Convolutional Neural Network

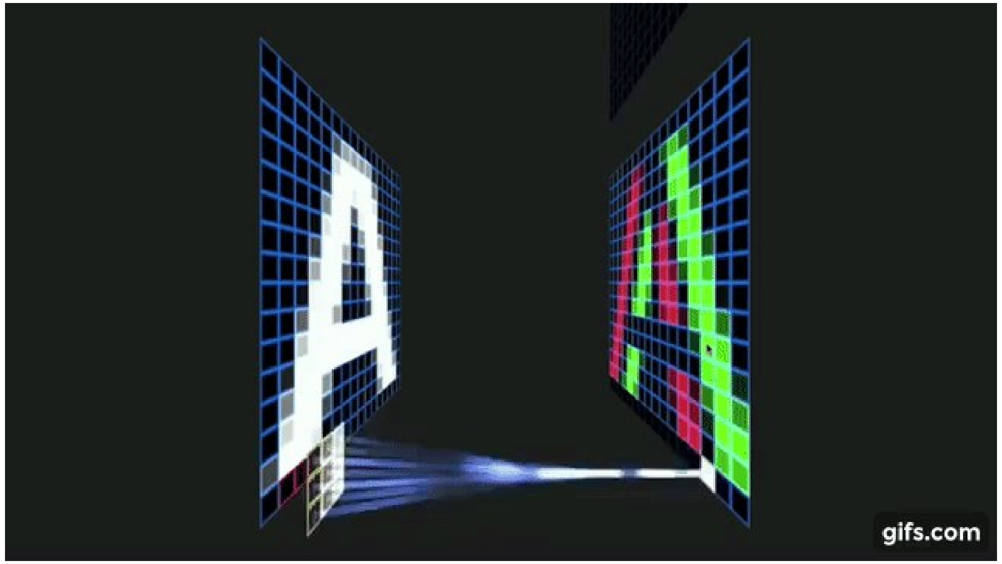
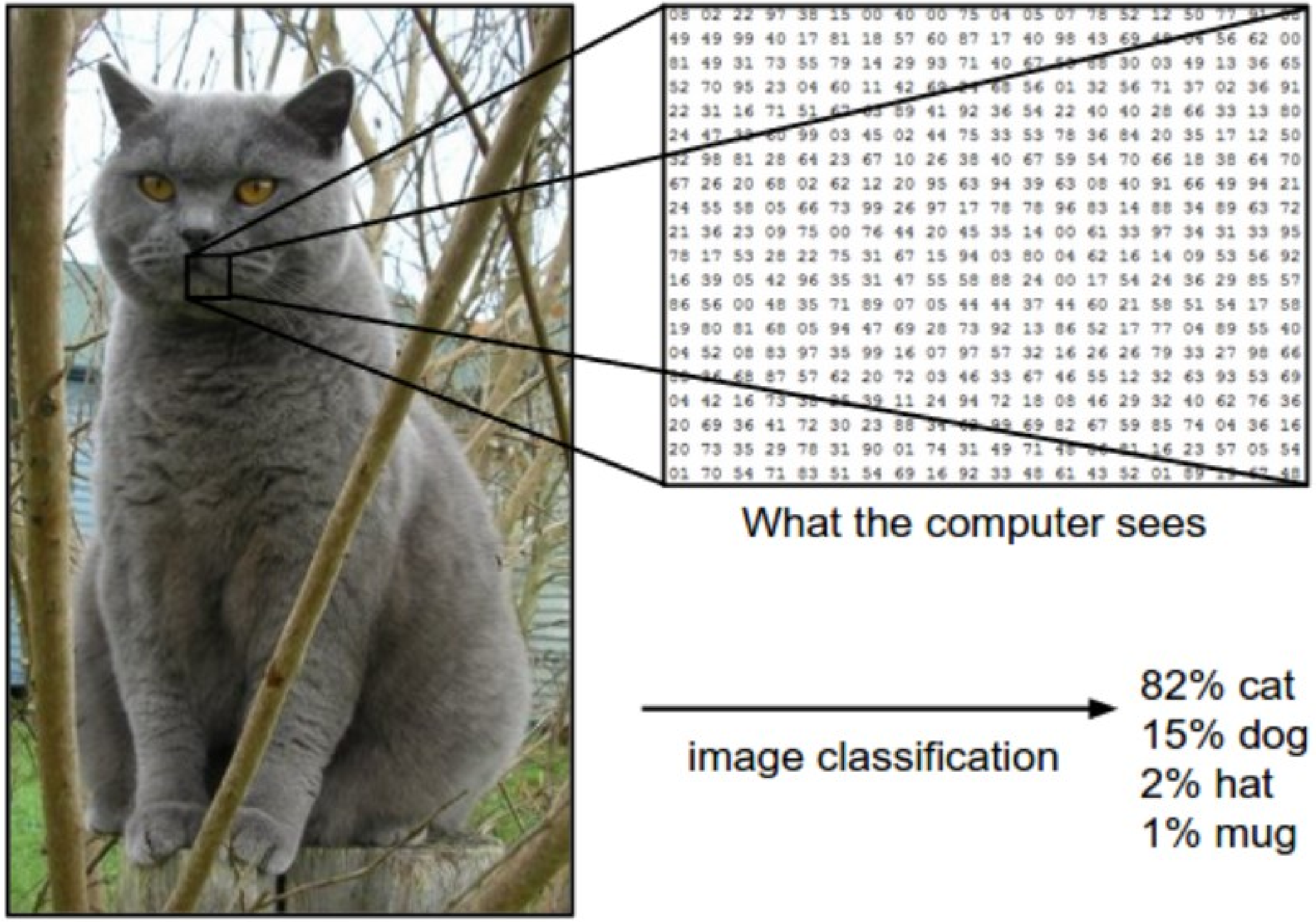


Image Classification

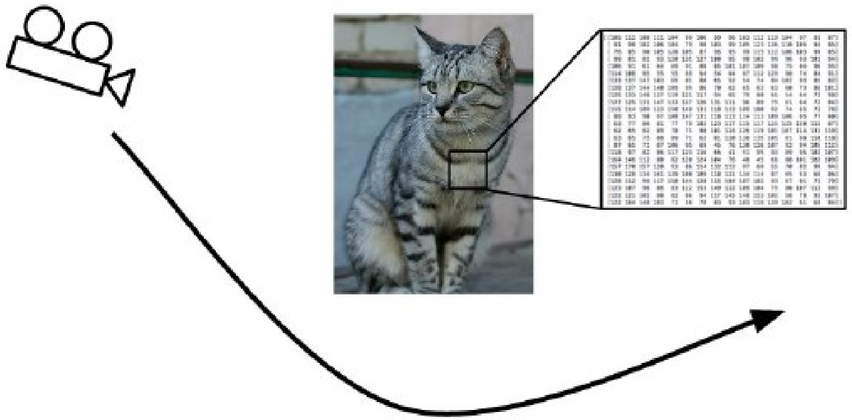
- A core task in Computer Vision



Slide Credit : Stanford CS231n

Challenges of Recognition

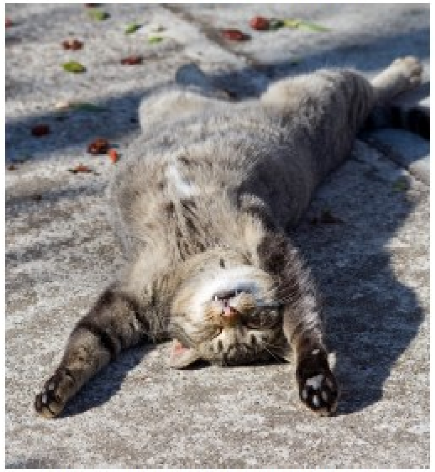
Viewpoint



Illumination

This image is CC0 1.0 public domain

Deformation



This image by Umberto Salvagnin is licensed under CC-BY 2.0

Occlusion

This image by jonsson is licensed under CC-BY 2.0

Clutter

This image is CC0 1.0 public domain

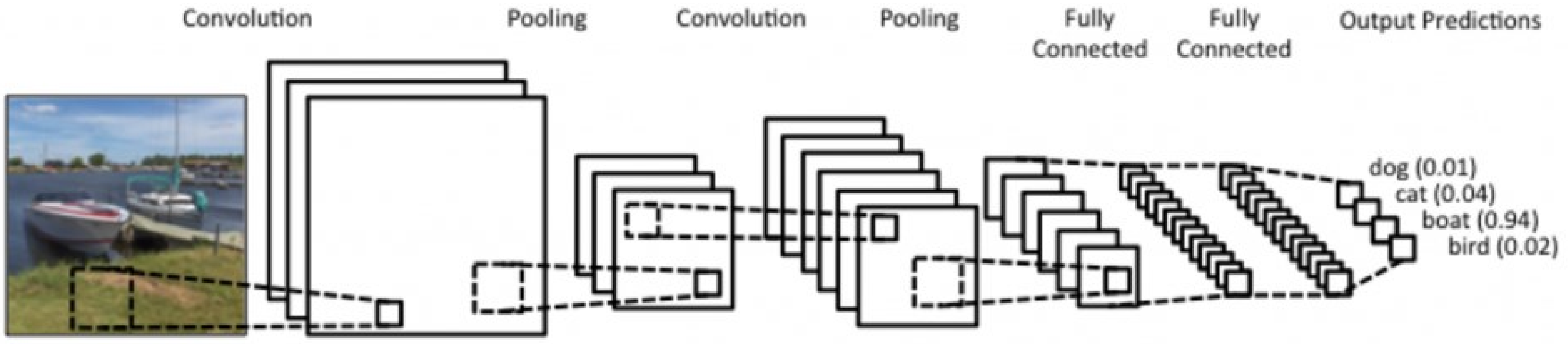
Intraclass Variation

This image is CC0 1.0 public domain

Slide Credit : Stanford CS231n

Convolutional Neural Network

- Most widely used for image classification.
- Generally, it consists of convolution layer, pooling layer and fully-connected layer.
- Convolution, Pooling layer – feature extraction
- Fully-connected layer – classification



Convolution Filters(Hand Crafted)

0	0	0	0	0
0	0	-1	0	0
0	-1	5	-1	0
0	0	-1	0	0
0	0	0	0	0

	0	1	0	
	1	-4	1	
	0	1	0	

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

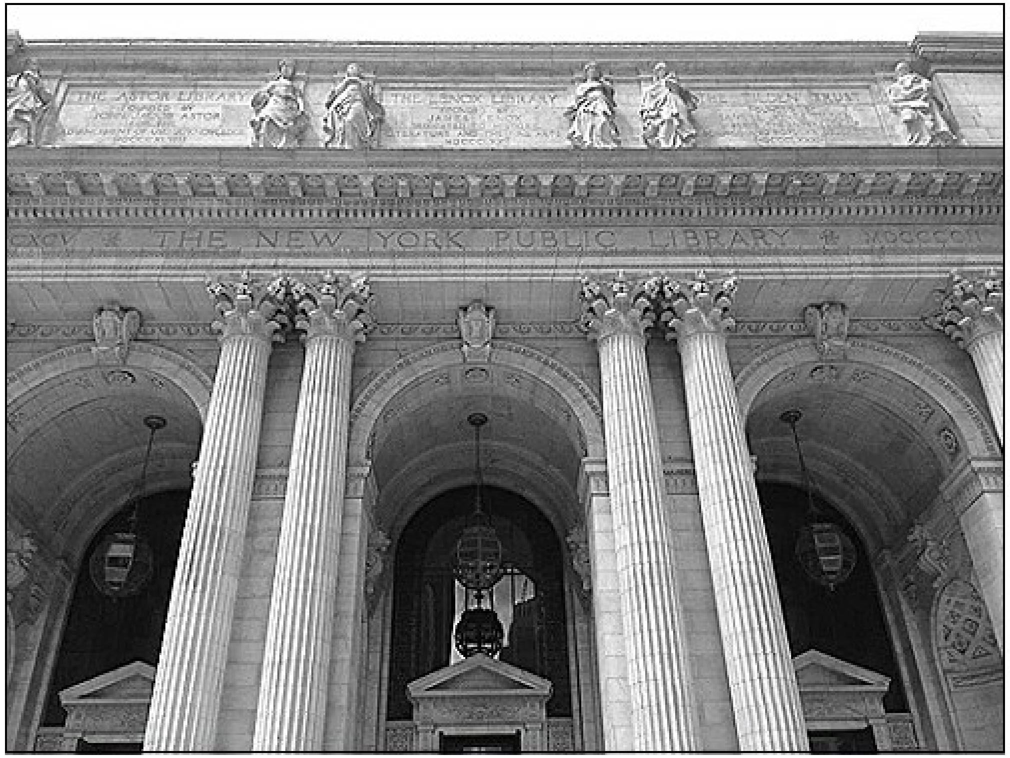
	-2	-1	0	
	-1	1	1	
	0	1	2	

Let’s Try!

- http://setosa.io/ev/image-kernels/

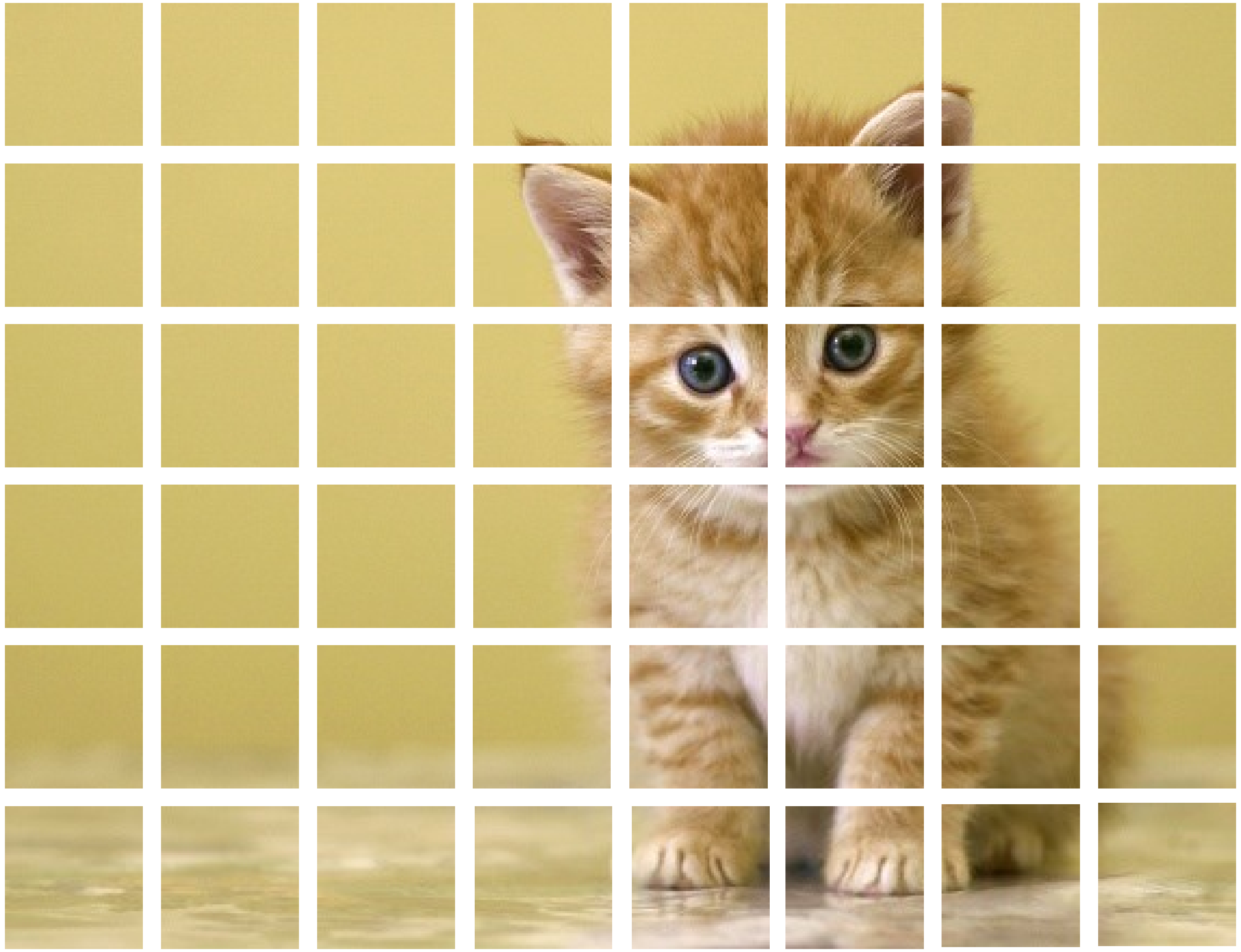
0	-1	0
-1	5	-1
0	-1	0

sharpen

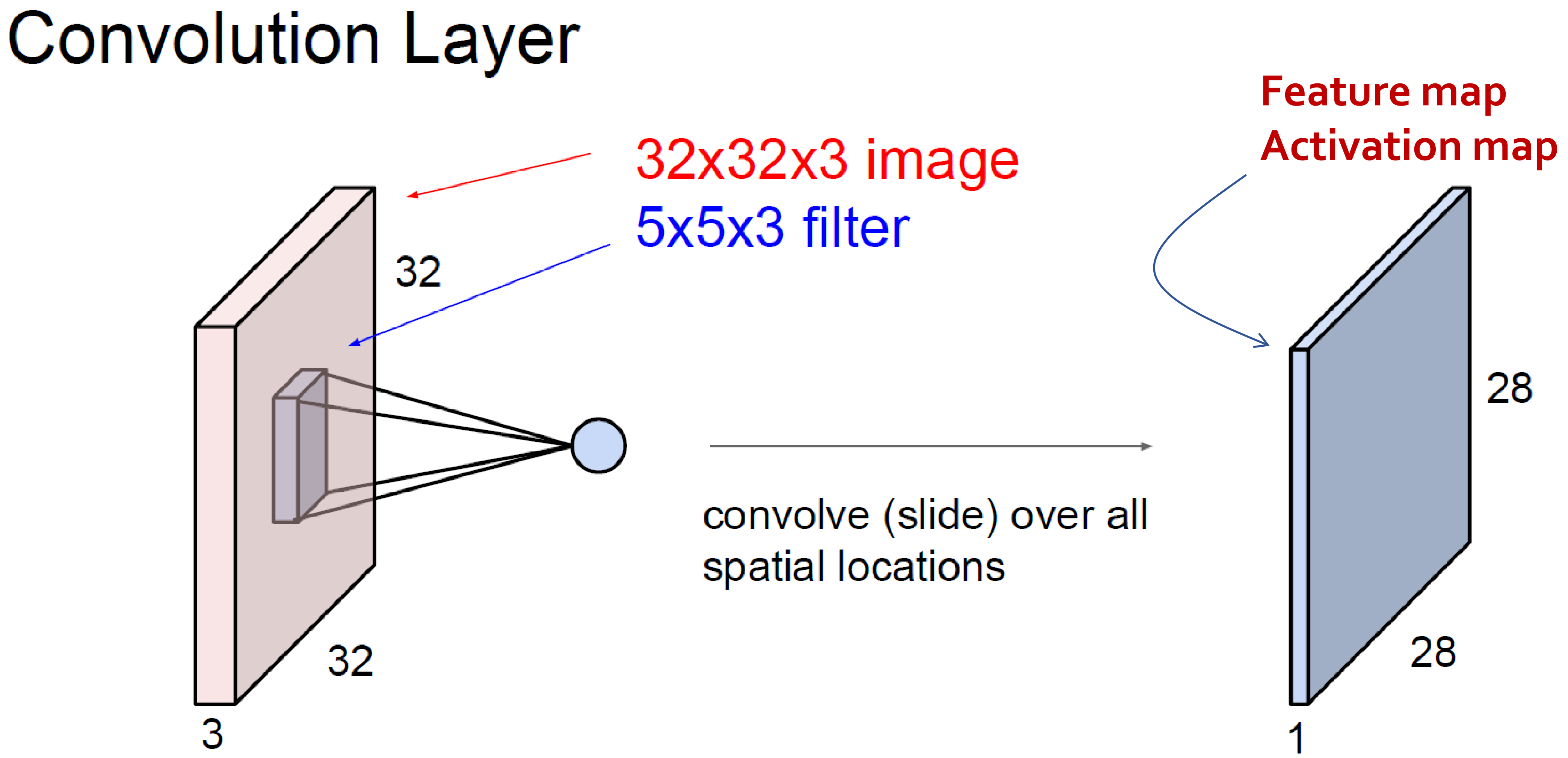


CNN 동작원리

- 이미지를 작은 tile로 나누고, convolution filter를 통해 tile에서 특정 feature를 추출(예: 귀)
- Filter가 다음 tile로 이동하면서 같은 방법으로feature를 추출(동일한 weight 사용)
- 다른 feature(예: 눈)를 추출하는 filter를 추가로 만들고 위와 같은 방법으로 tile을 하나씩 network에 적용
- 추출된 모든 feature들을 잘 조합하여 최종적으로 이미지를 판단

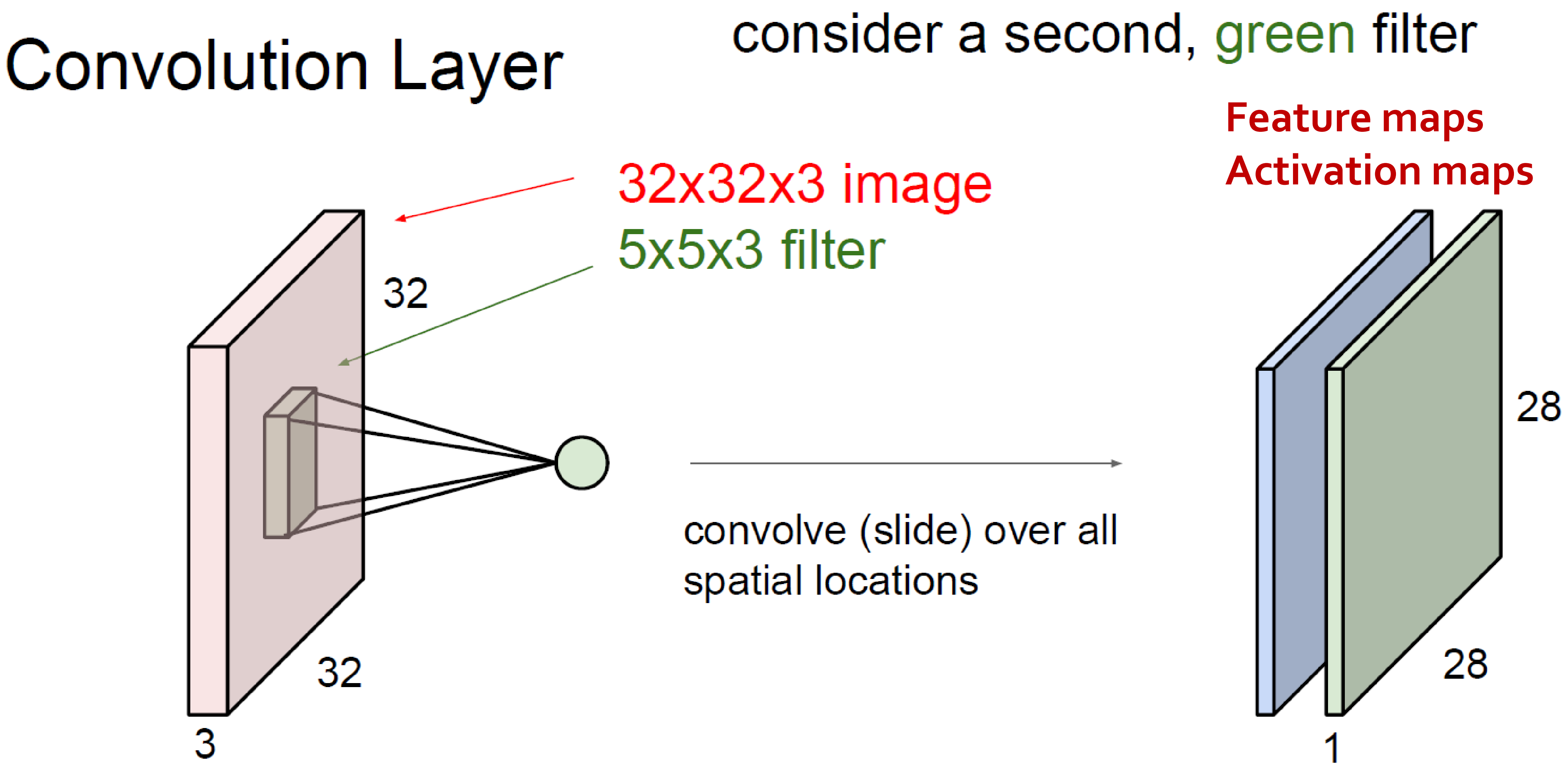


2D Convolution Layer



Slide Credit : Stanford CS231n

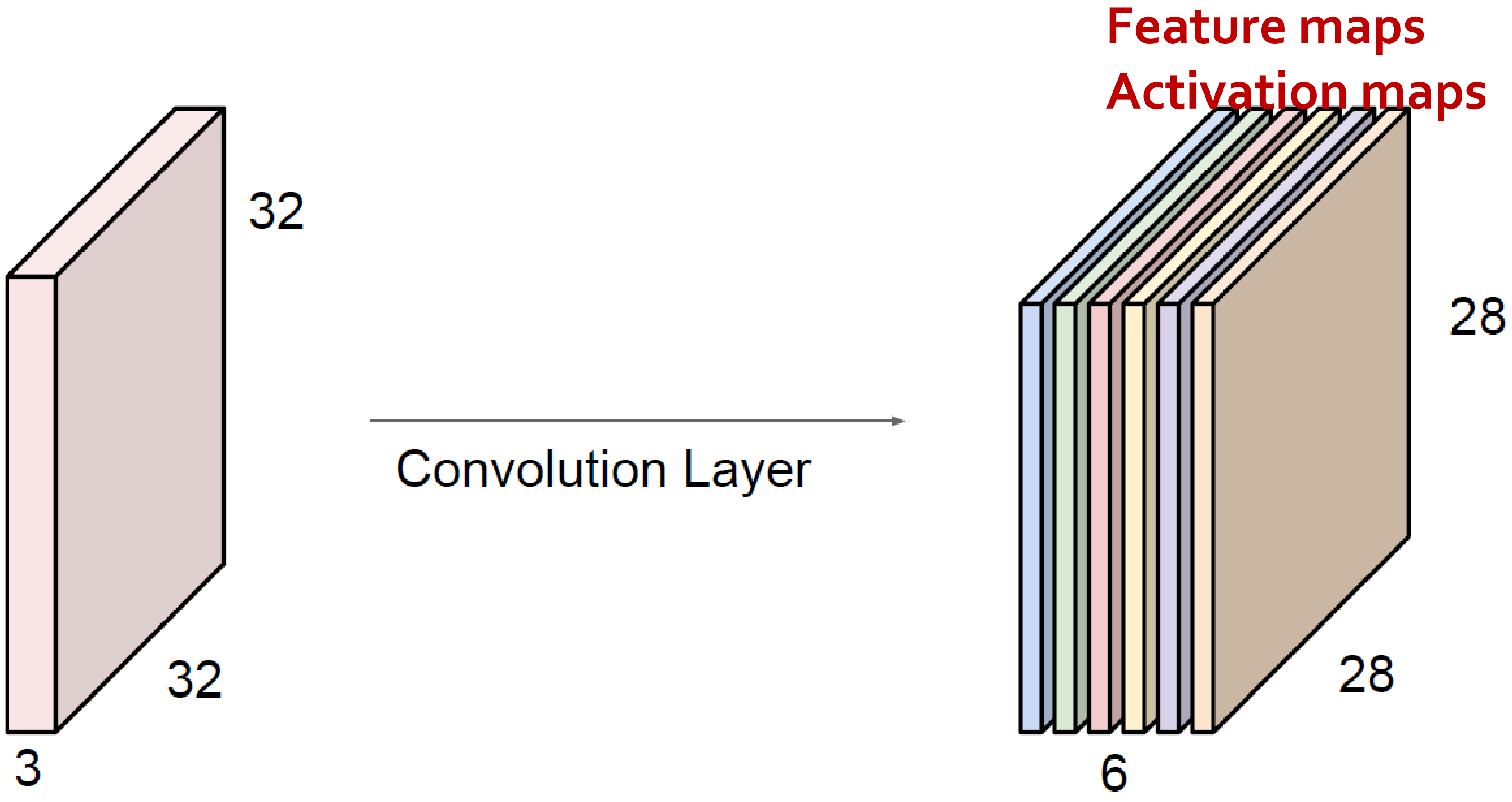
2D Convolution Layer



Slide Credit : Stanford CS231n

2D Convolution Layer

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

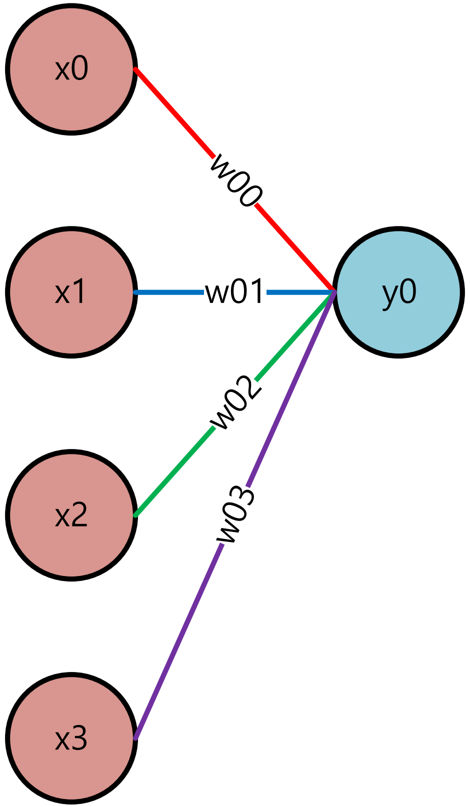


We stack these up to get a "new image" of size 28x28x6!

Slide Credit : Stanford CS231n

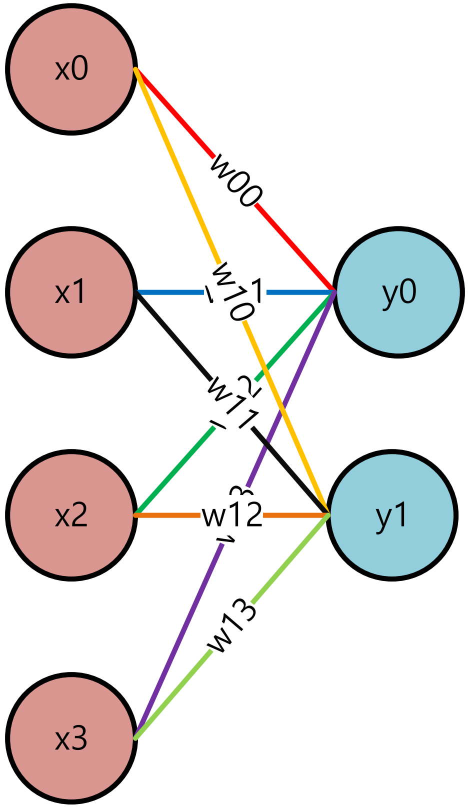
Dense Layer vs 1-D Convolution Layer

- Dense Layer(Fully Connected Layer)
  - $y0 = x0 \cdot w00 + x1 \cdot w01 + x2 \cdot w02 + x3 \cdot w03$



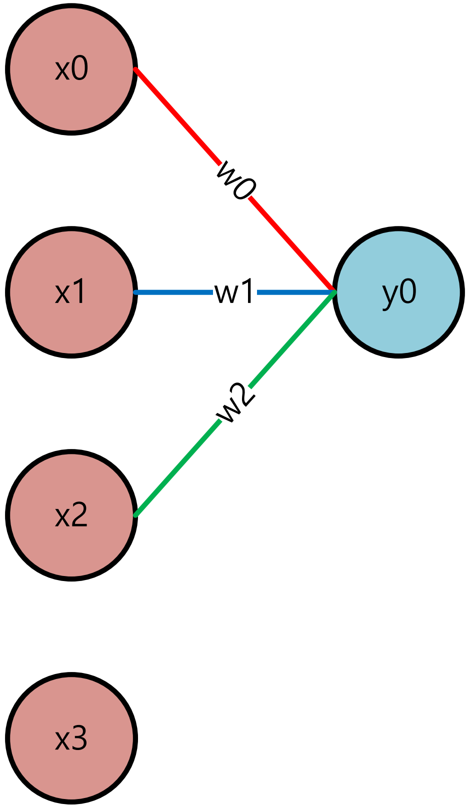
Dense Layer vs 1-D Convolution Layer

- Dense Layer(Fully Connected Layer)
  - $y0 = x0 \cdot w00 + x1 \cdot w01 + x2 \cdot w02 + x3 \cdot w03$
  - $y1 = x0 \cdot w10 + x1 \cdot w11 + x2 \cdot w12 + x3 \cdot w13$



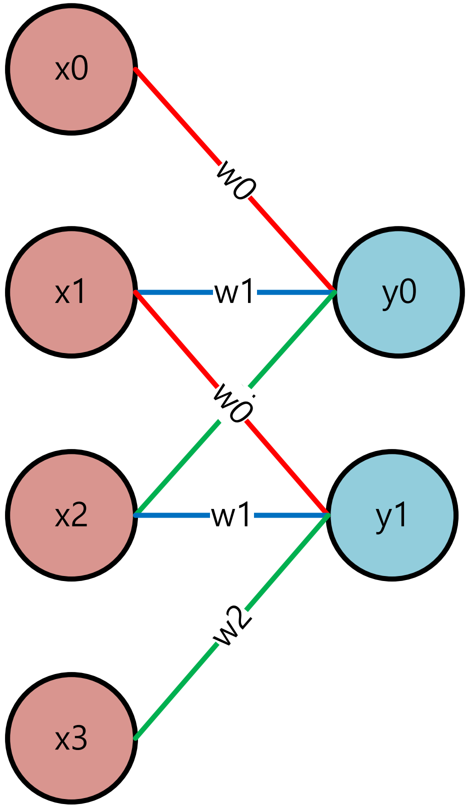
Dense Layer vs 1-D Convolution Layer

- 1-D Convolution Layer
  - $y0 = x0 \cdot w0 + x1 \cdot w1 + x2 \cdot w2$



Dense Layer vs 1-D Convolution Layer

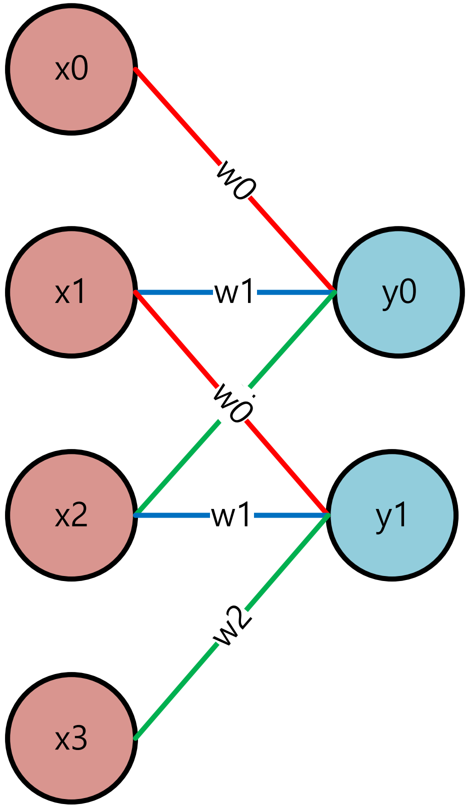
- 1-D Convolution Layer
  - $y0 = x0 \cdot w0 + x1 \cdot w1 + x2 \cdot w2$
  - $y0 = x1 \cdot w0 + x2 \cdot w1 + x3 \cdot w2$



Dense Layer vs 1-D Convolution Layer

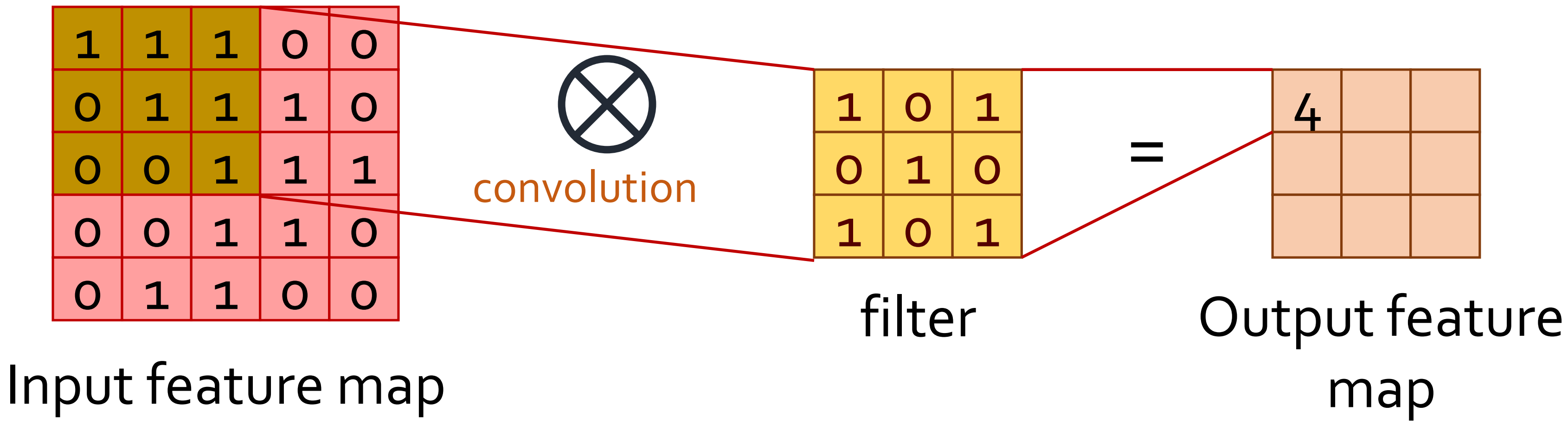
- 1-D Convolution Layer
  - $y0 = x0 \cdot w0 + x1 \cdot w1 + x2 \cdot w2$
  - $y0 = x1 \cdot w0 + x2 \cdot w1 + x3 \cdot w2$

Weight sharing
&
Locally connected



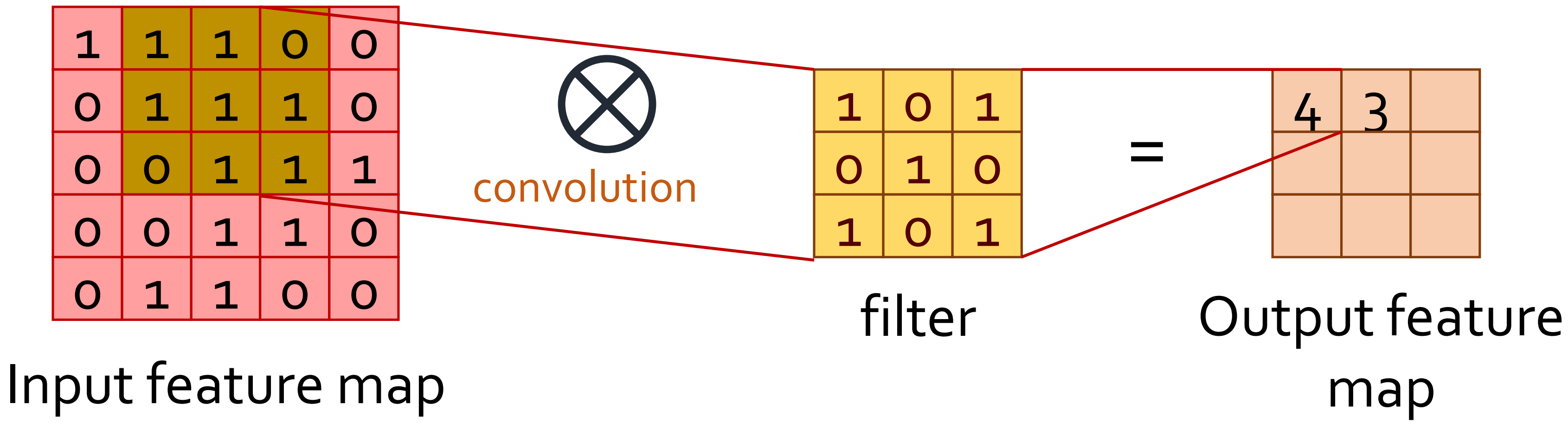
2D Convolution Layer – Computation

- 1X1 + 1X0 + 1X1 + 0X0 + 1X1 + 1X0 + 0X1 + 0X0 + 1X1 = 4



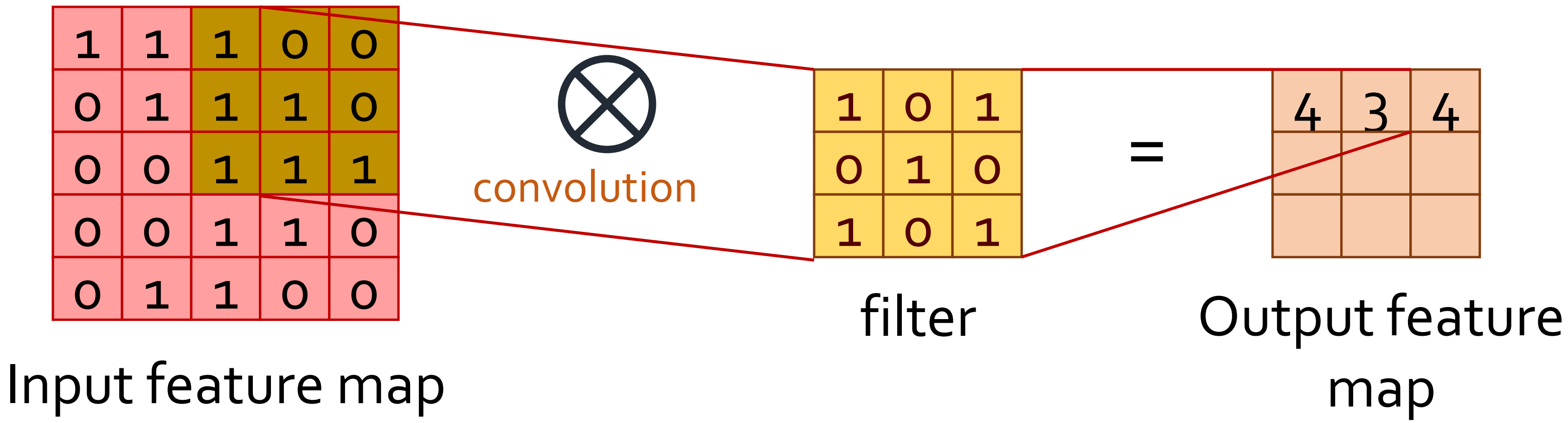
2D Convolution Layer – Computation

- 1X1 + 1X0 + 0X1 + 1X0 + 1X1 + 1X0 + 0X1 + 1X0 + 1X1 = 3



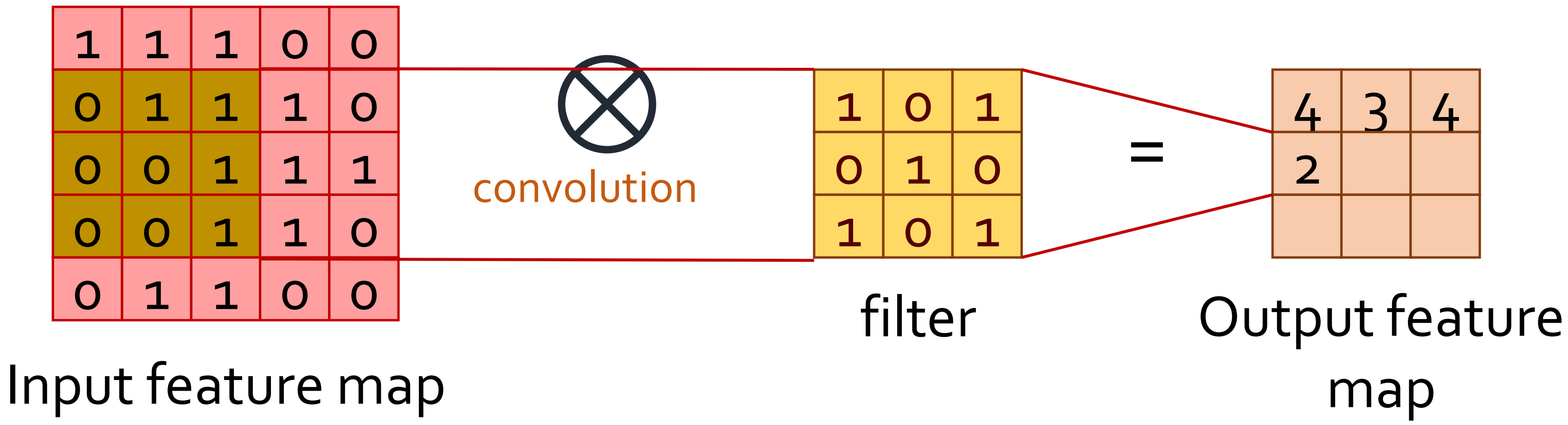
2D Convolution Layer – Computation

- $1X1 + 0X0 + 0X1 + 1X0 + 1X1 + 0X0 + 1X1 + 1X0 + 1X1 = 4$



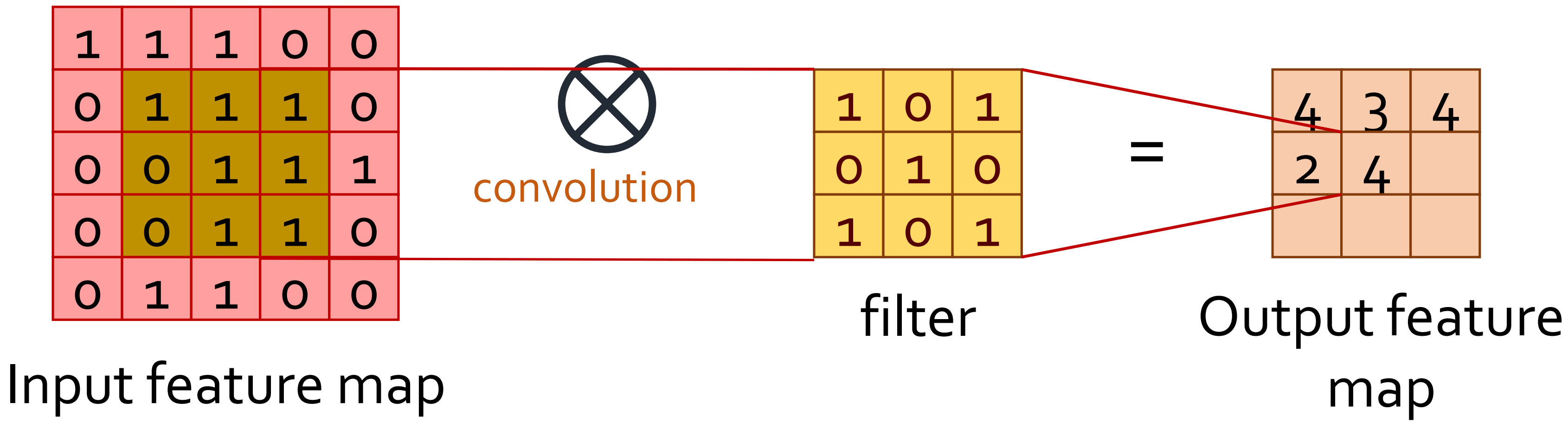
2D Convolution Layer – Computation

- 0X1 + 1X0 + 1X1 + 0X0 + 0X1 + 1X0 + 0X1 + 0X0 + 1X1 = 2



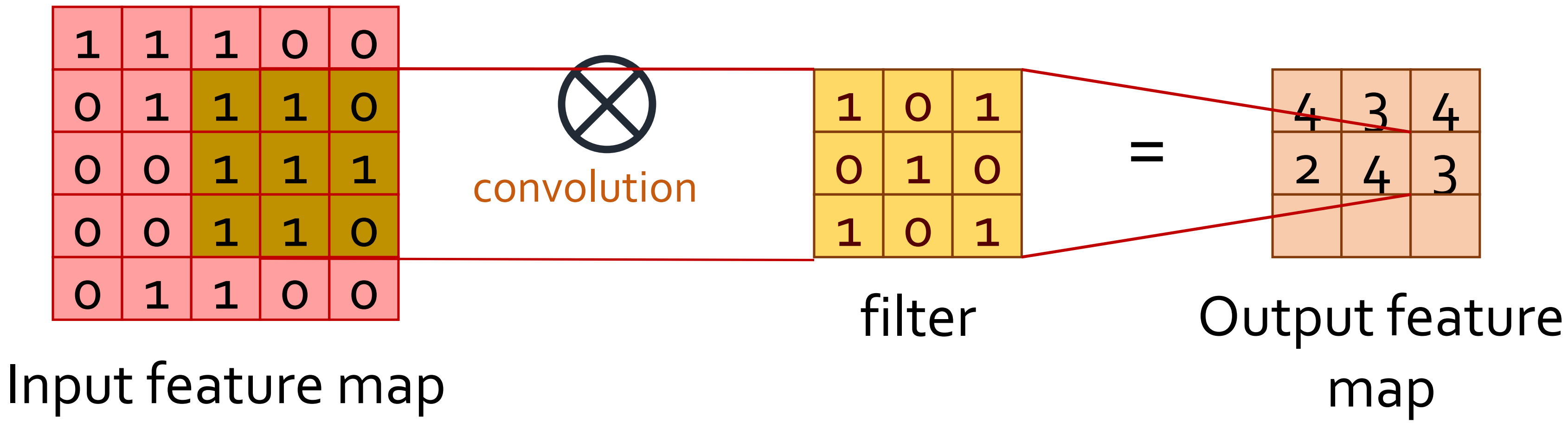
2D Convolution Layer – Computation

- 1X1 + 1X0 + 1X1 + 0X0 + 1X1 + 1X0 + 0X1 + 1X0 + 1X1 = 4



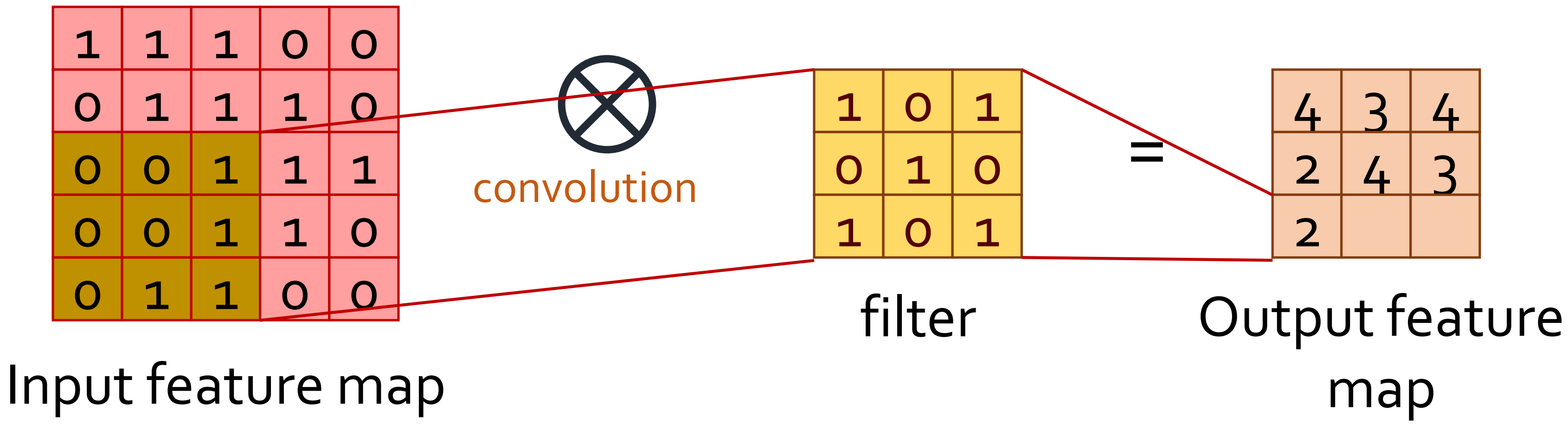
2D Convolution Layer – Computation

- 1X1 + 1X0 + 0X1 + 1X0 + 1X1 + 1X0 + 1X1 + 1X0 + 0X1 = 3



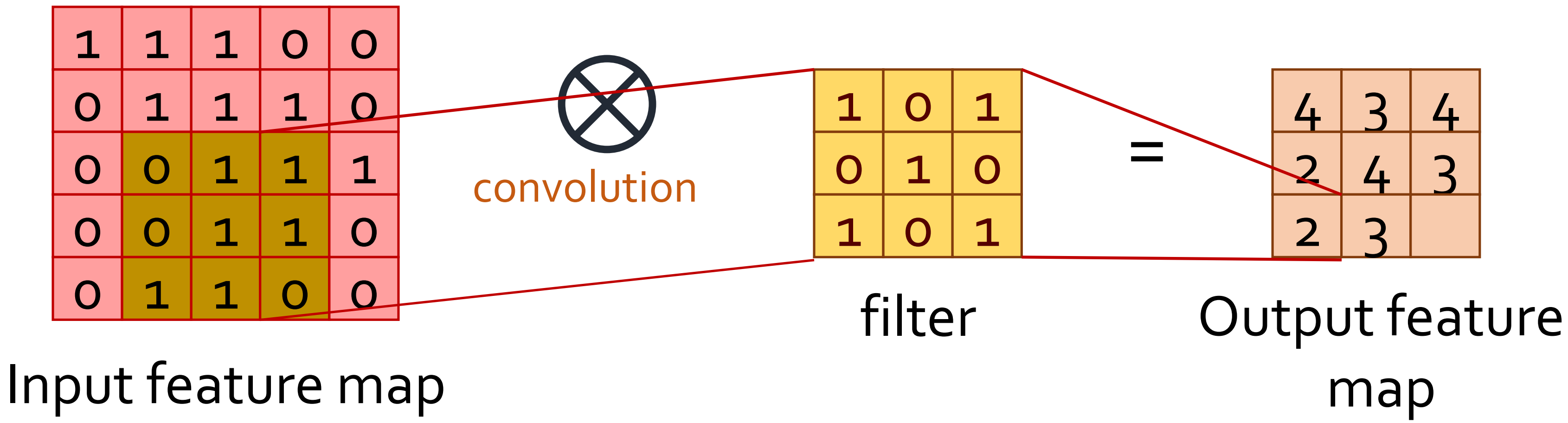
2D Convolution Layer – Computation

- 0X1 + 0X0 + 1X1 + 0X0 + 0X1 + 1X0 + 0X1 + 1X0 + 1X1 = 2



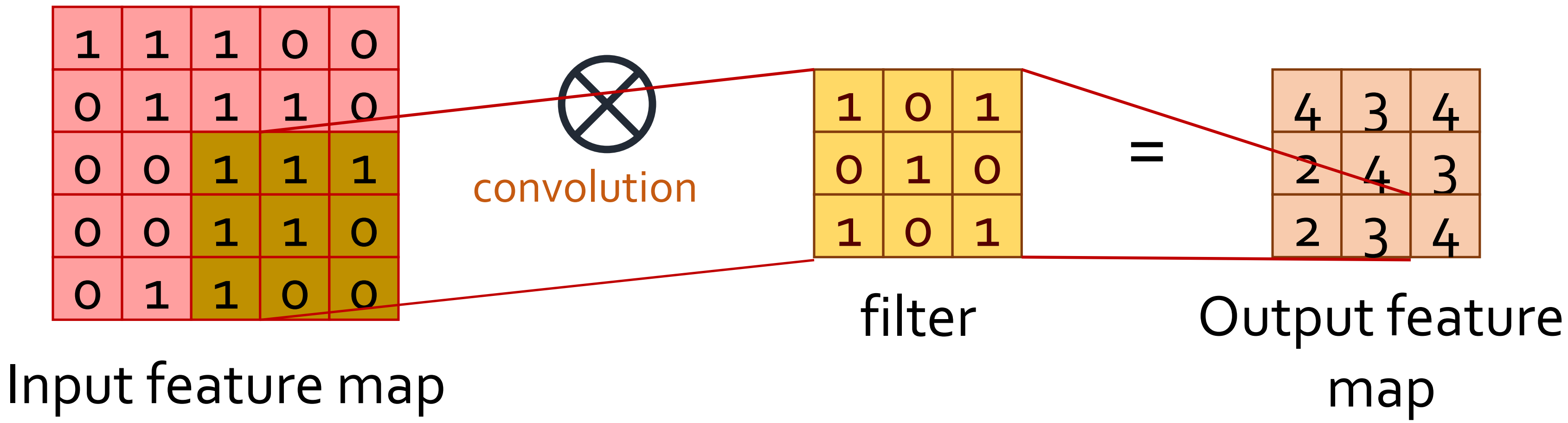
2D Convolution Layer – Computation

- 0X1 + 1X0 + 1X1 + 0X0 + 1X1 + 1X0 + 1X1 + 1X0 + 0X1 = 3

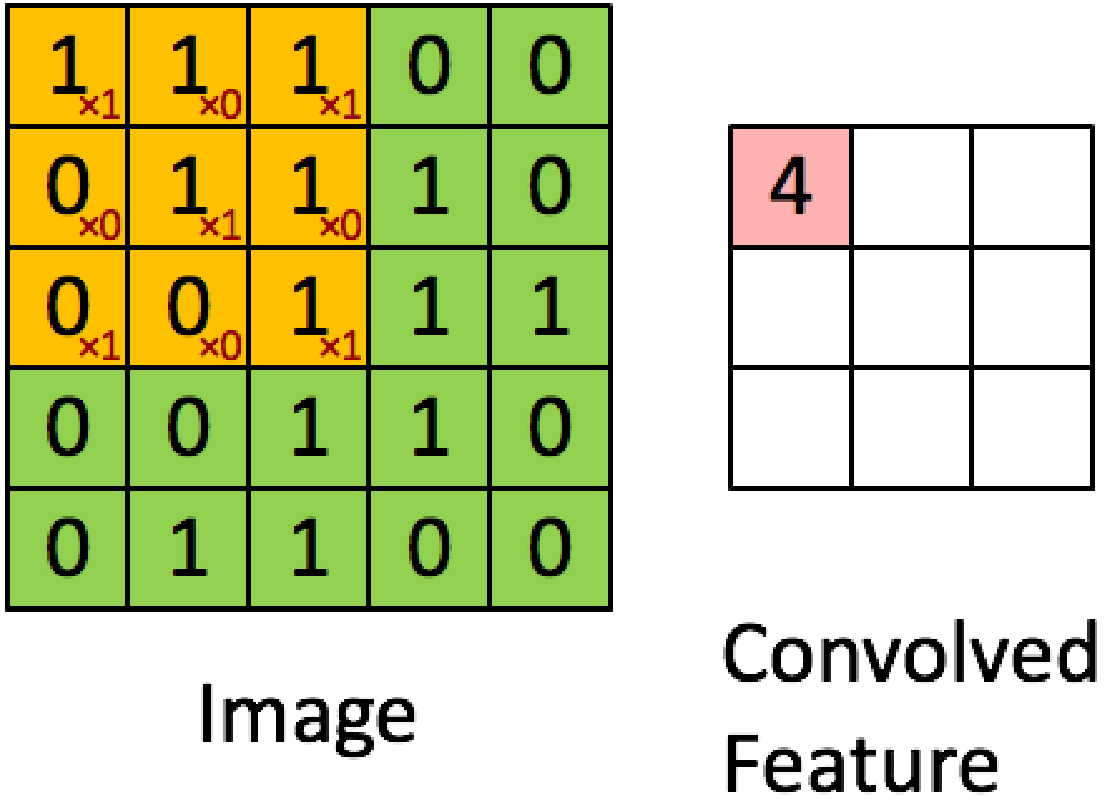


2D Convolution Layer – Computation

- 1X1 + 1X0 + 1X1 + 1X0 + 1X1 + 0X0 + 1X1 + 0X0 + 0X1 = 4



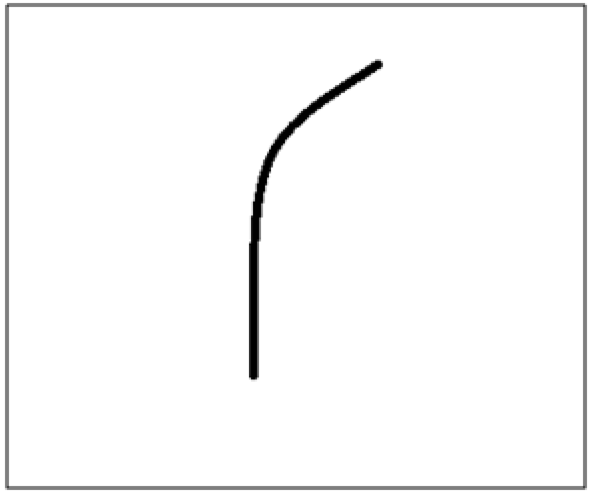
2D Convolution Layer – Computation



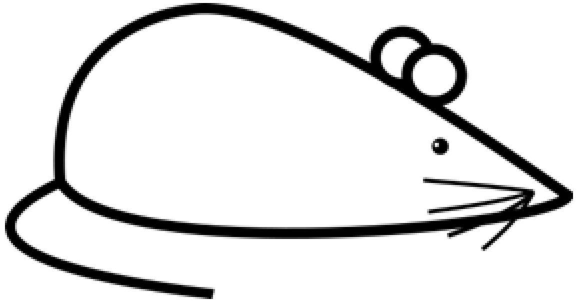
Feature Extractor

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

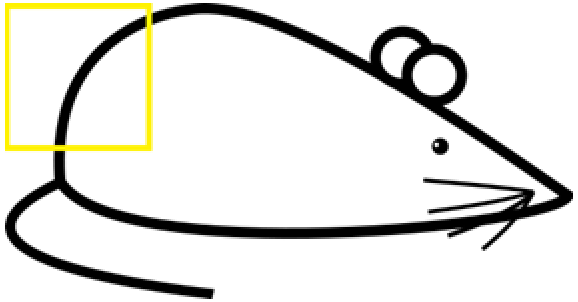
Pixel representation of filter



Visualization of a curve detector filter



Original image



Visualization of the filter on the image

Credit : Adit Deshpande's blog

Feature Extractor

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

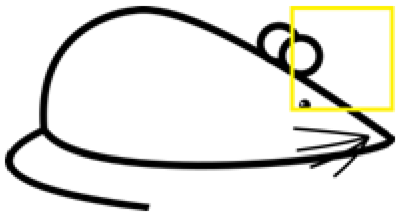
Pixel representation of the receptive field

*

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

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)



Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

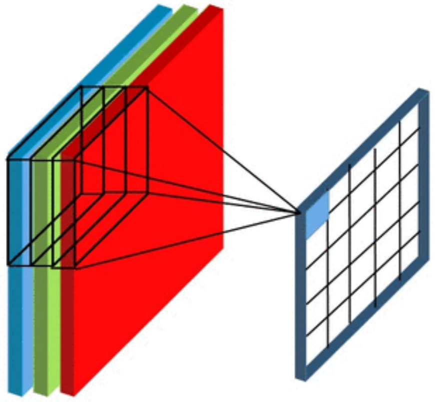
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

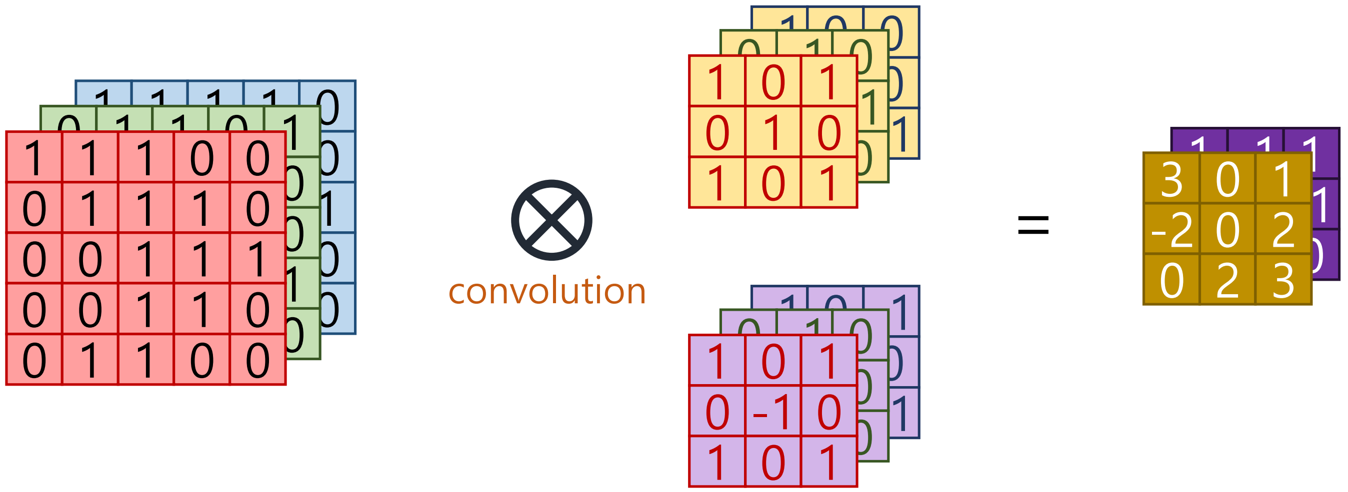
Pixel representation of filter

Multiplication and Summation = 0

Credit : Adit Deshpande's blog

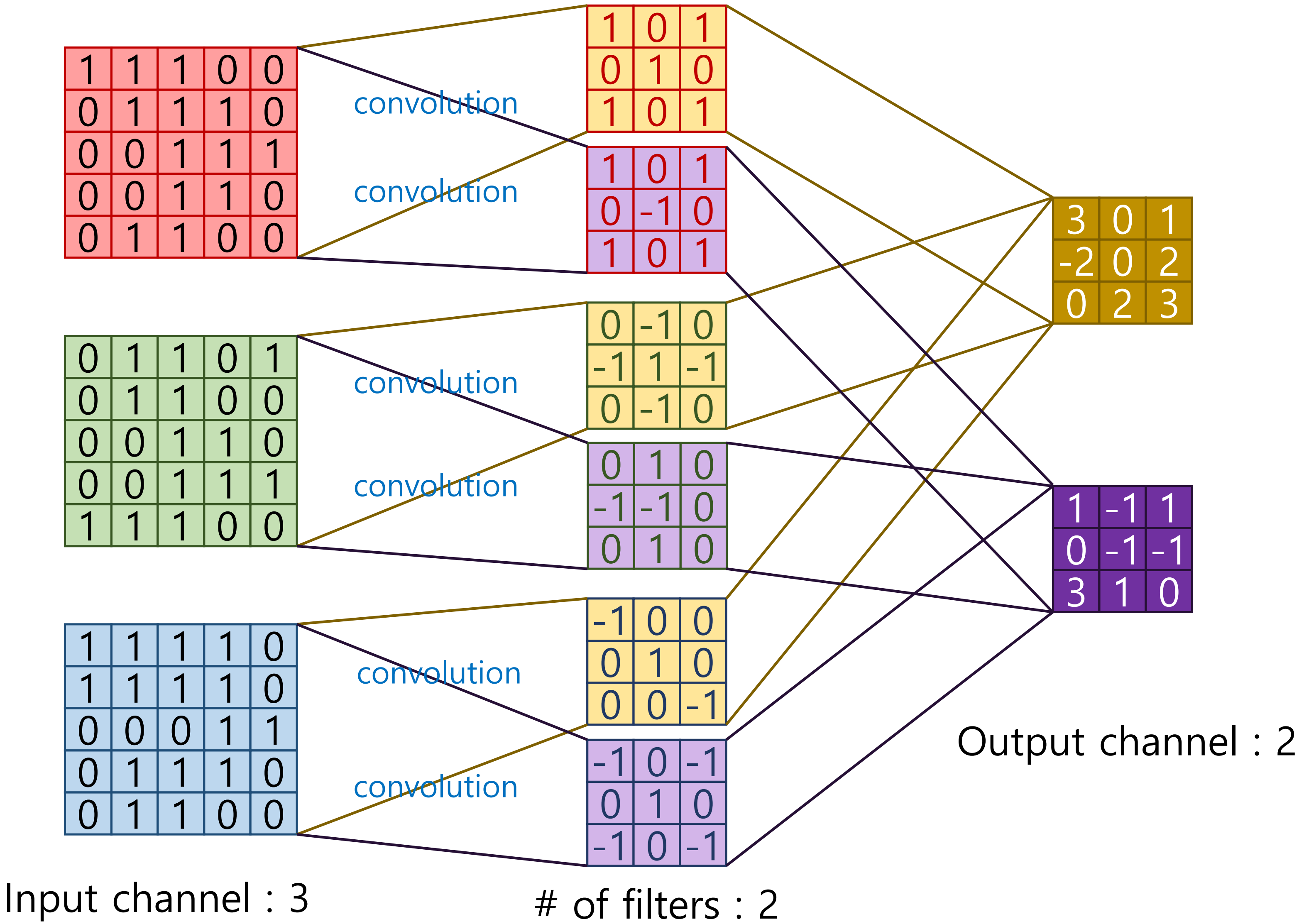
Convolution
(Multi Channel, Many Filters)



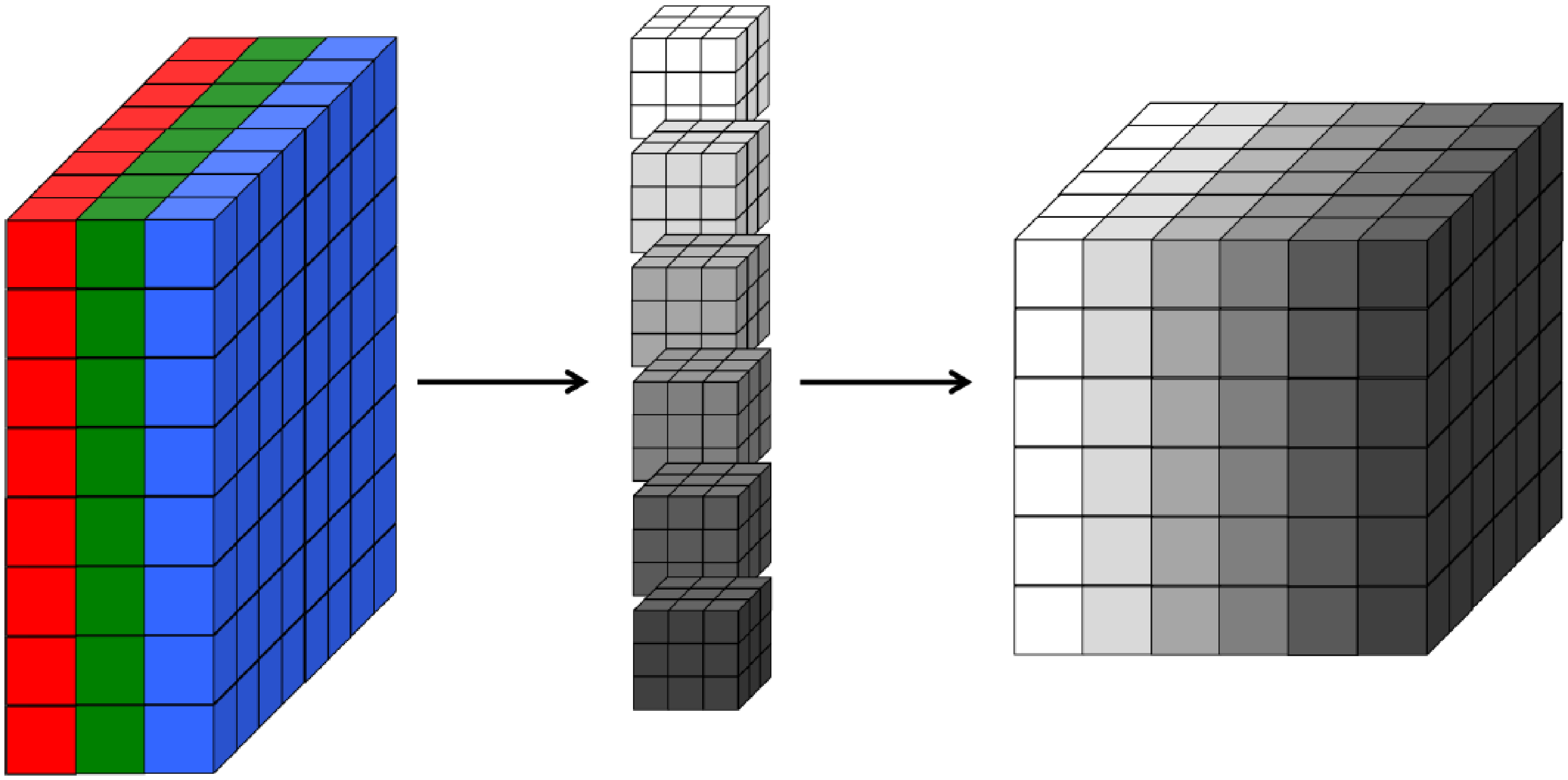


Input channel : 3 # of filters : 2 Output channel : 2

Convolution (Multi Channel, Many Filters)

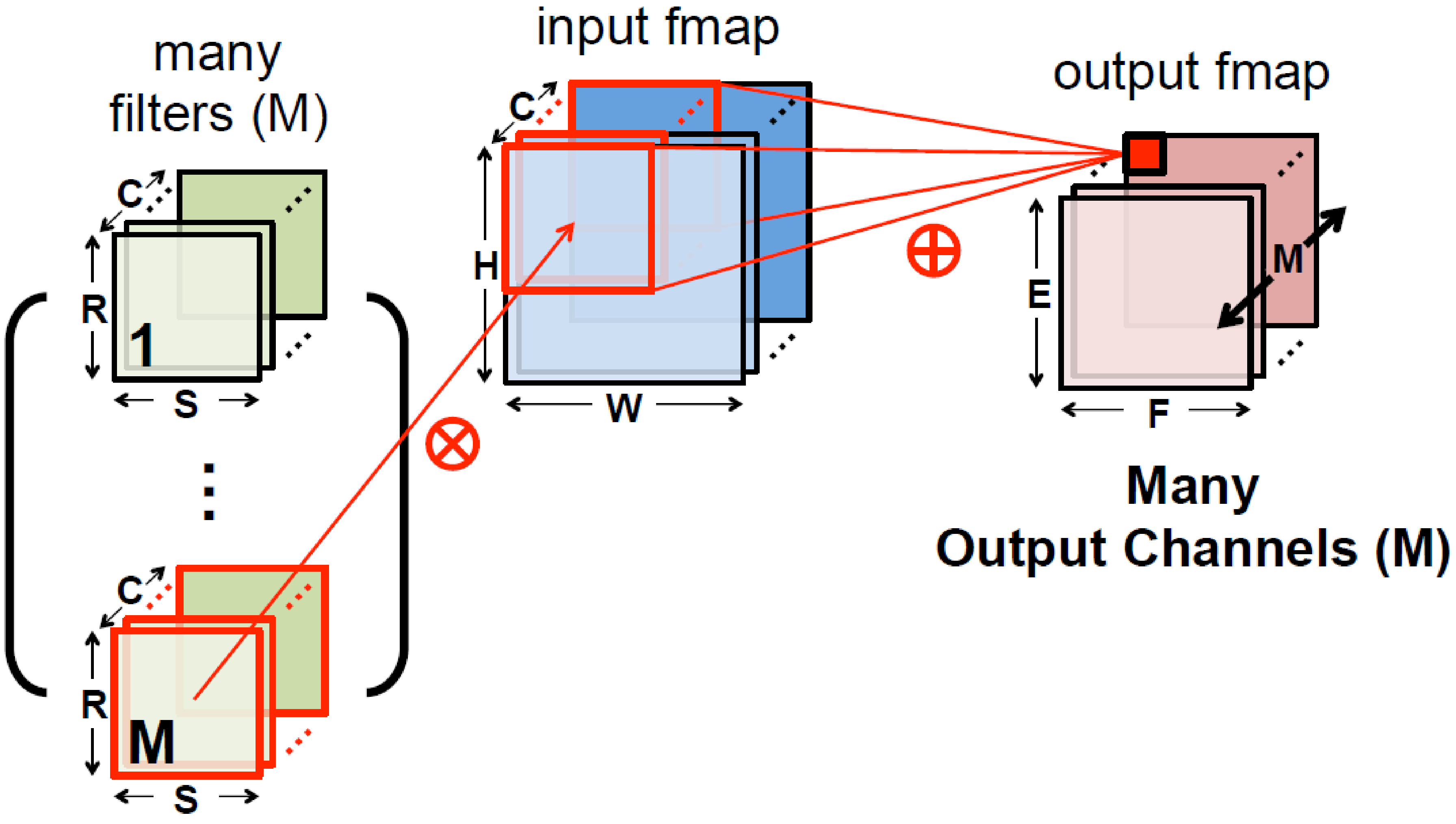


Visualization of a Convolution Layer



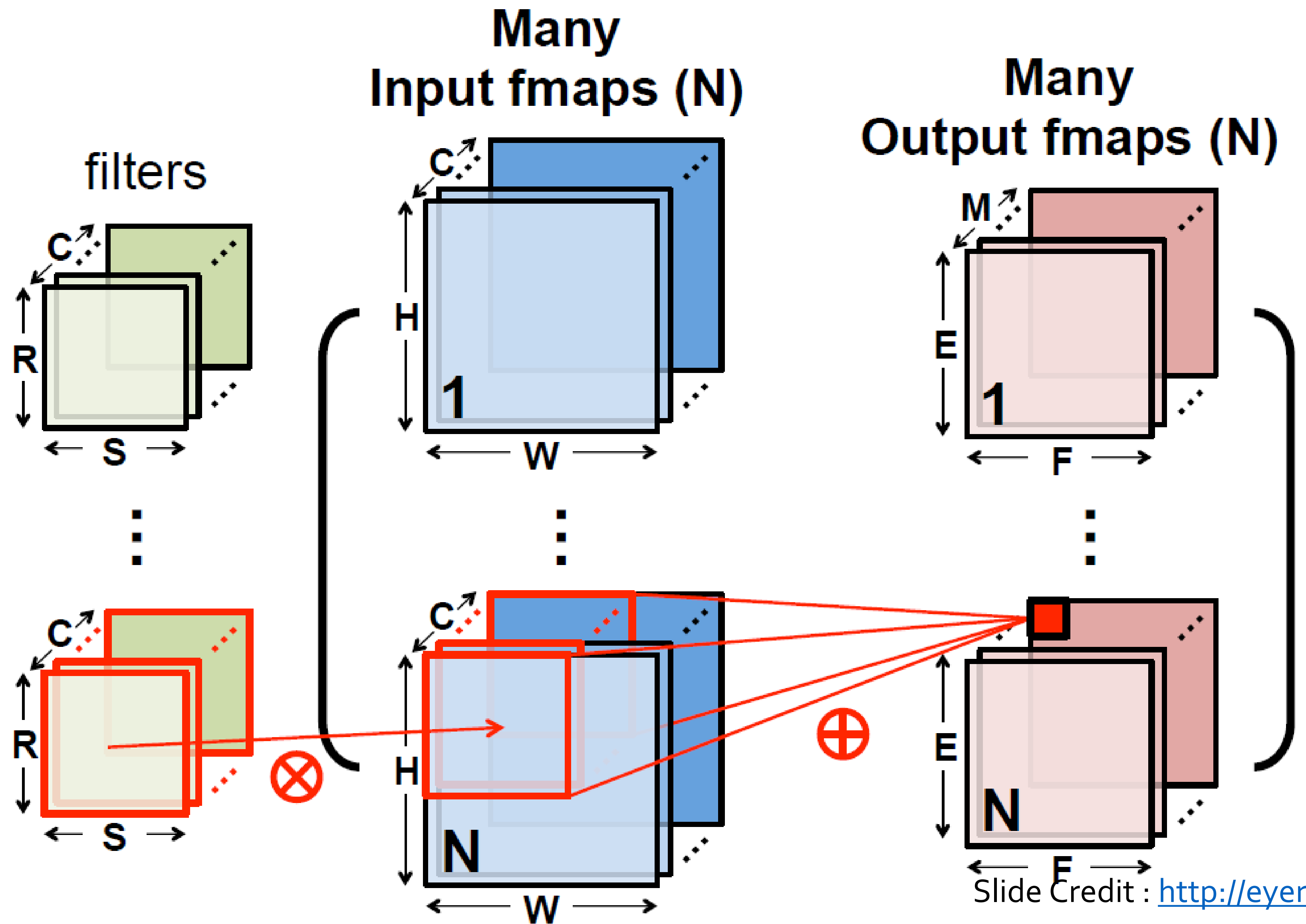
Picture Credit : fundamentals of deep learning by Nikhil

2D Convolution Layer



Slide Credit : http://eyeriss.mit.edu/tutorial.html

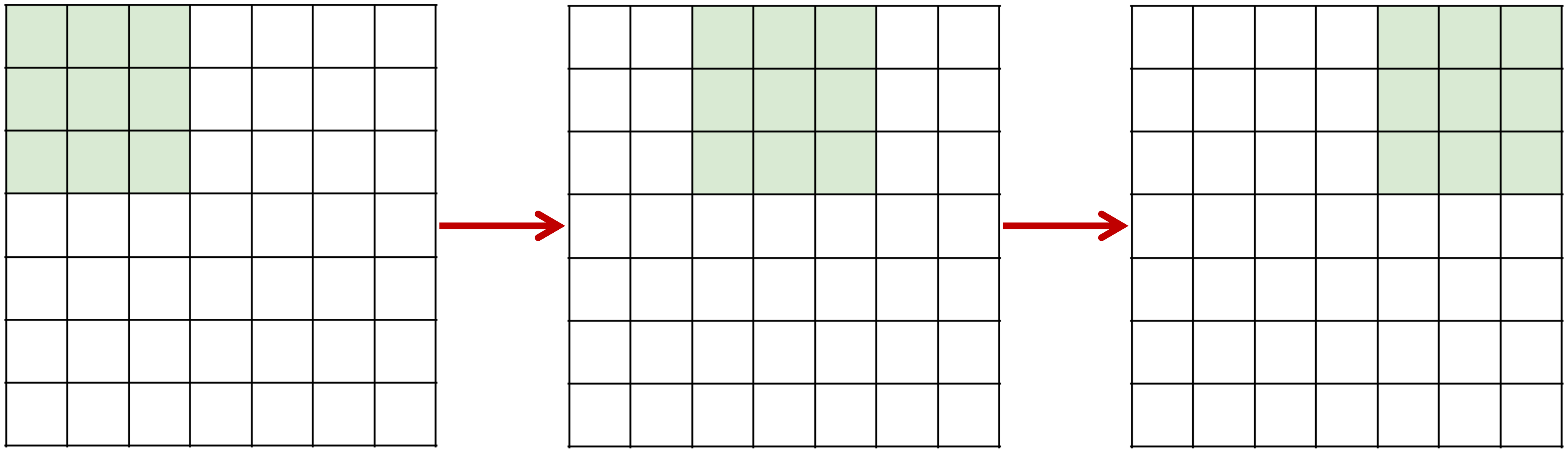
2D Convolution Layer – 4D Tensors



Slide Credit : http://eyeriss.mit.edu/tutorial.html

Options of Convolution

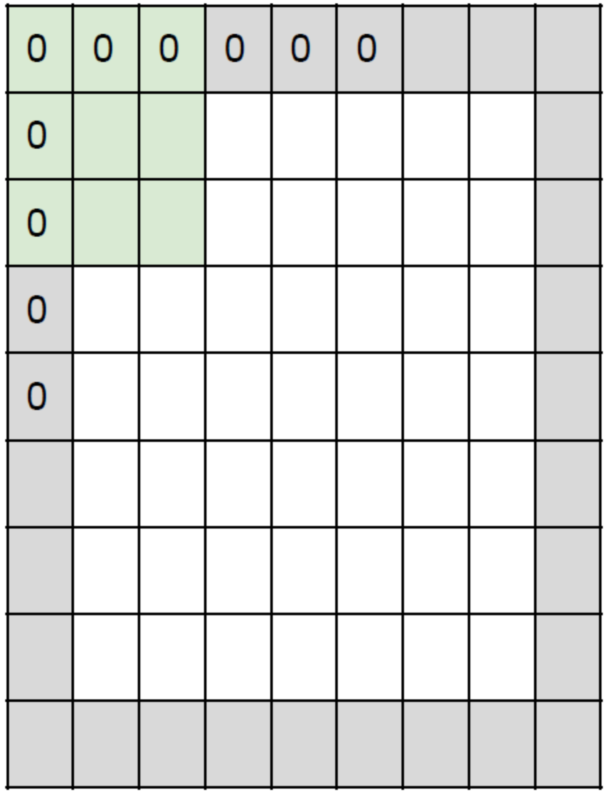
- Stride : filter가 한 번 convolution을 수행 한 후 옆으로(혹은 아래로) 얼마나 이동할 것인가
  - 예) 7x7 input, 3x3 convolution filter with stride 2 → 3x3 output!



Slide Credit : Stanford CS231n

Options of Convolution

- Zero Padding



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Slide Credit : Stanford CS231n

Quiz

- 다음의 각 경우에 convolution layer의 output size는?
  1. 32x32x3 input, 10 5x5 filters with stride 1, pad 0
  2. 32x32x3 input, 10 5x5 filters with stride 1, pad 2
  3. 32x32x3 input, 10 3x3 filters with stride 2, pad 1

Input이 $W_i \times H_i \times C_i$ *이고,*
$F \times F$ *filter를* K*개 사용하고,*
stride는 S*,*
zero padding은 P *만큼 했을 경우,*
output feature map size($W_o \times H_o \times C_o$*)는,*

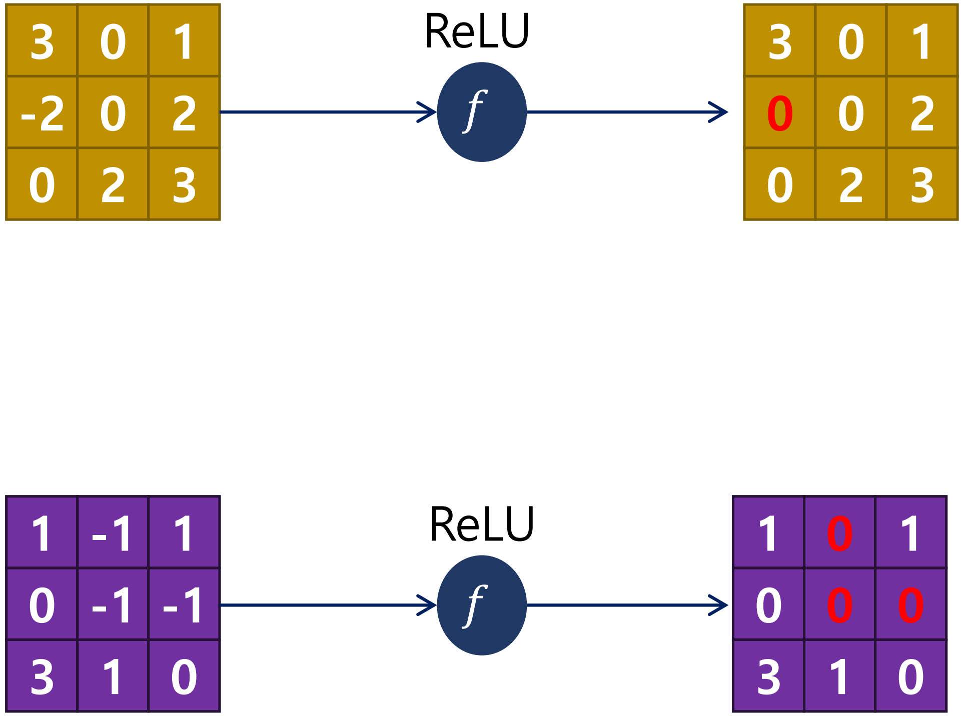
$$W_o = \frac{(W_i - F + 2P)}{S} + 1$$

$$H_o = \frac{(H_i - F + 2P)}{S} + 1$$

$$C_o = K$$

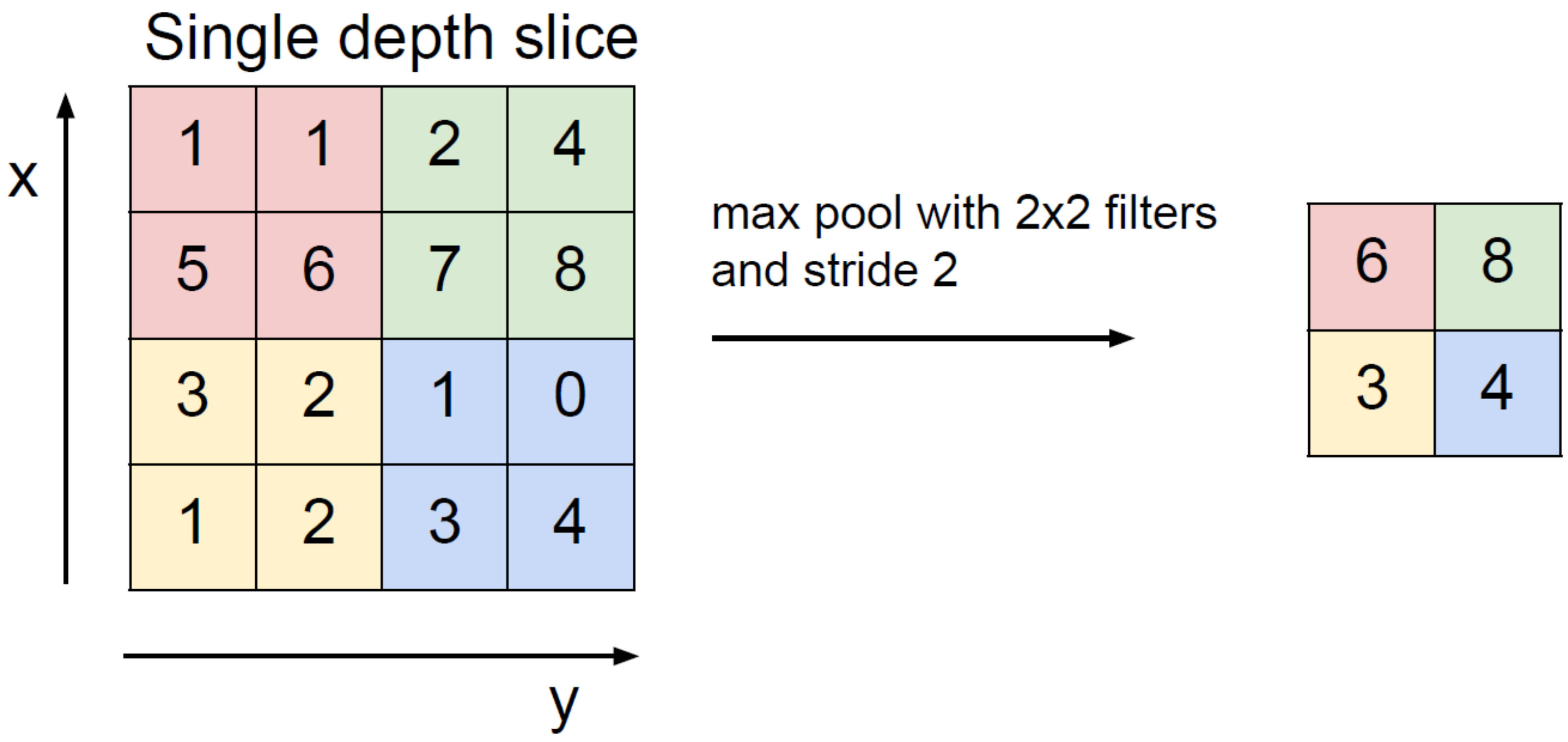
- Answer
  1. 28x28x10
  2. 32x32x10
  3. 16x16x10

ReLU



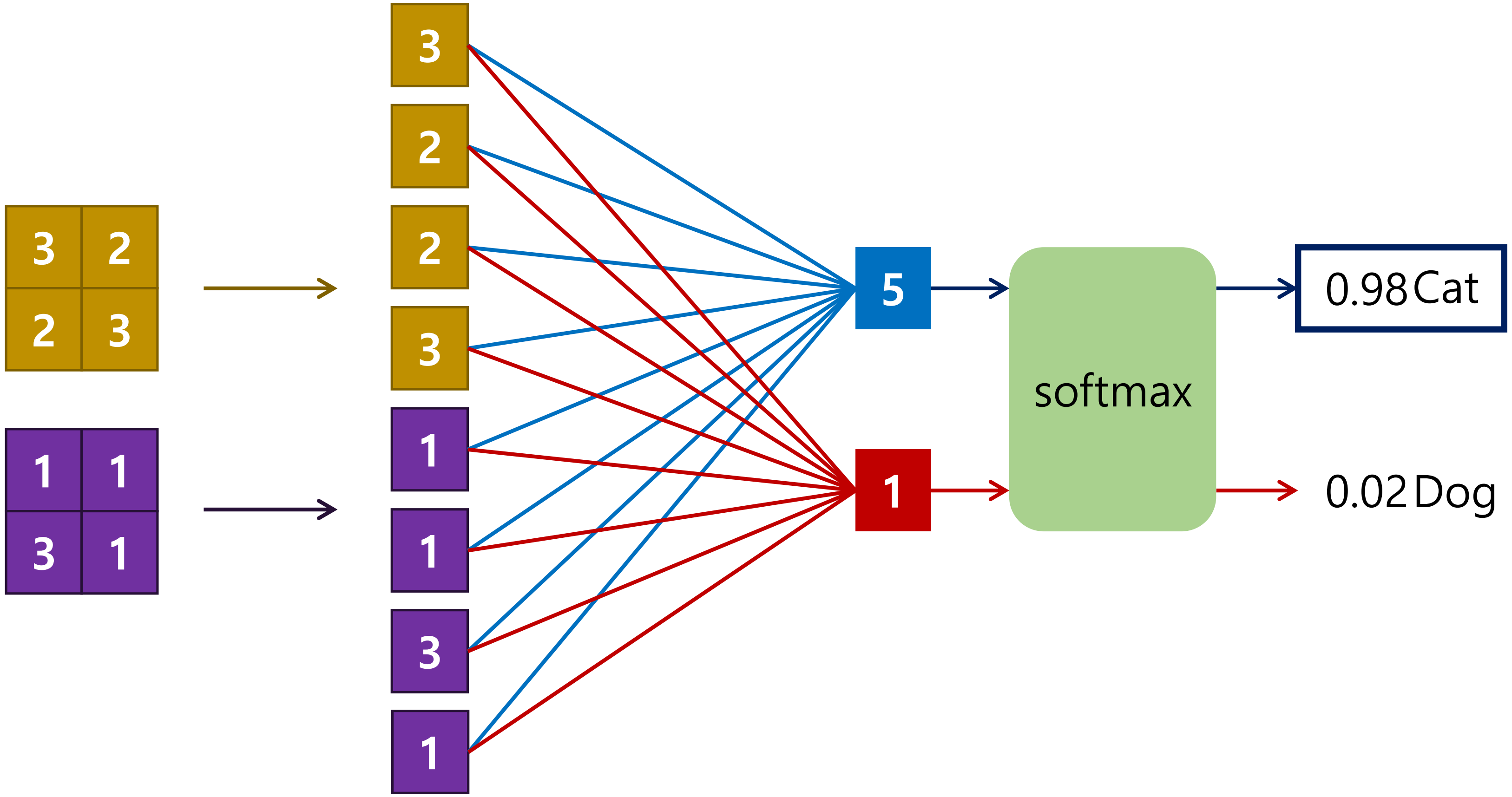
Pooling Layer

- Max pooling or Average Pooling

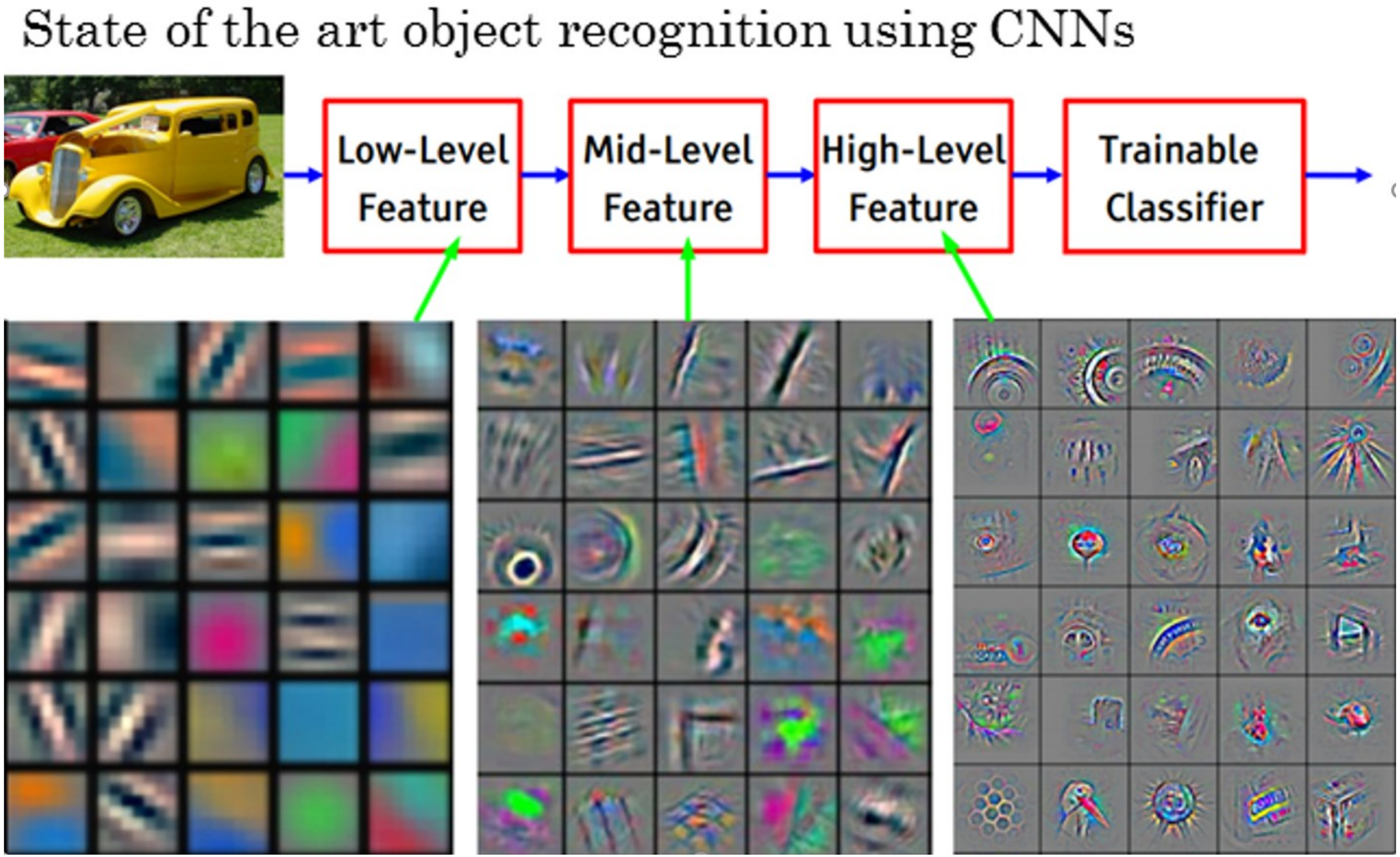


2x2 Max Pooling with Stride=1

Fully-Connected Layer



Convolutional Neural Network



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

고전적인 CNN의 특징

- Convolution Layer – parameter(weight) sharing
- Good for local invariance – pooling
- 연산량은 Convolution layer가 대부분을 차지
- Parameter 수는 FC layer가 대부분을 차지

Model	Params (M)	Conv (%)	FC (%)	Ops (M)	Conv (%)	FC (%)
AlexNet	61	3.8	96.2	725	91.9	8.1
VGG-F	99	2.2	97.8	762	87.4	12.6
VGG-M	103	6.3	93.7	1678	94.3	5.7
VGG-S	103	6.3	93.7	2640	96.3	3.7
VGG-16	138	10.6	89.4	15484	99.2	0.8
VGG-19	144	13.9	86.1	19647	99.4	0.6
NIN	7.6	100	0	1168	100.0	0.0
GoogLeNet	6.9	85.1	14.9	1566	99.9	0.1