第 3 章 卷积神经网络

3.1 卷积神经网络基础

卷积神经网络(Convolutional Neural Networks,CNN)是一类包含卷积计算且具有深度结构的前馈神经网络(Feedforward Neural Networks,FNN),是深度学习的代表算法之一。

对卷积神经网络的研究始于 20 世纪 80 至 90 年代,时间延迟网络和 LeNet-5 是最早出现的卷积神经网络;进入 21 世纪后,随着深度学习理论的提出和数值计算设备的改进,卷积神经网络得到了快速发展,并被应用于计算机视觉、自然语言处理等领域。

卷积神经网络仿造生物的视觉(Visual Perception)机制构建,可以进行监督学习和非监督学习,其隐含层内的卷积核参数共享和层间连接的稀疏性使卷积神经网络能够以较小的计算量对格点化(Grid-Like Topology)特征,例如素和音频进行学习、有稳定的效果且对数据没有额外的特征工程(Feature Engineering)。

3.1.1 卷积神经网络的计算

首先,要讲解基于深度学习的图像识别,卷积神经网络是读者必须掌握的前置知识。在讲解卷积之前,先来了解一下什么是核。

1. 核概念与卷积操作

$g(x,y)$即为核(Kernel),每个小方格上都有一个标量,用于代表权重 ω。$f(x,y)$为输入图像的像素矩阵,每个小方格上都有一个标量,用于代表该图片在该点上的像素值,如图 3-1 所示。图 3-1 中卷积操作定义为核中的元素 ω 与输入数据 $f(x,y)$对应元素进行相乘求和。

(1) 在图 3-1 的示例中,与核中的权重 ω 对应的一共有 9 个元素,相乘求和的结果即为卷积操作的输出。

(2) 核中的权重 ω 就是卷积神经网络训练需要求得的参数。

此外,通过图 3-1 可以发现,由于卷积核的尺寸一般远远小于图像的像素矩阵尺寸,因

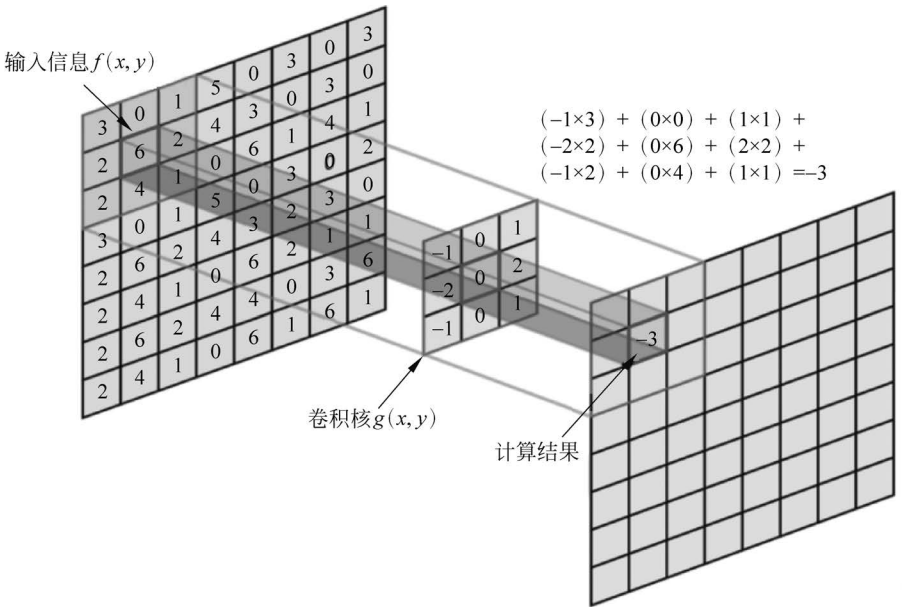


图 3-1 卷积

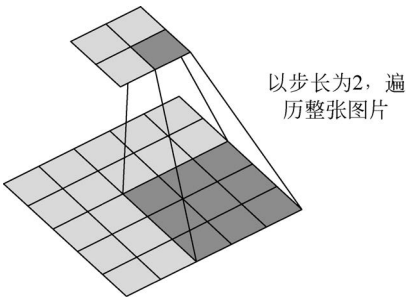


图 3-2 滑动遍历

此，当只对图像进行一次卷积操作时只能处理图像中的一小部分信息，这肯定是不合理的，所以使用卷积处理图像还应存在多个滑动遍历的过程，如图 3-2 所示。卷积核会从图像的起始位置（左上角）开始，以一定的顺序（从左到右，从上到下）遍历整张图像，每滑动一次就与对应的位置做一次卷积操作，直到遍历到最终位置（右下角），这个过程叫作卷积对图像的一次处理。

2．通道

将一个图片数据集抽象为四维[数量、长、宽、色彩]，每个维度都是一个通道（Channel）的概念，一个通道中往往存储着相同概念的数据。例如对于一张 32×32 分辨率的彩色照片来讲，一般将其抽象为向量[1,32,32,3]，其中 3 指的是特征通道，具体来讲，当前的特征是 RGB 共 3 种颜色。颜色通道的原理可以回忆一下小学美术课中的三原色：即世界上任何一种色彩都可以使用红色、绿色和蓝色这 3 种颜色调配得到，因此计算机在存储彩色图像时，也借鉴了这一原理，其存储模式是 3 个相同大小的像素矩阵一同表征一张彩色图像，如图 3-3 所示。卷积核的维度与输入图像的维度相同，也是四维信息，其维度是[3,kernel_H，kernel_W,3]。第 1 个维度的 3 代表图片数量通道，对应图中的 3 个卷积核。第二维度和第三维度是高、宽通道对应着的卷积核的尺寸。第四维度的 3 代表特征通道，对应的是卷积核的通道数目。需要注意的是，卷积核的特征通道与输入信息的特征通道必须是相等的，这是因为在一次卷积操作中，一个卷积核的特征通道会与一个输入信息的特征通道做卷积操作，每个通道一一对应，如图 3-3 所示的箭头。在示例

中，一个卷积核的 3 个特征通道会分别计算得到 3 张特征图，但是，这 3 张特征图最后对应位置求和会得到一个计算结果，即不管一个卷积核的特征通道数量是多少，一个卷积核只能计算得到一张特征图。当然，如果是 n 个卷积核，则可计算得到 n 张特征图。

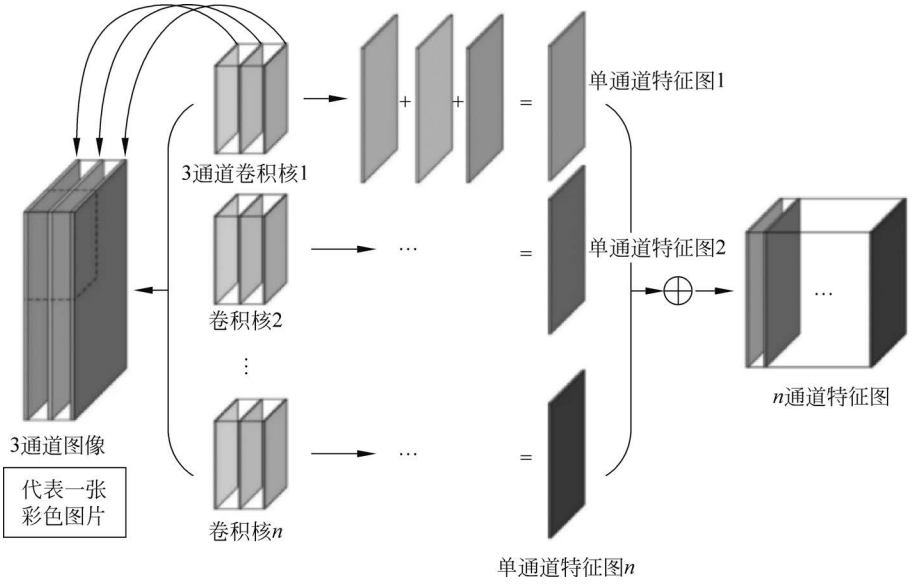


图 3-3　三像素矩阵表征一张彩色图像

最后，回到特征通道的概念。刚刚解释过计算机要表示整张彩色图片，需要特征通道作为三维来存储 RGB 这 3 种颜色。实际上，在深度学习，尤其是卷积神经网络（CNN）中，特征通道的概念被扩展。一开始，网络接受的输入可能仅仅是原始的 RGB 图像，即 3 个特征通道，但是，当数据通过网络的不同层时，网络会自动提取和学习更复杂的特征。每层可能会产生数十甚至数百个新的特征通道，每个通道代表了输入数据的不同特征。例如，在初级层，这些特征可能是边缘、角点或颜色斑块。在更深的层，这些通道可能代表更复杂的特征，如对象的部分或特定的纹理。重要的是，这些特征不是手动设计的，而是通过模型从数据中自动学习得到的。使用多个特征通道可以使网络捕捉到更多维度的信息，从而在执行任务（如图像分类、物体检测等）时更加精确。每个通道都携带着不同的信息，网络结合这些信息来做出判断。

3. 填充

经过对卷积处理图像的方式和核概念的认识后，可以将卷积神经网络理解为一个卷积核在输入图片上遍历的过程，在遍历过程中卷积核与输入信息之间的对应点的乘积求和即为卷积输出。输出结果的尺寸一般要小于输入信息，除非卷积核的大小为 1×1，如图 3-4 所示。

很多情况下，希望在不使用 1×1 大小卷积核的前提下，可以调整卷积输出结果的尺寸。此时可以在输入信息的四周填充一圈新像素（一般填充 0 值像素），使卷积核遍历图片后得

到的卷积输出大小不变。填充的像素多少与卷积核的尺寸大小成正相关，填充(Padding)过程如图 3-5 所示。当卷积核尺寸为 3 且步长大小为 1 时，需要填充一行一列像素使输出大小与输入信息一致。

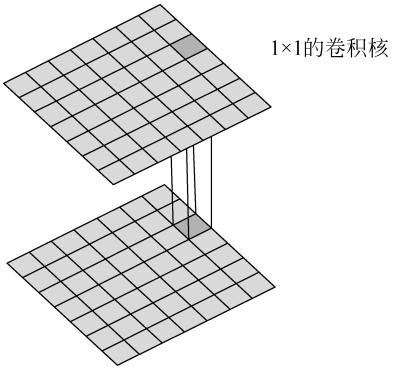


图 3-4　卷积

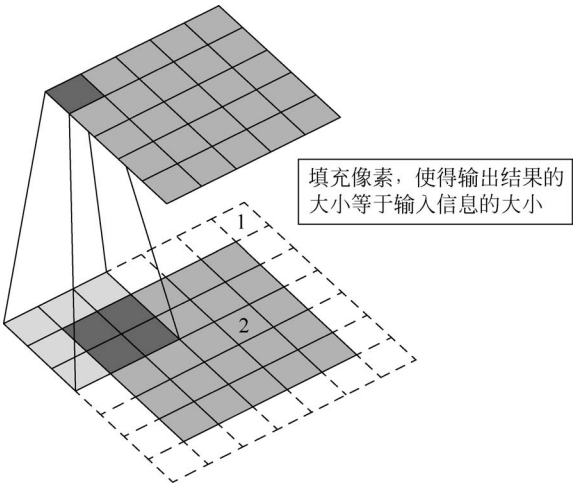


图 3-5　带填充的卷积

此外，可以观察图 3-5 中 1 号和 2 号的两像素。根据卷积的遍历规则，1 号像素在整个遍历过程中只会被计算一次，而 2 号像素在整个遍历过程中会被计算多次。这体现了卷积算法的一个性质：更关注图片中心区域的信息，因此，如果想让卷积操作对整张图片的关注度均衡，则可以通过填充的方式将原本在边缘的像素信息变换到靠近中心的位置。

最后，卷积计算结果的尺寸与卷积核、步长、填充和输入信息尺寸这 4 个因素相关。先定义几个参数：输入图片大小为 $W\times W$、卷积核大小为 $F\times F$、步长为 S、填充的像素数为 P。于是可以得出卷积结果的尺寸计算公式为

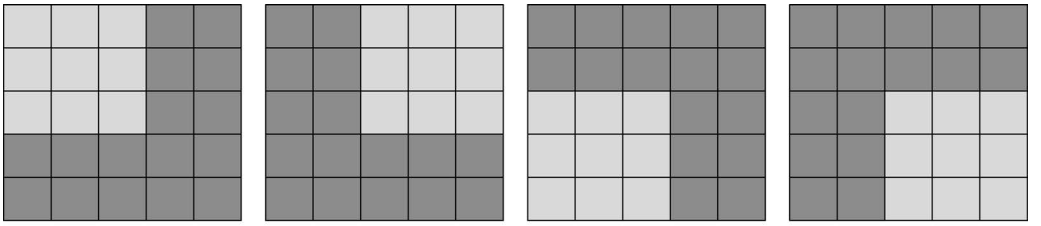
$$N=\frac{W-F+2P}{S}+1 \tag{3-1}$$

即，卷积输出结果的尺寸为 $N\times N$。

4. 步长

此外，通过图 3-1 可以发现，由于卷积核的尺寸一般远远小于图像的像素矩阵尺寸，因此，只对图像进行一次卷积操作只能处理图像中的一小部分信息，这肯定是不合理的，所以使用卷积处理图像还应存在多个滑动遍历的过程。卷积核会从图像的起始位置(左上角)开始，以一定的顺序(从左到右，从上到下)遍历整张图像，每滑动一次就与对应的位置做一次卷积操作，直到遍历到最终位置(右下角)，这个过程叫作卷积对图像的一次处理，卷积核每次滑动的像素大小被定义为步长(Stride)，如图 3-6 所示，卷积核的数量为 1 个，每个卷积核的形状为 1×3×3，其中 1 为卷积核特征通道的数量，这与输入数据的特征通道数量是相等的。卷积核在计算过程中的填充为 0，步长为 2，最后输出结果的通道数量也为 1，其中(a)图为卷积核的起始位置；(b)图表示卷积核向右滑动一次；(c)图表示遍历完上一行的像素

后向下滑动一次；(d)图为卷积核的最终位置。



(a) 初始情况　(b) 向右滑动　(c) 向下滑动　(d) 最终情况

图 3-6　步长

3.1.2　卷积的设计思想

1. 平移不变性

需要注意的一点是：卷积核从图像的初始位置(左上角)滑动到最后位置(右下角)，在这个过程中，卷积核中的参数是不会发生改变的，这叫作参数共享，是卷积操作的重要性质之一。相反，参数独立的情况为卷积核每滑动一次，其中的参数就要重新计算。

参数共享的目的从某种角度来讲是在模仿人类的一种视觉习惯：平移不变性。换句话说，就是在图像中，只要是同一种特征，那么不管这个特征被平移到图像中的什么位置，人类都能很好地进行识别。实际上，卷积的设计是天然符合这个特性的：首先，卷积核在输入信息上做卷积的目的就是为了识别某一种特征(本章后续会详细解释)；其次，由于滑动遍历的原因，不管要识别的特征出现在图片中的什么位置，卷积都可以通过滑动的方式，滑动到该特征的上面进行识别，但是，这有一个前提，就是该卷积核从图像的起始位置滑动到结尾位置，在这个过程中，寻找的都是同一种特征。这个前提可以通过参数共享的方式实现，即卷积核在对图像做一次完整的遍历过程中不发生改变，一个不变的卷积核寻找的肯定是同一种特征。

2. 局部相关性

从图像的性质来讲，卷积的设计天然符合图像的局部相关性。首先关于图像的局部相关性的理解可以列举一个场景：从图 3-7 中人物的眼睛附近随机选取一像素值 a_1，如果单独地把 a_1 从图像的像素矩阵中取出来，则 a_1 仅仅是一个数值，它代表不了任何东西，这是没有意义的，但是，如果把 a_1 再放回原本的像素矩阵中，它就可以跟周边的像素值一起表示眼睛这一特征，这叫作相关性。此外，考虑离 a_1 相距较远的其他像素值 a_2(例如图像右下角衣服中的像素值)，由于两像素在原图中的距离较远，所以它们之间的联系也是比较小的，这就体现了局部相关性，因此，图像是一种局部相关的数据，在此性质的背景下，全连接神经网络这种计算全局信息的方式反而是

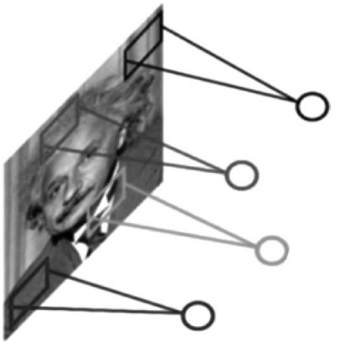


图 3-7　局部相关性

冗余的，是不符合图像性质的，而卷积处理局部的方式则显得更加合理。

实际上，对图片进行卷积操作就是对卷积核与原图片进行点积操作。点积的数学解释可以解释为两个向量之间的相似度。在当前的例子里，可以说成卷积核与原图的相似度，卷积的结果越大，说明图片中某位置和卷积核的相似度越大，反之亦然。如果把卷积核作为特征算子或者特征向量，则卷积的过程就是通过移动卷积核在原图中的对应位置，不断地去寻找原始数据中是否存在跟卷积核表征相似的特征，这在图片识别中意义重大。例如，判断一张图片是否为车子，假设卷积模型设置了 4 个核，它们的特征可能代表[“轱辘”，“车窗”，“方向盘”，“车门”]，通过卷积核在原图上进行匹配，进而综合判断图像中是否存在这 4 种特征，如果存在，则该图片大概率是车子。

实际上，深度卷积神经网络就是去求解这些卷积核的一种网络。它们不是凭借经验随便定义的，而是通过网络不断地学习更新参数得来的，而卷积神经网络的不易解释性就在于此，随着模型变得复杂，抽象出很多不同的卷积核，我们难以去解释每个核的具体含义，也难以介绍每个中间层和中间节点的含义。

3. 人类的视觉习惯

最后可以从人类的视觉习惯来理解卷积的设计思想。相较于全连接神经网络，卷积神经网络的计算方法更符合人类的视觉习惯。可以想象一个场景：当突然置身于一个复杂且陌生的环境时，是怎么快速地获得周边信息的？

如果以神经网络的方式，由于神经网络的计算是全局性的，为了模仿这一性质，在观察场景时需要同时观察全局，思考全局。这其实并不是人类视觉的观察习惯。我们往往会选择先观察一个感兴趣的局部区域，观察时，只需思考这个局部区域有什么。之后，再选择查看其他地方，看到哪里思考到哪里，通过有规律的扫视对全局做完整的观察。这其实与卷积神经网络先观察局部，再通过滑动的方式遍历全局，滑动到哪里就计算到哪里是一样的，所以说，卷积的设计更符合人类的视觉习惯。

3.1.3 卷积进行特征提取的过程

让我们来看一个识别字母 X 的图像识别示例，如图 3-8 所示。在图像的像素矩阵中，白色的像素块的值为 1，黑色的像素块的值为负 1。在图像上方的 3 个小矩阵分别是 3 个不同的卷积核。现在使用第 1 个卷积核在图像中进行滑动遍历：当这个卷积核滑动到图 3-8 所示的位置时(图像左上角的方框)，其卷积计算结果等于 9，此时 9 为这个 3×3 卷积核能够计算得到的最大值。

继续思考，当在图 3-8 所示的位置附近做卷积操作时，卷积计算结果应该是比 9 要小的数值，如图 3-9 所示，向右滑动一像素位置的卷积计算结果为－1。

现在，继续滑动卷积核，如图 3-10 所示，直到滑动到图 3-10 所示的右下角的位置时，其卷积计算结果又等于 9。思考图 3-8 所示的左上角的卷积核位置及图 3-10 所示的右下角的卷积核位置是否有什么联系？答案很简单，它们的特征相同。

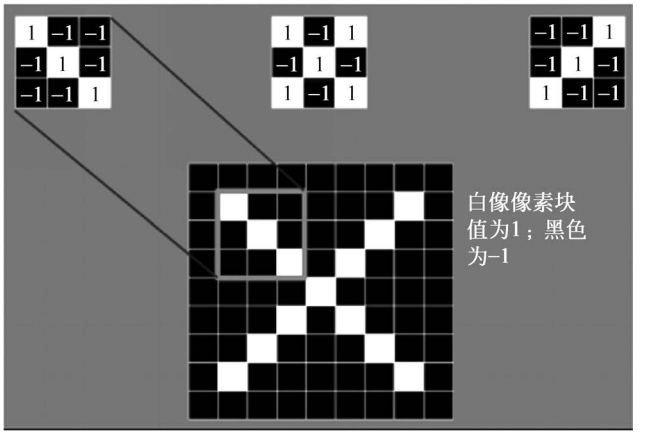


图 3-8　卷积的特征提取过程

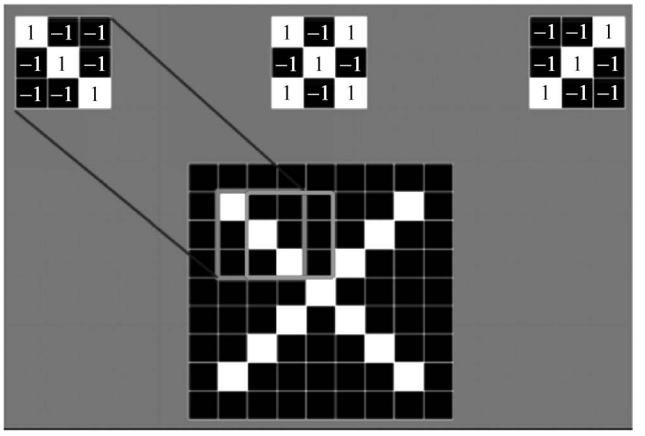


图 3-9　卷积的特征提取过程

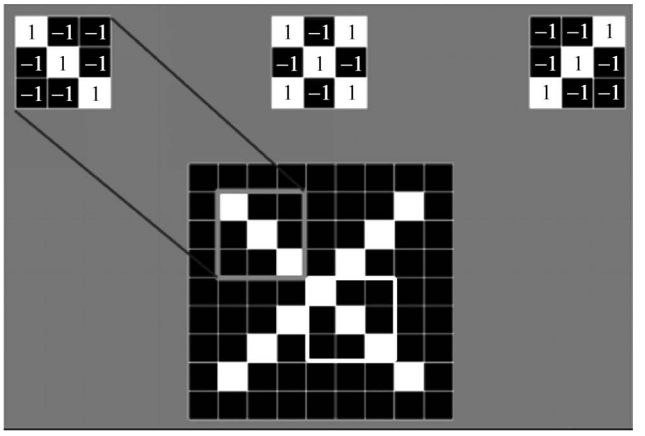


图 3-10　卷积的特征提取过程

因此，答案就出来了：第 1 个卷积核就是在图像里滑动遍历，然后寻找一个是否跟它长得一模一样的这种对角特征；如果找到了，卷积计算结果就是最大值；如果没有找到，卷积计算结果就是一个非极大值；通过极大值与非极大值的区分，就可以完成这种对角特征与其他特征的区分。卷积就是通过这种方法对输入信息进行特征提取的。简而言之，可以把卷积核看作识别某种特征的模式，卷积核的目的就是尝试在图像中提取这种特征。

需要注意的是，在一个卷积操作中，往往会选择使用不同的卷积核对图像做卷积操作，如图 3-11 所示。

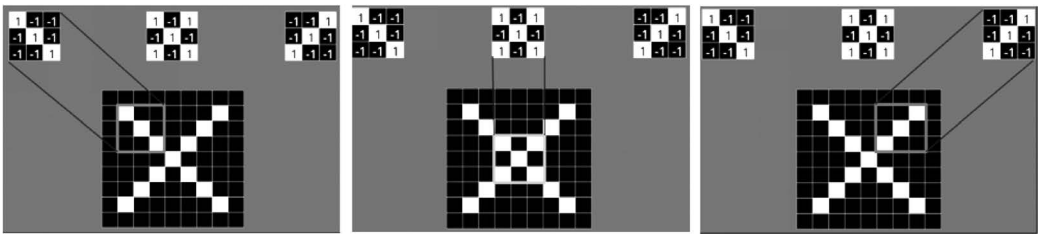


图 3-11　卷积的特征提取过程

其目的是：希望不同的卷积核可以从图像中提取不同的特征。因为当提取的特征太少时，是没有办法完成图像识别这个任务的。例如，我们不能凭借眼睛这一种特征来识别猫和狗这两个类别，往往需要根据眼睛、嘴巴、外形、毛发、耳朵等多种特征才能对猫和狗正确地进行识别。

3.1.4　池化与采样

从某种程度上，池化可以看作特殊的卷积，因为池化操作包含卷积中常见的概念（核、填充、步长）；区别在于，池化的核与输入信息的对应位置元素进行的操作不同，在卷积中，该操作是相乘求和，而在池化中，这个操作是求平均或最大，分别对应平均池化（Average Pooling）和最大池化（Max Pooling）。池化最主要的作用就是对输入信息进行降维，如图 3-12 所示。池化的核是左上角方框所示部分，大小为 2×2；原始输入数据的填充为 0，因为在原始数据的四周并没有像素填充；池化的初始位置为左上角的 2×2 矩阵，滑动一次后到了右上角的 2×2 矩阵，滑动了两像素位置，即池化操作的步长为 2。

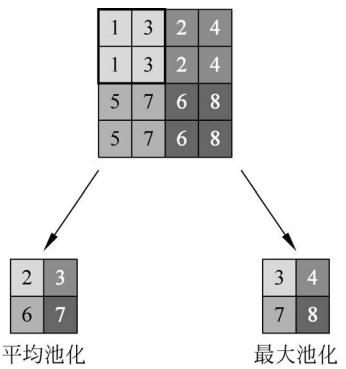


图 3-12　平均池化和最大池化

至于左下角的 2×2 矩阵为平均池化的结果，例如第 1 个元素 2 的计算过程为（1＋3＋1＋3）/4，即池化的核与输入信息的对应位置做平均的结果，而右下角的 2×2 矩阵为最大池化的结果，例如第 1 个元素 3 的计算过程为 Max(1，3，1，3)，即池化的核与输入信息的对应位置进行比较，取最大的结果。

最大池化的主要作用是特征提取。这种技术通过在每个窗口中保留最大值实现，从而有效地识别图像中

的显著特征，如边缘和纹理。同时，最大池化也起到了降维的作用，通过减小数据的空间尺寸（例如，从 2×2 像素块中选择最大值），帮助减小计算复杂性和防止过拟合。

继续刚才的例子，从图像的起始位置开始做卷积操作，每次滑动的步长为 2，直到末尾的位置。当第 1 个卷积核依次滑动到图中的 4 个位置时，卷积结果分别是 9、3、−1、−3，如图 3-13 所示。

这 4 个值在本次卷积的计算结果中大致如图 3-14 所示，注意，滑动到其他位置的卷积计算结果也应该有输出值，这里以“…”代替。池化操作一般接在卷积操作之后，即卷积的计算结果是池化操作的输入信息，在当前案例中，图 3-14 即为卷积的计算结果。当使用一个 2×2 大小的池化核对上述卷积结果进行操作时，我们发现被保留下来的数值是 9。对此的解释是：9 正是卷积操作在原图中找到的对角特征，通过最大池化的方式被保留了下来；同时把卷积操作认为不是很重要的特征（例如 3、−1、−3）删掉，这就完成了特征汇聚的过程。

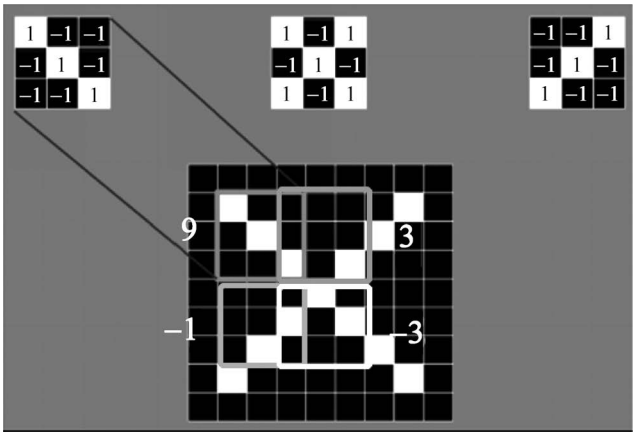


图 3-13　特征汇聚示例

…	…	…	…
…	9	3	…
…	−1	3	…
…	…	…	…

图 3-14　特征汇聚示例

相比之下，平均池化的作用是特征融合。这种技术通过计算每个窗口的平均值实现，有助于平滑图像的局部特征，使模型更加关注于图像的整体结构而非局部细节。与最大池化类似，平均池化也通过降低数据的空间尺寸来减少维度，有助于降低计算负担并减少过拟合的风险。

3.1.5　卷积神经网络的感受野

感受野（Receptive Field）是指卷积神经网络中某一神经元在输入图像上感受到的区域大小。换句话说，感受野描述了神经元对输入图像的哪些部分产生响应。在卷积神经网络中，随着网络层数的增加，每个神经元的感受野会逐渐变大，能够感受到更广阔的输入图像区域，从而提高网络对整个图像的理解能力。

举个例子来理解，假设现在有两层卷积神经网络，它们的卷积核大小都是 3×3。在网络的第 1 层中，每个神经元计算的输入信息范围是由卷积核定义的，即 3×3＝9，也就是说第 1 层神经元的感受野是 9，但在第 2 层中，因为层级结构的原因，感受野会明显增加，如图 3-15 所示。

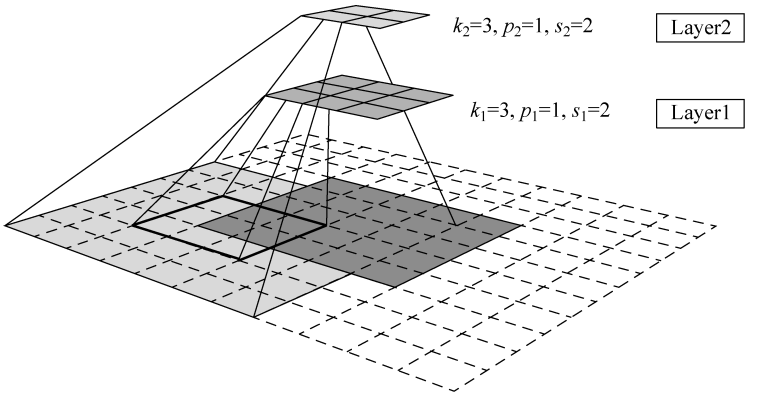


图 3-15　卷积层级结构的感受野

卷积神经网络感受野的作用如下。

(1) 提高特征表征能力：随着网络层数的增加，每个神经元的感受野增大，能够感受到更广阔的输入图像区域，从而提高网络对整个图像的理解能力，进一步提高特征表征能力。这也是卷积神经网络为什么要设计成层级结构的原因。

(2) 提高模型的稳健性：感受野的增大能够提高模型的稳健性，使网络在面对不同尺寸、姿态、光照等情况下都能够有效地进行特征提取和图像识别。此外，通过适当地调整卷积核的大小和填充等参数，可以实现对不同尺寸的输入图像进行有效处理。

计算感受野大小的公式是基于递归计算每层神经元在输入图像上感受野的大小，可以使用式(3-2)计算：

$$R_i = R_{i-1} + (k_i - 1) \times s_i \tag{3-2}$$

其中，R_i 表示第 i 层神经元在输入图像上感受野的大小，R_{i-1} 表示第 $i-1$ 层神经元在输入图像上感受野的大小，k_i 表示第 i 层卷积核的大小，s_i 表示第 i 层卷积核的步长。

在计算感受野时，一般从输入层开始逐层计算，假设输入图像的大小为 $H \times W$，则输入层中的每个神经元的感受野大小为 1，即 $R_1 = 1$。注意，这里的输入层可以看作模型的第 0 层，而不是图 3-15 中的 Layer 1。对于之后的每层都可以通过式(3-2)计算出感受野的大小。

需要注意的是，式(3-2)只考虑了卷积层的计算方式，而对于池化层等其他操作，其感受野的计算方式可能会有所不同。此外，还有一些基于卷积的变体操作也可以改变感受野，例如膨胀卷积等。到此，卷积模型的标准结构已经全部介绍完了。

3.1.6　卷积模型实现图像识别

卷积模型实现图像识别的标准网络结构如图 3-16 所示。卷积网络被分为两个阶段，分别用两个方框圈出。

在左侧的方框内，模型的第一阶段是通过不断堆叠卷积层和池化层组成的(虽然图中只显示了一层卷积和池化，实际的模型中会有多层)，其中卷积的目的是做特征提取；池化的目的是做特征汇聚。

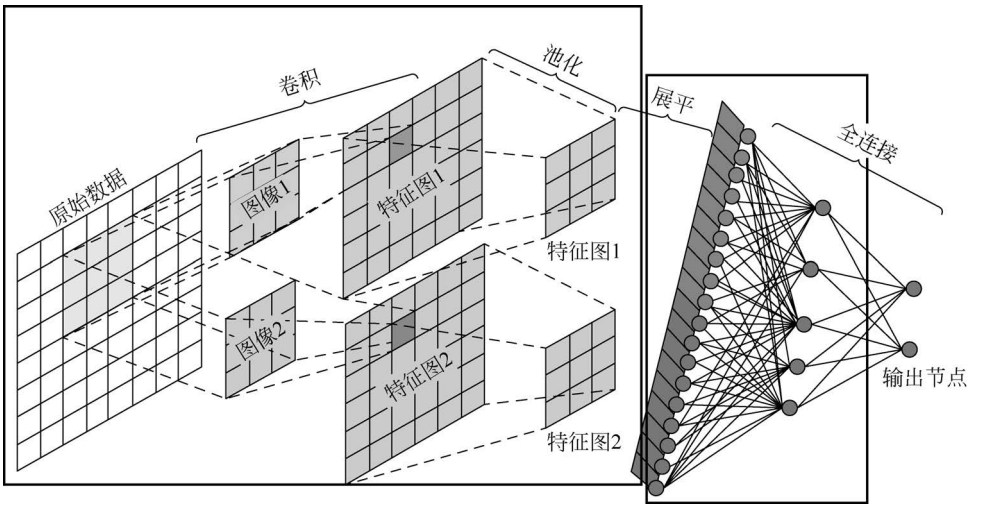


图 3-16　卷积模型实现图像识别

在右侧的方框内，模型的第二阶段是通过不断堆叠全连接神经网络层组成的，其目的是对上一阶段输出的特征进行学习，判断这些特征最有可能属于哪个类别。

在送进全连接神经网络之前，有一步叫作展平(Flatten)的操作。对此的解释是：卷积的计算结果是一组特征图，这些数据是有空间维度的(高度和宽度)，但是全连接神经网络层能接受的数据是向量格式(维度等于 1 的数据)，因此，展平操作的目的是把多维的特征图压缩成长度为“特征图高×特征图宽×特征图数量”的一维数组，然后与全连接层连接，通过全连接层处理卷积操作提取的特征并输出结果。

此外，在做图像识别时，一般习惯在模型的输出结果后增加一个简单的分类器，例如 Softmax 分类器，其作用是把输入数据归一化到[0，1]区间，并且所有归一化后的元素相加等于 1，归一化后的数值即可表示图像属于某种类别的可能性了。

Softmax 分类器的作用简单地说就是计算一组数值中每个值的占比，公式一般描述为设一共有 n 个用数值表示的分类 S_k，$k \in (0, n]$，其中 n 表示分类的个数。那么 Softmax 函数的计算公式为

$$P(S_i) = \frac{e^{gi}}{\sum_{k}^{n} e^{gk}} \tag{3-3}$$

其中，i 表示 k 中的某个分类，g_i 表示该分类的值。

3.1.7　第 1 个卷积神经网络模型：LeNet

1. LeNet 介绍

LeNet-5 模型诞生于 1994 年，是最早的卷积神经网络之一，由 Yann LeCun 完成，推动了深度学习领域的发展。彼时，没有 GPU 帮助训练模型，甚至 CPU 的速度也很慢，神经网

络模型处理图像时的大量参数并不能通过计算机很好地得到计算，LeNet-5 模型通过巧妙的设计，利用卷积、参数共享及池化等操作提取特征，避免了大量的计算成本，最后使用全连接神经网络进行分类识别。从此卷积成为图像处理中的可行方式。

LeNet 作为最初的卷积神经网络，其模型结构的组成较为简单：两个卷积层、两个下采样和 3 个全连接层，如图 3-17 所示。

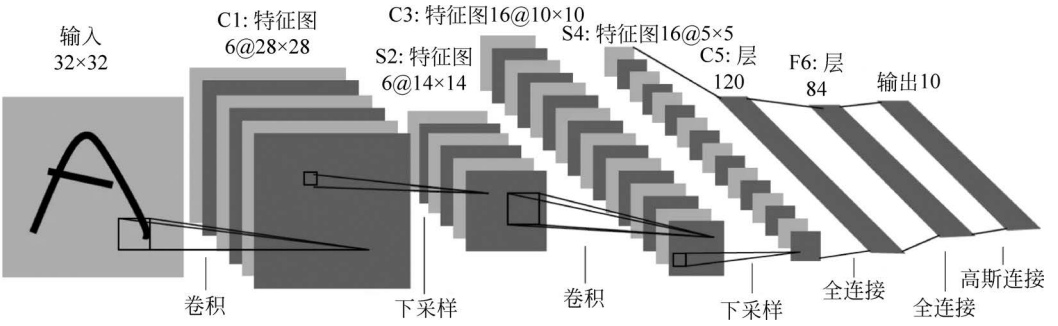


图 3-17　LeNet 结构图

其中，卷积层和池化层负责对原始图像进行特征提取，全连接层负责对卷积池化提取的特征进行学习，进一步根据这些特征来判断该输入图片属于哪一个类别。

本文的重点在于代码实现。首先介绍卷积、池化和全连接层在 PyTorch 深度学习框架中对应的 API。

2. 卷积操作在 PyTorch 中的 API

卷积操作在 PyTorch 中被封装成一个名称为 torch.nn.Conv2d 的类，代码如下：

```
class
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,
dilation=1,groups=1,bias=True,padding_mode='zeros',device= None,dtype=None)
```

其中，类的参数解释如下。

(1) in_channels (int)：输入图像中的通道数。

(2) out_channels (int)：由卷积产生的通道数。

(3) kernel_size (int or tuple)：卷积核的大小。

(4) stride (int or tuple, optional)：卷积的步长，默认值为 1。

(5) padding (int, tuple or str, optional)：在输入的 4 边都进行填充。默认值为 0。

(6) padding_mode (string, optional)：padding 的模式，可选参数有 zeros、reflect、replicate 和 circular，分别代表零填充、镜像填充、复制填充和循环填充。默认值为 0 填充。

(7) dilation (int or tuple, optional)：卷积核中元素之间的间距，对应卷积变体之扩张卷积，默认值为 1。

(8) groups (int, optional)：从输入通道到输出通道的阻塞连接的数量，对应卷积变体之组卷积，默认值为 1。

(9) bias (bool, optional)：如果为真，则给输出增加一个可学习的偏置，默认值为真。

3. 最大池化操作在 PyTorch 中的 API

池化操作在 PyTorch 中被封装成一个名称为 torch.nn.MaxPool2d 的类，代码如下：

```
class
torch.nn.MaxPool2d(kernel_size, stride=None, padding=0, dilation=1, return_
indices=False,ceil_mode=False)
```

其中，类的参数解释如下。

(1) kernel_size：池化核的大小。

(2) stride：池化的步长。默认值为 kernel_size。

(3) padding：在输入的 4 边都进行填充。隐含的零填充，在两边添加。

(4) dilation：池化核中元素之间的间距，默认值为 1。

(5) return_indices：如果为真，则将与输出一起返回最大索引。对以后的 torch.nn.MaxUnpool2d 有用。

(6) ceil_mode：如果为真，则将使用 ceil 而不是 floor 来计算输出形状。

4. 全连接神经网络层在 PyTorch 中的 API

全连接神经网络层在 PyTorch 中被封装成一个名称为 torch.nn.Linear 的类，代码如下：

```
class
torch.nn.Linear(in_features,out_features,bias=True,device=None,dtype=None)
```

其中，类的参数解释如下。

(1) in_features：每个输入样本的大小。

(2) out_features：每个输出样本的大小。

(3) bias：如果设置为 False，则该层将不学习加性偏置。默认值为 True。

5. 激活函数 ReLU 在 PyTorch 中的 API

常见的激活函数 ReLU 在 PyTorch 中被封装成一个名称为 torch.nn.ReLU 的类，代码如下：

```
class torch.nn.ReLU(inplace=False)
```

其中，参数一般习惯指定为 True，当参数为真时，ReLU 的计算在底层会节省计算和存储资源。

6. 使用 PyTorch 搭建 LeNet 卷积模型

使用 PyTorch 搭建 LeNet 卷积模型，代码如下：

```
#第 3 章/LeNet.py
import torch.nn as nn
```

```
import torch

class Model(nn.Module):
    def __init__(self):                          #函数 init 定义的模型层级结构
        super().__init__()
        self.conv1 = nn.Conv2d(1, 6, 5)          #输入为单通道灰度图,第 1 次卷积
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(2)             #第 1 次池化
        self.conv2 = nn.Conv2d(6, 16, 5)         #第 2 次卷积
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(2)             #第 2 次池化
        self.fc1 = nn.Linear(256, 120)           #第 1 次全连接
        self.relu3 = nn.ReLU()
        self.fc2 = nn.Linear(120, 84)            #第 2 次全连接
        self.relu4 = nn.ReLU()
        self.fc3 = nn.Linear(84, 10)             #第 3 次全连接
        self.relu5 = nn.ReLU()

#函数 forward 定义的模型的前向计算过程,其中参数 x 代表输入图像
#下述过程表示图像 x 先经过第 1 次卷积 conv1 得到结果 y,再送入激活函数,得到新的结果 y,
#再送入第 2 次卷积,以此类推
    def forward(self, x):
        y = self.conv1(x)
        y = self.relu1(y)
        y = self.pool1(y)
        y = self.conv2(y)
        y = self.relu2(y)
        y = self.pool2(y)
        y = y.view(y.shape[0], -1)               #展平操作(Flatten)
        y = self.fc1(y)
        y = self.relu3(y)
        y = self.fc2(y)
        y = self.relu4(y)
        y = self.fc3(y)
        y = self.relu5(y)
        return y
```

3.2 卷积的变体算法

3.2.1 逐通道卷积

逐通道卷积(Depthwise Convolution)又称逐层卷积。首先通过一个例子来对比讲解逐层卷积与常规卷积的关系。回忆一下常规的卷积操作,在卷积操作中对于 5×5×3 的输入信息,如果想要得到 3×3×4 的特征图输出,则卷积核的形状为 3×3×3×4,如图 3-18 所示。

卷积层共有 4 个卷积核,每个卷积核包含一个通道数为 3(与输入信息通道相同),并且

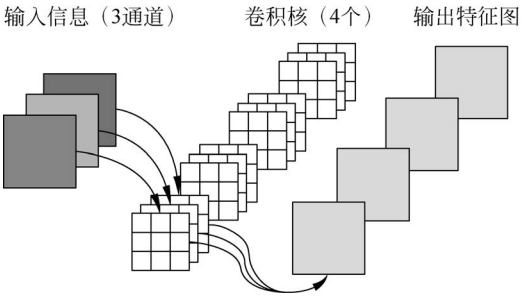


图 3-18　常规的卷积计算过程

尺寸为 3×3 的卷积核，因此卷积层的参数数量可以用公式：卷积层的参数量＝卷积核宽度×卷积核高度×输入通道数×输出通道数来计算，即

$$N_std = 4 \times 3 \times 3 \times 3 = 108 \tag{3-4}$$

卷积层的计算量公式为卷积层的计算量＝卷积核宽度×卷积核高度×(输入信息宽度－卷积核宽度＋1)×(输入信息高度－卷积核高度＋1)×输入通道数×输出通道数，即

$$C_std = 3 \times 3 \times (5-2) \times (5-2) \times 3 \times 4 = 972 \tag{3-5}$$

逐通道卷积的一个卷积核只有一个通道，输入信息的一个通道只被一个卷积核卷积，这个卷积过程产生的特征图通道数和输入的通道数相等，如图 3-19 所示。

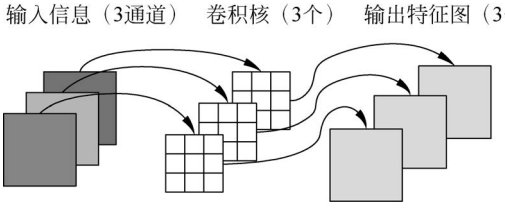


图 3-19　逐通道卷积的卷积计算过程

一张 5×5 像素的三通道彩色输入图片(形状为 5×5×3)，逐通道卷积每个卷积核只负责计算输入信息的某个通道。卷积核的数量与输入信息的通道数相同，所以一个三通道的图像经过卷积运算后一定生成了 3 张特征图。卷积核的形状一定为卷积核 W×卷积核 H×输入数据的通道数 C。

此时，卷积部分的参数个数的计算为

$$N_depthwise = 3 \times 3 \times 3 = 27 \tag{3-6}$$

卷积操作的计算量为

$$C_depthwise = 3 \times 3 \times (5-2) \times (5-2) \times 3 = 243 \tag{3-7}$$

由于这个计算量相较于常规卷积操作在参数量和计算复杂度上有非常大的下降，因此被广泛地运用在一些移动端设备上，但是，逐通道卷积输出的特征图的数量与输入层的通道数相同，无法在通道维度上扩展或压缩特征图的数量，而且这种运算对输入层的每个通道独立进行卷积运算，没有有效地利用不同通道在相同空间位置上的特征相关性。简而言之，虽然减少了计算量，但是失去了通道维度上的信息交互，因此需要逐点卷积来将这些特征图进

行组合，在通道维度上实现信息交互。

3.2.2 逐点卷积

逐点卷积(Pointwise Convolution)的运算与常规卷积运算非常相似，其实就是 1×1 的卷积。它的卷积核的形状为 1×1×M，M 为上一层输出信息的通道数。逐点卷积的每个卷积核会将上一步的特征图在通道方向上进行加权组合，计算生成新的特征图。每个卷积核都可以生成一个输出特征图，而卷积核的个数就是输出特征图的数量，逐点卷积如图 3-20 所示。

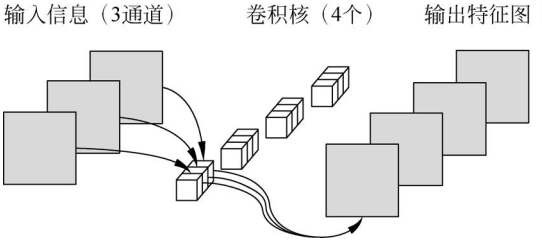


图 3-20 逐点卷积的卷积计算过程

此时，逐点卷积中卷积涉及的参数个数可以计算为

$$\text{N_pointwise} = 1 \times 1 \times 3 \times 4 = 12 \tag{3-8}$$

卷积操作的计算量则为

$$\text{C_pointwise} = 1 \times 1 \times 3 \times 3 \times 3 \times 4 = 108 \tag{3-9}$$

经过逐点卷积之后，4 个卷积核输出了 4 张特征图，与常规卷积的输出维度相同。逐点卷积是一种卷积神经网络中常用的操作，其主要的两个好处如下。

(1) 参数量较小：相比于传统的卷积操作，逐点卷积的卷积核大小为 1×1，因此需要学习的参数数量较少，可以在保证较高准确率的同时减少模型的大小，提高模型的训练和推理效率。

(2) 可以在不改变特征图大小的情况下进行通道数的变换：逐点卷积在每个位置独立地对输入的每个通道进行卷积操作，因此可以通过设置卷积核的数量将输入的通道数转换为任意数量的输出通道数，这种操作在 CNN 中非常常见，被广泛地用于实现网络的深度和宽度缩放。此外，由于逐点卷积可以保留原始特征图的空间维度大小，因此可以在保持空间分辨率不变的情况下对特征图的通道数进行变换，从而更好地提取特征。

最后，值得注意的是逐点卷积和全连接层在某些情况下是等价的，这主要因为它们的计算方式相似。

在逐点卷积中，卷积核大小为 1×1，通常用于调整通道数。假设输入张量大小为 $H \times W \times C_{\text{in}}$，输出张量大小为 $H \times W \times C_{\text{out}}$，则逐点卷积的权重矩阵大小为 $1 \times 1 \times C_{\text{in}} \times C_{\text{out}}$，其中 C_{in} 是输入张量的通道数，C_{out} 是输出张量的通道数。逐点卷积的计算可以表示为

$$y_{i,j,k} = \sum_{c=1}^{C_{\text{in}}} x_{i,j,c} \times w_{1,1,c,k} \tag{3-10}$$

其中，$x_{i,j,c}$ 表示输入张量在位置(i,j)处的第 c 个通道的值，$w_{1,1,c,k}$ 表示逐点卷积的第 c 个通道和第 k 个通道之间的权重，$y_{i,j,k}$ 表示输出张量在位置(i,j)处的第 k 个通道的值。

全连接层的作用是将输入张量中的每个像素与权重矩阵中的每个元素相乘，并将结果相加得到输出。如果将全连接层的输入张量表示为一维向量 $\boldsymbol{x}$，将权重矩阵表示为二维矩阵 $\boldsymbol{W}$，则全连接层的计算可以表示为

$$\boldsymbol{y}=\boldsymbol{W}\boldsymbol{x} \tag{3-11}$$

其中，$\boldsymbol{y}$ 表示输出向量，$\boldsymbol{x}$ 表示输入向量，$\boldsymbol{W}$ 表示权重矩阵。

从式(3-10)和式(3-11)可以看出，逐点卷积和全连接层的计算方式非常相似。实际上，当逐点卷积的卷积核大小为 1×1 时，它就等价于全连接层，因为它们都是对输入信息的全部元素逐一地进行权重相乘计算，然后求和得到输出。此时，逐点卷积的权重矩阵可以看作全连接层的权重矩阵，逐点卷积的计算可以看作全连接层的计算。

因此，当逐点卷积的卷积核大小为 1×1 时，它可以被视为全连接层的一种特殊形式，这也解释了为什么逐点卷积和全连接层在某些情况下是等价的。

3.2.3　深度可分离卷积

深度可分离卷积(Depthwise Separable Convolution)的概念很简单，它是逐通道卷积和逐点卷积配合使用而得来的。它在降低模型计算复杂度的同时，还能提升卷积的准确度，是更有效的卷积方式。继续使用 3.2.1 节和 3.2.2 节的例子，来对比一下常规卷积和深度可分离卷积的参数量和计算量。

常规卷积的参数个数由式(3-4)得出为 108，而深度可分离卷积的参数个数由式(3-6)的逐通道卷积参数和式(3-8)的逐点卷积参数两部分相加得到：

$$\text{N_separable}=\text{N_depthwise}+\text{N_pointwise}=12+27=39 \tag{3-12}$$

相同的输入，同样是得到 4 张特征图的输出，深度可分离卷积的参数个数是常规卷积的约 1/3，因此，在参数量相同的前提下，采用深度可分离卷积的神经网络层数可以做得更深。

在计算量对比方面，常规卷积的计算量由式(3-5)得出为 972，而深度可分离卷积的计算量由式(3-7)的逐通道卷积的计算量和式(3-9)的逐点卷积的计算量两部分相加得到：

$$\text{C_separable}=\text{C_depthwise}+\text{C_pointwise}=243+108=351 \tag{3-13}$$

相同的输入，同样是得到 4 张特征图的输出，深度可分离卷积的计算量也是常规卷积的约 1/3，因此，在计算量相同的情况下，通过深度可分离卷积，可以加深神经网络的层数。

除了计算更高效，深度可分离卷积可以看作对常规卷积计算特征的过程进行了分布规划。在常规的卷积中，卷积核是四维的(数量通道，特征通道，高度通道，宽度通道)，其中特征通道代表特征维度，高度通道和宽度通道代表空间维度，因此常规卷积可以同时在特征维度和空间维度对输入数据进行处理，而逐层卷积核的维度是(数量通道，1，高度通道，宽度通道)，只能对空间信息进行处理；逐点卷积核的维度是(数量通道，特征通道，1，1)，只能对特征信息进行处理，因此深度可分离卷积对特征信息和空间信息的处理是解耦的。

3.2.4 组卷积

对比常规卷积和深度可分离卷积来介绍组卷积(Group Convolution)。

常规卷积如图 3-21 所示,为了方便理解,图中只有一个卷积核,此时输入及输出数据如下。

(1) 输入数据尺寸:$W\times H\times C$,分别对应的宽、高、通道数。

(2) 单个卷积核尺寸:$k\times k\times C$,分别对应单个卷积核的宽、高、通道数。

(3) 输出特征图尺寸:$W'\times H'$,输出通道数等于卷积核数量,输出的宽和高与卷积步长有关,这里不关心这两个值。

(4) 参数量:$k\times k\times C$。

(5) 运算量:$k\times k\times C\times W'\times H'$(只考虑浮点乘数量,不考虑浮点加)。

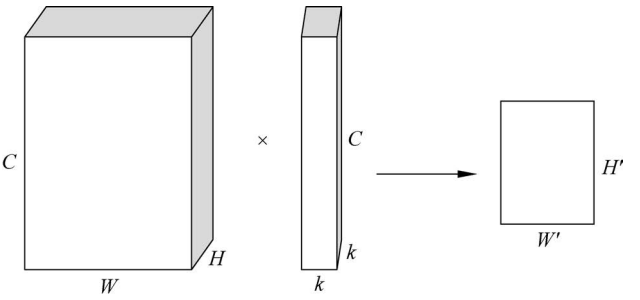


图 3-21 常规卷积

将图 3-21 卷积的输入特征图分成 2 组,卷积核也相应地分成 2 组,在对应的组内做卷积,如图 3-22 所示,其中分组数为 2,即图片上方的一组输入数据只和上方的一组卷积核做卷积,下方的一组输入数据只和下方的一组卷积核做卷积。每组卷积都生成一张特征图,共生成两张特征图,此时输入及输出数据如下。

(1) 每组输入数据尺寸:$W\times H\times C/g$,分别对应的宽、高、通道数,共有 g 组(图 3-22 中 $g=2$,组卷积的组数是一个可以任意调整的超参数)。

(2) 单个卷积核尺寸:$k\times k\times C/g$,分别对应单个卷积核的宽、高、通道数,一个卷积核被分成 g 组。

(3) 输出特征图尺寸:$W'\times H'\times g$,共生成 g 个特征图。

(4) 参数量:$k\times k\times C/g\times g=k^2\times C$。

(5) 运算量:$k\times k\times C/g\times W'\times H'\times g=k^2\times C\times W'\times H'$。

对比常规卷积来看,虽然参数两者的运算量相同,但是组卷积得到了相对于常规卷积 g 倍的特征图数量,所以组卷积常用在轻量型高效网络中,因为它用少量的参数量和运算量就能生成大量的特征图,大量的特征图意味着能提取更多的信息。从分组卷积的角度来看,分组数 g 就像一个控制旋钮,最小值是 1,此时的卷积就是普通卷积;最大值是输入数据的通道数,此时的卷积就是逐通道卷积。

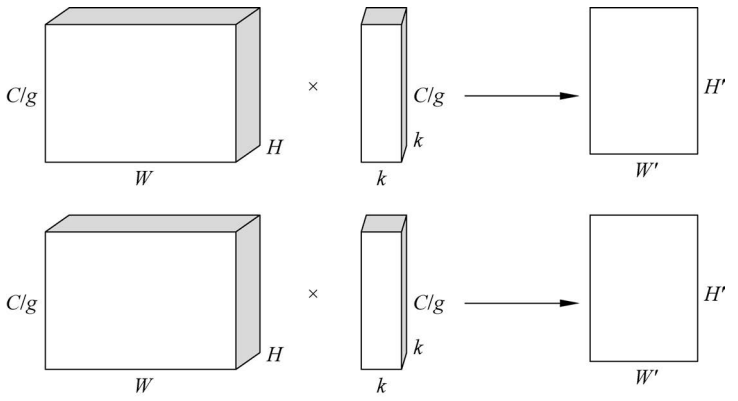


图 3-22　组卷积

逐层卷积是一种特殊形式的分组卷积，其分组数等于输入数据的通道数。这种卷积形式是最高效的卷积形式，相比普通卷积，使用同等的参数量和运算量就能够生成与输入通道数相等的特征图，而普通卷积只能生成一张特征图。逐层卷积过程如图 3-23 所示。

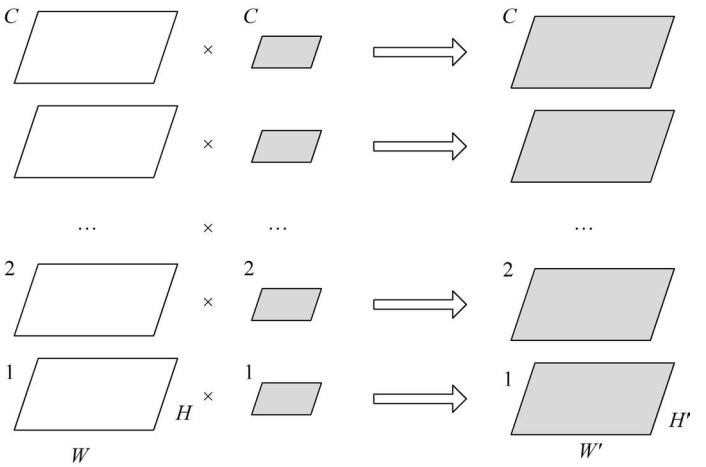


图 3-23　逐层卷积过程

由此可知深度分离卷积几乎是构造轻量高效模型的必用结构，如 Xception、MobiLeNet、MobiLeNet V2、ShuffLeNet、ShuffLeNet V2、CondenseNet 等轻量型网络结构中的必用结构。

但组卷积也存在缺点，一个显而易见的问题是在卷积过程中只有该组内的特征图进行融合，而不同组别之间缺乏计算。长此以往，不同组内的特征图对于其他组的特征了解就越来越少了，虽然网络顶层的全连接层会帮助不同特征图像互连接，但是这样的连接融合的次数较少，不如常规卷积的情况。

基于上述情况，ShuffLeNet 模型中提出了一个名为通道打散(Channel Shuffle)的解决方案。把组卷积的每个组计算得到的特征图进行一定程度的乱序排列后，再送入下一层组

卷积，以这样的方式增加特征图在不同组间的信息交互，过程如图 3-24 所示。

正常的组卷积模式如图 3-24(a)所示，不同分组几乎没有信息交流；通道打散方式如图 3-24(b)和图 3-24(c)所示。

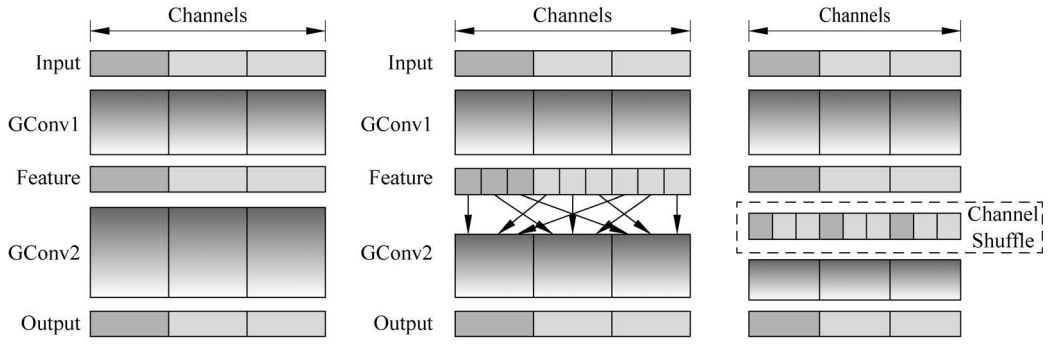


图 3-24　通道打散操作示意图

3.2.5　空间可分离卷积

空间可分离卷积(Spatially Separable Convolution)是一种用于卷积神经网络的卷积操作，其原理是将原来的二维卷积操作拆分成两个一维卷积操作，从而减少计算量和参数数量。

具体来讲，空间可分离卷积先将输入特征图在横向和纵向分别进行一维卷积操作，然后将两个卷积操作的结果按照原来的维度组合起来，得到最终的输出特征图。这个过程可以用下面的公式表示：

$$y_{i,j}=\sum_{m=1}^{M}\sum_{n=1}^{N}w_{m,n}\cdot x_{i-m,j-n} \tag{3-14}$$

其中，x 是输入特征图，y 是输出特征图，w 是卷积核，M 和 N 是卷积核的大小。式(3-14)中的卷积操作可以拆分成下面的两个一维卷积操作：

$$\begin{aligned} z_{i,j}&=\sum_{m=1}^{M}w_{m,1}\cdot x_{i-m,j}\\ y_{i,j}&=\sum_{n=1}^{N}w_{1,n}\cdot z_{i,j-n}\end{aligned} \tag{3-15}$$

这样就可以将原来的二维卷积操作拆分成两个一维卷积操作，从而大大减少了计算量和参数数量。具体来讲，如果原来的卷积核大小为 $M\times N$，则拆分成两个一维卷积操作后，参数数量从 $M\times N$ 减少到 $M+N$，计算量也大大降低了。

空间可分离卷积的好处在于可以提高卷积神经网络的计算效率和模型效果。由于空间可分离卷积的参数数量和计算量都比传统的二维卷积少，因此可以大大缩短训练时间和推

理时间。此外，由于使用空间可分离卷积可以提高计算效率，因此可以添加更多的层或者使用更大的卷积核，间接地增大了感受野，增强了模型对图像特征的提取能力。

下面对比分析一下深度可分离卷积和空间可分离卷积的区别和联系。

两者的区别主要有以下两个。

(1) 卷积核大小不同：空间可分离卷积和普通卷积核大小相同，一般是一个二维卷积核，而深度可分离卷积则是由两个不同大小的卷积核组成的，其中一个是 1×1 的卷积核(逐点卷积)，另一个是较小的二维卷积核(逐通道卷积)。

(2) 计算方式不同：空间可分离卷积先进行一次横向卷积，再进行一次纵向卷积，而深度可分离卷积则先对每个通道进行深度卷积，然后在输出通道上进行 1×1 卷积。

两者的联系也可以从以下两方面讲。

(1) 都是卷积操作：空间可分离卷积和深度可分离卷积都是卷积操作，都是对输入特征图进行局部感知并输出相应的特征图。

(2) 都是为了减少参数：空间可分离卷积和深度可分离卷积都是为了减少模型参数，提高模型效率。

总体来讲，空间可分离卷积和深度可分离卷积在某些方面有区别，但也有很多相似之处，它们都是卷积神经网络中的重要卷积操作，可以用来构建高效的深度神经网络。相比较一下，深度可分离卷积在一些应用中效果更好，也更加常见。

3.2.6　空洞卷积

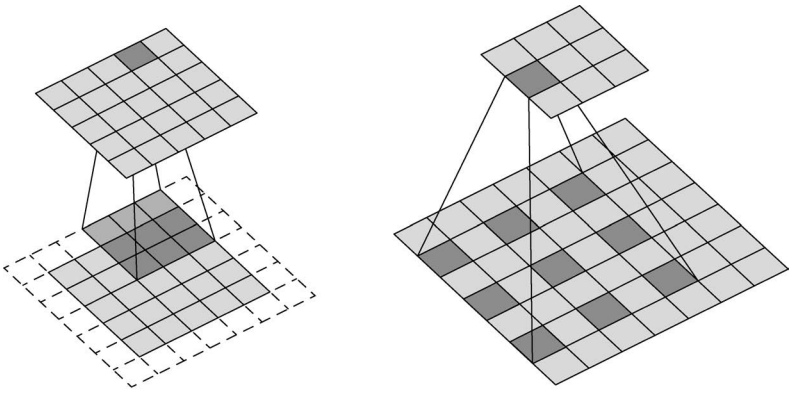
膨胀卷积(Dilated Convolution)也称作空洞卷积(Atrous Convolution)，是一种在卷积神经网络中常用的操作。膨胀卷积是在标准卷积的基础上增加了膨胀因子的概念，通过在卷积核中引入间隔性的空洞，使卷积核感受野变大，从而在保持特征图尺寸不变的情况下增加了卷积核的有效感受野，扩大了网络的感受野，同时也能有效地减少参数量。

膨胀卷积的原理：在标准卷积的基础上，膨胀卷积引入了一个膨胀系数 d，也称为膨胀因子，它表示在卷积过程中，卷积核中的元素之间相隔 $d-1$ 像素。具体来讲，对于一个 $k\times k$ 的卷积核，如果其膨胀系数为 d，则在卷积计算时，每次跳过 $d-1$ 像素进行卷积。膨胀卷积的计算方式与标准卷积类似，不同之处在于，卷积核的每个元素在计算时，需要与输入的特征图上距离自己为 d 的像素进行相乘求和，而不是与相邻的像素相乘求和，因此，膨胀卷积可以在保持特征图尺寸不变的情况下，增大卷积核的感受野，提高了网络的感受野和感知能力。

图 3-25(a)为常规的卷积操作，图 3-25(b)是膨胀系数 $d=2$ 的膨胀卷积。同时使用 3×3 大小的卷积核，膨胀卷积可以获得更大的感受野。实际上，常规的卷积操作也可以看作膨胀卷积的特殊形式，即膨胀系数 $d=1$ 的膨胀卷积。

膨胀卷积的好处主要有以下几点。

(1) 增大感受野：由于膨胀卷积中卷积核的感受野变大，所以可以更好地捕捉到输入特征图中的长程信息，提高模型对图像整体特征的感知能力。



(a) 常规卷积　　(b) 膨胀卷积

图 3-25　常规卷积与膨胀卷积

(2) 减少参数量：在保持特征图尺寸不变的情况下，膨胀卷积增加了卷积核的有效感受野，从而减少了卷积核的参数数量，降低了模型的计算复杂度和内存消耗。

(3) 节约计算资源：膨胀卷积可以通过增加膨胀系数来增大感受野，而不需要增加卷积核的大小和数量，从而节约了计算资源，使卷积神经网络能够处理大尺寸图像。

3.2.7　转置卷积

转置卷积(Transpose Convolution)也称为反卷积或上采样操作。在卷积操作中，通过滑动卷积核在输入图像上进行卷积操作来提取特征，这个过程一般是对输入数据进行下采样，而在转置卷积中，则是通过滑动反卷积核(也称为转置卷积核)在输入的特征图上进行操作，以实现图像的上采样，其计算过程如图 3-26 所示。

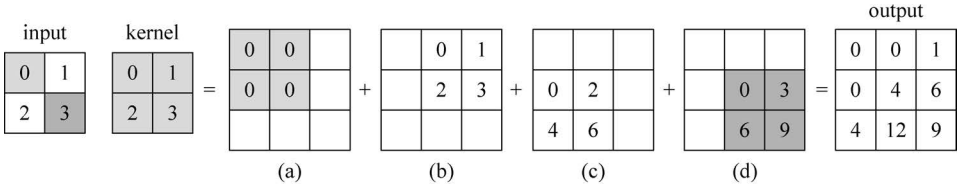


图 3-26　卷积计算

假设输入信息(input)和转置卷积核(kernel)都是一个 2×2 大小的矩阵，那么在计算过程中，input 中的第 1 个元素“0”会依次与 kernel 中的权重进行相乘，相乘的结果以如图 3-26(a)所示的方式进行存储；下一步，将转置卷积核滑动到 input 中的第 2 个元素“1”位置，依次与 kernel 中的权重进行相乘，计算结果因为滑动的原因以如图 3-26(b)所示的方式进行存储。以此类推，直到遍历完 input 中所有的元素。最后，把这 4 次相乘的中间结果图 3-26(a)、(b)、(c)、(d)在对应位置进行求和操作，便得到了最后的输出结果，其本质上与常规卷积方式相同，都是一个局部的相乘求和操作。只不过，相乘求和的顺序和方法与常规卷积存在差异。这个差异导致了转置卷积在输出特征图的尺寸上往往要大于输入信息，用于一些信息

重建的场景，而常规卷积的输出特征图尺寸往往小于输入信息，其目的是对输入信息进行特征提取。

转置卷积作为卷积神经网络中的一种操作，具有广泛的应用场景。在图像分割和目标检测领域，转置卷积可以对特征图进行上采样，使其与输入图像具有相同的大小，分别实现像素级别的分割和进行物体的定位识别；在图像重建领域，转置卷积可以将低分辨率的图像上采样到高分辨率，以便实现图像的重建和增强；在图像生成领域，转置卷积可以将随机噪声向量转换为图像，从而实现图像生成和合成。

转置卷积的主要优点包括以下几点。

(1) 上采样效果好：转置卷积可以将特征图上采样到输入图像的大小，从而实现像素级别的分割和对象检测等任务。

(2) 网络可逆性：转置卷积具有网络可逆性，可以反向传播误差，从而实现端到端的训练和优化。

(3) 可学习参数多：转置卷积具有大量的可学习参数，可以从数据中学习到有效的特征表示，从而提高模型的泛化能力。

但是，转置卷积也存在以下一些缺点。

(1) 参数量大：转置卷积具有大量的可学习参数，容易导致模型过拟合和计算资源消耗过大。这其实与第 3 条优点之间是一种权衡问题。

(2) 容易引起伪影：转置卷积的上采样效果好，但容易引起伪影和锯齿等问题，需要进行适当处理和优化。

(3) 计算量大：转置卷积需要进行矩阵乘法等复杂计算，计算量较大，需要采用一些高效的计算方法和技巧。

综上所述，转置卷积具有广泛的应用场景和优点，但也存在一些缺点和挑战。在实际应用中，需要根据具体的任务和需求，选择合适的转置卷积算法和优化方法，以提高模型的性能和效率。

3.2.8　稀疏卷积

稀疏卷积(Sparse Convolution)是一种在卷积神经网络中用于处理稀疏输入的卷积操作。相对于传统的卷积操作，稀疏卷积可以显著地减少计算量和存储需求，从而提高模型的计算效率和泛化能力。

稀疏卷积的核心思想是通过利用输入数据的稀疏性，减少卷积操作中需要计算的位置和权重。具体来讲，在传统的卷积操作中，每个卷积核都需要与输入张量中的所有位置进行卷积计算，而在稀疏卷积中，每个卷积核只需与输入张量中的一部分位置进行卷积计算，从而大大减少了计算量和存储需求。

稀疏卷积通常可以分为两种类型：位置稀疏卷积和通道稀疏卷积。位置稀疏卷积是指只对输入张量中的一部分位置进行卷积计算，而通道稀疏卷积是指只对输入张量中的一部分通道进行卷积计算。这两种稀疏卷积通常可以结合使用，从而进一步减少计算量和存储

需求。在实际应用中，通常有两种方法来选取卷积位置：固定间隔采样和自适应采样。

固定间隔采样是一种简单但常用的方法，它通过固定的采样间隔输入张量中的位置，以此来选取卷积位置。具体来讲，给定一个固定的采样间隔，固定间隔采样会从输入张量的每个维度中等间隔地选取一部分位置，并将这些位置作为卷积位置。这种方法的优点是简单、易于实现，并且可以减少计算量和存储需求。缺点是不能适应不同的输入数据分布，因此可能会影响模型的精度。

自适应采样是一种更加高级的方法，它可以根据输入数据的分布自适应地选取卷积位置。具体来讲，自适应采样通常通过一些启发式的方法来选取卷积位置，例如最大值池化、重心采样（在点云的某个局部区域内，选择与其他所有点的平均位置最接近的点）等，具体的采样策略应该根据输入张量的特点和任务需求进行选择。这些方法可以根据输入数据的分布来动态地选取卷积位置，并且能够在一定程度上提高模型的精度。缺点是计算量和存储需求较大，实现也较为复杂。

需要注意的是，稀疏卷积中的卷积位置通常是由自适应采样得到的，因此每次计算时都需要重新选择卷积位置。这也是稀疏卷积相对于传统的卷积操作在计算上更加复杂的原因之一。在实际应用中，需要权衡模型的准确性和计算效率，以便在保持一定精度的前提下提高模型的计算速度。

3.2.9 多维卷积

上述的所有卷积变体都是基于二维的，即二维卷积，主要用于处理图片数据。实际上卷积操作完全可以推广到其他维度上。下面讲解一维卷积、双向卷积、三维卷积和四维卷积。

一维卷积是卷积神经网络中常用的一种操作，主要用于处理一维序列数据，如时间序列数据、音频信号等。在一维卷积中，卷积核是一个一维的向量，可以对输入的一维序列进行卷积操作。卷积操作的过程是将卷积核从序列的左边开始移动，对每个位置进行卷积操作，得到一个新的特征值。这个过程可以用公式表示如下：

$$y_i = \sum_{j=0}^{m-1} w_j \cdot x_{i+j} \tag{3-16}$$

其中，x_{i+j} 表示输入序列在位置 $i+j$ 处的值，w_j 表示卷积核在位置 j 处的权重值，m 表示卷积核的大小。

一维卷积可以使用多个卷积核来提取不同的特征，每个卷积核都会产生一个新的特征图，多张特征图会被拼接在一起，形成一个新的特征张量。一维卷积在处理一维序列数据时具有一定的优势，因为它可以捕捉到序列中的局部关联信息，并且具有一定的平移不变性，即对于相同的序列模式，无论它们出现在序列的哪个位置，一维卷积都可以将其检测出来。

双向卷积（Bidirectional Convolution）是一种基于卷积神经网络的模型，主要用于处理时间序列数据，如音频信号、文本数据等。在传统的卷积神经网络中，输入张量的信息是从左往右进行传递的，而双向卷积则引入了从右往左的方式传递信息，从而实现对时间序列数据的双向处理。具体地，双向卷积的实现是通过对输入张量进行前向卷积和后向卷积两个

过程来完成的，其中，前向卷积是将输入张量从左到右进行卷积操作，得到一个新的张量，表示从左往右的特征信息。后向卷积则是将输入张量从右到左进行卷积操作，得到另一个新的张量，表示从右往左的特征信息。最终，这两个张量会被拼接在一起，形成一个新的特征张量，同时保留了输入张量的左右两个方向的特征信息。

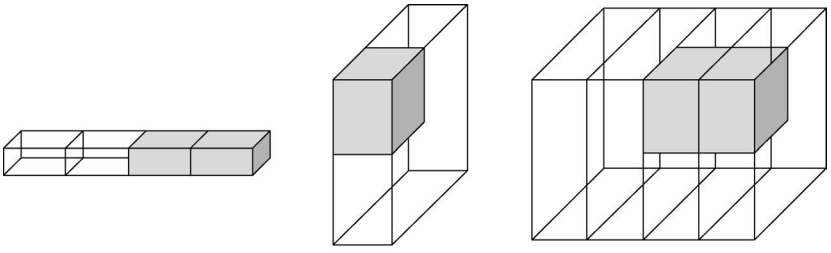
双向卷积在处理时间序列数据时具有一定的优势，因为时间序列数据往往具有左右对称性，双向卷积能够更全面地捕捉序列中的信息。同时，双向卷积也可以应用于其他领域，如在自然语言处理中的文本数据处理，可以帮助提高模型的性能和泛化能力。可以想象，当双向卷积配合一维卷积时，可以很好地处理文本、信号等带有时序特点的数据。

三维卷积是卷积神经网络中常用的一种操作，主要用于处理三维数据，如视频、医学图像等。在三维卷积中，卷积核是一个三维的张量，可以对输入的三维数据进行卷积操作。卷积操作的过程是将卷积核从输入数据的左上角开始移动，对每个位置进行卷积操作，得到一个新的特征值。这个过程可以用公式表示如下：

$$y_{ijk}=\sum_{m=-a}^{a}\sum_{n=-b}^{b}\sum_{p=-c}^{c}w_{mnp}\cdot x_{i+m,j+n,k+p} \tag{3-17}$$

其中，$x_{i+m,j+n,k+p}$ 表示输入数据在位置$(i+m,j+n,k+p)$处的值，w_{mnp} 表示卷积核在位置(m,n,p)处的权重值，a,b,c 表示卷积核在 3 个维度上的大小。

三维卷积可以使用多个卷积核来提取不同的特征，每个卷积核都会产生一个新的特征图，多张特征图会被拼接在一起，形成一个新的特征张量。三维卷积在处理三维数据时具有一定的优势，因为它可以捕捉到数据中的空间信息，并且具有一定的平移不变性，即对于相同的物体形状或纹理，无论它们出现在图像的哪个位置，三维卷积都可以将其检测出来。同时，三维卷积也可以通过不同大小的卷积核来检测不同尺度的特征，从而提高模型的性能和泛化能力。一维卷积的卷积核示例如图 3-27(a)所示，二维卷积的卷积核示例如图 3-27(b)所示，三维卷积的卷积核示例如图 3-27(c)所示。



(a) 一维卷积　(b) 二维卷积　(c) 三维卷积

图 3-27　卷积核示例

如果继续类比下去，则其实还存在四维卷积的概念。四维卷积也是卷积神经网络中常用的一种操作，它主要用于处理四维数据，如视频序列或多个图像的批量数据。这时的数据维度可表示为[图片数量，颜色通道，图片高度，图片宽度]。在四维卷积中，卷积核是一个四维的张量，可以对输入的四维数据进行卷积操作。

卷积操作的过程是将卷积核从输入数据的左上角开始移动，对每个位置进行卷积操作，得到一个新的特征值。这个过程可以用公式表示如下：

$$y_{ijkl}=\sum_{m=-a}^{a}\sum_{n=-b}^{b}\sum_{p=-c}^{c}\sum_{q=-d}^{d}w_{mnpq}\cdot x_{i+m,j+n,k+p,l+q} \tag{3-18}$$

其中，$x_{i+m,j+n,k+p,l+q}$ 表示输入数据在位置$(i+m,j+n,k+p,l+p)$处的值，w_{mnpq} 表示卷积核在位置(m,n,p,q)处的权重值，a,b,c,d 表示卷积核在 4 个维度上的大小。

类似于三维卷积，四维卷积也可以使用多个卷积核来提取不同的特征，每个卷积核都会产生一个新的特征图，多张特征图会被拼接在一起，形成一个新的特征张量。

需要注意的是，在实际应用中，通常会将四维卷积分解成两个步骤，先将输入的四维数据转换成两个三维数据，然后分别对它们进行三维卷积操作，最后将两个输出结果合并起来。这种方式可以减少计算量，提高模型的效率。