HeadStart: Enforcing Optimal Inceptions in Pruning Deep Neural Networks for Efficient Inference on GPGPUs

Ning Lin[1,2*], Hang Lu[1,2*], Xin Wei[1,2] and Xiaowei Li[1,2]
State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences[1]
University of Chinese Academy of Sciences[2]
{ linning, luhang, weixin, lxw }@ict.ac.cn

***Abstract*—Deep convolutional neural networks are well-known for the extensive parameters and computation intensity. Structured pruning is an effective solution to obtain a more compact model for the efficient inference on GPGPUs, without designing specific hardware accelerators. However, previous works resort to certain metrics in channel/filter pruning and count on labor intensive fine-tunings to recover the accuracy loss. The "inception" of the pruned model, as another form factor, has indispensable impact to the final accuracy but its importance is often ignored in these works. In this paper, we prove that optimal inception will be more likely to induce a satisfied performance and shortened fine-tuning iterations. We also propose a reinforcement learning based solution, termed as HeadStart, seeking to learn the best way of pruning aiming at the optimal inception. With the help of the specialized head-start network, it could automatically balance the tradeoff between the final accuracy and the preset speedup rather than tilting to one of them, which makes it differentiated from existing works as well. Experimental results show that HeadStart could attain up to 2.25x inference speedup with only 1.16% accuracy loss tested with large scale images on various GPGPUs, and could be well generalized to various cutting-edge DCNN models.**

I. INTRODUCTION

Modern computer vision tasks rely on deep convolutional neural network (DCNN) models to extract high-level features of the input images or video streams. These DCNN models are usually designed with deeper and wider convolutional layers with myriads of parameters [1, 2]. The over-parameterized models make it difficult to be deployed for high-throughput and energy-efficient inference, especially in lightweight devices like cellphones, drones and autonomous robotics.

Network pruning technique has recently drawn much attention due to its efficacy in compressing model parameters and reduce computation and storage cost imposed to the hardware. According to some excellent works, pruning schemes could be categorized into filter-wise [3-5], channel-wise [6-8] and connection-wise [9, 10]. The last scheme (i.e. [9]) must rely on specialized software libraries (i.e. cuSPARSE CSRMV) or hardware accelerators [11-14] to handle the inherited sparsity after pruning, so it is termed as unstructured pruning. In contrast, filter/channel-wise pruning does not violate the regularity of the feature map, and as the representatives of the structured pruning shown in Figure 1, it is more amenable to accelerating the model computation through of-the-shelf facilities like general purpose GPUs.

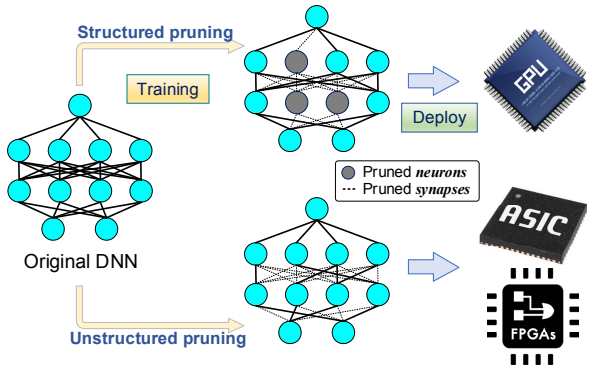


Figure 1 The affinity of structured/unstructured pruning to various hardware facilities. Structured pruning is more amenable to GPGPUs because the acceleration could be directly obtained instead of designing specific hardware.

For structured pruning, most existing schemes like [3, 4, 15] target the 'trivial' filters according to the specialized criticality metrics, and retrain the rest of the model to the target accuracy or within a certain range of the accuracy drop. Although these schemes have some potentials in reducing model computations, the performance of the pruned model, as the ultimate goal, is hard to be ensured. The reason is that the metrics employed in these schemes can only decide the worth of the filters within the current convolutional layer, but the final performance of the pruned model however, is not directly related with these metrics. To compensate for the accuracy loss after pruning, they bank on iterative fine-tunings with great uncertainty that the final accuracy is recoverable.

According to different metrics, pruning will generate different groups of survival filters with associated initial parameters inherited from the original non-pruned model. These initial parameters determine the initial output accuracy of the pruned model, and different survival filters will yield unpredictable results. From our observation, higher initial accuracy is also more prone to induce a higher final accuracy with shortened fine-tuning iterations, but this factor is easily ignored during pruning in previous works. Therefore, it is possible that a determined "trivial" filter might be less contributive in the current layer, but contains valuable knowledge that is critical to the final performance. Falsely cutting these filters is very likely to yield unrecoverable accuracy, especially at large speedup requirements, even if intense fine-tunings are imposed afterwards.

In order to achieve efficient inference for structured pruning on GPGPUs, in this paper, we propose a proactive pruning method --- *HeadStart*, to accelerate the model inference but at the same time retains the accuracy of the original DCNN models. Differed from previous works, HeadStart considers the importance of the initial accuracy by exploring different groups of survival filters, or what we

* The first two authors contributed equally.

DAC '19, June 2–6, 2019, Las Vegas, NV, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3317837

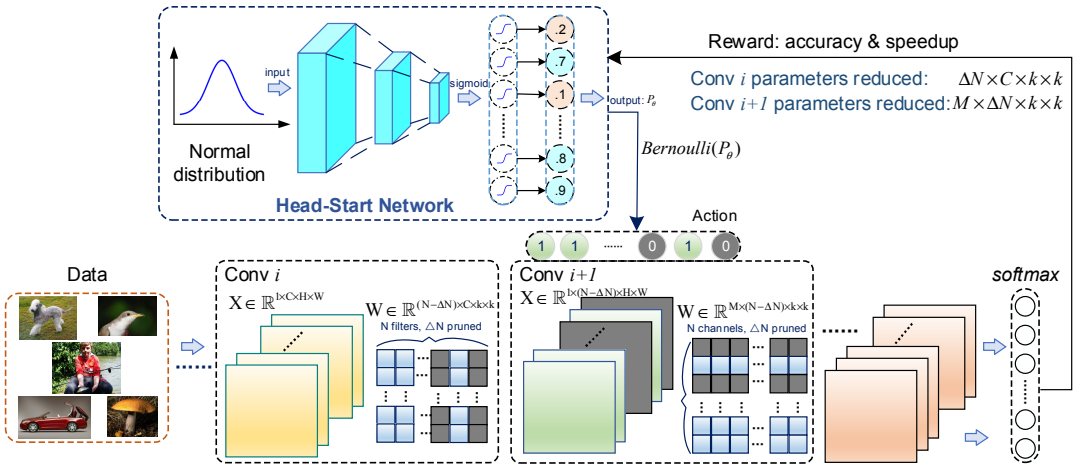


Figure 2 Framework of HeadStart pruning. We instantiate a dedicated "Head-Start network" in each layer to guide the best inception. The action for pruning is learnt from the rewards, which involves the final accuracy and preset speedup. Pruning channels in the current layer is in conjunction with pruning filters of the previous layer. For instance, the parameters of layer Conv $i+1$ are reduced by $M \times \Delta N \times k \times k$, and for layer Conv i, the reduction is $\Delta N \times C \times k \times k$.

call the "*inceptions*" of the pruned model in this paper. HeadStart does not nail particular metrics for pruning, but targets the optimal inception that yields a relatively higher initial accuracy before fine-tunings. The information reflected in the inception could be regarded as the basics in learning the new stuff, so an optimal inception can be more easily trained for an optimal final performance.

HeadStart is a reinforcement learning based solution, designed as a software approach but serves for the efficient model inference on GPGPUs, by considering the optimal inception of the pruned model. The inception is generated after applying the action from a dedicated policy network iteratively. By establishing the connection between the action and the model output, it could automatically *learn* from the rewards for the best action that will generate the optimal inception as well.

Besides, HeadSart is a generalized scheme. For the single-branch shallow networks like LeNet [16], AlexNet [17] and VGG [1] in which the network structure is less complex and the number of layers is relatively small, we can prune and fine-tune the network layer by layer; for the multi-branch deeper networks like ResNet [2], we can not only apply head-start network at a per-layer basis just like the single-branch models, but also apply the same concept to prune the residual blocks for the fast deployment on GPGPUs. Experimental results prove that HeadStart outperforms other inception-agnostic approach or training from scratch under various datasets and DCNN models.

II. Structured Pruning for Deep Neural Networks

The *key design tradeoff* of structured pruning is to obtain an acceptable model accuracy but with a significantly reduced computation intensity, without destroying the regularity of the computations in GPGPUs. Many classic schemes resort to the specialized criticality metric for pruning low ranked filters or channels in DNN models. For example, APoZ [15] uses a straight-forward ranking, based on the average percentage of zero values in the feature maps. The maps with more zeros are proposed to be pruned away. Li [3] proposes to rank the filters by the absolute summation of its values and prune the filters with the summation below the preset threshold. Entropy-based method [18] regards the channels with small entropy as less important and can be safely pruned. Liu [7] proposes channel scaling factors to determine the criticality of each layer. However, HeadStart does not rely on any of these metrics but considers the optimal inceptions.

Some recently proposed schemes formulate structured pruning as an optimization problem. As the representative, ThiNet [4] tries to approach the original intra-layer output by manipulating the non-pruned channels and uses linear regression to minimize the reconstruction error for a better weight initialization. He [6] uses LASSO regression to identify trivial channels and then employs least squares to update the filter weights and reconstruct the layer output. All these schemes are trying to optimize the intra-layer output before the activation functions. We will empirically show that the optimization based on the final output accuracy instead of individual layers is more beneficial to the model accuracy. In the next section, we will elaborate HeadStart concept, and how it utilizes reinforcement learning to tackle the tradeoff between model size and performance, and at the same time maintain the inference efficiency on GPGPUs.

III. Methodology

Deep neural network models extract features at a per-layer basis. The features generated in the higher layers are closer to the target features of a deep learning task. Following this idea, if we want an optimal inception, the solution procedure should also be layer-wise. Intuitively, we can use brutal-force enumerations to iterate the filters of each layer, and tries to verify the final performance one by one. For the lower layers with fewer feature maps and filters, it might be a viable solution. However, considering the increasing amount of filter volume in the high-level layers, the brutal-force emulation cannot sustain the vast solution space and computational cost. A cost-effective method would be letting the model *learn* the best inception, so we cast our generic model in the reinforcement learning framework to solve this problem.

A. *HeadStart Framework*

As demonstrated in Figure 2, we instantiate a "head-start Network" for each layer acting as the policy network in the reinforcement learning, whose input is a noise map following Gaussian distribution, and its output is the action denoted by the probabilities. The action is the policy used to prune filters (or channels) of this convolutional layer. The intrinsic structure of the head-start network is composed of three convolution layers and one fully connected layer. In each iteration, it observes a reward from the final output of the DNN model and update its internal parameters. Apart from classic channel/filter pruning scheme that is, for instance, computing the criticality metrics as the proof, HeadStart seeks to explore the optimal pruning manner by reinforcing the selection policy until it attains the stable state.

HeadStart targets the feature maps of each layer, as a way to prune relevant filters in local and previous neighboring layers at the same time. If a feature map is pruned in this layer, it reduces the computations in generating this feature map from the filters in the previous layer, and also eliminates the computations of the relevant filter channels within the same layer. Taking Figure 2 as an example, all the sigmoid output probabilities from head-start network are firstly binarized by passing them to the Bernoulli distribution function in Conv $i+1$. The binarized action indicates which channel is about to be pruned in this layer --- marked as 0 in the figure. These feature maps are generated by the filters in layer Conv i. Quantitatively, if the dimension of these filters is $\mathrm{N} \times \mathrm{C} \times \mathrm{k} \times \mathrm{k}$, the parameter reduction would be $\Delta\mathrm{N} \times \mathrm{C} \times \mathrm{k} \times \mathrm{k}$ with $\Delta\mathrm{N}$ the amount of pruned filters in the Conv i. Another part of the parameter reduction stems from Conv $i+1$. The filters in this layer cut the channels in correspondence with the pruned feature maps. The parameter reduction this time is $\mathrm{M} \times \Delta\mathrm{N} \times \mathrm{k} \times \mathrm{k}$. Similarly, the layer after Conv $i+1$ also prunes the filters in Conv $i+1$ in the same way, so the actual reduction is even larger in HeadStart.

B. *Problem Formulation*

The goal of HeadStart is the optimal inception, by minimizing the gap of the final accuracy before and after pruning each convolutional layer. If we use $f_{\mathcal{W}}(\mathcal{D}|\mathcal{W})$ to denote the accuracy before pruning and $f_{\mathcal{W}'}(\mathcal{D}|\mathcal{W}')$ to denote the accuracy after pruning, the problem could be formulated with Eq. (1) as follows:

$$\arg\min_{\mathcal{W}'} |f_{\mathcal{W}}(\mathcal{D}|\mathcal{W}) - f_{\mathcal{W}'}(\mathcal{D}|\mathcal{W}')| \quad (1)$$

$$s.t.\ \text{Number of remained filters}\ (\mathcal{W}') \leq C/sp$$

in which $\mathcal{D}$ denotes the dataset we use; $\mathcal{W}$ is the initial filters before pruning, and C is number of original filters $\mathcal{W}$ (w.r.t $\|\mathcal{W}\|_0 = C$). $\mathcal{W}'$is the remained filters after pruning. sp that denotes the speedup is a hyper-parameter set before pruning; for instance, sp equals to 2 means the compression ratio is 50%. The optimization is under the constraint that the zero norm of $\mathcal{W}'$ should be smaller than C/sp which is the expected amount of the remained filters.

Upon generating an action with the probability of pruning a channel, the forward propagation follows this action to obtain the softmax output denoting the final accuracy. This output is then fed back to head-start network used to compute the reward. HeadStart is designed to optimize the tradeoff between speedup and accuracy at the same time, so the reward function is also defined by reflecting the two terms. For the accuracy, we use the ratio of $f_{\mathcal{W}'}(\mathcal{D}|\mathcal{W}')$ and $f_{\mathcal{W}}(\mathcal{D}|\mathcal{W})$ and cast it in the logarithm as follows:

$$\overline{ACC} = log(\frac{f_{\mathcal{W}'}(\mathcal{D}|\mathcal{W}')}{f_{\mathcal{W}}(\mathcal{D}|\mathcal{W})} + 1) \quad (2)$$

The value of $\overline{ACC}$ in Eq. (2) is larger when $f_{\mathcal{W}'}(\mathcal{D}|\mathcal{W}')$ is closer to $f_{\mathcal{W}}(\mathcal{D}|\mathcal{W})$. For the other half of the reward, the speedup is also required to approach the target preset speedup, reflected in the following expression:

$$\overline{SPD} = \left|\frac{C}{\|A\|_0} - sp\right| \quad (3)$$

where $\|A\|_0$ is the number of non-zero elements in the action vector, that is, the number of remaining feature maps in this layer. The constant C is consistent with the one in Eq. (1). $\mathrm{C}/\|A\|_0$ is the learnt speedup via its head-start network, and because our goal is to make the learnt speedup as close to sp as possible, the $\overline{SPD}$ in Eq. (3) hence denotes the proximity of the learnt speedup to sp. In sharp contrast to $\overline{ACC}$, HeadStart optimizes the value of $\overline{SPD}$ as tiny round zero as possible, because that means the learnt speedup is closer to the preset speedup and the agent will be awarded. To this end, the overall reward function is defined as follows:

$$\mathcal{R}(\mathrm{A}) = \overline{ACC} - \overline{SPD} \quad (4)$$

C. *Training Procedure*

The goal of training is to maximize the positive expected reward $\overline{ACC}$ while minimize the negative expected reward $\overline{SPD}$. As long as $\mathcal{R}(\mathrm{A})$ is stable, the training process ceases and $f_{\mathcal{W}'}(\mathcal{D}|\mathcal{W}')$ and sp both get their optima. The *loss function* is also constructed based on the reward function:

$$L(\theta) = -\mathbb{E}[\mathcal{R}(A^s)] \quad (5)$$

$$A^s \sim Bernoulli(p_\theta) \quad (6)$$

where $A^s = (A_1^s, \ldots, A_k^s)$; A_k^s denotes the binary action sampled from the model k times. In practice, it is also applicable to use one typical sample to estimate $L(\theta)$, but here we use k Monte-Carlo samples in order to obtain a more precise estimation.

The policy gradient of $L(\theta)$ can be computed through REINFORCE algorithm [19]. The fundamental concept of REINFORCE is computing the logarithmic probability gradient of an action associated with its reward:

$$\nabla_\theta L(\theta) = -\mathbb{E}[\mathcal{R}(\mathrm{A}^s)\nabla_\theta \log p_\theta(A^s)] \quad (7)$$

In order to reduce the variance of the sampled reward, we generalize the policy gradient to include an arbitrary baseline b:

$$\nabla_\theta L(\theta) = -\mathbb{E}[(\mathcal{R}(\mathrm{A}^s) - \mathrm{b})\nabla_\theta \log p_\theta(A^s)] \quad (8)$$

where b could be a random value as long as it does not vary with each sampled action A^s. Adding baseline b will not change the expected value of $\nabla_\theta L(\theta)$, but it is very effective in reducing the variance of $\nabla_\theta L(\theta)$ which can significantly expedite the learning speed. Choosing an appropriate value of b should reference the sampled actions. Usually, if the sampled actions all demonstrate high probabilities, the baseline should also be set high to differentiate the higher valued actions from the less highly valued ones. Considering the different scenarios in pruning each layer, the baseline could be selected with the reward obtained by the current model under the inference algorithm used at test time:

$$\nabla_\theta L(\theta) = -\mathbb{E}[(\mathcal{R}(\mathrm{A}^s) - \mathcal{R}(A^I))\nabla_\theta \log p_\theta(A^s)] \quad (9)$$

$$A^I \sim \varphi_t(p_\theta), \quad \varphi_t(p_\theta) = \begin{cases} 1 & if\ p_\theta \geq t, \\ 0 & if\ p_\theta < t. \end{cases} \quad (10)$$

where $A^I = (A_1^I, \ldots, A_k^I)$; A_k^I denotes the binary inference action. According to Eq. (9), samples from the model that return higher reward than $\mathcal{R}(A^I)$ will be "pushed up", or increased in probability, while samples which result in lower reward will be suppressed. We assume that A^S obeys the Bernoulli distribution with p_θ as input. A^I obeys $\varphi_t(p_\theta)$ distribution which is defined in Eq. (10). It returns 1 when p_θ is greater than or equal to t , and 0 when p_θ is smaller than t.

After a set of training iterations until the loss function and reward function both remain stable, the lightweight pruned parameters could be verified with the test datasets. In the next sections, we will elaborate our evaluation platform including the software for training and the hardware used for inference, and show various results including the model accuracy, storage, computation intensity, as well as the inference speedup on the high-performance and edge-level GPUs, and compare them with the baselines.

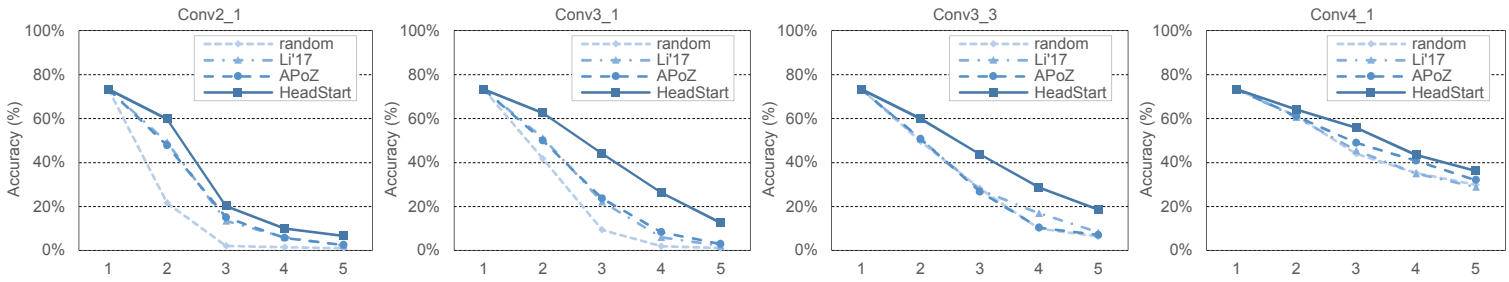


Figure 3 Single-layer pruning without fine-tuning, under different speedup configurations. Our scheme demonstrates relatively more robust performance (the higher the better).

Table 1 Whole model pruning using CUB-200 dataset. Top-1 accuracy is reported. The target compression ratio is set to 50%. #PARAMETERS is calculated based on #MAPS after pruning. #FLOPS denotes the computation intensity, measured by the floating point multiply-and-accumulate performed in GTX 1080Ti GPU. The accuracy result is shown using percentage (%). INC denotes "inception".

	W × H (#MAPS)	#MAPS AFTER PRUNING		#PARAMETERS (M)		#FLOPS (B)		ACC. (%, INC)		ACC. (%, W/ FT)	
		Li'17	**Ours**	Li'17	Ours	Li'17	Ours	Li'17	**Ours**	Li'17	**Ours**
CONV1_1	224×224 (64)	32	**32**	19.72	19.72	14.43	14.43	75.12	**76.02**	77.56	**78.82**
CONV1_2	224×224 (64)	32	**32**	19.68	19.68	13.49	13.49	41.62	**53.52**	75.34	**78.71**
CONV2_1	112×112 (128)	64	**64**	19.58	19.58	12.34	12.34	38.72	**65.60**	77.53	**78.56**
CONV2_2	112×112 (128)	64	**64**	19.39	19.39	11.41	11.41	3.21	**55.04**	76.36	**78.39**
CONV3_1	56×56 (256)	128	**127**	19.03	19.03	10.25	10.24	4.48	**55.40**	75.79	**78.44**
CONV3_2	56×56 (256)	128	**128**	18.59	18.58	8.86	8.86	18.98	**64.30**	75.21	**78.11**
CONV3_3	56×56 (256)	128	**128**	17.85	17.85	7.94	7.93	2.48	**52.86**	73.39	**77.79**
CONV4_1	28×28 (512)	256	**254**	16.37	16.36	6.78	6.76	6.81	**63.60**	73.57	**78.10**
CONV4_2	28×28 (512)	256	**258**	14.60	14.61	5.39	5.39	13.76	**63.58**	71.48	**77.37**
CONV4_3	28×28 (512)	256	**258**	12.83	12.85	4.70	4.70	16.76	**62.27**	70.65	**77.47**
CONV5_1	14×14 (512)	256	**254**	11.06	11.06	4.35	4.35	12.98	**62.90**	70.93	**76.20**
CONV5_2	14×14 (512)	256	**257**	9.29	9.30	4.00	4.00	45.37	**67.69**	71.84	**76.23**
CONV5_3	14×14 (512)	512	**512**	9.29	9.30	4.00	4.00	/	/	71.84	**76.23**

Table 2 Pruning results on VGG-16 using CUB-200. Top-1 accuracy is reported. Compression ratio (sp=2) is computed by Eq. (11); smaller compression ratio indicates a smaller parameter size.

	#PARAMETERS (M)	**#FLOPS (B)**	**ACC. (%)**	**COMP. RATIO (%)**
VGG-16 **ORI.**	19.74	15.40	77.39	100
RANDOM	9.87	4.81	70.25	50.00
THINET'17	9.87	4.81	73.00	50.00
AUTOPRUNER'18	9.87	4.81	73.45	50.00
LI'17	9.29	4.00	71.84	47.06
HEADSTART	**9.30**	**4.00**	**76.23**	**47.11**
FROM SCRATCH	9.30	4.00	28.88	47.11

Table 3 Pruning results on VGG-16 using CIFAR-100. HeadStart outperforms other baselines with a smaller compression ratio (sp=5). Top-1 accuracy is reported.

	#PARAMETERS (M)	**#FLOPS (B)**	**ACC. (%)**	**COMP. RATIO (%)**
VGG-16 **ORI.**	14.77	0.314	73.37	100
RANDOM	3.29	0.090	68.79	22.22
LI'17	3.29	0.090	70.79	22.22
APOZ	3.29	0.090	69.37	22.22
HEADSTART	**3.27**	**0.077**	**71.49**	**22.09**
FROM SCRATCH	3.27	0.077	70.04	22.09

IV. EVALUATIONS

A. *Experimental Setup*

Hardware platform. We employ 2 types of GPGPUs to verify the effectiveness of HeadStart, which is (1) GTX 1080Ti as a representative of high performance GPUs on the cloud and (2) NVIDIA Jetson TX2 for edge devices like cyber-physical systems, IoTs and robotics. The main HeadStart framework including problem modeling (Section III.B) and training (Section III.C) is implemented using PyTorch on GTX 1080Ti GPUs, and the inference of the pruned model is implemented on 2 types of GPUs. We evaluated the efficiency metrics like accuracy, storage and computation intensity, as well as the inference speed using 'frames per second (fps)'.

Datasets and baselines: we empirically study our method on two datasets: CIFAR-100 [20] and Caltech-UCSD Birds 200-2011 (CUB200-2011) [21]. The CIFAR-100 dataset consist of 60,000 32x32 pixel RGB images belonging to 100 classes, 50,000 images for training and 10,000 images for testing. The popular fine-grained CUB200-2011 dataset contains 11,788 RGB images of 200 bird species, 5,994 images for training and 5,794 images for testing. We compare our HeadStart with state-of-the-art baselines: including metric-based approaches like Li'17 [3] and APoZ [15] and intra-layer optimization approaches like Autopruner'18 [5] and Thinet'17 [4]. These schemes are all agnostic of the inceptions, and we hope HeadStart would outperform these schemes.

HeadStart specifics: hyper-parameter sp in Eq. (1) & (3) denotes the *speedup*, and is set to 2 and 5 in different sets of experiments. Note that other sp settings are totally applicable within HeadStart framework. The threshold t in Eq. (10) is set to 0.5. We use 3 Monte-Carlo samples ($k = 3$) for a more precise estimation of the binary action in Eq. (6). We use RMSprop [22] for optimizing parameter θ of the inceptions. We set the weight decay to 5×10^{-4} and the learning rate to 10^{-3}. We iteratively prune each layer until we observe a nearly constant loss and reward, which means the inception of this layer has been found. HeadStart only targets convolutional layers, so the whole model pruning terminates at the last convolutional layer and gets the final inception.

In the experiments, the compression ratio is computed by:

$$\text{Compression Ratio} = 1/Speedup = \mathcal{W}'/\mathcal{W} \qquad (11)$$

For instance, if speedup ratio is set to 5, compression ratio would be 20%. The smaller the compression ratio, the smaller the parameter size, which means the model is more compressed. The two parameters $\mathcal{W}'$ and $\mathcal{W}$ are consistent with Eq. (1).

Table 4 Pruning ResNet using CIFAR-100. HeadStart shows a higher performance compared with the original ResNet-56 and the training from scratch. "C.R." stands for compression ratio.

	#PARAM. (M)	#FLOPs(B)	ACC.(%)	C.R. (%)
RESNET-110 Original	1.730	0.254	74.70	100
RESNET-56 Original	0.892	0.131	72.98	51.56
RESNET -110. HeadStart	**0.774**	**0.131**	**74.33**	**44.74**
RESNET -110. HeadStart f. scratch	0.774	0.131	72.90	44.74

V. Results and Analysis

A. *Performance of HeadStart*

1) *VGG-16*

Single-layer pruning. In this section, we evaluate the single-layer pruning on VGG-16 model using CIFAR-100 dataset. As the first shot, we evaluate the performance by increasing the speedup without fine-tuning, as shown in Figure 3. Speedup could be regarded as the reciprocal of compression ratio, and as expected, larger speedup degrades final output accuracy. For the selected layers, we also demonstrate the results of random pruning under the same speedup. HeadStart shows much higher and robust final accuracy by learning the optimal inception that contains useful knowledge for accuracy recovery. An interesting phenomenon is that at higher speedup configuration, e.g. 4 or 5, the performance for baseline Li'17 and APoZ behave almost the same, even no larger than random pruning. This is reasonable because highly ranked filters or channels are not always useful to the final output accuracy. On the other side, it can hardly say that small filters are less important because they may also play a part in minimizing the loss function. When the speedup is set smaller, the accuracy degradation is marginal for the evaluated baselines. However, when the speedup is set higher, it will damage the accuracy greatly. In the figure, the largest gap is observed in layer Conv3_1 at speedup set to 4. HeadStart outperforms Li'17 by 20.28% and APoZ by 18%.

Another observation is that lower layers are more sensitive to the speedup scaling while the higher layers, e.g. Conv4_1 and Conv5_1 (not listed), are the opposite. Lower layers often contain more important abstract features and higher layers often contain more redundancy, which is in consistent with [3]. We can apply more aggressive pruning on the higher layers, but in either case, HeadStart always selects the inception based on the final accuracy rather than the values of the channel itself.

Pruning the whole VGG model. Pruning the whole model is through iteratively pruning each layer. After pruning each layer, we fine-tune the model 40 epochs using SGD with the hyper-parameters specified at the beginning of this section. We did not vary the learning rate during fine-tuning. HeadStart seeks to find the optimal inception before proceeding to the next layer. To illustrate the concrete procedure, we summarize all the intermediate results in Table 1. #MAPs means the amount of the feature maps in each convolution layer. #PARAMETERs and #FLOPs both use unified units: M indicating *million* and B indicating *billion*. We list the accuracy of the inception, marked as ACC. (%, INC) and after fine-tuning, marked as ACC. (%, W/ FT). In terms of the #MAPs, Li'17 ranks the filters and issues pruning based on the preset compression ratio (50% in this case), so each layer could be pruned by just one half of the parameters. By contrast, HeadStart does not directly use this preset value, but *learn*s to approach it. Therefore, the actual #MAPs may be slightly more or less than 50%. Taking layer Conv4_3 as an example, the amount of the remaining feature maps is 258, while for layer Conv4_1, the number is 254. The value is around 50% of the total maps, which is 256. This observation proves the ability of HeadStart in considering not only the amount of filters to prune as required, but also the balance with the best inception.

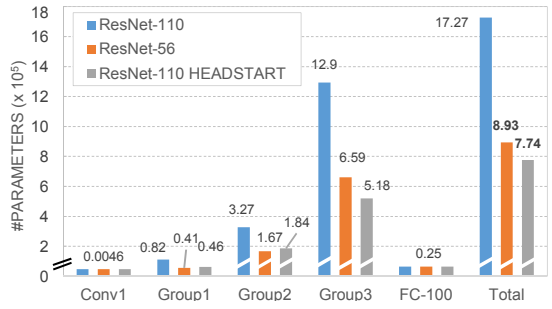


Figure 4 Comparison of #PARAMETERS (Y-axis is unified by 10^5). Benefit from learning the best inception for each group (<10, 10, 7> blocks each, in this case), HeadStart outperforms the original ResNet-110.

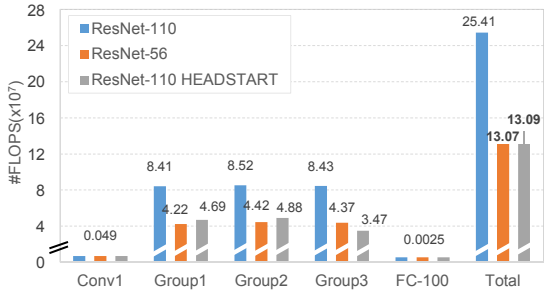


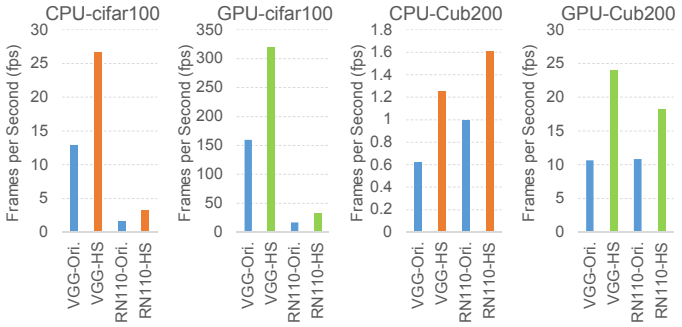
Figure 5 Comparison of #FLOPS (Y-axis is unified by 10^7).

Table 2 provides the additional results of pruning the whole VGG-16 model. For CUB-200 dataset, they only achieve 73.45% and 73% Top-1 accuracy under compression ratio 50%, while HeadStart learns a more promising 47.11% compression ratio and achieves 76.23% final accuracy. For CIFAR-100 dataset shown in Table 3, we observe the similar behavior. HeadStart achieves 71.49% accuracy as well as 22.09% learnt compression ratio. It is worth noting that baseline "from scratch" is employed to compare the performance with HeadStart if training the pruned model from scratch. The results confirm that reserving useful knowledge in the inception is a must.

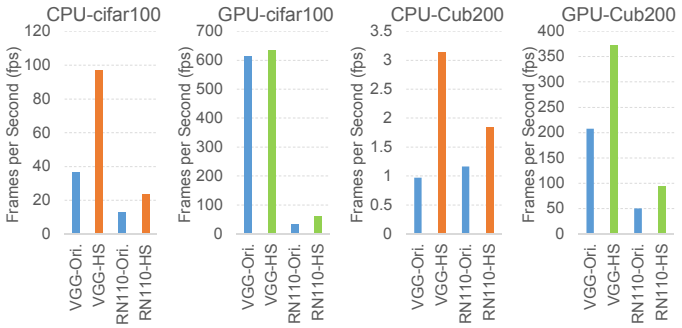
2) *ResNet*

We also explore the performance of HeadStart in pruning ResNet. Many researchers intend to take the "residual block" as the basic unit to construct diverse variants of the family. In recent studies, they also found that intensively disregarding some of these blocks during training could significantly reduce the training time [23], while keeping the performance of the *model* intact. The reason stems from the compact structure of the model and the homogeneity of the residual blocks. If one block is discarded, the loss could bypass these blocks via shortcuts in back propagation and the feature maps data could also bypass the inactive block during inference [24, 25]. Following the same paradigm, we apply HeadStart to the residual blocks instead of the individual layers in the block to make the deployment of the pruned model faster. But note that in practical use, the HeadStart concept could be directly applied to prune the convolutional layers in each block just like VGG.

We use ResNet-110 under CIFAR-100 dataset as the representative of the family. It has 3 groups in which there are 18 residual blocks each, and in each block it contains 2 convolution layers. The head-start network learns to prune the blocks in align with the preset compression ratio. In spite of the coarse-grained block level pruning, HeadStart also achieves prominent performance, as shown in Table 4. The accuracy after pruning (74.33%) is even close to the original non-pruned ResNet-110 model (74.70%), but the #FLOPs is reduced by nearly a half. Compared with the original ResNet-56, HeadStart also performs better. Although the original ResNet-56 has the same



(a) NVIDIA Jetson TX2



(b) CPU: Intel Xeon E2620, GPU: NVIDIA GTX 1080Ti

Figure 6 Statistics of HeadStart on different hardware platforms. In (a), the CPU result is read from the ARM Cortex A57 core inside NVIDIA TX2 SoC; the GPU result stems from its PASCAL with 256 cuda cores. We tested both small and large scale datasets.

amount of residual blocks as HeadStart, it is however not the optimal inception. By comparing the #PARAMETERS as shown in Figure 4, we find that for Group1 and Group2, the number of parameters of HeadStart is slightly larger than ResNet-56, while the parameters are much less for Group3. That is because the number of residual blocks in each group is <10, 10, 7>, learnt by HeadStart, while the original ResNet-56 is just constructed as <9, 9, 9>. Similarly in Figure 5, the computations increase for Group1 and Group2 due to one extra block, but decrease for Group3. Although the total computations are comparable, the symmetry of each group does not help bringing optimal inception. The total parameters are much less for HeadStart, but brought with a higher performance.

On the other hand, for the same pruned structure, it again confirms that optimal inception behaves much better than trained from scratch (only 72.90% accuracy as shown in Table 4).

B. Inference Speedup on GPUs

After presenting the accuracy and storage results, we evaluate the inference speed of DNN models pruned by HeadStart on two hardware platforms in this set of experiment. Jetson TX2 is a representative of lightweight SoC that is widely used in edge devices like robotics and autonomous machines. The results are shown in Figure 6(a). We use both small scale dataset -- Cifar-100 and large scale dataset -- CUB-200 to verify the frames per second (fps). Compared with the original VGG model, HeadStart yields 2.00x fps improvement for Cifar-100 and 2.25x improvement for CUB-200 on GPUs. Similarly, for the ResNet model, the fps enhancement is 1.96x and 1.68x for the two datasets respectively. Similarly, in Figure 6(b) we use more powerful 1080Ti GPU to verify the speed improvement on desktop computers. HeadStart exhibits 1.03x and 1.79x increased fps for VGG, as well as 1.89x and 1.88x increased fps for ResNet. We also observe more than 1.5x fps improvement on the CPUs of the two platforms. Note that for TX2 platform, HeadStart shows as high as 24fps inference speed using VGG net for the high resolution images in CUB-200 dataset, which means the model could already process high quality images in a speed that is not recognizable for human vision system, so it attains the real-time machine learning that is required in edge devices.

VI. Conclusion

In this paper, we propose a novel structured pruning method for the efficient inference of DCNN models on GPGPUs --- HeadStart, which leverages reinforcement learning to explore the optimal inception of the pruned model. By establishing the relationship between actions and the output accuracy, the dedicated head-start network is able to learn the best pruning strategy from the rewards. HeadStart is a generalized approach that could be applied to many cutting edges DNN models like VGG and ResNet, with promising performance compared with contemporary schemes. We also emphasized the importance of the optimal inception to the final performance. We hope the techniques proposed in this paper will motivate a reconsideration of the DNN model pruning by applying the same concept over other computer vision tasks, such as object detection or semantic segmentation that runs on general purpose GPUs both on the cloud or at the edge.

VII. Acknowledgement

This work is supported by the National Natural Science Foundation of China (NSFC) under grant No. (61532017, 61432017, 61602442, 61876173). Corresponding authors are Hang Lu and Xiaowei Li.

References

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556,* 2015.
[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770-778.
[3] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *ICLR*, 2017.
[4] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *ICLR*, 2017.
[5] J. Luo and J. Wu, "AutoPruner: An End-to-End Trainable Filter Pruning Method for Efficient Deep Model Inference," *arXiv preprint arXiv:1805.08941,* 2018.
[6] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *ICCV*, 2017, vol. 2, no. 6.
[7] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *ICCV*, 2017, pp. 2755-2763.
[8] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *arXiv preprint arXiv:1611.06440,* 2016.
[9] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *NIPS*, 2015, pp. 1135-1143.
[10] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *ICLR*, 2016, pp. 1-14.
[11] S. Han *et al.*, "EIE: efficient inference engine on compressed deep neural network," in *ISCA*, 2016: IEEE.
[12] H. Lu, X. Wei, N. Lin, G. Yan, and X. Li, "Tetris: re-architecting convolutional neural network computation for machine learning accelerators," in *ICCAD*, 2018: IEEE.
[13] L. Ke, X. He, and X. Zhang, "NNest: Early-Stage Design Space Exploration Tool for Neural Network Inference Accelerators," in *ISLPED*, 2018.
[14] X. He, L. Ke, W. Lu, G. Yan, and X. Zhang, "AxTrain: Hardware-Oriented Neural Network Training for Approximate Inference," in *ISLPED*, 2018.
[15] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv:1607.03250,* 2016.
[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86, no. 11, pp. 2278-2324, 1998.
[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1097-1105.
[18] J.-H. Luo and J. Wu, "An entropy-based pruning method for CNN compression," *arXiv preprint arXiv:1706.05791,* 2017.
[19] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," 2011.
[20] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer2009.
[21] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The caltech-ucsd birds-200-2011 dataset," 2011.
[22] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," 2012.
[23] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *ECCV*, 2016, pp. 646-661.
[24] Z. Wu *et al.*, "Blockdrop: Dynamic inference paths in residual networks," in *CVPR*, 2018, pp. 8817-8826.
[25] X. Wang, F. Yu, Z.-Y. Dou, and J. E. Gonzalez, "Skipnet: Learning dynamic routing in convolutional networks," in *ECCV*, 2018.