Convolutional Neural Network

Computational Linguistics @ Seoul National University

DL from Scratch

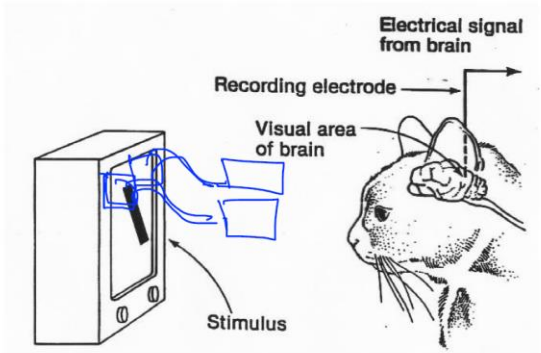
By Hyopil Shin

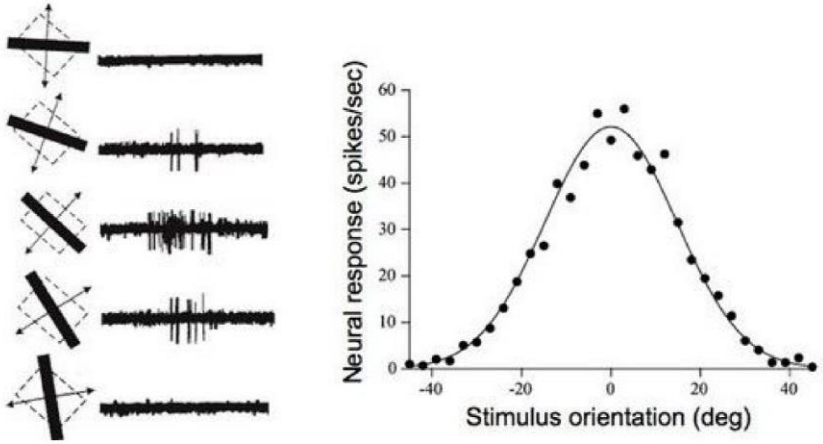
Convolutional Neural Network

History

A bit of history:

Hubel & Wiesel,
1959
RECEPTIVE FIELDS OF SINGLE NEURONES IN
THE CAT'S STRIATE CORTEX
1962
RECEPTIVE FIELDS, BINOCULAR INTERACTION
AND FUNCTIONAL ARCHITECTURE IN
THE CAT'S VISUAL CORTEX
1968...





CNN Structure

그림 7-1 완전연결 계층(Affine 계층)으로 이뤄진 네트워크의 예

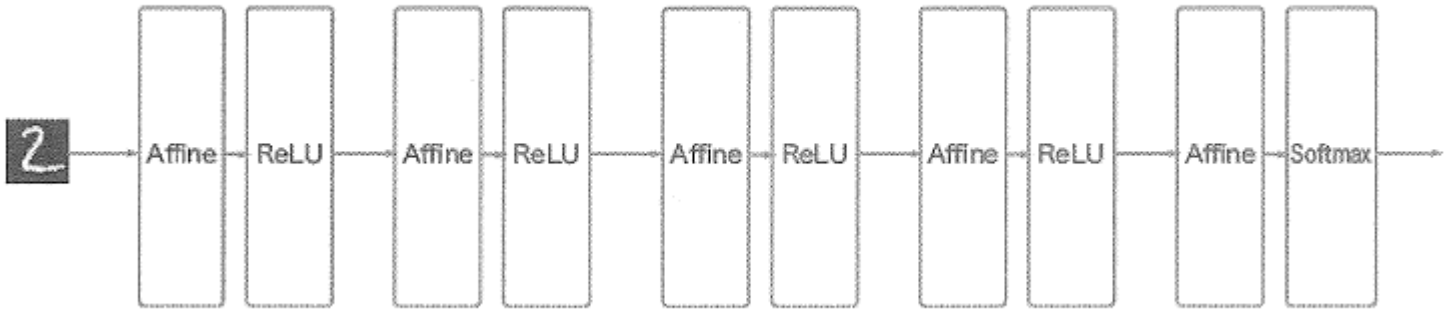
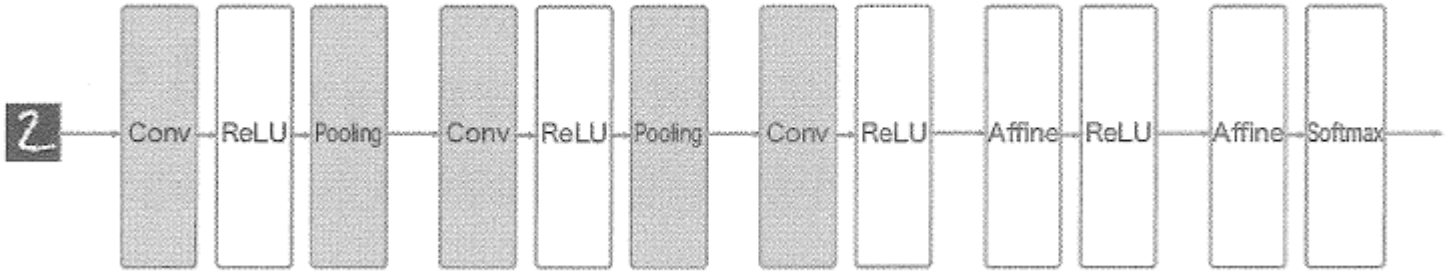
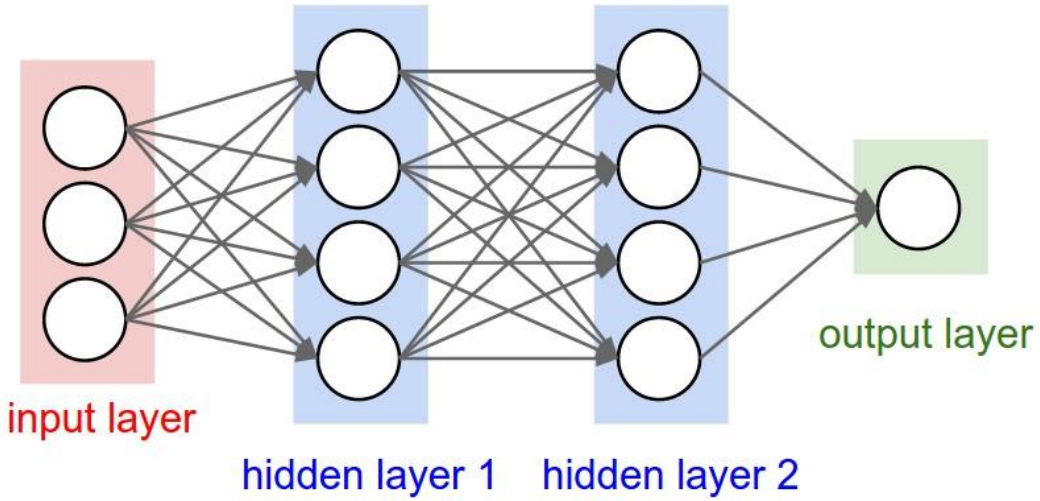
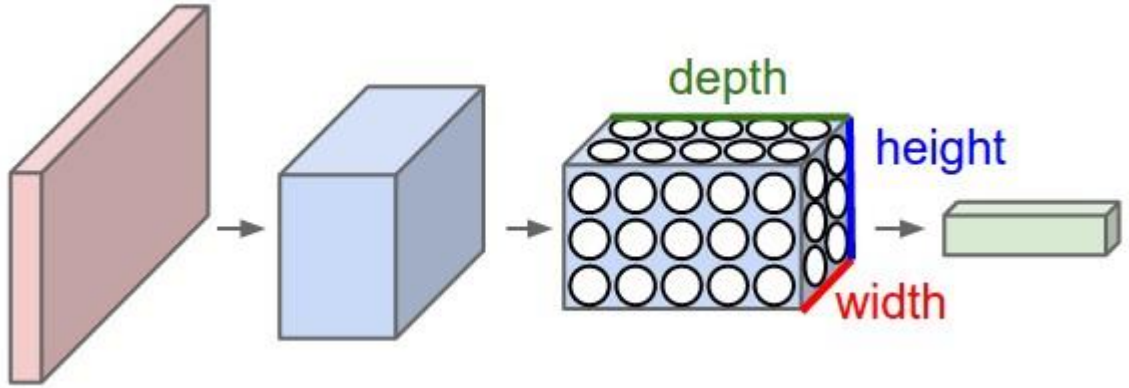


그림 7-2 CNN으로 이뤄진 네트워크의 예 : 합성곱 계층과 풀링 계층이 새로 추가(회색)

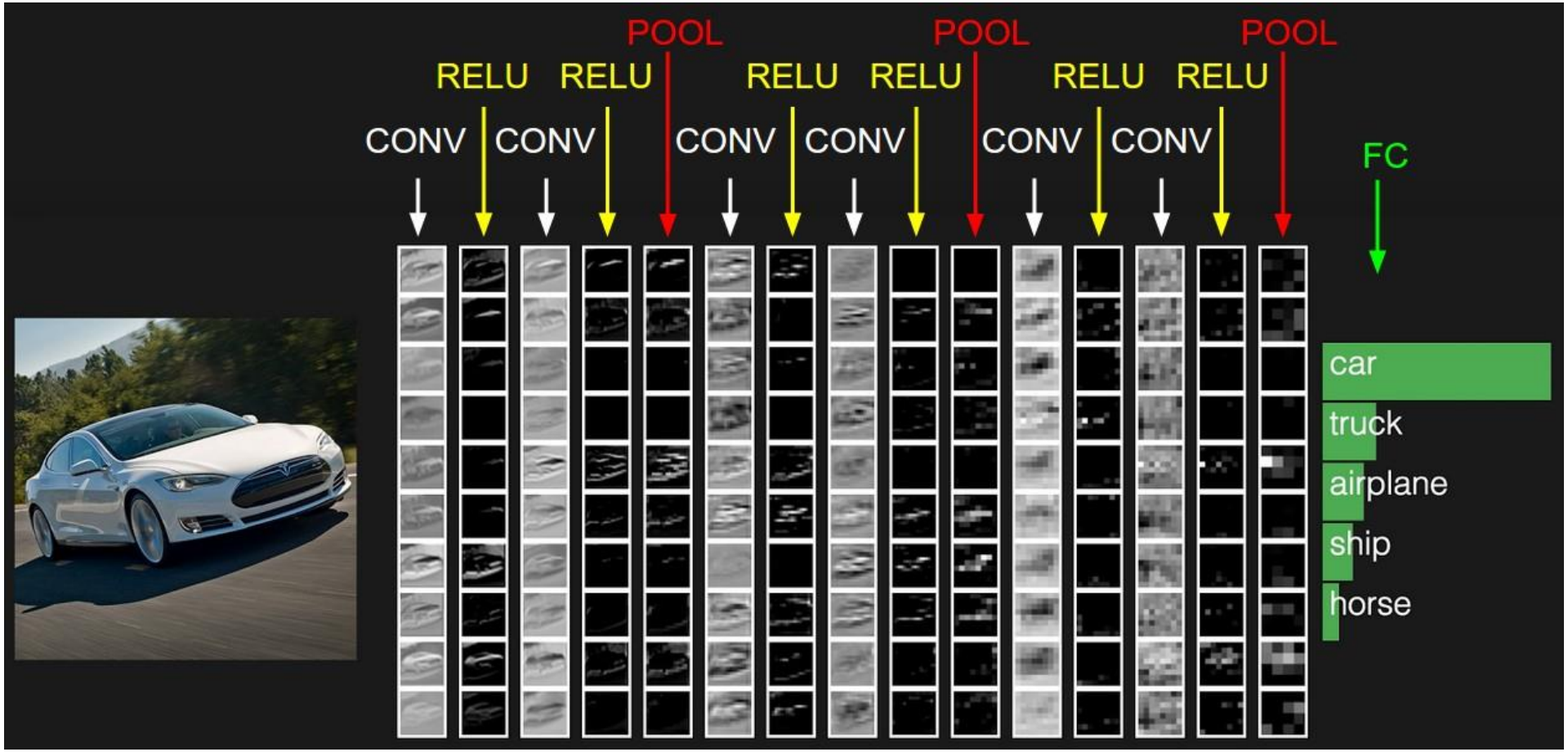


Convolutional Neural Network





Convolutional Neural Network

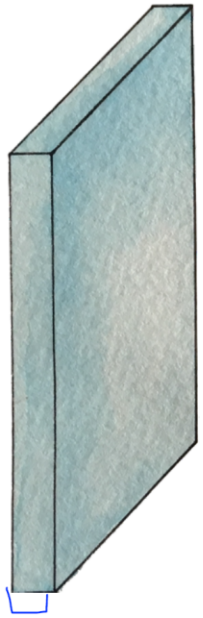


Affine Layers

- 문제점
  - 데이터의 형상이 무시됨
  - MNIST: 가로, 세로, 채널(색상)의 3D데이터 → (1채널, 세로 28픽셀, 가로 28픽셀)
  - 공간적 정보-공간적으로 가까운 픽셀은 값이 비슷하거나, RGB의 각 채널은 서로 밀접하게 관련, 거리가 먼 픽셀끼리는 별 연관이 없는 등
- 합성곱 계층은 형상을 유지할 수 있음
  - Feature map(특징맵): 합성곱의 입출력 데이터
    - Input feature map, output feature map

Convolution(from https://hunkim.github.io/ml/lec11.pdf)

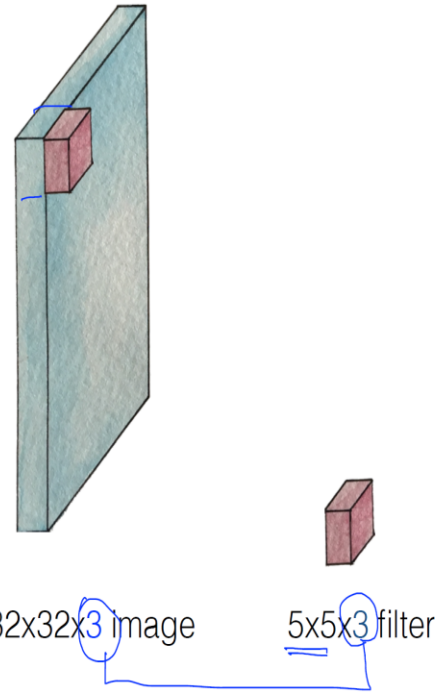
Start with an image (width x hight x depth)



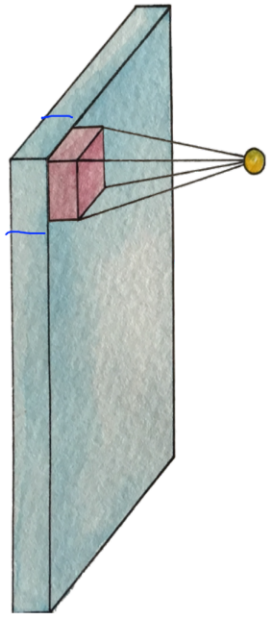
32x32x3 image

Convolution(from https://hunkim.github.io/ml/lec11.pdf)

Let's focus on a small area only (5x5x3)



Get one number using the filter



$\tilde{y} = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + \dots$

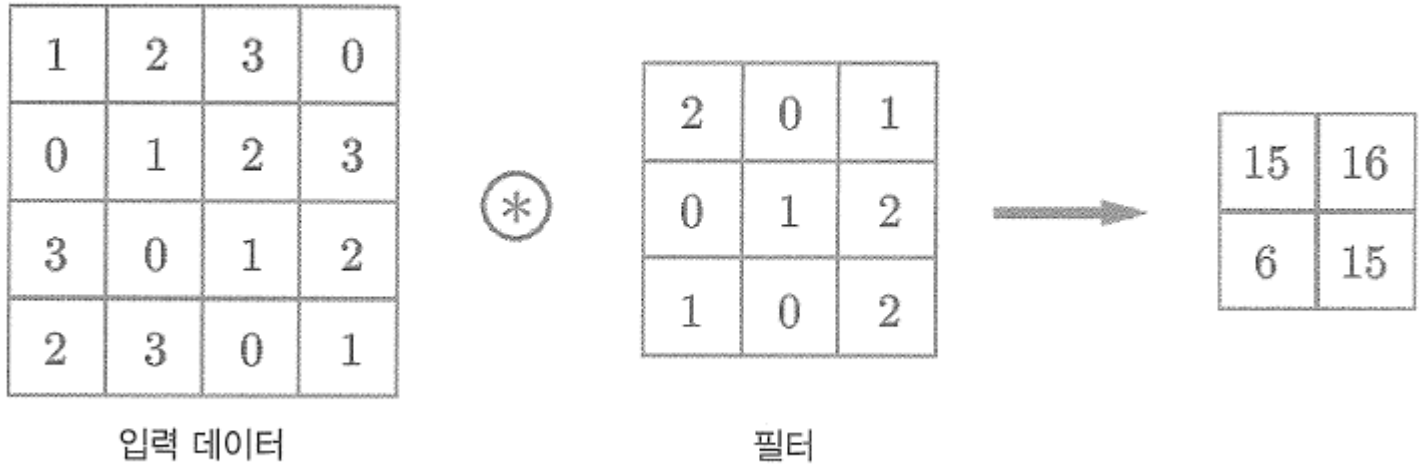
=Wx+b

=ReLU(Wx+b)

합성곱 연산

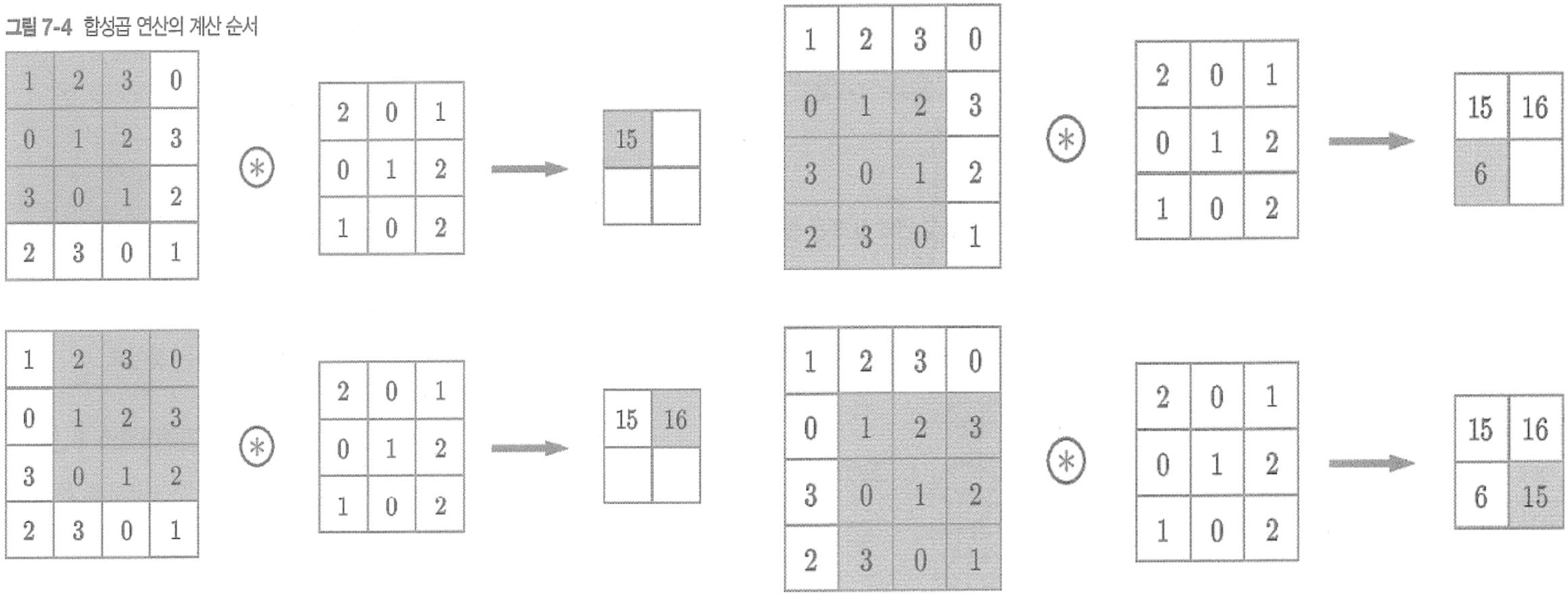
- 합성곱 계층->합성곱 연산 (예: 필터연산)
  - 합성곱 연산은 입력데이터에 필터를 적용
  - Fused multiply-add(FMA, 단일곱셈-누산)

그림 7-3 합성곱 연산의 예 : 합성곱 연산을 ⊛ 기호로 표기



합성곱 연산

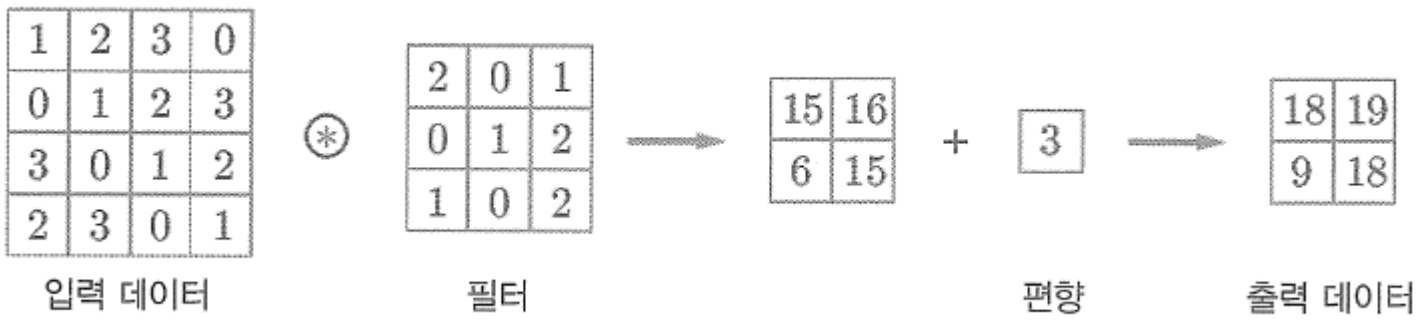
그림 7-4 합성곱 연산의 계산 순서



합성곱 연산

- Affine : CNN = 가중치 매개변수 : 필터의 매개변수 , 편향:편향

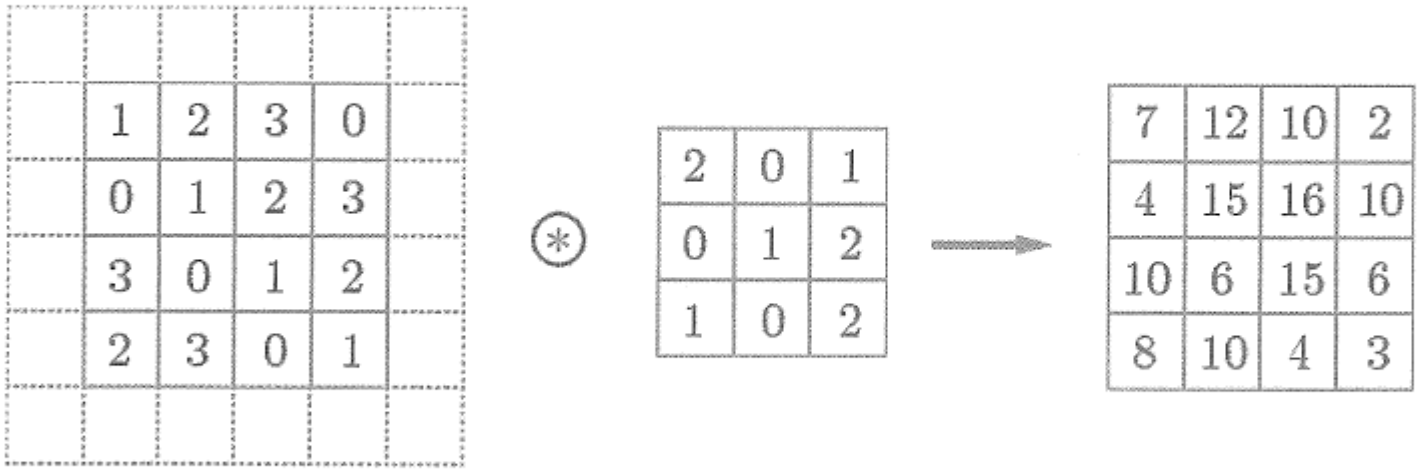
그림 7-5 합성곱 연산의 편향 : 필터를 적용한 원소에 고정값(편향)을 더한다.



Padding

- 출력 크기를 조정하기 위해 (입력의 크기를 보존)

그림 7-6 합성곱 연산의 패딩 처리 : 입력 데이터 주위에 0을 채운다(패딩은 점선으로 표시했으며 그 안의 값 '0'은 생략했다).



(4, 4)
입력 데이터(패딩 : 1)

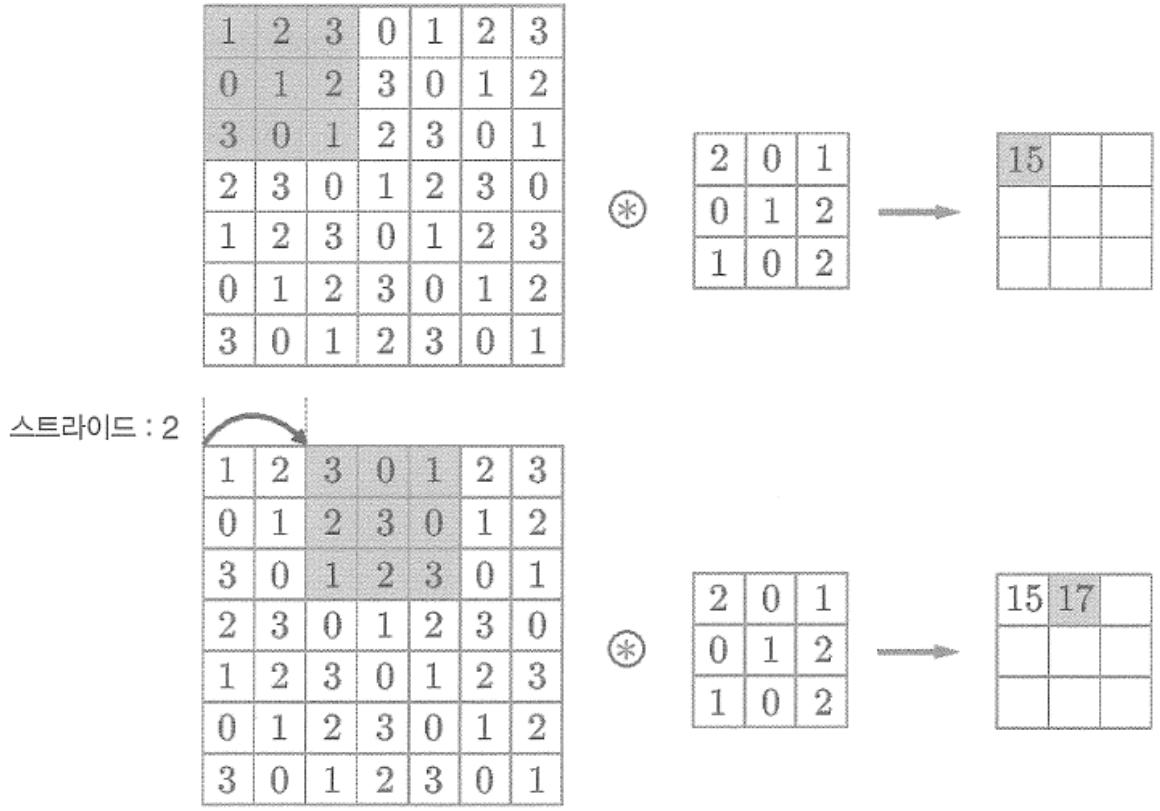
(3, 3)
필터

(4, 4)
출력 데이터

stride

- 필터를 적용하는 위치의 간격

그림 7-7 스트라이드가 2인 합성곱 연산



stride

- Stride를 키우면 출력크기는 작아지고 패팅을 크게 하면 출력크기가 커짐
  - Padding, stride에 따른 출력의 크기
  - 입력의 크기 (H, W), 필터 크기 (FH, FW), 출력 크기 (OH, OW), padding P, stride S

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

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

예 1 : [그림 7-6]의 예

입력 : (4, 4), 패딩 : 1, 스트라이드 : 1, 필터 : (3, 3)

$$OH = \frac{4+2 \cdot 1-3}{1} + 1 = 4$$

$$OW = \frac{4+2 \cdot 1-3}{1} + 1 = 4$$

예 2 : [그림 7-7]의 예

입력 : (7, 7), 패딩 : 0, 스트라이드 : 2, 필터 : (3, 3)

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

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

예 3

입력 : (28, 31), 패딩 : 2, 스트라이드 : 3, 필터 : (5, 5)

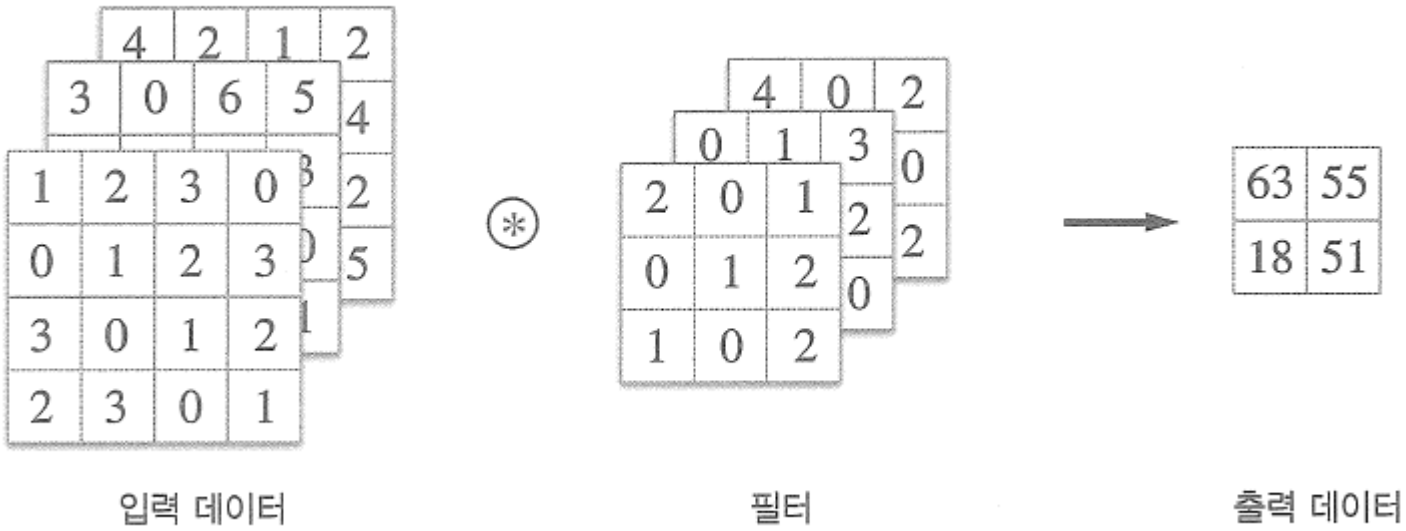
$$OH = \frac{28+2 \cdot 2-5}{1} + 1 = 10$$

$$OW = \frac{31+2 \cdot 2-5}{1} + 1 = 11$$

3D 데이터의 합성곱 연산

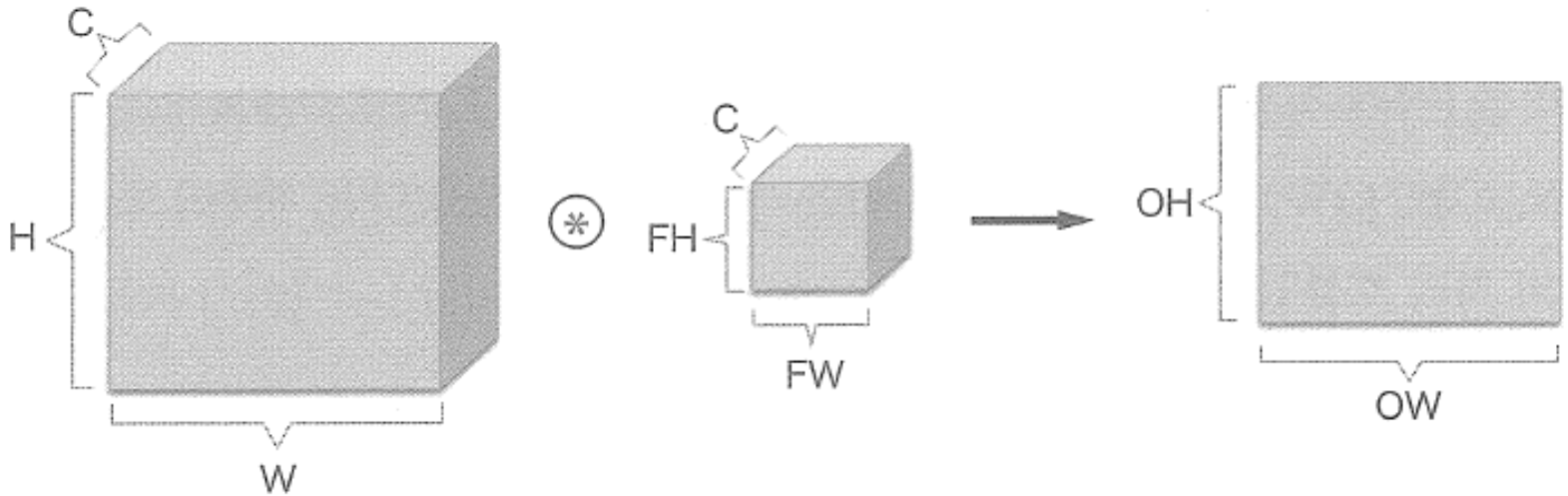
- 길이방향(채널방향)으로 특징맵이 늘어남
  - 입력 데이터의 채널 수와 필터의 채널 수가 동일

그림 7-8 3차원 데이터 합성곱 연산의 예



블록 형상

그림 7-10 합성곱 연산을 직육면체 블록으로 생각한다. 블록의 형상에 주의할 것!

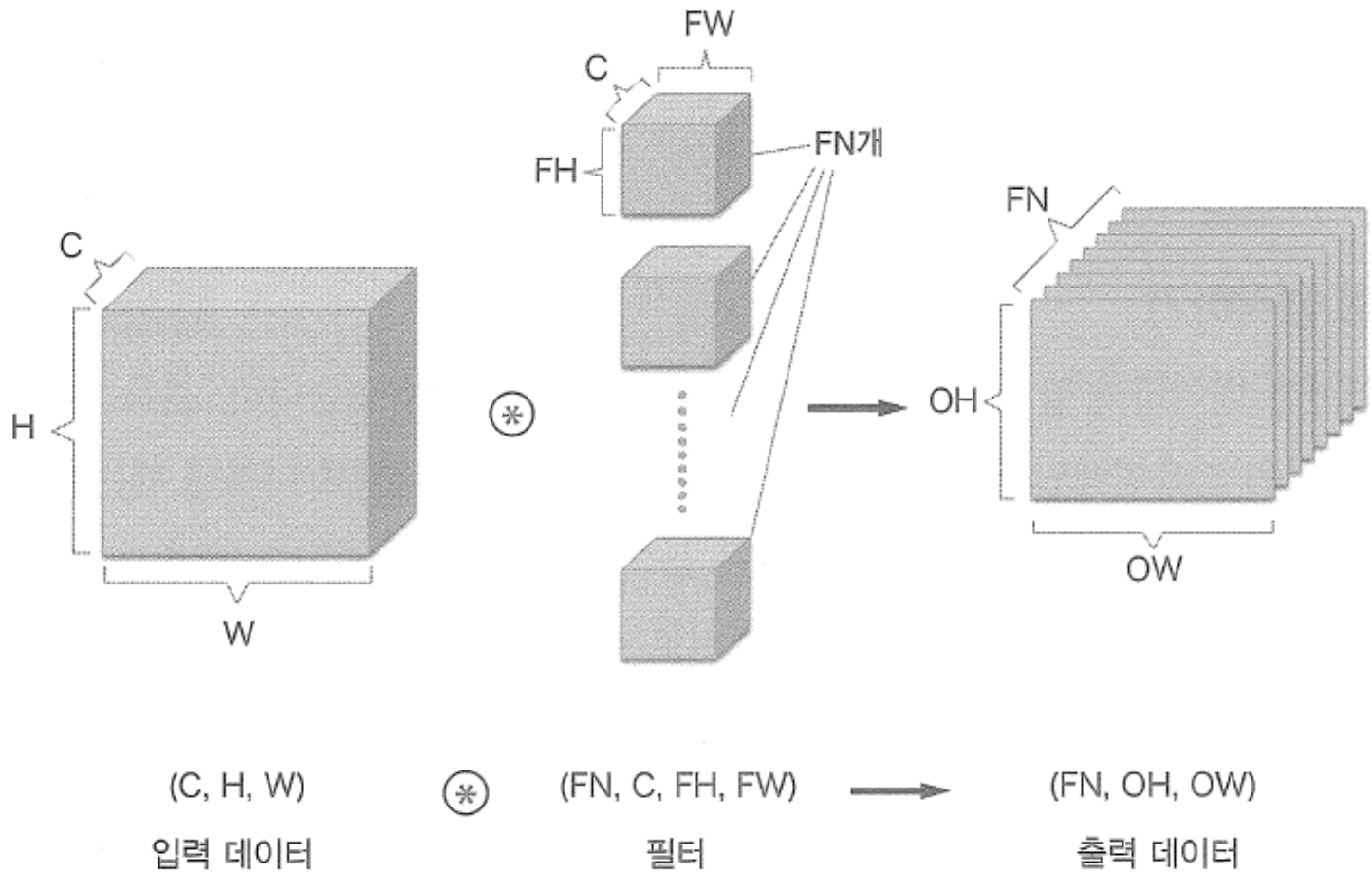


(C, H, W) * (C, FH, FW) → (1, OH, OW)

입력 데이터 | 필터 | 출력 데이터

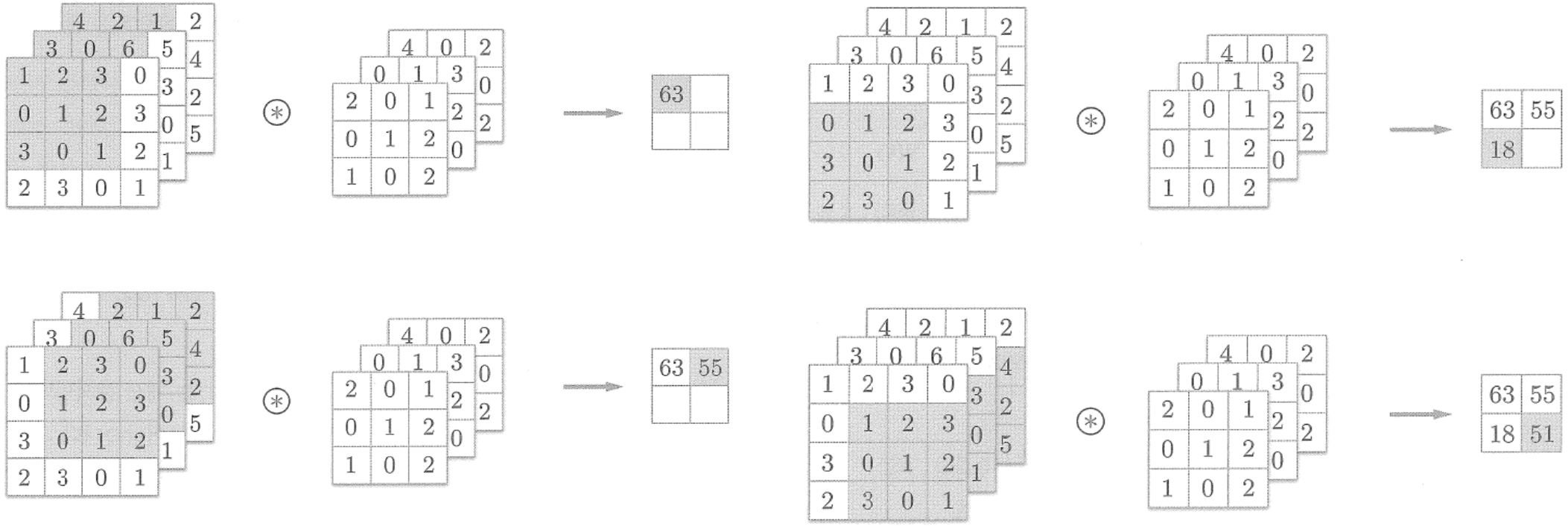
블록 형상

그림 7-11 여러 필터를 사용한 합성곱 연산의 예



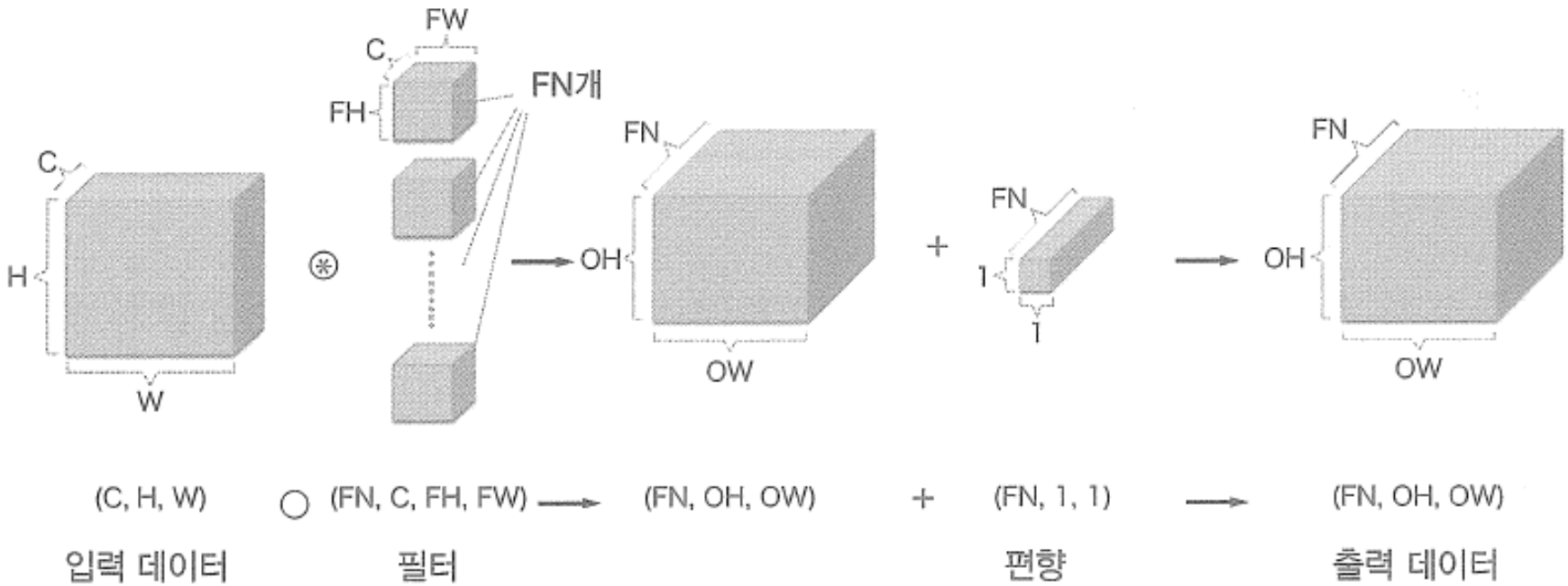
3D 데이터의 합성곱 연산

그림 7-9 3차원 데이터 합성곱 연산의 계산 순서



블록 형상

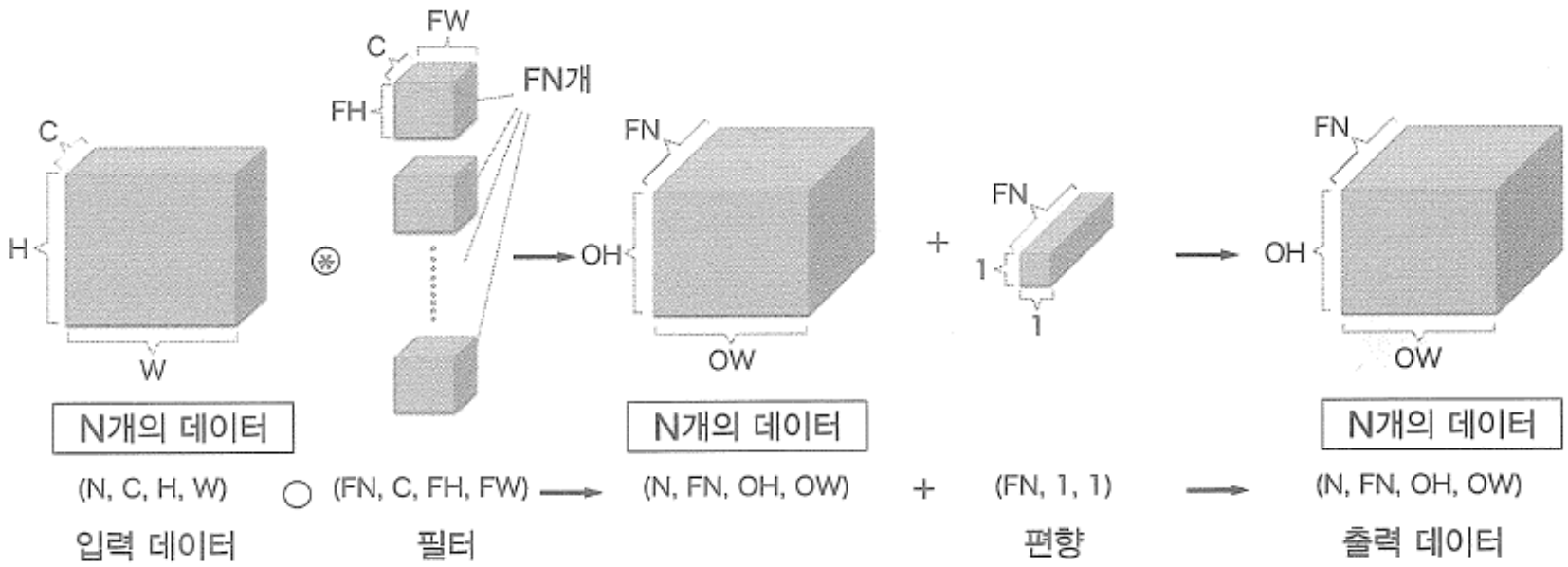
그림 7-12 합성곱 연산의 처리 흐름(편향 추가)



Batch

- N개의 데이터를 배치처리 할 때

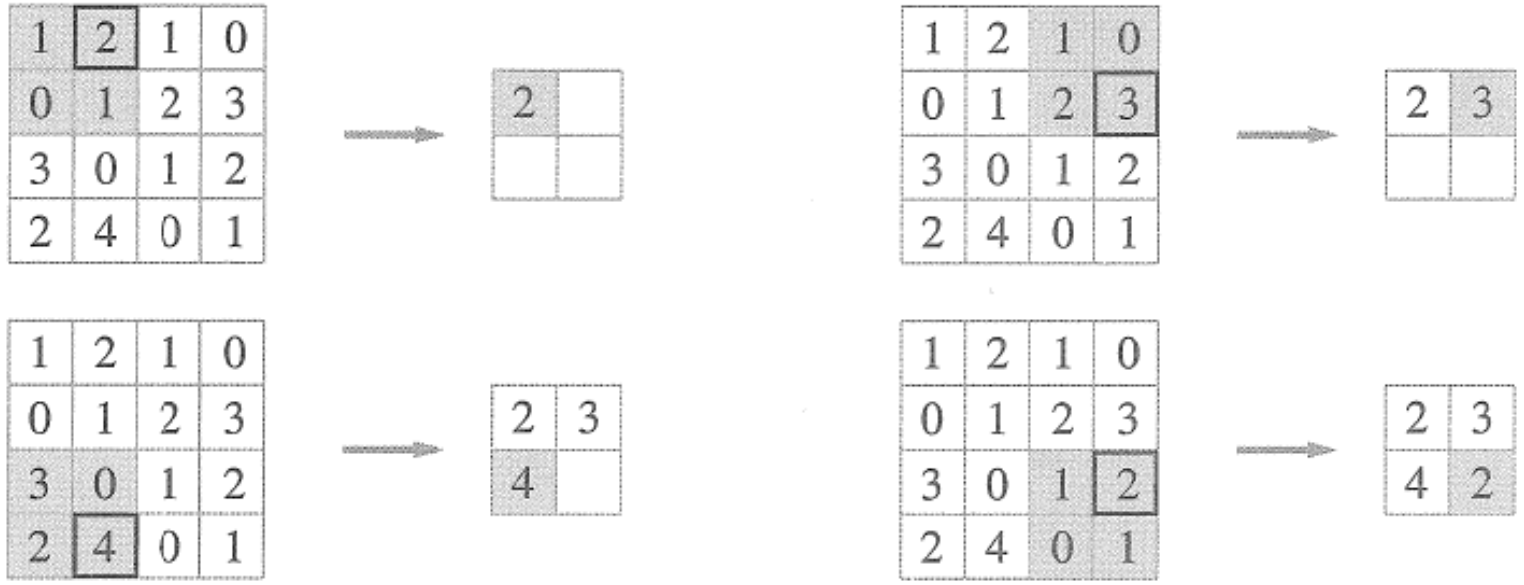
그림 7-13 합성곱 연산의 처리 흐름(배치 처리)



Pooling

- 세로, 가로 방향의 공간을 줄이는 연산
- 최대 풀링

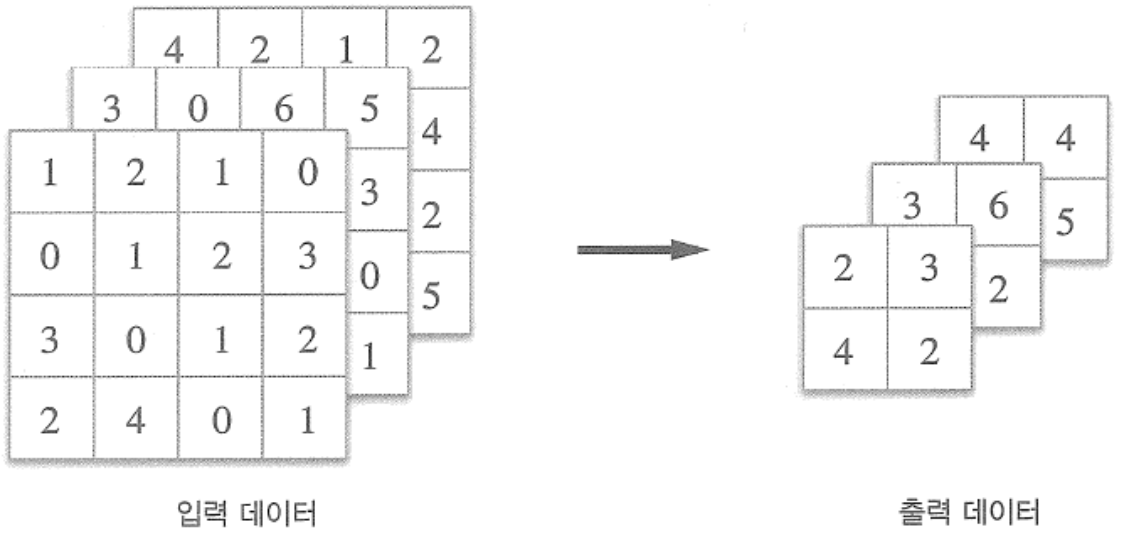
그림 7-14 최대 풀링의 처리 순서



Pooling계층의 특징

- 학습해야할 매개변수가 없다
  - 풀링 계층은 합성곱 계층과 달리 학습해야 할 매개변수가 없다. 풀링은 대상영역에서 최대값이나 평균을 취하는 명확한 처리이므로 특별히 학습할 것이 없다
- 채널 수가 변하지 않는다
  - 풀링 연산은 입력 데이터의 채널 수 그대로 출력 데이터로 내보낸다.

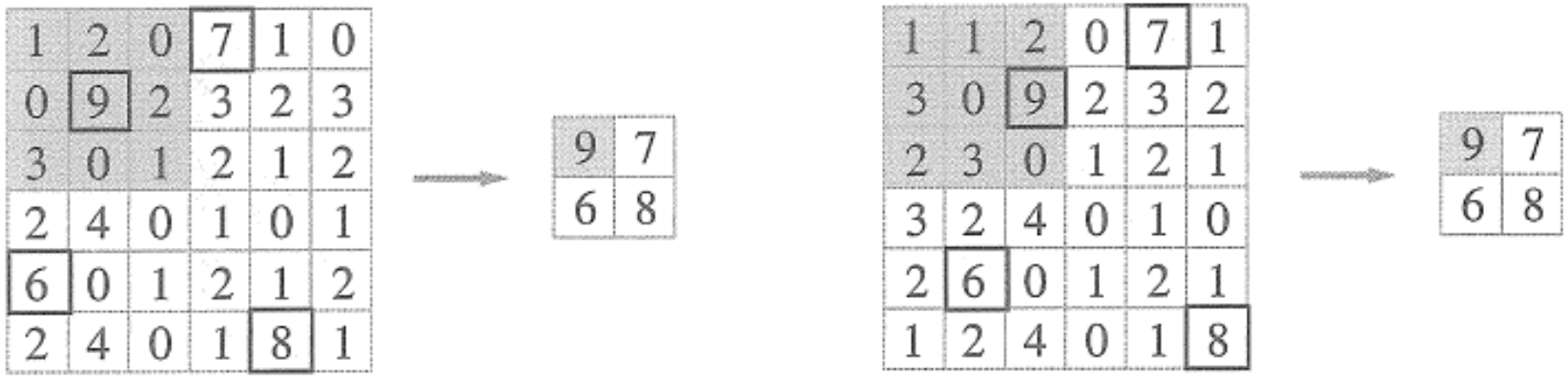
그림 7-15 풀링은 채널 수를 바꾸지 않는다.



Pooling 계층의 특징

- 입력의 변화에 영향을 적게 받는다 (강건하다)

그림 7-16 입력 데이터가 가로로 1원소만큼 어긋나도 출력은 같다(데이터에 따라서는 다를 수도 있다).



합성곱/풀링 계층 구현

- 4차원 배열: (10, 1, 28, 28)- 높이 28, 너비 28, 채널 1개인 데이터 10개
- Im2col로 데이터 전개
  - 입력데이터를 필터링(가중치 계산)하기 좋게 전개하는 함수

그림 7-17 (개략적인) im2col의 동작

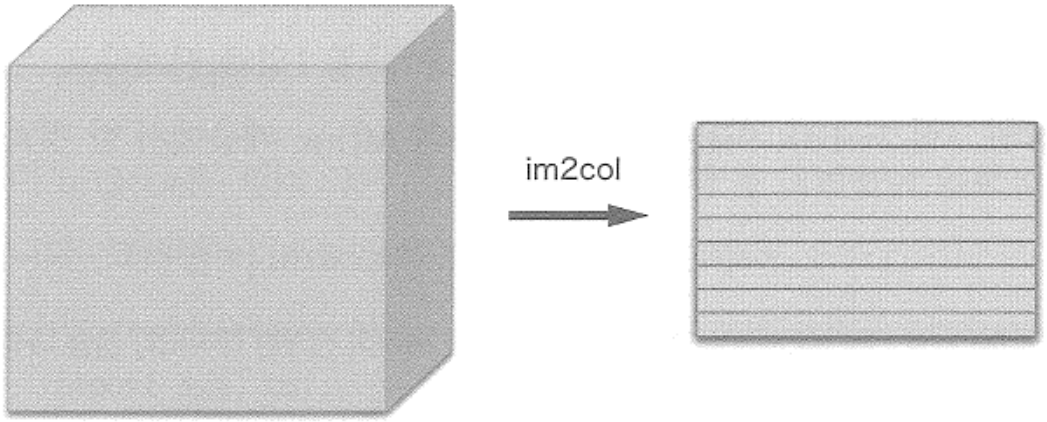
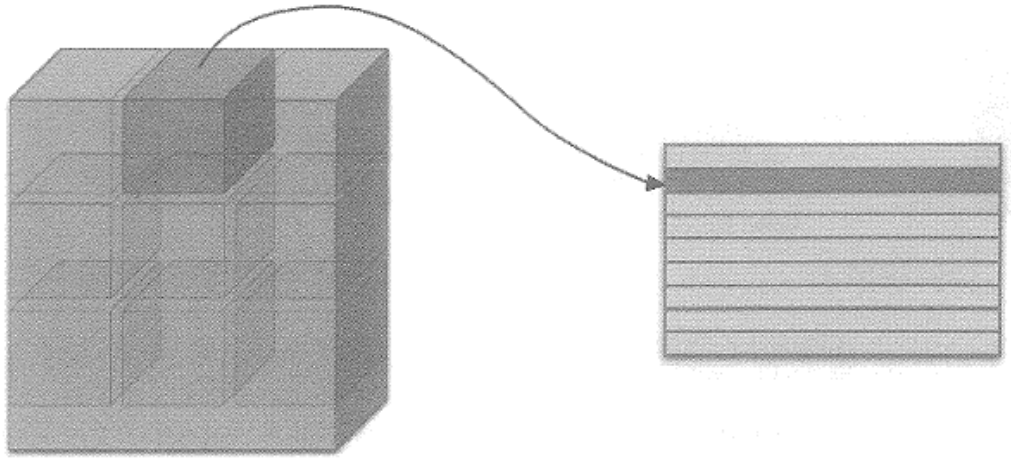


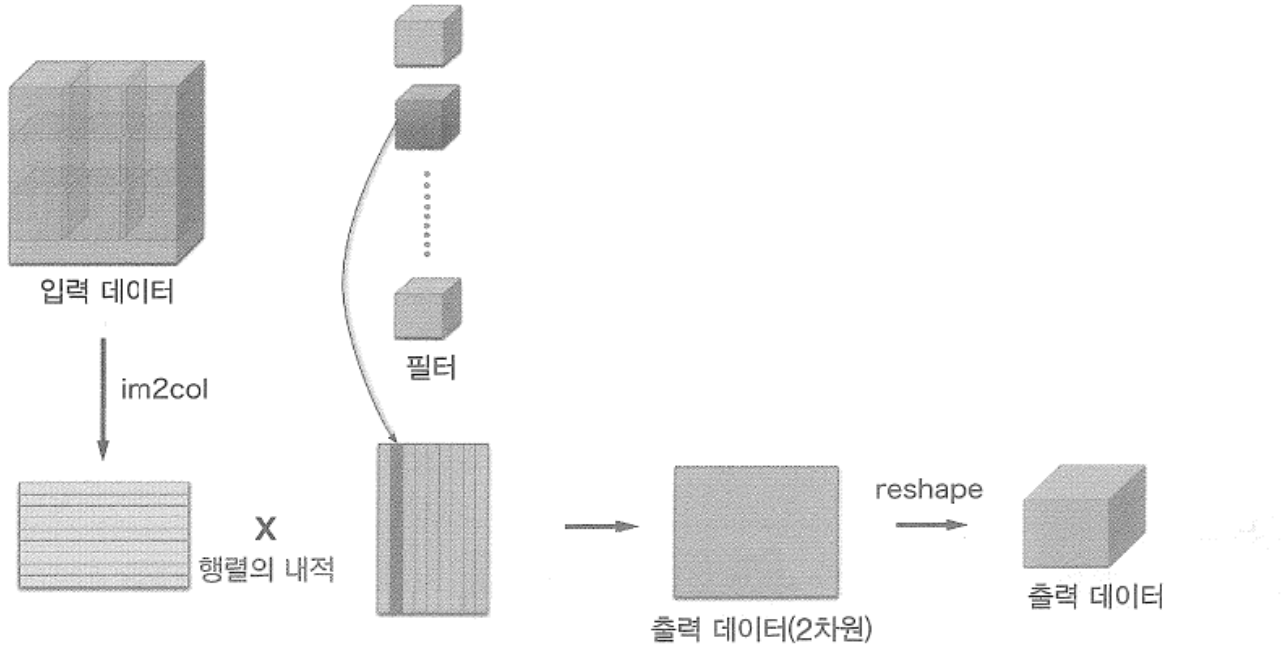
그림 7-18 필터 적용 영역을 앞에서부터 순서대로 1줄로 펼친다.



합성곱/풀링 계층 구현

- 합성곱 계층 구현
  - common/util.py

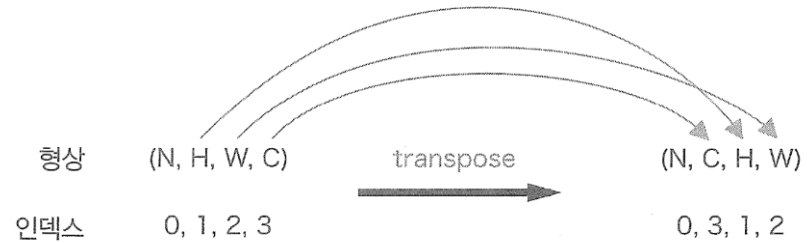
그림 7-19 합성곱 연산의 필터 처리 상세 과정 : 필터를 세로로 1열로 전개하고, im2col이 전개한 데이터와 행렬 내적을 계산합니다. 마지막으로 출력 데이터를 변형(reshape)합니다.



합성곱/풀링 계층 구현:Convolution class

```
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
```

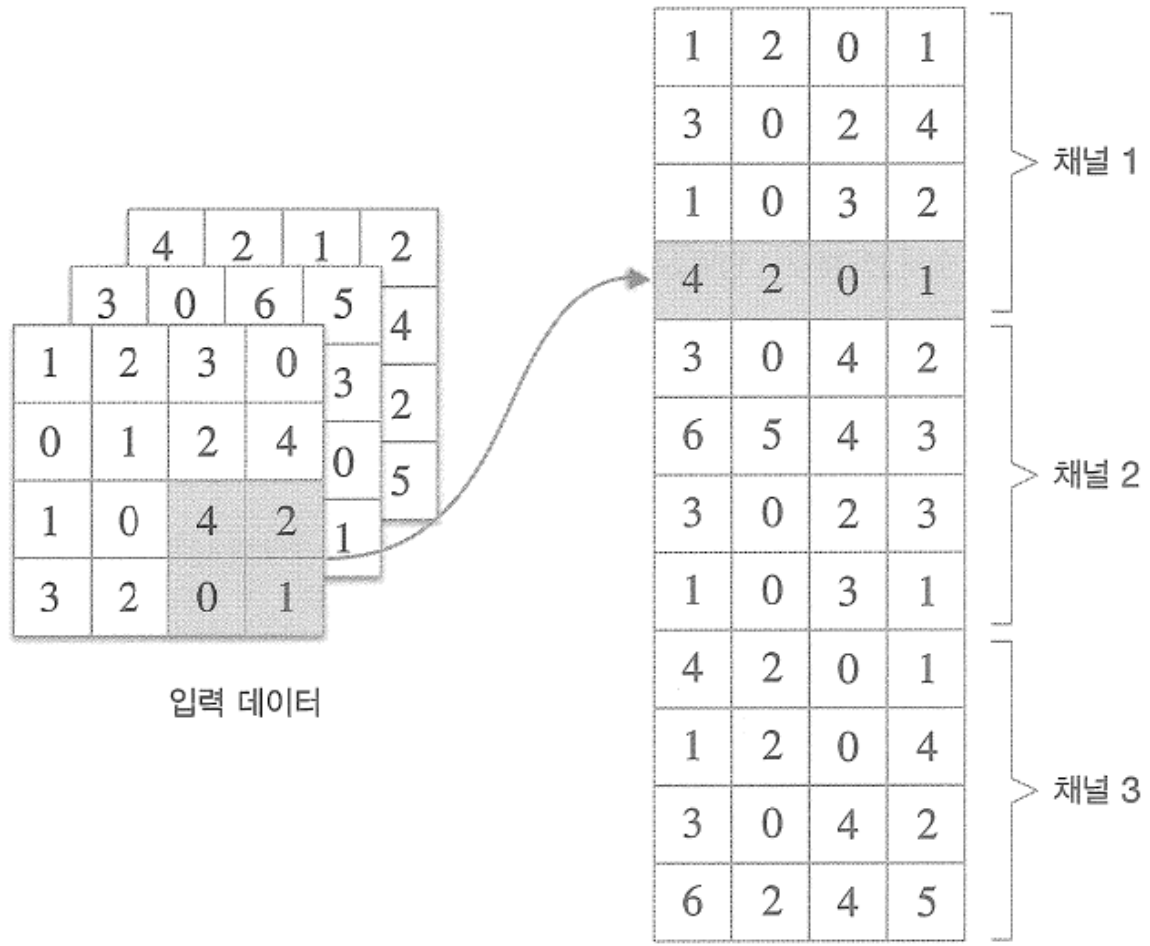
그림 7-20 넘파이의 transpose 함수로 축 순서 변경하기 : 인덱스(번호)로 축의 순서를 변경한다.



합성곱/풀링 계층 구현

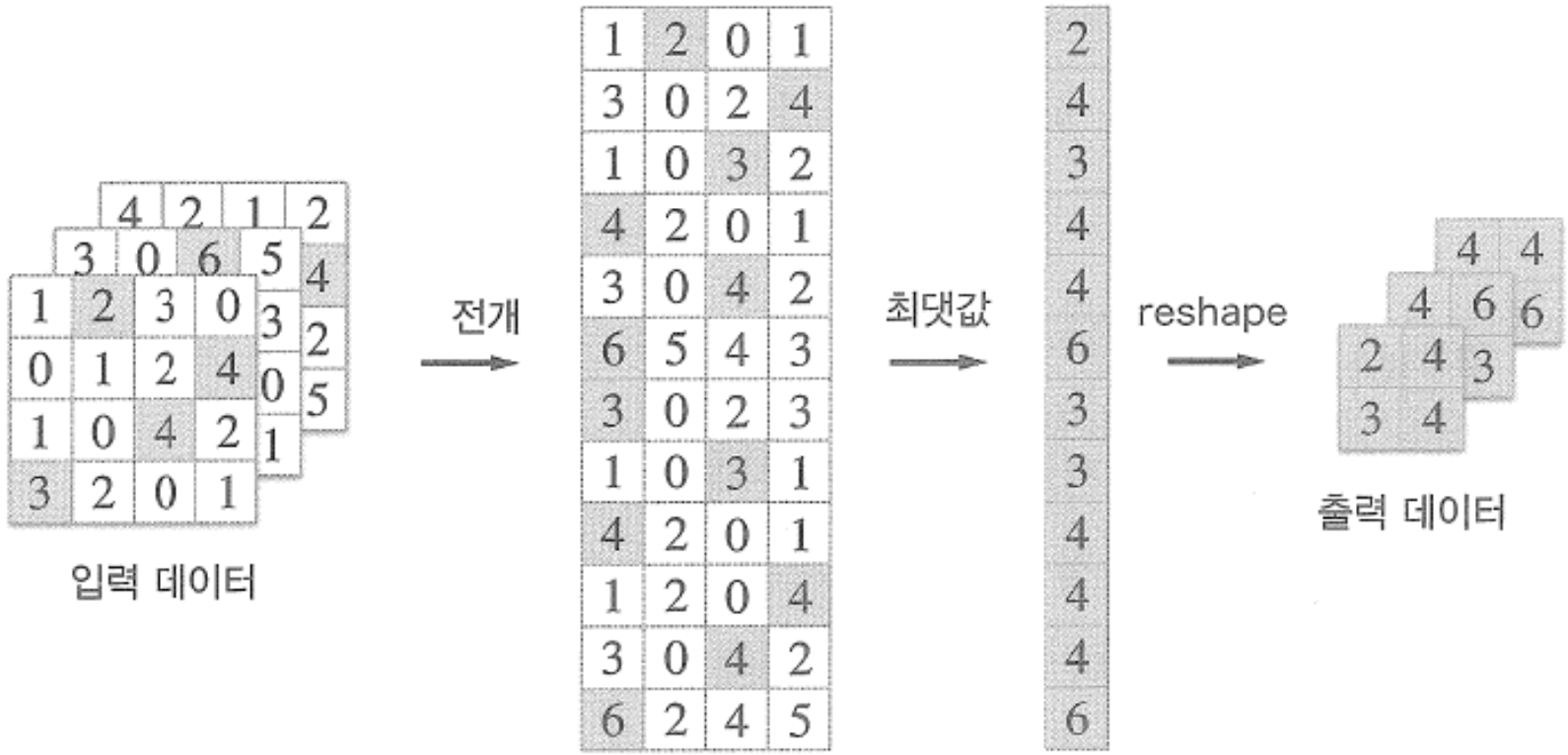
- 풀링 계층 구현

그림 7-21 입력 데이터에 풀링 적용 영역을 전개 (2×2 풀링의 예)



합성곱/풀링 계층 구현

그림 7-22 풀링 계층 구현의 흐름 : 풀링 적용 영역에서 가장 큰 원소는 회색으로 표시



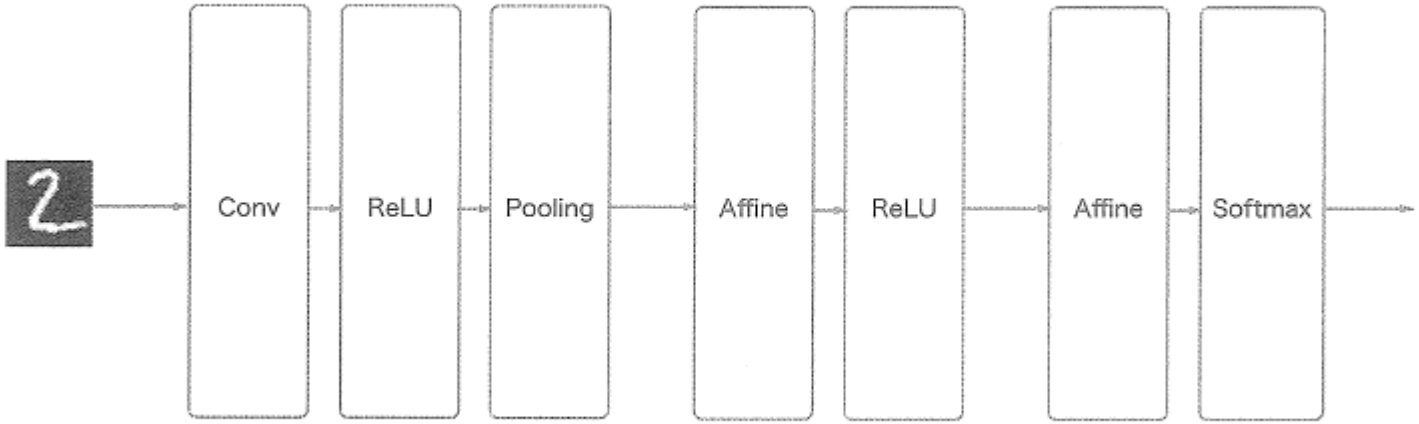
합성곱/풀링 계층 구현

```
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)

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

        # 최댓값 (2)
        out = np.max(col, axis=1)

        # 성형 (3)
        out = out.reshape(N, out_h, out_w, C).transpose(0, 3, 1, 2)

        return out
```

CNN 구현

- 07-train_convent.py

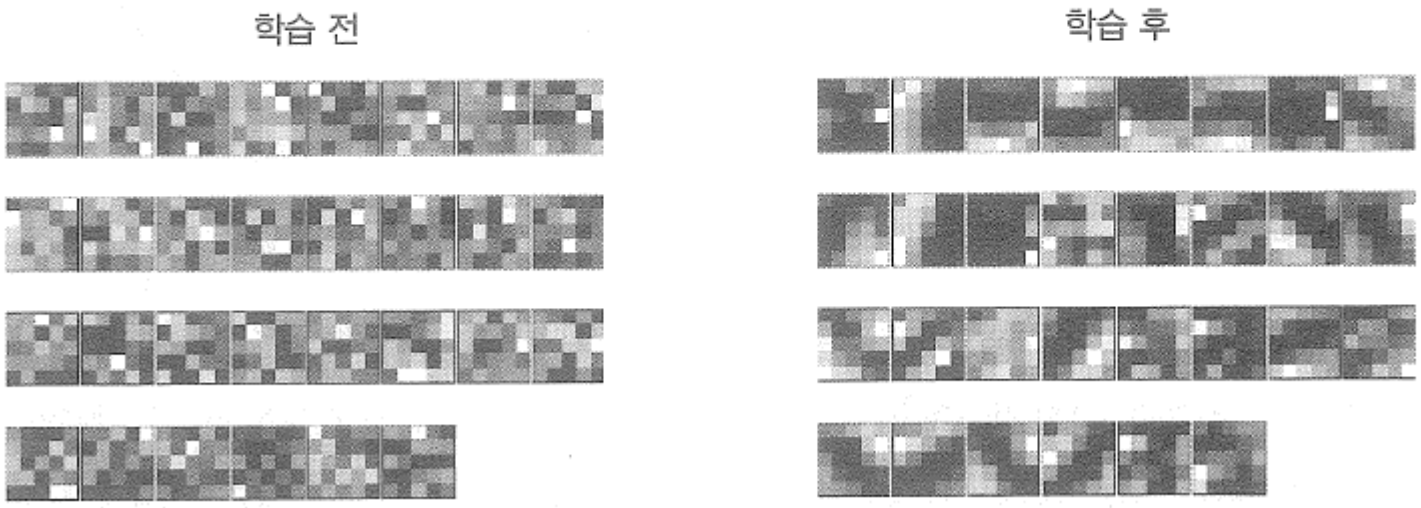
그림 7-23 단순한 CNN의 네트워크 구성



CNN Visualization

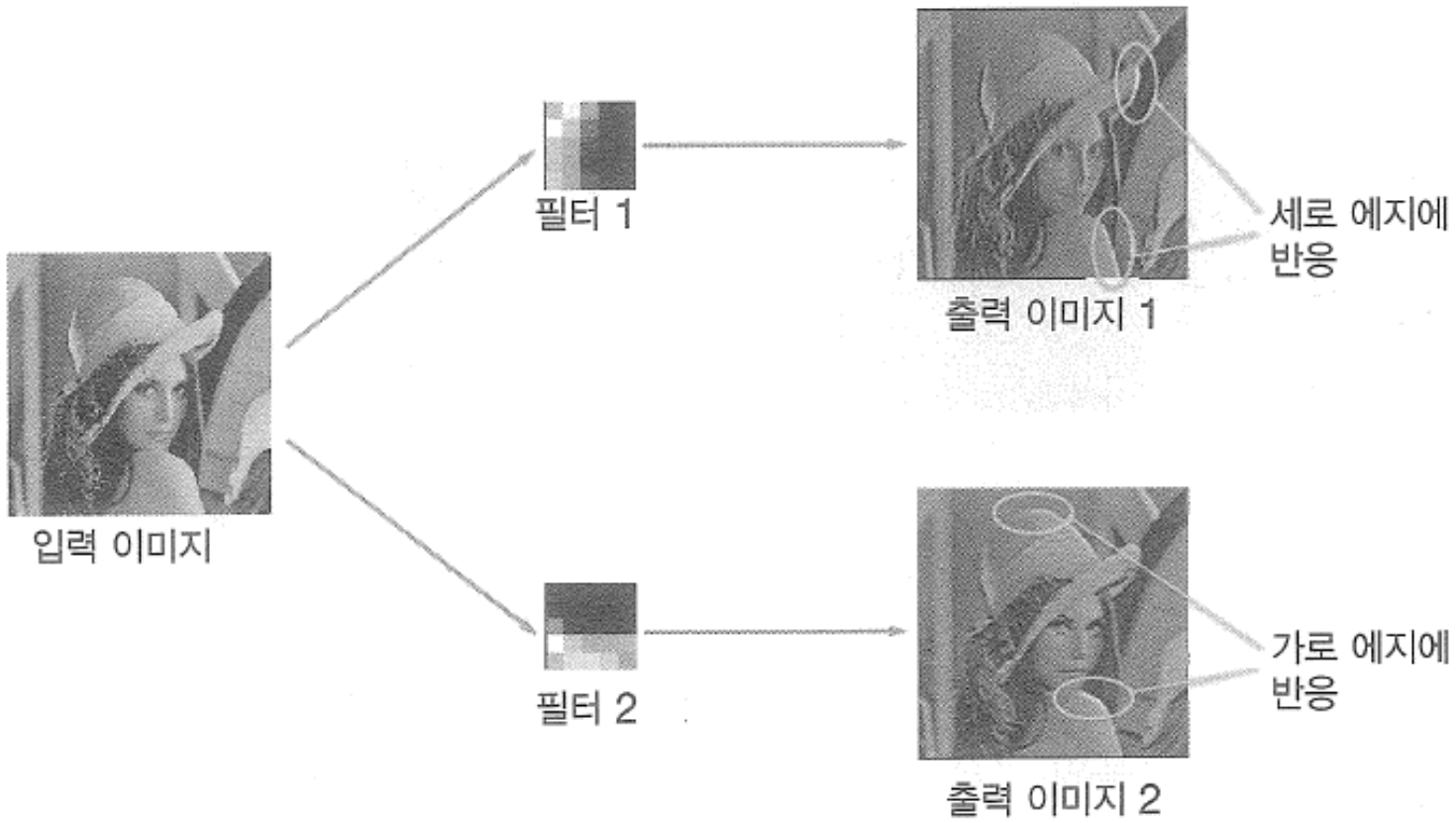
- 1번째 층의 가중치 시각화
  - 07-visualize_fileter.py

그림 7-24 학습 전과 후의 1번째 층의 합성곱 계층의 가중치 : 가중치의 원소는 실수이지만, 이미지에서는 가장 작은 값(0)은 검은색, 가장 큰 값(255)은 흰색으로 정규화하여 표시함



CNN Visualization

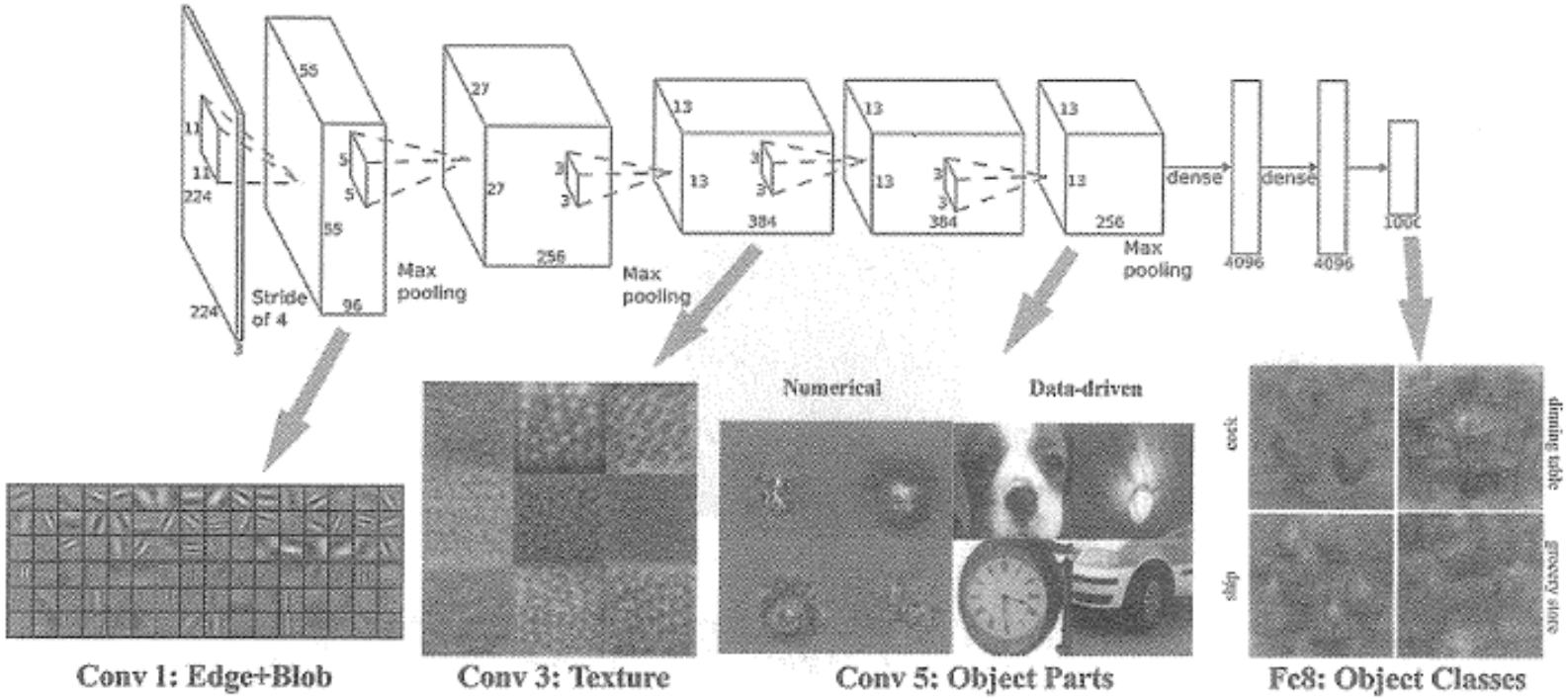
그림 7-25 가로 에지와 세로 에지에 반응하는 필터 : 출력 이미지 1은 세로 에지에 흰 픽셀이 나타나고, 출력 이미지 2는 가로 에지에 흰 픽셀이 많이 나온다.



CNN Visualization

- 층 깊이에 따른 추출 정보의 변화

그림 7-26 CNN의 합성곱 계층에서 추출되는 정보. 1번째 층은 에지와 블롭, 3번째 층은 텍스처, 5번째 층은 사물의 일부, 마지막 완전연결 계층은 사물의 클래스(개, 자동차 등)에 뉴런이 반응한다.[19]



AlexNet

- 2012년 발표됨. 딥러닝 열풍을 일으키는데 큰 역할
  - 활성화 함수로 ReLU 사용
  - Local Response Normalization이라는 국소적 정규화를 실시하는 계층을 이용
  - Dropout 사용

그림 7-28 AlexNet의 구성[21]