Deep Dive into Different Types of Convolutions for Deep Learning

Deformable Convolution

Data Representation

Convolution Arithmetic

Dilated Convolution

3D Convolution

Transposed Convolution

Receptive Field Calculation

Grouped Convolution

Spatially Separable Convolution

Depthwise Separable Convolution

Amir Hossein Karami
(Senior Deep Learning Research Scientist)

Deep Dive into Different Types of Convolutions for Deep Learning

Amir Hossein Karami

This book is for sale at http://leanpub.com/convolutions-for-deep-learning

This version was published on 2019-12-01

Tweet This Book!

Please help Amir Hossein Karami by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I just purchased ”Deep Dive into Different Types of Convolutions for Deep Learning” by @DeepLearninGuru (Amir Hossein Karami) on @leanpub - https://leanpub.com/convolutions-for-deep-learning #ConvolutionsforDeepLearning

The suggested hashtag for this book is #ConvolutionsforDeepLearning.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#ConvolutionsforDeepLearning

This book is dedicated to my loving and supportive parents: Ahmad, and Guiti.

Contents

Preface

Deep Dive into Different Types of Convolutions for Deep Learning[1] introduces wide range of convolution operators in the field of *Deep Learning*. As a deep learning researcher/engineer, in many real-world projects you will need to design special deep models that will require extensive knowledge of how to represent different data types and other components of deep networks (such as different types of convolution operators). I believe that there is a shortage of resources in the field of deep learning, according to the viewpoint that most of them have expressed their content in a very superficial way, and without describing the necessary details. Accordingly, since the convolution operator plays a critical role in the design of various types of *Convolutional Neural Networks* (CNN models), this book provides a detailed description of the various types of convolution operators with other necessary supplementary materials. Here are some of the concepts you will learn when reading this book:

- You will learn how to represent and model different kinds of data (*e.g.*, time series data, textual data, image data, video data, audio data, medical data (Genomics-based and Radiomics-based)) for deep learning projects
- You will learn the fundamental concepts of convolution operator (such as its relation to cross-correlation, how to apply it on different kinds of data)
- You will be thoroughly familiar with how the convolution operator is represented, and its related arithmetic (*i.e.*, a general representation format for different types of convolutions is presented which is compatible with PyTorch, Keras, and TensorFlow frameworks). Also all of the points about the activation map shapes (output tensors shapes), and their number of learnable parameters are also mentioned
- You will be well acquainted with the concept of 3D convolution and its applications (e.g., in video understanding tasks (such as video activity recognition, designing 3D ConvNets, etc.), and 3D data processing)
- You will be well acquainted with the concept of 1D convolution operator, and its applications in textual data processing, natural language processing (NLP) tasks, and data science projects
- You will be familiar with the 1×1 convolution operator, and its critical role in *Network Distillation* topic, *Inception* module of *GoogLeNet*, etc.
- You will learn unique notes about *Transposed convolution* or *Deconvolution*, how to apply it in various tasks (*e.g.*, semantic image segmentation, object detection, etc.), its arithmetic notes, and so on
- You will be well acquainted with the concepts of *Dilated convolution*, its different types, and its arithmetic notes
- You will completely learn necessary notes about the *Receptive Field* calculations

[1] https://leanpub.com/convolutions-for-deep-learning

- You will thoroughly learn extremely useful notes and techniques about the different methods of *Separable convolution* (*i.e.*, *Spatially Separable convolution*, *Depthwise Separable convolution*, and *Pseudo-3D Convolution*)
- You will learn how to apply *Grouped convolution* in general cases (*i.e.*, on 2D and 3D data types)
- You will get lots of interesting and useful ideas on advanced cutting edge convolution techniques, such as: *Deformable convolution*, *Shuffled Grouped convolution*, *3D Temporal Deformable convolution*, etc.

In fact, this book builds your understanding through intuitive explanations and practical examples. You'll explore challenging concepts and learn how to tackle with practical tasks in computer vision, natural-language processing, audio processing, etc. by using different kinds of convolutions. Reading this book is recommended to all researchers, and engineers in this field with any level of knowledge (especially it would be a great handbook for deep learning researchers to design state-of-the-art CNN models). By the time you finish, you'll have the great knowledge and hands-on skills to apply different convolution operators in your own projects (*i.e.*, for designing special CNN models).

You can also send any feedback or corrections to: ah.karami.dl@gmail.com
As a final note, I would like to express my appreciation toward Mr. Alireza Moeini for designing the book cover.

— Amir Hossein Karami, 2019

Chapter 1: Data Representation

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

1.1 Scalar (0D tensor)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

1.2 Vector (1D tensor)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

1.3 Matrix (2D tensor)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

1.4 3D tensor and tensor of higher dimensions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

1.5 The concept of data batches

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

1.6 Real-world examples of data tensors

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

1.7 Vector data

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

1.8 Time series data or sequence data

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

1.9 Image data

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

1.10 Video data

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

1.11 Audio data

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

Chapter 2: Convolution and Cross-Correlation

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

Chapter 3: Convolution on One-dimensional Images

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

Chapter 4: Convolution on Multi-dimensional Data

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

Chapter 5: 2D Convolution Arithmetic

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

Chapter 6: 3D Convolution

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

Chapter 7: 3D Convolution Arithmetic

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

Chapter 8: 1D Convolution

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

Chapter 9: 1D Convolution Arithmetic

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

Chapter 10: 1 × 1 Convolution

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

Chapter 11: Transposed Convolution (Deconvolution)

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

11.1 Transposed convolution arithmetic

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

11.2 Checkerboard artifacts

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

Chapter 12: Dilated Convolution

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

12.1 Gridding artifacts

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

12.2 Dilated convolution arithmetic

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

Chapter 13: Receptive Field

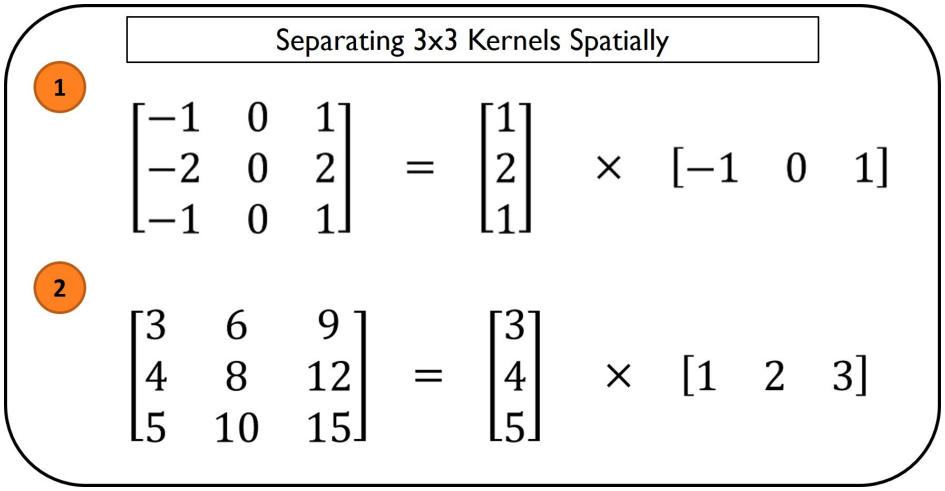
This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

Chapter 14: Separable Convolution

In this chapter we are going to take a look at one of the important topics of convolutional networks which is called *Separable Convolution*. Separable convolution, which can be considered as separating convolutional filters into several smaller filters, has been used in some convolutional network architectures such as MobileNet. There are two ways for separating the convolution operator filters: 1- *Spatially Separable Convolution*, and 2- *Depthwise Separable Convolution*. In this chapter, we will first provide the necessary explanations and tips regarding the spatially separable convolution, and then the necessary explanations will be given about the topic of the depthwise separable convolution.

14.1 Spatially separable convolution

In the case of spatially separable convolution, the aim is to separate (decompose) the convolutional kernels along the spatial dimensions (*i.e.*, height and width) into two separate kernels. In other words, spatially separable convolution operates on 2D space (*i.e.*, the height and width of the input data). We can intuitively consider the spatially separable convolution operation so that instead of using a relatively high-dimensional kernel, we use two smaller decomposed kernels in a convolution operator. That is, we model the original convolution operator with relatively high-dimensional kernel with two smaller-sized kernel convolution operators, which ultimately reduces the size of the deep network. For instance, in Figure 14.1 two different examples provided, in which the 3×3 kernel is separated into two smaller kernels of 3×1 and 1×3 dimensions.



$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 6 & 9 \\ 4 & 8 & 12 \\ 5 & 10 & 15 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

Figure 14.1 Examples of separating 3×3 kernels into two smaller kernels

Given the above example, we can say that in normal convolution, the kernel of 3×3 dimensions is directly convolved over the input data. However, in spatially separable convolution, the kernel of

3 × 1 is first convolved over the input data, and then the kernel of 1 × 3 is used. Therefore, in this method the number of learnable parameters (*i.e.*, weights) of the kernels of convolution operation is reduced from 9 to 6. Consequently, we can say that using spatially separable convolution can reduce the number of weights and learnable parameters of a network. In addition to reducing the number of weights and parameters, the use of spatially separable convolution can also reduce the number of matrix multiplication operations. In order to better understand this matter, in the following we have provided a complete example. Suppose we applied a convolution operator with 3 × 3 kernel size, stride value of one, and without zero-padding (*i.e.*, $k = 3, s = 1, p = 0$) over a 5 × 5 input data. Convolving the mentioned kernel over the input data allows the kernel to be positioned in 3 different locations in the horizontal direction as well as in 3 different positions in the vertical direction, thus scanning the entire data elements. Therefore, the kernel is placed on the input data in 9 (3 × 3 = 9) different positions, and a dot product is performed at each time. Figure 14.2 shows how this operation performs. Notice that in Figure 14.2, 9 different positions that the kernel's center placed on the data are shown as dots. Each time the kernel is placed on the input data, 9 element-wise multiplications are performed. Therefore, in this convolution operation, in total 81 (9 × 9 = 81) multiplications are performed.

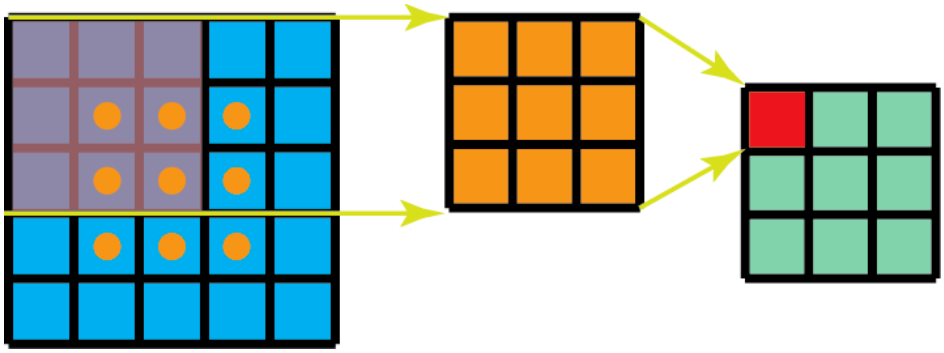


Figure 14.2 Performing a normal convolution operation on a single-channel data

Now we want to do the above example by using the spatially separable convolution. To this end, we first convolve a 3 × 1 kernel over the input data. The center of this 3 × 1 kernel will be placed on the input data in 5 different positions in the horizontal direction, and in 3 different positions in the vertical direction; where this means that in total it will be placed on 15 (5 × 3 = 15) different positions on the input data. These 15 positions are indicated in Figure 14.3 as dots. In each position where this kernel will be placed on the input data, there will be 3 element-wise multiplications. Therefore, to convolving this kernel over the input data, a total of 45 (15 × 3 = 45) multiplications will be performed. The output of the first convolution will be a 3 × 5 tensor. Then, in the second step, we convolve a 1 × 3 kernel over the first convolution output tensor. The center of the mentioned kernel is placed in 3 positions in the horizontal direction, and in 3 different positions in the vertical direction (the centers of these 9 positions are also illustrated in Figure 14.3 with dots). In each position where this kernel is placed on the data, there will also be 3 element-wise multiplications. Therefore, a total of 27 (9 × 3 = 27) multiplications are required to perform the second convolution operation. As a result, in order to perform the first and second convolution operations (*i.e.*, the spatially separable convolution) in this example, 72 (45 + 27 = 72) multiplications are required. As you have understood, the number

of multiplications of the spatially separable convolution is less than applying a normal convolution operation with a larger kernel (72 < 81). Figure 14.3 illustrates how to apply the spatially separable convolution in the above example.

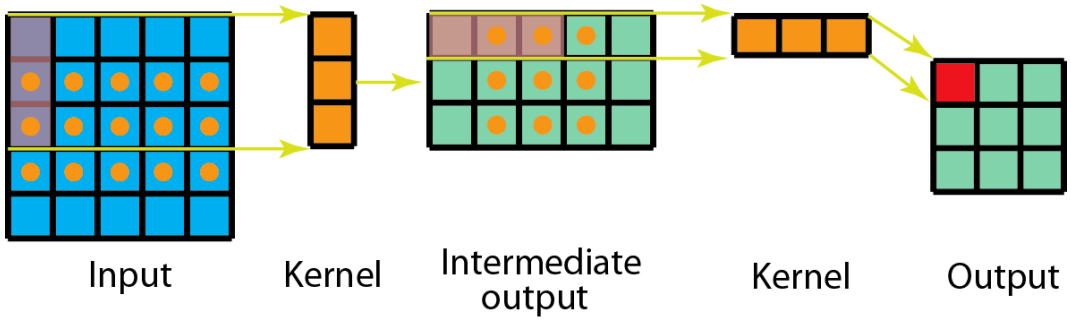


Figure 14.3 An example of how to apply a spatially separable convolution

Now let's generalize the above example and consider the general case. Suppose we want to perform a convolution with $m \times m$ kernel, stride value of one, and without zero-padding (*i.e.*, $k = m, s = 1, p = 0$) on an input data of $n \times n$ dimensions. Performing a normal convolution operation by the mentioned configuration requires m^2 ($m \times m = m^2$) parameters (weights), and $(n - m + 1)^2 \times m^2$ multiplications. Now if we want to do the same convolution operation using the spatially separable convolution method, we will need two kernels with $m \times 1$ and $1 \times m$ dimensions. Applying a convolution operation using a spatially separable convolution method will require a total of $2m$ ($m + m = 2m$) parameters (weights), and $n \times (n-m+1) \times m + (n - m + 1)^2 \times m$ multiplications. Also with a simple simplification we find that the number of multiplications required by this method is equal to $(n-m+1) \times m \times (2n - m + 1)$. The ratio of the number of multiplications required in the spatially separable convolution to the number of multiplications required in the normal convolution method is as follows:

$$\frac{2n - m + 1}{m(n - m + 1)} = \frac{1}{m} + \frac{n}{m(n - m + 1)} \qquad (14.1)$$

Given the equation 14.1 for layers of a network where the size of the input tensor (*i.e.*, n) is substantially greater than the size of the convolution kernel (m) (*i.e.*, $n \gg m$), we can approximate the equation 14.1 to be as $\frac{2}{m}$. Because the $\frac{n}{m(n-m+1)}$ ratio can be approximated as $\frac{n}{mn} = \frac{1}{m}$ when the value of n is substantially greater than m. If we consider the number of required multiplications as a measure of the computational cost of these methods, then in such a case the ratio of computational costs between the spatially separable convolution and the normal convolution for a 3 × 3 kernel is $\frac{2}{3}$, for a 5 × 5 kernel is $\frac{2}{5}$, for a 7 × 7 kernel is $\frac{2}{7}$, and so on.

Here maybe a question arises for the reader, and that is "given that the spatially separable convolution method is computationally more cost-effective than the normal convolution, so why isn't it always used in deep networks architectures instead of the normal convolution?". In order to answer this question, we can say that in two ways the normal convolution kernels can be spatially separated (*i.e.*, decomposed). In the first method, it is possible to design the network first by the normal convolutional layers, and all of its parameters and weights trained (*i.e.*, its training phase completely done); then we should separate all the trained kernels into two smaller decomposed kernels. Unfortunately, we must say that this method is not practical in many cases. Because, as

discussed in *Matrix Decomposition* or *Matrix Factorization* topics, in many cases it is not possible to separate any arbitrary matrix (or tensor) into two smaller ones (*i.e.*, multiplication of two smaller tensors). It is worth noting that the details of this argument and its proof are beyond the scope of this book, and require great knowledge of the matrix analysis and decomposition topics.
In the second method, before the training phase we can separate the kernels of the convolution operation into smaller kernels, as what we have mentioned above (*i.e.*, use the spatially separable convolution method when designing the network structure). This method will also have its own problems. More precisely, as the kernel parameters are reduced in this method, it has been found in various experiments that in practice the network search space will be small, and somehow the network after training phase will trapped in a sub-optimal state. Although this method will reduce the computational cost and the number of parameters of the network, it can also dramatically reduce the accuracy of the network. Therefore, the spatially separable convolution method is usually used in cases where it is extremely important for us to reduce the size of the network and increase its computational speed. For example, when we want to design a network that runs on smartphones or embedded devices, then we can reap the benefits of this approach.

14.2 Depthwise separable convolution

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

14.3 Pseudo-3D convolution

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

Chapter 15: Grouped Convolution

Grouped Convolution method was first introduced on the popular AlexNet network in 2012. The reason for using this method in that model was that the size of the AlexNet model was high and since GPU technology in that year had not yet reached its advanced level in recent years, so the network designer just has GPUs with 1.5 GB of memory, and he has not been able to train the entire model with the necessary training data by a single GPU. Therefore, the AlexNet network designer used the grouped convolution method to *model parallelization* on 2 GPUs. In general, grouped convolution method is an interesting convolution method that has many benefits including the possibility of model parallelization on multiple GPUs. In this chapter we will learn how to implement this method, and other tips and details on its benefits.

As we have remembered, in 2D normal convolution operation, the input data of shape (C_{in}, H_{in}, W_{in}) is converted to the output tensor of shape $(C_{out}, H_{out}, W_{out})$ by means of C_{out} filters of shape (C_{in}, k_H, k_W) (such as Figure 15.1). Note that the batch size is ignored for brevity.

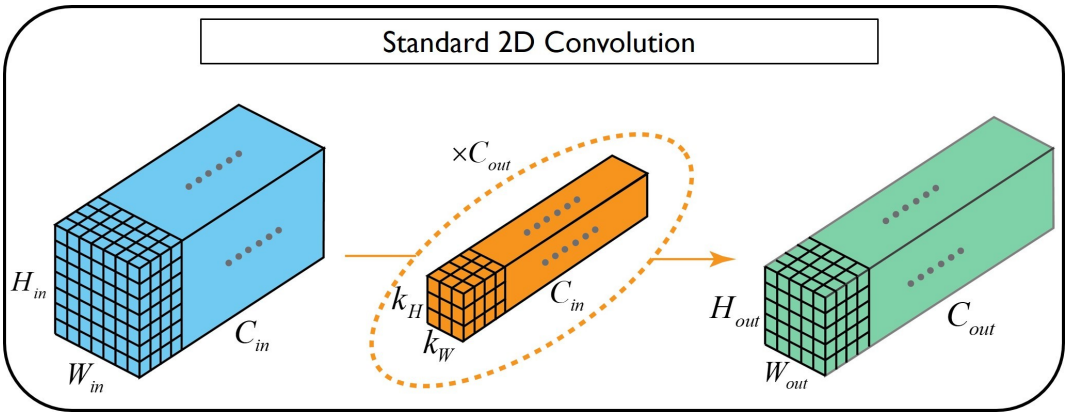


Figure 15.1 A demonstration of applying a 2D normal convolution

In the grouped convolution method, the filters are divided into different groups, and each group of filters has the task of performing the normal convolution operation on a specific part of the input data. The output tensors of each group are then joined together to form the final output tensor of the grouped convolution operation. Figure 15.2 shows a demonstration of the implementation of this method.

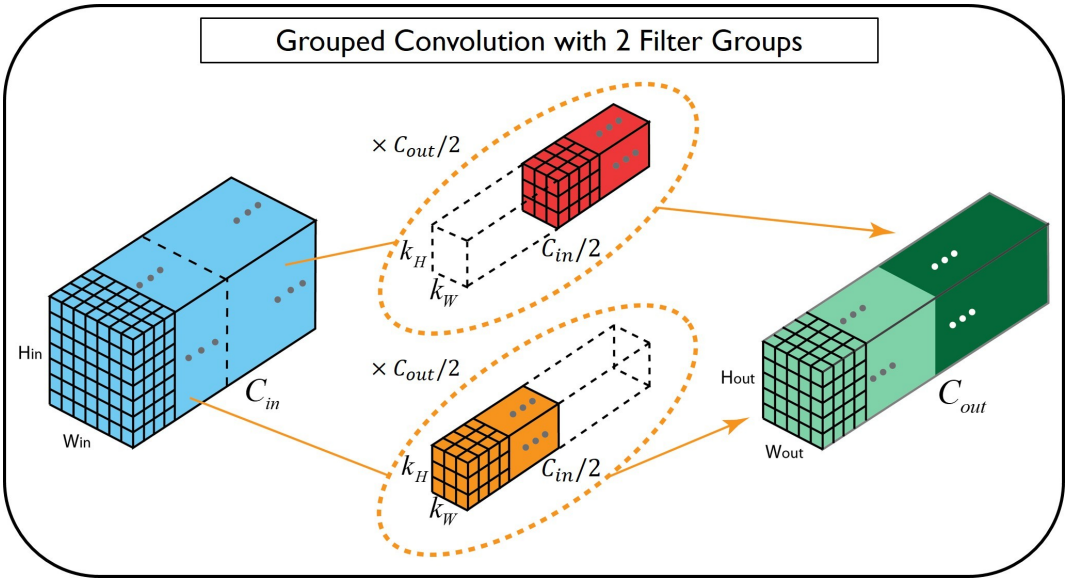


Figure 15.2 A demonstration of performing grouped convolution operation

In the example of Figure 15.2 you can see a demonstration of applying the grouped convolution method by two groups of filters. As you can see in Figure 15.2, in each group of filters, the number of channels per filter is $C_{in}/2$ (half of its normal value) (*i.e.*, the shape of each filter is as $(C_{in}/2, k_H, k_W)$). The number of filters in each group is also half of the normal value (*i.e.*, $C_{out}/2$). The filters in the first group (yellow filters and in the lower part of Figure 15.2) are convolved over the first part (first half) of the input data, and the filters in the second group (red filters in the upper part of Figure 15.2) are convolved over the second part (second half) of the input data. The output tensor of each filter groups is a tensor of shape $(C_{out}/2, H_{out}, W_{out})$. Finally, by putting the corresponding outputs of each group together, the final output of the operation, which is a tensor of shape $(C_{out}, H_{out}, W_{out})$, is obtained.

Figure 15.3 shows how to perform a 2D grouped convolution operation in general case. In this figure, it is assumed that there are groups of filters to perform the operation. Each group of filters has C_{out}/g filters of shape $(C_{in}/g, k_H, k_W)$. The filters in each group are convolved over C_{in}/g channels of input data to produce an output tensor of shape $(C_{out}/g, H_{out}, W_{out})$. Finally, by putting the corresponding output tensors of each group together, the final output tensor of the grouped convolution operation of shape $(C_{out}, H_{out}, W_{out})$ is obtained. Figure 15.4 also shows how to perform the 3D grouped convolution operation in general case. This operation is similar to the procedure of the 2D grouped convolution, except that in 3D case, one dimension will be added to the input data and filters.

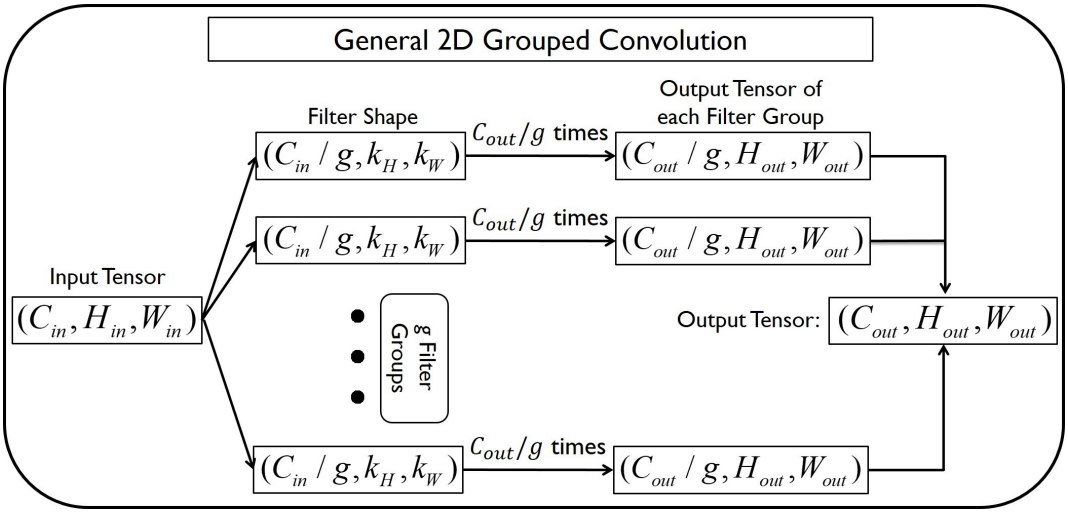


Figure 15.3 Demonstration of performing 2D grouped convolution operation in general case

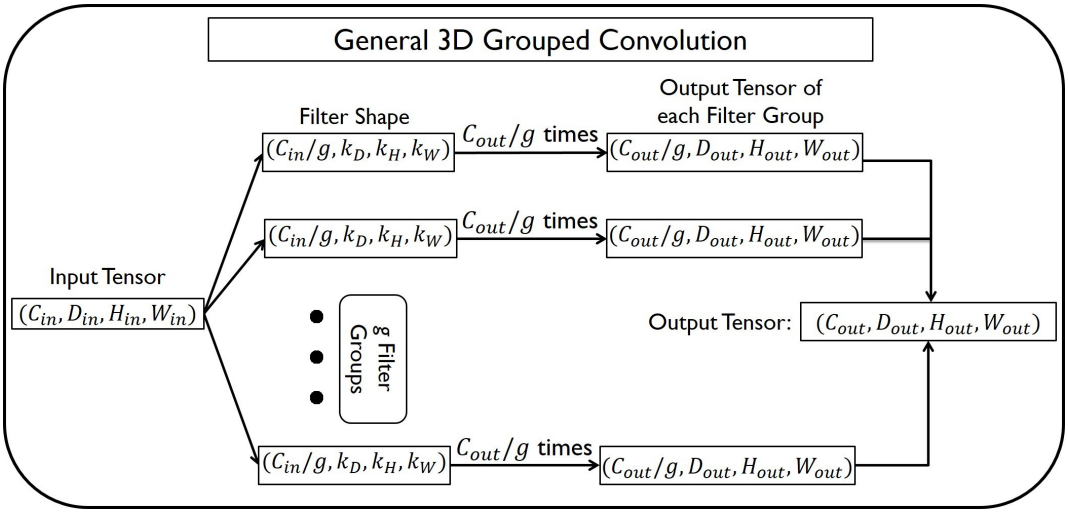


Figure 15.4 Demonstration of performing 3D grouped convolution operation in general case

Since the shape of each filter in a 2D grouped convolution operation is as $(C_{in}/g, k_H, k_W)$ with a bias value, so the total number of parameters to be trained in this method is as follows:

$$\underbrace{((C_{in}/g) \times k_H \times k_W) \times C_{out}}_{\#weights} + \underbrace{C_{out}}_{\#biases} \qquad (15.1)$$

Similarly, the total number of parameters to be trained in the 3D grouped convolution method is as follows:

$$\underbrace{((C_{in}/g) \times k_D \times k_H \times k_W) \times C_{out}}_{\#weights} + \underbrace{C_{out}}_{\#biases} \qquad (15.2)$$

The important point in the relations 15.1 and 15.2 is that the total number of parameters (weights and bias values) in the grouped convolution method is less than the total number of parameters of

the normal convolution method (The reduction rate is proportional to the number of groups, *i.e.*, the value of g parameter). This indicates that this method not only enables model parallelization but also reduces the number of model parameters.
Another interesting point about the grouped convolution method is its similarity to the first step of the depthwise separable convolution method. If the number of filter groups is equal to the number of input data channels (*i.e.*, $g = C_{in}$), then the shape of each filter will be as $(1, k_H, k_W)$, which will only be convolved on one channel of input data. Therefore, in this case, the grouped convolution method will partly resemble the first step of the depthwise separable convolution method. But it should be noted that in this case, in the grouped convolution method, the number of filters per group will be equal to $C_{out}/g = \lfloor C_{out}/C_{in} \rfloor$, and so the output tensor shape resulting from the convolution of each group of filters on the input data will be as $(\lfloor C_{out}/C_{in} \rfloor, H_{out}, W_{out})$. However, in the first step of the depthwise separable convolution method C_{in} filters have convolved on the input data, and the output tensor shape will be as $(C_{in}, H_{out}, W_{out})$.
So far, we are familiar with some of the benefits of the grouped convolution method. Now we want to summarize these advantages, and explain some of the other benefits of using this method. The first advantage of using the grouped convolution method is that the training phase of the model will be performed more efficiently. Since in the grouped convolution method the convolution operators are divided into different groups, and it is possible for each group or several groups of these convolutions to be trained on a single separate GPU, therefore, the model training phase takes place on multiple GPUs in parallel. This allows more data to be stored on each GPU, and overall the model is better trained, as well as the model weights converge faster to their optimum values. This advantage of the grouped convolution method has been widely used in recent years in the implementation of very deep models and networks. We have already mentioned the second advantage of using the grouped convolution method, and here we will briefly mention it again. As mentioned earlier, the use of the grouped convolution method reduces the number of model parameters, and thus reduces the model size. The amount of reduction in the number of parameters per convolutional layer also depend on the number of filter groups. In other words, if the number of filter groups in a convolutional layer is equal to g, then the number of parameters in this layer will be reduced by $1/g$ compared to that of the normal convolution operator.
A third advantage of using the grouped convolution method is that it may increase the final accuracy of the model compared to that of using the normal convolution operator! Note that the number of model parameters in this case is also reduced. There are various reasons for this amazing matter, and in general case, we expect that as the number of model parameters decreases, so the final accuracy of the model will be reduced. But some experiments have shown that if the number of filter groups is correctly selected, then the final accuracy of the model will not only decrease but also increase. One of the important reasons of this phenomenon is that when a normal convolutional layer is implemented via the grouped convolution method, then each of the different groups (of filters) represents and recognizes a particular type of feature and representation of the input data (*i.e.*, each group learns a specific representation of the input data). That is to say, it can be assumed that each group has the task of identifying and representing a particular range of features at that level; and in other words, a *learning distributed representation* mode is formed. It should be noted that, it is not always possible to improve the performance and the final accuracy of the model by increasing the number of filter groups. Because increasing the number of groups also makes each group only

convolves on a small portion of the input data, and this will eventually damage the local connectivity attribute of the convolution operator.
According to the above-mentioned notes, it is well understood that the grouped convolution method can be a suitable method for model parallelization while reducing the model size, and while at the same time either increasing the final model accuracy or slightly reduce its accuracy. As it probably has reached into the mind of the reader, the research area of the grouped convolution method still has a lot of work to do, and there are important underlying questions that have not been answered properly. Some of these questions are as follows:
1- What is the appropriate value for the number of filter groups, and on what criteria is it chosen to reach the maximum accuracy of the model?
2- Can filter groups overlap with each other? In other words, part of the input data on which the filters are to be convolved over it, is shared between several groups.
3- Should the size and dimensions of the filters of all groups be the same? And should each group of filters apply on the same portion of input data (in terms of size and dimension)?
For example, another strategy could be that half of the input data belongs to the first group, $1/4$ of the input data belongs to the second group, $1/8$ of the input data belongs to the third group, and so on. This model of grouped convolution is called *Convolution with Logarithmic Filter Groups*. In any case, working on each of these questions can lead to new valuable achievements in the field of deep learning that can extend the edge of technology in this field. At the end of this chapter, we will briefly investigate two specific types of grouped convolution methods.

15.1 Shuffled grouped convolution

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

15.2 Pointwise grouped convolution

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

Chapter 16: Deformable Convolution

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.

Chapter 17: Representation Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/convolutions-for-deep-learning.