MobileNets

📌 背景

为了使模型取得更高的准确率，在设计时都倾向于将模型变得更深、更大，但这些也是需要付出相应的“代价”。大而复杂的模型训练时间长、推理时间长、内存占用大，并且将其部署到手机终端或者一些移动设备上都无法取得实时的效果甚至无法运行。因此就有研究者开始着眼于轻量化模型。

模型轻量化的目的在于尽可能地减少“成本”，且不牺牲模型的性能。得到一个轻量的模型有很多不同的办法，比如说使用模型压缩技术压缩已经训练好的网络。模型压缩包含了剪枝（通过删除冗余和无关紧要的参数来工作），权值量化（减少存储参数需要的比特数），知识蒸馏（用大而深的网络作为教师网络，让浅而小的学生网络去学习拟合相同的知识）。而MobileNets系列的网络都是直接训练轻量化网络。

📌 方法

🖌 MobileNet V1

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 Conv / s1	$3 \times 3 \times 512$ dw $1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$ $14 \times 14 \times 512$
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	$1 \times 1 \times 1000$

图1

MobileNetV1网络主要的贡献点就是深度可分离卷积，最后为了使模型可以更方便地应用到不同的移动设备，设计了两个超参数来灵活地调整网络的参数量和计算量。

深度可分离卷积

深度可分离卷积由一层深度方向的卷积和一层点卷积组合而成，每一层卷积之后都紧跟着BN层和ReLU激活函数。

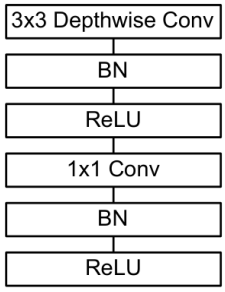

图2

- **深度方向卷积**

  如图3所示，是深度方向卷积操作，可以看出输入通道数=卷积核个数=输出通道数，且每个卷积核通道数为1。

  深度方向卷积操作一个通道只被一个卷积核卷积，没有利用不同通道在相同空间位置上的特征信息，因此还需要一种可以融合不同通道信息的卷积操作。

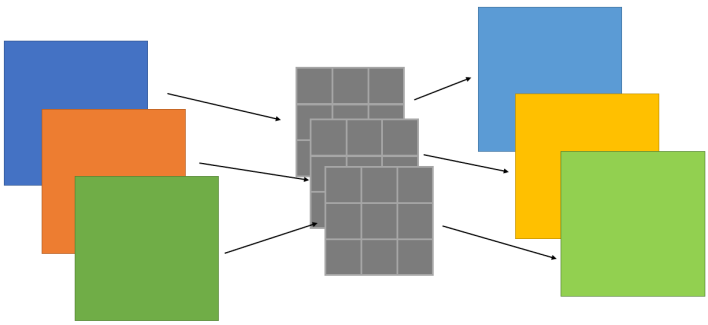

图3

- **点卷积**

如图4所示，点卷积操作每个卷积核通道数=输入通道数，输出通道数=卷积核个数，卷积核尺寸为1×1。

实现将输入特征图在通道方向上进行加权组合，融合不同通道之间的信息从而生成新的特征图。

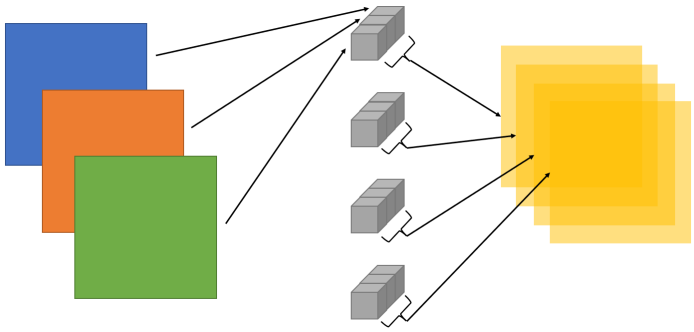

图4

- **比较标准卷积与深度可分离卷积计算量**

  假设输入一个尺寸为8×8×3的张量，希望得到的尺寸为8× 8× 256的输出张量，如果用普通的二维卷积（卷积核大小为5× 5），需要进行的计算量大小为：

  (8×8)×(5×5×3)×256=1,228,800

  如果使用深度可分离卷积，首先深度方向的卷积计算量大小为：

  (8×8)×(5×5×1)×3=4,800

  再使用一个点卷积：

  (8×8)×(1×1×3)×256=49,152

  所以计算量总计为：4,800+49,152=53,952

  从上面的例子中可以看出，使用深度可分离卷积确实有效地减少了计算量。

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

图5

从图5的实验结果中也可以看出，使用深度可分离卷积与使用标准卷积的MobileNet性能相比，虽然在准确率上降低了1%，但是参数量减少了很多。

两个超参数

即使现在得到的MobileNet从模型尺寸以及速度上已经足够小、足够快了，但是因为应用场合的多样以及还是有很多特殊的应用场景需要更小、更快的模型，为了适配这些特殊的应用场景，设计了宽度因子和分辨率因子来调整网络的结构。

- 宽度因子

  宽度因子α，通过调整每一层输入通道数以及输出通道数进而调整了计算量的大小。α常用的配置是1，0.75，0.5，0.25，当α=1时就是标准的MobileNet，设置为其他小于1的数值时可以将计算量以及参数量减小到约α^2倍。

  图6是使用不同α参数调整网络结构，在ImageNet上的准确率、计算量、参数量之间的关系，可以看出，得到更小的模型一定会牺牲模型的性能。

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

图6

- 分辨率因子

  分辨率因子ρ，通过调整网络输入图片的尺寸大小，由此之后的每一层特征图尺寸都随之改变。ρ的取值范围在(0,1]之间。

  从下图可以看出，改变图像的尺寸并不会影响网络的参数量，只改变了网络的Mult-Adds计算量。

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

图7

总的来说，宽度因子和分辨率因子都可以有效的减少网络的计算量，但还是需要根据实际情况在网络的性能和计算量中做出取舍。

MobileNet V2

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	-	

图8

MobileNet V2主要的贡献就是提出了带有反向残差的线性瓶颈结构。图8是网络的结构图，图中t参数表示扩展因子，c表示输出特征图的深度，n表示这个block中反向残差结构重复堆叠的次数，s是每个block中第一个反向残差结构的步距，其他层默认为1。

线性瓶颈结构

- **什么是流形**

  我们所能观察到的数据实际上是由一个低维流形映射到高维空间上的。由于数据内部特征的限制，一些高维中的数据会产生维度上的冗余，实际上只需要比较低的维度就能唯一地表示。

举个例子，比如说我们在平面上有个圆，如何表示这个圆呢？如果我们把圆放在一个平面直角坐标系中，那一个圆实际上就是由一堆二维点构成的。

比如一个单位圆：(1, 0)是一个在圆上的点，(0,1)也是一个在圆上的点，但(0,0)和(2,3)等等很多点是不在这个圆上的。

显然如果用二维坐标来表示，我们没有办法让这个二维坐标系的所有点都是这个圆上的点。也就是说，用二维坐标来表示这个圆其实是有冗余的。

我们希望，如果能建立某一种描述方法，让这个描述方法所确定的所有点的集合都能在圆上，甚至能连续不间断地表示圆上的点，那就好了！

有没有这种方法呢？对于圆来说，当然有！那就是用极坐标。在极坐标的表示方法下，圆心在原点的圆，只需要一个参数就能确定：半径。

当你连续改变半径的大小，就能产生连续不断的"能被转换成二维坐标表示"的圆。所以说，实际上二维空间中的圆就是一个一维流形。

（原文链接：https://www.zhihu.com/question/24015486/answer/194284643）

（**流形实际上只是一个术语，用来对任意维的空间进行分类。流形假设解释了为什么机器学习技术能够从具有潜在大量维度的数据集中发现有用的特征并产生准确的预测。实际感兴趣的数据集实际上存在于低维空间中，这意味着给定的机器学习模型只需要学习关注数据集的几个关键特征就可以做出决策。**）

- **假想**

  可以把神经网络想象成具有n层结构，且每一层都有h×w×d的**激活张量**。而这h×w×d激活张量其实就是每一层的特征图，将h×w×d维的特征图理解为特征图上的每一个点具有d维。这些激活张量其实就可以看作低维流形中的一部分，也称为manifold of interest，可以理解为是有助于分类或者其他视觉任务的信息。换句话说就是，每一层中的特征图所含有的编码信息是基于流形产生的，而这些流形可以嵌入到低维子空间中。所以直觉上，为了获得更"接近"流形的数据，可以简单地降低每一层的维度从而降低整个空间的维度。在MobileNet V1中通过宽度因子来调整每一层的通道数，降低激活空间的维度，使得manifold of interest充满整个激活空间。

  但网络一般都使用ReLU作为激活函数，其函数特性使得网络只具有将非零值进行线性变换的能力，而且激活张量在通过ReLU激活后的值永远非零。

  除此之外，ReLU还会损失一部分通道的信息，从图9可以看出嵌入到高维空间的低维流形经过ReLU变换后情况，当低维流形映射到低维的空间中其信息丢失较多，流形中的某些点坍缩，当低维流形映射到高维空间中，经过ReLU后信息保留较为完整。

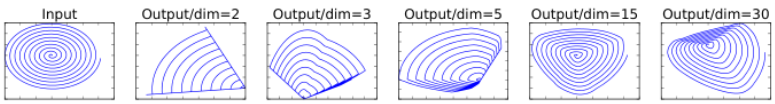

图9

  但是我们有很多的特征通道，并且通过将输入流形映射到激活空间中的一个低维子空间中，这样就可以保留特征信息。

- **本质**

  通过以上的假想，就提出了将线性瓶颈结构安插到卷积块中，以此来解决非线性激活函数造成的信息损失问题。**线性瓶颈结构本质上是不带ReLU的1x1的卷积层。**

反向残差结构

反向残差结构如图10所示：

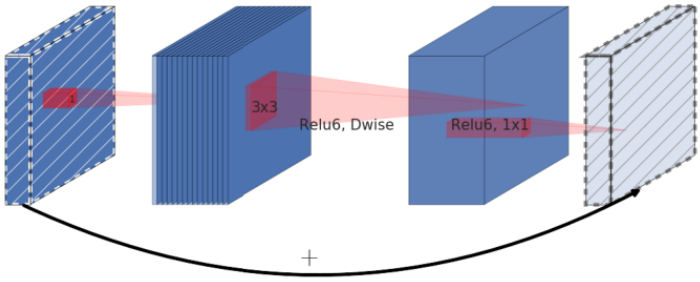

图10

图11是具体的操作：

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	$\frac{h}{s} \times \frac{w}{s} \times k'$

图11

- 先通过一个1×1的普通卷积将h×w×d大小的特征图升维得到h×w×td大小的特征图，t为扩展因子参数；
- 再通过3×3的深度方向的卷积操作，所得到的特征图大小为：

$$\frac{h}{s} \times \frac{w}{s} \times td$$

- 最后通过1×1的点卷积降维，然后使用的激活函数不是ReLU6激活函数，而是线性激活函数；

  （因为先升维再降维，导致模块两头小中间大，又由于ReLU激活函数在维度低时会损失掉一部分特征信息，所以在模块的最后使用线性激活函数来代替ReLU激活函数）

  先升维细化每块的特征，然后在降维时每个输出的通道可以融合更多的特征信息。

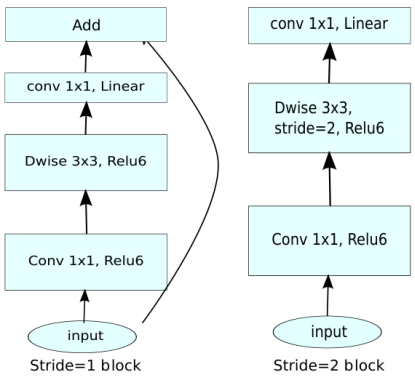

图12

- 反向残差结构中只有当步距(stride=1)，且输入与输出特征维度相同时，会有和残差结构中的shortcut连接
- 当扩展因子为1时，反向残差结构块第一层没有1×1的普通卷积

从下图可以看出通过线性瓶颈和反向残差结构优化了网络，使得网络体积更小、速度更快了。

Size	MobileNetV1	MobileNetV2
112x112	64/1600	16/400
56x56	128/800	32/200
28x28	256/400	64/100
14x14	512/200	160/62
7x7	1024/199	320/32
1x1	1024/2	1280/2
max	1600K	**400K**

图13

MobileNet V3

MobileNet V3引入了5×5大小的深度卷积代替部分3×3的深度卷积，引入SE注意力机制和h-swish激活函数提高模型的精度。

从下图可以看出，相比于MobileNet V2来说在ImageNet分类任务上的准确率提高了3.2%，在Pixel-1手机上的推理速度从64ms减少到了51ms。

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	66	2.9M	15.8	19.4	14.4
V3-Small 0.75	65.4	44	2.4M	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

图14

神经网络结构搜索技术（NAS）

MobileNet V3的网络结构是使用基于神经网络结构搜索技术学习得到的。

- **什么是NAS?**

  NAS是一种试图使用机器自动的在指定数据集的基础上通过某种算法，找到一个在此数据集上效果最好的神经网络结构和超参数的方法，可以一定程度上解决研究人员设计一个网络的耗时的问题。NAS甚至可以发现某些人类之前未曾提出的网络结构，这可以有效的降低神经网络的使用和实现成本。

- **NAS原理**

  NAS的原理是给定一个称为搜索空间的候选神经网络结构集合（即搜索空间包含了如：深度卷积、逐点卷积、常规卷积、卷积核、规范化、线性瓶颈、反向残差结构、SE模块、激活函数等等可作为原子的结构），用某种策略从中搜索出最优网络结构。神经网络结构的优劣即性能用某些指标如精度、速度来度量，称为性能评估。

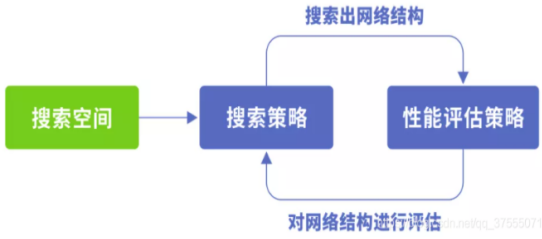

图15

搜索空间，搜索策略，性能评估策略是NAS算法的核心要素。

- 搜索空间定义了可以搜索的神经网络结构的集合，即解的空间。
- 搜索策略定义了如何在搜索空间中寻找最优网络结构。
- 性能评估策略定义了如何评估搜索出的网络结构的性能。

对这些要素的不同实现得到了各种不同的NAS算法。常见的算法有：全局搜索空间、cell-based 搜索空间、factorized hierarchical search space分层搜索空间、one-shot 架构搜索等，其中的优化算法又有：基于强化学习的算法、基于进化算法的算法、基于代理模型的算法等。

MobileNet V3使用platform-aware NAS搜索全局网络结构的优化block，也就是从搜索空间的集合中根据预定义的网络模板搜索出网络结构；之后使用NetAdapt算法针对block中的每一层搜索需要使用的卷积核个数。搜索出一种网络后需要进行性能评估。这时候需使用真实手机的CPU（MobileNet V3中使用pixel-1手机）运行TFLite Benchmark Tool进行性能评估。

（以上关于NAS部分参考博客）

做出的改进

如下图所示就是MobileNet V3网络中使用的模块：

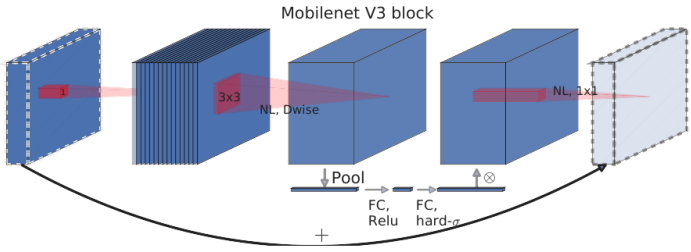

图16

- **使用了SE注意力机制**

  SE注意力机制通过精确的建模卷积特征各个通道之间的作用关系来改善网络模型的表达能力。将有效的特征信息赋予较大的权重，无效或效果小的特征信息赋予较小的权重参数。

  通过深度方向的卷积后对每个特征通道上的特征图进行平均池化操作，然后再进行全连接操作，第一层全连接层的节点个数是特征通道数的四分之一，然后第二层全连接层的节点个数又恢复到原来的通道数，最后得到每个通道的权重参数，将这个参数赋予到相应特征图上。

- **使用新的非线性激活函数**

  swish激活函数能够有效提高网络的精度，但是swish的计算量太大，且求导复杂，对量化过程不友好，并不适合后续应用到手机这类移动设备上。因此找到了h-swish函数来代替swish函数。

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

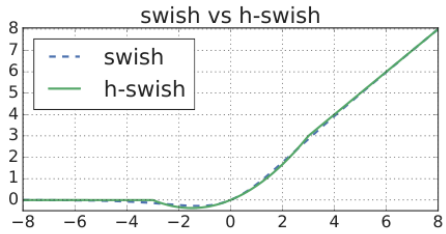

图17

图17是swish和h-swish的对比图，可以发现，h-swish很好地拟合了原swish函数的平滑性，而且很相似，所以可以进行替代。这种非线性在保持精度的情况下带来了很多优势：

- 首先，ReLU6在众多软硬件框架中都可以实现；
- 其次，量化时避免了数值精度的损失；
- 最后，运行快，这一非线性改变减少了模型的延时。

在网络进行到较深的层数时，使用非线性激活函数带来的负面影响将会减弱，因为在网络较深时，随着特征图尺寸的减小，内存占用也相应地减半。又因为swish函数应用在较深的网络层数中 可以发挥其优势，所以相应地，h-swish函数也在网络层数较深时使用：

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

图18

- **重新设计耗时层结构**
  - 减少第一层卷积核的个数

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	-	

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

图19

从原来的32减少到16，但是准确率不改变。

- 精简Last stage结构

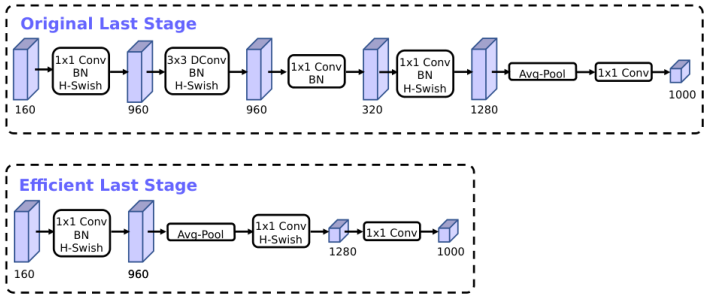

图20

使用NAS搜索算法得到的最后一个阶段的网络结构是上图的第一行，后来发现这样的结构比较耗时，所以对原结构做出了精简。精简后的结构在准确率上几乎没有什么影响，但是节省了11%的运行时间。