Padding

在应用多层卷积时，我们常常丢失边缘像素。 由于我们通常使用小卷积核，因此对于任何单个卷积，我们可能只会丢失几个像素。 但随着我们应用许多连续卷积层，累积丢失的像素数就多了。

解决这个问题的简单方法即为填充.

通常，如果我们添加行填充p_h（大约一半在顶部，一半在底部）和p_w列填充（左侧大约一半，右侧一半），则输出形状将为$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$

Stride

在计算互相关时，卷积窗口从输入张量的左上角开始，向下、向右滑动。默认每次滑动一个元素。但是，有时候为了高效计算或是缩减采样次数，卷积窗口可以跳过中间位置，每次滑动多个元素。

通常，当垂直步幅为s_h、水平步幅为p_h时，输出形状为$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor.$

如果我们设置了$p_h = k_h - 1$和$p_w = k_w - 1$,则输出形状将简化为$\lfloor (n_h + s_h - 1)/s_h \rfloor \times \lfloor (n_w + s_w - 1)/s_w \rfloor$

更进一步，如果输入的高度和宽度可以被垂直和水平步幅整除，则输出形状将为$(n_h/s_h) \times (n_w/s_w)$

Pytorch Code Demo

```
def comp_conv2d(conv2d,X):
    X = X.reshape((1,1) + X.shape)
    Y = conv2d(X)
    return Y.reshape(Y.shape[2:])

conv2d = nn.Conv2d(in_channels=1,out_channels=1, # 输入通道数，输出通道数
                   kernel_size=3,padding=1,stride=2) # 卷积核大小，填充，步长
X = torch.rand(size=(8,8))
print(comp_conv2d(conv2d,X).shape)

conv2d = nn.Conv2d(1,1,(5,3),padding=(2,1))
print(comp_conv2d(conv2d,X).shape)

conv2d = nn.Conv2d(1,1,3,2,1)
print(comp_conv2d(conv2d,X).shape) # (8+2-3)/2取floor为4

conv2d = nn.Conv2d(1,1,(3,5),(3,4),(0,1))
print(comp_conv2d(conv2d,X).shape)
'''
第一组：卷积核大小为 3x3，步长为 2，填充为 1，输入大小为 8x8，卷积后输出大小为 4x4。
输出公式：
(8+2x1-3)/2+1=4
第二组：卷积核大小为 5x3，填充为 2x1，步长为 1，输入大小为 8x8，卷积后输出大小为 8x8。
输出公式：
(8+2x2-5+1,8+2x1-3+1)=(8,8)
第三组：卷积核大小为 3x3，步长为 2，填充为 1，输入大小为 8x8，卷积后输出大小为 4x4。
输出公式：
(8+2x1-3)/2+1=4
第四组：卷积核大小为 3x5，步长为 3x4，填充为 0x1，输入大小为 8x8，卷积后输出大小为 2x2。
输出公式：
((8+0x1-3)/3+1,(8+2x1-5+1)/4)=(8,8)
'''
```