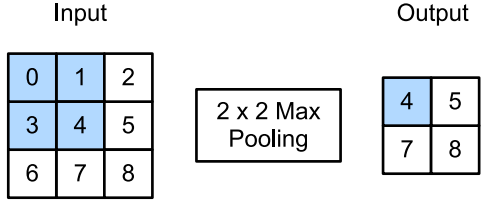
Maximum Pooling and Average Pooling



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

def pool2d(X, pool_size, mode='max'):
    p_h, p_w = pool_size
    Y = nd.zeros((X.shape[0] - p_h + 1, X.shape[1] - p_w + 1))
    for i in range(Y.shape[0]):
        for j in range(Y.shape[1]):
            if mode == 'max':
                Y[i, j] = X[i: i + p_h, j: j + p_w].max()
            elif mode == 'avg':
                Y[i, j] = X[i: i + p_h, j: j + p_w].mean()
    return Y
```

We can construct the input array `X` in the above diagram to validate the output of the two-dimensional maximum pooling layer.

In [2]:
```
X = nd.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
print(X)
print(pool2d(X, (2, 2)))
```

```
[[0. 1. 2.]
 [3. 4. 5.]
 [6. 7. 8.]]
<NDArray 3x3 @cpu(0)>

[[4. 5.]
 [7. 8.]]
<NDArray 2x2 @cpu(0)>
```

In [3]:
```
# Average pooling
pool2d(X, (2, 2), 'avg')
```

Out[3]:
```
[[2. 3.]
 [5. 6.]]
<NDArray 2x2 @cpu(0)>
```

Padding and Stride

In [4]:
```
X = nd.arange(16).reshape((1, 1, 4, 4))
X
```

Out[4]:
```
[[[[ 0.  1.  2.  3.]
   [ 4.  5.  6.  7.]
   [ 8.  9. 10. 11.]
   [12. 13. 14. 15.]]]]
<NDArray 1x1x4x4 @cpu(0)>
```

By default the stride for a window of `(3,3)` is `(3,3)`.

In [5]:
```
pool2d = nn.MaxPool2D(3)
pool2d(X)  # Because there are no model parameters in the pooling layer, we do not need to call the parameter initialization function.
```

Out[5]:
```
[[[[10.]]]]
<NDArray 1x1x1x1 @cpu(0)>
```

The stride and padding can be manually specified.

In [6]:
```
pool2d = nn.MaxPool2D(3, padding=1, strides=2)
pool2d(X)
```

Out[6]:
```
[[[[ 5.  7.]
   [13. 15.]]]]
<NDArray 1x1x2x2 @cpu(0)>
```

Arbitrary window

In [7]:
```
pool2d = nn.MaxPool2D((2, 3), padding=(1, 2), strides=(2, 3))
pool2d(X)
```

Out[7]:
```
[[[[ 0.  3.]
   [ 8. 11.]
   [12. 15.]]]]
<NDArray 1x1x3x2 @cpu(0)>
```

Multiple Channels

Pooling is applied per channel.

In [8]:
```
X = nd.concat(X, X + 1, dim=1)
X
```

Out[8]:
```
[[[[ 0.  1.  2.  3.]
   [ 4.  5.  6.  7.]
   [ 8.  9. 10. 11.]
   [12. 13. 14. 15.]]

  [[ 1.  2.  3.  4.]
   [ 5.  6.  7.  8.]
   [ 9. 10. 11. 12.]
   [13. 14. 15. 16.]]]]
<NDArray 1x2x4x4 @cpu(0)>
```

As we can see, the number of output channels is still 2 after pooling.

In [9]:
```
pool2d = nn.MaxPool2D(3, padding=1, strides=2)
pool2d(X)
```

Out[9]:
```
[[[[ 5.  7.]
   [13. 15.]]

  [[ 6.  8.]
   [14. 16.]]]]
<NDArray 1x2x2x2 @cpu(0)>
```