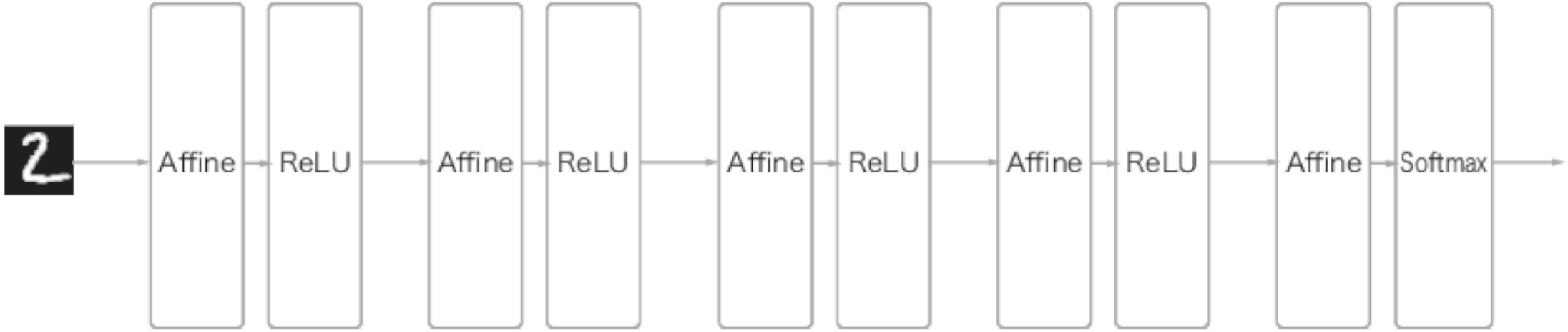
합성곱 신경망
(Convolutional Neural Network, CNN)

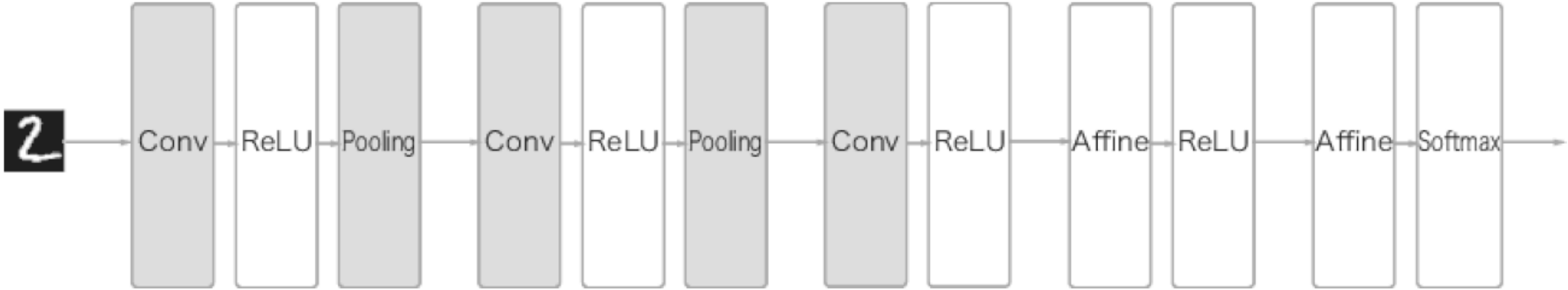
Chihuahua or Muffin?


@teenybiscuit

CNN 의 네트워크 구조



- 완전연결 계층 (Affine 계층) 으로 이뤄진 네트워크의 예



- CNN 으로 이뤄진 네트워크의 예 (합성곱 / 풀링 계층 추가)

합성곱 계층

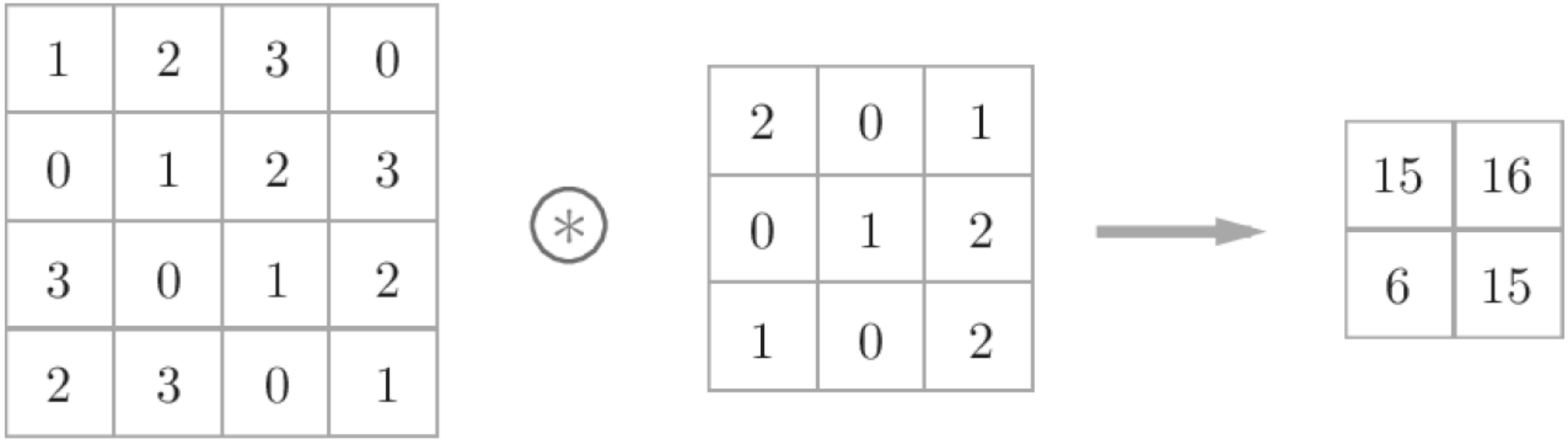
- 완전 연결 계층의 문제점.
  - 데이터의 형상이 무시.
  - 입력 데이터가 이미지인 경우, 이미지는 3차원(가로, 세로, 채널(색상))으로 구성된 데이터이나 1차원으로 평탄화 해줘야 함.
  - MNIST 데이터셋(1채널, 가로: 28픽셀, 세로: 28픽셀).
  - 형상을 무시하고 모든 입력 데이터를 동등한 뉴런(같은 차원의 뉴런)으로 취급하여 형상에 담긴 정보를 살릴 수 없음.

합성곱 계층

- 합성곱 계층의 특징
  - 입력 데이터의 형상을 유지.
  - 이미지도 3차원 데이터로 입력 받으며, 다음 계층에도 3차원 데이터로 전달.
  - 형상을 가진 데이터를 제대로 이해할 가능성이 큼.

- 특징 맵(feature map) : CNN에서 합성곱 계층의 입출력 데이터.

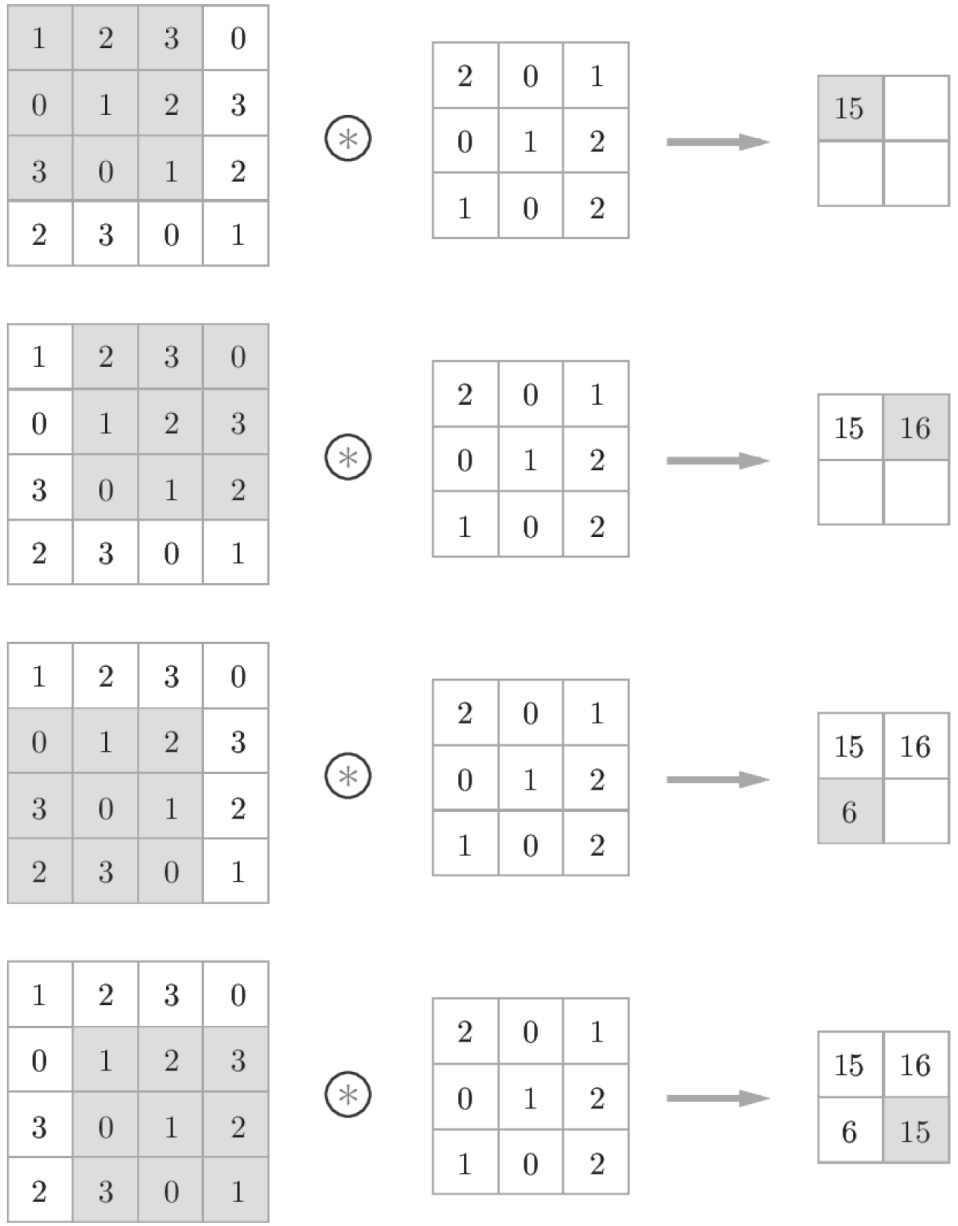
합성곱 연산 – 입력 데이터에 필터를 적용



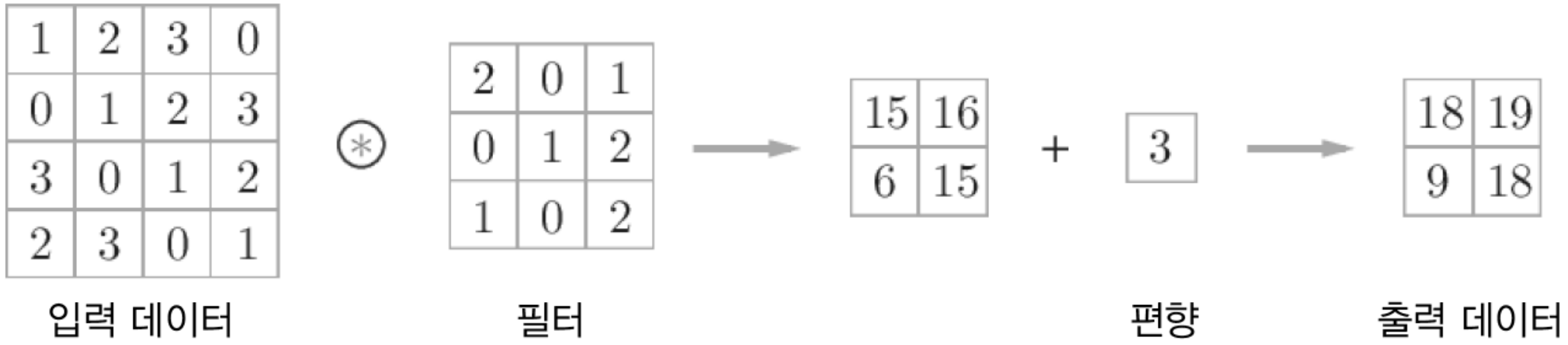
- 합성곱 연산을 * 기호로 표기.
- 이미지 처리에서 말하는 필터 연산에 해당.
- 데이터와 필터의 형상을 (높이, 너비)로 표기.
  - 위 예의 경우 입력은 (4, 4), 필터는 (3, 3), 출력은 (2, 2)가 됨.
- 문헌에 따라 필터를 커널이라 칭하기도 함.

단일 곱셈 – 누산(Fused Multiply-Add, FMA)

- 합성곱 연산의 계산 순서



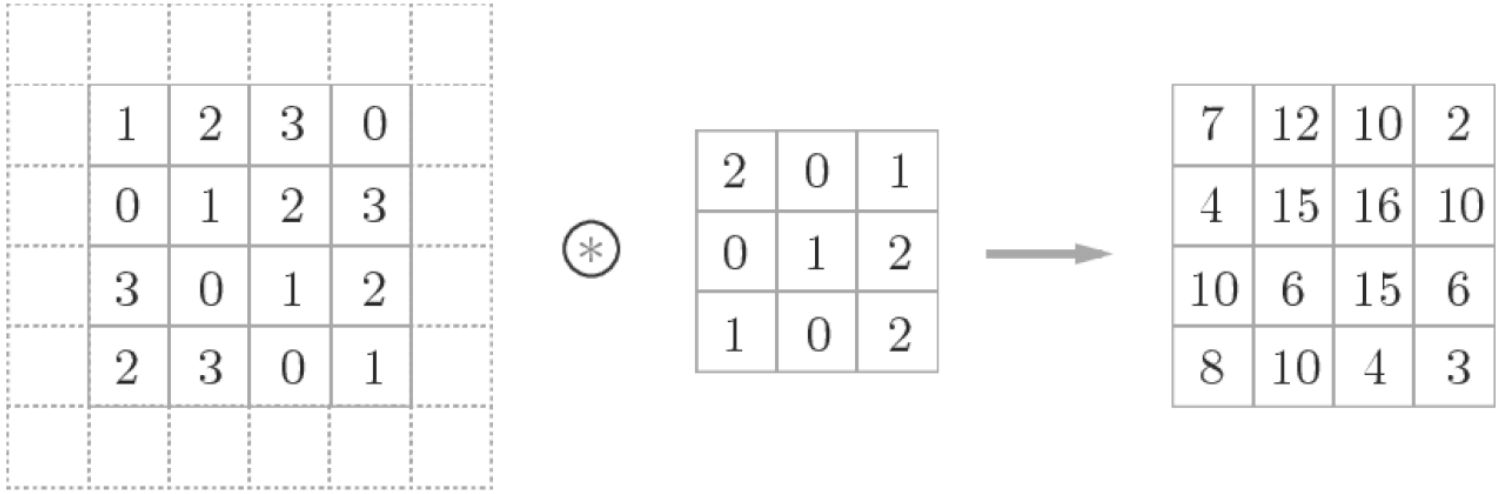
합성곱 연산의 편향



- 필터를 적용한 후 데이터에 더해 짐.
- 편향은 항상 (1 x 1)만 존재.

패딩(padding)

- 합성곱 연산을 수행하기 전에 입력 데이터 주변을 특정 값(예컨대 0)으로 채우는 것.
- 합성곱 연산에서 자주 이용하는 기법 – 출력 크기를 조정할 목적으로 사용.



(4, 4)
입력 데이터(패딩 : 1)

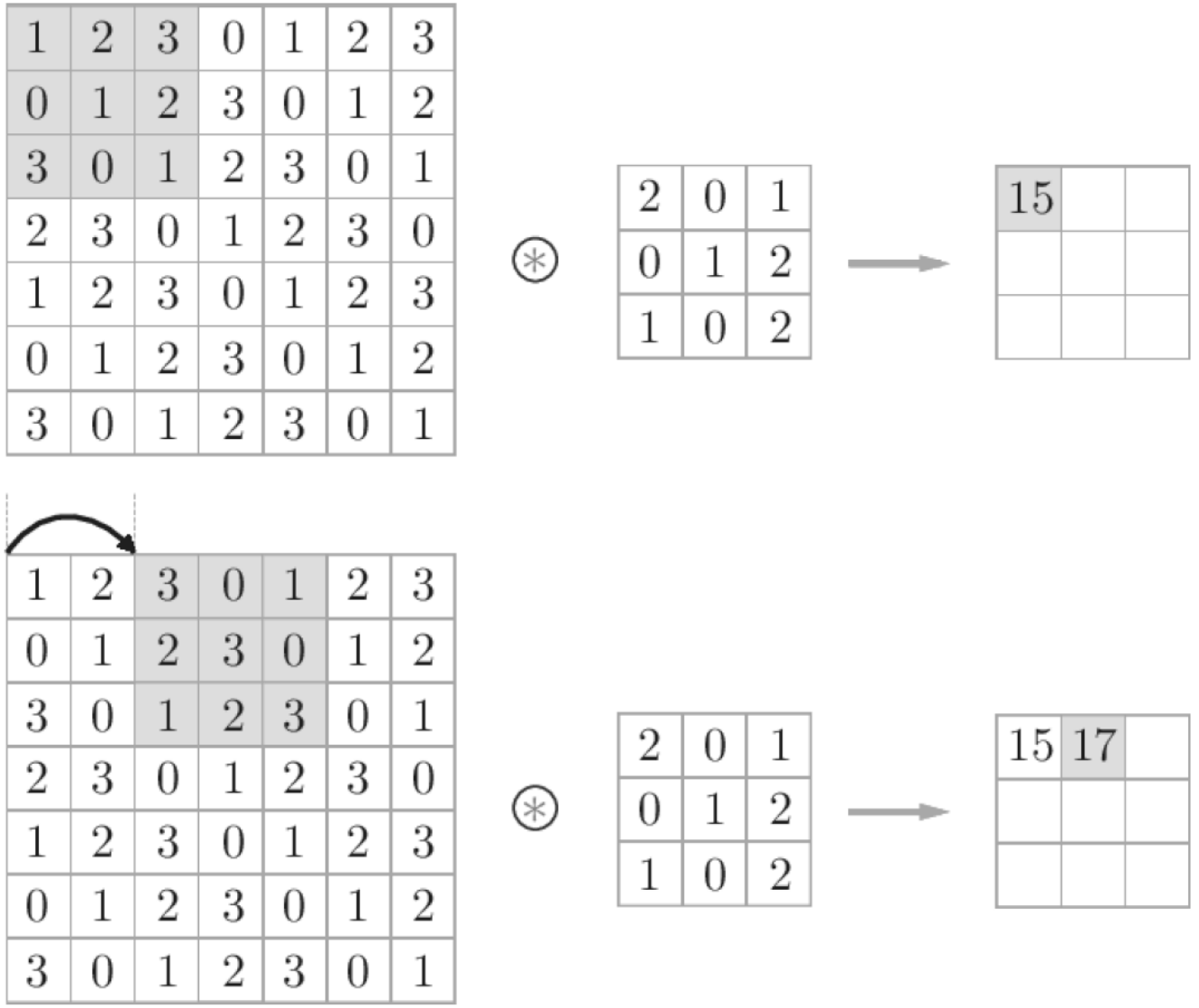
(3, 3)
필터

(4, 4)
출력 데이터

- 입력 데이터 주위에 0을 채운다 (0 생략함).

- 처음에 크기가 (4, 4)인 입력 데이터에 패딩이 추가되어 (6, 6)이 됨.
- (3, 3) 크기의 필터를 걸면 (4, 4) 크기의 출력 데이터가 생성.

스트라이드



- 필터를 적용하는 위치의 간격.
- 스트라이드를 키우면 출력 크기는 작아짐.

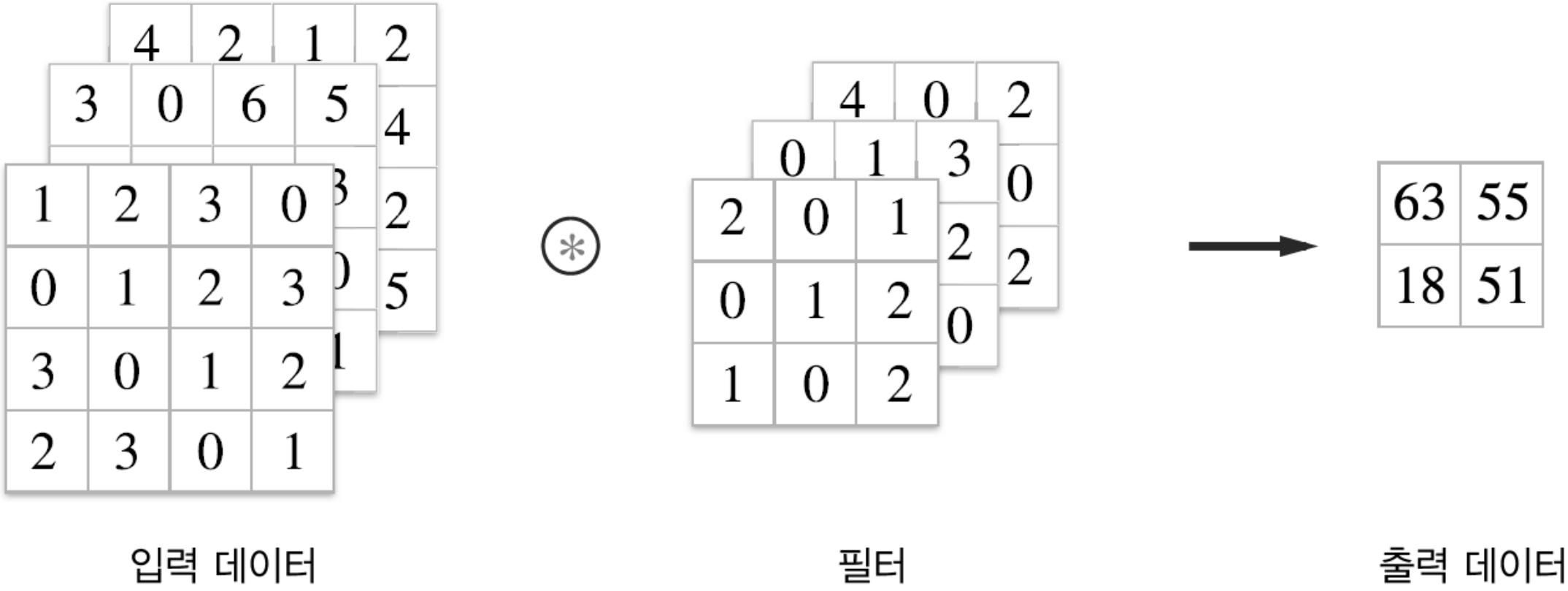
패딩, 스트라이드, 출력 크기 계산

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

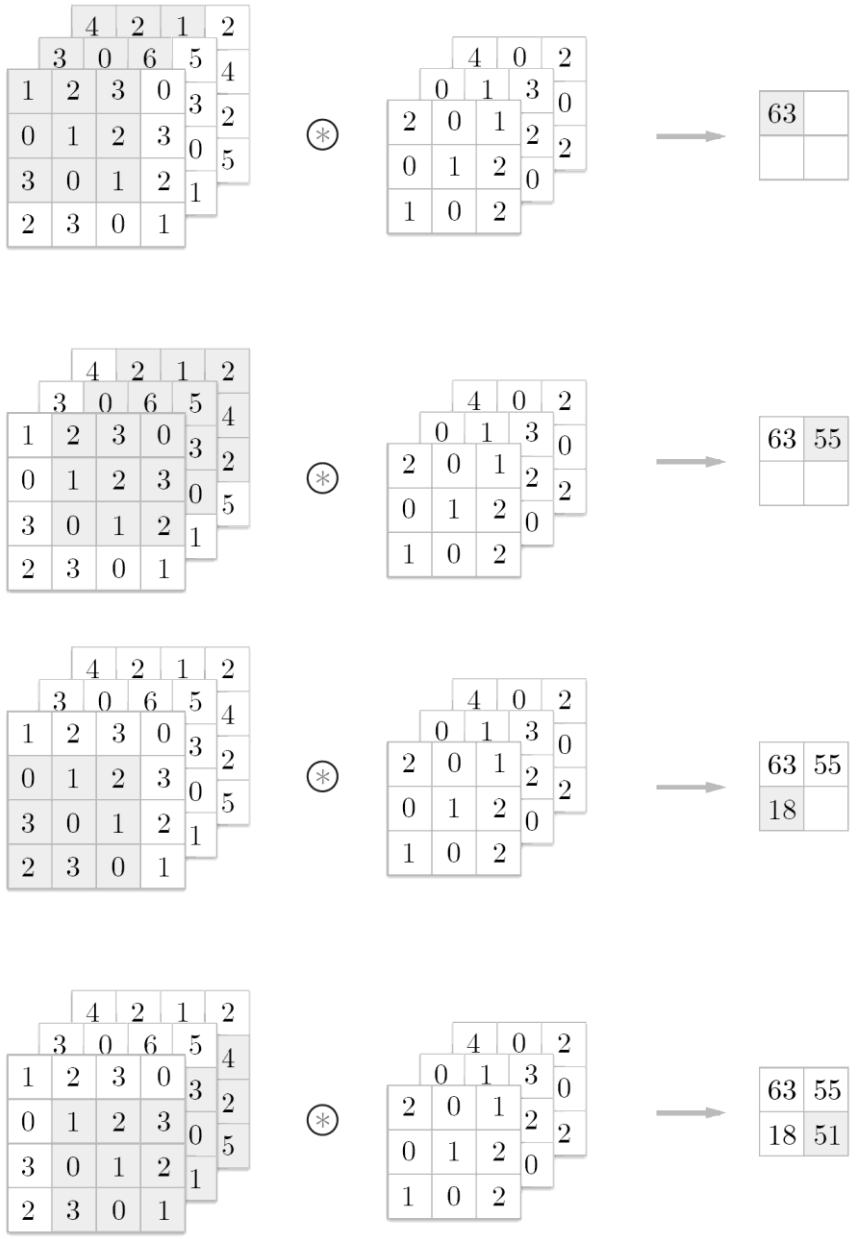
$$OW = \frac{W + 2P - FW}{S} + 1$$

- 입력 크기(H, W)
- 필터 크기(FH, FW)
- 출력 크기(OH, OW)
- 패딩 : P
- 스트라이드 : S

3차원 데이터의 합성곱 연산



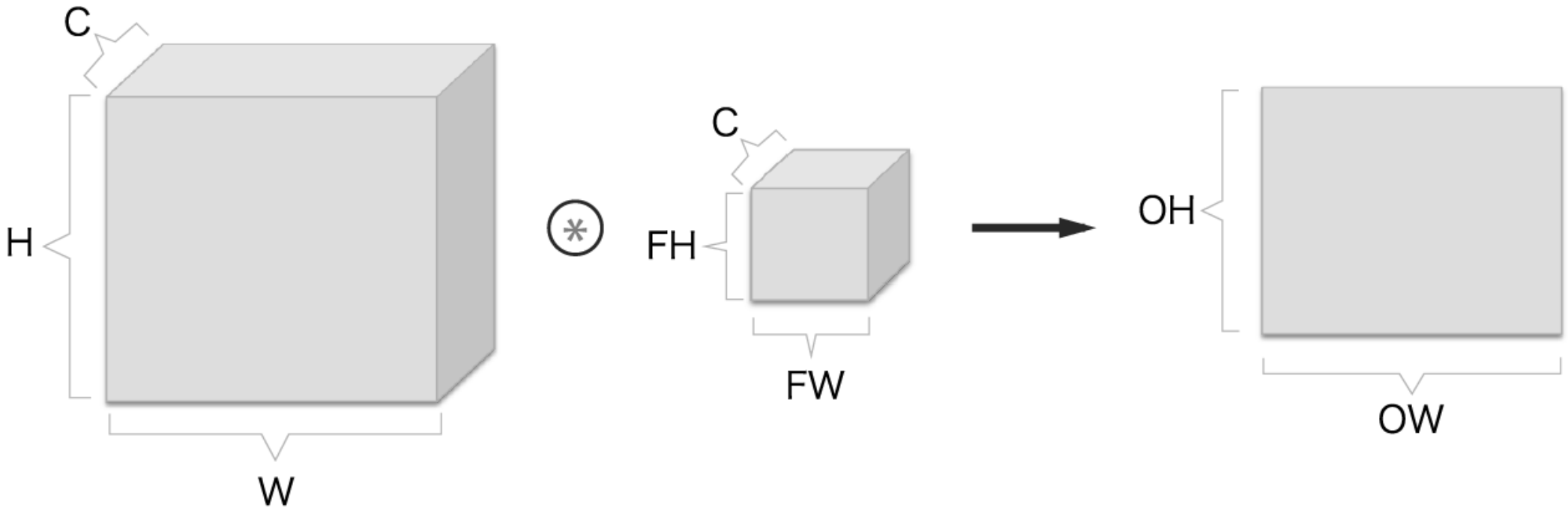
3차원 데이터 합성곱 연산의 계산 순서



- 3차원의 합성곱 연산에서 주의할 점.
  - 입력 데이터의 채널 수와 필터의 채널 수가 같아야 함.

- 필터 자체의 크기는 원하는 값으로 설정 가능(단, 모든 채널의 필터가 같은 크기여야 함).

3차원 합성곱 연산 – 블록으로 생각하기

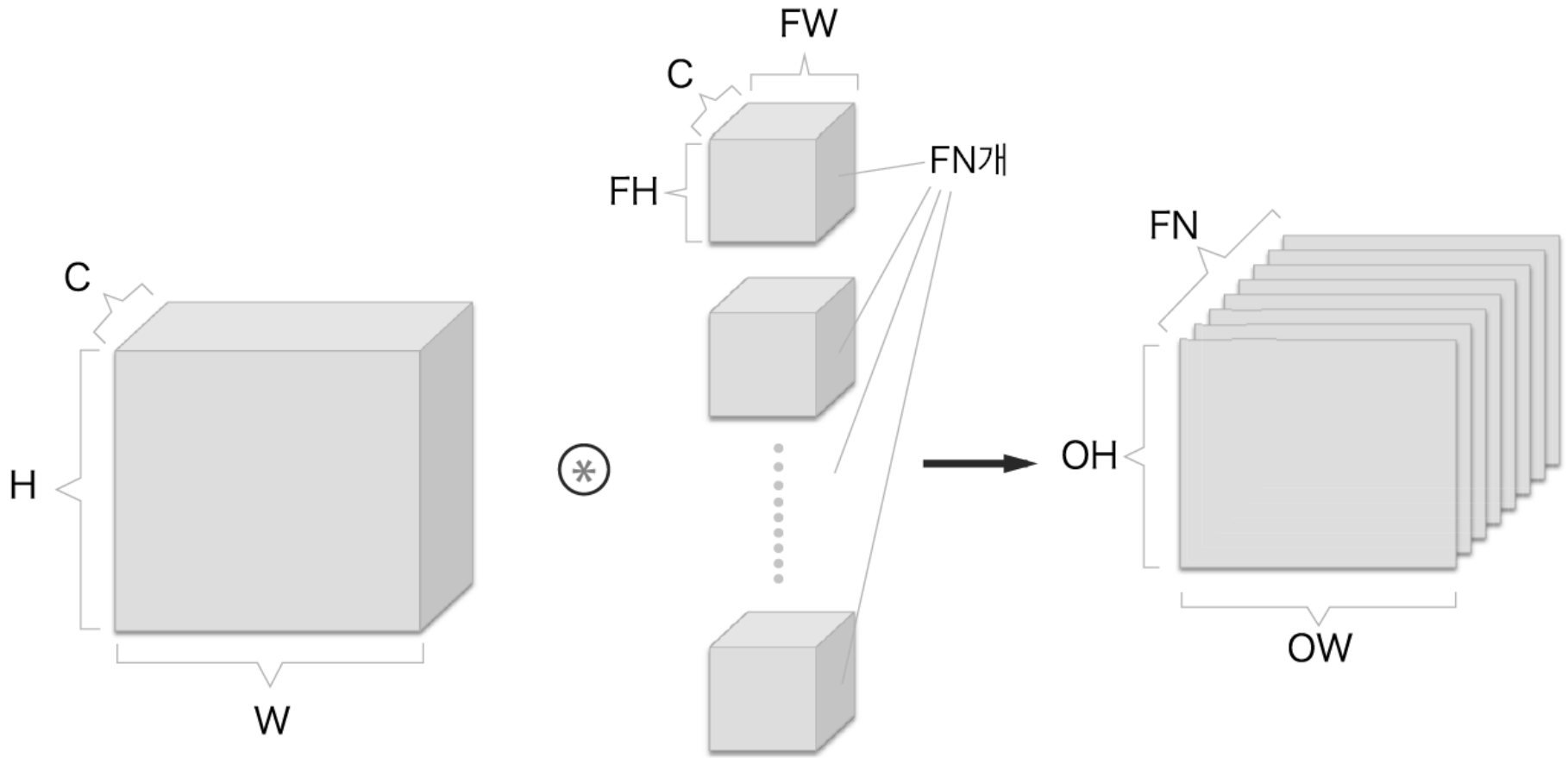


(C, H, W)
입력 데이터

(C, FH, FW)
필터

(1, OH, OW)
출력 데이터

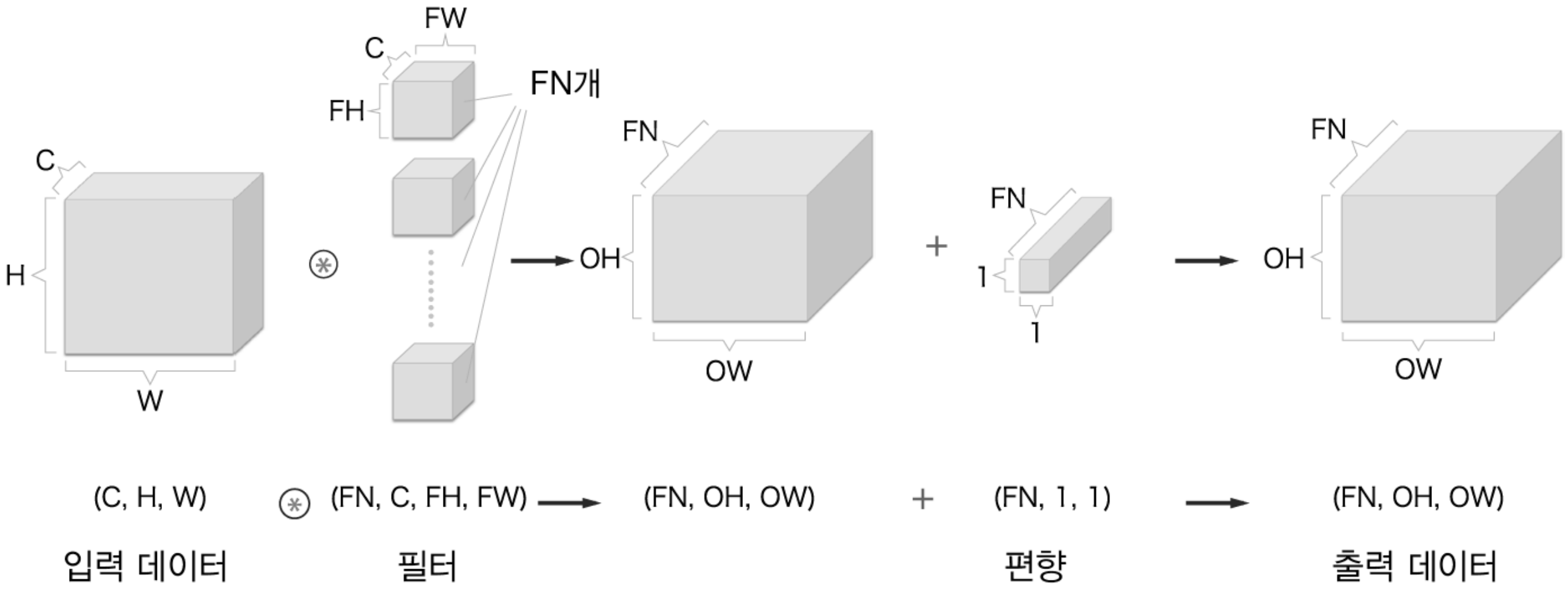
여러 필터를 사용한 합성곱 연산의 예



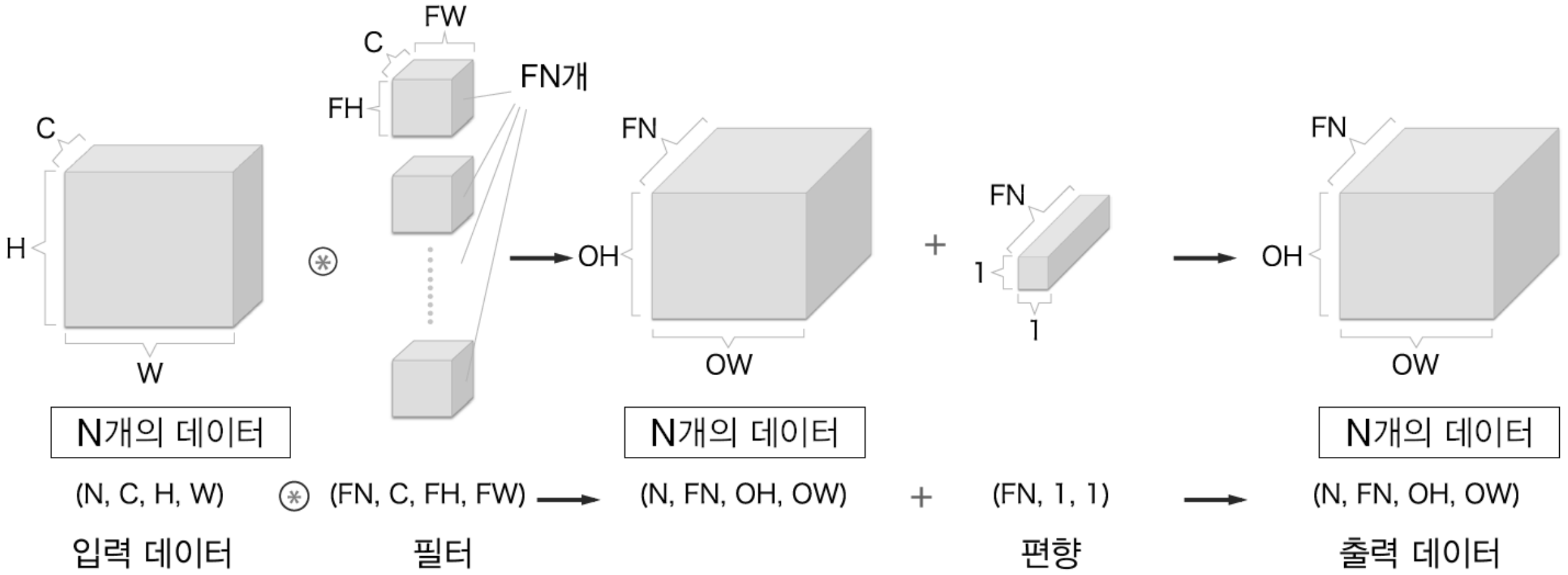
(C, H, W) ⊛ (FN, C, FH, FW) → (FN, OH, OW)

입력 데이터 필터 출력 데이터

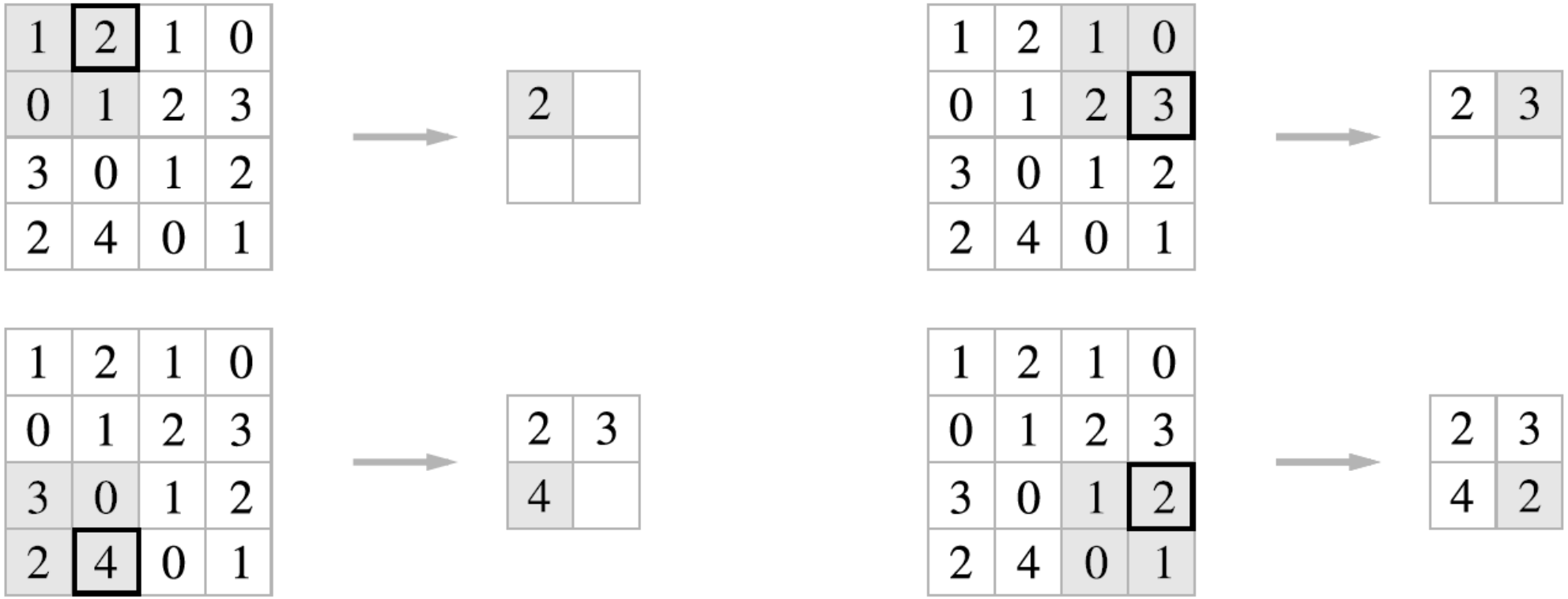
합성곱 연산의 처리 흐름(편향 추가)



합성곱 연산의 처리 흐름(배치 처리)



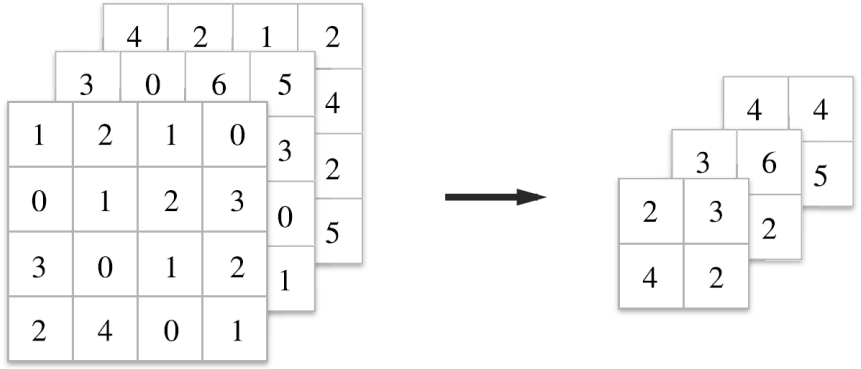
풀링 계층



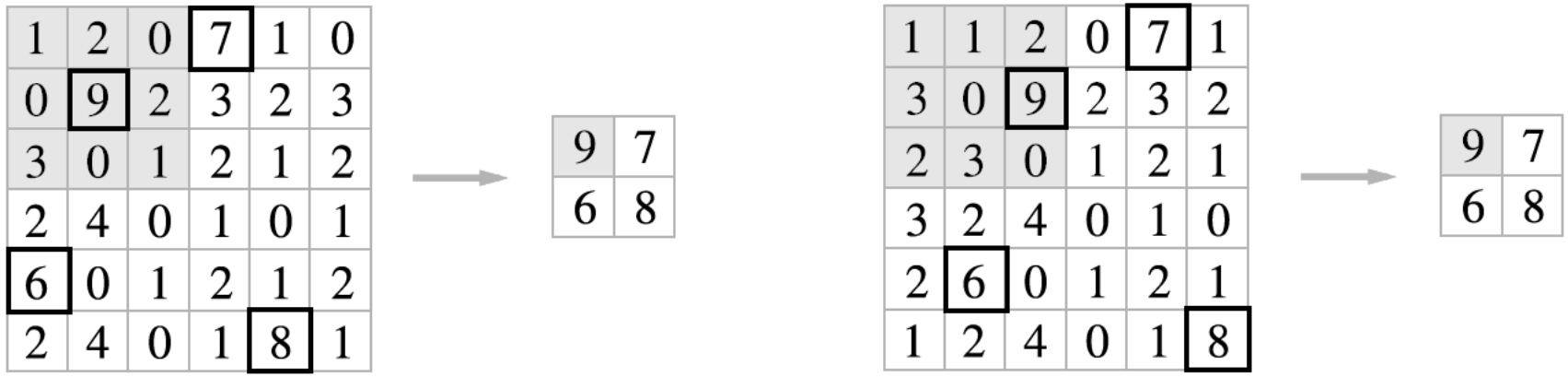
- 세로・가로 방향의 공간을 줄이는 연산.
- 위의 예 경우 2 x 2 최대 풀링(max pooling)을 스트라이드 2로 처리하는 순서.
- 풀링의 윈도우 크기와 스트라이드는 같은 값으로 설정하는 것이 일반적.

풀링 계층의 특징

- 학습해야 할 매개변수가 없다.
- 채널 수가 변하지 않는다.

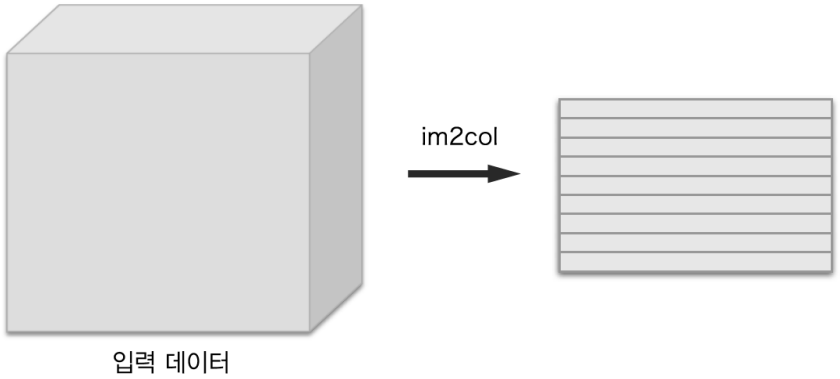


- 입력의 변화에 영향을 적게 받는다(강건하다).

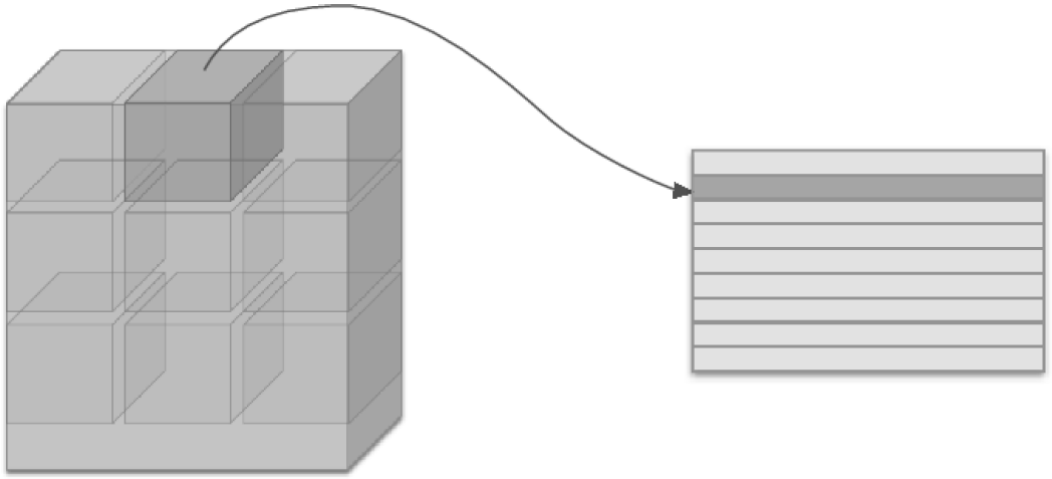


합성곱 계층 구현하기

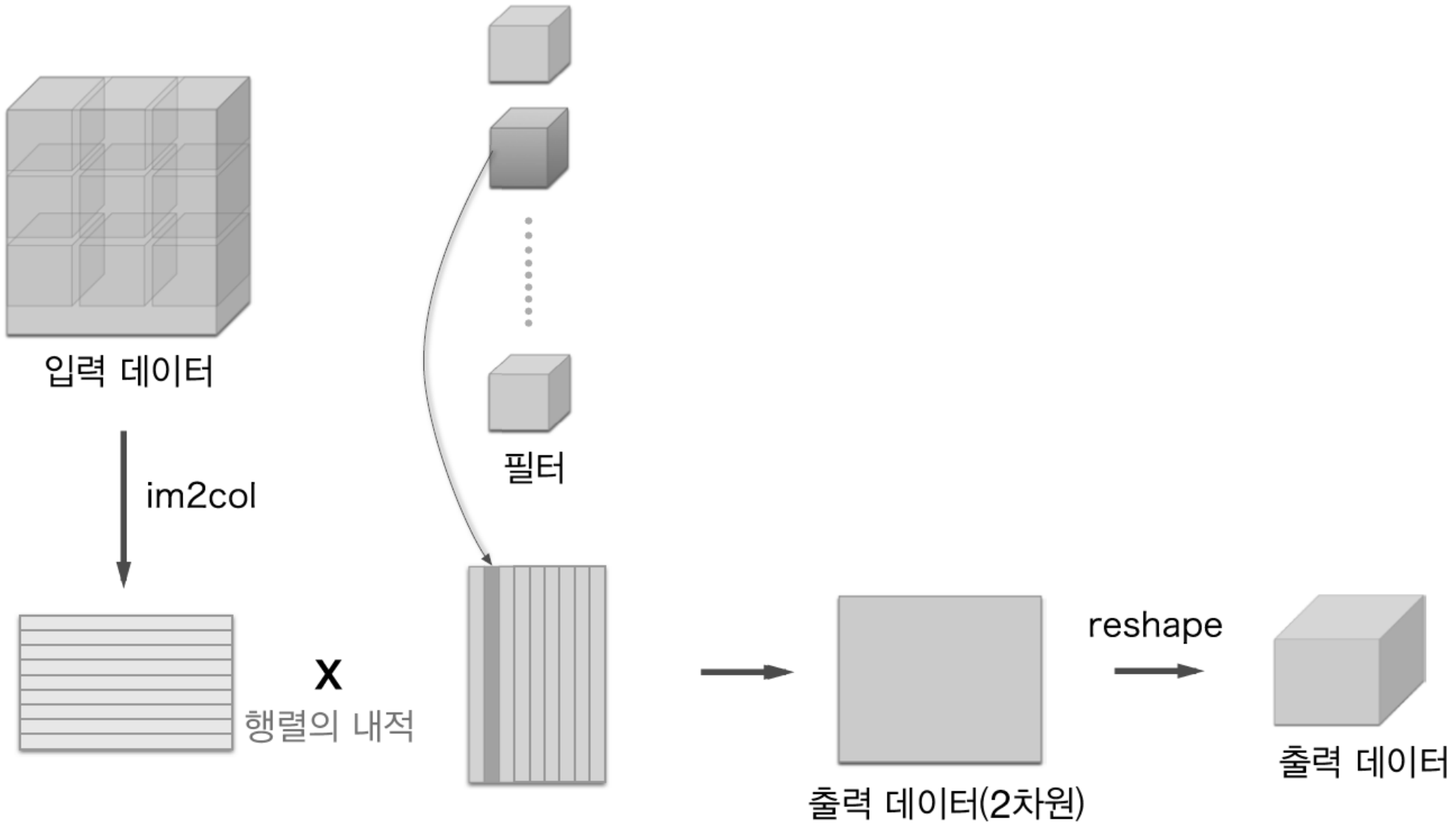
- 4차원 배열 : CNN에서 계층 사이를 흐르는 데이터는 4차원.
- Im2col로 데이터 전개하기 – 입력 데이터를 필터링(가중치 계산)하기 좋게 전개하는(펼치는) 함수.



- 입력 데이터에서 필터 적용 영역(3차원 블록)을 앞에서부터 순서대로 1줄로 펼친다.



합성곱 연산의 필터 처리 상세 과정



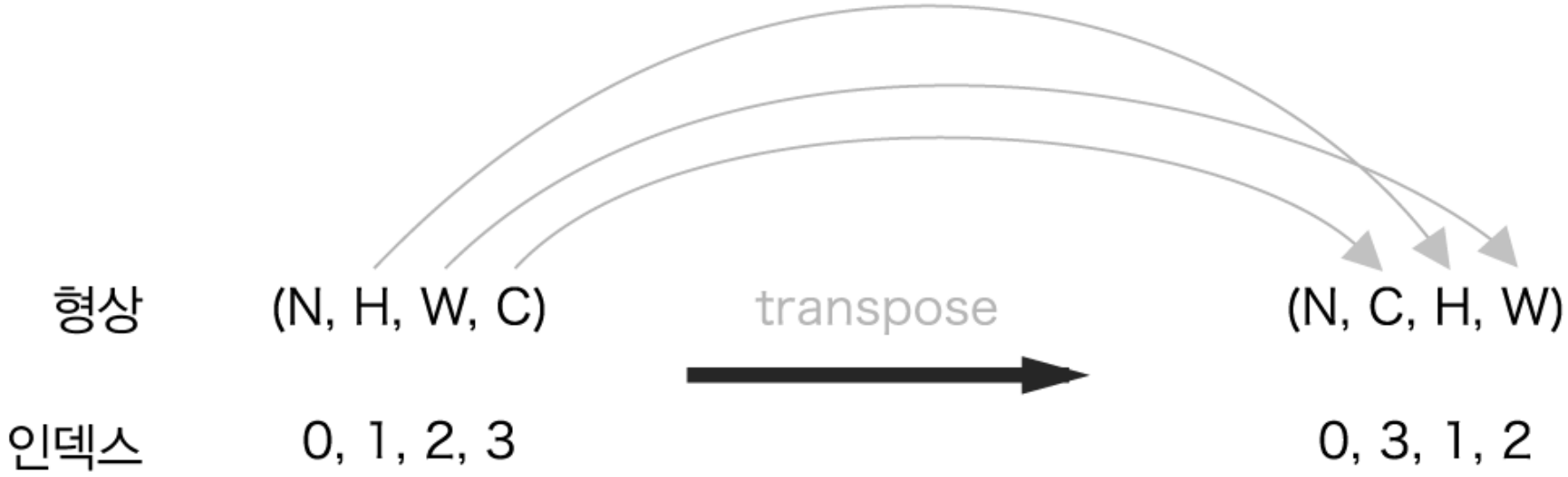
합성곱 계층 구현하기

- im2col(input_data, filter_h, filter_w, stride = 1, pad = 0)
  - input_data : (데이터 수, 채널 수, 높이, 너비)의 4차원 배열로 이뤄진 입력 데이터.
  - filter_h : 필터의 높이.
  - filter_w : 필터의 너비.
  - stride : 스트라이드.
  - pad : 패딩.

합성곱 계층 구현하기

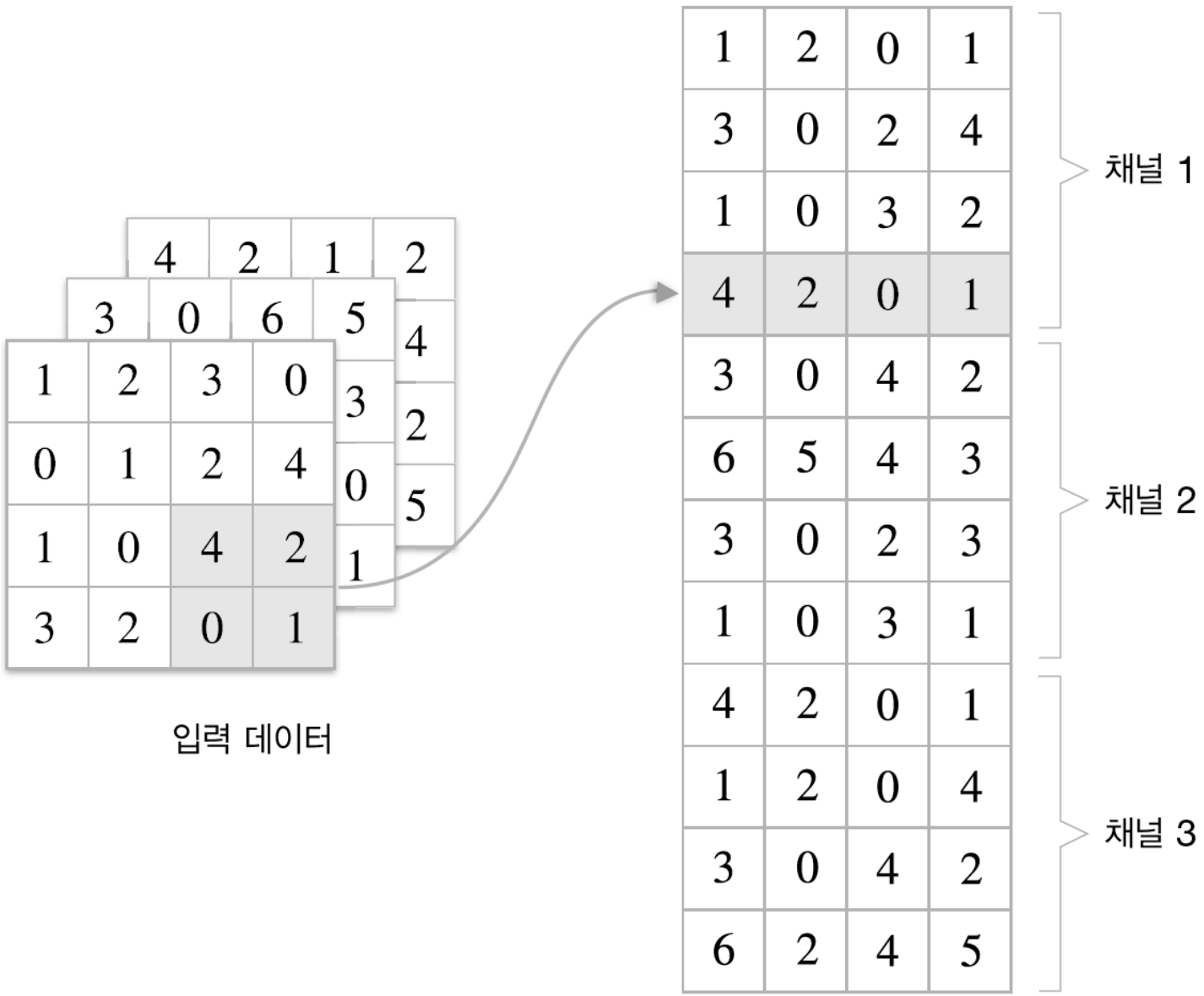
```
class Convolution:
    def __init__(self, W, b, stride=1, pad=0):
        self.W = W
        self.b = b
        self.stride = stride
        self.pad = pad

    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
```

넘파이의 transpose 함수로 축 순서 변경하기



- 인덱스(번호)로 축의 순서를 변경한다.
- 합성곱 계층의 역전파는 Affine 계층의 구현과 유사하다.

풀링 계층 구현하기



- 풀링 계층 구현도 합성곱 계층과 마찬가지로 im2col을 사용해 입력 데이터를 전개.
- 단, 풀링의 경우엔 채널쪽이 독립적이라는 점이 합성곱 계층 때와 다름.

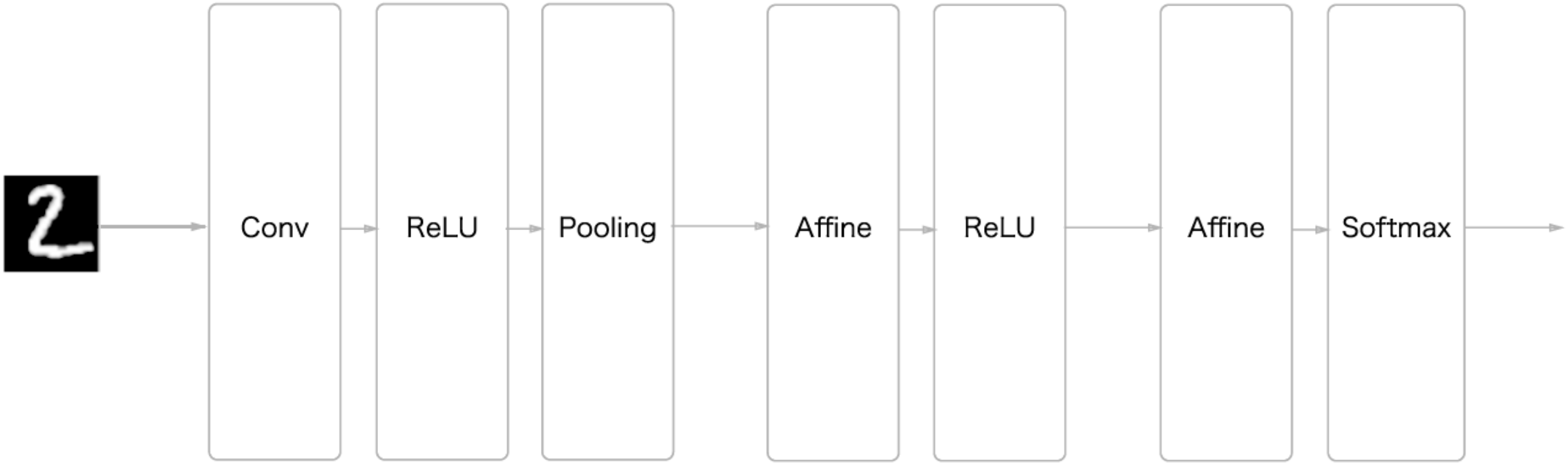
풀링 계층 구현의 흐름


4 2 1 2
3 0 6 5
1 2 3 0
0 1 2 4
1 0 4 2
3 2 0 1
4
3
2
5
1
입력 데이터
전개
1 2 0 1
3 0 2 4
1 0 3 2
4 2 0 1
3 0 4 2
6 5 4 3
3 0 2 3
1 0 3 1
4 2 0 1
1 2 0 4
3 0 4 2
6 2 4 5
최댓값
2
4
3
4
4
6
3
3
4
4
4
6
reshape
4 4
4 6 6
2 4 3
3 4
출력 데이터

풀링 계층의 forward 처리 흐름 구현 코드

```
class Pooling
    def __init__(self, pool_h, pool_w, stride=1, pad=0):
        self.pool_h = pool_h
        self.pool_w = pool_w
        self.stride = stride
        self.pad = pad

    def forward(self, x):
        N, C, H, W = x.shape
        out_h = int(1 + (H - self.pool_h) / self.stride)
        out_w = int(1 + (W - self.pool_w) / self.stride)

        col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
        col = col.reshape(-1, self.pool_h*self.pool_w)

        out = np.max(col, axis=1)
        out = out.reshape(N, out_h, out_w, C).transpose(0, 3, 1, 2)

        return out
```

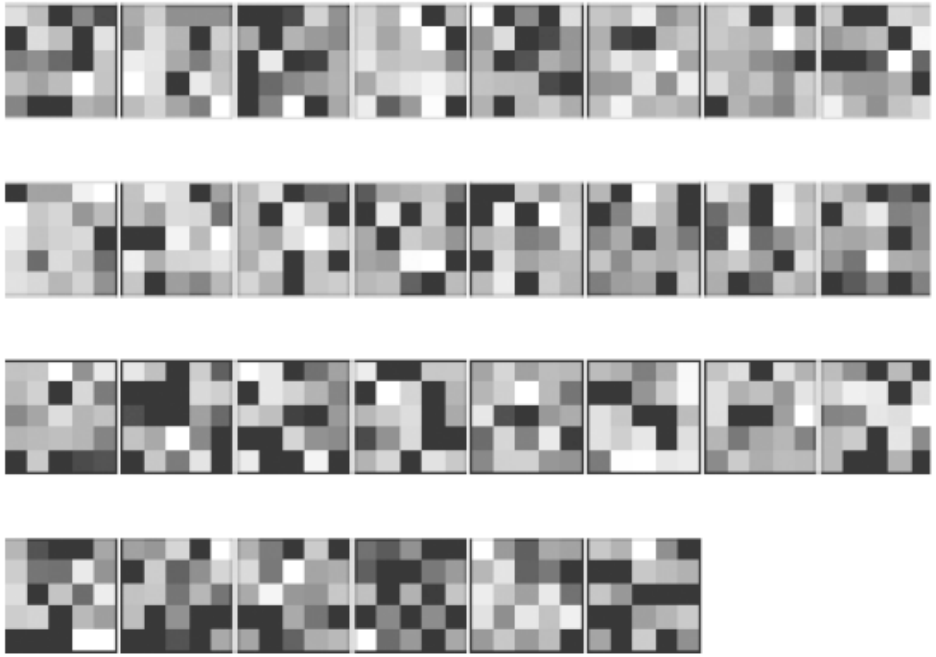
CNN 구현하기



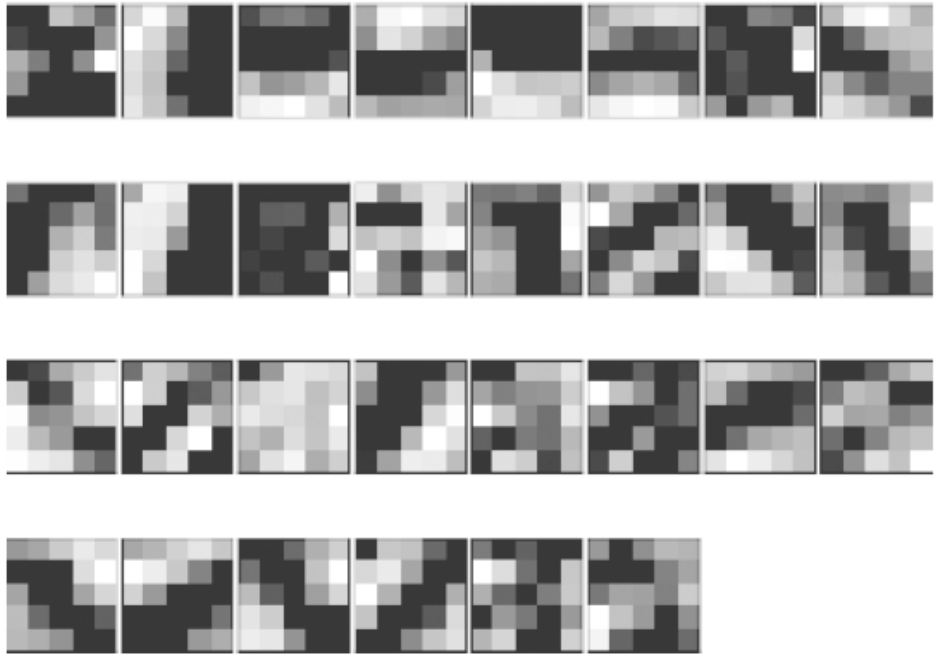
- 손글씨 숫자를 인식하는 CNN을 조립

CNN 시각화하기

학습 전

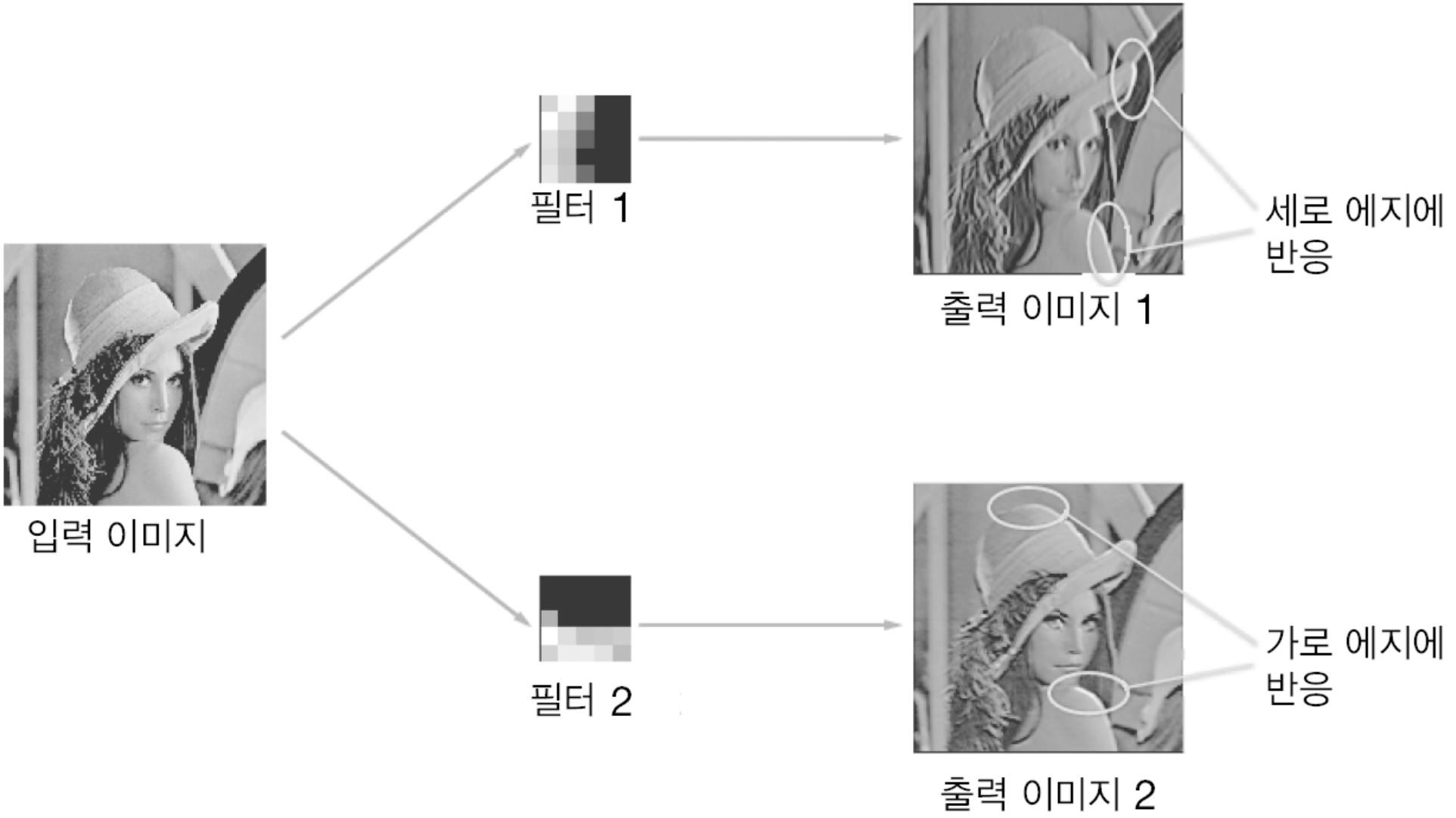


학습 후

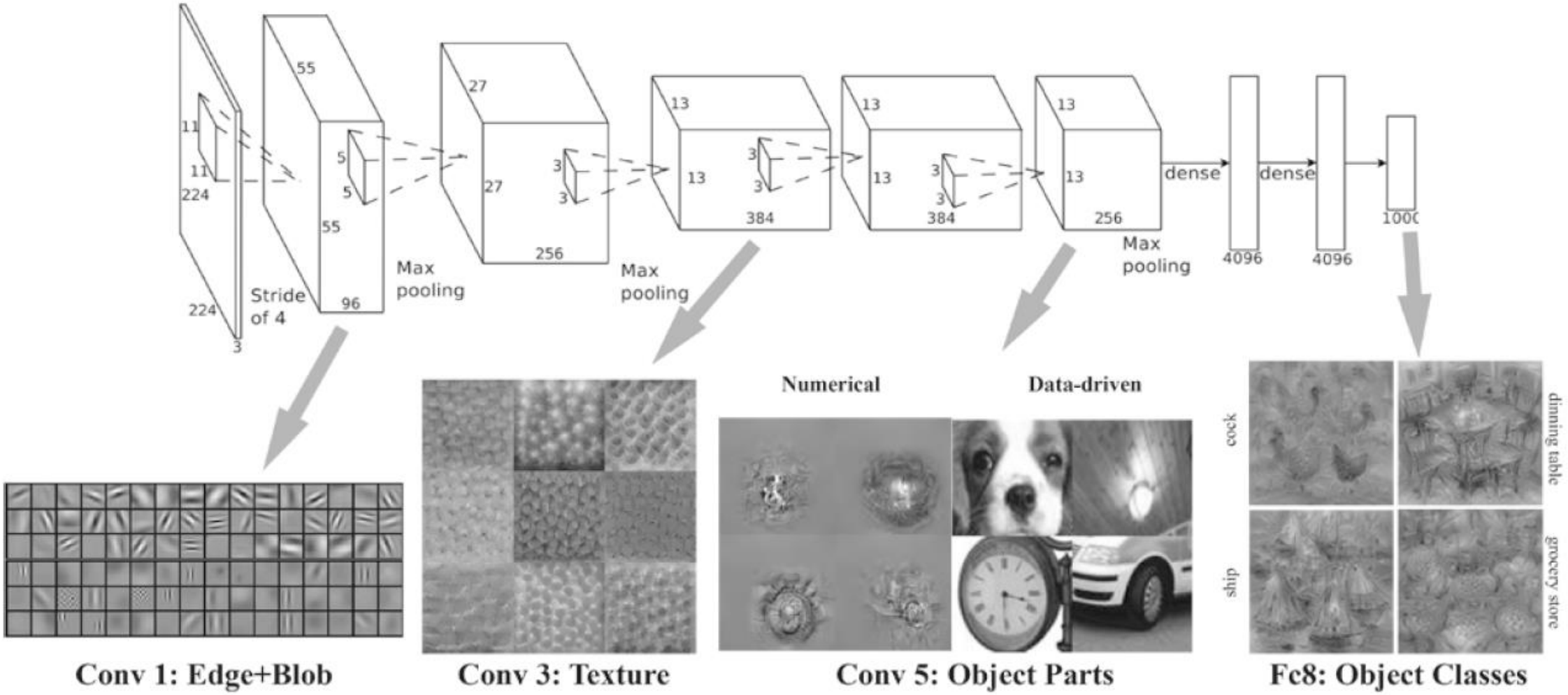


- 학습 전과 후의 1 번째 층의 합성곱 계층의 가중치 : 가중치의 원소는 실수이지만 , 이미지에서는 가장 작은값 (0) 은 검은색 , 가장 큰 값 (255) 은 흰색으로 정규화하여 표시함 .

가로 에지와 세로 에지에 반응하는 필터



층 깊이에 따른 추출 정보 변화

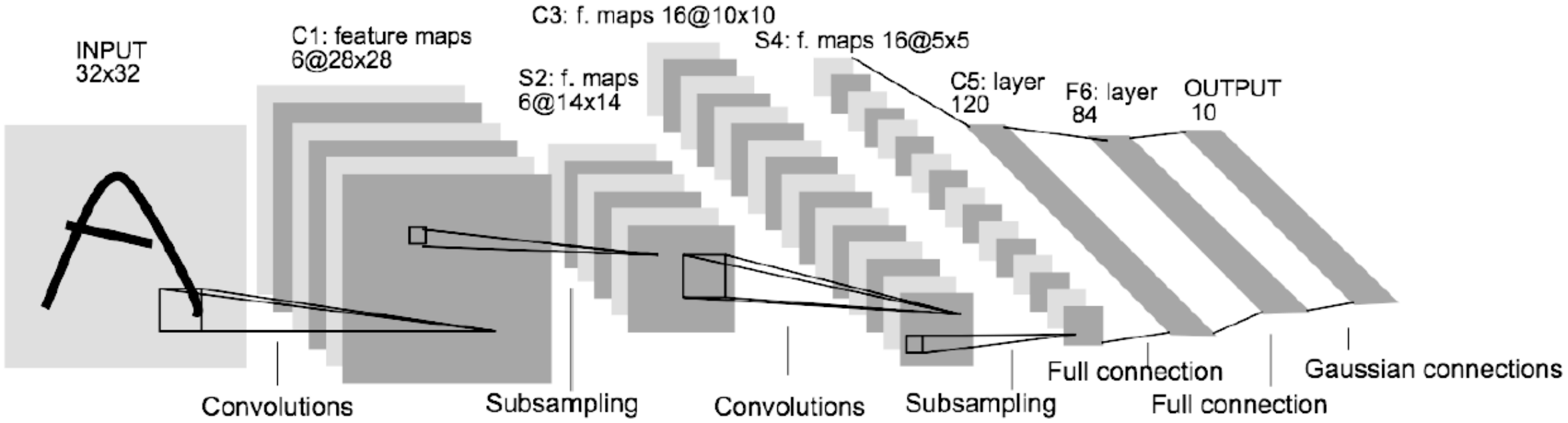


층 깊이에 따른 추출 정보 변화

- 합성곱 계층을 여러 겹 쌓으면, 층이 깊어지면서 더 복잡하고 추상화된 정보가 추출.
- 처음 층은 단순한 에지에 반응.
- 이어서 텍스처에 반응.
- 더 복잡한 사물의 일부에 반응하도록 변화.
- 즉, 층이 깊어지면서 뉴런이 반응하는 대상이 단순한 모양에서 '고급' 정보로 변화해 감 -> 사물의 '의미'를 이해하도록 변화하는 것.

대표적인 CNN - LeNet

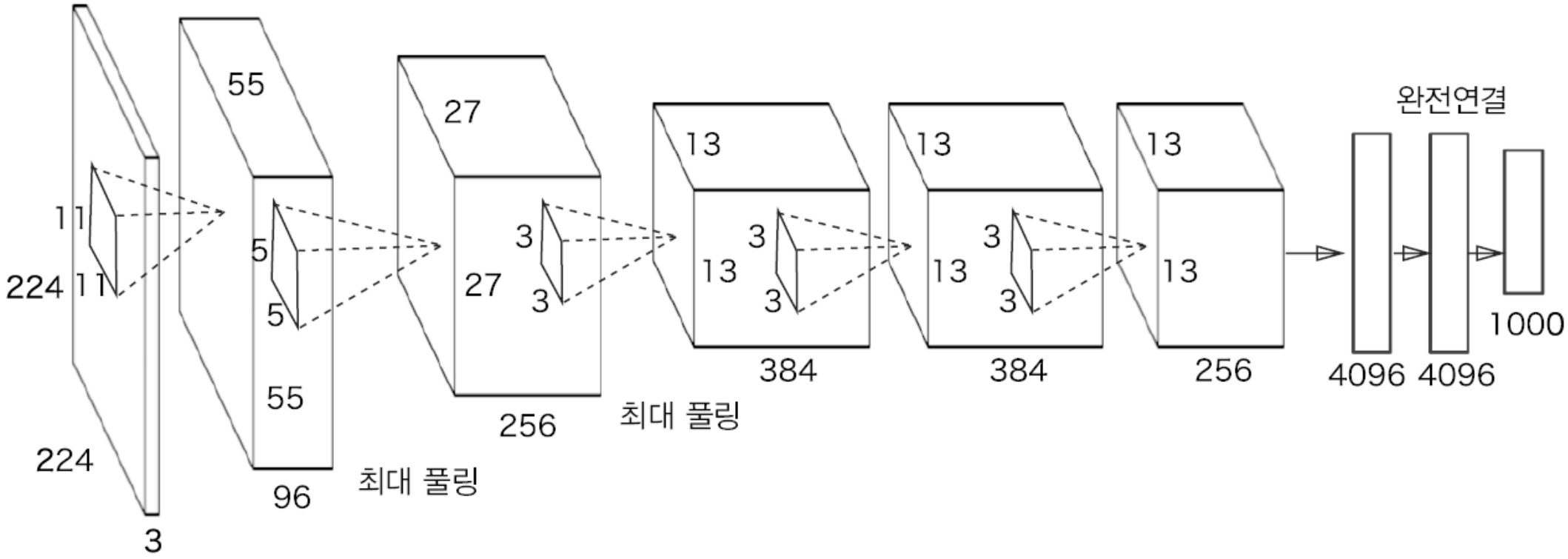
- 손글씨 숫자를 인식하는 네트워크로 1998년에 제안.
- 합성곱 계층과 풀링 계층(정확히는 단순히 '원소를 줄이기'만 하는 서브샘플링 계층)을 반복하고, 마지막으로 완전 계층을 거치면서 결과를 출력.



- 활성화 함수로 시그모이드 함수를 사용.
- 서브 샘플링을 하여 중간 데이터의 크기가 작아짐(현재는 최대 풀링이 주류).

대표적인 CNN - AlexNet

- 2012년에 발표된 AlexNet은 딥러닝 열풍을 일으키는 데 큰 역할.
- 구성



- 활성화 함수로 ReLU를 이용하고, 드롭아웃을 사용.
- LRN(Local Response Normalization)이라는 국소적 정규화를 실시하는 계층을 이용.