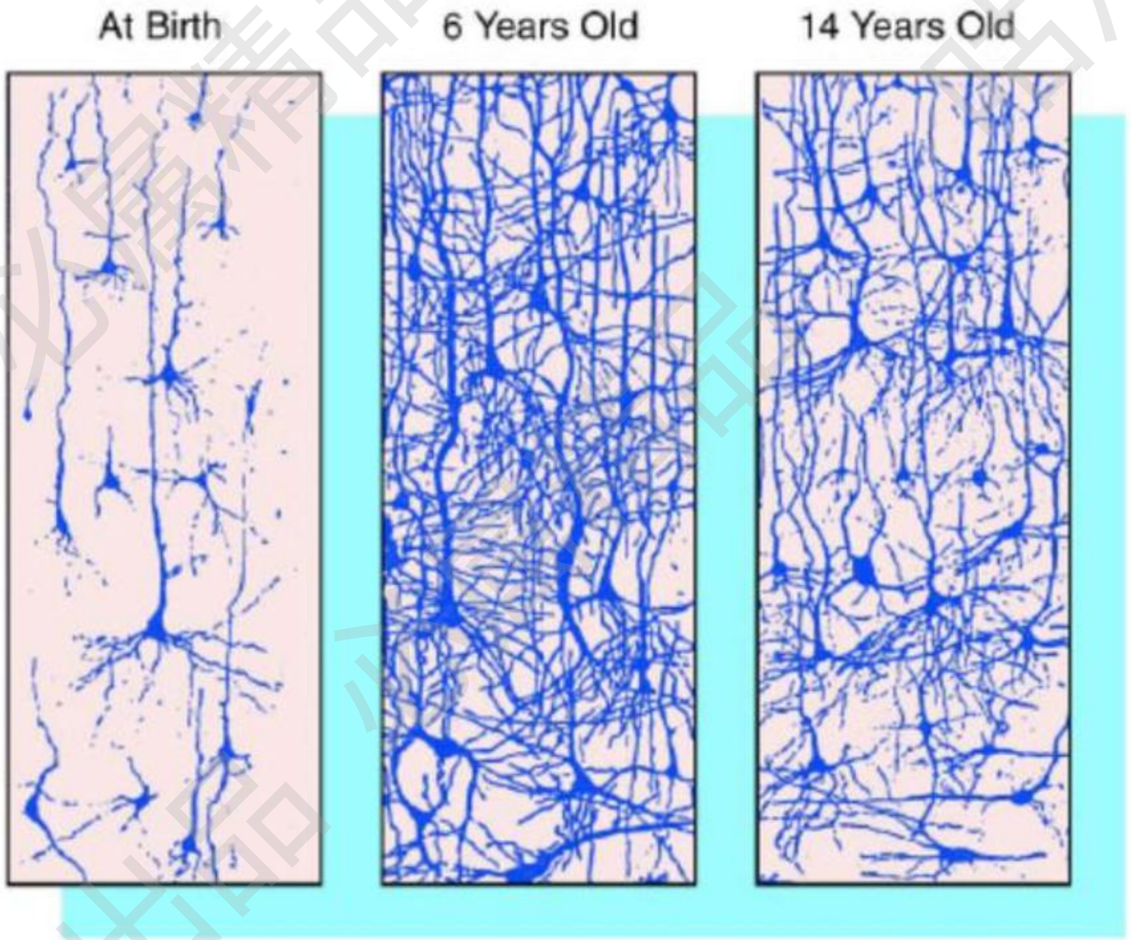
模型剪枝策略

- 模型如人脑
  - 模型剪枝主要目的就是减少参数量
  - 如右图所示，学习的内容会不断增多
  - 一定程度上会降低效果，提升性能
  - 通常会根据应用领域来选择剪枝



参考论文：Selective Brain Damage: Measuring the Disparate Impact of Model Pruning

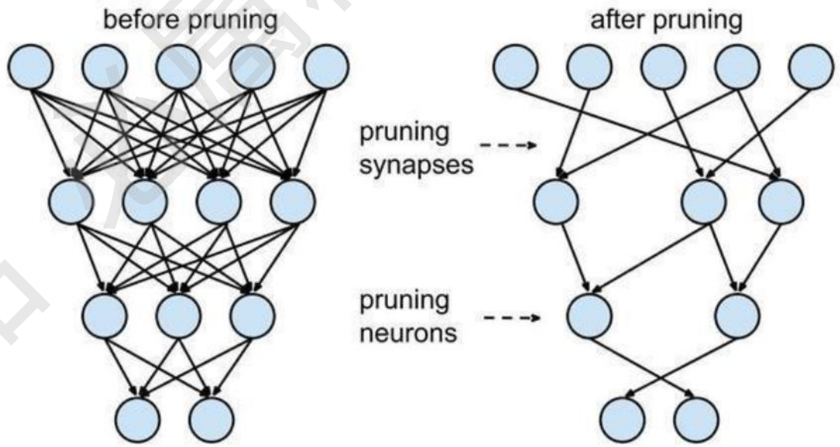
模型剪枝策略

- network pruning

基本路线：训练（大型模型），剪枝（策略）和微调（重新训练）

对冗余权重进行修剪并保留重要权重，以最大限度地保持精确性

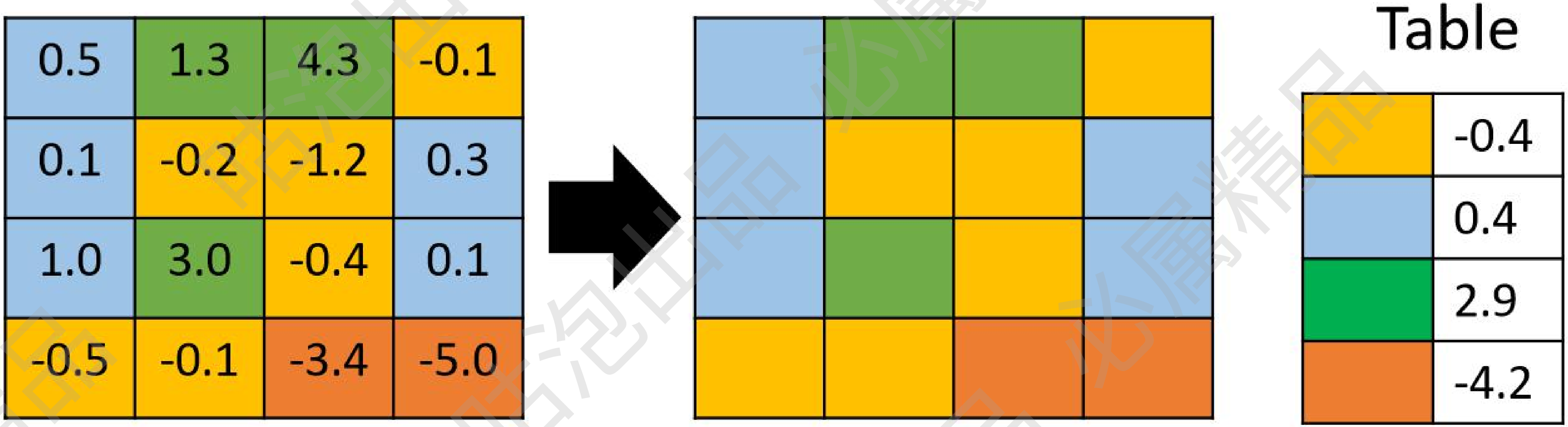
但是需要注意，剪枝效果并不一定完全有效，



参考论文： RETHINKING THE VALUE OF NETWORK PRUNING

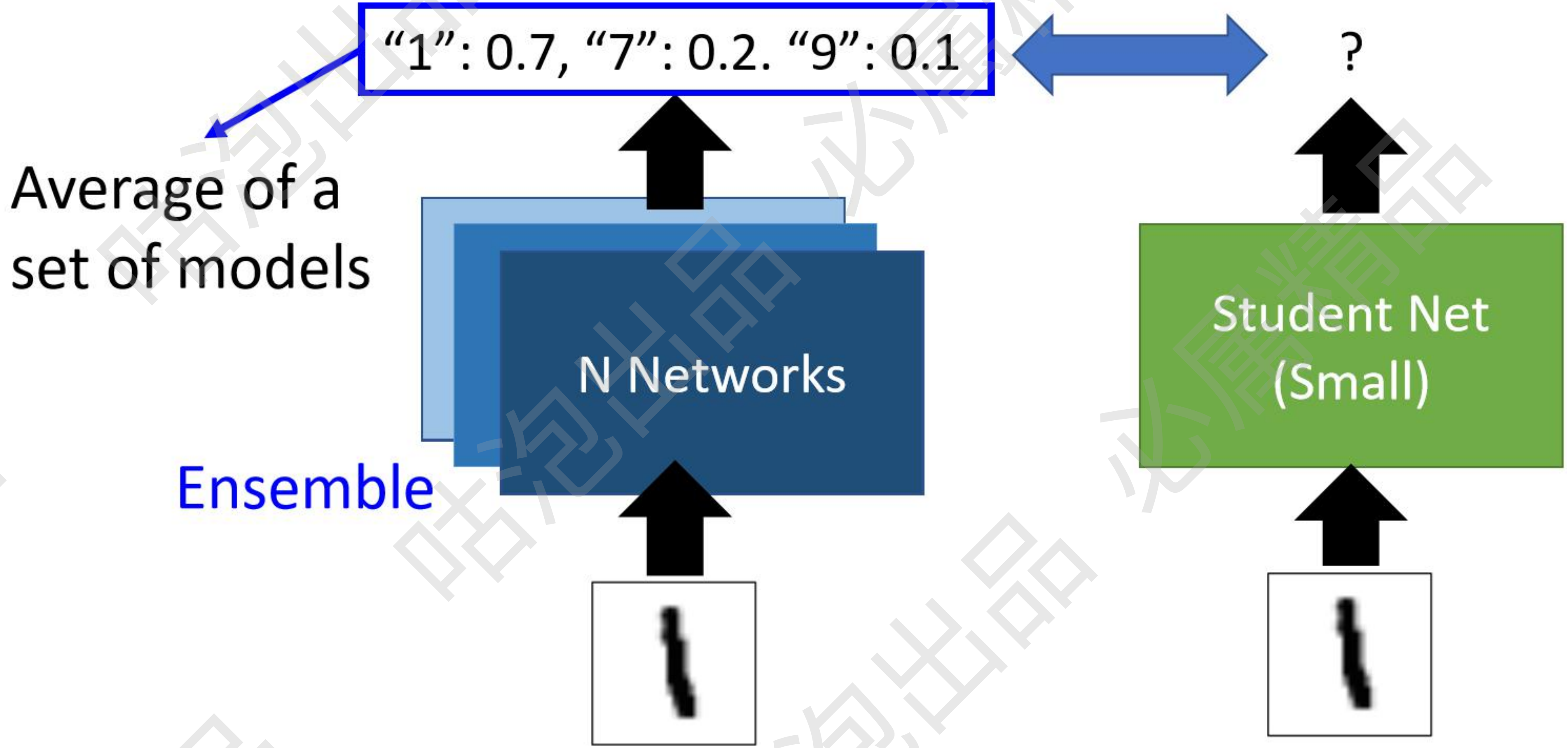
MobileNet

- 一些策略（仅供参考）



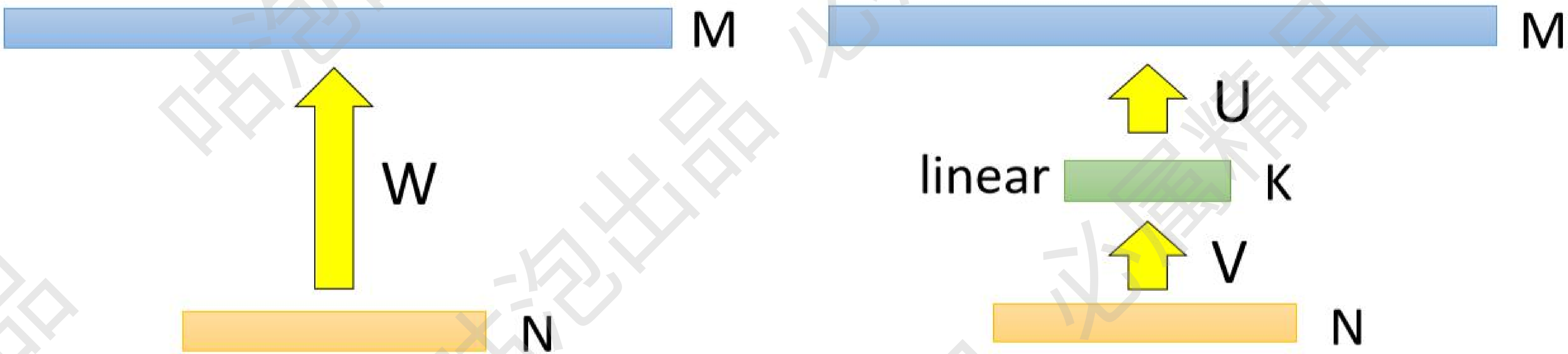
MobileNet

- 一些策略（仅供参考）



MobileNet

- 一些策略（仅供参考）



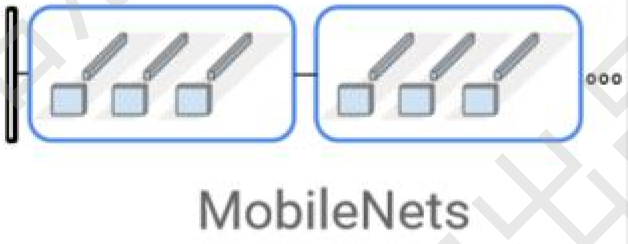
MobileNet

- 为什么需要它

  - 很多嵌入式，手机端无法使用庞大的网络结构

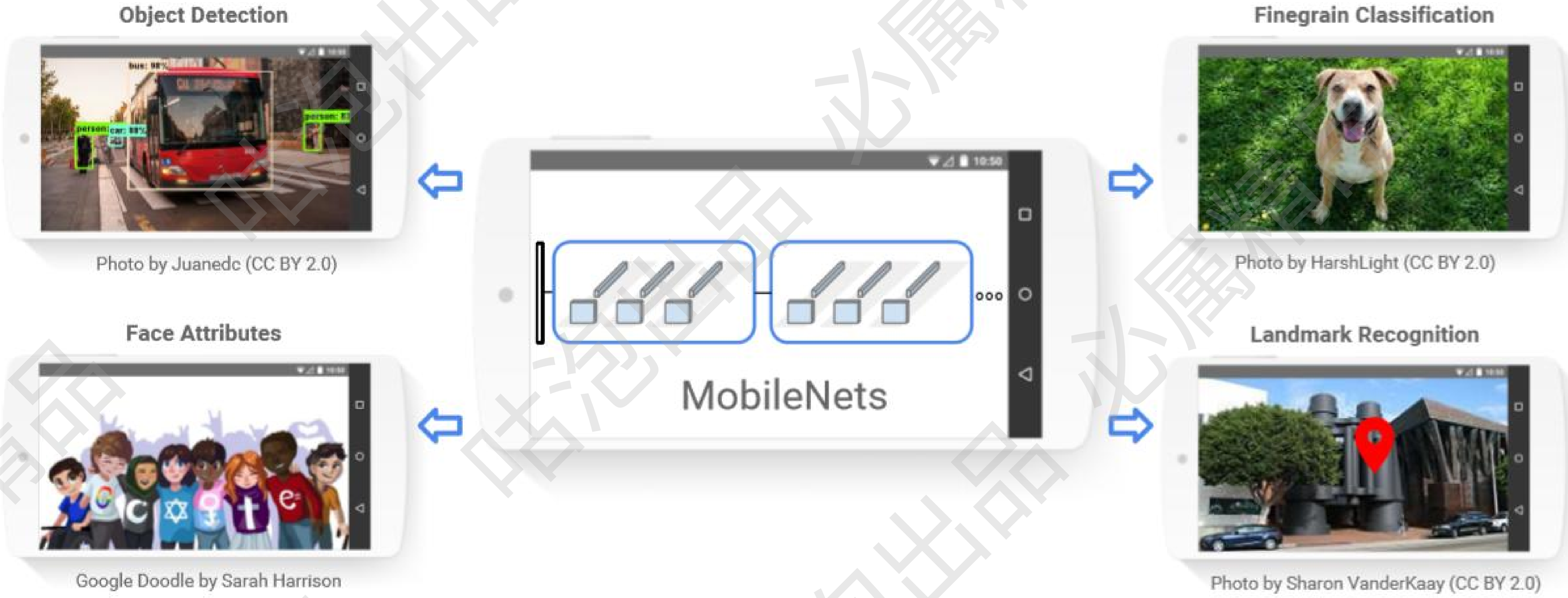
  - 一个几百M的模型能随便移植嘛。。。

  - Mobilenet不光模型参数小，计算量也很小，并且还能DIY



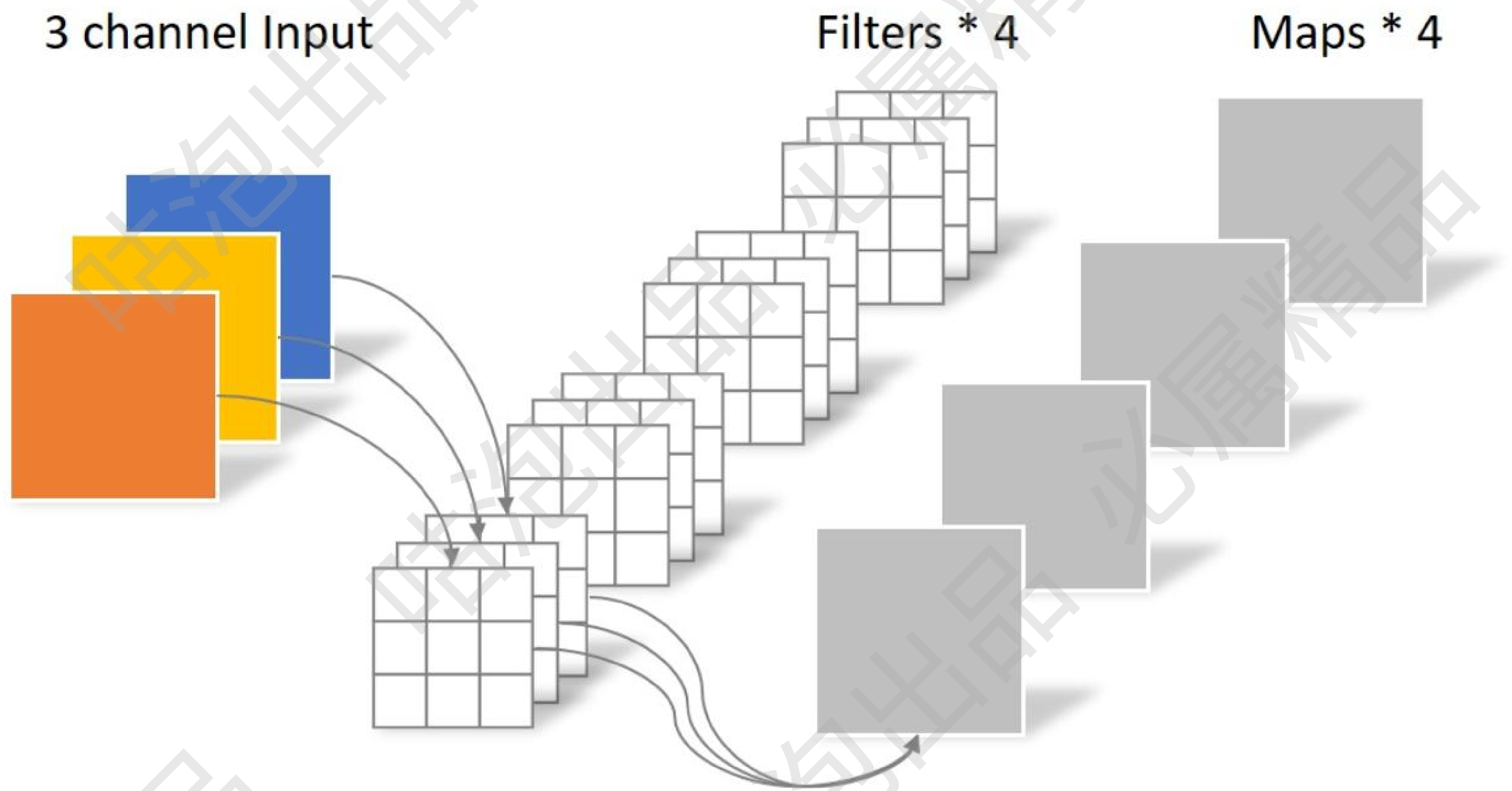
MobileNet

- V1版本



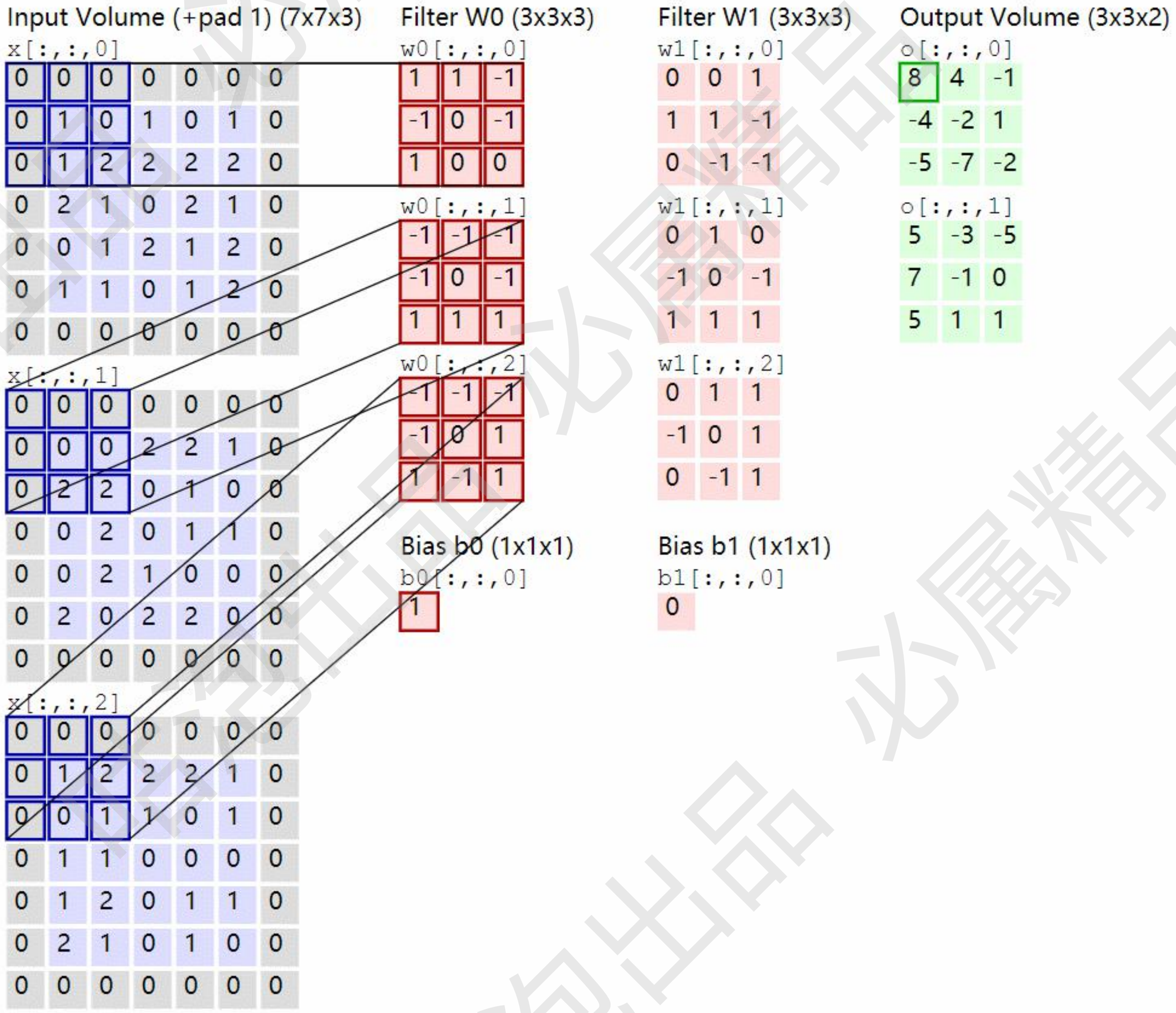
MobileNet

- 经典卷积计算



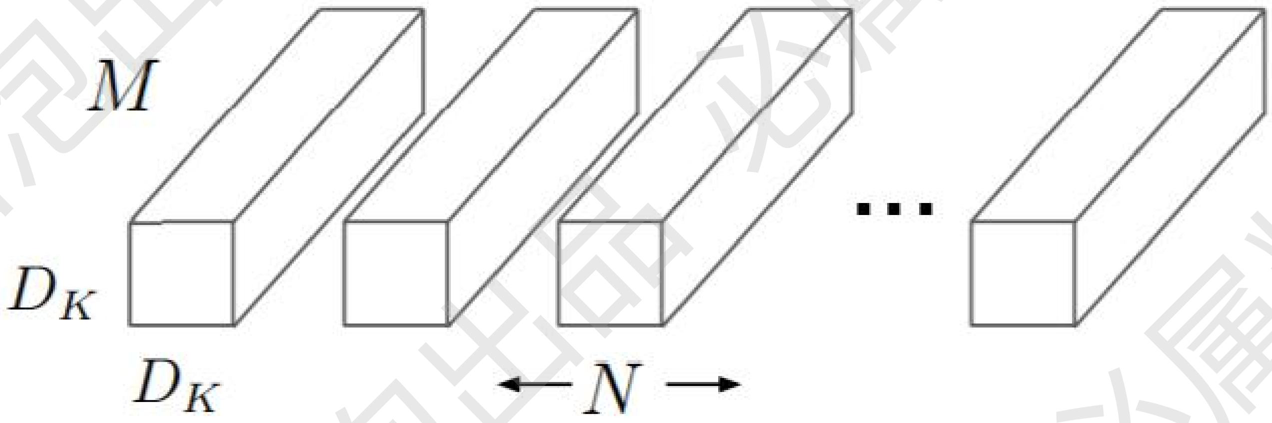
MobileNet

- 经典卷积计算



MobileNet

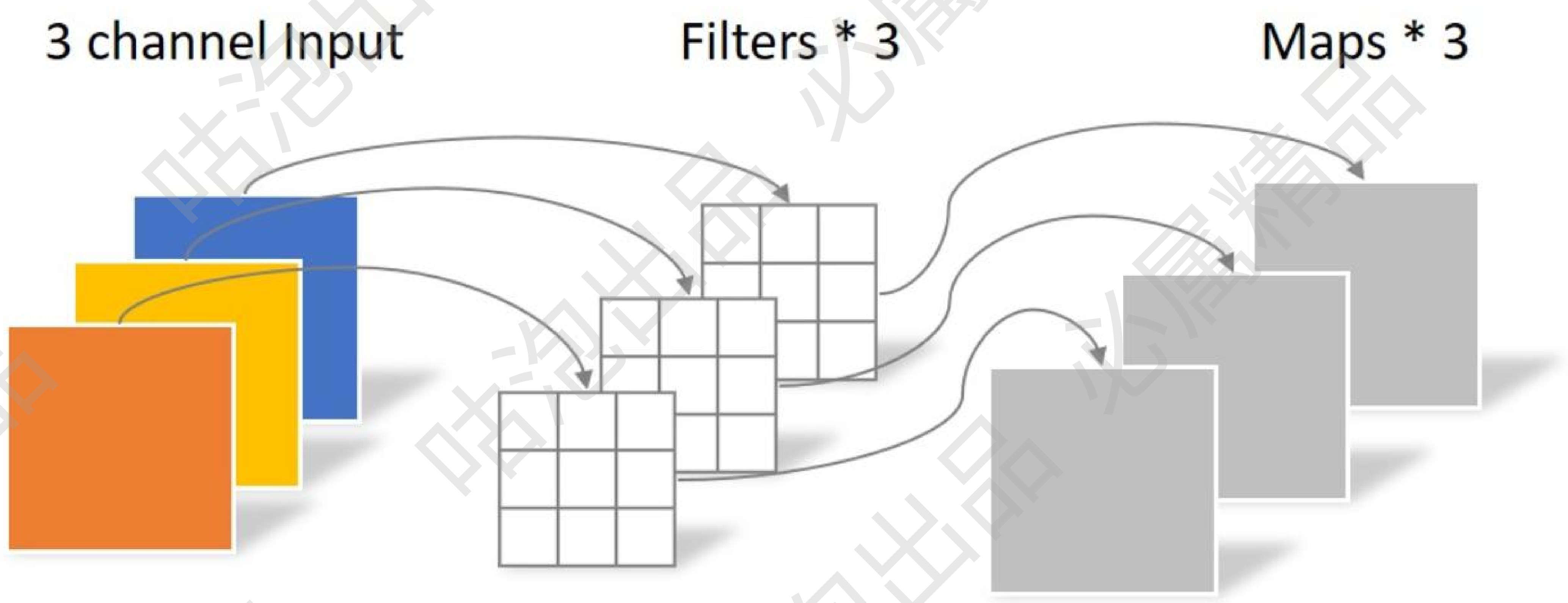
- 经典卷积计算



- 卷积核大小为DK*DK*M，使用N个卷积核来完成卷积操作
- 所需计算量为：$D_K \times D_K \times M \times N \times D_W \times D_H$（W,H为输入大小）

MobileNet

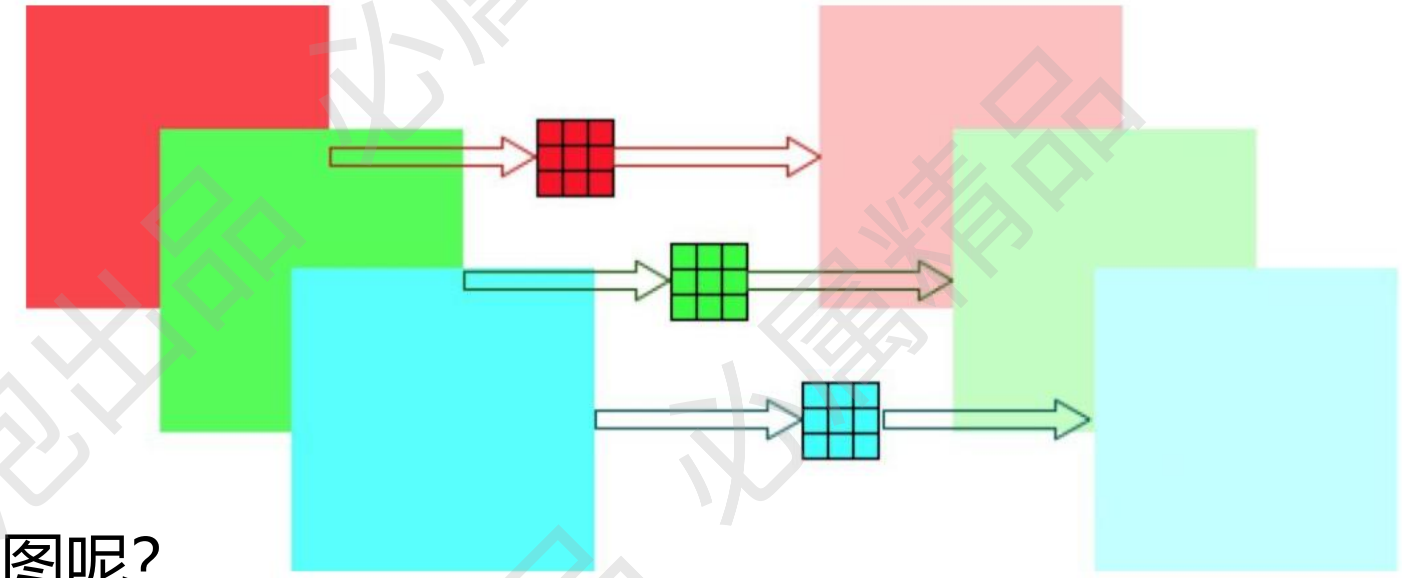
- Depthwise卷积



MobileNet

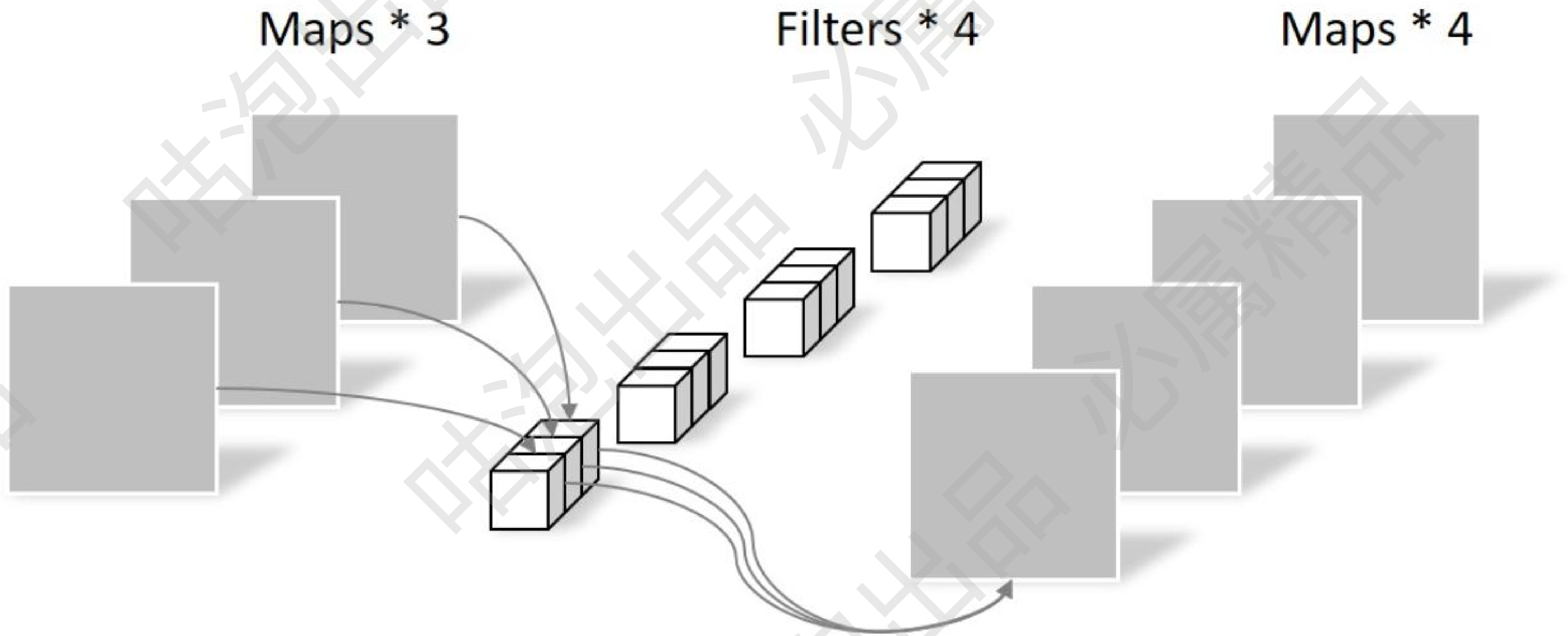
- Depthwise卷积

  - 注意卷积核的对应
  - 3个卷积核，不是1个
  - 输出与输入是相对应的
  - 如何才能得到更多的特征图呢?



MobileNet

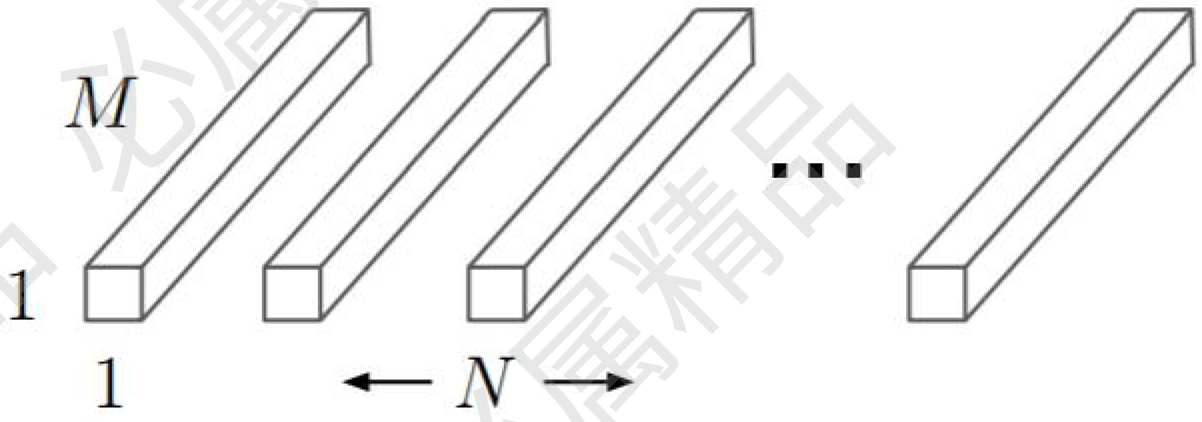
✓ Pointwise卷积



MobileNet

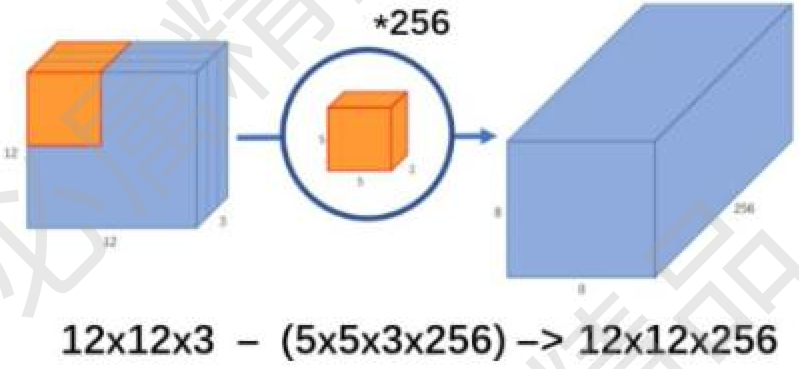
- Pointwise卷积

  - 1*1卷积来执行，参数比较省
  - 跟传统卷积计算方法一样
  - N个卷积核就可以得到N个特征图
  - V1版本其实就是把Depthwise卷积和Pointwise卷积组合到一起了！

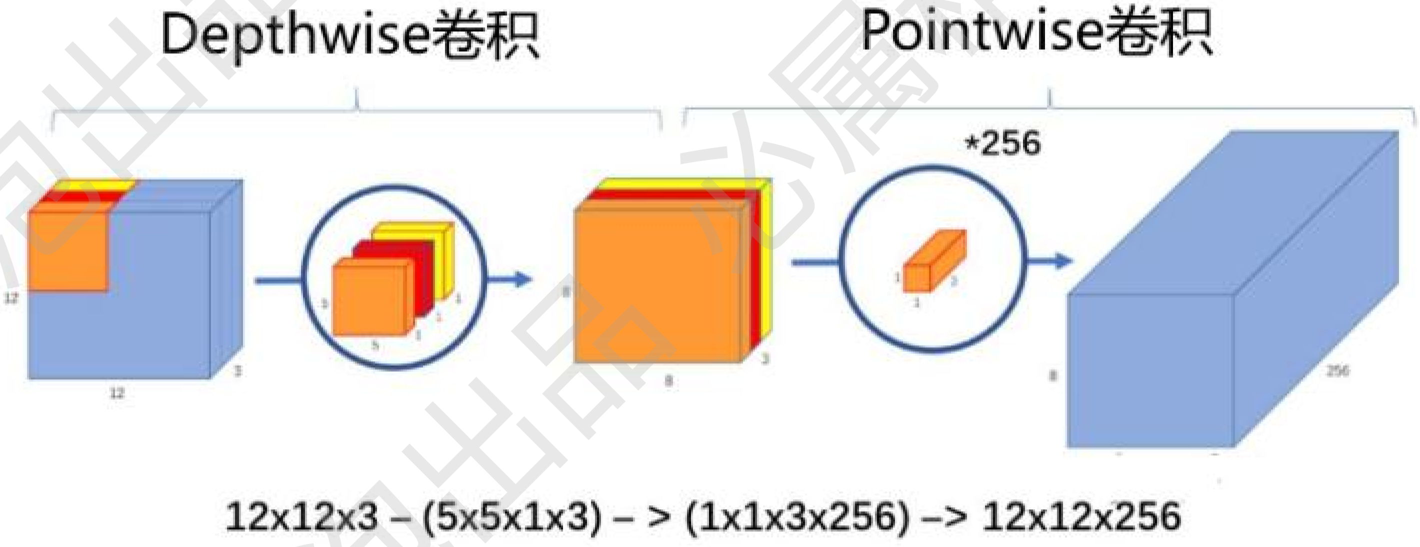


MobileNet

- Depthwise Separable卷积（之前那俩哥们合体）



  - 一件事分两次做了，但是效果不减
  - 虽然感觉麻烦点，但参数少
  - 接下来，该算账了！



MobileNet

- 参数量比较

- 传统卷积：$D_K \times D_K \times M \times N$

  （Dk为卷积核大小，M为输入通道数，N为卷积核个数）

- Depthwise Separable卷积：$D_K \times D_K \times M + M \times N$

  Depthwise卷积　　Pointwise卷积

- 算账：$$\frac{D_K \times D_K \times M + M \times N}{D_K \times D_K \times M \times N} = \frac{1}{N} + \frac{1}{D_K^2}$$

MobileNet

- 计算量比较

传统卷积：$D_K \times D_K \times M \times N \times D_W \times D_H$

（Dk为卷积核大小，M为输入通道数，N为卷积核个数，DH,DW为输入大小）

MobileNet：$D_K \times D_K \times M \times D_W \times D_H + M \times N \times D_W \times D_H$

Depthwise卷积　　Pointwise卷积

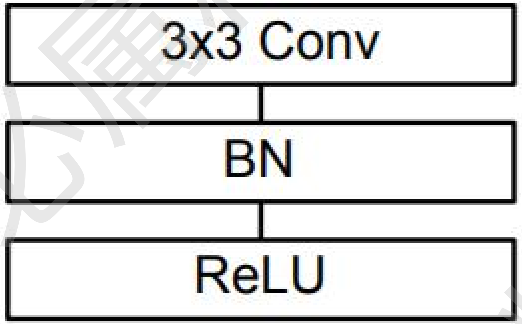
算账：$$\frac{D_K \times D_K \times M \times D_W \times D_H + M \times N \times D_W \times D_H}{D_K \times D_K \times M \times N \times D_W \times D_H} = \frac{1}{N} + \frac{1}{D_K^2}$$

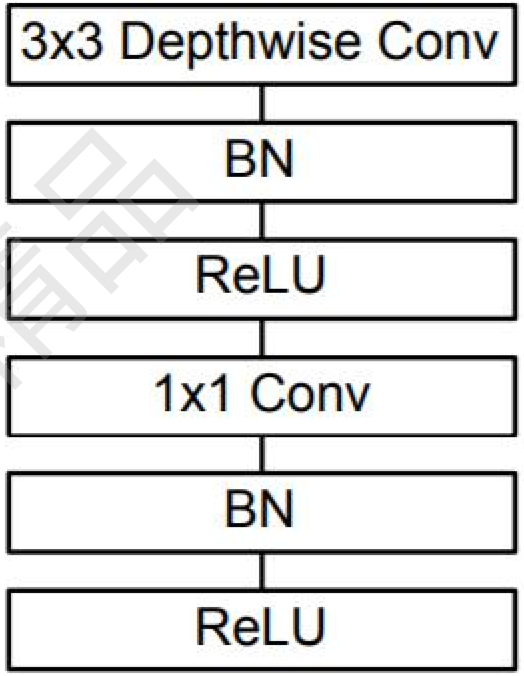
MobileNet

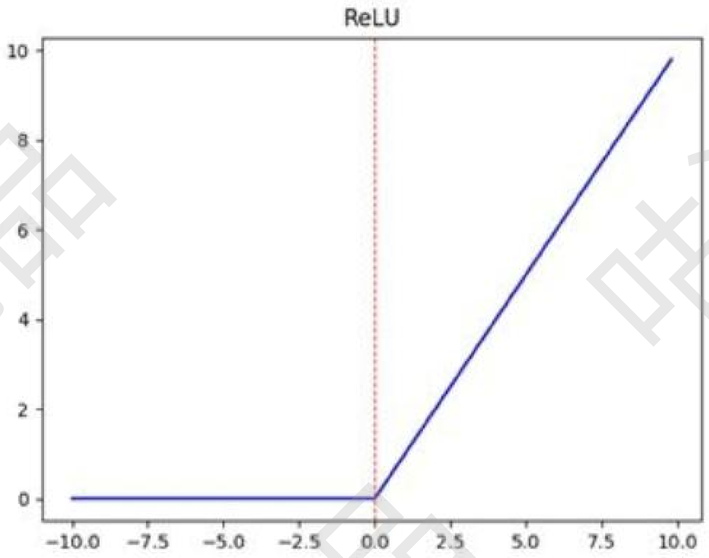
- 基础单元变化

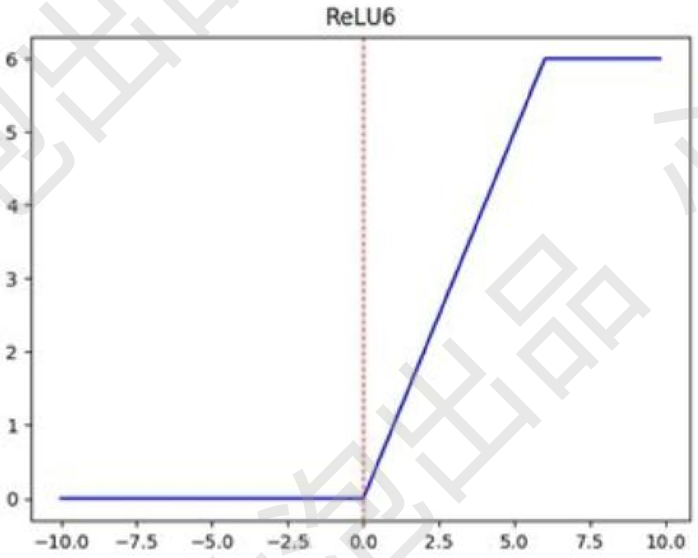
  - 分两步走，还来了个老六

  - 增加非线性与泛化能力









MobileNet

- 网络结构

  - 使用stride来进行降采样

  - 参数集中在1*1的卷积核中

Table 2. Resource Per Layer Type

Type	Mult-Adds	Parameters
Conv 1 × 1	94.86%	74.59%
Conv DW 3 × 3	3.06%	1.06%
Conv 3 × 3	1.19%	0.02%
Fully Connected	0.18%	24.33%

Table 1. MobileNet Body Architecture

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

MobileNet

网络效果

不同卷积的效果：

Table 4. Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

与经典网络对比：

Table 8. MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

MobileNet

- 再度压缩

- 按比例减少通道数，例如0.25，0.5，0.75等进行倍率改变

$$D_K \times D_K \times \alpha M \times D_F \times D_F + \alpha M \times \alpha N \times D_F \times D_F$$

- 压缩后效果：

Table 6. MobileNet Width Multiplier

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

MobileNet

- 再度压缩

- 按比例降低特征图大小，例如从224*224变成192*192

$$D_K \times D_K \times \alpha M \times \rho D_F \times \rho D_F + \alpha M \times \alpha N \times \rho D_F \times \rho D_F$$

- 压缩后效果：

Table 7. MobileNet Resolution

Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

MobileNet

✓ 应用表现

Table 13. COCO object detection results comparison using different frameworks and network architectures. mAP is reported with COCO primary challenge metric (AP at IoU=0.50:0.05:0.95)

Framework Resolution	Model	mAP	Billion Mult-Adds	Million Parameters
SSD 300	deeplab-VGG	21.1%	34.9	33.1
	Inception V2	22.0%	3.8	13.7
	MobileNet	19.3%	1.2	6.8
Faster-RCNN 300	VGG	22.9%	64.3	138.5
	Inception V2	15.4%	118.2	13.3
	MobileNet	16.4%	25.2	6.1
Faster-RCNN 600	VGG	25.7%	149.6	138.5
	Inception V2	21.9%	129.6	13.3
	Mobilenet	19.8%	30.5	6.1

MobileNet

- V2版本

  - Relu激活函数改变，用线性替代

  - 整体网络架构改变，类似resnet,但是反过来了

  - 各方面表现效果较V1都有不错的提升

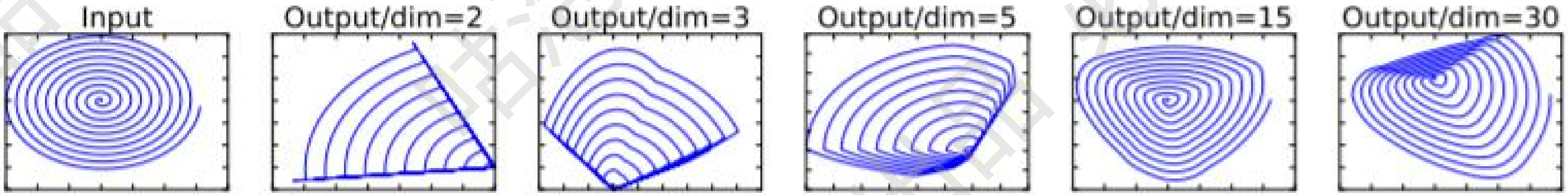
MobileNet

- Relu干了一件坏事

对不同输入特征进行对比分析，$y_{m\times n} = ReLU(T_{m\times 2} \cdot x_{2\times n})$

$$\tilde{x}_{2\times n} = T^{-1}_{2\times m} \cdot y_{m\times n}$$

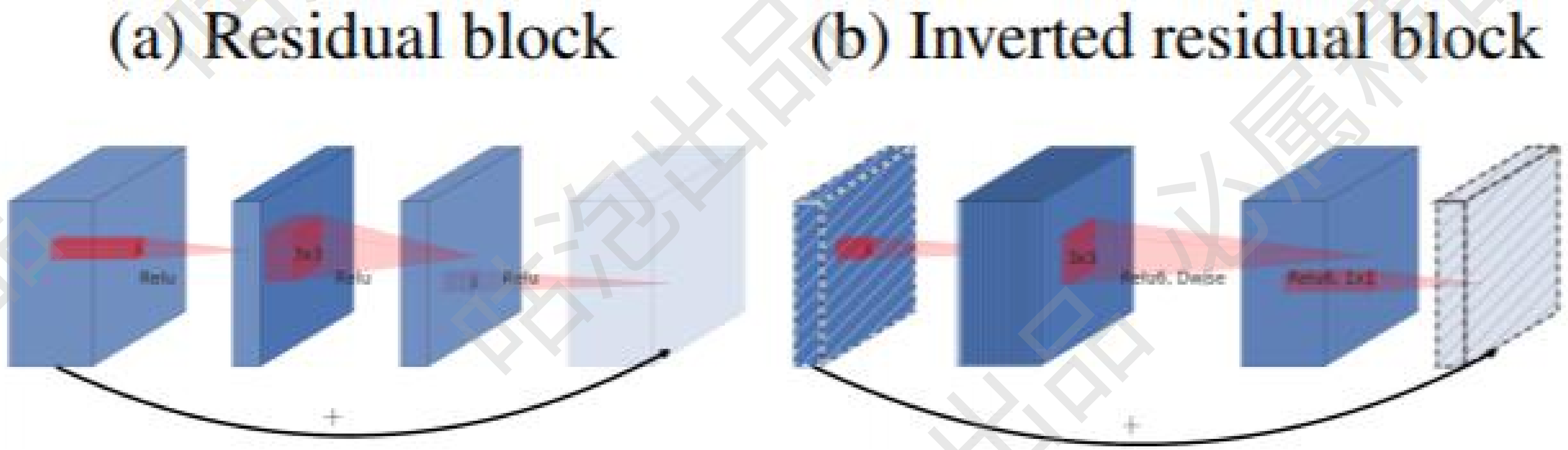
首先将2维特征经过矩阵T映射到M维，再进行relu计算，再还原回2维



MobileNet

- Inverted Residuals

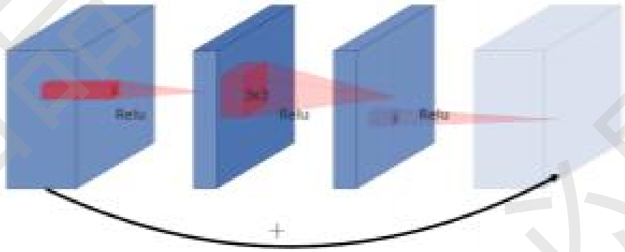
其实就是把残差模块给反过来了，为什么要升维后再接Relu呢？



MobileNet

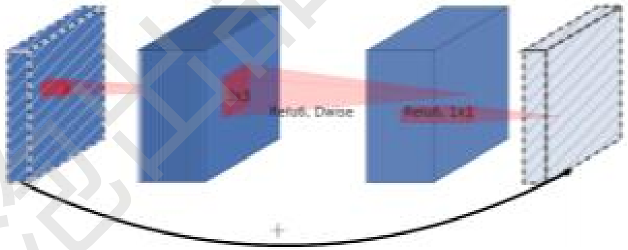
- 为什么改变残差模块

- 原始残差模块：



- 先1*1卷积进行压缩，然后3*3进行特征提取，再1*1卷积得到更多特征图（还原）

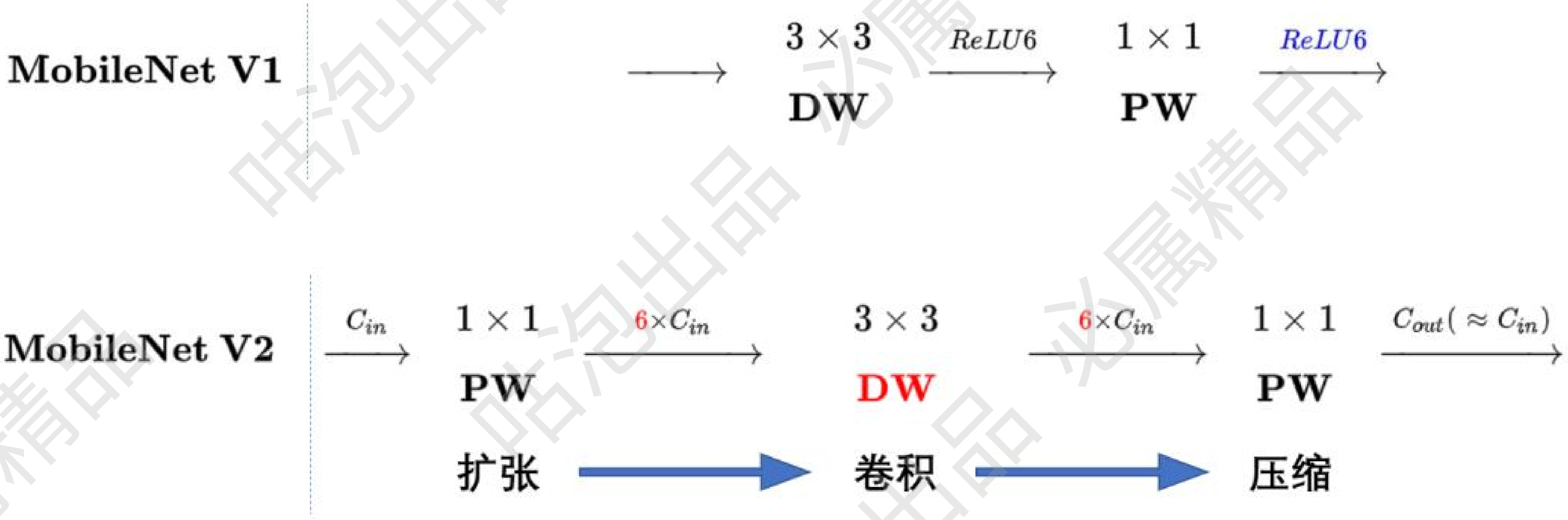
- Inverted Residuals：



- 先1*1扩充通道（ Depthwise 无法扩充），再使用3*3的Depthwise，最后再使用1*1卷积来进行还原

MobileNet

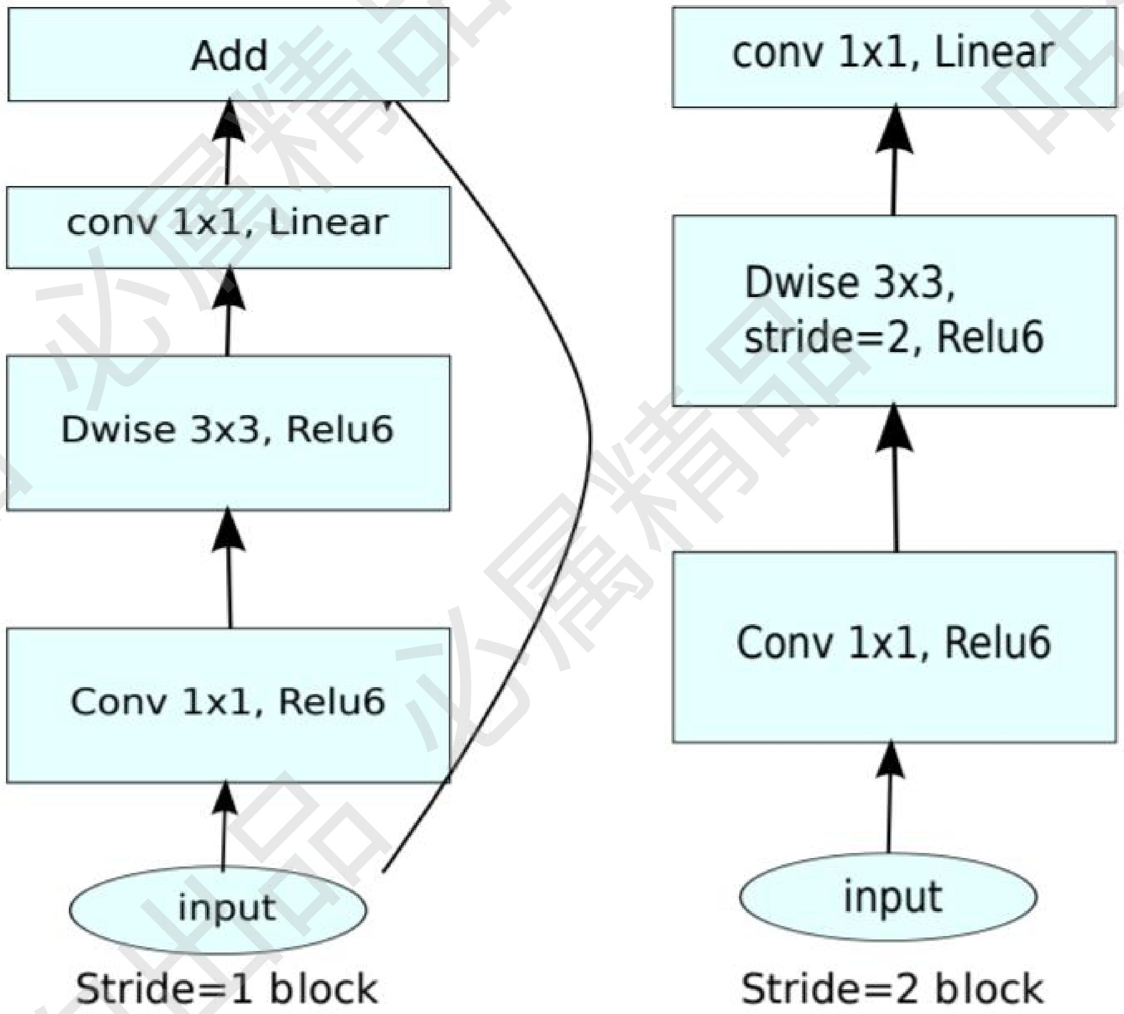
- V2结构分析



MobileNet

V2结构分析

- 整体模块特点：
- 先升维再Relu，减少破坏
- 残差链接要考虑特征图是否相同



MobileNet

- V2整体网络
  - t为倍率，n为次数
  - Shotcut结构，网络更深

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

MobileNet

- V2版本
  - 改进后的效果：

Network	Top 1	Params	MAdds	CPU
MobileNetV1	70.6	4.2M	575M	113ms
ShuffleNet (1.5)	71.5	**3.4M**	292M	-
ShuffleNet (x2)	73.7	5.4M	524M	-
NasNet-A	74.0	5.3M	564M	183ms
MobileNetV2	**72.0**	**3.4M**	**300M**	**75ms**
MobileNetV2 (1.4)	**74.7**	6.9M	585M	**143ms**

  - 应用效果：

Network	mAP	Params	MAdd	CPU
SSD300[34]	23.2	36.1M	35.2B	-
SSD512[34]	26.8	36.1M	99.5B	-
YOLOv2[35]	21.6	50.7M	17.5B	-
MNet V1 + SSDLite	22.2	5.1M	1.3B	270ms
MNet V2 + SSDLite	22.1	**4.3M**	**0.8B**	200ms

MobileNet

- V3版本

  - 引入Squeeze- Excitation结构

  - 标题的名字： SearchingforMobileNetV3

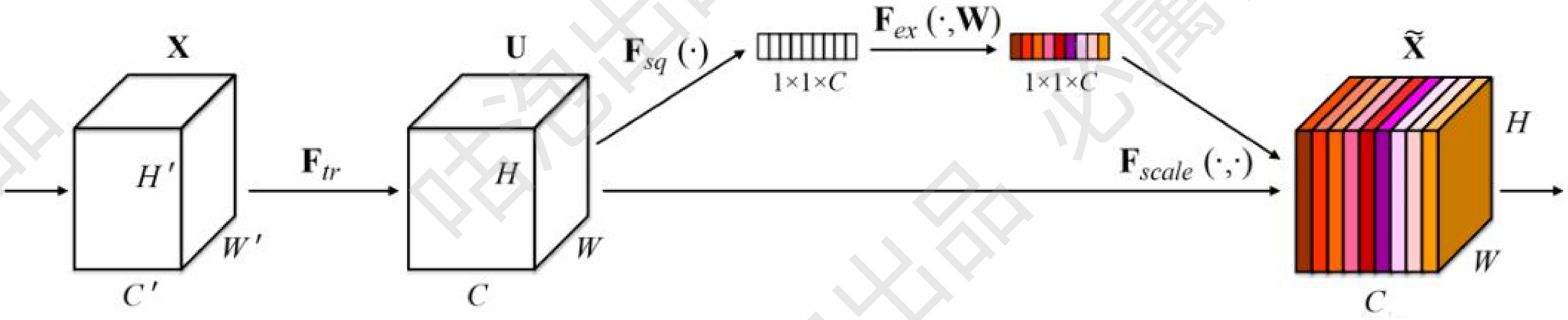
  - 非线性变换改变， h-swish替换swish

MobileNet

- SE-net

  - NLP中经常说到attention机制，那么特征图里有木有呢

  - 这个就是SE-net整体的结构，它可以融入到任何网络模型中



MobileNet

- S：操作

  - 就是对特征图采取全局平均池化，得到1*1*C的结果

  - 特征图中每个通道都相当于描述了一部分特征，操作后相当于是全局的

$$z_c = \mathbf{F}sq(\mathbf{u}_c) = \frac{1}{W \times H}\sum_{i=1}^{W}\sum_{j=1}^{H} u_c(i,j)$$

MobileNet

E：Excitation操作

现在想得到每个特征图的重要程度评分，还需要再来两个全连接层

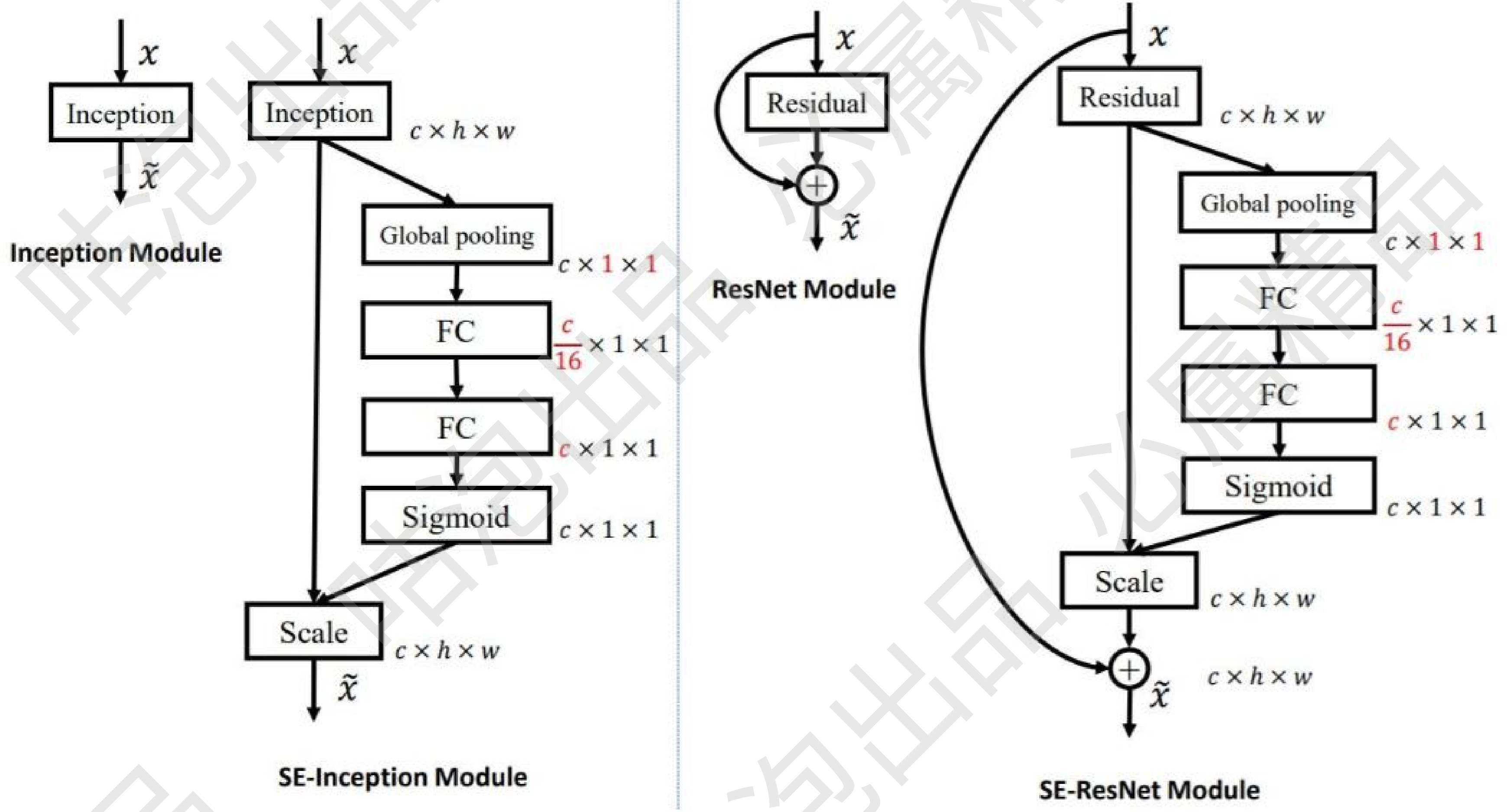
最终整个结果也是1*1*C，相当于attnetion：
z为全局描述，δ表示Relu函数，保证输出为正，σ表示sigmoid函数

$$\mathbf{s} = \mathbf{F}_{ex}(\mathbf{z}, \mathbf{W}) = \sigma(g(\mathbf{z}, \mathbf{W})) = \sigma(g(\mathbf{W}_2\delta(\mathbf{W}_1\mathbf{z})))$$

完成加权操作：$\tilde{x}_c = \mathbf{F}_{scale}(\mathbf{u}_c, s_c) = s_c \cdot \mathbf{u}_c$

MobileNet

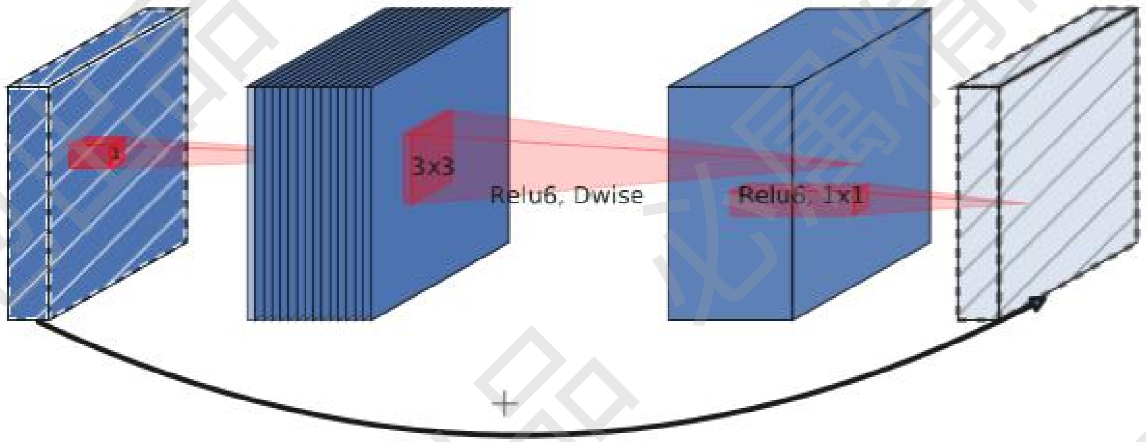
- RE模块可以和很多经典模型融合



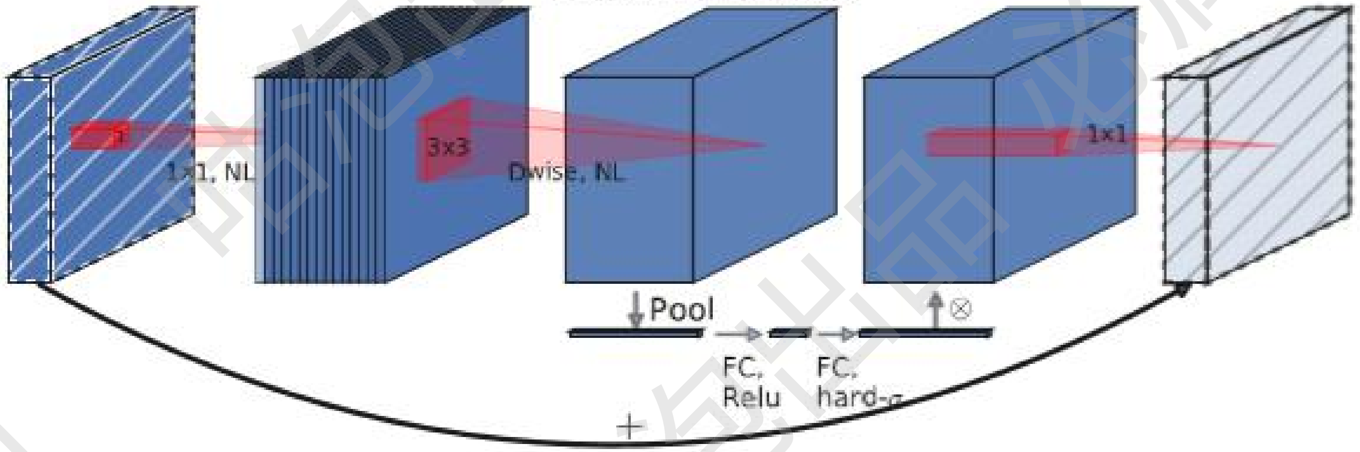
MobileNet

- 引入SE模块

V2版本：



V3版本：



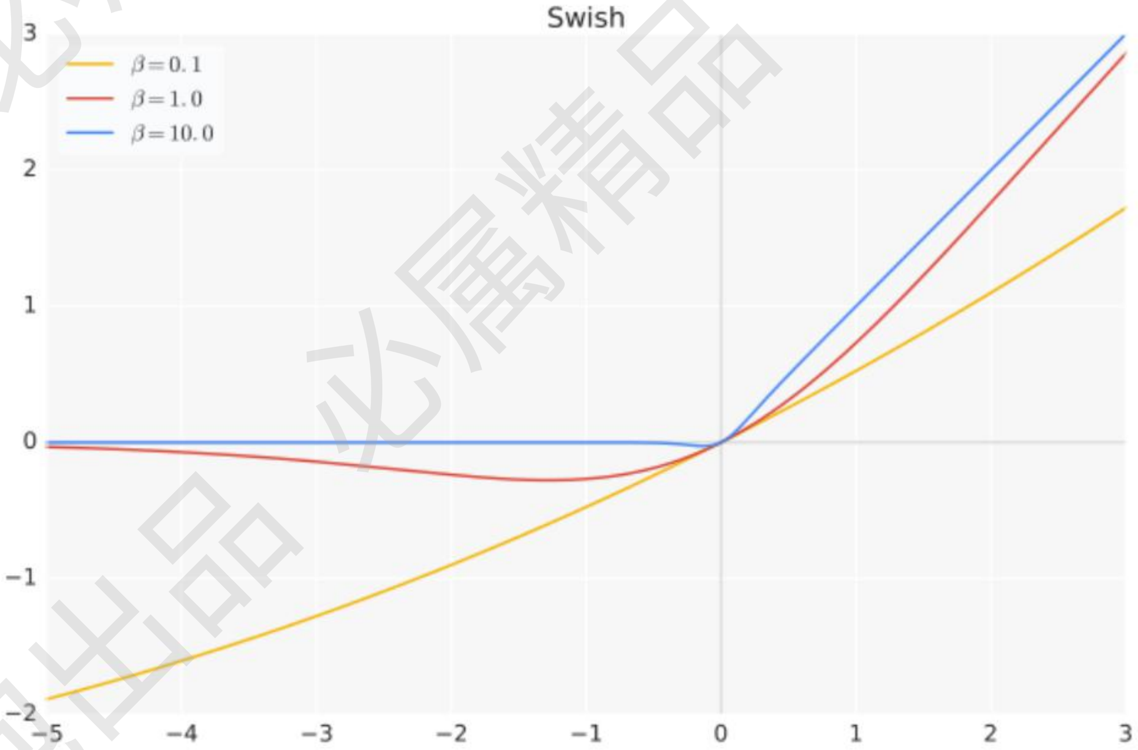
MobileNet

- Swish激活函数

  - 公式：$f(x) = x \cdot \mathrm{sigmoid}(\beta x)$

  - 无上界有下界、平滑、非单调

  - Swish 替换 ReLU，效果会好一丢丢



MobileNet

- hard-swish

  - 使用Relu来模拟sigmoid

$$\text{h-swish}[x] = x\frac{\text{ReLU6}(x+3)}{6}$$

  - Relu实现的更高效：

	Top 1	Latency P-1
V3	75.2	66
0.85 V3	74.3	55
ReLU	74.5 (-.7%)	59 (-12%)
h-swish @16	75.4 (+.2 %)	78 (+20%)
h-swish @112	75.0 (-.3%)	64 (-3%)

Table 5. Effect of non-linearities on MobileNetV3-Large. In h-swish @N, N denotes the number of channels, in the first layer that has h-swish enabled.

MobileNet

- 网络结构（large vs small）

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	2
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	conv2d, 1x1	-	960	-	HS	1
$7^2 \times 960$	pool, 7x7	-	-	-	-	1
$1^2 \times 960$	conv2d 1x1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	conv2d 1x1, NBN	-	k	-	-	1

Input	Operator	exp size	#out	SE	NL	s
$224^2 \times 3$	conv2d, 3x3	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	✓	RE	2
$56^2 \times 16$	bneck, 3x3	72	24	-	RE	2
$28^2 \times 24$	bneck, 3x3	88	24	-	RE	1
$28^2 \times 24$	bneck, 5x5	96	40	✓	HS	2
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	120	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	144	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	288	96	✓	HS	2
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	conv2d, 1x1	-	576	✓	HS	1
$7^2 \times 576$	pool, 7x7	-	-	-	-	1
$1^2 \times 576$	conv2d 1x1, NBN	-	1024	-	HS	1
$1^2 \times 1024$	conv2d 1x1, NBN	-	k	-	-	1

MobileNet

- 效果分析（ImageNet,详情见论文）

Network	Top-1	MAdds	Params	P-1	P-2	P-3
V3-Large 1.0	**75.2**	**219**	5.4M	**51**	**61**	**44**
V3-Large 0.75	73.3	155	4.0M	39	46	40
MnasNet-A1	75.2	315	3.9M	71	86	61
Proxyless[5]	74.6	320	4.0M	72	84	60
V2 1.0	72.0	300	3.4M	64	76	56
V3-Small 1.0	**67.4**	**56**	2.5M	**15.8**	**19.4**	**14.4**
V3-Small 0.75	65.4	44	2.0M	12.8	15.6	11.7
Mnas-small [43]	64.9	65.1	1.9M	20.3	24.2	17.2
V2 0.35	60.8	59.2	1.6M	16.6	19.6	13.9

Table 3. Floating point performance on the Pixel family of phones (P-n denotes a Pixel-n phone). All latencies are in ms and are measured using a single large core with a batch size of one. Top-1 accuracy is on ImageNet.