Convolution Layers

In [1]:
```
from mxnet import autograd, nd
from mxnet.gluon import nn
```

The Cross Correlation Operator

In [2]:
```
def corr2d(X, K):
    h, w = K.shape
    Y = nd.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
```

Sanity Test

In [3]:
```
X = nd.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
K = nd.array([[0, 1], [2, 3]])
corr2d(X, K)
```

Out[3]:
```
[[19. 25.]
 [37. 43.]]
<NDArray 2x2 @cpu(0)>
```

Convolutional Layers

In [4]:
```
class Conv2D(nn.Block):
    def __init__(self, kernel_size, **kwargs):
        super(Conv2D, self).__init__(**kwargs)
        self.weight = self.params.get('weight', shape=kernel_size)
        self.bias = self.params.get('bias', shape=(1,))

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

Object Edge Detection in Images

In [5]:
```
X = nd.ones((6, 8))
X[:, 2:6] = 0
X
```

Out[5]:
```
[[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.]]
<NDArray 6x8 @cpu(0)>
```

Detect Vertical Edges

In [6]:
```
K = nd.array([[1, -1]])
Y = corr2d(X, K)
print(X, Y)
```

```
[[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.]]
<NDArray 6x8 @cpu(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.]
 [ 0.  1.  0.  0.  0. -1.  0.]]
<NDArray 6x7 @cpu(0)>
```

Can't Detect Horizon Edges

In [7]:
```
Z = corr2d(X.T, K)
print(X.T,Z)
```

```
[[1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0.]
 [1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1.]]
<NDArray 8x6 @cpu(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. 0.]]
<NDArray 8x5 @cpu(0)>
```

Learning a Kernel

In [8]:
```
conv2d = nn.Conv2D(1, kernel_size=(1, 2))
conv2d.initialize()

X = X.reshape((1, 1, 6, 8))
Y = Y.reshape((1, 1, 6, 7))

for i in range(10):
    with autograd.record():
        Y_hat = conv2d(X)
        l = (Y_hat - Y) ** 2
    l.backward()
    conv2d.weight.data()[:] -= 3e-2 * conv2d.weight.grad()
    if (i + 1) % 2 == 0:
        print('batch %d, loss %.3f' % (i + 1, l.sum().asscalar()))
```

```
batch 2, loss 4.949
batch 4, loss 0.831
batch 6, loss 0.140
batch 8, loss 0.024
batch 10, loss 0.004
```

Learned Kernel

In [9]:
```
conv2d.weight.data().reshape((1, 2))
```

Out[9]:
```
[[ 0.9895    -0.9873705]]
<NDArray 1x2 @cpu(0)>
```