ADC: Automated Deep Compression and Acceleration with Reinforcement Learning

Yihui He
Xi'an Jiaotong University
heyihui@stu.xjtu.edu.cn

Song Han
Google Brain / MIT
songhan@google.com

Abstract

*Model compression is an effective technique facilitating the deployment of neural network models on mobile devices that have limited computation resources and a tight power budget. However, conventional model compression techniques [19, 20, 23] use hand-crafted features and require domain experts to explore the large design space trading off model size, speed, and accuracy, which is usually sub-optimal and time-consuming. In this paper, we propose Automated Deep Compression (ADC) that leverages reinforcement learning in order to efficiently sample the design space and greatly improve the model compression quality. We achieved state-of-the-art model compression results in a fully automated way without any human efforts. Under $4\times$ FLOPs reduction, we achieved **2.7%** better accuracy than hand-crafted model compression method for VGG-16 on ImageNet. We applied this automated, push-the-button compression pipeline to MobileNet and achieved a **2**$\times$ reduction in FLOPs, and a speedup of **1.49**$\times$ on Titan Xp and **1.65**$\times$ on an Android phone (Samsung Galaxy S7), with negligible loss of accuracy.*

1. Introduction

For many real-world applications, robotics, self-driving car and advertisement ranking, convolutional neural networks are limited by strict latency and size budget. Many works focus on compressing existing networks [28, 19, 23]. However, they all need a *hand-crafted* compression ratio for each layer, which requires human expertise and has a limited search space. Here, we aim to find a compression policy for a network automatically.

We propose to use reinforcement learning for Automated Deep Compression (ADC) that does not require human expertise. While compressing a network, human expertise has to make the design choice on the sparsity for each conv layer, given the required model complexity. We notice that the accuracy of the compressed model is very sensitive to sparsity, which requires fine-grained action space. We come up with continuous compression ratio control with DDPG [30], instead of searching on a discrete space. To mitigate the long training time brought about by reinforcement learning, we propose to reward the agent with approximated model performance without fine-tuning. Our algorithm is very sample efficient: it took one GPU only one hour on CIFAR-10, and eight GPUs only four hours on ImageNet, to search for the best model compression policy.

Specifically, our DDPG agent processes a network in a layer by layer manner. For each conv layer t, our agent receives the layer embedding s_t and outputs a precise compression ratio a_t. After this layer is compressed with a_t, the agent moves to the next layer $t+1$. With all the layers compressed, the agent will receive a reward related to accuracy and overall compression rate. Finally, the best-explored model is fine-tuned for a short time (1/10 of the original training epochs) to achieve the best performance.

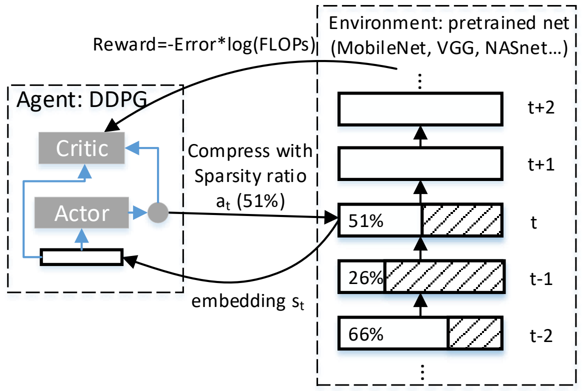
Environment: pretrained net (MobileNet, VGG, NASnet…)
Reward=-Error*log(FLOPs)
Agent: DDPG
Critic
Actor
Compress with Sparsity ratio a_t (51%)
embedding s_t
t+2
t+1
51%
t
26%
t-1
66%
t-2

Figure 1. Overview of our ADC. We process a pretrained net (*e.g.*, MobileNet) in a layer by layer manner. Our agent DDPG receives embedding s_t from a layer t, and outputs a sparsity ratio a_t. After the layer is compressed with a_t, it moves to the next layer $t+1$. With all layers the compressed, the approximate model is evaluated on validation images without fine-tuning. Finally, reward R is returned to our agent.

We further consider two application scenarios: **FLOPs-constrained compression**, to achieve the best accuracy given a maximum number of FLOPs, and **accuracy-guaranteed compression**, to achieve the smallest model given no loss of accuracy. The former corresponds to applications such as mobile AI, self-driving cars, and advertisement rankings, which have a tight **latency** requirement. The latter corresponds to cloud applications for which **quality** rather than latency is more critical; thus, we suffer no loss of accuracy after compression. In order to achieve FLOPs-constrained compression, we limit the action space (pruning ratio) so that the models compressed by the agent are always below the computation budget while trying to find the best accuracy (Sec. 3.4). For accuracy-guaranteed compression (Sec. 3.5), we come up with a novel reward R_{FLOPs}, which combines accuracy and computation. We can accurately find the limit of compression that does not lose accuracy.

Our ADC algorithm is evaluated on ResNet [22], VGG [51], and MobileNet [24] on both CIFAR-10 [32] and ImageNet [13]. We also tested the generalization ability of the compressed model on object detection with Pascal VOC. A series of experiments show that it offers better performance than hand-crafted heuristic policies. Under $4\times$ FLOP reduction, we improve VGG-16 channel decomposition [61] top-5 accuracy by **2.7%** and channel pruning [23] top-5 accuracy by **0.3%**. We further reduced the FLOPs of 1.0 MobileNet [24] by $2\times$ with 68.8% top-1 accuracy, which is on a better Pareto curve than the 0.75 MobileNet. We achieved a speedup of **1.49**$\times$ on Titan Xp and **1.65**$\times$ on the Android phone.

2. Related Work

CNN Compression and Acceleration. Since the development of optimal brain damage [35, 21], extensive work has been done on CNN compression [20, 19, 40, 15]. Quantization [63, 12, 48] and special conv implementation [42, 55, 33, 4] speed up CNN inherently. Tensor factorization [34, 18, 31, 41] decomposed weights into several pieces. [59, 14, 17] accelerated fully connected layers with truncated SVD. [28] factorized a layer into 1×3 and 3×1. [61] factorized a layer into 3×3 and 1×1. Channel pruning [47, 25, 2, 45] removed redundant channels on feature maps. A common problem, however, is how to determine sparsity ratio for each layer.

Neural Architecture Search. Many works on searching models with reinforcement learning or genetic algorithms greatly improve CNN [53, 49, 6, 43]. [65] proposed to search transferable network blocks. NASNets have surpassed *hand-designed* architectures [54, 22, 11] on ImageNet. [7] speeded up exploration via network transformation [9]. Integrating reinforcement learning into model compression becomes natural. Concurrent with our work, N2N [3] also aims to search a compressed model with reinforcement learning. However, their state space is not scalable to ImageNet. As shown in the Table. 1, we demonstrate several merits of our ADC over the others. Our ADC benefits from having no need for an RNN to produce the policy, no need for training to provide the reward, continuous compression ratio control, automatic accuracy-guaranteed compression, and FLOPs-constrained compression options.

	NAS	NT	N2N	**ADC**
policy given by RNN	✓	✓	✓	
fast exploration		✓	✓	✓
continuous action space				✓
budget limited search				✓

Table 1. Comparisons of reinforcement learning approaches for searching models (NAS: Neural Architecture Search [65], NT: Network Transformation [7], N2N: Network to Network [3], and ADC: our Automated Deep Compression). Our ADC benefits from reward without finetuning, continuous compression ratio control, accuracy-guaranteed, and FLOPs-constrained compression.

3. Methodology

Fig. 1 illustrates our Automated Deep Compression agent. We aim to automatically find the sparsity ratio for each layer of a network in order to improve the quality of compression algorithms. The sparsity ratio can drastically affect the compressed model's performance ([20, 23], Sec. 4.2.1). Motivated by this, we propose to search for sparsity ratios in fine-granularity with reinforcement learning that does not require human expertise. Specifically, we train a DDPG agent that receives a layer embedding s_t and outputs a fine grained sparsity ratio a_t for each layer. We efficiently search the state space with the validation accuracy without fine-tuning as the reward, which is a very effective delegate and extremely time-efficient.

3.1. Problem Definition

Model compression is achieved by reducing the number of channels from c to c' for each convolutional layer, where $c' < c$. We studied three compression algorithms. Given a $n \times c \times k_h \times k_w$ convolutional weight, spatial decomposition [28] factorizes it into $n \times c' \times k_h \times 1, c' \times c \times 1 \times k_w$. Channel decomposition [61] factorizes it into $n \times c' \times k_h \times k_w, c' \times c \times 1 \times 1$. Channel pruning [23] shrinks it into $n \times c' \times k_h \times k_w$.

Given the underlying network and compression algorithm, our goal is to precisely find the effective number of channels c' for each layer, which has usually been manually determined in previous studies [28, 36, 23].

3.2. The State Space

For each layer t, we have 11 features that characterize the state s_t:

$$(t, n, c, h, w, stride, k, FLOPs[t], reduced, rest, a_{t-1}) \tag{1}$$

where t is the layer id, the dimension of the kernel is $n \times c \times k \times k$, and the input is $c \times h \times w$. $FLOPs[t]$ is the FLOPs of layer t. $Reduced$ is the total number of reduced FLOPs in previous layers. $Rest$ is the number of remaining FLOPs in the following layers. Before being passed to the agent, they are scaled within $[0, 1]$. Such features are essential for the agent to distinguish one convolutional layer from another (Sec. 4.1.2).

3.3. The Action Space

Most of the existing works use discrete space as coarse-grained action space (*e.g.*, $\{64, 128, 256, 512\}$ for number of channels). Coarse-grained action space might not be a problem for a high-accuracy model architecture search. However, we observed that model compression is very sensitive to sparsity ratio and requires fine-grained action space, leading to an explosion of the number of discrete actions (Sec. 4.2.1). Such large action spaces are difficult to explore efficiently [37]. Discretization also throws away the order: for example, 10% sparsity is more aggressive than 20% and more aggressive than 30%.

We propose to use continuous action space $a = \frac{c'}{c}, a \in (0, 1]$, which enables more fine-grained and accurate compression.

3.4. FLOPs-Constrained Compression

By limiting the action space (the sparsity ratio for each layer), we can accurately arrive at the target compression ratio. Following [65, 5, 62], we use the following reward:

$$R_{err} = -Error \tag{2}$$

Obviously, this reward offers no incentive for FLOPs reduction, so we need to limit the action space: we allow arbitrary action a at the first few layers; we start to limit the action a when we find that the budget is insufficient even after compressing **all** the following layers. Algorithm 1 illustrates this process. (For channel pruning, the code will be longer but similar, since removing a filter affects the computation of two layers.) Note that our algorithm is not limited to constraining *FLOPs*. *FLOPs* can be replaced by other resources, such as the number of parameters, or even the actual inference time on the mobile device. Based on our experiments (Sec. 4.1.4), since the agent receives no incentive for going below the budget, it can accurately arrive at the target compression ratio.

Algorithm 1 search with constrained FLOPs

```
1:  procedure GETACTION($t$)
2:      $a = \mu'(s_t)$
3:      $desired \leftarrow$ desired FLOPs reduction
4:      if $t == 0$ then
5:          $reduced \leftarrow 0$
6:      $rest \leftarrow sum(FLOPs[t+1, ...])$
7:      $duty \leftarrow desired - reduced - rest$
8:      if $duty > 0$ then
9:          $a \leftarrow min(a, 1 - duty/FLOPs[t])$
10:     $reduced \leftarrow reduced + (1 - a) * FLOPs[t]$
11:     return $a$
```

3.5. Accuracy-Guaranteed Compression

By tweaking the reward function, we can accurately find the limit of compression that offers no loss of accuracy. We empirically observe that $Error$ is inversely-proportional to $log(FLOPs)$ [8]. Driven by this, we devise the following reward function:

$$R_{FLOPs} = -Error \cdot log(FLOPs) \tag{3}$$

This reward function is sensitive to $Error$; in the meantime, it offers a small incentive for reducing $FLOPs$. Based on our experiments in Fig. 4.1.5, we note that our agent automatically finds the limit of compression.

3.6. The Agent

Illustrated in Fig. 1, the agent receives layer t embedding state s_t from the environment and then outputs a sparsity ratio, action a_t. The underlying layer is compressed with a_i using a specified compression algorithm (*e.g.*, spatial decomposition [28]). Then the agent moves to the next layer $t + 1$, and receives the state s_{t+1}. After finishing the final layer T, the reward accuracy is evaluated on the validation set and gets returned to the agent. For fast exploration, we evaluate the reward accuracy without fine-tuning, and it works well (Sec. 4.1.3).

We utilize the Deep Deterministic Policy Gradient (DDPG) [37] for continuous control. Truncated normal distribution (TN) [30] is added to our actor policy μ to construct exploration policy μ':

$$\mu'(s_t) \sim TN(\mu(s_t|\theta_t^{\mu}), \sigma^2, 0, 1) \tag{4}$$

$\mu'(s_t) \in (0, 1)$. σ determines the randomness of policy. We exponentially decay σ during exploitation, illustrated in Fig. 2.

Following Block-QNN [62], which uses a variant form of Bellmans Equation [57], each transition in an episode is (s_t, a_t, R, s_{t+1}), where R is the reward after the network is compressed. While updating, baseline function b is incorporated to reduce the variance of gradient estimation, which is an exponential moving average of the previous rewards [64, 7]:

$$Loss = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$
$$y_i = r_i - b + \gamma Q(s_{i+1}, \mu(s_{i+1})|\theta^Q) \quad (5)$$

The discount factor γ is set to 1 to not over-prioritize short-term rewards [5].

4. Experimental Results

For spatial decomposition [28], we use *data independent* reconstruction (the compression only operates weights and features are not involved [52]). For channel pruning [23], we use *max response* selection (pruning the weights according to the magnitude [20]), and preserve Batch Normalization [27] layers during pruning instead of merging them into convolutional layers. Our agent first explores 100 episodes with a constant noise σ 0.5, and then exploits 300 episodes with exponentially decayed noise σ (Fig. 2). The learning rate $1e^{-4}$ and $1/10$ original number of iterations are used for fine-tuning [23]. Our implementation is based on open source projects: Caffe [29] [1], and TensorFlow [1] [2].

4.1. CIFAR-10 and Analysis

We conduct extensive experiments and fully analyze our ADC on CIFAR-10 [32], which consists of 50k training and 10k testing 32×32 tiny images in ten classes. We split the training images into 45k/5k train/validation. The accuracy reward is obtained on validation images. Our approach is extremely efficient. (*e.g.*, *data independent* and *data dependent* spatial decomposition [28] can finish searching within **5 minutes/1 hour** respectively[3].)

4.1.1 Comparison with Random Search

Theoretically, after a large number of trials, the random search result will be close to the reinforcement learning result. It is unclear how many trials are necessary for the random search to succeed, especially for small networks on CIFAR-10. As shown in Fig. 2, we search sparsity policy for spatial decomposition of Plain-20 under $2\times$. Before 100 episodes, our ADC behaves the same as random search. Both explore with constant noise $\sigma = 0.5$. After 100 episodes, our ADC exploits with exponentially decayed noise σ and start learning. It can find a good model quickly; however, random search plateaus. Exploring 400 episodes is insufficient for the random search to get good enough results, even for a small network on CIFAR-10.

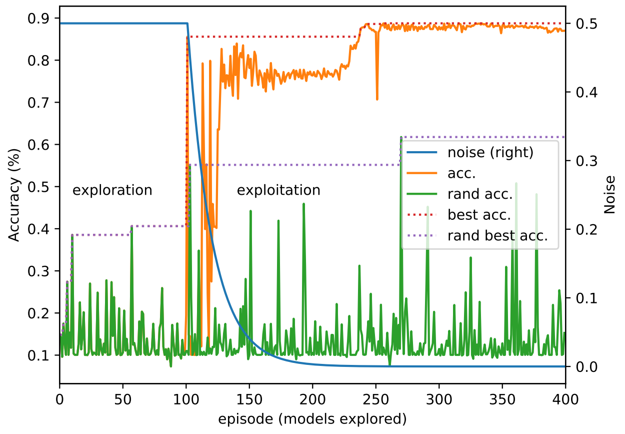

Figure 2. Validation accuracy and noise decay versus episode, using spatial decomposition on CIFAR-10 under $2\times$. We explore for 100 episodes with constant noise σ. Then we exploit for 300 episodes with exponentially decayed noise σ. Notice that best accuracy from random exploration plateaus for very long time. Random search cannot find good enough results with limited episodes. *(better viewed in color)*

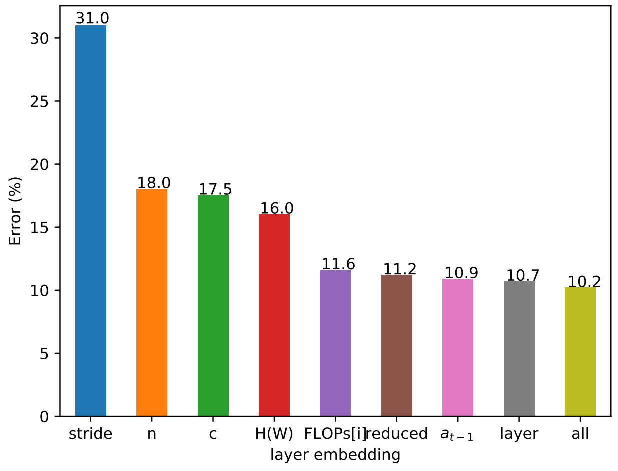

Figure 3. Effectiveness of each layer embedding, evaluated with spatial decomposition for Plain-20 $2\times$. Combining all embeddings together, the performance is even better. (*smaller is better*)

4.1.2 Effectiveness of Layer Embeddings

Layer embedding state s_t (Sec. 3.2) is essential for ADC to distinguish different convolutional layers. As shown in Fig. 3, we evaluate the effectiveness of each layer embedding, and the combination of all of them with spatial decomposition Plain-20 under $2\times$. Straightforwardly, with only $stride$, it is too difficult to distinguish different layers. The performances of n, c, H, W are similar, since their values change together when $stride$ is 2. $FLOPs[i], reduced, a_{t-1}$ and layer index are most effec-

[1]https://github.com/yihui-he/channel-pruning
[2]https://github.com/pemami4911/deep-rl
[3]On single GeForce GTX TITAN Xp GPU and Intel Xeon CPU E5-2680

Model	policies	FLOPs (%)	val acc. (%)	test acc. (%)	test acc after ft. (%)
Plain-20 (90.5%)	deep (handcraft)	50%	79.6	79.2	88.3
	shallow (handcraft)		83.2	82.9	89.2
	uniform (handcraft)		84.0	83.9	89.7
	ADC (ours)		**86.4**	**86.0**	**90.2**
	ADC (R_{FLOPs})	80%	90.7	90.5	-
ResNet-56 (92.8%)	uniform (handcraft)	50%	87.5	87.4	89.8
	deep (handcraft)		88.4	88.4	91.5
	ADC (ours)		**90.2**	**90.1**	**91.9**

Table 2. Pruning policies comparison of Plain-20, ResNet-56 [22] on CIFAR-10 [32]. For both shallow network Plain-20 and deeper network ResNet-56, our ADC outperforms *hand-crafted* policies by a large margin. Validation, test, and fine-tuned accuracy are consistent. This enables our efficient exploration without fine-tuning. Though our ADC makes many trials on model architecture, no overfitting is observed. (*larger is better*)

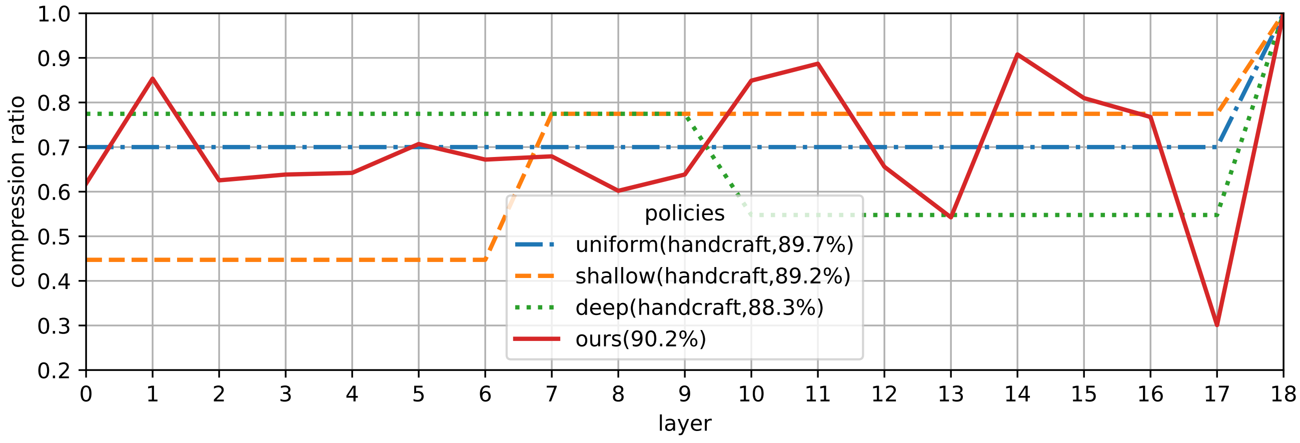

Figure 4. Comparisons of pruning strategies for Plain-20 under 2×. *Uniform* policy sets the same compression ratio for each layer uniformly. *Shallow* and *deep* policies aggressively prune shallow and deep layers respectively. Our ADC policy looks like sawtooth, which resembles the highly compact bottleneck architecture [22]. It outperforms *hand-crafted* policies by a large margin. (*better viewed in color*)

tive, since they dynamically change at each layer. a_{t-1} and $reduced$ resemble the RNN used by other architecture search approaches [65, 7], since s_t is affected by s_{t-1}. In the following experiments, we combine all these embeddings, which makes the performance even better.

4.1.3 Validation, Test, Fine-tuned Accuracy

Test versus Fine-tuned Accuracy. We observed a correlation between the final accuracy after fine-tuning and the validation accuracy before fine-tuning [20, 23]. As shown in Table 2, policies that obtained higher validation accuracy correspondingly have higher fine-tuned accuracy. This enables us to predict final model accuracy without training, which results in efficient exploration.
Validation versus Test Accuracy. We have separate validation and test sets, and only used the validation set to give the reward function during reinforcement learning. In addition, the compressed models we searched have fewer parameters. As shown in Table 2, the test accuracy and the validation accuracy are very close; there's no indication of overfitting.

4.1.4 FLOPs-Constrained Compression

We compared our approach to the three empirical policies [36, 23] illustrated in Fig. 4: *uniform* sets compression ratio uniformly, *shallow* and *deep* aggressively prune shallow and deep layers respectively. Based on sparsity distribution of different networks, a different strategy might be chosen (*e.g.*, VGG-16 favors *shallow* [23]).

In Table. 2, we show our use of reward R_{err} to accurately find the sparsity ratios for pruning 50% for Plain-20 and ResNet-56 [22] and compare it with empirical policies. We outperform empirical policies by a large margin. Counter-intuitively, the best pruning setting found by our approach differs from empirical ones a lot (Fig. 4). It resembles bottleneck architecture [22], which is highly compact.

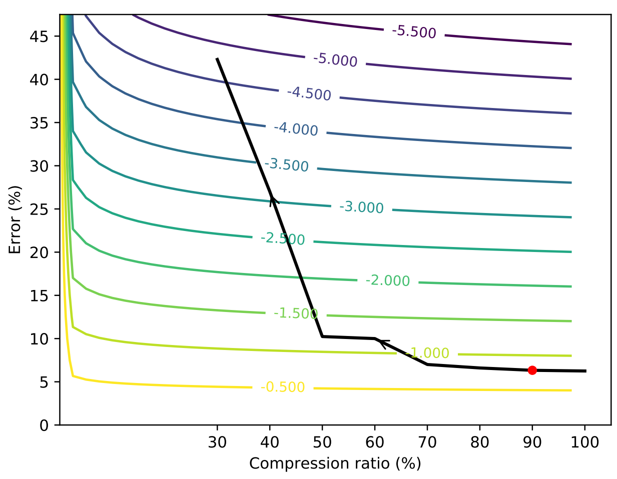

Figure 5. The $R_{FLOPs} = -Error \cdot log(FLOPs)$ manifold, where $FLOPs$ is the desired pruned model computation. R_{FLOPs} focuses on $Error$, and also offers a small incentive for reducing computation in the meantime. The black curve shows our spatial decomposition searched results on CIFAR-10 [32] by limiting action space (Sec. 3.4). As is shown, the accuracy drops sharply under 80% compression ratio. This reward naturally encourages models that have a compression ratio over 80%. The final model we found have a 90% compression ratio with no degradation. For channel pruning, the final model we found have an 80% compression ratio with no degradation. (*better viewed in color*)

4.1.5 Accuracy-Guaranteed Compression

By using our R_{FLOPs} reward, our agent can automatically find the limit of compression, with little loss of performance. As shown in Fig. 5, we decompose Plain-20 with spatial decomposition. The result we obtained has a 90% compression ratio with little degradation (marked with a red dot). The reward curve in black is obtained using the R_{err} reward. Notice that the reward drops sharply if compression goes below 80%. Since our reward R_{FLOPs} focuses on $Error$ and offers very little incentive to compression in the meantime, it prefers the high-performance model with harmless compression. For much more redundant networks, such as VGG-16 on ImageNet, our R_{FLOPs} reward automatically preserves only 64% FLOPs without any degradation (Sec. 4.2.1). To shorten the search time, we obtain the reward using the validation accuracy without fine-tuning. We believe that with fine-tuned accuracy as the reward, the agent will compress more aggressively, since the fine-tuned accuracy is much closer to the original accuracy.

4.2. ImageNet

On ImageNet [13], images are resized such that the shorter side is 256. We use 3000 images from the training set for reward evaluation. Only random crop and horizontal flip are used during fine-tuning, limited by Caffe. The testing is on the center 224 × 224 crop. Our approach is still extremely efficient (searching with *data dependent* pruning VGG-16 only takes **4 hours** [4]).

	policy	FLOP	Δacc
channel pruning	FP (handcraft) [36]	20%	-14.6
	RNP (handcraft) [38]		-3.58
	SPP (handcraft) [56]		-2.3
	empirical (handcraft) [23]		-1.7
	ADC (ours)		**-1.4**
spatial decomp.	heuristic (handcraft) [28]	25%	-11.7
	ADC (ours)		**-9.2**
	ADC (R_{FLOPs})	64%	**-0.0**
channel decomp.	uniform (handcraft) [61]	25%	-6.4
	heuristic (handcraft) [61]		-3.8
	ADC (ours)		**-1.1**

Table 3. Comparison with handcrafted approaches using channel pruning, spatial decomposition, and channel decomposition. The heuristic approach ranks filters with PCA energy and computation. Our approach could still improve performance. Spatial and channel decomposition are shown here without fine-tuning. For reference, the actual VGG-16 inference time speedup of channel pruning, spatial decomposition, and channel decomposition under 4× FLOP reduction are 2.50×, 1.01× and 1.55× respectively [23]. The baseline VGG-16 has 89.9% top-5 and 70.5% top-1 accuracy.

4.2.1 Comparison with Heuristic Approach

In channel pruning [23] VGG-16 [51][5], sparsity ratios are carefully tuned (`conv5` are skipped, preserving ratio for `conv4` and the remaining layers is 1.5 : 1). Consistent with our CIFAR-10 experiments (Sec. 4.1.4), fine-tuned top-5 error of *empirical* is **0.3%** greater than our approach, shown in Table 3.

Beyond empirical strategies, a heuristic algorithm [61] was also proposed to improve the sparsity determination in each layer. It gives a score of $\frac{\sigma_i}{C_i}$ to each filter i in a network, where σ_i is PCA energy and C_i is the computation it contributes. Then the sparsity ratio is set according to the ranking. The policies are illustrated in Fig. 6. As shown in Table. 3, for both spatial and channel decomposition, we outperform heuristic algorithm by a large margin.

Since the heuristic approach uses hand-crafted ranking, the improvement is still limited. For instance, the heuristic approach retains full rank for `conv1_1`, because it only contributes 0.6% to overall computation. However, ADC is able to figure out that decomposing `conv1_1` is not harmful, although its computation contribution is small. For `conv5`, the empirical analysis shows that the top 50% PCA energy is 80%, which is low compared to more than 90%

[4]On 8 GeForce GTX TITAN Xp GPUs and Intel Xeon CPU E5-2680

[5]https://github.com/BVLC/caffe/wiki/Model-Zoo#models-used-by-the-vgg-team-in-ilsvrc-2014

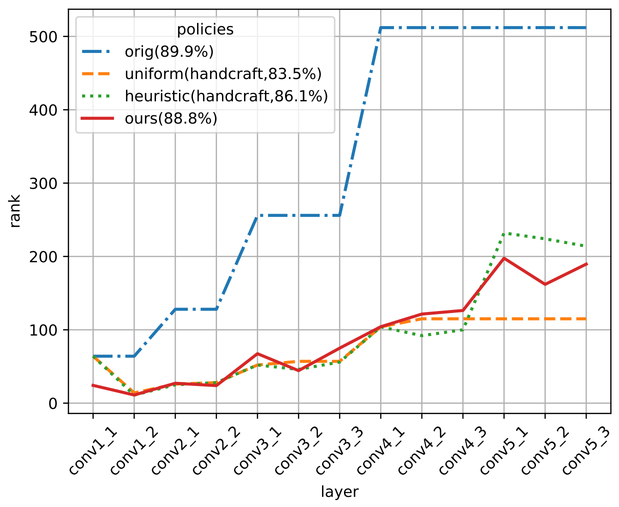

Figure 6. Comparisons of policies for decomposing VGG-16 [51] under $4\times$ FLOPs reduction (top-5 accuracy shown). $Uniform$ compresses each layer with the same sparsity ratio. $Heuristic$ [61] is based on scoring each filter with PCA energy and computation. Although the differences between the strategies are subtle, we gracefully outperformed The *heuristic* approach by 2.7%. (*better viewed in color*)

of `conv1` to `conv3` layers, and their computation contribution is also as low as 3% per layer. [23] even preserves all `conv5` channels for $4\times, 5\times$ FLOPs reduction. However, ADC is still able to squeeze out some computation from these sensitive layers. These patterns can not be easily discovered by analyzing PCA energy and computation.

By investigating the sparsity ratio settings in Fig. 6, we notice that for most layers before `conv5`, the sparsity ratio difference between policies is only about 3%. For `conv5`, the difference is around 20%. Although the sparsity ratio differences are subtle, the performance changes drastically (Table 3). Fine-grained action space can be detrimental to compression agents [3] using discrete action space [58, 44]. However, our DDPG agent naturally overcomes this problem with continuous control (Sec. 3.6).

4.2.2 Generalization Ability

To evaluate the generalization ability of the compressed model for transfer learning on the PASCAL VOC 2007 object detection task [16], we test our compressed VGG-16 ($4\times$ FLOP reduction, 1.1% increase of error by channel decomposition, Sec. 4.2.1). We simply use our compressed VGG-16 as the backbone for Faster R-CNN [50], and train with the same setting as the publicly released code[6]. The performance is evaluated by both mean Average Precision (mAP) and mAP[.5, .95] (the primary challenge metric of COCO [39]).

Shown in Table. 4, although our compressed model is only 0.6% better than [23] on ImageNet **classification**, the margin becomes larger on **object detection** (**2.1%** better mAP[.5, .95]). Our ADC even surpasses the baseline by **0.5%**. We conjecture that this significant gain occurs because of our precise compression policy and the produced compact architecture, which serves as good regularization.

	mAP (%)	mAP [.5, .95] (%)
baseline	68.7	36.7
$2\times$ handcrafted [23]	68.3 (-0.4)	36.7 (-0.0)
$4\times$ handcrafted [23]	66.9 (-1.8)	35.1 (-1.6)
$4\times$ handcrafted [61]	67.8 (-0.9)	36.5 (-0.2)
$4\times$ ADC (ours)	**68.8 (+0.1)**	**37.2 (+0.5)**

Table 4. Comparisons of compressed VGG-16 for Faster R-CNN detection on PASCAL VOC 2007. Consistent with recognition performance, the better-compressed model produced by our ADC also results in better performance on object detection task. On TITAN Xp, the actual inference times are 68ms, 56ms ,53ms, 55ms and 61ms respectively.

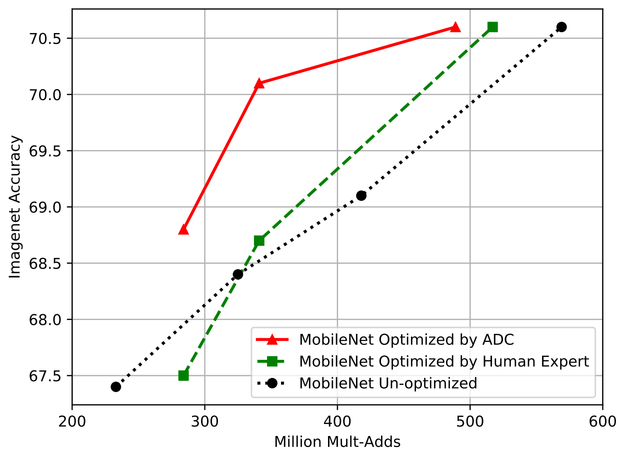

Figure 7. Pruning MobileNet [24]. The original MobileNets are shown as connected black dots. The result obtained by *shallow* policy is even worse, slightly, than that of the original MobileNets. With our ADC policies, channel pruning greatly boosts MobileNets' performances. Our R_{FLOPs} reward naturally reduces to 86% FLOPs without degradation. (*better viewed in color*)

4.2.3 Pruning Compact Networks

Recently, compactly designed networks [26, 24, 60] emerged for the increasing needs of faster and smaller networks. In this paper, we focus on pruning 1.0 MobileNet [24], a highly compact network consisting of depth-wise convolution and pointwise convolution layers. Pre-

[6]https://github.com/rbgirshick/py-faster-rcnn

	Million MAC	top-1 acc. (%)	top-5 acc. (%)	GPU latency (ms)	GPU speed	Android sep. 1×1 (ms)	Android depth.3×3 (ms)	Android total (ms)	Android speed	Android memory (VSS)
1.0 MobileNet	569	70.6	89.5	2.20	455 fps (1 ×)	76.4	26.1	107.1	9.3 fps (1 ×)	226MB
0.75 MobileNet	325	68.4	88.2	1.75	571 fps (1.25×)	51.6	26.2	82.2	12.2 fps (1.30×)	187MB
0.6× **ADC (ours)**	341	70.1	89.4	1.72	**581 fps (1.28×)**	47.1	21.1	72.6	**13.8 fps (1.48×)**	177MB
0.5× **ADC (ours)**	285	68.8	88.6	1.52	**658 fps (1.44×)**	40.8	19.8	65.1	**15.4 fps (1.65×)**	151MB

Table 5. Comparisons for pruning MobileNet [24] under different speedup ratios. Previous attempts using *hand-crafted* policy to prune MobileNets lead to significant accuracy degradation [36]. With our ADC pruning policy, the pruned models greatly improve MobileNets. In addition, we also gain gracefully inference time speed up on GPU. (Inference time is tested on GeForce GTX TITAN Xp with a mini batch size of 50 (the reported latency is amortized latency for each image), CUDA 8 [46] and cuDNN 6 [10].) On an Android phone, the speed-up is even better than on GPU. We achieved $1.87\times$ speed up for 1×1 convolution layers and $1.65\times$ speed up overall, under $2\times$ FLOPs reduction. Our pruned models are faster and more accurate than 0.75 MobileNet.

vious attempts using *hand-crafted* policy to prune MobileNets led to significant accuracy degradation [36]. Pruning 1.0 MobileNet to 75.5% original parameters results in 67.2% top-1 accuracy[7], which is even worse than the original 0.75 MobileNet (61.9% parameters with 68.4% top-1 accuracy). However, our ADC pruning policy significantly improves pruning quality: ADC-pruned MobileNet achieved 68.8% Top-1 accuracy with 285 FLOPs compared to the original 0.75 MobileNet's 68.4% Top-1 accuracy with 325 FLOPs, all evaluated on ImageNet.

As shown in Fig. 7, empirical policy *shallow* [23] is slightly worse than that of the original MobileNets under $2\times$ FLOPs reduction. With our ADC pruning policy, the pruned models gracefully improve MobileNets. For $2\times$ FLOPs reduction, we arrive at **68.8**% top-1 accuracy, improving empirical policy from 67.5%. With R_{FLOPs} reward, we prune 14% computation without any degradation. As shown in Table. 5, we gain visible inference time improvement on GPU. Our $0.6\times$ pruned model achieve **1.28**$\times$ real time speed up, with only **0.1**% increase of top-5 error. Also note that limited by Caffe, the augmentation we used during fine-tuning is weaker than [24].

We further test our pruned model on an Android phone[8]. The speed up is even better. We achieved **1.87**$\times$ speed up for 1×1 convolution layers and **1.65**$\times$ speed up overall, under $2\times$ FLOPs reduction. There's no much real time speed up for depthwise convolution. Notice that, though depthwise convolutional layers contribute far less computation, they still consume a lot of time. Efficient implementation of depthwise convolution is difficult due to the small ratio of computation over memory fetch. The compressed models also consume less memory (VSS, Virtual Set Size).

MobileNets are already highly compact. It is hard for *hand-crafted* policies to gain speed-up. Our ADC's precise policies are essential for pruning to succeed. We further demonstrate ADC also works on even more compact neural networks, which have architecture that is found by reinforcement learning, such as NASNet [65]. The results are in the appendix.

[7]http://machinethink.net/blog/compressing-deep-neural-nets/

[8]With TensorFlow 1.3 32-bit float on Galaxy S7 Edge with Qualcomm Snapdragon 820 SoC.

5. Conclusion

Conventional model compression techniques use handcrafted features and require domain experts to explore the large design space and trade off model size, speed, and accuracy, which is usually suboptimal and labor-consuming. In this paper, we propose Automated Deep Compression (ADC), which leverages reinforcement learning to automatically search the design space, greatly improving the model compression quality. We designed two novel reward schemes to perform both FLOPs-constrained compression and accuracy-guaranteed compression. Compelling results have been demonstrated for MobileNet, ResNet, and VGG Net on CIFAR-10, the complex ImageNet. The compressed model generalizes well to object detection on PASCAL VOC. On the Galaxy S7 mobile phone, we pushed the inference speed from 9.3 fps to 15.4 fps. Our ADC technique facilitates deep vision on mobile devices.

Acknowledgement

We would love to express our appreciation to Quoc Le, Raghu Krishnamoorthi, Mingxing Tan, Pete Warden, Samy Bengio, Jeff Dean, Fei-Fei Li, and Bill Dally for the useful discussions during the course of this research.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia,

R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 4

[2] S. Anwar and W. Sung. Compact deep convolutional neural networks with coarse pruning. *arXiv preprint arXiv:1610.09639*, 2016. 2

[3] A. Ashok, N. Rhinehart, F. Beainy, and K. M. Kitani. N2n learning: Network to network compression via policy gradient reinforcement learning. *arXiv preprint arXiv:1709.06030*, 2017. 2, 7

[4] H. Bagherinezhad, M. Rastegari, and A. Farhadi. Lcnn: Lookup-based convolutional neural network. *arXiv preprint arXiv:1611.06473*, 2016. 2

[5] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016. 3, 4

[6] A. Brock, T. Lim, J. M. Ritchie, and N. Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017. 2

[7] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang. Reinforcement learning for architecture search by network transformation. *arXiv preprint arXiv:1707.04873*, 2017. 2, 4, 5

[8] A. Canziani, A. Paszke, and E. Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016. 3

[9] T. Chen, I. Goodfellow, and J. Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015. 2

[10] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cudnn: Efficient primitives for deep learning. *CoRR*, abs/1410.0759, 2014. 8

[11] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016. 2

[12] M. Courbariaux and Y. Bengio. Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016. 2

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009. 2, 6

[14] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277, 2014. 2

[15] X. Dong, J. Huang, Y. Yang, and S. Yan. More is less: A more complicated network with less inference complexity. *arXiv preprint arXiv:1703.08651*, 2017. 2

[16] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html. 7

[17] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015. 2

[18] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014. 2

[19] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 1, 2

[20] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015. 1, 2, 4, 5

[21] B. Hassibi and D. G. Stork. *Second order derivatives for network pruning: Optimal brain surgeon*. Morgan Kaufmann, 1993. 2

[22] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 2, 5

[23] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1389–1397, 2017. 1, 2, 4, 5, 6, 7, 8, 11

[24] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 2, 7, 8

[25] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016. 2

[26] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 1mb model size. *arXiv preprint arXiv:1602.07360*, 2016. 7

[27] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 4

[28] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014. 1, 2, 3, 4, 6, 11

[29] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014. 4

[30] N. Johnson, S. Kotz, and N. Balakrishnan. Continuous univariate probability distributions,(vol. 1), 1994. 1, 3

[31] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015. 2

[32] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009. 2, 4, 5, 6

[33] A. Lavin. Fast algorithms for convolutional neural networks. *arXiv preprint arXiv:1509.09308*, 2015. 2

[34] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014. 2

[35] Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel. Optimal brain damage. In *NIPs*, volume 2, pages 598–605, 1989. 2

[36] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016. 2, 5, 6, 8

[37] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. 3

[38] J. Lin, Y. Rao, and J. Lu. Runtime neural pruning. In *Advances in Neural Information Processing Systems*, pages 2178–2188, 2017. 6

[39] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, Cham, 2014. 7

[40] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. *arXiv preprint arXiv:1707.06342*, 2017. 2

[41] M. Masana, J. van de Weijer, L. Herranz, A. D. Bagdanov, and J. M. Alvarez. Domain-adaptive deep network compression. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 2

[42] M. Mathieu, M. Henaff, and Y. LeCun. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*, 2013. 2

[43] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, A. Navruzyan, N. Duffy, and B. Hodjat. Evolving deep neural networks. *arXiv preprint arXiv:1703.00548*, 2017. 2

[44] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. 7

[45] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient inference. 2016. 2

[46] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with CUDA. *ACM Queue*, 6(2):40–53, 2008. 8

[47] A. Polyak and L. Wolf. Channel-level acceleration of deep face representations. *IEEE Access*, 3:2163–2175, 2015. 2

[48] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016. 2

[49] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, Q. Le, and A. Kurakin. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041*, 2017. 2

[50] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 7

[51] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 2, 6, 7

[52] S. Srinivas and R. V. Babu. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015. 4

[53] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002. 2

[54] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015. 2

[55] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun. Fast convolutional nets with fbfft: A gpu performance evaluation. *arXiv preprint arXiv:1412.7580*, 2014. 2

[56] H. Wang, Q. Zhang, Y. Wang, and R. Hu. Structured probabilistic pruning for deep convolutional neural network acceleration. *arXiv preprint arXiv:1709.06994*, 2017. 6

[57] C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989. 3

[58] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992. 7

[59] J. Xue, J. Li, and Y. Gong. Restructuring of deep neural network acoustic models with singular value decomposition. In *INTERSPEECH*, pages 2365–2369, 2013. 2

[60] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017. 7

[61] X. Zhang, J. Zou, K. He, and J. Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE transactions on pattern analysis and machine intelligence*, 38(10):1943–1955, 2016. 2, 6, 7, 11

[62] Z. Zhong, J. Yan, and C.-L. Liu. Practical network blocks design with q-learning. *arXiv preprint arXiv:1708.05552*, 2017. 3

[63] C. Zhu, S. Han, H. Mao, and W. J. Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016. 2

[64] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 4

[65] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017. 2, 3, 5, 8, 11

Supplementary Materials

A. Pruning NASNet

We further applied ADC to NASNet [65] with R_{FLOPs} reward. Pruning NASNet is challenging, since it is highly compact already. It has fewer FLOPs than MobileNet, but has more than 3% better Top-1 accuracy on ImageNet. However, with automatic deep compression, we can still squeeze out computation from `NASNet-A_Mobile_224` (Fig. 8), because ADC provides finer-grained action space than NAS. ADC is general purpose and works for all convolutional neural networks. We showed the result on MobileNet and NASNet in Fig. 8, both without any loss of accuracy. Due to the time limit, our current accuracy-guaranteed result is without fine-tuning. NASNet has the potential to be more aggressively pruned by allowing fine-tuning.

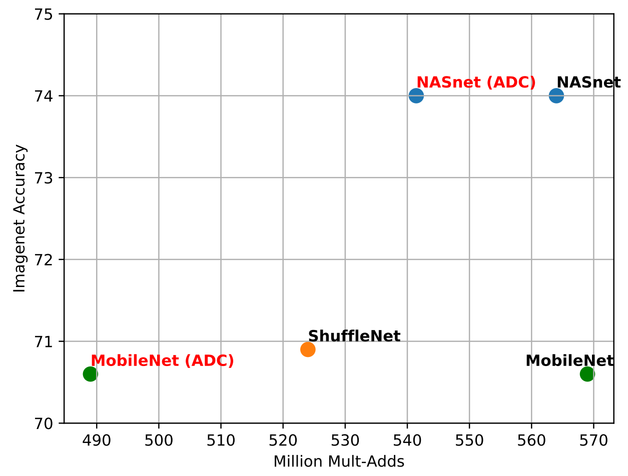

Figure 8. ADC makes NASNet more efficient. Although the architecture of NASNet [65] was already very efficient, we can still squeezed out computation from NASNet without losing any accuracy.

B. Environments

We experimented on three environments (three compression algorithms). Our ADC can also be extended to other environments. For a convolutional layer with filters $n \times c \times k_h \times k_w$, we need to determine rank c' for each algorithm.

Channel Pruning. Illustrated in Fig. 9 (b), channel pruning [23] first prunes redundant channels according to the magnitude of each filter. We sort each filter by the L2 norm and remove the filter with the smallest L2 norm. Therefore, the weights become $n \times c' \times k_h \times k_w$. Then linear regression is used to minimize the error on output feature maps and reconstruct the filters.

Spatial Decomposition. Illustrated in Fig. 9 (c), spatial decomposition [28] decomposes a layer into $c' \times c \times k_h \times 1$ and $n \times c' \times 1 \times k_w$ using SVD.

Channel Decomposition. Illustrated in Fig. 9 (d), channel decomposition [61] decomposes a layer into $c' \times c \times k_h \times k_w$ and $n \times c' \times 1 \times 1$ using GSVD.

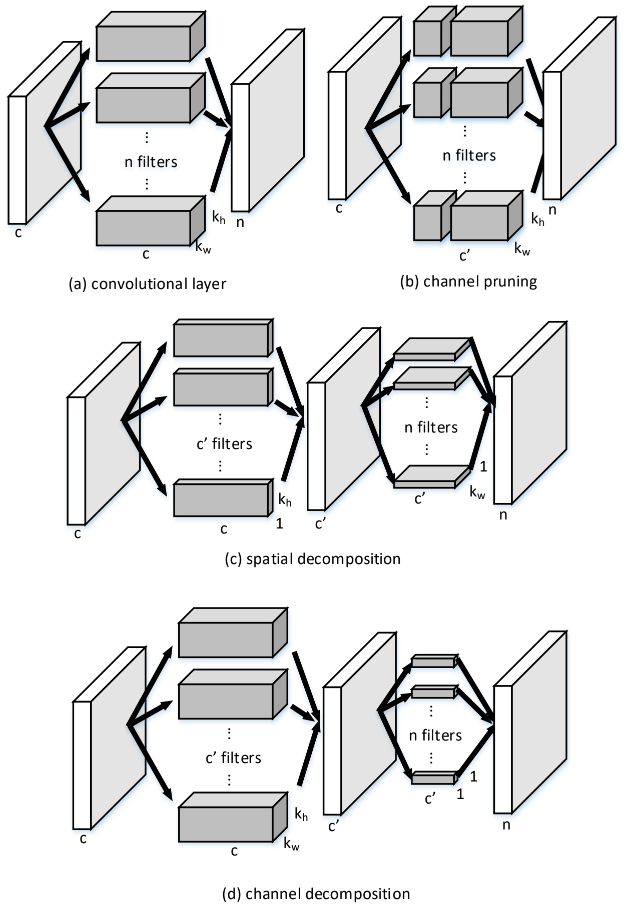

Figure 9. Three compression algorithms used in our experiments.

C. Deep Deterministic Policy Gradient

We used the deep deterministic policy gradient (DDPG) for contiguous control of the compression ratio, which is an off-policy actor-critic algorithm (Sec. 3.6). Our actor network μ had two hidden layers, each with 300 units. The final output layer was a sigmoid layer to bound the actions within $(0, 1)$. Our critic network Q also had two hidden layers, each with 300 units. Actions were included in the second hidden layer. We used $\tau = 0.01$ for the soft target updates. We trained with a batch size of 64 and a replay buffer size of 2000. For the exploration noise process, we used truncated normal distribution. Noise σ was initially 0.5 during exploration. During exploitation, it was decayed after each episode ($\sigma \leftarrow 0.95 \cdot \sigma$).

D. FLOPs-constrained for Channel Pruning

FLOPs-constrained algorithm 1 for channel pruning is shown in algorithm 2, mentioned in Sec. 3.4.

Algorithm 2 search with constrained FLOPs

```
procedure GETACTION($t$)
    if $t == T - 1$ then
        return $1$
    $desired \leftarrow$ desired FLOPs
    if $t == 0$ then
        $reduced \leftarrow 0$
        for $i = 0; i < T - 1; i{+}{+}$ do
            $input[i] = 0$
            if prunable(i) then
                $output[i] = 0$
            else
                $output[i] = 1$
    $preserved \leftarrow 0$
    $tmp \leftarrow 0$
    for $i = 0; i < T - 1; i{+}{+}$ do
        if prunable(i) and t is the next layer of i then
            $tmp \leftarrow tmp + FLOPs[i] * input[i]$
        else if $i == t$ then
            $tmp \leftarrow tmp + FLOPs[i] * output[i]$
        else
            $preserved \leftarrow preserved + FLOPs[i] * input[i] * output[i]$
    $a = \mu'(s_t)$
    return $min(a, (desired - tmp)/preserved)$

procedure PRUNABLE($i$)
    if the next layer of layer $i$ is a conv layer then
        return $True$
    return $False$
```