1. 参数量和计算量

1.1 定义

参数数量params：关系到模型大小，单位通常为M，通常参数用float32表示，所以模型大小是参数数量的4倍

计算量FLOPs:是floating point operations的缩写，可以用来衡量算法/模型的复杂度，这关系到算法速度，大模型的单位通常为G，小模型单位通常为M；计算量一般只考虑乘加操作(Multi-Adds)的数量，而且只考虑conv和fc等参数层的计算量，忽略BN和relu等。

1.2 计算公式

设卷积核大小为k*k，输入通道数为Cin,输出通道数为Cout,输出特征图的大小为HxW，忽略偏置项

conv标准卷积层：

params:KxKxCinxCout

flops:kxKxCinxCoutxHxW

FC全连接层（相当于k=1）

2.MobileNet v1

2.1 参数量对比

将标准卷积分解为深度可分离卷积和点卷积，深度卷积将每个卷积核应用到每一个通道，1x1卷积再组合通道卷积的输出。

参数量的对比：

我们来算一算，假设输入通道数为3，要求输出通道数为256，两种做法：

1. 直接接一个3×3×256的卷积核，参数量为：3×3×3×256 = 6,912

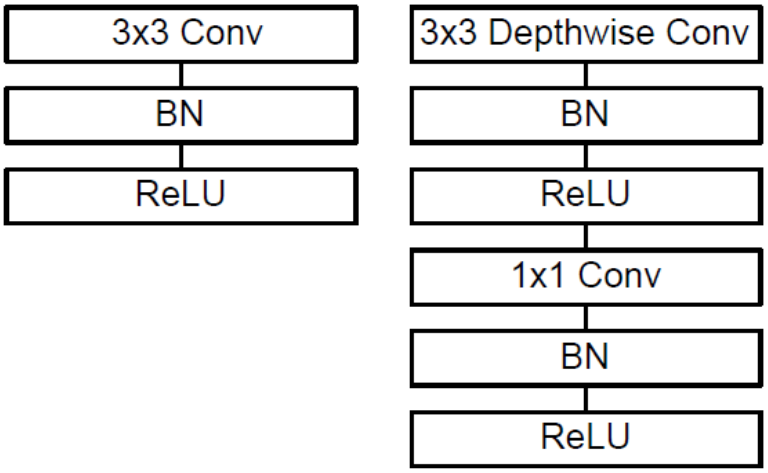
2. DW操作，分两步完成，参数量为：3×3×3 + 3×1×1×256 = 795，又把参数量降低到九分之一！

2.2 网络结构

v1涉及到了group卷积。

group卷积是：比如32个通道，我分成2组，则将每一个卷积核按照通道分为2组(每组的通道数=32/2= 16)，第1组卷积核对第1组通道进行卷积 ，第2组卷积核对第2组通道进行卷积。然后concat。

传统的3D卷积常见的使用方式如下图左侧所示，deep-wise卷积的使用方式如下图右边所示。



2.3 引入的两个超参数

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

分辨率因子β的取值范围在(0,1]之间，是作用于每一个模块输入尺寸的约减因子，简单来说就是将输入数据以及由此在每一个模块产生的特征图都变小了，结合宽度因子α，deep-wise结合1x1方式的卷积核计算量为：

$$D_k \times D_k \times \alpha M \times \beta D_F \times \beta D_F + \alpha N \times \alpha M \times \beta D_F \times \beta D_F$$

下图为使用不同的β系数作用于标准MobileNet时，对精度和计算量以的影响（α固定）

Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

要注意再使用宽度和分辨率参数调整网络结构之后，都要从随机初始化重新训练才能得到新网络。

2.MobileNer v2

2.1 改进点

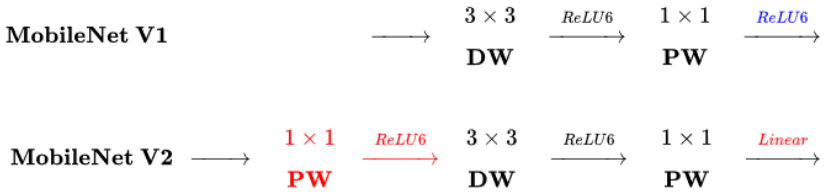
(1) Inverted residuals：

我们正常的残差块，都是先降通道数，再升，最后进行element_wise操作。v2则是先升后降，因为如果要做depth_wise，通道被压缩后提取的特征也会变少，所以先expand，再depth_wise。

（2）Linear Bottlenecks

论文中证明了relu对不同维度输入的信息丢失对比。当把原始维度增加到15或30后再作为relu的输入，输出恢复到原始维度后不会丢失太多的输入信息；相比之下，如果原始维度只增加到2或3后再作为relu的输入，输出恢复到原始维度后信息丢失较多。所以执行降维后的卷积层后面不会接类似relu这样的非线性激活函数，也就是linear bottleneck的含义。

由于relu会将小于0的数置为0，但这些信息也许有用，所以在point_wise操作后，弃用非线性的激活函数。因此这块point_wise操作称为linear，好记。



> 相同点

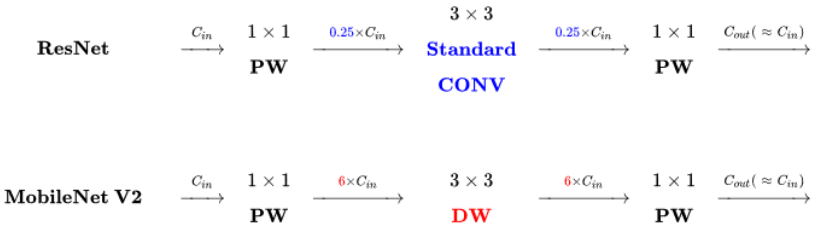
- **都采用 Depth-wise (DW) 卷积搭配 Point-wise (PW) 卷积的方式来提特征。**这两个操作合起来也被称为 Depth-wise Separable Convolution，之前在 Xception 中被广泛使用。这么做的好处是理论上可以成倍的减少卷积层的时间复杂度和空间复杂度。由下式可知，因为卷积核的尺寸 K 通常远小于输出通道数 C_{out}，因此标准卷积的计算复杂度近似为 DW + PW 组合卷积的 K^2 倍。

$$\text{Complexity} \quad \frac{\text{Depth-wise Separable CONV}}{\text{Standard CONV}} = \frac{1}{K^2} + \frac{1}{C_{out}} \sim \frac{1}{K^2}$$

> 不同点：Linear Bottleneck

- **V2 在 DW 卷积之前新加了一个 PW 卷积。**这么做的原因，是因为 DW 卷积由于本身的计算特性决定它自己没有改变通道数的能力，上一层给它多少通道，它就只能输出多少通道。所以如果上一层给的通道数本身很少的话，DW 也只能很委屈的在低维空间提特征，因此效果不够好。现在 V2 为了改善这个问题，给每个 DW 之前都配备了一个 PW，专门用来升维，定义升维系数 $t = 6$，这样不管输入通道数 C_{in} 是多是少，经过第一个 PW 升维之后，DW 都是在相对的更高维（$t \cdot C_{in}$）进行着辛勤工作的。
- **V2 去掉了第二个 PW 的激活函数。**论文作者称其为 Linear Bottleneck。这么做的原因，是因为作者认为激活函数在高维空间能够有效的增加非线性，而在低维空间时则会破坏特征，不如线性的效果好。由于第二个 PW 的主要功能就是降维，因此按照上面的理论，降维之后就不宜再使用 ReLU6 了。

3、和 ResNet 的区别



> 相同点

- MobileNet V2 借鉴 ResNet，都采用了 $1\times1\rightarrow3\times3\rightarrow1\times1$ 的模式。
- MobileNet V2 借鉴 ResNet，同样使用 Shortcut 将输出与输入相加（未在上式画出）

> 不同点：Inverted Residual Block

- ResNet 使用 **标准卷积** 提特征，MobileNet 始终使用 **DW卷积** 提特征。
- ResNet **先降维** (0.25倍)、卷积、再升维。而 MobileNet V2 则是 **先升维** (6倍)、卷积、再降维。直观的形象上来看，ResNet 的微结构是**沙漏形**，而 MobileNet V2 则是**纺锤形**，刚好相反。因此论文作者将 MobileNet V2 的结构称为 Inverted Residual Block。这么做也是因为使用DW卷积而作的适配，希望特征提取能够在高维进行。

3.ShuffleNet v1

group操作我在mobileNet中已经说了，但group会造成通道间信息不流通，mobileNet给出的方案的接一层point_wise。

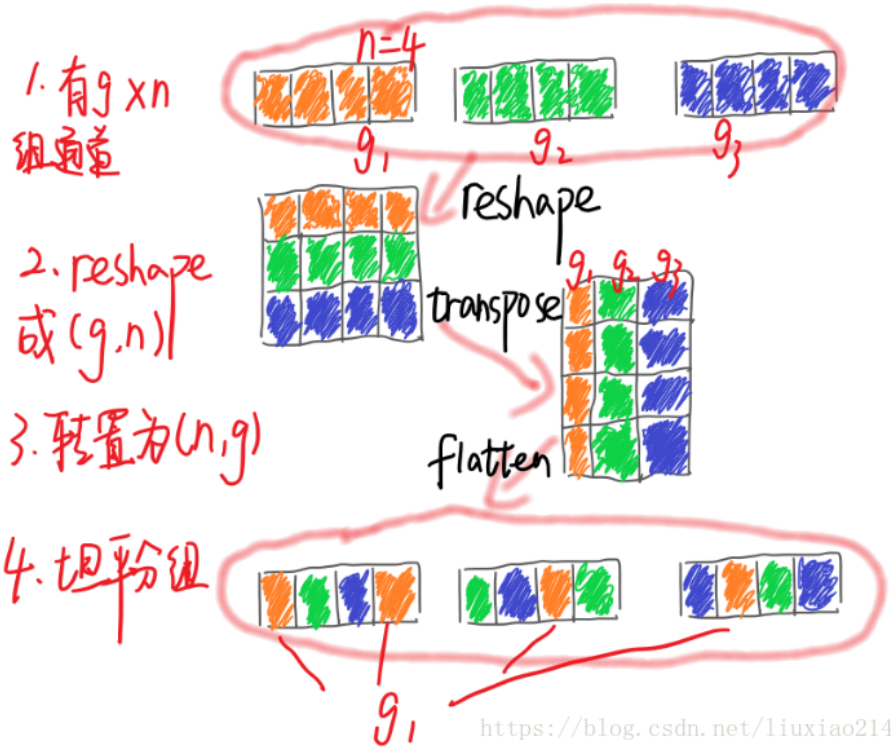
shuffleNet给出的方案则是打乱通道，并均匀重组。

比如我有32个通道，group=2，我将每个group再分成2份，这样有4份(1，2，3，4)，然后重组，1和3为第一个group，2和4为第2个group。然后对重组后的feature map进行depth_wise卷积。

3.1如何进行shuffle

对于一个卷积层分为g组，

1. 卷积后一共得到g×n个输出通道的feature map；
2. 将feature map 进行 reshape为(g,n);
3. 进行转置为(n,g)；
4. 对转置结果flatten，再分回g组作为下一层的输入。



3.2网络结构

下图中，a是标准的残差结构，不过是3x3卷积核使用了mobilenet中的depthwise convolution操作；

b是在a的基础上加了本文的通道shuffle操作，先对1x1卷积进行分组卷积操作，然后进行channel shuffle；

c是在旁路加了一步长为2的3x3的平均池化，并将前两者残差相加的操作改为了通道concat，增加了通道数量

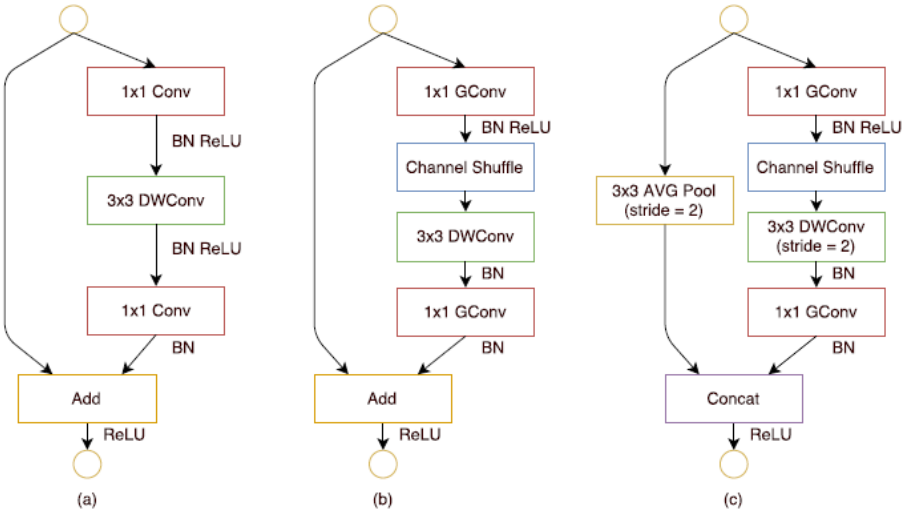


Figure 2. ShuffleNet Units. a) bottleneck unit [9] with depthwise convolution (DWConv) [3, 12]; b) ShuffleNet unit with pointwise group convolution (GConv) and channel shuffle; c) ShuffleNet unit with stride = 2.

借助 ShuffleNet 结构单元，作者构建了完整的 ShuffeNet 网络模型。它主要由 16 个 ShuffleNet 结构单元堆叠而成，分属网络的三个阶段，每经过一个阶段特征图的空间尺寸减半，而通道数翻倍。整个模型的总计算量约为 140 MFLOPs。通过简单地将各层通道数进行放缩，可以得到其他任意复杂度的模型。

ShuffleNet的核心就是用pointwise group convolution，channel shuffle和depthwise separable convolution代替ResNet block的相应层构成了ShuffleNet uint，达到了减少计算量和提高准确率的目的。

channel shuffle解决了多个group convolution叠加出现的边界效应，pointwise group convolution和depthwise separable convolution主要减少了计算量。

4.ShuffleNet v2

论文直接指出：FLOPS数量并不能直接线性的衡量计算速度，很多时候，计算速度还和你的平台框架，有些平台的多线程优化的就很不错，当然速度也会有提升。速度还和硬件设备有关.

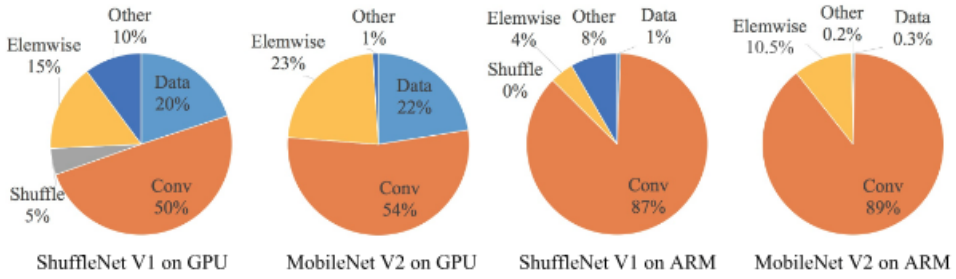


Fig. 2: Run time decomposition on two representative state-of-the-art network architectures, *ShuffeNet v1* [15] (1×, $g = 3$) and *MobileNet v2* [14] (1×).

我们一般在计算计算量的时候，只会去计算浮点计算量，也就是图中卷积操作所需要的时间，根据上图分析可以得出，其实，CONV虽然说是占用了大量的时间，但是并不是全部的时间，对于MobilenetV2,shufflenet v1来说很大一部分时间是在element-wise 操作上面.

这里简要说一下什么是element-wise的操作

1. Batch normalization ,归一化操作，每次进行归一化操作的时候会将所有的element全部计算一遍，而且这部分是很耗时间啊的，这部分是在CPU上面进行的，可以在GPU上面计算吗？至少我现在是没有找到可以在GPU加速的BN，GPU只是对于矩阵操作加速很大，一般情况下，加速其实不明显的，
2. activate function，激活函数，这显而易见，所有的元素都需要过一遍
3. identity-skip 就是res中的shortcut，我喜欢用shortcut称呼，这个是element-wise add 操作
4. 现在有的一个element-wise product操作
5. 所有的pooling都是，我在前面有讲过，pooling这玩意儿应该有很大的提升空间，其实我觉得avgpool操作在某种程度上是可以进行卷积加速的，就是卷积核的参数都为1，而且卷积和参数固定不变，但是至今没见到过有人这样写这个模块

另外一些占时间就是IO了，而且根据个人经验，网络层数越大，在IO上面占用的时间越多

四个指导方针：

1. 卷积核数量尽量与输入通道数相同（即输入通道数等于输出通道数）；

上面的字我加粗，说明很重要，为什么这么说？作者给出了一个实验证明，

		GPU (Batches/sec.)				ARM (Images/sec.)		
c1:c2	(c1,c2) for ×1	×1	×2	×4	(c1,c2) for ×1	×1	×2	×4
1:1	(128,128)	1480	723	232	(32,32)	76.2	21.7	5.3
1:2	(90,180)	1296	586	206	(22,44)	72.9	20.5	5.1
1:6	(52,312)	876	489	189	(13,78)	69.1	17.9	4.6
1:12	(36,432)	748	392	163	(9,108)	57.6	15.1	4.4

Table 1: Validation experiment for **Guideline 1**. Four different ratios of number of input/output channels ($c1$ and $c2$) are tested, while the total FLOPs under the four ratios is fixed by varying the number of channels. Input image size is 56×56.

这个实验说明，在input_channels equals output_channels时，计算的效率是最高的.

文章提出了一个MAC的概念（我以前没见过，如果这个MAC概念以前出现过，请通知我改正，谢谢）

这个MAC是什么呢？是 the memory access cost,或者是the number of memory access operations

MAC=hw(c1+c2)+c1c2

而我们的一般卷积的浮点计算怎么算的：

B=hwc1c2

这样就有

MAC >= 2*sqrt(h*w*B)+B/(hw)

当c1=c2的时候，MAC取下确界值

2. **谨慎使用group convolutions，注意group convolutions的效率；**
3. **降低网络碎片化程度；**(比如Inception中的多路径)
4. **减少元素级运算,**(比如element wise add)

shuffleNet_v2用到了通道分割:: 我们shuffle后，将channel一分为二，一部分不动(准则3)，另一部分用于卷积。为了满足准则(1),涉及到的卷积都是通道不变化的。首先是普通的conv，然后接一个depth_wise，point_size。这里只进行了一次group conv(准则2)。最后concat，而不是eltwise(准则4)