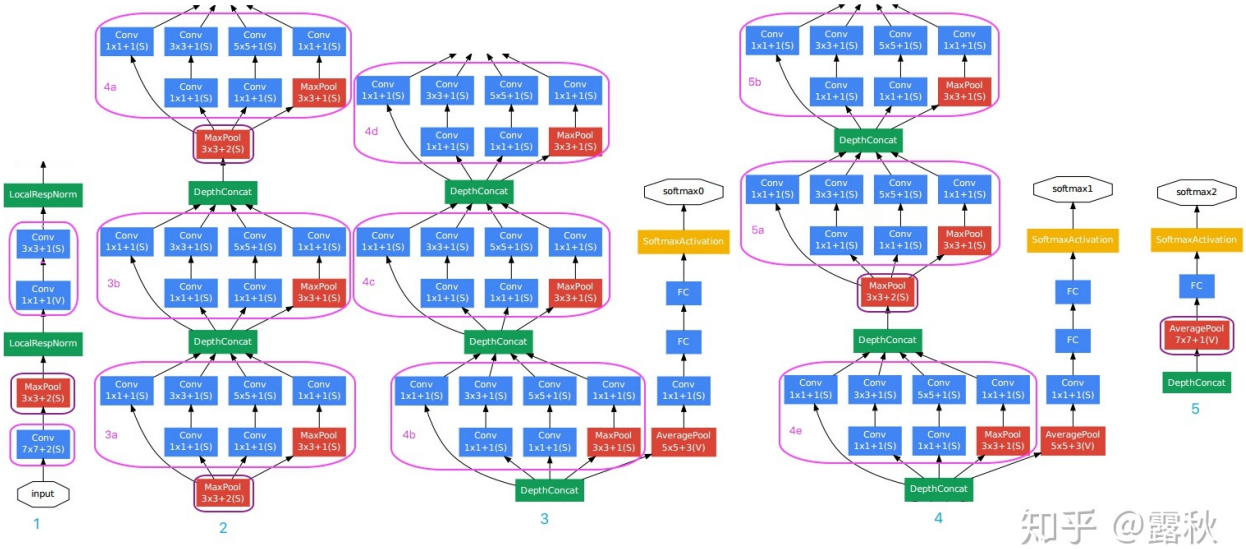
简介

GooLeNet Inception V1是 GoogLeNet 最早的版本，论文发布于2014年《Going deeper with convolutions》。之后 google 有相继发布了 Inception 的 V2、V3、V4（即 Xception）版本，是的 Inception 越来越完善。此处最终 Inception V1 论文的阅读笔记，中间遇到的问题、不懂的概念都做了记录。此处不对论文做过多解释，有兴趣可以查看原始论文，更多的相关资料可以查看此文章《CNN模型相关论文阅读资料》

GoogLeNet 架构图

GoogLeNet 的结构图如下：



原始的结构图是上下结构，放到文章中显得比较长，这里把每一部分接取下来，横着排放，这样就可以一眼看清楚整体的结构图，从左到右依次把 5 个部分连接起来，就可以得到完成的结构图。

GooLeNet 只算具有参数的层是 22 层，把没有参数的 pooling 也算上是 27 层。

图1 方框中的数字（3a、3b、4a、4b、5a 等）来源于原论文中的 Table1：

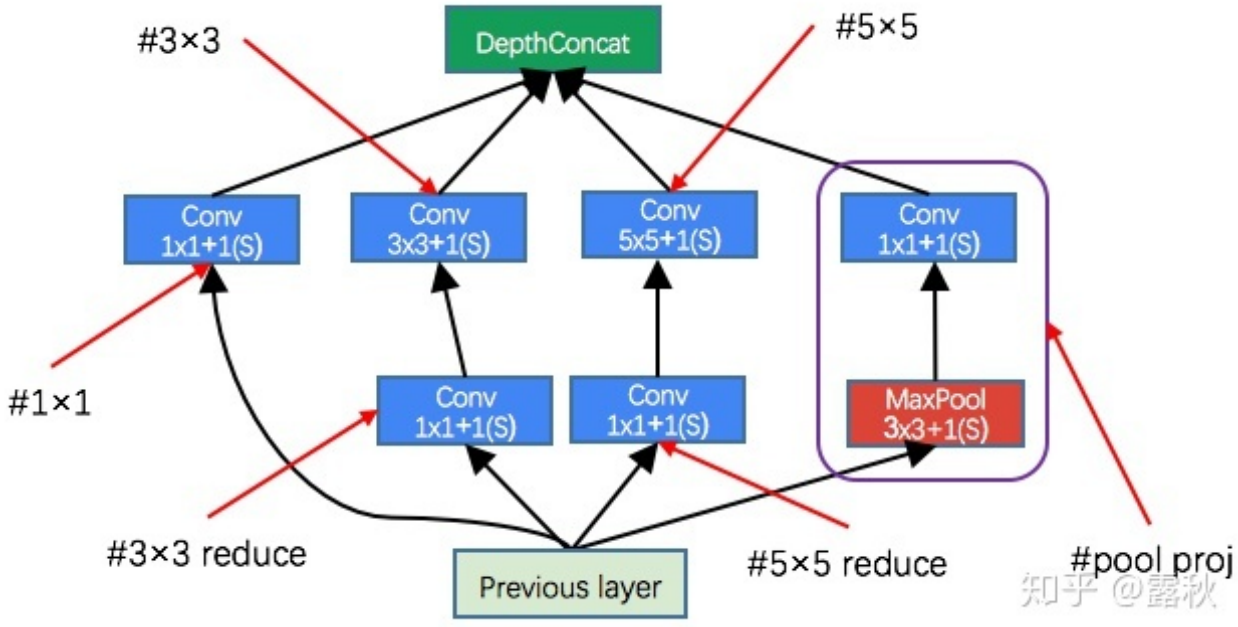
type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96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

Table 1: GoogLeNet incarnation of the Inception architecture

从图1 中可以看到，模型添加了两个辅助的 softmax 分类器，论文中提到是为了避免梯度消失，通过两个辅助 softmax 分类器向模型低层次注入梯度，同时也提供了额外的正则化，也让底层网络提取的特征更具判别性（discrimination）。文中也提到这两个辅助 softmax 分类器的损失函数（Loss Function）在计算总的损失是需要添加一个衰减系数，文中给出的是 0.3 。在进行实际推断（ inference）时，需要把这两个辅助 softmax 分类器丢弃。

Inception-3a 输出计算

上图 红色圈出来的部分就是 Inception 参数说明，表中值表示处理之后输出的通道数，也可以理解为该层的 filter 个数。以上图的 inception(3a) 行为例，对应的 GooLeNet 中的结构如下：



从 Table 1 中可以看到 Inception-3a 的输入是上一层 max pooling 的输出，即 Inception-3a 的输入大小为 28×28×192 （W×H×C）。那么 Inception-3a 的输出计算方式如下：

- # 1×1 的输出：此卷积使用的卷积核个数从 Table 1 中可知为 64，因此这一层的输出大小为 28×28×64 ①
- # 3×3 reduce 与 # 3×3 的输出： 通过 # 3×3 reduce 的操作输出为 28×28×96，将此输出作为 # 3×3 的输入，如果我们直接进行卷积操作就会发现，输出的结果为 26×26×128，与前面输出的结果就没有办法进行连接，因此需要在此处进行补零操作，即设置 padding=1 ，这样得到的输出结果就为 28×28×128 ②
- # 5×5 reduce 与 # 5×5 的输出：同上 # 5×5 reduce 的输出为 28×28×16，传入 # 5×5 ，此处依然需要进行补零操作 padding=2 ，从而得到输出 28×28×32 ③
- pool proj 的输出：Max Pool为 3×3 ，因此要得到大小为 28×28 的输出就需要进行补零操作，此处 padding=1。Max Pool 的输入为 28×28×192 ，经过补零操作之后输出依然为 28×28×192，将其输入 Conv 1×1×32 （从 Table 1 可知卷积核个数为32个）从而得到的输出为 28×28×32 ④

最后将计算的结果：
① 28×28×64
② 28×28×128
③ 28×28×32
④ 28×28×32

进行连接（即将通道数相加）可得到最终的输出结果：28×28×256

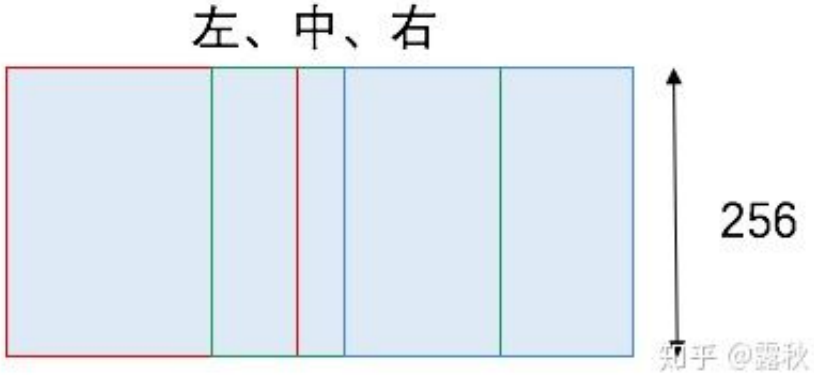
图像采样

看到论文中 ILSVRC 2014 Classification Challenge Setup and Results 章节，对在测试阶段使用的激进裁剪操作方式不是特别理解，到底是怎么操作的。通过查找资料，汇总下相关的观点。

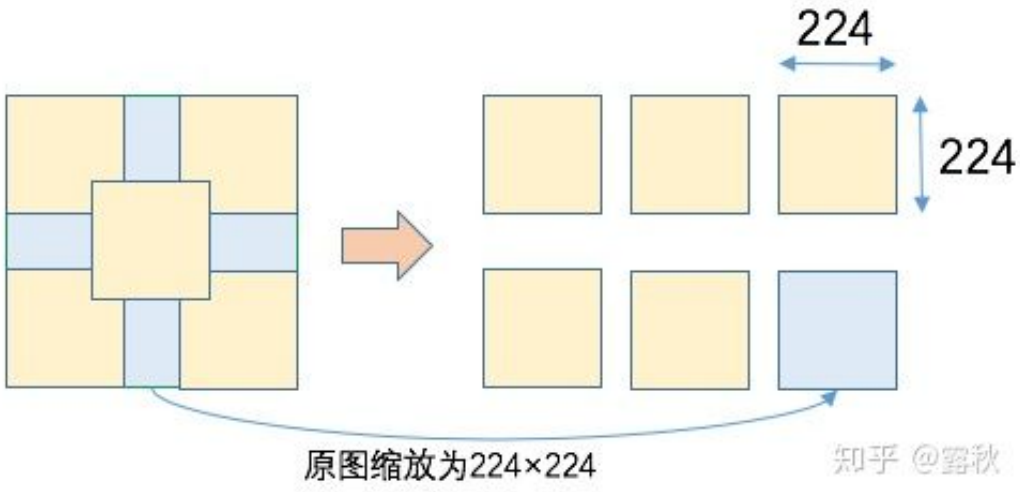
ILSVRC 2014 中的图像尺寸并不是都是正方形，正好适合输入 GooLeNet 模型中，对于图片进行相应的缩放操作是必要的。而论文中提到的裁剪的方式，实际是一种数据增强的方式。下面就具体分析下论文中是怎么操作的。

首先将一张图片的最短边缩放为 256、288、320 、352 四个尺寸，如一张 720×1024 的图片，通过上面的操作之后，短边 720 就变成了上面的四个值，相应的长边也进行了缩放，论文中提到缩放的比例在 4/3 与 3/4 之间。这样就得到了 4 张尺寸的图片。

然后在得到的缩放图片上，以左、中、右的方式截取方形区域，如果是人像则采取上、中、下的截取方式，这样就产生了 3 个区域：



接着从截取的区域中，选择四个角和中心进行截取 224 大小的区域，最后把原图缩放为 224×224大小，这样就得到了 6 张图片：



最后每张图再取一个镜像版本，这样就得到了 2 张图。

经过上面一系列操作之后得到的图片数量为 4(四个尺寸) × 3(左、中、右) × 6(4角 + 中心 + 原图缩放) × 2(镜像) = 144 张。

问题

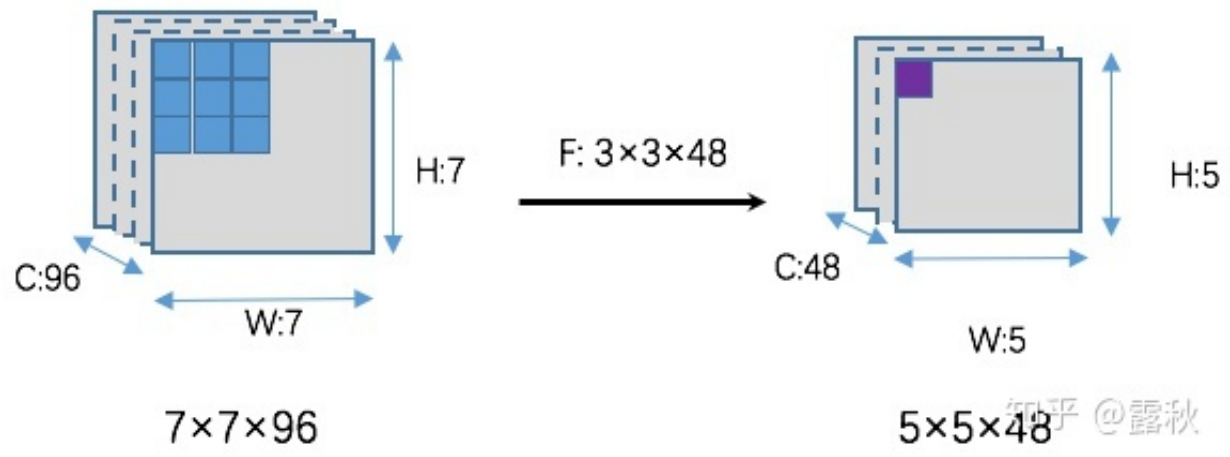
为什么可以在不同的 scale 上提取特征?

阅读论文的时候，很奇怪为什么 Inception 就可以实现视觉信息在不同尺寸上处理，然后融合，下一层就可以在不同的 scale 上提取特征。通过阅读一些文章，我的理解是这样的（可能有偏差）：

从 Inception-3a 输出计算 章节中，我们可以看到在 DepathConcat 之前一共有四类输出，即：28×28×64、28×28×128、28×28×32 、28×28×32，通过 concat 之后得到最终的输出结果 28×28×256。但我们可以看到这 256 个通道中的每一个神经元在前一层的视野是不同的：

① 28×28×64：1×1 的视野
② 28×28×128：3×3 的视野
③ 28×28×32 ： 5×5 的视野
④ 28×28×32：3×3 的视野

那么也就是说，下一层在 28×28×256 上操作时，其实是在不同的 scale 上提取特征。对比一下传统的卷积：



传统的卷积每一层使用的卷积核大小都是一致的，因此输出的每一个神经元在前一层的接受视野也都是一致的。这样下一层就没有办法在不同的 scale 进行特征提取。接受视野的计算可以参考此文 《CNN：接受视野（Receptive Field）》

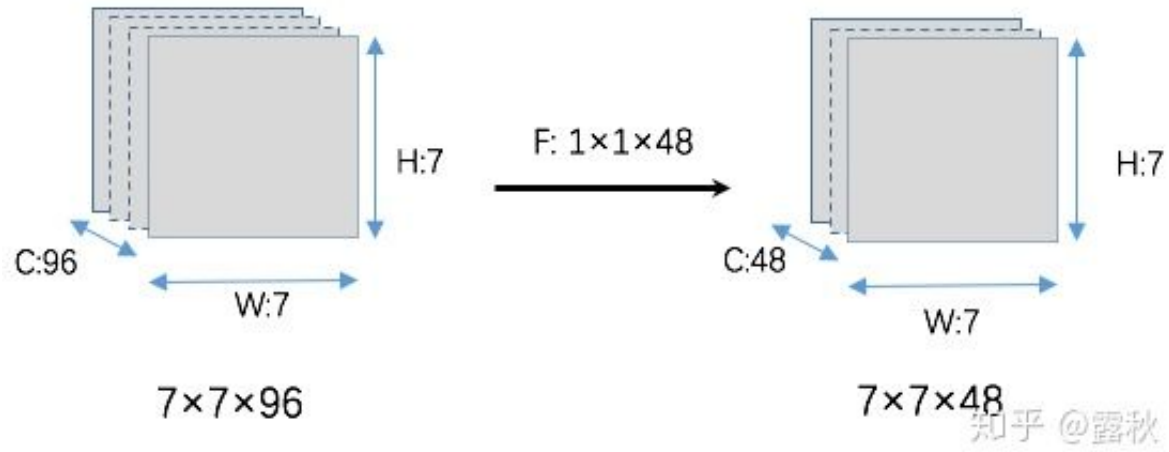
为什么 1×1 的卷积核可以降维?

在读原始论文的时候有一点非常疑惑，1×1 的卷积核（Filter F）操作输入以后输出的尺寸不是还是和原来的一样吗？比如输入是 7×7 ，那么经过 1×1 的卷积核操作之后，输出的的大小仍是 7×7，怎么就降维了呢?

查了资料之后才知道，上面的 7×7 的例子是只考虑了宽（Width W） 和高（ Height H）这两个维度，并没有考虑通道（Channel C）这一维度，当我们把 Channel 也考虑进去就不一样了。虽然1×1 的卷积核并没有改变 Width 和 Height，但是我们可以通过选择卷积核的个数来改变输出 Channel 的个数。

举个例子：

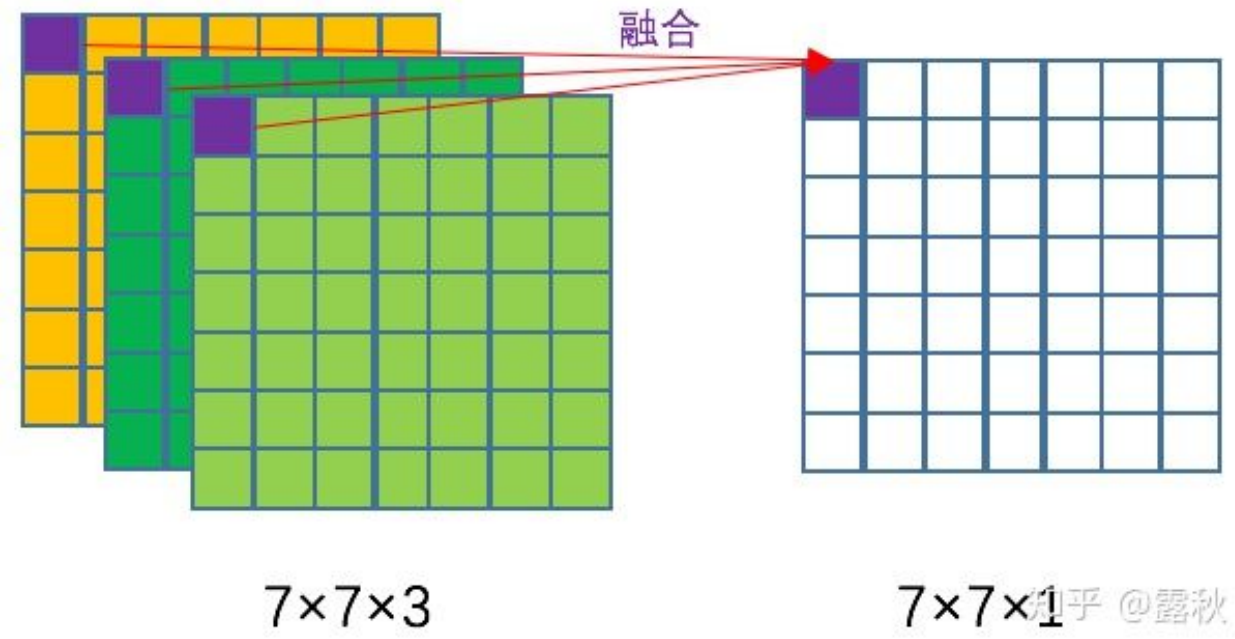
如果输入是 7×7×96 ，我们选择的卷积核为 1×1×48，即卷积核的大小为 1×1 个数为 48 。那么通过卷积之后得到的输出就为 7×7×48，可以看到输入的维度通过卷积之后被减少了。



如果输入是单通道的话，即 7×7×1，那么 1×1 的卷积操作其实就是对原特征的的缩放操作。

多通道融合

1×1 的卷积核不仅可以用于降维操作，还可以用于不同通道中的特征的融合，接跨通道特征融合。如，输入为 7×7×3，使用 1×1×1 的卷积核，得到的输出为 7×7×1，示意图如下：



多通道融合 注：此处有疑问，不知道多通道融合是不是也可指 在 Inception-3a 输出计算 章节中 Inception 输出的不同视野的组合?

减少参数

那么通过添加 1×1 的卷积核之后，参数到底比原始的并添加结构减少了多少呢？以论文中的下图为例：

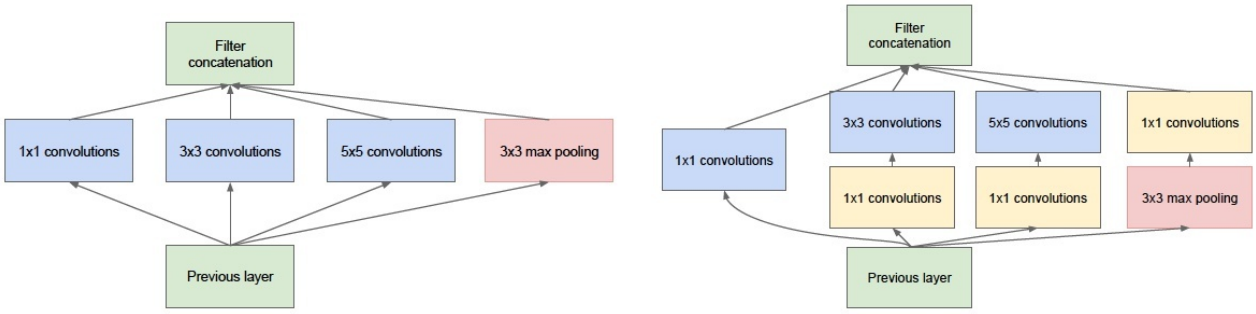


Figure 2: Inception module

左边是原始的版本，右边是添加了 1×1 卷积核的版本。此处以 Inception-3a 层为例，此层的输入为 28×28×192。每层的 filter 个数可以在 Table 1 中找到。

那么原始版本的每层参数个数为（MaxPool 不需要参数）：

- 1×1层：1×1×192×64
- 3×3层：3×3×192×128
- 5×5层：5×5×192×32

参数的总数为：387072

使用了 1×1 卷积核版本的参数个数为（MaxPool 不需要参数）：

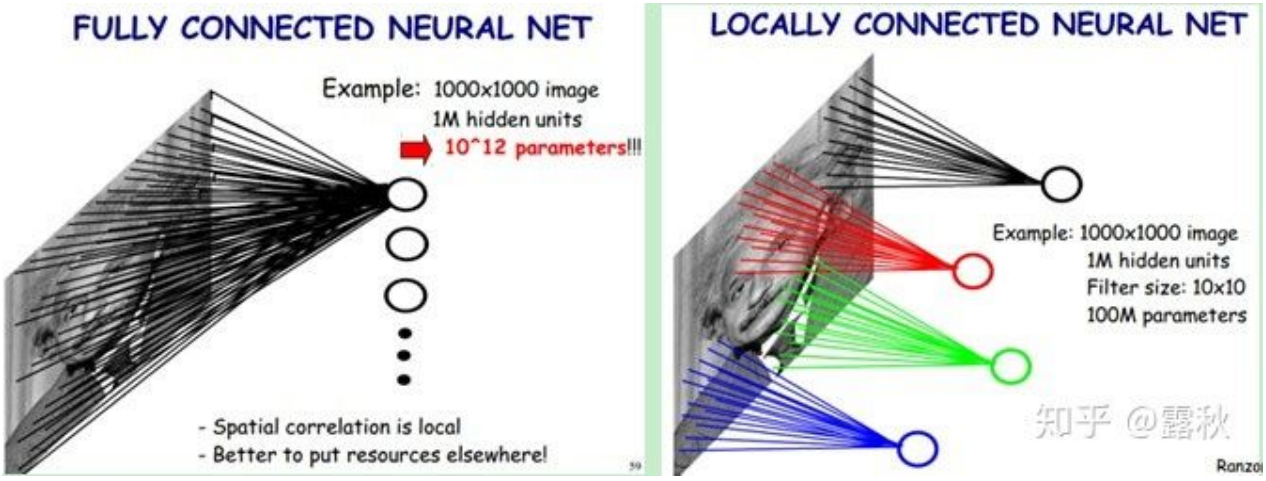
- # 1×1 层：1×1×192×64
- # 1×1 与 3×3： 1×1×192×96 + 3×3×96×128
- # 1×1 与 5×5： 1×1×192×16 + 5×5×16×32

- # MaxPool 与1×1：1×1×192×32 参数的总数为：163328

可以看到加了1×1 的卷积之后，网络中的参数大约减少了一半。

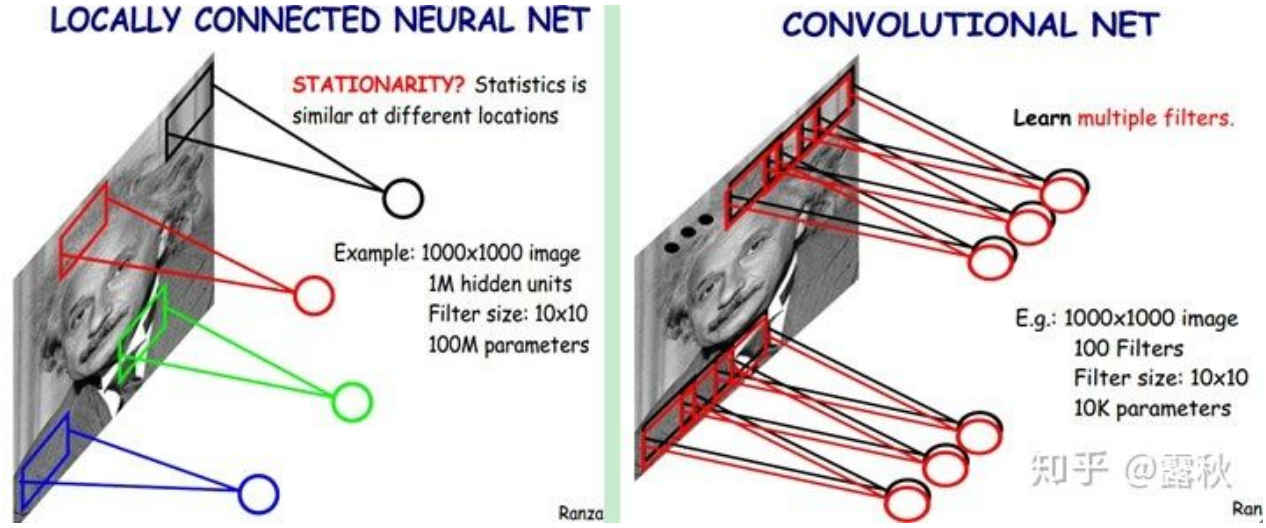
什么是稀疏性

在论文 Motivation and High Level Considerations 章节介绍了 Inception 产生的原因，Inception 就是为了解决在保留稀疏性的同时，又能利用密集矩阵的高计算性能。对其中的稀疏性产生了疑问，什么是稀疏性？最后看了一些资料才知道，其实稀疏性（sparsity）就是指局部连接（Local Connectivity），如下图：



左边是全连接，而右边是局部连接，通过局部连接大大的减少参数的数量。

然后在通过参数共享进一步减少参数的数量：



这里就可以看到，不同的术语表示的其实是一个意思，如果只知道其中的一个而不知道另一个，那么看到另一个的时候就会让你感到很迷惑，能意识到这些问题，会加快学习的速度。

其实 CNN 是对视觉皮层的模拟，下面是对《CNN学习笔记（1）稀疏连接和权值共享的理解》文章中CNN对visual cortex的模仿 章节的摘录：

> 视觉皮层包含了一系列复杂的细胞，这些细胞中的每个细胞只是对一个视觉区域内极小的一部分敏感，而对其他部分则可以视而不见，所以每一个这样的小部分区域我们称为receptive field（中文可以翻译为局部感受野吧）。而所有的 receptive field 的叠加就构成了 visual field，而这些cell在当中就扮演了一个local filter（局部滤波器）的角色。
> cell有两种类型，简单or复杂，简单的cell可以最大程度响应特定的边缘特征通过这些receptive field；复杂的cell相比而言有着更大的receptive field。
> 所以对应到 CNN 中，图像作为一个 visual field，而每一个 hidden layer 中 cells 也就是各种滤波器，每一个局部滤波器就是一个卷积核，通过局部滤波器对图像的 subset 产生响应，之后每个 cell 提取到的信息传递到更高层汇总后就是整个图像的信息。

总结

文中的主要想法是通过密集的块结构（Inception）来近似最优的稀疏结构（其实不是特别懂怎么近似了稀疏结构，有懂的可以交流下），从而在提高性能的基础上不增加计算量。

GooLeNet 在原本的 n 通道和 m 通道的两层之间添加一个 p 层（$n \times m > n \times p + p \times m \; if \; p << n, m$），降低了参数和计算量又符合 Hebbian principle。