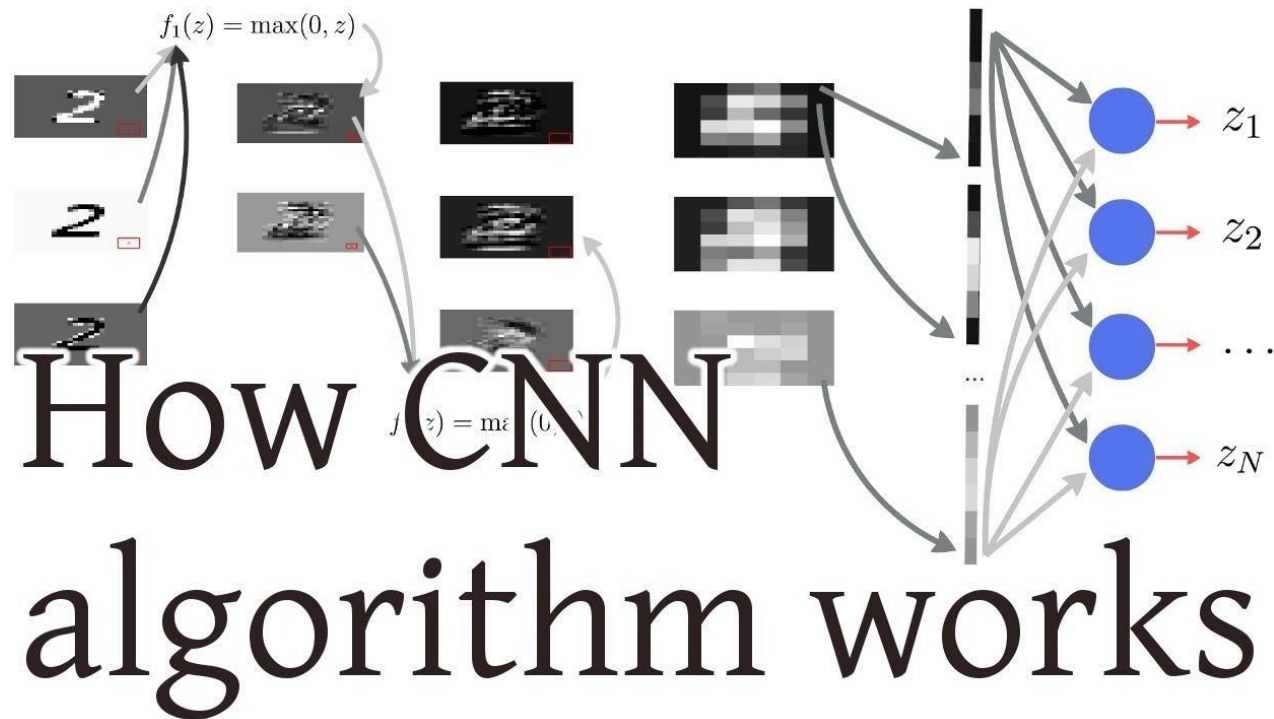
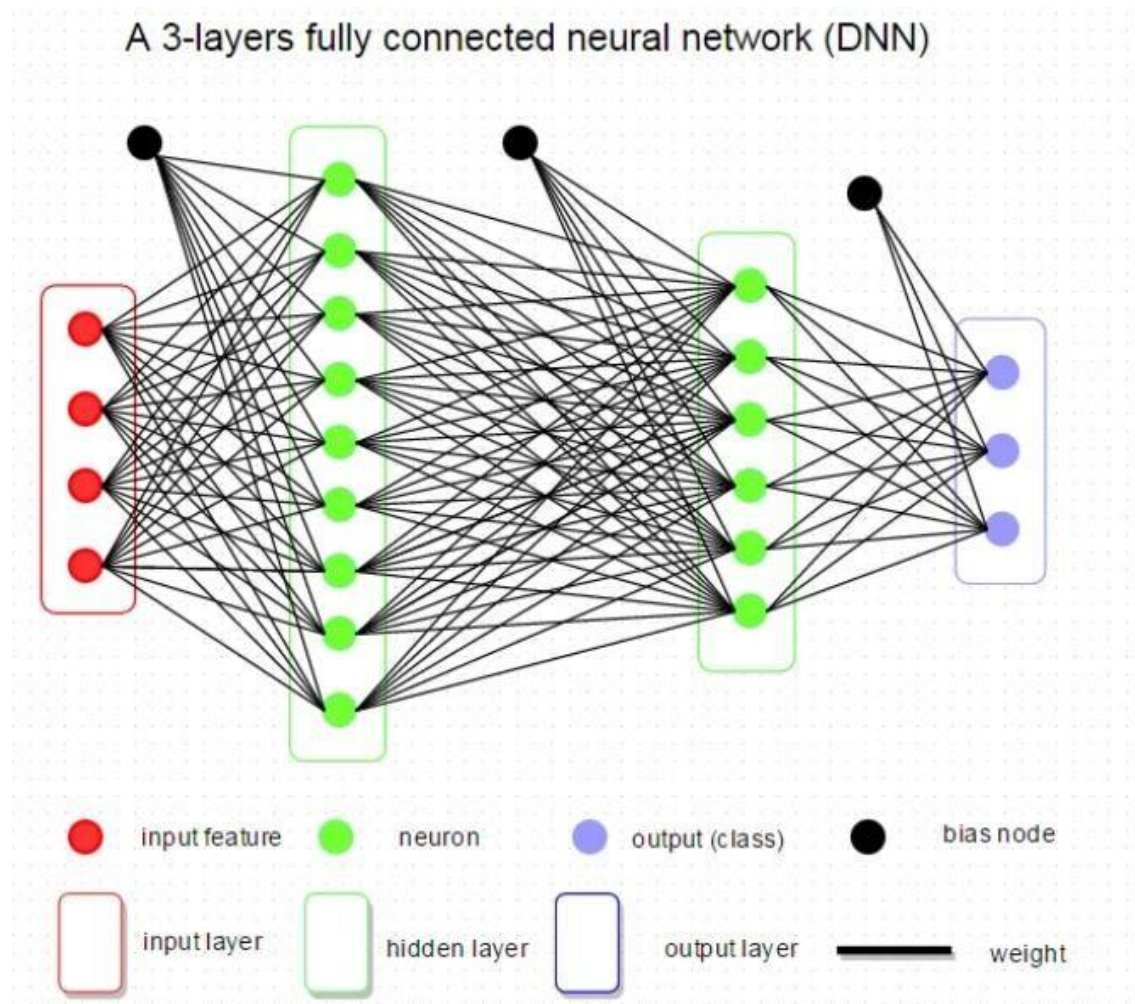


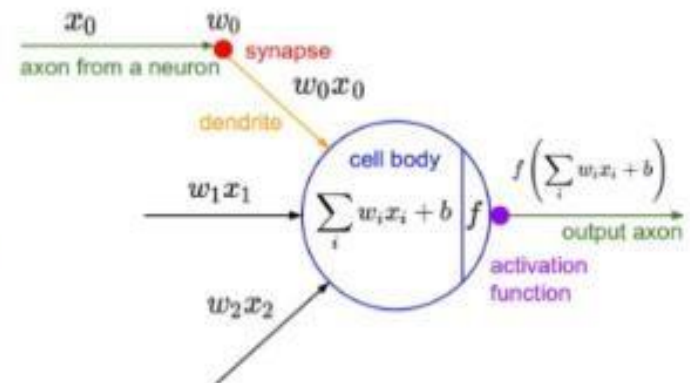
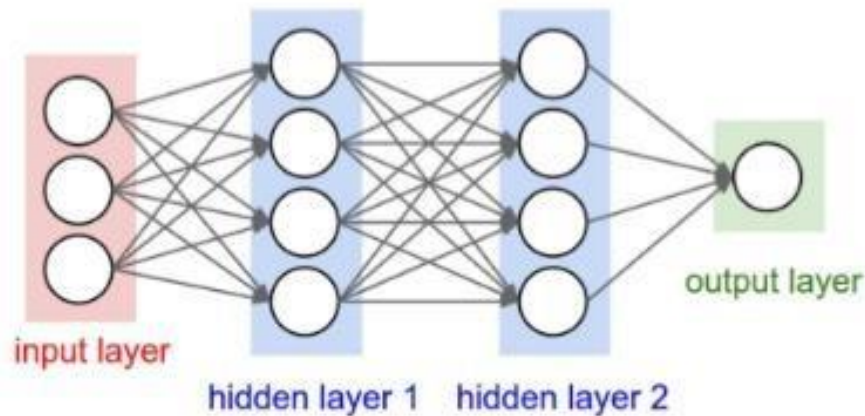
Convolutional Neural Network (CNN)



[복습] Fully Connected Layer (전결합 계층)

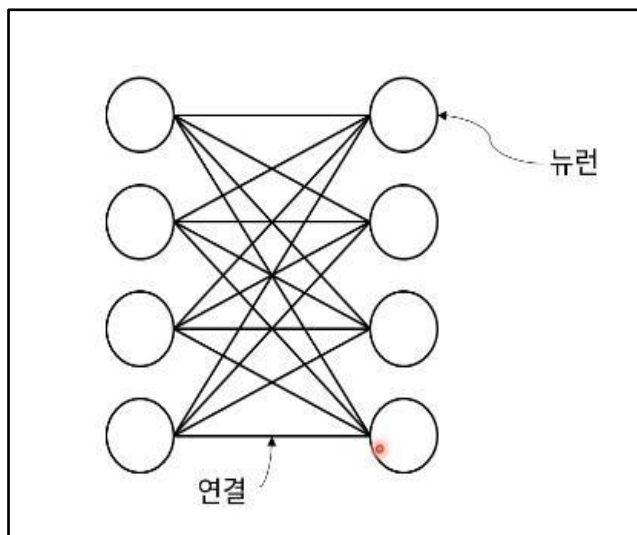


Fully Connected Layers

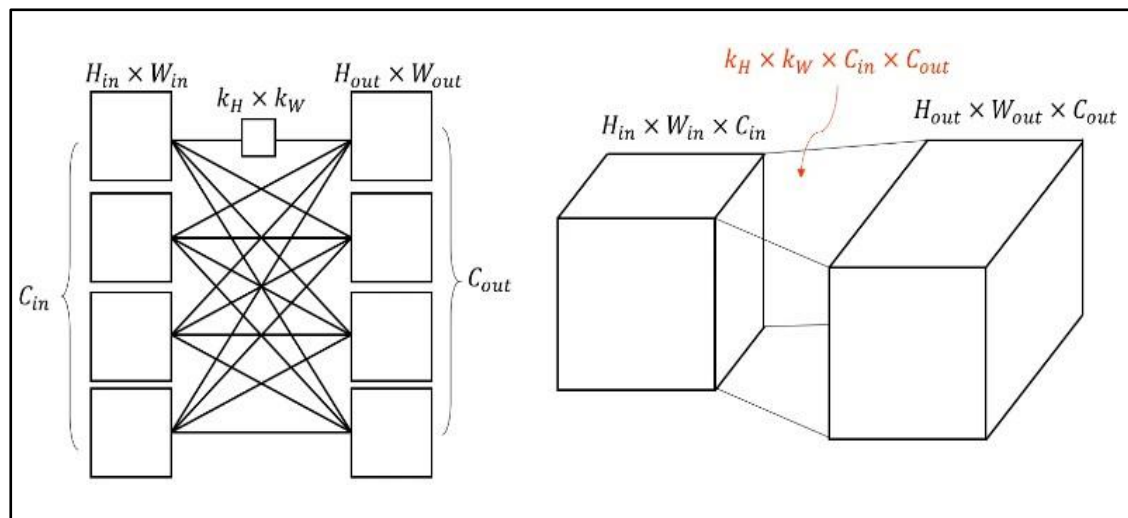


1. What is Convolutional Layer?

[Fully Connected Layer] 전 결합 계층



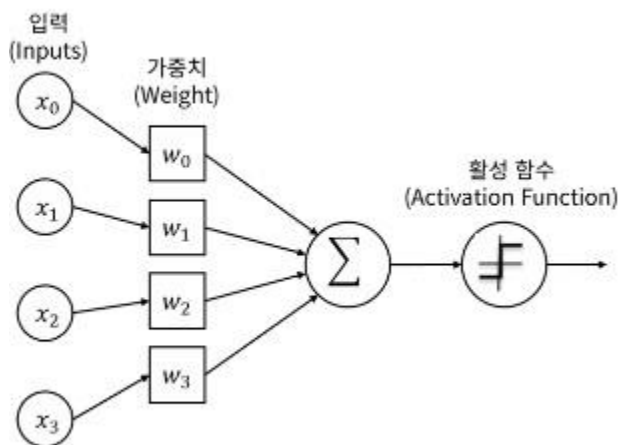
[Convolutional Layer] 합성곱 계층



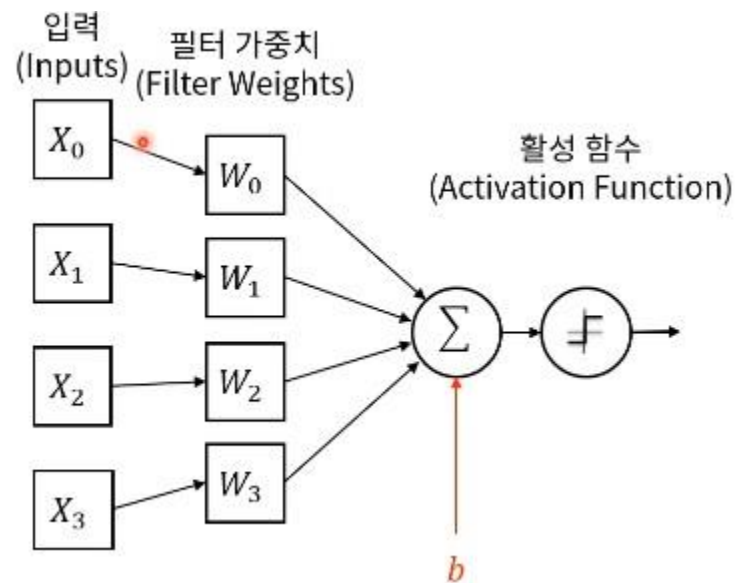
What's the difference?

1. What is Convolutional Layer?

[Fully Connected Layer] 적결
합 계층



[Convolutional Layer] 합
성곱 계층



(FC) 입력 -> (Conv) 사진/영상

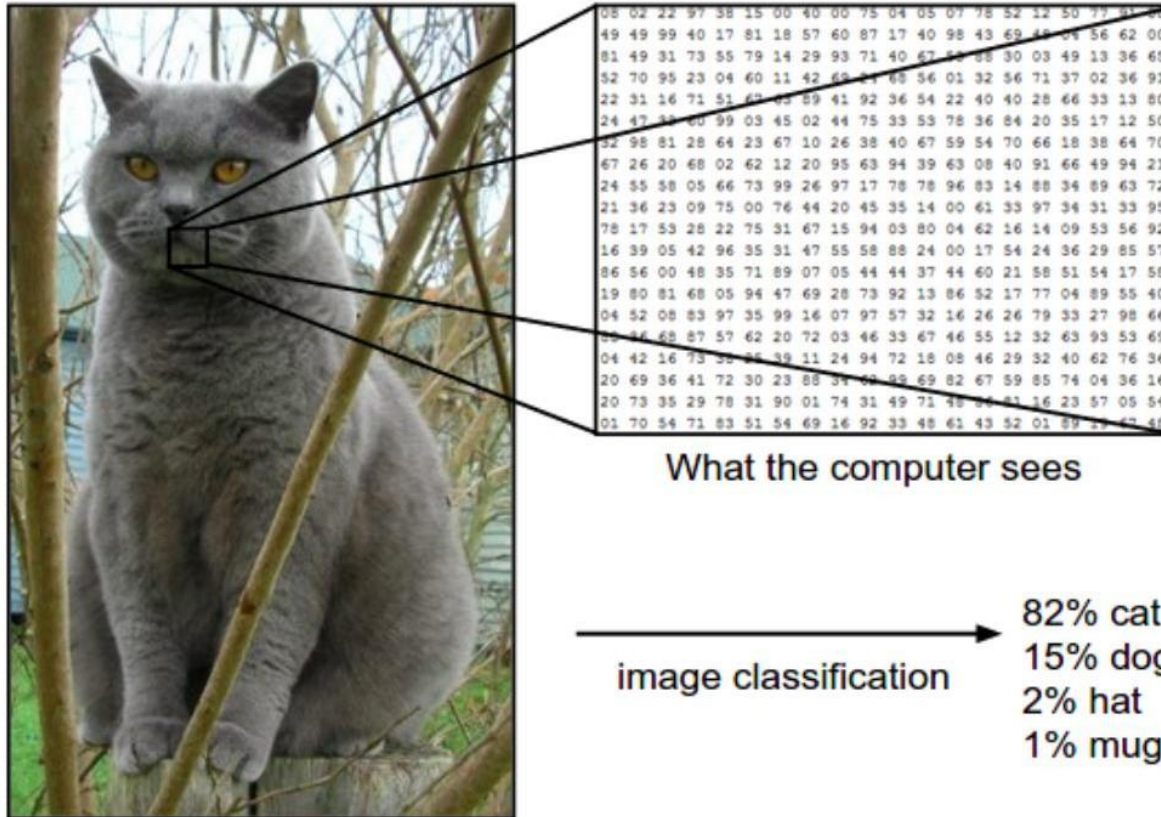
(FC) 가중치 -> (Conv) 필터

(FC) 곱 -> (Conv) 합성곱

Not so different!

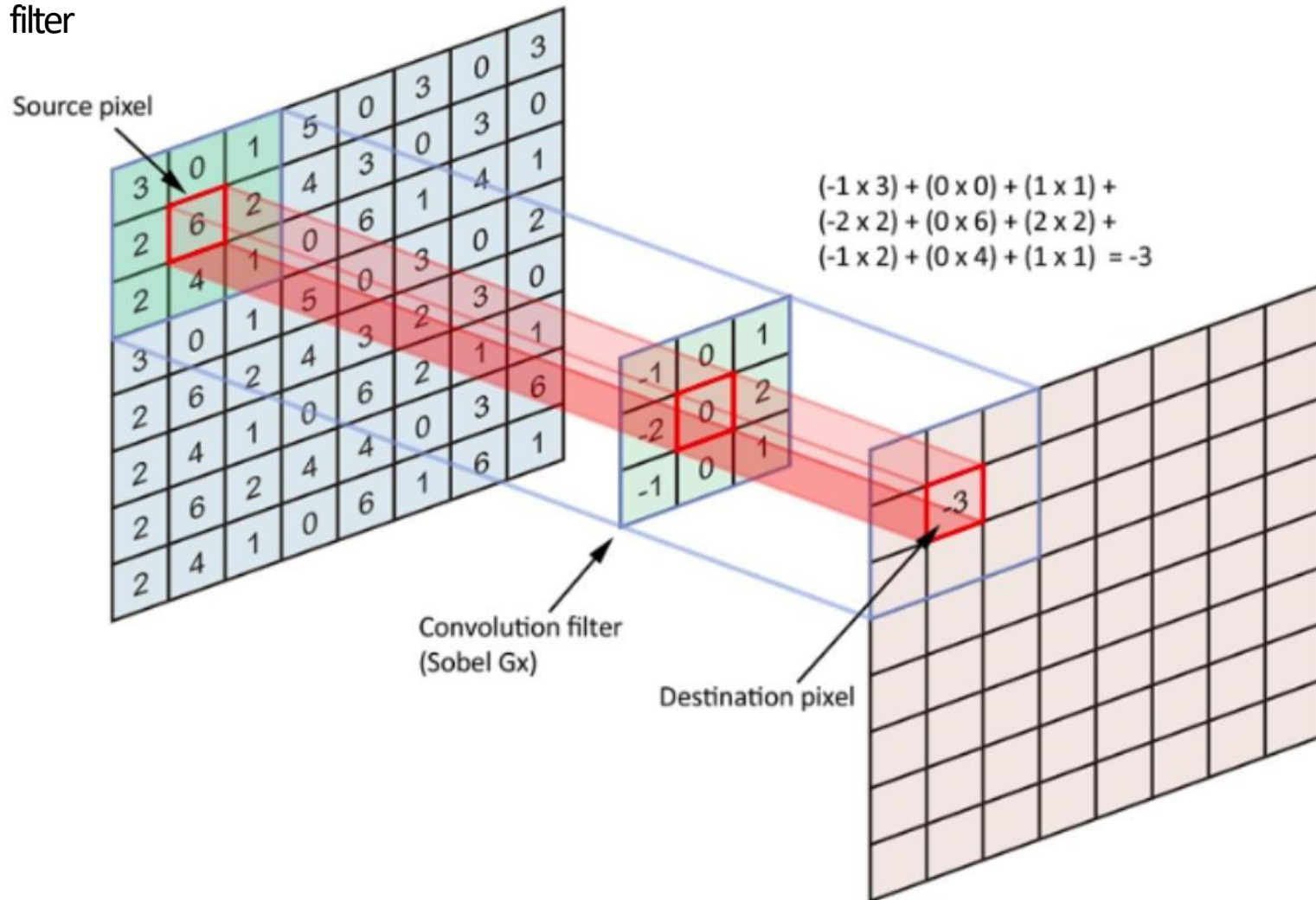
1. What is Convolutional Layer?

(1) input



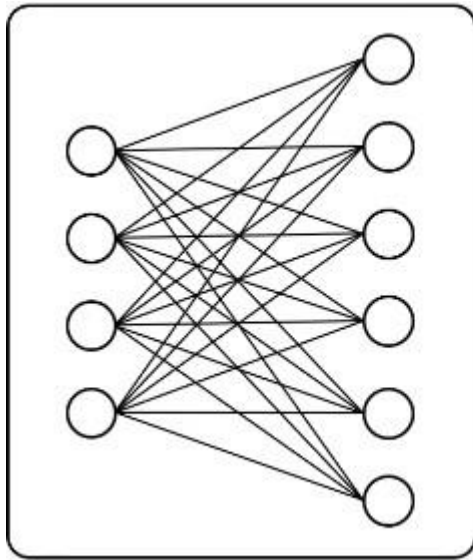
1. What is Convolutional Layer?

(2) filter



2. Convolutional Layer – 수 식 적 표 현

(1) FC의 수 식 적 표 현



$$W = [\mathbf{w}_0, \mathbf{w}_1, \dots, \mathbf{w}_{M-1}]^T$$

$$\mathbf{b} = [b_0, b_1, \dots, b_{M-1}]^T$$

$$y_0 = a(\mathbf{w}_0^T \mathbf{x} + b_0)$$

$$y_1 = a(\mathbf{w}_1^T \mathbf{x} + b_1)$$

$$\vdots$$

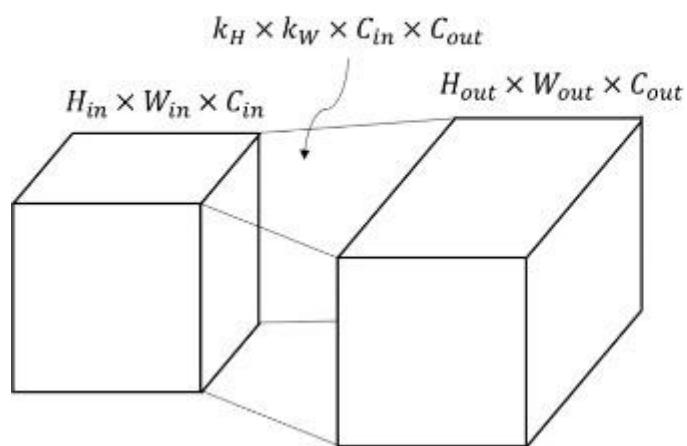
$$y_{M-1} = a(\mathbf{w}_{M-1}^T \mathbf{x} + b_{M-1})$$

$$\mathbf{y} = a(W\mathbf{x} + \mathbf{b})$$

뉴런들이 곱해진 뒤, 모두 더해짐! (Matrix 곱 연산으로 표현)

2. Convolutional Layer – 수 식 적 표 현

(2) Conv의 수 식 적 표 현



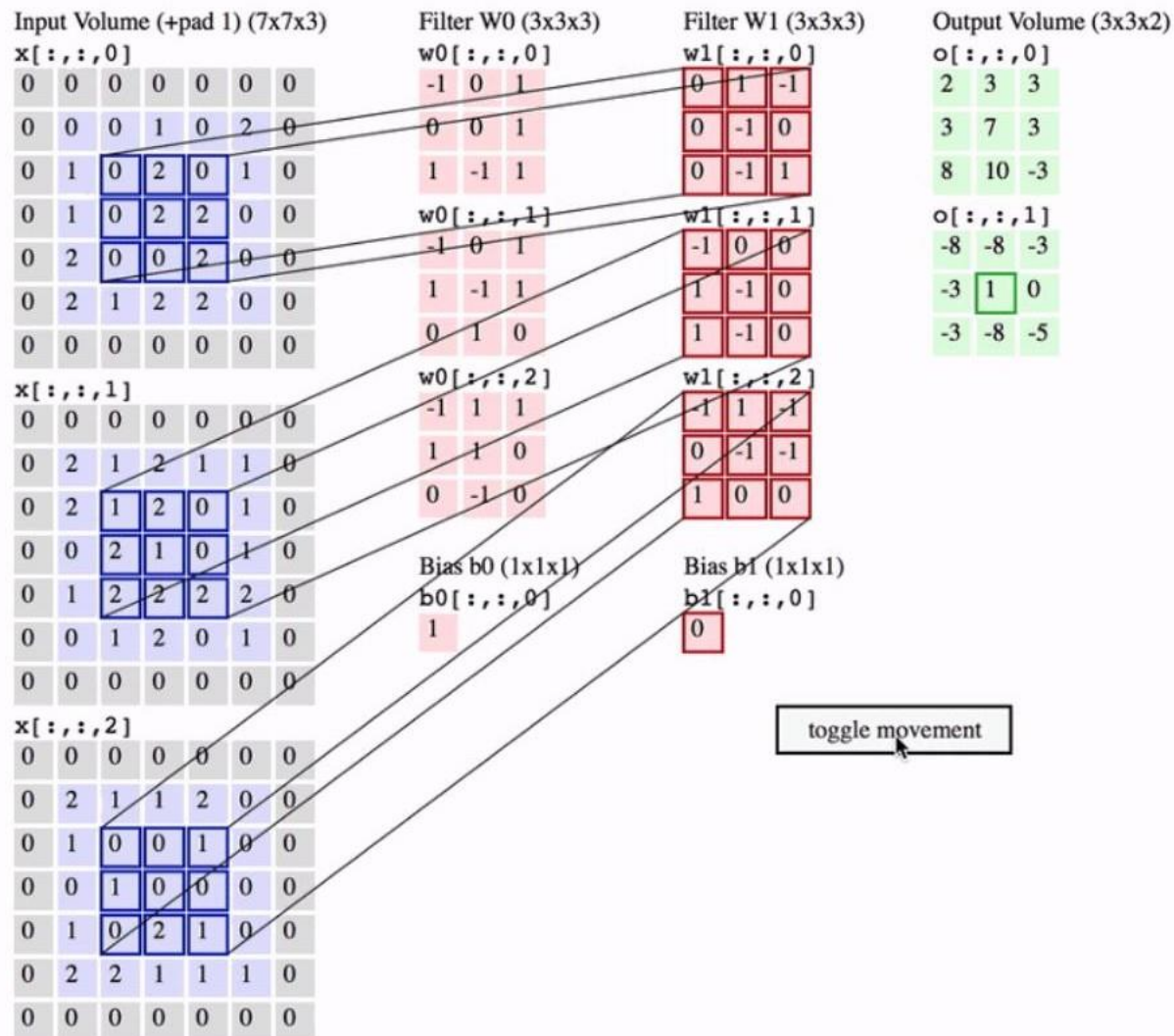
보통 3x3, 5x5, 7x7 등을 사용

$$\mathbb{R}^{k_H \times k_W \times C_{in} \times C_{out}}$$
$$\mathbf{W} = \begin{bmatrix} W_{0,0} & \cdots & W_{0,M-1} \\ \vdots & \ddots & \vdots \\ W_{N-1,0} & \cdots & W_{N-1,M-1} \end{bmatrix}$$
$$\mathbf{b} = [b_0, b_1, \dots, b_{M-1}]^T$$

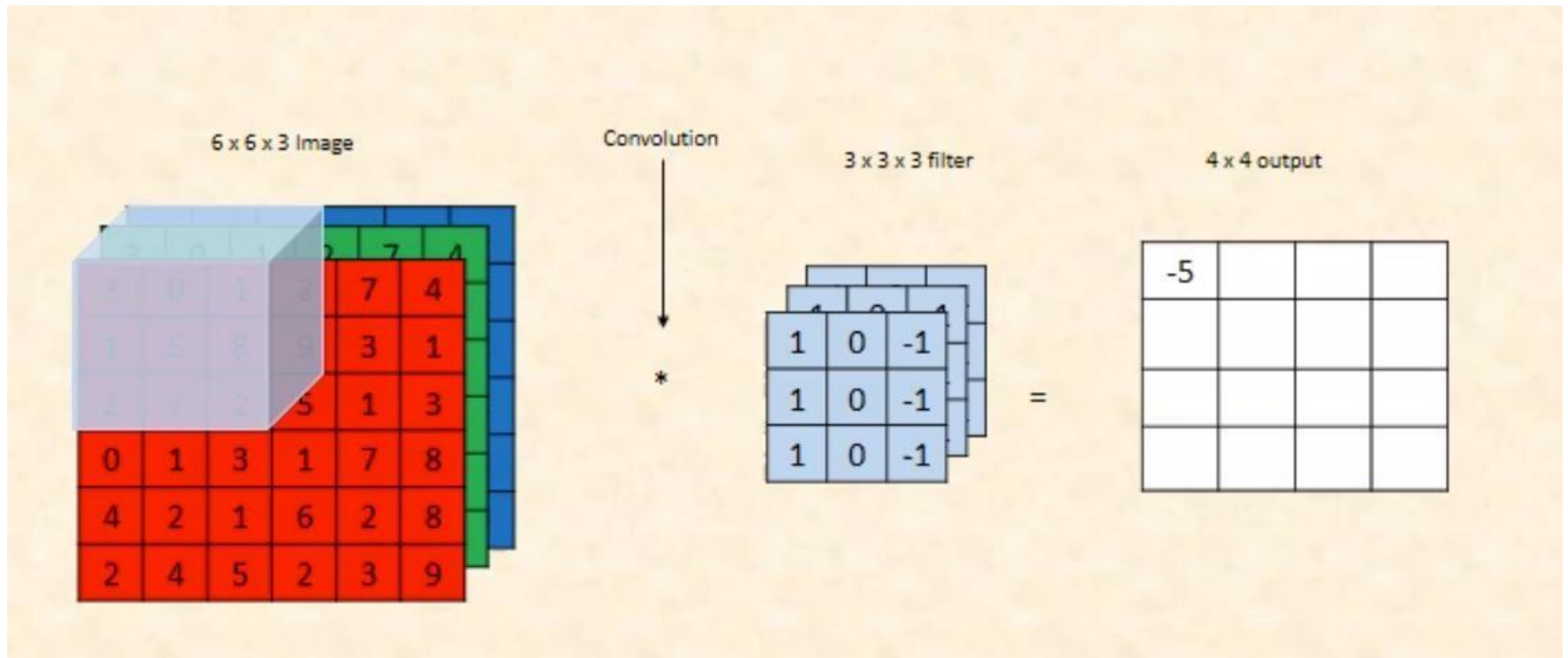
$$Y_{i,j} = a(W_{i,j} * X_i + b_j)$$

C(in) x C(out) 번의 합성곱 연산이 이루어짐!

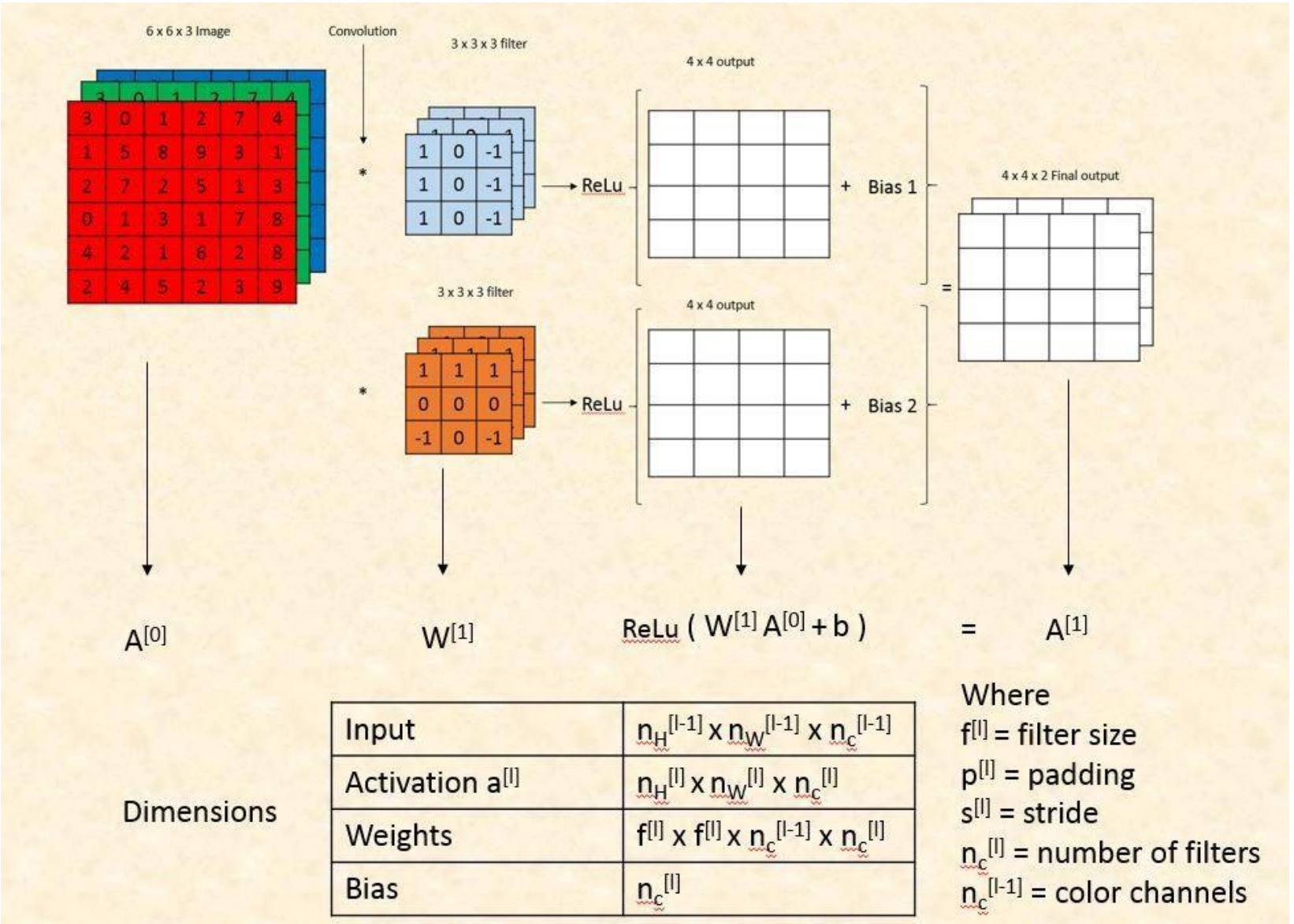
2. Convolutional Layer – 수식적 표현



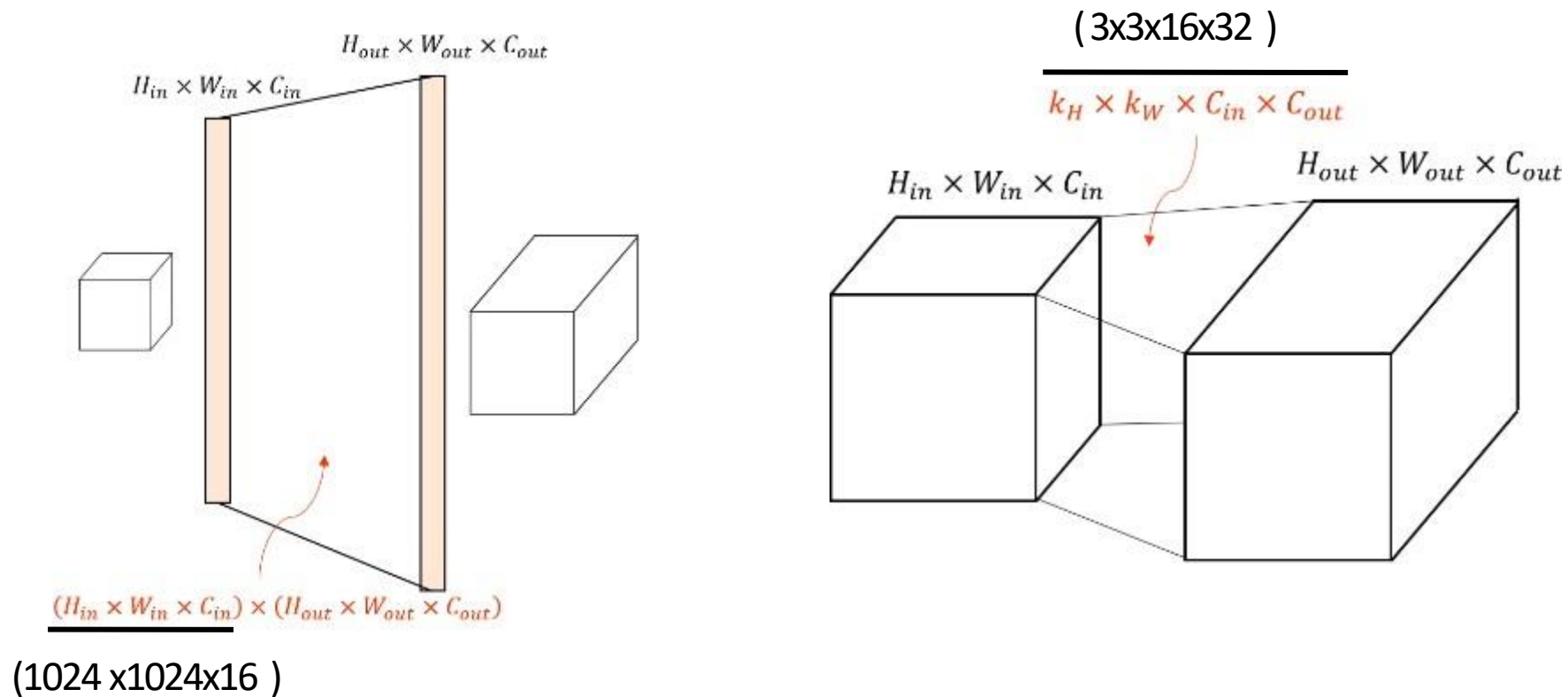
2. Convolutional Layer – 수 식 적 표 현



2. Convolutional Layer – 수 식 적 표 현



3. Why Convolutional Layer?



FC 대신 Convolutional Layer 써야하는 이유?

“Parameter 수의 감소!”

4. 용 어 설 명

(1) Channel

RED Channel



Green Channel



Blue Channel

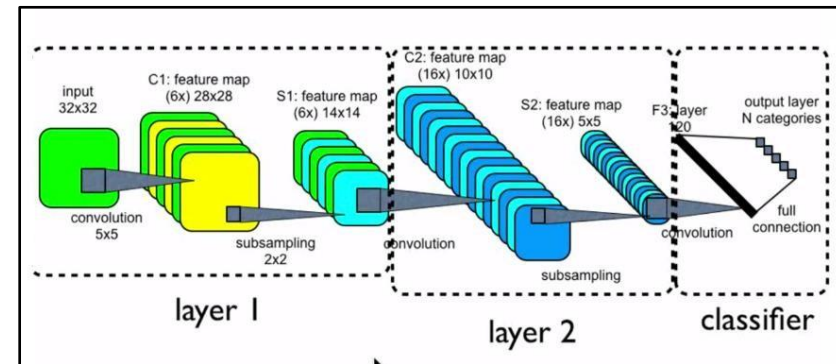
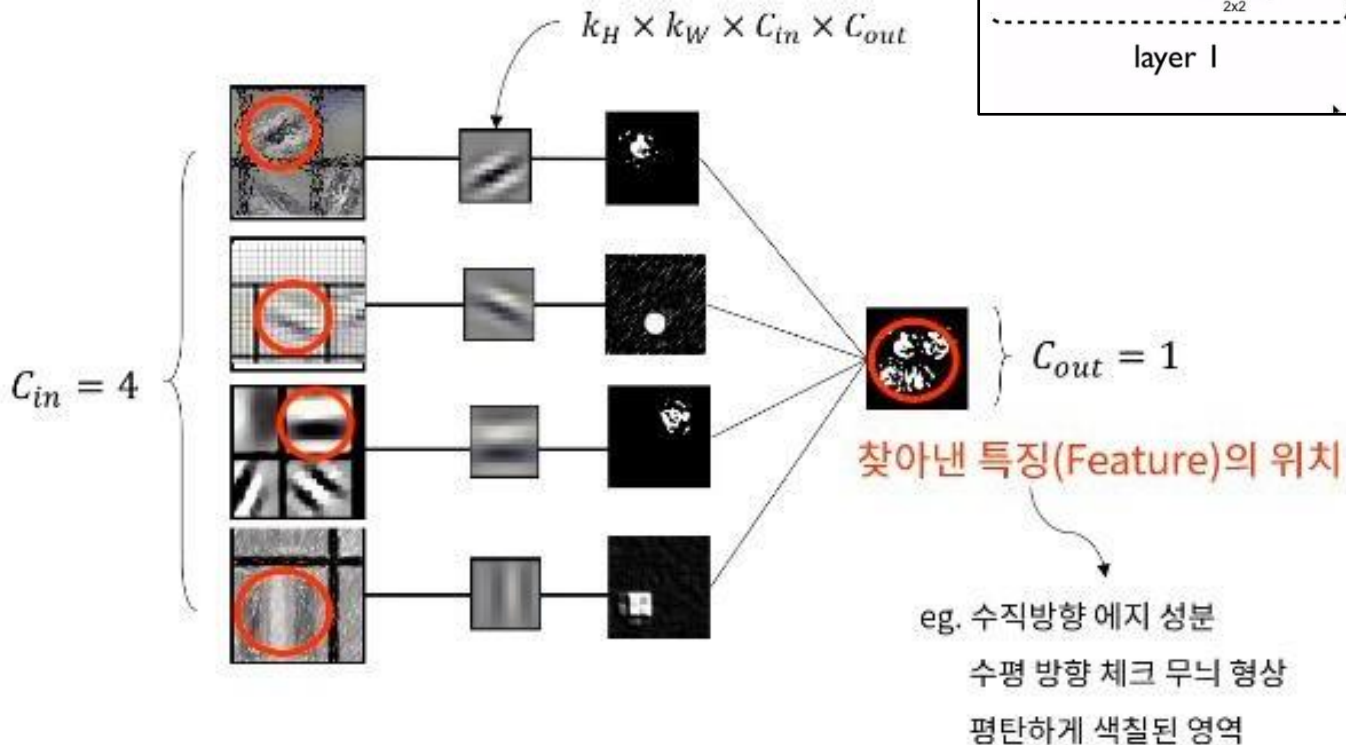


이미지 출처: [https://en.wikipedia.org/wiki/Channel_\(digital_image\)](https://en.wikipedia.org/wiki/Channel_(digital_image))

4. 용 어 설 명

(2) Feature Map

합성곱을 통해서 만들어진 출력!

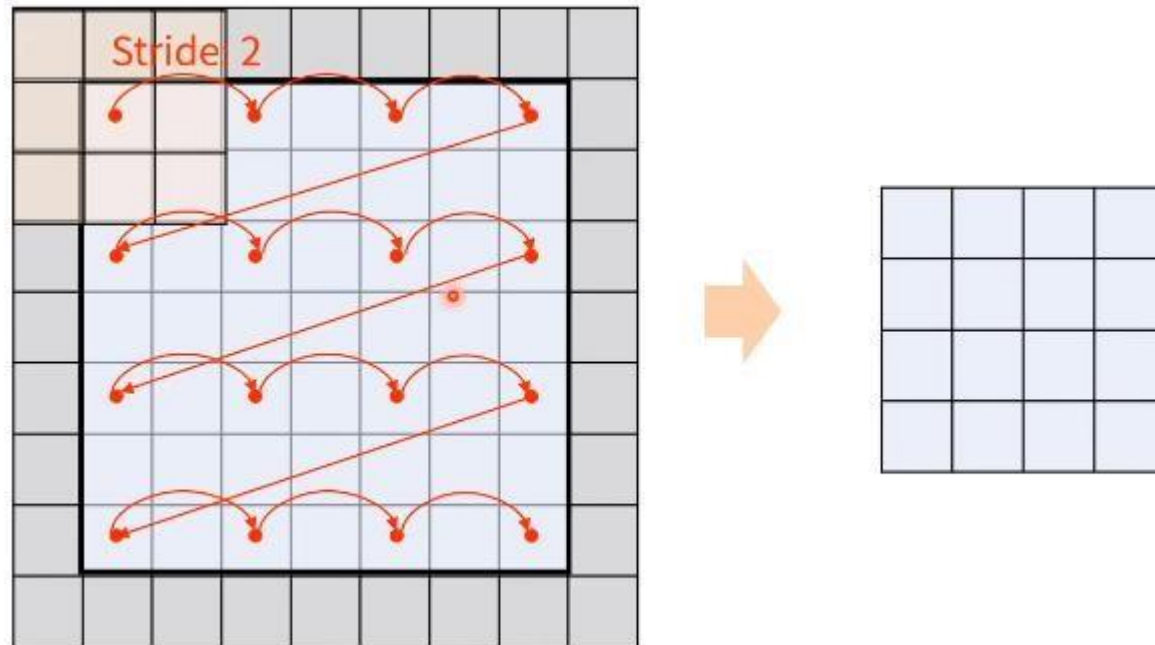


합성곱 계층의 의미?

여러 채널에서 특별한 “특징”이 나타내는 위치를 찾아내는 역할!

4. 용 어 설 명

(3) Stride



합성곱 계산시, 커널을 이동시키는 거리!

Stride를 크게 할 수록, 영상/사진의 크기는 줄어든다!

4. 용 어 설 명

(4) Padding

0	0	0	0	0	0
0	35	19	25	6	0
0	13	22	16	53	0
0	4	3	7	10	0
0	9	8	1	3	0
0	0	0	0	0	0

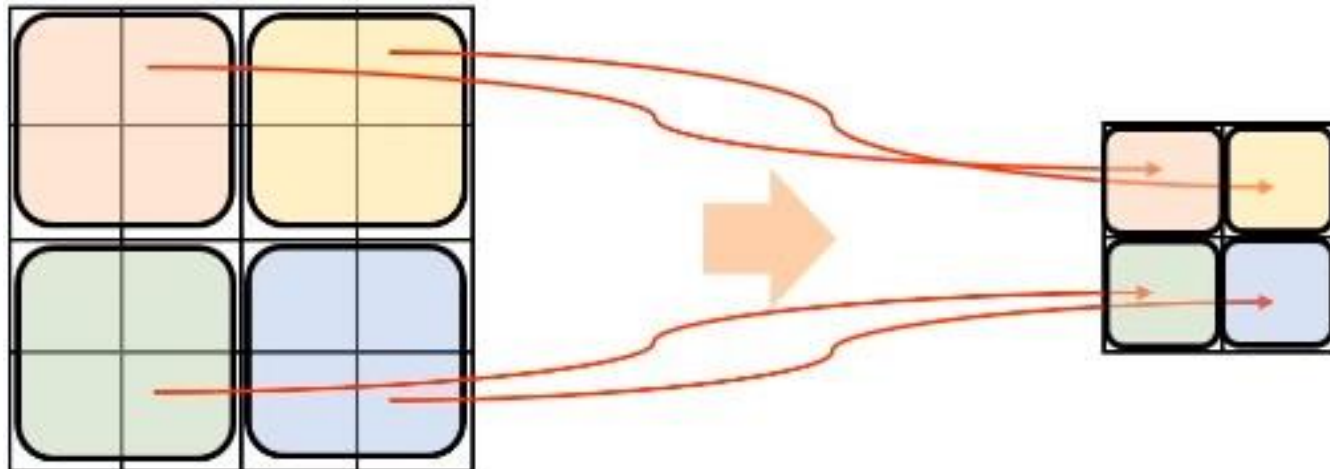
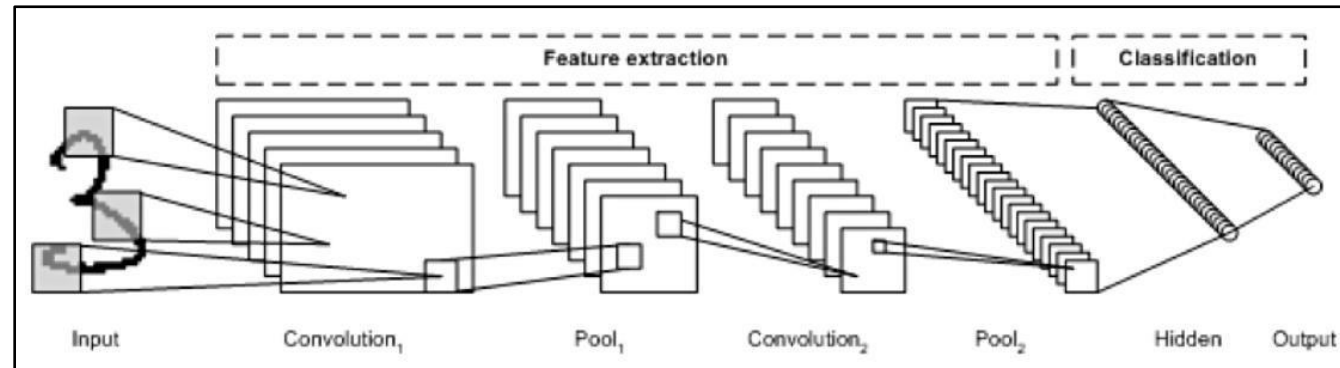
주위에 숫자(주로 0)을 둘러싸우는 것! (For 크기 보존)

합성 곱 연산시, 필터(커널)의 크기에 따라 영상의 크기가 줄어드는 문제 발생

ex) 크기가 $(2N+1)$ 인 커널 -> 상하좌우로 N 개의 zero-padding!

4. 용 어 설 명

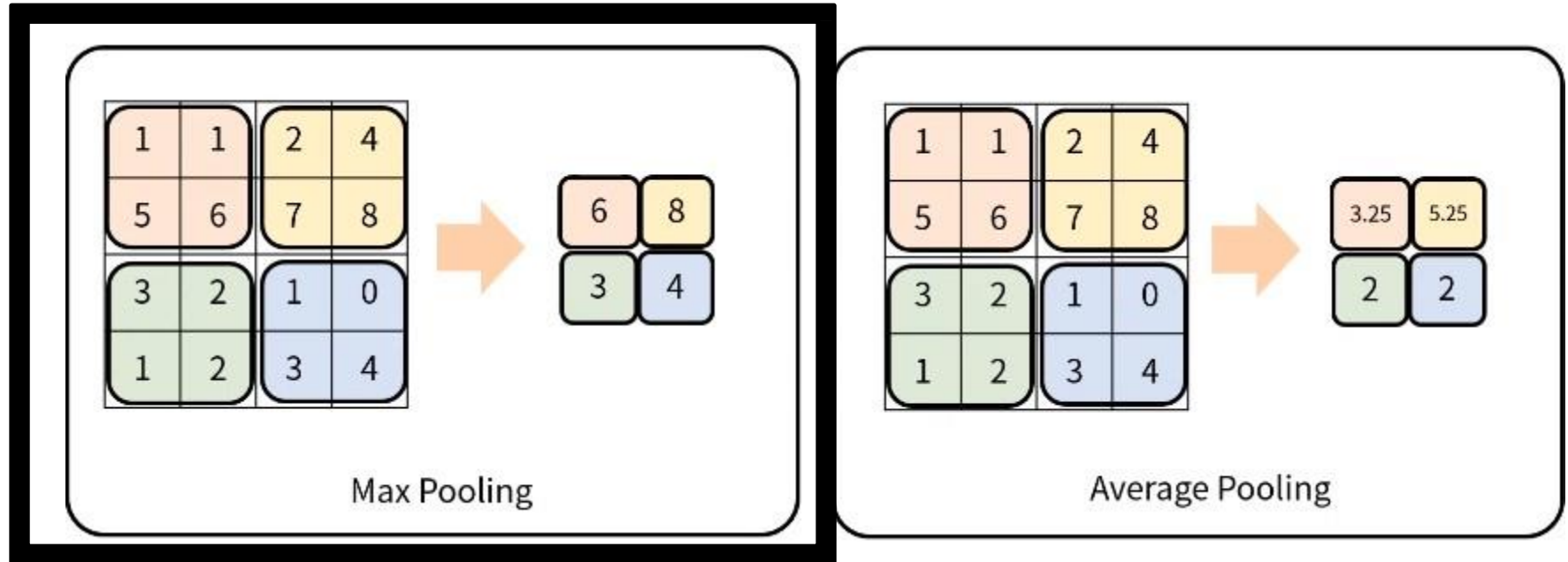
(5) Pooling



Sampling이라고 보면 됨! (여러 화소를 하나의 화소로 축약!)
즉, “영상/사진의 크기가 줄어들고, 정보가 종합(요약)된다!”

4. 용 어 설 명

(5) Pooling



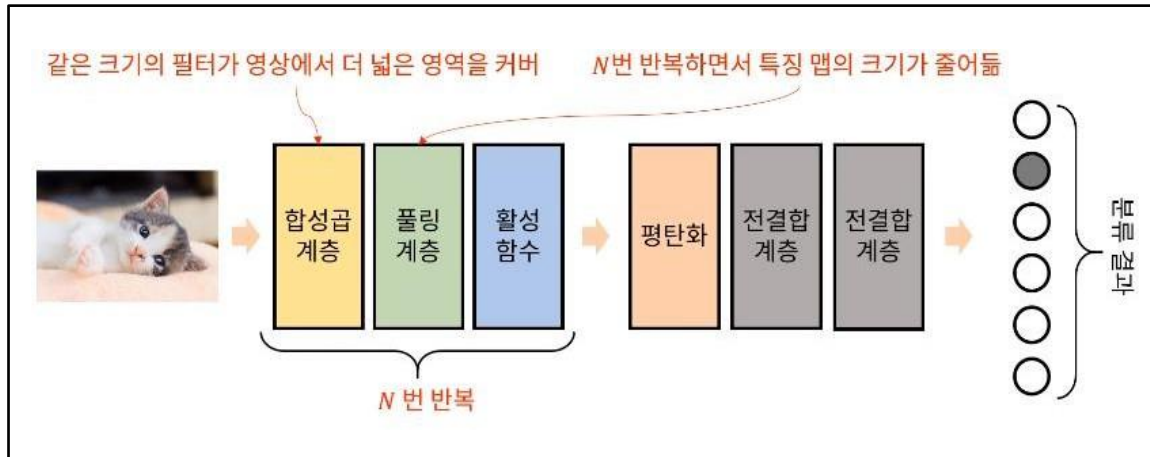
Pooling Layer (vs Convolution Layer)

- 1) 학습대상 parameter 없음
- 2) Pooling Layer를 통과하면 행렬의 크기 감소
- 3) Pooling Layer를 통해서 채널 수 변경 없음

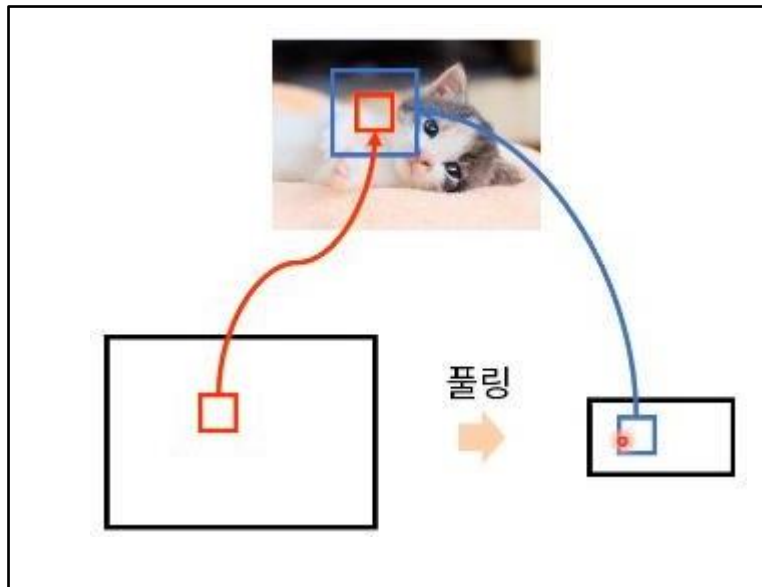
4. 용어 설명

(5) Pooling

why pooling? Receptive Field?



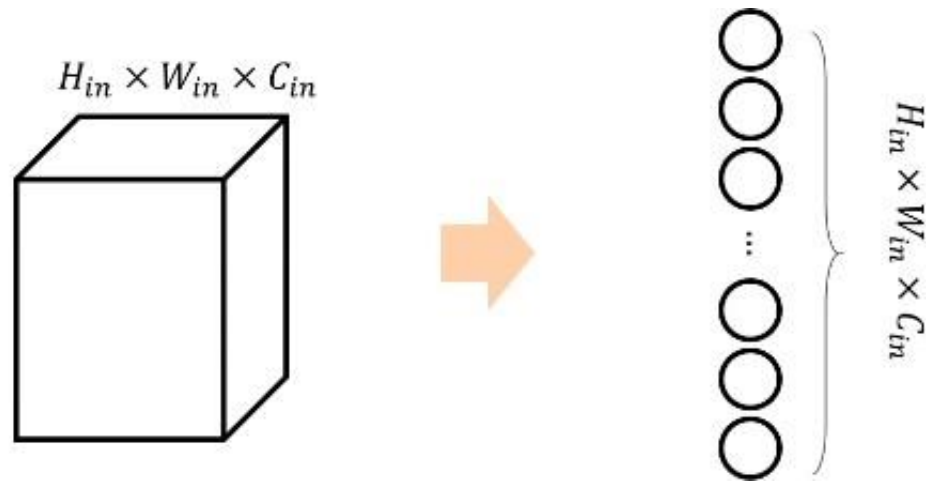
(h,w)영상 크기 작아지고,
(c)채널 수 늘어나게 될 것!



같은 크기의 filter여도, Pooling에 의해
작아진 feature map에 적용되면, “원본 영상에서
차지하는 범위(=receptive field)”가 넓다

4. 용 어 설 명

(6) Flatten



평탄화! -> Vector로 만들어줌

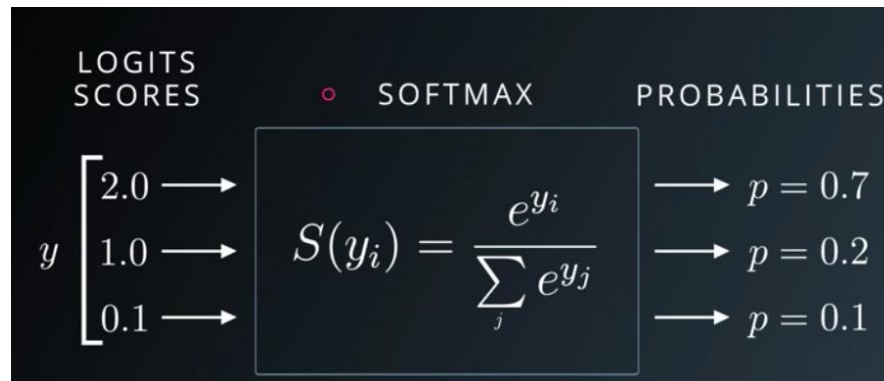
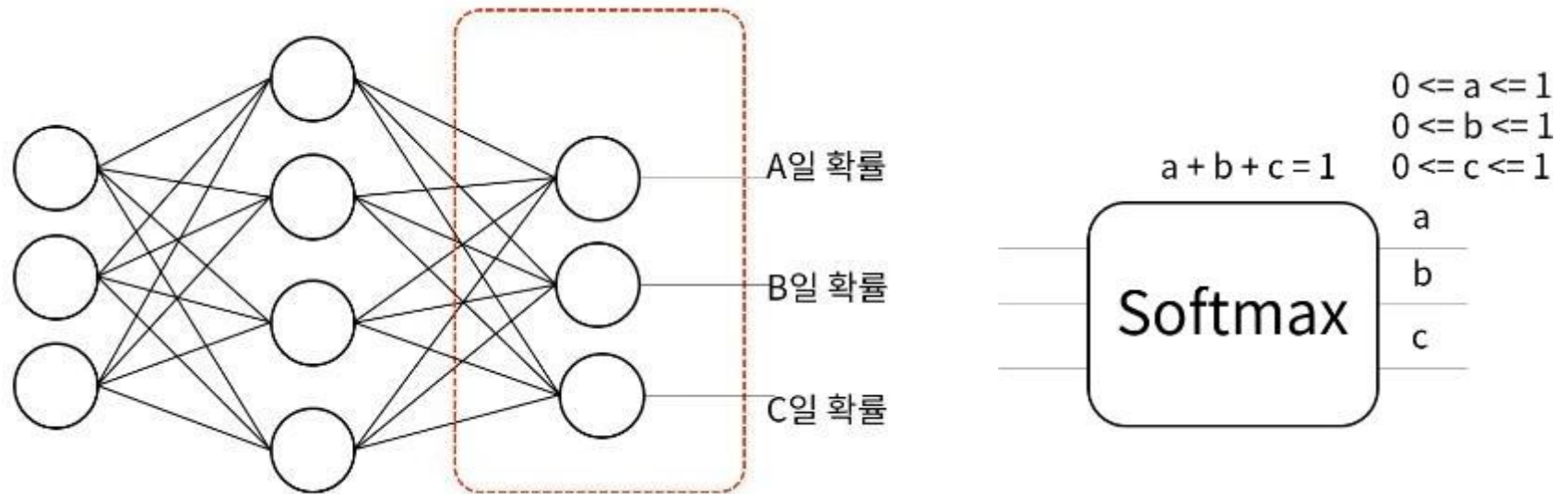
단지 쪽 피는 것을 의미! (연산 과정 x)

Conv & FC 연결해주는 역할

4. 용 어 설 명

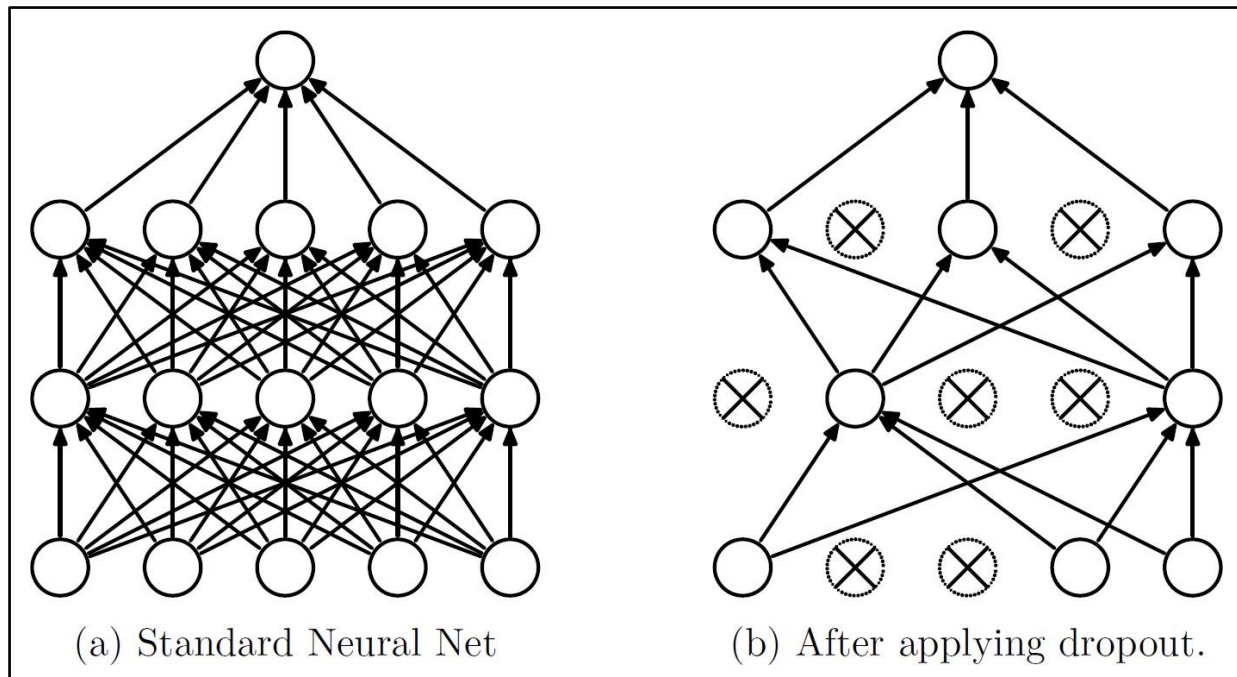
(7) SoftmaxFunction

for Multi-class Classification



5. Dropout

To prevent overfitting!



Training 할 때 만!
(testing 할 때는 X)

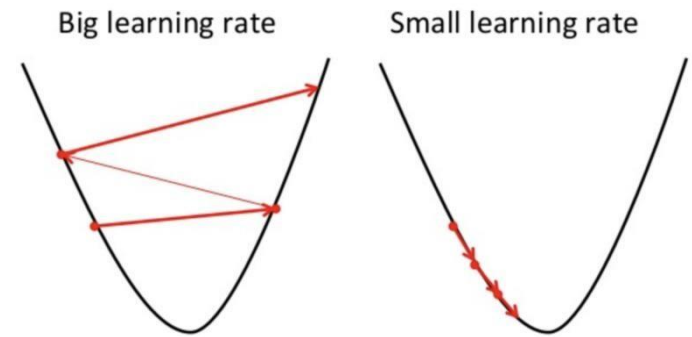
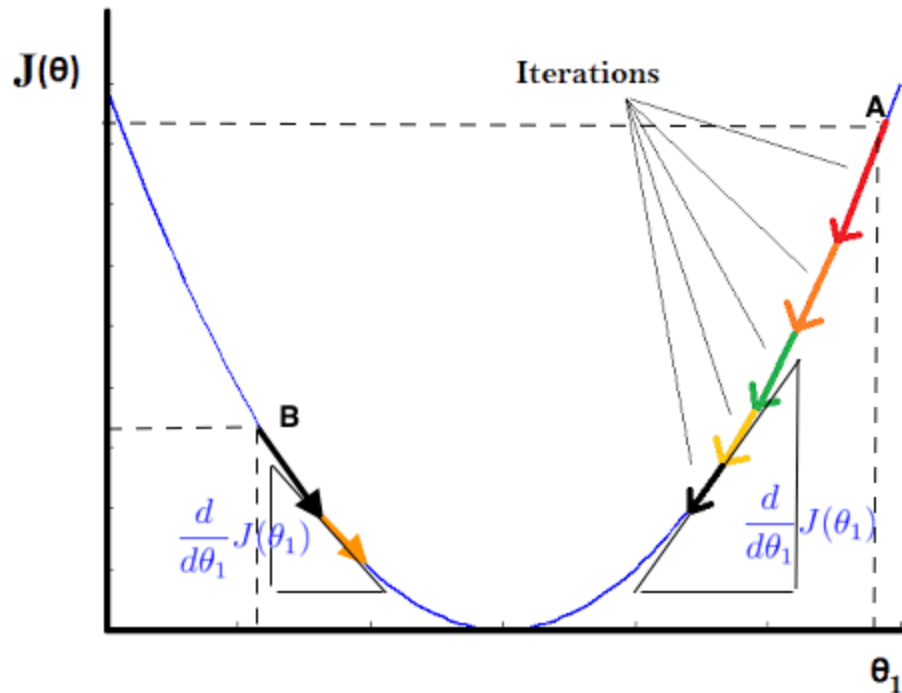
Ex) Dropout rate 0.3 : 30%의 node는 사용 X (less parameter)

Overfitting 문제가 해결되는 이유를, 직관적으로 생각해 보자!

100가지의 일, 100명의 사람이 하는 경우? 20명의 사람이 하는 경우?

6. Mini-Batch

[복습] Gradient Descent



Optimal point를 찾아가는 과정! (too big, too small LR (X))

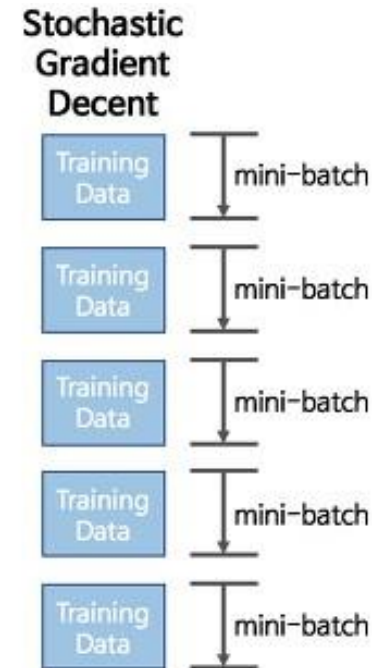
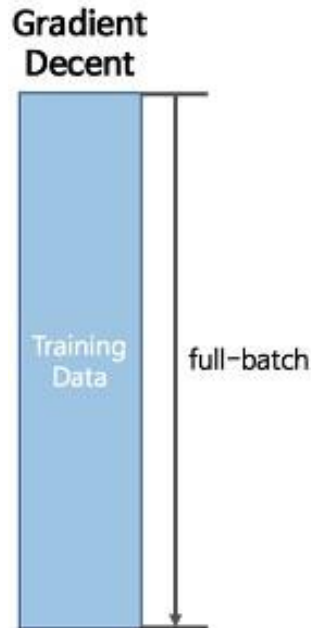
1 Iteration : dataset이 한번 반복되는 것을 의미!
(iteration 한번 할 때마다 parameter 조정해 나감)

6. Mini-Batch

Vanilla(Batch) Gradient Descent (일반 경사 하강법)

Gradient를 한번 update하기 위해, “

모든 training data”를 사용



Stochastic Gradient Descent (확률적 경사 하강법)

Gradient를 한번 update하기 위해,

“일부의 training data”만 활용! (일단, 일부 = 1개라고 일단은 생각하자!)

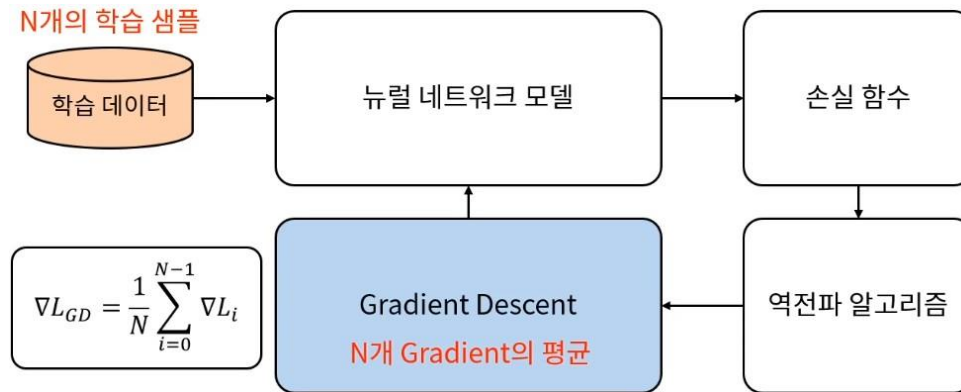
(일부=m개인 경우가 „Mini-Batch Gradient Descent“)

(SGD와 Mini-batch GD의 용어를 섞어 쓰기도)

6. Mini-Batch

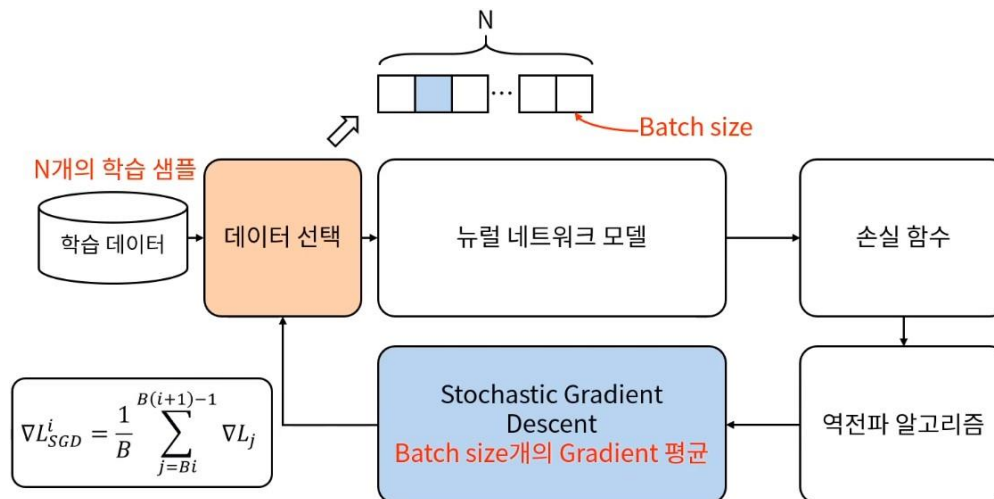
Vanilla(Batch) Gradient Descent

(일반 경사 하강법)



Stochastic Gradient Descent

(확률적 경사 하강법)



6. Mini-Batch

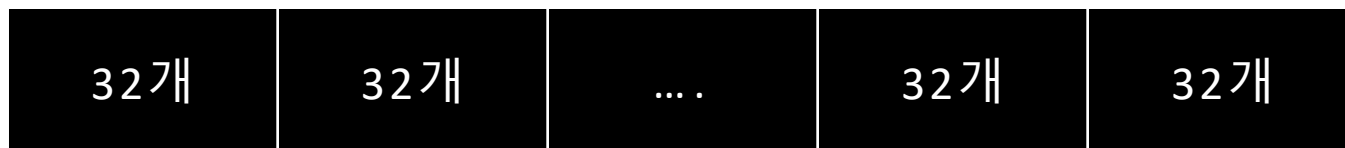
	Vanilla Gradient Descent (일반 경사 하강법)	Stochastic Gradient Descent (확률적 경사 하강법)
장점	<ul style="list-style-type: none">- 전체 data에 대해 update가 “1번” 이루어짐-> SGD에 비해 적은 update(계산) 횟수 필요- 전체 data에 대해 errorgradient 계산! (optimal로 수렴이 안정적)- 병렬 처리 GOOD	<ul style="list-style-type: none">- Local optimal에 빠질 위험 적음- step에 걸리는 시각이 짧기 때문에 학습- 속도(수렴 속도)가 빠름
단점	<ul style="list-style-type: none">- 학습이 오래 걸림! (한 스텝에 “전체 data”를 활용하여 학습하므로)- Local optimal에 빠질 위험	<ul style="list-style-type: none">- Global Optimal를 찾기 어려움- 병렬 처리 BAD

6. Mini-Batch

(Vanilla) Gradient Descent (GD)

TRAINING DATASET (1000개)

Mini-Batch Gradient Descent(M B G D)



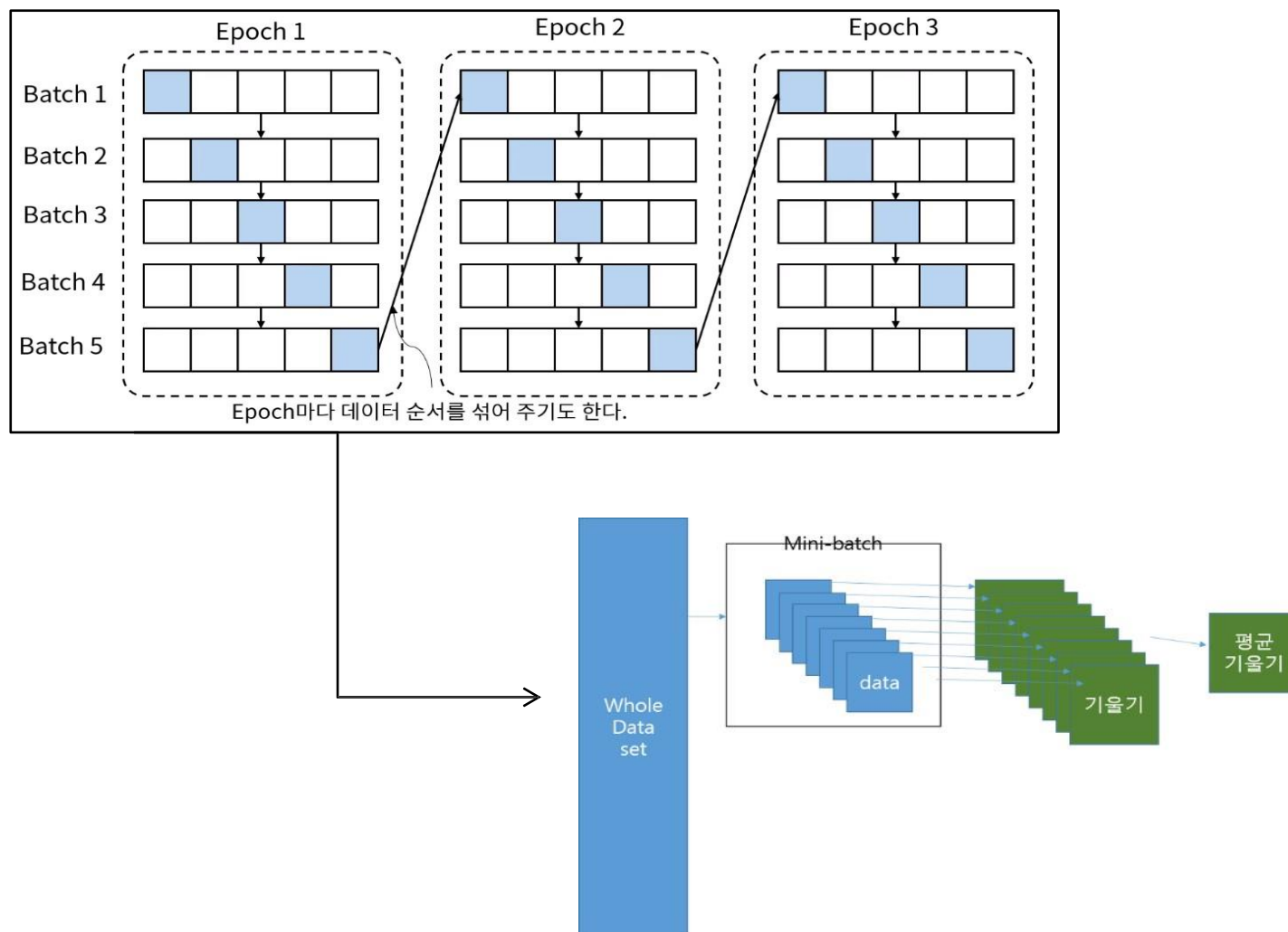
(용어 구분해서 쓰지 않는 경우도!)

Stochastic Gradient Descent (SGD)



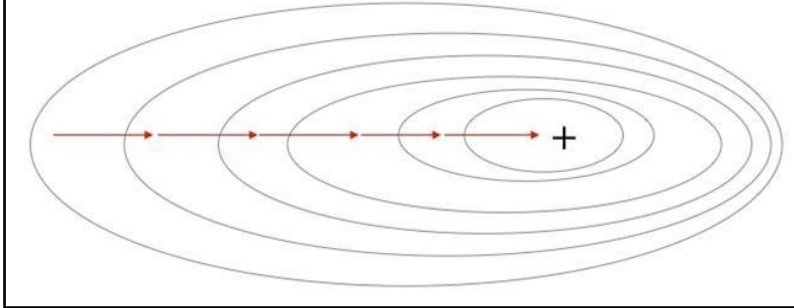
6. Mini-Batch

Training data 전체를 한 번 학습 하는 것을 “Epoch” 한
번 Gradient를 구하는 단위를 “Batch”라고 한다

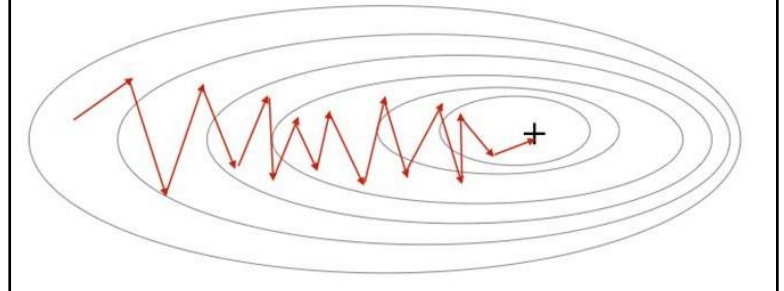


6. Mini-Batch

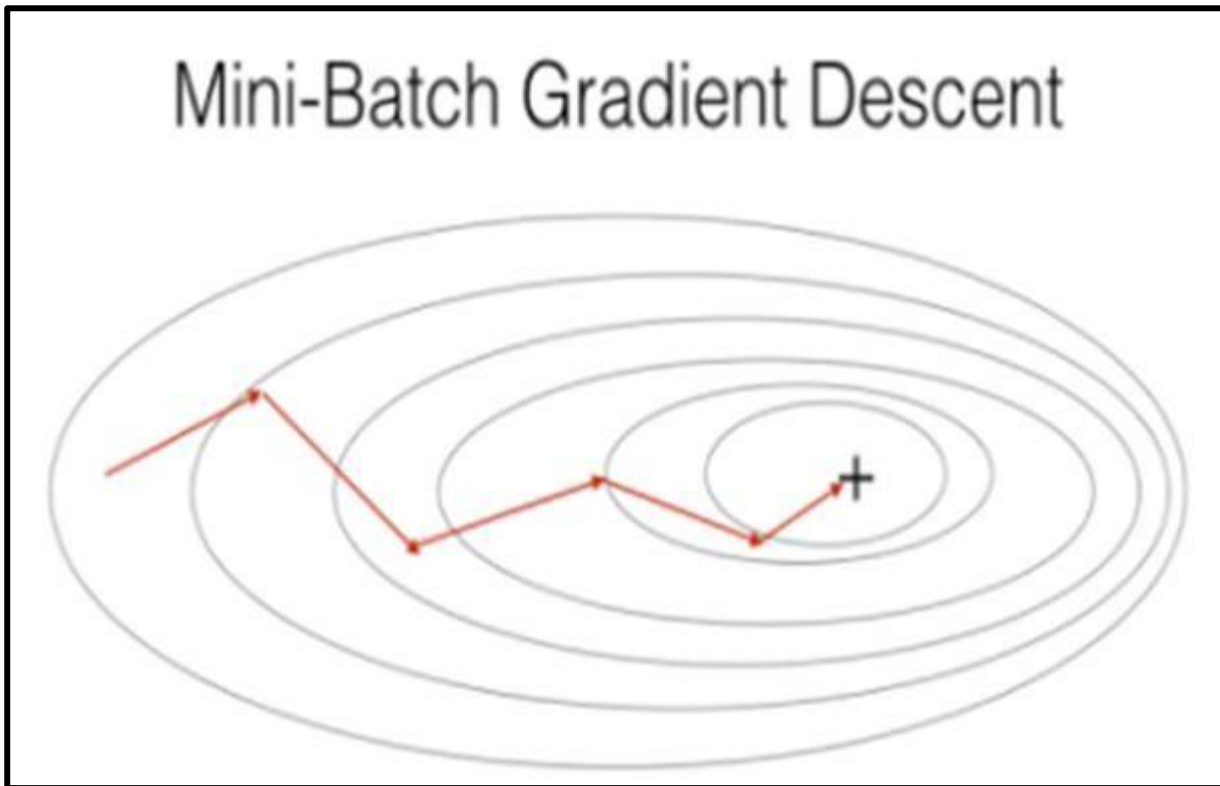
Gradient Descent



Stochastic Gradient Descent



Mini-Batch Gradient Descent

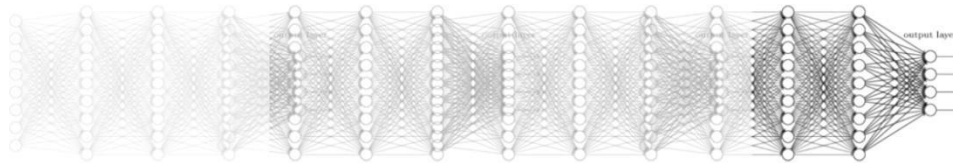


7. Batch Normalization

Gradient Vanishing?

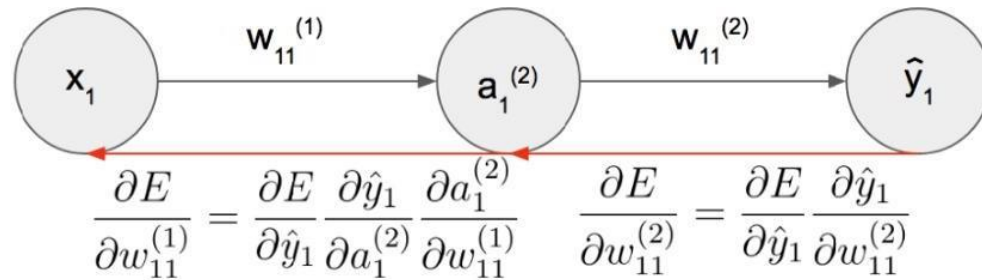
(기울기 소실 문제)

Vanishing gradient (NN winter2: 1986-2006)



DEEP 할 수록 gradient vanishing! WHY?

(Backpropagation의 수식을 생각해 보자)

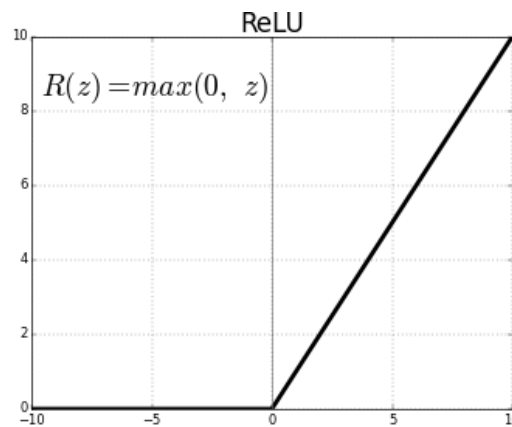
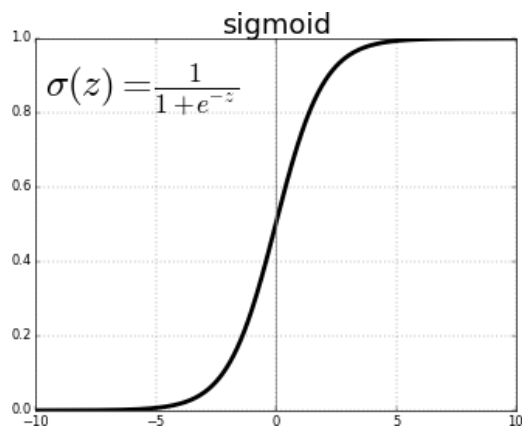


7. BatchNormalization

Gradient Vanishing?

How to Solve?

1) Activation Function으로 Sigmoid -> ReLU

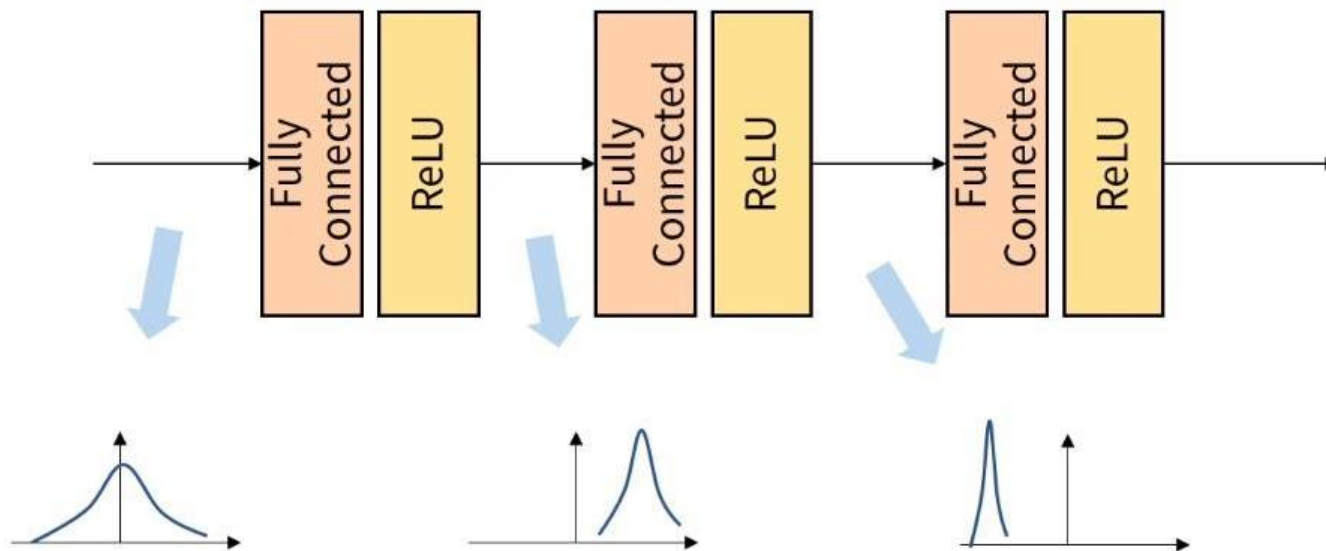


2) 적절한 weight 초기값 : Xavier Initialization, He Initialization

3) Batch Normalization

7. Batch Normalization

Internal Covariance Shift



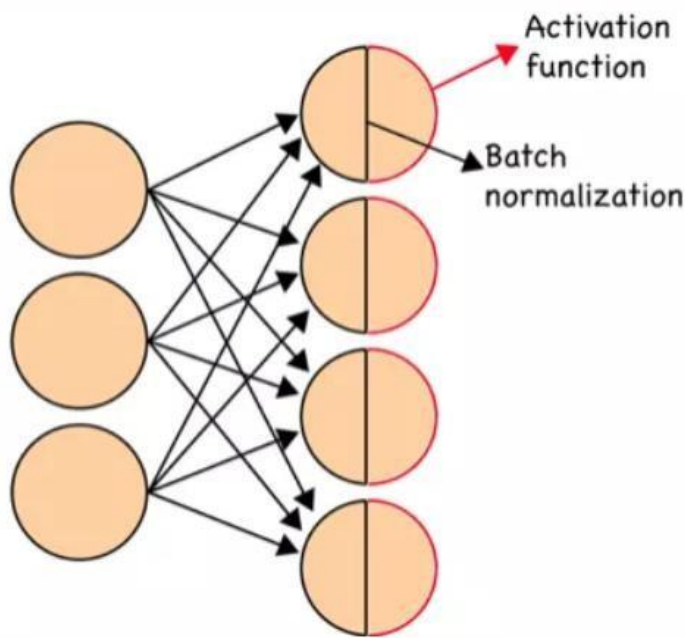
Training 과정에서, parameter의 변화로 인해 **input data의 distribution이 달라지는** 현상! -> 불안정한 training야기!

이 문제를 해결하는 방법이 **Batch Normalization**!

이로써 안정적인 training을 가능하게 하고, gradient vanishing problem 해결!

7. Batch Normalization

Batch Normalization



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots x_m\}$;

Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{ mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{ scale and shift}$$

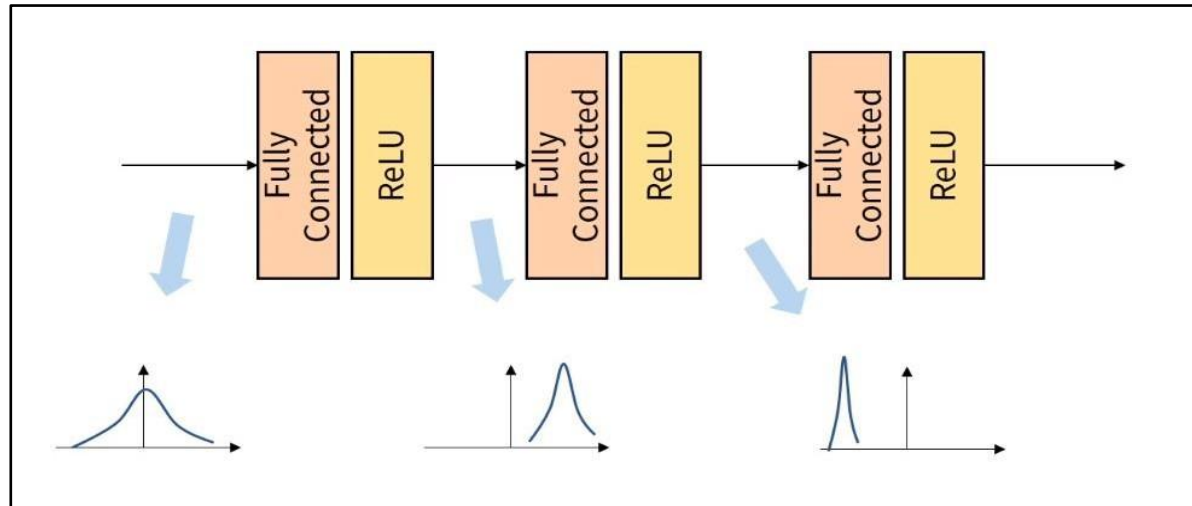
training data 전체에 대하여 normalize하는 것이 좋겠지만, Mini-batch Gradient Descent 방식을 사용하게 되면, parameter의 update가 Mini-batch단위로 이루어지기 때문에, Mini-batch를 단위로 Batch Normalization(BN) 실시!

Internal Covariance Shift 문제를 해결하기 위해, “**각층의 input의 distribution을 평균 0, 표준편차 1인 input으로 normalize**”시키는 방법

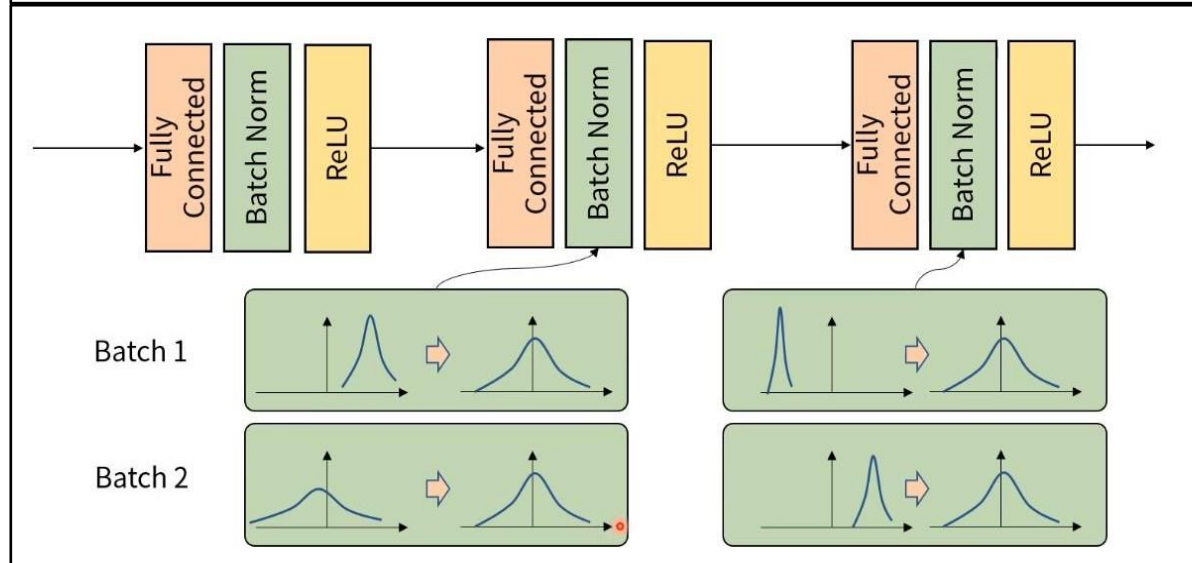
7. Batch Normalization

Internal Covariance Shift

B N (X)



B N (O)



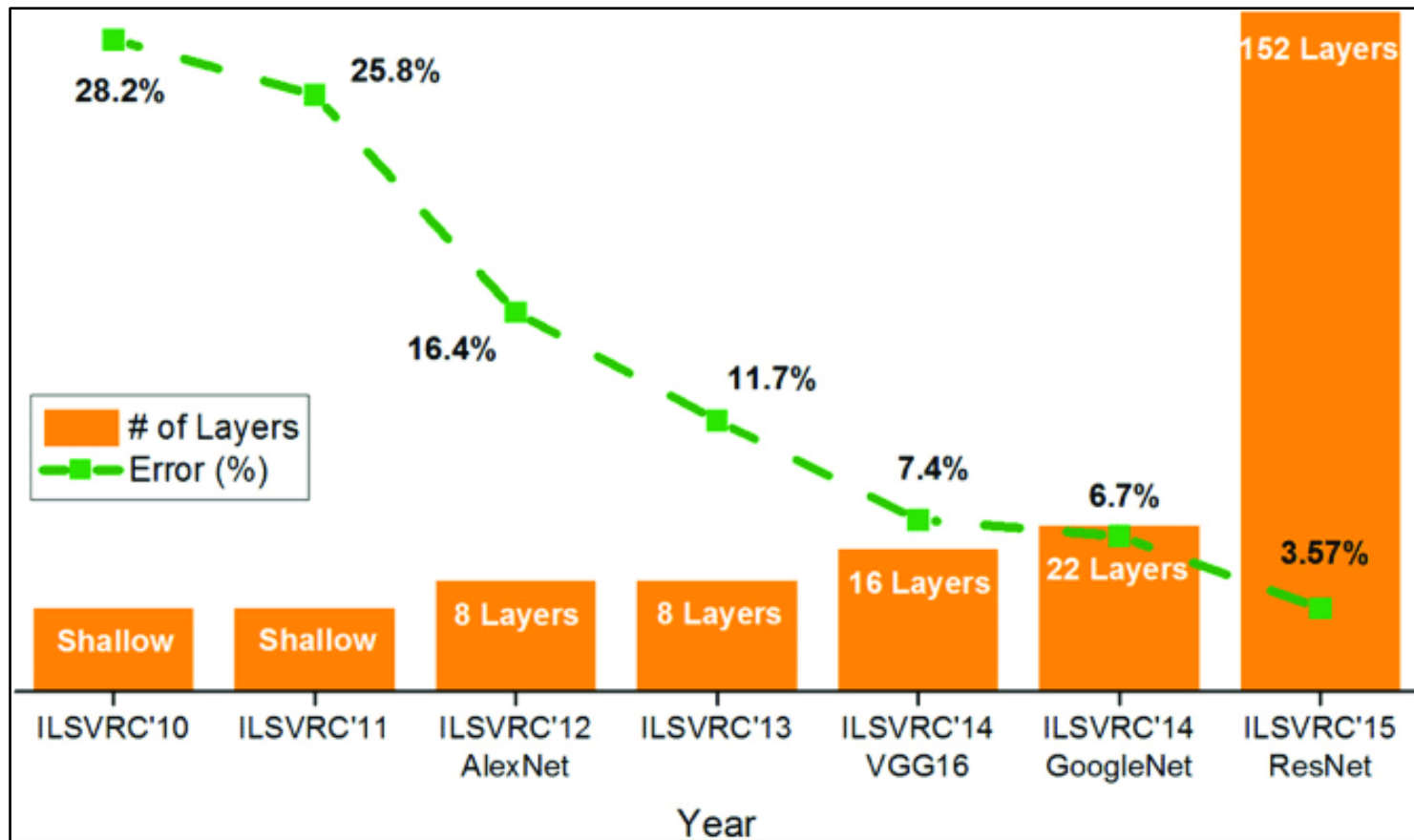
7. Batch Normalization

Batch Normalization

< BN의 장점 >

- 1) **Gradient vanishing** 문제 해결!
- 2) **Regularization** 효과가 있음 (overfitting 방지)
 - (regularization 효과를 주는) Dropout을 제외할 수 있게 해줌
 - Dropout 경우 효과는 좋지만 학습 속도가 느려진다는 단점!
- 3) Gradient의 **scale 영향을 덜 받음!**

7. 심화 CNN



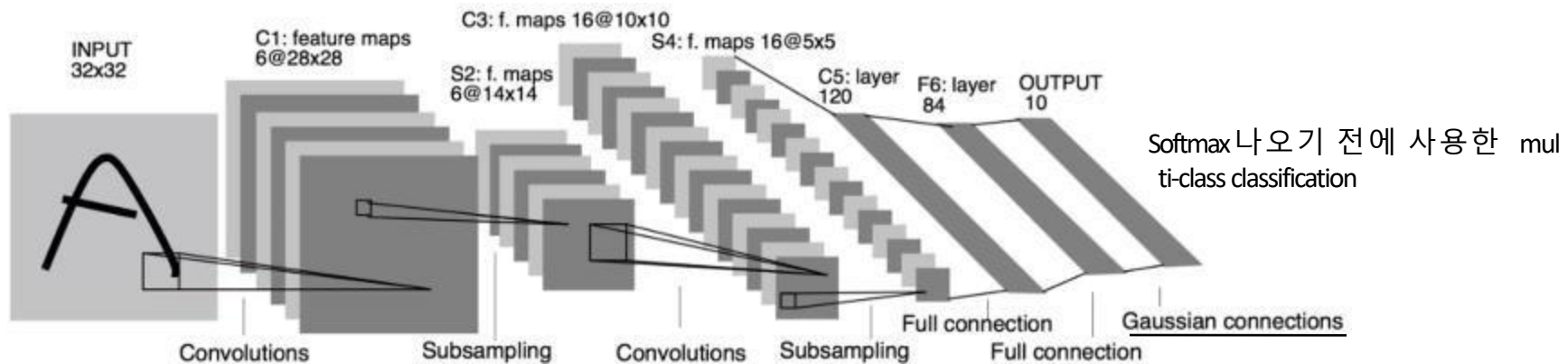
CNN의 대표 모델들에 대해 알아보자.

LeNet-5, AlexNet, VGG16, GoogLeNet, ResNet

7. 심화 CNN

1. LeNet-5

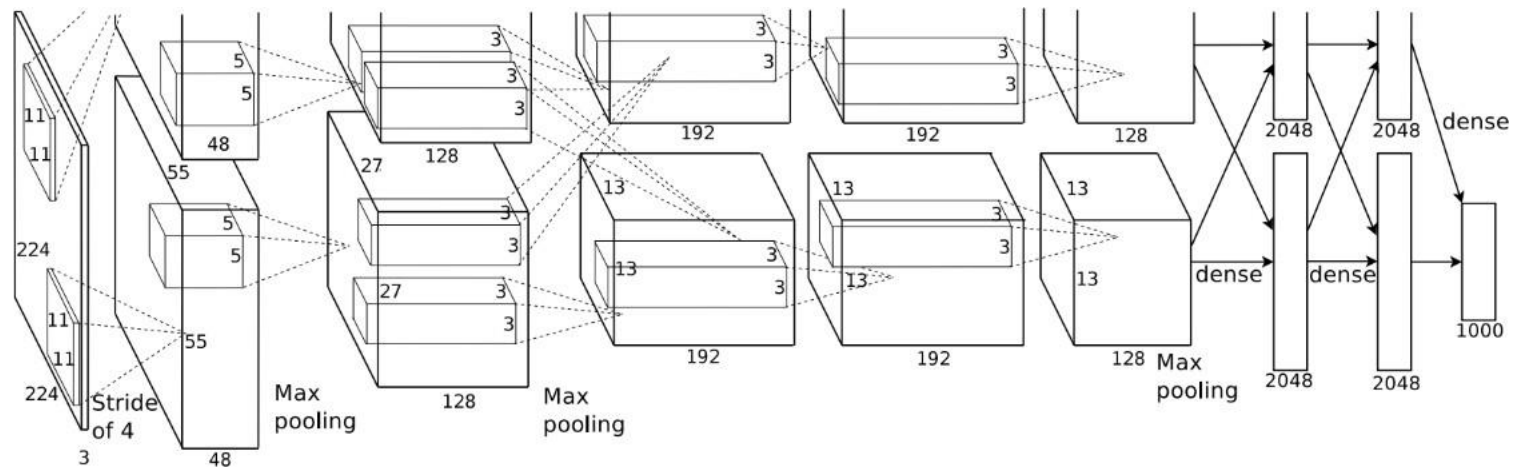
6채널의 28*28 feature map



- 1998. CNN의 기본구조 창립
- MNIST data 사용

7. 심화 CNN

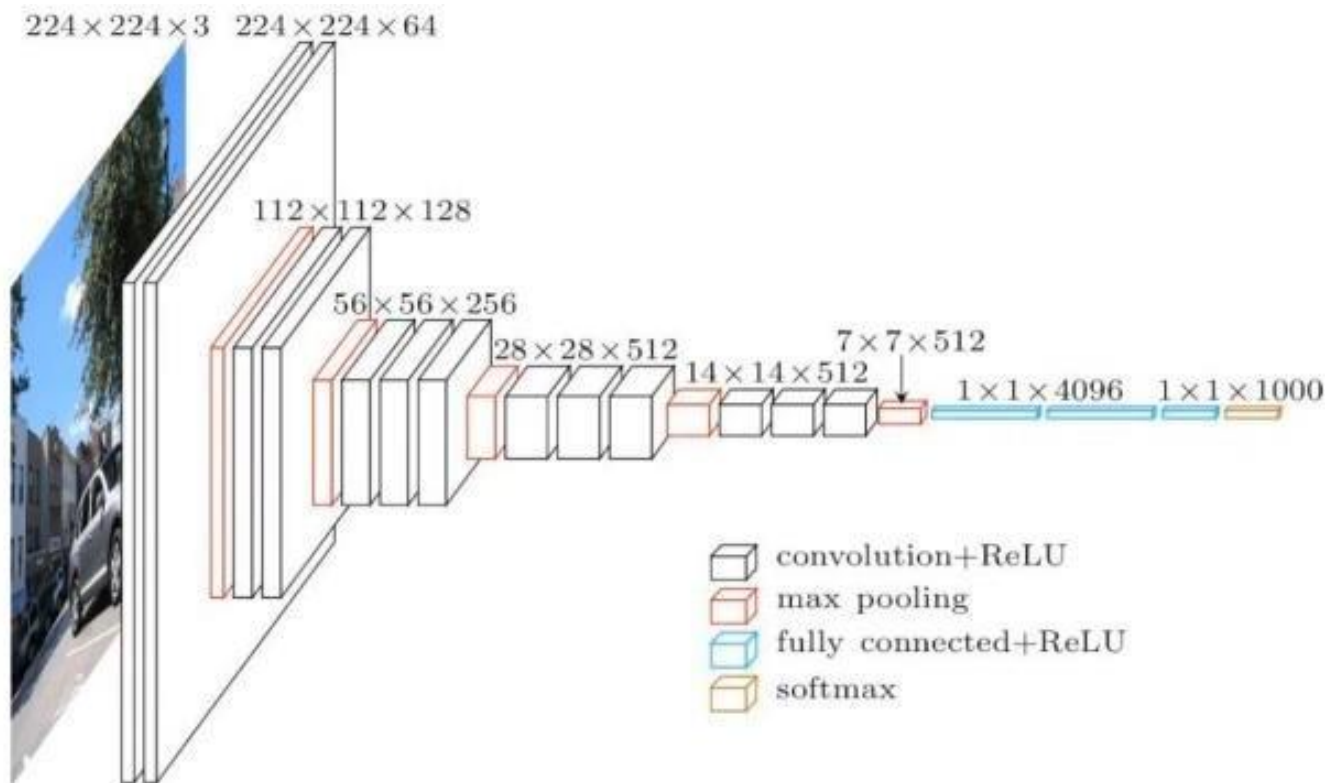
2. AlexNet



- Conv, Max Pooling, Dropout 5개
- Activation function : ReLU
- Batch Stochastic Gradient Descent

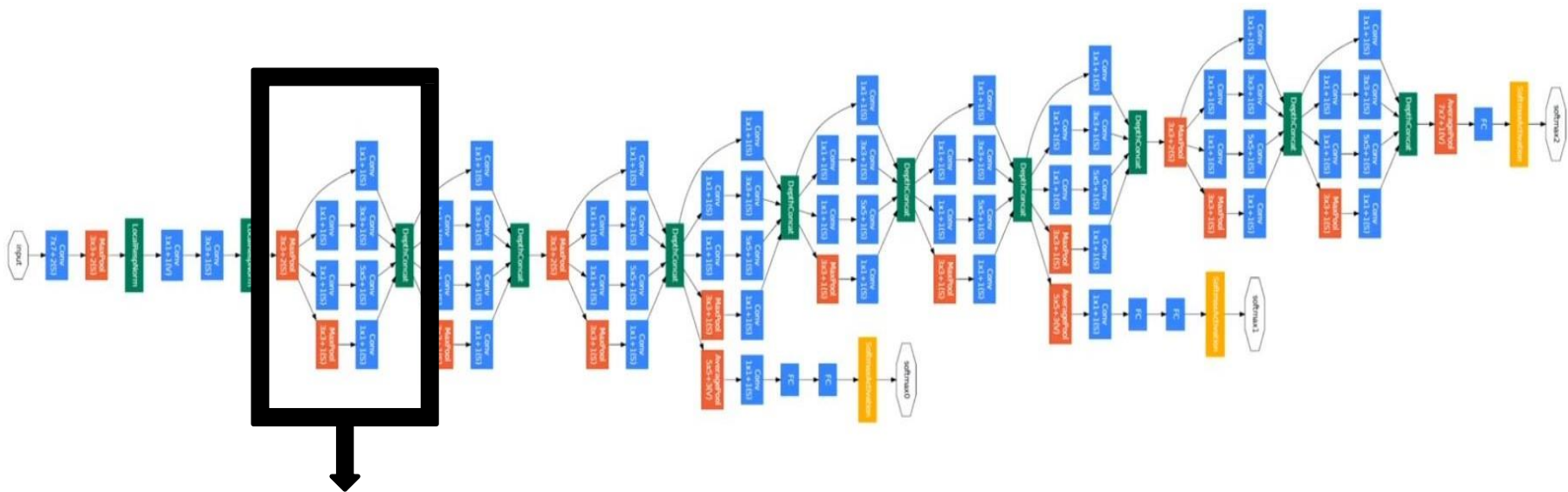
7. 심화 CNN

3. VGG-16



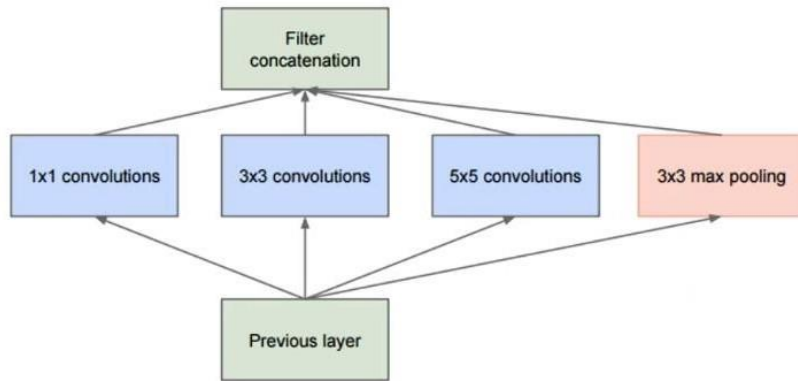
4. GoogLeNet (Inception)

“Let’s go Deeper and Deeper”

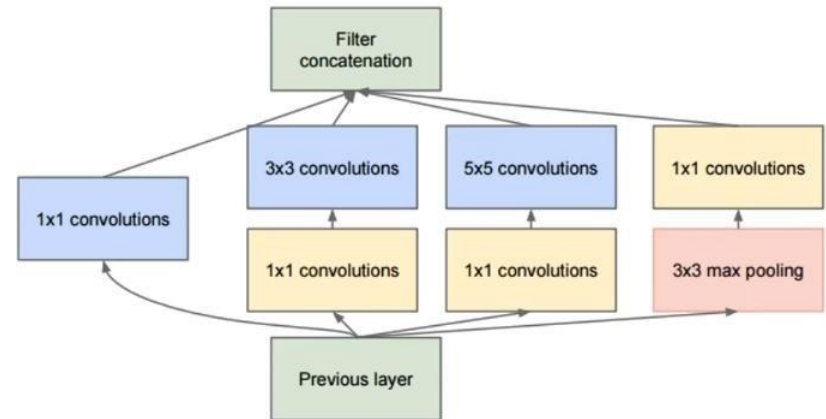


Inception 모듈에 대해 먼저 이해해야!

4. GoogLeNet (Inception)



(a) Inception module, naïve version



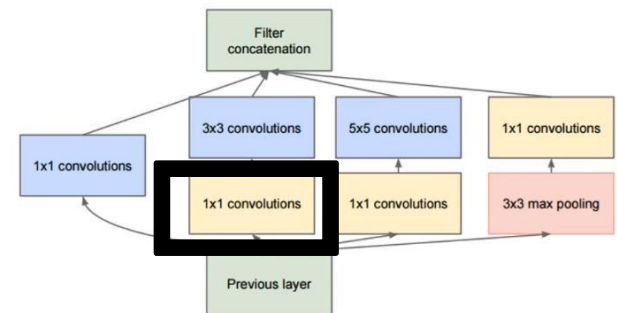
(b) Inception module with dimension reductions

Inception module: 다양한 크기의 합성곱 계층을 한 번에 계산!

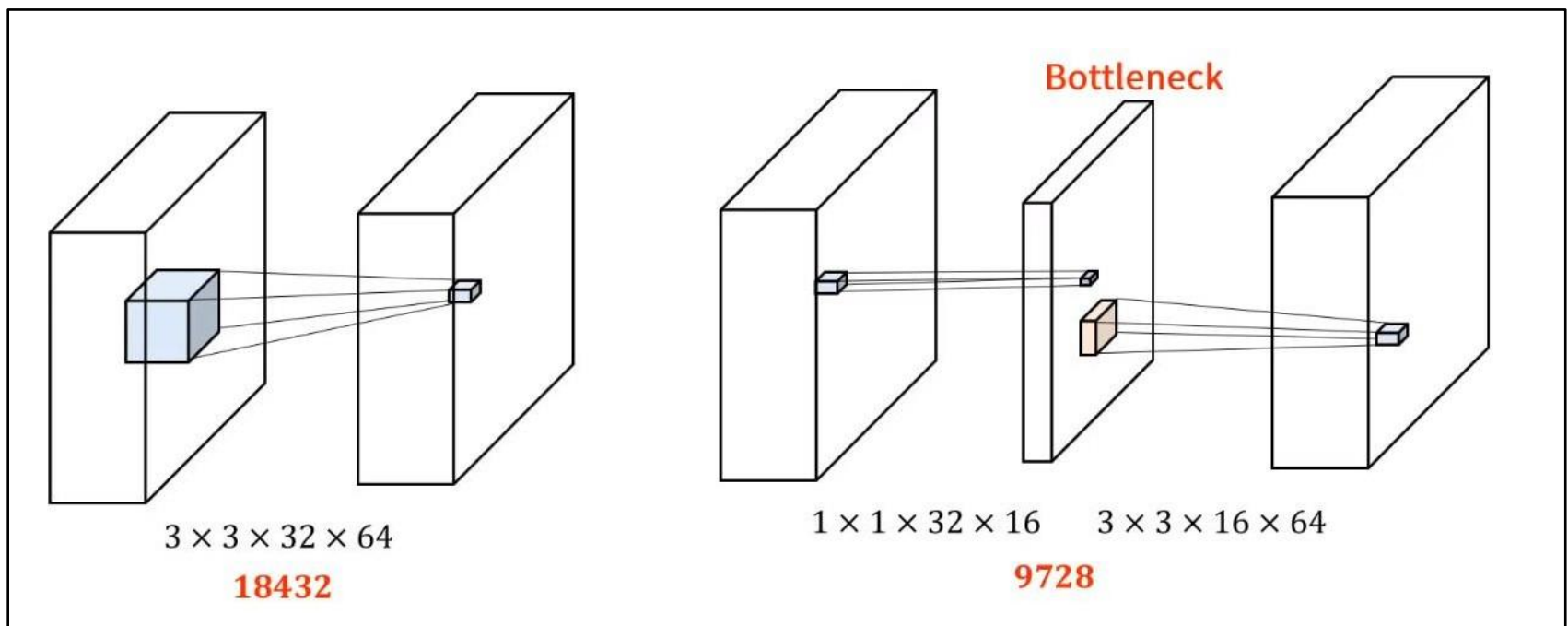
1x1 convolution : for 연산량 줄이기! (bottle neck 구조라고도 함)

7. 심화 CNN

4. GoogLeNet (Inception)



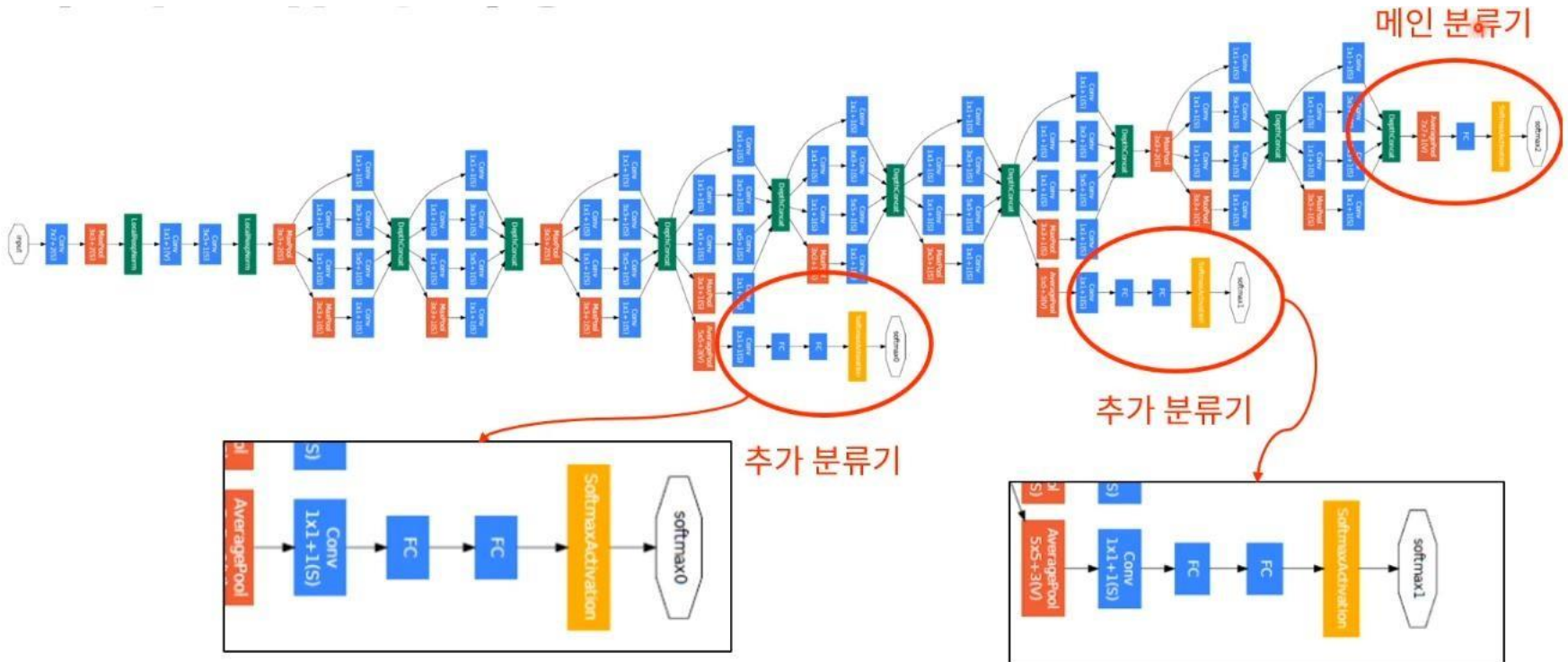
(b) Inception module with dimension reductions



1x1 Conv를 하면, 왜 연산량이 줄어들까?

7. 심화 CNN

4. GoogLeNet (Inception)

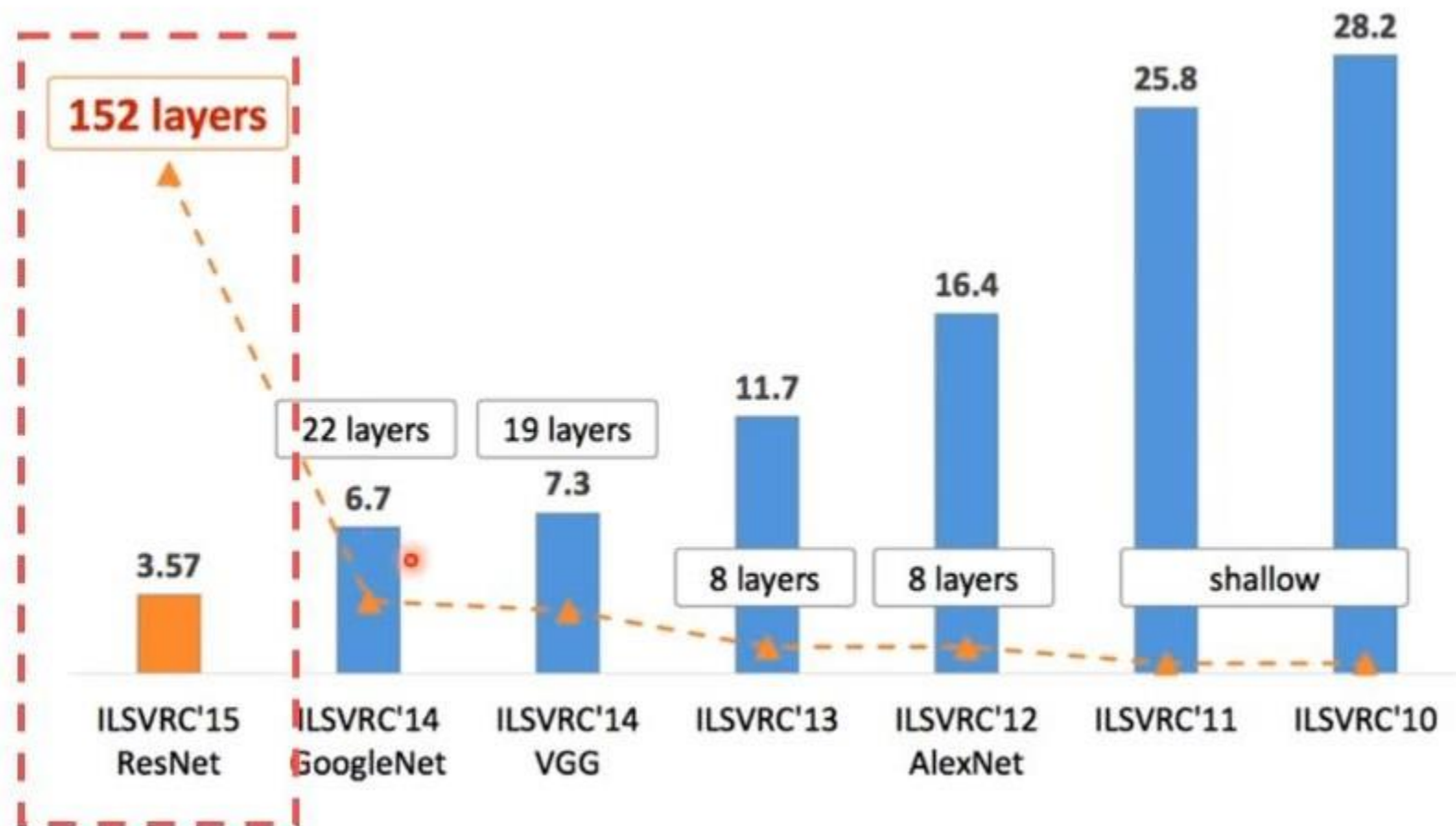


역전파에서 기울기 소실이 발생하는 것을 방지하기 위해 추가 분류기 존재

5. ResNet

152 Layers? H o w that deep?

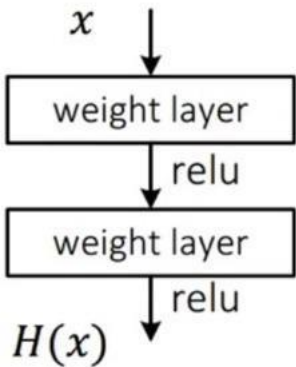
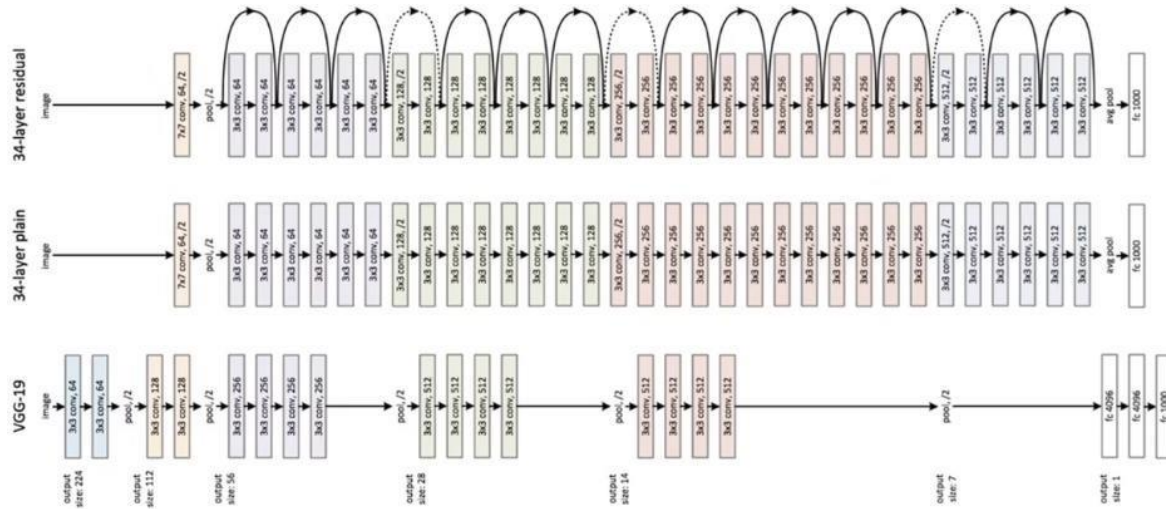
Residual Block



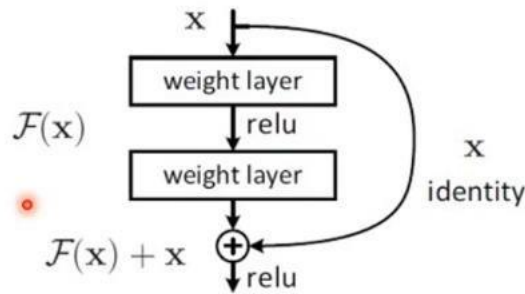
7. 심 화 CNN

5. ResNet

사이에 사이에 있는 Skip-Connection이 한 몫함!



일반적인 구조

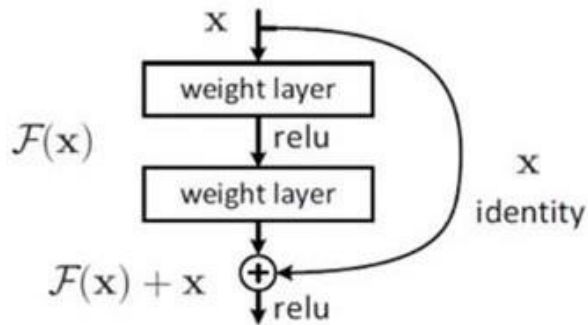


Residual 구조

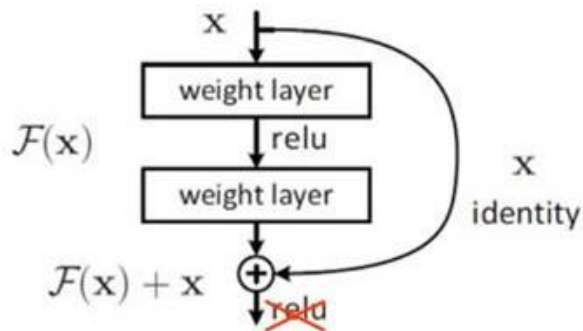
Feature를 추출하기 전/후를 더함!

7. 심 화 CNN

5. ResNet

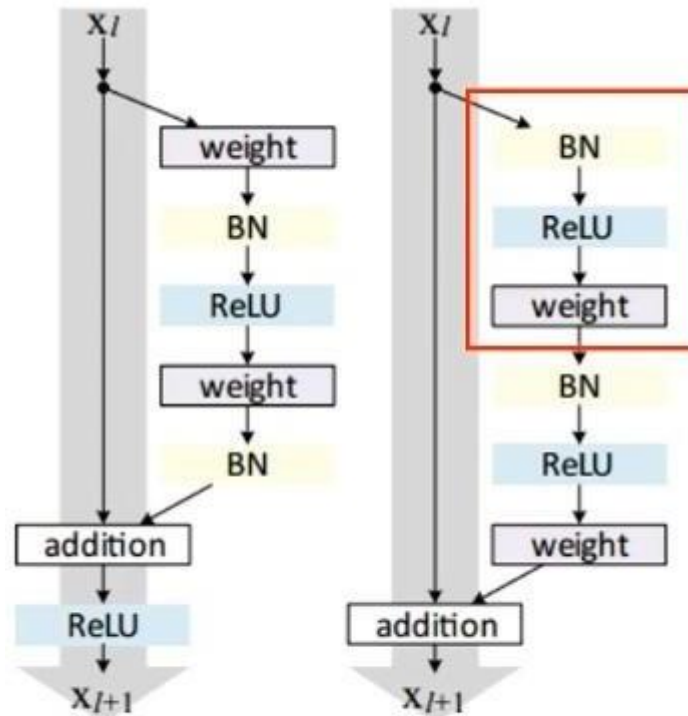


Residual 구조



Identity Mapping

Feature map을 추출한 뒤, activation function을 하던 것이 일반적이었으나, 개선된 구조에서는 “Pre-Activation” (activation function이 먼저 들어감!)

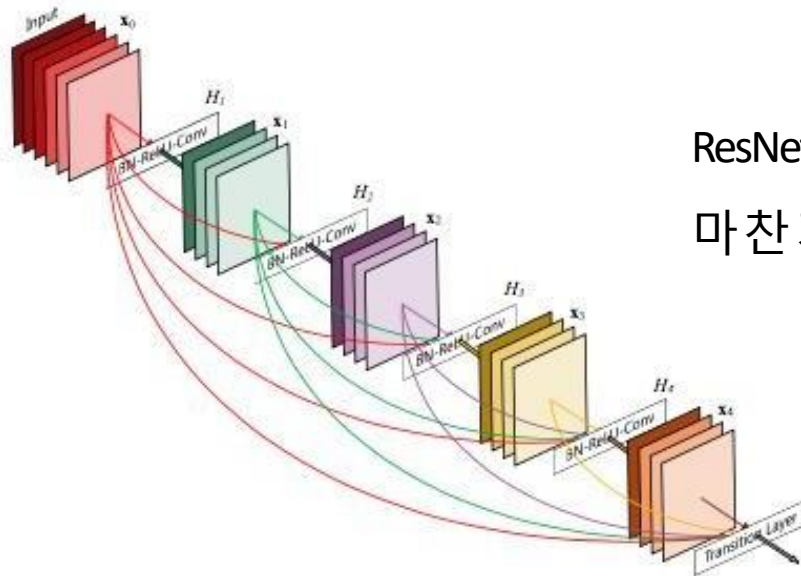


(a) original

(b) proposed

7. 심화 CNN

6. DenseNet



ResNet 아이디어의 연장선 상!

마찬가지로 Pre-Activation (BN-ReLU-Conv)

Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

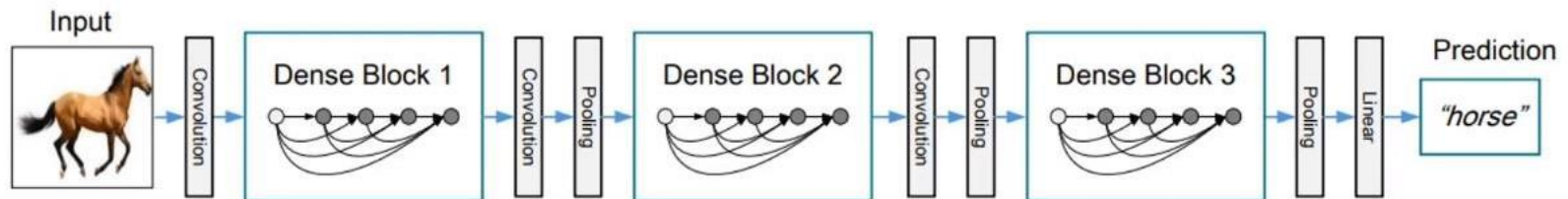
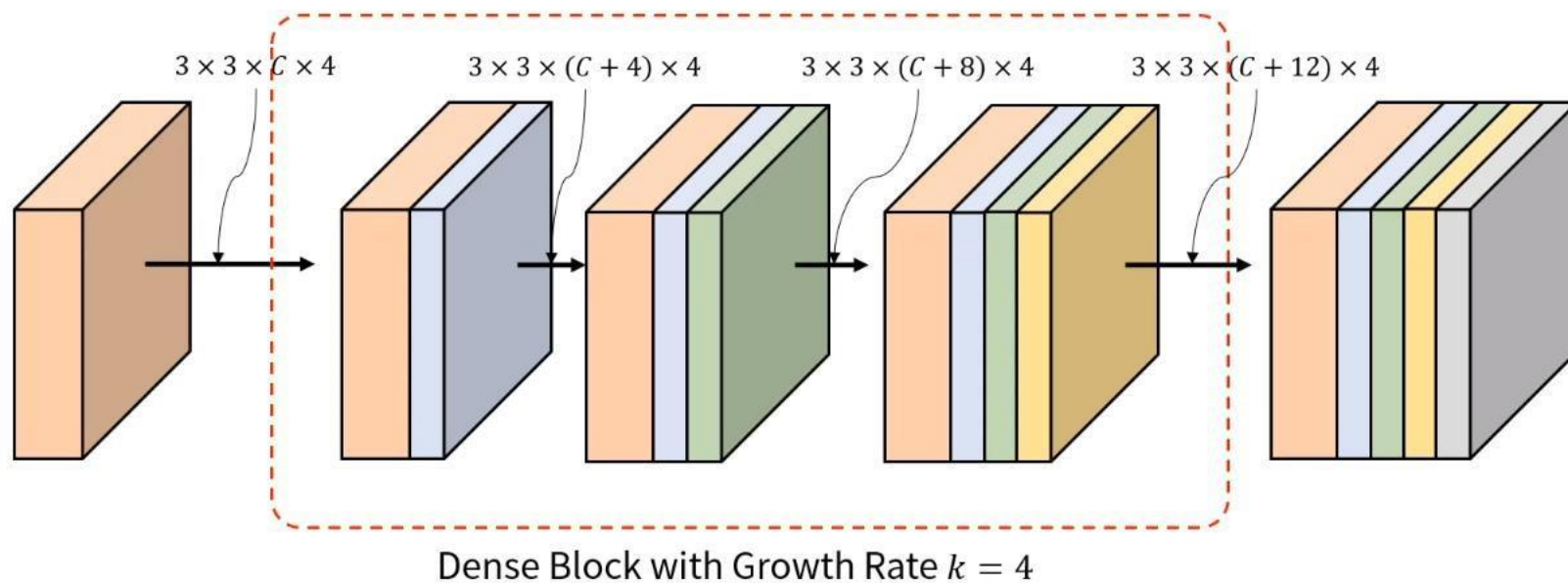


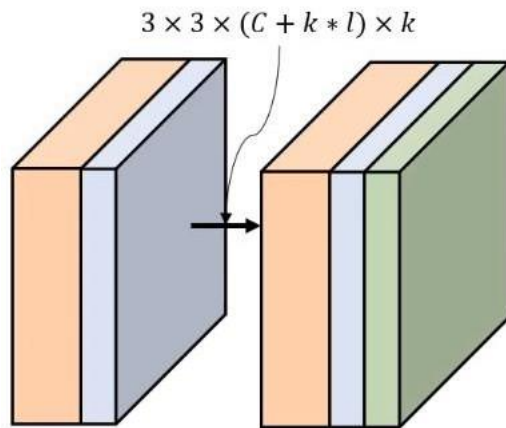
Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

6. DenseNet

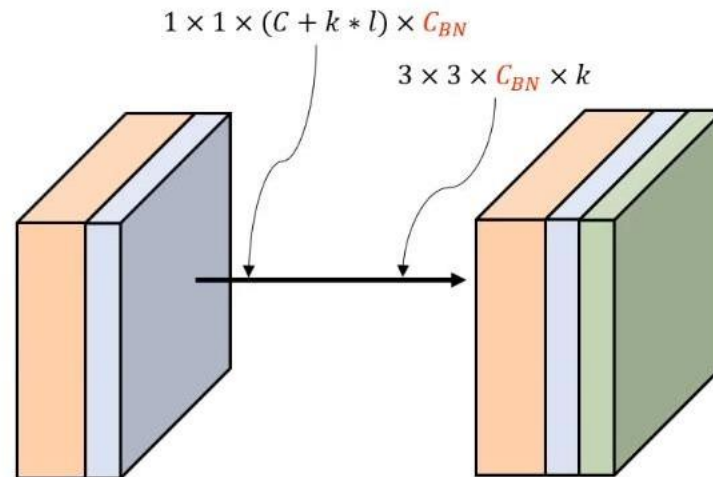


쉽게 말해, “이전 feature map에 누적해서 concatenate!”

6. DenseNet



일반 Dense block 계산



Bottleneck 구조

Deep -> 연산량 too much!

그래서 "1x1Conv (= Bottleneck Layer)" 사용 함

8. Summary

C N N is good for image classification!

