$ResNeXt$网络

历史

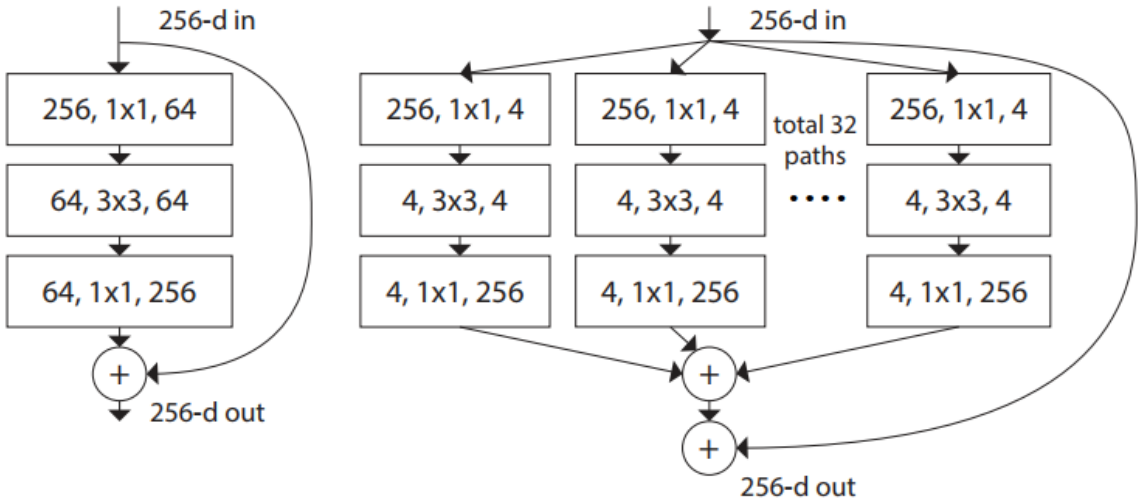
神经网络普遍存在的一个问题，如果要提高模型的准确率，往往采取加深网络或者加宽网络的方法。虽然这种方法是有效的，但是随之而来的，是网络设计的难度和计算开销的增加。为了一点精度的提升往往需要付出更大的代价。因此，需要一个更好的策略，在不额外增加计算代价的情况下，提升网络的精度。由此， 何凯明团队在2017年$CVPR$会议上提出$ResNeXt$新型图像分类网络，引入$cardinality$的概念，通过控制相同的拓扑结构。

模型原理

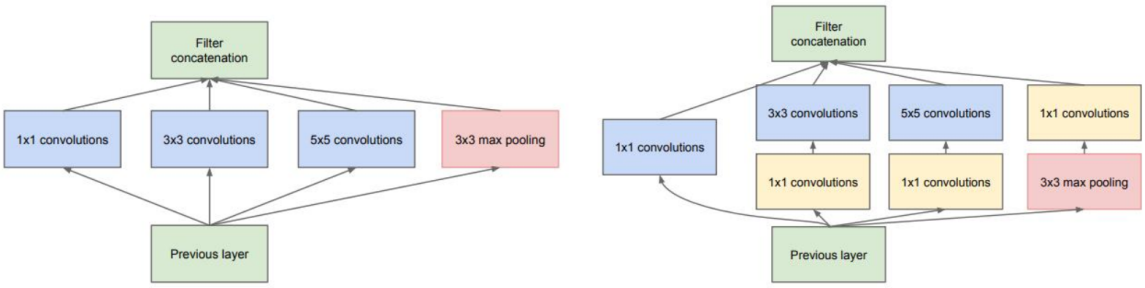
传统的 $split-transform-merge$ 结构，具有不同分支的不同拓扑结构的特征，调整 $Inception$ 的内部结构对应着大量的超参数，这些超参数调整起来是非常困难。所以作者的思想是每个结构使用相同的拓扑结构，那么这时候的 $Inception$ (这里简称简化 Inception) 表示为

$$\mathcal{F}=\sum_{i=1}^{C}\mathcal{T}_i(\mathbf{x})$$

下图是ResNet（左）与ResNeXt（右）block的差异。在ResNet中，输入的具有256个通道的特征经过1×1卷积压缩4倍到64个通道，之后3×3的卷积核用于处理特征，经1×1卷积扩大通道数与原特征残差连接后输出。ResNeXt也是相同的处理策略，但在ResNeXt中，输入的具有256个通道的特征被分为32个组，每组被压缩64倍到4个通道后进行处理。32个组相加后与原特征残差连接后输出。这里cardinatity指的是一个block中所具有的相同分支的数目。



下图是InceptionNet的两种 $inceptionmodule$结构，左边是inception module的naive版本，右边是使用了降维方法的inception module。相较于右边，左边很明显的缺点就是参数大，计算量巨大。使用不同大小的卷积核目的是为了提取不同尺度的特征信息，对于图像而言，多尺度的信息有助于网络更好地对图像信息进行选择，并且使得网络对于不同尺寸的图像输入有更好的适应能力，但多尺度带来的问题就是计算量的增加。因此在右边的模型中，InceptionNet很好地解决了这个问题，首先是1×1的卷积用于特征降维，减小特征的通道数后再采取多尺度的结构提取特征信息，在降低参数量的同时捕获到多尺度的特征信息。

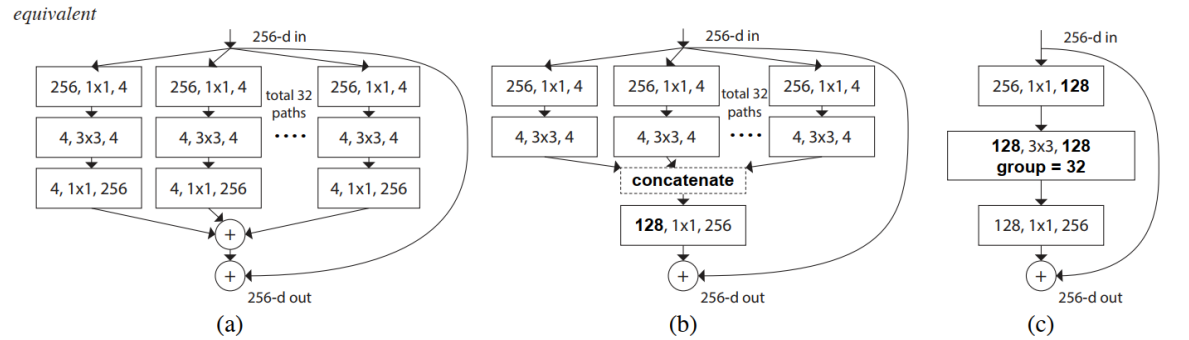


(a) Inception module, naïve version (b) Inception module with dimension reductions

ResNeXt正是借鉴了这种"分割-变换-聚合"的策略，但用相同的拓扑结构组建ResNeXt模块。每个结构都是相同的卷积核，保持了结构的简洁，使得模型在编程上更方便更容易。

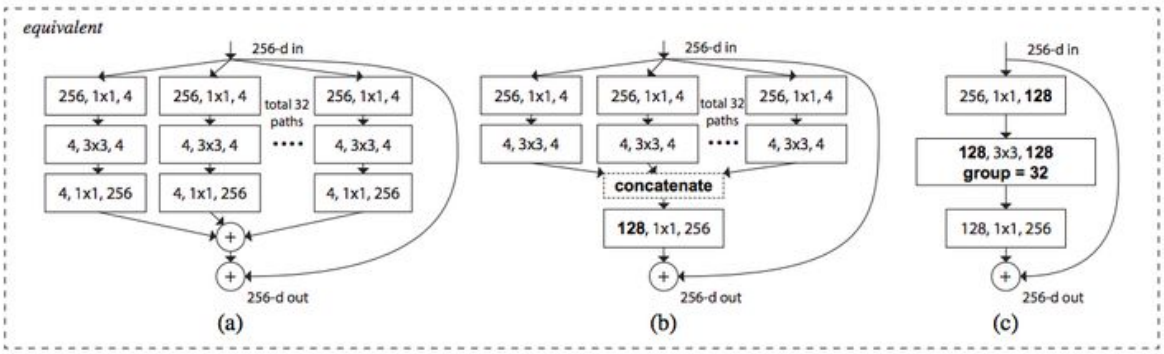
网络结构

如下图，左边是ResNet的基本结构，右边是ResNeXt的基本结构：



residual connection就是公式中的x直接连过来，然后剩下的是32组独立的同样结构的变换，最后再进行融合，符合split-transform-merge的模式。

split-transform-merge是通用的神经网络的标准范式，基本的神经元符合这个范式。而如下图所示：



a是ResNeXt基本单元，如果把输出那里的1x1合并到一起，得到等价网络b拥有和Inception-ResNet相似的结构，而进一步把输入的1x1也合并到一起，得到等价网络c则和通道分组卷积的网络有相似的结构。

事实上，该模型说明了Inception-ResNet和通道分组卷积网络，都只是ResNeXt这一范式的特殊形式而已，进一步说明了split-transform-merge的普遍性和有效性，以及抽象程度更高，更本质一点。

下面来看ResNeXt具体的网络结构。

类似ResNet，作者选择了很简单的基本结构，每一组C个不同的分支都进行相同的简单变换，下面是ResNeXt-50（32x4d）的配置清单，32指进入网络的第一个ResNeXt基本结构的分组数量C（即基数）为32，4d表示depth即每一个分组的通道数为4（所以第一个基本结构输入通道数为128）：

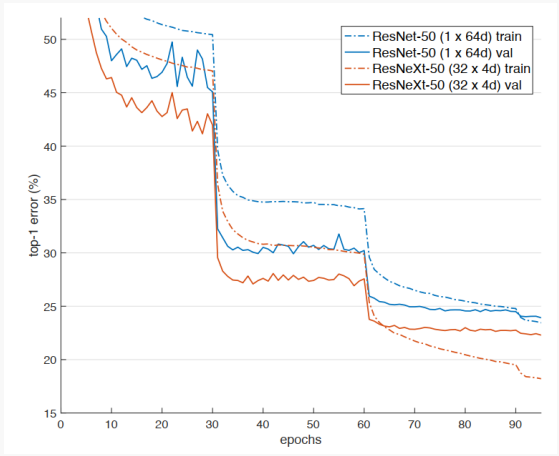
stage	output	ResNet-50	**ResNeXt-50 (32×4d)**
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2	3×3 max pool, stride 2
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, C=32 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, C=32 \\ 1\times1, 512 \end{bmatrix}\times4$
conv4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, C=32 \\ 1\times1, 1024 \end{bmatrix}\times6$
conv5	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024, C=32 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		$\mathbf{25.5}\times10^6$	$\mathbf{25.0}\times10^6$
FLOPs		$\mathbf{4.1}\times10^9$	$\mathbf{4.2}\times10^9$

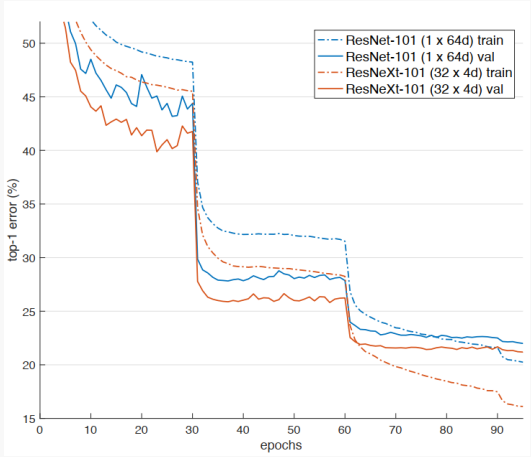
可以看到ResNet-50和ResNeXt-50（32x4d）拥有相同的参数，但是精度却更高。

具体实现上，因为1x1卷积可以合并，就合并了，代码更简单，并且效率更高。

参数量不变，但是效果太好，这个时候通常会有一个『但是』。。。但是，因为分组了，多个分支单独进行处理，所以相交于原来整个一起卷积，硬件执行效率上会低一点，训练ResNeXt-101（32x4d）每个mini-batch要0.95s，而ResNet-101只要0.70s，虽然本质上计算量是相同的，通过底层的优化因为能缩小这个差距。

数值测试


ResNet-50 (1 x 64d) train
ResNet-50 (1 x 64d) val
ResNeXt-50 (32 x 4d) train
ResNeXt-50 (32 x 4d) val
top-1 error (%)
epochs


ResNet-101 (1 x 64d) train
ResNet-101 (1 x 64d) val
ResNeXt-101 (32 x 4d) train
ResNeXt-101 (32 x 4d) val
top-1 error (%)
epochs