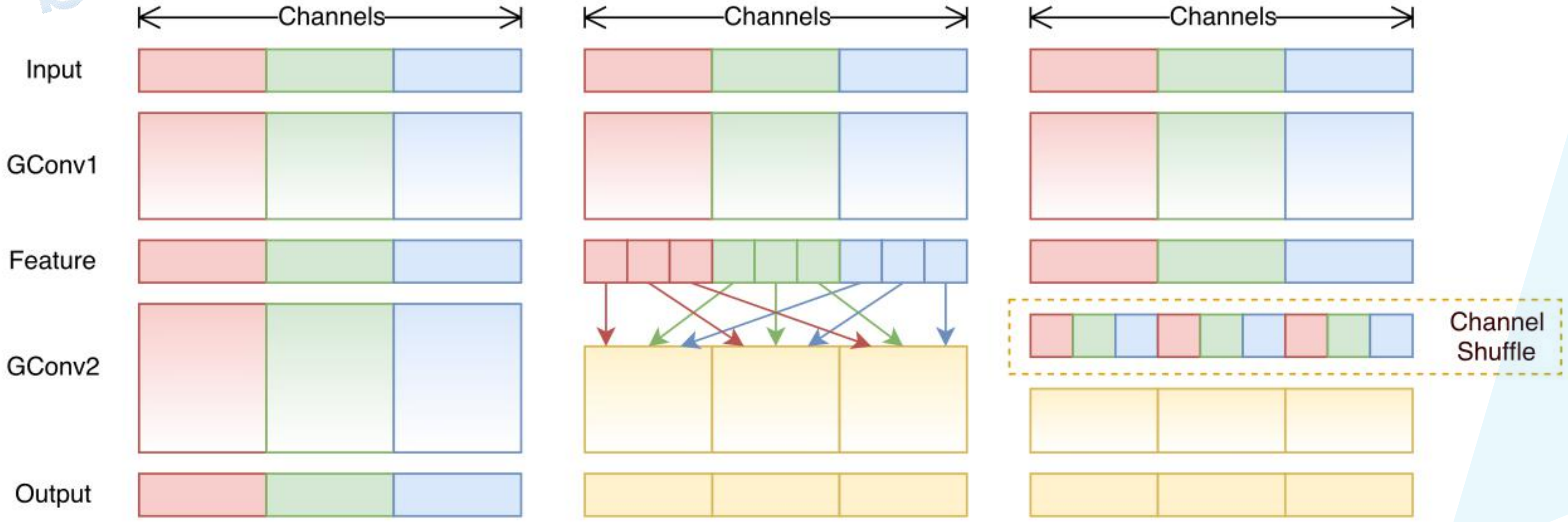
ShuffleNet v1

ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices

Xiangyu Zhang* Xinyu Zhou* Mengxiao Lin Jian Sun
Megvii Inc (Face++)
{zhangxiangyu,zxy,linmengxiao,sunjian}@megvii.com

提出了channel shuffle的思想

ShuffleNet Unit中全是GConv和DWConv



ShuffleNet v1

Model	Cls err. (%)	FLOPs	224 × 224	480 × 640	720 × 1280
ShuffleNet 0.5× ($g = 3$)	43.2	38M	15.2ms	87.4ms	260.1ms
ShuffleNet 1× ($g = 3$)	32.6	140M	37.8ms	222.2ms	684.5ms
ShuffleNet 2× ($g = 3$)	26.3	524M	108.8ms	617.0ms	1857.6ms
AlexNet [21]	42.8	720M	184.0ms	1156.7ms	3633.9ms
1.0 MobileNet-224 [12]	29.4	569M	110.0ms	612.0ms	1879.2ms

Table 8. Actual inference time on mobile device (*smaller number represents better performance*). The platform is based on a single Qualcomm Snapdragon 820 processor. All results are evaluated with **single thread**.

ShuffleNet v1

GConv虽然能够减少参数与计算量，
但GConv中不同组之间信息没有交流

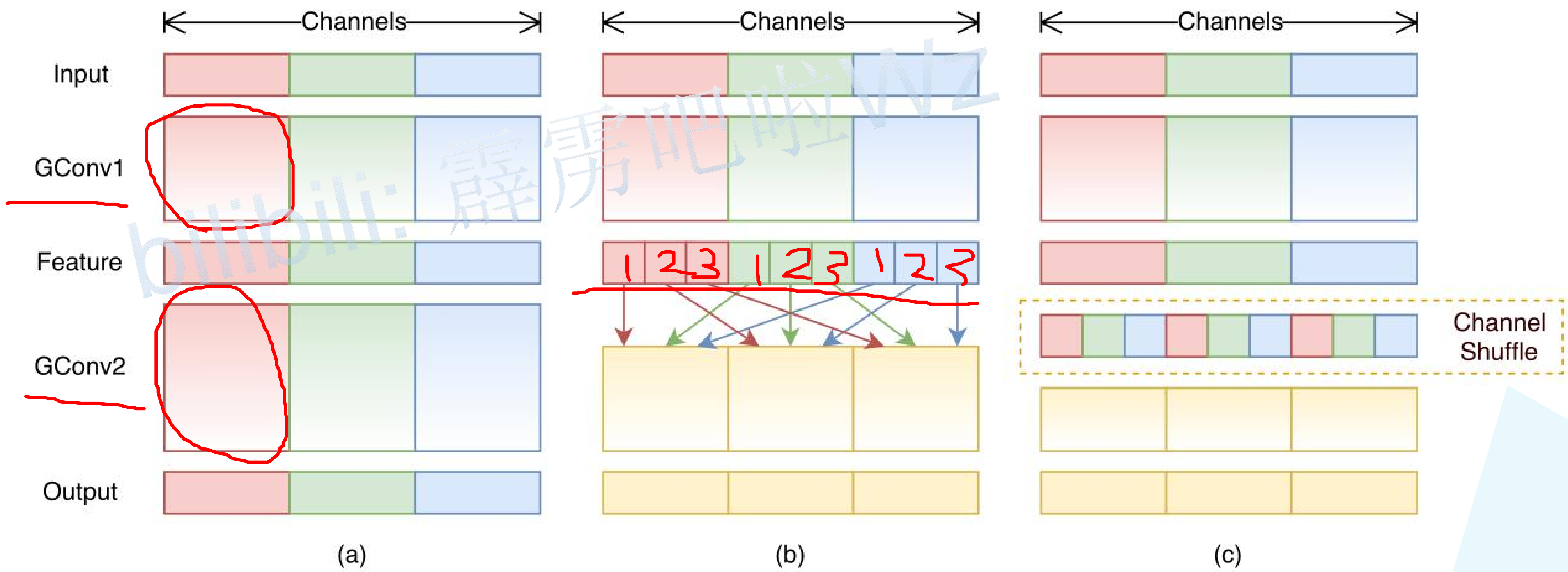
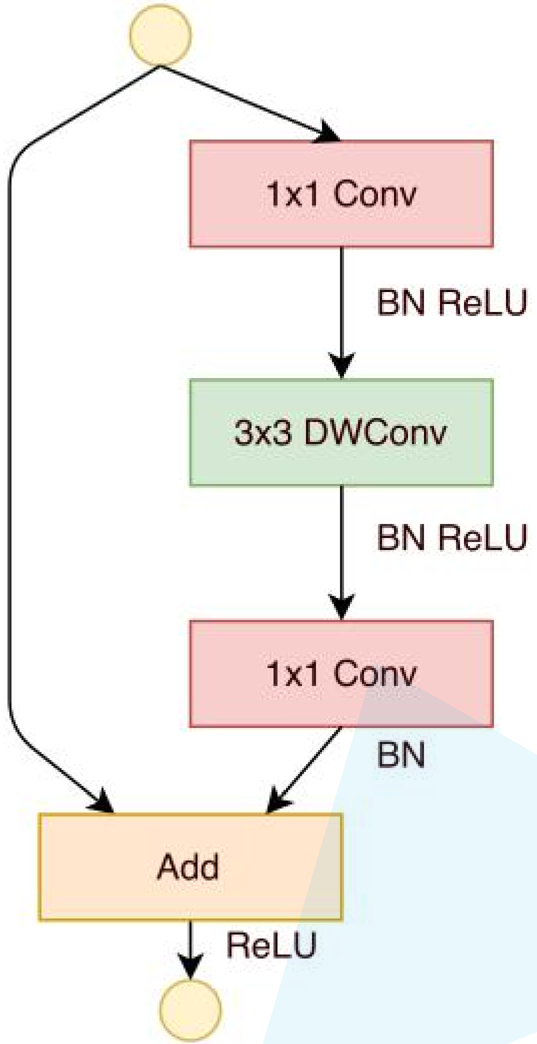


Figure 1. Channel shuffle with two stacked group convolutions. GConv stands for group convolution. a) two stacked convolution layers with the same number of groups. Each output channel only relates to the input channels within the group. No cross talk; b) input and output channels are fully related when GConv2 takes data from different groups after GConv1; c) an equivalent implementation to b) using channel shuffle.

ShuffleNet v1

3.1. Channel Shuffle for Group Convolutions

Modern convolutional neural networks [30, 33, 34, 32, 9, 10] usually consist of repeated building blocks with the same structure. Among them, state-of-the-art networks such as *Xception* [3] and *ResNeXt* [40] introduce efficient depthwise separable convolutions or group convolutions into the building blocks to strike an excellent trade-off between representation capability and computational cost. However, we notice that both designs do not fully take the 1×1 convolutions (also called *pointwise convolutions* in [12]) into account, which require considerable complexity. For example, in ResNeXt [40] only 3×3 layers are equipped with group convolutions. As a result, for each residual unit in ResNeXt the pointwise convolutions occupy 93.4% multiplication-adds (cardinality = 32 as suggested in [40]). In tiny networks, expensive pointwise convolutions result in limited number of channels to meet the complexity constraint, which might significantly damage the accuracy.



ShuffleNet v1

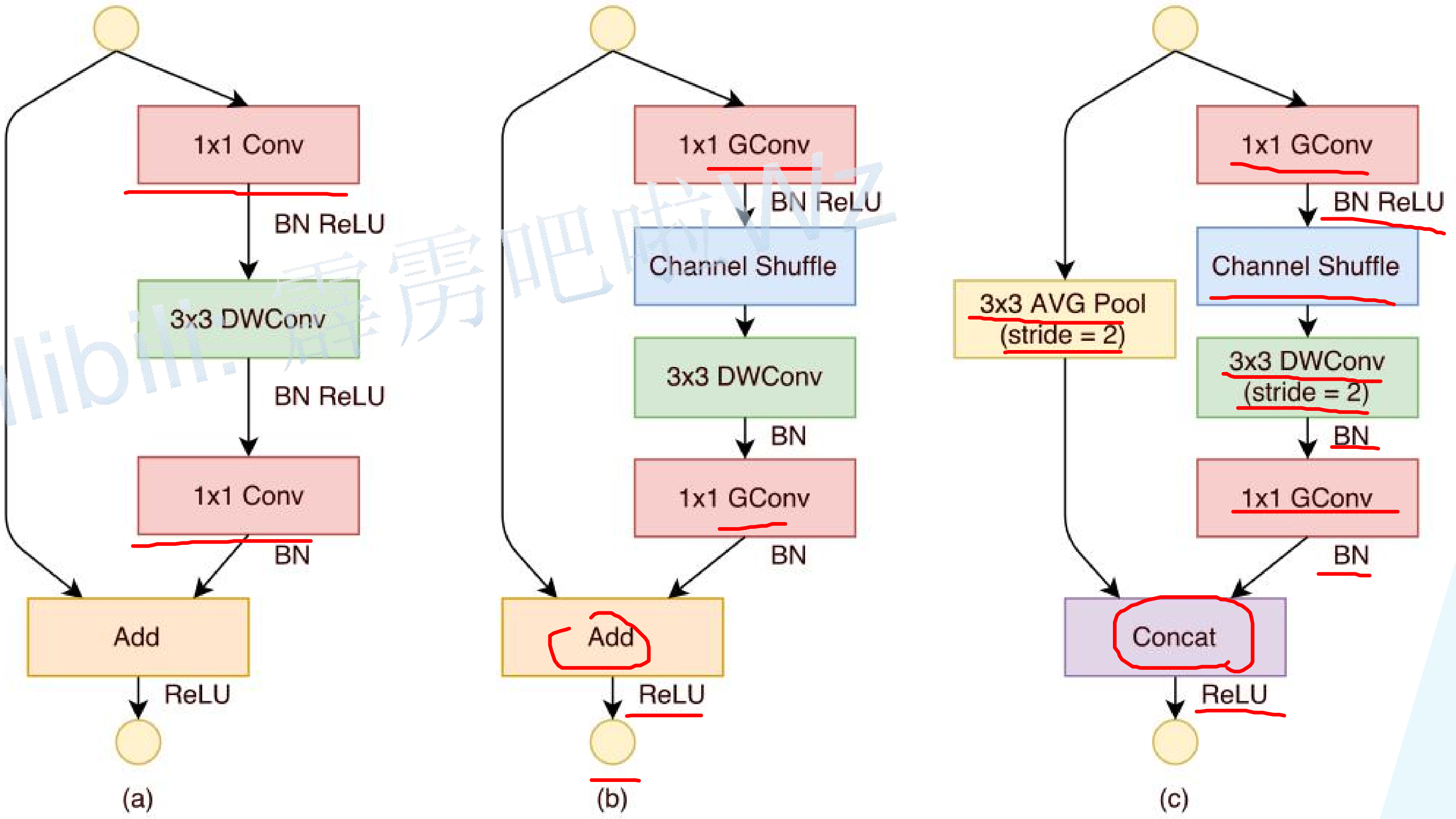
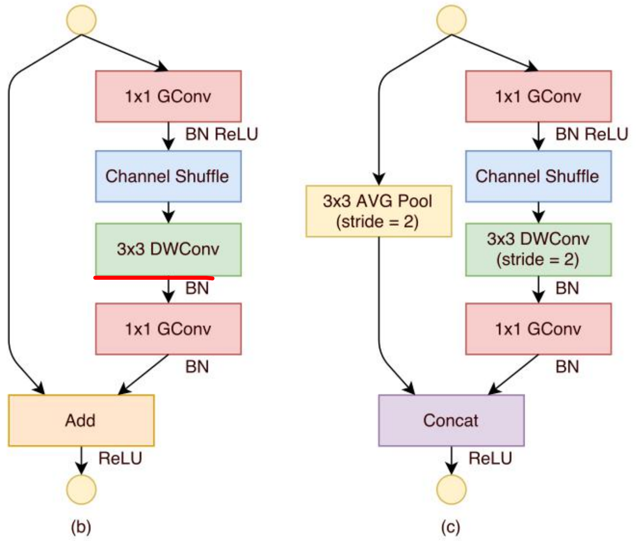
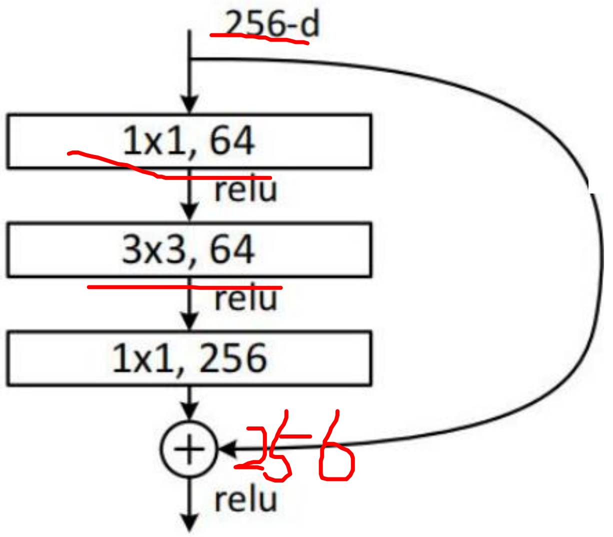


Figure 2. ShuffleNet Units. a) bottleneck unit [9] with depthwise convolution (DWConv) [3, 12]; b) ShuffleNet unit with pointwise group convolution (GConv) and channel shuffle; c) ShuffleNet unit with stride = 2.

ShuffleNet v1

Built on ShuffleNet units, we present the overall ShuffleNet architecture in Table 1. The proposed network is mainly composed of a stack of ShuffleNet units grouped into three stages. The first building block in each stage is applied with stride = 2. Other hyper-parameters within a stage stay the same, and for the next stage the output channels are doubled. Similar to [9], we set the number of bottleneck channels to 1/4 of the output channels for each ShuffleNet

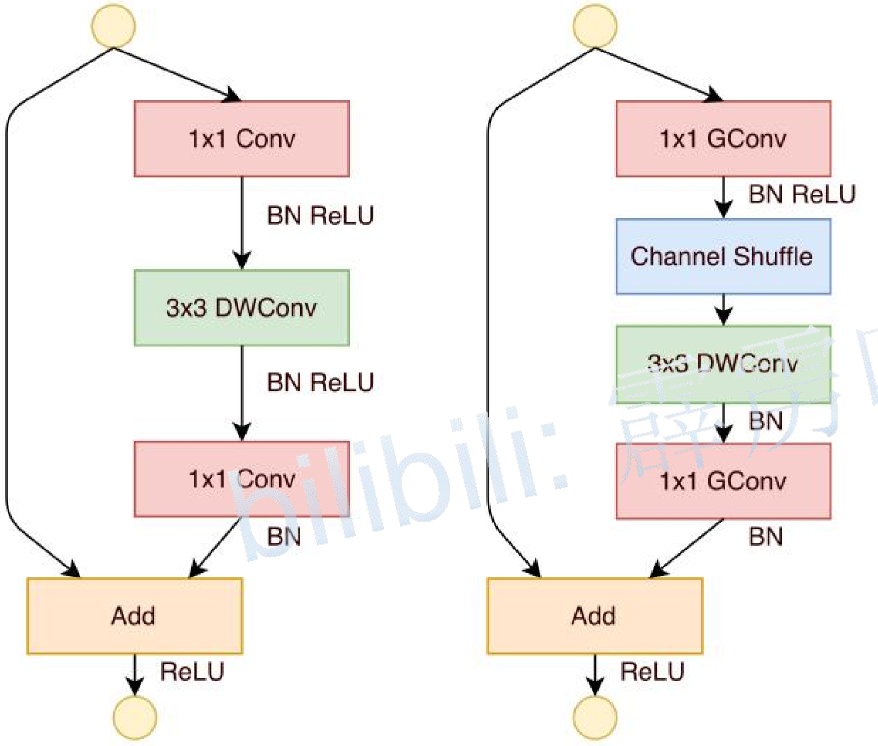




Layer	Output size	KSize	Stride	Repeat	Output channels (g groups)				
					$g = 1$	$g = 2$	$g = 3$	$g = 4$	$g = 8$
Image	224×224				3	3	3	3	3
Conv1	112×112	3×3	2	1	24	24	24	24	24
MaxPool	56×56	3×3	2						
Stage2	28×28		2	1	144	200	240	272	384
	28×28		1	3	144	200	240	272	384
Stage3	14×14		2	1	288	400	480	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

Table 1. ShuffleNet architecture. The complexity is evaluated with FLOPs, i.e. the number of floating-point multiplication-adds. Note that for Stage 2, we do not apply group convolution on the first pointwise layer because the number of input channels is relatively small.

ShuffleNet v1



Thanks to pointwise group convolution with channel shuffle, all components in ShuffleNet unit can be computed efficiently. Compared with ResNet [9] (bottleneck design) and ResNeXt [40], our structure has less complexity under the same settings. For example, given the input size $c \times h \times w$ and the bottleneck channels m, ResNet unit requires $hw(2cm + 9m^2)$ FLOPs and ResNeXt has $hw(2cm + 9m^2/g)$ FLOPs, while our ShuffleNet unit requires only $hw(2cm/g + 9m)$ FLOPs, where g means the

ResNet : $hw(1 \times 1 \times c \times m) + hw(3 \times 3 \times m \times m) + hw(1 \times 1 \times m \times c) = hw(2cm + 9m^2)$

ResNeXt : $hw(1 \times 1 \times c \times m) + hw(3 \times 3 \times m \times m)/g + hw(1 \times 1 \times m \times c) = hw(2cm + 9m^2/g)$

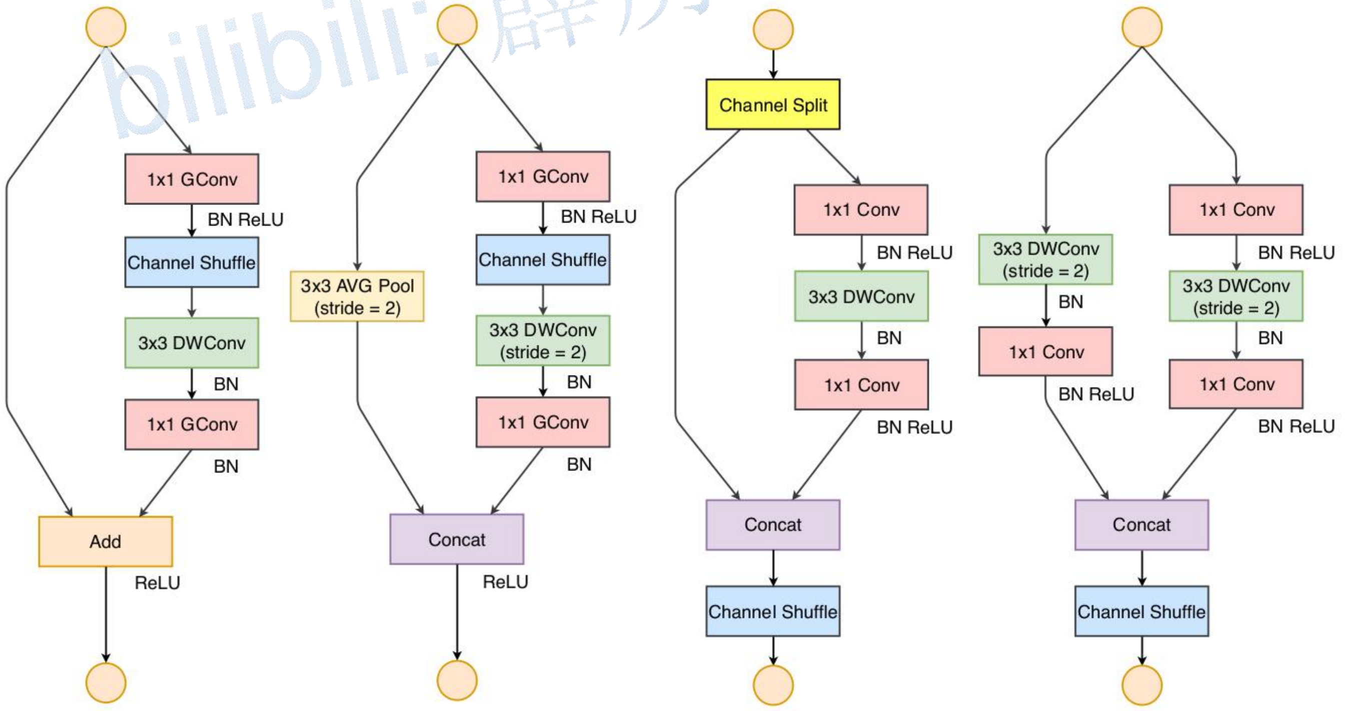
ShuffleNet : $hw(1 \times 1 \times c \times m)/g + hw(3 \times 3 \times m) + hw(1 \times 1 \times m \times c)/g = hw(2cm/g + 9m)$

ShuffleNet v2

ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design

Ningning Ma *[1,2] Xiangyu Zhang *[1] Hai-Tao Zheng[2] Jian Sun[1]

[1] Megvii Inc (Face++) [2] Tsinghua University
{maningning,zhangxiangyu,sunjian}@megvii.com
zheng.haitao@sz.tsinghua.edu.cn



FLOPS：**全大写**，指**每秒浮点运算次数**，可以理解为计算的速度。是衡量**硬件**性能的一个指标。（**硬件**）

FLOPs：**s小写**，**指浮点运算数**，理解为计算量。可以用来衡量算法/模型的复杂度。（**模型**）
在论文中常用GFLOPs（1 GFLOPs = 10^9 FLOPs）

计算复杂度不能只看FLOPs

↓

提出4条设计高效网络准则

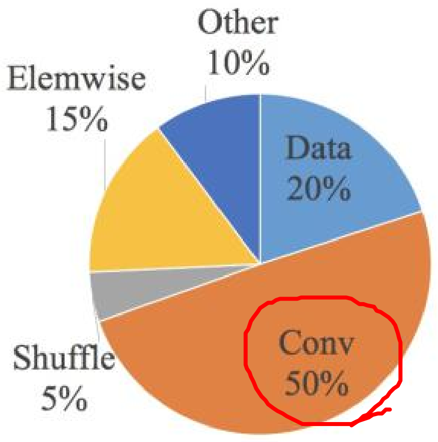
↓

提出新的block设计

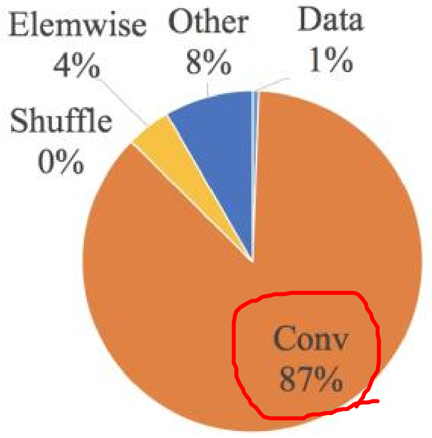
ShuffleNet v2

The discrepancy between the indirect (FLOPs) and direct (speed) metrics can be attributed to two main reasons. First, several important factors that have considerable affection on speed are not taken into account by FLOPs. One such factor is *memory access cost* (MAC). Such cost constitutes a large portion of runtime in certain operations like group convolution. It could be bottleneck on devices with strong computing power, e.g., GPUs. This cost should not be simply ignored during network architecture design. Another one is *degree of parallelism*. A model with high degree of parallelism could be much faster than another one with low degree of parallelism, under the same FLOPs.

Second, operations with the same FLOPs could have different running time, depending on the platform. For example, tensor decomposition is widely used in early works [20,21,22] to accelerate the matrix multiplication. However, the recent work [19] finds that the decomposition in [22] is even slower on GPU although it reduces FLOPs by 75%. We investigated this issue and found that this is because the latest CUDNN [23] library is specially optimized for 3 × 3 conv. We cannot certainly think that 3 × 3 conv is 9 times slower than 1 × 1 conv.



ShuffleNet V1 on GPU



ShuffleNet V1 on ARM

ShuffleNet v2

The overall runtime is decomposed for different operations, as shown in Figure 2. We note that the FLOPs metric only account for the convolution part. Although this part consumes most time, the other operations including data I/O, data shuffle and element-wise operations (AddTensor, ReLU, etc) also occupy considerable amount of time. Therefore, FLOPs is not an accurate enough estimation of actual runtime.

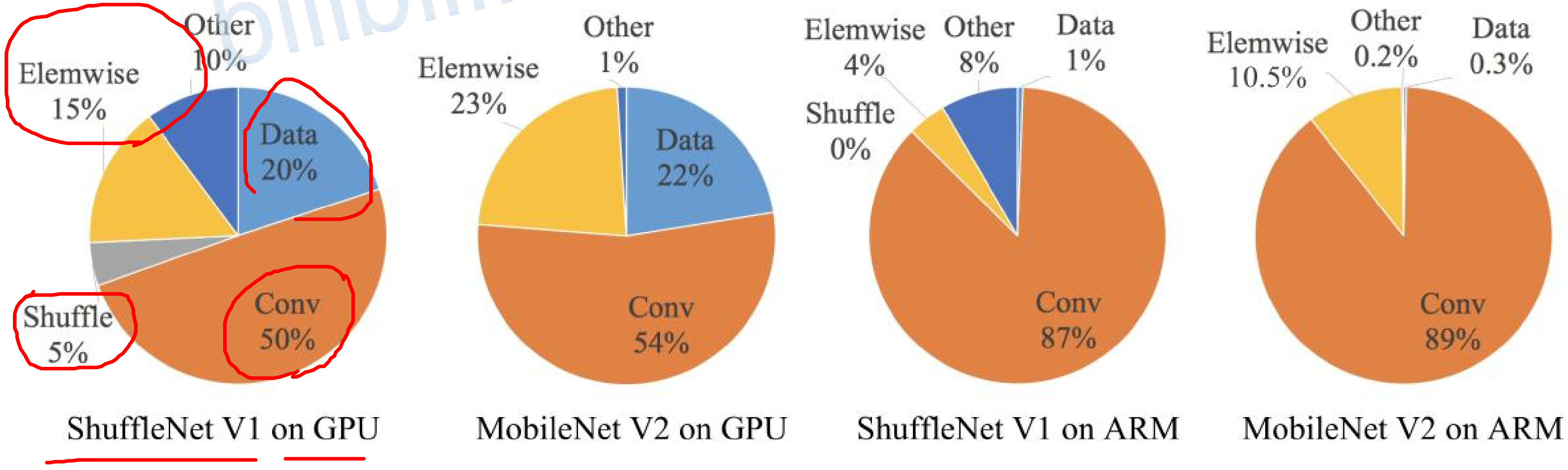


Fig. 2: Run time decomposition on two representative state-of-the-art network architectures, *ShuffeNet v1* [15] ($1\times$, $g = 3$) and *MobileNet v2* [14] ($1\times$).

ShuffleNet v2

Several practical guidelines for efficient network architecture design

G1: Equal channel width minimizes memory access cost (MAC)

G2: Excessive group convolution increases MAC

G3: Network fragmentation reduces degree of parallelism

G4: Element-wise operations are non-negligible

ShuffleNet v2

G1: Equal channel width minimizes memory access cost (MAC)

当卷积层的输入特征矩阵与输出特征矩阵channel相等时MAC最小(保持FLOPs不变时)

$$MAC \geq 2\sqrt{hwB} + \frac{B}{hw} \qquad B = hwc_1c_2 \quad \text{(FLOPs)}$$

算数平均数 几何平均数

$$MAC = hw(c_1 + c_2) + c_1c_2 \qquad \frac{c_1 + c_2}{2} \geq \sqrt{c_1c_2} \quad \text{均值不等式}$$

$$MAC \geq 2hw\sqrt{c_1c_2} + c_1c_2$$

$$\geq 2\sqrt{hwB} + \frac{B}{hw} \qquad B = hwc_1c_2$$

ShuffleNet v2

G1: Equal channel width minimizes memory access cost (MAC)

		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

The conclusion is theoretical. In practice, the cache on many devices is not large enough. Also, modern computation libraries usually adopt complex blocking strategies to make full use of the cache mechanism [24]. Therefore, the real MAC may deviate from the theoretical one. To validate the above conclusion, an experiment is performed as follows. A benchmark network is built by stacking 10 building blocks repeatedly. Each block contains two convolution layers. The first contains c_1 input channels and c_2 output channels, and the second otherwise.

Table 1 reports the running speed by varying the ratio $c_1 : c_2$ while fixing the total FLOPs. It is clear that when $c_1 : c_2$ is approaching $1 : 1$, the MAC becomes smaller and the network evaluation speed is faster.

ShuffleNet v2

G2: Excessive group convolution increases MAC

当GConv的groups增大时(保持FLOPs不变时)，MAC也会增大

$$MAC = hw(c_1 + c_2) + \frac{c_1 c_2}{g}$$

$$= hwc_1 + \frac{Bg}{c_1} + \frac{B}{hw} \qquad B = hwc_1c_2/g \quad \text{(FLOPs)}$$

where g is the number of groups and $B = hwc_1c_2/g$ is the FLOPs. It is easy to see that, given the fixed input shape $c_1 \times h \times w$ and the computational cost B, MAC increases with the growth of g.

ShuffleNet v2

G2: Excessive group convolution increases MAC

		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.

different group numbers while fixing the total FLOPs. It is clear that using a large group number decreases running speed significantly. For example, using 8 groups is more than two times slower than using 1 group (standard dense convolution) on GPU and up to 30% slower on ARM. This is mostly due to increased MAC. We note that our implementation has been specially optimized and is much faster than trivially computing convolutions group by group.

Therefore, we suggest that *the group number should be carefully chosen based on the target platform and task. It is unwise to use a large group number simply because this may enable using more channels, because the benefit of accuracy increase can easily be outweighed by the rapidly increasing computational cost.*

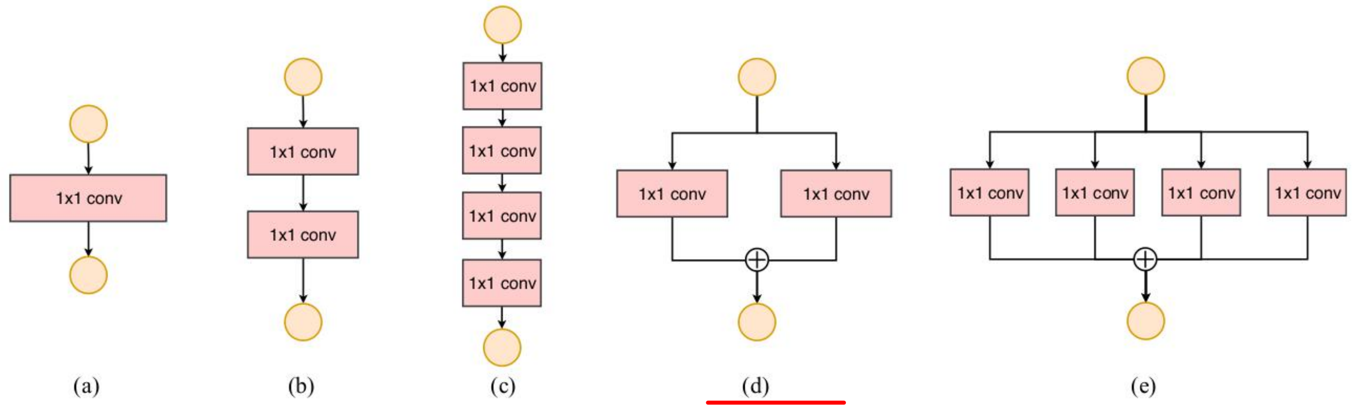
ShuffleNet v2

G3: Network fragmentation reduces degree of parallelism

网络设计的碎片化程度越高，速度越慢

Though such fragmented structure has been shown beneficial for accuracy, it could decrease efficiency because it is unfriendly for devices with strong parallel computing powers like GPU. It also introduces extra overheads such as kernel launching and synchronization.

	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



Appendix Fig. 1: Building blocks used in experiments for guideline 3. (a) *1-fragment*. (b) *2-fragment-series*. (c) *4-fragment-series*. (d) *2-fragment-parallel*. (e) *4-fragment-parallel*.

ShuffleNet v2

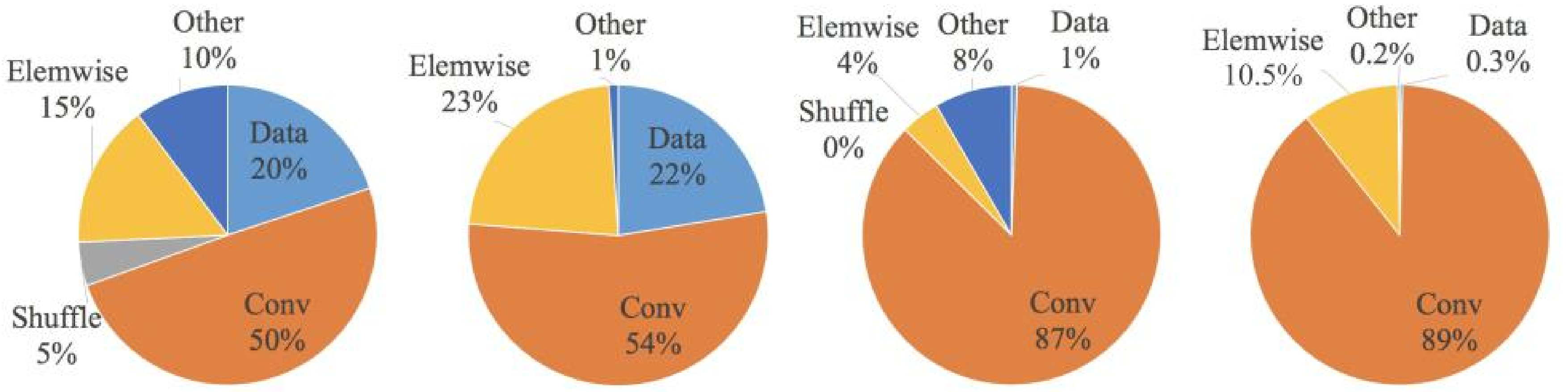
G4: Element-wise operations are non-negligible

Element-wise操作带来的影响是不可忽视的

G4) Element-wise operations are non-negligible. As shown in Figure 2, in light-weight models like [15,14], element-wise operations occupy considerable amount of time, especially on GPU. Here, the element-wise operators include ReLU, AddTensor, AddBias, etc. They have small FLOPs but relatively heavy MAC. Specially, we also consider *depthwise convolution* [12,13,14,15] as an element-wise operator as it also has a high MAC/FLOPs ratio.

time of different variants is reported in Table 4. We observe around 20% speedup is obtained on both GPU and ARM, after ReLU and shortcut are removed.

		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



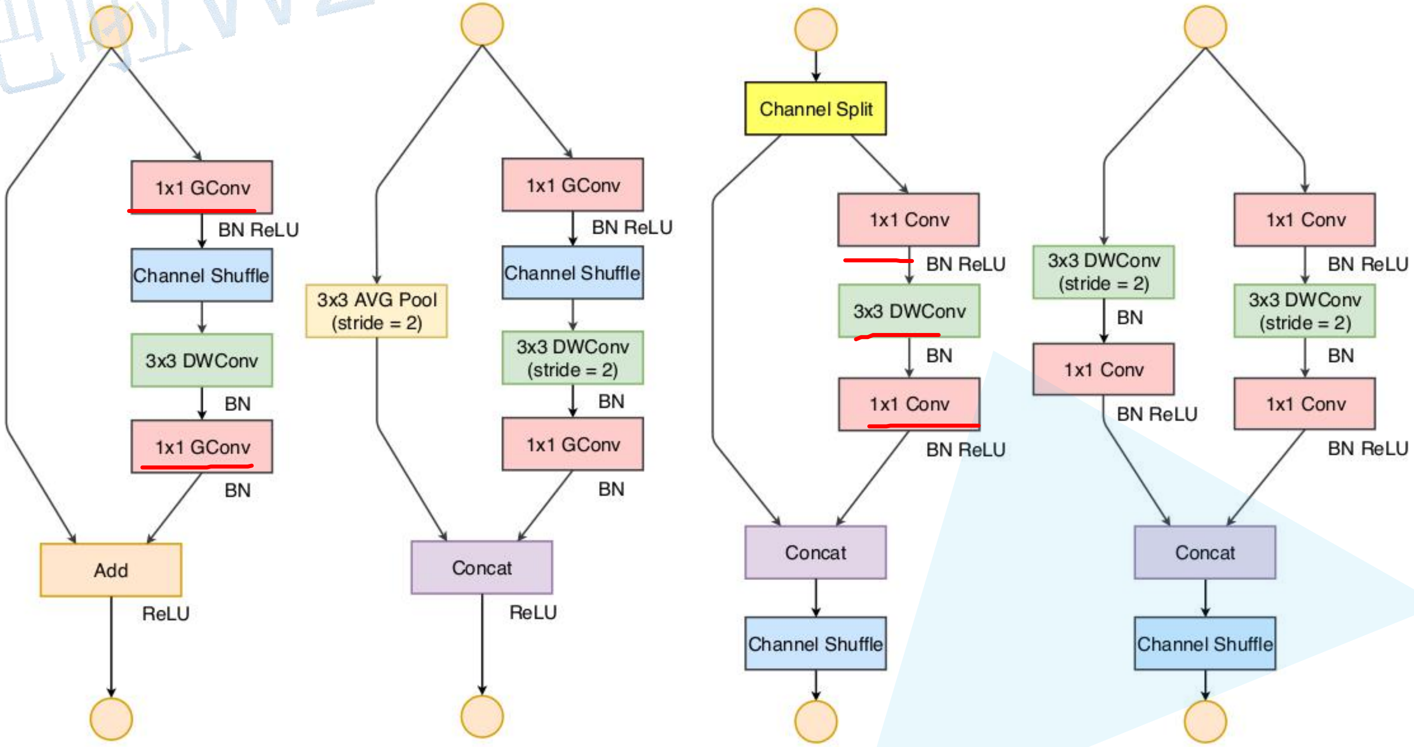
ShuffleNet V1 on GPU   MobileNet V2 on GPU   ShuffleNet V1 on ARM   MobileNet V2 on ARM

ShuffleNet v2

Conclusion and Discussions Based on the above guidelines and empirical studies, we conclude that an efficient network architecture should 1) use ”balanced“ convolutions (equal channel width); 2) be aware of the cost of using group convolution; 3) reduce the degree of fragmentation; and 4) reduce element-wise operations. These desirable properties depend on platform characterics (such as memory manipulation and code optimization) that are beyond theoretical FLOPs. They should be taken into accout for practical network design.

ShuffleNet v2

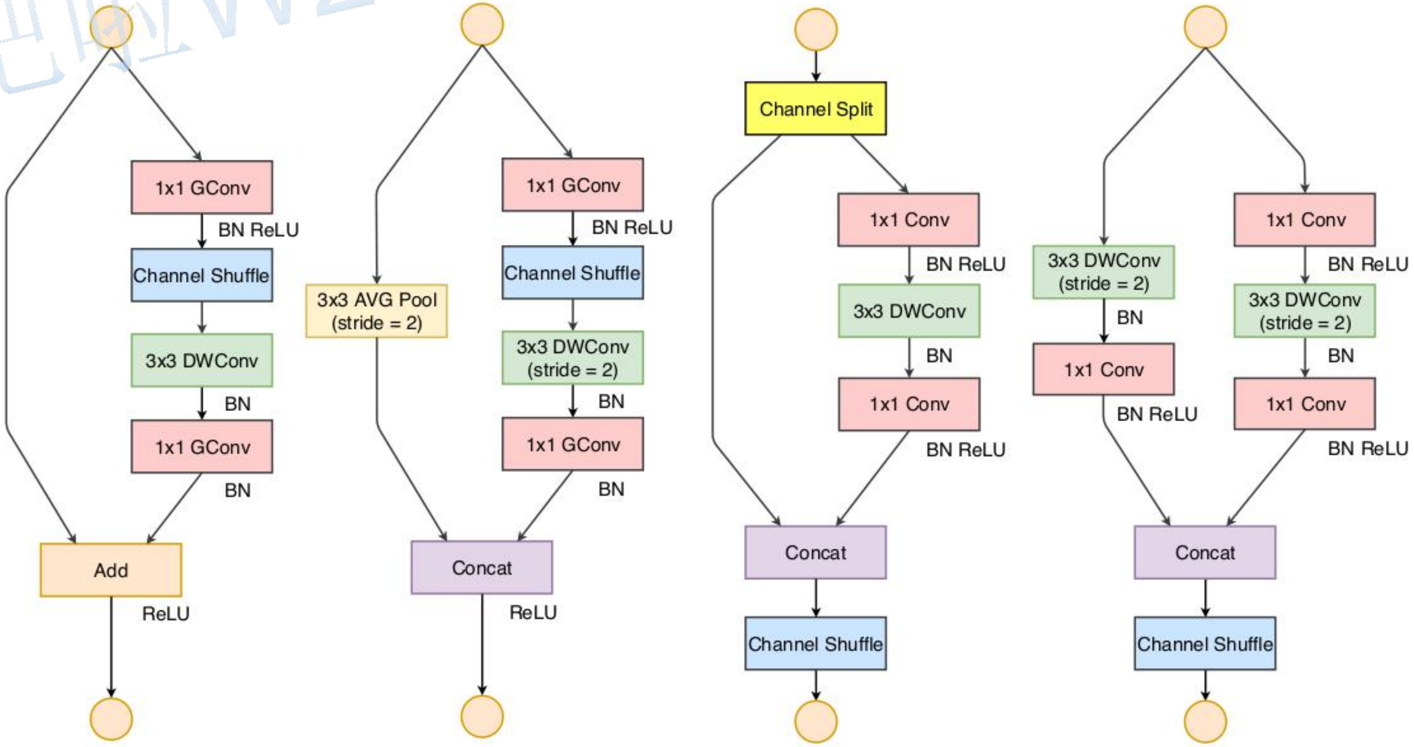
beginning of each unit, the input of c feature channels are split into two branches with $c - c'$ and c' channels, respectively. Following **G3**, one branch remains as identity. The other branch consists of three convolutions with the same input and output channels to satisfy **G1**. The two 1×1 convolutions are no longer group-wise, unlike [15]. This is partially to follow **G2**, and partially because the split operation already produces two groups.



(a) (b) (c) (d)

ShuffleNet v2

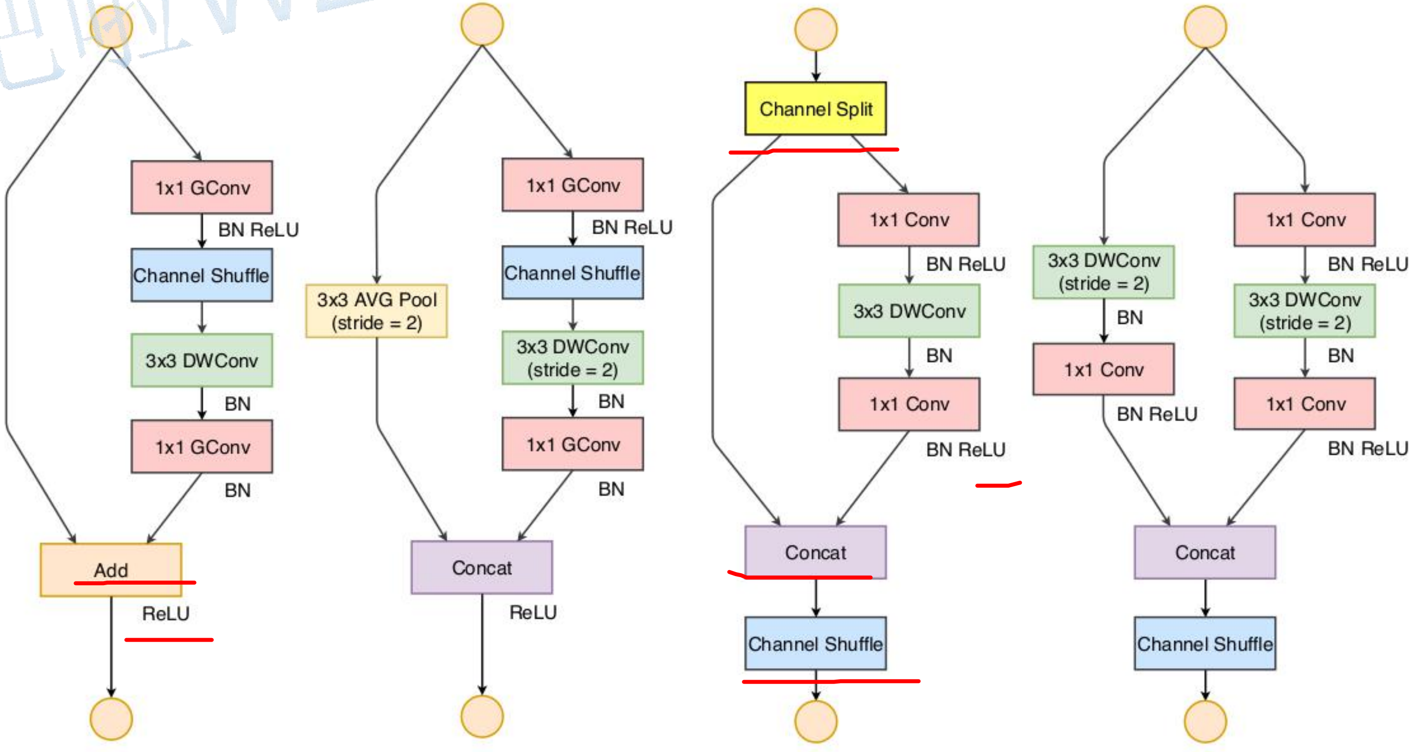
After convolution, the two branches are concatenated. So, the number of channels keeps the same (**G1**). The same "channel shuffle" operation as in [15]



(a) (b) (c) (d)

ShuffleNet v2

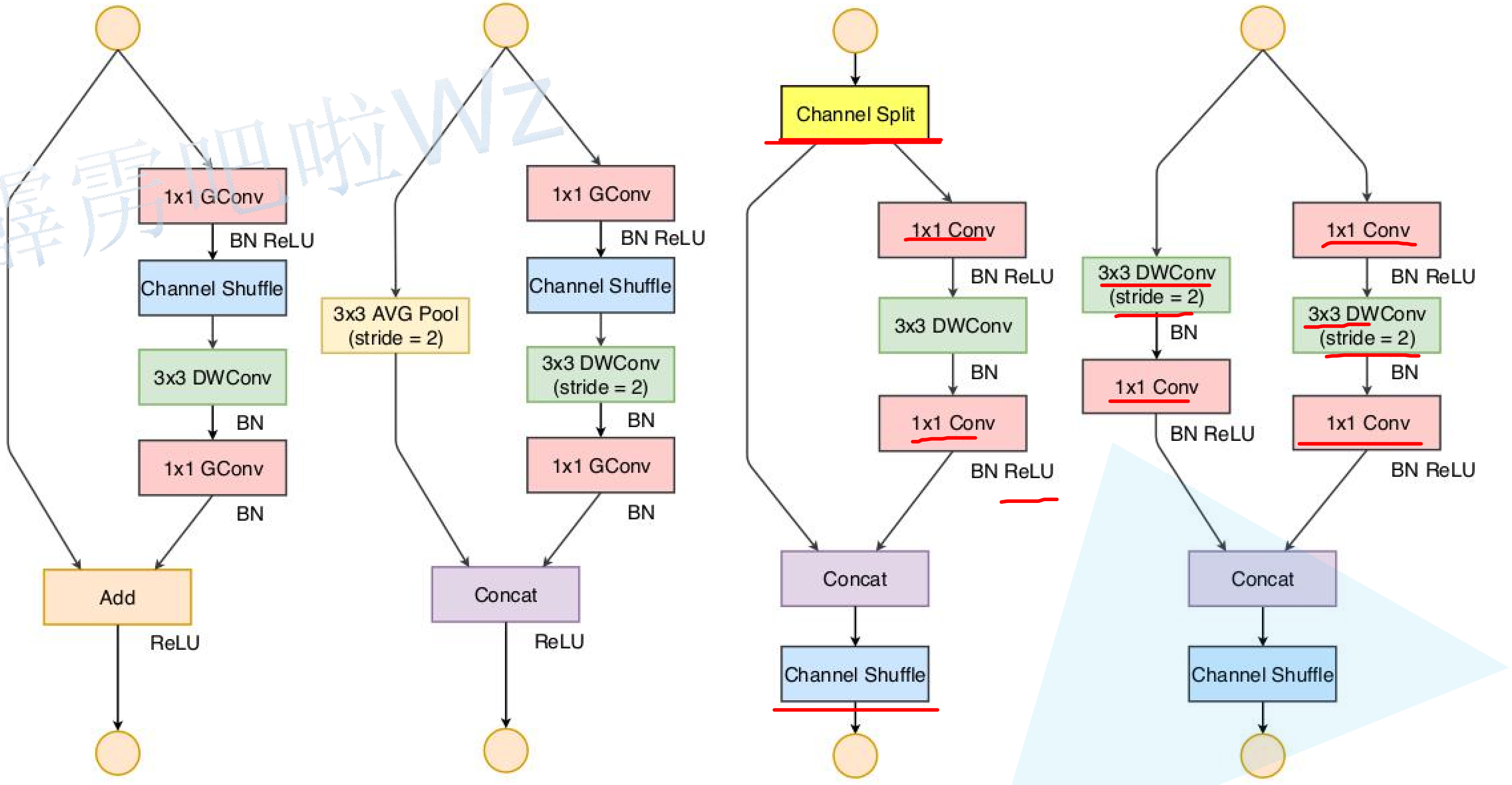
After the shuffling, the next unit begins. Note that the "Add" operation in ShuffleNet v1 [15] no longer exists. Element-wise operations like ReLU and *depth-wise convolutions* exist only in one branch. Also, the three successive element-wise operations, "Concat", "Channel Shuffle" and "Channel Split", are merged into a single element-wise operation. These changes are beneficial according to G4.



(a) (b) (c) (d)

ShuffleNet v2

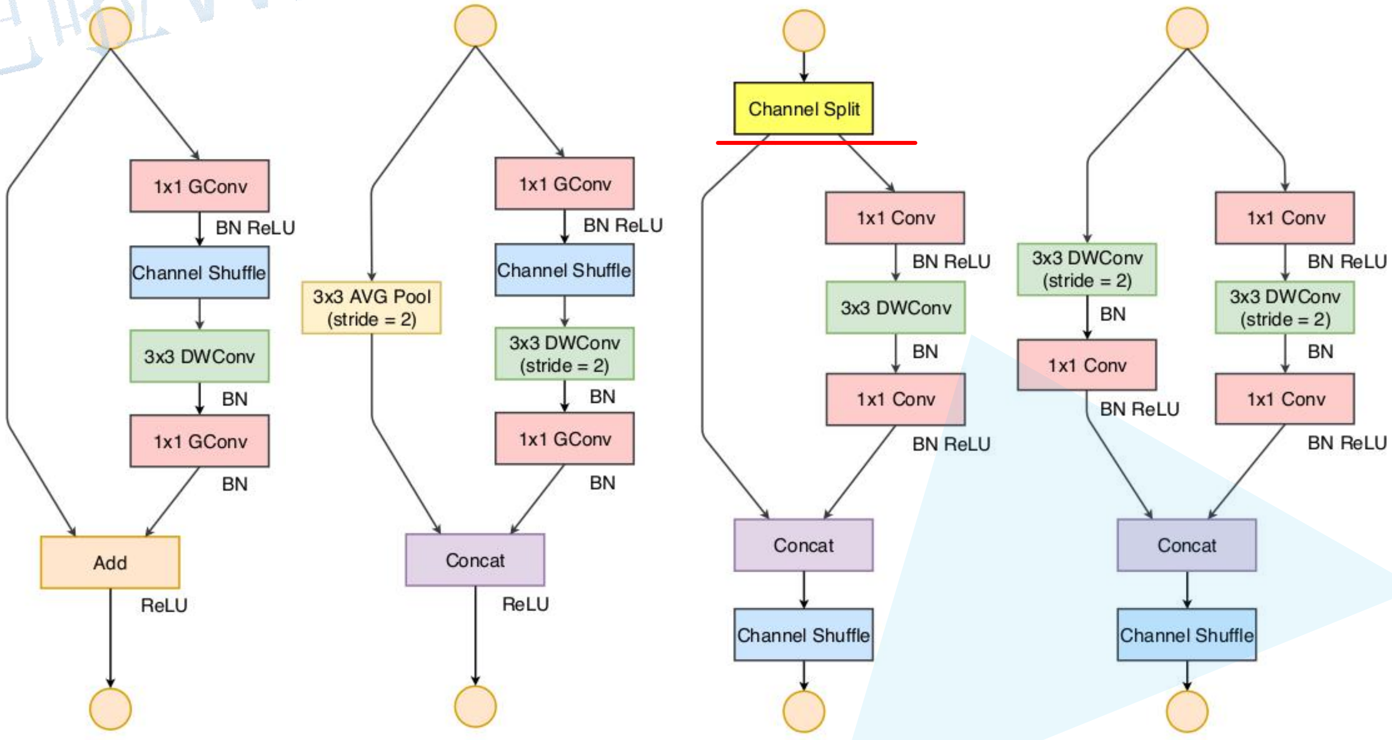
For spatial down sampling, the unit is slightly modified and illustrated in Figure 3(d). The channel split operator is removed. Thus, the number of output channels is doubled.



(a) (b) (c) (d)

ShuffleNet v2

The building blocks are repeatedly stacked to construct the whole network. For simplicity, we set $c' = c/2$. The overall network structure is similar to ShuffleNet v1 [15] and summarized in Table 5. There is only one difference: an additional 1×1 convolution layer is added right before *global averaged pooling* to mix up features, which is absent in ShuffleNet v1. Similar to [15], the number of channels in each block is scaled to generate networks of different complexities, marked as $0.5\times$, $1\times$, etc.



(a) (b) (c) (d)

ShuffleNet v2

fleNet v1 [15] and summarized in Table 5. There is only one difference: an additional 1×1 convolution layer is added right before *global averaged pooling* to mix up features, which is absent in ShuffleNet v1. Similar to [15], the number of channels in each block is scaled to generate networks of different complexities, marked as $0.5\times$, $1\times$, etc.

Layer	Output size	KSize	Stride	Repeat	Output channels			
					0.5×	1×	1.5×	2×
Image	224×224				3	3	3	3
Conv1 MaxPool	112×112 56×56	3×3 3×3	2 2	1	24	24	24	24
Stage2	28×28 28×28		2 1	1 3	48	116	176	244
Stage3	14×14 14×14		2 1	1 7	96	232	352	488
Stage4	7×7 7×7		2 1	1 3	192	464	704	976
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.

ShuffleNet v2

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

ShuffleNet v2

Model	Complexity (MFLOPs)	Top-1 err. (%)	GPU Speed (Batches/sec.)	ARM Speed (Images/sec.)
ShuffleNet v2 1.5× (ours)	299	**27.4**	255	**11.8**
0.75 MobileNet v1 [13]	325	31.6	**314**	10.6
1.0 MobileNet v2 [14]	300	28.0	180	8.9
1.0 MobileNet v2 [14] (our impl.)	301	28.3	180	8.9
ShuffleNet v1 1.5× (g=3) [15]	292	28.5	164	10.3
DenseNet 1.5× [6] (our impl.)	295	39.9	274	9.7
CondenseNet (G=C=8) [16]	274	29.0	-	-
Xception 1.5× [12] (our impl.)	305	29.4	219	10.5
IGCV3-D [28]	318	27.8	102	6.3
ShuffleNet v2 2× (ours)	591	**25.1**	217	**6.7**
1.0 MobileNet v1 [13]	569	29.4	**247**	6.5
1.4 MobileNet v2 [14]	585	25.3	137	5.4
1.4 MobileNet v2 [14] (our impl.)	587	26.7	137	5.4
ShuffleNet v1 2× (g=3) [15]	524	26.3	133	6.4
DenseNet 2× [6] (our impl.)	519	34.6	197	6.1
CondenseNet (G=C=4) [16]	529	26.2	-	-
Xception 2× [12] (our impl.)	525	27.6	174	**6.7**
IGCV2-1.0 [27]	564	29.3	81	4.9
IGCV3-D (1.4) [28]	610	25.5	82	4.5

ShuffleNet v2

Model	Complexity (MFLOPs)	Top-1 err. (%)	GPU Speed (Batches/sec.)	ARM Speed (Images/sec.)
ShuffleNet v2 2x (ours, with *SE* [8])	597	**24.6**	**161**	**5.6**
NASNet-A [9] (4 @ 1056, our impl.)	564	26.0	130	4.6
PNASNet-5 [10] (our impl.)	588	25.8	115	4.1

Table 8: Comparison of several network architectures over classification error (on validation set, single center crop) and speed, on two platforms and four levels of computation complexity. Results are grouped by complexity levels for better comparison. The batch size is 8 for GPU and 1 for ARM. The image size is 224×224 except: [*] 160×160 and [**] 192×192. We do not provide speed measurements for *CondenseNets* [16] due to lack of efficient implementation currently.