GhostNets on Heterogeneous Devices via Cheap Operations

Kai Han[1,2] · **Yunhe Wang**[2] · **Chang Xu**[3] · **Jianyuan Guo**[2,3] · **Chunjing Xu**[2] · **Enhua Wu**[1,4] · **Qi Tian**[2]

Abstract Deploying convolutional neural networks (CNNs) on mobile devices is difficult due to the limited memory and computation resources. We aim to design efficient neural networks for heterogeneous devices including CPU and GPU, by exploiting the redundancy in feature maps, which has rarely been investigated in neural architecture design.

For CPU-like devices, we propose a novel CPU-efficient Ghost (C-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 C-Ghost module can be taken as a plug-and-play component to upgrade existing convolutional neural networks. C-Ghost bottlenecks are designed to stack C-Ghost modules, and then the lightweight C-GhostNet can be easily established. We further consider the efficient networks for GPU devices. Without involving too many GPU-inefficient operations (*e.g.*, depth-wise convolution) in a building stage, we propose to utilize the stage-wise feature redundancy to formulate GPU-efficient Ghost (G-Ghost) stage structure. The features in a stage are split into two parts where the first part is processed using the original block with fewer output channels for generating intrinsic features, and the other are generated using cheap operations by exploiting stage-wise redundancy. Experiments conducted on benchmarks demonstrate the effectiveness of the proposed C-Ghost module and the G-Ghost stage. C-GhostNet and G-GhostNet can achieve the optimal trade-off of accuracy and latency for CPU and GPU, respectively. Code is available at https://github.com/huawei-noah/CV-Backbones.

1 Introduction

Deep convolutional neural networks have shown excellent performance on various computer vision tasks, such as image recognition (Han et al., 2018b; Krizhevsky et al., 2012), object detection (Lin et al., 2017b; Ren et al., 2015), and semantic segmentation (Chen et al., 2016). Traditional CNNs usually need a large number of parameters and floating point operations (FLOPs) to achieve a satisfactory accuracy, *e.g.*, ResNet50 (He et al., 2016) 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).

Over the years, a series of methods have been proposed to investigate compact deep neural networks such as network pruning (Han et al., 2016; Luo et al., 2017), low-bit quantization (Jacob et al., 2018; Rastegari et al., 2016), knowledge distillation (Hinton et al., 2015; You et al., 2017), *etc*. Han *et.al.* (Han et al., 2016) proposed to prune the unimportant weights in neural networks. (Li et al., 2017) utilized ℓ_1-norm regularization to prune filters for efficient CNNs. (Rastegari et al., 2016) quantized the weights and the activations to 1-bit data for achieving large compression and speed-up ratios. (Hinton et al., 2015) 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.
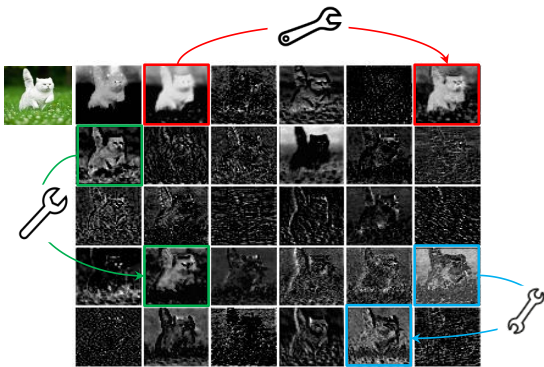
[1] State Key Lab. of Computer Science, ISCAS. & University of Chinese Academy of Sciences.
[2] Huawei Noah's Ark Lab.
[3] The University of Sydney.
[4] University of Macau.
E-mail: {hankai,weh}@ios.ac.cn,{yunhe.wang,tian.qi1}@huawei.com

Fig. 1 Visualization of some feature maps generated by the first residual group in ResNet50, 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).

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 (Chen et al., 2020b; Yang et al., 2020a; Zoph et al., 2018). For instance, MobileNet (Howard et al., 2019, 2017; Sandler et al., 2018) utilized the depth-wise and pointwise convolutions to construct a unit for approximating the original convolutional layer with larger filters and achieved comparable performance. ShuffleNet (Ma et al., 2018; Zhang et al., 2018) 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 ResNet50, 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 aim to design efficient neural networks for heterogeneous devices including CPU and GPU. For CPU devices, we introduce a novel CPU-efficient Ghost (C-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 C-Ghost module have been decreased, compared with those

in vanilla convolutional neural networks. Based on C-Ghost module, we establish an efficient neural architecture, namely, C-GhostNet. We first replace original convolutional layers in benchmark neural architectures to demonstrate the effectiveness of C-Ghost modules, and then verify the superiority of our C-GhostNet on several benchmark visual datasets. Experimental results show that, the proposed C-Ghost module is able to decrease computational costs of generic convolutional layer while preserving similar recognition performance, and C-GhostNet can surpass state-of-the-art efficient deep models such as MobileNetV3 (Howard et al., 2019), on various tasks with fast inference on mobile devices.

Moreover, we consider designing efficient CNNs for GPU devices. Most of existing FLOPs-aware lightweight architectures including the aforementioned C-GhostNet are designed for CPUs. Because the significant differences on hardware architectures of CPU and GPU, some of operations with fewer FLOPs (especially the widely-used depth-wise convolution and channel shuffle (Howard et al., 2017; Zhang et al., 2018)) are not perfectly efficient on GPU. In practice, these operators are usually of lower arithmetic intensity, *i.e.*, ratio of computation to memory operations, which cannot be sufficiently utilized by the parallel computing capability (Gholami et al., 2018; Williams et al., 2009). Thus, the reduction on latency of using them on GPUs is always not up to our expectations. In this paper, to improve the latency performance of CNN architectures on GPU devices, we present a new GPU-efficient Ghost (G-Ghost) stage for building efficient networks. Instead of decomposing each convolutional layer to several sub-layers with fewer FLOPs and parameters, we investigate the stage-wise feature redundancy across blocks in the conventional structure. Basically, a building stage in modern CNN architectures usually consists of several convolutional layers or blocks. Since intermediate features in an arbitrary building stage are of the same size (*e.g.*, 28×28 in the second stage of ResNet), the feature similarity and redundancy not only exist inside one layer but also exist across multiple layers in the stage (Figure 4). Therefore, we propose to generate intrinsic features utilizing the original stage with fewer number of channels and use cheap operation on GPUs to directly produce ghost features. To enhance the feature representation ability, we maximally excavate the information in intermediate layers and additively aggregate it into ghost features. The new G-Ghost stage can be well embedded into any mainstream CNN architectures. For the vanilla C-GhostNet, we eliminate all C-Ghost modules and using the proposed G-Ghost stage to replace original ones for establishing lightweight G-GhostNet. Experiments conducted on several benchmark models and datasets demonstrate the effectiveness of the proposed G-Ghost stage. The G-GhostNet architecture obtains highest inference speed on GPUs with state-of-the-art performance for image recognition.

The preliminary version of this work was presented as (Han et al., 2020a) on CVPR 2020. In this journal version, we extend the original work in significant ways. First, to promote the CNN speed on GPUs, we further introduce the G-Ghost stage structure by exploiting stage-wise feature redundancy. A mix operation is proposed for better preserving the intermediate representation inside a stage. Second, we build the GPU-efficient G-GhostNet architecture with G-Ghost stages, which provides a lightweight CNN with state-of-the-art performance on GPUs. Third, extensive experiments on different networks and visual tasks are conducted to demonstrate the effectiveness of our models.

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 (Chen et al., 2020a; Gui et al., 2019; Han et al., 2016; Wang et al., 2019; Xu et al., 2019). Pruning connections (Han et al., 2015, 2016; Wang et al., 2016) cuts out the unimportant connections between neurons. Channel pruning (He et al., 2017; Huang and Wang, 2018; Li et al., 2017; Liu et al., 2019a; Luo et al., 2017; Wen et al., 2016; Yu et al., 2018) further targets on removing useless channels for easier acceleration in practice. Model quantization (Hubara et al., 2016; Jacob et al., 2018; Rastegari et al., 2016; Yang et al., 2020b) represents weights or activations in neural networks with discrete values for compression and calculation acceleration. Specifically, binarization methods (Han et al., 2020b; Hubara et al., 2016; Liu et al., 2018; Rastegari et al., 2016; Shen et al., 2019) with only 1-bit values can extremely accelerate the model by efficient binary operations. Tensor decomposition (Denton et al., 2014; Han et al., 2021; Jaderberg et al., 2014) reduces the parameters or computation by exploiting the redundancy and low-rank property in weights. Knowledge distillation (Chen et al., 2019a; Guo et al., 2021; Han et al., 2018a; Hinton et al., 2015) 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 (Chollet, 2017; Howard et al., 2019, 2017; Ma et al., 2018; Sandler et al., 2018; Wu et al., 2018; Yang et al., 2019; Zhang et al., 2018). Xception (Chollet, 2017) utilizes depth-wise convolution operation for more efficient use of model parameters. MobileNets (Howard et al., 2017) are a series of light weight deep neural networks based on depth-wise separable convolutions. MobileNetV2 (Sandler et al., 2018) proposes inverted residual block and MobileNetV3 (Howard et al., 2019) further utilizes AutoML technology (Gong et al., 2019; Yang et al., 2020a; Zoph et al., 2018) achieving better performance with fewer FLOPs. ShuffleNet (Zhang et al., 2018) introduces channel shuffle operation to improve the information flow exchange between channel groups. ShuffleNetV2 (Ma et al., 2018) 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.

In addition, there are also several works studying the inference efficiency of CNNs on GPUs. SqueezeNext (Gholami et al., 2018) uses decomposed convolutions to reduce the complexity of residual block which is a kind of tensor decomposition technology. RegNet (Radosavovic et al., 2020) observes that the GPU speed is much related to the activations and gives the empirically guidance for GPU-efficient models. In fact, the structure of RegNet basically follows that of ResNet. Instead, this paper proposes a new structure which can be used to improve most of the existing CNNs including RegNet.

3 CPU-Efficient GhostNet

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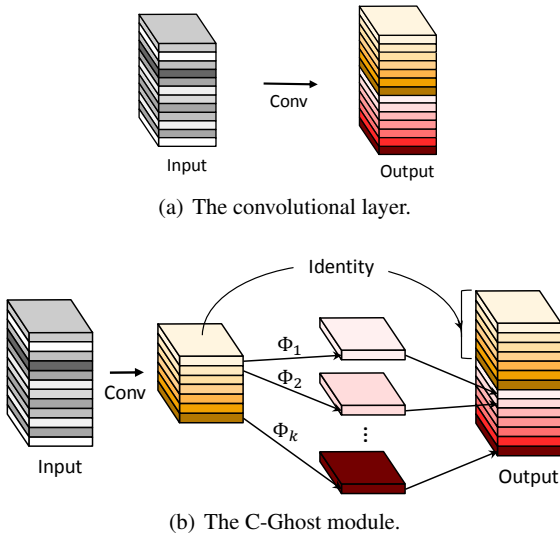
3.1 Ghost Module for More Features

Deep convolutional neural networks (He et al., 2016; Krizhevsky et al., 2012; Simonyan and Zisserman, 2015) often consist of a large number of convolutions that result in massive computational costs. Although recent works such as MobileNet (Howard et al., 2017; Sandler et al., 2018) and ShuffleNet (Ma et al., 2018) have introduced depth-wise

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 C-Ghost module.

Fig. 2 An illustration of the convolutional layer and the proposed C-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 generated by some cheap operations. 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 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) cheap operation for generating the j-th ghost feature map y_{ij}. Here y_i' can have one or more ghost feature maps $\{y_{ij}\}_{j=1}^s$. The cheap operation $\Phi_{i,j}$ can perform local feature aggregation and adjusting to generate ghost feature maps. 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 C-Ghost module as shown in Figure 2(b). Note that the cheap operations Φ operate on each channel whose computational cost is much less than the ordinary convolution. In practice, there could be several different cheap operations in a C-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 C-Ghost module has major differences from existing efficient convolution schemes. i) Compared with the units in (Howard et al., 2017; Zhang et al., 2018) which utilize 1×1 pointwise convolution widely, the primary convolution in C-Ghost module can have customized kernel size. ii) Existing methods (Howard et al., 2017; Ma et al., 2018; Sandler et al., 2018; Zhang et al., 2018) adopt pointwise convolutions to process features across channels and then take depth-wise convolution to process spatial information. In contrast, C-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 depth-wise convolution or shift operation in previous efficient architectures (Howard et al., 2017; Jeon and Kim, 2018; Wu et al., 2018; Zhang et al., 2018), while linear operations in C-Ghost module can have large diversity. iv) In addition,

the identity mapping is paralleled with linear transformations in C-Ghost module to preserve the intrinsic feature maps.

Analysis on Complexities. Since we can utilize the proposed C-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 C-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 C-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 hardware. Thus, we suggest to take linear operations of the same size (*e.g.*, 3×3 or 5×5) in one C-Ghost module for efficient implementation. The theoretical speed-up ratio of upgrading ordinary convolution with the C-Ghost module is

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

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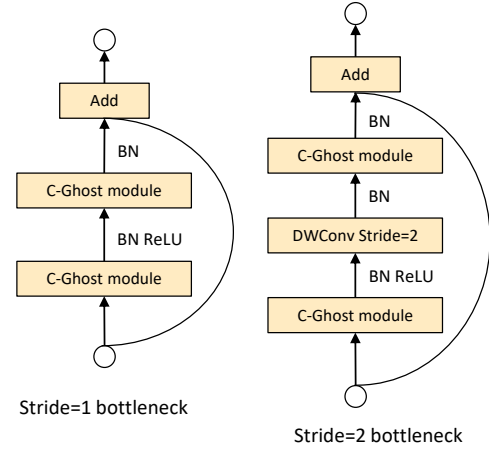
$$r_c = \frac{n \cdot c \cdot k \cdot k}{\frac{n}{s} \cdot c \cdot k \cdot k + (s-1) \cdot \frac{n}{s} \cdot d \cdot d} \approx \frac{s \cdot c}{s + c - 1} \approx s, \quad (5)$$

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

3.2 Building Lightweight C-GhostNet

C-Ghost Bottlenecks. Taking the advantages of C-Ghost module, we introduce the C-Ghost bottleneck (G-bneck) specially designed for small CNNs. As shown in Figure 3, the C-Ghost bottleneck appears to be similar to the basic residual block in ResNet (He et al., 2016) in which several convolutional layers and shortcuts are integrated. The proposed ghost bottleneck mainly consists of two stacked C-Ghost modules. The first C-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 C-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 C-Ghost modules. The batch normalization (BN) (Ioffe and Szegedy, 2015) and ReLU nonlinearity are applied after each layer, except that ReLU is not used after the second C-Ghost

module as suggested by MobileNetV2 (Sandler et al., 2018). The C-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 depth-wise convolution with stride=2 is inserted between the two C-Ghost modules. In practice, the primary convolution in C-Ghost module here is pointwise convolution for its efficiency.

Fig. 3 C-Ghost bottleneck. Left: C-Ghost bottleneck with stride=1; right: C-Ghost bottleneck with stride=2.

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

Stage	Output	Operator	#exp	#out	SE	Stride
stem	112×112	Conv3×3	-	16	-	2
	112×112	G-bneck	16	16	-	1
1	56×56	G-bneck	48	24	-	2
	56×56	G-bneck	72	24	-	1
2	28×28	G-bneck	72	40	1	2
	28×28	G-bneck	120	40	1	1
3	14×14	G-bneck	240	80	-	2
	14×14	G-bneck	200	80	-	1
	14×14	G-bneck	184	80	-	1
	14×14	G-bneck	184	80	-	1
	14×14	G-bneck	480	112	1	1
	14×14	G-bneck	672	112	1	1
4	7×7	G-bneck	672	160	1	2
	7×7	G-bneck	960	160	-	1
	7×7	G-bneck	960	160	1	1
	7×7	G-bneck	960	160	-	1
	7×7	G-bneck	960	160	1	1
head	7×7	Conv1×1	-	960	-	1
	1×1	AvgPool	-	960	-	-
	1×1	Conv1×1	-	1280	-	1
	1×1	FC	-	1000	-	-

C-GhostNet. Building on the C-Ghost bottleneck, we propose C-GhostNet as presented in Table 1. We basically follow the architecture of MobileNetV3 (Howard et al., 2019) for its superiority and replace the bottleneck block in MobileNetV3 with our C-Ghost bottleneck. C-GhostNet mainly consists of a stack of C-Ghost bottlenecks with the C-Ghost modules as the building block. The first layer is a standard convolutional layer with 16 filters, then a series of C-Ghost bottlenecks with gradually increased channels are followed. These C-Ghost bottlenecks are grouped into different stages according to the sizes of their input feature maps. All the C-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 (Hu et al., 2018) is also applied to the residual layer in some ghost bottlenecks as in Table 1. 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 1 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 C-GhostNet with width multiplier α as C-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 GPU-Efficient GhostNet

In this section, we describe the details of the proposed G-Ghost stage, a universal stage structure for building GPU-efficient network architectures.

4.1 G-Ghost Stage

Although our C-GhostNets can save FLOPs while maintaining high performance, the used cheap operation for generating more features are still not very cheap and efficient on GPU. Specifically, the depth-wise convolution is usually of lower arithmetic intensity, *i.e.*, ratio of computation to memory operations, which cannot be sufficiently utilized the

parallel computing capability (Gholami et al., 2018; Williams et al., 2009). How to derive CNNs with better trade-off between accuracy and GPU latency is still a neglected problem.
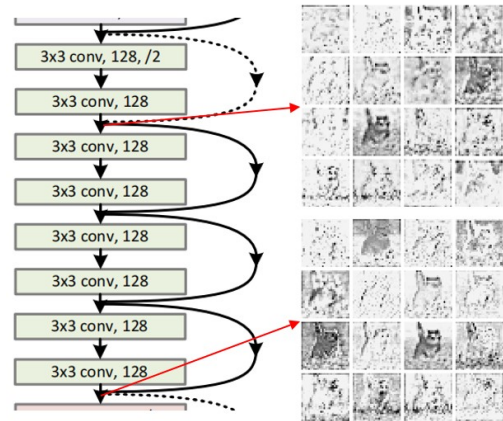
Besides the FLOPs and the number of parameters, Radosavovic et al. introduce *activations* to measure the complexity of networks, *i.e.*, the size of the output tensors of all convolutional layers (Radosavovic et al., 2020). The latency on GPUs is more relative to activations than FLOPs, that is to say, if we can remove part of feature maps and decrease activations, we can reduce the latency on GPUs with high probability. On the other hand, the main body of a CNN is usually composed of several stages with progressively reduced resolution, and each stage consists of a stack of blocks. Rather than block-wise enhancement which is investigated in previous works (Forrest et al., 2017; Sandler et al., 2018), we target on reducing *stage-wise* redundancy which can largely decrease the intermediate features and consequently cut down the related computational cost and memory usage.

The commonly-used backbone networks (He et al., 2016; Krizhevsky et al., 2012; Szegedy et al., 2015; Tan and Le, 2019) usually consists of three parts: stem, body and head (Radosavovic et al., 2020). The bulk of computation and parameters are consumed by network body, while the stem and head are relatively lightweight. Thus, here we target to simplify the network body for efficient inference. The network body is composed of several stages which process feature maps with progressively reduced resolution. For each stage in the CNN, we have n layers (*e.g.*, for AlexNet) or blocks (*e.g.*, for ResNet), denoted as $\{L_1, L_2, \cdots, L_n\}$. Given the input feature maps X, the outputs of the first block and the last block are

$$Y_1 = L_1(X), \tag{6}$$
$$Y_n = L_n(L_{n-1}(\cdots L_2(Y_1))), \tag{7}$$

respectively. To obtain the output feature Y_n, a sequence of blocks are executed to process and aggregate the input data, that is, large amount of computational cost is required.

Fig. 4 Feature maps from the first block and the last block in the second stage of ResNet34.

We argue that not all the features Y_n from the last block need to be produced using such expensive computation of multiple blocks. In Figure 4, we show the feature maps from the first block and the last block in the third stage of ResNet34. Although the features of the last block are processed by 5 more blocks, some of them are very similar to those of the first block, which means these features can be obtained by simple transformation from the low-level features.

In general, we divide the deep features into *complicated* and *ghost* ones, where the complicated features still require a sequence of blocks to generate, while the ghost ones can be easier to obtain from the shallow features. For a stage of n blocks with output features $X \in \mathbb{R}^{c \times h \times w}$ where c, h, w are the number of channels, the height and the width, respectively, we denote its complicated features as $Y_n^c \in \mathbb{R}^{(1-\lambda)c \times h \times w}$ and ghost features as $Y_n^g \in \mathbb{R}^{\lambda c \times h \times w}$, where $0 \leq \lambda \leq 1$ is the ratio of ghost features. The complicated features are produced with n blocks:

$$Y_n^c = L_n'(L_{n-1}'(\cdots L_2'(Y_1))), \tag{8}$$

where the blocks $L_2', \cdots, L_n'$ are thin blocks as Eq. 7 with $(1-\lambda)\times$ width. As analyzed in the above, the simple features Y_n^g are similar to features Y_1 of the first block, which can be viewed as ghosts of Y_1. We transform the features Y_1 to generate Y_n^g with cheap operation C:

$$Y_n^g = C(Y_1), \tag{9}$$

where cheap operation C can simply be 1×1 convolution, 3×3 convolution, *etc*. By concatenating Y_n^c and Y_n^g, we can obtain the final output of the stage:

$$Y_n = [Y_n^c, Y_n^g]. \tag{10}$$

The proposed structure as in Eq. 8-10 is called G-Ghost stage, and Figure 5(b) gives an illustration of G-Ghost stage with $\lambda = 0.5$.

4.2 Intrinsic Feature Aggregation

The proposed G-Ghost stage generates parts of features with cheap operations by exploiting the redundancy between the first block and the last block. In this way, the computational cost can be reduced largely compared to vanilla CNN stage (Figure 5(a)). Although the simple features can be approximately generated by cheap operations, Y_n^2 in Eq. 9 may lack deep information which need multiple layers to extract. To complement the lacked information, we propose to utilize the intermediate features in the complicated path to enhance the representation capacity of the cheap operations.

From the complicated path in G-Ghost stage, we can collect the intermediate features $Z \in \mathbb{R}^{c' \times h \times w} = [Y_2^c, Y_3^c, \cdots, Y_n^c]$

where c' is the total number of channels. These intermediate features extracted from multiple layers can provide rich information supplementation to the cheap operation. As shown in Figure 6, we first transform Z into the same domain as Y_n^g, and then mix the transformed information into Y_n^2:

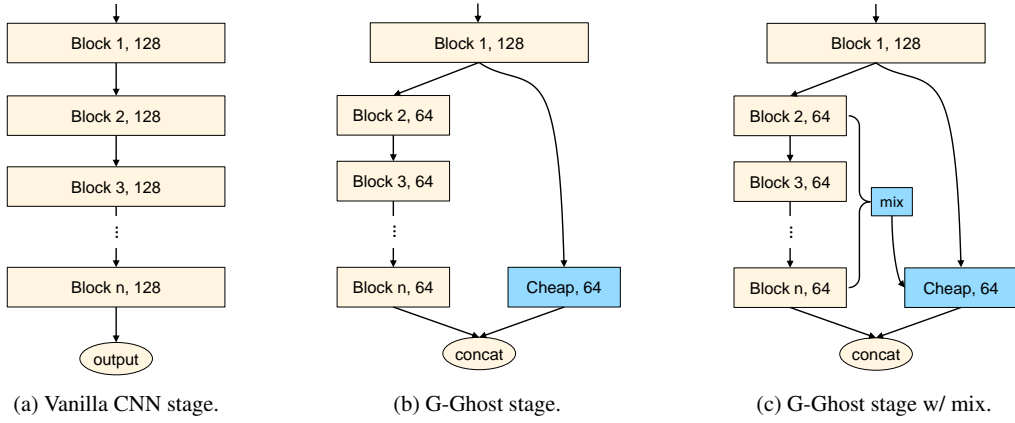$$Y_n^g \leftarrow Y_n^g + \tau(Z), \tag{11}$$

where τ is the transformation function. In order not to bring in much extra computation, we keep the transformation τ simple. It may have various candidates, and here we supply a simple design. We first apply a global average pooling to obtain the aggregated feature $z \in \mathbb{R}^{c'} = Pooling(Z)$. A fully connected layers are applied to transform z into the same domain as Y_n^2:

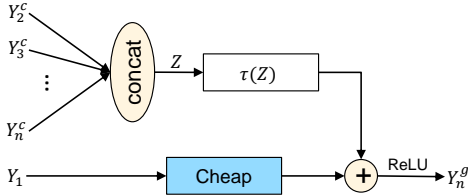$$\tau(Z) = W Pooling(Z) + b, \tag{12}$$

where $W \in \mathbb{R}^{c' \times \lambda c}$ and $b \in \mathbb{R}^{\lambda c}$ are weight and bias respectively. As shown in Figure 5(c), the mix operation is applied in G-Ghost stage for intrinsic feature aggregation.

Discussion with Existing Methods. The proposed G-Ghost stage and the existing efficient block design (*e.g.*, CondenseNet V2 (Yang et al., 2021)) both utilize the redundancy in feature maps to reduce the complexity of CNNs. They have many differences, mainly including 1) CondenseNet V2 focuses on the channel redundancy in the same layer, while G-Ghost reduces the redundancy across different layers or blocks; 2) CondenseNet V2 introduces a sparse feature reactivation (SFR) module for feature reusing in a block, while G-Ghost proposes a new structure to improve the CNN stage structure; 3) CondenseNet V2 adopts sparse connection in each SFR module which is unfriendly for memory-bound devices, while G-Ghost uses ordinary convolution and reduces activations effectively, resulting in faster inference on GPUs. 4) The mix operation in G-Ghost utilizes a global pooling (with a FC layer) to aggregate the information of intermediate layers. CondenseNet V2 utilizes a learned sparse connection to select important features from preceding layers.

Compared to the existing efficient network architecture (*e.g.*, DenseNet (Huang et al., 2017)), the proposed G-Ghost stage mainly differs at 1) the existing DenseNet is a specific network architecture, while G-Ghost stage is a general structure which can be applied to simplify most of CNNs; 2) The feature dimension in DenseNet increase as the layer goes deeper, while G-Ghost reduces the feature dimension in the intermediate layers and keeps the output as same as the original network; 4) In DenseNet, feature maps from previous blocks are concatenated to the output of each layer, while G-Ghost utilizes a cheap operation to transform the feature maps and concatenate them to the final output of the stage (not each layer); 4) We introduce a mix operation in G-Ghost stage to aggregate the information of intermediate layers.

(a) Vanilla CNN stage. (b) G-Ghost stage. (c) G-Ghost stage w/ mix.

Fig. 5 Illustration of vanilla CNN stage and the proposed G-Ghost stage, where the block can be any type of block, *e.g.*, residual block (He et al., 2016), inverted residual block (Sandler et al., 2018) and ghost block.

Fig. 6 Mix operation for aggregating the intermediate information into the ghost features.

4.3 Complexity Analysis

By upgrading the original CNN structure with the proposed G-Ghost stage, the computation and memory cost can be largely reduced. The reduction ratio is related to the original stage structure, the ghost ratio λ, the cheap operation and the mix operation. Given a stage with n blocks, the FLOPs of the ith blocks are f_i, and the parameters are p_i ($i = 1, \cdots, n$), respectively. G-Ghost stage reduces the FLOPs of the original n blocks to f_1, $(1-\lambda)f_2$, and $\{(1-\lambda)^2 f_i\}_{i=3}^n$, respectively. The parameters are reduced to p_1, $(1-\lambda)p_2$, and $\{(1-\lambda)^2 p_i\}_{i=3}^n$, respectively. The introduced cheap operation has f^c FLOPs and p^c parameters. The reduction ratio of FLOPs and parameters are

$$r_f \approx \frac{\sum_{i=1}^n f_n}{f_1 + (1-\lambda)f_2 + \sum_{i=3}^n (1-\lambda)^2 f_n + f^c}, \quad (13)$$

$$r_p \approx \frac{\sum_{i=1}^n p_n}{p_1 + (1-\lambda)p_2 + \sum_{i=3}^n (1-\lambda)^2 p_n + p^c}, \quad (14)$$

respectively. It can be seen from Eq. 13 and Eq. 14 that the FLOPs or parameters reduction is appreciable when $n \geq 3$, so we recommend to use G-Ghost stage in the stages with more than 2 blocks. In general, the larger ghost ratio λ leads to larger reduction ratio. If the ghost ratio is too large, the representation capacity of the network will be much weaker,

Table 2 Overall architecture of G-GhostNet. Block denotes residual bottleneck. output means the output spatial size. #out means the number of output channels.

Stage	Output	Operator	#out	Stride
stem	112×112	Conv3×3	16	2
1	56×56	Block	24	2
	56×56	Block×1 Cheap	24	1
	56×56	Concat	24	1
2	28×28	Block	40	2
	28×28	Block×1 Cheap	40	1
	28×28	Concat	40	1
3	14×14	Block	80	2
	14×14	Block×5 Cheap	80	1
	14×14	Concat	80	1
4	7×7	Block	160	2
	7×7	Block×5 Cheap	160	1
	7×7	Concat	160	1
head	7×7	Conv1×1	960	1
	1×1	Pool & Conv1×1	1280	1
	1×1	FC	1000	-

so we need to take a trade-off between the computational cost and the performance.

4.4 Building Lightweight G-GhostNet

We can utilize the proposed G-Ghost stage to remould the existing CNN architectures (*e.g.*, ResNet) by replacing the vanilla stage with G-Ghost stage. By exploiting stage-wise representation redundancy, G-Ghost stage can achieve a better trade-off between accuracy and GPU latency.

We also build a new lightweight network using G-Ghost stage, named G-GhostNet. The aforementioned C-GhostNet in the last section consisting of massive depth-wise convolutions is designed under FLOPs constraint and is unfriendly to GPUs. To develop a lightweight model for efficient inference on GPU devices, we build the G-GhostNet using our G-Ghost stage with mix operation. We follow the basic structure of

C-GhostNet and remould all the stages with G-Ghost stage as shown in Table 2. In all the G-Ghost stages, the 1×1 convolution is used as the cheap operation and the ghost ratio λ is set to 0.4. The expansion ratio in the residual bottleneck is set as 3, and SE module (Hu et al., 2018) is applied on each block. The simple ReLU is adopted as the activation function for its efficiency. We denote G-GhostNet-$\alpha \times$ as the G-GhostNet model with width multiplier of α.

5 Experiments

In this section, we conduct extensive experiments to verify the efficiency of the proposed C-Ghost module on CPU and G-Ghost stage on GPU.

5.1 Datasets and Experimental Settings

To verify the effectiveness of the proposed C-Ghost module and C-GhostNet architecture, we conduct experiments on several benchmark visual datasets, including CIFAR-10 (Krizhevsky and Hinton, 2009), ImageNet ILSVRC 2012 dataset (Deng et al., 2009), and MS COCO object detection benchmark (Lin et al., 2014).

CIFAR10. CIFAR10 dataset consists of 60000 32×32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images (Krizhevsky and Hinton, 2009). Data augmentation including random crop and random flip is applied.

ImageNet. ImageNet ILSVRC2012 is a large-scale image classification dataset, containing the 1000 categories and 1.2 million images. A subset of 50,000 of the images with labels are used as validation data (Deng et al., 2009). The commonly-used random crop, random clip and color jitter strategies are used for data augmentation (Szegedy et al., 2016).

MS COCO. We evaluate the generalization of our method on COCO 2014 (Lin et al., 2014) detection task. We train models on COCO *trainval35k* split (union of 80K training images and 35K subset of images from validation set) and evaluate on the *minival* split with 5K images (Lin et al., 2017a). The input image is resized to have an 800-pixel shorter side.

Implementation Details. All the models are implemented using PyTorch (Paszke et al., 2019) and trained on NVIDIA V100 GPUs. In CIFAR10 experiments, the training settings basically follow the common practice (He et al., 2019; Wilson et al., 2016). For ImageNet, we keep the training scheme the same as the baseline networks for fair comparison. ResNet models are trained for 100 epochs with hype-parameters in Pytorch official examples (Paszke et al., 2019). The settings

for RegNet models are the same as those in the original paper (Radosavovic et al., 2020). The C-GhostNet and G-GhostNet models in mobile setting are trained using the training setup as in (Howard et al., 2019). We do not use advanced data augmentation, *e.g.*, RandAugment (Cubuk et al., 2020), unless specifically noted otherwise. As for object detection experiments, we use MMDetection (Chen et al., 2019b) framework to implement our models and train them under the common $1 \times$ scheduler setup.

5.2 CPU-Efficient C-GhostNet

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

5.2.1 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 ResNet50 (He et al., 2016). Taking the feature on the left as input and the other one as output, we utilize a small depth-wise 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 3.

Table 3 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 3 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. The similarity phenomenon holds for most of images, as shown in this link: https://bit.ly/38jJtR6 (ResNet50-Group1 feature maps of randomly sampled 1000 ImageNet images). The toy experiment results hold for these images too. Besides convolutions used in the above experiments, we can also explore some other low-cost linear operations to construct the C-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, and motion. 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). Thus, we suggest to let d in a C-Ghost module be a fixed value and utilize depth-wise convolution to implement Eq. 3 for building highly efficient deep neural networks in the following experiments.

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

d	Params (M)	FLOPs (M)	Acc (%)
VGG16	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 5 The performance of the proposed C-Ghost module with different s on CIFAR-10.

s	Params (M)	FLOPs (M)	Acc (%)
VGG16	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

5.2.2 CIFAR-10 Experiments

We evaluate the proposed C-Ghost module on two popular network architectures, *i.e.*, VGG16 (Simonyan and Zisserman, 2015) and ResNet56 (He et al., 2016), on CIFAR-10 dataset. Since VGG16 is originally designed for ImageNet, we use its variant (Zagoruyko, 2015) which is widely used in literatures for conducting the following experiments. All the convolutional layers in these two models are replaced by the proposed C-Ghost module, and the new models are denoted as C-Ghost-VGG16 and Ghost-ResNet56, respectively. Our training strategy closely follows the settings in (He et al., 2016), including momentum, learning rate, *etc*. We first analyze the effects of the two hyper-parameters s and d in C-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 depth-wise convolution filters) for calculating ghost feature maps. The impact of these two parameters are tested on the VGG16 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 4. We can see that the proposed C-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 C-Ghost module, we keep $d = 3$ and tune the other hyperparameter 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 5, 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 VGG16 by $2\times$, our method performs even slightly better than the original model, indicating the superiority of the proposed C-Ghost module.

Table 6 Comparison of state-of-the-art methods for compressing VGG16 and ResNet56 on CIFAR-10. - represents no reported results available.

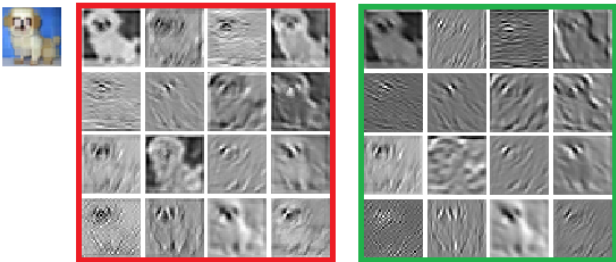| Model	Params (M)	FLOPs (M)	Acc (%)
VGG16	15	313	93.6
PFEC (Li et al., 2017)	5.4	206	93.4
CP (He et al., 2017)	-	136	92.5
C-Ghost (*s*=2)	7.7	158	**93.7**
ResNet56	0.85	125	94.1
CP (He et al., 2017)	-	63	92.0
PFEC (Li et al., 2017)	0.73	91	92.5
AMC (He et al., 2018b)	-	63	91.9
GAL (Lin et al., 2019)	0.75	78	93.0
HRank (Lin et al., 2020a)	0.49	63	93.2
C-Ghost (*s*=2)	0.43	63	**93.8**

Comparison with State-of-the-arts. We compare Ghost-Net with several representative state-of-the-art models on both VGG16 and ResNet56 architectures. The compared methods include different types of model compression approaches, PFEC (Li et al., 2017; Liu et al., 2019c), channel pruning (CP) (He et al., 2017), AMC (He et al., 2018b) and HRank (Lin et al., 2020a). For VGG16, 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 C-Ghost-VGG16 ($s = 2$) outperforms the competitors with the highest performance (93.7%) but with significantly fewer FLOPs. For ResNet56 which is already much smaller than VGG16, our model can

Table 7 Comparison of state-of-the-art methods for compressing ResNet50 on ImageNet dataset. The CPU latency is measured on an Intel CPU where - denotes no public code available.
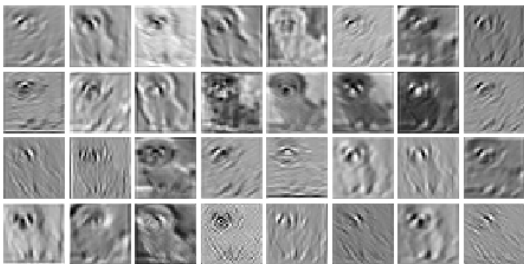
Model	Params (M)	FLOPs (B)	CPU Latency (ms)	Top-1 (%)	Top-5 (%)
ResNet50 (He et al., 2016)	25.6	4.1	158	76.1	92.8
NISP-B (Yu et al., 2018)	14.4	2.3	-	-	90.8
Versatile Filters (Wang et al., 2018)	11.0	3.0	172	74.5	91.8
SSS (Huang and Wang, 2018)	-	2.8	-	74.2	91.9
GAL-0.5 (Lin et al., 2019)	21.2	2.3	-	72.0	90.9
PFP-B (Liebenwein et al., 2020)	-	2.9	-	75.2	92.4
DSA (Ning et al., 2020)	-	2.1	174	74.7	92.1
C-Ghost (*s*=2)	13.0	2.2	129	**75.7**	**92.6**
Shift (Wu et al., 2018)	6.0	-	-	70.6	90.1
Taylor-FO-BN (Molchanov et al., 2019)	7.9	1.3	-	71.7	-
Slimmable 0.5× (Yu et al., 2019)	6.9	1.1	160	72.1	-
HRank (Lin et al., 2020a)	13.8	1.6	120	72.0	91.0
MetaPruning (Liu et al., 2019b)	-	1.0	78	73.4	-
LFPC (He et al., 2020)	-	1.6	119	74.5	92.0
C-Ghost (*s*=4)	6.5	1.2	112	**74.9**	**92.3**

achieve comparable accuracy with baseline with 2× 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 7. 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.

Fig. 7 The feature maps in the 2nd layer of C-Ghost-VGG16. 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 depth-wise transformation.

Fig. 8 The feature maps in the 2nd layer of vanilla VGG16.

5.2.3 Large Models on ImageNet

We next embed the C-Ghost module in the standard ResNet50 (He et al., 2016) and conduct experiments on the large-scale ImageNet dataset. ResNet50 has about 25.6M parameters and 4.1B FLOPs with a top-1 accuracy of 76.1%. We use our C-Ghost module to replace all the convolutional layers in ResNet50 to obtain compact models and compare the results with several state-of-the-art methods, as detailed in Table 7. The training settings such as the optimizer, learning rate, and batch size, are totally same as those in Pytorch examples (Paszke et al., 2019) for fair comparisons.

From the results in Table 7, we can see that our Ghost-ResNet50 (*s*=2) obtains about 2× acceleration and compression ratio, while maintaining the accuracy as that of the original ResNet50. Compared with the recent state-of-the-art methods including NISP (Yu et al., 2018), Versatile filters (Wang et al., 2018), Sparse structure selection (SSS) (Huang and Wang, 2018) and HRank (Lin et al., 2020a), 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 1.2% accuracy drop with an about 4× computation speed-up ratio. In contrast, compared methods (Lin et al., 2020a; Yu et al., 2019) with similar weights or FLOPs have much lower performance than ours.

5.2.4 CPU-Efficient C-GhostNet

After demonstrating the superiority of the proposed C-Ghost module for efficiently generating feature maps, we then evaluate the well designed C-GhostNet architecture as shown in Table 1 using C-Ghost bottlenecks on image classification and object detection tasks, respectively.

ImageNet Classification To verify the superiority of the proposed C-GhostNet, we conduct experiments on ImageNet
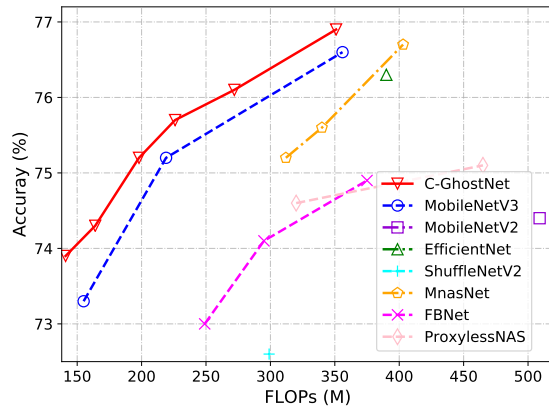
Table 8 Comparison of state-of-the-art small networks over classification accuracy, the number of weights and FLOPs on ImageNet dataset. Several models' latency is not reported because shuffle operation is not supported in TFLite or no public code is available.

Model	Params (M)	FLOPs (M)	ARM Latency (ms)	Top-1 (%)	Top-5 (%)
MobileNetV2 0.35× (Sandler et al., 2018)	1.7	59	14.5	60.3	82.9
ShuffleNetV2 0.5× (Ma et al., 2018)	1.4	41	-	61.1	82.6
MobileNetV3 Large 0.35× (Howard et al., 2019)	2.2	40	11.8	64.2	-
CondenseNetV2-A (Yang et al., 2021)	2.0	46	-	64.4	84.2
C-GhostNet 0.5×	2.6	42	10.8	**66.2**	**86.6**
MobileNetV1 0.5× (Howard et al., 2017)	1.3	150	19.7	63.3	84.9
ShuffleNetV1 1.0× (g=3) (Zhang et al., 2018)	1.9	138	-	67.8	87.7
MobileNetV2 0.75× (192×192) (Sandler et al., 2018)	2.6	153	28.6	68.7	-
ShuffleNetV2 1.0× (Ma et al., 2018)	2.3	146	-	69.4	88.9
MobileNetV3 Large 0.75× (Howard et al., 2019)	4.0	155	32.2	73.3	-
C-GhostNet 1.0×	5.2	141	31.1	**73.9**	**91.4**
MobileNetV2 1.0× (Sandler et al., 2018)	3.5	300	50.5	71.8	91.0
ShuffleNetV2 1.5× (Ma et al., 2018)	3.5	299	-	72.6	90.6
FE-Net 1.0× (Chen et al., 2019c)	3.7	301	-	72.9	-
FBNet-B (Wu et al., 2019)	4.5	295	53.8	74.1	-
MobileNeXt 1.0× (Zhou et al., 2020)	3.4	300	54.9	74.0	-
ProxylessNAS (Cai et al., 2019)	4.1	320	52.5	74.6	92.2
MnasNet-A1 (Tan et al., 2019)	3.9	312	76.2	75.2	92.5
MobileNetV3 Large 1.0× (Howard et al., 2019)	5.4	219	46.4	75.2	-
C-GhostNet 1.3×	7.3	226	46.9	**75.7**	**92.7**

classification task. We follow most of the training settings used in (Zhang et al., 2018), 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 C-GhostNet, we set kernel size $k = 1$ in the primary convolution and $s = 2$ and $d = 3$ in all the C-Ghost modules for simplicity.

Several modern small network architectures are selected as competitors, including MobileNet series (Howard et al., 2019, 2017; Sandler et al., 2018), ShuffleNet series (Ma et al., 2018; Zhang et al., 2018), ProxylessNAS (Cai et al., 2019), FBNet (Wu et al., 2019), MnasNet (Tan et al., 2019), *etc*. The results are summarized in Table 8. 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 C-GhostNet outperforms other competitors consistently at various computational complexity levels, since C-GhostNet is more efficient in utilizing computation resources for generating feature maps.

Actual Inference Speed. Since the proposed C-GhostNet is designed for mobile applications, we further measure the actual inference speed of C-GhostNet on an ARM-based mobile phone using the TFLite tool (Abadi et al., 2015). Following the common settings in (Howard et al., 2017; Sandler et al., 2018), we use single-threaded mode with batch size 1. From the results in Figure 10, we can see that C-GhostNet obtain about 0.5% higher top-1 accuracy than MobileNetV3 with the same latency, and C-GhostNet need less runtime to

Fig. 9 Top-1 accuracy *v.s.* FLOPs on ImageNet dataset.

achieve similar performance. For example, C-GhostNet with 75.2% accuracy only has 40 ms latency, while MobileNetV3 with similar accuracy requires about 46 ms to process one image. Overall, our models generally outperform the famous state-of-art models, *i.e.*, MobileNet series (Howard et al., 2019, 2017; Sandler et al., 2018), ProxylessNAS (Cai et al., 2019), FBNet (Wu et al., 2019), and MnasNet (Tan et al., 2019).

Ablation Study. From the results in Table 8, C-GhostNet has more parameters than other networks with similar FLOPs. We evaluate other manners to include more parameters without introducing more FLOPs, *e.g.*, using more SE modules, or using more channels and a smaller input resolution. We construct Net-A and Net-B which adopt the same network architecture as C-GhostNet in Table 1 and replace C-Ghost bottlenecks with inverted residual bottlenecks. SE module is
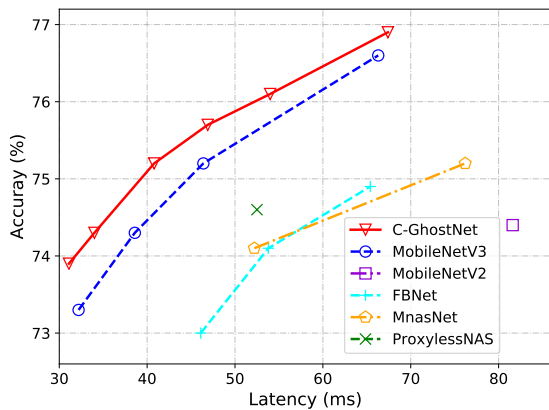
Fig. 10 Top-1 accuracy *v.s.* latency on ImageNet dataset.

applied in every block of Net-A. The input resolution of Net-B is set as 192×192. We adjust the width of Net-A and Net-B to obtain similar FLOPs to C-GhostNet. From the results in Table 9, the other manners to increase parameters perform weaker than our C-GhostNet, indicating the effectiveness of C-GhostNet.

C-Ghost module utilizes the depth-wise convolution to generate ghost features, which partly makes the model "deeper". Here we verify that the performance gain is not coming from simply making model deeper. We construct Net-C and Net-D with the same network architecture as C-GhostNet in Table 1 and replace C-Ghost bottlenecks with inverted residual bottlenecks. We add 1 more and 2 more blocks for each stage in Net-C and Net-D respectively. To keep the FLOPs similar, we adjust the width of Net-C and Net-D. From the results in Table 9, the deeper models perform much worse than our C-GhostNet, that is to say, the accuracy gain of C-GhostNet is not from simply making model deeper.

Table 9 Ablation study of C-GhostNet on ImageNet dataset.

Model	Params (M)	FLOPs (M)	Top-1 (%)
C-GhostNet 0.5×	2.6	42	66.2
Net-A	2.6	42	64.2
Net-B	2.6	46	65.7
Net-C	2.1	44	62.3
Net-D	2.0	42	61.0

Generalization to Object Detection. In order to further evaluate the generalization ability of C-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 (Lin et al., 2017a,b). The one-stage RetinaNet (Lin et al., 2017b) is used as our framework and C-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 (Lin et al., 2017a,b). The input images are re-sized to a short side of 800 and a long side not exceeding 1333. Table 10 shows the detection results, where the FLOPs are calculated using 224 × 224 images as common practice and the latency is measured on a smartphone as that in the above section. With significantly lower computational costs, C-GhostNet achieves similar mAP with MobileNetV2 and MobileNetV3.

Table 10 C-GhostNet Results on MS COCO dataset.

Backbone	Backbone FLOPs	ARM Latency	mAP
MobileNetV2 1.0×	300M	2.7 s	26.7
MobileNetV3 1.0×	219M	2.8 s	26.4
C-GhostNet 1.1×	164M	2.5 s	26.6

5.3 GPU-Efficient G-Ghost Structure

We apply the proposed G-Ghost stage on various networks, and conduct experiments on image recognition and object detection tasks to verify its effectiveness.

5.3.1 CIFAR10 Experiments

We first verify the effectiveness of our method and conduct ablation studies on CIFAR10 dataset. We use the widely-used ResNet56 as the baseline network for CIFAR10 classification.

Main Results. By upgrading all the stages with our G-Ghost stage, we construct G-Ghost-ResNet56 model. Here we set the ghost ratio $\lambda = 0.5$ and the effect of λ can be seen in the following paragraph. From the results in Table 11, G-Ghost-ResNet56 without mix operation outperform the baseline ResNet56-0.7× by a significant margin. By adding the mix operation for intrinsic feature aggregation, the performance is further improved which is 0.78% higher than baseline. G-Ghost-ResNet56 w/ mix reduces the FLOPs by 2.32× with only 0.21% accuracy drop compared to the vanilla ResNet56.

Table 11 Main results of G-Ghost stage on CIFAR10.

Model	Params (M)	FLOPs (M)	Acc (%)
ResNet56	0.85	125.0	94.1
ResNet56-0.7×	0.41	63.3	93.2
G-Ghost w/o mix	0.36	53.7	93.7
G-Ghost w/ mix	0.37	53.8	93.9

Comparison with Other Methods. We also compare our method with other state-of-the-art model compression methods or efficient models, including GAL (Lin et al., 2019), FPGM (He et al., 2019), LEGR (Chin et al., 2020), HRank (Lin et al., 2020a). As shown in Table 12, our method achieves higher accuracy with fewer FLOPs and parameters than other methods.

Table 12 Comparison with other methods on CIFAR10.

Model	Params (M)	FLOPs (M)	Acc (%)
ResNet56	0.85	125.0	94.1
PFEC (Li et al., 2017)	0.73	90.9	93.1
CP (He et al., 2017)	-	62.5	92.0
GAL (Lin et al., 2019)	0.75	78.3	93.0
FPGM (He et al., 2019)	-	59.4	93.5
LEGR (Chin et al., 2020)	-	58.9	93.7
HRank (Lin et al., 2020a)	0.49	62.7	93.2
G-Ghost w/ mix	0.37	53.8	**93.9**

Importance of Cheap Operations. The cross-layer cheap operation is one of the key-points in our method. We evaluate the importance of the cheap operation by 1) replacing it with identity mapping, 2) remove it and produce the removed ghost features from the last layer. From Table 13, we can see that whether using identity mapping or removing cheap operation performs worse than using 1×1 convolution as cheap operation.

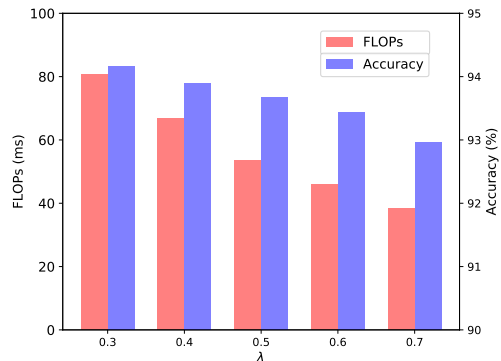**Table 13** Evaluating the importance of cheap operations.

Cheap Op	Params (M)	FLOPs (M)	Acc (%)
Identity	0.36	53.3	93.3
No	0.41	60.8	93.2
1×1 Conv	0.36	53.7	93.7

Type of Cheap Operations. There are various candidates for the cheap operation, such as convolution with different kernel size. We test different types of cheap operations where the ghost ratio λ is set as 0.5 and show the results in Table 14. The 3×3 Conv obtains the best accuracy. Including more parameters as in 5×5 Conv does not bring in higher accuracy. The 1×1 Conv achieves a good performance with fewer FLOPs and parameters, so we adopt it as cheap operation in the following experiments unless otherwise specified.

Table 14 Effect of different types of cheap operations.

Cheap Op	Params (M)	FLOPs (M)	Acc (%)
1×1 Conv	0.36	53.7	93.67
3×3 Conv	0.38	56.9	93.74
5×5 Conv	0.43	63.2	93.63

Ratio of Cheap Operations. The ghost ratio λ is a hyper-parameter controlling the ratio of cheap features and directly influences the compression ratio as in Eq. 13 and Eq. 14. We use 1×1 convolution as the cheap operation and tune λ in $[0.3, 0.4, 0.5, 0.6, 0.7]$. The results are shown in Figure 11. As λ increases, the FLOPs decrease largely while the accuracy drops gradually. We usually set the ghost ratio λ as 0.5 due to the trade-off between computational cost and performance unless otherwise specified.

Fig. 11 The impact of ghost ratio λ in G-Ghost stage.

5.3.2 ImageNet Experiments

To verify the generalization of the proposed method, we test it on the large-scale ImageNet benchmark under settings of various model sizes.

Table 15 ResNet results on ImageNet. Latency is tested on an NVIDIA Tesla V100 GPU with a batch size of 64.

Model	Params (M)	FLOPs (B)	Acts (M)	Latency (ms)	Top1 (%)	Top5 (%)
ResNet34	21.8	3.7	3.7	29.1	73.3	91.4
ResNet34-0.8×	14.0	2.3	3.0	24.0	71.7	90.3
G-Ghost w/o mix	14.3	2.3	3.3	24.1	72.8	90.9
G-Ghost w/ mix	14.6	2.3	3.3	24.5	73.1	91.2

5.3.3 ResNet

The proposed G-Ghost stage is GPU-efficient and can be used to accelerate the existing networks. We apply our method by replacing all the stage structure in ResNet34 using our G-Ghost stage. The results are shown in Table 15. Our G-Ghost-ResNet34 without mix operation outperforms ResNet34-0.8× by a significant margin with similar FLOPs and GPU latency. Including mix operation further improve the performance with negligible extra computational cost. Finally, our G-Ghost-ResNet34 with mix operation achieves comparable accuracy with original ResNet34 by reducing about 16% GPU latency.

Comparison with Network Pruning. Network pruning is a kind of widely-used method for simplifying neural networks by cutting out the unimportant channels. We compare G-Ghost-ResNet34 with the state-of-the-art pruning methods including SFP (He et al., 2018a), PFEC (Li et al., 2017), Taylor-FO-BN (Molchanov et al., 2019) and FPGM (He et al., 2019). From the results in Table 16, We can see that our G-Ghost-ResNet34 outperforms the other methods which shows the superiority of our method.

Table 16 Comparison with other methods on ImageNet.

Model	Params (M)	FLOPs (B)	Acts (M)	Top1 (%)	Top5
ResNet34	21.8	3.7	3.7	73.3	91.4
SFP (He et al., 2018a)	13.1	2.2	-	71.8	90.3
PFEC (Li et al., 2017)	19.3	2.8	-	72.2	-
Taylor-FO-BN (Molchanov et al., 2019)	17.2	2.8	-	72.8	-
FPGM (He et al., 2019)	13.1	2.2	-	72.6	91.1
ABCPruner (Lin et al., 2020b)	10.1	2.2	-	71.0	90.1
G-Ghost w/ mix	14.6	2.3	-	**73.1**	**91.2**

Fig. 12 The feature maps of the third stage of ResNet34.

Fig. 13 The feature maps of the third stage of G-Ghost-ResNet34. Left: features from complicated path, right: features from cheap operation.

Visualization of Feature Maps. To better understand the effect of the proposed G-Ghost stage, we show the feature maps of the original ResNet and the G-Ghost-ResNet in Figure 12 and Figure 13, respectively. The pattern in features of G-Ghost-ResNet34 (Figure 13) is also similar to that of the

original ResNet34 (Figure 12) so the cheap operation achieve its goal to generate informative features. The reduced complexity does not harm the capacity of the network.
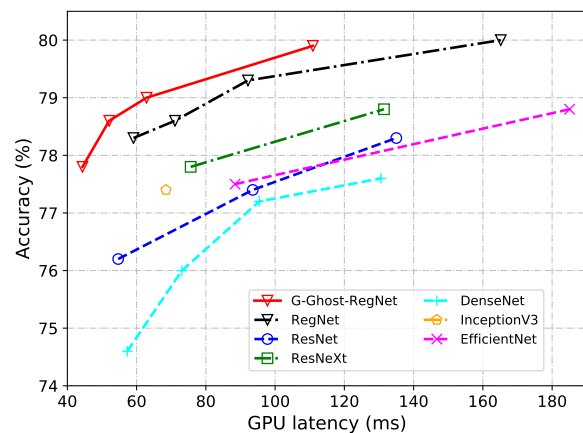
5.3.4 RegNet

RegNet is a series of networks derived by exploring the structure aspect of network design (Radosavovic et al., 2020), which is more efficient than ResNet. We test our G-Ghost stage by replacing all the stages with more than two blocks in RegNetX with our G-Ghost stage. We show the results in Table 17. From Table 17, we can see that our G-Ghost stage achieves 1.1% higher top-1 accuracy than RegNetX-3.2GF-0.75× with similar inference latency.

Table 17 RegNet Results on ImageNet. Latency is tested on an NVIDIA Tesla V100 GPU with batch size of 64.

Model	Params (M)	FLOPs (B)	Acts (M)	Latency (ms)	Top1 (%)	Top5 (%)
RegNetX-3.2GF	15.3	3.2	11.4	59.1	78.2	94.0
RegNetX-3.2GF-0.75×	8.8	1.8	8.6	46.1	76.3	93.0
G-Ghost w/o mix	9.7	1.8	8.7	43.9	76.8	93.2
G-Ghost w/ mix	10.5	1.8	8.7	44.4	77.8	93.7

We also compare G-Ghost-RegNet with the other modern CNN structures, such as RegNet (Radosavovic et al., 2020), ResNet (He et al., 2016), ResNeXt (Xie et al., 2017), DenseNet (Huang et al., 2017), InceptionV3 (Szegedy et al., 2016) and EfficientNet (Tan and Le, 2019). EfficientNet is trained under the same setting as in (Radosavovic et al., 2020) for fair comparison. The detailed results are shown in Table 18. Our G-Ghost-RegNet achieves the optimal accuracy-FLOPs trade-off and accuracy-latency trade-off (Figure 14). Note that although EfficientNet has a good trade-off between accuracy and FLOPs, it requires a large GPU latency, which is consistent with our analysis in the introduction.

Fig. 14 Accuracy *v.s.* GPU latency of G-Ghost-RegNet and other representative models.

Table 18 GhostX-RegNet Results on ImageNet. Latency is tested on an NVIDIA Tesla V100 GPU with batch size of 64.

Model	Params (M)	FLOPs (B)	Acts (M)	GPU latency (ms)	Top-1 (%)	Top-5 (%)
ResNet-50 (He et al., 2016)	25.6	4.1	11.1	54.7	76.2	92.9
ResNet-101 (He et al., 2016)	44.6	7.8	16.2	93.5	77.4	93.6
ResNet-152 (He et al., 2016)	60.2	11.5	22.6	135.0	78.3	94.1
Densenet-121 (Huang et al., 2017)	8.0	2.8	6.9	57.3	74.6	92.2
Densenet-169 (Huang et al., 2017)	14.2	3.4	7.3	73.1	76.0	93.0
Densenet-201 (Huang et al., 2017)	20.0	4.3	7.9	95.4	77.2	93.6
Densenet-161 (Huang et al., 2017)	28.7	7.7	11.1	130.6	77.6	93.8
ResNeXt-50-32x4d (Xie et al., 2017)	25.0	4.0	14.4	75.6	77.8	93.7
ResNeXt-101-32x4d (Xie et al., 2017)	42.3	8.0	21.2	131.4	78.8	94.3
InceptionV3 (Szegedy et al., 2016)	26.0	5.7	4.6	68.5	77.4	93.6
EfficientNet-B3 (Tan and Le, 2019)	12.0	1.8	23.8	88.4	77.5	-
EfficientNet-B4 (Tan and Le, 2019)	19.0	4.2	48.5	185.1	78.8	-
EfficientNet-B5 (Tan and Le, 2019)	30.0	9.9	98.9	380.6	78.5	-
RegNetX-1.6GF (Radosavovic et al., 2020)	9.2	1.6	7.9	40.8	77.0	-
RegNetX-3.2GF (Radosavovic et al., 2020)	15.3	3.2	11.4	59.1	78.3	-
RegNetX-4.0GF (Radosavovic et al., 2020)	22.1	4.0	12.2	71.1	78.6	-
RegNetX-8.0GF (Radosavovic et al., 2020)	39.6	8.0	14.1	92.2	79.3	-
RegNetX-16GF (Radosavovic et al., 2020)	54.3	15.9	25.5	165.2	80.0	-
GhostX-RegNetX-3.2GF	10.5	1.8	8.7	**44.4**	**77.8**	93.9
GhostX-RegNetX-4.0GF	15.7	2.2	9.5	**52.1**	**78.6**	94.3
GhostX-RegNetX-8.0GF	24.6	4.2	10.4	**62.9**	**79.0**	94.5
GhostX-RegNetX-16GF	32.4	9.0	19.5	**110.9**	**79.9**	95.1

Generalization to Object Detection. We choose the typical one-stage detector RetinaNet (Lin et al., 2017b) as basic framework, and use the proposed G-Ghost stage as a drop-in replacement for the backbone feature extractor. All the models are trained with $1\times$ learning rate schedule (12 epochs). Table 19 shows the results on COCO minival. Our G-Ghost stage can increase the GPU speed from 21.9 FPS to 25.9 FPS with a small mAP drop. In another hand, G-Ghost-RegNetX-3.2GF suppresses ResNet50 and RegNetX-3.2GF-0.75× by a significant margin, and meanwhile achieves a faster inference speed, which demonstrates the effectiveness and generalization ability of our G-Ghost backbone.

Table 19 Evaluation results based on RetinaNet on COCO minival. FLOPs denotes the FLOPs of backbone at 224×224 input size. The inference speed is tested on an NVIDIA Tesla V100 GPU.

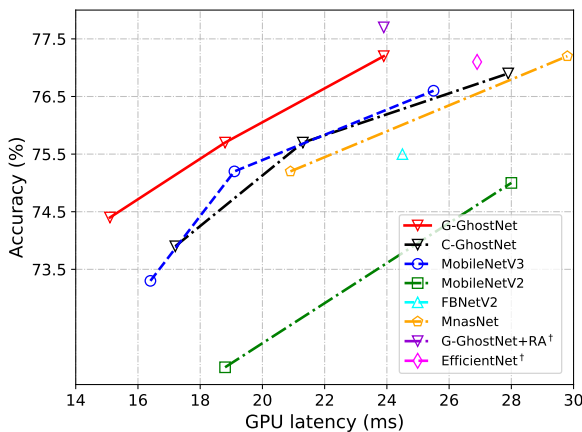| Backbone	mAP	AP_S	AP_M	AP_L	FLOPs	FPS
RegNetX-3.2GF	39.1	22.6	43.5	50.8	3.2B	21.9
ResNet50	36.8	20.5	40.6	48.5	4.1B	22.4
ResNeXt50	37.6	21.1	41.4	49.4	4.2B	19.6
RegNetX-3.2GF-0.75×	37.7	21.2	41.6	49.5	1.8B	25.2
G-Ghost-RegNetX-3.2GF	**38.4**	22.3	42.3	50.1	1.8B	**25.9**

5.3.5 Mobile Networks

Most of the existing lightweight models are designed with the aim to reduce FLOPs and latency on CPUs, but neglect the ef-

ficiency on GPUs. Our G-GhostNet remould the C-GhostNet for improve the GPU speed. We train G-GhostNet with the similar training setting as C-GhostNet and other mobile networks. The comparison of G-GhostNet and other state-of-the-art lightweight models including MobileNets (Howard et al., 2019; Sandler et al., 2018), FBNetV2 (Wan et al., 2020) and EfficientNet (Tan and Le, 2019), is shown in Figure 15. Although G-GhostNet has larger FLOPs and higher CPU latency, it achieves higher accuracy than the other models with much lower latency on GPUs. By adding RandAugment (Cubuk et al., 2020) in our model, G-GhostNet outperforms EfficientNet-B0 which also uses advanced autoaugment strategy by 0.6%, with a faster inference on the GPU.

6 Conclusion

In this paper, we present GhostNet series to accelerate the inference of convolutional neural networks on heterogeneous devices including CPU and GPU. First, we present a novel C-Ghost module for building CPU-efficient neural architectures. The basic C-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 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 com-

Fig. 15 Accuracy *v.s.* GPU latency of G-GhostNet and other representative models. GPU Latency is tested on an NVIDIA Tesla V100 GPU with batch size of 64. [†]Advanced auto-augment is applied.

pact ones while remaining the comparable performance. In addition, the C-GhostNet built using the proposed new module outperforms state-of-the-art portable neural architectures, in terms of efficiency and accuracy.

To improve the GPU efficiency of convolutional neural networks, we further introduce a new G-Ghost stage. In addition to reducing FLOPs, GPU-efficient CNNs should also concern about run-time memory and avoid trivial computation. G-Ghost stage investigates the feature redundancy cross layers in convolutional structures. The output features in a stage are split into two parts: intrinsic features and ghost features. The intrinsic features are produced with original stage with fewer channels while the ghost features are generated using GPU-efficient cheap operations from the intrinsic ones. The information inherited in intermediate intrinsic features is additively aggregated into the ghost features to enhance the representation capacity. The proposed G-Ghost stage can be used to simplify most of the mainstream CNNs. The experiments on various networks and datasets have demonstrated the effectiveness of G-Ghost stage on GPUs.

References

Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X (2015) TensorFlow: Large-scale machine learning on heterogeneous systems. URL https://www.tensorflow.org/, software available from tensorflow.org

Cai H, Zhu L, Han S (2019) Proxylessnas: Direct neural architecture search on target task and hardware. In: ICLR

Chen H, Wang Y, Xu C, Yang Z, Liu C, Shi B, Xu C, Xu C, Tian Q (2019a) Data-free learning of student networks. In: ICCV

Chen H, Wang Y, Xu C, Shi B, Xu C, Tian Q, Xu C (2020a) Addernet: Do we really need multiplications in deep learning? In: CVPR

Chen K, Wang J, Pang J, Cao Y, Xiong Y, Li X, Sun S, Feng W, Liu Z, Xu J, Zhang Z, Cheng D, Zhu C, Cheng T, Zhao Q, Li B, Lu X, Zhu R, Wu Y, Dai J, Wang J, Shi J, Ouyang W, Loy CC, Lin D (2019b) MMDetection: Open mmlab detection toolbox and benchmark. arXiv preprint arXiv:190607155

Chen LC, Papandreou G, Kokkinos I, Murphy K, Yuille AL (2016) Semantic image segmentation with deep convolutional nets and fully connected crfs. In: ICLR

Chen W, Xie D, Zhang Y, Pu S (2019c) All you need is a few shifts: Designing efficient convolutional neural networks for image classification. In: CVPR

Chen W, Gong X, Liu X, Zhang Q, Li Y, Wang Z (2020b) Fasterseg: Searching for faster real-time semantic segmentation. In: ICLR

Chin TW, Ding R, Zhang C, Marculescu D (2020) Towards efficient model compression via learned global ranking. In: CVPR

Chollet F (2017) Xception: Deep learning with depthwise separable convolutions. In: CVPR, pp 1251–1258

Cubuk ED, Zoph B, Shlens J, Le QV (2020) Randaugment: Practical automated data augmentation with a reduced search space. In: CVPR Workshops

Deng J, Dong W, Socher R, Li LJ, Li K, Fei-Fei L (2009) Imagenet: A large-scale hierarchical image database. In: CVPR, Ieee, pp 248–255

Denton EL, Zaremba W, Bruna J, LeCun Y, Fergus R (2014) Exploiting linear structure within convolutional networks for efficient evaluation. In: NeurIPS, pp 1269–1277

Forrest NI, Song H, Matthew W, Khalid A, Dally JW (2017) Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. In: ICLR

Gholami A, Kwon K, Wu B, Tai Z, Yue X, Jin P, Zhao S, Keutzer K (2018) Squeezenext: Hardware-aware neural network design. In: CVPR Workshops

Gong X, Chang S, Jiang Y, Wang Z (2019) Autogan: Neural architecture search for generative adversarial networks. In: ICCV

Gui S, Wang HN, Yang H, Yu C, Wang Z, Liu J (2019) Model compression with adversarial robustness: A unified optimization framework. In: NeurIPS

Guo J, Han K, Wang Y, Wu H, Chen X, Xu C, Xu C (2021) Distilling object detectors via decoupled features. In: CVPR

Han B, Yao Q, Yu X, Niu G, Xu M, Hu W, Tsang I, Sugiyama M (2018a) Co-teaching: Robust training of deep neural networks with extremely noisy labels. In: NeurIPS

Han K, Guo J, Zhang C, Zhu M (2018b) Attribute-aware attention model for fine-grained representation learning. In: ACM MM

Han K, Wang Y, Tian Q, Guo J, Xu C, Xu C (2020a) Ghostnet: More features from cheap operations. In: CVPR

Han K, Wang Y, Xu Y, Xu C, Wu E, Xu C (2020b) Training binary neural networks through learning with noisy supervision. In: ICML

Han K, Wang Y, Xu C, Xu C, Wu E, Tao D (2021) Learning versatile convolution filters for efficient visual recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence

Han S, Pool J, Tran J, Dally W (2015) Learning both weights and connections for efficient neural network. In: NeurIPS, pp 1135–1143

Han S, Mao H, Dally WJ (2016) Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In: ICLR

He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: CVPR

He Y, Zhang X, Sun J (2017) Channel pruning for accelerating very deep neural networks. In: ICCV

He Y, Kang G, Dong X, Fu Y, Yang Y (2018a) Soft filter pruning for accelerating deep convolutional neural networks. In: IJCAI

He Y, Lin J, Liu Z, Wang H, Li LJ, Han S (2018b) Amc: Automl for model compression and acceleration on mobile devices. In: ECCV

He Y, Liu P, Wang Z, Hu Z, Yang Y (2019) Filter pruning via geometric median for deep convolutional neural networks acceleration. In: CVPR

He Y, Ding Y, Liu P, Zhu L, Zhang H, Yang Y (2020) Learning filter pruning criteria for deep convolutional neural networks acceleration. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 2009–2018

Hinton G, Vinyals O, Dean J (2015) Distilling the knowledge in a neural network. arXiv preprint arXiv:150302531

Howard A, Sandler M, Chu G, Chen LC, Chen B, Tan M, Wang W, Zhu Y, Pang R, Vasudevan V, et al. (2019) Searching for mobilenetv3. In: ICCV

Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, Andreetto M, Adam H (2017) Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:170404861

Hu J, Shen L, Sun G (2018) Squeeze-and-excitation networks. In: CVPR

Huang G, Liu Z, Van Der Maaten L, Weinberger KQ (2017) Densely connected convolutional networks. In: CVPR

Huang Z, Wang N (2018) Data-driven sparse structure selection for deep neural networks. In: ECCV, pp 304–320

Hubara I, Courbariaux M, Soudry D, El-Yaniv R, Bengio Y (2016) Binarized neural networks. In: NeurIPS, pp 4107–4115

Ioffe S, Szegedy C (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: ICML

Jacob B, Kligys S, Chen B, Zhu M, Tang M, Howard A, Adam H, Kalenichenko D (2018) Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: CVPR

Jaderberg M, Vedaldi A, Zisserman A (2014) Speeding up convolutional neural networks with low rank expansions. In: BMVC

Jeon Y, Kim J (2018) Constructing fast network through deconstruction of convolution. In: NeurIPS

Krizhevsky A, Hinton G (2009) Learning multiple layers of features from tiny images. Tech. rep., Citeseer

Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: NeurIPS, pp 1097–1105

Li H, Kadav A, Durdanovic I, Samet H, Graf HP (2017) Pruning filters for efficient convnets. In: ICLR

Liebenwein L, Baykal C, Lang H, Feldman D, Rus D (2020) Provable filter pruning for efficient neural networks. In: ICLR

Lin M, Ji R, Wang Y, Zhang Y, Zhang B, Tian Y, Shao L (2020a) Hrank: Filter pruning using high-rank feature map. In: CVPR

Lin M, Ji R, Zhang Y, Zhang B, Wu Y, Tian Y (2020b) Channel pruning via automatic structure search. In: IJCAI

Lin S, Ji R, Yan C, Zhang B, Cao L, Ye Q, Huang F, Doermann D (2019) Towards optimal structured cnn pruning via generative adversarial learning. In: CVPR

Lin TY, Maire M, Belongie S, Hays J, Perona P, Ramanan D, Dollár P, Zitnick CL (2014) Microsoft coco: Common objects in context. In: ECCV, Springer

Lin TY, Dollár P, Girshick R, He K, Hariharan B, Belongie S (2017a) Feature pyramid networks for object detection. In: CVPR

Lin TY, Goyal P, Girshick R, He K, Dollár P (2017b) Focal loss for dense object detection. In: ICCV

Liu C, Wang Y, Han K, Xu C, Xu C (2019a) Learning instance-wise sparsity for accelerating deep models. In: IJCAI

Liu Z, Wu B, Luo W, Yang X, Liu W, Cheng KT (2018) Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In: ECCV

Liu Z, Mu H, Zhang X, Guo Z, Yang X, Cheng TKT, Sun J (2019b) Metapruning: Meta learning for automatic neural network channel pruning. In: ICCV

Liu Z, Sun M, Zhou T, Huang G, Darrell T (2019c) Rethinking the value of network pruning. In: ICLR

Luo JH, Wu J, Lin W (2017) Thinet: A filter level pruning method for deep neural network compression. In: ICCV, pp 5058–5066

Ma N, Zhang X, Zheng HT, Sun J (2018) Shufflenet v2: Practical guidelines for efficient cnn architecture design. In: ECCV

Molchanov P, Mallya A, Tyree S, Frosio I, Kautz J (2019) Importance estimation for neural network pruning. In: CVPR

Ning X, Zhao T, Li W, Lei P, Wang Y, Yang H (2020) Dsa: More efficient budgeted pruning via differentiable sparsity allocation. In: ECCV

Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, et al. (2019) Pytorch: An imperative style, high-performance deep learning library. In: NeurIPS

Radosavovic I, Kosaraju RP, Girshick R, He K, Dollár P (2020) Designing network design spaces. In: CVPR

Rastegari M, Ordonez V, Redmon J, Farhadi A (2016) Xnor-net: Imagenet classification using binary convolutional neural networks. In: ECCV, Springer, pp 525–542

Ren S, He K, Girshick R, Sun J (2015) Faster R-CNN: Towards real-time object detection with region proposal networks. In: NeurIPS

Sandler M, Howard A, Zhu M, Zhmoginov A, Chen LC (2018) Mobilenetv2: Inverted residuals and linear bottlenecks. In: CVPR, pp 4510–4520

Shen M, Han K, Xu C, Wang Y (2019) Searching for accurate binary neural architectures. In: ICCV Workshops

Simonyan K, Zisserman A (2015) Very deep convolutional networks for large-scale image recognition. In: ICLR

Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2015) Going deeper with convolutions. In: CVPR

Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z (2016) Rethinking the inception architecture for computer vision. In: CVPR

Tan M, Le Q (2019) Efficientnet: Rethinking model scaling for convolutional neural networks. In: ICML

Tan M, Chen B, Pang R, Vasudevan V, Sandler M, Howard A, Le QV (2019) Mnasnet: Platform-aware neural architecture search for mobile. In: CVPR, pp 2820–2828

Wan A, Dai X, Zhang P, He Z, Tian Y, Xie S, Wu B, Yu M, Xu T, Chen K, et al. (2020) Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In: CVPR

Wang Y, Xu C, You S, Tao D, Xu C (2016) Cnnpack: packing convolutional neural networks in the frequency domain. In: NeurIPS, pp 253–261

Wang Y, Xu C, XU C, Xu C, Tao D (2018) Learning versatile filters for efficient convolutional neural networks. In: NeurIPS

Wang Y, Jiang Z, Chen X, Xu P, Zhao Y, Lin Y, Wang Z (2019) E2-train: Training state-of-the-art cnns with over 80% energy savings. In: NeurIPS

Wen W, Wu C, Wang Y, Chen Y, Li H (2016) Learning structured sparsity in deep neural networks. In: NeurIPS, pp 2074–2082

Williams S, Waterman A, Patterson D (2009) Roofline: an insightful visual performance model for multicore architectures. Communications of the ACM 52(4):65–76

Wilson RC, Hancock ER, Smith WAP (2016) Wide residual networks. In: BMVC

Wu B, Wan A, Yue X, Jin P, Zhao S, Golmant N, Gholaminejad A, Gonzalez J, Keutzer K (2018) Shift: A zero flop, zero parameter alternative to spatial convolutions. In: CVPR

Wu B, Dai X, Zhang P, Wang Y, Sun F, Wu Y, Tian Y, Vajda P, Jia Y, Keutzer K (2019) Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In: CVPR, pp 10734–10742

Xie S, Girshick R, Dollár P, Tu Z, He K (2017) Aggregated residual transformations for deep neural networks. In: CVPR, pp 1492–1500

Xu Y, Wang Y, Chen H, Han K, Chunjing X, Tao D, Xu C (2019) Positive-unlabeled compression on the cloud. In: NeurIPS

Yang L, Jiang H, Cai R, Wang Y, Song S, Huang G, Tian Q (2021) Condensenet v2: Sparse feature reactivation for deep networks. In: CVPR

Yang Z, Wang Y, Liu C, Chen H, Xu C, Shi B, Xu C, Xu C (2019) Legonet: Efficient convolutional neural networks with lego filters. In: ICML

Yang Z, Wang Y, Chen X, Shi B, Xu C, Xu C, Tian Q, Xu C (2020a) Cars: Continuous evolution for efficient neural architecture search. In: CVPR

Yang Z, Wang Y, Han K, Xu C, Xu C, Tao D, Xu C (2020b) Searching for low-bit weights in quantized neural networks. In: NeurIPS

You S, Xu C, Xu C, Tao D (2017) Learning from multiple teacher networks. In: SIGKDD

Yu J, Yang L, Xu N, Yang J, Huang T (2019) Slimmable neural networks. In: ICLR, URL https://openreview.net/forum?id=H1gMCsAqY7

Yu R, Li A, Chen CF, Lai JH, Morariu VI, Han X, Gao M, Lin CY, Davis LS (2018) Nisp: Pruning networks using neuron importance score propagation. In: CVPR

Zagoruyko S (2015) 92.45 on cifar-10 in torch, 2015. URL http://torchch/blog/2015/07/30/cifar html

Zhang X, Zhou X, Lin M, Sun J (2018) Shufflenet: An extremely efficient convolutional neural network for mobile devices. CVPR

Zhou D, Hou Q, Chen Y, Feng J, Yan S (2020) Rethinking bottleneck structure for efficient mobile network design. In: ECCV

Zoph B, Vasudevan V, Shlens J, Le QV (2018) Learning transferable architectures for scalable image recognition. In: CVPR, pp 8697–8710