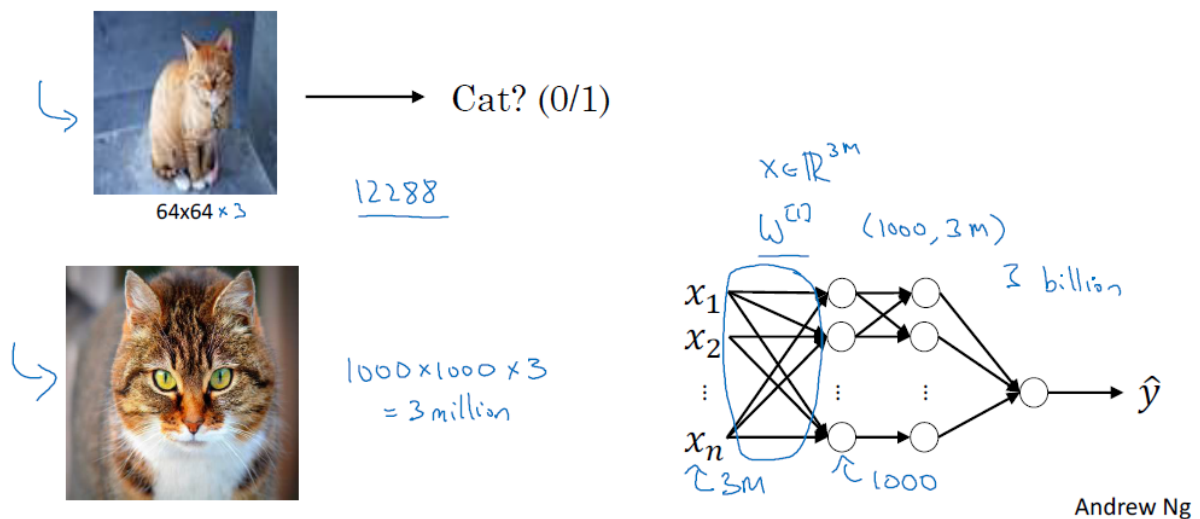


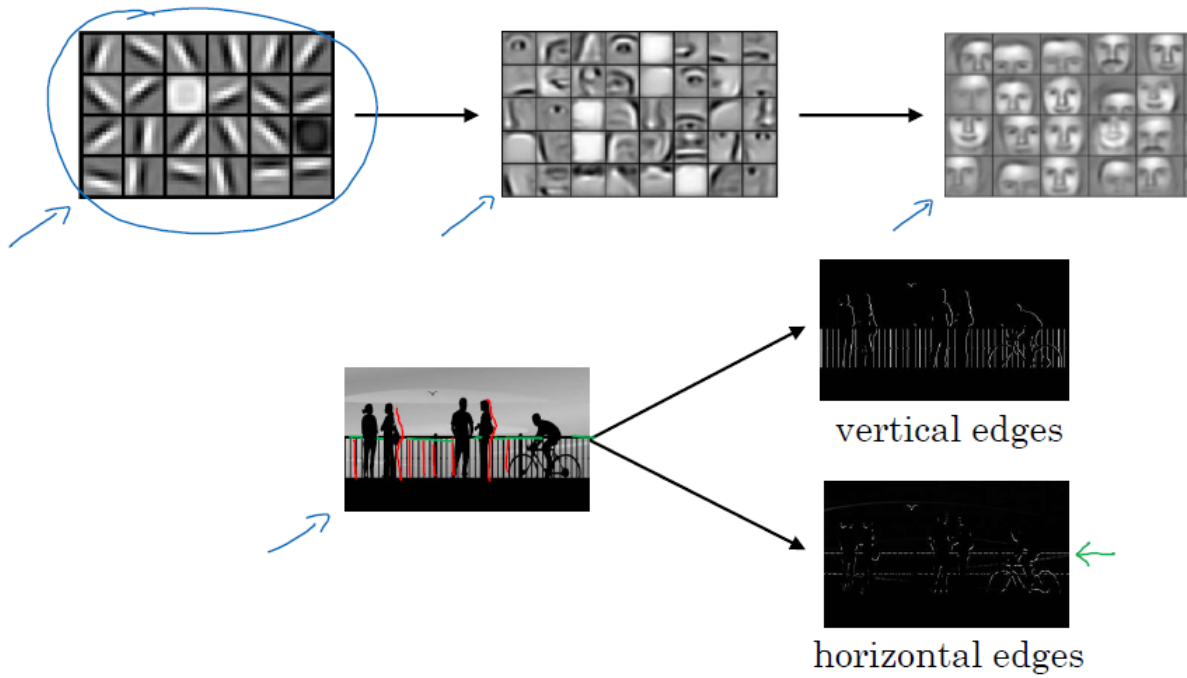
# 1. 합성곱 신경망

## 1. 컴퓨터 비전

- 이미지 인식에서 입력값: input feature  $n * m * k$
  - 300만 입력값( $x_n$ 의 개수)에 따라 첫 번째 은닉층에는 1,000개의 unit 존재
- 총 무게:  $w_1$  metrics
- $1,000 * 1,000 * 3$ (3M)으로 파라미터를 가지게 됨.
- 대용량 이미지인 고차원 훈련에 대해서 새롭게 훈련 모델을 설정할 필요가 있음.



## 2. 모서리 감지 예시



- 입력 이미지에서 수평선, 수직선을 감지하기 위해 필터 활용.
- \*를 CNN 연산을 위한 기호를 활용

- 이미지는 (높이 x 넓이) 로 표현할 수 있습니다.
- 합성곱 연산은 아래 그림과 같이 진행됩니다. 왼쪽 이미지는 원래 이미지, 중앙에 있는 3 x 3 행렬은 필터(커널)입니다. 각각의 원소곱 후 전부 더해줍니다.

6 x 6					
3 <sup>1</sup>	0 <sup>0</sup>	1 <sup>-1</sup>	2	7	4
1 <sup>1</sup>	5 <sup>0</sup>	8 <sup>-1</sup>	9	3	1
2 <sup>1</sup>	7 <sup>0</sup>	2 <sup>-1</sup>	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
4	2	5	2	3	9

\*

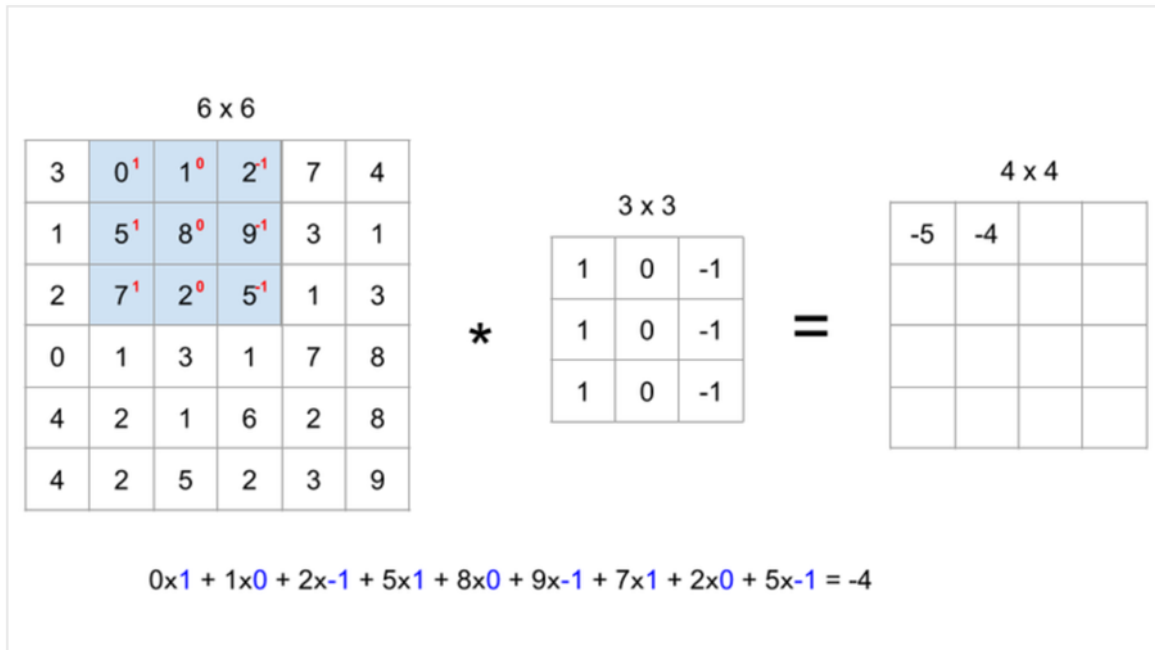
3 x 3		
1	0	-1
1	0	-1
1	0	-1

=

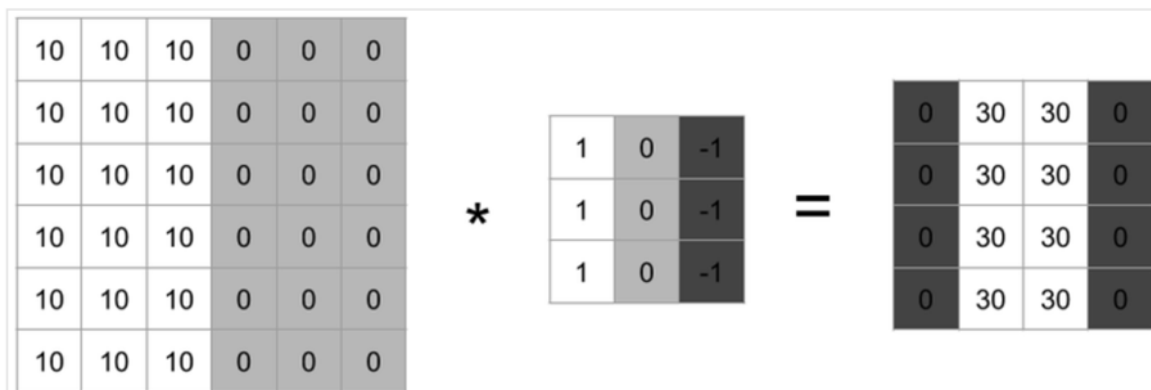
4 x 4			
-5			

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

- 그후 다음 스텝으로 필터(커널)을 한칸 이동하여 합성곱 연산을 진행합니다. 이렇게 이미지의 밑부분까지 진행하여 최종 4 x 4 의 새로운 행렬을 만들어 냅니다.



- 그렇다면 수직 윤곽선은 어떻게 탐지 할 수 있었을까요?
- 아래 그림의 왼쪽이미지에서 10과 0 사이의 경계선이 수직 윤곽선입니다.
- 필터를 통과해 합성곱 연산을 하게 되면 밝은 부분이 중앙으로 나타납니다. 이는 원래 이미지의 경계선을 해당 하는 부분입니다. 비록 크기가 안맞고 검출된 경계선이 조금은 두껍지만 이는 원래 이미지가 작아서 그렇습니다.



수직 윤곽선 감지 필터

- Python: ConvForward 함수
- tensorflow: tf.nn.conv2d
- keras: Conv2D

### 3. 더 많은 모서리 감지 예시

1	0	-1
1	0	-1
1	0	-1
vertical		

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

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

1	1	1
0	0	0
-1	-1	-1
horizontal		


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

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

- 수직선, 수평선 이외에도 다양한 윤곽선을 감지 가능.


→ ex) 밝은 곳에서 어두운 곳으로, 어두운 곳에서 밝은 곳으로 인식

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	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



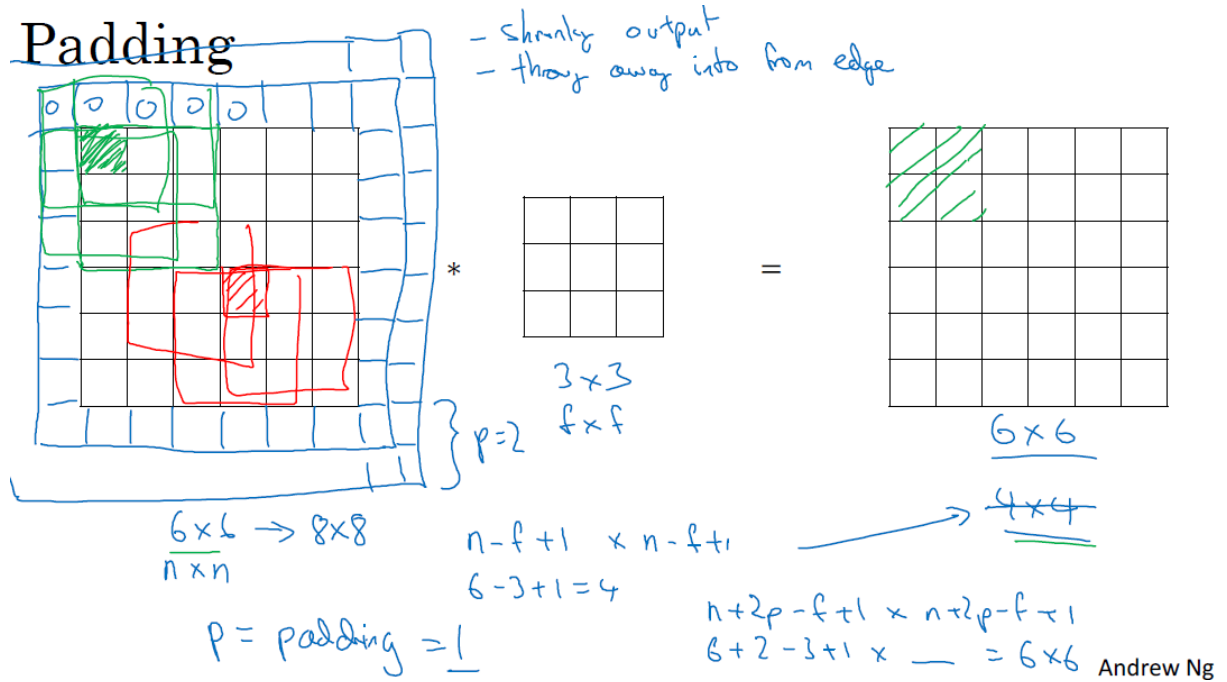
- sobel filter로 중앙 열, 중앙 픽셀에 무게를 두어 데이터에 좀 더 견고해짐.
- 윤곽선을 탐지 위한 필터: Sobel 필터, Scharr 필터 등
- 최근 딥러닝에서는 임의의 숫자로 만든 다음에 역전파를 통해 알아서 학습시켜서 문제에 적합한 필터를 만드는 방법을 사용

## 4. 패딩

- 컨볼루션 연산의 문제점
  - 계속 합성곱 연산을 하게 되면, 이미지가 계속 축소 됨.

- 이미지의 코너나 가장자리에 있는 정보들을 적게 활용하지만, 가운데에 있는 데이터는 많이 사용
- 깊은 신경망을 활용할 때 가장자리 정보는 덜 사용하며, 가운데 정보는 많이 사용하며 **데이터 훈련에 취약**해집니다.

- 위의 단점을 개선하기 위해 **이미지에 경계선을 덧대는(padding) 방법**을 적용



→ 원래 인풋 이미지를 유지 가능

- 패딩 값은 임의로 설정 가능
- n: 이미지 크기, p: 패딩 크기, f: 필터 크

## Valid and Same convolutions

→ no padding

“Valid”:  $n \times n$   $\times$   $f \times f$   $\rightarrow \frac{n-f+1}{1} \times \frac{n-f+1}{1}$   
 $6 \times 6$   $\times$   $3 \times 3$   $\rightarrow 4 \times 4$

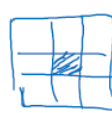
“Same”: Pad so that output size is the same as the input size.

$$n+2p-f+1 \times n+2p-f+1$$

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

$3 \times 3$   $p = \frac{3-1}{2} = 1$  |  $5 \times 5$   $f=5$   $p=2$

$f$  is usually odd



1x1  
3x3  
5x5  
7x7

Andrew Ng

- 컴퓨터 이미지 인식 분야의 관습적 접근에 따라 **필터(f)는 홀수를 사용**
  - f가 홀수: **인풋 사이즈와 아웃풋 사이즈가 동일하게 설정 가능.**
  - f가 짝수: **비대칭적 훈련**이 진행
- 필터 적용 전과 후의 이미지 사이즈가 똑같으려면,  **$n + 2p - f + 1 = n$** 을 만족하도록  
 → padding size인  $p = (f-1) / 2$ 로 설정

## 5. 스트라이드

: CNN의 블록 방식 중 하나

- stride 종과 횡으로 이동하는 거리를 의미
  - stride s가 2일 때 2칸씩 이동해서 3x3 metrics를 만든다.
  - stride: 필터의 이동 횟수

# Strided convolution

Diagram illustrating a strided convolution operation. The input is a 7x7 grid of numbers. A 3x3 kernel is applied to a 3x3 region of the input, shifted by a stride of 2. The result is a 3x3 grid of numbers.

Handwritten notes show the calculation of the output size using the formula:

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

For the example,  $n=7, p=0, f=3, s=2$ :

$$\left\lfloor \frac{7+0-3}{2} + 1 \right\rfloor = \left\lfloor \frac{4}{2} + 1 \right\rfloor = \left\lfloor 2 + 1 \right\rfloor = 3$$

- strided를 적용한 결괏값 레이어: " $(n+2p-f)/s + 1$ 의 내림"으로 계산

- 스트라이드는 필터의 이동 횟수를 뜻합니다. 즉, 기존에 필터가 한칸씩 이동해서 계산했다면, 스트라이드를 주게 되면 그 수만큼 필터가 이동해서 계산하게 됩니다.
- 따라서 최종 크기는  $\left(\frac{n+2p-f}{s} + 1\right) \times \left(\frac{n+2p-f}{s} + 1\right)$  가 됩니다. 만약에 소수점으로 만들었다면 내림을 하게 됩니다. 보통은 필터에 맞춰서 최대한 크기가 정수가 될수 있도록 패딩과 스트라이드 수치를 맞춥니다.
- 신호처리에서의 교차상관과 합성곱의 관계를 알아봅시다.
  - 일반적으로 수학에서 정의하는 합성곱은 합성곱을 하기 전에 필터를 가로축과 세로축으로 뒤집는 연산을 해줘야합니다.
  - 지금까지 배운 합성곱은 사실 교차상관이지만 딥러닝에서는 관습적으로 합성곱이라고 합니다.
  - 딥러닝에서는 뒤집는 연산을 생략합니다. 이 뒤집는 과정은 신호처리에서는 유용하지만 심층 신경망 분야에서는 아무런 영향이 없기 때문에 생략하게 됩니다.

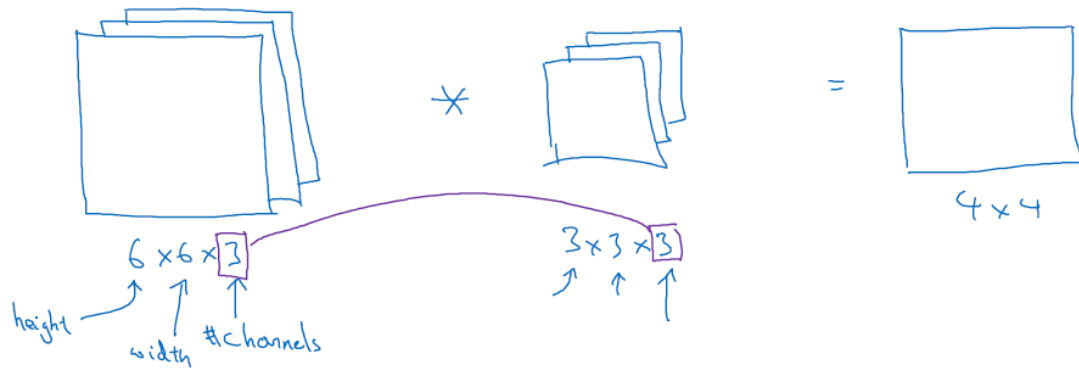
## 6. 입체형 이미지에서의 합성곱

- 3차원 이상의 volume에서 CNN을 적용하는 방법: **channel(RGB)이 적용되는 것을 제외하면 2차원 이미지와 동일합니다.**

ex) 6x6x3차원의 이미지를 3x3x3로 Convolution을 진행

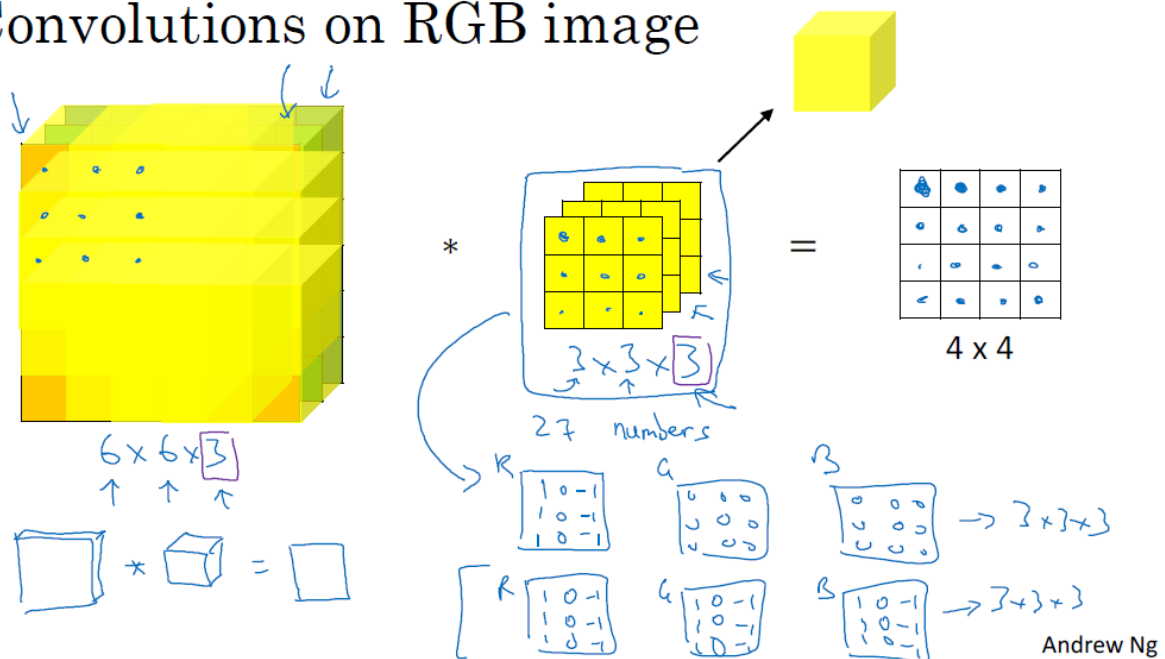
(height x width x channels, 이때 channels는 동일해야 함.)

# Convolutions on RGB images



- 결과물에는 channels가 적용되지 않고, 2차원으로 나타나는 것!
- 계산 방식

## Convolutions on RGB image

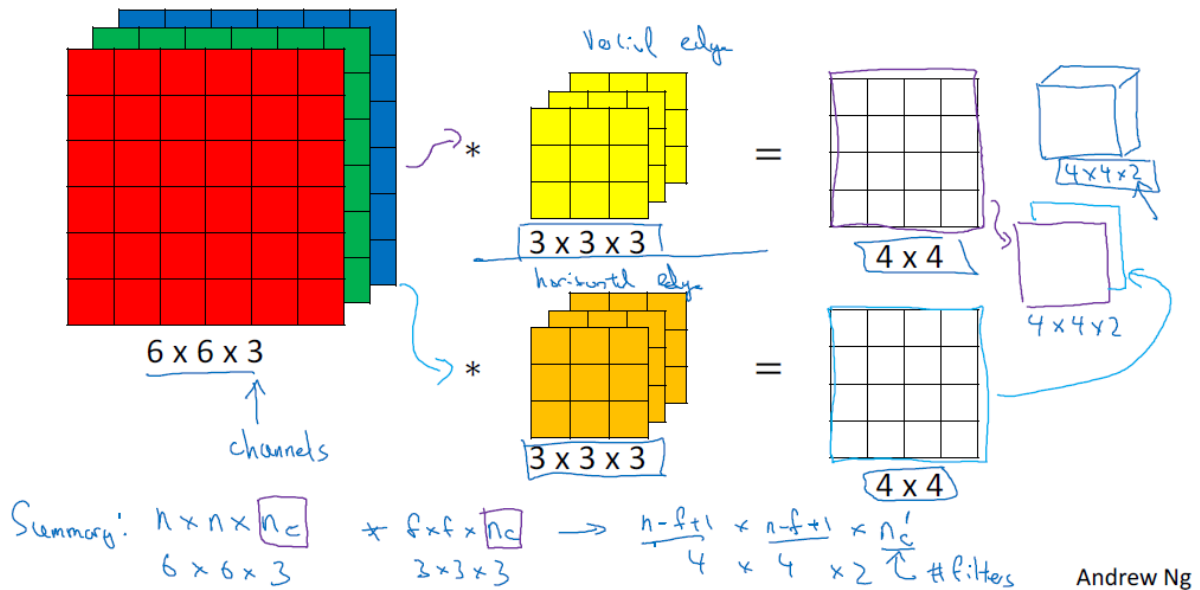


- 3x3x3을 RGB에 대응하여 곱하고, 이를 더한 값을 output metrics에 작성.
- 이를 한 칸씩 이동하면서 반복해서 진행
- 색상에 대해서 감지하는 필터를 만들 수 있다.
- 결과값은 동일한 채널인 input volume과 convolution volume을 2차원의 metrics로 반환



- Multiple filters

## Multiple filters



- 여러 필터를 동시에 진행하고 싶을 때, output volume을 각각의 층으로 합쳐서, 하나의 상자처럼 생각
- $n_c'$  필터 개수의 output volume이 됨.
- multiple filter의 장점: 수평, 수직 또는 수백 가지의 서로 다른 feature를 감지 가능

## 7. 합성곱 네트워크의 한 계층 구성하기

- 합성곱 신경망의 한 계층

: 합성곱 연산, 편향, 활성화 함수로 구성

- 활성화 함수: 비선형성 적용 위함(ReLU 많이 사용)
- 표기법

- $l$ :  $l$  번째 층
- $f^{[l]}$ : 필터의 크기
- $p^{[l]}$ : 패딩의 양
- $s^{[l]}$ : 스트라이드 크기
- $n_H$ : 이미지의 높이
- $n_W$ : 이미지의 넓이
- $n_c$ : 채널의 수

- $(l-1)$  번째 층의 이미지 크기 =

$$n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$$

- $l$  번째 층의 높이 혹은 넓이의 크기연산 공식은 아래와 같습니다.
- $n_H^{[l]} = \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1$
- $n_W^{[l]} = \frac{n_W^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1$

- 합성곱 연산

$n_c^{[l]}$  개의 크기가  $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$  인 필터가 합성곱 연산을 진행하게 됩니다. 그리고 활성화 함수를 거쳐  $l$  번째 층의 결과값이 계산됩니다. 합성곱 연산에 사용된 변수는 총  $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$  개 입니다. 추가된 편향까지 더하면 한 층의 합성곱 신경망에 필요한 변수는  $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]} + n_c^{[l]}$  개가 됩니다.

→ 기존의 단순 신경망을 사용하면,  $W^{[l]}$

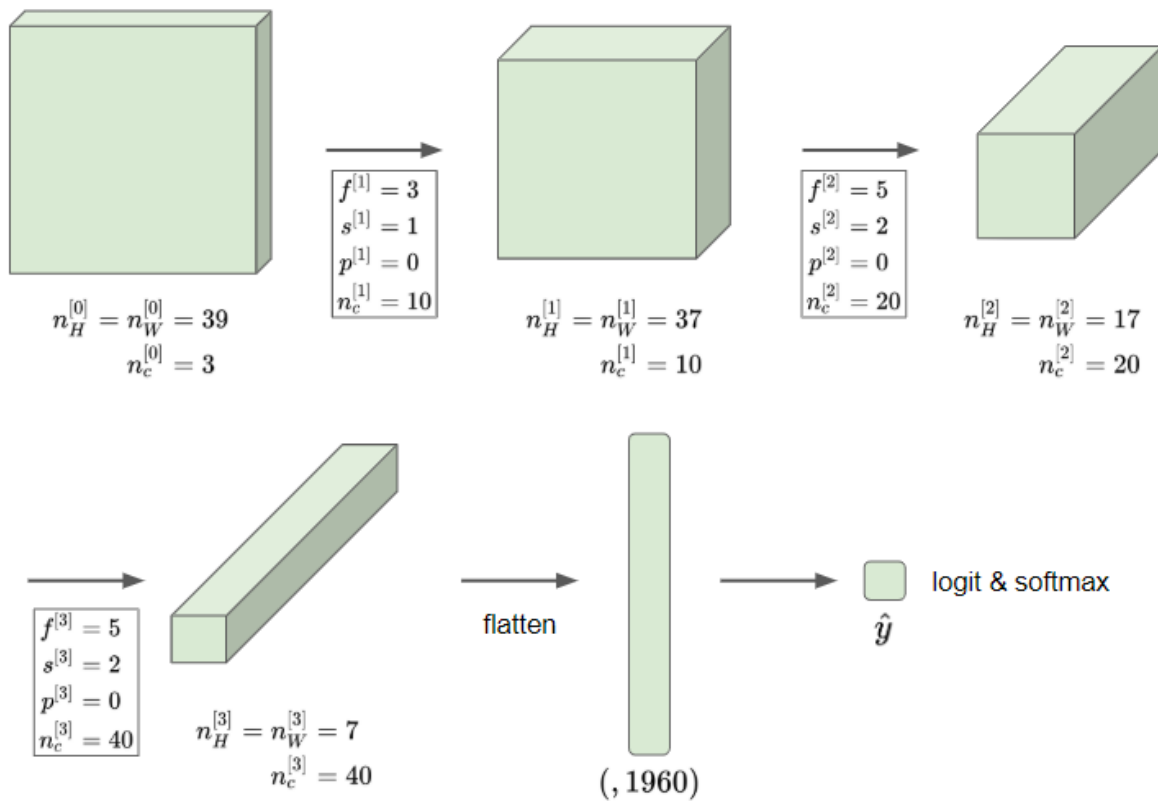
→ 이보다 더 적은 변수를 사용하여 계산 가능

- ex)  $28 \times 28 \times 3$  이미지를 동일한  $5 \times 5$  필터 20개를 사용하여 계산(패딩 X, 스트라이드 1)

→ 결과 크기:  $24 \times 24 \times 20$

- 합성곱 연산에 필요한 총 변수의 크기 =  $5 \times 5 \times 3 \times 20 + 20 = 1520$
- 단순 신경망 사용:  $(28 \times 28 \times 3) \times (24 \times 24 \times 20) = 27106560$  개의 변수 필요

## 8. 간단한 합성곱 네트워크 예시

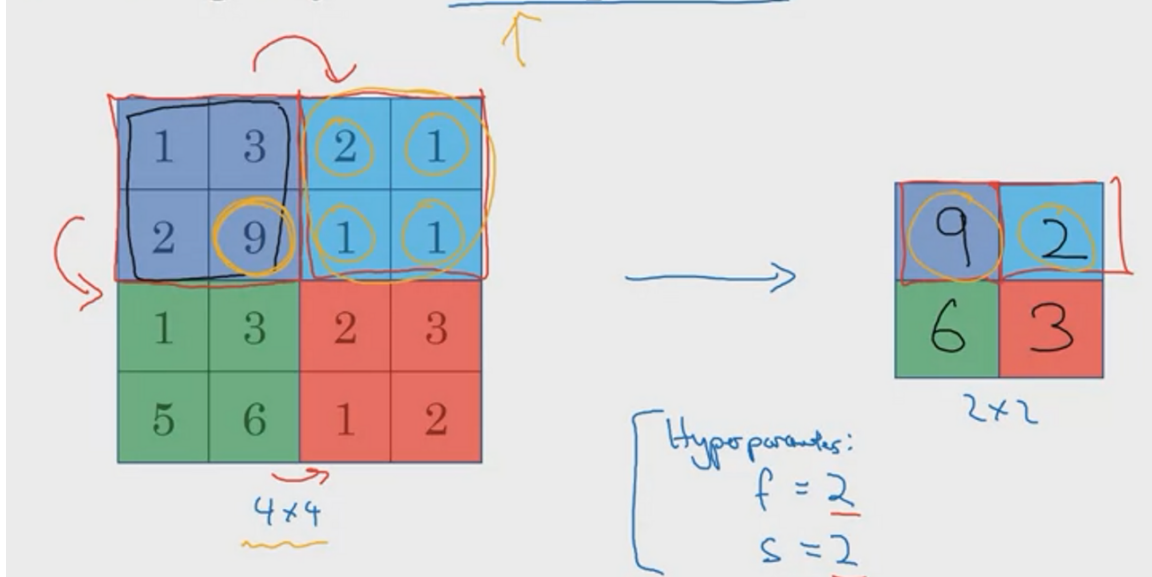


- 합성곱 신경망의 크기는 깊어질 수록 점점 줄어든다.
- 대부분의 신경망에는 합성곱 층, 풀링 층, 완전 연결 층으로 구성

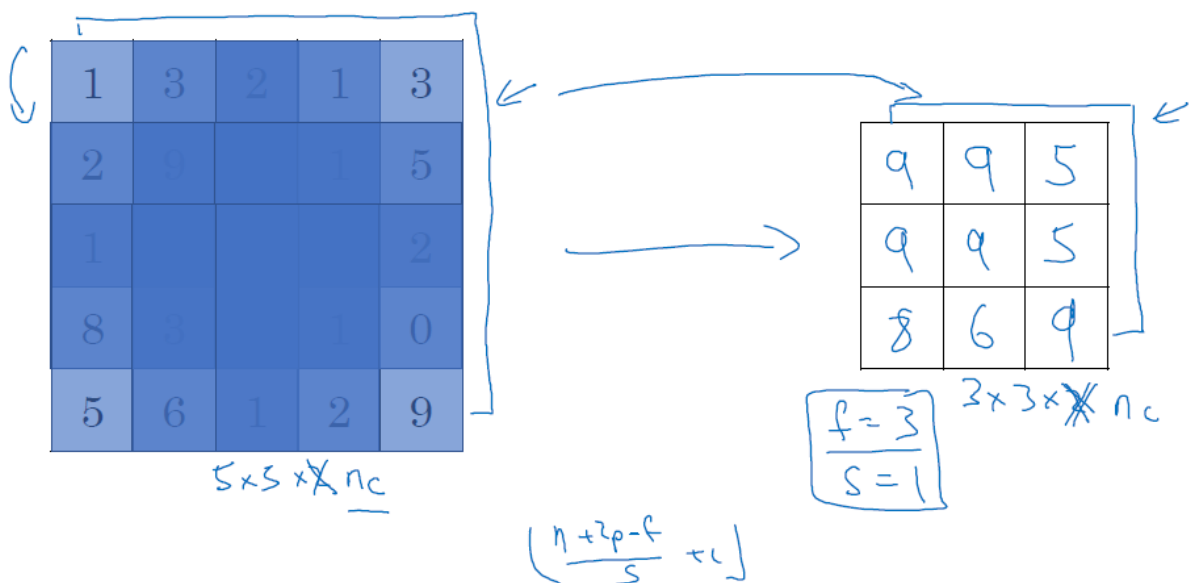
## 9. 풀링층

- **Max Pooling**
  - metrics를 구역으로 나눠서 '최댓값'을 filter에 적용
  - ex) 4x4 metrics에서 stride 2만큼 이동하여 2x2 filter에 적용

## Pooling layer: Max pooling

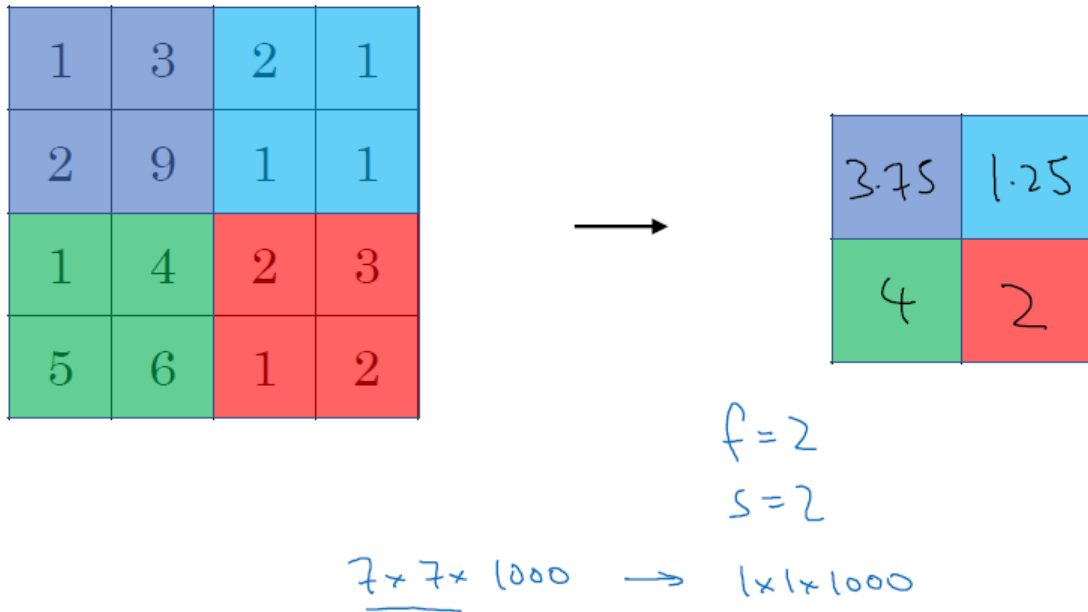


- 하이퍼파라미터: 필터 크기인 **filter size, f**와 이동 범위인 **stride, s**.
  - output layer 크기: convolution operation과 동일하게  $[(n+2p-f)/s + 1]$ 의 내림과 동일
  - feature들이 필터에 감지되면 입력하고, 감지되지 않으면 입력하지 않는 방법
- 직관적으로 feature의 분포를 확인 가능.
- ex) 5x5 input layer에 3x3 filter에 stride 1을 적용하면, output은 3x3 layer



- **Average Pooling**

- max pooling과 동일하게 filter를 적용하지만, 최댓값이 아닌 **평균값을 적용**. 신경망에서는 max pooling 보다는 average pooling을 많이 사용.



### • Summary of Pooling

- max pooling을 적용할 때는 대부분 padding은 사용X (padding = 0)
- 

Hyperparameters:

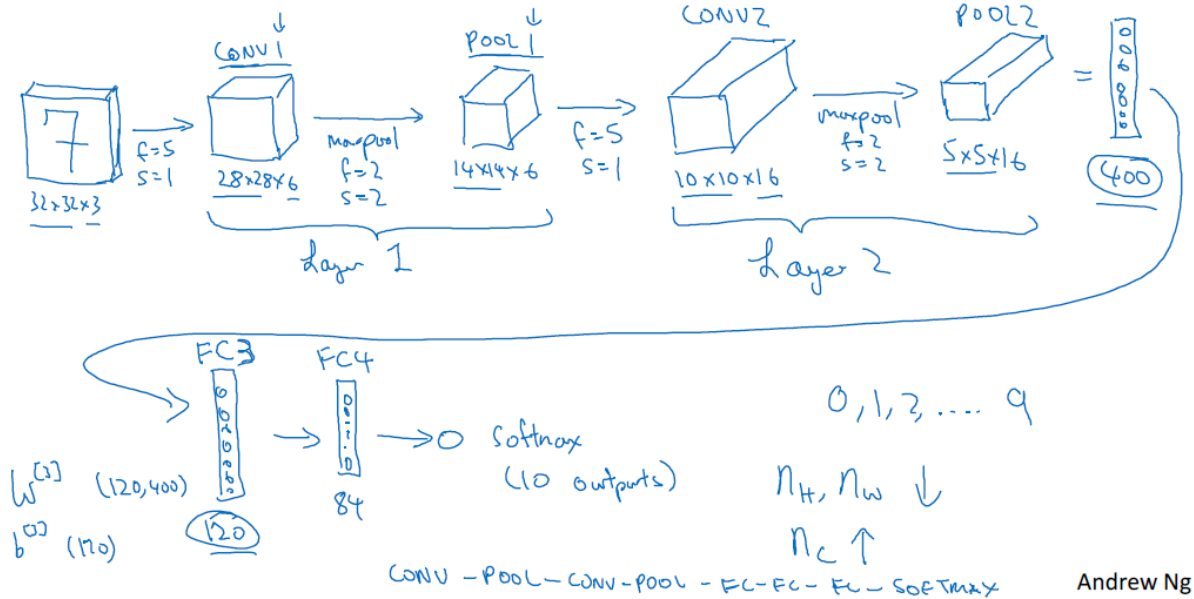
$$\begin{aligned}
 & \left[ \begin{array}{l} f : \text{filter size} \\ s : \text{stride} \\ \text{Max or average pooling} \end{array} \right. \quad \begin{array}{l} f=2, s=2 \\ f=3, s=2 \end{array} \\
 & \Rightarrow \text{padding} \\
 & \text{No parameters to learn!}
 \end{aligned}$$

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

- 합성곱 신경망에서는 풀링 층을 사용하여 **표현의 크기를 줄임**으로써, **계산 속도를 줄이고 특징을 더 잘 검출 가능**
- 최대 연산의 역할: 이미지의 특징이 필터의 한 부분에서 검출되면, 높은 수를 남기고 그렇지 않으면 다른 최대값들에 비해 상대적으로 작아져서 **특징을 더 잘 남긴다**.

## 10. CNN 예시

### Neural network example (LeNet-5)



- convolution net은 LeNet-5과 유사

- convolution layer과 pooling layer을 묶어서 하나의 레이어 1로 취급**

→ pooling layer에는 가중치, 파라미터도 없으며 하이퍼파라미터만 있기에, 단독 레이어로 취급하기보다는 convolution layer와 묶어서 1개의 레이어로 취급. (레이어를 셀 때는 가중치가 있는 레이어로 count)

- FC (Fully connected) layer로 120개 유닛을 만들기. (layer 3)
- 그리고 84개의 (120, 84) FC layer을 만들고, 출력 함수로 softmax로 손글씨를 인식.
- 신경망이 깊어질수록  $n_h, n_w$ 는 줄어들이지만,  $n_c$ 는 커짐.**

# Neural network example

	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 } 10,081 }
FC4	(84,1)	84	
Softmax	(10,1)	10	841

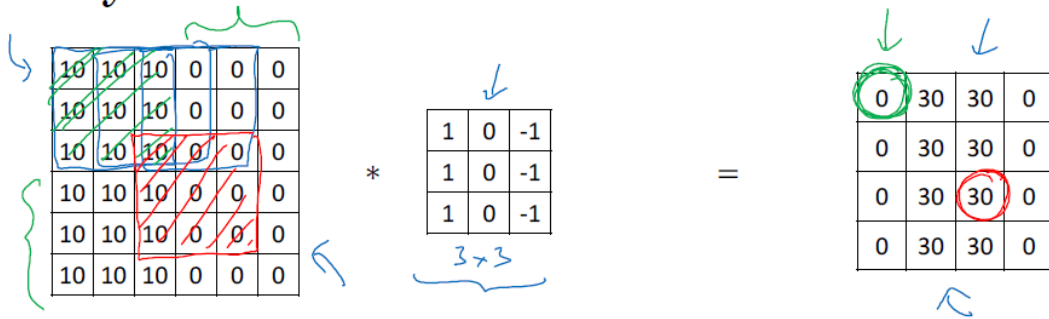
- max pooling layer: 파라미터 X
- convolution layer: 파라미터가 상대적으로 적고, FC layer가 많음.
- activation size는 점차 줄어들며, 빠르게 줄어드는 것은 성능에 좋지 않.

## 11. 왜 합성곱을 사용할까요?

- 합성곱 신경망을 사용하면 변수를 적게 사용할 수 있음.
  - ex) 32 x 32 x 3 이미지를 5 x 5 필터 6개를 통해 28 x 28 x 6 의 이미지로 합성곱 연산을 했을 경우, 필요한 변수의 개수는  $5 \times 5 \times 3 \times 6 + 6 = 456$
  - 일반적인 신경망으로는  $3,072 \times 4,704 + 4,704$ , 약 1400 만개의 변수가 필요.
- 합성곱 신경망이 이렇게 적은 변수를 필요로 하는 이유: **변수 공유.**
  - 어떤 한 부분에서 이미지의 특성을 검출하는 필터가 이미지의 다른 부분에서도 똑같이 적용되거나 도움이 됨
- 다른 이유: **희소 연결**
  - 출력값이 이미지의 일부(작은 입력값)에 영향을 받고, 나머지 픽셀들의 영향을 받지 않기 때문에, 과대적합을 방지 가능
- 합성곱 신경망은 이동 불변성을 포착하는데도 용이.
  - 이미지가 약간의 변형이 있어도 포착 가

# Why convolutions

translation invariance



**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.