A guide to convolution arithmetic for deep learning

Vincent Dumoulin[1★] and Francesco Visin[2★†]

[★]MILA, Université de Montréal
[†]AIRLab, Politecnico di Milano

2018.1.12

[1]dumouliv@iro.umontreal.ca
[2]francesco.visin@polimi.it

All models are wrong, but some are useful.

George E. P. Box

Acknowledgements

The authors of this guide would like to thank David Warde-Farley, Guillaume Alain and Caglar Gulcehre for their valuable feedback. We are likewise grateful to all those who helped improve this tutorial with helpful comments, constructive criticisms and code contributions. Keep them coming!

Special thanks to Ethan Schoonover, creator of the Solarized color scheme,[1] whose colors were used for the figures.

Feedback

Your feedback is welcomed! We did our best to be as precise, informative and up to the point as possible, but should there be anything you feel might be an error or could be rephrased to be more precise or comprehensible, please don't refrain from contacting us. Likewise, drop us a line if you think there is something that might fit this technical report and you would like us to discuss – we will make our best effort to update this document.

Source code and animations

The code used to generate this guide along with its figures is available on GitHub.[2] There the reader can also find an animated version of the figures.

[1] http://ethanschoonover.com/solarized

[2] https://github.com/vdumoulin/conv_arithmetic

目录

第一章　引言

深度卷积神经网络 (CNNs) 一直是深度学习取得巨大进步的核心. 尽管卷积神经网络在九十年代就被用来解决字符识别任务 (Le Cun *et al.*, 1997)，它们当前的广泛应用是由于最近的工作，当时一个深层 CNN 被用来打败最先进的 ImageNet 图像分类挑战 (Krizhevsky *et al.*, 2012).

因此卷积神经网络为机器学习的专业人士构建了一个非常有用的工具. 然而，第一次学习使用卷积神经网络通常是一种 "可怕" 的经历. 一个卷积层的输出形状受它的输入形状及卷积核的形状、零填充和步长的选择所影响，并且这些特性之间的关系不可忽视. 这不同于输出尺寸依赖于输入尺寸的全连接层. 另外，卷积神经网络通常还具有一个池化 (pooling) 阶段，对全连接网络来说增加了另一层的复杂性. 最后，所谓的转置卷积层（也被称为分步卷积层）近期已经在越来越多的工作 (Zeiler *et al.*, 2011; Zeiler and Fergus, 2014; Long *et al.*, 2015; Radford *et al.*, 2015; Visin *et al.*, 2015; Im *et al.*, 2016), 中被采用，并且它们与卷积层的关系已经从多个角度解释清楚了.

本指南的目标有两个:

1. 解释卷积层和转置卷积层.
2. 直观地理解卷积、池化和转置卷积层中输入形状、卷积核形状、零填充、步长和输出形状之间的关系.

为了保持普适性，本指南中所示结果不依赖实现细节，并适用于所有常用的机器学习框架，例如 Theano (Bergstra *et al.*, 2010; Bastien *et al.*, 2012), Torch (Collobert *et al.*, 2011), Tensorflow (Abadi *et al.*, 2015) 和 Caffe (Jia *et al.*, 2014).

本章简要回顾卷积神经网络的主要组成部分，即离散卷积和池化. 为了深入研究这一问题，请看深度学习教材的第九章 (Goodfellow *et al.*, 2016).

1.1　离散卷积

神经网络的中心思想是 *仿射变换* (affine transformations): 一个向量被接收为输入（input），并与矩阵相乘以产生输出（在进行非线性传递之前，通常会添加一个偏置向量）. 这适用于任何类型的输入，无论是图像、音频片段还是无序的特征集合：无论它们的维数如何，在变换之前，它们的表示都可以被展成向量.

图像、音频片段和许多其他相似类型的数据都有一个本质的结构. 从形式上讲，它们共享这些重要的性质：

- 它们被存储为多维数组.
- 它们以一个或多个轴为特征，它们之间的顺序很重要（例如，图像的宽度和高度轴，音频片段的时间轴）。
- 一个轴称为通道轴 (channel axis)，用于访问数据的不同视图（例如，彩色图像的红色，绿色和蓝色通道，或立体声音频轨道的左右通道）。

在应用仿射变换之前，这些性质还未被利用；事实上，所有的轴都以相同的方式处理，而且没有考虑拓扑的信息. 尽管如此，利用数据的隐式结构可能在完成某些任务，如计算机视觉和语义识别时非常方便，在这些情况下最好保存这些数据. 这就是离散卷积发挥作用的地方.

离散卷积是一种线性变换，它保留了顺序的概念. 离散卷积是稀疏的（只有几个输入单元对给定的输出单元有贡献），并且重用参数（相同的权重应用于输入的多个位置）.

Figure 1.1 给出了一个离散卷积的例子. 浅蓝色的网格称为*输入特征映射* (input feature map). 为了保持绘图的简洁，该图代表了一个输入特征映射，但多个特征映射一个置于另一个上也很常见. [1] *卷积核* （阴影区域）在输入特征映射中滑动. 在每个位置，计算卷积核的每个元素与它重叠的输入元素之间的乘积，

0	1	2
2	2	0
0	1	2

[1] 在这方面的一个例子就是之前提到的图像和音频片段*通道* (channels) .

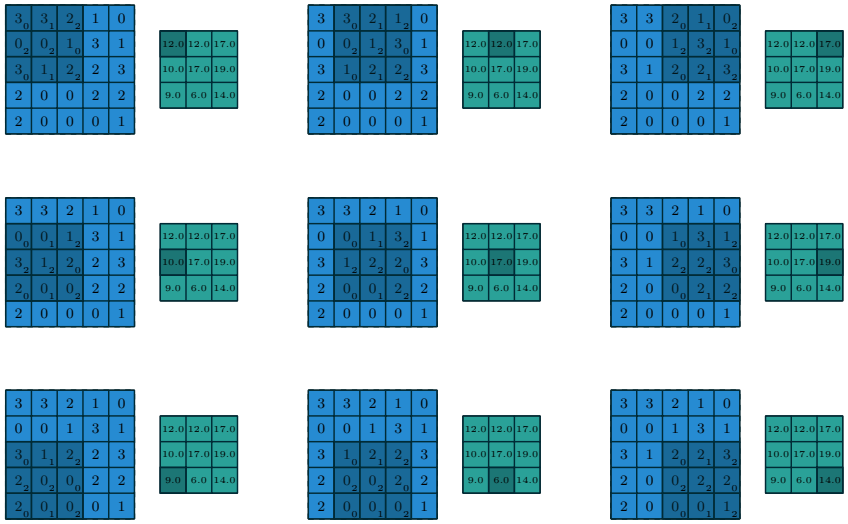

图 1.1: 计算离散卷积的输出值.

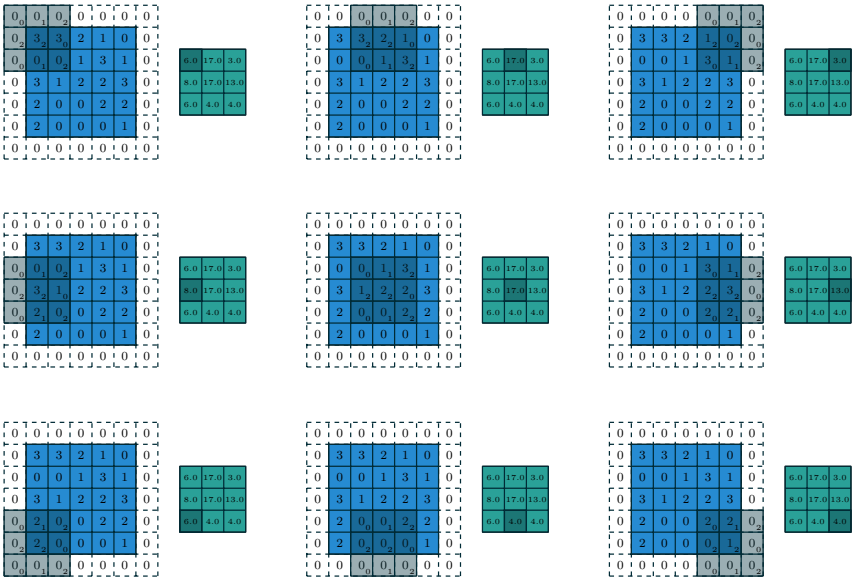

图 1.2: 对 $N = 2$, $i_1 = i_2 = 5$, $k_1 = k_2 = 3$, $s_1 = s_2 = 2$, 和 $p_1 = p_2 = 1$ 计算离散卷积的输出值.

并对结果进行求和. 可以用不同的卷积核重复该过程，以生成所需的输出特征映射 (Figure 1.3). 这个过程的最终输出被称为*输出特征映射* (output feature maps). [2]

如果有多个输入特征映射，则核必须是三维的——或者等价于每个特征映射都用一个不同的核进行卷积，并且生成的特征映射主元素相加，以生成输出特征映射。

卷积（见图Figure 1.1）是二维卷积的一种形式，但是它可以生成 N 维卷积. 例如，在三维卷积中，卷积核可以是一个*立方体* (cuboid)，而且会在输入特征映射的的高度、宽度和深度上滑动.

定义了一批离散卷积的核有一个形状对应 $(n, m, k_1, \ldots, k_N)$ 的某个排列，其中

$$
\begin{aligned}
n &\equiv \text{输出特征映射的数量}, \\
m &\equiv \text{输入特征映射的数量}, \\
k_j &\equiv \text{沿 } j \text{ 轴核的尺寸}.
\end{aligned}
$$

以下属性会影响沿 j 轴方向卷积层的输出尺寸 o_j :

- i_j: 沿 j 轴的输入尺寸,
- k_j: 沿 j 轴的核尺寸,
- s_j: 沿 j 轴的步长 (核的两个连续位置之间的距离),
- p_j: 沿 j 轴方向的零填充 (在轴开始和结尾处连接的 0 的数目).

例如, Figure 1.2 展示了一个应用于 5×5 输入的 3×3 卷积核，使用了 2×2 的步长填充了 1×1 的零边界.

注意，步长构成了一种 *子采样* (subsampling) 形式. 步长作为衡量核被翻译多少的一种替代方法，它也可以看作保留了多少输出. 例如，以两个跃点移动卷积核相当于以一个跃点移动卷积核，但只保留奇数个输出元素 (Figure 1.4).

[2] 虽然从信号处理的角度来看，卷积和交叉相关是有区别的，但当学习了卷积核时，两者就可以互换了. 为了简单起见，以及与大多数机器学习文献保持一致，我们将在这个指南中使用*卷积* (convolution) 这一术语.

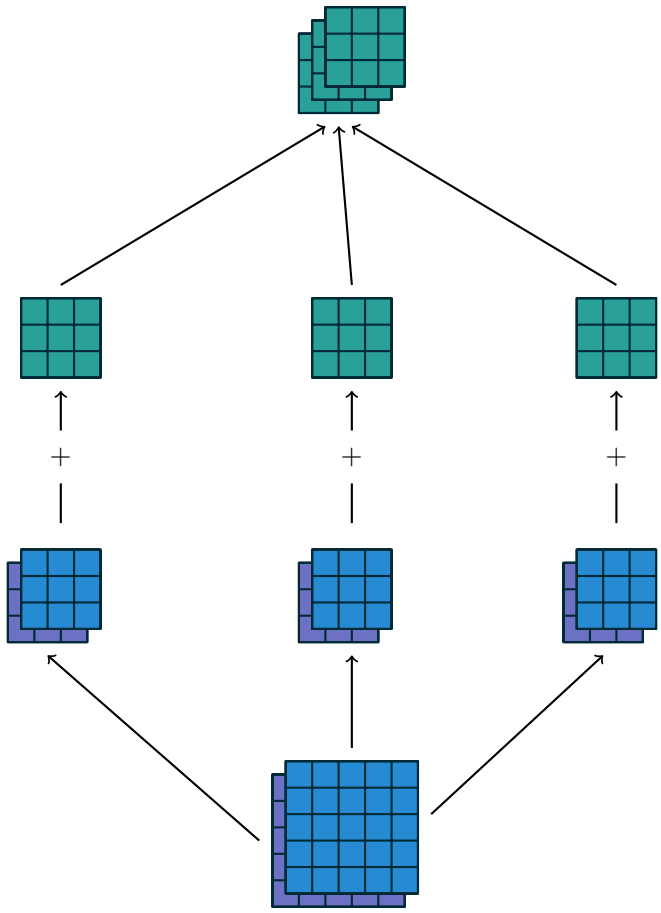

图 1.3:　一种从两个输入特征映射到三个输出特征映射的卷积映射，使用 $3 \times 2 \times 3 \times 3$ 的一批卷积核 $\mathbf{w}$. 在左边的路径中，输入特征映射 1 与核 $\mathbf{w}_{1,1}$ 做卷积且输入特征映射 2 与核 $\mathbf{w}_{1,2}$ 做卷积，并将结果逐元素相加，形成第一个输出特征映射. 对中间和右侧的路径重复相同的步骤，以形成第二和第三个特征映射，并将所有的三个输出特征映射组合在一起形成输出.

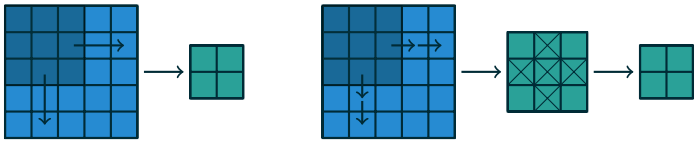

图 1.4:　另一种步长可视方法. 不是以 $s = 2$ （左边）的增量去转换 3×3 的卷积核，而是以 1 为增量进行转换，且只保留一个 $s = 2$ 的输出元素（右边）.

1.2　池化

除了离散卷积本身，*池化* (pooling) 操作是构成卷积神经网络的另一个重要部分. 池化操作通过使用某种函数来汇总子区域，例如取平均值或最大值，以此减少特征映射的大小.

池化的工作原理是在输入中滑动一个窗口，并将窗口的内容提供给*池化函数* (pooling function). 从某种意义上讲，池化的工作原理与离散卷积十分类似，但它用其他函数代替了卷积核形成的线性组合. Figure 1.5 给出了一个平均池化的例子，而 Figure 1.6 给出了最大值池化的例子.

以下属性会影响沿 j 轴池化层的输出大小 o_j:

- i_j: 沿 j 轴的输入尺寸,
- k_j: 沿 j 轴的池化窗口大小,
- s_j: 沿 j 轴的步长 (池化窗口的两连续位置之间的距离).

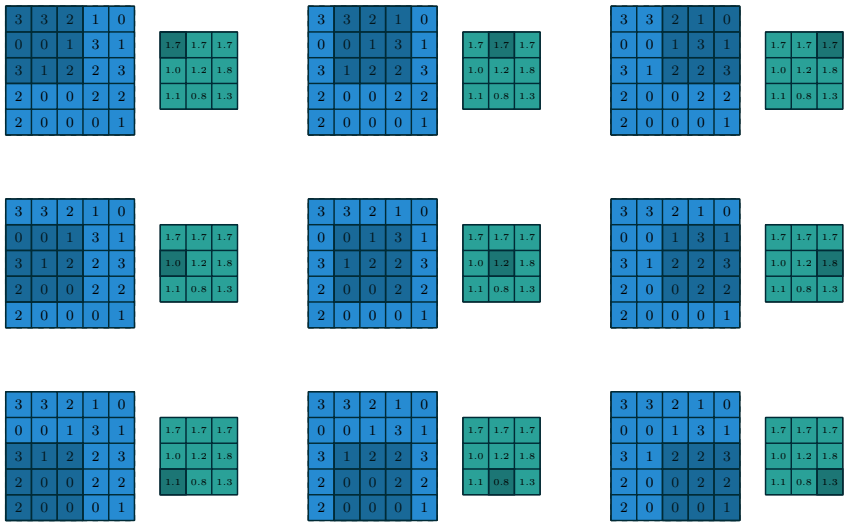

图 1.5: 使用 1×1 步长计算 5×5 输入、3×3 平均池化操作的输出值.

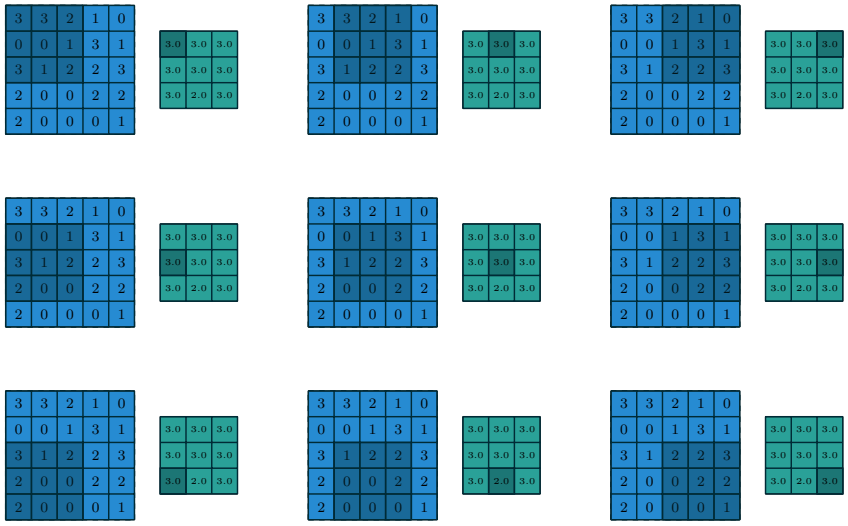

图 1.6: 使用 1×1 步长计算 5×5 输入、3×3 最大值池化操作的输出值.

第二章　卷积运算 (Convolution arithmetic)

卷积层属性之间的关系分析起来很容易，因为它们不跨轴交互，即沿 j 轴选定的卷积核大小、步长和零填充只影响 j 轴的输出大小. 因此，本章将重点介绍以下的简化设置:

- 二维离散卷积 ($N = 2$),
- 方形输入 ($i_1 = i_2 = i$),
- 方形核尺寸 ($k_1 = k_2 = k$),
- 沿着两轴的相同步长 ($s_1 = s_2 = s$),
- 沿着两轴的相同零填充 ($p_1 = p_2 = p$).

这有助于分析及可视化，但请记住，这里概述的结果也可推广到 N 维和非方形情形.

2.1　无零填充，单位步长

要分析的最简单情形是卷积核在输入 (即 $s = 1$ 且 $p = 0$ 时) 的每个位置滑动. Figure 2.1 提供了当 $i = 4$ 且 $k = 3$ 时的例子.

在这种情况下，定义输出大小的一种方法是确定在输入层上可放置卷积核的个数. 让我们考虑宽度轴：内核从输入特征映射的最左边开始，一步步滑动，直到它接触输入的右侧. 输出的大小等于所执行步数加一，即内核的初始位置 (Figure 2.8a). 相同的逻辑应用于高度轴.

更正式地说，可以推断出以下关系:

关系 1. 对任何 i 和 k, 以及对 $s = 1$ 和 $p = 0$,

$$o = (i - k) + 1.$$

2.2　有零填充，单位步长

为了考虑零填充 (即仅限于 $s = 1$), 让我们考虑其对有效输入大小的影响: 使用 p 零填充将有效输入大小从 i 改为 $i + 2p$. 在一般情况下, 关系 1 可推出以下关系:

关系 2. 对任何 i, k 和 p, 且对 $s = 1$, 有

$$o = (i - k) + 2p + 1.$$

Figure 2.2 给出了当 $i = 5$, $k = 4$ 且 $p = 2$ 时的一个例子.

实际上，零填充的两个特定实例由于各自的性质而被广泛使用. 让我们更详细地讨论一下.

2.2.1　半填充 (相似填充)

输出大小和输入大小相同 (即 $o = i$) 是一个理想的性质:

关系 3. 对任何 i 和奇数 k ($k = 2n + 1, \quad n \in \mathbb{N}$), $s = 1$, $p = \lfloor k/2 \rfloor = n$, 有

$$\begin{aligned} o &= i + 2\lfloor k/2 \rfloor - (k - 1) \\ &= i + 2n - 2n \\ &= i. \end{aligned}$$

这有时被称为半填充 (或相似填充). Figure 2.3 给出了当 $i = 5$, $k = 3$ 且 (因此) $p = 1$ 时的一个例子.

2.2.2　全填充 (Full padding)

虽然对一个核做卷积时，通常会减小 相对于输入的输出大小，但有时需要相反的结果. 这可以通过适当的零填充来实现:

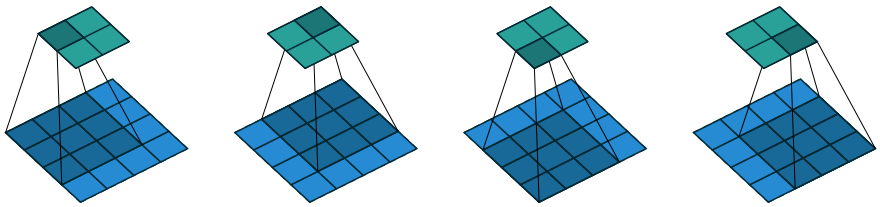

图 2.1: (无零填充，单位步长) 用单位步长 (即 $i = 4, k = 3, s = 1$ 和 $p = 0$)，将 3×3 的卷积核与 4×4 的输入做卷积.

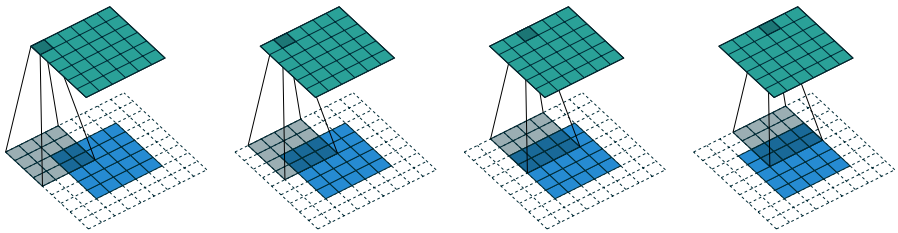

图 2.2: (任意填充, 单位步长) 用单位步长 (即 $i = 5, k = 4, s = 1$ 和 $p = 2$)，将 4×4 的卷积核与 5×5 的输入做卷积，并用 2×2 的零边界填充.

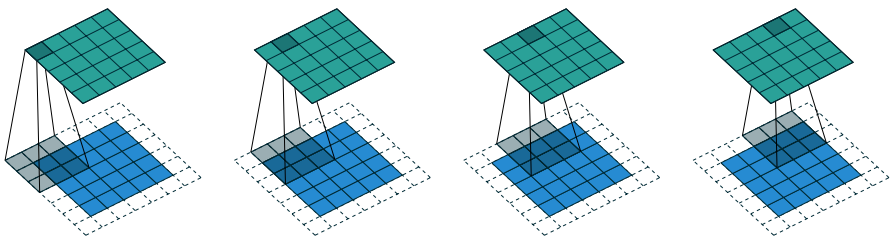

图 2.3: (半填充, 单位步长) 用半填充和单位步长 (即 $i = 5, k = 3, s = 1$ 和 $p = 2$)，将 3×3 的卷积核与 5×5 的输入做卷积.

关系 4. 对任何 i 和 k, 且对 $p = k - 1$ 和 $s = 1$, 有

$$\begin{aligned} o &= i + 2(k-1) - (k-1) \\ &= i + (k-1). \end{aligned}$$

这有时被称为全 填充, 因为在这个设定中，输入特征映射上卷积核的每一个可能的部分或完全重叠都被考虑在内. Figure 2.4 提供了当 $i = 5$, $k = 3$ 且 (因此) $p = 2$.

2.3　无零填充，非单位步长

迄今为止导出的所有关系仅适用于单位步长卷积. 合并非单位步长需要另一种进一步的推理. 为了便于分析，让我们暂时忽略零填充 (即 $s > 1$ 且 $p = 0$). Figure 2.5 给出了当 $i = 5$, $k = 3$ 且 $s = 2$ 时的一个例子.

同样，输出大小可以根据卷积核在输入上的可能位置的数量来定义. 让我们考虑宽度轴：卷积核像往常一样从输入的最左边开始，但是这次它以大小为 s 滑动，直到它接触到输入的右侧. 这个输出的大小再次等于所执行的步骤数加一，说明了卷积核的初始位置 (Figure 2.8b). 相同的逻辑应用于高度轴.

From this, the following relationship can be inferred:

关系 5. 对任何 i, k 和 s, 并对 $p = 0$, 有

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

下取整函数说明了这样一个事实：有时最后可能的步骤与到达输入层末尾的卷积核不 一致, 也就是说, 一些输入单元被忽略了 (关于这种情况，参见 Figure 2.7).

2.4　有零填充，非单位步长

The most general case (convolving over a zero padded input using non-unit strides) can be derived by applying 关系 5 on an effective input of size $i + 2p$, in analogy to what was done for 关系 2:

关系 6. *For any i, k, p and s,*

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

As before, the floor function means that in some cases a convolution will produce the same output size for multiple input sizes. More specifically, if $i+2p-k$ is a multiple of s, then any input size $j = i+a, \quad a \in \{0, \ldots, s-1\}$ will produce the same output size. Note that this ambiguity applies only for $s > 1$.

Figure 2.6 shows an example with $i = 5$, $k = 3$, $s = 2$ and $p = 1$, while Figure 2.7 provides an example for $i = 6$, $k = 3$, $s = 2$ and $p = 1$. Interestingly, despite having different input sizes these convolutions share the same output size. While this doesn't affect the analysis for *convolutions*, this will complicate the analysis in the case of *transposed convolutions*.

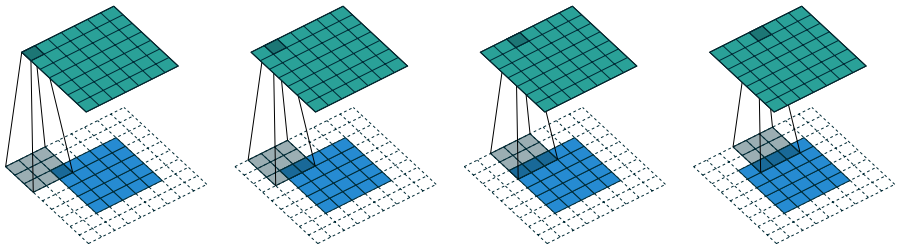

图 2.4: (全填充，单位步长) 使用全填充和单位步长 (即 $i = 5$, $k = 3$, $s = 1$ 和 $p = 2$)，将 3×3 的卷积核与 5×5 的输入做卷积.

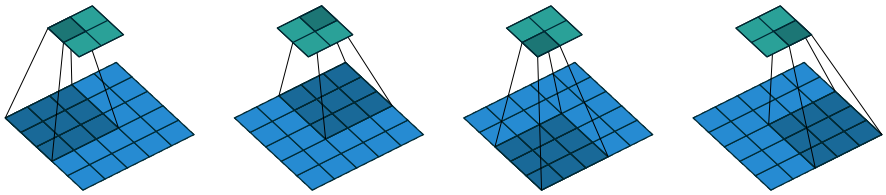

图 2.5: (No zero padding, arbitrary strides) Convolving a 3×3 kernel over a 5×5 input using 2×2 strides (i.e., $i = 5$, $k = 3$, $s = 2$ and $p = 0$).

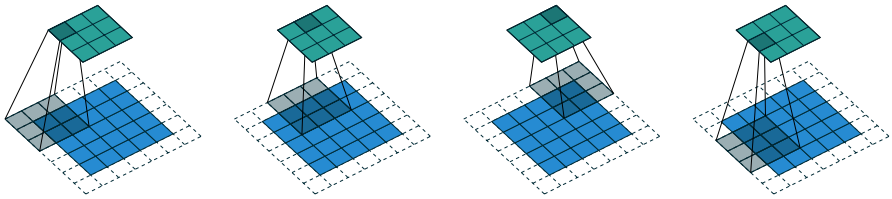

图 2.6: (Arbitrary padding and strides) Convolving a 3×3 kernel over a 5×5 input padded with a 1×1 border of zeros using 2×2 strides (i.e., $i = 5$, $k = 3$, $s = 2$ and $p = 1$).

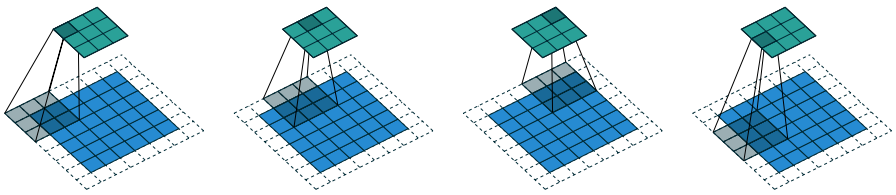

图 2.7: (Arbitrary padding and strides) Convolving a 3×3 kernel over a 6×6 input padded with a 1×1 border of zeros using 2×2 strides (i.e., $i = 6$, $k = 3$, $s = 2$ and $p = 1$). In this case, the bottom row and right column of the zero padded input are not covered by the kernel.

(a) The kernel has to slide two steps to the right to touch the right side of the input (and equivalently downwards). Adding one to account for the initial kernel position, the output size is 3×3.

(b) The kernel has to slide one step of size two to the right to touch the right side of the input (and equivalently downwards). Adding one to account for the initial kernel position, the output size is 2×2.

图 2.8: Counting kernel positions.

第三章　池化运算 (Pooling arithmetic)

In a neural network, pooling layers provide invariance to small translations of the input. The most common kind of pooling is *max pooling*, which consists in splitting the input in (usually non-overlapping) patches and outputting the maximum value of each patch. Other kinds of pooling exist, e.g., mean or average pooling, which all share the same idea of aggregating the input locally by applying a non-linearity to the content of some patches (Boureau *et al.*, 2010a,b, 2011; Saxe *et al.*, 2011).

Some readers may have noticed that the treatment of convolution arithmetic only relies on the assumption that some function is repeatedly applied onto subsets of the input. This means that the relationships derived in the previous chapter can be reused in the case of pooling arithmetic. Since pooling does not involve zero padding, the relationship describing the general case is as follows:

关系 7. *For any i, k and s,*

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

This relationship holds for any type of pooling.

第四章　转置卷积运算 (Transposed convolution arithmetic)

The need for transposed convolutions generally arises from the desire to use a transformation going in the opposite direction of a normal convolution, i.e., from something that has the shape of the output of some convolution to something that has the shape of its input while maintaining a connectivity pattern that is compatible with said convolution. For instance, one might use such a transformation as the decoding layer of a convolutional autoencoder or to project feature maps to a higher-dimensional space.

Once again, the convolutional case is considerably more complex than the fully-connected case, which only requires to use a weight matrix whose shape has been transposed. However, since every convolution boils down to an efficient implementation of a matrix operation, the insights gained from the fully-connected case are useful in solving the convolutional case.

Like for convolution arithmetic, the dissertation about transposed convolution arithmetic is simplified by the fact that transposed convolution properties don't interact across axes.

The chapter will focus on the following setting:

- 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 ($p_1 = p_2 = p$).

Once again, the results outlined generalize to the N-D and non-square cases.

4.1　作为矩阵运算的卷积

Take for example the convolution represented in Figure 2.1. If the input and output were to be unrolled into vectors from left to right, top to bottom, the convolution could be represented as a sparse matrix $\mathbf{C}$ where the non-zero elements are the elements $w_{i,j}$ of the kernel (with i and j being the row and column of the kernel respectively):

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

This linear operation takes the input matrix flattened as a 16-dimensional vector and produces a 4-dimensional vector that is later reshaped as the 2×2 output matrix.

Using this representation, the backward pass is easily obtained by transposing $\mathbf{C}$; in other words, the error is backpropagated by multiplying the loss with $\mathbf{C}^T$. This operation takes a 4-dimensional vector as input and produces a 16-dimensional vector as output, and its connectivity pattern is compatible with $\mathbf{C}$ by construction.

Notably, the kernel $\mathbf{w}$ defines both the matrices $\mathbf{C}$ and $\mathbf{C}^T$ used for the forward and backward passes.

4.2　转置卷积 (Transposed convolution)

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.

For instance, although the kernel $\mathbf{w}$ defines a convolution whose forward and backward passes are computed by multiplying with $\mathbf{C}$ and $\mathbf{C}^T$ respectively, it *also* defines a transposed convolution whose forward and backward passes are computed by multiplying with $\mathbf{C}^T$ and $(\mathbf{C}^T)^T = \mathbf{C}$ respectively.[2]

Finally note that it is always possible to emulate a transposed convolution with a direct convolution. The disadvantage is that it usually involves adding many columns and rows of zeros to the input, resulting in a much less efficient implementation.

Building on what has been introduced so far, this chapter will proceed somewhat backwards with respect to the convolution arithmetic chapter, deriving the properties of each transposed convolution by referring to the direct convolution with which it shares the kernel, and defining the equivalent direct convolution.

4.3　无零填充，单位步长，转置

The simplest way to think about a transposed convolution on a given input is to imagine such an input as being the result of a direct convolution applied on some initial feature map. The trasposed convolution can be then considered as the operation that allows to recover the *shape* [3] of this initial feature map.

Let's consider the convolution of a 3×3 kernel on a 4×4 input with

[1] The term "deconvolution" is sometimes used in the literature, but we advocate against it on the grounds that a deconvolution is mathematically defined as the inverse of a convolution, which is different from a transposed convolution.

[2] The transposed convolution operation can be thought of as the gradient of *some* convolution with respect to its input, which is usually how transposed convolutions are implemented in practice.

[3] Note that the transposed convolution does not guarantee to recover the input itself, as it is not defined as the inverse of the convolution, but rather just returns a feature map that has the same width and height.

unitary stride and no padding (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$). As depicted in Figure 2.1, this produces a 2×2 output. The transpose of this convolution will then have an output of shape 4×4 when applied on a 2×2 input.

Another way to obtain the result of a transposed convolution is to apply an equivalent – but much less efficient – direct convolution. The example described so far could be tackled by 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 $p' = 2$), as shown in Figure 4.1. Notably, the kernel's and stride's sizes remain the same, but the input of the transposed convolution is now zero padded.[4]

One way to understand the logic behind zero padding is to consider the connectivity pattern of the transposed convolution and use it to guide the design of the equivalent convolution. For example, the top left pixel of the input of the direct convolution only contribute to the top left pixel of the output, the top right pixel is only connected to the top right output pixel, and so on.

To maintain the same connectivity pattern in the equivalent convolution it is necessary to zero pad the input in such a way that the first (top-left) application of the kernel only touches the top-left pixel, i.e., the padding has to be equal to the size of the kernel minus one.

Proceeding in the same fashion it is possible to determine similar observations for the other elements of the image, giving rise to the following relationship:

关系 8. *A convolution described by $s = 1$, $p = 0$ and k has an associated transposed convolution described by $k' = k$, $s' = s$ and $p' = k - 1$ and its output size is*

$$o' = i' + (k - 1).$$

Interestingly, this corresponds to a fully padded convolution with unit

[4]Note that although equivalent to applying the transposed matrix, this visualization adds a lot of zero multiplications in the form of zero padding. This is done here for illustration purposes, but it is inefficient, and software implementations will normally not perform the useless zero multiplications.

strides.

4.4 Zero padding, unit strides, transposed

Knowing that the transpose of a non-padded convolution is equivalent to convolving a zero padded input, it would be reasonable to suppose that the transpose of a zero padded convolution is equivalent to convolving an input padded with *less* zeros.

It is indeed the case, as shown in Figure 4.2 for $i = 5$, $k = 4$ and $p = 2$.

Formally, the following relationship applies for zero padded convolutions:

> **关系 9.** *A convolution described by $s = 1$, k and p has an associated transposed convolution described by $k' = k$, $s' = s$ and $p' = k - p - 1$ and its output size is*
>
> $$o' = i' + (k - 1) - 2p.$$

4.4.1 Half (same) padding, transposed

By applying the same inductive reasoning as before, it is reasonable to expect that the equivalent convolution of the transpose of a half padded convolution is itself a half padded convolution, given that the output size of a half padded convolution is the same as its input size. Thus the following relation applies:

> **关系 10.** *A convolution described by $k = 2n + 1, \quad n \in \mathbb{N}$, $s = 1$ and $p = \lfloor k/2 \rfloor = n$ has an associated transposed convolution described by $k' = k$, $s' = s$ and $p' = p$ and its output size is*
>
> $$\begin{aligned} o' &= i' + (k - 1) - 2p \\ &= i' + 2n - 2n \\ &= i'. \end{aligned}$$

Figure 4.3 provides an example for $i = 5$, $k = 3$ and (therefore) $p = 1$.

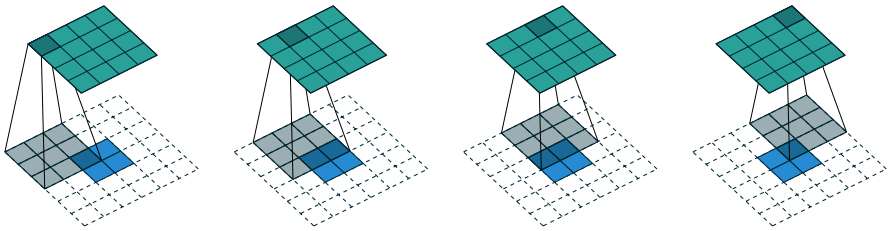

图 4.1: The transpose of convolving a 3×3 kernel over a 4×4 input using unit strides (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$). It is equivalent to convolving a 3×3 kernel over a 2×2 input padded with a 2×2 border of zeros using unit strides (i.e., $i' = 2$, $k' = k$, $s' = 1$ and $p' = 2$).

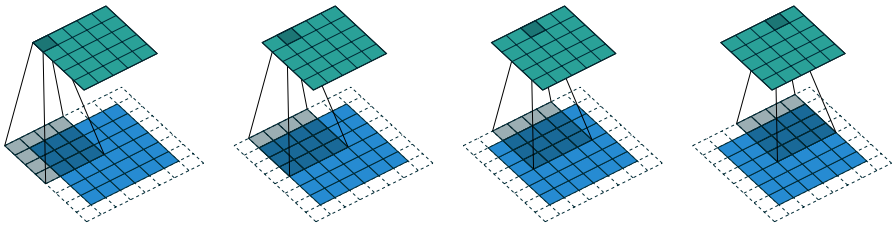

图 4.2: The transpose of convolving a 4×4 kernel over a 5×5 input padded with a 2×2 border of zeros using unit strides (i.e., $i = 5$, $k = 4$, $s = 1$ and $p = 2$). It is equivalent to convolving a 4×4 kernel over a 6×6 input padded with a 1×1 border of zeros using unit strides (i.e., $i' = 6$, $k' = k$, $s' = 1$ and $p' = 1$).

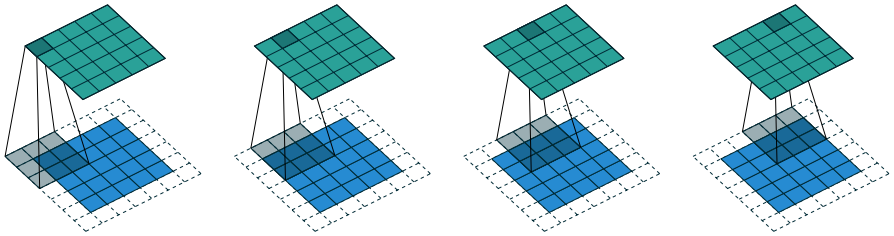

图 4.3: The transpose of convolving a 3×3 kernel over a 5×5 input using half padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 1$). It is equivalent to convolving a 3×3 kernel over a 5×5 input using half padding and unit strides (i.e., $i' = 5$, $k' = k$, $s' = 1$ and $p' = 1$).

4.4.2 Full padding, transposed

Knowing that the equivalent convolution of the transpose of a non-padded convolution involves full padding, it is unsurprising that the equivalent of the transpose of a fully padded convolution is a non-padded convolution:

关系 11. *A convolution described by* $s = 1$, k *and* $p = k - 1$ *has an associated transposed convolution described by* $k' = k$, $s' = s$ *and* $p' = 0$ *and its output size is*

$$\begin{aligned} o' &= i' + (k - 1) - 2p \\ &= i' - (k - 1) \end{aligned}$$

Figure 4.4 provides an example for $i = 5$, $k = 3$ and (therefore) $p = 2$.

4.5 No zero padding, non-unit strides, transposed

Using the same kind of inductive logic as for zero padded convolutions, one might expect that the transpose of a convolution with $s > 1$ involves an equivalent convolution with $s < 1$. As will be explained, this is a valid intuition, which is why transposed convolutions are sometimes called *fractionally strided convolutions*.

Figure 4.5 provides an example for $i = 5$, $k = 3$ and $s = 2$ which helps understand what fractional strides involve: zeros are inserted *between* input units, which makes the kernel move around at a slower pace than with unit strides.[5]

For the moment, it will be assumed that the convolution is non-padded ($p = 0$) and that its input size i is such that $i - k$ is a multiple of s. In that case, the following relationship holds:

[5] Doing so is inefficient and real-world implementations avoid useless multiplications by zero, but conceptually it is how the transpose of a strided convolution can be thought of.

关系 12. *A convolution described by $p = 0$, k and s and whose input size is such that $i - k$ is a multiple of s, has an associated transposed convolution described by $\tilde{i}'$, $k' = k$, $s' = 1$ and $p' = k-1$, where $\tilde{i}'$ is the size of the stretched input obtained by adding $s - 1$ zeros between each input unit, and its output size is*

$$o' = s(i' - 1) + k.$$

4.6 Zero padding, non-unit strides, transposed

When the convolution's input size i is such that $i + 2p - k$ is a multiple of s, the analysis can extended to the zero padded case by combining 关系 9 and 关系 12:

关系 13. *A convolution described by k, s and p and whose input size i is such that $i + 2p - k$ is a multiple of s has an associated transposed convolution described by $\tilde{i}'$, $k' = k$, $s' = 1$ and $p' = k - p - 1$, where $\tilde{i}'$ is the size of the stretched input obtained by adding $s - 1$ zeros between each input unit, and its output size is*

$$o' = s(i' - 1) + k - 2p.$$

Figure 4.6 provides an example for $i = 5$, $k = 3$, $s = 2$ and $p = 1$.

The constraint on the size of the input i can be relaxed by introducing another parameter $a \in \{0, \ldots, s - 1\}$ that allows to distinguish between the s different cases that all lead to the same i':

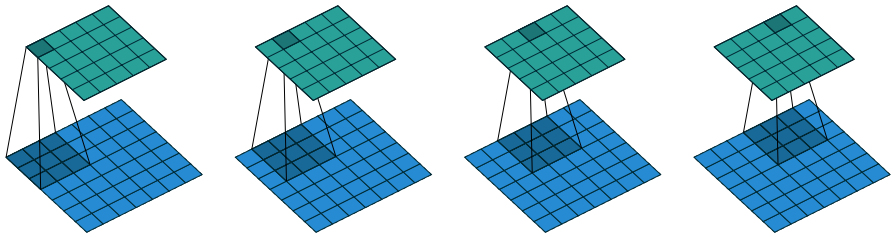

图 4.4: The transpose of convolving a 3×3 kernel over a 5×5 input using full padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 2$). It is equivalent to convolving a 3×3 kernel over a 7×7 input using unit strides (i.e., $i' = 7$, $k' = k$, $s' = 1$ and $p' = 0$).

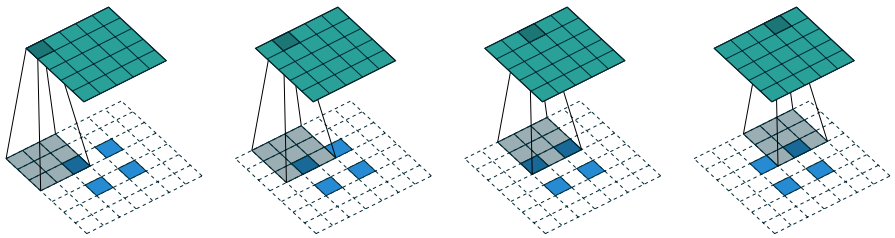

图 4.5: The transpose of convolving a 3×3 kernel over a 5×5 input using 2×2 strides (i.e., $i = 5$, $k = 3$, $s = 2$ and $p = 0$). It is equivalent to convolving a 3×3 kernel over a 2×2 input (with 1 zero inserted between inputs) padded with a 2×2 border of zeros using unit strides (i.e., $i' = 2$, $\tilde{i}' = 3$, $k' = k$, $s' = 1$ and $p' = 2$).

关系 14. *A convolution described by k, s and p has an associated transposed convolution described by a, $\tilde{i}'$, $k' = k$, $s' = 1$ and $p' = k - p - 1$, where $\tilde{i}'$ is the size of the stretched input obtained by adding $s-1$ zeros between each input unit, and $a = (i + 2p - k) \mod s$ represents the number of zeros added to the bottom and right edges of the input, and its output size is*

$$o' = s(i' - 1) + a + k - 2p.$$

Figure 4.7 provides an example for $i = 6$, $k = 3$, $s = 2$ and $p = 1$.

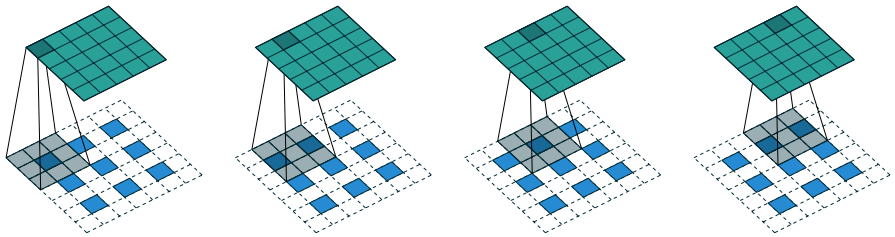

图 4.6: The transpose of convolving a 3×3 kernel over a 5×5 input padded with a 1×1 border of zeros using 2×2 strides (i.e., $i = 5$, $k = 3$, $s = 2$ and $p = 1$). It is equivalent to convolving a 3×3 kernel over a 3×3 input (with 1 zero inserted between inputs) padded with a 1×1 border of zeros using unit strides (i.e., $i' = 3$, $\tilde{i}' = 5$, $k' = k$, $s' = 1$ and $p' = 1$).

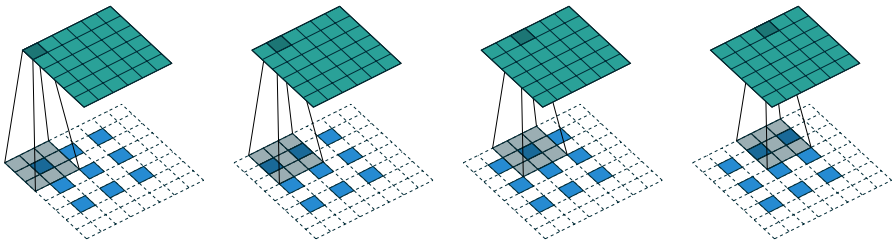

图 4.7: The transpose of convolving a 3×3 kernel over a 6×6 input padded with a 1×1 border of zeros using 2×2 strides (i.e., $i = 6$, $k = 3$, $s = 2$ and $p = 1$). It is equivalent to convolving a 3×3 kernel over a 2×2 input (with 1 zero inserted between inputs) padded with a 1×1 border of zeros (with an additional border of size 1 added to the bottom and right edges) using unit strides (i.e., $i' = 3$, $\tilde{i}' = 5$, $a = 1$, $k' = k$, $s' = 1$ and $p' = 1$).

第五章　其他卷积
(Miscellaneous convolutions)

5.1　扩张卷积 (Dilated convolutions)

Readers familiar with the deep learning literature may have noticed the term "dilated convolutions" (or "atrous convolutions", from the French expression *convolutions à trous*) appear in recent papers. Here we attempt to provide an intuitive understanding of dilated convolutions. For a more in-depth description and to understand in what contexts they are applied, see Chen *et al.* (2014); Yu and Koltun (2015).

Dilated convolutions "inflate" the kernel by inserting spaces between the kernel elements. The dilation "rate" is controlled by an additional hyperparameter d. Implementations may vary, but there are usually $d-1$ spaces inserted between kernel elements such that $d=1$ corresponds to a regular convolution.

Dilated convolutions are used to cheaply increase the receptive field of output units without increasing the kernel size, which is especially effective when multiple dilated convolutions are stacked one after another. For a concrete example, see Oord *et al.* (2016), in which the proposed WaveNet model implements an autoregressive generative model for raw audio which uses dilated convolutions to condition new audio frames on a large context of past audio frames.

To understand the relationship tying the dilation rate d and the output size o, it is useful to think of the impact of d on the *effective kernel size*. A

kernel of size k dilated by a factor d has an effective size

$$\hat{k} = k + (k-1)(d-1).$$

This can be combined with 关系 6 to form the following relationship for dilated convolutions:

关系 15. *For any i, k, p and s, and for a dilation rate d,*

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

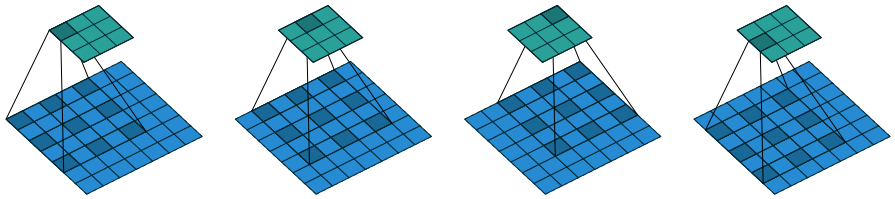

图 5.1: (Dilated convolution) Convolving a 3×3 kernel over a 7×7 input with a dilation factor of 2 (i.e., $i = 7$, $k = 3$, $d = 2$, $s = 1$ and $p = 0$).

Figure 5.1 provides an example for $i = 7$, $k = 3$ and $d = 2$.

参考文献

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., *et al.* (2015). Tensorflow: Large-scale machine learning on heterogeneous systems. *Software available from tensorflow.org*.

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I., Bergeron, A., Bouchard, N., Warde-Farley, D., and Bengio, Y. (2012). Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: A cpu and gpu math compiler in python. In *Proc. 9th Python in Science Conf*, pages 1–7.

Boureau, Y., Bach, F., LeCun, Y., and Ponce, J. (2010a). Learning mid-level features for recognition. In *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR'10)*. IEEE.

Boureau, Y., Ponce, J., and LeCun, Y. (2010b). A theoretical analysis of feature pooling in vision algorithms. In *Proc. International Conference on Machine learning (ICML'10)*.

Boureau, Y., Le Roux, N., Bach, F., Ponce, J., and LeCun, Y. (2011). Ask the locals: multi-way local pooling for image recognition. In *Proc. International Conference on Computer Vision (ICCV'11)*. IEEE.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2014). Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*.

Collobert, R., Kavukcuoglu, K., and Farabet, C. (2011). Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep learning. Book in preparation for MIT Press.

Im, D. J., Kim, C. D., Jiang, H., and Memisevic, R. (2016). Generating images with recurrent adversarial networks. *arXiv preprint arXiv:1602.05110*.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Le Cun, Y., Bottou, L., and Bengio, Y. (1997). Reading checks with multilayer graph transformer networks. In *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*, volume 1, pages 151–154. IEEE.

Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440.

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Saxe, A., Koh, P. W., Chen, Z., Bhand, M., Suresh, B., and Ng, A. (2011). On random weights and unsupervised feature learning. In L. Getoor and

T. Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 1089–1096, New York, NY, USA. ACM.

Visin, F., Kastner, K., Courville, A. C., Bengio, Y., Matteucci, M., and Cho, K. (2015). Reseg: A recurrent neural network for object segmentation.

Yu, F. and Koltun, V. (2015). Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Computer vision–ECCV 2014*, pages 818–833. Springer.

Zeiler, M. D., Taylor, G. W., and Fergus, R. (2011). Adaptive deconvolutional networks for mid and high level feature learning. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2018–2025. IEEE.