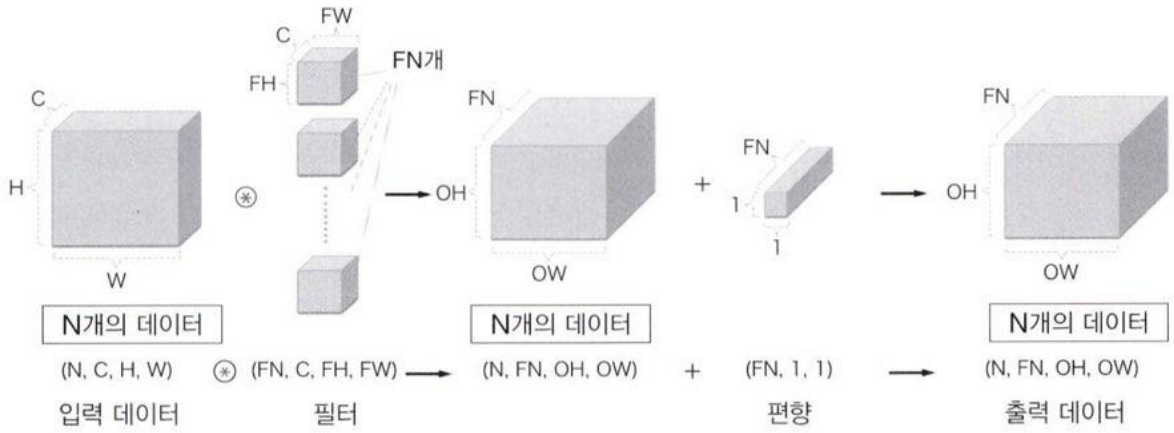
Ch07 Convolution Network

각 계층마다 데이터의 형태가 어떻게 변화하는지 자세히 살펴보자.

Convoltuion layer

```
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
        N, C, H, W = x.shape
        out_h = int(1 + (H + 2*self.pad - FH) / self.stride)
        out_w = int(1 + (W + 2*self.pad - FW) / self.stride)

        col = im2col(x, FH, FW, self.stride, self.pad)
        col_W = self.W.reshape(FN, -1).T # 필터 전개
        out = np.dot(col, col_W) + self.b

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

        return out
```



im2col

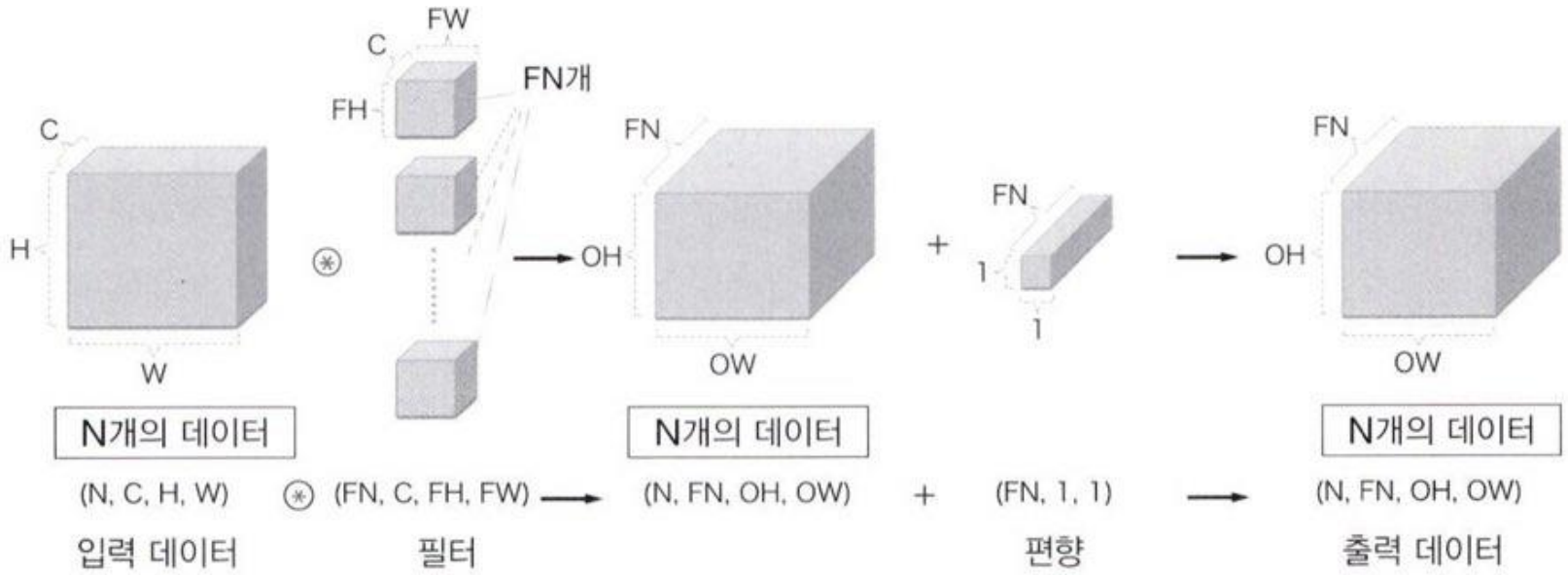
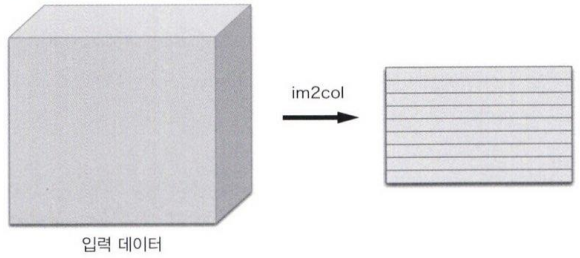


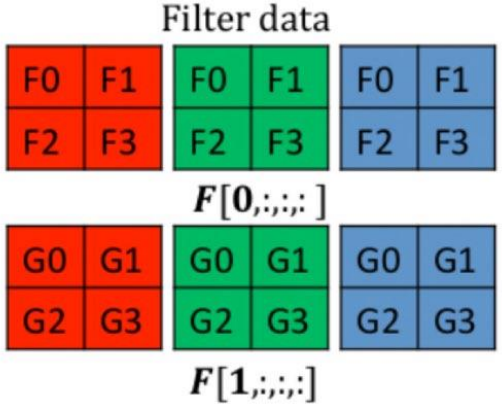
그림 7-17 (대략적인) im2col의 동작



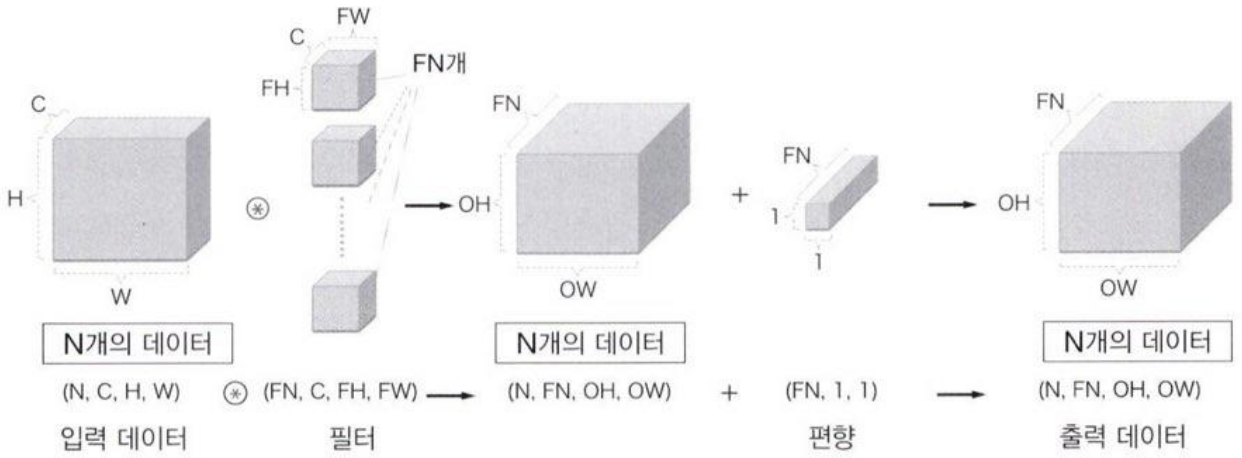
필터링하기 좋게 입력 데이터를
2차원 행렬로 전개

Convoltuion layer

Input data : (1,2,3,3)=(N, C, H, W)
Filter : (2, 3, 2, 2)=(FN, C, FH, FW)
Output : (1, 2, 2, 2)=(N, FN, CH, OW)



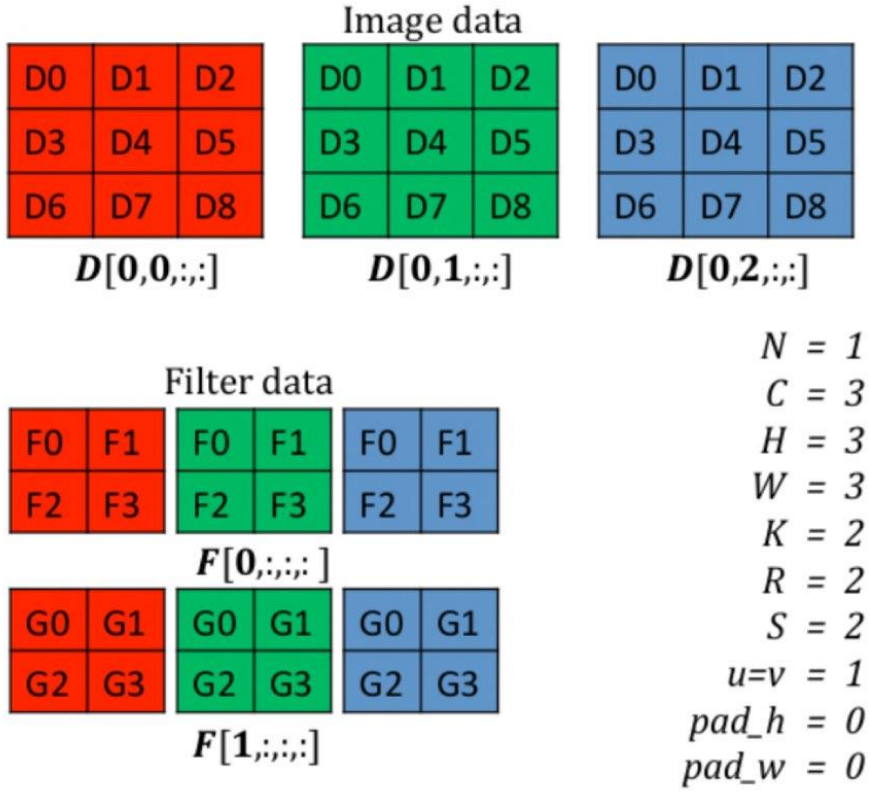
$N = 1$
$C = 3$
$H = 3$
$W = 3$
$K = 2$
$R = 2$
$S = 2$
$u=v = 1$
$pad\_h = 0$
$pad\_w = 0$




```
col = im2col(x, FH, FW, self.stride, self.pad)
col_W = self.W.reshape(FN, -1).T # 필터 전개
out = np.dot(col, col_W) + self.b
```

Convoltuion layer

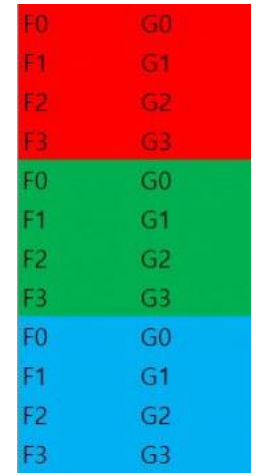
Input data : (1,2,3,3)=(N, C, H, W)
Filter : (2, 3, 2, 2)=(FN, C, FH, FW)
Output : (1, 2, 2, 2)=(N, FN, CH, OW)



$N = 1$
$C = 3$
$H = 3$
$W = 3$
$K = 2$
$R = 2$
$S = 2$
$u=v = 1$
$pad\_h = 0$
$pad\_w = 0$

```
col = im2col(x, FH, FW, self.stride, self.pad)
col_W = self.W.reshape(FN, -1).T # 필터 전개
out = np.dot(col, col_W) + self.b
```

FN의 개수



Filter size*channel의 개수=4*3

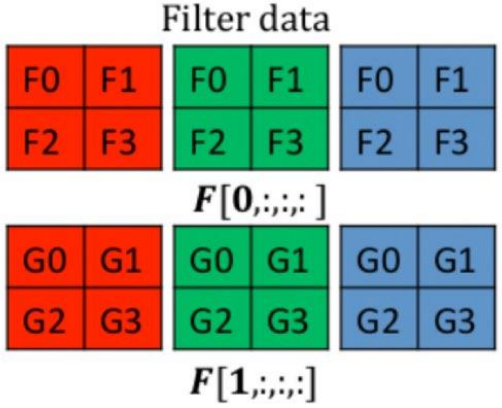
(3-2+1)*(3-2+1)

4*12와 12*2가 np.dot을 만나서 4*2가 됨
=>(N, FN, out_h, out_w)=(1,2,2,2)

```
out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)
```

Convoltuion layer, N>=2

Input data : (2,2,3,3)=(N, C, H, W)
Filter : (2, 3, 2, 2)=(FN, C, FH, FW)
Output : (1, 2, 2, 2)=(N, FN, CH, OW)

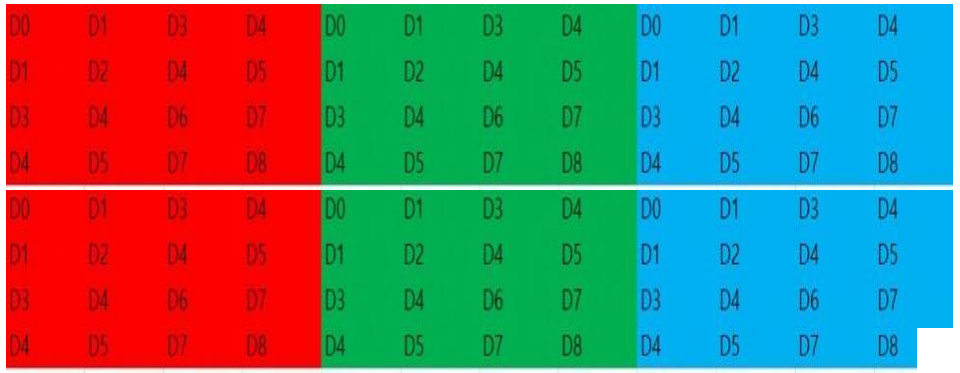


$N = 1$
$C = 3$
$H = 3$
$W = 3$
$K = 2$
$R = 2$
$S = 2$
$u=v = 1$
$pad\_h = 0$
$pad\_w = 0$

```
col = im2col(x, FH, FW, self.stride, self.pad)
col_W = self.W.reshape(FN, -1).T # 필터 전개
out = np.dot(col, col_W) + self.b
```

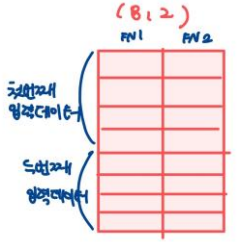
FN의 개수

Filter size*channel의 개수=4*3



(3-2+1)*(3-2+1)*2

Np.dot

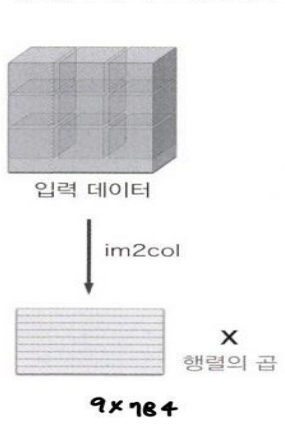


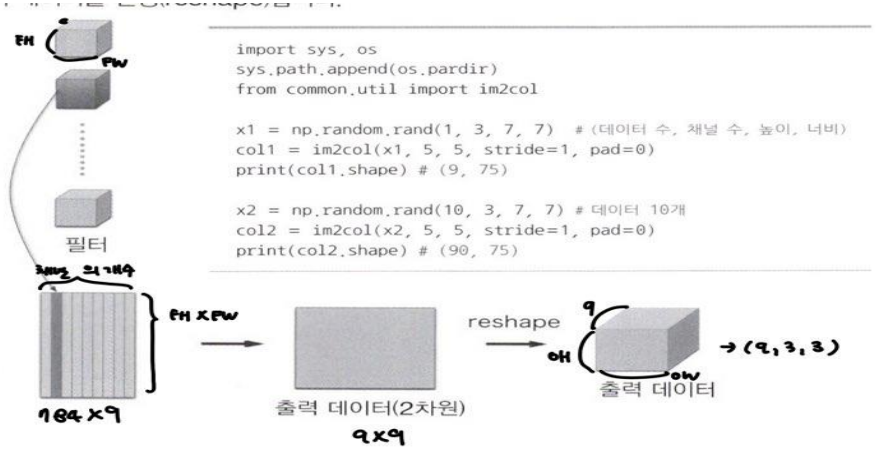
8*12와 12*2가 np.dot을 만나서 8*2가 됨
=>(N, FN, out_h, out_w)=(2,2,2,2)

```
out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)
```

Pooling layer

```
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
        out_h = int(1 + (H - self.pool_h) / self.stride)
        out_w = int(1 + (W - self.pool_w) / self.stride)

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

        # 최댓값 (2)
        out = np.max(col, axis=1)

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

        return out
```






```
import sys, os
sys.path.append(os.pardir)
from common.util import im2col

x1 = np.random.rand(1, 3, 7, 7)  # (데이터 수, 채널 수, 높이, 너비)
col1 = im2col(x1, 5, 5, stride=1, pad=0)
print(col1.shape) # (9, 75)

x2 = np.random.rand(10, 3, 7, 7) # 데이터 10개
col2 = im2col(x2, 5, 5, stride=1, pad=0)
print(col2.shape) # (90, 75)
```

Pooling layer

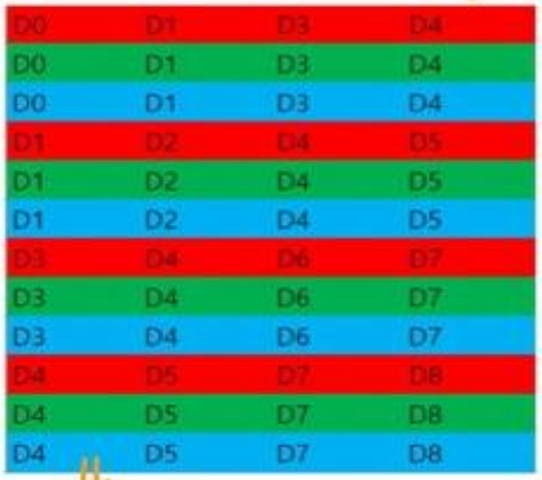
Pool 계층 (데이터가 1개일 때)

input data : (1, 3, 3, 3)

pool_h * pool_w = (2, 2)

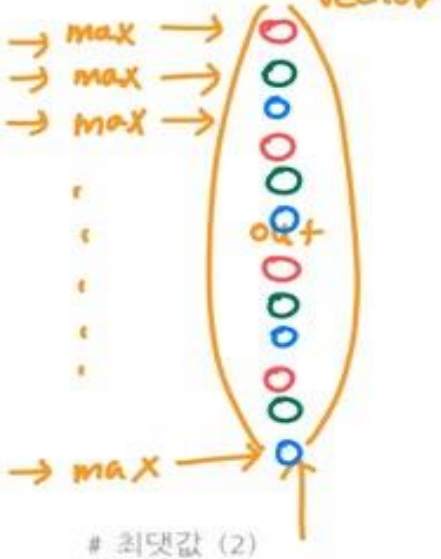
```
# 전개 (1)
col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
col = col.reshape(-1, self.pool_h*self.pool_w)
```

→ (4, 12)



(12, 4)

(12, pool_h * pool_w)

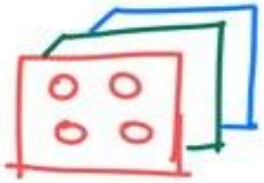



```
# 최댓값 (2)
out = np.max(col, axis=1)
```

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

⇒ (N, C, out_h, out_w)

⇒ (1, 3, 2, 2)

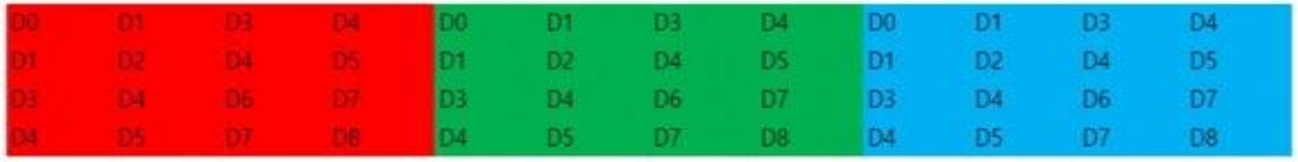


Pooling layer

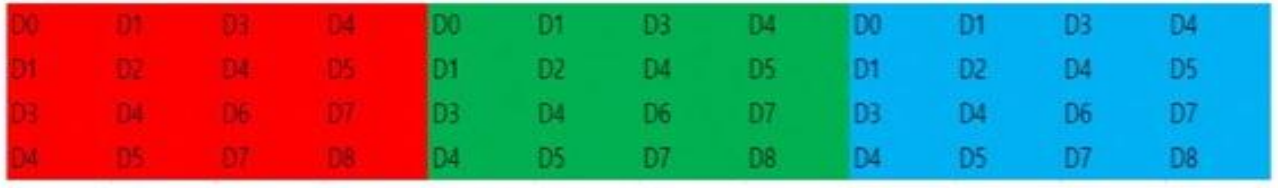
Pool 계층 (데이터가 2개 이상일 때)

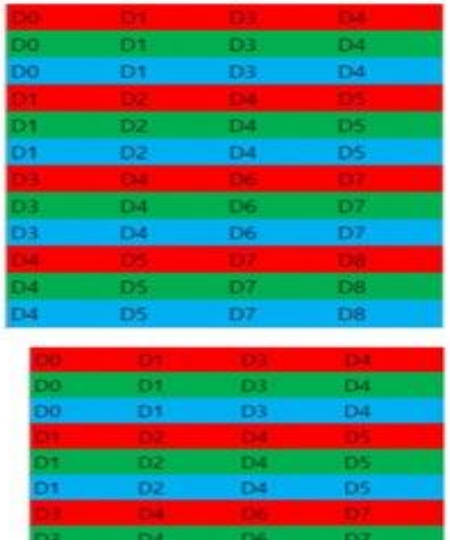
input data : (2,3,3,3)

pool_h * pool_w = (2,2)



→ (8,12)





→

vector (24,1)

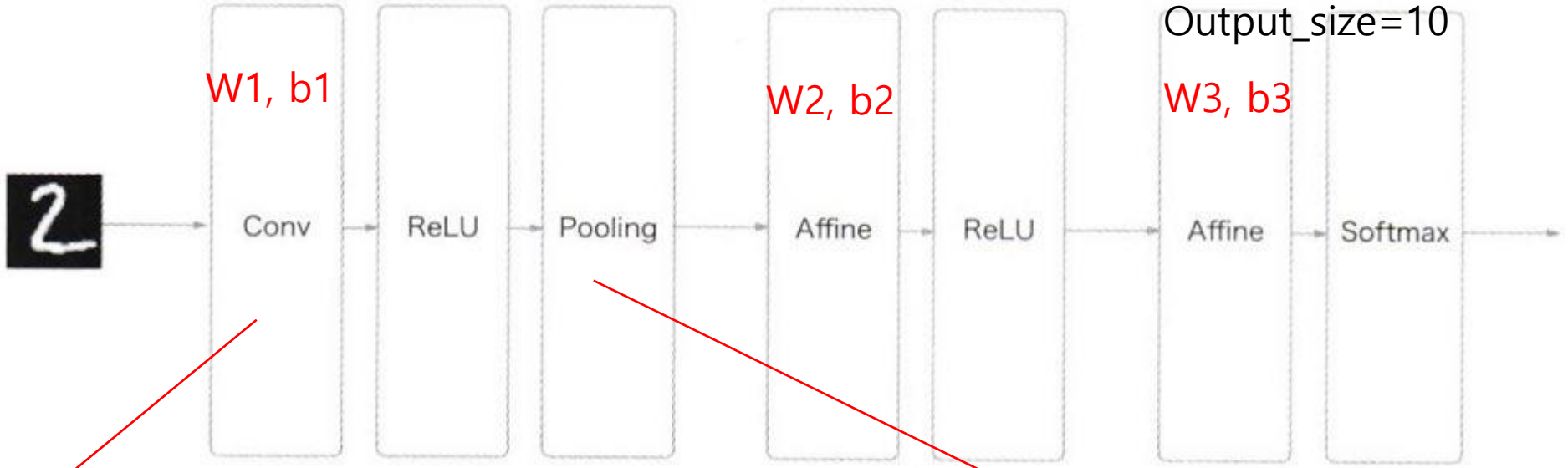
(24,4)

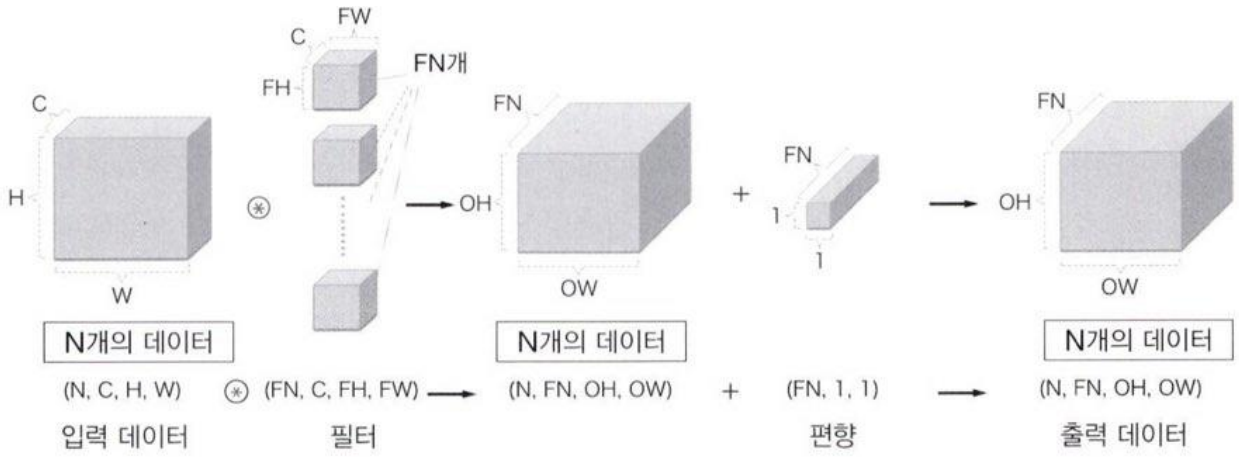
⇒ (N, C, out_h, out_w)

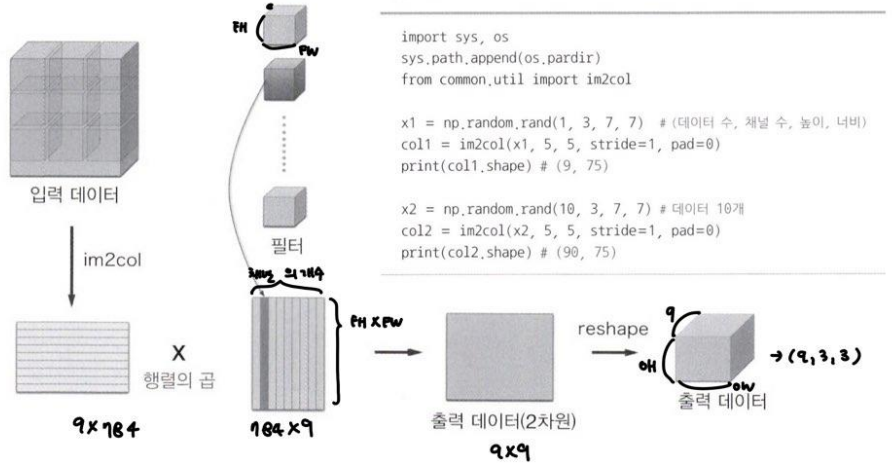
(2,3,2,2)

MNIST 실습

Input : (N, 1, 28, 28)=(N, C, W, H)
Filter: (30, 1, 5, 5)=(FN, C, FW, FH)
Pad=0, stride=1
Hidden_size=100
Output_size=10

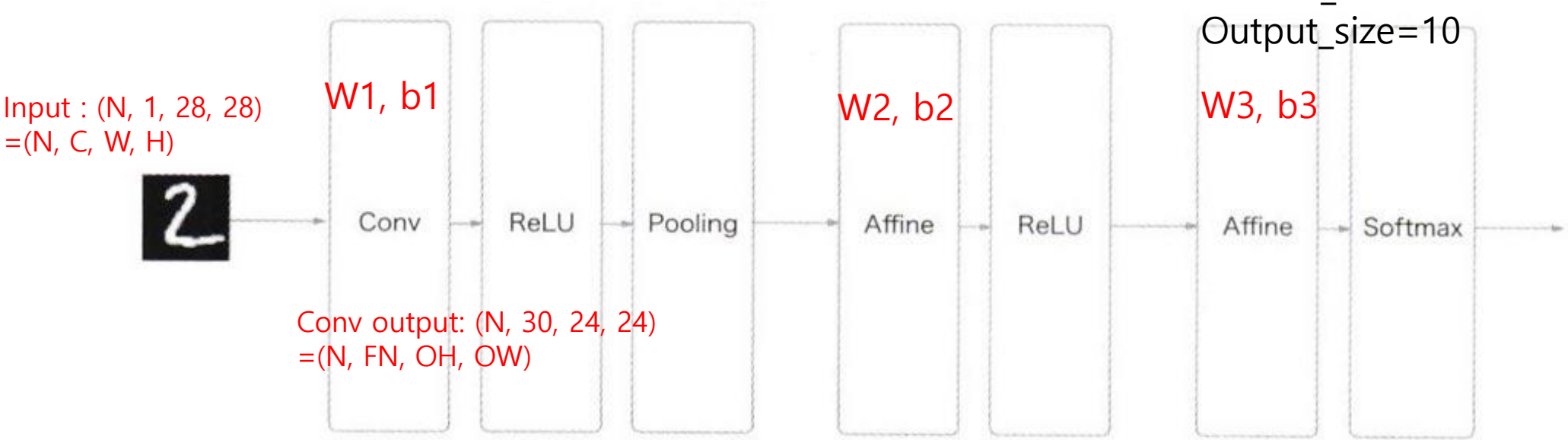




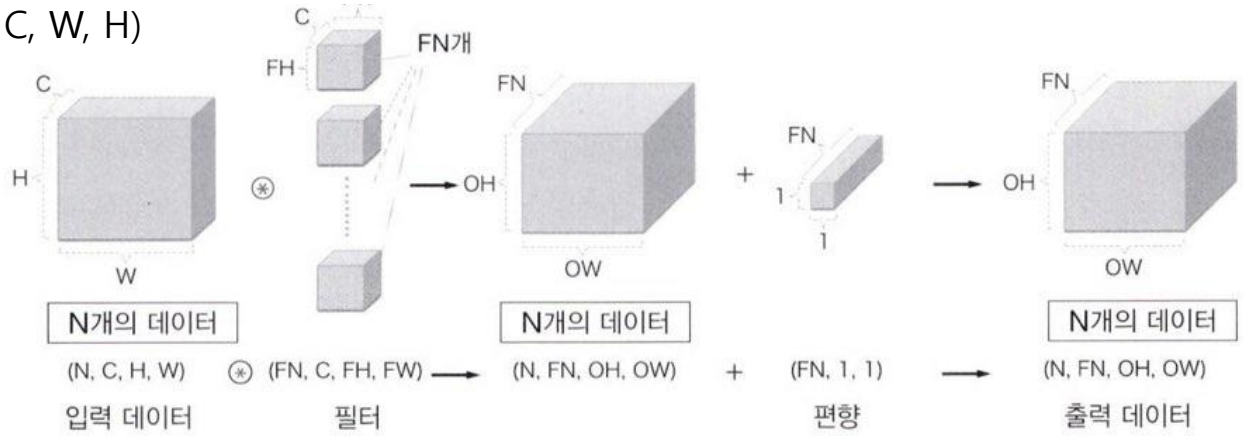


MNIST 실습

Input : (N, 1, 28, 28)=(N, C, W, H)
Filter: (30, 1, 5, 5)=(FN, C, FW, FH)
Pad=0, stride=1
Hidden_size=100
Output_size=10



Input : (N, 1, 28, 28)=(N, C, W, H)



$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$

Filter: (30, 1, 5, 5)=(FN, C, FW, FH)

Conv output: (N, 30, 24, 24)=(N, FN, OH, OW)

MNIST 실습

Input : (N, 1, 28, 28)
=(N, C, W, H)

Conv

W1, b1

ReLU input/output:
(N, 30, 24, 24)
=(N, FN, OH, OW)

ReLU

Pooling

Conv output: (N, 30, 24, 24)
=(N, FN, OH, OW)

Pooling output : (N, 1, 12, 12)
=(N, C, out_h, out_w)

Affine

W2, b2

ReLU

Affine

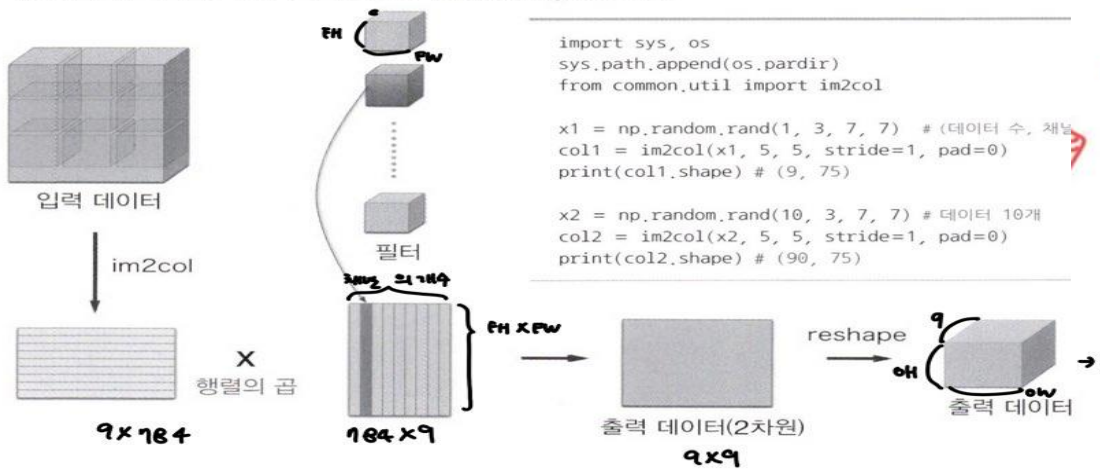
W3, b3

Softmax

Input : (N, 1, 28, 28)=(N, C, W, H)
Filter: (30, 1, 5, 5)=(FN, C, FW, FH)
Pad=0, stride=1
Hidden_size=100
Output_size=10

여기서 주의할 것!
C가 Convolution layer에서 연산에 의해 사라지고, FN이 pooling에서 마치 Channel과 같이 쓰임!!

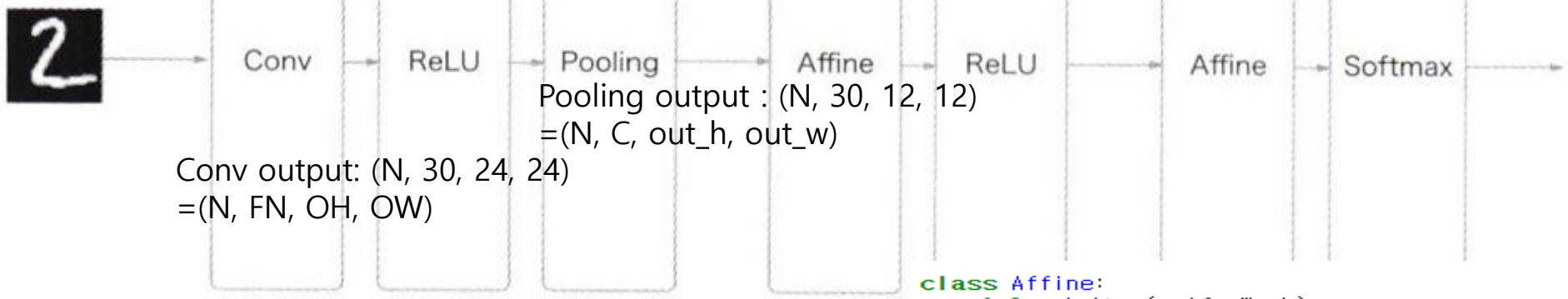
ReLU output=pooling input:
(N, 30, 24, 24)
=(N, FN, OH, OW)



Pooling output : (N, 30, 12, 12)
=(N, C=FN, out_h, out_w)

MNIST 실습

Input : (N, 1, 28, 28)
=(N, C, W, H)



ReLU input/output:
(N, 30, 24, 24)
=(N, FN, OH, OW)

W1, b1

W2, b2

W3, b3

Pooling output : (N, 30, 12, 12)
=(N, C, out_h, out_w)

Conv output: (N, 30, 24, 24)
=(N, FN, OH, OW)

Input : (N, 1, 28, 28)=(N, C, W, H)
Filter: (30, 1, 5, 5)=(FN, C, FW, FH)
Pad=0, stride=1
Hidden_size=100
Output_size=10

```
class Affine:
    def __init__(self, W, b):
        self.W = W
        self.b = b

        self.x = None
        self.original_x_shape = None
        # 가중치와 편향 매개변수의 미분
        self.dW = None
        self.db = None

    def forward(self, x):
        # 텐서 대응
        self.original_x_shape = x.shape
        x = x.reshape(x.shape[0], -1)
        self.x = x

        out = np.dot(self.x, self.W) + self.b

        return out
```

Pooling의 output은 4차원이지만
Affine은 2차원 곱을 해주는데 과연 어떤 트릭이 숨어있는가?

->(N, c, out_h, out_w)=(N, c*out_h*out_w) 로 2차원 행렬로 바꿈

MNIST 실습

Input : (N, 1, 28, 28)=(N, C, W, H)
Filter: (30, 1, 5, 5)=(FN, C, FW, FH)
Pad=0, stride=1
Hidden_size=100
Output_size=10