2022 WINTER ESC Week4

Convolutional Neural Networks

김민수, 윤이경

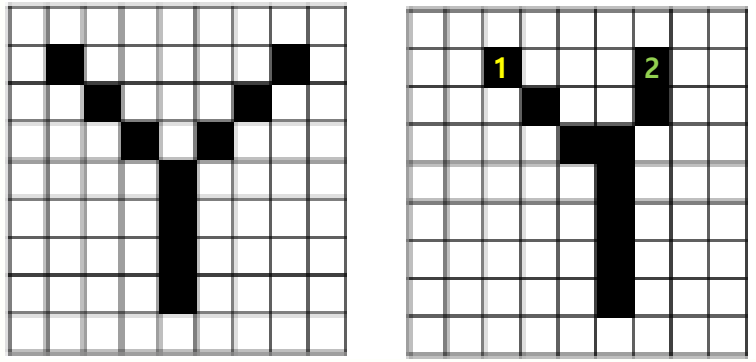
C O N T E N T S

01. Introduction to CNN

01 Introduction to CNN

When we use CNN?

grid-like topology : time-series data, image/video data



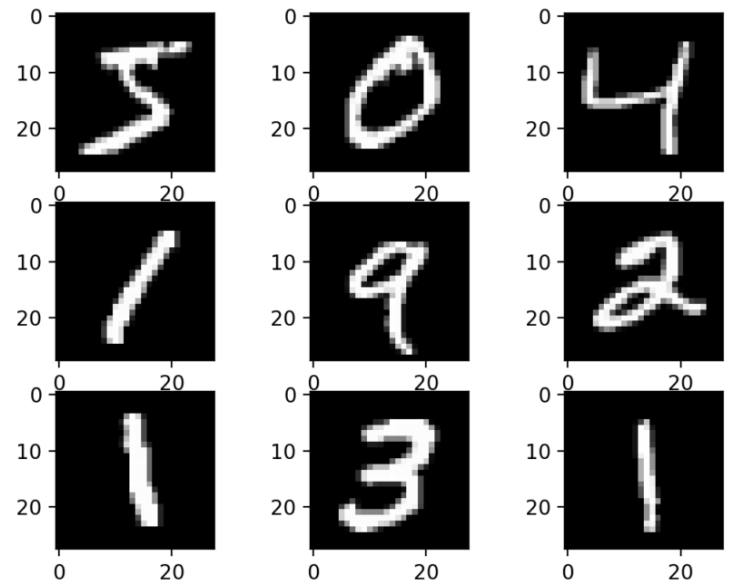
신경망에 입력하기 위해 1차원 배열로 변환하는 과정에서 공간 정보 손실

이미지의 공간적인 구조 정보를 보존하면서 학습할 수 있는 방법이 필요 → 합성곱 신경망

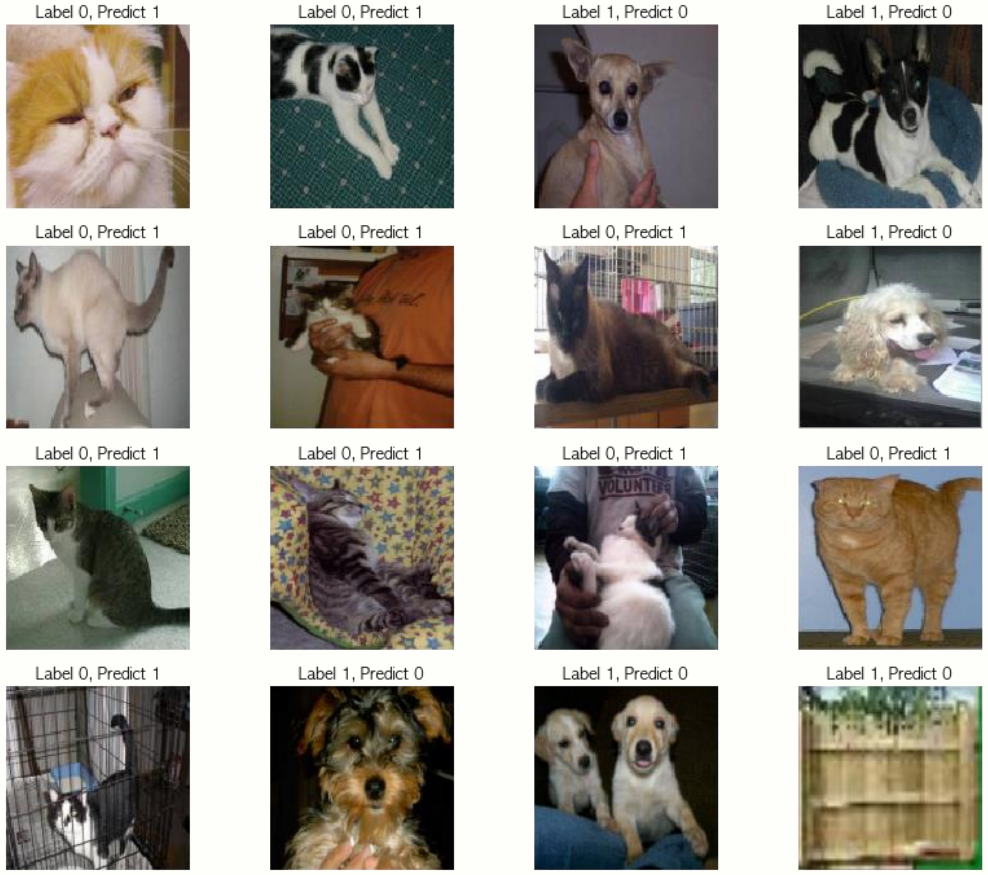
01

When we use CNN?

Introduction to CNN



MNIST Handwritten Digit Classification



Classify Images of Dogs and Cats

02. Basic Principles of CNN

Channel

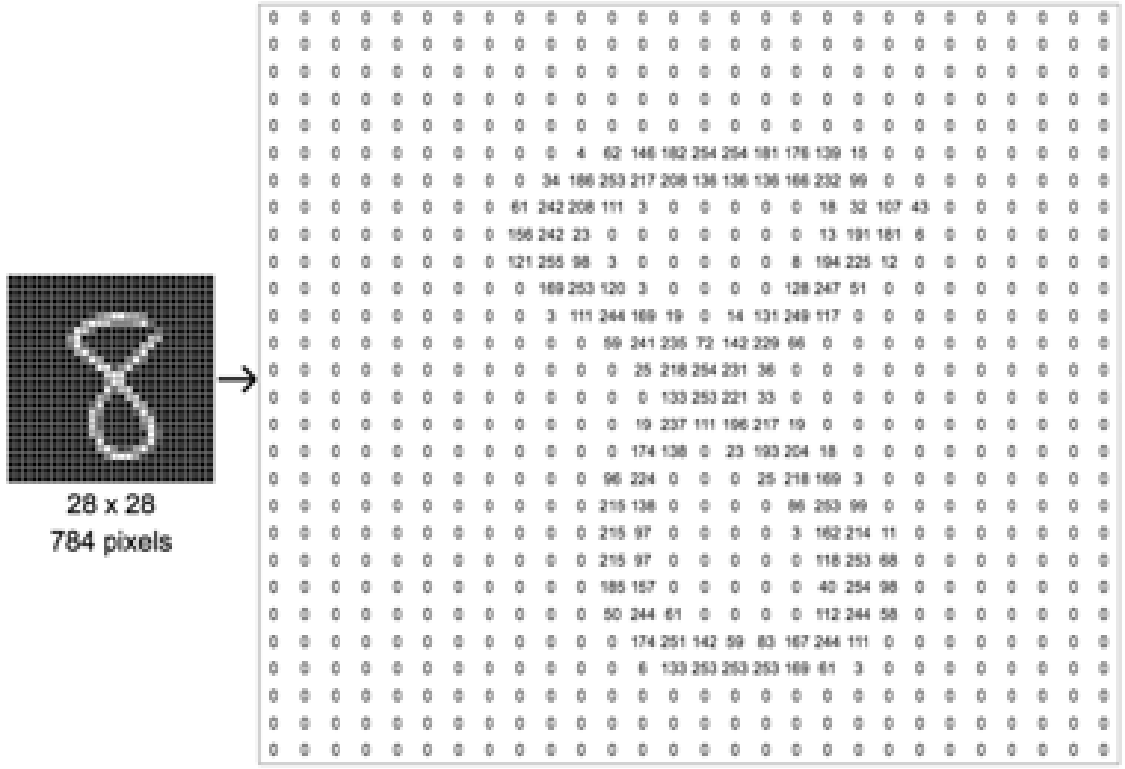
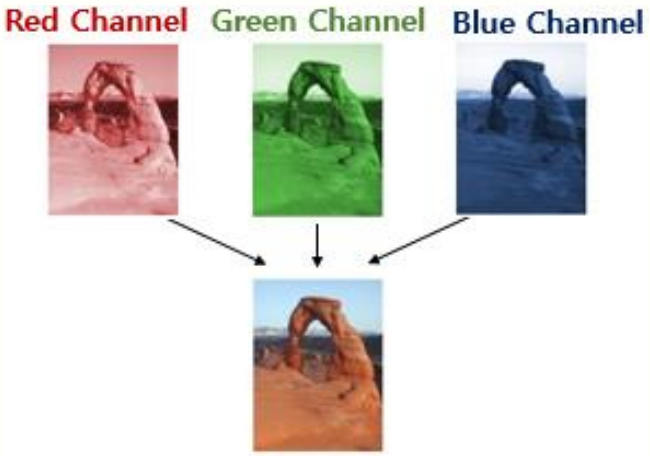


Image = **(Channel, Height, Width)**

Height = 이미지의 세로 방향 픽셀 수
Width = 이미지의 가로 방향 픽셀 수
Channel = 색 성분



1-D Convolution operation

motivate the definition convolution (1차원)

Q : to find the location of spaceship with a laser sensor

$$s(t) = \int x(a)w(t-a)da$$

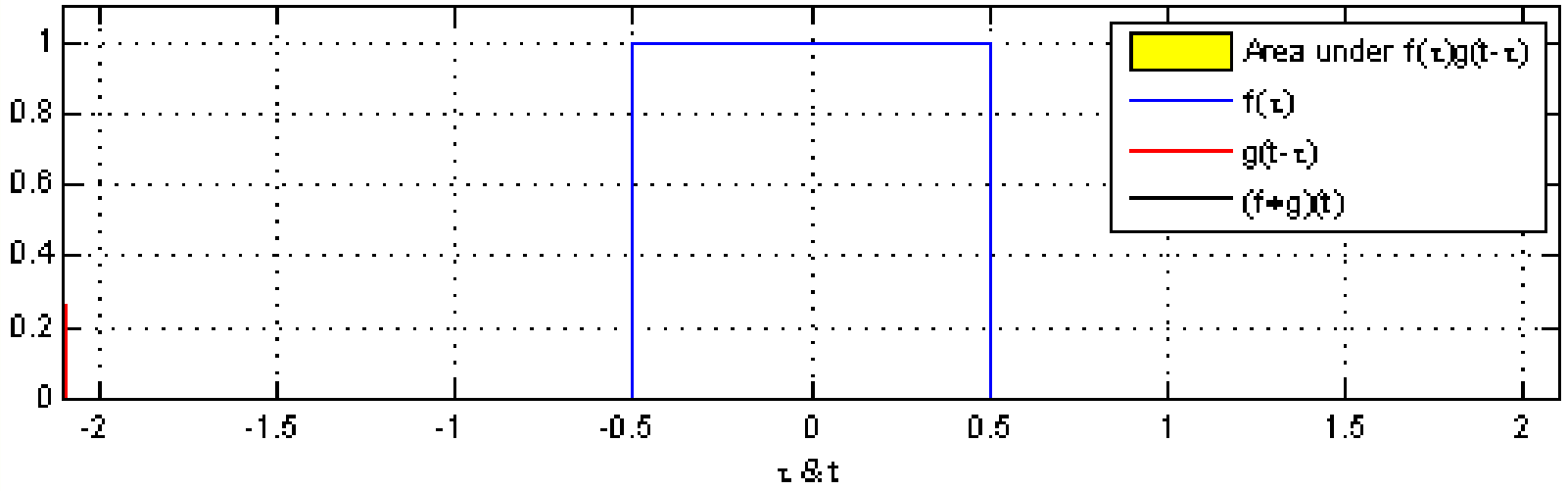
$$= (x * w)(t)$$

$$= \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

Input **x(t)** : the position of the spaceship at time t
Kernel **w(a)** : weighting function, more weight to recent measurements **= filter**
Output **s(t)** : location of spaceship **= feature map**

1-D Convolution operation

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau)\, d\tau$$

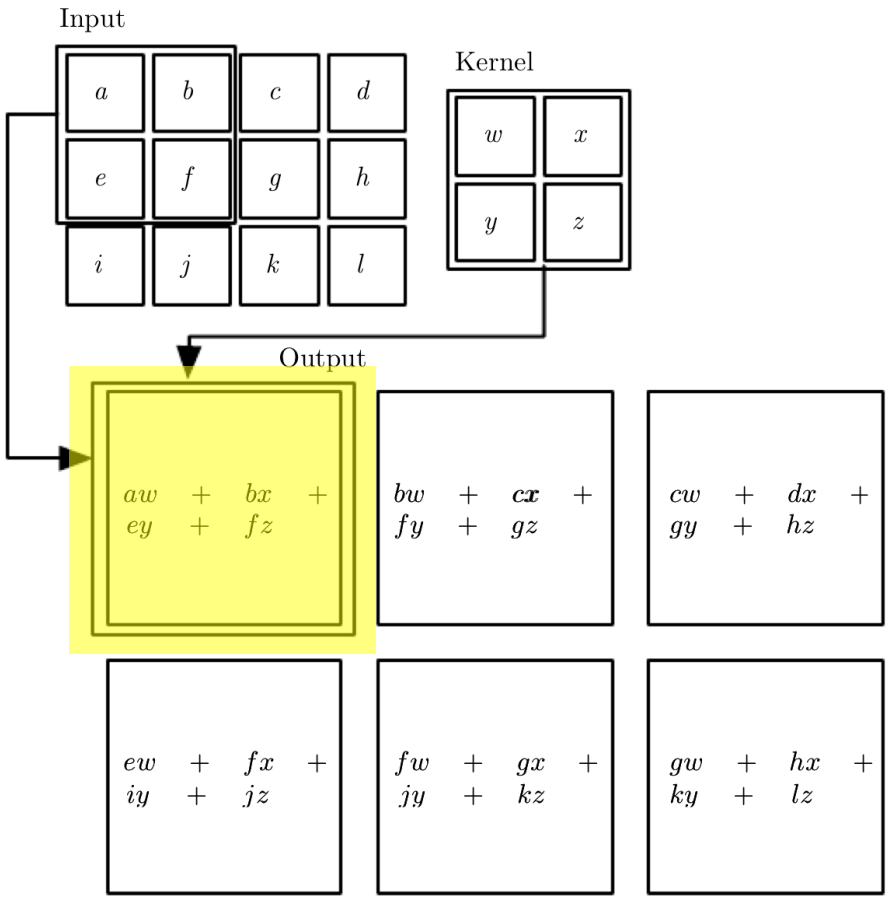


2-D Convolution operation

Input & kernel in CNN is usually multi-dimentional array

$$S(i,j) = (I * K)(i,j)$$

$$= \sum_m \sum_n I(m,n)K(i-m, j-n)$$

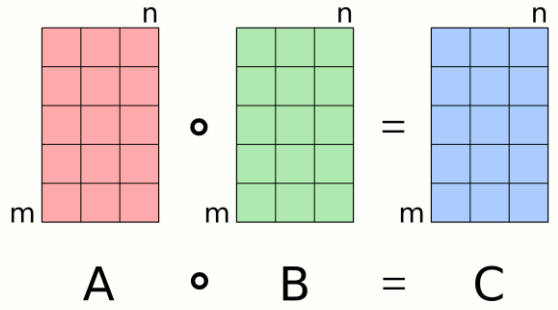


Hadamard product (아다마르 곱)

For two matrices A and B of the same dimension $m \times n$,
the Hadamard product is a matrix of the same dimension as the operands, with elements given by

[Definition]

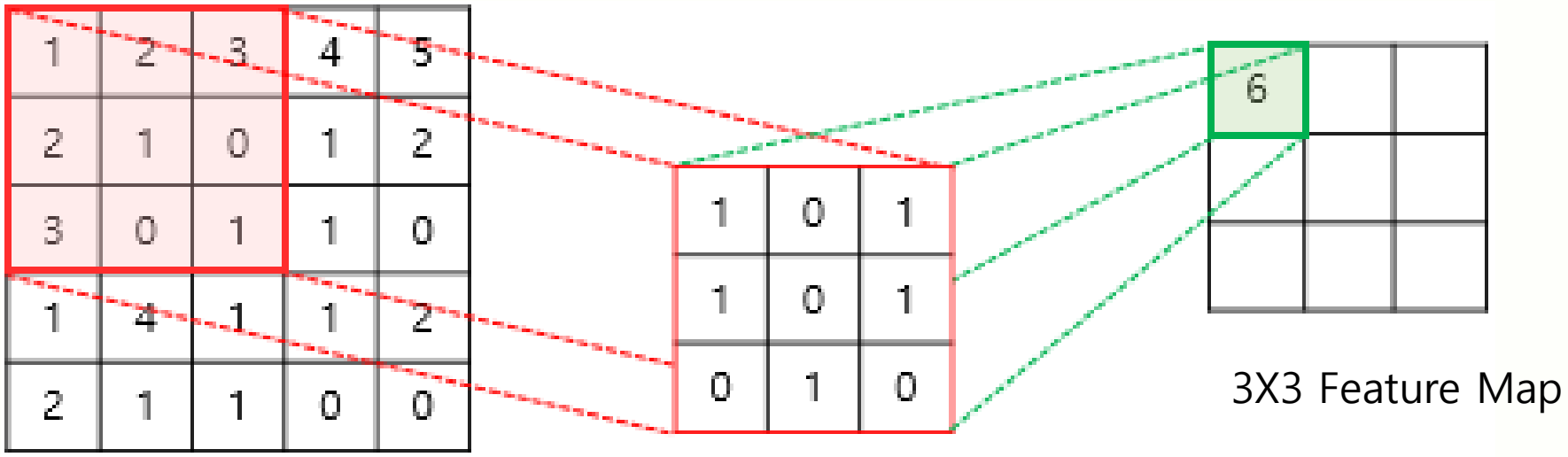
$$(A \circ B)_{ij} = (A \odot B)_{ij} = (A)_{ij}(B)_{ij}.$$



[Example]

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \circ \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}\,b_{11} & a_{12}\,b_{12} & a_{13}\,b_{13} \\ a_{21}\,b_{21} & a_{22}\,b_{22} & a_{23}\,b_{23} \\ a_{31}\,b_{31} & a_{32}\,b_{32} & a_{33}\,b_{33} \end{bmatrix}.$$

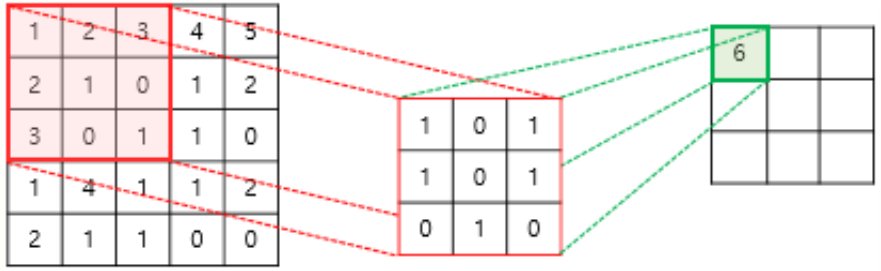
2-D Convolution operation



$$(1\times1) + (2\times0) + (3\times1) + (2\times1) + (1\times0) + (0\times1) + (3\times0) + (0\times1) + (1\times0) = 6$$

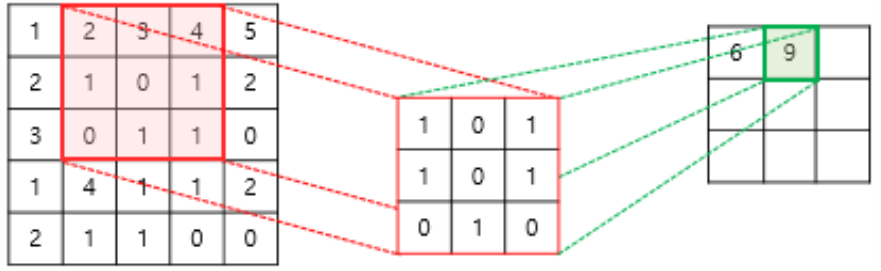
2-D Convolution operation

1 STEP



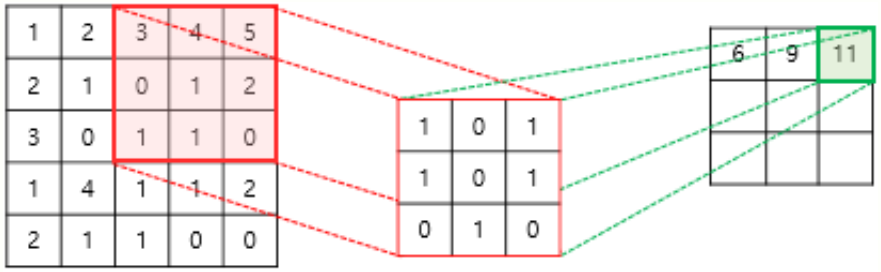
(1×1) + (2×0) + (3×1) + (2×1) + (1×0) + (0×1) + (3×0) + (0×1) + (1×0) = 6

2 STEP



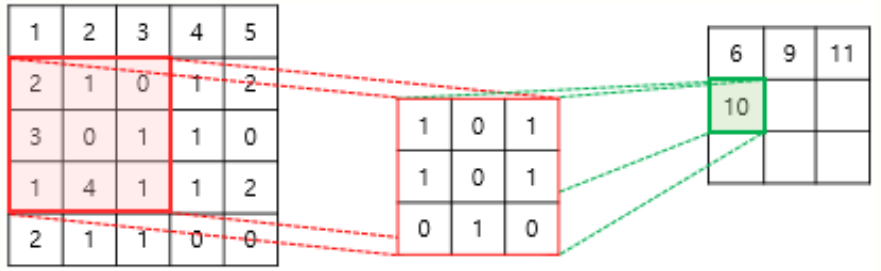
(2×1) + (3×0) + (4×1) + (1×1) + (0×0) + (1×1) + (0×0) + (1×1) + (1×0) = 9

3 STEP



(3×1) + (4×0) + (5×1) + (0×1) + (1×0) + (2×1) + (1×0) + (1×1) + (0×0) = 11

4 STEP



(2×1) + (1×0) + (0×1) + (3×1) + (0×0) + (1×1) + (1×0) + (4×1) + (1×0) = 10

6	9	11
10	4	4
7	7	4

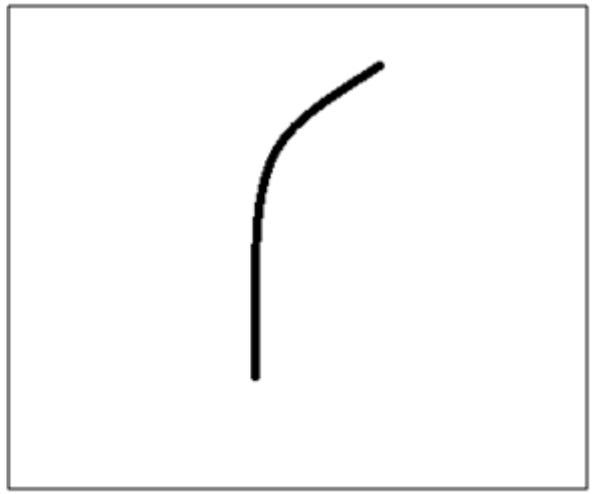
3X3 Feature Map

2-D Convolution operation / Kernel

Kernel = Filter : whether the input data has its characteristics or not.
Kernel (Filter) can be thought as **feature identifiers.**

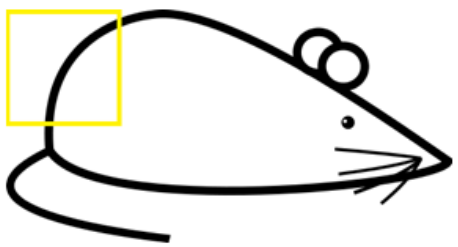
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

2-D Convolution operation / Kernel



Visualization of the filter on the image

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation : (50×30) + (50×30) + (50×30) + (20×30) + (50×30) = 6600

If the filter has a corresponding characteristic in the input data, the result value will be large.

2-D Convolution operation / Kernel

Visualization of the filter on the image

Visualization of the receptive field

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation : 0 (zero)

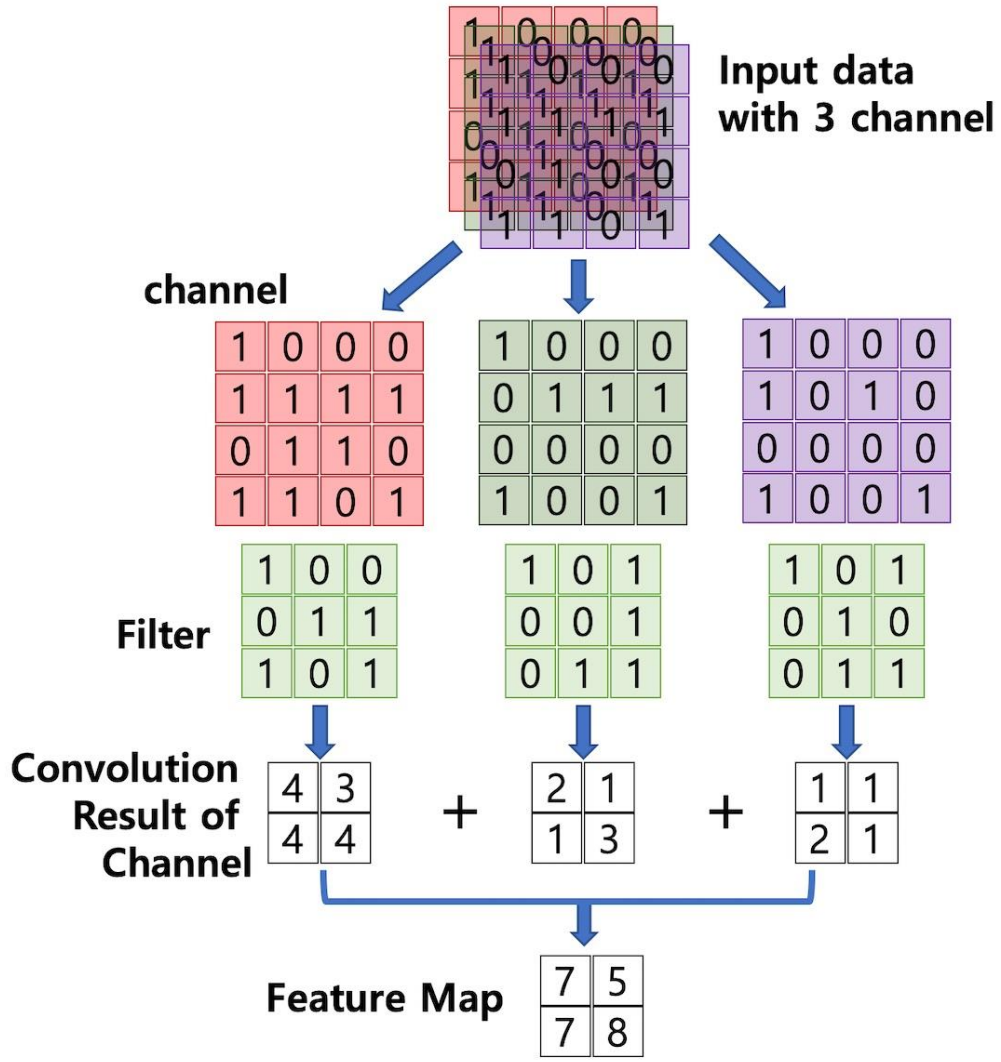
If the characteristics are not similar or have no characteristics, the result value is close to zero.

Convolutional is used to find the same feature in different places of an images.

3-D Convolution operation

If Input data = 3 channels
Kernel (Filter) also needs 3 channels

.. But feature map = 1 channel

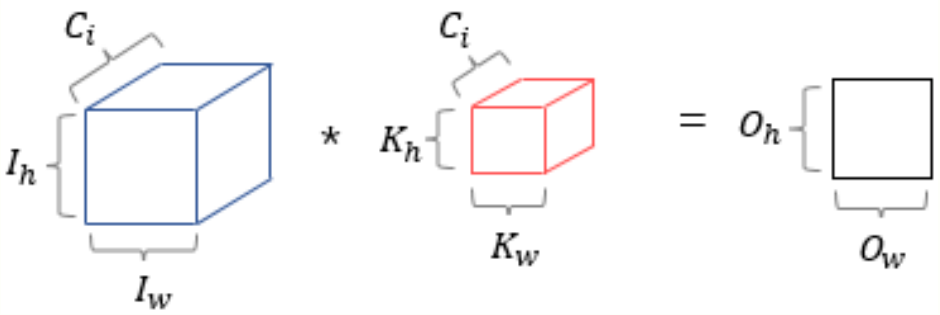


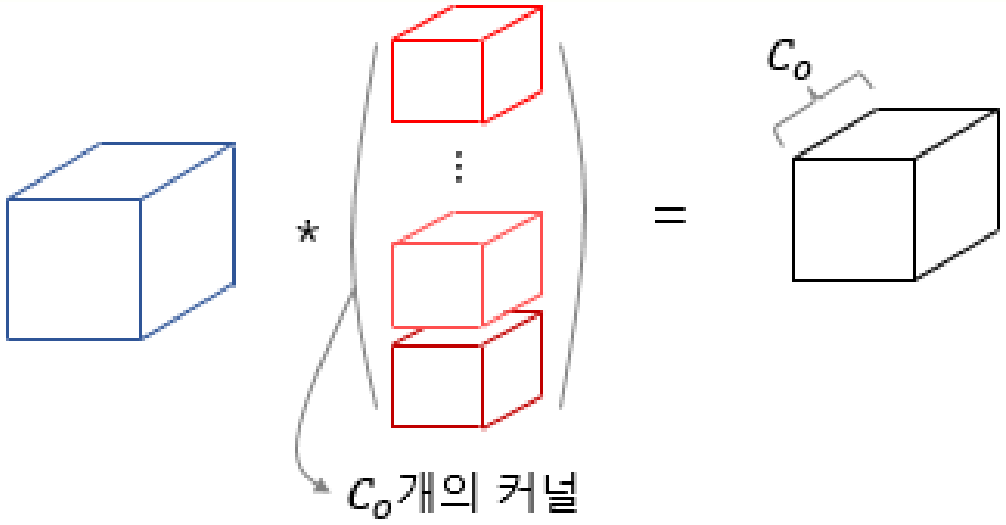
3-D Convolution operation

Ih : input height
Iw : input weight

Kh : kernel height
Kw : kernel weight

Oh : feature map height
Ow : feature map weight
Ci : input data channel





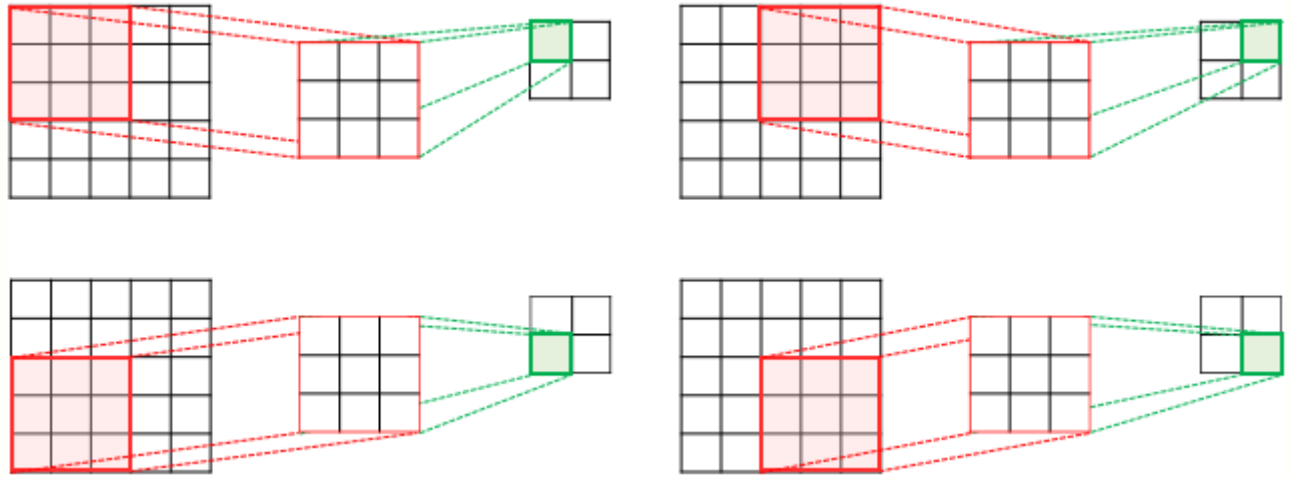
If you want to increase the number of output channels, you can use multiple filters.

Stride

Stride controls how the filter convolves around the input : **Filter를 적용하는 간격**

Increase stride if we want smaller dimensions

Stride 2



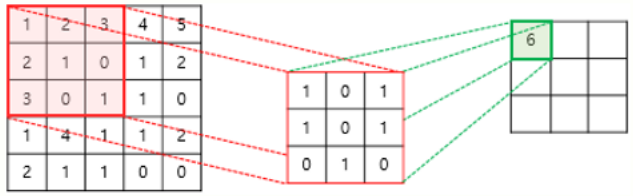
Stride

Stride 1

02 Basic Principles of CNN

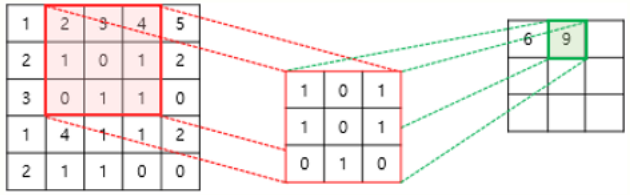
2-D Convolution operation

1 STEP



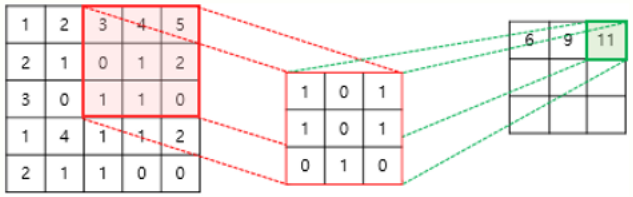
(1×1) + (2×0) + (3×1) + (2×1) + (1×0) + (0×1) + (3×0) + (0×1) + (1×0) = 6

2 STEP



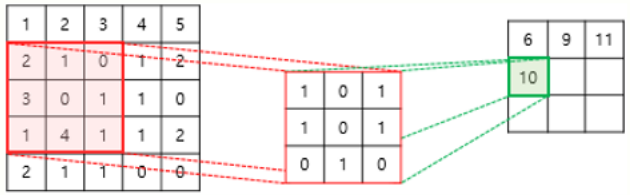
(2×1) + (3×0) + (4×1) + (1×1) + (0×0) + (1×1) + (0×0) + (1×1) + (1×0) = 9

3 STEP



(3×1) + (4×0) + (5×1) + (0×1) + (1×0) + (2×1) + (1×0) + (1×1) + (0×0) = 11

4 STEP



(2×1) + (1×0) + (0×1) + (3×1) + (0×0) + (1×1) + (1×0) + (4×1) + (1×0) = 10

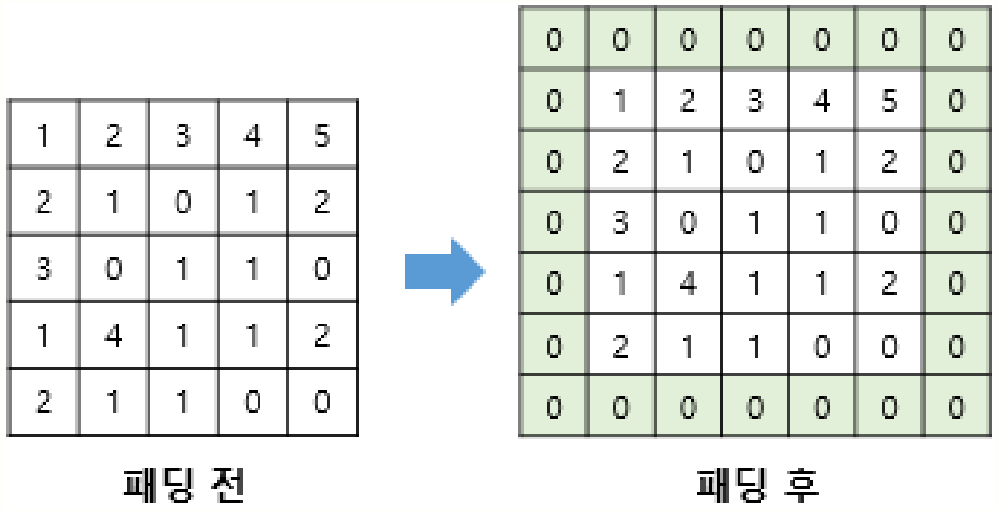
6	9	11
10	4	4
7	7	4

3X3 Feature Map

Padding (패딩) : Zero - Padding

Result of convolution, the dimension of the volume decreased.

If you want to keep the size of the Feature map the same as the size of the input even after the convolution operation, use padding.

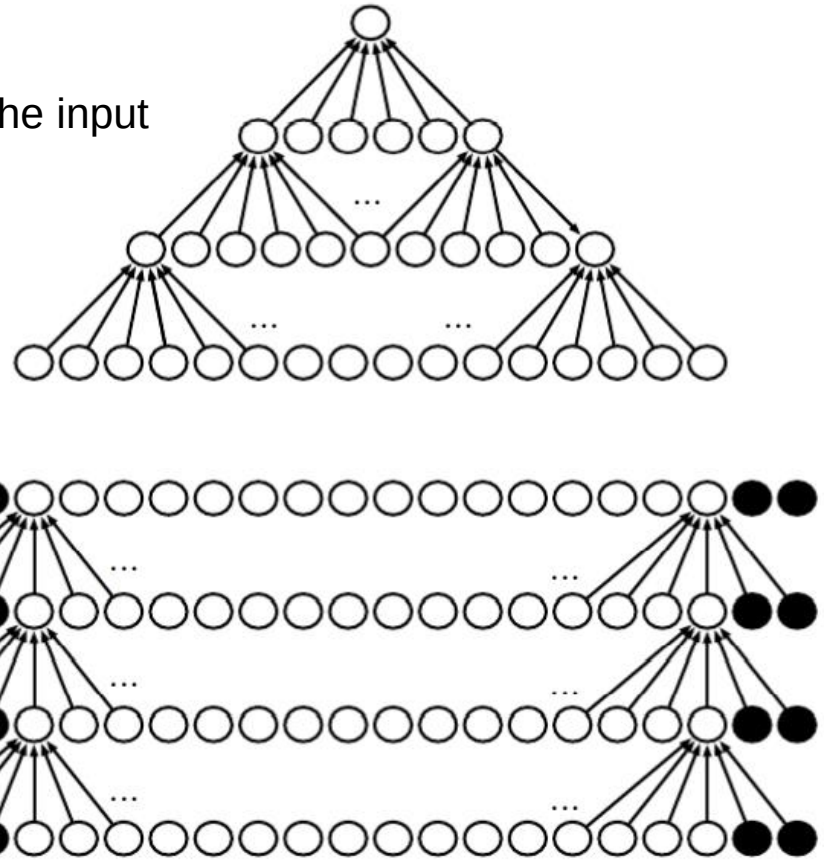


1	2	3	4	5
2	1	0	1	2
3	0	1	1	0
1	4	1	1	2
2	1	1	0	0

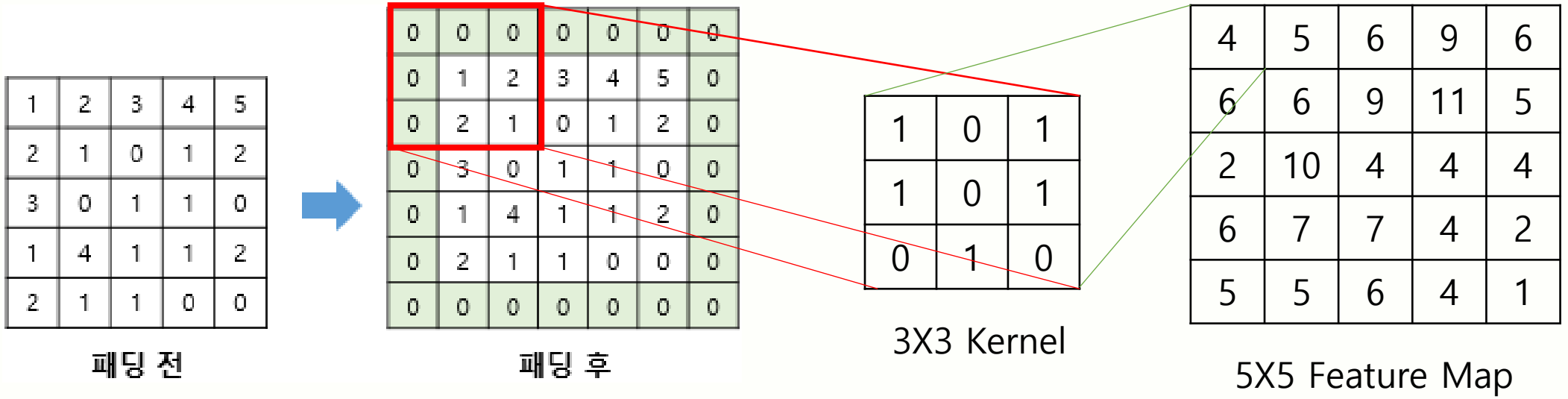
패딩 전

0	0	0	0	0	0	0
0	1	2	3	4	5	0
0	2	1	0	1	2	0
0	3	0	1	1	0	0
0	1	4	1	1	2	0
0	2	1	1	0	0	0
0	0	0	0	0	0	0

패딩 후



Padding (패딩) : Zero - Padding



Zero padding the input allows us to control the kernel width and the size of the output independently.

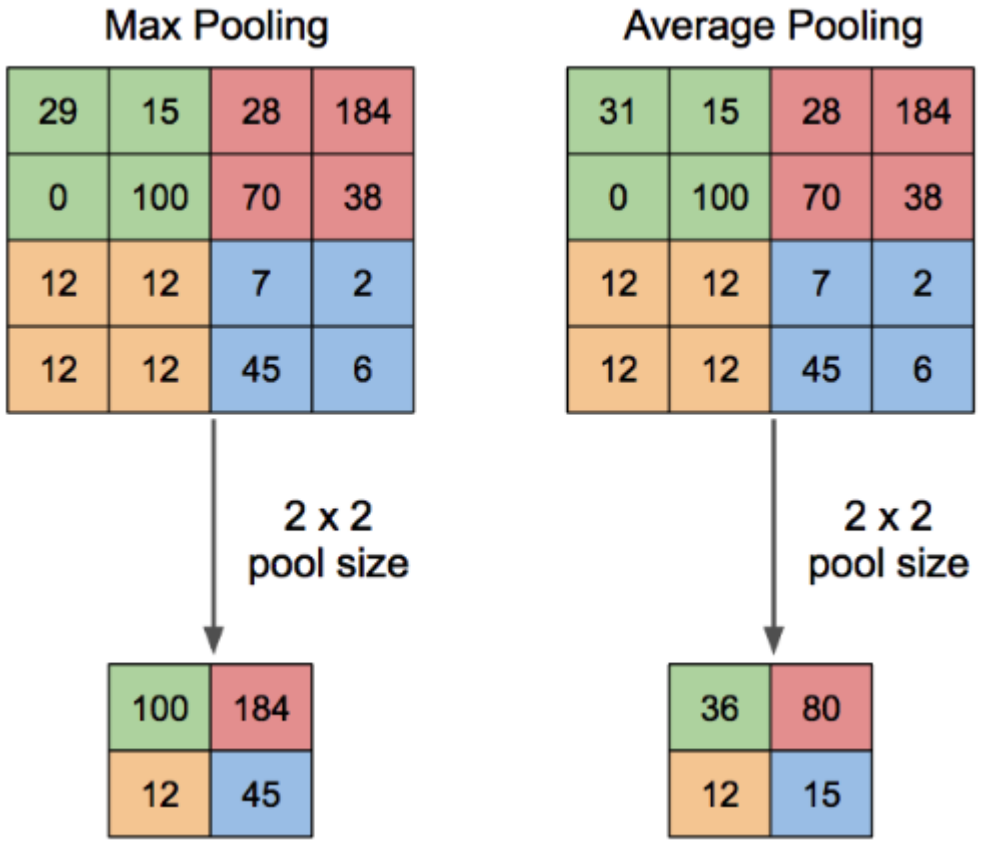
Pooling (풀링)

4X4 → 2X2 down sampling

Pooling layer deceases spatial resolution

- essential for handling inputs of varying size.
- the computational efficiency of the network.
- result in improved statistical efficiency
  and reduced memory requirements
- 가장 중요한 특성이 사라지지 않고 다음 특성으로 넘어감.

Max pooling, Average pooling



Dimension Formula

[Convolution Layer 출력 크기]

$$Output\ size = \frac{input\ size - filter\ size + (2 * padding)}{Stride} + 1$$

[Pooling Layer 출력 크기]

$$OutputRowSize = \frac{InputRowSize}{PoolingSize}$$

$$OutputColumnSize = \frac{InputColumnSize}{PoolingSize}$$

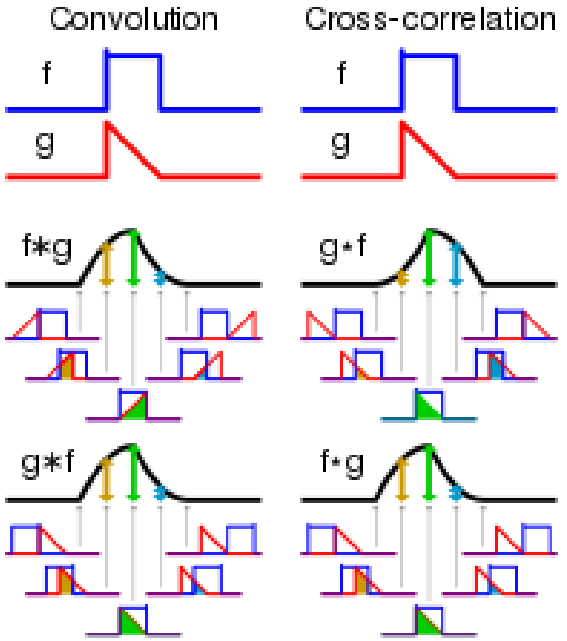
Cross Correlation

[Convolution]

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n).$$

[Cross - Correlation]

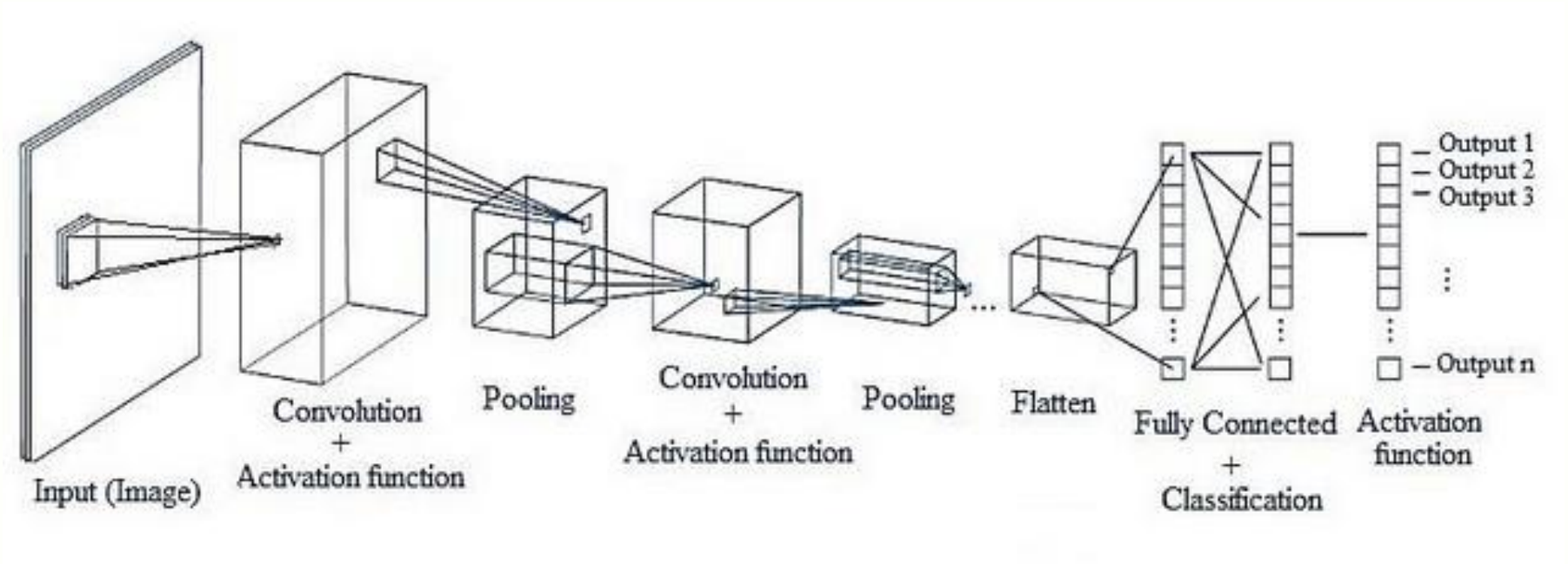
$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n).$$



Cross-correlation is the same as convolution but without flipping the kernel

03. CNN Entire Structure

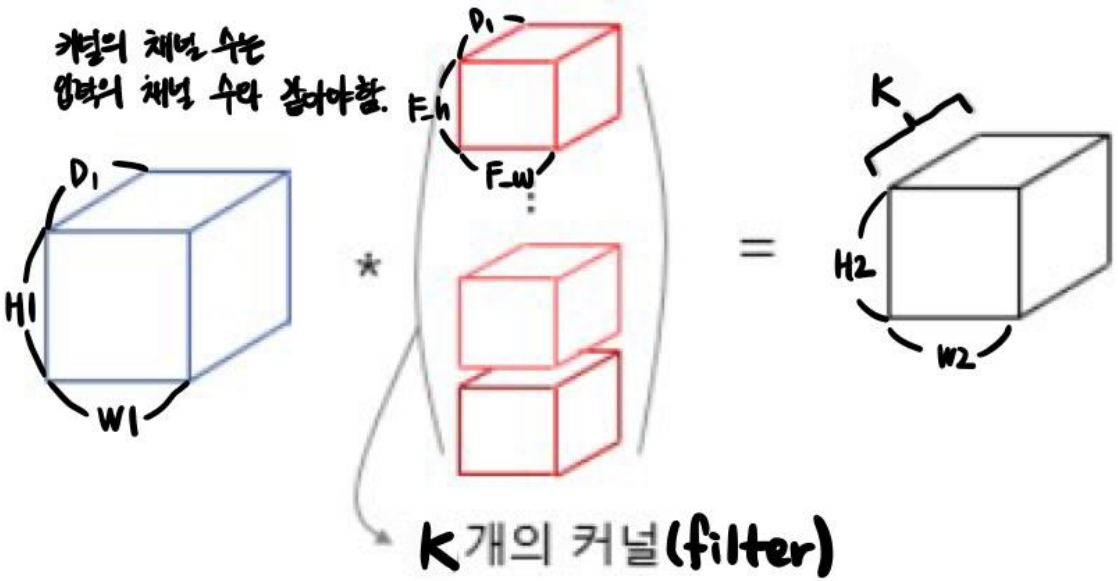
03 CNN Entire Structure



03 CNN Entire Structure

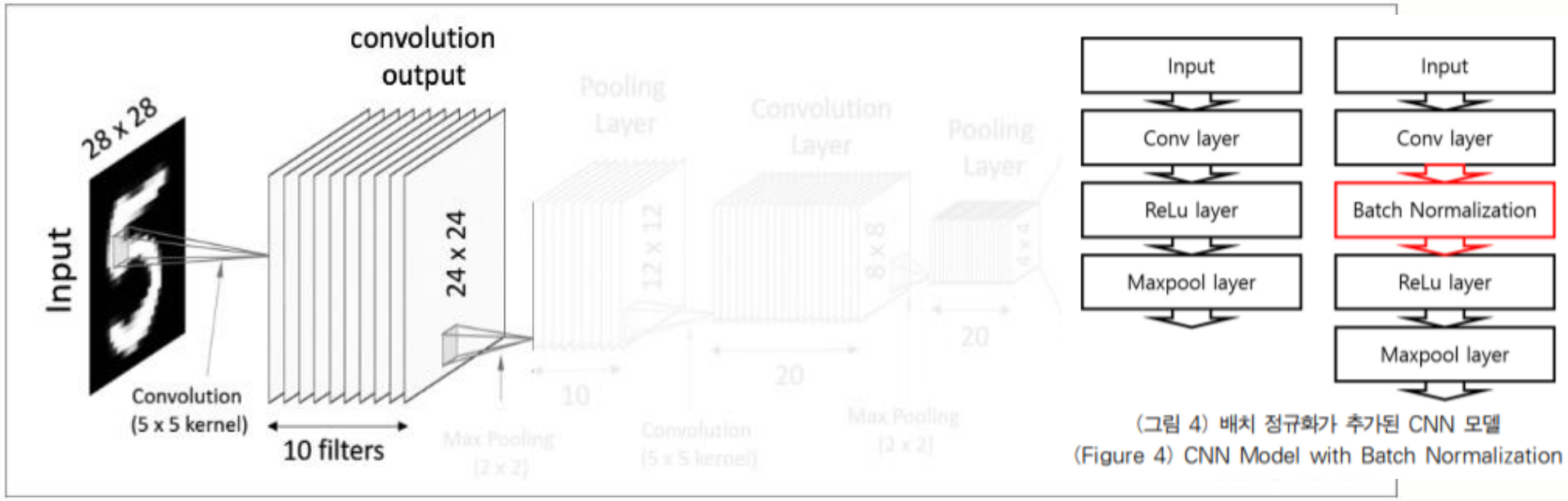
Dimension Formula

- Accepts a volume of size : W1 * H1 * D1
- Requires four hyperparameters
  - Number of filters : K
  - Height of filters : F_h
  - Width of filters : F_w
  - the stride : S
  - the amount of zero padding : P
- Produces a volume of size W2 * H2 * D2
  - W2 = floor((W1-F_w+2P)/S+1)
  - H2 = floor((W2-F_h+2P)/S+1)
  - D2 = K



03 CNN Entire Structure

STEP 1 **1st convolution layer : convolution + activation func**

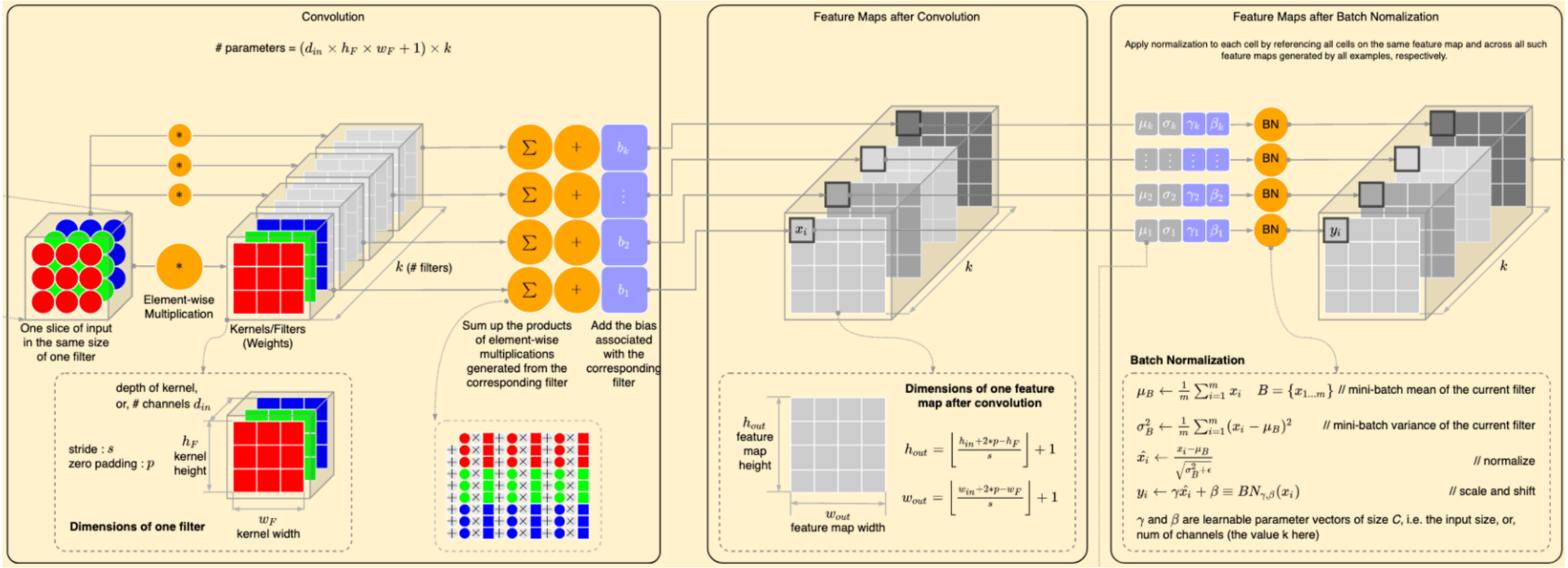


(그림 4) 배치 정규화가 추가된 CNN 모델
(Figure 4) CNN Model with Batch Normalization

03 CNN Entire Structure

STEP 1 **convolution layer에서의 batch normalization**

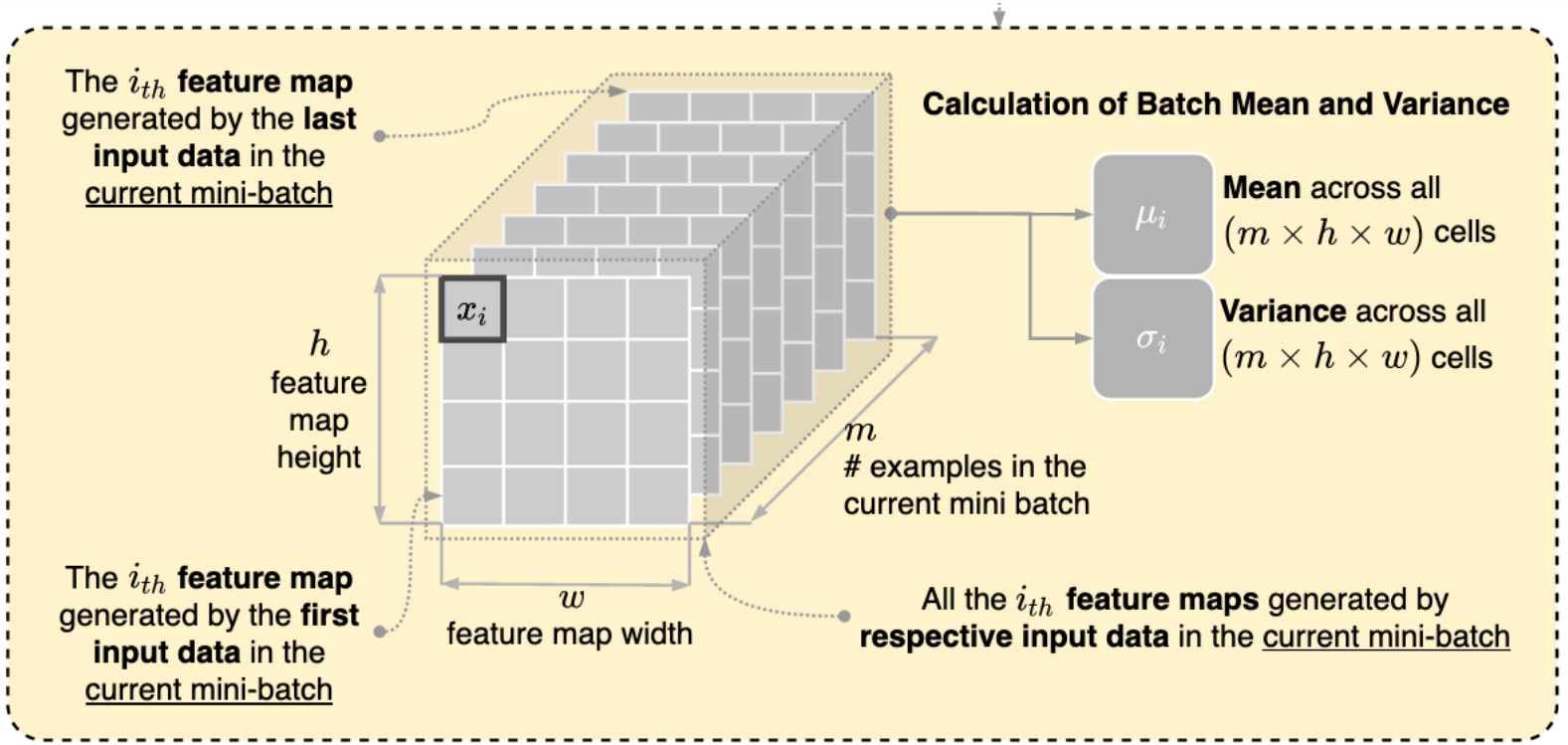
하나의 filter 같은 ɤ, β를 갖는다. 즉, kernel의 개수=feature map의 채널 수=K라고 하면 K개의 독립적인 Batch Normalization 변수들이 생긴다.



03 CNN Entire Structure

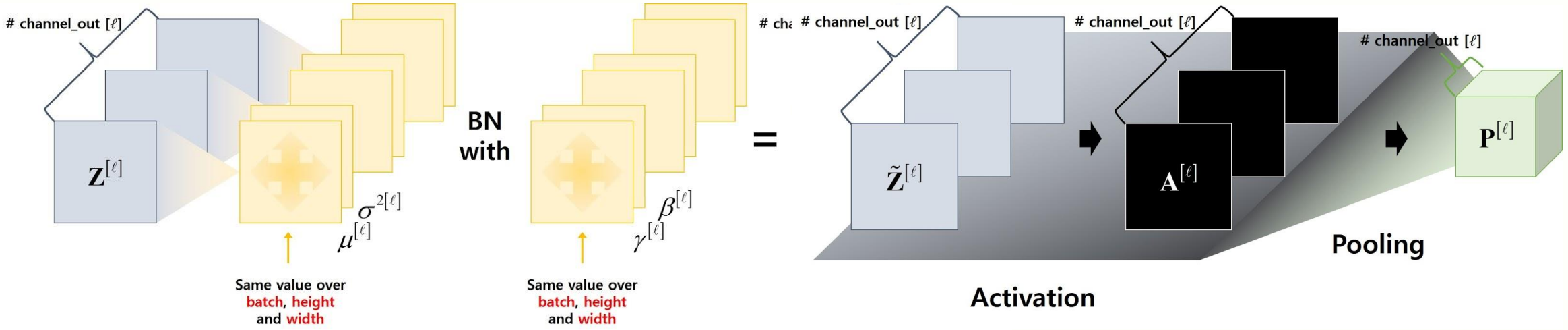
STEP 1 **convolution layer에서의 batch normalization**

하나의 filter 같은 ɤ, β를 갖는다. 즉, kernel의 개수=feature map의 채널 수=K라고 하면 K개의 독립적인 Batch Normalization 변수들이 생긴다.



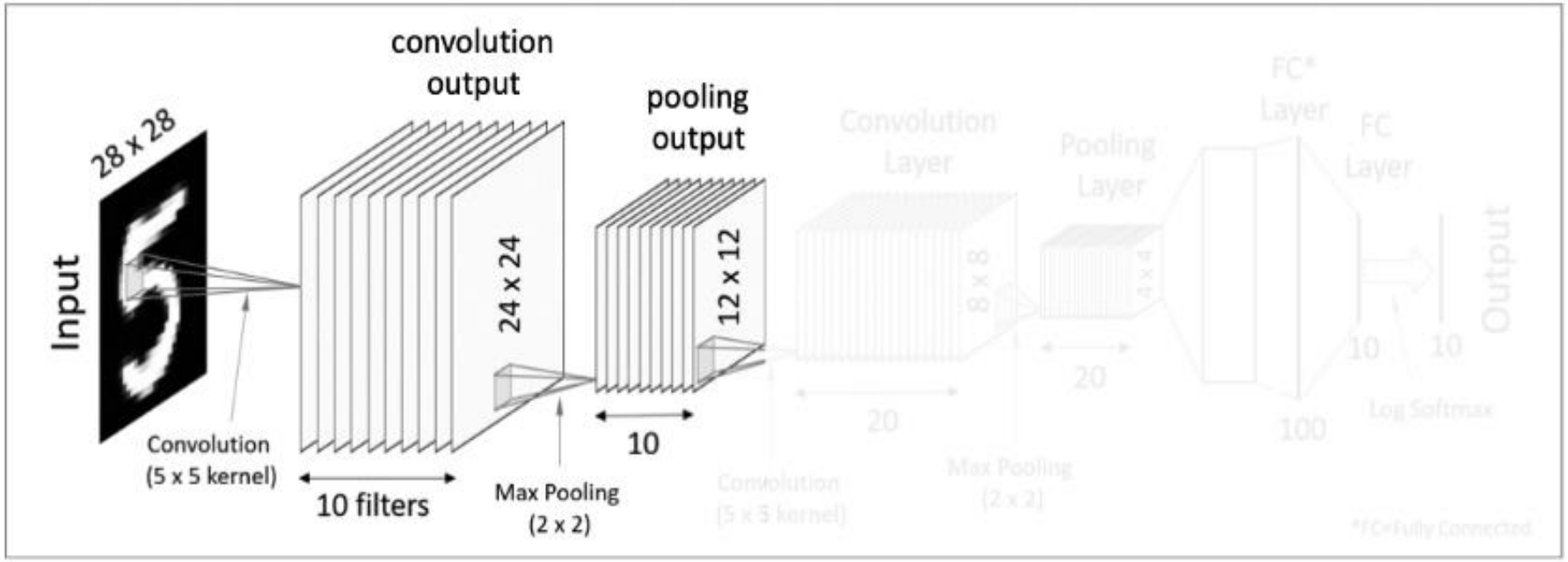
03 CNN Entire Structure

STEP 1 **convolution layer에서의 batch normalization**



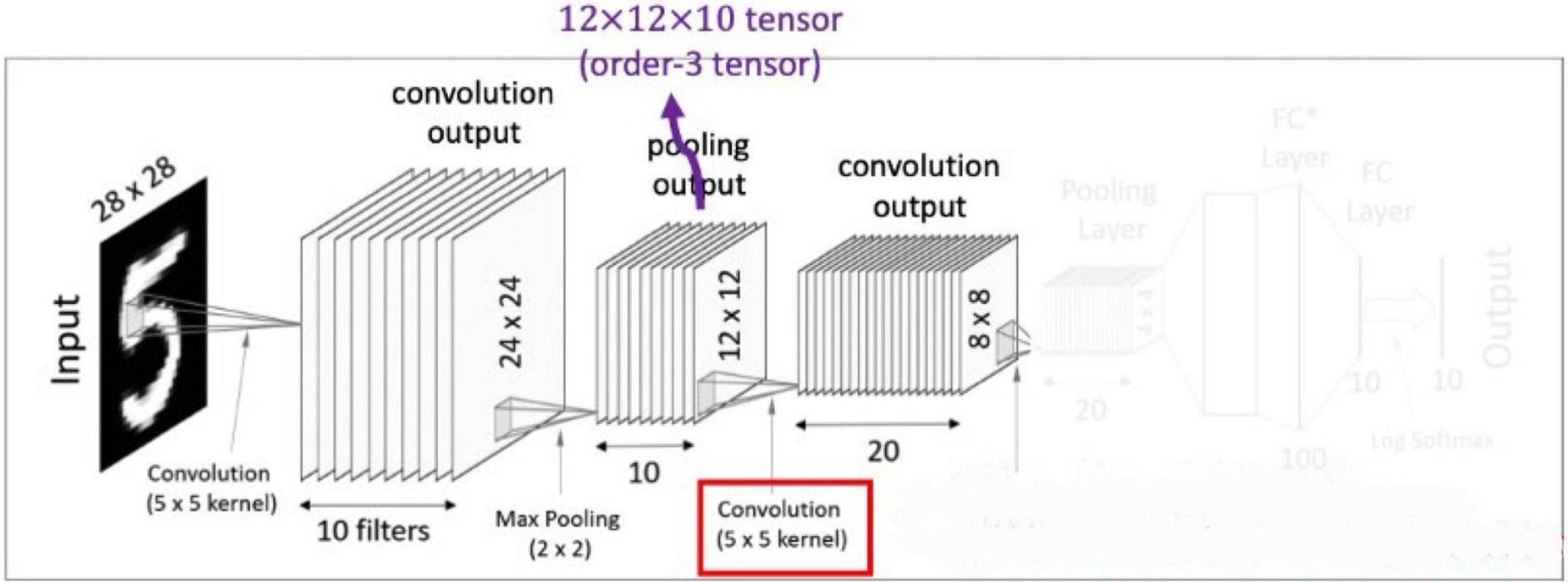
03 CNN Entire Structure

STEP 2 **pooling layer**



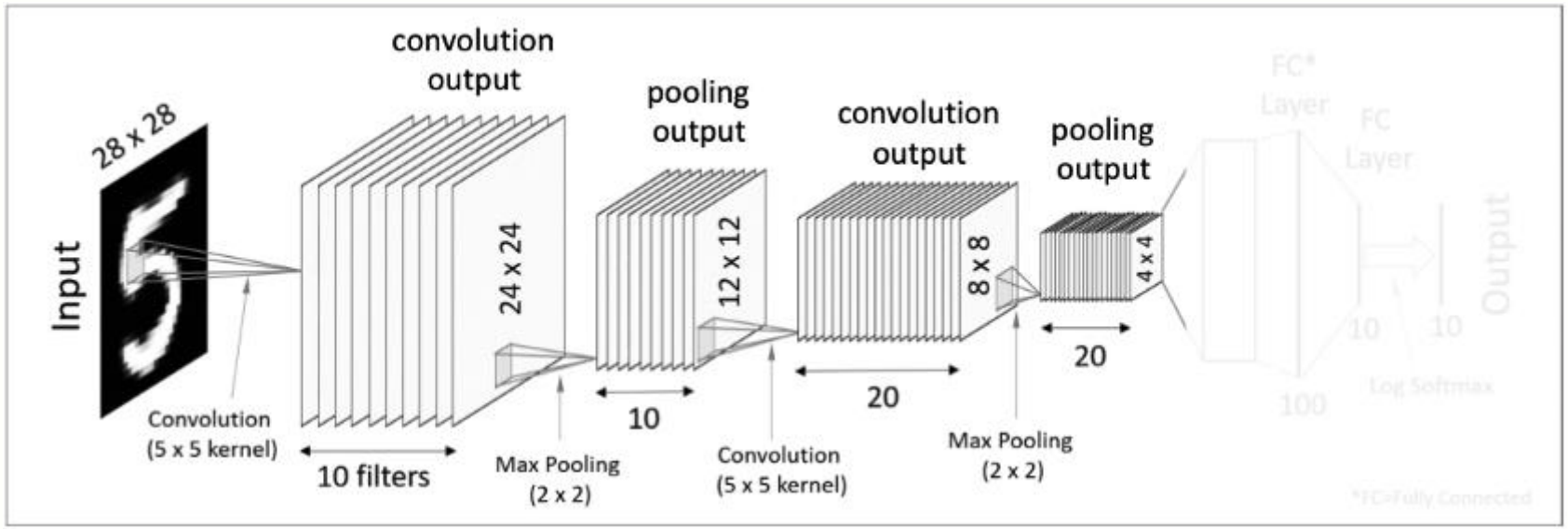
03 CNN Entire Structure

STEP 3 **convolution layer**



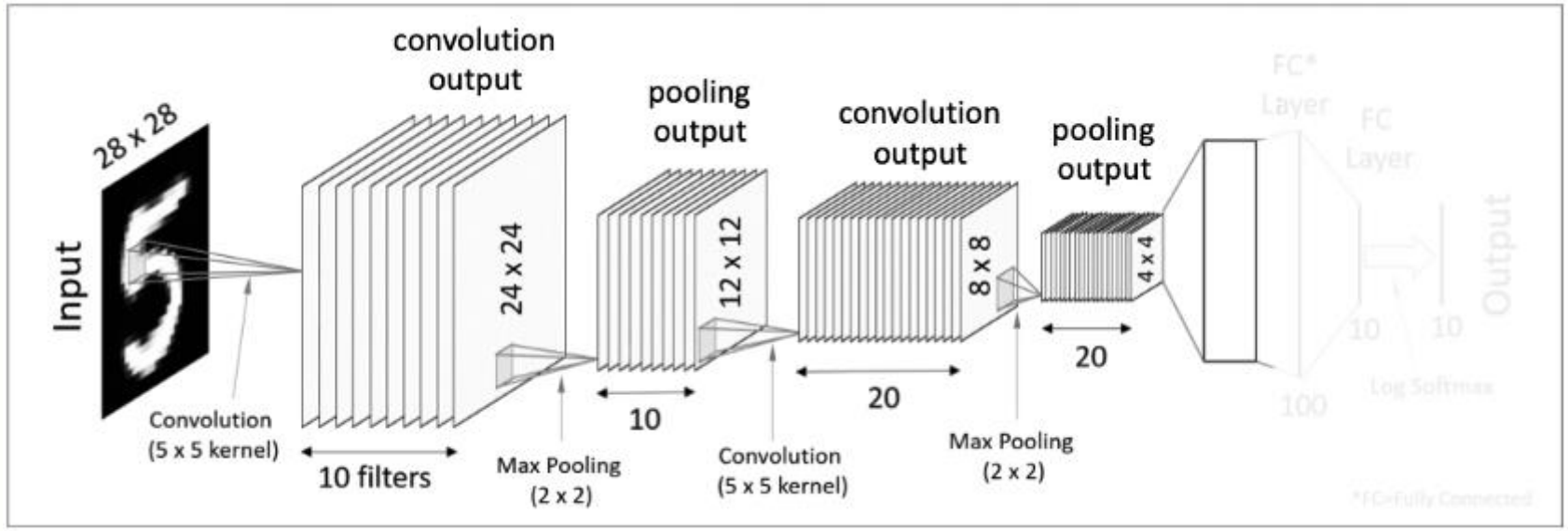
03 CNN Entire Structure

STEP 4 **pooling layer**



03 CNN Entire Structure

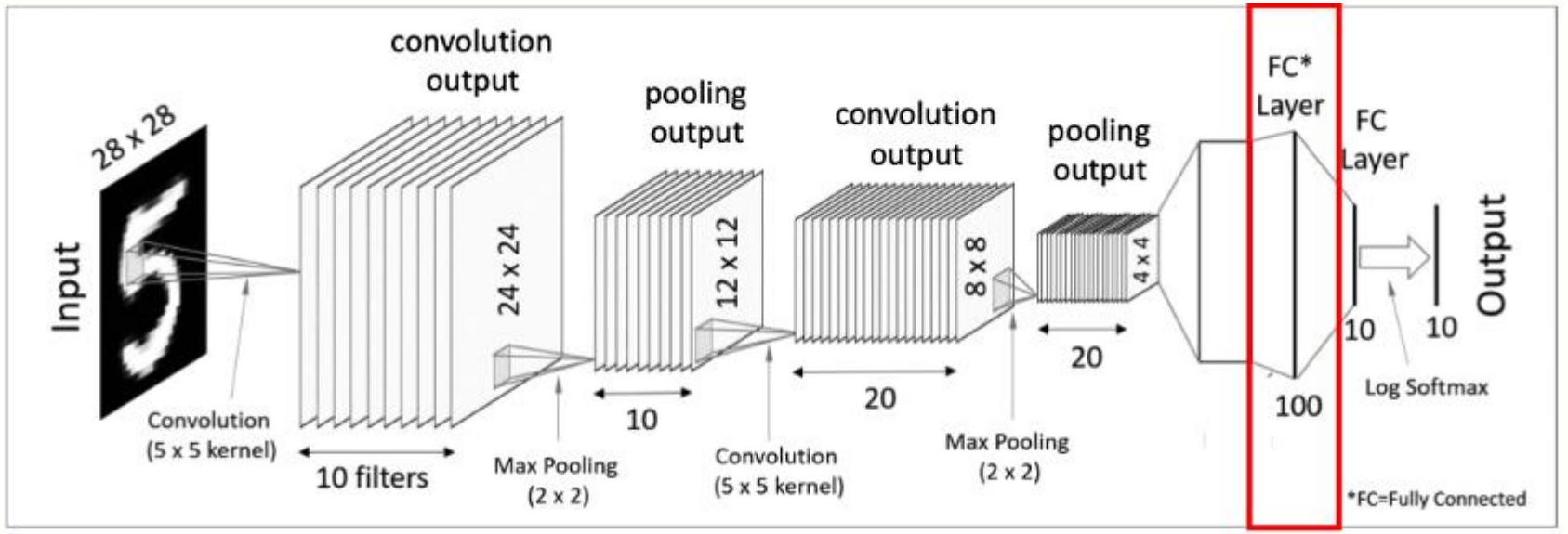
STEP 5 **Flatten(Vectorization)**



Convolution과 Pooling을 반복함으로써 더 이상 이미지가 아닌 이미지로부터 얻은 특이점 데이터가 된다!
따라서 더 이상 2D 데이터로 처리할 이유가 없음.

03 CNN Entire Structure

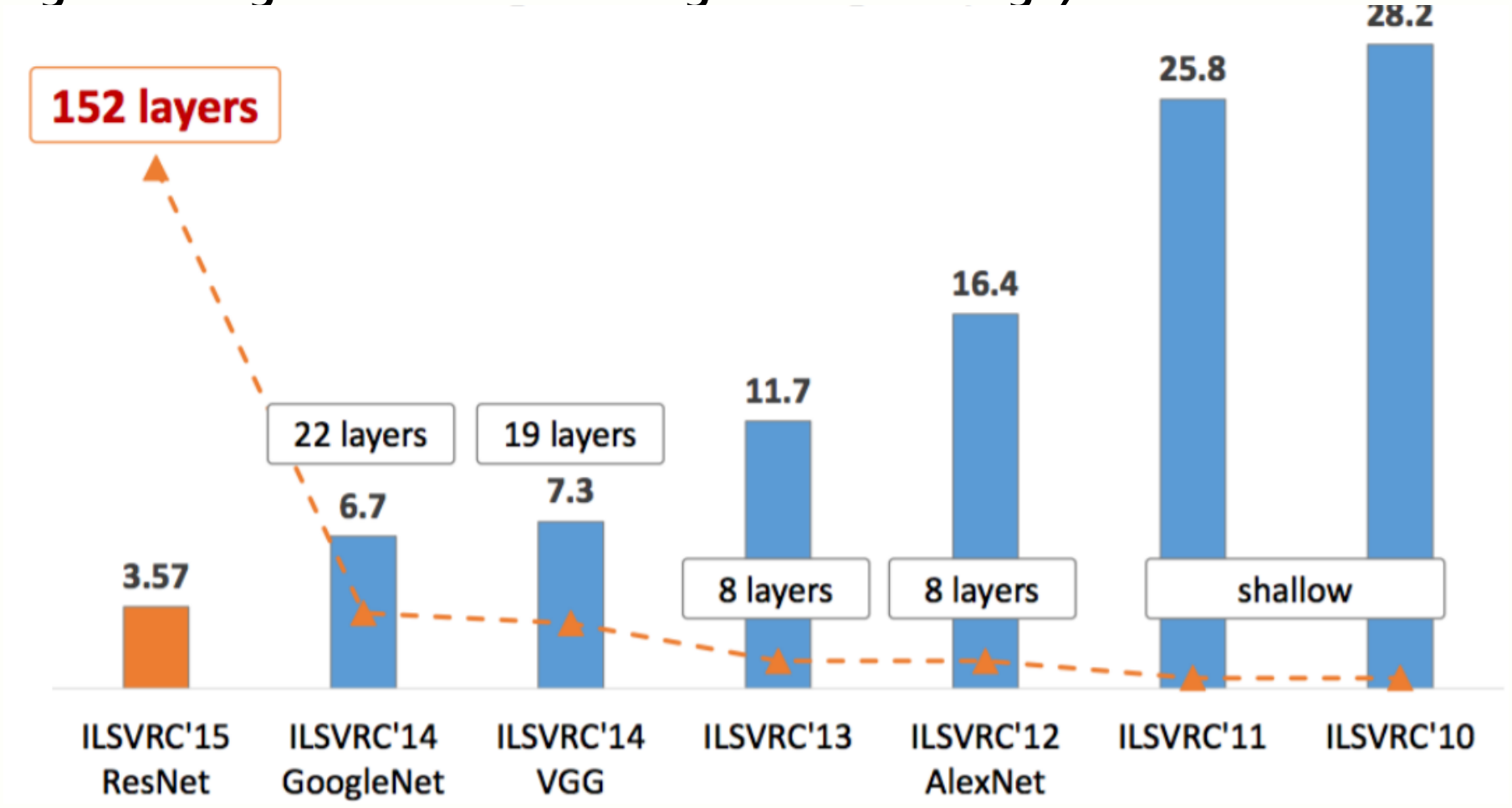
STEP 6 **Fully-Connected Layers**



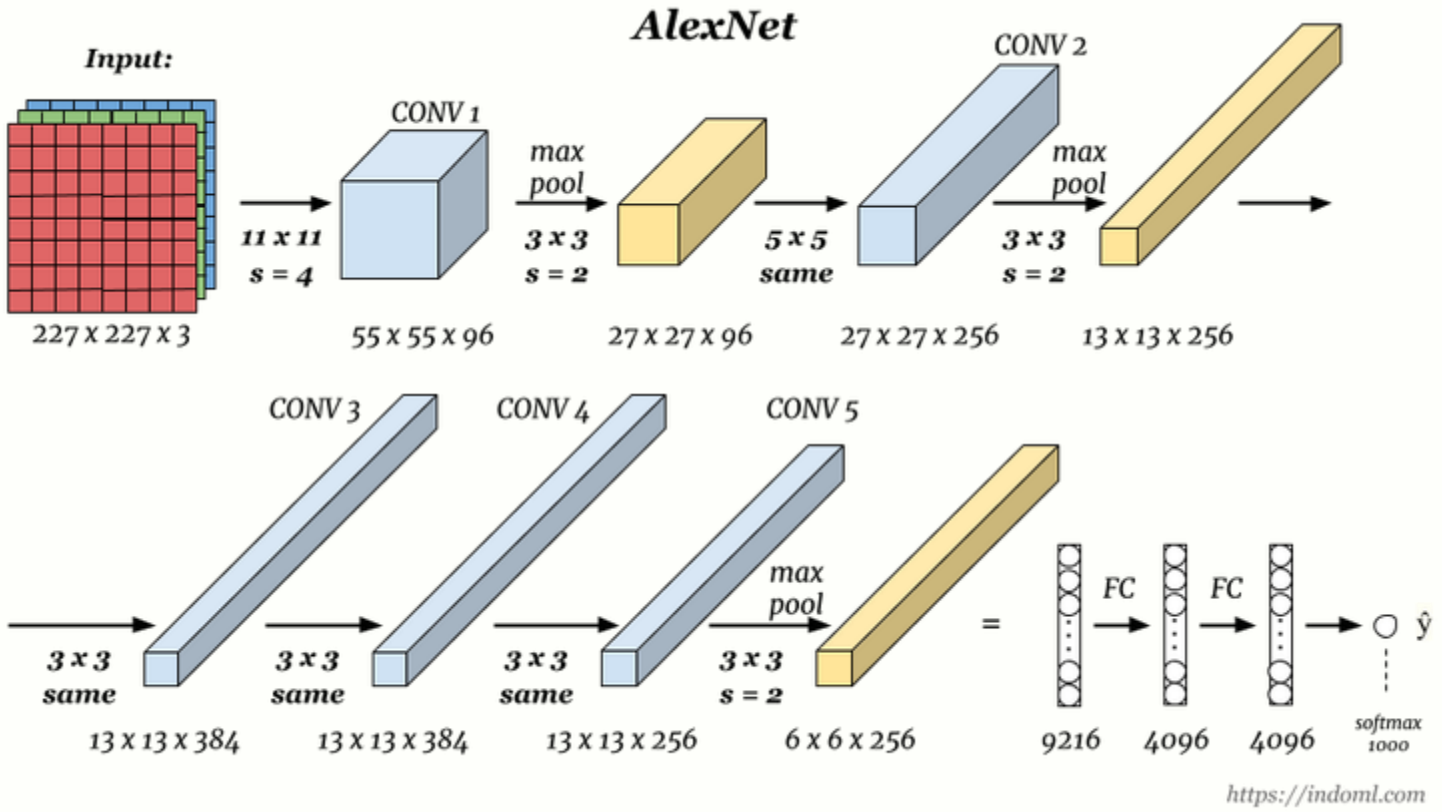
04. Advanced Architecture of CNN

04 Advanced Architectures of CNN

ILSVRC(ImageNet Large Scale Visual Recognition Challenge)



04 Advanced Architectures of CNN : AlexNet

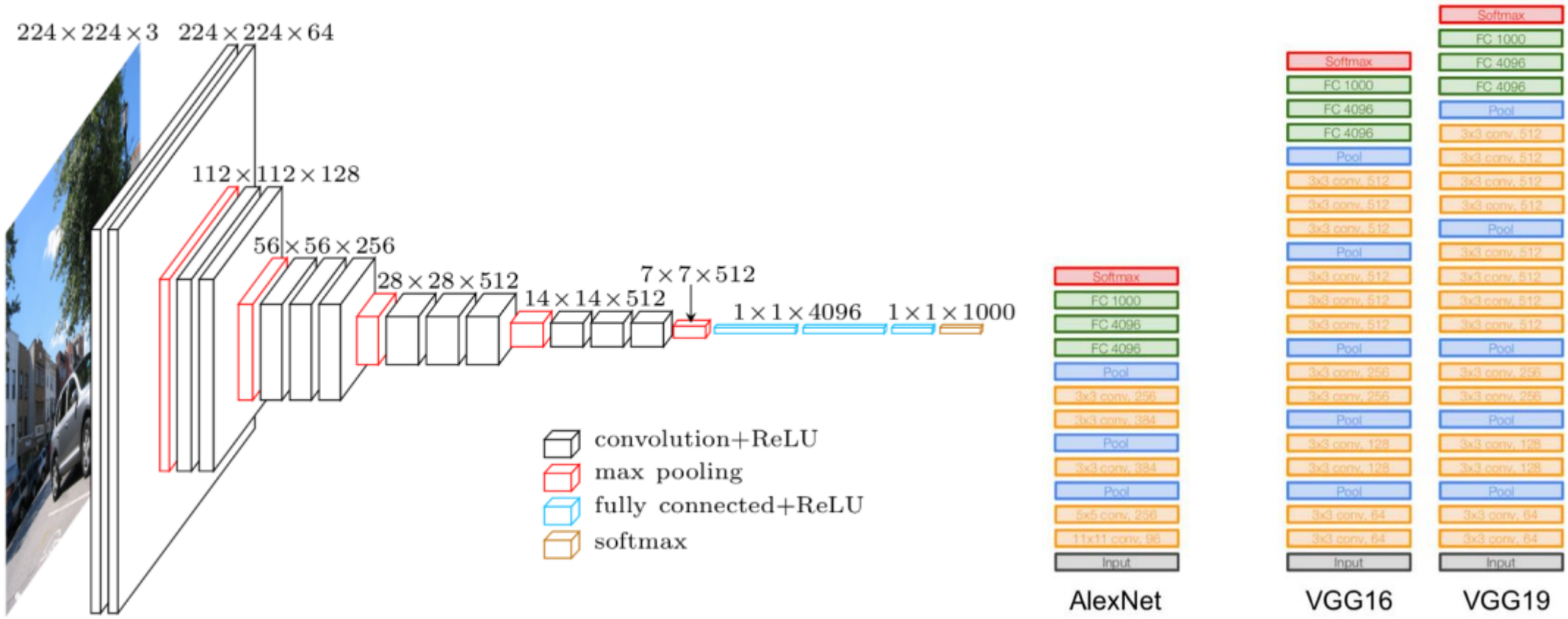


04 Advanced Architectures of CNN : AlexNet

Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

04 Advanced Architectures of CNN : VGGNet



04 Advanced Architectures of CNN : VGGNet

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13
(ZFNet)
-> 7.3% top 5 error in ILSVRC'14

(not counting biases)

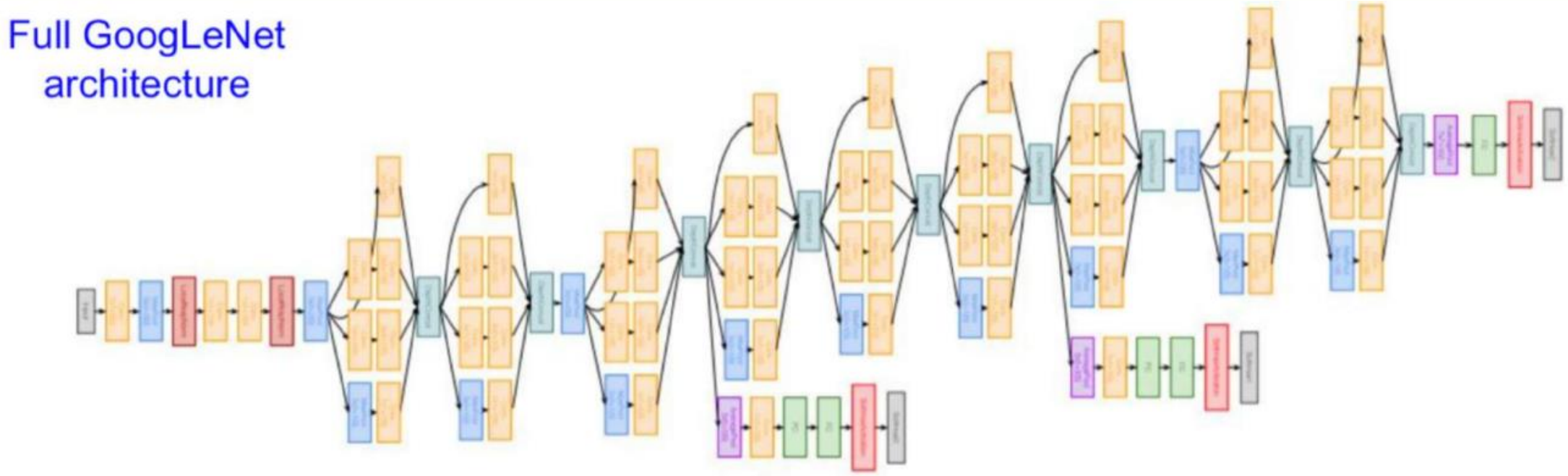
INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

04 Advanced Architectures of CNN : VGGNet

Receptive Field : feature map 한 칸이 영향을 받는 input space의 영역!

small filter 쓰면 같은 reception field를 얻기 위해 더 적은 parameter 요구.
그리고 convolution layer를 더 많이 거치면서 non-linearlity 추가됨.

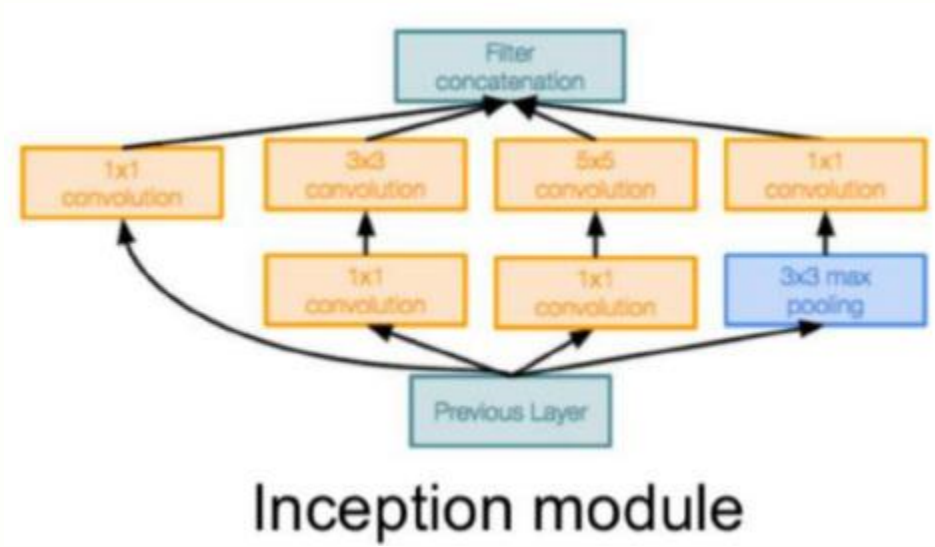
04 Advanced Architectures of CNN : GoogLeNet



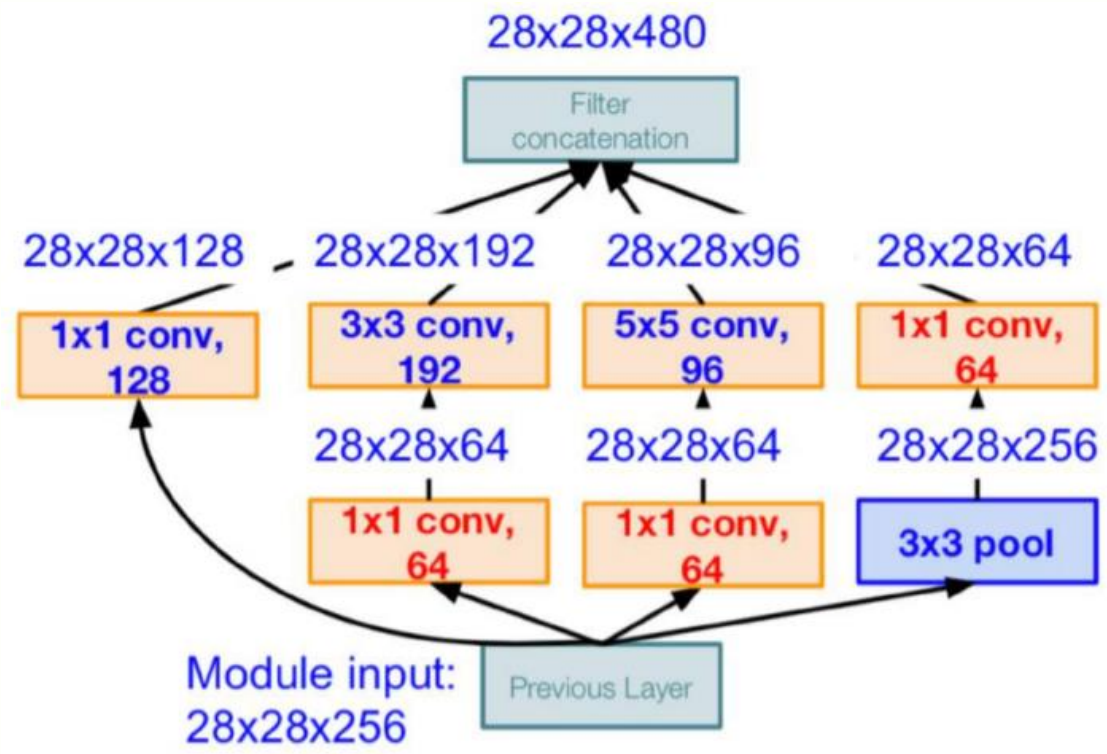
1) 22 layers 2) inception module을 사용하여 연산량을 줄였다.
3) FC layer을 쓰지 않고 **global average pooling** 방법을 사용한다.

04 Advanced Architectures of CNN : GoogLeNet

Inception Module



Inception module



04 Advanced Architectures of CNN : GoogLeNet

1x1 Convolution

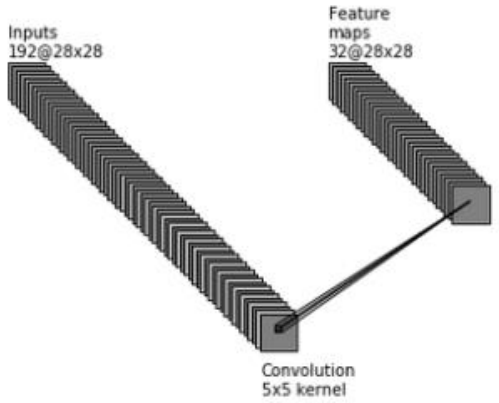


Figure 6. 5×5 convolutions inside the Inception module using the naive model

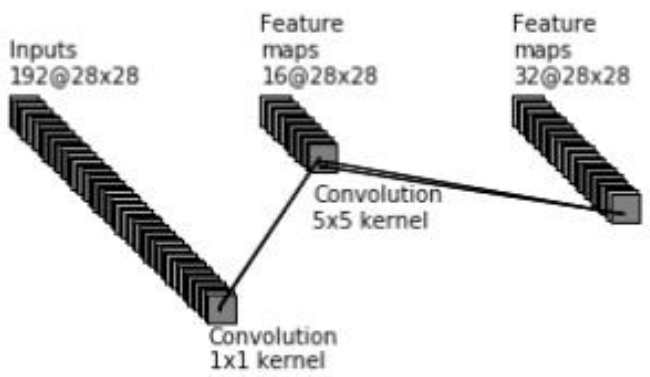


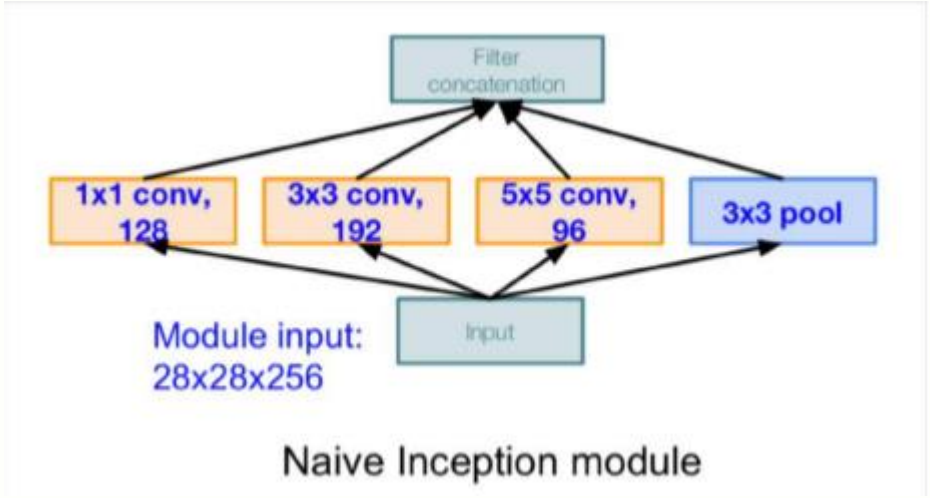
Figure 7. 1×1 convolutions serve as the dimensionality reducers that limit the number of expensive 5×5 convolutions that follow

Without 1x1 convolution:
$5^2 * 28^2 * 192 * 32 = 120,422,400$ operations

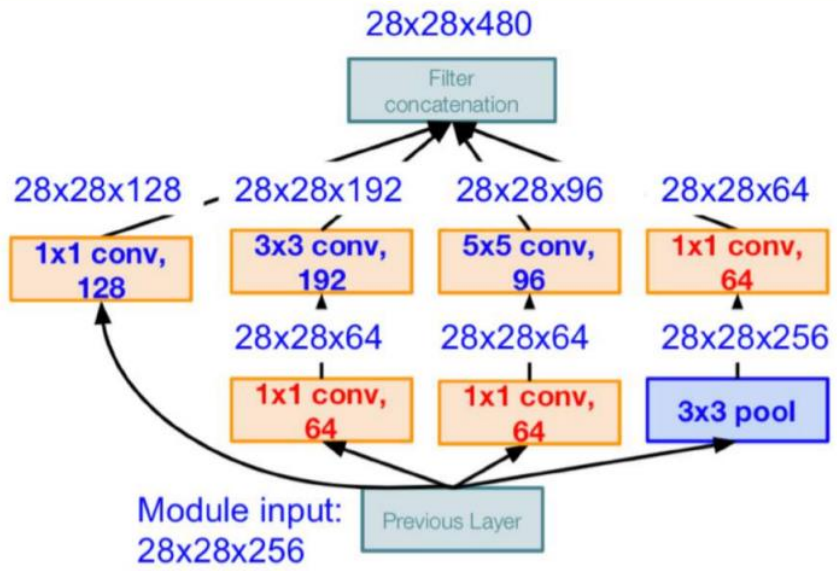
With 1x1 convolution:
$1^2 * 28^2 * 192 * 16$
$+ 5^2 * 28^2 * 16 * 32$
$= 12,443,648$ operations

04 Advanced Architectures of CNN : GoogLeNet

Inception Module



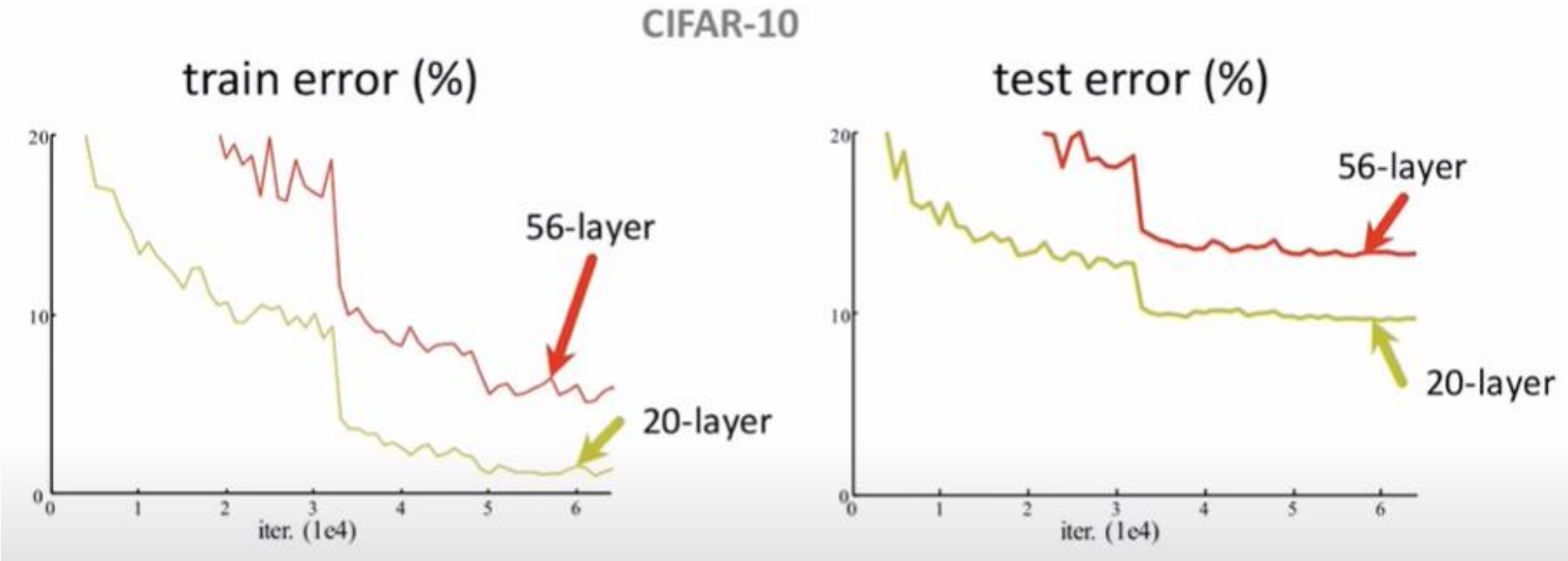
Conv Ops:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x256
[5x5 conv, 96] 28x28x96x5x5x256
Total: 854M ops



Conv Ops:
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256
Total: 358M ops

04 Advanced Architectures of CNN : ResNet

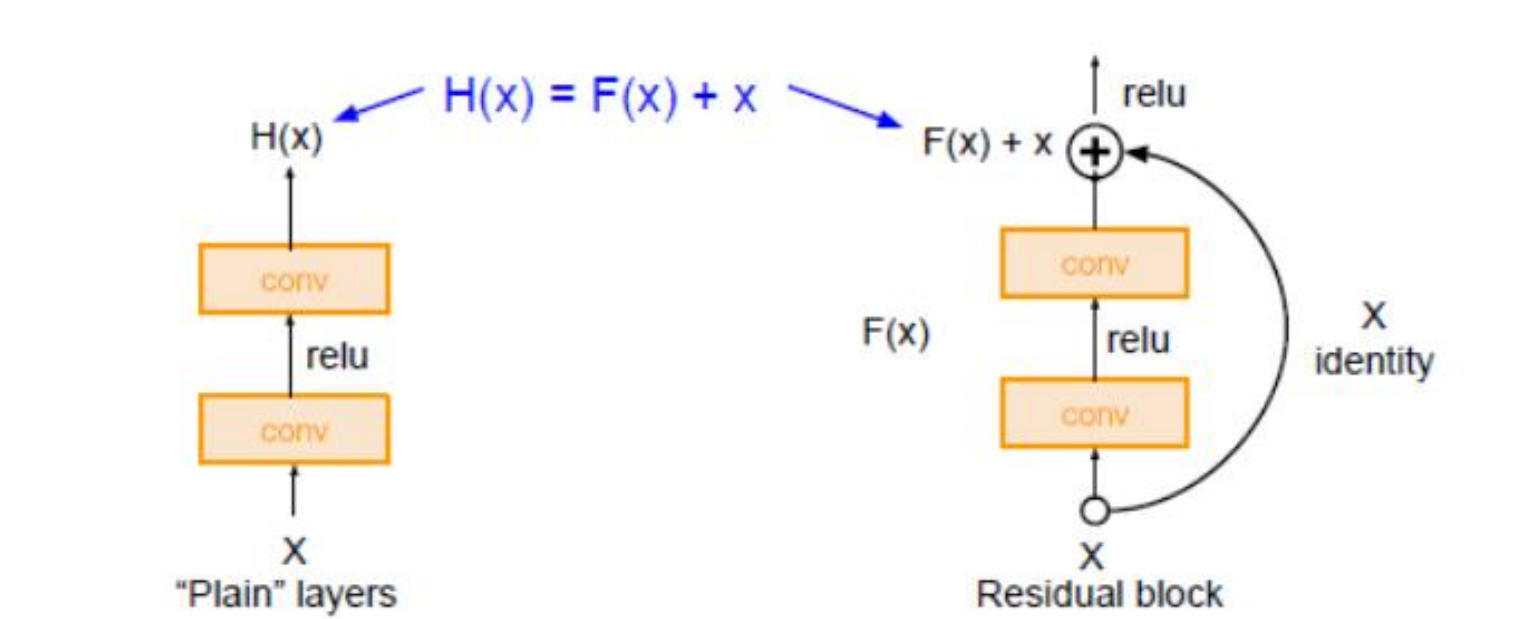
why ResNet?



Layer수가 더 많은, 즉 deep한 network가 train error와 test error 둘 다 높다는 것을 알 수 있다.
Test error의 경우 overfitting의 문제가 발생했다고 볼 수 있고, train error의 경우 layer가 깊어질수록 gradient vanishing 효과때문에 weight에 미치는 영향이 점점 작아진다고 볼 수 있다.

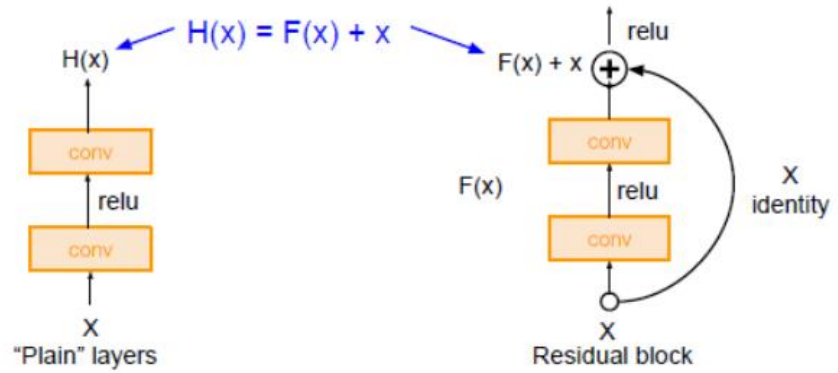
04 Advanced Architectures of CNN : ResNet

Residual Block(BottleNeck Architecture)



04 Advanced Architectures of CNN : ResNet

Residual Block(BottleNeck Architecture)



Mathematically, we can represent the residual block like this.

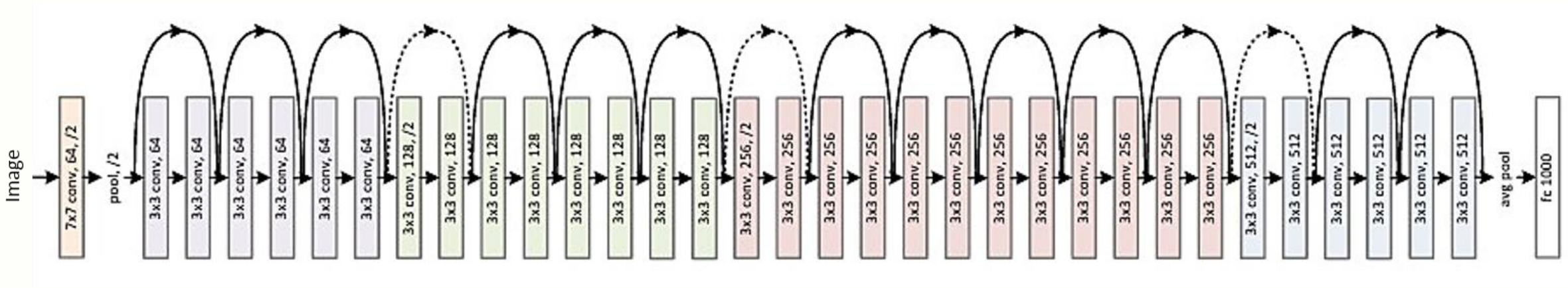
$$H(x) = F(x) + x$$

So when we find the partial derivative of the cost function C with respect to x, we get the following.

$$\begin{aligned}\frac{\partial C}{\partial x} &= \frac{\partial C}{\partial H}\frac{\partial H}{\partial x} \\ &= \frac{\partial C}{\partial H}(\frac{\partial F}{\partial x} + 1) \\ &= \frac{\partial C}{\partial H}\frac{\partial F}{\partial x} + \frac{\partial C}{\partial H}\end{aligned}$$

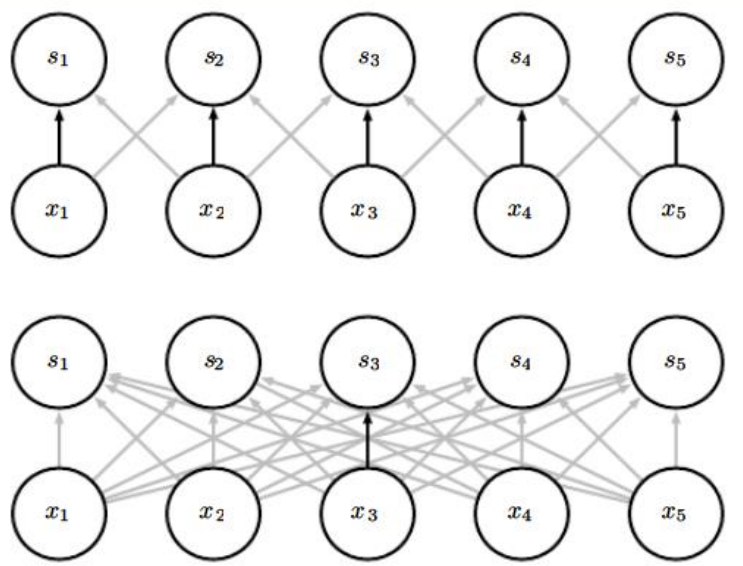
기존 신경망은 입력값 x를 target y로 mapping하는 함수 H(x)를 찾는 것이 목적이었음. 그러나 Residual Block에서는 F(x)+x를 최소화하는 것을 목적으로 하는데 x(identity function)는 주어진 값이고 F를 최소화 (0에 가깝게) 하는 것이랑 동일하다. F(x)=H(x)-x : 잔차

04 Advanced Architectures of CNN : ResNet



05. Varients of the Basic Convolution function

05 Variants of the Basic Convolution Function

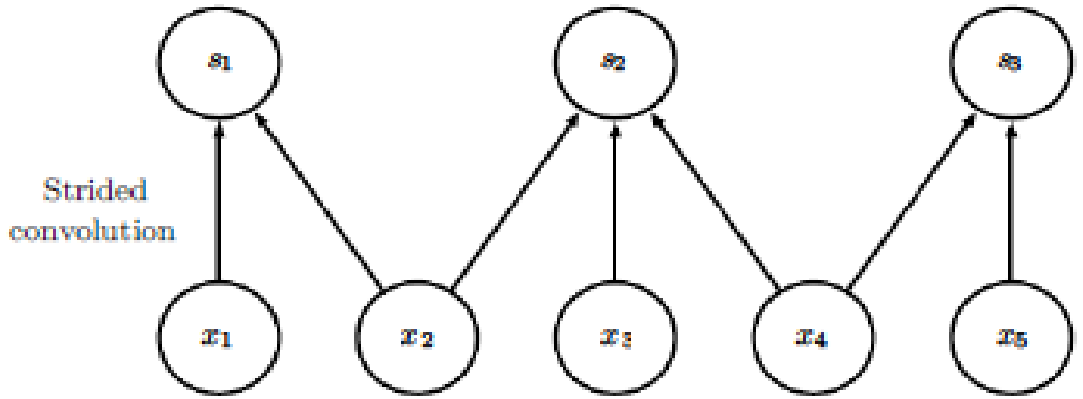


(Top) Convolution Model
(Bottom) Fully Connected Model

$$Z(i,j,k) = \sum_l \sum_m \sum_n I(l, j+m-1, k+n-1) * K(i,l,m,n)$$

05 Variants of the Basic Convolution Function

1) Convolution with stride



$$Z(i,j,k) = c(K,V,s)$$
$$= \sum_l \sum_m \sum_n I(l,(j-1)*s+m,(k-1)*s+n) * K(i,l,m,n)]$$

s : stride

05 Variants of the Basic Convolution Function

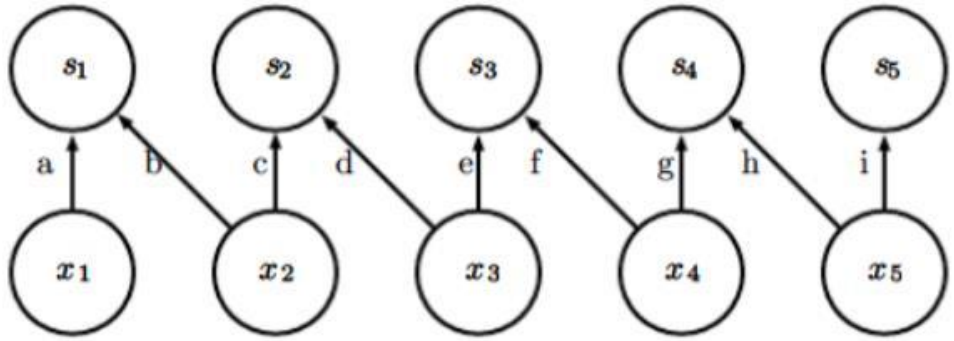
2) Unshared Convolution(Locally connected)

$$Z(i,j,k) = \sum_{l}\sum_{m}\sum_{n} I(l, j+m-1, k+n-1) * w(i,j,k,l,m,n)$$

i : output's channel j : output's row k : output's column
l : input's channel m : input's row n : input's column
$w(i,j,k,l,m,n)$: 6D tensor

05 Variants of the Basic Convolution Function

2) Unshared Convolution(Locally connected)



A locally connected layer
Has no sharing at all
Each connection has its own weight

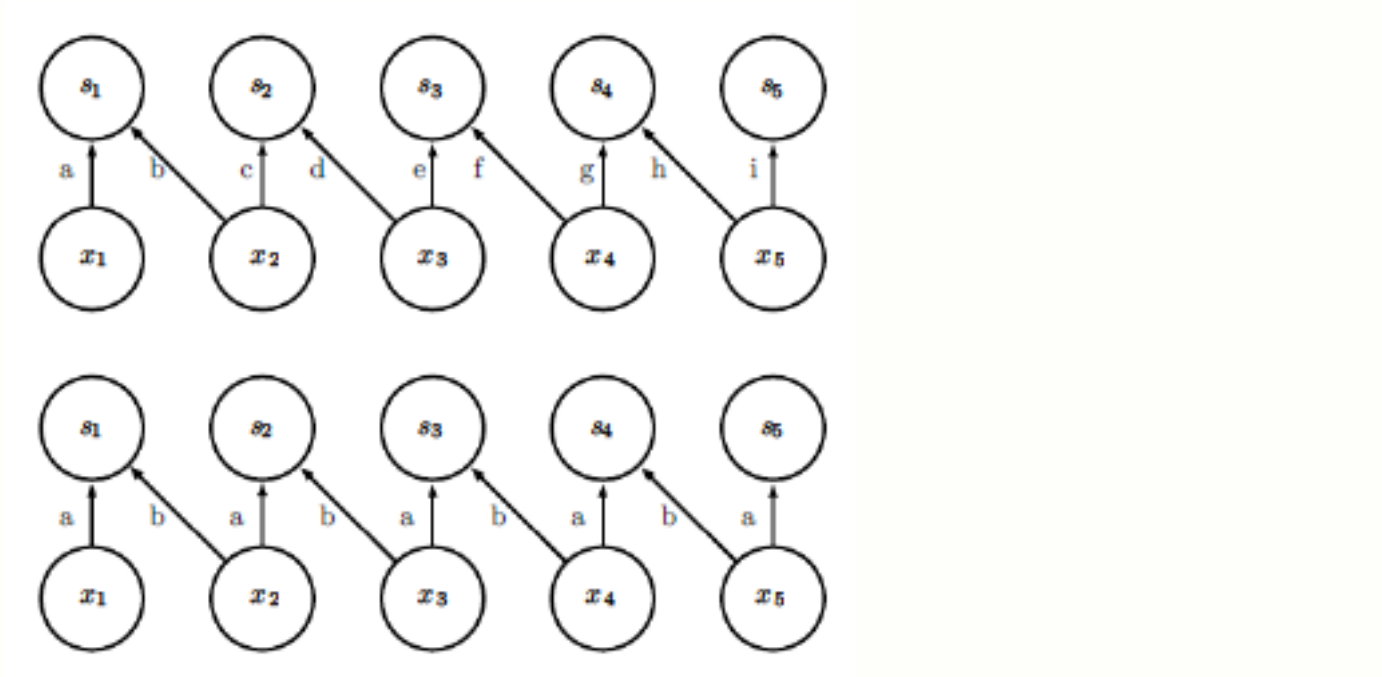
05 Variants of the Basic Convolution Function

2) Unshared Convolution(Locally connected)

Pros : each feature should be a function of a small part of space and same feature is not occurred across all of space (useful)
Cons : 다른 convolution network보다 연산량, parameter 수가 많아진다.

05 Variants of the Basic Convolution Function

3) Tiled Convolution



Comparison between unshared convolution(TOP)
and tiled convolution(BOTTOM)

05 Variants of the Basic Convolution Function

3) Tiled Convolution

Pros : Unshared Convolution과 standard convolution의 절충. 매개변수를 저장하는 데 필요한 메모리 요구량이 Kernel의 크기에만 비례하여 증가. (vs. Unshared Convolution은 entire output feature map size에 비례하여 증가.)

05 Variants of the Basic Convolution Function

3) Tiled Convolution

$$Z(i,j,k) = \sum_{l}\sum_{m}\sum_{n} I(l, j+m-1, k+n-1) * K(i,j,m,n,j\%t+1,k\%t+1)$$

%는 나머지 연산을 의미. Ex) t%t=0, (t+1)%t=1

05 Variants of the Basic Convolution Function

이 외에도 많은 convolution 기법들이 존재합니다!

Homework

앞서 배운 convolution function을 제외한 **3가지의 다른 convolution 기법**들을 조사하고, **어떠한 원리로 작동**하는지, 기존의 convolution과 비교했을 때

어떠한 특성을 가지고 있는지 조사해주세요!

ex) Dilated convolution, Transposed convolution, Depthwise convolution, ...

감 사 합 니 다

T H A N K Y O U