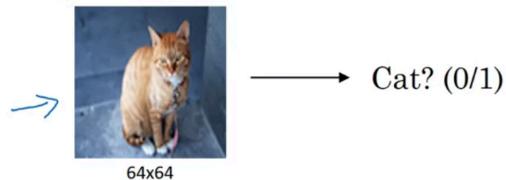


Computer vision

Computer Vision Problems

Image Classification



Neural Style Transfer

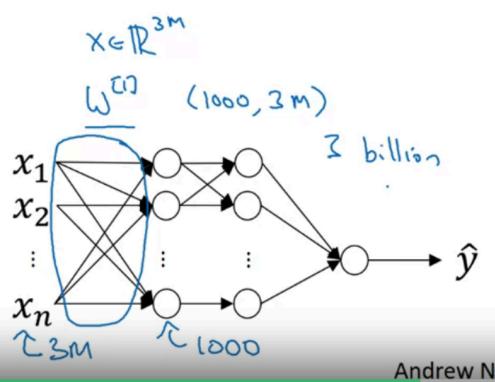
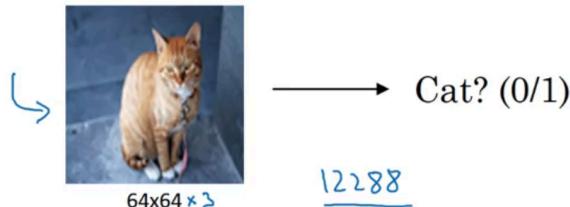


Object detection



- CV를 통해 이미지의 분류, 자율주행에서 차나 보행자 등 object detection, Neural Style Transfer처럼 새로운 작품을 만드는 등 여러 작업이 가능.

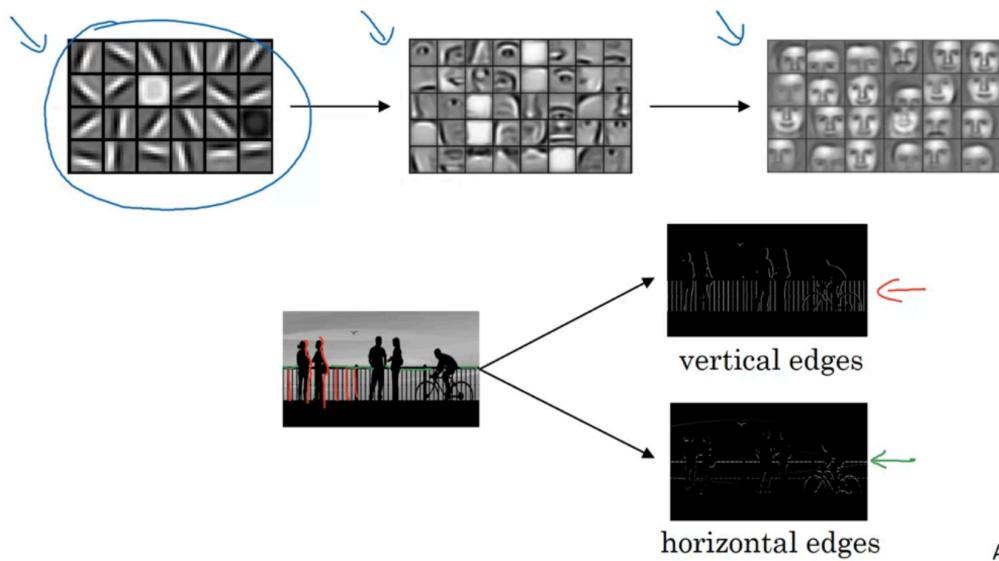
Deep Learning on large images



- 1000x1000x3 크기의 고화질 이미지가 있다면, 30만의 input을 가지게 된다. hidden layer의 unit 수는 1000이라면 $(1000, 3M)$ 차원.
- 위 경우, 파라미터의 개수가 30억개가 되며, 계산에 3억 쪽을 걸을 수 있다.

Edge detection example

Computer Vision Problem



Andrew Ng

• 뭐 학습기준에서 어떤가 Vertical Edge or Horizontal Edge인가? 강자?

• Vertical Edge

$$3 \times 1 + 1 \times 1 + 2 \times 1 + 0 \times 0 + 3 \times 0 + 7 \times 0 + 1 \times -1 + 8 \times -1 + 2 \times -1 = -5$$

3	1	0	1	-1	2	7	4
-1	5	0	-1	9	3	1	
-2	7	0	-1	5	1	3	
0	1	3	1	7	8		
4	2	1	6	2	8		
2	4	5	2	3	9		

6×6

"convolution"

$$\begin{matrix} & \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix} \\ * & \begin{matrix} -5 & 4 & 8 \\ -10 & -2 & 2 \\ 0 & -2 & -7 \end{matrix} \\ \begin{matrix} 3 \times 3 \\ filter \end{matrix} & = \begin{matrix} -5 & 4 & 8 \\ -10 & -2 & 2 \\ 0 & -2 & -7 \end{matrix} \\ & \begin{matrix} 4 \times 4 \end{matrix} \end{matrix}$$

6×6 학습기준을 따,

3×3 filter를 사용하여

적용된다.

이를 적용하면

4×4 학습기준을 사용.

Vertical edge detection

10	10	10	0	0	0	0
10	10	10	0	0	0	0
10	10	10	0	0	0	0
10	10	10	0	0	0	0
10	10	10	0	0	0	0
10	10	10	0	0	0	0

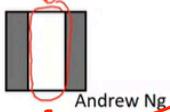
6×6

$$\begin{matrix} & \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix} \\ * & \begin{matrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{matrix} \\ \begin{matrix} 3 \times 3 \\ filter \end{matrix} & = \begin{matrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{matrix} \\ & \begin{matrix} 4 \times 4 \end{matrix} \end{matrix}$$

3×3 vertical edge detection filter를 사용

4×4 학습기준을 적용할 수 있다.

이 부분에 의해
vertical edge 가능.



Andrew Ng

More Edge Detection

$$\begin{matrix}
 0 & 0 & 0 & 10 & 10 & 10 \\
 0 & 0 & 0 & 10 & 10 & 10 \\
 0 & 0 & 0 & 10 & 10 & 10 \\
 0 & 0 & 0 & 10 & 10 & 10 \\
 0 & 0 & 0 & 10 & 10 & 10 \\
 0 & 0 & 0 & 10 & 10 & 10
 \end{matrix}$$

→

*

$$\begin{matrix}
 1 & 0 & -1 \\
 1 & 0 & -1 \\
 1 & 0 & -1
 \end{matrix}$$

=

$$\begin{matrix}
 0 & -30 & -30 & 0 \\
 0 & -30 & -30 & 0 \\
 0 & -30 & -30 & 0 \\
 0 & -30 & -30 & 0
 \end{matrix}$$

위와 반대로, 왼쪽이 짙은고, 오른쪽이 밝은 이미지라면 어떻게 될까?

→ 이는 negative edge, 위의 경우 positive edge.

- CONV 연산 값이 $-30 \rightarrow$ 이것은 edge가 아웃풋 색상에서 100% 빠져나온 것을 감지, 만약 positive or negative를 신경하지 않는다면, output matrix의 절대값을 사용하면 된다.

Vertical and Horizontal Edge Detection

→

$$\begin{matrix}
 1 & 0 & -1 \\
 1 & 0 & -1 \\
 1 & 0 & -1
 \end{matrix}$$

Vertical

$$\begin{matrix}
 1 & 1 & 1 \\
 0 & 0 & 0 \\
 -1 & -1 & -1
 \end{matrix}$$

Horizontal

→ horizontal edge detection.

$$\begin{matrix}
 10 & 10 & 10 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 \\
 0 & 0 & 0 & 10 & 10 & 10 \\
 0 & 0 & 0 & 10 & 10 & 10 \\
 0 & 0 & 0 & 10 & 10 & 10
 \end{matrix}$$

*

$$\begin{matrix}
 1 & 1 & 1 \\
 0 & 0 & 0 \\
 -1 & -1 & -1
 \end{matrix}$$

=

$$\begin{matrix}
 0 & 0 & 0 & 0 \\
 30 & 10 & -10 & -30 \\
 30 & 10 & -10 & -30 \\
 0 & 0 & 0 & 0
 \end{matrix}$$

Learning to detect edges

1	0	-1
1	0	-1
1	0	-1

→

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

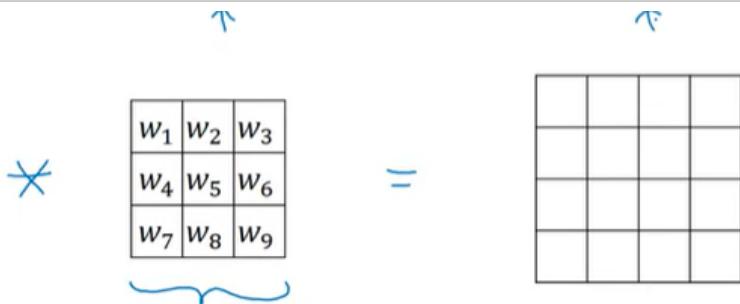
Sobel filter

3	0	-3
10	0	-10
3	0	-3

Scharr filter

3x3 filter를 사용시, 어떤 숫자를 택할 수 있다. 이는
딥러닝 학습을 통해 이 값을 알 수 있다. 즉, 9개의 parameter를
갖는다.

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9



그렇기 때문에, 어느 번경의 edge detection 를 데이터로 하면
학습할 수 있다. (이것은 low-level의 feature 가 된다)

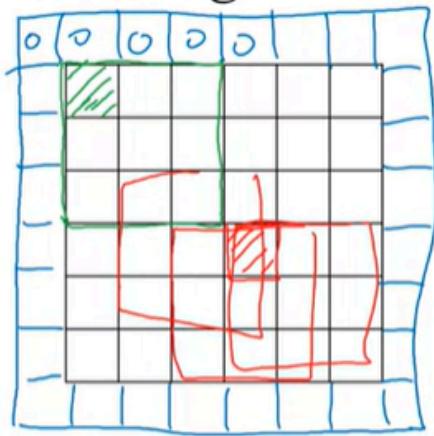
∴ CNN의 장점은 어떤 특징 filter를 선택하는 것인 듯하다.

학습을 통해 최적의 filter(일종의 weights)를 얻는 것!

Padding

- 위에서 6×6 이미지에 3×3 filter를 적용했을 때, output matrix는 4×4 ($n \times n$) image \star ($f \times f$) filter $\rightarrow (n-f+1) \times (n-f+1)$ 의 output
- 이 부분에서 2가지 단점 존재.
 1. Convolution 연산을 수행할 때마다 이미지가 축소된다. 계속해서 작아지면 conv 연산의 횟수가 제한적.
 2. 코너나 모서리에 존재하는 pixels들은 많이 사용되지 않는다.
- 이 문제를 해결하기 위해 우리는 정계에 이미지를 채워서 (padding) Conv 연산을 이미지 전체에 적용.

Padding



- shrink output
- throw away info from edge

*

$$\begin{matrix} & & \\ & & \\ & & \\ & & \end{matrix}$$

3×3
 $f \times f$

=

$$\begin{matrix} & & \\ & & \\ & & \\ & & \\ & & \end{matrix}$$

6×6

- 위 그림과 같이, 6×6 이미지에 padding을 추가해서 3×3 filter를 적용하여 conv 연산을 수행하면 output matrix는 원래 이미지와 동일하게 6×6 matrix인 것을 수 있다.
- padding의 정도를 p 라고 한다면, 아래와 같이 공식으로 크기가 결정된다.
 $(n+2p, n+2p) \text{ image } \star (f, f) \text{ filter} = (n+2p-f+1, n+2p-f+1) \text{ output.}$
- Padding의 존재에 따라 두 가지 유형의 Convolution이 있다.
Valid Convolution, Same Convolution

- Valid Convolution: 기본적으로 padding 없이 conv 연산.

- Same Convolution: Padding을 추가해서 input size가 output size와 동일하도록 conv 연산.

Valid and Same convolutions

↑ no padding

"Valid": $n \times n \star f \times f \rightarrow \frac{n-f+1}{4} \times \frac{n-f+1}{4}$

output matrix: $(n+2p-f+1, n+2p-f+1)$

$$n = n+2p-f+1 \Rightarrow p = \frac{f-1}{2}$$

"Same": Pad so that output size is the same as the input size.

$$\begin{aligned} n+2p-f+1 &= n+2p-f+1 & f \text{ is usually odd} \\ n+2p-f+1 &= n \Rightarrow p = \frac{f-1}{2} \\ 3 \times 3 & \quad p = \frac{3-1}{2} = 1 & \boxed{S \times S} \\ & \quad \boxed{E \times E} & p=2 \end{aligned}$$

- 전형적으로 f 는 거의 홀수, 만약 f 가 짝수라면 비대칭 padding을 해야함.

Strided convolutions

- Strided Convolution은 CNN에서 사용되는 기본적인 building block 중 하나.
7x7 이미지를 3x3 filter로 conv 연산 수행 + stride = 2, 하면 아래와 같이 작동.

Strided convolution

2	3	7	3	4	4	6	4	2	9
6	6	9	1	8	0	7	2	4	3
3	4	8	-1	3	0	8	3	9	7
7	8	3	6	6	6	3	4		
4	2	1	8	3	4	6			
3	2	4	1	9	8	3			
0	1	3	9	2	1	4			

$$\begin{matrix} 2 & 3 & 7 \\ 6 & 6 & 9 \\ 3 & 4 & 8 \\ 7 & 8 & 3 \\ 4 & 2 & 1 \\ 3 & 2 & 4 \\ 0 & 1 & 3 \end{matrix} \quad * \quad \begin{matrix} 3 & 4 & 4 \\ 1 & 0 & 2 \\ -1 & 0 & 3 \end{matrix} = \begin{matrix} 91 & 100 \\ \vdots & \vdots \end{matrix}$$

3×3
 $\text{stride} = 2$

- Input 과 Output 크기는 아래 공식에 따라 결정 (padding과 stride가 있으면)

$$(n \times n) * (f \times f) = \left(\left[\frac{n+2p-f}{s} + 1 \right], \left[\frac{n+2p-f}{s} + 1 \right] \right)$$

여기서 $p=0$, $s=2$ 이기 때문에 $(3,3)$ 크기의 output을 가진다.

* 수학적 의미의 Convolution과 CNN에서의 Convolution은 조금 다른 방식.

Convolution in math textbook:

2	3	7	4	6	2
6	6	9	8	7	4
3	4	8	3	8	9
7	8	3	6	6	3
4	2	1	8	3	4
3	2	4	1	9	8

$$\begin{matrix} 2 & 3 & 7 \\ 6 & 6 & 9 \\ 3 & 4 & 8 \\ 7 & 8 & 3 \\ 4 & 2 & 1 \\ 3 & 2 & 4 \end{matrix} \quad * \quad \begin{matrix} 3 & 4 & 5 \\ 1 & 0 & 2 \\ -1 & 9 & 7 \end{matrix} = \begin{matrix} 9 & 10 & 11 \\ 14 & 15 & 16 \\ 19 & 20 & 21 \end{matrix}$$

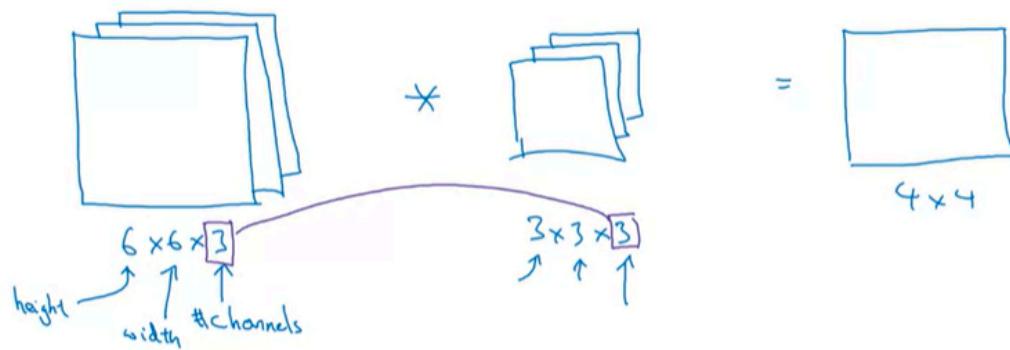
$(A * B) * C = A * (B * C)$

- 수학적 의미에서의 Convolution은 모소간의 공유를 진행하기 전에, filter를 상하좌우 반전 시키는 과정이 추가 된다. 따라서 CNN에서 수행하는 Conv 연산은 cross-correlation에 해당하지만, 딥러닝에서는 그냥 Convolution이라 칭한다

- 위와 같이 filter를 반전시키는 방식은 신호처리 분야에서는 적합하지만, 딥러닝과 크게 상관 없기 때문에 생략하고, 진행 가능.

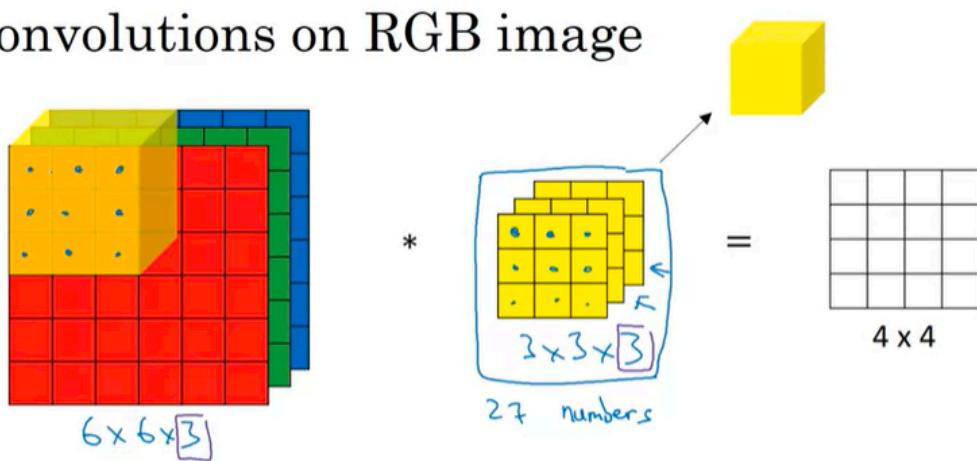
Convolutions over volume

Convolutions on RGB images



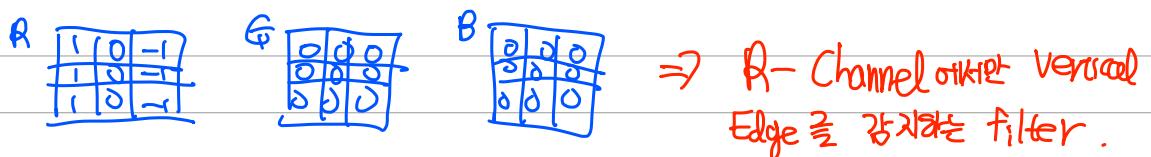
- RGB Channel을 가지고 있는 이미지를 convolution 연산하면, 연산에 사용되는 filter의 channel은 input 이미지의 channel과 동일해야 한다.
- 이렇게 연산을 하면 output은 $4 \times 4 \times 1$ 이 된다.

Convolutions on RGB image

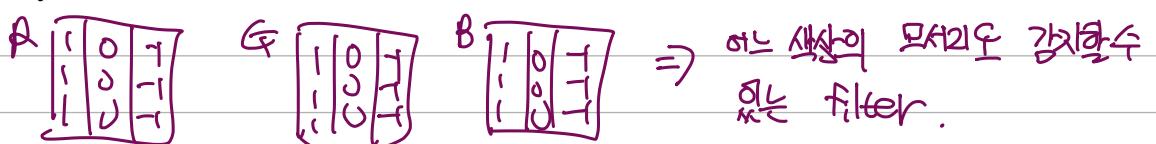


- input 과 filter를 각각의 채널에서 요소간의 곱을 수행.
→ 각 채널마다 9개의 결과값이 나오고 총 27개의 결과값이 나온다.
→ 이 27개의 값을 모두 더해주어 최종 output.

- 만약 R-Channel에서만 Edge를 감지하고 싶다면 R-Channel에 해당하는 filter만 채운고, 나머지 B, G Channel에 해당하는 filter를 모두 0으로 채운다.



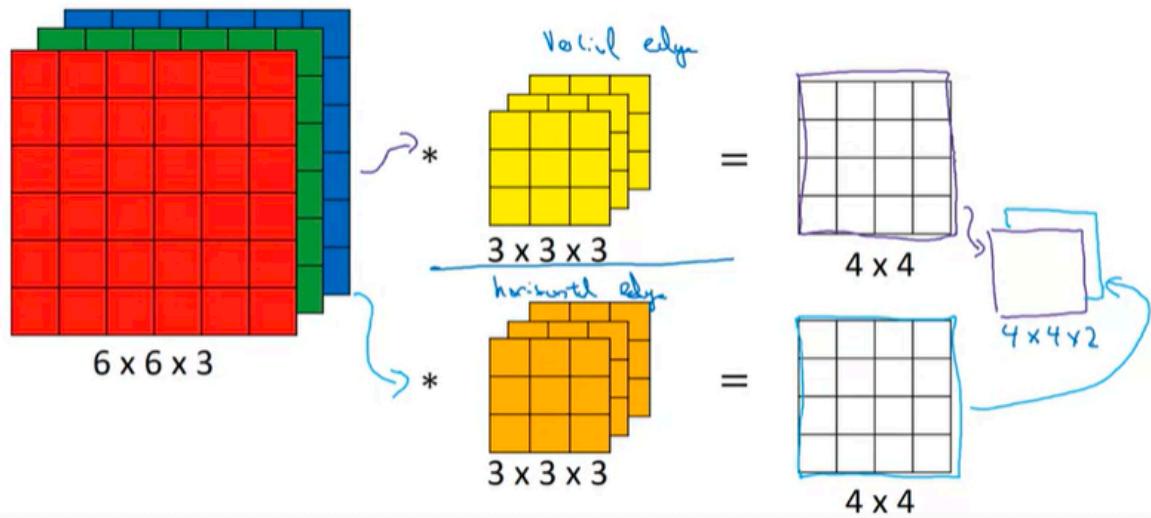
- Vertical Edge의 색상을 신경쓰지 않는다면.



Multiple filters

- 동시에 여러 filter를 사용하고 싶다면?

Multiple filters

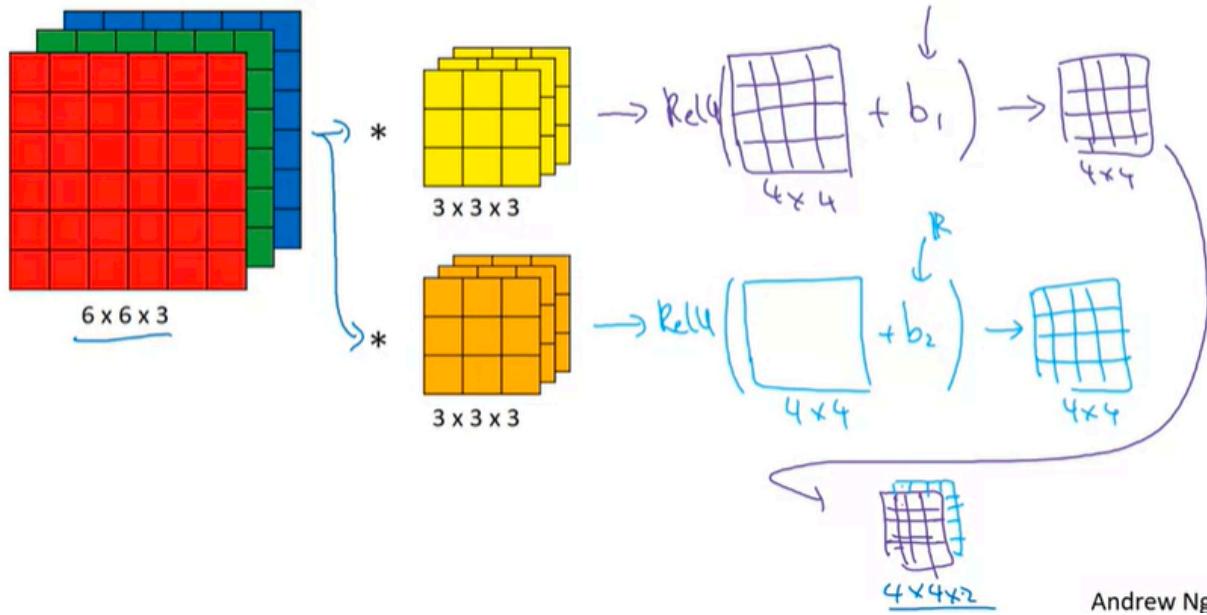


- 2개의 filter (위에서는 Vertical, horizontal)에 대해 Conv 연산을 수행하고. 결과 matr;x를 쌓는다. $\Rightarrow 4 \times 4 \times 2$ matr;x.

$$(n \times n \times n_c) * (f \times f \times n_c) \rightarrow (n-f+1) \times (n-f+1) \times \underbrace{n_c}_{\text{filter의 수}}$$

One layer of a convolution network

Example of a layer



- CNN에서 하나의 layer

$$z^{[l]} = w^{[l]} a^{[l]} + b^{[l]}$$

$$a^{[l]} = g(z^{[l]})$$

- $a^{[l]}$ 은 input x , filter들이 $w^{[l]}$ 과 같은 역할.
- 위 예제에서는 2개의 filter를 사용하기 때문에 output은 $4 \times 4 \times 2$.

* 만약 CNN Layer에서 $3 \times 3 \times 3$ filter 10개를 사용한다면, 이 layer의 parameter 개수는 ?

- 하나의 filter에는 $3 \times 3 \times 3 = 27$ 개의 parameter + 1개의 bias $\Rightarrow 28$ 개의 parameter
- 총 10개의 filter $\Rightarrow 28 \times 10 = 280$ 개의 parameter 존재

\Rightarrow CNN의 장점 : Input의 크기가 아무리 커더라도, Parameter의 수는 280개로 고정.

- Input : $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$ \Rightarrow 매우 큰 이미지가 있어도 overfitting 하지 않는다.

- Output : $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

- $n_H^{[l]} = \left\lfloor \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$

- $n_W^{[l]} = \left\lfloor \frac{n_W^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$

- Each filter : $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$, $n_c^{[l-1]}$: 이전 layer output의 $n_c^{[l]}$ 와 동일

- Activations : $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

m개의 example이 있다면, activation, weight, bias는 다음과 같이 나타낼 수 있다.

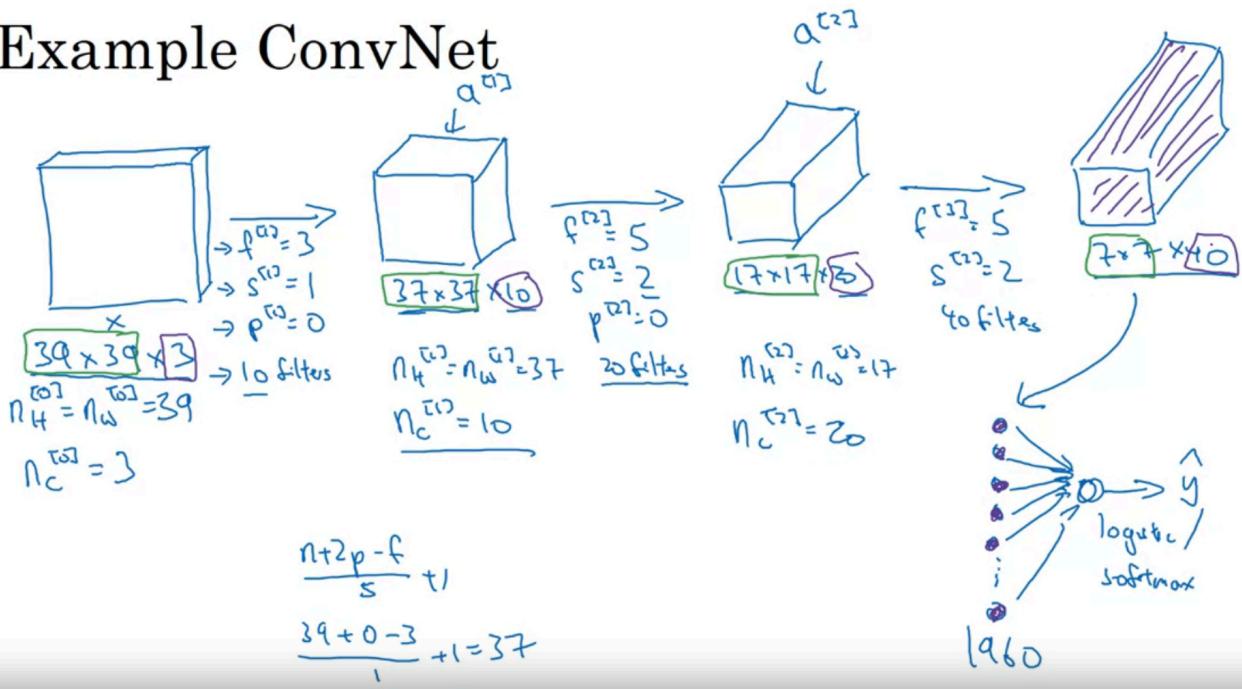
- $A^{[l]} : m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

- weights : $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$, $n_c^{[l-1]}$ 개의 filter 존재

- bias : $n_c^{[l]}$, 실수이므로 차원으로 표현하, $1 \times 1 \times 1 \times n_c^{[l]}$ 가 된다.

Simple Convolution Neural Network Example

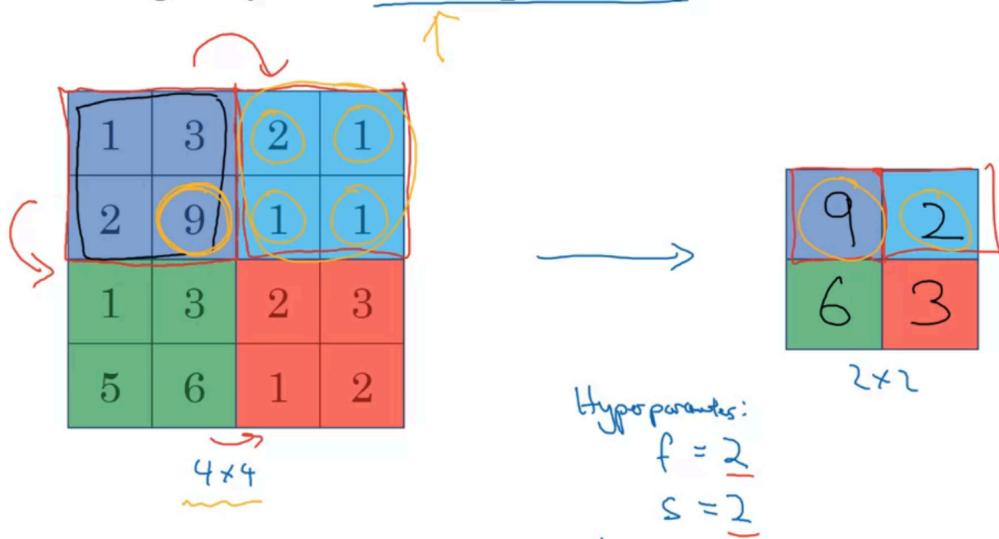
Example ConvNet



Pooling layers

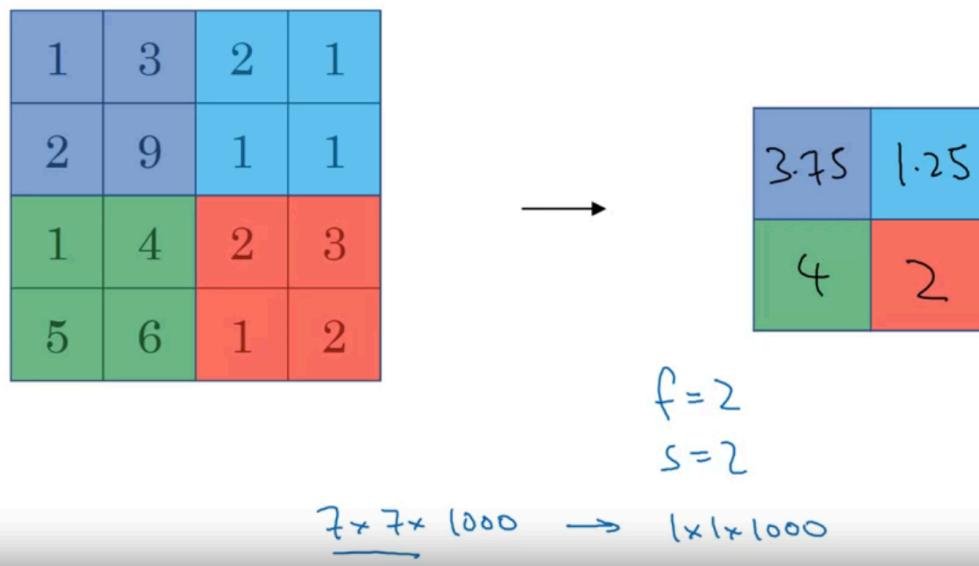
- Pooling layer를 사용하면, 표현 크기를 줄임으로써 계산 속도를 높이고 특성을 잘 강조할 수 있다.

Pooling layer: Max pooling



- 각 칸에서 최대값만 가져온다.

Pooling layer: Average pooling



- 평균값을 적용.

→ stride convolution과 stride 때문에 resolution 줄어든다.

- (Stride convolution)과 Pooling의 차이점은 Stride Convolution에는 학습해야하는 parameter가 존재하나, Pooling은 parameter가 없다.

Why convolutions?

Why convolutions

The diagram shows a 6x6 input matrix and a 3x3 kernel matrix. The input matrix has values 10 and 0 in its first two columns, and 30 in its last three columns. The kernel matrix has values 1, 0, -1 in each row. A green bracket on the left indicates a stride of 2. The result of the convolution is a 4x4 output matrix. The top-left cell of the output matrix is circled in green and highlighted in green, while the bottom-right cell is circled in red. Handwritten annotations above the output matrix say "translation invariance".

10	10	10	0	0	0	0
10	10	10	0	0	0	0
10	10	10	0	0	0	0
10	10	10	0	0	0	0
10	10	10	0	0	0	0
10	10	10	0	0	0	0

1	0	-1
1	0	-1
1	0	-1

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

Parameter sharing: A feature detector (such as a vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.

→ **Sparsity of connections:** In each layer, each output value depends only on a small number of inputs.

- Parameter Sharing

- Parameter 를 줄일 수 있다.
- Vertical Edge Detector처럼 image의 한 부분에 useful한 어떤 feature detector가 image의 다른 부분에도 useful 할 수 있다.

- Sparsity of connections.

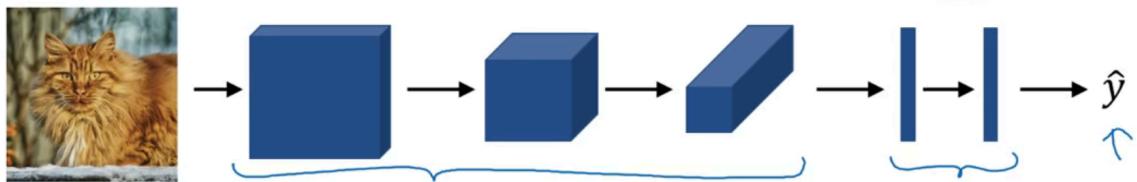
- 각 layer의 각 output 값은 적은 수의 input에만 의존.
- 각 output이 영향을 미치는 input 사이의 관계성이 낮다는 의미.

Putting it together

Putting it together

Training set $(x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})$.

$\underline{\omega}, \underline{b}$



$$\text{Cost } J = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

Use gradient descent to optimize parameters to reduce J

- Cost J 에는 parameter의 gradient를 빼어 계산.
- J 를 줄이기 위해 gradient descent로 parameter를 optimize.

