转置卷积(Transposed Convolution)

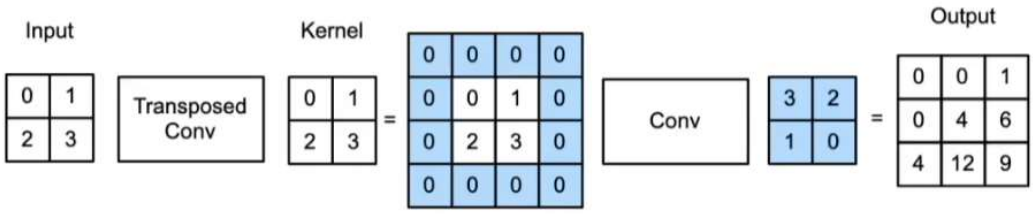
- 卷积操作一般不会用来增大高宽，要么不变要么减半
- **转置卷积**可以用来增大高宽
- 卷积操作可以理解为矩阵乘法$Y = X \star W$,可以理解为把X, Y都展平为X', Y',则卷积的操作可以等价为 $Y' = V \cdot X'$
- 那么转置卷积就是把Y' and X'调转过来，把Y'当成输入，X'当成输出 $X' = V^T \cdot Y'$ 写一下矩阵维度就能明白了，相当于把卷积操作逆回来了
- 转置卷积一般是做上采样

重谈转置卷积

- 下面的正常卷积都是padding=0， stride=1

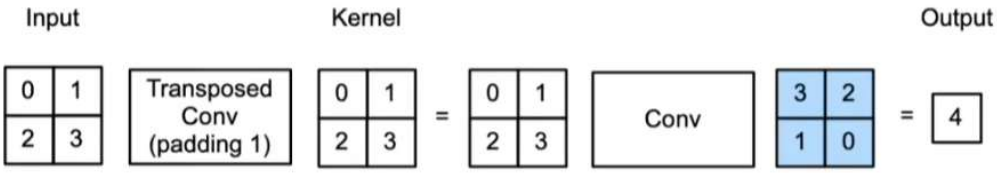
当填充为0， 步幅为1的时候

- 将输入填充$k-1$
- 将核矩阵上下左右翻转
- 然后做正常的卷积



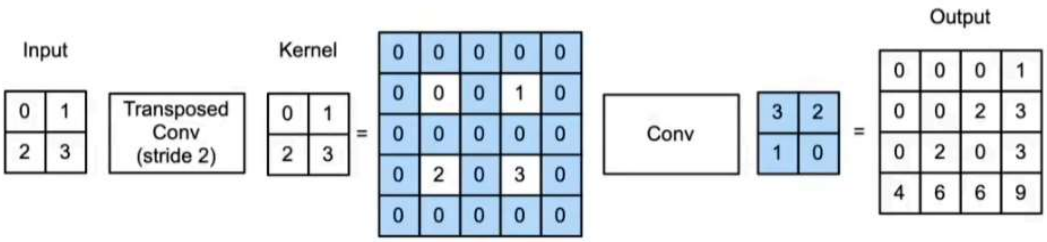
当填充为p，步幅为1的时候

- 将输入填充$k-p-1$
- 将核矩阵上下左右翻转
- 然后做正常卷积



当填充为p，步幅为s时

- 将输入矩阵行和列之间插入$s-1$行
- 将输入填充$k-p-1$
- kernel上下左右翻转
- 然后做正常卷积



转置卷积形状换算

- 输入高宽为n， 核k， 填充p， 步幅s
- 转置卷积 $n' = n * s + k - 2p - s$
  - 卷积: $n' = \lfloor (n - k + 2p + s)/s \rfloor \Rightarrow n \geq sn' + k - 2p - s$
- 如果让高宽成倍增加，那么$k = 2p + s$

转置卷积 $\neq$反卷积

- **反卷积**的概念是，$Y = conv(X, K)$, 反卷积操作是$X = deconv(Y, K)$，是将输出和核通过反卷积操作得到输入，值相等，但是转置卷积的值并不相等，只是形状可以上采样到一样的大小

In [1]:
```
import torch
import torchvision
from d2l import torch as d2l
from torch import nn
```

In [2]:
```
def tran_conv(X:torch.Tensor, K:torch.Tensor):
    """给定输入X和卷积核K，对其进行转置卷积运算"""
    h, w = K.shape
    Y = torch.zeros((X.shape[0] + h - 1, X.shape[1] + w - 1))
    for i in range(X.shape[0]):
        for j in range(X.shape[1]):
            Y[i:i + h, j:j + w] += X[i, j] * K
    return Y
```

In [3]:
```
X = torch.tensor([[0., 1.],[2., 3.]])
K = torch.tensor([[0., 1.],[2., 3.]])
tran_conv(X, K)
```

Out[3]:
```
tensor([[ 0.,  0.,  1.],
        [ 0.,  4.,  6.],
        [ 4., 12.,  9.]])
```

- 使用torch内置API进行实现

In [4]:
```
X = X.reshape(1, 1, 2, 2)
K = K.reshape(1, 1, 2, 2)
tconv = nn.ConvTranspose2d(1, 1, kernel_size=2, bias=False)
tconv.weight.data = K
tconv(X)
```

Out[4]:
```
tensor([[[[ 0.,  0.,  1.],
          [ 0.,  4.,  6.],
          [ 4., 12.,  9.]]]], grad_fn=<ConvolutionBackward0>)
```

转置卷积的填充，步幅和多通道

- 转置卷积的填充是在输出上的填充，因为转置卷积是卷积的逆运算，所以padding相当于是在输出（原始输入）上的padding，其实是不需要的部分，所以应该加以扣除，也就是说，转置卷积的padding是在输出的tensor上扣掉padding的部分

In [5]:
```
tconv = nn.ConvTranspose2d(1, 1, kernel_size=2, padding=1, bias=False)
tconv.weight.data = K
tconv(X)
# 在这个例子中，输入图像时2 * 2，经过转置卷积后是3 * 3， 因为padding是1，所以在上
```

Out[5]:
```
tensor([[[[4.]]]], grad_fn=<ConvolutionBackward0>)
```

- 转置卷积中的stride是控制特征图放大倍数的，因为卷积操作中是把图片的宽高减少到原来的$1/stride$，所以转置卷积的操作就是把特征图放大到原来的$stride$倍

In [6]:
```
tconv = nn.ConvTranspose2d(1, 1, kernel_size=2, stride=2, bias=False)
tconv.weight.data = K
tconv(X).shape
```

Out[6]:
```
torch.Size([1, 1, 4, 4])
```

- 转置卷积的输入输出通道和卷积的是类似

In [7]:
```
X = torch.rand((1, 10, 16, 16))
conv = nn.Conv2d(10, 20, kernel_size=5, padding=2, stride=3)
tconv = nn.ConvTranspose2d(20, 10, kernel_size=5, padding=2, stride=3)
tconv(conv(X)).shape == X.shape
```

Out[7]:
```
True
```

转置卷积与矩阵变换的联系

- 上文已经说了，我们可以把图片拉成向量的形式，然后卷积核就可以写成权重矩阵的形式

In [ ]:
```
X = torch.arange(9).reshape(3, 3)
K = torch.tensor([[1.0, 2.0], [3.0, 4.0]])
Y = d2l.corr2d(X, K)
Y
```

Out[ ]:
```
(tensor([[27., 37.],
         [57., 67.]]),
 tensor([[0, 1, 2],
         [3, 4, 5],
         [6, 7, 8]]))
```

- 接下来我们对X，Y，K做矩阵变换，X被拉成向量的话维度是(9,),同理Y(4,)，所以权重的形状为(4, 9)

In [10]:
```
def kernel2matrix(K):
    """把卷积核变成权重矩阵"""
    k, W = torch.zeros(5), torch.zeros((4, 9))
    k[:2], k[3:] = K[0, :], K[1, :]
    W[0, :5], W[1, 1:6], W[2, 3:8], W[3, 4:] = k, k, k, k
    return W

W = kernel2matrix(K)
W
```

Out[10]:
```
tensor([[1., 2., 0., 3., 4., 0., 0., 0., 0.],
        [0., 1., 2., 0., 3., 4., 0., 0., 0.],
        [0., 0., 0., 1., 2., 0., 3., 4., 0.],
        [0., 0., 0., 0., 1., 2., 0., 3., 4.]])
```

In [13]:
```
Y == torch.matmul(W, X.reshape(-1).float()).reshape(2, 2)
```

Out[13]:
```
tensor([[True, True],
        [True, True]])
```