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

In [8]:
```
def corr2d(X,K):
    h,w=K.shape
    Y = torch.zeros((X.shape[0]-h+1, X.shape[1]-w+1))
    for i in range(Y.shape[0]):
        for j in range(Y.shape[1]):
            Y[i,j] = (X[i:i+h,j:j+w]*K).sum()
    return Y
```

In [9]:
```
X = torch.tensor([[0.0,1.0,2.0],[3.0,4.0,5.0],[6.0,7.0,8.0]])
K = torch.tensor([[0.0,1.0],[2.0,3.0]])
corr2d(X,K)
```

Out[9]:
```
tensor([[19., 25.],
        [37., 43.]])
```

In [10]:
```
class Conv2D(nn.Module):
    def __init__(self, kernel_size):
        super().__init__()
        self.weight = nn.Parameter(torch.rand(kernel_size))
        self.bias = nn.Parameter(torch.zeros(1))

    def forward(self, x):
        return corr2d(x, self.weight) + self.bias
```

In [11]:
```
X = torch.ones((6, 8))
X[:, 2:6] = 0
X
```

Out[11]:
```
tensor([[1., 1., 0., 0., 0., 0., 1., 1.],
        [1., 1., 0., 0., 0., 0., 1., 1.],
        [1., 1., 0., 0., 0., 0., 1., 1.],
        [1., 1., 0., 0., 0., 0., 1., 1.],
        [1., 1., 0., 0., 0., 0., 1., 1.],
        [1., 1., 0., 0., 0., 0., 1., 1.]])
```

In [12]:
```
K = torch.tensor([[1.0, -1.0]])
```

In [14]:
```
Y = corr2d(X, K)
Y
```

Out[14]:
```
tensor([[ 0.,  1.,  0.,  0.,  0., -1.,  0.],
        [ 0.,  1.,  0.,  0.,  0., -1.,  0.],
        [ 0.,  1.,  0.,  0.,  0., -1.,  0.],
        [ 0.,  1.,  0.,  0.,  0., -1.,  0.],
        [ 0.,  1.,  0.,  0.,  0., -1.,  0.],
        [ 0.,  1.,  0.,  0.,  0., -1.,  0.]])
```

In [15]:
```
corr2d(X.t(), K)
```

Out[15]:
```
tensor([[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]])
```

In [16]:
```
conv2d = nn.LazyConv2d(1, kernel_size=(1, 2), bias=False)

X = X.reshape((1, 1, 6, 8))
Y = Y.reshape((1, 1, 6, 7))
lr = 3e-2

for i in range(10):
    Y_hat = conv2d(X)
    l = (Y_hat - Y) ** 2
    conv2d.zero_grad()
    l.sum().backward()
    conv2d.weight.data[:] -= lr * conv2d.weight.grad
    if (i + 1) % 2 == 0:
        print(f'epoch {i + 1}, loss {l.sum():.3f}')
```

```
epoch 2, loss 9.297
epoch 4, loss 3.076
epoch 6, loss 1.137
epoch 8, loss 0.445
epoch 10, loss 0.179
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/torch/nn/modules/lazy.py:181: UserWarning: Lazy modules are a new feature under heavy development so changes to the API or functionality can happen at any moment.
  warnings.warn('Lazy modules are a new feature under heavy development '
```

In [17]:
```
conv2d.weight.data.reshape((1, 2))
```

Out[17]:
```
tensor([[ 0.9476, -1.0343]])
```

Discussion: 이번 단원에서는 CNN에서 사용하는 연산인 Cross-correlation Operation, 교차 상관 연산을 배웠다. 입력 텐서와 커널 텐서를 통해 출력 텐서를 생성하는 연산이다. 이때, 커널이라 함은 이미지의 특정 부분에 대한 연산을 수행하여 해당 부분에서 중요한 정보를 추출하는 것을 뜻한다. 합성곱에서는 커널을 뒤집어서 연산을 수행하지만, 이러한 커널은 학습 과정에서 데이터를 통해 자동으로 학습되기 때문에 CNN에서는 이 cross-correlation과 합성곱 연산의 결과가 크게 달라지지 않고 그렇기 때문에 이 분야에서는 거의 비슷한 말로 쓰인다고 한다. 이런 방식으로 이미지 데이터의 특징을 활용해서 다른 방법의 학습을 한다는 것이 인상적이었다.

Excercise: Construct an image X with diagonal edges. <- X를 단순화해서 대각행렬로 생각해보자.

What happens if you apply the kernel K in this section to it?

어떤 K를 말하는지 살짝 헷갈리지만, 수정한 K = [1 -1]을 기준으로 하면, 이 커널은 가로로 인접한 두 픽셀의 차이를 계산하기 때문에, 어느 정도 이미지 경계를 파악 가능하지만 가로+세로를 포함한 대각선의 특성상 완전한 탐지는 힘들 것 같다.

What happens if you transpose X?

X를 행렬로 표현하면, 대각 행렬이고, 대각 행렬에서 X^T = X 이므로 동일할 것이다.

What happens if you transpose K?

K를 전치하면, 세로의 변화를 잘 감지할 것 같다. K^T를 해보면..