

# Ch07 Convolution Network

각 계층마다 데이터의 형태가 어떻게 변화하는지 자세히 살펴보자.

# Convolution 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)
```

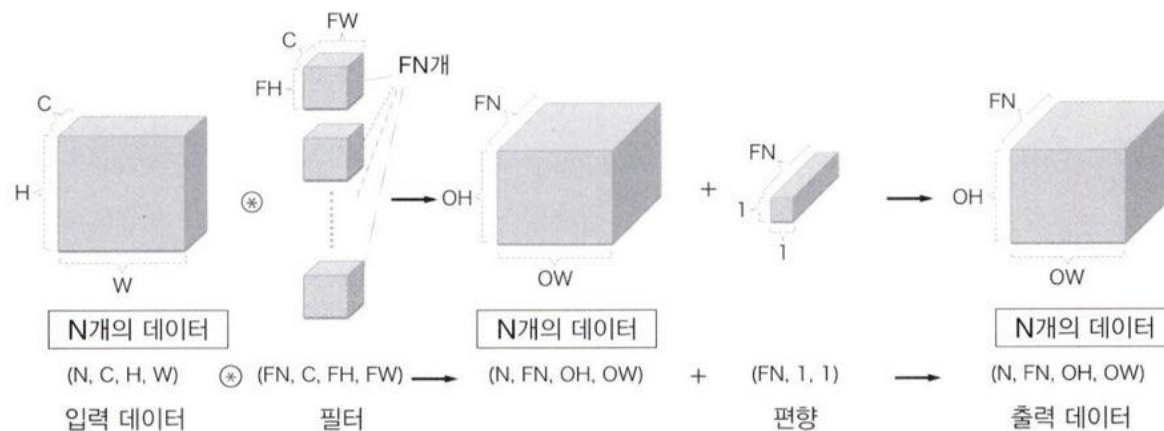
```
        col = im2col(x, FH, FW, self.stride, self.pad)
```

```
        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

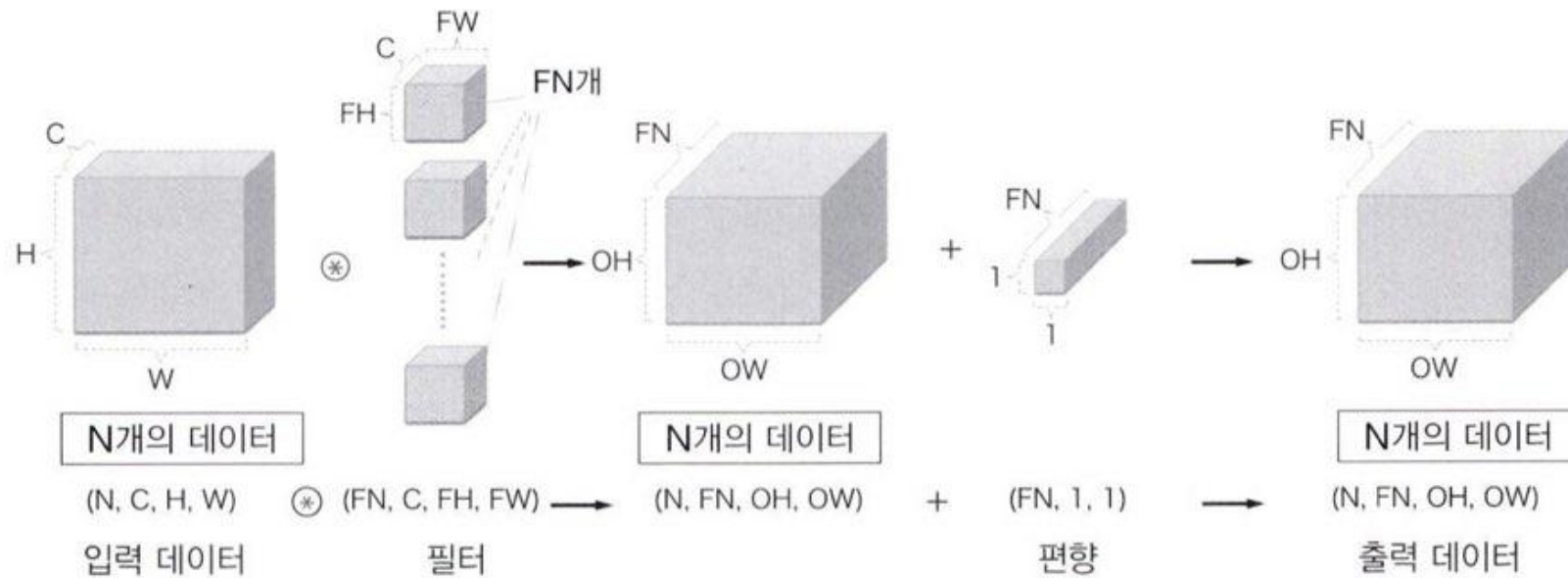
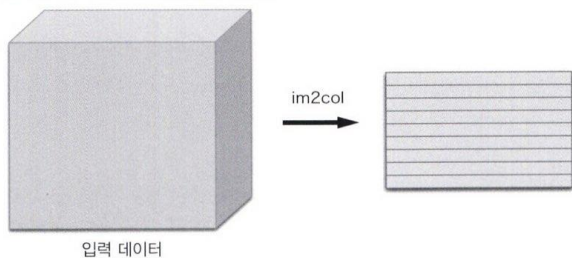


그림 7-17 (대략적인) im2col의 동작



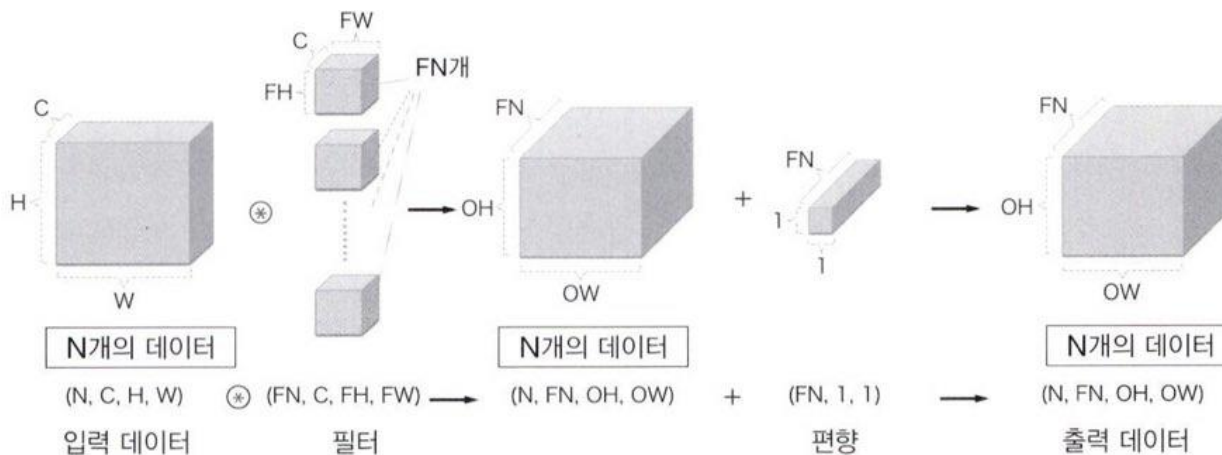
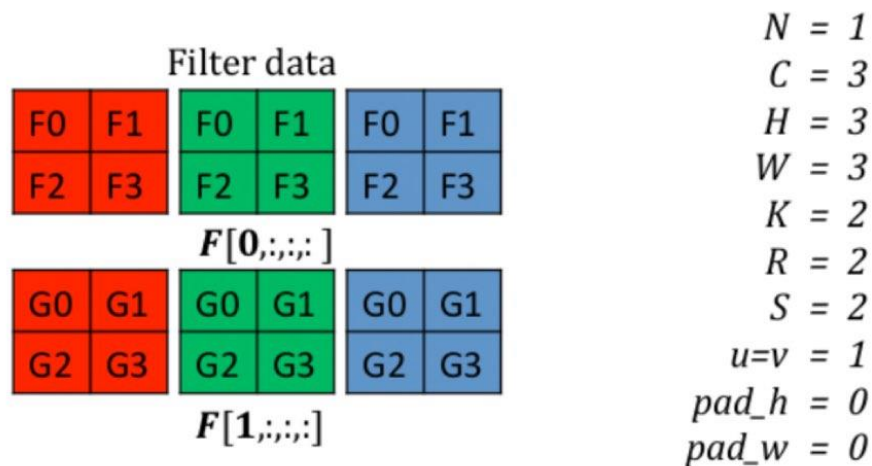
필터링하기 좋게 입력 데이터를  
2차원 행렬로 전개

# Convolution 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)



```
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
```

# Convolution 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)

Image data

D0	D1	D2
D3	D4	D5
D6	D7	D8

$D[0,0,:,:]$

D0	D1	D2
D3	D4	D5
D6	D7	D8

$D[0,1,:,:]$

D0	D1	D2
D3	D4	D5
D6	D7	D8

$D[0,2,:,:]$

Filter data

F0	F1
F2	F3

$F[0,:,:,:]$

G0	G1
G2	G3

$F[1,:,:,:]$

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

```
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

D0	D1	D3	D4	D0	D1	D3	D4	D0	D1	D3	D4
D1	D2	D4	D5	D1	D2	D4	D5	D1	D2	D4	D5
D3	D4	D6	D7	D3	D4	D6	D7	D3	D4	D6	D7
D4	D5	D7	D8	D4	D5	D7	D8	D4	D5	D7	D8

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

F0	G0
F1	G1
F2	G2
F3	G3
F0	G0
F1	G1
F2	G2
F3	G3
F0	G0
F1	G1
F2	G2
F3	G3

4\*12와 12\*2가 np.dot을 만나서 4\*2가 됨  
 $\Rightarrow (N, FN, out\_h, out\_w)=(1,2,2,2)$

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

# Convolution layer, $N \geq 2$

Input data :  $(2, 2, 3, 3) = (N, C, H, W)$

Filter :  $(2, 3, 2, 2) = (FN, C, FH, FW)$

Output :  $(1, 2, 2, 2) = (N, FN, CH, OW)$

Image data

D0	D1	D2
D3	D4	D5
D6	D7	D8

$D[0,0,:,:]$        $D[0,1,:,:]$        $D[0,2,:,:]$

Filter data

F0	F1
F2	F3

$F[0,:,:,:]$

G0	G1
G2	G3

$F[1,:,:,:]$

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

```
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
```

Filter size\*channel의 개수=4\*3

D0	D1	D3	D4
D1	D2	D4	D5
D3	D4	D6	D7
D4	D5	D7	D8

D0	D1	D3	D4
D1	D2	D4	D5
D3	D4	D6	D7
D4	D5	D7	D8

D0	D1	D3	D4
D1	D2	D4	D5
D3	D4	D6	D7
D4	D5	D7	D8

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

8\*12와 12\*2가 np.dot을 만나서 8\*2가 됨  
 $\Rightarrow (N, FN, out\_h, out\_w) = (2, 2, 2, 2)$

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

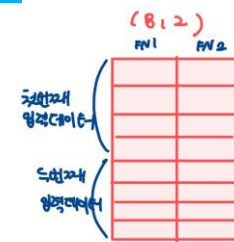
FN의 개수

F0	G0
F1	G1
F2	G2
F3	G3

F0	G0
F1	G1
F2	G2
F3	G3

F0	G0
F1	G1
F2	G2
F3	G3

Np.dot





# 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)

D0	D1	D3	D4	D0	D1	D3	D4	D0	D1	D3	D4
D1	D2	D4	D5	D1	D2	D4	D5	D1	D2	D4	D5
D3	D4	D6	D7	D3	D4	D6	D7	D3	D4	D6	D7
D4	D5	D7	D8	D4	D5	D7	D8	D4	D5	D7	D8

→ (4, 12)

D0	D1	D3	D4
D0	D1	D3	D4
D0	D1	D3	D4
D1	D2	D4	D5
D1	D2	D4	D5
D1	D2	D4	D5
D3	D4	D6	D7
D3	D4	D6	D7
D3	D4	D6	D7
D4	D5	D7	D8
D4	D5	D7	D8
D4	D5	D7	D8

(12, 4)

(12, pool-h \* pool-w)

→ max

→ max

→ max

⋮

⋮

⋮

→ max

⋮

⋮

# 최댓값 (2)

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

vector



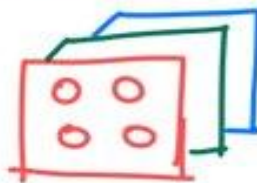
# 성형 (3)

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)

D0	D1	D3	D4
D1	D2	D4	D5
D3	D4	D6	D7
D4	D5	D7	D8

D0	D1	D3	D4
D1	D2	D4	D5
D3	D4	D6	D7
D4	D5	D7	D8

→ (8, 12)

D0	D1	D3	D4
D0	D1	D3	D4
D0	D1	D3	D4
D1	D2	D4	D5
D1	D2	D4	D5
D1	D2	D4	D5
D3	D4	D6	D7
D3	D4	D6	D7
D3	D4	D6	D7
D4	D5	D7	D8
D4	D5	D7	D8
D4	D5	D7	D8

→

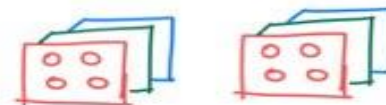
vector  
(24, 1)

out

(24, 4)

⇒ (N, C, out-h, out-w)

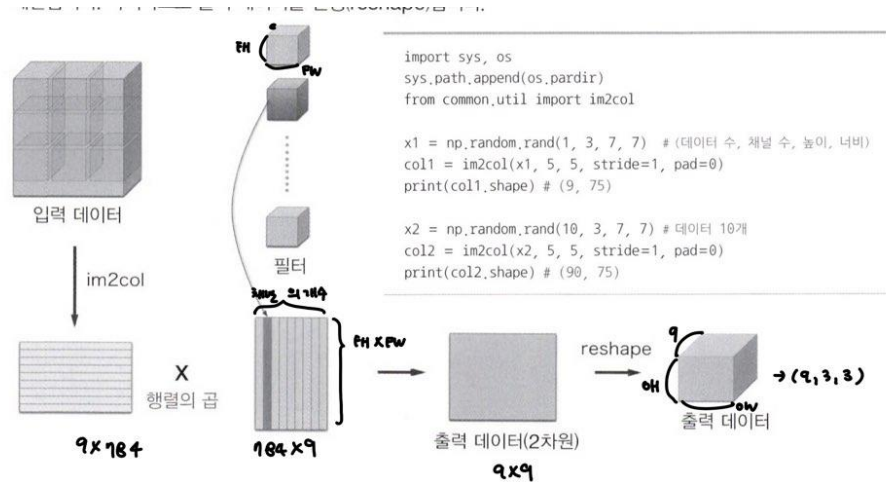
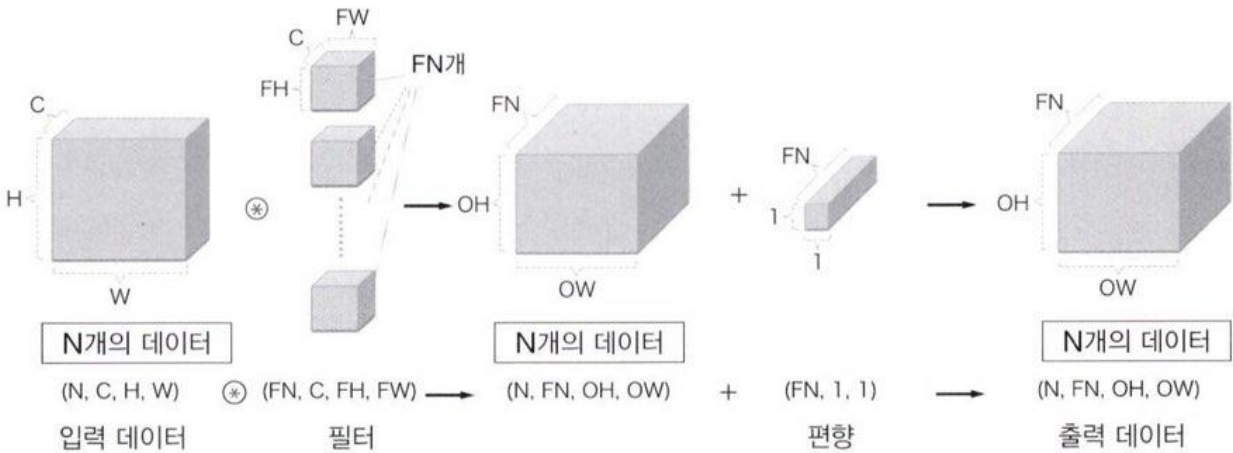
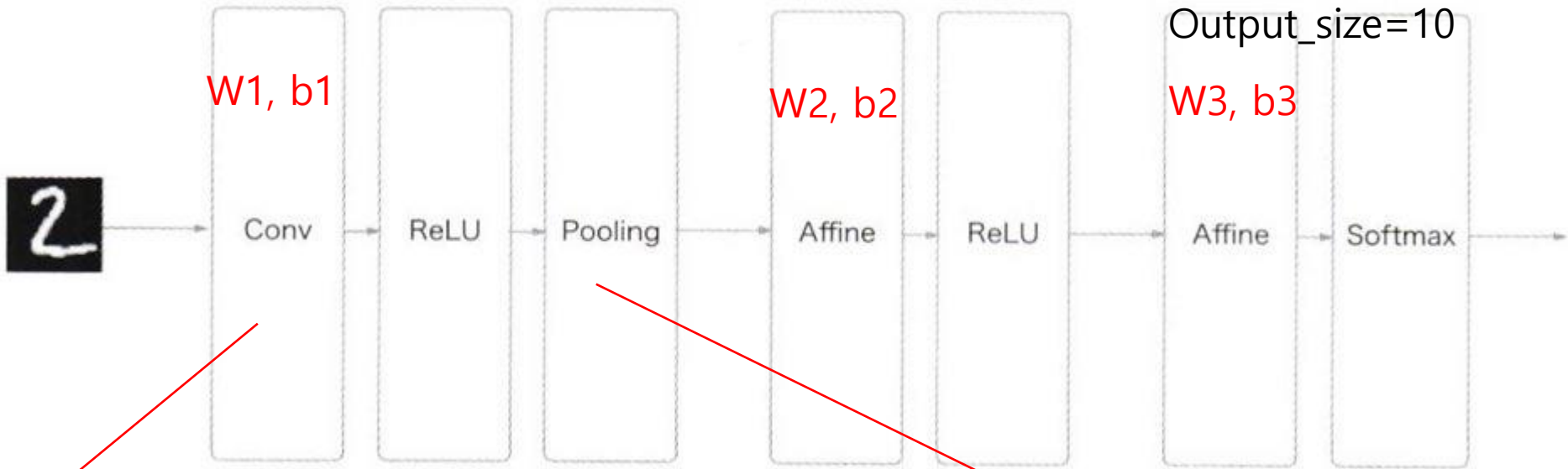
(2, 3, 2, 2)





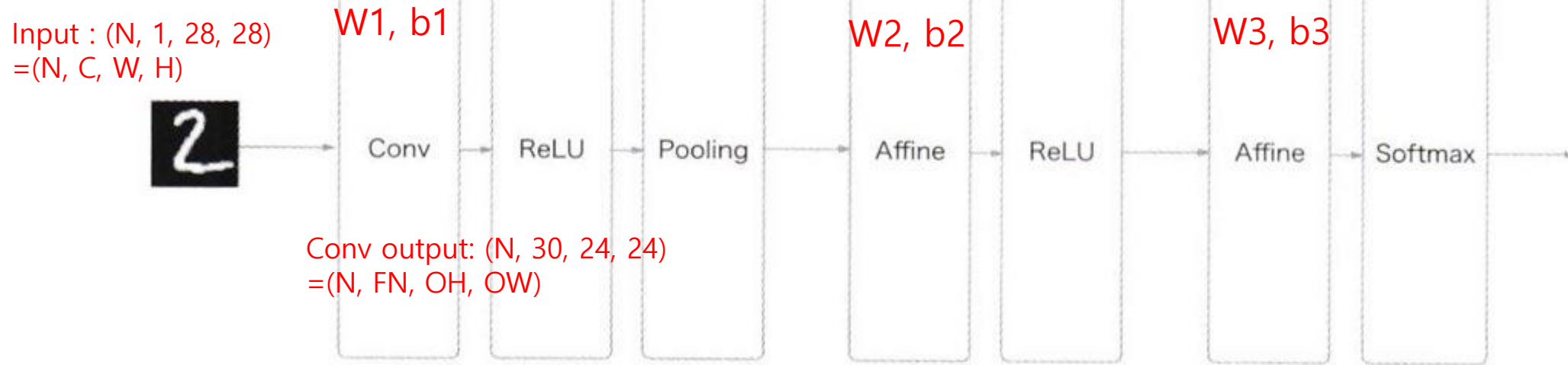
# MNIST 실습

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

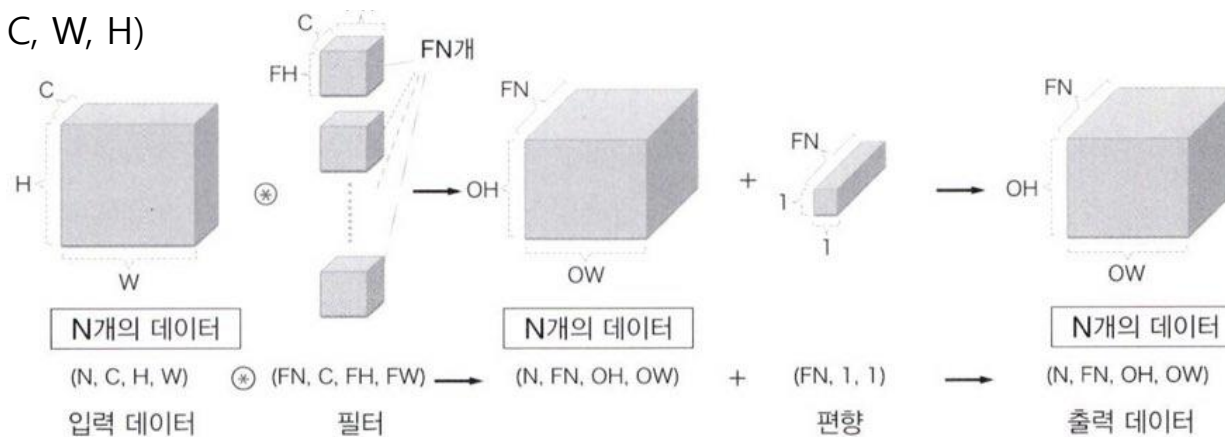


# MNIST 실습

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)



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

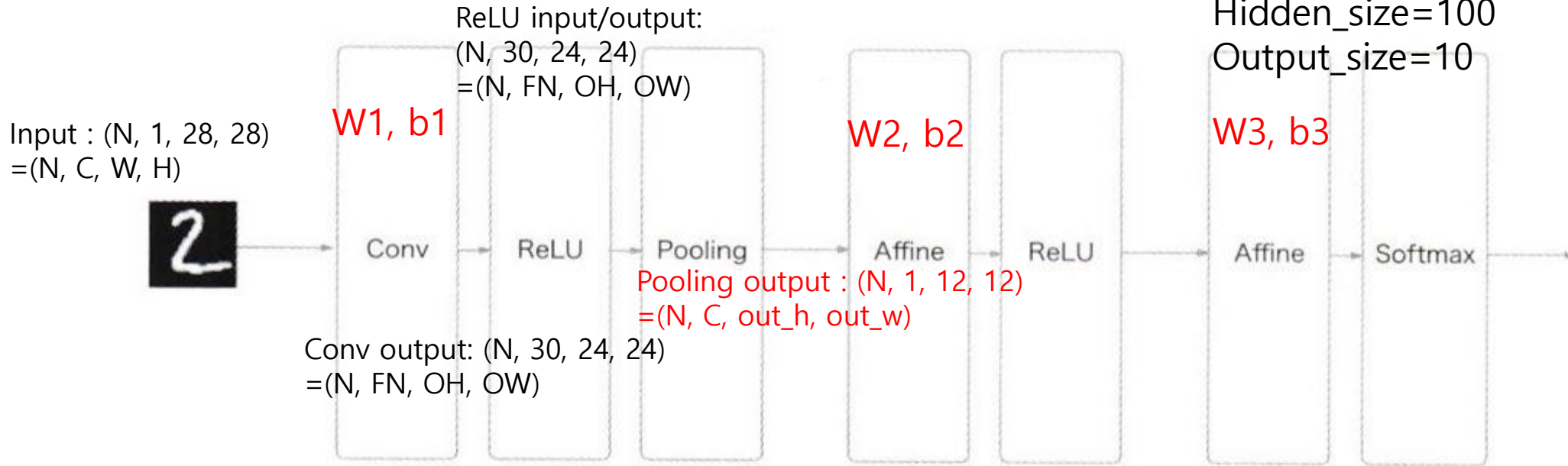
Conv output: (N, 30, 24, 24)=(N, FN, OH, OW)

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

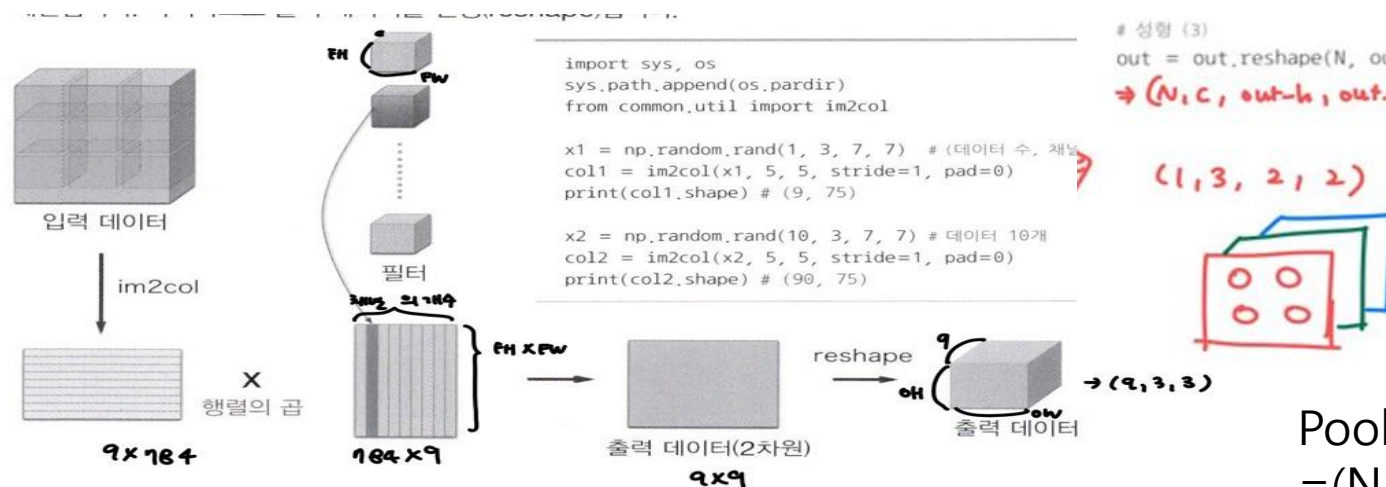
$$OW = \frac{W + 2P - FW}{S} + 1$$

# 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



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



Pooling output : (N, 1, 12, 12)  
 =(N, C, out\_h, out\_w)

# MNIST 실습

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



W1, b1

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

Conv

ReLU

Pooling

Pooling output : (N, 1, 12, 12)  
=(N, C, out\_h, out\_w)

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

W2, b2

Affine

ReLU

W3, b3

Affine

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

```
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)

