CNN

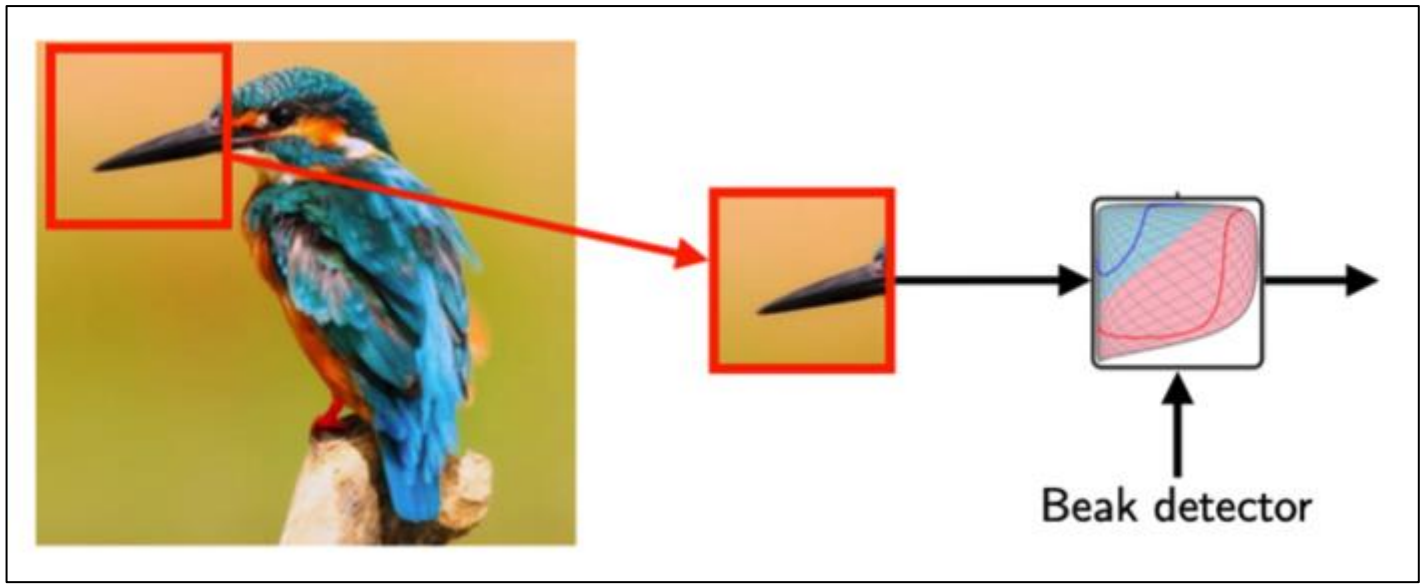
조상연

Convolution Neural Network(CNN)

- 기존의 Fully-Connected 모델은 1차원의 데이터 말고 2차원 이상의 데이터를 사용하게 된다면, 해당 입력 데이터를 Flatten시켜 한 줄의 데이터로 만들어야 한다.
- 이 과정에서 데이터의 손상이 발생하게 된다.
  - 이미지의 경우에는 상하좌우 이웃 픽셀의 정보가 손실된다.
- 위 문제를 해결하기 위해 고안한 해결책이 바로 CNN이다.

- 단순 Fully-connected 보다 학습시킬 weight가 적다.
- 학습과 연산에 속도가 빠르며, 효율적이다.
- 이미지나 영상데이터를 처리할 때 사용한다.

CNN 접근방법



어떠한 이미지가 주어졌을때,
새인지 아닌지를 결정하는 모델을 만든다 가정한다면,

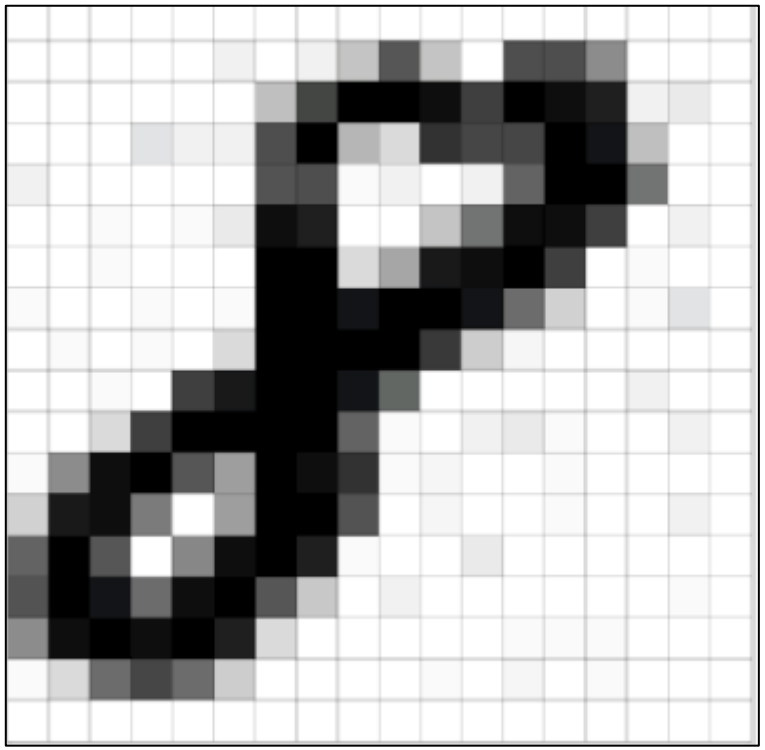
해당 이미지에서 새의 특징(Feature)이라고 할 수 있는 “부리”를 찾는 것이 중요한 포인트일 것이다.
즉, 해당 이미지에서 새의 부리를 찾는것이다.

따라서 이미지 전체를 보기보다는 특징을 찾아내는 작은 부분(Filter)을 사용한다.

https://halfundecided.medium.com/%EB%94%A5%EB%9F%AC%EB%8B%9D-%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-cnn-convolutional-neural-networks-%EC%89%BD%EA%B2%8C-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0-836869f88375

이미지 표현 방법

Mnist datasets 손글씨 8



손으로 작성한 숫자 8 이미지를
픽셀단위로 쪼개서 보며는,

28x28단위의 행렬로 표현할 수 있겠다.
즉, 이미지를 2차원의 Matrix로 표현한 것이다.

그렇다면 임의의 크기의 이미지를 다음과 같이 표현 할 수 있겠다.

5x5크기의 이미지 Matrix

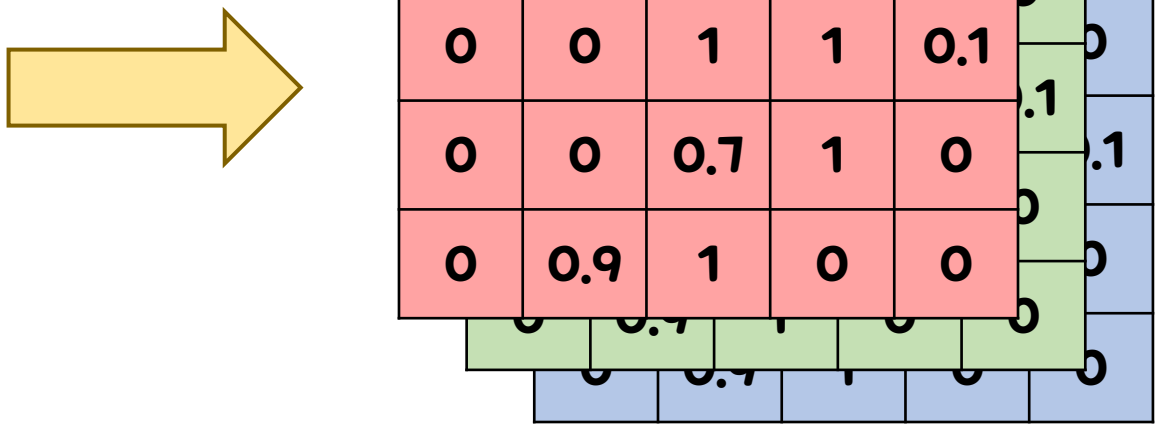
1	0.4	1	0	0
0	1	0.2	0.6	0
0	0	1	1	0.1
0	0	0.7	1	0
0	0.9	1	0	0

https://halfundecided.medium.com/%EB%94%A5%EB%9F%AC%EB%8B%9D-%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-cnn-convolutional-neural-networks-%EC%89%BD%EA%B2%8C-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0-836869f88375

이미지 표현 방법

뿐만 아니라 흔히 사용되는 컬러이미지의 경우에는
3차원의 Matrix로 표현된다.
Channel x Width x Height 총 3개의 차원이다.
Channel은 RGB 3개이다.

컬러 이미지



https://halfundecided.medium.com/%EB%94%A5%EB%9F%AC%EB%8B%9D-%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-cnn-convolutional-neural-networks-%EC%89%BD%EA%B2%8C-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0-836869f88375

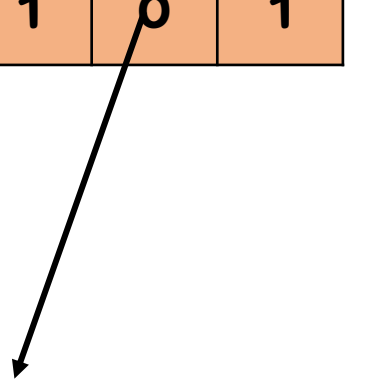
합성 곱 연산(Convolution Dot)

5x5크기의 이미지 Matrix

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

3x3크기의 Filter

1	0	1
0	1	0
1	0	1



필터의 각각의 값은 가중치로서 학습이 되는 파라미터이다.

앞서 언급했던 특징을 추출하기위한
작은 필터를 구성한다고 했는데,

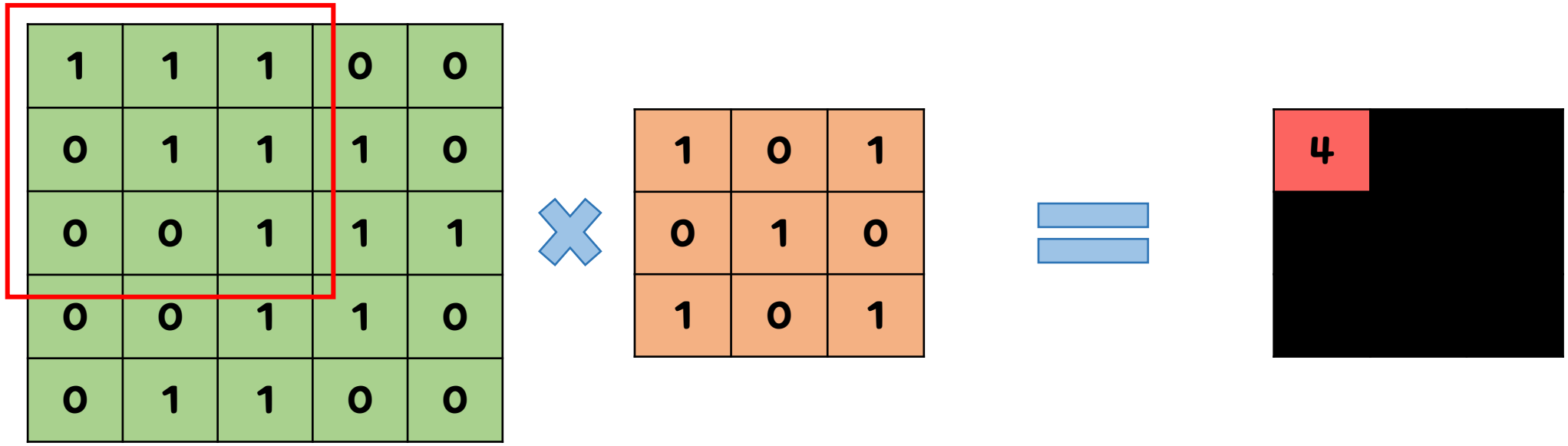
어떠한 특징을 찾아내는 일정한 크기를 가진 필터를 구성한다.
해당 필터는 이미지 전체를 훑으면서 이미지의 모든 영역에서
해당되는 특징(Feature)를 찾아낸다.

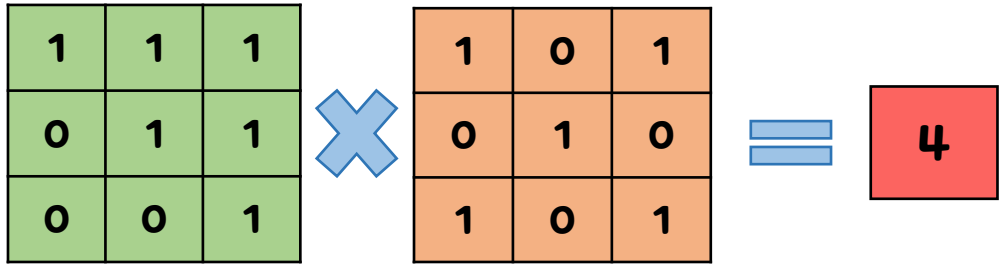
정확히는 필터와 이미지간의 내적연산을 처리하는 것이다.

합성 곱 연산(Convolution Dot)

Stride : filter가 이미지를 훑을 때의 간격

Stride : 1x1

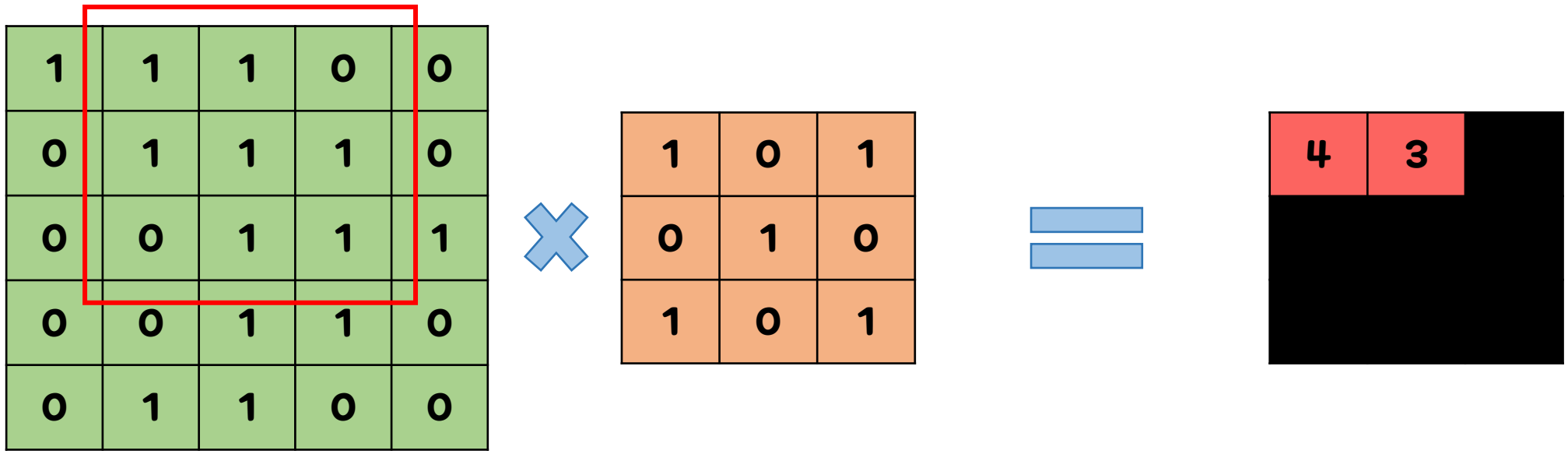


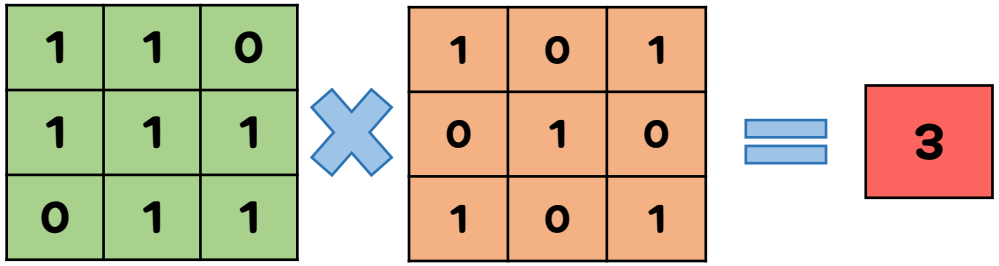


합성 곱 연산(Convolution Dot)

Stride : filter가 이미지를 훑을 때의 간격

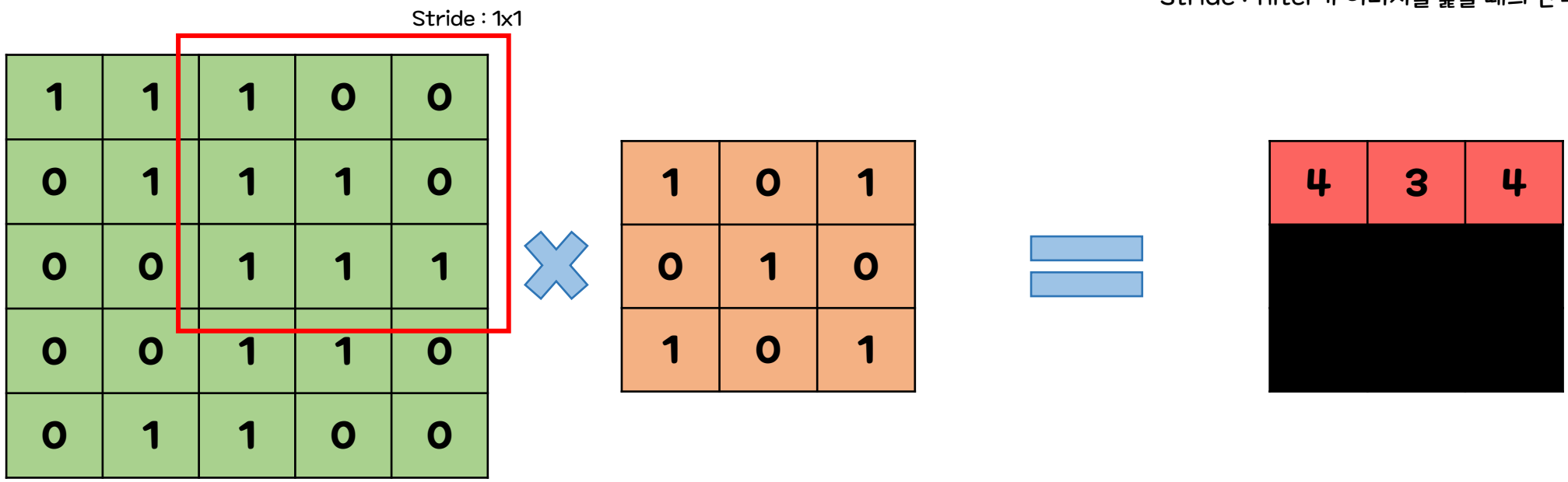
Stride : 1x1

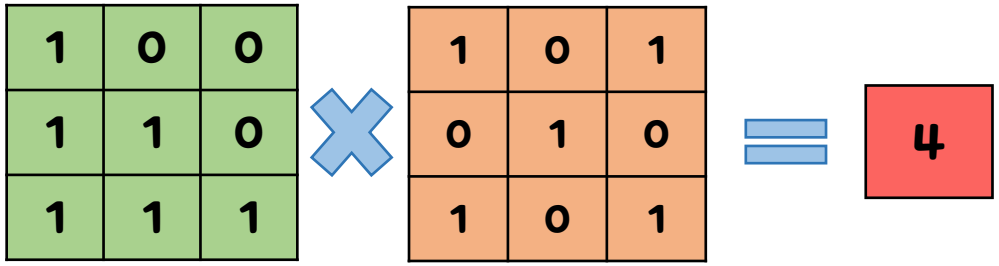




합성 곱 연산(Convolution Dot)

Stride : filter가 이미지를 훑을 때의 간격

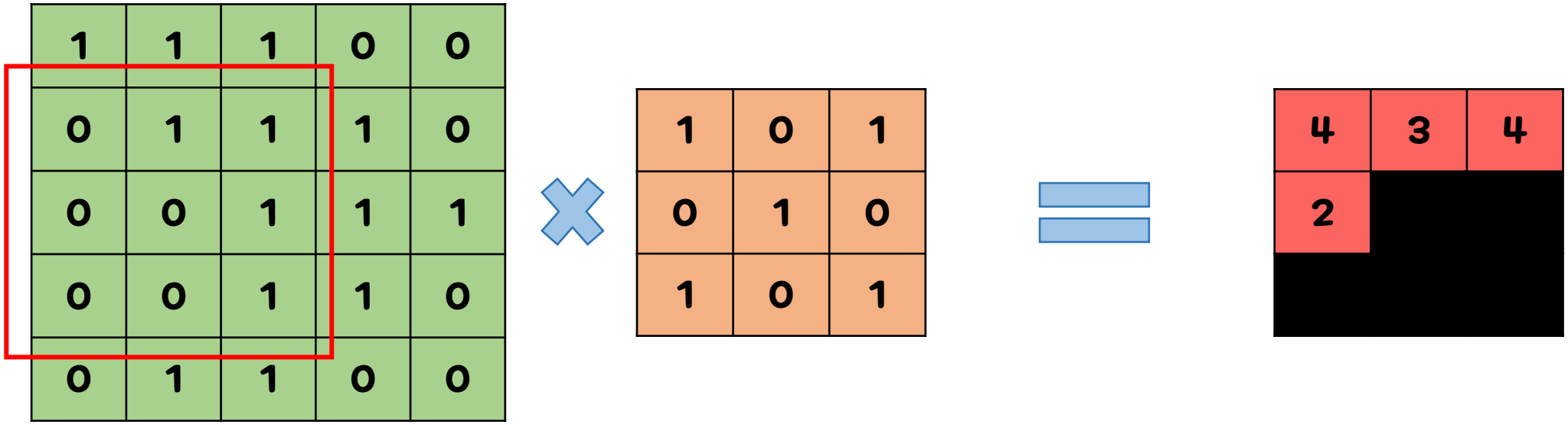


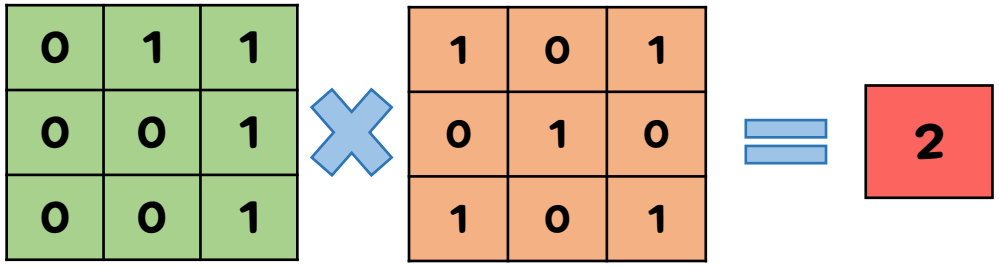


합성 곱 연산(Convolution Dot)

Stride : 1x1

Stride : filter가 이미지를 훑을 때의 간격

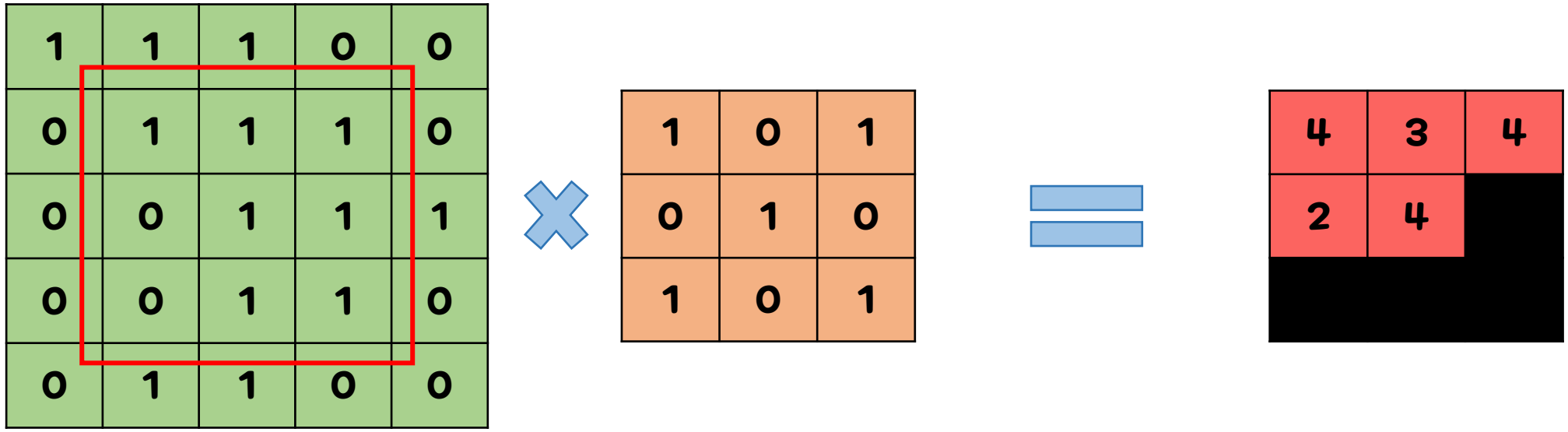


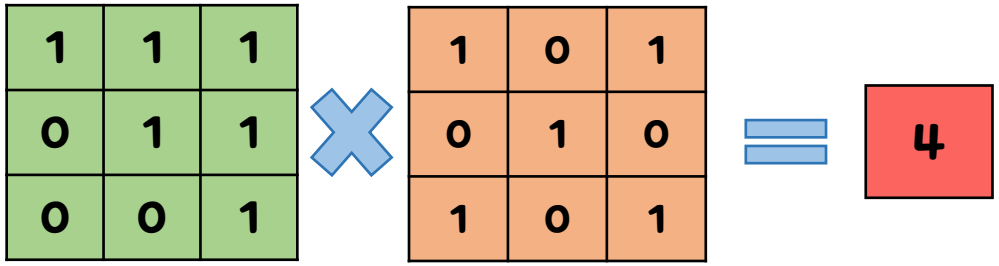


합성 곱 연산(Convolution Dot)

Stride : 1x1

Stride : filter가 이미지를 훑을 때의 간격

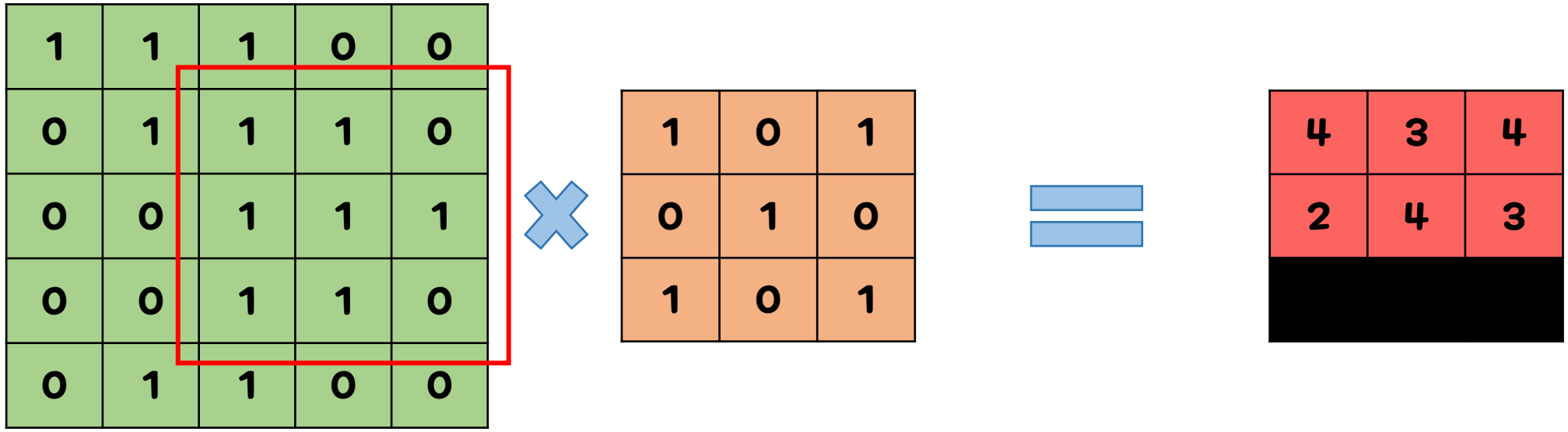


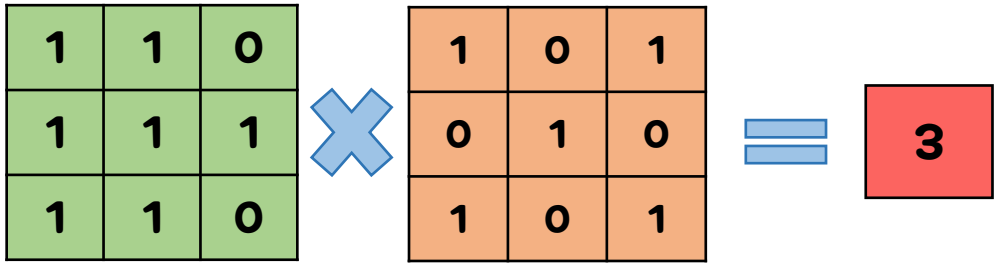


합성 곱 연산(Convolution Dot)

Stride : 1x1

Stride : filter가 이미지를 훑을 때의 간격

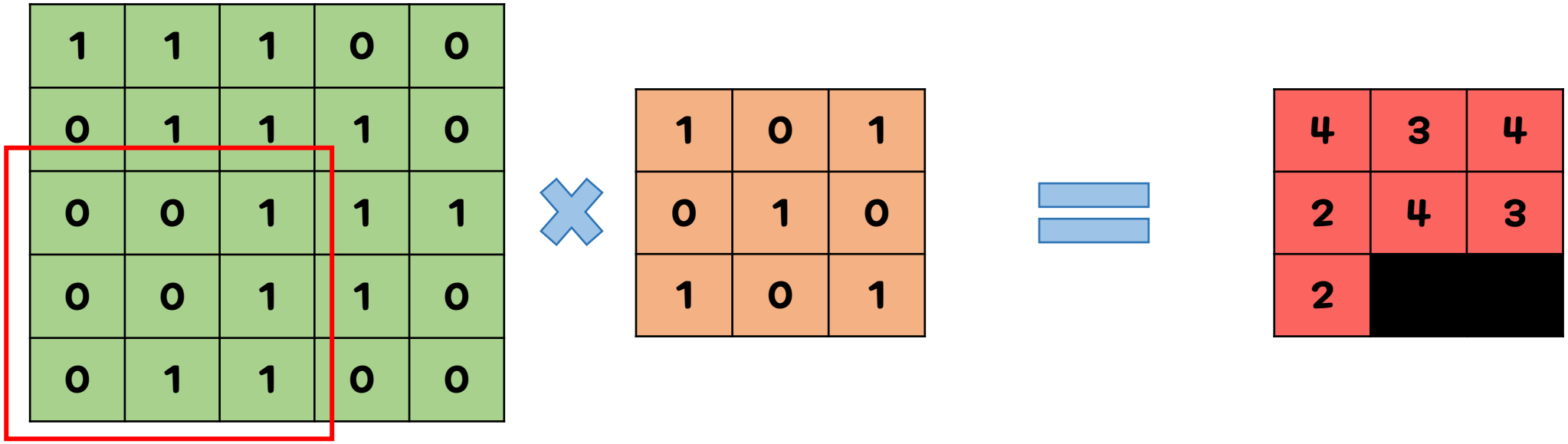


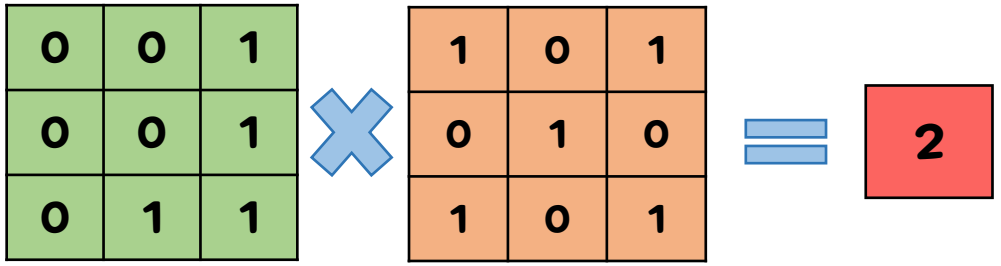


합성 곱 연산(Convolution Dot)

Stride : 1x1

Stride : filter가 이미지를 훑을 때의 간격

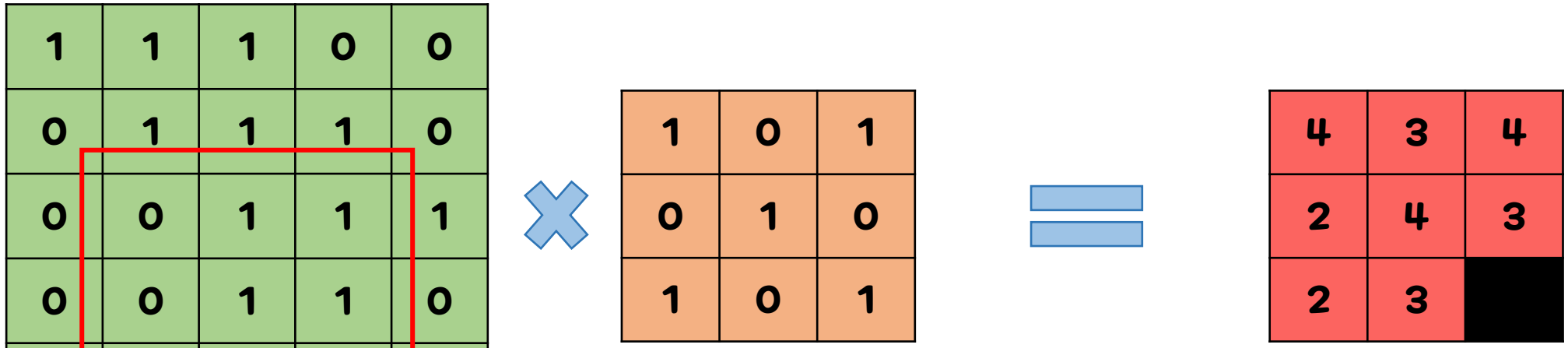


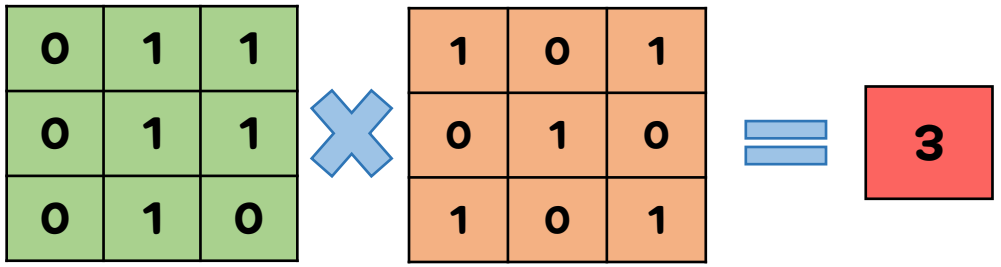


합성 곱 연산(Convolution Dot)

Stride : 1x1

Stride : filter가 이미지를 훑을 때의 간격

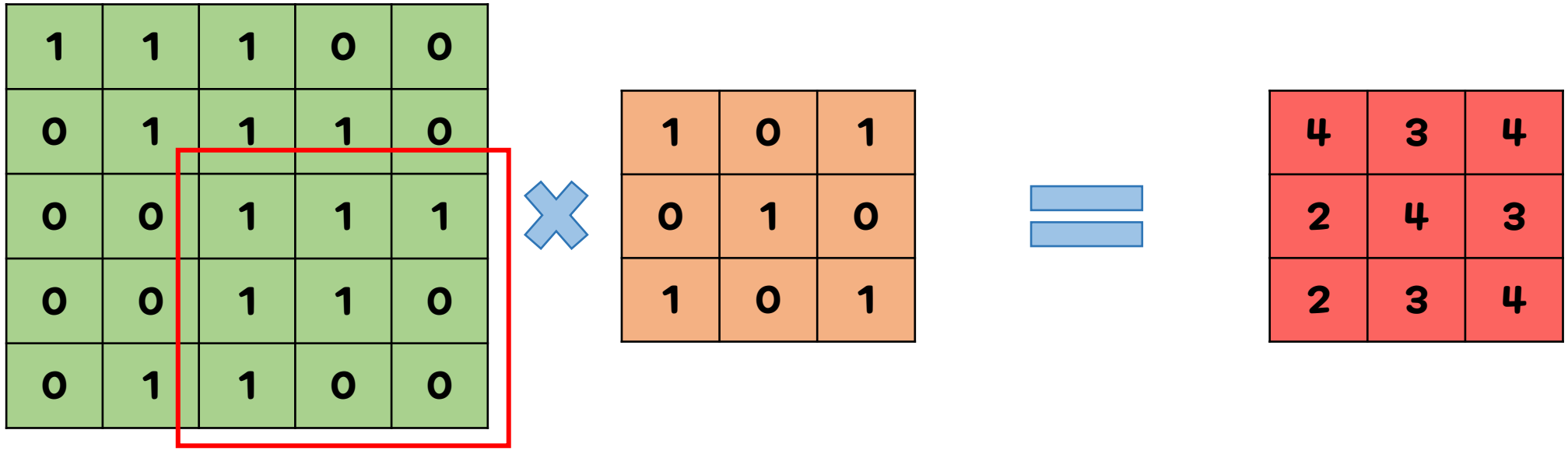


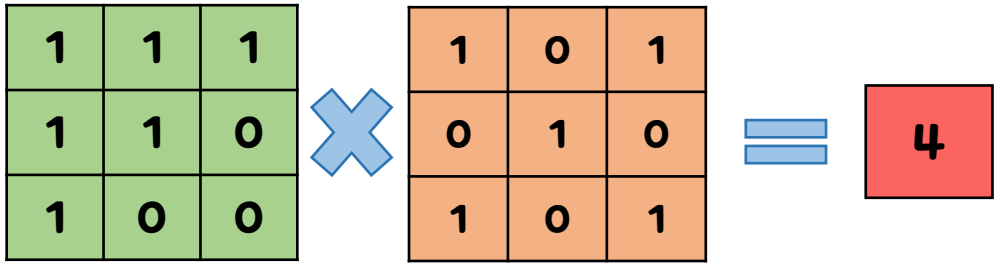


합성 곱 연산(Convolution Dot)

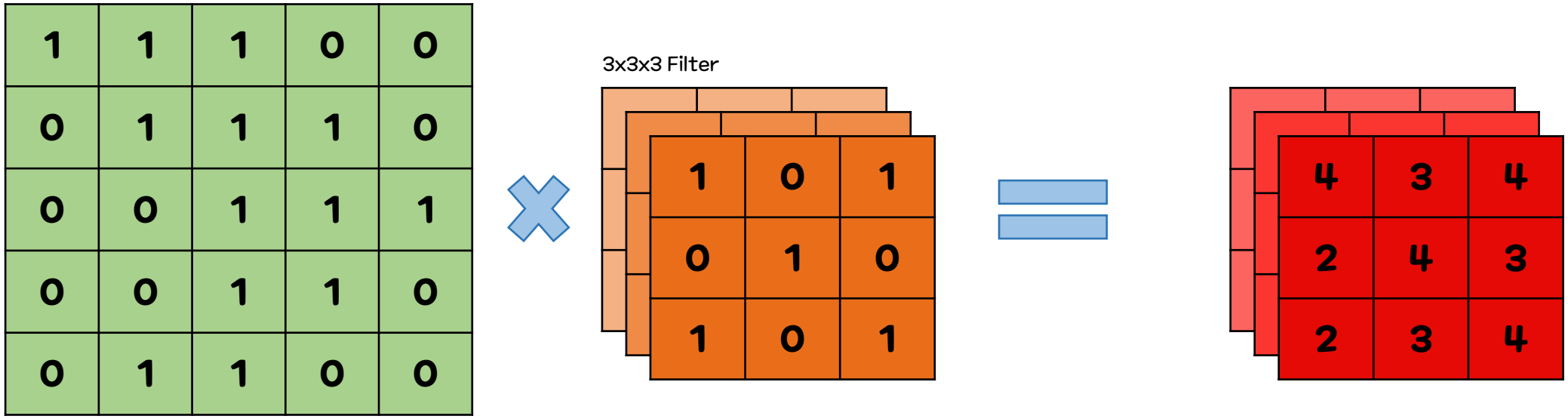
Stride : 1x1

Stride : filter가 이미지를 훑을 때의 간격



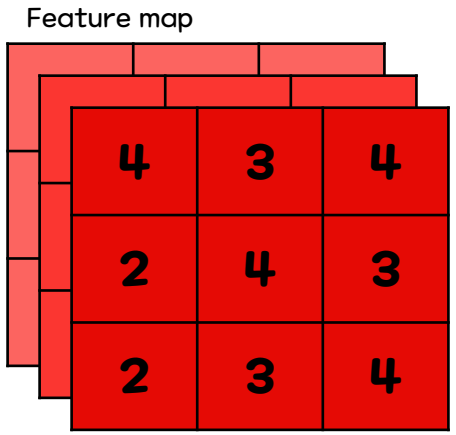


합성 곱 연산(Convolution Dot)



Filter를 여러 개 사용하면 각각의 필터에 대한 결과값 또한 여러 개가 될 것이다.

합성 곱 연산(Convolution Dot)

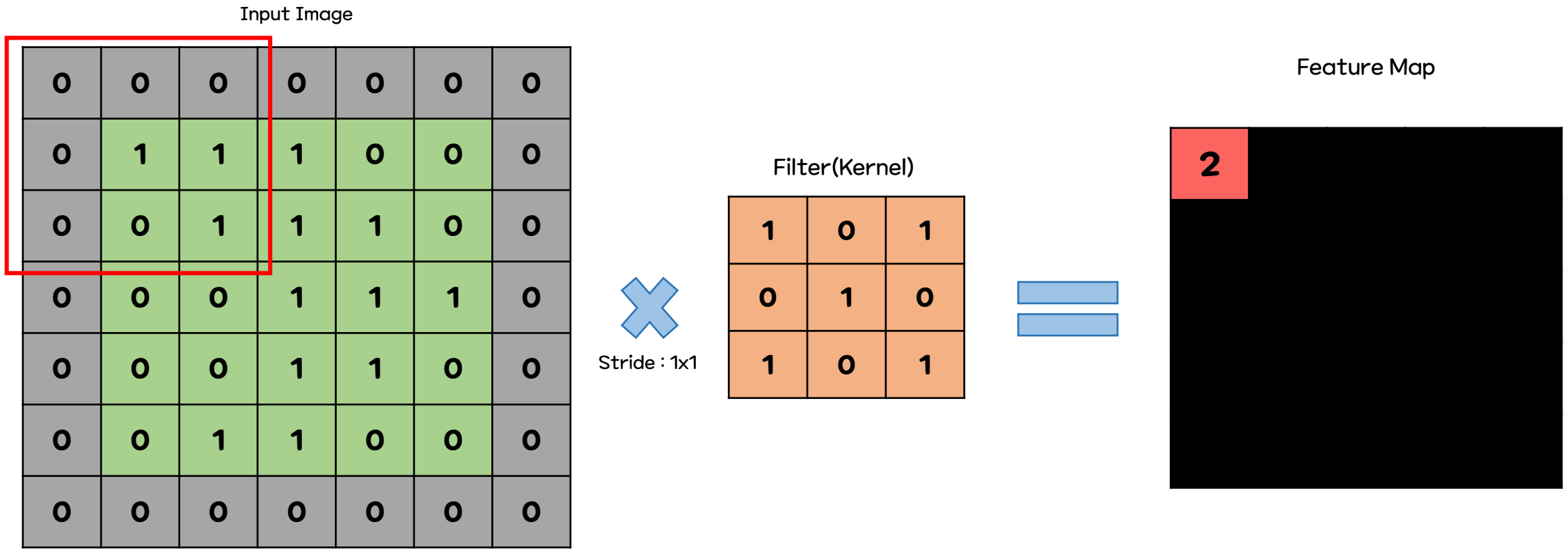


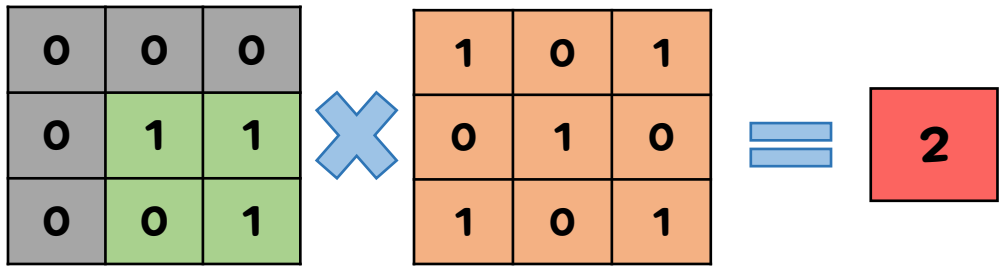
이미지와 필터의 연산을 통해 나온 결과값을 Feature map이라 하며,
Feature map 의 크기는
((Image - Filter) / Stride) + 1 이다.

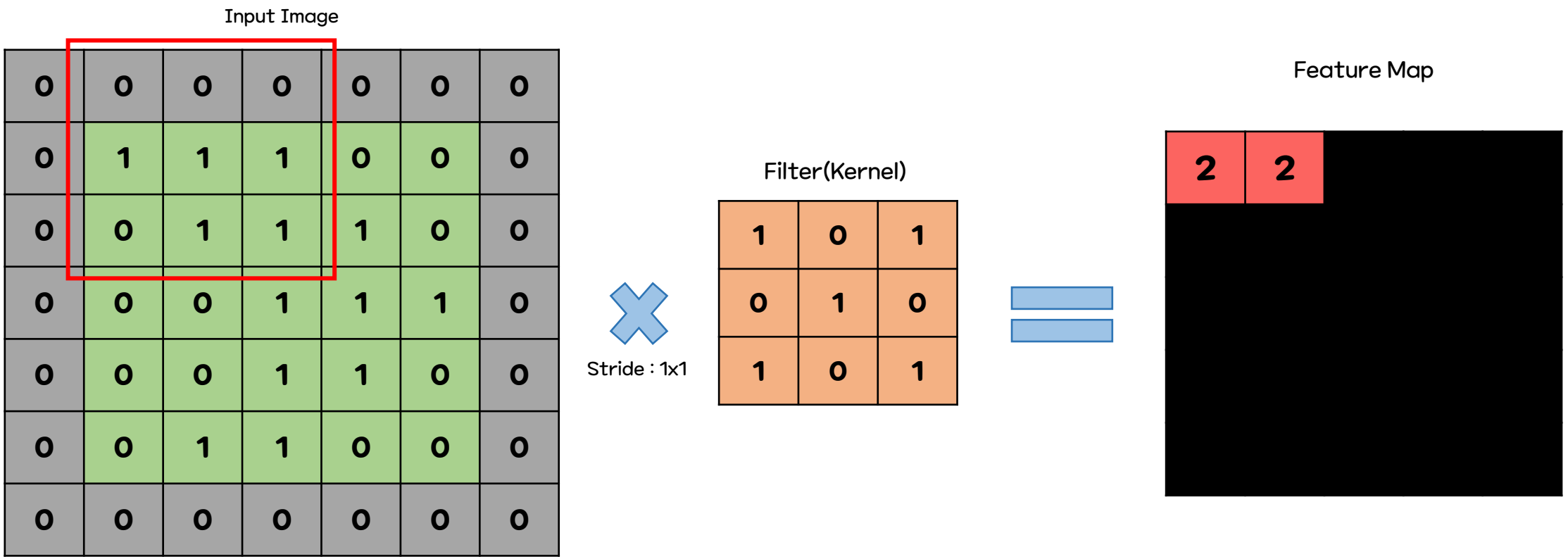
$$Feature\ Map = \frac{(Image\ -filter)}{Stride} + 1$$

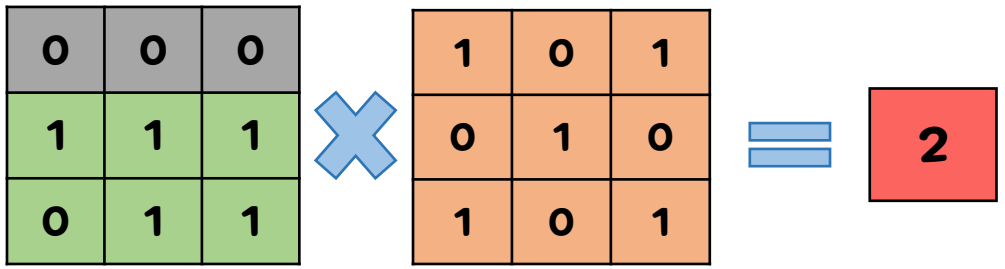
겁듭해서 filter를 거치게 되면, 점점 feature map의 크기가 작아지는 데,,, 이는 Padding이라는 방법을 통해 해결 할 수 있다.
Padding이 있는 경우 : ((Image - Filter + 2*Pad) / Stride) + 1

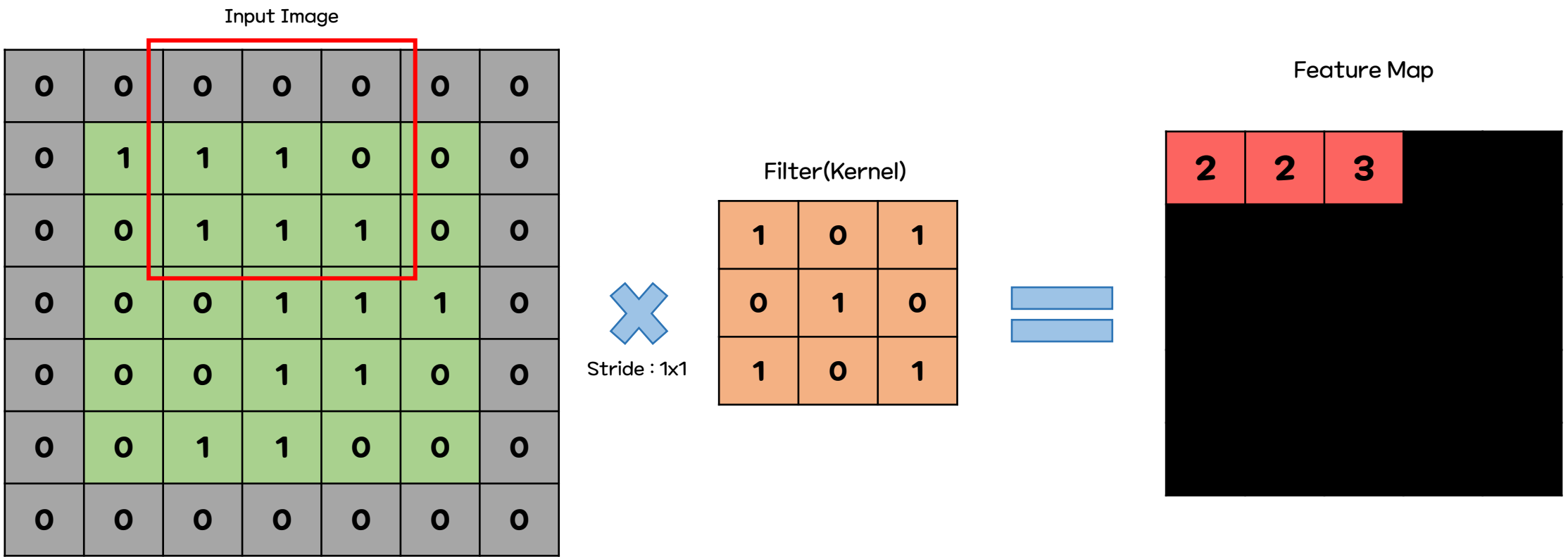
$$Feature\ Map = \frac{(Image\ -filter + 2Pad)}{Stride} + 1$$

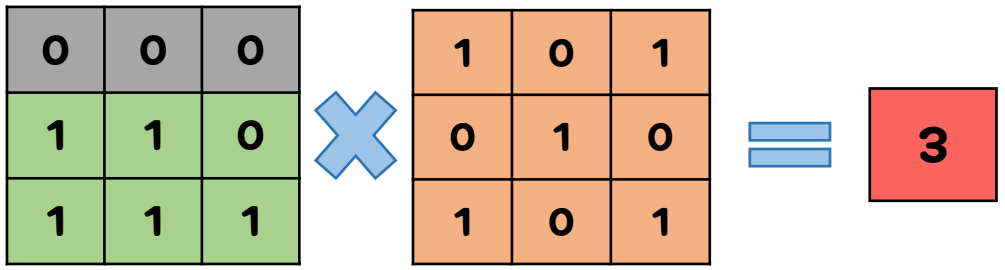

Input Image
0 0 0 0 0 0 0
0 1 1 1 0 0 0
0 0 1 1 1 0 0
0 0 0 1 1 1 0
0 0 0 1 1 0 0
0 0 1 1 0 0 0
0 0 0 0 0 0 0
Stride : 1x1
Filter(Kernel)
1 0 1
0 1 0
1 0 1
Feature Map
2

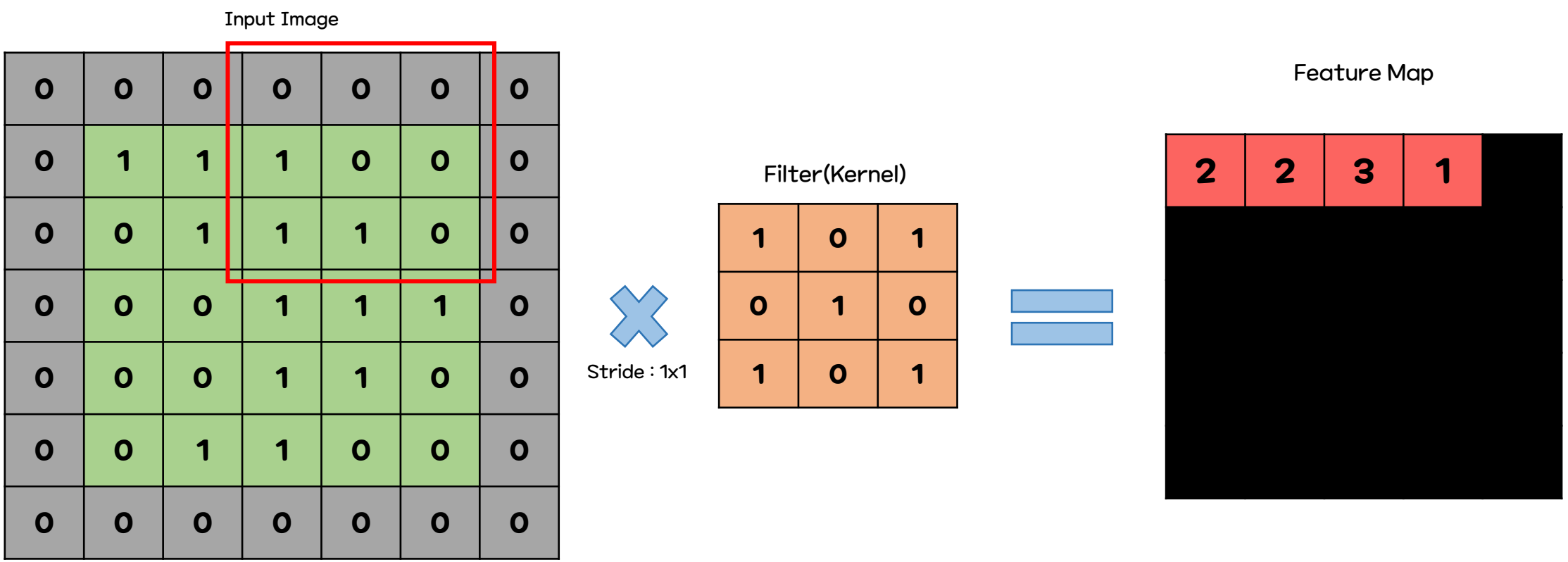

0 0 0
0 1 1
0 0 1
1 0 1
0 1 0
1 0 1
2

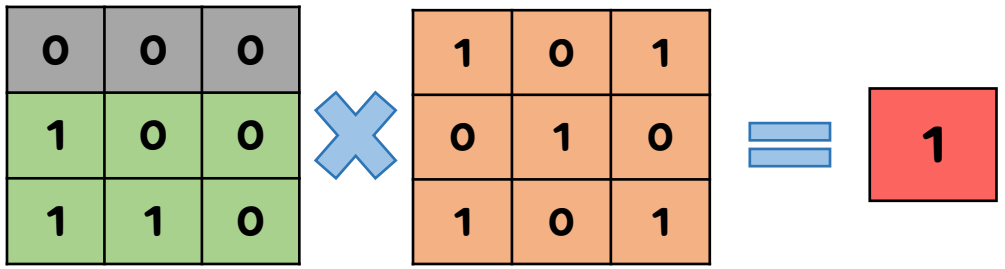

Input Image
0 0 0 0 0 0 0
0 1 1 1 0 0 0
0 0 1 1 1 0 0
0 0 0 1 1 1 0
0 0 0 1 1 0 0
0 0 1 1 0 0 0
0 0 0 0 0 0 0
Stride : 1x1
Filter(Kernel)
1 0 1
0 1 0
1 0 1
Feature Map
2 2

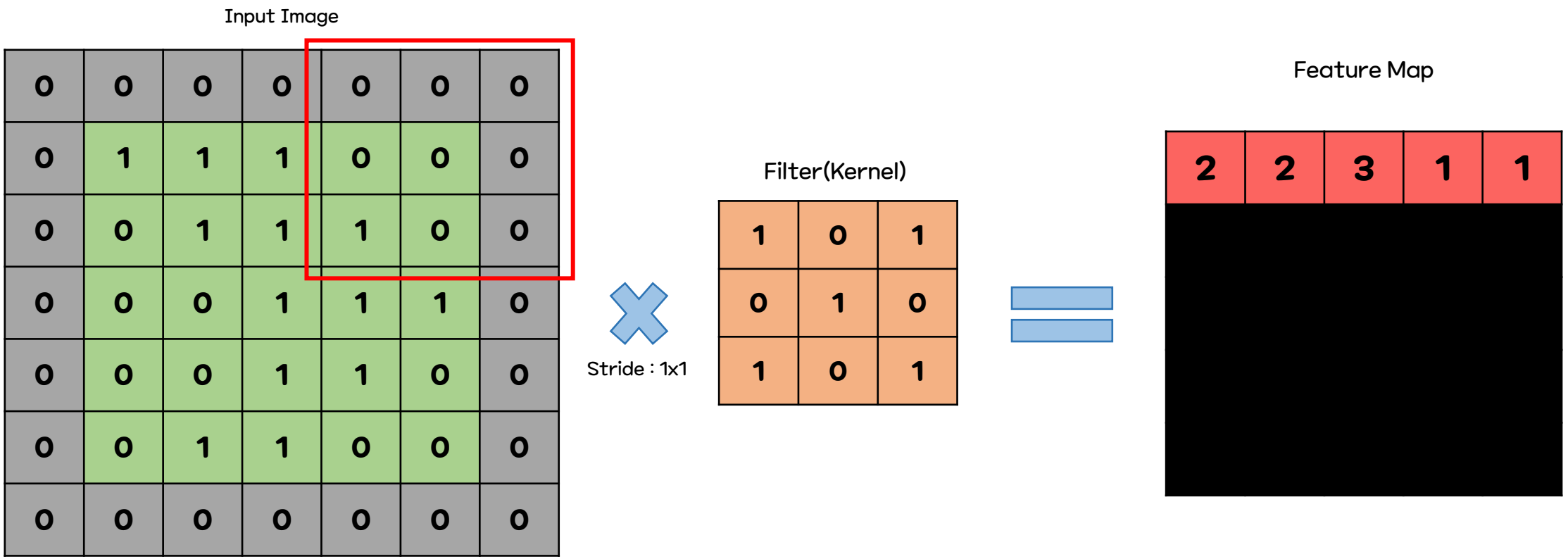

0 0 0
1 1 1
0 1 1
1 0 1
0 1 0
1 0 1
2

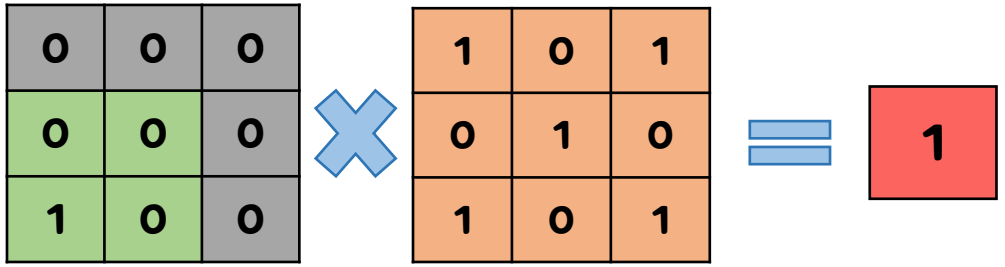

Input Image
0 0 0 0 0 0 0
0 1 1 1 0 0 0
0 0 1 1 1 0 0
0 0 0 1 1 1 0
0 0 0 1 1 0 0
0 0 1 1 0 0 0
0 0 0 0 0 0 0
Stride : 1x1
Filter(Kernel)
1 0 1
0 1 0
1 0 1
Feature Map
2 2 3

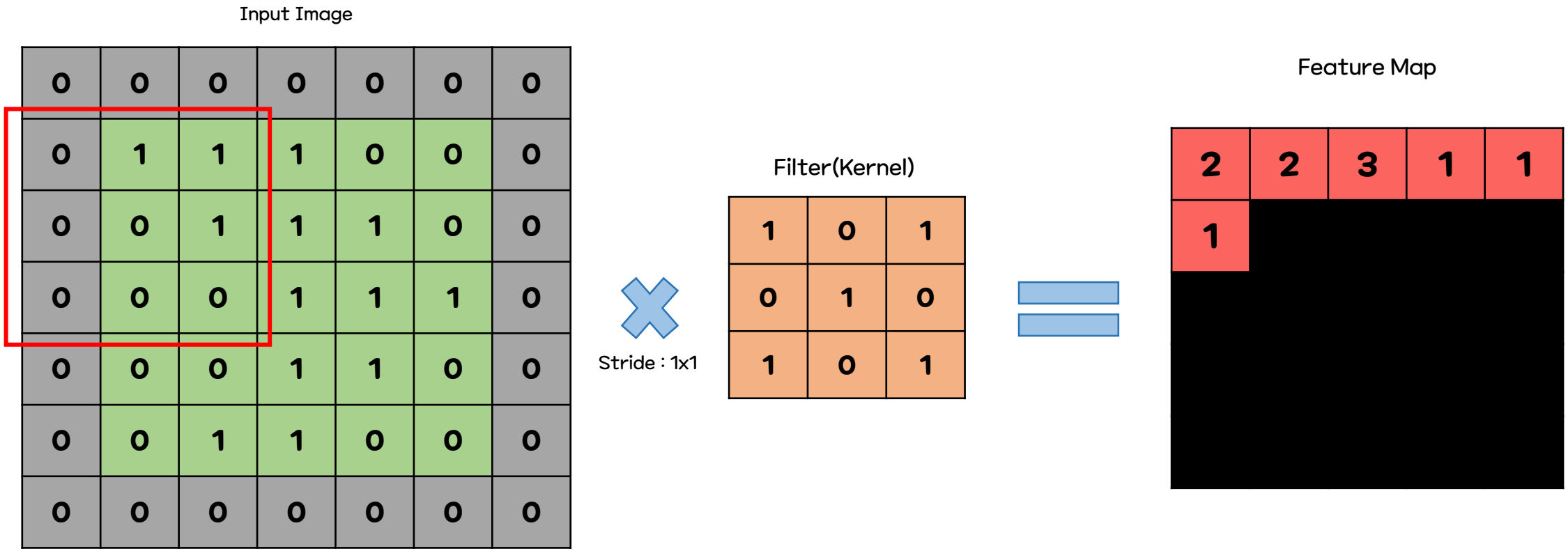

0 0 0
1 1 0
1 1 1
1 0 1
0 1 0
1 0 1
3

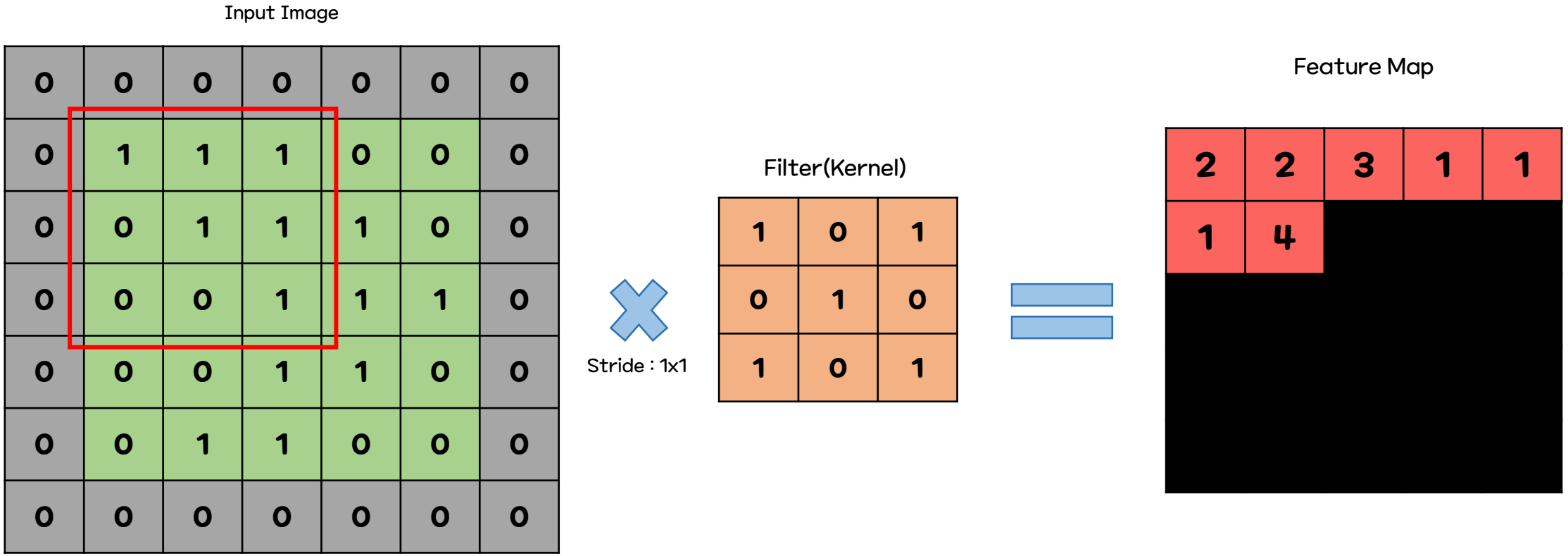

Input Image
0 0 0 0 0 0 0
0 1 1 1 0 0 0
0 0 1 1 1 0 0
0 0 0 1 1 1 0
0 0 0 1 1 0 0
0 0 1 1 0 0 0
0 0 0 0 0 0 0
Stride : 1x1
Filter(Kernel)
1 0 1
0 1 0
1 0 1
Feature Map
2 2 3 1


0 0 0
1 0 0
1 1 0
1 0 1
0 1 0
1 0 1
1


Input Image
0 0 0 0 0 0 0
0 1 1 1 0 0 0
0 0 1 1 1 0 0
0 0 0 1 1 1 0
0 0 0 1 1 0 0
0 0 1 1 0 0 0
0 0 0 0 0 0 0
Stride : 1x1
Filter(Kernel)
1 0 1
0 1 0
1 0 1
Feature Map
2 2 3 1 1


0 0 0
0 0 0
1 0 0
1 0 1
0 1 0
1 0 1
1


Input Image
0 0 0 0 0 0 0
0 1 1 1 0 0 0
0 0 1 1 1 0 0
0 0 0 1 1 1 0
0 0 0 1 1 0 0
0 0 1 1 0 0 0
0 0 0 0 0 0 0
Stride : 1x1
Filter(Kernel)
1 0 1
0 1 0
1 0 1
Feature Map
2 2 3 1 1
1
0 1 1
0 0 1
0 0 0
1 0 1
0 1 0
1 0 1
1


Input Image
0 0 0 0 0 0 0
0 1 1 1 0 0 0
0 0 1 1 1 0 0
0 0 0 1 1 1 0
0 0 0 1 1 0 0
0 0 1 1 0 0 0
0 0 0 0 0 0 0
Stride : 1x1
Filter(Kernel)
1 0 1
0 1 0
1 0 1
Feature Map
2 2 3 1 1
1 4
1 1 1
0 1 1
0 0 1
1 0 1
0 1 0
1 0 1
4

Input Image

0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

Stride : 1x1

Filter(Kernel)

1	0	1
0	1	0
1	0	1

Feature Map

2	2	3	1	1
1	4	3		

1	1	0
1	1	1
0	1	1

×

1	0	1
0	1	0
1	0	1

=

3

Input Image

0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

Stride : 1x1

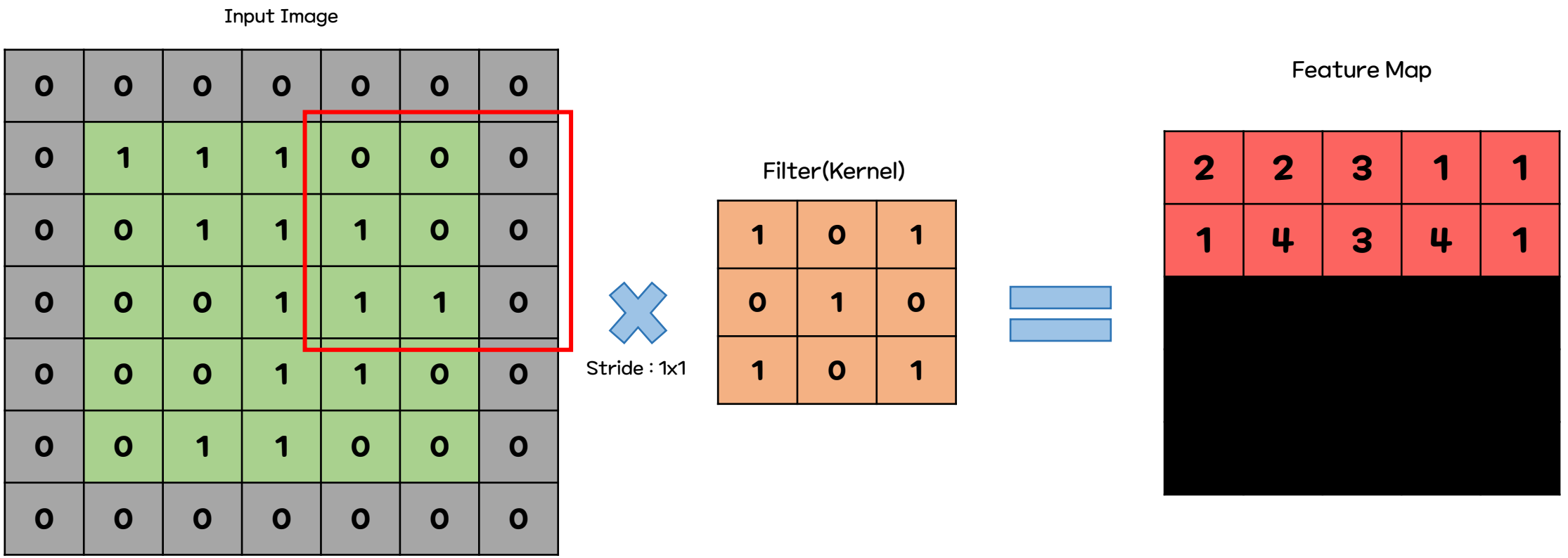
Filter(Kernel)

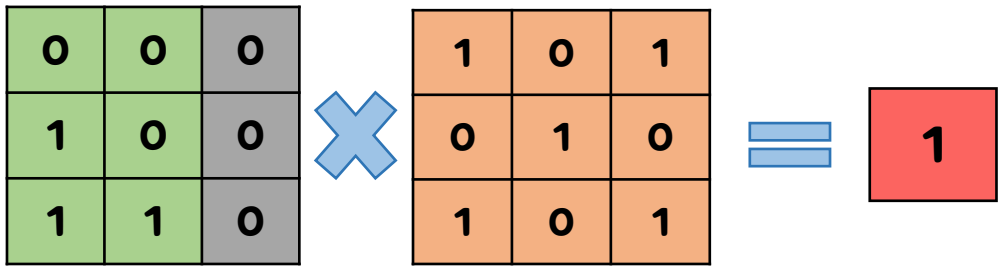
1	0	1
0	1	0
1	0	1

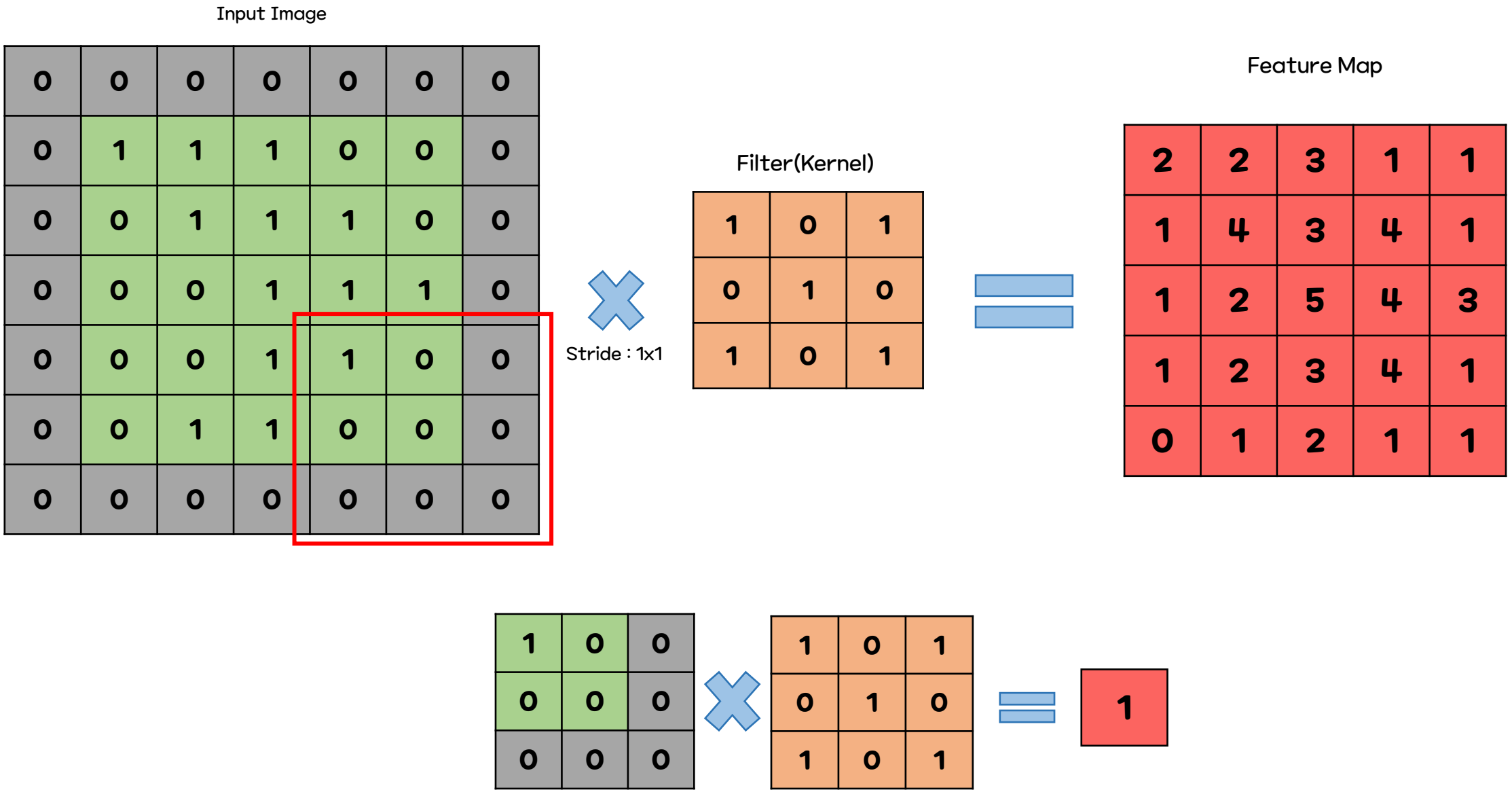
Feature Map

2	2	3	1	1
1	4	3	4	

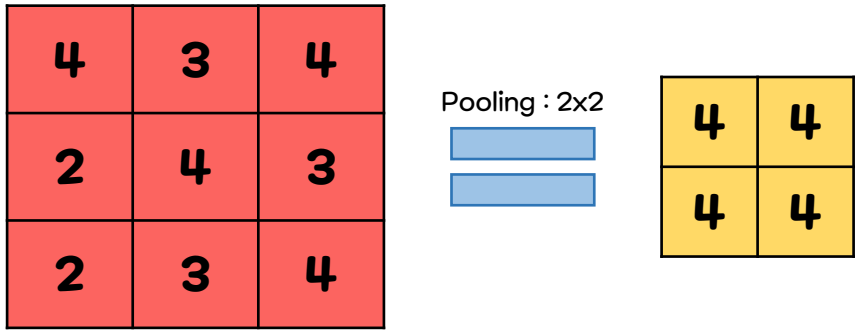
1	0	0		1	0	1					
1	1	0		0	1	0				4	
1	1	1		1	0	1					


Input Image
0 0 0 0 0 0 0
0 1 1 1 0 0 0
0 0 1 1 1 0 0
0 0 0 1 1 1 0
0 0 0 1 1 0 0
0 0 1 1 0 0 0
0 0 0 0 0 0 0
Stride : 1x1
Filter(Kernel)
1 0 1
0 1 0
1 0 1
Feature Map
2 2 3 1 1
1 4 3 4 1


0 0 0
1 0 0
1 1 0
1 0 1
0 1 0
1 0 1
1


Input Image
0 0 0 0 0 0 0
0 1 1 1 0 0 0
0 0 1 1 1 0 0
0 0 0 1 1 1 0
0 0 0 1 1 0 0
0 0 1 1 0 0 0
0 0 0 0 0 0 0
Stride : 1x1
Filter(Kernel)
1 0 1
0 1 0
1 0 1
Feature Map
2 2 3 1 1
1 4 3 4 1
1 2 5 4 3
1 2 3 4 1
0 1 2 1 1
1 0 0
0 0 0
0 0 0
1 0 1
0 1 0
1 0 1
1

Pooling



Pooling은 각 결과값(Feature map)의 차원을 축소해 주는 것을 목적으로 둔다.

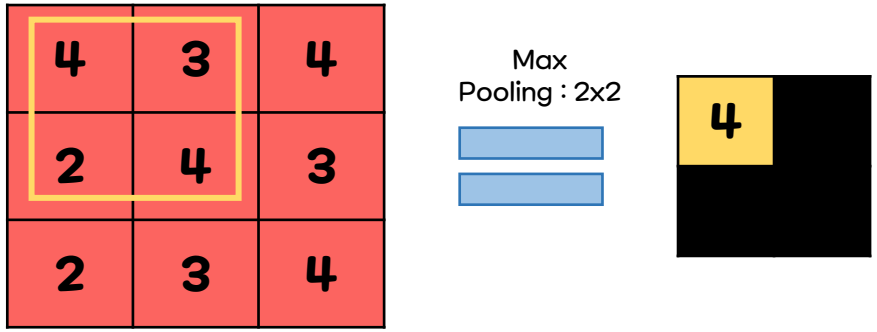
특징이 낮은 부분을 과감히 삭제하고 특징이 도드라진 부분을 취하여 결과값의 크기(차원)을 줄이는 과정이다.

Pooling의 대표적인 방법으로, 2가지가 있다.

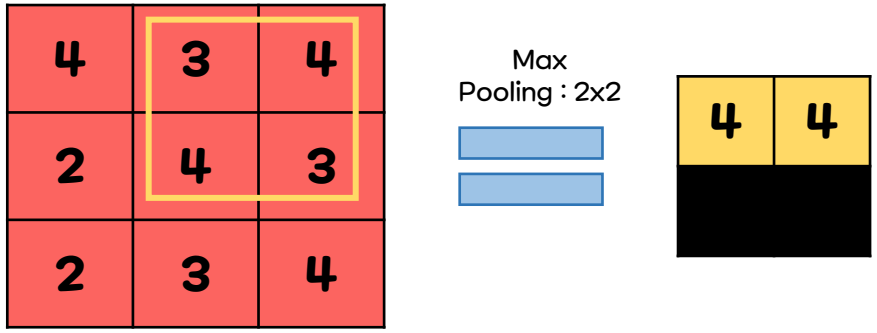
1. Max pooling
2. Average pooling

Pooling의 크기가 2x2일 경우, Feature map의 2x2부분마다 가장 큰(Max)값이나 평균값(average)를 가져와 결과값의 크기를 줄인다.

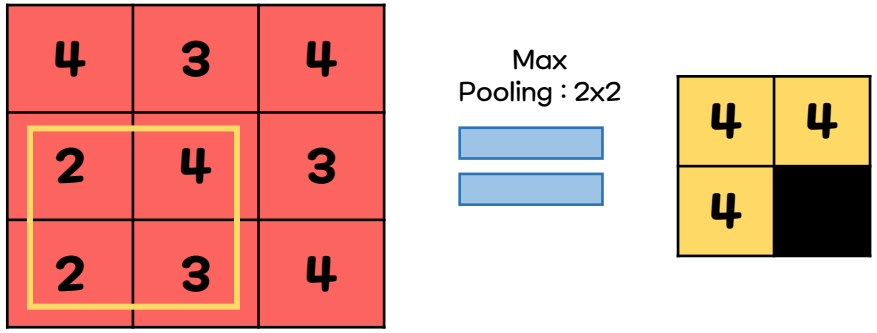
Pooling



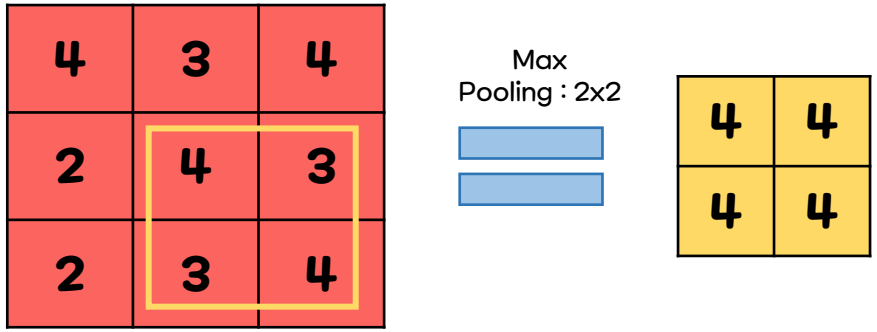
Pooling



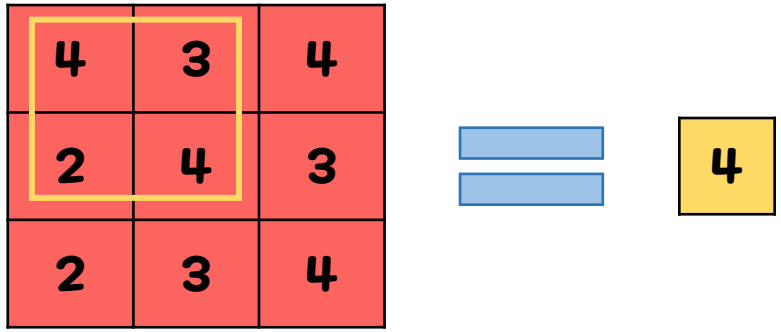
Pooling



Pooling



Pooling

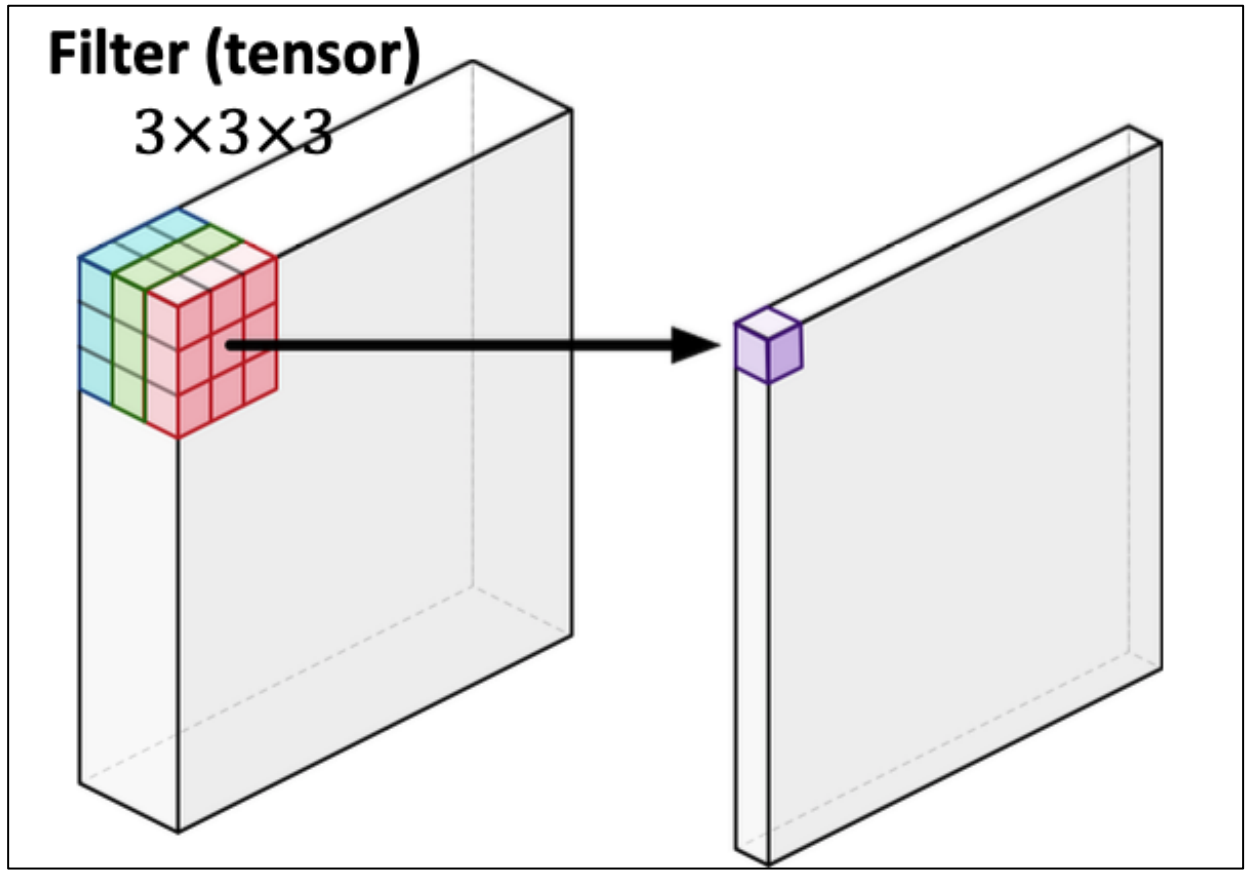


Pooling도 따지고 보면 Filter의 개념이 있지만, 학습을 하지않는다.

따라서 Pooling에는 학습가능한 가중치는 없다.

WHY : max pooling은 필터 내에서 가장 큰 수를 찾는 작업이고 average pooling또한 필터 내의 수의 평균을 구하는 단순 작업이기 때문
즉, 가중치가 사용되지 않는다.

CNN in Color Image



2차원 Matrix뿐만 아니라, 3차원 이상의 데이터가 가능하다.

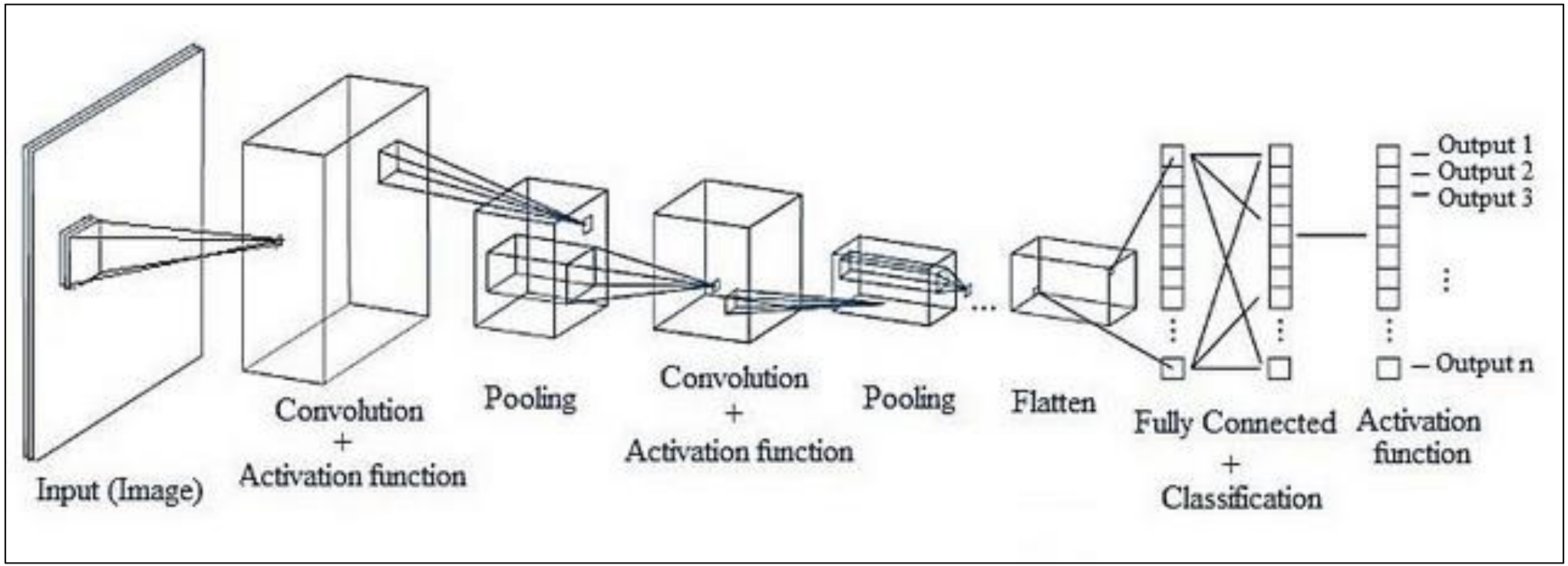
예를 들어, 컬러이미지의 경우는 RGB 세가지 채널이 추가 된다.

이때 필터 또한 이미지의 채널의 맞게 차원이 구성되어야 한다.

그리고 연산은 각각의 채널의 내적연산을 하고 그 값을 더해서,
1개의 숫자로 표현한다.

https://halfundecided.medium.com/%EB%94%A5%EB%9F%AC%EB%8B%9D-%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-cnn-convolutional-neural-networks-%EC%89%BD%EA%B2%8C-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0-836869f88375

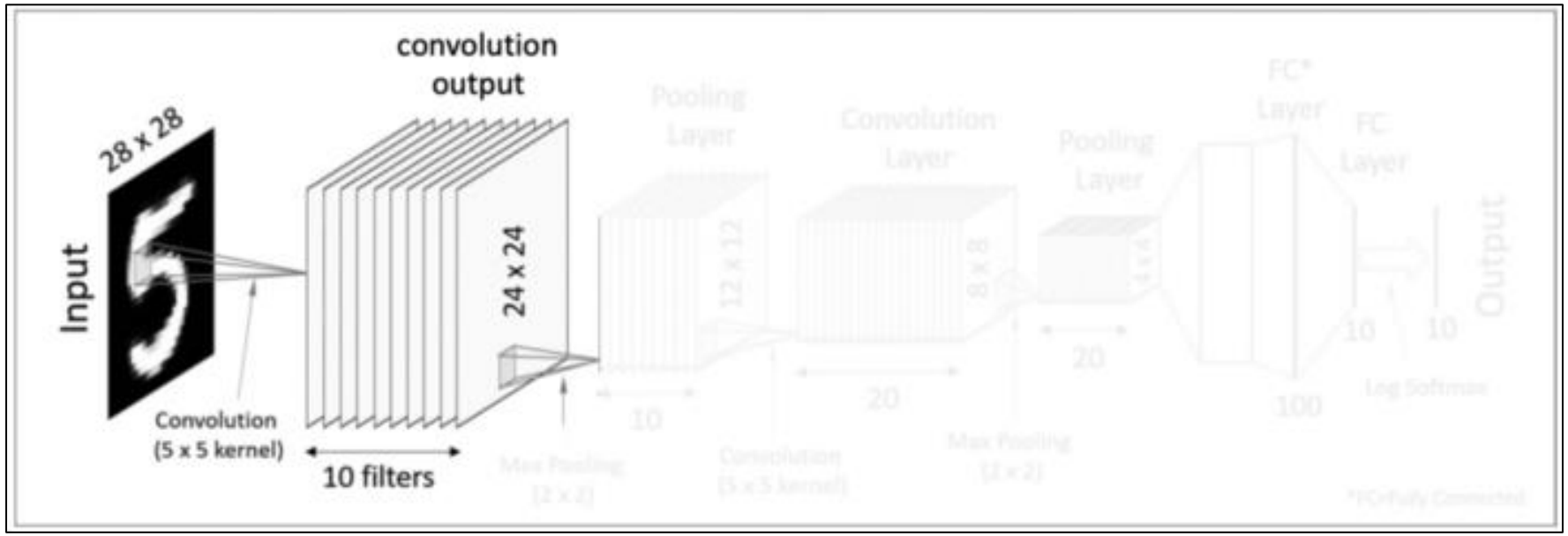
CNN 작동



CNN의 구조는 Dense Layer와 다르게, Convolution layer와
Pooling layer사이에 활성화함수를 배치하며 구성된다.

https://halfundecided.medium.com/%EB%94%A5%EB%9F%AC%EB%8B%9D-%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-cnn-convolutional-neural-networks-%EC%89%BD%EA%B2%8C-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0-836869f88375

CNN 작동



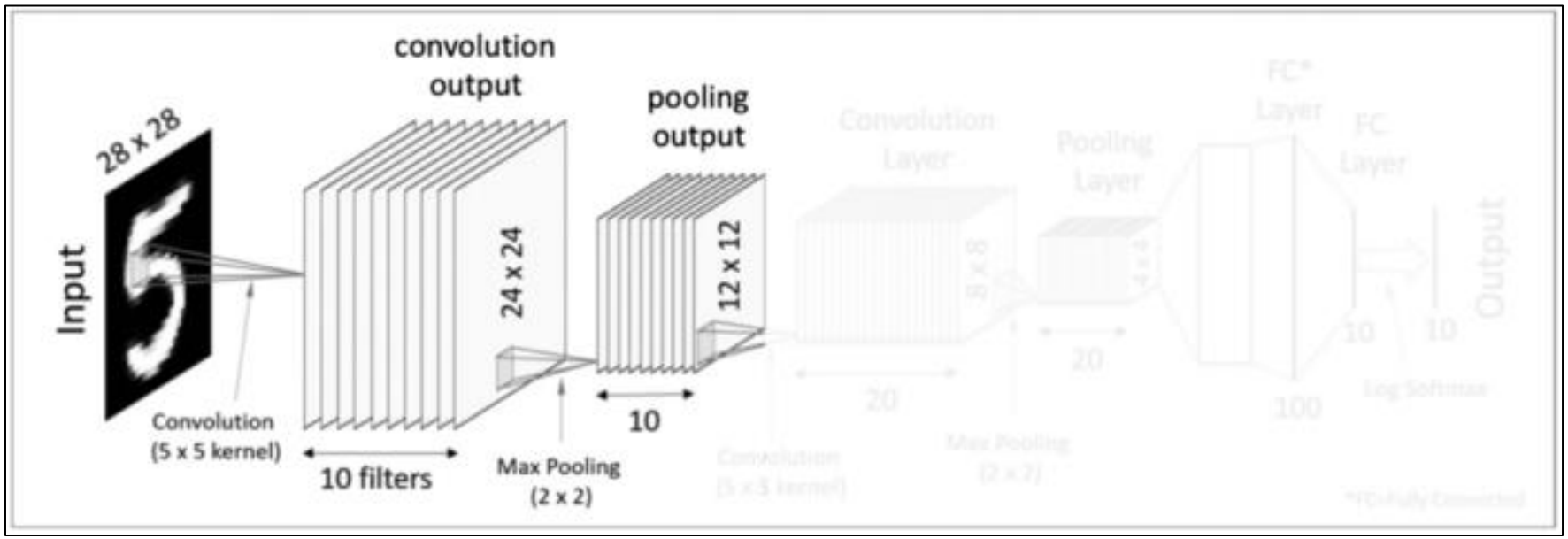
위 그림에서 28x28크기의 그림을 5x5x10의 필터로 Convolution하면
24x24x10의 Feature map이 구해질 것이다.

이 Featrue map에 활성화함수(Relu같은...)를 적용해 선형에서 비선형성을 추가한다.

즉, Convolutional Layer = convolution + activation
으로 표현할 수 있겠다.

https://halfundecided.medium.com/%EB%94%A5%EB%9F%AC%EB%8B%9D-%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-cnn-convolutional-neural-networks-%EC%89%BD%EA%B2%8C-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0-836869f88375

CNN 작동

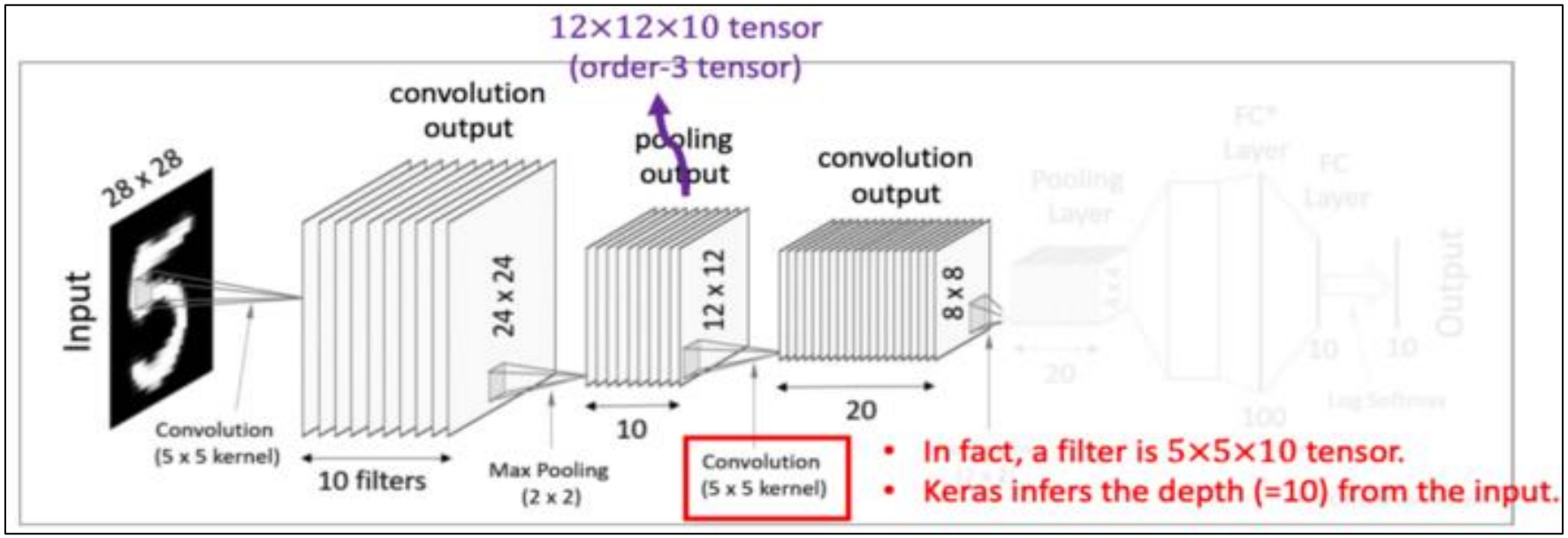


Convolution layer에서 구한 Feature map을 Pooling Layer를 통해
차원을 축소시킨다.

2x2 Max Pooling을 통해 24x24x10 feature map을
12x12x10 Feature map으로 축소시킨다.

https://halfundecided.medium.com/%EB%94%A5%EB%9F%AC%EB%8B%9D-%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-cnn-convolutional-neural-networks-%EC%89%BD%EA%B2%8C-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0-836869f88375

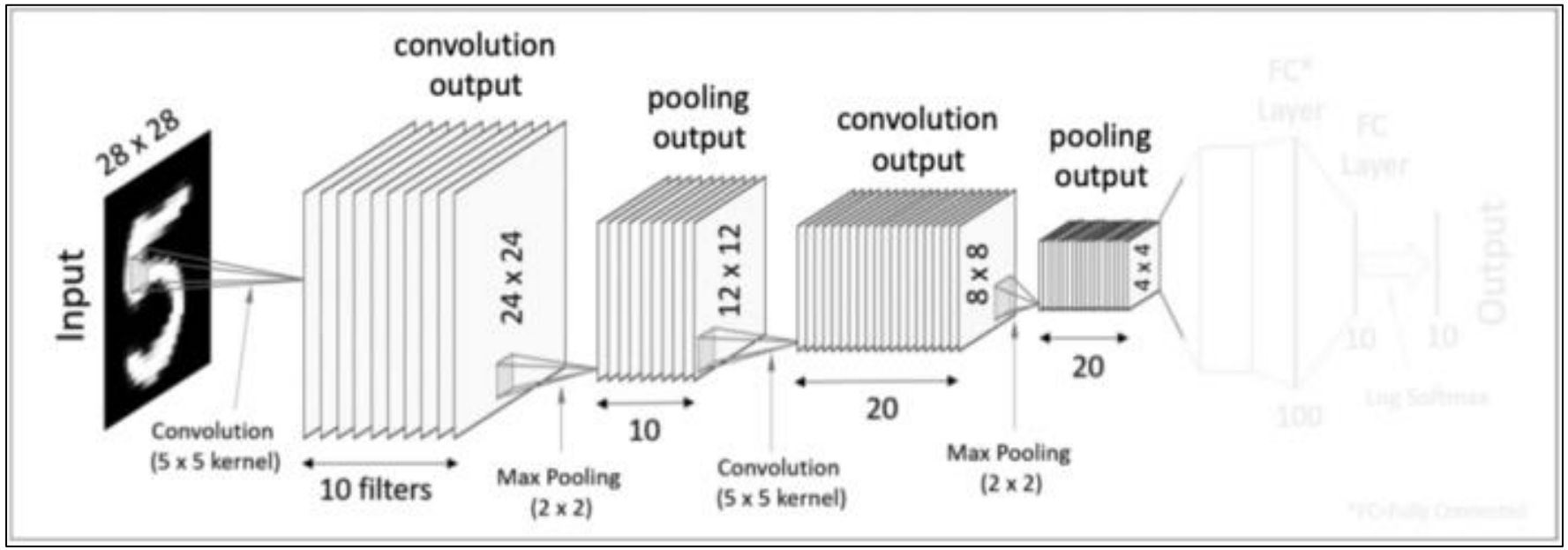
CNN 작동



그 다음의 Convolution layer에서
5x5x10필터를 20개를 사용하여
8x8x20의 Feature map을 얻어낸다.

https://halfundecided.medium.com/%EB%94%A5%EB%9F%AC%EB%8B%9D-%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-cnn-convolutional-neural-networks-%EC%89%BD%EA%B2%8C-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0-836869f88375

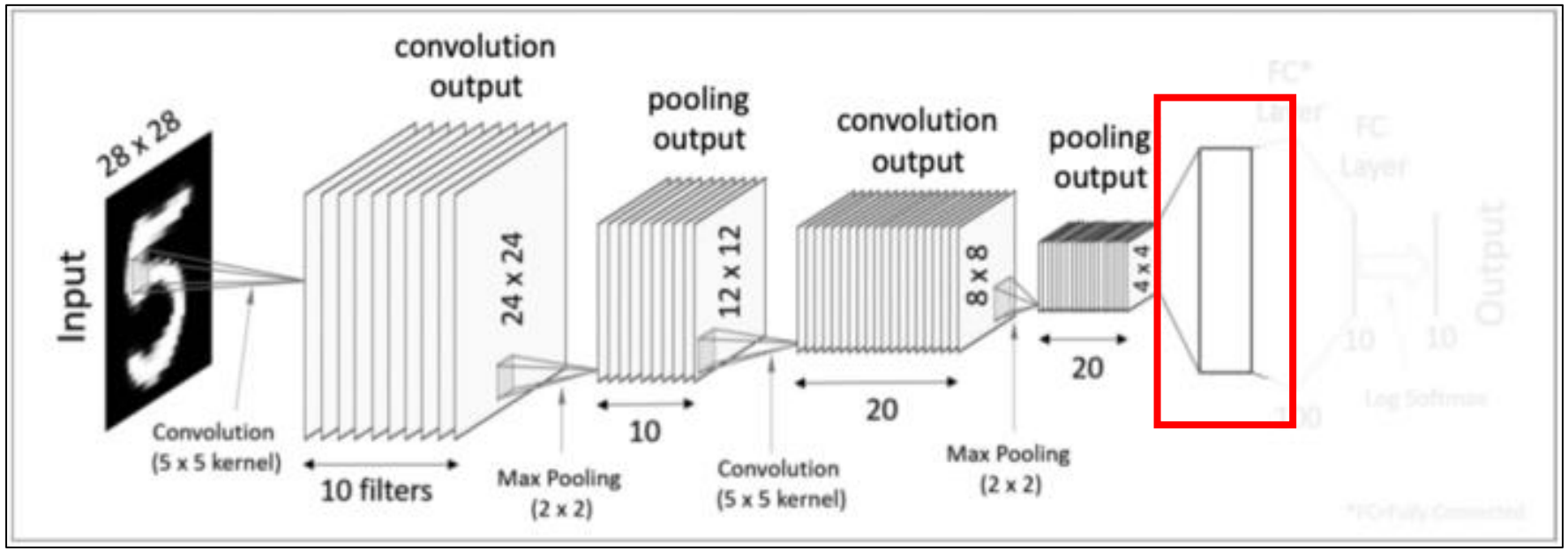
CNN 작동



2번째 Pooling layer에서
2x2 Max Pooling으로
4x4x20 Feature Map으로 축소시킨다.

https://halfundecided.medium.com/%EB%94%A5%EB%9F%AC%EB%8B%9D-%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-cnn-convolutional-neural-networks-%EC%89%BD%EA%B2%8C-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0-836869f88375

CNN 작동

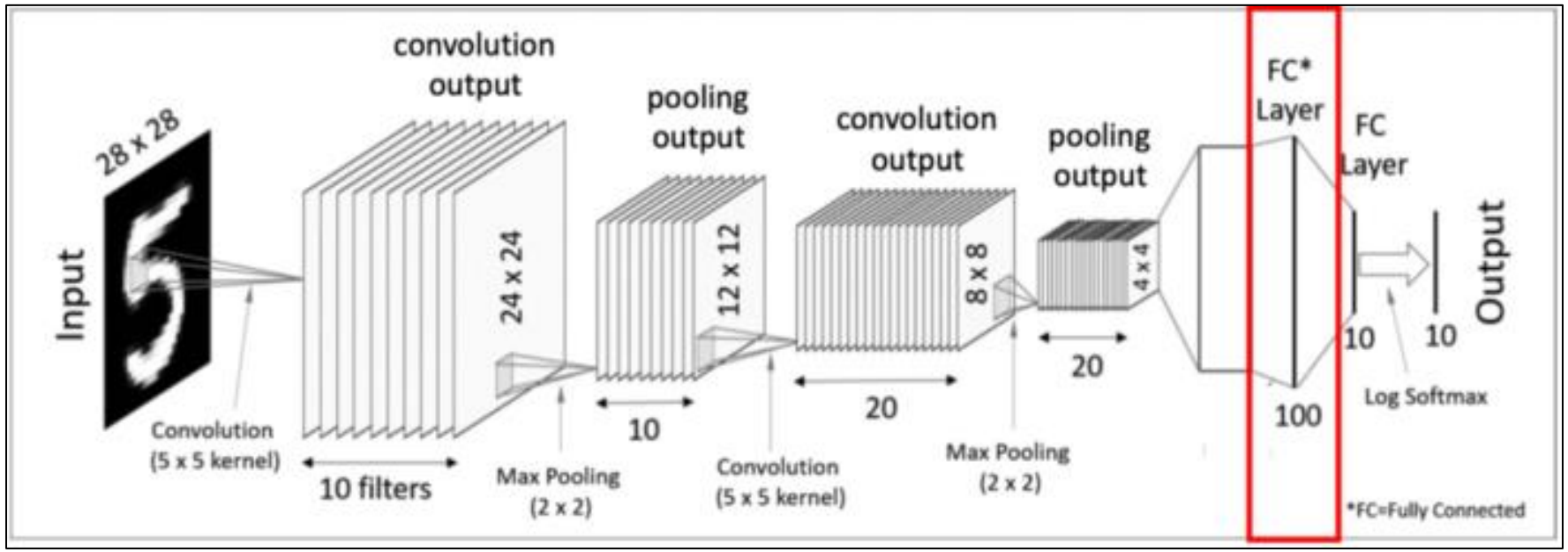


4x4x20크기의 텐서를 한줄로 Flatten 한다.
총 320의 차원을 가진 벡터로 표현된다.

Feature map은 이미지 자체라기 보다는 입력된 이미지에서 얻어온 특징 데이터가 된다.
이 데이터는 1차원의 벡터로 변형시켜도 무관하다.

https://halfundecided.medium.com/%EB%94%A5%EB%9F%AC%EB%8B%9D-%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-cnn-convolutional-neural-networks-%EC%89%BD%EA%B2%8C-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0-836869f88375

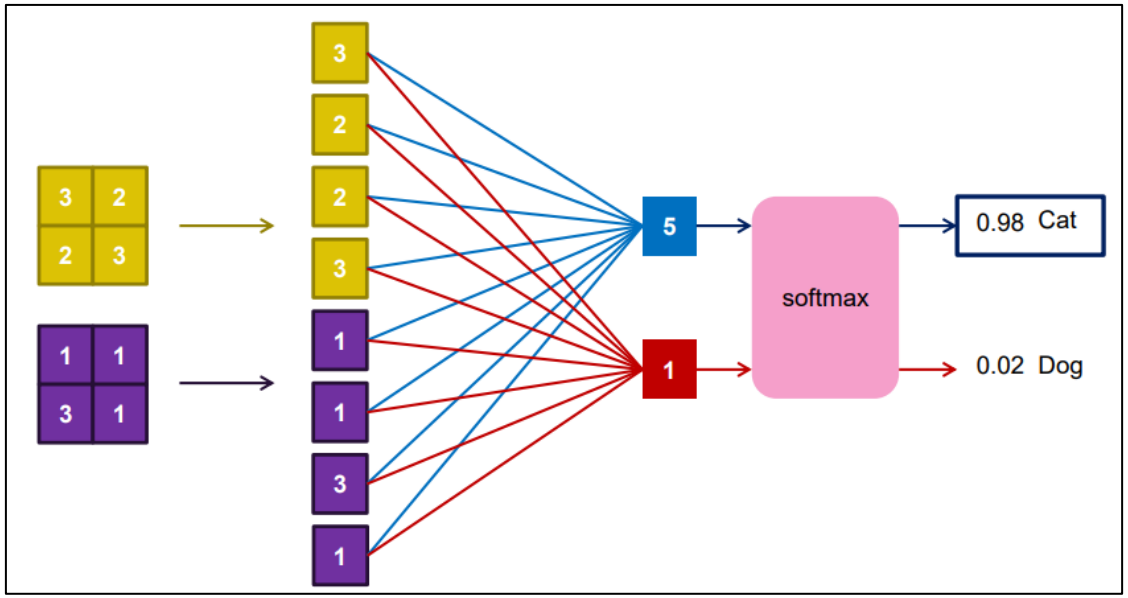
CNN 작동



마지막으로 하나(혹은 이상의) FC Layer에서 Softmax나 Sigmod등 활성환 함수를 적용하여 최종결과물을 출력하게 된다.

https://halfundecided.medium.com/%EB%94%A5%EB%9F%AC%EB%8B%9D-%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-cnn-convolutional-neural-networks-%EC%89%BD%EA%B2%8C-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0-836869f88375

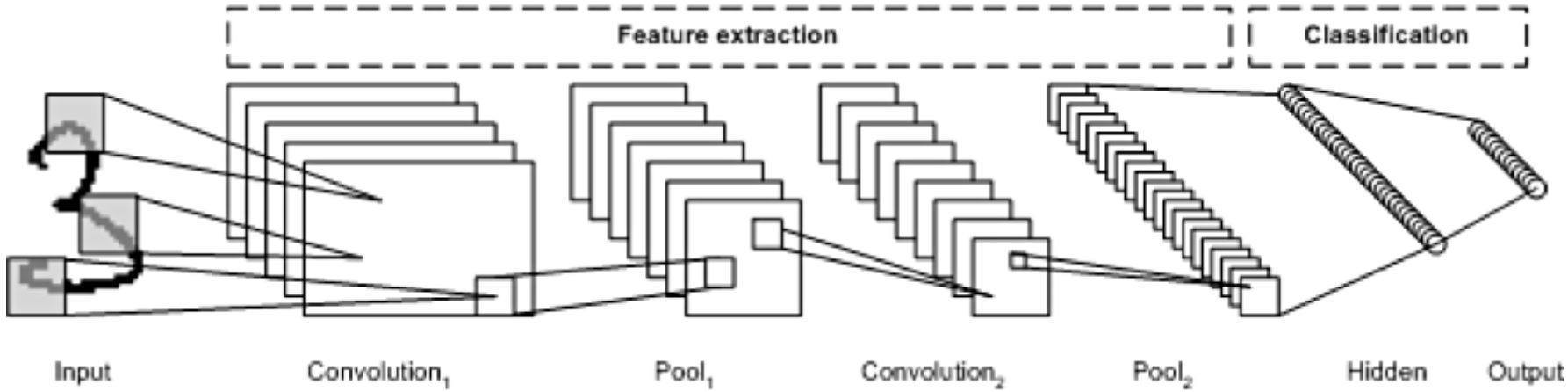
CNN 작동



마지막으로 하나(혹은 이상의) FC Layer에서 Softmax나 Sigmod등 활성환 함수를 적용하여
최종결과물을 출력하게 된다.

https://halfundecided.medium.com/%EB%94%A5%EB%9F%AC%EB%8B%9D-%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-cnn-convolutional-neural-networks-%EC%89%BD%EA%B2%8C-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0-836869f88375

CNN 작동



CNN을 통해서 입력 이미지에 대한 Feature를 잘 추출하여 수로 표현한다.
그 수를 분류기(선형 변환)하여 Label에 맞게 분류한다.

위 그림을 통해서 이미지를 분류할때는 총 2가지의 작업이 일어난다.
CNN를 통해서 이미지에대한 특징을 추출하고
FC를 통해 추출된 특징(벡터)를 선형변환하여 분류기로 사용한다.

https://halfundecided.medium.com/%EB%94%A5%EB%9F%AC%EB%8B%9D-%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-cnn-convolutional-neural-networks-%EC%89%BD%EA%B2%8C-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0-836869f88375