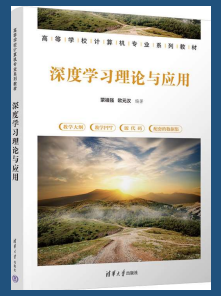
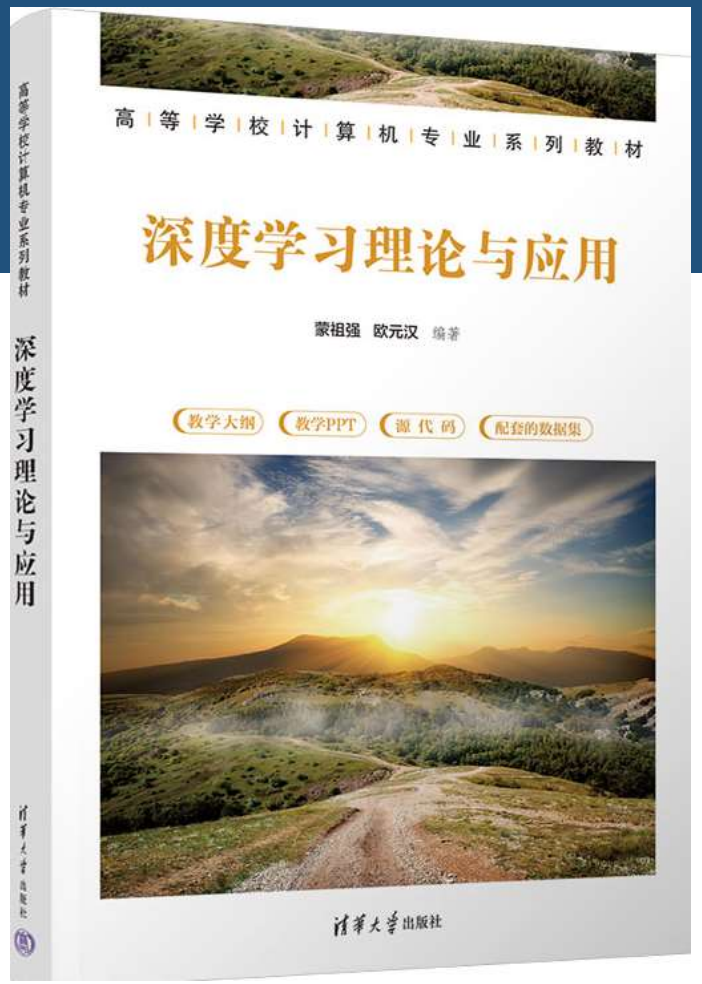


深度学习理论与应用

Deep Learning Theory and Applications

蒙祖强，欧元汉 编著



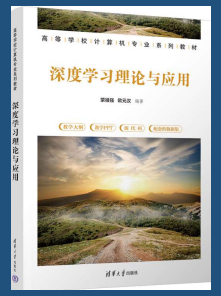
教　材

全国各大书店网店均有销售

- **教学大纲**：提供面向教育工程认证的教学大纲
- **教学PPT**：提供课堂教学用的PPT课件
- **源代码**：提供教材涉及的全部源代码
- **数据集**：提供教材示例、案例用到的全部数据集

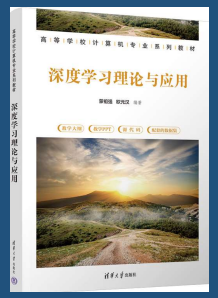
获取教学资源：
http://www.tup.tsinghua.edu.cn/booksCenter/book_09988101.html

教材： 蒙祖强，欧元汉 编著. 深度学习理论与应用. 北京: 清华大学出版社，2023年7月. (书号: 978-7-302-63508-6)



第4章　卷积神经网络

蒙祖强，欧元汉 编著. 深度学习理论与应用. 北京: 清华大学出版社，2023年7月.



本章内容

contents

4.1 一个简单的卷积神经网络——手写数字识别

4.2 卷积神经网络的主要操作

4.3 卷积神经网络的设计方法

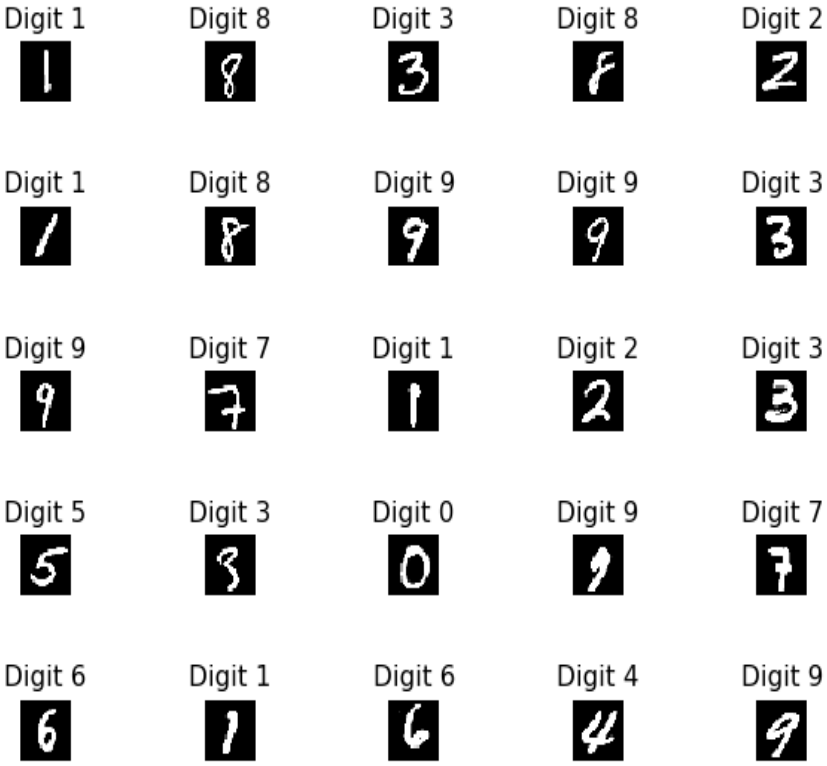
4.4 过拟合及其解决方法

4.1 一个简单的卷积神经网络——手写数字识别

4.1.1 程序代码

【例4.1】创建一个卷积神经网络，使之能够识别手写数字图片。

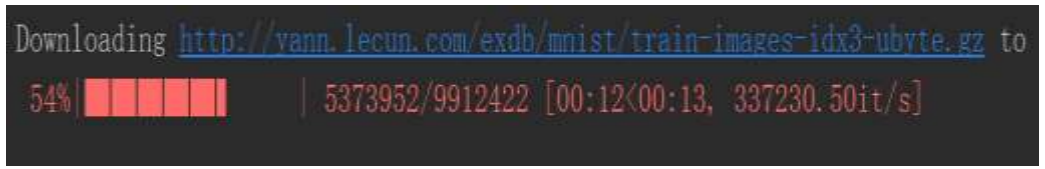
- **手写数字图片数据集**（MNIST数据集）：来自MNIST官网站（http://yann.lecun.com/exdb/mnist/）
- **MNIST数据集**：已经分为训练集和测试集，分别是6万和1万张图片，也可以从利用DataLoader()函数自动下载
- **数据集示例**：下图是MNIST数据集中25个手写数字图片样例，图片大小为28×28：



4.1 一个简单的卷积神经网络——手写数字识别

4.1.1 程序代码

第一次运行该程序时，会产生如下图所示的界面，这表示DataLoader()函数在下载数据集。只要网络顺畅，过一会就会下载完毕，数据集保存在./data/MNIST目录下，其中目录./data是在代码中指定的，子目录MNIST是自动生成的。



DataLoader()函数下载数据集的进度

```
#定义卷积神经网络
class Example4_1(nn.Module):
    def __init__(self):
        super().__init__()
        #下面创建两个卷积层：
            self.conv1 = nn.Conv2d(1, 10, 5)        #第一个卷积层
            self.conv2 = nn.Conv2d(10, 20, 3)       #第二个卷积层
        #下面创建两个全连接层：
        self.fc1 = nn.Linear(2000, 500)     #第一个全连接层
        self.fc2 = nn.Linear(500, 10)       #第二个全连接层
    def forward(self,x):            #x 为输入进来的数据包，其形状为 batch×1×28×28
        out = self.conv1(x)         #将 x 输入第一个卷积层
        out = torch.relu(out)                       #运用激活函数 relu
        out = torch.max_pool2d(out, 2, 2)           #将 x 输入池化层
        out = self.conv2(out)                       #将 x 输入第二个卷积层
        out = torch.relu(out)                       #运用激活函数 relu
        out = out.view(x.size(0), -1)               #对 out 扁平化后，以送入全连接层
        out = self.fc1(out)                         #将 x 输入第一个全连接层
        out = torch.relu(out)
        out = self.fc2(out)                         #将 x 输入第二个全连接层
        return out
```

该程序的核心代码（全部代码见教材P92）

4.1 一个简单的卷积神经网络——手写数字识别

4.1.1 程序代码

此后，程序进入训练阶段。下图展示了程序训练过程中输出的进度信息。

```
训练的代数：7 [当前代完成进度：46080 / 60000 (76%)]，当前损失函数值：0.013703
训练的代数：8 [当前代完成进度：0 / 60000 (0%)]，当前损失函数值：0.014499
训练的代数：8 [当前代完成进度：15360 / 60000 (25%)]，当前损失函数值：0.004509
训练的代数：8 [当前代完成进度：30720 / 60000 (51%)]，当前损失函数值：0.016566
训练的代数：8 [当前代完成进度：46080 / 60000 (76%)]，当前损失函数值：0.016867
训练的代数：9 [当前代完成进度：0 / 60000 (0%)]，当前损失函数值：0.011448
训练的代数：9 [当前代完成进度：15360 / 60000 (25%)]，当前损失函数值：0.024032
训练的代数：9 [当前代完成进度：30720 / 60000 (51%)]，当前损失函数值：0.028095
训练的代数：9 [当前代完成进度：46080 / 60000 (76%)]，当前损失函数值：0.016648
```

4.1 一个简单的卷积神经网络——手写数字识别

4.1.1 程序代码

程序运行完毕后，在笔者计算机上输出如下结果：

```
运行时间：151.7 秒
在测试集上的预测准确率：99.2%
```

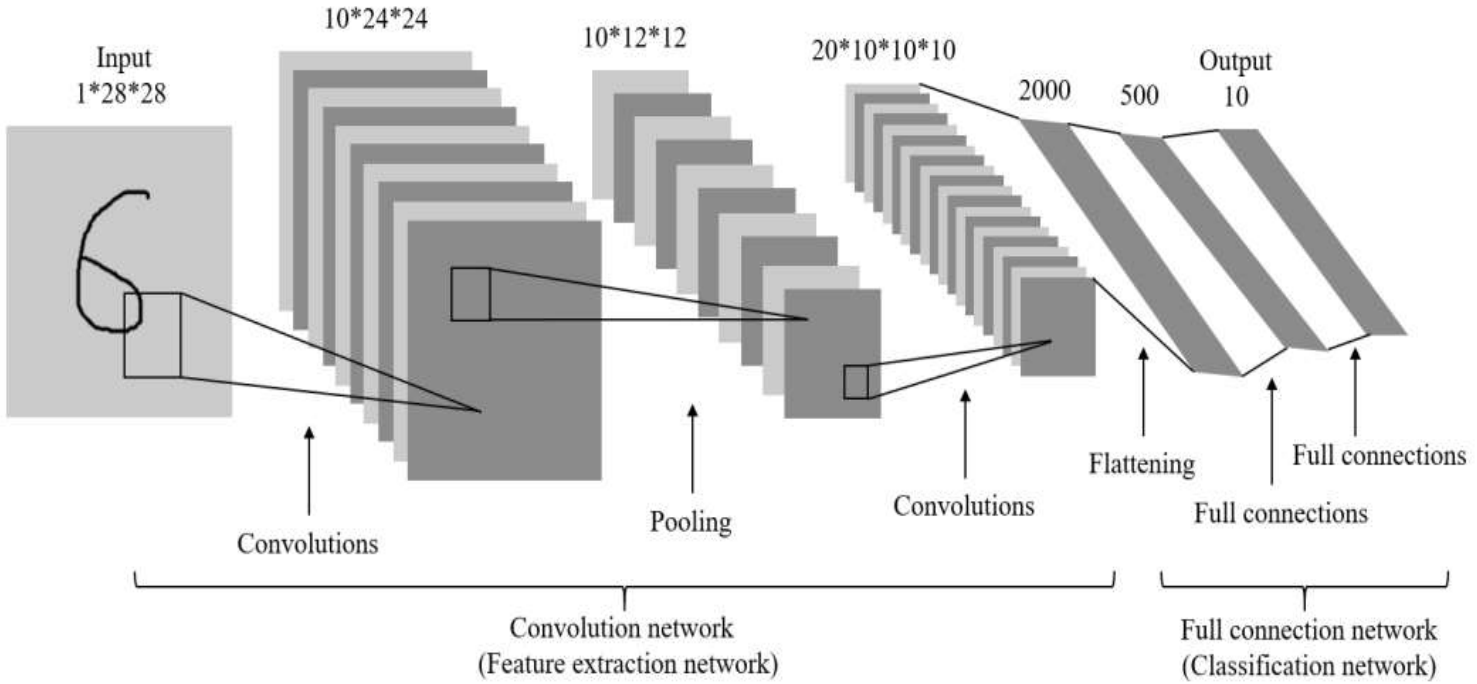
笔者计算机带有GPU卡，运行时间为149.7秒。如果在CPU上运行，运行时间为375.1秒，是前者的2.5倍。这说明，GPU可以加倍提速网络程序的运行速度。

上面程序是一个完整的程序，包括训练过程和测试过程。只要网络畅通（第一次运行时需要下载数据集，故需要网络，此后再运行就不再需要网络了），不用做任何“数据安装”即可运行，并输出相应的测试结果。

4.1 一个简单的卷积神经网络——手写数字识别

4.1.2 代码解释

网络结构：一部分是由两个卷积层和一个池化层构成的**卷积网络**，另一部分是由两个网络层构成的**全连接网络**：



4.1 一个简单的卷积神经网络——手写数字识别

4.1.2 代码解释

- 卷积网络：主要用于提取图像的特征，故也称为**特征提取（学习）网络**；
- 全连接网络：则根据提取的特征对样本进行分类，也常称**分类网络**。

上图所示的整个神经网络实际上是由一个卷积神经网络和一个全连接网络组成。为了区别，本书一般称之为**深度神经网络**。也有的书直接称为卷积神经网络，而弱化了其全连接神经网络部分，这主要是为了突出卷积神经网络的地位。

4.1 一个简单的卷积神经网络——手写数字识别

4.1.2 代码解释

卷积网络由两个卷积层和一个池化层组成，定义代码如下：

```
self.conv1 = nn.Conv2d(1, 10, 5)
self.conv2 = nn.Conv2d(10, 20, 3)
```

- 在第一条语句中第一个参数表示输入图像的通道数为1，第二个参数表示用了10个卷积核，因而输出通道数也为10（每个卷积核会产生一个输出通道），第三个参数则表示卷积核的大小为5×5；

- 在第二条语句中，输入通道数为10，卷积核数量为20，因而输出通道数也为20，卷积核的大小3×3。

4.1 一个简单的卷积神经网络——手写数字识别

4.1.2 代码解释

全连接层的定义代码如下:

```
self.fc1 = nn.Linear(2000, 500)
self.fc2 = nn.Linear(500, 10)
```

这两条语句共同表示建立一个输入节点数为2000、隐含层节点数为500、输出节点数为10的全连接网络。

4.1 一个简单的卷积神经网络——手写数字识别

4.1.2 代码解释

输入卷积网络的是28×28的单通道手写数字图像，这些图像都经过下列操作：

（1）输入第一个卷积层，产生一个10×24×24的特征图，实现代码如下：

```
out = self.conv1(x)   #batch×1×28×28 → batch×10×24×24
```

注：特征图（feature map）是一个重要的概念，它是由一个或多个同等尺寸的二维数值矩阵构成的数据立方体，其中每个二维数值矩阵称为特征图的**通道**，有时也形象称为**通道图像**。本质上，一个特征图是一个三维张量（如果不考虑批量大小的话），其形状为(channel, height, width)，其中channel表示特征图通道的数量，也称为特征图的**深度**，height和width分别是特征图的高和宽（实际上是其通道图像的高和宽）。

在卷积网络中，每一个网络层的输入和输出都是特征图，但它们的尺寸不一样。例如，对self.conv1网络层而言，输入的图像是形状为1×28×28的特征图（不考虑批量大小在内），输出的特征图的形状为10×24×24。

4.1 一个简单的卷积神经网络——手写数字识别

4.1.2 代码解释

（2）运用relu激活函数，将小于0的元素值变为0，大于或等于0的值不变，同时不改变out的形状：

```
out = torch.relu(out)      # batch×10×24×24 → batch×10×24×24 （形状不变）
```

（3）将上述特征图输入池化层，产生形状为10×12×12的特征图（高和宽减半），代码如下：

```
out = torch.max_pool2d(out, 2, 2)    # batch×10×24×24 → batch×10×12×12
```

（4）再输入第二个卷积层，该卷积层运用3×3卷积核，产生20×10×10的特征图：

```
out = self.conv2(out)                    # batch×10×12×12 → batch×20×10×10
```

4.1 一个简单的卷积神经网络——手写数字识别

4.1.2 代码解释

（5）再次运用relu激活函数，然后对输出的特征图进行扁平化操作，产生长度为2000的向量，形成2000个节点；

```
out = torch.relu(out)                #形状不变
out = out.view(x.size(0), -1)        #batch×20×10×10 → batch×2000
```

注意，x.size(0)不能替换为batch_size，因为最后一个数据包的规模可能小于既定的包的规模batch_size。

4.1 一个简单的卷积神经网络——手写数字识别

4.1.2 代码解释

（6）最后输入全连接网络进行分类：

```
out = self.fc1(out)          #batch×2000 → batch×500
out = torch.relu(out)            #运用relu激活函数
out = self.fc2(out)          #batch×500 -> batch×10
```

全连接网络的输出out是各样本在10个类别上的概率分布。注意，上述“batch”是指一个数据包中样本的数量。

说明：此后的代码是用于实现网络的训练和测试，这跟之前的例子是一样的，因此不再做具体分析。

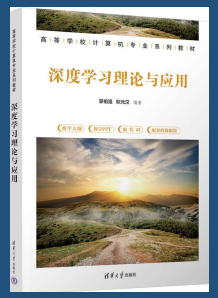
4.1 一个简单的卷积神经网络——手写数字识别

4.1.2 代码解释

本程序构建的卷积神经网络并不复杂，只包含了新增的两类操作：卷积操作和池化操作，具体的操作系列如下：

卷积→relu激活函数→池化→卷积→relu激活函数

这些操作的基本原理和设计方法将在下一节介绍。



本 章 内 容

contents

4.2 卷积神经网络的主要操作

4.2.1 单通道卷积

卷积（Convolution）可以说是卷积神经网络中核心的操作，其主要作用是提取图像的局部特征，如边沿特征、纹理特征以及高层语义特征等。下面先介绍单通道卷积的基本原理。

4.2 卷积神经网络的主要操作

4.2.1 单通道卷积

考虑如下图所示的两个数值矩阵X和K。

-1	-1	-1
1	-1	1
1	0	-1

（2）矩阵 $\boldsymbol{K}$

-3	2	1	-2	-1
2	1	0	-2	1
-3	-3	-1	-1	-3
-1	-2	-1	-2	-3
0	2	2	-3	1

（1）矩阵 $\boldsymbol{X}$

从矩阵$\boldsymbol{X}$的左上角元素开始，将矩阵$\boldsymbol{K}$“扣到”矩阵$\boldsymbol{X}$上去，将二者“重叠”的元素相乘

$-3_{\times -1}$	$2_{\times -1}$	$1_{\times -1}$
$2_{\times 1}$	$1_{\times -1}$	$0_{\times 1}$
$-3_{\times 1}$	$-3_{\times 0}$	$-1_{\times -1}$

二矩阵对应元素相乘

4.2 卷积神经网络的主要操作

4.2.1 单通道卷积

（1）上图中，对乘积结果求和：[(-3)×(-1)+2×(-1)+1×(-1)]+[2×1+1×(-1)+0×1]+[(-3)×1+(-3)×0+(-1)×(-1)]=-1，最后将得到的结果-1放到另一个新矩阵（称为矩阵***Y***）中，作为矩阵***Y***的第一行上的第一个元素（-1）。

（2）接着，将矩阵***K***向右平移一列，重复上述计算方法，得到第二个数值-4，并将之作为矩阵***Y***中第一行上的第二个元素（-4）。然后，再将矩阵***K***向右平移一列，用类似方法得到矩阵***Y***中第一行上的第三个元素（7）。

（3）之后，就不能再向右平移了，因为矩阵***K***在矩阵***X***中已经移到最右边了。这时，将矩阵***K***退回到矩阵***X***的最左边（使得矩阵***X***和矩阵***K***左对齐），并向下移一行，然后用上述同样方法，计算得到矩阵***Y***第二行上的元素，它们的值分别为-4, -2和0。最后，将矩阵***K***再往下移一行，用同样方法，得到矩阵***Y***第三行上的元素，它们是5, 7和4。

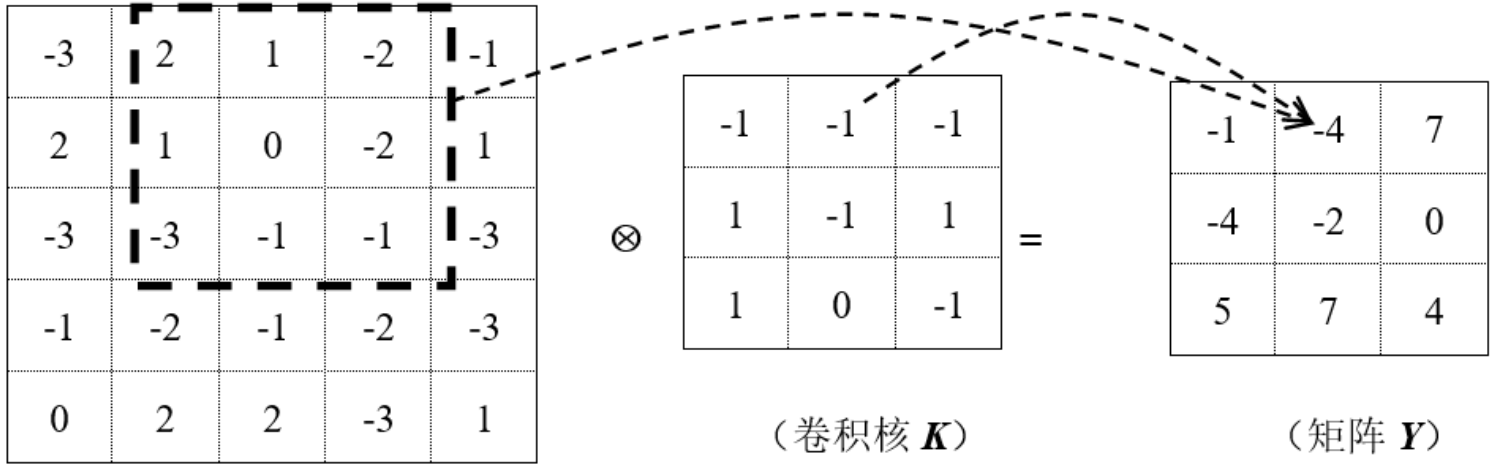
-1	-4	7
-4	-2	0
5	7	4

形成的矩阵 ***Y***

4.2 卷积神经网络的主要操作

4.2.1 单通道卷积

矩阵***X***可以视为输入的单通道图像或特征图，矩阵***K***则充当了**卷积核**（也称为**过滤器**）的作用，而矩阵***Y***则是形成的一个中间结果——特征图:



（矩阵 X）　　（卷积核 K）　　（矩阵 Y）

卷积过程示意图

4.2 卷积神经网络的主要操作

4.2.1 单通道卷积

卷积公式：令$x_{i,j}$, $w_{i,j}$, $y_{i,j}$分别表示矩阵$\boldsymbol{X}$、$\boldsymbol{K}$和$\boldsymbol{Y}$中的第i+1行第j+1列处的元素（矩阵元素从0开始编号），则矩阵$\boldsymbol{Y}$中的元素可用矩阵$\boldsymbol{X}$和$\boldsymbol{K}$中相关元素来表示：

$$\boldsymbol{y_{i,j}} = \sigma\left(\sum_{s=0}^{3-1}\sum_{t=0}^{3-1} \boldsymbol{w_{s,t}x_{i+s,j+t}} + \boldsymbol{k_b}\right)$$

其中，k_b为偏置项。

-1	-4	7
-4	-2	0
5	7	4

（矩阵 $\boldsymbol{Y}$）

例如：

$$y_{0,1} = \sum_{s=0}^{2}\sum_{t=0}^{2} w_{s,t}x_{s,1+t}$$
$$= [w_{0,0}x_{0,1}+w_{0,1}x_{0,2}+w_{0,2}x_{0,3}]+[w_{1,0}x_{1,1}+w_{1,1}x_{1,2}+w_{1,2}x_{1,3}]+[w_{2,0}x_{2,1}+w_{2,1}x_{2,2}+w_{2,2}x_{2,3}]$$
$$= [(-1)\times2+(-1)\times1+(-1)\times(-2)]+[1\times1+(-1)\times0+1\times(-2)]+[1\times(-3)+0\times(-1)+(-1)\times(-1)]$$
$$= -4$$

（此处假设偏置项k_b为0）

4.2 卷积神经网络的主要操作

4.2.1 单通道卷积

卷积公式也可以写成向量相乘的形式：

也可以写成向量形式的卷积公式：

$$\boldsymbol{Y} = \sigma(\boldsymbol{K} \otimes \boldsymbol{X} + \boldsymbol{K}_b)$$

其中，符号“⊗”表示**卷积操作**，$\boldsymbol{K}_b$是一个矩阵，其中每个元素均等于偏置项值，其规模跟$\boldsymbol{K} \otimes \boldsymbol{X}$的规模一样，$\sigma$为激活函数，一般用relu激活函数。

4.2 卷积神经网络的主要操作

4.2.1 单通道卷积

输出特征图的尺寸计算公式：假设假设输入特征图$\boldsymbol{X}$为$W_1 \times H_1$矩阵（即W_1行W_2列矩阵），产生的输出特征图$\boldsymbol{Y}$为$W_2 \times H_2$矩阵，且假设每次移动S行或S列时（S为大于或等于1的整数），那么$\boldsymbol{Y}$的规模（尺寸）的计算公式：

$$\boldsymbol{W_2 = \frac{W_1 - F}{S} + 1}$$
$$\boldsymbol{H_2 = \frac{H_1 - F}{S} + 1}$$

其中，S称为滑动的**步长**。

例如，对于上述例子中，由于$\boldsymbol{X}$和$\boldsymbol{K}$分别为5×5矩阵和3×3矩阵，每次滑动一行或一列，即S=1，所以$\boldsymbol{Y}$的行数和列数均为5−3+1=3。

4.2 卷积神经网络的主要操作

4.2.1 单通道卷积

有时候出于某些目的，需要在矩阵$\boldsymbol{X}$的外围填上几圈0，这种操作称为**填充**（Padding），所填充0的圈数称为**填充数**。假设在矩阵$\boldsymbol{X}$的外围填上P圈0，则矩阵$\boldsymbol{X}$由原来的W_1行和H_1列变为W_1+2P行和H_1+2P列。据此不难推出，当填上P圈0时，上述关于$\boldsymbol{Y}$的规模的计算公式变为：

$$\boldsymbol{W_2 = \frac{W_1 - F + 2P}{S} + 1}$$

$$\boldsymbol{H_2 = \frac{H_1 - F + 2P}{S} + 1}$$

4.2 卷积神经网络的主要操作

4.2.1 单通道卷积

例如，对于上例子，当在***X***的外围填上1圈0后（见左图），再用***K***对其进行卷积，则产生的***Y***的行数和列数均为(5-3+2×1)/1+1=5 (P = 1)。

从这个例子也可以看出，在卷积操作中，如果设置步长*S*为1，卷积核大小*F*为3，填充圈数*P*为1，则卷积后***X***和***Y***的规模完全一样，即形状不变。**这个性质在卷积神经网络中我们经常会运用到**。

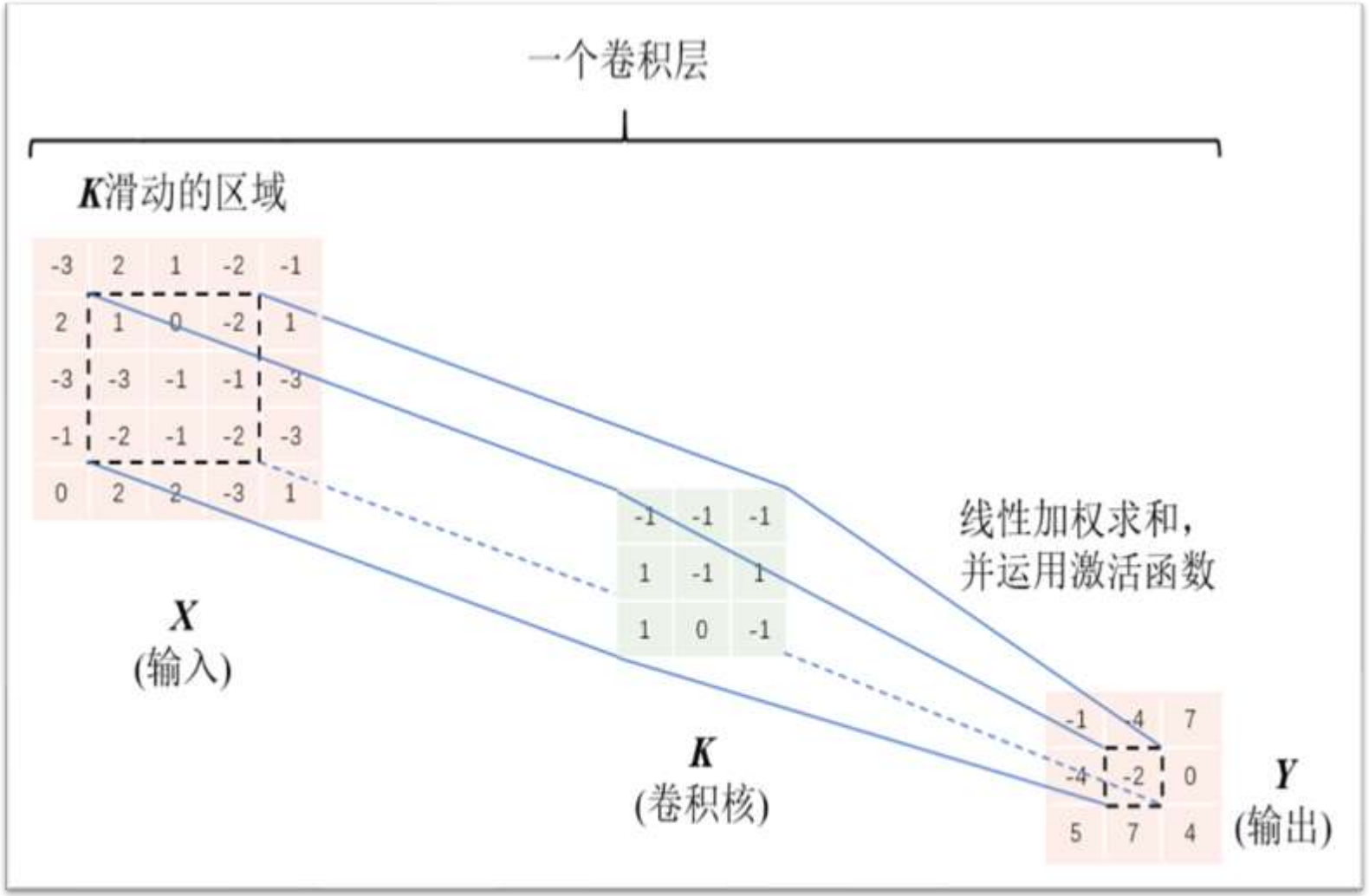
0	**0**	**0**	**0**	**0**	**0**	**0**
0	-3	2	1	-2	-1	**0**
0	2	1	0	-2	1	**0**
0	-3	-3	-1	-1	-3	**0**
0	-1	-2	-1	-2	-3	**0**
0	0	2	2	-3	1	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**

在 *X* 的外围填充一圈 0 后的效果

4.2 卷积神经网络的主要操作

4.2.1 单通道卷积

单个卷积层的原理示意图：一个卷积层实际上就是一个在卷积操作作用下从输入到输出的一个计算单元，这其中的主要操作包括从左到右从上到下的滑动、线性加权（包括与偏置项之和）和激活函数运算等。该原理可以用左图来表示：



4.2 卷积神经网络的主要操作

4.2.1 单通道卷积

单通道卷积中参数个数的计算方法：卷积层的参数包括卷积核*K*中的参数以及一个偏置项k_b，参数个数为*F*×*F*+1，其中*F*为卷积核的行数和列数。而且，参数的个数完全由设定的卷积核来确定，跟输入节点数和输出节点数无关。

例如，对于上图所示的卷积层，其卷积核为3 × 3矩阵，因此它一共包含3×3+1=10个参数。

4.2 卷积神经网络的主要操作

4.2.1 单通道卷积

卷积网络的优点：

- 在全连接网络中，在任意两个输入节点和输出节点之间都有一条边相连（一条边对应一个参数）。假如输入节点为1百万个（对图像这类输入而言，这并不算多），且要求有2百个输出节点，则在输入节点和输出节点之间有1亿个参数。显然，参数总数是两边节点数的乘积，跟两边的节点成正比。

- 如果在两边节点之间不采用全连接方式，而改用一个3×3的卷积核$\boldsymbol{K}$来“连接”，那么通过卷积核的滑动，使得输入节点都可以共享$\boldsymbol{K}$中的权值（边），所使用的参数个数永远为3×3+1=9，而与输入和输出节点的数量无关——这就是卷积网络的优势之一，即，**参数共享**是卷积网络的重要优点之一。

- 一个输出节点是由卷积核“连接”的$F\times F$个局部输入节点来产生，这就是所谓的**局部连接**。而在全连接网络中，一个输出节点的形成是利用所有的输入节点来计算而得到的。实践表明，局部连接是非常有效的，而且也极大地减少了计算量。局部连接是卷积神经网络的另一个重要优点。

4.2 卷积神经网络的主要操作

4.2.2 多通道卷积

对含有多个通道的输入（特征图）进行卷积的操作称为**多通道卷积**。例如，彩色图像有三个通道，对其卷积就属于多通道卷积；如果上一网络层输出的特征图包含了多个通道，则该特征图对于当前卷积层而言也是多通道输入，需要进行多通道卷积。

假设输入$\boldsymbol{X}$包含了d个通道：$\boldsymbol{X}^{(0)}, \boldsymbol{X}^{(1)}, \ldots, \boldsymbol{X}^{(d-1)}$，其中每个$\boldsymbol{X}^{(j)}$都是一个同等规模的数值矩阵，$j = 0,1,\ldots,d-1$。

对于给定的d通道输入特征图$\boldsymbol{X}$，在卷积层中必须设置一个也包含d个数值矩阵的卷积核$\boldsymbol{K}$，这d个数值矩阵也就是d个通道，分别是：$\boldsymbol{K}^{(0)}, \boldsymbol{K}^{(1)}, \ldots, \boldsymbol{K}^{(d-1)}$，其规模$F$是超参数，需要事先设置。我们约定：称$\boldsymbol{K}$为$F$阶卷积核，意指$\boldsymbol{K}$包含的数值矩阵（通道）都是$F$阶矩阵。

4.2 卷积神经网络的主要操作

4.2.2 多通道卷积

对输入特征图X的卷积操作过程如下：

（1）首先，在卷积时 $K^{(0)}, K^{(1)}, \ldots, K^{(d-1)}$分别对$X^{(0)}, X^{(1)}, \ldots, X^{(d-1)}$进行按位同步卷积。也就是说，对所有$j \in [0,1,\ldots,d-1]$，同步执行下列卷积操作（同步$i$和$j$）：

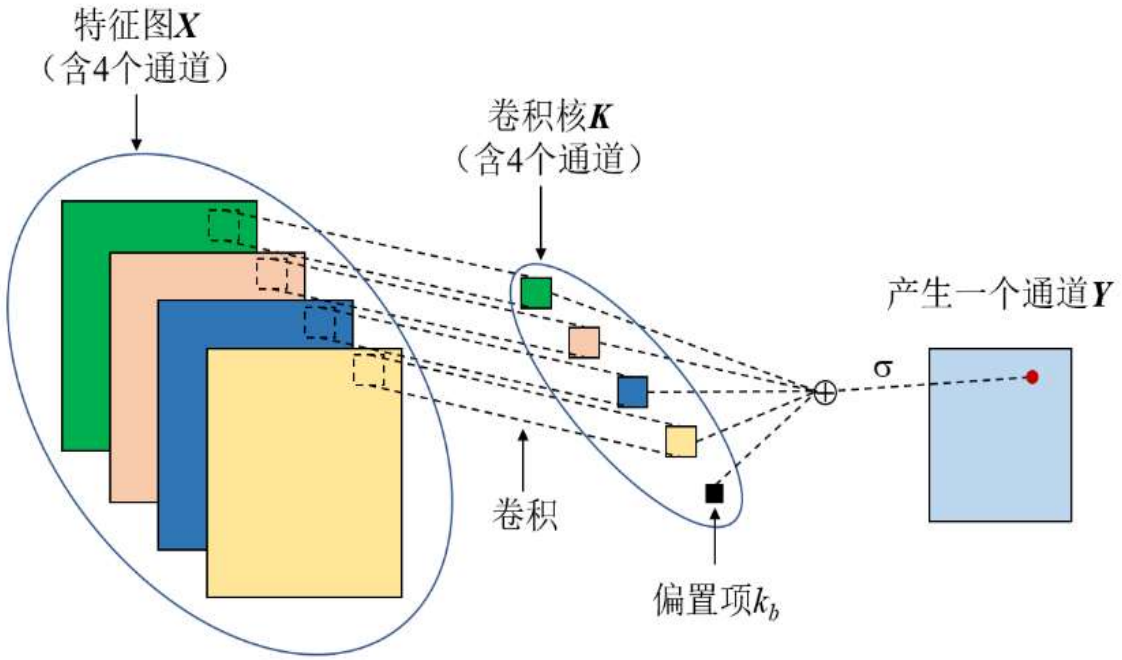
$$y_{i,j}^{(j)} = \sum_{s=0}^{F-1} \sum_{t=0}^{F-1} w_{s,t}^{(j)} x_{i+s,j+t}^{(j)}$$

（2）然后，令$y_{i,j} = \sigma(\sum_{j=0}^{d-1} y_{i,j}^{(j)} + k_b)$。这样，$y_{i,j}$所构成的矩阵$Y$便是深度为$d$的卷积核$K$对$d$通道特征图$X$进行卷积的结果。相对$X$而言，矩阵$Y$是一个输出。由于$Y$只是一个数值矩阵，因而它也称为**单通道输出**。也就是说，**一个卷积核只能产生一个通道**。

4.2 卷积神经网络的主要操作

4.2.2 多通道卷积

输入特征图和卷积核的这种卷积操作可用下图简要表示：



注意：

（1）1个卷积核产生1个通道，n个卷积核才能产生含n个通道的特征图；

（2）卷积核$\boldsymbol{K}$的深度与输入特征图$\boldsymbol{X}$的通道数永远是相等的；

（3）每个卷积核可带有一个偏置项（也可以不设置偏置项），但不能带多个偏置项。

说明：由于卷积核$\boldsymbol{K}$的深度与输入特征图$\boldsymbol{X}$的通道数永远是相等的，所以在提及卷积核时，通常会省略其深度（实际上是默认其深度与通道数相等），只说卷积核的高和宽。例如，我们说“3×3卷积核”或“3阶卷积核”，意指该卷积核包含的通道都是3×3的矩阵，矩阵的数量（即卷积核的深度或通道数）默认等于特征图$\boldsymbol{X}$的通道数。

4.2 卷积神经网络的主要操作

4.2.2 多通道卷积

【例4.2】实现对3通道特征图的卷积计算。假设给定含3个通道的特征图$\boldsymbol{X}$，如图4-12所示，同时构造了2个卷积核$\boldsymbol{K}_1$和$\boldsymbol{K}_2$，它们的偏置项分别为1和3，分别如图4-13和图4-14所示。现要求利用这两个卷积核对特征图$\boldsymbol{X}$进行卷积，并给出关键卷积操作过程及卷积操作结果，其中步长为2。

-3	2	1	-2	-1
2	1	0	-2	1
-3	-3	-1	-1	-3
-1	-2	-1	-2	-3
0	2	2	-3	1

（a）通道 $X^{(0)}$

0	0	0	2	0
2	-1	1	0	-1
2	1	-1	2	-1
-1	-1	0	2	-3
1	-3	-3	-1	-1

（b）通道 $X^{(1)}$

-3	-2	-2	1	-3
-3	1	0	-2	0
0	2	-3	-1	-2
-2	-1	2	0	1
-3	0	2	-1	2

（c）通道 $X^{(2)}$

图 4-12　特征图 X 的 3 通道

1	0	1
1	1	0
0	1	1

（a）通道 $K^{(0)}$

-1	-1	1
-1	-1	0
1	0	-1

（b）通道 $K^{(1)}$

0	0	0
0	1	-1
0	-1	0

（c）通道 $K^{(2)}$

图 4-13　卷积核 K_1 的 3 通道（偏置项为 1）

1	1	1
1	1	-1
-1	-1	1

（a）通道 $K^{(0)}$

0	0	-1
0	-1	1
1	-1	1

（b）通道 $K^{(1)}$

0	0	1
-1	-1	-1
-1	-1	-1

（c）通道 $K^{(2)}$

图 4-14　卷积核 K_2 的 3 通道（偏置项为 3）

4.2 卷积神经网络的主要操作

4.2.2 多通道卷积

先考虑卷积核K_1按位同步对X进行卷积的过程，并假设在用K_1对X卷积后生成通道Y_1，即生成的一个输出通道。显然，按照前面关于规模的计算公式，Y_1的行数和列数均为：

$$\frac{5-3+2\times0}{2}+1=2$$

即Y_1是2×2矩阵。

进一步考虑Y_1中元素$y_{0,1}$的同步计算过程。如图4-15所示，将K_1的三个通道$K^{(0)}$、$K^{(1)}$、$K^{(2)}$分别跟X的三个通道$X^{(0)}$、$X^{(1)}$、$X^{(2)}$的右上角区域按位同步相乘，然后得到三个3×3矩阵，再将这三个矩阵中的元素分别求和，得到-6、4和-2个数值，接着将这三个数值累加起来，同时加上K_1的偏置项1，结果得到-3。如果同时还考虑激活函数σ，并假定激活函数σ为relu函数，则在运用该激活函数后，-3变为0。这个0就是$y_{0,1}$的值，即$y_{0,1}$=0。计算$y_{0,1}$的整个过程如图4-15所示。

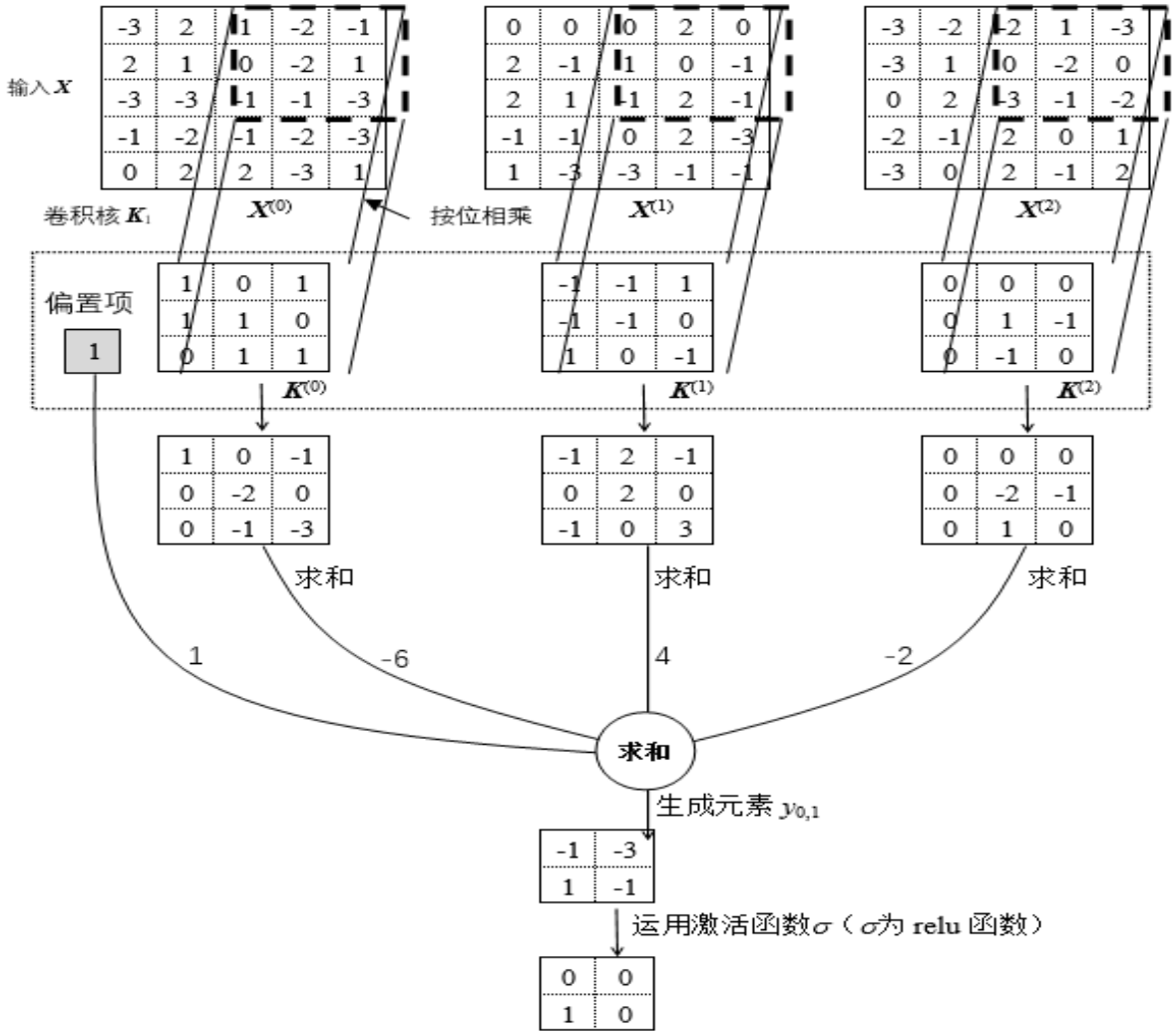


图 4-15　Y_1 中元素 $y_{0,1}$ 的同步计算过程

4.2 卷积神经网络的主要操作

4.2.2 多通道卷积

当以$\boldsymbol{K}_2$为卷积核时，卷积后得到的矩阵$\boldsymbol{Y}_2$的元素如下：

13	3
0	11

这两个输出通道$\boldsymbol{Y}_1$和$\boldsymbol{Y}_2$的计算过程可表示为：

$$\boldsymbol{Y}_1 = \sigma(\boldsymbol{K}_1 \otimes \boldsymbol{X} + \boldsymbol{K}_{1b})$$
$$\boldsymbol{Y}_2 = \sigma(\boldsymbol{K}_2 \otimes \boldsymbol{X} + \boldsymbol{K}_{2b})$$

其中，$\boldsymbol{K}_{1b} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$，$\boldsymbol{K}_{2b} = \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix}$。

如果将$\boldsymbol{K}_1$和$\boldsymbol{K}_2$作为一个卷积层的两个卷积核，则该卷积层的输入为特征图$\boldsymbol{X}$，输出为由通道$\boldsymbol{Y}_1$和$\boldsymbol{Y}_2$构成的特征图$\boldsymbol{Y}$。

4.2 卷积神经网络的主要操作

4.2.2 多通道卷积

计算一个卷积层包含的参数总数（参数量）的方法：对于给定的卷积层，假设其输入是通道数为d的特征图$\boldsymbol{X}$，设置了n个F阶卷积核，则该卷积层中参数个数为：

$$n \times d \times F \times F + n$$

说明：卷积核的一个通道就有$F \times F$个参数，一个深度为d的卷积核一共有$d \times F \times F$个参数，因而n个F阶卷积核一共有$n \times d \times F \times F$个参数，再加上每个卷积核都有一个偏置项，因此该卷积层总共有$n \times d \times F \times F + n$个参数。当然，如果不设置偏置项，则参数量为$n \times d \times F \times F$。

4.2 卷积神经网络的主要操作

4.2.3 卷积操作的PyTorch代码实现

1. 卷积层的定义及其参数量的计算

在PyTorch中，二维卷积的定义是利用nn.Conv2d()函数来完成的。我们先看看该函数的使用方法。nn.Conv2d()函数的参数格式如下：

nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0)

其参数意义：

- in_channels：输入的通道数
- out_channels：输出的通道数
- kernel_size：卷积核的大小
- stride：卷积的步长，默认值为1。
- padding：填充0的圈数，默认值为0

4.2 卷积神经网络的主要操作

4.2.3 卷积操作的PyTorch代码实现

该函数定义的卷积层的参数总数为：

out_channels×in_channels×kernel_size×kernel_size+out_channels

例如，下面函数表示输入特征图的通道数为4，输出通道数为16（同时也意味着有16个卷积核），卷积核规模为3×3，步长为1，没有填充：

```
nn.Conv2d(in_channels=4, out_channels=16, kernel_size=3, stride=1, padding=0)
```

该语句等价于下面的语句：

```
nn.Conv2d(4, 16, 3)
```

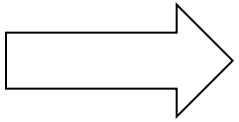
该语句定义的卷积层一共有16×4×3×3+16=592个参数。

4.2 卷积神经网络的主要操作

4.2.3 卷积操作的PyTorch代码实现

为验证参数个数，执行下列代码：

```
conv1 = nn.Conv2d(4, 16, 3)
for para in conv1.parameters():
    print(para.shape)
```



```
torch.Size([16, 4, 3, 3])
torch.Size([16])
```

说明：第一行表示卷积核中参数的个数为16×4×3×3=576，第二行表示偏置项个数为16，一共有592个参数。这与预先计算的结果是一样的。

4.2 卷积神经网络的主要操作

4.2.3 卷积操作的PyTorch代码实现

2. 卷积层的输入

在定义了卷积层以后，能够送入该卷积层的特征图）必须是具有下列形状的4维张量：

(batch_size, in_channels, height, width)

其中，batch_size表示批量的大小，in_channels表示一个样本（如图像或特征图）包含的通道数，height和width分别表示图像或特征图的高度（行数）和宽度（列数）。也就是说，**输入张量时，其第二维的大小必须等于in_channels（其他维的大小不受限制）**，而且**输入的张量必须是4维张量**。

4.2 卷积神经网络的主要操作

4.2.3 卷积操作的PyTorch代码实现

例如，对于由conv1 = nn.Conv2d(7, 16, 3)定义的卷积层，如果输入张量*x*的形状为(128,7,28,28)或(1,7,10,12)，那么下面调用语句会正确执行：

```
y = conv1(x)
```

然而，如果输入特征图*x*的形状为(128,1,28,28)、(4,10,12)或(128,7,28,28,1)，则调用上面语句都将报错，原因在于：前面两个特征图的第二维的大小都不等于7，而第三个特征图的维数为5，不等于4。

4.2 卷积神经网络的主要操作

4.2.3 卷积操作的PyTorch代码实现

3. 卷积层的输出

假设输入特征图*x*的形状为(batch_size, in_channels, height, width)，那么卷积后得到的输出特征图*y*的形状可表示如下：

$$(\text{batch\_size}, \text{out\_channels}, \frac{\text{height}-\text{kernel\_size}+2\times\text{padding}}{\text{stride}}+1, \frac{\text{width}-\text{kernel\_size}+2\times\text{padding}}{\text{stride}}+1)$$

说明：输出特征图*y*的第一个维的大小batch_size保持不变，这是因为输入时有batch_size个样本，输出时也应该分别对这batch_size个样本有“回复”；第二个维的大小等于设定的输出通道数out_channels；第三和第四个维的大小实际上是特征图（或图像）被卷积后的高（行数）和宽（列数），相应公式在前面也已经介绍过。

4.2 卷积神经网络的主要操作

4.2.3 卷积操作的PyTorch代码实现

例如，对于上面提及的卷积层conv1 = nn.Conv2d(4, 16, 3)，当输入张量x的形状为(128,4,28,28)时，则输出张量的形状

$$(128, 16, \frac{28-3+2\times0}{1}+1, \frac{28-3+2\times0}{1}+1) = (128, 16, 26, 26)$$

这个效果可以通过执行下列代码来观察：

```
conv1 = nn.Conv2d(4, 16, 3)
x = torch.randn(128,4,28,28)     #随机产生形状为(128,4,28,28)的输入特征图x
y = conv1(x)
print(x.shape,y.shape)
```

输出： torch.Size([128, 4, 28, 28]) torch.Size([128, 16, 26, 26])

4.2 卷积神经网络的主要操作

4.2.4 池化操作及其PyTorch代码实现

- **池化**（Pooling）：是卷积神经网络中的另一个重要操作，其作用主要是下采样，提取特征图局部区域的显著特征，减少特征图的规模，从而减少计算量。池化层一般放在卷积层后面。需要注意的是，池化层不包含任何待优化的参数。

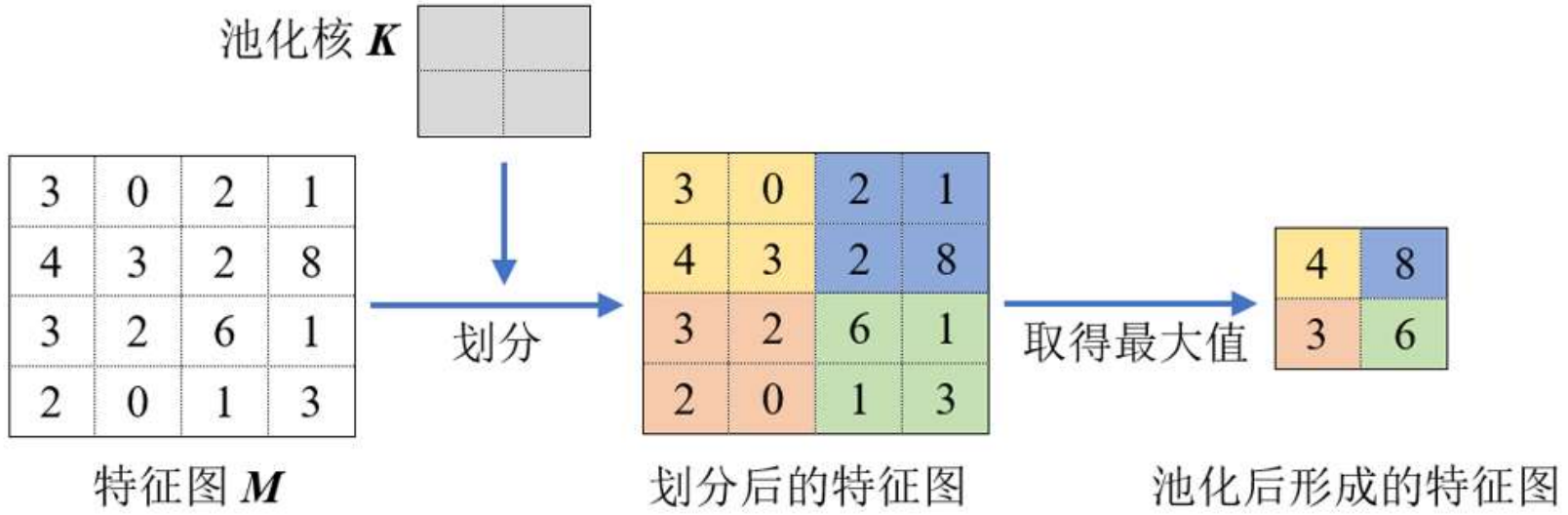
- **池化方法**：有多种池化方法，其中最常用的方法是最大池化（Max Pooling）。

4.2 卷积神经网络的主要操作

4.2.4 池化操作及其PyTorch代码实现

最大池化方方法：要设置一个池化核，但它不包含任何可学习的参数，其主要作用是按照池化核窗口的大小对特征图进行分割。方法是，从左到右、从上往下，将特征图划分为跟池化核窗口一样大小的若干数据区域，最后从每个数据区域中选择一个最大值作为这个区域的代表，所有这些代表按数据区域的位置摆放，重新构成新的特征图。这个新的特征图就是原来特征图被最大池化后的结果。

例如：



4.2 卷积神经网络的主要操作

4.2.4 池化操作及其PyTorch代码实现

平均池化（Mean Pooling）：取每个区域取中数据的平均值作为本区域的代表，从而构成新的特征图。

多通道池化：对于深度为d的特征图，各通道（即各矩阵）独立、但按位同步进行池化，因此池化后形成的特征图的深度仍然为d，即池化操作不改变特征图的深度，或者说不改变输出通道的数量，但在纵向和横向上都极大地缩减了特征图的规模，减少了参与计算的数据量。一般来说，使用的池化核越大，特征图就缩减得越多。一般情况下，通常使用2×2池化核。

4.2 卷积神经网络的主要操作

4.2.4 池化操作及其PyTorch代码实现

最大池化的实现函数：用torch.max_pool2d()函数来实现：

torch.max_pool2d(input, kernel_size, stride, padding)

其参数意义如下：

- **input**：用于接受输入池化层的特征图（张量）
- **kernel_size**：池化核窗口的大小
- **stride**：步长，默认步长跟最大池化窗口大小一致
- **padding**：填充0的方式，默认值为0，表示不填充

4.2 卷积神经网络的主要操作

4.2.4 池化操作及其PyTorch代码实现

假设输入特征图*x*的形状为：(128, 3, 60, 60)

则torch.max_pool2d()返回的特征图是形状为：(128, 3, 30, 30)

经对比可以发现，该函数不改变前面两个参数的值（即不改变特征图的数量和通道数），但后面两个分别表示高度和宽度的参数值都减半了。

4.2 卷积神经网络的主要操作

4.2.4 池化操作及其PyTorch代码实现

在调用该函数时，参数名都可以省略。例如，上面函数与下面三个语句是等价的：

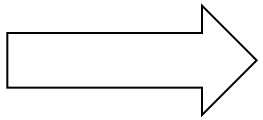
```
torch.max_pool2d(input=x, kernel_size = (2,2), stride=(2,2), padding=0)
y = torch.max_pool2d(x, (2,2), (2,2), 0)        #省略了参数名
y = torch.max_pool2d(x, 2)                      #设置池化窗口大小为2×2，使用了参数的默认值
```

4.2 卷积神经网络的主要操作

4.2.4 池化操作及其PyTorch代码实现

一个例子：下面给出一段代码，它用于随机产生输入特征图*x*，然后对其进行最大池化，最后形成输出特征图*y*：

```
x = torch.randint(0,9,[1,1,4,4])
print(x)
x = x.float()
y = torch.max_pool2d(x, 2)
print(y)
print('x和y的形状分别为：',x.shape, y.shape)
```



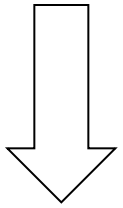
```
tensor([[[[0, 1, 8, 4],
          [8, 2, 6, 4],
          [5, 8, 3, 7],
          [8, 8, 4, 7]]])
tensor([[[[8., 8.],
          [8., 7.]]]])
x 和 y 的形状分别为：  torch.Size([1, 1, 4, 4]) torch.Size([1, 1, 2, 2])
```

4.2 卷积神经网络的主要操作

4.2.4 池化操作及其PyTorch代码实现

平均池化的实现函数：用nn.AvgPool2d ()函数来实现，其参数含义类似于torch.max_pool2d()函数。例如，如果对上述的特征图*x*执行平均池化，则可用下列代码：

```
avgPool = nn.AvgPool2d(2)    #采用2×2的窗口，默认步长为2的平均池化
y = avgPool(x)
print(y)
```



```
tensor([[[[2.7500, 5.5000],
          [7.2500, 5.2500]]]])
```

4.2 卷积神经网络的主要操作

4.2.5 relu激活函数及其应用

relu函数的数学定义如下：

$$\text{relu}(x) = \begin{cases} x, & \text{当 } x \geq 0 \\ 0, & \text{其他} \end{cases}$$

该函数的功能是将小于0的元素变换为0，而大于或等于0的元素保持不变。在[0, +∞]上，其导数为1。

该函数的功能可以利用nn.ReLU()函数来实现。例如：

```
from torch import nn as nn
y = nn.ReLU()(x)
```

4.2 卷积神经网络的主要操作

4.2.5 relu激活函数及其应用

在nn.ReLU()函数中可以设置参数inplace，其默认值为False，表示在函数作用于x后，x的值不变，产生的新结果赋给y；当inplace=True时，表示将函数作用于x后产生的新结果覆盖x，这时y和x是相同的张量，从而节省内存，但原来的x没有了。读者通过执行下列代码来理解二者的区别：

```
from torch import nn as nn
x = torch.randint(-9,9,[3,3])
y = nn.ReLU(inplace=False)(x) #x未被覆盖，其值保持不变
print(x)
print(y)
print(id(x),id(y))
print('------------------')
y = nn.ReLU(inplace=True)(x) #x被覆盖了，节省内存
print(x)
print(y)
print(id(x),id(y))
```

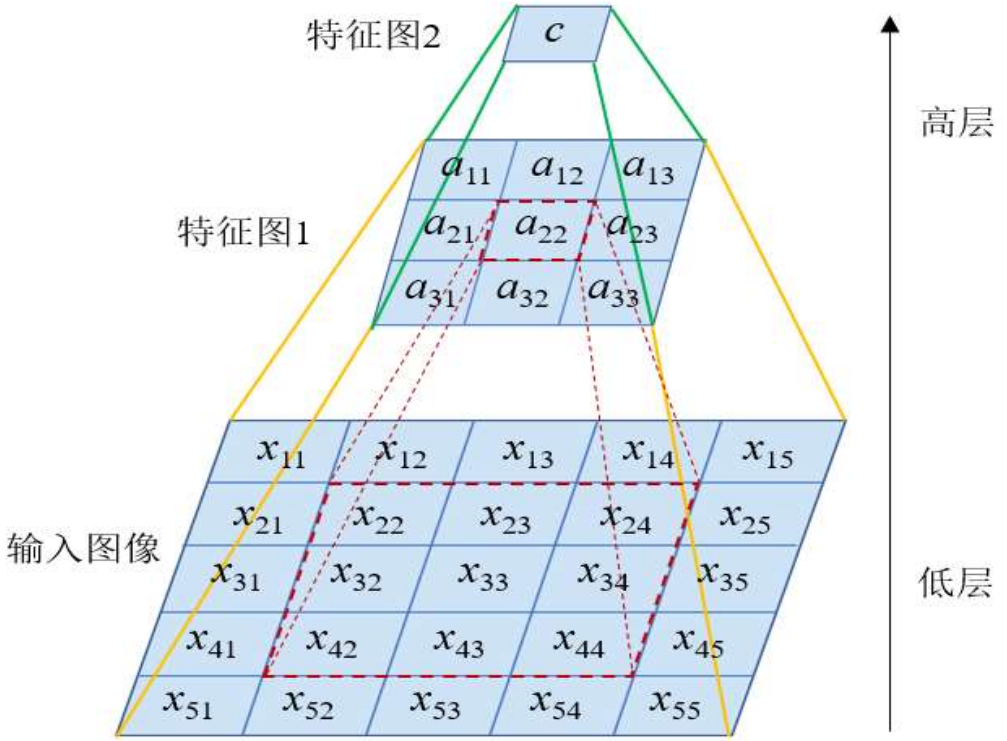
```
tensor([[-8,  7, -1],
        [-2, -1,  5],
        [-4, -2,  8]])
tensor([[0, 7, 0],
        [0, 0, 5],
        [0, 0, 8]])
2657695006656 2657695006512
------------------
tensor([[0, 7, 0],
        [0, 0, 5],
        [0, 0, 8]])
tensor([[0, 7, 0],
        [0, 0, 5],
        [0, 0, 8]])
2657695006656 2657695006656
```

4.2 卷积神经网络的主要操作

4.2.6 感受野

感受野: 在卷积神经网络中，每一层输出的特征图中的元素（或称像素）在原始输入图像上映射的区域大小称为该元素的**感受野**。

例子：右图给出了一个有关感受野的例子。在该图中，元素a_{22}的感受野为由元素$x_{22}, x_{23}, x_{24}, x_{32}, x_{33}, x_{34}, x_{42}, x_{43}, x_{44}$在输入图像上构成的区域（假设卷积核尺寸为3×3）。也就是说，特征图1上每个元素只能感受到下面3×3=9个像素的信息。特征图2上的元素c虽然也只能感受到特征图1上9个元素的信息，但是它通过这9个元素可以进一步感受到输入图像上5×5=25个像素的信息。这说明，c的感受野被扩大了。



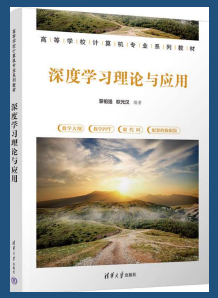
4.2 卷积神经网络的主要操作

4.2.6 感受野

主要结论：

（1）低层特征图（靠近网络输入端）上元素的感受野比较小，相应卷积核主要用于提取局部细粒度特征，如纹理特征、边缘特征等；

（2）高层特征图（靠近网络输出端）上元素的感受野就比较大，相应卷积核则用于提取比较粗粒度的全局语义特征，如脸部轮廓特征、脚部的整体特征等。



本章内容

contents

4.3 卷积神经网络的设计方法

4.3.1 基本设计原则

一个深度神经网络通常采用下列组合方式：

输入层 + [[卷积层]×n + [池化层]]×m + [全连接层]×k

这个组合方式表示：首先是输入层，然后是n个卷积层的叠加，后面紧跟着一个可选的池化层，并重复这种“[卷积层]×n + [池化层]”结构m次，最后面连接k个全连接层。当然，这种组合方式只是一种参考，在实践中并非一定要按照这种组合方式来构建网络。

例如：在例4.1中，卷积神经网络采用了下列组合方式：

输入层 + 卷积层×1 + 池化层 + 卷积层×1 + 全连接层×2

4.3 卷积神经网络的设计方法

4.3.1 基本设计原则

大型网络的构建方法：

（1）加宽每个网络层，即增加网络层中卷积核的数量；

（2）加深网络，即增加网络层的数量。

说明：增加卷积核数量会极大增加计算开销，往往“得不偿失”，因而在实践中通常是通过增加网络层来构建大型网络。

注意：神经网络是不是越大越好呢？这主要是由问题的复杂度和可获得训练数据的量来决定。一般来说，问题越复杂，就需要表达能力越强的网络来解决；网络越大（参数越多），其表达能力和泛化能力就越强，但同时就需要更多的带标注的训练数据，也需要更强算力支持。因此，小问题用小型网络来解决，大问题才考虑用大网络模型类解决。

4.3 卷积神经网络的设计方法

4.3.2 网络结构查看和参数量计算

粗略查看网络结构的方法：直接将创建的网络类的实例打印出来，从中可领会到所创建的网络的拓扑结构。例如，欲查看例4.1中创建的网络的结构，可将相应的实例打印出来：

print(example4_1)

```
Example4_1(
    (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
    (conv2): Conv2d(10, 20, kernel_size=(3, 3), stride=(1, 1))
    (fc1): Linear(in_features=2000, out_features=500, bias=True)
    (fc2): Linear(in_features=500, out_features=10, bias=True)
)
```

说明：该网络包含了两个卷积层和两个全连接层以及相关参数设计，其中网络接受的输入的通道数为1。当然，这些网络层是否真的用于构建网络，那还得看forward()函数中编写的代码逻辑。也就是说，这种查看方式虽然简单，但不准确，也没有给出网络的参数量。

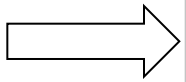
4.3 卷积神经网络的设计方法

4.3.2 网络结构查看和参数量计算

准确查看网络结构信息：利用模块summary实现，它可以提供更为详细的信息。例如，为查看例4.1中的网络结构及参数量，可利用下列代码来实现：

```
from torchsummary import summary
                #引入模块summary
print(summary(example4_1, input_size=(1, 28, 28)))   #输出example4_1的网络结构信息
```

其中，(1, 28, 28)为网络接受的输入张量的形状（但不带batch_size）。



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 10, 24, 24]             260
            Conv2d-2           [-1, 20, 10, 10]           1,820
            Linear-3                 [-1, 500]       1,000,500
            Linear-4                  [-1, 10]           5,010
================================================================
Total params: 1,007,590
Trainable params: 1,007,590
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.06
Params size (MB): 3.84
Estimated Total Size (MB): 3.91
```

4.3 卷积神经网络的设计方法

4.3.2 网络结构查看和参数量计算

通过执行下列代码，我们也可以看到整个网络的参数量为1007590：

```
sum = 0
for param in example4_1.parameters():  #计算整个网络的参数量
    sum += torch.numel(param)
print('该网络参数总量：  %d'%sum)
```

4.3 卷积神经网络的设计方法

4.3.3 一个猫狗图像分类实例

【例4.3】构造一个用于实现猫和狗图像分类的深度神经网络。

数据集：猫和狗图像是来自Kaggle 竞赛的一个赛题数据集Cat vs Dog。我们从该数据集中随机抽取10028张图片来构造训练集和测试集，相关信息见表4-1。

表 4-1　所设置数据集的基本信息

	训练集	测试集	合计
猫	4000	1011	5011
狗	4005	1012	5017
合计	8005	2023	10028

图4-18　部分猫和狗的图像样例

4.3 卷积神经网络的设计方法

4.3.3 一个猫狗图像分类实例

开发的主要步骤：

（1）编写Dataset类的子类来加载数据，这也是常用的图像加载方法。Dataset类提供两种方法来实现数据加载。一种是__len__()方法，通过重写该方法，让其提供数据集的大小；另一种是__getitem__()方法，该方法支持从0到__len__(self)的索引，从而可以自动获取数据集中的每一条样本，以对其进行相应的“加工”，如数据预处理、张量化、添加标签等。这两个方法都是被隐式调用，它们与DataLoader类结合，可以通过多线程加速数据的加载速度。

（2）构造一个深度神经网络，它包含4个卷积层和3个全连接层。这个结构并非一定是最佳的，读者可以进一步调整其宽度（每层卷积核的个数）和高度（网络层数）以及学习率等超参数，不断尝试各种组合，以寻求更好的分类结果。

（3）最后编写训练和测试代码，包括在训练集上的测试和在测试集上的测试结果。

以下是该程序的所有代码。

4.3 卷积神经网络的设计方法

4.3.3 一个猫狗图像分类实例

开发的主要步骤：

（1）编写Dataset类的子类来加载数据，这也是常用的图像加载方法。Dataset类提供两种方法来实现数据加载。一种是__len__()方法，通过重写该方法，让其提供数据集的大小；另一种是__getitem__()方法，该方法支持从0到__len__(self)的索引，从而可以自动获取数据集中的每一条样本，以对其进行相应的“加工”，如数据预处理、张量化、添加标签等。这两个方法都是被隐式调用，它们与DataLoader类结合，可以通过多线程加速数据的加载速度。

相应代码：

```
class cat_dog_dataset(Dataset):
    def __init__(self, dir):
        self.dir = dir
        self.files = os.listdir(dir)
    def __len__(self):    #需要重写该方法，返回数据集大小
        t = len(self.files)
        return t
    def __getitem__(self, idx):
        file = self.files[idx]
        fn = os.path.join(self.dir, file)
        img = Image.open(fn).convert('RGB')
        img = transform(img)      #调整图像形状为(3,224,224), 并转为张量
        img = img.reshape(-1,224,224)
        y = 0 if 'cat' in file else 1    #构造图像的类别
        return img,y
```

4.3 卷积神经网络的设计方法

4.3.3 一个猫狗图像分类实例

（2）构造一个深度神经网络，它包含4个卷积层和3个全连接层。这个结构并非一定是最佳的，读者可以进一步调整其宽度（每层卷积核的个数）和高度（网络层数）以及学习率等超参数，不断尝试各种组合，以寻求更好的分类结果。

（3）最后编写训练和测试代码，包括在训练集上的测试和在测试集上的测试结果。

4.3 卷积神经网络的设计方法

4.3.3 一个猫狗图像分类实例

核心代码:

```
#定义卷积神经网络
class Model_CatDog(nn.Module):
    def __init__(self):
        super().__init__()
        #以下定义4个卷积层:
self.conv1 = nn.Conv2d(3, 64, kernel_size=5, padding=2) #5*5卷积核,
#默认步长1，填充2
        self.conv2 = nn.Conv2d(64, 128, 5)  #默认步长1，填充0，下同
        self.conv3 = nn.Conv2d(128, 128, 3)
        self.conv4 = nn.Conv2d(128, 256, 3)
        # 以下定义3个全连接层:
        self.fc1 = nn.Linear(256*12*12, 2048)
        self.fc2 = nn.Linear(2048, 512)
        self.fc3 = nn.Linear(512, 2)
```

4.3 卷积神经网络的设计方法

4.3.3 一个猫狗图像分类实例

```
def forward(self,x): #此时，x的形状为(batch, 224, 224, 3)
        out = self.conv1(x)     # (batch, 3, 224, 224)--->(batch, 64, 224, 224)
        out = nn.ReLU(inplace=True)(out) #形状不变（以下，形状不变的地方，
#不再注释说明）
        out = nn.MaxPool2d(2, 2)(out)  # (batch, 64, 224, 224)--->
#(batch, 64, 112, 112)
        out = self.conv2(out)  #(batch, 64, 112, 112)--->(batch, 128, 108, 108)
        out = nn.ReLU(inplace=True)(out)
        out = nn.MaxPool2d(2, 2)(out)       #(batch, 128, 108, 108)--->
#(batch, 128, 54, 54)
        out = self.conv3(out)       #(batch, 128, 54, 54)--->(batch, 128, 52, 52)
        out = nn.ReLU(inplace=True)(out)
        out = nn.MaxPool2d(2, 2)(out) #(batch, 128, 52, 52)--->(batch, 128, 26, 26)
        out = self.conv4(out)    #(batch, 128, 26, 26)--->(batch, 256, 24, 24)
        out = nn.ReLU(inplace=True)(out)
```

4.3 卷积神经网络的设计方法

4.3.3 一个猫狗图像分类实例

```
out = nn.MaxPool2d(2, 2)(out)       #(batch, 256, 24, 24)--->(batch, 256, 12, 12)
out = out.reshape(x.size(0), -1)     #(batch, 256, 12, 12)--->(batch, 36864)
out = nn.Dropout(0.5)(out)
out = self.fc1(out)            #(batch, 36864)--->(batch, 2048)
out = nn.ReLU(inplace=True)(out)
out = nn.Dropout(0.5)(out)
out = self.fc2(out)            #(batch, 2048)--->(batch, 512)
out = nn.ReLU(inplace=True)(out)
out = self.fc3(out)            #(batch, 512)--->(batch, 2)
return out #(batch, 2)
```

注：请从教材P122或出版社网站上查看或下载全部代码

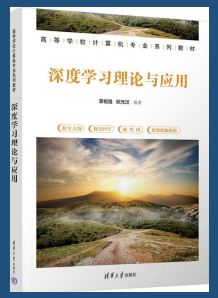
4.3 卷积神经网络的设计方法

4.3.3 一个猫狗图像分类实例

执行上述代码，结果如下（部分）：

```
…… ……
第 29 轮循环中，损失函数的平均值为: 0.0337
第 30 轮循环中，损失函数的平均值为: 0.0396
训练时间为:  1428.0 秒
1. 网络模型在训练集上的准确率：99.16%
2. 网络模型在测试集上的准确率：81.27%
```

说明：这个准确率不算高，读者可以进一步调试一些超参数，尝试更好的结果。



本 章 内 容

contents

4.4 过拟合及其解决方法

过拟合问题：在训练神经网络的时候，有时可利用的样本比较少，而模型参数又很多。这样，经过多次循环训练后模型将“记住”数据分布的几乎所有细节，因而模型在训练集上可以表现出良好的性能，如非常高的准确和非常低的损失函数值等。但是，一旦用到测试集上，模型的性能往往非常差。这种过渡拟合了训练数据分布的现象称为**过拟合**。

判断和解决过拟合的方法之一：可以每间隔一定循环代数做一次在训练集和测试集上的性能测试，以找到这样的迭代次数N：从第N代以后，模型在训练集上的性能继续走高，而在测试集上的性能却开始走低，如图4-19所示。这样，我们让模型训练到第N代即可，或者选择第N代时的模型即可。

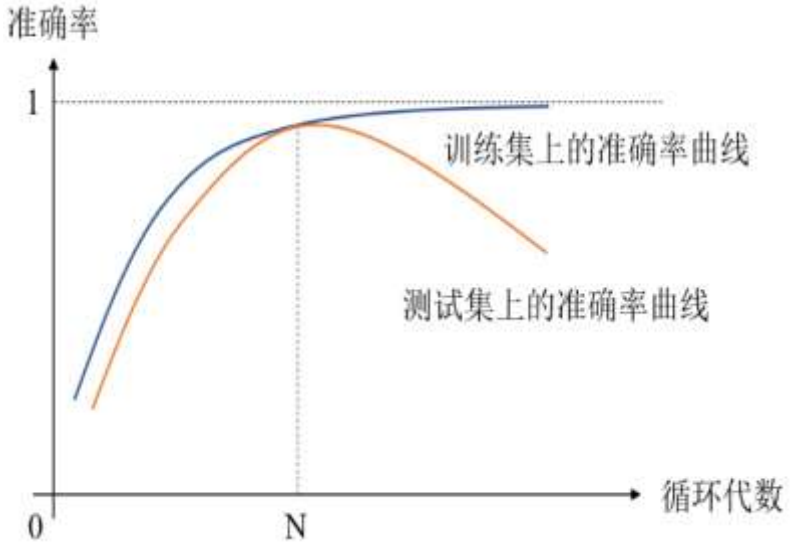


图4-19 一种判断过拟合的示意图

4.4 过拟合及其解决方法

解决过拟合的方法之二：正则化项法，该方法是在原有损失函数$\mathcal{L}$上增加一个正则化项$||w||_2^2$，得到：

$$\mathcal{L} + ||w||_2^2 = \mathcal{L} + (w_1^2 + w_2^2 + \cdots w_n^2)$$

其中，$w_1, w_2, \ldots, w_n$为模型包含的权重。当模型的损失函数设计为上述形式的时候，损失函数在被极小化时，各个权重也在被极小化。这样，有的参数就变得很小，其效果相当于抑制了相应的神经元，从而保持了各个神经元的多元化，降低过拟合的可能性。

4.4 过拟合及其解决方法

解决过拟合的方法之三：丢弃法，实际上就是Dropout方法，它的基本原理就是在训练的时候按既定的比例随机冻结部分神经元，以避免部分神经过于“强势”而导致其他神经元失去功能，从而保证神经元的多样化，减缓过拟合问题。

说明：过拟合是样本过少造成的，因此也可以通过数据增强的方法来补充更多的样本数据，从而在源头上解决过拟合问题。

4.5 本章小结

本章内容：

- 单通道卷积
- 多通道卷积
- 池化方法
- 卷积神经网络的设计方法
- 过拟合问题及其解决方法