Yusuke Uchida Follow
Apr 23 · 8 min read

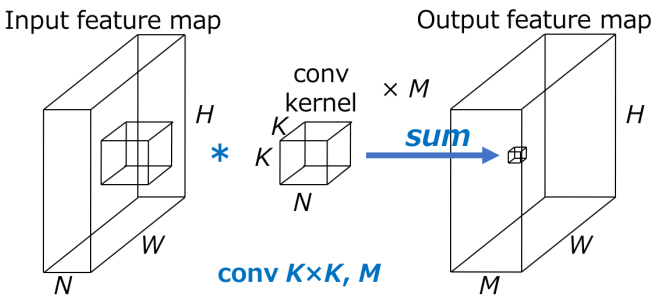
Why MobileNet and Its Variants (e.g. ShuffleNet) Are Fast

Introduction

In this article, I give an overview of building blocks used in efficient CNN models like MobileNet and its variants, and explain why they are so efficient. In particular, I provide intuitive illustrations about how convolution in both spatial and channel domain is done.

Building Blocks Used in Efficient Models

Before explaining specific efficient CNN models, let's check computational cost of building blocks used in efficient CNN models, and see how convolution are performed in spatial and channel domain.



Letting HxW denotes the spatial size of input feature map, N denotes the number of input channels, KxK denotes the size of convolutional kernel, and M denotes the number of output channels, the computational cost of standard convolution becomes $HWNK^2M$.

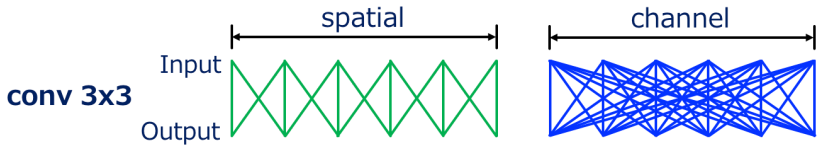
An important point here is that the computational cost of the standard convolution is proportional to (1) the spatial size of feature map HxW, (2) the size of convolution kernel K^2, (3) the numbers of input and output channels NxM.

The above computational cost is required when convolution is performed on both spatial and channel domain. CNNs can be speeded up by factorizing this convolution as shown below.

Convolution

First of all, I provide an intuitive illustration about how convolution in both spatial and channel domain is done for standard convolution, whose computational cost is $HWNK^2M$.

I connect lines between input and output to visualize dependency between input and output. The number of lines roughly indicate the computational cost of convolution in spatial and channel domain respectively.



For instance, conv3x3, the most commonly used convolution, can be visualized as shown above. We can see that the input and output are locally connected in spatial domain while in channel domain, they are fully connected.

Next, conv1x1 [1], or pointwise convolution, which is used to change the size of channels, is visualized above. The computational cost of this convolution is HWNM because the size of kernel is 1x1, resulting in 1/9 reduction in computational cost compared with conv3x3. This convolution is used in order to “blend” information among channels.

Grouped Convolution

Grouped convolution is a variant of convolution where the channels of the input feature map are grouped and convolution is performed independently for each grouped channels.

Letting G denote the number of groups, the computational cost of grouped convolution is $HWNK^2M/G$, resulting in 1/G reduction in computational cost compared with standard convolution.

The case of grouped conv3x3 with G=2. We can see that the number of connections in channel domain becomes smaller than standard convolution, which indicates smaller computational cost.

The case of grouped conv3x3 with G=3. The connections become more sparse.

The case of grouped conv1x1 with G=2. Thus, conv1x1 can also be grouped. This type of convolution is used in ShuffleNet.

The case of grouped conv1x1 with G=3.

Depthwise Convolution

In depthwise convolution [2,3,4], convolution is performed independently for each of input channels. It can also be defined as a special case of grouped convolution where the numbers of input and output channels are same and G equals the number of channels.

As shown above, depthwise convolution significantly reduces the computational cost by omitting convolution in channel domain.

Channel Shuffle

Channel shuffle is an operation (layer) which changes the order of the channels used in ShuffleNet [5]. This operation is implemented by tensor reshape and transpose.

More precisely, letting GN' (=N) denote the number of input channels, the input channel dimension is first reshaped into (G, N'), then

transpose (G, N') into (N', G), and finally flatten into the same shape as input. Here, G represents the number of groups for grouped convolution, which is used together with channel shuffle layer in ShuffleNet.

While the computational cost of channel shuffle can not be defined in terms of the number of multiply-add operations (MACs), there should be some overhead.

The case of channel shuffle with G=2. Convolution is not performed, and simply the order of the channels is changed.

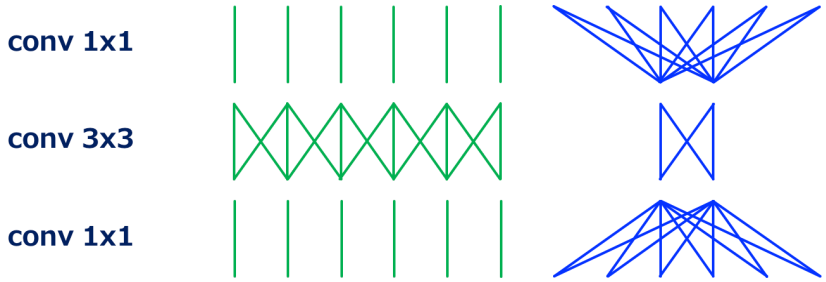
The case of channel shuffle with G=3.

Efficient Models

In the following, for efficient CNN models, I provide intuitive illustrations about why they are efficient and how convolution in both spatial and channel domain is done.

ResNet (Bottleneck Version)

Residual unit with bottleneck architecture used in ResNet [6] is a good start point for further comparison with the other models.

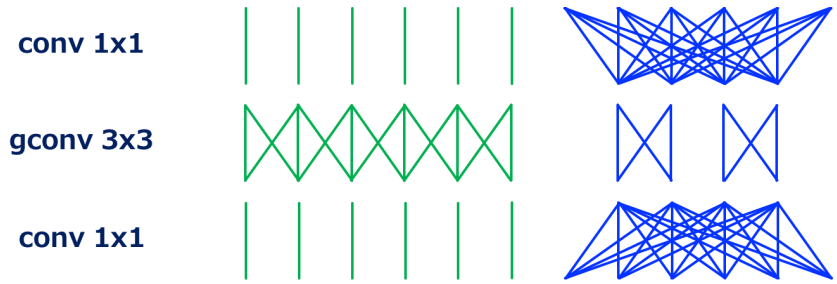


As shown above, a residual unit with bottleneck architecture is composed of conv1x1, conv3x3, and conv1x1. The first conv1x1 reduces the dimension of the input channel, reducing the

computational cost of subsequent relatively expensive conv3x3. The final conv1x1 recover the dimension of the output channel.

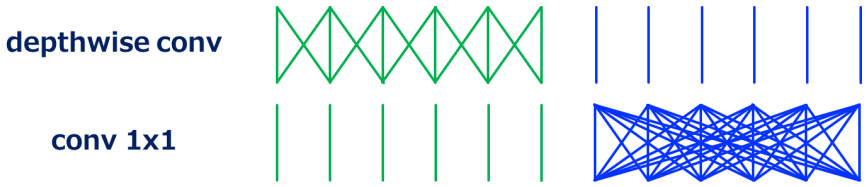
ResNeXt

ResNeXt [7] is an efficient CNN model, which can be seen as a special case of ResNet whose conv3x3 is replaced by grouped conv3x3. By using efficient grouped conv, the channel reduction rate in conv1x1 becomes moderate compared with ResNet, resulting in better accuracy with the same computational cost.



MobileNet (Separable Conv)

MobileNet [8] is a stack of the separable convolution modules which are composed of depthwise conv and conv1x1 (pointwise conv).

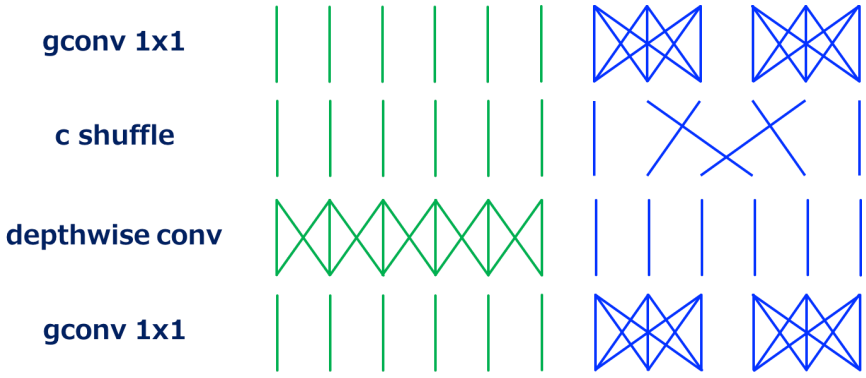


The separable conv independently performs convolution in spatial and channel domains. This factorization of convolution significantly reduces the computational cost from $HWNK^2M$ to $HWNK^2$ (depthwise) + $HWNM$ (conv1x1), $HWN(K^2 + M)$ in total. In general, $M >> K^2$ (e.g. $K=3$ and $M \geq 32$), the reduction rate is roughly 1/8–1/9.

The important point here is that the bottleneck of the computational cost is now conv1x1!

ShuffleNet

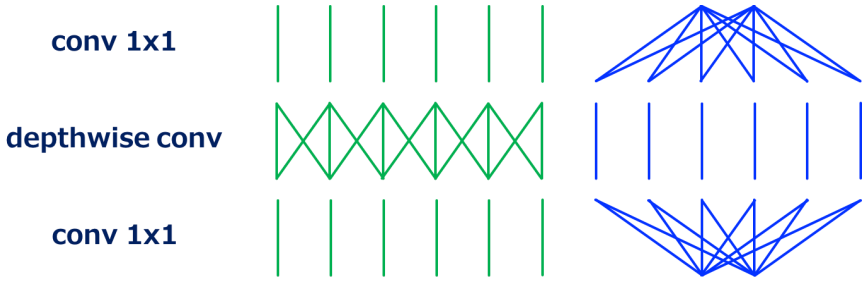
The motivation of ShuffleNet is the fact that conv1x1 is the bottleneck of separable conv as mentioned above. While conv1x1 is already efficient and there seems to be no room for improvement, grouped conv1x1 can be used for this purpose!



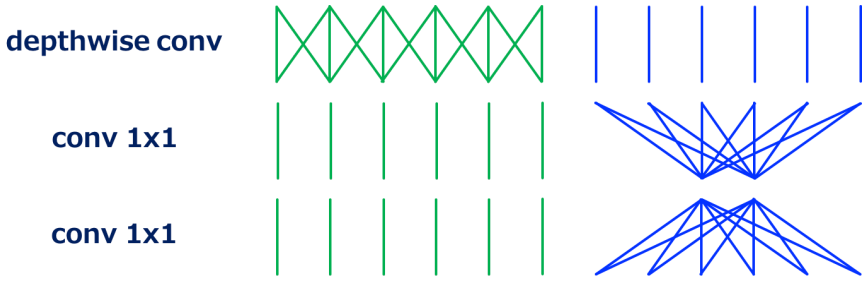
The above figure illustrates the module for ShuffleNet. The important building block here is the channel shuffle layer which “shuffles” the order of the channels among groups in grouped convolution. Without channel shuffle, the outputs of grouped convolutions are never exploited among groups, resulting in the degradation of accuracy.

MobileNet-v2

MobileNet-v2 [9] utilizes a module architecture similar to the residual unit with bottleneck architecture of ResNet; the modified version of the residual unit where conv3x3 is replaced by depthwise convolution.



As you can see from the above, contrary to the standard bottleneck architecture, the first conv1x1 increases the channel dimension, then depthwise conv is performed, and finally the last conv1x1 decreases the channel dimension.



By reordering the building blocks as above and comparing it with MobileNet-v1 (separable conv), we can see how this architecture works

(this reordering does not change the overall model architecture because the MobileNet-v2 is the stack of this module).

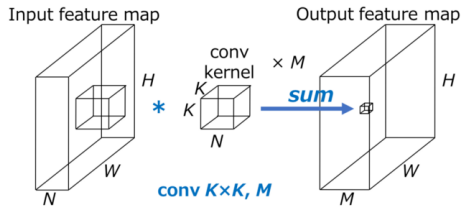
That is to say, the above module be regarded as a modified version of separable conv where the single conv1x1 in separable conv is factorized into two conv1x1s. Letting T denote an expansion factor of channel dimension, the computational cost of two conv1x1s is $2HWN^2/T$ while that of conv1x1 in separable conv is HWN^2. In [5], $T = 6$ is used, reducing the computational cost for conv1x1 by a factor of 3 ($T/2$ in general).

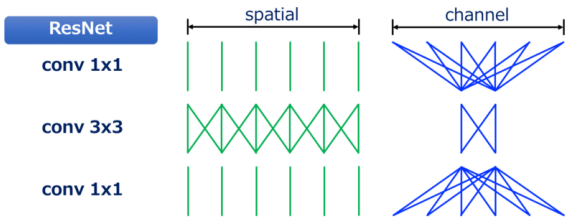
FD-MobileNet

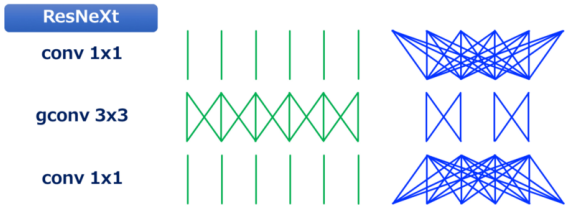
Finally, I introduce Fast-Downsampling MobileNet (FD-MobileNet) [10]. In this model, downsamplings are performed in earlier layers compared with MobileNet. This simple trick can reduce total computational cost. The reason lies in the traditional downsampling strategy and the computational cost of separable conv.

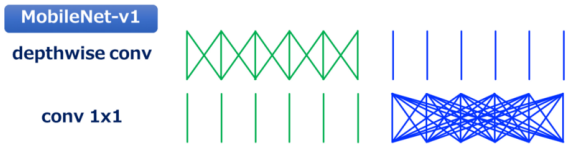
Starting with VGGNet, many models adopt the same downsampling strategy: perform downsampling and then double the number of channels of subsequent layers. For standard convolution, the computational cost does not change after downsampling because it is defined by $HWNK^2M$. However, for separable conv, its computational cost becomes smaller after downsampling; it is reduced from $HWN(K^2 + M)$ to $H/2\ W/2\ 2N(K^2 + 2M) = HWN(K^2/2 + M)$. This is relatively dominant when M is not so large (i.e. earlier layers).

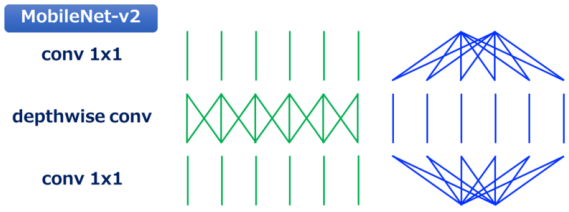
I end up this article with the following cheat sheet, thank you :P

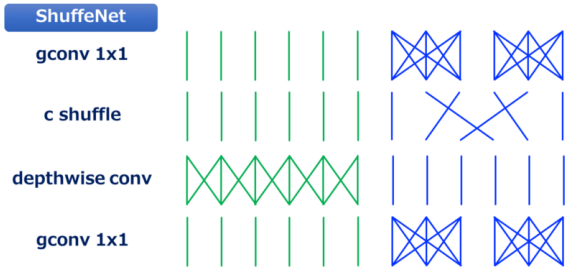


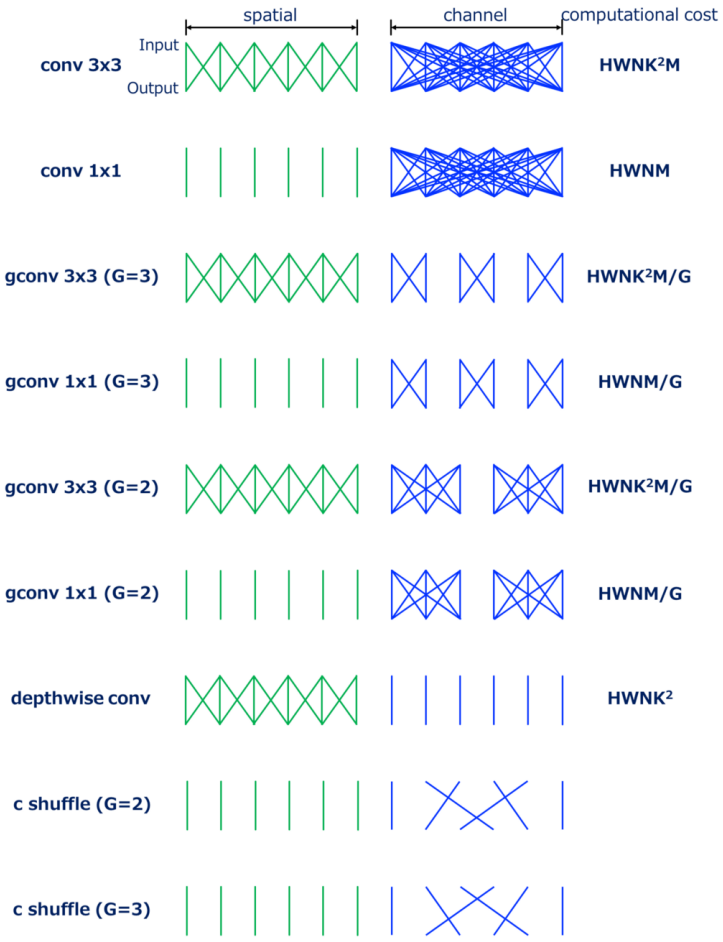












References

[1] M. Lin, Q. Chen, and S. Yan, “Network in Network,” in Proc. of ICLR, 2014.

[2] L. Sifre, “Rigid-motion Scattering for Image Classification, Ph.D. thesis, 2014.

[3] L. Sifre and S. Mallat, “Rotation, Scaling and Deformation Invariant Scattering for Texture Discrimination,” in Proc. of CVPR, 2013.

[4] F. Chollet, “Xception: Deep Learning with Depthwise Separable Convolutions,” in Proc. of CVPR, 2017.

[5] X. Zhang, X. Zhou, M. Lin, and J. Sun, “ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices,” in arXiv:1707.01083, 2017.

[6] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in Proc. of CVPR, 2016.

[7] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, “Aggregated Residual Transformations for Deep Neural Networks,” in Proc. of CVPR, 2017.

[8] 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,” in arXiv:1704.04861, 2017.

[9] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” in arXiv:1801.04381v3, 2018.

[10] Z. Qin, Z. Zhang, X. Chen, and Y. Peng, “FD-MobileNet: Improved MobileNet with a Fast Downsampling Strategy,” in arXiv:1802.03750, 2018.