GhostNet: More Features from Cheap Operations

Kai Han[1] Yunhe Wang[1] Qi Tian[1*] Jianyuan Guo[2] Chunjing Xu[1] Chang Xu[3]
[1]Noah's Ark Lab, Huawei Technologies. [2]Peking University.
[3]School of Computer Science, Faculty of Engineering, University of Sydney.
{kai.han,yunhe.wang,tian.qi1,xuchunjing}@huawei.com jyguo@pku.edu.cn c.xu@sydney.edu.au

Abstract

*Deploying convolutional neural networks (CNNs) on embedded devices is difficult due to the limited memory and computation resources. The redundancy in feature maps is an important characteristic of those successful CNNs, but has rarely been investigated in neural architecture design. This paper proposes a novel Ghost module to generate more feature maps from cheap operations. Based on a set of intrinsic feature maps, we apply a series of linear transformations with cheap cost to generate many ghost feature maps that could fully reveal information underlying intrinsic features. The proposed Ghost module can be taken as a plug-and-play component to upgrade existing convolutional neural networks. Ghost bottlenecks are designed to stack Ghost modules, and then the lightweight GhostNet can be easily established. Experiments conducted on benchmarks demonstrate that the proposed Ghost module is an impressive alternative of convolution layers in baseline models, and our GhostNet can achieve higher recognition performance (*e.g.* 75.7% *top-1 accuracy) than MobileNetV3 with similar computational cost on the ImageNet ILSVRC-2012 classification dataset. Code is available at* `https://github.com/huawei-noah/ghostnet`.

1. Introduction

Deep convolutional neural networks have shown excellent performance on various computer vision tasks, such as image recognition [30, 13], object detection [43, 33], and semantic segmentation [4]. Traditional CNNs usually need a large number of parameters and floating point operations (FLOPs) to achieve a satisfactory accuracy, *e.g.* ResNet-50 [16] has about 25.6M parameters and requires 4.1B FLOPs to process an image of size 224×224. Thus, the recent trend of deep neural network design is to explore portable and efficient network architectures with acceptable performance for mobile devices (*e.g.* smart phones and self-driving cars).

*Corresponding author

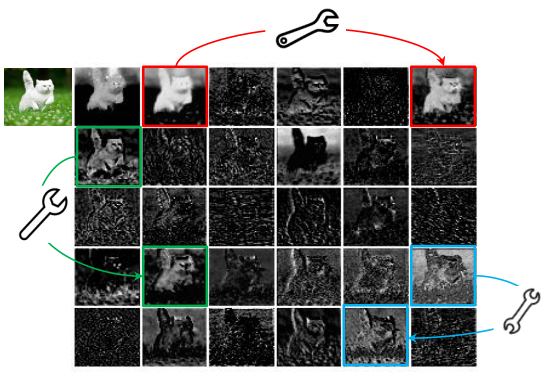


Figure 1. Visualization of some feature maps generated by the first residual group in ResNet-50, where three similar feature map pair examples are annotated with boxes of the same color. One feature map in the pair can be approximately obtained by transforming the other one through cheap operations (denoted by spanners).

Over the years, a series of methods have been proposed to investigate compact deep neural networks such as network pruning [14, 39], low-bit quantization [42, 26], knowledge distillation [19, 57], *etc*. Han *et al*. [14] proposed to prune the unimportant weights in neural networks. [31] utilized ℓ_1-norm regularization to prune filters for efficient CNNs. [42] quantized the weights and the activations to 1-bit data for achieving large compression and speed-up ratios. [19] introduced knowledge distillation for transferring knowledge from a larger model to a smaller model. However, performance of these methods are often upper bounded by pre-trained deep neural networks that have been taken as their baselines.

Besides them, efficient neural architecture design has a very high potential for establishing highly efficient deep networks with fewer parameters and calculations, and recently has achieved considerable success. This kind of methods can also provide new search unit for automatic search methods [62, 55, 5]. For instance, MobileNet [21, 44, 20] utilized the depthwise and pointwise convolutions to construct a unit for approximating the original convolutional layer with larger filters and achieved comparable performance. ShuffleNet [61, 40] further explored a channel shuffle operation

to enhance the performance of lightweight models.

Abundant and even redundant information in the feature maps of well-trained deep neural networks often guarantees a comprehensive understanding of the input data. For example, Figure 1 presents some feature maps of an input image generated by ResNet-50, and there exist many similar pairs of feature maps, like a *ghost* of each another. Redundancy in feature maps could be an important characteristic for a successful deep neural network. Instead of avoiding the redundant feature maps, we tend to embrace them, but in a cost-efficient way.

In this paper, we introduce a novel Ghost module to generate more features by using fewer parameters. Specifically, an ordinary convolutional layer in deep neural networks will be split into two parts. The first part involves ordinary convolutions but their total number will be rigorously controlled. Given the intrinsic feature maps from the first part, a series of simple linear operations are then applied for generating more feature maps. Without changing the size of output feature map, the overall required number of parameters and computational complexities in this Ghost module have been decreased, compared with those in vanilla convolutional neural networks. Based on Ghost module, we establish an efficient neural architecture, namely, GhostNet. We first replace original convolutional layers in benchmark neural architectures to demonstrate the effectiveness of Ghost modules, and then verify the superiority of our GhostNets on several benchmark visual datasets. Experimental results show that, the proposed Ghost module is able to decrease computational costs of generic convolutional layer while preserving similar recognition performance, and GhostNets can surpass state-of-the-art efficient deep models such as MobileNetV3 [20], on various tasks with fast inference on mobile devices.

The rest of the paper is organized as follows: section 2 briefly concludes the related work in the area, followed by the proposed Ghost module and GhostNet in section 3, the experiments and analysis in section 4, and finally, conclusion in section 5.

2. Related Work

Here we revisit the existing methods for lightening neural networks in two parts: model compression and compact model design.

2.1. Model Compression

For a given neural network, model compression aims to reduce the computation, energy and storage cost [14, 48, 11, 54]. Pruning connections [15, 14, 50] cuts out the unimportant connections between neurons. Channel pruning [51, 18, 31, 39, 59, 23, 35] further targets on removing useless channels for easier acceleration in practice. Model quantization [42, 24, 26] represents weights or activations in neural networks with discrete values for compression and calculation acceleration. Specifically, binarization methods [24, 42, 38, 45] with only 1-bit values can extremely accelerate the model by efficient binary operations. Tensor decomposition [27, 9] reduces the parameters or computation by exploiting the redundancy and low-rank property in weights. Knowledge distillation [19, 12, 3] utilizes larger models to teach smaller ones, which improves the performance of smaller models. The performances of these methods usually depend on the given pre-trained models. The improvement on basic operations and architectures will make them go further.

2.2. Compact Model Design

With the need for deploying neural networks on embedded devices, a series of compact models are proposed in recent years [7, 21, 44, 20, 61, 40, 53, 56]. Xception [7] utilizes depthwise convolution operation for more efficient use of model parameters. MobileNets [21] are a series of light weight deep neural networks based on depthwise separable convolutions. MobileNetV2 [44] proposes inverted residual block and MobileNetV3 [20] further utilizes AutoML technology [62, 55, 10] achieving better performance with fewer FLOPs. ShuffleNet [61] introduces channel shuffle operation to improve the information flow exchange between channel groups. ShuffleNetV2 [40] further considers the actual speed on target hardware for compact model design. Although these models obtain great performance with very few FLOPs, the correlation and redundancy between feature maps has never been well exploited.

3. Approach

In this section, we will first introduce the Ghost module to utilize a few small filters to generate more feature maps from the original convolutional layer, and then develop a new GhostNet with an extremely efficient architecture and high performance.

3.1. Ghost Module for More Features

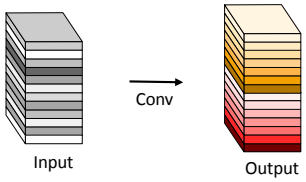
Deep convolutional neural networks [30, 46, 16] often consist of a large number of convolutions that results in massive computational costs. Although recent works such as MobileNet [21, 44] and ShuffleNet [40] have introduced depthwise convolution or shuffle operation to build efficient CNNs using smaller convolution filters (floating-number operations), the remaining 1×1 convolution layers would still occupy considerable memory and FLOPs.

Given the widely existing redundancy in intermediate feature maps calculated by mainstream CNNs as shown in Figure 1, we propose to reduce the required resources, *i.e.* convolution filters used for generating them. In practice, given the input data $X \in \mathbb{R}^{c \times h \times w}$, where c is the number of input channels and h and w are the height and width of

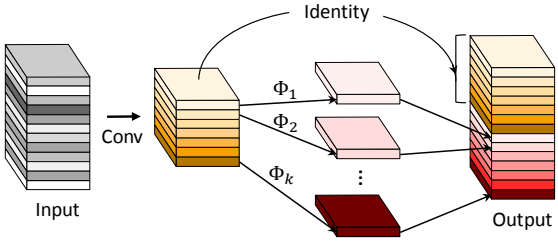
the input data, respectively, the operation of an arbitrary convolutional layer for producing n feature maps can be formulated as

$$Y = X * f + b, \tag{1}$$

where $*$ is the convolution operation, b is the bias term, $Y \in \mathbb{R}^{h' \times w' \times n}$ is the output feature map with n channels, and $f \in \mathbb{R}^{c \times k \times k \times n}$ is the convolution filters in this layer. In addition, h' and w' are the height and width of the output data, and $k \times k$ is the kernel size of convolution filters f, respectively. During this convolution procedure, the required number of FLOPs can be calculated as $n \cdot h' \cdot w' \cdot c \cdot k \cdot k$, which is often as large as hundreds of thousands since the number of filters n and the channel number c are generally very large (*e.g.* 256 or 512).



(a) The convolutional layer.



(b) The Ghost module.

Figure 2. An illustration of the convolutional layer and the proposed Ghost module for outputting the same number of feature maps. Φ represents the cheap operation.

According to Eq. 1, the number of parameters (in f and b) to be optimized is explicitly determined by the dimensions of input and output feature maps. As observed in Figure 1, the output feature maps of convolutional layers often contain much redundancy, and some of them could be similar with each other. We point out that it is unnecessary to generate these redundant feature maps one by one with large number of FLOPs and parameters. Suppose that the output feature maps are "ghosts" of a handful of intrinsic feature maps with some cheap transformations. These intrinsic feature maps are often of smaller size and produced by ordinary convolution filters. Specifically, m intrinsic feature maps $Y' \in \mathbb{R}^{h' \times w' \times m}$ are generated using a primary convolution:

$$Y' = X * f', \tag{2}$$

where $f' \in \mathbb{R}^{c \times k \times k \times m}$ is the utilized filters, $m \leq n$ and the bias term is omitted for simplicity. The hyper-parameters such as filter size, stride, padding, are the same as those in the ordinary convolution (Eq. 1) to keep the spatial size (*i.e.* h' and w') of the output feature maps consistent. To further obtain the desired n feature maps, we propose to apply a series of cheap linear operations on each intrinsic feature in Y' to generate s ghost features according to the following function:

$$y_{ij} = \Phi_{i,j}(y'_i), \quad \forall\, i = 1, ..., m, \;\; j = 1, ..., s, \tag{3}$$

where y'_i is the i-th intrinsic feature map in Y', $\Phi_{i,j}$ in the above function is the j-th (except the last one) linear operation for generating the j-th ghost feature map y_{ij}, that is to say, y'_i can have one or more ghost feature maps $\{y_{ij}\}_{j=1}^{s}$. The last $\Phi_{i,s}$ is the identity mapping for preserving the intrinsic feature maps as shown in Figure 2(b). By utilizing Eq. 3, we can obtain $n = m \cdot s$ feature maps $Y = [y_{11}, y_{12}, \cdots, y_{ms}]$ as the output data of a Ghost module as shown in Figure 2(b). Note that the linear operations Φ operate on each channel whose computational cost is much less than the ordinary convolution. In practice, there could be several different linear operations in a Ghost module, *e.g.* 3×3 and 5×5 linear kernels, which will be analyzed in the experiment part.

Difference from Existing Methods. The proposed Ghost module has major differences from existing efficient convolution schemes. i) Compared with the units in [21, 61] which utilize 1×1 pointwise convolution widely, the primary convolution in Ghost module can have customized kernel size. ii) Existing methods [21, 44, 61, 40] adopt pointwise convolutions to process features across channels and then take depthwise convolution to process spatial information. In contrast, Ghost module adopts ordinary convolution to first generate a few intrinsic feature maps, and then utilizes cheap linear operations to augment the features and increase the channels. iii) The operation to process each feature map is limited to depthwise convolution or shift operation in previous efficient architectures [21, 61, 53, 28], while linear operations in Ghost module can have large diversity. iv) In addition, the identity mapping is paralleled with linear transformations in Ghost module to preserve the intrinsic feature maps.

Analysis on Complexities. Since we can utilize the proposed Ghost module in Eq. 3 to generate the same number of feature maps as that of an ordinary convolutional layer, we can easily integrate the Ghost module into existing well designed neural architectures to reduce the computation costs. Here we further analyze the profit on memory usage and theoretical speed-up by employing the Ghost module. For example, there are 1 identity mapping and $m \cdot (s-1) = \frac{n}{s} \cdot (s-1)$ linear operations, and the averaged kernel size of each linear operation is equal to $d \times d$. Ideally, the $n \cdot (s-1)$ linear operations can have different shapes and parameters, but the online inference will be obstructed especially considering

the utility of CPU or GPU cards. Thus, we suggest to take linear operations of the same size (*e.g.* 3×3 or 5×5) in one Ghost module for efficient implementation. The theoretical speed-up ratio of upgrading ordinary convolution with the Ghost module is

$$r_s = \frac{n \cdot h' \cdot w' \cdot c \cdot k \cdot k}{\frac{n}{s} \cdot h' \cdot w' \cdot c \cdot k \cdot k + (s-1) \cdot \frac{n}{s} \cdot h' \cdot w' \cdot d \cdot d} = \frac{c \cdot k \cdot k}{\frac{1}{s} \cdot c \cdot k \cdot k + \frac{s-1}{s} \cdot d \cdot d} \approx \frac{s \cdot c}{s + c - 1} \approx s, \tag{4}$$

where $d \times d$ has the similar magnitude as that of $k \times k$, and $s \ll c$. Similarly, the compression ratio can be calculated as

$$r_c = \frac{n \cdot c \cdot k \cdot k}{\frac{n}{s} \cdot c \cdot k \cdot k + (s-1) \cdot \frac{n}{s} \cdot d \cdot d} \approx \frac{s \cdot c}{s + c - 1} \approx s, \tag{5}$$

which is equal to that of the speed-up ratio by utilizing the proposed Ghost module.

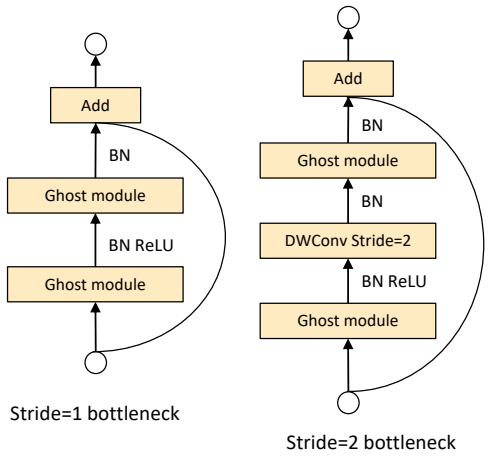


Figure 3. Ghost bottleneck. Left: Ghost bottleneck with stride=1; right: Ghost bottleneck with stride=2.

3.2. Building Efficient CNNs

Ghost Bottlenecks. Taking the advantages of Ghost module, we introduce the Ghost bottleneck (G-bneck) specially designed for small CNNs. As shown in Figure 3, the Ghost bottleneck appears to be similar to the basic residual block in ResNet [16] in which several convolutional layers and shortcuts are integrated. The proposed ghost bottleneck mainly consists of two stacked Ghost modules. The first Ghost module acts as an expansion layer increasing the number of channels. We refer the ratio between the number of the output channels and that of the input as *expansion ratio*. The second Ghost module reduces the number of channels to match the shortcut path. Then the shortcut is connected between the inputs and the outputs of these two Ghost modules. The batch normalization (BN) [25] and ReLU nonlinearity are applied after each layer, except that ReLU is not used after the second Ghost module as suggested by MobileNetV2 [44].

Table 1. Overall architecture of GhostNet. G-bneck denotes Ghost bottleneck. #exp means expansion size. #out means the number of output channels. SE denotes whether using SE module.

Input	Operator	#exp	#out	SE	Stride
$224^2 \times 3$	Conv2d 3×3	-	16	-	2
$112^2 \times 16$	G-bneck	16	16	-	1
$112^2 \times 16$	G-bneck	48	24	-	2
$56^2 \times 24$	G-bneck	72	24	-	1
$56^2 \times 24$	G-bneck	72	40	1	2
$28^2 \times 40$	G-bneck	120	40	1	1
$28^2 \times 40$	G-bneck	240	80	-	2
$14^2 \times 80$	G-bneck	200	80	-	1
$14^2 \times 80$	G-bneck	184	80	-	1
$14^2 \times 80$	G-bneck	184	80	-	1
$14^2 \times 80$	G-bneck	480	112	1	1
$14^2 \times 112$	G-bneck	672	112	1	1
$14^2 \times 112$	G-bneck	672	160	1	2
$7^2 \times 160$	G-bneck	960	160	-	1
$7^2 \times 160$	G-bneck	960	160	1	1
$7^2 \times 160$	G-bneck	960	160	-	1
$7^2 \times 160$	G-bneck	960	160	1	1
$7^2 \times 160$	Conv2d 1×1	-	960	-	1
$7^2 \times 960$	AvgPool 7×7	-	-	-	-
$1^2 \times 960$	Conv2d 1×1	-	1280	-	1
$1^2 \times 1280$	FC	-	1000	-	-

The Ghost bottleneck described above is for stride=1. As for the case where stride=2, the shortcut path is implemented by a downsampling layer and a depthwise convolution with stride=2 is inserted between the two Ghost modules. In practice, the primary convolution in Ghost module here is pointwise convolution for its efficiency.

GhostNet. Building on the ghost bottleneck, we propose GhostNet as presented in Table 7. We basically follow the architecture of MobileNetV3 [20] for its superiority and replace the bottleneck block in MobileNetV3 with our Ghost bottleneck. GhostNet mainly consists of a stack of Ghost bottlenecks with the Ghost modules as the building block. The first layer is a standard convolutional layer with 16 filters, then a series of Ghost bottlenecks with gradually increased channels are followed. These Ghost bottlenecks are grouped into different stages according to the sizes of their input feature maps. All the Ghost bottlenecks are applied with stride=1 except that the last one in each stage is with stride=2. At last a global average pooling and a convolutional layer are utilized to transform the feature maps to a 1280-dimensional feature vector for final classification. The squeeze and excite (SE) module [22] is also applied to the residual layer in some ghost bottlenecks as in Table 7. In contrast to MobileNetV3, we do not use hard-swish nonlinearity function due to its large latency. The presented architecture provides a basic design for reference, although further hyper-parameters tuning or automatic architecture searching based ghost module will

further boost the performance.

Width Multiplier. Although the given architecture in Table 7 can already provide low latency and guaranteed accuracy, in some scenarios we may require smaller and faster models or higher accuracy on specific tasks. To customize the network for the desired needs, we can simply multiply a factor α on the number of channels uniformly at each layer. This factor α is called *width multiplier* as it can change the width of the entire network. We denote GhostNet with width multiplier α as GhostNet-$\alpha\times$. Width multiplier can control the model size and the computational cost quadratically by roughly α^2. Usually smaller α leads to lower latency and lower performance, and vice versa.

4. Experiments

In this section, we first replace the original convolutional layers by the proposed Ghost module to verify its effectiveness. Then, the GhostNet architecture built using the new module will be further tested on the image classification and object detection benchmarks.

Datasets and Settings. To verify the effectiveness of the proposed Ghost module and GhostNet architecture, we conduct experiments on several benchmark visual datasets, including CIFAR-10 [29], ImageNet ILSVRC 2012 dataset [8], and MS COCO object detection benchmark [34].

CIFAR-10 dataset is utilized for analyzing the properties of the proposed method, which consists of 60,000 32×32 color images in 10 classes, with 50,000 training images and 10,000 test images. A common data augmentation scheme including random crop and mirroring [16, 18] is adopted. ImageNet is a large-scale image dataset which contains over $1.2M$ training images and $50K$ validation images belonging to 1,000 classes. The common data preprocessing strategy including random crop and flip [16] is applied during training. We also conduct object detection experiments on MS COCO dataset [34]. Following common practice [32, 33], we train models on COCO *trainval35k* split (union of $80K$ training images and a random $35K$ subset of images from validation set) and evaluate on the *minival* split with $5K$ images.

4.1. Efficiency of Ghost Module

4.1.1 Toy Experiments.

We have presented a diagram in Figure 1 to point out that there are some similar feature map pairs, which can be efficiently generated using some efficient linear operations. Here we first conduct a toy experiment to observe the reconstruction error between raw feature maps and the generated ghost feature maps. Taking three pairs in Figure 1 (*i.e*. red, greed, and blue) as examples, features are extracted using the first residual block of ResNet-50 [16]. Taking the feature on the left as input and the other one as output, we utilize a small depthwise convolution filter to learn the mapping, *i.e*. the linear operation Φ between them. The size of the convolution filter d is ranged from 1 to 7, MSE (mean squared error) values of each pair with different d are shown in Table 2.

Table 2. MSE error *v.s.* different kernel sizes.

MSE (10^{-3})	d=1	d=3	d=5	d=7
red pair	4.0	3.3	3.3	3.2
green pair	25.0	24.3	24.1	23.9
blue pair	12.1	11.2	11.1	11.0

It can be found in Table 2 that all the MSE values are extremely small, which demonstrates that there are strong correlations between feature maps in deep neural networks and these redundant feature maps could be generated from several intrinsic feature maps. Besides convolutions used in the above experiments, we can also explore some other low-cost linear operations to construct the Ghost module such as affine transformation and wavelet transformation. However, convolution is an efficient operation already well support by current hardware and it can cover a number of widely used linear operations such as smoothing, blurring, motion, *etc*. Moreover, although we can also learn the size of each filter w.r.t. the linear operation Φ, the irregular module will reduce the efficiency of computing units (*e.g*. CPU and GPU). Thus, we suggest to let d in a Ghost module be a fixed value and utilize depthwise convolution to implement Eq. 3 for building highly efficient deep neural networks in the following experiments.

Table 3. The performance of the proposed Ghost module with different d on CIFAR-10.

d	Weights (M)	FLOPs (M)	Acc. (%)
VGG-16	15.0	313	93.6
1	7.6	157	93.5
3	7.7	158	93.7
5	7.7	160	93.4
7	7.7	163	93.1

Table 4. The performance of the proposed Ghost module with different s on CIFAR-10.

s	Weights (M)	FLOPs (M)	Acc. (%)
VGG-16	15.0	313	93.6
2	7.7	158	93.7
3	5.2	107	93.4
4	4.0	80	93.0
5	3.3	65	92.9

4.1.2 CIFAR-10.

We evaluate the proposed Ghost module on two popular network architectures, *i.e*. VGG-16 [46] and ResNet-56 [16],

on CIFAR-10 dataset. Since VGG-16 is originally designed for ImageNet, we use its variant [60] which is widely used in literatures for conducting the following experiments. All the convolutional layers in these two models are replaced by the proposed Ghost module, and the new models are denoted as Ghost-VGG-16 and Ghost-ResNet-56, respectively. Our training strategy closely follows the settings in [16], including momentum, learning rate, *etc*. We first analyze the effects of the two hyper-parameters s and d in Ghost module, and then compare the Ghost-models with the state-of-the-art methods.

Analysis on Hyper-parameters. As described in Eq. 3, the proposed Ghost Module for efficient deep neural networks has two hyper-parameters, *i.e*. s for generating $m = n/s$ intrinsic feature maps, and kernel size $d \times d$ of linear operations (*i.e*. the size of depthwise convolution filters) for calculating ghost feature maps. The impact of these two parameters are tested on the VGG-16 architecture.

First, we fix $s = 2$ and tune d in $\{1, 3, 5, 7\}$, and list the results on CIFAR-10 validation set in Table 3. We can see that the proposed Ghost module with $d = 3$ performs better than smaller or larger ones. This is because that kernels of size 1×1 cannot introduce spatial information on feature maps, while larger kernels such as $d = 5$ or $d = 7$ lead to overfitting and more computations. Therefore, we adopt $d = 3$ in the following experiments for effectiveness and efficiency.

After investigating the kernel sizes used in the proposed Ghost module, we keep $d = 3$ and tune the other hyper-parameter s in the range of $\{2, 3, 4, 5\}$. In fact, s is directly related to the computational costs of the resulting network, that is, larger s leads to larger compression and speed-up ratio as analyzed in Eq. 5 and Eq. 4. From the results in Table 4, when we increase s, the FLOPs are reduced significantly and the accuracy decreases gradually, which is as expected. Especially when $s = 2$ which means compress VGG-16 by $2\times$, our method performs even slightly better than the original model, indicating the superiority of the proposed Ghost module.

Table 5. Comparison of state-of-the-art methods for compressing VGG-16 and ResNet-56 on CIFAR-10. - represents no reported results available.

Model	Weights	FLOPs	Acc. (%)
VGG-16	15M	313M	93.6
ℓ_1-VGG-16 [31, 37]	5.4M	206M	93.4
SBP-VGG-16 [18]	-	136M	92.5
Ghost-VGG-16 (s=2)	7.7M	158M	**93.7**
ResNet-56	0.85M	125M	93.0
CP-ResNet-56 [18]	-	63M	92.0
ℓ_1-ResNet-56 [31, 37]	0.73M	91M	92.5
AMC-ResNet-56 [17]	-	63M	91.9
Ghost-ResNet-56 (s=2)	0.43M	63M	**92.7**

Comparison with State-of-the-arts. We compare GhostNet with several representative state-of-the-art models on both VGG-16 and ResNet-56 architectures. The compared methods include different types of model compression approaches, ℓ_1 pruning [31, 37], SBP [18], channel pruning (CP) [18] and AMC [17]. For VGG-16, our model can obtain an accuracy slightly higher than the original one with a $2\times$ acceleration, which indicates that there is considerable redundancy in the VGG model. Our Ghost-VGG-16 ($s = 2$) outperforms the competitors with the highest performance (93.7%) but with significantly fewer FLOPs. For ResNet-56 which is already much smaller than VGG-16, our model can achieve comparable accuracy with baseline with $2\times$ speed-up. We can also see that other state-of-the-art models with similar or larger computational cost obtain lower accuracy than ours.

Visualization of Feature Maps. We also visualize the feature maps of our ghost module as shown in Figure 4. Although the generated feature maps are from the primary feature maps, they exactly have significant difference which means the generated features are flexible enough to satisfy the need for the specific task.

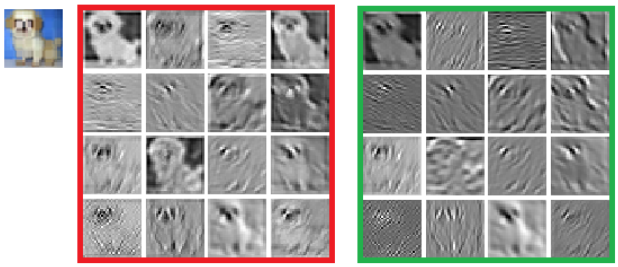


Figure 4. The feature maps in the 2nd layer of Ghost-VGG-16. The left-top image is the input, the feature maps in the left red box are from the primary convolution, and the feature maps in the right green box are after the depthwise transformation.

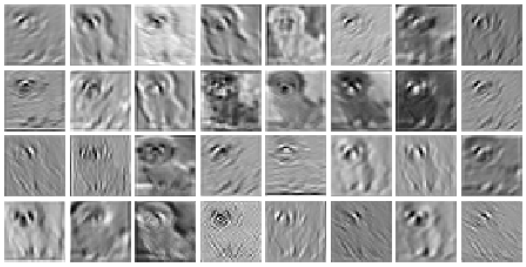


Figure 5. The feature maps in the 2nd layer of vanilla VGG-16.

4.1.3 Large Models on ImageNet

We next embed the Ghost module in the standard ResNet-50 [16] and conduct experiments on the large-scale ImageNet dataset. ResNet-50 has about 25.6M parameters and 4.1B FLOPs with a top-5 error of 7.8%. We use our Ghost

Table 6. Comparison of state-of-the-art methods for compressing ResNet-50 on ImageNet dataset.

Model	Weights (M)	FLOPs (B)	Top-1 Acc. (%)	Top-5 Acc. (%)
ResNet-50 [16]	25.6	4.1	75.3	92.2
Thinet-ResNet-50 [39]	16.9	2.6	72.1	90.3
NISP-ResNet-50-B [59]	14.4	2.3	-	90.8
Versatile-ResNet-50 [49]	11.0	3.0	74.5	91.8
SSS-ResNet-50 [23]	-	2.8	74.2	91.9
Ghost-ResNet-50 (s=2)	13.0	2.2	**75.0**	**92.3**
Shift-ResNet-50 [53]	6.0	-	70.6	90.1
Taylor-FO-BN-ResNet-50 [41]	7.9	1.3	71.7	-
Slimmable-ResNet-50 0.5× [58]	6.9	1.1	72.1	-
MetaPruning-ResNet-50 [36]	-	1.0	73.4	-
Ghost-ResNet-50 (s=4)	6.5	1.2	**74.1**	**91.9**

module to replace all the convolutional layers in ResNet-50 to obtain compact models and compare the results with several state-of-the-art methods, as detailed in Table 6. The training settings such as the optimizer, the learning rate, and the batch size, are totally the same as those in [16] for fair comparisons.

From the results in Table 6, we can see that our Ghost-ResNet-50 (s=2) obtains about 2× acceleration and compression ratio, while maintaining the accuracy as that of the original ResNet-50. Compared with the recent state-of-the-art methods including Thinet [39], NISP [59], Versatile filters [49] and Sparse structure selection (SSS) [23], our method can obtain significantly better performance under the 2× acceleration setting. When we further increase s to 4, Ghost-based model has only a 0.3% accuracy drop with an about 4× computation speed-up ratio. In contrast, compared methods [53, 58] with similar weights or FLOPs have much lower performance than ours.

4.2. GhostNet on Visual Benchmarks

After demonstrating the superiority of the proposed Ghost module for efficiently generating feature maps, we then evaluate the well designed GhostNet architecture as shown in Table 7 using Ghost bottlenecks on image classification and object detection tasks, respectively.

4.2.1 ImageNet Classification

To verify the superiority of the proposed GhostNet, we conduct experiments on ImageNet classification task. We follow most of the training settings used in [61], except that the initial learning rate is set to 0.4 when batch size is 1,024 on 8 GPUs. All the results are reported with single crop top-1 performance on ImageNet validation set. For GhostNet, we set kernel size $k = 1$ in the primary convolution and $s = 2$ and $d = 3$ in all the Ghost modules for simplicity.

Several modern small network architectures are selected as competitors, including MobileNet series [21, 44, 20], ShuffleNet series [61, 40], ProxylessNAS [2], FBNet [52], MnasNet [47], *etc*. The results are summarized in Table 7. The models are grouped into three levels of computational complexity typically for mobile applications, *i.e*. ∼50, ∼150, and 200-300 MFLOPs. From the results, we can see that generally larger FLOPs lead to higher accuracy in these small networks which shows the effectiveness of them. Our GhostNet outperforms other competitors consistently at various computational complexity levels, since GhostNet is more efficient in utilizing computation resources for generating feature maps.

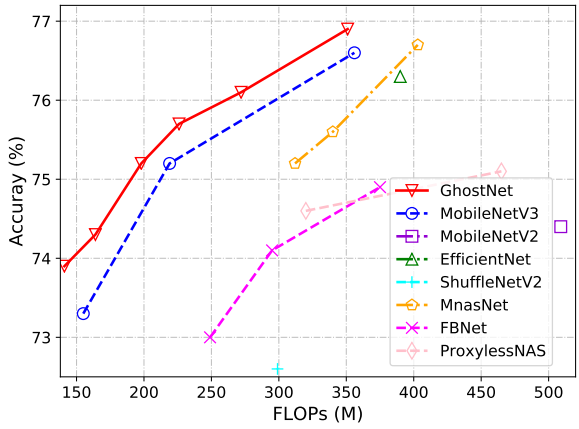


Figure 6. Top-1 accuracy *v.s.* FLOPs on ImageNet dataset.

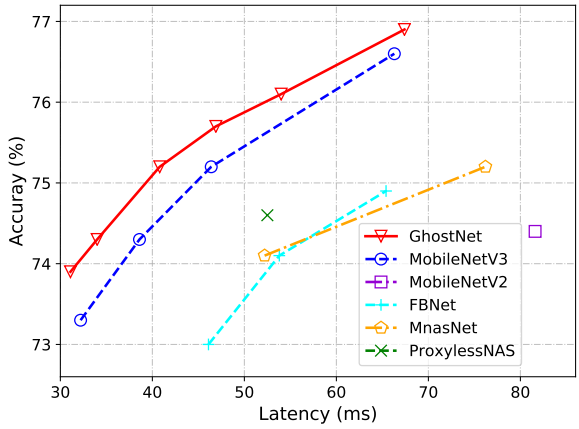


Figure 7. Top-1 accuracy *v.s.* latency on ImageNet dataset.

Table 7. Comparison of state-of-the-art small networks over classification accuracy, the number of weights and FLOPs on ImageNet dataset.

Model	Weights (M)	FLOPs (M)	Top-1 Acc. (%)	Top-5 Acc. (%)
ShuffleNetV1 0.5× (g=8) [61]	1.0	40	58.8	81.0
MobileNetV2 0.35× [44]	1.7	59	60.3	82.9
ShuffleNetV2 0.5× [40]	1.4	41	61.1	82.6
MobileNetV3 Small 0.75× [20]	2.4	44	65.4	-
GhostNet 0.5×	2.6	42	**66.2**	**86.6**
MobileNetV1 0.5× [21]	1.3	150	63.3	84.9
MobileNetV2 0.6× [44, 40]	2.2	141	66.7	-
ShuffleNetV1 1.0× (g=3) [61]	1.9	138	67.8	87.7
ShuffleNetV2 1.0× [40]	2.3	146	69.4	88.9
MobileNetV3 Large 0.75× [20]	4.0	155	73.3	-
GhostNet 1.0×	5.2	141	**73.9**	**91.4**
MobileNetV2 1.0× [44]	3.5	300	71.8	91.0
ShuffleNetV2 1.5× [40]	3.5	299	72.6	90.6
FE-Net 1.0× [6]	3.7	301	72.9	-
FBNet-B [52]	4.5	295	74.1	-
ProxylessNAS [2]	4.1	320	74.6	92.2
MnasNet-A1 [47]	3.9	312	75.2	92.5
MobileNetV3 Large 1.0× [20]	5.4	219	75.2	-
GhostNet 1.3×	7.3	226	**75.7**	**92.7**

Actual Inference Speed. Since the proposed GhostNet is designed for mobile applications, we further measure the actual inference speed of GhostNet on an ARM-based mobile phone using the TFLite tool [1]. Following the common settings in [21, 44], we use single-threaded mode with batch size 1. From the results in Figure 7, we can see that GhostNet obtain about 0.5% higher top-1 accuracy than MobileNetV3 with the same latency, and GhostNet need less runtime to achieve similar performance. For example, GhostNet with 75.0% accuracy only has 40 ms latency, while MobileNetV3 with similar accuracy requires about 45 ms to process one image. Overall, our models generally outperform the famous state-of-art models, *i.e*. MobileNet series [21, 44, 20], ProxylessNAS [2], FBNet [52], and MnasNet [47].

4.2.2 Object Detection

In order to further evaluate the generalization ability of GhostNet, we conduct object detection experiments on MS COCO dataset. We use the *trainval35k* split as training data and report the results in mean Average Precision (mAP) on *minival* split, following [32, 33]. Both the two-stage Faster R-CNN with Feature Pyramid Networks (FPN) [43, 32] and the one-stage RetinaNet [33] are used as our framework and GhostNet acts as a drop-in replacement for the backbone feature extractor. We train all the models using SGD for 12 epochs from ImageNet pretrained weights with the hyper-parameters suggested in [32, 33]. The input images are resized to a short side of 800 and a long side not exceeding 1333. Table 8 shows the detection results, where the FLOPs are calculated using 224×224 images as common practice. With significantly lower computational costs, GhostNet achieves similar mAP with MobileNetV2 and MobileNetV3, both on one-stage RetinaNet and two-stage Faster R-CNN frameworks.

Table 8. Results on MS COCO dataset.

Backbone	Detection Framework	Backbone FLOPs	mAP
MobileNetV2 1.0× [44]	RetinaNet	300M	26.7%
MobileNetV3 1.0× [20]		219M	26.4%
GhostNet 1.1×		164M	26.6%
MobileNetV2 1.0× [44]	Faster R-CNN	300M	27.5%
MobileNetV3 1.0× [20]		219M	26.9%
GhostNet 1.1×		164M	26.9%

5. Conclusion

To reduce the computational costs of recent deep neural networks, this paper presents a novel Ghost module for building efficient neural architectures. The basic Ghost module splits the original convolutional layer into two parts and utilizes fewer filters to generate several intrinsic feature maps. Then, a certain number of cheap transformation operations will be further applied for generating ghost feature maps efficiently. The experiments conducted on benchmark models and datasets illustrate that the proposed method is a plug-and-play module for converting original models to compact ones while remaining the comparable performance. In addition, the GhostNet built using the proposed new module outperforms state-of-the-art portable neural architectures, in both terms of efficiency and accuracy.

Acknowledgment

We thank anonymous reviewers for their helpful comments. Chang Xu was supported by the Australian Research Council under Project DE180101438.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019.

[3] Hanting Chen, Yunhe Wang, Chang Xu, Zhaohui Yang, Chuanjian Liu, Boxin Shi, Chunjing Xu, Chao Xu, and Qi Tian. Data-free learning of student networks. In *ICCV*, 2019.

[4] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. In *ICLR*, 2016.

[5] Wuyang Chen, Xinyu Gong, Xianming Liu, Qian Zhang, Yuan Li, and Zhangyang Wang. Fasterseg: Searching for faster real-time semantic segmentation. In *ICLR*, 2020.

[6] Weijie Chen, Di Xie, Yuan Zhang, and Shiliang Pu. All you need is a few shifts: Designing efficient convolutional neural networks for image classification. In *CVPR*, 2019.

[7] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, pages 1251–1258, 2017.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255. Ieee, 2009.

[9] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *NeurIPS*, pages 1269–1277, 2014.

[10] Xinyu Gong, Shiyu Chang, Yifan Jiang, and Zhangyang Wang. Autogan: Neural architecture search for generative adversarial networks. In *ICCV*, 2019.

[11] Shupeng Gui, Haotao N Wang, Haichuan Yang, Chen Yu, Zhangyang Wang, and Ji Liu. Model compression with adversarial robustness: A unified optimization framework. In *NeurIPS*, 2019.

[12] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor Tsang, and Masashi Sugiyama. Co-teaching: Robust training of deep neural networks with extremely noisy labels. In *NeurIPS*, 2018.

[13] Kai Han, Jianyuan Guo, Chao Zhang, and Mingjian Zhu. Attribute-aware attention model for fine-grained representation learning. In *ACM MM*, 2018.

[14] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

[15] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *NeurIPS*, pages 1135–1143, 2015.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[17] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *ECCV*, 2018.

[18] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017.

[19] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[20] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *ICCV*, 2019.

[21] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[22] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, 2018.

[23] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *ECCV*, pages 304–320, 2018.

[24] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *NeurIPS*, pages 4107–4115, 2016.

[25] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[26] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *CVPR*, pages 2704–2713, 2018.

[27] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *BMVC*, 2014.

[28] Yunho Jeon and Junmo Kim. Constructing fast network through deconstruction of convolution. In *NeurIPS*, 2018.

[29] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, pages 1097–1105, 2012.

[31] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *ICLR*, 2017.

[32] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017.

[33] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017.

[34] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*. Springer, 2014.

[35] Chuanjian Liu, Yunhe Wang, Kai Han, Chunjing Xu, and Chang Xu. Learning instance-wise sparsity for accelerating deep models. In *IJCAI*, 2019.

[36] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Tim Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. In *ICCV*, 2019.

[37] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *ICLR*, 2019.

[38] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *ECCV*, 2018.

[39] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, pages 5058–5066, 2017.

[40] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *ECCV*, 2018.

[41] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *CVPR*, 2019.

[42] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, pages 525–542. Springer, 2016.

[43] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NeurIPS*, 2015.

[44] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, pages 4510–4520, 2018.

[45] Mingzhu Shen, Kai Han, Chunjing Xu, and Yunhe Wang. Searching for accurate binary neural architectures. In *ICCV Workshops*, 2019.

[46] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[47] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, pages 2820–2828, 2019.

[48] Yue Wang, Ziyu Jiang, Xiaohan Chen, Pengfei Xu, Yang Zhao, Yingyan Lin, and Zhangyang Wang. E2-train: Training state-of-the-art cnns with over 80% energy savings. In *NeurIPS*, 2019.

[49] Yunhe Wang, Chang Xu, Chunjing XU, Chao Xu, and Dacheng Tao. Learning versatile filters for efficient convolutional neural networks. In *NeurIPS*, 2018.

[50] Yunhe Wang, Chang Xu, Shan You, Dacheng Tao, and Chao Xu. Cnnpack: packing convolutional neural networks in the frequency domain. In *NeurIPS*, pages 253–261, 2016.

[51] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *NeurIPS*, pages 2074–2082, 2016.

[52] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, pages 10734–10742, 2019.

[53] Bichen Wu, Alvin Wan, Xiangyu Yue, Peter Jin, Sicheng Zhao, Noah Golmant, Amir Gholaminejad, Joseph Gonzalez, and Kurt Keutzer. Shift: A zero flop, zero parameter alternative to spatial convolutions. In *CVPR*, 2018.

[54] Yixing Xu, Yunhe Wang, Hanting Chen, Kai Han, XU Chunjing, Dacheng Tao, and Chang Xu. Positive-unlabeled compression on the cloud. In *NeurIPS*, 2019.

[55] Zhaohui Yang, Yunhe Wang, Xinghao Chen, Boxin Shi, Chao Xu, Chunjing Xu, Qi Tian, and Chang Xu. Cars: Continuous evolution for efficient neural architecture search. *arXiv preprint arXiv:1909.04977*, 2019.

[56] Zhaohui Yang, Yunhe Wang, Chuanjian Liu, Hanting Chen, Chunjing Xu, Boxin Shi, Chao Xu, and Chang Xu. Legonet: Efficient convolutional neural networks with lego filters. In *ICML*, 2019.

[57] Shan You, Chang Xu, Chao Xu, and Dacheng Tao. Learning from multiple teacher networks. In *SIGKDD*, 2017.

[58] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *ICLR*, 2019.

[59] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *CVPR*, 2018.

[60] Sergey Zagoruyko. 92.45 on cifar-10 in torch, 2015. *URL http://torch.ch/blog/2015/07/30/cifar. html*.

[61] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CVPR*, 2018.

[62] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, pages 8697–8710, 2018.