背景

因为之前的一些 SOTA 网络中运用了大量的 1×1 卷积操作，例如图 1 所示的在 MobileNetV1 中，1×1 卷积占总计算量的 94.86%，占总参数量的 74.59%，可见 1×1 卷积导致计算量很大，这样的网络效率不高，本文提出了逐点分组卷积来减少 1×1 卷积带来的计算量。

Type	Mult-Adds	Parameters
Conv 1×1	94.86%	74.59%
Conv DW 3×3	3.06%	1.06%
Conv 3×3	1.19%	0.02%
Fully Connected	0.18%	24.33%

图 1

传统的分组卷积（如图 2 所示）存在的问题是，每一组卷积仅仅对输入通道的一部分进行特征映射，这就导致了每一组的输出仅与其输入有关，不同组之间没有任何的信息传递，导致最后学习的特征非常局限，因此提出了通道重排操作以此来增加特征表达能力。

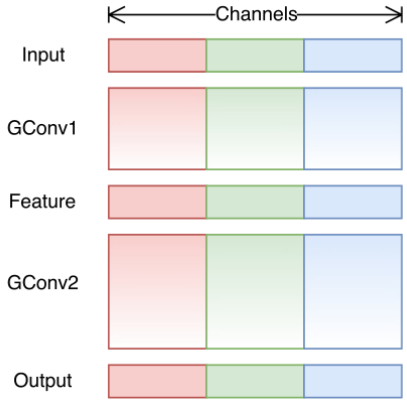

图 2

方法

①逐点分组卷积

分组卷积，就是将原始的特征图分成几组后再分别对每一组进行卷积，逐点

分组卷积就是将分组卷积核逐点卷积结合起来。

为什么要采用分组卷积？

假设输入特征图大小为 H×W×C，卷积核尺寸为 K×K，c 个卷积核，输出特征图尺寸为 h×w×c。

那么使用普通卷积参数量为：

$$K \times K \times C \times c = cCK^2 \tag{1}$$

计算量为：

$$K \times K \times C \times h \times w \times c = hwcCK^2 \tag{2}$$

使用分组卷积，假设分 4 组，参数量为：

$$K \times K \times \left(\frac{C}{4}\right) \times \left(\frac{c}{4}\right) \times 4 = \frac{cCK^2}{4} \tag{3}$$

计算量为：

$$K \times K \times \left(\frac{C}{4}\right) \times h \times w \times \left(\frac{c}{4}\right) \times 4 = \frac{hwcCK^2}{4} \tag{4}$$

通过对比(1)、(3)以及(2)、(4)可以发现分组卷积的参数量和计算量都比普通卷积要小。

②通道重排

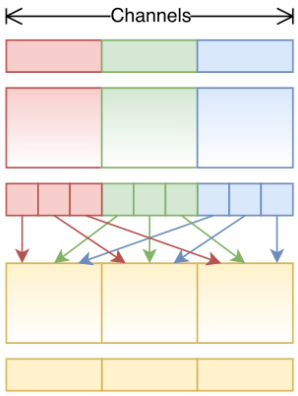

图 3

如图 3 所示，通道重排就是从分组卷积后得到的特征图中提取不同组的特征通道，然后将它们重新组合，其过程如下图所示（图片来源于博客）：

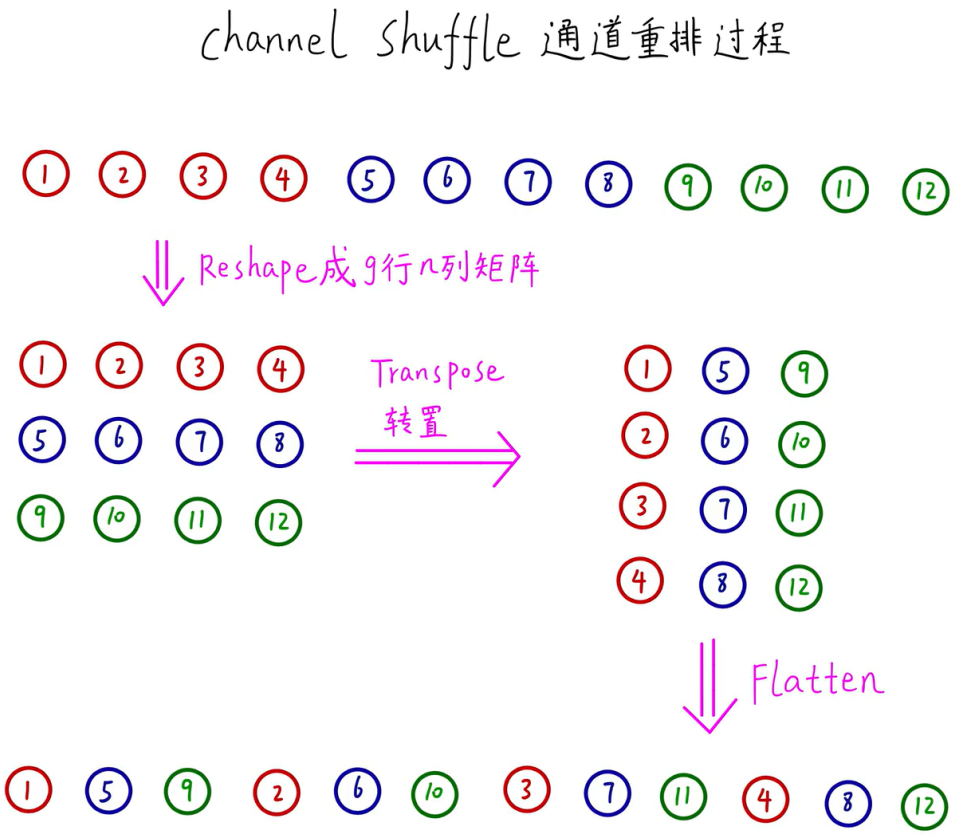
channel shuffle 通道重排过程
Reshape成g行n列矩阵
Transpose
转置
Flatten

图 4

③单元结构

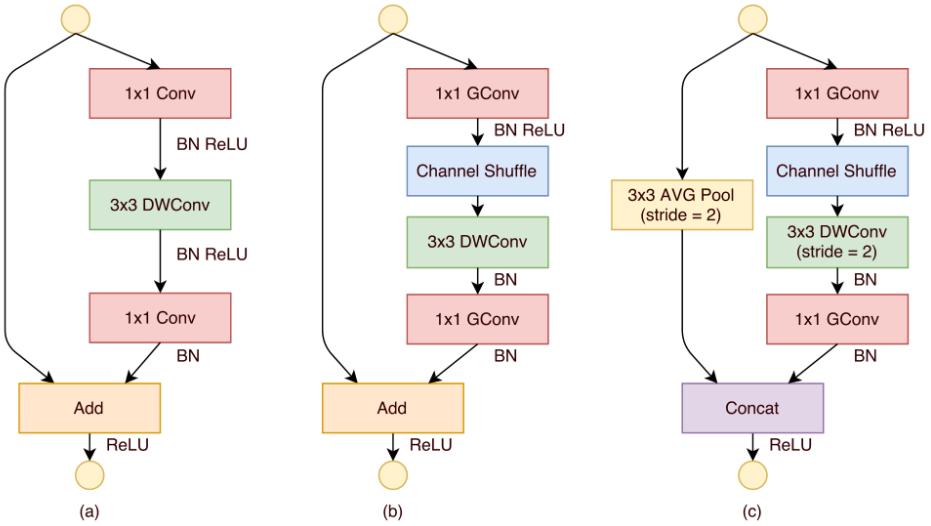
1x1 Conv
BN ReLU
3x3 DWConv
BN ReLU
1x1 Conv
BN
Add
ReLU
(a)
1x1 GConv
BN ReLU
Channel Shuffle
3x3 DWConv
BN
1x1 GConv
BN
Add
ReLU
(b)
3x3 AVG Pool
(stride = 2)
1x1 GConv
BN ReLU
Channel Shuffle
3x3 DWConv
(stride = 2)
BN
1x1 GConv
BN
Concat
ReLU
(c)

图 5

从图 5 可以看出 ShuffleNet 的单元结构的演变过程。图 5(a)是 ResNet 中的瓶颈结构，然后使用逐点分组卷积将瓶颈结构中第一层的 1×1 卷积层代替掉，然后接了一层通道重排操作，中间的 3×3 深度方向的卷积操作保留，但是，注意，这里深度方向卷积后并没有使用 ReLU 激活函数，猜测可能的原因正如 MobileNetV2 中提到（图 6 所示），低维时使用 ReLU 会造成信息丢失，所以 MobileNetV2 采取的措施是倒残差结构，先将维度升高，避免信息丢失，而在这里是直接取消 ReLU 激活函数。

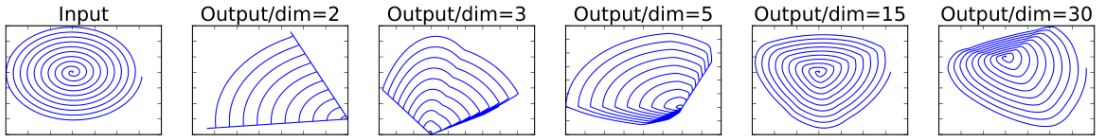

图 6

随后又将最后一层的 1×1 卷积替换为逐点分组卷积，注意，在这一层逐点分组卷积后没有使用通道重排操作是因为此时的效果已经很可观了。

那么图 5(c)的结构和(b)的主要区别就是在跳跃连接分支中采用了 3×3，步长为 2 的平均池化、主分支中深度方向的卷积步长变成了 2，并且最后使用的是 concat 拼接而不是逐像素相加，这样的结构会使输入图像尺寸减半，通道数翻倍。

疑问

对单元结构中最后将 add 操作换为 concat 拼接有一些疑问。

自己在查资料的过程中发现

add 是描述图像的特征下的信息量增多了，但是描述图像的维度本身并没有增加，只是每一维下的信息量在增加，这显然是对最终的图像的分类是有益的。而 concatenate 是通道数的合并，也就是说描述图像本身的特征数（通道数）增加了，而每一特征下的信息是没有增加，并且 add 的计算量要比 concat 的计算量小得多（内容来源）

那么为什么偏偏要增加通道数呢？每个特征下的信息又没有增加，并且相比 add 来说计算量还增加了，这样做是为什么呢？