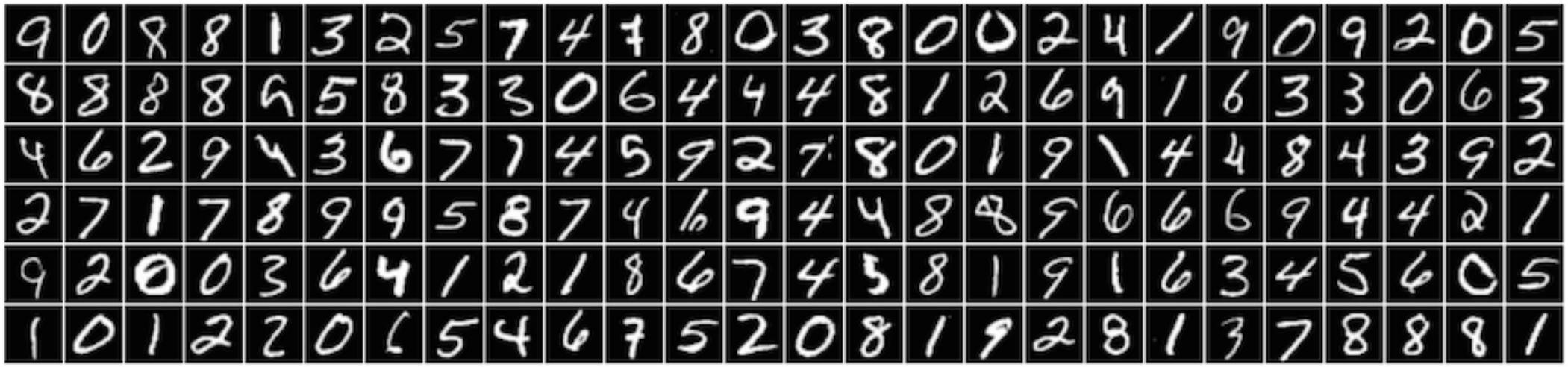
ABC to Deep Learning

Myeonghun Park

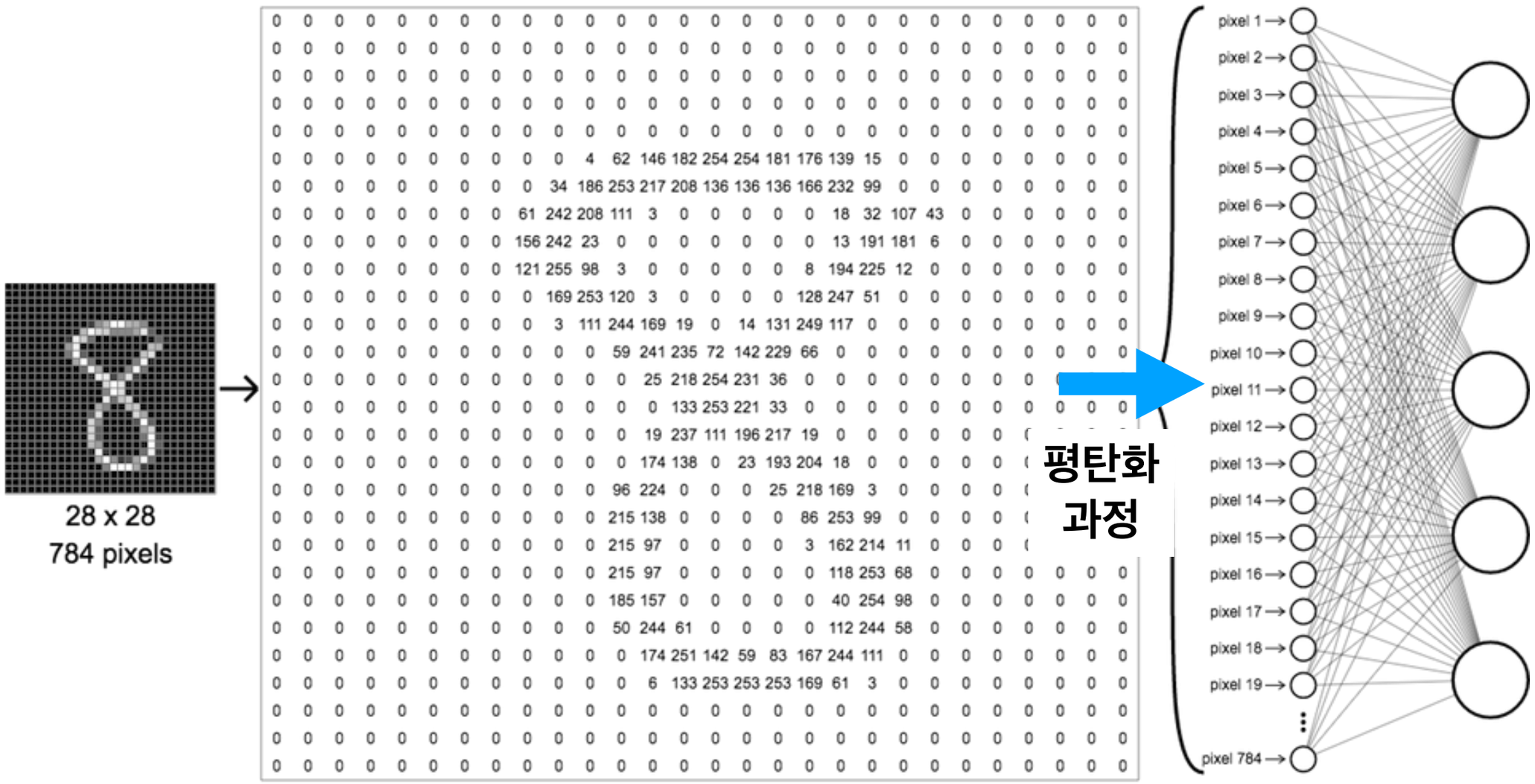
합성곱 신경망

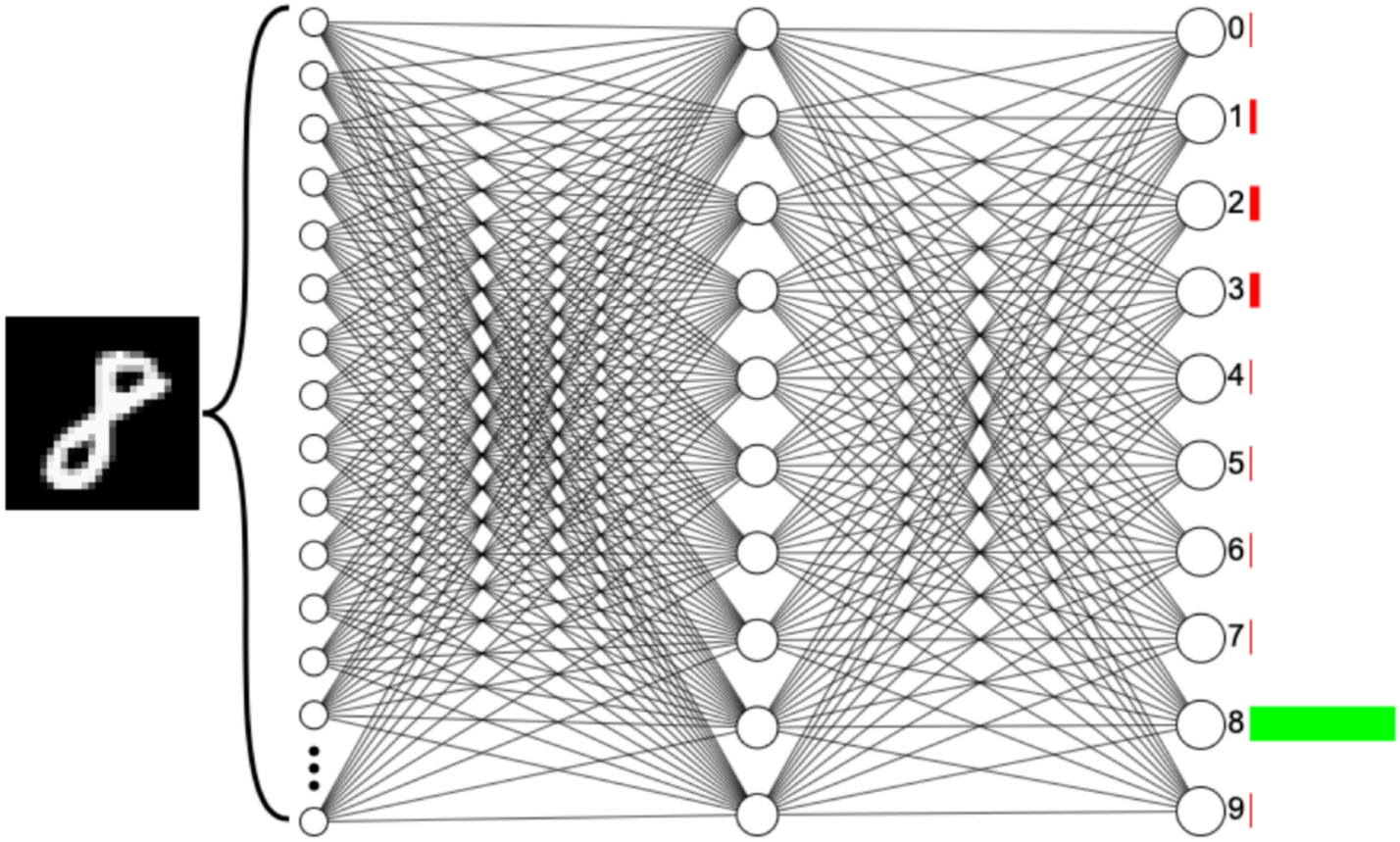
Convolutional Neural Network (CNN)

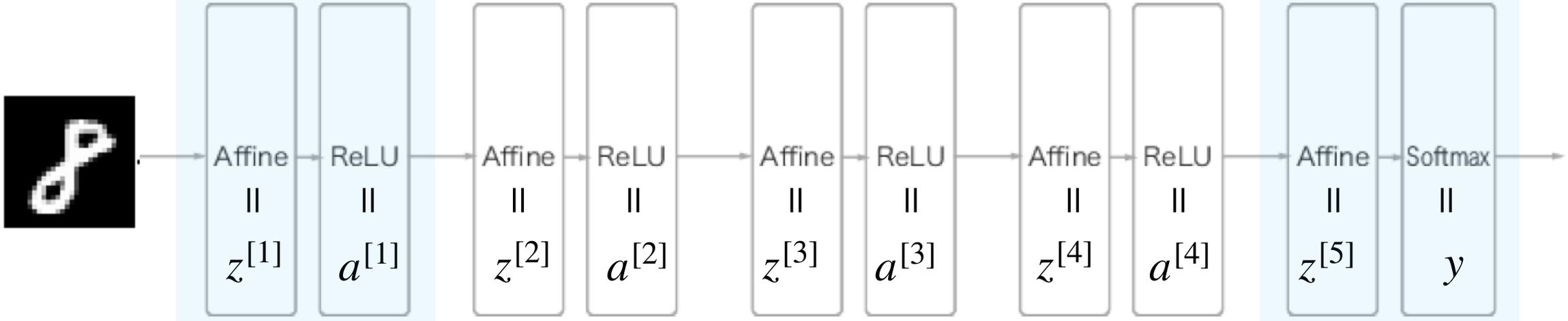
- 이미지 처리 : MNIST Database (손글씨 숫자: 60K 학습용 데이터, 10K 테스트 데이터)



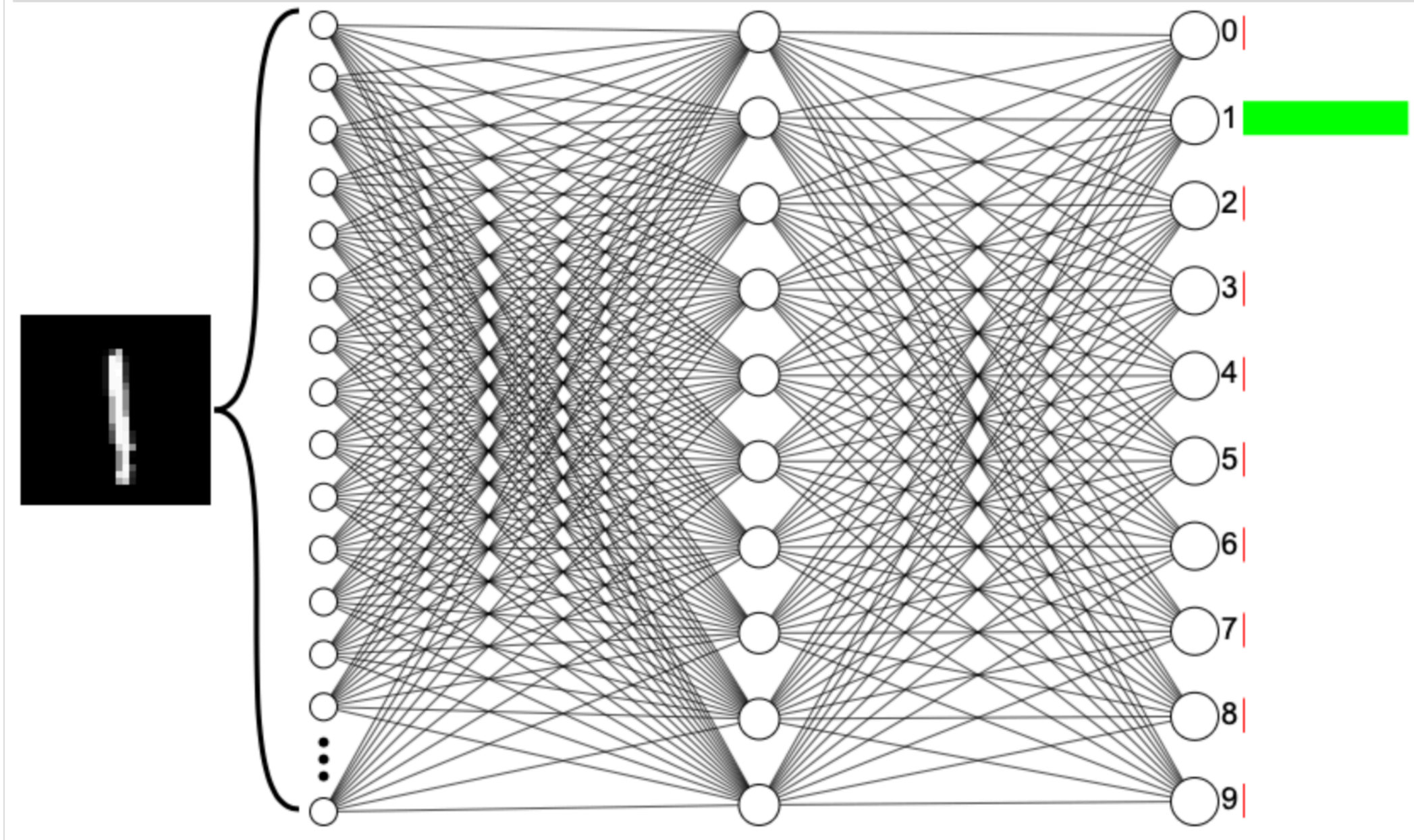
- **이미지 크기 =** $28 \times 28 = 784$ **픽셀, 각 픽셀의 세기 :** $[0,255]$







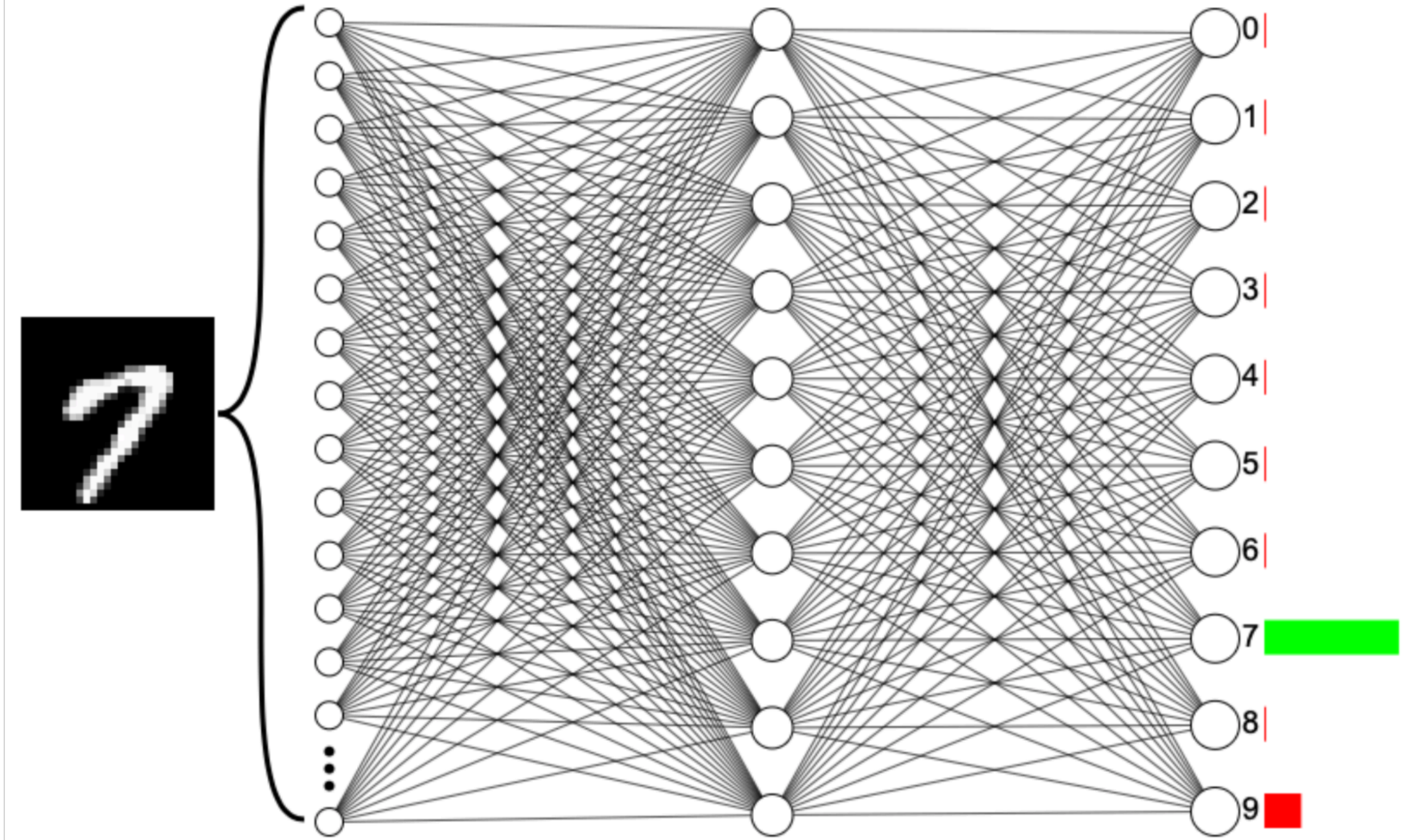
- Affine transformation (아핀변환) $z^{[1]} = W^{[1]}x + b^{[1]}, \quad z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$
- Activation Function (활성화 함수) $a^{[1]} = g^{[1]}(z^{[1]}), \quad a^{[2]} = g^{[2]}(z^{[2]})$
- Classification (분류) $y_k = \frac{exp(z_k)}{\sum_{i=0}^{9} exp(z_i)}$



입력층
(Input layer)

은닉층
(Hidden layer)

출력층
(Output layer)

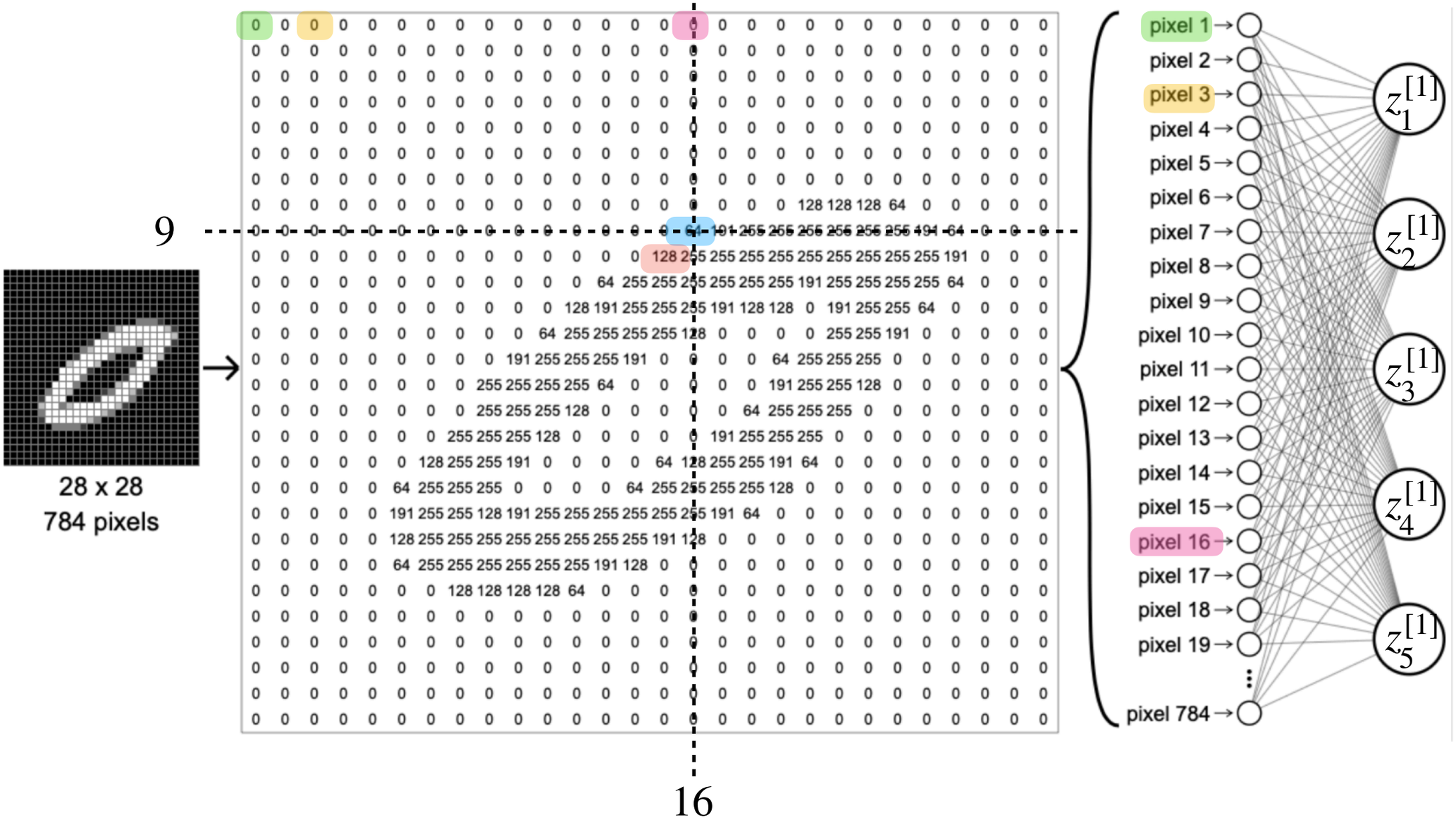


입력층
(Input layer)

은닉층
(Hidden layer)

출력층
(Output layer)

Fully-Connected DNN의 문제점



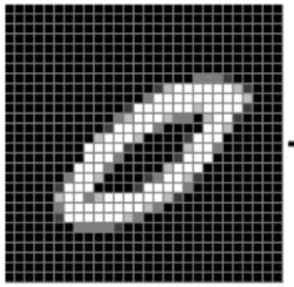
1. 단순 나열이므로, 이미지의 구조와 같은 픽셀 사이들의 관계를 즉각적으로 파악못함.

$= 8 \times 28 + 16 = 240$

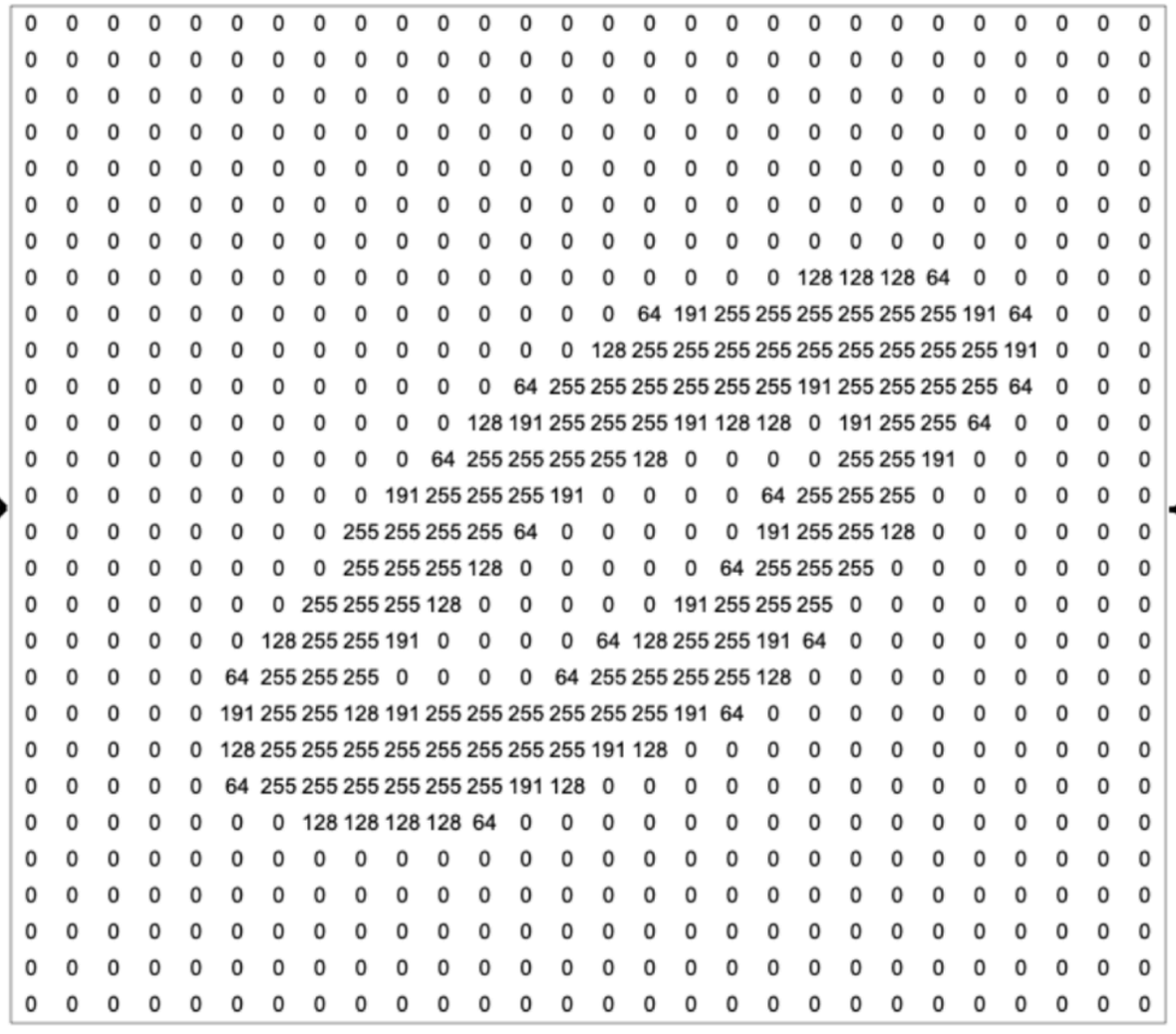
$= 9 \times 28 + 15 = 267$

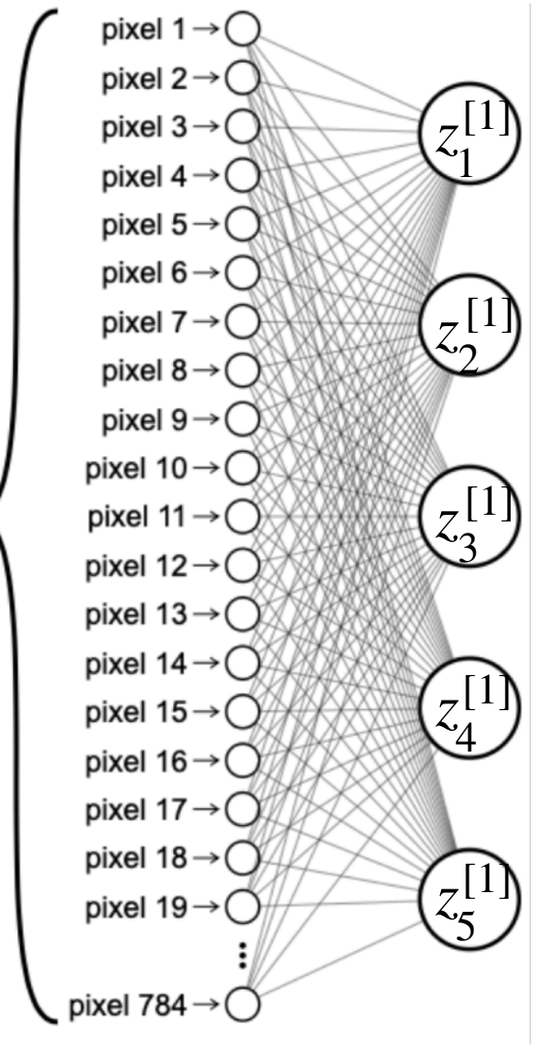
바로 옆의 픽셀들이 입력층에서는 두 노드 사이에 26개의 다른 픽셀들이 들어가게 됨.
즉, 두 인접 픽셀들의 거리가 증가함.

Fully-Connected DNN의 문제점



28 x 28
784 pixels





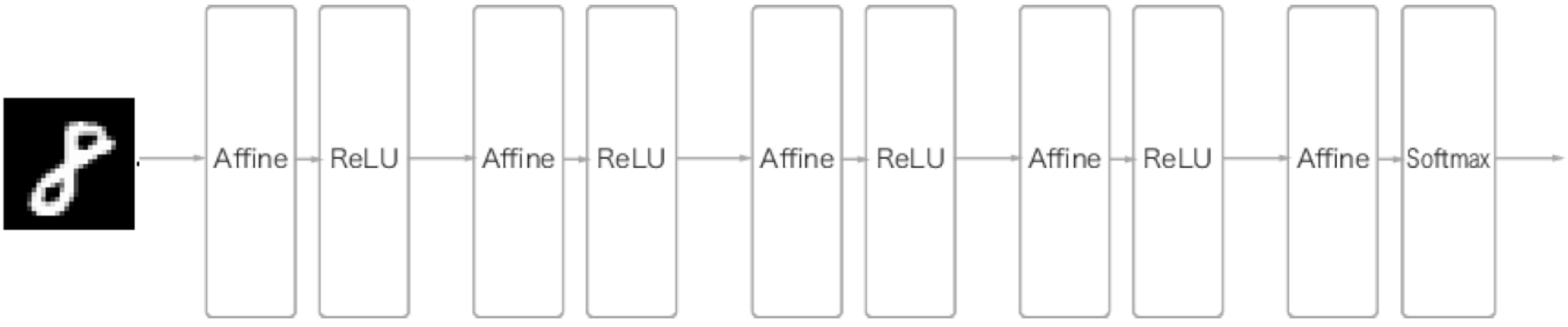
2. 매개변수 (Weight, bias) 가 **너무 많이 필요**

$$z^{[1]} = \begin{pmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \\ z_5^{[1]} \end{pmatrix} = \begin{pmatrix} w_{(1,1)}^{[1]} & w_{(1,2)}^{[1]} & \cdots & w_{(1,784)}^{[1]} \\ w_{(2,1)}^{[1]} & w_{(2,2)}^{[1]} & \cdots & w_{(2,784)}^{[1]} \\ w_{(3,1)}^{[1]} & w_{(3,2)}^{[1]} & \cdots & w_{(3,784)}^{[1]} \\ w_{(4,1)}^{[1]} & w_{(4,2)}^{[1]} & \cdots & w_{(4,784)}^{[1]} \\ w_{(5,1)}^{[1]} & w_{(5,2)}^{[1]} & \cdots & w_{(5,784)}^{[1]} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{784} \end{pmatrix} + \begin{pmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \\ \vdots \\ b_{784}^{[1]} \end{pmatrix}$$

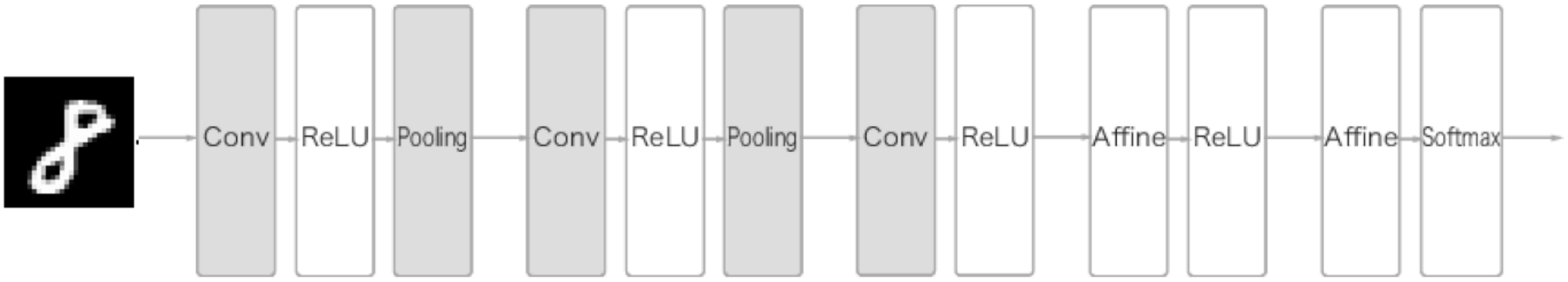
입력층에서 노드 5개짜리 1번째 은닉층을 fully-connect 할 때 필요한 매개변수

$$784 \times 5 + 784 = 4{,}704$$

Fully-connected DNN



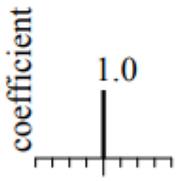
Convolutional Neural Network (CNN)



Convolution (합성곱)

- 이미지 처리: 다양한 필터를 사용하여 특징을 추출

original



0	0	0
0	1	0
0	0	0

Filtered
(no change)

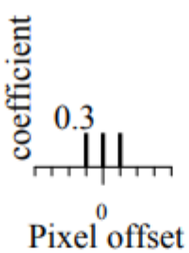
Identity

original

Box filter:

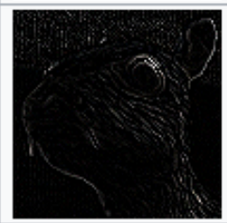
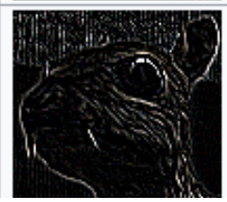
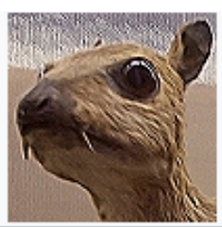
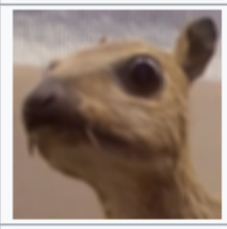
$\frac{1}{9}$

1	1	1
1	1	1
1	1	1



Blurred (filter applied in **both dimensions**).

Blur

Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

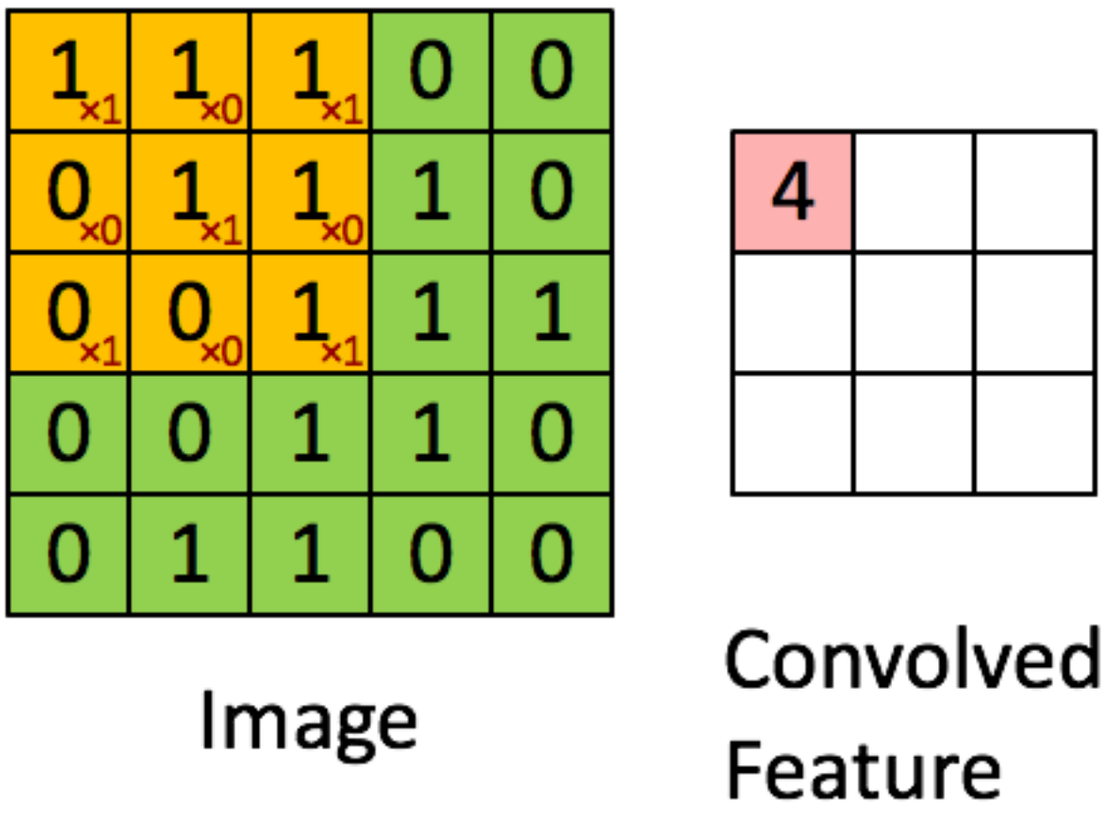
Convolution (합성곱)

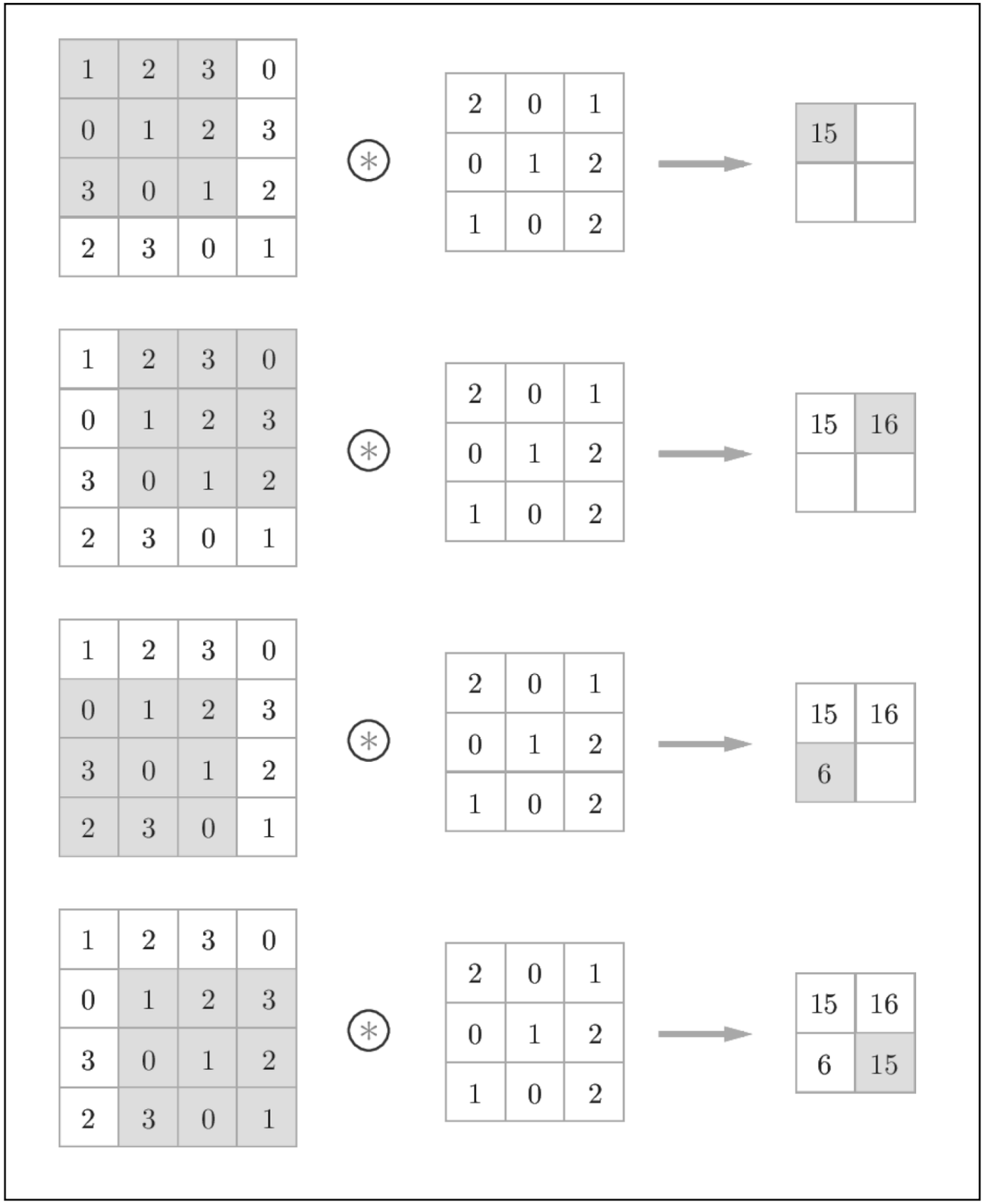
- 합성곱: 이미지 처리에서의 필터 연산

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix} \circledast \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 4 & 3 & 4 \\ 2 & 4 & 3 \\ 2 & 3 & 4 \end{pmatrix}$$

필터 (커널)

입력 데이터



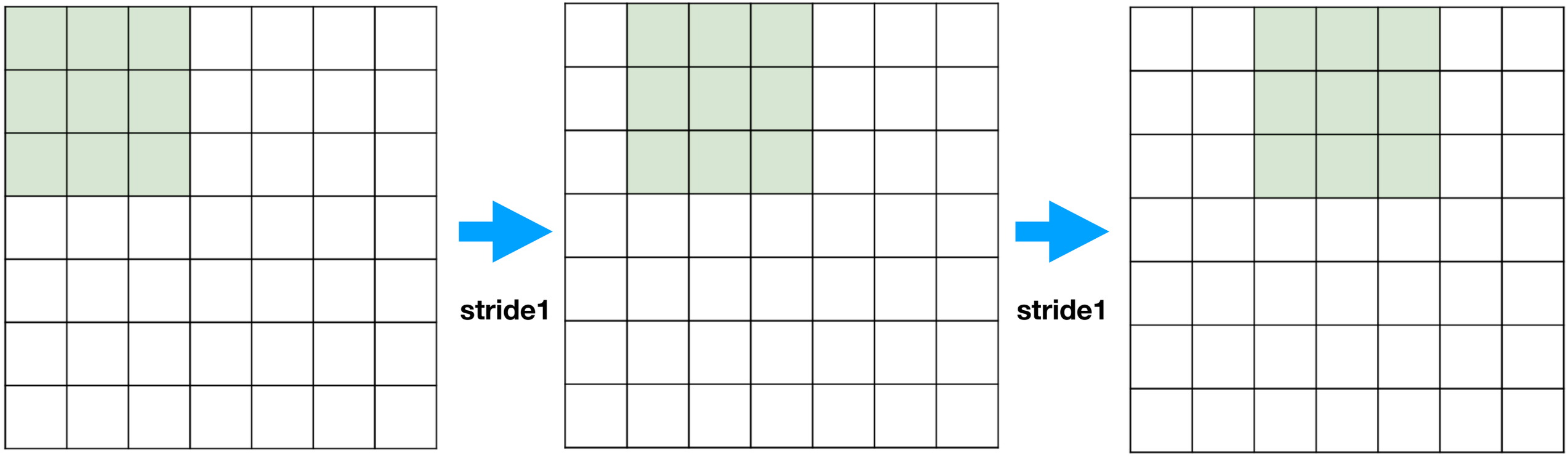


입력 데이터

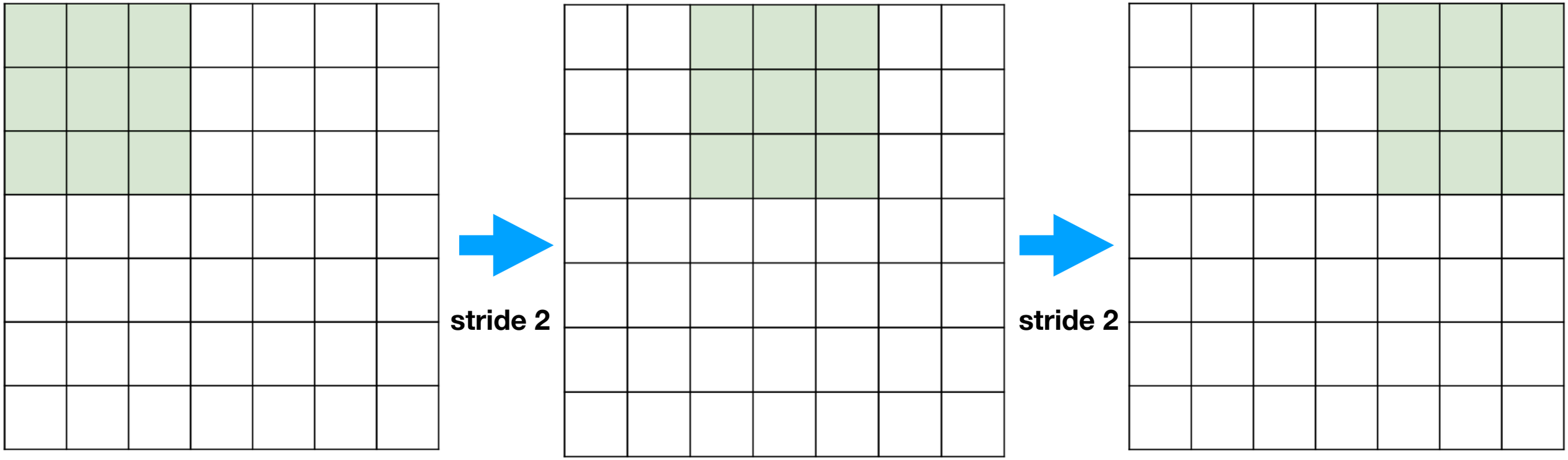
필터

- $(4,4) \circledast (3,3) = (2,2)$

- 스트라이드: 필터를 적용하는 위치의 간격 : (7,7) 이미지에 (3,3) 필터 (커널) 적용

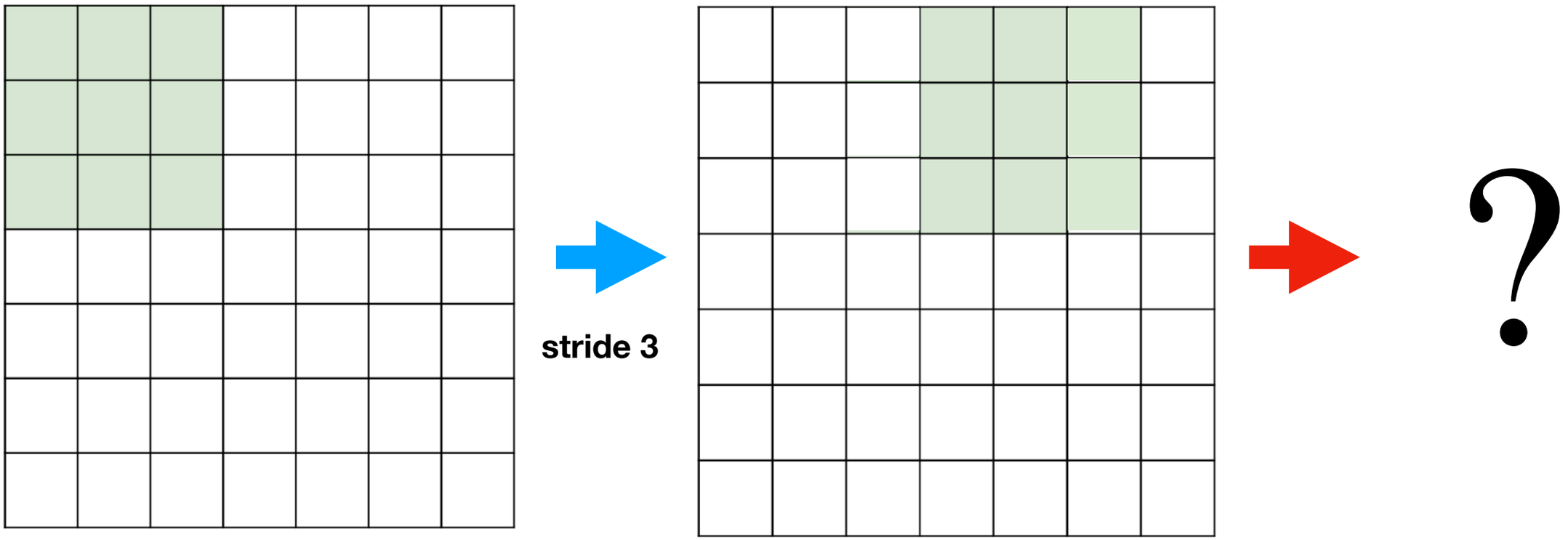


스트라이드 : 1 인 경우: 최종 이미지 = (5,5)

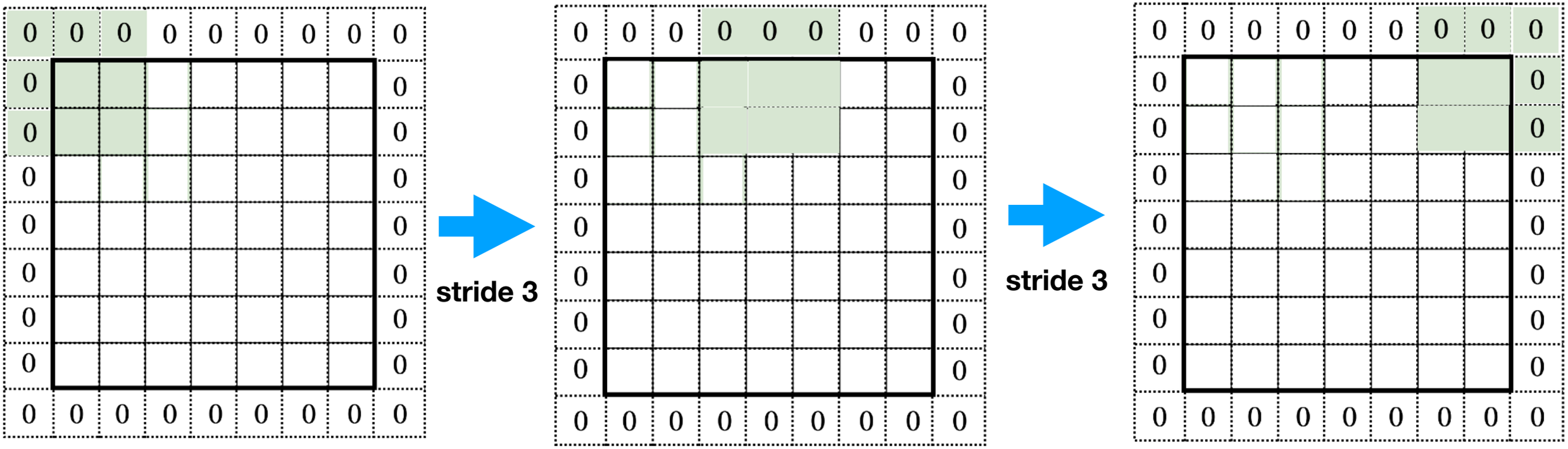


스트라이드 : 2 인 경우: 최종 이미지 = (3,3)

- 입력 데이터에 다양한 크기의 필터를 적용할 때 문제점:



- 패딩: 필터(커널) 연산을 하기 위해 주변을 넓히는 작업



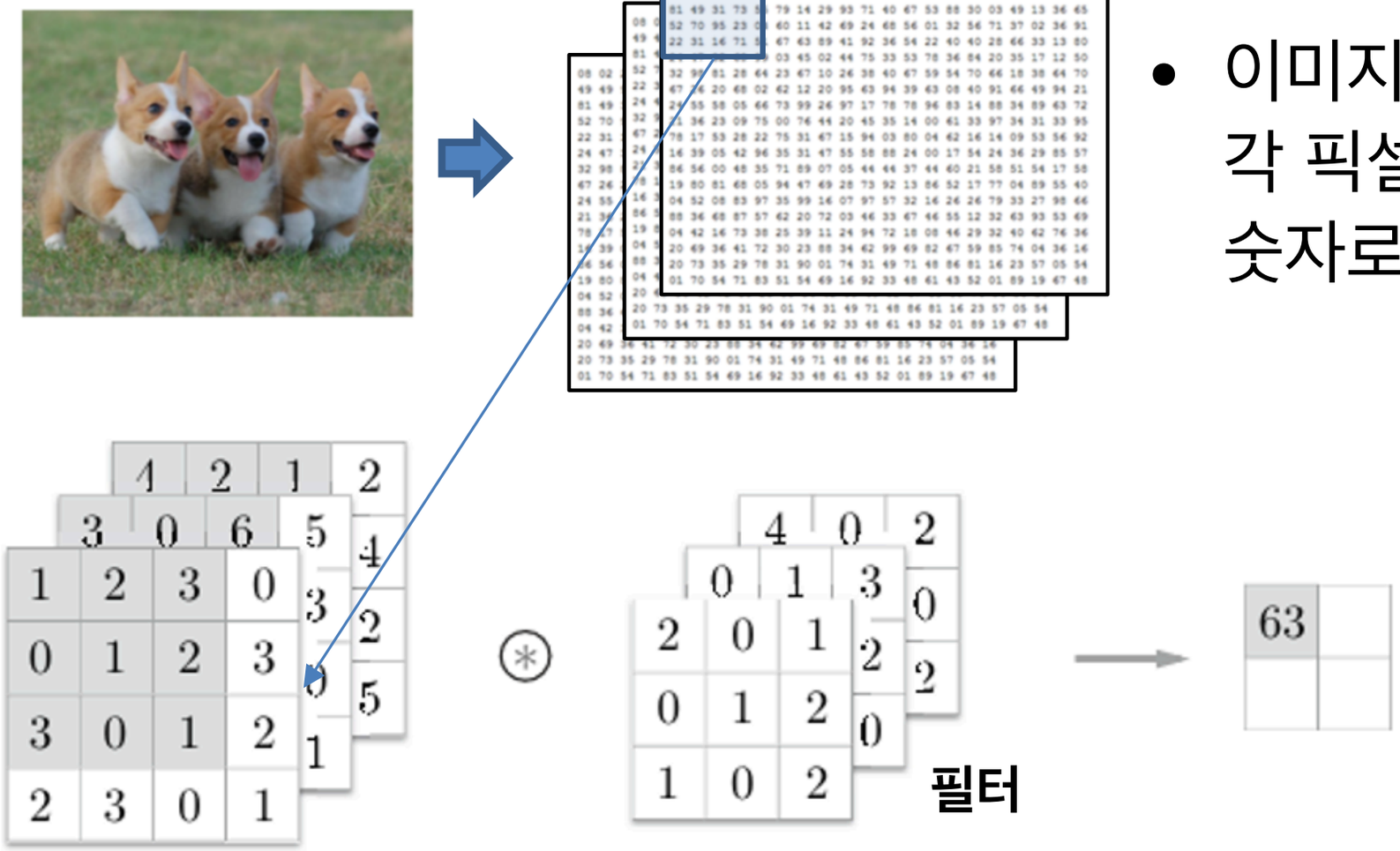
- 입력 데이터 (H, W) , 필터 크기 (F_H, F_W), 패딩 크기 P, 스트라이드 S 인 경우

  출력 크기 (O_H, O_W) 는 다음과 같음.

$$O_H = \frac{(H + 2P) - F_H}{S} + 1$$

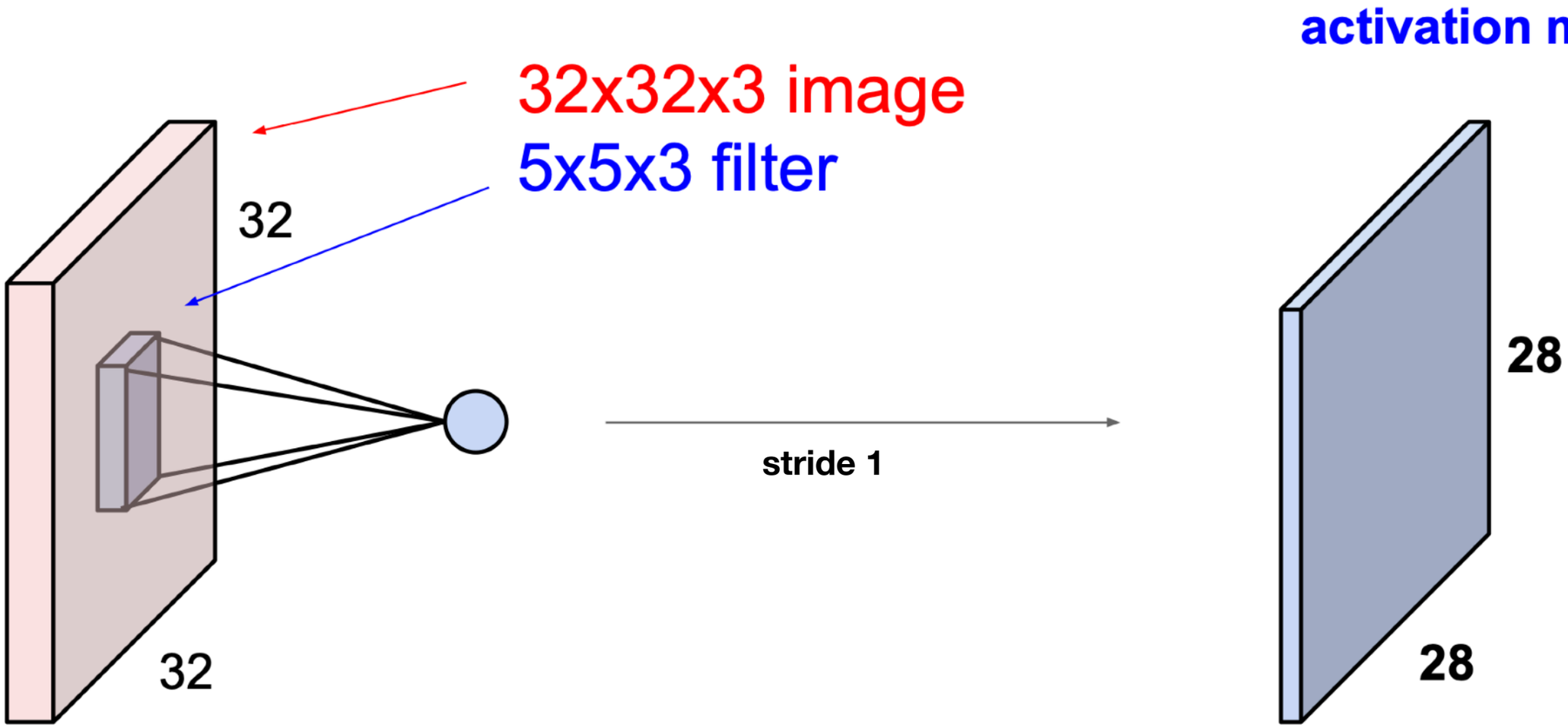
$$O_W = \frac{(W + 2P) - F_W}{S} + 1$$

- 3차원 데이터의 합성곱 연산 : (예: Color image)



- 이미지를 각 채널 (R, G, B)에서, 각 픽셀의 intensity (0-255)의 숫자로 데이터화 시킴

$(1\times2 + 2\times0+3\times1+ 0\times0+1\times1+2\times2+3\times1+0\times0+1\times2) + \cdots = 63$



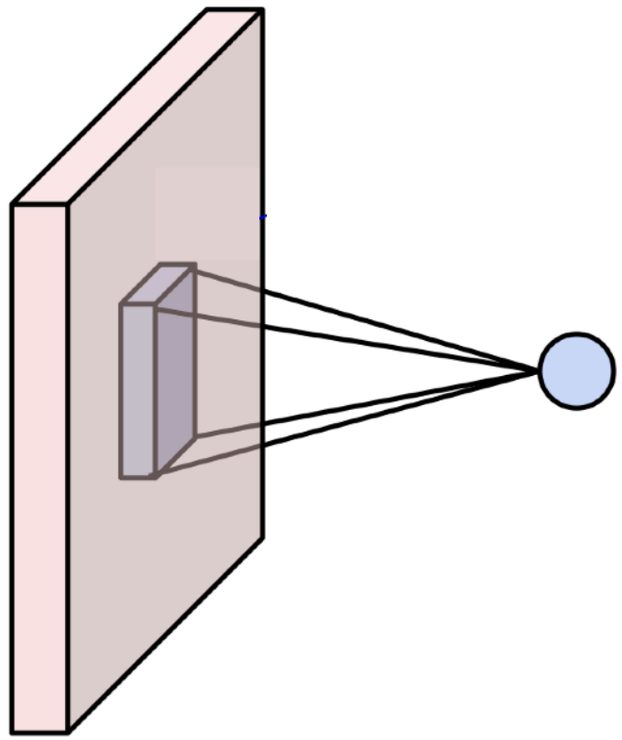
3 1

$$O_H = \frac{(H + 2P) - F_H}{S} + 1 \rightarrow \frac{(32 + 2 \times 0) - 5}{1} + 1 = 28$$

$$O_W = \frac{(W + 2P) - F_W}{S} + 1 \rightarrow \frac{(32 + 2 \times 0) - 5}{1} + 1 = 28$$

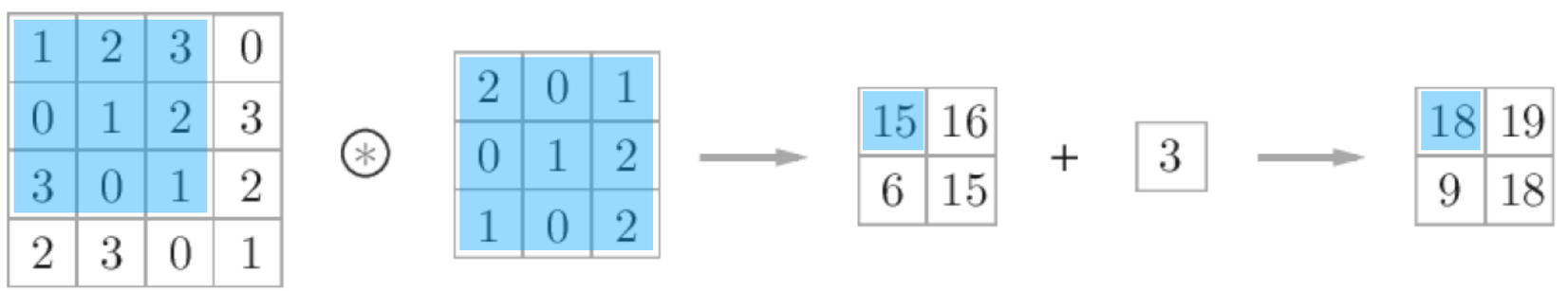
CNN 의 장점

1. 픽셀 사이의 관계를 파악하기 쉬움.

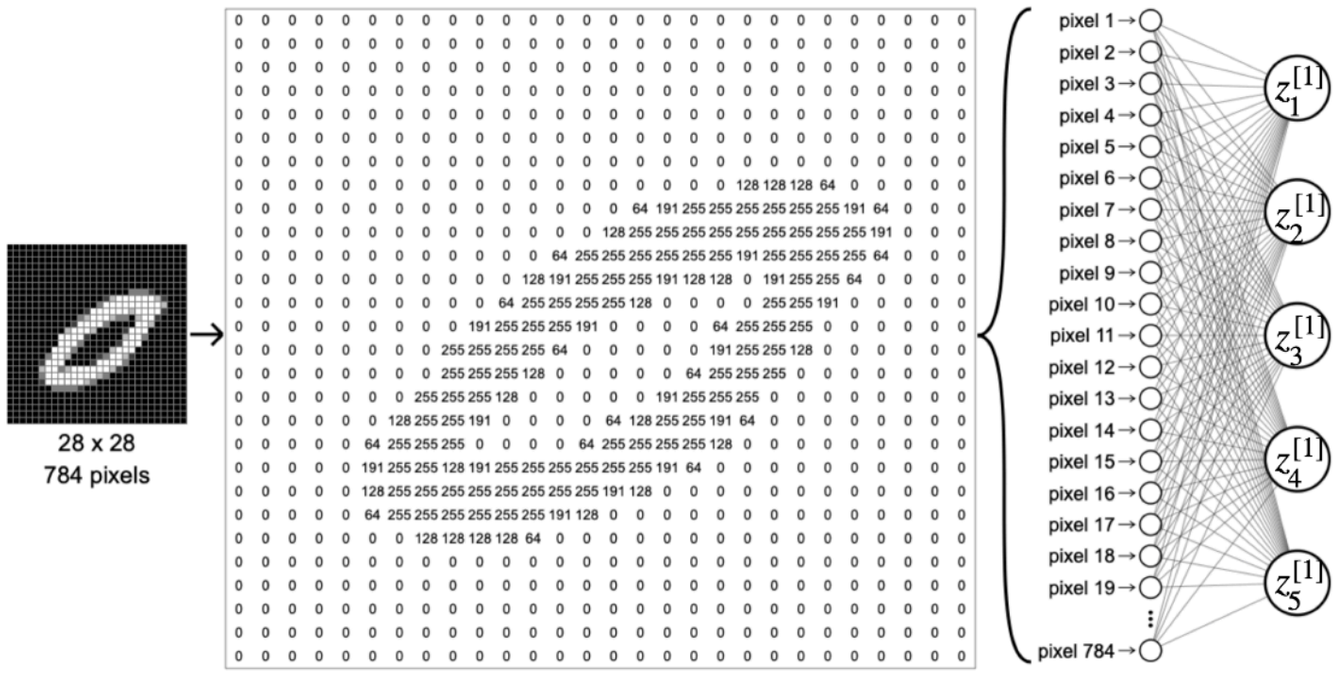


2. 필터를 통한 **매개변수 (가중치) sharing** 으로, 파라메터 갯수의 감소

$$\vec{w} \cdot \vec{x} + b = (w_1, w_2, \cdots, w_9) \cdot (x_1, x_2, \cdots, x_9) + b$$



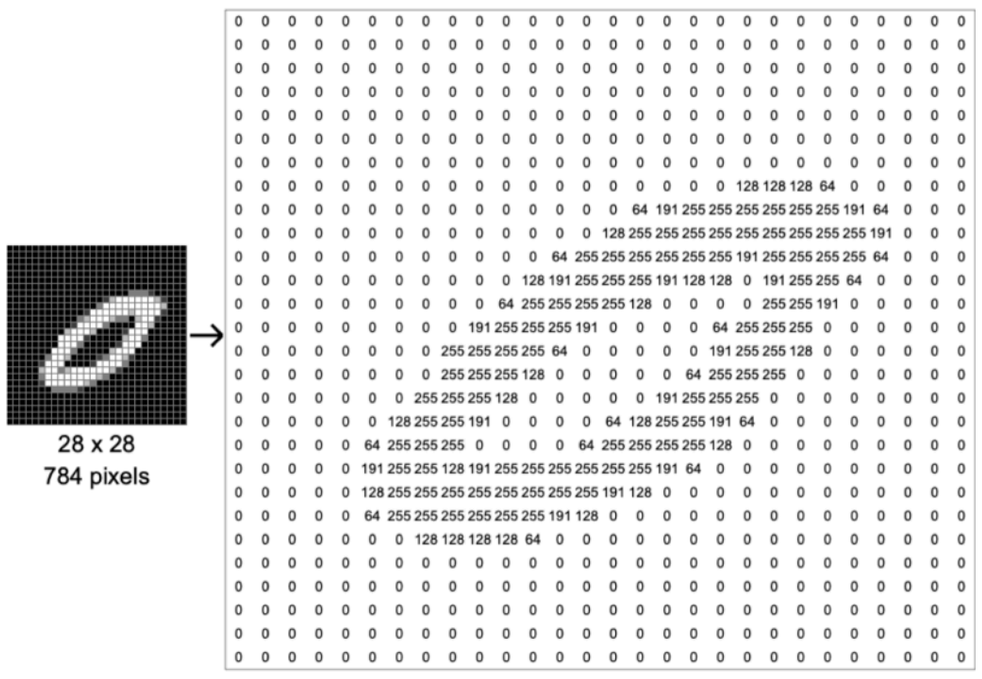
- Fully-Connected DNN



$$z^{[1]} = \begin{pmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \\ z_5^{[1]} \end{pmatrix} = \begin{pmatrix} w_{(1,1)}^{[1]} & w_{(1,2)}^{[1]} & \cdots & w_{(1,784)}^{[1]} \\ w_{(2,1)}^{[1]} & w_{(2,2)}^{[1]} & \cdots & w_{(2,784)}^{[1]} \\ w_{(3,1)}^{[1]} & w_{(3,2)}^{[1]} & \cdots & w_{(3,784)}^{[1]} \\ w_{(4,1)}^{[1]} & w_{(4,2)}^{[1]} & \cdots & w_{(4,784)}^{[1]} \\ w_{(5,1)}^{[1]} & w_{(5,2)}^{[1]} & \cdots & w_{(5,784)}^{[1]} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{784} \end{pmatrix} + \begin{pmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \\ \vdots \\ b_{784}^{[1]} \end{pmatrix}$$

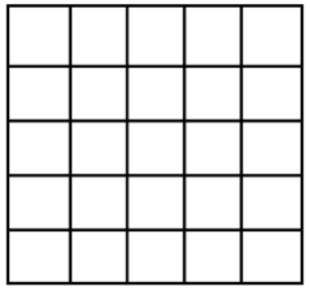
$$784 \times 5 + 784 = 4{,}704$$

- Convolutional Neural Network (CNN)



$$O_H = O_W = \frac{(28 + 2 \times 0) - 24}{1} + 1 = 5$$

$$\circledast\ (24,24) =$$



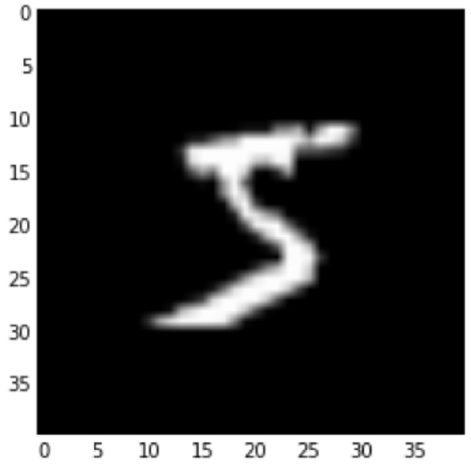
(5,5)

- 필터 크기 (가중치)
  $= 24 \times 24 = 576$
- 매개변수 개수
  $= 576 + 1 = 577$

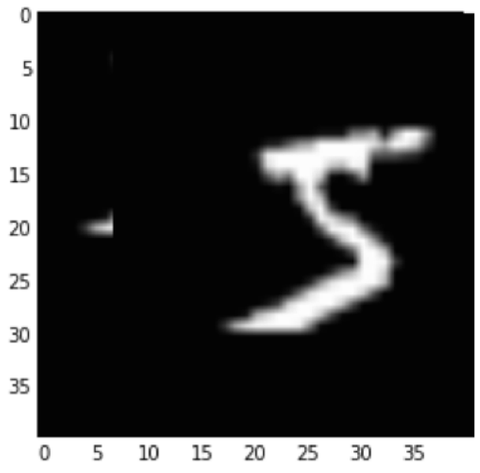
CNN 의 장점

- Parameter sharing (stride) 를 통한 이미지 이동에 대한 이해

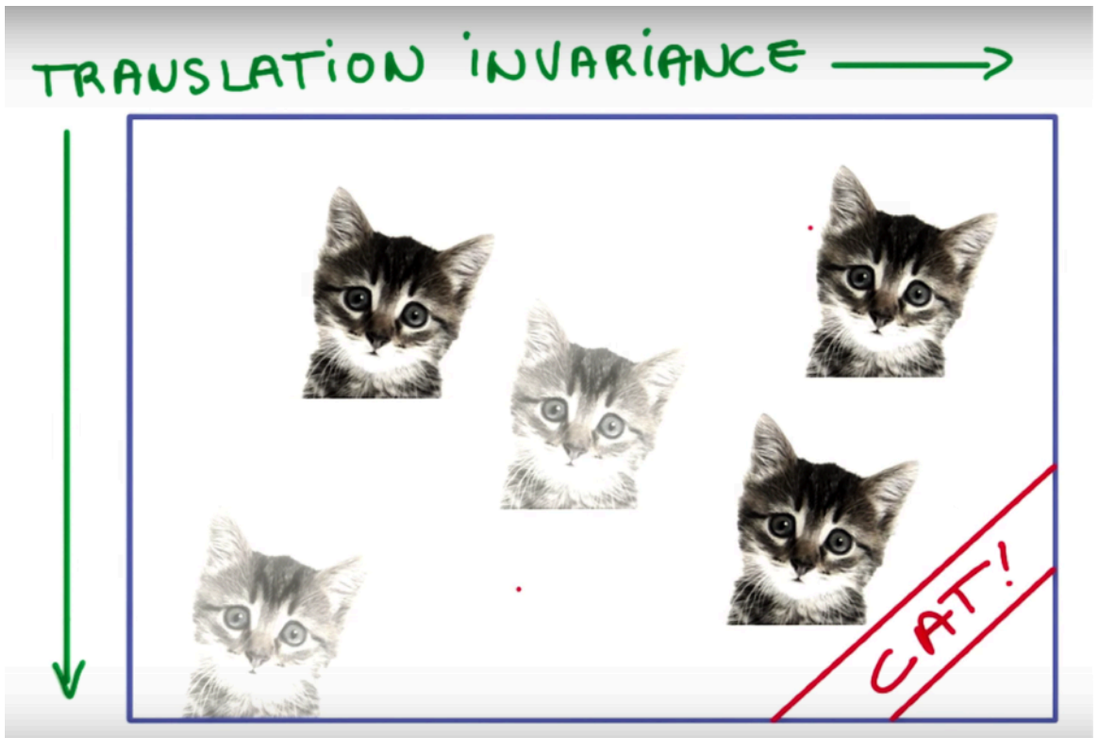
학습



테스트

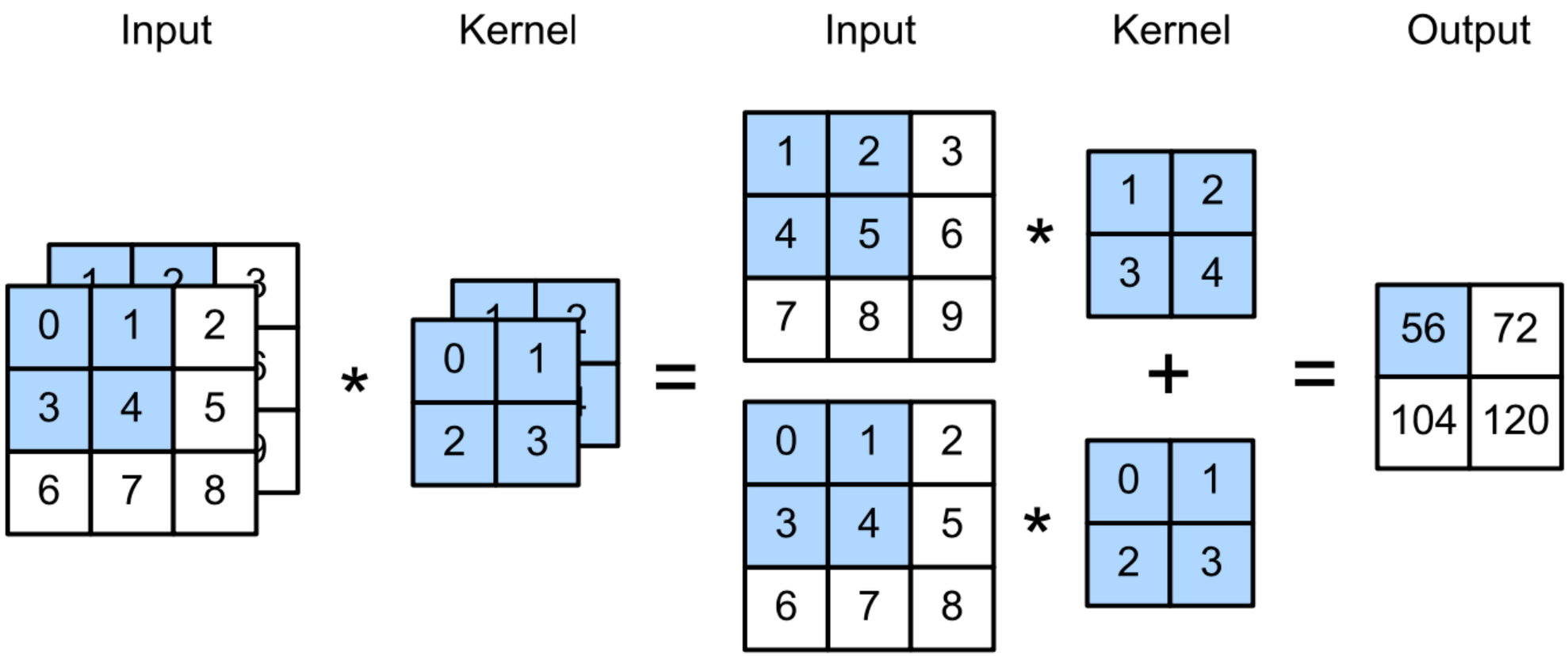


- DNN: ???
- CNN: 5



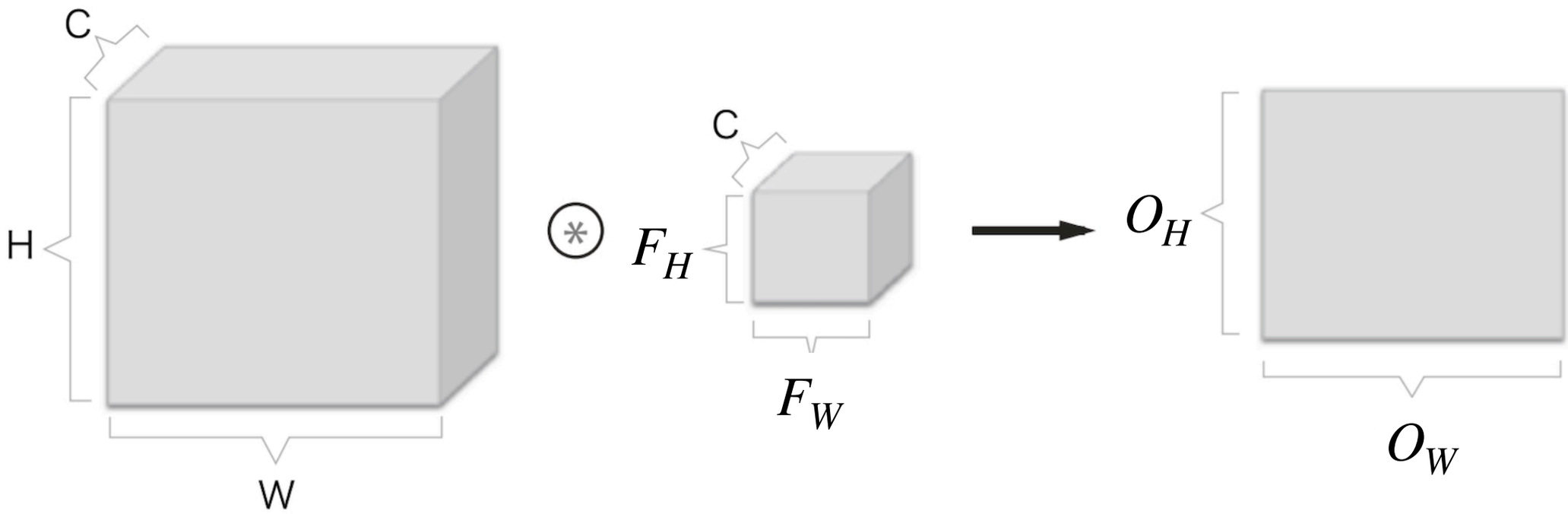
- CNN을 이용하여 학습하는 경우,
  타겟의 위치에 상관없이 타겟을 이해

- 일반적인 합성곱 연산



입력 데이터와 필터의 채널 갯수는 같다

- 일반적인 합성곱 연산



입력 데이터

(채널, 높이, 너비)

= (C, H, W)

필터

(채널, 높이, 너비)

= (C, F_H, F_W)

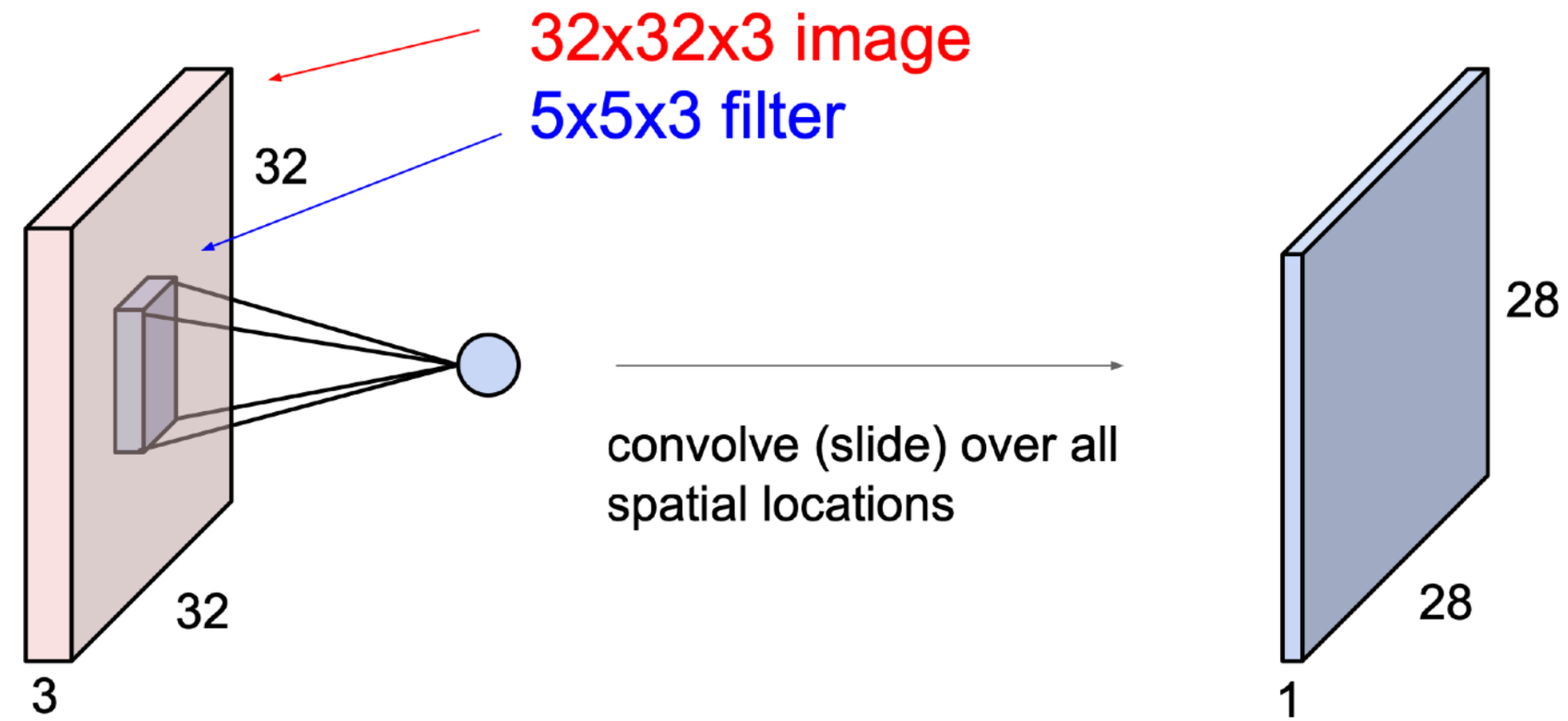
출력 데이터

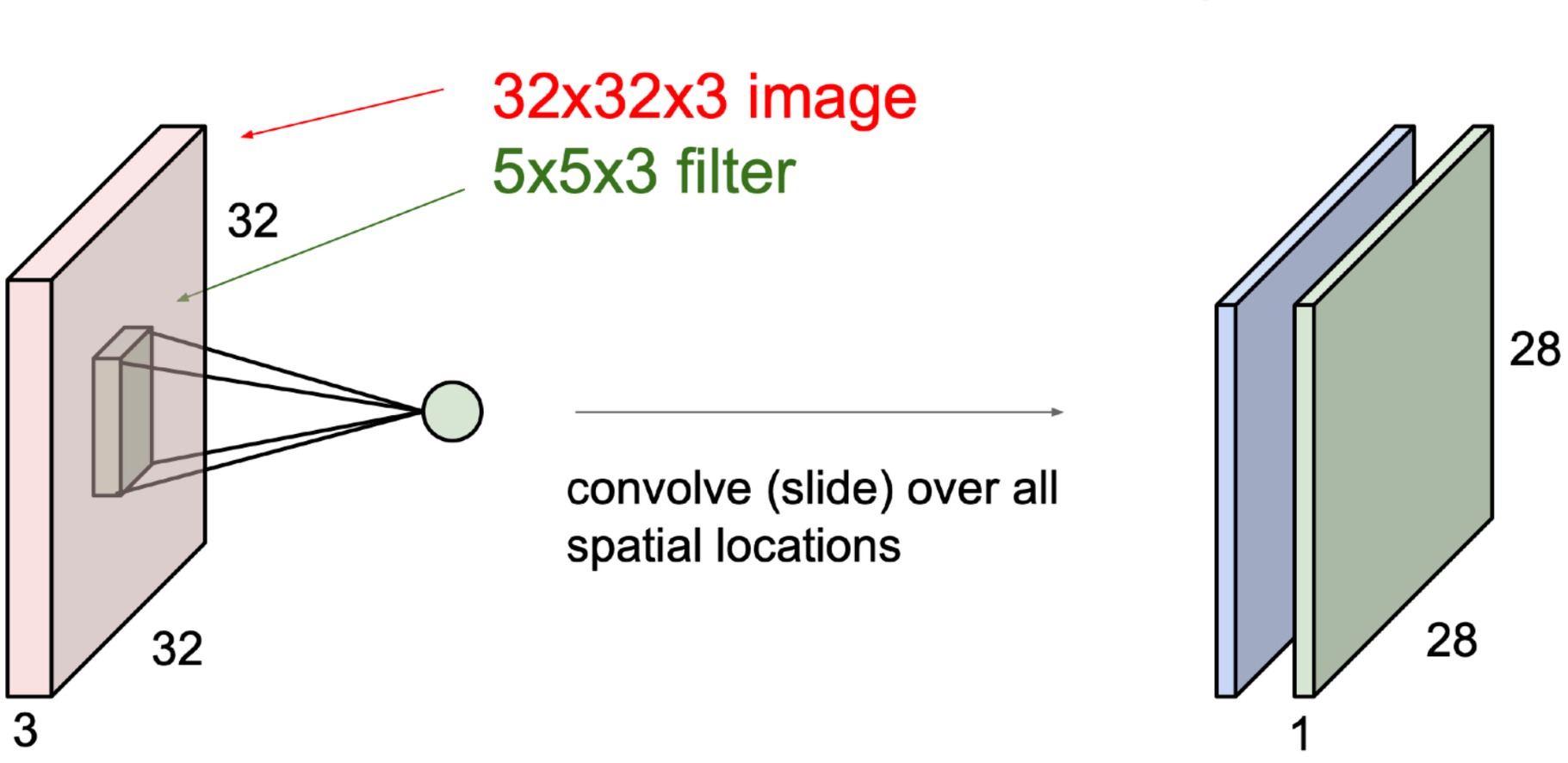
(채널, 높이, 너비)

= (1, O_H, O_W)

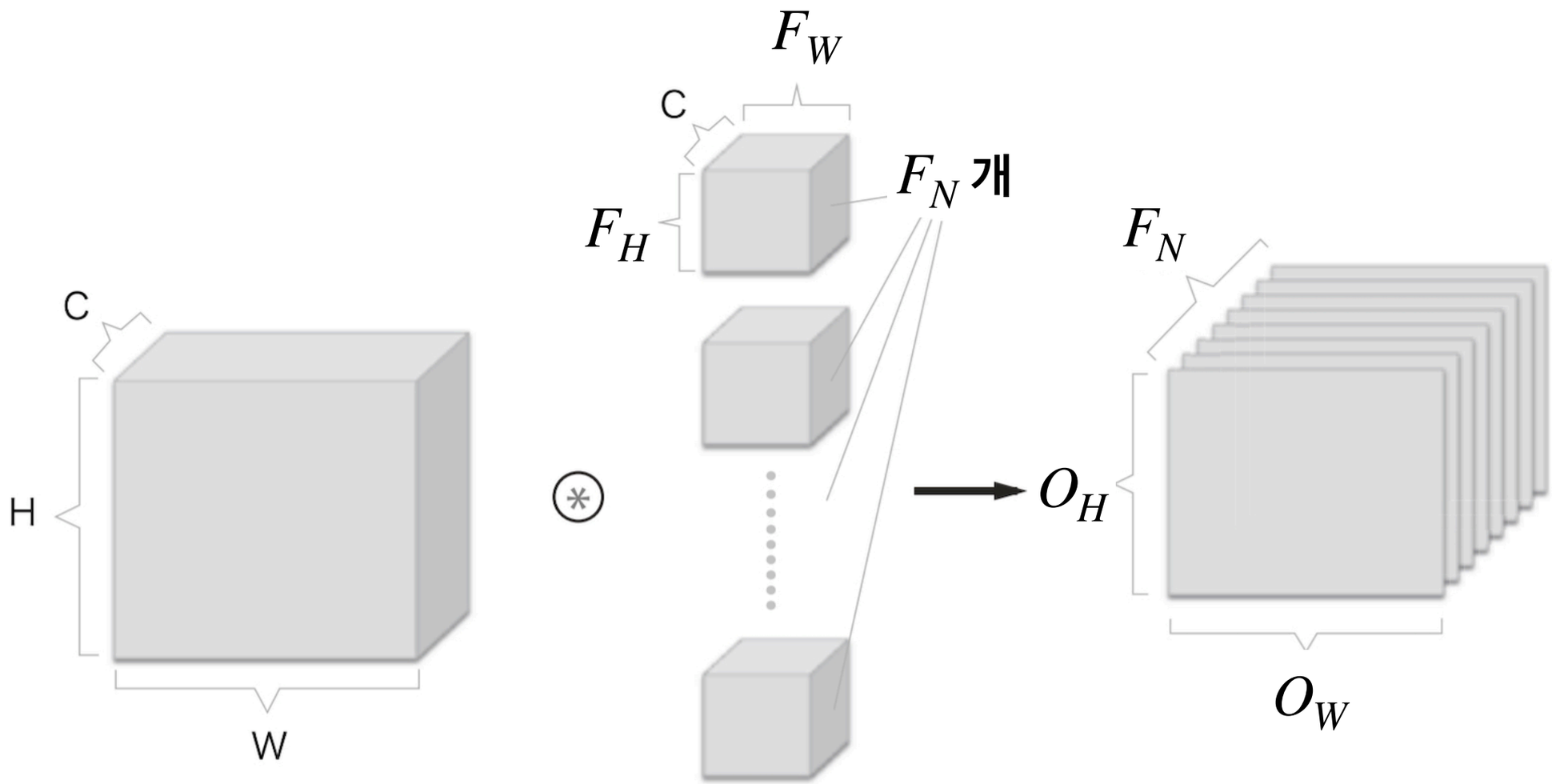
입력 데이터와 필터의 채널 갯수는 같다

- **여러개의 필터**를 적용한 경우

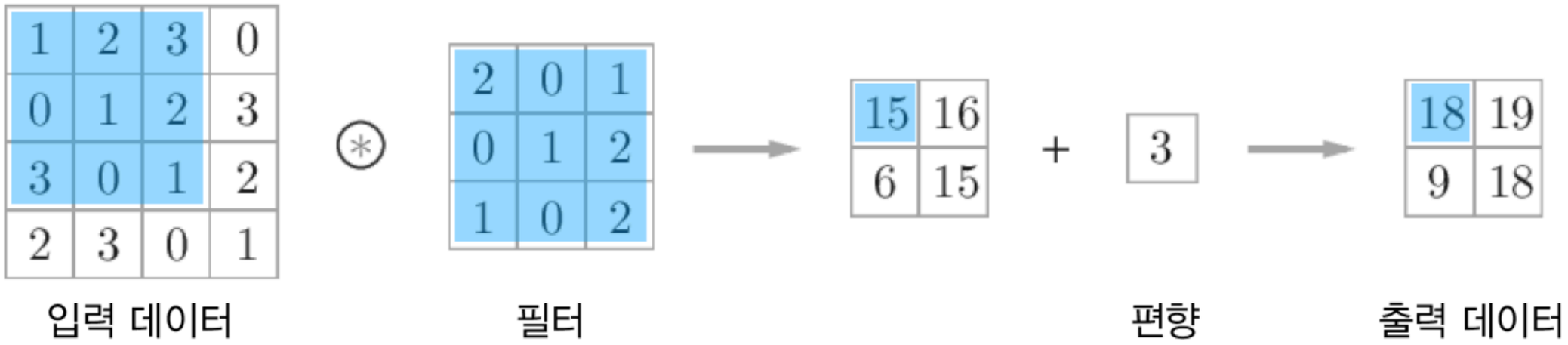




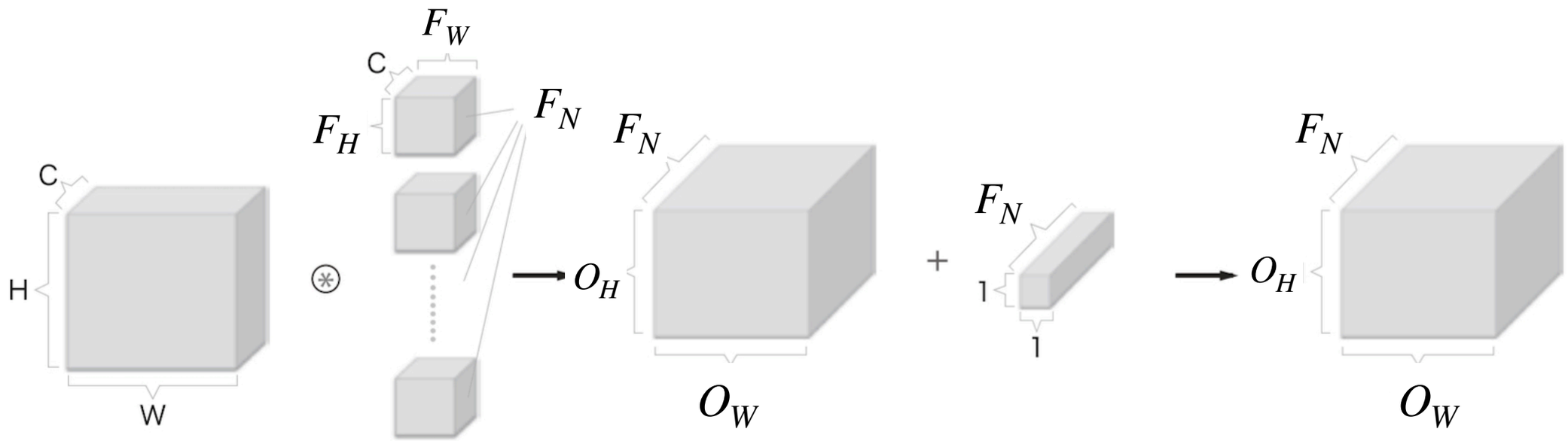
- 여러개의 필터를 적용한 경우



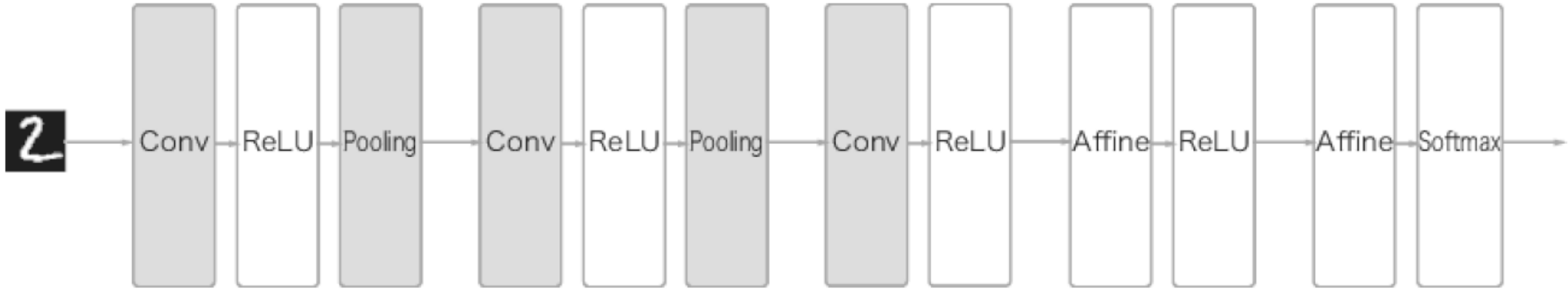
F_N 개의 필터를 적용하면, 출력 데이터도 F_N 개가 생성됨.



- 일반적인 Convolution 의 경우 : 가중치와 편향

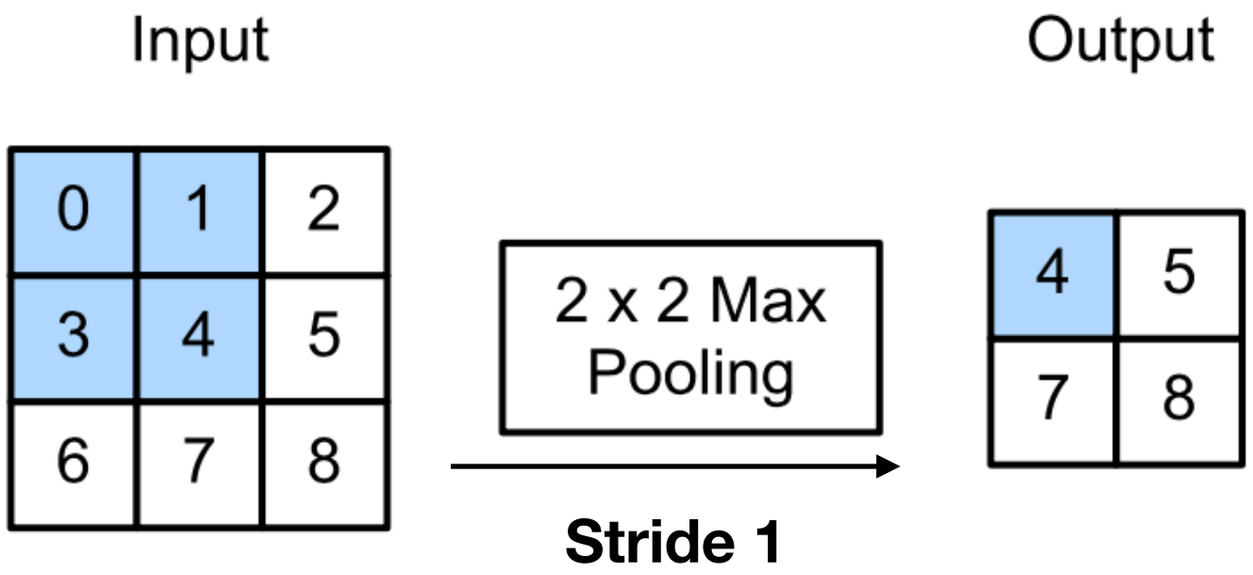


$$(C, H, W) \circledast (F_N, C, F_H, F_W) = (F_N, O_H, O_W) + (F_N, 1, 1) = (F_N, O_H, O_W)$$

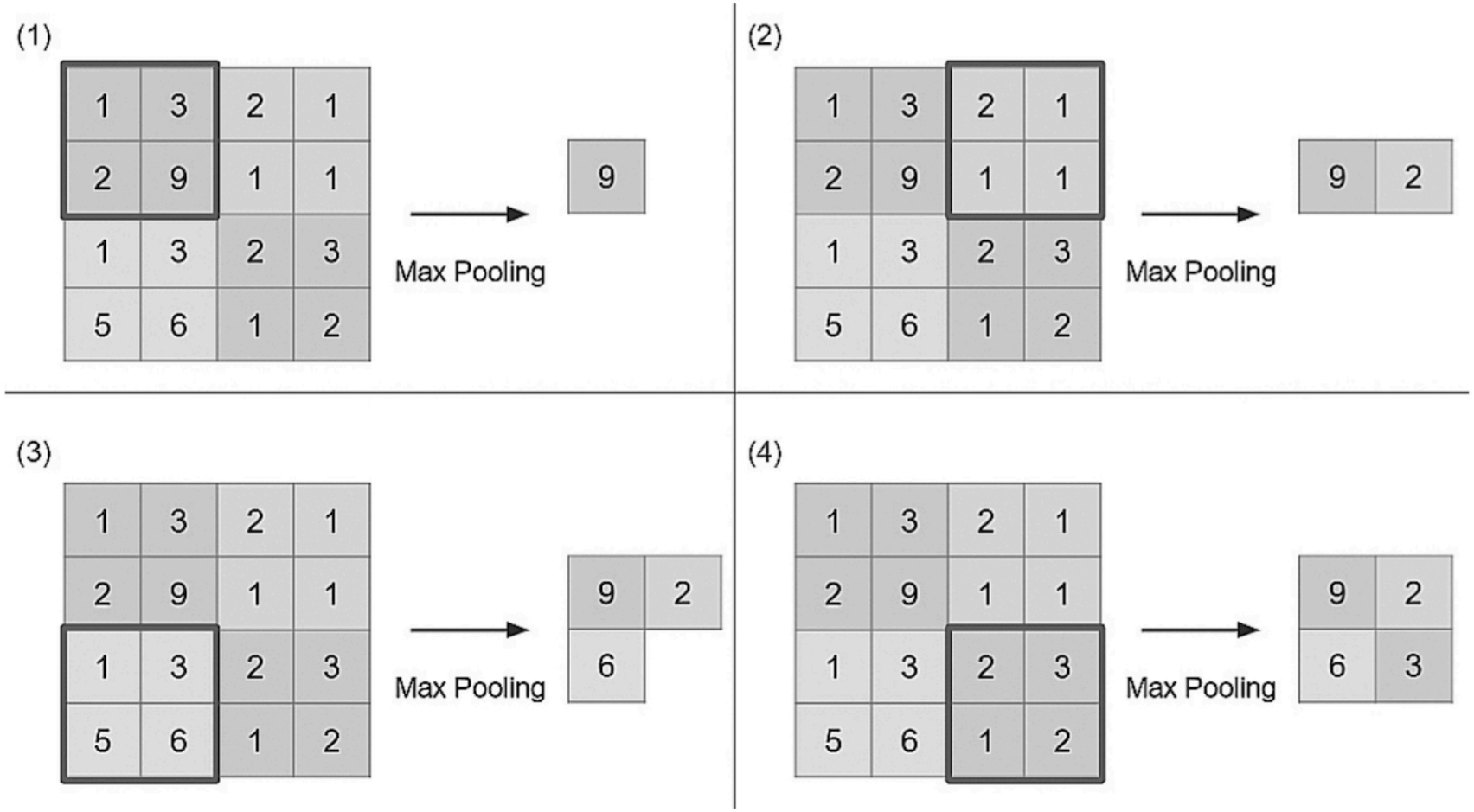


Pooling (풀링)

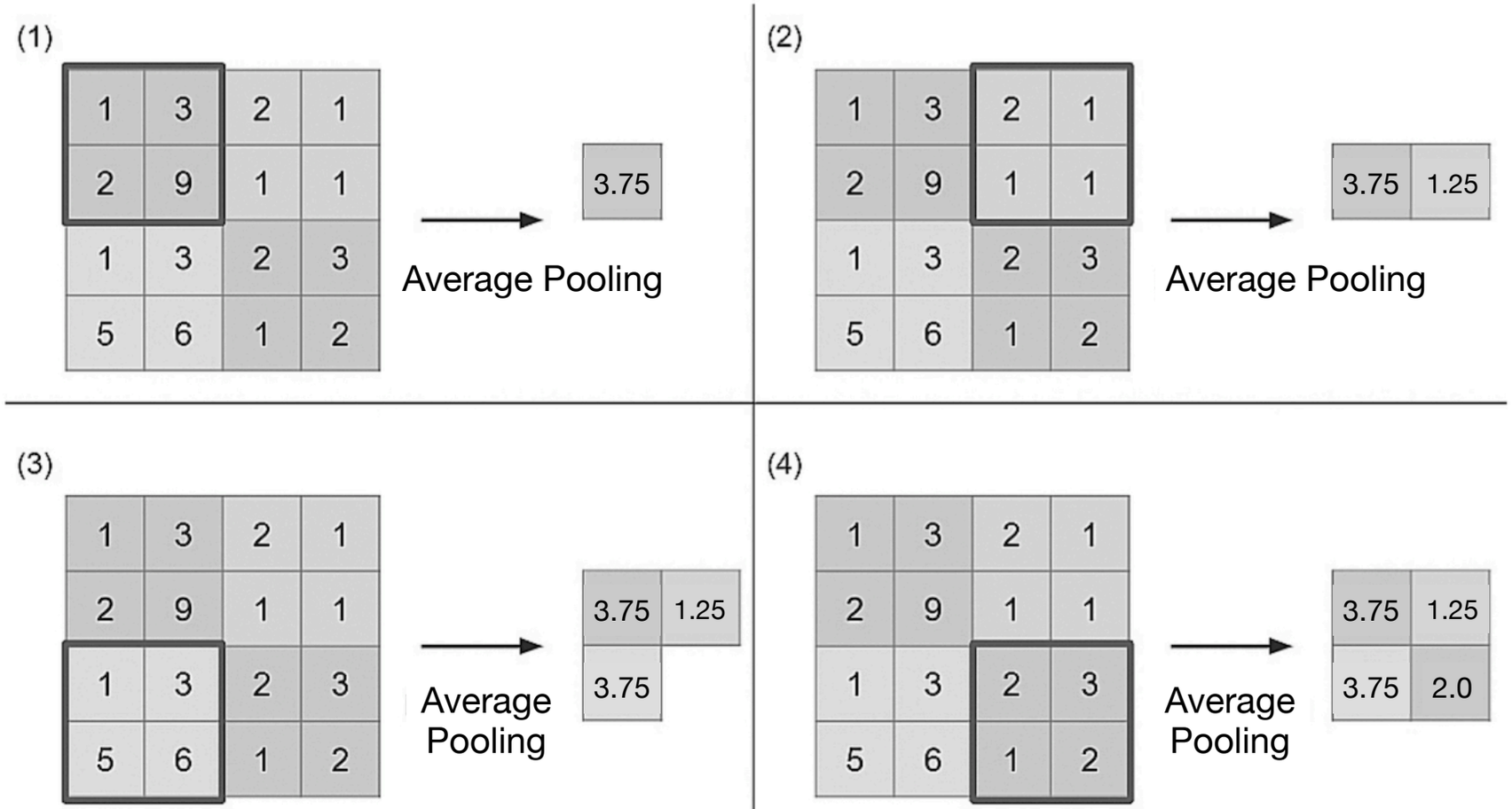
- 풀링: 데이터의 크기를 줄이는 연산
  - 최대 풀링 (Max Pooling)
  - 평균 풀링 (Average Pooling)



- 최대 풀링

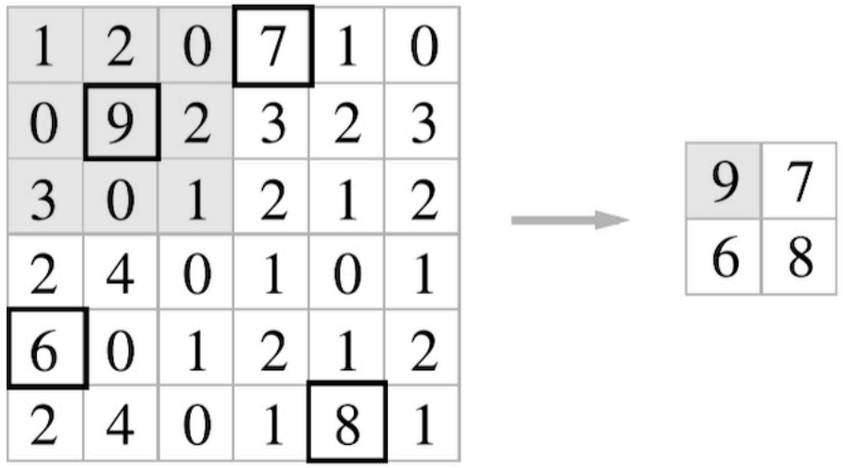

(1)
1 3 2 1
2 9 1 1
1 3 2 3
5 6 1 2
Max Pooling
9
(2)
1 3 2 1
2 9 1 1
1 3 2 3
5 6 1 2
Max Pooling
9 2
(3)
1 3 2 1
2 9 1 1
1 3 2 3
5 6 1 2
Max Pooling
9 2
6
(4)
1 3 2 1
2 9 1 1
1 3 2 3
5 6 1 2
Max Pooling
9 2
6 3

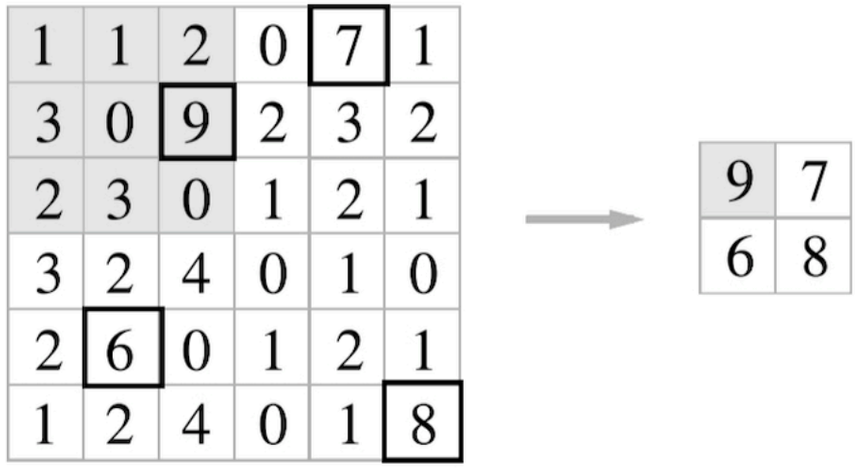
- 평균 풀링


(1)
1 3 2 1
2 9 1 1
1 3 2 3
5 6 1 2
Average Pooling
3.75
(2)
1 3 2 1
2 9 1 1
1 3 2 3
5 6 1 2
Average Pooling
3.75 1.25
(3)
1 3 2 1
2 9 1 1
1 3 2 3
5 6 1 2
Average Pooling
3.75 1.25
3.75
(4)
1 3 2 1
2 9 1 1
1 3 2 3
5 6 1 2
Average Pooling
3.75 1.25
3.75 2.0

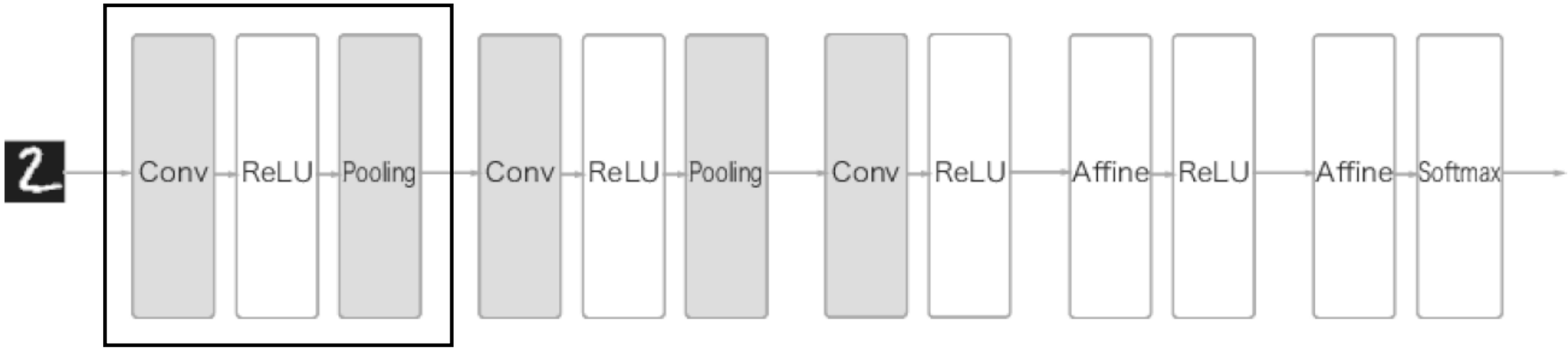
- 풀링의 특징

1. 학습해야 할 매개변수가 없음.
2. 채널 수가 변하지 않음.
3. 입력의 변화에 영향을 적게 받음 (국소 변화에 강함)

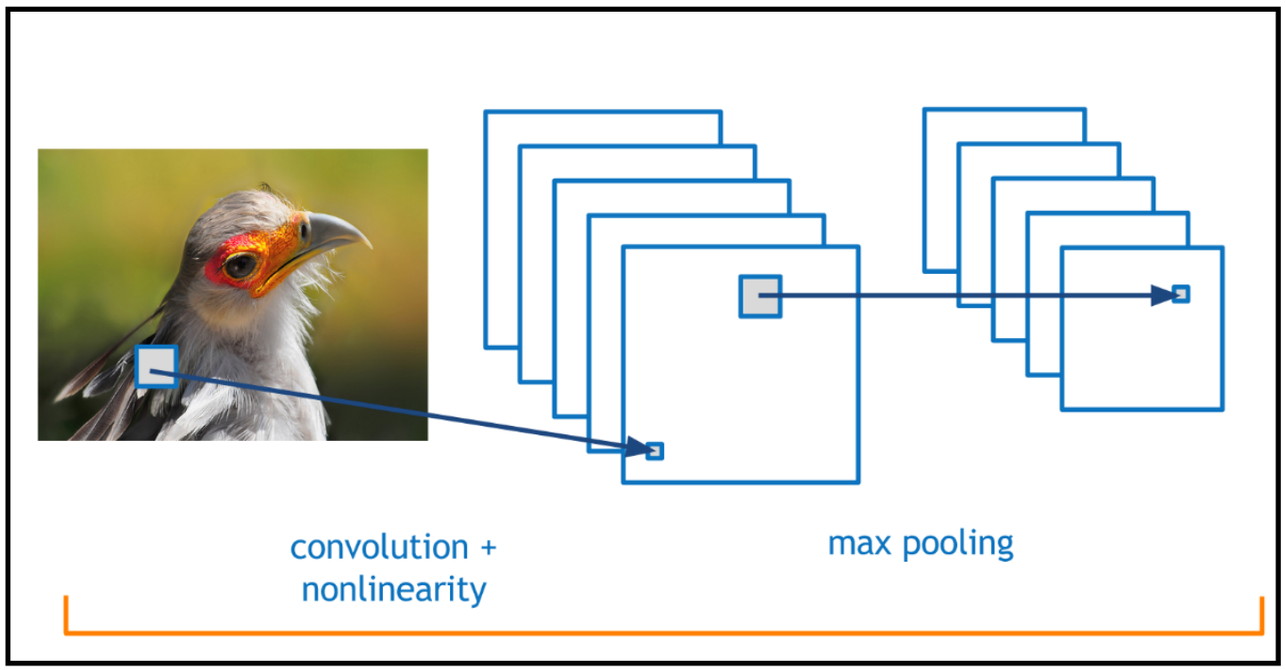


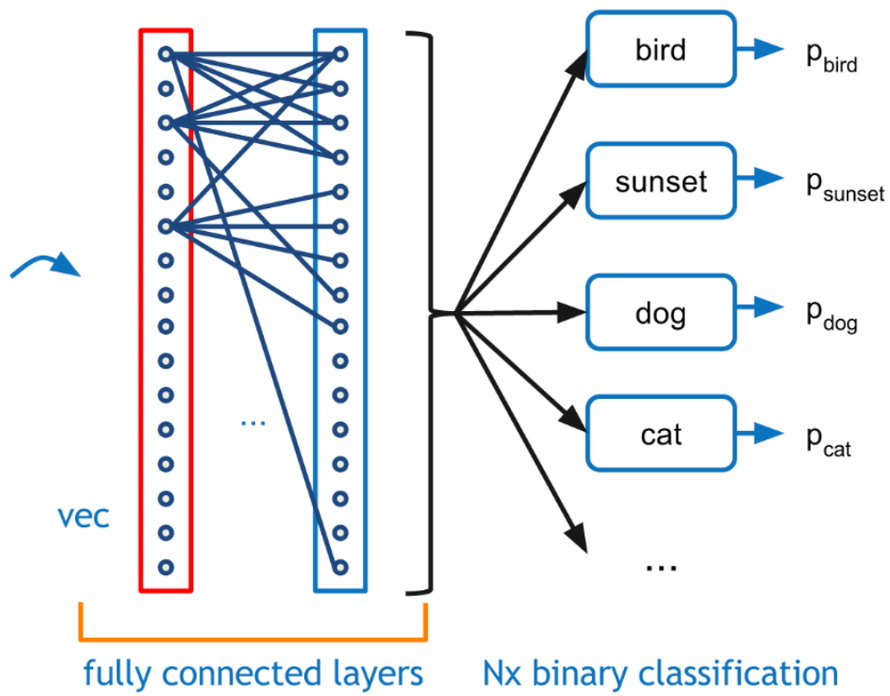


- 신경망이 학습하는 함수가 **"국소 이동"에 대해 불변**인 경우에 풀링을 적용하여야 학습 효과를 얻을 수 있음.



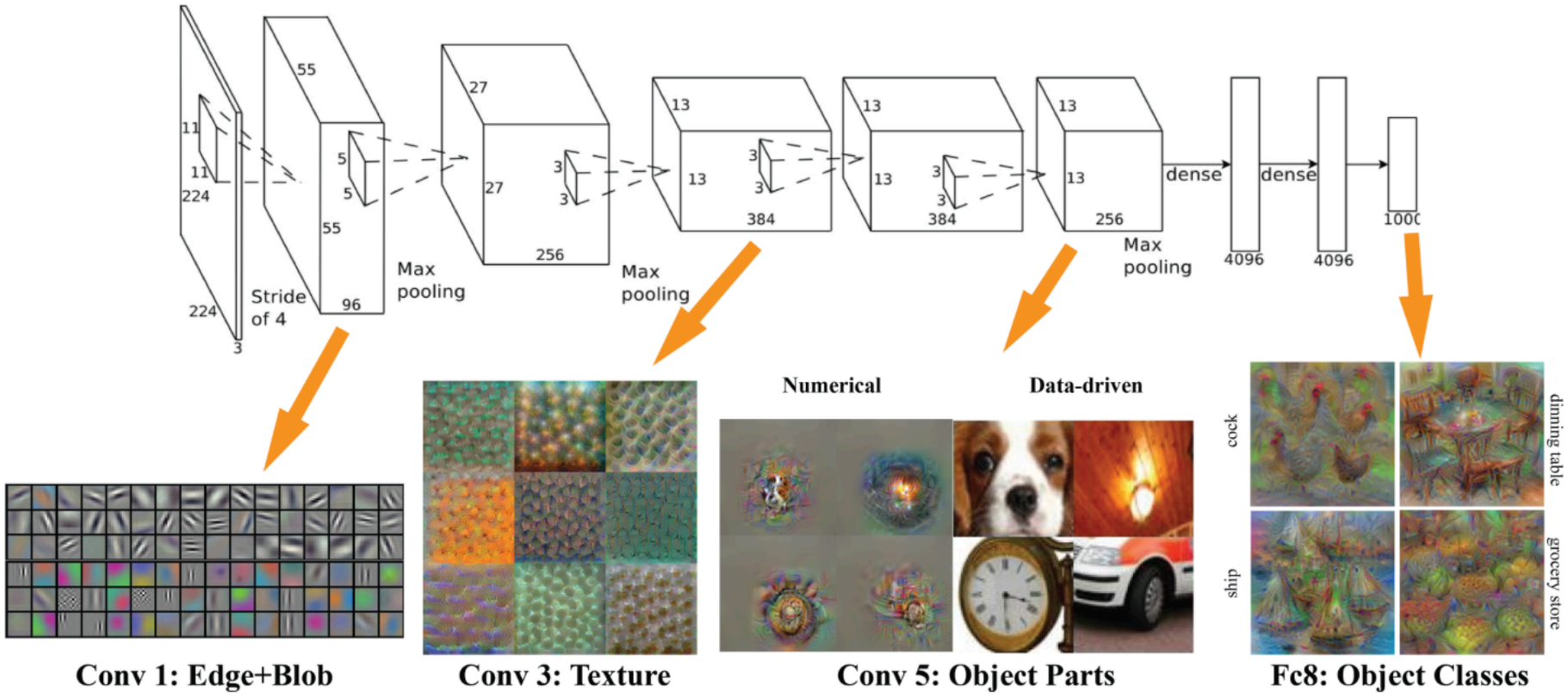
Convolution Layer





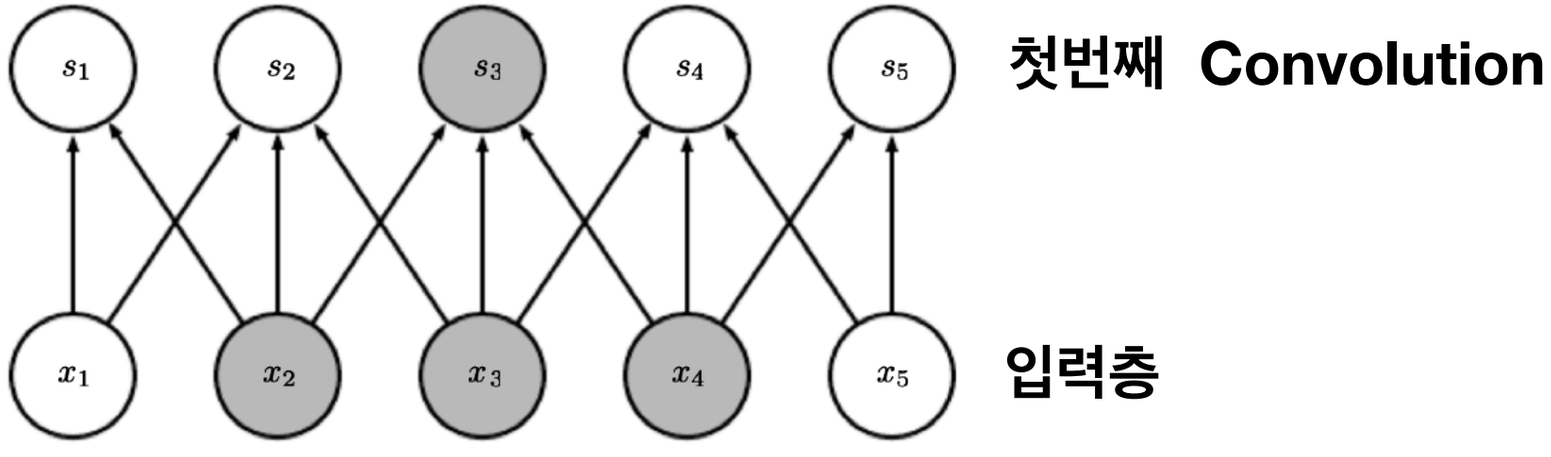
Convolution Layer

- CNN 은 층이 깊어질 수록, 더욱 추상적인 내용을 이해함.



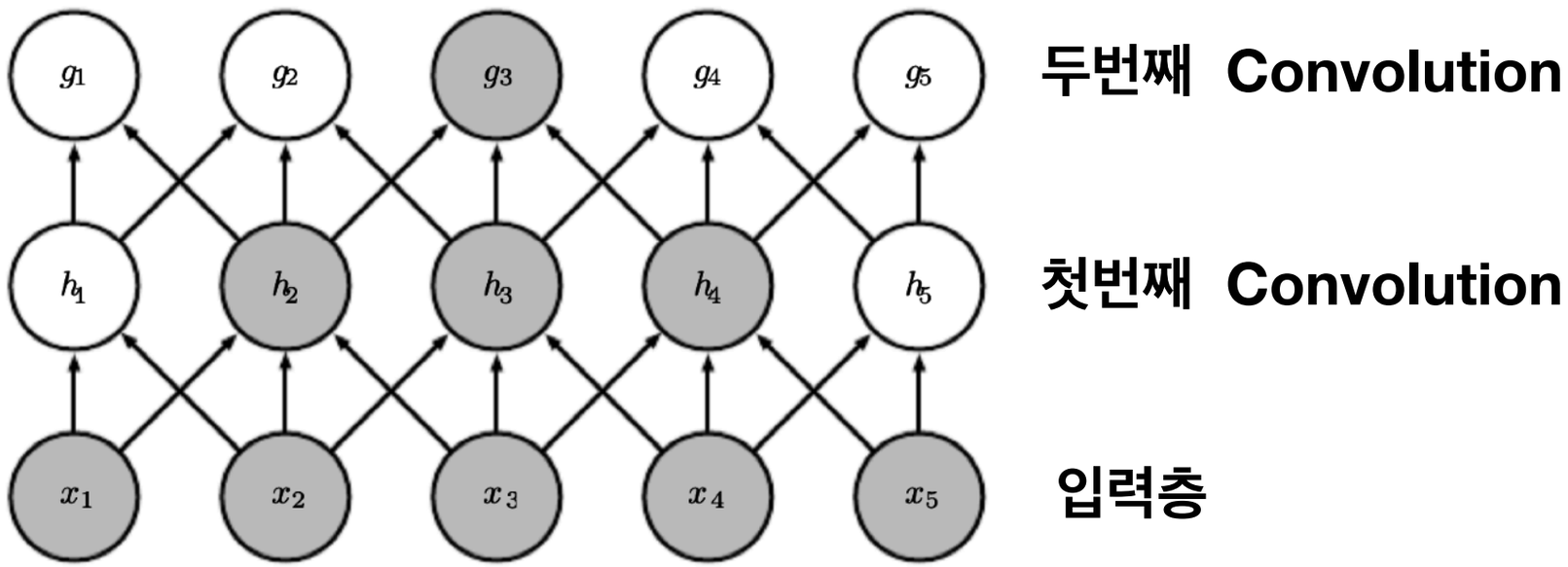
- 희소 상호작용 (Sparse Interaction)

  - **필터의 크기가 3인 경우**

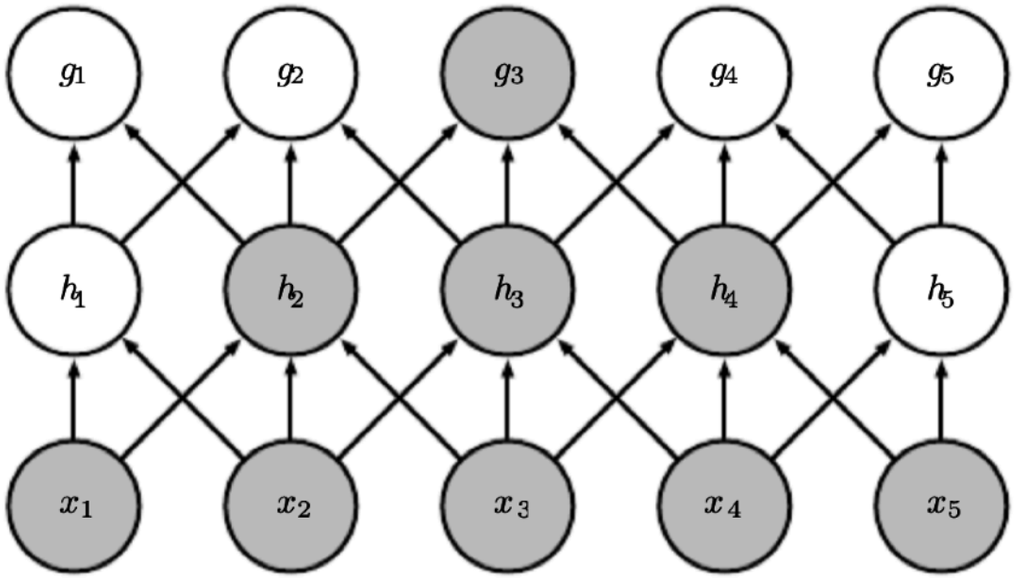


  - **필터의 크기가 3인 경우**

  - **필터의 크기가 3인 경우**



- 필터의 크기가 3인 경우

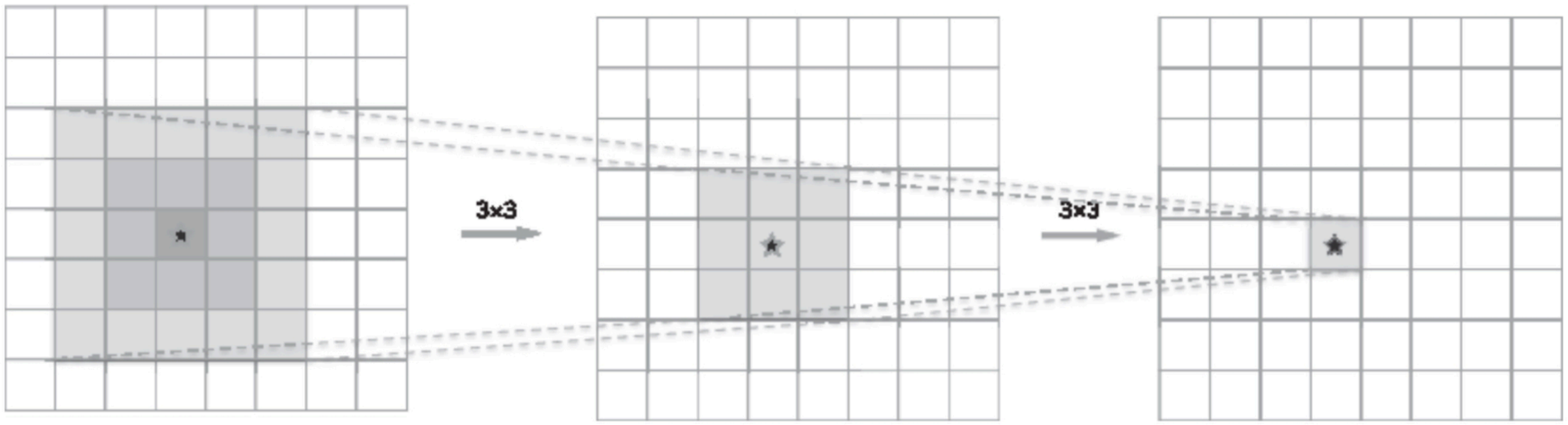


두번째 Convolution

첫번째 Convolution

입력층

- 필터의 크기가 3인 경우

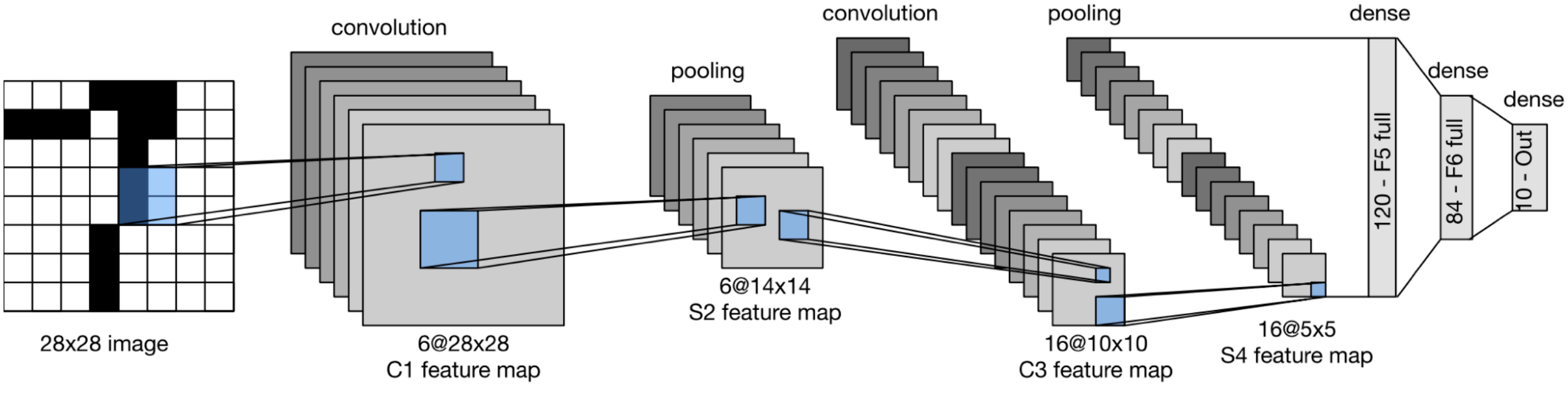


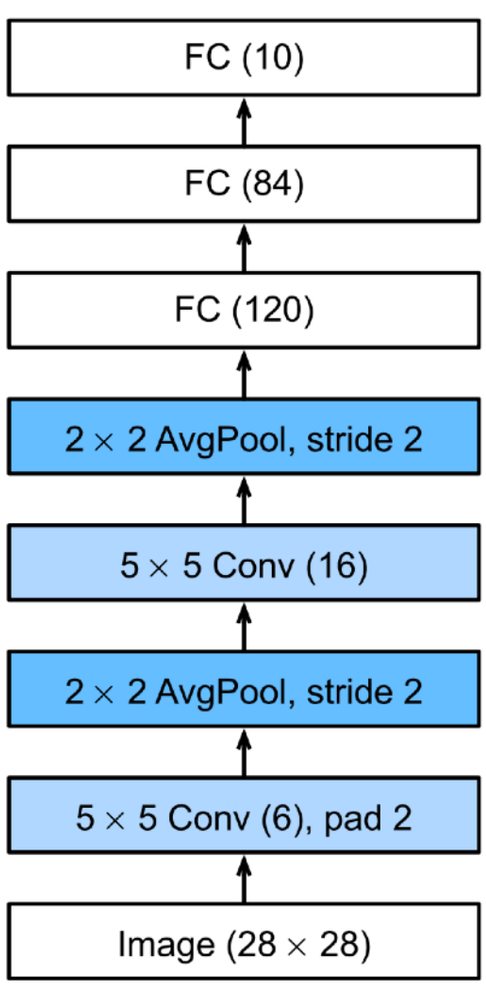
입력층

첫번째 Convolution

두번째 Convolution

- CNN의 예: LeNet (손글씨 숫자 인식 네트워크: 1998)





- Convolutional layer에 비선형맵(activation)으로 sigmoid 함수 사용.
  Pooling에 Average Pooling사용

- 1번째 Convolutional layer에 $F_N = 6$ 개의 $(1,5,5)$ 필터 적용.
  2번째에는 $F_N = 16$ 개의 $(6,5,5)$ 필터가 적용됨.
  $$O_H = O_W = \frac{28 + 2 \cdot 2 - 5}{1} + 1 = 28, \; O_H = O_W = \frac{14 + 2 \cdot 0 - 5}{1} + 1 = 10$$

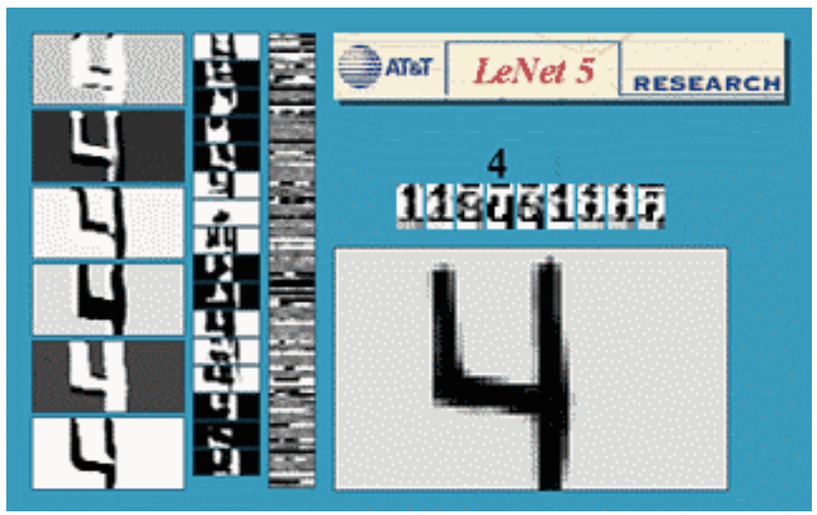
- Pooling은 $(2,2)$ 커널이 stride-2로 적용됨.
  $$O_H = O_W = \frac{28 + 2 \cdot 0 - 2}{2} + 1 = 14, \; O_H = O_W = \frac{10 + 2 \cdot 0 - 2}{2} + 1 = 5$$

- 2D에서 1D 로 변환할 때, $(16,5,5) \rightarrow (120,1,1)$ 즉 $F_N = 120$ 개의 $(16,5,5)$ 필터 적용

- 이후 Fully-connected NN 로 $120 \rightarrow 84 \rightarrow 10$ 로 노드 갯수 감소 (최종 10개 노드)

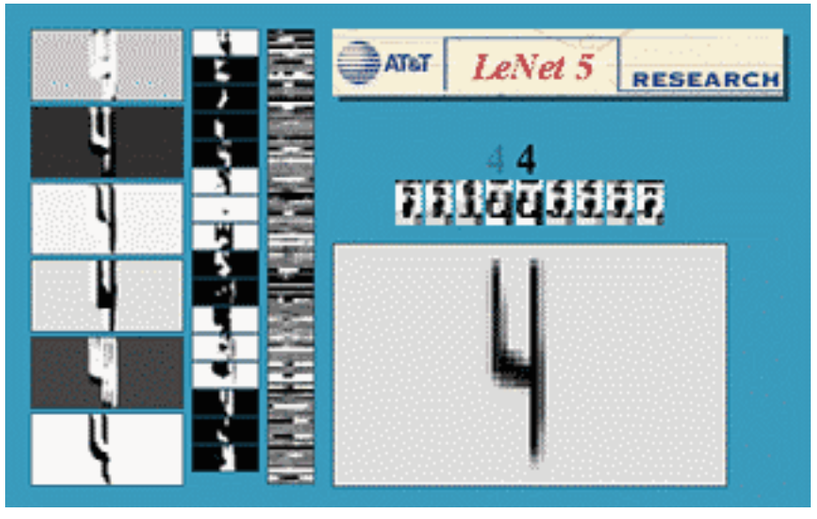
- LeNet 5의 특징

이동변환

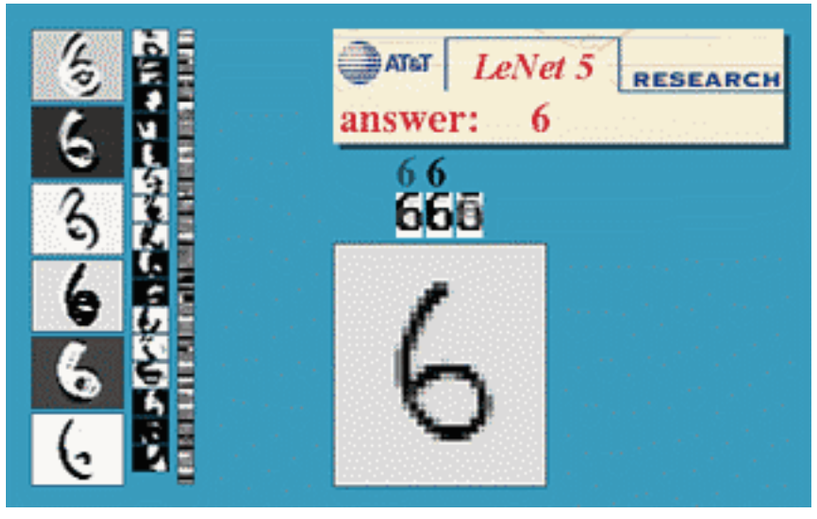


크기변환

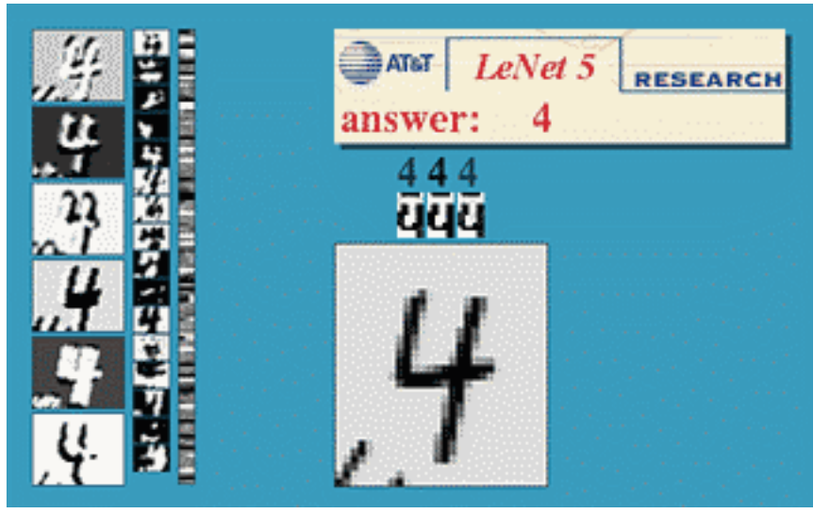
(불완전) 회전변환 $\pm 40°$



압력변환



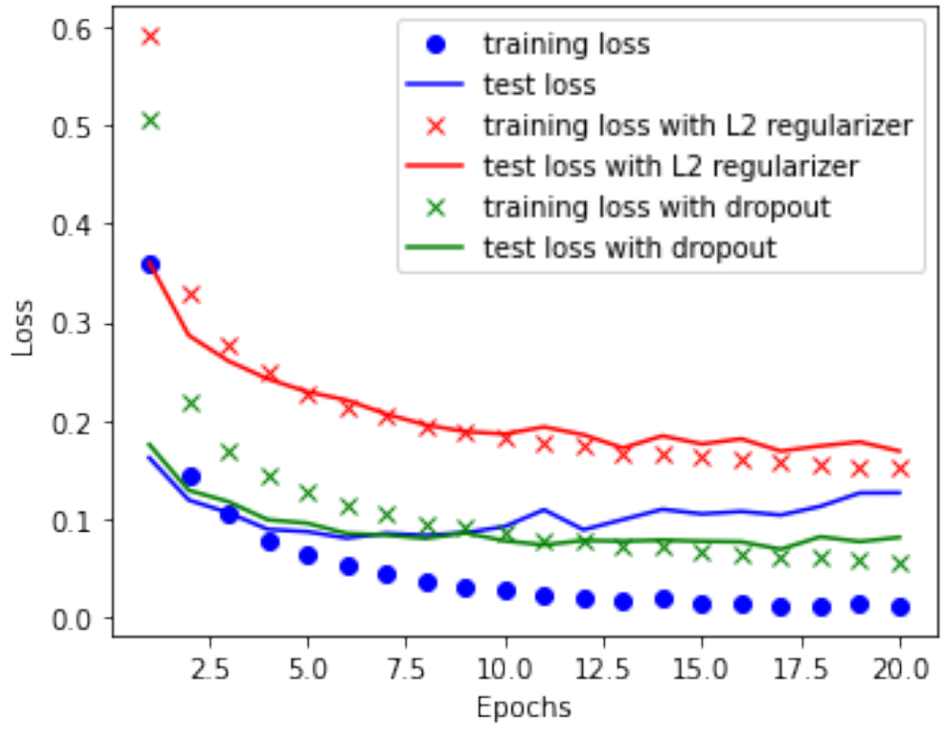
두께변환

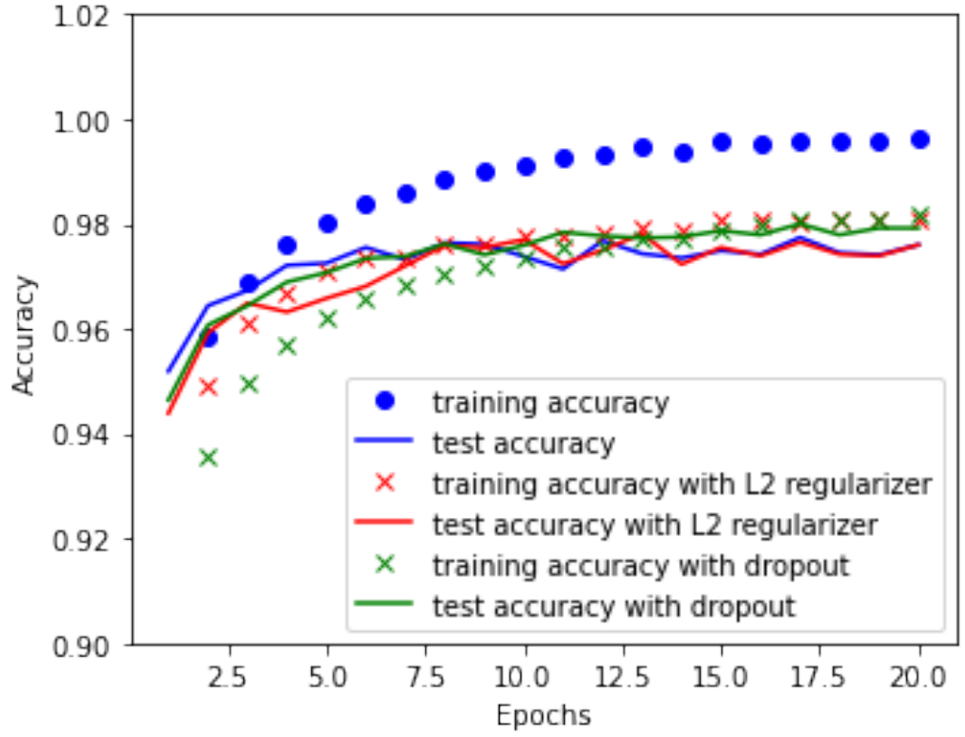


노이즈

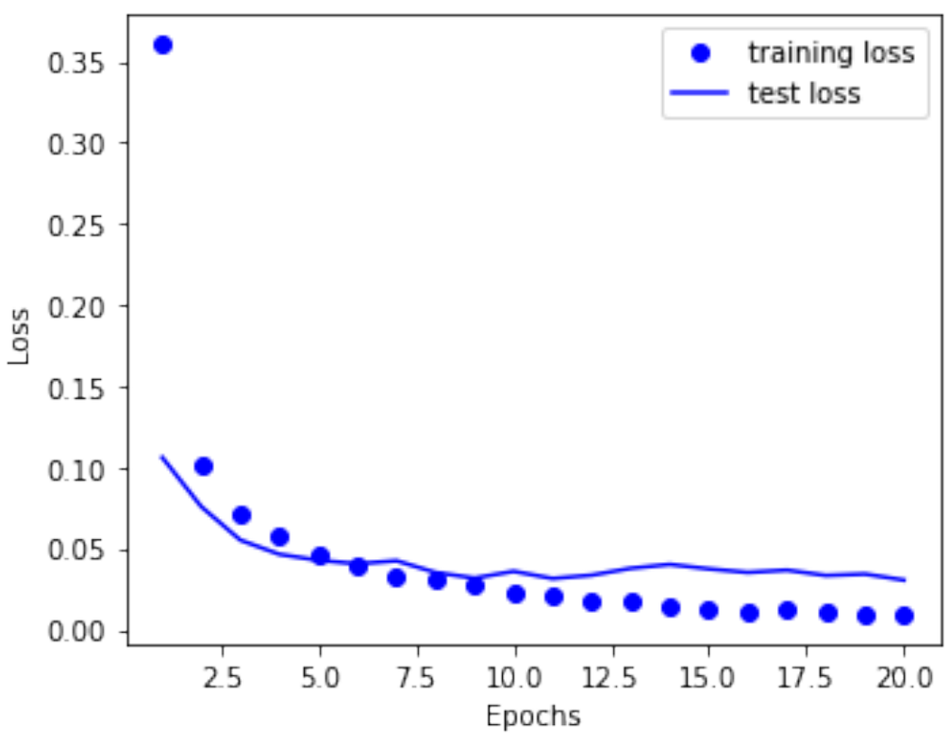
- MNIST performance 비교

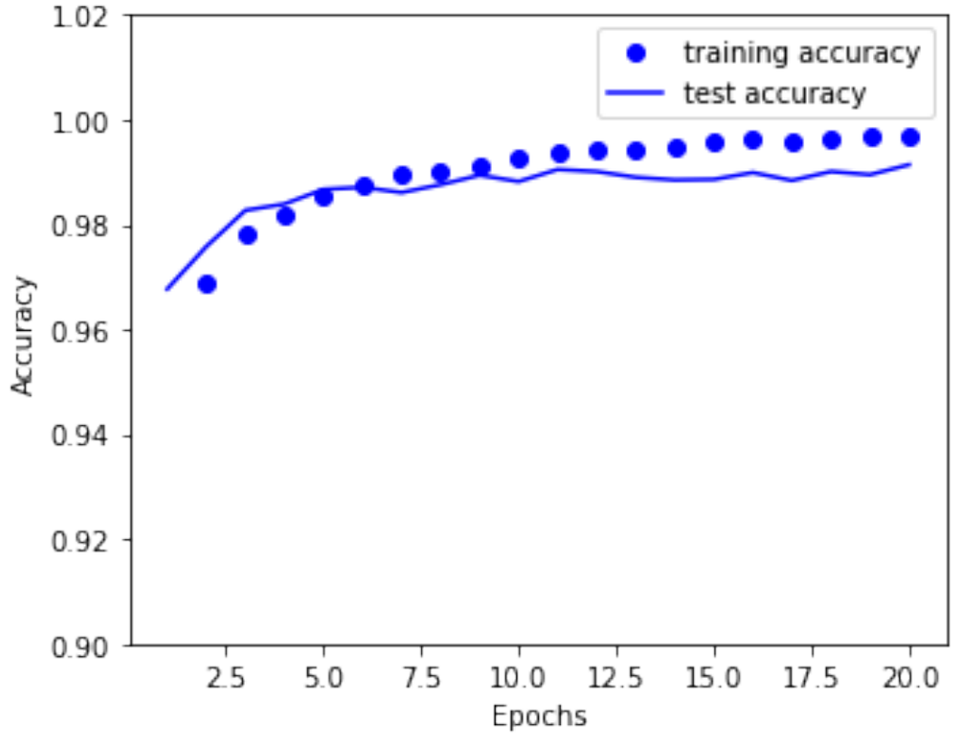
DNN





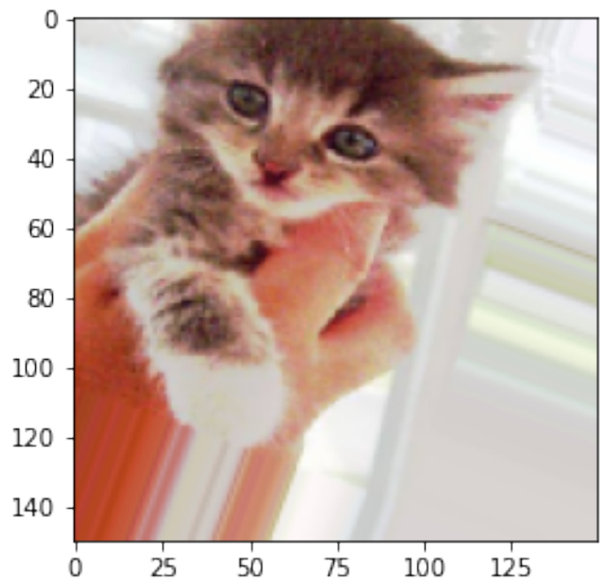
CNN

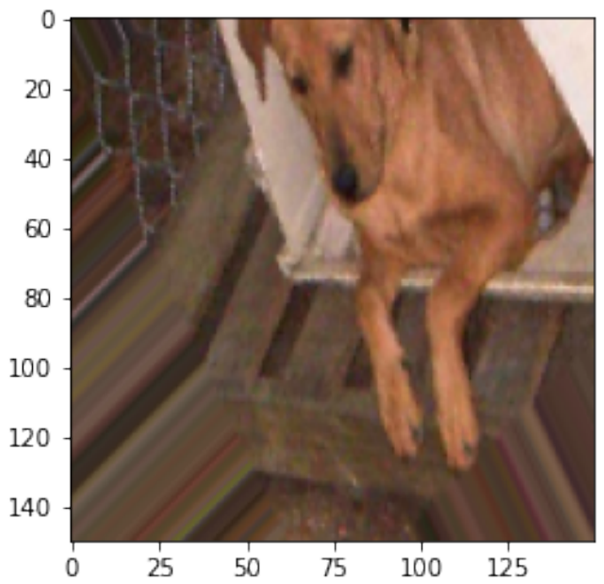




- **복잡한 데이터** 학습시키기: 고양이 vs 강아지

  - 입력 채널: RGB (Red, Green, Blue)



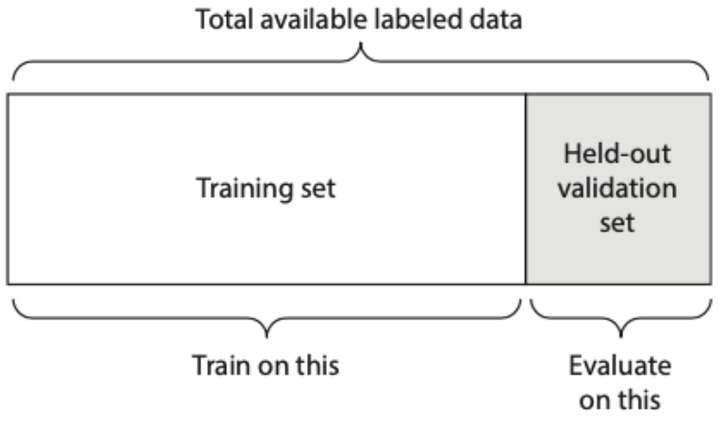


- 복잡한 데이터일수록, Deep 한 네트워크 구조가 필요
  - 여러가지 기술적인 문제를 해결해야 함.

- **Hyper-parameter tunning**

  1. 학습 세트, 테스트 세트
  2. 학습 세트, 검증 세트, 테스트 세트
     : Hyper-parameters
     (학습률, epoch 수, hidden layers, nodes, Momentum...) 변화하며 튜닝

- Simple Hold-Out Validation




```
num_validation_samples = 10000

np.random.shuffle(data)

validation_data = data[:num_validation_samples]
data = data[num_validation_samples:]

training_data = data[:]

model = get_model()
model.train(training_data)
validation_score = model.evaluate(validation_data)

# At this point you can tune your model,
# retrain it, evaluate it, tune it again...

model = get_model()
model.train(np.concatenate([training_data,
                            validation_data]))
test_score = model.evaluate(test_data)
```

Shuffling the data is usually appropriate.

Defines the validation set

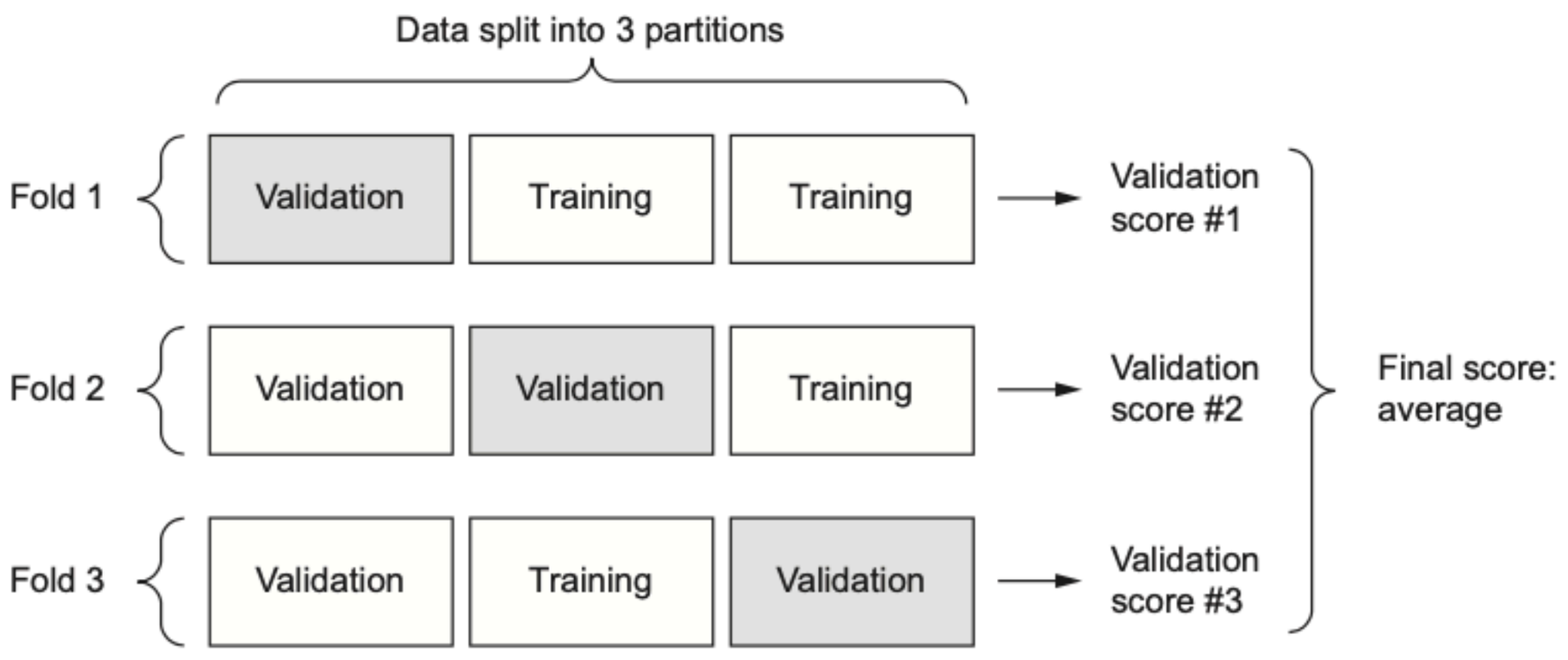
Defines the training set

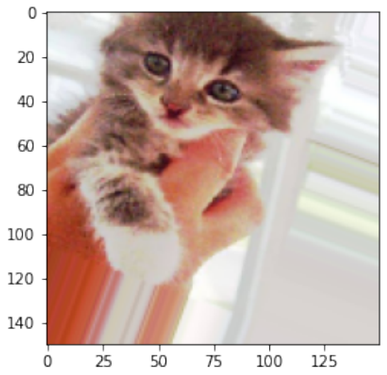
Trains a model on the training data, and evaluates it on the validation data

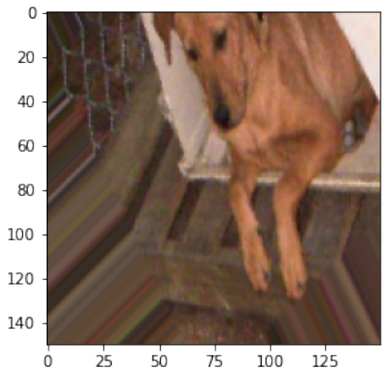
Once you've tuned your hyperparameters, it's common to train your final model from scratch on all non-test data available.

- K-fold Validation (K-겹 교차 검증)

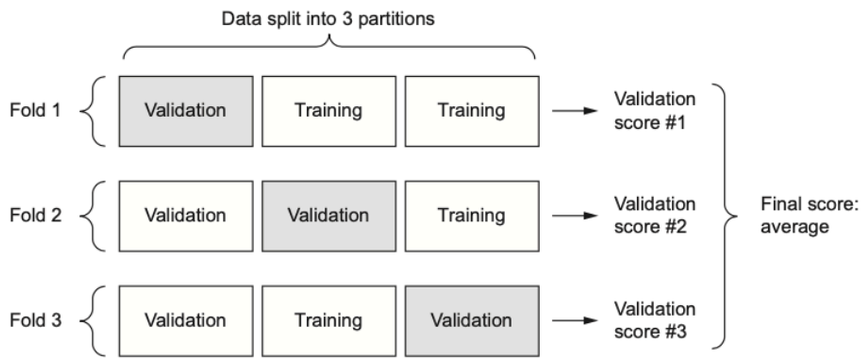
: 모델의 성능이 데이터 분할에 따라 편차가 클 수 있을 때 사용하는 방법
데이터를 K 개로 분할 후, 각 분할 i에 대해 K-1로 학습을 하고, i 로 검증







- Kaggle data set : www.kaggle.com/c/dogs-vs-cats/data
  25,000개의 고양이 사진과 25,000개의 강아지 사진



- 적은 데이터 (각 클래스마다 2,000개로 데이터를 추출)

```
print('훈련용 고양이 이미지 전체 개수:', len(os.listdir(train_cats_dir)))
print('훈련용 강아지 이미지 전체 개수:', len(os.listdir(train_dogs_dir)))

print('검증용 고양이 이미지 전체 개수:', len(os.listdir(validation_cats_dir)))
print('검증용 강아지 이미지 전체 개수:', len(os.listdir(validation_dogs_dir)))

print('테스트용 고양이 이미지 전체 개수:', len(os.listdir(test_cats_dir)))
print('테스트용 강아지 이미지 전체 개수:', len(os.listdir(test_dogs_dir)))
```

```
훈련용 고양이 이미지 전체 개수: 1000
훈련용 강아지 이미지 전체 개수: 1000
검증용 고양이 이미지 전체 개수: 500
검증용 강아지 이미지 전체 개수: 500
테스트용 고양이 이미지 전체 개수: 500
테스트용 강아지 이미지 전체 개수: 500
```

```python
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.summary()
```

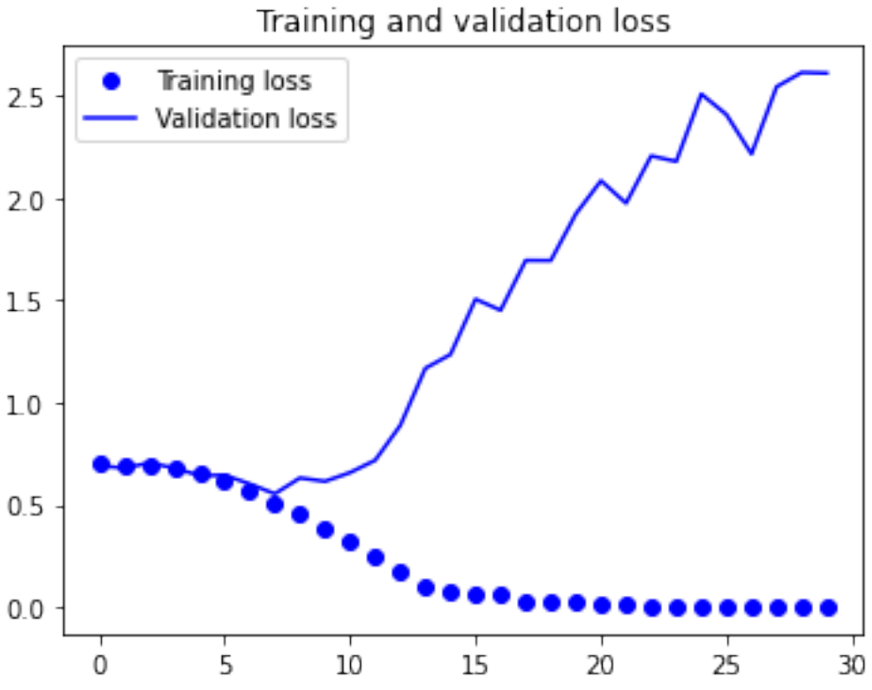
: 총 4번의 Convolution과 Pooling (Max), Fully-connected NL

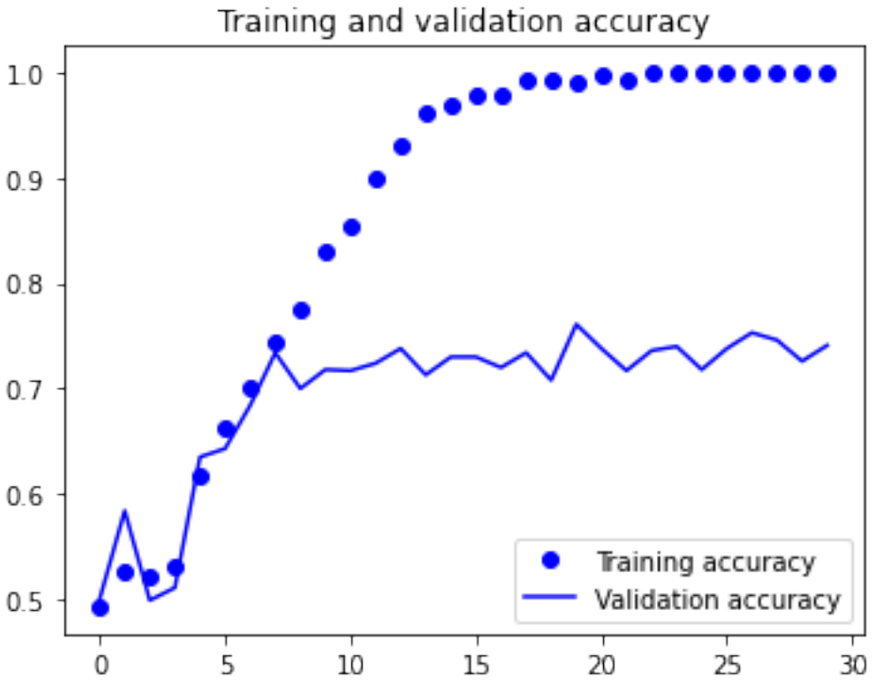
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3211776
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
```

$3 \rightarrow 32$

$32 \rightarrow 64$

$64 \rightarrow 128$





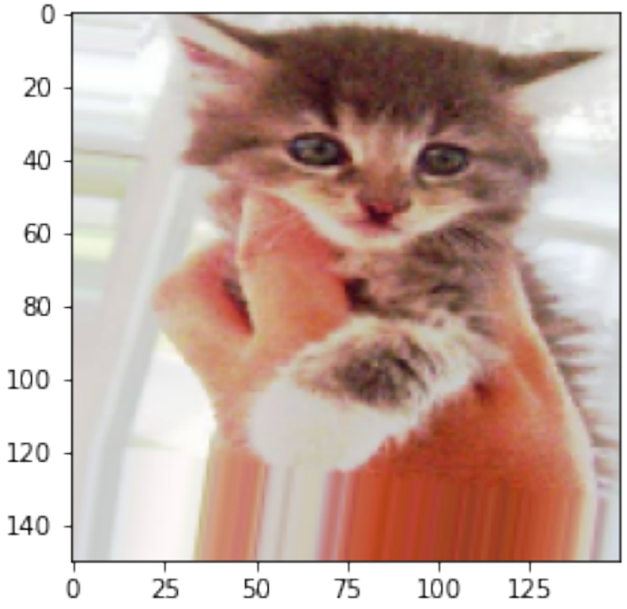
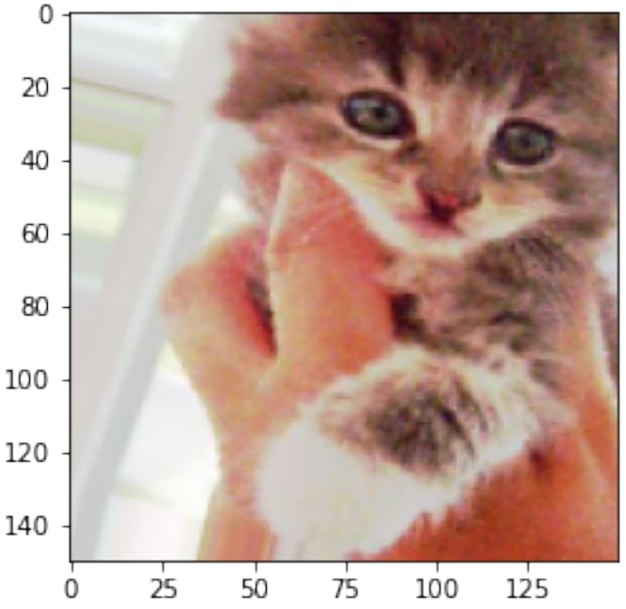
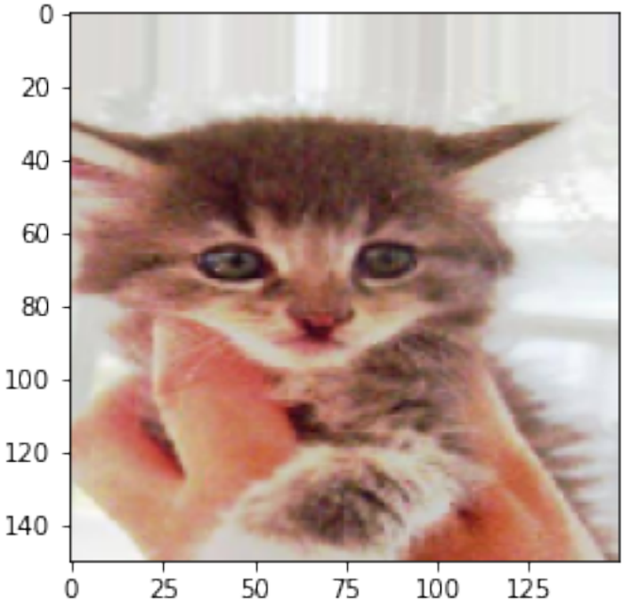
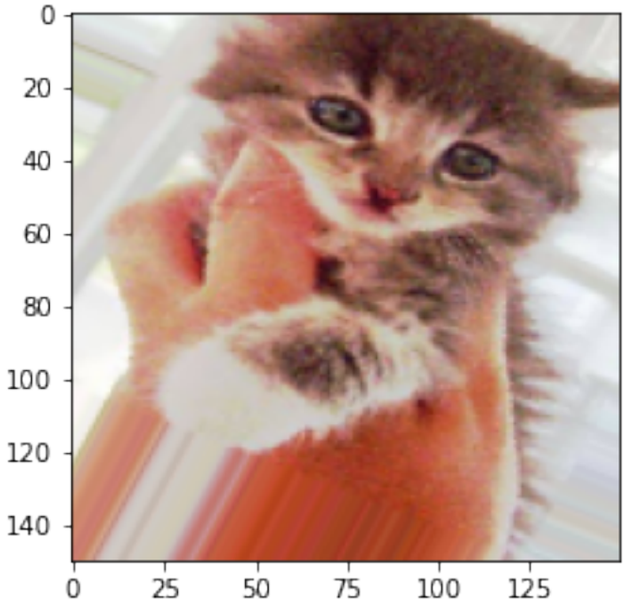
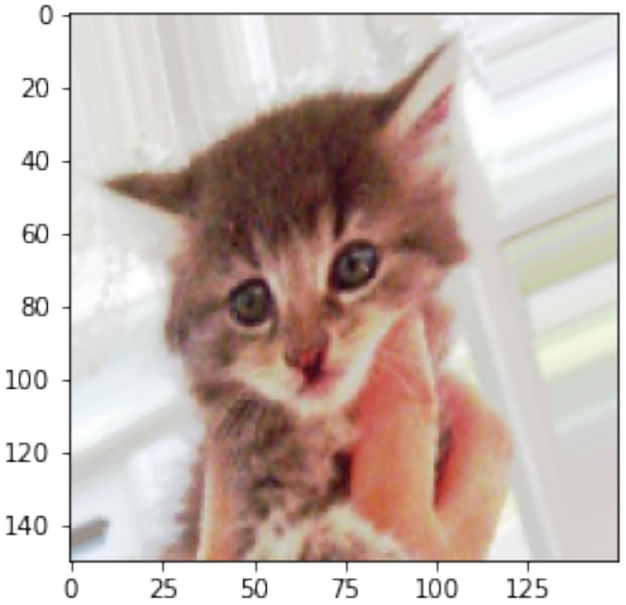
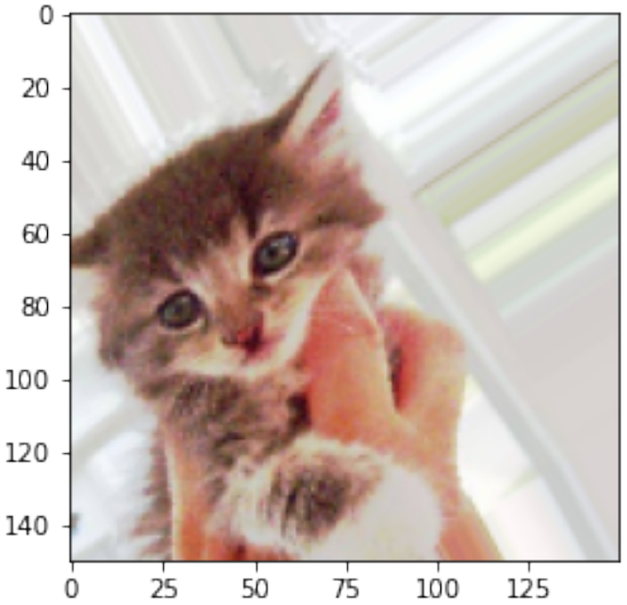
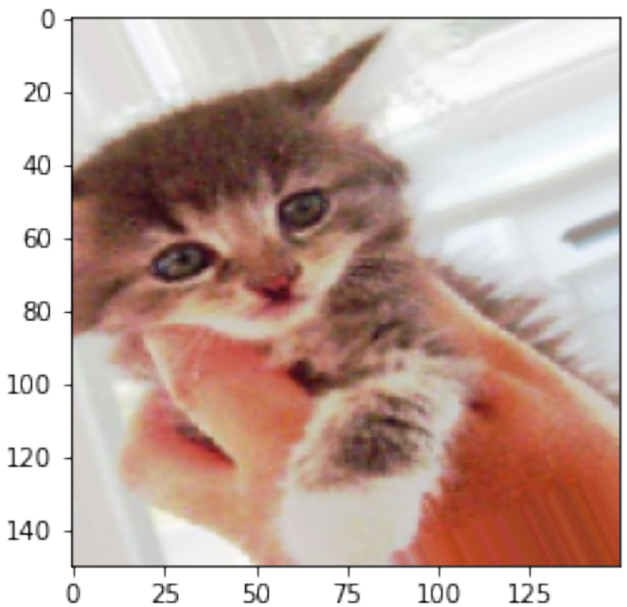
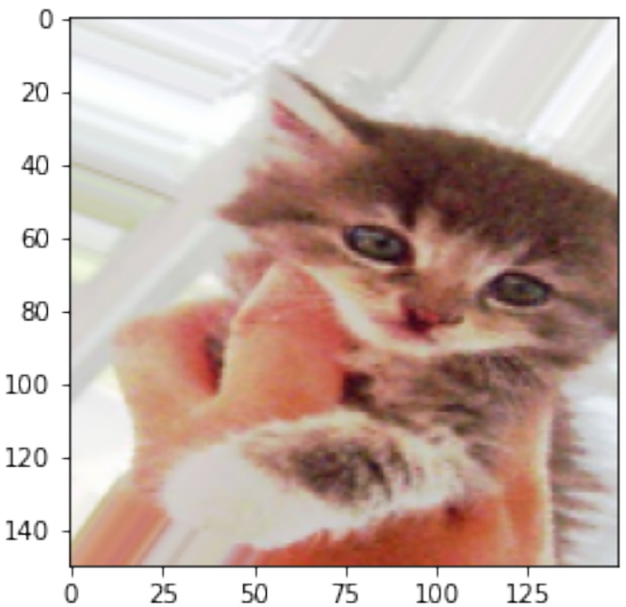
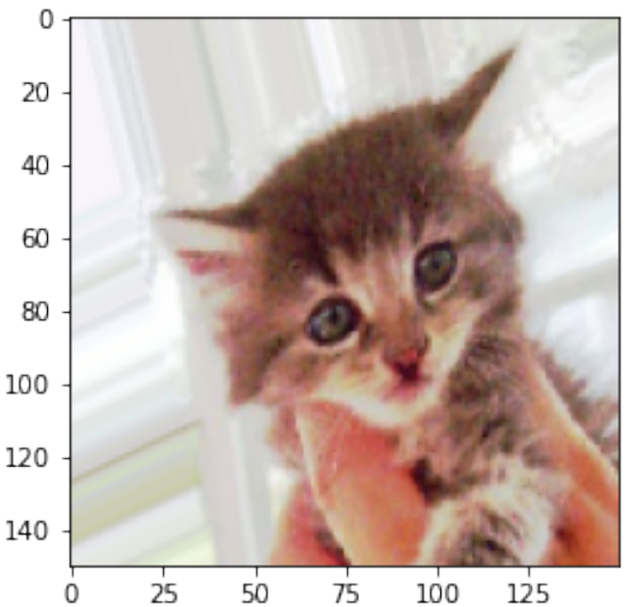
- 복잡한 데이터 구조의 경우, 많은 양의 데이터가 필요함.
  현재 학습 데이터가 1000개밖에 되지 않아 **(적은 데이터), Overfitting 이 발생**.

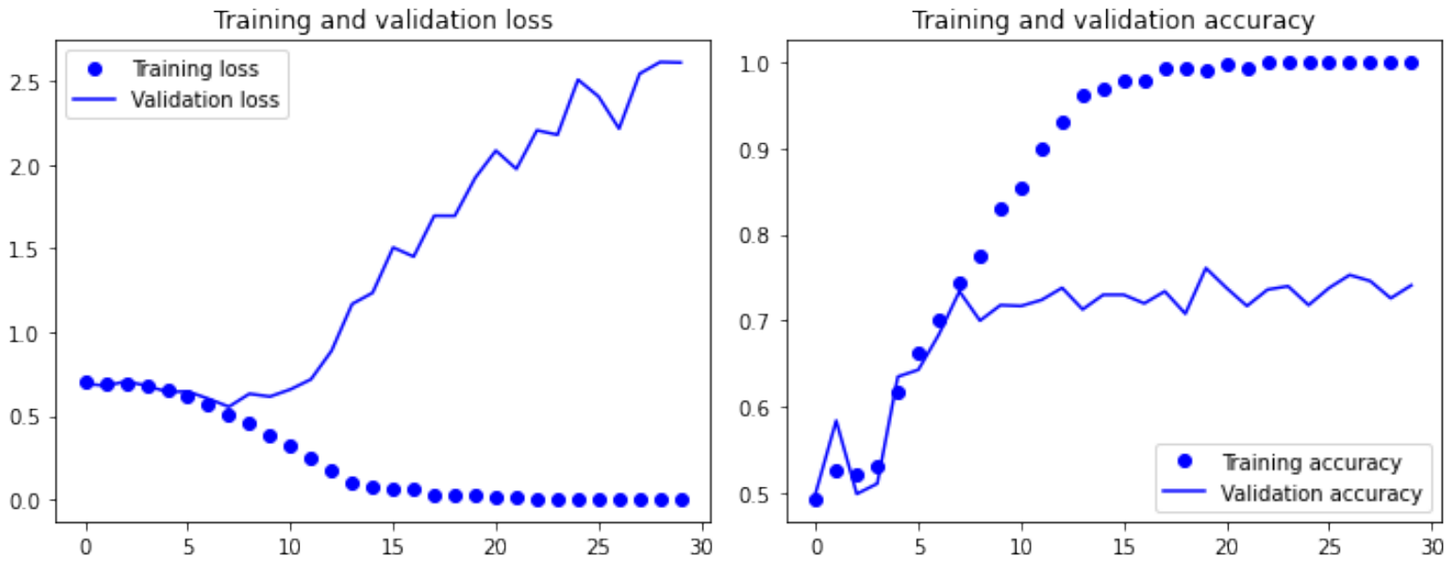
- **CNN: 데이터의 회전, 반전, 이동에 대해 충분히 강하지 못함**
  - Data augmentation

- Data augmentation

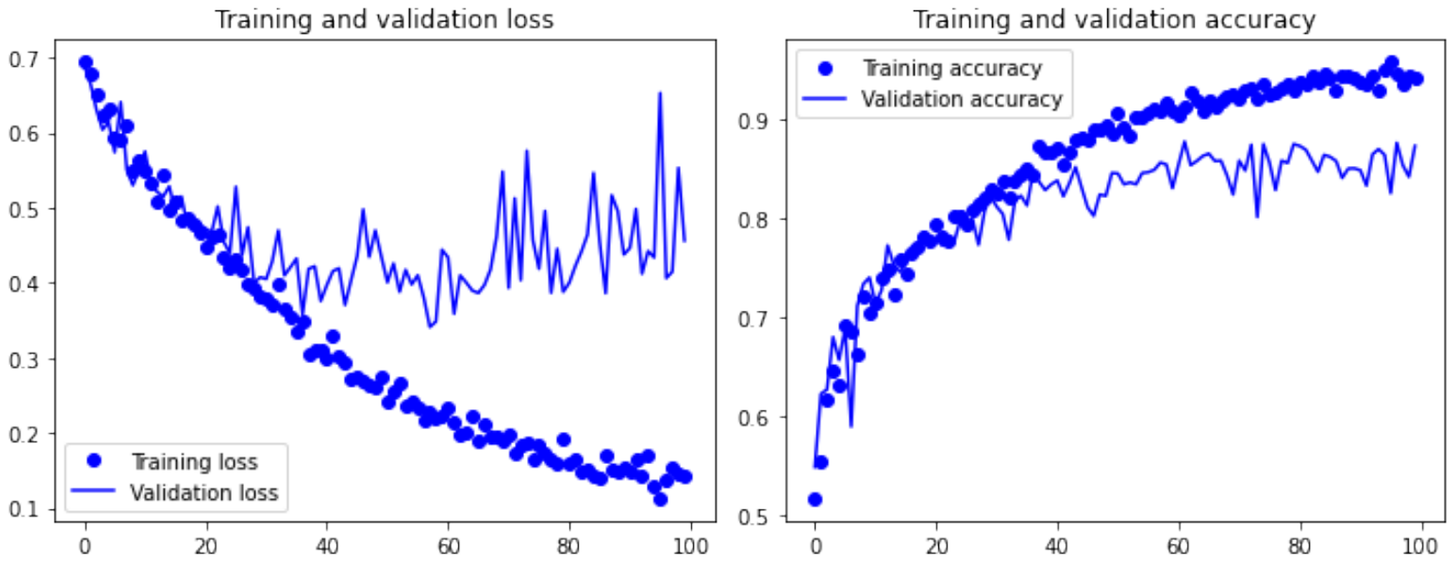
```
datagen = ImageDataGenerator(
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
```

- rotation_range: 랜덤하게 사진을 **회전**시킬 각도 범위입니다(0-180 사이).
- width_shift_range, height_shift_range: 사진을 수평과 수직으로 랜덤하게 **평행 이동**시킬 범위 (전체 넓이와 높이에 대한 비율).
- shear_range: 랜덤하게 **전단 변환(층밀림)**을 적용할 각도 범위.
- zoom_range: 랜덤하게 사진을 **확대**할 범위.
- horizontal_flip: 랜덤하게 이미지를 **수평으로 뒤집**음.
- fill_mode: 회전이나 가로/세로 이동으로 인해 새롭게 생성해야 할 픽셀을 채울 전략입니다.

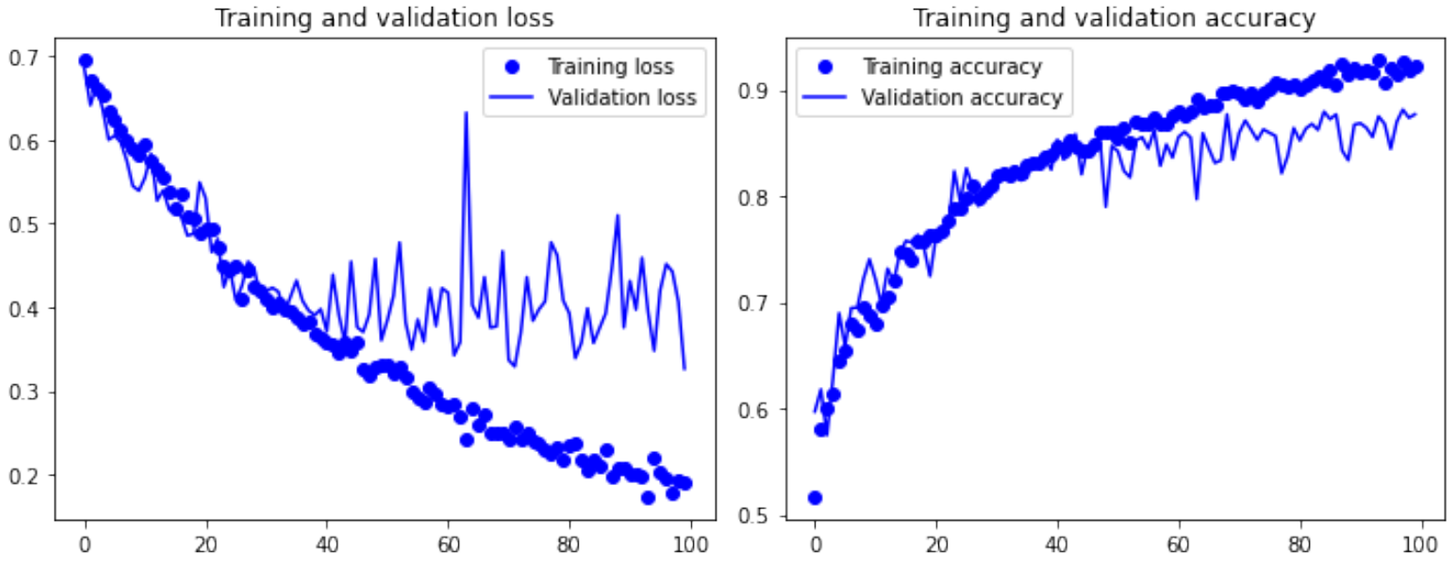












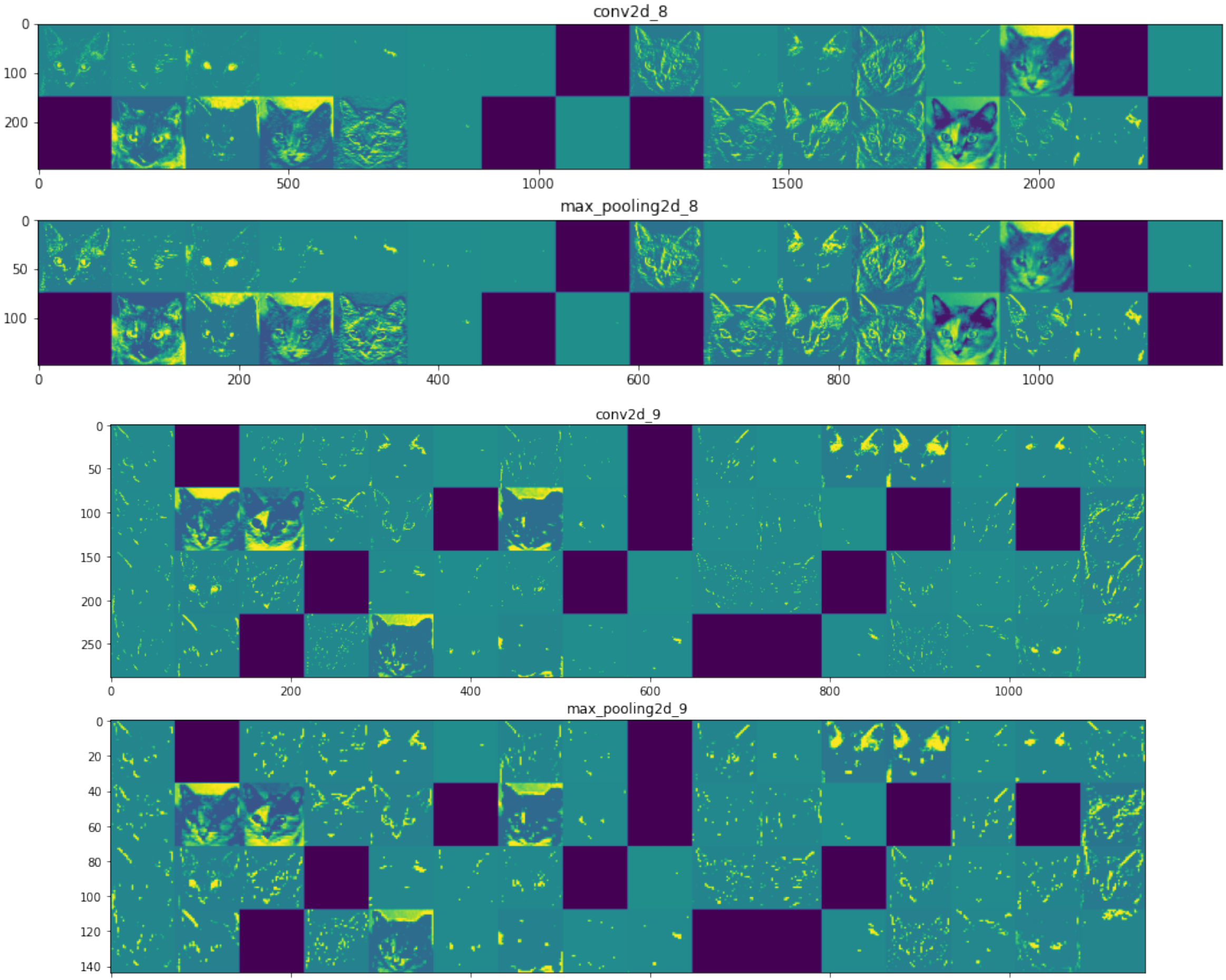
- 2000개의 훈련데이터 (각 클래스별 1000개) 배치크기: 20 사용



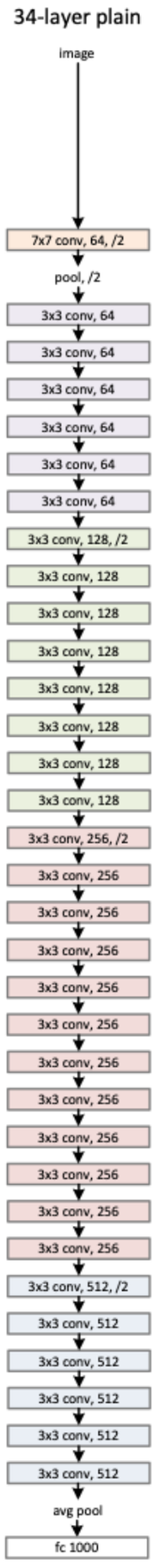
- Generator 사용.
  배치 크기: 32 사용

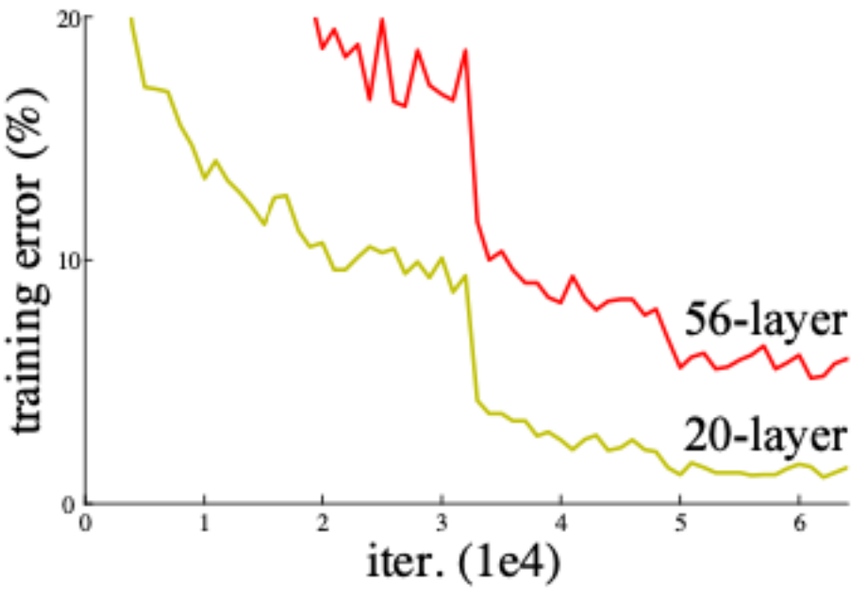


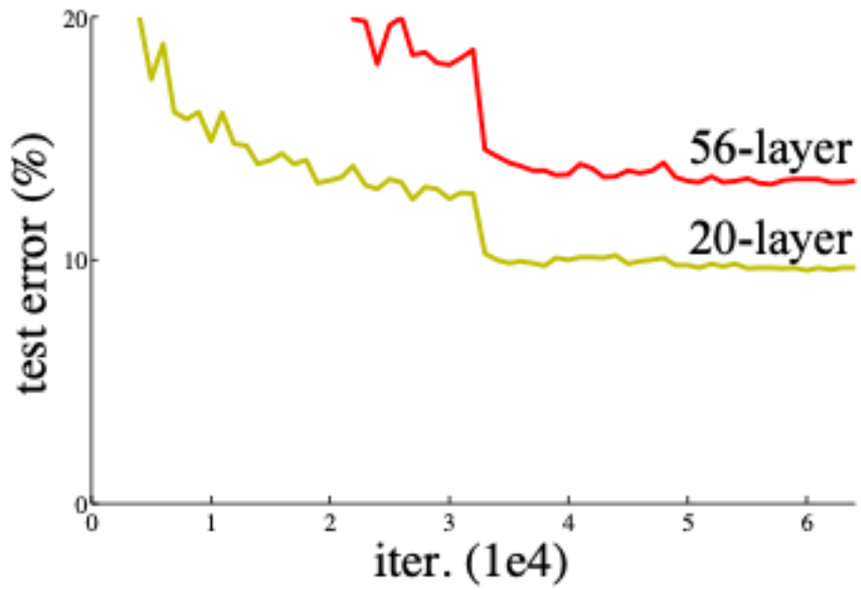
- Generator 사용.
  1. 배치 크기: 32 사용
  2. 완전연결 직전,
     Drop-out(50%) 사용


conv2d_8
max_pooling2d_8
conv2d_9
max_pooling2d_9

- Deeper and Deeper : Overcome Vanishing Gradient Problem



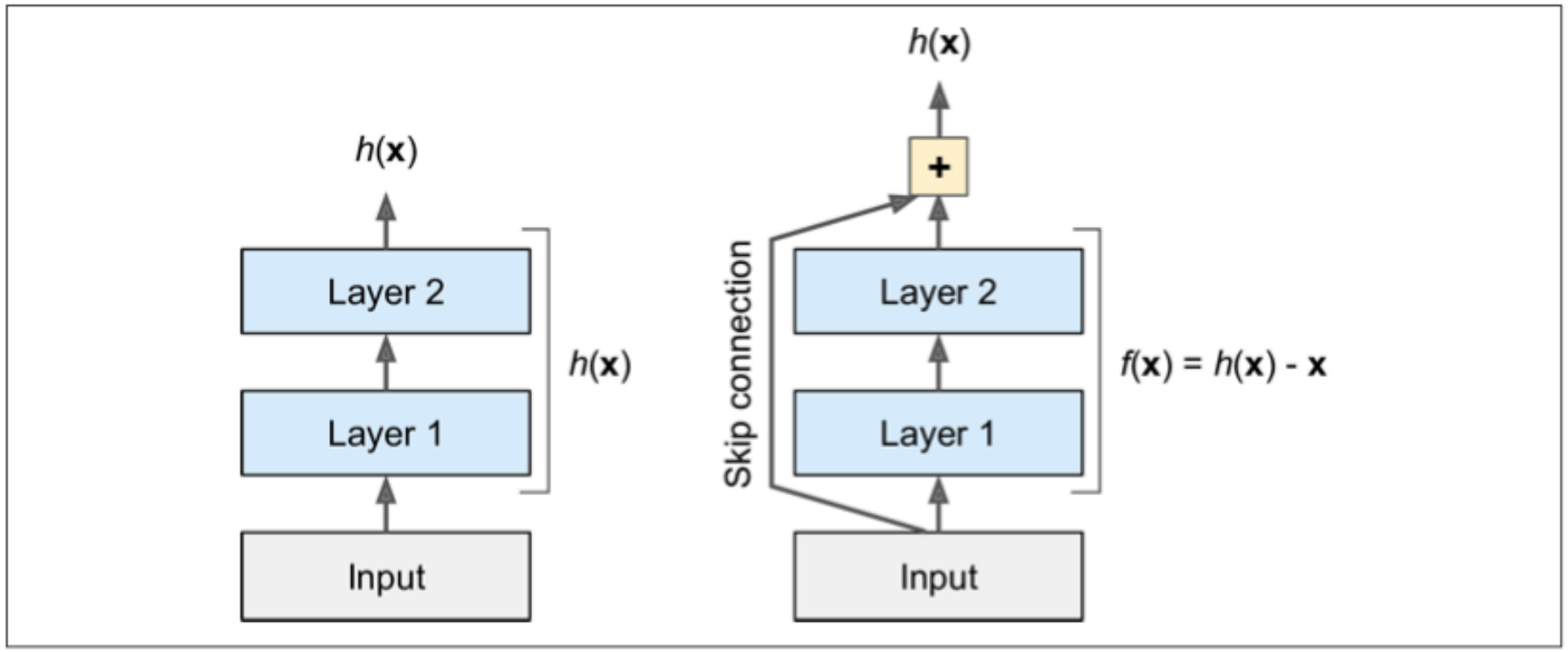




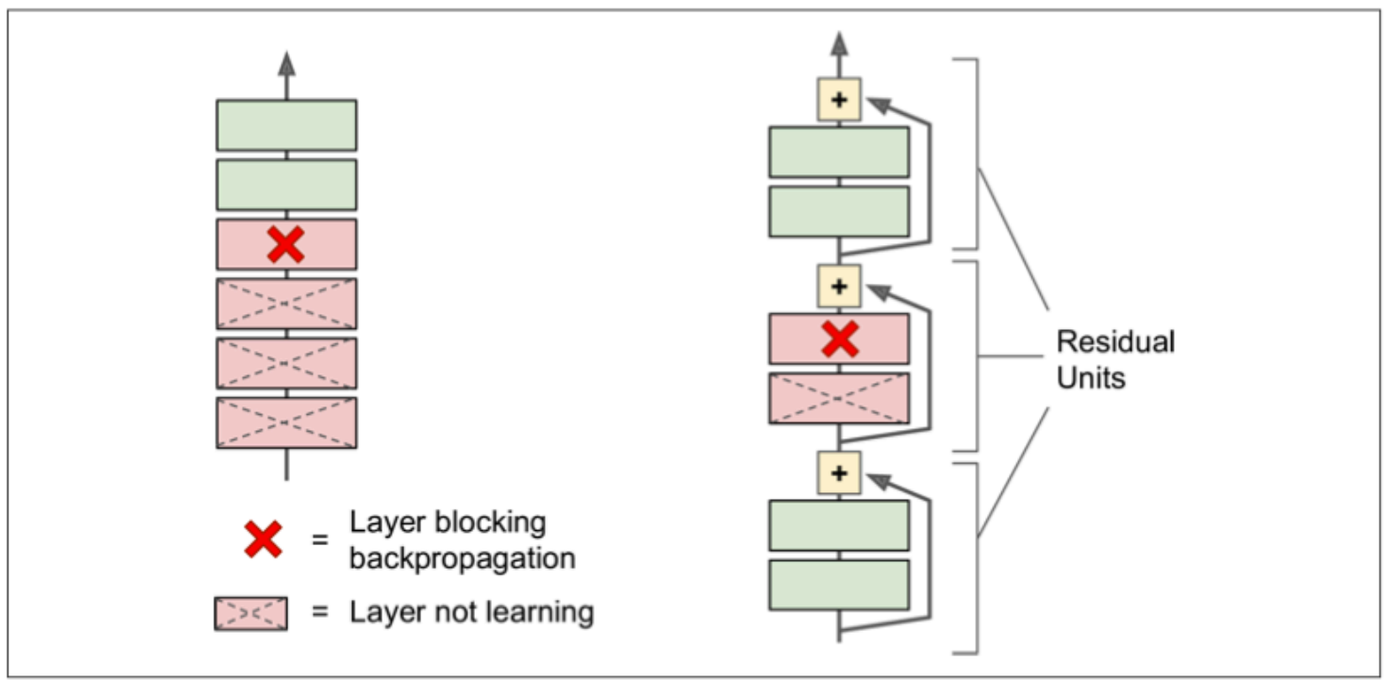
: 복잡한 데이터(사진)을 이해하기 위해 Hidden Layer를 늘릴 경우
- 학습률이 떨어지게 됨 (Vanishing Gradient problem 발생)

"Deep Residual Learning form Image Recognition"
(CVPR 2016, arXiv:1512.03385 (cs)) 에서 문제 해결

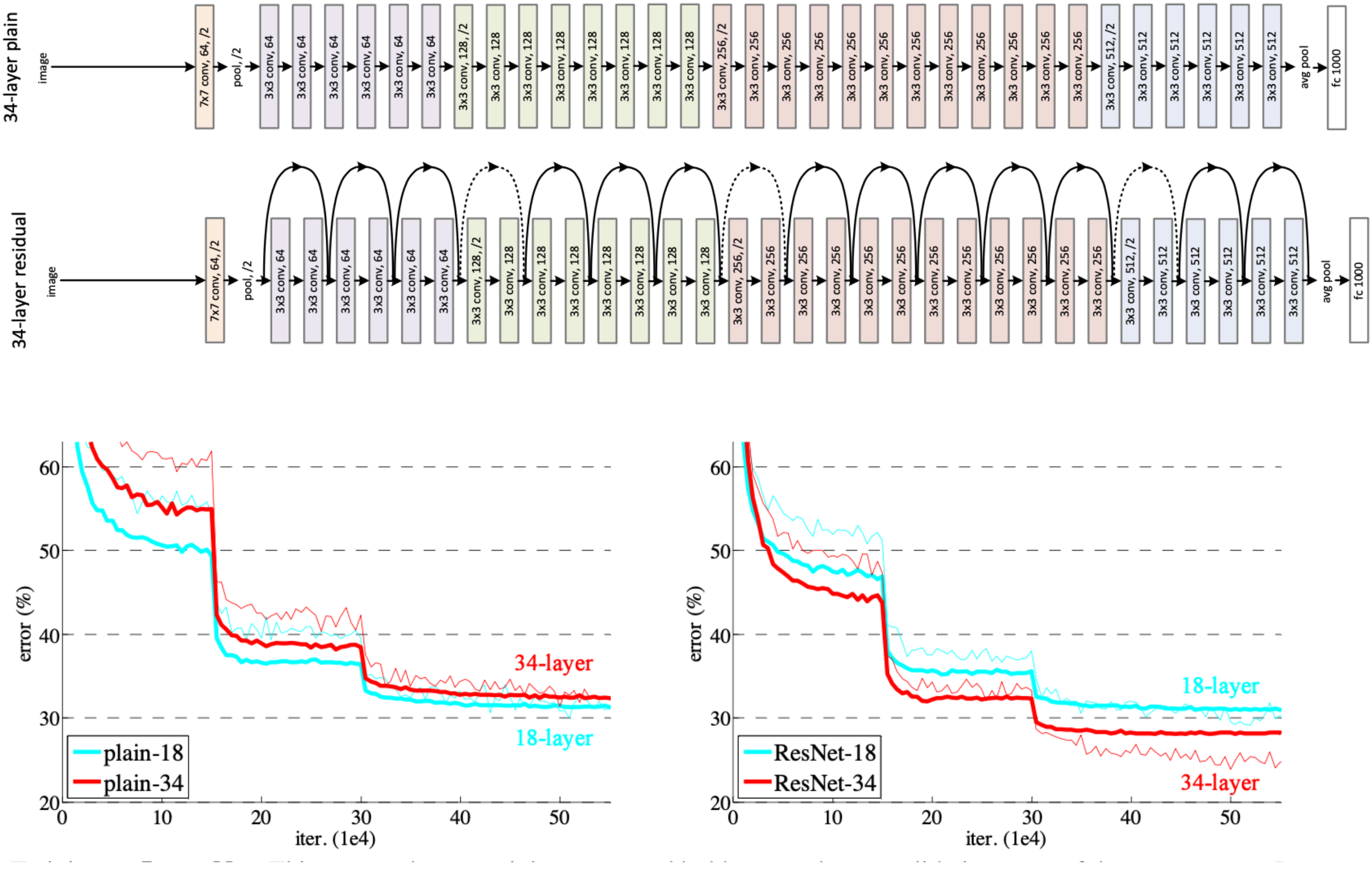
- ResNet (Residual Neural Network)



: Skip connection을 더함으로써, Identity 함수를 제공



- 학습 능률이 떨어지는 경우에도, 다음 블록은 Skip connection을 통해서 학습을 진행할 수 있음.


34-layer plain
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000
34-layer residual
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000
error (%)
iter. (1e4)
plain-18
plain-34
34-layer
18-layer
ResNet-18
ResNet-34
34-layer
18-layer

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				

$$50 = 1 + 1 + 3 \times 3 + 3 \times 4 + 3 \times 6 + 3 \times 3$$

To be continued...

- RNN (Recurrent Neural Network)

- Unsupervised Learning
  - Auto encoder

- Generative Model