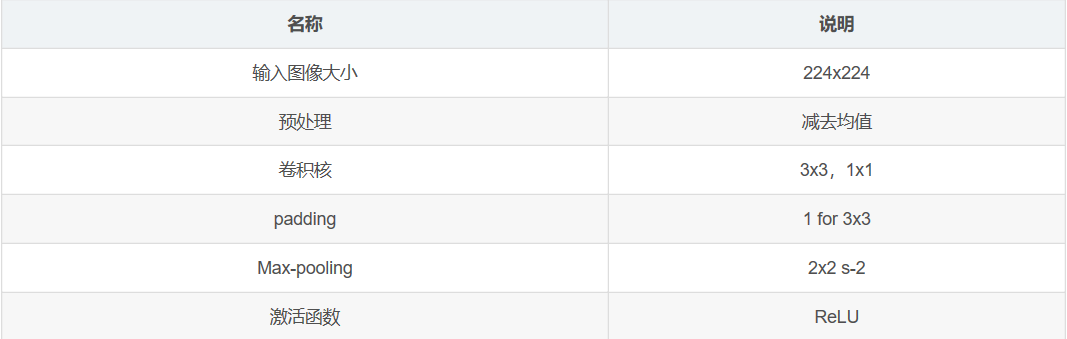
**图像分类模型**

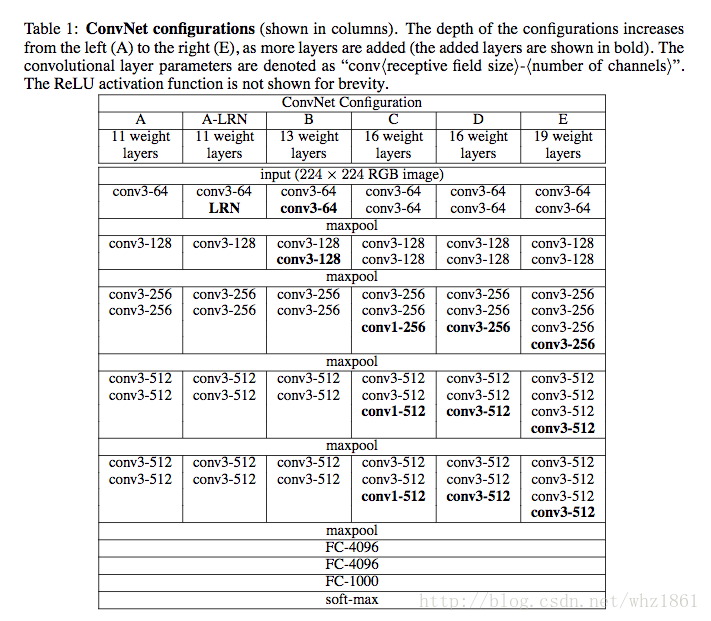
**1.VGG模型**

VGG论文给出了一个非常[振奋人心](https://www.baidu.com/s?wd=%E6%8C%AF%E5%A5%8B%E4%BA%BA%E5%BF%83&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)的结论：**卷积神经网络的深度增加和小卷积核的使用对网络的最终分类识别效果有很大的作用**。记得在AlexNet论文中，也做了最后指出了网络深度的对最终的分类结果有很大的作用。这篇论文则更加直接的论证了这一结论。



**结构Architecture**

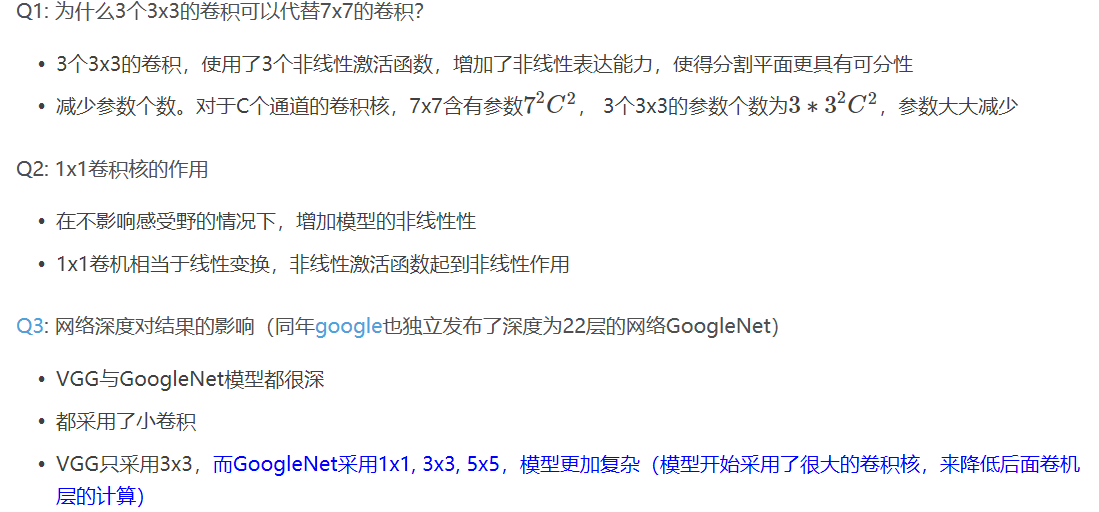
****



说明：

\*1x1卷积核：降维，增加非线性性

\*3x3卷积核：多个卷积核叠加，增加空间感受野，减少参数

****

**2.Resnet**

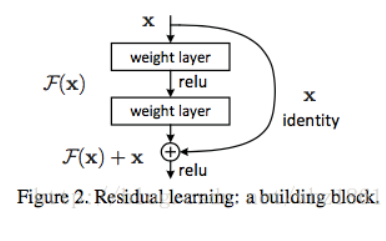
网络越深，越容易出现梯度消失，导致模型训练难度变大，出现“退化”现象

Q：如何有效的解决网络深度的增加，带来的“退化”问题？

A：出现退化的问题，主要是由于网络深度的增加，带来的在网络训练的时候，梯度无法有效的传递到浅层网络，导致出现梯度弥散（vanishing）。**BN**（BatchNormalization）通过规范化输出数据来改变数据分布，是一个前向的过程来解决梯度弥散问题，而resNet网络通过增加skip connection（Identity Map）来直接连接浅层网络与深层网络，从而使得梯度能够很好的传递到浅层。

假设：如果多层非线性层叠加可以渐进逼近复杂的函数，那么也能渐进逼近残差函数：H(X)-X

其中一个残差结构如下：



说明：

如果添加了identity mapings，则结构更深的网络不会比浅的网络效果差

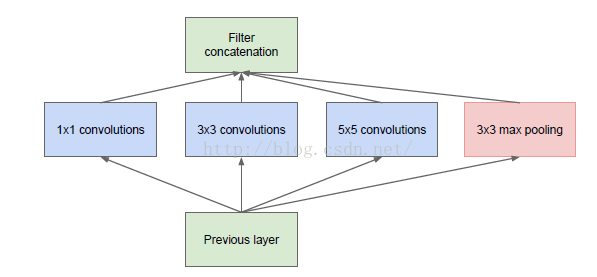
多层非线性层叠加在一起构成的网络结构，很难学习到identify mapings

如果identity mapings是最优的链接方式，那么F(X)F(X)方面的权重参数则会趋于0

如果最优的映射接近identity mapings，则优化的时候，找到相对于identity mapings的F(x)F(x)（初始参数在0附近），要比逼近一个全新的函数要简单的多

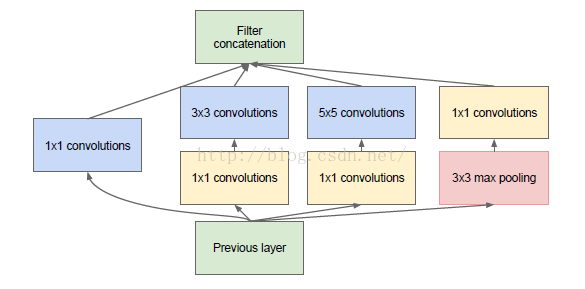
**3. Inception**

**Inception V1**



**Inception module** 的提出主要考虑多个不同 size 的卷积核能够增强网络的适应力，paper 中分别使用1\*1、3\*3、5\*5卷积核，同时加入3\*3 max pooling。  
随后文章指出这种 naive 结构存在着问题：每一层 Inception module 的 filters 参数量为所有分支上的总数和，多层 Inception 最终将导致 model 的参数数量庞大，对计算资源有更大的依赖。

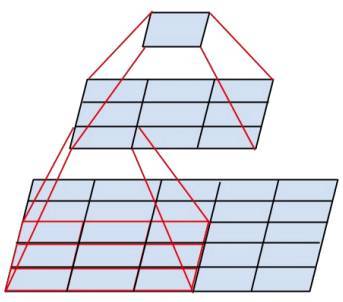
在 NIN 模型中与1\*1卷积层等效的 MLPConv 既能跨通道组织信息，提高网络的表达能力，同时可以对输出有效进行**降维**，因此文章提出了**Inception module with dimension reduction**，在不损失模型特征表示能力的前提下，尽量减少 filters 的数量，达到降低模型复杂度的目的：



Inception Module 的4个分支在最后通过一个聚合操作合并（在输出通道数这个维度上聚合，在 TensorFlow 中使用 tf.concat(3, [], []) 函数可以实现合并）。

**Inception V2**

Inception V2 学习了 **VGG**用两个3´3的卷积代替5´5的大卷积，在降低参数的同时建立了更多的非线性变换，使得 CNN 对特征的学习能力更强：

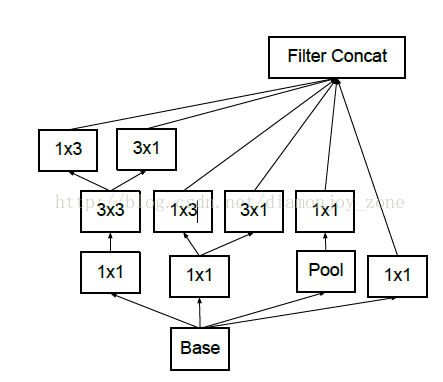


**两个3´3的卷积层功能类似于一个5´5的卷积层**

另外提出了著名的 **Batch Normalization**（以下简称BN）方法。BN 是一个非常有效的正则化方法，可以让大型卷积网络的训练速度加快很多倍，同时收敛后的分类准确率也可以得到大幅提高。BN 在用于神经网络某层时，会对每一个 mini-batch 数据的内部进行标准化（**normalization**）处理，使输出规范化到 N(0,1) 的正态分布，减少了 Internal Covariate Shift（内部神经元分布的改变）。

BN 的论文指出，传统的深度神经网络在训练时，每一层的输入的分布都在变化，导致训练变得困难，我们只能使用一个很小的学习速率解决这个问题。而对每一层使用 BN 之后，我们就可以有效地解决这个问题，学习速率可以增大很多倍，达到之前的准确率所需要的迭代次数只有1/14，训练时间大大缩短。因为 BN 某种意义上还起到了正则化的作用，所以可以减少或者取消 **Dropout** 和 **LRN**，简化网络结构。

**Inception V3**



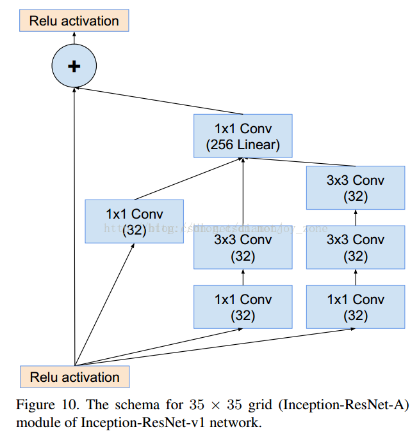
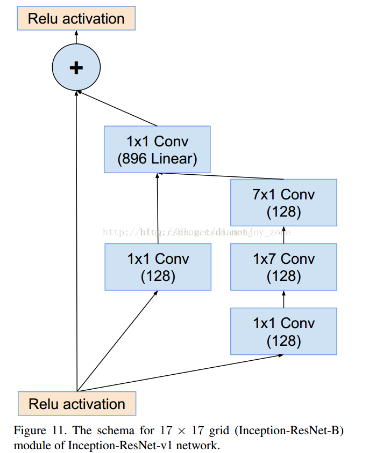
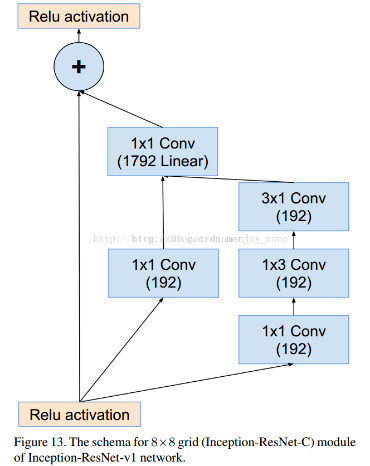
将一个3´3卷积拆成1´3卷积和3´1卷积

一是引入了 **Factorization into small convolutions** 的思想，将一个较大的二维卷积拆成两个较小的一维卷积，比如将7´7卷积拆成1´7卷积和7´1卷积，或者将3´3卷积拆成1´3卷积和3´1卷积，如上图所示。一方面节约了大量参数，加速运算并减轻了过拟合（比将7´7卷积拆成1´7卷积和7´1卷积，比拆成3个3´3卷积更节约参数），同时**增加了一层非线性扩展模型表达能力**。论文中指出，这种非对称的卷积结构拆分，其结果比对称地拆为几个相同的小卷积核效果更明显，可以处理更多、更丰富的空间特征，增加特征多样性。

另一方面，Inception V3 优化了 Inception Module 的结构，现在 Inception Module 有35´35、17´17和8´8三种不同结构。这些 Inception Module 只在网络的后部出现，前部还是普通的卷积层。并且 Inception V3 除了在 Inception Module 中使用分支，还在分支中使用了分支（8´8的结构中），可以说是Network In Network In Network。最终取得 top-5 错误率 **3.5%**。

**Inception V4**

Inception V4 相比 V3 主要是结合了微软的 **ResNet**，将错误率进一步减少到 **3.08%**。

**总结**

**Inception V1**——构建了1x1、3x3、5x5的 conv 和3x3的 pooling 的**分支网络**，同时使用 **MLPConv**和**全局平均池化**，扩宽卷积层网络宽度，增加了网络对尺度的适应性；

**Inception V2**——提出了 **Batch Normalization**，代替 **Dropout**和 **LRN**，其正则化的效果让大型卷积网络的训练速度加快很多倍，同时收敛后的分类准确率也可以得到大幅提高，同时学习 **VGG** 使用两个3´3的卷积核代替5´5的卷积核，在降低参数量同时提高网络学习能力；

**Inception V3**——引入了 **Factorization**，将一个较大的二维卷积拆成两个较小的一维卷积，比如将3´3卷积拆成1´3卷积和3´1卷积，一方面节约了大量参数，加速运算并减轻了过拟合，同时增加了一层非线性扩展模型表达能力，除了在**Inception Module**中使用分支，还在分支中使用了分支（**Network In Network In Network**）；

**Inception V4**——研究了 **Inception Module**结合 **Residual Connection**，结合 **ResNet** 可以极大地加速训练，同时极大提升性能，在构建 Inception-ResNet 网络同时，还设计了一个更深更优化的 Inception v4 模型，能达到相媲美的性能

**4. Xception**

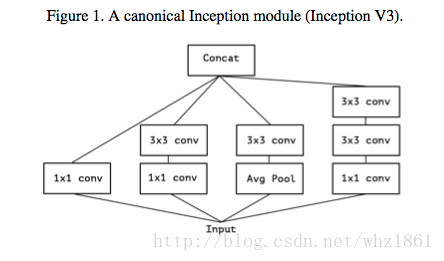
思想:

基于Inception系列网络结构的基础上，结合depthwise separable convolution

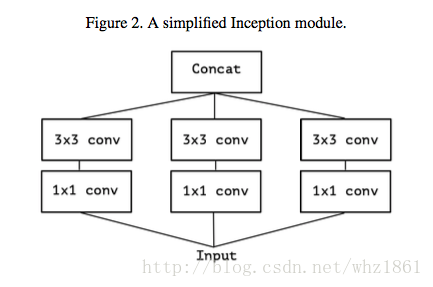
将Inception modul拆分成一系列操作，独立处理spatial-correlations和cross-channel correlations，网络处理起来更加简单有效.

Q:为什么Inception module可以用depthwise separable convolution替代

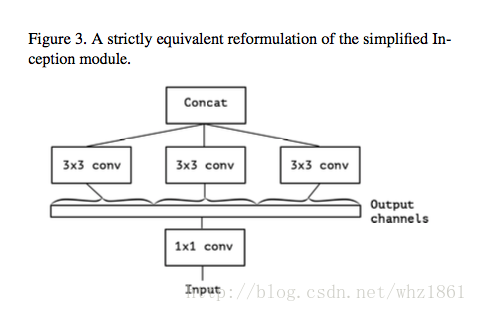
首先，对于原始的Inception模块，如Inception V3，结构如下：



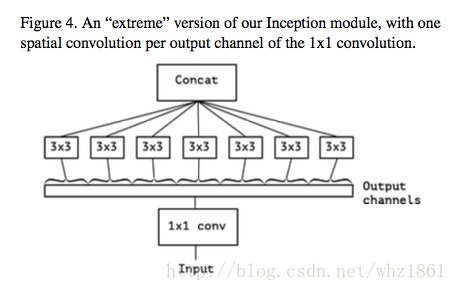
模块中，基本上先通过一系列1x1卷积降维，然后再通过3x3卷积提取特征。如果我们将上述结构再进行简化，可以得到如下简化结构：



从上图中我们可以看出，1x1卷积将输入数据的channel维度上进行了拆解，再输送到空间卷积3x3，改写成下图：



可以考虑一种更加极端的情况：3x3卷积在1x1卷积后的每一个通道上运行，则有：



我们可以从上图上，看到该模块将输入数据在channel维度上进行解耦合，该模块称之为‘extreme’version of Inception module，这个思想和[depthwise separable convolution](http://blog.csdn.net/whz1861/article/details/78335146)非常相似。

先用一个统一的1\*1卷积核卷积，然后连接3个3\*3的卷积，这3个卷积操作只将前面1\*1卷积结果中的一部分作为自己的输入（这里是将1/3channel作为每个3\*3卷积的输入）。再从Figure3延伸就得到Figure4，也就是3\*3卷积的个数和1\*1卷积的输出channel个数一样，每个3\*3卷积都是和1个输入channel做卷积。

* 操作顺序不同：

depth wise separable convolutions：先进行channel-wise 空间卷积，然后1x1卷积进行融合

Inception：先进行1x1卷积，然后进行channel-wise空间卷积

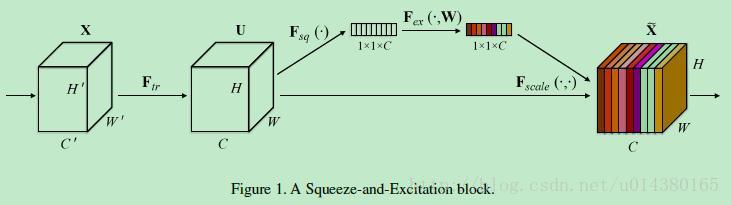
* 非线性激励函数：

depthwise separable convolution：两个操作之间没有激励函数

Inception：两个操作之间，添加了**ReLU非线性激励**

**5. Squeeze-and-Excitation Networks**

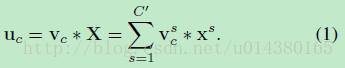
SENet的核心思想在于通过网络根据loss去学习特征权重，使得有效的feature map权重大，无效或效果小的feature map权重小的方式训练模型达到更好的结果.



首先Ftr这一步是转换操作（严格讲并不属于SENet，而是属于原网络，可以看后面SENet和Inception及ResNet网络的结合），在文中就是一个标准的卷积操作而已，输入输出的定义如下表示。

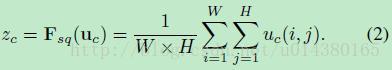
这里写图片描述

那么这个Ftr的公式就是下面的公式1（卷积操作，vc表示第c个卷积核，xs表示第s个输入）。



Ftr得到的U就是Figure1中的左边第二个三维矩阵，也叫tensor，或者叫C个大小为H\*W的feature map。而uc表示U中第c个二维矩阵，下标c表示channel。

接下来就是**Squeeze**操作，公式非常简单，就是一个global average pooling：



因此公式2就将H*W*C的输入转换成1*1*C的输出，对应Figure1中的Fsq操作。**为什么会有这一步呢？这一步的结果相当于表明该层C个feature map的数值分布情况，或者叫全局信息。**

再接下来就是Excitation操作，如公式3。直接看最后一个等号，前面squeeze得到的结果是z，这里先用W1乘以z，就是一个全连接层操作，W1的维度是(输出/输入)C/r \* C，这个r是一个缩放参数，在文中取的是16，这个参数的目的是为了减少channel个数从而降低计算量。又因为z的维度是11C，所以W1z的结果就是**11C/r**；然后再经过一个ReLU层，输出的维度不变；然后再和W2相乘，和W2相乘也是一个全连接层的过程，W2的维度是C\*C/r，因此输出的维度就是**11C**；最后再经过sigmoid函数，得到s。

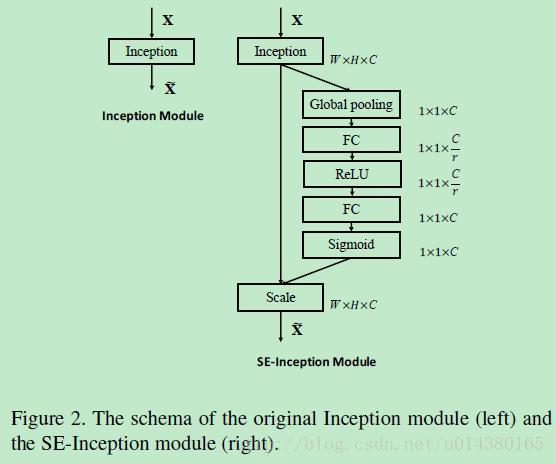
这里写图片描述

也就是说最后得到的这个s的维度是11C，C表示channel数目。这个s其实是本文的核心，它是用来刻画tensor U中C个feature map的权重。而且这个权重是通过前面这些全连接层和非线性层学习得到的，因此可以end-to-end训练。这两个全连接层的作用就是融合各通道的feature map信息，因为前面的squeeze都是在某个channel的feature map里面操作。

在得到s之后，就可以对原来的tensor U操作了，就是下面的公式4。也很简单，就是channel-wise multiplication，什么意思呢？uc是一个二维矩阵，sc是一个数，也就是权重，因此相当于把uc矩阵中的每个值都乘以sc。对应Figure1中的Fscale。

这里写图片描述

了解完上面的公式，就可以看看在实际网络中怎么添加SE block。Figure2是在Inception中加入SE block的情况，这里的Inception部分就对应Figure1中的Ftr操作。



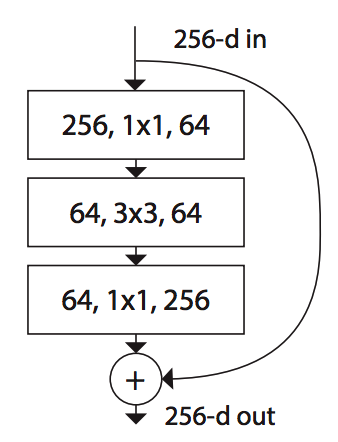
**6. ResNeXt - Aggregated Residual Transformations for Deep Neural Networks**

VGG-nets/ResNets： 堆叠相同形状的网络 building blocks；—— 网络 depth

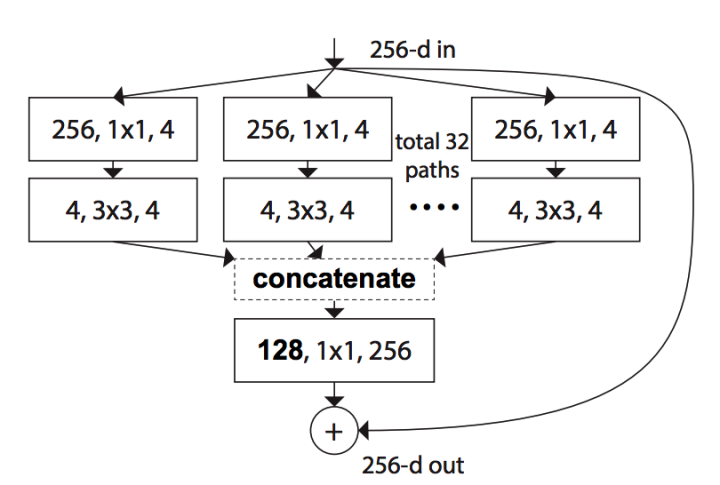
Inceptions：split-transform-merge，将输入采用(1×11×1 Conv)分裂为几个低维 embedding，再经过一系列特定 filters (如3×33×3，5×55×5)的变换，最后连接在一起.

ResNeXt：采用 VGGs/ResNets 的网络的 depth 加深方式，同时利用 split-transform-merge 策略.

在ResNext模型中，作者提出了Cardinality这个概念，它表示了通道分成的组。假如说我们本来有这样一段模型结构：



此时中间的计算是经典的卷积操作，所有的通道信息合并处理，也相当于把通道分成了1组，如果想要将通道分成两组，那么运算过程中，在中间的层次，我们会得到两个结果，这两个结果分别计算，最后合并起来（concatenate）或者加和起来（sum）：



对于Xception模型中，作者提出了“depth-wise separable convolution”的结构，也就是说，在做卷积计算的时候，每一个通道内的数据做单独计算，也就是说，如果有N维通道输出，那么Cardinality就等于N。

两个模型的其他方面各有异同，但思路上却十分一致，都是要对通道下手，把convolution进化成为group convolution。

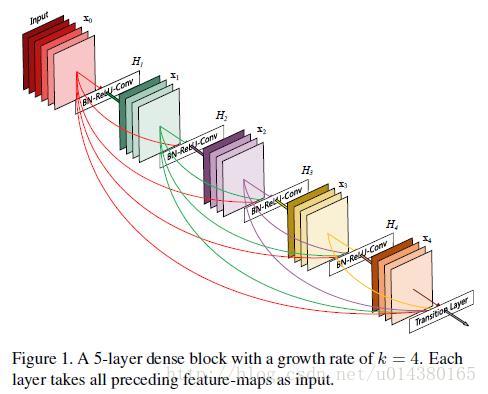
**7. DenseNet（Dense Convolutional Network）**

作者是从feature入手，通过对feature的极致利用达到更好的效果和更少的参数。

先列下DenseNet的几个优点，感受下它的强大： 减轻了vanishing-gradient（梯度消失） ；加强了feature的传递 ；更有效地利用了feature ；一定程度上较少了参数数量。

**保证网络中层与层之间最大程度的信息传输的前提下，直接将所有层连接起来！**

先放一个dense block的结构图。在传统的卷积神经网络中，如果你有L层，那么就会有L个连接，但是在DenseNet中，会有L(L+1)/2个连接。简单讲，就是每一层的输入来自前面所有层的输出。如下图：x0是input，H1的输入是x0（input），H2的输入是x0和x1（x1是H1的输出）……



DenseNet的一个优点是网络更窄，参数更少，很大一部分原因得益于这种dense block的设计，后面有提到在dense block中每个卷积层的输出feature map的数量都很小（小于100），而不是像其他网络一样动不动就几百上千的宽度。同时这种连接方式使得特征和梯度的传递更加有效，网络也就更加容易训练。原文的一句话非常喜欢：Each layer has direct access to the gradients from the loss function and the original input signal, leading to an implicit deep supervision.直接解释了为什么这个网络的效果会很好。前面提到过梯度消失问题在网络深度越深的时候越容易出现，原因就是输入信息和梯度信息在很多层之间传递导致的，而现在这种dense connection相当于每一层都直接连接input和loss，因此就可以减轻梯度消失现象，这样更深网络不是问

另外作者还观察到这种dense connection有正则化的效果，因此对于过拟合有一定的抑制作用，博主认为是因为参数减少了（后面会介绍为什么参数会减少），所以过拟合现象减轻。

第一个公式是ResNet的。这里的l表示层，xl表示l层的输出，Hl表示一个非线性变换。所以对于ResNet而言，l层的输出是l-1层的输出加上对l-1层输出的非线性变换。

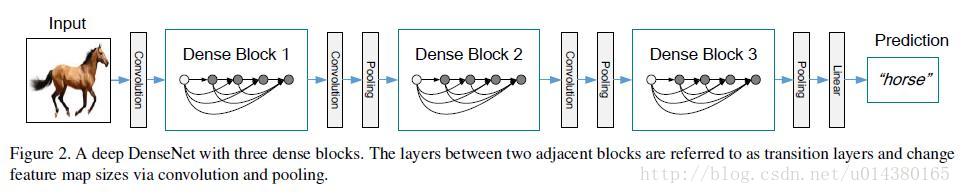
这里写图片描述

第二个公式是DenseNet的。[x0,x1,…,xl-1]表示将0到l-1层的输出feature map做concatenation。concatenation是做通道的合并，就像Inception那样。而前面resnet是做值的相加，通道数是不变的。Hl包括BN，ReLU和3\*3的卷积。

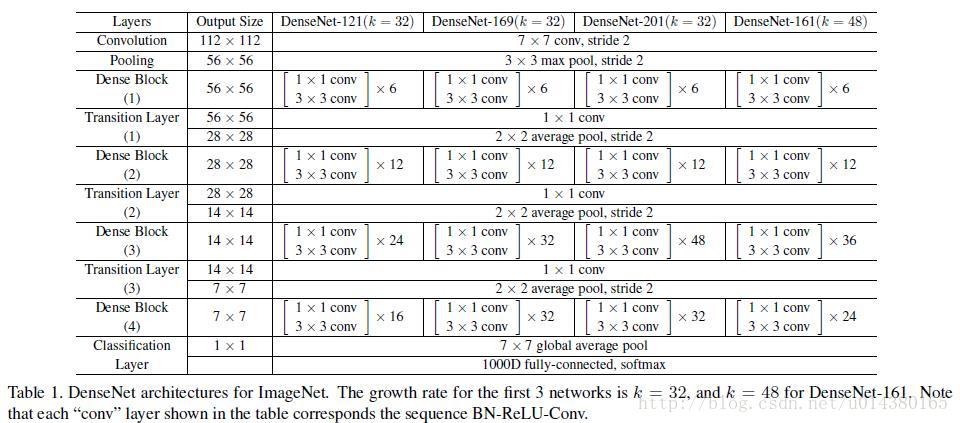
这里写图片描述

**所以从这两个公式就能看出DenseNet和ResNet在本质上的区别，太精辟。**

前面的Figure 1表示的是dense block，而下面的Figure 2表示的则是一个DenseNet的结构图，在这个结构图中包含了3个dense block。作者将DenseNet分成多个dense block，原因是希望各个dense block内的feature map的size统一，这样在做concatenation就不会有size的问题。



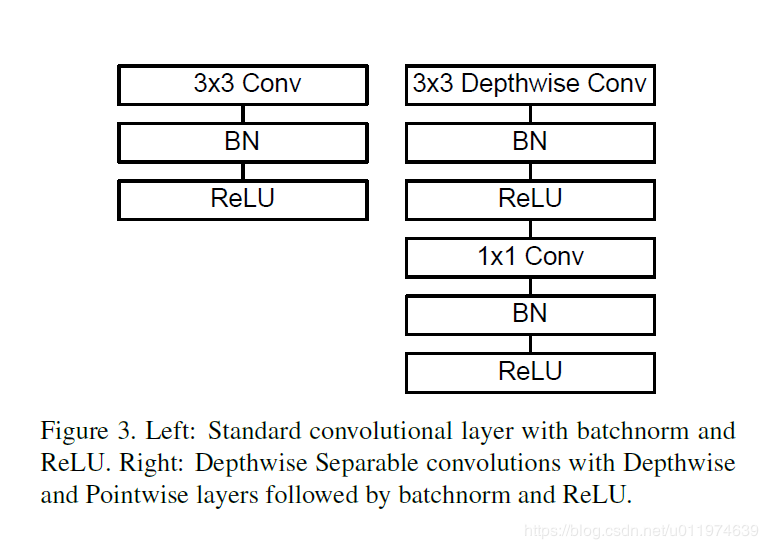
这个Table1就是整个网络的结构图。这个表中的k=32，k=48中的k是growth rate，表示每个dense block中每层输出的feature map个数。为了避免网络变得很宽，作者都是采用较小的k，比如32这样，作者的实验也表明小的k可以有更好的效果。根据dense block的设计，后面几层可以得到前面所有层的输入，因此concat后的输入channel还是比较大的。另外这里每个dense block的3\*3卷积前面都包含了一个1\*1的卷积操作，就是所谓的bottleneck layer，目的是减少输入的feature map数量，既能降维减少计算量，又能融合各个通道的特征，何乐而不为。另外作者为了进一步压缩参数，在每两个dense block之间又增加了1\*1的卷积操作。因此在后面的实验对比中，如果你看到DenseNet-C这个网络，表示增加了这个Translation layer，该层的1\*1卷积的输出channel默认是输入channel到一半。如果你看到DenseNet-BC这个网络，表示既有bottleneck layer，又有Translation layer。



再详细说下bottleneck和transition layer操作。在每个Dense Block中都包含很多个子结构，以DenseNet-169的Dense Block（3）为例，包含32个1\*1和3\*3的卷积操作，也就是第32个子结构的输入是前面31层的输出结果，每层输出的channel是32（growth rate），那么如果不做bottleneck操作，第32层的3\*3卷积操作的输入就是31\*32+（上一个Dense Block的输出channel），近1000了。而加上1\*1的卷积，代码中的1\*1卷积的channel是growth rate\*4，也就是128，然后再作为3\*3卷积的输入。这就大大减少了计算量，这就是bottleneck。至于transition layer，放在两个Dense Block中间，是因为每个Dense Block结束后的输出channel个数很多，需要用1\*1的卷积核来降维。还是以DenseNet-169的Dense Block（3）为例，虽然第32层的3\*3卷积输出channel只有32个（growth rate），但是紧接着还会像前面几层一样有通道的concat操作，即将第32层的输出和第32层的输入做concat，前面说过第32层的输入是1000左右的channel，所以最后每个Dense Block的输出也是1000多的channel。因此这个transition layer有个参数reduction（范围是0到1），表示将这些输出缩小到原来的多少倍，默认是0.5，这样传给下一个Dense Block的时候channel数量就会减少一半，这就是transition layer的作用。文中还用到dropout操作来随机减少分支，避免过拟合，毕竟这篇文章的连接确实多。

**8.Mobilenet**

标准卷积和MobileNet中使用的深度分离卷积结构对比如下



**宽度因子和分辨率因子**

宽度因子α是一个属于(0,1]之间的数，附加于网络的通道数。简单来说就是新网络中每一个模块要使用的卷积核数量相较于标准的MobileNet比例。对于deep-wise结合1x1方式的卷积核，计算量为：

https://img-blog.csdn.net/20171223141628444

α常用的配置为1,0.75,0.5,0.25；当α等于1时就是标准的MobileNet。通过参数α可以非常有效的将计算量和参数数量约减到α的平方倍。

分辨率因子β的取值范围在(0,1]之间，是作用于每一个模块输入尺寸的约减因子，简单来说就是将输入数据以及由此在每一个模块产生的特征图都变小了，结合宽度因子α，deep-wise结合1x1方式的卷积核计算量为：

https://img-blog.csdn.net/20171223141716599

**9. NASNet**

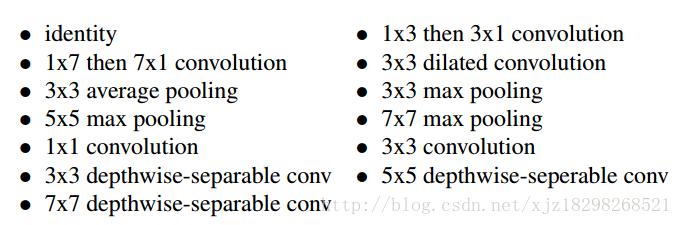
看完NAS论文和此篇论文，我将从以下几点描述论文的核心点。

核心一：延续NAS论文的核心机制使得能够自动产生网络结构；

核心二：采用resnet和Inception重复使用block结构思想；

核心三：利用迁移学习将生成的网络迁移到大数据集上提出一个new search space。

Normal Cell：不改变输入feature map的大小的卷积；   
   Reduction Cell：将输入feature map的长宽各减少为原来的一半的卷积，是通过增加stride的大小来降低size



连接方式一般有两种选择

（1）element-wise addition between two hidden states

（2）concatenation between two hidden states along the filter dimension

