Research Article

Research on Efficient Deep Learning Algorithm Based on ShuffleGhost in the Field of Virtual Reality

Bangtong Huang,1,2 Hongquan Zhang,3 Zihong Chen,4 Lingling Li,5 and Lihua Shi1

1School of Management, Shanghai University of Engineering Science, China
2Key Lab of Information Network Security, The Third Institute of Ministry of Public Security, China
3Department of Computer Science and Engineering, The Chinese University of Hong Kong, China
4College of Engineering, Shantou University, China
5Department of Central Laboratory, Children’s Hospital of Shanghai Jiao Tong University, China

Correspondence should be addressed to Hongquan Zhang; 03200001@sues.edu.cn

Received 24 May 2021; Revised 23 June 2021; Accepted 26 July 2021; Published 15 August 2021

1. Introduction

Deep learning has achieved great performance in computer vision and natural language processing tasks. The SOTA often makes use of a large-scale backbone, and ResNet-50 [1] has about 25.6 M parameters and requires 4.1B FLOPs to process an image of size $224 \times 224$. On the one hand, massive trainable parameters enhance the performance of these deep networks. On the other hand, the large-scale backbone faces a risk of costing much resources of memory and computation, which means it would be difficult to replant it to mobiles or cars with efficient and acceptable performance.

Over the years, a series of methods have been proposed to investigate compact deep neural networks such as network pruning [2], low-bit quantization [3], and knowledge distillation [4]. Ren and Lee [5] recently proposed using deep networks to learn high-level visual feature representations and synthetic images to train neural networks. Li et al. [6] used deep neural network and hash mapping to extract remote sensing image features with complex background. Since the feature dimensionality of the direct output of the classification network is high, the storage space is occupied with the efficiency reducing. Long et al. [7] constructed a complete convolutional network, which combines the semantic information of
the deep coarse layer with the appearance information of the shallow fine layer to generate an accurate and detailed segmentation set. At the same time, object recognition is one of the important contents of image recognition, and the number of object recognition will change according to different images, such as face recognition [8] and visual search engine [9].

Therefore, simplifying the output feature of the classification network is the key to improving the retrieval efficiency. However, performances of these methods are often upper bounded by pretrained deep neural networks that have been taken as their baselines. Besides them, efficient network design with fewer parameters and calculations have achieved considerable success such as MobileNet [10]. Thus, this paper focuses on how to reduce the trainable parameters and improve the efficiency of model training. We did a lot of experiments and tried to combine the advantages of several models like ShuffleNet [11], SENet, and GhostNet and found an architecture named ShuffleGhost, which has a better performance than the original GhostNet. Liu et al. [12, 13] proposed that edge computing method is used to solve the problems of supply chain network optimization, and good performance is obtained.

2. Related Models

2.1. GhostNet. Han et al. [14] found the abundant and even redundant information in the feature maps of well-trained deep neural networks. They came up with a method called Ghost module to generate the abundant information in a cost-efficient way. It reduces the trainable parameters and training time.

GhostNet contains two key parts: ghost convolution part and primary convolution part. And the feature maps produced from ghost convolution part donated as Ghost Feature Maps, as well as the feature maps produced from primary convolution part donated as Primary Feature Maps.

In GhostNet, researchers designed a module named Ghost module to do convolution, as shown in Figure 1. The key idea of GhostNet is to use one-half channels of feature maps to do convolution in primary convolution part, which shown as the green regions. The orange region in Figure 1, which is the ghost convolution part, adopted a ghost convolution to produce Ghost Features. The Ghost Features and the result of primary convolution are concatenated together as the output of the Ghost module. In the implementation of GhostNet, researcher used C/2 number of 3×3 kernels to do group convolution, with the number of the group is C/2, that is, the channel of each kernel is 1, and each kernel corresponds to one feature map. So the computation N_{CC} needed of this group convolution is much less than that of primary convolution N_{PC}:

\[ N_{CC} = (w - k_w + 1) \times (h - k_h + 1) \times k_w \times k_h \times \frac{C}{2}, \]
\[ N_{PC} = (w - k_w + 1) \times (h - k_h + 1) \times k_w \times k_h \times \frac{C}{2} \times \frac{C}{2}, \]

where \( w, h, \) and \( C \) are the width, height, and channel of the feature map, respectively, and \( k_w \) and \( k_h \) are the width and height of kernels, respectively. In primary convolution module and ghost convolution module, the channels of input feature map and output feature map are all equal to C/2. It can be found that the ghost convolution module could produce C/2 Ghost Features with a low computation cost, which is C/2 lower than that of primary convolution module. Since the channels of feature maps from primary convolution and ghost convolution are all equal to C/2, after concatenating, the channel of Ghost module output is equal to the channel of input feature map.

The Ghost bottleneck is designed as an architecture like the block of ResNet, so that Ghost bottleneck could be used to build a deeper convolution network, as shown in Figure 2. Generally, each block in ResNet contains two or three convolutions and one shortcut, while in GhostNet, each Ghost bottleneck only contains two Ghost modules and one shortcut.

However, the GhostNet mainly has four drawbacks:

(i) The Ghost Features are always placed at the end of the feature maps, which would introduce noise if the Ghost Features are used to predict new Ghost Features

(ii) Convolution in Ghost module and shortcut have high computation cost

(iii) When 1×1 kernel of convolution and one-padding are configured in the first convolution of Ghost module at the same time, the one-padding field is used to do 1×1 kernel of convolution and extract features which are meaningless and would introduce noise

(iv) It is only downsampled by the large kernel. This drawback would make it hard to design GhostNet into other backbone

2.2. ShuffleNet. ShuffleNet has two key parts: shuffle layer and group convolution. These two components are used together to do convolution. As for group convolution, it could use sparse kernels to replace the original one, which allows it to have a partial dense field to extract features. And for shuffle layer, it is always placed after group convolution to shuffle the channels of feature maps together. ShuffleNet achieves great success in reducing the parameters and ensures the performance of the model in the meantime.

2.2.1. Shuffle Layer. As mentioned in the related paper of ShuffleNet, it was found that some state-of-the-art networks like Xception [15] use group convolution instead of doing 1×1 convolutions, namely, pointwise convolutions. If group convolution is applied, the channels would be divided into several parts, and group convolution would be conducted on them. However, the correlation among these feature maps from different channels would be block. In order to tackle this problem, ShuffleNet introduces shuffle layer to redistribute the feature maps, and it achieves a good performance.
2.2.2. Group Convolution. Group convolution used sparse kernels to do convolution, and it can reduce the parameters and FLOPs of convolution. And Figure 3 shows the architecture of group convolution. As we known, if we do original convolution (stride = 1, no padding) on figure maps with shape $1 \times C \times w \times h$ with $k$ number of kernels with shape $k_w, k_h, C$, it would take computation $N_{CON}$:

$$N_{CON} = (w - k_w + 1) \times (h - k_h + 1) \times k_w \times k_h \times C \times k.$$  \hspace{1cm} (2)

And it can be found that the computation of group convolution reduces significantly. If $g$ groups and $k'$ number of kernels are used, it would follow this ReLU.

$$C = g \times k'.$$  \hspace{1cm} (3)

And the computation of group convolution $N_{GC}$ is as follows:

$$N_{GC} = (w - k_w + 1) \times (h - k_h + 1) \times k_w \times k_h \times k' \times g.$$ \hspace{1cm} (4)

It can be found that the computation of $N_{CON}$ is $k$ times larger than that of $N_{GC}$. In fact, in group convolution, the zero field of kernels also joins the computation, and the advantage of group convolution might be less obvious.

2.2.3. SENet. Hu et al. [16] investigated the relationship between channels of network design and introduced a new architectural SE block unit, which allows the network to perform feature recalibration and learn to use the global information. Only a small amount of computation is needed to improve the performance.

For traditional CNN, the core computation is convolution operator, which learns new feature maps from input one by convolution kernel. A large part of existing works is improving the receptive field, that is, combining more features in space or extracting multiscale spatial information, while the SENet is more concentrating on the relationship between different channels and expecting that the model can automatically learn the importance of different channel features. To achieve this goal, SENet proposed the squeeze-and-excitation block [17–19].

In SE block, first stage is squeeze. The squeeze operation is performed on the feature map obtained by convolution to
get the global information. Namely, it shrinks feature maps through spatial dimensions. Then, in the second stage, the excitation, it learns the relationship between channels and gets the weight of different channels. Finally, multiply the original feature map to get the final feature [20–22].

1. Squeeze

Convolution is only operated in a local space, which means it is difficult for the output feature map to obtain enough information to extract the relationship between channels. This problem becomes more serious in the front layer of the network due to the relatively small receptive field. The essence of squeeze operation is to encode the whole spatial feature of a channel into a global feature, which is implemented by global average pooling. The following equation shows the computation of squeeze [23, 24].

\[
Z_C = F_{sq}(u_C) = \frac{1}{h \times w} \sum_{i=1}^{h} \sum_{j=1}^{w} u_C(i, j),
\]

where \( h \) and \( w \) are the height and width of feature maps, respectively, and \( u_C \) is the input of feature maps.

2. Excitation

When we get the global feature, then we need another operation to learn the relationship between channels, which is the excitation. The excitation should meet the two criteria: firstly, it should be flexible. And it should be able to learn the nonlinear relationship between channels. Secondly, the learning relationship is not mutually exclusive because it allows multiple channel features instead of one hot form. Based on the criteria, there is the equation [25, 26].

\[
S = F_{ex}(z, W) = \sigma(g(z, W)) = \sigma(W_2 \delta(W_1 z)),
\]

where \( \delta \) is the ReLU function, \( W_1 \in \mathbb{R}^{C \times r} \) and \( W_2 \in \mathbb{R}^{C \times r} \) are the trainable parameters, and \( z \) is the input feature maps.

In order to reduce the complexity of the model and improve the generalization ability, a bottleneck structure with two FC layers is adopted. The first FC layer is used to reduce the dimension. The final FC layer restores the original dimension.

3. Methods

In the paper, we proposed a model named ShuffleGhost and compared with a GhostNet-like network to discuss the performance. In this section, we would first introduce the GhostNet-like network we used as baseline, as illustrated in Figure 3. The GhostNet-like module, we still adopted Ghost bottleneck with GhostNet-like module. And we designed a GhostNet-like network we used as baseline to compare the performance with the ShuffleGhost. The GhostNet-like network maintained most features from GhostNet but a 1 × 1 convolution operation in Ghost module was replaced with 3 × 3 convolution, and we could have a fair discussion and comparison among them.

3.1. GhostNet-Like Network

The method ShuffleGhost we proposed would utilize shuffle operation and group convolution to mix the information from the Ghost Feature Maps and Primary Feature Maps. In order to evaluate the performance of ShuffleGhost fairly, we designed a Ghost-like module as illustrated in Figure 3. In the Ghost-like module, we used 3 × 3 convolution to replace the 1 × 1 convolution in primary convolution in Ghost module. This modification would not influence the principle of GhostNet, as we remained the primary convolution and ghost convolution. Apart from Ghost-like module, we still adopted Ghost bottleneck with GhostNet-like module. And we designed a GhostNet-like network as baseline, as illustrated in Figure 4.

3.2. ShuffleGhost

The GhostNet employs ghost convolution and primary convolution to design Ghost module, the computation of Ghost Feature Map would be cheap since the Ghost module adopted 3 × 3 group convolutions, and the number of groups equal to the channels, which means that the channel of all kernels is equal to one. In fact, they are a series of linear transforms, and in GhostNet, these linear transforms are assigned to 3 × 3 convolution on each intrinsic feature. In this paper, we tried other linear transforms. We
found the Ghost Feature Maps from ghost convolution and the Primary Feature Maps from primary convolution are concatenated together, and the Ghost Feature Maps are always placed at a fixed position. In the next Ghost module, it introduces noise if the primary convolution is replaced with group convolution, especially if the group number is set to two. In this situation, all the Ghost Feature Maps would be used to produce half of the feature maps, and the real feature maps produce the other, and these two kinds of feature maps share the equal importance.

Apart from the ghost convolution to produce Ghost Feature, traditional convolution is adopted in Ghost module and shortcut in Ghost bottleneck, and this paper finds it possible to optimize them.

3.2.1. ShuffleGhost Module. This module is designed based on the GhostNet-like module. Since the computation of the first convolution in GhostNet-like module would take high cost of computations, ShuffleGhost replaced this first convolution with group convolution. But as mentioned above, group convolution in GhostNet-like module would reduce the robustness of the model due to the concatenate operation and cascade of Ghost module.

Comparing with GhostNet-like module and ignoring the trainable parameter reduction from the group convolution, ghost convolution mainly makes difference. In ghost convolution part, if ShuffleGhost follows the rule of GhostNet-like module with the kernel size set to $3 \times 3$, then the trainable parameters and the computation cost of ShuffleGhost would be much higher. One possible way to solve it is replacing the $3 \times 3$ kernels in ghost convolution part with $1 \times 1$ kernels. Formally, the modification is to swap the places of two kinds of kernels, but in ShuffleGhost module, the expensive kernel, which is the $3 \times 3$ size one, produces the feature maps in primary convolution, and these feature maps are also utilized to produce Ghost Features. However, the feature maps produced by expensive kernel are only used in ghost convolution, as well as the Primary Feature Maps produced from ghost kernel, which are lack of precision and are utilized to conduct primary convolution and ghost convolution. So, ShuffleGhost module has larger capacity and more precision than Ghost module. Figure 4 shows the structure of ShuffleGhost module.

3.2.2. ShuffleGhost. In ShuffleGhost, we would introduce versions of bottlenecks, which use Ghost module or ShuffleGhost module to form themselves, so that this bottleneck could have some methods like residual block in ResNet. As mentioned before and illustrated in Figure 1, the Ghost module consists of two Ghost modules and one shortcut and adopts $3 \times 3$ convolution operation in shortcut part which would suffer from high computational cost. Meanwhile, group convolution in ShuffleGhost module would introduce noise because group would separate the information of Ghost Feature Maps from ghost convolution and Primary Feature Maps apart. To tackle this problem, one of direct ways is to mix the feature maps which concatenate from Ghost Feature Maps and Primary Feature Maps. Thus, each group of feature maps would contain the information from Ghost Feature Maps and Primary Feature Maps simultaneously.

(1) ShuffleGhostV1

In order to mix the channels of the primary feature maps and ghost feature maps, the shuffle layer is placed at the top of the ShuffleGhost BottleneckV1. Besides, 16 group convolutions with $3 \times 3$ kernels are configured after the shuffle layer to make combination and extract features from both Ghost Feature Maps and Primary Feature Maps. The ShuffleGhost BottleneckV1 contains two Ghost-like modules, which have the same architecture with the baseline, Ghost-like network. And in the first Ghost-like module, the padding could be set to zero or one, which depends on whether this bottleneck is going to do downsampling or not. The padding of the second Ghost-like module should be one, so that the feature map would not lose too much information and ensure the resolution of the feature map.

Since the bottleneck is modified, the shortcut should also be modified to ensure the dimension of the data from the second Ghost-like module that has the same shape as that from shortcut. In shortcut, the padding configure in the first convolution should be the same as the configure of the first
Ghost-like module in ShuffleGhost BottleneckV1, but this module also used group convolution in shortcut to lighten the computation.

ShuffleGhostV1 used ShuffleGhost BottleneckV1 and Ghost-like module and has the same layer configuration with Ghost-like network which is illustrated in Figure 5. It is found that ShuffleGhostV1 does not introduce more parameters than Ghost-like network although it used group convolution and shuffling in ShuffleGhost BottleneckV1. We placed the $3 \times 3$ group convolution in shortcut layer instead of $3 \times 3$ convolution, and the parameters we saved in shortcut layers are used to form group convolution in ShuffleGhost BottleneckV1.

(2) ShuffleGhostV2

ShuffleGhostV2 still used ShuffleGhost BottleneckV1 architecture to form a network, but ShuffleGhost module is used instead of Ghost-like module. The output of ShuffleGhost module shares the same shape with Ghost-like module, and the shortcut and bottleneck do not need modification.

(3) ShuffleGhostV3

In the bottleneck of ShuffleGhostV2, we adopted ShuffleGhost module and the same bottleneck structure as Shuffle-GhostV1. We found in bottleneck of ShuffleGhostV2, the kernel size in group convolution after the shuffle layer is $3 \times 3$, and it would suffer computation cost although it could integrate the information from shuffle layer across the channels and improve performance. In order to reduce the computation of this group convolution, we utilize SENet layer from SENet to design ShuffleGhost BottleneckV2 and build ShuffleGhostV3.

SENet layer used two steps to extract feature maps: squeeze and excitation. The first step is to squeeze the origin feature maps to a small-scale feature maps, and the second step is finding the importance of among these scale feature maps. Finally, the feature maps from excitation multiply with origin feature maps. In ShuffleGhost, SENet layer could have a better performance than group convolution. On the one hand, SENet costs smaller computation. On the other hand, SENet could evaluate the importance of the feature maps which concatenated from Ghost Feature Maps and Primary Feature Maps and combine them together. The SENet layer and the structure of ShuffleGhostV3 are illustrated in Figure 6.

SENet layer is adopted to take the place of group convolution, and Figure 6(b) shows the detailed structure of SENet layer.
| Layers              | Output size | Ghost-like network          | ShuffleGhostV1               | ShuffleGhostV2               | ShuffleGhostV3               |
|---------------------|-------------|------------------------------|------------------------------|------------------------------|------------------------------|
| conv                | 28 × 28     | 3 × 3 conv, stride 1         | 3 × 3 conv, stride 1         | 3 × 3 conv, stride 1         | 3 × 3 conv, stride 1         |
| BatchNorm, ReL.U    | 28 × 28     |                              |                              |                              |                              |
| Bottleneck          | 26 × 26     | Ghost bottleneck (module × 2) | ShuffleGhost BottleneckV1    | ShuffleGhost BottleneckV1    | ShuffleGhost BottleneckV1    |
|                     |             | (Ghost-like module × 2)      | (ShuffleGhost module × 2)    | (ShuffleGhost module × 2)    | (ShuffleGhost module × 2)    |
| Bottleneck          | 24 × 24     | Ghost bottleneck (module × 2) | ShuffleGhost BottleneckV1    | ShuffleGhost BottleneckV1    | ShuffleGhost BottleneckV1    |
|                     |             | (Ghost-like module × 2)      | (ShuffleGhost module × 2)    | (ShuffleGhost module × 2)    | (ShuffleGhost module × 2)    |
| Bottleneck          | 22 × 22     | Ghost bottleneck (module × 2) | ShuffleGhost BottleneckV1    | ShuffleGhost BottleneckV1    | ShuffleGhost BottleneckV1    |
|                     |             | (Ghost-like module × 2)      | (ShuffleGhost module × 2)    | (ShuffleGhost module × 2)    | (ShuffleGhost module × 2)    |
| Bottleneck          | 20 × 20     | Ghost bottleneck (module × 2) | ShuffleGhost BottleneckV1    | ShuffleGhost BottleneckV1    | ShuffleGhost BottleneckV1    |
|                     |             | (Ghost-like module × 2)      | (ShuffleGhost module × 2)    | (ShuffleGhost module × 2)    | (ShuffleGhost module × 2)    |
| Bottleneck          | 18 × 18     | Ghost bottleneck (module × 2) | ShuffleGhost BottleneckV1    | ShuffleGhost BottleneckV1    | ShuffleGhost BottleneckV1    |
|                     |             | (Ghost-like module × 2)      | (ShuffleGhost module × 2)    | (ShuffleGhost module × 2)    | (ShuffleGhost module × 2)    |
| Bottleneck × 3      | 16 × 16     | Ghost bottleneck (module × 2) | ShuffleGhost BottleneckV1    | ShuffleGhost BottleneckV1    | ShuffleGhost BottleneckV1    |
|                     |             | (Ghost-like module × 2)      | (ShuffleGhost module × 2)    | (ShuffleGhost module × 2)    | (ShuffleGhost module × 2)    |
| Bottleneck          | 14 × 14     | Ghost bottleneck (module × 2) | ShuffleGhost BottleneckV1    | ShuffleGhost BottleneckV1    | ShuffleGhost BottleneckV1    |
|                     |             | (Ghost-like module × 2)      | (ShuffleGhost module × 2)    | (ShuffleGhost module × 2)    | (ShuffleGhost module × 2)    |
| Bottleneck × 3      | 12 × 12     | Ghost bottleneck (module × 2) | ShuffleGhost BottleneckV1    | ShuffleGhost BottleneckV1    | ShuffleGhost BottleneckV1    |
|                     |             | (Ghost-like module × 2)      | (ShuffleGhost module × 2)    | (ShuffleGhost module × 2)    | (ShuffleGhost module × 2)    |
| conv                | 12 × 12     | 1 × 1 conv, stride 1         | 1 × 1 conv, stride 1         | 1 × 1 conv, stride 1         | 1 × 1 conv, stride 1         |
| BatchNorm, ReL.U    | 12 × 12     |                              |                              |                              |                              |
| Pooling             | 4 × 4       | Global average pooling       | Global average pooling       | Global average pooling       | Global average pooling       |
| Dense               |             | 10D fully connected, softmax | 10D fully connected, softmax | 10D fully connected, softmax | 10D fully connected, softmax |
In summary, the three visions of ShuffleGhost have the architectures and parameters as illustrated in Table 1. They have the same layer configuration with Ghost-like network.

## 4. Experiment

In tiny ResNet experiment, we used the bottleneck in ShuffleGhost or GhostNet-like network to replace the residual block and trained without pretrained models. This experiment focuses on comparing and discussing the appearance of bottleneck. The second experiment is VGG16-like [27] experiment. We used the bottleneck and the module in ShuffleGhost or GhostNet-like network to replace the convolution operations in VGG16. The strategy is that if two convolutions are cascaded, then we used one bottleneck to replace them; otherwise, we used one module to replace the convolution operation. We conducted the experiments on CIFAR-10 datasets and compared the performance on trainable parameters’ effectiveness.

### 4.1. Tiny ResNet Experiment

The experiment focuses on finding the difference between the GhostNet-like network and ShuffleGhost.

| Layers             | Output size | Ghost-like network            | Parameters |
|--------------------|-------------|-------------------------------|------------|
| Convolution        | 28 x 28     | 3 x 3 conv, stride 1          | 216        |
| BatchNorm, ReLU    | 28 x 28     |                               | 6          |
| Ghost bottleneck   | 26 x 26     | Ghost-like module x 2         | 3604       |
| Ghost bottleneck   | 24 x 24     | Ghost-like module x 2         | 13800      |
| Ghost bottleneck   | 22 x 22     | Ghost-like module x 2         | 34680      |
| Ghost bottleneck   | 20 x 20     | Ghost-like module x 2         | 61214      |
| Ghost bottleneck   | 18 x 18     | Ghost-like module x 2         | 116432     |
| Ghost bottleneck x 3| 16 x 16     | Ghost-like module x 2         | 218960, 174880, 174880 |
| Ghost bottleneck   | 14 x 14     | Ghost-like module x 2         | 276592     |
| Ghost bottleneck x 3| 12 x 12     | Ghost-like module x 2         | 718288, 695360, 695360 |
| Convolution        | 12 x 12     | 1 x 1 conv, stride 1         | 4800       |
| BatchNorm, ReLU    | 12 x 12     |                               | 960        |
| Pooling            | 4 x 4       | Global average pooling       | 0          |
| Dense              |             | 10D fully connected, softmax | 76810      |

| Structure          | Trainable parameters | Param size |
|--------------------|----------------------|------------|
| GhostNet-like      | 3,271,594            | 12.48      |
| ShuffleGhostV1     | 3,102,970 (-5.1%)    | 11.84      |
| ShuffleGhostV2     | 3,087,802 (-5.6%)    | 11.78      |
| ShuffleGhostV3     | 3,028,266 (-7.4%)    | 11.55      |

**Figure 7:** The structures of VGG16-like experiment.

**Table 2:** The structure of Ghost-like network for tiny ResNet experiment.

| Layers             | Output size | Ghost-like network            | Parameters |
|--------------------|-------------|-------------------------------|------------|
| Convolution        | 28 x 28     | 3 x 3 conv, stride 1          | 216        |
| BatchNorm, ReLU    | 28 x 28     |                               | 6          |
| Ghost bottleneck   | 26 x 26     | Ghost-like module x 2         | 3604       |
| Ghost bottleneck   | 24 x 24     | Ghost-like module x 2         | 13800      |
| Ghost bottleneck   | 22 x 22     | Ghost-like module x 2         | 34680      |
| Ghost bottleneck   | 20 x 20     | Ghost-like module x 2         | 61214      |
| Ghost bottleneck   | 18 x 18     | Ghost-like module x 2         | 116432     |
| Ghost bottleneck x 3| 16 x 16     | Ghost-like module x 2         | 218960, 174880, 174880 |
| Ghost bottleneck   | 14 x 14     | Ghost-like module x 2         | 276592     |
| Ghost bottleneck x 3| 12 x 12     | Ghost-like module x 2         | 718288, 695360, 695360 |
| Convolution        | 12 x 12     | 1 x 1 conv, stride 1         | 4800       |
| BatchNorm, ReLU    | 12 x 12     |                               | 960        |
| Pooling            | 4 x 4       | Global average pooling       | 0          |
| Dense              |             | 10D fully connected, softmax | 76810      |

**Table 3:** The comparison of trainable parameters and size of parameters of GhostNet-like network and three versions of ShuffleGhost.
Firstly, we build a tiny ResNet, whose structure illustrated as Table 2, with the original GhostNet as baseline to judge the performance of ShuffleGhost. Then, we implement ShuffleGhost in 3 versions to train on CIFAR-10 and compare the results with original GhostNet. We trained these models with $10^{-3}$ learning rate, 16 batch size, and 50 maximum epochs and employed Adam as the optimizer.

4.2. VGG16-Like Experiment. In a brief, for Ghost module, we use it to replace every convolution layer in VGG16. For Ghost bottleneck, since bottleneck is made up of 2 consistent convolution layers, we replace every 2 convolution layers by bottleneck and replace the left single one conv layer by Ghost module. The structure is like that of Figure 6. We use the same strategy for ShuffleGhost, and the structures of VGG16 experiment are shown in Figure 7.

Based on what we implemented, we have 6 structures in total. We will train them on the CIFAR-10 datasets and set epoch to be 50 and learning rate to be 0.01. Despite the fact that we focus on the portable devices’ efficiency, which can be regarded as CPU, we would also try the GPU training too and prove we make some progress in both sides.

4.3. Result. Tables 3 and 4 described the models from 2 parts, the trainable parameters and accuracy. As shown in the 2 tables, firstly, for the model size, we can see the baseline GhostNet has around the size of 3.27 M trainable parameters while ShuffleGhost has smaller model size in both 3 versions. These 3 models decrease the model size more than 5% (-5.1%, -5.6%, and -7.4% separately), and the most outstanding model is the ShuffleGhost version 3; the size drops from 3.27 M to 3.02 M, and the FLOP drops from 989.25 M to 895.79 M; the FLOP drops around 9.4%; this could lead to a better performance of embedded devices in both efficiency and portability. Then, for the accuracy part, which is essential, we cannot sacrifice the accuracy to have a small model. The ShuffleGhostV2 has around 86.6% accuracy, which is 1% higher than the baseline GhostNet-like network. So, from this experiment, we can see that in the peer comparison, ShuffleGhost could perform better in both model size part and accuracy part.

After comparing the GhostNet-like network and Shuffle-Ghost, we will go deeper to see the performance of the different modules and bottlenecks. As mentioned before, we set the original VGG16 as the baseline model and redesigned VGG16-like models for experiment. In Table 4, it shows the parameters after replacing the convolution layers in VGG16 network; it is known that GhostNet-like network and all the ShuffleGhost decrease the model size in a significant level (at least 40%). As for both ShuffleGhost module and Shuffle-Ghost bottleneck, they perform better than the original ones. Combining the accuracy in Figure 8 and the training time in Table 4, the advantages and disadvantages of applying modules and bottlenecks can be known from the data. Namely, for the model size part, ShuffleGhost module can bring us smaller model size than ShuffleGhost bottleneck, but the cost is worse accuracy. As for the accuracy consideration, Shuffle-Ghost bottleneck can guarantee a better accuracy, while the trade-off will be size. For the efficiency part, all the Ghost and ShuffleGhost run faster than the original one on GPU. But on GPU, only module can perform better; bottlenecks will be slower than the original one. In accuracy trend from Figure 9, all the models get to the best result in almost the same epochs.

In VGG16-like experiment, Ghost module and Shuffle-Ghost module form an abstract comparison, that is, both modules used their module to take the place of the convolution in original VGG. And Ghost bottleneck forms the other abstract comparison, and each of them uses its bottleneck to replace two convolution layers in VGG.
5. Conclusion

In the tiny ResNet experiment, Tables 3 and 5 compare the original GhostNet and ShuffleGhost, and we can see that ShuffleGhost not only decreases the trainable parameters and FLOP, but also has a slightly better accuracy. In the VGG16-like experiment, Table 4 compares different modules and bottlenecks, and we can see that for the module, ShuffleGhost module has smaller trainable parameters and higher training speed both on CPU and GPU. But the speed to get the convergence of accuracy is a little slower. And for the bottleneck, both ShuffleGhost bottlenecks V2 and V3 have smaller trainable parameters than original Ghost. For the CPU training speed, they are similar, but for the GPU training speed, ShuffleGhost will be slower. As for the accuracy part in both experiments, as can be seen in Figure 8, the blue color one is the original VGG, which has the best performance, and all the other models have around 89.9% accuracy, 1% lower than the best. One thing needs to be noticed is that in ShuffleGhost BottleneckV3, the brown one with 90.2% accuracy has approximately the best accuracy result. Another thing we can notice in Figure 9 is that all the different modules and bottlenecks take nearly the same epoch to get to the best accuracy.

As discussed above, from the tiny ResNet experiment, we know that ShuffleGhost has smaller trainable parameters and a little higher accuracy compared to the original GhostNet-like network. From VGG16-like experiment, we can clearly see that the ShuffleGhost holds approximately the same accuracy result compared to the original one. But it has smaller trainable parameters and FLOPs and can be faster than the original one. All in all, we learn from GhostNet and design new ShuffleGhosts, which making use of the redundancy in feature maps properly and combined with shuffle to solve the too consistent problem in feature map produced by original GhostNet. Besides, we also implement different versions to tackle the problems we found in the original Ghost, which are already being introduced before.

Data Availability

The data used to support the findings of this study are available from the author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

The finding is sponsored by the Key Lab of Information Network Security of Ministry of Public Security (Grant no. C20609) and Municipal Key Curriculum Construction Project of University in Shanghai (Grant no. S202003002).
References

[1] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778, Las Vegas, NV, USA, 2016.

[2] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag, “What is the state of neural network pruning,” 2020, https://arxiv.org/abs/2003.03033.

[3] Y. Choukroun, E. Kravchik, and P. Kisilev, “Low-bit quantization of neural networks for efficient inference,” in 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pp. 3009–3018, Seoul, Korea, 2019.

[4] J. Gou, B. Yu, S. J. Maybank, and D. Tao, “Knowledge distillation: a survey,” International Journal of Computer Vision, vol. 129, no. 6, pp. 1789–1819, 2021.

[5] Z. Ren and Y. J. Lee, “Cross-domain self supervised multi-task feature learning using synthetic imagery,” in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 762–771, Salt Lake City, UT, USA, 2018.

[7] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3431–3440, Boston, Massachusetts, USA, 2015.

[8] Y. Li, Y. Zhang, X. Huang, H. Zhu, and J. Ma, “Large-scale remote sensing image retrieval by deep hashing neural networks,” IEEE Transactions on Geoscience and Remote Sensing, vol. 56, no. 2, pp. 950–965, 2018.

[9] Y. Zhang, P. Pan, Y. Zheng et al., “Visual search at Alibaba,” in Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 993–1001, London, United Kingdom, 2018.

[10] A. G. Howard, M. Zhu, B. Chen et al., “Mobilenets: efficient convolutional neural networks for mobile vision applications,” 2017, https://arxiv.org/abs/1704.04861.

[11] X. Zhang, X. Zhou, M. Lin, and J. Sun, “Shufflenet: an extremely efficient convolutional neural network for mobile devices,” in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 6848–6856, Salt Lake City, UT, USA, 2018.

[12] Z. Liu, L. L. Lang, B. Hu, L. H. Shi, B. T. Huang, and Y. J. Zhao, “Emission reduction decision of agricultural supply chain considering carbon tax and investment cooperation,” Journal of Cleaner Production, vol. 294, 2021.

[13] Z. Liu, B. Hu, Y. J. Zhao et al., “Research on intelligent decision of low carbon supply chain based on carbon tax constraints in human-driven edge computing,” IEEE ACCESS, vol. 8, pp. 48264–48273, 2020.

[14] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, “Ghostnet: more features from ghost operations,” in 2020 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1580–1589, Seattle, Washington, USA, 2020.

[15] F. Chollet, “Xception: deep learning with depthwise separable convolutions,” in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1251–1258, Honolulu, Hawaii, USA, 2017.

[16] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, “Squeeze-and-excitation networks,” in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 7132–7141, Honolulu, Hawaii, USA, 2017.

[17] H. Shen, M. Zhang, H. Wang, F. Guo, and W. Susilo, “A cloud-aided privacy-preserving multi-dimensional data comparison protocol,” Information Sciences, vol. 545, no. 2, pp. 739–752, 2021.

[18] Z. Zhang, M. Liu, M. Zhou, and J. Chen, “Dynamic reliability analysis of nonlinear structures using a Duffing-system- based equivalent nonlinear system method,” International Journal of Approximate Reasoning, vol. 126, no. 11, pp. 84–97, 2020.

[19] W. Bo, Z. B. Fang, L. X. Wei, Z. F. Cheng, and Z. X. Hua, “Malicious URLs detection based on a novel optimization algorithm,” IEICE Transactions on Information and Systems, vol. E104.D, no. 4, pp. 513–516, 2021.

[20] X. Xue, Y. Li, X. Yin, C. Shang, T. Peng, and Q. Shen, “Affine transformation-enhanced multifactorial optimization for heterogeneous problems,” IEEE Transactions on Cybernetics, pp. 1–12, 2020.

[21] H. Sheng, S. Wang, Y. Zhang et al., “Near-online tracking with co-occurrence constraints in blockchain-based edge computing,” IEEE Internet of Things Journal, vol. 8, no. 4, pp. 2193–2207, 2021.

[22] Z. Wu, J. Cao, Y. Wang, Y. Wang, L. Zhang, and J. Wu, “HPSD: a hybrid PU-learning-based spambot monitoring model for product reviews,” IEEE Transactions on Cybernetics, vol. 50, no. 4, pp. 1595–1606, 2020.

[23] N. Xiao, R. Xinyi, Z. Xiong et al., “A diversity-based selfish node detection algorithm for socially aware networking,” Journal of Signal Processing Systems, vol. 93, no. 7, pp. 811–825, 2021.

[24] Z. Lv, D. Chen, R. Lou, and H. Song, “Industrial security solution for virtual reality,” IEEE Internet of Things Journal, vol. 8, no. 8, pp. 6273–6281, 2021.

[25] M. A. Medvedeva, T. E. Simos, and C. Tsitouras, “Variable step-size implementation of sixth-order Numerov-type methods,” Mathematical Methods in the Applied Sciences, vol. 43, no. 3, pp. 1204–1215, 2020.

[26] C. C. Hou, T. E. Simos, and I. T. Famelis, “Neural network solution of pantograph type differential equations,” Mathematical Methods in the Applied Sciences, vol. 43, no. 6, pp. 3369–3374, 2020.

[27] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2014, https://arxiv.org/abs/1409.1556.