

[1주차] Foundations of Convolutional Neural Networks

Edge Detection Example

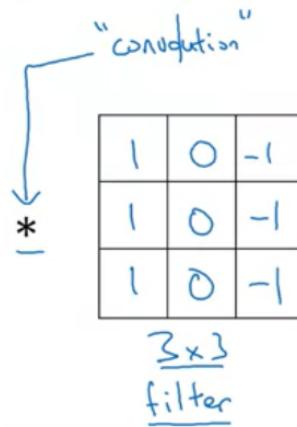
이미지 속의 수직 테두리 감지

Vertical edge detection

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

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

6x6



=

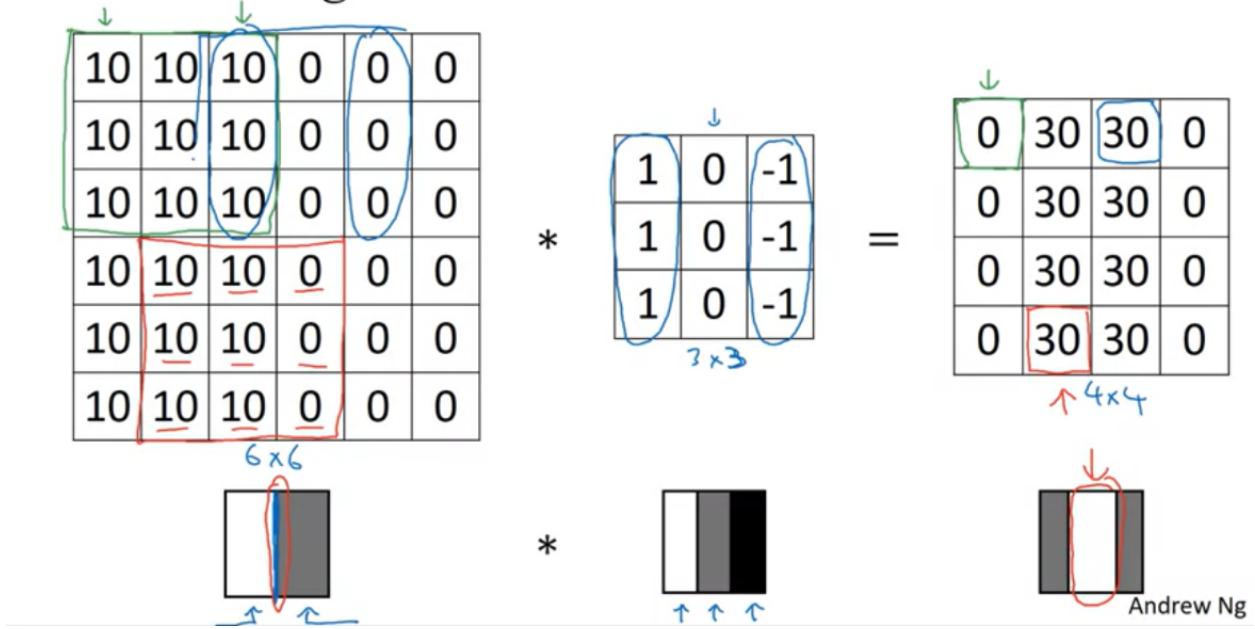
-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

4x4

python: conv-forward
tensorflow: tf.nn.conv2d
keras: Conv2D

Andrew Ng

Vertical edge detection



10이 있는 왼쪽 절반은 더 밝은 픽셀 집약 값을 가지고 있고 오른쪽 절반은 더 어두운 픽셀 집약 값을 가지고 있습니다. 가운데에 바로 매우 강한 vertical edge가 있습니다. 흰색에서 검정 혹은 흰색에서 더 어두운 색으로 전환되기 때문이죠.

이 예시에서, 가운데에 있는 밝은 영역이 바로 아웃풋 이미지인데요, 마치 이미지 중간에 강한 수직 가장자리가 있는 것처럼 보입니다. 이는 아마도 vertical edge detection으로부터 생각해볼 때, 우리가 3×3 필터를 사용하고 있기 때문에 vertical edge가 3×3 영역이라는 점입니다.

More Edge Detection

밝음에서 어두운 에지로의 변화와 어두움에서 밝은 에지로의 변화되는 차이점을 알아보도록 하겠습니다.

Vertical edge detection examples

$$\begin{array}{c}
 \begin{array}{ccccccc}
 10 & 10 & 10 & 0 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 & 0
 \end{array} \\
 \xrightarrow{\quad\quad\quad} \boxed{\quad\quad\quad}
 \end{array}
 \begin{array}{c} * \\ \quad\quad\quad \end{array}
 \begin{array}{c}
 \begin{array}{ccc}
 1 & 0 & -1 \\
 1 & 0 & -1 \\
 1 & 0 & -1
 \end{array} \\
 \boxed{\quad\quad\quad}
 \end{array}
 \begin{array}{c} = \\ \quad\quad\quad \end{array}
 \begin{array}{c}
 \begin{array}{cccc}
 0 & 30 & 30 & 0 \\
 0 & 30 & 30 & 0 \\
 0 & 30 & 30 & 0 \\
 0 & 30 & 30 & 0
 \end{array} \\
 \boxed{\quad\quad\quad}
 \end{array}$$

$$\begin{array}{c}
 \begin{array}{ccccccc}
 0 & 0 & 0 & 10 & 10 & 10 & 10 \\
 0 & 0 & 0 & 10 & 10 & 10 & 10 \\
 0 & 0 & 0 & 10 & 10 & 10 & 10 \\
 0 & 0 & 0 & 10 & 10 & 10 & 10 \\
 0 & 0 & 0 & 10 & 10 & 10 & 10 \\
 0 & 0 & 0 & 10 & 10 & 10 & 10
 \end{array} \\
 \xrightarrow{\quad\quad\quad} \boxed{\quad\quad\quad}
 \end{array}
 \begin{array}{c} * \\ \quad\quad\quad \end{array}
 \begin{array}{c}
 \begin{array}{ccc}
 1 & 0 & -1 \\
 1 & 0 & -1 \\
 1 & 0 & -1
 \end{array} \\
 \boxed{\quad\quad\quad}
 \end{array}
 \begin{array}{c} = \\ \quad\quad\quad \end{array}
 \begin{array}{c}
 \begin{array}{cccc}
 0 & -30 & -30 & 0 \\
 0 & -30 & -30 & 0 \\
 0 & -30 & -30 & 0 \\
 0 & -30 & -30 & 0
 \end{array} \\
 \boxed{\quad\quad\quad}
 \end{array}$$

Andrew Ng

전환되는 명암이 이렇게 반대로 되기 때문에, 이제 30 또한 반대로 뒤바뀌게 되는 것입니다.
그리고 -30은 밝은 색에서 어둠으로의 변화가 아니라 어두운 색에서 밝은 색으로 변화하는
것이죠.

Vertical and Horizontal Edge Detection

$$\begin{array}{c}
 \begin{array}{ccc}
 1 & 0 & -1 \\
 1 & 0 & -1 \\
 1 & 0 & -1
 \end{array} \\
 \xrightarrow{\quad\quad\quad} \boxed{\quad\quad\quad}
 \end{array}
 \begin{array}{c} \text{Vertical} \\ \quad\quad\quad \end{array}
 \qquad\qquad\qquad
 \begin{array}{c}
 \begin{array}{ccc}
 1 & 1 & 1 \\
 0 & 0 & 0 \\
 -1 & -1 & -1
 \end{array} \\
 \xrightarrow{\quad\quad\quad} \boxed{\quad\quad\quad}
 \end{array}
 \begin{array}{c} \text{Horizontal} \\ \quad\quad\quad \end{array}$$

$$\begin{array}{c}
 \begin{array}{ccccccc}
 10 & 10 & 10 & 0 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 10 & 10 & 10 & 10 \\
 0 & 0 & 0 & 10 & 10 & 10 & 10 \\
 0 & 0 & 0 & 10 & 10 & 10 & 10
 \end{array} \\
 \xrightarrow{\quad\quad\quad} \boxed{\quad\quad\quad} \quad 6 \times 6
 \end{array}
 \begin{array}{c} * \\ \quad\quad\quad \end{array}
 \begin{array}{c}
 \begin{array}{ccc}
 1 & 1 & 1 \\
 0 & 0 & 0 \\
 -1 & -1 & -1
 \end{array} \\
 \boxed{\quad\quad\quad}
 \end{array}
 \begin{array}{c} = \\ \quad\quad\quad \end{array}
 \begin{array}{c}
 \begin{array}{cccc}
 0 & 0 & 0 & 0 \\
 30 & 10 & -10 & -30 \\
 30 & 10 & -10 & -30 \\
 0 & 0 & 0 & 0
 \end{array} \\
 \boxed{\quad\quad\quad}
 \end{array}$$

Andrew Ng

몇 가지 예시를 보자면, 여기 있는 이 30은 이 3×3 영역에 대응하는데요, 실제로 위쪽엔 밝은 픽셀이 있고 아래쪽엔 어두운 픽셀들이 있습니다. 바로 여기죠. 강한 포지티브 에지를 만들어내는 것입니다. 그리고 여기에 이 -30은 이 영역에 대응하게 되는데요, 아래쪽은 더 밝고, 위쪽은 어둡게 되는 것이죠 그리고 이게 바로 네거티브 에지의 예시입니다. 다시 말해서, 우리가 6×6 같은 상대적으로 작은 이미지들로 작업하고 있다는 사실을 보여주는 것입니다. 하지만 예를 들어 -10같은 이 중간 값들은 왼쪽에 포지티브 에지 부분을 담아내고 오른쪽에 네거티브 에지를 담아낸다는 사실을 보여주는 것입니다. 따라서 그것들을 함께 섞으면 중간 값을 얻게 되는 것이죠. 하지만 이게 매우 큰, 체크 패턴을 가진 1000×1000 정도의 이미지라면, 이런 10의 전환영역은 보이지 않습니다. 중간 값은 이미지 사이즈에 비해 꽤 작을 겁니다.

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

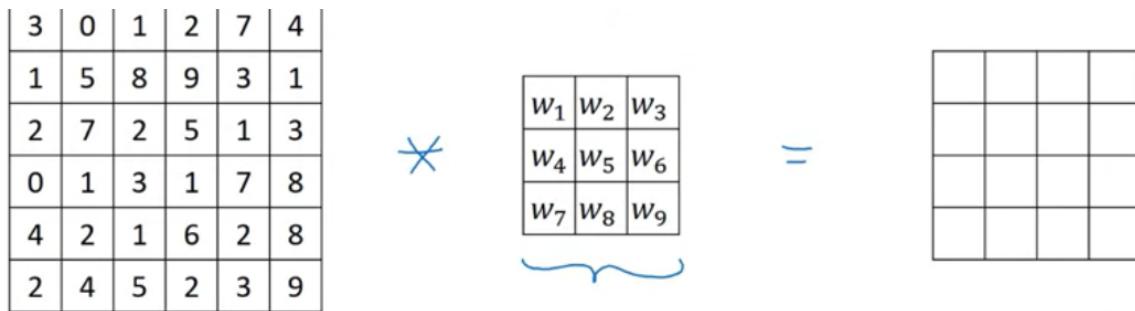
여러분이 사용할 수 있는 또 다른 것은, 아마도 1, 2 1, 0, 0, 0, -1, -2, -1 일겁니다. 이건 **Sobel Filter**라고 불리는 겁니다.

[장점]

중앙 열, 중앙픽셀에 좀 더 무게를 두어 좀 더 견고하게 만든다는 것입니다.

그러나 컴퓨터 비전 연구원들은 또 다른 세트의 숫자들도 사용하겠죠. 1, 2, 1, 대신에 3, 10, 3, 이렇게 될 수 있을 겁니다, 그렇죠? 그리고 -3, -10, -3 이렇게요. 그리고 이건 **Scharr Filter**라는 겁니다.

이건 vertical edge detection만을 위한 겁니다. 그리고 이걸 90도 돌려보시면, horizontal edge detection을 볼 수 있죠.



Andrew Ng

딥러닝이 떠오르면서, 우리가 배운 것 중의 하나는 복잡한 이미지에서 모서리들을 감지할 때, 컴퓨터 비전 연구원들이 이 9개의 숫자를 택하게 하지 않아도 된다는 겁니다. 여러분은 그냥 그걸 배워서 이 매트릭스의 9개의 숫자들을 파라미터로 처리하면, 백 프로퍼게이션 (역전파)을 사용하는 것을 배울 수 있을 테니까요. 목표는 9개의 파라미터를 배워서 6×6 이미지를 얻어낼 때, 그리고 그걸 3×3 필터와 컨볼브해서 좋은 edge detector를 만들어내도록 하는 거죠.

Padding

[기존 문제점]

1. 각 레이어마다 이미지 크기가 줄어든다면, 100 레이어 이후에는 매우 작은 이미지만 남게 될 것
2. 이미지의 가장자리에서 많은 정보를 버리고 있다는 것

[용어]

Valid and Same convolutions

$\xrightarrow{n \rightarrow \text{padding}}$

“Valid”: $n \times n$ \times $f \times f$ $\rightarrow \frac{n-f+1}{f} \times \frac{n-f+1}{f}$

$6 \times 6 \quad \times \quad 3 \times 3 \quad \rightarrow \quad 4 \times 4$

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

$$\begin{aligned} & n + 2p - f + 1 = n + 2p - f + 1 \\ & n + 2p - f + 1 = n \Rightarrow p = \frac{f-1}{2} \\ & 3 \times 3 \quad p = \frac{3-1}{2} = 1 \quad \left| \begin{array}{c} S \times S \\ f = S \end{array} \right. \quad p = 2 \end{aligned}$$

Andrew Ng

1. Valid \Rightarrow 패딩이 없는 것
2. Same \Rightarrow 아웃풋 size = 인풋 size가 되도록

[filter size]

컴퓨터 비전 분야의 관습에 따르면 f 는 주로 홀수이며, 사실 대부분 항상 홀수입니다, 짝수로 된 필터는 보기 어렵습니다.

이유 1. f 가 짝수이면 몇 가지 비대칭적인 패딩을 해야함

이유 2. 홀수 차원의 필터를 사용한다면 예를 들어 3×3 이나 5×5 와 같이 중앙 포지션을 가지고 있고 때로는 컴퓨터 비전분야에 있어서 특징점을 가지고 있다

Strided Convolution

[계산법]

Strided convolution

$\begin{matrix} 2 & 3 & 7 & 4 & 6 & 2 & 9 \\ 6 & 6 & 9 & 8 & 7 & 4 & 3 \\ 3 & 4 & 8 & 3 & 8 & 9 & 7 \\ 7 & 8 & 3 & 6 & 6 & 3 & 4 \\ 4 & 2 & 1 & 8 & 3^3 & 4^4 & 6^4 \\ 3 & 2 & 4 & 1 & 9^1 & 8^0 & 3^2 \\ 0 & 1 & 3 & 9 & 2^{-1} & 1^0 & 4^3 \end{matrix}$

 $\boxed{7 \times 7}$

$\begin{matrix} 3 & 4 & 4 \\ 1 & 0 & 2 \\ -1 & 0 & 3 \end{matrix}$
 $=$
 $\begin{matrix} 91 & 100 & 83 \\ 69 & 91 & 127 \\ 44 & 72 & 74 \end{matrix}$

 $\boxed{3 \times 3}$

$\text{stride} = 2$
 $\lfloor \frac{?}{2} \rfloor = \text{floor}(\frac{?}{2})$

$n \times n$ * $f \times f$
 padding p stride s
 $s=2$

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

$$\frac{7+0-3}{2} + 1 = \frac{4}{2} + 1 = 3$$

Andrew Ng

$N \times N$ 이미지가 있는 경우 F 와 F 필터로 컨벌루션됩니다. 그리고 패딩 P 와 스트라이드 S 를 사용하는 경우, 이 예제에서 $S=2$ 이고, $N+2P$ 에서 F 를 뺀 아웃풋이 나오게 되는 거죠. 이제 S 단계를 거치므로, 한 번에 한 칸 옮기고, S 로 나눈 다음 1을 더해주세요. 그런 다음 똑같은 방식을 적용하면 되겠습니다. 예시에서, $7+0-3$ 가 되고, 스트라이드 2로 나눈 다음 1을 더하면 2분의 $4+1$, 즉 3이 됩니다. 이런 방식으로 3×3 아웃풋으로 결론 낼 수 있는 겁니다.

+아웃풋 크기를 계산해내는 올바른 방법은 $N+2P-f$ 를 S 로 나누었을 때 정수가 아닌 경우 내림 하는 것입니다.

Summary of convolutions

$n \times n$ image $f \times f$ filter

padding p stride s

Output size:

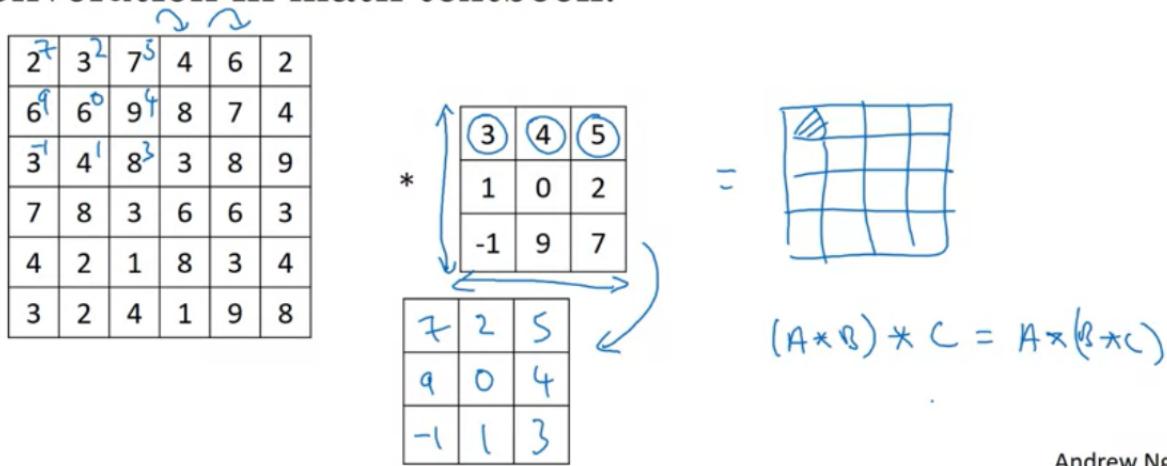
$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \quad \times \quad \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$



Andrew Ng

Technical note on cross-correlation vs. convolution

Convolution in math textbook:



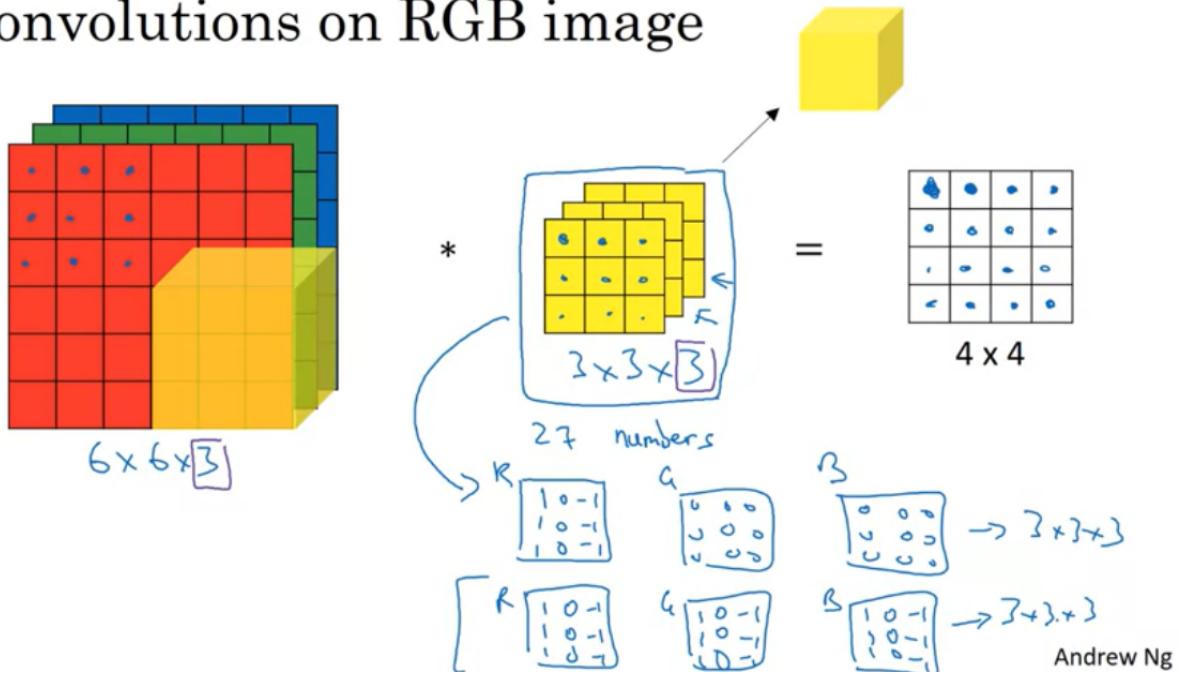
Andrew Ng

신호 처리나 특정 수학 분야에서 컨벌루션 연산과 컨볼루션 정의대로 뒤집기를 하면 $A * B$ 먼저 컨벌브하고 C 와 컨벌브하면 A 를 컨벌브 된 B 와 C 에 나중에 컨벌브하는 것과 같다는 것이 밝혀졌습니다. 이것은 일부 신호 처리 애플리케이션에 적합하지만 심층 신경망의 경우에는

그다기 상관이 없기 때문에 이 이중 mirroring operation을 생략해서 코드를 단순화시키고, 신경망도 제대로 작동하게 할 수 있습니다.

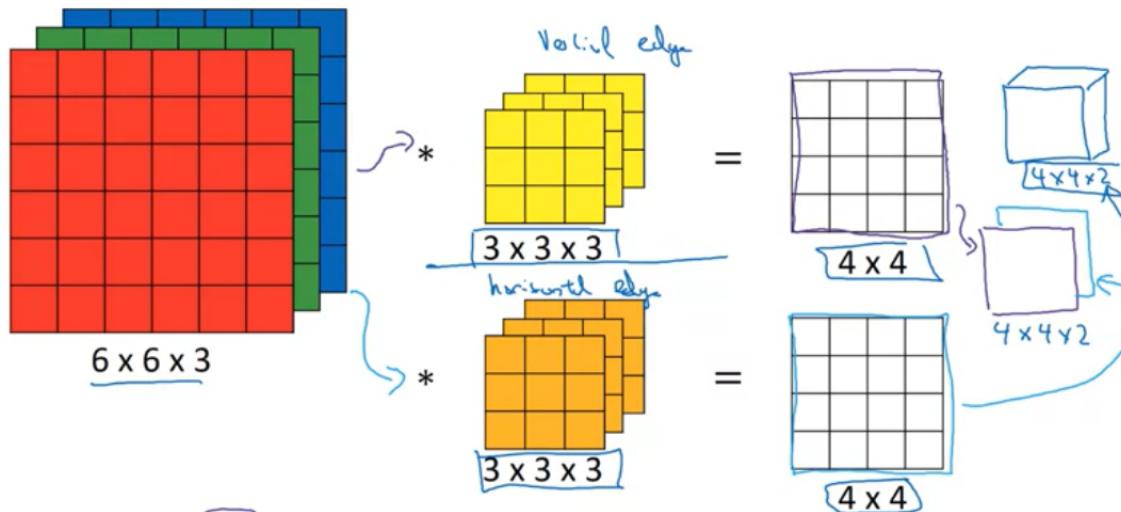
3차원 이상에서 컨볼루션 구현

Convolutions on RGB image



그래서, 이렇게 하면 여러분이 할 수 있게 되는 건 무엇일까요? 예시를 하나 들어보죠. 이건 $3 \times 3 \times 3$ 필터죠. 이미지의 빨간 채널에서 모서리를 감지하고 싶다면, 첫 번째 필터에 1, 1, 1, -1, -1, -1 이렇게 하던 대로 하시면 됩니다. 그리고 녹색 채널은 모두 0이 되어야 하고, 파란색 채널도 모두 0으로 채웁니다. $3 \times 3 \times 3$ 크기의 필터를 만들기 위해서 이 세 가지를 함께 묶여두면, 이는 빨간색 채널에서만 수직 가장자리를 감지하는 필터가 될 것입니다. 또는 수직 모서리의 색상을 신경 쓰지 않으면 이와 비슷한 필터를 사용할 수 있습니다. 반면에 1, 1, 1 -1, -1, -1 세 가지 채널 모두 다 이렇게 될 수 있습니다, 따라서 이 두 번째 대안을 설정하여, 파라미터를 설정하십시오. 그럼 edge detector를 생성할 수 있게 되고, 이 $3 \times 3 \times 3$ edge detector는 어느 색상의 모서리도 감지할 수 있게 됩니다.

Multiple filters



$$\text{Summary: } n \times n \times n_c \times f \times f \times n_c \rightarrow \frac{n-f+1}{4} \times \frac{n-f+1}{4} \times n'_c \uparrow \# \text{filters}$$

Andrew Ng

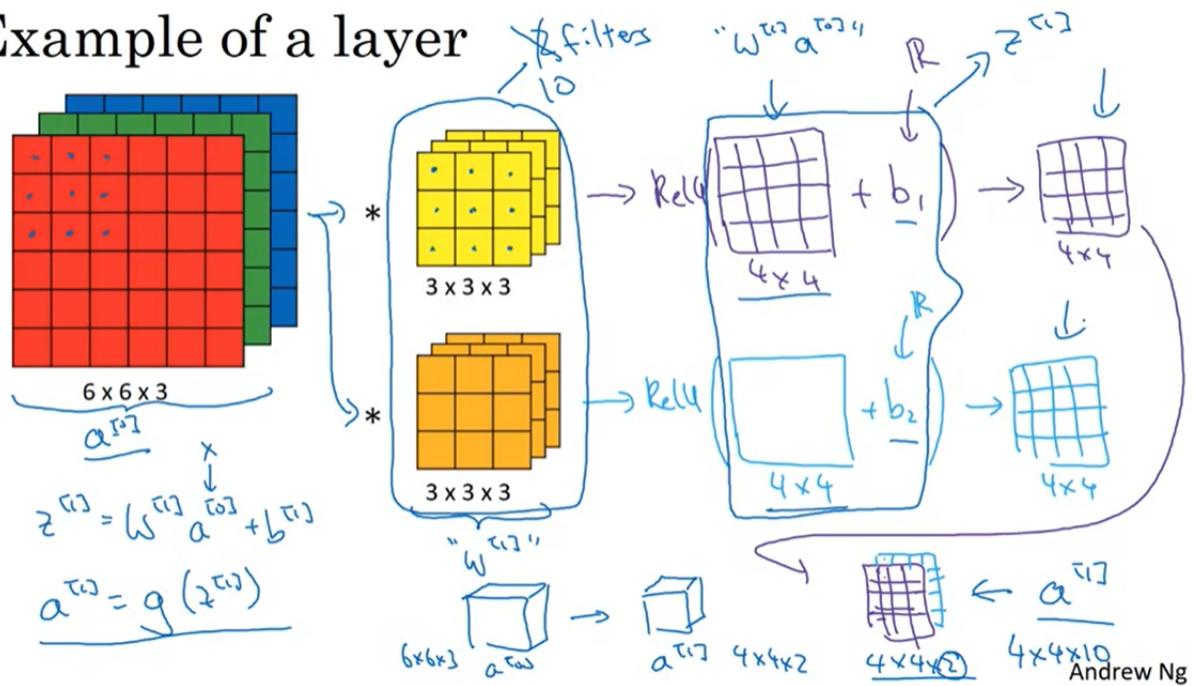
아마도 이것은 수직 모서리 감지기 일 수도 있고, 아니면 다른 feature를 감지 할 수도 있습니다. 이제, 두 번째 필터가 이 오렌지 빛 색상으로 표시 될 수 있습니다. 이건 수평 edge detector라고 생각해보죠. 따라서 첫 번째 필터로 컨볼루션하면 이 첫 번째 4×4 아웃풋이 나오고, 두 번째 필터와 컨볼루션하면 4×4 아웃풋이 달라집니다. 그리고 나서 우리가 할 수 있는 것은 이 두 개의 4×4 아웃풋을 산출해서, 앞에다가 이 첫 번째 것을 놓고 이 두 번째 필터를 취해서, 여기 그려보겠습니다. 이렇게 뒤에다 두십시오. 이 두 개를 쌓아두는 겁니다. 그럼 $4 \times 4 \times 2$ 의 아웃풋 볼륨이 만들어지게 됩니다. 그렇죠? 그리고 우리는 이걸 하나의 상자인 것처럼 볼륨을 생각할 수 있습니다.

dimension을 요약해보도록 하죠. $n \times n \times n_c$ 즉 채널 개수, 이렇게 생긴 인풋 이미지를 가지고 있다고 가정해 봅시다. 예를 들어, $6 \times 6 \times 3$ 여기서 n 옆에 붙은 대문자 C는 채널의 수를 가리키는 겁니다. 이제 $f \times f \times n_c$ 이렇게 컨볼루션하십시오 이건 $3 \times 3 \times 3$ 이렇게 됐었죠. 관례대로, 이 마지막 숫자는 같은 숫자여만 하죠. 이렇게 해서 도출되는 결과는 $\frac{n-f+1}{4} \times \frac{n-f+1}{4} \times n'_c$ 이렇게 나옵니다. 아니면 이 n'_c 는 다음 레이어가 될 수도 있고, 여러분이 사용하는 필터의 개수가 될 수도 있겠죠. 이 예시에서 이는 곧 $4 \times 4 \times 2$ 가 됩니다. 스트라이드 1을 쓰고, 패딩을 하지 않는다는 가정에서 이렇게 썼는데요, $n-f+1$ 이 아닌 다른 패딩 스트라이드를 사용한다면, $n-f+1$ 은 일반적인 방법으로 영향을 받을 겁니다.

이전 강의들에서 보셨던 대로 말이죠. 자, 볼륨에 대한 컨벌루션은 이렇게 정말 강력합니다. 그 중 극히 일부만이 3 채널로 RGB 이미지에서 직접 조작 할 수 있습니다. 그러나 더 중요한 것은 수직, 수평 모서리 같은 두 가지 feature들, 또는 10, 또는 어쩌면 128 또는 수백 가지의 서로 다른 feature를 감지 할 수 있다는 것입니다. 아웃풋은 탐지중인 필터의 수와 동일한 수의 채널을 가지게 될 것입니다. 표기법에 대한 설명을 덧붙이자면, 저는 문헌에 나오는 이 마지막 차원을 나타내기 위해 '채널의 수'라는 용어를 사용했습니다. 이는 종종 3D볼륨의 깊이라고 불리기도 합니다, channel 과 depth, 이 둘 모두 문헌들에서는 흔하게 사용되는 표현입니다. 하지만 depth라는 표현은 더 혼란을 주기도 하는데요, 신경망의 depth라는 용어도 사용하기 때문입니다. 그래서 저는 이 강의에서 필터의 세 번째 dimension의 사이즈를 가리킬 때 channel이라는 용어를 사용하려고 합니다. 볼륨을 컨볼루션에 어떻게 실행시킬 수 있을지 이제 배웠으니, 컨볼루션 신경망 레이어를 구현할 준비가 이제 되신 겁니다. 어떻게 하면 되는지 다음 강좌에서 같이 알아보도록 하겠습니다.

1 Layer of Convolutional Network

Example of a layer

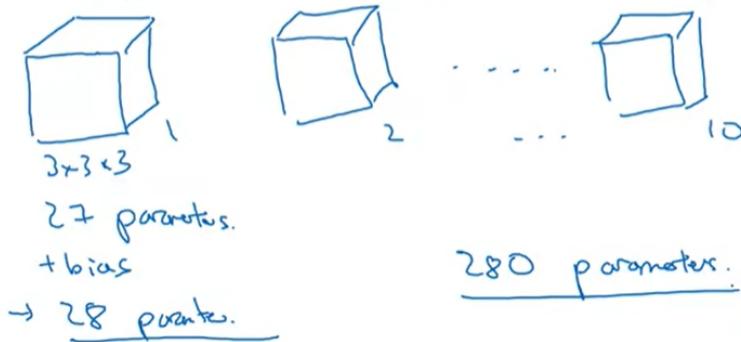


이 예제에서 우리는 두 개의 필터를 가지고 있습니다, 즉, 두 가지의 feature이죠. 이것으로 우린 $4 \times 4 \times 2$ 의 아웃풋을 생성할 수 있었습니다 하지만 예를 들어 2개가 아니라 필터가 10개가 있다면, $4 \times 4 \times 10$ dimension의 아웃풋 볼륨을 도출하게 되겠지요. 2개가 아니라 10개의

망을 취하게 될 것이고, $4 \times 4 \times 10$ 의 아웃풋 볼륨을 형성하기 위해 쌓아 올리게 될 것이기 때문입니다. 그게 바로 $a[1]$ 이 되는 것이죠.

Number of parameters in one layer

If you have 10 filters that are $3 \times 3 \times 3$ in one layer of a neural network, how many parameters does that layer have?



Andrew Ng

Q. 여러분에게 필터가 10개 있다고 가정해봅시다. 2개가 아니라 $3 \times 3 \times 3$ 의 볼륨이고, 신경망의 하나의 레이어라고 생각해보세요. 이 레이어는 얼마나 많은 파라미터를 가지고 있을까요?

A. 각각의 필터가 $3 \times 3 \times 3$ 의 볼륨이므로, 각각은 27개의 파라미터로 채워지겠죠, 그렇죠? 들어갈 숫자가 27개가 될 것이고, 거기에 바이어스가 있습니다. b 는 파라미터이므로, 그럼 28개의 파라미터가 되는 것이죠. $\rightarrow 28 \times 10 = 280$ 개의 파라미터

Summary of notation

If layer l is a convolution layer:

$f^{[l]}$ = filter size

$p^{[l]}$ = padding

$s^{[l]}$ = stride

$n_c^{[l]}$ = number of filters

→ Each filter is: $f^{[l]} \times f^{[l]} \times n_c^{[l]}$

Activations: $A^{[l-1]} \rightarrow n_H^{[l-1]} \times n_W^{[l-1]} \times n_C^{[l-1]}$

Weights: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$

bias: $n_c^{[l]} - (1, 1, 1, \dots, n_c^{[l]})$ # f : filters in layer l .

Input: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_C^{[l-1]}$

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

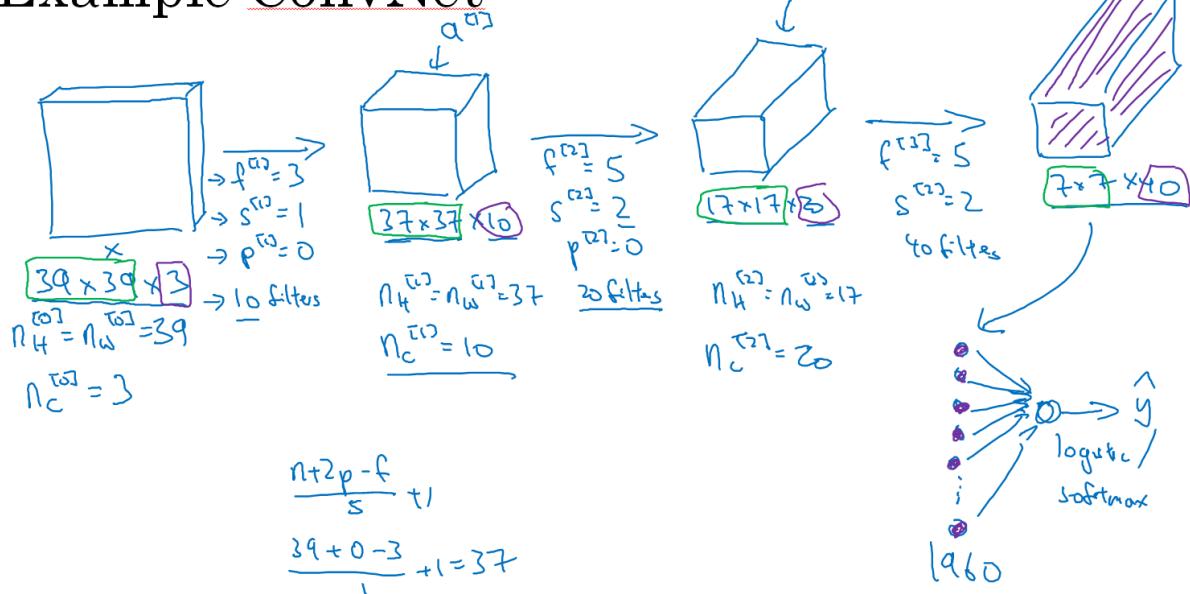
$$n_H^{[l]} = \left\lfloor \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

$$A^{[l]} \rightarrow m \times n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$$

Andrew Ng

간단한 예시

Example ConvNet

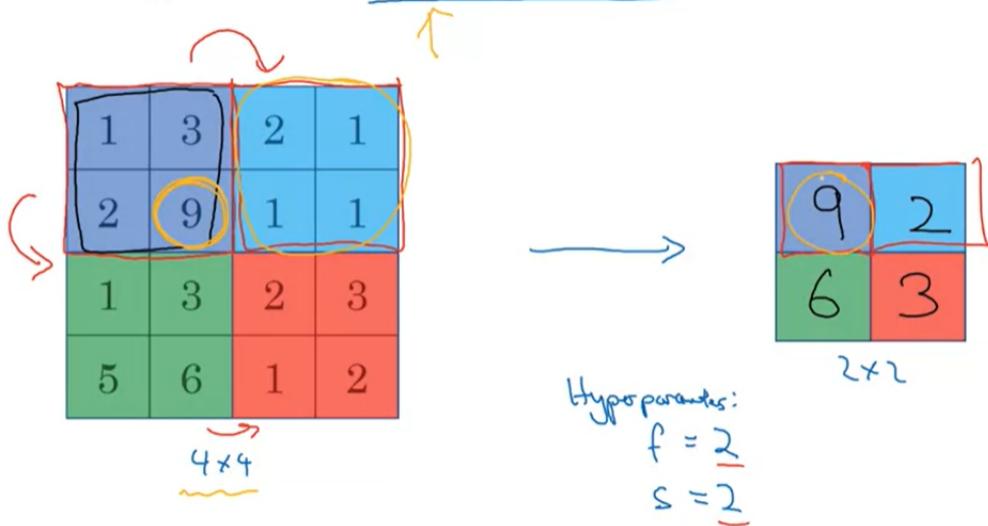


Andrew Ng

Pooling Layers

- 표현되는 크기를 줄이고, 계산 속도를 높이고 조금 더 견고하게 감지하는 기능을 만들기 위해 pooling 레이어를 사용

Pooling layer: Max pooling



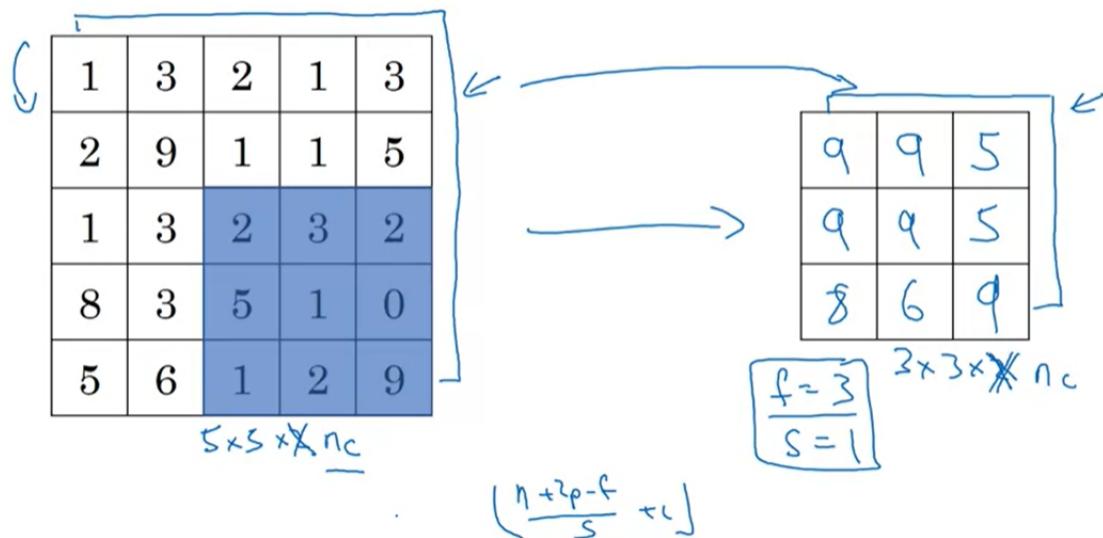
Andrew Ng

정사각형이 2×2 이므로, $f=2$ 스트라이드 = 2 이므로 $s=2$ 입니다.

max가하는 일은 말하자면, 이 feature들이 이 필터 어디에서든 감지되면, 최대 값을 잡아두는 것입니다. 하지만 이 feature가 감지되지 않으면, 아마도 이 feature는 오른쪽 상단 사분면에 존재하지 않을 겁니다.

1. Max Pooling

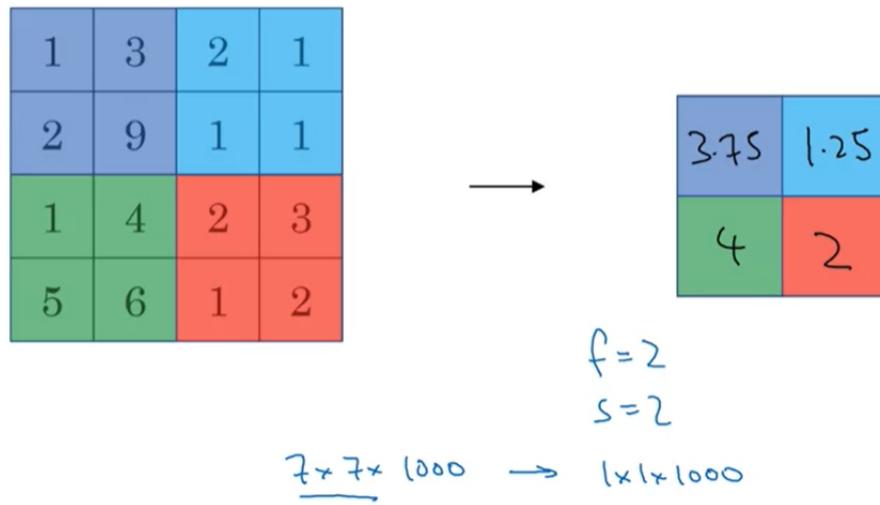
Pooling layer: Max pooling



Andrew Ng

2. Average Pooling(자주사용x)

Pooling layer: Average pooling



Andrew Ng

Summary of pooling

Hyperparameters:

f : filter size $f=2, s=2$
 s : stride $f=3, s=2$
Max or average pooling
 $\rightarrow p$: padding.

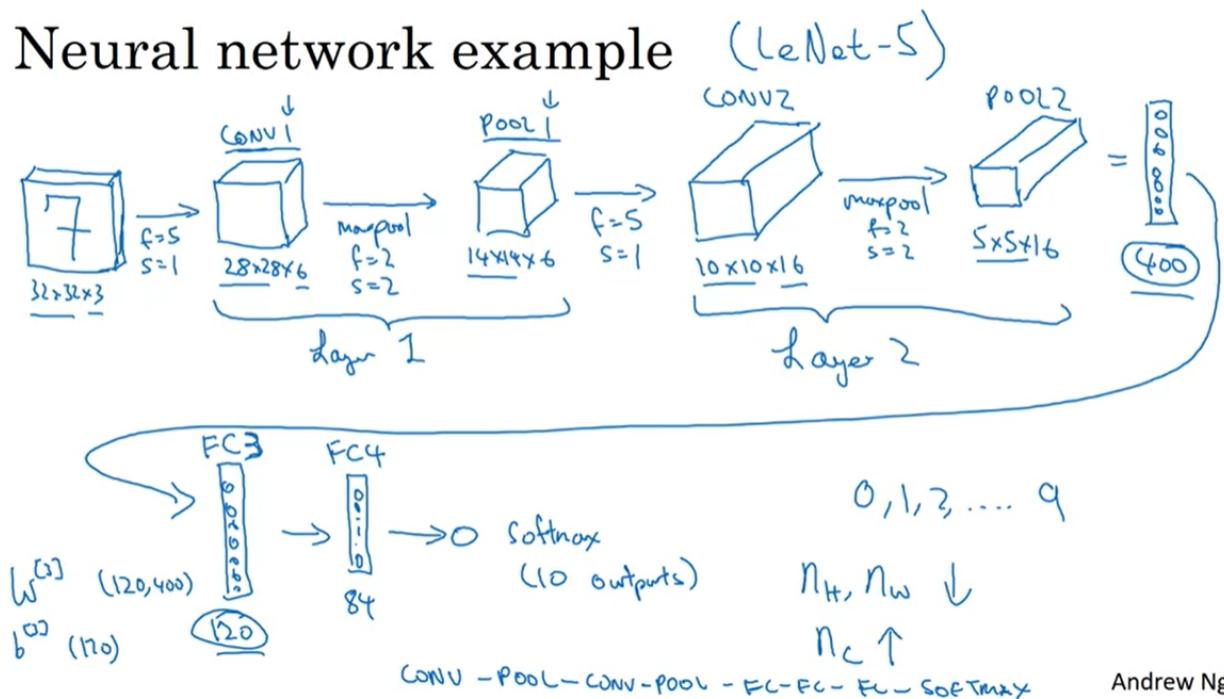
No parameters to learn!

$$\begin{aligned}
 & n_H \times n_w \times n_c \\
 & \downarrow \\
 & \left\lfloor \frac{n_H-f+1}{s} \right\rfloor \times \left\lfloor \frac{n_w-f}{s} + 1 \right\rfloor \\
 & \quad \times n_c
 \end{aligned}$$

Andrew Ng

CNN Example

Neural network example



Neural network example

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	— 3,072 $a^{(1)}$	0
CONV1 (f=5, s=1)	(28,28,8)	6,272	208 ←
POOL1	(14,14,8)	1,568	0 ←
CONV2 (f=5, s=1)	(10,10,16)	1,600	416 ←
POOL2	(5,5,16)	400	0 ←
FC3	(120,1)	120	48,001 ↗
FC4	(84,1)	84	10,081 ↘
Softmax	(10,1)	10	841

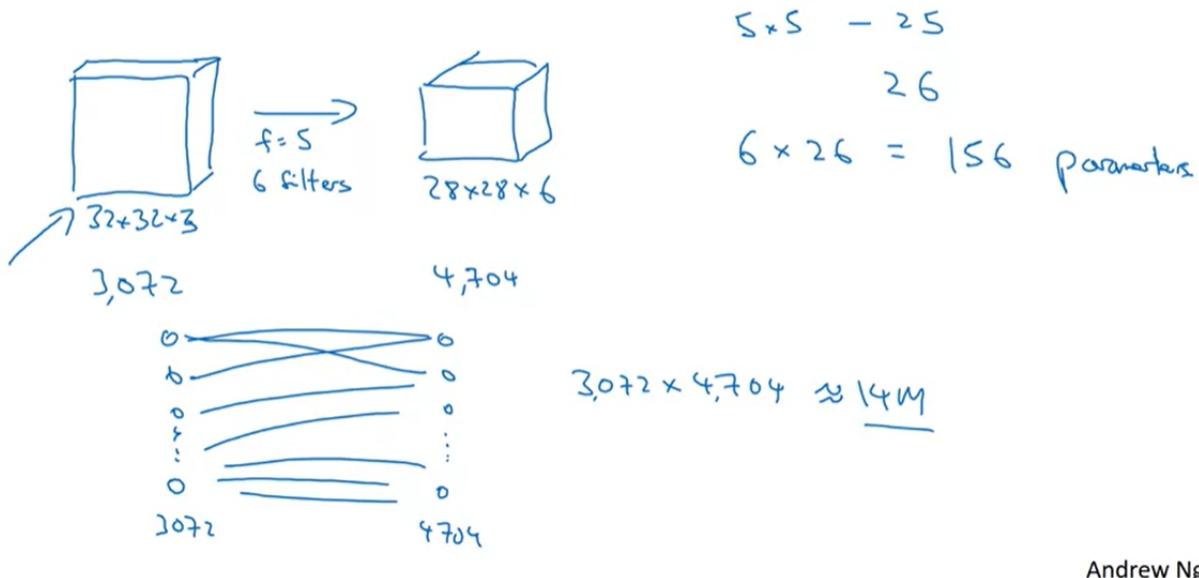
Andrew Ng

- 오타(208→608, 416→3216, 48001→48120, 10081→10164, 841→850)

몇 가지 확인해보자면, 우선, max pooling 레이어는 그 어떤 파라미터도 가지지 않는다는 걸 기억하세요. 둘째로, 이전 강의에서 말씀그린대로, conv 레이어는 상대적으로 파라미터가 거의 없는 경향이 있다는 것을 기억해주십시오. 사실 많은 파라미터가 신경망의 완전히 집약된 레이어안에 있는 경향이 있습니다. 또한 activation 사이즈는 신경망 안에 깊게 진행될 수록 점차 줄어들 것입니다. 너무 빨리 떨어지면, 수행 능력이 그다지 좋지 않습니다.

왜 컨볼루션이 유용한가?

Why convolutions

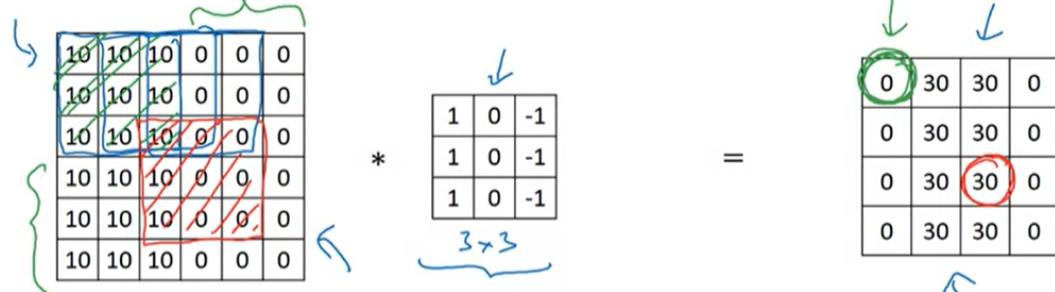


Andrew Ng

- 오차($(5*5+1)*6=156 \rightarrow (553+1)*6 = 456$)

총 파라미터의 수는 456 파라미터입니다. 따라서 이 conv 레이어의 파라미터 개수는 매우 작습니다.

Why convolutions



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.

Andrew Ng

작은 파라미터를 실행하는 이유는 실제로 두 가지 이유

1. 하나는 파라미터 공유

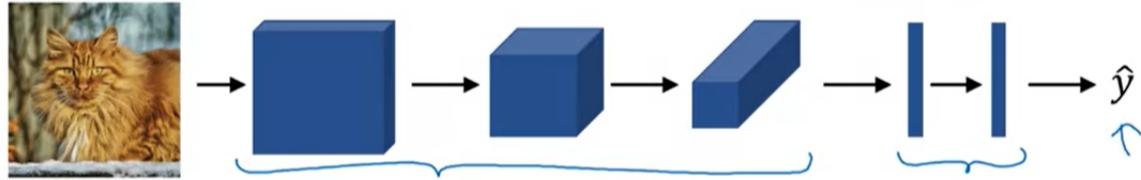
→ 수직 에지 감지기 같은 feature 감지기가 이미지의 한 부분에서 유용하다면 이미지의 다른 부분에서도 유용 할 것이라는 견해에 따라 사용됨

2. Sparsity of connections

→ 9 개의 인풋 feature만이 아웃풋에 연결되어 있고, 다른 픽셀은 이 아웃풋에 전혀 영향을 미치지 않는 것처럼 보입니다

Putting it together

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



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

Use gradient descent to optimize parameters to reduce J

Andrew Ng