강원지역혁신플랫폼

기계학습

Machine Learning

합성곱 신경망의 전체구조와 합성곱 연산

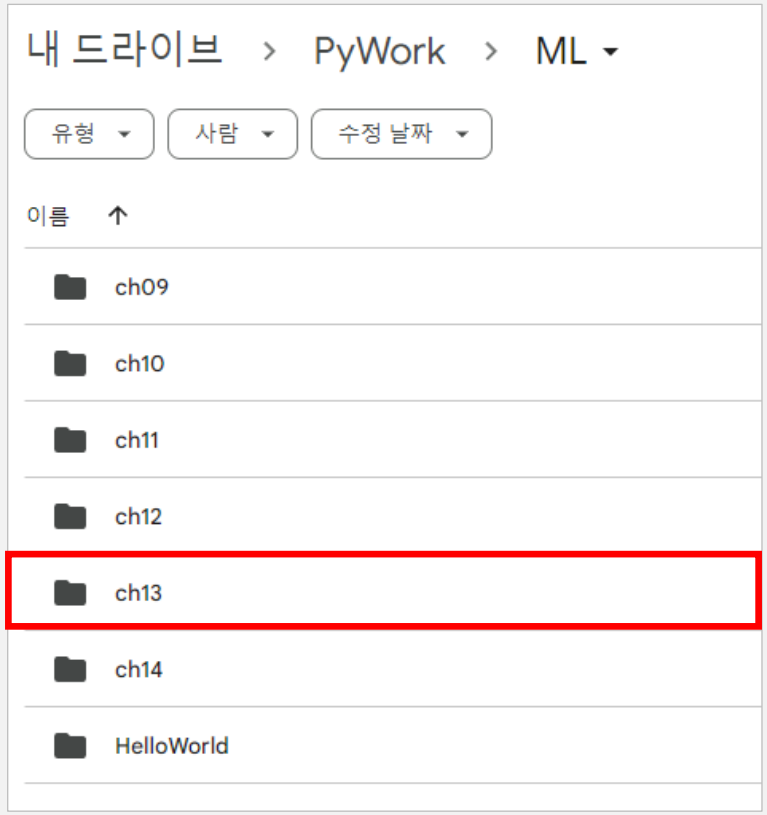
학습목표

- 합성곱 신경망의 전체구조와 합성곱 연산을 이해하고 구현할 수 있습니다.

01 | 실습

(권장) 아래와 같은 경로에 실행 소스가 존재하면 환경 구축 완료

- 구글 드라이브 "PyWork > ML" 폴더로 이동함
  - 아래의 [ch13] 폴더를 클릭하면 됨

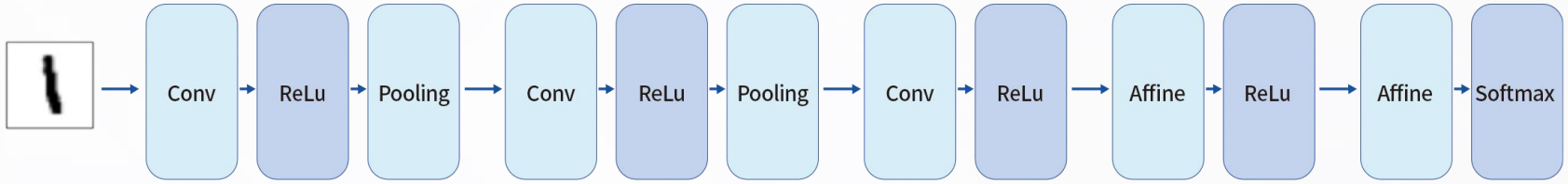


01 | 실습

- “ML > ch13 >” 폴더를 클릭함
  - 아래의 [ch13_01_합성곱 연산.ipynb] 스크립트를 클릭함

합성곱 신경망(Convolutional Neural Network, CNN)

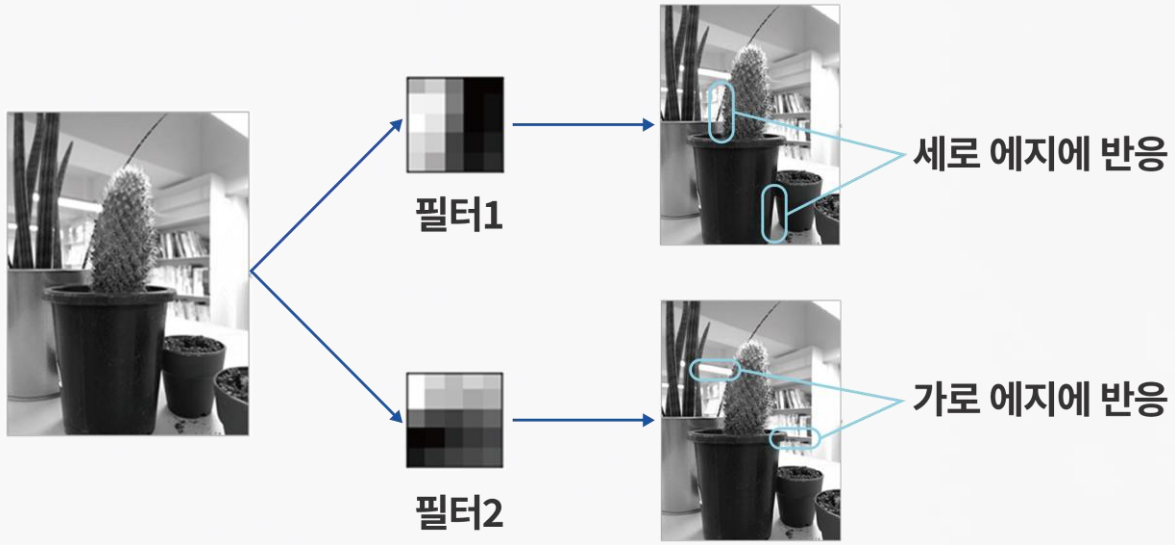
- 합성곱 신경망(CNN)은 이미지 인식과 음성 인식 등 다양한 곳에서 사용됨
  - 특히, 이미지 인식 분야에서 딥러닝을 활용한 기법은 거의 다 CNN을 기초로 하고 있음
    - CNN은 기본적으로 합성곱 계층(Convolution layer) – 풀링 계층(Pooling layer) – 완전연결계층(Fully Connected layer) 순서로 진행됨



CNN으로 이루어진 네트워크 예

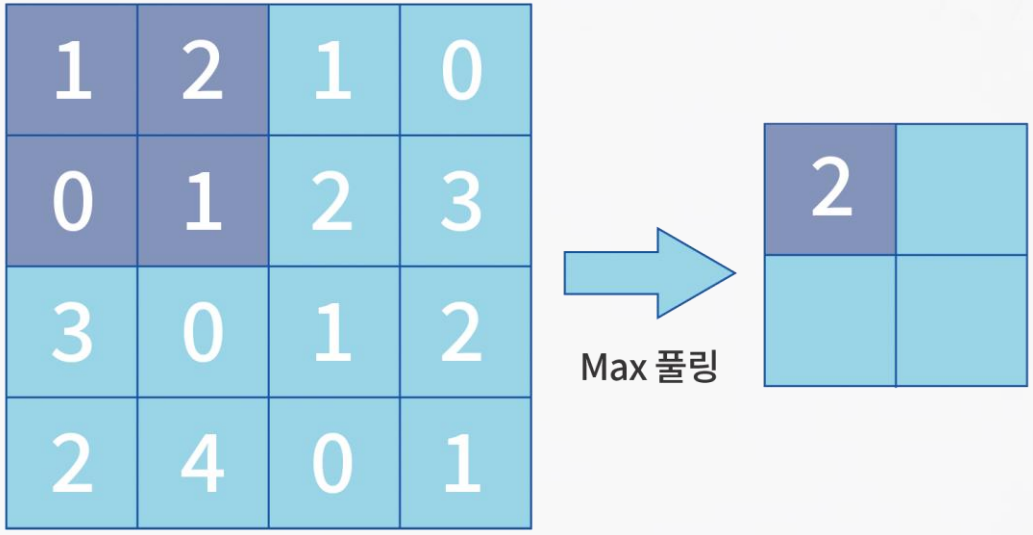
02 | 합성곱 신경망

- CNN에서 핵심이 되는 부분은 합성곱 계층(Convolution layer)임
  - 합성곱 계층에서 이미지를 분류(classification)하는데 필요한 특징(feature)정보들을 뽑아냄
  - 합성층 계층에는 수 많은 필터(filter)들이 있고, 이러한 필터들을 통해 특징들을 뽑아냄
    - 여기서 합성곱 필터(convolution filter)의 크기(size)는 변함이 없음



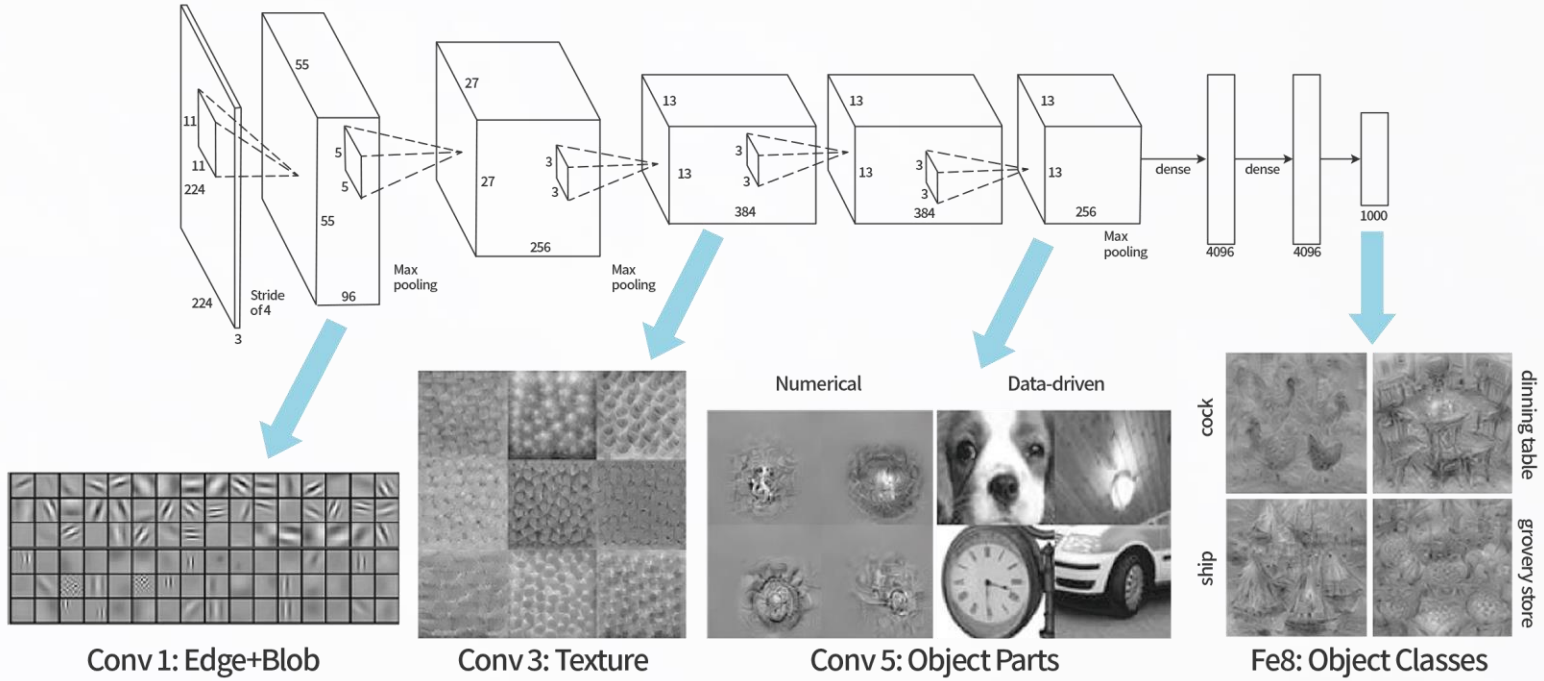
02 | 합성곱 신경망

- 풀링 계층(Pooling layer)은 최대 풀링(Max pooling)연산을 통해 특정영역에서 가장 큰 값을 꺼냄
  - 이러한 풀링(Pooling)연산을 통해서 형태는 유지하면서 기존의 이미지 크기를 작게 만들어 줄 수 있음
    - 이러한 작업을 다운 샘플링(Down sampling)이라고 함



02 | 합성곱 신경망

- 아래 그림과 같이 **첫 번째 합성곱 계층**(First Convolution layer)에서의 **합성곱 필터**(Convolution filter)는 **엣지**(Edge) **정보만**(선과 같은 단순한 모양)**추출**할 수 있음
  - 하지만, **풀링 계층**(Pooling layer)을 통해 **전체 이미지 사이즈**가 **작아**지니 **합성곱 필터**(Conv filter)의 **크기**가 **상대적**으로 **조금 더 커지게 됨**
    - 그래서, **이미지**에서 **조금 더 추상적**(Abstract)인 **정보**를 **볼 수** 있는 **합성곱 필터**(Conv filter)를 **갖게 됨**



03 | CNN의 전체구조

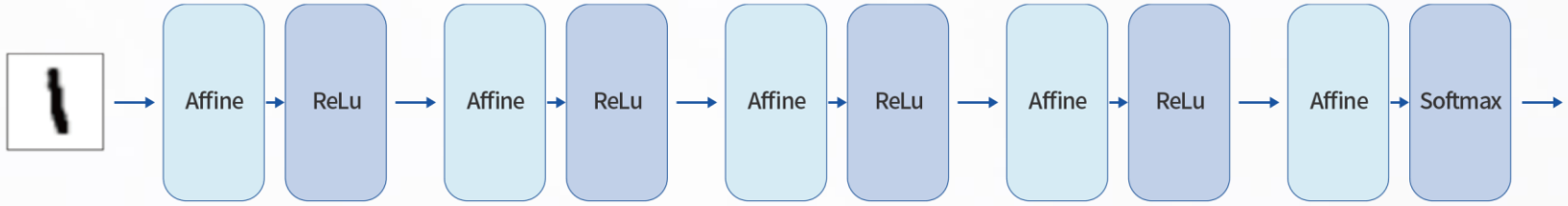
CNN의 전체 구조

CNN의 네트워크 구조를 살펴보며 전체 틀을 이해해보자.

- CNN도 신경망과 같이 레고 블록처럼 계층을 조합하여 만들 수 있음
  - CNN은 기본적으로 합성곱 계층(Convolution layer) – 풀링 계층(Pooling layer) – 완전연결계층(Fully Connected layer)순서로 진행됨
    - 완전연결계층은 인접하는 계층이 모든 뉴런과 결합되어 있음
      - 여기서는 완전히 연결된 계층을 Affine 계층 클래스로 구현함

03 | CNN의 전체구조

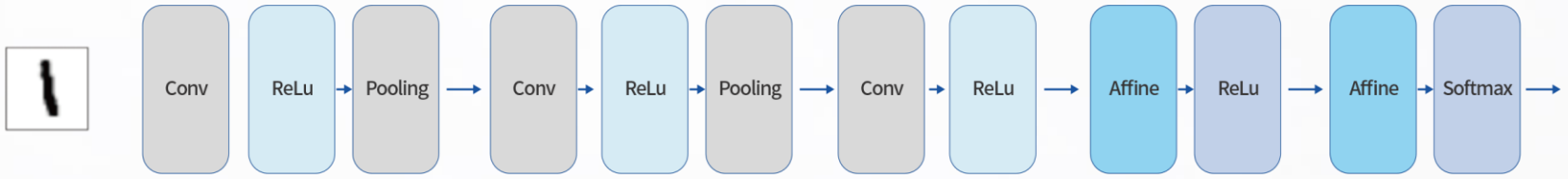
- 예를 들어 층이 5개인 완전연결 신경망(fully-connected Neural Network)은 아래 그림과 같이 구현할 수 있음
  - 아래 그림과 같이 완전연결 신경망은 Affine 계층 뒤에 활성화 함수를 갖는 ReLU 계층(혹은 Sigmoid 계층)이 이어짐
    - Affine-ReLU 조합이 4개가 쌓였고, 마지막 5번째 층은 Affine 계층에 이어 소프트맥스 계층에서 최종 결과(확률)를 출력함



완전연결 계층 (Affine 계층)으로 이루어진 네트워크 예

03 | CNN의 전체구조

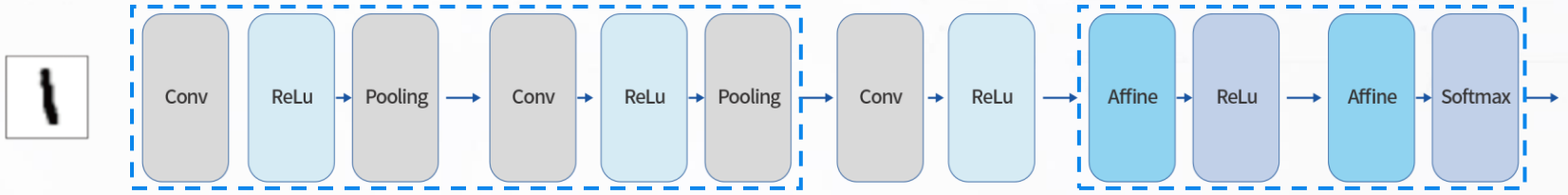
- 합성곱(Convolution)계층과 풀링(Pooling)계층을 추가하면,
  예를 들어 층이 5개인 CNN은 아래 그림과 같이 구현할 수 있음
  - 아래 그림의 CNN에서는 합성곱 계층과 풀링 계층이 추가된 것을 볼 수 있음
    - CNN 계층은 'Conv-ReLU-(Pooling)' 흐름으로 연결됨
      - 여기서 풀링 계층은 생략하기도 함
    - 완전연결 계층과 CNN 구조의 네트워크에서 'Affine-ReLU' 연결이 'Conv-ReLU-(Pooling)'으로 바뀌었다고 생각할 수 있음



CNN으로 이루어진 네트워크 예: 합성곱 계층과 풀링 계층이 새로 추가(회색)

03 | CNN의 전체구조

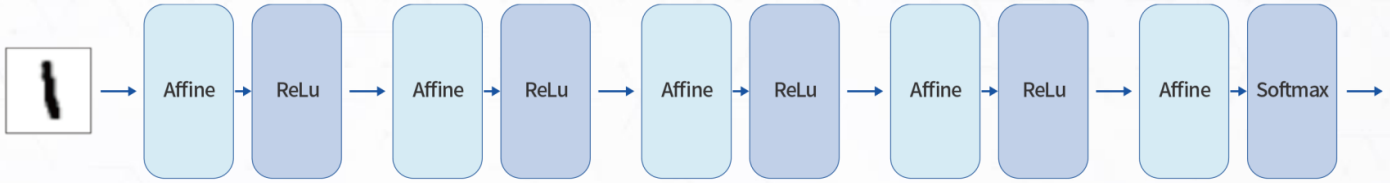
- 아래 그림의 CNN에서 주목할 다른 점은 출력에 가까운 층에서는 완전연결 계층과 같이 'Affine-ReLU'구성을 사용할 수 있다는 것임
  - 또, 마지막 출력 계층에서 'Affine-Softmax'조합을 그대로 사용함



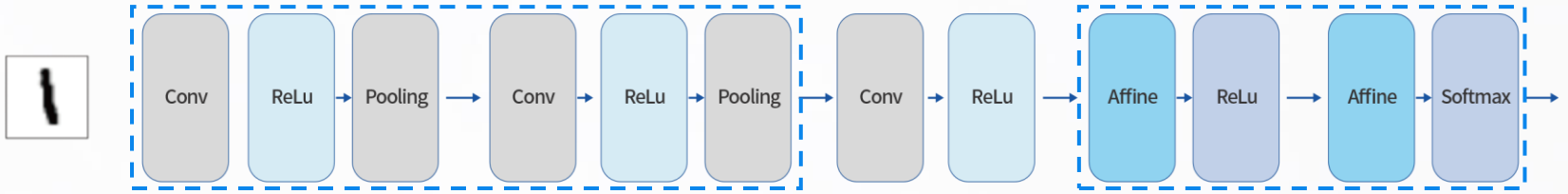
CNN으로 이루어진 네트워크 예: 합성곱 계층과 풀링 계층이 새로 추가(회색)

03 | CNN의 전체구조

아래 그림은 5층 신경망의 완전연결 신경망와 CNN으로 이뤄진 네트워크임



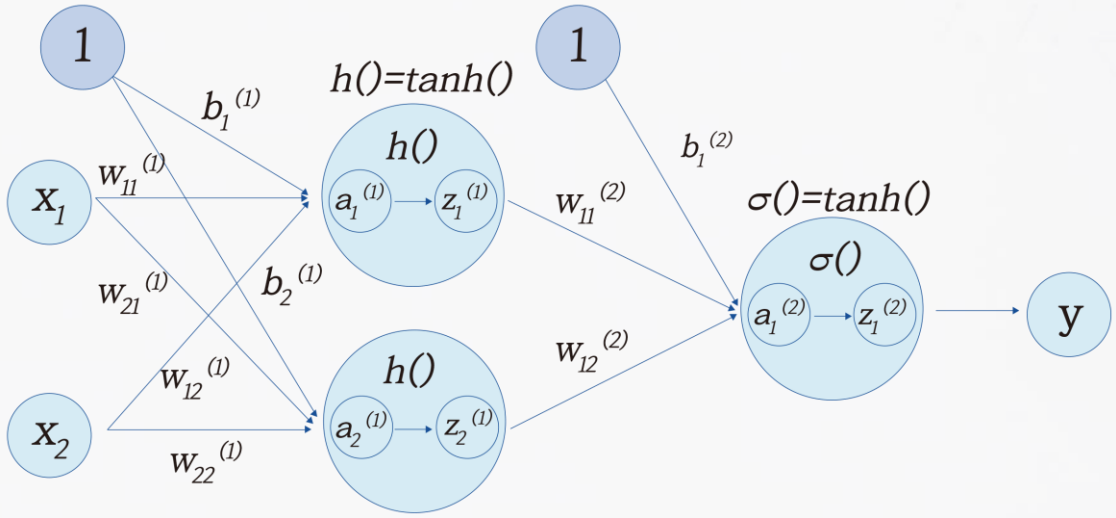
완전연결 계층 (Affine 계층)으로 이루어진 네트워크 예



CNN으로 이루어진 네트워크 예: 합성곱 계층과 풀링 계층이 새로 추가(회색)

완전연결 계층의 문제점

- 완전연결 신경망에서는 완전연결 계층(Affine 계층)을 사용함
  - 완전연결 계층에서는 인접하는 계층의 뉴런이 모두 연결되고 출력의 수는 임의로 정할 수 있음



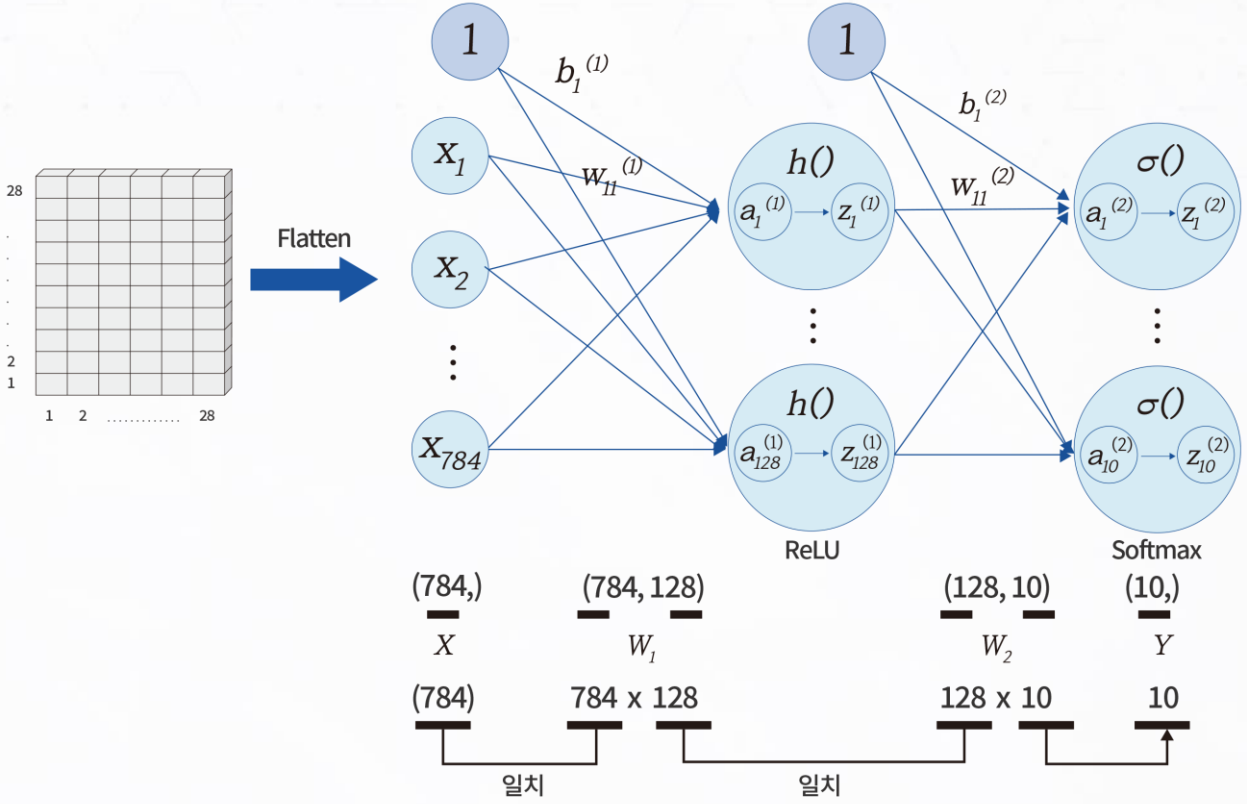
XOR 신경망 네트워크 구조

04 | 완전연결 계층의 문제점

- 완전연결 계층의 문제점은 무엇일까요?
  - 바로 '데이터의 형상이 무시'된다는 사실임
    - 입력 데이터가 이미지인 경우를 예로 들어 생각해 보자.
      - 이미지는 통상 세로(H)·가로(W)·채널(C, 색상)로 구성된 3차원 데이터임
      - 완전연결 계층에 입력할 때는 3차원 데이터를 평평한 1차원 데이터로 평탄화해야 함

04 | 완전연결 계층의 문제점

아래 그림에서 손글씨 숫자 MNIST 데이터셋을 사용한 형상이 (28, 28, 1)인 이미지를 한 줄로 세운 784개의 데이터를 첫 Affine 계층에 입력함



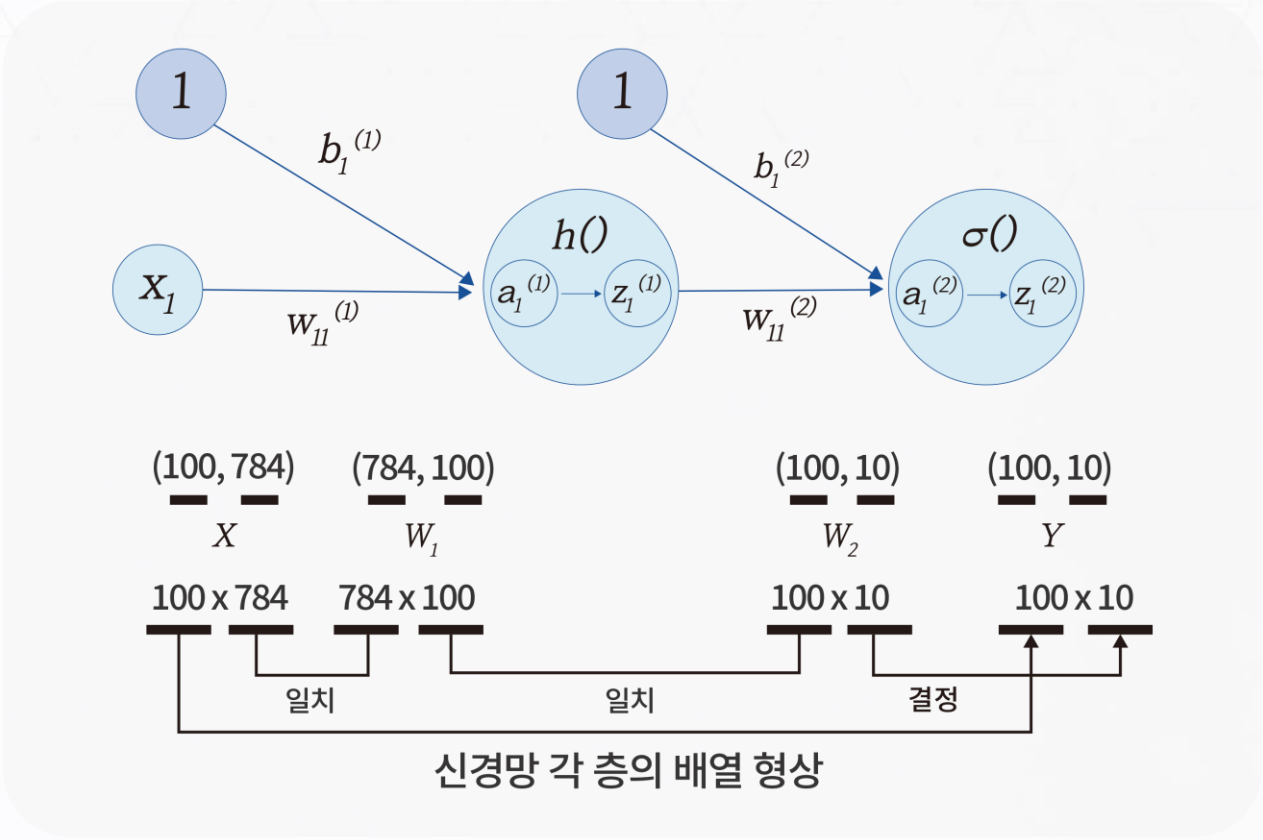
인공신경망 각 층의 배열 형상

04 | 완전연결 계층의 문제점

- 이미지는 3차원 형상이며, 이 형상에는 소중한 공간적 정보가 담겨있음
  - 예를 들면 다음과 같음
    - 공간적으로 가까운 픽셀은 값이 비슷함
    - RGB의 각 채널은 서로 밀접하게 관련되어 있음
    - 거리가 먼 픽셀끼리는 별 연관이 없는 등 3차원 속에서 의미를 갖는 본질적인 패턴이 숨어 있을 것임

04 | 완전연결 계층의 문제점

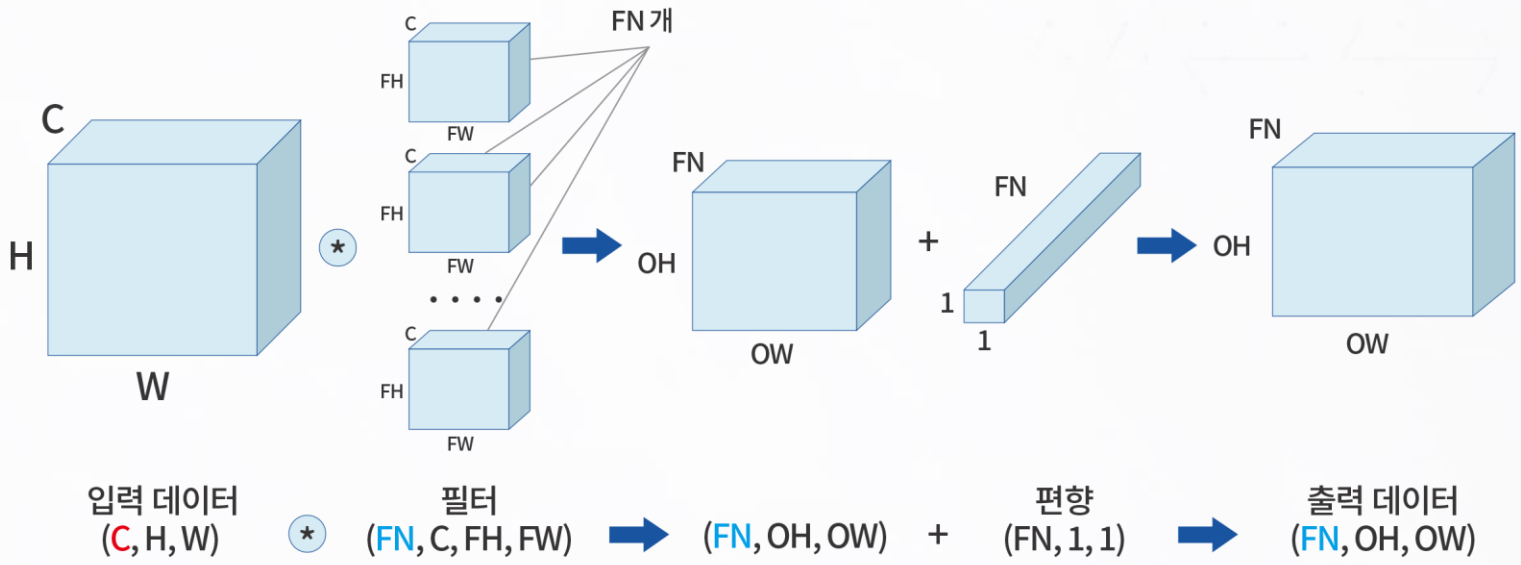
- 완전연결 계층은 형상을 무시하고 모든 입력 데이터를 동등한 뉴런 (같은 차원의 뉴런)으로 취급하여 형상에 담긴 정보를 살릴 수 없음



2층 신경망의 구현 예

04 | 완전연결 계층의 문제점

- 합성곱 계층은 형상을 유지함
  - ✦ 이미지도 3차원 데이터로 입력 받으며, 마찬가지로 다음 계층에도 3차원 데이터로 전달함
    - › 그래서 CNN에서는 이미지처럼 형상을 가진 데이터를 제대로 이해할(가능성이 있는)것임

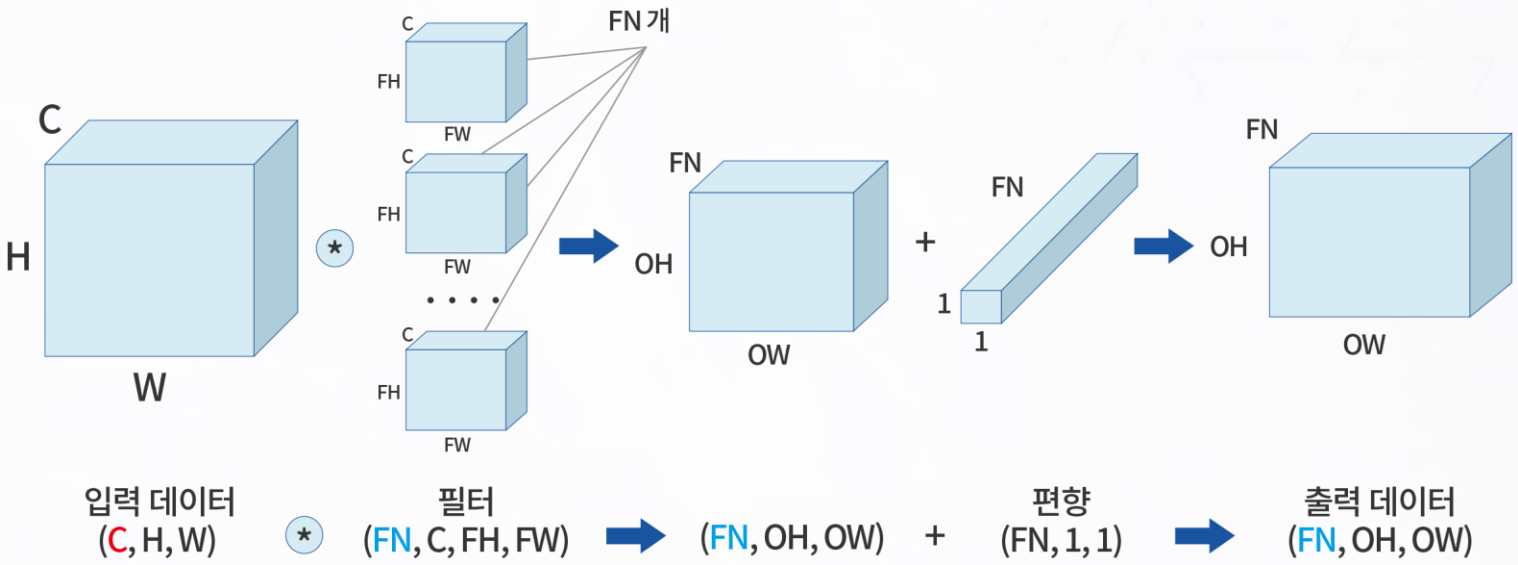


합성곱 연산의 처리 흐름 (편향 추가)

합성곱 계층

CNN에서는 패딩(Padding), 스트라이드(Stride) 등 CNN 고유의 개념들이 있음

✦각 계층 사이에는 3차원 데이터 같이 입체적인 데이터가 흐른다는 점에서 완전연결 신경망과 다른 점임



FN = filter의 개수, filter의 개수가 여러 개 될 수 있음

05 | 합성곱 계층: 합성곱 연산

- 합성곱 계층에서는 합성곱 연산을 처리함
  - 합성곱 연산은 이미지 처리에서 말하는 필터 연산에 해당함
    - 합성곱 연산을 ⊛ 기호로 표기함

Image == x

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

입력 데이터
(4, 4)

*

filter = weight

2	0	1
0	1	2
1	0	2

필터
(3, 3)

→

Convolution(합성곱)

$\sum$ (입력 * filter)

15	16
6	15

(2, 2)

합성곱 연산의 예: 합성곱 연산을 ⊛기호로 표기

$1*2+2*0+3*1+0*0+1*1+2*2+3*1+0*0+1*2=15$

Image의 3행 3열 (3*3)의 filter 맨버의 곱 후 합산 = 내적

05 | 합성곱 계층: 합성곱 연산

- 아래 그림과 같이 **합성곱 연산**은 **입력 데이터**에 **필터**를 **적용**함
  - 아래의 예에서 **입력 데이터**는 **세로**(H)·**가로**(W)**방향**의 **형상**을 가졌음
    - **필터** 역시 **세로**(H)·**가로**(W)**방향**의 **차원**을 가짐

(4, 4)

Image == x

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

입력 데이터

*

(3, 3)

filter = weight

2	0	1
0	1	2
1	0	2

필터

(2, 2)

Convolution(합성곱)

$\sum$ (입력 * filter)

15	16
6	15

합성곱 연산의 예: 합성곱 연산을 ⊛기호로 표기

05 | 합성곱 계층: 합성곱 연산

- 아래 그림과 같이 데이터와 필터의 형상을 (높이, 너비)로 표기함
  - 입력은 (4, 4), 필터는 (3, 3), 출력은 (2, 2)가 됨
    - 필터를 커널(Kernel)이라 하기도 함

합성곱 연산의 예: 합성곱 연산을 ⊛기호로 표기

05 | 합성곱 계층: 합성곱 연산

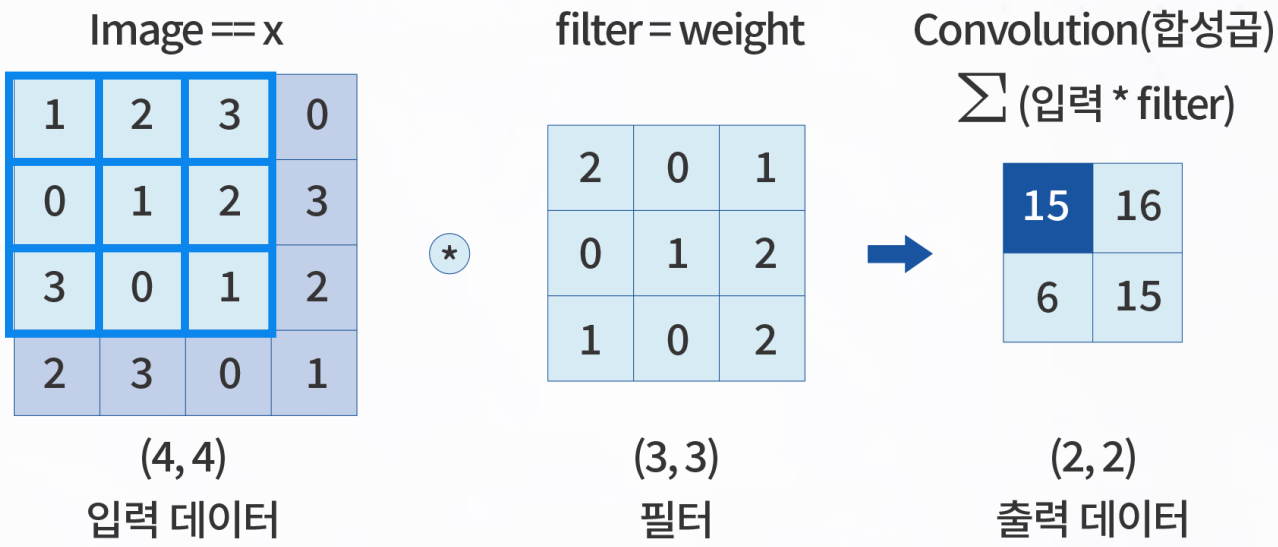
- 합성곱 연산은 필터의 윈도우(Window)를 일정 간격으로 이동해가며 입력 데이터에 적용함
  - 아래의 경우, 윈도우는 파란색 3 x 3 부분을 가리킴
    - 출력은 입력과 필터에서 대응하는 원소끼리 곱한 후 그 총합을 구함
      - 이 계산을 단일 곱셈-누산(Fused Multiply-Add, FMA)이라 함

$$1*2+2*0+3*1+0*0+1*1+2*2+3*1+0*0+1*2=15$$

합성곱 연산의 예: 합성곱 연산을 ⊛기호로 표기

05 | 합성곱 계층: 합성곱 연산

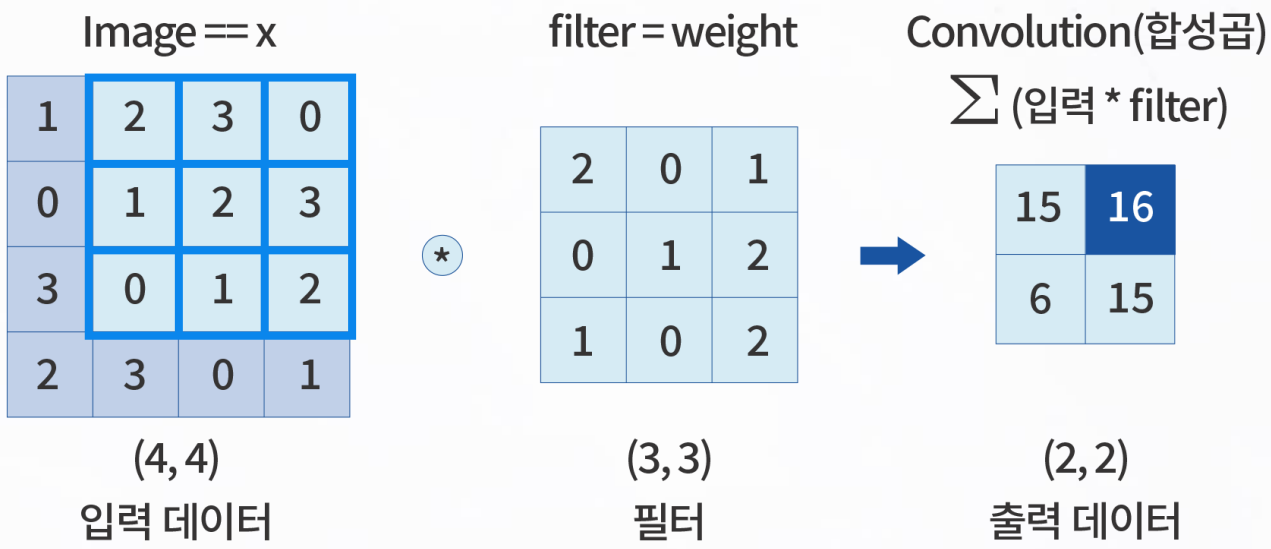
- 아래의 그림은 첫 번째 윈도우 부분으로 합성곱 연산의 계산을 그린 것임
  - 아래의 그림에서 윈도우는 파란색 3 x 3 부분임
    - 출력 데이터는 입력 데이터의 윈도우 부분과 필터에서 대응하는 원소끼리 곱한 후 그 총합임



$$1 * 2 + 2 * 0 + 3 * 1 + 0 * 0 + 1 * 1 + 2 * 2 + 3 * 1 + 0 * 0 + 1 * 2 = 15$$

05 | 합성곱 계층: 합성곱 연산

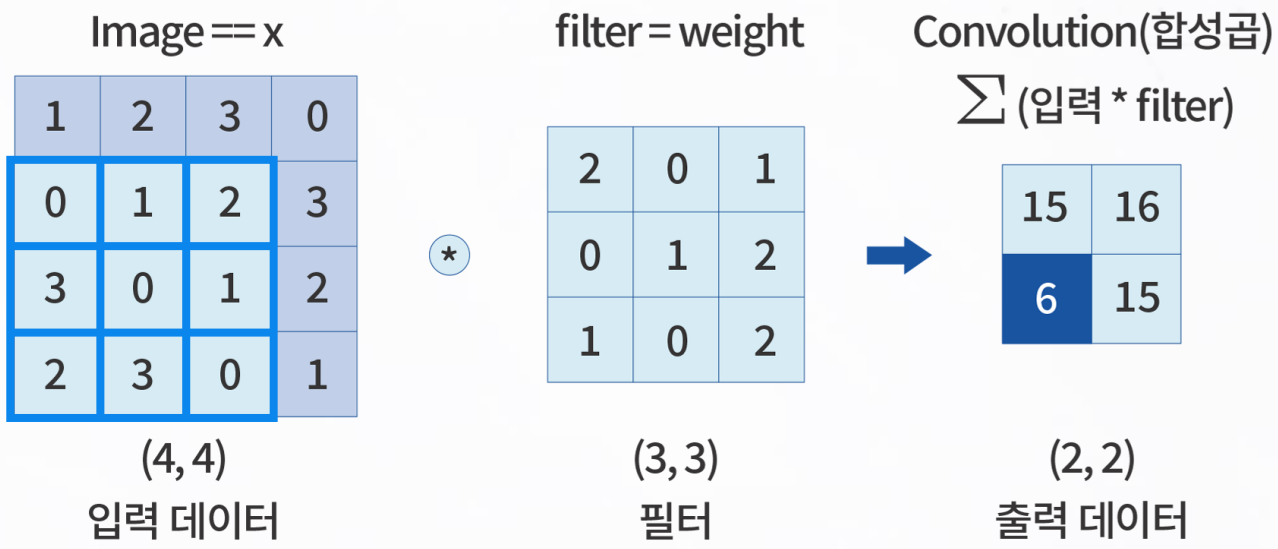
- 아래의 그림은 두 번째 윈도우 부분으로 합성곱 연산의 계산을 그린 것임
  - 아래의 그림에서 윈도우는 파란색 3 x 3 부분임
    - 출력 데이터는 입력 데이터의 윈도우 부분과 필터에서 대응하는 원소끼리 곱한 후 그 총합임



$$2 * 2 + 3 * 0 + 0 * 1 + 1 * 0 + 2 * 1 + 3 * 2 + 0 * 1 + 1 * 0 + 2 * 2 = 16$$

05 | 합성곱 계층: 합성곱 연산

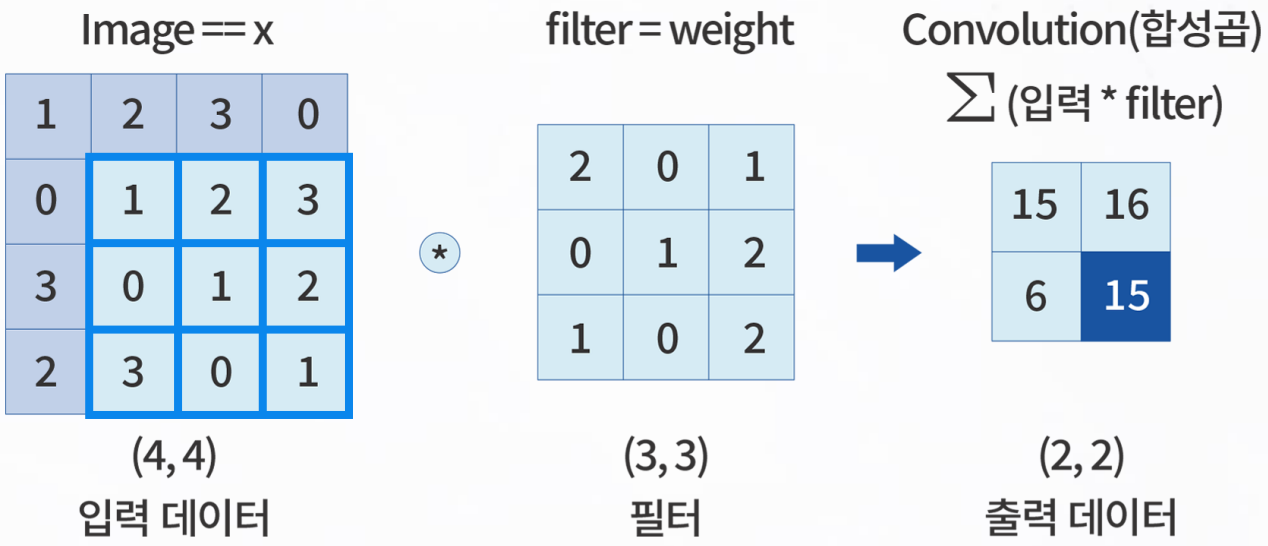
- 아래의 그림은 세 번째 윈도우 부분으로 합성곱 연산의 계산을 그린 것임
  - 아래의 그림에서 윈도우는 파란색 3 x 3 부분임
    - 출력 데이터는 입력 데이터의 윈도우 부분과 필터에서 대응하는 원소끼리 곱한 후 그 총합임



$$0 * 2 + 1 * 0 + 2 * 1 + 3 * 0 + 0 * 1 + 1 * 2 + 2 * 1 + 3 * 0 + 0 * 2 = 6$$

05 | 합성곱 계층: 합성곱 연산

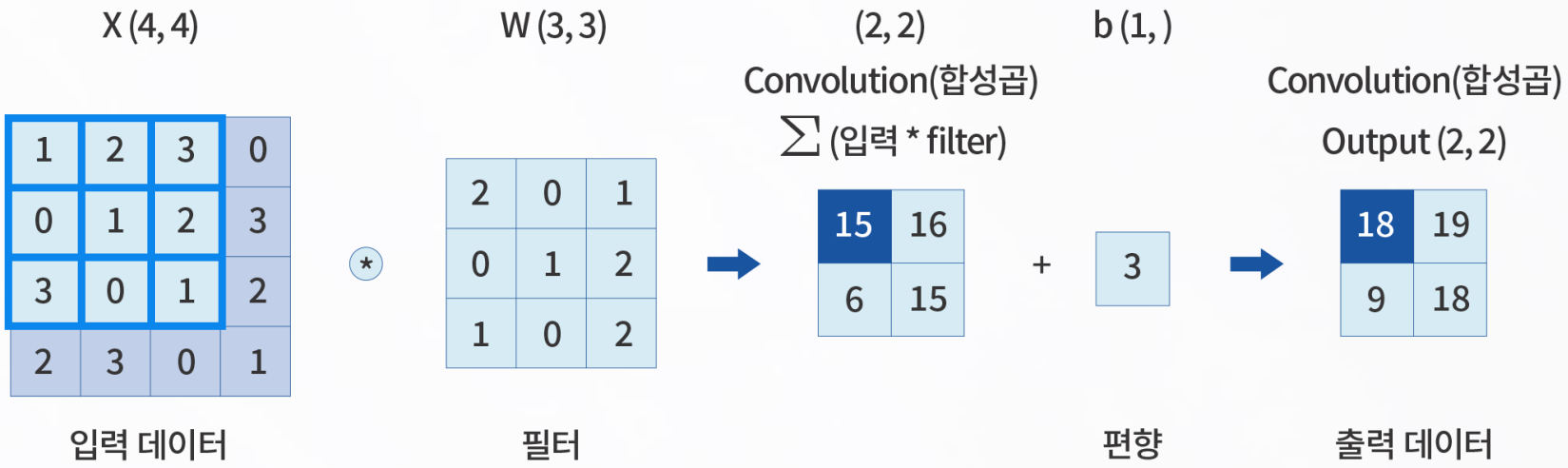
- 아래의 그림은 네 번째 윈도우 부분으로 합성곱 연산의 계산을 그린 것임
  - 아래의 그림에서 윈도우는 파란색 3 x 3 부분임
    - 출력 데이터는 입력 데이터의 윈도우 부분과 필터에서 대응하는 원소끼리 곱한 후 그 총합임



$$1 * 2 + 2 * 0 + 3 * 1 + 0 * 0 + 1 * 1 + 2 * 2 + 3 * 1 + 0 * 0 + 1 * 2 = 15$$

05 | 합성곱 계층: 합성곱 연산

- 완전연결 신경망에는 가중치 매개변수와 편향이 존재함
  - 합성곱 신경망(CNN)에서는 필터의 매개변수가 '가중치'에 해당함
    - CNN에도 편향이 존재하며, 필터를 적용한 후의 데이터에 더해줌
      - 아래 그림과 같이 편향은 항상 하나(1 x 1)만 존재함
        - 그 하나의 값을 필터를 적용한 모든 원소에 더하는 것임



$$(1 * 2 + 2 * 0 + 3 * 1 + 0 * 0 + 1 * 1 + 2 * 2 + 3 * 1 + 0 * 0 + 1 * 2) + 3 = 18$$

05 | 합성곱 계층: 합성곱 연산

- 다음은 아래 그림의 합성곱 연산을 파이썬 코드로 구현해 보자.
  - 아래의 그림에서 윈도우는 파란색 3 x 3 부분임
    - 여기서 윈도우는 한 칸씩 이동함

합성곱 연산의 예: 합성곱 연산을 ⊛기호로 표기

05 | 합성곱 계층: 합성곱 연산

다음은 앞의 **합성곱 연산**을 **구현**한 **파이썬 코드**이다.

- 실행결과 아래와 같이 합성곱 연산이 잘 계산된 것을 볼 수 있음

```
# 입력 행렬 (4x4)
input_matrix = np.array([
    [1, 2, 3, 0],
    [0, 1, 2, 3],
    [3, 0, 1, 2],
    [2, 3, 0, 1] ])

# 커널 (3x3)
kernel = np.array([
    [2, 0, 1],
    [0, 1, 2],
    [1, 0, 2] ])

# 출력 행렬 초기화 (2x2)
output_matrix = np.zeros((2, 2))

# 합성곱 연산 수행
for i in range(2):  # 출력 행렬의 행 크기
    for j in range(2):  # 출력 행렬의 열 크기
        sub_matrix = input_matrix[i:i+3, j:j+3]
        output_matrix[i, j] = np.sum(sub_matrix * kernel)

print(output_matrix)
```

```
[[15. 16.]
 [ 6. 15.]]
```