

验证GhostNet在移动端设备上使用的可行性

具体内容：将自己复现的GhostNet模型和已有的MobileNet系列，EfficientNet系列，MnasNet，ShuffleNet系列等轻量级网络做比较，主要比较一下几个方面：

- 1.比较网络结构；
- 2.比较模型的大小、parameter参数数量，以及FLOPs运算次数；
- 3.在ImageNet数据集上，比较模型的性能（精确度和运行时间）；

一、网络结构对比

1.GhostNet

(1)ghost module

考虑到主流CNNs计算的中间特征图中广泛存在的冗余，文章提出减少所需资源，即生成所需资源的卷积滤波器。

卷积层生成特征图可以写成公式：

$$Y = X * f + b$$

其中 $X \in R^{c \times h \times w}$ 是输入数据， c 是输入数据的通道， h, w 分别为输入数据的高和宽，如果产生 n 个特征图，则 $Y \in R^{h' \times w' \times n}$ ，对于的卷积核 $f \in R^{c \times k \times k \times n}$ ，这个卷积操作需要的FLOPs为 $n \cdot h' \cdot w' \cdot c \cdot k \cdot k$ ，一般滤波器个数 n 和通道数 c 基本上都很大，所以导致一个卷积层的FLOPs比较大。

$$Y = X * f + b, X \in R^{c \times h \times w}, Y \in R^{h' \times w' \times n}, f \in R^{c \times k \times k \times n}$$

卷积层输出的特征图包含了非常多的冗余，当中有一些非常相似，所以没必要通过这么多FLOPs和参数的卷积操作去一个个的生成这些特征图，而是假设这些输出的特征图是一些少数固有特征图和他们们的“伪影”(ghosts，通过很少的变换产生的)，这些固有特征图是由传统的卷积产生的。

假设输出的特征图中，有 m 个固有特征图， $Y' \in R^{h' \times w' \times m}$ 是通过传统的卷积操作得到：

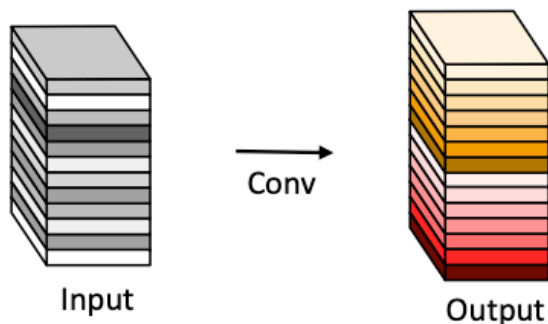
$$Y' = X * f'$$

其中 $f' \in R^{c \times k \times k \times m}$ 是传统运算的卷积核， $m < n$ ，为了简单起见，省略了偏执项，并且filter size, stride, padding等参数与之前的普通卷积保持一致。

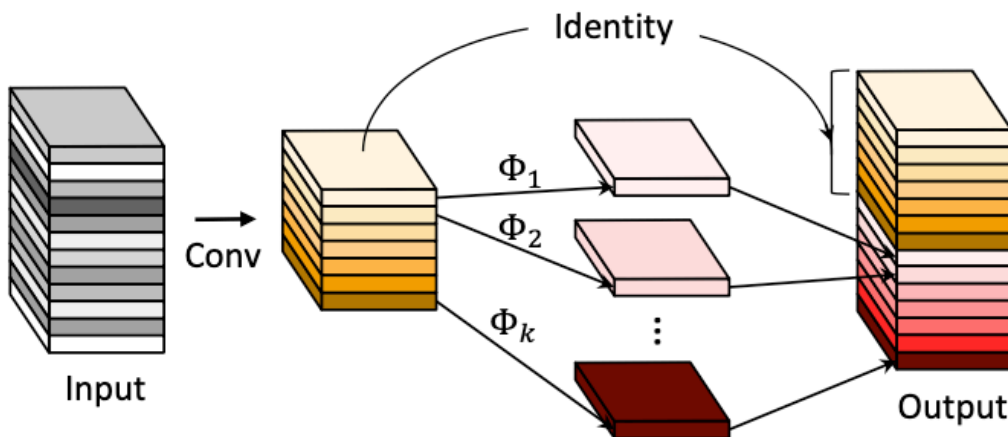
为了进一步得到相同的 n 个特征图，利用下面的简单的线性映射在固有特征图 Y' 上产生 s 个Ghost特征图：

$$y_{ij} = \Phi_{i,j}(y'_i), \forall i = 1, \dots, m, j = 1, \dots, s$$

其中 y'_i 是固有特征图 Y' 中第 i 个特征图， $\Phi_{i,j}$ 是生成第 j 个特征图 y_{ij} 的线性算子，也就是说对于固有特征图 y'_i 会有多个ghost特征图 $\{y_{ij}\}_{j=1}^s$ ，且 $\Phi_{i,s}$ 是恒等映射用来保留固有特征图（恒等映射和线性算子并行，保留了固有特征图）。通过上述这些线性映射可以得到最终的输出特征图 $Y = [y_{11}, y_{12}, \dots, y_{ms}]$ ，通道数 $n = m \cdot s$ 。



(a) The convolutional layer.



(b) The Ghost module.

Figure 2. An illustration of the convolutional layer and the proposed Ghost module for outputting the same number of feature maps. Φ represents the cheap operation.

(2) Ghost Bottlenecks

Ghost Bottlenecks由ghost模块构成，整体布局和ResNet中的Bottlenecks非常类似，ResNet-50中也有两种类型的Bottlenecks，一种是主路三个卷积层，然后加一个shortcut；另一种是主路三个卷积层，然后shortcut上再加一个卷积层。这里也提出了两种Ghost Bottlenecks。

第一种提出的Ghost Bottlenecks主要由两个堆叠的ghost模块组成。第一个ghost模块作为一个扩展层增加通道的数量。我们把输出通道数与输入通道数之比称为扩展比。第二个ghost模块减少通道的数量以匹配shortcut。然后将这两个ghost模块的输入和输出连接起来。

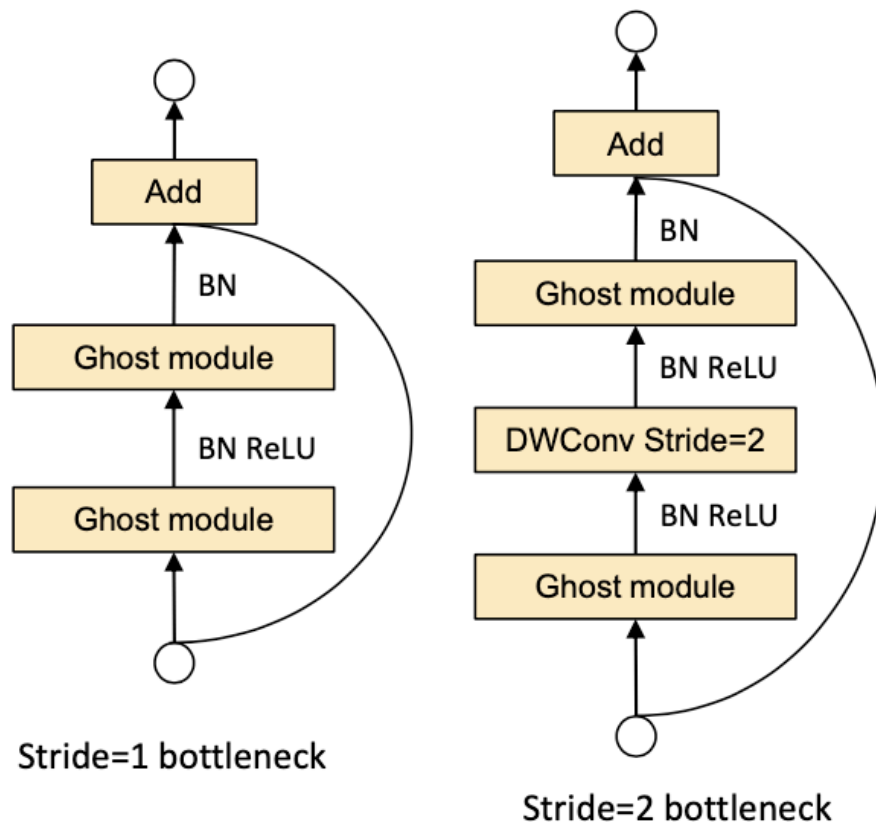


Figure 3. Ghost bottleneck. Left: Ghost bottleneck with stride=1; right: Ghost bottleneck with stride=2.

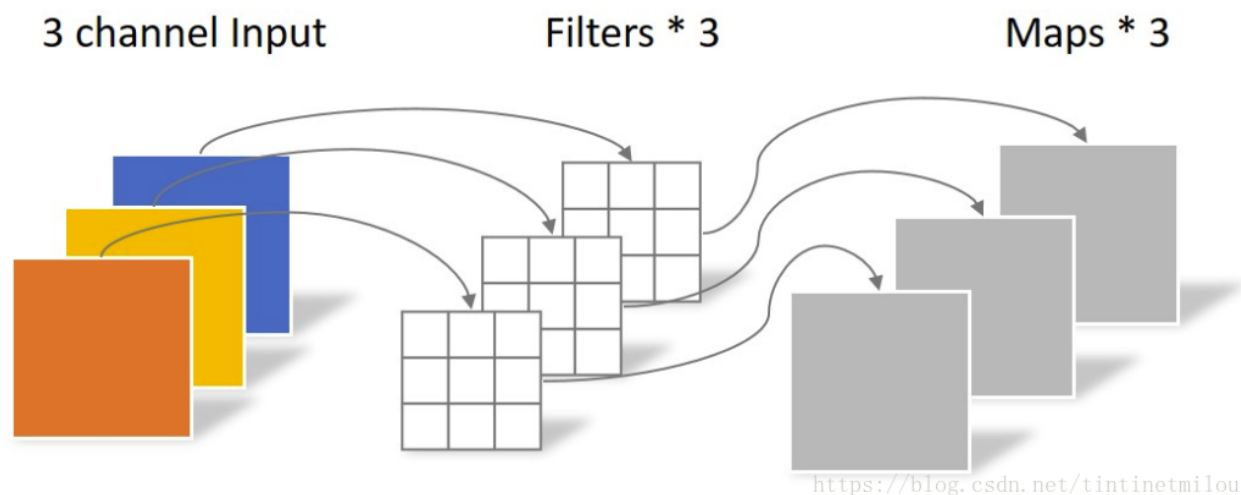
第二种提出的Ghost Bottlenecks与第一种不一样的，在中间加了一个stride为2的depthwise convolution，且shortcut这一条路是下采样的结果，并不是恒等映射。在所提出的两个Ghost Bottlenecks中，ghost模块中的基础卷积都是pointwise convolution，为了提高效率。

2.MobileNetV1

MobileNetV1将传统卷积的过程改进为深度可分离卷积，相比于常规卷积操作，其参数量和运算成本较低，具体分为两个阶段：

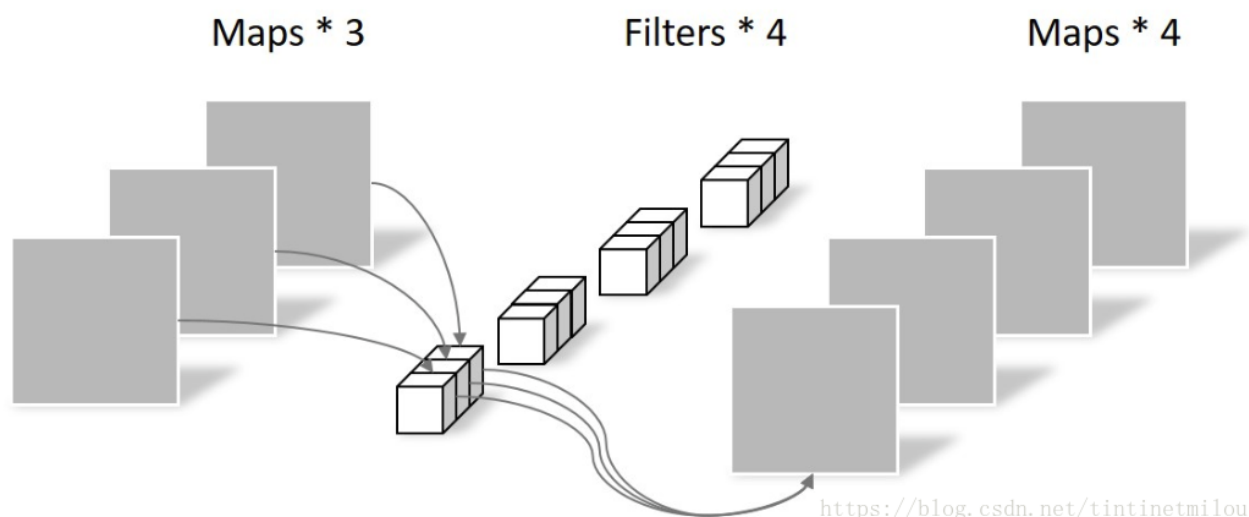
(1) depthwise convolution

不同于常规卷积操作，Depthwise Convolution的一个卷积核负责一个通道，一个通道只被一个卷积核卷积。上面所提到的常规卷积每个卷积核是同时操作输入图片的每个通道。Depthwise Convolution完成后的Feature map数量与输入层的通道数相同，无法扩展Feature map。而且这种运算对输入层的每个通道独立进行卷积运算，没有有效的利用不同通道在相同空间位置上的feature信息。因此需要Pointwise Convolution来将这些Feature map进行组合生成新的Feature map，即只改变图像大小，不改变通道数。



(2) pointwise convolution

Pointwise Convolution的运算与常规卷积运算非常相似，它的卷积核的尺寸为 $1 \times 1 \times M$ ， M 为上一层的通道数。所以这里的卷积运算会将上一步的map在深度方向上进行加权组合，生成新的Feature map。有几个卷积核就有几个输出Feature map，即不改变大小，只改变通道数。



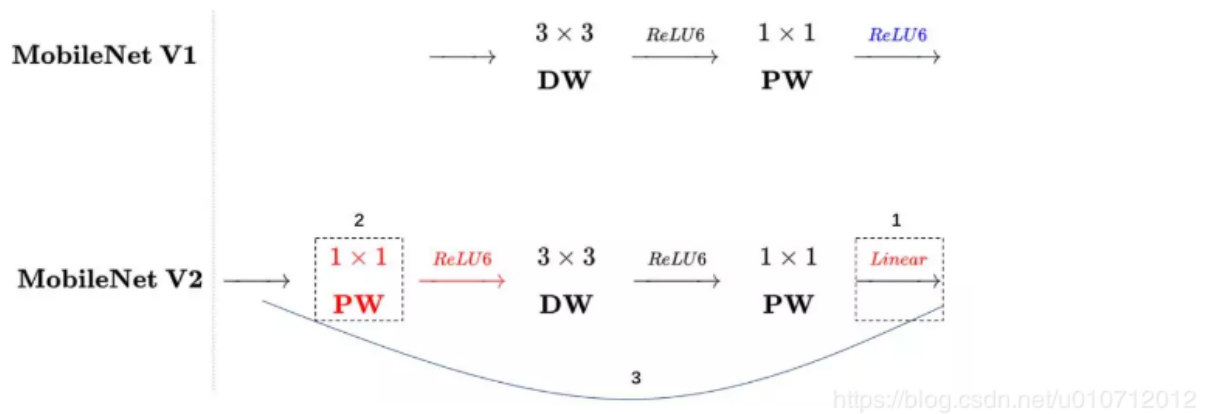
Ghost Bottlenecks总结：联系后续的GhostNet结构发现，stride为1的Ghost Bottlenecks，在后续使用中输出和输入尺寸完全一样；stride为2的Ghost Bottlenecks，在后续使用中输出和输入尺寸减半；这是因为stride为1的Ghost Bottlenecks中基础卷积使用的是pointwise convolution，只改变通道数，不改变大小；stride为2的Ghost Bottlenecks，中间还隔着一层depthwise convolution，不改变通道数，但是会改变大小（减半），所以总体尺寸是减半的，因为前后两个ghost 模块中的卷积是pointwise convolution，不改变大小。

3.MobileNetV2

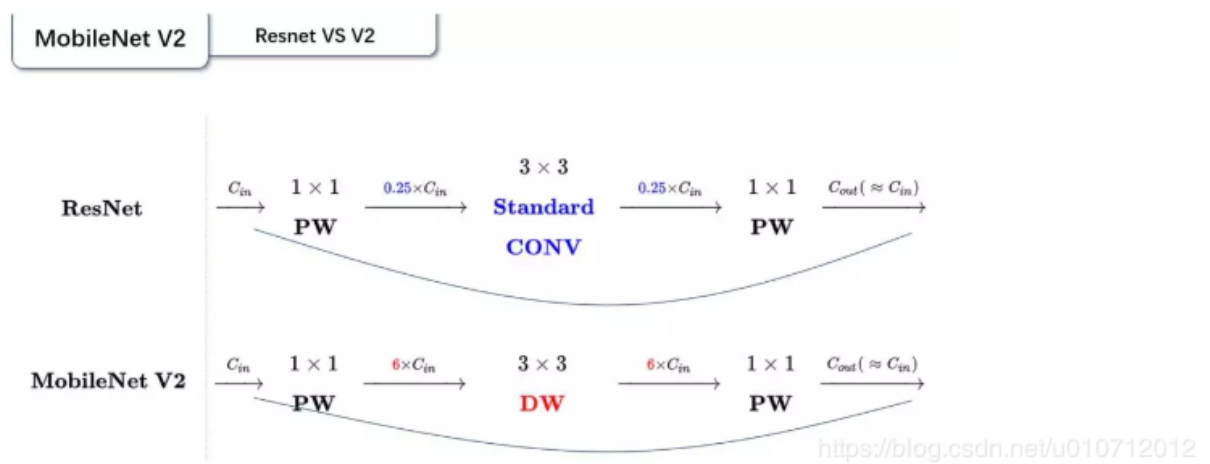
1) depthwise convolution本身没有改变通道的能力，来的是多少通道输出就是多少通道。如果来的通道很少的话，DW深度卷积只能在低维度上工作，这样效果并不会很好，所以MobileNetV2“扩张”通道。PW逐点卷积也就是 1×1 卷积可以用来升维和降维，那就可以在DW深度卷积之前使用PW卷积进行升维（升维倍数为 t ， $t=6$ ），再在一个更高维的空间中进行卷积操作来提取特征；

2) 回顾MobileNetV1的网络结构，V1像是一个直筒型的VGG网络。MobileNetV2引入了shortcut结构；

可以对比下MobileV1,V2:



以及MobileV2,ResNet:

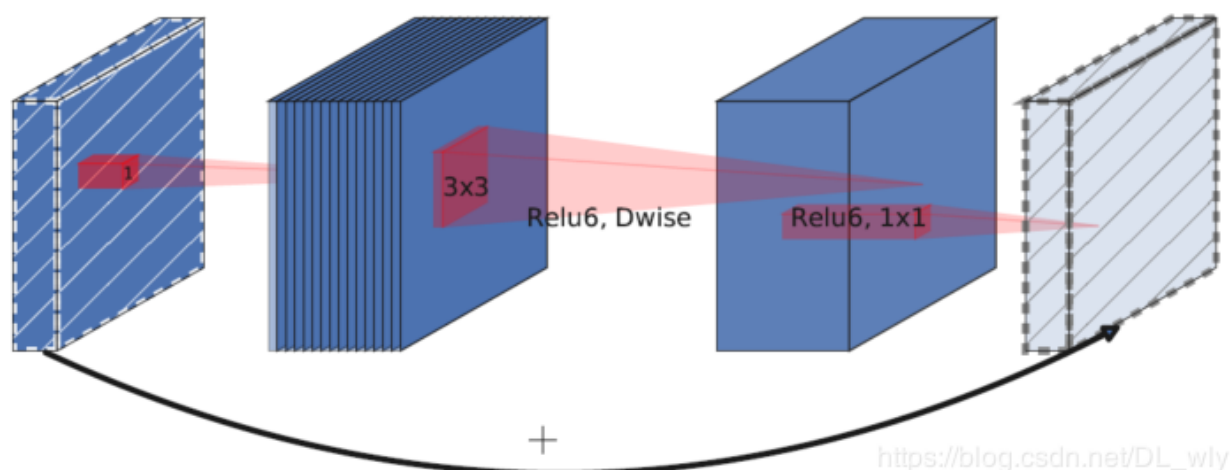


可以发现，都采用了 $1 \times 1 \rightarrow 3 \times 3 \rightarrow 1 \times 1$ 的模式，以及都使用Shortcut结构。但是不同点：
ResNet 先降维 (0.25倍)、卷积、再升维。
MobileNetV2 则是 先升维 (6倍)、卷积、再降维。
刚好MobileNetV2的block刚好与Resnet的block相反，文章作者将其命名为Inverted residuals。

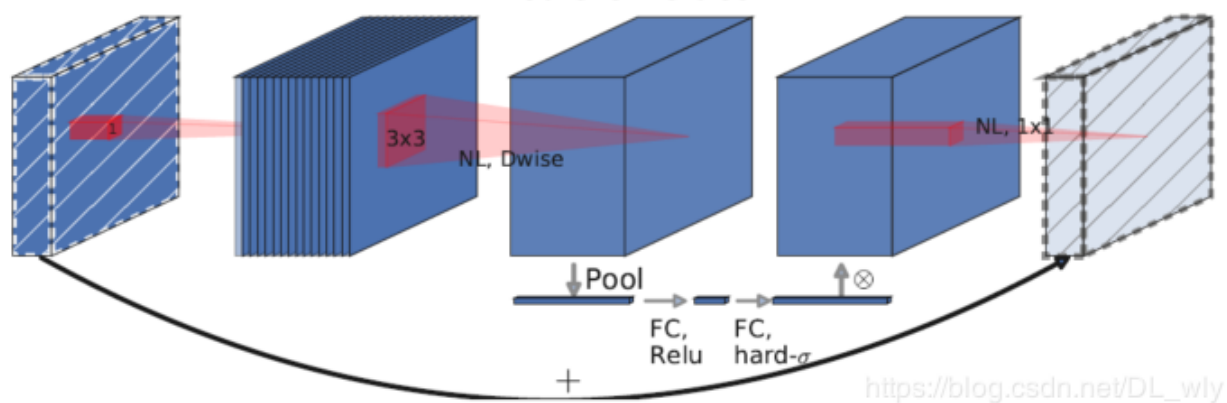
4.MobileNetV3

MobileNetV2和MobileNetV3的结构对比：

Mobilenet V2: bottleneck with residual



Mobilenet V3 block

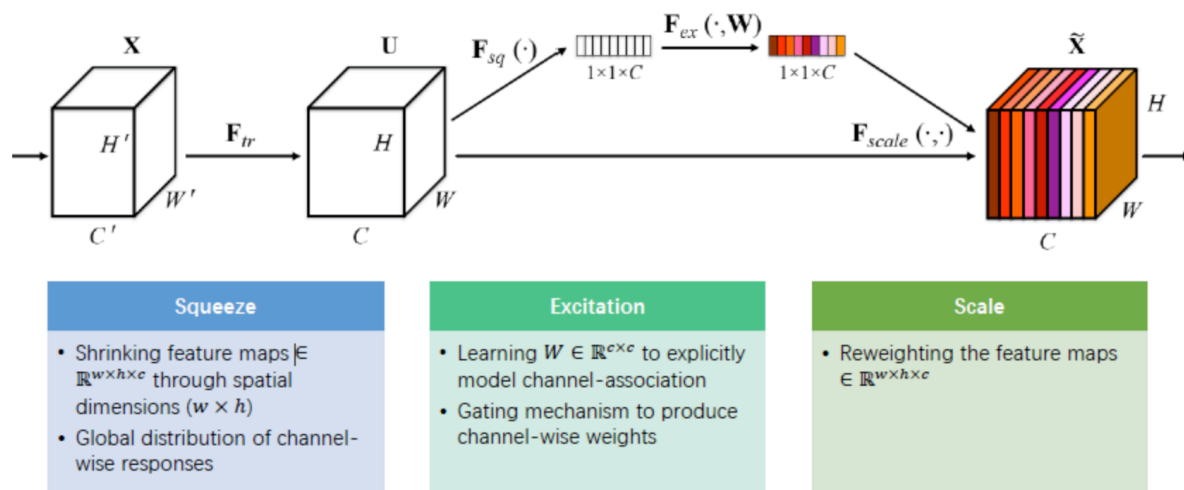


上面两张图是MobileNetV2和MobileNetV3的网络块结构。可以看出，MobileNetV3是综合了以下三种模型的思想：MobileNetV1的深度可分离卷积（depthwise separable convolutions）、MobileNetV2的具有线性瓶颈的逆残差结构(the inverted residual with linear bottleneck)和SENet的基于squeeze and excitation结构的轻量级注意力模型。

SE module 结构:

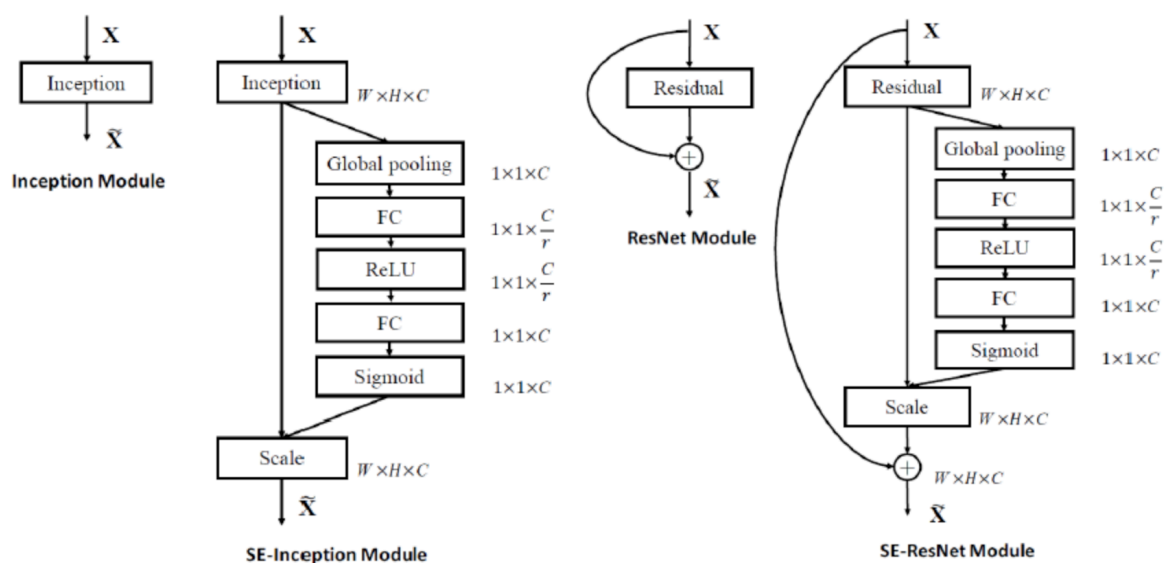
- (1) F_{tr} : X 到 U 的卷积过程，但是通道之间的关系并没有发生变化；
- (2) F_{sq} : 将每个通道做了一个 $squeeze$ 操作（主要是池化操作），将每个通道表示成了一个标量，得到per channel的描述；
- (3) F_{ex} : 将per channel标量进行“激活”，可以理解为算出了per channel的 W （权值），实际上这一步就是全连接；
- (4) 最后将per channel的 W （权重）乘回到原来的feature map上得到加权后的channel，将channel做了恰当的融合；

SE-Module 可以用于网络的任意阶段，且 $squeeze$ 操作保证了在网络的早期感受野就可以大到全图的范围。



具体实践中的SE module有SE-inception Module and SE-ResNet Module:

左图是SE-inception Module，第(2)步中的squeeze采用average pooling，得到 $1 \times 1 \times C$ 的向量；后面再接FC，但是为了减少参数，做了降维操作，增加了一个降维系数 r ，输出 $1 \times 1 \times \frac{C}{r}$ ；后接 $ReLU$ ，再做一个升维操作，得到 $1 \times 1 \times C$ ，最终采用 $Sigmoid$ 函数激活。激活之后，将每一个通道的权值向量 $1 \times 1 \times C$ 乘到相应的通道上，结构如下：(右图是resnet module，改造和inception分支很类似。)



下图是MobileNetV3的结构:

Input	Operator	exp size	#out	SE	NL	s
$224^2 \times 3$	conv2d	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	-	RE	1
$112^2 \times 16$	bneck, 3x3	64	24	-	RE	2
$56^2 \times 24$	bneck, 3x3	72	24	-	RE	1
$56^2 \times 24$	bneck, 5x5	72	40	✓	RE	2
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 3x3	240	80	-	HS	2
$14^2 \times 80$	bneck, 3x3	200	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	480	112	✓	HS	1
$14^2 \times 112$	bneck, 3x3	672	112	✓	HS	1
$14^2 \times 112$	bneck, 5x5	672	160	✓	HS	1
$14^2 \times 112$	bneck, 5x5	672	160	✓	HS	2
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	conv2d, 1x1	-	960	-	HS	1
$7^2 \times 960$	Pool, 7x7	-	-	-	HS	-
$1^2 \times 960$	conv2d 1x1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	conv2d 1x1, NBN	-	k	-	-	-

可以看出文章中提出的GhostNet模型结构用Ghost Bottlenecks代替了MobileNetV3中的Bottlenecks, 剩下的模型中不同层的输入输出规模与MobileNetV3的基本一样。

5.ShuffleNet

V1

1.Channel Shuffle for Group Convolutions

(1) Xception和ResNeXt都没有考虑 1×1 (pointwise) 的卷积的计算量, 比如在ResNeXt中, 组卷积只在 3×3 的层中使用, 导致pointwise卷积占了Madds的93.4%; 直观的解决方法就是在 1×1 的层中也使用组卷积;

(2) Channel Shuffle: 对于前一组层的channel, 先将所有的channel分为几个组, 再将每个组中的通道划分为几个子组, 然后用不同的子组来填充下一层中的每个组。这可以通过channel shuffle操作高效地实现 (如下图)

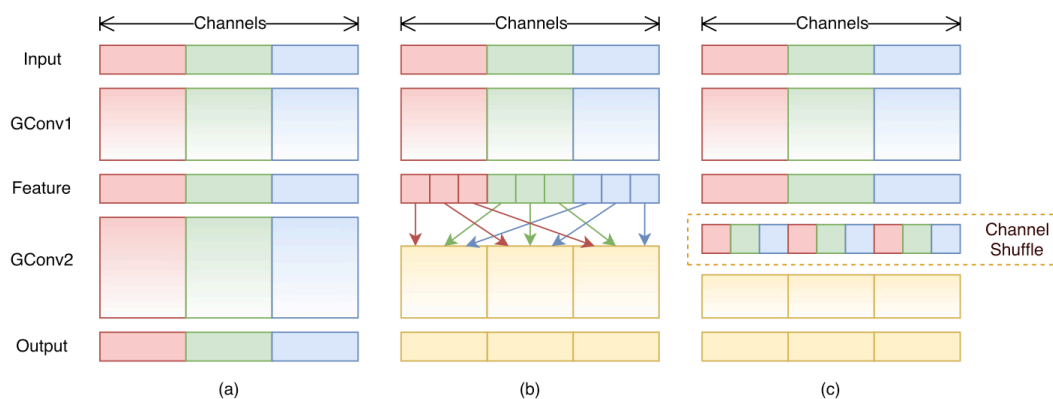


Figure 1. Channel shuffle with two stacked group convolutions. GConv stands for group convolution. a) two stacked convolution layers with the same number of groups. Each output channel only relates to the input channels within the group. No cross talk; b) input and output channels are fully related when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

2.ShuffleNet Unit

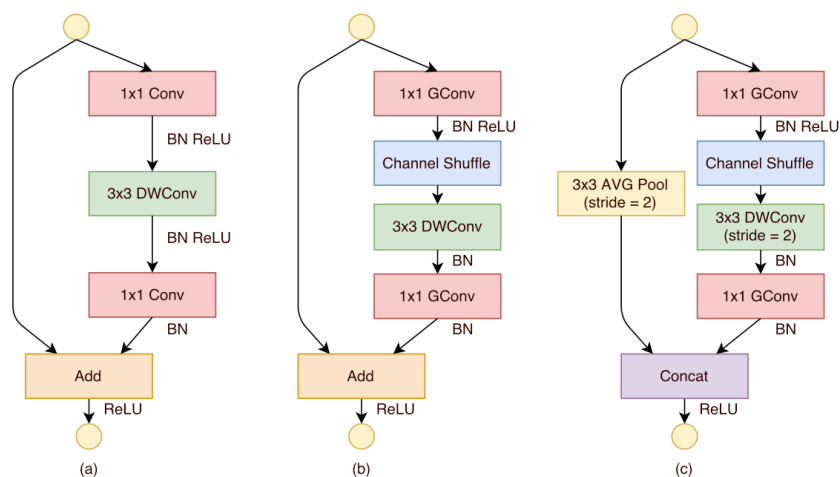


Figure 2. ShuffleNet Units. a) bottleneck unit [9] with depthwise convolution (DWConv) [3, 12]; b) ShuffleNet unit with pointwise group convolution (GConv) and channel shuffle; c) ShuffleNet unit with stride = 2.

a)是MobileNetV2的bottleneck；b)是stride为1的ShuffleNet Unit，在 1×1 的pointwise卷积过程采用组卷积，以及Channel Shuffle，与shortcut的结果逐体素相加；c)是stride为2的ShuffleNet Unit，中间的DWConv步长为2，shortcut是一个步长为2的均值池化，两个结果进行一个channel concat。

3.Network Architecture

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.

V2

1. Practical Guidelines for Efficient Network Design

- (1) Equal channel width minimizes memory access cost (MAC).卷积层的输入输出通道数相等时最节约存储空间；
- (2) Excessive group convolution increases MAC.组卷积可以减少FLOPs，但是过度的组卷积增大存储空间；
- (3) Network fragmentation reduces degree of parallelism.网络碎片化降低了并行度，应该减少分支；
- (4) Element-wise operations are non-negligible.元素级别操作是不可忽略的，比如ReLU逐像素激活和depthwise convolutions。

2. ShuffleNet V2: an Efficient Architecture

ShuffleNetV2 Unit:

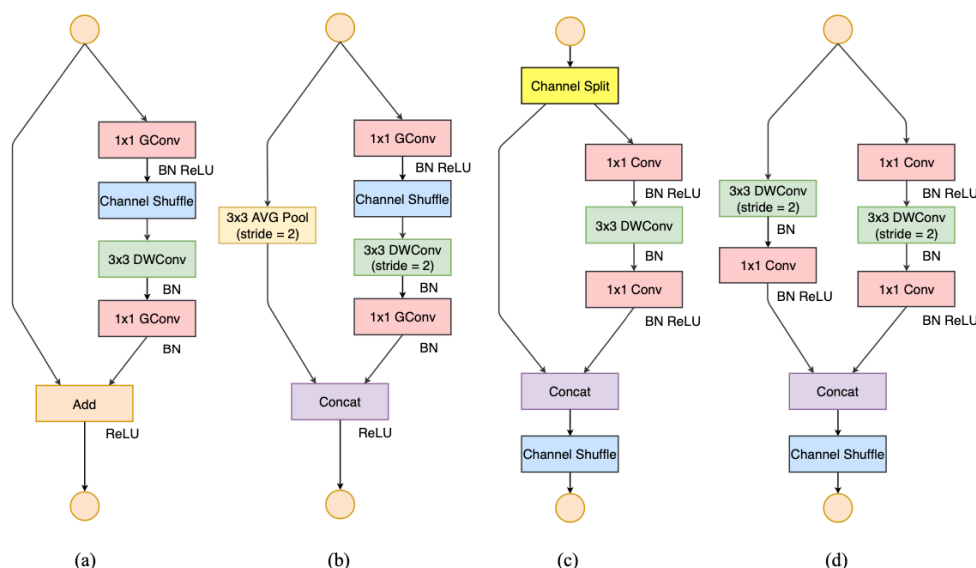


Fig. 3: Building blocks of ShuffleNet v1 [15] and this work. (a): the basic ShuffleNet unit; (b) the ShuffleNet unit for spatial down sampling ($2\times$); (c) our basic unit; (d) our unit for spatial down sampling ($2\times$). **DWConv**: depthwise convolution. **GConv**: group convolution.

(1) 对于结构(c):

- 1) 在每个Unit开始，引入Channel Split操作，将输入的feature channels分为两个分支，分别有 $c - c'$ 和 c' 个channels；
- 2) 其中一个分支直接恒等映射到下一层（G3），另一个分支经过三个卷积层，但是通道数不变（G1），三个卷积层中的 1×1 卷积不再像ShuffleNetV1中使用组卷积（G2），因为在channel split阶段已经分组了；
- 3) 两个分支再经过concat融合，使得通道数和最开始的输入通道数一致（G1），最后再经过channel shuffle，加强两个分支的通道交互。

(2) 对于结构(d)（经过unit(c)的channel shuffle之后，接上unit(d)），和ShuffleNetV1一样，提出一个下采样的unit。

在结构(c)(d)中，都没有ShuffleNetV1中的两分支add的操作，直接将两分支的结果进行concat，因为根据（G4），逐像素操作很费时间。

根据结构(c)(d)，构成ShuffleNetV2：（和ShuffleNetV1基本一致）

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.

6.MnasNet(Mobile neural architecture search Net)

1.Problem Formulation

(1) common method, $ACC(m)$ 代表模型准确率, $LAT(m)$ 代表模型的latency, T 是目标latency限制, 则优化问题简化如下:

$$\begin{aligned} & maximize_m ACC(m) \\ & subject\ to\ LAT(m) \leq T \end{aligned}$$

(2) 这种方法只是在优化单一指标, 得不到多重帕累托最优解, 帕累托最优是指模型有最高的准确率的同时, 也有最低的latency。为了获得帕累托最优, 改变优化问题如下:

$$\begin{aligned} & maximize_m ACC(m) \times \left[\frac{LAT(m)}{T} \right]^w \\ & w = \alpha\ if\ LAT(m) \leq T, \beta\ otherwise \end{aligned}$$

α, β 这样选择, 在不同的acc-latency对应的模型之下, 应该有相同的reward, 如一个模型的准确率为 a , latency为 l , 另一个模型的latency为 $2l$, 但是准确率提高了5, 此时应该有:

$a(1 + 5\%) \cdot (\frac{2l}{T})^\beta = a \cdot (\frac{l}{T})^\alpha$, 求解得 $\beta = -0.07$, 再取 $\alpha = \beta = -0.07$, 后续模型都是采用这个参数, 除非特殊说明。

2.Mobile Neural Architecture Search (移动端神经网络框架搜索)

2.1.Factorized Hierarchical Search Space (分解层次搜索空间)

以前的大多数方法只搜索几个复杂的单元，然后重复地堆叠相同的单元。与之前的方法不同，我们引入了一种新的分解层次搜索空间，它将CNN模型分解为独特的块，然后分别搜索每个块的操作数和连接数，从而允许在不同块中使用不同的层结构，如下图：

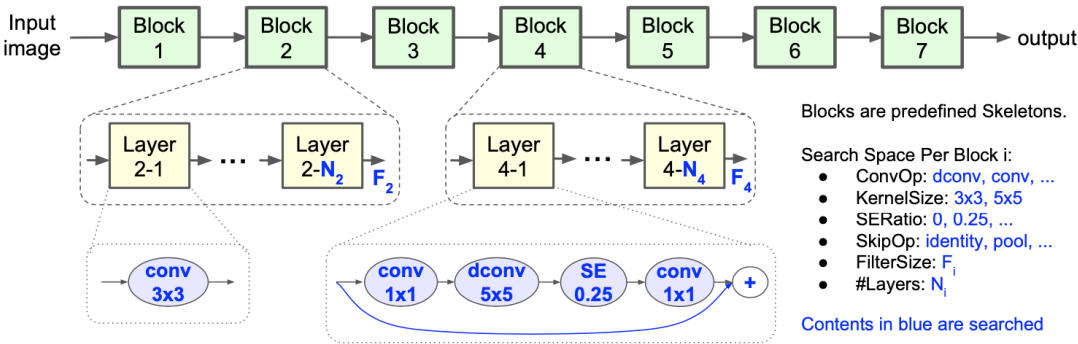


Figure 4: **Factorized Hierarchical Search Space.** Network layers are grouped into a number of predefined skeletons, called blocks, based on their input resolutions and filter sizes. Each block contains a variable number of repeated identical layers where only the first layer has stride 2 if input/output resolutions are different but all other layers have stride 1. For each block, we search for the operations and connections for a single layer and the number of layers N , then the same layer is repeated N times (e.g., Layer 4-1 to 4- N_4 are the same). Layers from different blocks (e.g., Layer 2-1 and 4-1) can be different.

将整个网络分为若干个block，每个block包含多个Layer。在搜索时，不同的block之间的结构是不同的，从相应的子空间中候选网络结构，选定之后，每个block里面的Layer是相同重复的。相比于之前单一搜索结构然后重复堆砌的方法，这种分层搜索的方法大大增加了层次多样性。

2.2.搜索算法：强化学习方法来搜索

具体的网络结构（MnasNet-A1）：

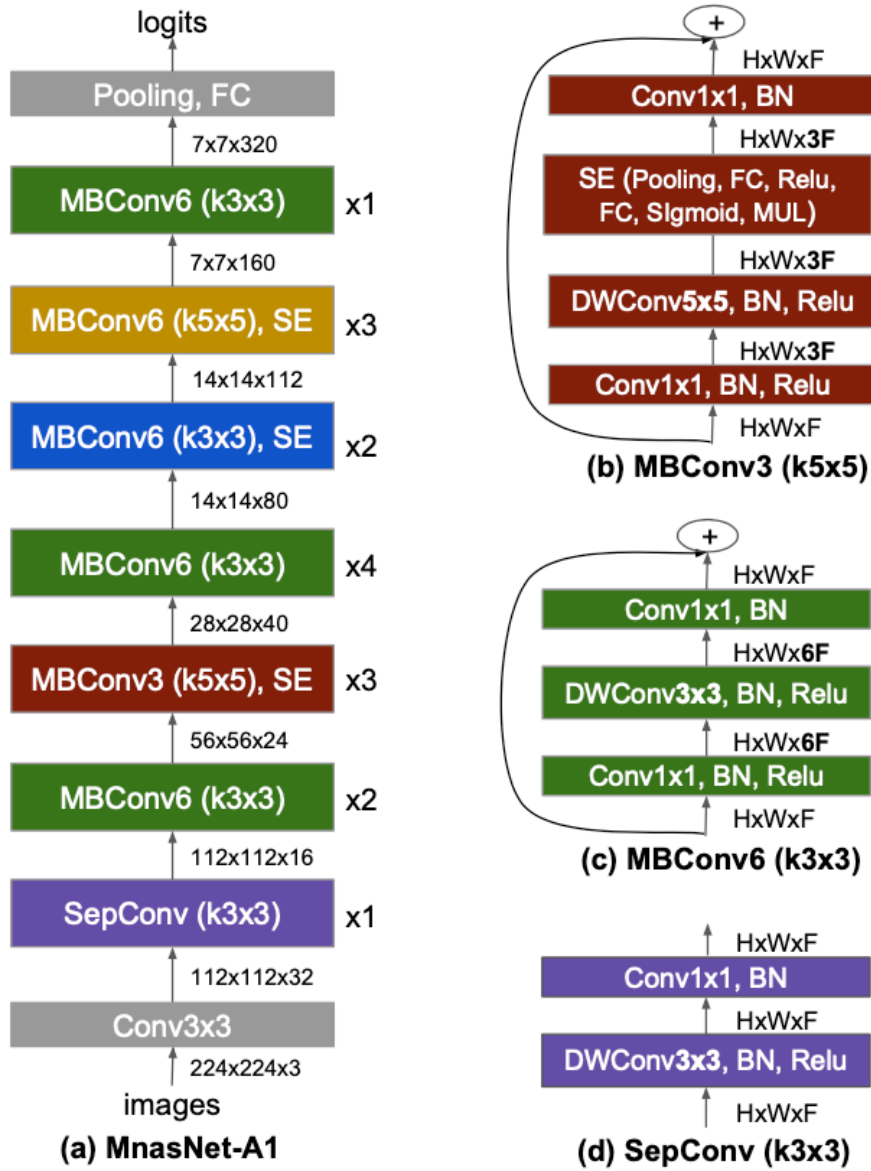


Figure 7: **MnasNet-A1 Architecture** – (a) is a representative model selected from Table 1; (b) - (d) are a few corresponding layer structures. *MBConv* denotes mobile inverted bottleneck conv, *DWConv* denotes depthwise conv, k3x3/k5x5 denotes kernel size, *BN* is batch norm, HxWxF denotes tensor shape (height, width, depth), and $\times 1/2/3/4$ denotes the number of repeated layers within the block.

7.EfficientNet

1.Compound Model Scaling (复合模型扩展)

1.1.模型数学表达

(1) 实际中，卷积网络一般分为好几个阶段，但是每个阶段都是相同的结构；如ResNet有5个阶段，每个阶段都是一样的卷积操作，除了第一层有个下采样；

所以将一个卷积神经网络表示如下：

$$N = \triangle_{i=1,\dots,s} F_i^{L_i}(X_{\langle H_i, W_i, C_i \rangle})$$

(2) 和一般的卷积网络设计聚焦在找到最好的网络结构 F_i 不一样，model scaling要找到 L_i, C_i, H_i, W_i 之间合适的比例关系，而不改变baseline网络中的 F_i 结构；

(3) 限定 F_i, L_i, H_i, W_i, C_i 不变，调整 w, d, r 系数（分别是通道数 C_i ，每阶段的卷积层数 L_i 以及空间尺寸 H_i, W_i 的比例系数）的值，使得模型获得最高的精度；

1.2.调整单一的参数

(1) 增大网络的深度比例参数 d 可以获得更复杂的特征，但同时也会出现梯度消失的情形，尽管ResNet或者DenceNet中有skip connection的结构，但是ResNet-1000和ResNet-101的表现差不多；

(2) 增大网络的宽度比例参数 w 可以使得网络更容易训练，获得更好的细粒度模式（fine-grained pattern），但是极度宽但是层数少的网络得不到高层次的表示特征，并且表现随着 w 的增大容易饱和；

(3) 基础的分辨率是 $224 * 224$ ，在一定程度上，提高分辨率系数 r 会提高表现，非常高的分辨率时增长幅度消失；

总结：在一定范围内调整任何一个维度的参数都会适当提高模型表现，但是对于更大的模型，表现趋于饱和。

1.3.复合调整参数

(1) 增大图像分辨率时，相应的要增大模型深度（获得更好的表示特征）和模型宽度（获得更好的细粒度模式）；

总结：调整模型参数 w, d, r ，要注意相互平衡。

2.EfficientNet Architecture

(1) 受到MnasNet的启发，开发了基线网络——通过利用多目标神经结构搜索优化准确性和FLOPs；

(2) 网络的结构如下：（主要卷积模块为MBConv，和MnasNet一样）

Table 1. EfficientNet-B0 baseline network – Each row describes a stage i with \hat{L}_i layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels \hat{C}_i . Notations are adopted from equation 2.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels \hat{C}_i	#Layers \hat{L}_i
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

以此网络结构为baseline网络，应用上述的复合调整参数的方法来获得EfficientNet-B系列网络，步骤为：

- 1) 先固定 $\phi = 1$ ，在小范围内搜索 α, β, γ 的值使得满足FLOPs以及模型大小约束的条件下，模型准确率最大的最优解，最终得到 $\alpha = 1.2, \beta = 1.1, \gamma = 1.15$ ；
- 2) 然后固定 α, β, γ ，再选择 ϕ 的值，获得EfficientNet-B0-B7的网络。

8.模型结构对比总结

1.ResNet和MobileNetV1是最基本的两个trick，一个提出残差结构，引入shortcut；另一个将传统的卷积分离。后续的MobileNetV2,V3和GhostNet都采用了残差结构和卷积分离；

2.MobileNetV2在MobileNetV1的block之前添加了一层pointwise convolution来增加通道数，且添加了shortcut，和ResNet相比形状相反，相当于Inverted residuals；

3.MobileNetV3是综合了以下三种模型的思想：MobileNetV1的深度可分离卷积（depthwise separable convolutions）、MobileNetV2的具有线性瓶颈的逆残差结构(the inverted residual with linear bottleneck)和SENet的基于squeeze and excitation结构的轻量级注意力模型；

4.GhostNet与MobileNetV2网络的bottleneck非常类似，都采用了shortcut结构和深度分离卷积的结构，且整体上都呈现出逆残差的结构（即在bottleneck中，都是先扩大通道数，再减少通道数）。不同的是，MobileNetV2的bottleneck中，基本单元是pointwise和depthwise卷积，而GhostNet的bottleneck中的基本单元是ghost module结构，ghost module中包含了pointwise和depthwise卷积，并且有的bottleneck中还包含SE module；

5.GhostNet在MobileNetV3的基础上，将Bottlenecks替换成Ghost Bottlenecks，Ghost Bottlenecks本身也是一个残差结构，其中的ghost module也是传统卷积的另一种分解和加速；

6.ShuffleNetV1中提出的两个ShuffleNet Unit和GhostNet中的两个Ghost Bottleneck有点相似，stride分别是1和2，在stride为2的bottleneck中，shortcut都采用了下采样；ShuffleNetV2中的channel split，和ghost module中的固有channel很类似，都是把输入的channel的一部分channel直接恒等映射到下一层，另一部分再经过特定设计的卷积层作用；

7.对于所有的轻量级网络，MobileNet系列、ShuffleNet系列以及GhostNet都属于手动设计网络结构的小网络，MnasNet和EfficientNet属于神经网络框架自搜索的小网络，网络结构是在一个特定搜索空间中运用强化学习搜索算法搜索得到的结构。

二、比较模型的大小、parameters多少，以及FLOPs大小

1.GhostNet的评测

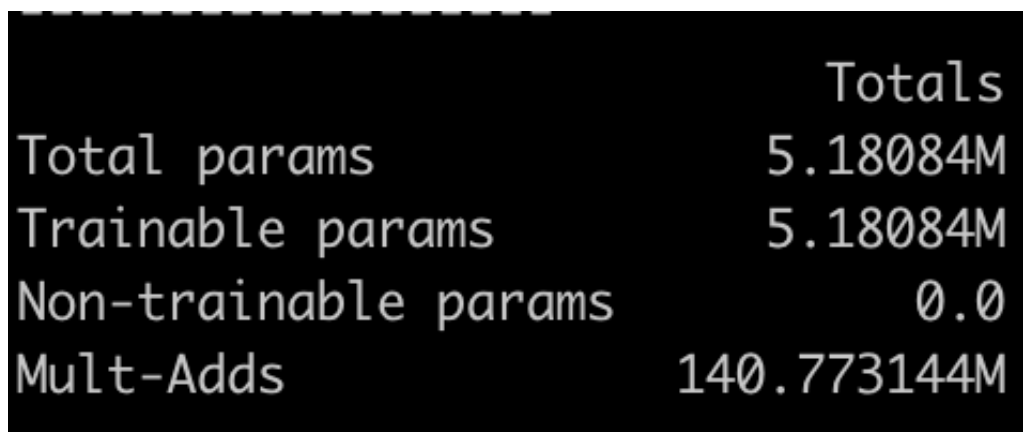
(1) 计算模型大小和参数量

假设模型中有1M的parameters，每个参数以float32的形式存储，需要1M*32个比特（bit），每8个构成一个字节（byte），所以实际文件大小是1M*32/8=4M。在计算一个模型中的parameter的量时，直接模型文件实际大小除以4即得到parameters大小。parameters的大小也可以通过下面的工具来计算。

(2) 计算FLOPs（同时也计算了parameters）

(参考<https://github.com/nmhkahn/torchsummaryX>)

计算代码：param_flops.py，计算结果截图如下：

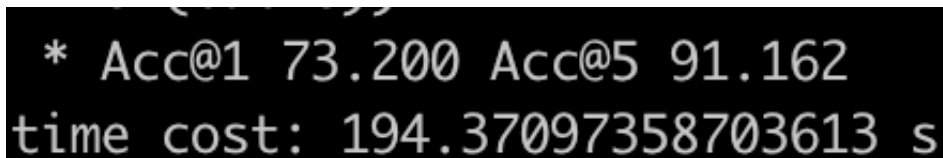


	Totals
Total params	5.18084M
Trainable params	5.18084M
Non-trainable params	0.0
Mult-Adds	140.773144M

(3) 测试GhostNet在ImageNet测试集上的分类精度

```
python test_time.py --evaluate
```

测试结果为：



```
* Acc@1 73.200 Acc@5 91.162
time cost: 194.37097358703613 s
```

(4) pytorch模型GhostNet在5000张图像上的测试，代码test_time_pt.py

测试GhostNet的时间为：36.18s，单张图像的latency时间为7.23ms/--。

2.MobileNet系列网络的评测

该部分评测只测试了各个网络模型的单张图像的latency，其余指标采用官方数据。tensorflow官方发布的MobileNet系列网络运行时间的计算：

参考链接 (https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet/mobilenet_example.ipynb)

1.tensorflow模型MobileNet系列在5000张图像上的测试，代码test_time_tf2.py

分别测试MobileNetV1， MobileNetV2， MobileNetV3的FP32模型和INT8模型，结果分别为：time cost: 43.20 s, time cost: 50.119 s, 57.36s/56.75s，单张图像的latency分别为8.64ms/-, 10.02ms/-, 11.47ms/11.35ms。

3.ShuffleNet系列网络的评测

测试代码eval_pytorch_model_image.py，测试ShuffleNetV1,V2两个网络模型的运行时间，结果分别为：92.76s和94.24s。

4.MnasNet系列网络的评测

参考链接：https://github.com/tensorflow/tpu/blob/master/models/official/mnasnet/mnasnet_example.ipynb，可以评测时间，具体见代码test_time_mnasnet.py。

5.EfficientNet系列网络的评测

先尝试了链接 (<https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet/lite>) 中的代码测试eval_ckpt_main.py测试时间，发现运行结果里每个模型（EfficientNet-lite0-lite2）单张运行图像时间在4s左右，和之前测试的MobileNet系列，GhostNet以及官方发布的时间不在一个量级，这段代码不能用于时间测试，测不出真实的latency。

再参考链接 (https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/tools/evaluation/tasks/imagenet_image_classification)，从源码来运行.tflite模型，还是出现很多问题。

再使用亚勇哥给的一个脚本tflite_predict_file.py来测.tflite模型，步骤如下：

(1) 在本地安装工具：<https://github.com/lutzroeder/netron>，用于可视化.tflite模型各个节点的名称；

(2) 获悉节点名称之后更改代码tflite_predict_file.py：

1) 46行需要根据模型更改输入节点的名称；

```
self.assertEqual('input', input_details[0]['name']) #源代码
self.assertEqual('images', input_details[0]['name']) # 替换之后
```

2) 48行需要根据模型更改输入尺寸；

```

self.assertTrue([1, 224, 224, 3] == input_details[0]['shape']).all()) # 源代码
self.assertTrue([1, 224, 224, 3] == input_details[0]['shape']).all()) # lite0
不需要修改
self.assertTrue([1, 240, 240, 3] == input_details[0]['shape']).all()) # lite1
修改为240
self.assertTrue([1, 260, 260, 3] == input_details[0]['shape']).all()) # lite2
修改为260

```

3) 49行均值及方差，用于quantization；

```

self.assertEqual((0.007843137718737125, 128), input_details[0]
['quantization']) # 源代码
self.assertEqual((0.007843137718737125, 128), input_details[0]
['quantization']) # lite0_uint8

```

4) 56行需要根据模型更改输出节点的名称；

```

self.assertEqual('MobilenetV2/Predictions/Softmax', output_details[0]['name'])
# 源代码
self.assertEqual('Softmax', output_details[0]['name']) # lite0_uint8

```

5) 129行输出是int类型的0-255，需要除以255，更改为浮点型0-1之间的概率。注意83-89行以及126-132行

更改之后的脚本为test_tflite.py，该脚本在有的tensorflow版本中会报错，再次改写为test2_tflite.py，成功测试完EfficientNet系列模型（实际上该脚本不仅仅可以测试EfficientNet系列的网络，可以测试任意的tflite模型）。

移动端网络评测总结表格见（<https://docs.google.com/spreadsheets/d/1Vqz9puFFImWB5TQHMKqnlkbSRmEXSMTpOgAnpvPgWSY/edit#gid=0>）

注：关于latency的测试，MobileNet,MnasNet,EfficientNet网络的latency都是单张图像在CPU上的时间，ShuffleNet,GhostNet网络的latency都是单张图像在GPU上的时间。