11.ShuffleNet

2018年提出，论文地址[ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices](), [ShuffeNet V2: Practical Guidelines for Effcient CNN Architecture Design]()

卷积的group操作从AlexNet就已经有了，当时主要是解决模型在双GPU上的训练。ResNeXt借鉴了这种group操作改进了原本的ResNet。MobileNet则是采用了depthwise separable convolution代替传统的卷积操作，在几乎不影响准确率的前提下大大降低计算量，Xception主要也是采用depthwise separable convolution改进Inception v3的结构。

ShuffleNet主要采用channel shuffle、pointwise group convolutions和depthwise separable convolution来修改原来的ResNet单元，接下来依次讲解。

11.1 ShuffleNet V1

Channel Shuffle

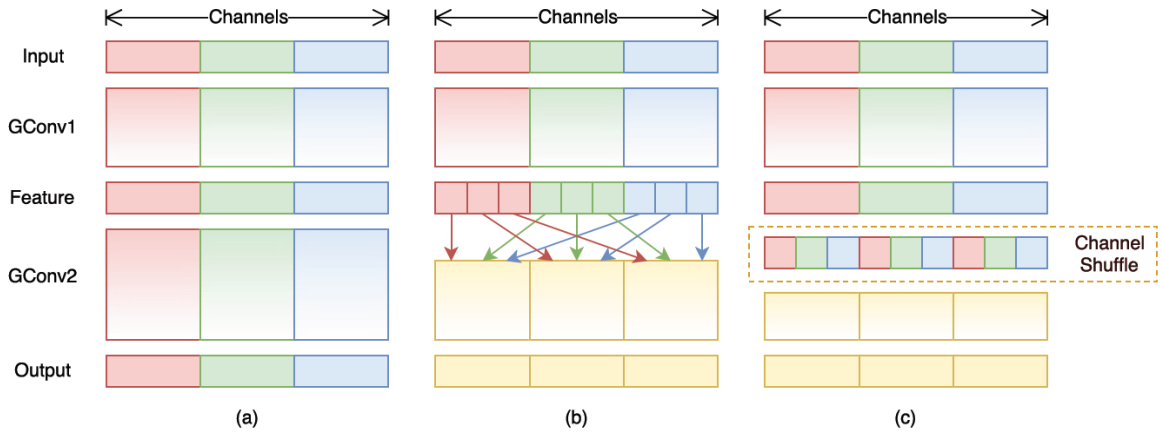

图38 Channel Shuffle

借助group思想，这种操作可以大大减少计算量，因为你每个filter不再是和输入的全部feature map做卷积，而是和一个group的feature map做卷积。但是如果多个group操作叠加在一起，如（a）的两个卷积层都有group操作，显然就会产生边界效应，什么意思呢？就是某个输出channel仅仅来自输入channel的一小部分。这样肯定是不行的的，学出来的特征会非常局限。于是就有了channel shuffle来解决这个问题，先看（b），在

进行GConv2之前，对其输入feature map做一个分配，也就是每个group分成几个subgroup，然后将不同group的subgroup作为GConv2的一个group的输入，使得GConv2的每一个group都能卷积输入的所有group的feature map，这和（c）的channel shuffle的思想是一样的。

pointwise group convolutions

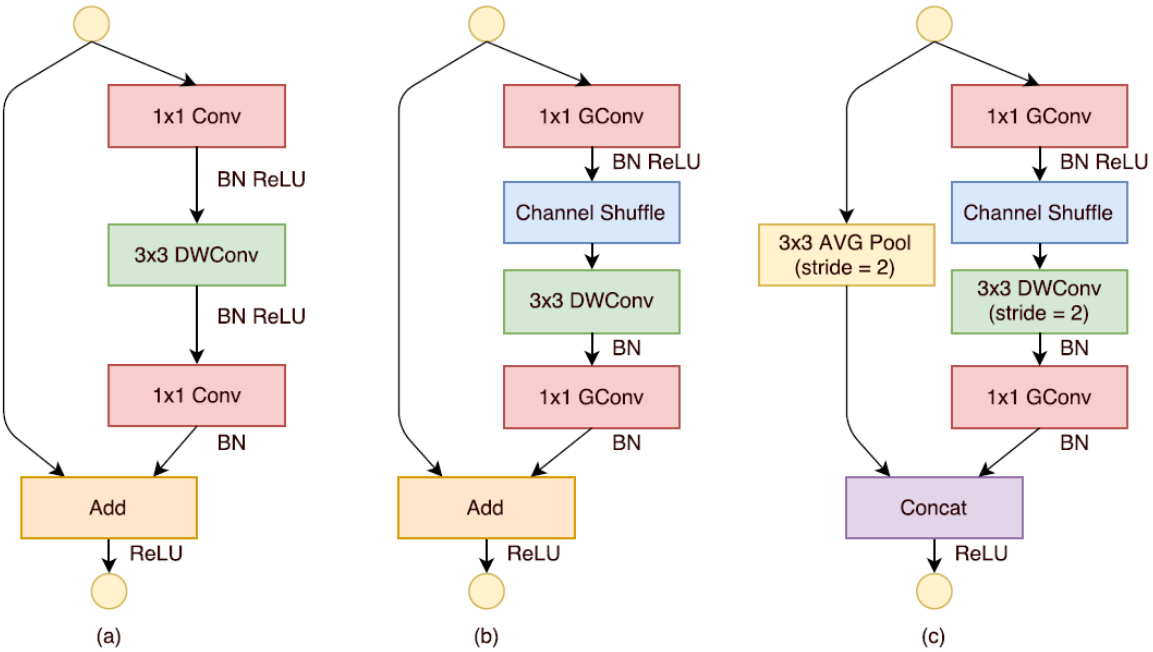

图39 ShuffleNet Uints

pointwise group convolutions，其实就是带group的卷积核为1×1的卷积，也就是说pointwise convolution是卷积核为1×1的卷积。在ResNeXt中主要是对3×3的卷积做group操作，但是在ShuffleNet中，作者是对1×1的卷积做group的操作，因为作者认为1×1的卷积操作的计算量不可忽视。可以看（b）中的第一个1×1卷积是GConv，表示group convolution。（a）是ResNet中的bottleneck unit，不过将原来的3×3 Conv改成3×3DWConv，作者的ShuffleNet主要也是在这基础上做改动。首先用带group的1×1卷积代替原来的1×1卷积，同时跟一个channel shuffle操作，这个前面也介绍过了。然后是3*3 DWConv表示depthwise separable convolution。Figur（c）添加了一个Average pooling和设置了stride=2，另外原来Resnet最后是一个Add操作，也就是元素值相加，而在（c）中是采用concat的操作，也就是按channel合并，类似googleNet的Inception操作。

Layer	Output size	KSize	Stride	Repeat	Output channels (g groups)				
					$g=1$	$g=2$	$g=3$	$g=4$	$g=8$
Image	224×224				3	3	3	3	3
Conv1	112×112	3×3	2	1	24	24	24	24	24
MaxPool	56×56	3×3	2						
Stage2	28×28		2	1	144	200	240	272	384
	28×28		1	3	144	200	240	272	384
Stage3	14×14		2	1	288	400	480	544	768
	14×14		1	7	288	400	480	544	768
Stage4	7×7		2	1	576	800	960	1088	1536
	7×7		1	3	576	800	960	1088	1536
GlobalPool	1×1	7×7							
FC					1000	1000	1000	1000	1000
Complexity					143M	140M	137M	133M	137M

图40 ShuffleNet V1网络结构

跟ResNet类似，用ShuffleNet Uint代替Residual Block。

11.2 ShuffleNet V2

作者在特定的平台下研究ShuffleNetv1和MobileNetv2的运行时间，并结合理论与实验得到了4条实用的指导原则：

G1.同等通道大小最小化内存访问量；

G2.过量使用组卷积会增加MAC；

G3.网络碎片化会降低并行度；

G4.不能忽略元素级操作。

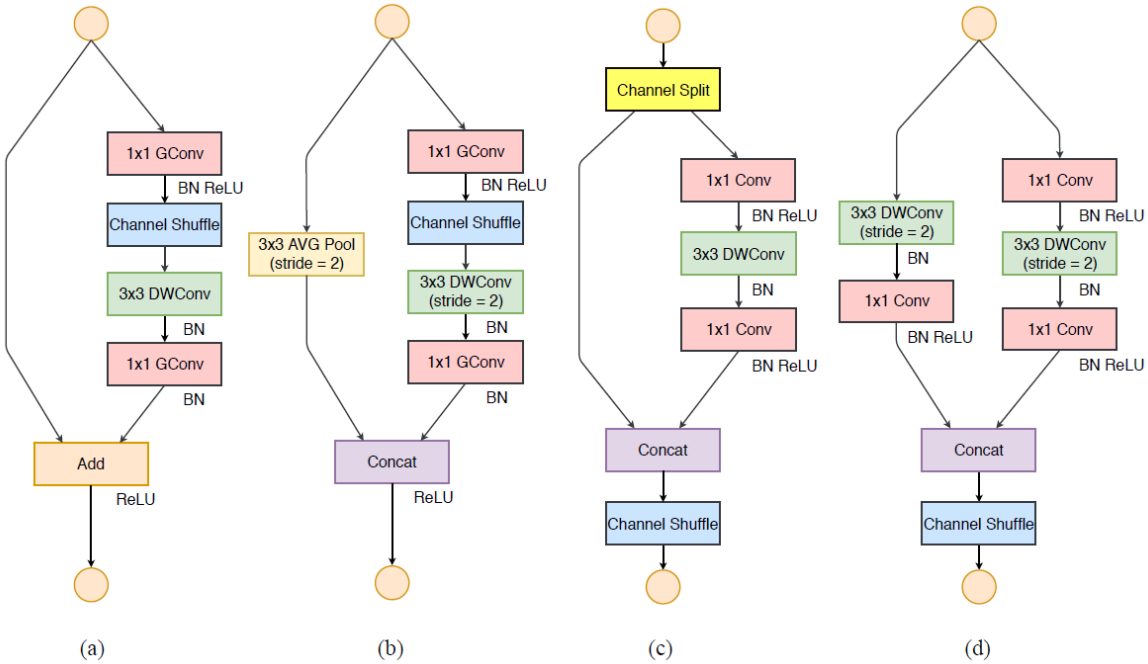

图42 ShuffleNet V2 Uints

在ShuffleNetv1的模块中，大量使用了1x1组卷积，这违背了G2原则，另外v1采用了类似ResNet中的瓶颈层（bottleneck layer），输入和输出通道数不同，这违背了G1原则。同时使用过多的组，也违背了G3原则。短路连接中存在大量的元素级Add运算，这违背了G4原则。

为了改善v1的缺陷，v2版本引入了一种新的运算：channel split。具体来说，在开始时先将输入特征图在通道维度分成两个分支 。左边分支做同等映射，右边的分支包含3个连续的卷积，并且输入和输出通道相同，这符合G1。而且两个1x1卷积不再是组卷积，这符合G2，另外两个分支相当于已经分成两组。两个分支的输出不再是Add元素，而是concat在一起，紧接着是对两个分支concat结果进行channle shuffle，以保证两个分支信息交流。其实concat和channel shuffle可以和下一个模块单元的channel split合成一个元素级运算，这符合原则G4。

对于下采样模块，不再有channel split，而是每个分支都是直接copy一份输入，每个分支都有stride=2的下采样，最后concat在一起后，特征图空间大小减半，但是通道数翻倍。

Layer	Output size	KSize	Stride	Repeat	Output channels			
					0.5×	1×	1.5×	2×
Image	224×224				3	3	3	3
Conv1	112×112	3×3	2	1	24	24	24	24
MaxPool	56×56	3×3	2					
Stage2	28×28		2	1	48	116	176	244
	28×28		1	3				
Stage3	14×14		2	1	96	232	352	488
	14×14		1	7				
Stage4	7×7		2	1	192	464	704	976
	7×7		1	3				
Conv5	7×7	1×1	1	1	1024	1024	1024	2048
GlobalPool	1×1	7×7						
FC					1000	1000	1000	1000
FLOPs					41M	146M	299M	591M
# of Weights					1.4M	2.3M	3.5M	7.4M

图43 ShuffleNet V2 网络结构

pytorch实现