Deep Model Compression via Deep Reinforcement Learning

Huixin Zhan [1] **and Yongcan Cao** [1]
[1]Department of Electrical and Computer Engineering
University of Texas, San Antonio, Texas 78249
yongcan.cao@utsa.edu

Abstract

Besides accuracy, the storage of convolutional neural networks (CNN) models is another important factor considering limited hardware resources in practical applications. For example, autonomous driving requires the design of accurate yet fast CNN for low latency in object detection and classification. To fulfill the need, we aim at obtaining CNN models with both high testing accuracy and small size/storage to address resource constraints in many embedded systems. In particular, this paper focuses on proposing a generic reinforcement learning based model compression approach in a two-stage compression pipeline: pruning and quantization. The first stage of compression, *i.e.*, pruning, is achieved via exploiting deep reinforcement learning (DRL) to co-learn the accuracy of CNN models updated after layer-wise channel pruning on a testing dataset and the FLOPs, number of floating point operations in each layer, updated after kernel-wise variational pruning using information dropout. Layer-wise channel pruning is to remove unimportant kernels from the input channel dimension while kernel-wise variational pruning is to remove unimportant kernels from the 2D-kernel dimensions, namely, height and width. The second stage, *i.e.*, quantization, is achieved via a similar DRL approach but focuses on obtaining the optimal weight bits for individual layers. We further conduct experimental results on CIFAR-10 and ImageNet datasets. For the CIFAR-10 dataset, the proposed method can reduce the size of VGGNet by $9\times$ from 20.04MB to 2.2MB with 0.2% accuracy increase. For the ImageNet dataset, the proposed method can reduce the size of VGG-16 by $33\times$ from 138MB to 4.14MB with no accuracy loss.

1 Introduction

CNN has shown advantages in producing highly accurate classification in various computer vision tasks evidenced by the development of numerous techniques such as VGG (Simonyan and Zisserman 2014), ResNet (He et al. 2016), DenseNet (Huang et al. 2017), and numerous automatic neural architecture search approaches (Liu et al. 2018; Real et al. 2018; Veit and Belongie 2018; Yu, Yu, and Ramalingam 2018). Albeit promising, the complex structure and large number of weights in these neural networks often lead to explosive computation complexity. Real world tasks often aim at obtaining high accuracy under limited computational resources. This motivates a series of works towards a light-weight architecture design and better speed-up ratio-accuracy trade-off, including Xception (Chollet 2017), MobileNet/MobileNet-V2 (Howard et al. 2017; Sandler et al.), ShuffleNet (Zhang et al. 2018), and CondenseNet (Huang et al. 2018), where group and depthwise convolutions are crucial.

In addition to the development of the aforementioned efficient CNN models for fast inference, many results have been reported on the compression of large scale models, *i.e.,* reducing the size of large-scale CNN models with little or no impact on their accuracies. Examples of the developed methods include low-rank approximation (Denton et al. 2014; Lebedev et al. 2014; Hinton and Van Camp 1993; Kavukcuoglu et al. 2010), network quantization (Chen et al. 2015; Han et al. 2015; Courbariaux et al. 2016; Rastegari et al. 2016), knowledge distillation(Hinton, Vinyals, and Dean 2015), and weight pruning (Han et al. 2015; Han, Mao, and Dally 2015; Zhuang et al. 2018; He, Zhang, and Sun 2017; Jia et al. 2018; Liu et al. 2017; Zhu and Gupta 2017), which focus on identifying unimportant channels that can be pruned. However, one key limitation in these methods is the lack of automatic learning of the pruning policies or quantization strategies for reduced (and minimized) models.

Instead of identifying insignificant channels and then conducting compression during training, another potential approach is to use reinforcement learning (RL) based policies to determine the compression policy automatically. There are limited results on RL based model compression (He et al. 2018; Wu et al. 2018). In particular, (He et al. 2018; Wu et al. 2018) proposed a deep deterministic policy gradient (DDPG) approach that uses reinforcement learning to efficiently sample the designed space for the improvement of model compression quality. While DDPG can provide good performance in some cases, it often suffers from performance volatility with respect to the hyperparameter setup and other tuning methods. Another typical issue is that the learned Q-function becomes dramatically overestimated, leading to the policy instability due to the incorrect exploitation of Q-functions. Recently, RL based search strategies have been developed to formulate neural architecture search. For example, (Baker et al. 2016;

Zoph and Le 2016; Zhong et al. 2018; Zoph et al. 2018) considered the generation of a neural architecture via considering agent's action space as the search space in order to model neural architecture search as a RL problem. Different RL approaches were developed to emphasize different representations of the agent's policies along with the optimization methods. In particular, (Zoph and Le 2016; Zoph et al. 2018) used a recurrent neural network based policy to sequentially sample a string that in turn encodes the neural architecture. Both REINFORCE policy gradient algorithm (Sutton et al. 2000) and Proximal Policy Optimization (PPO) (Schulman et al. 2017) were used to train the network. Differently, (Baker et al. 2016) used Q-learning to train a policy that sequentially chooses the type of each layer and its corresponding hyperparameters. Note that (Baker et al. 2016; Zoph and Le 2016; Zhong et al. 2018; Zoph et al. 2018) focuses on generating CNN models with efficient architectures, while not on the compression of large scale CNN models.

In this paper, we propose to develop a novel two-stage DRL framework for deep model compression. In particular, the proposed framework integrates per-layer pruning rate learning based on testing accuracy and FLOPs, and per-layer weight bits learning based on testing accuracy. Based on the obtained per-layer pruning rate, we first conduct channel pruning that will prune the input channel dimension (*i.e.,* C dimension) with minimized accumulated error in feature maps. We then conduct fine-tuning with kernel-wise pruning using information dropout, named iDropPruning, to prune the height and width of the kernel (*i.e.,* H and W dimensions).

This paper has three main contributions:
1. We propose a novel DRL algorithm that can obtain stabilized policy and address Q-value overestimation in DDPG by introducing four improvements: (1) *computational constrained* PPO: Instead of collecting T timesteps of action advantages in each of N parallel actors and updating the gradient in each iteration based on NT action advantages in one iteration of the typical PPO, we propose to collect *one* timestep Q-value in each of N parallel actors and update the gradient each timestep based on the N sampled Q-values; (2) adaptive Kullback-Leibler (KL) penalty / PPO-Clip Objective: We propose to maximize the expected return of the policy subject to a constraint on the size of the policy update measured by the KL penalty of policies. The coefficients of KL penalty are adjusted based on a targeted KL penalty to guarantee that the policy is updated within a range in each iteration. Another alternative is to modify the expected return of the policy by clipping subject to policy change penalization. (3) *smoothed* policy update: Our algorithm first enables multiple agents to collect one minibatch of Q-values based on the prior policy and updates the policy while penalizing policy change. The target networks are then updated by slowly tracking the learned policy network and critic network; and (4) target policy regularization: We propose to smooth Q-functions along regularized actions via adding noise to the target action. The four improvements altogether can substantially improve performance of DRL to yield more stabilized per-layer prune ratio and weight bits for deep compression, hence outperforming the traditional DDPG. We experimentally show the volatility of DDPG-based compression method in order to backup some common failure mode of policy exploitation in DDPG-based method as shown in Figure 4.
2. Pruning: We propose a new DRL-supported compression structure with iDropPruning (fine-tuning) that can prune three dimensions of CNN. We further learn the Pareto front of a set of models with two-dimensional outputs, namely, model size and accuracy, such that at least one output is better than, or at least as good as, all other models by constraining the actions. More compressed models can be obtained with little or no accuracy loss.
3. Quantization: We propose a new quantization method that uses the same DRL-supported compression structure, where the optimal bit allocation strategy (per-layer weight bits) is obtained in each iteration via learning the updated accuracy. Fine-tuning is further executed after each rollout. DRL-supported compress structure can enable more fine-grained quantization and better performance.

2 A Deep Reinforcement Learning Compression Framework

In this section, we focus on presenting the proposed new generic reinforcement learning based model compression approach in a two-stage compression pipeline: pruning and quantization. Fig. 1 shows the overall structure.

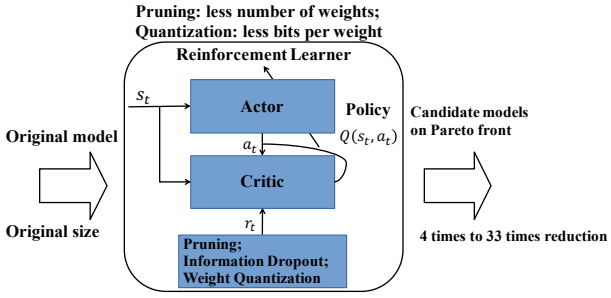


Figure 1: The two-stage compression pipeline: pruning and quantization. Adopting the pipeline can achieve a typical model compression rate between $4\times$ and $33\times$. Investigating the Pareto front of candidate compression models shows little or no accuracy loss.

Modeling of the State

In both pruning and quantization, in order to discriminate each layer in the neural network, we use a 8-dimension vector space to model a continuous state space:

$$s_t = \left[N_{Lr} \quad N \quad C \quad H \quad W \quad \text{stride} \quad A_H^t \quad \text{FLOPs}\right],$$

where N_{Lr} is the layer number, N and C are the dimension of, respectively, *output channels* and *input channels*, H is the kernel height, W is the kernel width, stride is the number of pixels shifts over the input matrix, A_H^t is the maximum pruning rate in pruning (respectively, the maximum number of weight bits in quantization) with respect to layer t. FLOPs is the number of floating point operations in each layer.

Modeling of the Action

In pruning, determining the compression policy is challenging because the pruning rate of each layer in CNN is related in an unknown way to the accuracy of the post-compression model. Since our goal is to simultaneously prune the C, H, and W dimensions, β is a vector whose dimension matches the $4D$ tensor with shape $N \times C \times W \times H$ in each layer. Define β_i, the ith entry of β, as a binary mask for each weight in the kernel. As the dimension of channels increases or the network goes deeper, the computation complexity increases exponentially. Instead of searching over a discrete space, a continuous reinforcement learning control strategy is needed to get a more stabilized *scalar continuous action space*, which can be represented as $a_t = \{pr_t | pr_t \in [pr_l, pr_h]\}$, where pr_l and pr_h are the lowest and highest and pruning rates, respectively. The compression rate in each layer is taken as a replacement of high-dimensional discrete masks at each weight of the kernels. Similarly, in quantization, the action is also modeled in a *scalar continuous action space*, which can be represented as $a_t = \{b_t | b_t \in N^+\}$, where b_t is the number of bits in layer t.

Reward

To evaluate the performance of the proposed two-stage compression pipeline, we propose to construct two reward structures, labeled $r1$, and $r2$. $r1$ is a synthetic reward system as the normalization of current accuracy and FLOPs. $r2$ is an accuracy-concentrated reward system. In pruning, the reward for each layer can be chosen from $r_t \in \{r1, r2\}$. In quantization, we use $r2$ as our selected reward structure. In particular, $r2 = p_{ac}$ and $r1 = 1 - \frac{\text{FLOPs}_t - \text{FLOPs}_{low}}{\text{FLOPs}_{high} - \text{FLOPs}_{low}} + p_{ac}$, where p_{ac} is the current accuracy.

The Proposed DRL Compression Structure

In the proposed model compression method, we learn the Pareto front of a set of models with two-dimensional outputs (model size and accuracy) such that at least one output is better than (or at least as good as) all other outputs. We adopt a popular synchronous RL framework to compress a pretrained network layer-by-layer. At time step t, we denote the observed state by s_t, which corresponds to the per-layer features. The action set is denoted by $\mathcal{A}$ of size 1. An action, $a_t \in \mathcal{A}$, is drawn from a policy function distribution: $a_t \sim \mu(s|\theta^\mu) \in \mathbb{R}^1$, referred to as an Actor, where θ^μ is the current policy network parameter. The actor receives the state s_t, and outputs an action a_t. After this layer is compressed with pruning rate or weight bits a_t, the environment then returns a reward r_t according to the reward function structure $r1$ or $r2$. The updated state s_{t+1} at next time step $t+1$ is observed by a known state transition function $s_{t+1} = f(s_t, a_t)$, governed by the next layer. In this way, we can observe a random minibatch of transitions consisting of a sequence of $B = \{(s_t, a_t, r_t, s_{t+1})\}$. In typical PPO, the surrogate objective is represented by $\hat{\mathbb{E}}_t[\frac{\pi^{\theta^\mu}(a|s)}{\pi^{\theta^{\mu^-}}(a|s)}\hat{A}_t]$, where the expectation $\hat{\mathbb{E}}_t[\cdot]$ is the empirical average over a finite batch of samples, θ^{μ^-} is the prior policy network parameter,

$$\hat{A}_t = \sum_{k=1}^{T-t+1} \gamma^k [r_{t+k} + \gamma V(s_{t+k+1}) - V(s_{t+k})],$$
$$V(s_t) = \mathbb{E}[\sum_{i=t}^{t+T} \gamma^{i-t} r_i | s_t], \forall s \in \mathbb{S}, \quad (1)$$

γ is the discount factor, and $\mathbb{S}$ is the state space. If we compute the action advantage $\hat{A}_t$ in each layer, T-step time difference rewards are needed, which is computationally intensive. We propose to replace the action advantages by Q-functions given by $Q(s_t, a_t) = \mathbb{E}[\sum_{i=t}^{t+T} \gamma^{i-t} r_i | s_t, a_t]$, referred to as Critic.

The policy network parameterized by θ^μ and the value function parameterized by θ^Q are then jointly modeled by two neural networks. Let $a = \mu(s_i|\theta^{\mu^-})$. We can learn θ^Q via Q-function regression, namely, (2), and learn θ^μ over the trace B with simultaneous KL-Penalty, namely, (3), or PPO-Clip objective stochastic policy gradient, namely, (4) as

$$\theta^Q = \arg\min_\theta \frac{1}{|B|} \sum_{(s_i, a_i, r_i, s_{i+1}) \in B} (y_i - Q(s_i, a_i|\theta))^2, \quad (2)$$

$$\theta^\mu = \arg\max_\theta \frac{1}{|B|} \sum_{(s_i, a_i, r_i, s_{i+1}) \in B} \Bigg[\frac{\pi_\theta(a|s_i) Q(s_i, a|\theta^Q)}{\pi^{\theta^{\mu^-}}(a|s_i)} - \alpha KL[\pi_{\theta^{\mu^-}}(\cdot|s_i), \pi_\theta(\cdot|s_i)] \Bigg], \quad (3)$$

$$\theta^\mu = \arg\max_\theta \hat{\mathbb{E}}_{(s_i, a_i, \cdots) \in B} \min \Bigg\{ \frac{\pi_\theta(a|s_i)}{\pi^{\theta^{\mu^-}}(a|s_i)} Q(s_i, a|\theta^Q), clip(\frac{\pi_\theta(a|s)}{\pi^{\theta^{\mu^-}}(a|s)}, 1-c, 1+c) \times Q(s_i, \mu(s_i|\theta^{\mu^-})|\theta^Q) \Bigg\}. \quad (4)$$

We further adjust the coefficient α of the KL penalty based on a targeted KL penalty δ to guarantee that the policy is updated within a range in each iteration. We denote the average KL penalty of policies within a minibatch of traces by δ'. If $\delta' \geq 1.5\delta$, the coefficient is adjusted to 2α. If $\delta' \leq 1.5\delta$, α is updated by $\alpha \leftarrow \frac{\alpha}{2}$. Let the target actor network and critic network be parameterized by $\theta^{\mu'}$ and $\theta^{Q'}$. Then the target networks are updated by $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$ and $\theta^{Q^{\mu'}} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$, where τ is the learning rate. A pseudocode of DRL compression structure is shown in Algorithm 2.

3 Pruning

In this section, we present two schemes to compress CNN with little or no loss in accuracy by employing reinforcement learning to co-learn the layer-wise pruning rate and the kernel-wise FLOPS. Similar to the aforementioned actor-critic framework, the layer-wise pruning rate is computed by the actor. After obtaining pruning rate a_t, layer s_t is pruned by a typical channel pruning method (He, Zhang, and Sun

2017), whose detail will be given below, to select the most representative channels and reduce the accumulated error of feature maps. In other words, after we get the pruning rate, channel pruning can be used to determine which specific channels are less important or we can simply prune based on the weight magnitude. In each iteration, the CNN is further compressed and fine-tuned by *iDropPruning*.

Pruning from C Dimension: Channel Pruning

The C-dimension channel pruning can be formulated as: $\arg\min_{\beta,W} \frac{1}{2N} \left\| Y - \sum_{i=1}^{C} \beta_i X_i W_i^T \right\|_F^2 + \lambda \|\beta\|_1$, subject to $\|\beta\|_0 \leq pr \times C$, $\|W_i\|_F = 1, \forall i$, where pr is the pruning rate, X_i and Y are the input volume and the output volume in each layer, W_i is the kernel weight, β is the coefficient vector of length C for channel selection, and λ is a positive weight to be selected by users. This optimization is solved in two steps. First, solve the LASSO regression problem given by $\hat{\beta}^{LASSO}(\lambda) = \arg\min_{\beta} \frac{1}{2N} \left\| Y - \sum_{i=1}^{C} \beta_i X_i W_i^T \right\|_F^2 + \lambda \|\beta\|_1$ subject to $\|\beta\|_0 \leq pr \times C$. Then train $\arg\min_{W'} \left\| Y - X'(W')^T \right\|_F^2$ for one step and reshape W' back to W. Finally, assign $\beta_i \leftarrow \beta_i \|W_i\|_F$ and $W_i \leftarrow \frac{W_i}{\|W_i\|_F}$.

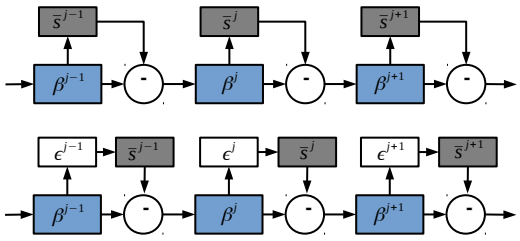


Figure 2: W and H dimensional pruning: integrating traditional pruning (Han, Mao, and Dally 2015) with information dropout.

Pruning from H and W Dimensions

Figure 2 shows the pruning flow from H and W dimensions. In particular, the scalar mask of the ith weight β_i for a small value is set to zero if the weights are pruned based on their magnitudes. These small weights are moved to $\bar{s}^i$, defined as a set of pruned weights. If the weights are pruned based on information dropout, discussed later in this subsection, the scalar mask of the ith weight β_i will be moved to $\bar{s}^i$ with probability ϵ^i. The weights that play more important role in reducing the classification error are less likely to be pruned.

In addition to the static pruning method, another way to compress a dense high-accuracy baseline model is to adopt stochastic pruning, *i.e.,* stochastically deleting some unimportant weights from the H and W dimensions. In other words, elements will be dropped randomly from each kernel in the H and W dimensions during fine-tuning. In each rollout, pruning all layers is then followed by iDropPruning, namely, finetuning with kernel-wise pruning using information dropout. The proposed method is to fine-tune the pruned model for accuracy improvement and further prune in the H and W dimensions via information dropout to achieve deep model compression. We next show how the information dropout addresses redundancy among different kernels.

In information dropout, we propose a solution to: (1) efficiently approximate posterior inference of the latent variable $\mathbf{z}$ given an observed value $\mathbf{x}$ based on parameter θ, where $\mathbf{z}$ is a representation of $\mathbf{x}$ and defined as some (possibly nondeterministic) function of $\mathbf{x}$ that has some desirable properties in classification task $\mathbf{y}$,[1] and (2) efficiently approximate marginal inference of the variable $\mathbf{x}$ to allow for various inference tasks where a prior over $\mathbf{x}$ is required.

Without loss of generality, let us consider Bayesian analysis of some dataset $\mathcal{D} = \left\{ (\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \right\}_{i=1}^{N}$ consisting of N i.i.d samples of some discrete variable $\mathbf{x}$. We assume that the data are generated by some random process, involving an unobserved continuous random variable $\mathbf{z}$. Bayesian inference in such a scenario consists of (1) updating some initial belief over parameters $\mathbf{z}$ in the form of a prior distribution $p_{\theta^\star}(\mathbf{z})$, and (2) a belief update over these parameters in the form of (an approximation to) the posterior distribution $p_\theta(\mathbf{z}|\mathbf{x})$ after observing data $\mathbf{x}$. In *variational inference* (Kingma and Welling 2013), inference is considered as an optimization problem where we optimize the parameters θ of some parameterized model $p_\theta(\mathbf{z})$ such that $p_\theta(\mathbf{z})$ is a close approximation of $p_\theta(\mathbf{z}|\mathbf{x})$ as measured by the KL divergence $D_{KL}(p_\theta(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}))$. The divergence between $p_\theta(\mathbf{z}|\mathbf{x})$ and the true posterior is minimized by minimizing the negative variational lower bound $\mathcal{L}(\theta)$ of the marginal likelihood of the data, namely,

$$\mathcal{L}(\theta, \theta^\star; \mathbf{x}^{(i)}) = - \sum_{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \in \mathcal{D}} \mathbb{E}_{p_\theta(\mathbf{z}|\mathbf{x}^{(i)})} [\log p(\mathbf{y}^{(i)}|\mathbf{z})] + \beta D_{KL}(p_\theta(\mathbf{z}|\mathbf{x}^{(i)})||p_{\theta^\star}(\mathbf{z})). \quad (5)$$

As shown in (Kingma, Salimans, and Welling 2015), the neural network weight parameters θ are less likely to overfit the training data if adding input noise during training. We propose to represent $\mathbf{z}$ by computing a deterministic map of activations $f(\mathbf{x})$, and then multiply the result in an element-wise manner by a random noise ξ, drawn from a parametric distribution p_α with the variance that depends on the input data $\mathbf{x}$, as

$$\mathbf{z} = (\mathbf{x} \circ \xi)\theta, \qquad \xi_{i,j} \sim p_{\alpha_\theta(\mathbf{x})}(\xi_{i,j}), \quad (6)$$

where $\circ$ denotes the element product operation of two vectors. A choice for the distribution $p_{\alpha_\theta(\mathbf{x})}(\xi)$ is the log-normal distribution $\log(p_{\alpha_\theta(\mathbf{x})}(\xi_{i,j})) = \mathcal{N}(0, \alpha_\theta^2(\mathbf{x}))$ (Achille and Soatto 2018) that makes the normally fixed dropout rates p_α adaptive to the input data, namely,

$$\log(p_{\alpha_\theta(\mathbf{x})}(\xi_{i,j})) \sim \mathcal{N}(\mathbf{z}; 0, \alpha_\theta^2(\mathbf{x})I),$$
$$\log(p_{\theta^\star}(\mathbf{z})) \sim \mathcal{N}(\mathbf{z}; \mu, \sigma^2 I), \quad (7)$$

[1] This is useful for coding tasks.

where $\alpha_\theta(\mathbf{x})$ is an unspecified function of $\mathbf{x}$. The resulting estimator becomes

$$\mathcal{L}(\theta;\mathbf{x}^{(i)}) \simeq \frac{1}{N}\sum_{l=1}^{N}\left[-\log p(y^{(i)}|\mathbf{z}^{(i,l)})\right] + \beta\left[\frac{1}{2\sigma^2}(\alpha_\theta^2(x^{(i)})+\mu^2) - \log\frac{\alpha_\theta(x^{(i)})}{\sigma} - \frac{1}{2}\right], \quad (8)$$

where $\mathbf{z}^{(i,l)} \sim (\mathbf{x}^{(i)} \circ \epsilon^{(i,l)})\theta$ and $\epsilon^{(l)} \sim p_\alpha(\epsilon) = \log\mathcal{N}(0,\alpha_\theta^2(\mathbf{x}))$. This loss can be optimized using stochastic gradient descent. A pseudocode of iDropPruning is shown in Algorithm 1 and an illustrative experiment is given in Section 8.

Algorithm 1 Fine-tuning with Kernel-wise Pruning using Information Dropout (iDropPruning)

Require: Pruned model parameters at this iteration θ, the number of fine-tuning iterations Z, learning rate γ and decay of learning rate τ.

Ensure: Further compressed and tuned model parameters θ

for z = $1, \cdots, Z$ **do**

Randomly choose a mini-batch of samples from the training set

Compute gradient of $\mathcal{L}(\theta;\mathbf{x}^{(i)})$ by $\frac{\partial\mathcal{L}(\theta;\mathbf{x}^{(i)})}{\partial\theta}$, where $\mathcal{L}(\theta;\mathbf{x}^{(i)})$ is computed by (8)

Update θ using $\theta \leftarrow \theta - \gamma\frac{\partial\mathcal{L}(\theta;\mathbf{x}^{(i)})}{\partial\theta}$

$\gamma \leftarrow \tau\gamma$

end for

Time Complexity

Time complexity analysis of the proposed DRL-supported compression structure is given in Section 9.

4 Quantization

Given a pre-defined full-precision model, the learner inserts quantization nodes and operations into the computation graph of the model. With activation quantization enabled, quantization nodes will be placed after activation operations such as ReLU(Ramachandran, Zoph, and Le 2017).

In quantization-aware training, both full-precision and quantized weights are kept. In the forward pass, quantized weights are obtained by applying the quantization function on the full-precision weights. To address that the partial derivative of the full-precision weights with respect to the quantized weights are zeros almost everywhere, we use the straight-through estimator (Bengio, Léonard, and Courville 2013) to pass gradients of the quantized weights directly to the full-precision weights.

In the proposed DRL-based quantization-aware training, the RL agent automatically searches for the optimal bit allocation strategy for each layer. The modeling of quantization state, action, and rewards are defined in Section 2. The DRL algorithm is the same as the one for pruning in Section 3 and its pseudo code is given Algorithm 2.

5 Experiments

In all experiments, the CIFAR-10 dataset is divided by 50000 samples for training, 5000 samples for validation, and 10000 samples for evaluation. The ILSVRC-12 dataset is divided by 1281167 samples for training, 10000 samples for validation, and 50000 samples for evaluation. We adopt a neural network policy with one hidden layer of size 64 and one fully-connected layer using Sigmoid as the activation function. We use the proximal policy optimization clipping algorithm with $c = 0.2$ as the optimizer and the initial KL penalty coefficient is set as $\alpha = 1$. The critic also has one hidden layer of size 64. The discounting factor is selected as $\gamma = 0.99$ and the targeted KL-divergence is selected as $\delta = 0.2$. The learning rate of the actor and the critic is set as 1×10^{-3}. In CIFAR-10, the per GPU batch size for training is 128 and the batch size for evaluation is 100. In ILSVRC-12, the per GPU batch size for training is 64 and the batch size for evaluation is 100. The fine tuning steps for each layer are selected as 2000. The parameters are optimized using the SGD with momentum algorithm (Sutskever et al. 2013). For CIFAR-10, the initial learning rate is set as 0.1 for ResNet and VGGNet. For ILSVRC-12, the initial learning rate is set as 1 and divided by 10 at rollouts $30, 60, 80$, and 90. The decay of learning rate is set to 0.99. All experiments were performed using TensorFlow, allowing for automatic differentiation through the gradient updates (Abadi et al. 2016), on 8 NVIDIA Tesla K80 GPUs.

CIFAR-10

The CIFAR-10 dataset (Krizhevsky, Nair, and Hinton 2014) consists of images with a 32×32 resolution. Table 1 shows the performance of the proposed method. It can be observed that the proposed method can not only reduce model size but also improve the accuracy (*i.e.,* reduce error rate).

In addition, we compare our algorithm with the commonly adopted weight magnitude channel selection strategy and channel pruning strategy to demonstrate the importance of iDropPruning. Please refer to the Section 10 for more details.

Table 1: VGGNet and ResNet-152 on CIFAR-10 dataset

Model	Error (%)	Para.	Pruned Para. (%)	FLOPs (%)
VGGNet(Baseline)	6.54	20.04M	0	100
VGGNet	6.33	2.20M	89.0	48.7
VGGNet	6.20	2.29M	88.6	49.1
ResNet-152(Baseline)	5.37	1.70M	0	0
ResNet-152	5.19	1.30M	23.5	71.2
ResNet-152	5.33	1.02M	40.0	55.1

ImageNet

To evaluate the effect of different pruning rates a_H^t, we select 30%, 50%, 60%, and 70% for ResNet-18 and ResNet-50 and then evaluate the model pruning on ImageNet ILSVRC-2012 dataset (Deng et al. 2012). Experimental results are shown in Table 2 while the per-layer weight bits policy for the quantization is shown in Figure 3. From Table 2, it can be seen that the error increases as the pruning rate increases. However, our pruned ResNet-50 with 30% pruning rate outperforms the

Algorithm 2 DRL-supported compression structure

Require: Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weight θ^Q and θ^μ. Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^{\mu^-}$. Initialize replay buffer $\mathcal{D}$. Initialize target KL-divergence δ and hyperparameter c reflecting how far away the new policy is allowed to deviate from the prior.

Ensure: Target network weights $\theta^{Q'}, \theta^{\mu'}$

for episode = $1, \cdots, M$ **do**

Initialize a random process $\mathcal{N}$ for action exploration

Receive initial observation state s_1

for t = $1, \cdots, T$ **do**

Select action $a_t = clip(\mu(s_t|\theta^{\mu^-}) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \in \mathcal{N}$

Execute action a_t and observe reward r_t and the new state s_{t+1}

Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{R}$

Sample a random minibatch of transitions $B = \{(s_i, a_i, r_i, s_{i+1})\}$ from $\mathcal{R}$

Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$

Update critic by one step of gradient descent using:

$$\bigtriangledown_{\theta^Q} \frac{1}{|B|} \sum_{(s_i,a_i,r_i,s_{i+1})\in B} (y_i - Q(s_i, a_i|\theta^Q))^2$$

Update the actor policy by maximizing the KL Penalty PPO via stochastic gradient ascent with Adam:

$$\theta^\mu = \arg\max_\theta \frac{1}{|B|} \sum_{(s_i,a_i,r_i,s_{i+1})\in B} \left[\frac{\pi_\theta(a|s_i)}{\pi^{\theta^{\mu^-}}(a|s_i)} Q(s_i, a|\theta^Q)|_{a=\mu(s_i|\theta^{\mu^-})} - \alpha KL[\pi_{\theta^{\mu^-}}(\cdot|s_i), \pi_\theta(\cdot|s_i)] \right]$$

or update the policy by maximizing the PPO-Clip objective:

$$\theta^\mu = \arg\max_\theta \frac{1}{|B|} \sum_{(s_i,a_i,r_i,s_{i+1})\in B} \min \left(\frac{\pi_\theta(a|s_i)}{\pi^{\theta^{\mu^-}}(a|s_i)} Q(s_i, a|\theta^Q)|_{a=\mu(s_i|\theta^{\mu^-})}, clip(\frac{\pi_\theta(a|s_i)}{\pi^{\theta^{\mu^-}}(a|s_i)}, 1-c, 1+c) Q(s_i, a|\theta^Q)|_{a=\mu(s_i|\theta^{\mu^-})} \right)$$

if $\frac{1}{|B|} \sum_{(s_i,a_i,r_i,s_{i+1})\in B} KL[\pi_{\theta^{\mu^-}}(\cdot|s), \pi_\theta(\cdot|s)]|_{s=s_i}] \geq 1.5\delta$ **then**

$\alpha = 2\alpha$

else if $\frac{1}{|B|} \sum_{(s_i,a_i,r_i,s_{i+1})\in B} KL[\pi_{\theta^{\mu^-}}(\cdot|s), \pi_\theta(\cdot|s)]|_{s=s_i}] \leq 1.5\delta$ **then**

$\alpha = \frac{\alpha}{2}$

end if

Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{Q^{\mu'}} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

end for

end for

Table 2: ResNet-18 and ResNet-50 on ImageNet with different speed-ups.

Model	Top-1/Top-5 Error (%)	Pruned Para. (%)	FLOPs (%) Pruning	Speed-up× Pruning + Quantization
ResNet-18(Baseline)	29.36/10.02	0	100	1
ResNet-18	30.29/10.43	30.0	71.4	11.4
ResNet-18	30.65/11.93	50.0	44.2	16.0
ResNet-18	33.40/13.37	66.7	29.5	28.2
ResNet-50(Baseline)	24.87/6.95	0	100	1
ResNet-50	23.42/6.93	30.0	66.7	12.0
ResNet-50	24.21/7.65	50.0	47.6	16.0
ResNet-50	28.73/8.37	75.3	27.0	29.6

Table 3: MobileNet-v1 on ILSVRC-12

Model	FLOPs (%)	Error (%): RL + channel pruning	Error (%): RL + iDropPruning	Pruned Para.
MobileNet-v1 (Baseline)	100	32.0	32.0	0
MobileNet-v1	50	30.2, ΔACC% $= -0.2$	31.9, ΔACC% $= +0.1$	45.8
MobileNet-v1	40	33.1, ΔACC% $= -1.1$	32.8, ΔACC% $= -0.8$	55.2
MobileNet-v1 (DDPG) (He et al. 2018)	50	ΔACC% $= -0.4$		
MobileNet-v1 (DDPG) (He et al. 2018)	40	ΔACC% $= -1.7$		
0.75 MobileNet-224 (Uniform) (Howard et al. 2017)	56	ΔACC% $= -2.5$		
0.75 MobileNet-224 (Uniform) (Howard et al. 2017)	41	ΔACC% $= -3.7$		

pre-trained baseline model in the top-1 error and our pruned ResNet-50 with 30% and 50% pruning rate outperforms the pre-trained baseline model in both the top-1 and top-5 errors.

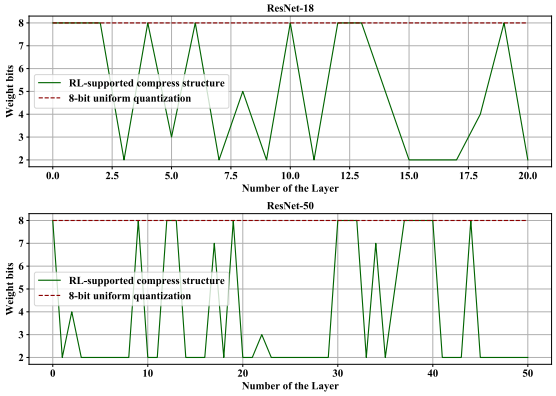
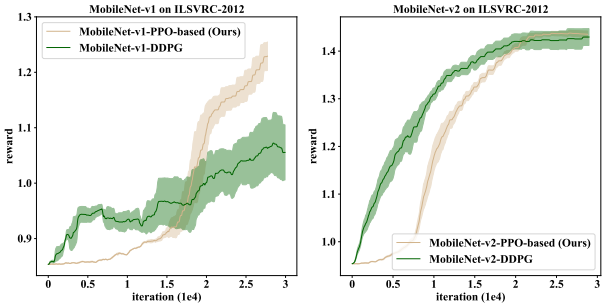


Figure 3: ResNet-18 and ResNet-50 on ImageNet with different bit allocation strategies. The 8-bit uniform quantization strategy is shown in orange line, and the RL-supported policy is shown in blue line. The RL-supported policy generates a more compressed model with a faster inference speed. By observing the RL-supported policy, the 3×3 layer is more important than the 1×1 layers because the 1×1 layers are represented by less bits.

To show the importance of our DRL-supported compression structure with iDropPruning, we compare RL with channel pruning and RL with iDropPruning. Table 3 shows that RL with channel pruning can find the optimal layer-wise pruning rates while RL with iDropPruning can further decrease the testing error of the compressed model. A comparison of the reward $r1$ for AMC (He et al. 2018) (DDPG-based pruning) and our proposed method (PPO-based pruning) is shown in Figure 4. All experiments are executed from 5 runs. Even if the average performance is similiar in MobileNet-v2(4b), DDPG still has a higher variance, which leads to the instability of policy. We report the results for ILSVRC-12 on both MobileNet-v1 on Table 3 and MobileNet-v2 on Table 4.

We further examine the results when applying both pruning and quantization on ILSVRC-12. We use the VGG-16 model with 138 million parameters as the reference model. Table 5 shows that VGG-16 can be compressed to 3.0% of its original size (*i.e.,* $33\times$ speed-up) when weights in the convolution layers are represented with 8 bits, and fully-connected layers



(a) MobileNet-v1 (b) MobileNet-v2

Figure 4: Comparison of RL-based pruning methods for MobileNet-v1 on ILSVRC-2012.

use 5 bits. In addition, the compressed model outperforms the baseline model in both the top-1 and top-5 errors.

6 Conclusion

Using hand-crafted features to get compressed models requires domain experts to explore a large design space and the trade-off among model size, speed-up, and accuracy, which is often suboptimal and time-consuming. This paper proposed a deep model compression method that uses reinforcement learning to automatically search the action space, improve the model compression quality, and use the FLOPs obtained from fine-tuning with information dropout pruning for the further adjustment of the policy to balance the trade-off among model size, speed-up, and accuracy. Experimental results were conducted on CIFAR-10 and ImageNet to achieve $4 \times -33\times$ model compression with limited or no accuracy loss, proving the effectiveness of the proposed method.

References

[Abadi et al. 2016] Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation*, 265–283.

[Achille and Soatto 2018] Achille, A., and Soatto, S. 2018. Information dropout: Learning optimal representations through noisy computation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40(12):2897–2905.

[Baker et al. 2016] Baker, B.; Gupta, O.; Naik, N.; and Raskar, R.

Table 4: Comparison with other RL based compression methods on ILSVRC-12

Model	Policy	FLOPs	Δ acc%
Pruning on MobileNet-V2 (baseline)	Uniform (0.75MobileNet-224) (Sandler et al. 2018)	70%	-2.0
	DDPG (AMC) (He et al. 2018)	70%	-1.0
Pruning on MobileNet-V2* (Ours)	Pruned Parameters	FLOPs	Δacc%
PPO based RL+iDropPruning (Ours)	65.9%	21%	-2.4
PPO based RL+iDropPruning (Ours)	53.2%	59%	-1.0
PPO based RL+iDropPruning (Ours)	18.6%	70%	-0.8
Quantization on MobileNet-V2**	Model size	acc	
Deep Compression (Han, Mao, and Dally 2015)	0.96MB	58.07%	
HAQ (Wang et al. 2019)	0.95MB	66.75%	
Ours	0.89MB	66.80%	
Ours	0.81MB	66.40%	

Table 5: Comparison with another non-RL three-stage compression method (VGG-16 on ILSVRC-12)

Method	Layer	Parameters	Pruned Para. (%)	Weight bits Pruning + Quantization	Speed-up Pruning + Quantization
Ours	conv1_1 / conv1_2	2K / 37K	42 / 89	8 / 8	2.5× / 10.2×
	conv2_1 / conv2_2	74K / 148K	72 / 69	8 / 8	7.0× / 6.8×
	conv3_1 / conv3_2 / conv3_3	295K / 590K / 590K	50 / 76 / 58	8 / 8 / 8	4.6× / 10.3× / 5.9×
	conv4_1 / conv4_2 / conv4_3	1M / 2M / 2M	68 / 88 / 76	8 / 8 / 8	7.6× / 9.1× / 7.2×
	conv5_1 / conv5_2 / conv5_3	2M / 2M / 2M	70 / 76 / 69	8 / 8 / 8	7.1× / 8.5× / 7.1×
	fc_6 / fc_7 / fc_8	103M / 17M / 4M	96 / 96 / 77	5 / 5 / 5	62.5× / 66.7× / 14.1×
	Total	138M	93.1	5	33×
Deep Compression(Han, Mao, and Dally 2015)	conv1_1 / conv1_2	2K / 37K	42 / 78	5 / 5	2.5× / 10.2×
	conv2_1 / conv2_2	74K / 148K	66 / 64	5 / 5	6.9× / 6.8×
	conv3_1 / conv3_2 / conv3_3	295K / 590K / 590K	47 / 76 / 58	5 / 5 / 5	4.5× / 10.2× / 5.9×
	conv4_1 / conv4_2 / conv4_3	1M / 2M / 2M	68 / 73 / 66	5 / 5 / 5	7.6× / 9.2× / 7.1×
	conv5_1 / conv5_2 / conv5_3	2M / 2M / 2M	65 / 71 / 64	5 / 5 / 5	7.0× / 8.6× / 6.8×
	fc_6 / fc_7 / fc_8	103M / 17M / 4M	96 / 96 / 77	5 / 5 / 5	62.5× / 66.7× / 14.0×
	Total	138M	92.5	5	31×

2016. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*.

[Bengio, Léonard, and Courville 2013] Bengio, Y.; Léonard, N.; and Courville, A. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

[Chen et al. 2015] Chen, W.; Wilson, J.; Tyree, S.; Weinberger, K.; and Chen, Y. 2015. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, 2285–2294.

[Chollet 2017] Chollet, F. 2017. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1251–1258.

[Courbariaux et al. 2016] Courbariaux, M.; Hubara, I.; Soudry, D.; El-Yaniv, R.; and Bengio, Y. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to + 1 or -1. *arXiv preprint arXiv:1602.02830*.

[Deng et al. 2012] Deng, J.; Berg, A.; Satheesh, S.; Su, H.; Khosla, A.; and Li, F.-F. 2012. Ilsvrc-2012. *URL http://www.image-net.org/challenges/LSVRC*.

[Denton et al. 2014] Denton, E. L.; Zaremba, W.; Bruna, J.; LeCun, Y.; and Fergus, R. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, 1269–1277.

[Han et al. 2015] Han, S.; Pool, J.; Tran, J.; and Dally, W. 2015. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, 1135–1143.

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

[He et al. 2016] He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.

[He et al. 2018] He, Y.; Lin, J.; Liu, Z.; Wang, H.; Li, L.-J.; and Han, S. 2018. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision*, 784–800.

[He, Zhang, and Sun 2017] He, Y.; Zhang, X.; and Sun, J. 2017. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, 1389–1397.

[Hinton and Van Camp 1993] Hinton, G., and Van Camp, D. 1993. Keeping neural networks simple by minimizing the description length of the weights. In *Proceedings of the ACM Conference on Computational Learning Theory*. Citeseer.

[Hinton, Vinyals, and Dean 2015] Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

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

[Huang et al. 2017] Huang, G.; Liu, Z.; Van Der Maaten, L.; and Weinberger, K. Q. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4700–4708.

[Huang et al. 2018] Huang, G.; Liu, S.; Van der Maaten, L.; and Weinberger, K. Q. 2018. Condensenet: An efficient densenet using learned group convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2752–2761.

[Jia et al. 2018] Jia, H.; Xiang, X.; Fan, D.; Huang, M.; Sun, C.;

Meng, Q.; He, Y.; and Chen, C. 2018. Droppruning for model compression. *arXiv preprint arXiv:1812.02035*.

[Kavukcuoglu et al. 2010] Kavukcuoglu, K.; Sermanet, P.; Boureau, Y.-L.; Gregor, K.; Mathieu, M.; and Cun, Y. L. 2010. Learning convolutional feature hierarchies for visual recognition. In *Advances in Neural Information Processing Systems*, 1090–1098.

[Kingma and Welling 2013] Kingma, D. P., and Welling, M. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

[Kingma, Salimans, and Welling 2015] Kingma, D. P.; Salimans, T.; and Welling, M. 2015. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, 2575–2583.

[Krizhevsky, Nair, and Hinton 2014] Krizhevsky, A.; Nair, V.; and Hinton, G. 2014. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html* 55.

[Lebedev et al. 2014] Lebedev, V.; Ganin, Y.; Rakhuba, M.; Oseledets, I.; and Lempitsky, V. 2014. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*.

[Liu et al. 2017] Liu, Z.; Li, J.; Shen, Z.; Huang, G.; Yan, S.; and Zhang, C. 2017. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, 2736–2744.

[Liu et al. 2018] Liu, C.; Zoph, B.; Neumann, M.; Shlens, J.; Hua, W.; Li, L.-J.; Fei-Fei, L.; Yuille, A.; Huang, J.; and Murphy, K. 2018. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 19–34.

[Ramachandran, Zoph, and Le 2017] Ramachandran, P.; Zoph, B.; and Le, Q. V. 2017. Searching for activation functions. *arXiv preprint arXiv:1710.05941*.

[Rastegari et al. 2016] Rastegari, M.; Ordonez, V.; Redmon, J.; and Farhadi, A. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Proceedings of the European Conference on Computer Vision*, 525–542. Springer.

[Real et al. 2018] Real, E.; Aggarwal, A.; Huang, Y.; and Le, Q. V. 2018. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*.

[Sandler et al.] Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. arxiv 2018. *arXiv preprint arXiv:1801.04381*.

[Sandler et al. 2018] Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4510–4520.

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

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

[Sutskever et al. 2013] Sutskever, I.; Martens, J.; Dahl, G.; and Hinton, G. 2013. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning*, 1139–1147.

[Sutton et al. 2000] Sutton, R. S.; McAllester, D. A.; Singh, S. P.; and Mansour, Y. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, 1057–1063.

[Veit and Belongie 2018] Veit, A., and Belongie, S. 2018. Convolutional networks with adaptive inference graphs. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 3–18.

[Wang et al. 2019] Wang, K.; Liu, Z.; Lin, Y.; Lin, J.; and Han, S. 2019. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 8612–8620.

[Wu et al. 2018] Wu, J.; Zhang, Y.; Bai, H.; Zhong, H.; Hou, J.; Liu, W.; Huang, W.; and Huang, J. 2018. Pocketflow: An automated framework for compressing and accelerating deep neural networks.

[Yu, Yu, and Ramalingam 2018] Yu, X.; Yu, Z.; and Ramalingam, S. 2018. Learning strict identity mappings in deep residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4432–4440.

[Zhang et al. 2018] Zhang, X.; Zhou, X.; Lin, M.; and Sun, J. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 6848–6856.

[Zhong et al. 2018] Zhong, Z.; Yan, J.; Wu, W.; Shao, J.; and Liu, C.-L. 2018. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2423–2432.

[Zhu and Gupta 2017] Zhu, M., and Gupta, S. 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*.

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

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

[Zoph et al. 2018] Zoph, B.; Vasudevan, V.; Shlens, J.; and Le, Q. V. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 8697–8710.

Supplemental Materials: Deep Model Compression via Deep Reinforcement Learning

7 Demonstration of 3D Model Compression in the C, H, and W Dimensions

To explain the main idea of the proposed 3D model compression in the C, H, and W dimensions, we next show a simple yet illustrative example, shown in Figure S1, to demonstrate how pruning is conducted in the three dimensions. In Figure S1, the kernels are shown in yellow. The feature maps are shown in grey. The pruned components are shown in white dotted blocks. In the C dimensional pruning, Figure S1 shows a situation when the first two channels of feature map $\mathbf{z}$ are removed. We can then remove the corresponding two channels of the kernels α, shown as white dotted blocks, that take these channels as input. Kernels S_1 and S_2 that produce these channels can also be removed. Afterwards, in H and W dimensional pruning, Figure S1 shows one situation when kernel S_C is further pruned by iDropPruning. In other words, after C dimensional pruning, the CNN is further compressed by H and W dimensional pruning.

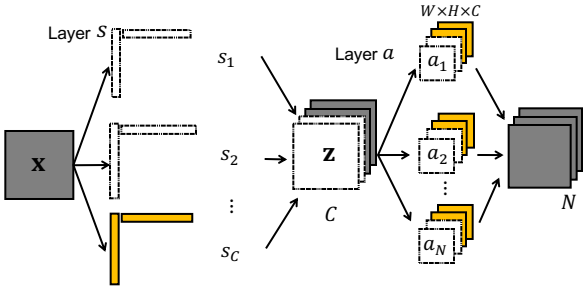


Figure S1: An example of CNN model compression by C, H and W dimensional pruning.

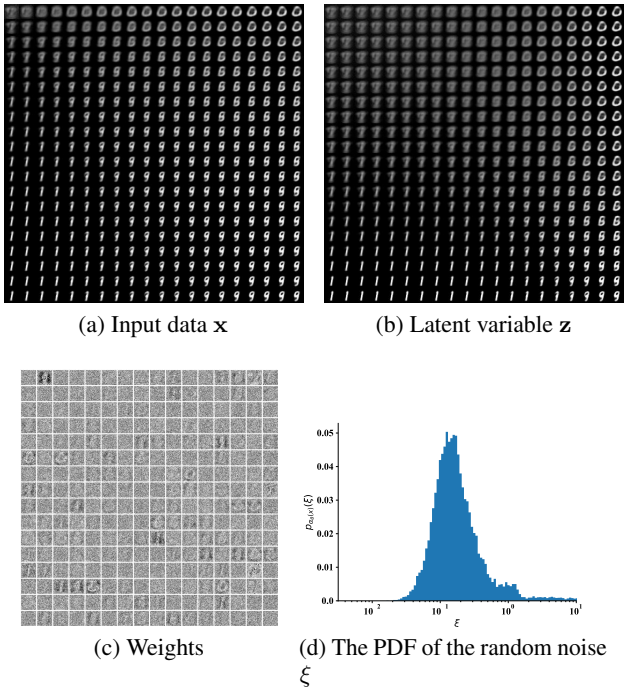
8 Information Dropout Experiment

The goal of this illustrative experiment is to validate the approach in Subsection 3 and show that our regularized loss function $\mathcal{L}(\theta; \mathbf{x}^{(i)})$ shown in (8) can automatically adapt to the data and can better exploit architectures for further compression. The random noise ξ is drawn from a distribution $p_{\alpha_\theta(\mathbf{x})}(\xi)$ with a unit mean u, $u = 1$, and a variance $\alpha_\theta(\mathbf{x})$ that depends on the input data $\mathbf{x}$. The variance $\alpha_\theta(\mathbf{x})$ is parameterized by θ. To determine the best allocation of parameter θ to minimize the KL-divergence term $D_{KL}(p_\theta(\mathbf{z}|\mathbf{x})||p_{\theta^\star}(\mathbf{z}))$, we still need to have a prior distribution $p_{\theta^\star}(\mathbf{z})$. The prior distribution is identical to the expected distribution of the activation function $f(\mathbf{x})$, which represents how much data $\mathbf{x}$ lets flow to the next layer. For a network that is implemented using the softplus activation function, a log-normal distribution is a good fit for the prior distribution (Achille and Soatto 2018). After we fix this prior distribution as $\log(p_{\theta^\star}(\mathbf{z})) \sim \mathcal{N}(0, 1)$, the loss in (8) can be computed using stochastic gradient descent to back-propagate the gradient through the sampling of $\mathbf{z}$ to obtain the optimized parameter θ. Even if $\log(p_{\theta^\star}(\mathbf{z})) \sim \mathcal{N}(0, 1)$, the actual value of u is not necessarily equal to 1 during the runtime. Hence, the mean u and the variance $\alpha_\theta(\mathbf{x})$ of the random noise ξ can be computed via solving the following two equations

$$E(\xi) = e^{u + \frac{\alpha^2_\theta(\mathbf{x})}{2}},$$
$$D(\xi) = (e^{\alpha^2_\theta(\mathbf{x})} - 1)e^{2u + \alpha^2_\theta(\mathbf{x})},$$

where $E(\xi)$ is the mean of sampled ξ and $D(\xi)$ is the variance of sampled ξ. We add a constraint, $\alpha_\theta(\mathbf{x}) \leq 0.8$, to avoid a large noise variance. Figure S2d shown the probability density function (PDF) of the noise parameter ξ by experiment, which matches a log-normal distribution with $\alpha_\theta(\mathbf{x}) = 0.5$. The result shows that we optimize the parameters θ of the parameterized model $p_{\theta^\star}(\mathbf{z})$ such that $p_{\theta^\star}(\mathbf{z})$ is a close approximation of $p_\theta(\mathbf{z}|\mathbf{x})$ as measured by the KL divergence $D_{KL}(p_\theta(\mathbf{z}|\mathbf{x})||p_{\theta^\star}(\mathbf{z}))$.

After the noise distribution is known, the distribution of $p_\theta(\mathbf{z}|\mathbf{x})$ in (5) can be obtained. In order to show how much information from images that information dropout is transmitting to the second layer, Figure S2b shows the latent variable $\mathbf{z}$ while Figure S2c shows the weights. As shown in Figure S2b, the network lets through the input data (Figure S2a).



(a) Input data $\mathbf{x}$ (b) Latent variable $\mathbf{z}$

(c) Weights (d) The PDF of the random noise ξ

Figure S2: An illustrative information dropout experiment. Figure S2a shows the input data $\mathbf{x}$. Figure S2b shows the plot of the latent variable $\mathbf{z}$ at a choice of parameter θ at each spatial location in the third information dropout layer of LeNet trained on MNIST with $\beta = 1$. The resulting representation $\mathbf{z}$ is robust to nuisances, and provides good performance. Figure S2c shows the weights. Figure S2d shows the PDF of the noise parameter ξ.

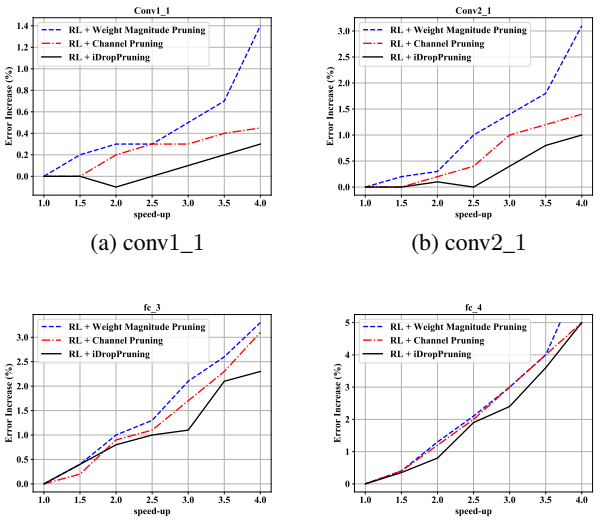
9 Time Complexity

As described in Figure S1, a single convolutional layer with N kernels requires evaluating a total number of NC of the $2D$ kernels $W_n^c * z^c$: $F = \{W_n^c \in \mathbb{R}^{d \times d} | n = 1, \cdots, N; c = 1, \cdots, C\}$. Note that there are N kernels $F = \{W_n^c | n = 1, \cdots, N\}$ operating on each input channel z^c with cost $O(NCd^2HW)$. The CNN pruning via information dropout involves computing a total number of NC' of the $2D$ kernels $W_n^{c'} * z^{c'}$ with cost $O(NC'd^2HW)$, indicating that efficiency inference requires that $C' \ll C$. In Subsection 3, we consider ameliorating the inference efficiency by

information dropout. As shown in Figure 1, in the kernels $s^c = \{s_m^c | m = 1, \cdots, M\}$, the cost can be reduced to $O(NC'dHW)$. Figure S3 shows an illustrative comparison.

10 Single Layer Acceleration Performance

In order to further show the importance of the Algorithm 1 after obtaining the optimized pruning rate based on the same DRL supported compression structure, we define a simple 4-layer convolutional neural network, including 2 convolution (conv) layers and 2 fully connected (fc) layers, for image classification on the CIFAR-10 dataset under the proposed DRL structure. We evaluate single layer acceleration performance using the proposed iDropPruning algorithm in Section 3 and compare it with the weight magnitude channel pruning strategy, *i.e.,* pruning channels based on the weights, to demonstrate the advantages of iDropPruning. Figure S3 shows the performance comparison measured by the error increase after a certain layer is pruned. By analyzing this figure, we can observe that the proposed policy considers the fully-connected layers more important than the convolutional layers because the error increase for fully-connected layers is typically larger under the same compression rate.



(a) conv1_1 (b) conv2_1

(c) fc_3 (d) fc_4

Figure S3: Single layer error increase under different compression rates. To verify the importance of iDropPruning in Subsection 3, we considered two baselines: (1) RL + Channel Pruning, and (2) RL + iDropPruning.