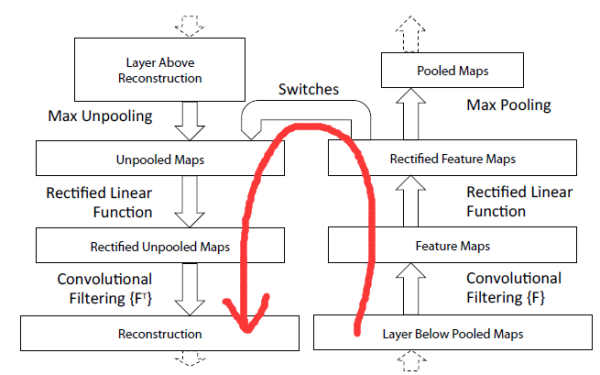
反卷积的名字的确容易误导，所以后来出现的一些名字例如转置卷积，可以理解成正常前向传播的倒向传递过程(但是不是更新梯度的反向传播)。在某一层的倒向传递的示意图如下



其中switch是记录在正常传递过程中maxpooling的最大值的坐标。

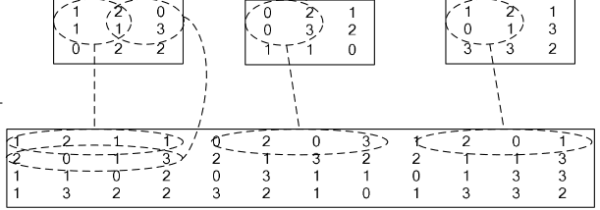
在[1]这篇文章中，引入了**反卷积层，**但是反卷积层和反卷积网络[2]不是一回事。至于反卷积网络，

[@谭旭](https://www.zhihu.com/people/4a4562549ff1c5af97ece0cd3be1747d)

已经说的很清楚了。但是最近很多文章中出现的deconv(反卷积，转置卷积，微步卷积)大多起源于[2]这篇文章。

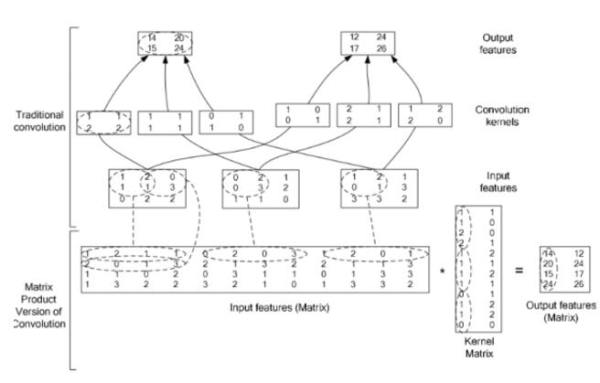
首先要想了解反卷积的原理要补充一下卷积的相关操作：

caffe里有im2col层，如果对matlab比较熟悉的话，就应该知道im2col是什么意思。它先将一个大矩阵，重叠地划分为多个子矩阵，对每个子矩阵序列化成向量，最后得到另外一个矩阵。看一看图就知道了：



在caffe中，卷积运算就是先对数据进行im2col操作，再进行内积运算（inner product)。这样做，比原始的卷积操作速度更快。

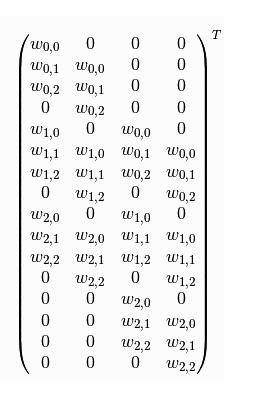
看看两种卷积操作的异同：



我们通过实验来进行解释：

**【1. 无 padding】**那么如果我们想从输入一个4\*4，channel为1的的在i层的feature map，用一个3\*3的kernel进行卷积，**padding=0,stride=1**，我们理应得到一个2\*2，在i+1层的feature map。

在实现过程，根据博客[3],会首先把3\*3的kernel展成**C=**

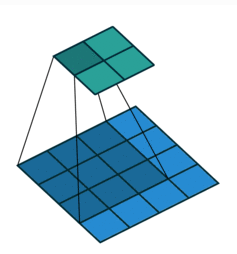


并把输入x展成16\*1的向量，这样进行**矩阵的相乘(不是卷积也不是点乘)**，根据矩阵的运算法则(4\*16)\*(16\*1)=(4,1)，把(4,1)的结果再展成（2,2），这也是我们根据卷积运算法则得到的大小:2\*2.

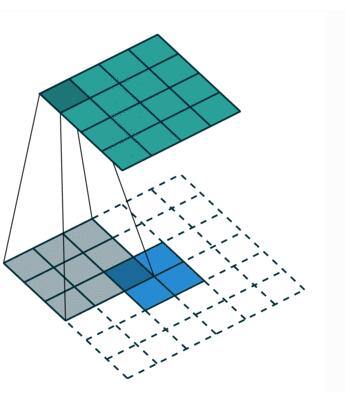
现在对输出的结果，进行转置卷积，也就是对得到的2\*2的feature map进行操作得到4\*4大小的原先的输入，我们用上一步得到的(4,1)的结果，乘上kernel的转置C',整个运算过程如果看矩阵的维度的话是(4,1)\*(1\*16)=(4\*16),这个维度和我们输入的x是相同的。

如果从直观上理解上述过程：

正向卷积过程(蓝色为输入x，阴影为卷积核C，绿色为输出)



对于No zero padding, unit strides, 的卷积来说，转置卷积：



其对应关系可以理解为正向卷积为0 padding，那么转置卷积就是full padding。

公式关系为正向卷积s=1,p=0那么转置卷积的k'=k,s'=s,p'=k-1,转置卷积的输出尺寸

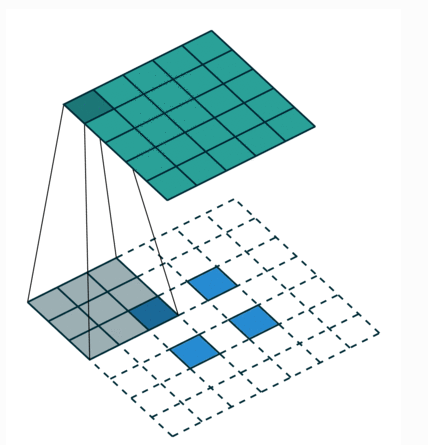
o'=i'+2(k-1)-(k-1)

可以从直观上理解当padding=0,stride=1转置卷积的运算过程。

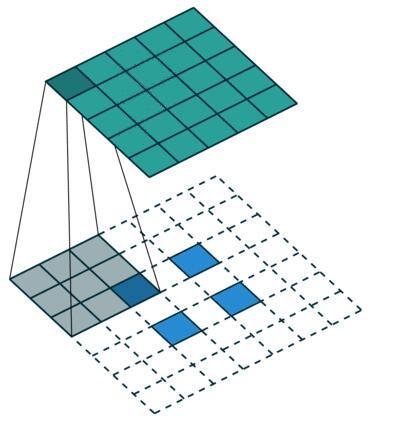
了解零填充逻辑的一种方法是考虑转置卷积的连接模式，并使用它来指导等效卷积的设计。 例如，直接卷积的输入的左上像素仅对输出的左上像素有贡献，右上像素仅连接到右上输出像素，等等。为了在等效卷积中保持相同的连接模式，有必要以使得核的第一（左上）乘法 仅接触左上像素的方式对输入进行零填充，即，填充必须相等 到内核的大小减一::Padding =ks-1 也是full卷积的模式

这也是为什么转置卷积乘上C\_T

**【2. padding】**那么当stride不为1的时候，转置卷积的卷积核就变成了一个带'洞'的卷积，也就是fractional stride convolution（微步卷积）



带洞是为了使转置卷积的步长变为正向卷积的1/i倍，核将以一个更小的步伐移动，使得转置卷积的输出比输入要大一些。 例如正向卷积输入x，i=5,p=0,k=3,stride=2,那么得到的正向卷积结果大小是2\*2的，若要恢复其5\*5的大小，转置卷积的参数应该为i'=i\_transformed(即在正向卷积的输出之间填补0), k'=k.s'=1,padding=k-1



得到的输出的大小为o'=s(i'-1)+k,这里的s指的是正向卷积的步长。

**【3】**以上是对转置卷积的直观理解，针对转置卷积为何乘上C\_T,可以参照

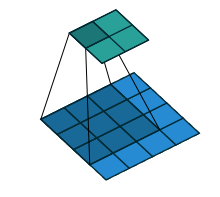
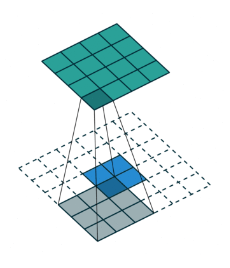
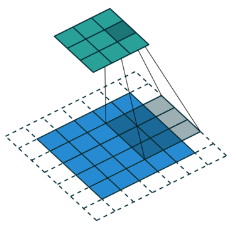
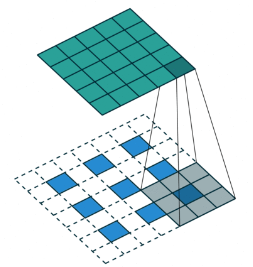
[@张萌](https://www.zhihu.com/people/a133f5be4780a91f797a750afd3e47fc)

的答案

[1] Zeiler M D, Taylor G W, Fergus R, et al. Adaptive deconvolutional networks for mid and high level feature learning[C]. International Conference on Computer Vision, 2011.

[2] Zeiler M D, Fergus R. Visualizing and Understanding Convolutional Networks[C]. European Conference on Computer Vision, 2013.

[3][Convolution arithmetic tutorial](https://link.zhihu.com/?target=http%3A//deeplearning.net/software/theano_versions/dev/tutorial/conv_arithmetic.html%23transposed-convolution-arithmetic)

卷积(convolution):  
卷积核为 3x3；no padding , strides=1  
"反卷积"(the transpose of conv) 可以理解为upsample conv.  
卷积核为:3x3; no padding , strides=1  
  
那看下strides=2的时候。  
卷积:  
  
  
反卷积:  
  
在实际计算过程中，我们要转化为矩阵的乘积的形式，一个转化为  
Toeplitz matrix一个reshape为列矩阵。  
  
举个简单的例子  
  
比如 input= [3,3],Reshape之后，为A=[1,9]  
B(可以理解为滤波器)=[9,4](Toeplitz matrix)  
那么  
A\*B=C=[1,4]。Reshape C=[2，2]  
  
所以，通过B **卷积**，我们从shape=[3,3]变成了shape=[2,2]  
  
反过来。  
输入A=[2,2],reshape之后为[1,4]   
B的转置为，[4,9]  
那么  
A\*B=C=[1,9],reshape为[3,3]  
  
所以，通过B的转置 - "反卷积"，我们从shape=[2,2]得到了shape=[3,3]  
  
也就是  
输入feature map A=[3,3]经过了卷积滤波B=[2,2] 输出为 [2,2] ,所以padding=0,stride=1  
  
反卷积则是  
输入feature map A=[2,2],经过了反卷积滤波B=[2,2].输出为[3,3].padding=0,stride=1  
  
那么[2,2]的卷积核(滤波器)是怎么转化为[4,9]或者[9,4]的呢  
  
通过  
Toeplitz matrix

不懂的，自己查吧。

**所以所谓的卷积其实是转置卷积。**

**那为什么不能叫反卷积？**

反卷积的数学含义，通过反卷积可以将通过卷积的输出信号，完全还原输入信号。

而事实是，转置卷积只能还原shape大小，不能还原value.

你可以自己通过代码验证下。