## 人工神经网络VS卷积神经网络[#](https://www.cnblogs.com/zhangfengxian/p/10674216.html#611884669)

* 参数太多，在cifar-10的数据集中，只有32\*32\*3，就会有这么多权重，如果说更大的图片，比如200\*200\*3就需要120000多个，这完全是浪费。
* 没有利用像素之间位置信息，对于图像识别任务来说，每个像素与周围的像素都是联系比较紧密的。
* 网络层数限制 我们知道网络层数越多其表达能力越强，但是通过梯度下降方法训练深度人工神经网络很困难，因为全连接神经网络的梯度很难传递超过3层。因此，我们不可能得到一个很深的全连接神经网络，也就限制了它的能力。

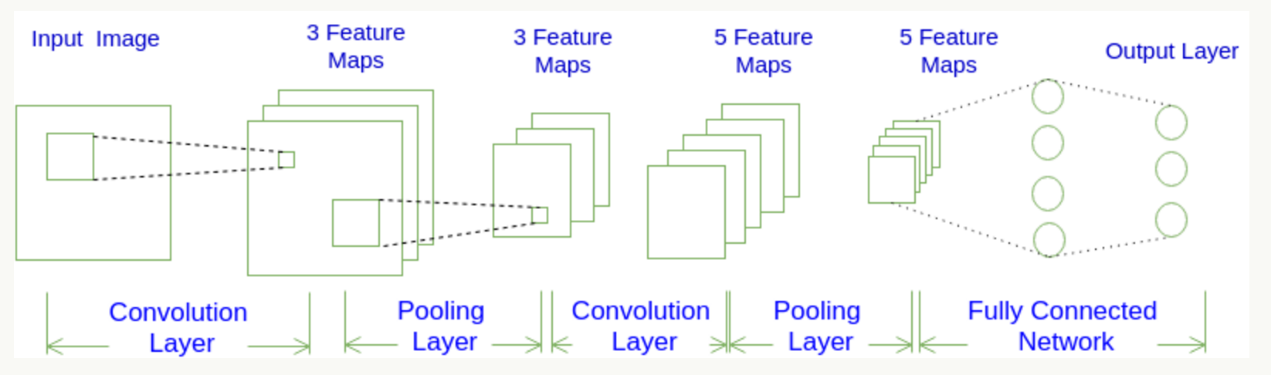
那么，卷积神经网络又是怎样解决这个问题的呢？主要有三个思路：

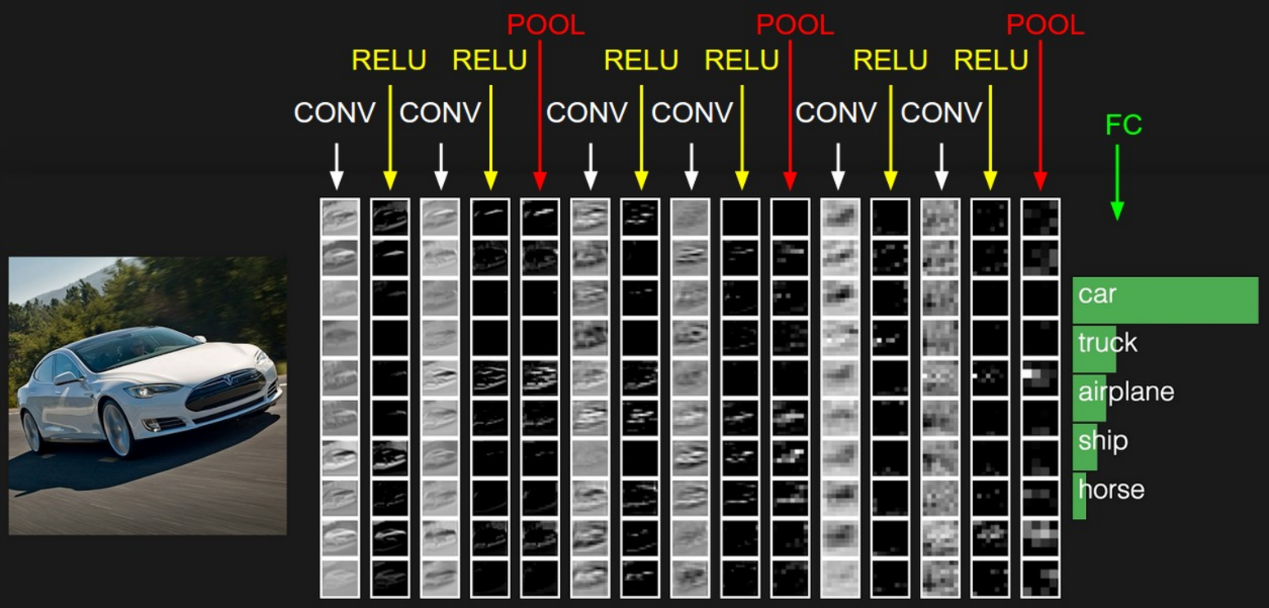
* 局部连接：这个是最容易想到的，每个神经元不再和上一层的所有神经元相连，而只和一小部分神经元相连。这样就减少了很多参数。
* 权值共享：一组连接可以共享同一个权重，而不是每个连接有一个不同的权重，这样又减少了很多参数。
* 下采样：可以使用Pooling来减少每层的样本数，进一步减少参数数量，同时还可以提升模型的鲁棒性。对于图像识别任务来说，卷积神经网络通过尽可能保留重要的参数，去掉大量不重要的参数，来达到更好的学习效果

## 卷积神经网络CNN[#](https://www.cnblogs.com/zhangfengxian/p/10674216.html#1482944464)

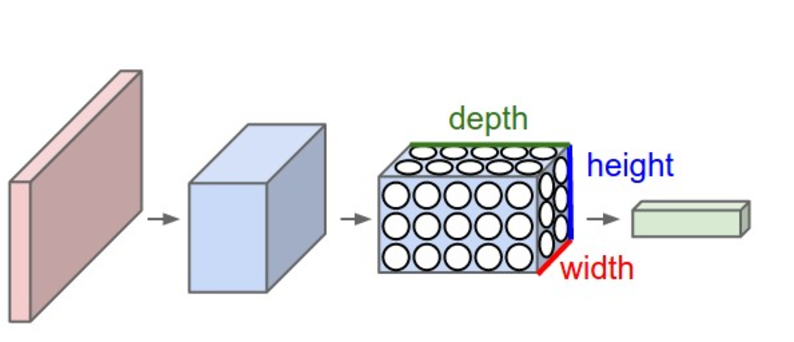
卷积神经网络与上一篇文章中的普通神经网络非常相似：它们由具有学习权重和偏差的神经元组成。每个神经元接收一些输入，执行点积，并且可选地以非线性跟随它。整个网络仍然表现出单一的可微分评分功能：从一端的原始图像像素到另一个类的分数。并且在最后（完全连接）层上它们仍然具有损失函数（例如SVM / Softmax），并且我们为学习正常神经网络开发的所有技巧/技巧仍然适用。

CNN每一层都通过可微分的函数将一个激活的值转换为另一个，一般来说CNN具有卷积层，池化层和完全连接层FC（正如在常规神经网络中所见）,在池化层之前一般会有个激活函数，我们将堆叠这些层，形成一个完整的架构。我们先看下大概的一个图：

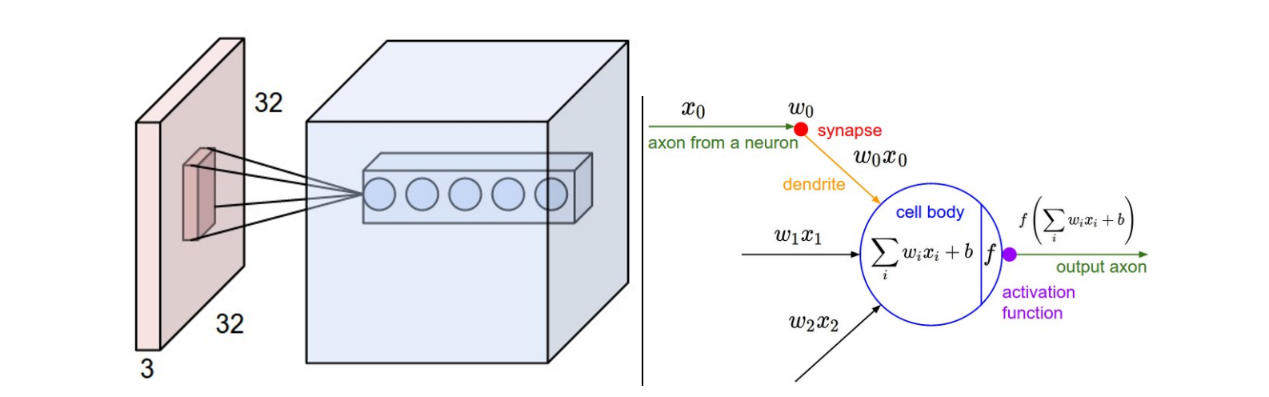
[](https://img2018.cnblogs.com/blog/1156642/201904/1156642-20190407093306370-1687769668.png)

[](https://img2018.cnblogs.com/blog/1156642/201904/1156642-20190407093323826-1895454065.png)

CNN它将一个输入3D体积变换为输出3D体积，正常的神经网络不同，CNN具有三维排列的神经元：宽度，高度，深度。

[](https://img2018.cnblogs.com/blog/1156642/201904/1156642-20190407093539594-1151662948.png)

### 卷积层[#](https://www.cnblogs.com/zhangfengxian/p/10674216.html#2483035974)

[](https://img2018.cnblogs.com/blog/1156642/201904/1156642-20190407094315006-1966090800.png)

#### 参数及结构[#](https://www.cnblogs.com/zhangfengxian/p/10674216.html#159844765)

四个超参数控制输出体积的大小：过滤器大小，深度，步幅和零填充。得到的每一个深度也叫一个Feature Map。

卷积层的处理：在卷积层有一个重要的就是过滤器大小（需要自己指定），若输入值是一个[32x32x3]的大小（例如RGB CIFAR-10彩色图像）。如果每个过滤器（Filter）的大小为5×5，则CNN层中的每个Filter将具有对输入体积中的[5x5x3]区域的权重，总共5 \*5\* 3 = 75个权重（和+1偏置参数），输入图像的3个深度分别与Filter的3个深度进行运算。请注意，沿着深度轴的连接程度必须为3，因为这是输入值的深度，并且也要记住这只是一个Filter。

* 假设输入卷的大小为[16x16x20]。然后使用3x3的示例接收字段大小，CNN中的每个神经元现在将具有总共3 \*3\* 20 = 180个连接到输入层的连接。

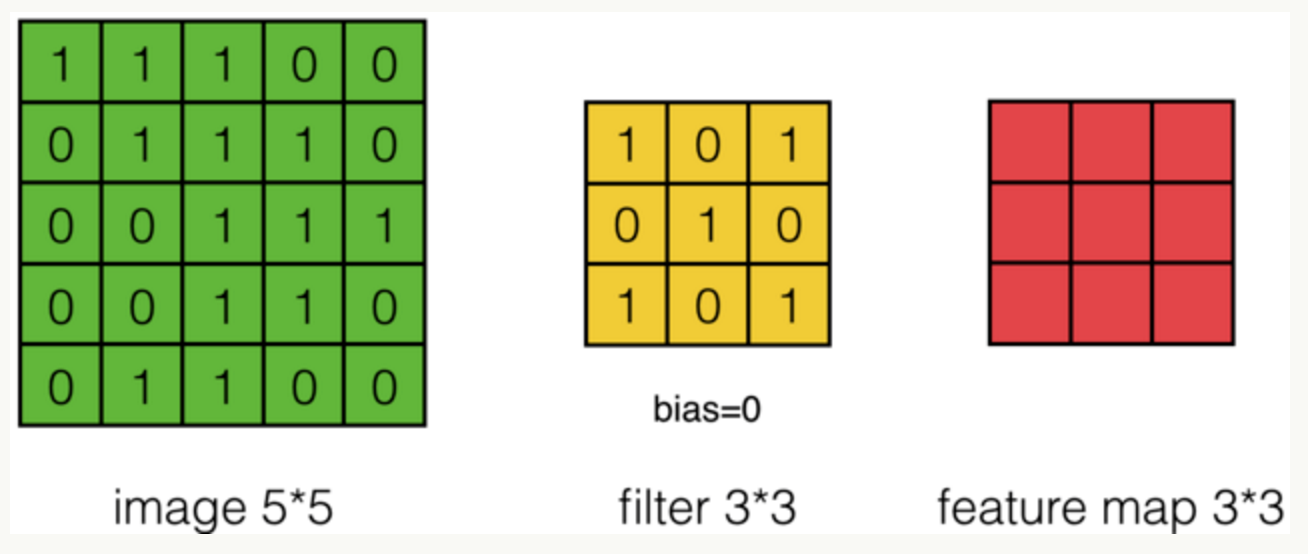
卷积层的输出深度：那么一个卷积层的输出深度是可以指定的，输出深度是由你本次卷积中Filter的个数决定。加入上面我们使用了64个Filter，也就是[5,5,3,64]，这样就得到了64个Feature Map，这样这64个Feature Map可以作为下一次操作的输入值。

卷积层的输出宽度：输出宽度可以通过特定算数公式进行得出，后面会列出公式。

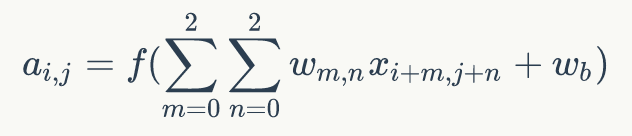
#### 卷积输出值的计算[#](https://www.cnblogs.com/zhangfengxian/p/10674216.html#3274683633)

我们用一个简单的例子来讲述如何计算卷积，然后，我们抽象出卷积层的一些重要概念和计算方法。

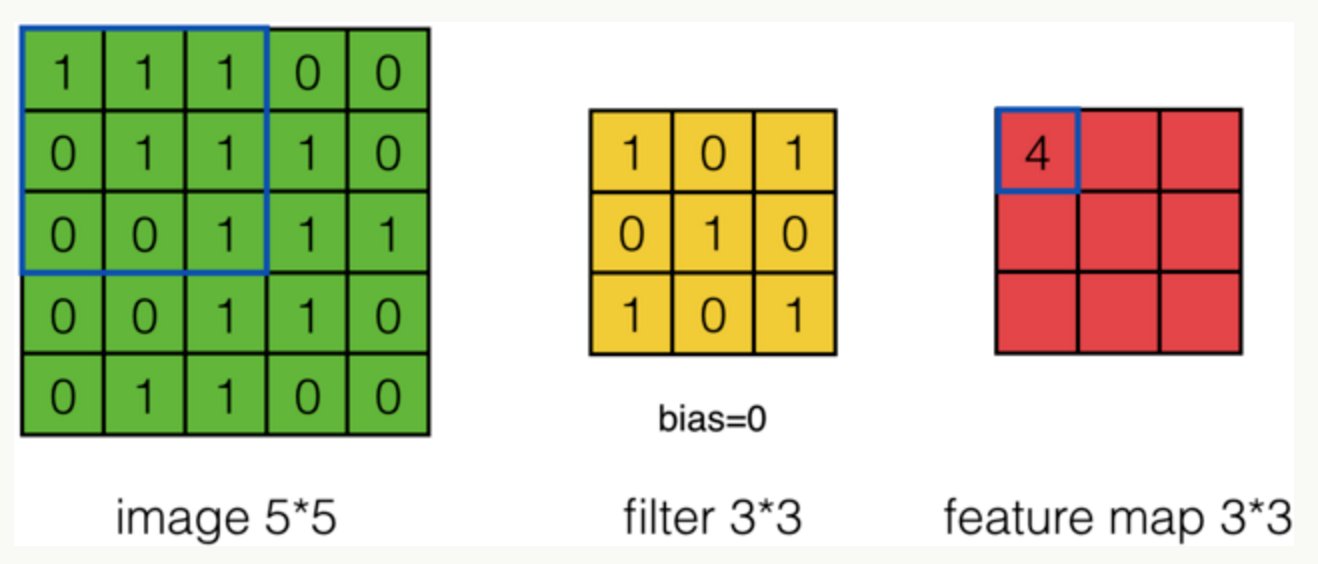
假设有一个5\*5的图像，使用一个3\*3的filter进行卷积，得到了到一个3\*3的Feature Map，至于得到3\*3大小，可以自己去计算一下。如下所示：

[](https://img2018.cnblogs.com/blog/1156642/201904/1156642-20190407095826552-1285321699.png)

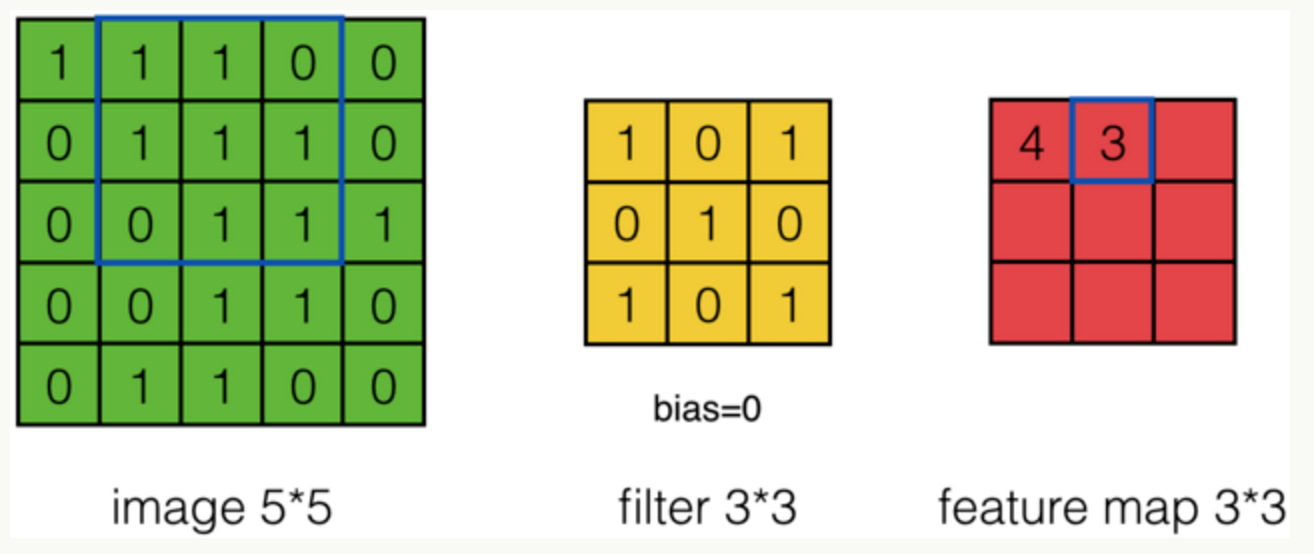
我们看下它的计算过程，首先计算公式如下：

[](https://img2018.cnblogs.com/blog/1156642/201904/1156642-20190407095915831-922141029.png)

根据计算的例子，第一次：

[](https://img2018.cnblogs.com/blog/1156642/201904/1156642-20190407100142496-1128565979.png)

第二次：

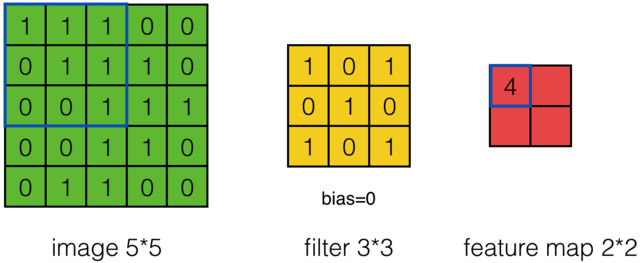
[](https://img2018.cnblogs.com/blog/1156642/201904/1156642-20190407100219028-507208874.png)

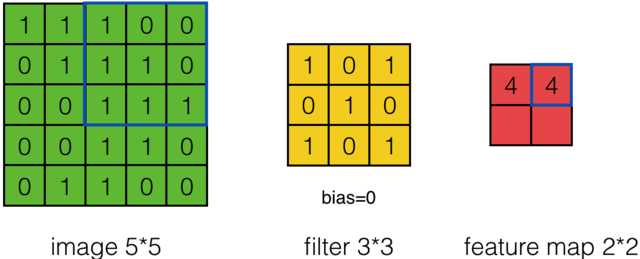
通过这样我们可以依次计算出Feature Map中所有元素的值。下面的动画显示了整个Feature Map的计算过程：

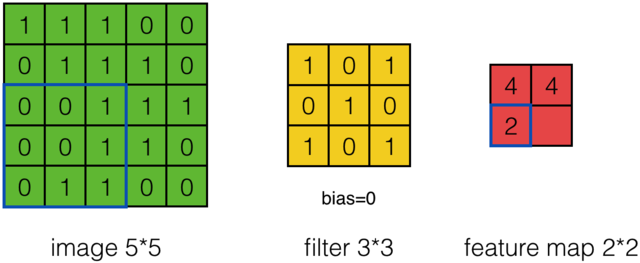
[](https://img2018.cnblogs.com/blog/1156642/201904/1156642-20190407100359513-1637399213.gif)

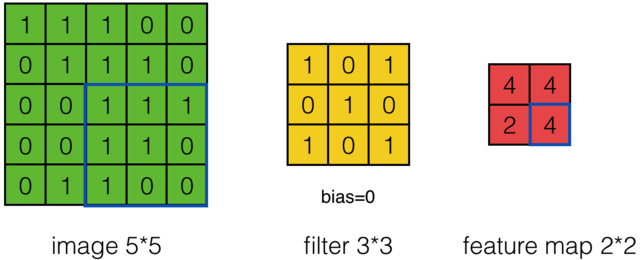
#### 步长[#](https://www.cnblogs.com/zhangfengxian/p/10674216.html#2552175920)

那么在卷积神经网络中有一个概念叫步长，也就是Filter移动的间隔大小。上面的计算过程中，步幅(stride)为1。步幅可以设为大于1的数。例如，当步幅为2时，我们可以看到得出2\*2大小的Feature Map，发现这也跟步长有关。Feature Map计算如下：

[](https://img2018.cnblogs.com/blog/1156642/201904/1156642-20190407100624120-697526989.png)

[](https://img2018.cnblogs.com/blog/1156642/201904/1156642-20190407100642172-711247617.png)

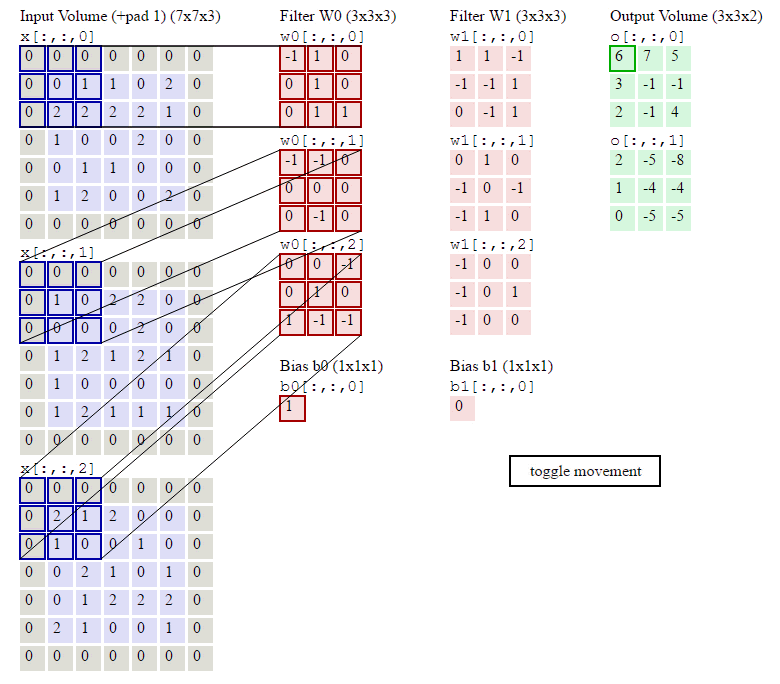
[](https://img2018.cnblogs.com/blog/1156642/201904/1156642-20190407100659217-1549973820.png)

[](https://img2018.cnblogs.com/blog/1156642/201904/1156642-20190407100708453-61604128.png)

#### 外围补充与多Filter[#](https://www.cnblogs.com/zhangfengxian/p/10674216.html#3263568468)

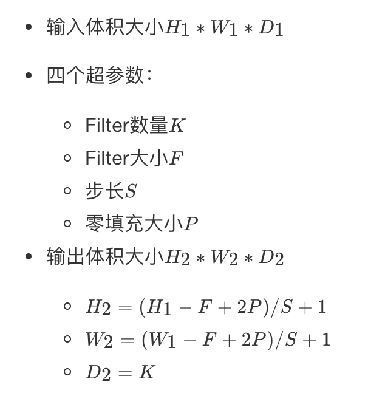
我们前面还曾提到，每个卷积层可以有多个filter。每个filter和原始图像进行卷积后，都可以得到一个Feature Map。因此，卷积后Feature Map的深度(个数)和卷积层的filter个数是相同的。

如果我们的步长移动与filter的大小不适合，导致不能正好移动到边缘怎么办？

[](https://img2018.cnblogs.com/blog/1156642/201904/1156642-20190407100854470-1302549228.gif)

以上就是卷积层的计算方法。这里面体现了局部连接和权值共享：每层神经元只和上一层部分神经元相连(卷积计算规则)，且filter的权值对于上一层所有神经元都是一样的。

#### 总结输出大小[#](https://www.cnblogs.com/zhangfengxian/p/10674216.html#675829956)

[](https://img2018.cnblogs.com/blog/1156642/201904/1156642-20190407101255301-970154070.png)

#### 卷积网络API[#](https://www.cnblogs.com/zhangfengxian/p/10674216.html#4175361580)

tf.nn.conv2d(input, filter, strides=, padding=, name=None)：

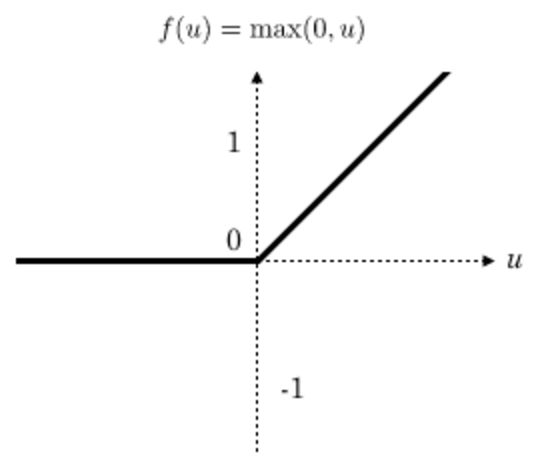
* 计算给定4-D input和filter张量的2维卷积
* input：给定的输入张量，具有[batch,heigth,width,channel]，类型为float32,64
* filter：指定过滤器的大小，[filter\_height, filter\_width, in\_channels, out\_channels]
* strides：strides = [1, stride, stride, 1],步长
* padding：“SAME”, “VALID”，使用的填充算法的类型，使用“SAME”。其中”VALID”表示滑动超出部分舍弃，“SAME”表示填充，使得变化后height,width一样大

### 新的激活函数-Relu[#](https://www.cnblogs.com/zhangfengxian/p/10674216.html#828831986)

一般在进行卷积之后就会提供给激活函数得到一个输出值。我们不使用sigmoid,softmax，而使用Relu。该激活函数的定义是：

[](https://img2018.cnblogs.com/blog/1156642/201904/1156642-20190407102904641-1076592887.png)

Relu函数如下：

[](https://img2018.cnblogs.com/blog/1156642/201904/1156642-20190407102919143-79360280.png)

**特点**

* 速度快，和sigmoid函数需要计算指数和倒数相比，relu函数其实就是一个max(0,x)，计算代价小很多
* 稀疏性，通过对大脑的研究发现，大脑在工作的时候只有大约5%的神经元是激活的，而采用sigmoid激活函数的人工神经网络，其激活率大约是50%。有论文声称人工神经网络在15%-30%的激活率时是比较理想的。因为relu函数在输入小于0时是完全不激活的，因此可以获得一个更低的激活率。

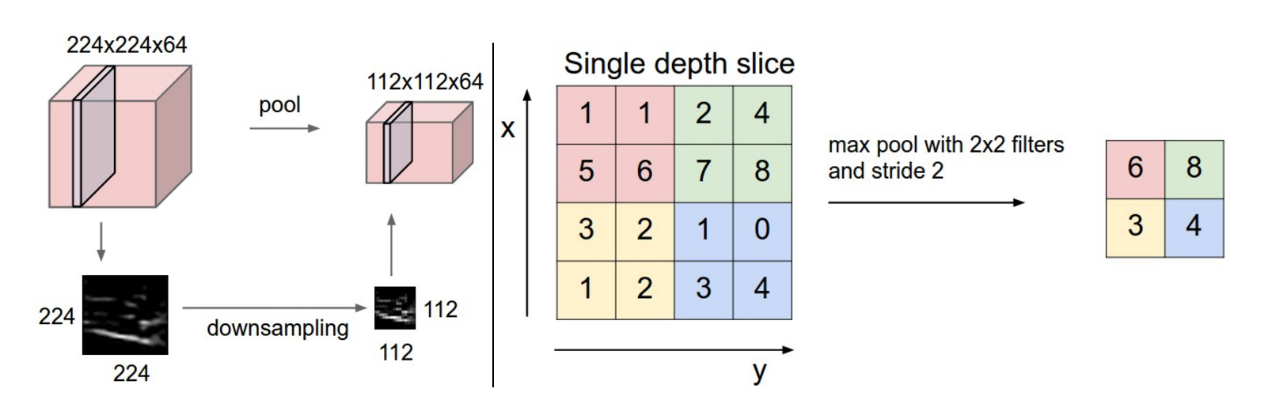
#### rule激活函数API[#](https://www.cnblogs.com/zhangfengxian/p/10674216.html#968239550)

tf.nn.relu(features, name=None)

* features:卷积后加上偏置的结果
* return:结果

### Pooling计算[#](https://www.cnblogs.com/zhangfengxian/p/10674216.html#4110373195)

Pooling层主要的作用是特征提取，通过去掉Feature Map中不重要的样本，进一步减少参数数量。Pooling的方法很多，最常用的是Max Pooling。

[](https://img2018.cnblogs.com/blog/1156642/201904/1156642-20190407103522255-126688587.png)

除了Max Pooing之外，常用的还有Mean Pooling——取各样本的平均值。对于深度为D的Feature Map，各层独立做Pooling，因此Pooling后的深度仍然为D。

#### Pooling API[#](https://www.cnblogs.com/zhangfengxian/p/10674216.html#3898219311)

tf.nn.max\_pool(value, ksize=, strides=, padding=,name=None)

* 输入上执行最大池数
* value:4-D Tensor形状[batch, height, width, channels]
* ksize:池化窗口大小，[1, ksize, ksize, 1]
* strides:步长大小，[1,strides,strides,1]
* padding:“SAME”, “VALID”，使用的填充算法的类型

## Mnist数据集卷积网络实现[#](https://www.cnblogs.com/zhangfengxian/p/10674216.html#4277374224)

Copy

import tensorflow as tf

from tensorflow.examples.tutorials.mnist import input\_data

def weight\_variables(shape):

"""权重初始化函数"""

w = tf.Variable(tf.random\_normal(shape=shape, mean=0.0, stddev=1.0))

return w

def bias\_variables(shape):

"""偏置初始化函数"""

b = tf.Variable(tf.constant(0.0, shape=shape))

return b

def model():

"""自定义的卷积模型"""

# 建立数据的占位符

with tf.variable\_scope("data"):

x = tf.placeholder(tf.float32, [None, 28 \* 28])

y\_true = tf.placeholder(tf.float32, [None, 10])

# 第一层卷积 5\*5\*1，32个 strides=1

with tf.variable\_scope("conv1"):

w\_conv1 = weight\_variables([5, 5, 1, 32])

b\_conv1 = bias\_variables([32])

x\_reshape = tf.reshape(x, [-1, 28, 28, 1])

x\_relu1 = tf.nn.relu(tf.nn.conv2d(x\_reshape, w\_conv1, strides=[1, 1, 1, 1], padding="SAME") + b\_conv1)

# 池化

x\_pool1 = tf.nn.max\_pool(x\_relu1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding="SAME")

# 第二层卷积层，5\*5\*32，64个filter，strides=1

with tf.variable\_scope("conv2"):

w\_conv2 = weight\_variables([5, 5, 32, 64])

b\_conv2 = bias\_variables([64])

x\_relu2 = tf.nn.relu(tf.nn.conv2d(x\_pool1, w\_conv2, strides=[1, 1, 1, 1], padding="SAME") + b\_conv2)

# 池化

x\_pool2 = tf.nn.max\_pool(x\_relu2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding="SAME")

# 全连接

with tf.variable\_scope("fc"):

w\_fc = weight\_variables([7 \* 7 \* 64, 10])

b\_fc = weight\_variables([10])

x\_fc\_reshape = tf.reshape(x\_pool2, [-1, 7 \* 7 \* 64])

y\_predict = tf.matmul(x\_fc\_reshape, w\_fc) + b\_fc

return x, y\_true, y\_predict

def conv\_fc():

# 准备数据

mnist = input\_data.read\_data\_sets("./data/mnist/", one\_hot=True)

x, y\_true, y\_predict = model()

# 所有样本损失值的平均值

with tf.variable\_scope("soft\_loss"):

loss = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y\_true, logits=y\_predict))

# 梯度下降

with tf.variable\_scope("optimizer"):

train\_op = tf.train.GradientDescentOptimizer(0.0001).minimize(loss)

# 计算准确率

with tf.variable\_scope("acc"):

equal\_list = tf.equal(tf.argmax(y\_true, 1), tf.argmax(y\_predict, 1))

accuracy = tf.reduce\_mean(tf.cast(equal\_list, tf.float32))

init\_op = tf.global\_variables\_initializer()

with tf.Session() as sess:

sess.run(init\_op)

for i in range(4000):

mnist\_x, mnist\_y = mnist.train.next\_batch(50)

sess.run(train\_op, feed\_dict={x: mnist\_x, y\_true: mnist\_y})

print("训练第%d步, 准确率为%f" % (i, sess.run(accuracy, feed\_dict={x: mnist\_x, y\_true: mnist\_y})))

if \_\_name\_\_ == '\_\_main\_\_':

conv\_fc()