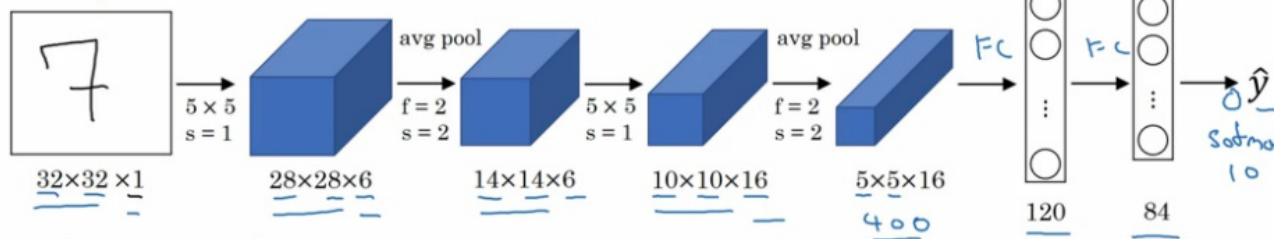


第四篇第二周

LeNet - 5



60K parameters.

$n_H, n_w \downarrow n_c \uparrow$

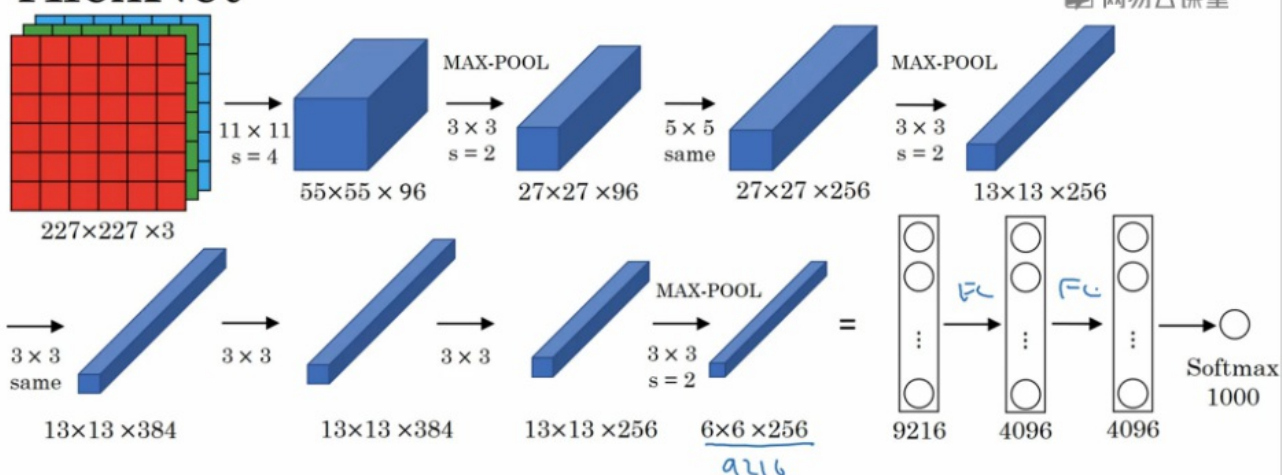
conv pool conv pool fc fc output

这种排列方式很常用



So this type of arrangement of layers is quite common.

AlexNet

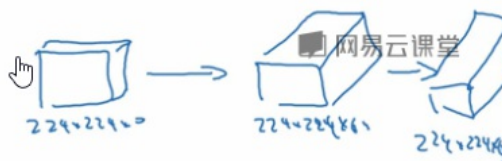
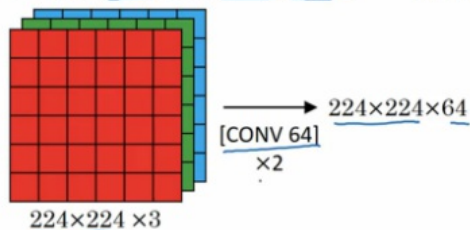


看它究竟是1000个可能的对象中的哪一个
which one of 1,000 classes the object could be.

VGG - 16

CONV = 3×3 filter, $s = 1$, same

MAX-POOL = 2×2 , $s = 2$

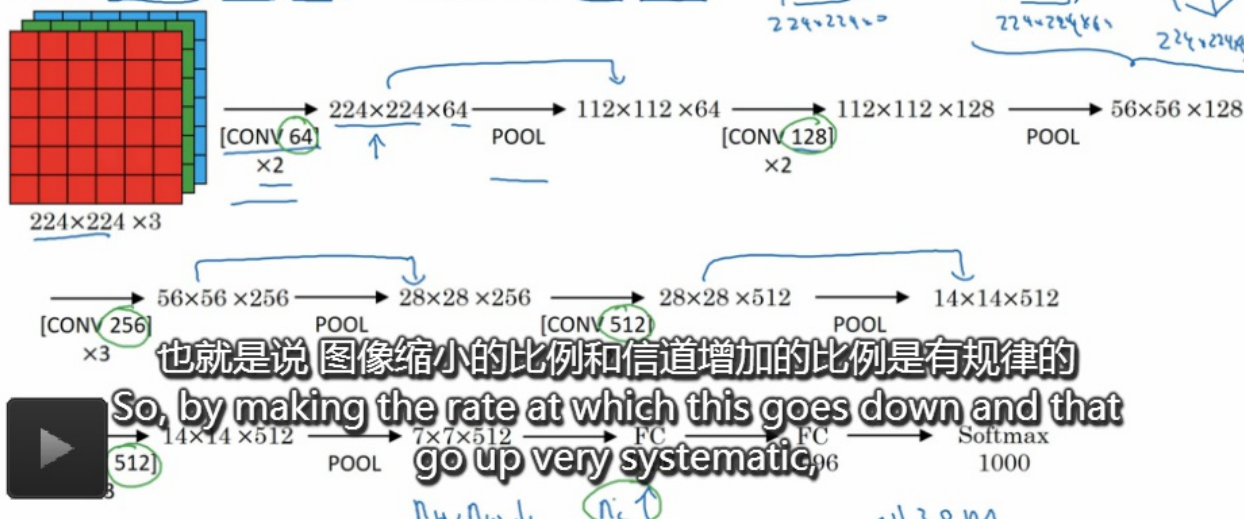


得到这样2个厚度为64的卷积层
So this Conv 64 times 2,

VGG - 16

CONV = 3×3 filter, $s = 1$, same

MAX-POOL = 2×2 , $s = 2$



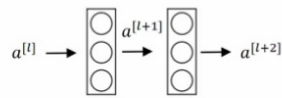
3、残差网络

非常非常深的神经网络是非常难训练的，因为存在梯度爆炸或者梯度消失问题，接下来我们学习跳远连接，它可以从某一网络层获得激活，然后迅速反馈给另外一层甚至神经网络的更深层，我们可以利用跳远连接构建能够训练深度网络的 resNet。

resNet是由残差块构建的，什么是残差块呢？

Residual block

网易云课



这是一个两层神经网络 在L层进行激活

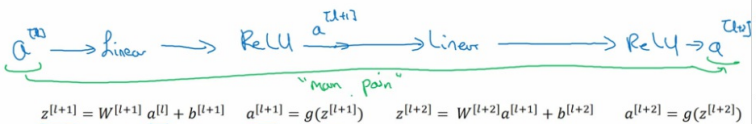
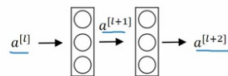


Here are two layers of a neural network where you start off with some activation in layer l,

计算过程是从 $a^{[l]}$ 开始，首先进行线性激活，然后通过relu激活函数得到 $a^{[l+1]}$ ，接下来再次进行线性激活，然后再次使用relu激活函数得到 $a^{[l+2]}$ ，换句话说，信息流从 $a^{[l]}$ 流到 $a^{[l+2]}$ ，需要经过上面的所有步骤，即这个网络的主路径：

Residual block

网易云课堂



$$z^{[l+1]} = W^{[l+1]} a^{[l]} + b^{[l+1]} \quad a^{[l+1]} = g(z^{[l+1]}) \quad z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]} \quad a^{[l+2]} = g(z^{[l+2]})$$



在残差网络中有一点变化
In a residual net, we're going to make a change to this,

在残差网络中有一点变化，我们将 $a^{[l]}$ 直接向后拷贝到神经网络的深层，在relu激活函数前加一个 $a^{[l]}$ ，

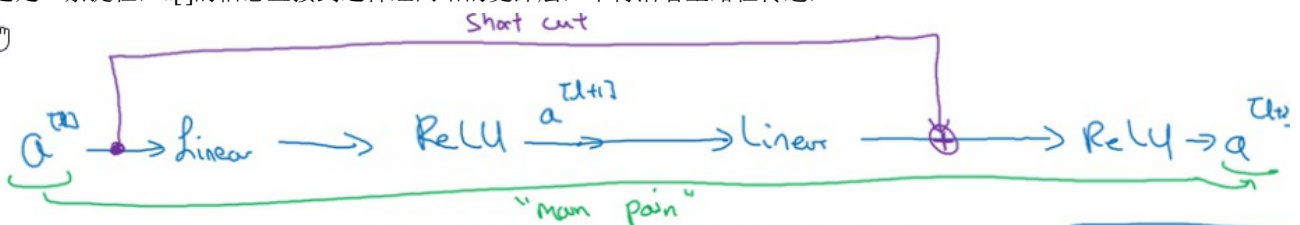


$$z^{[l+1]} = W^{[l+1]} a^{[l]} + b^{[l+1]} \quad \text{在ReLU非线性激活前加上 } a^{[l]} \quad z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]} \quad a^{[l+2]} = g(z^{[l+2]})$$



and just add $a^{[l]}$ before applying to nonlinearity, the ReLU nonlinearity,

这是一条捷径， $a^{[l]}$ 的信息直接到达神经网络的更深层，不再沿着主路径传递，

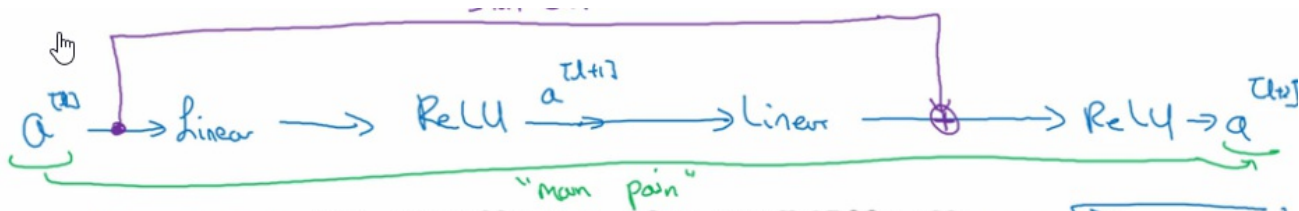


$$z^{[l+1]} = W^{[l+1]} a^{[l]} + b^{[l+1]} \quad \text{这就意味着最后这个等式去掉了} \quad z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]} \quad a^{[l+2]} = g(z^{[l+2]})$$



And what that means is that this last equation goes away,

取而代之的是另外一个relu的非线性函数，



$z^{[l+1]} = W^{[l+1]} a^{[l]} + b^{[l+1]}$ 取而代之的是另一个ReLU非线性函数

$$a^{[l+2]} = g(z^{[l+2]})$$

and we instead have that the output $a^{[l+2]}$ is the ReLU nonlinearity.

仍然会 $z^{[l+2]}$ 进行 z 函数处理，但这次要加上 $a^{[l]}$ 。也就是加上的这个 $a^{[l]}$ 产生的一个残差块，实际上这条捷径是在进行 ReLU 非线性激活之前加上的，而这里的每一个节点都执行了非线性函数和 relu 激活函数，所以 $a^{[l]}$ 插入的时机是在非线性激活之后 relu 激活之前。

$$z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$$

$$a^{[l+2]} = g(z^{[l+2]})$$

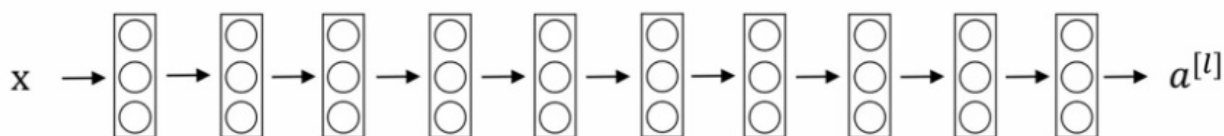
候

$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

也可以叫做跳远连接，是指 $a^{[l]}$ 跳过一层或者好几层，从而将信息传递到神经网络的更深层，使用残差块能够训练更深的神经网络，所以构建 resNet 网络就是通过将很多这样的残差块，堆积在一起，形成一个深度神经网络。

Residual Network

网易云课堂



这并不是一个残差网络 而是一个普通网络

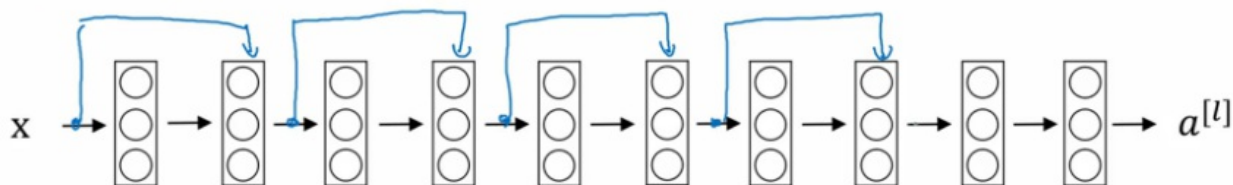
This is not a residual network, this called as a plain network.

把它变成 resNet 的方法就是加上所有的跳远连接。

Residual Network

"Plain network"

网易云课堂



正如前一张幻灯片中看到的 每两层增加一个捷径

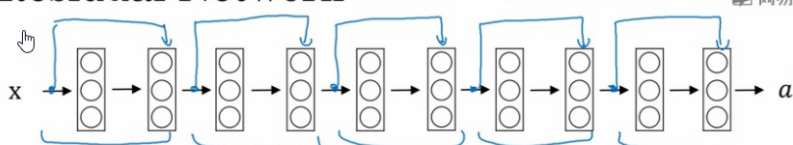
So every two layers ends up with that additional change that we saw on the previous slide

构成一个残差块

Residual Network

"Plain network"

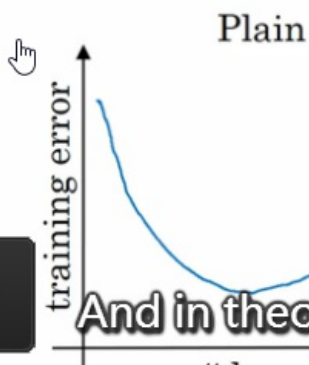
网易



如图所示 5个残差块连接在一起

So this picture shows five residual blocks stacked together,

构成一个残差网络，如果我们使用标准的优化算法去训练一个普通的网络，比如说梯度下降或者其他热门的优化算法，如果没有多余的残差，没有这些捷径，或者说跳远连接，凭经验，你会发现随着网络深度的加深，训练误差会先减小然后增加。



而理论上 随着网络深度的加深

And in theory as you make a neural network de

理论上随着网络深度的加深，应该训练的越来越好才对

Plain

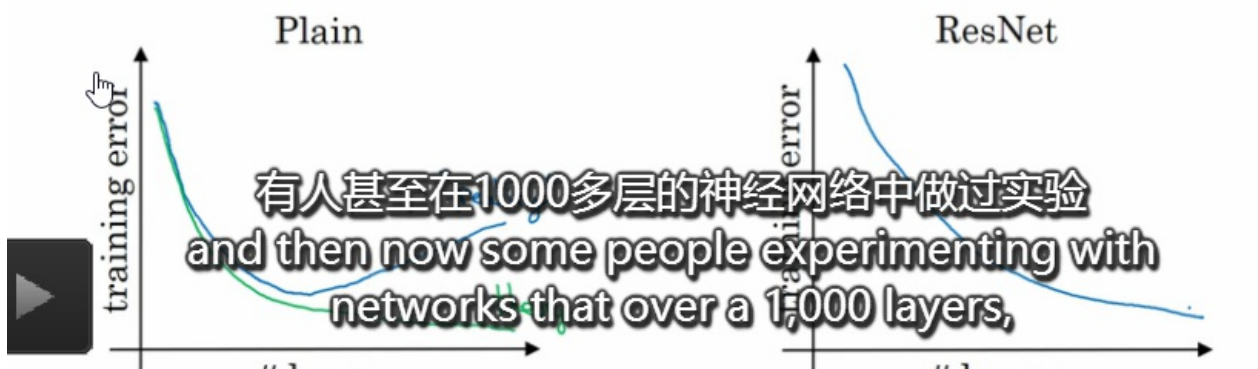


应该训练得越来越好才对

it should only do better and better on the tr right.

也就是说，理论上网络深度越深越好，但是实际上，如果没有残差网络，对于一个普通的网络来说，深度越深意味着，用优化算法越难训练，随着网络深度的加深，训练误差会越来越多，但是有了resNet就不一定了，即使网络再深，训练的

表现却不错，比如说误差会减小，就算是训练深度达到100层的网络也不例外看。



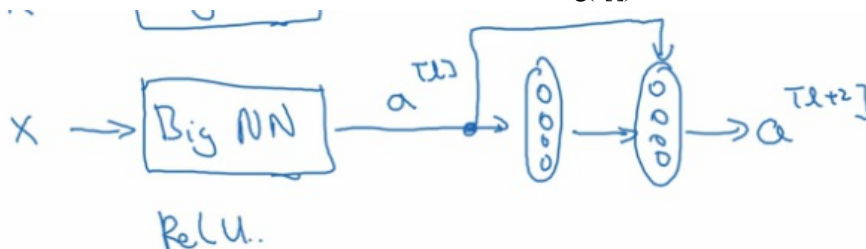
但是对于x的激活或者这些中间的激活，能够达到网络的更深层，这种方式确实有助于解决梯度消失或者梯度爆炸问题。

4、残差网络为什么有用。

残差网络为什么构建更深的网络时，还不降低他们在训练集上的效率

神经网络越深，他在训练集上的表现效率就会有所减弱。

$a[l+2] = g(w[l+2]a[l+1] + b[l+2] + a[l])$ 如果 $w[l+2]$ 为0，为了方便起见 b 也为0，前面几项就没了，因为为0.最后等于 $g(a[l])$ 也就是 $a[l]$ ，因为我们假设使用 **relu** 激活函数，并且所有激活值都是负的， $g(a[l])$ 是对应于非负数的 **relu** 函数。所以 $a[l+2] = a[l]$ 。



假设我们在整个网络中使用ReLU激活函数

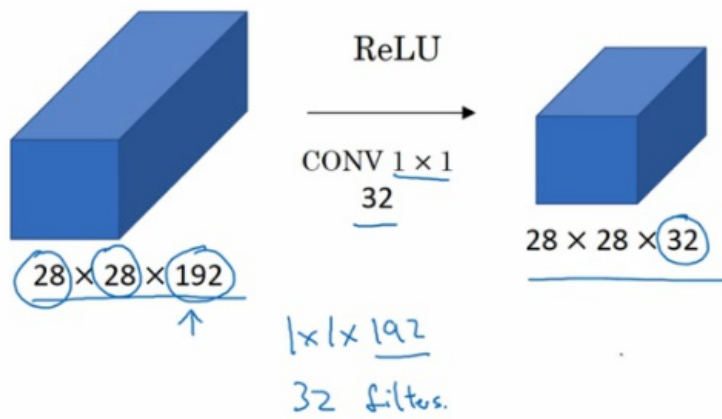
let's say throughout this network we're using the ReLU activation functions.

假设我们整个神经网络使用 **relu** 激活函数，所有激活值都大于等于0，包括输入X的非零异常值，因为 **relu** 激活函数输出的值要么为0要么为正数

展开这个表达式 $a[l+2] = g(W[l+2]a[l+1] + b[l+2] + a[l])$
And if we expand this out, this is equal to $g(W[l+2]) \times a[l+1] + b[l+2]$,

$a[l+2] = g(w[l+2]a[l+1] + b[l+2] + a[l])$ 如果使用正则化或者系数权值矩阵，他们会压缩 $W[l+2]$ 的值，如果对 b 进行权重衰减，亦可达到同样的效果

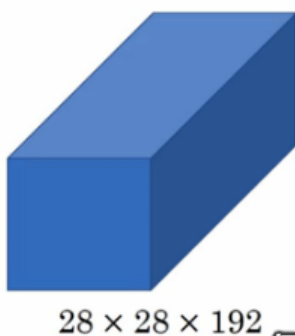
5、1*1的卷积神经网络，可以压缩通道的数目，



这就是压缩nC的方法
So this is a way to let you shrink nc as well,

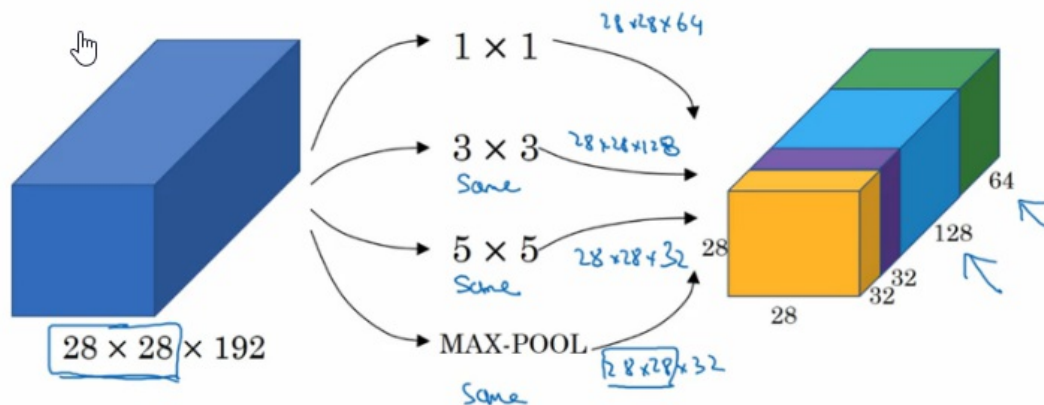
1*1的卷积层实现的重要功能就是，它给神经网络添加了一层非线性的激活函数，从而减少或者保持通道数不变，当然如果你愿意，你也可以增加通道数量。

6、Inception网络的作用代替你来决定采用1*3,3*3还是5*5之类的过滤器，虽然网络结构变得更复杂，但是网络表现确实非常好，例子：



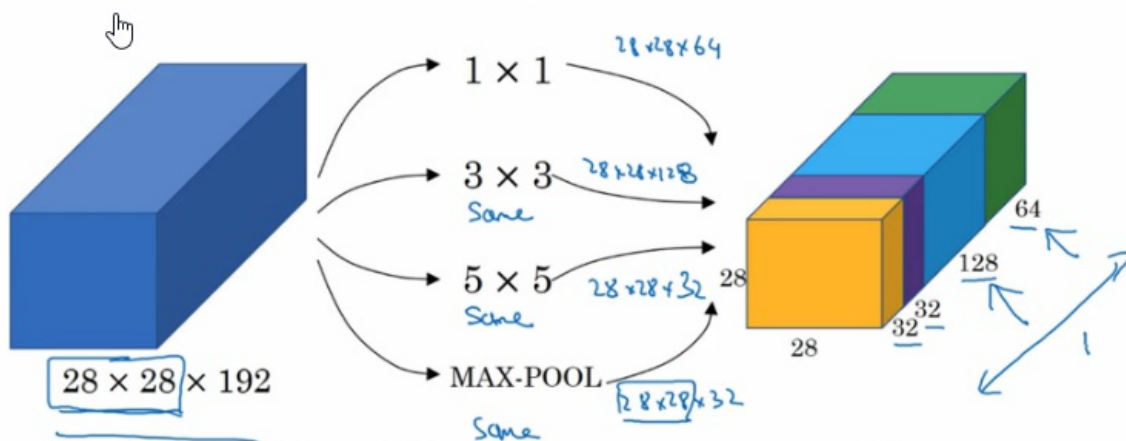
例如 这是28x28x192维度的输入层
Let's say for the sake of example that you have input $28 \times 28 \times 192$ dimensional volume.

28*28*192维度的输入层，Inception网络的作用是代替人工来确定卷积层中的过滤器类型，或者确定是否要构建卷积层或者池化层，演示开始，如果使用1*1的卷积，输出结果会是28*28*某个值，假设输出是28*28*64，如果是3*3的过滤器，那么输出是28*28*128，然后把第二个值堆在第一个值上，为了匹配维度，我们应用相同的卷积，输出维度依然是28*28，和输入维度相同，或许你会说，我希望提高网络的表现，用5*5的过滤器会更好，输出变成28*28*32，注意维度还是28*28，然后你就不想再卷积了，那么讲究用池化操作，得到一些不同的输出结果，我们也把它堆积起来，这里的池化输出是28*28*32，为了匹配所有维度，我们需要对最大池化使用padding，它是一种特殊的池化形式，因为如果输入维度是28*28，则输出相应维度也是28*28，然后再进行池化padding不变步长为1。



我们继续学习后面内容
but let's keep going,

有了这样的Inception模块，你就可以输出某个量，因为它累加了所有数据



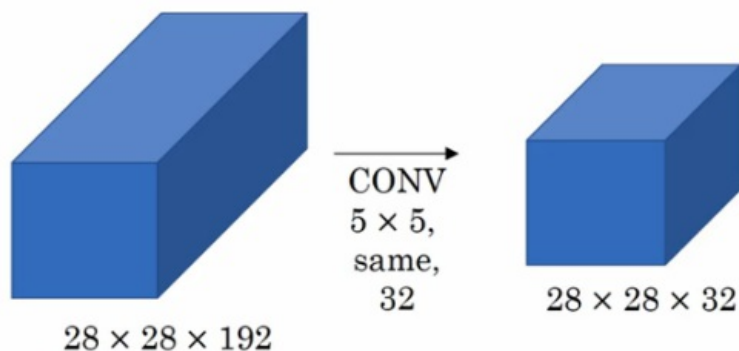
这里的最终输出为 $32+32+128+64$ 等于256
 $32+32+128+64$ that's equal to 256.

它的输入模块是 $28 \times 28 \times 129$ ，输出为 $28 \times 28 \times 256$ ，这就是Inception模块的核心内容，基本思想是Inception网络不需要人为决定使用哪个过滤器、是否需要池化，而是由网络自行确定这些参数，你可以给网络添加这些参数的所有可能的值，然后把这些输出连接起来，让网络自己学习它需要怎么样的参数，采用哪些过滤器组合，不难发现，Inception层有一个问题，就是计算代价成本，接下来就来计算 5×5 过滤器在该模块中的计算成本。



我们把重点集中在前一张幻灯片中的 5×5 的过滤器
So just focusing on the 5×5 part on the previous slide,

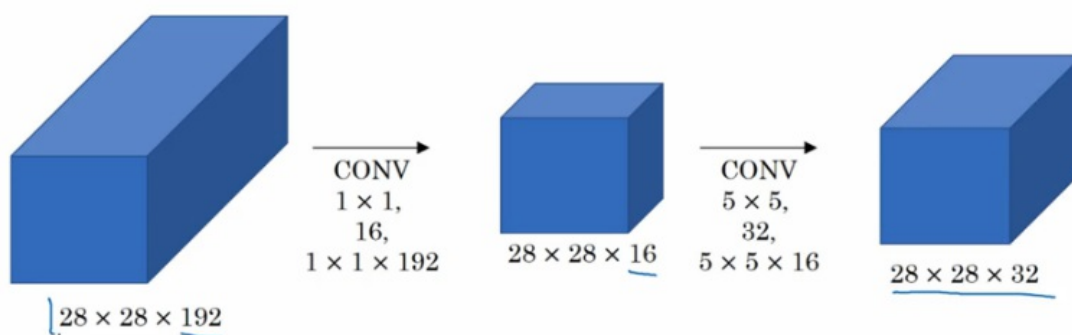
输入是 $28 \times 28 \times 192$



执行一个5x5卷积 它有32个过滤器 输出为28x28x32
and you implement a 5x5 same convolution with 32
filters output 28x28x32.

接下来计算 $28 \times 28 \times 32$ 的计算成本，它有32个过滤器，因为输出有32个信道，每个过滤器大小为 $5 \times 5 \times 192$ ，输出大小为 $28 \times 28 \times 32$ ，所以要计算 $28 \times 28 \times 192$ 个数，对于输出的每个值，都要执行 $5 \times 5 \times 192$ 次乘法运算，所以乘法运算的总数为，每个输出值所需要的乘法次数乘以输出值个数，把这些数相乘，结果为 $28 \times 28 \times 32 \times 28 \times 28 \times 192 = 1.2$ 亿，为了降低计算成本，我们用计算成本除以因子10，结果它减少了到了原来的10分之1，

还有一种方法是，先进行 1×1 的卷积，使得原来的信道减小到16，然后再进行 5×5 的卷积得到结果为 $28 \times 28 \times 32$ 。：



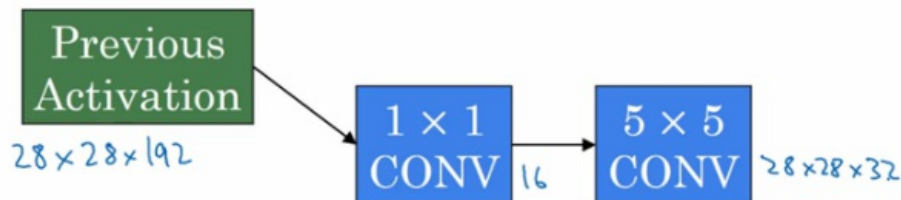
请注意 输入和输出的维度依然相同
So notice the input and output dimensions are still the
same.

Andrew Ng

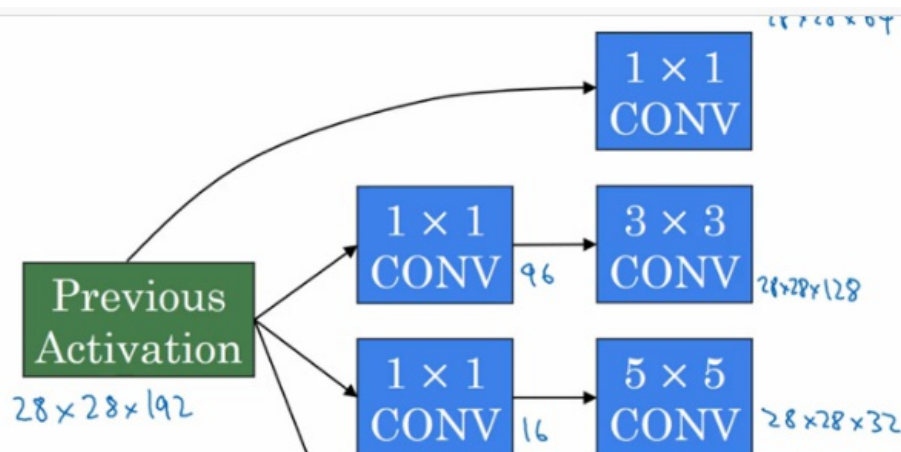
1×1 的计算成本为 $28 \times 28 \times 16 \times 192$, 240万， 5×5 过滤器的计算成本为 $28 \times 28 \times 32 \times 5 \times 5 \times 16$ 等于1000万，所以总成本为1240万个参数的计算量，总结一下，你在构建神经网络的时候，不想要决定使用 1×1 还是 3×3 还是 5×5 的过滤器的时候，Inception模块是最好的选择， 1×1 的卷积核，可以构建瓶颈层这样就可以大大减小计算的成本。你可能会问，仅仅大幅缩小表示层的规模，会不会影响神经网络的性能，事实证明，主要构建合理的瓶颈层，你既可以缩小表示层的规模，又不会降低网络性能，从而大大节省计算，这就是Inception模块的主要思想，

7、Inception网络

Inception module

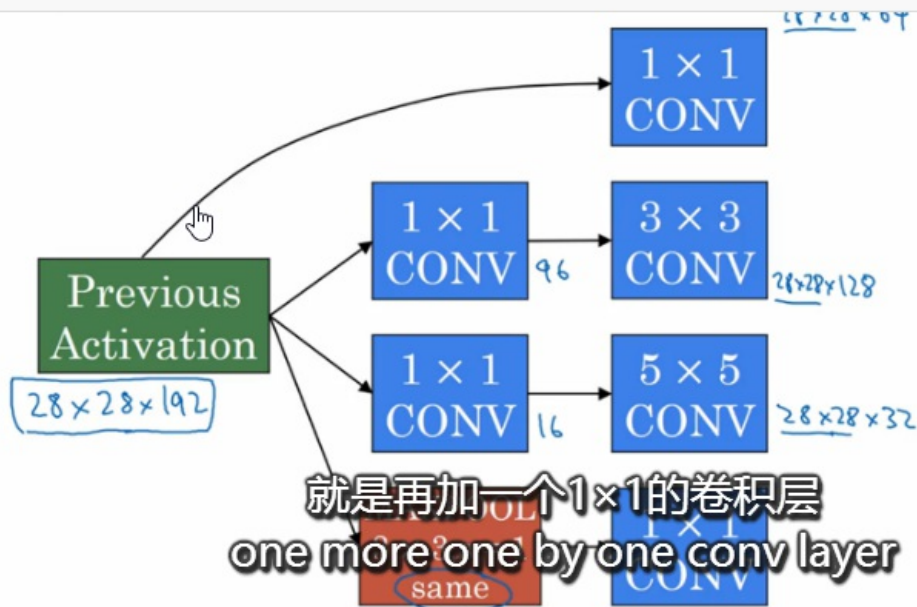


讲到的我们处理的例子
on the last slide of the previous video.



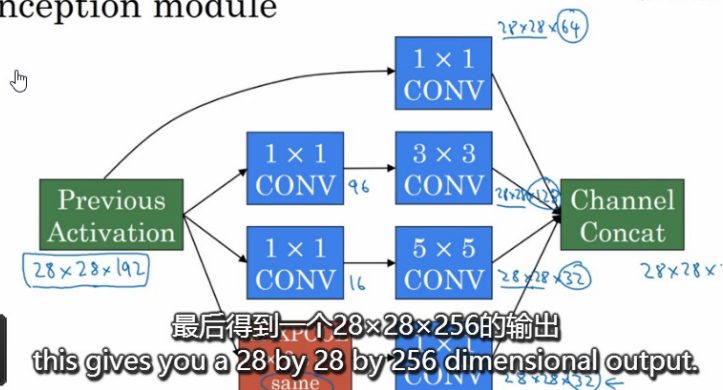
我们会使用same类型的
we are going to use a same
same

为了将最后每个模块都组合在一起，我们就要使得输出的长和宽的维度都保持一致，所以max pooling要进行填充same类型，但是如果你进行了最大池化，通道数还是不变的，还是192个通道，所以看起来它会有很多通道，我们实际要做的，就是再加上一个卷积层：

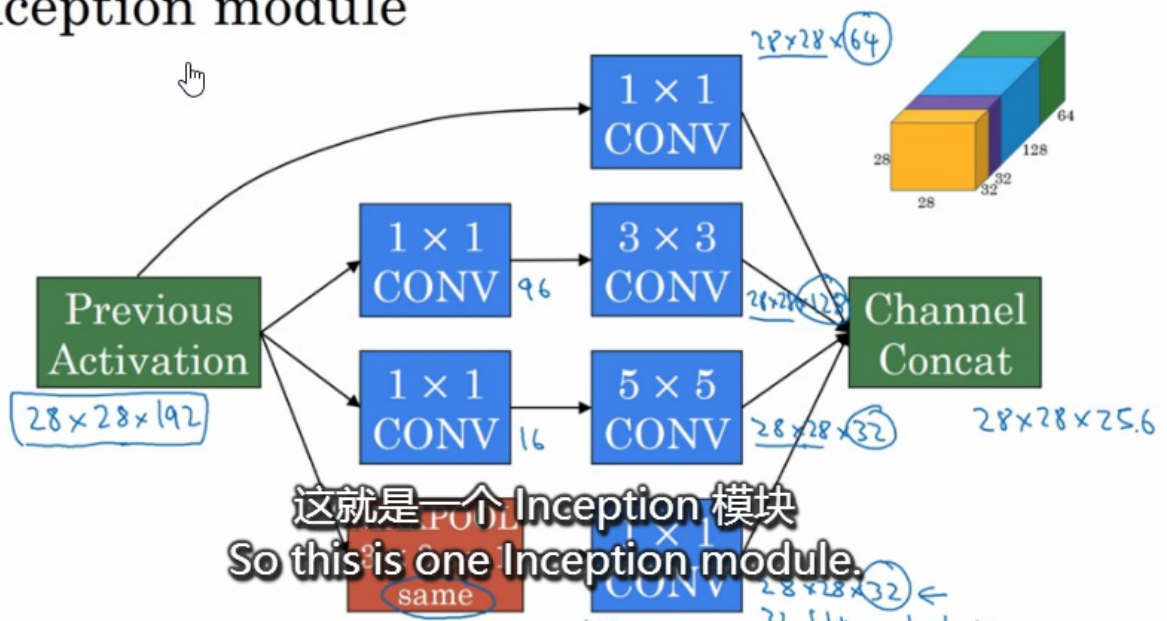


目的是为了将通道数缩小，也就是使用32个 $1 \times 1 \times 192$ 的过滤器，最终的输出结果为 $28 \times 28 \times 32$ 维度，所以最后通道数缩小为32个，所以这样就避免池化层占据大多数的通道，最后将这些方块全部拼接在一起，其实就是把各层的通道都加起来，最后得到一个 $28 \times 28 \times 256$ 维度的特征图

Inception module

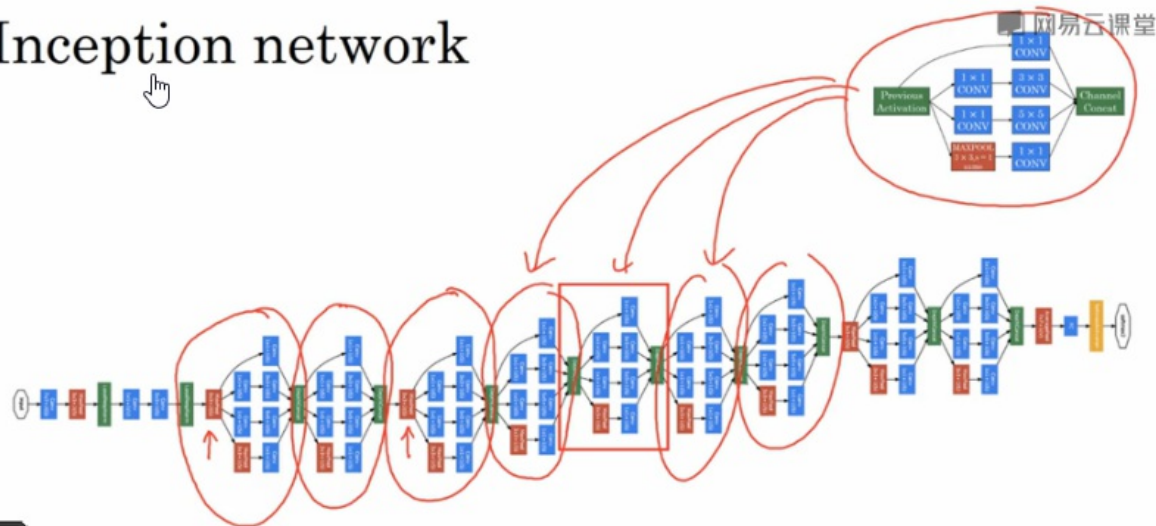


Inception module



而Inception所做的，就是将这些模块都组合再一起。

Inception network



其实论文中每个Inception层还有很多分支，其实就是softmax层这样就可以确保我们即使是在隐藏层或者中间层，我们仍然可以参与特征的计算，他们也能预测图片的分类，它在Inception网络中起到调节的作用，并且能防止网络发生过拟合。

8、如何使用开源的实现方案

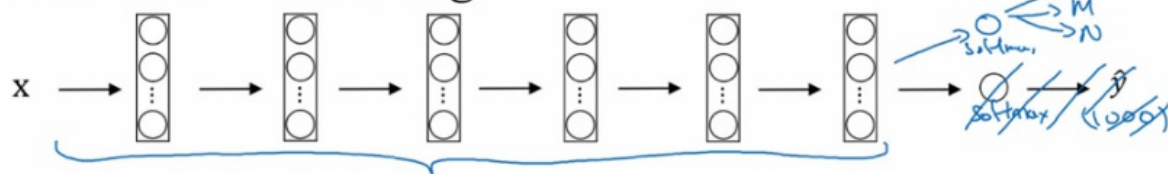
实用性建议：事实证明，很多神经网络复杂细致，难以复制，因为一些参数调整的细节问题，例如学习率的衰减等等，会影响性能，所以吴恩达建议使用论文上思想时，最好在网络上寻找一个开源的实现，因为如果你能得到作者的实现，比你从头开始实践要快得多，虽然从零开始是一个很好的锻炼，假设你对残差网络感兴趣，并想使用它，那么就直接找就可以了。你要开发视觉识别项目，选择一个你喜欢的框架，下载下来，这样做的优点在于，这些网络通常都需要训练很长的时间，这些可能都是别人预先训练好的网络，你就可以利用这些网络进行迁移学习，

9、迁移学习

如果你要做一个计算机视觉应用，相比从头训练权重，或者说随机初始化权重开始，如果你下载别人已经训练好的网络权重，你通常能够进展的相当快，用这个作为预训练，然后转到你感兴趣的任務上，计算机视觉的研究社区，非常喜欢，把许多数据集上传到网上，比如ImageNet等等数据集，你就可以下载别人训练好的网络权重参数，把它当做一个很好的初始化，用到你自己的神经网络上去，用迁移学习，把公共的数据集的知识迁移到你的问题上去，举一个例子：

比如你要训练一个猫的检测系统，用来检测你自己的宠物猫，但是现在你的训练数据集很小，你该怎么办呢，你可以从网上下载神经网络的开源实现，不仅把代码下载下来，也要把权重下载下来，有很多训练好的神经网络，你都可以下载下来，比如下ImageNet上训练好的1000个分类任务的网络，因此这个网络最后一定有一个softmax层，它可以输出1000个可能类别之一，你可以去掉这个softmax层，创建你自己的softmax单元

Transfer Learning



对于网络的其他层，我们使用它训练好的参数就可以了，也就是freeze其他所有层的参数，你只要训练softmax相关层的参数就可以了，最后你的softmax层有三个输出，通过使用其他人训练好的权重，你可能得到一个很好的性能，即使只有一个小的数据集，所以这个能加速训练的技巧就是，如果我们先计算第一层，计算特征或者激活值，然后把它保存到硬盘里，你所要做的就是将前面冻结的当做一个固定的函数，在这个神经网络的前半部分，取任意输入图像X，然后计算它的某个特征向量，这样你就训练一个很浅的softmax层，这样做的优点在于你就不用遍历整个训练集，再重新训练这个激活值了，如果你的数据集越来越大，那么你就可以考虑冻结更少的层，当你的训练集非常大的时候，你可以把他们训练好的权重值作为初始值，重新进行训练，

