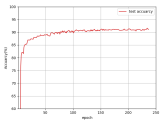
DRL lr	CNN lr	Acc.
10^{-6}	10^{-3}	**91.23**
10^{-6}	10^{-2}	10.00
10^{-6}	10^{-4}	90.67
10^{-5}	10^{-3}	89.95
10^{-7}	10^{-3}	90.79

Reviewer #1: Q1: The number of iterations ... **A1:** We did not present values of all the hyper-parameters in submission due to the limit of space. Hyper-parameters are described as follows. The batch size for training is 128. For CIFAR-10, we train ($N_1 = 1560$) batches (4 epochs) for DRL agents (corresponding to line 12-15 in Algorithm 1) and ($N_2 = 780$) batches (2 epochs) for $h(\cdot)$, v, $f(\cdot)$ and CNN (line 16-19 in Algorithm 1) at each iteration. The total number of iterations is 40 (for line 1-19 in Algorithm 1). For ImageNet, at each iteration of training, $N_1 = 1200$ and $N_2 = 600$ except last iteration. The total number of iterations is 64. At last iteration, $N_2 = 200,000$ for finetuning the CNN. The learning rate for DRL agent is 10^{-6}. Learning rate for CNN is 10^{-3} in CIFAR-10 and 10^{-4} in ImageNet.

We show experiment results on CIFAR-10 with various learning rate setting in table on the right side. The sparsity is 0.5 and $R_r = 0.5$. It shows that our learning rate setting is optimal.



Q2: The experimental results **A2:** We show the training curve of our method. We train on CIFAR-10 with sparsity 0.5 and $R_r = 0.5$. The y-aix is test accuracy and x-aix is the number of epochs. This figure shows that our method reaches optimal and robust during training.

Reviewer #2: Q1: Although combining ... **A1:** Our method combines static and runtime pruning under a DRL-based framework which determines sparsity of layers and addresses the storage efficiency problem of runtime pruning. Our method takes advantages of both static and runtime pruning to provide trade-off with better accuracy and lower storage.

Q2: DRL methods usually ... **A2:** For training cost of DRL, kindly refer to **Reviewer #1 A1**. The extra cost of DRL at inference is tiny comparing to convolutional layers. The extra cost of each layer includes the computation of layer-dependent encoder (described in paragraph **State** in Sec. 3.3) and runtime DRL agent. The layer-dependent encoder is a fully-connected layer where input size is input channel of this convolutional layer and output size is 128. The runtime DRL agent network consists of one layer of RNN with hidden state size of 128 and one layer fully-connected Actor network with output size of 1 (the mean of Gaussian policy). The MACs of layer-dependent encoder and runtime DRL agent is around 0.3% of pre-trained CNN.

Q3: DRL-based methods usually ... **A3:** We are preparing the code and will open-source after paper is accepted.

Reviewer #3: Q1: There are one... **A1:** The training process of Ref[22] is dynamic pruning but in inference it is static pruning. We will revise the related work to claim Ref[22] as static pruning and also update bibtex of Ref[22].

Q2: The "sparsity" of Table 1 and 2 means the ratio of preserved output channels after pruning at every layer. This "sparsity" setting is mainly for FBS[22] because FBS uses a same sparsity value for all convolutional layers. However, our proposed method predicts layer-specific sparsity ratios by DRL agents. Therefore, for our method in Table 1 and 2, we calculate the computation and storage budget constraints according to the "sparsity" value (which is 0.5 in Table 1 and 0.7 in Table 2), then our approach learns layer-specific sparsity ratio according to the constraints by DRL method.

Method	Acc.	$\Delta acc.$	GPU Time	CPU Time
FBS	89.88	-1.49	10.9 ms	172.0 ms
RNP	84.93	-7.14	11.1 ms	175.3 ms
FPGM	89.8	-2.27	**5.0 ms**	**48.2 ms**
ours ($R_r = 1$)	91.425	**-0.645**	11.2 ms	178.3 ms
ours ($R_r = 0.5$)	91.228	**-0.842**	9.8 ms	110.7

Q3: Wall clock ... **A3:** The "Inference Time" in Sec. 4 Table 1 is wall clock of running on GPU. We show the CPU wall clock and static pruning method FPGM in following table. We also show experiment result of FPGM which is static pruning approach in this table. Since it is static pruning method, it has smaller wall clock but lower accuracy compared to runtime pruning approach FBS and our method.

Method	Δ top-1 acc.	Δ top-5 acc.	Speed-up	#Params
DCP	-2.29	**-0.12**	1.71×	0.71×
FPGM	-1.87	-1.15	1.71×	0.72×
Dynamic Sparse Graph	-4.68	-	1.4 ×	-
CGNN	-1.07	-0.63	1.63×	-
FBS	-2.54	-1.46	1.98×	1.12×
AMC	-3.13	-1.88	2.00×	0.76×
Ours ($R_r = 0.5$)	**-1.03**	-0.43	1.94×	0.81×

Q4: One important ... **A4:** For the comparison of FPGM, please refer to **A3**. We report model size (number of parameters stored) of methods pruning ResNet-18 on ImageNet in following table. Static method can reduce more parameters (storage consumption) while runtime method FBS increases the number of parameters. Our method can reduce more storage compared to runtime method. The wall clock comparison of Table 4 and 5 in Sec. 4 is unavailable, for the method doesn't report actual time nor release codes for re-run. Although some methods released codes, it is not fair to compare wall clock because their implementation is not optimized for latency.

Reviewer #4: Q1: In Line 113, ... **A1:** We will carefully revise the definition of trade-off pruner in our next version of paper. Here let me clarify the trade-off pruner: In Sec. 3.2, we don't mentioned a_t^r and a_t^s because we use the mask $\mathbf{M}_r$ and $\mathbf{M}_s$ to represent the pruning results which are computed according to a_t^r and a_t^s. $\mathbf{M}_r$ is computed by $\mathbf{u}_r$ and a_t^r as mentioned in line 134-139 of Sec. 3.1 and $\mathbf{M}_s$ is computed by $\mathbf{u}_s$ and a_t^s as mentioned in line 163-166 of Sec. 3.1.

Q2: Some implementation ... **A2:** We use 1-layer RNN with hidden state size of 128. For more implementation details, please refer to **Reviewer #1 A1** and **Reviewer #2 A2** line 19 to 22.

Q3: All variables in Fig. 1 should ... **A3:** Thanks for your suggestion. We will update the font in Fig.1.

Q4: All vectors variables should ... **A4:** We will change decision mask to **m** and will carefully revise the matrix/vector variables in math equations.