ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design

Abstract.

Currently, the neural network architecture design is mostly guided by the **indirect metric** of computation complexity, i.e., FLOPs. However, the **direct metric**, e.g., speed, also depends on the other factors such as **memory access cost and platform characterics.** Thus, this work proposes to evaluate the direct metric on the target platform, **beyond** only considering FLOPs. Based on a series of controlled experiments, this work derives several practical guidelines for efficient network design. Accordingly, a new architecture is presented, called ShuffleNet V2. **Comprehensive ablation experiments verify** that our model is the state-of-the-art in terms of speed and accuracy **tradeoff**.

目前，神经网络架构的设计主要是以计算复杂性的间接指标，即FLOPs为指导。然而，直接指标，如速度，也取决于其他因素，如内存访问成本和平台特性。因此，这项工作提出在目标平台上评估直接指标，而不是只考虑FLOPs。基于一系列的控制性实验，这项工作得出了高效网络设计的几个实用指南。据此，提出了一个新的架构，称为ShuffleNet V2。综合消融实验验证了我们的模型在速度和精度的权衡方面是最先进的。

2. Introduction

The architecture of deep convolutional neutral networks (CNNs) has **evolved** for years, becoming more accurate and faster. Since **the milestone work of** AlexNet [1], the ImageNet classification accuracy has been significantly improved by novel structures, including VGG [2], GoogLeNet [3], ResNet [4,5], DenseNet [6], ResNeXt [7], SE-Net [8], and automatic neutral architecture search [9,10,11], to name a few.

深度卷积中性网络（CNN）的结构已经发展了多年，变得更加准确和快速。自从AlexNet[1]的里程碑式的工作以来，ImageNet的分类精度已经通过新颖的结构得到了显著的提高，包括VGG[2]、GoogLeNet[3]、ResNet[4,5]、DenseNet[6]、ResNeXt[7]、SE-Net[8]，以及自动中性架构搜索[9,10,11]，等等。

Besides accuracy, computation complexity is another important consideration. Real world tasks often aim at obtaining best accuracy under a limited computational budget, given by target platform (e.g., hardware) and application scenarios (e.g., auto driving requires low latency). This motivates a series of works towards light-weight architecture design and better speed-accuracy tradeoff, including Xception [12], MobileNet [13], MobileNet V2 [14], ShuffleNet [15], and CondenseNet [16], to name a few. Group convolution and depth-wise convolution are crucial in these works. To measure the computation complexity, a widely used metric is the number of float-point operations, or FLOPs1 . However, FLOPs is an indirect metric. It is an approximation of, but usually not equivalent to the direct metric that we really care about, such as speed or latency. Such discrepancy has been noticed in previous works [17,18,14,19]. For example, MobileNet v2 [14] is much faster than NASNET-A [9] but they have comparable FLOPs. This phenomenon is further exmplified in Figure 1(c)(d), which show that networks with similar FLOPs have different speeds. Therefore, using FLOPs as the only metric for computation complexity is insufficient and could lead to sub-optimal design.

除了准确性，计算的复杂性是另一个重要的考虑因素。现实世界的任务通常是在有限的计算预算下获得最佳精度，这是由目标平台（如硬件）和应用场景（如自动驾驶需要低延迟）所决定的。这促使了一系列的工作，包括Xception[12]、MobileNet[13]、MobileNet V2[14]、ShuffleNet[15]和CondenseNet[16]等，都是朝着轻量级架构设计和更好的速度-准确度权衡的方向发展。在这些作品中，分组卷积和深度卷积是至关重要的。为了衡量计算复杂性，一个广泛使用的指标是浮点运算的数量，即FLOPs1。然而，FLOPs是一个间接指标。它是一个近似值，但通常不等同于我们真正关心的直接指标，如速度或延迟。这种差异在以前的工作中已经被注意到了[17,18,14,19]。例如，MobileNet v2[14]比NASNET-A[9]快得多，但它们的FLOPs是相当的。这一现象在图1(c)(d)中得到了进一步的说明，这表明具有相似FLOPs的网络具有不同的速度。因此，使用FLOPs作为计算复杂性的唯一衡量标准是不够的，可能会导致次优设计。

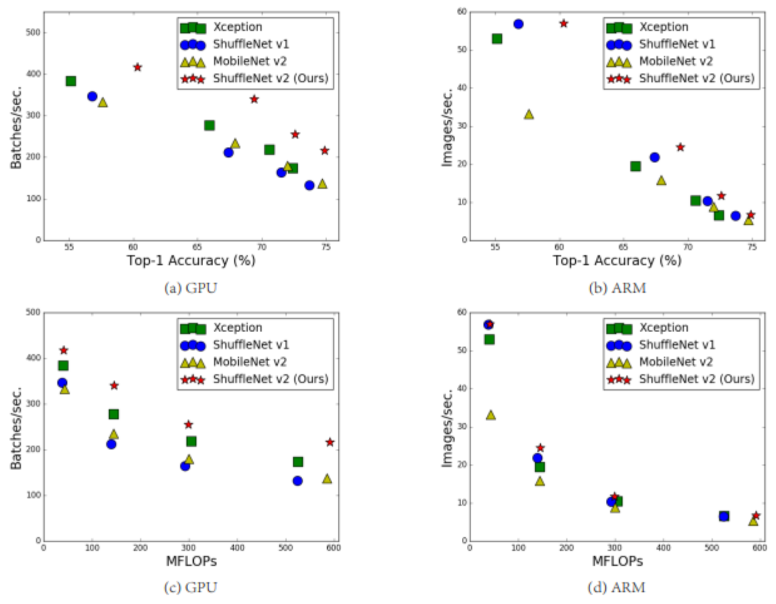


Fig. 1: Measurement of accuracy (ImageNet classification on validation set), speed and FLOPs of four network architectures on two hardware platforms with four different level of computation complexities (see text for details). (a, c) GPU results, $batchsize = 8$. (b, d) ARM results, $batchsize = 1$. The best performing algorithm, our proposed ShuffleNet v2, is on the top right region, under all cases.

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.

间接指标（FLOPs）和直接指标（速度）之间的差异可以归结为两个主要原因。首先，FLOPs没有考虑到对速度有相当影响的几个重要因素。其中一个因素是内存访问成本（MAC）。这种成本在某些操作（如分组卷积）中构成了运行时间的很大一部分。在具有强大计算能力的设备上，例如GPU，它可能是瓶颈。在网络架构设计中，这一成本不应简单地被忽视。另一个是平行度。在相同的FLOPs下，一个具有高平行度的模型可能比另一个具有低平行度的模型快很多。其次，同样的FLOPs的操作可能有不同的运行时间，这取决于平台。例如，在早期的工作中[20,21,22]，张量分解被广泛用于加速矩阵乘法。然

而，最近的工作[19]发现，[22]中的分解在GPU上更慢，尽管它减少了75%的FLOPs。我们调查了这个问题，发现这是因为最新的CUDNN[23]库是专门为3×3 conv优化的，我们不能肯定地认为3×3 conv比1×1 conv慢9倍。

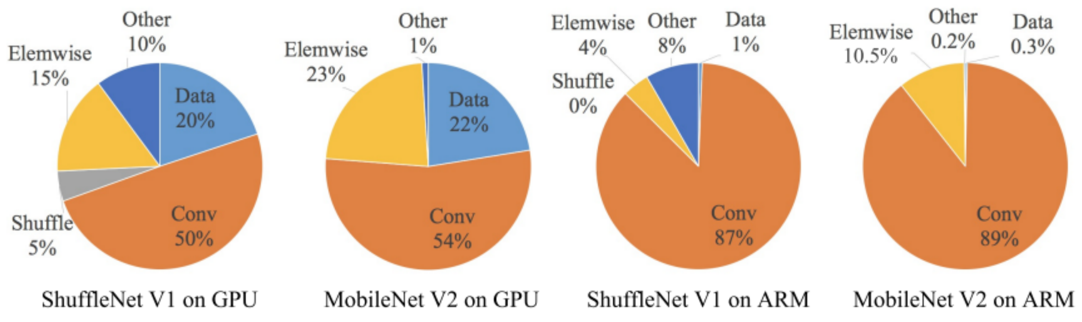


Fig. 2: Run time decomposition on two representative state-of-the-art network architectures, *ShuffeNet v1* [15] ($1\times$, $g = 3$) and *MobileNet v2* [14] ($1\times$).

With these observations, we propose that two principles should be considered for effective network architecture design. First, the direct metric (e.g., speed) should be used instead of the indirect ones (e.g., FLOPs). Second, such metric should be evaluated on the target platform.

有了这些观察，我们建议在进行有效的网络结构设计时应考虑两个原则。首先，应该使用直接指标（如速度）而不是间接指标（如FLOPs）。第二，这种指标应该在目标平台上进行评估。

In this work, we follow the two principles and propose a more effective network architecture. In Section 2, we firstly analyze the runtime performance of two representative state-of-the-art networks [15,14]. Then, we **derive four guidelines for efficient network design**, which are beyond only considering FLOPs. While these guidelines are platform independent, we perform a series of controlled experiments to validate them on two different platforms (GPU and ARM) with **dedicated code optimization,** ensuring that our conclusions are **state-of-the-art.** In Section 3, according to the guidelines, we design a new network structure. As it is inspired by ShuffleNet [15], it is called ShuffleNet V2. It is demonstrated much faster and more accurate than the previous networks on both platforms, via comprehensive validation experiments in Section 4. Figure 1(a)(b) gives an overview of comparison. For example, given the computation complexity budget of 40M FLOPs, ShuffleNet v2 is 3.5% and 3.7% more accurate than ShuffleNet v1 and MobileNet v2, respectively.

在这项工作中，我们遵循这两个原则，提出了一个更有效的网络架构。在第2节中，我们首先分析了两个有代表性的最先进的网络[15,14]的运行时间性能。然后，我们推导出高效网络设计的四个准则，这些准则超出了只考虑FLOPs的范围。虽然这些准则与平台无关，但我们通过专门的代码优化，在两个不同的平台（GPU和ARM）上进行了一系列受控实验来验证它们，确保我们的结论是最先进的。在第3节，根据指导方针，我们设计了一个新的网络结构。由于它受到ShuffleNet[15]的启发，所以被称为ShuffleNet V2。通过第4节中的综合验证实验，我们证明了它在两个平台上都比以前的网络更快、更准确。图1(a)(b)给出了一个比较的概况。例如，考虑到40M FLOPs的计算复杂度预算，ShuffleNet v2比ShuffleNet v1和MobileNet v2分别准确3.5%和3.7%。

2. Practical Guidelines for Efficient Network Design

Our study is performed on two widely adopted hardwares with industry-level optimization of CNN library. We note that our CNN library is more efficient than most open source libraries. Thus, we ensure that our observations and conclusions are solid and of significance for practice in industry

– GPU. A single NVIDIA GeForce GTX 1080Ti is used. The convolution library is CUDNN 7.0 [23]. We also activate the benchmarking function of CUDNN to select the fastest algorithms for different convolutions respectively.

 – ARM. A Qualcomm Snapdragon 810. We use a highly-optimized Neon-based implementation. A single thread is used for evaluation

Other settings include: full optimization options (e.g. **tensor fusion**, which is used to reduce the **overhead** of small operations) are switched on. The input image size is 224 × 224. Each network is randomly initialized and evaluated for 100 times. The average runtime is used.

其他设置包括：全部优化选项（例如张量融合，用于减少小操作的开销）被打开。输入图像大小为224×224。每个网络随机初始化并评估100次。使用的是平均运行时间。

To initiate our study, we analyze the runtime performance of two state-of-the-art networks, ShuffleNet v1 [15] and MobileNet v2 [14]. They are both highly efficient and accurate on ImageNet classification task. They are both widely used on **low end devices** such as mobiles. Although we only analyze these two networks, we note that they are representative for the current trend. At their core are group convolution and depth-wise convolution, which are also crucial components for other state-of-the-art networks, such as ResNeXt [7], Xception [12], MobileNet [13], and CondenseNet [16].

为了开始我们的研究，我们分析了两个最先进的网络，ShuffleNet v1 [15] 和 MobileNet v2 [14] 的运行性能。它们在ImageNet分类任务上都非常有效和准确。它们都被广泛用于低端设备，如移动电话。尽管我们只分析了这两个网络，但我们注意到它们对当前的趋势具有代表性。它们的核心是分组卷积和深度卷积，这也是其他最先进的网络的关键组成部分，如ResNeXt[7]、Xception[12]、MobileNet[13]和CondenseNet[16]。

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. Based on this observation, we perform a detailed analysis of runtime (or speed) from several different aspects and derive several practical guidelines for efficient network architecture design.

如图2所示，整体运行时间被分解为不同的操作。我们注意到，FLOPs指标只考虑了卷积部分。虽然这部分消耗了大部分时间，但其他操作包括数据输入/输出、数据洗牌和元素操作（AddTensor、ReLU等）也占用了大量时间。因此，FLOPs并不是对实际运行时间的足够准确的估计。基于这一观察，我们从几个不同的方面对运行时间（或速度）进行了详细的分析，并得出了高效网络架构设计的几个实用指南。

Based on this observation, we perform a detailed analysis of runtime (or speed) from several different aspects and derive several practical guidelines for efficient network architecture design.

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

 The modern networks usually adopt depthwise separable convolutions [12,13,15,14], where the pointwise convolution (i.e., 1 × 1 convolution) accounts for most of the complexity [15]. We study the kernel shape of the 1 × 1 convolution. The shape is specified by two parameters: the number of input channels c1 and output channels c2. Let h and w be the spatial size of the feature map, the FLOPs of the 1 × 1 convolution is B = hwc1c2.

 现代网络通常采用深度可分离卷积[12,13,15,14]，其中点状卷积（即1×1卷积）占了大部分的复杂性[15]。我们研究1×1卷积的内核形状。该形状由两个参数指定：输入通道c1和输出通道c2的数量。设h和w是特征图的空间大小，1×1卷积的FLOPs为B=hwc1c2。

For simplicity, we assume the cache in the computing device is large enough to store the entire feature maps and parameters. Thus, the memory access cost (MAC), or the number of memory access operations, is MAC = hw(c1+c2)+c1c2. Note that the two terms correspond to the memory access for input/output feature maps and kernel weights, respectively. From mean value inequality, we have

为简单起见，我们假设计算设备中的缓存足够大，可以存储整个特征图和参数。因此，内存访问成本（MAC），或内存访问操作的数量，为MAC=hw(c1+c2)+c1c2。注意，这两个词分别对应于输入/输出特征图和内核权重的内存访问。根据均值不等式，我们有

		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.

$$\text{MAC} \geq 2\sqrt{hwB} + \frac{B}{hw}. \tag{1}$$

Therefore, MAC has a lower bound given by FLOPs. It reaches the lower bound when the numbers of input and output channels are equal.

因此，MAC有一个由FLOPs给出的下限。当输入和输出通道的数量相等时，它就达到了下限。

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 c1 input channels and c2 output channels, and the second otherwise. Table 1 reports the running speed by varying the ratio c1 : c2 while fixing the total FLOPs. It is clear that when c1 : c2 is approaching 1 : 1, the MAC becomes smaller and the network evaluation speed is faster.

这个结论是理论上的。在实践中，许多设备上的缓存都不够大。而且，现代计算库通常采用复杂的阻塞策略来充分利用缓存机制[24]。因此，实际的MAC可能会偏离理论上的情况。为了验证上述结论，进行了如下实验。通过重复堆叠10个构建块来构建一个基准网络。每个模块包含两个卷积层。第一个包含c1输入通道和c2输出通道，第二个则不然。表1报告了在固定总FLOPs的情况下，通过改变c1：c2的比例而得到的运行速度。很明显，当c1 : c2接近1 : 1时，MAC变小，网络评估速度更快。

		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.

G2) Excessive group convolution increases MAC.

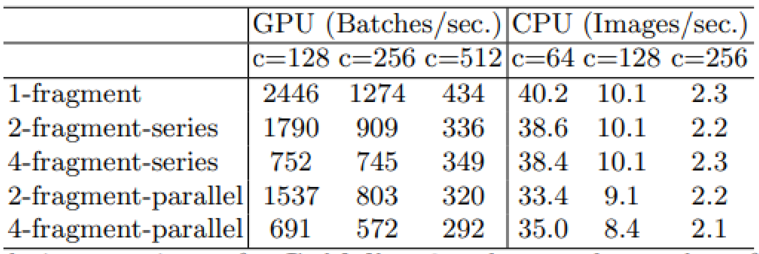
Group convolution is at the core of modern network architectures [7,15,25,26,27,28]. It reduces the computational complexity (FLOPs) by changing the dense convolution between all channels to be sparse (only within groups of channels). On one hand, it allows usage of more channels given a fixed FLOPs and increases the network capacity (thus better accuracy). On the other hand, however, the increased number of channels results in more MAC. Formally, following the notations in G1 and Eq. 1, the relation between MAC and FLOPs for 1 × 1 group convolution is

组卷积是现代网络架构的核心[7,15,25,26,27,28]。它通过将所有通道之间的密集卷积改为稀疏（仅在通道组内）来降低计算复杂性（FLOPs）。一方面，它允许在固定的FLOPs条件下使用更多的通道，并增加网络容量（从而提高精度）。然而，另一方面，通道数量的增加导致了更多的MAC。形式上，按照G1中的符号和公式1，1×1组卷积的MAC和FLOPs之间的关系是

$$\begin{aligned} \mathrm{MAC} &= hw(c_1 + c_2) + \frac{c_1 c_2}{g} \\ &= hwc_1 + \frac{Bg}{c_1} + \frac{B}{hw}, \end{aligned} \tag{2}$$

where g is the number of groups and B = hwc1c2/g is the FLOPs. It is easy to see that, given the fixed input shape c1 × h × w and the computational cost B, MAC increases with the growth of g. To study the affection in practice, a benchmark network is built by stacking 10 pointwise group convolution layers. Table 2 reports the running speed of using

其中g是组数，B = hwc1c2/g是FLOPs。很容易看出，在固定的输入形状c1×h×w和计算成本B的情况下，MAC随着g的增长而增加。为了研究实践中的affection，通过堆叠10个点式组卷积层建立一个基准网络。表2报告了使用以下方法的运行速度



	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.

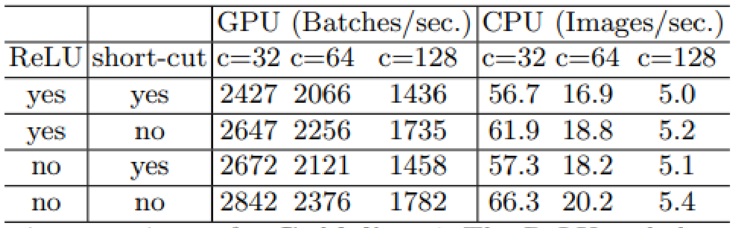
different group numbers while fixing the total FLOPs. It is clear that using a large group number decreases running speed significantly. For example, using 8 groups is more than two times slower than using 1 group (standard dense convolution) on GPU and up to 30% slower on ARM. This is mostly due to 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.

在固定总FLOPs的情况下，不同的组数。很明显，使用大组数会大大降低运行速度。例如，在GPU上使用8组比使用1组（标准密集卷积）慢2倍多，在ARM上慢30%。这主要是由于增加了MAC。我们注意到，我们的实现已经被特别优化，比逐组计算卷积要快得多。因此，我们建议，应该根据目标平台和任务仔细选择组数。仅仅因为可以使用更多的通道而使用大的组数是不明智的，因为精度提高的好处很容易被迅速增加的计算成本所抵消。

G3) Network fragmentation reduces degree of parallelism.

In the GoogLeNet series [29,30,3,31] and auto-generated architectures [9,11,10]), a “multipath” structure is widely adopted in each network block. A lot of small operators (called “fragmented operators” here) are used instead of a few large ones. For example, in NASNET-A [9] the number of fragmented operators (i.e. the number of individual convolution or pooling operations in one building block) is 13. In contrast, in regular structures like ResNet [4], this number is 2 or 3. 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. To quantify how network fragmentation affects efficiency, we evaluate a series of network blocks with different degrees of fragmentation. Specifically, eachbuilding block consists of from 1 to 4 1 × 1 convolutions, which are arranged in sequence or in parallel. The block structures are illustrated in appendix. Each block is repeatedly stacked for 10 times. Results in Table 3 show that fragmentation reduces the speed significantly on GPU, e.g. 4-fragment structure is 3× slower than 1-fragment. On ARM, the speed reduction is relatively small.

在GoogLeNet系列[29,30,3,31]和自动生成的架构[9,11,10]中，每个网络块广泛采用 "多路径 "结构。大量的小运算符（这里称为 "碎片化运算符"）被使用，而不是几个大运算符。例如，在NASNET-A[9]中，零散运算符的数量（即一个构件中单独的卷积或池化运算的数量）是13个。相比之下，在ResNet[4]这样的常规结构中，这个数字是2或3。尽管这种碎片化的结构已被证明有利于提高准确性，但它可能会降低效率，因为它对具有强大并行计算能力的设备如GPU不友好。它还引入了额外的开销，如内核启动和同步。为了量化网络碎片对效率的影响，我们评估了一系列具有不同碎片化程度的网络块。具体来说，每个构建块由1到4个1×1的卷积组成，它们依次或平行排列。块的结构在附录中有所说明。每个区块重复堆叠10次。表3中的结果显示，碎片化在GPU上大大降低了速度，例如，4个碎片结构比1个碎片的速度慢3倍。在ARM上，速度的降低相对较小。



		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.

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.

如图2所示，在像[15,14]这样的轻量级模型中，元素明智操作占据了相当多的时间，特别是在GPU上。这里，元素明智运算包括ReLU、AddTensor、AddBias等。它们的FLOPs很小，但MAC相对较重。特别是，我们还考虑将深度卷积[12,13,14,15]作为一个明智的元素运算，因为它也有一个高的MAC/FLOPs比率。

For validation, we experimented with the “bottleneck” unit (1 × 1 conv followed by 3×3 conv followed by 1×1 conv, with ReLU and shortcut connection) in ResNet [4]. The ReLU and shortcut operations are removed, separately. Runtime 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.

为了验证，我们用ResNet[4]中的 "瓶颈 "单元（1×1 conv之后是3×3 conv之后是1×1 conv，有ReLU和快捷连接）进行了实验。ReLU和快捷方式的操作被分别删除。表4中报告了不同变体的运行时间。我们观察到，在去除ReLU和快捷方式后，在GPU和ARM上都获得了大约20%的速度提升。

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.

基于上述准则和经验研究，我们得出结论，一个高效的网络架构应该：1）使用 "平衡的 "卷积（通道宽度相等）；2）意识到使用组卷积的成本；3）减少碎片化的程度；4）减少元素明智的操作。这些理想的特性取决于平台特性（如内存操作和代码优化），这些特性超出了理论上的FLOPs。在实际的网络设计中，应该考虑到这些特性。

Recent advances in light-weight neural network architectures [15,13,14,9,11,10,12] are mostly based on the metric of FLOPs and do not consider these properties above. For example, ShuffleNet v1 [15] heavily depends group convolutions (against G2) and bottleneck-like building blocks (against G1). MobileNet v2 [14] uses an inverted bottleneck structure that violates G1. It uses depthwise convolutions and ReLUs on “thick” feature maps. This violates G4. The auto-generated structures [9,11,10] are highly fragmented and violate G3.

最近在轻量级神经网络架构方面的进展[15,13,14,9,11,10,12]大多是基于FLOPs的度量，没有考虑上述这些特性。例如，ShuffleNet v1 [15]严重依赖分组卷积（针对G2）和类似瓶颈的构建块（针对G1）。MobileNet v2 [14]使用了一个违反G1的倒置瓶颈结构。它在 "厚 "的特征图上使用深度卷积和ReLU。这违反了G4。自动生成的结构[9,11,10]是高度碎片化的，违反了G3。

3. ShuffleNet V2: an Efficient Architecture

Review of ShuffleNet v1 [15].

ShuffleNet is a state-of-the-art network architecture. It is widely adopted in low end devices such as mobiles. It inspires our work. Thus, it is reviewed and analyzed at first.

ShuffleNet是一个最先进的网络架构。它被广泛采用于低端设备，如移动电话。它启发了我们的工作。因此，首先对它进行了审查和分析。

According to [15], the main challenge for light-weight networks is that only a limited number of feature channels is affordable under a given computation budget (FLOPs). To increase the number of channels without significantly increasing FLOPs, two techniques are adopted in [15]: pointwise group convolutions and bottleneck-like structures. A “channel shuffle” operation is then introduced to enable information communication between different groups of channels and improve accuracy. The building blocks are illustrated in Figure 3(a)(b).

根据[15]，轻量级网络的主要挑战是，在给定的计算预算（FLOPs）下，只能负担有限数量的特征通道。为了在不显著增加FLOPs的情况下增加通道的数量，[15]中采用了两种技术：点分组卷积和类似瓶颈的结构。然后引入了 "通道洗牌 "操作，以实现不同通道组之间的信息交流并提高精度。图3(a)(b)说明了这些构建模块。

As discussed in Section 2, both pointwise group convolutions and bottleneck structures increase MAC (G1 and G2). This cost is non-negligible, especially for light-weight models. Also, using too many groups violates G3. The element-wise “Add” operation in the shortcut connection is also undesirable (G4). Therefore, in order to achieve high model capacity and efficiency, the key issue is how to maintain a large number and equally wide channels with neither dense convolution nor too many groups.

正如第2节中所讨论的，点状分组卷积和瓶颈结构都会增加MAC（G1和G2）。这个代价是不可忽略的，尤其是对于轻量级的模型。另外，使用过多的组也违反了G3。捷径连接中的元素明智的 "添加 "操作也是不可取的（G4）。因此，为了实现较高的模型容量和效率，关键问题是如何在既不密集卷积也不太多组的情况下保持大量和同样宽的通道。

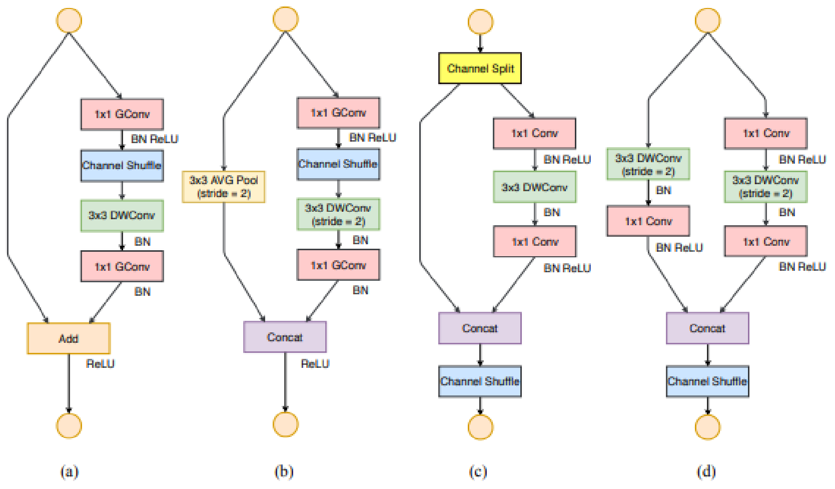


Fig. 3: Building blocks of ShuffleNet v1 [15] and this work. (a): the basic ShuffleNet unit; (b) the ShuffleNet unit for spatial down sampling (2×); (c) our basic unit; (d) our unit for spatial down sampling (2×). **DWConv**: depthwise convolution. **GConv**: group convolution.

Channel Split and ShuffleNet V2

Towards above purpose, we introduce a simple operator called channel split. It is illustrated in Figure 3(c). At thebeginning of each unit, the input of c feature channels are split into two branches with c − c 0 and c 0 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. After convolution, the two branches are concatenated. So, the number of channels keeps the same (G1). The same “channel shuffle” operation as in [15] is then used to enable information communication between the two branches.

为了达到上述目的，我们引入了一个简单的算子，称为信道分割。它在图3(c)中得到说明。在每个单元的开始，c特征通道的输入被分成两个分支，分别是c - c 0和c 0通道。在G3之后，一个分支仍然是特性。另一个分支由三个具有相同输入和输出通道的卷积组成，以满足G1。两个1×1的卷积不再是分组的，这与[15]不同。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。在卷积之后，两个分支被串联起来。所以，通道的数量保持不变（G1）。然后使用与[15]相同的 "信道洗牌 "操作来实现两个分支之间的信息交流。

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 depthwise 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.

洗牌之后，下一个单元开始。请注意，ShuffleNet v1[15]中的 "添加 "操作已经不存在了。像ReLU和深度卷积这样的逐元操作只存在于一个分支。另外，三个连续的元素操作，"Concat"，"Channel Shuffle "和 "Channel Split"，被合并为一个单一的元素操作。根据G4，这些变化是有益的。

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.

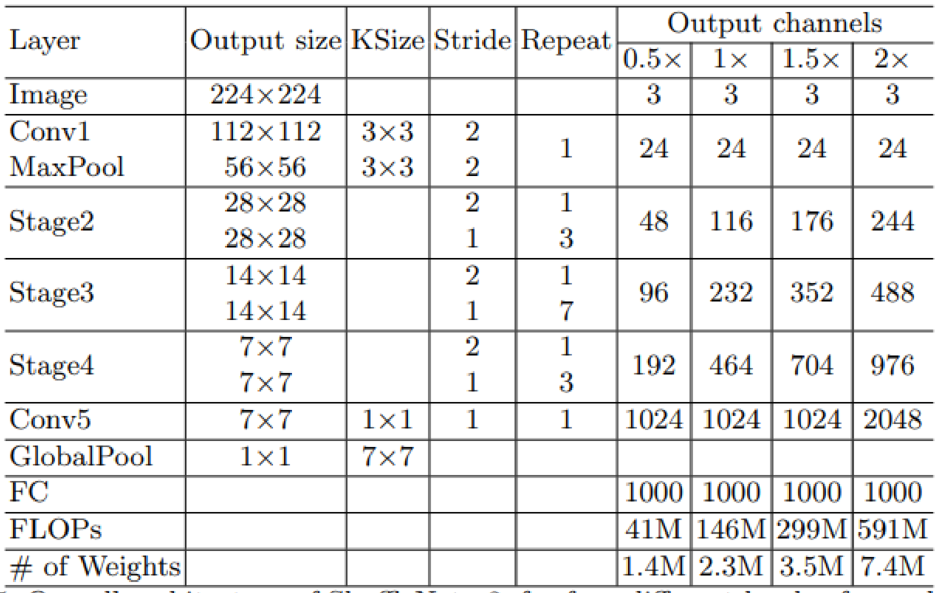
The proposed building blocks (c)(d), as well as the resulting networks, are called ShuffleNet V2. Based the above analysis, we conclude that this architecture design is highly efficient as it follows all the guidelines.

对于空间向下采样，该单元稍作修改，如图3（d）所示。通道分割运算符被删除。因此，输出通道的数量增加了一倍。

所提出的构件（c）（d），以及由此产生的网络，被称为ShuffleNet V2。基于上述分析，我们得出结论，这个架构设计是非常有效的，因为它遵循了所有的准则。

The building blocks are repeatedly stacked to construct the whole network. For simplicity, we set c 0 = c/2. The overall network structure is similar to Shuf□fleNet v1 [15] and summarized in Table 5. There is only one difference: an ad□ditional 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×, 1×, etc.

这些积木被反复堆叠以构建整个网络。为了简单起见，我们设定c 0 = c/2。整个网络结构与ShuffleNet v1[15]相似，总结在表5中。只有一个区别：在全局平均池化之前增加了一个额外的1×1卷积层来混合特征，这在ShuffleNet v1中是没有的。与[15]类似，每个块中的通道数量被缩放以产生不同复杂度的网络，标记为0.5×，1×等。



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.

Analysis of Network Accuracy

ShuffleNet v2 is not only efficient, but also accurate. There are two main reasons. First, the high efficiency in each building block enables using more feature channels and larger network capacity.

Second, in each block, half of feature channels (when c 0 = c/2) directly go through the block and join the next block. This can be regarded as a kind of feature reuse, in a similar spirit as in DenseNet [6] and CondenseNet [16].

In DenseNet[6], to analyze the feature reuse pattern, the l1-norm of the weights between layers are plotted, as in Figure 4(a). It is clear that the connections between the adjacent layers are stronger than the others. This implies that the dense connection between all layers could introduce redundancy. The recent CondenseNet [16] also supports the viewpoint.

In ShuffleNet V2, it is easy to prove that the number of “directly-connected” channels between i-th and (i+j)-th building block is r j c, where r = (1−c 0)/c. In other words, the amount of feature reuse decays exponentially with the distance between two blocks. Between distant blocks, the feature reuse becomes much weaker. Figure 4(b) plots the similar visualization as in (a), for r = 0.5. Note that the pattern in (b) is similar to (a).

ShuffleNet v2不仅高效，而且准确。主要有两个原因。首先，每个构件的高效率使其能够使用更多的特征通道和更大的网络容量。

第二，在每个区块中，一半的特征通道（当c 0 = c/2时）直接穿过区块并加入下一个区块。这可以看作是一种特征重用，与DenseNet[6]和CondenseNet[16]的精神类似。

在DenseNet[6]中，为了分析特征重用模式，绘制了层间权重的l1-norm，如图4（a）。很明显，相邻层之间的连接要比其他层强。这意味着所有层之间的密集连接可能会引入冗余。最近的CondenseNet[16]也支持这个观点。

在ShuffleNet V2中，很容易证明第i个和（i+j）个构件之间 "直接连接 "的通道数量为r j c，其中r=（1-c 0 ）/c。换句话说，特征重用的数量随着两个区块之间的距离呈指数级衰减。在遥远的区块之间，特征重用变得更弱。图4(b)是与(a)类似的可视化图，r=0.5。请注意，（b）中的模式与（a）相似。

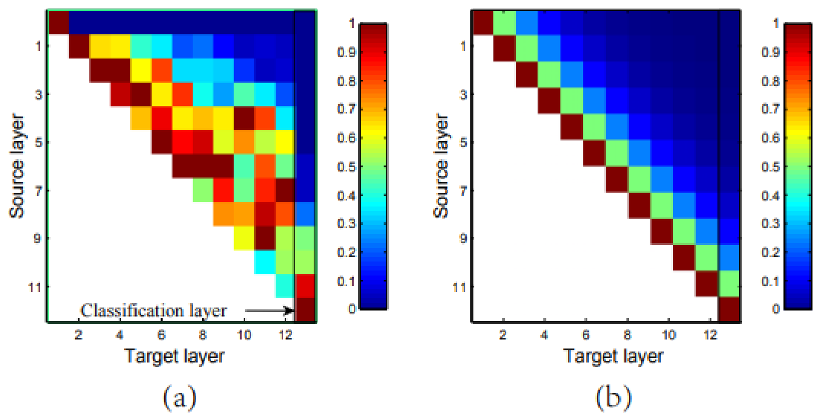


Fig. 4: Illustration of the patterns in feature reuse for *DenseNet* [6] and ShuffleNet V2. (a) (courtesy of [6]) the average absolute filter weight of convolutional layers in a model. The color of pixel (s, l) encodes the average $l1$-norm of weights connecting layer s to l. (b) The color of pixel (s, l) means the number of channels *directly* connecting block s to block l in ShuffleNet v2. All pixel values are normalized to $[0, 1]$.

Thus, the structure of ShuffleNet V2 realizes this type of feature re-use pat□tern by design. It shares the similar benefit of feature re-use for high accuracy as in DenseNet [6], but it is much more efficient as analyzed earlier. This is verified in experiments, Table 8.

因此，ShuffleNet V2的结构通过设计实现了这种类型的特征重用模式。它与DenseNet[6]一样具有特征重用以获得高精确度的类似好处，但正如前面分析的那样，它的效率要高很多。这在实验中得到了验证，表8。