深度学习快速入门

07 卷积原理

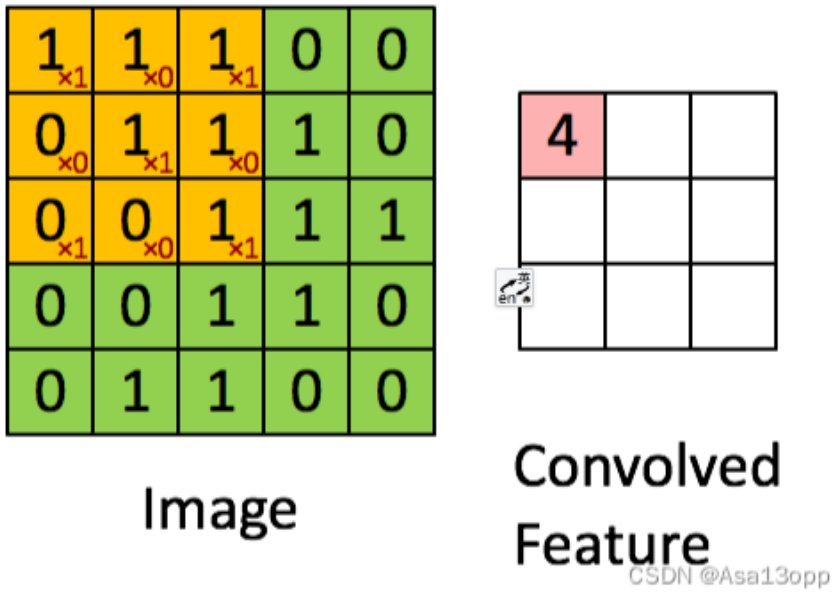
POET

2024 年 2 月 12 日

1 卷积原理

卷积核不停的在原图上进行滑动，对应元素相乘再相加。

下图为每次滑动移动 1 格，然后再利用原图与卷积核上的数值进行计算得到缩略图矩阵的数据，如下图所示：



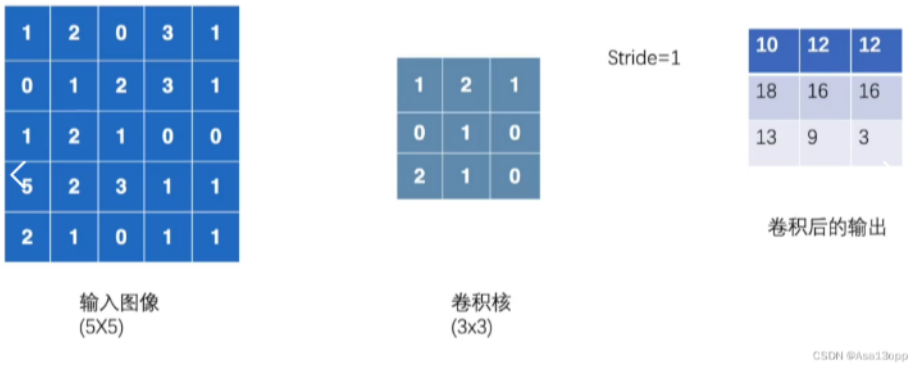

```
import torch
import torch.nn.functional as F
input = torch.tensor([[1, 2, 0, 3, 1],
                      [0, 1, 2, 3, 1],
                      [1, 2, 1, 0, 0],
                      [5, 2, 3, 1, 1],
                      [2, 1, 0, 1, 1]])

kernel = torch.tensor([[1, 2, 1],
                       [0, 1, 0],
                       [2, 1, 0]])
```

```
#这里面conv2d(N,C,H,W)里面的四个是 N就是batch size 也就是输入图片的数量，
#C就是通道数这只是一个二维张量所以通道为1，H就是高，W就是宽，所以是1 1 5 5
input = torch.reshape(input, (1, 1, 5, 5))
kernel = torch.reshape(kernel, (1, 1, 3, 3))
output = F.conv2d(input, kernel, stride=1)
print(output)
```

输出结果如下所示：

$$\text{tensor}\left(\left[\left[\begin{bmatrix} 10 & 12 & 12 \\ 18 & 16 & 16 \\ 13 & 9 & 3 \end{bmatrix}\right]\right]\right)$$




```
import torch
import torch.nn.functional as F

input = torch.tensor([[1, 2, 0, 3, 1],
                      [0, 1, 2, 3, 1],
                      [1, 2, 1, 0, 0],
                      [5, 2, 3, 1, 1],
                      [2, 1, 0, 1, 1]])
```

```
9
10      kernel = torch.tensor([[1, 2, 1],
11                             [0, 1, 0],
12                             [2, 1, 0]])
13
14      print(input.shape)
15      print(kernel.shape)
16      input = torch.reshape(input, (1,1,5,5))
17      kernel = torch.reshape(kernel, (1,1,3,3))
18      print(input.shape)
19      print(kernel.shape)
20
21      output2 = F.conv2d(input, kernel, stride=2)  # 步
            伐为2
22      print(output2)
```

输出结果：

torch.Size([5, 5])
torch.Size([3, 3])
torch.Size([1, 1, 5, 5])
torch.Size([1, 1, 3, 3])
tensor([[[[10, 12], [13, 3]]]])

```
1       import torch
2       import torch.nn.functional as F
3
4       input = torch.tensor([[1, 2, 0, 3, 1],
5                             [0, 1, 2, 3, 1],
6                             [1, 2, 1, 0, 0],
7                             [5, 2, 3, 1, 1],
8                             [2, 1, 0, 1, 1]])
9
10      kernel = torch.tensor([[1, 2, 1],
```

```
                                [0, 1, 0],
                                [2, 1, 0]])

    print(input.shape)
    print(kernel.shape)
    input = torch.reshape(input, (1,1,5,5))
    kernel = torch.reshape(kernel, (1,1,3,3))
    print(input.shape)
    print(kernel.shape)

    output3 = F.conv2d(input, kernel, stride=1,
        padding=1) # 周围只填充一层，填充部分默认为零
    print(output3)
```

输出结果：

torch.Size([5, 5])
torch.Size([3, 3])
torch.Size([1, 1, 5, 5])
torch.Size([1, 1, 3, 3])
tensor([[[[1, 3, 4, 10, 8],
[5, 10, 12, 12, 6],
[7, 18, 16, 16, 8],
[11, 13, 9, 3, 4],
[14, 13, 9, 7, 4]]]])

2 步幅、填充原理

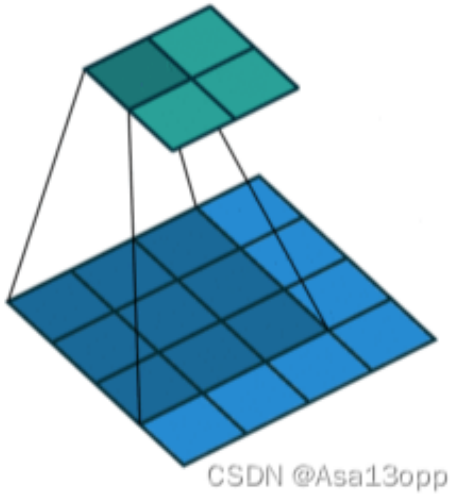
1. **步幅：**卷积核经过输入特征图的采样间隔。设置步幅的目的：希望减小输入参数的数目，减少计算量。

2. **填充：**在输入特征图的每一边添加一定数目的行列。设置填充的目的：希望每个输入方块都能作为卷积窗口的中心，或使得输出的特征图的

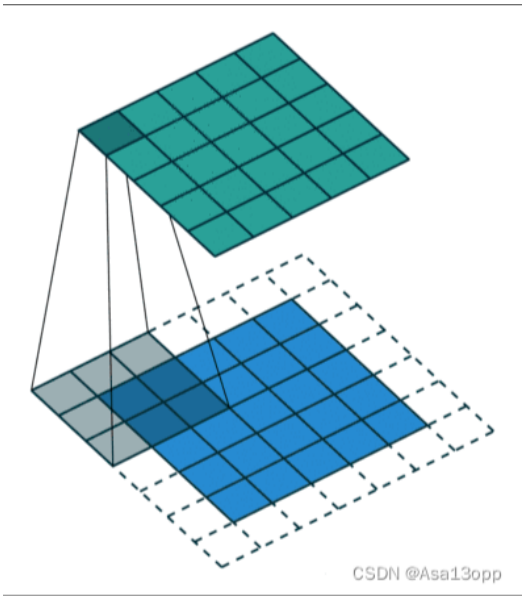
长、宽 = 输入的特征图的长、宽。

3. 一个尺寸 a * a 的特征图，经过 b * b 的卷积层，步幅（stride）= c，填充（padding）= d，若 d 等于 0，也就是不填充，输出的特征图的尺寸 =（a-b）/ c+1；若 d 不等于 0，也就是填充，输出的特征图的尺寸 =（a+2d-b）/ c+1。

例子 1 一个特征图尺寸为 4 * 4 的输入，使用 3 * 3 的卷积核，步幅 =1，填充 =0，输出的尺寸 =(4 - 3)/1 + 1 = 2。



例子 2 一个特征图尺寸为 5 * 5 的输入，使用 3 * 3 的卷积核，步幅 =1，填充 =1，输出的尺寸 =(5 + 2 * 1 - 3)/1 + 1 = 5。



例子 3 一个特征图尺寸为 6 * 6 的输入，使用 3 * 3 的卷积核，步幅 =2，填充 =1，输出的尺寸 $=(6+2*1-3)/2+1=2.5+1=3.5$ 向下取整 =3（降采样：边长减少 1/2）。