# ChannelNets: Compact and Efficient Convolutional Neural Networks via Channel-Wise Convolutions

Hongyang Gao, Zhengyang Wang, Lei Cai, and Shuiwang Ji, *Senior Member, IEEE*

**Abstract**—Convolutional neural networks (CNNs) have shown great capability of solving various artificial intelligence tasks. However, the increasing model size has raised challenges in employing them in resource-limited applications. In this work, we propose to compress deep models by using channel-wise convolutions, which replace dense connections among feature maps with sparse ones in CNNs. Based on this novel operation, we build light-weight CNNs known as ChannelNets. ChannelNets use three instances of channel-wise convolutions; namely group channel-wise convolutions, depth-wise separable channel-wise convolutions, and the convolutional classification layer. Compared to prior CNNs designed for mobile devices, ChannelNets achieve a significant reduction in terms of the number of parameters and computational cost without loss in accuracy. Notably, our work represents an attempt to compress the fully-connected classification layer, which usually accounts for about $25\%$ of total parameters in compact CNNs. Along this new direction, we investigate the behavior of our proposed convolutional classification layer and conduct detailed analysis. Based on our in-depth analysis, we further propose convolutional classification layers without weight-sharing. This new classification layer achieves a good trade-off between fully-connected classification layers and the convolutional classification layer. Experimental results on the ImageNet dataset demonstrate that ChannelNets achieve consistently better performance compared to prior methods.

**Index Terms**—Deep learning, group convolution, channel-wise convolution, convolutional classification, model compression.

## 1 Introduction

Convolutional neural networks (CNNs) have demonstrated great capability of solving visual recognition tasks. Since AlexNet [1] achieved remarkable success on the ImageNet Challenge [2], various deeper and more complicated networks [3], [4], [5], [6], [7], [8] have been proposed to set the performance records. However, the higher accuracy usually comes with an increasing amount of parameters and computational cost. For example, the VGG16 [3] has $128$ million parameters and requires $15,300$ million floating point operations (FLOPs) to classify an image. In many real-world applications, predictions need to be performed on resource-limited platforms such as sensors and mobile phones, thereby requiring compact models with higher speed. Model compression aims at exploring a trade-off between accuracy and efficiency.

Recently, significant progress has been made on model compression [9], [10], [11], [12]. The strategies for building compact and efficient CNNs can be divided into two categories; those are, compressing pre-trained networks or designing new compact architectures that are trained from scratch. Studies in the former category were mostly based on traditional compression techniques such as product quantization [11], [13], [14], pruning [15], [16], [17], hashing [18], [19], Huffman coding [20], and factorization [21], [22], [23].

- *Hongyang Gao, Zhengyang Wang, and Shuiwang Ji are with the Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843.*
  *E-mail: {hongyang.gao, zhengyang.wang, sji}@tamu.edu*
- *Lei Cai is with the School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99164.*
  *E-mail: lei.cai@wsu.edu*

*Manuscript received March 04, 2019.*

The second category has already been explored before model compression. Inspired by the Network-In-Network architecture [24], GoogLeNet [4] included the Inception module to build deeper networks without increasing model sizes and computational cost. Through factorizing convolutions, the Inception module was further improved by [25]. The depth-wise separable convolution, proposed in [26], generalized the factorization idea [27] and decomposed the convolution [28], [29] into a depth-wise convolution and a $1 \times 1$ convolution. The operation has been shown to be able to achieve competitive results with fewer parameters. In terms of model compression, MobileNets [12] and ShuffleNets [30] designed CNNs for mobile devices by employing depth-wise separable convolutions.

In this work, we focus on the second category and build a new family of light-weight CNNs known as ChannelNets. By observing that the fully-connected pattern accounts for most parameters in CNNs, we propose channel-wise convolutions, which are used to replace dense connections among feature maps with sparse ones. Early work like LeNet-5 [31] has shown that sparsely-connected networks work well when resources are limited. To apply channel-wise convolutions in model compression, we develop group channel-wise convolutions, depth-wise separable channel-wise convolutions, and the convolutional classification layer. They are used to compress different parts of CNNs, leading to our ChannelNets. ChannelNets achieve a better trade-off between efficiency and accuracy than prior compact CNNs, as demonstrated by experimental results on the ImageNet ILSVRC 2012 dataset. It is worth noting that ChannelNets are the first models that attempt to compress the fully-connected classification layer, which accounts for about 25%

of total parameters in compact CNNs.

Based on the conference version of this work [32], we continue to explore the compression of the fully-connected classification layer. We analyze the behavior of the convolutional classification layer by looking into the major differences between it and the fully-connected classification layer. We conduct additional experiments to obtain further insights and propose the convolutional classification layer without weight-sharing. This new classification layer involves the same computation as the original convolutional classification layer but leads to improved compact CNNs in terms of performance and stability. While the convolutional classification layer without weight-sharing introduces slightly more training parameters than the convolutional classification layer, the advantages brought by it include improved performance and robustness to class orders. We conduct experiments to evaluate our proposed methods.

## 2 Background and Motivations

The trainable layers of CNNs [5], [33], [34] are commonly composed of convolutional layers and fully-connected layers. Most prior studies, such as MobileNets [12] and ShuffleNets [30], focused on compressing convolutional layers, where most parameters and computation lie. To make the discussion concrete, suppose a 2-D convolutional operation takes $m$ feature maps with a spatial size of $d_f \times d_f$ as inputs, and outputs $n$ feature maps of the same spatial size with appropriate padding. $m$ and $n$ are also known as the number of input and output channels, respectively. The convolutional kernel size is $d_k \times d_k$ and the stride is set to 1. Here, without loss of generality, we use square feature maps and convolutional kernels for simplicity. We further assume that there is no bias term in the convolutional operation, as modern CNNs employ the batch normalization [35] with a bias after the convolution. In this case, the number of parameters in the convolution is $d_k \times d_k \times m \times n$ and the computational cost in terms of FLOPs is $d_k \times d_k \times m \times n \times d_f \times d_f$. Since the convolutional kernel is shared for each spatial location, for any pair of input and output feature maps, the connections are sparse and weighted by $d_k \times d_k$ shared parameters. However, the connections among channels follow a fully-connected pattern, *i.e.*, all $m$ input channels are connected to all $n$ output channels, which results in the $m \times n$ term. For deep convolutional layers, $m$ and $n$ are usually large numbers like 512 and 1024 [3], [36], thus $m \times n$ is usually very large.

Based on the above insights, one way to reduce the size and cost of convolutions is to circumvent the multiplication between $d_k \times d_k$ and $m \times n$. MobileNets [12] applied this approach to explore compact deep models for mobile devices. The core operation employed in MobileNets is the depth-wise separable convolution [37], which consists of a depth-wise convolution and a $1 \times 1$ convolution, as illustrated in Figure 1(a). The depth-wise convolution applies a single convolutional kernel independently for each input feature map, thus generating the same number of output channels. The following $1 \times 1$ convolution is used to fuse the information of all output channels using a linear combination. The depth-wise separable convolution actually decomposes the regular convolution into a depth-wise convolution step and a channel-wise fuse step. Through this decomposition, the number of parameters becomes

$$d_k \times d_k \times m + m \times n, \tag{1}$$

and the computational cost becomes

$$d_k \times d_k \times m \times d_f \times d_f + m \times n \times d_f \times d_f. \tag{2}$$

In both equations, the first term corresponds to the depth-wise convolution and the second term corresponds to the $1 \times 1$ convolution. By decoupling $d_k \times d_k$ and $m \times n$, the amounts of parameters and computations are reduced.

While MobileNets successfully employed depth-wise separable convolutions to perform model compression and achieve competitive results, it is noted that the $m \times n$ term still dominates the number of parameters in the models. As pointed out in [12], $1 \times 1$ convolutions, which lead to the $m \times n$ term, account for 74.59% of total parameters in MobileNets. The analysis of regular convolutions reveals that $m \times n$ comes from the fully-connected pattern, which is also the case in $1 \times 1$ convolutions. To understand this, first consider the special case where $d_f = 1$. Now the inputs are $m$ units as each feature map has only one unit. As the convolutional kernel size is $1 \times 1$, which does not change the spatial size of feature maps, the outputs are also $n$ units. It is clear that the operation between the $m$ input units and the $n$ output units is a fully-connected operation with $m \times n$ parameters. When $d_f > 1$, the fully-connected operation is shared for each spatial location, leading to the $1 \times 1$ convolution. Hence, the $1 \times 1$ convolution actually outputs a linear combination of input feature maps. More importantly, in terms of connections between input and output channels, both the regular convolution and the depth-wise separable convolution follow the fully-connected pattern.

As a result, a better strategy to compress convolutions is to change the dense connection pattern between input and output channels. Based on the depth-wise separable convolution, it is equivalent to circumventing the $1 \times 1$ convolution. A simple method, previously used in AlexNet [1], is the group convolution [38], [39], [40], [41], [42]. Specifically, the $m$ input channels are divided into $g$ mutually exclusive groups. Each group goes through a $1 \times 1$ convolution independently and produces $n/g$ output feature maps. It follows that there are still $n$ output channels in total. For simplicity, suppose both $m$ and $n$ are divisible by $g$. As the $1 \times 1$ convolution for each group requires $1/g^2$ parameters and FLOPs, the total amount after grouping is only $1/g$ as compared to the original $1 \times 1$ convolution. Figure 1(b) describes a $1 \times 1$ group convolution with 2 groups.

However, the grouping operation usually compromises performance because there is no interaction among groups. As a result, information of feature maps in different groups is not combined, as opposed to the original $1 \times 1$ convolution that combines information of all input channels. To address this limitation, ShuffleNet [30] was proposed, where a shuffling layer was employed after the $1 \times 1$ group convolution. Through random permutation, the shuffling layer partly achieves interactions among groups. But any output group accesses only $m/g$ input feature maps and thus collects partial information. Due to this reason, ShuffleNet had to employ a deeper architecture than MobileNets to achieve competitive results.

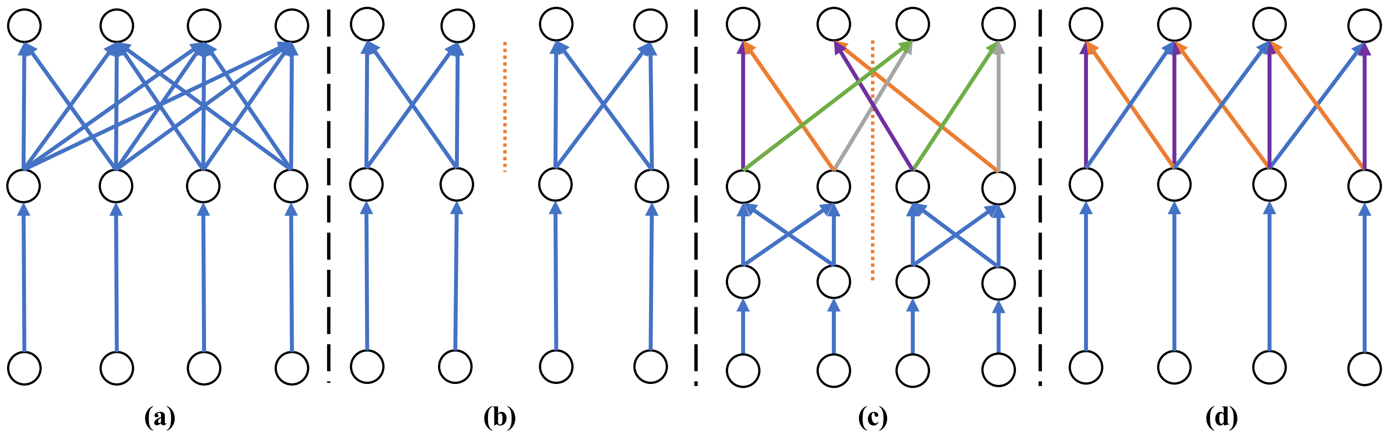


Fig. 1. Illustrations of different compact convolutions. Part (a) shows the depth-wise separable convolution, which is composed of a depth-wise convolution and a $1 \times 1$ convolution. Part (b) shows the case where the $1 \times 1$ convolution is replaced by a $1 \times 1$ group convolution. Part (c) illustrates the use of the proposed group channel-wise convolution for information fusion. Part (d) shows the proposed depth-wise separable channel-wise convolution, which consists of a depth-wise convolution and a channel-wise convolution. For channel-wise convolutions in (c) and (d), the same color represents shared weights.

## 3 Channel-Wise Convolutions and ChannelNets

In this work, we propose channel-wise convolutions in Section 3.1, based on which we build our ChannelNets. In Section 3.2, we apply group channel-wise convolutions to address the information inconsistency problem caused by grouping. Afterwards, we generalize our method in Section 3.3, which leads to a direct replacement of depth-wise separable convolutions in deeper layers. Through analysis of the generalized method, we propose a convolutional classification layer to replace the fully-connected output layer in Section 3.4, which further reduces the amounts of parameters and computations. Finally, Section 3.5 introduces the architecture of our ChannelNets. In Sections 3.6 and 3.7, we investigate the behavior of the convolutional classification layer in detail and propose the convolutional classification layer without weight-sharing as a good trade-off between the fully-connected classification layer and the convolutional classification layer.

### 3.1 Channel-Wise Convolutions

We begin with the definition of channel-wise convolutions in general. As discussed above, the $1 \times 1$ convolution is equivalent to using a shared fully-connected operation to scan every $d_f \times d_f$ locations of input feature maps. A channel-wise convolution employs a shared 1-D convolutional operation, instead of the fully-connected operation. Consequently, the connection pattern between input and output channels becomes sparse, where each output feature map is connected to a part of input feature maps.

To be specific, we again start with the special case where $d_f = 1$. The $m$ input units (feature maps) can be considered as a 1-D feature map of size $m$. Similarly, the output becomes a 1-D feature map of size $n$. Note that both the input and output have only 1 channel. The channel-wise convolution performs a 1-D convolution with appropriate padding to map the $m$ units to the $n$ units. In the cases where $d_f > 1$, the same 1-D convolution is computed for every spatial locations. As a result, the number of parameters in a channel-wise convolution with a kernel size of $d_c$ is simply $d_c$ and the computational cost is

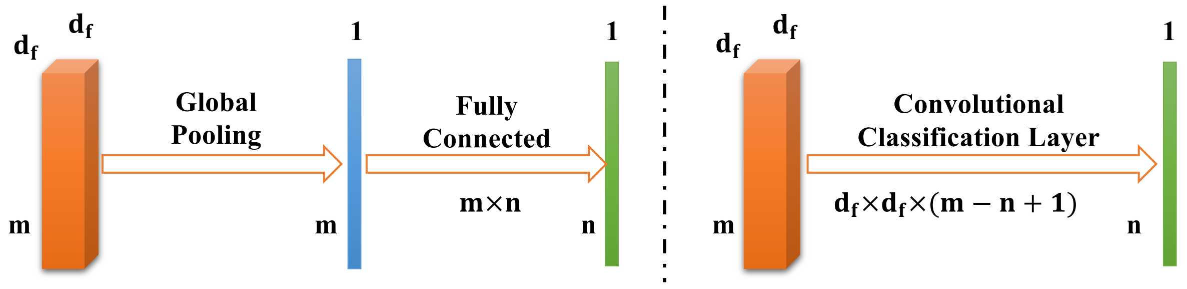
$$d_c \times n \times d_f \times d_f. \tag{3}$$

By employing sparse connections, we avoid the $m \times n$ term. Therefore, channel-wise convolutions consume a negligible amount of computations and can be performed efficiently compared to regular convolutions.

### 3.2 Group Channel-Wise Convolutions

We apply channel-wise convolutions to develop a solution to the information inconsistency problem incurred by grouping. After the $1 \times 1$ group convolution, the outputs are $g$ groups, each of which includes $n/g$ feature maps. As illustrated in Figure 1(b), the $g$ groups are computed independently from completely separate groups of input feature maps. To enable interactions among groups, an efficient information fusion layer is needed after the $1 \times 1$ group convolution. The fusion layer is expected to retain the grouping for following group convolutions while allowing each group to collect information from all the groups. Concretely, both inputs and outputs of this layer should be $n$ feature maps that are divided into $g$ groups. Meanwhile, the $n/g$ output channels in any group should be computed from all the $n$ input channels. More importantly, the layer must be compact and efficient; otherwise the advantage of grouping will be compromised.

Based on channel-wise convolutions, we propose the group channel-wise convolution, which serves elegantly as the fusion layer. Given $n$ input feature maps that are divided into $g$ groups, this operation performs $g$ independent channel-wise convolutions. Each channel-wise convolution uses a stride of $g$ and outputs $n/g$ feature maps with appropriate padding. Note that, in order to ensure all $n$ input channels are involved in the computation of any output group of channels, the kernel size of channel-wise convolutions needs to satisfy $d_c \geq g$. The desired outputs of the fusion layer is obtained by concatenating the outputs



$d_f$
$d_f$
m
Global Pooling
1
m
Fully Connected
m×n
1
n
$d_f$
$d_f$
m
Convolutional Classification Layer
$d_f \times d_f \times (m-n+1)$
1
n

Fig. 2. An illustration of the convolutional classification layer. The left part describes the original output layers, *i.e.*, a global average pooling layer and a fully-connected classification layer. The global pooling layer reduces the spatial size $d_f \times d_f$ to $1 \times 1$ while keeping the number of channels. Then the fully-connected classification layer changes the number of channels from $m$ to $n$, where $n$ is the number of classes. The right part illustrates the proposed convolutional classification layer, which performs a single 3-D convolution with a kernel size of $d_f \times d_f \times (m - n + 1)$ and no padding. The convolutional classification layer saves a significant amount of parameters and computation.

of these channel-wise convolutions. Figure 1(c) provides an example of using the group channel-wise convolution after the $1\times1$ group convolution, which replaces the original $1\times1$ convolution.

To see the efficiency of this approach, the number of parameters of the $1 \times 1$ group convolution followed by the group channel-wise convolution is

$$\frac{m}{g} \times \frac{n}{g} \times g + d_c \times g, \tag{4}$$

and the computational cost is

$$\frac{m}{g} \times \frac{n}{g} \times d_f \times d_f \times g + d_c \times \frac{n}{g} \times d_f \times d_f \times g. \tag{5}$$

Since in most cases we have $d_c \ll m$, our approach requires approximately $1/g$ training parameters and FLOPs, as compared to the second terms in Eqs. 1 and 2.

## 3.3 Depth-Wise Separable Channel-Wise Convolutions

Based on the above descriptions, it is worth noting that there is a special case where the number of groups and the number of input and output channels are equal, *i.e.*, $g = m = n$. A similar scenario resulted in the development of depth-wise convolutions [12], [37], [43]. In this case, there is only one feature map in each group. The $1 \times 1$ group convolution simply scales the convolutional kernels in the depth-wise convolution. As the batch normalization [35] in each layer already involves a scaling term, the $1 \times 1$ group convolution becomes redundant and can be removed. Meanwhile, instead of using $m$ independent channel-wise convolutions with a stride of $m$ as the fusion layer, we apply a single channel-wise convolution with a stride of 1. Due to the removal of the $1 \times 1$ group convolution, the channel-wise convolution directly follows the depth-wise convolution, resulting in the depth-wise separable channel-wise convolution, as illustrated in Figure 1(d).

In essence, the depth-wise separable channel-wise convolution replaces the $1 \times 1$ convolution in the depth-wise separable convolution with the channel-wise convolution. The connections among channels are changed directly from a dense pattern to a sparse one. As a result, the number of parameters is

$$d_k \times d_k \times m + d_c, \tag{6}$$

and the cost is

$$d_k \times d_k \times m \times d_f \times d_f + d_c \times n \times d_f \times d_f, \tag{7}$$

which saves dramatic amounts of parameters and computations. This layer can be used to directly replace the depth-wise separable convolution.

## 3.4 Convolutional Classification Layer

Most prior model compression methods pay little attention to the very last layer of CNNs, which is a fully-connected layer used to generate classification results. Taking MobileNets on the ImageNet dataset as an example, this layer uses a $1,024$-component feature vector as inputs and produces $1,000$ logits corresponding to $1,000$ classes. Therefore, the number of parameters is $1,024 \times 1,000 \approx 1$ million, which accounts for 24.33% of total parameters as reported in [12]. In this section, we explore a special application of the depth-wise separable channel-wise convolution, proposed in Section 3.3, to reduce the large amount of parameters in the classification layer.

We note that the second-to-the-last layer is usually a global average pooling layer, which reduces the spatial size of feature maps to 1. For example, in MobileNets, the global average pooling layer transforms $1,024$ $7 \times 7$ input feature maps into $1,024$ $1 \times 1$ output feature maps, corresponding to the $1,024$-component feature vector fed into the classification layer. In general, suppose the spatial size of input feature maps is $d_f \times d_f$. The global average pooling layer is equivalent to a special depth-wise convolution with a kernel size of $d_f \times d_f$, where the weights in the kernel is fixed to $1/d_f^2$. Meanwhile, the following fully-connected layer can be considered as a $1 \times 1$ convolution as the input feature vector can be viewed as $1\times1$ feature maps. Thus, the global average pooling layer followed by the fully-connected classification layer is a special depth-wise convolution followed by a $1 \times 1$ convolution, resulting in a special depth-wise separable convolution.

As the proposed depth-wise separable channel-wise convolution can directly replace the depth-wise separable convolution, we attempt to apply the replacement here. Specifically, the same special depth-wise convolution is employed, but is followed by a channel-wise convolution with a kernel size of $d_c$ whose number of output channels is equal to the number of classes. However, we observe that such an operation can be further combined using a regular 3-D convolution [44].

In particular, the $m$ $d_f \times d_f$ input feature maps can be viewed as a single 3-D feature map with a size of $d_f \times d_f \times m$.

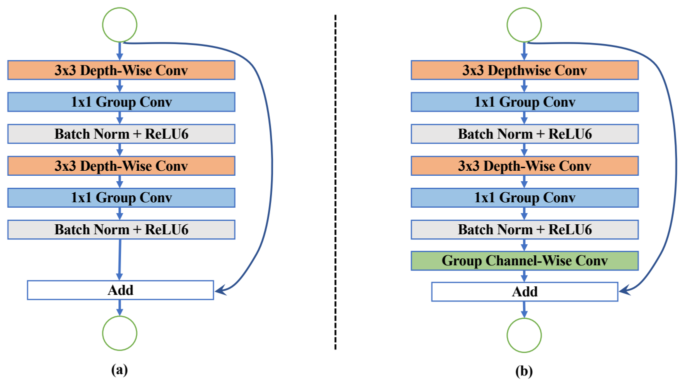


Fig. 3. Illustrations of the group module (GM) and the group channel-wise module (GCWM). Part (a) shows GM, which has two depth-wise separable convolutional layers. Note that $1 \times 1$ convolutions is replaced by $1 \times 1$ group convolutions to save computations. A skip connection is added to facilitate model training. GCWM is described in part (b). Compared to GM, it has a group channel-wise convolution to fuse information from different groups.

TABLE 1
ChannelNets Architectures. The operations are described in the format of "Type / Stride / # Output channels". "Conv" denotes the regular convolution; "DWSConv" denotes the depth-wise separable convolution; "DWSCWConv" denotes the depth-wise separable channel-wise convolution; "CCL" denotes the convolutional classification layer. The kernel size in regular convolutions and depth-wise convolutions is $3 \times 3$.

| v1 | v2 | v3 |
|---|---|---|
| Conv / 2 / 32 | | |
| DWSConv / 1 / 64 | | |
| DWSConv / 2 / 128 | | |
| DWSConv / 1 / 128 | | |
| DWSConv / 2 / 256 | | |
| DWSConv / 1 / 256 | | |
| DWSConv / 2 / 512 | | |
| GCWM / 1 / 512 | | |
| GCWM / 1 / 512 | | |
| GM / 1 / 512 | | |
| DWSConv / 2 / 1024 | | |
| DWSConv / 1 / 1024 | DWSCWConv | DWSCWConv |
| AvgPool + FC | AvgPool + FC | CCL |

The special depth-wise convolution, or equivalently the global average pooling layer, is essentially a 3-D convolution with a kernel size of $d_f \times d_f \times 1$, where the weights in the kernel is fixed to $1/d_f^2$. Moreover, in this view, the channel-wise convolution is a 3-D convolution with a kernel size of $1 \times 1 \times d_c$. These two consecutive 3-D convolutions follow a factorized pattern. As proposed in [25], a $d_k \times d_k$ convolution can be factorized into two consecutive convolutions with kernel sizes of $d_k \times 1$ and $1 \times d_k$, respectively. Based on this factorization, we combine the two 3-D convolutions into a single one with a kernel size of $d_f \times d_f \times d_c$. Suppose there are $n$ classes, to ensure that the number of output channels equals to the number of classes, $d_c$ is set to $(m - n + 1)$ with no padding on the input. This 3-D convolution is used to replace the global average pooling layer followed by the fully-connected layer, serving as a convolutional classification layer.

While the convolutional classification layer dramatically reduces the number of parameters, there is a concern that it may cause a signification loss in performance. In the fully-connected classification layer, each prediction is based on the entire feature vector by taking all features into consideration. In contrast, in the convolutional classification layer, the prediction of each class uses only $(m - n + 1)$ features. However, our experiments show that the weight matrix of the fully-connected classification layer is very sparse, indicating that only a small number of features contribute to the prediction of a class. Meanwhile, our ChannelNets with the convolutional classification layer achieve much better results than other models with similar amounts of parameters.

## 3.5 ChannelNets

With the proposed group channel-wise convolutions, depth-wise separable channel-wise convolutions, and convolutional classification layer, we build our compact CNNs, which are known as ChannelNets. ChannelNets follow the basic architecture of MobileNets [12] to allow for fair comparison and having different compression levels. We apply our proposed methods to gradually increase the compression level, resulting in three versions of ChannelNets.

To be specific, ChannelNet-v1 employs group channel-wise convolutions. As the number of parameters in CNNs typically grows with network depth, we apply group channel-wise convolutions in deeper layers, in order to make the compression more efficient. ChannelNet-v2 further compresses ChannelNet-v1 by applying depth-wise separable channel-wise convolutions. Here, we only use one depth-wise separable channel-wise convolution to replace the last depth-wise separable convolution of ChannelNet-v1, due to the trade-off between efficiency and performance. On one hand, as pointed out in Section 3.2, the kernel size of the channel-wise convolution in group channel-wise convolutions needs to satisfy $d_c \geq g$, in order to fuse information from all input channels in any output group of channels. In Section 3.3, we reduce group channel-wise convolutions to depth-wise separable channel-wise convolutions by setting $g = m = n$ and relax the constraint. However, it hampers information communication among feature maps, leading to performance loss. Using depth-wise separable channel-wise convolutions frequently will compromise the performance significantly. On the other hand, compared to group channel-wise convolutions, depth-wise separable channel-wise convolutions reduce much more parameters and computation. Therefore, we only apply it to replace the last depth-wise separable convolution, which contributes most to the amount of parameters and computation except for the classification layer. It achieves a better compression level than ChannelNet-v1 with acceptable performance loss. On top of ChannelNet-v2, we apply the convolutional classification layer to compress the classification layer and obtain ChannelNet-v3. The details of network architectures are described below and shown in Table 1.

It is worth noting that our proposed methods are orthogonal to the work of MobileNetV2 [45]. Similar to MobileNets, we can apply our methods to MobileNetV2 to further reduce the parameters and computational cost, as illustrated in Section 4.4.

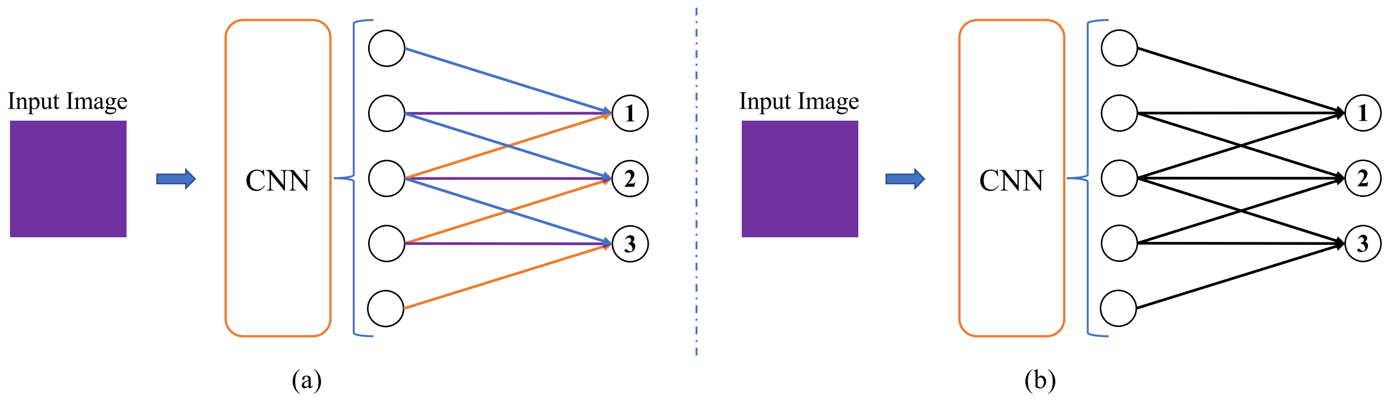


Fig. 4. Illustrations of two different convolutional classification layers introduced in Sections 3.4 and 3.7, respectively. Here, the classification layer takes $m = 5$ features as inputs and predicts $n = 3$ classes. The prediction of each class is based on $m - n + 1 = 3$ features. (a) An example of the convolutional classification layer. The same color indicates the same weight, indicating the same importance. We can see the restriction that adjacent features have the same importance to adjacent classes. (b) An example of the convolutional classification layer without weight-sharing. There is no restriction on the weights.

**ChannelNet-v1:** To employ the group channel-wise convolutions, we design two basic modules; those are, the group module (GM) and the group channel-wise module (GCWM). They are illustrated in Figure 3. GM simply applies $1 \times 1$ group convolution instead of $1 \times 1$ convolution and adds a residual connection [5]. As analyzed above, GM saves computations but suffers from the information inconsistency problem. GCWM addresses this limitation by inserting a group channel-wise convolution after the second $1 \times 1$ group convolution to achieve information fusion. Either module can be used to replace two consecutive depth-wise separable convolutional layers in MobileNets. In our ChannelNet-v1, we choose to replace depth-wise separable convolutions with larger numbers of input and output channels. Specifically, six consecutive depth-wise separable convolutional layers with 512 input and output channels are replaced by two GCWMs followed by one GM. In these modules, we set the number of groups to 2. The number of parameters in ChannelNet-v1 is about 3.7 million.

**ChannelNet-v2:** We apply the depth-wise separable channel-wise convolutions on ChannelNet-v1 to further compress the network. The last depth-wise separable convolutional layer has 512 input channels and $1,024$ output channels. We use the depth-wise separable channel-wise convolution to replace this layer, leading to ChannelNet-v2. The number of parameters reduced by this replacement of a single layer is 1 million, which accounts for about 25% of total parameters in ChannelNet-v1.

**ChannelNet-v3:** We employ the convolutional classification layer on ChannelNet-v2 to obtain ChannelNet-v3. For the ImageNet image classification task, the number of classes is $1,000$, which means the number of parameters in the fully-connected classification layer is $1024 \times 1000 \approx 1$ million. Since the number of parameters for the convolutional classification layer is only $7 \times 7 \times 25 \approx 1$ thousand, ChannelNet-v3 reduces 1 million parameters approximately.

## 3.6 Analysis of Convolutional Classification Layer

Among compact CNNs, ChannelNets make an attempt to compress the classification layer by replacing the final fully-connected layer with the proposed convolutional classification layer. To understand the behavior of the proposed convolutional classification layer, we conduct further analysis in this extended work. Specifically, there are two major differences between the fully-connected classification layer and the convolutional classification layer. We investigate these two differences separately.

The first difference has been pointed out in Section 3.4. In the fully-connected classification layer, the prediction of each class is based on all features in the previous layer. In contrast, only a small subset of features are used when predicting a class in the convolutional classification layer. In another word, the convolutional classification layer makes the assumption that it is sufficient to predict a class based on a small number of features. This assumption is supported by the experimental results in Section 4.8, which show that the weight matrix of the fully-connected classification layer is very sparse. As each weight in the weight matrix can be interpreted as the importance of a feature to a class, the sparsity pattern indicates that the prediction of each class has used only a few important features. These observations motivate the use of the convolutional classification layer.

There may be a concern that the convolutional classification layer not only limits that each prediction uses a fixed small number of features, but also requires that these features are put together spatially. However, this requirement can be naturally met through the training of CNNs. Note that changing the order of features before the classification layer is equivalent to changing the order of corresponding feature maps in previous layers, which can be achieved by swapping the set of convolutional kernels. In CNNs, all the convolutional kernels are learned, meaning that the training of CNNs has the ability of choosing the order of features according to the classification layer.

The second difference is brought by the weight-sharing property of convolutions. In the fully-connected classification layer, the predictions of two different classes have two independent sets of weights, given features in the previous layer. That is, with the same set of features, each class determines the weights on these features independently. However, the predictions of all classes share the same set of weights in the convolutional classification layer, as the channel-wise convolution scans the features using the same kernel. Different classes just have different features corresponding to the same weights. Our in-depth analysis below demonstrates that this requirement might be problematic. Suppose that the classification layer takes $m$ features as inputs and predicts $n$ classes, where each prediction is based on $m - n + 1$ features. There are $m - n$ shared features for adjacent classes. Meanwhile, the weights between the features and the classes are identical after shifting. Such a relationship casts a strong assumption that adjacent features have the same importance to adjacent classes, as illustrated in Figure 4 (a). Considering the variety of features and classes, this assumption might not be valid in practice. In addition, the shared features and the shift-and-share relationship in weights make CNNs affected by the order of classes, as demonstrated by experiments in Section 4.9.

To conclude, there are two assumptions behind the two differences between the fully-connected classification layer and the convolutional classification layer. While the first assumption is reasonable as shown in Section 4.8, the second assumption might be too restrictive and thus lead to performance loss. In addition, the performance of CNNs becomes sensitive to the order of classes, which might not be desirable in practice.

### 3.7 Convolutional Classification Layer without Weight-Sharing

Based on these insights above, we explore to improve the convolutional classification layer. Note that the second assumption is caused by the weight-sharing property of convolutions. To address this potential limitation, we propose to remove the weights-sharing requirement in the convolutional classification layer. To be specific, the prediction of each class now has independent sets of weights like the fully-connected classification layer, while still using a subset of features as in the convolutional classification layer. The convolutional classification layer without weight-sharing is illustrated in Figure 4 (b). It results in better performance than the convolutional classification layer and alleviate the sensitivity to the order of classes by relaxing the second assumption, as shown by experiments in Section 4.10. Note that, unlike the convolutional classification layer, we need a global average pooling layer before the convolutional classification layer without weight-sharing, in order to avoid parameter explosion.

?

?

In terms of compact CNNs, the advantage of using the convolutional classification layer is the dramatic reduction of training parameters with acceptable performance loss. The convolutional classification layer without weight-sharing introduces more training parameters but suffers less from performance loss. In addition, it has the same amount of computation as the convolutional classification layer. The convolutional classification layer without weight-sharing achieves a good trade-off between performance and computational resource requirement. Depending on the required compression level, either the original or the convolutional classification layer without weight-sharing might used to replace the fully-connected classification layer in compact CNNs in practice.

根据所需的压缩级别，在实践中，原始或没有权重共享的卷积分类层可能会用来替换紧凑型 CNN 中的全连接分类层。

## 4 Experimental Studies

In this section, we evaluate the proposed methods and ChannelNets on the ImageNet ILSVRC 2012 image classification dataset [2], which commonly serves as the benchmark for model compression. We compare different versions of ChannelNets with other compact CNNs under various compression levels. In addition to building ChannelNets, we apply our proposed methods directly to other compact CNNs like MobileNetV2. We also perform thorough ablation studies to show the effectiveness of each proposed method and the designed ChannelNets. To fully study the compression of the classification layer, we perform an experiment to show the sparsity of weights in the fully-connected classification layer. Moreover, we conduct experiments to investigate the sensitivity of the convolutional classification layer on different class orders, and illustrate the effectiveness of the convolutional classification layer without weight-sharing. Finally, we demonstrate the advantage of our proposed convolutional classification layers over a simple use of the channel-wise convolution on compressing the classification layer.

### 4.1 Dataset

The ImageNet ILSVRC 2012 dataset contains 1.2 million training images and 50 thousand validation images. Each image is labeled by one of $1,000$ classes. We follow the same data augmentation process in [5]. Images are scaled to $256 \times 256$. Randomly cropped patches with a size of $224 \times 224$ are used for training. During inference, $224 \times 224$ center crops are fed into the networks. To compare with other compact CNNs [12], [30], we train our models using training images and report accuracies computed on the validation set, since the labels of test images are not available.

### 4.2 Experimental Setup

We train our ChannelNets using the same settings as those for MobileNets except for a minor change. For depth-wise separable convolutions, we remove the batch normalization and activation function between the depth-wise convolution and the $1 \times 1$ convolution. We observe that it has no influence on the performance while accelerating the training speed. For the proposed GCWMs, the kernel size of group channel-wise convolutions is set to 8. In depth-wise separable channel-wise convolutions, we set the kernel size to 64. In the convolutional classification layer, the kernel size of the 3-D convolution is $7 \times 7 \times 25$. All models are trained using the stochastic gradient descent optimizer with a momentum of 0.9 for 80 epochs. The learning rate starts at 0.1 and decays by 0.1 at the $45^{th}$, $60^{th}$, $65^{th}$, $70^{th}$, and $75^{th}$ epoch. Dropout [46] with a rate of 0.0001 is applied after $1 \times 1$ convolutions. We use 4 TITAN Xp GPUs and a batch size of 512 for training, which takes about 3 days.

### 4.3 Comparison of ChannelNet-v1 with Other Models

We compare ChannelNet-v1 with other CNNs, including regular networks and compact ones, in terms of the top-1 accuracy, the number of parameters and the computational cost in terms of FLOPs. The results are reported in Table 2. We can see that ChannelNet-v1 is the most compact and

TABLE 2
Comparison between ChannelNet-v1 and other CNNs in terms of the top-1 accuracy on the ImageNet validation set, the number of total parameters, and the number of FLOPs needed for classifying an image.

| Models | Top-1 | Params | FLOPs |
|---|---|---|---|
| GoogleNet | 0.698 | 6.8m | 1550m |
| VGG16 | 0.715 | 128m | 15300m |
| AlexNet | 0.572 | 60m | 720m |
| SqueezeNet | 0.575 | 1.3m | 833m |
| 1.0 MobileNet | 0.706 | 4.2m | 569m |
| ShuffleNet 2x | 0.709 | 5.3m | 524m |
| **ChannelNet-v1** | 0.705 | 3.7m | 407m |

TABLE 3
Comparison between MobileNetV2 and MobileNetV2 with group channel-wise convolutions (MobileNetV2+GCWConv) in terms of the top-1 accuracy on the ImageNet validation set, the number of total parameters, and the number of FLOPs.

| Models | Top-1 | Params | FLOPs |
|---|---|---|---|
| MobileNetV2 | 0.720 | 3.4m | 300m |
| MobileNetV2+GCWConv | 0.718 | 3.0m | 280m |

TABLE 4
Comparison between ChannelNets and other compact CNNs with width multipliers in terms of the top-1 accuracy on the ImageNet validation set, the number of total parameters, and the number of FLOPs. The numbers before the model names represent width multipliers.

| Models | Top-1 | Params | FLOPs |
|---|---|---|---|
| 0.75 MobileNet | 0.684 | 2.6m | 325m |
| 0.75 ChannelNet-v1 | 0.678 | 2.3m | 232m |
| **ChannelNet-v2** | **0.695** | 2.7m | 361m |
| 0.5 MobileNet | 0.637 | 1.3m | 149m |
| 0.5 ChannelNet-v1 | 0.627 | 1.2m | 107m |
| **ChannelNet-v3 (group = 4)** | **0.654** | 1.3m | 245m |
| **ChannelNet-v3** | **0.667** | 1.7m | 360m |

TABLE 5
Comparison between ChannelNet-v1 and ChannelNet-v1 without group channel-wise convolutions, denoted as ChannelNet-v1(-). The comparison is in terms of the top-1 accuracy on the ImageNet validation set, and the number of total parameters.

| Models | Top-1 | Params |
|---|---|---|
| ChannelNet-v1(-) | 0.697 | 3.7m |
| ChannelNet-v1 | 0.705 | 3.7m |

efficient network, as it achieves the best trade-off between efficiency and accuracy.

We can see that SqueezeNet [9] has the smallest size. However, the speed is even slower than AlexNet and the accuracy is not competitive to other compact CNNs. By replacing depth-wise separable convolutions with GMs and GCWMs, ChannelNet-v1 achieves nearly the same performance as 1.0 MobileNet with a 11.9% reduction in parameters and a 28.5% reduction in FLOPs. Here, the 1.0 represents the width multiplier in MobileNets, which is used to control the width of the networks. MobileNets with different width multipliers are compared with ChannelNets under similar compression levels in Section 4.5. ShuffleNet 2x can obtain a slightly better performance. However, it employs a much deeper network architecture, resulting in even more parameters than MobileNets. This is because more layers are required when using shuffling layers to address the information inconsistency problem in $1 \times 1$ group convolutions. Thus, the advantage of using group convolutions is compromised. In contrast, our group channel-wise convolutions can overcome the problem without more layers, as shown by experiments in Section 4.6.

## 4.4 Results on MobileNetV2

As pointed out in Section 3.5, our proposed methods are orthogonal to the work of MobileNetV2 and can be used together to build compact CNNs. Therefore, we apply our methods on MobileNetV2 to explore broader application scenarios. To be concrete, the main components of MobileNetV2 are bottleneck blocks with inverted residuals. Each bottleneck block consists of a $1 \times 1$ convolution for expansion, a $3 \times 3$ depth-wise convolution, and another $1 \times 1$ convolution. We use our proposed group channel-wise convolution to replace the second $1 \times 1$ convolution in the last four blocks of MobileNetV2. These blocks contain many parameters and account for a large portion of trainable parameters in MobileNetV2.

The comparison results are summarized in Table 3. MobileNetV2 with group channel-wise convolutions achieves almost the same performance as MobileNetV2 while saving 0.4 million trainable parameters, which accounts for 11.8% of total parameters in MobileNetV2. These results show the effectiveness, scalability, and flexibility of our proposed methods.

## 4.5 Comparison of ChannelNets with Models Using Width Multipliers

The width multiplier is proposed in [12] to make the network architecture thinner by reducing the number of input and output channels in each layer, thereby increasing the compression level. This approach simply compresses each layer by the same factor. Note that most of parameters lie in deep layers of the model. Hence, reducing widths in shallow layers does not lead to significant compression, but hinders model performance, since it is important to maintain the number of channels in the shallow part of deep models. Our ChannelNets explore a different way to achieve higher compression levels by replacing the deepest layers in CNNs. Remarkably, ChannelNet-v3 is the first compact network that attempts to compress the last layer, *i.e.,* the fully-connected classification layer.

We perform experiments to compare ChannelNet-v2 and ChannelNet-v3 with compact CNNs using width multipliers. The results are shown in Table 4. We apply width multipliers $\{0.75, 0.5\}$ on both MobileNet and ChannelNet-v1 to illustrate the impact of applying width multipliers. In order to make the comparison fair, compact networks with similar compression levels are compared together. Specifically, we compare ChannelNet-v2 with 0.75 MobileNet and 0.75 ChannelNet-v1, since the numbers of total parameters are in the same 2.x million level. For ChannelNet-v3, 0.5 MobileNet and 0.5 ChannelNet-v1 are used for comparison, as all of them contain 1.x million parameters. We also include a version of ChannelNet-v3 with nearly the same

TABLE 6
Comparison between ChannelNet-v1 using with different numbers of groups in terms of the top-1 accuracy on the ImageNet validation set, and the number of total parameters. The values in brackets indicate the number of groups used in GM and GCWMs.

| Models | Top-1 | Params |
|---|---|---|
| ChannelNet-v1 (group = 2) | 0.705 | 3.7m |
| ChannelNet-v1 (group = 3) | 0.685 | 3.5m |
| ChannelNet-v1 (group = 4) | 0.679 | 3.3m |

amount of parameters, denoted as ChannelNet-v3 (group = 4), in order to allow for more direct comparisons. To be specific, we set the number of groups in GM and GCWMs to 4 in ChannelNet-v3. Experiments showing the effect of the number of groups are presented in Section 4.7.

We can observe from the results that ChannelNet-v2 outperforms 0.75 MobileNet with an absolute 1.1% gain in accuracy, which demonstrates the effect of our depth-wise separable channel-wise convolutions. In addition, note that using depth-wise separable channel-wise convolutions to replace depth-wise separable convolutions is a more flexible way than applying width multipliers. It only affects one layer, as opposed to all layers in the networks. ChannelNet-v3 has significantly better performance than 0.5 MobileNet by 3% in accuracy. ChannelNet-v3 (group = 4) using 4 groups in GM and GCWMs outperforms 0.5 MobileNet by a margin of 1.7% in top-1 accuracy. It shows that our convolutional classification layer can retain the accuracy to most extent while increasing the compression level. The results also show that applying width multipliers on ChannelNet-v1 leads to poor performance.

## 4.6 Study of Group Channel-Wise Convolutions

To demonstrate the effect of our group channel-wise convolutions, we conduct an ablation study on ChannelNet-v1. Based on ChannelNet-v1, we replace the two GCWMs with GMs, thereby removing all group channel-wise convolutions. The model is denoted as ChannelNet-v1(-). It follows exactly the same experimental setup as ChannelNet-v1 to ensure fairness. Table 5 provides comparison results between ChannelNet-v1(-) and ChannelNet-v1. ChannelNet-v1 outperforms ChannelNet-v1(-) by 0.8%, which is significant as ChannelNet-v1 has only 32 more parameters with group channel-wise convolutions. Therefore, group channel-wise convolutions are extremely efficient and effective information fusion layers for solving the problem incurred by group convolutions.

## 4.7 Performance Study of the Number of Groups in ChannelNets

The number of groups is an important hyper-parameter in both GM and GCWMs, which also affects the trade-off between compression and performance. In the original ChannelNet-v1, we use 2 groups in GM and GCWMs. We conduct experiments to investigate how different numbers of groups affect the performance of ChannelNets. Based on ChannelNet-v1, we vary the number of groups in GM and GCWMs in $[2, 3, 4]$, which cover a reasonable range of choices. We report the performance in terms of the top-1 accuracy in Table 6. We can observe that the performance decreases as the number of groups increases, while the number of total parameters decreases. However, the performance loss is relatively large with a small amount of reduction in model size. This indicates that setting the number of groups to large values in GMs and GCWMs will significantly degrade the performance of ChannelNets. Therefore, in order to further compress ChannelNet-v1, we employ the proposed depth-wise separable channel-wise convolutions and convolutional classification layer, instead of simply increasing the number of groups in GMs and GCWMs, as illustrated by ChannelNet-v2 and ChannelNet-v3.

## 4.8 Sparsity of Weights in Fully-Connected Classification Layers

In ChannelNet-v3, we replace the fully-connected classification layer with our convolutional classification layer. Each prediction is based on only $(m - n + 1)$ features instead of all $n$ features, which raises a concern of potential loss in performance. To investigate this further, we analyze the weight matrix in the fully-connected classification layer, as shown in Figure 5. We take the fully-connected classification layer of ChannelNet-v1 as an example. The analysis shows that the weights are sparsely distributed in the weight matrix, which indicates that each prediction only makes use of a small number of features, even with the fully-connected classification layer. Based on this insight, we propose the convolutional classification layer and ChannelNet-v3. As shown in Section 4.5, ChannelNet-v3 is highly compact and efficient with promising performance.

## 4.9 Impact of Class Orders on Convolutional Classification Layer

The analysis in Section 3.6 shows that the shift-and-share relationship for weights in the convolutional classification layer (CCL) constrains the capacity of the output layer. A consequence of the weight-sharing pattern is that the order of classes during training may have a significant impact on the performance of the classification layer. We perform experiments to explore the impact of class orders on performance. To fully explore the impacts of class orders on networks using CCL, we employ the network architecture of ChannelNet-v1. Based on ChannelNet-v1, we replace the global average pooling layer and the fully-connected output layer with CCL. Note that we do not use ChannelNet-v3, since the last depth-wise separable channel-wise convolution in ChannelNet-v3 may influence evaluation results.

We train the network with different class orders, including two random orders, the original class order, optimal leaf order, and inverse optimal leaf order. For two random orders, we use different random seeds to generate class orders. Since there is feature sharing in CCL, we propose to evaluate the optimal leaf order, which places similar classes close to each other. Specifically, since adjacent classes share features and classification weights, more effective feature sharing can be achieved when similar classes are placed to be close to each other. For image classification tasks on the ImageNet ILSVRC 2012 dataset, there is an $1024 \times 1000$ weights matrix

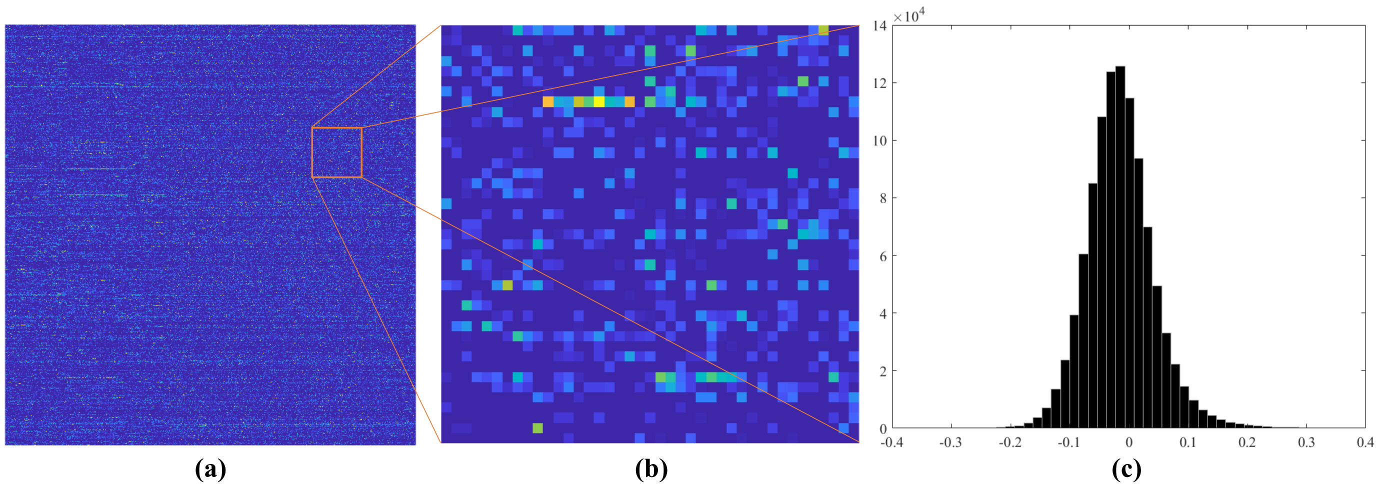


Fig. 5. An example of the weight patterns in the fully-connected classification layer of ChannelNet-v1. Part (a) shows the weight matrix of the fully-connected classification layer. We can see that the weights are sparsely distributed, as most of the weights are in blue color, which indicates zero or near zero values. Part (b) gives a close look of weights in a small region. The histogram in part (c) statistically shows the sparsity of weights.

TABLE 7
Comparison of ChannelNet-v1 using the convolutional classification layer with different class orders in terms of the top-1 accuracy and number of parameters.

| Models | Top-1 | Params |
|---|---|---|
| Random Order (Seed 0) | 0.671 | 2.86m |
| Random Order (Seed 100) | 0.676 | 2.86m |
| Original Order | 0.671 | 2.86m |
| Optimal Leaf Order | 0.674 | 2.86m |
| Inverse Optimal Leaf Order | 0.672 | 2.86m |

TABLE 8
Comparison of ChannelNet-v1 using the convolutional classification layer without weight-sharing with different class orders in terms of the top-1 accuracy and number of parameters.

| Models | Top-1 | Params |
|---|---|---|
| Random Order (Seed 0) | 0.688 | 2.88m |
| Random Order (Seed 100) | 0.688 | 2.88m |
| Original Order | 0.688 | 2.88m |
| Optimal Leaf Order | 0.689 | 2.88m |
| Inverse Optimal Leaf Order | 0.687 | 2.88m |

in the last fully-connected layer from a trained ChannelNet-v1. For similar classes, they may employ similar important features for prediction, which results in similar weight vectors for similar classes. Based on this insight, each column vector of size 1024 in the weight matrix can be used as the feature vector of a class. We employ the optimal leaf ordering [47] to rank classes, which places similar classes close to each other based on their cosine similarity scores. We also evaluate the inverse optimal leaf order, which is obtained by the optimal leaf ordering using inverse distance similarity. This serves as the opposite of classes ordering by the optimal leaf order.

The results are summarized in Table 7. We can observe from the results that the variance of top-1 accuracies is very large. The network trained using the random order with seed 100 outperforms that trained with the original order by a margin of 0.5%. This indicates that the shift-and-share pattern on weights makes CCL sensitive to the class order. Notably, the model trained with the optimal leaf order achieves lower performance than that with the second random order. This means the weight sharing pattern makes it hard to distinguish testing examples of similar classes with shared features. The rearrangement of classes does not overcome the limitations of CCL.

### 4.10 Results on Convolutional Classification Layer without Weight-Sharing

In Section 3.7, we propose the convolutional classification layer without weight-sharing (CCL w/o WS), which removes the sharing pattern of weights in CCL. For classifiers of different classes, the non-sharing weights may overcome the limitations of CCL. Following the same network architecture in Section 4.9, we replace CCL with CCL w/o WS. We perform experiments with the same set of class orders evaluated in Section 4.9. The results are summarized in Table 8. We can observe from the results that the networks trained with different class orders achieve very similar performance in terms of the top-1 classification accuracy. The best performance is achieved by employing the optimal leaf order with a margin of 0.1% over networks trained with other class orders. This demonstrates that our CCL w/o WS overcomes the limitations of CCL and makes the network not sensitive to the class order. When comparing with performance of ChannelNet-v1 with CCL in Table 7, ChannelNet-v1 with CCL w/o WS achieves performance improvement by at least a margin of 1.1%. Note that the ChannelNet-v1 with CCL w/o WS only has 0.02 million more parameters than the ChannelNet-v1 with CCL. The significant performance improvement of the network with CCL w/o WS over that with CCL is due to the removal of sharing pattern on classifier weights.

### 4.11 Ablation Study of Different Classification Layers

We present two strategies to compress the classification layer. We propose convolutional classification layer (CCL)

TABLE 9
Comparison between ChannelNet-v3 with different classification layers in terms of the top-1 accuracy on the ImageNet validation set, and the number of total parameters. ChannelNet-v3 uses the convolutional classification layer. ChannelNet-v3 (AVG+CWConv) uses an average pooling followed by a channel-wise convolution. ChannelNet-v3 (AVG+CCL w/o WS) uses the convolutional classification layer without weight-sharing.

| Models | Top-1 | Params |
|---|---|---|
| ChannelNet-v3 | 0.667 | 1.73m |
| ChannelNet-v3 (AVG+CWConv) | 0.649 | 1.73m |
| ChannelNet-v3 (AVG+CCL w/o WS) | 0.667 | 1.76m |

to replace the global average pooling layer and fully-connected (FC) layer, leading to ChannelNet-v3. However, CCL may be sensitive to different class orders, as illustrated in Section 4.9. The convolutional classification layer without weight-sharing (CCL w/o WS) is proposed to alleviate this problem. Note that CCL w/o WS keeps the global average pooling layer and only replaces the FC layer. Besides these two CCL-based compression strategies, there is another strategy that simply replaces the FC layer with a channel-wise convolution (CWConv). In this section, we compare these methods based on ChannelNet-v3. The comparison results are summarized in Table 9. The three models have similar numbers of total parameters. However, both CCL-based methods outperform the simple strategy that uses a CWConv to replace the FC layer significantly. These results show the effectiveness of our proposed convolution classification layers, with or without weight-sharing. Note that CCL w/o WS achieves the same performance as CCL on ChannelNet-v3. This is due to the influence of the last depth-wise separable channel-wise convolution in ChannelNet-v3, which compromises the benefit of CCL w/o WS.

## 5 Conclusion and Future Work

In this work, we propose channel-wise convolutions to perform model compression by replacing dense connections in deep networks. We build a new family of compact and efficient CNNs, known as ChannelNets, by using three instances of channel-wise convolutions; namely group channel-wise convolutions, depth-wise separable channel-wise convolutions, and the convolutional classification layer. Group channel-wise convolutions are used together with $1 \times 1$ group convolutions as information fusion layers. Depth-wise separable channel-wise convolutions can be directly used to replace depth-wise separable convolutions in compact CNNs. The convolutional classification layer is an attempt in the field of model compression to compress the fully-connected classification layer. Along this new direction, we further analyze the behavior of the convolutional classification layer and propose the convolutional classification layer without weight-sharing. Both the original and convolutional classification layer without weight-sharing serve as good options in compressing the fully-connected classification layer. Compared to prior methods, ChannelNets achieve a better trade-off between efficiency and accuracy. The current study evaluates the proposed methods on image classification tasks, but the methods can be applied to other tasks, such as segmentation.

## Acknowledgments

This work was supported in part by National Science Foundation grants IIS-1633359 and DBI-1641223.

## References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, 2009, pp. 248–255.

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *Proceedings of the International Conference on Learning Representations*, pp. 1–14, 2015.

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

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[6] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[7] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2018.

[8] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 2, pp. 295–307, 2016.

[9] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, pp. 1–13, 2016.

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

[11] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4820–4828.

[12] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, pp. 1–9, 2017.

[13] F. Tung and G. Mori, "Deep neural network compression by in-parallel pruning-quantization," *IEEE transactions on pattern analysis and machine intelligence*, pp. 1–1, 2018.

[14] Y. Wei, X. Pan, H. Qin, W. Ouyang, and J. Yan, "Quantization mimic: Towards very tiny cnn for object detection," in *Proceedings of the European Conference on Computer Vision*, 2018, pp. 267–283.

[15] A. See, M.-T. Luong, and C. D. Manning, "Compression of neural machine translation models via pruning," in *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, 2016, pp. 291–301.

[16] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5058–5066.

[17] S. Chen and Q. Zhao, "Shallowing deep networks: Layer-wise pruning based on feature representations," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 12, pp. 3048–3056, 2018.

[18] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *International Conference on Machine Learning*, 2015, pp. 2285–2294.

[19] V. E. Liong, J. Lu, L.-Y. Duan, and Y. Tan, "Deep variational and structural hashing," *IEEE transactions on pattern analysis and machine intelligence*, pp. 1–1, 2018.

[20] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *International Conference on Learning Representations*, pp. 1–14, 2015.

[21] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," *arXiv preprint arXiv:1412.6553*, pp. 1–11, 2014.

[22] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *Proceedings of the British Machine Vision Conference. BMVA Press*, 2014, pp. 1–13.

[23] E. Romera, J. M. Alvarez, L. M. Bergasa, and R. Arroyo, "Erfnet: Efficient residual factorized convnet for real-time semantic segmentation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 1, pp. 263–272, 2018.

[24] M. Lin, Q. Chen, and S. Yan, "Network in network," *Proceedings of the International Conference on Learning Representations*, pp. 1–10, 2013.

[25] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.

[26] L. Sifre and P. Mallat, "Rigid-motion scattering for image classification," Ph.D. dissertation, Ecole Polytechnique, 2014.

[27] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 2017, pp. 4278–4284.

[28] H. Kim, J. Sim, Y. Choi, and L.-S. Kim, "A kernel decomposition architecture for binary-weight convolutional neural networks," in *Proceedings of the 54th Annual Design Automation Conference*. ACM, 2017, pp. 1–6.

[29] J. Chang and J. Sha, "An efficient implementation of 2d convolution in cnn," *IEICE Electronics Express*, pp. 13–34, 2016.

[30] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6848–6856.

[31] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, November 1998.

[32] H. Gao, Z. Wang, and S. Ji, "Channelnets: Compact and efficient convolutional neural networks via channel-wise convolutions," in *Advances in Neural Information Processing Systems*, 2018, pp. 5203–5211.

[33] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[34] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, "Deeper depth prediction with fully convolutional residual networks," in *2016 Fourth International Conference on 3D Vision*. IEEE, 2016, pp. 239–248.

[35] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

[36] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[37] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[38] T. Zhang, G.-J. Qi, B. Xiao, and J. Wang, "Interleaved group convolutions," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 4373–4382.

[39] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.

[40] G. Huang, S. Liu, L. Van der Maaten, and K. Q. Weinberger, "Condensenet: An efficient densenet using learned group convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2752–2761.

[41] R. Zhao, Y. Hu, J. Dotzel, C. D. Sa, and Z. Zhang, "Building efficient deep neural networks with unitary group convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 303–11 312.

[42] B. Peng, W. Tan, Z. Li, S. Zhang, D. Xie, and S. Pu, "Extreme network compression via filter group approximation," in *Proceedings of the European Conference on Computer Vision*, 2018, pp. 300–316.

[43] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2820–2828.

[44] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 221–231, 2013.

[45] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[46] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[47] Z. Bar-Joseph, D. K. Gifford, and T. S. Jaakkola, "Fast optimal leaf ordering for hierarchical clustering," *Bioinformatics*, vol. 17, no. suppl_1, pp. S22–S29, 2001.

**Hongyang Gao** received his B.S. in Biomedical English from Peking University of China in 2009 and his M.S. in Computer Science from Tsinghua University of China in 2012. He is currently a Ph.D. candidate in the Department of Computer Science and Engineering, Texas A&M University, College Station, Texas. His research interests include machine learning, deep learning, and data mining.

**Zhengyang Wang** received the Master of Science degree in mathematics and computer science from New York University, New York City, New York, in 2015. Currently, he is a PhD student in the Department of Computer Science and Engineering, Texas A&M University, College Station, Texas. His research interests include machine learning, deep learning, and data mining.

**Lei Cai** received the B.S. and M.S. degrees from Dalian University of Technology in 2013 and 2016, respectively. He is currently a Ph.D. student in the School of Electrical Engineering and Computer Science, Washington State University, Pullman, Washington. His research interests include data mining, machine learning and deep learning.

**Shuiwang Ji** received the PhD degree in computer science from Arizona State University, Tempe, Arizona, in 2010. Currently, he is an Associate Professor in the Department of Computer Science and Engineering, Texas A&M University, College Station, Texas. His research interests include machine learning, data mining, and computational biology. He received the National Science Foundation CAREER Award in 2014. He is currently a Section Editor for BMC Bioinformatics, an Action Editor for Data Mining and Knowledge Discovery, and an Associate Editor for ACM Transactions on Knowledge Discovery from Data. He is a senior member of IEEE.