CPrune: Compiler-Informed Model Pruning for Efficient Target-Aware DNN Execution (Supplementary Materials)

Taeho Kim[1], Yongin Kwon[2], Jemin Lee[2], Taeho Kim[2], and Sangtae Ha[1]

[1] University of Colorado Boulder
{taeho.kim,sangtae.ha}@colorado.edu
[2] Electronics and Telecommunications Research Institute
{yongin.kwon,leejaymin,taehokim}@etri.re.kr

1 Identifying α and β for Each Pruning Iteration

CPrune evaluates the execution time and accuracy of the pruned candidate model at each pruning iteration. Each iteration compares its performance with the thresholds obtained by multiplying α and β to the last execution time and accuracy, respectively. α is the ratio to represent the minimum allowable accuracy used during each iterative step of the pruning process, and β is the ratio to define the target execution time of the next pruning iteration. We conducted the following experiments to determine the α and β used in our experiments.

1.1 Identifying α

This experiment measures the accuracy and processed figures per second (FPS) according to various α and determines our experiment's α value. When the pruned candidate model of CPrune shows an execution time lower than the target execution time of the current iteration, it compares the accuracy of a pruned model with the threshold $(\alpha \cdot a_p)$. Then, if its accuracy is higher than the threshold, the pruned candidate model is selected as the pruned model of the current iteration.

We execute CPrune process for the ResNet-18 using ImageNet on Kryo 385 CPU and compare their final CPrune models when choosing different α ($\alpha \in \{0.9975, 0.995, 0.99\}$). We observe a clear trade-off between fast FPS and high accuracy based on the choice of α, as shown in Table 1. When $\alpha = 0.9975$, we can see that the FPS is improved, and the top-5 accuracy is slightly improved, as shown in Figure 1. On the other hand, when $\alpha = 0.99$, we obtain a high execution speed by sacrificing some accuracy. 0.9975 only marginally improves the FPS, and 0.99 makes its accuracy too low. Therefore, although the appropriate α varies depending on the accuracy requirement, we chose $\alpha = 0.995$ as a suitable heuristic to address the observed trade-off during the pruning process. The results in Table 1 confirm that we obtain significant increase rate in FPS ($1.38\times \sim 2.23\times$) by using CPrune with $\alpha = 0.995$.

[2] Yongin Kwon is the corresponding author.

Table 1: Mobile CPU (Kryo 385 and 585) and GPU (Mali-G72) performance test (ResNet-18, MobileNetV2, MnasNet1.0 with ImageNet dataset)

Model	α	FPS		FLOPS	Params	Top-1	Top-5
		TFLite	CPrune	(Pruned)	(Pruned)	Acc	Acc
ResNet-18 (Kryo 385)	Original	4.39	18.86	1.81B	11.7M	69.76%	89.08%
	0.9975	4.60 (1.05×)	25.99 (1.38×)	1.61B (11.0%)	11.0M (5.74%)	69.73%	89.21%
	0.995	5.94 (1.35×)	36.91 (1.96×)	1.17B (35.7%)	10.3M (11.8%)	68.30%	88.34%
	0.99	6.57 (1.50×)	42.09 (2.23×)	844M (53.5%)	7.17M (38.6%)	65.02%	86.28%

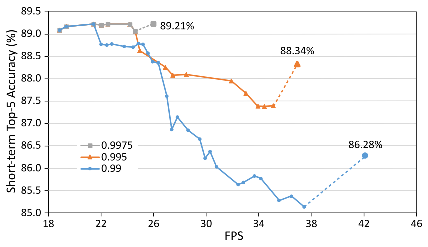

Fig. 1: Comparison of CPrune performance according to different α values. 0.9975 only marginally improves the FPS, and 0.99 makes its accuracy too low. Therefore we chose $\alpha = 0.995$.

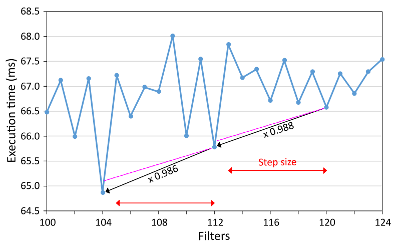

Fig. 2: Comparison of the results of measuring execution time while reducing the number of filters. The dotted line is the expected minimum execution time reduction with $\beta = 0.99$ when pruning the filters by the step size.

1.2 Identifying β

This experiment measures a DNN model's execution time after tuning while pruning the filters one by one and determines our experiment's β value. We pruned filters in one of the convolution layers of the ResNet-18 model. The experimental results using ImageNet on Kryo 280 CPU are shown in Figure 2. The step size in Figure 2 is the number of pruning filters decided by analyzing the corresponding subgraph in the model. When we prune the number of filters by the step size, its execution time is reduced by about 1% or more. Furthermore, we also confirmed that the execution time is reduced by $1 \sim 2\%$ in other layers after pruning the filters as much as their step size. In other words, if we decide the target execution time of the next pruning iteration as $0.99 \times$ the current execution time, it fulfills the execution time condition in the next pruning iteration with step size pruning. Therefore, we selected $\beta = 0.99$.

2 Details of Algorithm 1

This supplementary section includes additional information on Algorithm 1, including the computational complexity and a table that summarizes the variables used.

2.1 Computational Complexity

The algorithm consists of (1) finding a subgraph that could largely impact the DNN execution and (2) pruning that subgraph. Finding a subgraph has the worst-case running time of $O(T \cdot F)$ where T is the number of tasks, and F is the number of filters. Pruning a subgraph involves calculating the least common multiple (LCM) of two values, which results in $O(FlogF)$. Finally, the worst-case running time of CPrune becomes $O(T \cdot F \cdot FlogF)$.

2.2 Variables in Algorithm 1

Table 2: Notation

Symbol	**Description**
M	Current model
a_g	The minimum accuracy requirement
p_r	Pruning rate of each subgraph
l_t	Target execution time of the following pruning iteration
a_p	Short-term accuracy of the previous best model
C	Connection table among tasks, subgraphs and fast programs of M
R	List of tasks of M, prioritized by tuning
S	Associated subgraphs of a task r in R
P	the fastest program of a task r in R
M'	Pruned candidate model
C'	Connection table among tasks, subgraphs and fast programs of M'
R'	List of tasks of M', prioritized by tuning
l_m	Measured execution time on the target device
a_s	Short-term training accuracy
α	The ratio to represent the minimum allowable accuracy after pruning
β	The ratio to define the target execution time of the next pruning iteration

3 Impact of Tuning on CPrune's Pruning Performance

This experiment checks if performing the tuning for task ordering is necessary despite the relatively long time consumed in the Main step of CPrune. In the

Table 3: Mobile CPU performance test (ResNet-18 with CIFAR-10 dataset)

Model	Method	FPS (Increase rate)	FLOPS	Params	Top-1 Acc
ResNet-18 (Kryo 280)	Original (TVM)	33.82	555M	11.2M	94.37%
	CPrune	109.45 (3.24×)	161M	2.62M	93.74%
ResNet-18 (Kryo 585)	Original	40.50	555M	11.2M	94.37%
	CPrune	93.63 (2.31×)	297M	3.54M	94.14%
	CPrune (w/o tuning)	57.77 (1.43×)	390M	5.08M	94.51%
	CPrune (single subgraph pruning)	79.62 (1.97×)	294M	4.55M	94.27%

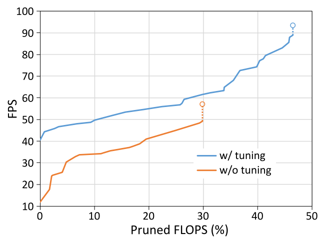

Fig. 3: Performance difference according to the tuning application

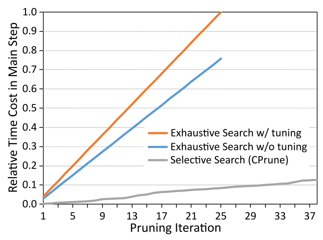

Fig. 4: Selective vs Exhaustive Search (ResNet-18, Kryo 585, ImageNet)

Main step, CPrune estimates the execution time of the current pruned candidate model M' involving tuning. To check the necessity of tuning, we compare CPrune with and without tuning with ResNet-18, Kryo 585 CPU, and CIFAR-10, as shown in Table 3.

If CPrune measures FPS immediately without tuning, the FPS is significantly lower than FPS with tuning, as shown in Figure 3. This indicates that CPrune without tuning may not select proper tasks and the number of filters to prune, and as a result, pruning does not proceed sufficiently. This is why we think the tuning process is necessary for CPrune.

4 Selective Search vs. Exhaustive Search

This experiment checks the effect of selective search used in CPrune. An on-device measurement-based approach like NetAdapt [1] exhaustively measures execution time and accuracy for each subgraph. After that, one optimal subgraph is selected and pruned before moving to the next iteration. However, CPrune introduces a mechanism for assigning dynamic priority for each task and reducing the time taken in the Main step. We compare the relative time cost in the Main step with CPrune's selective search. Figure 4 shows that CPrune reduces the time cost by nearly 90% with similar or better performance than when all subgraphs are considered, respectively.

References

1. Yang, T.J., Howard, A., Chen, B., Zhang, X., Go, A., Sandler, M., Sze, V., Adam, H.: Netadapt: Platform-aware neural network adaptation for mobile applications. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 285–300 (2018) 4