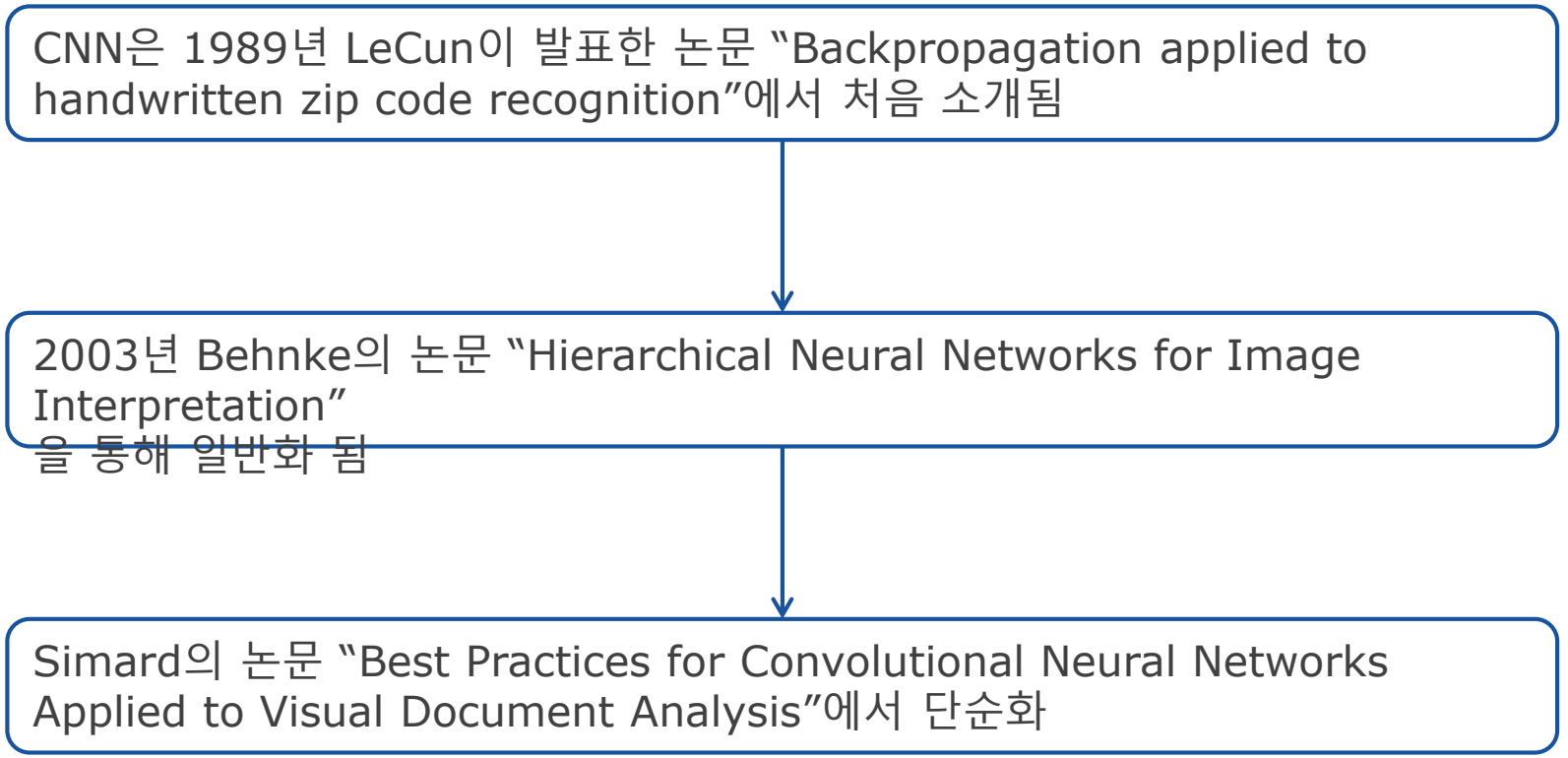
합성곱 신경망(CNN)

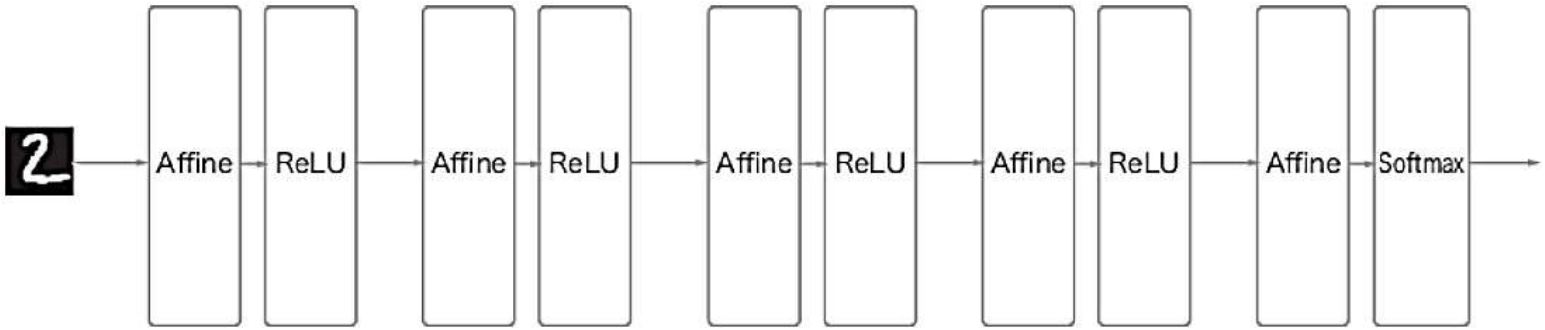
아꿈사
최종현

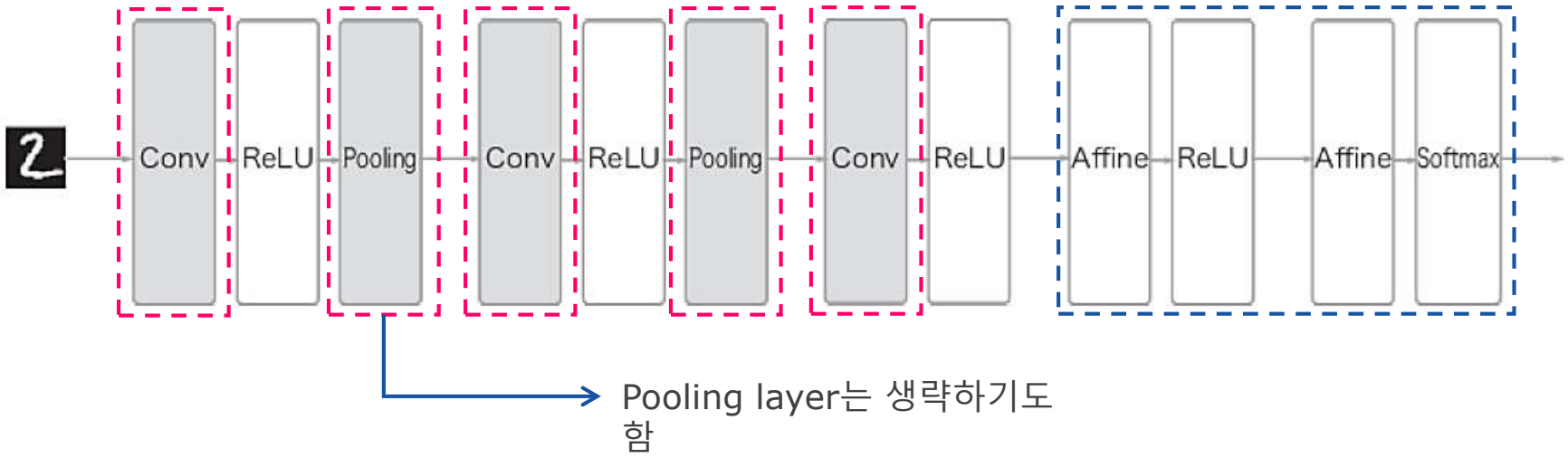
CNN(Convolutional Neural Network)의 역사



CNN의 구조

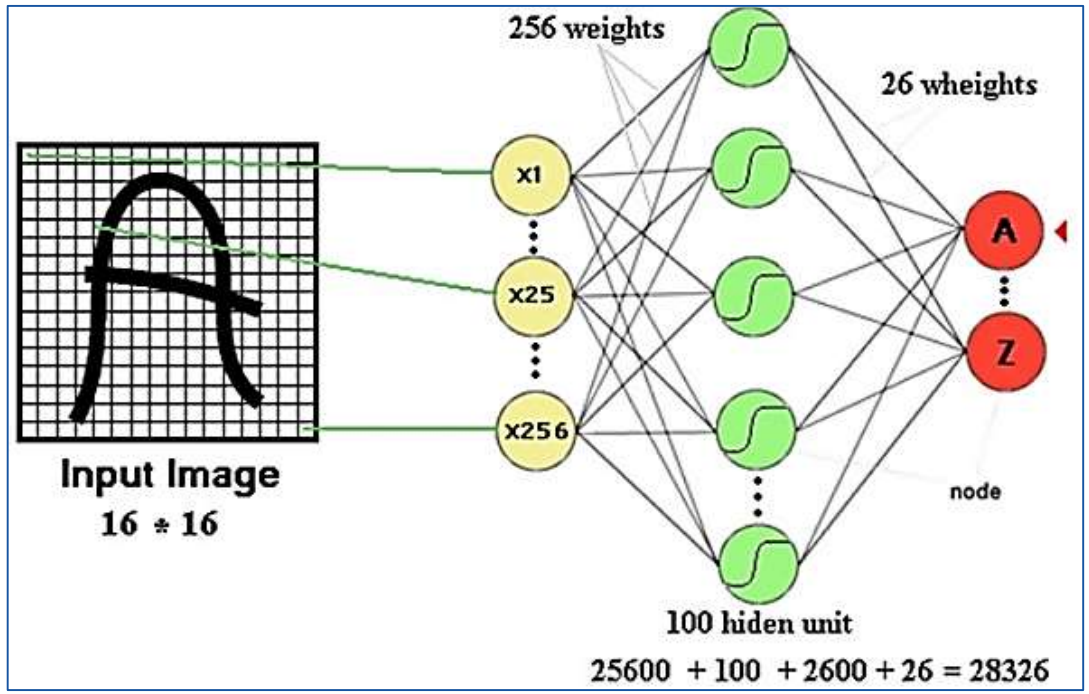
✓ **합성곱 계층**(Convolutional layer) 과 **풀링 계층**(pooling layer)이 추가됨

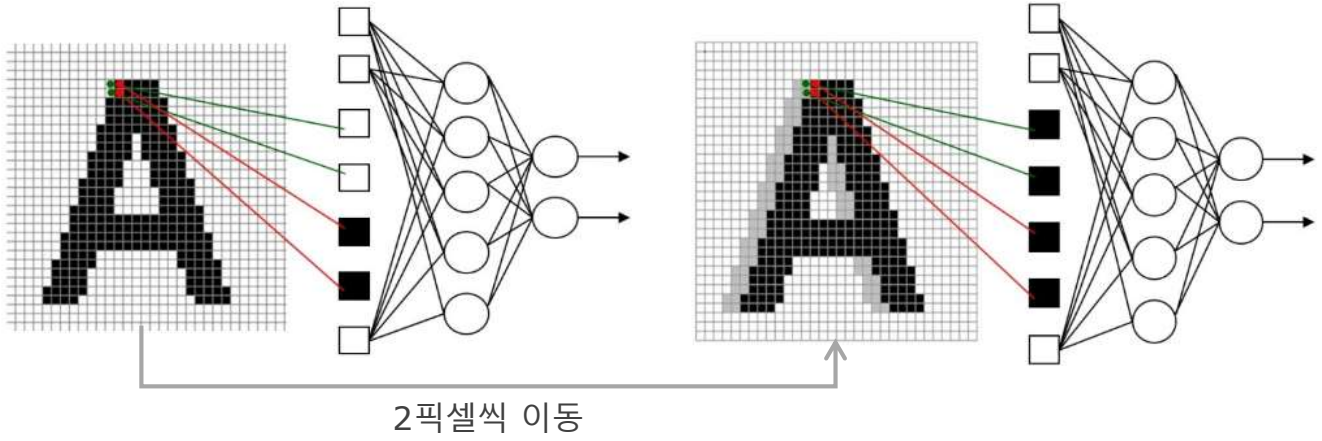




기존 MLNN(Multi-Layer NN)의 문제점

- ✓ 변수의 개수(Weight, Bias)
- ✓ 네트워크 크기
- ✓ 학습 시간



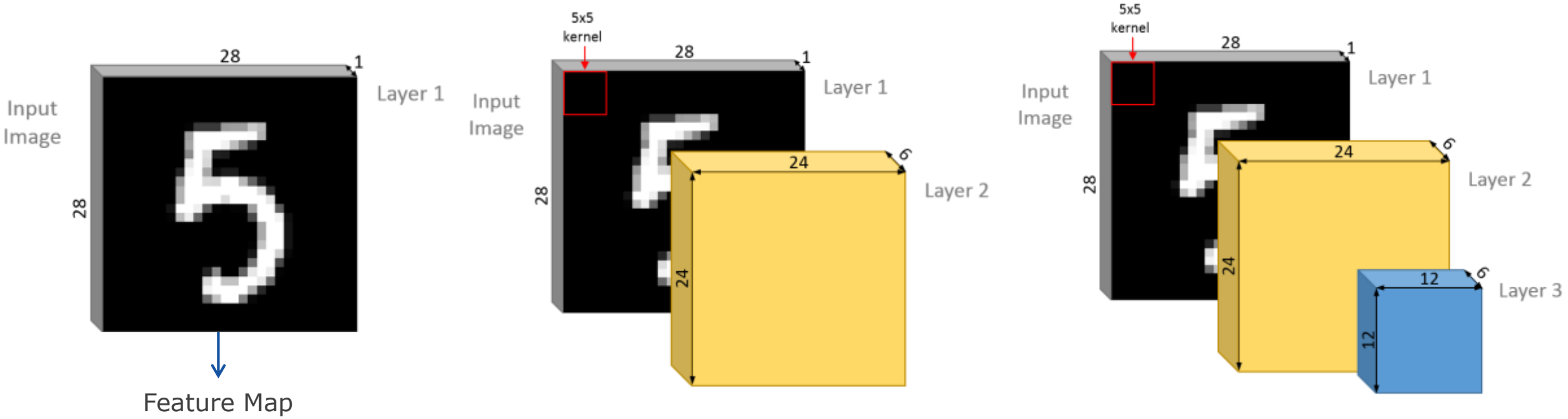


글자의 크기, 회전,
변형에 영향을 받음

→ *글자의 **형상**은 **고려하지 않고, raw data를 직접 처리**하기 때문에 **많은 양의 학습 데이터**가 필요하고, 따라서 **학습 시간이 길어짐***

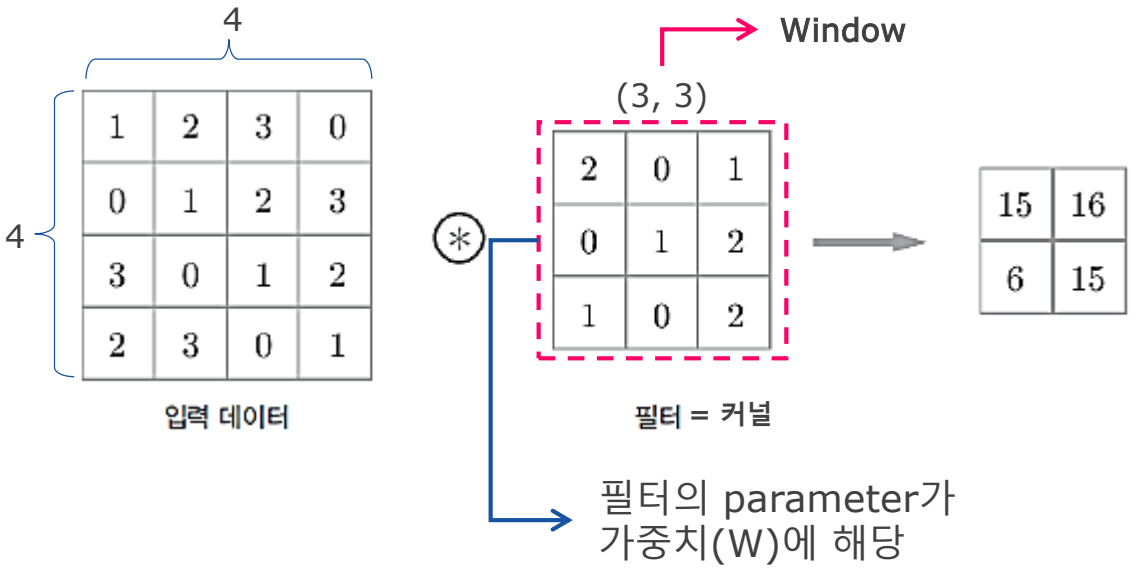
합성곱 계층(Convolutional layer)

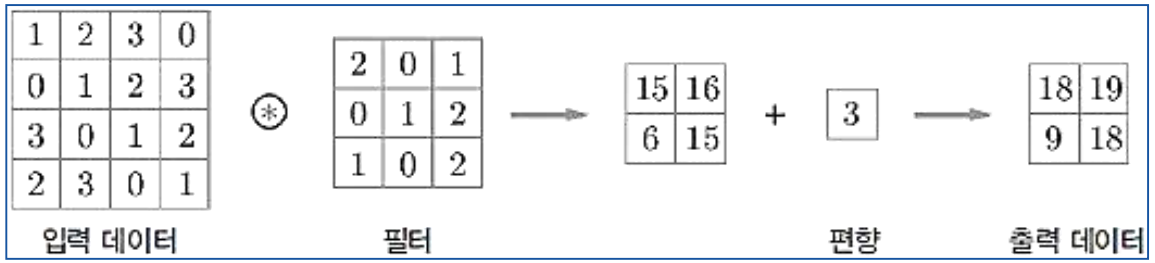
- ✓ 이미지 데이터는 **세로 · 가로 · 채널(색상)** 으로 구성된 데이터
- ✓ MNIST 데이터는 원래 (1, 28, 28)인 3차원 데이터 → Affine계층에 입력 시 784(=28 × 28)개의 1차원으로 입력
- ✓ 합성곱 계층의 입출력 데이터를 특징 맵(Feature Map)이라고 함
- ✓ 합성곱 계층에서 입력 데이터를 3차원으로 입력 받으며, 출력 또한 3차원으로 출력
  → *형상을* ***유지***

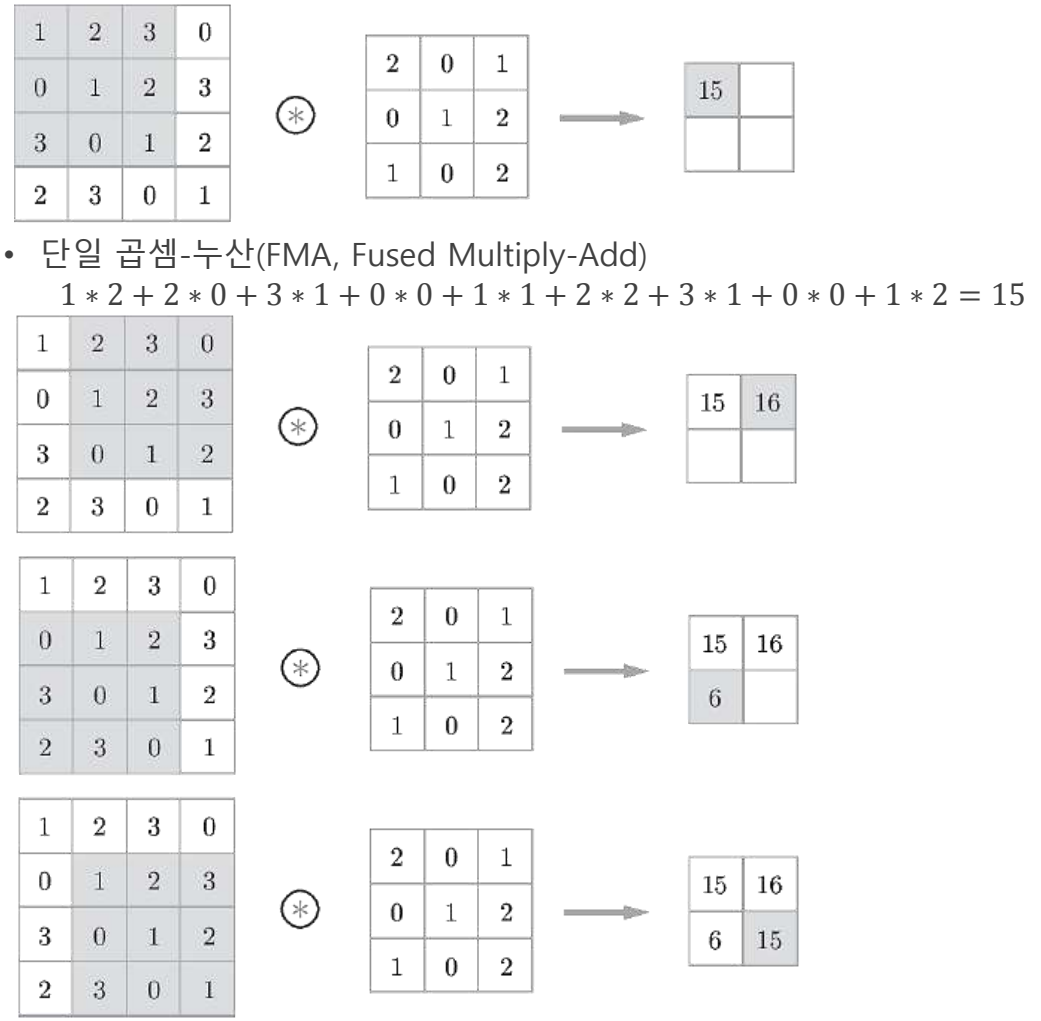


합성곱 계층 - 연산

- ✓ 합성곱 계층에서 연산 수행 → 필터(커널) 연산
- ✓ 데이터와 필터의 형상을 (높이height, 너비width)로 표기
- ✓ 윈도우window 를 일정 간격(Stride)으로 이동하며 계산







- 단일 곱셈-누산(FMA, Fused Multiply-Add)

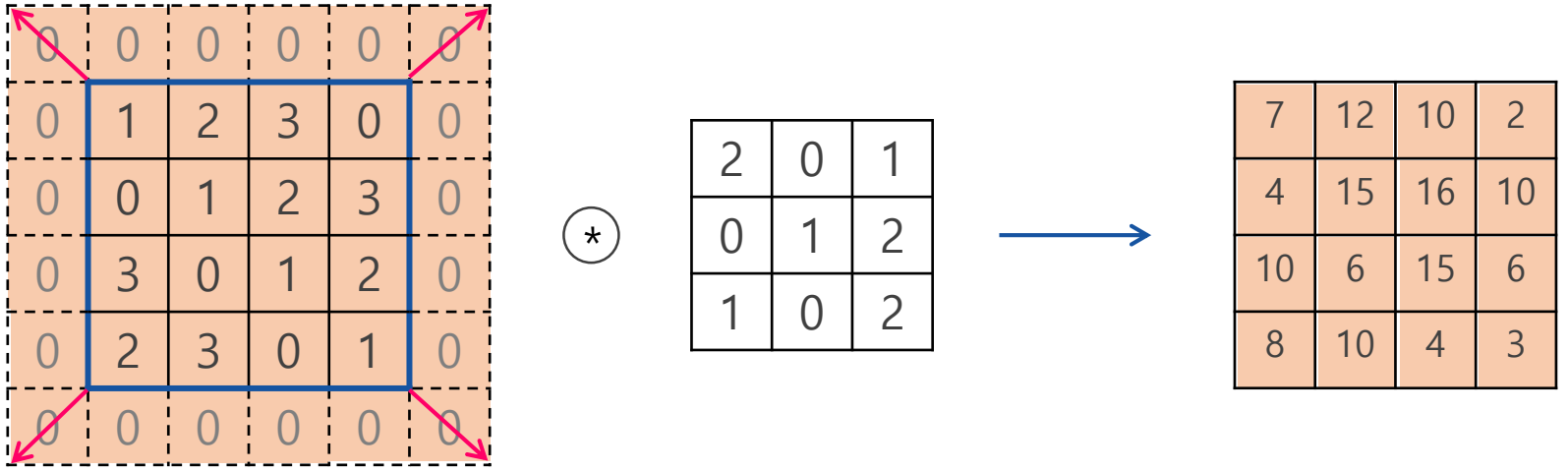
$$1*2+2*0+3*1+0*0+1*1+2*2+3*1+0*0+1*2=15$$

합성곱 계층 - 패딩

- ✓ 합성곱 연산을 수행하기 전, 입력데이터 주변을 특정값으로 채우는 것을 **패딩**Padding이라고 함
- ✓ 패딩은 출력데이터의 공간적 크기(Spatial size)를 조절하기 위해 사용
- ✓ 패딩은 hyperparameter로 어떤 값으로 채울지 결정할 수 있음 → 보통 zero-padding을 주로 사용

Padding을 사용하는 이유는?

→ Padding을 사용하지 않을 경우, 데이터의 Spatial 크기는 Conv 레이어를 지날때 마다 작아지게 되고, 가장자리의 정보들이 사라지게 된다.



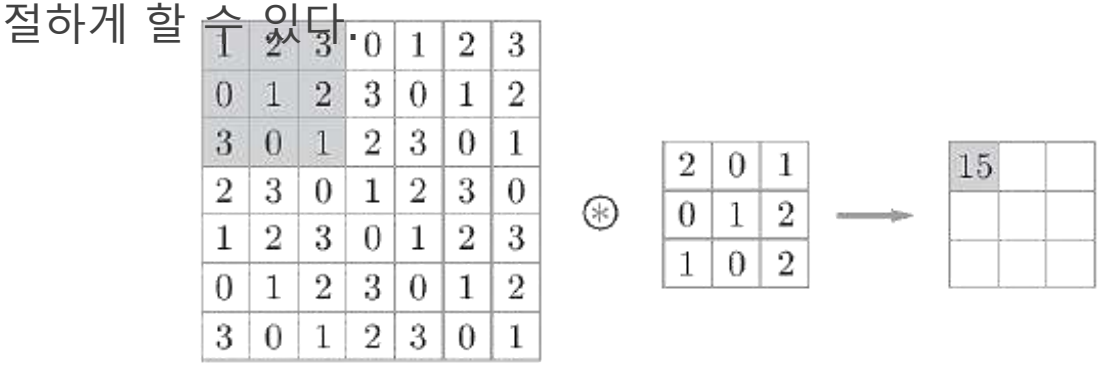
→ 1폭짜리 padding

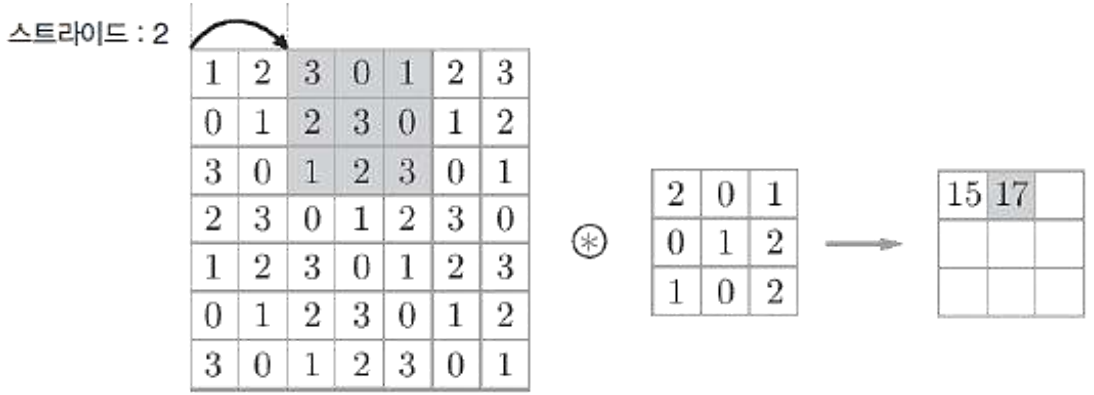
합성곱 계층 - 스트라이드

- ✓ 필터를 적용하는 위치의 간격을 **스트라이드**stride라고
- ✓ 스트라이드는 출력 데이터의 크기를 조절하기 위해 사용

Stride *값은 어떤것이 좋을까?*

보통 1과 같이 작은 값이 더 잘 작동한다. 또한, strid가 1일 경우 데이터의 spatial 크기는 Pooling
계층에서만 조절하게 할 수 있다.



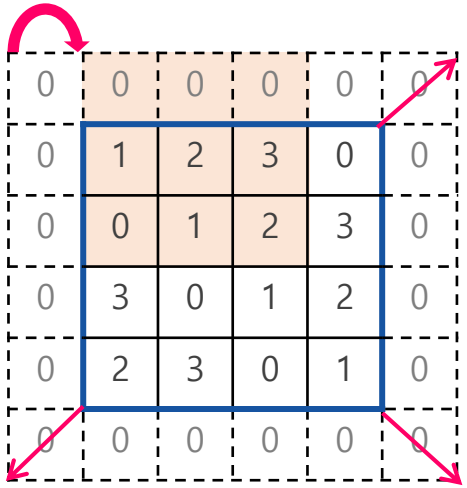


합성곱 계층 – 출력크기 계산

$$(OH, OW) = \left(\frac{H + 2P - FH}{S} + 1\ , \frac{W + 2P - FW}{S} + 1\right)$$

- $(H,\ W)$: 입력크기
- $(FH,\ FW)$: 필터크기
- $(OH,\ OW)$: 출력크기
- P : 패딩
- S : 스트라이드

*패딩(**P**) : 1,*
*스트라이드(**S**) : 1*



*

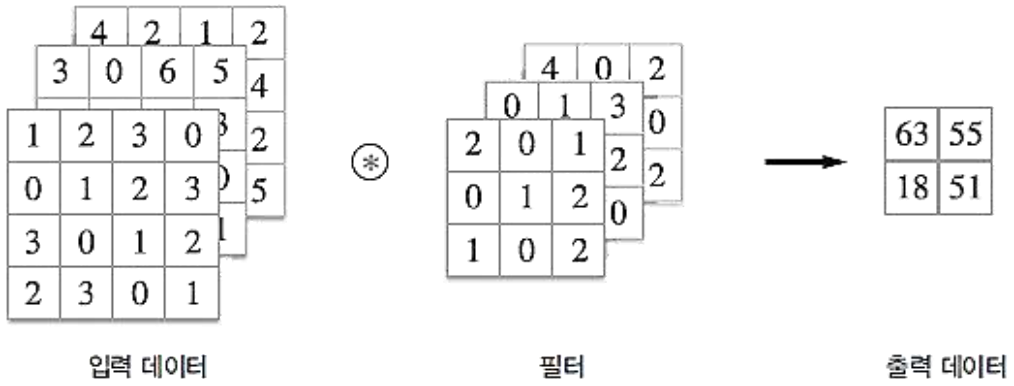
2	0	1
0	1	2
1	0	2

→

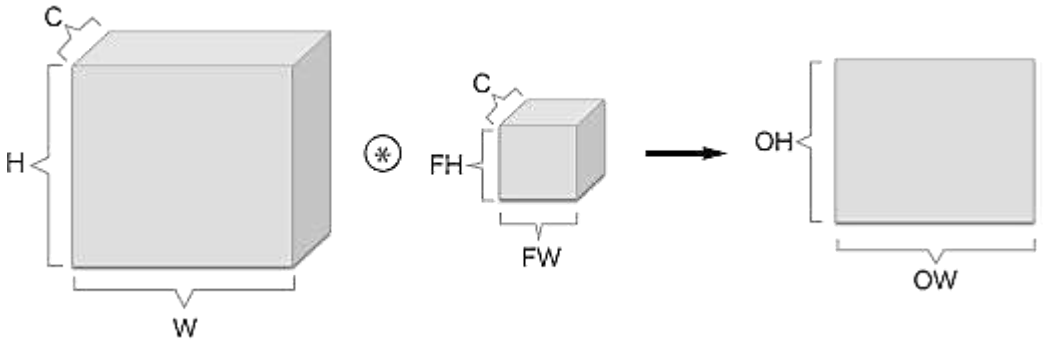
7	12	10	2
4	15	16	10
10	6	15	6
8	10	4	3

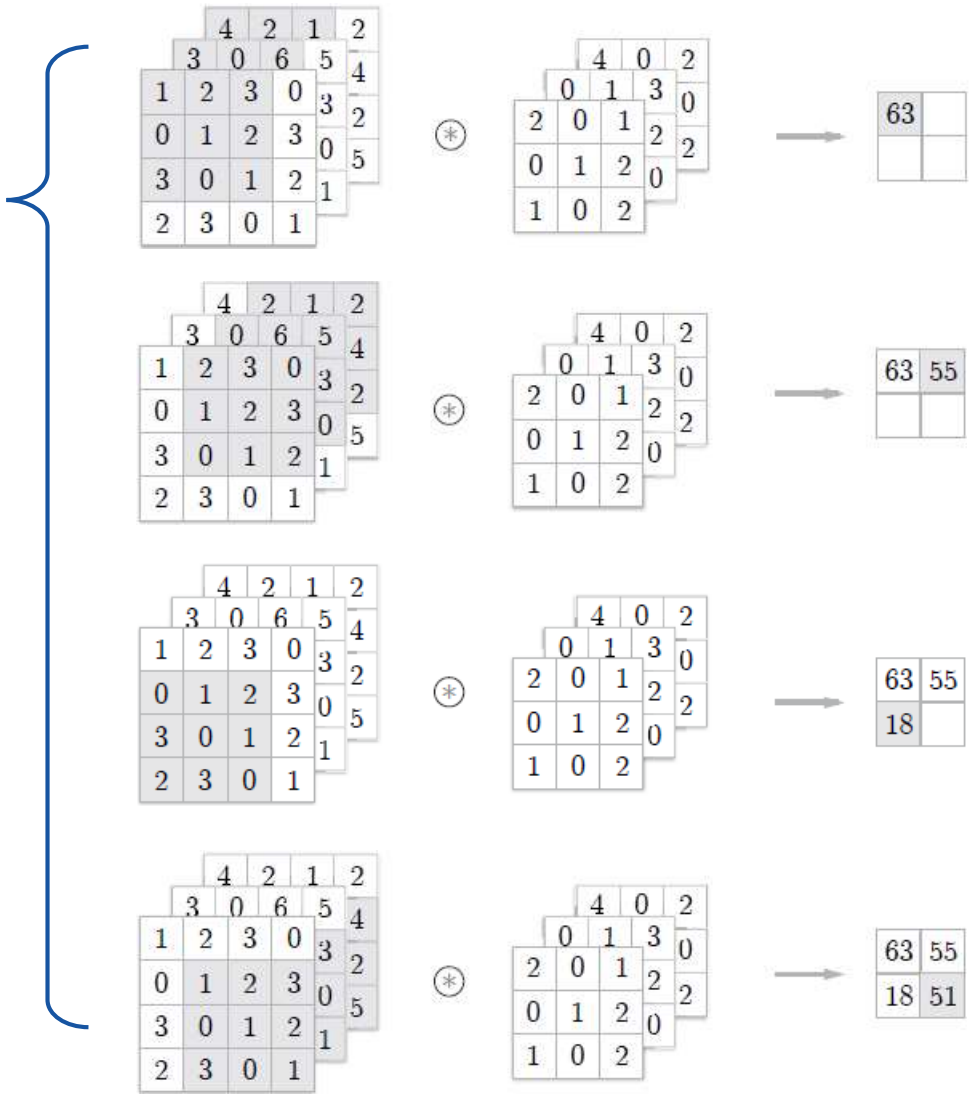
$$(OH, OW) = \left(\frac{4 + 2 * 1 - 3}{1} + 1\ , \frac{4 + 2 * 1 - 3}{1} + 1\right) = (\mathbf{4}, \mathbf{4})$$

합성곱 계층 – 3차원 데이터의 연산

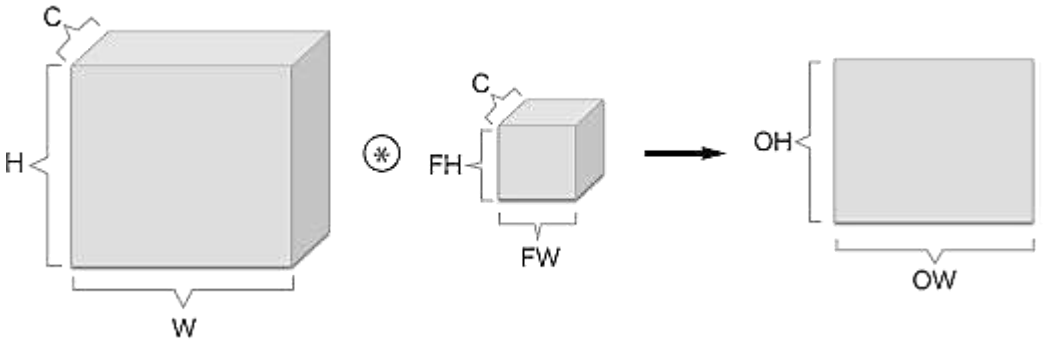


블록으로 생각하기

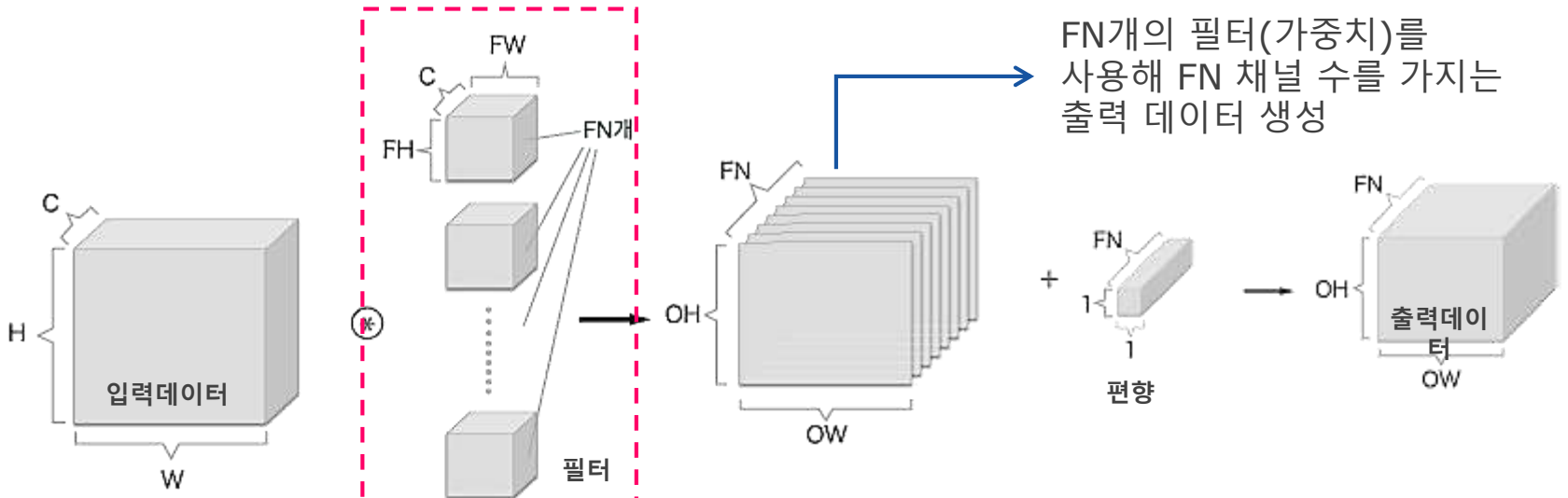




합성곱 계층 – 3차원 데이터의 연산



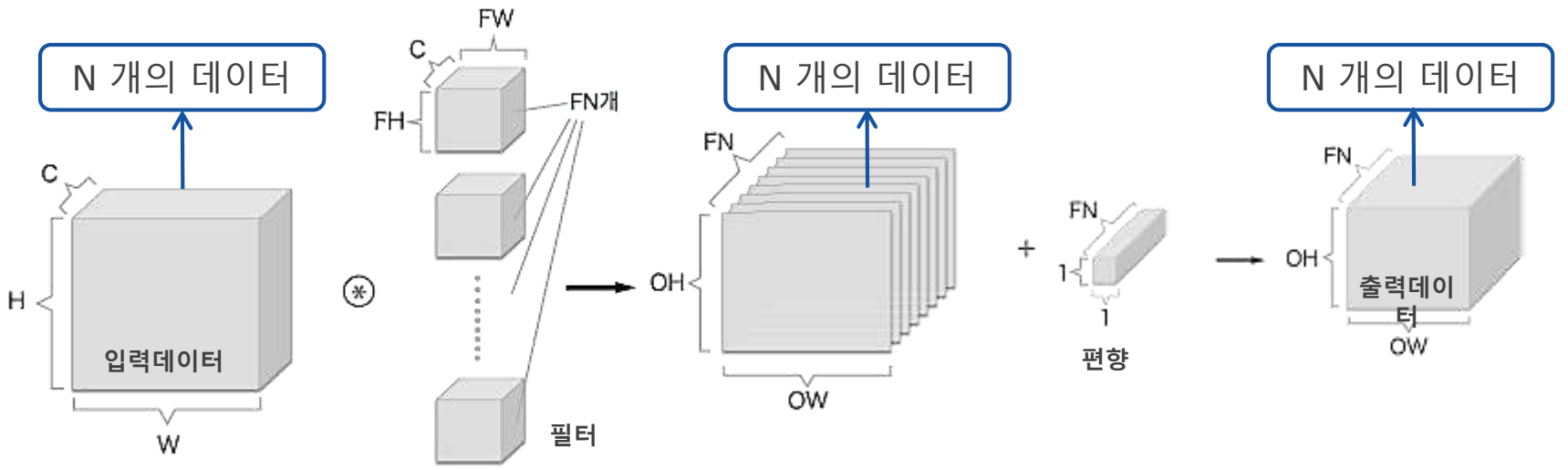
$(C, H, W) \circledast (C, FH, FW) \rightarrow (1, OH, OW)$



$(C, H, W) \circledast (FN, C, FH, FW) \rightarrow (FN, OH, OW) + (FN, 1, 1) \rightarrow (FN, OH, OW)$

합성곱 계층 – 배치 처리

- ✓ 데이터의 차원을 늘려 4차원 데이터로 저장
- ✓ (데이터 수, 채널 수, 높이, 너비)로 저장



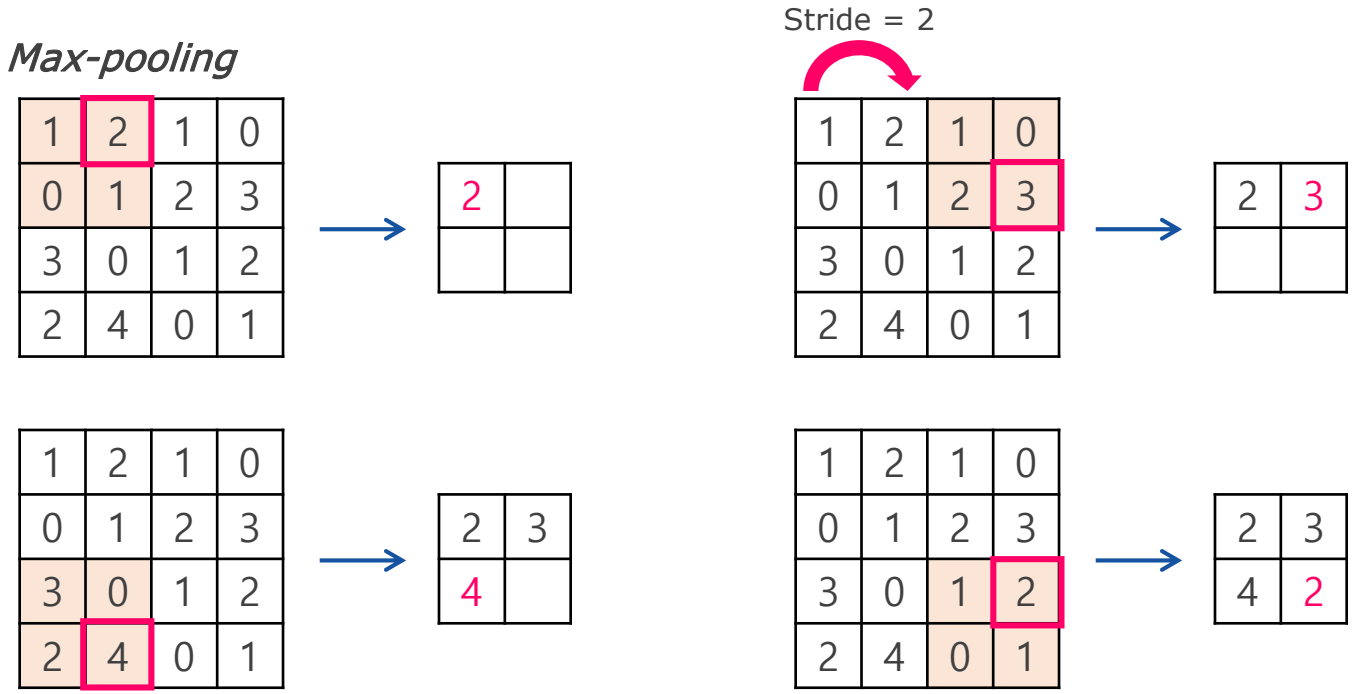
$$(N, C, H, W) \circledast (N, FN, C, FH, FW) \longrightarrow (N, FN, OH, OW) + (FN, 1, 1) \longrightarrow (N, FN, OH, OW)$$

풀링 계층 (Pooling Layer)

- ✓ 데이터의 공간적 크기Spatial size를 축소하는데 사용
- ✓ (보통)풀링의 윈도우 크기와 스트라이드는 같은 값으로 설정 (e.g *윈도우: 3X3, 스트라이드 : 3)*

*합성곱 계층(Conv Layer)에서도 **Padding**과 **Stride**를 통해 출력 데이터의 크기를 조절할 수 있는데?*

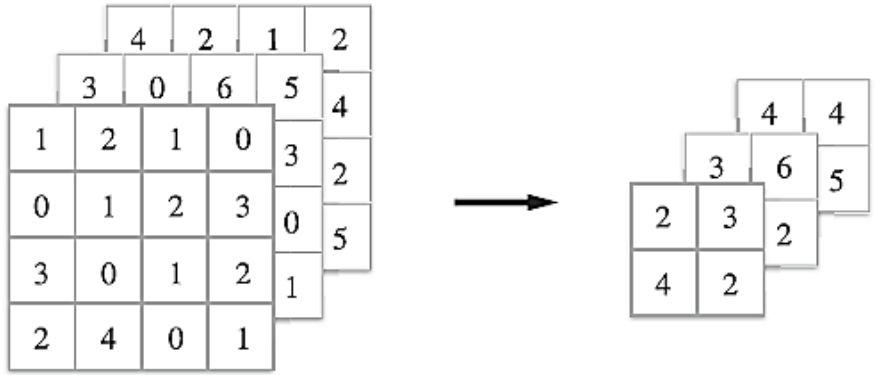
Conv 레이어에서는 출력 데이터의 Spatial 크기를 입력데이터의 크기를 그대로 유지하고,
Pooling 계층에서만 Spatial 크기를 조절할 수 있도록 한다.

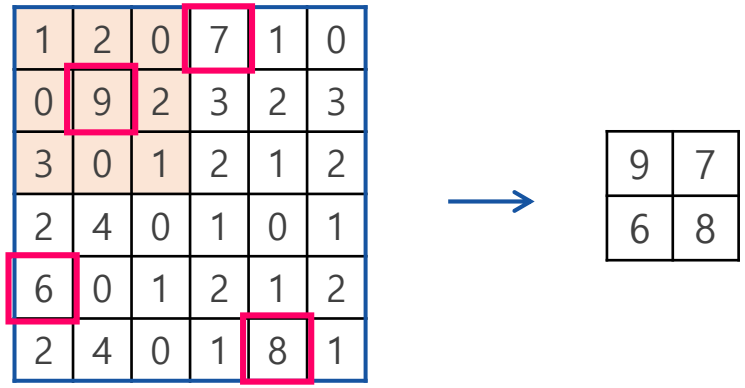


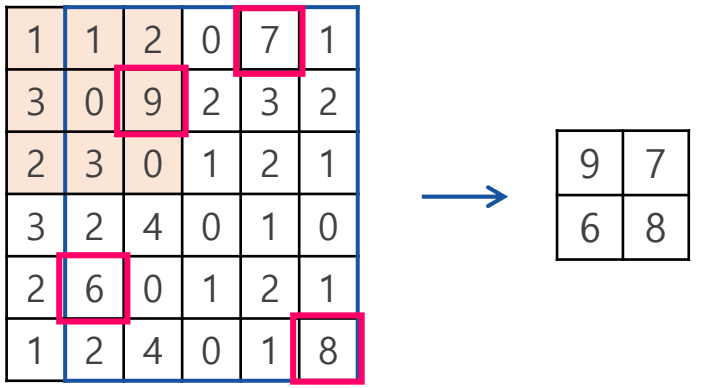
풀링 계층 - 특징

- ✓ 학습해야 할 매개변수가 없다 → ***최대값*** *or* ***평균값****을 취하기 때문에*
- ✓ 채널 수가 변하지 않는다 → *입력 데이터의 채널 수를 그대로 출력 데이터로*
- ✓ 입력의 변화에 영향을 적게 받는다(강건하다)

채널 수 그대로!

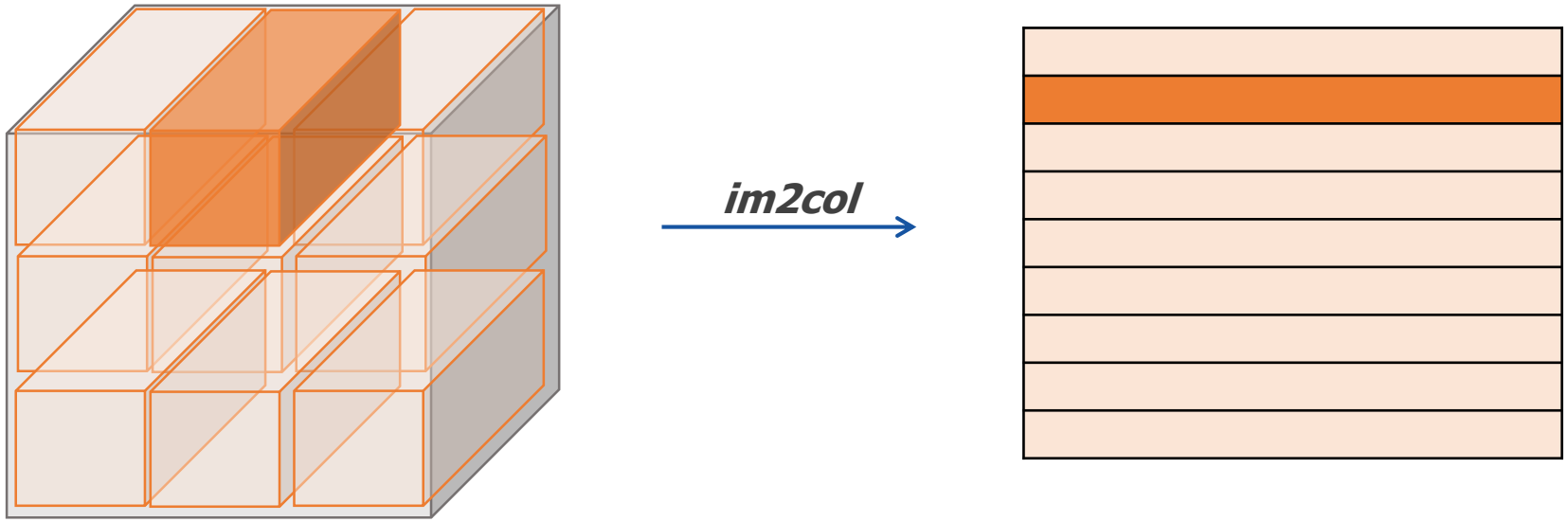




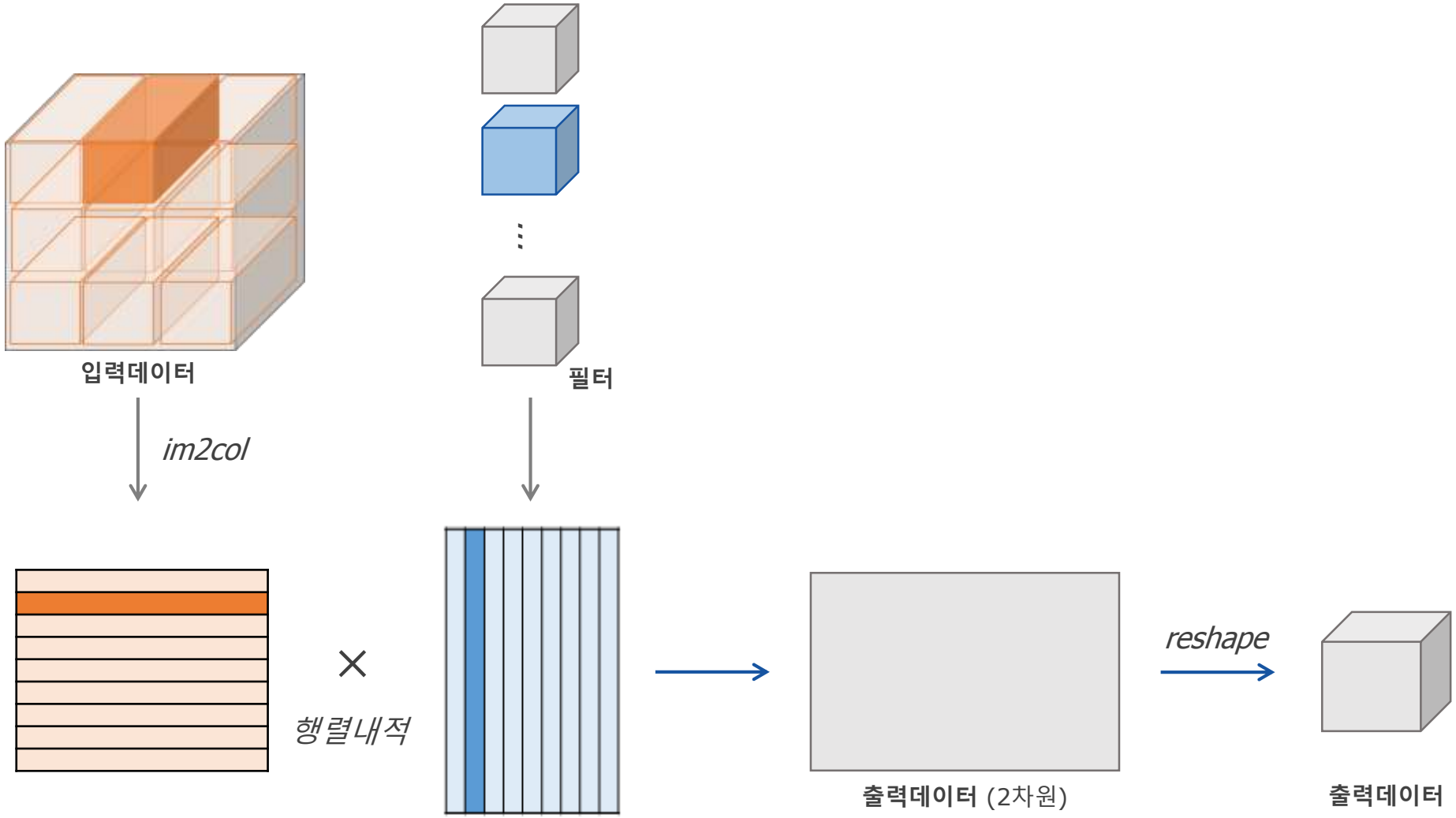


Conv Layer 구현 – im2col

- ✓ im2col 함수를 통해 입력데이터를 필터링(가중치 계산)하기 쉽도록 전개 → Caffe, Chainer등에서 제공
- ✓ Numpy 에서 for문 사용 방지
- ✓ im2col을 적용하여 3차원의 데이터를 2차원으로 변환



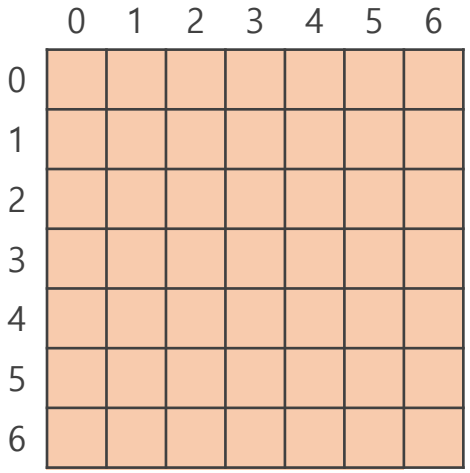
Conv Layer 구현 – im2col



Conv Layer 구현 – im2col

예제

```
x1 = np.random.rand(1, 3, 7, 7)

col1 = im2col(x1, 5, 5, stride = 1, pad = 0)

print(col1.shape) # (9, 75)
#결과
(9, 75)
```



- N = 1, C = 3, H = 7, W = 7
- filter_h, filter_w = 5, 5
- out_h, out_w = 3, 3
- img == input_data : True
- col.shape = (1,3,5,5,3)
- Slicing & indexing
  x[start : stop : step]
  x = [0,1,2,3,4,5,6,7,8,9]
  x[1 : 7 : 2] = [1, 3, 5]

transpose(0, 4, 5, 1, 2, 3)

형상 (N, C, FH, FW, OH, OW) → (N, OH, OW, C, FH, FW)

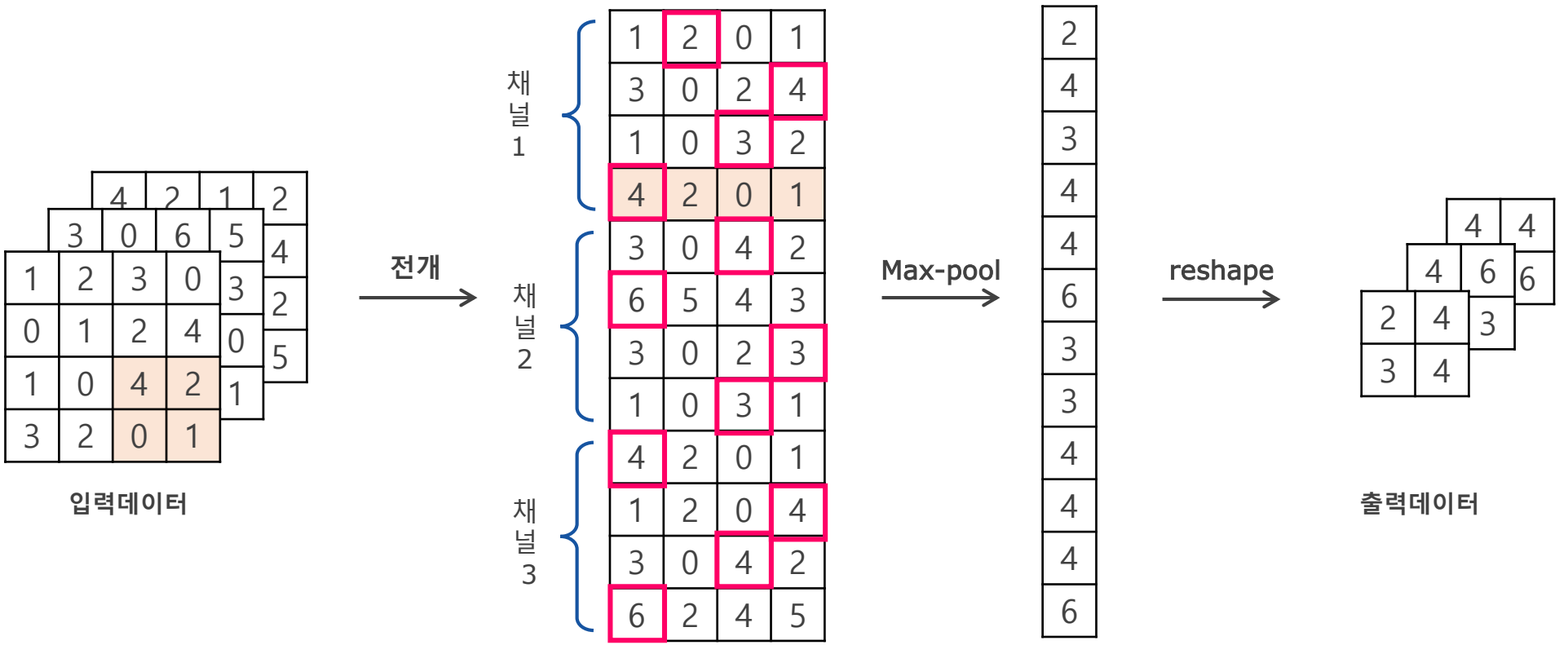
인덱스 (0, 1, 2, 3, 4, 5) → (0, 4, 5, 1, 2, 3)

➔ col.shape = (1, 3, 3, 3, 5, 5)

```
def im2col(input_data, filter_h, filter_w, stride=1, pad=0):
    """
    다수의 이미지를 입력받아 2차원 배열로 변환한다(평탄화)
    Parameters
    ----------
    :param input_data: (데이터 수, 채널 수, 높이, 너비) 입력 데이터
    :param filter_h: 필터의 높이 ,:param filter_w: 필터의 너비
    :param stride: 스트라이드 , :param pad: - 패딩(padding)
    :return: col: 2차원 배열
    """
    N, C, H, W = input_data.shape
    out_h = (H + 2*pad - filter_h)//stride + 1
    out_w = (W + 2*pad - filter_w)//stride + 1

    img = np.pad(input_data, [(0, 0), (0, 0), (pad, pad), (pad, pad)], 'constant')
    col = np.zeros((N, C, filter_h, filter_w, out_h, out_w))

    for y in range(filter_h):
        y_max = y + stride * out_h
        for x in range(filter_w):
            x_max = x + stride * out_w
            col[:, :, y, x, :, :] = img[:, :, y:y_max:stride, x:x_max:stride]

    col = col.transpose(0, 4, 5, 1, 2, 3).reshape(N * out_h * out_w, -1)
    return col
```

Pool Layer 구현

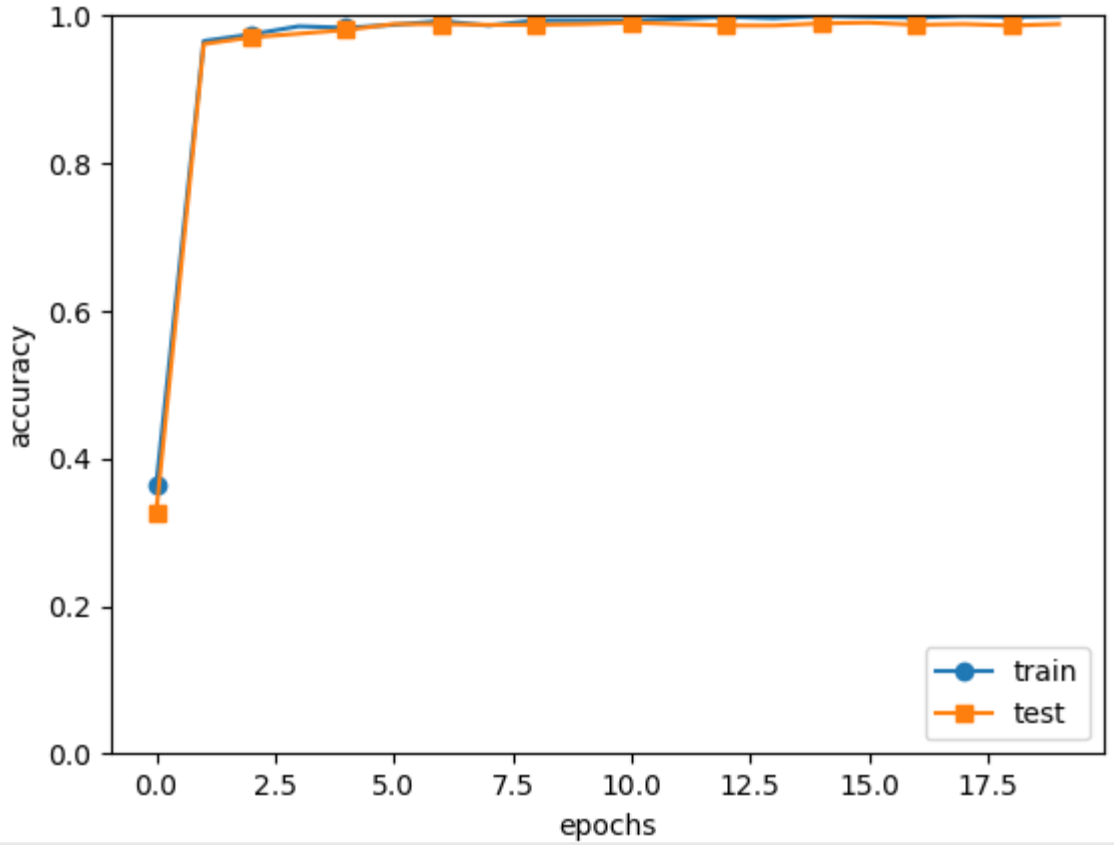
- ✓ 풀링 계층 또한 im2col 함수를 통해 입력데이터를 전개
- ✓ Conv Layer와는 달리 채널 쪽이 독립적 → 채널마다 독립적으로 전개



CNN 구현 테스트

✓ ‘train_convent.py’ 예제 소스코드 확인

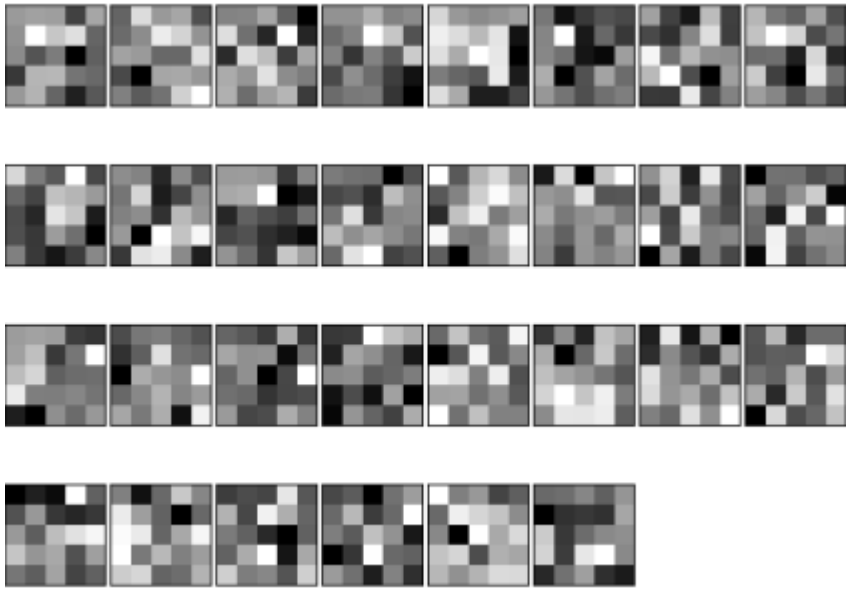
```
network = SimpleConvNet(input_dim=(1,28,28),
                        conv_param = {'filter_num': 30, 'filter_size': 5, 'pad': 0, 'stride': 1},
                        hidden_size=100, output_size=10, weight_init_std=0.01)


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)
```



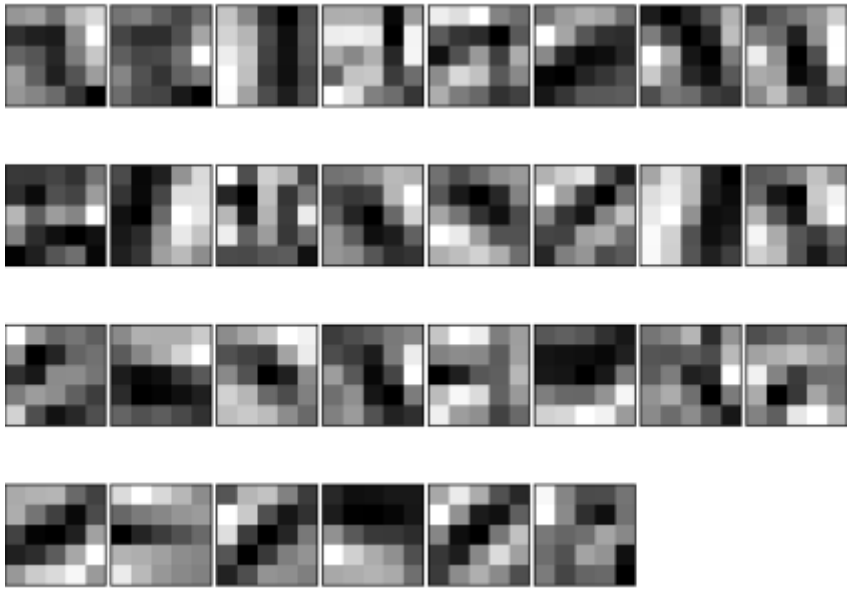
CNN 시각화

- ✓ 앞의 'train_convent.py' 예제 에서 필터의 형상 = (30, 1, 5, 5)
- ✓ 필터의 크기가 5 X 5며, 채널이 1개 ➔ 회색조 이미지로 시각화 할 수 있음
- ✓ 'visualize_filter.py' 소스코드 확인 → ConV계층에서의 가중치(필터) 시각화
- ✓ 학습 후 필터는 흰색에서 검은색으로 변화하는 필터와 Blob(국소적으로 덩어리진 영역) 등으로 변화

학습 전

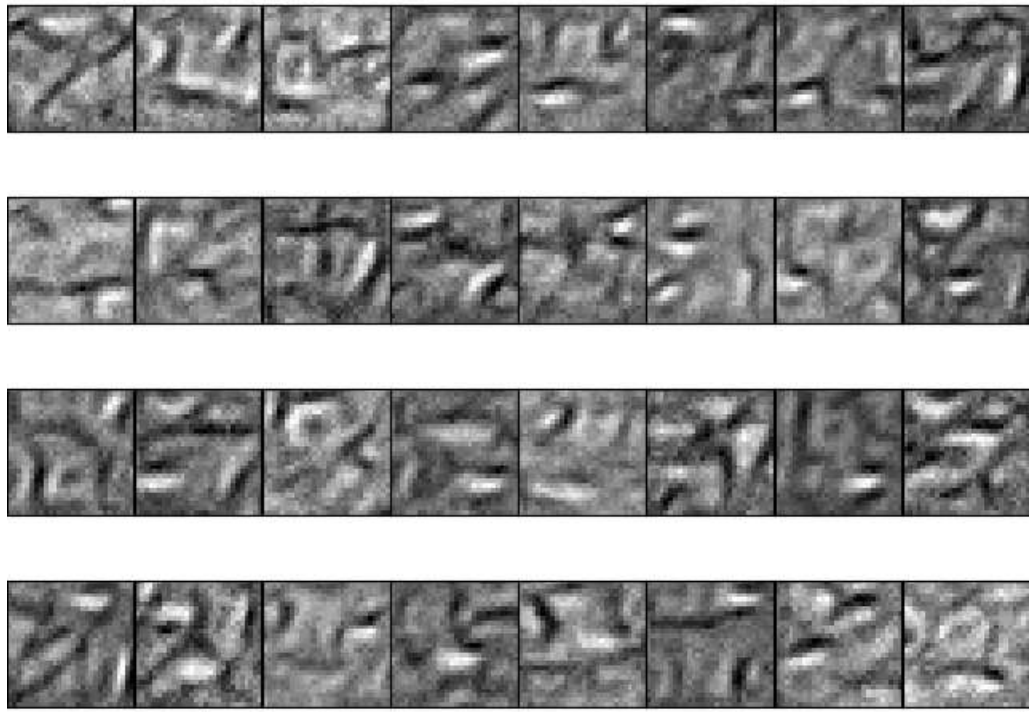


학습 후

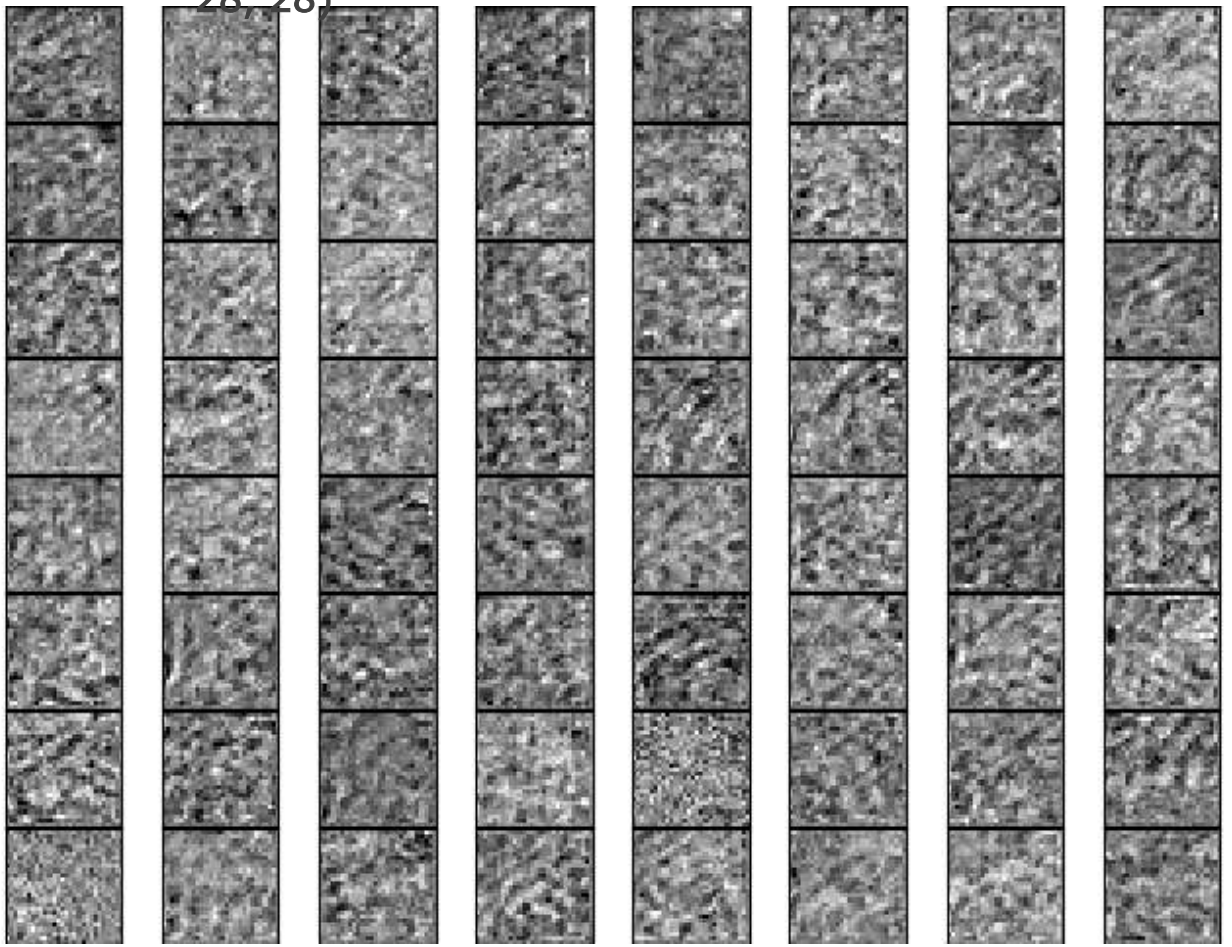


CNN 시각화

Conv 1st Layer (31, 1, 28,

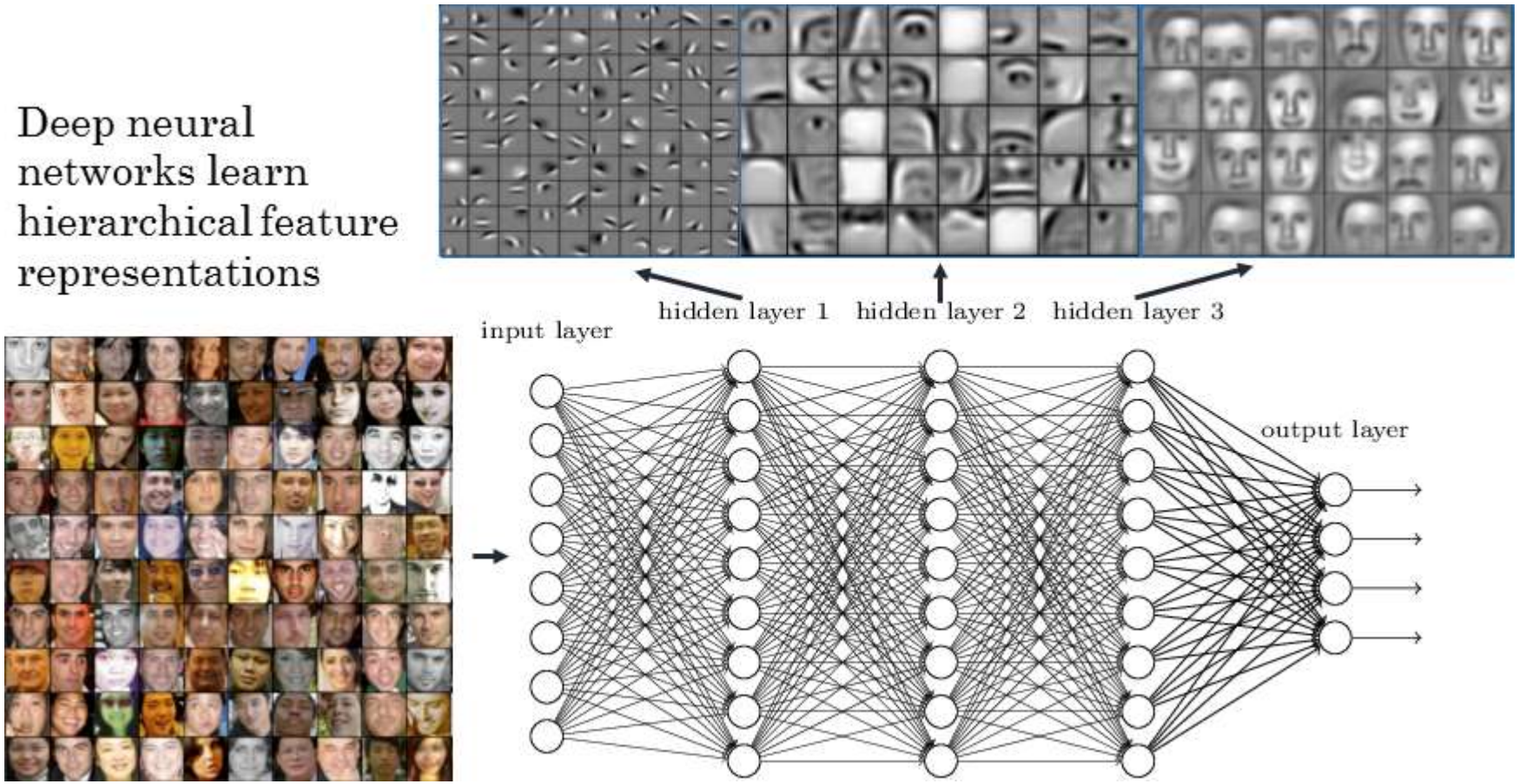


Conv 2nd Layer (64, 32, 28, 28)



CNN 시각화 – 층 깊이에 따른 정보

- ✓ 계층이 깊어질 수록 추출되는 정보는 더 추상화 됨
- ✓ 처음 층에는 단순한 에지 → 텍스처 → 사물의 일부



CNN 종류 - LeNet

- ✓ 손글씨 숫자를 인식하는 네트워크, 1998년에 제안
- ✓ 합성곱 계층과 풀링 계층(단순 서브샘플링)으로 이루어짐
- ✓ 활성화 함수로 Sigmoid 함수 사용

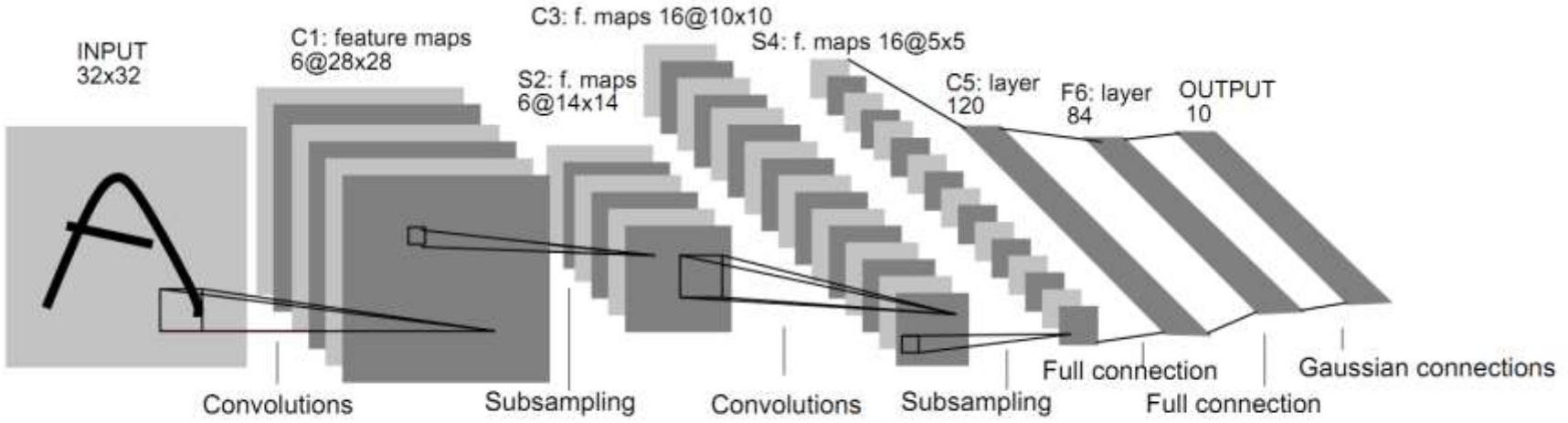
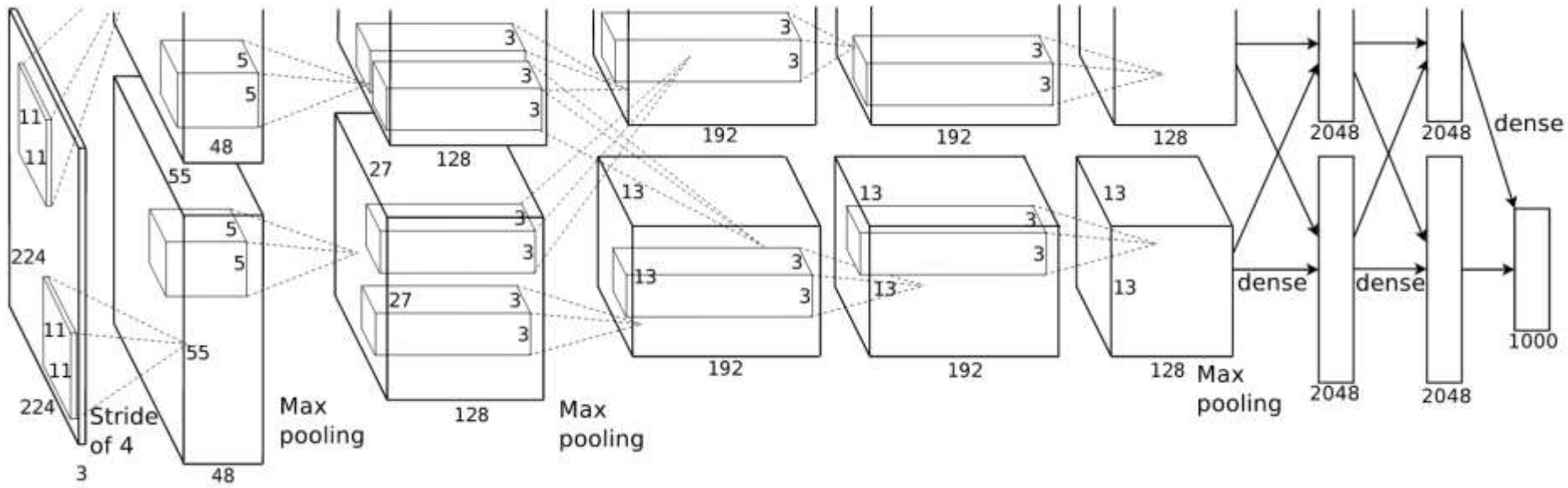


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

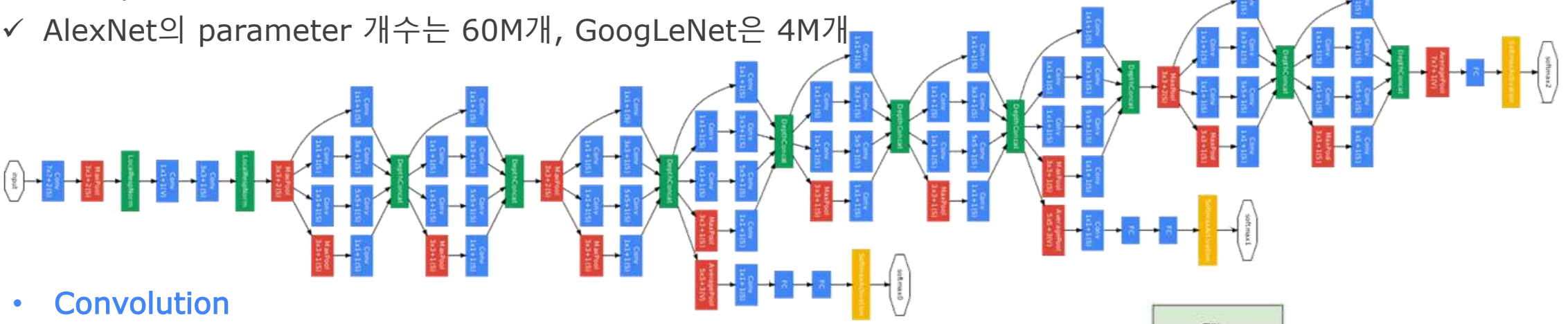
CNN 종류 - AlexNet

- ✓ 2012년 ImageNet ILSVRC에서 1위
- ✓ 활성화 함수로 ReLu 사용
- ✓ LRN(Local Response Normalization) : 국소적 정규화 계층 사용
- ✓ 드롭아웃(Dropout) 사용

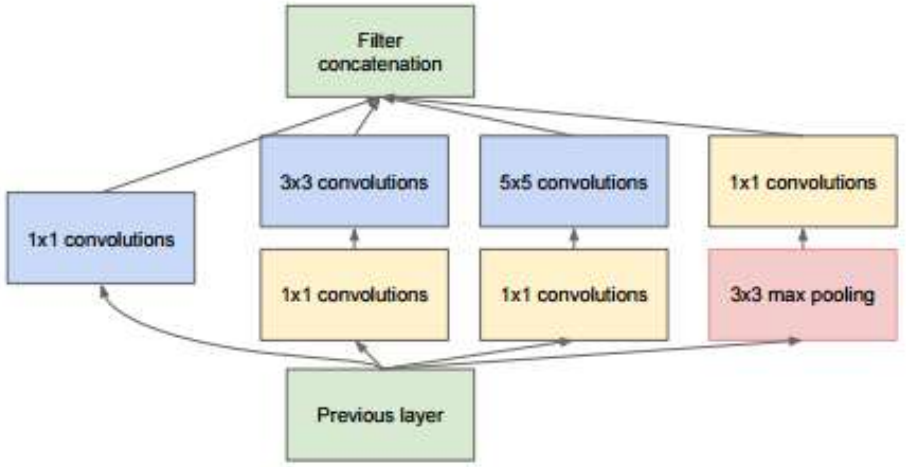


CNN 종류 - GoogLeNet

- ✓ 2014년 ImageNet ILSVRC에서 1위
- ✓ "Inception Module" 개념을 도입하여 네트워크의 파라미터 수를 대폭 줄임 → 9개의 Inception Module
- ✓ AlexNet의 parameter 개수는 60M개, GoogLeNet은 4M개



- Convolution
- Pooling
- Softmax
- Concat/Normalize



(b) Inception module with dimension reductions


THANK YOU