MobileNetV2:

《Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation》

于2018年1月公开在arXiv(美['ɑ:rkaɪv]) :
https://arxiv.org/abs/1801.04381

MobileNetV2是对MobileNetV1的改进，同样是一个轻量化卷积神经网络。

创新点:

1. Inverted residuals，通常的residuals block是先经过一个1*1的Conv layer，把feature map的通道数“压”下来，再经过3*3 Conv layer，最后经过一个1*1 的Conv layer，将feature map 通道数再“扩张”回去。即先“压缩”，最后“扩张”回去。

而 inverted residuals就是 先“扩张”，最后“压缩”。为什么这么做呢？请往下看。

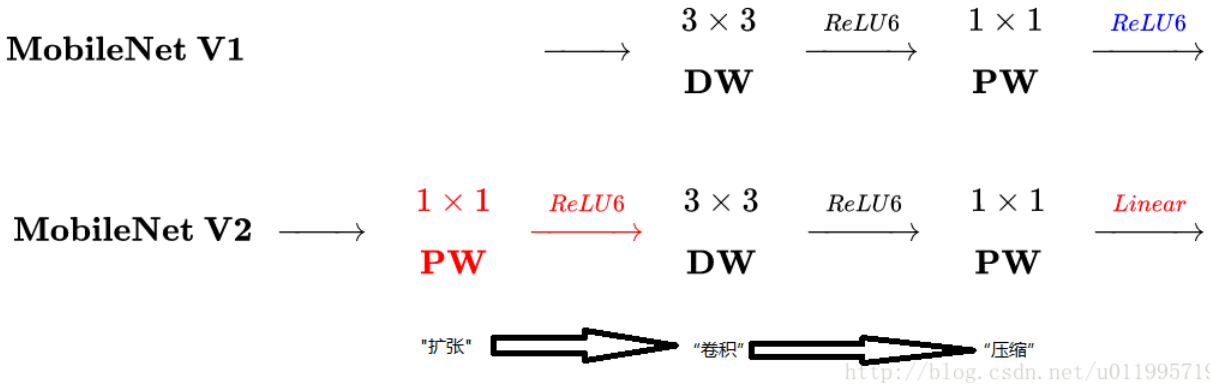
2.Linear bottlenecks，为了避免Relu对特征的破坏，在residual block的Eltwise sum之前的那个 1*1 Conv 不再采用Relu，为什么？请往下看。

创新点全写在论文标题上了！

由于才疏学浅，对本论文理论部分不太明白，所以选取文中重要结论来说明MobileNet-V2。

先看看MobileNetV2 和 V1之间有啥不同

（原图链接）

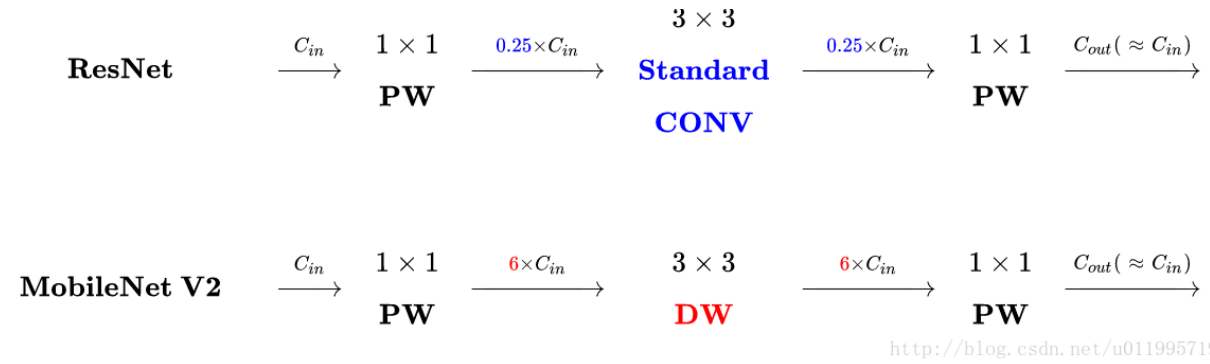


主要是两点：

1. Depth-wise convolution之前多了一个1*1的“扩张”层，目的是为了提升通道数，获得更多特征；
2. 最后不采用Relu，而是Linear，目的是防止Relu破坏特征。

再看看MobileNetV2的block 与ResNet 的block：

（原图链接）



主要不同之处就在于，ResNet是：压缩”→“卷积提特征”→“扩张”，MobileNetV2则是Inverted residuals,即：“扩张”→“卷积提特征”→ “压缩”

正文：

MobileNet-V1 最大的特点就是采用depth-wise separable convolution来减少运算量以及参数量，而在网络结构上，没有采用shortcut的方式。

Resnet及Densenet等一系列采用shortcut的网络的成功，表明了shortcut是个非常好的东西，于是MobileNet-V2就将这个好东西拿来用。

拿来主义，最重要的就是要结合自身的特点，MobileNet的特点就是depth-wise separable convolution，但是直接把depth-wise separable convolution应用到 residual block中，会碰到如下问题：

1.DWConv layer层提取得到的特征受限于输入的通道数，若是采用以往的residual block，先“压缩”，再卷积提特征，那么DWConv layer可提取得特征就太少了，因此一开始不“压缩”，MobileNetV2反其道而行，一开始先“扩张”，本文实验“扩张”倍数为6。 通常residual block里面是 “压缩”→“卷积提特征”→“扩张”，MobileNetV2就变成了 “扩张”→“卷积提特征”→ “压缩”，因此称为*Inverted residuals*

2.当采用“扩张”→“卷积提特征”→ “压缩”时，在“压缩”之后会碰到一个问题，那就是Relu会破坏特征。为什么这里的Relu会破坏特征呢？这得从Relu的性质说起，Relu对于负的输入，输出全为零；而本来特征就已经被“压缩”，再经过Relu的话，又要“损失”一部分特征，因此这里不采用Relu，实验结果表明这样做是正确的，这就称为*Linear bottlenecks*

MobileNet-V2网络结构

附赠苏师兄的prototxt：

https://github.com/suzhenghang/MobileNetv2/tree/master/.gitignore

另外一位朋友的prototxt：

https://github.com/austingg/MobileNet-v2-caffe

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
$28^2 \times 64$	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	-
$1 \times 1 \times k$	conv2d 1x1	-	k	-	

Σ=17≠19

其中：t表示“扩张”倍数，c表示输出通道数，n表示重复次数，s表示步长stride。

先说两点有误之处吧：

1. 第五行，也就是第7~10个bottleneck，stride=2，分辨率应该从28降低到14；如果不是分辨率出错，那就应该是stride=1；

2. 文中提到共计采用19个bottleneck，但是这里只有17个。

Conv2d 和avgpool和传统CNN里的操作一样；最大的特点是bottleneck，一个bottleneck由如下三个部分构成：

Input	Operator	Output	
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$	”扩张”
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$	”卷积”
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$	”压缩”

这就是之前提到的inverted residuals结构，一个inverted residuals结构的Multiply Add=

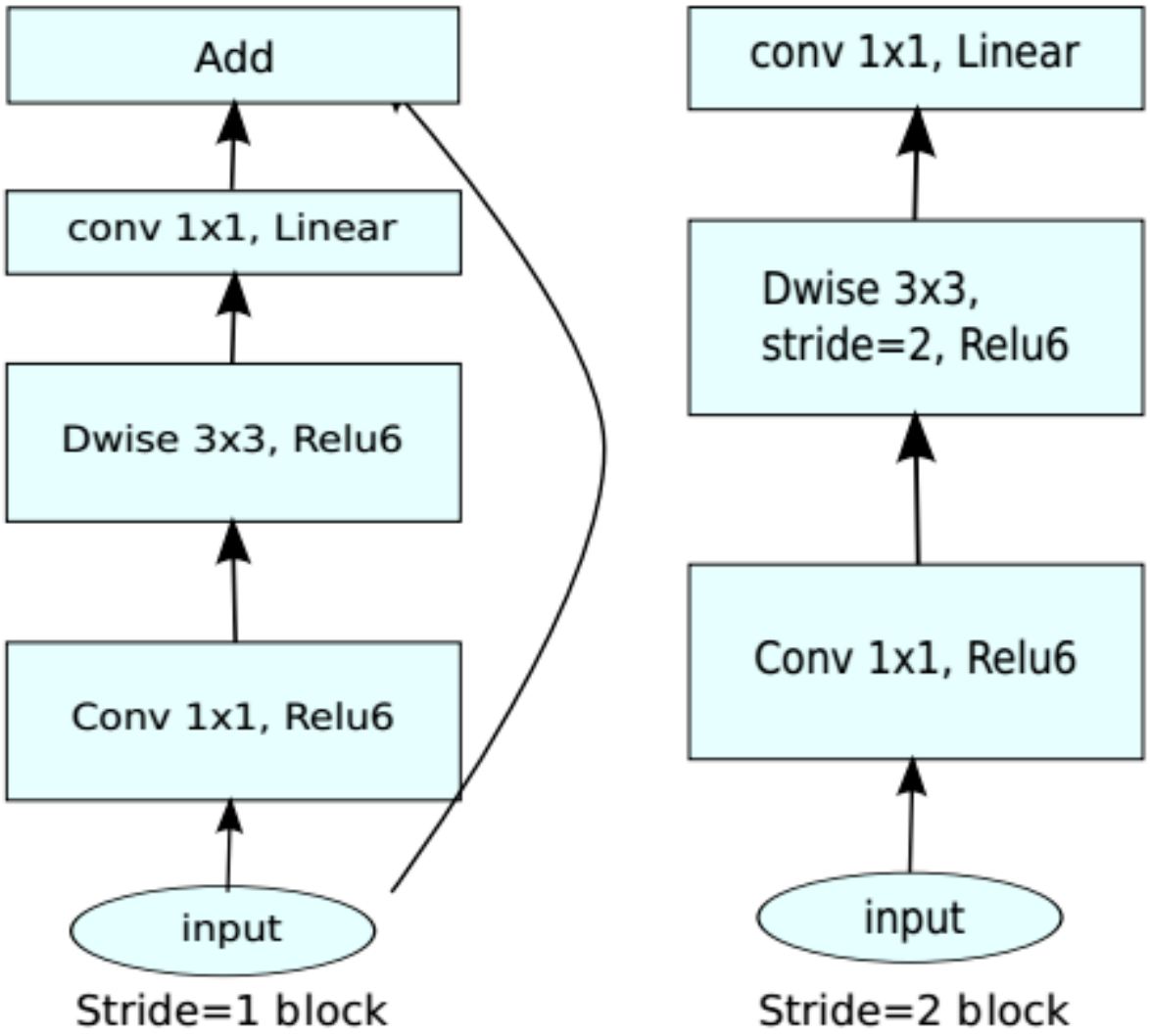
h*w*d’ * 1*1*td’ +

h*w*td’ * k*k*1 +

h*w*t d’ * 1*1*d” =

h*w*d’*t(d’ + k*k + d”)

特别的，针对stride=1 和stride=2，在block上有稍微不同，主要是为了与shortcut的维度匹配，因此，stride=2时，不采用shortcut。 具体如下图：



(d) Mobilenet V2

可以发现，除了最后的avgpool，整个网络并没有采用pooling进行下采样，而是利用stride=2来下采样，此法已经成为主流，不知道是否pooling层对速度有影响，因此舍弃pooling层?是否有朋友知道那篇论文里提到这个操作？

看看MobileNet-V2 分类时，inference速度：

Network	Top 1	Params	MAdds	CPU
MobileNetV1	70.6	4.2M	575M	123ms
ShuffleNet (1.5)	69.0	**2.9M**	292M	-
ShuffleNet (x2)	70.9	4.4M	524M	-
NasNet-A	74.0	5.3M	564M	192ms
MobileNetV2	**71.7**	**3.4M**	**300M**	**80ms**
MobileNetV2 (1.4)	**74.7**	6.9M	585M	**149ms**

这是在手机的CPU上跑出来的结果(Google pixel 1 for TF-Lite)

同时还进行了目标检测和图像分割实验，效果都不错，详细请看原文。