ShuffleNetV2

论文地址：ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design

1 创新点

(1) 基于主流网络进行多组对比实验，总结出了4条关于 CNN 网络结构设计的准则来帮助神经网络可以更高效。这是本文最大的创新点

(2) 基于上述4条准则，在shuffleNet的基础上，提出了 shuffleNet V2，并通过大量实验验证了网络结构的有效性。

2 核心思想

2.1 评估方法存在的不足

以前的轻量级CNN框架取得了很好的效果，例如Xception、MobileNet、MobileNetV2和ShuffleNet等，主要采用的技术手段是分组卷积Group convolution和深度卷积depth-wise convolution。

在以前文章中，除了精度要求外，评价模型的复杂程度都是采用FLOPs指标(每秒浮点加乘运算次数)，在mobilnetv1、v2和ShuffleNet都是采用该指标的。然而，FLOPs 是一种间接指标，它只是真正关心的直接指标（如速度或延迟）的一种近似形式，通常无法与直接指标划等号。先前研究已对这种差异有所察觉。比如 MobileNet V2要远快于 NASNET-A(自动搜索出的神经网络)，但是两者 FLOPs 相当。它表明 FLOPs 近似的网络也会有不同的速度。所以，将 FLOPs 作为衡量计算复杂度的唯一标准是不够的，这样会导致次优设计。

研究者注意到 FLOPs 仅和卷积部分相关，尽管这一部分需要消耗大部分的时间，但其它过程例如数据 I/O、数据重排和元素级运算（张量加法、ReLU 等）也需要消耗一定程度的时间。

间接指标 (FLOPs) 和直接指标（速度）之间存在差异的原因可以归结为两点。首先，对速度有较大影响的几个重要因素对 FLOPs 不产生太大作用。其中一个因素是内存访问成本 (MAC)。在某些操作（如组卷积）中，MAC 占运行时间的很大一部分。对于像 GPU 这样具备强大计算能力的设备而言，这就是瓶颈。在网络架构设计过程中，内存访问成本不能被简单忽视。另一个因素是并行度。当 FLOPs 相同时，高并行度的模型可能比低并行度的模型快得多。

其次，FLOPs 相同的运算可能有着不同的运行时间，这取决于平台。例如，早期研究广泛使用张量分解来加速矩阵相乘。但是，近期研究发现张量分解在 GPU 上甚至更慢，尽管它减少了 75% 的 FLOPs。本文研究人员对此进行了调查，发现原因在于最近的 CUDNN库专为 3×3 卷积优化：当然不能简单地认为 3×3 卷积的速度是 1×1 卷积速度的 1/9。

总结如下就是：

1. **影响模型运行速度还有别的指标，例如MAC(memory access)，并行度(degree of parallelism)**
2. **不同平台有不同的加速算法，这导致flops相同的运算可能需要不同的运算时间。**

据此，研究者提出了高效网络架构设计应该考虑的两个基本原则：第一，应该用直接指标（例如速度）替换间接指标（例如 FLOPs）；第二，这些指标应该在目标平台上进行评估。在这项研究中，作者遵循这两个原则，并提出了一种更加高效的网络架构。

2.2 高效网络设计原则

(1) G1. 相同的通道宽度可最小化内存访问成本（MAC）

假设内存足够大一次性可存储所有feature maps and parameters；卷积核大小为$1*1$ ；输入通道有c1个；输出通道有c2个；feture map的分辨率为$h*w$，则在1x1的卷积上，FLOPS:$B=hwc1c2$，
$MAC=hw(c1+c2)+c1c2$，容易推出：

$$\mathrm{MAC} \geq 2\sqrt{hwB}+\frac{B}{hw}.$$

MAC是值内存访问量，hwc1是输入特征图内存大小，hwc2是输出特征图内存大小，c1xc2是卷积核内存大小。

从公式中我们可以得出MAC的一个下界，即当c1==c2 时，MAC取得最小值。以上是理论证明，下面是实验证明：

		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.

可以看出，当c1==c2时，无论在GPU平台还是ARM平台，均获得了最快的runtime 。

(2) G2. 过度的组卷积会增加 MAC

和(1)假设一样，设g表示分组数，则有：

$$\begin{aligned}\mathrm{MAC} &= hw(c_1+c_2)+\frac{c_1c_2}{g}\\ &= hwc_1+\frac{Bg}{c_1}+\frac{B}{hw},\end{aligned}$$

其中$B=hwc1c2/g$，当固定c1 w h和B，增加分组g，MAC也会增加，证明了上述说法。其中c1 w h固定，g增加的同时B复杂度也要固定，则需要把输出通道c2增加，因为g增加，复杂度是要下降的。以上是理论证明，下面是实验证明：

		GPU (Batches/sec.)				CPU (Images/sec.)		
g	c for ×1	×1	×2	×4	c for ×1	×1	×2	×4
1	128	2451	1289	437	64	40.0	10.2	2.3
2	180	1725	873	341	90	35.0	9.5	2.2
4	256	1026	644	338	128	32.9	8.7	2.1
8	360	634	445	230	180	27.8	7.5	1.8

Table 2: Validation experiment for **Guideline 2**. Four values of group number g are tested, while the total FLOPs under the four values is fixed by varying the total channel number c. Input image size is 56×56.

可以看出，g增加，runtime减少。

(3) G3. 网络碎片化（例如 GoogLeNet 的多路径结构）会降低并行度

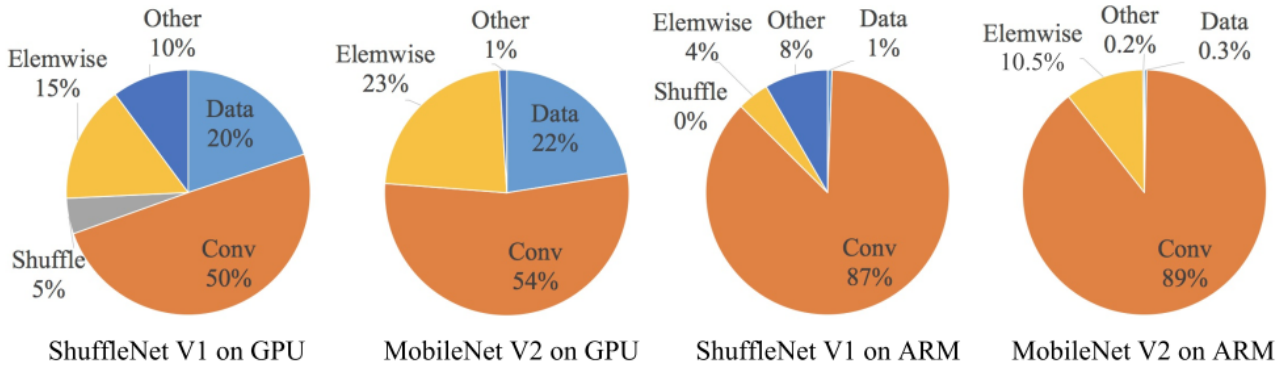
理论上，网络的碎片化虽然能给网络的accuracy带来帮助，但是在平行计算平台（如GPU）中，网络的碎片化会引起并行度的降低，最终增加runtime，同样的该文用实验验证:

	GPU (Batches/sec.)			CPU (Images/sec.)		
	c=128	c=256	c=512	c=64	c=128	c=256
1-fragment	2446	1274	434	40.2	10.1	2.3
2-fragment-series	1790	909	336	38.6	10.1	2.2
4-fragment-series	752	745	349	38.4	10.1	2.3
2-fragment-parallel	1537	803	320	33.4	9.1	2.2
4-fragment-parallel	691	572	292	35.0	8.4	2.1

Table 3: Validation experiment for **Guideline 3**. c denotes the number of channels for *1-fragment*. The channel number in other fragmented structures is adjusted so that the FLOPs is the same as *1-fragment*. Input image size is 56×56.

可以看出，碎片化越多，runtime越大。

(4) 元素级运算不可忽视



element-wise operations也占了不少runtime，尤其是GPU平台下，高达15%。ReLU、AddTensor及AddBias等这一类的操作就属于element-wise operations. 这一类操作，虽然flops很低，但是MAC很大，因而也占据相当时间。同样地，通过实验分析element-wise operations 越少的网络，其速度越快.

		GPU (Batches/sec.)			CPU (Images/sec.)		
ReLU	short-cut	c=32	c=64	c=128	c=32	c=64	c=128
yes	yes	2427	2066	1436	56.7	16.9	5.0
yes	no	2647	2256	1735	61.9	18.8	5.2
no	yes	2672	2121	1458	57.3	18.2	5.1
no	no	2842	2376	1782	66.3	20.2	5.4

Table 4: Validation experiment for **Guideline 4**. The ReLU and shortcut operations are removed from the "bottleneck" unit [4], separately. c is the number of channels in unit. The unit is stacked repeatedly for 10 times to benchmark the speed.

以上就是作者提出的4点设计要求。ShuffleNet V1 严重依赖组卷积（违反 G2）和瓶颈形态的构造块（违反 G1）。MobileNet V2 使用倒置的瓶颈结构（违反G1），并且在"厚"特征图上使用了深度卷积和 ReLU 激活函数（违反了 G4）。自动生成结构的碎片化程度很高，违反了 G3。

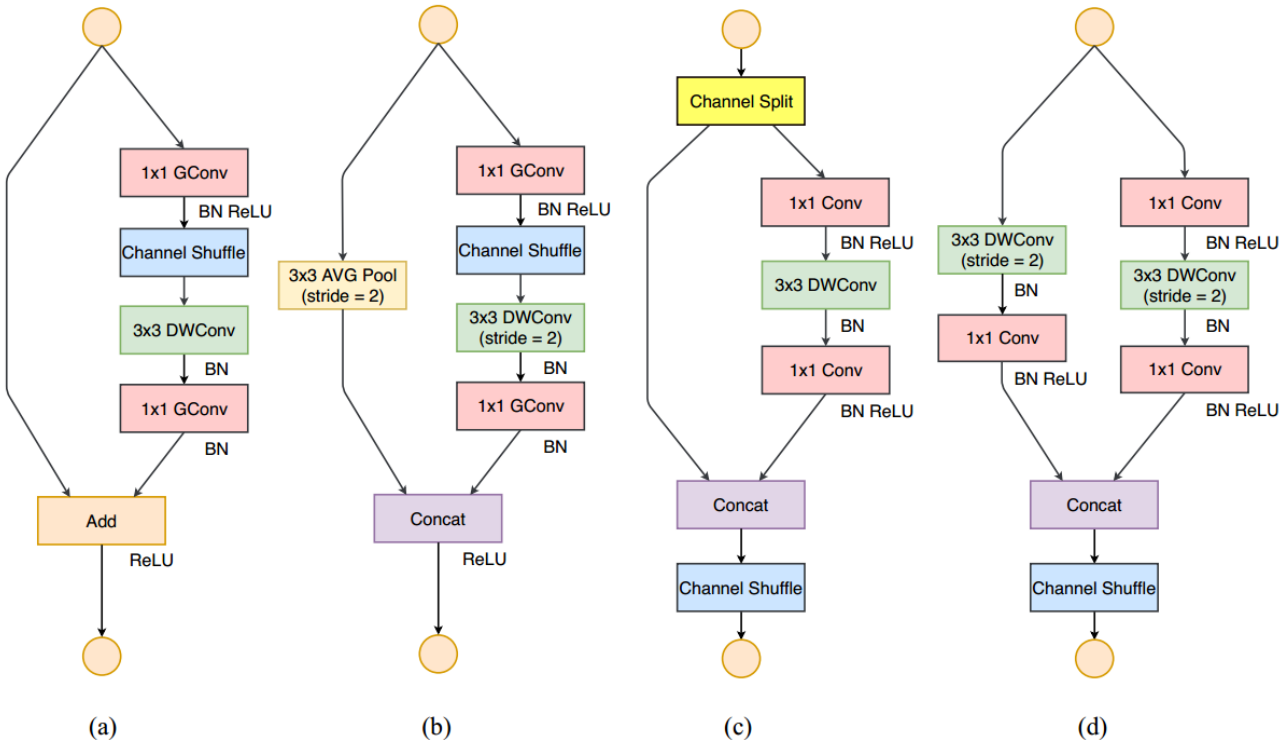
3 网络模型

在ShuffleNet V1中，特色在于：

1. pointwise group convolution，这违反了本文提出的G2；
2. channel shuffle。ShuffleNet V1还采用了 bottleneck-like 结构，在bottleneck unit中，shortcut connection中的element-wise operations——add tensor，又违反了G4；

基于以上缺点，作者重新设计了v2版本。ShuffleNet V2主要结构是多个blocks堆叠构成，block又分为两种，一种是带Channel Spilit的，一种是带Stride=2的（为了降分辨率）。

首先分析带Channel Spilit的block。



(a)为v1中的stride=1的block，(b)为v1中的stride=2的block，(c)是新设计的v2中的stride=1的block，(d)是新设计的v2中的stride=2的block。

注意：在这里3x3的DW卷积操作相当于线性操作，没有加ReLu，至于为啥不加，在mobilenetv2中讲的比较清楚了，mobilenetv2是采用1x1的点卷积充当线性层。反正就应该在通道数少或者说低维度采用线性层

在每个单元的开始，c 特征通道的输入被分为两支，分别带有 c−c'和 c'个通道。按照准则 G3，一个分支仍然保持不变。另一个分支由三个卷积组成，为满足 G1，令输入和输出通道相同。与 ShuffleNet V1 不同的是，两个 1×1 卷积不再是组卷积。这部分是为了遵循 G2，部分是因为分割操作已经产生了两个组。卷积之后，把两个分支拼接起来，从而通道数量保持不变 (G1)。然后进行与 ShuffleNet V1 相同的「Channel Shuffle」操作来保证两个分支间能进行信息交流。

Shuffle之后，下一个单元开始运算。注意，ShuffleNet V1 中的「加法」操作不再存在。像 ReLU 和深度卷积这样的操作只存在一个分支中。另外，三个连续的操作「拼接」、「Channel Shuffle」和「通道分割」合并成一个操作。根据 G4，这些变化是有利的。

对于空间下采样stride=2，该单元经过稍微修改，见(d)。通道分割运算被移除。因此，输出通道数量翻了一倍。

以上两个部分就可以构成各种block。在block基础上构造网络，如下图：

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

Table 5: Overall architecture of ShuffleNet v2, for four different levels of complexities.

令 c' = c/2。整个网络结构类似于 ShuffleNet V1。二者之间只有一个区别：前者在全局平均池化层之前添加了一个额外的 1×1 卷积层来混合特征，ShuffleNet V1 中没有该层。与 ShuffleNet V1 类似，每个构建块中的通道数量可以扩展以生成不同复杂度的网络，标记为 0.5×、1×等。

4 实验结果

Model	FLOPs	Top-1 err. (%)
ShuffleNet v2-50 (ours)	**2.3G**	**22.8**
ShuffleNet v1-50 [15] (our impl.)	**2.3G**	25.2
ResNet-50 [4]	3.8G	24.0
SE-ShuffleNet v2-164 (ours, with residual)	**12.7G**	**18.56**
SENet [8]	20.7G	18.68

Table 6: Results of large models. See text for details.

可以看出，和v1相比，在相同FLOP情况下，错误率更低，且和其他模型对比，复杂度明显下降。

Model	mmAP(%)				GPU Speed (Images/sec.)			
FLOPs	40M	140M	300M	500M	40M	140M	300M	500M
Xception	21.9	29.0	31.3	32.9	178	131	101	83
ShuffleNet v1	20.9	27.0	29.9	32.9	152	85	76	60
MobileNet v2	20.7	24.4	30.0	30.6	146	111	94	72
ShuffleNet v2 (ours)	22.5	29.0	31.8	33.3	**188**	**146**	**109**	**87**
ShuffleNet v2* (ours)	**23.7**	**29.6**	**32.2**	**34.2**	183	138	105	83

Table 7: Performance on COCO object detection. The input image size is 800 × 1200. *FLOPs* row lists the complexity levels at 224 × 224 input size. For GPU speed evaluation, the batch size is 4. We do not test ARM because the *PSRoI Pooling* operation needed in [34] is unavailable on ARM currently.

上述结果是在coco目标检测上的结果，输入图像大小是 800×1200。FLOPs 行列出了输入图像大小为 224×224 时的复杂度水平。带*的shuffleNet v2是变种，区别是扩大了感受野，具体为在每个block的第一个pointwise卷积的前面插入一个额外的3x3深度卷积。可以看出，精度提高不少，且FLOPs仅仅增加一点。这就留下一个疑问：如何增加网络的感受野也是一个值得探讨的问题。

Model	Complexity (MFLOPs)	Top-1 err. (%)	GPU Speed (Batches/sec.)	ARM Speed (Images/sec.)
ShuffleNet v2 0.5× (ours)	41	**39.7**	417	**57.0**
0.25 MobileNet v1 [13]	41	49.4	**502**	36.4
0.4 MobileNet v2 [14] (our impl.)*	43	43.4	333	33.2
0.15 MobileNet v2 [14] (our impl.)	39	55.1	351	33.6
ShuffleNet v1 0.5× (g=3) [15]	38	43.2	347	56.8
DenseNet 0.5× [6] (our impl.)	42	58.6	366	39.7
Xception 0.5× [12] (our impl.)	40	44.9	384	52.9
IGCV2-0.25 [27]	46	45.1	183	31.5
ShuffleNet v2 1× (ours)	146	**30.6**	341	**24.4**
0.5 MobileNet v1 [13]	149	36.3	**382**	16.5
0.75 MobileNet v2 [14] (our impl.)**	145	32.1	235	15.9
0.6 MobileNet v2 [14] (our impl.)	141	33.3	249	14.9
ShuffleNet v1 1× (g=3) [15]	140	32.6	213	21.8
DenseNet 1× [6] (our impl.)	142	45.2	279	15.8
Xception 1× [12] (our impl.)	145	34.1	278	19.5
IGCV2-0.5 [27]	156	34.5	132	15.5
IGCV3-D (0.7) [28]	210	31.5	143	11.7

参考文献

[1] http://www.sohu.com/a/243995156_129720

[2] https://www.zhihu.com/question/287433673/answer/455350957

[3] https://blog.csdn.net/u011995719/article/details/81409245