GhostNet- More Features from Cheap Operation

1

Abstract

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

1

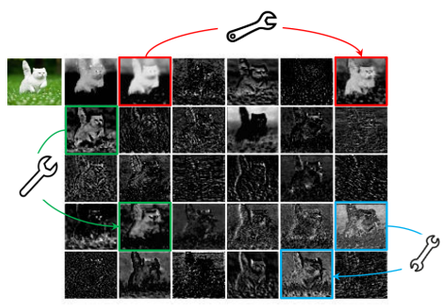


Figure 1. Visualization of some feature maps generated by the first residual group in ResNet-50, where three similar feature map pair examples are annotated with boxes of the same color. One feature map in the pair can be approximately obtained by transforming the other one through cheap operations (denoted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

2

Figure 1 presents some feature maps of an input image generated by ResNet-50, and there exist many similar pairs of feature maps, like a ghost of each another. Redundancy in feature maps could be an important characteristic for a successful deep neural network. Instead of avoiding the redundant feature maps, we tend to embrace them, but in a cost-efficient way.

2

2.1. Model Compression

For a given neural network, model compression aims to reduce the computation, energy and storage cost [14, 48, 11, 54]. Pruning connections [15, 14, 50] cuts out the unimportant connections between neurons. Channel pruning [51, 18, 31, 39, 59, 23, 35] further targets on removing useless channels for easier acceleration in practice. Model quantization [42, 24, 26] represents weights or activations

2

binarization methods [24, 42, 38, 45] with only 1-bit values can extremely accelerate the model by efficient binary operations

2

Tensor decomposition [27, 9] reduces the parameters or computation by exploiting the redundancy and low-rank property in weights

2

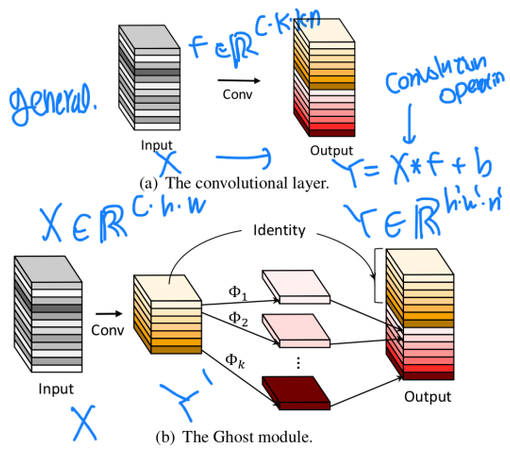
Knowledge distillation [19, 12, 3] utilizes larger models to teach smaller ones, which improves the performance of smaller models.

2

2.2. Compact Model Design

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

3



(a) The convolutional layer.

(b) The Ghost module.

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

3

of FLOPs and parameters. Suppose that the output feature maps are "ghosts" of a handful of intrinsic feature maps with some cheap transformations. These intrinsic feature maps are often of smaller size and produced by ordinary convolution filters. Specifically, m intrinsic feature maps $Y' \in \mathbb{R}^{h' \times w' \times m}$ are generated using a primary convolution:

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

where $f' \in \mathbb{R}^{c \times k \times k \times m}$ is the utilized filters, $m \leq n$ and the bias term is omitted for simplicity. The hyper-parameters

3

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

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

4

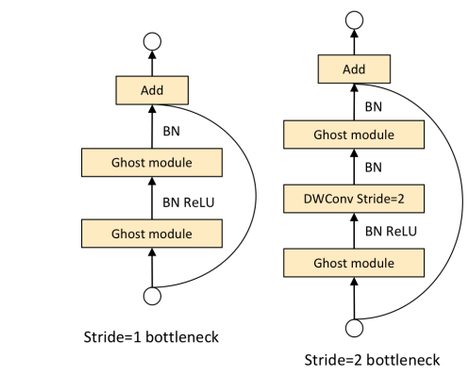


Figure 3. Ghost bottleneck. Left: Ghost bottleneck with stride=1; right: Ghost bottleneck with stride=2.

6

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

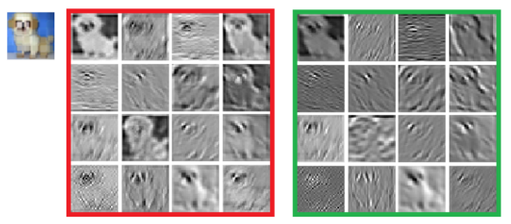


Figure 4. The feature maps in the 2nd layer of Ghost-VGG-16. The left-top image is the input, the feature maps in the left red box are from the primary convolution, and the feature maps in the right green box are after the depthwise transformation.

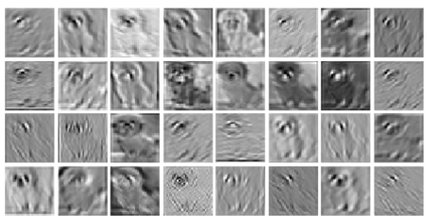


Figure 5. The feature maps in the 2nd layer of vanilla VGG-16.