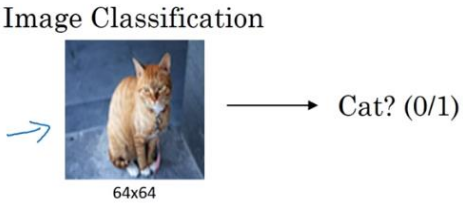
딥러닝 4단계 : 합성곱 신경망 네트워크 (CNN) 1. 합성곱 신경망

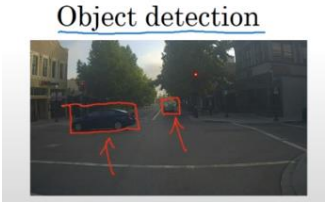
<컴퓨터 비전>

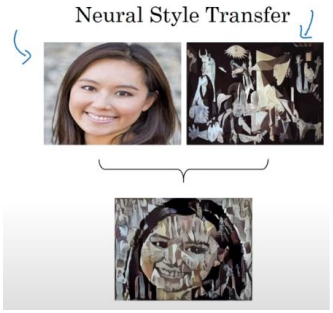
<딥러닝에 관심이 가는 이유>

1. 컴퓨터 비전의 빠른 발전이 새로운 애플리케이션을 가능하게 함

2. 컴퓨터 비전을 구축하지 않더라도 컴퓨터 비전을 연구하는 사회가 알고리즘에 많은 영감을 주면서 영향을 끼침 (ex : 음성인식)







- 이미지 분류
- 물체 감지 (자율주행 차 설계할 때 어느 위치에 차가 있는지 알아내야 함)
- 신경망 스타일 변형 (다른 스타일로 그리고 싶을 때)

  컨텐츠 이미지, 스타일 이미지를 신경망을 이용하여 재구성하여 새로운 형태 제공

<장애물>

입력이 매우 클 수 있다는 것

변수가 많으면 충분한 데이터를 얻어 과적합을 방지하기 어려움

⇨ 합성곱 연산 필요

<모서리 감지 예시(edge detection)>

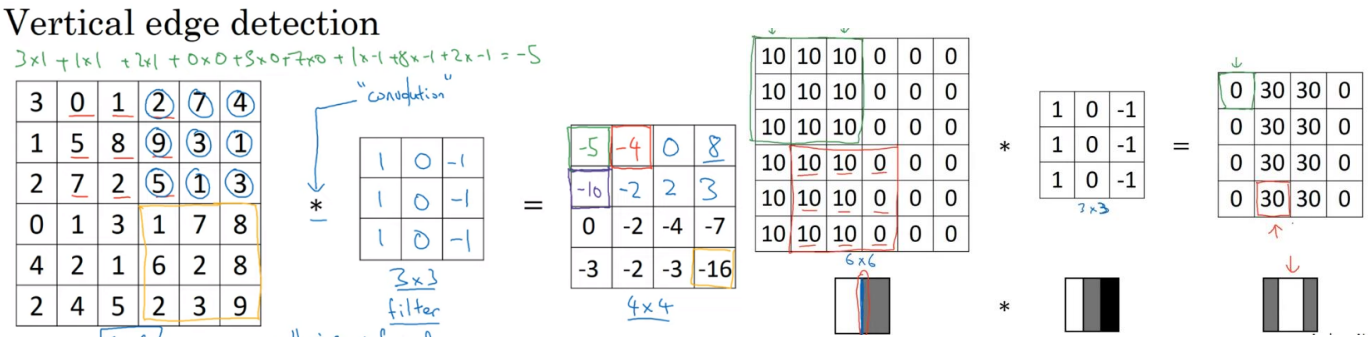
신경망의 하위층이 모서리를 감지 => 이후 층들이 물체를 감지

<이미지의 모서리를 감지하는 방법>

1. 수직인 모서리를 감지

2. 수평의 모서리를 감지

HOW?



EX) 6x6 그레이 스케일 이미지 6x6x1

3x3 행렬(필터/커널)을 만들어서 합성곱(*) 진행

단점 : 수학에서는 별표가 합성곱을 나타내지만 파이썬에서는 이 표현이 곱셈을 나타냄

=> 결과는 4x4 행렬 (이미지)

1. 필터에 있는 수와 원래의 이미지와 요소별 곱셈을 진행하여 더한다

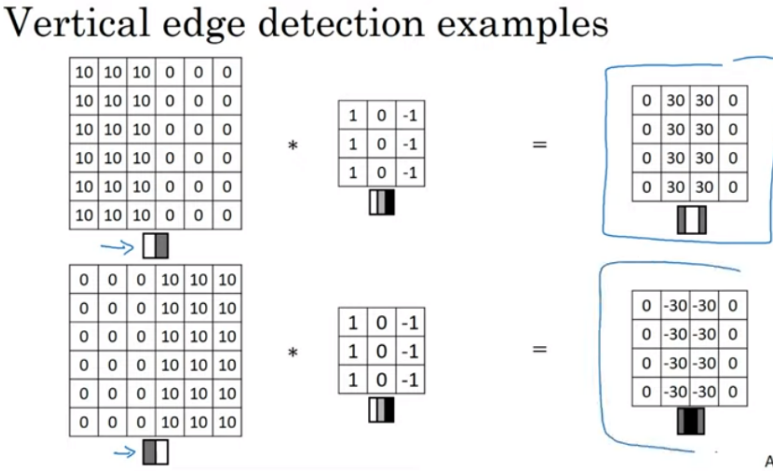
2. 필터를 오른쪽으로 한 칸 옮긴 후 같은 요소별 곱셈을 진행하여 더한다

3. 필터를 한 칸 아래로 내린 후 같은 작업을 진행한다

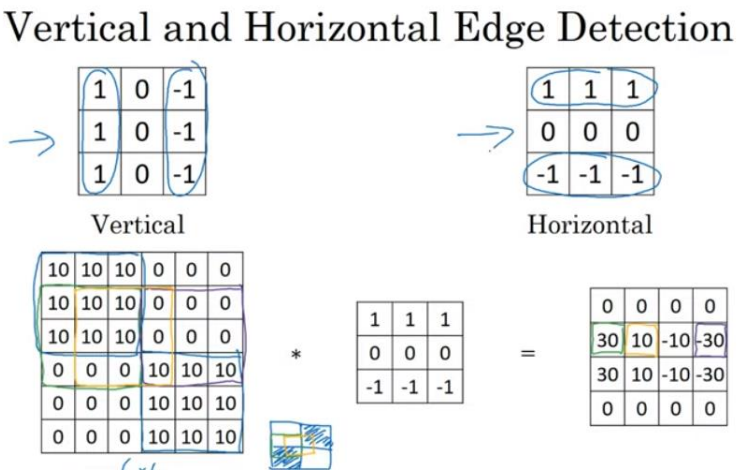
=> 다른 이미지가 나온다

ConvForward함수/ 텐서플로우에서 tf.nn.conv2d / keras에서 Conv2d

<더 많은 모서리 감지 예시>



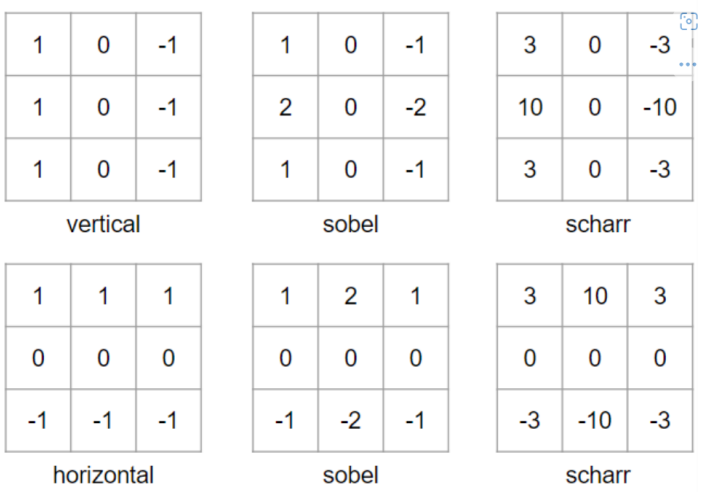
★ 양과 음의 윤곽선 차이 (밝기의 전환)



- Vertical => 왼쪽이 밝고 오른쪽이 상대적으로 어두운
- Horizontal => 위쪽이 상대적으로 밝고 아래쪽이 상대적으로 어두운

★ 서로 다른 필터는 세로 가로 윤곽선을 검출

어떤 숫자의 조합을 사용해야하는지 논쟁이 있었음



소벨필터 : 중간부분의 픽셀에 중점을 줘서 선명해 보이게 함

Scharr 필터 : 세로 윤곽선 검출을 위한 것 (90도 회전하면 가로 윤곽선 검출)

★ 복잡한 이미지에서는 스스로 학습하게 두고 9개의 숫자를 변수로 설정한 후, 역전달 방법으로 합성곱을 했을 때 좋은 윤곽선 검출기를 찾을 수 있음.

<패딩(Padding)>

결과 이미지 : n-f+1*n-f+1

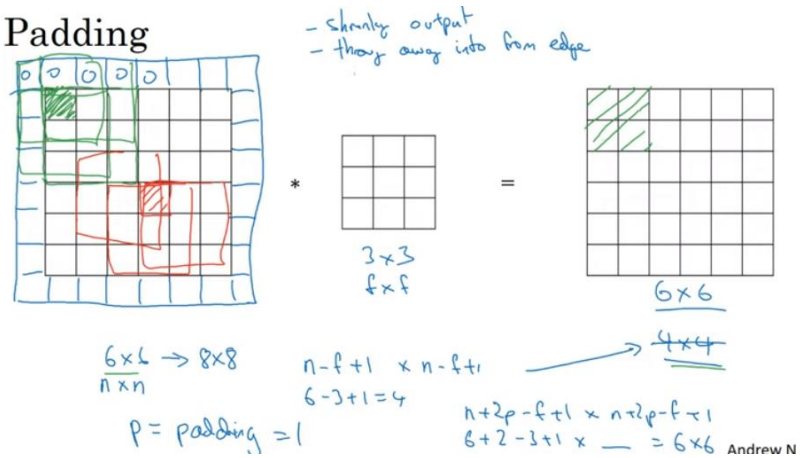
단점

1. 이미지가 축소

2. 가장자리 픽셀은 이미지에서 한번만 사용되고, 중간의 픽셀은 여러 영역에 걸쳐있기 때문에 많이 쓰임

가장자리 근처의 정보를 날리게 되는 것

⇨ **합성곱연산을 하기전에 이미지를 덧댄다 => 패딩**



- 1픽셀만큼 가장자리에 0을 덧대줌

⇨ 기존의 크기를 유지할 수 있게 된다

★ p가 패딩의 양이면 p는 1 (1픽셀만큼 가장자리에 더해줌)

결과 : n+2p-f+1*n+2p-f+1

- 가장자리의 정보를 덜 가져오는 일을 줄일 수 있음

<얼만큼 패딩 할 것인지>

·○ **유효 합성곱** : 패딩이 없음 => n-f+1*n-f+1 차원의 결과

·○ **동일 합성곱** : 결과 이미지의 크기가 기존 이미지와 동일하도록

n+2p-f+1=n => p=(f-1)/2

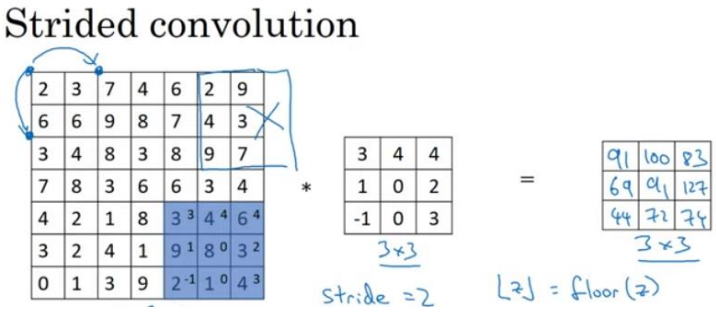
f가 홀수이면 이미지의 크기가 기존의 이미지와 동일해짐

컴퓨터 비전에서 f는 거의 항상 홀수 WHY?

1. f가 짝수이면 패딩이 비대칭이 됨

2. 홀수크기의 필터가 있으면 중심 위치(중심 픽셀)가 존재

<스트라이드 (stride)>



★ stride=2로 합성곱을 진행하면 두칸을 옮겨서 요소별 곱셈 합을 진행하는 것

(n+2p-f)/s +1 * (n+2p-f)/s +1

- 분수 값이 정수가 아니라면 **내림**을 진행

교차상관 vs 합성곱

합성곱 : 요소별 곱셈 합 전에 필터를 가로축과 세로축으로 뒤집고, 뒤집은 행렬로 결과를 계산

미러링 과정을 생략한 것!

지금까지의 연산 과정은 교차 상관으로 불림

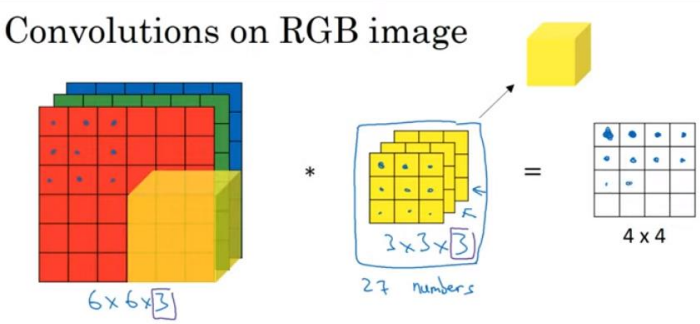
BUT 딥러닝에서는 관습적으로 합성곱이라고 함

<입체 이미지에서의 합성 곱>

6x6x3 의 이미지 stack에는 3x3x3 필터(3개의 층을 가짐) => 결과 이미지는 4x4x1

높이/넓이/채널의 수

★ 채널의 수는 필터의 채널의 수와 같아야함



- 3x3x3 필터는 27개의 파라미터를 가짐=> 이 숫자를 빨간채널, 초록채널, 파랑채널과 곱셈 후 덧셈
- 초록채널부분=0, 파란채널부분=0 합쳐져서 필터를 만들면 빨간 채널의 세로 윤곽선을 검출하는 필터

여러 개의 필터를 동시에 사용하려면?

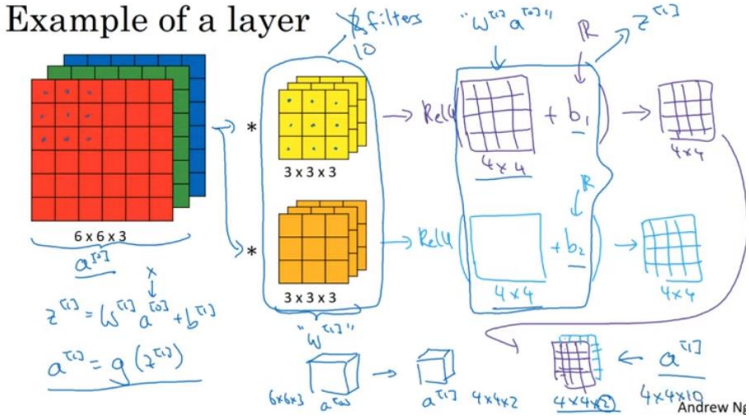
- 4x4x2의 부피 (2개의 필터 의미)
- nxnxnc (채널수) * fxfxnc (채널 수) => n-f+1 * n-f+1 * nc′ (필터의 수) (스트라이드=1, 패딩=0일 때)

필터의 마지막 크기 = 채널 = 입체형의 깊이

$(n \times n \times n_c) * (f \times f \times n_c) = (n-f+1) \times (n-f+1) \times n_{c'}$ 형태

- n : 이미지의 크기
- n_c : 채널의 개수
- f : 필터의 크기
- $n_{c'}$: 사용된 필터의 개수

<합성곱 네트워크의 한 계층 구성하기>



결과 행렬 각각에 편향을 더해주고 비선형성 적용하고 쌓기

합성곱 신경망의 한 계층

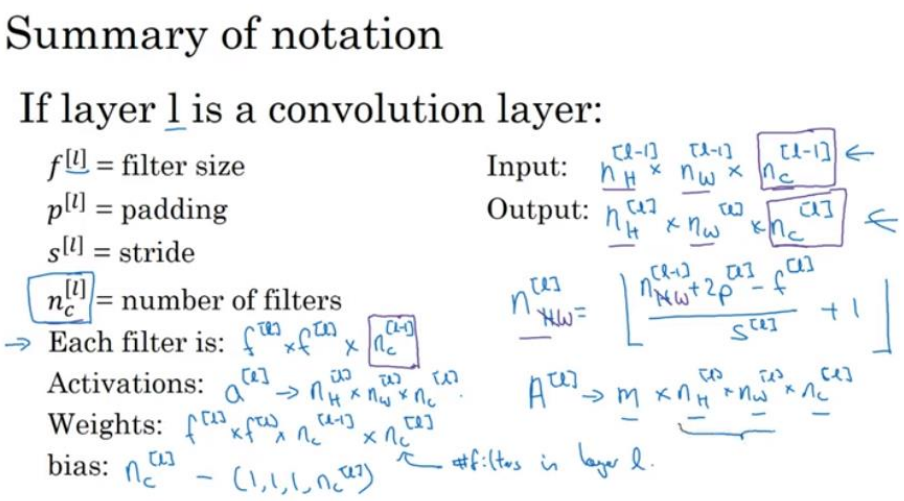
합성곱에서는 편향을 더해주고 relu 연산을 하는 것

6x6x3 => 4x4x2 (두개의 필터라서)

3x3x3=27+1(bias) =28 파라미터 * 10 = 280 파라미터

이미지가 매우 커도 변수는 이 숫자로 고정

⇨ 과대적합을 방지하는 합성곱 신경망의 한 성질



- l: 합성곱 계층
- f[l] : 필터의 크기가 fxf
- p[l] : 패딩의 양 (유효 합성곱 or 동일 합성곱)
- s[l] : stride
- 입력 : nH[l-1] x nW[-1] x nc[l-1](채널의 수)
- 출력: nH[l] x nW[l] x nc[l]

nH[l]=(nH[l-1]+2p[l]-f[l]) / s[l] +1

nW[l]=(nWl-1]+2p[l]-f[l]) / s[l] +1

채널의 수 = 필터의 수

필터의 크기 = f[l] x f[l] x nc[l-1]

편향과 비선형성 추가 = a[l] =nH[l] x nW[l] x nc[l]

A[l] = m x nH[l] x nW[l] x nc[l]

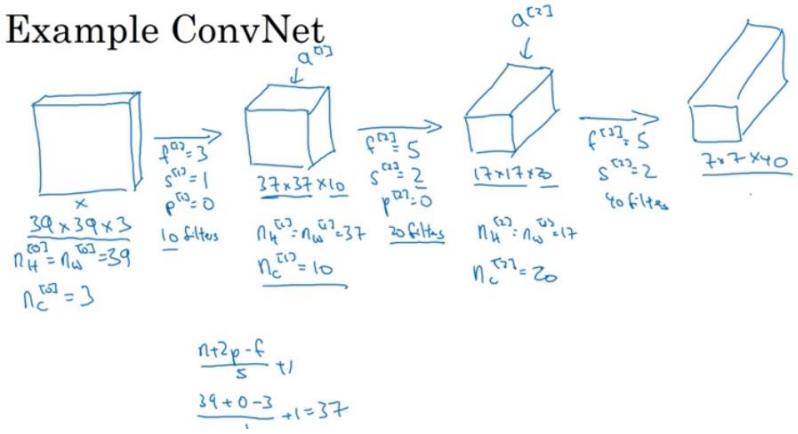
가중치 W = f[l] x f[l] x nc[l-1] x nc[l] (필터의 개수)

편향 : nc[l]= (1,1,1,nc[l]) 4차원 행렬

<간단한 합성곱 네트워크 예시>

x라는 이미지에서 고양이인지 아닌지 0 1 로 분류하는 예시

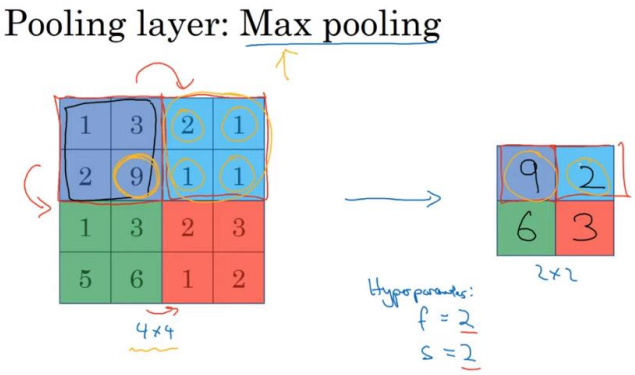
39x39x3의 이미지일 때



펼쳐서 하나의 벡터로 만들어서 softmax에 대입 => 최종 예측값

- 합성곱 층 CONV
- 풀링 층 POOL
- 완전 연결 층 FC

<풀링 층>



4x4 => 2x2 & stride=2 (최대 풀링 시) f=2, s=2

★ 가장 큰 수가 특정 특성을 의미

=> 최대연산특성을 발견하면 최대 풀링의 결과로

필터의 한 부분에서 특성이 검출되면 높은 수를 남기고 검출되지 않으면 그 안의 최대값은 작은 수로 남음

★ 여러 하이퍼파라미터가 있지만 학습할 수 있는 변수가 없음

경사하강으로 학습 불가

5x5 f=3 s=1 => 결과 3x3

n+2p-f/s + 1 로 최대 풀링 출력 결과

5x5xnc => 결과 3x3xnc

- 각 채널에 개별적으로 풀링 적용

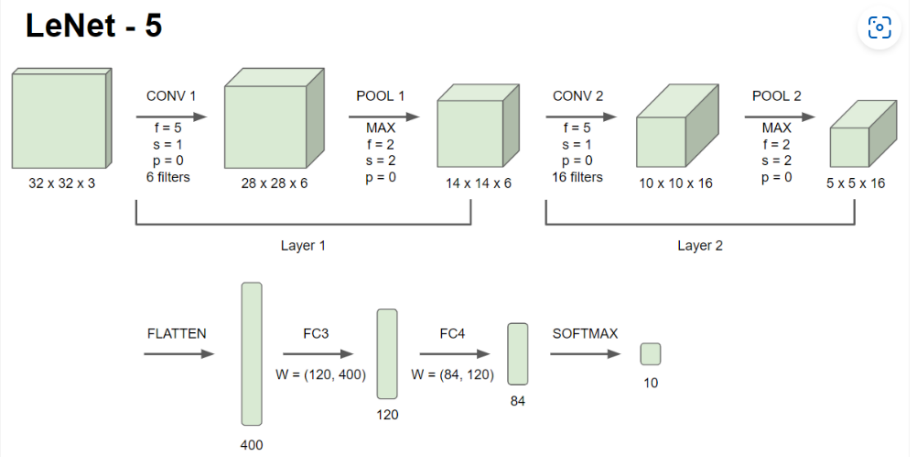
평균 풀링 : 각 필터의 평균을 취함

(아주 깊은 신경망 속에서 쓰일 수도 있음)

- f= 필터 크기
- s = 스트라이드
- *f=2,s=2 높이와 너비를 절반으로 줄여주는 효과*
- 최대풀링 / 평균 풀링
- p = padding (잘 사용 X p=0)

- nH x nW x nc => [(nH-f+1) / s + 1] * [(nH-f+1) / s + 1] * nc

<CNN 예시>



EX) 32x32x3 RGB (7이라고 쓰여있는 이미지)

[layer1]

- 첫번째 층 : 5x5 s=1,p=0 => 출력 CONV1: 28x28x6
- 최대 풀링 적용 : f=2,s=2 => POOL1: 14x14x6

풀링층은 변수가 없고 하이퍼파라미터만 있어서

[layer2]

- f=5,s=1,p=0 => CONV2 : 10x10x16
- 최대 풀링 적용 : f=2,s=2 => POOL2: 5x5x16

flatten => 400x1 벡터

120개의 유닛을 가진 층으로 만들기 : FC3 (완전 연결 층)

더 작은 층 84 를 더하기 : FC4

softmax 적용

•◌ 깊어질수록 높이와 너비는 줄어들고 채널의 수는 늘어남

합성곱=>풀링층=>합성곱층=>풀링층=>완전연결층=>softmax

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	3,072 $a^{[0]}$	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

1. 최대 풀링 층은 변수가 따로 없음
2. 합성곱 층이 상대적으로 적은 파라미터를 가짐 (완전 연결층에 많음)
3. 활성값 크기도 신경망이 깊어질수록 점점 감소 (너무 빠르게 감소하면 성능 bad)

<왜 합성곱을 사용할까요?>

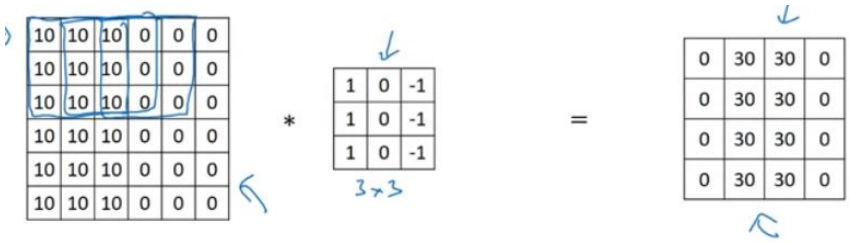
1. 변수공유
2. 희소 연결

EX) 32x32x2 이미지

f=5,6filters => 28x8x6 = 4704

모든 층을 연결하면 가중치 형렬의 변수 개수는 3072*4074 => 변수가 너무 많음

필터당 26개의 변수, 6*26=156 (적은 파라미터)



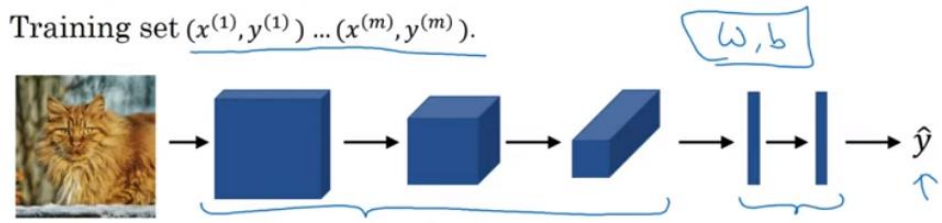
Parameter sharing: A feature detector (such as a vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.

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

1. **변수 공유** : 한 부분에서 이미지의 특성을 검출하는 필터가 이미지의 다른 부분에서도 똑같이 적용

2. **희소 연결** : 출력값이 이미지의 일부(작은 입력값)에 영향을 받고, 나머지 픽셀들의 영향을 받지 않기 때문에, 과대적합을 방지

x : 이미지 y: 이진분류 레이블



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