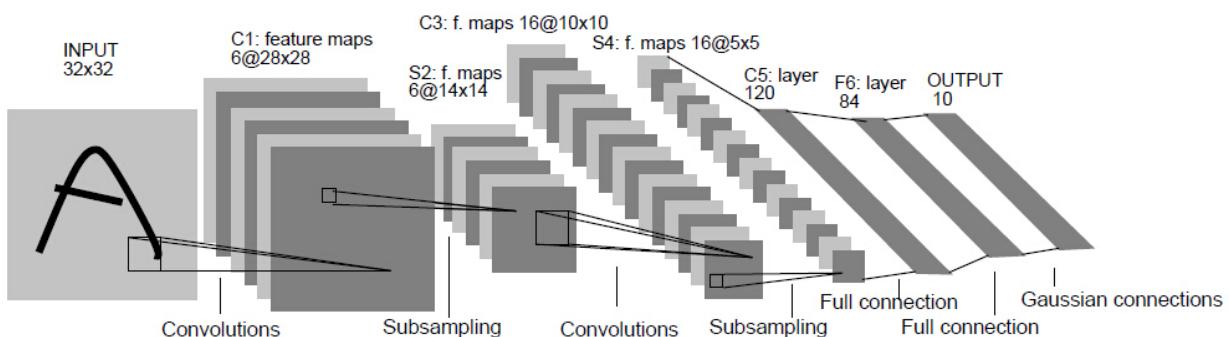


# 常见CNN：图像分类

## 一、LeNet

1. 1998年，LeCun推出了LeNet网络。它是第一个广为流传的卷积神经网络。
2. LeNet网络包含了卷积层、池化层、全连接层。这些都是现代CNN网络的基本组件。
  - 输入层：二维图像，尺寸为32x32。  
它比mnist数据集中的图片(28x28)要大。其目的是为了希望潜在的明显特征（如：笔画断续、角点）能够出现在最高层卷积核视野的中心。
  - C1、C3、C5层：二维卷积层。
    - C1层有6个卷积核，可以提取6种局部特征。核大小为5x5，输出feature map尺寸为6x28x28。
    - C3层有16个卷积核，可以提取16种局部特征。核大小为5x5，输出feature map尺寸为16x10x10。
    - C5层有120个卷积核，可以提取120种局部特征。核大小为5x5，输出feature map尺寸为120x1x1，转换为长度为120的一维向量。
  - S2、S4层：池化层。
    - S2层：窗口大小为2x2，步长为2。输入张量尺寸为6x28x28，输出feature map尺寸为6x14x14。
    - S4层：窗口大小为2x2，步长为2。输入张量尺寸为16x10x10，输出feature map尺寸为16x5x5。
  - F6层：全连接层。输出为长度为84的一维向量。  
全连接层使用sigmoid函数作为激活函数。
  - 输出层：由欧式径向基函数单元组成。输出为长度为10的一维向量。



## 二、AlexNet

1. 2012年，AlexNet横空出世。AlexNet是具有历史意义的一个网络结构，在它之前，深度学习沉寂很久。AlexNet在当年的ImageNet图像分类竞赛中，以远超第二名的成绩夺冠，使得深度学习重回历史舞台。
2. AlexNet成功的主要原因在于：
  - 使用ReLU激活函数。

- 使用 `dropout`、`data augmentation`、重叠池化等防止过拟合的方法。
- 使用百万级的大数据集来训练。
- 使用 `GPU` 训练，以及的 `LRN` 使用。
- 使用带动量的 `mini batch` 随机梯度下降来训练。

## 2.1 数据集增强

### 1. `AlexNet` 中使用的数据集增强手段：

- 随机裁剪，然后随机水平翻转。

原始图片的尺寸为 `256xx256`，裁剪大小为 `224x224`。每一个 `epoch` 中，对同一张图片进行随机性的裁剪，然后随机性的水平翻转。

理论上相当于扩充了数据集  $(256 - 224)^2 \times 2 = 2048$  倍。

在预测阶段不是随机裁剪，而是固定裁剪图片四个角、一个中心位置。再加上水平翻转，一共获得 10 张图片。用这 10 张图片的预测结果的均值作为原始图片的预测结果。

- 对 `RGB` 空间做 `PCA` 变换来完成去噪功能。同时在特征值上放大一个随机性的因子倍数（单位 `1` 加上一个  $\mathcal{N}(0, 0.1)$  的高斯扰动），从而保证图像的多样性。
  - 每一个 `epoch` 重新生成一个随机因子。
  - 该操作使得错误率下降 `1%`。

### 2. `AlexNet` 的预测方法存在两个问题：

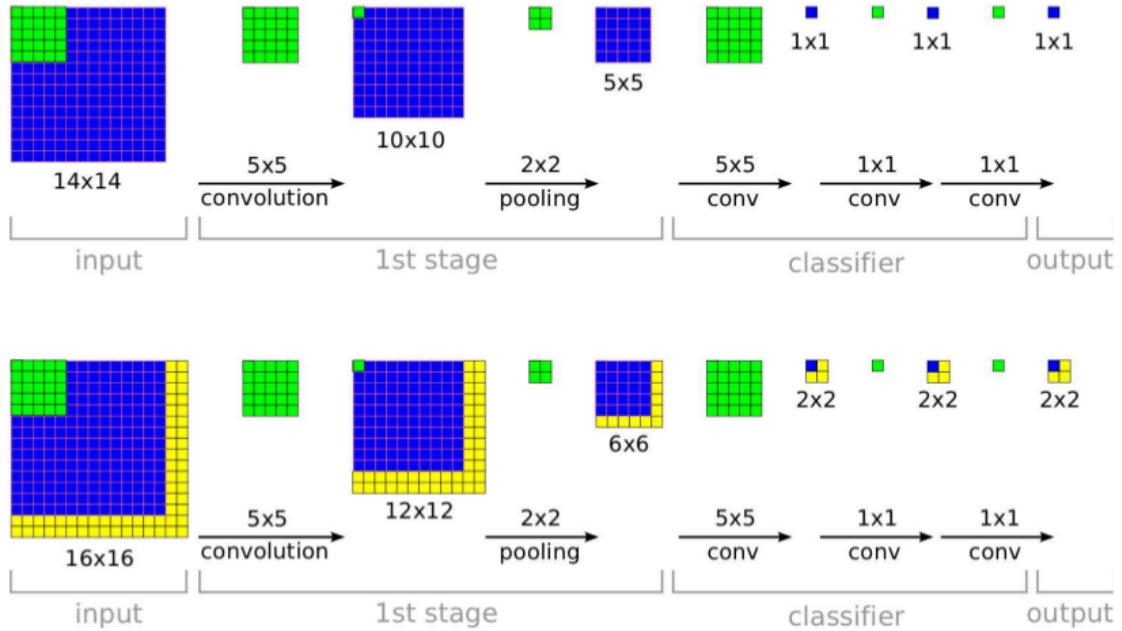
- 这种“固定裁剪四个角、一个中心”的方式，把图片的很多区域都给忽略掉了。  
很有可能一些重要的信息就被裁剪掉。
- 裁剪窗口重叠，这会引起很多冗余的计算。

### 3. 改进的思路是：

- 执行所有可能的裁剪方式，对所有裁剪后的图片进行预测。  
将所有预测结果取平均，即可得到原始测试图片的预测结果。
- 减少裁剪窗口重叠部分的冗余计算。

具体做法为：将全连接层用等效的卷积层替代，然后直接使用原始大小的测试图片进行预测。将输出的各位置处的概率值按每一类取平均，则得到原始测试图像的输出类别概率。

下图中：上半图为 `AlexNet` 的预测方法；下半图为改进的预测方法。



## 2.2 局部响应规范化

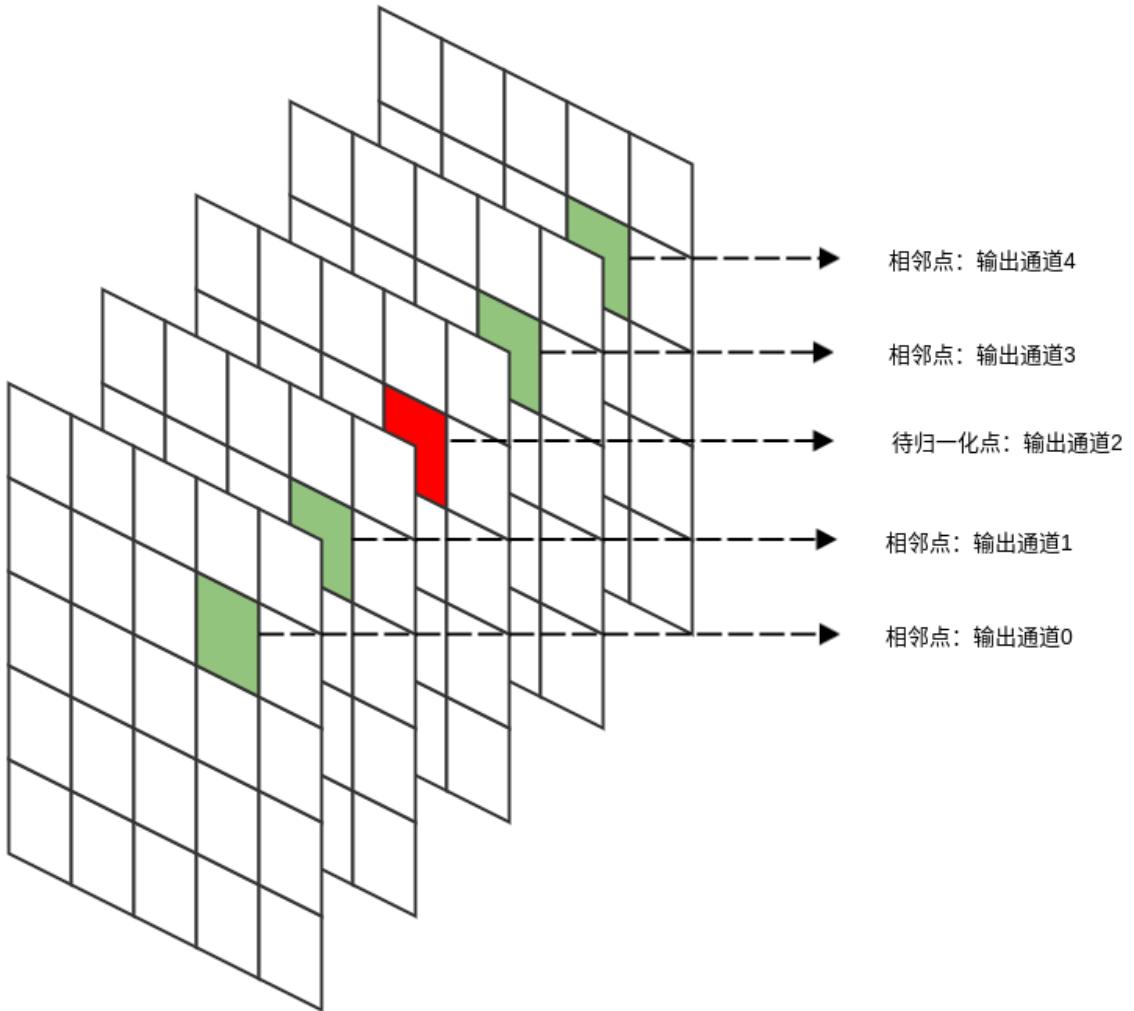
1. LRN：局部响应规范化层。目的是为了进行一个横向抑制，使得不同的卷积核所获得的响应产生竞争。
2. LRN 的思想：输出通道  $i$  在位置  $(x, y)$  处的输出会收到相邻通道在相同位置输出的影响。

为了刻画这种影响，将输出通道  $i$  的原始值除以一个归一化因子。

$$\hat{a}_i^{(x,y)} = \frac{a_i^{(x,y)}}{\left( k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a_j^{(x,y)})^2 \right)^\beta}, \quad i = 0, 1, \dots, N-1$$

其中： $a_i^{(x,y)}$  为输出通道  $i$  在位置  $(x, y)$  处的原始值， $\hat{a}_i^{(x,y)}$  为归一化之后的值。 $n$  为影响第  $i$  通道的通道数量（分别从左侧、右侧  $n/2$  个通道考虑）。 $\alpha, \beta, k$  为超参数。

一般考虑  $k = 2, n = 5, \alpha = 10^{-4}, \beta = 0.75$ 。



3. LRN 层现在很少使用，因为效果不是很明显，而且增加了内存消耗和计算时间。

4. 在 AlexNet 中，该策略贡献了 1.2% 的贡献率。

## 2.3 多GPU 训练

1. AlexNet 使用两个 GPU 训练。

网络结构图由上、下两部分组成；一个 GPU 运行图上方的层，一个 GPU 运行图下方的层。

2. 两个 GPU 只在特定的层通信。

- 第二、四、五层卷积层的核只和同一个 GPU 上的前一层的 feature map 相连。
- 第三层卷积层的核和前一层所有 GPU 的 feature map 相连。
- 全连接层中的神经元和前一层中的所有神经元相连。

## 2.4 重叠池化

1. 一般的池化是不重叠的，池化区域的大小与步长相同。

Alexnet 中，池化是可重叠的，即：步长小于池化区域的大小。重叠池化可以缓解过拟合，该策略贡献了 0.4% 的错误率。

2. 为什么重叠池化会减少过拟合，很难用数学甚至直观上的观点来解答。一个稍微合理的解释是：重叠池化会带来更多的特征，这些特征很可能有利于提高模型的预测能力。

## 2.5 优化算法

1. AlexNet 使用了带动量的 mini-batch 随机梯度下降法。

2. 标准的带动量的 mini-batch 随机梯度下降法为：

$$\begin{aligned}\vec{v} &\leftarrow \alpha \vec{v} - \epsilon \nabla_{\vec{\theta}} J(\vec{\theta}) \\ \vec{\theta} &\leftarrow \vec{\theta} + \vec{v}\end{aligned}$$

而论文中，作者使用了修正：

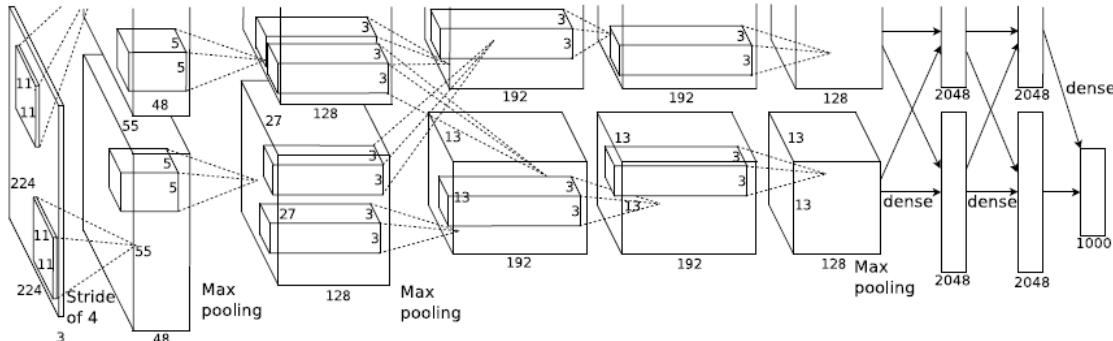
$$\begin{aligned}\vec{v} &\leftarrow \alpha \vec{v} - \beta \epsilon \vec{\theta} - \epsilon \nabla_{\vec{\theta}} J(\vec{\theta}) \\ \vec{\theta} &\leftarrow \vec{\theta} + \vec{v}\end{aligned}$$

- 其中  $\alpha = 0.9$ ,  $\beta = 0.0005$ 。 $\epsilon$  为学习率。
- $-\beta \epsilon \vec{\theta}$  为权重衰减。论文指出：权重衰减对于模型训练非常重要，不仅可以起到正则化效果，还可以减少训练误差。

## 2.6 网络结构

1. AlexNet 有5个广义卷积层和3个广义全连接层。

- 广义的卷积层：包含了卷积层、池化层、ReLU、LRN 层等。
- 广义全连接层：包含了全连接层、ReLU、Dropout 层等。



2. 输入层：224x224 的三维图片。实际上会预处理变成 227x227 的三维图片。

3. 第一层（广义卷积层）：两个 GPU 并行运算，每个 GPU 负责运行其中的一半通道。

- 卷积：采用 11x11 的卷积核，步长为 4，输出 96 个通道。生成的 feature map 尺寸为 96x55x55。
- ReLU：卷积后的数据经过 ReLU 单元处理。
- 池化：ReLU 输出的数据经过池化处理。池化层的尺寸为 3x3，步长为 2。生成的 feature map 尺寸为 96x27x27。
- LRN：池化之后的数据经过归一化处理。

4. 第二层（广义卷积层）：两个 GPU 并行运算，每个 GPU 负责运行其中的一半通道。

- 卷积：采用 5x5 的卷积核，输出 256 个通道。生成的 feature map 尺寸为 256x27x27。

为了处理方便，这里每个输入通道层的上下左右边缘都填充了2个像素的0。

- ReLU：卷积后的数据经过 ReLU 单元处理。
- 池化：ReLU 输出的数据经过池化处理。池化层的尺寸为 3x3，步长为 2。生成的 feature map 尺寸为 256x13x13。

- LRN : 池化之后的数据经过归一化处理。
5. 第三层 (广义卷积层) : 两个 GPU 并行运算, 每个 GPU 负责运行其中的一半通道。  
两个 GPU 有交叉虚线连接, 表示每个 GPU 需要处理来自前一层的所有 GPU 的输入。
- 卷积: 采用  $3 \times 3$  的卷积核, 输出 384 个通道。生成的 feature map 尺寸为  $384 \times 13 \times 13$ 。  
为了处理方便, 这里每个输入通道层的上下左右边缘都填充了1个像素的0。
  - ReLU : 卷积后的数据经过 ReLU 单元处理。
6. 第四层 (广义卷积层) : 两个 GPU 并行运算, 每个 GPU 负责运行其中的一半通道。
- 卷积: 采用  $3 \times 3$  的卷积核, 输出 384 个通道。生成的 feature map 尺寸为  $384 \times 13 \times 13$ 。  
为了处理方便, 这里每个输入通道层的上下左右边缘都填充了1个像素的0。
  - ReLU : 卷积后的数据经过 ReLU 单元处理。
7. 第五层 (广义卷积层) : 两个 GPU 并行运算, 每个 GPU 负责运行其中的一半通道。
- 卷积: 采用  $3 \times 3$  的卷积核, 输出 256 个通道。生成的 feature map 尺寸为  $256 \times 13 \times 13$ 。  
为了处理方便, 这里每个输入通道层的上下左右边缘都填充了1个像素的0。
  - ReLU : 卷积后的数据经过 ReLU 单元处理。
  - 池化: ReLU 输出的数据经过池化处理。池化层的尺寸为  $3 \times 3$ , 步长为 2。生成的 feature map 尺寸为  $256 \times 6 \times 6$ 。
8. 第六层 (广义全连接层) : 两个 GPU 并行运算, 每个 GPU 负责运行其中的一半通道。  
两个 GPU 有交叉实线连接, 表示每个 GPU 需要处理来自前一层的所有 GPU 的输入。
- 全连接层: 本层其实是一个卷积层。采用  $6 \times 6$  的卷积核, 输出 4096 个通道。生成的 feature map 尺寸为  $4096 \times 1 \times 1$ 。转换为长度为 4096 的一维向量。
  - ReLU : 经过全连接层后的数据经过 ReLU 单元处理。
  - Dropout : 经过 ReLU 层之后的数据经过 Dropout 运算。
9. 第七层 (广义全连接层) : 两个 GPU 并行运算, 每个 GPU 负责运行其中的一半通道。  
两个 GPU 有交叉实线连接, 表示每个 GPU 需要处理来自前一层的所有 GPU 的输入。
- 全连接层: 输入为长度为 4096 的一维向量, 输出为长度 4096 的一维向量。
  - ReLU : 经过全连接层后的数据经过 ReLU 单元处理。
  - Dropout : 经过 ReLU 层之后的数据经过 Dropout 运算。
10. 第八层 (广义全连接层) : 由一个 GPU 运算。
- 全连接层: 输入为长度为 4096 的一维向量, 输出为长度 1000 的一维向量, 就是输出结果。

## 三、VGG-Net

1. VGG-Net 的主要贡献是:
  - 证明了小尺寸的卷积核 ( $3 \times 3$ ) 的深层网络要优于大尺寸卷积核的浅层网络。
  - 证明了深度对网络的预测性能的重要性。
  - 验证了尺寸抖动 scale jittering 这一数据增强技术的有效性。
2. VGG-Net 最大的问题在于参数数量。  
VGG-19 基本上是参数数量最多的卷积网络架构。

## 3.1 通用配置

1. 输入：固定大小的  $224 \times 224$  的 RGB 图像。
2. 卷积层：
  - 卷积核的尺寸有两种：
    - $3 \times 3$ ：这是捕获左右、上下、中心等概念的最小尺寸。
    - $1 \times 1$ ：用于输入通道的线性变换。

在它之后接一个  $\text{ReLU}$  激活函数，使得输入通道执行了非线性变换。
  - 卷积步长：固定为 1。
  - 卷积的输入填充：卷积层的输入空间需要填充，以满足卷积之后保持同样的空间分辨率。
    - $3 \times 3$  卷积：输入的上下左右各填充 1 个像素。
    - $1 \times 1$  卷积：不需要填充。
3. 池化层：最大池化。
  - 池化层连接在卷积层之后，但并不是所有的卷积层之后都有池化。
  - 池化窗口为  $2 \times 2$ ，步长为 2。
4. 网络最后为：三个全连接层 + 一个  $\text{softmax}$  层。
  - 前两个全连接层都是 4096 个通道
  - 第三个全连接层执行 1000 类的分类，因此包含 1000 个通道。
  - 最后一层是  $\text{softmax}$  层，用于输出类别的概率。
5. 所有隐层都使用  $\text{ReLU}$  激活函数。
6. 网络不包含  $\text{LRN}$ 。

## 3.2 结构

1.  $\text{VGG-Net}$  一共有五组结构，从 A~E。
  - 每一组都遵循上述的通用配置。
  - 每一组的区别都是深度上的不同。不同的部分用黑色粗体给出。
  - 卷积层的通道数刚开始很小（64 通道），然后在每个池化层之后的卷积层通道数翻倍，直到 512。

卷积层的参数为  $\text{conv}x-y$ ，其中  $x$  为感受野大小（即卷积核大小）， $y$  为卷积核通道数。

每个卷积层之后都跟随一个  $\text{ReLU}$  激活函数，这里没有标出。

| ConvNet Configuration       |                        |                               |  |  |  |
|-----------------------------|------------------------|-------------------------------|--|--|--|
| A                           | A-LRN                  | B                             | C  | D  | E  |
| 11 weight layers            | 11 weight layers       | 13 weight layers              | 16 weight layers                           | 16 weight layers                           | 19 weight layers   |
| input (224 × 224 RGB image) |                        |                               |  |  |  |
| conv3-64                    | conv3-64<br><b>LRN</b> | conv3-64<br><b>conv3-64</b>   | conv3-64<br>conv3-64                       | conv3-64<br>conv3-64                       | conv3-64<br>conv3-64   |
| maxpool                     |                        |                               |  |  |  |
| conv3-128                   | conv3-128              | conv3-128<br><b>conv3-128</b> | conv3-128<br>conv3-128                     | conv3-128<br>conv3-128                     | conv3-128<br>conv3-128   |
| maxpool                     |                        |                               |  |  |  |
| conv3-256<br>conv3-256      | conv3-256<br>conv3-256 | conv3-256<br>conv3-256        | conv3-256<br>conv3-256<br><b>conv1-256</b> | conv3-256<br>conv3-256<br><b>conv3-256</b> | conv3-256<br>conv3-256<br><b>conv3-256</b><br><b>conv3-256</b> |
| maxpool                     |                        |                               |  |  |  |
| conv3-512<br>conv3-512      | conv3-512<br>conv3-512 | conv3-512<br>conv3-512        | conv3-512<br>conv3-512<br><b>conv1-512</b> | conv3-512<br>conv3-512<br><b>conv3-512</b> | conv3-512<br>conv3-512<br><b>conv3-512</b><br><b>conv3-512</b> |
| maxpool                     |                        |                               |  |  |  |
| conv3-512<br>conv3-512      | conv3-512<br>conv3-512 | conv3-512<br>conv3-512        | conv3-512<br>conv3-512<br><b>conv1-512</b> | conv3-512<br>conv3-512<br><b>conv3-512</b> | conv3-512<br>conv3-512<br><b>conv3-512</b><br><b>conv3-512</b> |
| maxpool                     |                        |                               |  |  |  |
| FC-4096                     |                        |                               |  |  |  |
| FC-4096                     |                        |                               |  |  |  |
| FC-1000                     |                        |                               |  |  |  |
| soft-max                    |                        |                               |  |  |  |

2. 网络的参数数量：

| 网络   | A , A-LRN | B     | C     | D     | E     |
|------|-----------|-------|-------|-------|-------|
| 参数数量 | 113百万     | 133百万 | 134百万 | 138百万 | 144百万 |

### 3.3 技巧

1. 输入预处理：通道像素零均值化。

- 先统计训练集中全部样本的：所有红色通道的像素均值  $\overline{Red}$ 、所有绿色通道的像素均值  $\overline{Green}$ 、所有蓝色通道的像素均值  $\overline{Blue}$ 。

$$\begin{aligned}\overline{Red} &= \sum_n \sum_i \sum_j I_{n,0,i,j} \\ \overline{Green} &= \sum_n \sum_i \sum_j I_{n,1,i,j} \\ \overline{Blue} &= \sum_n \sum_i \sum_j I_{n,2,i,j}\end{aligned}$$

其中：

- $n$  遍历所有的训练样本。
- $i, j$  遍历图片空间上的所有坐标。
- 假设红色通道为通道 0，绿色通道为通道 1，蓝色通道为通道 2。
- 对每个样本：
  - 红色通道的每个像素值减去  $\overline{Red}$
  - 绿色通道的每个像素值减去  $\overline{Green}$
  - 蓝色通道的每个像素值减去  $\overline{Blue}$

2. 多尺度训练：将原始的图像缩放到最小的边  $S \geq 224$ 。然后在整副图像上截取  $224 \times 224$  的区域来训练。

有两种方案：

- 在所有图像上固定  $S$ 。用  $S = 256$  来训练一个模型，用  $S = 384$  来训练另一个模型。  
最后使用两个模型来评估。
- 对每个图像，在  $[S_{\min}, S_{\max}]$  之间随机选取一个  $S$ ，然后进行裁剪来训练一个模型。最后使用单个模型来评估。
  - 该方法可以使用一个单一的模型。
  - 该方法相当于使用了尺寸抖动(`scale jittering`)的数据增强。

3. 多尺度测试：将测试的原始图像等轴的缩放到预定义的最小图像边，表示为  $Q$ ，称作测试尺度。

- $Q$  不一定等于  $S$ 。
- 在一张测试图像的几个归一化版本上运行模型（对于不同的  $Q$  值），然后对得到的结果进行平均。

4. 评估方式：

有两种方案：

- `dense`：将最后三个全连接层用等效的卷积层替代，成为一个全卷积网络。其中：第一个全连接层用  $7 \times 7$  的卷积层替代，后面两个全连接层用  $1 \times 1$  的卷积层替代。  
 用  $7 \times 7$  的原因是：经过了 5 个池化层（窗口为  $2 \times 2$ 、步长为 2），`feature map` 的尺寸为：  

$$\frac{224}{2^5} = 7$$
  - 该全卷积网络应用到整张图片上（无需裁剪），得到一个多位置的、各类别的概率字典。  
通过原始图片、水平翻转图片的各类别预测的均值，得到原始图片的各类别概率。
  - 该方法的优点是：不需要裁剪图片，支持多尺度的图片测试。
- `crop`：类似 `AlexNet` 的做法，对每个测试图像采样多个裁剪图像，平均每个裁剪图像的预测结果为原始图像的预测结果。
  - 该方法的缺点是：需要网络重新计算每个裁剪图像，效率较低
  - 该方法的优点是：使用大量的裁剪图像可以提高准确度。

5. 权重初始化：由于网络深度较深，因此网络权重的初始化很重要。不良的初始化可能会阻碍学习。

- 论文的权重初始化方案为：

- 先训练结构 A。
- 当训练更深的配置时，使用结构 A 的前四个卷积层和最后三个全连接层来初始化网络。网络的其它层被随机初始化。
- 作者后来指出：可以通过 Xavier 均匀初始化来直接初始化权重而不需要进行预训练。

### 3.3 结论

1. 实验结果表明：

- 分类误差随着网络深度的增加而减小
- 从 A-LRN 和 A 的比较发现：局部响应归一化层 LRN 对于模型没有任何改善。
- 训练时的尺寸抖动有助于提升模型的预测准确率。

Table 3: ConvNet performance at a single test scale.

| ConvNet config. (Table 1) | smallest image side |              | top-1 val. error (%) | top-5 val. error (%) |
|---------------------------|---------------------|--------------|----------------------|----------------------|
|                           | train ( $S$ )       | test ( $Q$ ) |                      |                      |
| A                         | 256                 | 256          | 29.6                 | 10.4                 |
| A-LRN                     | 256                 | 256          | 29.7                 | 10.5                 |
| B                         | 256                 | 256          | 28.7                 | 9.9                  |
| C                         | 256                 | 256          | 28.1                 | 9.4                  |
|                           | 384                 | 384          | 28.1                 | 9.3                  |
|                           | [256;512]           | 384          | 27.3                 | 8.8                  |
| D                         | 256                 | 256          | 27.0                 | 8.8                  |
|                           | 384                 | 384          | 26.8                 | 8.7                  |
|                           | [256;512]           | 384          | 25.6                 | 8.1                  |
| E                         | 256                 | 256          | 27.3                 | 9.0                  |
|                           | 384                 | 384          | 26.9                 | 8.7                  |
|                           | [256;512]           | 384          | <b>25.5</b>          | <b>8.0</b>           |

2. 实验结果表明：测试时的尺寸抖动导致了更好的性能。

Table 4: ConvNet performance at multiple test scales.

| ConvNet config. (Table 1) | smallest image side |              | top-1 val. error (%) | top-5 val. error (%) |
|---------------------------|---------------------|--------------|----------------------|----------------------|
|                           | train ( $S$ )       | test ( $Q$ ) |                      |                      |
| B                         | 256                 | 224,256,288  | 28.2                 | 9.6                  |
|                           | 256                 | 224,256,288  | 27.7                 | 9.2                  |
| C                         | 384                 | 352,384,416  | 27.8                 | 9.2                  |
|                           | [256; 512]          | 256,384,512  | 26.3                 | 8.2                  |
| D                         | 256                 | 224,256,288  | 26.6                 | 8.6                  |
|                           | 384                 | 352,384,416  | 26.5                 | 8.6                  |
|                           | [256; 512]          | 256,384,512  | <b>24.8</b>          | <b>7.5</b>           |
| E                         | 256                 | 224,256,288  | 26.9                 | 8.7                  |
|                           | 384                 | 352,384,416  | 26.7                 | 8.6                  |
|                           | [256; 512]          | 256,384,512  | <b>24.8</b>          | <b>7.5</b>           |

3. 实验结果表明：

- crop 评估方式要比 dense 评估方式表现更好。
- 二者是互补的。其组合要优于任何单独的一种。

Table 5: **ConvNet evaluation techniques comparison.** In all experiments the training scale  $S$  was sampled from [256; 512], and three test scales  $Q$  were considered: {256, 384, 512}.

| ConvNet config. (Table I) | Evaluation method  | top-1 val. error (%) | top-5 val. error (%) |
|---------------------------|--------------------|----------------------|----------------------|
| D                         | dense              | 24.8                 | 7.5                  |
|                           | multi-crop         | 24.6                 | 7.5                  |
|                           | multi-crop & dense | <b>24.4</b>          | <b>7.2</b>           |
| E                         | dense              | 24.8                 | 7.5                  |
|                           | multi-crop         | 24.6                 | 7.4                  |
|                           | multi-crop & dense | <b>24.4</b>          | <b>7.1</b>           |

## 四、Inception

1. Inception 网络是卷积神经网络的一个重要里程碑。

- 在 Inception 之前，大部分流行的卷积神经网络仅仅是把卷积层堆叠得越来越多，使得网络越来越深。
- 而 Inception 网络考虑的是多种卷积核的并行计算，扩展了网络的宽度。

这使得网络越来越复杂，参数越来越多，从而导致网络容易出现过拟合，增加计算量。

而 Inception 网络考虑的是卷积层的并行，使得网络的宽度较大。

2. Inception Net 核心思想是：稀疏连接。因为生物神经连接是稀疏的。

3. Inception 网络的最大特点是大量使用了 Inception 模块。

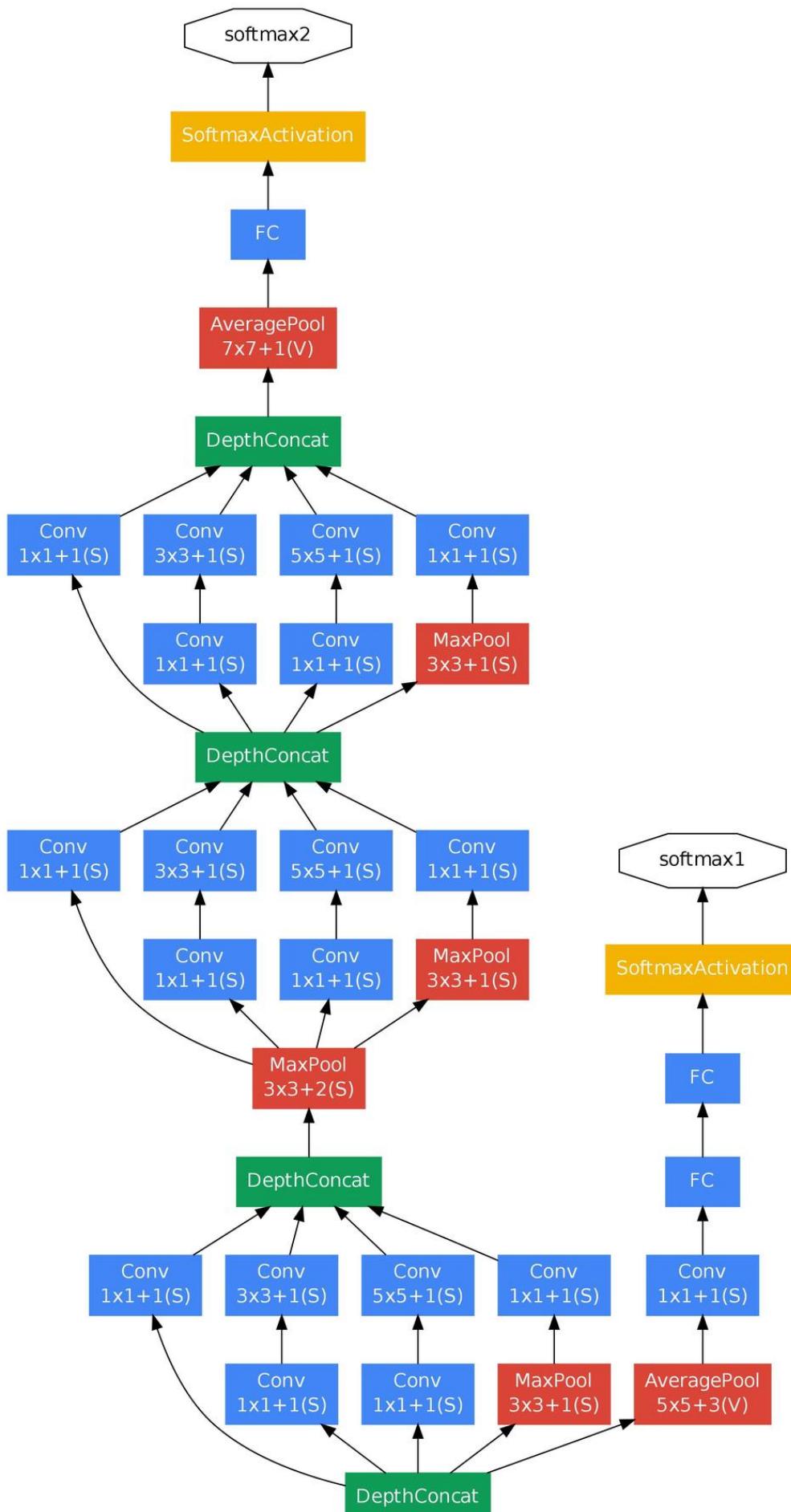
4. 图像中目标对象的大小可能差别很大。如下图所示，每张图像中，狗占据区域都是不同的。

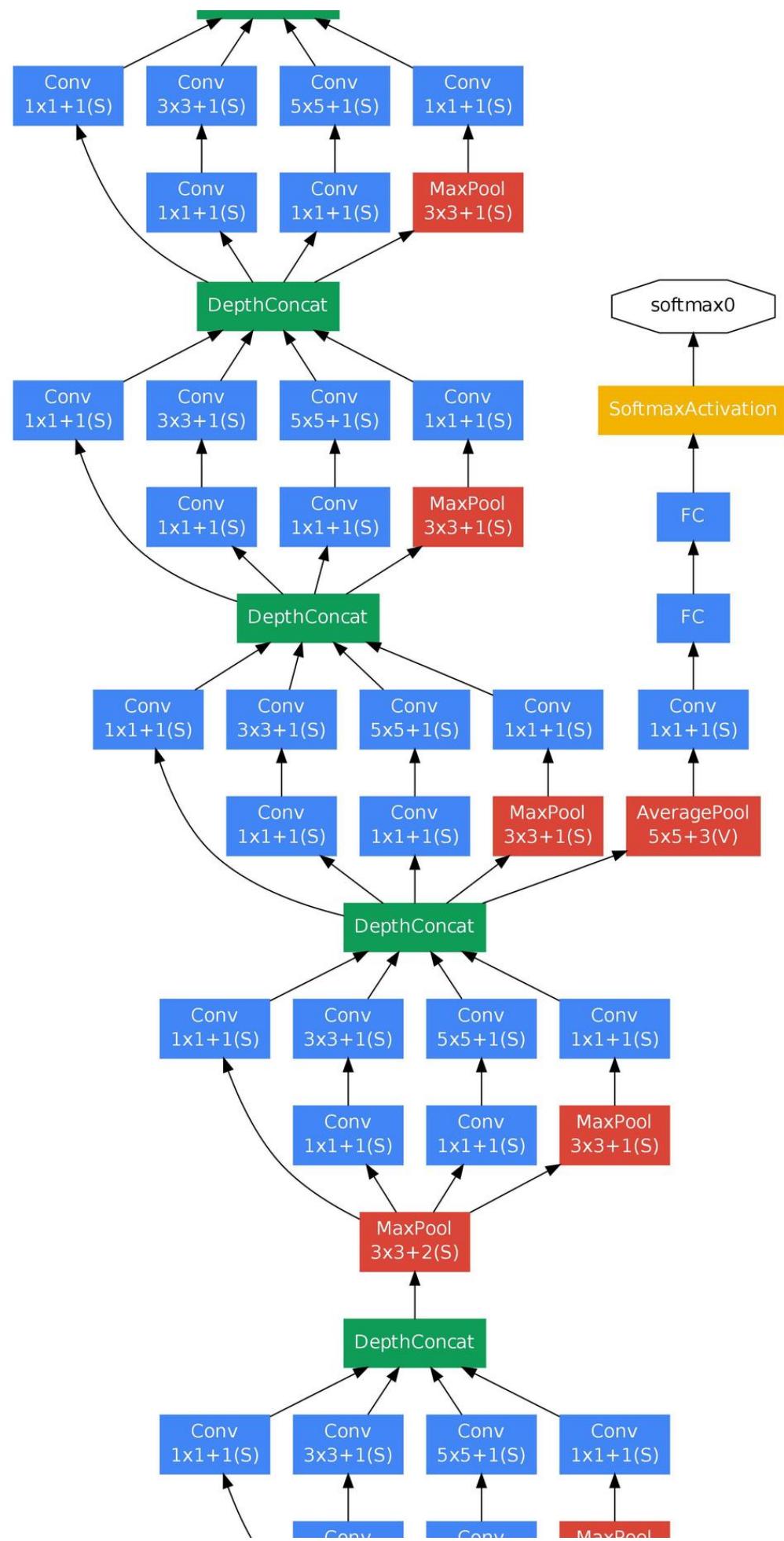
由于信息区域的巨大差异，为卷积操作选择合适的卷积核尺寸就非常困难。

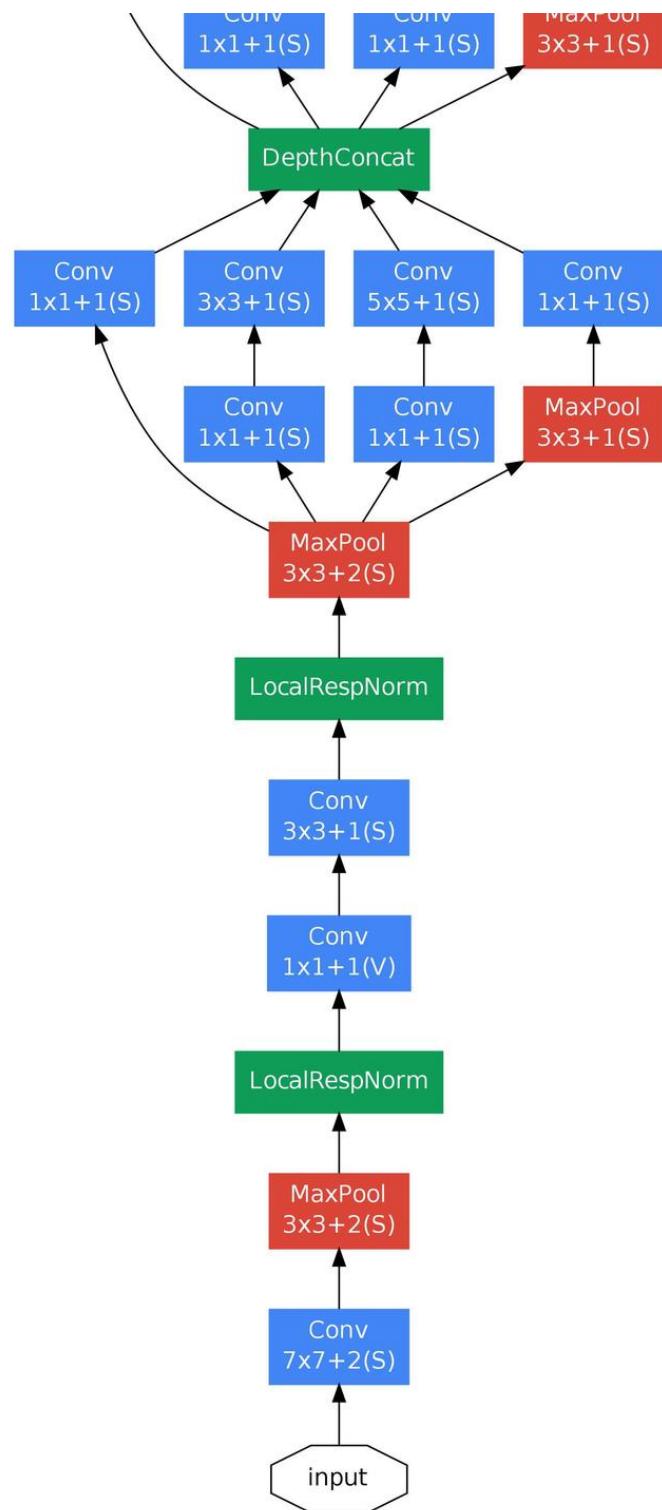
- 信息分布更具有全局性的图像中，更倾向于使用较大的卷积核。如最左侧的图片所示。
- 信息分布更具有局部性的图像中，更倾向于使用较小的卷积核。如最右侧的图片所示。



### 4.1 Inception v1







#### 4.1.1 Inception 模块

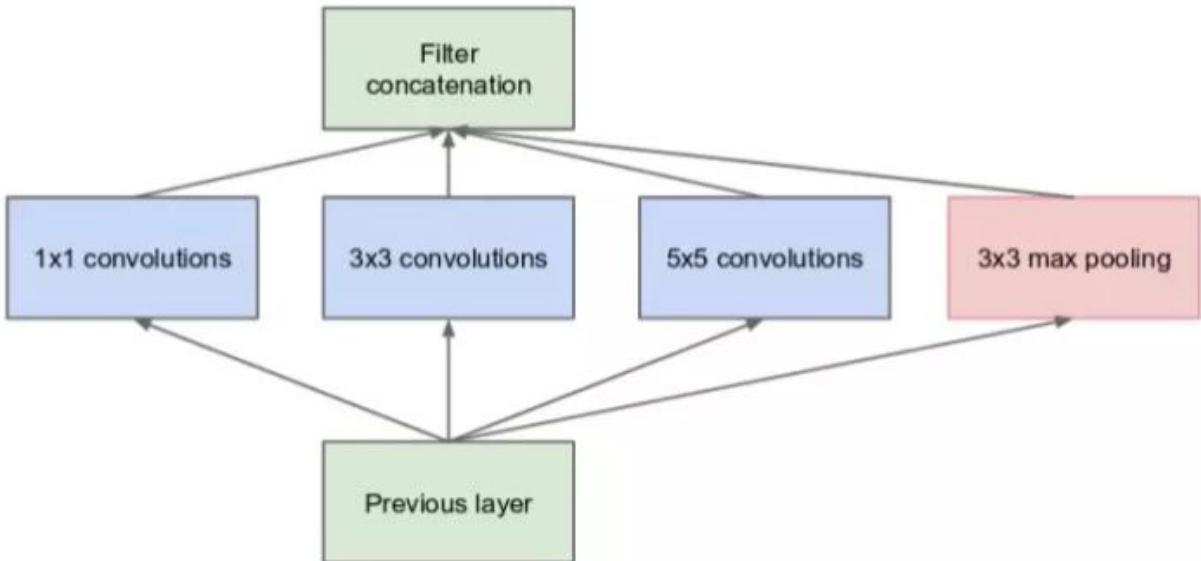
1. 原始的 **Inception** 模块对输入同时执行：3个不同大小的卷积操作（ $1 \times 1$ 、 $3 \times 3$ 、 $5 \times 5$ ）、1个最大池化操作（ $3 \times 3$ ）。

所有操作的输出都在深度方向拼接起来，向后一级传递。

- 采用这三种不同大小卷积操作的原因：希望通过不同尺寸的卷积核抓取不同尺度的特征。

而使用 `1x1`、`3x3`、`5x5` 这些具体尺寸仅仅是为了便利性。事实上也可以使用更多的、其它尺寸的滤波器。

- 采用1个最大池化操作的原因：考虑到池化操作对于当前的卷积网络的成功至关重要，所以增加一个池化路径可能具有额外的、有益的收益。
- 3个不同尺寸的滤波器可以抓取不同大小的对象的特征。
- 池化层提取的是图像的原始特征（不经过过滤器）



(a) Inception module, naïve version

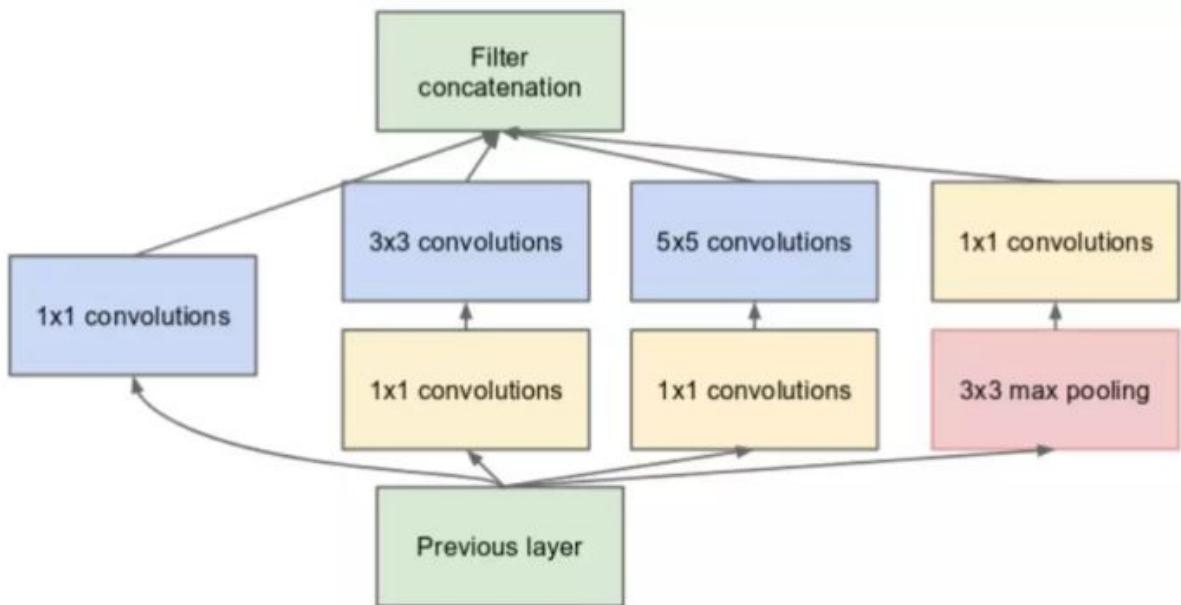
2. 原始 `Inception` 模块中，模块的输出通道数量为四个子层的输出通道数的叠加。

这种叠加不可避免的使得 `Inception` 模块的输出通道数（相对于输入通道数）增加。

如果每个 `Inception` 模块的输出通道数增加，则在经过若干个模块之后，计算量会爆炸性增长。

解决方案是：在 `3x3` 和 `5x5` 卷积层之前额外添加 `1x1` 卷积层，来限制输入给卷积层的输入通道的数量。

注意：`1x1` 卷积是在最大池化层之后，而不是之前。



(b) Inception module with dimension reductions

#### 4.1.2 辅助分类器

1. 为了缓解梯度消失的问题, InceptionNet V1 给出了两个辅助分类器。

这两个辅助分类器的添加到网络的中间层, 它们和主分类器共享同一套训练数据及其标记。其中:

- 第一个辅助分类器位于 Inception(4a) 之后。Inception(4a) 模块的输出作为它的输入。
- 第二个辅助分类器位于 Inception(4d) 之后。Inception(4d) 模块的输出作为它的输入。
- 两个辅助分类器的结构相同, 包括以下组件:
  - 一个尺寸为  $5 \times 5$ 、步长为3的平均池化层。
  - 一个尺寸为  $1 \times 1$ 、输出通道数为 128 的卷积层。
  - 一个具有 1024 个单元的全连接层。
  - 一个丢弃70%输出的 dropout 层。
  - 一个使用 softmax 损失的线性层作为输出层。

2. 在训练期间, 两个辅助分类器的损失函数的权重是0.3, 它们的损失被叠加到网络的整体损失上。

- 在推断期间, 这两个辅助网络被丢弃。
- 在 Inception v3 的实验中表明: 辅助网络的影响相对较小, 只需要其中一个就能够取得同样的效果。

事实上辅助分类器在训练早期并没有多少贡献。只有在训练接近结束, 辅助分支网络开始发挥作用, 获得超出无辅助分类器网络的结果。

3. 两个辅助分类器的作用: 提供正则化的同时, 克服了梯度消失问题。

#### 4.1.3 网络参数

1. InceptionNet V1 是一个22层的深度网络。如果考虑池化层, 则有29层。如下图中的 depth 列所示。

2. 网络具有三组 Inception 模块, 分别为: inception(3a)/inception(3b)、

inception(4a)/inception(4b)/inception(4c)/inception(4d)/inception(4e)、inception(5a)、inception(5b)。

三组 Inception 模块被池化层分隔。

3. 下图给出了网络的层次结构和参数，其中：

- `type` 列：给出了每个模块/层的类型。
- `patch size/stride` 列：给出了卷积层/池化层的尺寸和步长
- `output size` 列：给出了每个模块/层的输出尺寸和输出通道数
- `depth` 列：给出了每个模块/层包含的、含有训练参数层的数量
- `#1x1` 列：给出了每个模块/层包含的 `1x1` 卷积核的数量（它就是 `1x1` 卷积核的输出通道数）
- `#3x3 reduce` 列：给出了每个模块/层包含的、放置在 `3x3` 卷积层之前的 `1x1` 卷积核的数量（它就是 `1x1` 卷积核的输出通道数）
- `#3x3` 列：给出了每个模块/层包含的 `3x3` 卷积核的数量（它就是 `3x3` 卷积核的输出通道数）
- `#5x5 reduce` 列：给出了每个模块/层包含的、放置在 `5x5` 卷积层之前的 `1x1` 卷积核的数量（它就是 `1x1` 卷积核的输出通道数）
- `#5x5` 列：给出了每个模块/层包含的 `5x5` 卷积核的数量（它就是 `5x5` 卷积核的输出通道数）
- `pool proj` 列：给出了每个模块/层包含的、放置在池化层之后的 `1x1` 卷积核的数量（它就是 `1x1` 卷积核的输出通道数）
- `params` 列：给出了每个模块/层的参数数量 (?)
- `ops` 列：给出了每个模块/层的操作数量。

| <code>type</code> | <code>patch size/stride</code> | <code>output size</code>   | <code>depth</code> | <code>#1x1</code> | <code>#3x3 reduce</code> | <code>#3x3</code> | <code>#5x5 reduce</code> | <code>#5x5</code> | <code>pool proj</code> | <code>params</code> | <code>ops</code> |
|-------------------|--------------------------------|----------------------------|--------------------|-------------------|--------------------------|-------------------|--------------------------|-------------------|------------------------|---------------------|------------------|
| convolution       | $7 \times 7 / 2$               | $112 \times 112 \times 64$ | 1                  |                   |                          |                   |                          |                   |                        | 2.7K                | 34M              |
| max pool          | $3 \times 3 / 2$               | $56 \times 56 \times 64$   | 0                  |                   |                          |                   |                          |                   |                        |                     |                  |
| convolution       | $3 \times 3 / 1$               | $56 \times 56 \times 192$  | 2                  |                   | 64                       | 192               |                          |                   |                        | 112K                | 360M             |
| max pool          | $3 \times 3 / 2$               | $28 \times 28 \times 192$  | 0                  |                   |                          |                   |                          |                   |                        |                     |                  |
| inception (3a)    |                                | $28 \times 28 \times 256$  | 2                  | 64                | 96                       | 128               | 16                       | 32                | 32                     | 159K                | 128M             |
| inception (3b)    |                                | $28 \times 28 \times 480$  | 2                  | 128               | 128                      | 192               | 32                       | 96                | 64                     | 380K                | 304M             |
| max pool          | $3 \times 3 / 2$               | $14 \times 14 \times 480$  | 0                  |                   |                          |                   |                          |                   |                        |                     |                  |
| inception (4a)    |                                | $14 \times 14 \times 512$  | 2                  | 192               | 96                       | 208               | 16                       | 48                | 64                     | 364K                | 73M              |
| inception (4b)    |                                | $14 \times 14 \times 512$  | 2                  | 160               | 112                      | 224               | 24                       | 64                | 64                     | 437K                | 88M              |
| inception (4c)    |                                | $14 \times 14 \times 512$  | 2                  | 128               | 128                      | 256               | 24                       | 64                | 64                     | 463K                | 100M             |
| inception (4d)    |                                | $14 \times 14 \times 528$  | 2                  | 112               | 144                      | 288               | 32                       | 64                | 64                     | 580K                | 119M             |
| inception (4e)    |                                | $14 \times 14 \times 832$  | 2                  | 256               | 160                      | 320               | 32                       | 128               | 128                    | 840K                | 170M             |
| max pool          | $3 \times 3 / 2$               | $7 \times 7 \times 832$    | 0                  |                   |                          |                   |                          |                   |                        |                     |                  |
| inception (5a)    |                                | $7 \times 7 \times 832$    | 2                  | 256               | 160                      | 320               | 32                       | 128               | 128                    | 1072K               | 54M              |
| inception (5b)    |                                | $7 \times 7 \times 1024$   | 2                  | 384               | 192                      | 384               | 48                       | 128               | 128                    | 1388K               | 71M              |
| avg pool          | $7 \times 7 / 1$               | $1 \times 1 \times 1024$   | 0                  |                   |                          |                   |                          |                   |                        |                     |                  |
| dropout (40%)     |                                | $1 \times 1 \times 1024$   | 0                  |                   |                          |                   |                          |                   |                        |                     |                  |
| linear            |                                | $1 \times 1 \times 1000$   | 1                  |                   |                          |                   |                          |                   |                        | 1000K               | 1M               |
| softmax           |                                | $1 \times 1 \times 1000$   | 0                  |                   |                          |                   |                          |                   |                        |                     |                  |

Table 1: GoogLeNet incarnation of the Inception architecture.

## 4.2 Inception v2

1. Inception v2 的主要贡献是：提出了 Batch Normalization。

2. 论文指出，使用了 Batch Normalization 之后：

- 可以加速网络的学习。相比 Inception v1，训练速度提升了14倍。

应用了 BN 之后，：

- 可以使用更高的学习率
- 可以删除 dropout
- 可以加速学习率衰减
- 可以删除 LRN 层
- 可以不用太在意初始化
- 更好的预测能力。

在 ImageNet 分类问题的 top5 上达到 4.8%，超过了人类标注 top5 准确率。

## 4.3 Inception v3

1. Inception v3 重点探讨了网络结构设计的原则。
2. 虽然 Inception v1 的参数较少，但是它的结构比较复杂，难以进行修改。原因有以下两点：
  - 如果单纯的放大网络（如增加 Inception 模块的数量、扩展 Inception 模块的大小），则参数的数量会显著增长，其计算成本的好处会丧失。
    - | AlexNet 大约使用了6千万参数，VGGNet 使用了超过1亿参数。而 Inception v1 大约使用了500万参数。
    - Inception v1 结构中的各种设计，其对最终结果的贡献尚未明确。

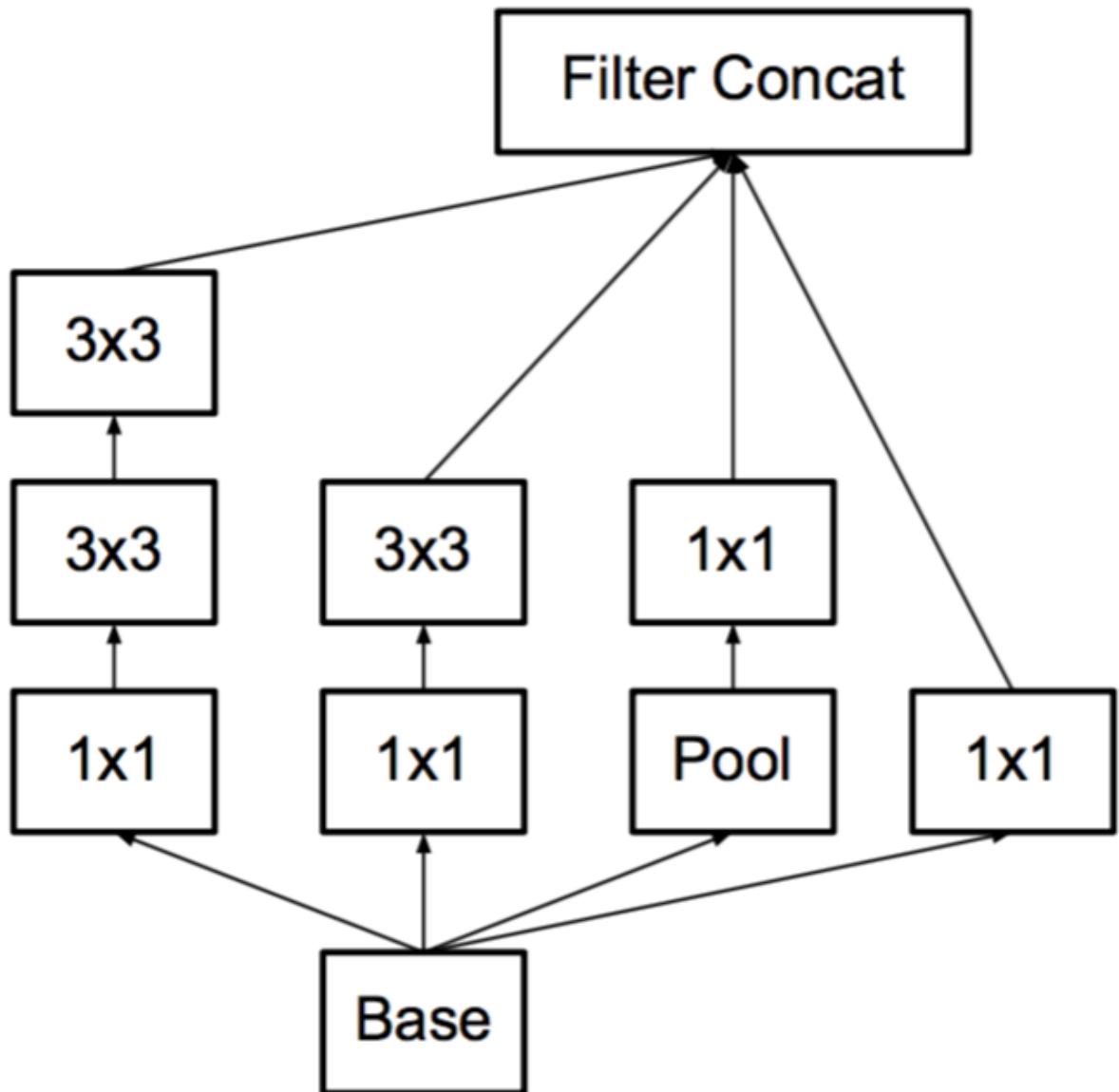
### 4.3.1 设计原则

1. 通用设计原则：
  - 避免表征瓶颈：表征的大小应该从输入到输出缓缓减小，避免极端压缩。在缩小 feature map 尺寸的同时，增加 feature map 的通道数。
    - | 表征大小通常指的是 feature map 的大小（包括尺寸、通道数）。
  - 空间聚合：可以通过空间聚合来完成低维嵌入，而不会在表达能力上有较大的损失。
    - | 因此通常在  $n \times n$  卷积之前，先利用  $1 \times 1$  卷积来降低输入维度。
    - | 猜测的原因是：相邻空间维度之间的强相关性导致了空间聚合过程中的信息丢失较少。
  - 平衡网络的宽度和深度：增加网络的宽度或者深度都可以提高网络的质量，因此计算资源需要在网络的深度和宽度之间取得平衡。

#### 4.3.1.1 卷积尺寸分解

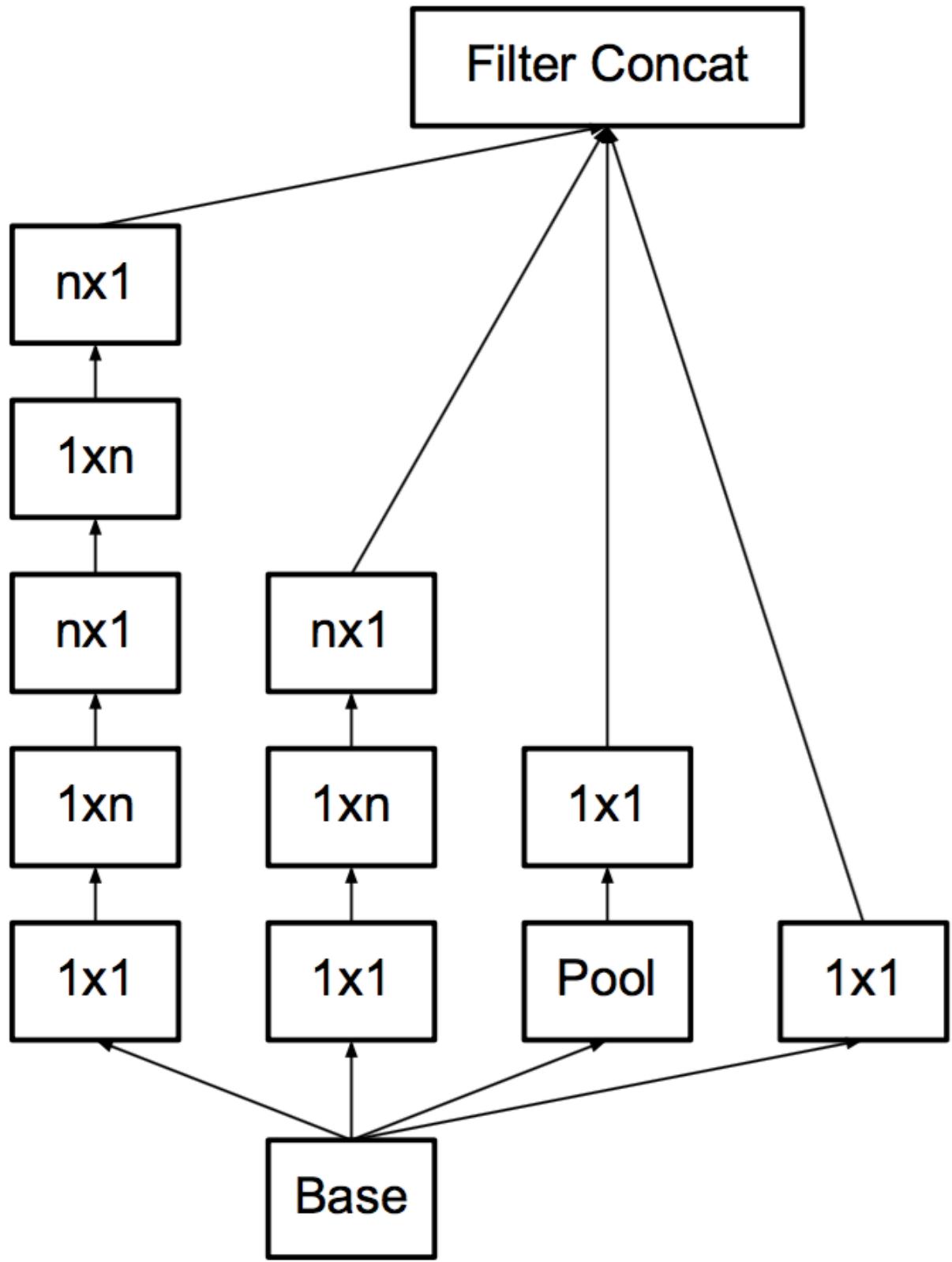
1. 大卷积核的分解：将大卷积核分解为多个小的卷积核。

如：使用2个  $3 \times 3$  卷积替换每个  $5 \times 5$  卷积。



2.  $n \times n$  卷积核的非对称分解：将  $n \times n$  卷积替换为  $1 \times n$  卷积和  $n \times 1$  卷积。

- 随着  $n$  的增加，计算成本的节省非常显著。
- 论文指出：对于较大的特征图，这种分解不能很好的工作；但是对于中等大小的特征图（尺寸在  $12 \sim 20$  之间），这种分解效果非常好。



#### 4.3.1.2 网格尺寸缩减

1. 假设输入的 feature map 尺寸为  $d \times d$ ，通道数为  $k$ 。如果希望输出的 feature map 尺寸为  $d/2 \times d/2$ ，通道数为  $2k$ 。则有以下的两种方式：

- 首先使用  $2k$  个  $1 \times 1$  的卷积核，执行步长为1的卷积。然后执行一个  $2 \times 2$  的、步长为2的池化操作。

该方式需要执行  $2d^2 k^2$  次卷积操作，计算代价较大。

- 直接使用  $2k$  个  $1 \times 1$  的卷积核，执行步长为2的卷积。

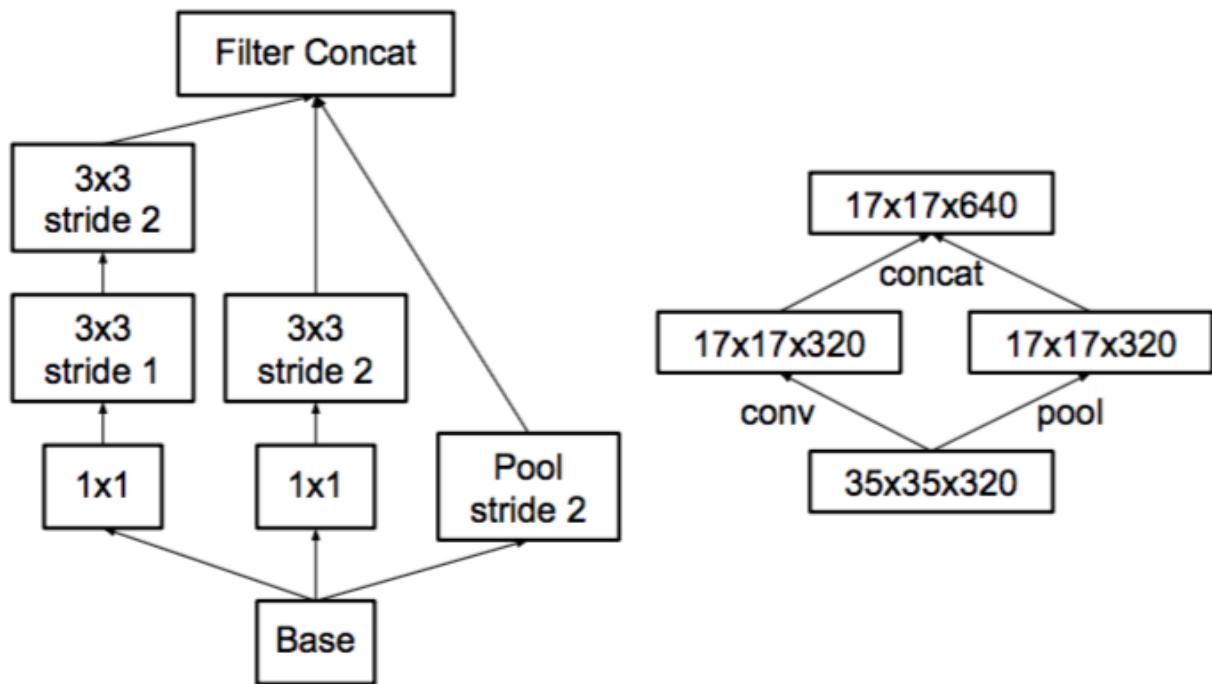
该方式需要执行  $2(\frac{d}{2})^2 k^2$  次卷积操作，计算代价相对较小。但是表征能力下降，产生了表征瓶颈。

2. 解决方案是：采用两个模块  $P$  和  $C$ ：

- 模块  $P$ ：使用  $k$  个  $1 \times 1$  的卷积核，执行步长为2的卷积。其输出  $feature map$  尺寸为  $d/2 \times d/2$ ，通道数为  $k$ 。
- 模块  $C$ ：使用步长为2的池化。其输出  $feature map$  尺寸为  $d/2 \times d/2$ ，通道数为  $k$ 。

将模块  $P$  和模块  $C$  的输出按照通道数拼接，产生最终的输出  $feature map$ 。

下图中：左图为采用了网格尺寸缩减的 Inception 模块；右图为  $feature map$  示意图。



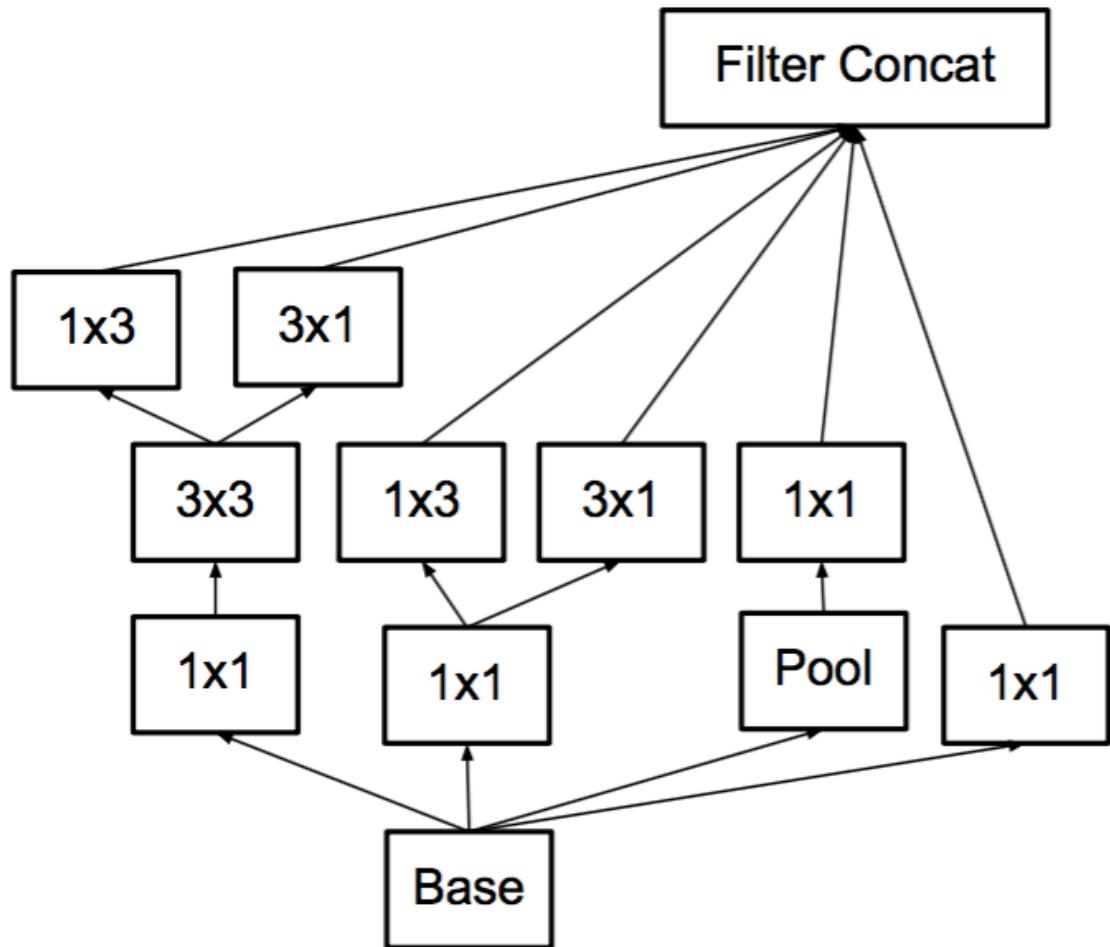
#### 4.3.2 网络参数

1. Inception v3 的网络深度为42层。

2. Inception v3 网络相对于 Inception v1 网络主要做了以下改动：

- $7 \times 7$  卷积替换为3个  $3 \times 3$  卷积
- 3个 Inception 模块：模块中的  $5 \times 5$  卷积替换为2个  $3 \times 3$  卷积，同时使用前面描述的网格尺寸缩减技术
- 5个 Inception 模块：模块中的  $5 \times 5$  卷积替换为2个  $3 \times 3$  卷积之后，所有的  $n \times n$  卷积进行非对称分解，同时使用前面描述的网格尺寸缩减技术
- 2个 Inception 模块：结构如下。它也使用了网格尺寸缩减技术，以及卷积分解技术。

修改成该结构的原因：提升高级表征。

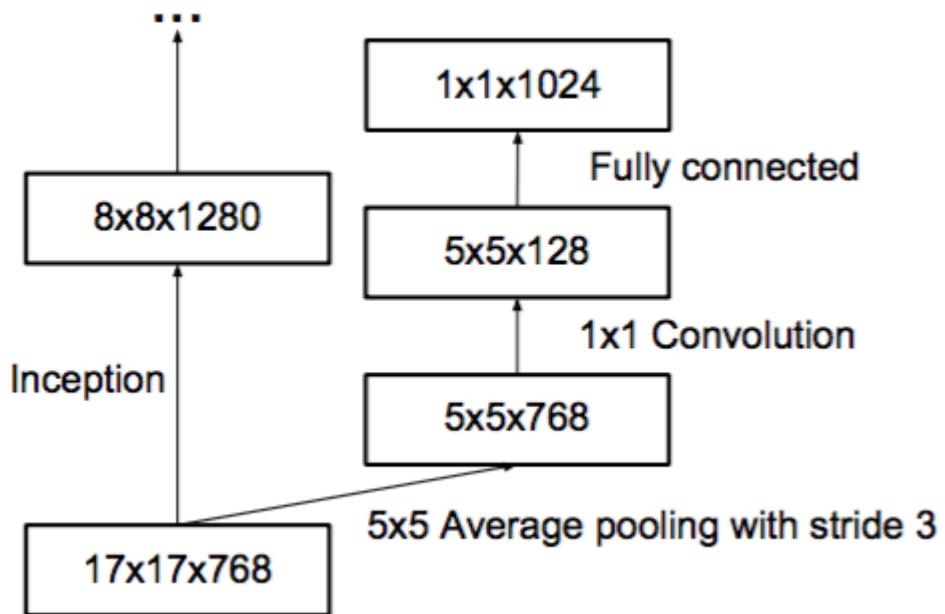


3. 网络结构如下所示，其中：

- type 列：给出了每个模块/层的类型。
  - 其中 `3xInception` 表示三个 `Inception` 模块； `4xInception` 表示四个 `Inception` 模块； `5xInception` 表示五个 `Inception` 模块
  - `conv_padded` 表示使用0填充的卷积，它可以保持 `feature map` 的尺寸。  
在 `Inception` 模块内的卷积也使用0填充。  
所有其它的卷积/池化不再使用填充。
- patch size/stride 列：给出了卷积层/池化层的尺寸和步长
- input size 列：给出了每个模块/层的输入尺寸和输入通道数

| <b>type</b>          | <b>patch size/stride<br/>or remarks</b> | <b>input size</b>          |
|----------------------|---|----------------------------|
| conv                 | $3 \times 3 / 2$                        | $299 \times 299 \times 3$  |
| conv                 | $3 \times 3 / 1$                        | $149 \times 149 \times 32$ |
| conv padded          | $3 \times 3 / 1$                        | $147 \times 147 \times 32$ |
| pool                 | $3 \times 3 / 2$                        | $147 \times 147 \times 64$ |
| conv                 | $3 \times 3 / 1$                        | $73 \times 73 \times 64$   |
| conv                 | $3 \times 3 / 2$                        | $71 \times 71 \times 80$   |
| conv                 | $3 \times 3 / 1$                        | $35 \times 35 \times 192$  |
| $3 \times$ Inception | As in figure 5                          | $35 \times 35 \times 288$  |
| $5 \times$ Inception | As in figure 6                          | $17 \times 17 \times 768$  |
| $2 \times$ Inception | As in figure 7                          | $8 \times 8 \times 1280$   |
| pool                 | $8 \times 8$                            | $8 \times 8 \times 2048$   |
| linear               | logits                                  | $1 \times 1 \times 2048$   |
| softmax              | classifier                              | $1 \times 1 \times 1000$   |

4. 在 `3xInception` 模块的输出之后设有一个辅助分类器。其结构如下：



#### 4.3.3 标签平滑正则化

1. 假设样本  $\vec{x}$  的类别为  $y$ 。

- 常规训练中，该样本的类别分布为一个  $\delta$  函数： $P(Y = k \mid X = \vec{x}) = \delta(y - k)$ ,  $k = 1, 2, \dots, K$ 。记做  $\delta_{k,y}$ 。
- 使用标签平滑正则化 (LSR: Label Smoothing Regularization) 之后，该样本的类别分布为：

$$P(Y = k \mid X = \vec{x}) = (1 - \epsilon)\delta_{k,y} + \frac{\epsilon}{K}, \quad k = 1, 2, \dots, K$$

它表示：

- 样本  $\vec{x}$  的类别为  $y$  的概率为  $1 - \frac{(K-1)\epsilon}{K}$
- 样本  $\vec{x}$  的类别为  $1, 2, \dots, y-1, y+1, \dots, K$  的概率均  $\frac{\epsilon}{K}$

2. 论文指出：标签平滑正则化对 top-1 错误率和 top-5 错误率提升了大约 0.2%。

3. 标签平滑正则化有效的原因：对样本的真实标记不是那么自信。即：假设样本的真实标记存在一定程度上的噪声。

## 4.4 Inception v4 & Inception - ResNet

1. Inception v4 和 Inception-ResNet 在同一篇论文中给出。

其主要贡献在于：通过实验证明了结合残差连接可以显著加速 Inception 的训练。

### 4.4.1 Inception v4

1. 在 Inception v4 结构的主要改动：

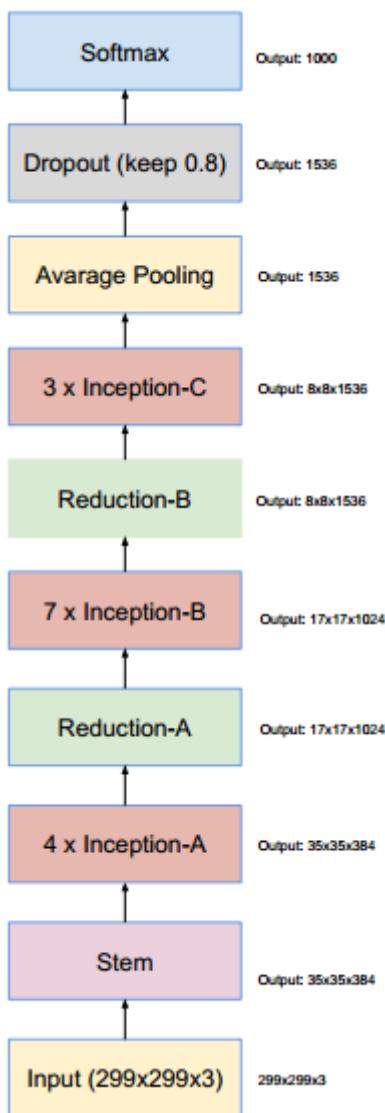
- 修改了 stem 部分
- 引入了 Inception-A、Inception-B、Inception-C 三个模块。

这些模块看起来和 Inception v3 变体非常相似。

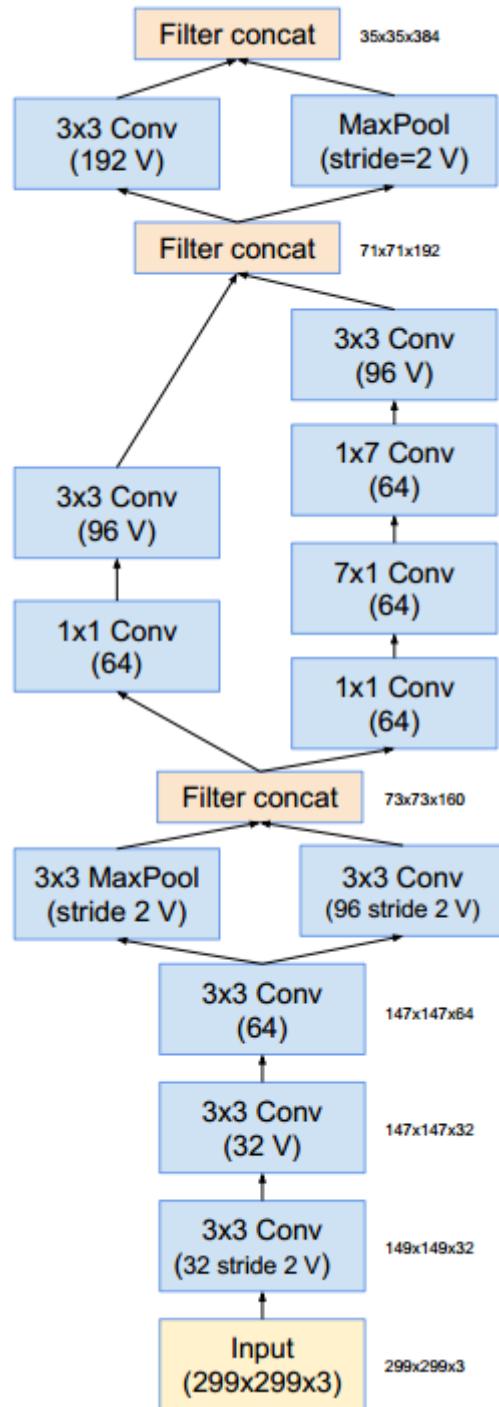
- 引入了专用的“缩减块”( reduction block )，它被用于改变网格的宽度和高度。

早期的版本并没有明确使用缩减块，但是也实现了其功能。

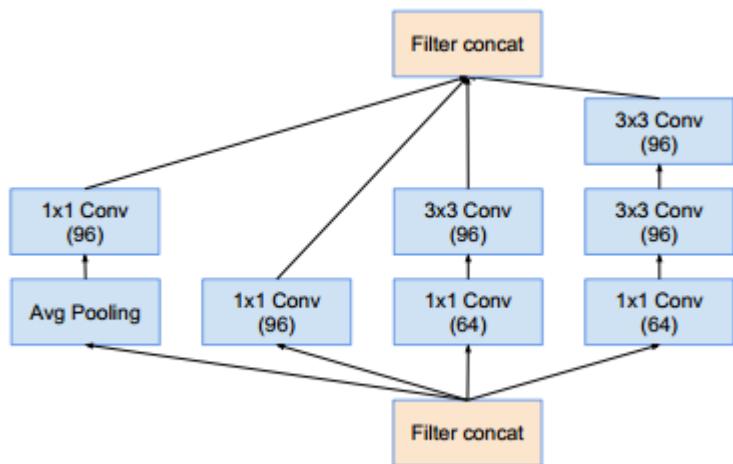
2. Inception v4 结构如下：(没有标记 V 的卷积使用 same 填充；标记 V 的卷积使用 valid 填充)



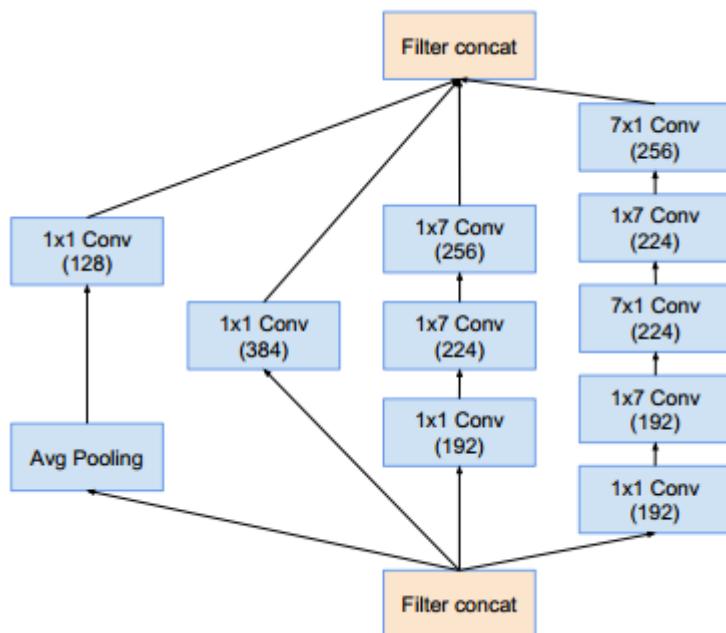
- `stem` 部分的结构：



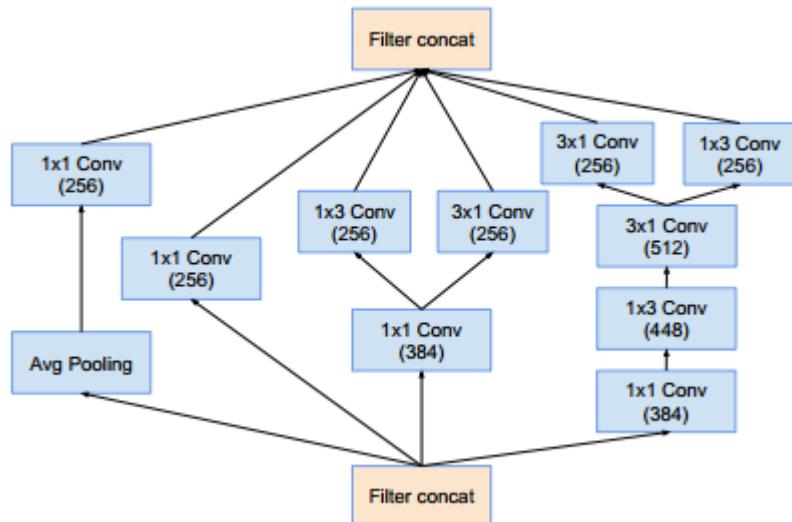
- Inception-A 模块（这样的模块有4个）：



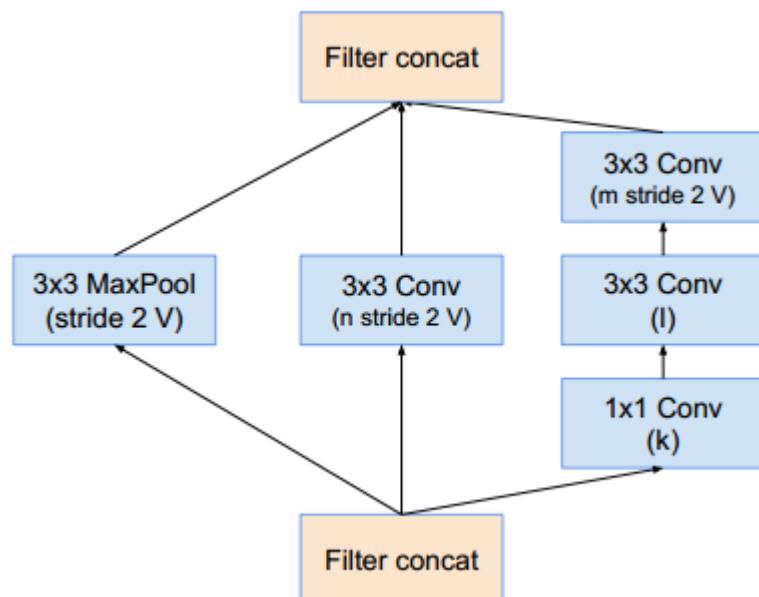
- Inception-B 模块（这样的模块有7个）：



- Inception-C 模块（这样的模块有3个）：

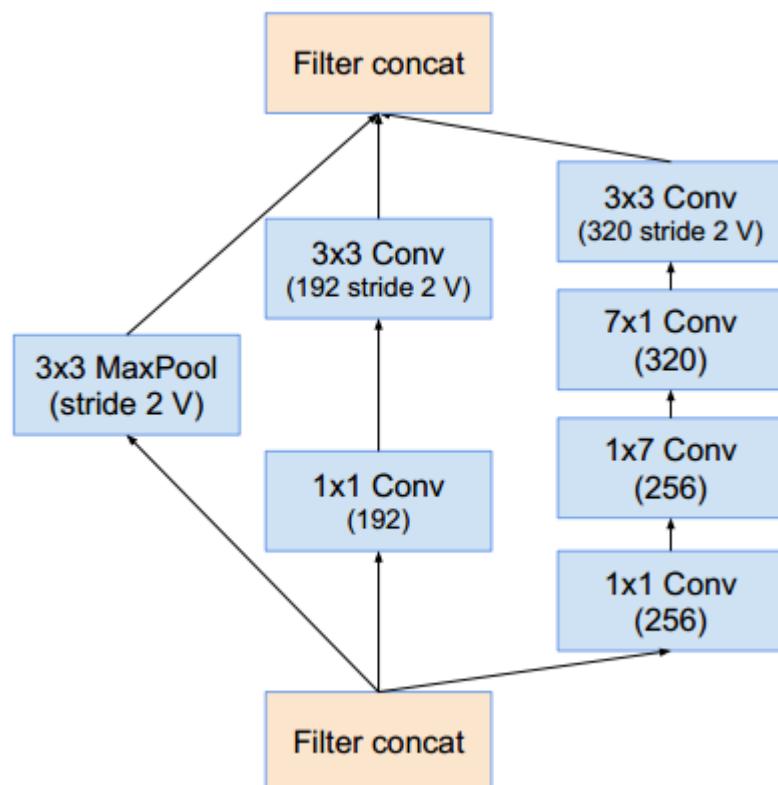


- Reduction-A 模块：(其中  $k, l, m, n$  分别表示滤波器的数量)



| 网络                  | <b>k</b> | <b>l</b> | <b>m</b> | <b>n</b> |
|---------------------|----------|----------|----------|----------|
| Inception-v4        | 192      | 224      | 256      | 384      |
| Inception-ResNet-v1 | 192      | 192      | 256      | 384      |
| Inception-ResNet-v2 | 256      | 256      | 256      | 384      |

- Reduction-B 模块：



#### 4.4.2 Inception-ResNet

1. 在 Inception-ResNet 中，使用了更廉价的 Inception 块。

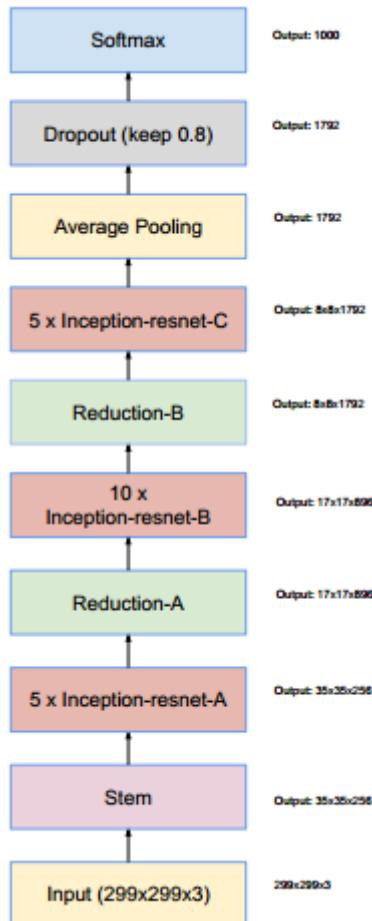
主要 inception 模块的池化运算由残差连接替代。

在 Reduction 模块中能够找到池化运算。

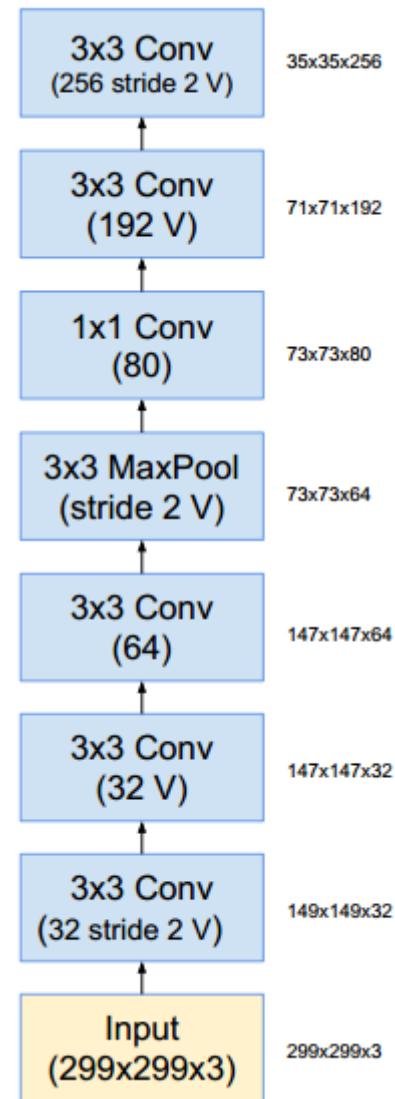
2. Inception ResNet 有两个版本：v1 和 v2。

- v1 的计算成本和 Inception v3 的接近。
- v2 的计算成本和 Inception v4 的接近。
- 它们有不同的 stem。
- 两个版本都有相同的模块 A、B、C 和缩减块结构。唯一不同在于超参数设置。

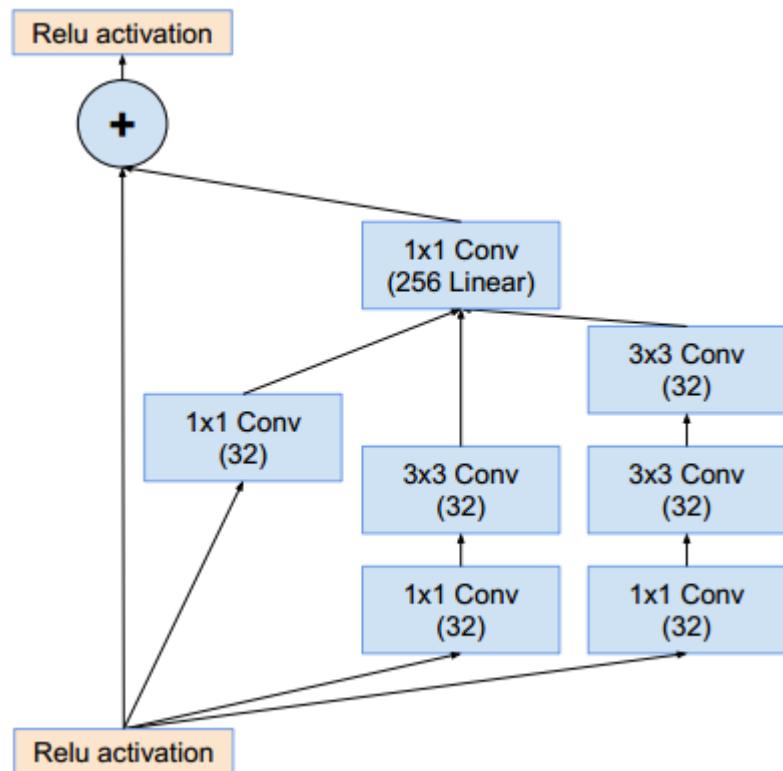
3. Inception-ResNet-v1 结构如下：



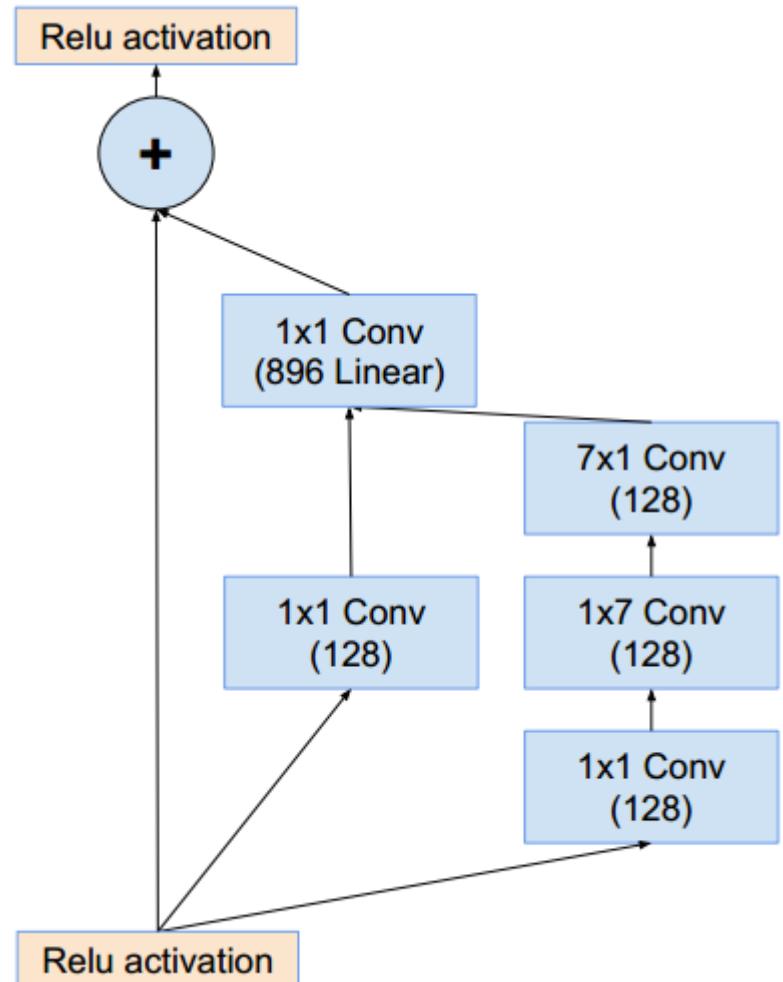
- stem 部分的结构：



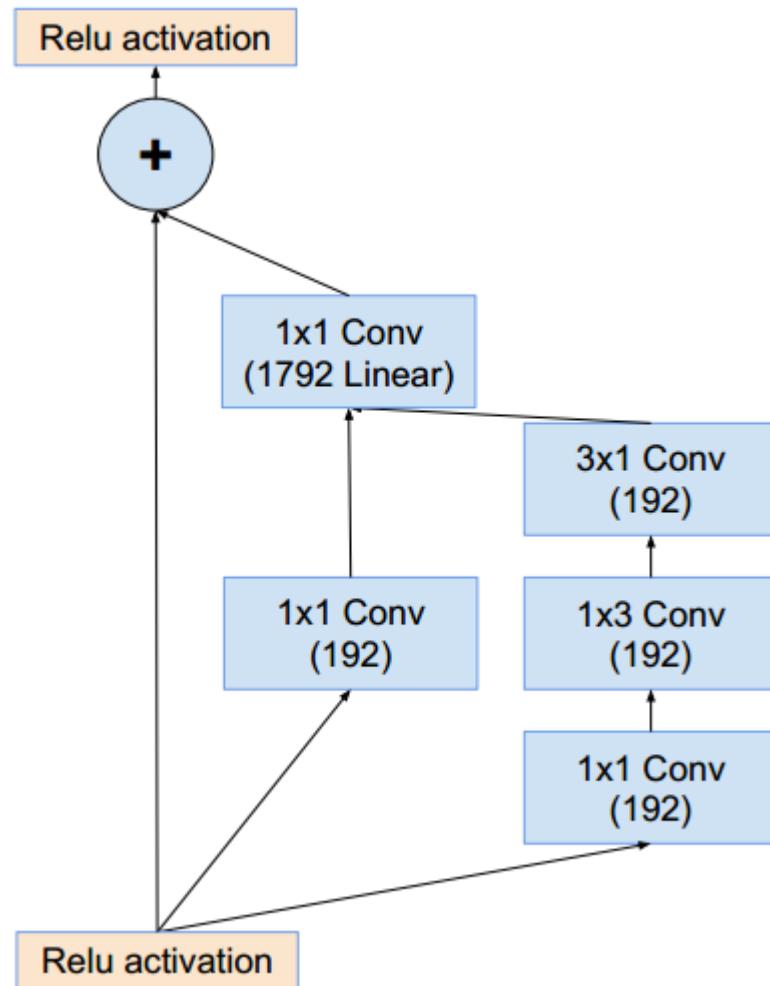
- Inception-ResNet-A 模块 (这样的模块有5个) :



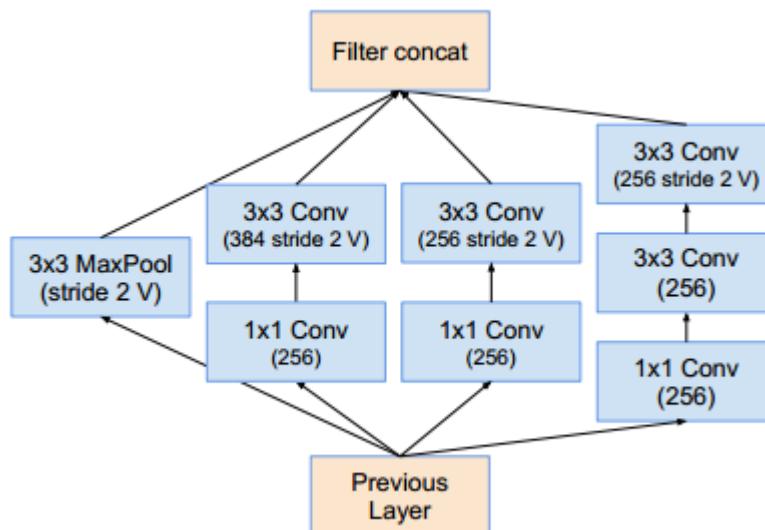
- Inception-B 模块（这样的模块有10个）：



- Inception-C 模块（这样的模块有5个）：



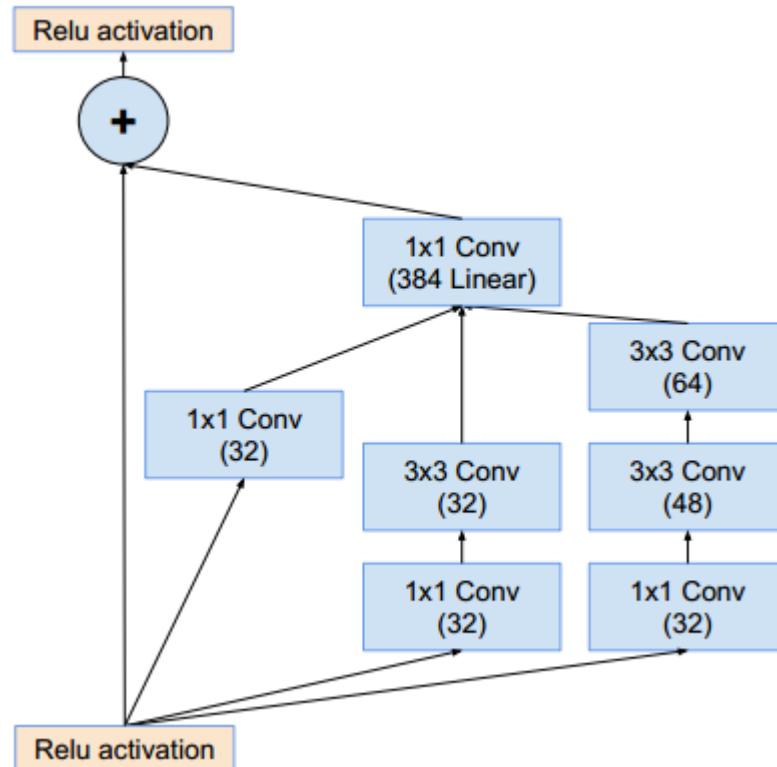
- Reduction-A 模块：同 `inception_v4` 的 Reduction-A 模块
- Reduction-B 模块：



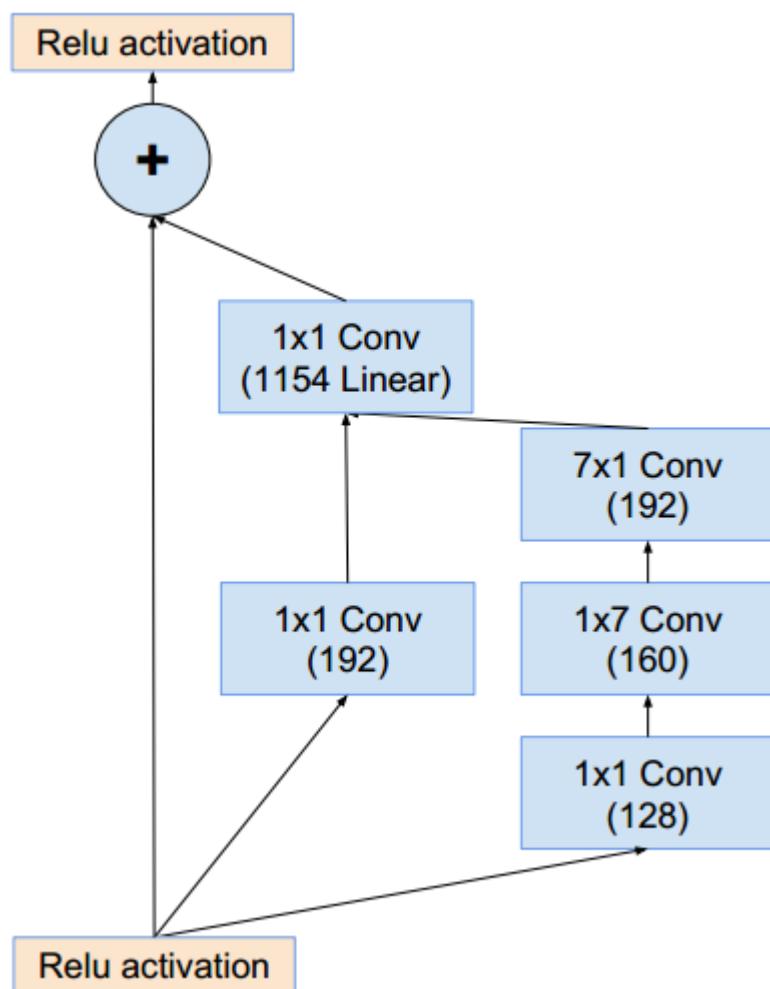
4. Inception-ResNet-v2 结构与 Inception-ResNet-v1 基本相同：

- | Inception-ResNet-v2 使用了 `inception_v4` 的 `stem` 部分。因此后续的通道数量都发生了改变。
- `stem` 部分的结构：同 `inception_v4` 的 `stem` 部分

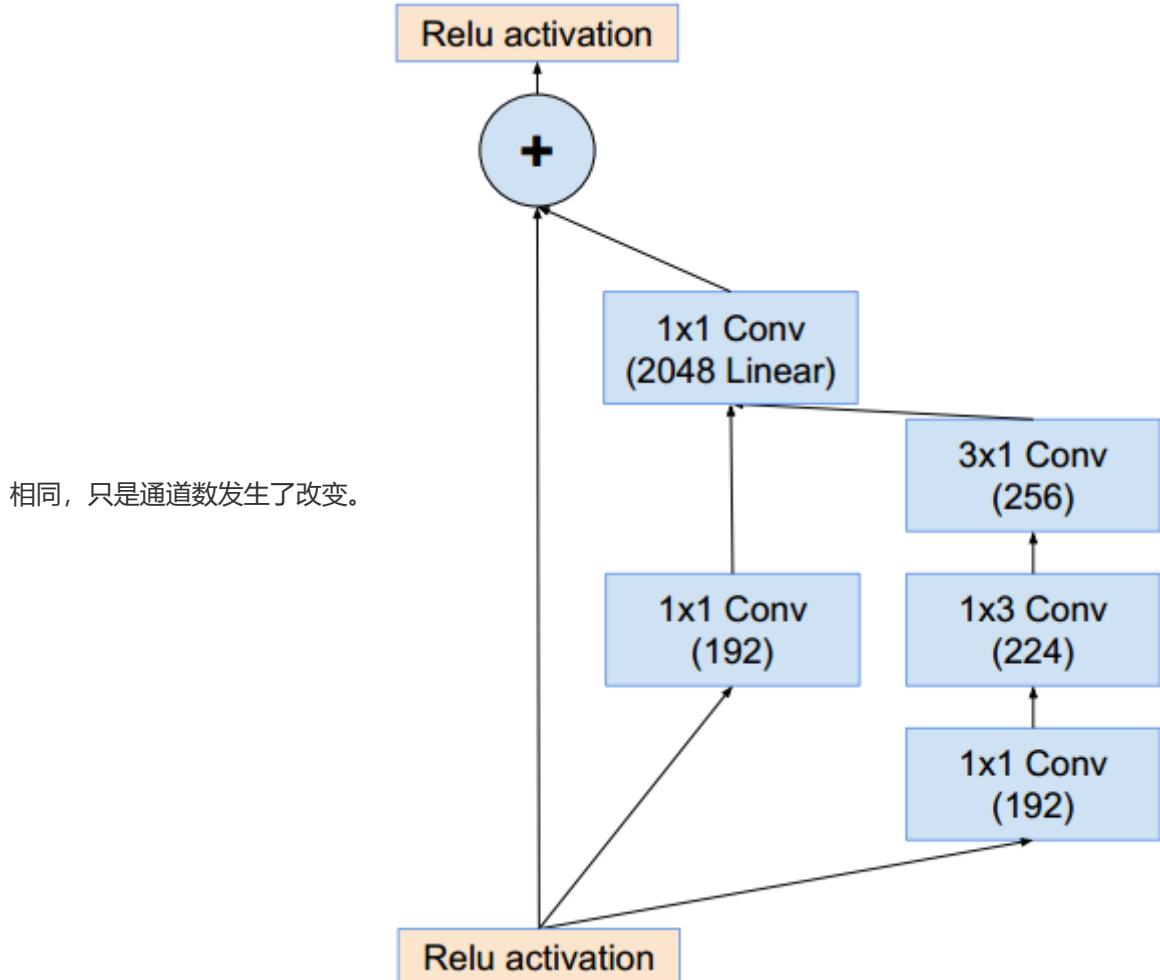
- Inception-ResNet-A 模块（这样的模块有5个）：它的结构与 Inception-ResNet-v1 的 Inception-ResNet-A 相同，只是通道数发生了改变。



- Inception-B 模块（这样的模块有10个）：它的结构与 Inception-ResNet-v1 的 Inception-ResNet-B 相同，只是通道数发生了改变。

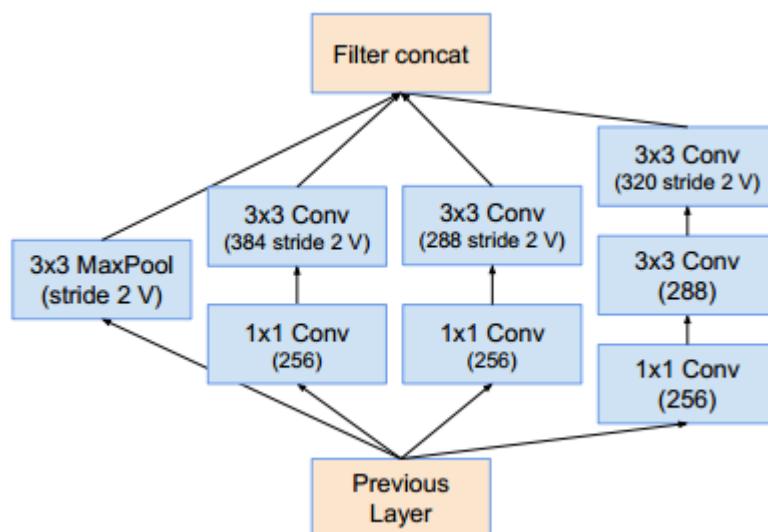


- Inception-C 模块（这样的模块有5个）：它的结构与 Inception-ResNet-v1 的 Inception-ResNet-C



相同，只是通道数发生了改变。

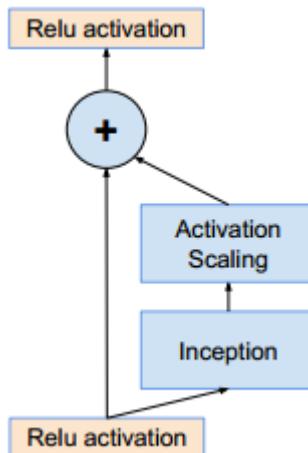
- Reduction-A 模块：同 inception\_v4 的 Reduction-A 模块。
- Reduction-B 模块：它的结构与 Inception-ResNet-v1 的 Reduction-B 相同，只是通道数发生了改变。



5. 如果滤波器数量超过1000，则残差网络开始出现不稳定，同时网络会在训练过程早期出现“死亡”：经过成千上万次迭代之后，在平均池化之前的层开始只生成0。

解决方案：在残差模块添加到 activation 激活层之前，对其进行缩放能够稳定训练。

- 这里将缩放因子定在  $0.1 \sim 0.3$ 。
- 降低学习率、或者增加额外的 BN 都无法避免这种状况。



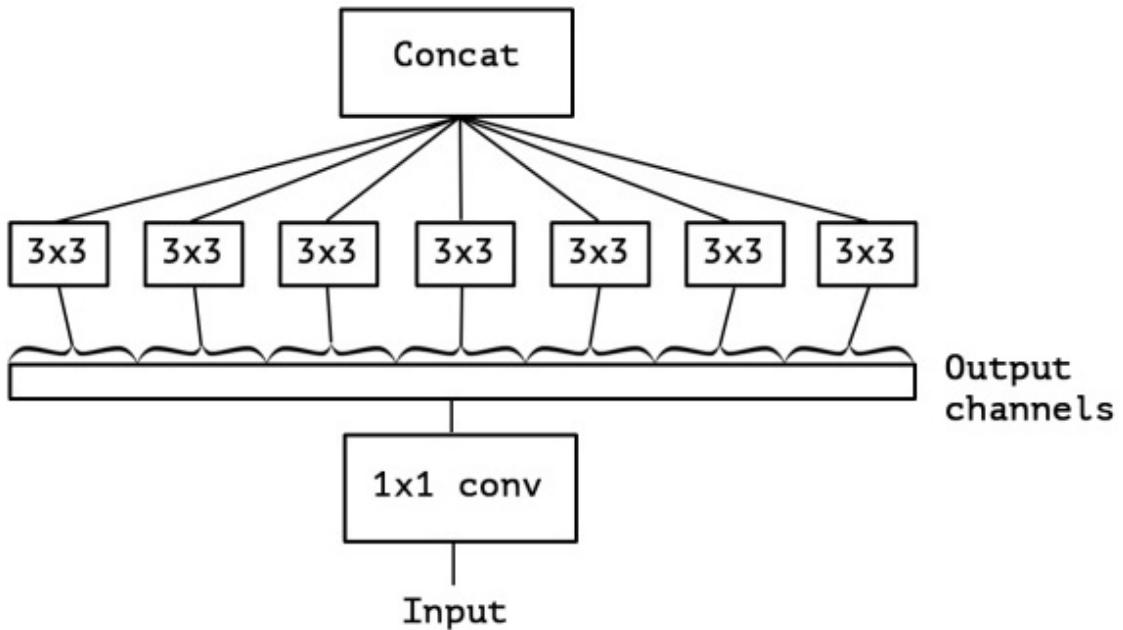
6. 在 Inception-resnet 中，仅仅在传统层（非残差块）的顶部使用 BN，而不是在所有层的顶部使用 BN。  
也可以在所有层的顶部使用 BN，但是更消耗计算资源。

## 4.5 Xception

1. Xception 的主要贡献是：在卷积神经网络的特征映射中，将跨通道相关性和空间相关性解耦。
2. Xception 的参数数量与 Inception V3 相同，但是性能表现显著优于 Inception V3。  
这表明 Xception 更加高效的利用了模型参数。

### 4.5.1 跨通道的相关性 & 空间相关性

1. 一个常规的卷积核尝试在三维空间中使用滤波器抽取特征，包括：两个空间维度（宽度和高度）、一个通道维度。  
因此单个卷积核的任务是：同时映射跨通道的相关性和空间相关性。
2. Inception 将这个过程明确的分解为一系列独立的相关性的映射：要么考虑跨通道相关性，要么考虑空间相关性。  
Inception 的做法是：
  - 首先通过一组  $1 \times 1$  卷积来查看跨通道的相关性，将输入数据映射到比原始输入空间小的三个或者四个独立空间。
  - 然后通过常规的  $3 \times 3$  或者  $5 \times 5$  卷积，将所有的相关性（包含了跨通道相关性和空间相关性）映射到这些较小的三维空间中。
3. Xception 将这一思想发挥到极致：
  - 首先使用  $1 \times 1$  卷积来映射跨通道相关性。
  - 然后分别映射每个输出通道的空间相关性。



因此该网络被称作 `Xception: Extreme Inception`。

#### 4.5.2 性质

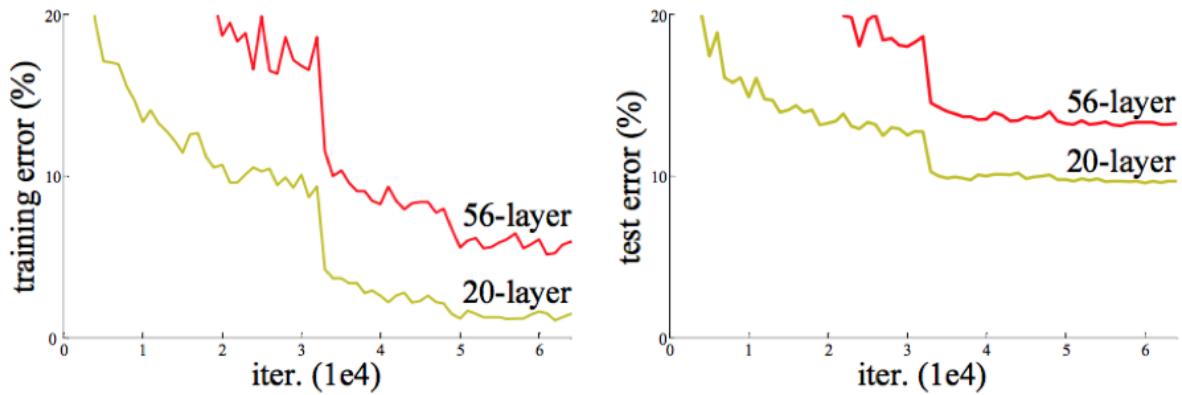
1. `Inception` 模块可以解释为介于常规卷积与深度可分离卷积之间的中间步骤。
  2. `Xception` 块类似于深度可分离卷积，但是它与深度可分离卷积之间有两个细微的差异：
    - 操作顺序不同：
      - 深度可分离卷积通常首先执行 `channel-wise` 空间卷积，然后再执行 `1x1` 卷积。
      - `Xception` 块首先执行 `1x1` 卷积，然后再进行 `channel-wise` 空间卷积。
    - 第一次卷积操作之后是否存在非线性：
      - 深度可分离卷积只有第二个卷积(`1x1`)使用了 `ReLU` 非线性激活函数
      - `Xception` 块的两个卷积 (`1x1` 和 `3x3`) 都使用了 `ReLU` 非线性激活函数
- 其中第二个差异更为重要。
3. 对 `Xception` 进行以下的修改，都可以加快网络收敛速度，并获取更高的准确率：
    - 引入类似 `ResNet` 的残差连接机制。
    - 在 `1x1` 卷积和 `3x3` 卷积之间不加入任何非线性。

## 五、ResNet

1. `ResNet` 主要贡献：提出了一种残差学习框架来解决网络退化问题，从而训练更深的网络。
2. 在 `ImageNet` 分类数据集中，作者给出了152层残差网络，以 `3.75% top-5` 的错误率获得了 `ILSVRC 2015` 分类比赛的冠军。

### 5.1 网络退化问题

1. 学习更深的网络的一个障碍是梯度消失/爆炸，该问题可以通过 `Batch Normalization` 在很大程度上解决。
2. `ResNet` 论文作者发现：随着网络的深度的增加，准确率达到饱和之后迅速下降，而这种下降不是由过拟合引起的。这称作网络退化问题。  
如下所示：更深的网络导致更高的训练误差和测试误差。



3. 理论上讲，较深的模型不应该比它对应的、较浅的模型更差。因为较深的模型是较浅的模型的超空间。

- 较深的模型可以这样得到：先构建较浅的模型，然后添加很多恒等的映射层。
- 退化问题表明：通过多个非线性层来近似恒等映射可能是困难的。

4. 解决网络退化问题的方案：学习残差。

5. 经过很多证据表明：残差学习准则是通用的，而且期望它也适用于视觉问题和非视觉问题。

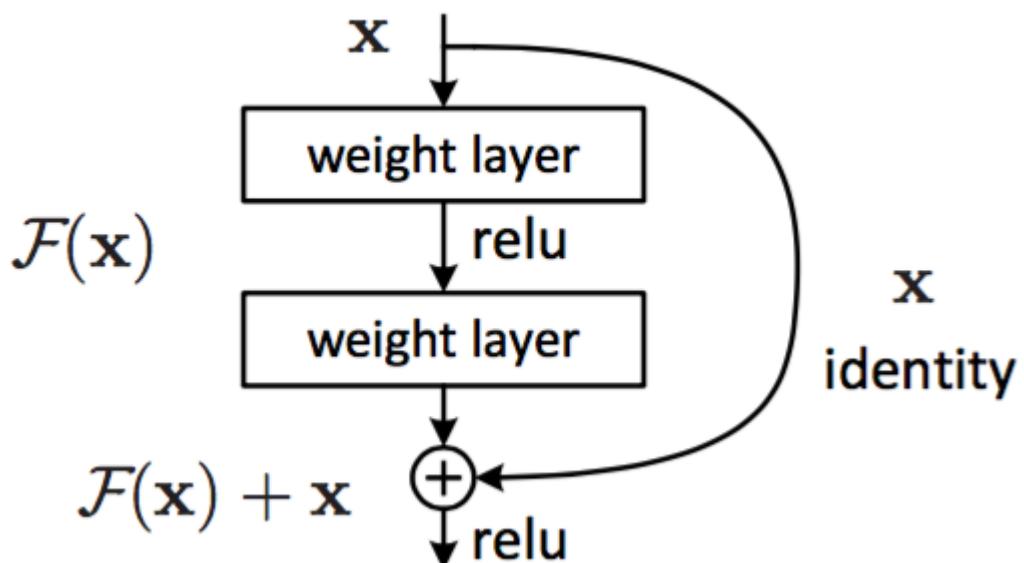
## 5.2 残差块

1. 假设需要学习的是映射  $\vec{y} = H(\vec{x})$ ，残差块使用堆叠的非线性层拟合残差：

$$\vec{y} = F(\vec{x}, \mathbf{W}_i) + \vec{x}$$

其中：

- $\vec{x}$  和  $\vec{y}$  是块的输入和输出向量
- $F(\vec{x}, \mathbf{W}_i)$  是要学习的残差映射。因为  $F(\vec{x}, \mathbf{W}_i) = H(\vec{x}) - \vec{x}$ ，因此称作残差。
- $+ \quad$ ：通过快捷连接逐个元素相加来执行。相加之后通过非线性激活函数。



2. “快捷连接”：那些跳过一层或者更多层的连接。

- 快捷连接简单的执行恒等映射，并将其输出添加到堆叠层的输出。
- 快捷连接既不增加额外的参数，也不增加计算复杂度。

3. 前面给出的残差块隐含了一个假设:  $F(\vec{x}, \mathbf{W}_i)$  和  $\vec{x}$  的维度相等。

如果它们的维度不等, 则需要在快捷连接中对  $\vec{x}$  执行线性投影来匹配维度:

$$\vec{y} = F(\vec{x}, \mathbf{W}_i) + \mathbf{W}_s \vec{x}$$

事实上当它们维度相等时, 也可以执行线性变换。但是实践表明:

- 使用恒等映射足以解决退化问题。
- 使用线性投影会增加参数和计算复杂度。

因此  $\mathbf{W}_s$  仅在匹配维度时使用。

4. 残差函数  $F$  的形式是可变的。

- 论文中的实验包含有两层堆叠、三层堆叠。也可以包含更多层的堆叠。
- 如果  $F$  只有一层, 则残差块退化为线性层:  $\vec{y} = \mathbf{W}_1 \vec{x} + \vec{x}$ , 此时对网络并没有什么提升。

5. 尽管这里讨论的符号全部是基于全连接层的, 它们同样适用于卷积层。

当用于卷积层时:

- $F(\vec{x}, \mathbf{W}_i)$  可以表示多个卷积层的堆叠
- 元素加法在两个 feature map 上逐通道进行。

6. 残差学习成功的原因: 学习残差  $F(\vec{x}, \mathbf{W}_i)$  比学习原始映射  $H(\vec{x})$  要更容易。

- 当原始映射  $H$  就是一个恒等映射时,  $F$  就是一个零映射。

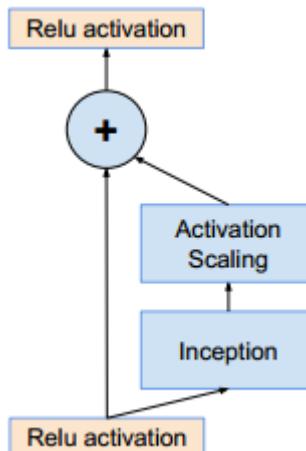
此时求解器只需要简单的将堆叠的非线性连接的权重推向零即可。

- 虽然实际任务中原始映射  $H$  可能不是一个恒等映射, 但是如果  $H$  更偏向于恒等映射 (而不是更偏向于非恒等映射), 则  $F$  就是关于恒等映射的抖动, 也会更容易学习。
- 如果原始映射  $H$  更偏向于零映射, 那么学习  $H$  本身要更容易。

但是在实际应用中, 零映射非常少见, 它会导致输出全为0。

- 如果原始映射  $H$  是一个非恒等映射, 则可以考虑对残差模块使用缩放因子。

- 如 Inception-Resnet 中: 在残差模块添加到 activation 激活层之前, 对其进行缩放。
- 注意: ResNet 作者在随后的论文中指出: 不应该对恒等映射进行缩放。因此这里对残差模块进行缩放。



- 可以通过观察  $F$  的输出来判断:

- 如果  $F$  的输出均为0附近的、较小的数, 则说明原始映射  $H$  更偏向于横等映射
- 否则, 说明原始映射  $H$  更偏向于非横等映射。

7. Veit et al. 认为 ResNet 工作较好的原因是: 一个 ResNet 网络可以看做是一组较浅的网络的集成模型。

但是 ResNet 的作者认为这个解释是不正确的。因为集成模型要求每个子模型是独立训练的，而这组较浅的网络是共同训练的。

## 5.3 ResNet 分析

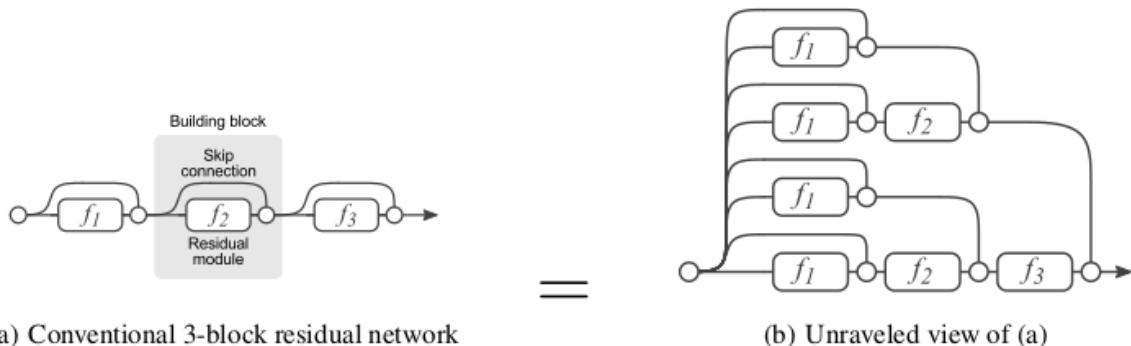
1. 论文《Residual Networks Behave Like Ensemble of Relatively Shallow Networks》对 ResNet 进行了深入的分析。
2. ResNet 以三种方式挑战了传统的机器视觉架构：
  - ResNet 通过引入跳跃连接来绕过残差层，这允许数据直接流向任何后续层。  
这与传统的、顺序的 pipeline 形成鲜明对比：传统的架构中，网络依次处理低级 feature 到高级 feature。
  - ResNet 的层数非常深，高达1202层。  
而 AlexNet 这样的架构，网络层数要小两个量级。
  - 通过实验发现，训练好的 ResNet 中去掉单个层并不会影响其预测性能。  
而训练好的 AlexNet 等网络中，移除层会导致预测性能损失。
3. 论文的主要贡献：
  - 通过分解视图，表明：ResNet 可以被视作许多路径的集合
  - 通过研究 ResNet 的梯度流，表明：网络训练期间只有短路径才会产生梯度流，深的路径不是必须的。
  - 通过破坏性实验，表明：
    - 即使这些路径是共同训练的，它们也不是相互依赖的。
    - 这些路径的行为类似集成模型，其预测准确率平滑地与有效路径的数量有关。

### 5.3.1 分解视图

1. 考虑从输出  $\vec{y}_0$  到  $\vec{y}_3$  的三个 ResNet 块构建的网络。根据：

$$\begin{aligned}\vec{y}_3 &= \vec{y}_2 + f_3(\vec{y}_2) = [\vec{y}_1 + f_2(\vec{y}_1)] + f_3(\vec{y}_1 + f_2(\vec{y}_1)) \\ &= [\vec{y}_0 + f_1(\vec{y}_0) + f_2(\vec{y}_0 + f_1(\vec{y}_0))] + f_3(\vec{y}_0 + f_1(\vec{y}_0) + f_2(\vec{y}_0 + f_1(\vec{y}_0)))\end{aligned}$$

下图中，左图为原始形式，右图为分解视图。分解视图中展示了数据从输入到输出的多条路径。



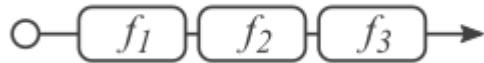
2. 分解视图中，每条路径可以通过二进制编码向量  $\vec{b} = (b_1, b_2, \dots, b_n)$ ,  $b_i \in \{0, 1\}$  来索引：

- 如果流过残差块  $f_i$ ，则  $b_i = 1$
- 如果跳过残差块  $f_i$ ，则  $b_i = 0$

因此 ResNet 从输入到输出具有  $2^n$  条路径。

3. ResNet 中，第  $i$  个残差块  $f_i$  的输入汇聚了之前的  $i - 1$  个残差块的  $2^{i-1}$  条路径。
4. 对于严格顺序的网络（如 VGG），这些网络中的输入总是在单个路径中从第一层直接流到最后一层。如下图所示。

$$y_3 = f_3(f_2(f_1(y_0)))$$



5. 普通的前馈神经网络也可以在单个神经元（而不是网络层）这一粒度上运用分解视图。这也可以将网络分解为不同路径的集合。

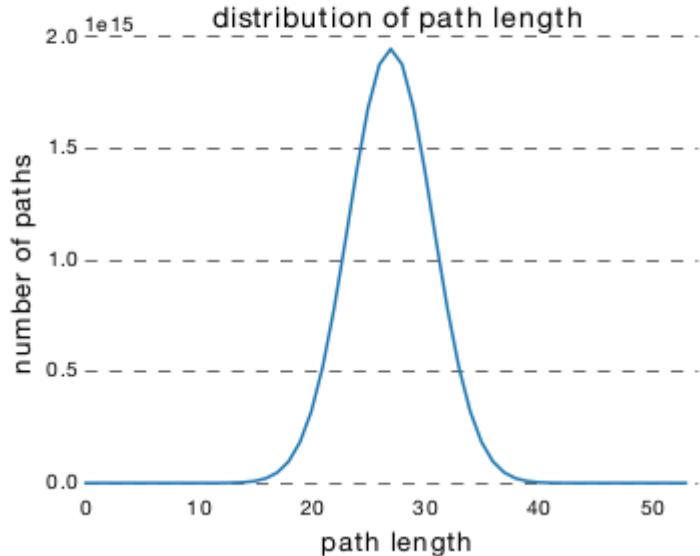
它与 ResNet 分解的区别是：

- 普通前馈神经网络的神经元分解视图中，所有路径都具有相同的长度。
- ResNet 网络的残差块分解视图中，所有路径具有不同的路径长度。

### 5.3.1 路径长度分析

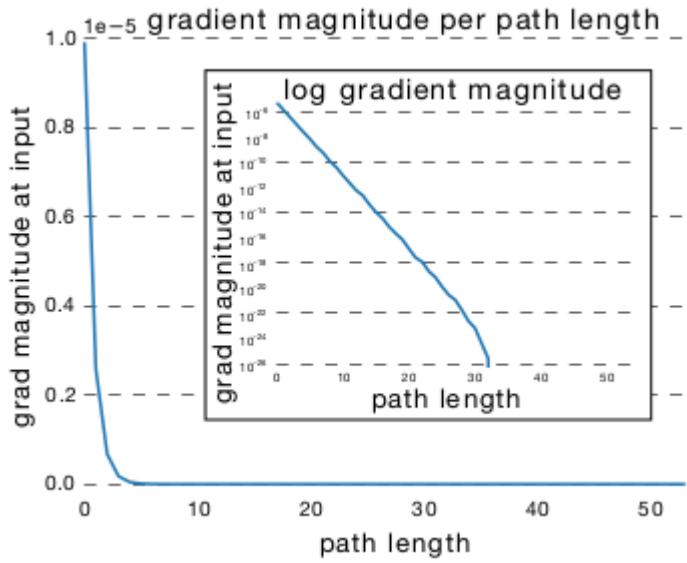
1. ResNet 中，从输入到输出存在许多条不同长度的路径。
2. ResNet 路径长度的分布服从二项分布。对于  $n$  层深的 ResNet，大多数路径的深度为  $\frac{n}{2}$ 。

下图为一个 54 个块的 ResNet 网络的路径长度的分布，其中 95% 的路径只包含 19 ~ 35 个块。

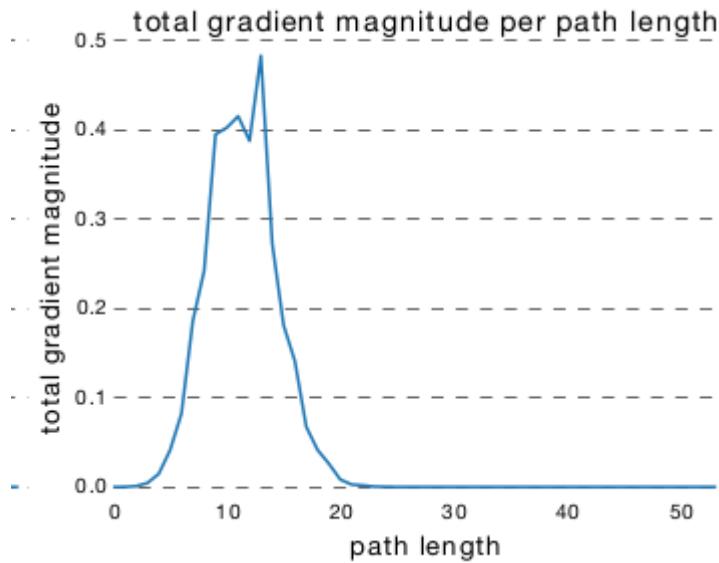


### 5.3.2 路径梯度分析

1. ResNet 中，路径的梯度幅度随着它在反向传播中经过的残差块的数量呈指数减小。因此，训练期间大多数梯度来源于更短的路径。
2. 对于一个包含 54 个残差块的 ResNet 网络：
  - 下图表示：单条长度为  $k$  的路径在反向传播到 input 处的梯度的幅度的期望。
  - 它刻画了长度为  $k$  的单条路径的对于更新的影响。
  - 因为长度为  $k$  的路径有多条，因此取其期望（也就是均值）



- 下图表示：长度为  $k$  的所有路径在反向传播到 `input` 处的梯度的幅度的和。
- 它刻画了长度为  $k$  的所有路径对于更新的影响。它不仅取决于长度为  $k$  的单条路径的对于更新的影响，还取决于长度为  $k$  的单条路径的数量。



3. `ResNet` 中有效路径相对较浅，而且有效路径数量占比较少。

在一个54个块的 `ResNet` 网络中：

- 几乎所有的梯度更新都来自于长度为 5~17 的路径
- 长度为 5~17 的路径占网络所有路径的 0.45%

4. 作者从头开始重新训练 `ResNet`，同时在训练期间只保留有效路径，确保不使用长路径。

实验结果表明：相比于完整模型的 6.10% 的错误率，这里实现了 5.96% 的错误率。二者没有明显的统计学上的差异，这表明确实只需要有效路径。

5. `ResNet` 不是让梯度流流通整个网络深度来解决梯度消失问题，而是引入能够在非常深的网络中传输梯度的短路径来避免梯度消失问题。

6. 随机深度网络起作用有两个原因：

- 训练期间，网络看到的路径分布会发生变化，主要是变得更短。

- 训练期间，每个 `mini-batch` 选择不同的短路径的子集，这会鼓励路径独立地产生良好的结果。

### 5.3.3 路径破坏性分析

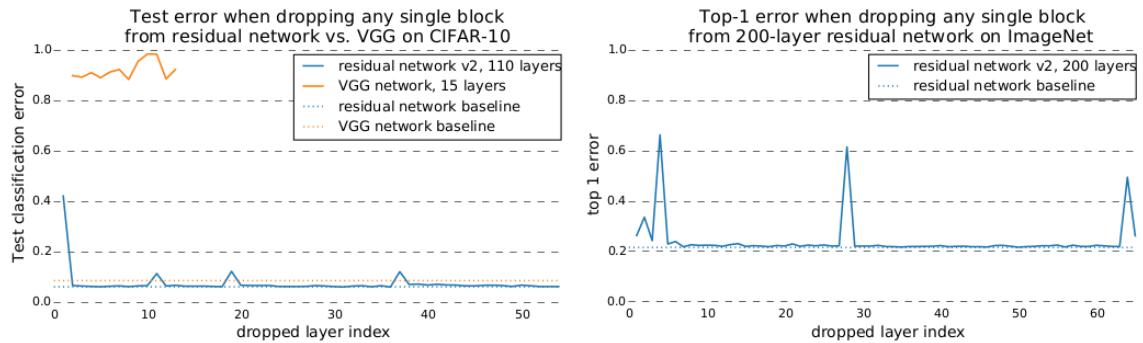
1. 块的删除：

- 在 `ResNet` 网络训练完成之后，如果随机丢弃单个残差块，则测试误差基本不变。

因为移除一个残差块时，`ResNet` 中路径的数量从  $2^n$  减少到  $2^{n-1}$ ，留下了一半的路径。

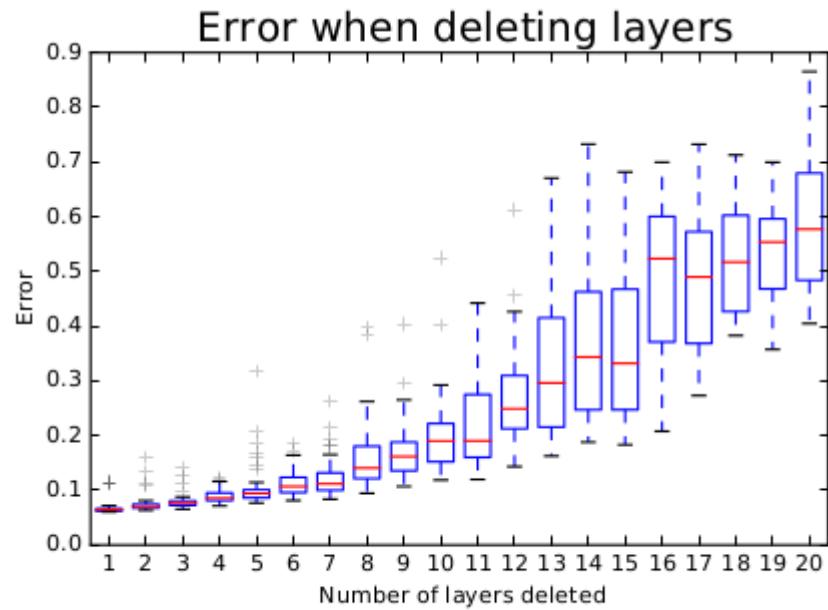
- 在 `VGG` 网络训练完成之后，如果随机丢弃单个块，则测试误差急剧上升，预测结果就跟随机猜测差不多。

因为移除一个块时，`VGG` 中唯一可行的路径被破坏。



2. `ResNet` 网络中，路径的集合表现出一种类似集成模型的效果。一个关键证据是：它们的整体表现平稳地取决于路径的数量。

随着网络删除越来越多的残差块，网络路径的数量降低，测试误差平滑地增加。



(a)

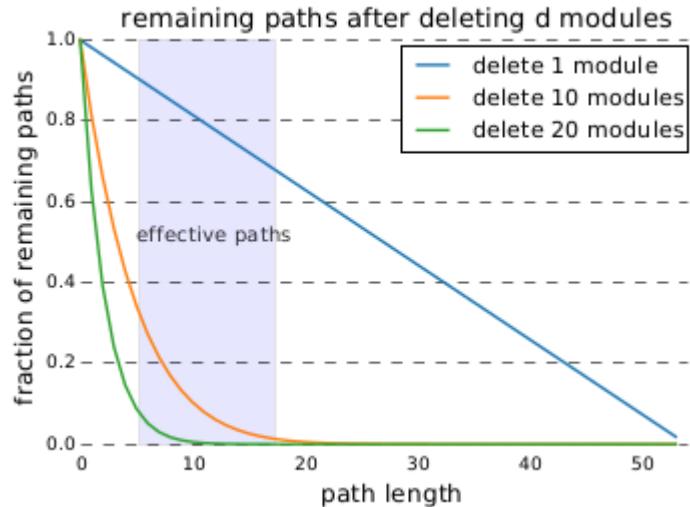
3. 删除 `ResNet` 残差块通常会删除长路径。

当删除了  $k$  个残差块时，长度为  $x$  的路径的剩余比例由下式给定：

$$percent = \frac{C_{n-k}^x}{C_n^x}$$

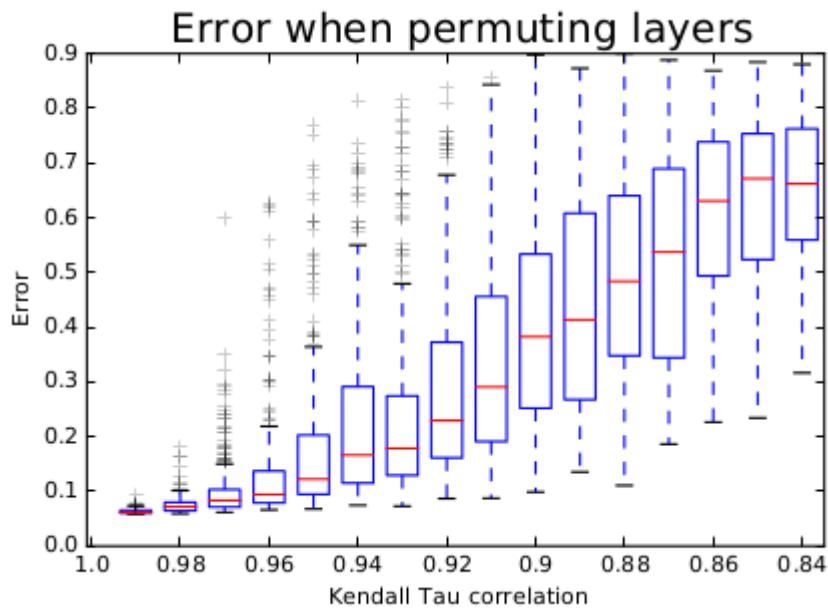
下图中：

- 删除10个残差模块，一部分有效路径（路径长度为 5~17）仍然被保留，模型测试性能会部分下降
- 删除20个残差模块，绝大部分有效路径（路径长度为 5~17）被删除，模型测试性能会大幅度下降



4. 如果在测试时，重新排序网络的残差块，这意味着交换了低层转换和高层转换。通过 Kendall Tau rank 来衡量网络结构被破坏的程度。

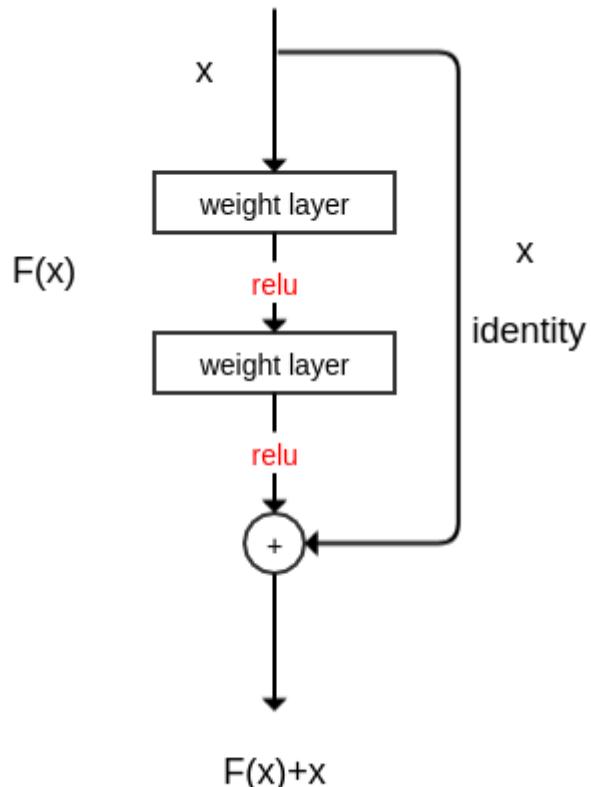
结果表明：随着 Kendall Tau rank 的增加，预测错误率也在增加。



## 5.4 ResNet 变种

### 5.4.1 恒等映射修正

- 在论文《Identity Mappings in Deep Residual Networks》中，ResNet 的作者通过实验证明了恒等映射的重要性，并且提出了一个新的残差单元来简化恒等映射。
- 新的残差单元中，恒等映射添加到 ReLU 激活函数之后。  
它使得训练变得更简单，并且提高了网络的泛化能力。



- 假设  $\vec{x}_l$  是第  $l$  个残差单元的输入特征； $\mathcal{W}_l = \{\mathbf{W}_{l,k} \mid 1 \leq k \leq K\}$  为一组与第  $l$  个残差单元相关的权重（包括偏置项）， $K$  是残差单元中的层的数量； $\mathcal{F}$  代表残差函数。则对于新的残差单元，有：

$$\vec{x}_{l+1} = \vec{x}_l + \mathcal{F}(\vec{x}_l, \mathcal{W}_l)$$

考虑递归，对于任意深的残差单元  $L$ ，则有：

$$\begin{aligned}\vec{x}_L &= \vec{x}_{L-1} + \mathcal{F}(\vec{x}_{L-1}, \mathcal{W}_{L-1}) \\ &= \vec{x}_{L-2} + \mathcal{F}(\vec{x}_{L-1}, \mathcal{W}_{L-1}) + \mathcal{F}(\vec{x}_{L-2}, \mathcal{W}_{L-2}) \\ &\quad \vdots \\ &= \vec{x}_l + \sum_{i=l}^{L-1} \mathcal{F}(\vec{x}_i, \mathcal{W}_i)\end{aligned}$$

#### 5.4.1.1 新残差单元性质

1. 对任意深的单元  $L$ ，其特征  $\vec{x}_L$  可以表示为浅层单元  $l$  的特征  $\vec{x}_l$  加上一个形如  $\sum_{i=l}^{L-1} \mathcal{F}(\vec{x}_i, \mathcal{W}_i)$  的残差函数。

这意味着：任意单元  $L$  和  $l$  之间都具有残差性。

2. 对于任意深的单元  $L$ ，其特征  $\vec{x}_L$  可以表示为：

$$\vec{x}_L = \vec{x}_0 + \sum_{i=0}^{L-1} \mathcal{F}(\vec{x}_i, \mathcal{W}_i)$$

即为：之前所有残差函数输出的总和，再加上  $\vec{x}_0$ 。

与之形成鲜明对比的是常规网络中，特征  $\vec{x}_L$  是一系列矩阵向量的乘积。即为： $\vec{x}_L = \prod_{i=0}^{L-1} \mathbf{W}_i \vec{x}_0$ （忽略了激活函数和 BN）。

3. 新的残差单元也更具有良好的反向传播特性。

对于损失函数  $\mathcal{L}$ ，有：

$$\frac{\partial \mathcal{L}}{\partial \vec{x}_l} = \left( \frac{\partial \vec{x}_L}{\partial \vec{x}_l} \right)^T \frac{\partial \mathcal{L}}{\partial \vec{x}_L} = \left( 1 + \frac{\partial \sum_{i=l}^{L-1} \mathcal{F}(\vec{x}_i, \mathcal{W}_i)}{\partial \vec{x}_l} \right)^T \frac{\partial \mathcal{L}}{\partial \vec{x}_L}$$

可以看到：

- 梯度  $\frac{\partial \mathcal{L}}{\partial \vec{x}_l}$  可以分解为两个部分：
  - $\frac{\partial \mathcal{L}}{\partial \vec{x}_L}$ ：直接传递信息而不涉及任何权重。它保证了信息能够直接传回给任意浅层  $l$ 。
  - $\left( \frac{\partial \sum_{i=l}^{L-1} \mathcal{F}(\vec{x}_i, \mathcal{W}_i)}{\partial \vec{x}_l} \right)^T \frac{\partial \mathcal{L}}{\partial \vec{x}_L}$ ：通过权重层来传递
- 在一个 mini-batch 中，不可能出现梯度消失的情况。

可能对于某个样本，存在  $\frac{\partial \sum_{i=l}^{L-1} \mathcal{F}(\vec{x}_i, \mathcal{W}_i)}{\partial \vec{x}_l} = -1$  的情况。但是不可能出现 mini-batch 中所有的样本满足  $\frac{\partial \sum_{i=l}^{L-1} \mathcal{F}(\vec{x}_i, \mathcal{W}_i)}{\partial \vec{x}_l} = -1$ 。

这意味着：哪怕权重是任意小的，也不可能出现梯度消失的情况。

对于旧的残差单元，由于恒等映射还需要经过 ReLU 激活函数，因此当  $\vec{x} + \mathcal{F}(\vec{x}, \mathcal{W}) < 0$  时饱和，其梯度为0。

4. 根据 2. 和 3. 的讨论表明：在前向和反向阶段，信号都能够直接的从一个单元传递到其他任意单元。

#### 5.4.1.2 实验和结论

1. 快捷连接验证：

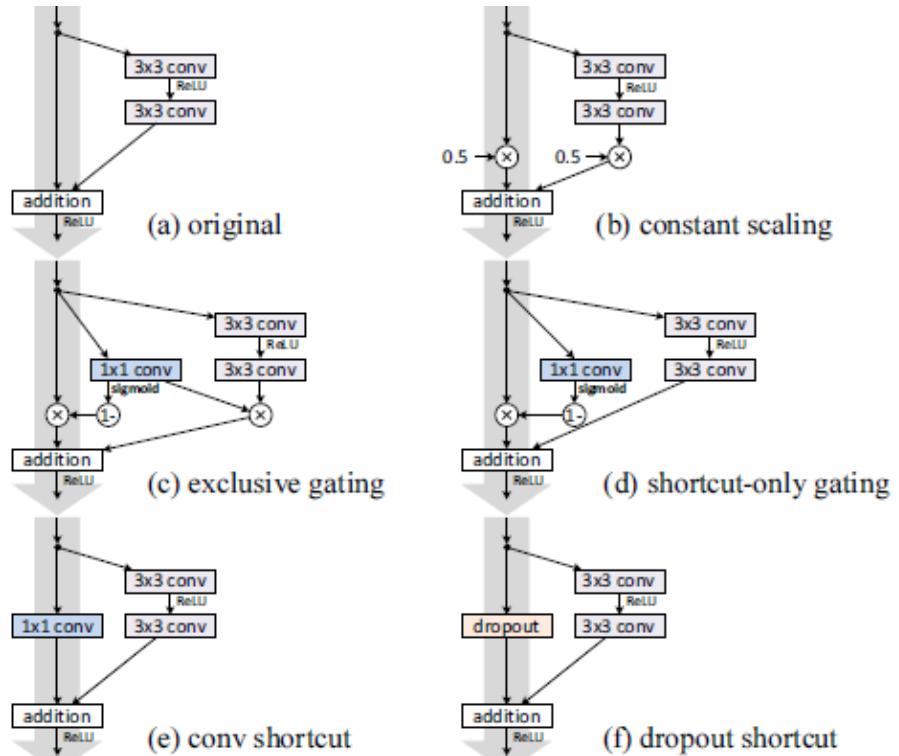
为了简化，这里没有画出 BN 层。每个权重层的后面实际上都有一个 BN 层。

- (a)：原始的、旧的残差块。
- (b)：对所有的快捷连接，设置缩放。其中缩放因子  $\lambda = 0.5$ 。
- (c)：对快捷连接执行门控机制。残差由  $g$  来缩放，快捷连接由  $1 - g$  来缩放。
- (d)：对快捷连接执行门控机制，但是残差并不进行缩放。

- (e) : 对快捷连接执行  $1 \times 1$  卷积。
- (f) : 对快捷连接执行 `dropout` , 其中遗忘比例为0.5。

在统计学上，它等效于一个缩放比例为0.5的缩放操作。

最终结果表明：快捷连接是信息传递最直接的路径，快捷连接中的各种操作都会阻碍信息的传递，以致于对优化造成困难。

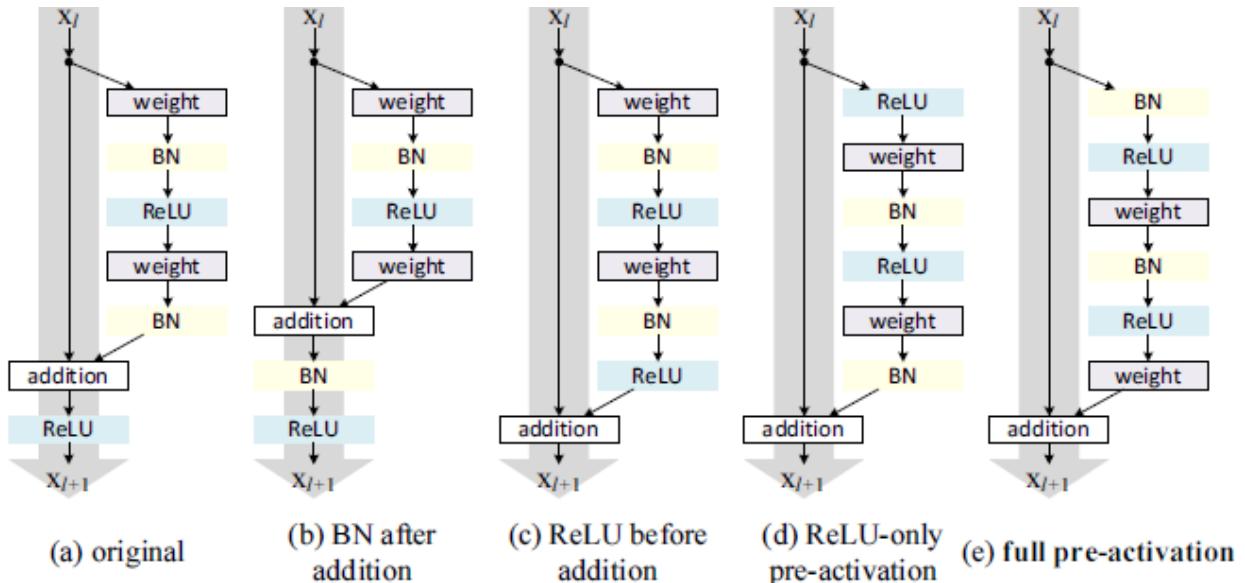


2. 理论上，对快捷连接执行  $1 \times 1$  卷积，会引入更多的参数。它应该比恒等连接具备更强大的表达能力。

事实上，其训练误差要比恒等连接的训练误差高的多。这意味着模型退化是因为优化问题，而不是网络表达能力的问题。

3. 预激活验证：

所有的结构的组件都相同，只是组件放置的位置不同。



- (a) : 原始的、旧的残差块。
- (b) : 将 BN 移动到 addition 之后。
- (c) : 将 ReLU 移动到 addition 之前。这种结构问题较大，因为：
  - 理想的残差块的输出范围是  $(-\infty, +\infty)$ 。
  - 这里的残差块经过个 ReLU 之后的输出为非负，从而使得残差的输出为  $[0, +\infty)$ 。从而使得前向信号会逐级递增。这会影响网络的表达能力。
- (d) : 将 ReLU 移动到残差块之前。
- (e) : 将 BN 和 ReLU 移动到残差块之前。

最终结果表明：full pre-activation 效果最好。有两个原因：

- 快捷连接通路是顺畅的，这使得优化更加简单
- 对两个权重层的输入都执行了 BN。

在其它四组结构中，只有第二个权重层的输入的到了标准化。第一个权重层的输入并未的到标准化。

## 5.4.2 ResNeXt

1. 以往提高模型准确率的方法都是：加深网络深度或者加宽网络宽度。但这些方法会增加超参数的数量，网络的计算开销也会增加。

ResNeXt 网络可以在不增加网络参数复杂度的前提下提高准确率，同时还减少了超参数的数量。

ResNeXt 的做法是：改进了 ResNet 网络结构，并提出了一个新的维度，称作“基数” cardinality。

- 基数是网络的深度和网络的宽度之外的另一个重要因素。
- 作者通过实验表明：增加基数比增加网络的深度或者网络的宽度更有效。

2. 在相同的参数数量和计算复杂度的情况下，ResNeXt 的预测性能要优于 ResNet。

- 它在 ILSVRC 2016 分类任务中取得了第二名的成绩。
- 101 层的 ResNeXt 就能够获得超过 200 层 ResNet 的准确率，并且计算量只有后者的一半。

3. ResNeXt 的提出参考了 VGG 和 Inception 的设计哲学。

- VGG：网络通过简单地层叠相同结构的层来实现，因此网络结构简单。

其缺点是网络参数太多，计算量太大。

- Inception：通过执行“分裂-变换-合并”策略来精心设计拓扑结构，使得网络参数较少，计算复杂度较低。

这种“分裂-变换-合并”行为预期能够达到一个大的 dense 层的表达能力，但是计算复杂度要低的多。

其缺点是：

- 每个“变换”中，滤波器的数量和尺寸等超参数都需要精细的设计。
- 一旦需要训练新的任务（如新任务是一个 NLP 任务），可能需要重新设计网络结构。因此可扩展性不高。

- ResNeXt 结合了二者的优点：

- 网络结构也是通过简单地层叠相同结构的层来实现。
- 网络的每一层都执行了“分裂-变换-合并”策略。

### 5.4.2.1 分裂-变换-合并

1. 考虑全连接网络中的一个神经元。假设输入为  $\vec{x} = (x_1, x_2, \dots, x_D)$ ，为一个  $D$  通道的输入向量。

假设对应的权重为  $\vec{w} = (w_1, w_2, \dots, w_D)$ 。

不考虑偏置和激活函数，则神经元的输出为：

$$\sum_{i=1}^D w_i x_i$$

它可以视作一个最简单的“分裂-变换-合并”：

- 输入被分割成  $D$  个低维嵌入：实际上分割到 0 维。
- 每个低维嵌入执行变换：通过对应的权重  $w_i$ 。
- 变换之后的结果进行合并：通过直接相加来合并。

2. Inception 的“分裂-变换-合并”策略：

- 输入通过  $1 \times 1$  卷积被分割成几个低维嵌入
- 每个低维嵌入分别使用一组专用滤波器（ $3 \times 3$ 、 $5 \times 5$  等）执行变换
- 变换之后的结果进行合并

3. 对一个 ResNeXt 块，其“分裂-变换-合并”策略用公式表述为：

$$\begin{aligned}\mathcal{F}(\vec{x}) &= \sum_{i=1}^C \mathcal{T}_i(\vec{x}) \\ \vec{y} &= \vec{x} + \mathcal{F}(\vec{x})\end{aligned}$$

其中：

- $\mathcal{T}_i$  为任意函数，它将  $\vec{x}$  映射为  $\vec{x}$  的一个低维嵌入，并对该低维嵌入执行转换。
- $C$  为转换的数量，也就是基数 cardinality。

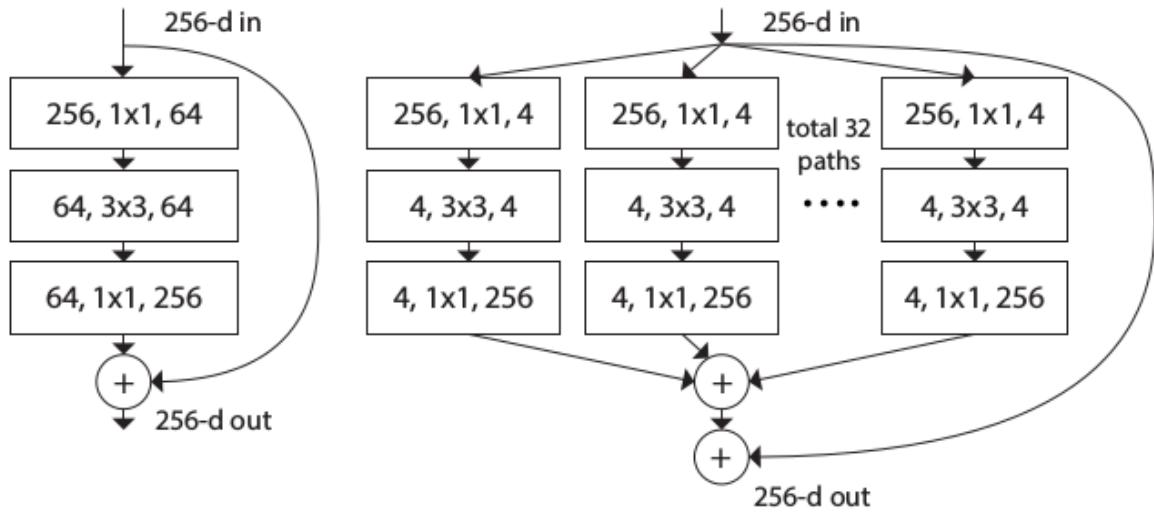
4. 在 ResNeXt 中，为了设计方便， $\mathcal{T}_i$  采取以下设计原则：

- 所有的  $\mathcal{T}_i$  具有相同的结构。这是参考了 VGG 的层叠相同结构的层的思想。
- $\mathcal{T}_i$  的结构通常是：
  - 第一层：执行  $1 \times 1$  的卷积来产生  $\vec{x}$  的一个低维嵌入
  - 第二层 ~ 倒数第二层：执行卷积、池化等等变换
  - 最后一层：执行  $1 \times 1$  的卷积来将结果提升到合适的维度

#### 5.4.2.2 ResNeXt 块

1. 一个 ResNeXt 模块执行了一组相同的“变换”，每一个“变换”都是输入的一个低维嵌入。变换的数量就是基数 C。

如下所示：左图为 ResNet 块；右图为 ResNeXt 块。



2. ResNeXt 模块有两种等效的形式：

- 图 (a) 为标准形式。
- 图 (b) 类似 Inception-ResNet 模块。这里的拼接是沿着输入通道数的方向拼接。
  - 它与 Inception-ResNet 模块的区别在于：这里每一条路径都是相同的。
  - 等效的原因是：输入通道数为 128 的  $1 \times 1$  卷积可以如下拆分：(设输入张量为  $\mathbf{I}$ ，输出张量为  $\mathbf{O}$ ，核张量为  $\mathbf{K}$ )

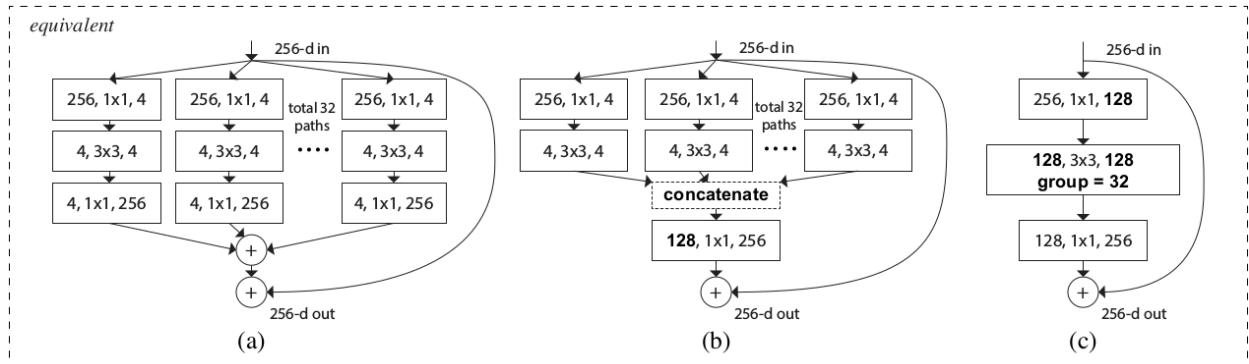
$$\begin{aligned}\mathbf{O}_{i,j,k} &= \sum_{s=1}^{128} \mathbf{I}_{s,j,k} \times \mathbf{K}_{i,s} \\ &= \left( \sum_{s=1}^4 \mathbf{I}_{s,j,k} \times \mathbf{K}_{i,s} \right) + \left( \sum_{s=5}^8 \mathbf{I}_{s,j,k} \times \mathbf{K}_{i,s} \right) + \cdots + \left( \sum_{s=124}^{128} \mathbf{I}_{s,j,k} \times \mathbf{K}_{i,s} \right)\end{aligned}$$

经过这种拆分，图 (b) 就等效于图 (a)。

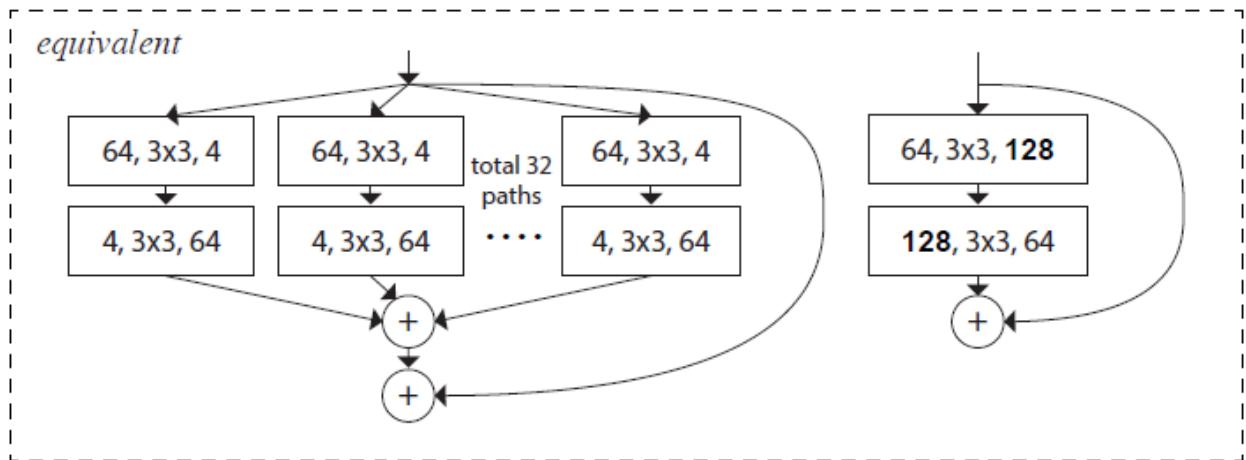
其中：

- $i$  表示输出单元位于  $i$  通道
- $l$  表示输入单元位于  $l$  通道
- $j, k$  表示通道中的坐标

图 (c) 是一个分组卷积的形式，它就是用分组卷积来实现图 (b)。它也是图 (b) 在代码中的实现方式。

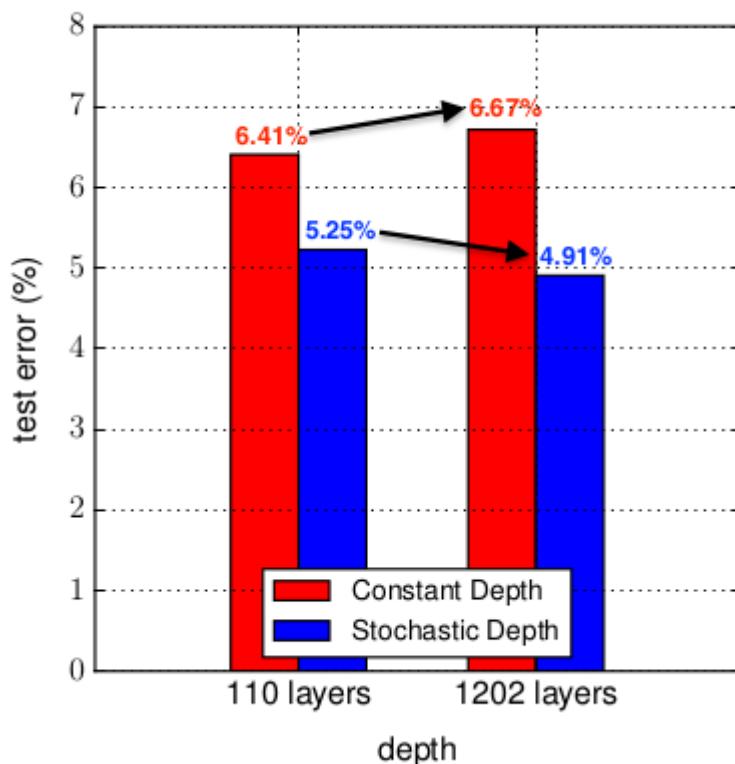


3. 通常 ResNeXt 模块至少有三层。事实上它也可以有两层，此时它等效于一个宽的、密集模块。



### 5.4.3 随机深度网络

- 随机深度网络的主要贡献是：提出了训练时随机丢弃网络层的思想，从而能够让网络深度增加到超过1000层，并仍然可以减少测试误差。
- 在 CIFAR-10 上，1202 层的 ResNet 测试误差要高于 110 层的 ResNet，表现出明显的过拟合。而 1202 层的随机深度网络（结合了 ResNet）的测试误差要低于 110 层的 ResNet。



#### 5.4.3.1 动机

- 神经网络的表达能力主要由网络深度来决定，但是过深的网络会带来三个问题：

- 反向传播过程中的梯度消失
- 前向传播过程中的 feature 消失
- 训练时间过长

虽然较浅的网络能够缓解这几个问题，但是较浅的网络表达能力不足，容易陷入欠拟合。

2. 随机深度网络解决这一矛盾的策略是：构建具有足够表达能力的深度神经网络（具有数百层甚至数千层），然后：

- 在网络训练期间，对每个 `mini batch` 随机地移除部分层来显著的减小网络的深度。

移除操作：删除对应的层，并用跳跃连接来代替。

- 在网络测试期间，使用全部的网络层。

3. 随机深度的思想可以和 `ResNet` 结合。因为 `ResNet` 已经包含了跳跃连接，因此可以直接修改。

#### 5.4.3.2 结构

1. 假设 `ResNet` 有  $L$  个残差块，则有：

$$\vec{\mathbf{H}}_l = \text{ReLU} \left( F(\vec{\mathbf{H}}_{l-1}, \mathbf{W}_l) + \vec{\mathbf{H}}_{l-1} \right), \quad l = 1, 2, \dots, L$$

其中：

- $\vec{\mathbf{H}}_l$  表示第  $l$  个残差块的输出
- $\vec{\mathbf{H}}_{l-1}$  为第  $l-1$  个残差块的输出，也是第  $l$  个残差块的输入
- $\mathbf{W}_l$  表示第  $l$  个残差块的权重
- $F$  为待学习的残差映射

2. 假设第  $l$  个残差块是否随机丢弃由伯努利随机变量  $b_l \in \{0, 1\}$  来指示。

- 当  $b_l = 0$  时，第  $l$  个残差块被丢弃。
- 当  $b_l = 1$  时，第  $l$  个残差块被保留。

因此有：

$$\vec{\mathbf{H}}_l = \text{ReLU} \left( b_l \times F(\vec{\mathbf{H}}_{l-1}, \mathbf{W}_l) + \vec{\mathbf{H}}_{l-1} \right), \quad l = 1, 2, \dots, L$$

3. 对随机变量  $b_l$ ，令：

$$P(b_l) = \begin{cases} p_l, & b_l = 1 \\ 1 - p_l, & b_l = 0 \end{cases}$$

其中  $p_l$  称做保留概率或者存活概率，它是一个非常重要的超参数。

4.  $p_l$  的选择有两个策略：

- 所有残差块的存活概率都相同： $p_1 = p_2 = \dots = p_{L-1} = p_L$
- 所有残差块的存活概率都不同，且根据残差块的深度进行线性衰减：

$$p_l = 1 - \frac{l}{L}(1 - p_L), \quad l = 1, 2, \dots, L$$

其背后的思想是：靠近输入的层提取的是被后续层使用的低级特征，因此更应该被保留下。

5. 给定第  $l$  个残差块的保留概率  $p_l$ ，则网络的深度为（以残差块数量为单位）：

$$\mathbb{E}(\tilde{L}) = \sum_{l=1}^L p_l$$

- 对于均匀存活概率： $\mathbb{E}(\tilde{L}) = p_L \times L$

- 对于线性衰减存活概率：

$$\mathbb{E}(\tilde{L}) = \frac{(1 + p_L) \times L - (1 - p_L)}{2}$$

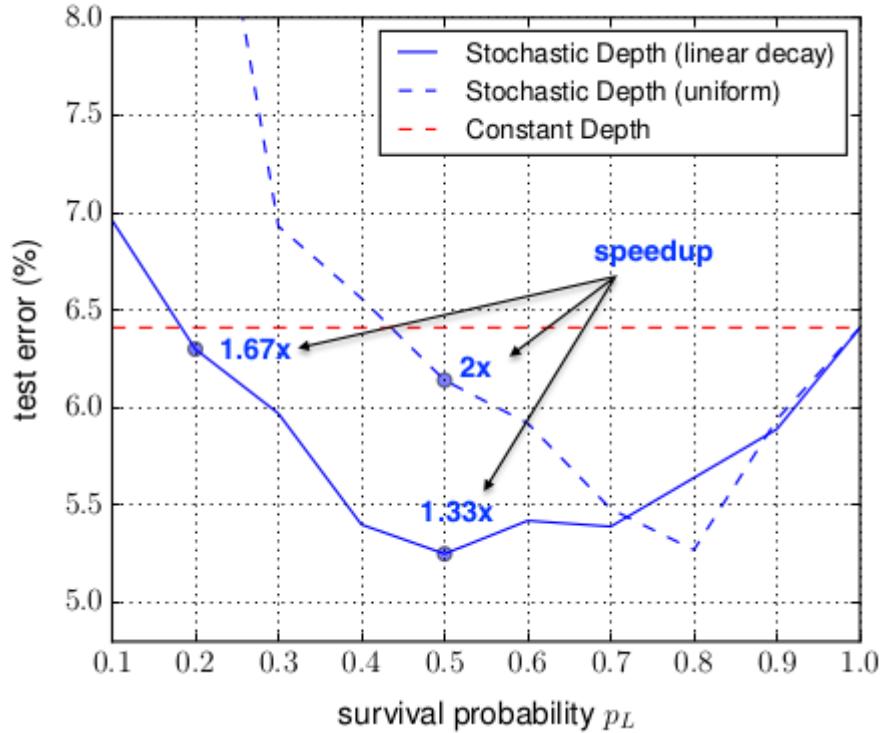
因此随机深度网络在训练时具有更短的期望深度，从而节省了训练时间。

6.  $p_l$  的选择策略，以及  $p_L$  的大小的选取需要根据实验仔细选择。

- 根据作者的实验结果，作者推荐使用线性衰减存活概率，并选择  $p_L = 0.5$ 。

$$\text{此时有: } \mathbb{E}(\tilde{L}) = \frac{(3L-1)}{4}$$

- 如果选择更小的  $p_L$  将会带来更大的测试误差，但是会更大的加速训练过程。



7. 测试时，需要调整残差块的输出：

$$\vec{\mathbf{H}}_l^{test} = \text{ReLU} \left( p_l \times F(\vec{\mathbf{H}}_{l-1}^{test}, \mathbf{W}_l) + \vec{\mathbf{H}}_{l-1}^{test} \right), \quad l = 1, 2, \dots, L$$

#### 5.4.3.3 性质

1. 随机深度网络能够大大减少训练时间和测试误差。
  - 训练时间减小是因为：网络训练时，期望深度减小。
  - 测试误差减小是因为：
    - 网络训练时期望深度的减少，使得梯度链变短，从而加强了反向传播期间靠近输入层的梯度。
    - 随机深度网络可以被理解为一系列不同深度的网络的隐式集成的集成模型。
2. 随机深度网络可以视作  $2^L$  个隐式的神经网络的集成。
  - 这些被集成的网络都是权重共享的。对每个 `mini batch`，只有其中之一得到了权重更新。
  - 在测试时，取所有被集成的网络的平均。
3. 随机深度和 `Dropout` 都可以被理解为一系列网络的隐式集成。
  - 随机深度集成了一系列具有不同深度的神经网络；而 `Dropout` 集成了一系列具有不同宽度的神经网络。
  - `Dropout` 与 `BN` 配合使用时会失效。而随机深度可以和 `BN` 配合使用。
4. 在随机深度网络中，由于训练时的随机深度，模型的测试误差的抖动相对于 `ResNet` 会偏大。

## 六、SENet

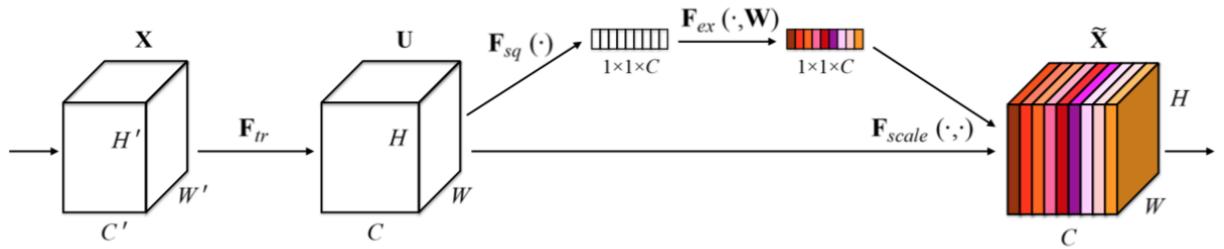
1. `SENet` 的主要贡献：提出了一种新的架构单元来解决通道之间相互依赖的问题。
2. `SENet` 以  $2.251\% \text{ top-5}$  的错误率获得了 `ILSVRC 2017` 分类比赛的冠军。
3. `SENet` 的目标：显式的建模卷积特征通道之间的相互依赖关系来提高网络的表达能力。  
做法是：显式地对通道之间的相互依赖关系建模，自适应的重新校准通道维的特征响应。
4. `SENet` 优势：可以直接与其他架构一起融合使用，只需要付出微小的计算成本就可以产生显著的性能提升。

## 6.1 SE 块

1. `SENet` 中提出的新的架构单元就是 `SE` 块 (`Squeeze-and-Excitation`)。它主要由 `squeeze` 操作和 `excitation` 操作组成。
2. 对于给定的任何变换  $\mathbf{F}_{tr} : \mathbf{X} \rightarrow \mathbf{U}$ ，其中：
  - $\mathbf{X} \in \mathbb{R}^{W' \times H' \times C'}$  为输入 `feature map`。它的尺寸为  $W' \times H'$ ，通道数为  $C'$ 。
  - $\mathbf{U} \in \mathbb{R}^{W \times H \times C}$  为输出 `feature map`。它的尺寸为  $W \times H$ ，通道数为  $C$ 。

可以构建一个相应的 `SE` 块来执行特征重新校准，如下所示：

- 特征  $\mathbf{U}$  首先执行 `squeeze` 操作
- 然后是一个 `excitation` 操作
- 然后特征映射  $\mathbf{U}$  被重新加权以生成 `SE` 块的输出
- 最后可以将输出传递给后续的层中



3. 假设  $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_C]$ 。  $\mathbf{u}_c$  为第  $c$  个通道，是一个  $W \times H$  的矩阵；

假设  $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{C'}]$ 。  $\mathbf{x}_j$  为第  $j$  个通道，是一个  $W' \times H'$  的矩阵。

需要学习的变换  $\mathbf{F}_{tr} = [\mathbf{V}_1, \mathbf{V}_2, \dots, \mathbf{V}_C]$  就是一组卷积核。  $\mathbf{V}_c$  为第  $c$  个卷积核，记做：

$\mathbf{V}_c = [\mathbf{v}_c^{(1)}, \mathbf{v}_c^{(2)}, \dots, \mathbf{v}_c^{(C')}]$ 。 $\mathbf{v}_c^{(j)}$  为第  $c$  个卷积核的第  $j$  通道，是一个二维矩阵。则：

$$\mathbf{u}_c = \mathbf{V}_c * \mathbf{X} = \sum_{j=1}^{C'} \mathbf{v}_c^{(j)} * \mathbf{x}_j$$

这里  $*$  表示卷积。同时为了描述简洁，这里忽略了偏置项。

4. 输出  $\mathbf{u}_c$  考虑了输入  $\mathbf{X}$  的所有通道，因此通道依赖性被隐式的嵌入到  $\mathbf{V}_c$  中。

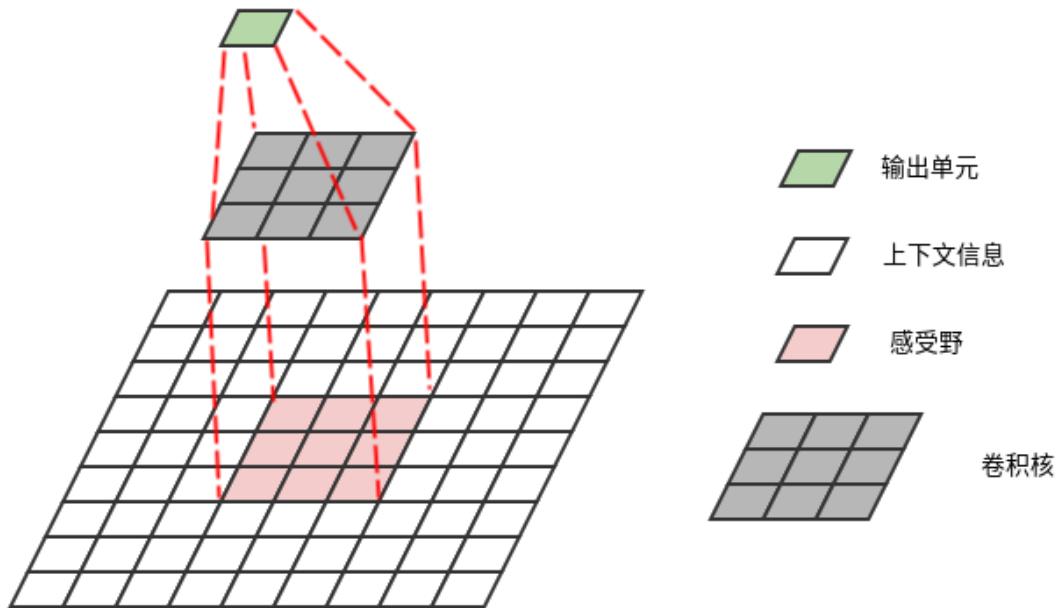
### 6.1.1 squeeze 操作

1. `squeeze` 操作的作用是：跨空间  $W \times H$  聚合特征来产生通道描述符。

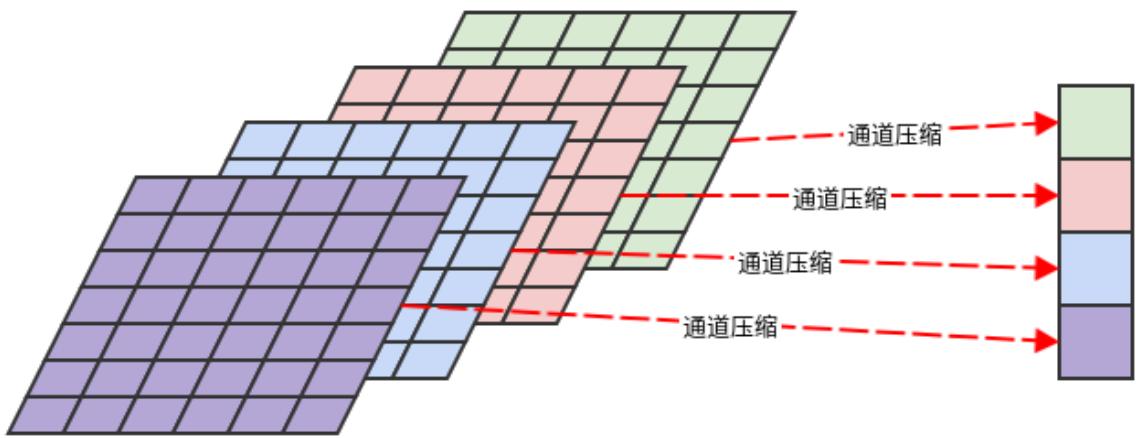
该描述符嵌入了通道维度特征响应的全局分布，包含了全局感受野的信息。

2. 每个学到的滤波器都是对局部感受野进行操作，因每个输出单元都无法利用局部感受野之外的上下文信息。

- 在网络的低层，其感受野尺寸很小，这个问题更严重。



- 为减轻这个问题，可以将全局空间信息压缩成一个通道描述符。然后利用该通道描述符。



3. 通常基于通道的全局平均池化来生成通道描述符（也可以考虑使用更复杂的聚合策略）。

设所有通道的通道描述符组成一个向量  $\vec{z} \in \mathbb{R}^C$ 。则有：

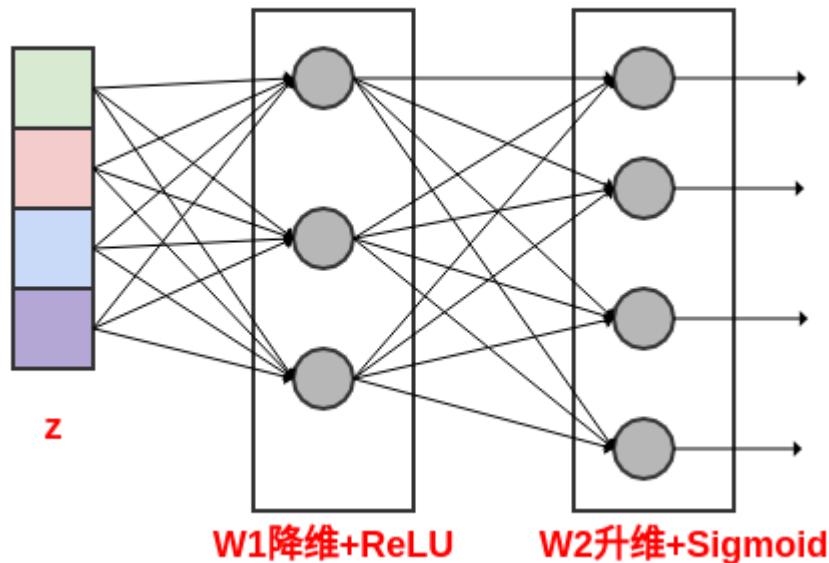
$$\vec{z} = [z_1, z_2, \dots, z_C]$$

$$z_c = \frac{1}{W \times H} \sum_{i=1}^W \sum_{j=1}^H u_c(i, j); \quad c = 1, 2, \dots, C$$

### 6.1.2 excitation 操作

1. `excitation` 操作的作用是：通过自门机制来学习每个通道的激活值，从而控制每个通道的权重。
2. `excitation` 操作利用了 `squeeze` 操作输出的通道描述符  $\vec{z}$ 。
  - 首先，通道描述符  $\vec{z}$  经过线性降维之后，通过一个 `ReLU` 激活函数。
  - 降维通过一个输出单元的数量为  $\frac{C}{r}$  的全连接层来实现。其中  $r$  为降维比例。

- 然后，ReLU 激活函数的输出经过线性升维之后，通过一个 sigmoid 激活函数。  
升维通过一个输出单元的数量为  $C$  的全连接层来实现。



3. 设 excitation 操作的输出为向量  $\vec{s}$ ，则有：

$$\vec{s} = \sigma(\mathbf{W}_2 \delta(\mathbf{W}_1 \vec{z}))$$

其中：

- $\sigma$  为 sigmoid 激活函数
- $\delta$  为 ReLU 激活函数
- $\mathbf{W}_1 \in \mathbb{R}^{C/r \times C}$  为降维层的权重
- $\mathbf{W}_2 \in \mathbb{R}^{C \times C/r}$  为升维层的权重
- $r$  为降维比例

4. 在经过 excitation 操作之后，块的最终输出通过重新调节  $\mathbf{U}$  得到。

设块的最终输出为  $\tilde{\mathbf{X}} = [\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_C]$ ，则有：

$$\tilde{x}_c = s_c \times u_c, \quad c = 1, 2, \dots, C$$

这里  $s_c$  为 excitation 操作的输出结果，它作为通道  $c$  的权重。

$s_c$  不仅考虑了本通道的全局信息（由  $z_c$  引入），还考虑了其它通道的全局信息（由  $\mathbf{W}_1, \mathbf{W}_2$  引入）。

## 6.2 性质

1. SE 块可以理解为注意力机制的一个应用。

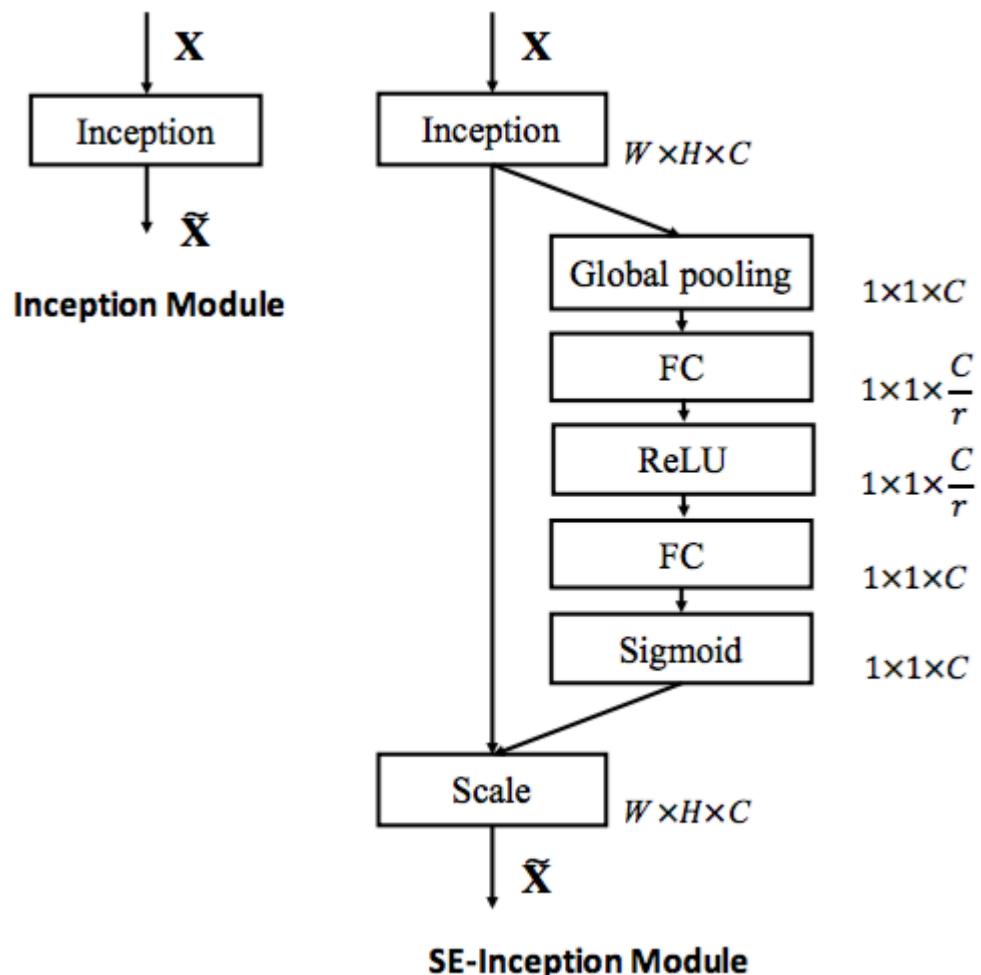
它是一个轻量级的门机制，用于对通道关系进行建模。

通过该机制，网络学习全局信息来选择性的强调部分特征，并抑制其它特征。

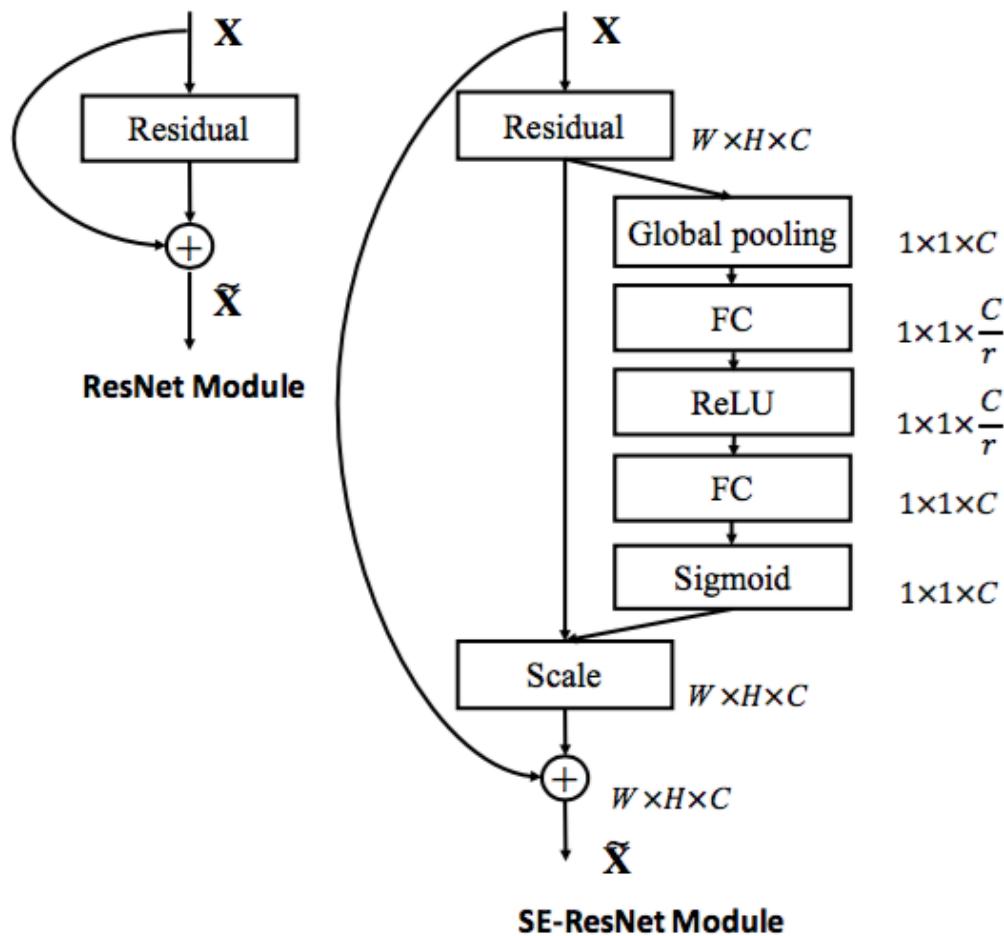
2. SE 网络构建方式：

- 简单的堆叠 SE 块来构建一个新的网络。
- 在现有网络架构中，用 SE 块来替代原始块。

■ 下图中，左侧为原始 Inception 模块；右侧为 SE-Inception 模块。



- 下图中，左侧为原始残差模块；右侧为 SE-ResNet 模块。



3. 在 **SENet** 中，所有额外的参数都包含在门机制的两个全连接层中。

引入的额外参数的数量为： $\frac{2}{r} \sum_{s=1}^S C_s^2$ 。其中：

- $r$  表示降维比例。论文中设定为16。
- $S$  指的是 **SE** 块的数量。
- $C_s$  表示第  $s$  个 **SE** 块的输出通道的维度。

**SE-ResNet-50** 在 **ResNet-50** 所要求的大约2500万参数之外，额外引入了约250万参数，相对增加了10%。

4. 超参数  $r$  称作减少比率。它刻画了需要将通道描述符组成的向量压缩的比例。

- 它是一个重要的超参数，需要在精度和复杂度之间平衡。
- 网络的精度并不会随着  $r$  的增加而单调上升。因此需要多次实验并选取其中最好的那个值。

5. 虽然 **SE** 块可以应用在网络的任何地方，但是它在不同深度中承担了不同的作用。

- 在网络较低的层中：对于不同类别的样本，特征通道的权重分布几乎相同。

这说明在网络的最初阶段，特征通道的重要性很可能由不同的类别共享。即：低层特征通常更具有普遍性。

- 在网络较高的层中：对于不同类别的样本，特征通道的权重分布开始分化。

这说明在网络的高层，每个通道的值变得更具有类别特异性。即：高层特征通常更具有特异性。

- 在网络的最后阶段： **SE** 块为网络提供重新校准所起到的作用，相对于前面的 **SE** 块来说，更加不重要。

这意味着可以删除最后一个阶段的 `SE` 块，从而显著减少总体参数数量，仅仅带来一点点的损失。（对于 `ImageNet`, `top-1` 错误率的损失小于 `0.1%`）

因此：`Se` 块执行特征通道重新校准的好处可以通过整个网络进行累积。

## 七、DenseNet

1. `DenseNet` 的主要贡献在于：不是通过更深或者更宽的结构，而是通过特征重用来提升网络的学习能力。

2. `DenseNet` 从 `ResNet` 中获得的灵感：创建从“靠近输入的层”到“靠近输出的层”的直连。

而 `DenseNet` 做得更为彻底：将所有层以前馈的形式相连。这种网络因此称作 `DenseNet`。

3. `DenseNet` 的参数数量和计算量相对 `ResNet` 明显减少。

具有 `20M` 个参数的 `DenseNet-201` 与具有 `40M` 个参数的 `ResNet-101` 验证误差接近。

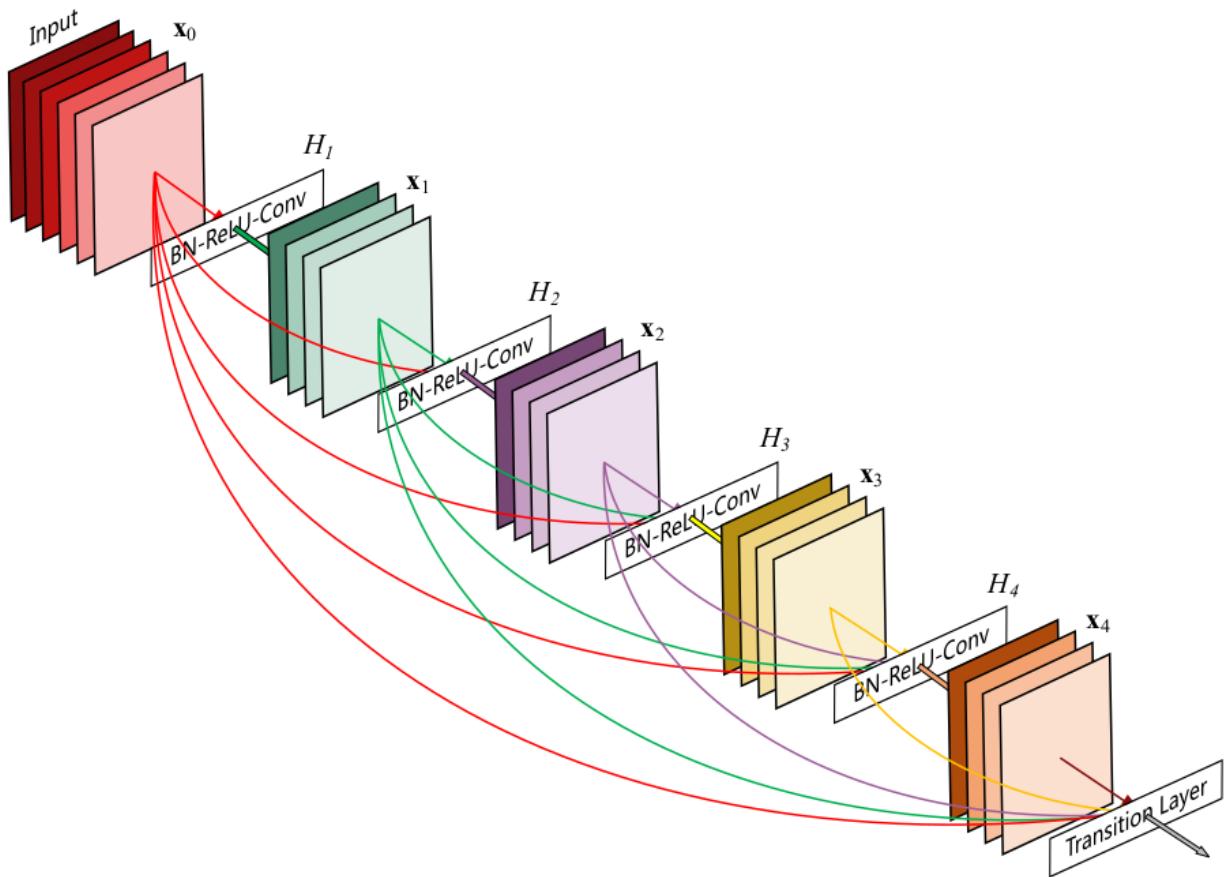
4. `DenseNet` 具有以下的优点：

- 缓解梯度消失的问题。因为每层都可以直接从损失函数和原始输入中获取梯度，从而易于训练。
- 密集连接还具有正则化的效应，缓解了小训练集任务的过拟合。
- 鼓励特征重用。
- 大幅度减少参数数量。因为每层的卷积核尺寸都比较小，输出通道数较少（由增长率  $k$  决定）。

### 7.1 DenseNet 块

1. 一个 `DenseNet` 网络具有多个 `DenseNet` 块，`DenseNet` 块之间由过渡层连接。

- 具有  $L$  层的传统卷积网络有  $L$  个连接，每层仅仅与后继层相连。
- 具有  $L$  层的 `DenseNet` 块有  $\frac{L(L+1)}{2}$  个连接，每层以前馈的方式将该层与它后面的所有层相连。对于第  $l$  层：
  - 所有先前层的 `feature map` 都作为本层的输入。第  $l$  层具有  $l$  个输入 `feature map`。
  - 本层输出的 `feature map` 都将作为后面  $L - l$  层的输入。



2. 假设 `DenseNet` 块包含  $L$  层，每一层都实现了一个非线性变换  $H_l(\cdot)$ ，其中  $l$  表示层的索引。

$H_l(\cdot)$  可以是包含了 `Batch Normalization(BN)`、`ReLU` 单元、池化或者卷积等操作的复合函数。

假设 `DenseNet` 块的输入为  $x_0$ ，`DenseNet` 块的第  $l$  层的输出为  $x_l$ 。

则有：

$$x_l = H_l([x_0, x_1, \dots, x_{l-1}])$$

其中  $[x_0, x_1, \dots, x_{l-1}]$  表示  $0, \dots, l-1$  层输出的 `feature map` 沿着通道数的拼接。

相比之下，`ResNet` 块中，不同 `feature map` 是通过直接相加来作为块的输出。

3. `DenseNet` 块中，每层的  $H_l(\cdot)$  输出的 `feature map` 通道数都相同，都是  $k$  个。

- $k$  是一个重要的超参数，称作网络的增长率。

事实上你也可以为每一层设置不同的增长率。但是为了设计简单，这里设置了一个统一值。

- 第  $l$  层的输入 `feature map` 的通道数为： $k_0 + k \times (l - 1)$ 。

其中  $k_0$  为输入层的通道数。

4. `DenseNet` 的  $H_l(\cdot)$  有两种类型：

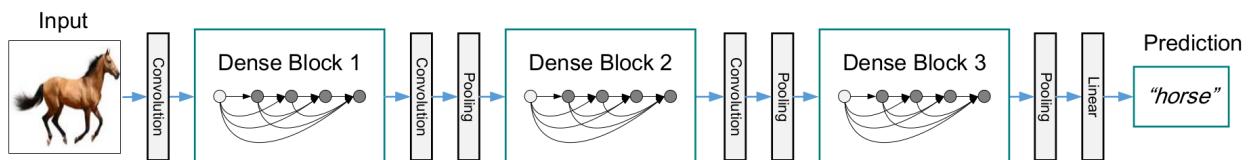
- 以下的三个连续运算的符复合函数：先执行 `BN`，再执行 `ReLU`，最后接一个  $3 \times 3$  的卷积。
- 在  $3 \times 3$  卷积之前放置了  $1 \times 1$  卷积层：先执行 `BN`，再执行 `ReLU`，再接一个  $1 \times 1$  的卷积，再执行 `BN`，再执行 `ReLU`，最后接一个  $3 \times 3$  的卷积。该版本的 `DenseNet` 称作 `DenseNet-B`。

其中： $1 \times 1$  卷积层输出 `feature map` 的通道数为  $4k$ 。

## 7.2 过渡层

1. `DenseNet` 块之间的层称为过渡层。其主要作用是连接不同的 `DenseNet` 块。
2. 过渡层可以包含卷积或池化操作，从而改变前一个 `DenseNet` 块的输出 `feature map` 的大小（包括尺寸大小、通道数量）。

论文中的过渡层由一个 `BN` 层、一个 `1x1` 卷积层、一个 `2x2` 平均池化层组成。



3. 为了进一步提高模型的紧凑性，可以在过渡层减少 `feature map` 的通道数。

如果不减少 `DenseNet` 块的输出通道数，则经过了  $N$  个 `DenseNet` 块之后，网络的 `feature map` 的通道数为：

$$k_0 + N \times k \times (L - 1)$$

其中  $k_0$  为输入图片的通道数， $L$  为 `DenseNet` 块的层数。

4. 如果 `Dense` 块输出 `feature map` 的通道数为  $m$ ，则可以使得过渡层输出 `feature map` 的通道数为  $\theta m$ 。其中  $0 < \theta \leq 1$  为压缩因子。

- 当  $\theta = 1$  时，经过过渡层的 `feature map` 通道数不变。
- 当  $\theta < 1$  时，经过过渡层的 `feature map` 通道数减小。

此时的 `DenseNet` 称做 `DenseNet-C`。

结合了 `DenseNet-C` 和 `DenseNet-B` 的改进的网络称作 `DenseNet-BC`。

## 7.3 性质

1. `DenseNet` 块中，每层都可以访问块内的所有早前层输出的 `feature map`。这些 `feature map` 可以视作 `DenseNet` 块的全局状态。

- 每层输出的 `feature map` 都将被添加到块的这个全局状态中。

增长率为  $k$  决定了新增特征占全局状态的比例。

- 该全局状态可以理解为网络块的“集体知识”，由块内所有层共享。

因此 `feature map` 无需逐层复制（因为它是全局共享）。这也是 `DenseNet` 与传统网络结构不同的地方。

这有助于整个网络的特征重用，并产生更紧凑的模型。

2. `ResNet` 通过恒等映射直连，明确的传递了“保留”的信息，因此网络只需要传递“变动”的信息。

`DenseNet` 也明确区分“变动”信息和“保留”的信息，将“变动”的信息（当前层输出的 `feature map`）添加到“集体知识”。最终分类器基于 `DenseNet` 块的所有 `feature map`。

3. 当  $L$  和  $k$  增加时，`DenseNet` 表现更好，这主要是因为模型容量相应地增长。

网络可以利用更大、更深的模型提高其表达学习能力，这也表明了 `DenseNet` 不会受到优化困难的影响。

4. `DenseNet` 提高精度的一个可能的解释是：各层通过较短的连接（最多需要经过两个或者三个过渡层）直接从损失函数中接收额外的监督信息。

5. `DenseNet` 的增长率  $k$  可以非常小，一个相对较小的  $k$  就可以达到很好的结果，如  $k = 12$ 。

而传统的网络结构中，`feature map` 的通道数一般都较大。因此 `DenseNet` 具有比传统网络更少的参数。

## 八、小型网络

1. 虽然卷积神经网络在图像识别领域超越了人类的表现，但是这些先进的网络需要较高的计算资源。这些资源需求超出了很多移动设备和嵌入式设备的能力。
2. 小型网络的设计和优化目标并不是模型的准确率，而是在满足一定准确率的条件下，尽可能的使得模型小，从而降低对计算资源的需求，降低计算延迟。
3. 小型网络的一个重要应用是为了满足移动设备和嵌入式设备的视觉应用。
4. 小型高效的神经网络的构建方法大致可以归为两类：
  - 对已经训练好的网络进行压缩
  - 直接训练小网络模型

### 8.1 MobileNet

1. `MobileNet` 的主要贡献是：应用了 `Depthwise` 深度可分离卷积来代替常规卷积，从而降低计算量，减少模型参数。
2. `MobileNet` 不仅产生了小型网络，还重点优化了延迟。

与之相比，有一些小型网络虽然网络参数较少，但是预测的延迟较大。
3. `MobileNet` 和 `VGG16` 准确率相近，但是模型大小是后者的  $\frac{1}{26.9}$ ，计算复杂度是后者的  $\frac{1}{34.3}$ 。

`MobileNet` 比 `GoogleNet` 准确率更高，模型大小是后者的  $\frac{1}{2.7}$ ，计算复杂度是后者的  $\frac{1}{1.6}$ 。

如果考虑超参数宽度乘子和分辨率乘子，`MobileNet` 还可以进一步压缩。

#### 8.1.1 动机

1. 对于传统的卷积层，单个输出 `feature` 这样产生：
  - 首先由一组滤波器对输入的各通道执行滤波，生成滤波 `feature`。它仅仅考虑空间相关性。
  - 然后计算各通道的滤波 `feature` 的加权和，得到单个 `feature`。它在空间相关性的基础上，叠加了通道相关性。

其中不同空间位置时，滤波 `feature` 的加权和采用的权重不同。
2. `Depthwise` 深度可分离卷积打破了空间相关性和通道相关性的混合。
  - 首先由一组滤波器对输入的各通道执行滤波，生成滤波 `feature`。它仅仅考虑空间相关性。
  - 然后执行  $1 \times 1$  卷积来组合不同滤波 `feature`。它仅仅考虑通道相关性。

其中不同空间位置时，组合不同滤波 `feature` 采用的权重相同。

这也是为什么这一步仅仅考虑通道相关性。
3. 假设：
  - 输入 `feature map` 是一个尺寸为  $D_F \times D_F$ 、输入通道数为  $M$  的张量  $\mathbf{F}$ 。
  - 输出 `feature map` 是一个尺寸为  $D_K \times D_K$ 、输出通道数为  $N$  的张量  $\mathbf{G}$ 。
  - 假设核张量为  $\mathbf{K}$ ，其形状为  $M \times N \times D_K \times D_K$ 。

则对于标准卷积过程，有：

$$G_{j,m,n} = \sum_i \sum_{m'} \sum_{n'} K_{i,j,m',n'} \times F_{i,m+m',n+n'}$$

其中：

- $i$  为输入通道的索引
- $j$  为输出通道的索引
- $m, n$  为空间索引，  $m', n'$  分别为对应空间方向上的遍历变量

其计算代价为： $D_K \times D_K \times M \times N \times D_F \times D_F$ 。

4. `depthwise` 深度可分离卷积中，假设核张量为  $\mathbf{K}$ ，其形状为： $M \times D_K \times D_K$ 。

则对于 `depthwise` 深度可分离卷积过程：

- 首先对输入的各通道执行滤波，有：

$$\hat{G}_{i,m,n} = \sum_{m'} \sum_{n'} K_{i,m',n'} \times F_{i,m+m',n+n'}$$

其中：

- $i$  为输入通道的索引
- $m, n$  为空间索引，  $m', n'$  分别为对应空间方向上的遍历变量

其计算代价为： $D_K \times D_K \times M \times D_F \times D_F$ 。

- 然后对每个通道得到的滤波 `feature` 执行 `1x1` 卷积，有：

$$G_{j,m,n} = \sum_i \tilde{K}_{i,j} \times \hat{G}_{i,m,n}$$

其中：

- $\tilde{\mathbf{K}}$  为 `1x1` 卷积的核张量
- $i$  为输入通道的索引
- $j$  为输出通道的索引
- $m, n$  为空间索引
- 总的计算代价为： $D_K \times D_K \times M \times D_F \times D_F + D_F \times D_F \times M \times N$ 。

因此 `depthwise` 深度可分离卷积的计算量占标准卷积计算量的比例为：

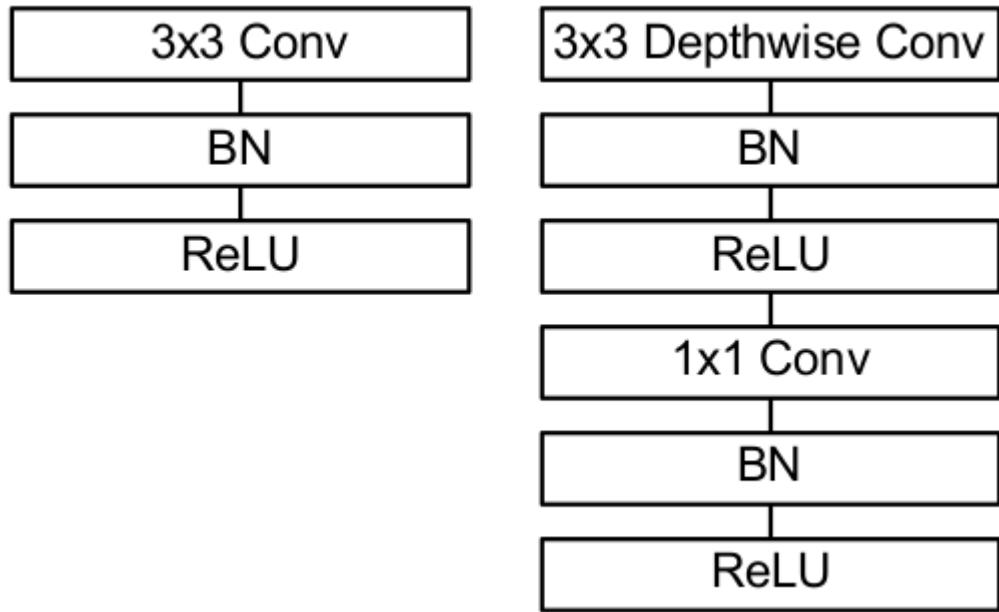
$$\frac{D_K \times D_K \times M \times D_F \times D_F + D_F \times D_F \times M \times N}{D_K \times D_K \times M \times N \times D_F \times D_F} = \frac{1}{N} + \frac{1}{D_K^2}$$

通常卷积核采用 `3x3` 卷积，而  $N \gg 9$ ，因此 `depthwise` 卷积的计算代价是常规卷积的  $\frac{1}{8}$  到  $\frac{1}{9}$ 。

## 8.1.2 网络结构

1. 常规卷积和 `Depthwise` 可分离卷积的结构区别（带 `BN` 和 `ReLU`）：

左图为常规卷积，右图为 `Depthwise` 可分离卷积。



2. MobileNet 网络结构如下表所示。其中：

- 第一层是全卷积。
- 所有层之后都是 BN 和 ReLU，但是最后的全连接层例外。

最后一个全连接层没有 ReLU 非线性激活函数，而是直接送到 softmax 层进行分类。

Table 1. MobileNet Body Architecture

| 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× Conv dw / s1 | $3 \times 3 \times 512$ dw           | $14 \times 14 \times 512$  |
|                 | $1 \times 1 \times 512 \times 512$   | $14 \times 14 \times 512$  |
| 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 $7 \times 7$                    | $7 \times 7 \times 1024$   |
| FC / s1         | $1024 \times 1000$                   | $1 \times 1 \times 1024$   |
| Softmax / s1    | Classifier                           | $1 \times 1 \times 1000$   |

### 8.1.3 超参数

- 尽管基本的 MobileNet 架构已经很小，延迟很低，但特定应用需要更快的模型。
- MobileNet 引入了两个超参数：
  - 宽度乘子 width multiplier，记做  $\alpha$ 。
    - 宽度乘子的作用是：在每层均匀的缩减网络。
    - 宽度乘子应用于第一层是全卷积的输出通道数上。

这也影响了后续所有 Depthwise 可分离卷积层的输入 feature map 通道数、输出 feature map 通道数。
  - 分辨率乘子 resolution multiplier，记做  $\rho$ 。

■ 分辨率乘子的作用是：降低输出的 `feature map` 的尺寸。

■ 分辨率乘子应用于输入图片上，改变了输入图片的尺寸。

这也影响了后续所有 `Depthwise` 可分离卷积层的输入 `feature map` 尺寸、输出 `feature map` 尺寸。

3. 设宽度乘子为  $\alpha$ ，则对一个给定的 `Depthwise` 可分离卷积层：

- 输入通道的数量从  $M$  变成  $\alpha M$
- 输出通道的数量从  $N$  变成  $\alpha N$

则计算复杂度为： $D_K \times D_K \times \alpha M \times D_F \times D_F + D_F \times D_F \times \alpha M \times \alpha N$ 。因此模型大概以  $\alpha^2$  的速度减少计算复杂度和参数数量。

4.  $\alpha = 1$  表示基准 `MobileNet`。随着  $\alpha$  降低，模型的准确率一直下降。

| <b>with multiplier</b> | <b>ImageNet 准确率</b> | <b>乘-加 操作 (单位: 百万)</b> | <b>参数数量 (单位: 百万)</b> |
|------------------------|---------------------|------------------------|----------------------|
| 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                  |

5. 设分辨率乘子为  $\rho$ ，则对一个给定的 `Depthwise` 可分离卷积层，输入 `feature map` 的尺寸从  $D_F \times D_F$  变成  $\rho D_F \times \rho D_F$ 。

则计算复杂度为： $D_K \times D_K \times M \times \rho D_F \times \rho D_F + \rho D_F \times \rho D_F \times M \times N$ 。因此模型大概以  $\alpha^2$  的速度减少计算复杂度。

由于分辨率乘子并未改变输入通道数、输出通道数、卷积核大小，因此该超参数并不会改变模型的参数数量。

6.  $\rho = 1$  表示基准 `MobileNet`。随着  $\rho$  的降低，模型的准确率一直下降。

| <b>resolution</b> | <b>ImageNet 准确率</b> | <b>乘-加 操作 (单位: 百万)</b> | <b>参数数量 (单位: 百万)</b> |
|-------------------|---------------------|------------------------|----------------------|
| 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                  |

7. 如果同时应用宽度乘子和分辨率乘子，则对一个给定的 `Depthwise` 可分离卷积层，计算复杂度为：

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

## 8.2 MobileNet V2

1. MobileNet V2 的主要贡献是：创新性的提出了具有线性 bottleneck 的 Inverted 残差块。

这种块特别适用于移动设备和嵌入式设备，因为它用到的张量都较小，因此减少了推断期间的内存需求。

2. 在 ImageNet 分类任务上，MobileNet V2 相比 MobileNet V1 准确率更高，但是计算量和推断时间大幅降低。

| 网络               | Top 1 | Params (百万) | 乘-加 数量 (百万) | CPU   |
|------------------|-------|-------------|-------------|-------|
| MobileNet V1     | 70.6  | 4.2         | 575         | 113ms |
| ShuffleNet (1.5) | 71.5  | 3.4         | 292         | -     |
| ShuffleNet (x2)  | 73.7  | 5.4         | 524         | -     |
| NasNet-A         | 74.0  | 5.3         | 564         | 183ms |
| MobileNet V2     | 72.0  | 3.4         | 300         | 143ms |

### 8.2.1 线性bottleneck

1. 一直以来，人们认为与任务相关的信息是嵌入到 feature map 中的一个低维子空间。因此 feature map 事实上有一定的信息冗余。

- 如果缩减 feature map 的通道数量，则可以降低计算量。
- MobileNet V1 就是采用宽度乘子，从而在计算复杂度和准确率之间平衡。

2. 由于 ReLU 非线性激活的存在，缩减的输入 feature map 的通道数量可能会产生不良的影响。

考虑输入 feature map 中的某个通道，设该通道的数据为  $x$ 。经过线性变换  $B$  之后，再经过一个 ReLU 之后，其输出为  $x' = \text{ReLU}(Bx)$ 。

- 如果  $Bx$  的每个分量均大于 0，则输入  $x$  的所有信息都得到保留。
- 如果  $Bx$  在某些分量上小于 0，则输入  $x$  的部分信息被丢失。

当输入 feature map 的通道数较小时，会有相对较高的概率使得经过 ReLU 之后，输出 feature map 的某个维度全为 0。这使得输出 feature map 的有效维度降低了，从而降低了模型的表征能力。

最重要的是：这一过程是不可逆的，无法恢复。

因此 ReLU 会对通道数较少的输入 feature map 造成较大的损耗。

3. 当输入 feature map 的通道数较大时，即使经过 ReLU 之后输出 feature map 的某个维度全为 0，信息仍然可能保留在输出的其它维度中。

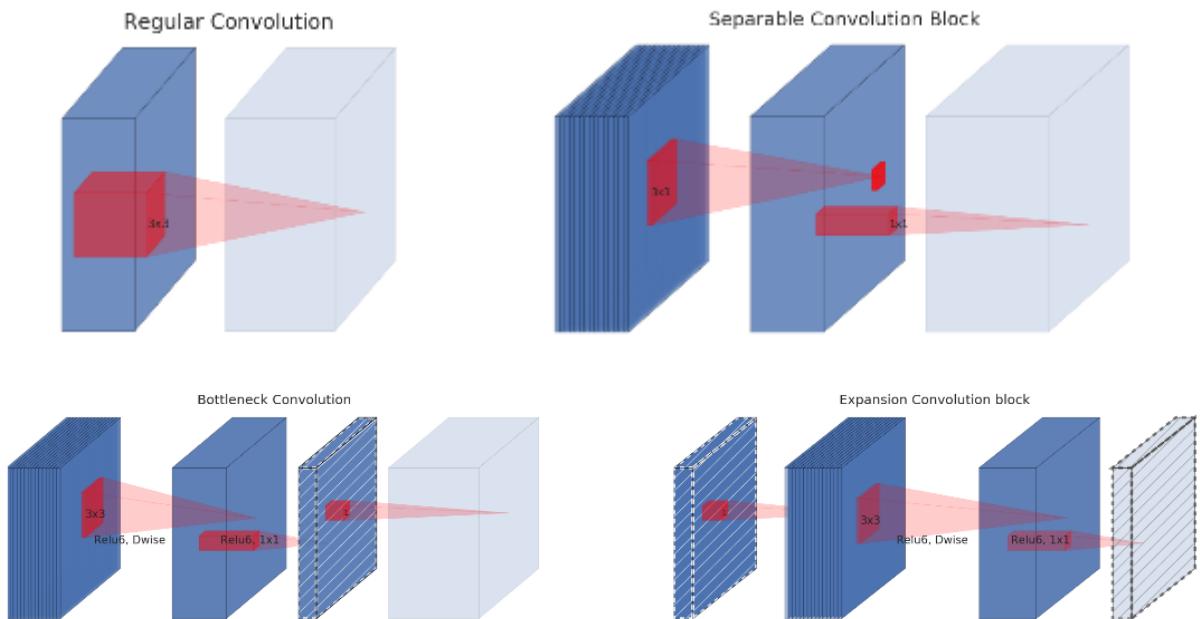
4. 线性 bottleneck 可以解决这一问题。

线性 bottleneck 是不带 ReLU 的  $1 \times 1$  卷积。可以通过线性 bottleneck 先将输入 feature map 通道数扩张到足够大，再通过 ReLU。这可以降低 ReLU 的信息损失。

线性 bottleneck 输出通道数与输入通道数的比例称作膨胀比。

如下图所示：（阴影块表示重复前面的结构，斜线块表示不包含非线性）

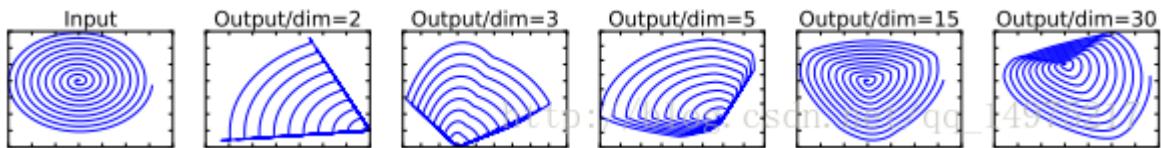
- 左上图：为常规的  $3 \times 3$  卷积。
- 右上图：为深度可分离卷积。
- 左下图和右下图：为带线性 **bottleneck** 的深度可分离卷积：在 **depthwise** 卷积之前插入一个不带 **ReLU** 的  $1 \times 1$  的卷积层。
  - 它们都是同一个结构，只是块的起始部分选择在不同地方而已。
  - **ReLU6** 表示  $f(x) = \max(0, 6, x)$ ，它将  $x$  映射到  $[0, 6]$  之间。



5. 下图中，**Input** 为输入 **feature map** 中的某个通道。

先将 **Input** 通过  $1 \times 1$  卷积映射成  $n$  个通道（膨胀比为  $n$ ），通过 **ReLU** 后再通过  $1 \times 1$  卷积映射回单通道。

可以看到：当  $n$  较小时，恢复后的张量坍塌严重； $n$  较大时信号恢复得较好。



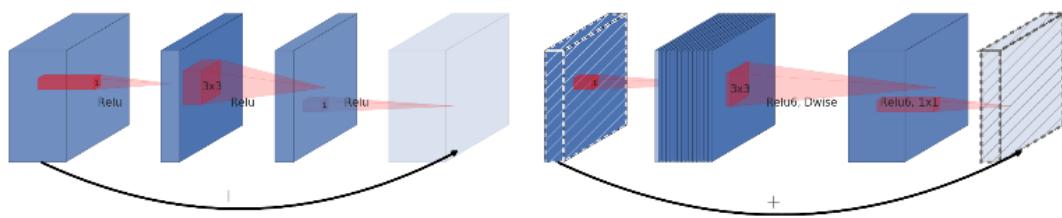
6. 实验表明：**bottleneck** 中使用线性是非常重要的。

虽然引入非线性会提升模型的表达能力，但是引入非线性会破坏太多信息，会引起准确率的下降。

### 8.2.2 Inverted 残差块

1. **Inverted** 残差块的结构类似 **ResNet** 残差块。它的输入 **feature map** 首先经过线性 **bottleneck** 来扩张通道数，然后经过深度可分离卷积。

## (a) Residual block      (b) Inverted residual block



2. 在 ResNeXt 残差块中，首先对输入 feature map 执行  $1 \times 1$  卷积来压缩通道数，最后通过  $1 \times 1$  卷积来恢复通道数。这对应了一个输入 feature map 通道数先压缩、后扩张的过程。

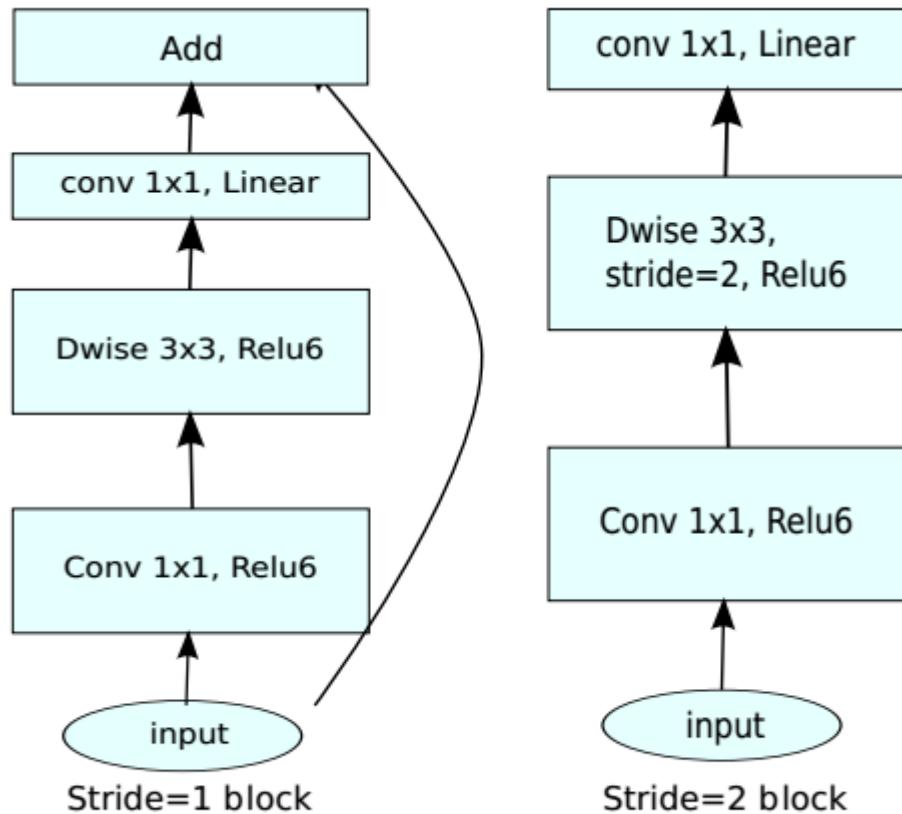
在 Inverted 残差块中，首先对输入 feature map 执行  $1 \times 1$  卷积来扩张通道数，最后通过  $1 \times 1$  卷积来恢复通道数。这对应了一个输入 feature map 通道数先扩张、后压缩的过程。

这也是 Inverted 残差块取名为 Inverted 的原因。

3. 当深度可分离卷积的步长为 1 时，Inverted 残差块包含了跳跃连接。

当深度可分离卷积的步长不为 1 时，Inverted 残差块 不包含跳跃连接，因此也没有残差块。这是因为：

- 输入 feature map 的通道数与残差块输出 feature map 的通道数不相同，二者无法拼接。
- 如果像 ResNet 一样，对跳跃连接增加线性投影，则会增加计算量。这对于移动设备或者嵌入式设备来说，可能无法满足。



### 8.2.3 网络结构

1. 网络结构如下。其中：

- $t$  表示膨胀比。  
通常较小的网络使用略小的膨胀比效果更好；较大的网络使用略大的膨胀比效果更好。
- $c$  表示通道数量
- $n$  表示块的重复数量
- $s$  表示卷积步长。跨层直连只有在步长为 1 时才有。

| 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    | -   | -   |

2. 与 MobileNet V1 类似，MobileNet V2 也可以引入宽度乘子、分辨率乘子这两个超参数。

## 8.3 ShuffleNet

1. ShuffleNet 的主要贡献是：提出了逐点分组卷积+通道混洗的策略，在保证准确率的同时大幅降低计算成本。

2. ShuffleNet 网络专为计算能力有限的设备（如：10~150MFLOPs）设计。

在基于 ARM 的移动设备上，ShuffleNet 与 AlexNet 相比，在保持相当的准确率的同时，大约 13 倍的加速。

### 8.3.1 动机

1. 在 Xception 和 ResNeXt 中，因为它们有大量的昂贵的  $1 \times 1$  卷积，所以计算效率较低。

如 ResNeXt 的每个残差块中， $1 \times 1$  卷积占据了乘-加运算的 93.4%（基数为 32 时）。

2. 在小型网络中，为了满足计算性能的约束（因为计算资源不够），此时需要限制通道数量。这可能会严重降低准确率。

解决思路是：对  $1 \times 1$  卷积应用分组卷积，将每个  $1 \times 1$  卷积仅仅在相应的通道分组上操作。这样就可以扩充通道数量。

3.  $1 \times 1$  卷积仅在相应的通道分组上操作会带来一个副作用：任意通道的输出，仅仅与该通道所在分组的输入（一般占总输入的比例较小）有关，与其它分组的输入（一般占总输入的比例较大）无关。

这会阻止通道之间的信息流动，降低网络的表达能力。

解决思路是：采用通道混洗，允许分组卷积从不同分组中获取输入。

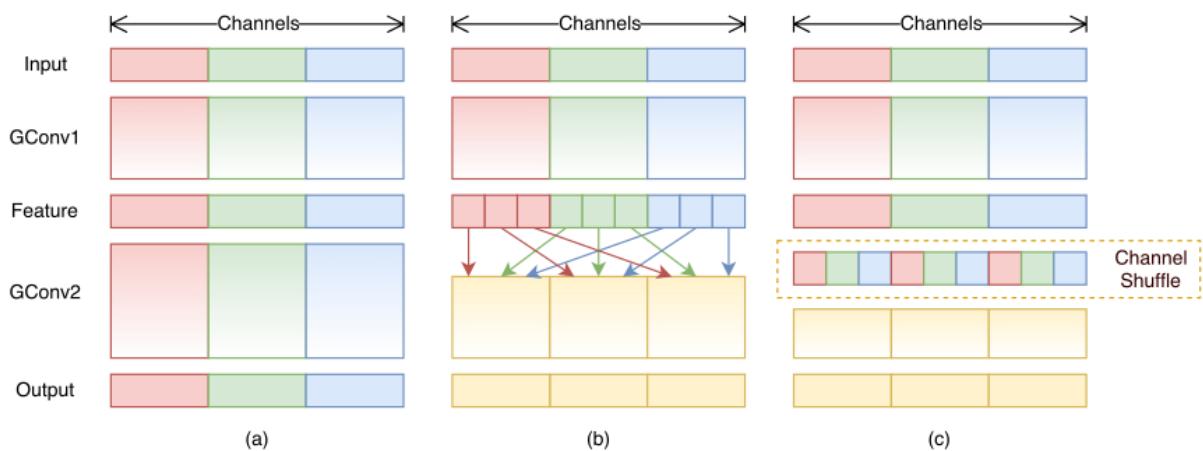
4. ShuffleNet 的解决  $1 \times 1$  卷积计算代价较高的方案：用分组卷积来代替传统的  $1 \times 1$  卷积，通过通道混洗来克服分组卷积的副作用。

给定输入 feature map：

- 首先对输入通道进行分组
- 然后在每个分组中执行  $1 \times 1$  卷积
- 然后执行通道混洗

5. 如下图所示：

- (a)：没有通道混洗的分组卷积
- (b)：通道混洗的分组卷积
- (c)：(b) 的等效表示

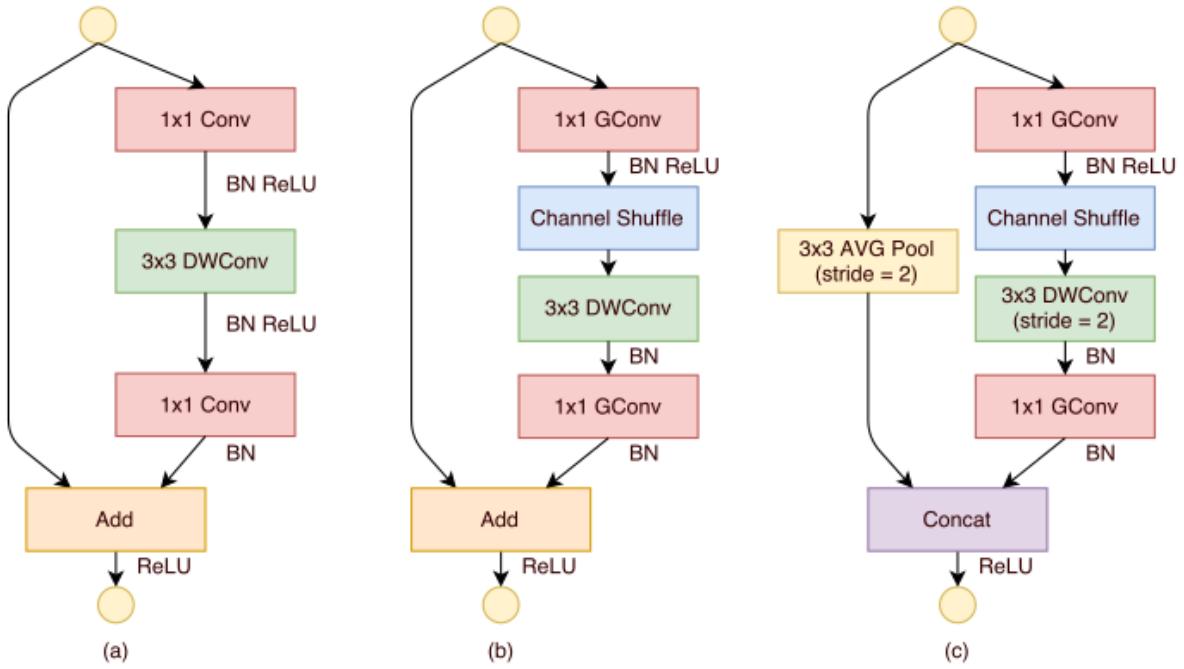


6. 由于通道混洗是可微的，因此它可以嵌入到网络中以进行端到端的训练。

### 8.3.2 结构

1. 下图为 ShuffleNet 块与 ResNeXt 块的区别：

- (a)：一个 ResNeXt 块，其中  $3 \times 3$  卷积采取的是 depthwise 卷积。
- (b)：一个 ShuffleNet 块。
- (c)：一个步长为 2 的 ShuffleNet 块。



## 2. ShuffleNet 块中：

- 第一个 **1x1** 卷积替换为： **1x1** 分组卷积+通道随机混洗。
- 第二个 **1x1** 卷积替换为 **1x1** 分组卷积，但是并没有附加通道随机混洗。  
这是为了简单起见，因为不附加通道随机混洗已经有了很好的结果。
- 在 **3x3 depthwise** 卷积之后只有 **BN** 而没有 **ReLU**。

## 3. 当 ShuffleNet 块的步长为 2 时：

- 恒等映射直连替换为一个尺寸为 **3x3**、步长为 **2** 的平均池化。
- **3x3 depthwise** 卷积的步长为 **2**。
- 将残差部分与直连部分的 **feature map** 拼接，而不是相加。

因为当 **feature map** 减半时，为了缓解信息丢失，需要将输出通道数加倍。

## 4. ShuffleNet 网络由 ShuffleNet 块组成。

- 网络主要由三个 **Stage** 组成。每个 **Stage** 的第一个块的步长为 **2**。
- 每个 **ShuffleNet** 块中的第一个 **1x1** 分组卷积的输出通道数为：该块的输出通道数的 **1/4**。
- 超参数 **g** 控制了分组的数量。

| Layer      | Output size      | KSize        | Stride | Repeat | Output channels ( $g$ groups) |         |         |         |         |
|------------|------------------|--------------|--------|--------|-------------------------------|---------|---------|---------|---------|
|            |                  |              |        |        | $g = 1$                       | $g = 2$ | $g = 3$ | $g = 4$ | $g = 8$ |
| Image      | $224 \times 224$ |              |        |        | 3                             | 3       | 3       | 3       | 3       |
| Conv1      | $112 \times 112$ | $3 \times 3$ | 2      | 1      | 24                            | 24      | 24      | 24      | 24      |
| MaxPool    | $56 \times 56$   | $3 \times 3$ | 2      |        |                               |         |         |         |         |
| Stage2     | $28 \times 28$   |              | 2      | 1      | 144                           | 200     | 240     | 272     | 384     |
|            | $28 \times 28$   |              | 1      | 3      | 144                           | 200     | 240     | 272     | 384     |
| Stage3     | $14 \times 14$   |              | 2      | 1      | 288                           | 400     | 480     | 544     | 768     |
|            | $14 \times 14$   |              | 1      | 7      | 288                           | 400     | 480     | 544     | 768     |
| Stage4     | $7 \times 7$     |              | 2      | 1      | 576                           | 800     | 960     | 1088    | 1536    |
|            | $7 \times 7$     |              | 1      | 3      | 576                           | 800     | 960     | 1088    | 1536    |
| GlobalPool | $1 \times 1$     | $7 \times 7$ |        |        |                               |         |         |         |         |
| FC         |                  |              |        |        | 1000                          | 1000    | 1000    | 1000    | 1000    |
| Complexity |                  |              |        |        | 143M                          | 140M    | 137M    | 133M    | 137M    |

### 8.3.3 性质

- 在 `Shufflenet` 中，`depthwise` 卷积仅仅在 `bottleneck feature map` 上执行。
  - 这是因为 `depthwise` 很难在移动设备上高效的实现，因为移动设备的 `计算/内存访问` 比率较低。
  - 每个 `ShuffleNet` 块中的第一个 `1x1` 分组卷积为 `bottleneck`。
- 更大的  $g$  表示更多的分组数量，它会导致更多的输出通道数，也会导致更多的计算量和准确率。
  - 对于给定计算复杂性约束的条件下，分组越大，则允许生成更多的 `feature map` 通道数量。这有助于编码更多的信息。
    - 分组越大，准确率越高的增益来自于更大的通道数。
    - 网络通道数越小，则它从通道数量的扩张中获取的收益越大。
  - 随着分组越来越大，准确率饱和甚至下降。

这是因为：随着分组数量增大，每个组内的通道数量变少。虽然总体通道数增加，但是每个分组提取有效信息的能力变弱，降低了模型整体的表征能力。

下图中： $0.5x$  表示 `stage2` 的输入 `feature map` 的通道数量减半

| Model            | Complexity<br>(MFLOPs) | Classification error (%) |         |         |             |             |
|------------------|------------------------|--------------------------|---------|---------|-------------|-------------|
|                  |                        | $g = 1$                  | $g = 2$ | $g = 3$ | $g = 4$     | $g = 8$     |
| ShuffleNet 1×    | 140                    | 33.6                     | 32.7    | 32.6    | 32.8        | <b>32.4</b> |
| ShuffleNet 0.5×  | 38                     | 45.1                     | 44.4    | 43.2    | <b>41.6</b> | 42.3        |
| ShuffleNet 0.25× | 13                     | 57.1                     | 56.8    | 55.0    | 54.2        | <b>52.7</b> |

- 虽然较大  $g$  的 `ShuffleNet` 通常具有更好的准确率。但是由于它的实现效率较低，会带来较大的推断时间。

## 九、趋势

- 当前越来越多的卷积神经网络模型从巨型网络逐步演变为轻量级网络。如：从 `AlexNet`、`VGGNet`，到体积小一点的 `Inception`、`Resnet`，再到目前能移植到移动端的 `mobilenet`、`shufflenet`。  
目前工业界追求的重点已经不是准确率的提升（因为准确率都很高），而是速度与准确率的折中。
- 卷积核方面的一些趋势：

- 大卷积核用多个小卷积核替代
- 单一尺寸卷积核用多尺寸卷积核替代
- 固定形状卷积核趋向于使用可变形卷积核
- 使用 `1x1` 卷积核。

3. 卷积层通道方面的一些趋势：

- 标准卷积用 `depthwise` 卷积替代
- 使用分组卷积
- 分组卷积之前使用通道混洗
- 通道加权计算

4. 卷积层连接方面的一些趋势：

- 利用跳跃连接让模型更深
- 利用密集连接 使每一层都融合上其它层的特征输出