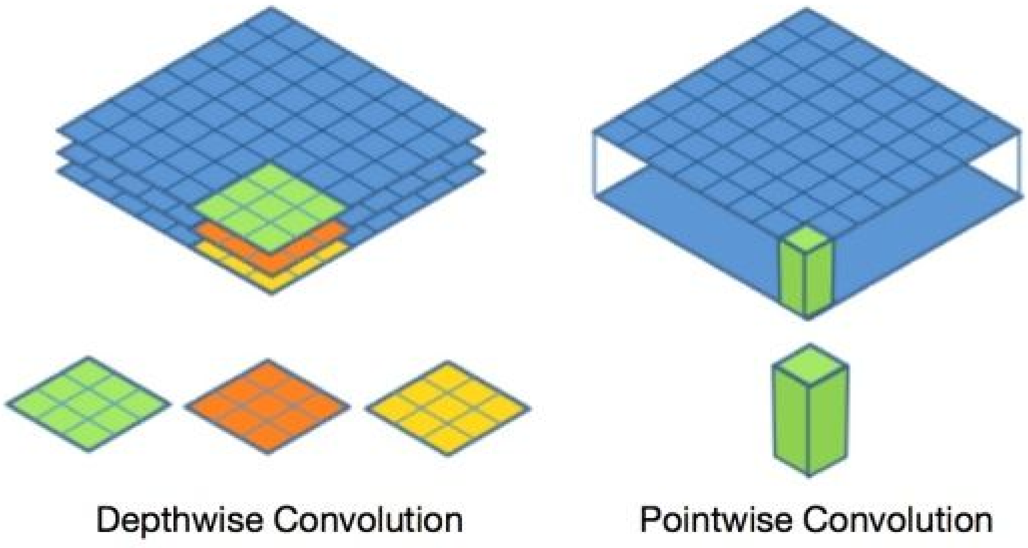
Evolution of MobileNets (and ShuffleNets?)

MobileNets are the benchmarks in lightweight networks. They have relative small sizes (less parameters than others) and comparable efficiencies, yet they don't have too much complicated combinations across blocks.

Besides, we will also cover ShuffleNets, which bring many creative and valuable ideas to this area.

MobileNetV1

In 2017, Google proposed the first generation MobileNet: [MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications](). It introduces a new type of convolution, namely ***Depthwise Separable Convolution***.



As we can see from the figure, its computation cost equals:

$$\begin{aligned} DepthwiseSeparableConv &= DepthwiseConv + PointwiseConv \\ &= (N*N*K*K*C_{in}) + (N*N*C_{in}*C_{out}) \end{aligned}$$

compared with ***Standard Convolution***'s':

$$StandardConv = N*N*K*K*C_{in}*C_{out}$$

and the ratio is:

$$Ratio = C_{out} + K*K$$

Normally $K = 3\ or\ 5$, but C_{out} could be much bigger, a significant speedup in convolutions! With this, we can build rather efficient blocks and stack them to be **MobileNetV1**.

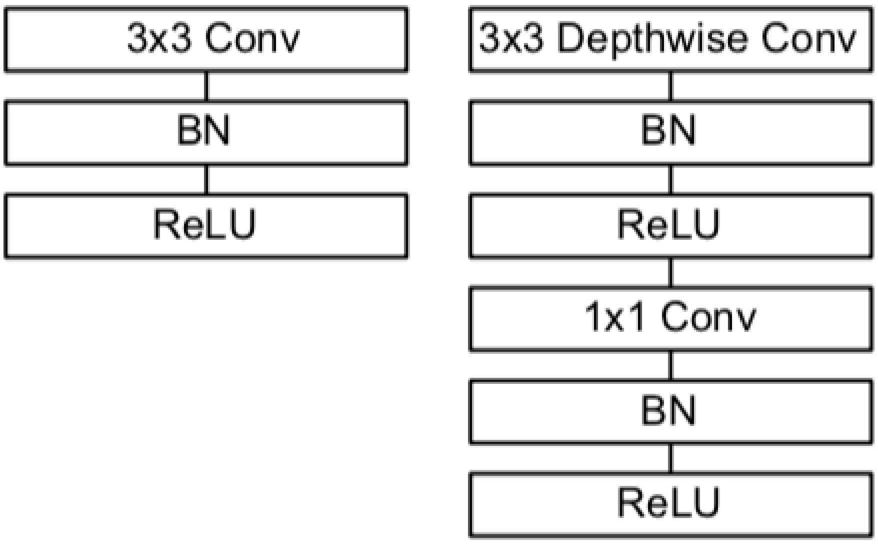
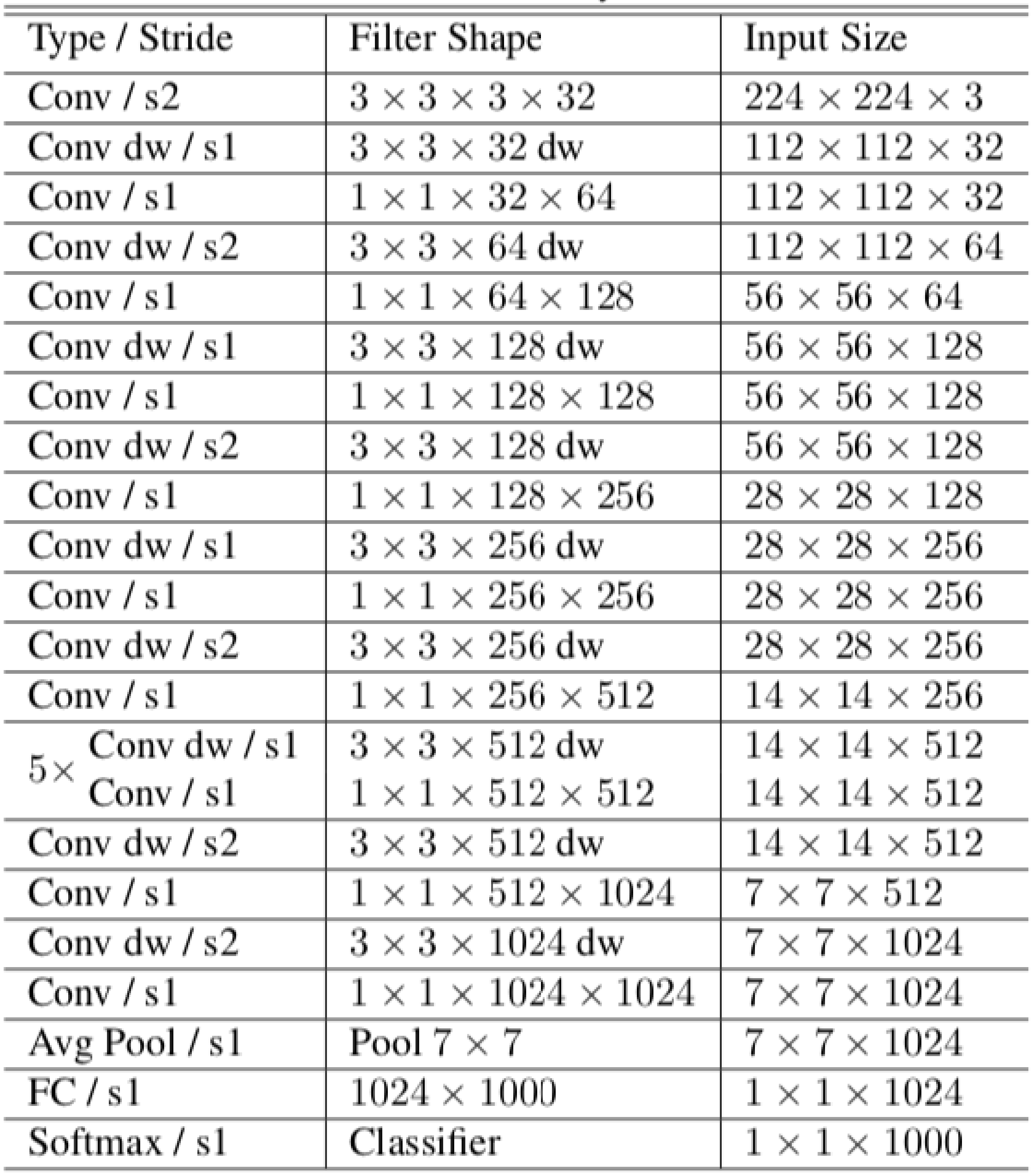


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

Table 1. MobileNet Body Architecture



Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

In addition, two useful hyperparameters were also introduced: ***Width Multiplier*** α and ***Resolution Multiplier*** ρ. The former mainly control ***the number of output channels***, and the latter is to reduce ***input images' sizes*** (224, 192, 160...).

ShuffleNetV1

A few months later, Face++ introduced another efficient network ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices, which employs two novel operations, ***Channel Shuffle*** and ***Pointwise Group Convolution***, and it outperforms **MobileNetV1** on *ImageNet classification task*.

The metric issue must be emphasized. At that time, we were using a proxy metric ***MFLOP*** to measure computation cost, not using direct metric ***Speed***. As you can imagine, some operations' costs may be **underestimated**, like ***Channel Shuffle*** (extra overheads in memory copy) .

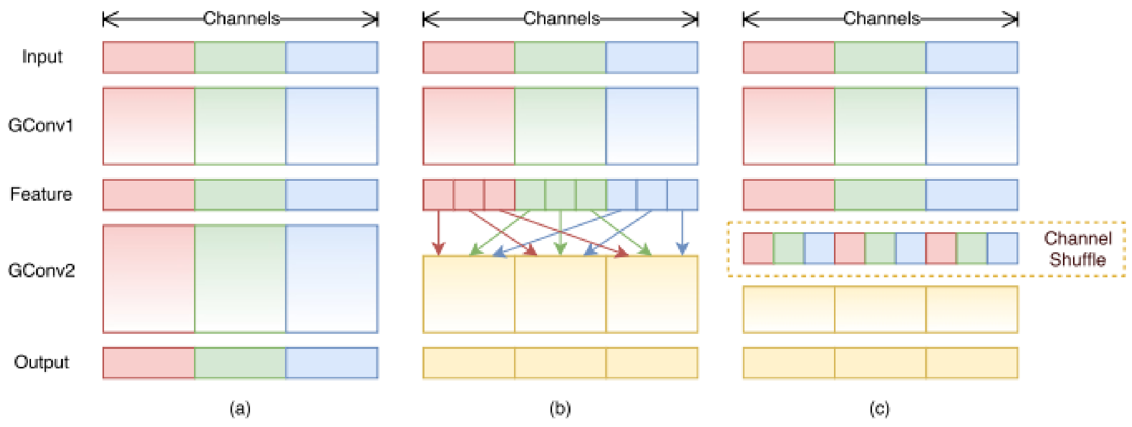


Figure 1. Channel shuffle with two stacked group convolutions. GConv stands for group convolution. a) two stacked convolution layers with the same number of groups. Each output channel only relates to the input channels within the group. No cross talk; b) input and output channels are fully related when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

For ***Group Gonvolution***, the authors split channels into different groups, then do convolutions separately. ***Channel Shuffle*** is the bridge across convolution groups, making it possible to utilize information from other groups (emmm... just sounds reasonable).

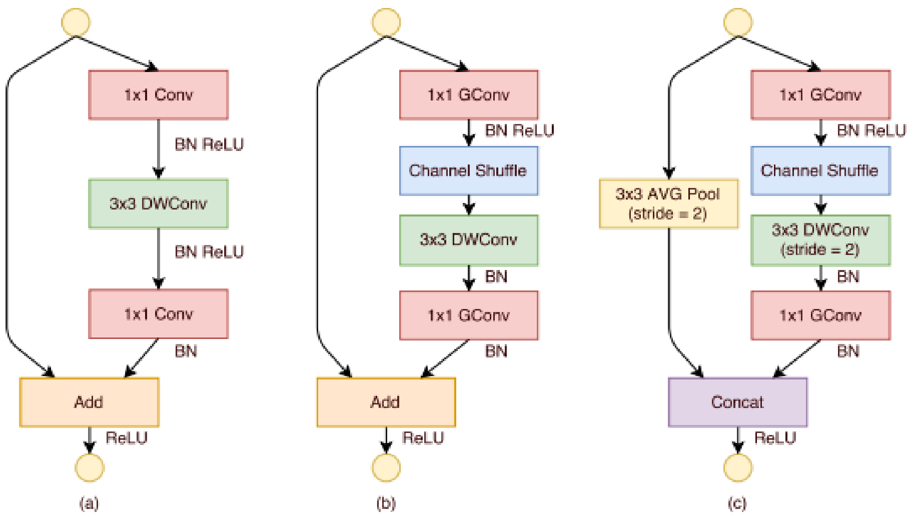
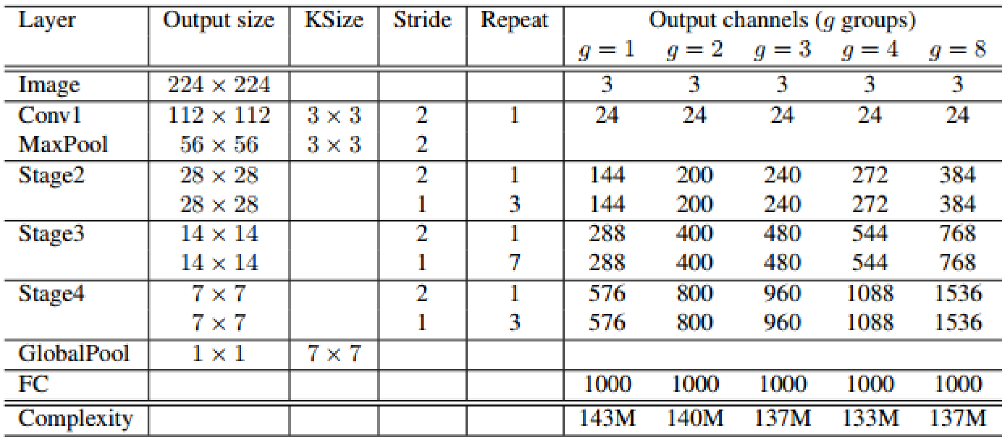


Figure 2. ShuffleNet Units. a) bottleneck unit [9] with depthwise convolution (DWConv) [3, 12]; b) ShuffleNet unit with pointwise group convolution (GConv) and channel shuffle; c) ShuffleNet unit with stride = 2.

There are two types of Units for different ***stride*** (b=1 or c=2), and they are quite similar to **ResNet Units**. We can stack them to be an efficient network.



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.

We should also note that ***channel shuffle*** is not a built-in operation in many frameworks, so you may need to combine ***slice*** and ***gather*** operations to mimic it, and be aware of its cost.

MobileNetV2

In January 2018, MobileNetV2: Inverted Residuals and Linear Bottlenecks was proposed. The key feature is **Inverted Residual Structure** where the shortcut connections are between the thin bottleneck layers. Within this structure, the intermediate expansion layer uses lightweight depthwise convolutions followed by non-linearity activations. Meanwhile, they found non-linearities should not be used in the narrow layers.