乍一看到某个问题，你会觉得很简单，其实你并没有理解其复杂性。当你把问题搞清楚之后，又会发现真的很复杂，于是你就拿出一套复杂的方案来。实际上，你的工作只做了一半，大多数人也都会到此为止……。但是，真正伟大的人还会继续向前，直至找到问题的关键和深层次原因，然后再拿出一个优雅的、堪称完美的有效方案。

—— from 乔布斯

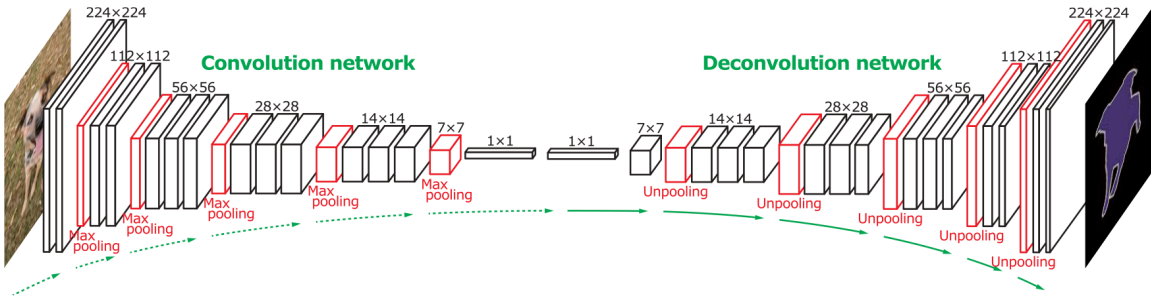
一文搞懂 deconvolution、transposed convolution、sub-pixel or fractional convolution

2019-09-20 20:55 shine-lee 29247 0 编辑 收藏 举报

目录

博客：blog.shinelee.me | 博客园 | CSDN

写在前面



开篇先上图，图为 **deconvolution** 在像素级语义分割中的一种应用，直观感觉deconvolution是一个upsampling的过程，像是convolution的对称过程。

本文将深入deconvolution的细节，并通过如下方式展开：

- 先回答 什么是 **deconvolution**？为什么会有 **transposed convolutionon** 、 **subpixel or fractional convolution** 这样的名字?
- 再介绍 各种情形下 **transposed convolution** 是如何进行的，并提供 **一种统一的计算方法** 。

导航目录

什么是deconvolution

首先要明确的是，**deconvolution并不是个好名字**，因为它存在歧义：

1. **deconvolution**最初被定义为“inverse of convolution”或者“inverse filter”或者“解卷积”，是指**消除先前滤波作用的方法**。比如，我们认为原始图像是清晰的，但是通过透镜观测到的图像却变得模糊，如果假设透镜的作用相当于以某个kernel作用在原始图像上，由此导致图像变得模糊，那么根据模糊的图像估计这个kernel或者根据模糊图像恢复原始清晰图像的过程就叫**deconvolution**。
2. 后来论文Adaptive Deconvolutional Networks for Mid and High Level Feature Learning和Visualizing and Understanding Convolutional Networks又重新定义了**deconvolution**，实际上与transposed convolution、sub-pixel or fractional convolution指代相同。**transposed convolution**是一个更好的名字，sub-pixel or fractional convolution可以看成是transposed convolution的一个特例。对一个常规的卷积层而言，前向传播时是convolution，将input feature map映射为output feature map，反向传播时则是transposed convolution，根据output feature map的梯度计算出input feature map的梯度，梯度图的尺寸与feature map的尺寸相同。

本文谈论的是deconvolution的第2个含义，后面统一使用**transposed convolution**这个名字。

什么是transposed convolution？A guide to convolution arithmetic for deep learning中有这样一段话：

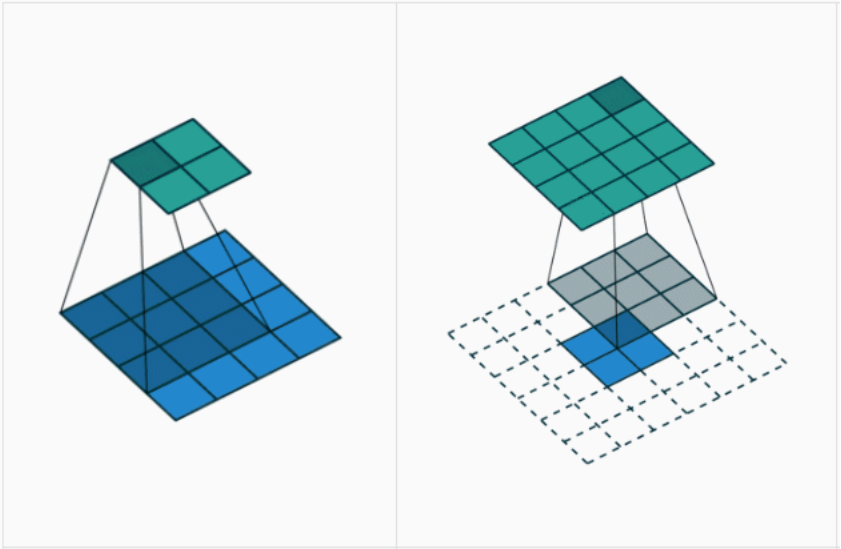
> Let's now consider what would be required to go the other way around, i.e., map from a 4-dimensional space to a 16-dimensional space, while keeping the connectivity pattern of the convolution depicted in Figure 2.1. This operation is known as a *transposed convolution*.
>
> Transposed convolutions – also called *fractionally strided convolutions* or *deconvolutions*[1] – work by swapping the forward and backward passes of a convolution. One way to put it is to note that the kernel defines a convolution, but whether it's a direct convolution or a transposed convolution is determined by how the forward and backward passes are computed.

看完好像仍不是很直观，transposed convolution到底对应的是什么操作？等到文章的后面，这个问题的答案会逐渐清晰起来。

下面先以1个例子来对比convolution过程和transposed convolution过程，采用与A guide to convolution arithmetic for deep learning相同的设置：

- 2-D transposed convolutions ($N = 2$)
- square inputs ($i_1 = i_2 = i$)
- square kernel size ($k_1 = k_2 = k$)
- same strides along both axes ($s_1 = s_2 = s$)
- same zero padding along both axes ($p_1 = p_2 = p$)
- square outputs ($o_1 = o_2 = o$)

若令$i = 4$、$s = 1$、$p = 0$、$k = 3$，输出尺寸$o = 2$，则convolution过程是将4×4的map映射为2×2的map，而transposed convolution过程则是将2×2的map映射为4×4的map，两者的kernel size均为3，如下图所示：



No padding, no strides	No padding, no strides, transposed

可以看到，convolution过程zero padding的数量与超参数p一致，但是transposed convolution实际的zero padding的数量为2，为什么会这样？是 **为了保持连接方式相同** ，下面具体看一下。

convolution过程

先看convolution过程，连接方式 如下图所示，绿色表示输出，蓝色表示输入，每个绿色块具与9个蓝色块连接。

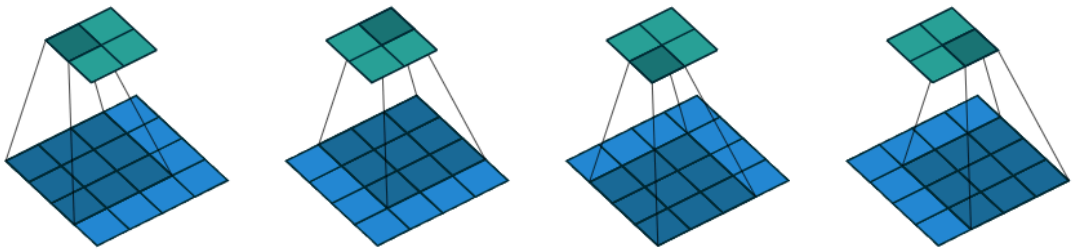


Figure 2.1: (No padding, unit strides) Convolving a 3×3 kernel over a 4×4 input using unit strides (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$).

令卷积核$\mathbf{w} = \begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,0} & w_{1,2} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix}$，为了便于理解， **将卷积写成矩阵乘法形式** ，令$\mathbf{x}$为4×4输入矩阵以行优先方式拉成的长度为16的向量，$\mathbf{y}$为2×2输出矩阵以同样方式拉成的长度为4的向量，同时将$\mathbf{w}$表示成4×16的稀疏矩阵$\mathbf{C}$，

$$\begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix}$$

则convolution过程可以描述为$\mathbf{Cx} = \mathbf{y}$，若 $\mathbf{C}_{i,j} = 0$**表示**$\mathbf{x}_j$**和**$\mathbf{y}_i$**间没有连接** 。

transposed convolution过程

再看transposed convolution过程，如何将长度为4的向量$\mathbf{y}$映射为长度为16的向量且 **保持连接方式相同** ？只需将$\mathbf{C}$转置，令$\mathbf{C}^T\mathbf{y} = \mathbf{x}'$，同样地， $\mathbf{C}^T_{j,i} = 0$**表示**$\mathbf{x}'_j$**和**$\mathbf{y}_i$**间没有连接** 。

此时，$\mathbf{C}^T$对应的卷积操作恰好相当于**将kernel中心对称**，FULL zero padding，然后卷积，此时，1个蓝色块与9个绿色块连接，且权重与Convolution过程相同

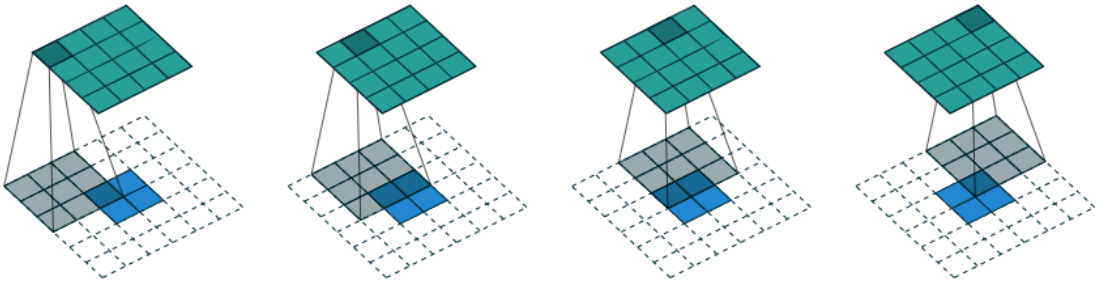


Figure 4.1: The transpose of convolving a 3×3 kernel over a 4×4 input using unit strides (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$). It is equivalent to convolving a 3×3 kernel over a 2×2 input padded with a 2×2 border of zeros using unit strides (i.e., $i' = 2$, $k' = k$, $s' = 1$ and $p' = 2$).

需要注意的是，transposed convolution的kernel与convolution的kernel可以有关，也可以无关，需要看应用在什么场景，

- 在**特征可视化**、**训练阶段的反向传播**中应用的transposed convolution，**并不是作为一个真正的layer存在于网络中**，其kernel与convolution共享（但要经过中心对称后再卷积，相当于上面的 $\mathbf{C}^T$）。
- 在**图像分割**、**生成模型**、**decoder**中使用的transposed convolution，**是网络中真实的layer**，其kernel经初始化后需要通过学习获得（所以卷积核也就无所谓中心对称不对称了）。
- 前向传播为convolution/transposed convolution，则反向传播为transposed convolution/convolution。

在上面举的简化的例子中，我们可以通过分析得知transposed convolution该如何进行，但是，对于更一般情况应该怎么做？

transposed convolution的计算

对于一般情况，只需把握一个**宗旨：transposed convolution将output size恢复为input size且保持连接方式相同**。

对于convolution过程，我们知道其output map与input map的尺寸关系如下：

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

若要将o恢复为i，需考虑2种情况，$\frac{i+2p-k}{s}$整除以及不整除，先看整除的情况。

整除的情况

如果$\frac{i+2p-k}{s}$可以整除，则由上式可得

$$i = so - s + k - 2p = [o + (s-1)(o-1)] + (k - 2p - 1)$$

因为transposed convolution也是卷积，**为了符合上面卷积操作尺寸关系的数学形式**，可进一步整理成

$$i = \frac{[o + (s-1)(o-1)] + [(k-1) + (k-2p-1)] - k}{1} + 1$$

令$i' = o + (s-1)(o-1)$、$p' = \frac{(k-1)+(k-2p-1)}{2} = k - p - 1$、$s' = 1$、$k' = k$，即transposed convolution实际卷积时使用的超参数，可以这样理解：

- $i' = o + (s-1)(o-1)$：convolution的输出为$o \times o$，每行每列都是o个元素，有$o-1$个间隔，transposed convolution时在每个间隔处插入$s-1$个0，整体构成transposed convolution的input map；
- $p' = \frac{(k-1)+(k-2p-1)}{2} = k - p - 1$：在上一步input map的基础上再进行padding，考虑convolution常用的几种padding情况：

- **VALID**：$p=0$，transposed convolution则需padding $p'=k-1$，即 **FULL** padding
- **SAME**：$p=\frac{k-1}{2}=r$，这里考虑$k=2r+1$为奇数的一般情况，此时$p'=r$，即 **SAME** padding
- **FULL**：$p=k-1$，则$p'=0$，即 **VALID** padding

可见，convolution和transposed convolution的padding也具有某种对称性$p'+p=k-1$；

- $k'=k$：transposed convolution的kernel size与convolution相同；
- $s'=1$：**transposed convolution的stride均为1**，但也可以换个角度理解，如果认为$o\times o$相邻元素间的距离为1个像素，那么在间隔处插入$s-1$个0后（$s>1$），得到的input map相邻元素间的距离就是亚像素的（sub-pixel），所以此时也可以称之为 **sub-pixel or fractional convolution**；
- $o'=i=\frac{i'+2p'-k'}{s'}+1$：transposed convolution的输出与convolution的输入具有相同尺寸。

不整除的情况

接下来再看$\frac{i+2p-k}{s}$不整除的情况，此时再按上面的方式计算得到的$o'=\frac{i'+2p'-k'}{s'}+1$将小于$i$，小多少呢？不难得出少$a=[(i+2p-k)\mod s]$，即

$$o'=\frac{i'+2p'-k'}{s'}+1=i-a$$

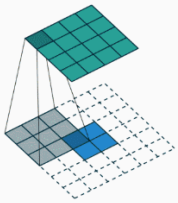
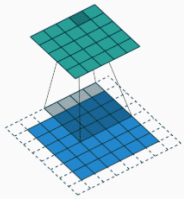
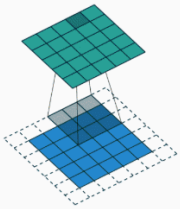
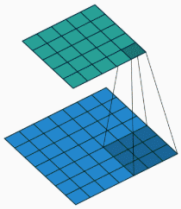
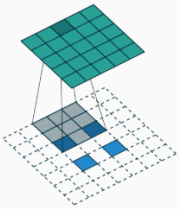
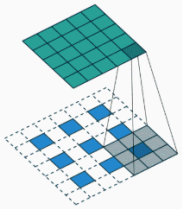
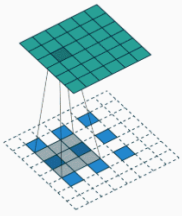
为了让$o'=i$，可写成

$$o'=\frac{i'+2p'+a-k'}{s'}+1$$

只需在padding后，在下边和右边再扩展a行和列0，然后进行卷积即可。注意，因为$s'=1$，我们可以将a放在分母也可以放在外面，之所以放在分母，是因为convolution过程中input map下边和右边的a行或列中的元素可能参与了运算，即与output map间存在连接，所以在transposed convolution时，为了保持同样的连接，最后扩展的a行和列也要参与卷积，所以放在分母。

至此，再看transposed convolution的各种情况，就很容易推算了，更多例子可参见A guide to convolution arithmetic for deep learning。

N.B.: Blue maps are inputs, and cyan maps are outputs.

			
No padding, no strides, transposed	Arbitrary padding, no strides, transposed	Half padding, no strides, transposed	Full padding, no strides, transposed
			
No padding, strides, transposed	Padding, strides, transposed	Padding, strides, transposed (odd)	

总结

最后，总结一下，

- **convolution和transposed convolution互为对称过程**，存在一个convolution，就存在一个与之对应的transposed convolution，反之亦然；

- convolution是将input size的map映射为output size的map，transposed convolution是将output size的map映射为input size的map——**旨在将尺寸恢复**；
- **两者均使用卷积操作**，为了方便，**两者使用同样的stride、padding、kernel size超参数，但实际执行时的操作不同**，一般情况下，transposed convolution与convolution实际超参数关系为：$i' = o + (s-1)(o-1)$、$p' = \frac{(k-1)+(k-2p-1)}{2} = k-p-1$、$s' = 1$、$k' = k$。
- 之所以做这样的操作，是为了**保证map间的连接方式相同（权重不一定相同）**，权重的设置需根据应用的场景，可能通过学习得到，也可能与convolution共享（但需要中心对称后再使用）。

参考

- vdumoulin/conv_arithmetic
- A guide to convolution arithmetic for deep learning
- winter1516_lecture13.pdf
- Is the deconvolution layer the same as a convolutional layer?
- What are deconvolutional layers?

分类: 深度学习基础
标签: CNN

« 上一篇： 从AlexNet(2012)开始
» 下一篇： ZFNet(2013)及可视化的开端