[**深度学习与计算机视觉系列(10)\_细说卷积神经网络**](https://blog.csdn.net/yaoqiang2011/article/details/50542880)

作者：[寒小阳](http://blog.csdn.net/han_xiaoyang?viewmode=contents)   
时间：2016年1月。   
出处：<http://blog.csdn.net/han_xiaoyang/article/details/50542880>   
声明：版权所有，转载请联系作者并注明出处

## 1. 前言

前面九讲对神经网络的结构，组件，训练方法，原理等做了介绍。现在我们回到本系列的核心：**计算机视觉**，神经网络中的一种特殊版本在计算机视觉中使用最为广泛，这就是大家都知道的**卷积神经网络**。卷积神经网络和普通的神经网络一样，由『神经元』按层级结构组成，其间的权重和偏移量都是可训练得到的。同样是输入的数据和权重做运算，输出结果输入激励神经元，输出结果。从整体上看来，整个神经网络做的事情，依旧是对于像素级别输入的图像数据，用得分函数计算最后各个类别的得分，然后我们通过最小化损失函数来得到最优的权重。之前的博文中介绍的各种技巧和训练方法，以及注意事项，在这个特殊版本的神经网络上依旧好使。

既然提到卷积神经网络了，我们就来说说它的特殊之处，首先这里的卷积神经网络一般假定输入就是图片数据，也正是因为输入是图片数据，我们可以利用它的像素结构特性，去做一些假设来简化神经网络的训练复杂度(减少训练参数个数)。

## 2.卷积神经网总体结构一览

我们前面讲过的神经网络结构都比较一致，**输入层**和**输出层**中间夹着数层**隐藏层**，每一层都由多个神经元组成，层和层之间是全连接的结构，同一层的神经元之间没有连接。

卷积神经网络是上述结构的一种特殊化处理，因为对于图像这种数据而言，上面这种结构实际应用起来有较大的困难：就拿CIFAR-10举例吧，图片已经很小了，是32\*32\*3(长宽各32像素，3个颜色通道)的，那么在神经网络当中，我们只看隐藏层中的一个神经元，就应该有32\*32\*3=3072个权重，如果大家觉得这个权重个数的量还行的话，再设想一下，当这是一个包含多个神经元的多层神经网(假设n个)，再比如图像的质量好一点(比如是200\*200\*3的)，那将有200\*200\*3\*n= 120000n个权重需要训练，结果是拉着这么多参数训练，基本跑不动，跑得起来也是『气喘吁吁』，当然，最关键的是这么多参数的情况下，分分钟模型就过拟合了。别急，别急，一会儿我们会提到卷积神经网络的想法和简化之处。

卷积神经网络结构比较固定的原因之一，是图片数据本身的合理结构，类图像结构(200\*200\*3)，**我们也把卷积神经网络的神经元排布成 width\*height\*depth的结构**，也就是说这一层总共有width\*height\*depth个神经元，如下图所示。举个例子说，CIFAR-10的输出层就是1\*1\*10维的。另外我们后面会说到，**每一层的神经元，其实只和上一层里某些小区域进行连接，而不是和上一层每个神经元全连接**。

## 3.卷积神经网络的组成层

在卷积神经网络中，有3种最主要的层：

* 卷积运算层
* pooling层
* 全连接层

一个完整的神经网络就是由这三种层叠加组成的。

**结构示例**   
我们继续拿CIFAR-10数据集举例，一个典型的该数据集上的卷积神经网络分类器应该有[INPUT - CONV - RELU - POOL - FC]的结构，具体说来是这样的：

* INPUT[32\*32\*3]包含原始图片数据中的全部像素，长宽都是32，有RGB 3个颜色通道。
* CONV卷积层中，没个神经元会和上一层的若干小区域连接，计算权重和小区域像素的内积，举个例子可能产出的结果数据是[32\*32\*12]的。
* RELU层，就是神经元激励层，主要的计算就是*max*(0,*x*)
* ，结果数据依旧是[32\*32\*12]。
* POOLing层做的事情，可以理解成一个下采样，可能得到的结果维度就变为[16\*16\*12]了。
* 全连接层一般用于最后计算类别得分，得到的结果为[1\*1\*10]的，其中的10对应10个不同的类别。和名字一样，这一层的所有神经元会和上一层的所有神经元有连接。

这样，卷积神经网络作为一个中间的通道，就一步步把原始的图像数据转成最后的类别得分了。有一个点我们要提一下，刚才说到了有几种不同的神经网络层，**其中有一些层是有待训练参数的，另外一些没有。**详细一点说，卷积层和全连接层包含权重和偏移的；而RELU和POOLing层只是一个固定的函数运算，是不包含权重和偏移参数的。不过POOLing层包含了我们手动指定的超参数，这个我们之后会提到。

总结一下：

* 一个卷积神经网络由多种不同类型的层(卷几层/全连接层/RELU层/POOLing层等)叠加而成。
* 每一层的输入结构是3维的数据，计算完输出依旧是3维的数据。
* 卷积层和全连接层包含训练参数，RELU和POOLing层不包含。
* 卷积层，全连接层和POOLing层包含超参数，RELU层没有。

下图为CIFAR-10数据集构建的一个卷积神经网络结构示意图：

既然有这么多不同的层级结构，那我们就展开来讲讲：

### 3.1 卷积层

说起来，这是卷积神经网络的核心层(从名字就可以看出来对吧-\_-||)。

#### 3.1.1 卷积层综述

**直观看来**，卷积层的参数其实可以看做，一系列的可训练/学习的过滤器。在前向计算过程中，我们输入一定区域大小(width\*height)的数据，和过滤器点乘后等到新的二维数据，然后滑过一个个滤波器，组成新的3维输出数据。而我们可以理解成每个过滤器都只关心过滤数据小平面内的部分特征，当出现它学习到的特征的时候，就会呈现激活/activate态。

**局部关联度**。这是卷积神经网络的独特之处其中之一，我们知道在高维数据(比如图片)中，用全连接的神经网络，实际工程中基本是不可行的。**卷积神经网络中每一层的神经元只会和上一层的一些局部区域相连，这就是所谓的局部连接性**。你可以想象成，上一层的数据区，有一个滑动的窗口，只有这个窗口内的数据会和下一层神经元有关联，当然，这个做法就要求我们手动敲定一个超参数:窗口大小。通常情况下，这个窗口的长和宽是相等的，我们把长x宽叫做receptive field。实际的计算中，这个窗口是会『滑动』的，会近似覆盖图片的所有小区域。

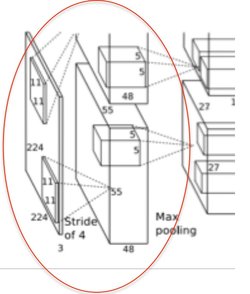
举个实例，CIFAR-10中的图片输入数据为[32\*32\*3]的，如果我们把receptive field设为5\*5，那receptive field的data都会和下一层的神经元关联，所以共有5\*5\*3=75个权重，注意到最后的3依旧代表着RGB 3个颜色通道。

如果不是输入数据层，中间层的data格式可能是[16\*16\*20]的，假如我们取3\*3的receptive field，那单个神经元的权重为3\*3\*20=180。

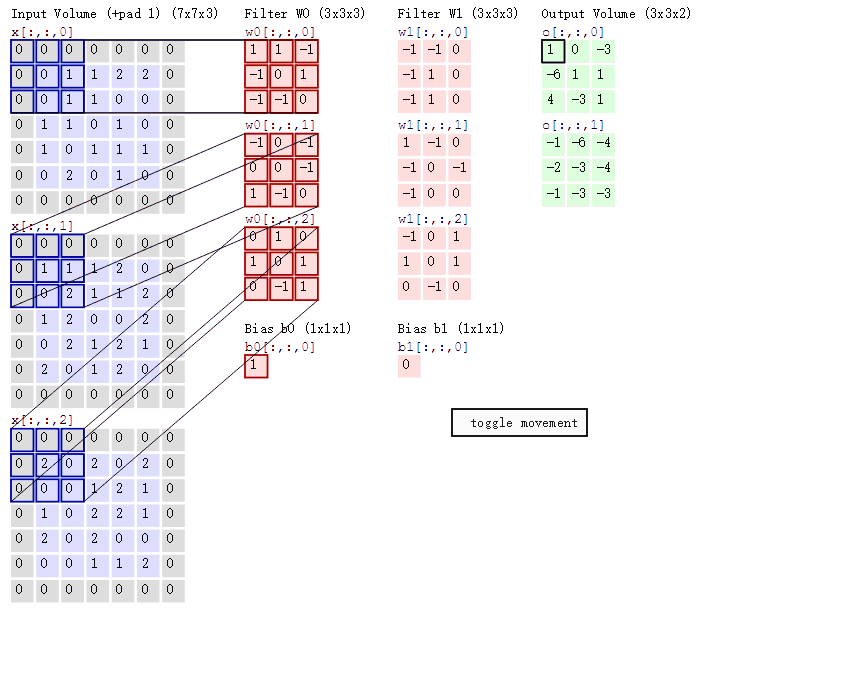
**局部关联细节**。我们刚才说到卷积层的局部关联问题，这个地方有一个receptive field，也就是我们直观理解上的『滑动数据窗口』。从输入的数据到输出数据，有三个超参数会决定输出数据的维度，分别是**深度/depth，步长/stride 和 填充值/zero-padding**：

1. 所谓深度/depth，简单说来指的就是卷积层中和上一层同一个输入区域连接的神经元个数。这部分神经元会在遇到输入中的不同feature时呈现activate状态，举个例子，如果这是第一个卷积层，那输入到它的数据实际上是像素值，不同的神经元可能对图像的边缘。轮廓或者颜色会敏感。
2. 所谓步长/stride，是指的窗口从当前位置到下一个位置，『跳过』的中间数据个数。比如从图像数据层输入到卷积层的情况下，也许窗口初始位置在第1个像素，第二个位置在第5个像素，那么stride=5-1=4.
3. 所谓zero-padding是在原始数据的周边补上0值的圈数。(下面第2张图中的样子)

这么解释可能理解起来还是会有困难，我们找两张图来对应一下这三个量：



这是解决ImageNet分类问题用到的卷积神经网络的一部分，我们看到卷积层直接和最前面的图像层连接。图像层的维度为[227\*227\*3]，而receptive field设为11\*11，图上未标明，但是滑动窗口的步长stride设为4，深度depth为48+48=96(这是双GPU并行设置)，边缘没有补0，因此zero-padding为0，因此窗口滑完一行，总共停留次数为**(data\_len-receptive\_field\_len+2\*zero-padding)/stride+1**=(227-11+2\*0)/4+1=55，因为图像的长宽相等，因此纵向窗口数也是55，最后得到的输出数据维度为55\*55\*96维。



这是一张动态的卷积层计算图，图上的zero-padding为1，所以大家可以看到数据左右各补了一行0，窗口的长宽为3，滑动步长stride为2。

**关于zero-padding**，补0这个操作产生的根本原因是，为了保证窗口的滑动能从头刚好到尾。举个例子说，上2图中的上面一幅图，因为**(data\_len-receptive\_field\_len+2\*zero-padding)/stride**刚好能够整除，所以窗口左侧贴着数据开始位置，滑到尾部刚好窗口右侧能够贴着数据尾部位置，因此是不需要补0的。而在下面那幅图中，如果滑动步长设为4，你会发现第一次计算之后，窗口就无法『滑动』了，而尾部的数据，是没有被窗口『看到过』的，因此补0能够解决这个问题。

**关于窗口滑动步长**。大家可以发现一点，窗口滑动步长设定越小，两次滑动取得的数据，重叠部分越多，但是窗口停留的次数也会越多，运算律大一些；窗口滑动步长设定越长，两次滑动取得的数据，重叠部分越少，窗口停留次数也越少，运算量小，但是从一定程度上说数据信息不如上面丰富了。

#### 3.1.2 卷积层的参数共享

首先得说卷积层的参数共享是一个非常赞的处理方式，它使得卷积神经网络的训练计算复杂度和参数个数降低非常非常多。就拿实际解决ImageNet分类问题的卷积神经网络结构来说，我们知道输出结果有55\*55\*96=290400个神经元，而每个神经元因为和窗口内数据的连接，有11\*11\*3=363个权重和1个偏移量。所以总共有290400\*364=105705600个权重。。。然后。。。恩，训练要累挂了。。。

因此我们做了一个大胆的假设，我们刚才提到了，每一个神经元可以看做一个filter，对图片中的数据窗区域做『过滤』。那既然是filter，我们干脆就**假设这个神经元用于连接数据窗的权重是固定的**，这意味着，对同一个神经元而言，不论上一层数据窗口停留在哪个位置，连接两者之间的权重都是同一组数。那代表着，上面的例子中的卷积层，我们只需要 神经元个数\*数据窗口维度=96\*11\*11\*3=34848个权重。

如果对应每个神经元的权重是固定的，那么整个计算的过程就可以看做，一组固定的权重和不同的数据窗口数据做内积的过程，这在数学上刚好对应『卷积』操作，这也就是卷积神经网的名字来源。另外，因为每个神经元的权重固定，它可以看做一个恒定的filter，比如上面96个神经元作为filter可视化之后是如下的样子：

需要说明的一点是，参数共享这个策略并不是每个场景下都合适的。有一些特定的场合，我们不能把图片上的这些窗口数据都视作作用等同的。一个很典型的例子就是人脸识别，一般人的面部都集中在图像的中央，因此我们希望，数据窗口滑过这块区域的时候，权重和其他边缘区域是不同的。我们有一种特殊的层对应这种功能，叫做**局部连接层/Locally-Connected Layer**

#### 3.1.3 卷积层的简单numpy实现

我们假定输入到卷积层的数据为X，加入X的维度为X.shape: (11,11,4)。假定我们的zero-padding为0，也就是左右上下不补充0数据，数据窗口大小为5，窗口滑动步长为2。那输出数据的长宽应该为(11-5)/2+1=4。假定第一个神经元对应的权重和偏移量分别为*W*0

和*b*0

，那我们就能算得，在第一行数据窗口停留的4个位置，得到的结果值分别为：

* V[0,0,0] = np.sum(X[:5,:5,:] \* W0) + b0
* V[1,0,0] = np.sum(X[2:7,:5,:] \* W0) + b0
* V[2,0,0] = np.sum(X[4:9,:5,:] \* W0) + b0
* V[3,0,0] = np.sum(X[6:11,:5,:] \* W0) + b0

注意上述计算过程中，\*运算符是对两个向量进行点乘的，因此*W*0

应该维度为(5,5,4)，同样你可以计算其他位置的计算输出值：

* V[0,0,1] = np.sum(X[:5,:5,:] \* W1) + b1
* V[1,0,1] = np.sum(X[2:7,:5,:] \* W1) + b1
* V[2,0,1] = np.sum(X[4:9,:5,:] \* W1) + b1
* V[3,0,1] = np.sum(X[6:11,:5,:] \* W1) + b1
* …

每一个神经元对应不同的一组W和b，在每个数据窗口停留的位置，得到一个输出值。

我们之前提到了卷积层在做的事情，是不断做权重和窗口数据的点乘和求和。因此我们也可以把这个过程整理成一个大的矩阵乘法。

1. 看看数据端，我们可以做一个操作**im2col**将数据转成一个可直接供神经元filter计算的大矩阵。举个例子说，输入是[227\*227\*3]的图片，而神经元权重为[11\*11\*3]，同时窗口移动步长为4，那我们知道数据窗口滑动过程中总共产生[(227-11)/4+1]\*[(227-11)/4+1]=55\*55=3025个局部数据区域，又每个区域包含11\*11\*3=363个数据值，因此我们想办法把原始数据重复和扩充成一个[363\*3025]的数据矩阵X\_col，就可以直接和filter进行运算了。
2. 对于filter端(卷积层)，假如厚度为96(有96个不同权重组的filter)，每个filter的权重为[11\*11\*3]，因此filter矩阵W\_row维度为[96\*363]
3. 在得到上述两个矩阵后，我们的输出结果即可以通过np.dot(W\_row, X\_col)计算得到，结果数据为[96\*3025]维的。

这个实现的弊端是，因为数据窗口的滑动过程中有重叠，因此我们出现了很多重复数据，占用内存较大。好处是，实际计算过程非常简单，如果我们用类似BLAS这样的库，计算将非常迅速。

另外，在反向传播过程中，其实卷积对应的操作还是卷积，因此实现起来也很方便。

### 3.2 Pooling层

简单说来，在卷积神经网络中，Pooling层是夹在连续的卷积层中间的层。它的作用也非常简单，就是\*\*逐步地压缩/减少数据和参数的量，也在一定程度上减小过拟合的现象。\*\*Pooling层做的操作也非常简单，就是将原数据上的区域压缩成一个值(区域最大值/MAX或者平均值/AVERAGE)，最常见的Pooling设定是，将原数据切成2\*2的小块，每块里面取最大值作为输出，这样我们就自然而然减少了75%的数据量。需要提到的是，除掉MAX和AVERAGE的Pooling方式，其实我们也可以设定别的pooling方式，比如L2范数pooling。说起来，历史上average pooling用的非常多，但是近些年热度降了不少，工程师们在实践中发现max pooling的效果相对好一些。

一个对Pooling层和它的操作直观理解的示意图为：

上图为Pooling层的一个直观示例，相当于对厚度为64的data，每一个切片做了一个下采样。下图为Pooling操作的实际max操作。

Pooling层(假定是MAX-Pooling)在反向传播中的计算也是很简单的，大家都知道如何去求max(x,y)函数的偏导（也是分段的）。

### 3.3 归一化层(Normalization Layer)

卷积神经网络里面有时候会用到各种各样的归一化层，尤其是早期的研究，经常能见到它们的身影，不过近些年来的研究表明，似乎这个层级对最后结果的帮助非常小，所以后来大多数时候就干脆拿掉了。

### 3.4 全连接层

这是我们在介绍神经网络的时候，最标准的形式，任何神经元和上一层的任何神经元之间都有关联，然后矩阵运算也非常简单和直接。现在的很多卷积神经网络结构，末层会采用全连接去学习更多的信息。

## 4. 搭建卷积神经网结构

从上面的内容我们知道，卷积神经网络一般由3种层搭建而成：卷积层，POOLing层(我们直接指定用MAX-Pooling)和全连接层。然后我们一般选用最常见的神经元ReLU，我们来看看有这些『组件』之后，怎么『拼』出一个合理的卷积神经网。

### 4.1 层和层怎么排

最常见的组合方式是，用ReLU神经元的卷积层组一个神经网络，同时在卷积层和卷积层之间插入Pooling层，经过多次的[卷积层]=>[Pooling层]叠加之后，数据的总体量级就不大了，这个时候我们可以放一层全连接层，然后最后一层和output层之间是一个全连接层。所以总结一下，最常见的卷积神经网结构为：

[输入层] => [[ReLU卷积层]\*N => [Pooling层]?]\*M => [ReLU全连接层]\*K => [全连接层]

解释一下，其中\\*操作代表可以叠加很多层，而[Pooling层]?表示Pooling层其实是可选的，可有可无。N和M是具体层数。比如说[输入层] -> [[ReLU卷积层]=>[ReLU卷积层]=>[Pooling层]]\*3 -> [ReLU全连接层]\*2 -> [全连接层]就是一个合理的深层的卷积神经网。

**『在同样的视野范围内，选择多层叠加的卷积层，而不是一个大的卷积层』**

这句话非常拗口，但这是实际设计卷积神经网络时候的经验，我们找个例子来解释一下这句话：如果你设计的卷积神经网在数据层有3层连续的卷积层，同时每一层滑动数据窗口为3\*3，第一层每个神经元可以同时『看到』3\*3的原始数据层，那第二层每个神经元可以『间接看到』(1+3+1)\*(1+3+1)=5\*5的数据层内容，第三层每个神经元可以『间接看到』(1+5+1)\*(1+5+1)=7\*7的数据层内容。那从最表层看，还不如直接设定滑动数据窗口为7\*7的，为啥要这么设计呢，我们来分析一下优劣：

* 虽然第三层对数据层的『视野』范围是一致的。但是单层卷积层加7\*7的上层滑动数据窗口，结果是这7个位置的数据，都是线性组合后得到最后结果的；而3层卷积层加3\*3的滑动数据窗口，得到的结果是原数据上7\*7的『视野』内数据多层非线性组合，因此这样的特征也会具备更高的表达能力。
* 如果我们假设所有层的『厚度』/channel数是一致的，为C，那7\*7的卷积层，会得到*C*×(7×7×*C*)=49*C*2

个参数，而3层叠加的3\*3卷积层只有3×(*C*×(3×3×*C*))=27*C*2

* 个参数。在计算量上后者显然是有优势的。
* 同上一点，我们知道为了反向传播方便，实际计算过程中，我们会在前向计算时保留很多中间梯度，3层叠加的3\*3卷积层需要保持的中间梯度要小于前一种情况，这在工程实现上是很有好处的。

### 4.2 层大小的设定

话说层级结构确定了，也得知道每一层大概什么规模啊。现在我们就来聊聊这个。说起来，每一层的大小(神经元个数和排布)并没有严格的数字规则，但是我们有一些通用的工程实践经验和系数：

* 对于输入层(图像层)，我们一般把数据归一化成2的次方的长宽像素值。比如CIFAR-10是32\*32\*3，STL-10数据集是64\*64\*3，而ImageNet是224\*224\*3或者512\*512\*3。
* 卷积层通常会把每个[滤子/filter/神经元]对应的上层滑动数据窗口设为3\*3或者5\*5，滑动步长stride设为1(工程实践结果表明stride设为1虽然比较密集，但是效果比较好，步长拉太大容易损失太多信息)，zero-padding就不用了。
* Pooling层一般采用max-pooling，同时设定采样窗口为2\*2。偶尔会见到设定更大的采样窗口，但是那意味着损失掉比较多的信息了。
* 比较重要的是，我们得预估一下内存，然后根据内存的情况去设定合理的值。我们举个例子，在ImageNet分类问题中，图片是224\*224\*3的，我们跟在数据层后面3个3\*3『视野窗』的卷积层，每一层64个filter/神经元,把padding设为1，那么最后每个卷积层的output都是[224\*224\*64]，大概需要1000万次对output的激励计算(非线性activation)，大概花费72MB内存。而工程实践里，一般训练都在GPU上进行，GPU的内存比CPU要吃紧的多，所以也许我们要稍微调动一下参数。比如AlexNet用的是11\*11的的视野窗，滑动步长为4。

### 4.3 典型的工业界在用卷积神经网络

几个有名的卷积神经网络如下：

* **LeNet**，这是最早用起来的卷积神经网络，Yann LeCun在论文[LeNet](http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf)提到。
* **AlexNet**，2012 ILSVRC比赛远超第2名的卷积神经网络，和LeNet的结构比较像，只是更深，同时用多层小卷积层叠加提到大卷积层。
* **ZF Net**，2013 ILSVRC比赛冠军，可以参考论文[ZF Net](http://arxiv.org/abs/1311.2901)
* **GoogLeNet**，2014 ILSVRC比赛冠军，Google发表的论文[Going Deeper with Convolutions](http://arxiv.org/pdf/1409.4842v1.pdf)有具体介绍。
* **VGGNet**，也是2014 ILSVRC比赛中的模型，有意思的是，即使这个模型当时在分类问题上的效果，略差于google的GoogLeNet，但是在很多图像转化学习问题(比如object detection)上效果奇好，它也证明卷积神经网的『深度』对于最后的效果有至关重要的作用。预训练好的模型在[pretrained model site](http://www.robots.ox.ac.uk/~vgg/research/very_deep/)可以下载。

具体一点说来，[VGGNet](http://www.robots.ox.ac.uk/~vgg/research/very_deep/)的层级结构和花费的内存如下：

INPUT: [224x224x3] memory: 224\*224\*3=150K weights: 0

CONV3-64: [224x224x64] memory: 224\*224\*64=3.2M weights: (3\*3\*3)\*64 = 1,728

CONV3-64: [224x224x64] memory: 224\*224\*64=3.2M weights: (3\*3\*64)\*64 = 36,864

POOL2: [112x112x64] memory: 112\*112\*64=800K weights: 0

CONV3-128: [112x112x128] memory: 112\*112\*128=1.6M weights: (3\*3\*64)\*128 = 73,728

CONV3-128: [112x112x128] memory: 112\*112\*128=1.6M weights: (3\*3\*128)\*128 = 147,456

POOL2: [56x56x128] memory: 56\*56\*128=400K weights: 0

CONV3-256: [56x56x256] memory: 56\*56\*256=800K weights: (3\*3\*128)\*256 = 294,912

CONV3-256: [56x56x256] memory: 56\*56\*256=800K weights: (3\*3\*256)\*256 = 589,824

CONV3-256: [56x56x256] memory: 56\*56\*256=800K weights: (3\*3\*256)\*256 = 589,824

POOL2: [28x28x256] memory: 28\*28\*256=200K weights: 0

CONV3-512: [28x28x512] memory: 28\*28\*512=400K weights: (3\*3\*256)\*512 = 1,179,648

CONV3-512: [28x28x512] memory: 28\*28\*512=400K weights: (3\*3\*512)\*512 = 2,359,296

CONV3-512: [28x28x512] memory: 28\*28\*512=400K weights: (3\*3\*512)\*512 = 2,359,296

POOL2: [14x14x512] memory: 14\*14\*512=100K weights: 0

CONV3-512: [14x14x512] memory: 14\*14\*512=100K weights: (3\*3\*512)\*512 = 2,359,296

CONV3-512: [14x14x512] memory: 14\*14\*512=100K weights: (3\*3\*512)\*512 = 2,359,296

CONV3-512: [14x14x512] memory: 14\*14\*512=100K weights: (3\*3\*512)\*512 = 2,359,296

POOL2: [7x7x512] memory: 7\*7\*512=25K weights: 0

FC: [1x1x4096] memory: 4096 weights: 7\*7\*512\*4096 = 102,760,448

FC: [1x1x4096] memory: 4096 weights: 4096\*4096 = 16,777,216

FC: [1x1x1000] memory: 1000 weights: 4096\*1000 = 4,096,000

TOTAL memory: 24M \* 4 bytes ~= 93MB / image (only forward! ~\*2 for bwd)

TOTAL params: 138M parameters

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25

有意思的是，大家会注意到，在VGGNet这样一个神经网络里，大多数的内存消耗在前面的卷积层，而大多数需要训练的参数却集中在最后的全连接层，比如上上面的例子里，全连接层有1亿权重参数，总共神经网里也就1.4亿权重参数。

### 4.4 考虑点

组一个实际可用的卷积神经网络最大的瓶颈是GPU的内存。毕竟现在很多GPU只有3/4/6GB的内存，最大的GPU也就12G内存，所以我们应该在设计卷积神经网的时候多加考虑：

* 很大的一部分内存开销来源于卷积层的激励函数个数和保存的梯度数量。
* 保存的权重参数也是内存的主要消耗处，包括反向传播要用到的梯度，以及你用momentum, Adagrad, or RMSProp这些算法时候的中间存储值。
* 数据batch以及其他的类似版本信息或者来源信息等也会消耗一部分内存。

## 5. 更多的卷积神经网络参考资料

* [DeepLearning.net tutorial](http://deeplearning.net/tutorial/lenet.html)是一个用Theano完整实现卷积神经网的教程。
* [cuda-convnet2](https://code.google.com/p/cuda-convnet2/)是多GPU并行化的实现。
* [ConvNetJS CIFAR-10 demo](http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html)允许你手动设定参数，然后直接在浏览器看卷积神经网络的结果。
* [Caffe](http://caffe.berkeleyvision.org/)，主流卷积神经网络开源库之一。
* [Example Torch 7 ConvNet](https://github.com/nagadomi/kaggle-cifar10-torch7)，在CIFAR-10上错误率只有7%的卷积神经网络实现。
* [Ben Graham’s Sparse ConvNet](https://www.kaggle.com/c/cifar-10/forums/t/10493/train-you-very-own-deep-convolutional-network/56310)，CIFAR-10上错误率只有4%的实现。
* [Face recognition for right whales using deep learning](http://deepsense.io/deep-learning-right-whale-recognition-kaggle/?from=singlemessage&isappinstalled=0#rd)，Kaggle看图识别濒临灭绝右鲸比赛的冠军队伍卷积神经网络。

## 参考资料与原文

[cs231n 卷积神经网络](http://cs231n.github.io/convolutional-networks/)