
indicates that vanilla transfer learning may not always benefit
the learning process, and may lead to degraded performance.
However, in later experiments, we show that this problem
may be avoided by our proposed augmented transfer learning
and AL.

Fig. 9(c) shows the transfer learning for ResNet18 from dif-
ferent models and datasets. In this case, the baseline converges
in 125 trials, while its transfer learning-based counterpart can
converge in 25 trials. Therefore, an around $5\times$ speedup can
be achieved.
In Fig. 9(d) and (e), we show the transfer learning for other DNN models. Transfer learning from ResNet18 to ResNet34 is shown in Fig. 9(d), while transfer learning from ResNet18 to a light-weight network running on embedded systems, such as MobileNet, is shown in Fig. 9(e). In both cases, a significant speedup of around 5× in convergence time can be observed.

In summary, our experiments in this section show that the transfer learning process can boost the learning process by 1.67×–5× for different pruning scenarios. However, it may harm the accuracy in a few cases; for example, Figs. 9(b) and 10(a). In the later experiments, we show that our proposed algorithms can solve this performance degradation problem and further speed up the learning process.

### C. Augmented Transfer Learning

In this section, we propose experiments to verify the validity of our augmented transfer learning algorithm. In order to have a fair comparison, we conducted experiments for all the transfer learning scenarios within the preservation ratio list (0.4, 0.5, 0.6, 0.7), similar to the experiments to obtain Fig. 10. The corresponding experiments are shown in Fig. 11. In (c) and (d), we observe that the performance of the augmented transfer learning is similar to its counterpart in Fig. 10, since they are already close to optimal. However, in (a) and (b), we found that the converging speedup can be significantly increased. We also observe a substantial improvement in the final accuracy of the pruned model. More specifically, the accuracy loss problem in “res20 0.6 to res20 0.4” and “res20 0.7 to res20 0.4,” as we mentioned in the previous section, has been solved.

We summarized the experiments for augmented transfer learning for other models in Section VII-E to have a more clear comparison.

### D. Assistant Learning

In this section, we provide the experiments for auto-pruning after the AL process. Note that AL relies on the output of the augmented transfer learning. The experiment for AL here indicates that both augmented transfer learning and AL are applied.

Similar to that reported in the previous section, we conducted experiments for all transfer learning scenarios as in Figs. 10 and 11. The corresponding results are shown in Fig. 12. In Fig. 12(c) and (d), the performance of the AL is still similar to its counterpart in Figs. 10 and 11, due to their near-optimal performance. However, in Fig. 12(a) and (b), we found that the converging time can be further reduced. For example, in Fig. 11(a), the augmented transfer learning converges in 125 trials, while its AL counterpart in Fig. 12(a) converges in 60 trials. The experiments for other models can be seen in Section VII-E.

In summary, by combining augmented transfer learning and AL, we observe that we can achieve an around 1.5×–2.5× speedup for ResNet20 with superior or comparable pruned accuracy.

### E. Experiments for Other DNNs

In this section, we provide the experiments for other neural networks, such as ResNet56, ResNet18, and MobileNet v1. We mainly focus on auto-pruning with a target preservation ratio of 0.4 since it is the most challenging case among the 4 cases, as we observed from previous experiments. We present the PocketFlow-based baseline, vanilla transfer learning, augmented transfer learning, and AL-based auto-pruning in the same table, so that the gain of our proposed framework can be illustrated more clearly.

Table II shows the auto-pruning experiments for ResNet56. For transfer learning from ResNet20 with a preservation ratio of 0.4, we observe that vanilla transfer learning, augmented transfer learning and AL have similar converging speeds, which are much faster than the baseline version. A 2× speedup can be achieved. Additionally, our proposed AL can achieve higher accuracy. This is because the history-based sampling can possibly lead to better-design points in the design space. For transfer learning from ResNet20 with a preservation ratio...
of 0.7, we show that our proposed augmented transfer learning and AL have superior converging speed compared to their vanilla transfer learning counterpart. Our proposed framework can achieve an around 1.81× acceleration over the PocketFlow baseline and 1.36× acceleration over the vanilla transfer learning counterpart.

Similarly, Table III shows the experiments for ResNet18, which has a larger dataset, e.g., ImageNet. For transfer learning from ResNet20 with a preservation ratio of 0.4, we observe that both the augmented transfer learning and AL have little performance gain in converging speed and accuracy since the vanilla transfer learning is already close to optimal performance. However, for transfer learning from ResNet20 with a preservation ratio of 0.7, our proposed framework can achieve higher performance in accuracy. This is because transfer learning from “res20 0.7” is more challenging than from “res20 0.4” and therefore, our proposed learning framework can unleash its superior performance compared to vanilla transfer learning.

In Table IV, we show the experiments for MobileNet v1, which is also trained on ImageNet. A time acceleration of 2.375× can be achieved compared to the PocketFlow baseline. A significant accuracy gain in AL can also be observed.

We are aware that the converging speed of an RL agent is more sensitive to the network architecture than to the input model or target sparsity. Therefore, in this work, we use the same 3-layer fully connected network architecture as [9] and [21] to ensure a fair comparison. In [9], the RL agent takes 400 epochs to converge during the channel pruning process.

Work [21] improves the convergence epoch to 55, whereas in our experiments, the RL agent converges in just 50 epochs, indicating that we have pushed the RL training speed to its limit.

In summary, in this section, we conducted comprehensive experiments for different preservation ratios and DNN models. A speedup of 1.81× –2.375× can be observed for different auto-pruning scenarios. A substantial performance gain in accuracy is also observed.

F. Experiment for Latency-Based Auto Pruning

In this section, we provide the experiments for latency oriented transfer learning.

1) Within the Same Platform: First, we show the experiments on the single platform, which is a Nvidia GeForce GTX TITAN X, in this section. The latency is measured by running single-image on it. To show the generality of our experiments, we conducted experiments for all the transfer pairs on the ratio list [0.2, 0.4, 0.6, 0.8]. The results of the experiments are shown in Tables V and VI.

In Table V, we show the latency oriented pruning setup and the corresponding inference time measurements in milliseconds. The “source p” indicates the preservation ratio of the source model, while “target p” indicates the preservation ratio of the target model. The data in each block are the optimized latency (in milliseconds) achieved by vanilla transfer learning and our proposed AL, separately. ResNet20 is employed as the benchmark as our experiments are intensive and the computing workload will significantly increase when pruning larger networks on ImageNet. We observe that in most cases our AL-based pruning can achieve better latency.

We also provide the converging time (in trials) comparison in Table VI. The notations can be interpreted in a similar way to those in Table V. It is obvious that in most cases our AL can achieve a comparable or better-converging speed.
learning has a much shorter converging time since it is stuck in the local minimum and failed to find the optimized latency.

2) Transfer Learning Across Different Platforms: The results of transfer learning across different platforms are provided in Table VII. The source model and the target model is executed on a GPU and a CPU, respectively. We notice that the vanilla transfer learning has even worse latency than the baseline provided by [30]. This is because of the large latency gap between the auto-pruning in the CPU and that in the GPU. However, the result of the AL shows a significant improvement in the convergence time and comparable optimized latency. Therefore, we conclude that vanilla transfer learning fails to improve the learning speed because the latency on the CPU is significantly different from that on the GPU. However, this learning gap can be minimized and the learning process can be significantly boosted by performing DA and NA.

In summary, our AL can significantly accelerate the auto pruning process while improving the quality of the pruned network that RL can find in the latency oriented auto pruning tasks.

VIII. Conclusion

In this article, we proposed a comprehensive transfer learning framework for the RL agent. An augmented transfer learning algorithm and an AL algorithm were proposed to take advantage of the historical data from other pruning scenarios to boost the convergence speed of the network inside the pruning agent, thus saving computing resources and time. The experiments show that our framework can significantly reduce the convergence time with superior or comparable pruning accuracy. As pruned models have a compact structure and are often utilized in resource-constrained devices, such as mobile phones and wearable devices, we believe that our pruning methodology can be practical in scenarios, such as training personal networks for these devices. In the future, we would like to extend our framework to applications, such as NAS, auto quantization, etc.

REFERENCES

[1] S. Han, J. Pool, J. Tran, and W. J. Dally, “Learning both weights and connections for efficient neural network,” in Proc. Adv. Neural Inf. Process. Syst., 2015, pp. 1–9.
[2] Y. He, X. Zhang, and J. Sun, “Channel pruning for accelerating very deep neural networks,” in Proc. IEEE ICCV, 2017, pp. 1398–1406.
[3] Z. Zhuang et al., “Discrimination-aware channel pruning for deep neural networks,” in Proc. Adv. Neural Inf. Process. Syst., 2018, pp. 1–12.
[4] J.-H. Luo and J. Wu, “An entropy-based pruning method for CNN compression,” 2017, arXiv:1706.05791.
[5] C. Louizos et al., “Learning sparse neural networks through L0 regularization,” 2017, arXiv:1712.01312.
[6] Z. Tan et al., “PCNN: Pattern-based fine-grained regular pruning towards optimizing CNN accelerators,” 2020, arXiv:2002.04997.
[7] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, “Rethinking the value of network pruning,” 2018, arXiv:1810.05270.
[8] Y. Li, “Deep reinforcement learning: An overview,” 2017, arXiv:1701.07274.
[9] 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.
[10] L. Torrey and J. Shavlik, “Transfer learning,” in Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques, Hershey, PA, USA: IGI Global, 2010.
[11] Z. Liu et al., “MetaPruning: Meta learning for automatic neural network channel pruning,” in Proc. IEEE ICCV, 2019, pp. 3295–3304.
[12] M. Lin, R. Ji, Y. Zhang, Y. Wu, and Y. Tian, “Channel pruning via automatic structure search,” 2020, arXiv:2001.08565.
[13] D. Karaboga, “An idea based on honey bee swarm for numerical optimization,” Comput. Eng. Dept., Erciyes Univ., Rep. tr06, 2005.
[14] R. Tibshirani, “Regression shrinkage and selection via the lasso,” J. Roy. Statist. Soc.: Ser. B (Methodological), vol. 58, no. 1, pp. 267–288, 1996.
[15] H. Fan, J. Mu, and W. Zhang, “High-dimensional Bayesian optimization for CNN auto pruning with clustering and rollback,” in Proc. ECCV, 2022, pp. 1–17.
[16] Z. Wang, X. Xie, and G. Shi, “RFPruning: A retraining-free pruning method for accelerating convolutional neural networks,” Appl. Soft Comput., vol. 113, Dec. 2021, Art. no. 107860.
[17] T. P. Lillicrap et al., “Continuous control with deep reinforcement learning,” 2015, arXiv:1509.02971.
[18] A. Nair et al., “Accelerating online reinforcement learning with offline datasets,” 2020, arXiv:2006.09359.
[19] K. Pertsch et al., “Accelerating reinforcement learning with learned skill priors,” 2020, arXiv:2012.03054.
[20] W. Wen et al., “Neural predictor for neural architecture search,” in Proc. Comput. Vis.–ECCV 2020: 16th Eur. Conf., 2020, pp. 660–670.
[21] M. Gupta et al., “Learning to prune deep neural networks via reinforcement learning,” 2020, arXiv:2007.04756.
[22] C. Wang and S. Mahadevan, “Heterogeneous domain adaptation using manifold alignment,” in Proc. 22nd Int. Joint Conf. Artif. Intell., 2011, pp. 1541–1546.
[23] L. Duan, D. Xu, and I. Tsang, “Learning with augmented features for heterogeneous domain adaptation,” 2012, arXiv:1206.4660.
[24] B. Kulis, K. Saenko, and T. Darrell, “What you saw is not what you get: Domain adaptation using asymmetric kernel transforms,” in Proc. CVPR 2011, 2011, pp. 1785–1792.
[25] Y. Zhu et al., “Heterogeneous transfer learning for image classification,” in Proc. 25th AAAI Conf. Artif. Intell., 2011, pp. 1304–1309.
[26] J. T. Zhou, I. W. Tsang, S. J. Pan, and M. Tan, “Heterogeneous domain adaptation for multiple classes,” in Proc. Artif. Intell. Statist., 2014, pp. 1095–1103.
[27] M. E. Taylor and P. Stone, “Transfer learning for reinforcement learning domains: A survey,” J. Mach. Learn. Res., vol. 10, pp. 1633–1685, Dec. 2009.
[28] Z. Zhu et al., “Transfer learning in deep reinforcement learning: A survey,” 2020, arXiv:2009.07888.
[29] Y. Yu, “Towards sample efficient reinforcement learning,” in Proc. IJCAI, 2018, pp. 5739–5743.
[30] J. Wu et al., “PocketFlow: An automated framework for compressing and accelerating deep neural networks,” in Proc. NIPS 2018 Workshop CDNNRIA, 2018, pp. 1–5.
[31] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. Cambridge, MA, USA: MIT Press, 2018.
[32] P. I. Frazier, “A tutorial on Bayesian optimization,” 2018, arXiv:1807.02811.
[33] K. M. Tao, “A closer look at the radial basis function (RBF) networks,” in Proc. 27th Asilomar Conf. Signals, Syst. Comput., 1993, pp. 401–405.
[34] T. Chen et al., “XGBOOST: A scalable tree boosting system,” in Proc. 22nd ACM SIGKDD, 2016, pp. 785–794.
[35] Y. Koren, R. Bell, and C. Volinsky, “Matrix factorization techniques for recommender systems,” Computer, vol. 42, no. 8, pp. 30–37, 2009.
[36] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014, arXiv:1412.6980.
[37] A. Krizhevsky, “Learning multiple layers of features from tiny images,” Dept. Comput. Sci., Univ. Toronto, Rep. 2009-TR, 2009.
[38] G. Welch and G. Bishop, “An introduction to the Kalman filter,” Univ. North Carolina, Chapel Hill, NC, USA, Rep. TR 95-041, 1995.
[39] X. Dai et al., “ChamNet: Towards efficient network design through platform-aware model adaptation,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2019, pp. 11398–11407.

Table VII
Boosting the Convergence of ResNet20 $p=0.4$ on CPU with History Data from GPU

| pruning scenarios | source model | convergence time (trials) | latency (ms) |
|-------------------|--------------|--------------------------|-------------|
| baseline [30]     | None         | 110                      | 9.3412      |
| vanilla transfer  | res20 0.4 GPU | 100                      | 9.807       |
| augmented transfer| res20 0.4 GPU | 30                       | 9.7565      |
| assistant Learning| res20 0.4 GPU | 50                       | 9.4966      |
| vanilla transfer  | res20 0.7 GPU | 30                       | 9.6681      |
| augmented transfer| res20 0.7 GPU | 30                       | 9.6331      |
| assistant Learning| res20 0.7 GPU | 50                       | 9.5395      |
Jiandong Mu received the B.S. degree in physics from the University of Science and Technology of China, Langfang, China, in 2014. He is currently pursuing the Ph.D. degree with the Department of Electronic and Computer Engineering, The Hong Kong University of Science and Technology, Hong Kong.

His current research include FPGA design for DNN applications, DNN modeling and optimization on FPGA, low-bits training, and software and hardware joint optimization for auto compression for DNN.

Mengdi Wang received the Bachelor of Engineering degree from Tsinghua University, Beijing, China, in 2012, the Bachelor of Economics degree from Peking University, Beijing, in 2014, and the Ph.D. degree in electronic engineering from Tsinghua University in 2017.

She is currently a Senior Algorithm Engineer with Alibaba Group, Hangzhou, China. Her research mainly focus on deep learning, low-bits training, and software and hardware joint optimization for auto compression for DNN.

Feiwen Zhu received the bachelor’s degree from Shanghai University, Shanghai, China, in 2009, and the master’s degree from Fudan University, Shanghai, in 2012.

He is currently a Senior Algorithm Engineer with Alibaba Group, Hangzhou, China. His research mainly focused on GPGPU, parallel computing, medical image, speech recognition (RNN), and software and hardware joint optimization for auto compression for DNN.

Jun Yang received the Bachelor of Computer Science degree from the University of Science and Technology of China, Langfang, China, in 2004, and the Master of Computer Architecture degree from the Chinese Academy of Sciences, Beijing, China, in 2007.

He has been top architect in multiple company, such as Yahoo, Sunnyvale, CA, USA; Qihoo, Beijing; and Alibaba, Hangzhou, China. He is currently the Director of Compute Arch with Nvidia, Santa Clara, CA, USA. His research interests mainly focus on parallel computing, deep learning acceleration on hardware, low-bit training, and large language model.

Wei Lin received the Bachelor of Computer Science degree from the University of Science and Technology of China, Langfang, China, in 2002, and the Master of Computer Science degree from the Chinese Academy of Sciences, Beijing, China, in 2005.

He is currently the Chief Architect of Big-Data Computation Platform and the Senior Director of Platform of Artificial Intelligence with Alibaba Group, Hangzhou, China. His research mainly focus on deep learning acceleration on hardware, parallel computing, novel computing platform for emerging AI algorithm, LLM, and model pruning.

Wei Zhang (Member, IEEE) received the Ph.D. degree from Princeton University, Princeton, NJ, USA, in 2009.

She was an Assistant Professor with the School of Computer Engineering, Nanyang Technological University, Singapore, from 2010 to 2013. She joined The Hong Kong University of Science and Technology, Hong Kong, in 2013, where she is currently an Associate Professor and established the Reconfigurable Computing System Laboratory. Her current research interests include reconfigurable systems, FPGA-based design, low-power high-performance multicore systems, electronic design automation, embedded systems, and emerging technologies.