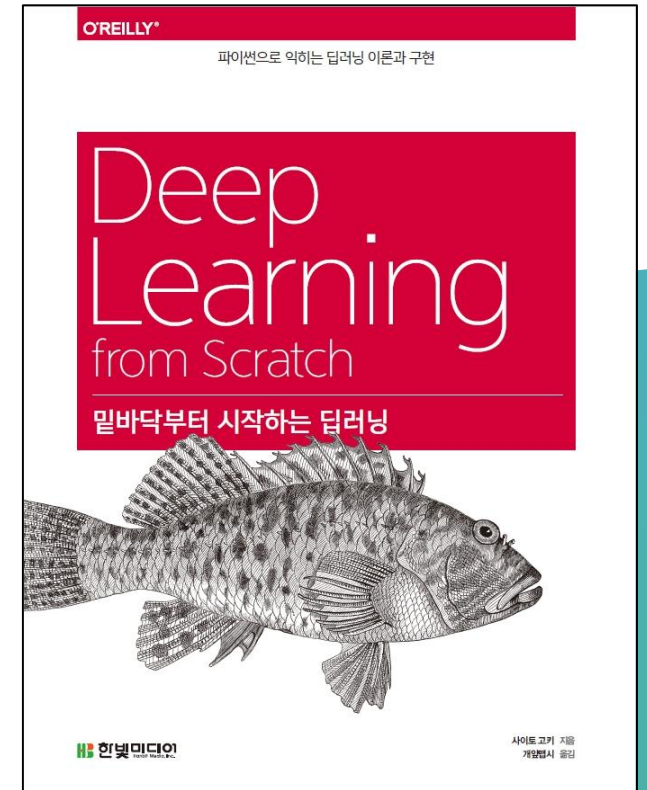


▶ 밑바닥부터 시작하는 딥러닝

CHAPTER 7

합성곱 신경망(CNN)



Contents

- CHAPTER 7 합성곱 신경망(CNN)
 - 7.1 전체 구조
 - 7.2 합성곱 계층
 - 7.2.1 완전연결 계층의 문제점
 - 7.2.2 합성곱 연산
 - 7.2.3 패딩
 - 7.2.4 스트라이드
 - 7.2.5 3차원 데이터의 합성곱 연산
 - 7.2.6 블록으로 생각하기
 - 7.2.7 배치 처리
 - 7.3 풀링 계층
 - 7.3.1 풀링 계층의 특징
 - 7.4 합성곱/풀링 계층 구현하기

Contents

- CHAPTER 7 합성곱 신경망(CNN)
 - 7.4.1 4차원 배열
 - 7.4.2 im2col로 데이터 전개하기
 - 7.4.3 합성곱 계층 구현하기
 - 7.4.4 풀링 계층 구현하기
 - 7.5 CNN 구현하기
 - 7.6 CNN 시각화하기
 - 7.6.1 1번째 층의 가중치 시각화하기
 - 7.6.2 층 깊이에 따른 추출 정보 변화
 - 7.7 대표적인 CNN
 - 7.7.1 LeNet
 - 7.7.2 AlexNet
 - 7.8 정리



CHAPTER 7 합성곱 신경망(CNN)

CNN의 메커니즘을 자세히 설명하고 파이썬으로 구현하기

SECTION 07 합성곱 신경망(CNN)



7.1 전체 구조

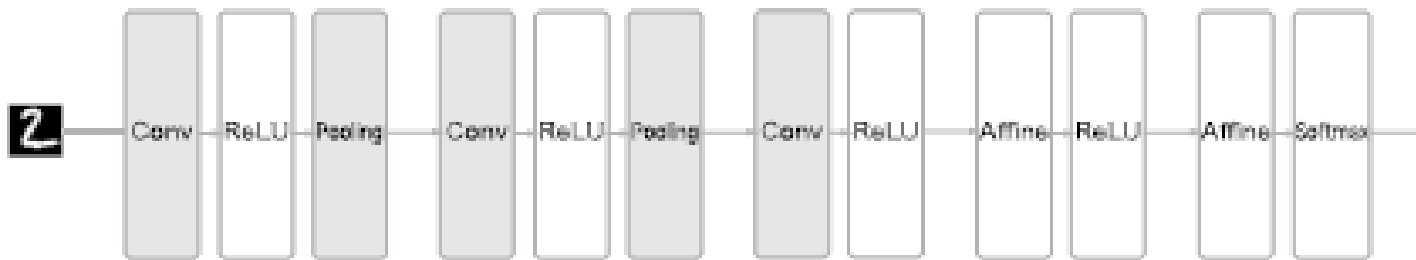
CNN도 지금까지 본 신경망과 같이 레고 블록처럼 계층을 조합하여 만들 수 있다.

다만, 합성곱 계층(convolutional layer)과 풀링 계층(pooling layer)이 새롭게 등장한다

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)





7.2.1 완전연결 계층의 문제점

'데이터의 형상이 무시'된다는 사실이다.
입력데이터가 이미지인 경우를 예로 들면, 이미지는 통상
세로·가로·채널(색상)로 구성된 3차원 데이터이다

그러나 완전연결 계층은 형상을 무시하고 모든 입력 데이
터를 동등한 뉴런(같은 차원의 뉴런)으로 취급하여 형상에
담긴 정보를 살릴 수 없다.
한편, 합성곱 계층은 형상을 유지한다.

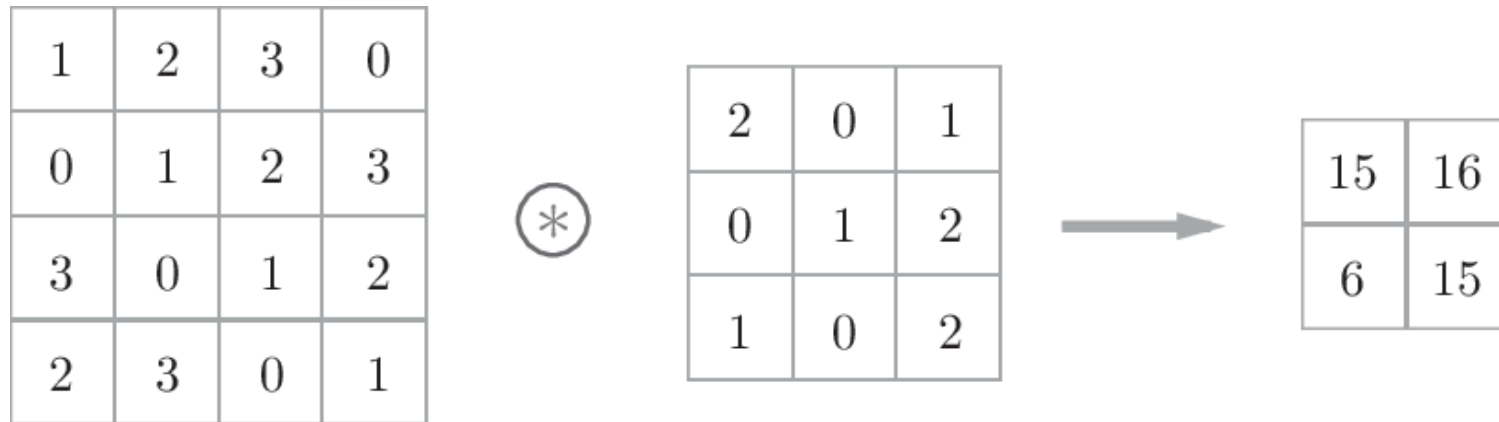
CNN에서는 합성곱 계층의 입출력 데이터를 특징 맵 feature map
이라고도 한다. 합성곱 계층의
입력 데이터를 입력 특징 맵 input feature map, 출력 데이터를 출력 특
징 맵 output feature map이라고 하는 식이다.
이 책에서는 '입출력 데이터'와 '특징 맵'을 같은 의미로 사
용한다

SECTION 07 합성곱 신경망(CNN)



7.2.2 합성곱 연산

그림 7-3 합성곱 연산의 예 : 합성곱 연산을 기호로 표기



합성곱 연산은 필터의 윈도우_{window}를 일정 간격으로 이동해가며 입력 데이터에 적용한다.

여기에서 말하는 윈도우는 [그림 7-4]의 회색 3×3 부분을 가리킨다.

이 그림에서 보듯 입력과 필터에서 대응하는 원소끼리 곱한 후 그 총합을 구한다(이 계산을 단일 곱셈-누산_{fusedmultiply-add, FMA}이라 한다).* 그리고 그 결과를 출력의 해당 장소에 저장

SECTION 07 합성곱 신경망(CNN)



7.2.2 합성곱 연산

그림 7-4 합성곱 연산의 계산 순서

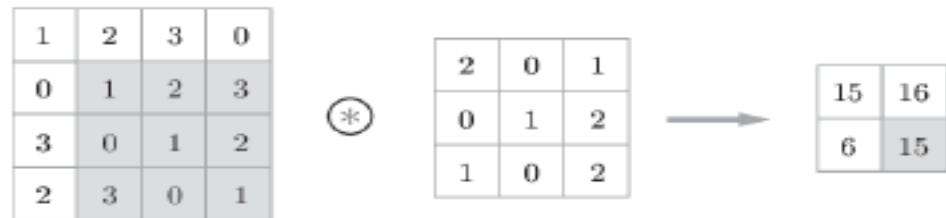
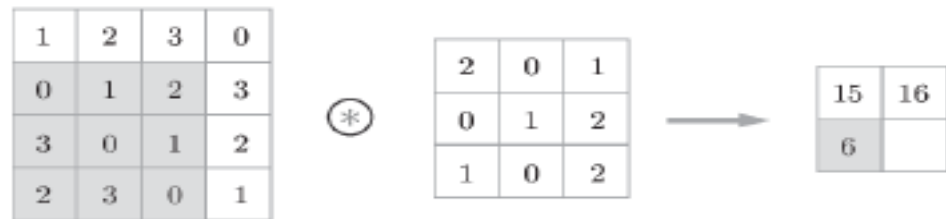
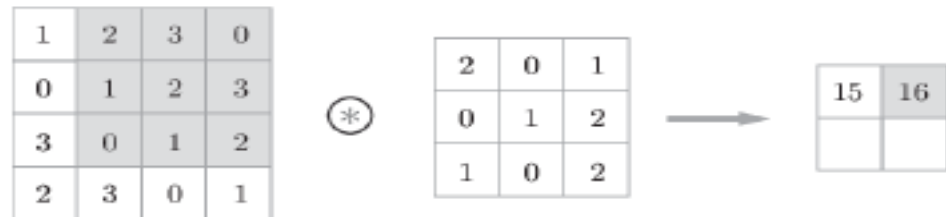
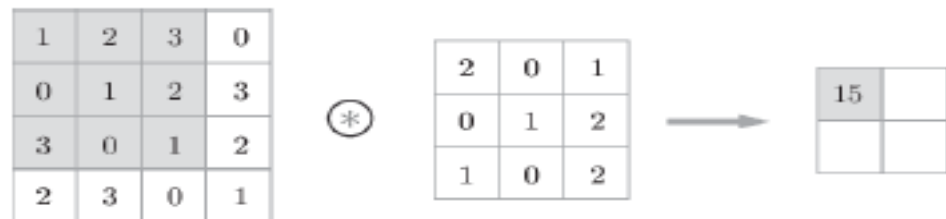
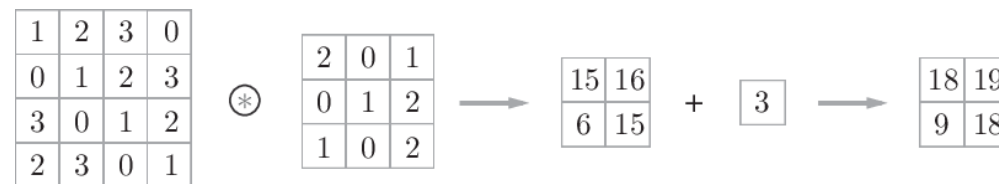


그림 7-5 합성곱 연산의 편향 : 필터를 적용한 원소에 고정값(편향)을 더한다



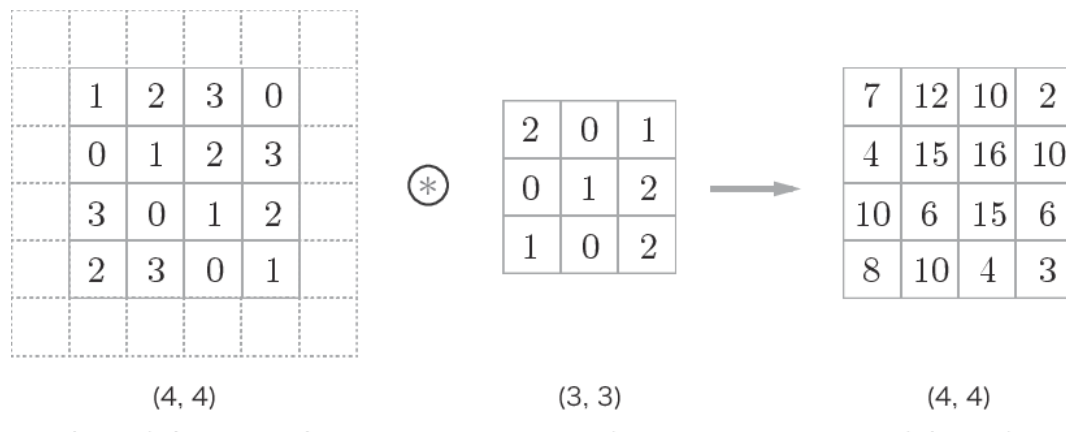
SECTION 07 합성곱 신경망(CNN)



7.2.3 패딩

합성곱 연산을 수행하기 전에 입력 데이터 주변을 특정 값(예컨대 0)으로 채우기도 한다. 이를 패딩_{padding}이라 하며, 합성곱 연산에서 자주 이용하는 기법이다

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



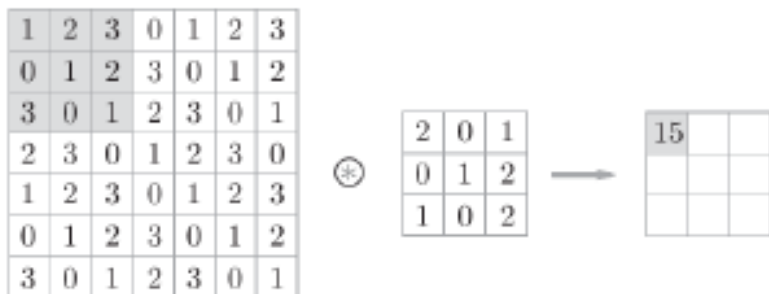
SECTION 07 합성곱 신경망(CNN)



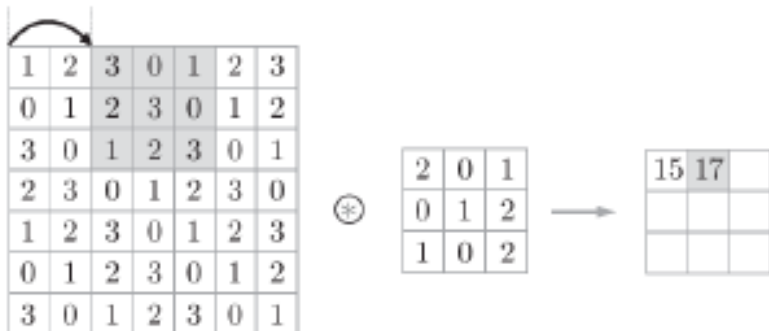
7.2.4 스트라이드

필터를 적용하는 위치의 간격을 스트라이드_{stride}라고 한다

그림 7-7 스트라이드가 2인 합성곱 연산



스트라이드 : 2



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

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

[식 7.1]

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)

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

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

예 3

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

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

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

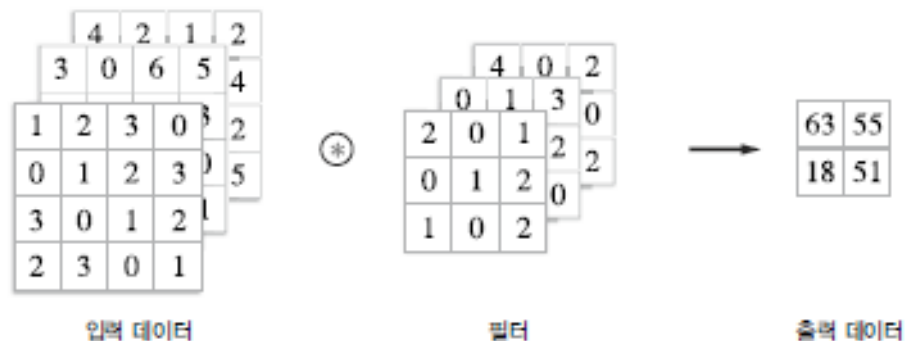
※ 교재의 상세 과정을 참고하여 실습을 진행합니다.

SECTION 07 합성곱 신경망(CNN)



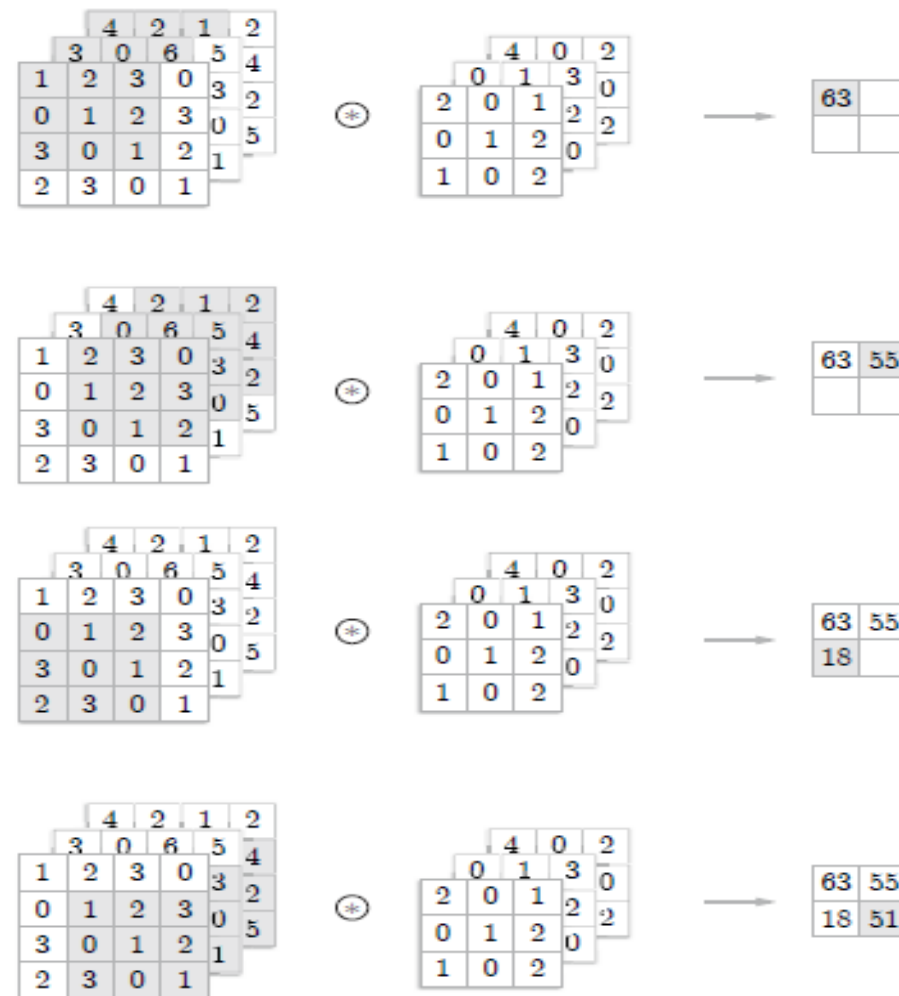
7.2.5 3차원 데이터의 합성곱 연산

그림 7-8 3차원 데이터 합성곱 연산의 예



3차원의 합성곱 연산에서 주의할 점은 입력 데이터의 채널 수와 필터의 채널 수가 같아야 한다는 것이다

그림 7-9 3차원 데이터 합성곱 연산의 계산 순서



SECTION 07 합성곱 신경망(CNN)



7.2.6 블록으로 생각하기

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

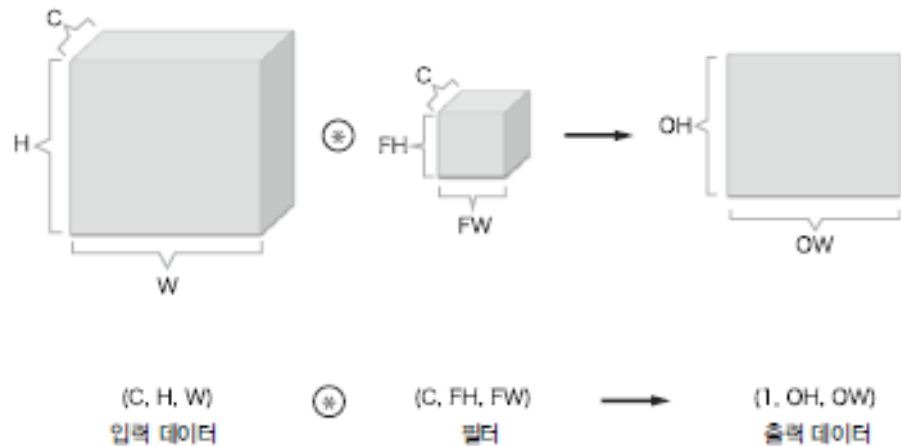


그림 7-11 여러 필터를 사용한 합성곱 연산의 예

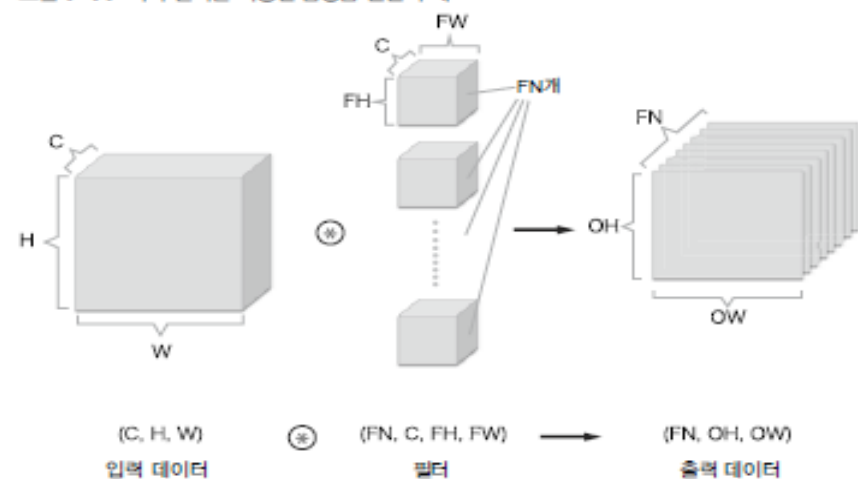
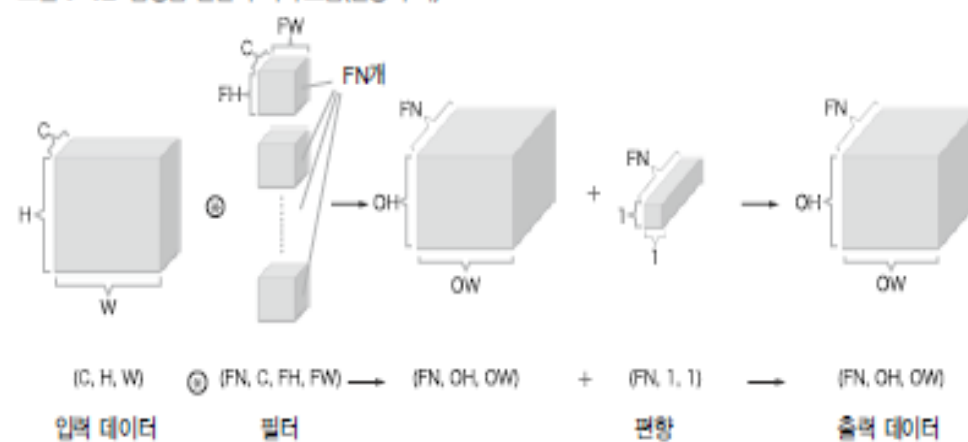


그림 7-12 합성곱 연산의 처리 흐름(편향 추가)

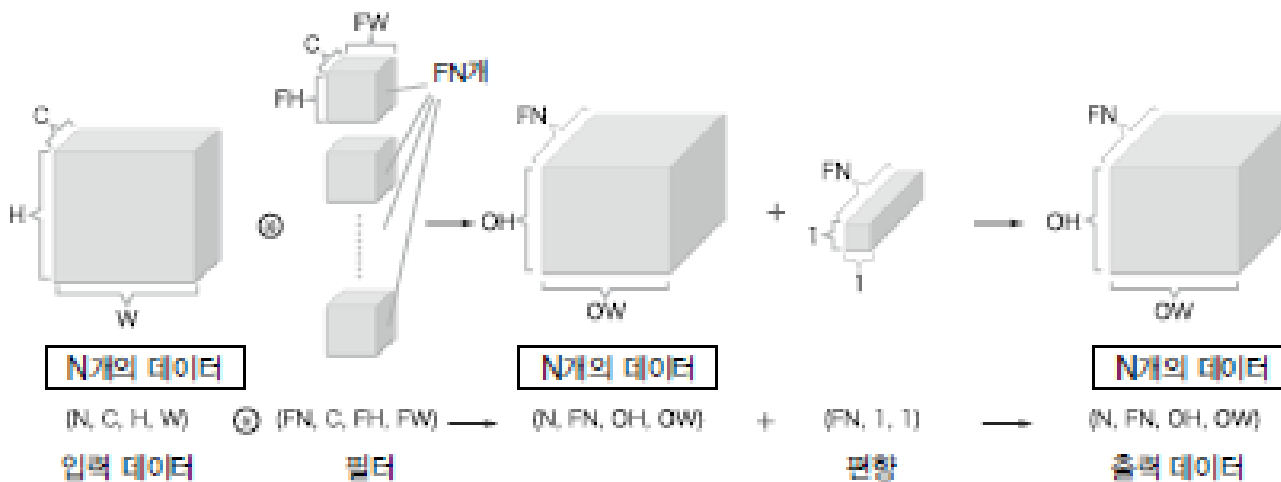


SECTION 07 합성곱 신경망(CNN)



7.2.7 배치 처리

그림 7-13 합성곱 연산의 처리 흐름(배치 처리)



SECTION 07 합성곱 신경망(CNN)



7.3 풀링 계층

그림 7-14 최대 풀링의 처리 순서

1	2	1	0
0	1	2	3
3	0	1	2
2	4	0	1

→

2	

1	2	1	0
0	1	2	3
3	0	1	2
2	4	0	1

→

2	3
4	

1	2	1	0
0	1	2	3
3	0	1	2
2	4	0	1

→

2	3

1	2	1	0
0	1	2	3
3	0	1	2
2	4	0	1

→

2	3
4	2



7.3.1 풀링 계층의 특징

학습해야 할 매개변수가 없다

풀링 계층은 합성곱 계층과 달리 학습해야 할 매개변수가 없다.

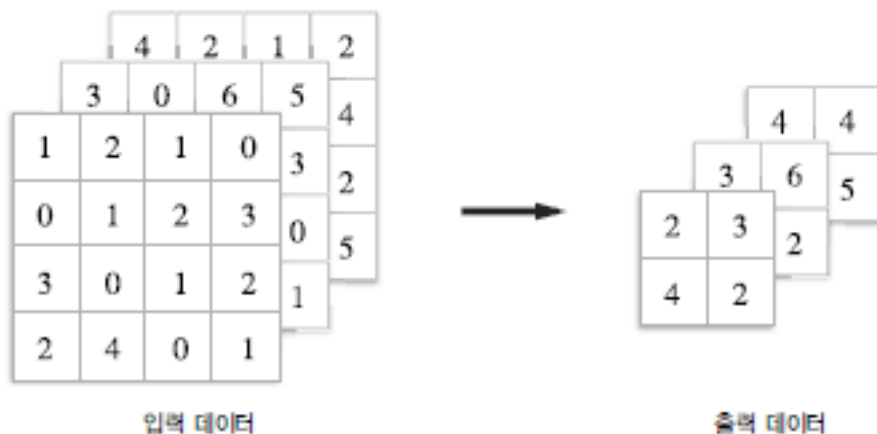
풀링은 대상 영역에서 최댓값이나 평균을 취하는 명확한 처리이므로 특별히 학습할 것이 없다.

채널 수가 변하지 않는다

풀링 연산은 입력 데이터의 채널 수 그대로 출력 데이터로 내보낸다.

[그림 7-15]처럼 채널마다 독립적으로 계산하기 때문이다.

그림 7-15 풀링은 채널 수를 바꾸지 않는다.

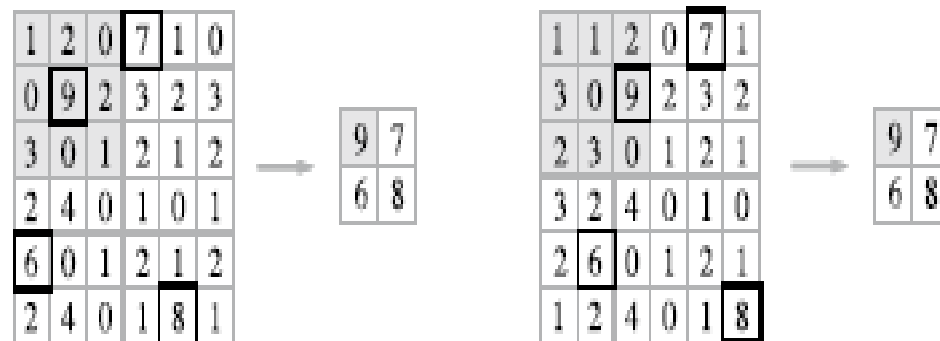


입력의 변화에 영향을 적게 받는다(강건하다)

입력 데이터가 조금 변해도 풀링의 결과는 잘 변하지 않는다.

예를 들어 [그림 7-16]은 입력 데이터의 차이(데이터가 오른쪽으로 1칸씩 이동)를 풀링이 흡수해 사라지게 하는 모습을 보여준다

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



SECTION 07 합성곱 신경망(CNN)



7.4.1 4차원 배열

앞에서 설명한 대로 CNN에서 계층 사이를 흐르는 데이터는 4차원이다
이를 파이썬으로 구현하면 다음과 같다

```
>>> x = np.random.rand(10, 1, 28, 28) # 무작위로 데이터 생성
>>> x.shape
(10, 1, 28, 28)
```

```
>>> x[0].shape # (1, 28, 28)
>>> x[1].shape # (1, 28, 28)
```

```
>>> x[0, 0] # 또는 x[0][0]
```

SECTION 07 합성곱 신경망(CNN)



7.4.2 im2col로 데이터 전개하기

이번 절에서는 for 문 대신 im2col이라는 편의 함수를 사용해 간단하게 구현해보겠다

그림 7-17 (대략적인) im2col의 동작

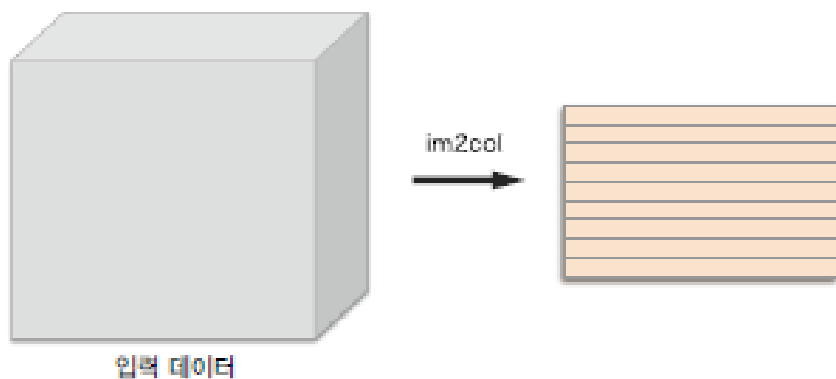


그림 7-18 필터 적용 영역을 앞에서부터 순서대로 1줄로 펼친다.

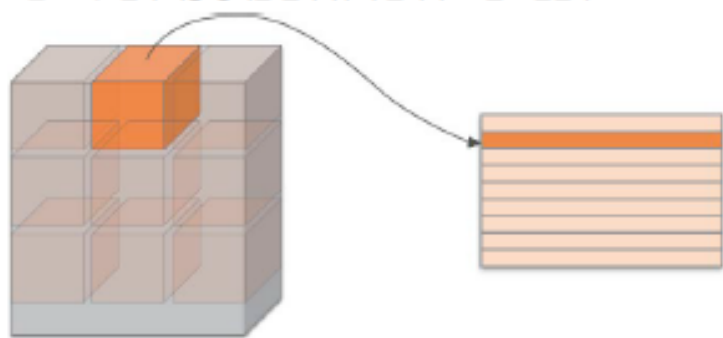
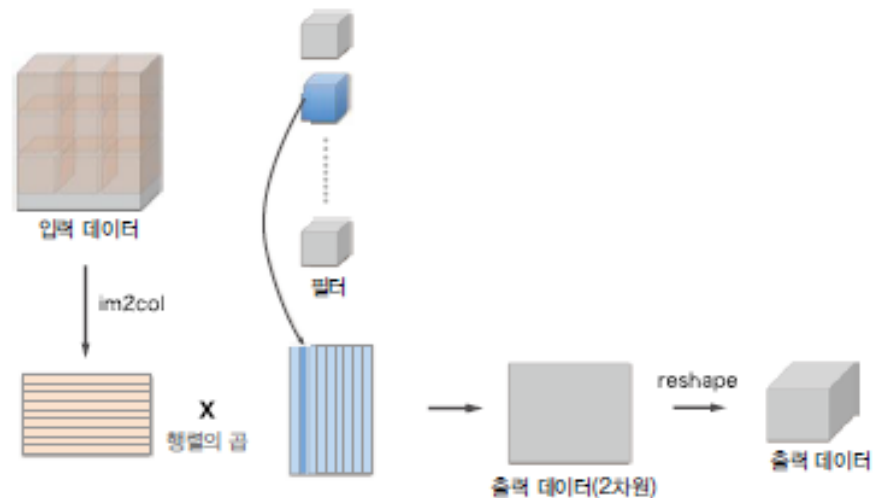


그림 7-19 합성곱 연산의 필터 처리 상세 과정: 필터를 세로로 1열로 전개하고, im2col이 전개한 데이터와 행렬 곱을 계산합니다. 마지막으로 출력 데이터를 변형(reshape)합니다





7.4.3 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

```
import sys, os
sys.path.append(os.pardir)
from common.util import im2col

x1 = np.random.rand(1, 3, 7, 7) # (데이터 수, 채널 수, 높이, 너비)
col1 = im2col(x1, 5, 5, stride=1, pad=0)
print(col1.shape) # (9, 75)

x2 = np.random.rand(10, 3, 7, 7) # 데이터 10개
col2 = im2col(x2, 5, 5, stride=1, pad=0)
print(col2.shape) # (90, 75)
```

SECTION 07 합성곱 신경망(CNN)



7.4.3 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

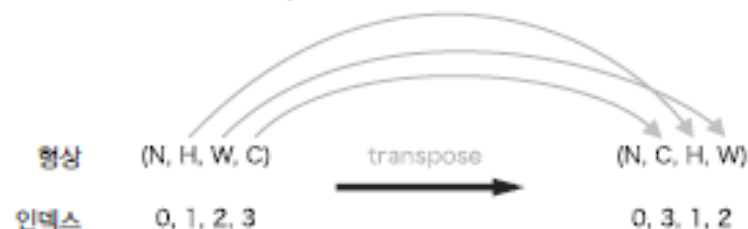
    def forward(self, x):
        FN, C, FH, FW = self.W.shape
        N, C, H, W = x.shape
        out_h = int(1 + (H + 2*self.pad - FH) / self.stride)
        out_w = int(1 + (W + 2*self.pad - FW) / self.stride)

        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
```

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



SECTION 07 합성곱 신경망(CNN)



7.4.4 풀링 계층 구현하기

그림 7-21 입력 데이터에 풀링 적용 영역을 전개 (2x2 풀링의 예)



```
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)

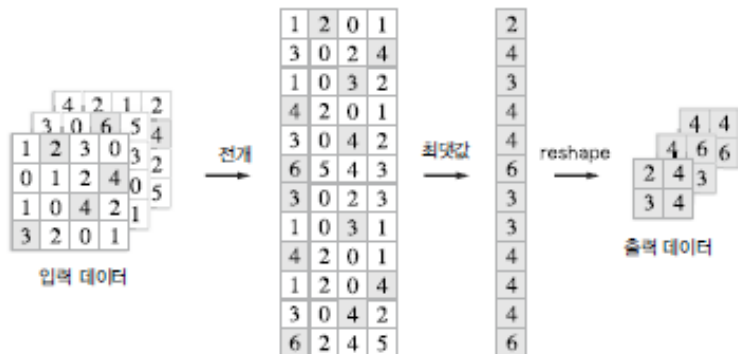
        # 전개 (1)
        col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
        col = col.reshape(-1, self.pool_h*self.pool_w)

        # 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
```

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



풀링 계층 구현은 [그림 7-22]와 같이 다음의 세 단계로 진행한다.

1. 입력 데이터를 전개한다.
2. 행별 최댓값을 구한다.
3. 적절한 모양으로 성형한다.

SECTION 07 합성곱 신경망(CNN)



7.5 CNN 구현하기

그림 7-23 단순한 CNN의 네트워크 구성



초기화 때 받는 인수

- input_dim - 입력 데이터(채널 수, 높이, 너비)의 차원
- conv_param - 합성곱 계층의 하이퍼파라미터(딕셔너리). 딕셔너리의 키는 다음과 같다.
 - filter_num - 필터 수
 - filter_size - 필터 크기
 - stride - 스트라이드
 - pad - 패딩
- hidden_size - 은닉층(완전연결)의 뉴런 수
- output_size - 출력층(완전연결)의 뉴런 수
- weight_init_std - 초기화 때의 가중치 표준편차

SECTION 07 합성곱 신경망(CNN)



7.6.1 1번째 층의 가중치 시각화하기

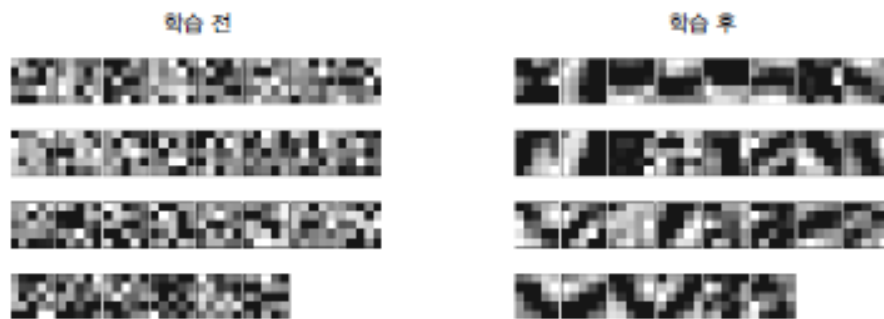


그림 7-25 가로 에지와 세로 에지에 반응하는 필터 : 출력 이미지 1은 세로 에지에 흰 픽셀이 나타나고, 출력 이미지 2는 가로 에지에 흰 픽셀이 많이 나온다.



SECTION 07 합성곱 신경망(CNN)



7.6.2 층 깊이에 따른 추출 정보 변화

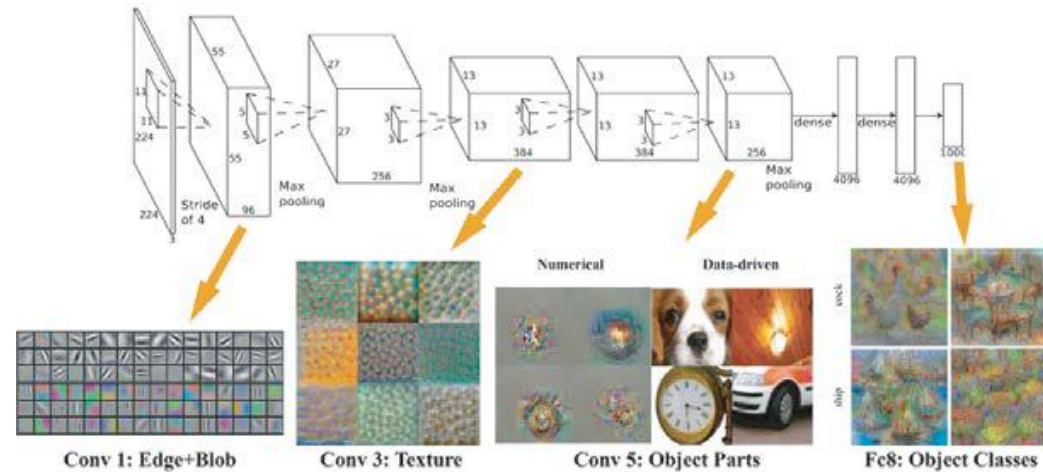


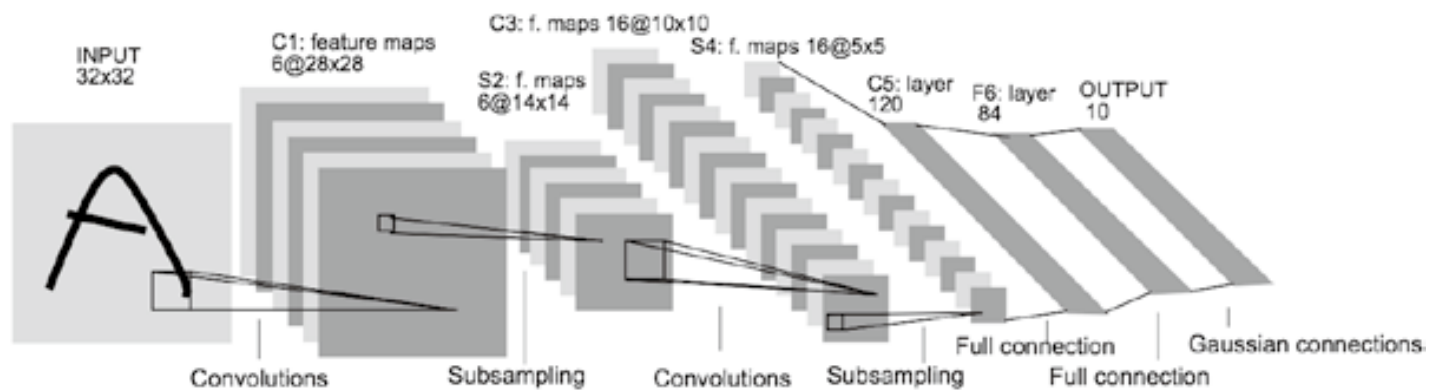
그림 7-26 CNN의 합성곱 계층에서 추출되는 정보. 1번째 층은 에지와 블롭, 3번째 층은 텍스처, 5번째 층은 사물의 일부, 마지막 완전연결 계층은 사물의 클래스(개, 자동차 등)에 뉴런이 반응한다. [19]

SECTION 07 합성곱 신경망(CNN)



7.7.1 LeNet

그림 7-27 LeNet의 구성^[20]

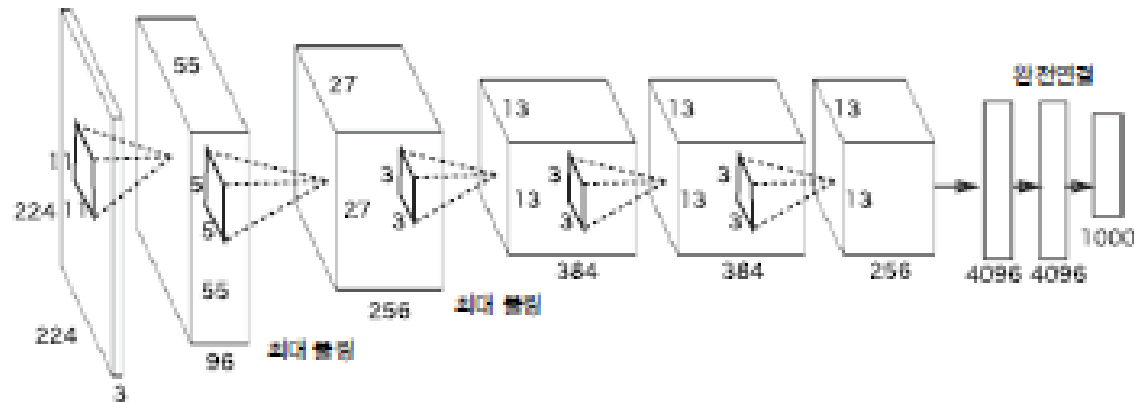


SECTION 07 합성곱 신경망(CNN)



7.7.2 AlexNet

그림 7-28 AlexNet의 구성²¹⁾



LeNet과 비교해 훨씬 최근인 2012년에 발표된 AlexNet은 딥러닝 열풍을 일으키는 데 큰 역할을 했다

- 활성화 함수로 ReLU를 이용한다.
- LRN_{Local Response Normalization}이라는 국소적 정규화를 실시하는 계층을 이용한다.
- 드롭아웃을 사용한다.