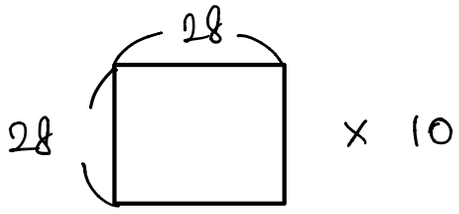
CH 7.3 – 7.4

7.4 합성곱 계층 구현하기

1. 4차원 배열



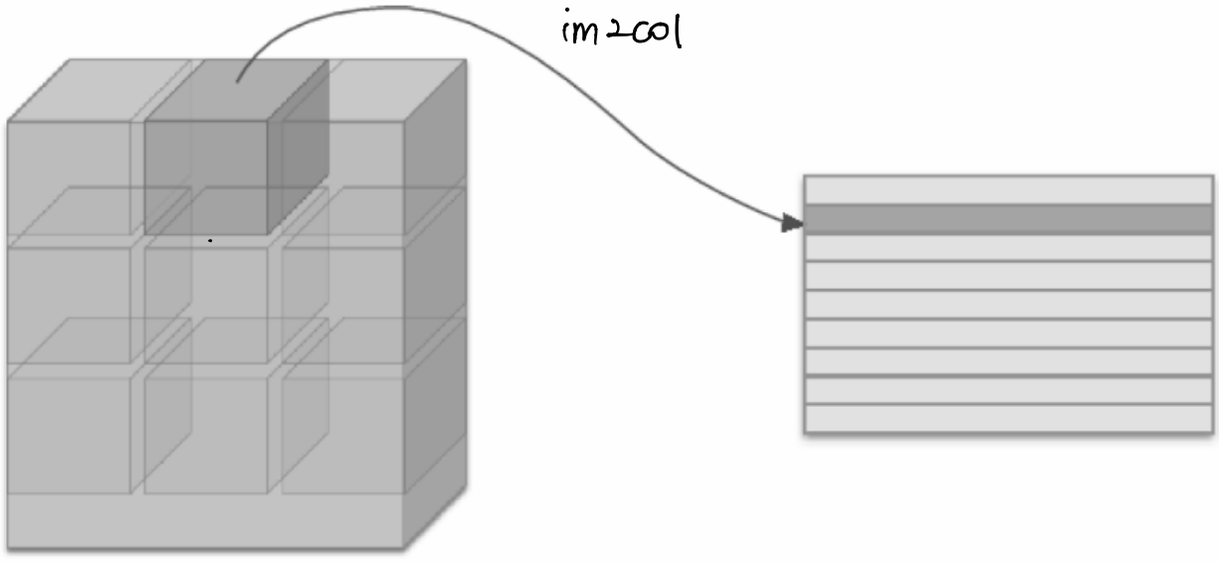

```
[2.53907354e-01, 7.09628682e-01, 6.47883286e-01, 1.43634698e-01,
 7.10550601e-01, 9.81212250e-01, 6.11336610e-01, 8.19469295e-01,
 5.80179369e-01, 5.44957208e-01, 2.56307613e-01, 8.39176912e-01,
 9.26080843e-01, 9.90383382e-01, 3.41432638e-03, 3.00087121e-01,
 2.31937711e-01, 2.20708452e-02, 1.26138458e-01, 4.63908301e-01,
 3.51725432e-01, 3.13205191e-01, 2.84630587e-01, 6.95508196e-01,
 2.36684182e-01, 4.49725554e-01, 5.88274702e-01, 7.57963491e-01]
```

→ 28개

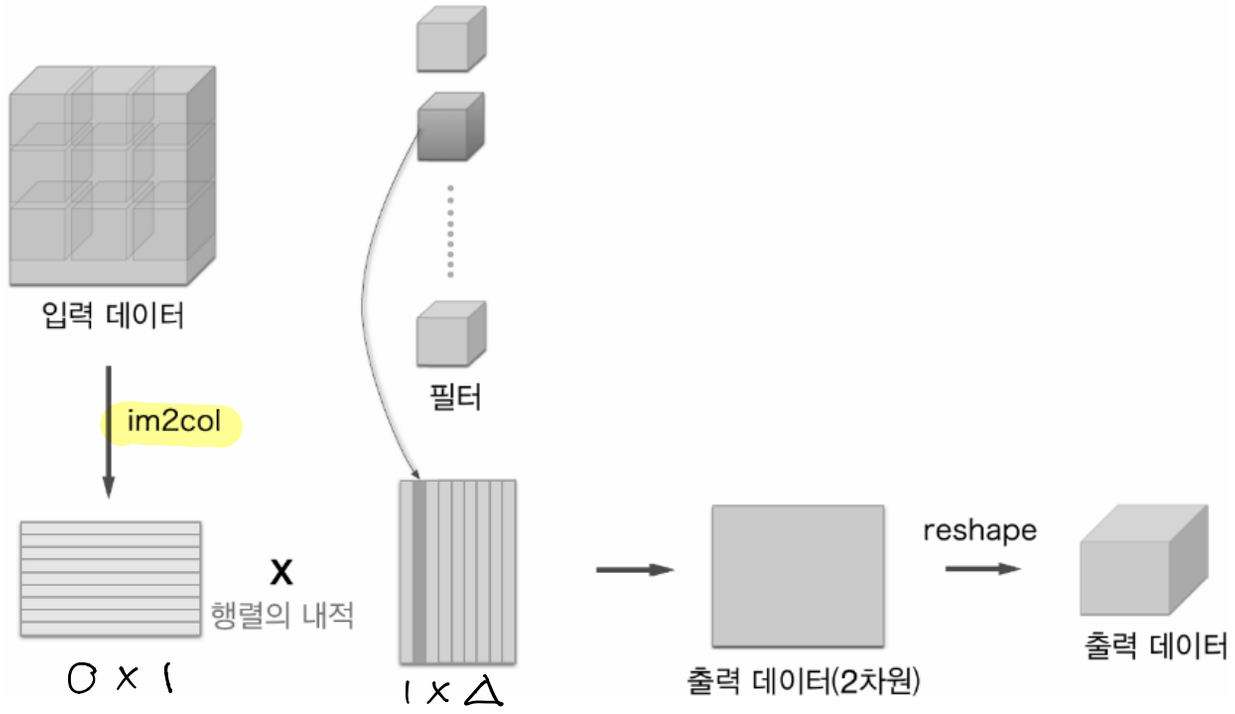
2. im2col로 데이터 전개하기

- 합성곱 연산을 for문으로 사용하면 성능이 떨어진다. (넘파이에서 원소 접근 시 for문을 사용하지 않는 것이 바람직하다.

- im2col은 입력 데이터를 필터링하기 좋게 펼치는 함수, 3차원 입력데이터에 im2col 적용 시 2차원 행렬로 바뀐다. (실제 데이터 수까지 포함하면 입력 데이터 4차원)



위 그림에서는 스트라이드를 크게 잡아 필터 영역이 겹치지 않게 하였다. 하지만 실제로 영역이 겹치게 하는 경우가 많기 때문에 im2col로 전개 후 원소 수가 원래 블록 원소 수보다 많아진다.



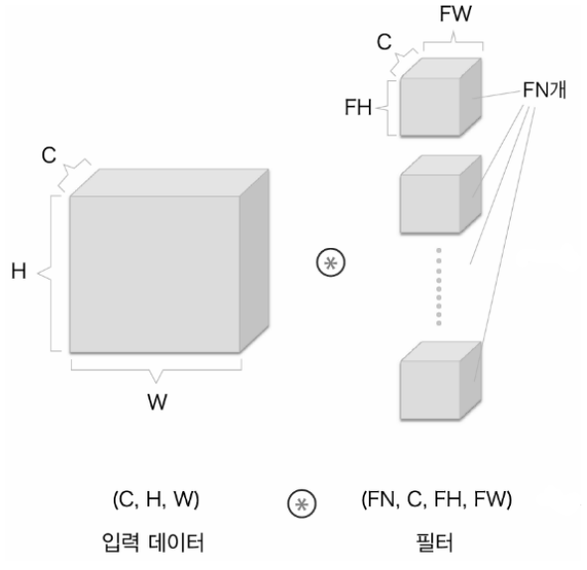
합성곱 필터를 세로 1열로 전개하고, im2col로 전개한 입력 데이터와 내적 (행렬 곱)을 수행한다. 그렇게 출력된 2차원 데이터를 4차원으로 reshape한다.

3. 합성곱 계층 구현하기

- im2col 함수 : common/util.py

- im2col(input_data, filter_h, filter_w, stride=1, pad=0)

* input_data : (데이터 수, 채널 수, 높이, 너비)

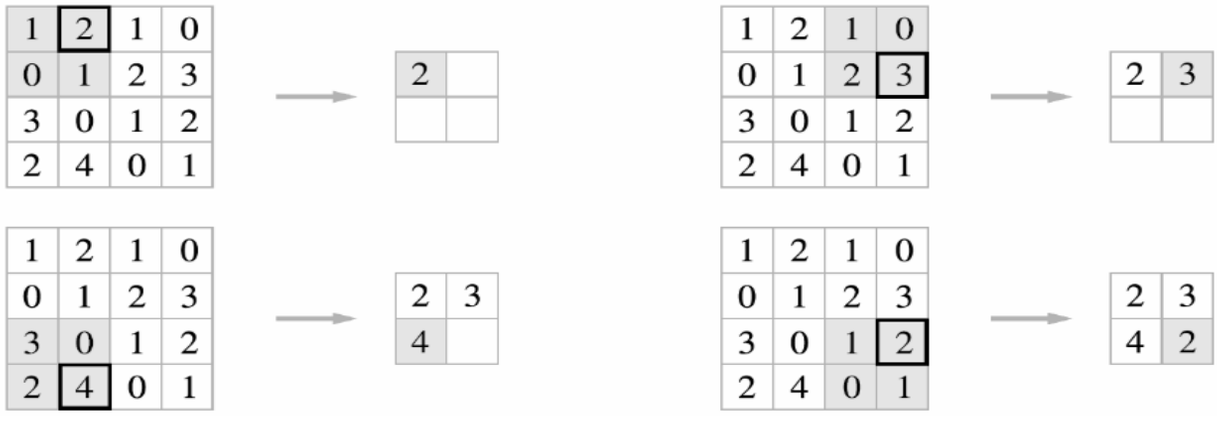


```python
class Convolution:
    def __init__(self, W, b, stride=1, pad=0):
        self.W = W
        self.b = b
        self.stride = stride
        self.pad = pad

    def forward (self, x):
        FN,C,FH,FW = self.W.shape  # 필터(가중치)는 4차원
                                   # FN:필터 개수, C:채널, FH:필터 높이, FW:필터 너비
        N,C,H,W = x.shape # 입력 데이터 4차원(데이터 수, 채널 수, 높이, 너비)
        out_h = int(1+(H+2*self.pad-FH)/self.stride) # 출력 크기 (높이)
        out_w = int(1+(W+2*self.pad-FW)/self.stride) # 출력 크기 (너비)

        col = im2col(x,FH,FW,self.stride,self.pad) # 입력 데이터 전개
        col_W = self.W.reshape(FN,-1).T #필터 전개
                                        # 두 번째 인수 -1, 다차원 배열 원소 수가 변환 후에도 똑같도록 묶어줌
        out= np.dot(col, col_W) + self.b

        out=out.reshape(N,out_h,out_w,-1).transpose(0,3,1,2) # transpose는 다차원 배열의 축 순서를 바꿔주는 함수

        return out
```

7.3 풀링 계층

1. 풀링 (Pooling) ?

: 세로, 가로 방향의 공간을 줄이는 연산



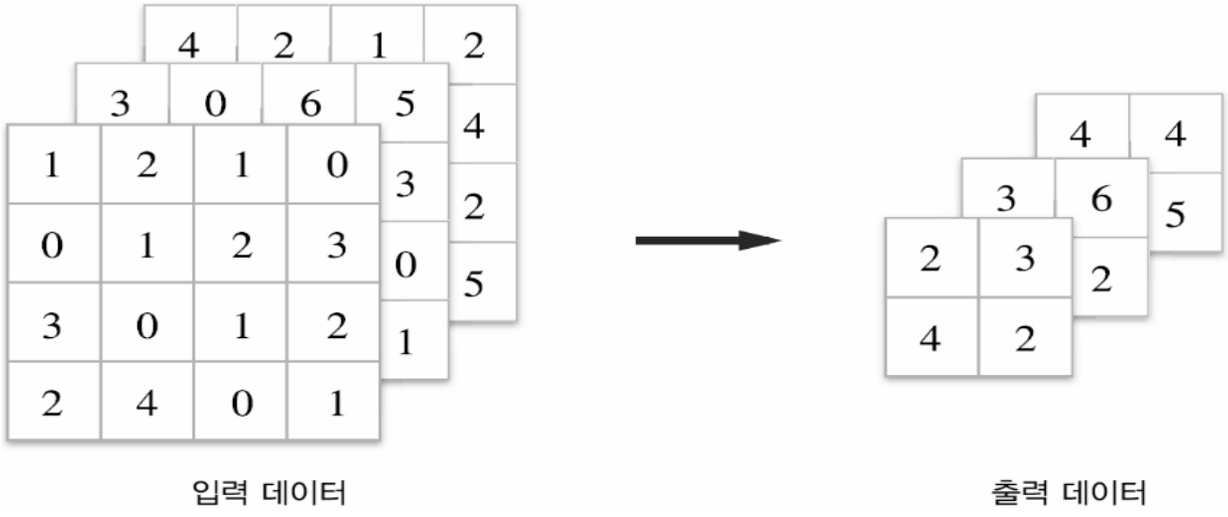
<최대 풀링, 스트라이드 2>

* 스트라이드 : 필터를 적용하는 위치 간격, 일반적으로, 풀링 윈도우 크기와 스트라이드는 같은 값으로 설정한다. 최대 풀링 외에도 평균 풀링 등이 있지만, 이미지 인식 분야에서는 주로 최대 풀링을 사용한다.

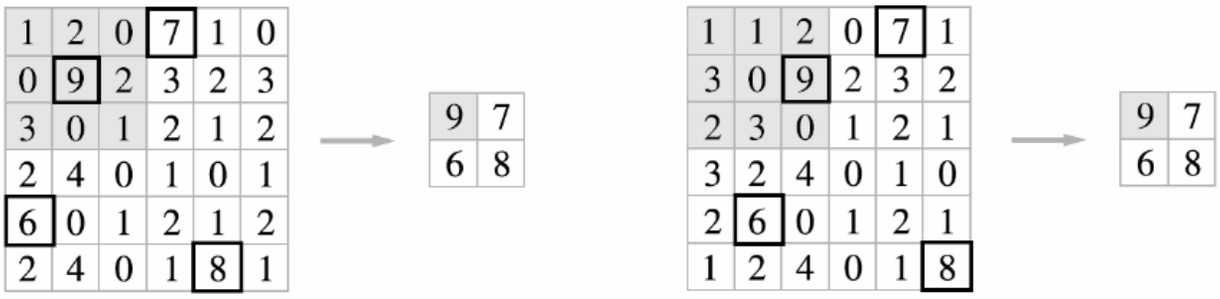
2. 풀링 계층의 특징

- 학습해야 할 매개변수가 없다.

- 채널 수가 변하지 않는다.

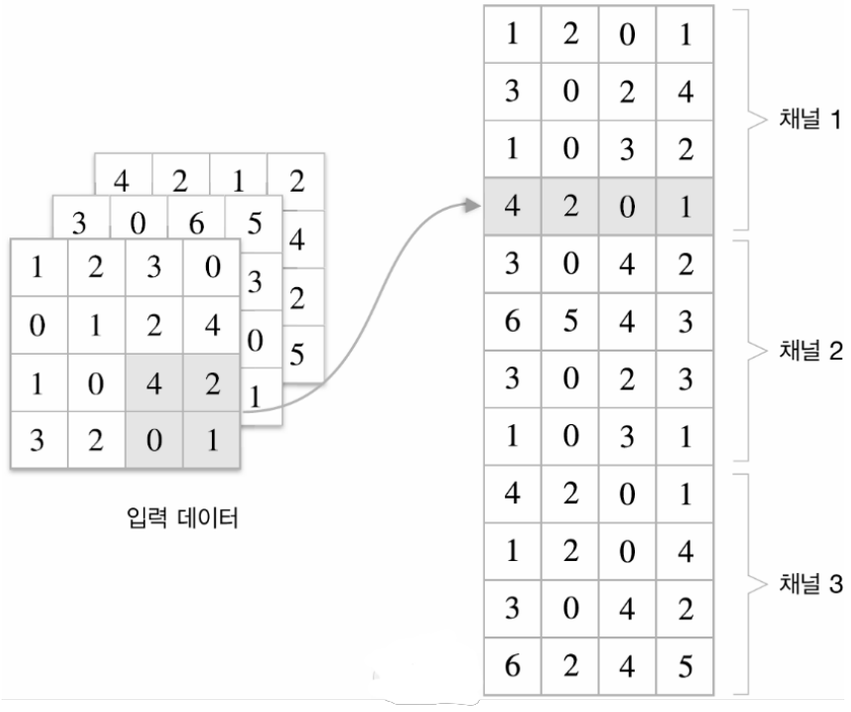


- 입력 변화에 영향을 적게 받는다.

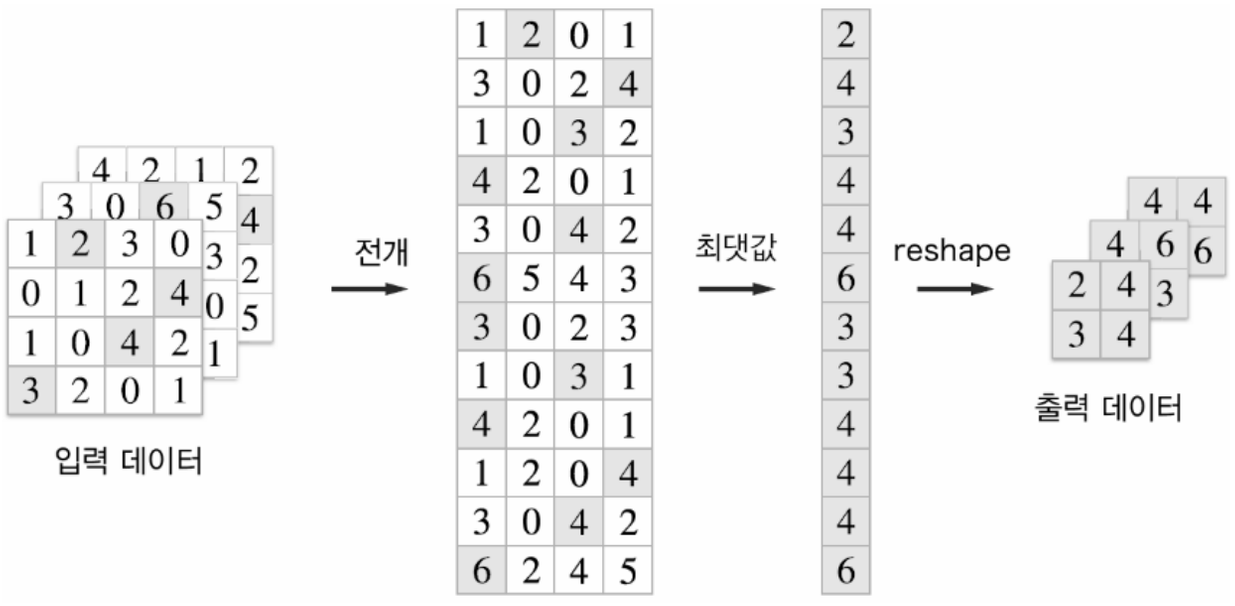


7.4. 풀링 계층 구현하기

: 마찬가지로 입력 데이터를 im2col처럼 전개 후, 풀링을 수행한다.



<im2col을 이용한 전개>



```python
class Pooling:
    def __init__(self, pool_h, pool_w, stride=1, pad=0):
        self.pool_h = pool_h
        self.pool_w = pool_w
        self.stride = stride
        self.pad = pad

    def forward(self, x):
        N,C,H,W = x.shape # 입력 데이터
        out_h = int(1+(H-self.pool_h)/self.stride) # 출력 크기
        out_w = int(1+(W-self.pool_w)/self.stride) # 출력 크기

        # 전개
        col1 = im2col(x,self.pool_h, self.pool_w, self.stride, self.pad)
        col2 = col1.reshape(-1, self.pool_h*self.pool_w)

        # 최대값
        out = np.max(col, axis=1)

        # reshape
        out = out.reshape(N, out_h, out_w, C).transpose(0,3,1,2)

        return out

pooling = Pooling(2,2)
x=np.random.rand(1,3,4,4)
y = pooling.forward(x)

'''
pooling = Pooling(2,2)
x=np.random.rand(1,3,4,4)

col1.shape # (9,12)
col2.shape # (27,4)
'''
```