강원지역혁신플랫폼

기계학습

Machine Learning

패딩, 스트라이드, 3차원과 4차원 데이터의 합성곱 연산,
합성공 계층에서 배치 처리의 개념

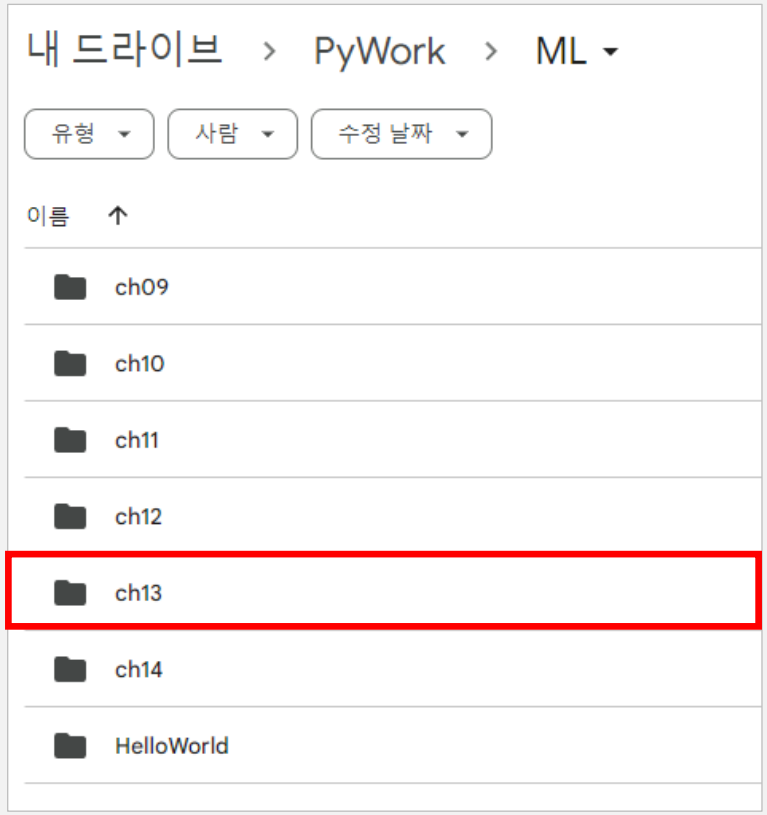
학습목표

패딩, 스트라이드, 3차원 데이터의 합성곱 연산,
합성곱 계층에서 배치 처리의 개념을 이해할 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



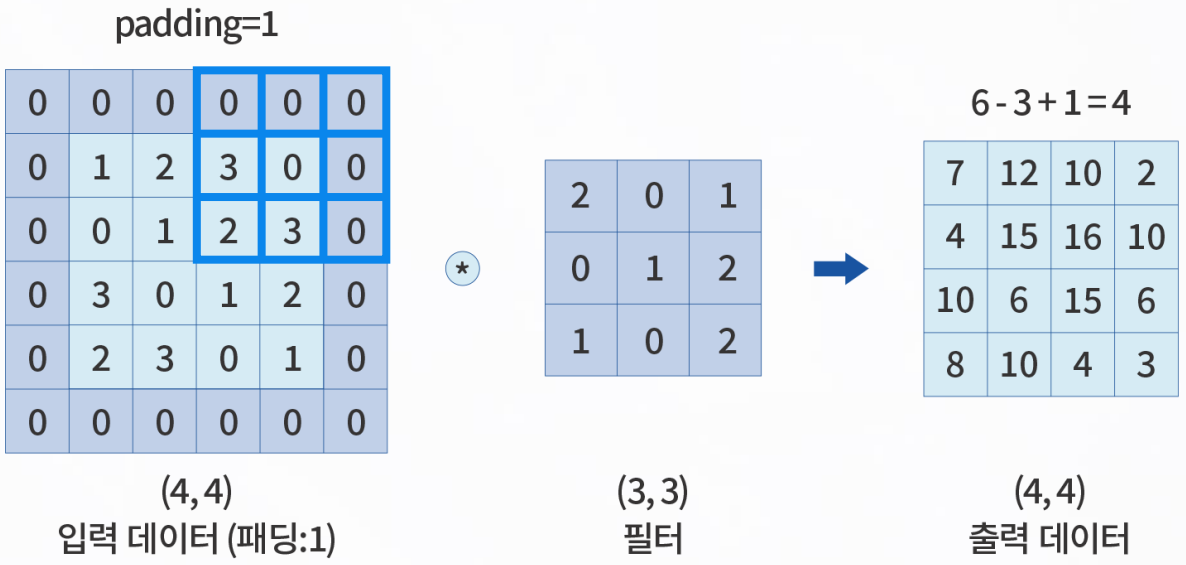
01 | 실습

✦ “ML > ch13 >” 폴더를 클릭함

› 아래의 [ch13_02_패딩, 스트라이드, 3차원과 4차원 데이터의 합성곱 연산.ipynb] 스크립트를 클릭함

패딩

- 합성곱 연산을 수행하기 전에 입력 데이터 주변을 특정 값(예컨대 0)으로 채우기도 함
  - 이를 패딩(Padding)이라 함
    - 합성곱 연산에서 자주 이용하는 기법임



02 | 합성곱 계층: 패딩 (Padding)

- 아래 그림은 (4, 4) 크기의 입력 데이터에 폭이 1인 패딩을 적용한 모습임
  - 폭 1짜리 패딩이라 하면 입력 데이터 사방 1픽셀을 특정 값으로 채우는 것임
    - 여기서는 0으로 채웠음

padding=1

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

(4, 4)
입력 데이터 (패딩:1)

02 | 합성곱 계층: 패딩 (Padding)

- 아래 그림에서 처음에 크기가 (4, 4)인 입력 데이터에 폭 1짜리 패딩이 추가됨
  - 입력 데이터의 크기가 (4, 4)에서 (6, 6)이 됨

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

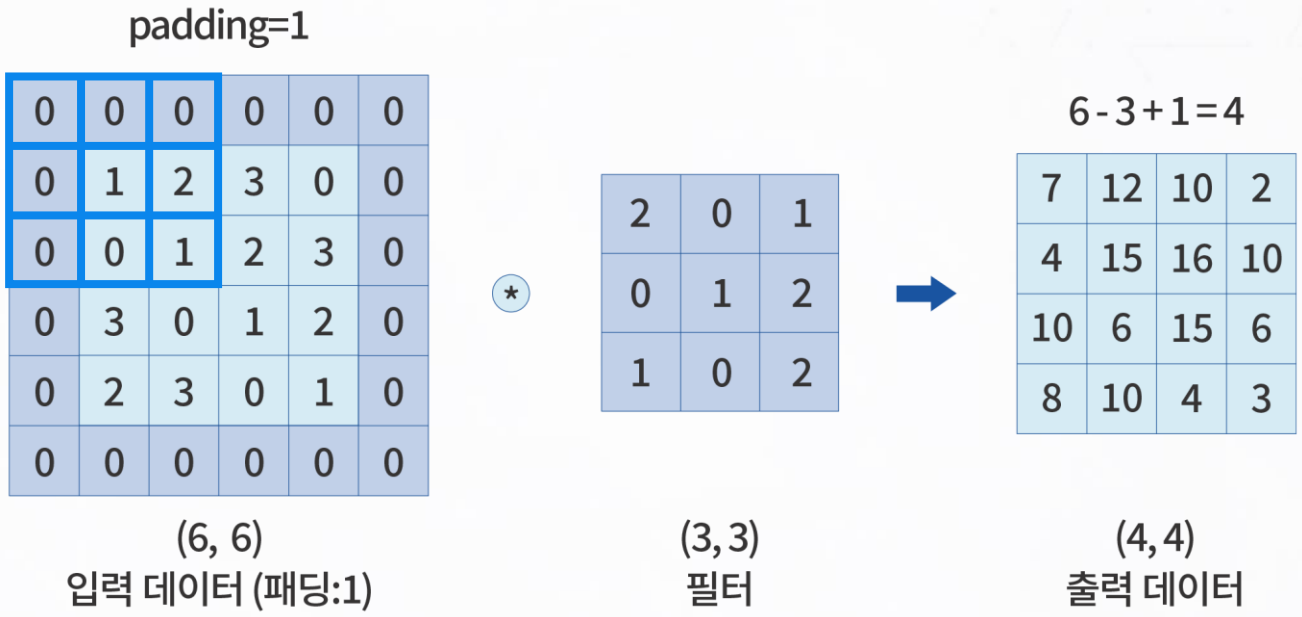
(4, 4)
입력 데이터

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

(6, 6)
입력 데이터 (패딩:1)

02 | 합성곱 계층: 패딩 (Padding)

- 아래 그림에서 (6, 6) 입력에 (3, 3) 크기의 필터를 걸면 (4, 4) 크기의 출력 데이터가 생성됨
  - ✦ 패딩은 1, 2, 3 등 원하는 정수로 설정할 수 있음
    - › 패딩을 2로 설정하면 입력 데이터는 (8, 8)이 되고, 패딩을 3로 설정하면 입력 데이터는(10, 10)이 됨



02 | 합성곱 계층: 패딩 (Padding)

- 패딩은 주로 **출력 크기**를 **조정할 목적**으로 **사용**함
  - 예를 들어 **(4, 4) 입력** 데이터에 **(3, 3) 필터**를 적용하면 **출력**은 **(2, 2)**가 되어, 입력보다 2만큼 줄어듦
  - **합성곱 연산**을 **거칠 때**마다 **크기**가 **작아지면 어느 시점**에서는 **출력 크기**가 1이 되어, 더 이상은 **합성곱 연산**을 **적용할 수 없게 됨**
    - **이러한 사태**를 **막기 위해 패딩**을 **사용**함

02 | 합성곱 계층: 패딩 (Padding)

- 아래와 같이 패딩의 폭을 1로 설정하니 (4, 4) 입력에 대한 출력이 같은 크기인 (4, 4)로 유지된 것을 알 수 있음
  - 한 마디로 입력 데이터의 공간적 크기를 고정한 채로 다음 계층에 전달할 수 있음

padding=1

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

(6, 6)
입력 데이터 (패딩:1)

*

2	0	1
0	1	2
1	0	2

(3, 3)
필터

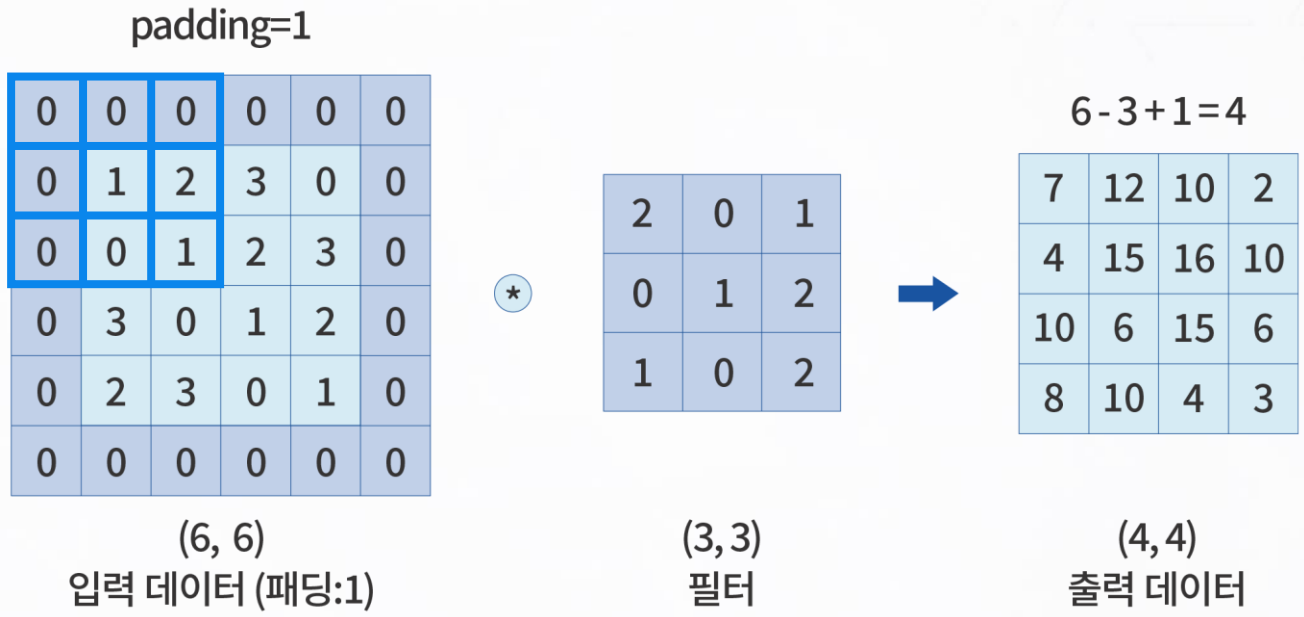
6 - 3 + 1 = 4

7	12	10	2
4	15	16	10
10	6	15	6
8	10	4	3

(4, 4)
출력 데이터

02 | 합성곱 계층: 패딩 (Padding)

- 다음은 아래 그림과 같이 **처음**에 **크기**가 **(4, 4)**인 **입력 데이터**에 **폭 1짜리 패딩**이 **추가**되어 **합성곱 연산**을 수행하는 **파이썬 코드**로 구현해 보자.
  - 아래의 그림에서 **윈도우**는 **파란색 3 x 3 부분**임
    - 여기서 **윈도우**는 **한 칸씩 이동**함



02 | 합성곱 계층: 패딩 (Padding)

다음은 앞의 합성곱 연산을 구현한 파이썬 코드이다.

- 실행결과 아래와 같이 입력 데이터에 폭 1짜리 패딩이 적용된 합성곱 연산의 출력 데이터의 형상이 (4, 4)인 것을 볼 수 있음

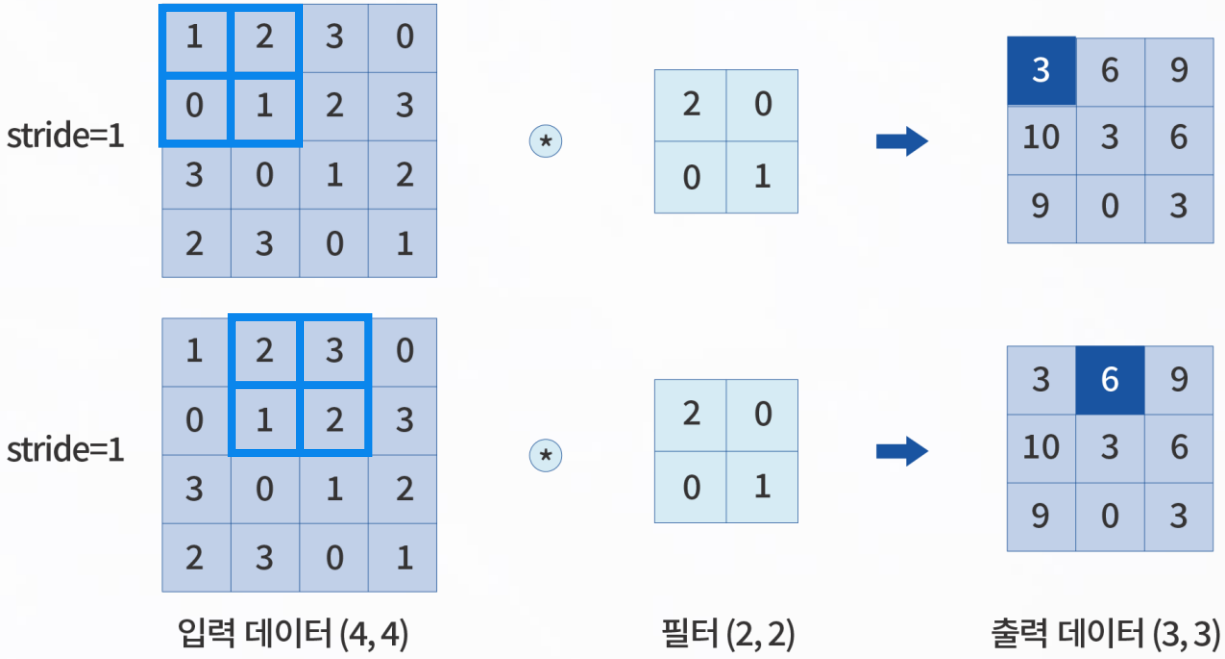
```
# 원본 입력 행렬 (4x4)
input_matrix = np.array([
   [1, 2, 3, 0],
   [0, 1, 2, 3],
   [3, 0, 1, 2],
   [2, 3, 0, 1] ])

# 패딩 적용 (6x6)
padded_input = np.pad(input_matrix, pad_width=1, mode='constant', constant_values=0)
print(padded_input.shape)
print(padded_input)

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

# 출력 행렬 초기화 (4x4)
output_matrix = np.zeros((4, 4))

# 합성곱 연산 수행
for i in range(4):  # 출력 행렬의 행 크기
   for j in range(4):  # 출력 행렬의 열 크기
      sub_matrix = padded_input[i:i+3, j:j+3]
      output_matrix[i, j] = np.sum(sub_matrix * kernel)

print(output_matrix)
```

```
(6, 6)
[[0 0 0 0 0 0]
 [0 1 2 3 0 0]
 [0 0 1 2 3 0]
 [0 3 0 1 2 0]
 [0 2 3 0 1 0]
 [0 0 0 0 0 0]]
[[ 7. 12. 10.  2.]
 [ 4. 15. 16. 10.]
 [10.  6. 15.  6.]
 [ 8. 10.  4.  3.]]
```

02 | 합성곱 계층: 스트라이드 (Stride)

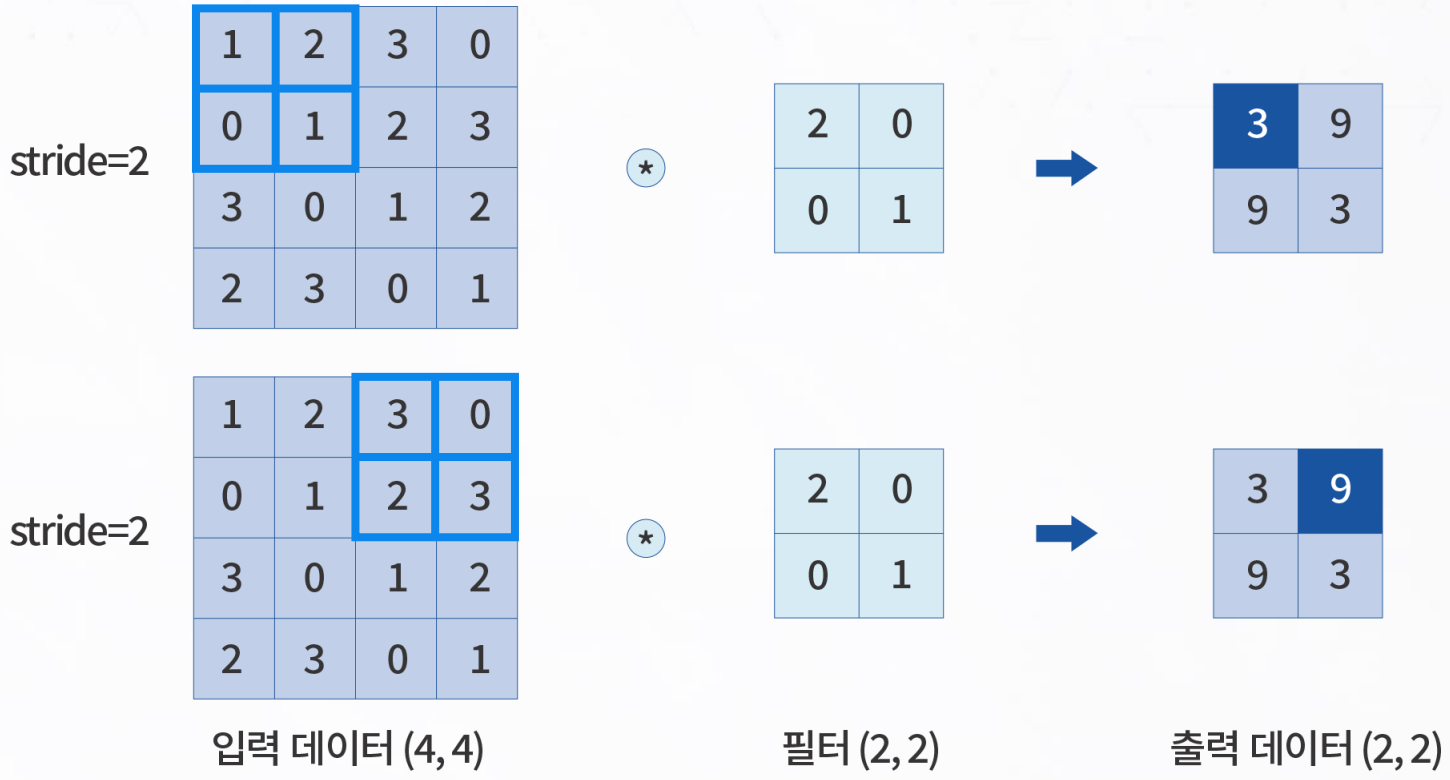
스트라이드

- 필터를 적용하는 위치의 간격을 스트라이드(stride)라고 함
  - 예를 들어 스트라이드를 1로 하면 필터를 적용하는 윈도우가 한 칸씩 이동함
    - 아래 그림에서 크기가 (4, 4)인 입력 데이터에 스트라이드를 1로 설정한 필터를 적용하면 출력은 (3, 3)으로 크기는 작아짐



02 | 합성곱 계층: 스트라이드 (Stride)

- 예를 들어 스트라이드를 2로 하면 필터를 적용하는 윈도우가 두 칸씩 이동함
  - 아래 그림에서 크기가 (4, 4)인 입력 데이터에 스트라이드를 2로 설정한 필터를 적용하면 출력은 (2, 2)로 크기는 작아짐



02 | 합성곱 계층: 스트라이드 (Stride)

- 정리해 보면 스트라이드를 키우면 출력 크기는 작아지고, 패딩을 크게 하면 출력 크기가 커짐
  - 이러한 관계를 수식화하면 다음과 같음
    - 입력 크기를 (H, W), 필터 크기를 (FH, FW), 출력 크기를 (OH, OW)
    - 패딩을 P, 스트라이드를 S라고 함
    - 출력 크기는 다음 식으로 계산함

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

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

OH : 출력 높이 크기
H : 입력 높이 크기
P : 패딩
FH: 필터 높이 크기
S : 스트라이드
OW: 출력 너비 크기
W: 입력 너비 크기
FW: 필터 너비 크기

H: height
P: padding
F: filter
O: output
W: width
S: stride
OH: output height

02 | 합성곱 계층: 스트라이드 (Stride)

다음은 입력 크기를 (4, 4), 필터 크기를 (3, 3), 패딩 1, 스트라이드 1로 하는 출력 크기 (OH, OW)를 계산하는 예제임

$$OH = \frac{H + 2P - FH}{S} + 1 \qquad OW = \frac{W + 2P - FW}{S} + 1$$

$$OH = \frac{4 + 2 * 1 - 3}{1} + 1 \qquad OW = \frac{4 + 2 * 1 - 3}{1} + 1$$

$$= 3 + 1 = 4 \qquad = 3 + 1 = 4$$

padding=1
stride=1

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

(6, 6)
입력 데이터

*

2	0	1
0	1	2
1	0	2

(3,3)
필터

→

7	12	10	2
4	15	16	10
10	6	15	6
8	10	4	3

(4,4)
출력 데이터

02 | 합성곱 계층: 스트라이드 (Stride)

- 다음은 입력 크기를 (7, 7), 필터 크기를 (3, 3), 패딩 0, 스트라이드 2로 하는 출력 크기 (OH, OW)를 계산하는 예제임

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

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

$$OH = \frac{7 + 2 * 0 - 3}{2} + 1 = 2 + 1 = 3$$

$$OW = \frac{7 + 2 * 0 - 3}{2} + 1 = 2 + 1 = 3$$

$$(OH, OW) = (3, 3)$$

02 | 합성곱 계층: 스트라이드 (Stride)

- 다음은 입력 크기를 (28, 31), 필터 크기를 (5, 5), 패딩 2, 스트라이드 3으로 하는 출력 크기 (OH, OW)를 계산하는 예제임

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

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

$$OH = \frac{28 + 2 * 2 - 5}{3} + 1 = 9 + 1 = 10$$

$$OW = \frac{31 + 2 * 2 - 5}{3} + 1 = 10 + 1 = 11$$

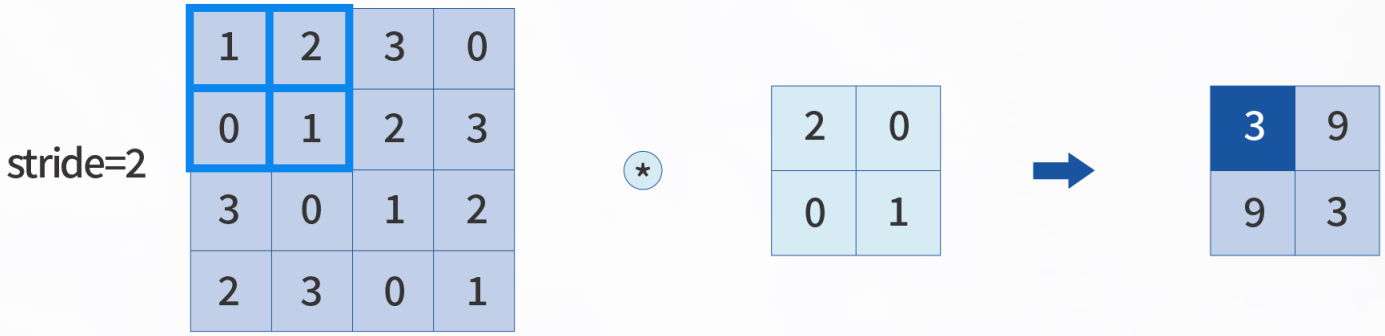
$$(OH, OW) = (10, 11)$$

02 | 합성곱 계층: 스트라이드 (Stride)

- 출력 크기 (OH, OW)가 정수로 나눠 떨어지는 값이어야 한다는 점에 주의해야 함
  - 참고로, 출력 크기가 정수로 딱 나눠 떨어지지 않을 때는 가장 가까운 정수로 반올림하는 등의 처리가 필요함
    - 특별히 에러를 내지 않고 진행하도록 구현하는 경우도 있음

02 | 합성곱 계층: 스트라이드 (Stride)

- 다음은 아래 그림의 합성곱 연산을 파이썬 코드로 구현해 보자.
  - 아래의 그림과 같이 윈도우는 파란색 2 x 2 부분임
    - 스트라이드를 2로 하면 필터를 적용하는 윈도우가 두 칸씩 이동함
    - 아래 그림에서 크기가 (4, 4)인 입력 데이터에 스트라이드를 2로 설정한 필터를 적용하면 출력은 (2, 2)으로 크기는 작아짐



02 | 합성곱 계층: 스트라이드 (Stride)

다음은 앞의 합성곱 연산을 구현한 파이썬 코드이다.

- 실행결과 스트라이드를 2로 설정한 경우 합성곱 연산을 수행한 결과 출력 데이터의 형상이 (2, 2)로 크기가 작아진 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 (2x2)
kernel = np.array([
  [2, 0],
  [0, 1] ])

# 스트라이드 값
stride = 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*stride:i*stride+2, j*stride:j*stride+2]
    output_matrix[i, j] = np.sum(sub_matrix * kernel)

print(output_matrix)
```

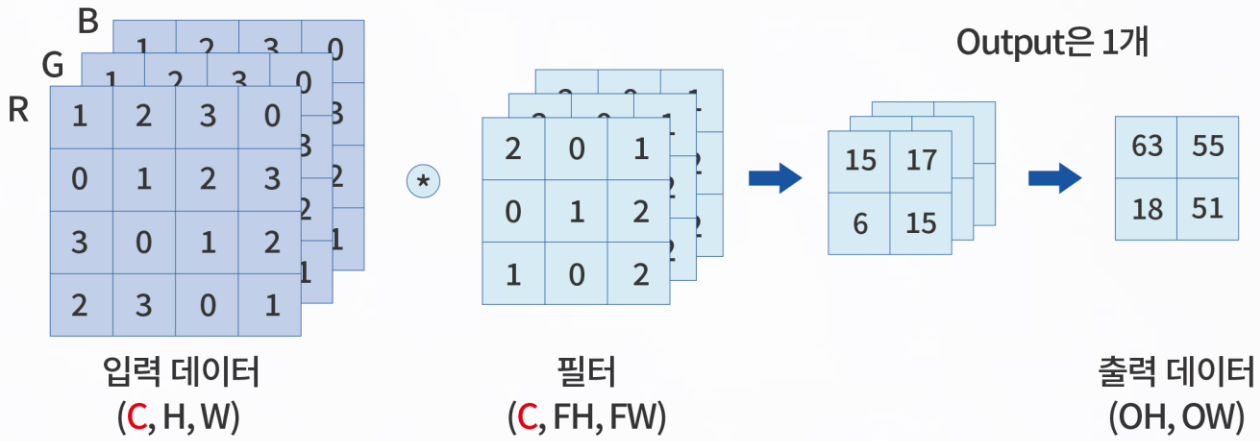
```
[[3. 9.]
 [9. 3.]]
```

03 | 3차원 데이터의 합성곱 연산

3차원 데이터의 합성곱 연산

앞에서 2차원 형상을 다루는 합성곱 연산을 살펴보았음

- 이미지는 아래 그림과 같이 세로(height) · 가로(width)에 더해서 채널(channel)까지 고려한 3차원 데이터임

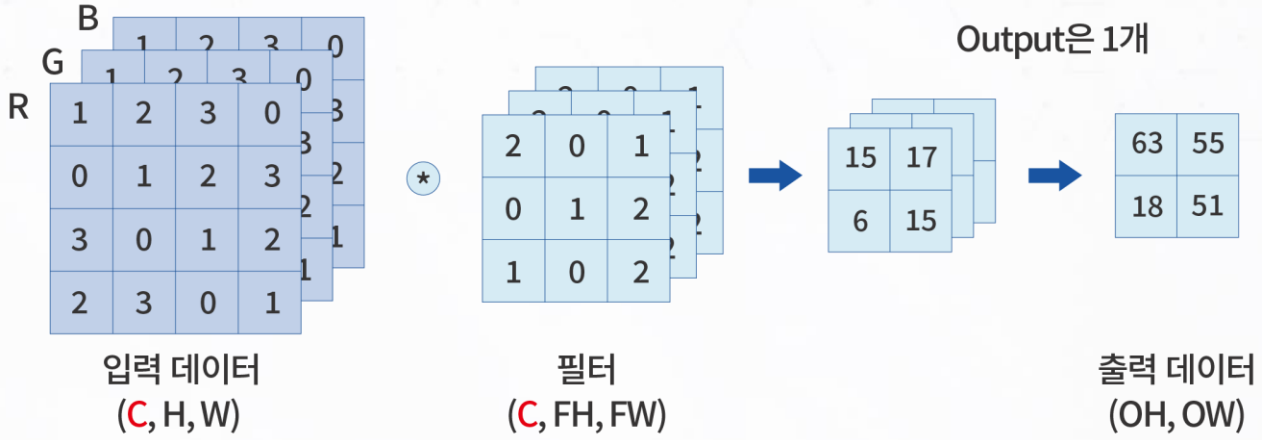


3차원 데이터 합성곱 연산의 예

03 | 3차원 데이터의 합성곱 연산

다음은 3차원 데이터의 합성곱 연산 예임

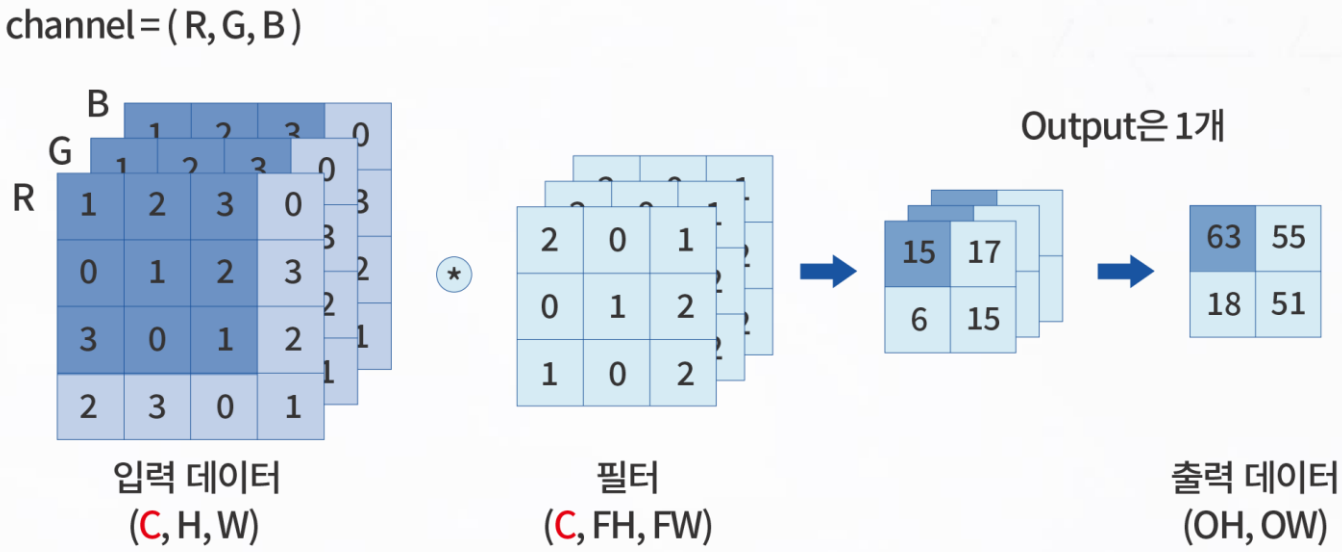
✦ 2차원 데이터의 합성곱 일 때와 비교하면, 길이 방향(채널 방향)으로 특징 맵이 늘어났음



3차원 데이터 합성곱 연산의 예

03 | 3차원 데이터의 합성곱 연산

- 아래의 그림은 계산 순서를 나타냄
  - ✦ 채널 쪽으로 특징 맵이 여러 개 있다면 입력 데이터와 필터의 합성곱 연산을 채널마다 수행함
    - › 그 결과를 더해서 하나의 출력을 얻음

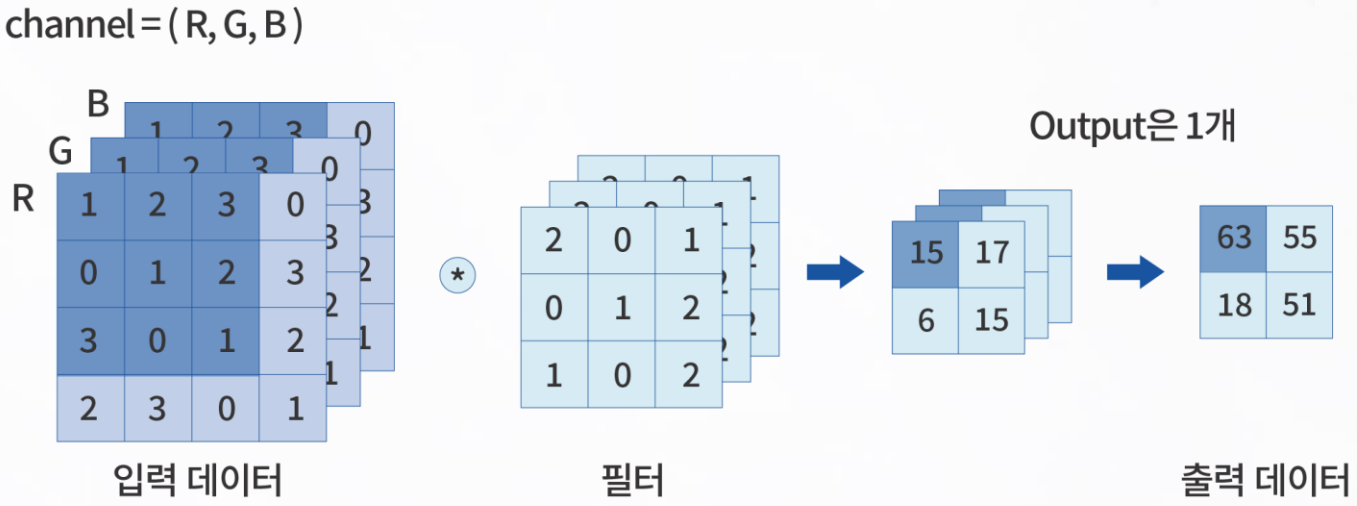


3차원 데이터 합성곱 연산의 예

03 | 3차원 데이터의 합성곱 연산

- 다음은 3차원 데이터의 합성곱 연산 예임
  - 아래 그림은 첫 번째 채널 출력 데이터 계산 결과임
    - 출력 데이터의 값은 첫 번째, 두 번째와 세 번째 계산 결과를 더함

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

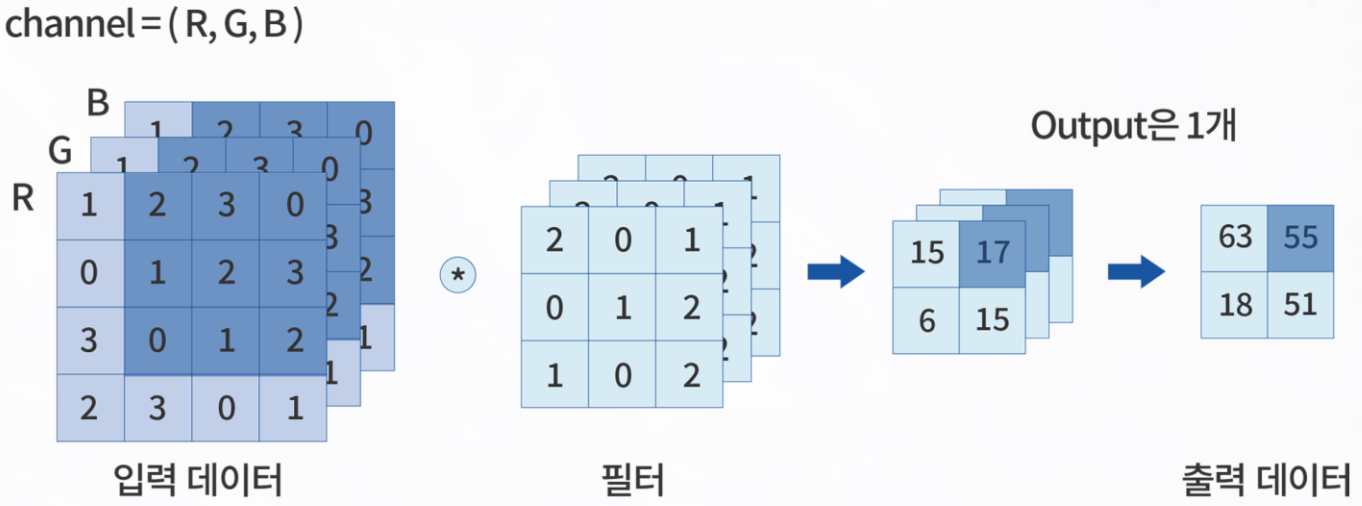


3차원 데이터 합성곱 연산의 예

03 | 3차원 데이터의 합성곱 연산

- 다음은 3차원 데이터의 합성곱 연산 예임
  - 아래 그림은 첫 번째 채널 출력 데이터 계산 결과임
    - 출력 데이터의 값은 첫 번째, 두 번째와 세 번째 계산 결과를 더함

$$2 * 2 + 3 * 0 + 0 * 1 + 1 * 0 + 2 * 1 + 3 * 2 + 0 * 1 + 1 * 0 + 2 * 2 = 17$$

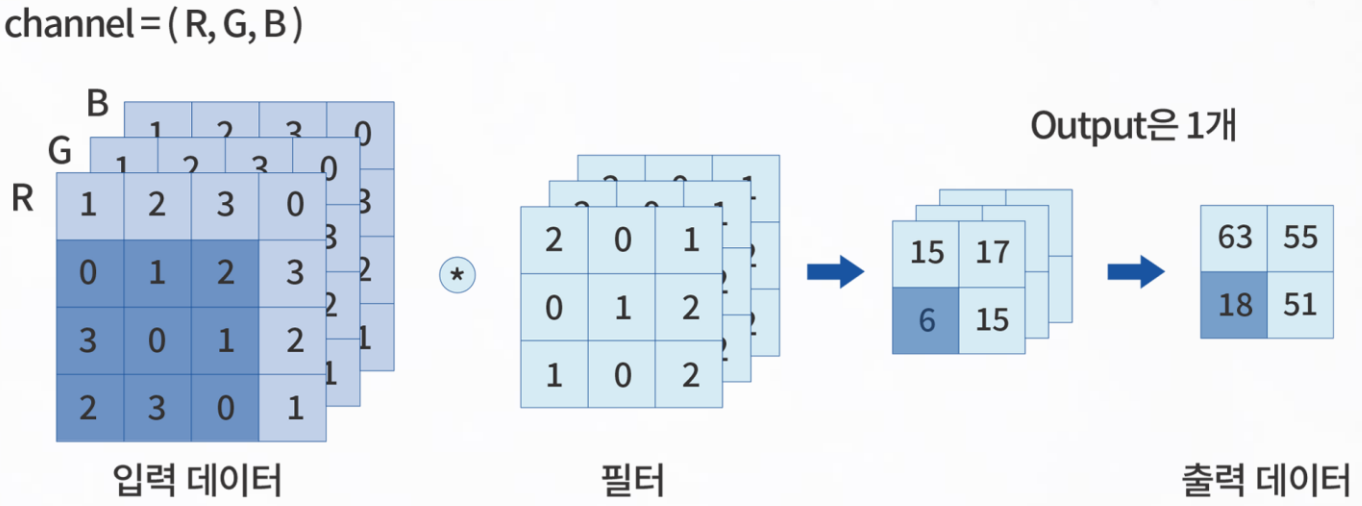


3차원 데이터 합성곱 연산의 예

03 | 3차원 데이터의 합성곱 연산

- 다음은 3차원 데이터의 합성곱 연산 예임
  - 아래 그림은 첫 번째 채널 출력 데이터 계산 결과임
    - 출력 데이터의 값은 첫 번째, 두 번째와 세 번째 계산 결과를 더함

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



3차원 데이터 합성곱 연산의 예

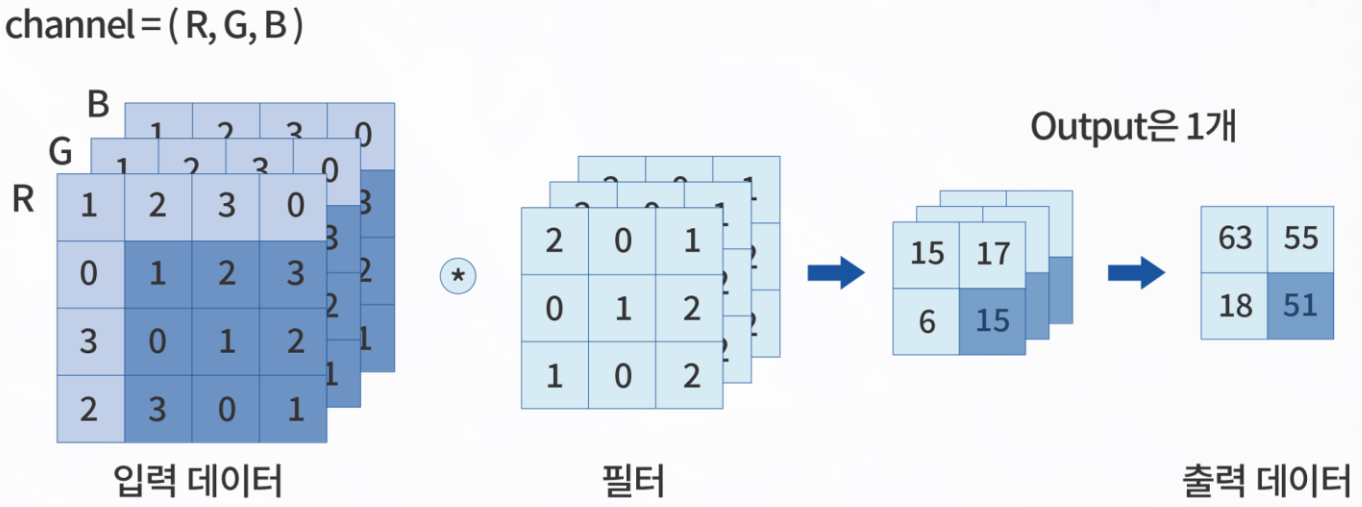
03 | 3차원 데이터의 합성곱 연산

다음은 3차원 데이터의 합성곱 연산 예임

아래 그림은 첫 번째 채널 출력 데이터 계산 결과임

출력 데이터의 값은 첫 번째, 두 번째와 세 번째 계산 결과를 더함

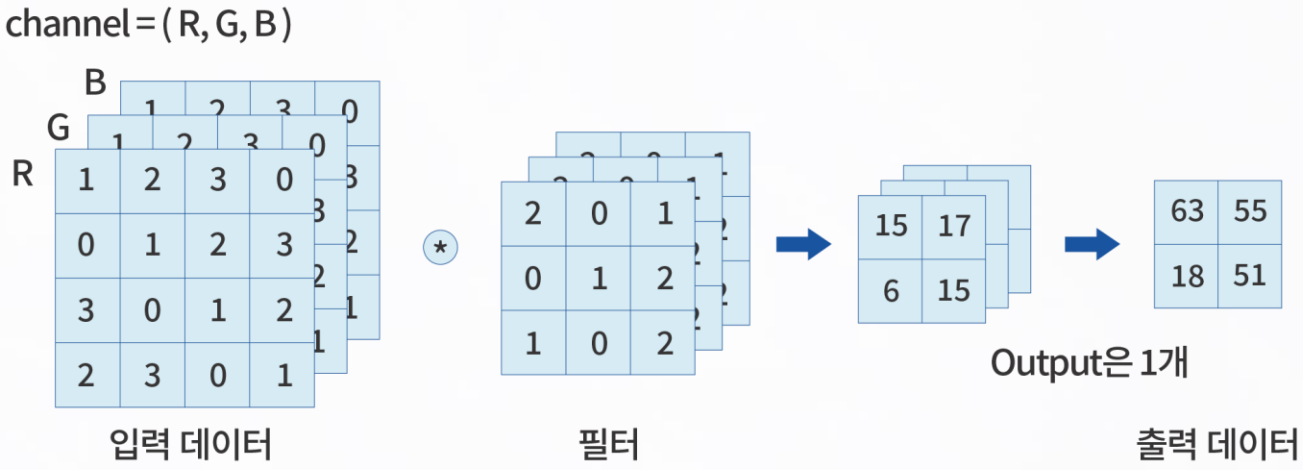
$$1 * 2 + 2 * 0 + 3 * 1 + 0 * 0 + 1 * 1 + 2 * 2 + 3 * 1 + 0 * 0 + 1 * 2 = 15$$



3차원 데이터 합성곱 연산의 예

03 | 3차원 데이터의 합성곱 연산

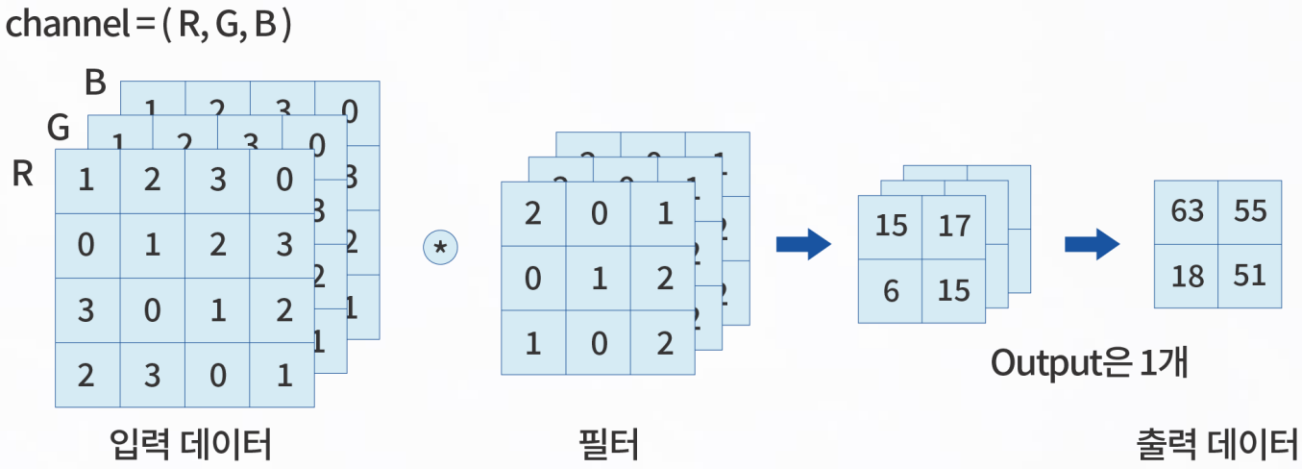
- 3차원의 합성곱 연산에서 주의할 점은 입력 데이터의 채널 수와 필터의 채널 수가 같아야 한다는 것임
  - 아래의 예에서는 모두 세 개로 일치함
    - 필터 자체의 크기는 원하는 값으로 설정할 수 있음
      - 단, 모든 채널의 필터가 같은 크기여야 함



3차원 데이터 합성곱 연산의 예

03 | 3차원 데이터의 합성곱 연산

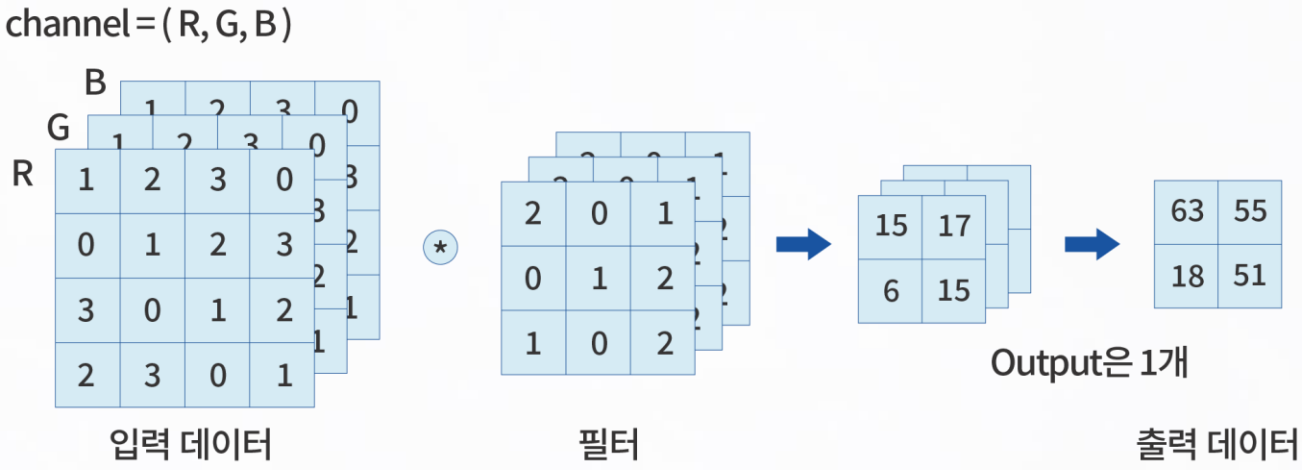
- 아래 그림에서는 필터의 크기가 (3, 3)이지만, 원한다면 (2, 2)나 (1, 1)또는 (5, 5)등으로 설정할 수 있음
  - 정리하면, 필터의 채널 수는 입력 데이터의 채널 수와 같도록 설정해야 함



3차원 데이터 합성곱 연산의 예

03 | 3차원 데이터의 합성곱 연산

- 다음은 아래 그림과 같이 3차원 데이터의 합성곱 연산을 파이썬 코드로 구현해 보자.
  - 아래의 그림에서 윈도우는 파란색 3 x 3 부분임
    - 여기서 윈도우는 1칸씩 이동함



3차원 데이터 합성곱 연산의 예

03 | 3차원 데이터의 합성곱 연산

다음은 앞의 합성곱 연산을 구현한 파이썬 코드이다.

실행결과 아래와 같이 3차원 데이터의 합성곱 연산이 잘 계산된 것을 볼 수 있음

```
# 입력 행렬 (4x4x3)
input_matrix = np.array([
  [
    [1, 2, 3, 0],
    [0, 1, 2, 3],
    [3, 0, 1, 2],
    [2, 3, 0, 1]
  ],
  [
    [1, 2, 3, 0],
    [0, 1, 2, 3],
    [3, 0, 1, 2],
    [2, 3, 0, 1]
  ],
  [
    [1, 2, 3, 0],
    [0, 1, 2, 3],
    [3, 0, 1, 2],
    [2, 3, 0, 1]
  ]
])
```

```
# 커널 (3x3x3)
kernel = np.array([
  [
    [2, 0, 1],
    [0, 1, 2],
    [1, 0, 2]
  ],
  [
    [2, 0, 1],
    [0, 1, 2],
    [1, 0, 2]
  ],
  [
    [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
    conv_sum = 0
    for k in range(3):  # 각 채널에 대해 합성곱 수행
      sub_matrix = input_matrix[k, i:i+3, j:j+3]
      conv_sum += np.sum(sub_matrix * kernel[k])
    output_matrix[i, j] = conv_sum

print(output_matrix)
```

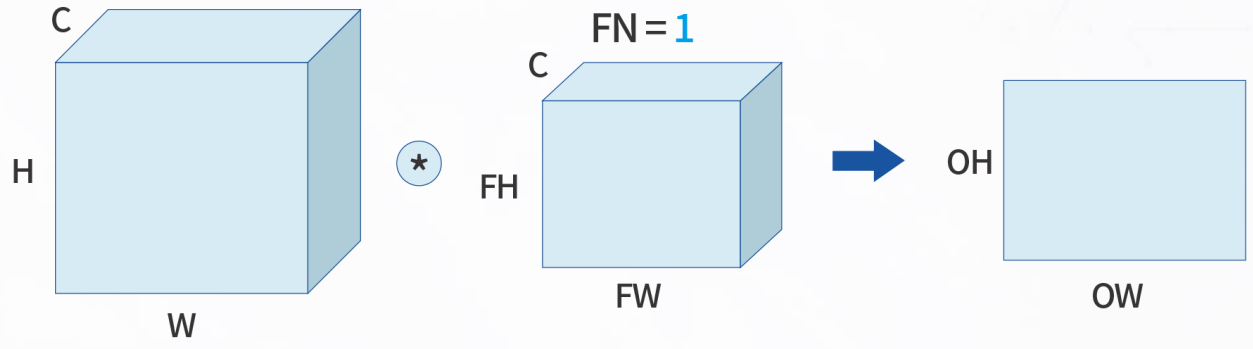
```
[[45. 48.]
 [18. 45.]]
```

04 | 3차원 데이터의 합성곱 연산을 블록(C,H,W)으로

3차원 데이터의 합성곱 연산을 블록으로 생각하기

3차원의 합성곱 연산은 데이터와 필터를 직육면체 블록으로 나타낼 수 있음

- 블록은 아래 그림과 같은 3차원 직육면체임

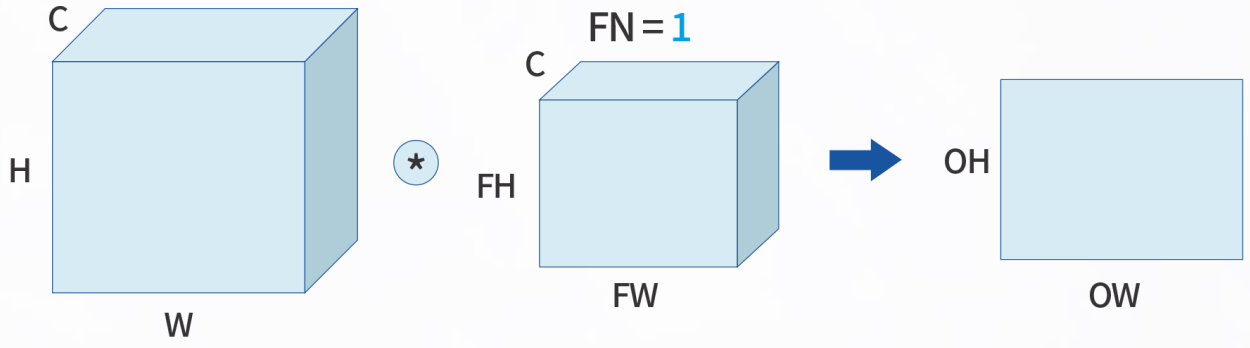


입력 데이터 (C, H, W) | 필터 (C, FH, FW) | 출력 데이터 (1, OH, OW)

3차원 데이터 합성곱 연산을 블록으로 나타냄 예

04 | 3차원 데이터의 합성곱 연산을 블록(C,H,W)으로

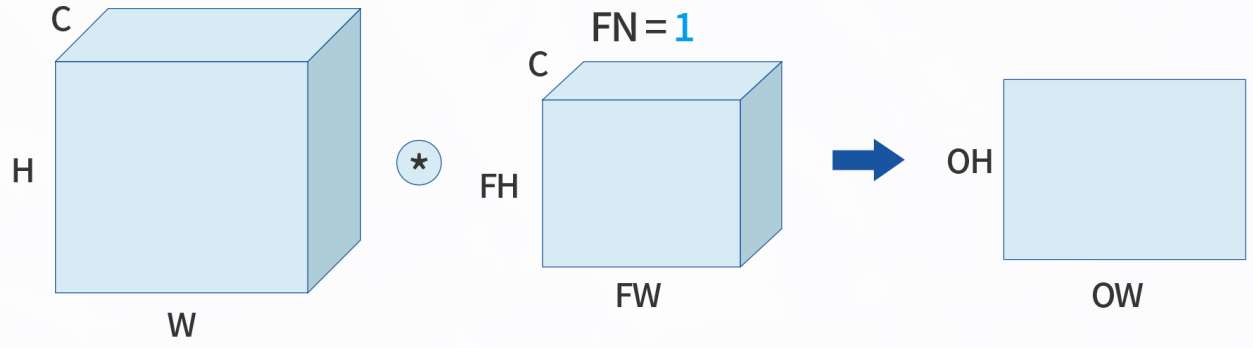
- 3차원 데이터를 다차원 배열로 나타낼 때는 (채널, 높이, 너비)순서로 표기하기로 함
  - 예를 들어 채널 수 C, 높이 H, 너비 W인 데이터의 형상은 (C, H, W)로 씀
    - 채널 수 C, 필터 높이 FH(Filter Height), 필터 너비 FW(Filter Width)의 경우 데이터의 형상은 (C, FH, FW)로 씀
    - 합성곱 연산을 직육면체 블록으로 생각할 수 있고, 블록의 형상에 주의해야 함



3차원 데이터 합성곱 연산을 블록으로 나타냄 예

04 | 3차원 데이터의 합성곱 연산을 블록(C,H,W)으로

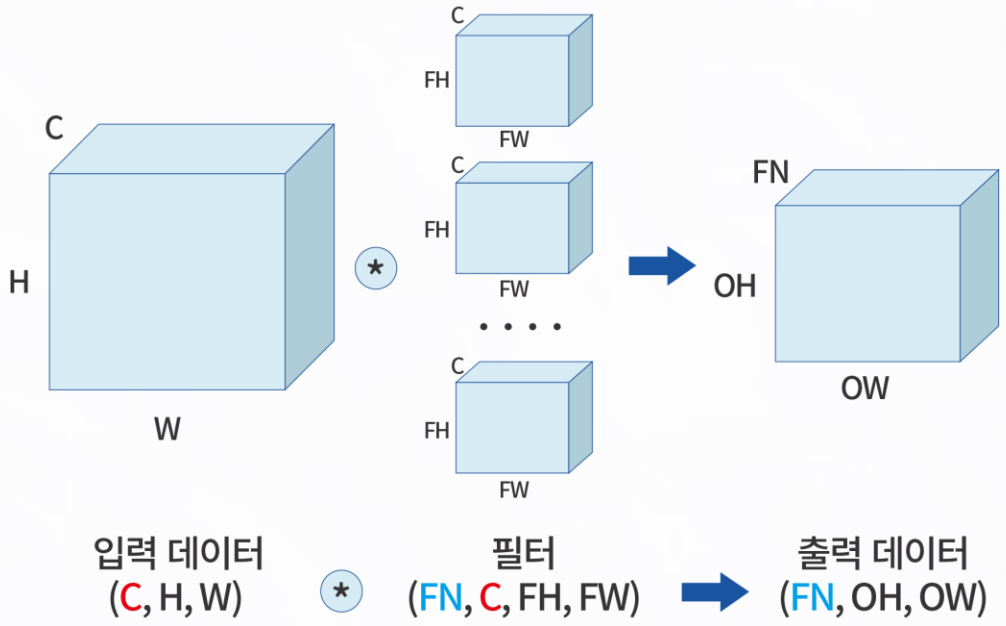
- 아래 그림에서 출력 데이터는 한 장의 특징 맵인 것을 볼 수 있음
  - 한 장의 맵을 다른 말로 하면 채널이 한 개인 특징 맵이라 함
  - 그럼 합성곱 연산의 출력으로 다수의 채널을 내보내려면 어떻게 해야 할까요?
    - 그 답은 필터(가중치)를 다수 사용하는 것임



3차원 데이터 합성곱 연산을 블록으로 나타냄 예

04 | 3차원 데이터의 합성곱 연산을 블록(C,H,W)으로

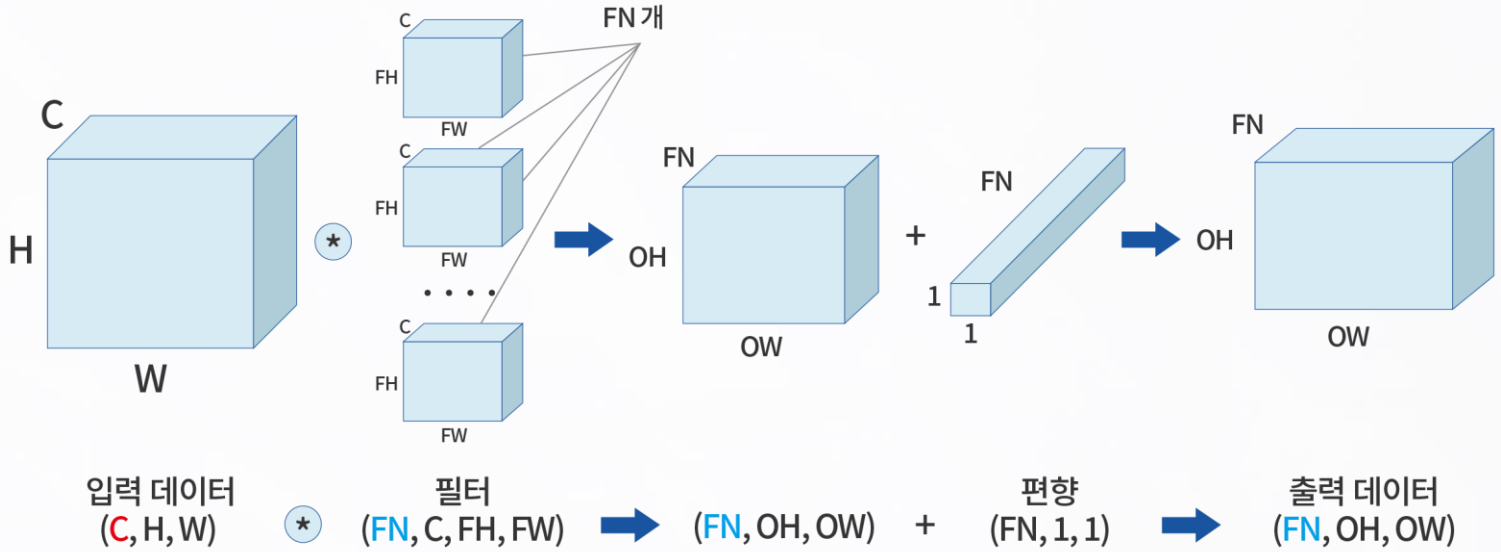
- 아래 그림은 여러 필터를 사용한 합성곱 연산임
  - 아래의 그림처럼 필터를 FN(필터의 수)개 적용하면 출력 맵도 FN개가 생성됨
    - 그 FN개의 맵을 모으면 출력 형상이 (FN, OH, OW)인 블록이 완성됨
    - 이 완성된 블록을 다음 계층으로 넘기겠다는 것이 CNN의 처리 흐름임



여러 필터를 사용한 합성곱 연산의 예

04 | 3차원 데이터의 합성곱 연산을 블록(C,H,W)으로

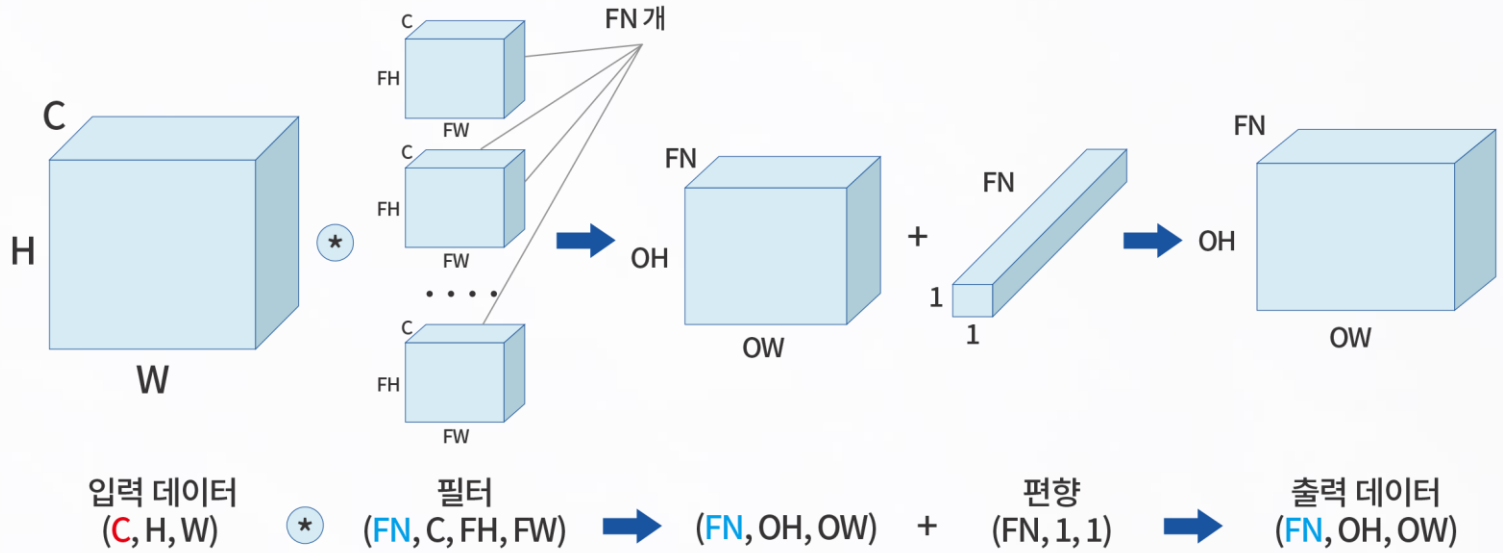
- 합성곱 연산에서는 필터의 수도 고려해야 함
  - 그런 이유로 필터의 가중치 데이터는 4차원 데이터임
    - 즉 출력 채널 수 FN, 입력 채널 수 C, 필터 높이 FH, 필터 너비 FW 순으로 씀
      - 필터의 데이터 형상 (FN, C, FH, FW)



합성곱 연산의 처리 흐름 (편향 추가)

04 | 3차원 데이터의 합성곱 연산을 블록(C,H,W)으로

- 예를 들어 채널 수 3, 크기 5 x 5인 필터가 20개 있다면 (20, 3, 5, 5)로 씀
  - 합성곱 연산에도 편향이 쓰임
    - 아래 그림에서와 같이 출력 형상(FN, OH, OW)에 편향을 더한 모습임

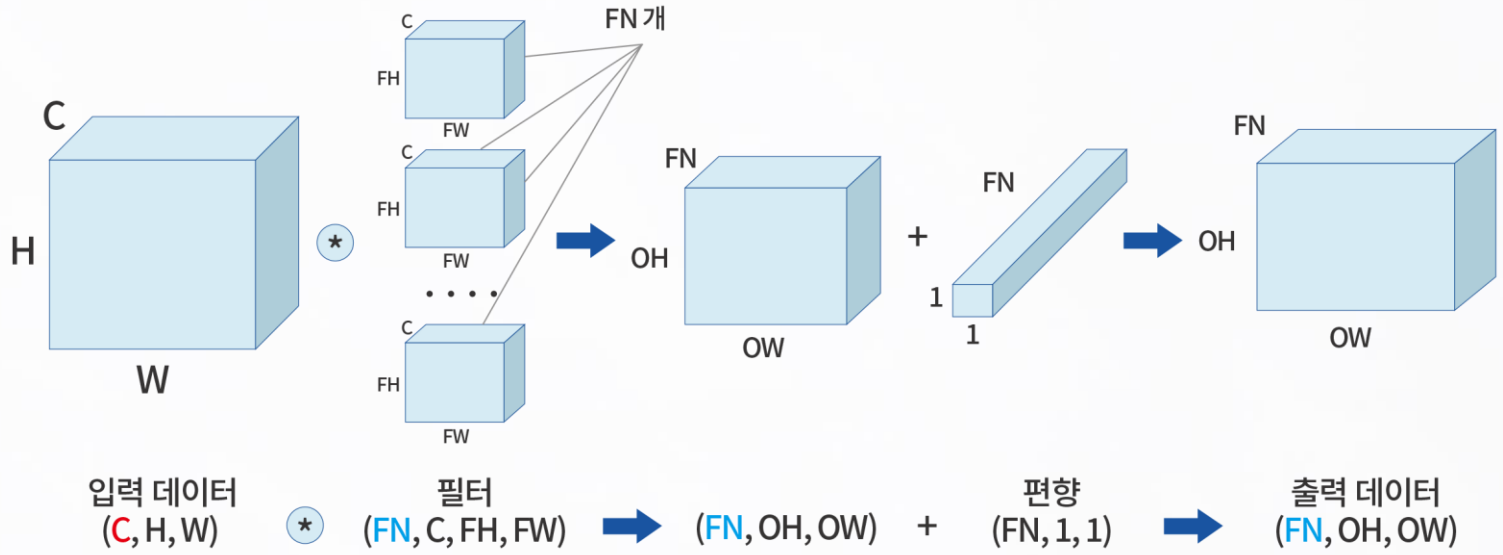


합성곱 연산의 처리 흐름 (편향 추가)

04 | 3차원 데이터의 합성곱 연산을 블록(C,H,W)으로

아래의 그림에서 보듯 편향은 채널 하나에 값 하나씩으로 구성됨

- 편향의 형상은 (FN, 1, 1)이고, 필터의 출력 결과의 형상은 (FN, OH, OW)임
  - 이 두 블록을 더하면 편향의 각 값이 필터의 출력인 (FN, OH, OW) 블록의 대응 채널의 원소 모두에 더해짐



합성곱 연산의 처리 흐름 (편향 추가)

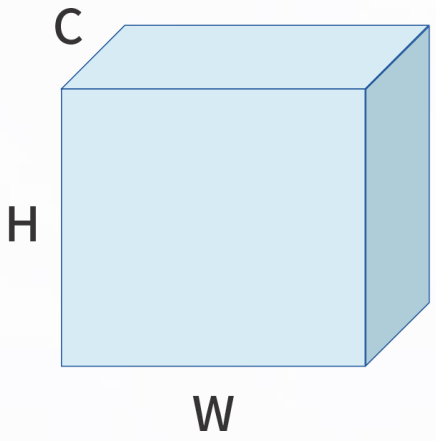
05 | 합성곱 계층에서 배치 처리

합성곱 계층에서 배치 처리

- 신경망 처리에서는 입력 데이터를 한 덩어리로 묶어 배치로 처리함
  - 완전연결 신경망 구현에서 이 방식을 지원하여 처리 효율을 높이고, 미니배치 방식의 학습도 지원하도록 했음

05 | 합성곱 계층에서 배치 처리

- 합성곱 연산도 마찬가지로 배치 처리를 지원함
  - 아래 그림과 같이 각 계층을 흐르는 데이터의 차원을 하나 늘려 4차원 데이터로 저장함
    - 좀 더 구체적으로 데이터를 (데이터 수=N, 채널 수=C, 높이=H, 너비=W)순으로 저장함



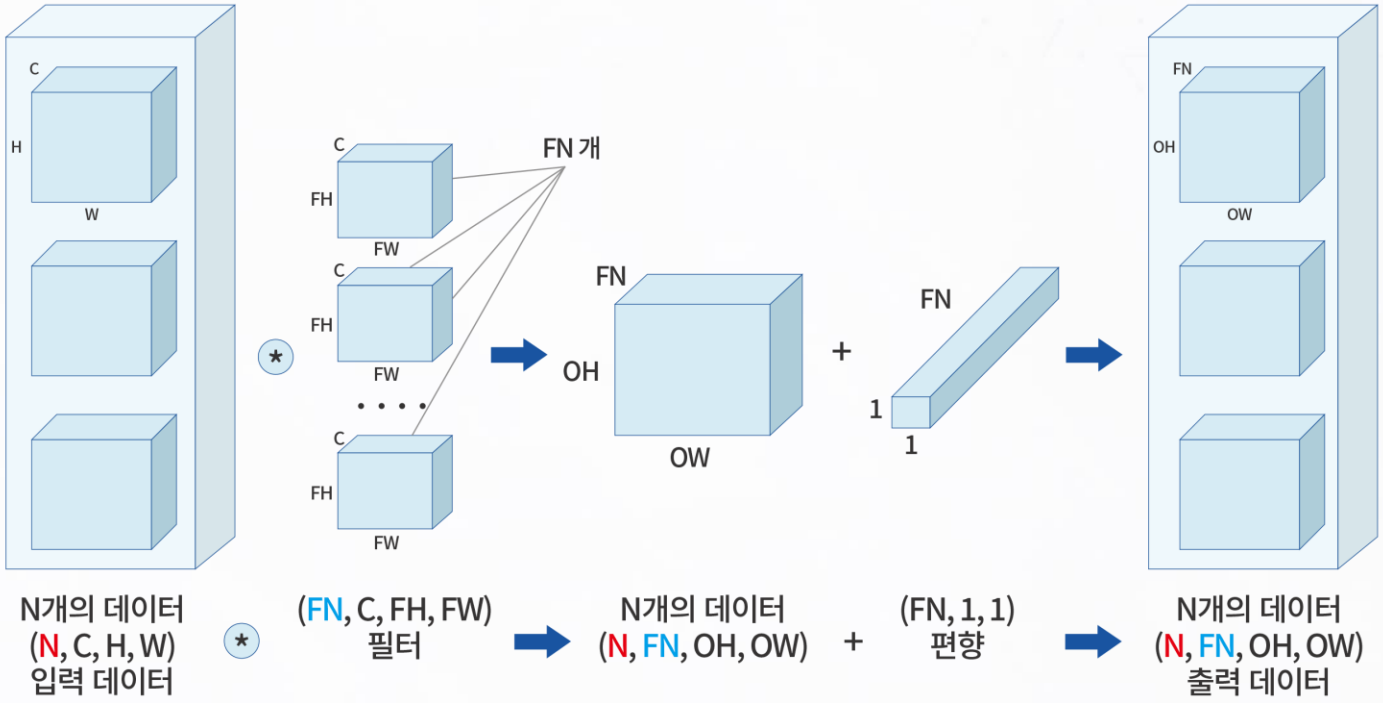
N개의 입력 데이터
(N, C, H, W)

05 | 합성곱 계층에서 배치 처리

데이터가 N개일 때 배치 처리한다면 데이터 형태가 아래 그림처럼 됨

✦ 아래의 그림처럼 배치 처리 시의 데이터 흐름을 보면, 각 데이터의 선두에 배치용 차원을 추가함

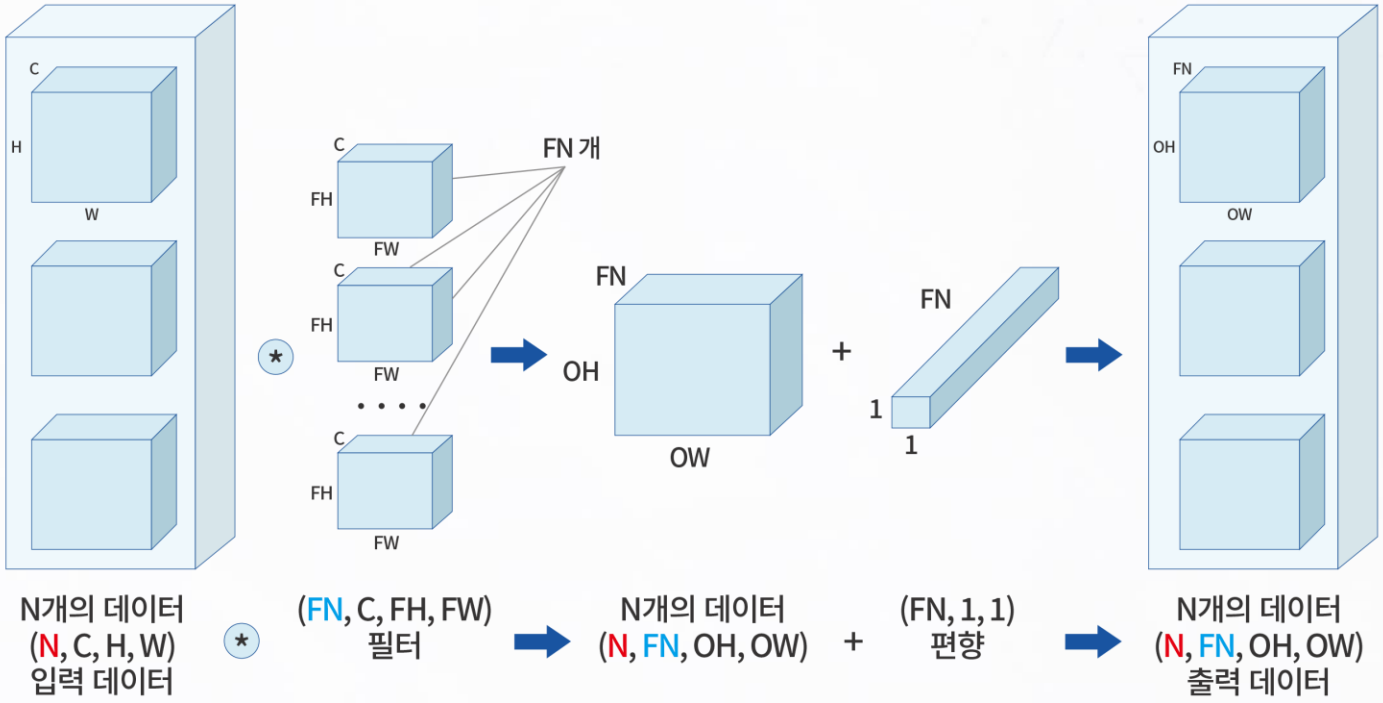
› 데이터는 4차원 형상을 가진 채 각 계층을 타고 흐름



합성곱 연산의 처리 흐름 (배치 처리)

05 | 합성곱 계층에서 배치 처리

- 여기에서 **주의할 점**으로는 **신경망**에 **4차원 데이터**가 **하나 흐를 때**마다 **데이터 N개**에 대한 **합성곱 연산**이 이뤄진다는 것임
  - 즉, **N회 분의 처리**를 **한 번**에 **수행하는 것**임



합성곱 연산의 처리 흐름 (배치 처리)