MobileNet-V2

论文地址：MobileNetV2: Inverted Residuals and Linear Bottlenecks

1 创新点

MobileNetV2是对MobileNetV1的改进，同样是一个轻量化卷积神经网络，v2版本效果提升非常明显。主要创新点已经写在标题上了。

(1) Linear Bottlenecks。也就是去掉了小维度输出层后面的非线性激活层，目的是为了保证模型的表达能力，后面细说

(2) Inverted Residual block。该结构和传统residual block中维度先缩减再扩增正好相反，因此shotcut也就变成了连接的是维度缩减后的feature map

简单来说，mobilenetv2是基于mobilenetv1的通道可分离卷积优势，然后再加入了残差模块功能，最后通过理论分析和实验证明，对残差模块进行了一定程度的修改，使其在移动端发挥最大性能。

2 核心思想

MobileNet V1中主要是引入了depthwise separable convolution代替传统的卷积操作，相当于实现了spatial和channel之间的解耦，达到模型加速的目的，整体网络结构还是延续了VGG网络直上直下的特点。可以看出 MobileNet v1 的结构其实非常简单， 是一个非常原始的直筒结构， 类似于VGG一样。 这种结构的性价比其实不高， 后续一系列的ResNet, DenseNet等结构已经证明通过复用图像特征， 使用concat/eltwise+ 等操作进行融合， 能极大提升网络的性价比。自然而言，mobilenet中也应该引入残差块。

我们知道，mobileNet V1是应用了depthwise separable convolution进行加速，但是论文所设计的结构其实存在缺陷。在实际使用的时候， 可以发现Depthwise 部分的kernel比较容易训废掉即训完之后发现depthwise训出来的kernel有不少是空的，即大部分输出为0，这个问题在定点化低精度训练的时候会进一步放大。针对这个问题，本论文进行了深入分析。

假设经过激活层后的张量被称为兴趣流形(manifold of interest)，具有维HxWxD。根据研究，兴趣流形可能仅分布在激活空间的一个低维子空间里，或者说兴趣流形是可以用一个低维度空间来表达的，这实际上是说神经网络某层特征，其主要信息可以用一个低维度特征来表征。据此，我们可以在网络引入“bottlenck layer”来表征某一个深层特征的“主要流形”，具体实现上，很容易使用1x1卷积将张量降维，但由于ReLU的存在，这种降维实际上会损失较多的信息，如下图：

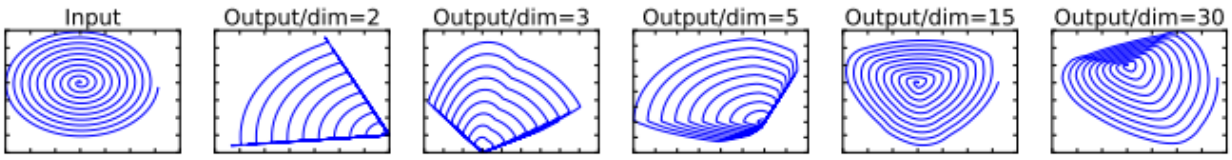


Figure 1: Examples of ReLU transformations of low-dimensional manifolds embedded in higher-dimensional spaces. In these examples the initial spiral is embedded into an n-dimensional space using random matrix T followed by ReLU, and then projected back to the 2D space using T^{-1}. In examples above $n = 2, 3$ result in information loss where certain points of the manifold collapse into each other, while for $n = 15$ to 30 the transformation is highly non-convex.

上图中，利用MxN的矩阵T将张量（2D，即N=2）变换到M维的空间中，通过ReLU后（y=ReLU(Bx)），再用此矩阵T之逆恢复原来的张量。可以看到，当M较小时，恢复后的张量坍缩严重，M较大(30)时则恢复较好。这意味着，**在较低维度的张量表示（兴趣流形）上进行ReLU等线性变换会有很大的信息损耗。**显然，当把原始输入维度增加到15或30后再作为ReLU的输入，输出恢复到原始维度后基本不会丢失太多的输入信息；相比之下如果原始输入维度只增加到2或3后再作为ReLU的输入，输出恢复到原始维度后信息丢失较多。**因此在MobileNet V2中，执行降维的卷积层后面不会接类似ReLU这样的非线性激活层，也就是linear bottleneck。**总结如下：

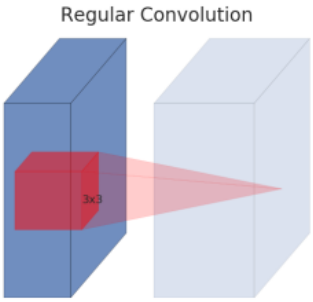
(1) 对于ReLU层输出的非零值而言，ReLU层起到的就是一个线性变换的作用

(2) ReLU层可以保留input manifold的信息，但是只有当input manifold是输入空间的一个低维子空间时才有效。

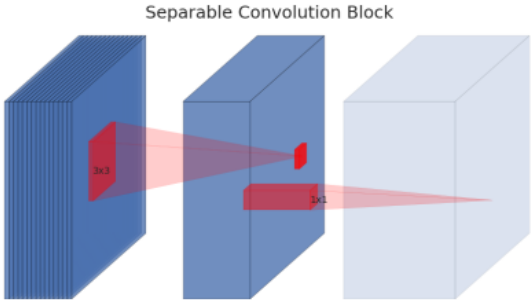
简要说明下，第一点说明ReLu会保留所有非0输入值，而将所有0的输入值一直维持为0(梯度也为0)，不再生效。ReLu的线性变换作用仅仅适用于非0值。第二点说明如果输入的数值正好是输入空间的一个低维子空间时，经过ReLu线性激活层后不会损失信息，此时允许输入维度较低；但是通常难以满足，如果输入维度较低，经过ReLu后会损失很多信息，为了确保信息不会丢失，一般都要求升维。

我们知道残差结构是先压缩维度，再3x3卷积，然后再扩展到原始维度，最后进行求和操作。之所以要先压缩的目的是减少3x3卷积计算量，但是我们这里是使用depth-wise卷积的，其计算量本来就很小，基于前面的分析，如果压缩后，再3x3的depth-wise，然后再进行ReLu，会丢失大部分信息，就会出现depthwise训出来的kernel有不少是空的情况。所以作者采用了一种反向残差结构，即**先扩展，再depth-wise，再压缩的残差结构，且压缩的1x1卷积不使用ReLu激活函数。**

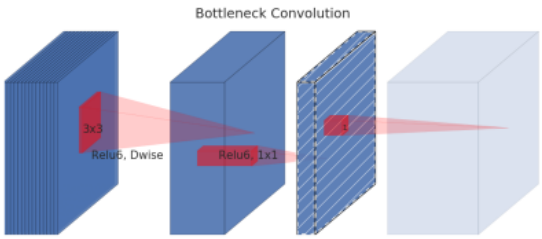
(a) Regular



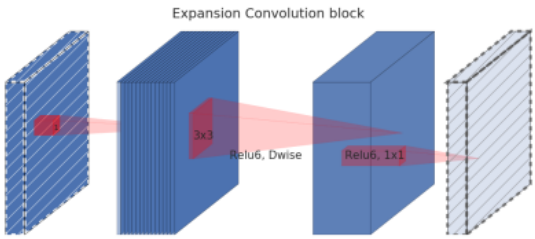
(b) Separable



(c) Separable with linear bottleneck

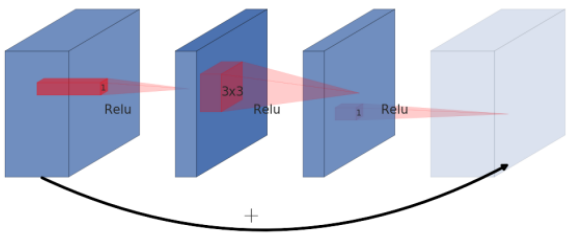


(d) Bottleneck with expansion layer

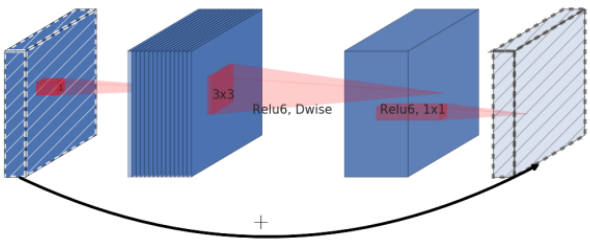


如上图所示，(a)是普通卷积，(b)是深度可分离卷积，(c)和(d)是修改的深度可分离卷积，具体体现在虚线部分不再使用ReLu等线性激活函数，(c)和(d)是功能相同的两种不同堆叠的结构。最终设计的反向残差结构如下：

(a) Residual block

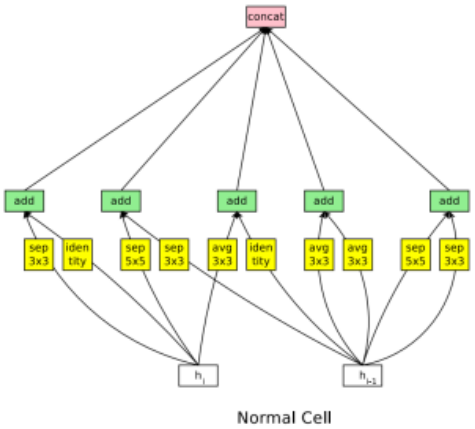


(b) Inverted residual block

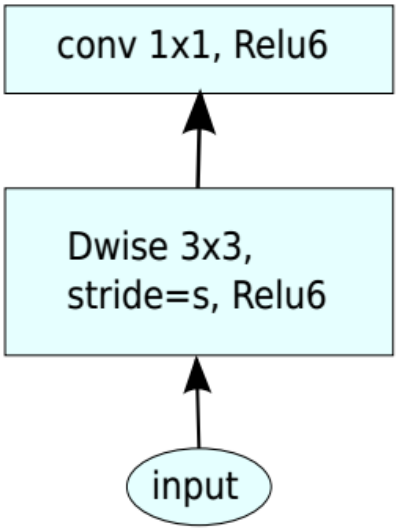


Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

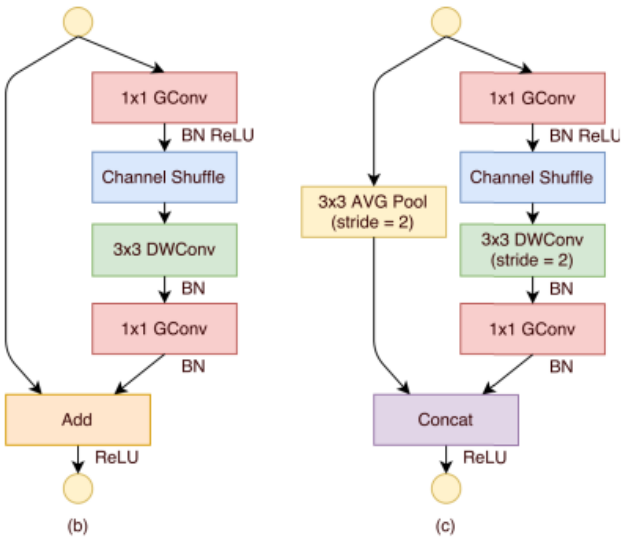
Table 1: *Bottleneck residual block* transforming from k to k' channels, with stride s, and expansion factor t.



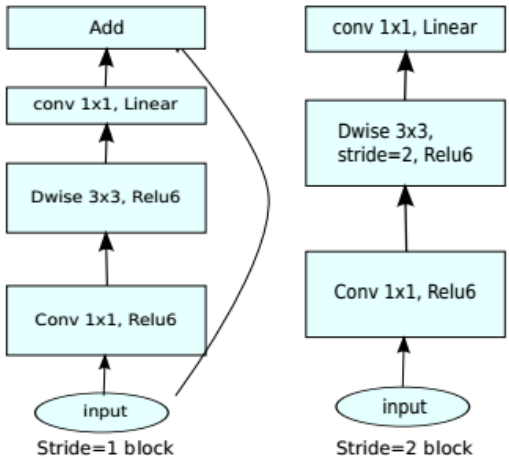
(a) NasNet[23]



(b) MobileNet[27]

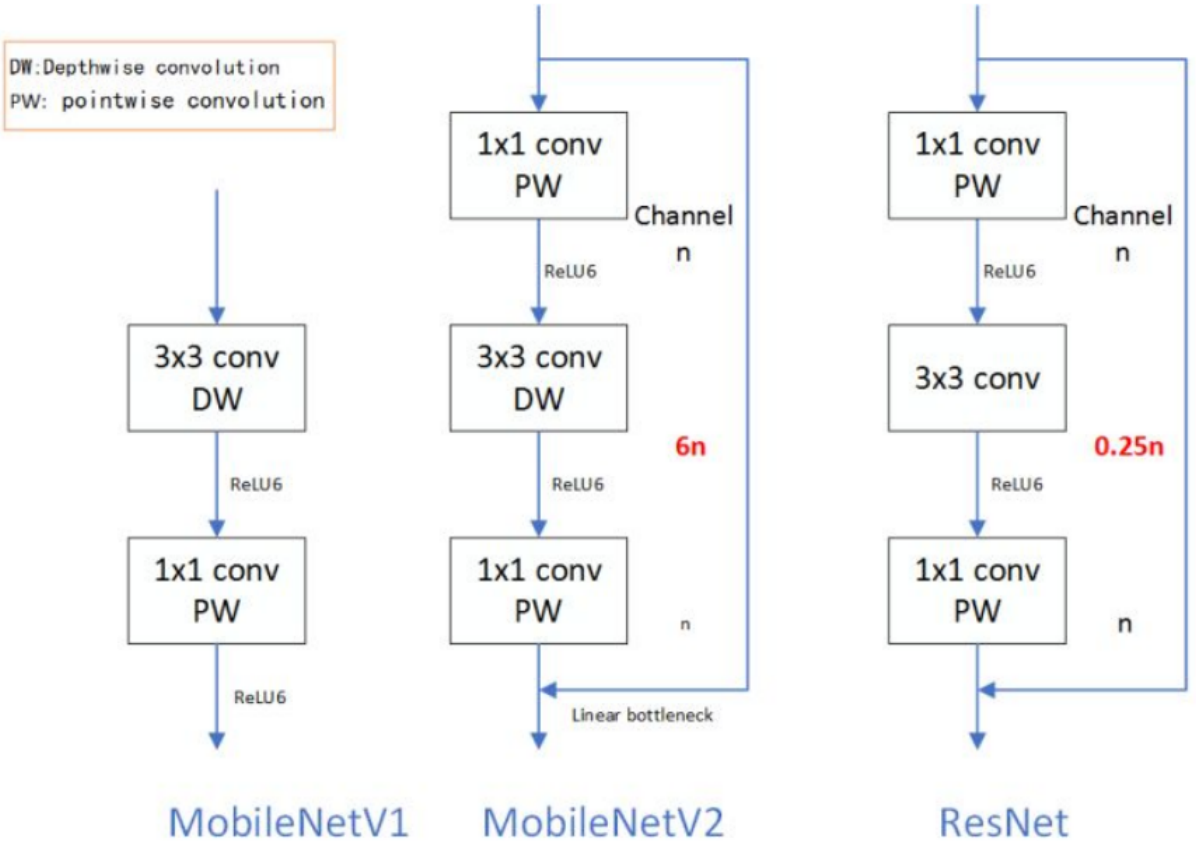


(c) ShuffleNet [20]



(d) Mobilenet V2

t是扩张系数，作者设置为6。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。

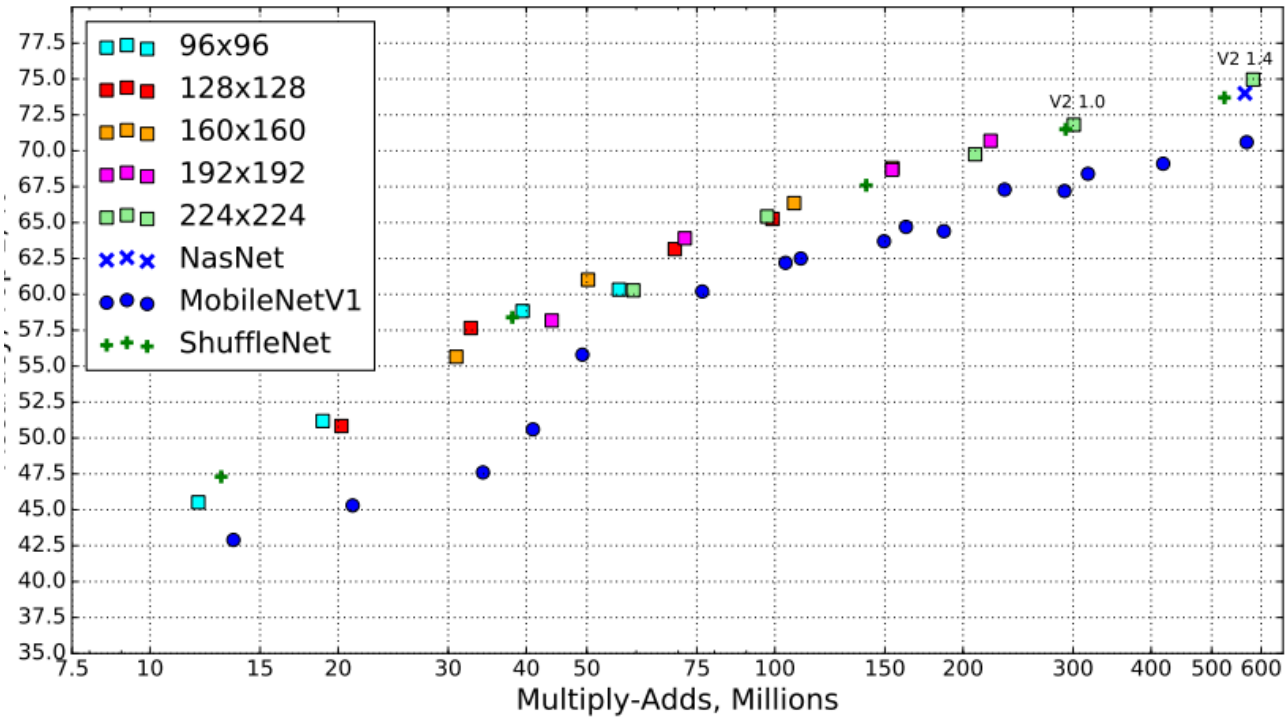


3 网络模型

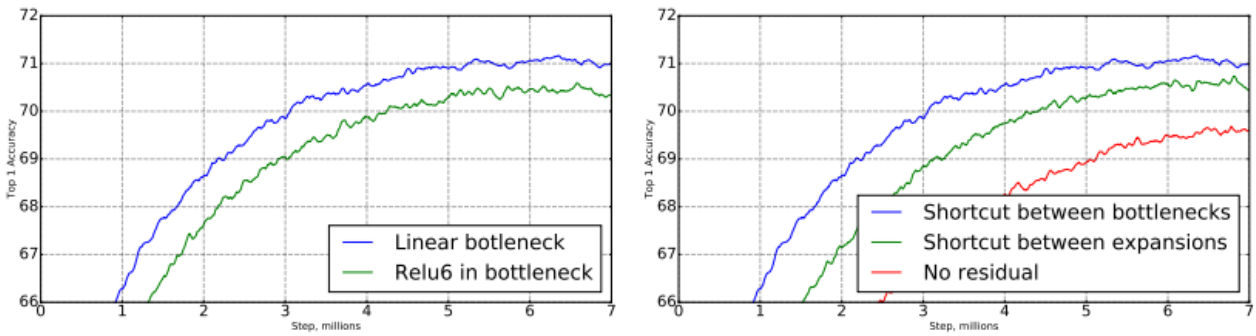
Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck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

t是扩张系数，c是输出通道数，n是重复次数，s是stride。可以看出这个基础网络都没有pool操作，而是应用stride代替。

4 实验结果



如上图可以看出：mobileNet-v2明显优于mobileNet-v1，且和NasNet性能一致。



(a) Impact of non-linearity in the bottleneck layer. (b) Impact of variations in residual blocks.

(a)可以看出，linear bottleneck的优点，(b)可以看出应该把shortcut连接到bottlenecks处，且使用了残差后提升明显。

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**

从上表可以看出，mobilenet v2在imagenet数据集上性能最好，且速度最快，效率最高。

	Params	MAdds
SSD[34]	14.8M	1.25B
SSDLite	**2.1M**	**0.35B**

Table 5: Comparison of the size and the computational cost between SSD and SSDLite configured with MobileNetV2 and making predictions for 80 classes.

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

上表为目标检测性能，SSDLite是将SSD的所有预测层的卷积全部替换为通道可分离卷积。MobileNet V2+SSDLite达到了极高的性能，且速度非常块。注意以上结果是在google的pixel 1手机上（无GPU），基于TF-Lite开发的测试结果，可以达到5FPS，依然无法达到实时性要求。

5 实现

重点是瓶颈层的实现，如下：

```
class ConvBNReLU(nn.Sequential):
    def __init__(self, in_planes, out_planes, kernel_size=3, stride=1, groups=1):
        padding = (kernel_size - 1) // 2
        super(ConvBNReLU, self).__init__(
            nn.Conv2d(in_planes, out_planes, kernel_size, stride, padding,
groups=groups, bias=False),
            nn.BatchNorm2d(out_planes),
            nn.ReLU6(inplace=True)
        )


class InvertedResidual(nn.Module):
    def __init__(self, inp, oup, stride, expand_ratio):
        super(InvertedResidual, self).__init__()
        self.stride = stride
        assert stride in [1, 2]

        hidden_dim = int(round(inp * expand_ratio))
        self.use_res_connect = self.stride == 1 and inp == oup

        layers = []
        if expand_ratio != 1:
            # pw
            layers.append(ConvBNReLU(inp, hidden_dim, kernel_size=1))
        layers.extend([
            # dw
            ConvBNReLU(hidden_dim, hidden_dim, stride=stride, groups=hidden_dim),
            # pw-linear
            nn.Conv2d(hidden_dim, oup, 1, 1, 0, bias=False),
            nn.BatchNorm2d(oup),
        ])
        self.conv = nn.Sequential(*layers)

    def forward(self, x):
        if self.use_res_connect:
            return x + self.conv(x)
        else:
            return self.conv(x)
```

pytorch的model里面已经实现了mobilenetv2结构，可以直接调用。

参考文献

[1] https://blog.ddlee.cn/posts/c9816b0a/

[2] https://www.zhihu.com/question/265709710