

# CNN 기초

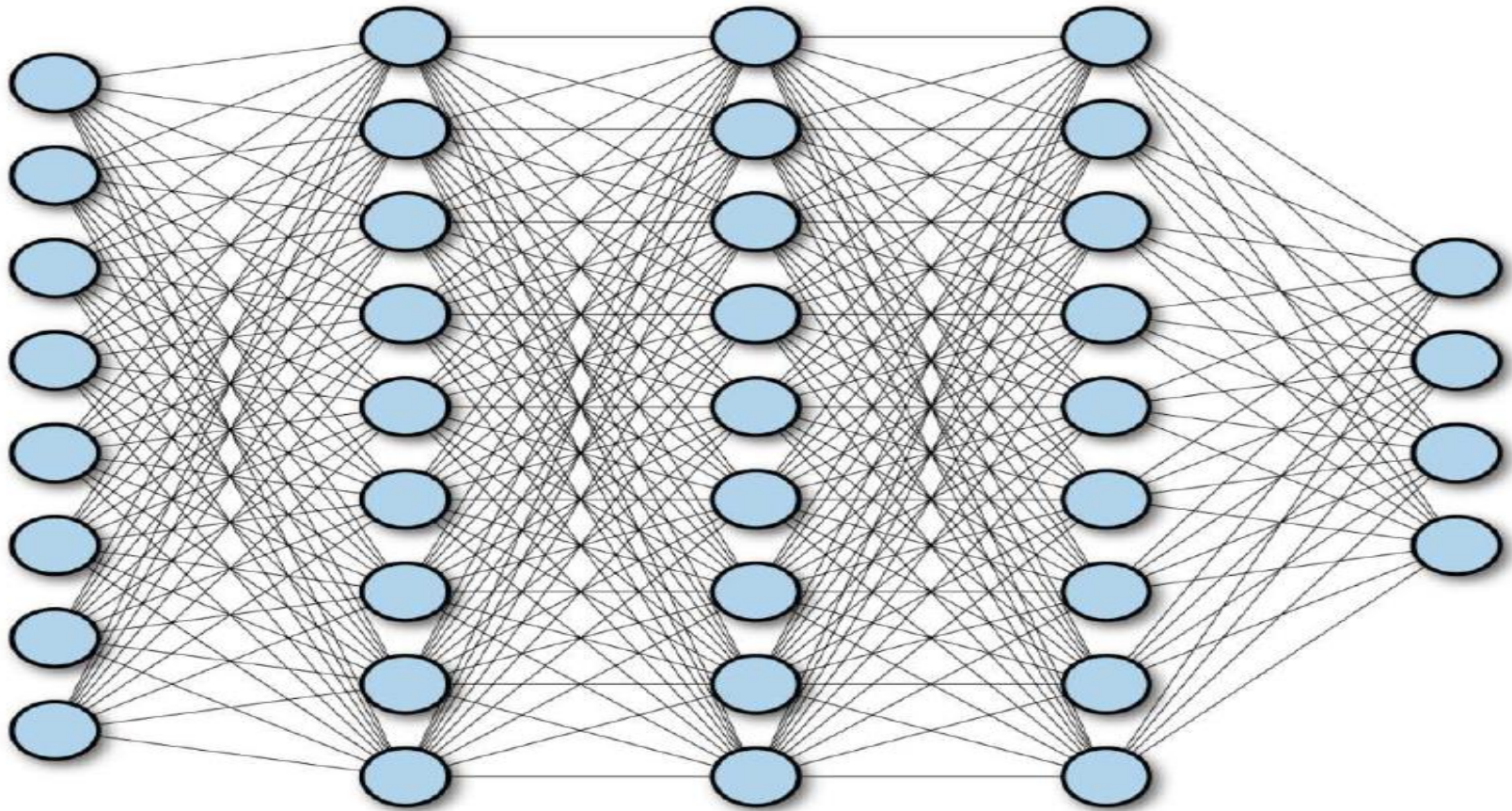
*컴퓨터 비전을 중심으로*

*Dec 2020*

## 합성곱 신경망 기초

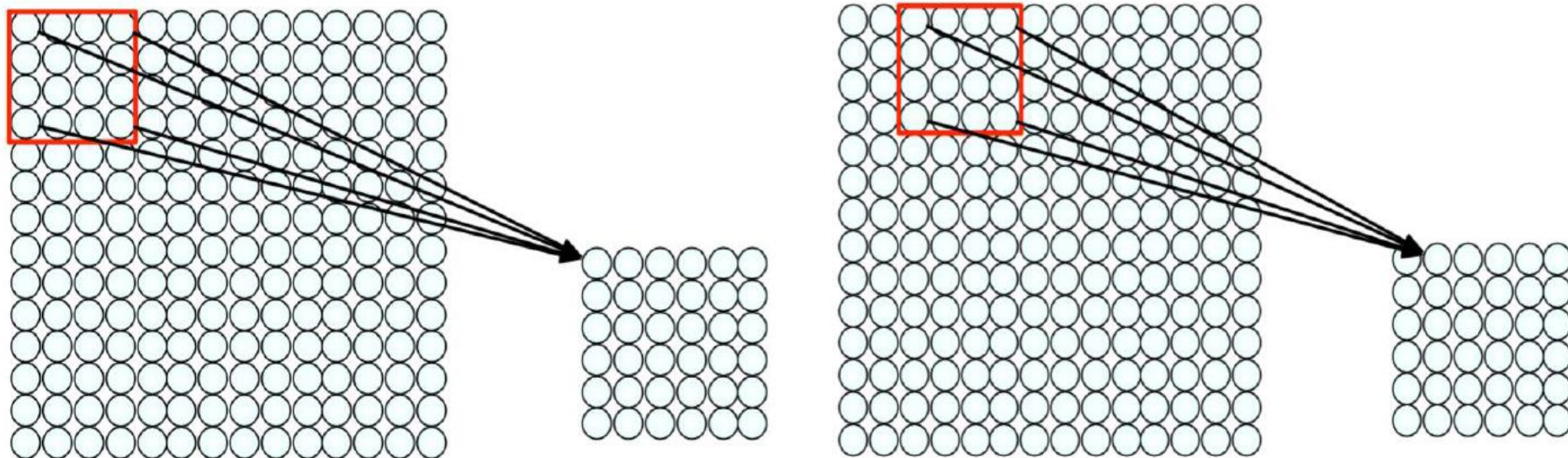
# 전결합 신경망

- 전결합 신경망 (Fully Connected: FC)



# 합성곱층의 이해

- 합성곱을 통한 공간적 특성추출



1) Apply a set of weights – a filter – to extract **local features**

2) Use **multiple filters** to extract different features

3) **Spatially share** parameters of each filter

# 합성곱층의 이해

- 합성곱이란?

1 <small>×1</small>	1 <small>×0</small>	1 <small>×1</small>	0	0
0 <small>×0</small>	1 <small>×1</small>	1 <small>×0</small>	1	0
0 <small>×1</small>	0 <small>×0</small>	1 <small>×1</small>	1	1
0	0	1	1	0
0	1	1	0	0



1	0	1
0	1	0
1	0	1

filter



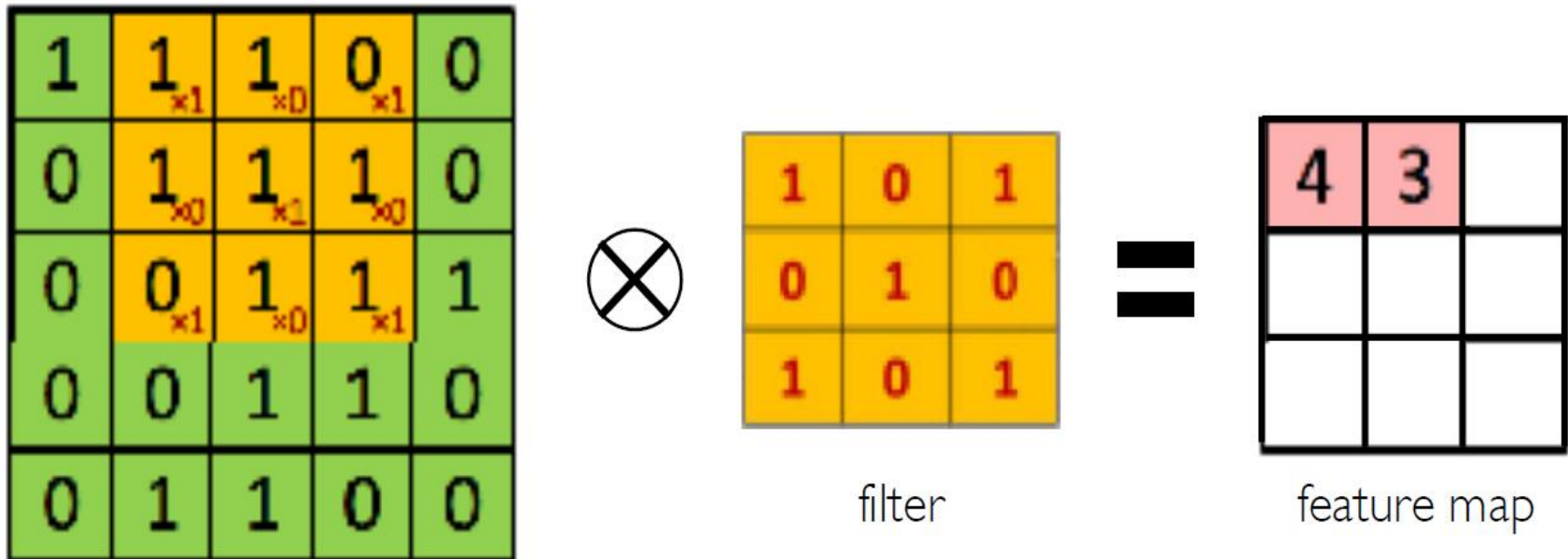
4		

feature map



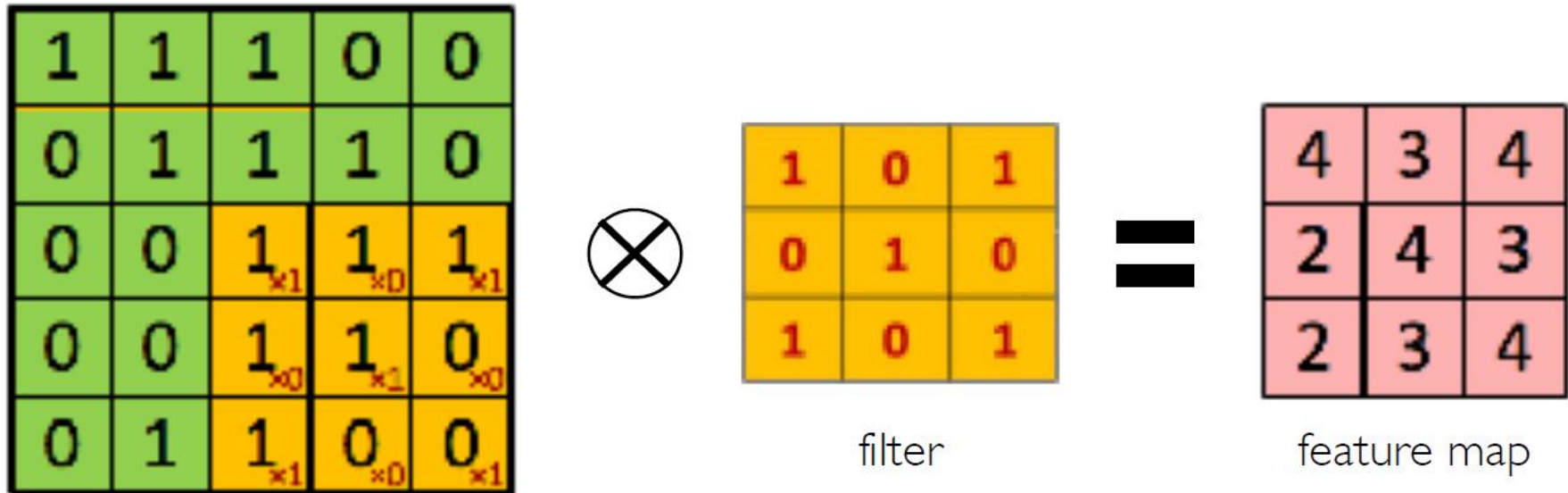
# 합성곱층의 이해

- 합성곱이란?



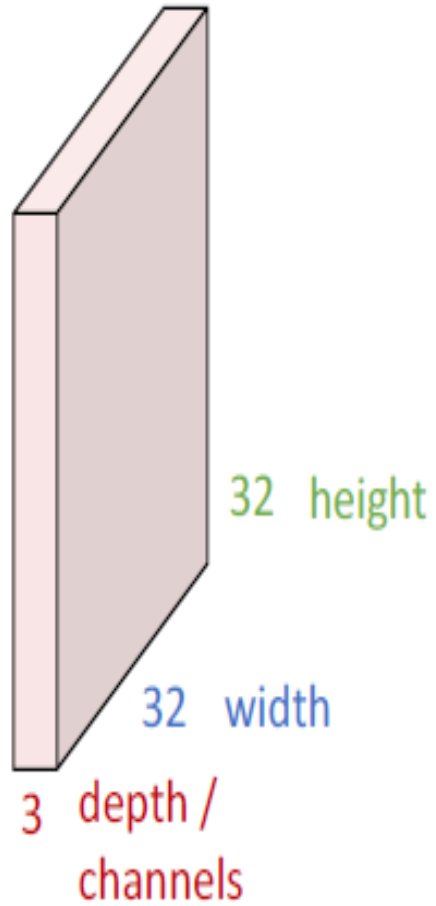
# 합성곱층의 이해

- 합성곱이란?



# 합성곱층의 이해

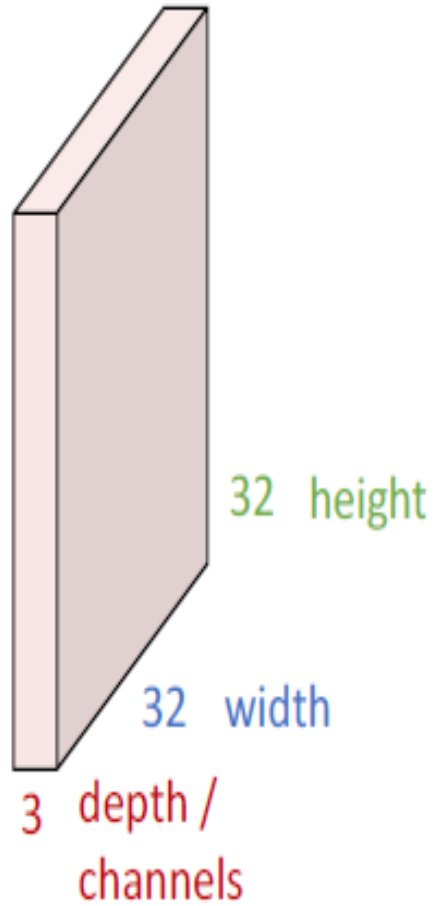
- 3x32x32 이미지





# 합성곱층의 이해

- 3x32x32 이미지

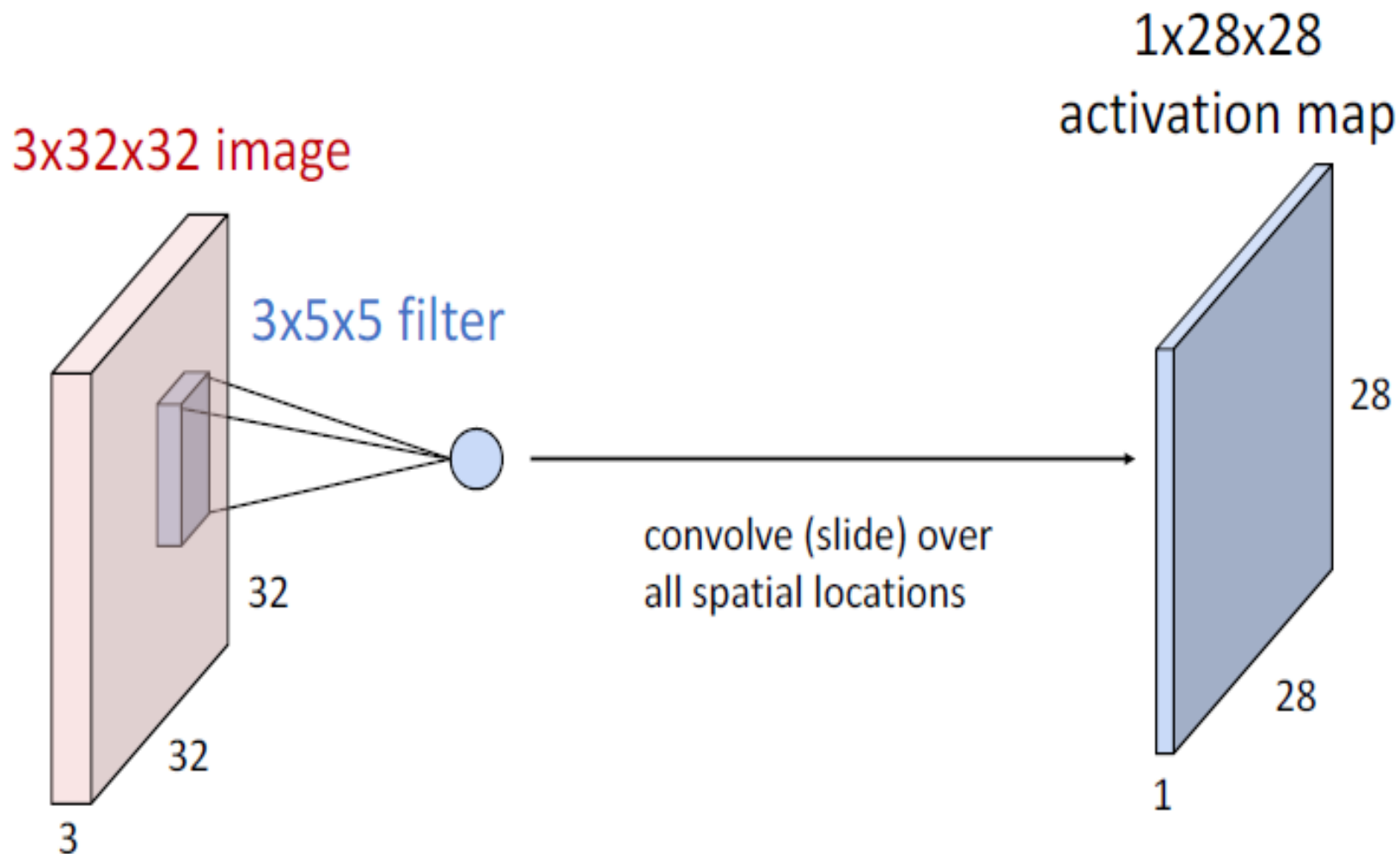


3x5x5 filter



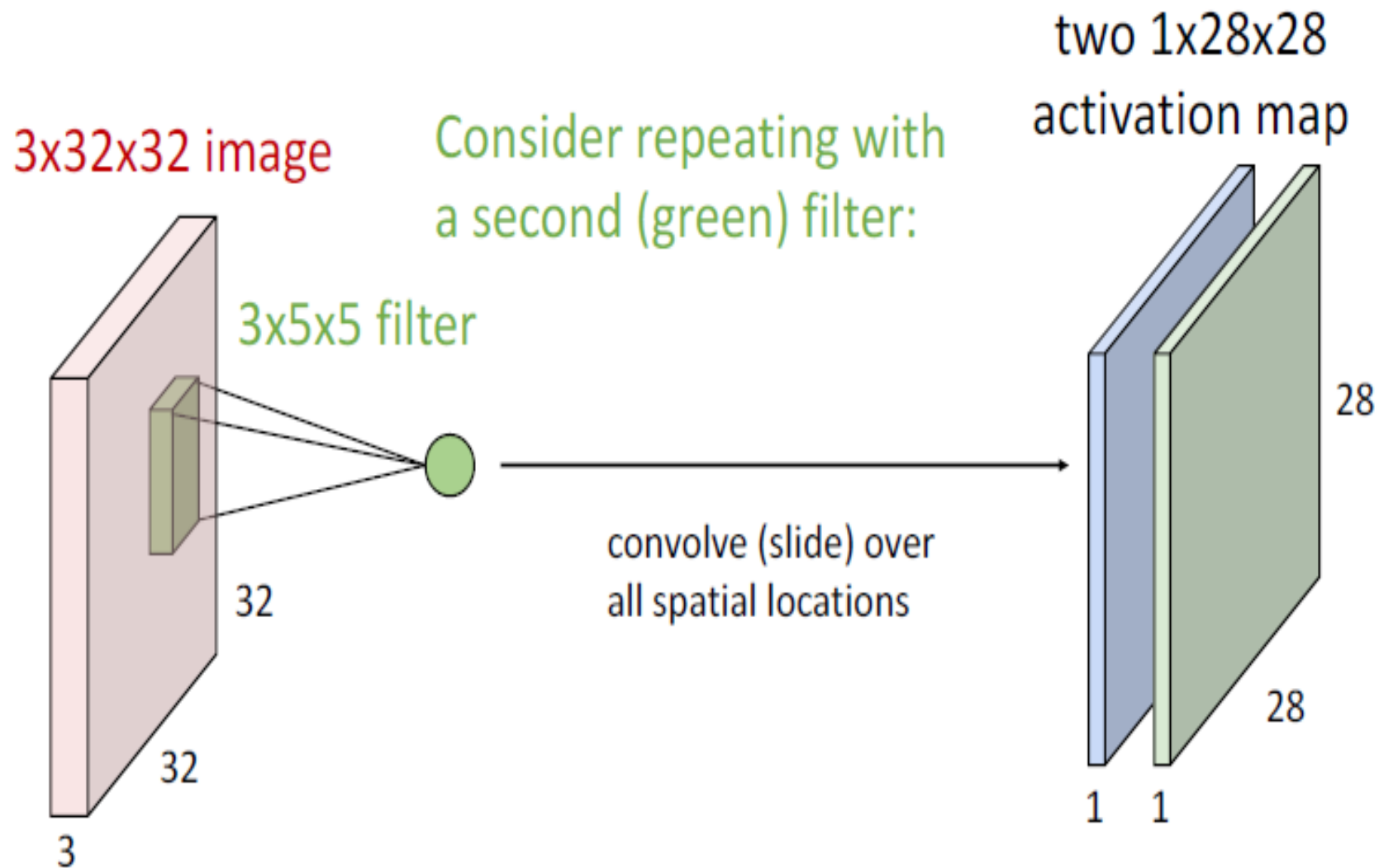
# 합성곱층의 이해

- 첫번째 활성 맵



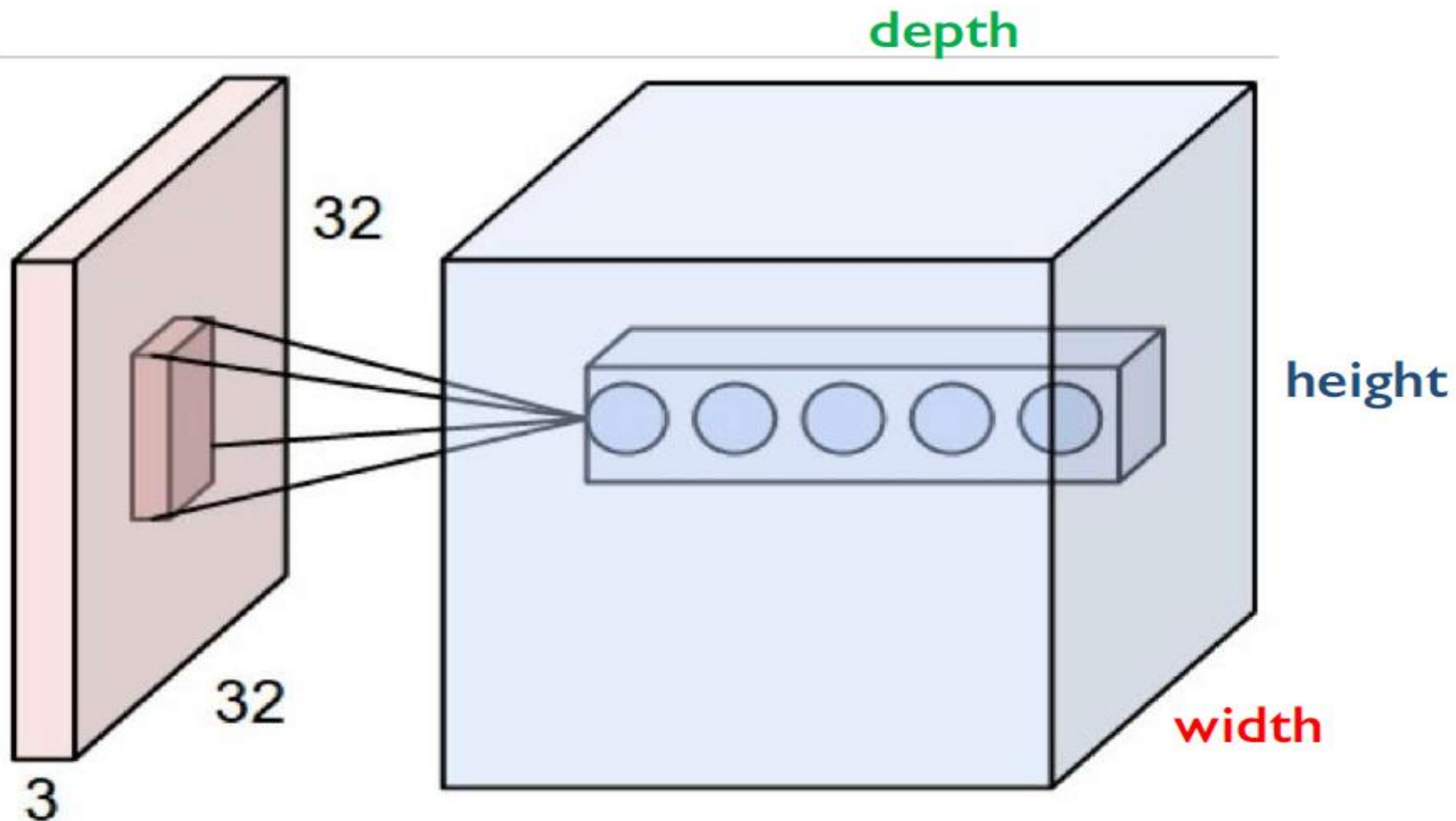
# 합성곱층의 이해

- 두번째 활성 맵



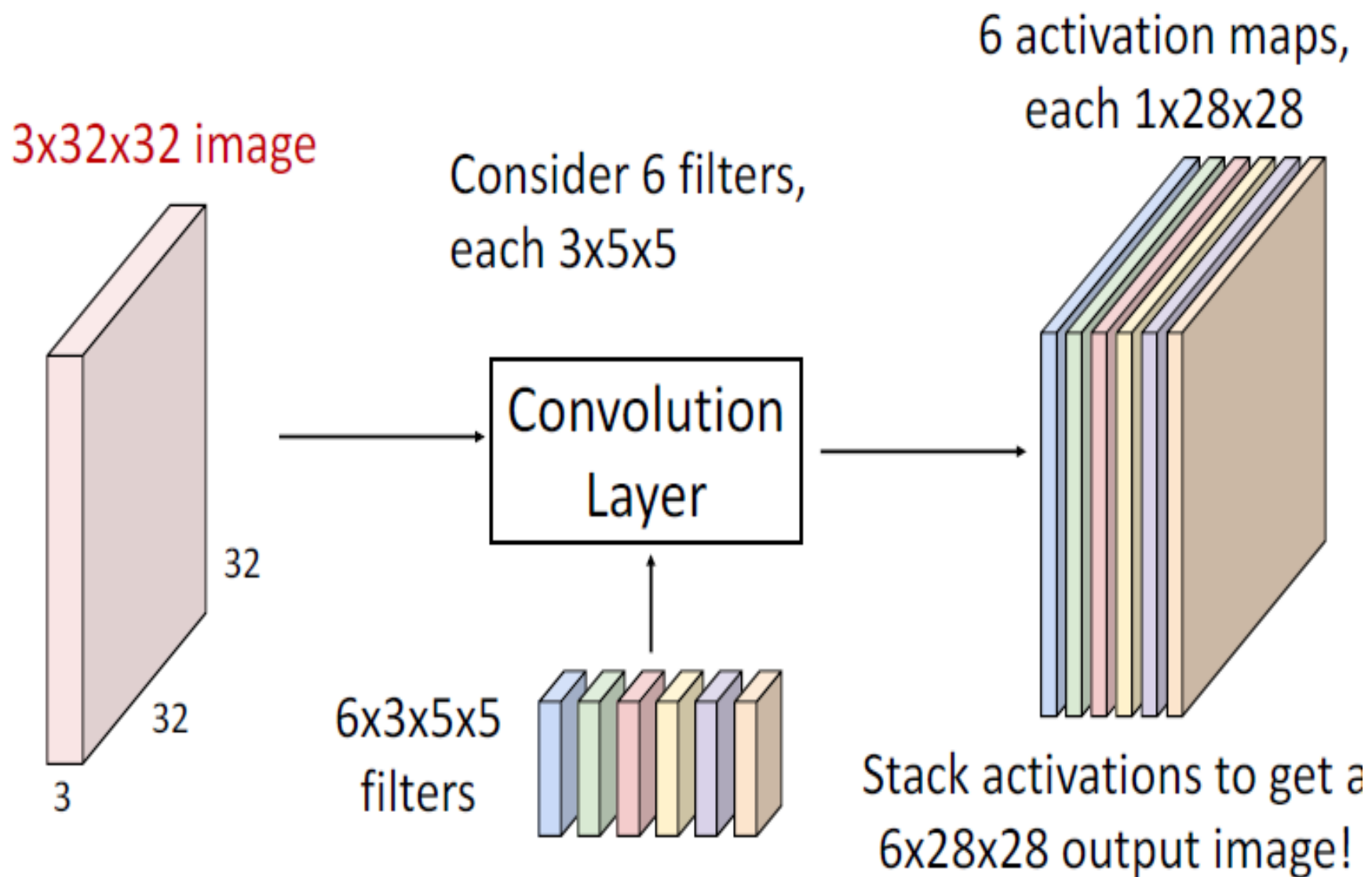
## 합성곱층의 이해

- 여기서 depth는 필터의 개수임.



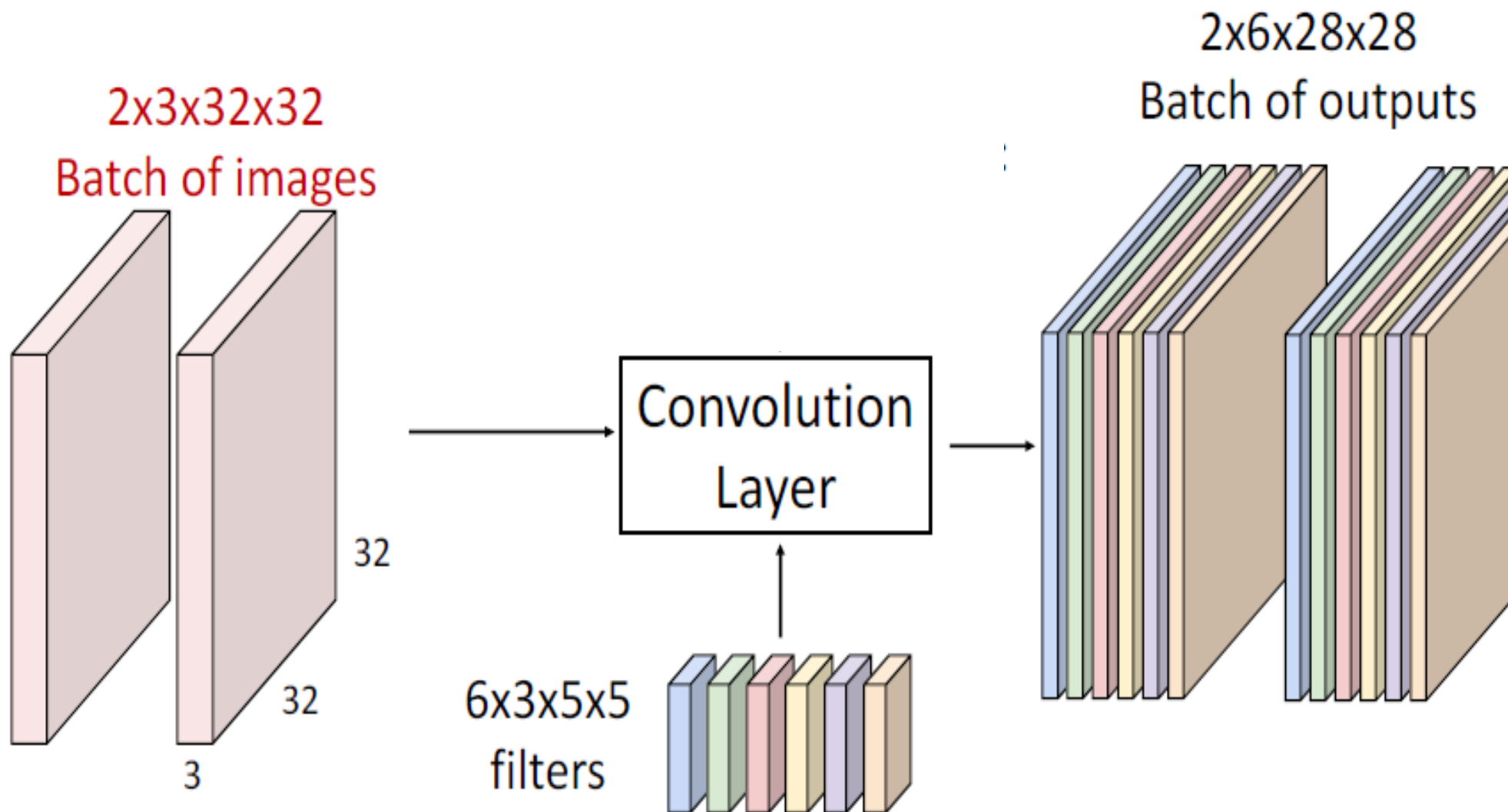
# 합성곱층의 이해

- 6개의 필터로 6개의 활성화 맵을 만든다.



## 합성곱층의 이해

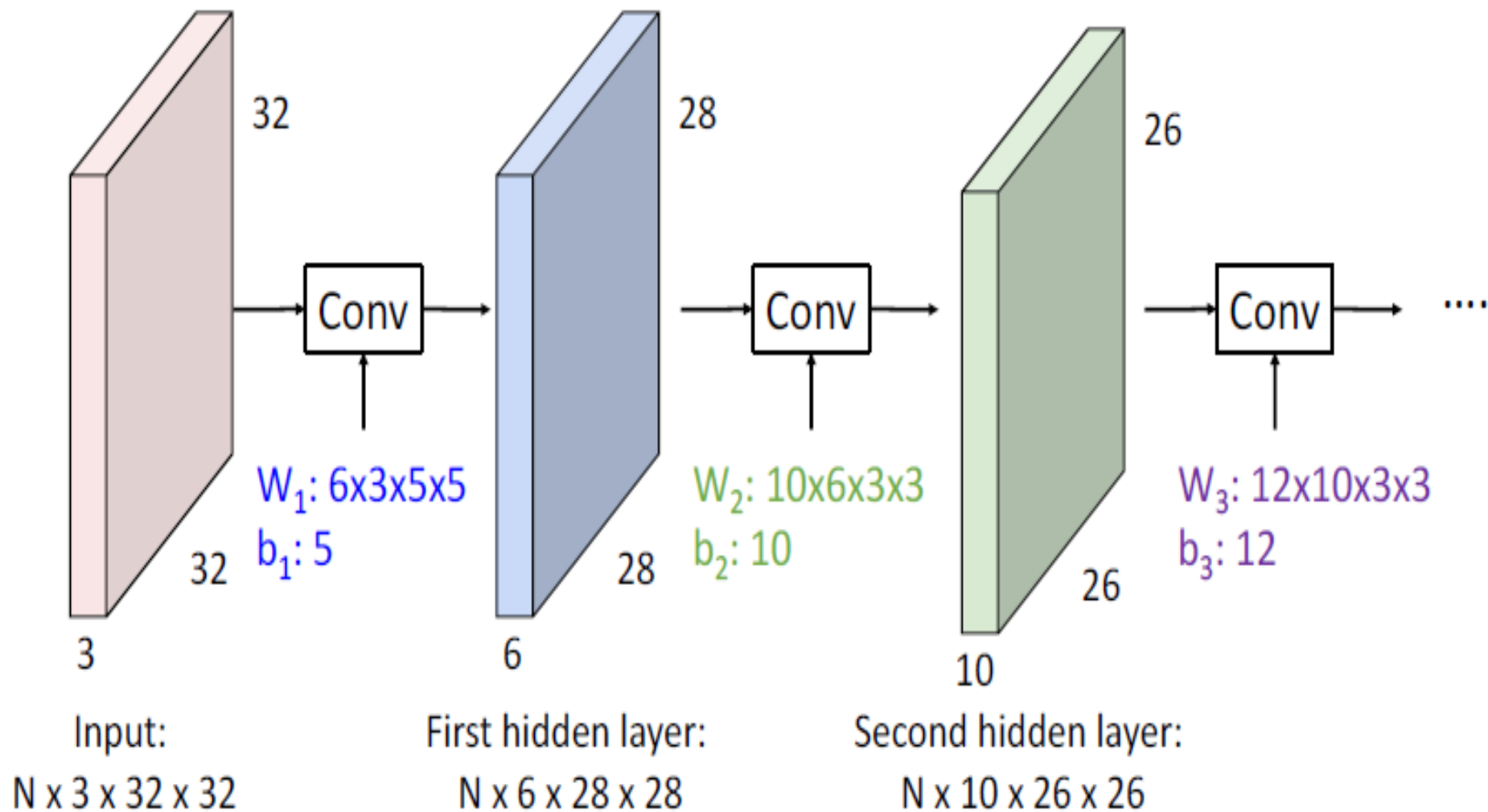
- 2개의 이미지에 대해 6개의 필터로 6개짜리 2세트의 활성맵을 만든다.





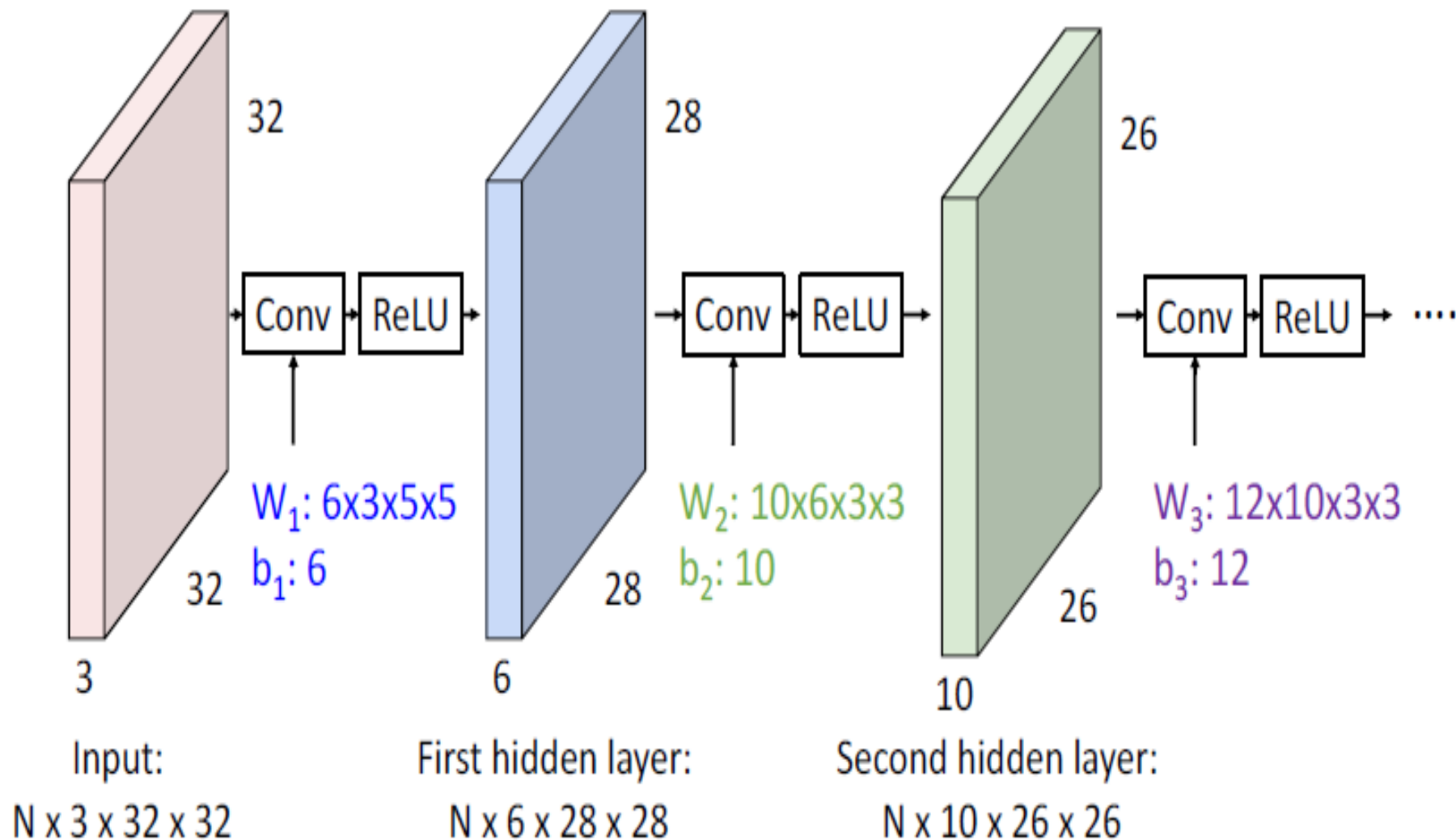
# 합성곱층의 이해

- 합성곱의 적층



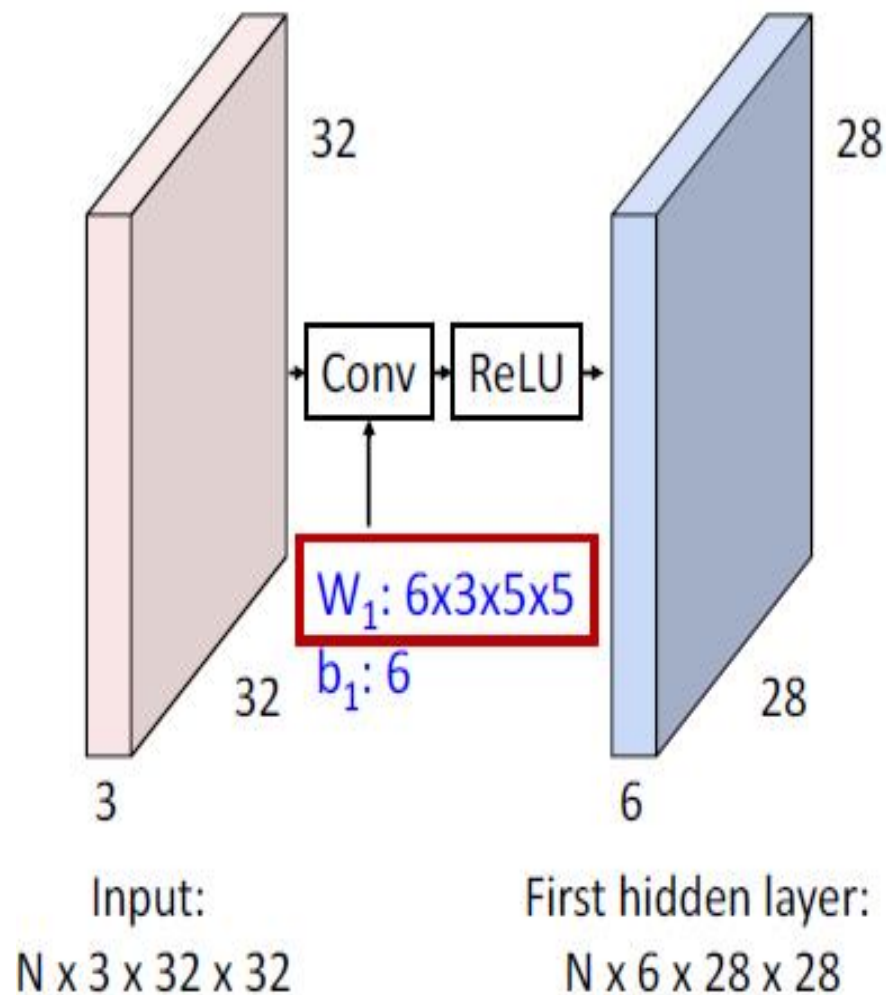
# 합성곱층의 이해

- RELU 층의 추가로 진정한 활성맵을 만든다.

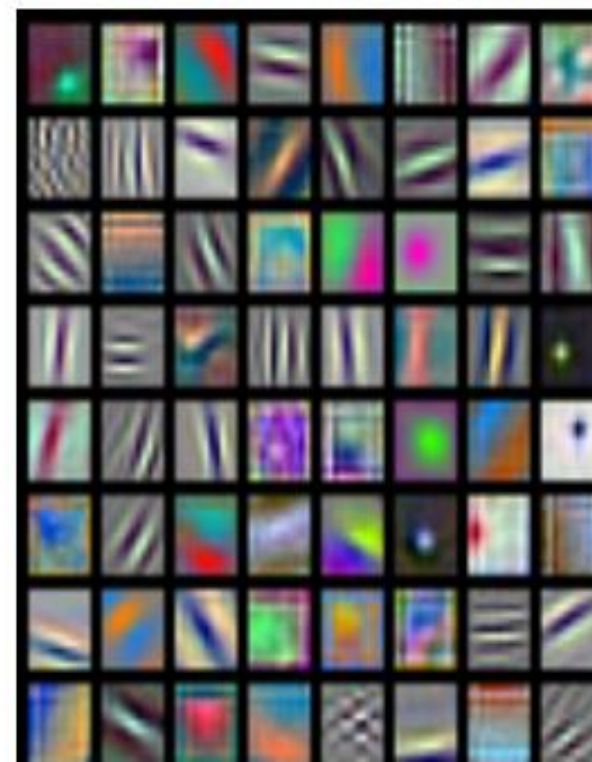


# 합성곱층의 이해

- 필터의 시각화 및 의미



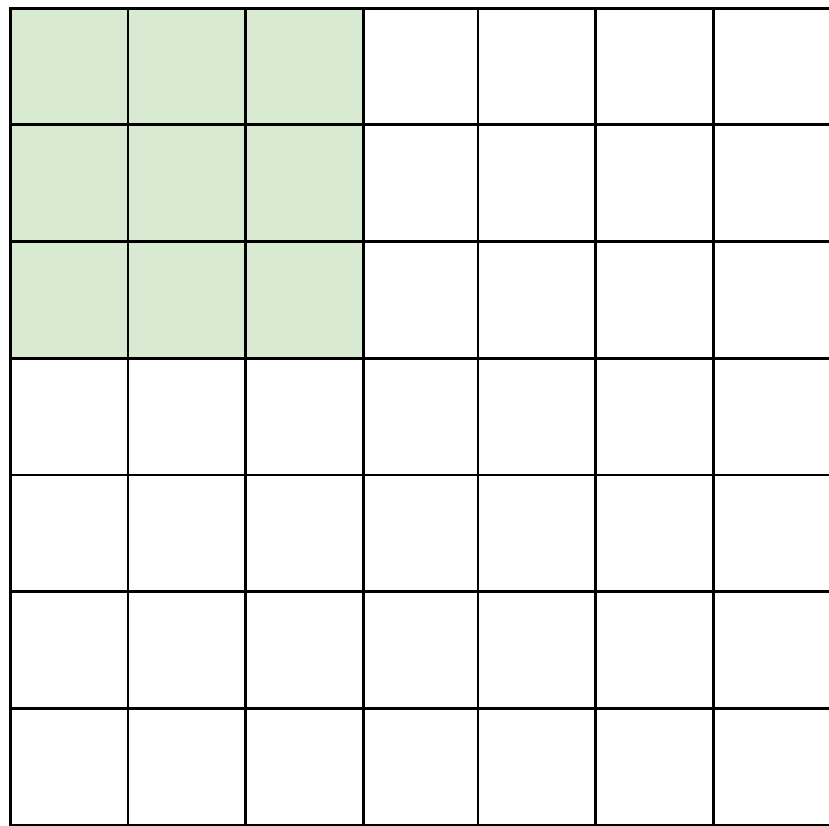
첫번째 합성곱층은은 국지적 이미지 템플릿이다. (방향성 모서리, 색상 대조 등)



AlexNet: 64 filters, each  $3 \times 11 \times 11$

# 합성곱 신경망의 이해

- 공간적 이해



7

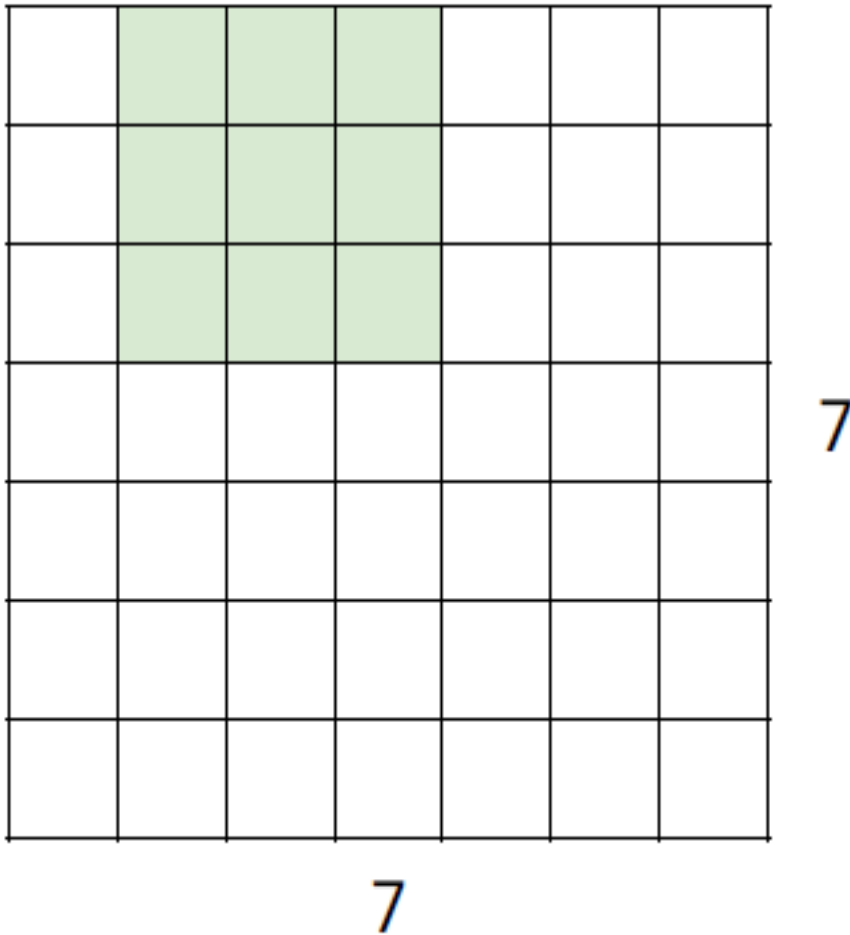
7

Input: 7x7

Filter: 3x3

# 합성곱 신경망의 이해

- 공간적 이해

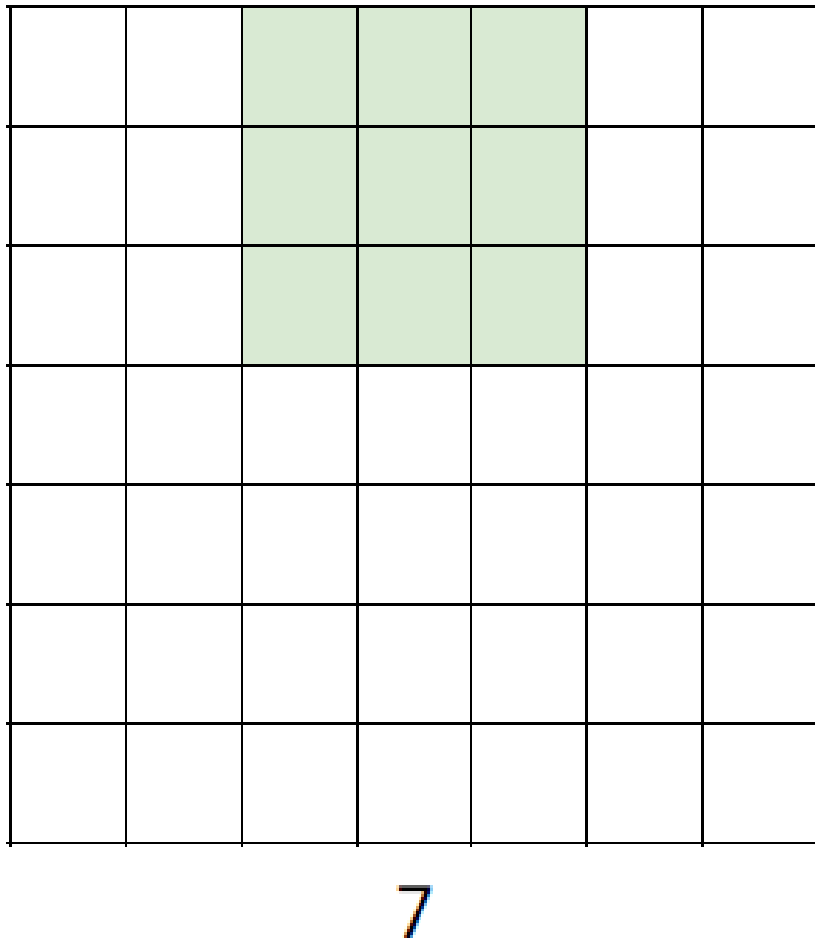


Input: 7x7

Filter: 3x3

# 합성곱 신경망의 이해

- 공간적 이해

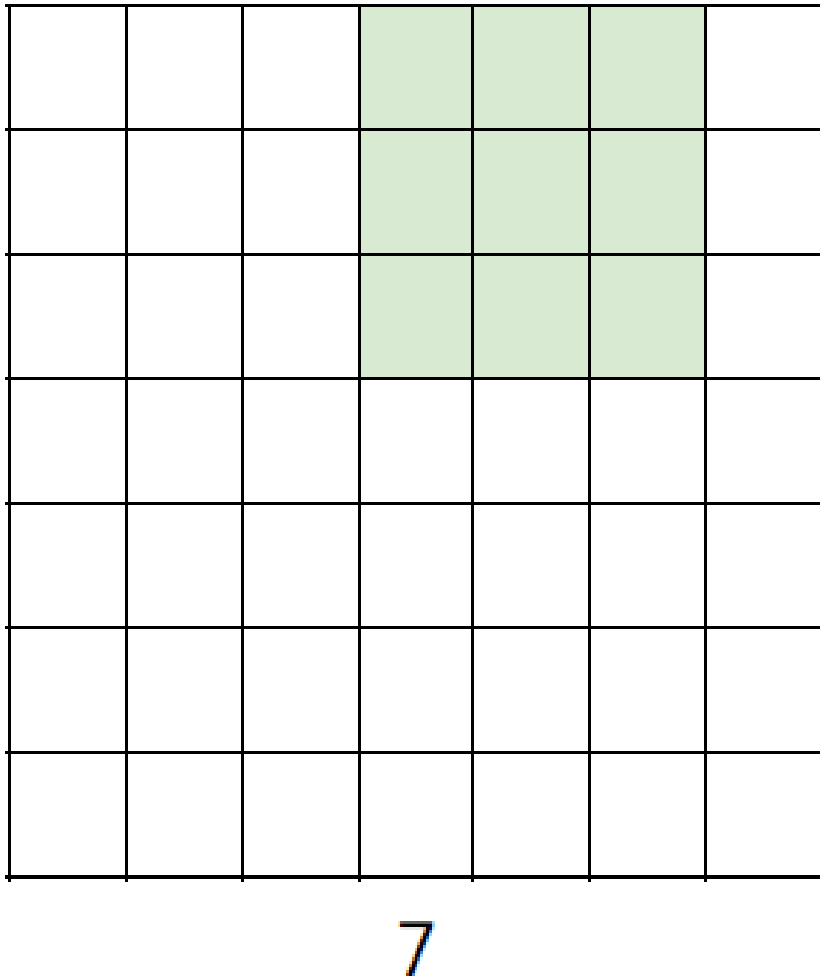


Input: 7x7  
Filter: 3x3



# 합성곱 신경망의 이해

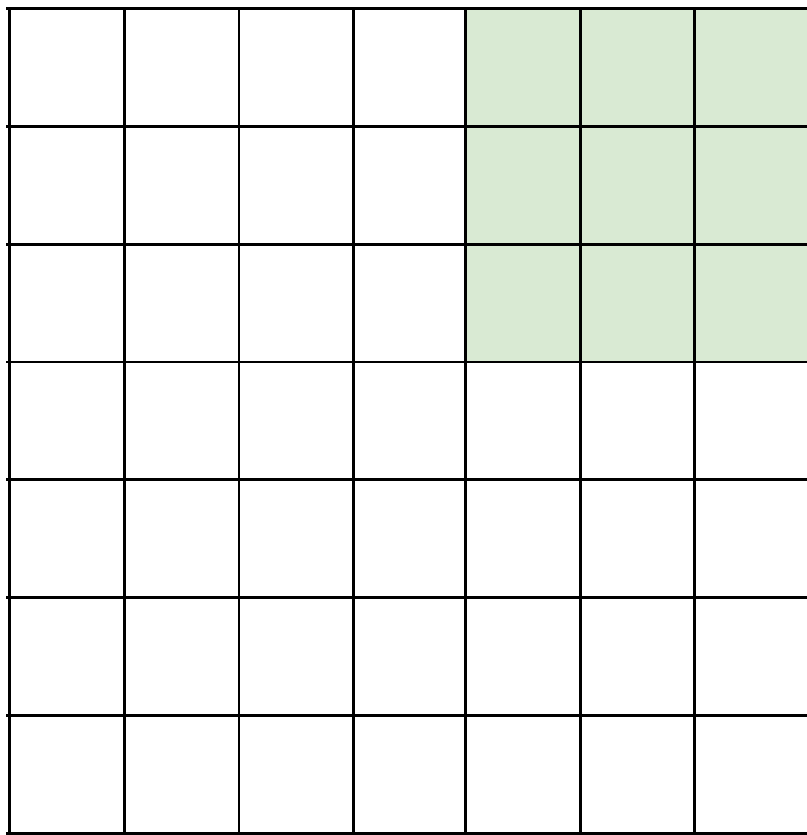
- 공간적 이해



Input: 7x7  
Filter: 3x3

# 합성곱 신경망의 이해

- 공간적 이해



7

7

Input: 7x7

Filter: 3x3

Output: 5x5

- 일반적으로 입력 크기가  $W$ 이고 필터크기가  $K$ 이면 출력은  $W-k+1$ 이다.

# 합성곱 신경망의 이해

- 패딩(padding)

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

Input: 7x7

Filter: 3x3

Output: 5x5

In general:

Input:  $W$

Filter:  $K$

Padding:  $P$

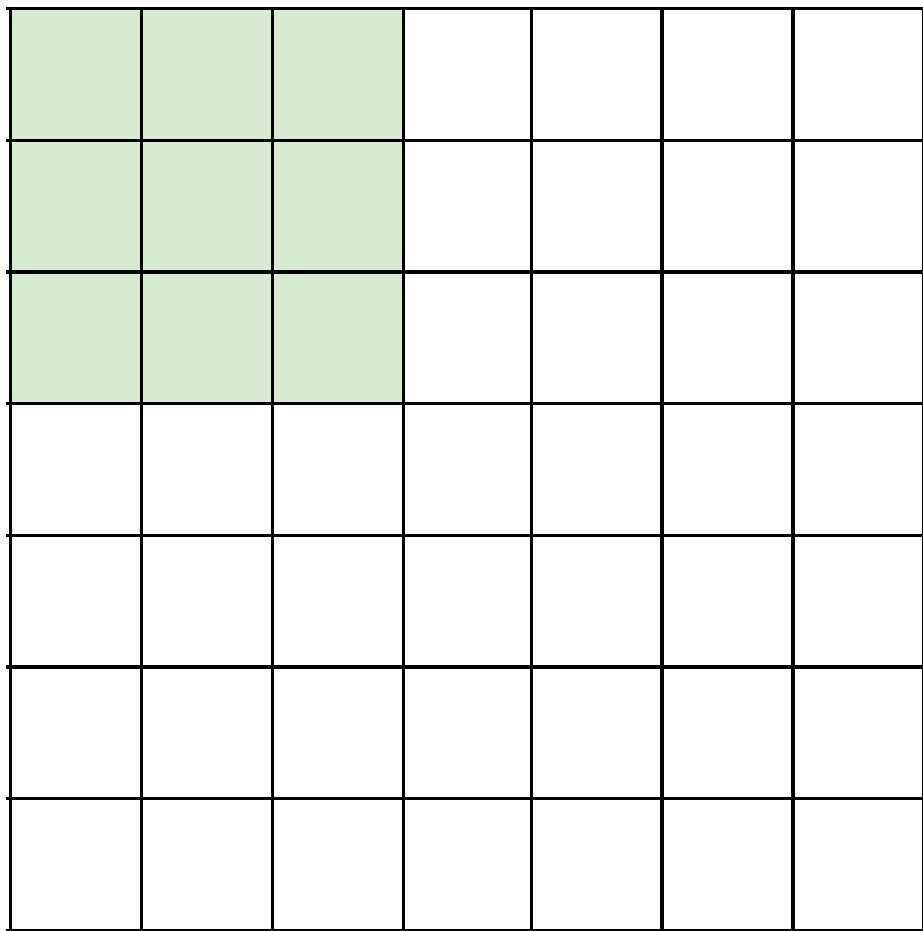
Output:  $W - K + 1 + 2P$

Very common:

Set  $P = (K - 1) / 2$  to  
make output have  
same size as input!

# 합성곱 신경망의 이해

- 스트라이드(Stride)



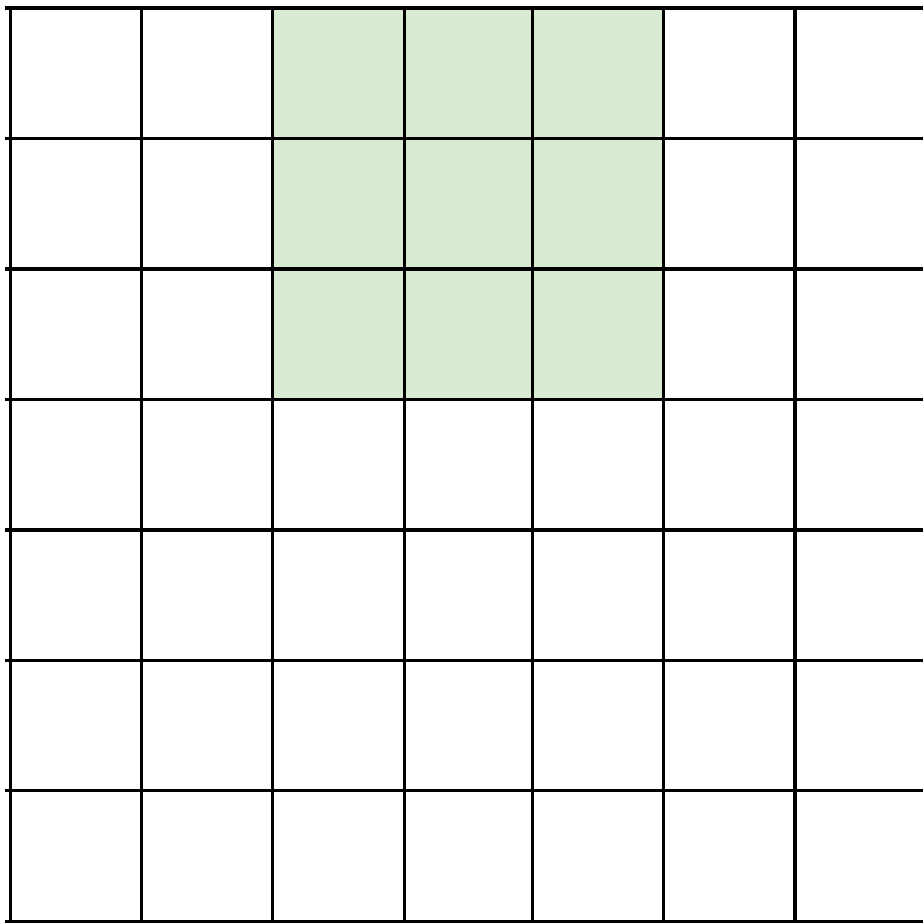
Input: 7x7

Filter: 3x3

Stride: 2

# 합성곱 신경망의 이해

- 스트라이드(Stride)



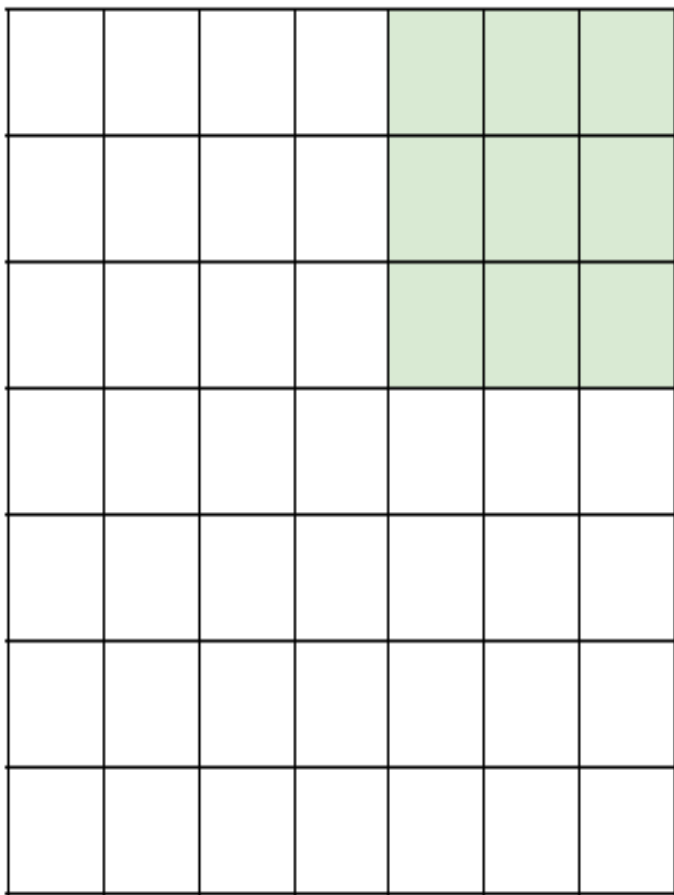
Input: 7x7

Filter: 3x3

Stride: 2

# 합성곱 신경망의 이해

- 스트라이드(Stride)



Input: 7x7

Filter: 3x3

Output: 3x3

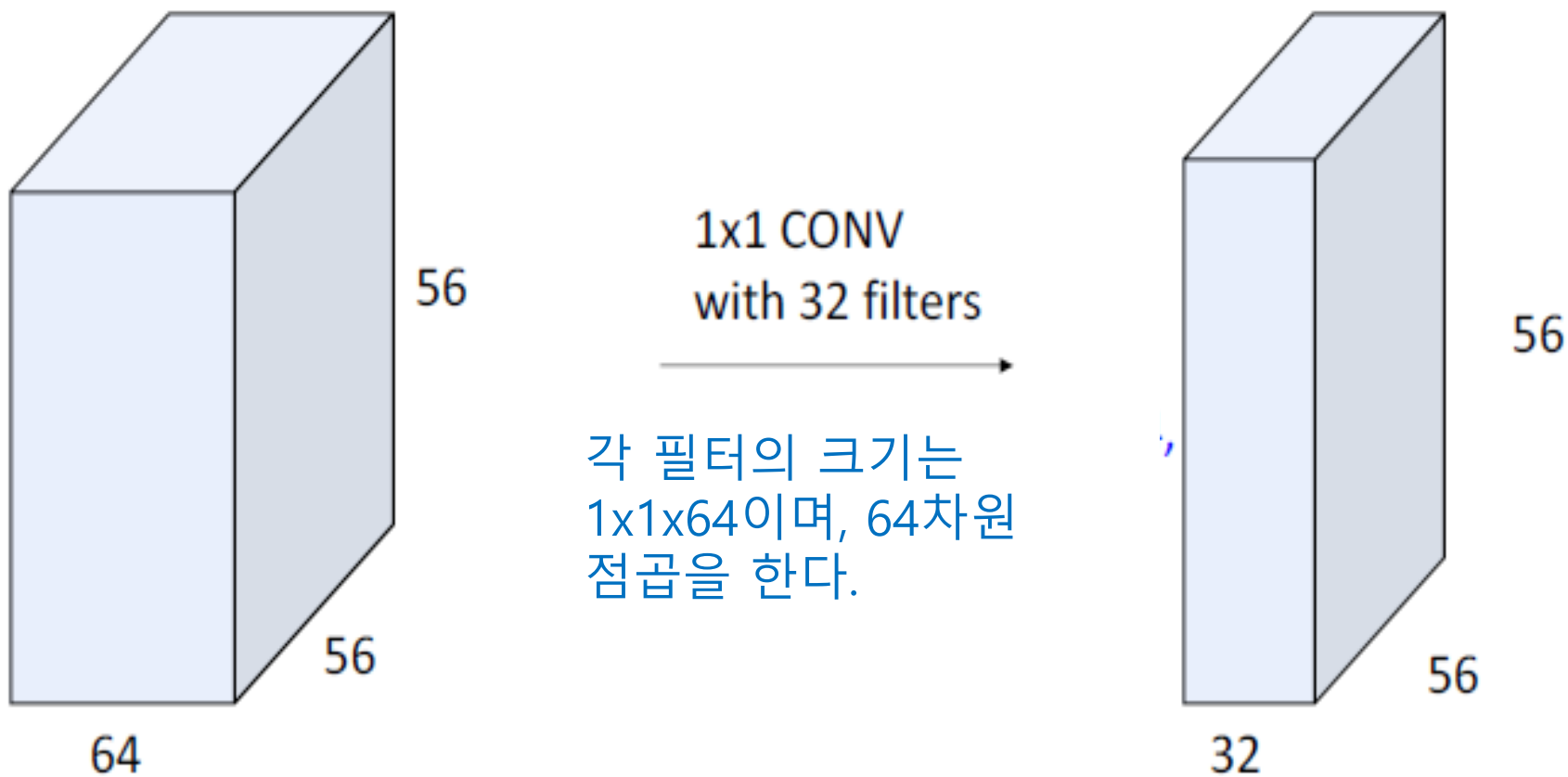
Stride: 2

- 일반적으로 입력 크기가  $W$ 이고 필터크기가  $K$ 이면 패딩이  $P$  그리고 Stride가  $S$ 이면 출력은  $(W-K+2P)/S+1$ 이다.



# 합성곱 신경망의 이해

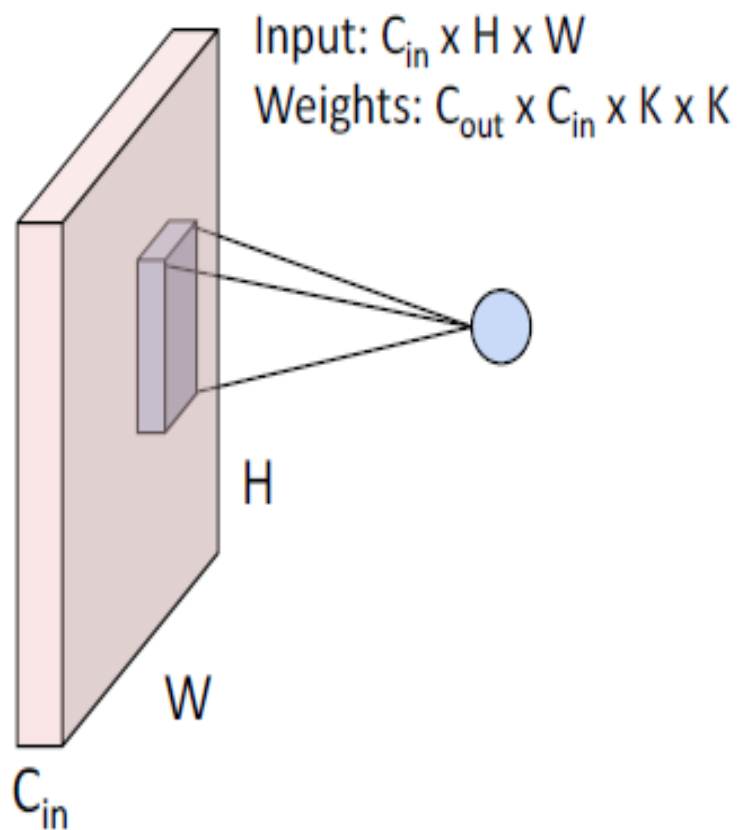
- 1x1 합성곱



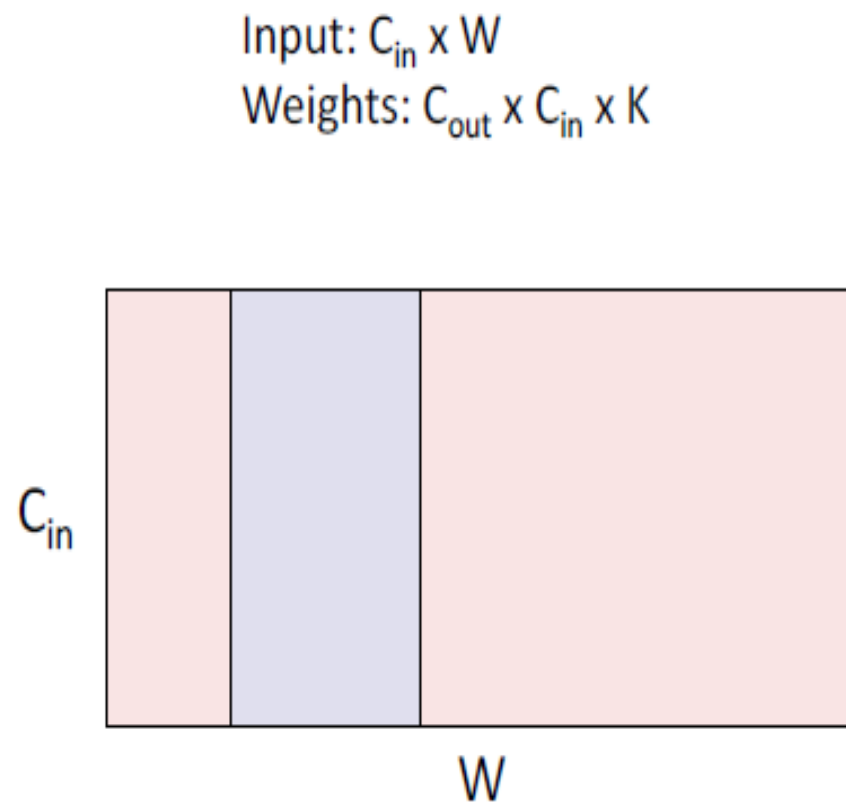
# 합성곱 신경망의 이해

- 2D 대 1D 합성곱

: 2D Convolution



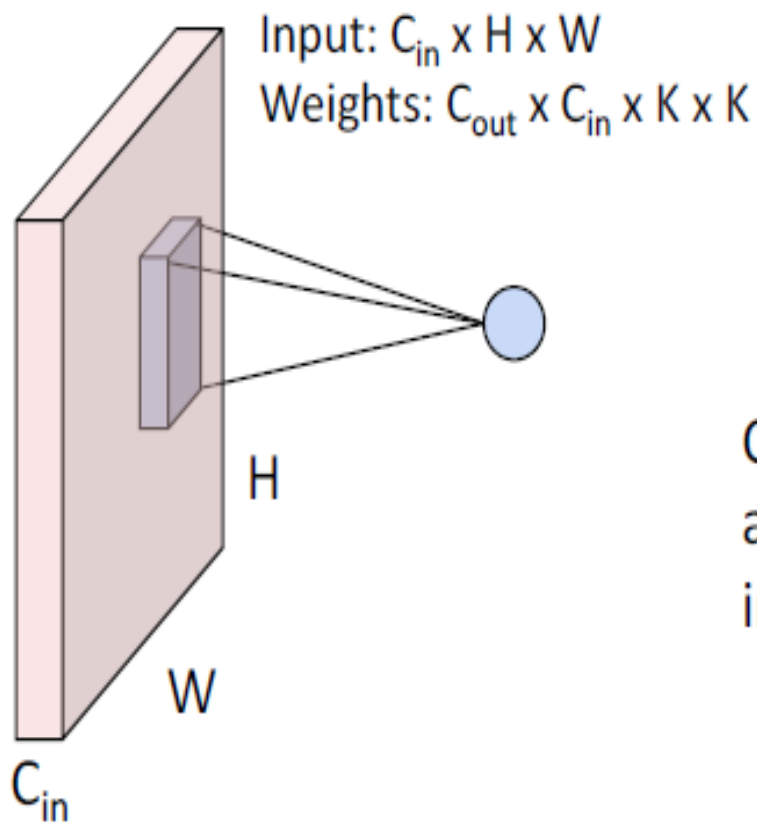
1D Convolution



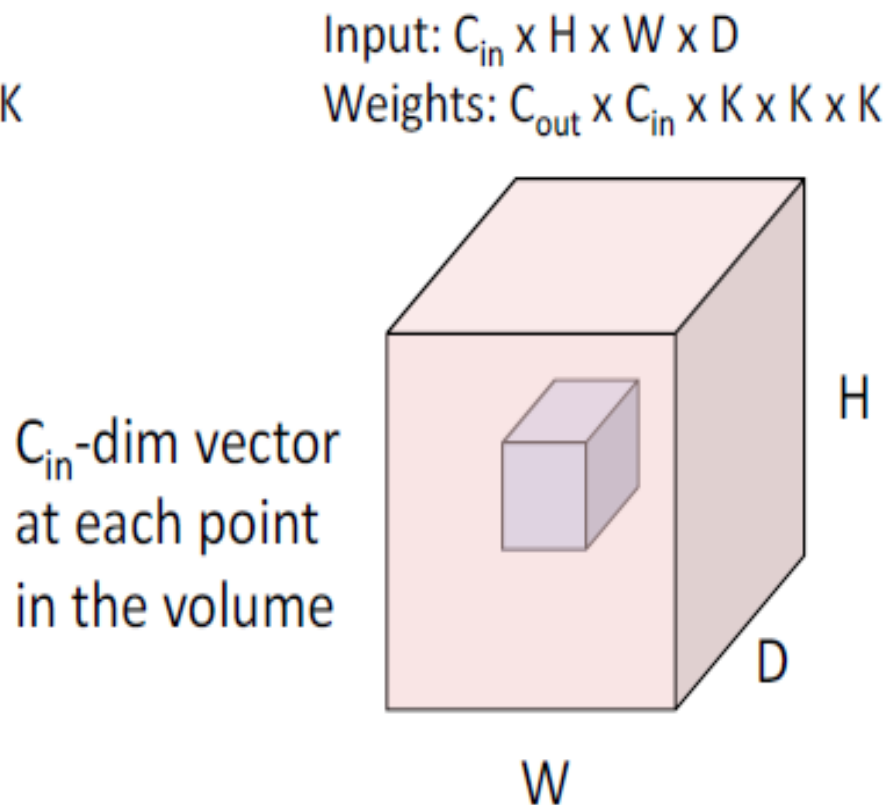
# 합성곱 신경망의 이해

- 2D 대 3D 합성곱

2D Convolution

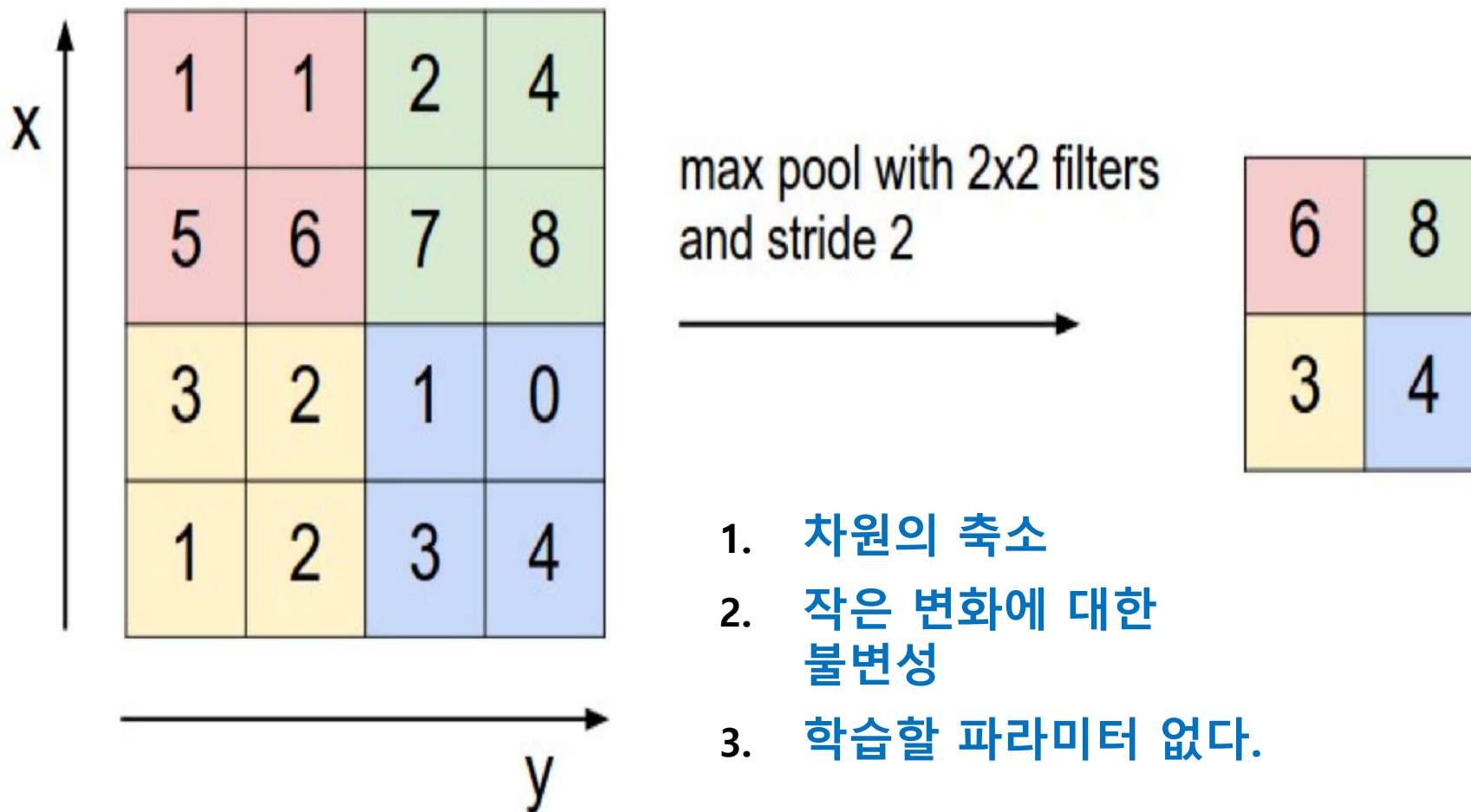


3D Convolution



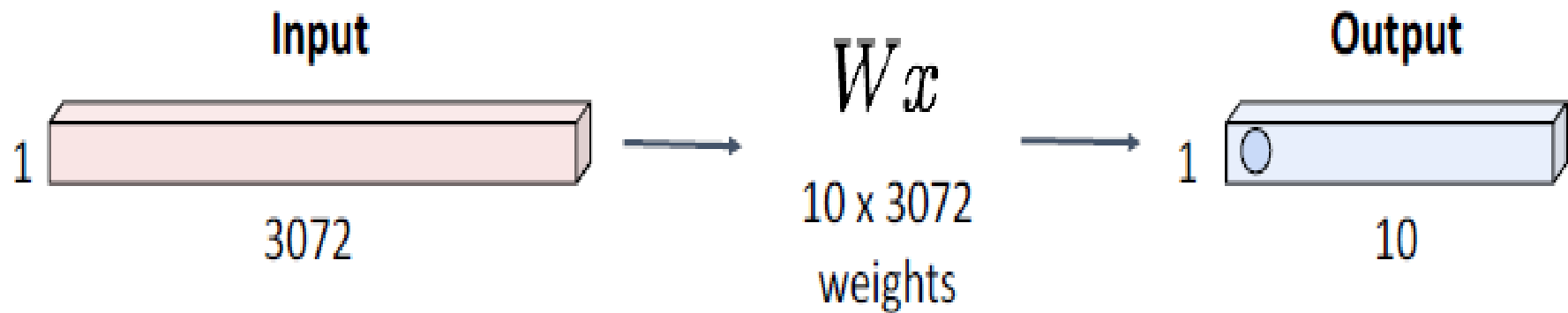
# 풀링층의 이해

- 풀링: 다운샘플링



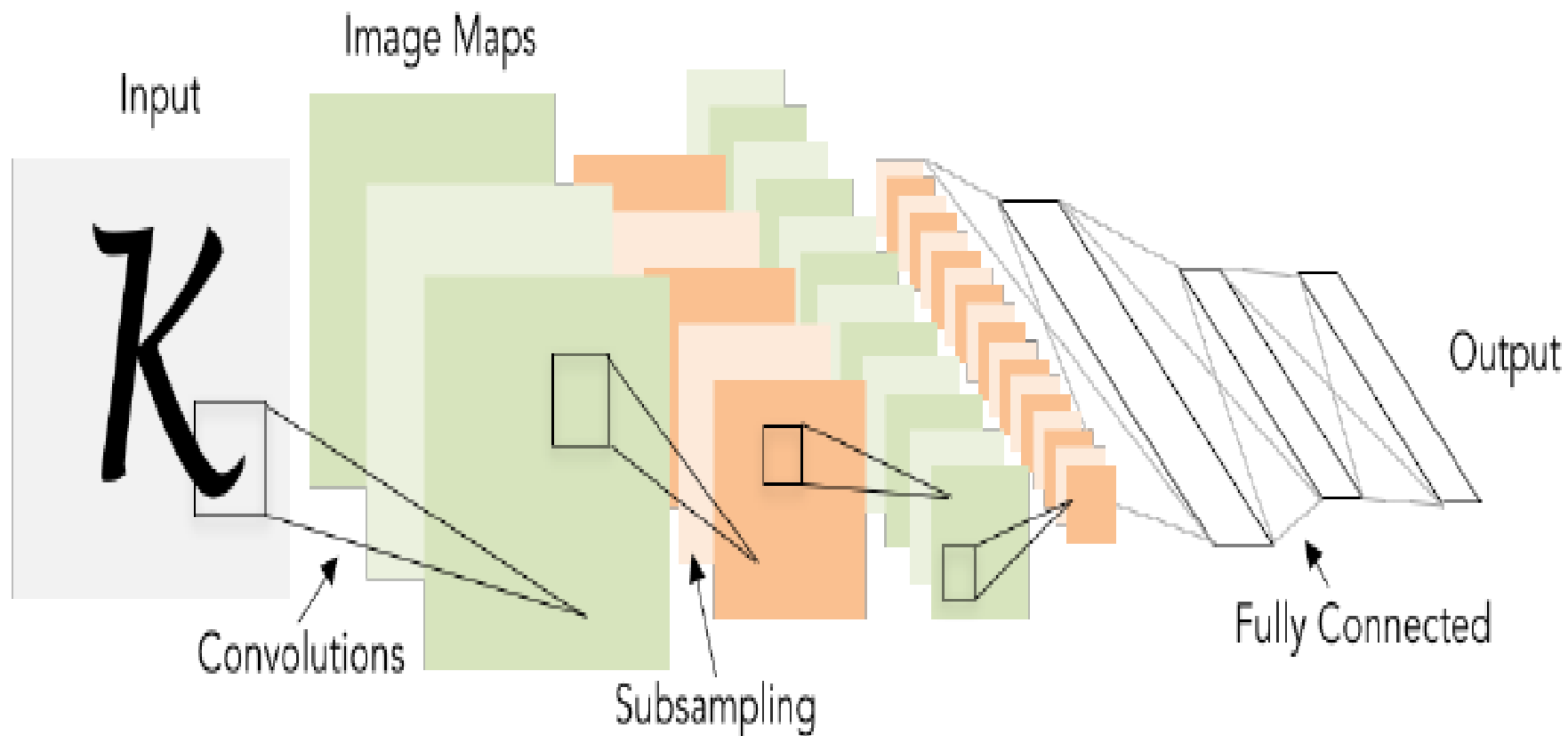
# 완전결합층 (Fully Connected Layer: FC)

- Flatten -> FC



# 합성곱 신경망의 이해

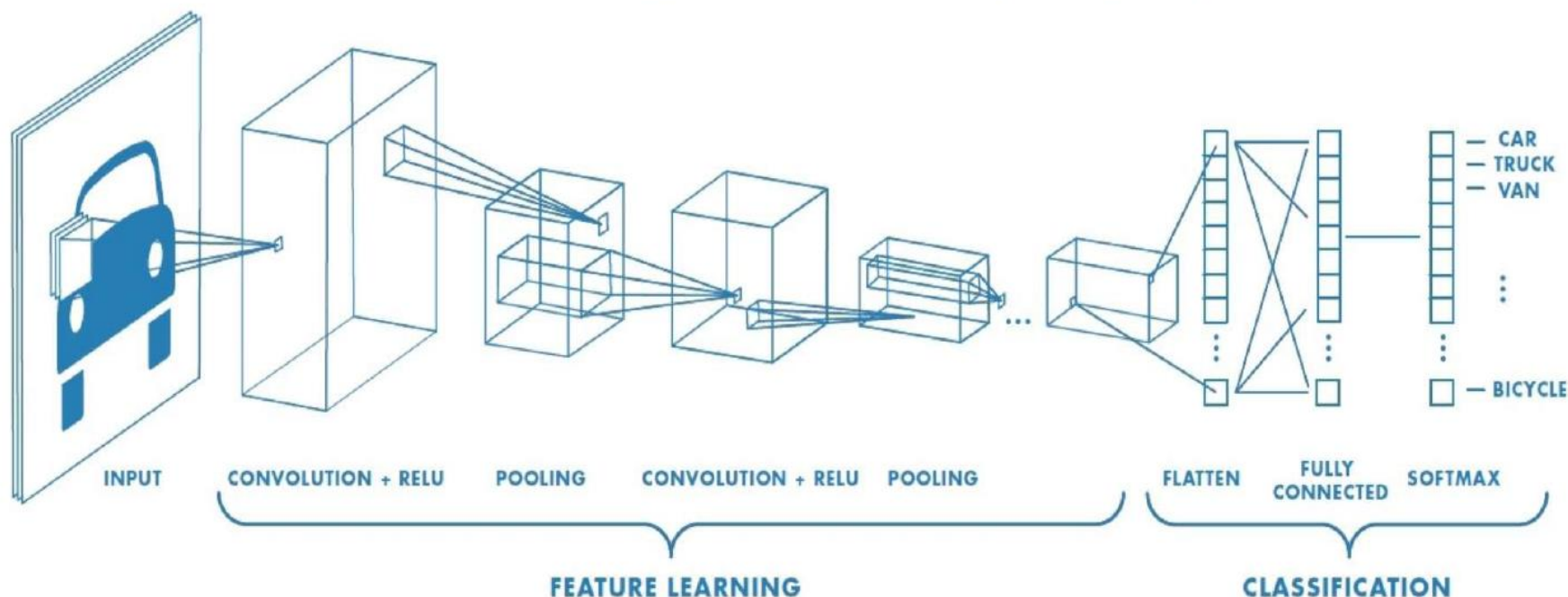
- 고전적 구조: [Conv, ReLU, Pool]xN, flatten, [FC, ReLU]xN, FC





# 합성곱 신경망의 이해

- Softmax와 Cross-Entropy 비용함수를 통한 분류

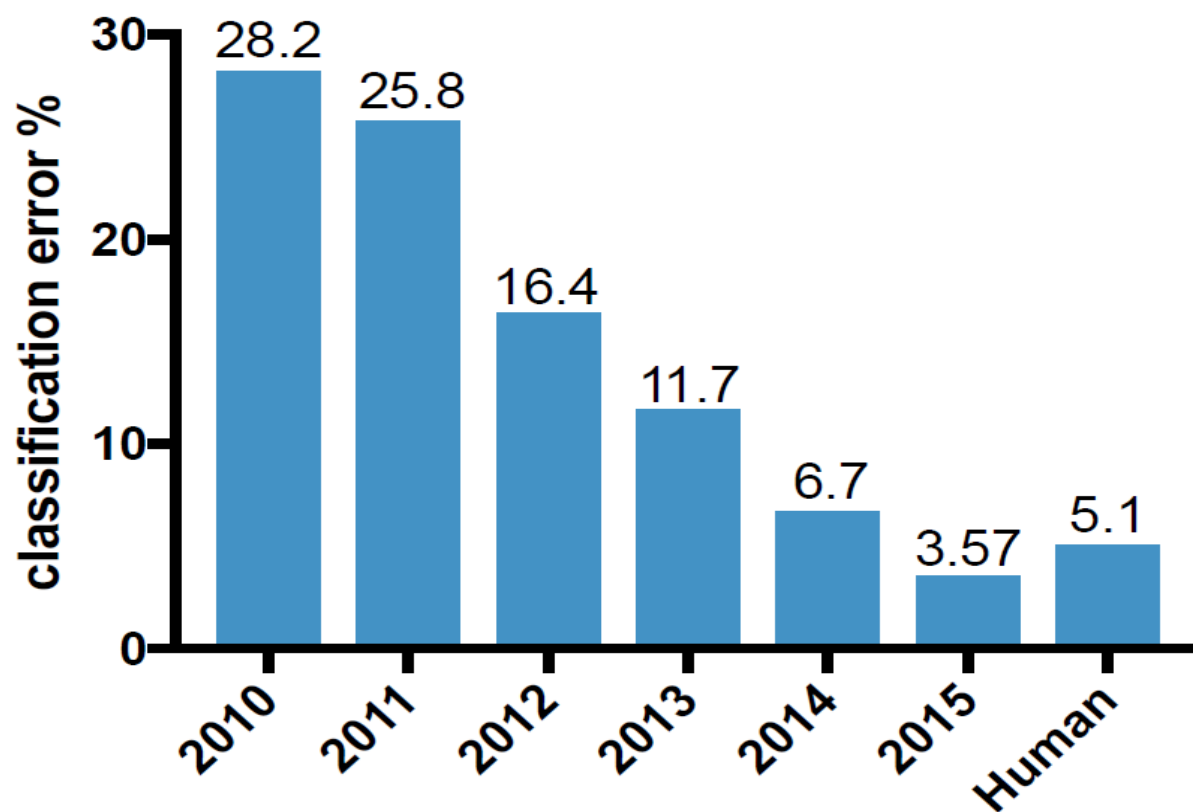


$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

$$J(\theta) = \sum_i y^{(i)} \log(\hat{y}^{(i)})$$

# 합성곱 신경망의 발전

## ● ImageNet 대회



**2012: AlexNet.** First CNN to win.

- 8 layers, 61 million parameters

**2013: ZFNet**

- 8 layers, more filters

**2014: VGG**

- 19 layers

**2014: GoogLeNet**

- "Inception" modules
- 22 layers, 5 million parameters

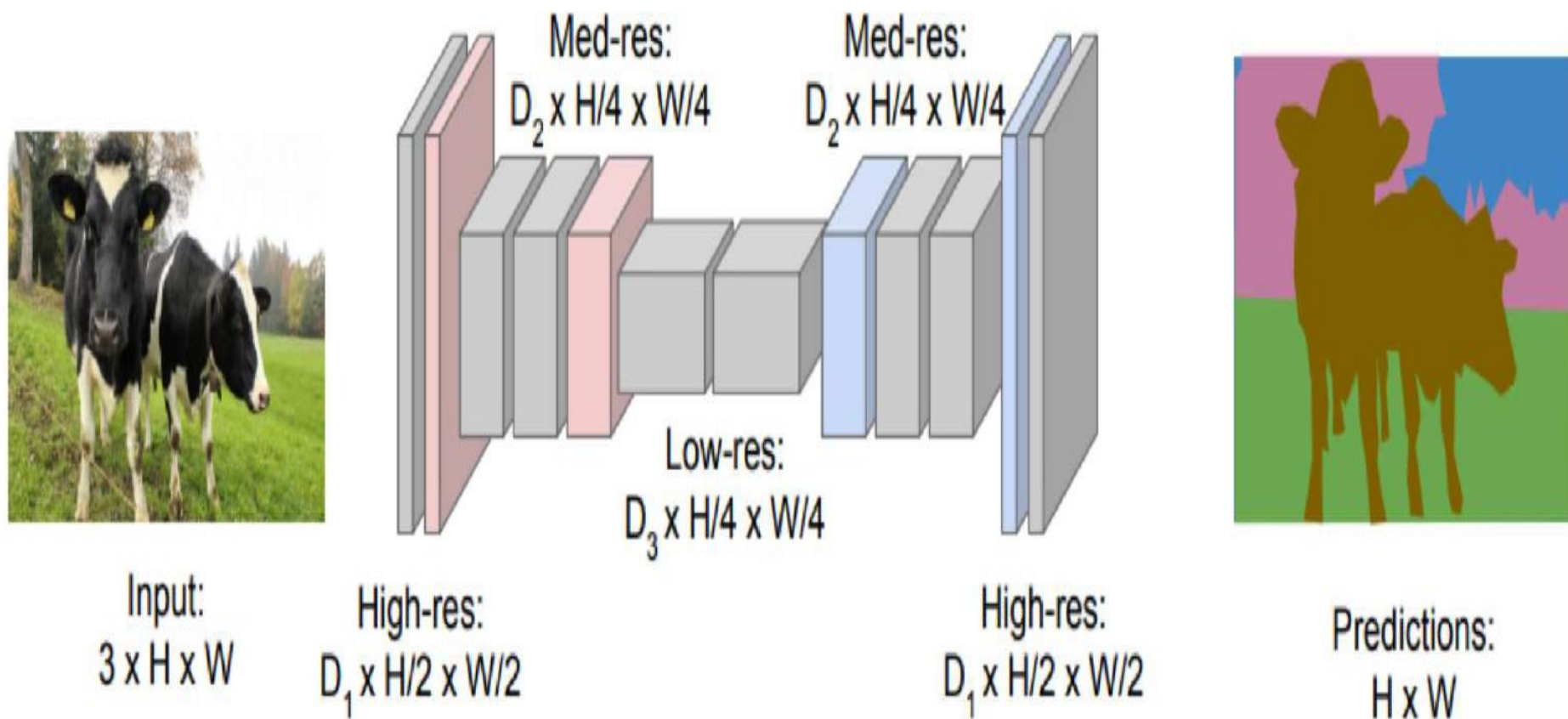
**2015: ResNet**

- 152 layers

## 업샘플링 (전치 합성곱의 이해)

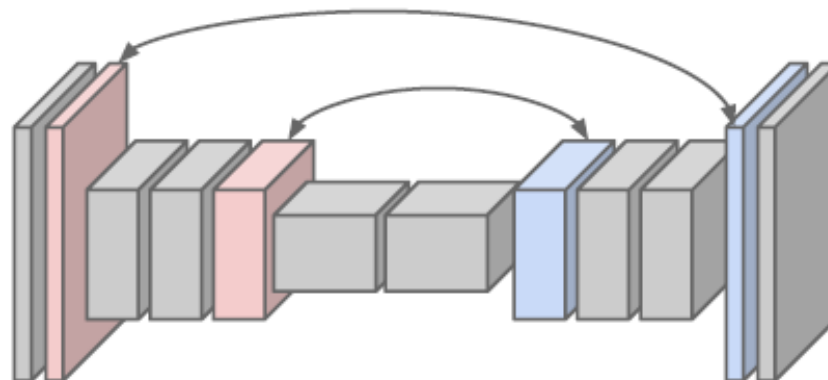
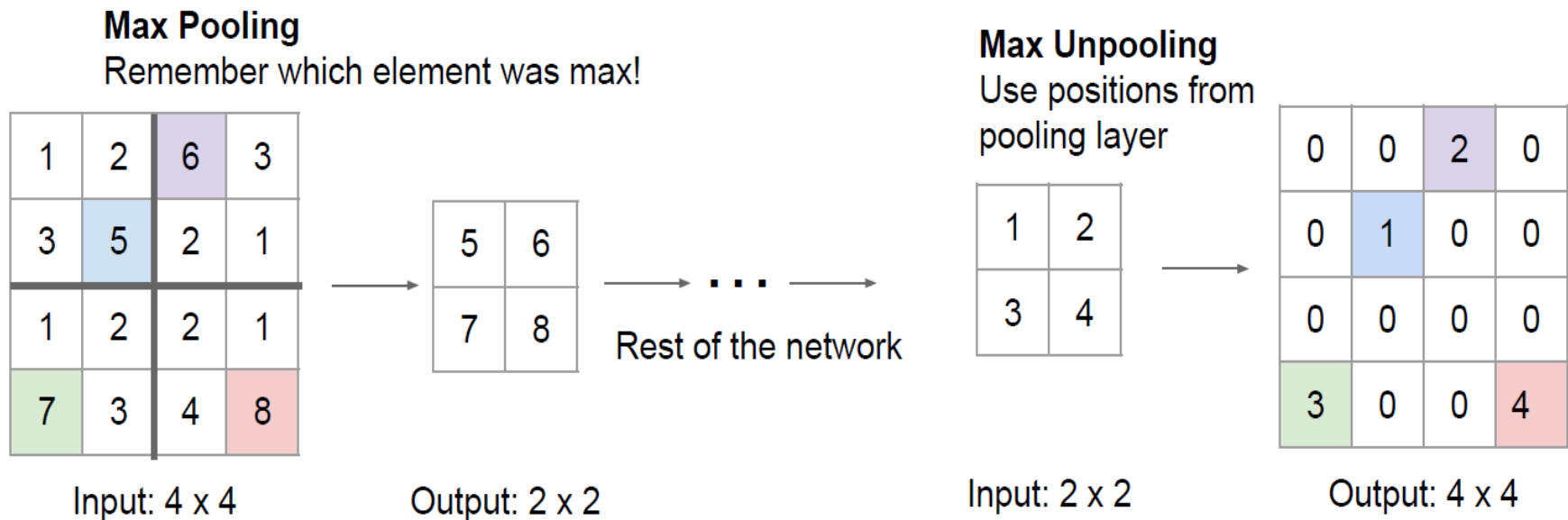
# 업샘플링

- ImangeNet 대회: 이미지 분할



# 업샘플링

- Max Unpooling: 다운샘플링과 업샘플링이 대칭이 되도록 Up&Un샘플링



# 업샘플링

- 언폴링 (풀링을 푸는 것)

Nearest Neighbor

1	2
3	4



1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

Input: 2 x 2

Output: 4 x 4

“Bed of Nails”

1	2
3	4



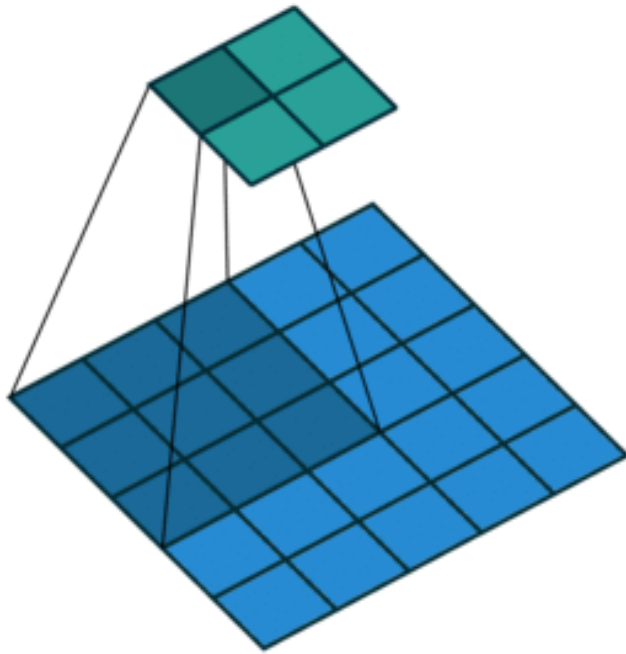
1	0	2	0
0	0	0	0
3	0	4	0
0	0	0	0

Input: 2 x 2

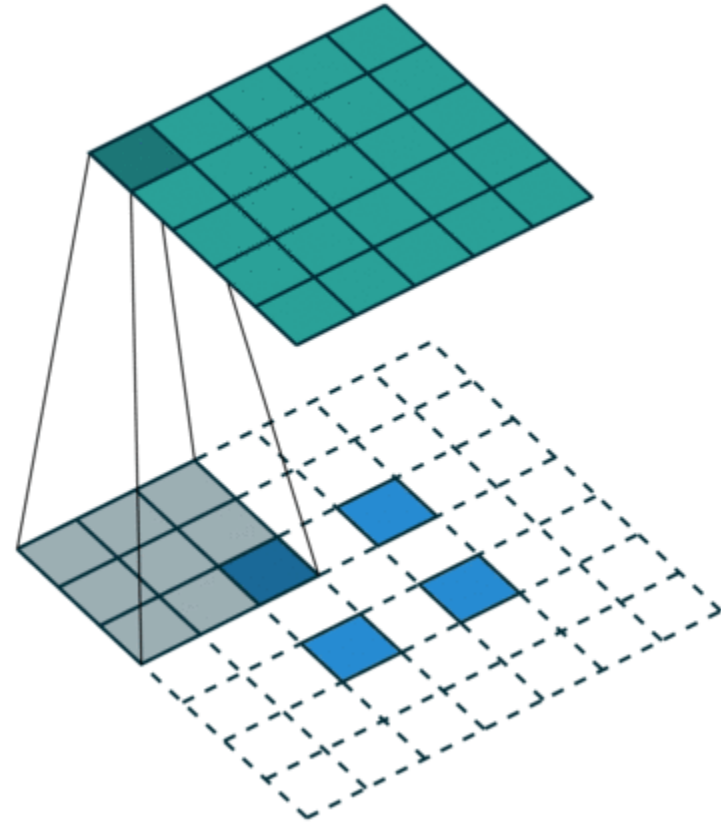
Output: 4 x 4

# 업샘플링: 전치 합성곱

- 원합성곱과 전치합성곱



- 원합성곱(3x3 필터의 0 패딩, 스트라이드 2 합성곱)

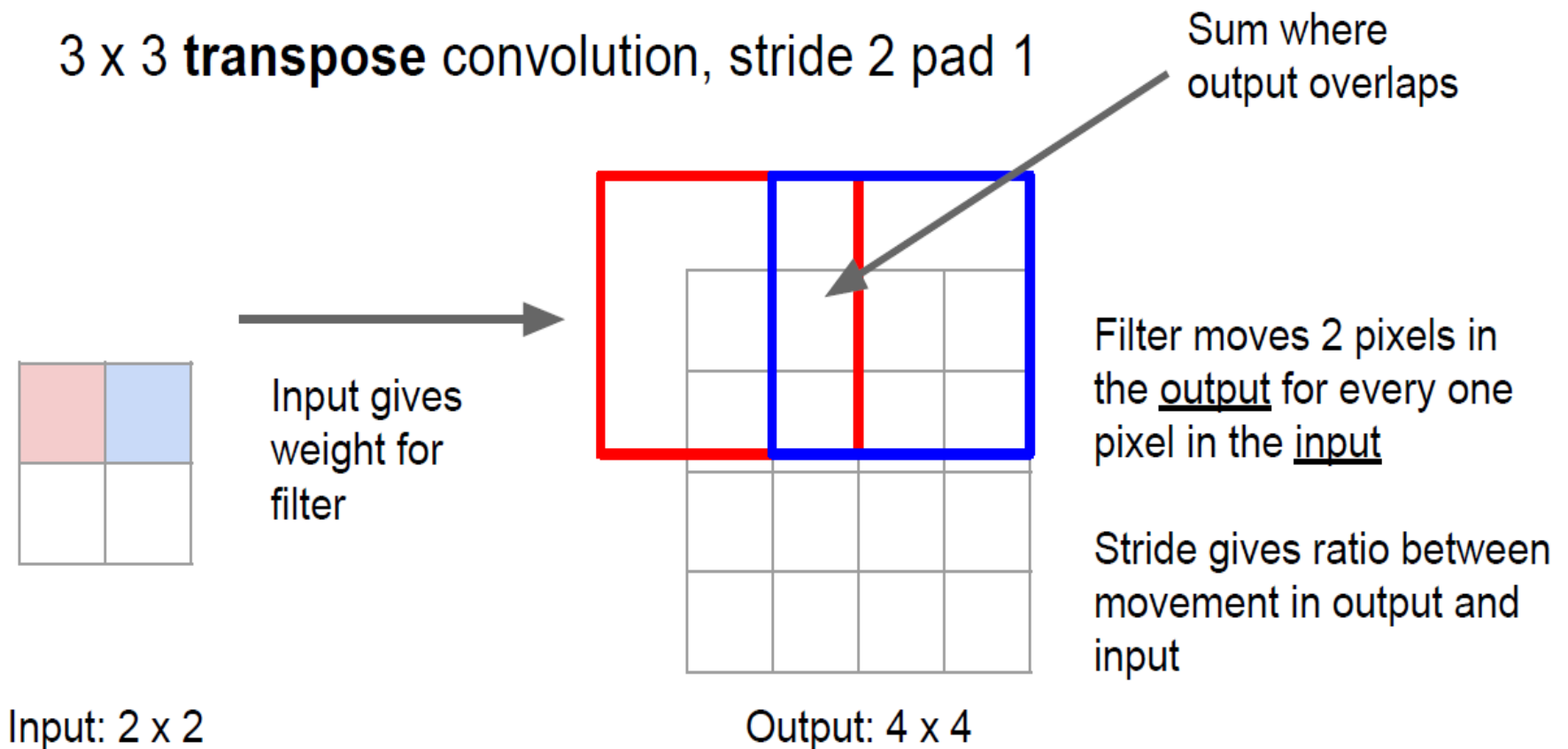


- 2x2의 출력에서 5x5의 입력으로 되돌리는 전치합성곱

# 업샘플링

- 전치 업샘플링의 여러이름: transposed convolution, Deconvolution, Upconvolution, Fractionally strided convolution, Backward strided convolution

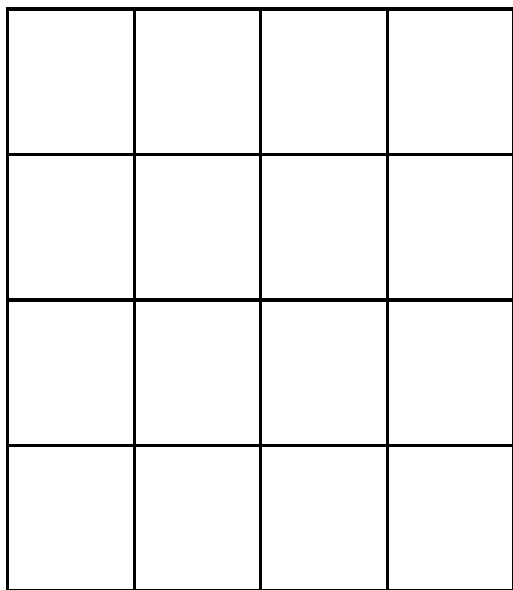
3 x 3 **transpose** convolution, stride 2 pad 1



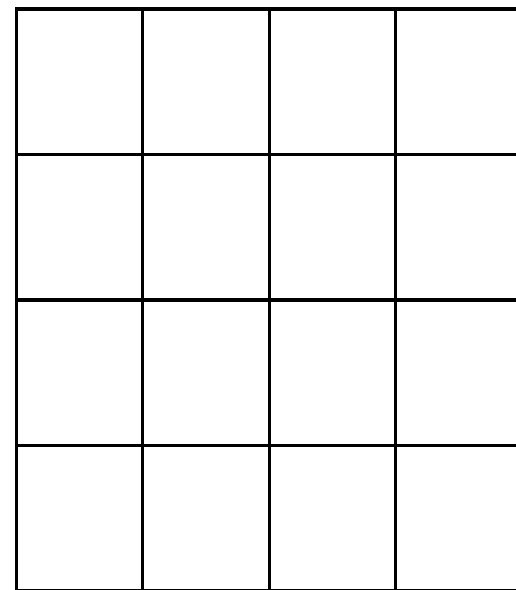


## 업샘플링: 전치 합성곱

- 3x3 합성곱, 스트라이드 1, 패드 1



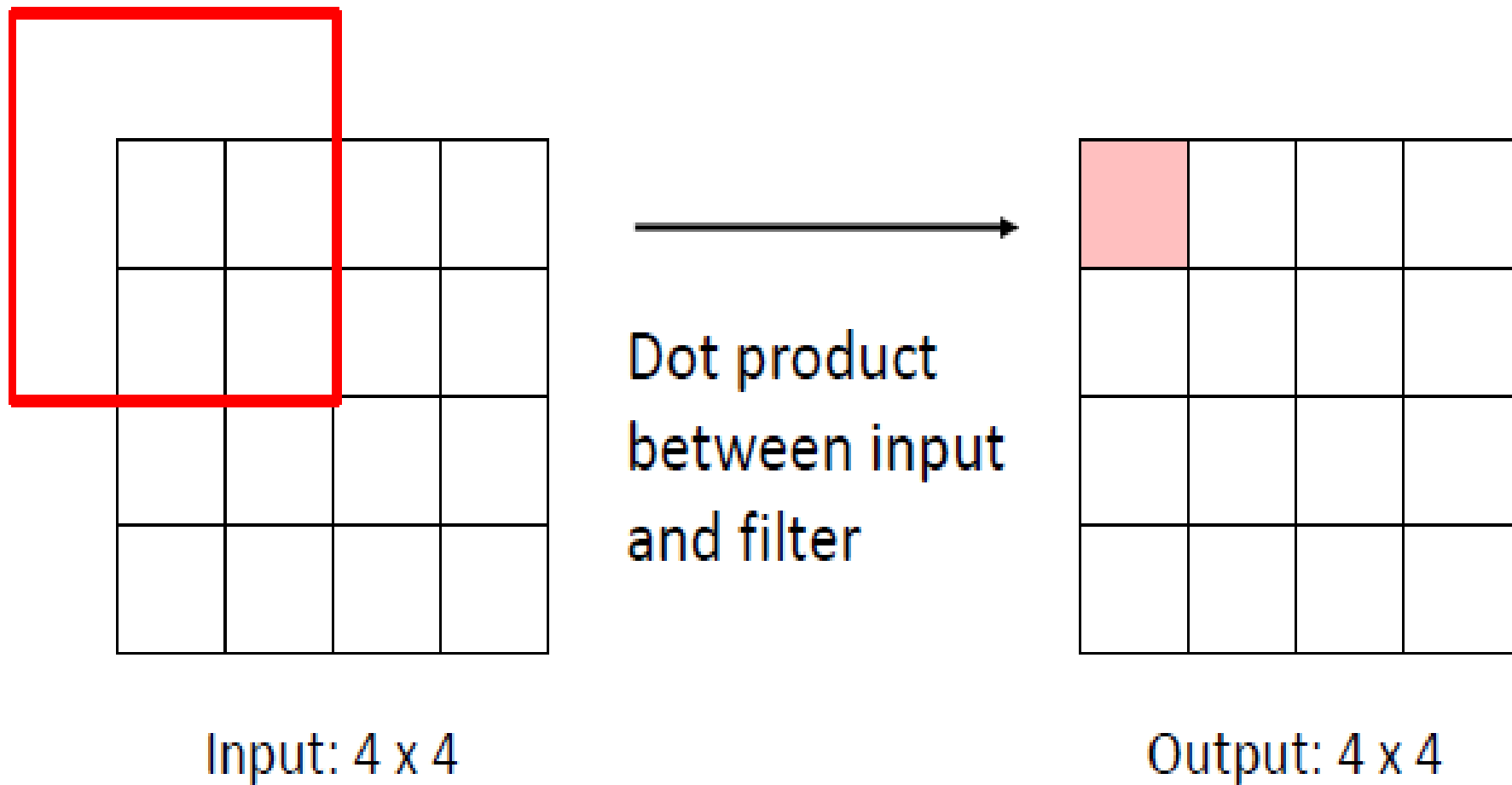
Input: 4 x 4



Output: 4 x 4

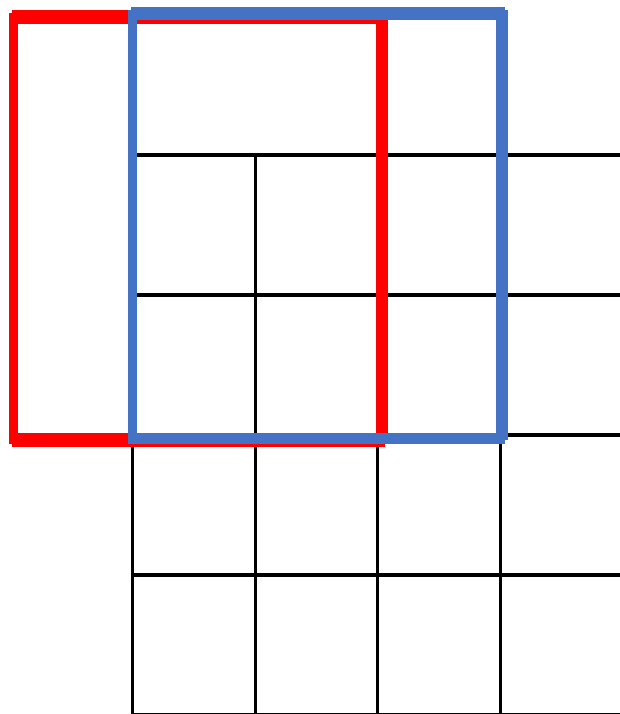
## 업샘플링: 전치 합성곱

- 3x3 합성곱, 스트라이드 1, 패드 1



## 업샘플링: 전치 합성곱

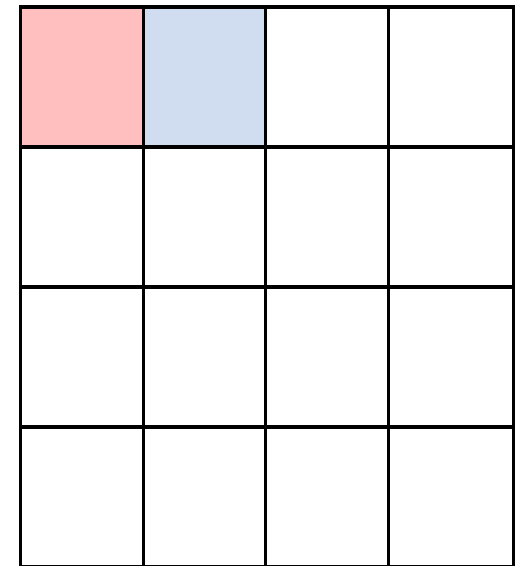
- 3x3 합성곱, 스트라이드 1, 패드 1



Input: 4 x 4



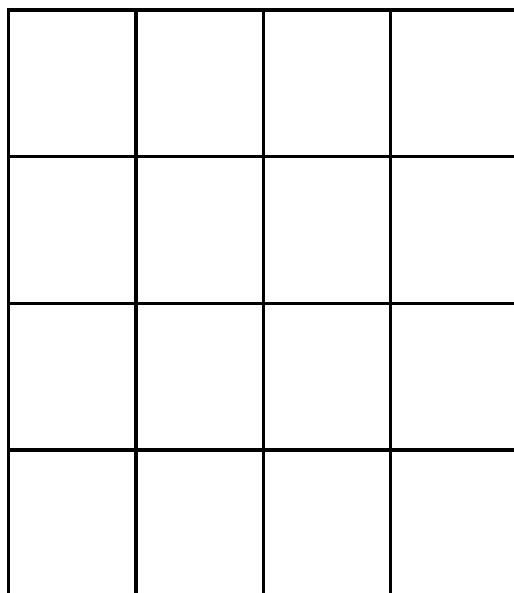
Dot product  
between input  
and filter



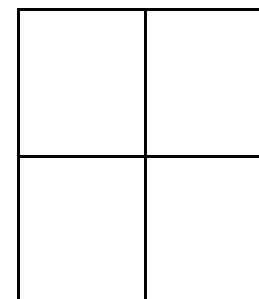
Output: 4 x 4

## 업샘플링: 전치 합성곱

- 3x3 합성곱, 스트라이드 2, 패드 1



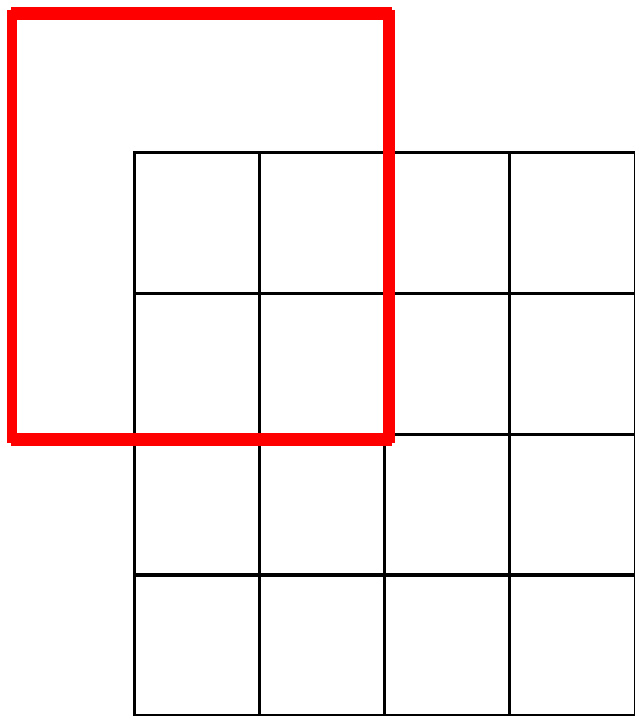
Input: 4 x 4



Output: 2 x 2

## 업샘플링: 전치 합성곱

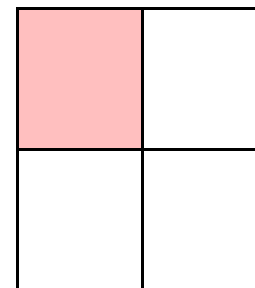
- 3x3 합성곱, 스트라이드 2, 패드 1



Input: 4 x 4



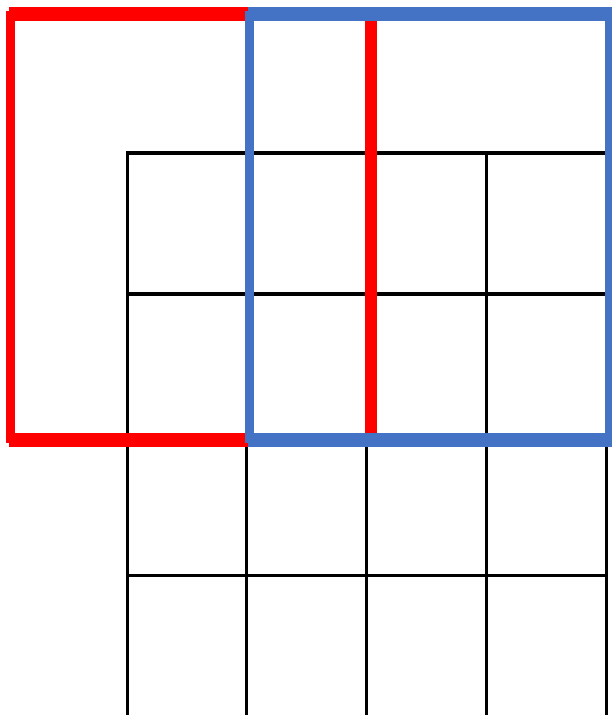
Dot product  
between input  
and filter



Output: 2 x 2

## 업샘플링: 전치 합성곱

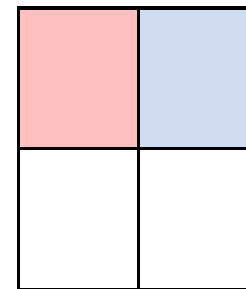
- 3x3 합성곱, 스트라이드 2, 패드 1



Input: 4 x 4



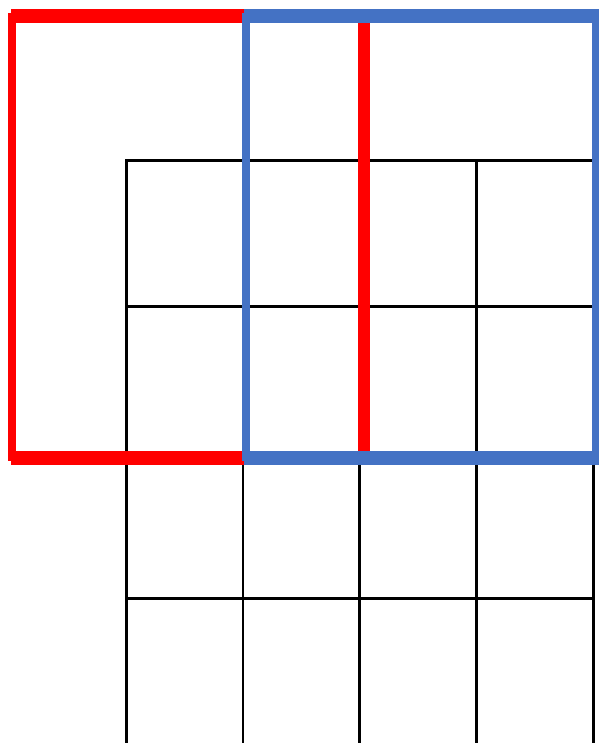
Dot product  
between input  
and filter



Output: 2 x 2

## 업샘플링: 전치 합성곱

- 3x3 합성곱, 스트라이드 2, 패드 1

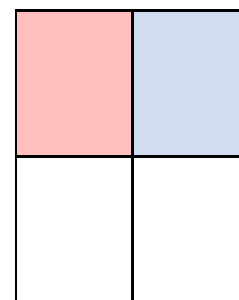


Input: 4 x 4

Convolution with stride  $> 1$  is “Learnable Downsampling”  
Can we use stride  $< 1$  for “Learnable Upsampling”?



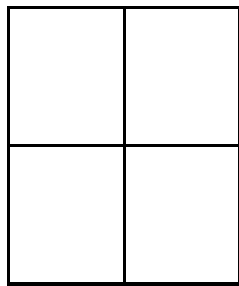
Dot product  
between input  
and filter



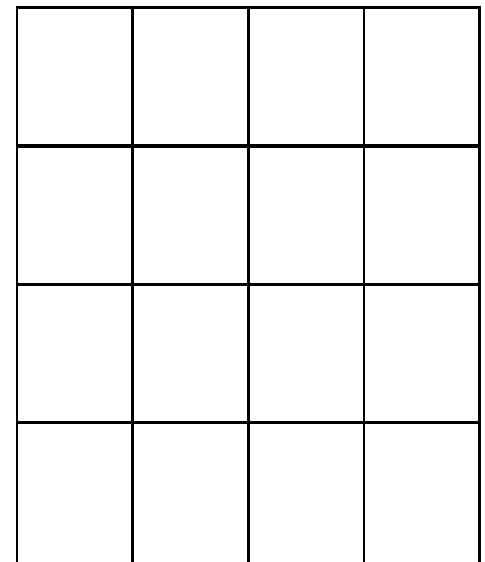
Output: 2 x 2

## 업샘플링: 전치 합성곱

- 3x3 합성곱, 스트라이드 2



Input: 2 x 2

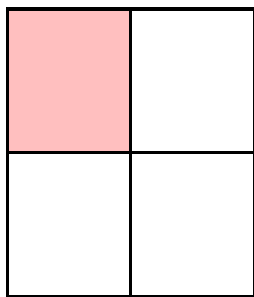


Output: 4 x 4



## 업샘플링: 전치 합성곱

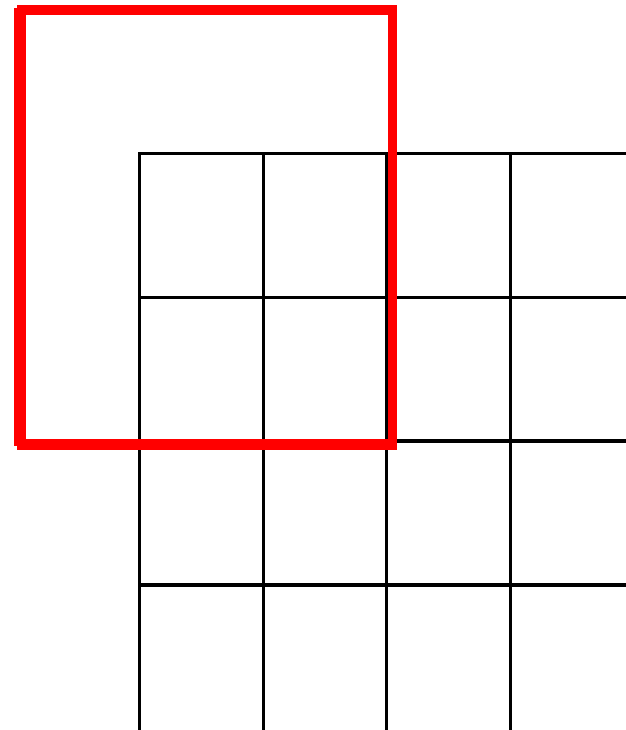
- 3x3 합성곱, 스트라이드 2



Input: 2 x 2



Weight filter by  
input value and  
copy to output

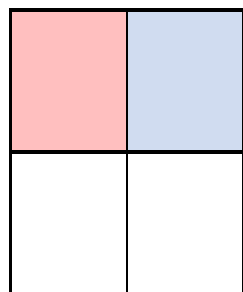


Output: 4 x 4

## 업샘플링: 전치 합성곱

- 3x3 합성곱, 스트라이드 2

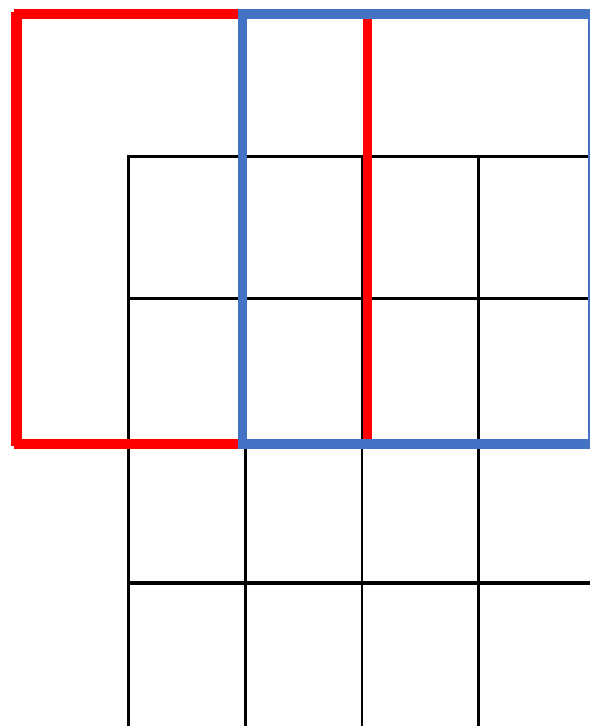
Filter moves 2 pixels in output  
for every 1 pixel in input



Input: 2 x 2



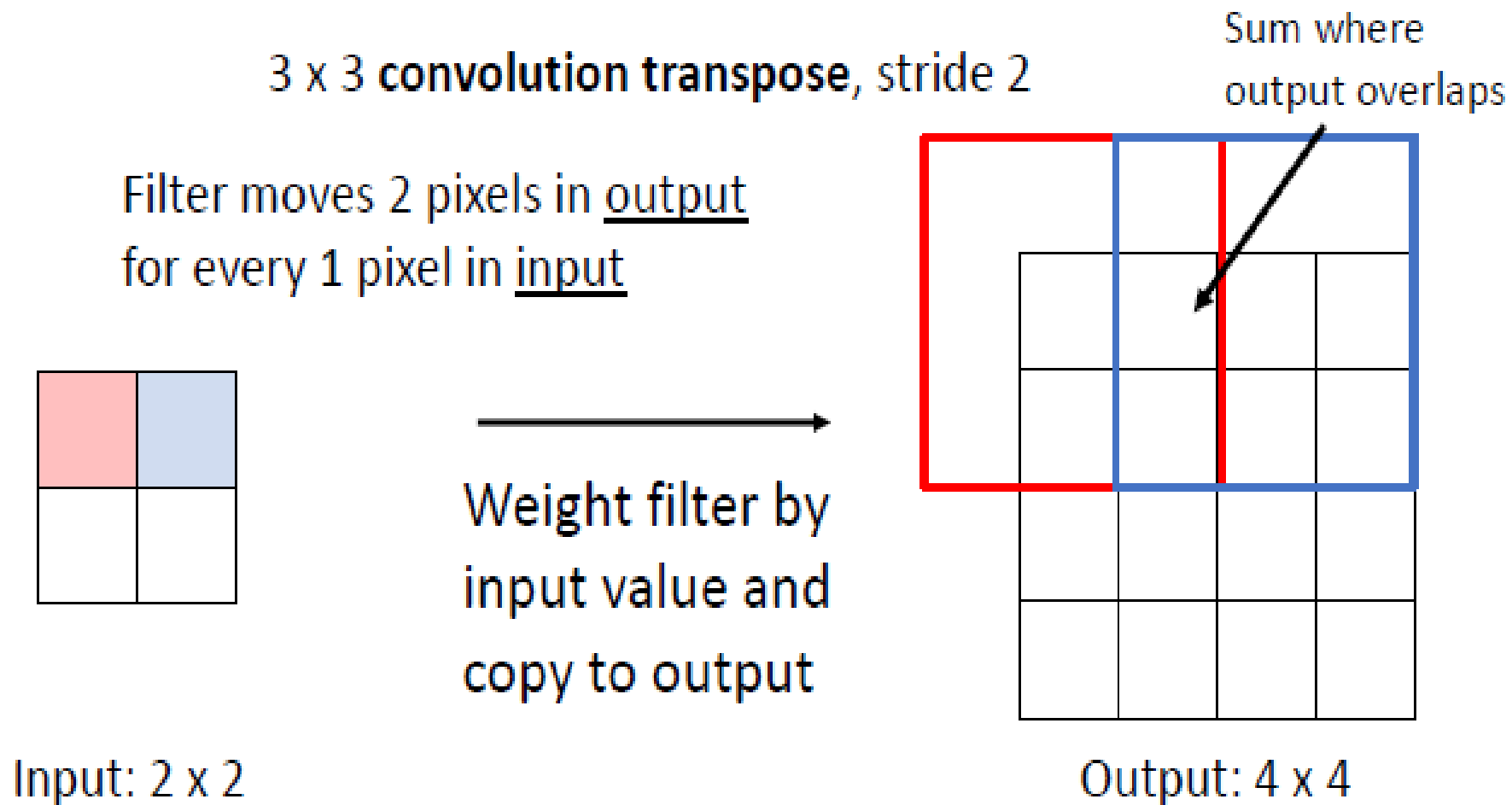
Weight filter by  
input value and  
copy to output



Output: 4 x 4

## 업샘플링: 전치 합성곱

- 3x3 합성곱, 스트라이드 2

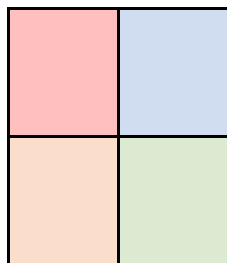


## 업샘플링: 전치 합성곱

- 3x3 합성곱, 스트라이드 2

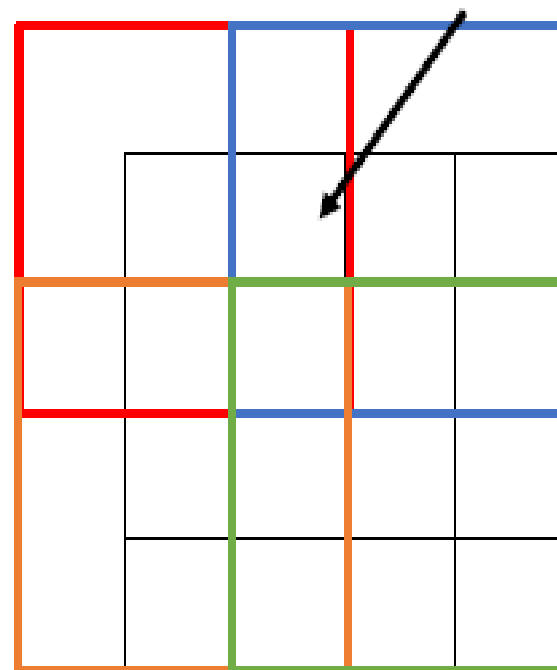
3 x 3 **convolution transpose**, stride 2

This gives 5x5 output – need to trim one pixel from top and left to give 4x4 output



Input: 2 x 2

→  
Weight filter by  
input value and  
copy to output

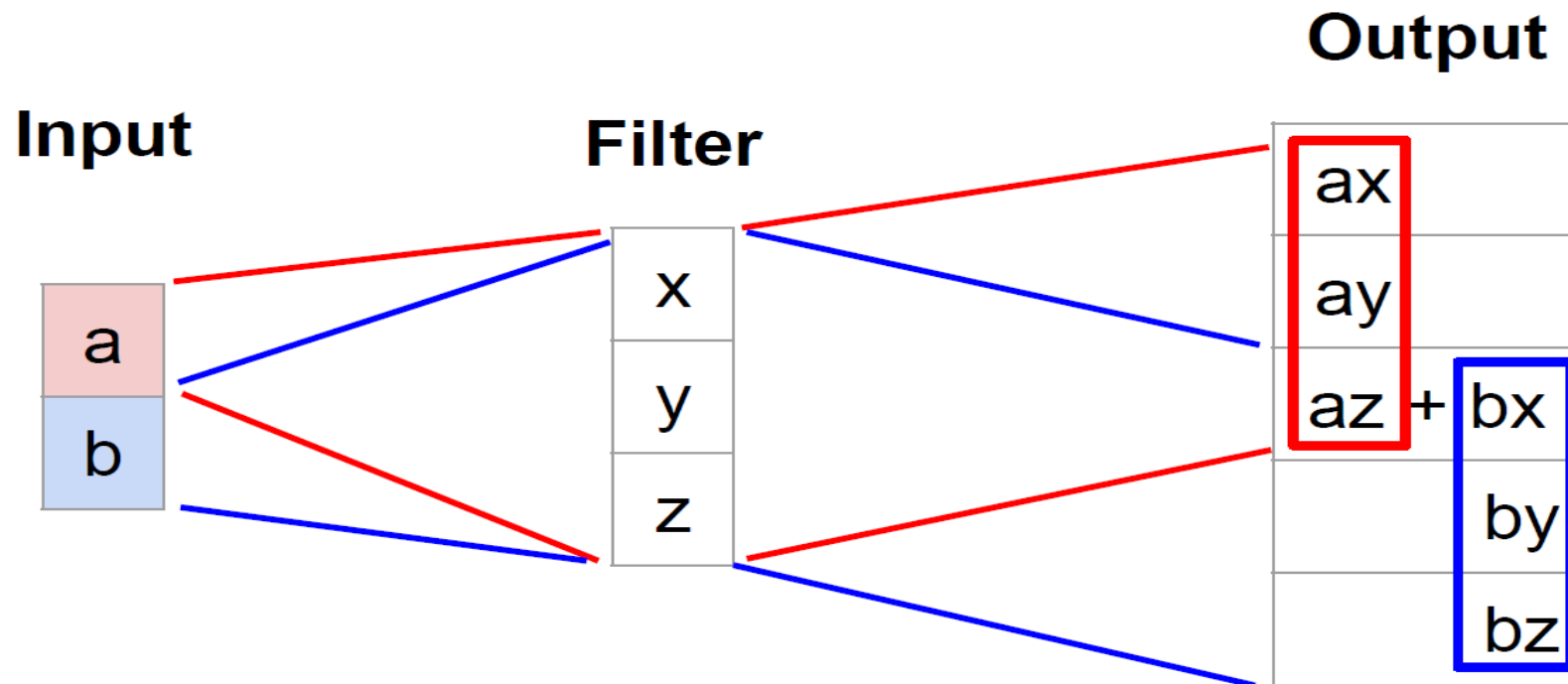


Sum where  
output overlaps

Output: 4 x 4

# 업샘플링

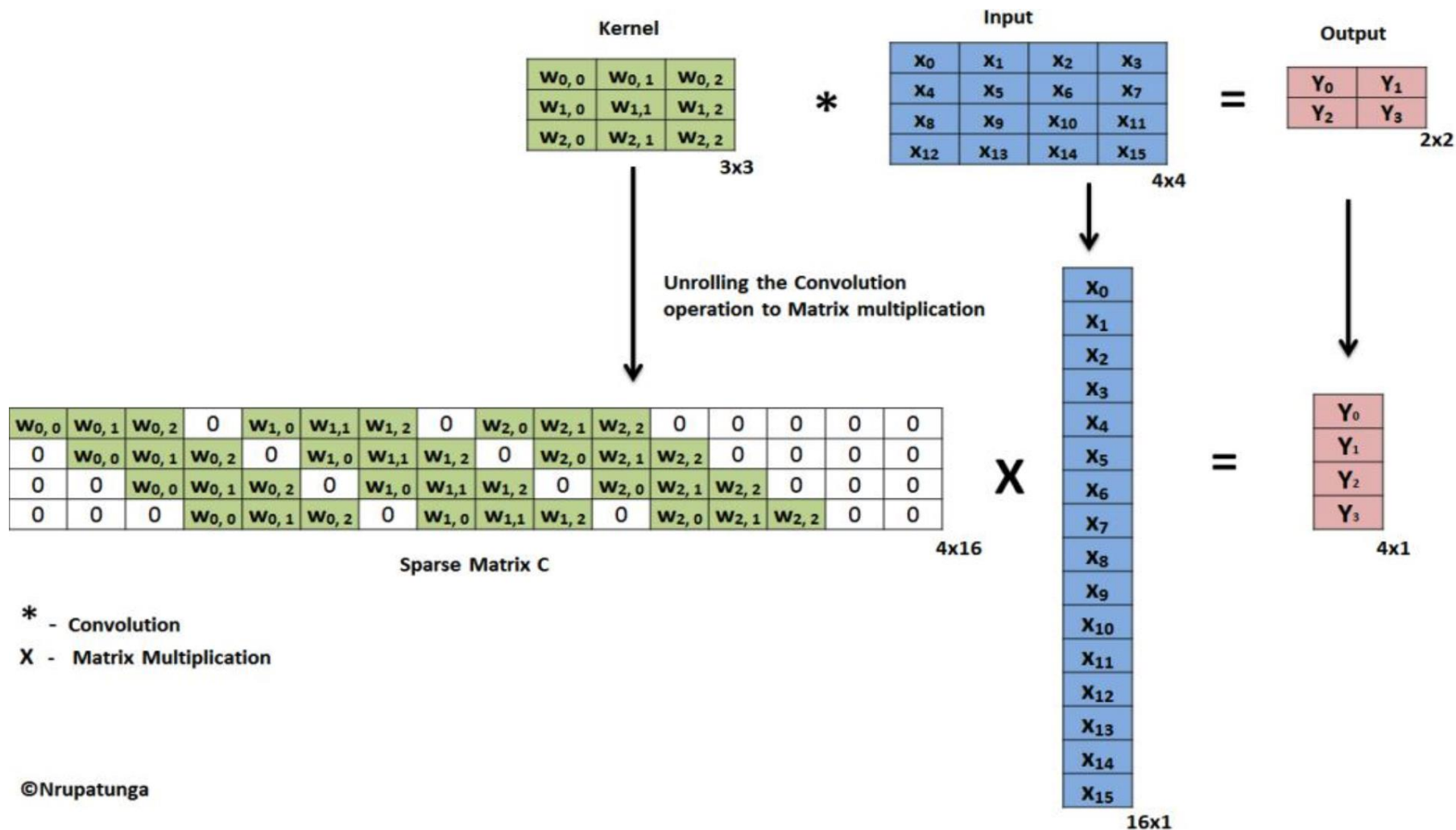
- Transpose 업샘플링: 1d 예시



- 위의 경우, 3X1 필터로 stride=2인 경우로 정확하게 2배 ( $2 \times 2 = 4$ )로 만들기 위해서는 output의 한 픽셀을 제거할 필요가 있음.

# 전치 업샘플링

- 왜 전치인가?



# 전치 업샘플링

- 왜 전치인가?

$w_{0,0}$	0	0	0
$w_{0,1}$	$w_{0,0}$	0	0
$w_{0,2}$	$w_{0,1}$	$w_{0,0}$	0
0	$w_{0,2}$	$w_{0,1}$	$w_{0,0}$
$w_{1,0}$	0	$w_{0,2}$	$w_{0,1}$
$w_{1,1}$	$w_{1,0}$	0	$w_{0,2}$
$w_{1,2}$	$w_{1,1}$	$w_{1,0}$	0
0	$w_{1,2}$	$w_{1,1}$	$w_{1,0}$
$w_{2,0}$	0	$w_{1,2}$	$w_{1,1}$
$w_{2,1}$	$w_{2,0}$	0	$w_{1,2}$
$w_{2,2}$	$w_{2,1}$	$w_{2,0}$	0
0	$w_{2,2}$	$w_{2,1}$	$w_{2,0}$
0	0	$w_{2,2}$	$w_{2,1}$
0	0	0	$w_{2,2}$
0	0	0	0
0	0	0	0

Sparse Matrix  $C^T$

16x4

**X**

$Y_0$
$Y_1$
$Y_2$
$Y_3$

4x1

**=**

$x_0$
$x_1$
$x_2$
$x_3$
$x_4$
$x_5$
$x_6$
$x_7$
$x_8$
$x_9$
$x_{10}$
$x_{11}$
$x_{12}$
$x_{13}$
$x_{14}$
$x_{15}$

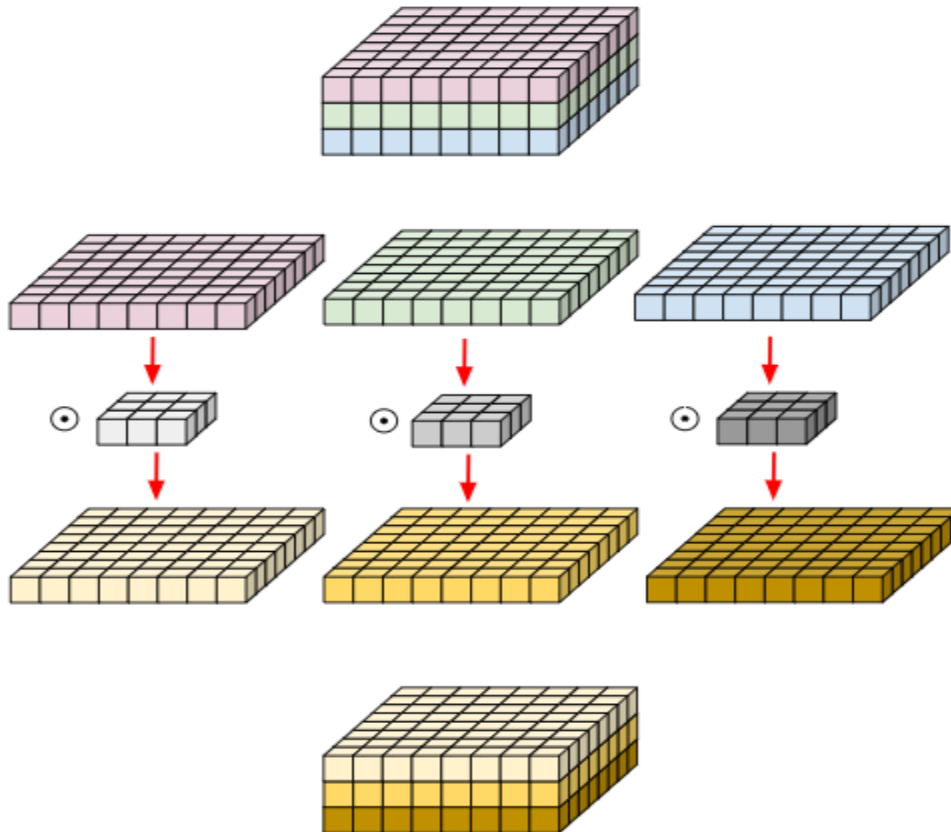
16x1

## 채널별 개별 포인트 합성곱



# 채널별 합성곱

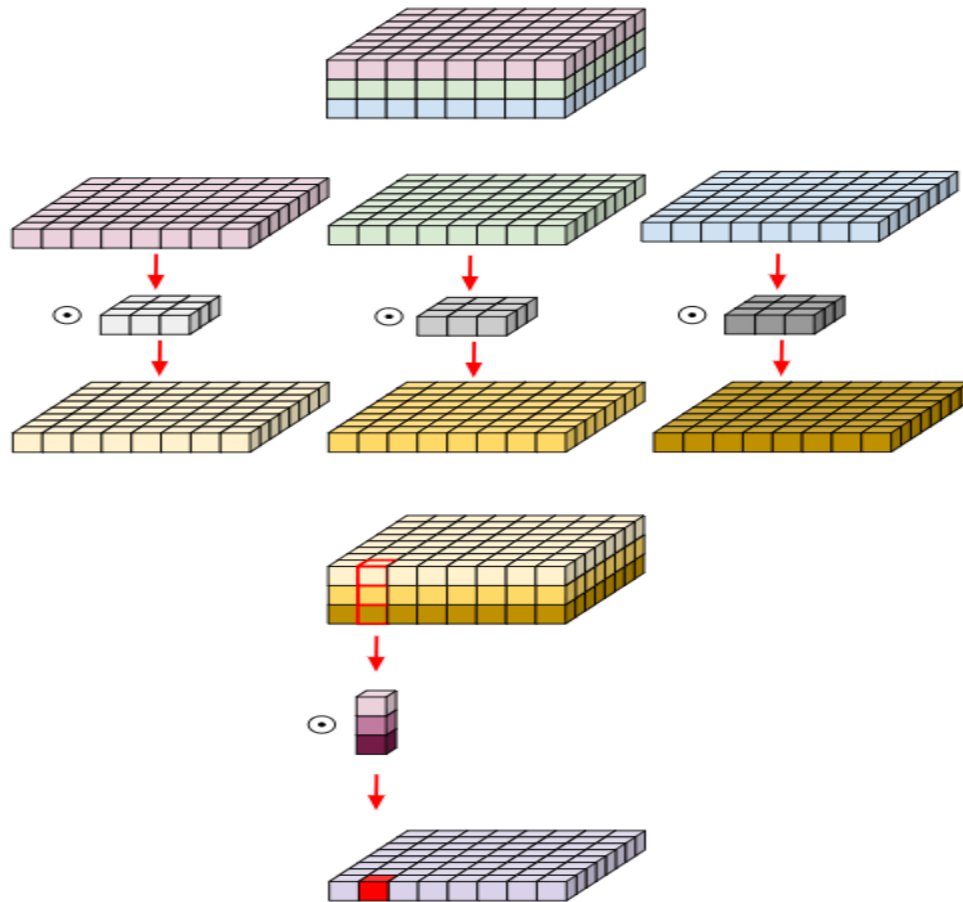
- 채널별 합성곱 (Depthwise Convolution)



- 채널별로 합성곱 실행 즉 채널방향이 아니라 공간 방향으로만 합성곱실행

# 채널별 합성곱 + 채널방향 합성곱

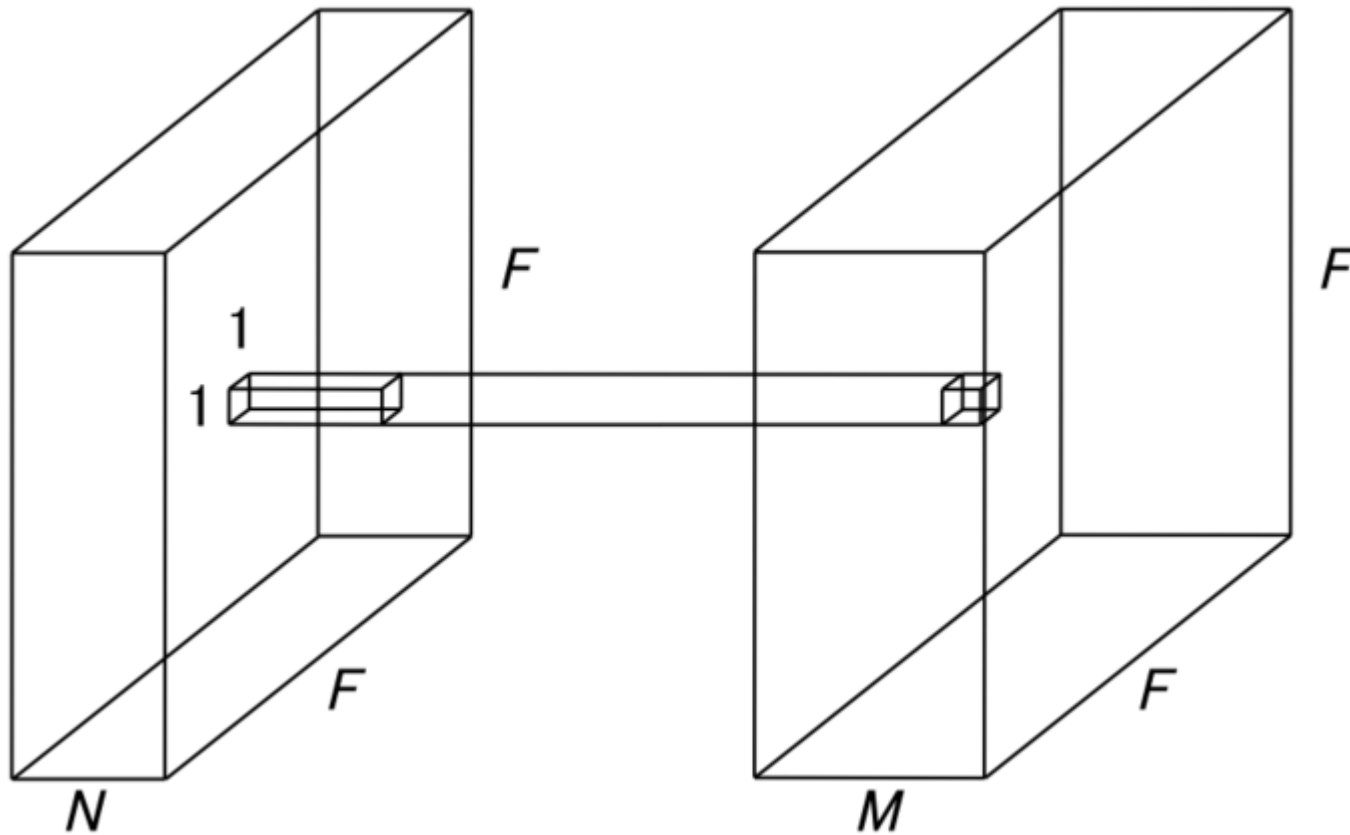
- 채널별 합성곱 이후 채널방향 개별 포인트 합성곱 (Depthwise Separable Convolution)



- 1단계에서는 공간방향으로 2단계에서는 채널방향으로 합성곱 실행한다.

# 개별 포인트 합성곱

- 개별 포인트 합성곱 (Pointwise Convolution)



- 채널 방향으로 압축을 해 채널 수를 줄인다.