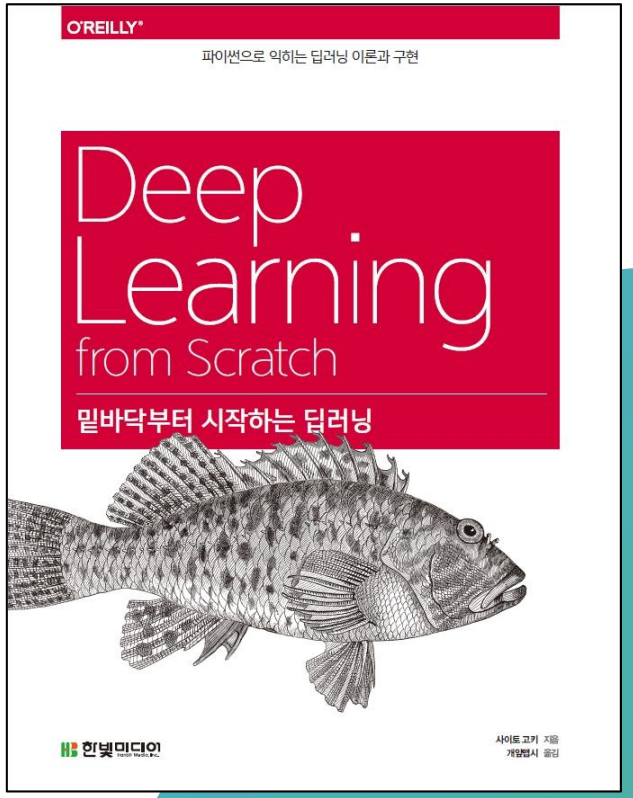
CHAPTER 7 합성곱 신경망(CNN)

밑바닥부터 시작하는 딥러닝



이 책의 학습 목표

- CHAPTER 1 파이썬에 대해 간략하게 살펴보고 사용법 익히기
- CHAPTER 2 퍼셉트론에 대해 알아보고 퍼셉트론을 써서 간단한 문제를 풀어보기
- CHAPTER 3 신경망의 개요, 입력 데이터가 무엇인지 신경망이 식별하는 처리 과정 알아보기
- CHAPTER 4 손실 함수의 값을 가급적 작게 만드는 경사법에 대해 알아보기
- CHAPTER 5 가중치 매개변수의 기울기를 효율적으로 계산하는 오차역전파법 배우기
- CHAPTER 6 신경망(딥러닝) 학습의 효율과 정확도를 높이기
- **CHAPTER 7 CNN의 메커니즘을 자세히 설명하고 파이썬으로 구현하기**
- CHAPTER 8 딥러닝의 특징과 과제, 가능성, 오늘날의 첨단 딥러닝에 대해 알아보기

Contents

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 계층convolutional layer과 풀링 계층pooling layer이 새롭게 등장한다

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

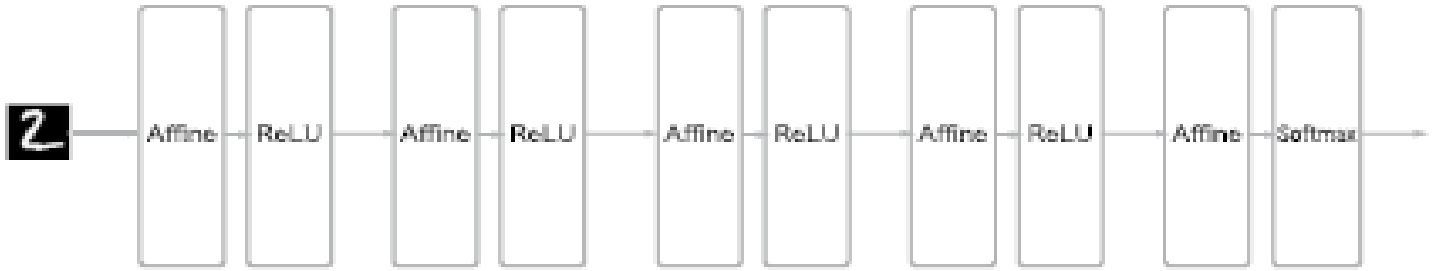
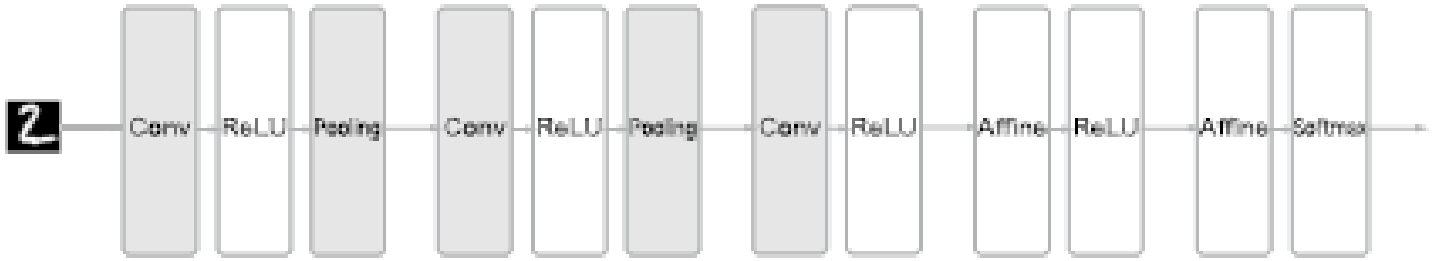


그림 7-2 CNN으로 이뤄진 네트워크의 예 : 합성곱 계층과 풀링 계층이 새로 추가(회색)



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 데이터를 동등한 뉴런(같은 차원의뉴런)으로 취급하여 형상에 담긴 정보를 살릴 수 없다.

한편, 합성곱 계층은 형상을 유지한다.
CNN에서는 합성곱 계층의 입출력 데이터를 특징 맵feature map이라고도 한다.
합성곱 계층의 입력 데이터를 입력 특징 맵input feature map, 출력 데이터를 출력 특징 맵output feature map이라고 하는 식이다.
이 책에서는 '입출력 데이터'와 '특징 맵'을 같은 의미로 사용한다

7.2.2 합성곱 연산

그림 **7-3** 합성곱 연산의 예 : 합성곱 연산을 기호로 표기

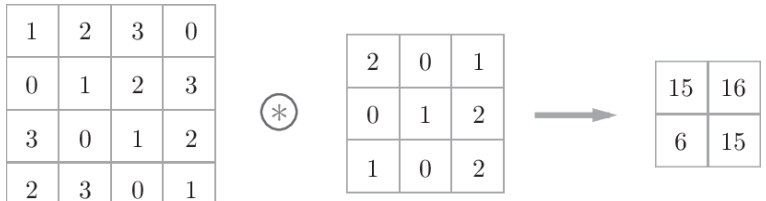
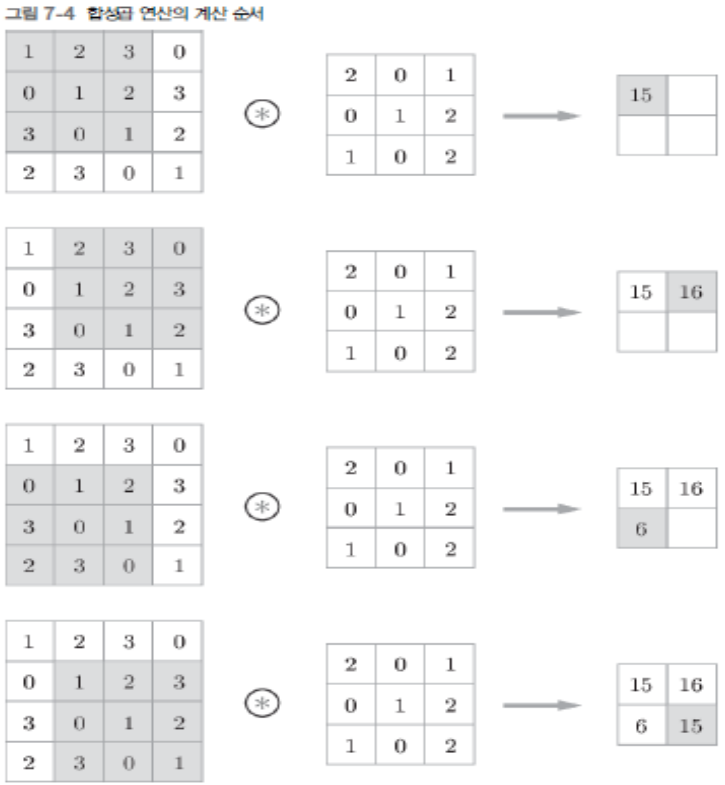


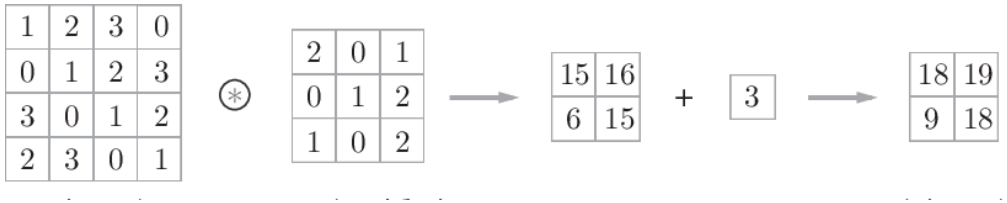
그림 7-4 합성곱 연산의 계산 순서



합성곱 연산은 필터(혹은 커널)의 윈도우window를 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-누산fusedmultiply-add, FMA이라 한다).*
그리고 그 결과를 출력의 해당 장소에 저장

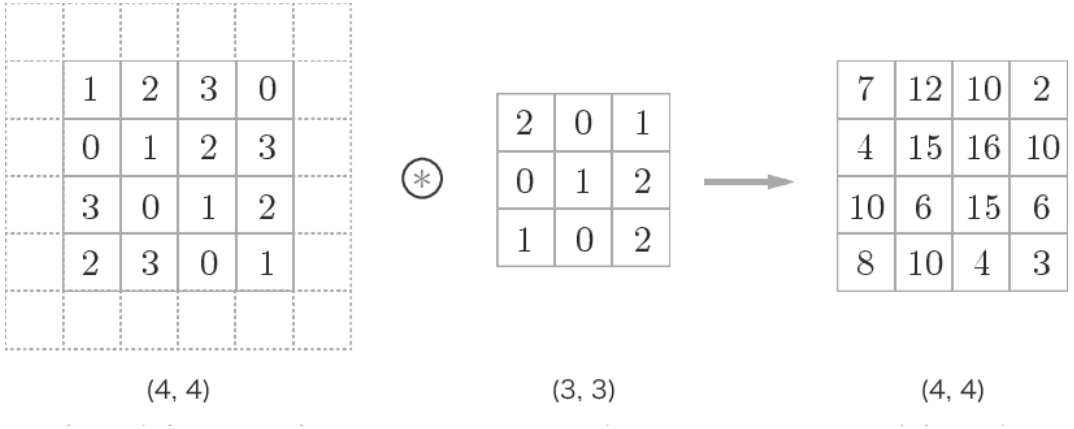
그림 **7-5** 합성곱 연산의 편향 : 필터를 적용한 원소에 고정값(편향)을 더한다



7.2.3 패딩 (padding)

합성곱 연산을 수행하기 전에 입력 데이터 주변을 특정 값(예컨대 0)으로 채우기도 란다. 이를 패딩padding이라 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).

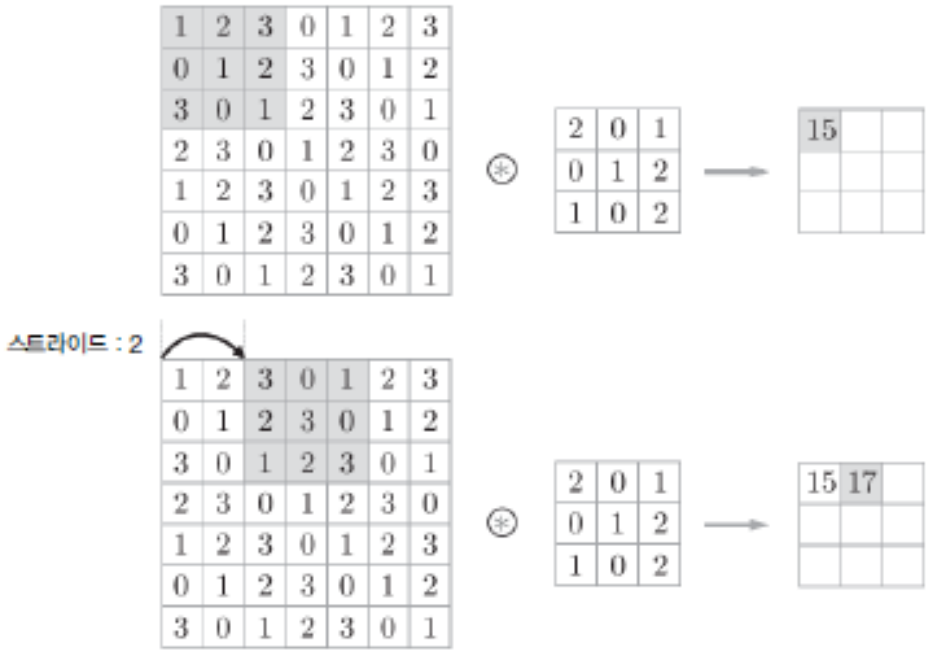


SECTION 07 합성곱 신경망(CNN)

7.2.4 스트라이드(stride 이동 폭)

필터를 적용하는 위치의 간격을 스트라이드stride라고 한다

그림 7-7 스트라이드가 2인 합성곱 연산



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

$$OW = \frac{W + 2P - FW}{S} + 1 \qquad [식 7.1]$$

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)

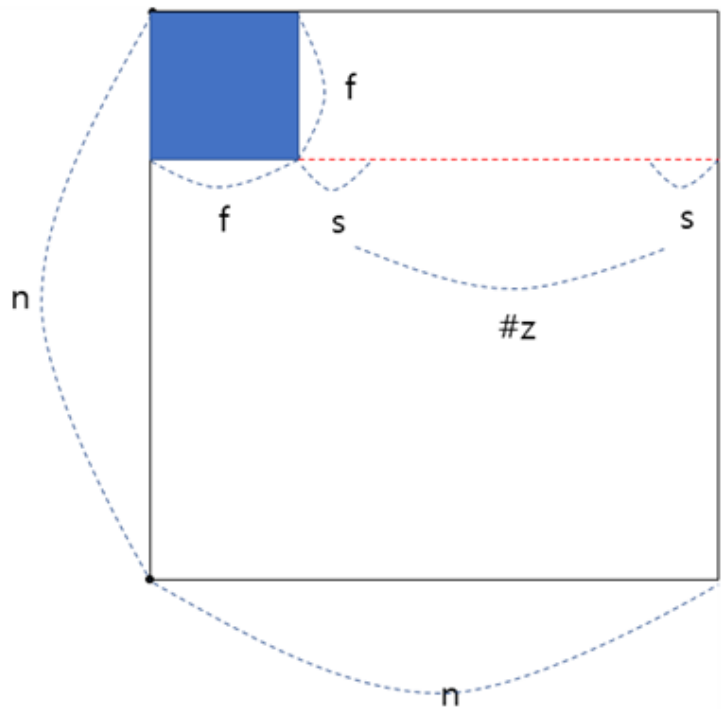
$$OH = \frac{28+2\cdot 2-5}{3} + 1 = 10$$

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

예제 9-3-2. $n \times n$ 크기의 입력에 패딩 p와 이동폭(stride) s를 사용하여 $f \times f$ 크기의 필터를 적용하였다. 필터 적용결과 얻게 되는 특징 맵의 크기를 구하라.

풀이) 먼저 패딩을 적용하지 않고 $n \times n$ 입력에 이동폭 s를 사용하여 $f \times f$ 필터를 $z+1$ 번 적용한 경우 다음 그림과 같이 $n = f + s \times z$이 된다. 즉, 특징 맵 크기는 $(\frac{n-f}{s}+1) \times (\frac{n-f}{s}+1)$가 된다.

이제, 패딩 p를 사용하면, 입력의 크기가 $(n+2p) \times (n+2p)$이 되고, 특징 맵 크기는 $(\frac{n+2p-f}{s}+1) \times (\frac{n+2p-f}{s}+1)$ 이 된다.

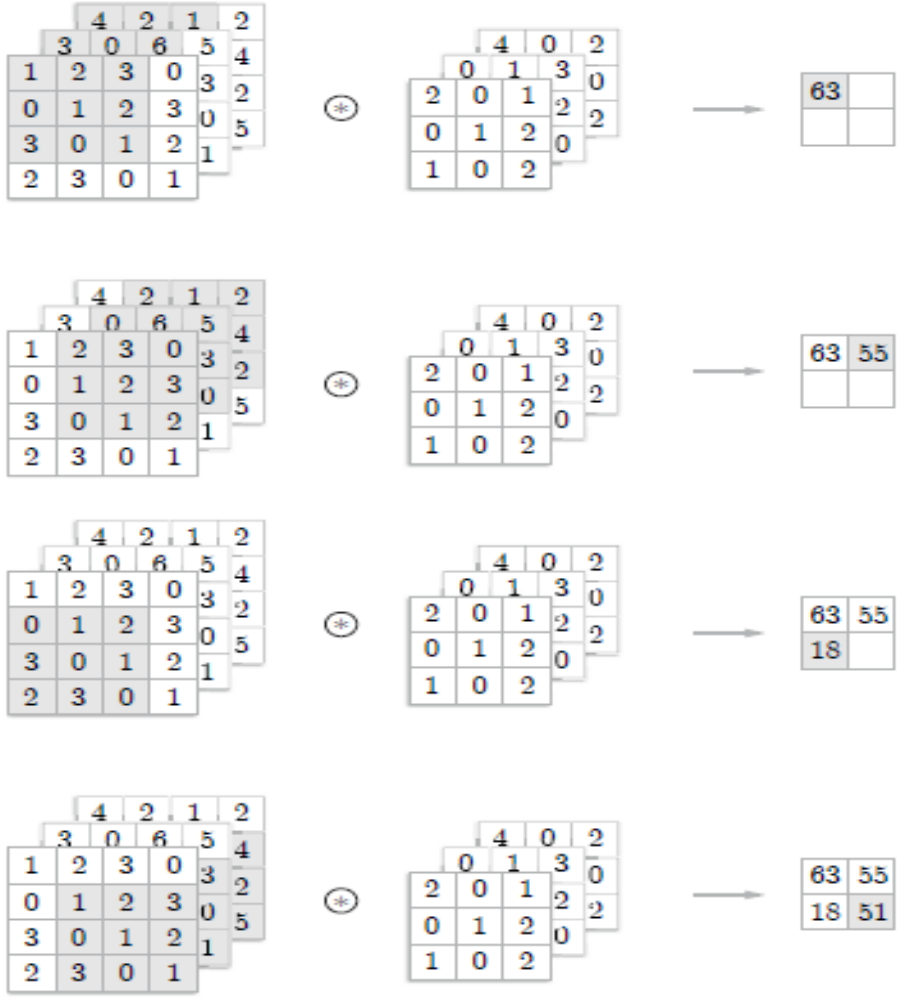


7.2.5 3차원 데이터의 합성곱 연산

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



7.2.6 블록으로 생각하기

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

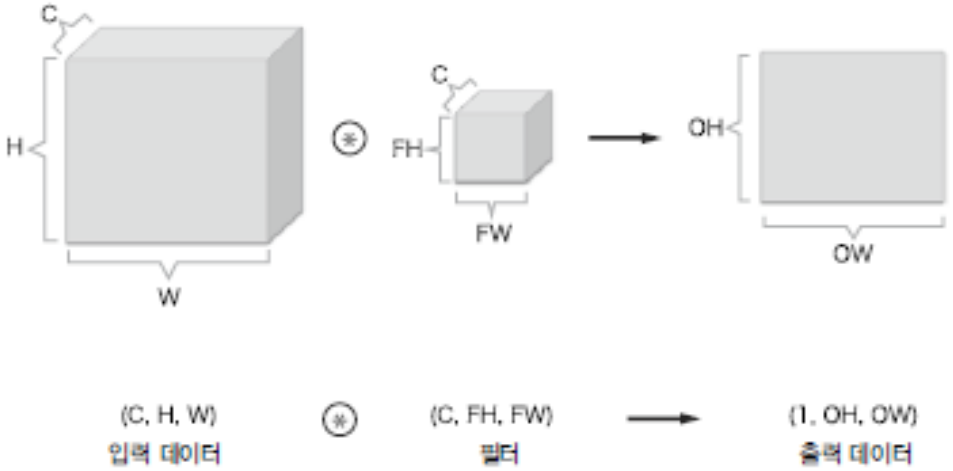


그림 7-11 여러 필터를 사용한 합성곱 연산의 예

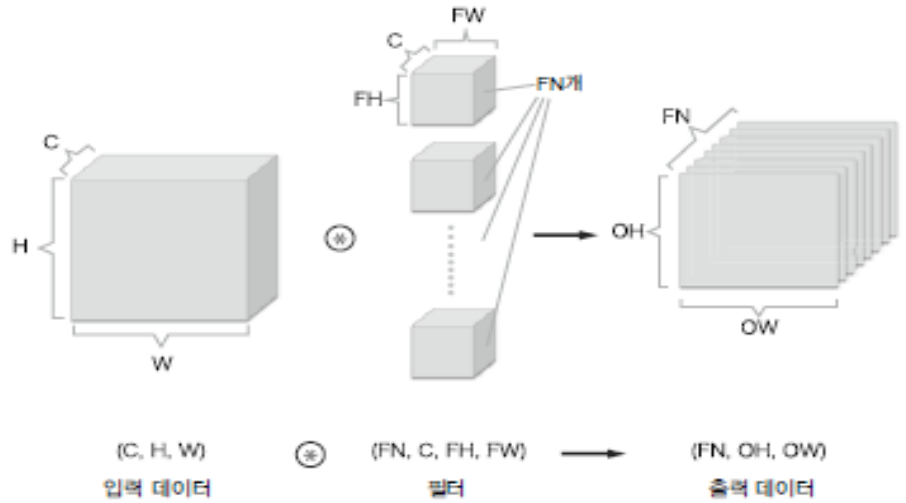
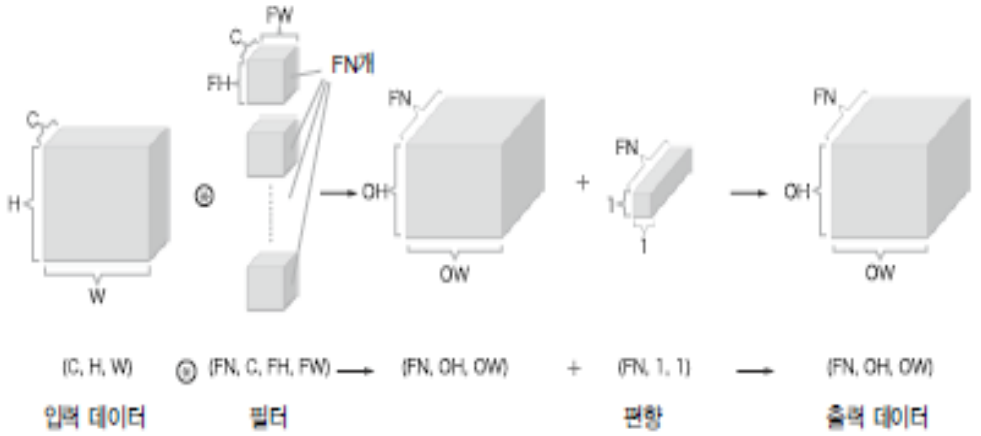


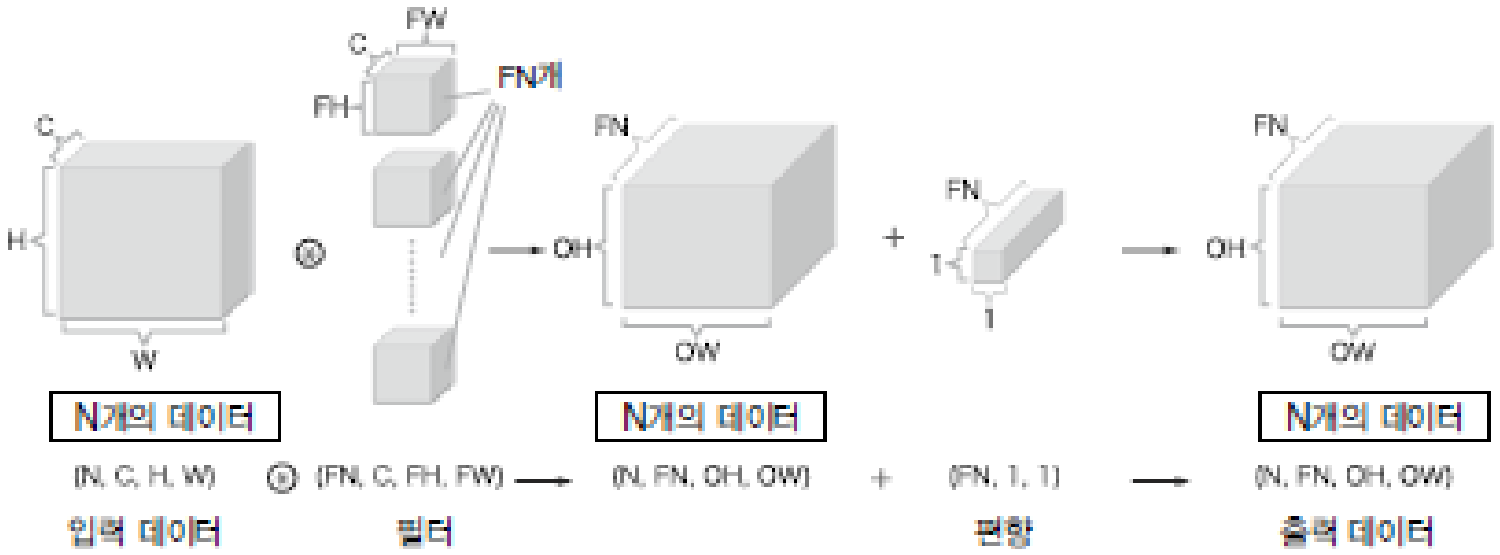
그림 7-12 합성곱 연산의 처리 흐름(편향 추가)



형상이 다른 블록의 덧셈은 numpy의 broadcasting 기능으로 구현

7.2.7 배치 처리

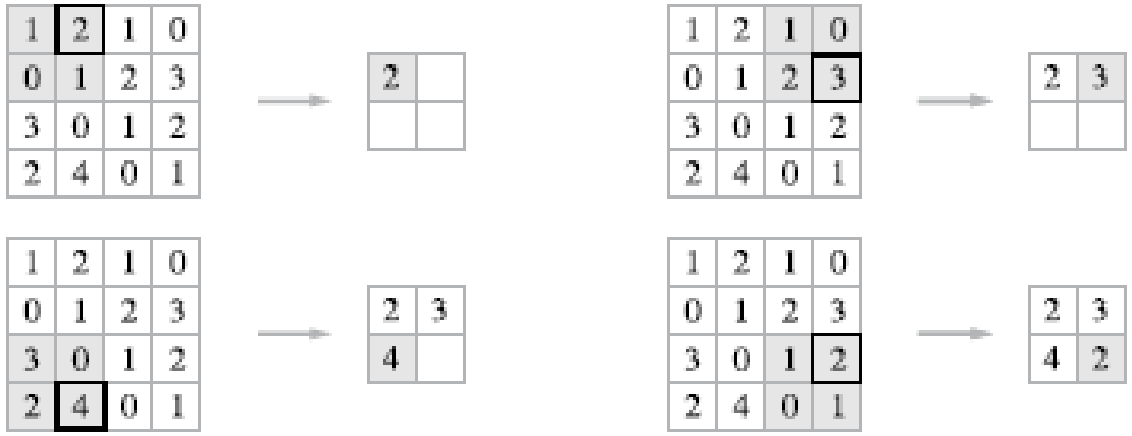
그림 7-13 합성곱 연산의 처리 흐름(배치 처리)



각 계층을 흐르는 데이터의 차원을 하나 늘려 4차원 데이터로 저장 (데이터수, 체널수, 높이, 너비)

7.3 풀링 계층

그림 7-14 최대 풀링의 처리 순서



풀링의 윈도우 크기와 스트라이드는 같은 값으로 설정하는 것이 보통임

7.3.1 풀링 계층의 특징

학습해야 할 매개변수가 없다

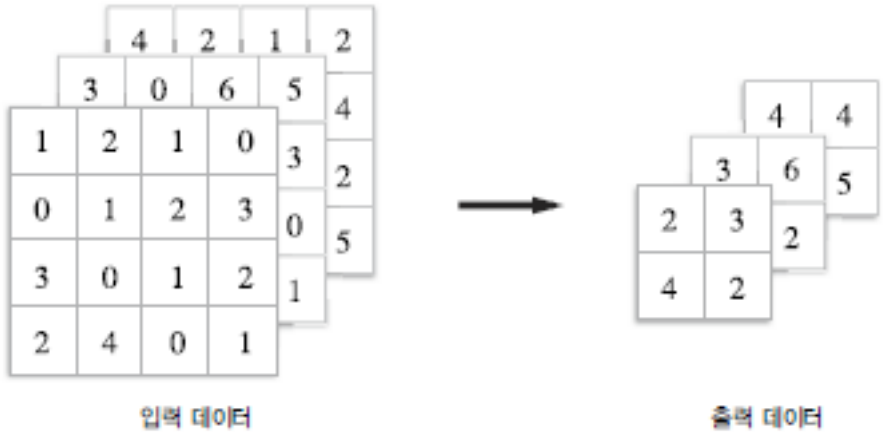
풀링 계층은 합성곱 계층과 달리 학습해야 할 매개변수가 없다.
풀링은 대상 영역에서최댓값이나 평균을 취하는 명확한 처리이므로 특별히 학습할 것이 없다.

채널 수가 변하지 않는다

풀링 연산은 입력 데이터의 채널 수 그대로 출력 데이터로 내보낸다.

[그림 7-15]처럼 채널마다 독립적으로 계산하기 때문이다.

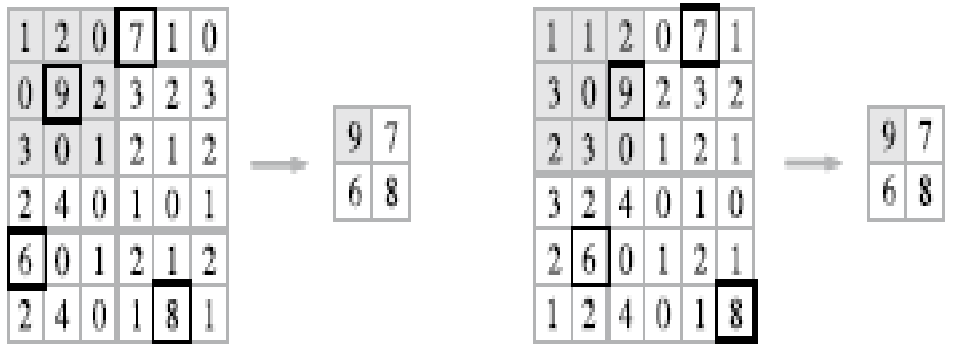
그림 7-15 풀링은 채널 수를 바꾸지 않는다.



입력의 변화에 영향을 적게 받는다(강건하다)

입력 데이터가 조금 변해도 풀링의 결과는 잘 변하지 않는다.
예를 들어 [그림 7-16]은입력 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).



7.4 합성곱/풀링 계층 구현하기

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]
```

7.4.2 im2col로 데이터 전개하기

이번 절에서는 for 문 대신 **im2col**이라는 편의 함수를 사용해 간단하게 구현해보겠다

그림 7-17 (대략적인) im2col의 동작

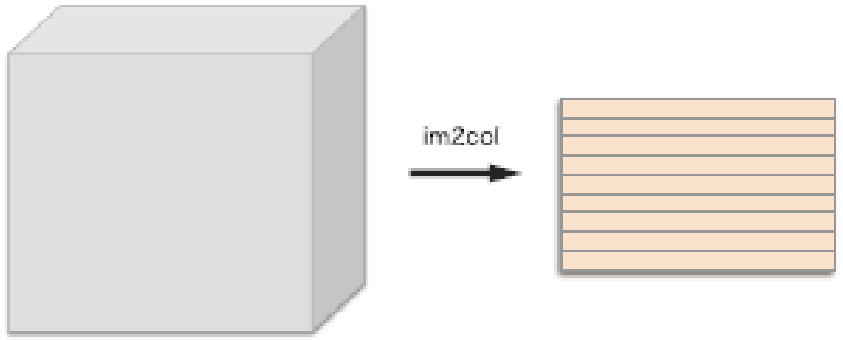
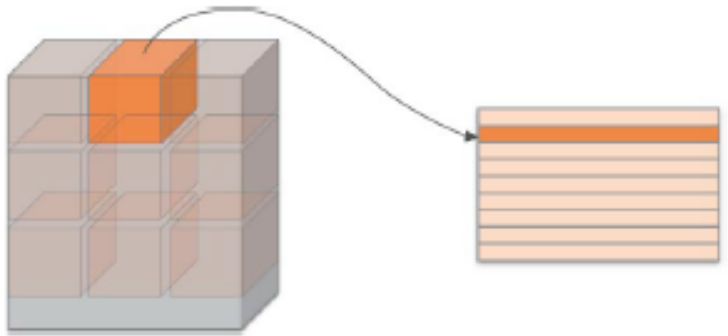


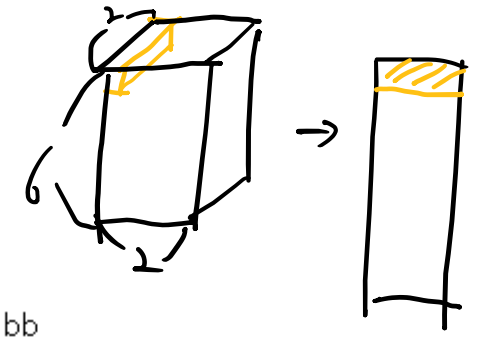
그림 7-18 필터 적용 영역을 앞에서부터 순서대로 1줄로 펼친다.



ch07/im2coltest.py

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

aa=np.arange(48).reshape(2,2,6,2)


bb=im2col(aa,1,1)
```

```
>>> aa
array([[[[ 0,  1],
         [ 2,  3],
         [ 4,  5],
         [ 6,  7],
         [ 8,  9],
         [10, 11]],

        [[12, 13],
         [14, 15],
         [16, 17],
         [18, 19],
         [20, 21],
         [22, 23]]],


       [[[24, 25],
         [26, 27],
         [28, 29],
         [30, 31],
         [32, 33],
         [34, 35]],

        [[36, 37],
         [38, 39],
         [40, 41],
         [42, 43],
         [44, 45],
         [46, 47]]]])
```



```
>>> bb
array([[ 0., 12.],
       [ 1., 13.],
       [ 2., 14.],
       [ 3., 15.],
       [ 4., 16.],
       [ 5., 17.],
       [ 6., 18.],
       [ 7., 19.],
       [ 8., 20.],
       [ 9., 21.],
       [10., 22.],
       [11., 23.],
       [24., 36.],
       [25., 37.],
       [26., 38.],
       [27., 39.],
       [28., 40.],
       [29., 41.],
       [30., 42.],
       [31., 43.],
       [32., 44.],
       [33., 45.],
       [34., 46.],
       [35., 47.]])
```

7.4.2 im2col로 데이터 전개하기

이번 절에서는 for 문 대신 **im2col**이라는 편의 함수를 사용해 간단하게 구현해보겠다

그림 7-17 (대략적인) im2col의 동작

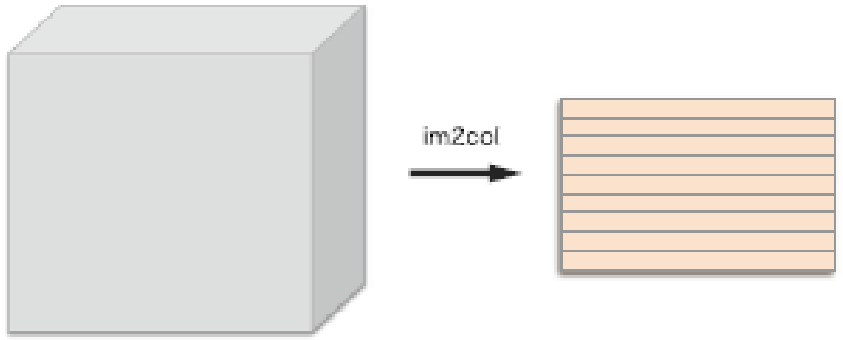
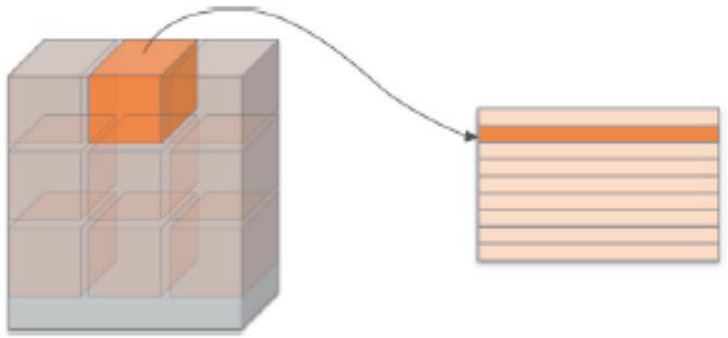


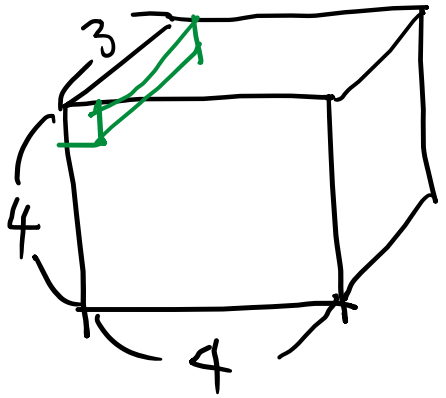
그림 7-18 필터 적용 영역을 앞에서부터 순서대로 1줄로 펼친다.

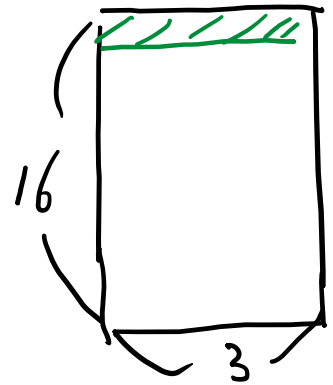


```
x1=np.arange(48).reshape(1,3,4,4)
col1=im2col(x1,1,1)
```

```
>>> x1
array([[[[ 0,  1,  2,  3],
         [ 4,  5,  6,  7],
         [ 8,  9, 10, 11],
         [12, 13, 14, 15]],

        [[16, 17, 18, 19],
         [20, 21, 22, 23],
         [24, 25, 26, 27],
         [28, 29, 30, 31]],

        [[32, 33, 34, 35],
         [36, 37, 38, 39],
         [40, 41, 42, 43],
         [44, 45, 46, 47]]]])
```

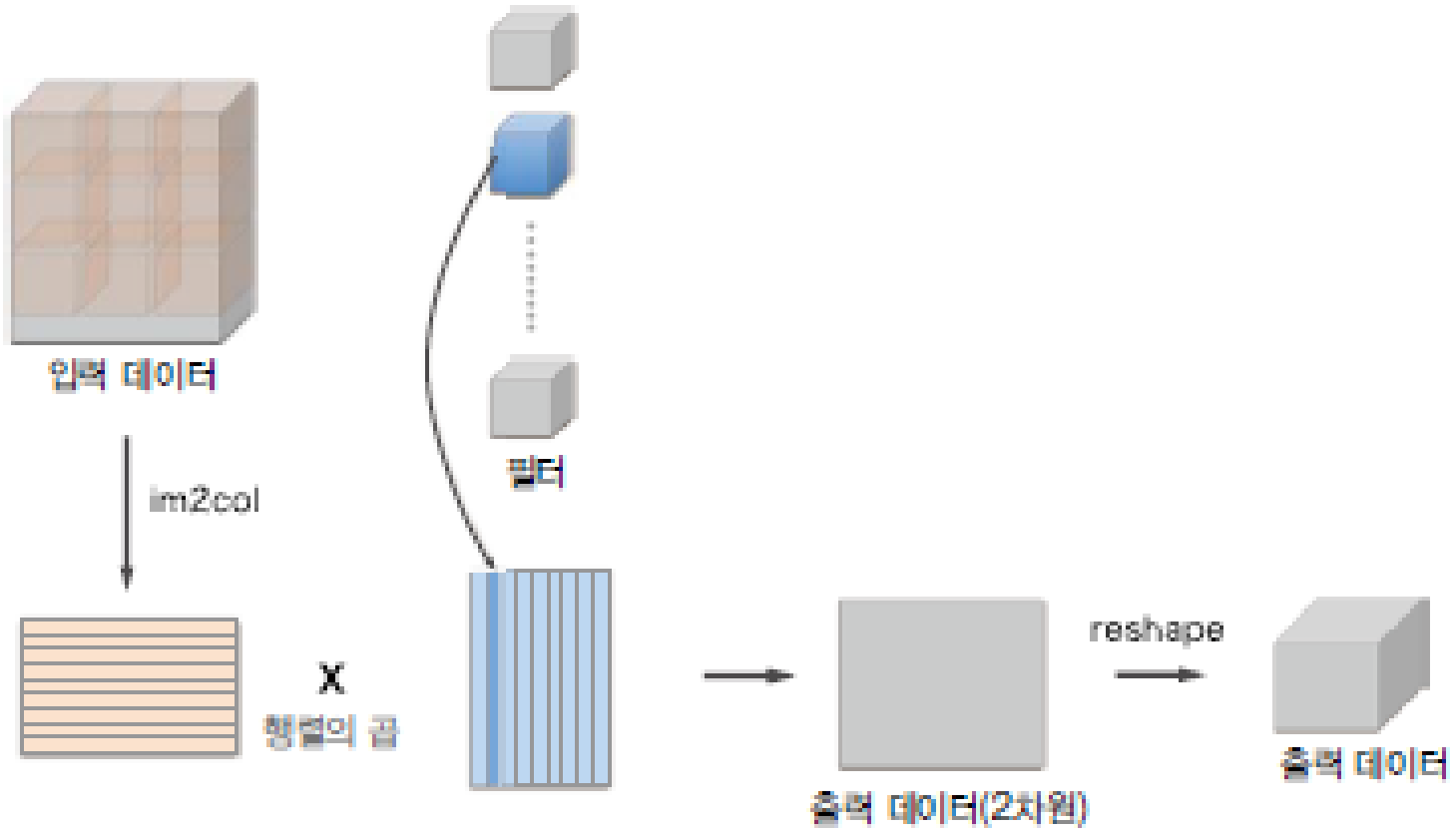
```
>>> col1
array([[ 0., 16., 32.],
       [ 1., 17., 33.],
       [ 2., 18., 34.],
       [ 3., 19., 35.],
       [ 4., 20., 36.],
       [ 5., 21., 37.],
       [ 6., 22., 38.],
       [ 7., 23., 39.],
       [ 8., 24., 40.],
       [ 9., 25., 41.],
       [10., 26., 42.],
       [11., 27., 43.],
       [12., 28., 44.],
       [13., 29., 45.],
       [14., 30., 46.],
       [15., 31., 47.]])
```





7.4.2 im2col로 데이터 전개하기

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)
```

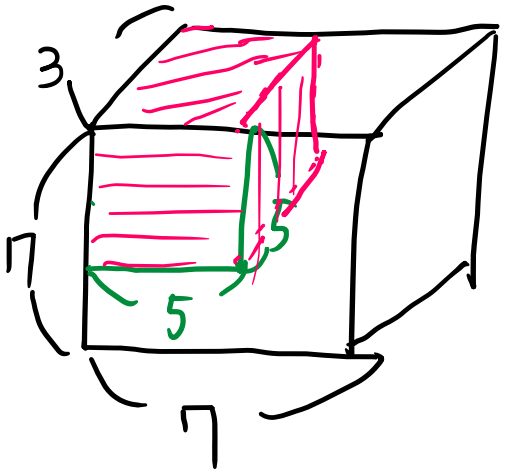
* input_data – (데이터 수, 채널 수, 높이, 너비)의 4차원 배열로 이뤄진 입력 데이터
* filter_h – 필터의 높이
* filter_w – 필터의 너비
* stride – 스트라이드
* pad – 패딩

ch07/im2coltest.py

```
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)
```

↓ x1의 10배

• 7x7에 5x5가 3x3x3 = 9개 나옴



3x5x5=75

7.4.3 합성곱 계층 구현하기

common/layer.py

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

        # 중간 데이터 (backward 시 사용)
        self.x = None
        self.col = None
        self.col_W = None

        # 가중치와 편향 매개변수의 기울기
        self.dW = None
        self.db = None

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

        col = im2col(x, FH, FW, self.stride, self.pad)
        col_W = self.W.reshape(FN, -1).T

        out = np.dot(col, col_W) + self.b
        out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)

        self.x = x
        self.col = col
        self.col_W = col_W

        return out
```

FN이 결정되었기에 나머지는 알아서 선택

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

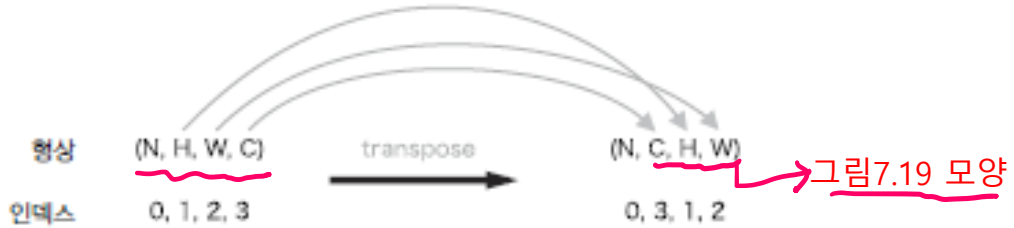


그림7.19 모양

common/layer.py

7.4.4 풀링 계층 구현하기

풀링의 경우 채널 쪽이 독립

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

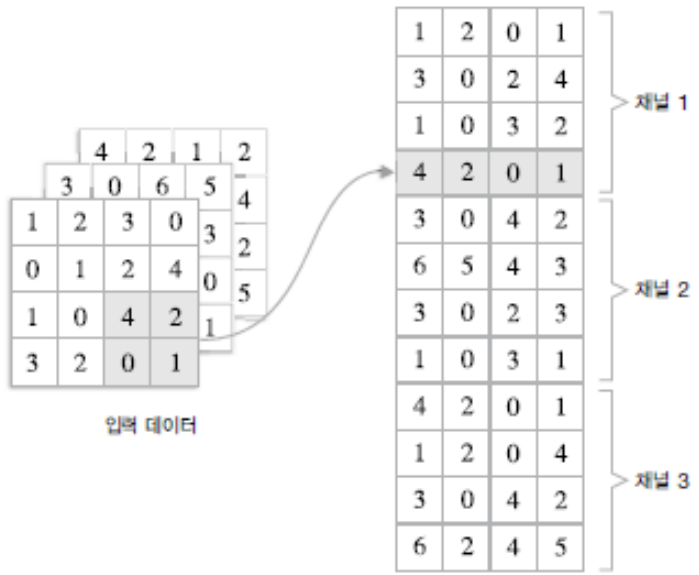
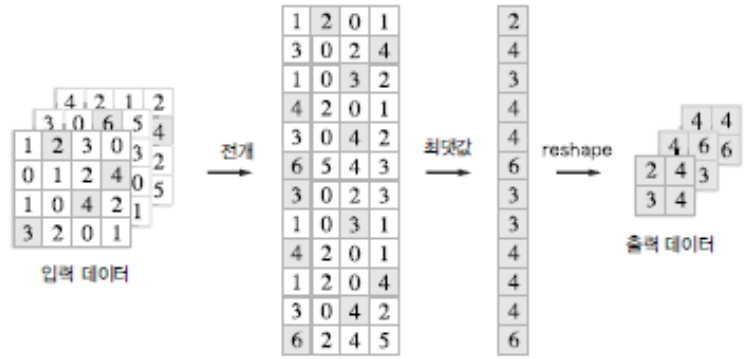


그림 7-22 풀링 계층 구현의 흐름 : 풀링 적용 영역에서 가장 큰 원소는 회색으로 표시




```
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

        self.x = None
        self.arg_max = None

    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)

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

        self.x = x
        self.arg_max = arg_max

        return out
```

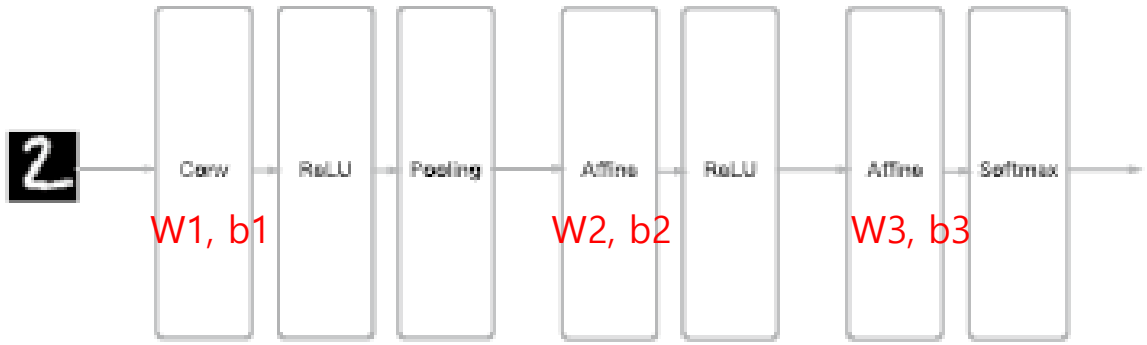
Pooling size 만큼만 한 줄에 있도록..

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

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

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

7.5 CNN 구현하기

ch07/simple_convnet.py

```
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import pickle
import numpy as np
from collections import OrderedDict
from common.layers import *
from common.gradient import numerical_gradient


class SimpleConvNet:
    """단순한 합성곱 신경망

    conv - relu - pool - affine - relu - affine - softmax

    Parameters
    ----------
    input_size : 입력 크기 (MNIST의 경우엔 784)
    hidden_size_list : 각 은닉층의 뉴런 수를 담은 리스트 (e.g. [100, 100, 100])
    output_size : 출력 크기 (MNIST의 경우엔 10)
    activation : 활성화 함수 - 'relu' 혹은 'sigmoid'
    weight_init_std : 가중치의 표준편차 지정 (e.g. 0.01)
        'relu'나 'he'로 지정하면 'He 초깃값'으로 설정
        'sigmoid'나 'xavier'로 지정하면 'Xavier 초깃값'으로 설정
    """
    def __init__(self, input_dim=(1, 28, 28),
                 conv_param={'filter_num':30, 'filter_size':5, 'pad':0, 'stride':1},
                 hidden_size=100, output_size=10, weight_init_std=0.01):
        filter_num = conv_param['filter_num']
        filter_size = conv_param['filter_size']
        filter_pad = conv_param['pad']
        filter_stride = conv_param['stride']
        input_size = input_dim[1]
        conv_output_size = (input_size - filter_size + 2*filter_pad) / filter_stride + 1
        pool_output_size = int(filter_num * (conv_output_size/2) * (conv_output_size/2))
```

●● conv_param - 합성곱 계층의 하이퍼파라미터(딕셔너리). 딕셔너리의 키는 다음과 같다.
- filter_num - 필터 수
- filter_size - 필터 크기
- stride - 스트라이드
- pad - 패딩

7.5 CNN 구현하기

ch07/simple_convnet.py

```
# 가중치 초기화
self.params = {}
self.params['W1'] = weight_init_std * ₩
                    np.random.randn(filter_num, input_dim[0], filter_size, filter_size)
self.params['b1'] = np.zeros(filter_num)
self.params['W2'] = weight_init_std * ₩
                    np.random.randn(pool_output_size, hidden_size)
self.params['b2'] = np.zeros(hidden_size)
self.params['W3'] = weight_init_std * ₩
                    np.random.randn(hidden_size, output_size)
self.params['b3'] = np.zeros(output_size)

# 계층 생성
self.layers = OrderedDict()
self.layers['Conv1'] = Convolution(self.params['W1'], self.params['b1'],
                                   conv_param['stride'], conv_param['pad'])
self.layers['Relu1'] = Relu()
self.layers['Pool1'] = Pooling(pool_h=2, pool_w=2, stride=2)
self.layers['Affine1'] = Affine(self.params['W2'], self.params['b2'])
self.layers['Relu2'] = Relu()
self.layers['Affine2'] = Affine(self.params['W3'], self.params['b3'])

self.last_layer = SoftmaxWithLoss()
```

7.5 CNN 구현하기

ch07/simple_convnet.py

```
def predict(self, x):
    for layer in self.layers.values():
        x = layer.forward(x)

    return x

def loss(self, x, t):
    """손실 함수를 구한다.

    Parameters
    ----------
    x : 입력 데이터
    t : 정답 레이블
    """
    y = self.predict(x)
    return self.last_layer.forward(y, t)
```

7.5 CNN 구현하기

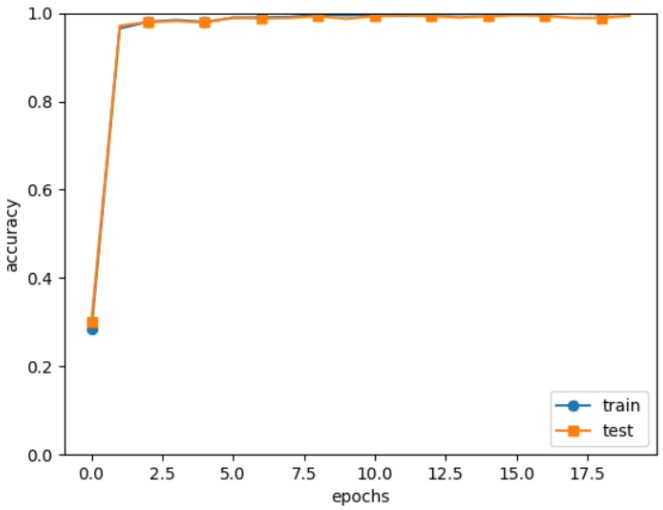
ch07/simple_convnet.py

```
def gradient(self, x, t):
    """기울기를 구한다(오차역전파법).

    Parameters
    ----------
    x : 입력 데이터
    t : 정답 레이블

    Returns
    -------
    각 층의 기울기를 담은 사전(dictionary) 변수
        grads['W1']、grads['W2']、... 각 층의 가중치
        grads['b1']、grads['b2']、... 각 층의 편향
    """
    # forward
    self.loss(x, t)

    # backward
    dout = 1
    dout = self.last_layer.backward(dout)

    layers = list(self.layers.values())
    layers.reverse()
    for layer in layers:
        dout = layer.backward(dout)

    # 결과 저장
    grads = {}
    grads['W1'], grads['b1'] = self.layers['Conv1'].dW, self.layers['Conv1'].db
    grads['W2'], grads['b2'] = self.layers['Affine1'].dW, self.layers['Affine1'].db
    grads['W3'], grads['b3'] = self.layers['Affine2'].dW, self.layers['Affine2'].db

    return grads
```

```
def save_params(self, file_name="params.pkl"):
    params = {}
    for key, val in self.params.items():
        params[key] = val
    with open(file_name, 'wb') as f:
        pickle.dump(params, f)

def load_params(self, file_name="params.pkl"):
    with open(file_name, 'rb') as f:
        params = pickle.load(f)
    for key, val in params.items():
        self.params[key] = val

    for i, key in enumerate(['Conv1', 'Affine1', 'Affine2']):
        self.layers[key].W = self.params['W' + str(i+1)]
        self.layers[key].b = self.params['b' + str(i+1)]
```

7.5 CNN 구현하기

ch07/train_convnet.py




```
test acc:0.987
```

```python
import numpy as np
import matplotlib.pyplot as plt
from dataset.mnist import load_mnist
from simple_convnet import SimpleConvNet
from common.trainer import Trainer

# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(flatten=False)

# 시간이 오래 걸릴 경우 데이터를 줄인다.
#x_train, t_train = x_train[:5000], t_train[:5000]
#x_test, t_test = x_test[:1000], t_test[:1000]

max_epochs = 20

network = SimpleConvNet(input_dim=(1,28,28),
                        conv_param = {'filter_num': 30, 'filter_size': 5, 'pad': 0, 'stride': 1},
                        hidden_size=100, output_size=10, weight_init_std=0.01)

trainer = Trainer(network, x_train, t_train, x_test, t_test,
                  epochs=max_epochs, mini_batch_size=100,
                  optimizer='Adam', optimizer_param={'lr': 0.001},
                  evaluate_sample_num_per_epoch=1000)
trainer.train()

# 매개변수 보존
network.save_params("params.pkl")
print("Saved Network Parameters!")

# 그래프 그리기
markers = {'train': 'o', 'test': 's'}
x = np.arange(max_epochs)
plt.plot(x, trainer.train_acc_list, marker='o', label='train', markevery=2)
plt.plot(x, trainer.test_acc_list, marker='s', label='test', markevery=2)
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.legend(loc='lower right')
plt.show()
```

SECTION 07 합성곱 신경망(CNN)

7.6 CNN 시각화하기

7.6.1 1번째 층의 가중치 시각화하기

```
import numpy as np
import matplotlib.pyplot as plt
from simple_convnet import SimpleConvNet

def filter_show(filters, nx=8, margin=3, scale=10):
    """
    c.f. https://gist.github.com/aidiary/07d530d5e08011832b12#file-draw_weight-py
    """
    FN, C, FH, FW = filters.shape
    ny = int(np.ceil(FN / nx))

    fig = plt.figure()
    fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05)

    for i in range(FN):
        ax = fig.add_subplot(ny, nx, i+1, xticks=[], yticks=[])
        ax.imshow(filters[i, 0], cmap=plt.cm.gray_r, interpolation='nearest')
    plt.show()


network = SimpleConvNet()
# 무작위(랜덤) 초기화 후의 가중치
filter_show(network.params['W1'])

# 학습된 가중치
network.load_params("params.pkl")
filter_show(network.params['W1'])
```

(30,1,5,5)

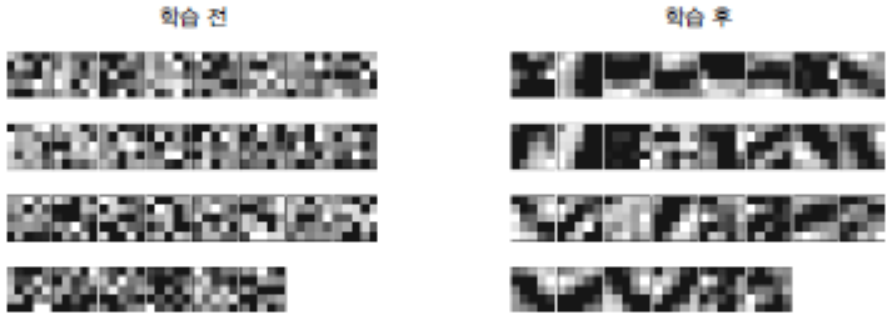
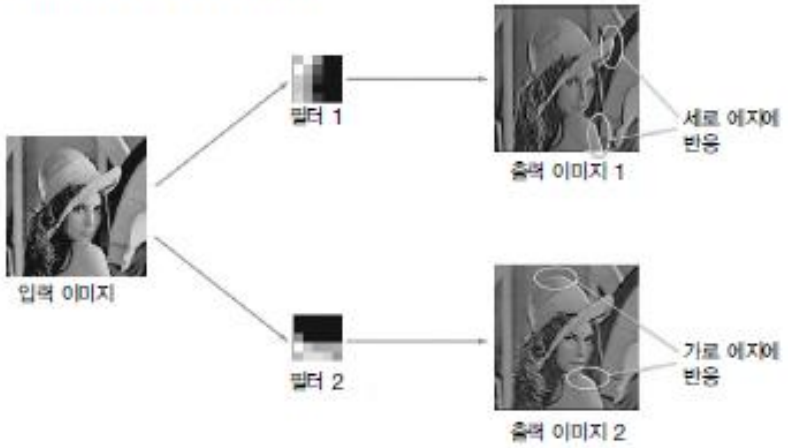


그림 7-25 가로 에지와 세로 에지에 반응하는 필터 : 출력 이미지 1은 세로 에지에 흰 픽셀이 나타나고, 출력 이미지 2는 가로 에지에 흰 픽셀이 많이 나온다.



7.6.2 층 깊이에 따른 추출 정보 변화

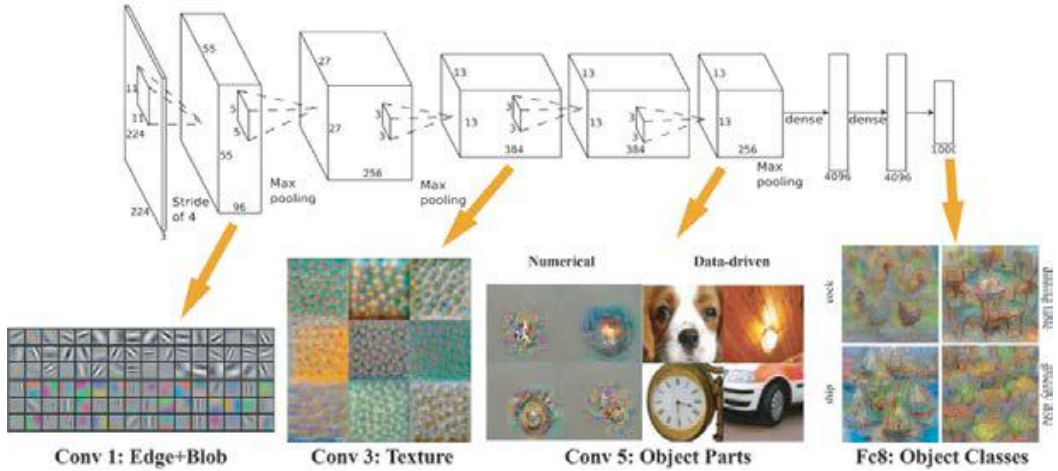
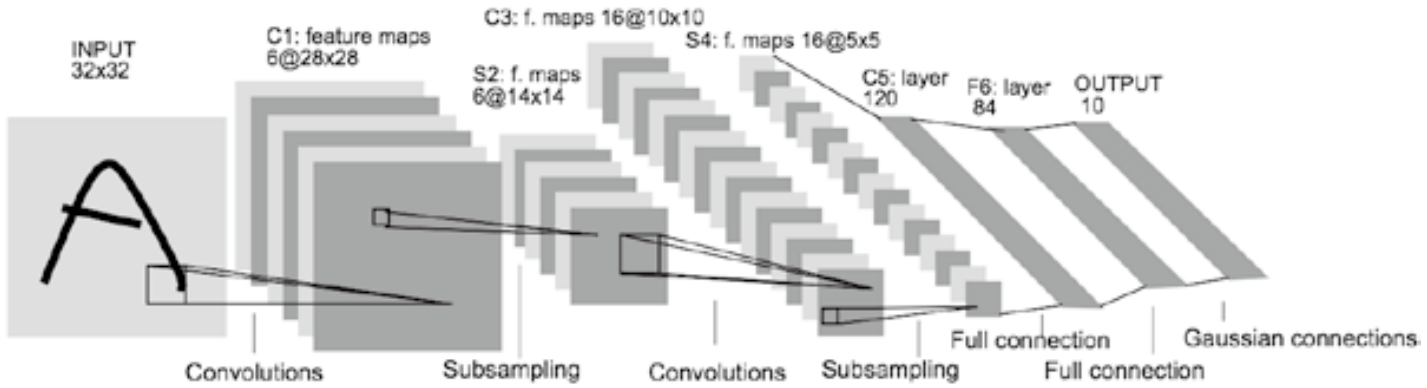


그림 7-26 CNN의 합성곱 계층에서 추출되는 정보. 1번째 층은 에지와 블롭, 3번째 층은 텍스처, 5번째 층은 사물의일부, 마지막 완전연결 계층은 사물의 클래스(개, 자동차 등)에 뉴런이 반응한다. [19]

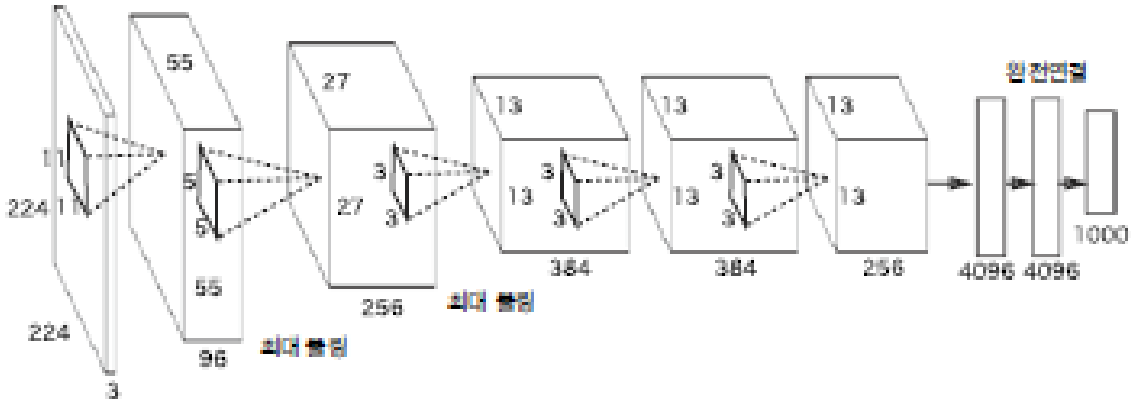
7.7.1 LeNet

그림 7-27 LeNet의 구성[20]



7.7.2 AlexNet

그림 7-28 AlexNet의 구성[71]



LeNet과 비교해 훨씬 최근인 2012년에 발표된 **AlexNet**은 딥러닝 열풍을 일으키는 데 큰 역할을 했다

- 활성화 함수로 ReLU를 이용한다.
- LRN Local Response Normalization이라는 국소적 정규화를 실시하는 계층을 이용한다.
- 드롭아웃을 사용한다.