

Convolutional Neural Networks

-Basic-

2018.02.28

최건호

INDEX

01

Before CNN

- Limit of NN
- Hubel & Wiesel
- LeNet-5

02

Convolution
Layer

- Kernel
- Stride
- Padding

03

Basic
Architecture

- Subsampling
- ReLU
- Channels
- Loss Function

04

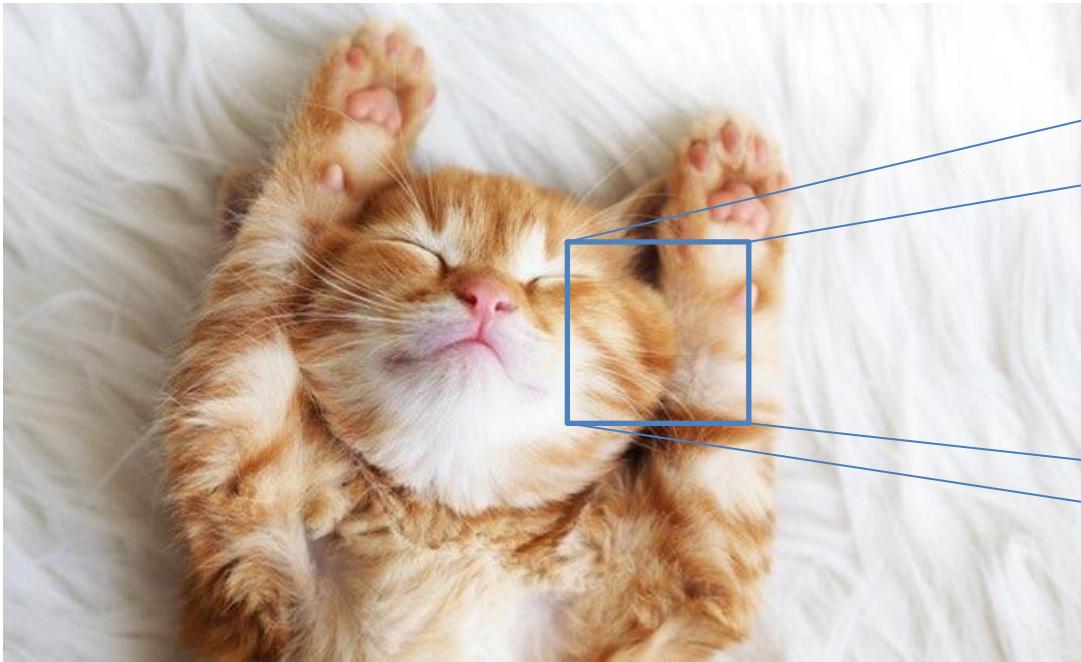
MNIST Example

- Dataset
- Model Design
- Code

Before CNN

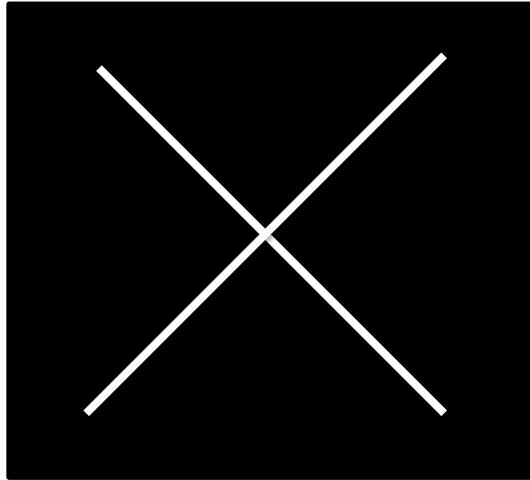


Before CNN



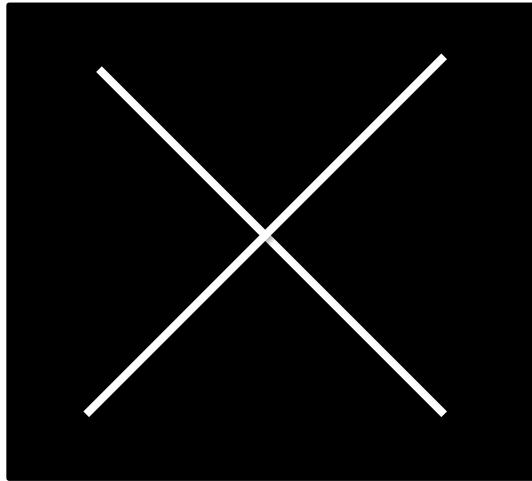
```
010111010101001  
010010100100101  
00010111101010  
101010100101110  
101010010100101  
001001010001011  
111010101010101
```

Before CNN

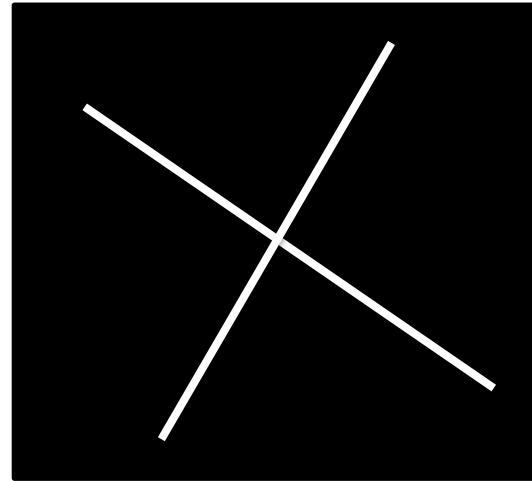
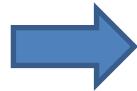


=X

Before CNN



=X



=?

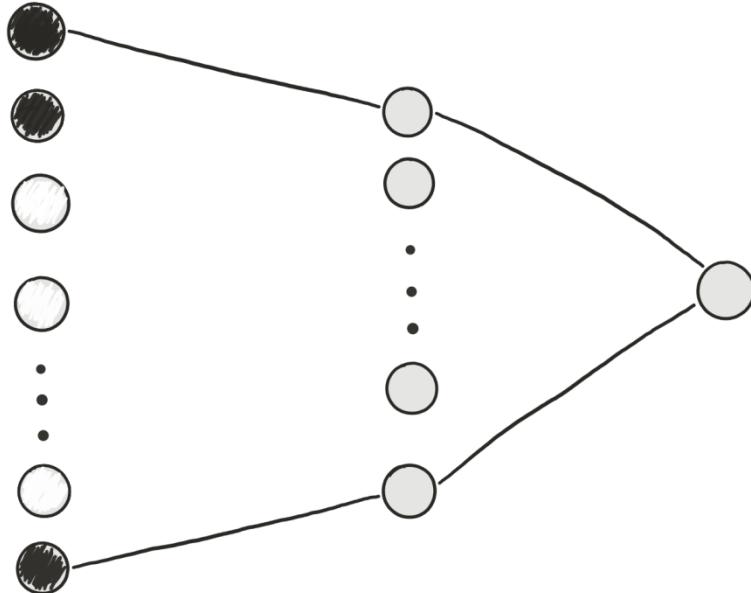
Before CNN

?

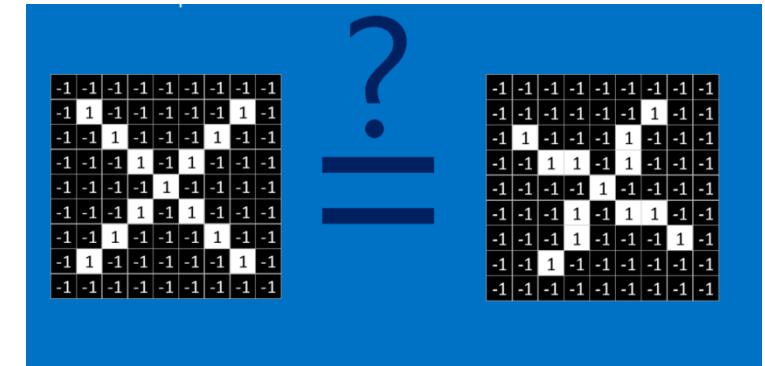


Before CNN

Input



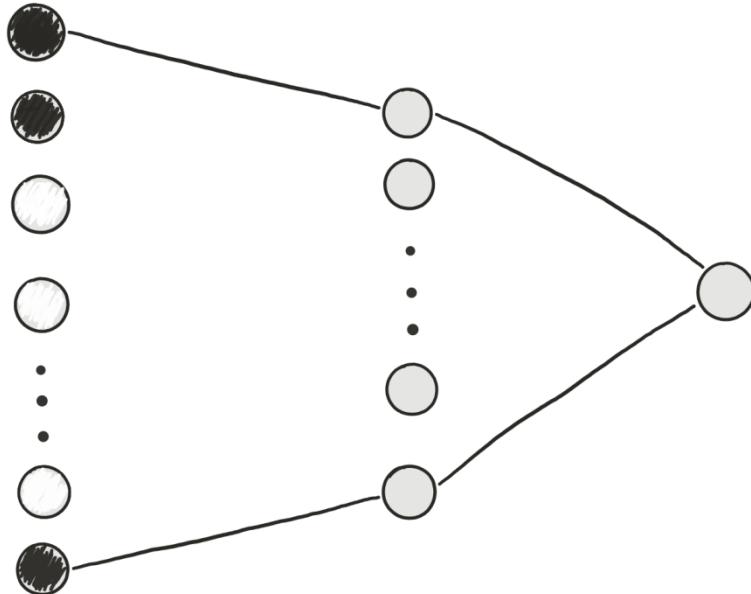
$$9 \times 9 = 81$$



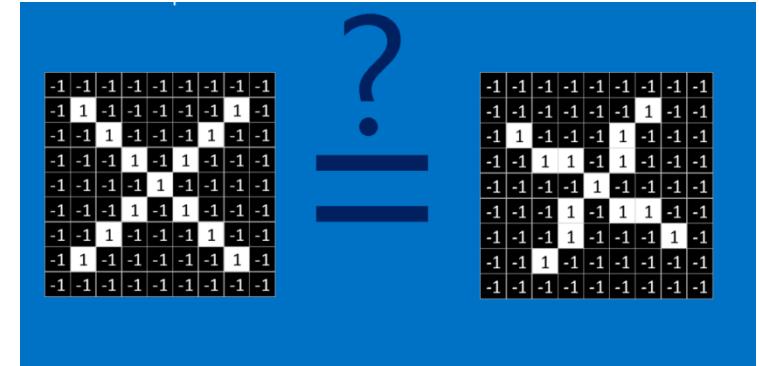
모양이 x인가
아닌가에 대한 결과값

Before CNN

Input



$$9 \times 9 = 81$$

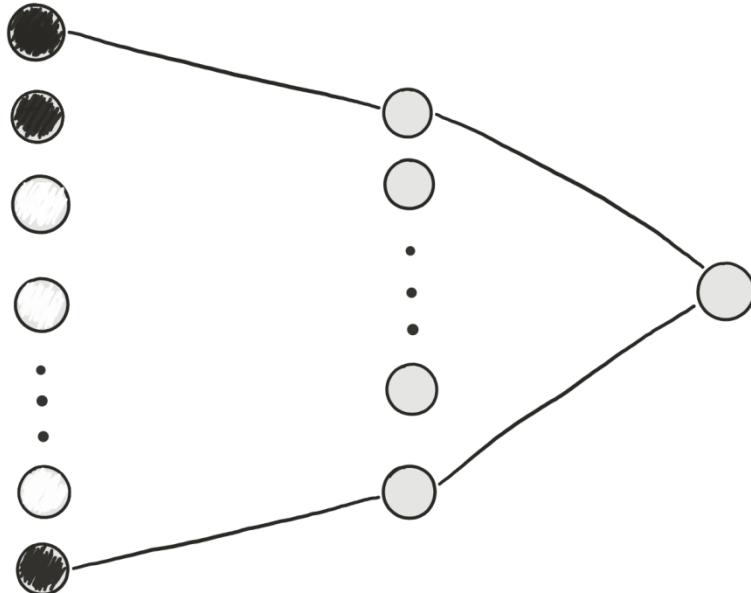


모양이 x인가
아닌가에 대한 결과값

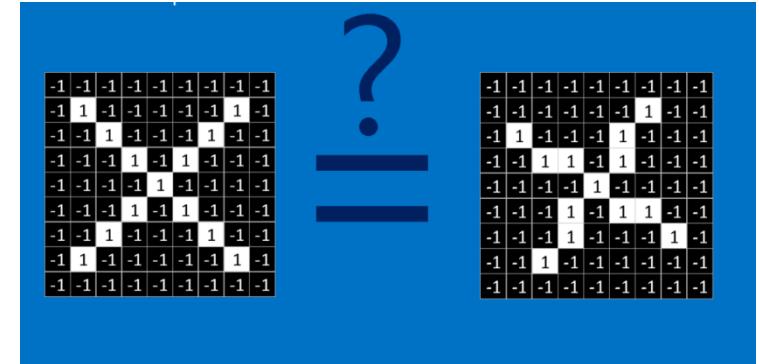
하지만 그림이 기울어진다거나
하면 완전히 다른 입력 값이
들어오는 것

Before CNN

Input



$$9 \times 9 = 81$$



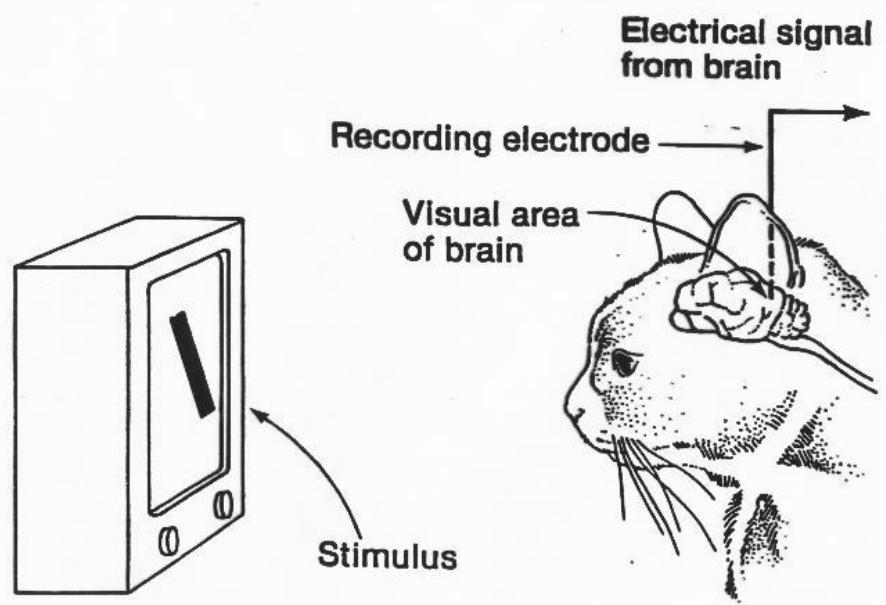
모양이 x인가
아닌가에 대한 결과값

하지만 그림이 기울어진다거나
하면 완전히 다른 입력 값이
들어오는 것

보기에는 같은 모양이지만
기계는 전혀 다르게 인식

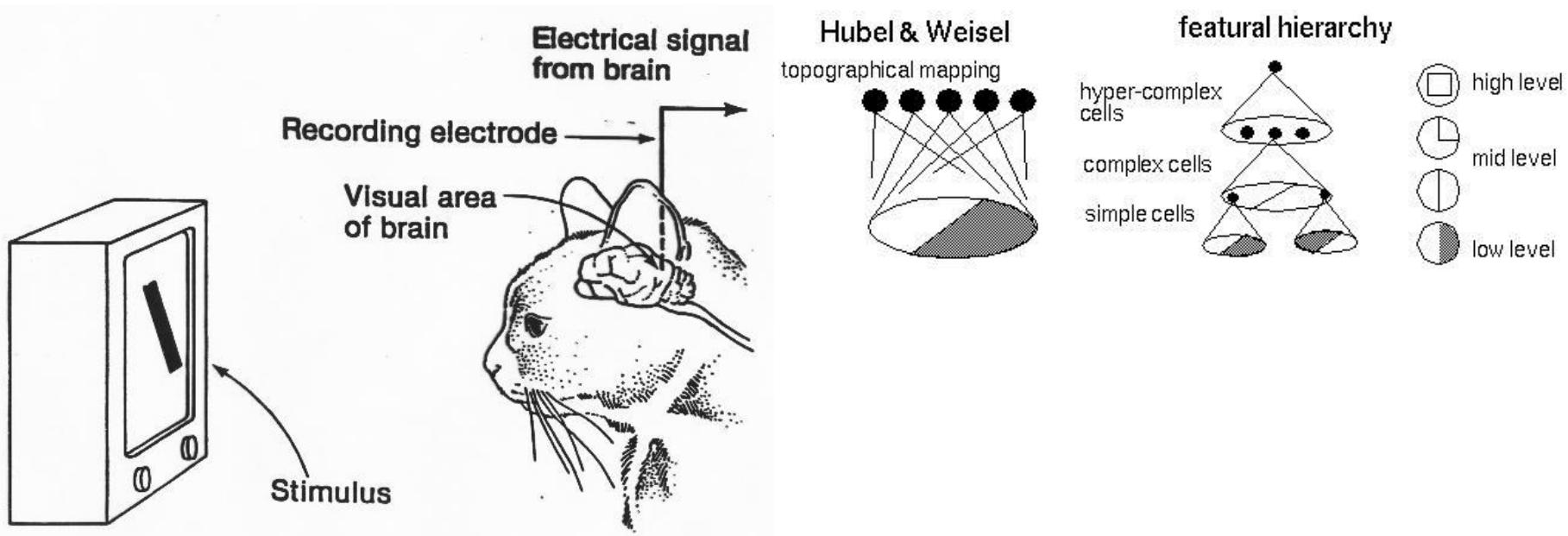
Before CNN

Hubel & Wiesel (1950s)



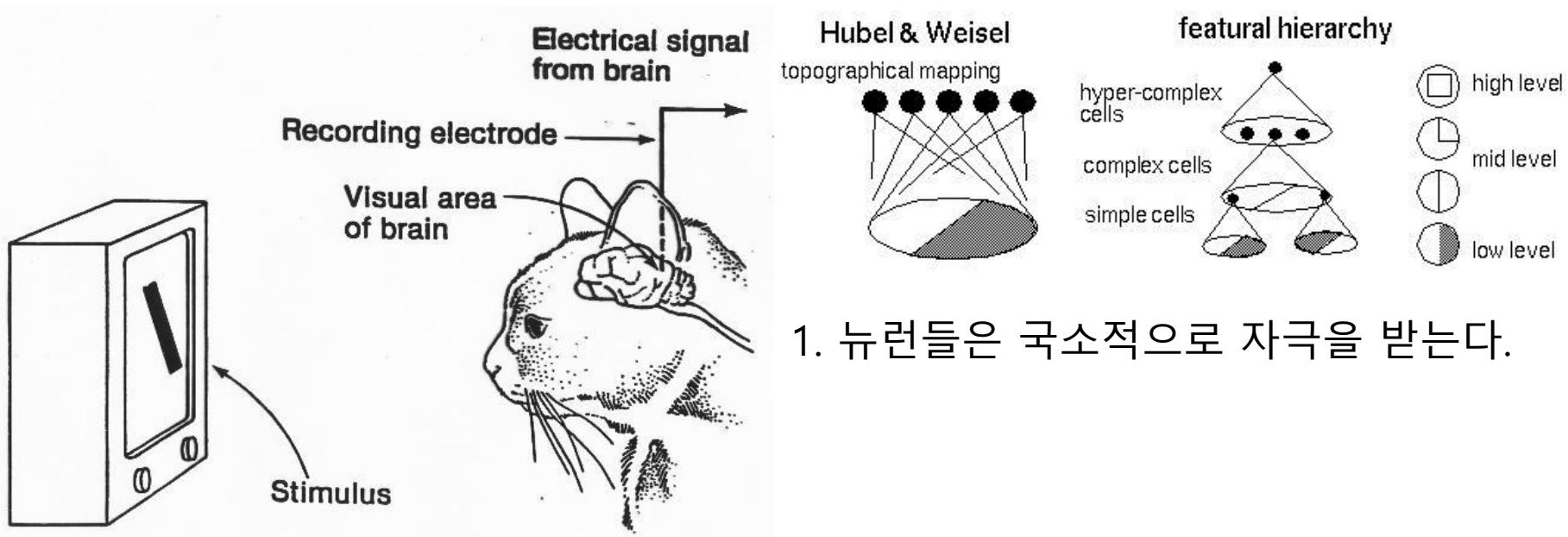
Before CNN

Hubel & Wiesel (1950s)



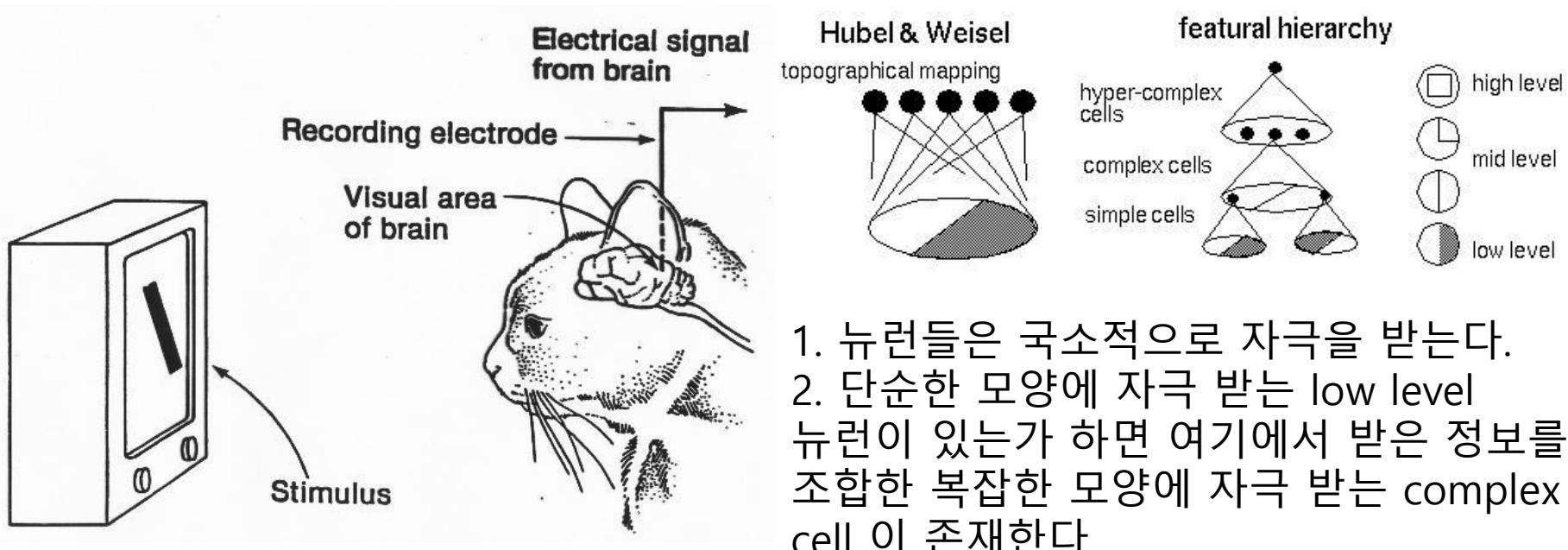
Before CNN

Hubel & Wiesel (1950s)



Before CNN

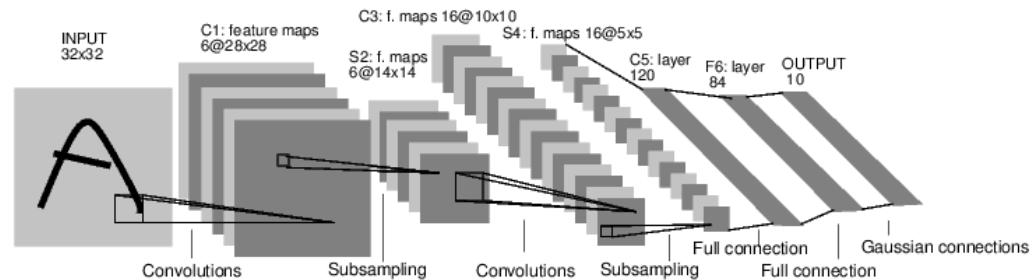
Hubel & Wiesel (1950s)



Before CNN

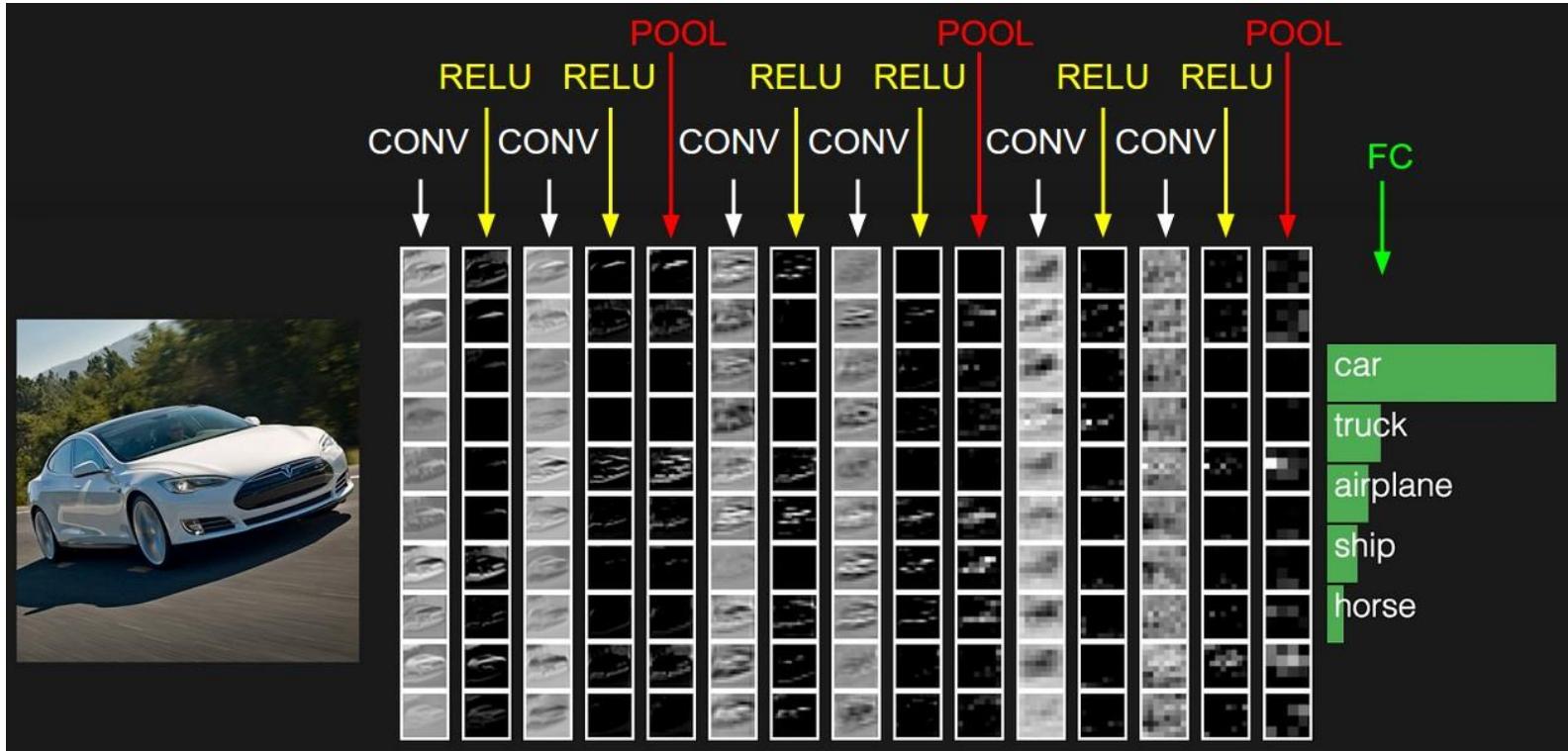


LeNet-5 (1998)



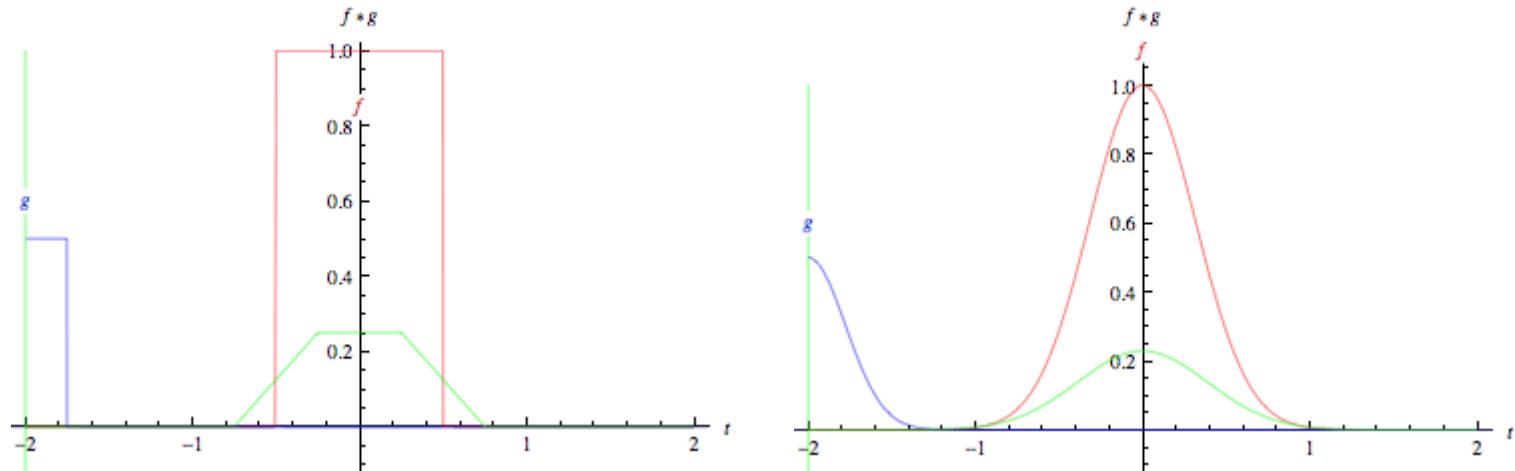
Convolutional Neural Network의 시초

Before CNN



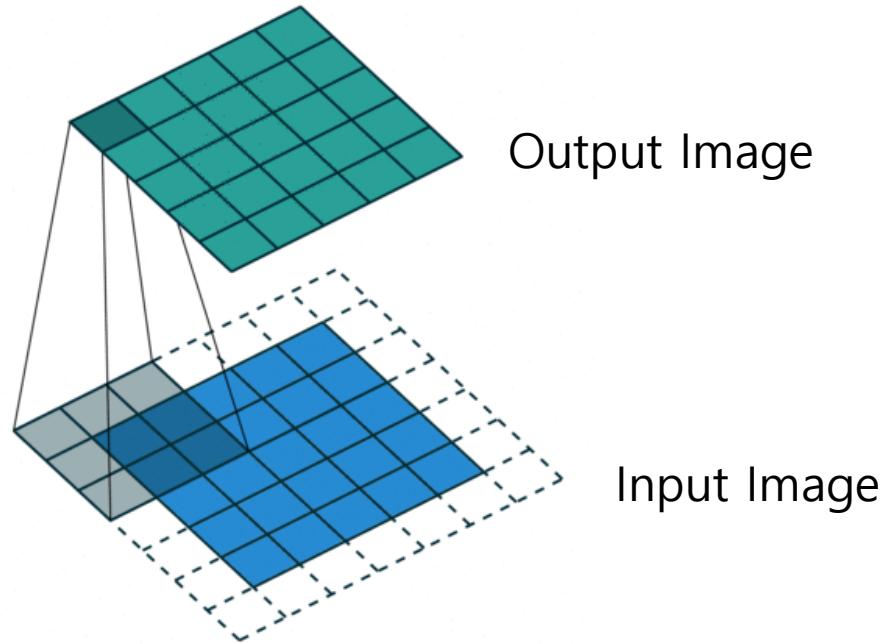
Convolution Layer

Convolution 연산?

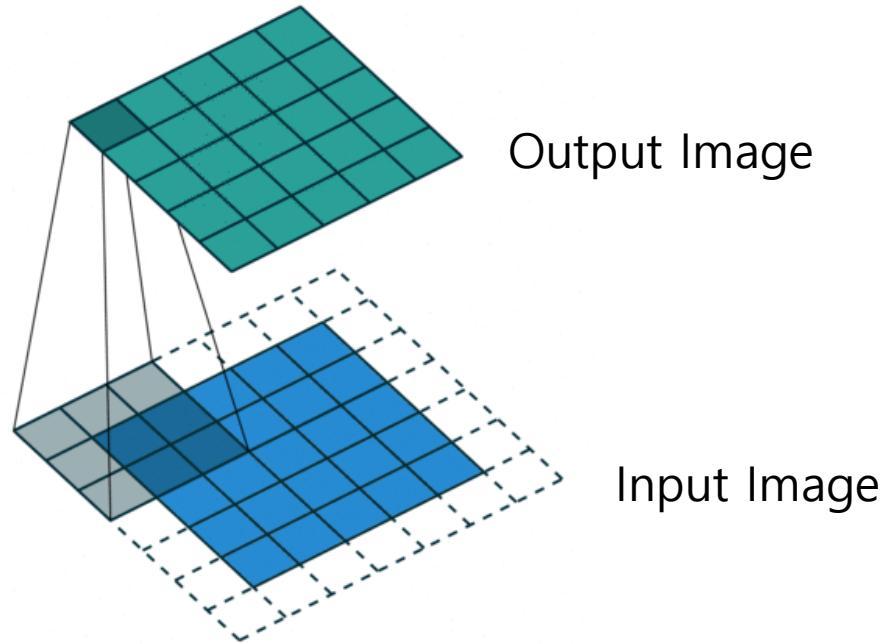


함수 f 에 대하여 함수 g 가 얼마나 겹치는가

Convolution Layer



Convolution Layer



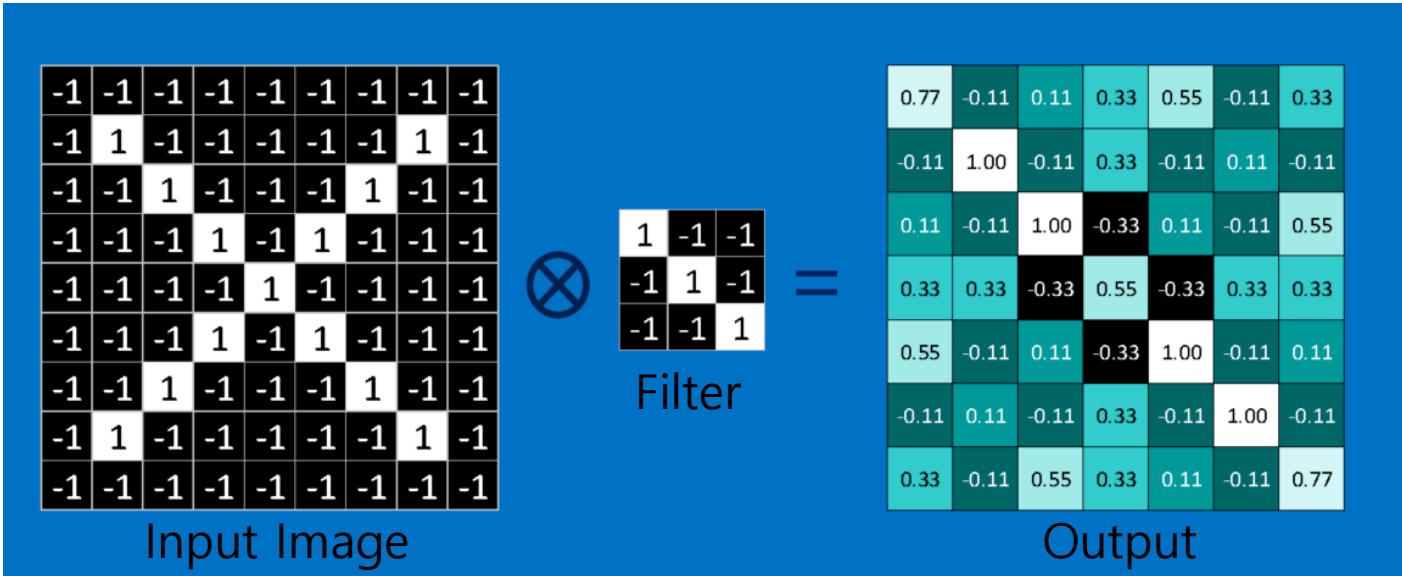
1. Kernel(or filter)
2. Stride
3. Padding

Convolution Layer

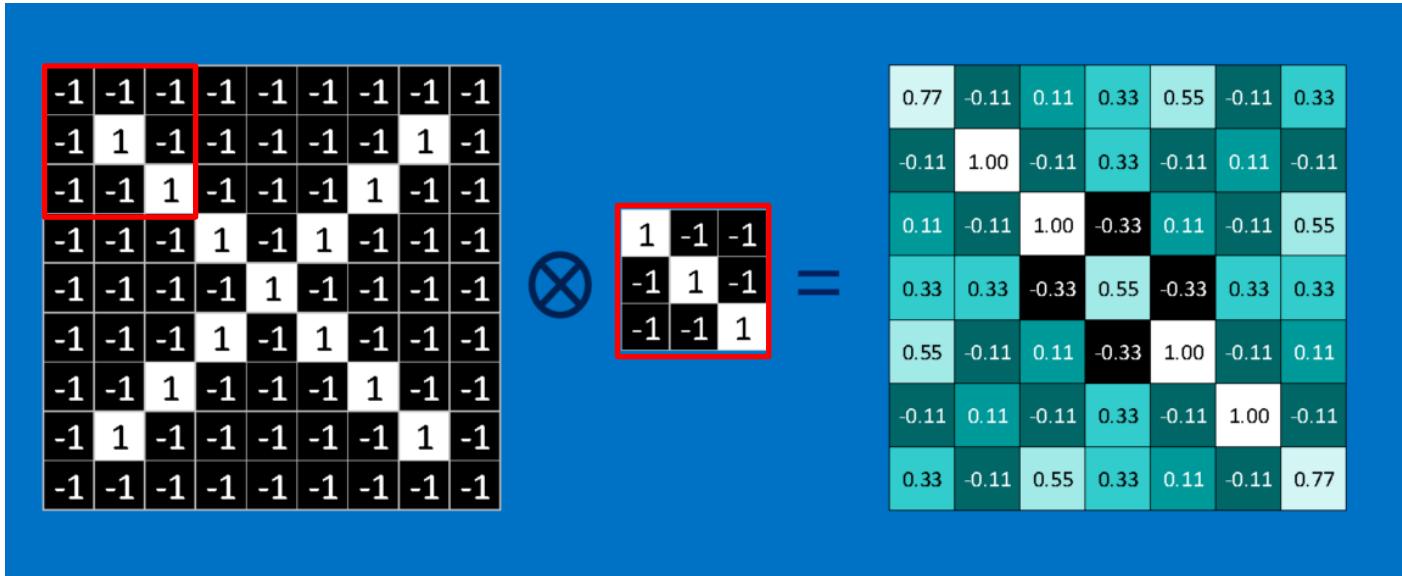
The diagram illustrates a convolution operation. It shows an input matrix (9x9) and a kernel matrix (3x3) being multiplied to produce an output matrix (7x7). The input matrix has alternating 1s and -1s. The kernel matrix has values 1, -1, -1; -1, 1, -1; -1, -1, 1. The output matrix is calculated as follows:

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 |
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |

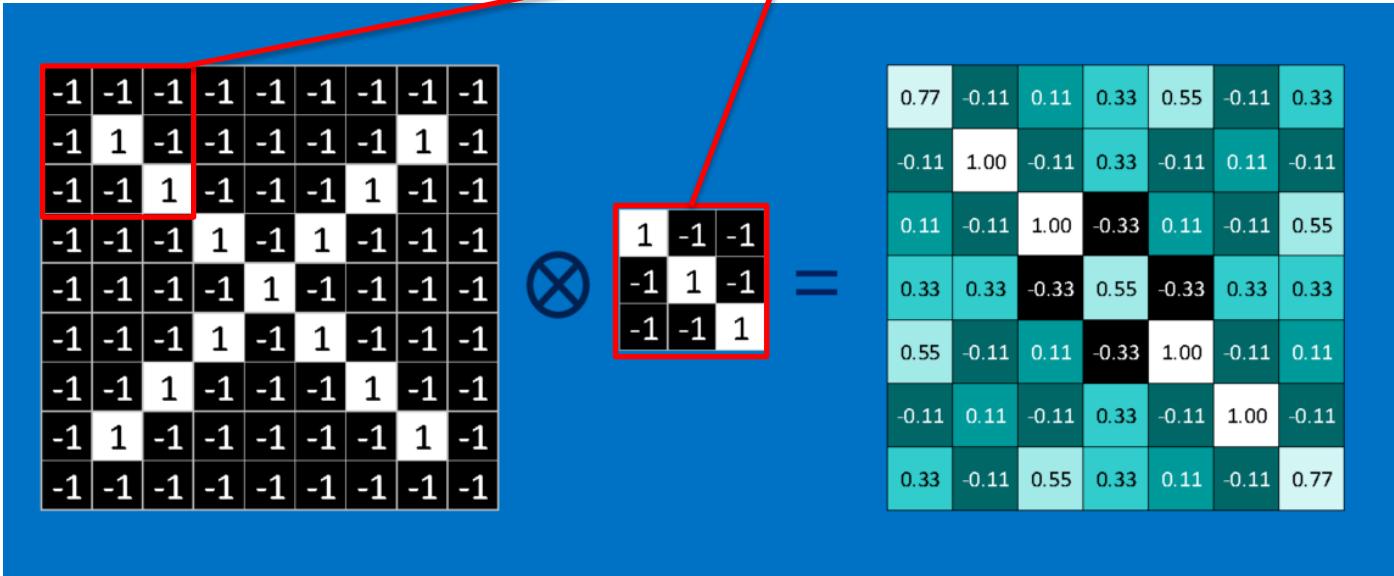
Convolution Layer



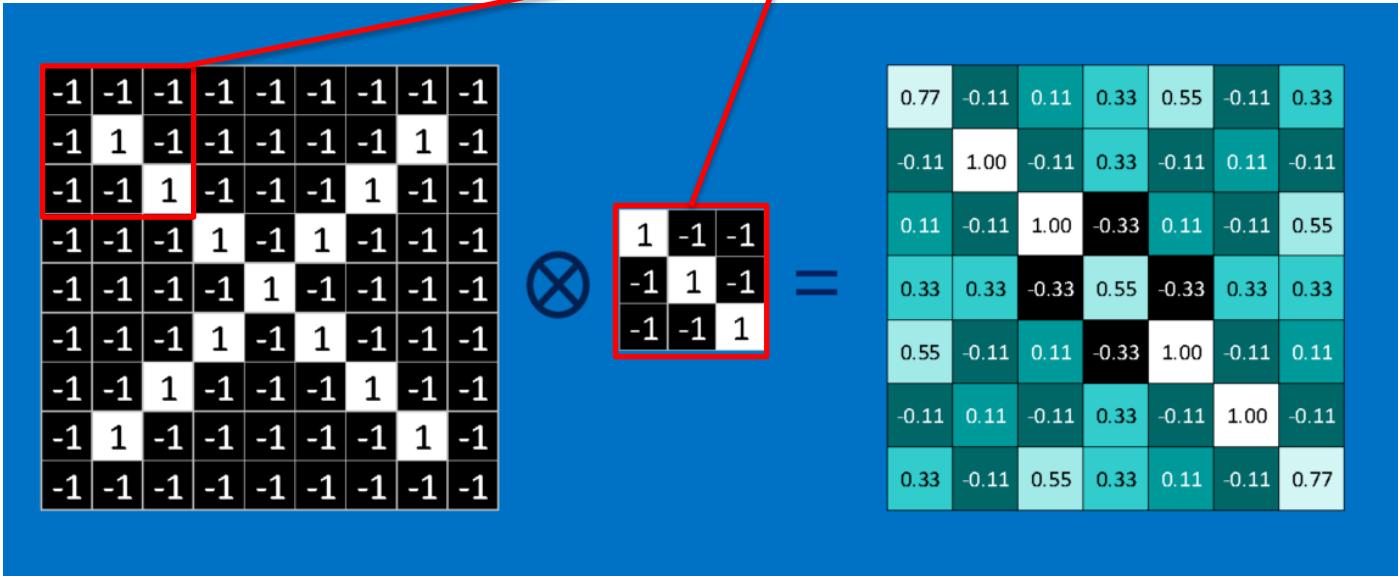
Convolution Layer



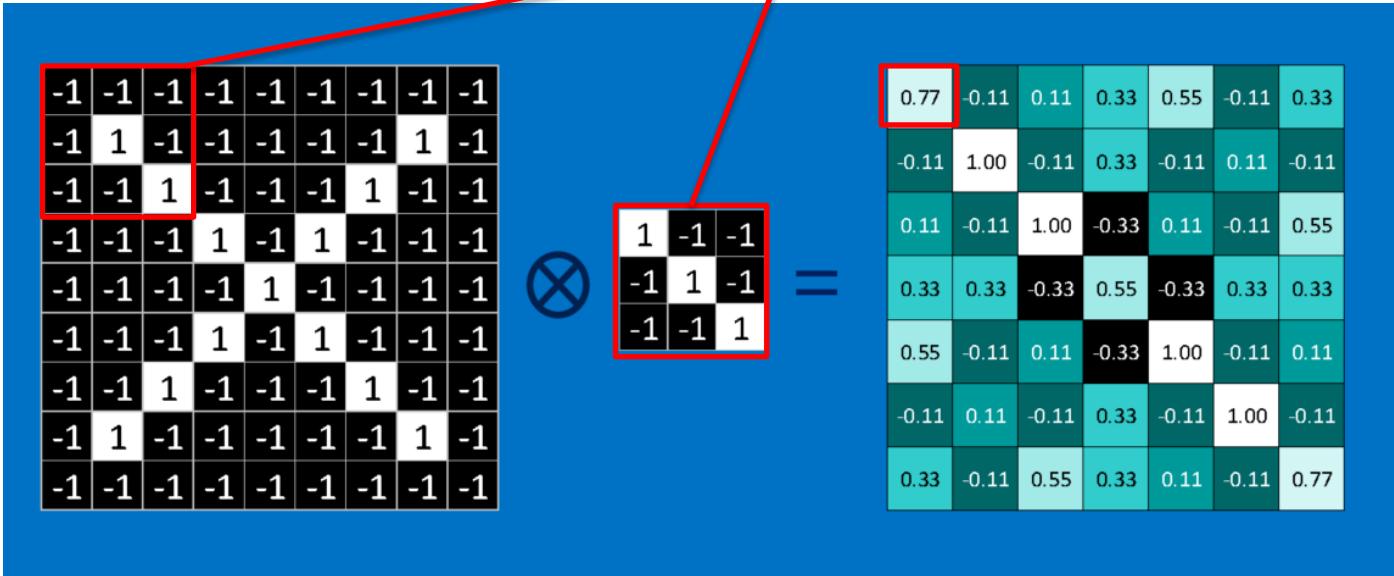
Convolution Layer



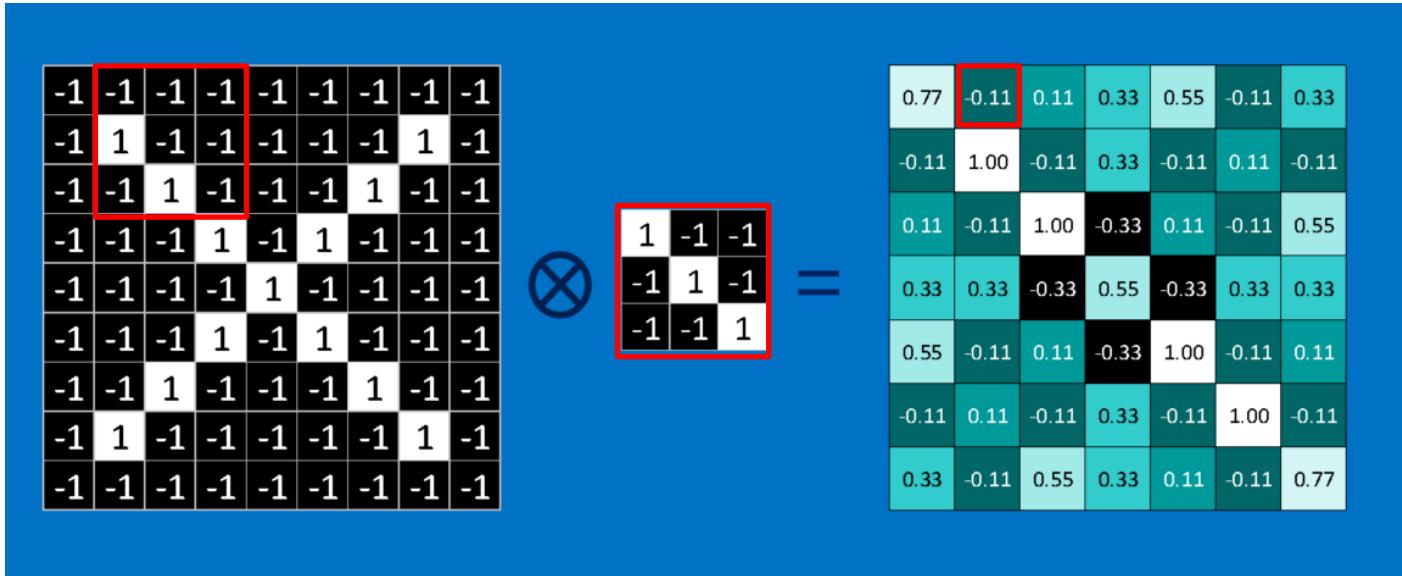
Convolution Layer



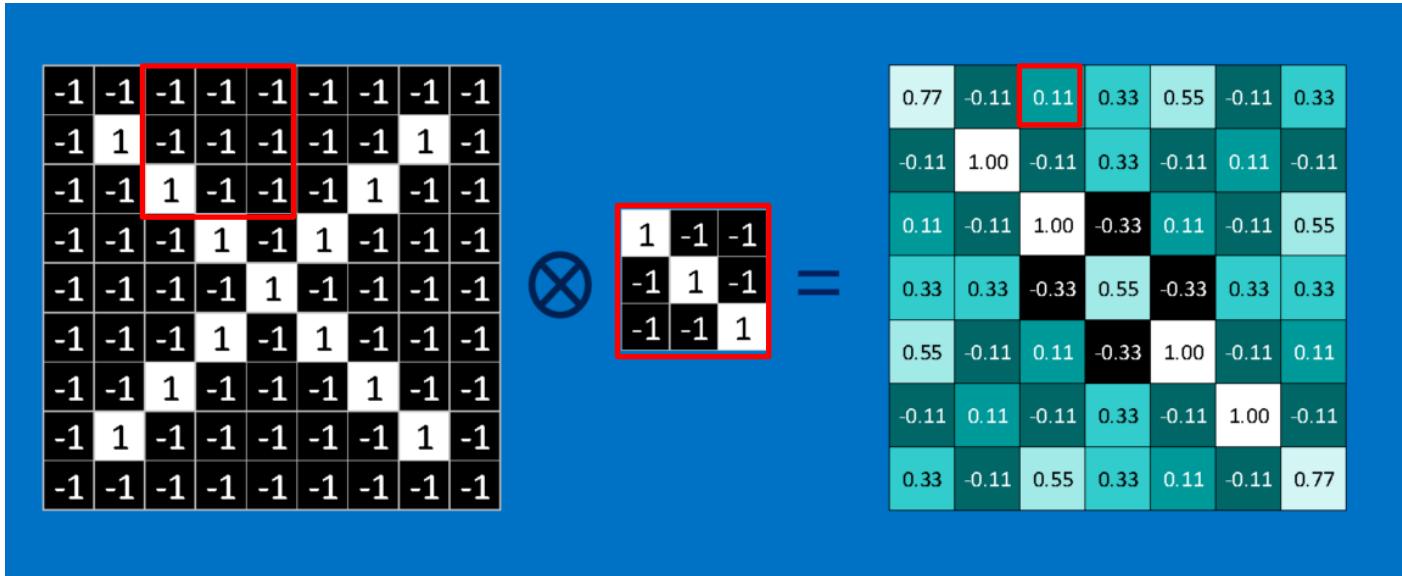
Convolution Layer



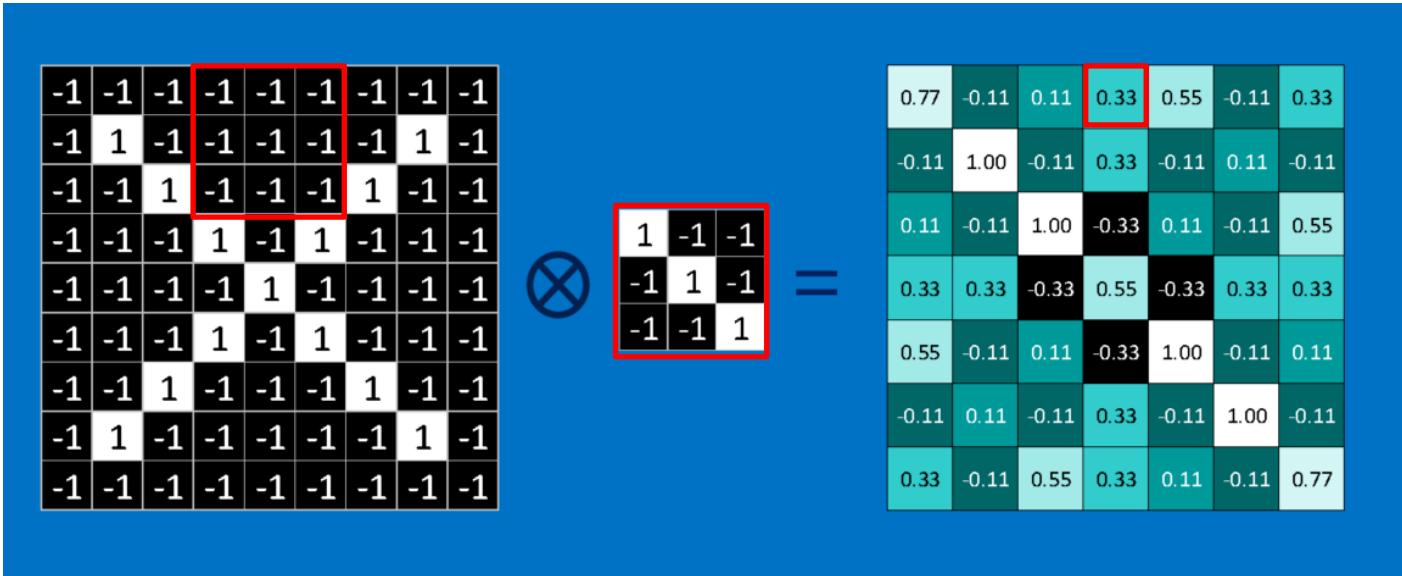
Convolution Layer



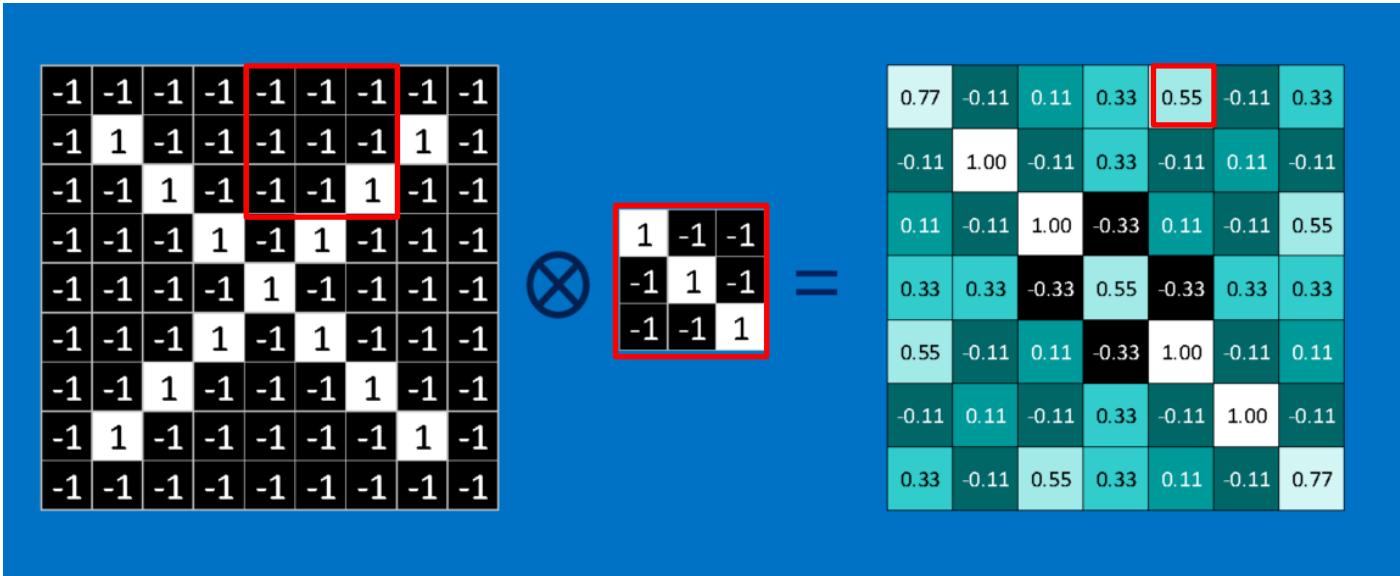
Convolution Layer



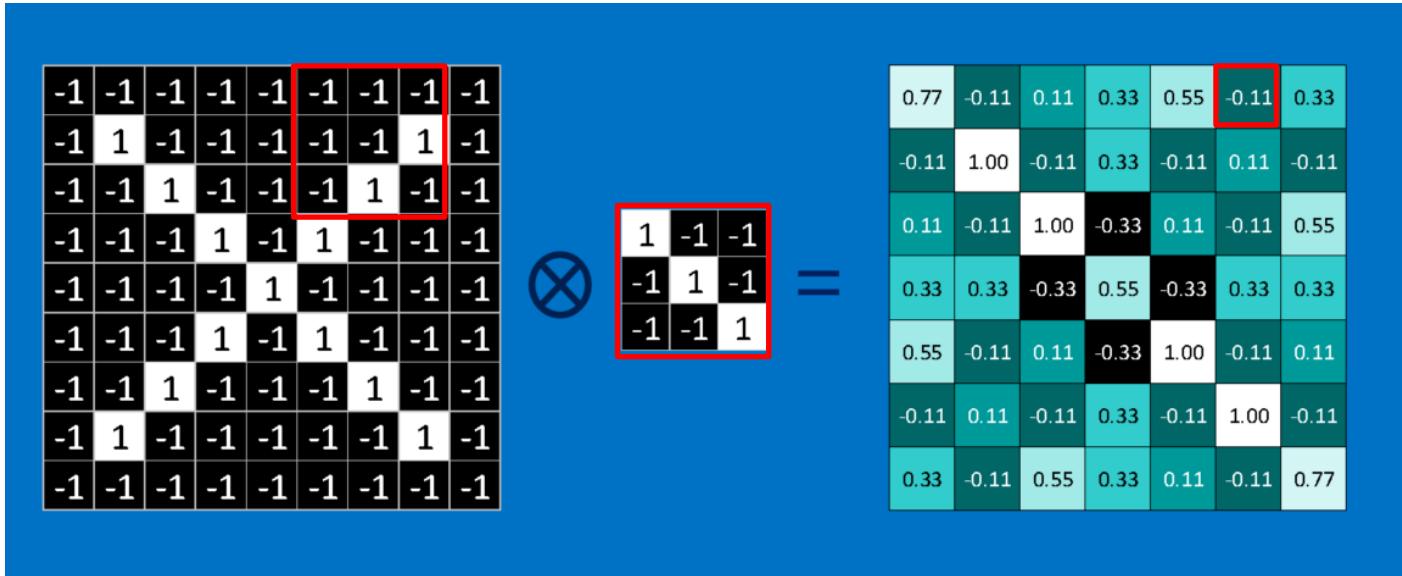
Convolution Layer



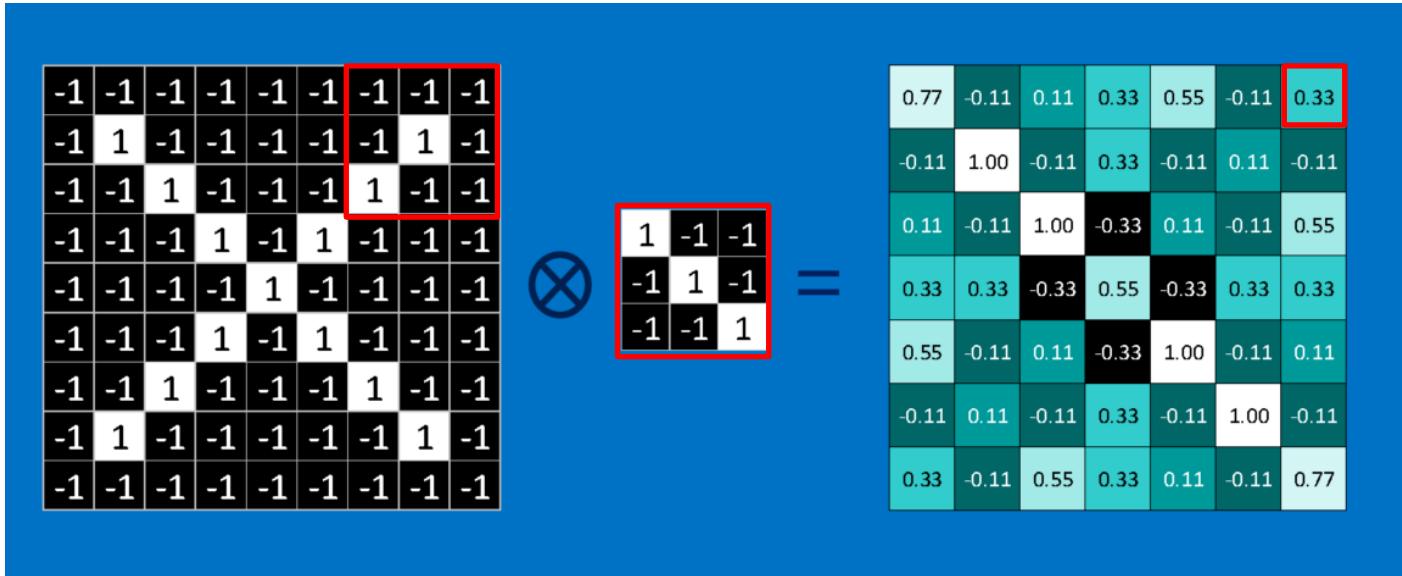
Convolution Layer



Convolution Layer



Convolution Layer



Convolution Layer

The diagram illustrates a convolution operation. On the left, an input image of size 9x9 is shown with values -1 or 1. In the center, a kernel of size 3x3 with values 1, -1, -1; -1, 1, -1; -1, -1, 1 is shown. To the right of the multiplication symbol (\otimes) is an equals sign (=). On the far right, the resulting output image of size 4x4 is shown with values 0.77, -0.11, 0.11, 0.33; -0.11, 1.00, -0.11, 0.33; 0.11, -0.11, 1.00, -0.33; 0.33, 0.33, -0.33, 0.55.

Image size를 I, kernel size를 K, stride를 S라고 하면
output size O는 아래의 식과 같음.

Convolution Layer

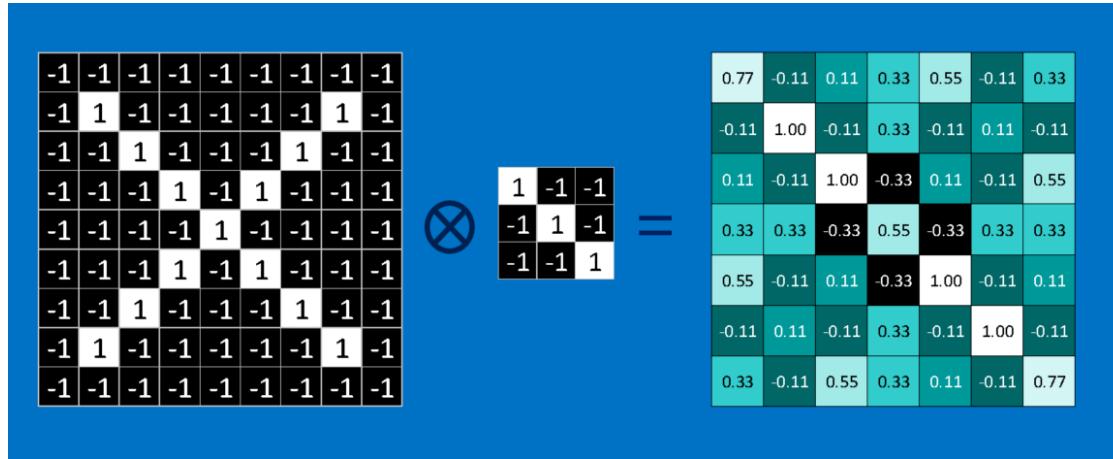
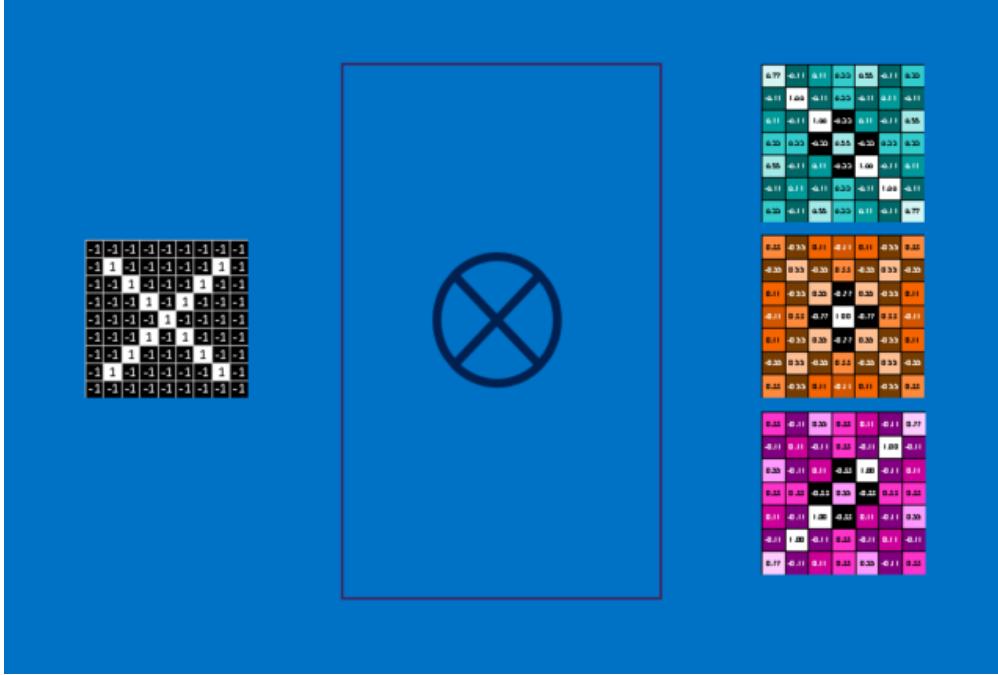


Image size를 I, kernel size를 K, stride를 S라고 하면
output size O는 아래의 식과 같음.

$$O = \text{floor}\left(\frac{I - K}{S} + 1\right)$$

Convolution Layer



하나의 이미지에 대해서 여러 종류의 필터를 적용할 수 있음

Basic Architecture

| | | |
|---|---|---|
| 0 | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |

Image

$$\begin{matrix} & \times & \\ \text{Image} & \times & \text{Filter} \end{matrix} = \begin{matrix} \text{Output} \end{matrix}$$

X

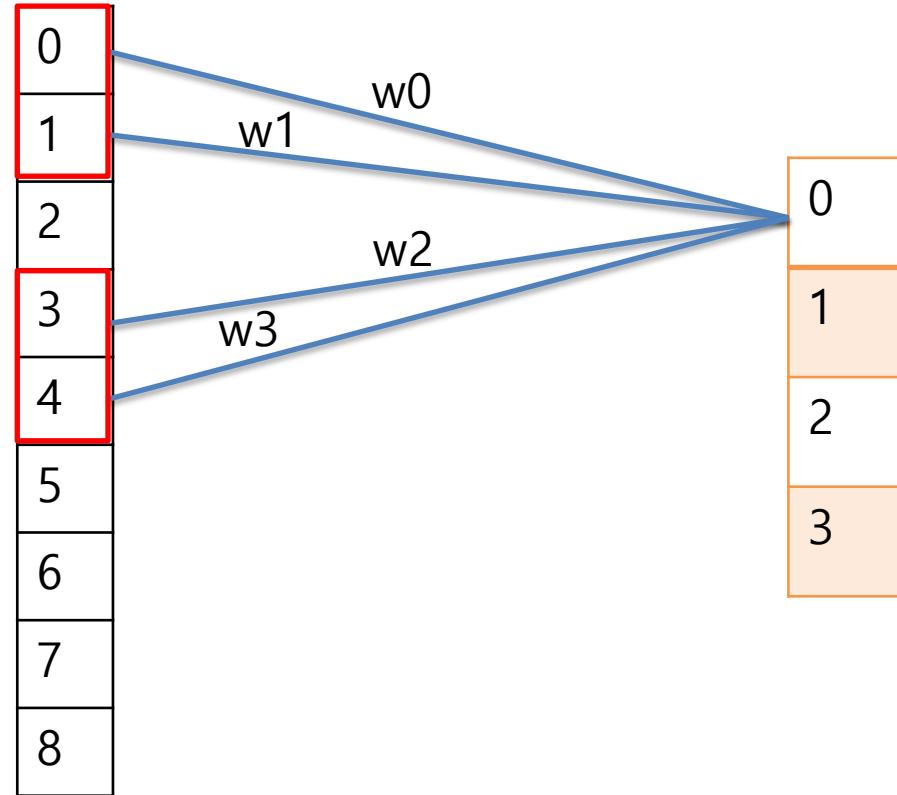
| | |
|----|----|
| w0 | w1 |
| w2 | w3 |

| | |
|---|---|
| 0 | 1 |
| 2 | 3 |

Output

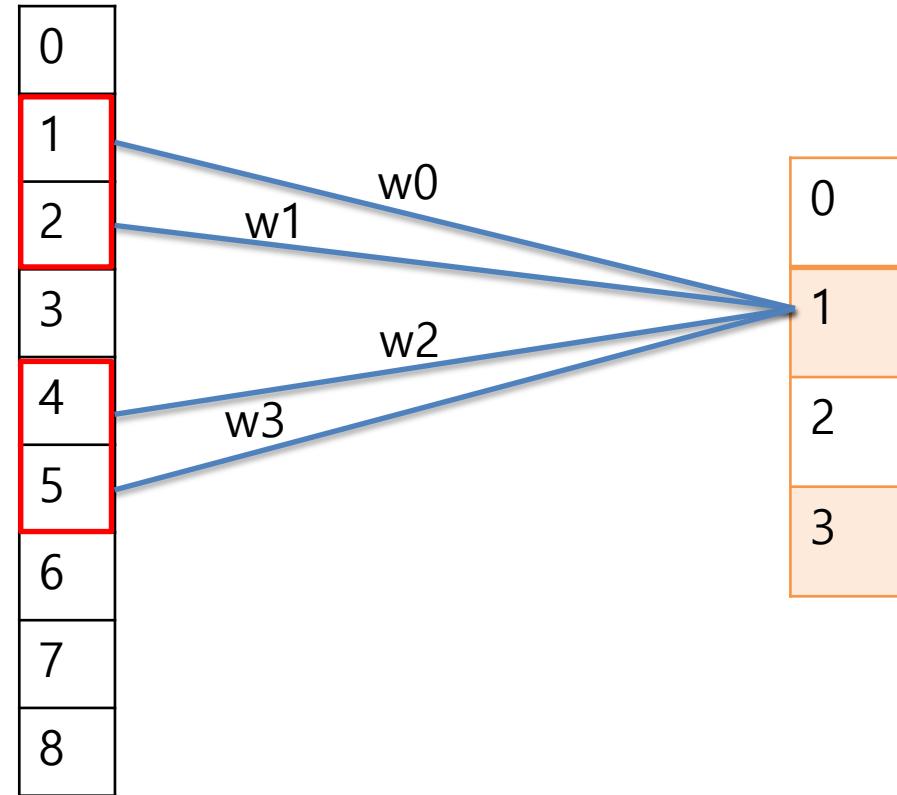
Basic Architecture

| | | |
|---|---|---|
| 0 | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |



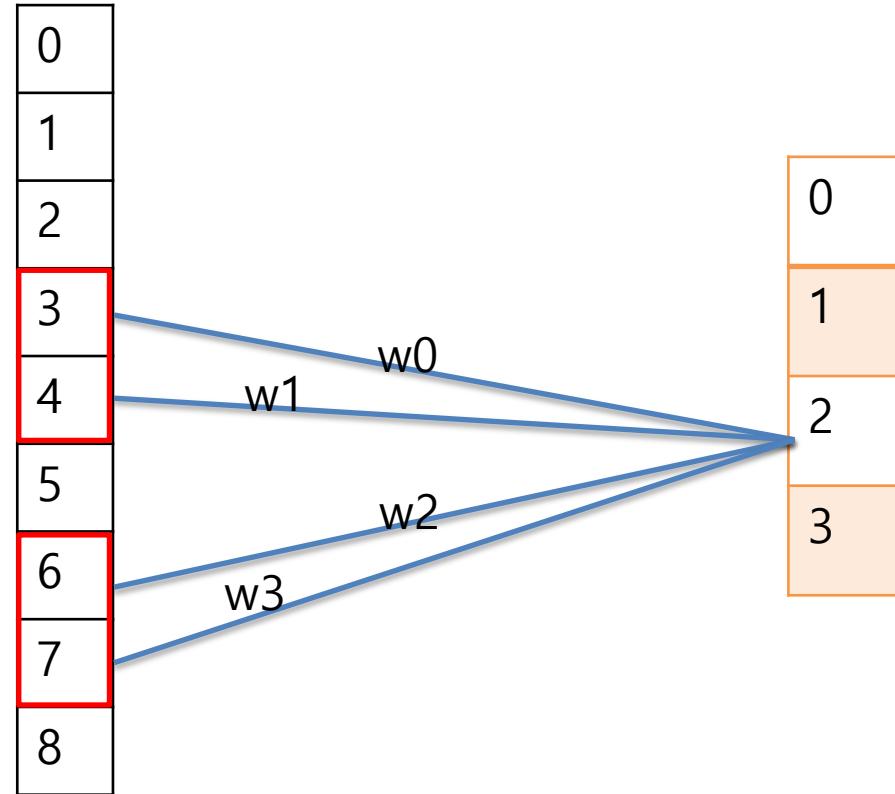
Basic Architecture

| | | |
|---|---|---|
| 0 | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |



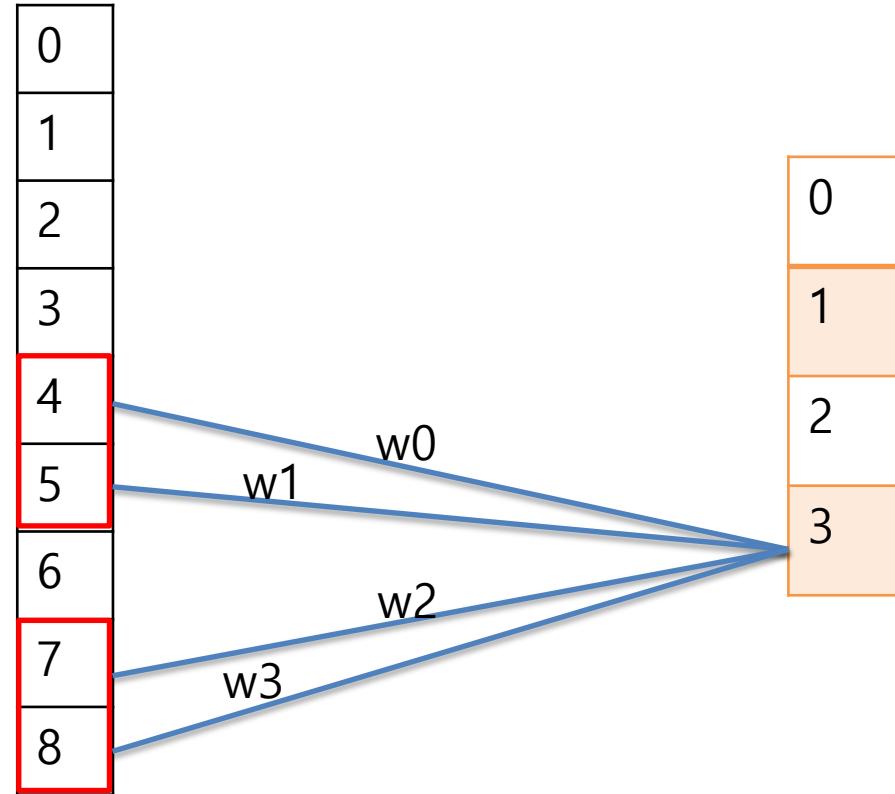
Basic Architecture

| | | |
|---|---|---|
| 0 | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |

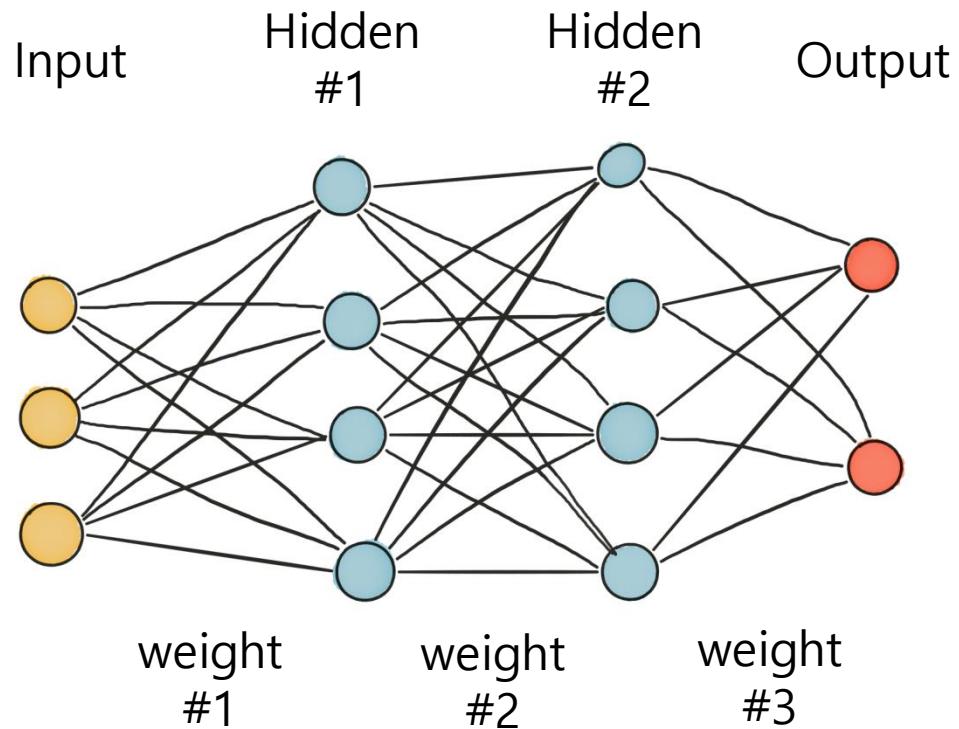


Basic Architecture

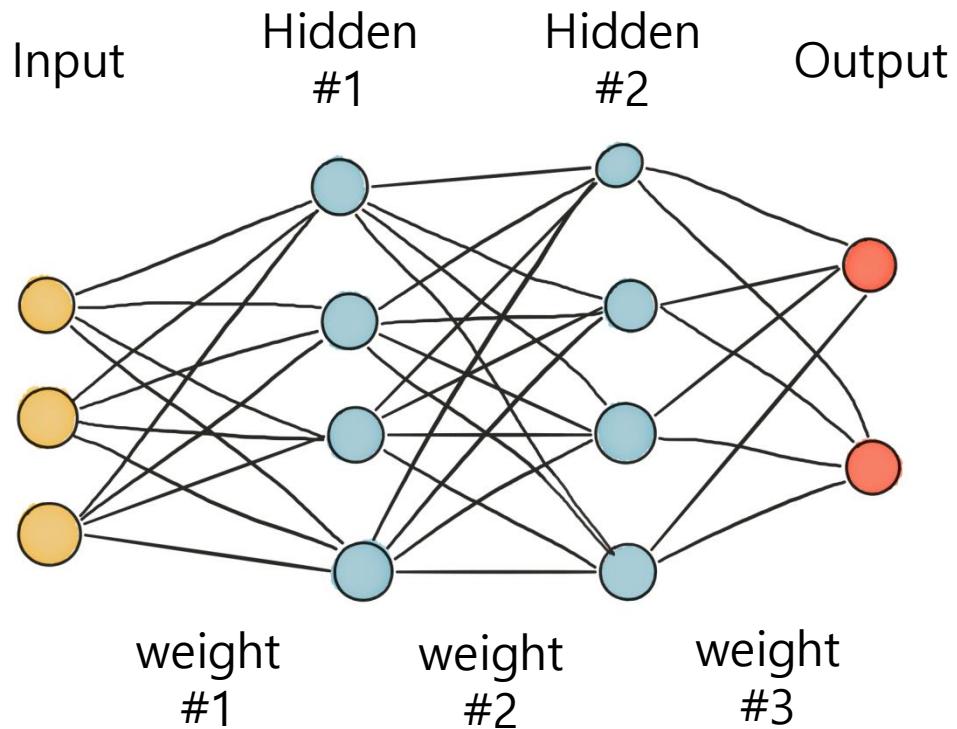
| | | |
|---|---|---|
| 0 | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |



Basic Architecture

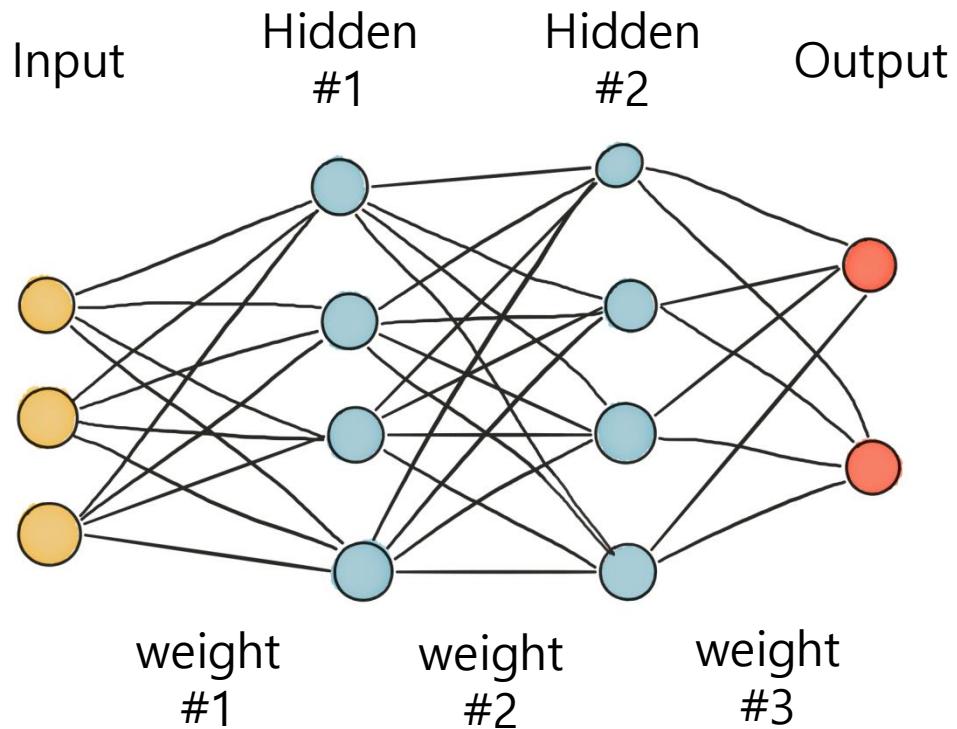


Basic Architecture



Convolutional Neural
Network 역시 일반
뉴럴넷과 유사한 구조

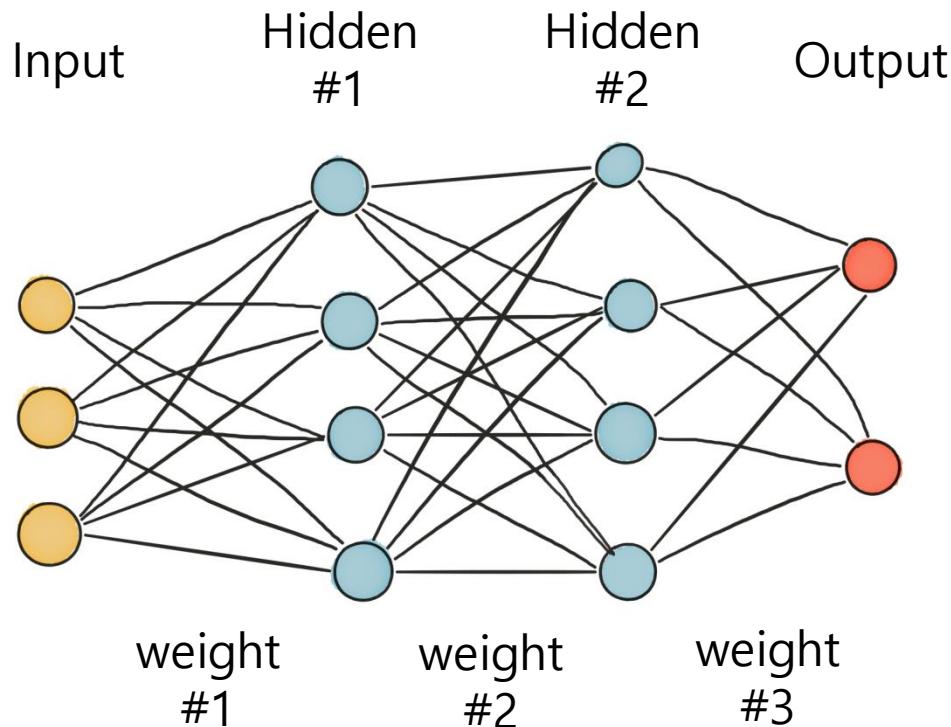
Basic Architecture



Convolutional Neural
Network 역시 일반
뉴럴넷과 유사한 구조



Basic Architecture



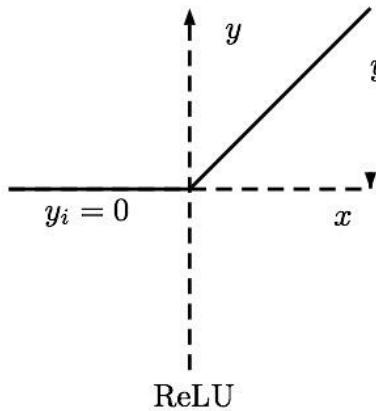
Convolutional Neural
Network 역시 일반
뉴럴넷과 유사한 구조



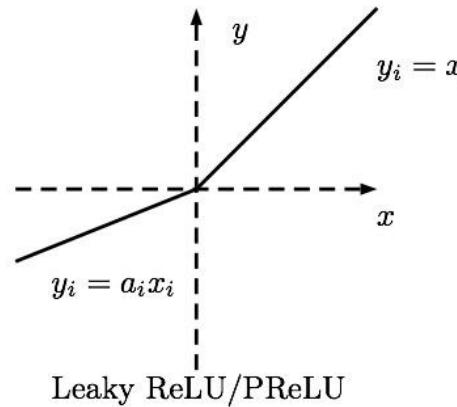
그렇다면 activation
function은?

Basic Architecture

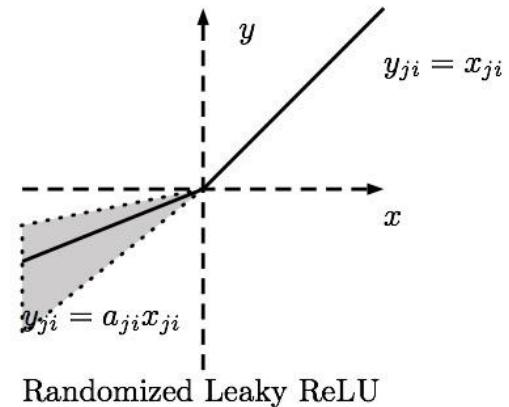
Rectified Linear Unit (ReLU)



$$f(x) = \max(0, x)$$

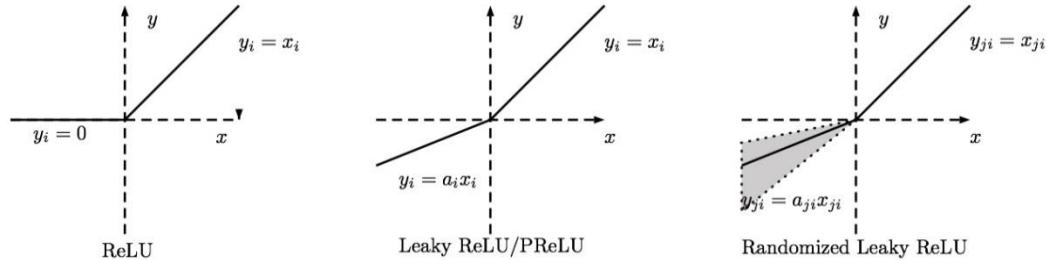


$$f(x) = \max(ax, x)$$



$$f(x) = \max(ax, x)$$

Basic Architecture



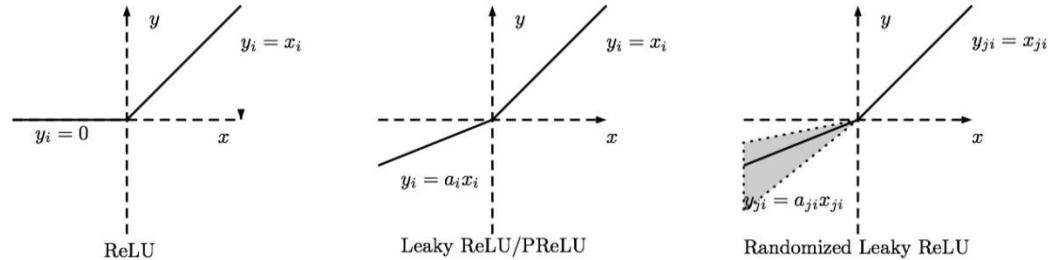
장점

- 학습이 빠름
- 연산면에서 효율적

단점

- Dying Neuron

Basic Architecture



장점

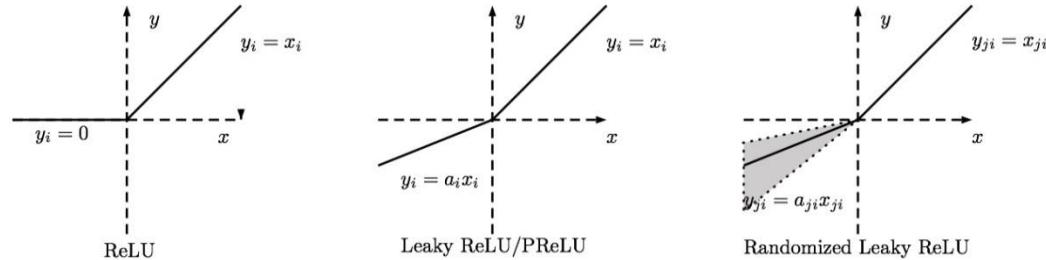
- 학습이 빠름
- 연산면에서 효율적

$y = x$ 를 미분하면 *gradient*가 1이니까
loss * gradient하면 sigmoid나 tanh보다
*gradient*가 크게 전달됨.

단점

- Dying Neuron

Basic Architecture



장점

- 학습이 빠름
- 연산면에서 효율적

$y = x$ 를 미분하면 *gradient*가 1이니까
loss * gradient하면 sigmoid나 tanh보다
*gradient*가 크게 전달됨.

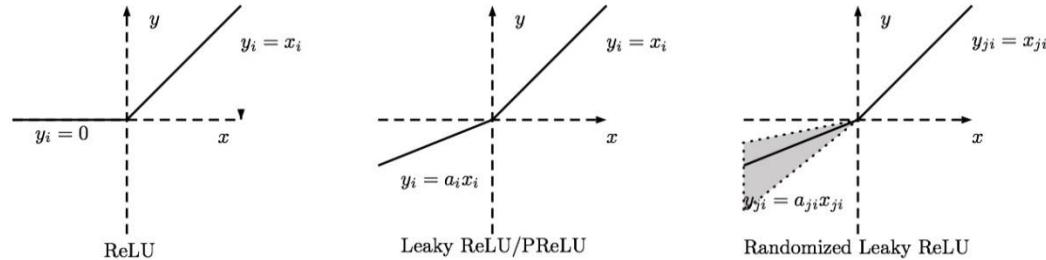
단점

- Dying Neuron

ex)

$$\begin{aligned} \text{sigmoid}'(x) &= \text{sigmoid}(x) * (1 - \text{sigmoid}(x)) \\ 0 \leq \text{sigmoid}(x) &\leq 1 \\ 0 \leq \text{sigmoid}'(x) &\leq 1 \end{aligned}$$

Basic Architecture



장점

- 학습이 빠름
- 연산면에서 효율적

$y = x$ 를 미분하면 *gradient*가 1이니까
loss * gradient하면 sigmoid나 tanh보다
*gradient*가 크게 전달됨.

단점

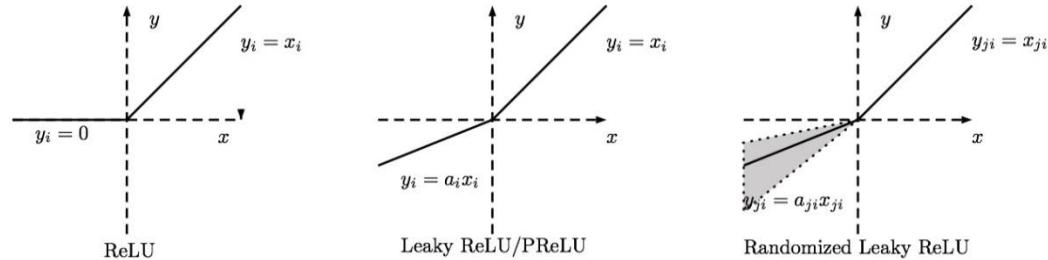
- Dying Neuron

ex)

$$\begin{aligned} \text{sigmoid}'(x) &= \text{sigmoid}(x) * (1 - \text{sigmoid}(x)) \\ 0 \leq \text{sigmoid}(x) &\leq 1 \\ 0 \leq \text{sigmoid}'(x) &\leq 1 \end{aligned}$$

그렇기 때문에 업데이트도 비교적 크다

Basic Architecture



장점

- 학습이 빠름
- 연산면에서 효율적

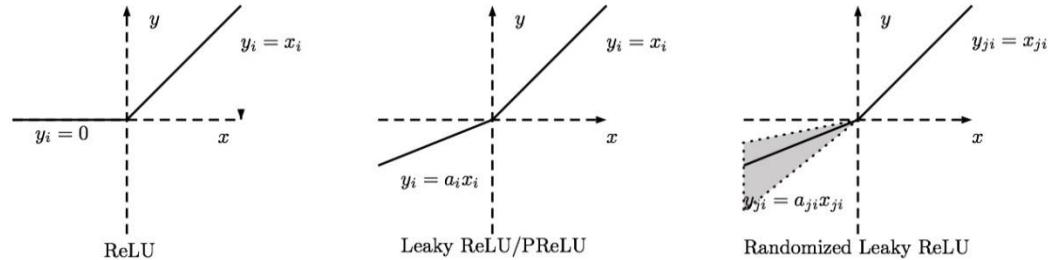


exponential이 들어간 연산보다는 ReLU가 더 간단하기 때문에 연산 속도가 빠름.

단점

- Dying Neuron

Basic Architecture



장점

- 학습이 빠름
- 연산면에서 효율적

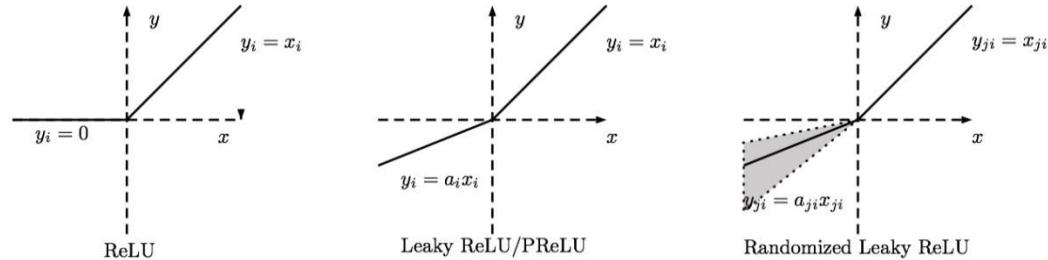
ReLU가 값이 0 이하인 값들은 무조건 gradient를 0로 만들어 버리고 back propagation이 일어나지 않음.

단점

- Dying Neuron



Basic Architecture



장점

- 학습이 빠름
- 연산면에서 효율적

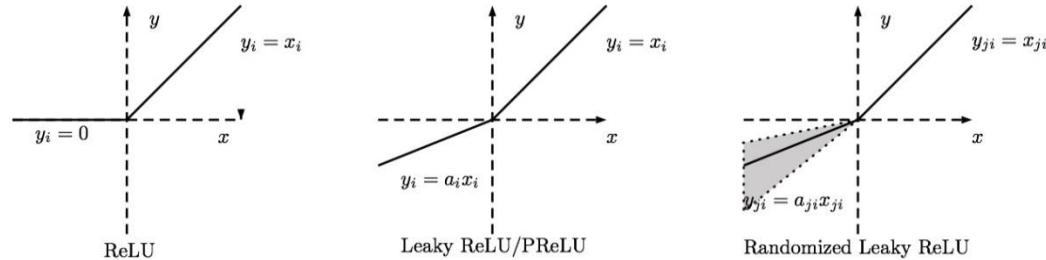
단점

- Dying Neuron

ReLU가 값이 0 이하인 값들은 무조건 gradient를 0로 만들어 버리고 back propagation이 일어나지 않음.

어쩌다 loss가 커지면 weight, bias가 크게 업데이트 됨.

Basic Architecture



장점

- 학습이 빠름
- 연산면에서 효율적

ReLU가 값이 0 이하인 값들은 무조건 gradient를 0로 만들어 버리고 back propagation이 일어나지 않음.

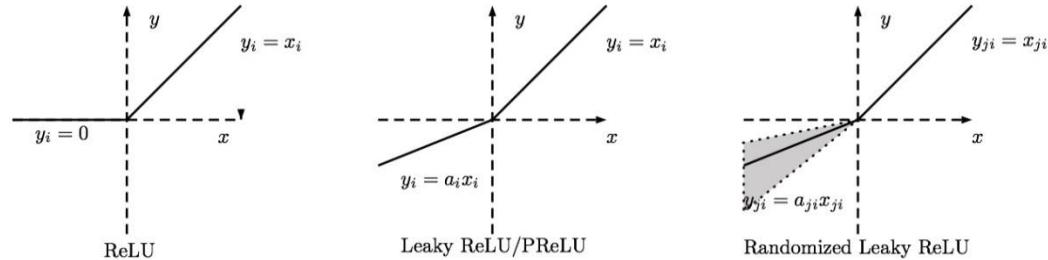
단점

- Dying Neuron

어쩌다 loss가 커지면 weight, bias가 크게 업데이트 됨.

ex) $y = -3x - 5$ 면 $-1 \sim 1$ 로 normalize된 input이 들어오면 무조건 0 이하. 영원히 업데이트 안됨.

Basic Architecture



장점

- 학습이 빠름
- 연산면에서 효율적

단점

- Dying Neuron

해결책으로 나온게 Leaky ReLU, PReLU
 $x < 0$ 일 때도 update가 이루어 지도록 개선

Convolution Layer



Convolution Layer

| | | |
|----|----|----|
| -1 | -1 | -1 |
| -1 | 8 | -1 |
| -1 | -1 | -1 |



outline

Convolution Layer

| | | |
|----|----|---|
| -2 | -1 | 0 |
| -1 | 1 | 1 |
| 0 | 1 | 2 |



emboss

Convolution Layer

| | | |
|------------|-----------|------------|
| 0.06 25 | 0.12 5 | 0.06 25 |
| 0.12 5 | 0.25 | 0.12 5 |
| 0.06 25 | 0.12 5 | 0.06 25 |



blur

Convolution Layer



위의 예시는 하나의 필터만 적용됐을 때

Convolution Layer



Hubel & Wiesel의 연구를 통해
단순한 필터로 뽑힌 feature들의
조합으로 complex feature를 뽑아낼
수 있다는 것을 배움

Convolution Layer



Hubel & Wiesel의 연구를 통해
단순한 필터로 뽑힌 feature들의
조합으로 complex feature를 뽑아낼
수 있다는 것을 배움



다양하고 많은 층의 Convolution
layer가 있으면 더 복잡한 모양도
구분할 수 있음.

Convolution Layer

$$\begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ \hline -1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & -1 \\ \hline -1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\ \hline -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ \hline -1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 & -1 \\ \hline -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ \hline -1 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ \hline -1 & -1 & 1 & -1 & -1 & -1 & 1 & -1 & -1 \\ \hline -1 & 1 & -1 & -1 & -1 & -1 & 1 & -1 & -1 \\ \hline -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ \hline \end{array} \otimes \begin{array}{|c|c|c|} \hline 1 & -1 & -1 \\ \hline -1 & 1 & -1 \\ \hline -1 & -1 & 1 \\ \hline \end{array} = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0.77 & -0.11 & 0.11 & 0.33 & 0.55 & -0.11 & 0.33 \\ \hline -0.11 & 1.00 & -0.11 & 0.33 & -0.11 & 0.11 & -0.11 \\ \hline 0.11 & -0.11 & 1.00 & -0.33 & 0.11 & -0.11 & 0.55 \\ \hline 0.33 & 0.33 & -0.33 & 0.55 & -0.33 & 0.33 & 0.33 \\ \hline 0.55 & -0.11 & 0.11 & -0.33 & 1.00 & -0.11 & 0.11 \\ \hline -0.11 & 0.11 & -0.11 & 0.33 & -0.11 & 1.00 & -0.11 \\ \hline 0.33 & -0.11 & 0.55 & 0.33 & 0.11 & -0.11 & 0.77 \\ \hline \end{array}$$

Convolution Layer

The diagram shows a convolution operation. On the left, a 9x9 input image is shown with values ranging from -1 to 1. A 3x3 kernel is applied to it, indicated by a circled multiplication symbol (\otimes). The result is an output image on the right, also 9x9, with values ranging from -0.77 to 0.77. The output image is labeled with a double equals sign (=) followed by its data.

| | | | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|--|--|
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 | | |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 | | |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 | | |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 | | |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 | | |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 | | |
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 | | |
| | | | | | | | | |
| | | | | | | | | |

9x9 이미지의 경우

3x3 커널을

1번 적용: 7x7

2번 적용: 5x5

3번 적용: 3x3

4번 적용: 1x1

Convolution Layer

The diagram shows a 9x9 input matrix with values ranging from -1 to 1. A 3x3 kernel matrix is applied to it. The result is a 7x7 output matrix where each value is the sum of the products of the corresponding input and kernel elements. The resulting output matrix has values ranging from 0.77 to -0.77.

| | | | | | | | | |
|----|----|----|----|----|----|----|----|----|
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 |
| -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 |
| -1 | 1 | -1 | -1 | -1 | -1 | 1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |

\otimes =

| | | |
|----|----|----|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 |
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |

9x9 이미지의 경우
3x3 커널을
1번 적용: 7x7
2번 적용: 5x5
3번 적용: 3x3
4번 적용: 1x1



단순한 X자 모양이면
모르겠지만 만약 복잡한
모양을 구분해야 한다면
더 많은 층이 필요하지
않을까?

Convolution Layer

Convolution Layer

| | | | | | | | | | | | | |
|---|----|----|----|----|----|----|----|----|----|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 0 | 0 | 0 |
| 0 | -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 | 0 | 0 | 0 |
| 0 | -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 | 0 | 0 | 0 |
| 0 | -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 | 0 | 0 | 0 |
| 0 | -1 | -1 | -1 | -1 | 1 | 1 | -1 | -1 | -1 | 0 | 0 | 0 |
| 0 | -1 | -1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 | 0 | 0 | 0 |
| 0 | -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 | 0 | 0 | 0 |
| 0 | -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 | 0 | 0 | 0 |
| 0 | -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 | 0 | 0 | 0 |
| 0 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 |
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |

padding = 1

Convolution Layer

| | | | | | | | | | | | |
|---|----|----|----|----|----|----|----|----|----|----|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 0 |
| 0 | -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 | 0 | 0 |
| 0 | -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 | 0 | 0 |
| 0 | -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 | 0 | 0 |
| 0 | -1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 | -1 | 0 | 0 |
| 0 | -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 | 0 | 0 |
| 0 | -1 | -1 | 1 | -1 | -1 | 1 | -1 | -1 | -1 | 0 | 0 |
| 0 | -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 | 0 | 0 |
| 0 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | | |
|----|----|----|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 |
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |

padding = 1

Convolution Layer

$$(-1+1+1+1)/9=0.2222$$

| | | | | | | | | | | | |
|---|----|----|----|----|----|----|----|----|----|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 0 | 0 |
| 0 | -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 | 0 | 0 |
| 0 | -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 | 0 | 0 |
| 0 | -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 | 0 | 0 |
| 0 | -1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 | -1 | 0 | 0 |
| 0 | -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 | 0 | 0 |
| 0 | -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 | 0 | 0 |
| 0 | -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 | 0 | 0 |
| 0 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | | |
|----|----|----|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 |
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |

padding = 1

Convolution Layer

$$(-1+1+1+1)/9=0.2222$$

| | | | | | | | | | | | |
|---|----|----|----|----|----|----|----|----|----|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 0 | 0 |
| 0 | -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 | 0 | 0 |
| 0 | -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 | 0 | 0 |
| 0 | -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 | 0 | 0 |
| 0 | -1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 | -1 | 0 | 0 |
| 0 | -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 | 0 | 0 |
| 0 | -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 | 0 | 0 |
| 0 | -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 | 0 | 0 |
| 0 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | | |
|----|----|----|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

| | | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|--|
| 0.22 | | | | | | | |
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 | |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 | |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 | |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 | |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 | |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 | |
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 | |

padding = 1

Convolution Layer

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 |
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |

Image size를 I , kernel size를 K , stride를 S , padding을 P 라고 하면 output size O 는 아래의 식과 같음.

Convolution Layer

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 |
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |

Image size를 I , kernel size를 K , stride를 S , padding을 P 라고 하면 output size O 는 아래의 식과 같음.

$$O = \text{floor}\left(\frac{I - K + 2P}{S} + 1\right)$$

Convolution Layer

이미 충분한 정보를 뽑아냈다면?



1800

3600

Convolution Layer

이미 충분한 정보를 뽑아냈다면?

3600



1800

3x3 Kernel로 특성을 300단계 뽑아내도 3000×1200
이렇게 큰 이미지에 3x3 필터를 적용하면 연산량도
너무 많고 뽑힌 필터로 결과값을 내는 것도 비효율적

Convolution Layer

이미 충분한 정보를 뽑아냈다면?

3600



1800

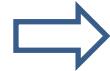
그래서 필요한 게 Subsampling(≈Downsampling)

Convolution Layer



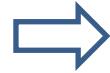
이미지에서 충분한 정보를 뽑아냈다면 해당 정보를 전부 전달하는 것이 아니라 중요한 정보만 또는 평균적 정보를 전달해도 결정을 내릴 수 있다.

Convolution Layer



이미지에서 충분한 정보를 뽑아냈다면 해당 정보를 전부 전달하는 것이 아니라 중요한 정보만 또는 평균적 정보를 전달해도 결정을 내릴 수 있다.

Convolution Layer



이미지에서 충분한 정보를 뽑아냈다면 해당 정보를 전부 전달하는 것이 아니라 중요한 정보만 또는 평균적 정보를 전달해도 결정을 내릴 수 있다.

Convolution Layer



이미지에서 충분한 정보를 뽑아냈다면 해당 정보를 전부 전달하는 것이 아니라 중요한 정보만 또는 평균적 정보를 전달해도 결정을 내릴 수 있다.

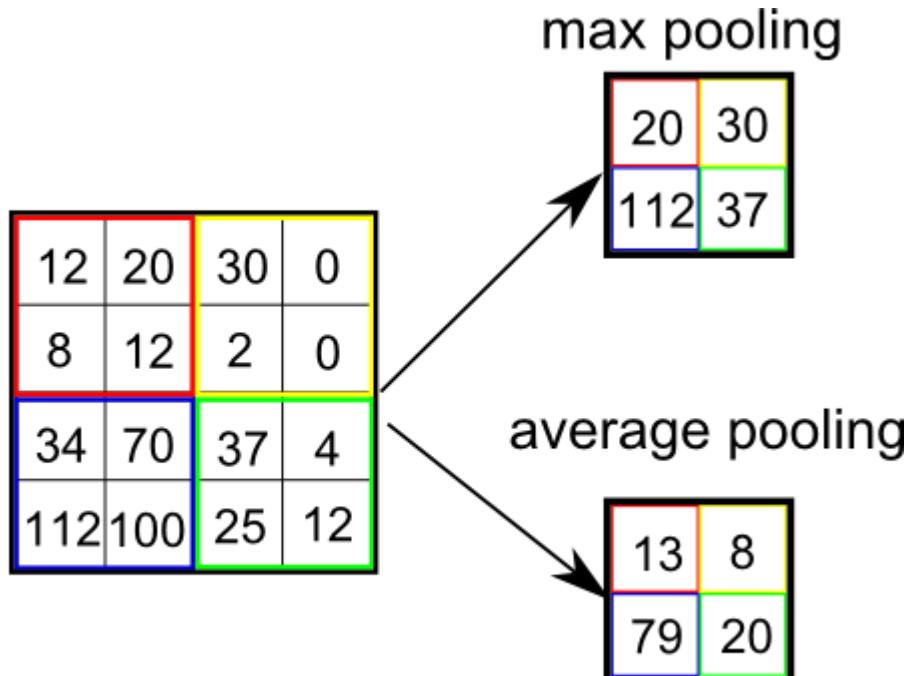
정보의 압축 및 축소

Convolution Layer

어떻게 축소시킬 것인가?

Convolution Layer

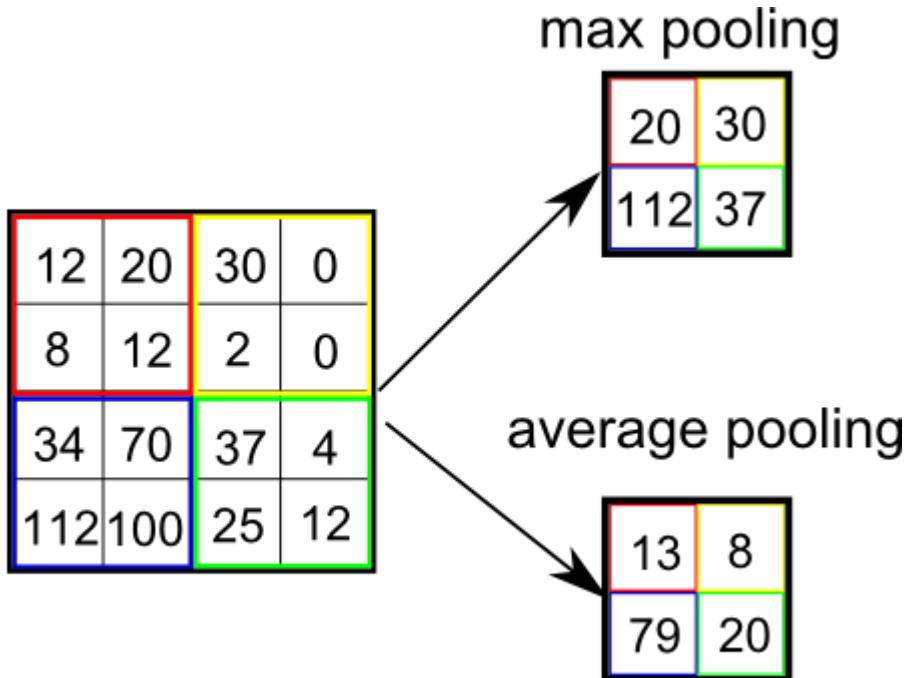
어떻게 축소시킬 것인가?



(출처: <https://stackoverflow.com/questions/44287965/trying-to-confirm-average-pooling-is-equal-to-dropping-high-frequency-fourier-co>)

Convolution Layer

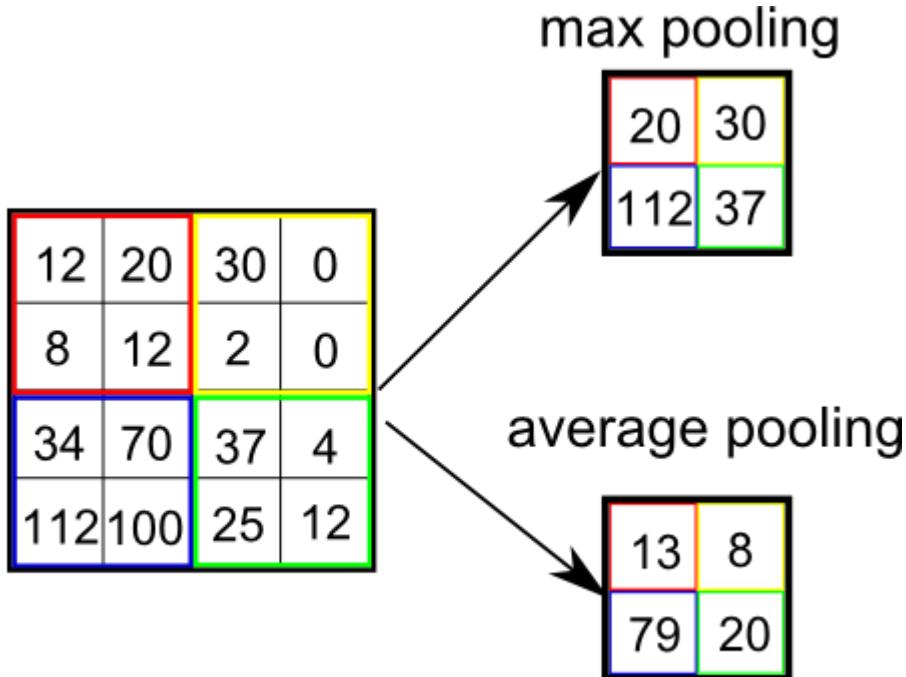
어떻게 축소시킬 것인가?



2x2 grid, stride 2로
최대값을 뽑아내서 축소
-> 강한 자극만 기억

Convolution Layer

어떻게 축소시킬 것인가?



2x2 grid, stride 2로
최대값을 뽑아내서 축소
-> 강한 자극만 기억

2x2 grid, stride 2로
평균값을 뽑아내서 축소
-> 평균적 자극을 기억

Basic Architecture

Convolution

- Kernel
 - Stride
 - Padding
-

Basic Architecture

Convolution

- Kernel
- Stride
- Padding

Subsampling

- Max Pooling
- Average Pooling

Basic Architecture

Convolution

- Kernel
- Stride
- Padding

Subsampling

- Max Pooling
- Average Pooling

Activation Function

- ReLU
- Leaky ReLU

Basic Architecture

Convolution

- Kernel
- Stride
- Padding

Subsampling

- Max Pooling
- Average Pooling

Activation Function

- ReLU
- Leaky ReLU

하나하나는 대충 알겠는데
대체 어떤 순서로 배열해야 할까?

Basic Architecture

conv

Basic Architecture

$conv * \alpha$



Basic Architecture

$conv * \alpha + Activation$



Basic Architecture

$$(conv * \alpha + Activation) * \beta$$

Basic Architecture

$(conv * \alpha + Activation) * \beta + Pooling$

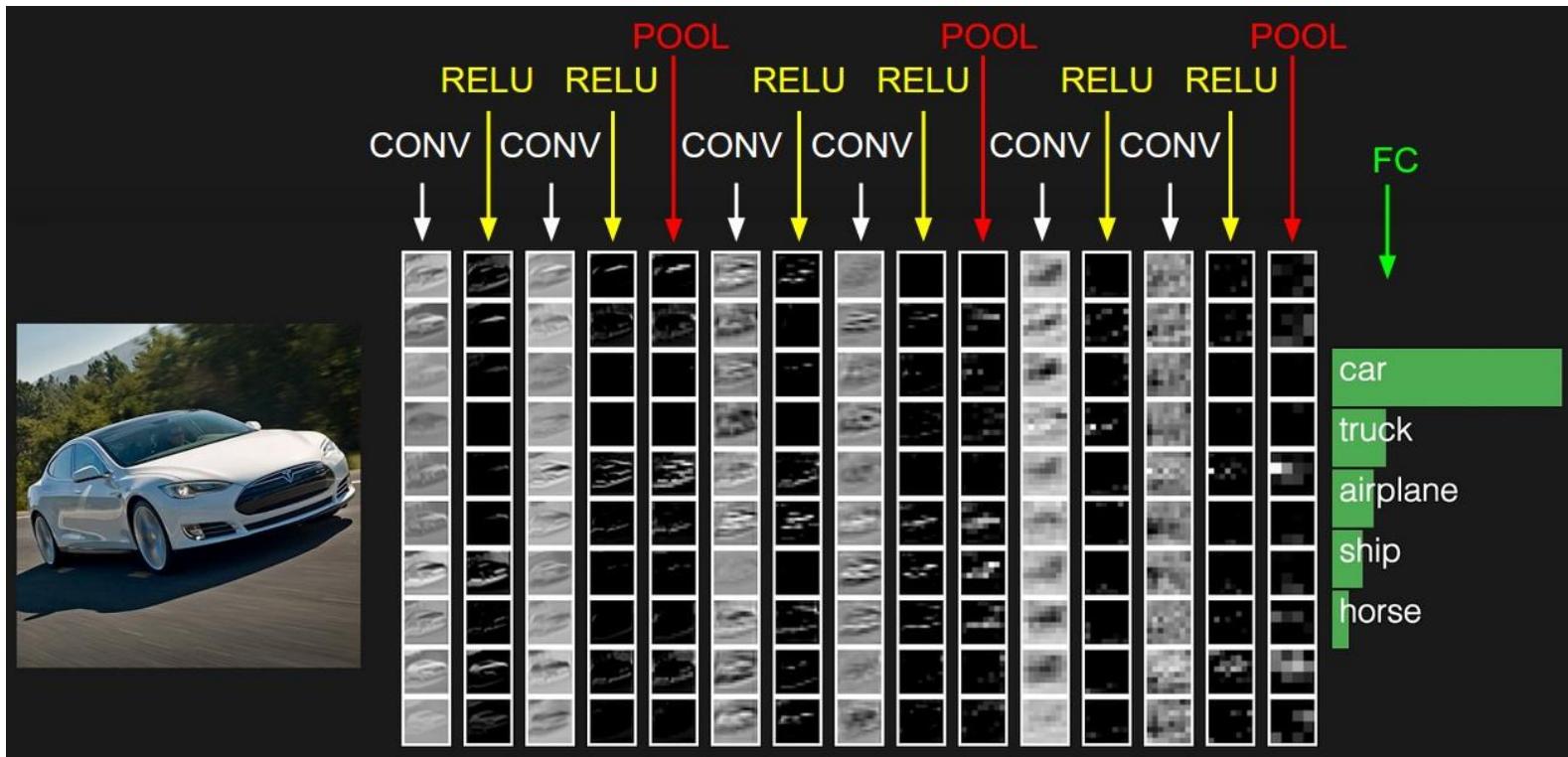
Basic Architecture

$$[(conv * \alpha + Activation) * \beta + Pooling] * \gamma$$

Basic Architecture

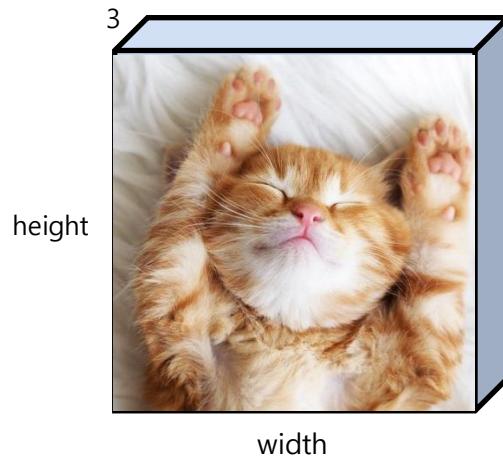
$[(conv * \alpha + Activation) * \beta + Pooling] * \gamma + Fully\ Connected\ Layer$

Basic Architecture

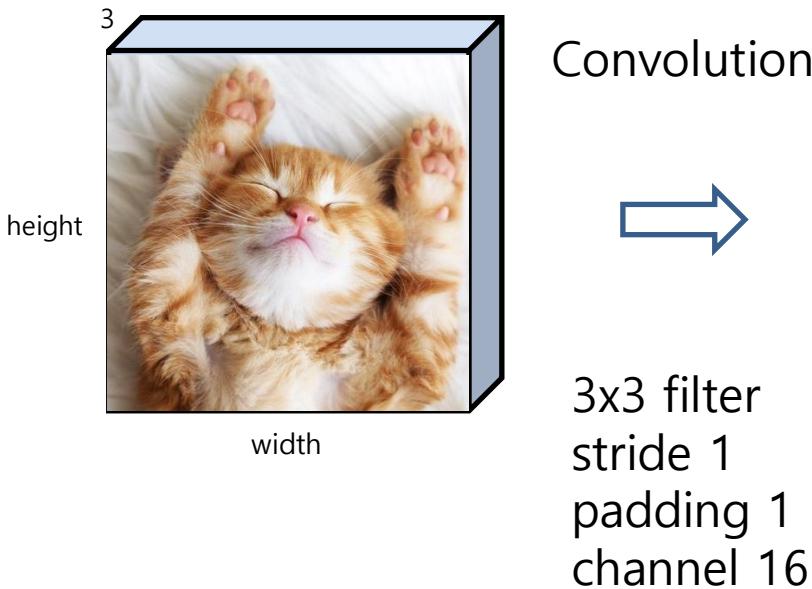


Basic Architecture

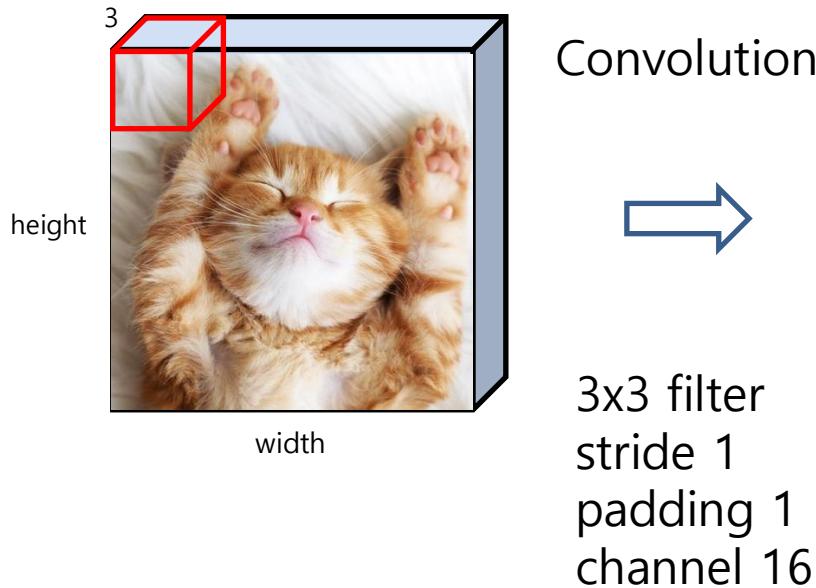
CNN 연산의 3차원적 이해



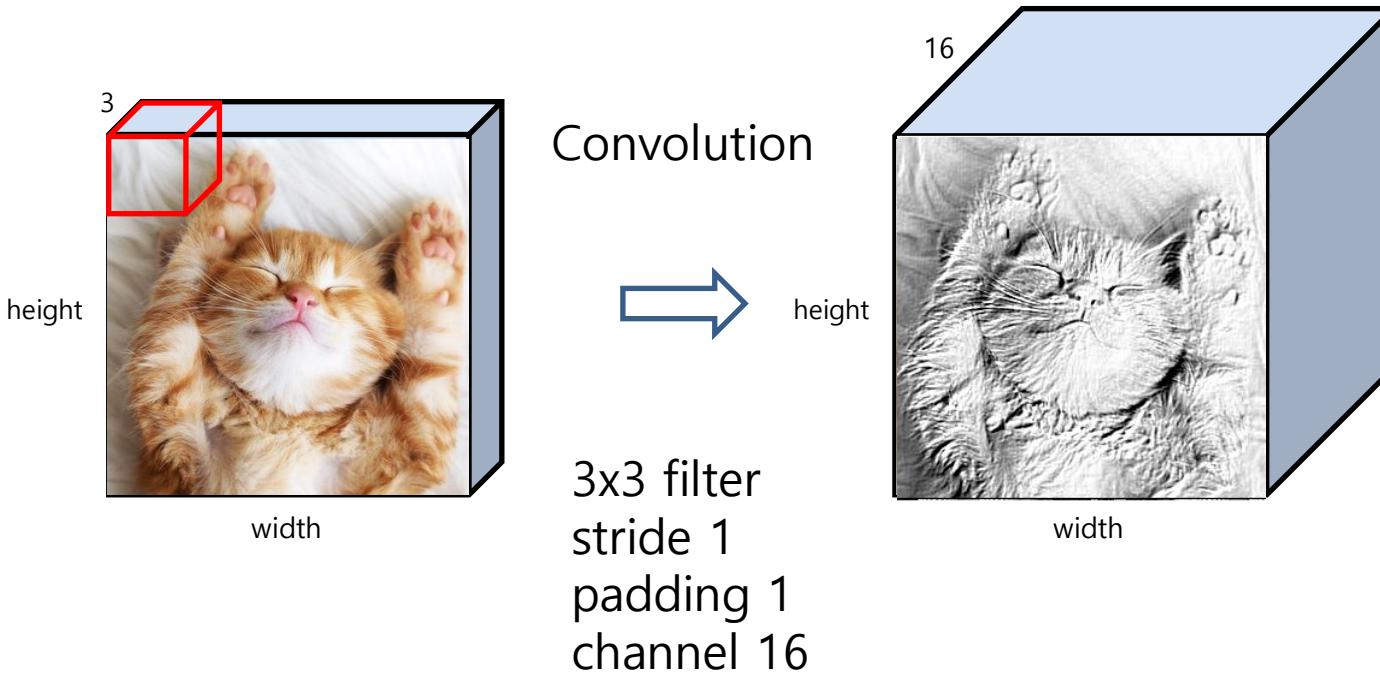
Basic Architecture



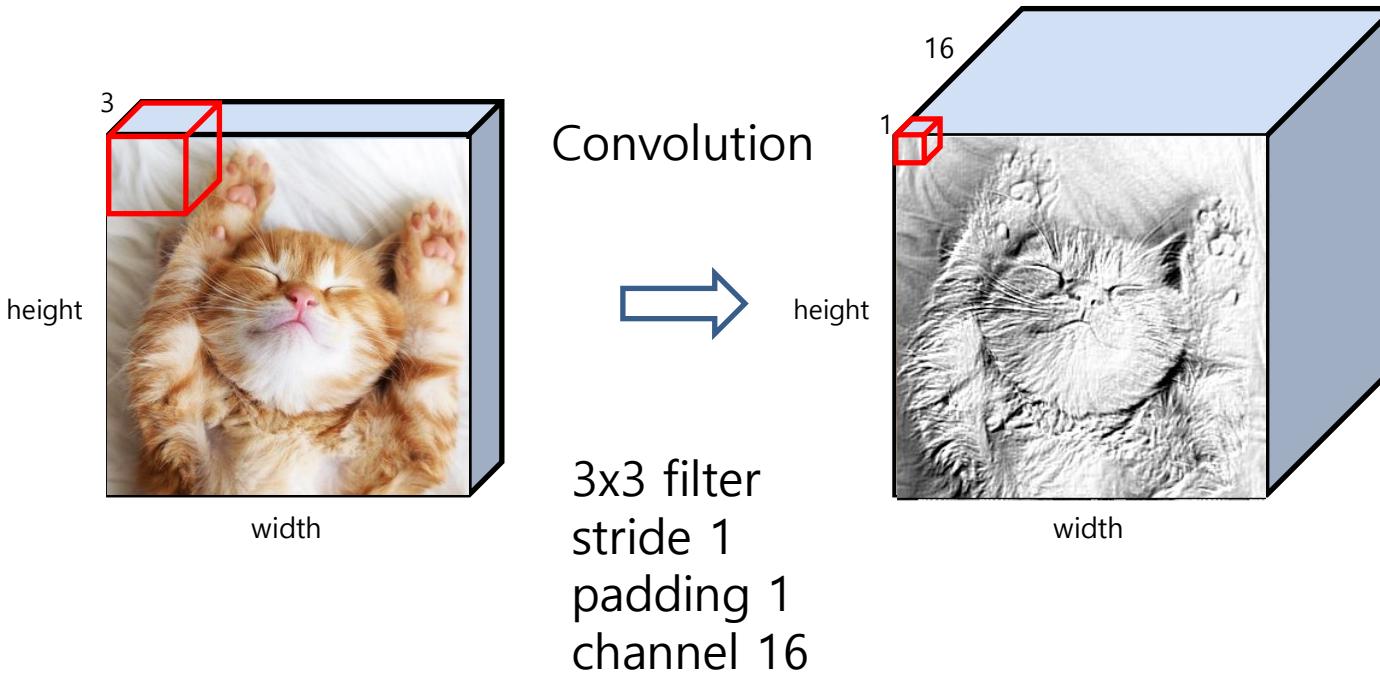
Basic Architecture



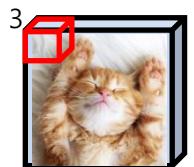
Basic Architecture



Basic Architecture



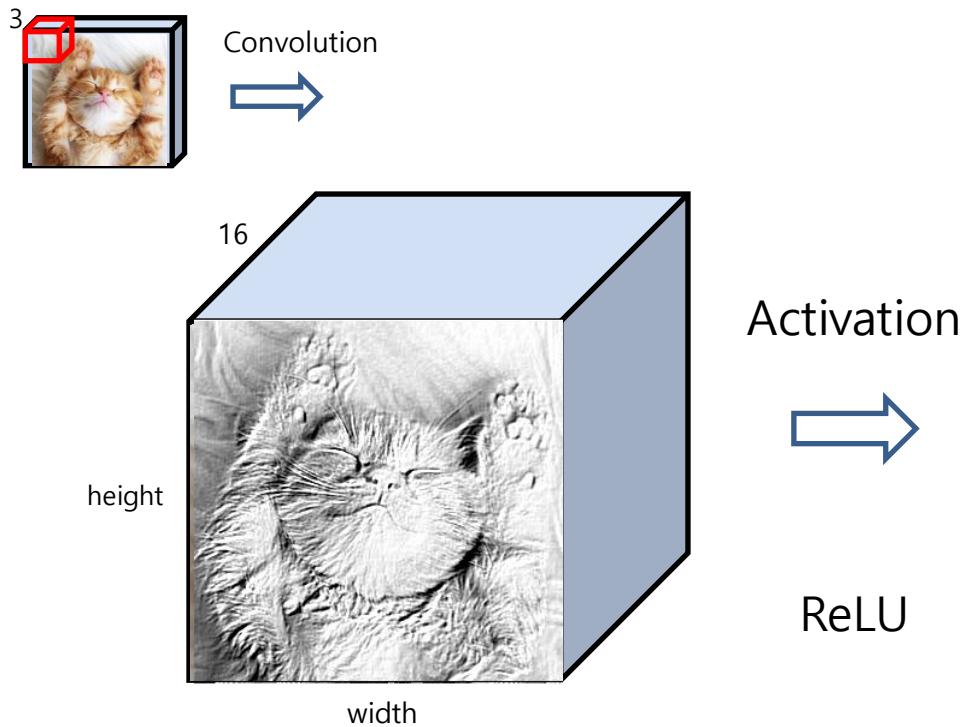
Basic Architecture



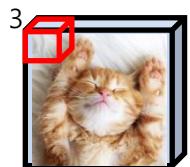
Convolution
→



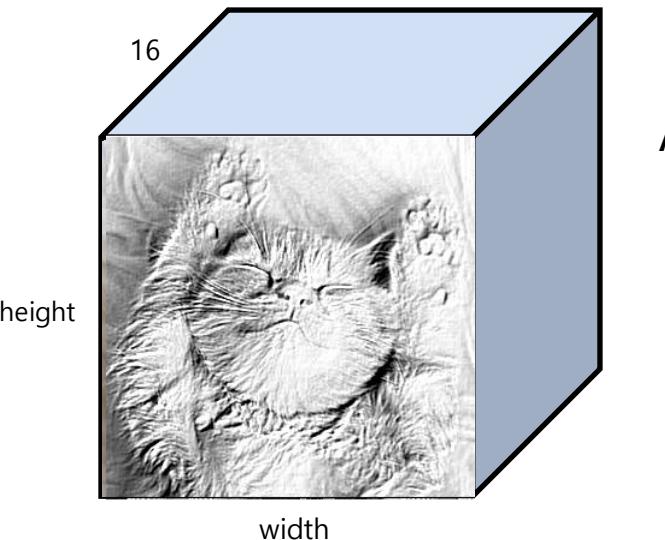
Basic Architecture



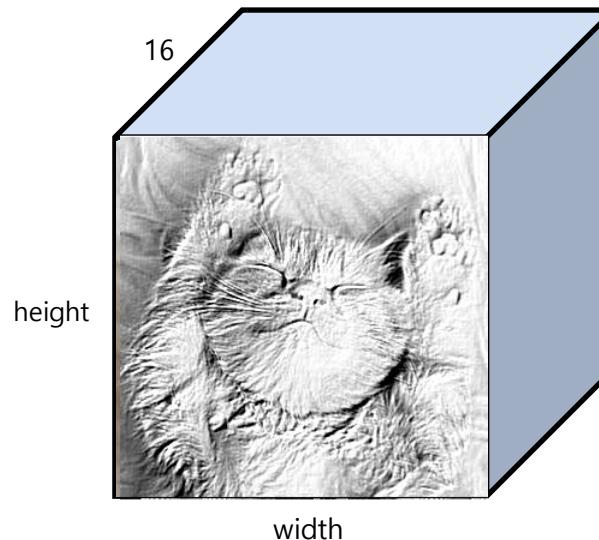
Basic Architecture



Convolution
→

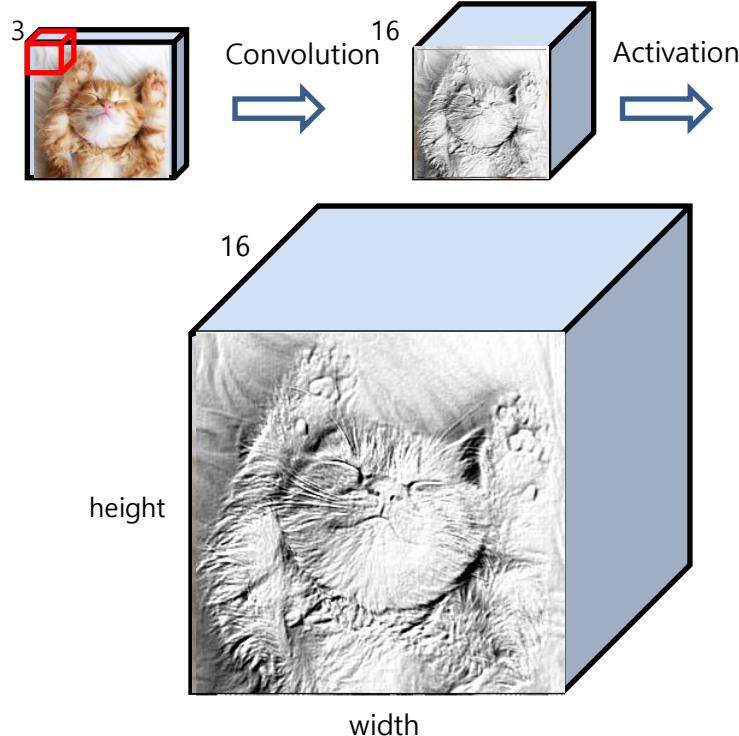


Activation
→

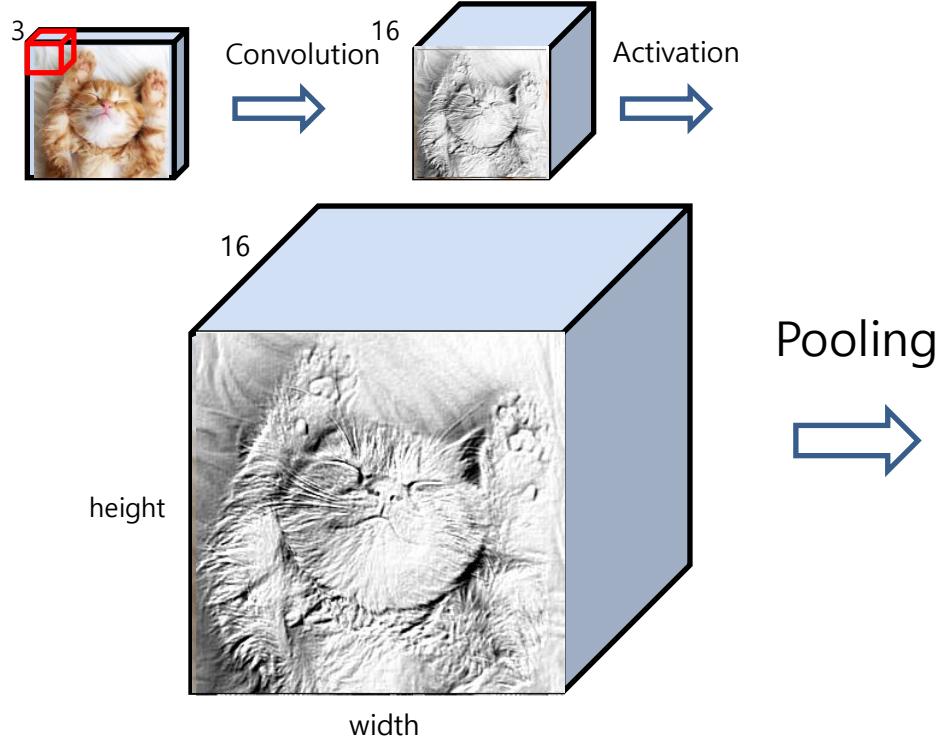


ReLU

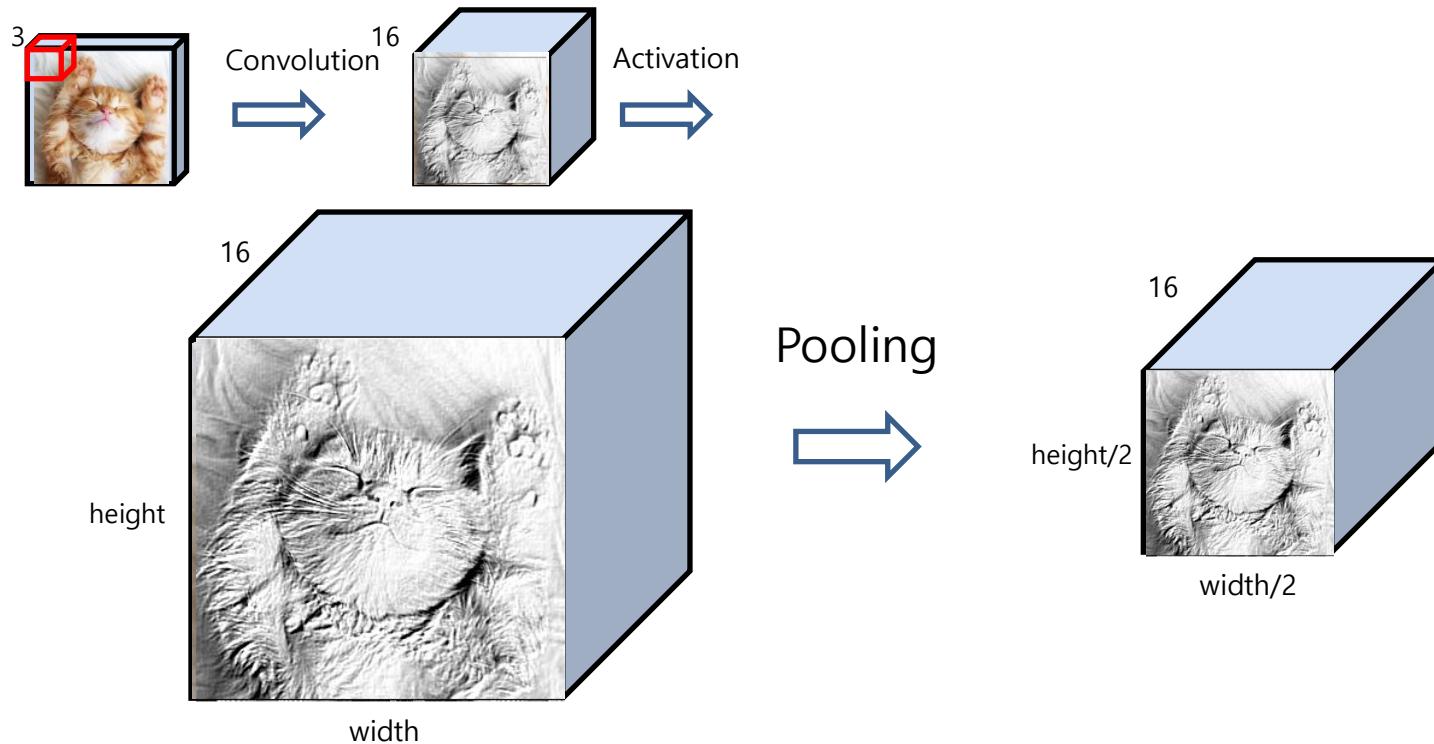
Basic Architecture



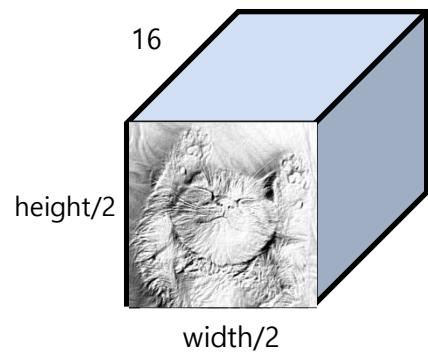
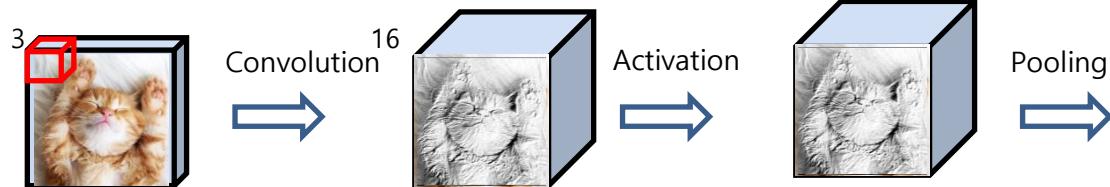
Basic Architecture



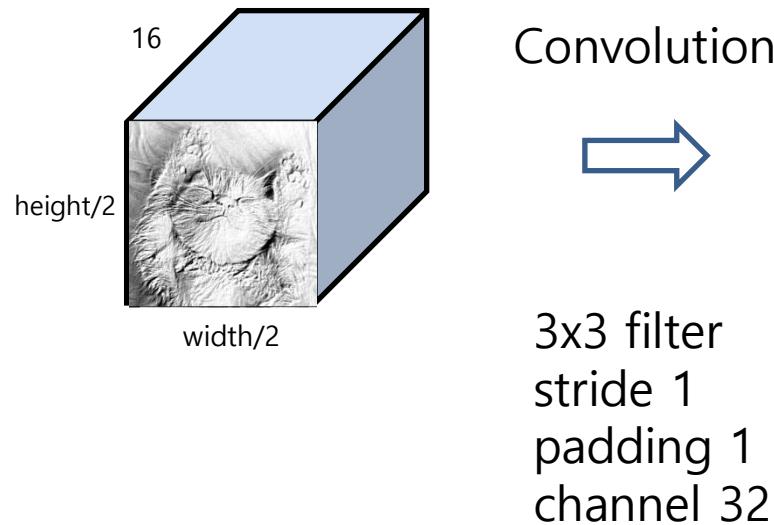
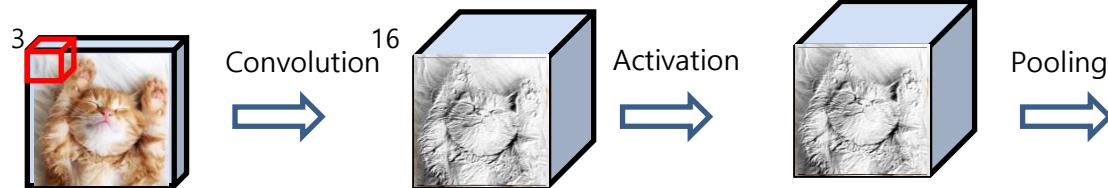
Basic Architecture



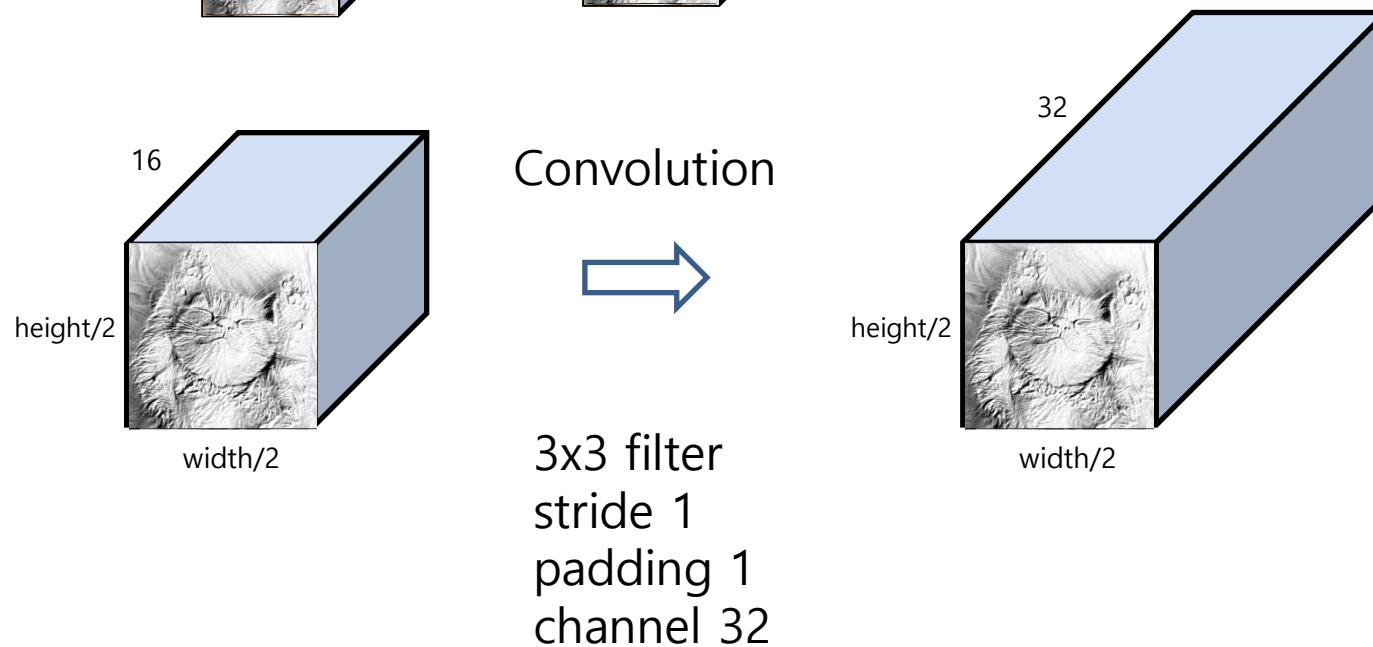
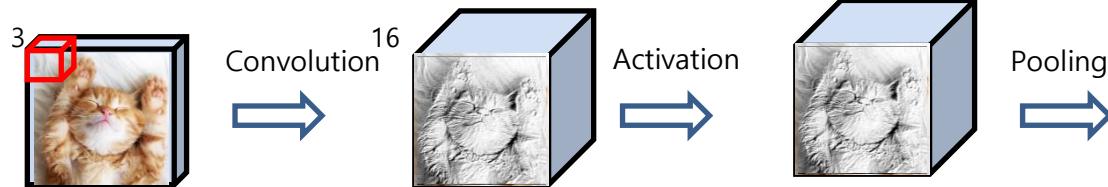
Basic Architecture



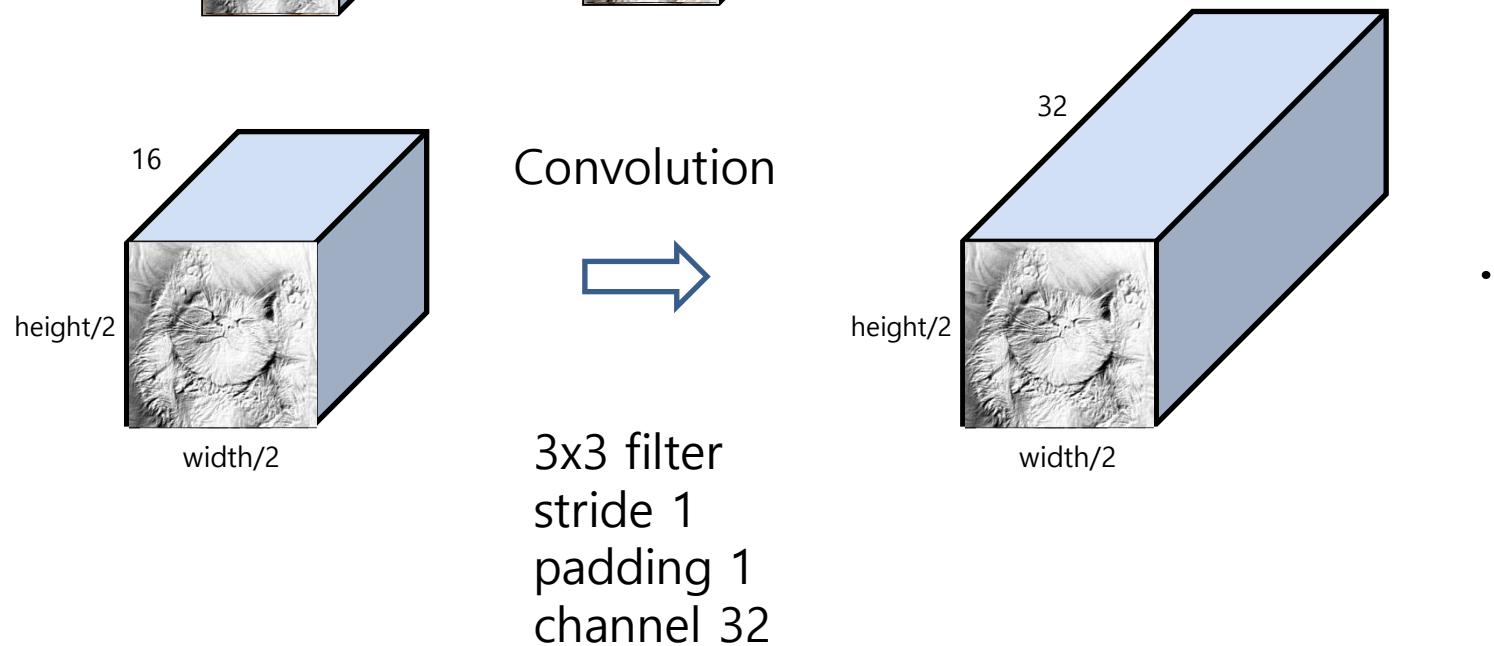
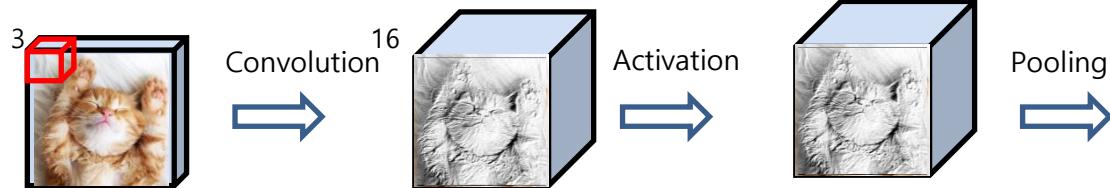
Basic Architecture



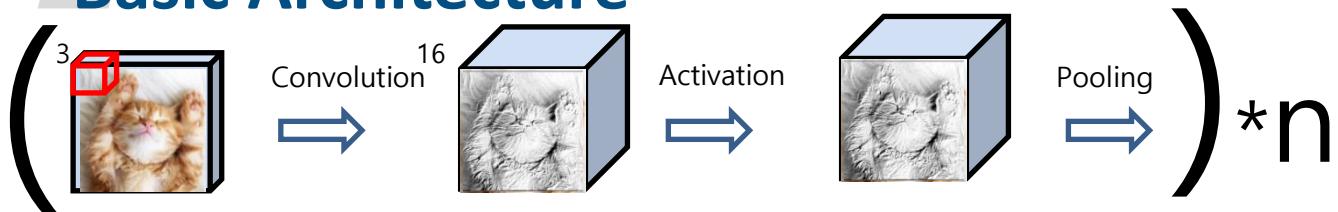
Basic Architecture



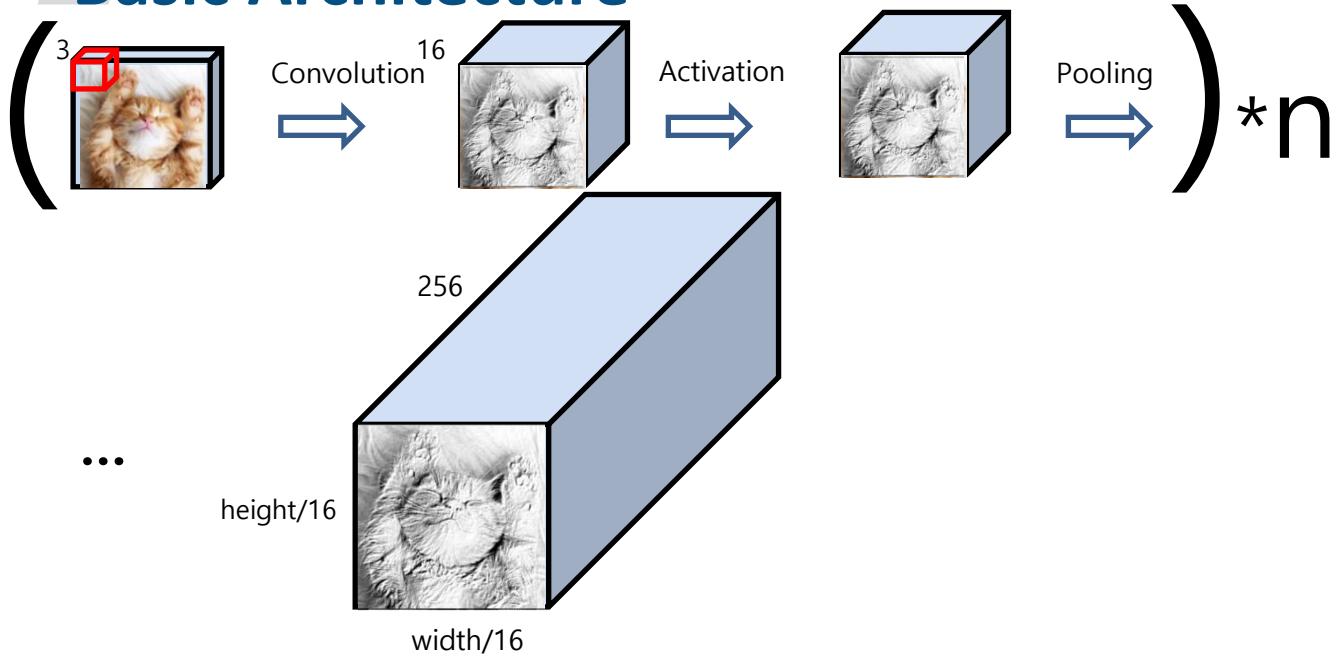
Basic Architecture



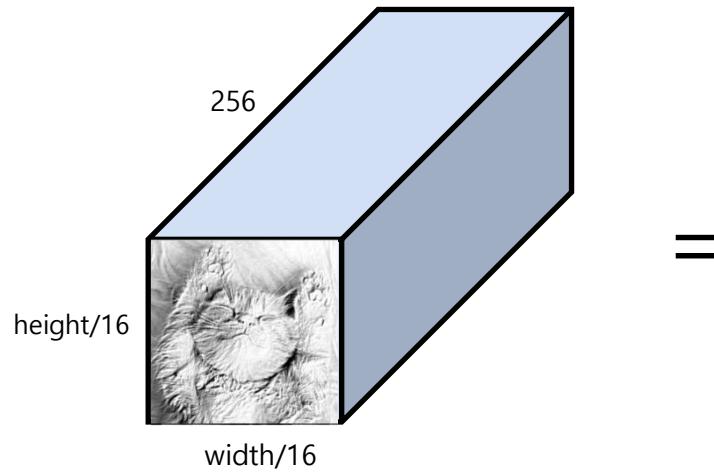
Basic Architecture



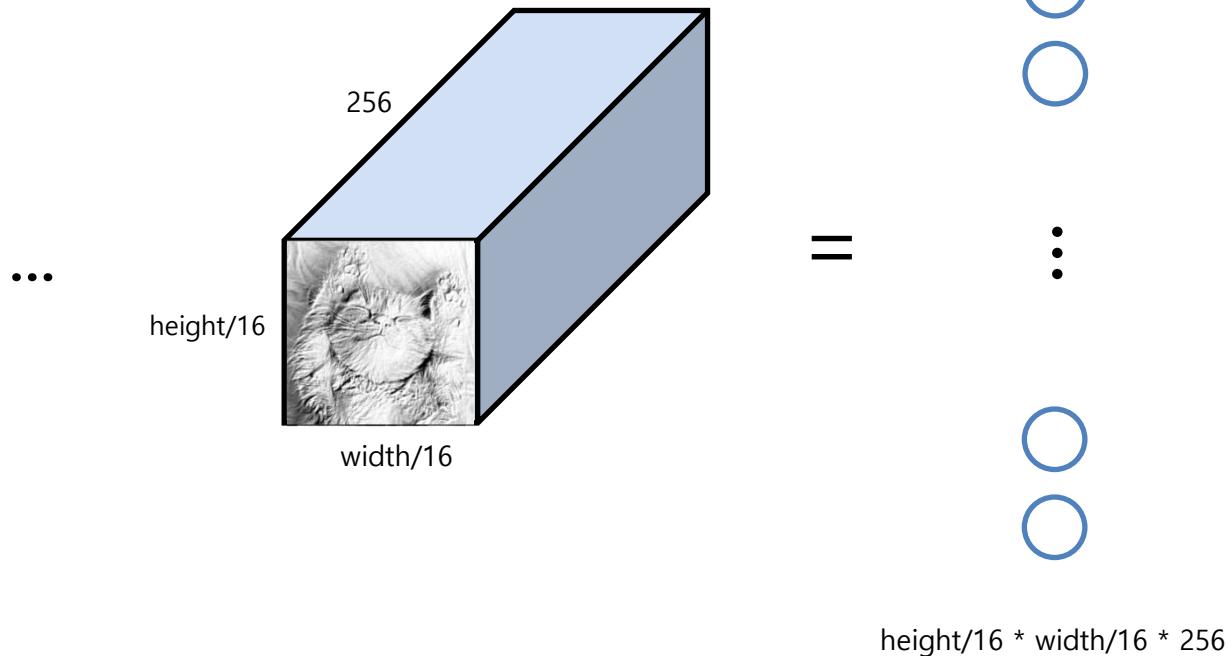
Basic Architecture



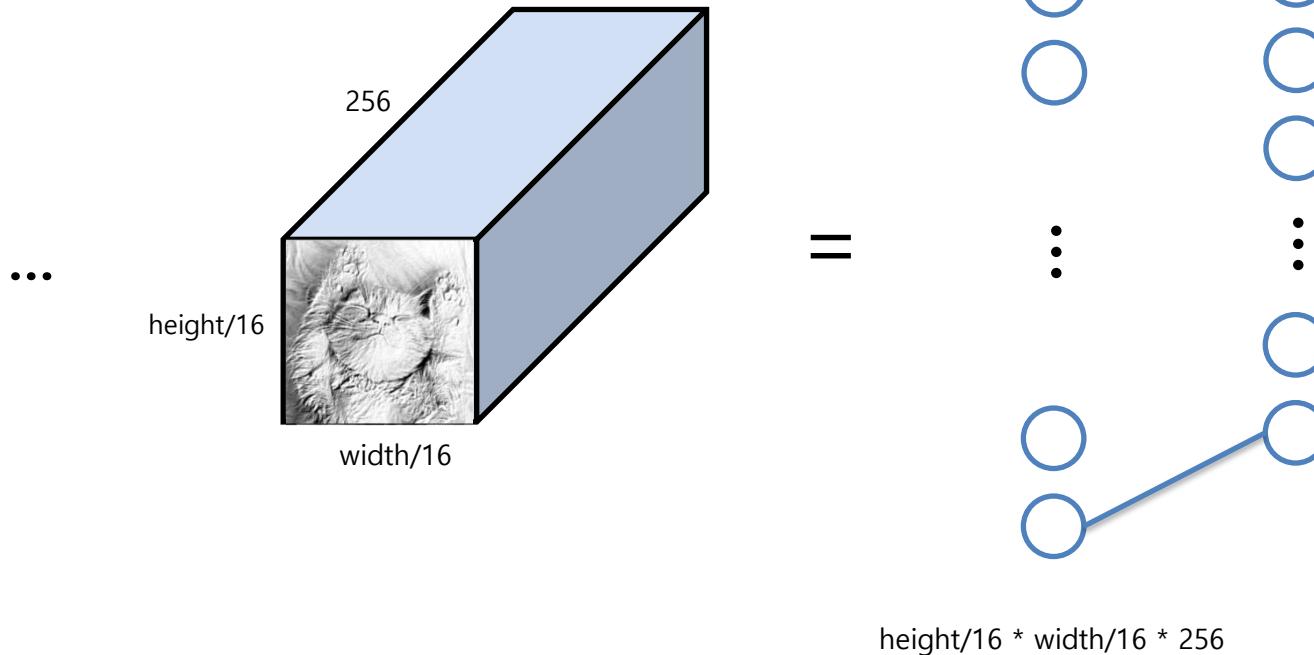
Basic Architecture



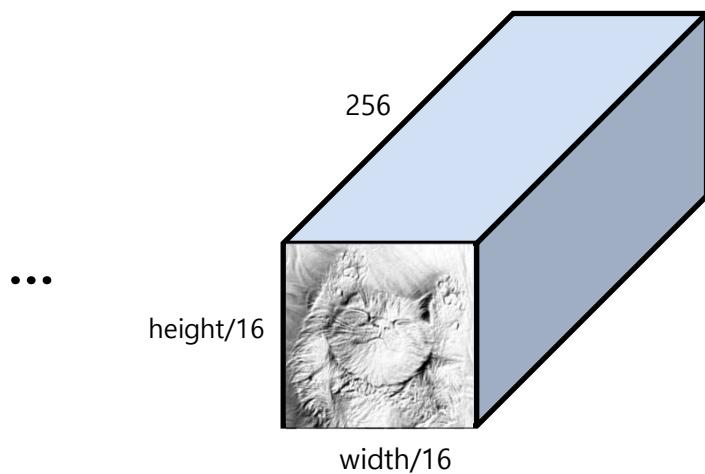
Basic Architecture



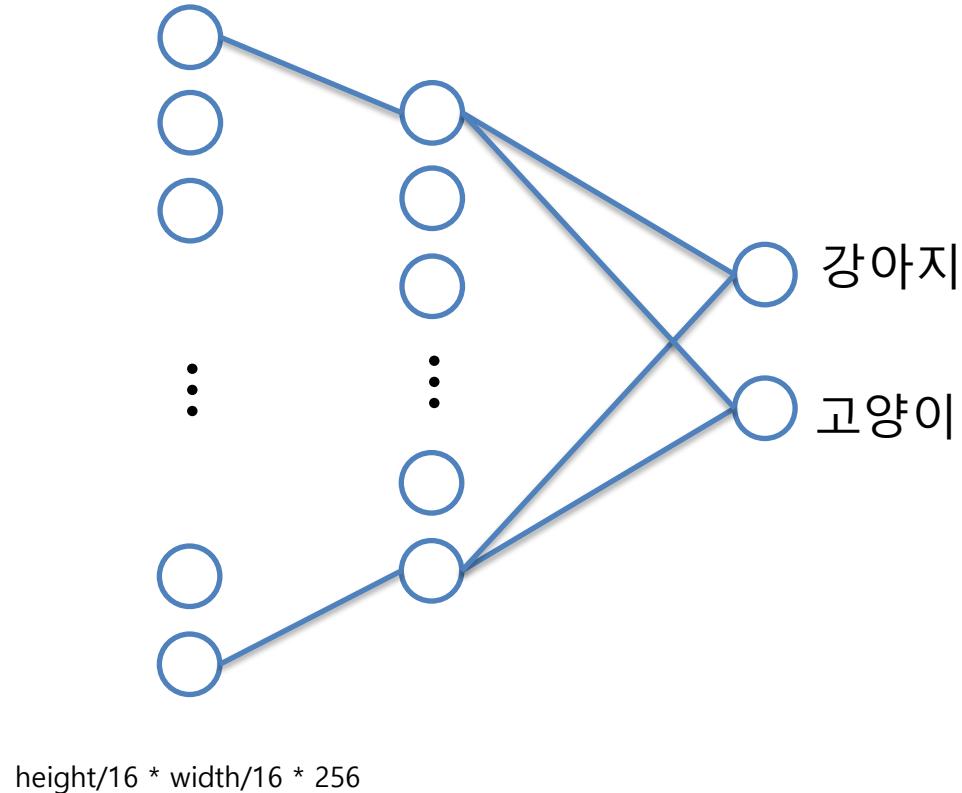
Basic Architecture



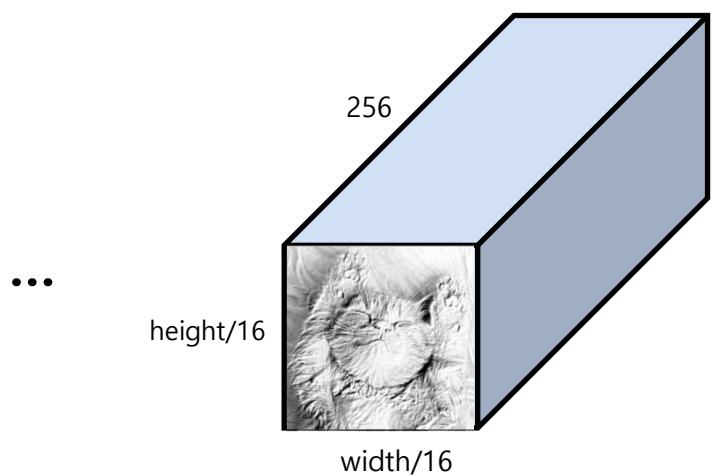
Basic Architecture



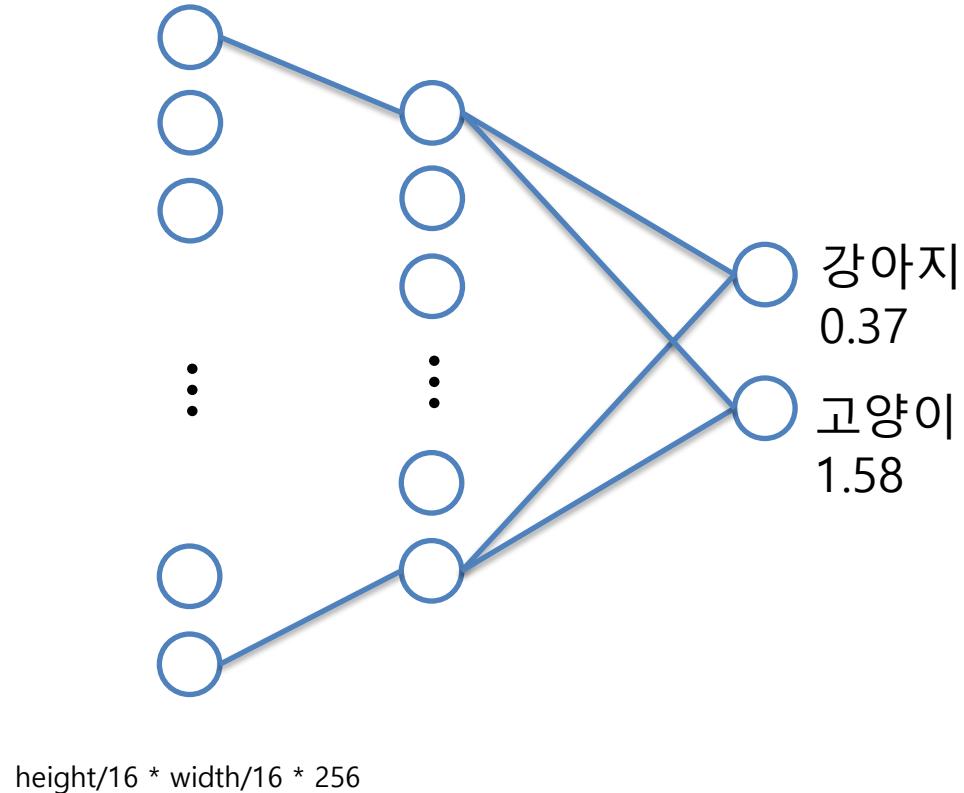
=



Basic Architecture

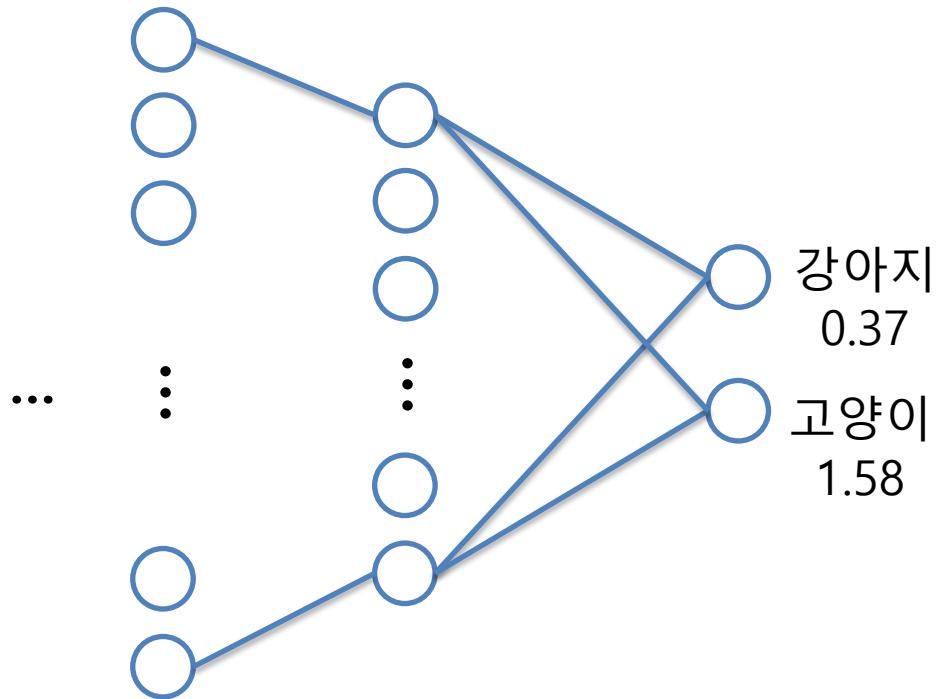


=

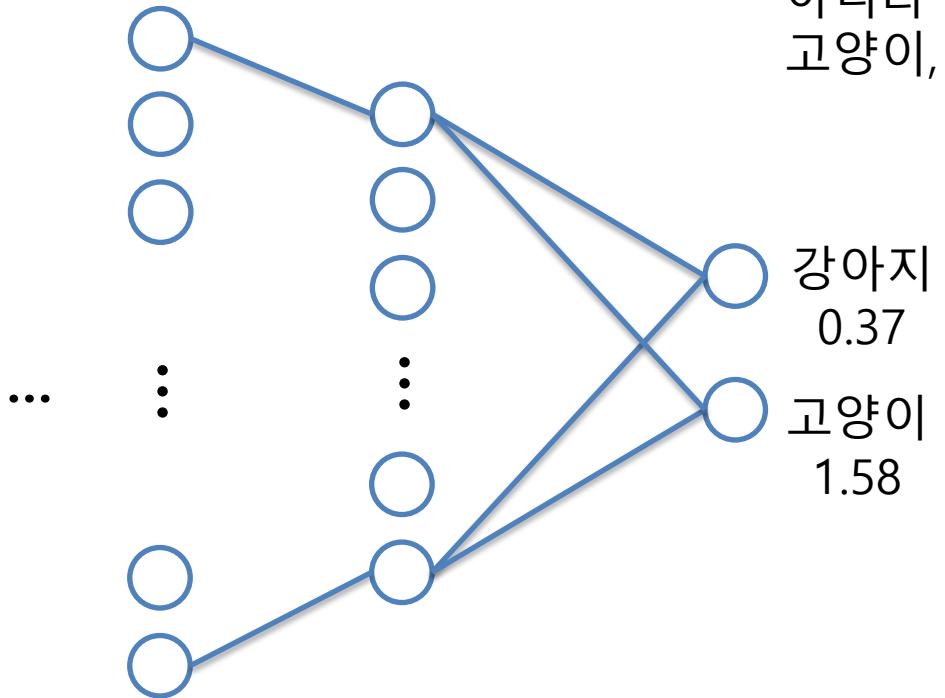


Basic Architecture

loss function은?



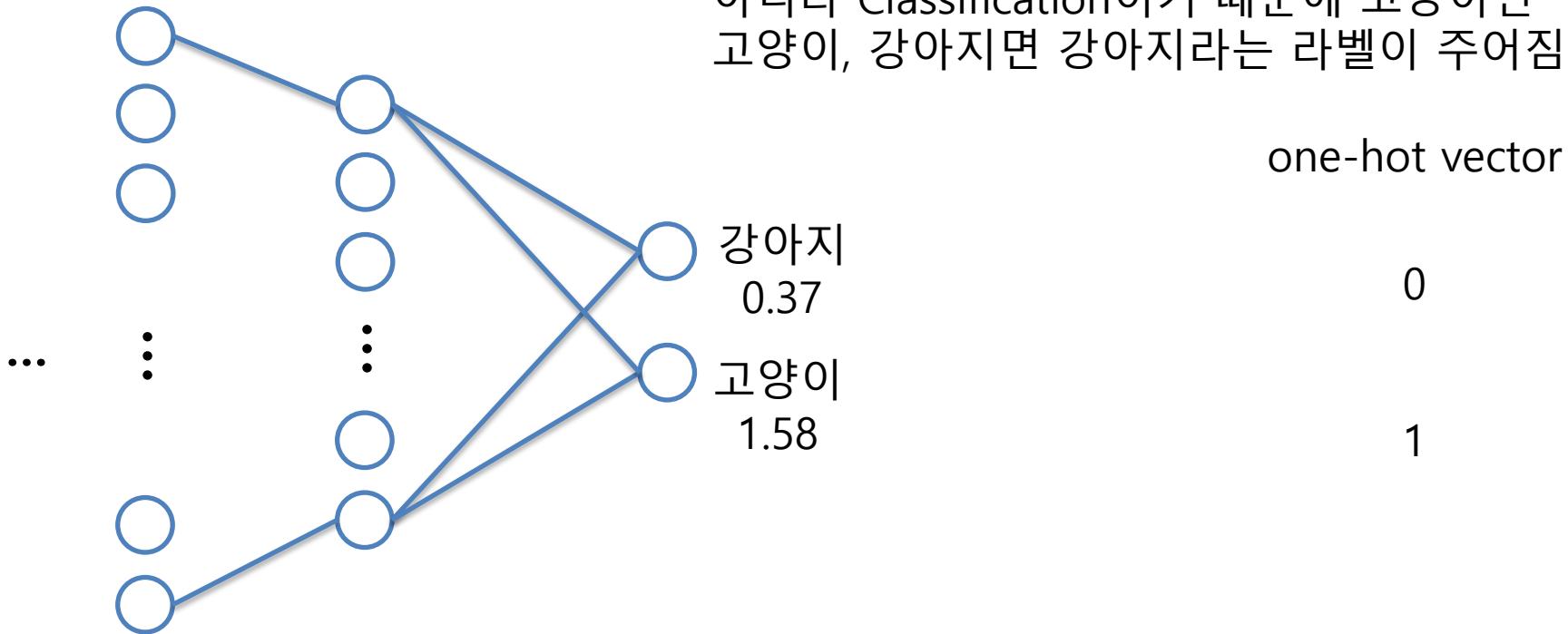
Basic Architecture



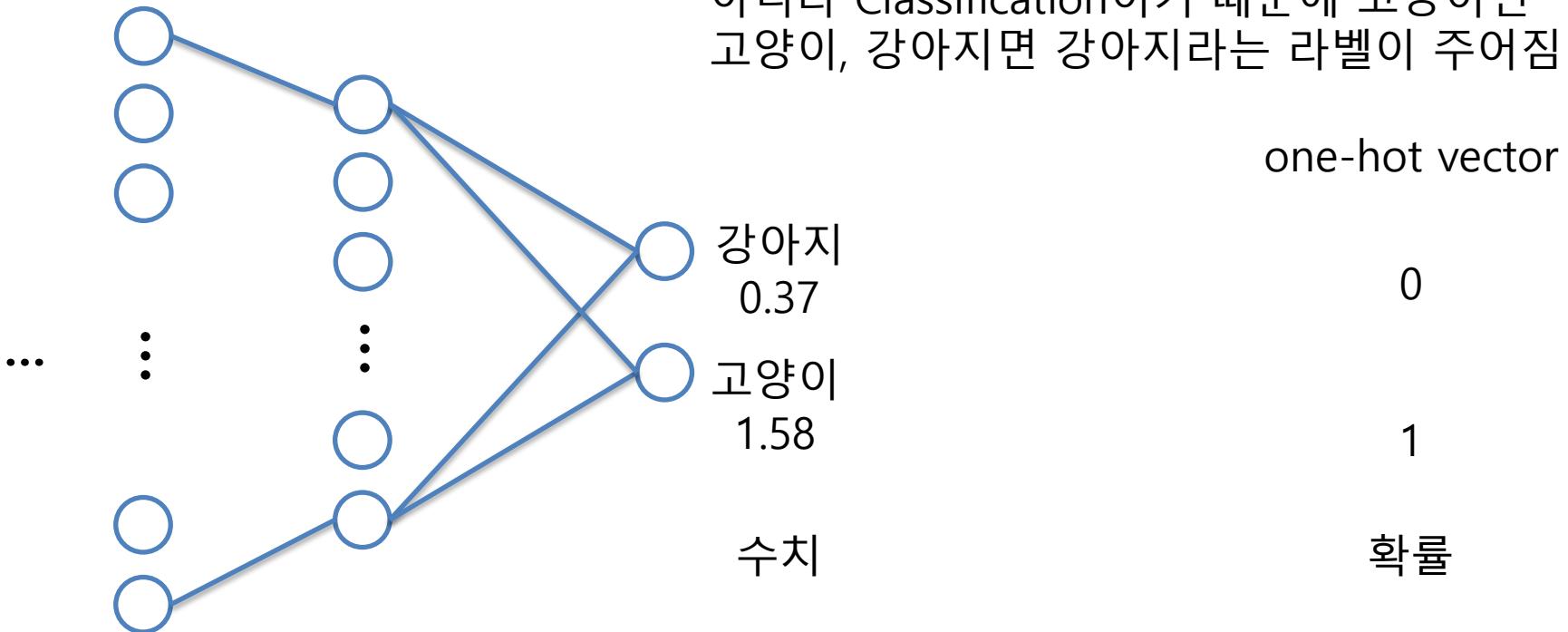
loss function은?

기존의 Regression처럼 수치가 주어지는게 아니라 Classification이기 때문에 고양이면 고양이, 강아지면 강아지라는 라벨이 주어짐

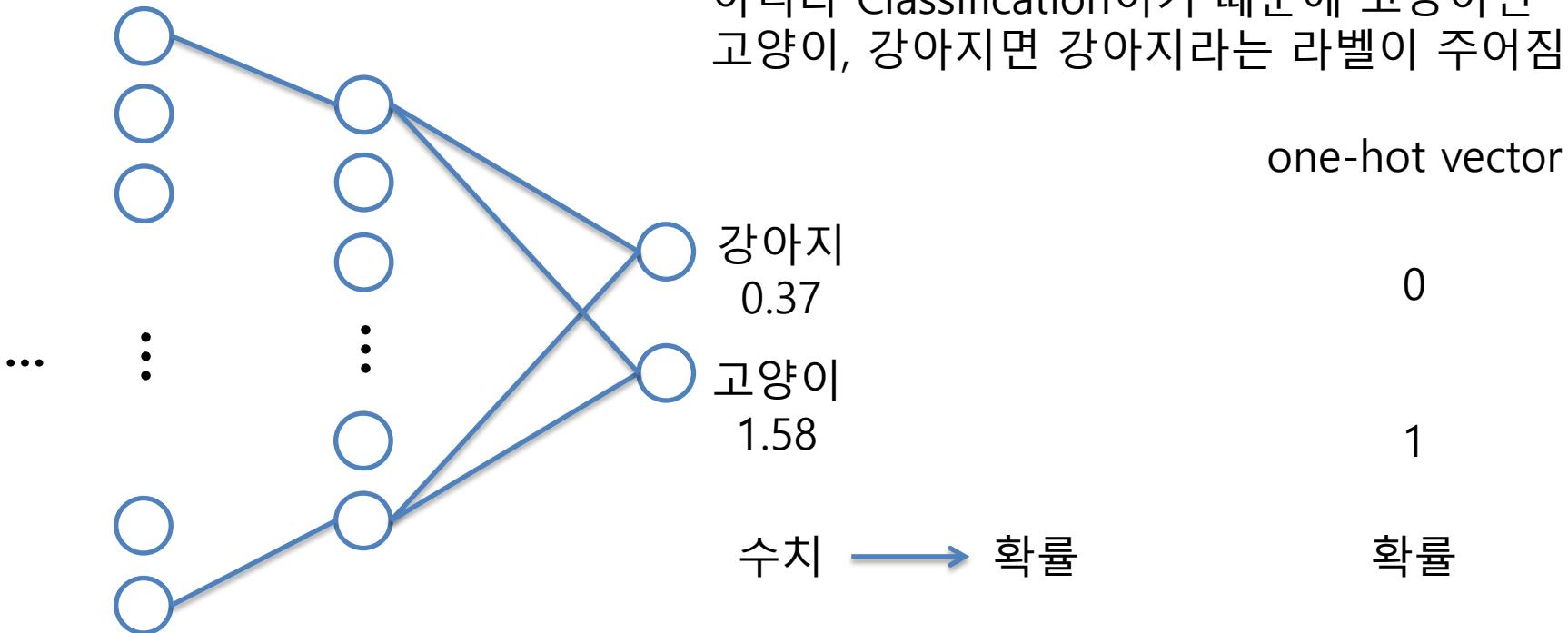
Basic Architecture



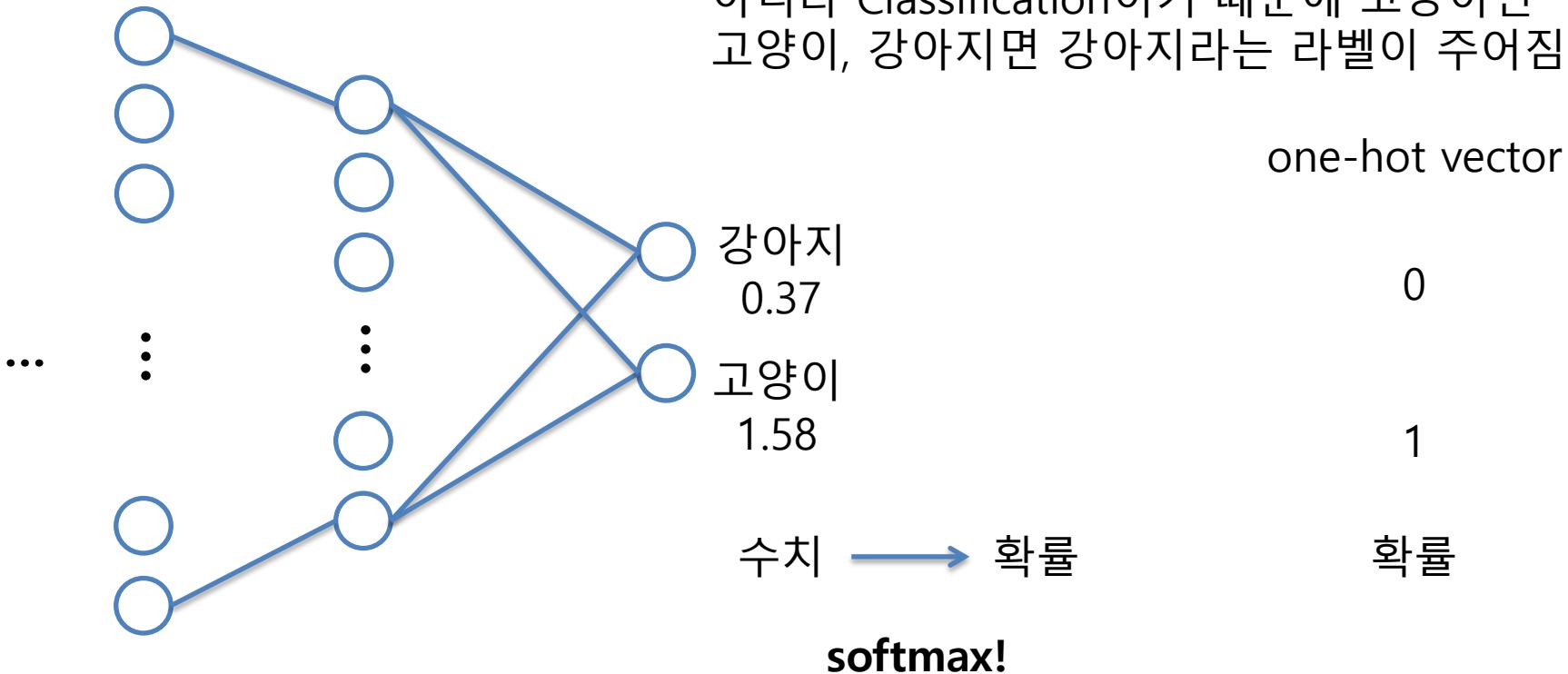
Basic Architecture



Basic Architecture



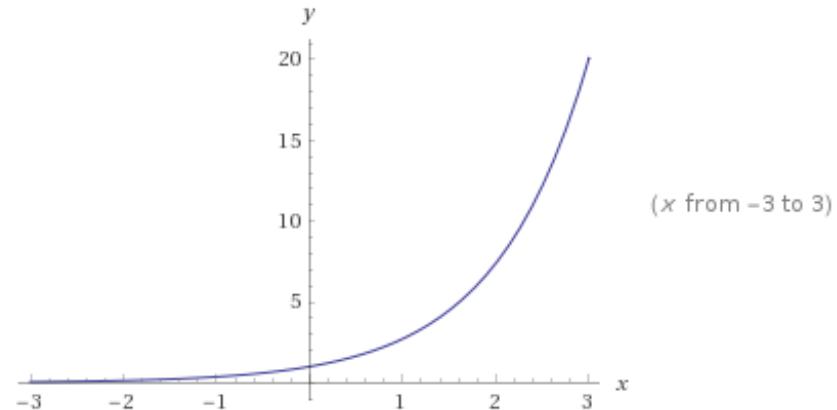
Basic Architecture



Basic Architecture

$$\text{softmax}(\mathbf{y})_i = \frac{\exp(y_i)}{\sum_j \exp(y_j)}$$

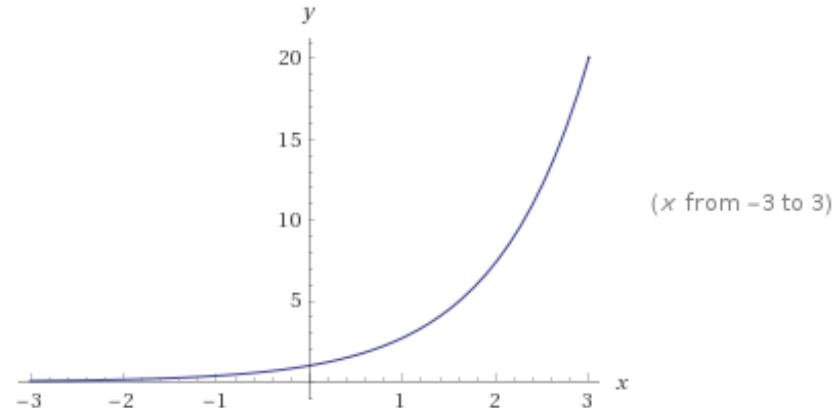
수치적 결과를 확률로 바꿔줌



Basic Architecture

$$\text{softmax}(\mathbf{y})_i = \frac{\exp(y_i)}{\sum_j \exp(y_j)}$$

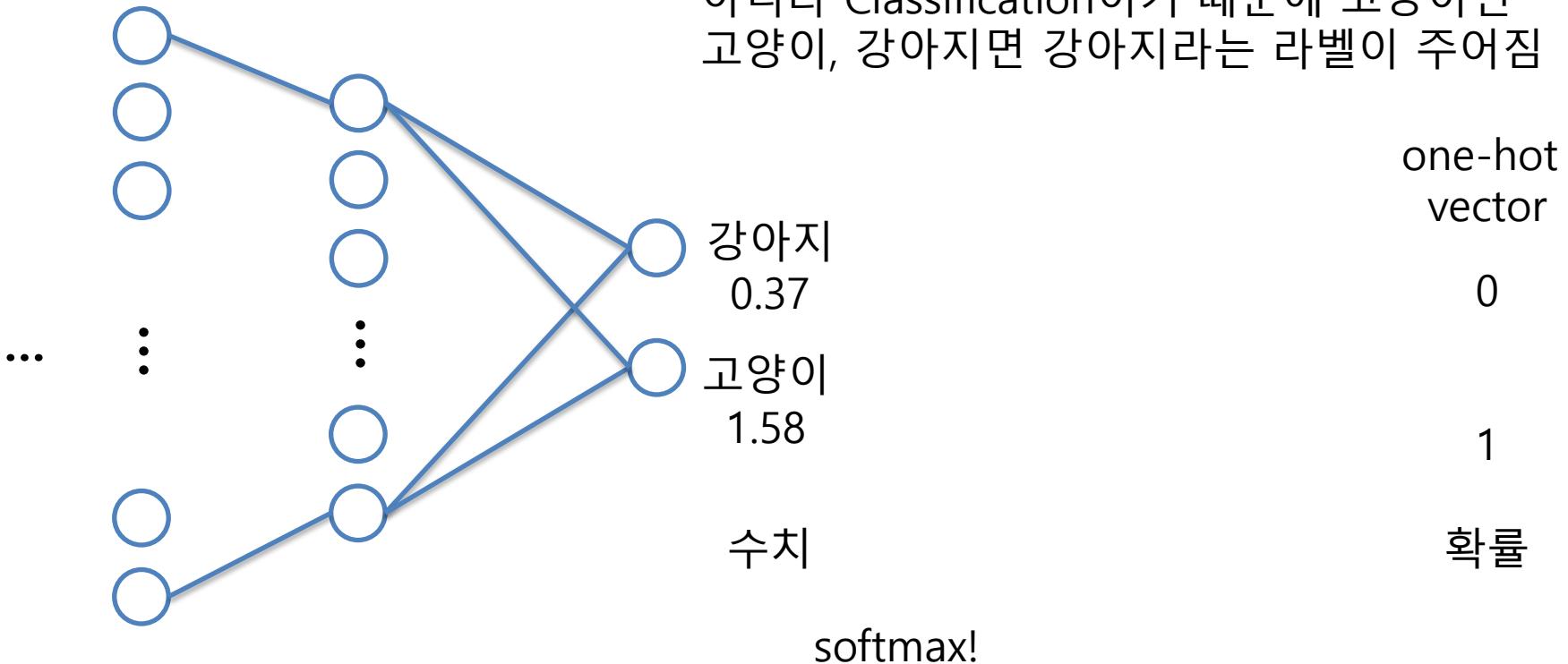
수치적 결과를 확률로 바꿔줌



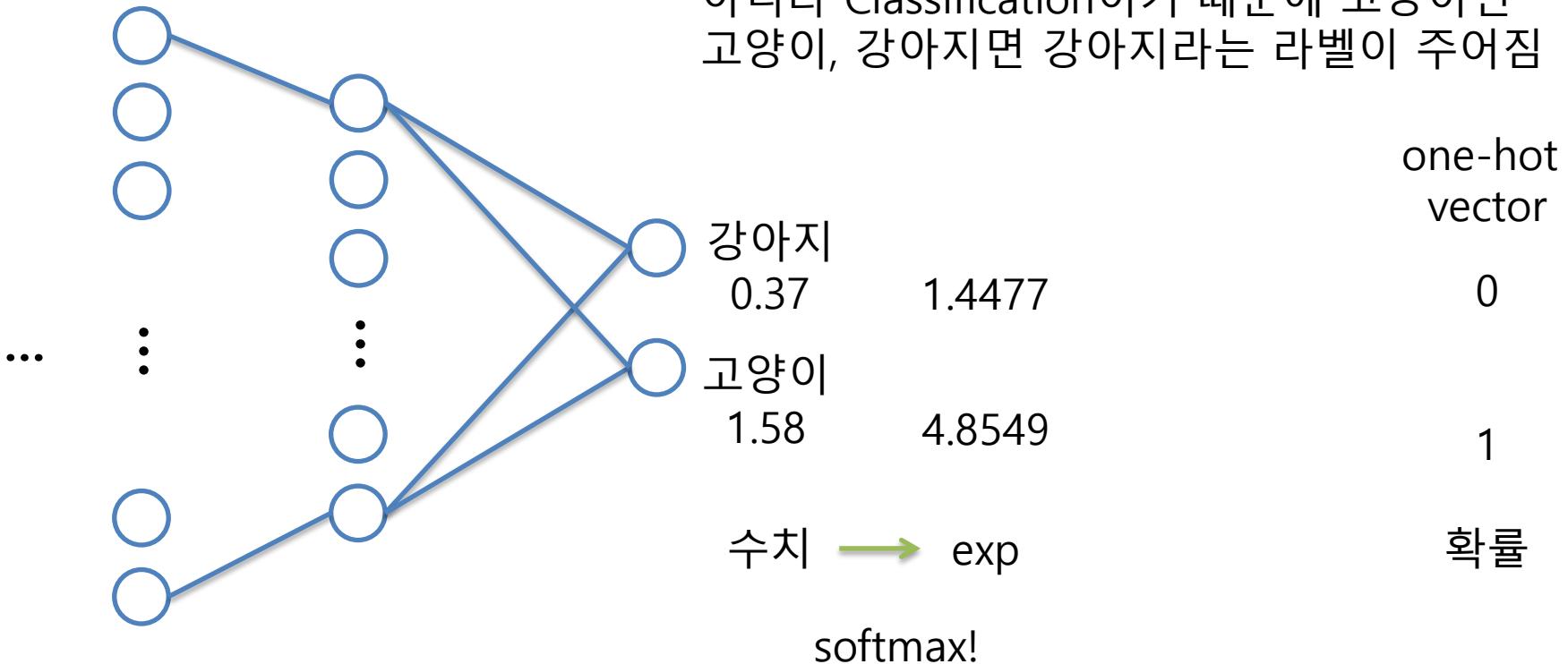
사용되는 이유에는 Information Theory, Probabilistic Theory 및 Numerical Stability적 근거들이 있지만 시간관계상 링크로 대체

<http://cs231n.github.io/linear-classify/#softmax>

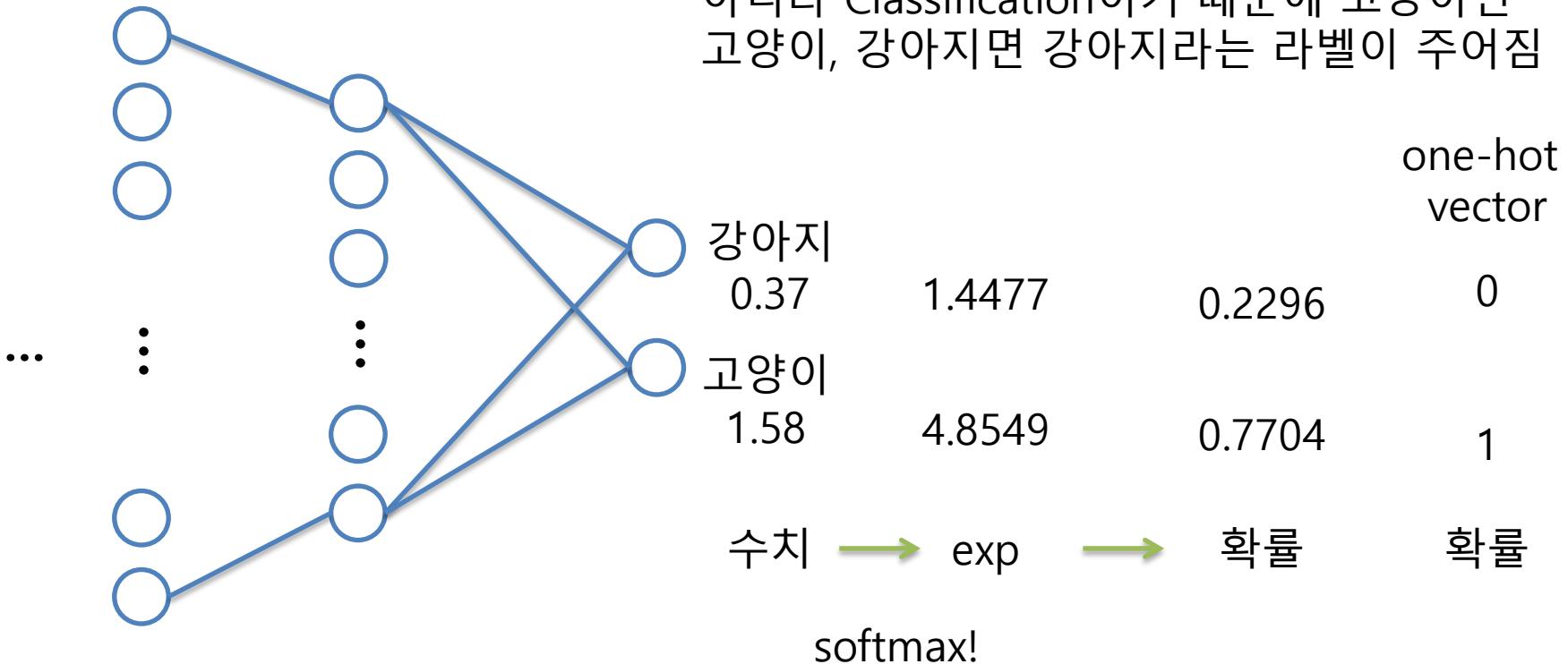
Basic Architecture



Basic Architecture



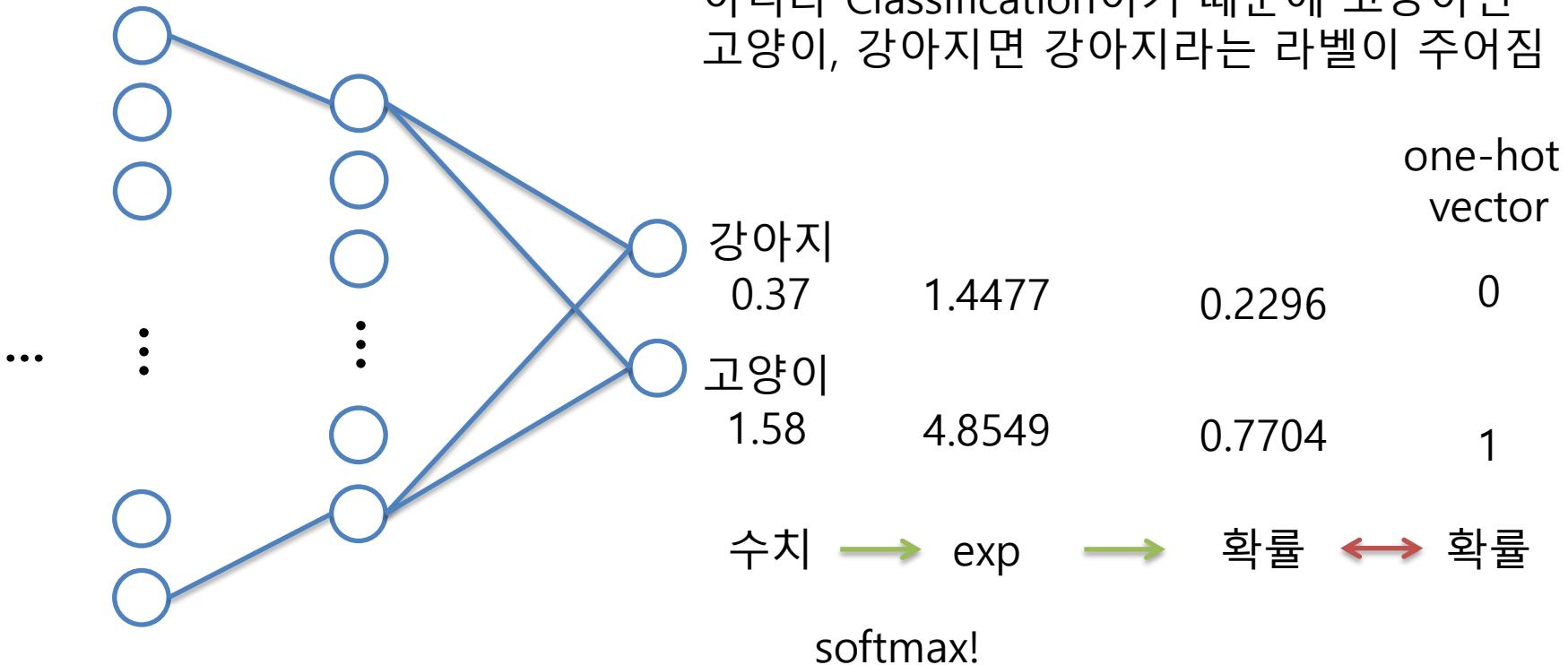
Basic Architecture



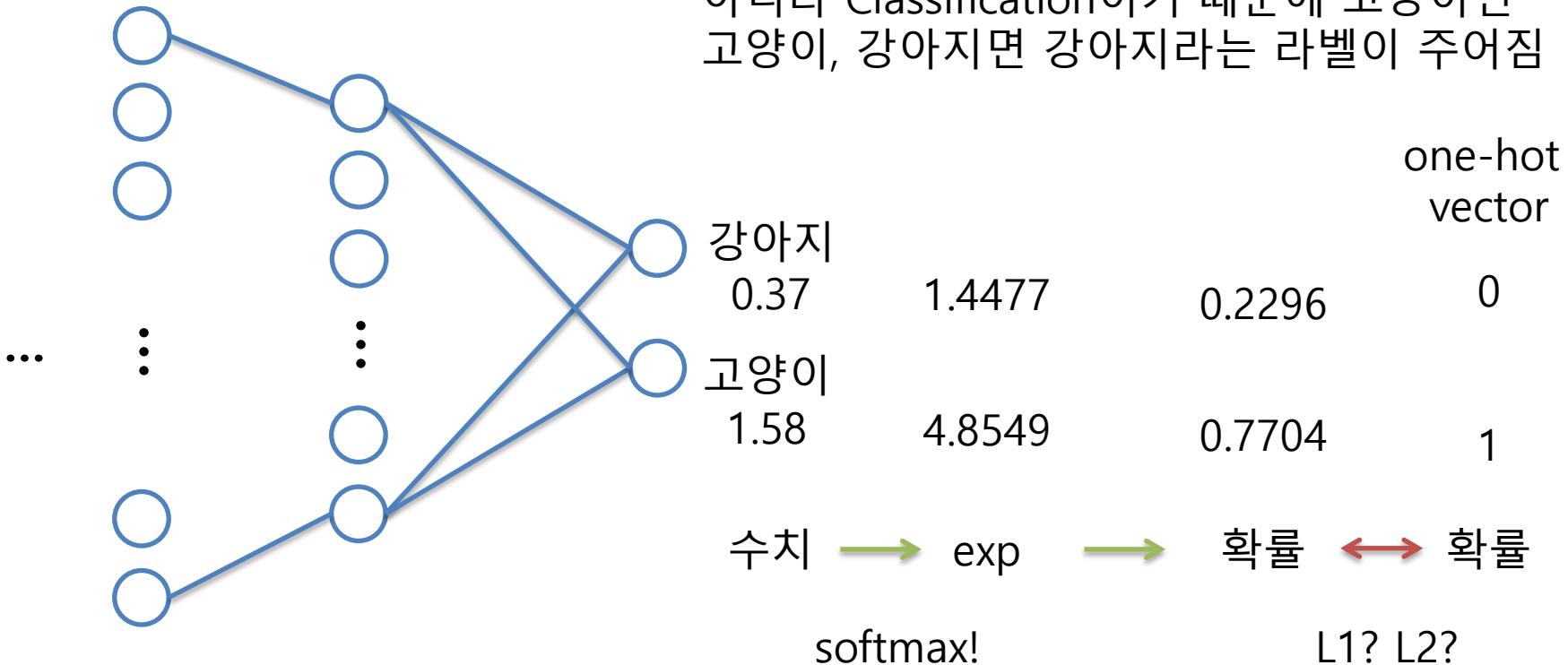
loss function은?

기존의 Regression처럼 수치가 주어지는게 아니라 Classification이기 때문에 고양이면 고양이, 강아지면 강아지라는 라벨이 주어짐

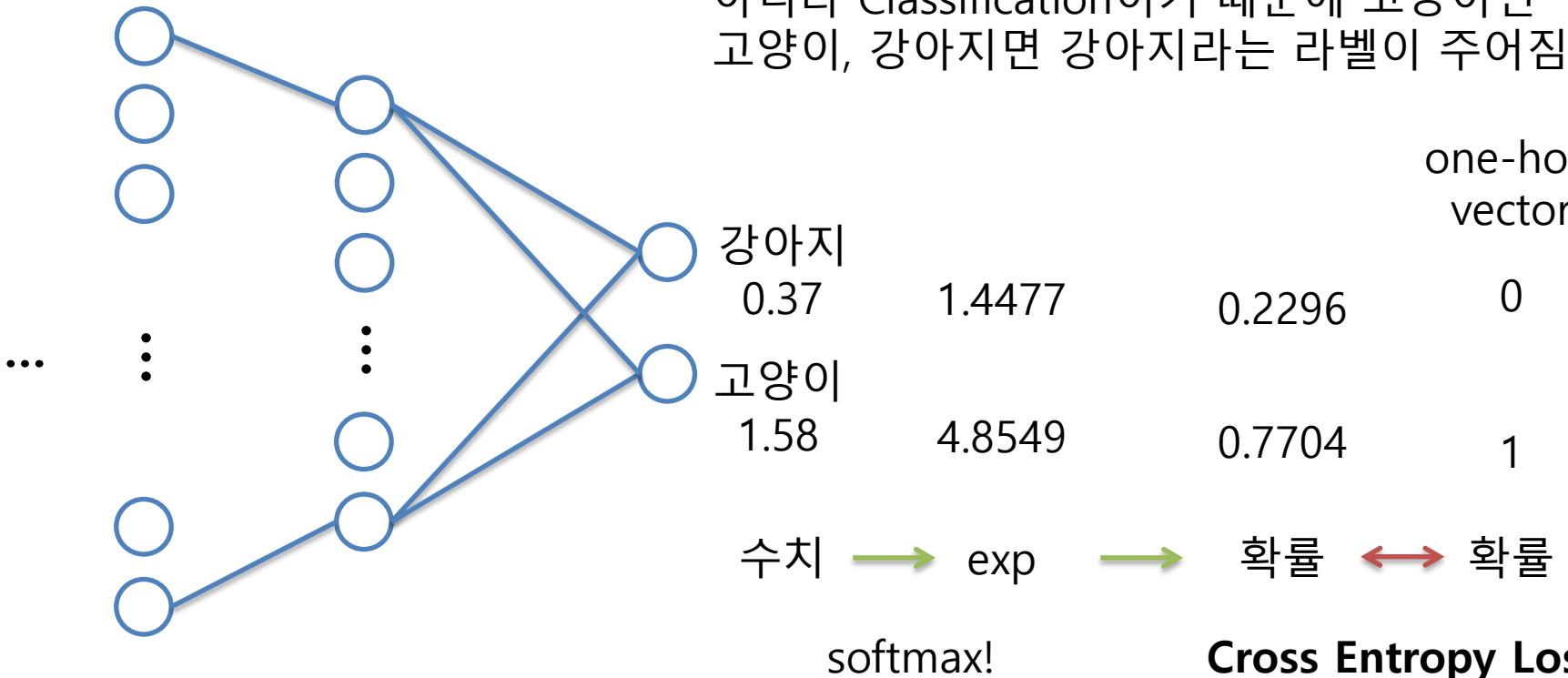
Basic Architecture



Basic Architecture

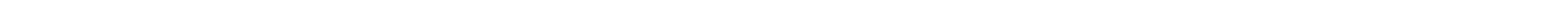


Basic Architecture



Basic Architecture

Entropy?



Basic Architecture

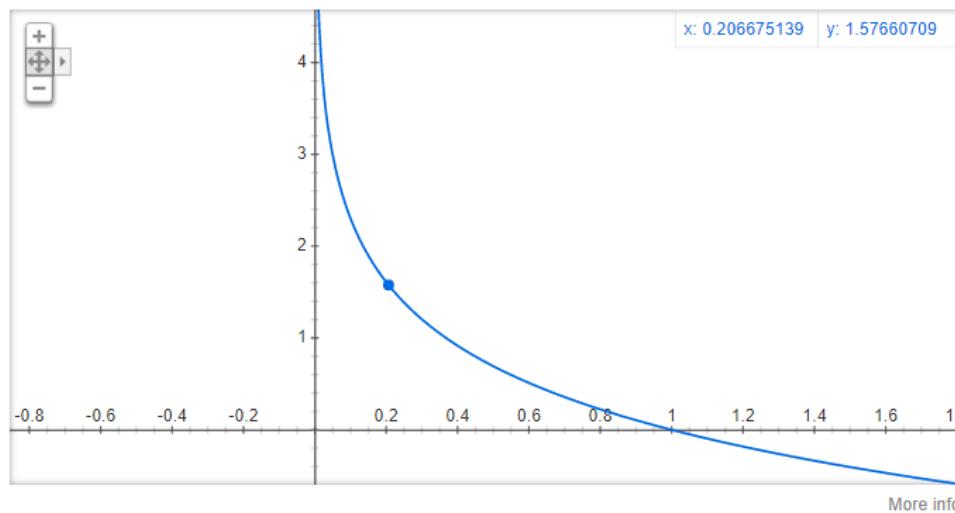
Entropy? $H(X) = E[I(X)] = E[-\ln(P(X))]$. 정보량의 기대값

Basic Architecture

Entropy? $H(X) = E[I(X)] = E[-\ln(P(X))]$.

정보량의 기대값

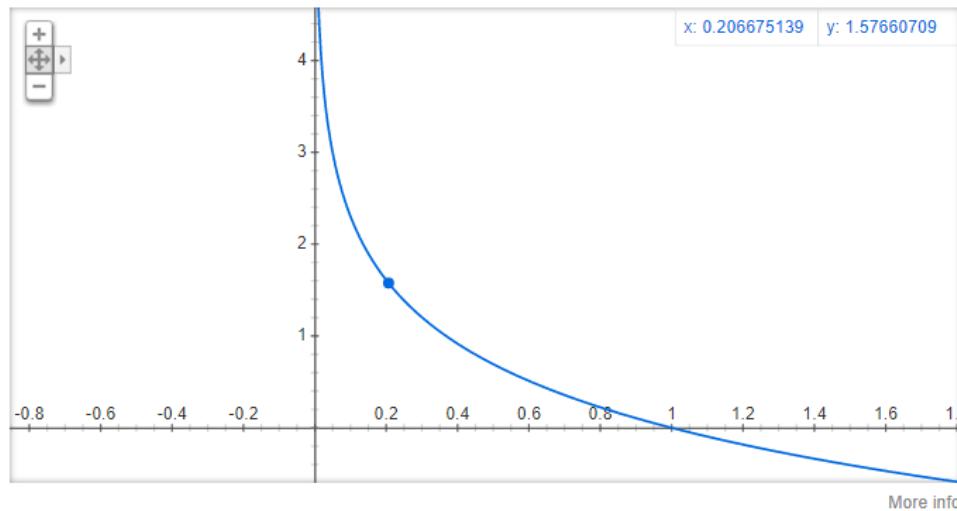
Graph for $-\ln(x)$



Basic Architecture

Entropy? $H(X) = E[I(X)] = E[-\ln(P(X))]$. 정보량의 기대값

Graph for $-\ln(x)$

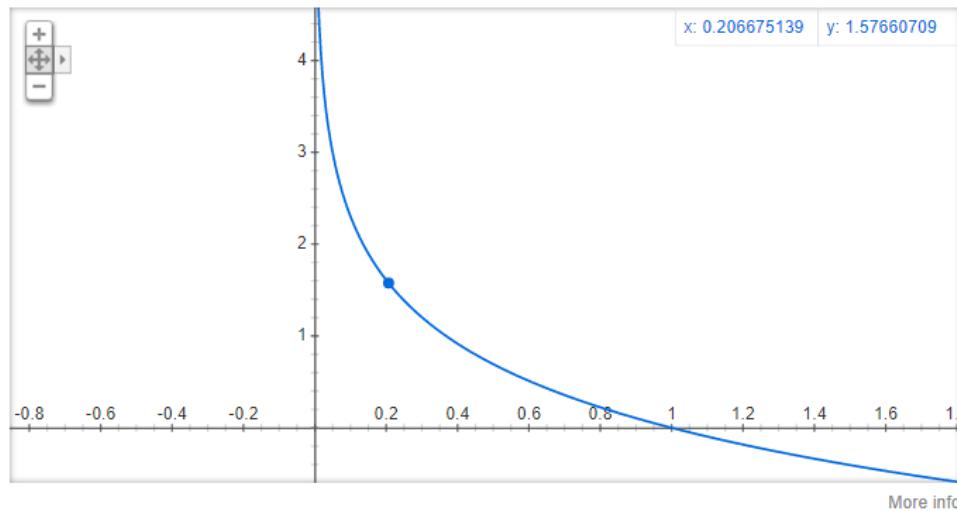


만약 항상 일어나는 사건이라면 $p=1$ 이고 이것이 가지고 있는 정보는 0.

Basic Architecture

Entropy? $H(X) = E[I(X)] = E[-\ln(P(X))]$. 정보량의 기대값

Graph for $-\ln(x)$



만약 항상 일어나는 사건이라면 $p=1$ 이고 이것이 가지고 있는 정보는 0.

일어날 확률이 적은 사건일수록 많은 정보를 담고 있다.

Basic Architecture

discrete 한 경우

Entropy? $H(X) = E[I(X)] = E[-\ln(P(X))]$.

Basic Architecture

discrete 한 경우

Entropy?

$$\begin{aligned} H(X) &= E[I(X)] = E[-\ln(P(X))]. \\ &= \sum_{i=1}^n P(x_i) I(x_i) = - \sum_{i=1}^n P(x_i) \log_b P(x_i), \end{aligned}$$

Basic Architecture

discrete 한 경우

Entropy?

$$\begin{aligned} H(X) &= E[I(X)] = E[-\ln(P(X))]. \\ &= \sum_{i=1}^n P(x_i) I(x_i) = - \sum_{i=1}^n P(x_i) \log_b P(x_i), \end{aligned}$$

ex) 동전 던지기

$$\begin{aligned} P(\text{앞}) &= 1/2 \\ P(\text{뒤}) &= 1/2 \end{aligned}$$

$$\begin{aligned} H(X) &= 0.5 * (-\log_2 0.5) + 0.5 * (-\log_2 0.5) \\ &= 0.5 + 0.5 \\ &= 1 \end{aligned}$$

Basic Architecture

discrete 한 경우

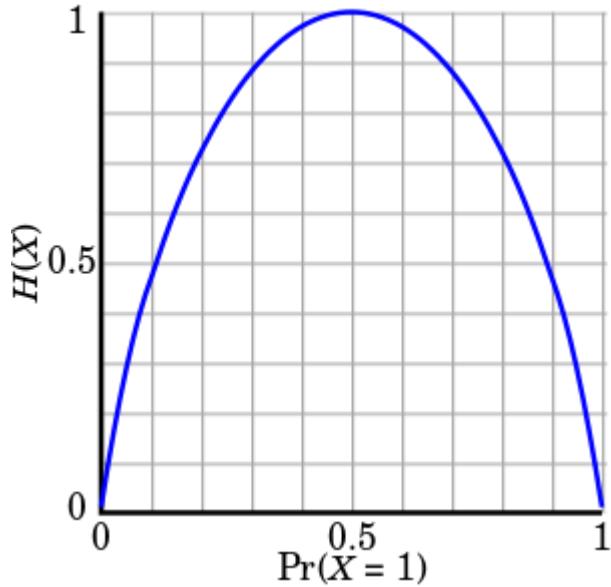
Entropy? $H(X) = E[I(X)] = E[-\ln(P(X))].$

$$= \sum_{i=1}^n P(x_i) I(x_i) = - \sum_{i=1}^n P(x_i) \log_b P(x_i),$$

ex) 동전 던지기

$$\begin{aligned}P(\text{앞}) &= 1/2 \\P(\text{뒤}) &= 1/2\end{aligned}$$

$$\begin{aligned}H(X) &= 0.5 * (-\log_2 0.5) + 0.5 * (-\log_2 0.5) \\&= 0.5 + 0.5 \\&= 1\end{aligned}$$



Basic Architecture

discrete 한 경우

Entropy? $H(X) = E[I(X)] = E[-\ln(P(X))].$

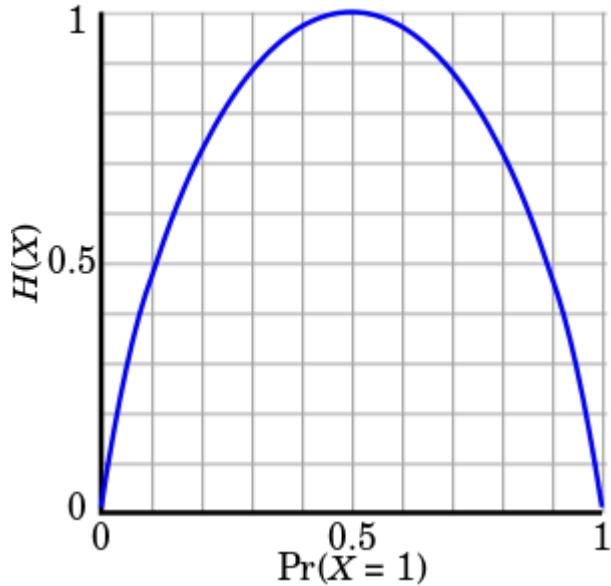
$$= \sum_{i=1}^n P(x_i) I(x_i) = - \sum_{i=1}^n P(x_i) \log_b P(x_i),$$

ex) 동전 던지기

$$\begin{aligned}P(\text{앞}) &= 1/2 \\P(\text{뒤}) &= 1/2\end{aligned}$$

$$\begin{aligned}H(X) &= 0.5 * (-\log_2 0.5) + 0.5 * (-\log_2 0.5) \\&= 0.5 + 0.5 \\&= 1\end{aligned}$$

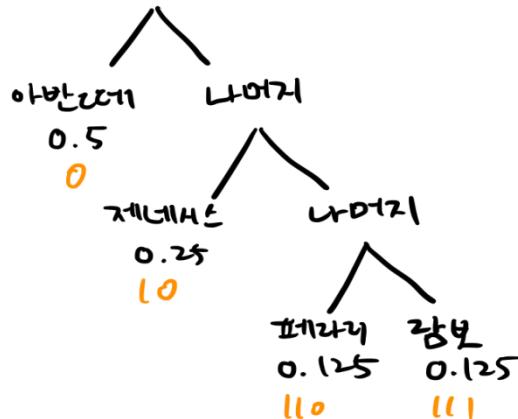
이 확률 분포를 표현하기 위해
서는 1 bit가 필요하다



Basic Architecture

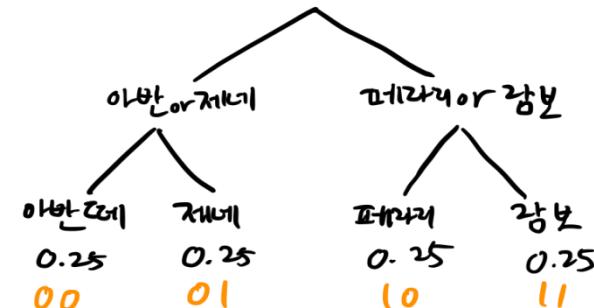
ex) 길에서 아반떼, 제네시스, 페라리, 람보르기니를 볼 확률이 각각 $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, $\frac{1}{8}$ 일 때 entropy는 얼마인가?

그림으로 그려보면



$$\begin{aligned} & 1 \times 0.5 + 2 \times 0.25 + 3 \times 0.125 + 3 \times 0.125 \\ & = 0.5 + 0.5 + 0.75 \\ & = 1.75 \end{aligned}$$

만약 전부 $\frac{1}{4}$ 이었다면?



$$\begin{aligned} & 2 \times 0.25 + 2 \times 0.25 + 2 \times 0.25 + 2 \times 0.25 \\ & = 0.5 \times 4 \\ & = 2 \end{aligned}$$

Basic Architecture

Cross Entropy 는?

Basic Architecture

Cross Entropy 는?

원래 블록 y^2 을 \hat{y} 로 인코딩해볼 때 필요한 비트의 수

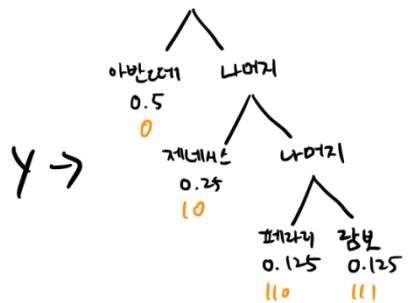
$$\Rightarrow H(y, \hat{y}) = -\sum_{i=1}^n y_i \log \hat{y}_i$$

Basic Architecture

Cross Entropy 는?

원래 블로 y_2 를 \hat{y} 로 인코딩해볼 때 필요한 비트의 수

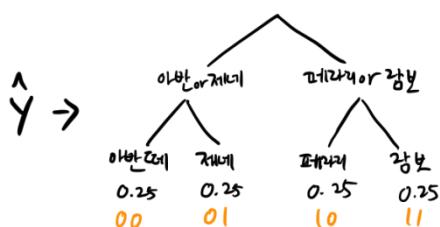
$$\Rightarrow H(y, \hat{y}) = -\sum_{i=1}^n y_i \log \hat{y}_i$$



$$y = [0.5, 0.25, 0.125, 0.125]$$

$$\hat{y} = [0.25, 0.25, 0.25, 0.25]$$

\hat{y} 기준으로 모든 차는 2bit.



y_2 를 \hat{y} 기준으로 표현하면

$$2\text{bit} \times 0.5 + 2 \times 0.25 + 2 \times 0.125 + 2 \times 0.125$$

$$= 1 + 0.5 + 0.5 = 2\text{bit}$$

원래는 1.75

Basic Architecture

Cross Entropy

$$H(y, \hat{y}) = -\sum_{i=1}^n y_i \log \hat{y}_i$$

$$\text{cf.) } H(y) = -\sum_{i=1}^n y_i \log y_i$$

$$= -\sum_{i=1}^n y_i \log \hat{y}_i - \underbrace{\sum_{i=1}^n y_i \log y_i}_{H(y)} + \sum_{i=1}^n y_i \log y_i$$

$$= \underbrace{H(y)}_{H(y)} + \sum_{i=1}^n y_i \log y_i - \sum_{i=1}^n y_i \log \hat{y}_i$$

$$= \underbrace{H(y)}_{H(y)} + \sum_{i=1}^n y_i \log \frac{y_i}{\hat{y}_i}$$

$$\Rightarrow H(y, \hat{y}) - H(y) = \sum_{i=1}^n y_i \log \frac{y_i}{\hat{y}_i}$$

☞ By def. KL Divergence

= Cross Entropy - Entropy

결과적으로 Cross Entropy를 minimize 하는 것은 $H(y)$ 가 고정이기 때문에
KL Divergence를 Minimize 하는 것과 같다.

Basic Architecture

true distribution p ,
unnatural distribution q

Cross Entropy Loss

$$\begin{aligned} H(p, q) &= \text{E}_p[-\log q] = H(p) + D_{\text{KL}}(p\|q), \\ &= - \sum_x p(x) \log q(x). \end{aligned}$$

실제 p 의 distribution을 갖는 정보
를 q 로 표현했을 때 드는 정보량



Basic Architecture

true distribution p ,
unnatural distribution q

Cross Entropy Loss

p 자체의 엔트로피



$$H(p, q) = \mathbb{E}_p[-\log q] = H(p) + D_{\text{KL}}(p\|q),$$

$$= - \sum_x p(x) \log q(x).$$

실제 p 의 distribution을 갖는 정보
를 q 로 표현했을 때 드는 정보량

Basic Architecture

true distribution p ,
unnatural distribution q

Cross Entropy Loss

p 자체의 엔트로피

$$H(p, q) = \text{E}_p[-\log q] = H(p) + D_{\text{KL}}(p\|q),$$

$$= - \sum_x p(x) \log q(x).$$

실제 p 의 distribution을 갖는 정보
를 q 로 표현했을 때 드는 정보량

$$D_{\text{KL}}(P\|Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}.$$

Basic Architecture

true distribution p ,
unnatural distribution q

Cross Entropy Loss

p 자체의 엔트로피

$$H(p, q) = \text{E}_p[-\log q] = H(p) + D_{\text{KL}}(p\|q),$$

$$= - \sum_x p(x) \log q(x).$$

실제 p 의 distribution을 갖는 정보
를 q 로 표현했을 때 드는 정보량

$$D_{\text{KL}}(P\|Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}.$$

p 를 기준으로 q 가 얼마나 다른가

Basic Architecture

Cross Entropy Loss

$$\begin{aligned} H(p, q) &= \text{E}_p[-\log q] = H(p) + D_{\text{KL}}(p\|q), \\ &= - \sum_x p(x) \log q(x). \end{aligned}$$

실제 p 의 distribution을 갖는 정보
를 q 로 표현했을 때 드는 정보량

p 자체의 엔트로피

true distribution p ,
unnatural distribution q

결국 Cross Entropy를 minimize
하는 것은 p, q 의 분포가 가까워
-지도록 만드는 것

$$D_{\text{KL}}(P\|Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}.$$

p 를 기준으로 q 가 얼마나 다른가

Basic Architecture

true distribution p ,
unnatural distribution q

Cross Entropy Loss

$$H(p, q) = - \sum_x p(x) \log q(x).$$

Basic Architecture

true distribution p ,
unnatural distribution q

Cross Entropy Loss

$$H(p, q) = - \sum_x p(x) \log q(x).$$

만약 실제 라벨이 고양이인데

0.9로 예측했다면 Loss = 0.1053

0.3으로 예측했다면 Loss = 1.2039

0.1으로 예측했다면 Loss = 2.3025

Basic Architecture

true distribution p ,
unnatural distribution q

Cross Entropy Loss

$$H(p, q) = - \sum_x p(x) \log q(x).$$

만약 실제 라벨이 고양이인데
0.9로 예측했다면 Loss = 0.1053
0.3으로 예측했다면 Loss = 1.2039
0.1으로 예측했다면 Loss = 2.3025



L1 Loss 였다면
Loss = 0.1
Loss = 0.7
Loss = 0.9

Basic Architecture

true distribution p ,
unnatural distribution q

Cross Entropy Loss

$$H(p, q) = - \sum_x p(x) \log q(x).$$

만약 실제 라벨이 고양이인데
0.9로 예측했다면 Loss = 0.1053
0.3으로 예측했다면 Loss = 1.2039
0.1으로 예측했다면 Loss = 2.3025



L1 Loss 였다면
Loss = 0.1
Loss = 0.7
Loss = 0.9

잘못된 예측에 대하여 penalty가 더 크기 때문에
Classification task에 대해서는 Cross Entropy Loss를 씀

MNIST Example

```
4 import torch
5 import torch.nn as nn
6 import torch.utils as utils
7 from torch.autograd import Variable
8 import torchvision.datasets as dset
9 import torchvision.transforms as transforms
10 import matplotlib.pyplot as plt
11
12 # Set Hyperparameters
13
14 epoch = 1000
15 batch_size = 16
16 learning_rate = 0.001
17
18 # Download Data
19
20 mnist_train = dset.MNIST("./", train=True, transform=transforms.ToTensor(), target_transform=None, download=True)
21 mnist_test = dset.MNIST("./", train=False, transform=transforms.ToTensor(), target_transform=None, download=True)
22
23 # Check the datasets downloaded
24
25 print(mnist_train.__len__())
26 print(mnist_test.__len__())
27 img1,label1 = mnist_train.__getitem__(0)
28 img2,label2 = mnist_test.__getitem__(0)
29
30 print(img1.size(), label1)
31 print(img2.size(), label2)
32
33 # Set Data Loader(input pipeline)
34
35 train_loader = torch.utils.data.DataLoader(dataset=mnist_train,batch_size=batch_size,shuffle=True)
36 test_loader = torch.utils.data.DataLoader(dataset=mnist_test,batch_size=batch_size,shuffle=True)
```

MNIST Example

Import Libraries

```
4 import torch
5 import torch.nn as nn
6 import torch.utils as utils
7 from torch.autograd import Variable
8 import torchvision.datasets as dset
9 import torchvision.transforms as transforms
10 import matplotlib.pyplot as plt
11
12 # Set Hyperparameters
13
14 epoch = 1000
15 batch_size = 16
16 learning_rate = 0.001
17
18 # Download Data
19
20 mnist_train = dset.MNIST("./", train=True, transform=transforms.ToTensor(), target_transform=None, download=True)
21 mnist_test = dset.MNIST("./", train=False, transform=transforms.ToTensor(), target_transform=None, download=True)
22
23 # Check the datasets downloaded
24
25 print(mnist_train.__len__())
26 print(mnist_test.__len__())
27 img1,label1 = mnist_train.__getitem__(0)
28 img2,label2 = mnist_test.__getitem__(0)
29
30 print(img1.size(), label1)
31 print(img2.size(), label2)
32
33 # Set Data Loader(input pipeline)
34
35 train_loader = torch.utils.data.DataLoader(dataset=mnist_train,batch_size=batch_size,shuffle=True)
36 test_loader = torch.utils.data.DataLoader(dataset=mnist_test,batch_size=batch_size,shuffle=True)
```

MNIST Example

Import Libraries

Hyperparameter
setting

```
4 import torch
5 import torch.nn as nn
6 import torch.utils as utils
7 from torch.autograd import Variable
8 import torchvision.datasets as dset
9 import torchvision.transforms as transforms
10 import matplotlib.pyplot as plt
11
12 # Set Hyperparameters
13
14 epoch = 1000
15 batch_size = 16
16 learning_rate = 0.001
17
18 # Download Data
19
20 mnist_train = dset.MNIST("./", train=True, transform=transforms.ToTensor(), target_transform=None, download=True)
21 mnist_test = dset.MNIST("./", train=False, transform=transforms.ToTensor(), target_transform=None, download=True)
22
23 # Check the datasets downloaded
24
25 print(mnist_train.__len__())
26 print(mnist_test.__len__())
27 img1,label1 = mnist_train.__getitem__(0)
28 img2,label2 = mnist_test.__getitem__(0)
29
30 print(img1.size(), label1)
31 print(img2.size(), label2)
32
33 # Set Data Loader(input pipeline)
34
35 train_loader = torch.utils.data.DataLoader(dataset=mnist_train,batch_size=batch_size,shuffle=True)
36 test_loader = torch.utils.data.DataLoader(dataset=mnist_test,batch_size=batch_size,shuffle=True)
```

MNIST Example

Import Libraries

```
4 import torch
5 import torch.nn as nn
6 import torch.utils as utils
7 from torch.autograd import Variable
8 import torchvision.datasets as dset
9 import torchvision.transforms as transforms
10 import matplotlib.pyplot as plt
11
12 # Set Hyperparameters
13
14 epoch = 1000
15 batch_size = 16
16 learning_rate = 0.001
17
18 # Download Data
19
20 mnist_train = dset.MNIST("./", train=True, transform=transforms.ToTensor(), target_transform=None, download=True)
21 mnist_test = dset.MNIST("./", train=False, transform=transforms.ToTensor(), target_transform=None, download=True)
22
23 # Check the datasets downloaded
24
25 print(mnist_train.__len__())
26 print(mnist_test.__len__())
27 img1,label1 = mnist_train.__getitem__(0)
28 img2,label2 = mnist_test.__getitem__(0)
29
30 print(img1.size(), label1)
31 print(img2.size(), label2)
32
33 # Set Data Loader(input pipeline)
34
35 train_loader = torch.utils.data.DataLoader(dataset=mnist_train,batch_size=batch_size,shuffle=True)
36 test_loader = torch.utils.data.DataLoader(dataset=mnist_test,batch_size=batch_size,shuffle=True)
```

Hyperparameter setting

Download Data

MNIST Example

Import Libraries

```
4 import torch
5 import torch.nn as nn
6 import torch.utils as utils
7 from torch.autograd import Variable
8 import torchvision.datasets as dset
9 import torchvision.transforms as transforms
10 import matplotlib.pyplot as plt
```

Hyperparameter setting

```
12 # Set Hyperparameters
13
14 epoch = 1000
15 batch_size = 16
16 learning_rate = 0.001
```

Download Data

```
18 # Download Data
19
20 mnist_train = dset.MNIST("./", train=True, transform=transforms.ToTensor(), target_transform=None, download=True)
21 mnist_test = dset.MNIST("./", train=False, transform=transforms.ToTensor(), target_transform=None, download=True)
```

Check Data

```
22 # Check the datasets downloaded
23
24 print(mnist_train.__len__())
25 print(mnist_test.__len__())
26 img1,label1 = mnist_train.__getitem__(0)
27 img2,label2 = mnist_test.__getitem__(0)
28
29 print(img1.size(), label1)
30 print(img2.size(), label2)
```

```
31
32
33 # Set Data Loader(input pipeline)
34
35 train_loader = torch.utils.data.DataLoader(dataset=mnist_train,batch_size=batch_size,shuffle=True)
36 test_loader = torch.utils.data.DataLoader(dataset=mnist_test,batch_size=batch_size,shuffle=True)
```

MNIST Example

Import Libraries

Hyperparameter setting

Download Data

Check Data

Data to Batch

```
4 import torch
5 import torch.nn as nn
6 import torch.utils as utils
7 from torch.autograd import Variable
8 import torchvision.datasets as dset
9 import torchvision.transforms as transforms
10 import matplotlib.pyplot as plt
11
12 # Set Hyperparameters
13
14 epoch = 1000
15 batch_size = 16
16 learning_rate = 0.001
17
18 # Download Data
19
20 mnist_train = dset.MNIST("./", train=True, transform=transforms.ToTensor(), target_transform=None, download=True)
21 mnist_test = dset.MNIST("./", train=False, transform=transforms.ToTensor(), target_transform=None, download=True)
22
23 # Check the datasets downloaded
24
25 print(mnist_train.__len__())
26 print(mnist_test.__len__())
27 img1,label1 = mnist_train.__getitem__(0)
28 img2,label2 = mnist_test.__getitem__(0)
29
30 print(img1.size(), label1)
31 print(img2.size(), label2)
32
33 # Set Data Loader(input pipeline)
34
35 train_loader = torch.utils.data.DataLoader(dataset=mnist_train,batch_size=batch_size,shuffle=True)
36 test_loader = torch.utils.data.DataLoader(dataset=mnist_test,batch_size=batch_size,shuffle=True)
```

MNIST Example

```
47 class CNN(nn.Module):
48     def __init__(self):
49         super(CNN, self).__init__()
50         self.layer1 = nn.Sequential(
51             nn.Conv2d(1,16,5),    # batch x 16 x 24 x 24
52             nn.ReLU(),
53             nn.Conv2d(16,32,5),   # batch x 32 x 20 x 20
54             nn.ReLU(),
55             nn.MaxPool2d(2,2)    # batch x 32 x 10 x 10
56         )
57         self.layer2 = nn.Sequential(
58             nn.Conv2d(32,64,5),   # batch x 64 x 6 x 6
59             nn.ReLU(),
60             nn.Conv2d(64,128,5),  # batch x 128 x 2 x 2
61             nn.ReLU()
62         )
63         self.fc = nn.Linear(2*2*128,10)
64
65     def forward(self,x):
66         out = self.layer1(x)
67         out = self.layer2(out)
68         out = out.view(batch_size, -1)
69         out = self.fc(out)
70         return out
71
72 cnn = CNN().cuda()
73
74 loss_func = nn.CrossEntropyLoss()
75 optimizer = torch.optim.SGD(cnn.parameters(), lr=learning_rate)
```

MNIST Example

Define Model

conv->relu->pool

(이 예시에는 패딩X)

cf) super는 base class
접근에 관한 것

```
47 class CNN(nn.Module):
48     def __init__(self):
49         super(CNN, self).__init__()
50         self.layer1 = nn.Sequential(
51             nn.Conv2d(1, 16, 5),    # batch x 16 x 24 x 24
52             nn.ReLU(),
53             nn.Conv2d(16, 32, 5),   # batch x 32 x 20 x 20
54             nn.ReLU(),
55             nn.MaxPool2d(2, 2)    # batch x 32 x 10 x 10
56         )
57         self.layer2 = nn.Sequential(
58             nn.Conv2d(32, 64, 5),   # batch x 64 x 6 x 6
59             nn.ReLU(),
60             nn.Conv2d(64, 128, 5),  # batch x 128 x 2 x 2
61             nn.ReLU()
62         )
63         self.fc = nn.Linear(2*2*128, 10)
64
65     def forward(self, x):
66         out = self.layer1(x)
67         out = self.layer2(out)
68         out = out.view(batch_size, -1)
69         out = self.fc(out)
70
71     return out
72
73
74 cnn = CNN().cuda()
75 loss_func = nn.CrossEntropyLoss()
76 optimizer = torch.optim.SGD(cnn.parameters(), lr=learning_rate)
```

MNIST Example

Define Model

conv->relu->pool

(이 예시에는 패딩X)

Forward 함수는 모델을 쭉 이어서 실행하는 부분. Forward라는 이름은 관습

```
47 class CNN(nn.Module):
48     def __init__(self):
49         super(CNN, self).__init__()
50         self.layer1 = nn.Sequential(
51             nn.Conv2d(1, 16, 5),    # batch x 16 x 24 x 24
52             nn.ReLU(),
53             nn.Conv2d(16, 32, 5),   # batch x 32 x 20 x 20
54             nn.ReLU(),
55             nn.MaxPool2d(2, 2)    # batch x 32 x 10 x 10
56         )
57         self.layer2 = nn.Sequential(
58             nn.Conv2d(32, 64, 5),   # batch x 64 x 6 x 6
59             nn.ReLU(),
60             nn.Conv2d(64, 128, 5),  # batch x 128 x 2 x 2
61             nn.ReLU()
62         )
63         self.fc = nn.Linear(2*2*128, 10)
64
65     def forward(self, x):
66         out = self.layer1(x)
67         out = self.layer2(out)
68         out = out.view(batch_size, -1)
69         out = self.fc(out)
70
71     return out
72
73
74 cnn = CNN().cuda()
75 loss_func = nn.CrossEntropyLoss()
76 optimizer = torch.optim.SGD(cnn.parameters(), lr=learning_rate)
```

MNIST Example

Define Model

conv->relu->pool

(이 예시에는 패딩X)

Forward 함수는 모델을 쭉 이어서 실행하는 부분. Forward라는 이름은 관습

Initialize Model

```
47 class CNN(nn.Module):
48     def __init__(self):
49         super(CNN, self).__init__()
50         self.layer1 = nn.Sequential(
51             nn.Conv2d(1, 16, 5),      # batch x 16 x 24 x 24
52             nn.ReLU(),
53             nn.Conv2d(16, 32, 5),    # batch x 32 x 20 x 20
54             nn.ReLU(),
55             nn.MaxPool2d(2, 2)     # batch x 32 x 10 x 10
56         )
57         self.layer2 = nn.Sequential(
58             nn.Conv2d(32, 64, 5),   # batch x 64 x 6 x 6
59             nn.ReLU(),
60             nn.Conv2d(64, 128, 5),  # batch x 128 x 2 x 2
61             nn.ReLU()
62         )
63         self.fc = nn.Linear(2*2*128, 10)
64
65     def forward(self, x):
66         out = self.layer1(x)
67         out = self.layer2(out)
68         out = out.view(batch_size, -1)
69         out = self.fc(out)
70
71         return out
72
73
74 cnn = CNN().cuda()
75 loss_func = nn.CrossEntropyLoss()
76 optimizer = torch.optim.SGD(cnn.parameters(), lr=learning_rate)
```

MNIST Example

Define Model

conv->relu->pool

(이 예시에는 패딩X)

Forward 함수는 모델을 쭉 이어서 실행하는 부분. Forward라는 이름은 관습

Initialize Model

Loss Function
& Optimizer

```
47 class CNN(nn.Module):
48     def __init__(self):
49         super(CNN, self).__init__()
50         self.layer1 = nn.Sequential(
51             nn.Conv2d(1, 16, 5),      # batch x 16 x 24 x 24
52             nn.ReLU(),
53             nn.Conv2d(16, 32, 5),    # batch x 32 x 20 x 20
54             nn.ReLU(),
55             nn.MaxPool2d(2, 2)      # batch x 32 x 10 x 10
56         )
57         self.layer2 = nn.Sequential(
58             nn.Conv2d(32, 64, 5),    # batch x 64 x 6 x 6
59             nn.ReLU(),
60             nn.Conv2d(64, 128, 5),   # batch x 128 x 2 x 2
61             nn.ReLU()
62         )
63         self.fc = nn.Linear(2*2*128, 10)
64
65     def forward(self, x):
66         out = self.layer1(x)
67         out = self.layer2(out)
68         out = out.view(batch_size, -1)
69         out = self.fc(out)
70
71         return out
72
73
74 cnn = CNN().cuda()
75 loss_func = nn.CrossEntropyLoss()
76 optimizer = torch.optim.SGD(cnn.parameters(), lr=learning_rate)
```

MNIST Example

```
81  for i in range(epoch):
82      for j,[image,label] in enumerate(train_loader):
83          image = Variable(image).cuda()
84          label = Variable(label).cuda()
85
86
87          optimizer.zero_grad()
88          result = cnn.forward(image)
89          loss = loss_func(result,label)
90          loss.backward()
91          optimizer.step()
92
93          if j % 100 == 0:
94              print(loss)
95
96  # Test with test data
97
98  cnn.eval()
99  correct= 0
100 total = 0
101
102 for image, label in test_loader:
103     image = Variable(image).cuda()
104     result = cnn(image).cuda()
105     _, predicted = torch.max(result.data, 1)
106     total += label.size(0)
107     correct += (predicted == label.cuda()).sum()
108
109 print('Test Accuracy of the model on the 10000 test images: %f %%' % (100 * correct / total))
```

MNIST Example

Train Model

```
81 for i in range(epoch):
82     for j,[image,label] in enumerate(train_loader):
83         image = Variable(image).cuda()
84         label = Variable(label).cuda()
85
86
87         optimizer.zero_grad()
88         result = cnn.forward(image)
89         loss = loss_func(result,label)
90         loss.backward()
91         optimizer.step()
92
93         if j % 100 == 0:
94             print(loss)
95
96 # Test with test data
97
98 cnn.eval()
99 correct= 0
100 total = 0
101
102 for image, label in test_loader:
103     image = Variable(image).cuda()
104     result = cnn(image).cuda()
105     _, predicted = torch.max(result.data, 1)
106     total += label.size(0)
107     correct += (predicted == label.cuda()).sum()
108
109 print('Test Accuracy of the model on the 10000 test images: %f %%' % (100 * correct / total))
```

MNIST Example

Train Model

```
81 for i in range(epoch):
82     for j,[image,label] in enumerate(train_loader):
83         image = Variable(image).cuda()
84         label = Variable(label).cuda()
85
86
87         optimizer.zero_grad()
88         result = cnn.forward(image)
89         loss = loss_func(result,label)
90         loss.backward()
91         optimizer.step()
92
93         if j % 100 == 0:
94             print(loss)
95
96 # Test with test data
97
98
99
100
101
102
103
104
105
106
107
108
109
```

Test Model

```
cnn.eval()
correct= 0
total = 0

for image, label in test_loader:
    image = Variable(image).cuda()
    result = cnn(image).cuda()
    _, predicted = torch.max(result.data, 1)
    total += label.size(0)
    correct += (predicted == label.cuda()).sum()

print('Test Accuracy of the model on the 10000 test images: %f %%' % (100 * correct / total))
```

Q&A
