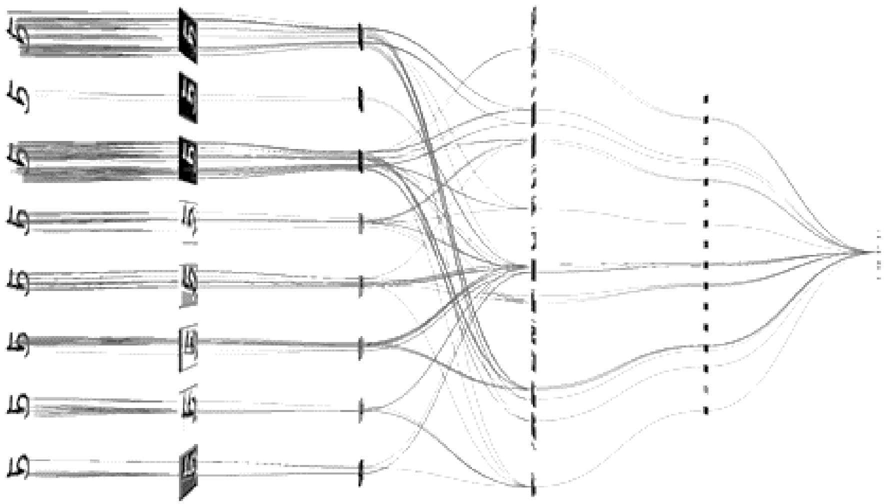
Convolutional Neural Network



Fast Campus
Start Deep Learning with Tensorflow

Image Classification

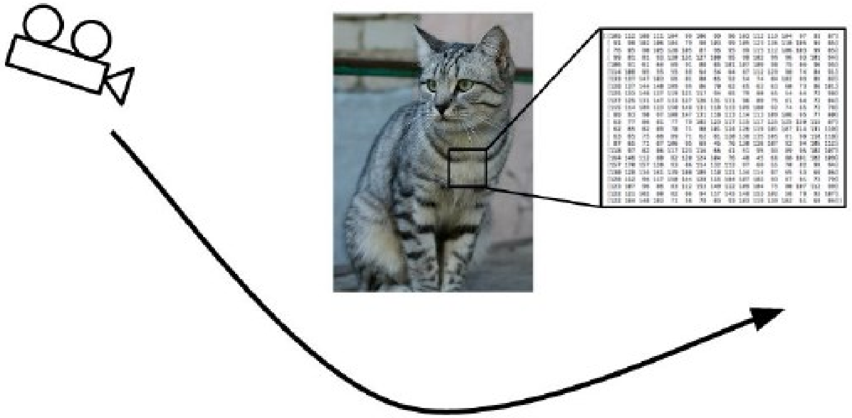
- A core task in Computer Vision

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

→ cat

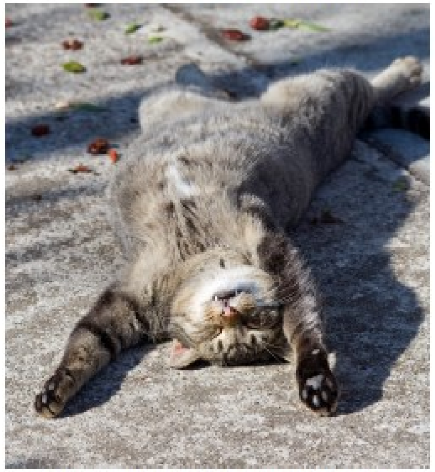
Challenges of Recognition

Viewpoint



Illumination

Deformation



Occlusion

Clutter

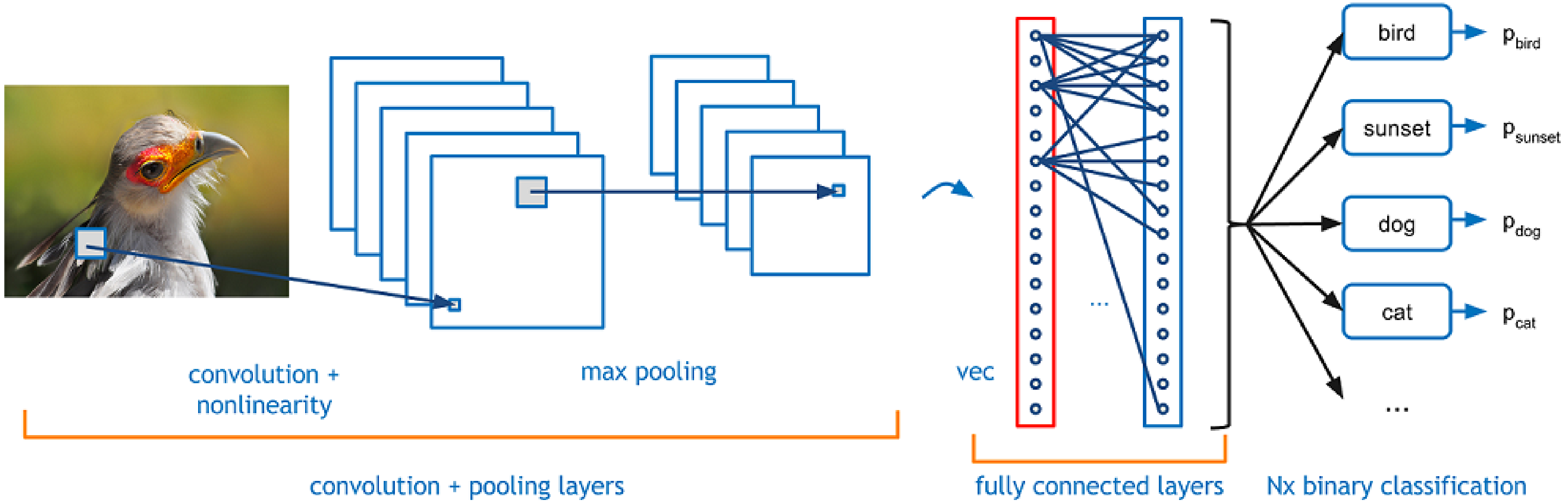
Intraclass Variation

Slide Credit : Stanford CS231n

Convolutional Neural Network

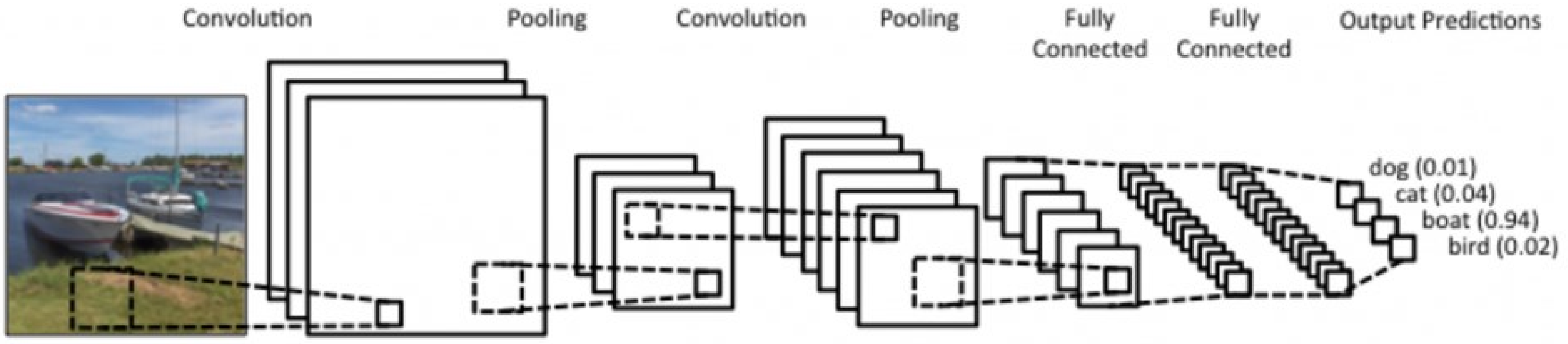
What We See

What Computers See



Convolutional Neural Network

- 이미지 인식에 가장 널리 사용됨
- 일반적으로 convolution layer, pooling layer, fully-connected layer로 구성
- Parameter(weight) sharing
- Convolution과 pooling layer는 feature를 추출하고 fully-connected layer는 어떤 class에 속하는지 판단하는 역할을 수행



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

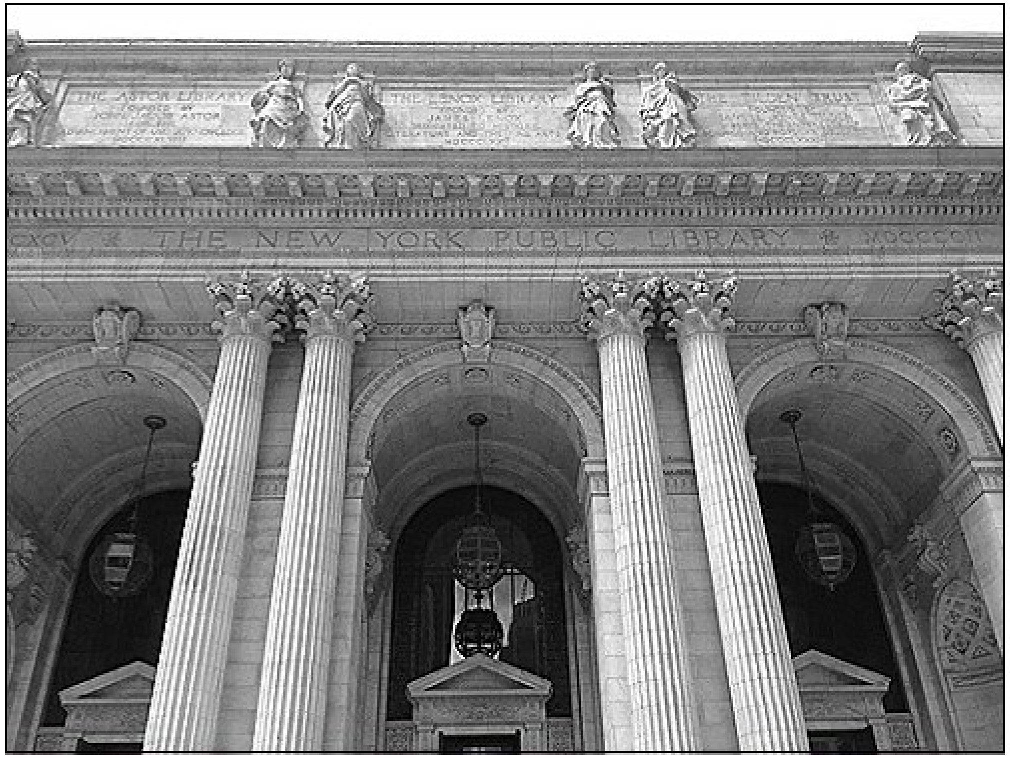
	-2	-1	0	
	-1	1	1	
	0	1	2	

Let’s Try!

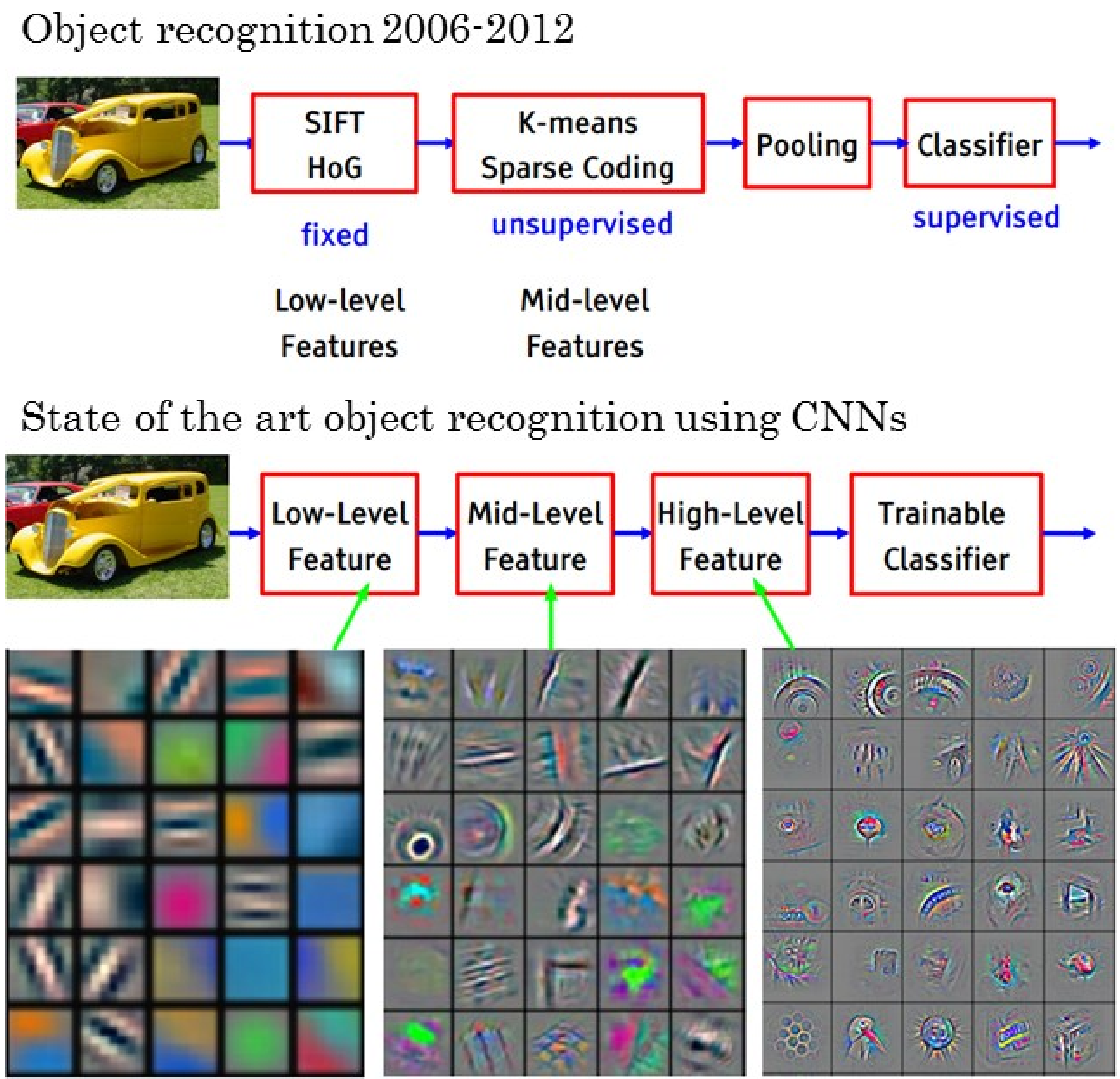
- http://setosa.io/ev/image-kernels/

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

sharpen

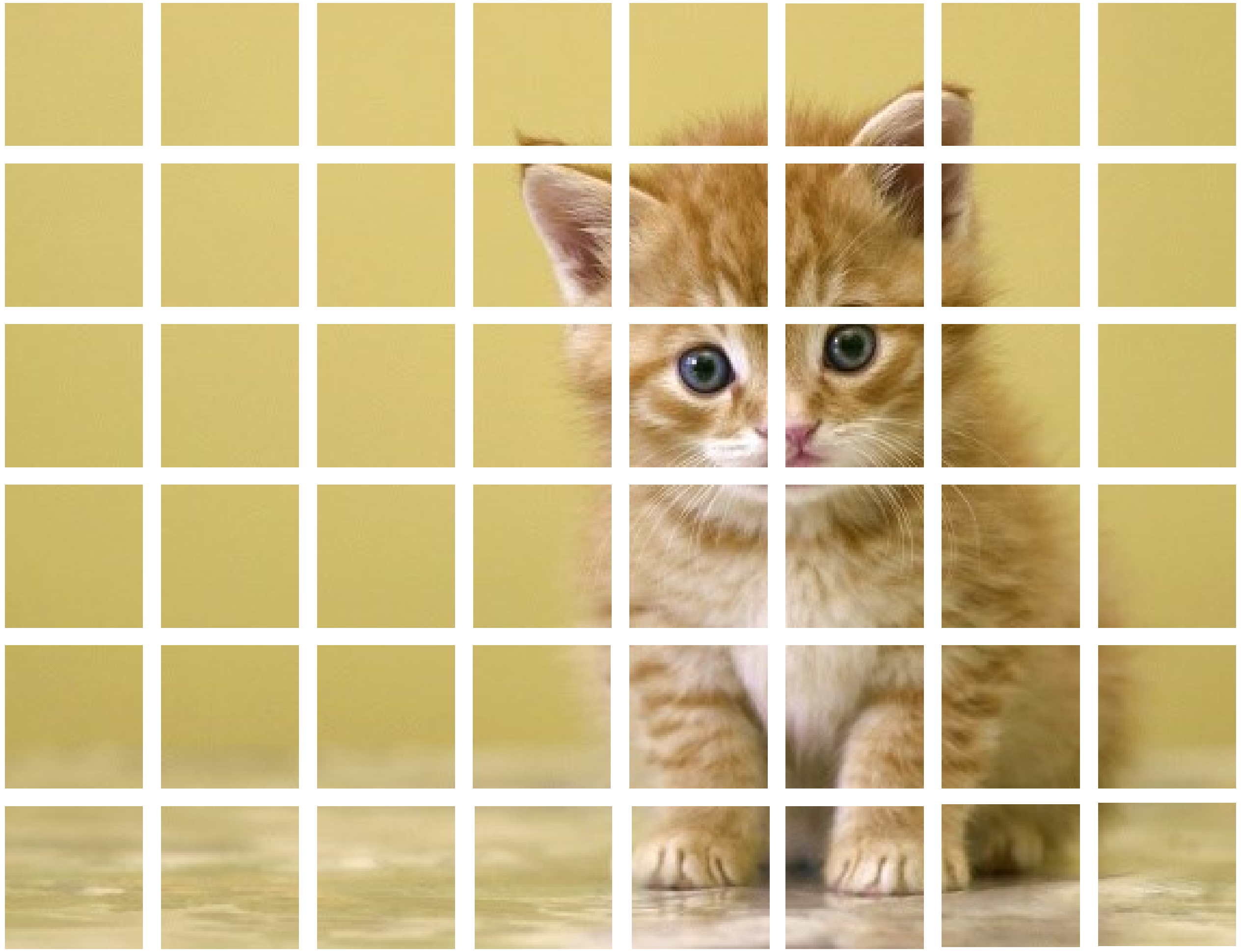


Before and After



CNN 동작원리

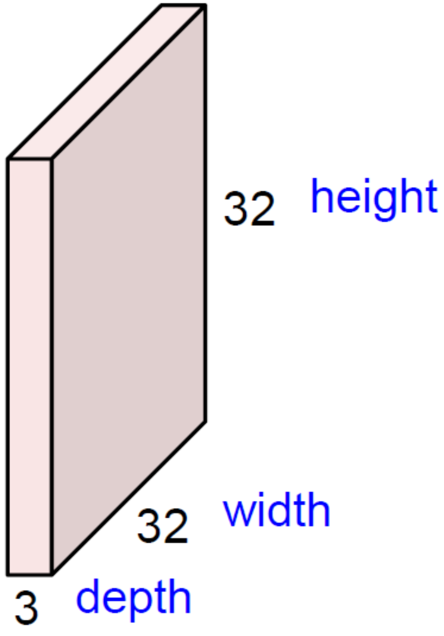
- 이미지를 작은 tile로 나누고, 작은 network를 통해 tile에서 특정 feature를 추출(예: 귀)
- Newtork가 다음 tile로 이동하면서 같은 방법으로feature를 추출(동일한 weight 사용)
- 다른 feature(예: 눈)를 추출하는 network를 추가로 만들고 위와 같은 방법으로 tile을 하나씩 network에 적용
- 추출된 모든 feature들을 잘 조합하여 최종적으로 이미지를 판단



Convolution Layer
- From Stanford CS231n

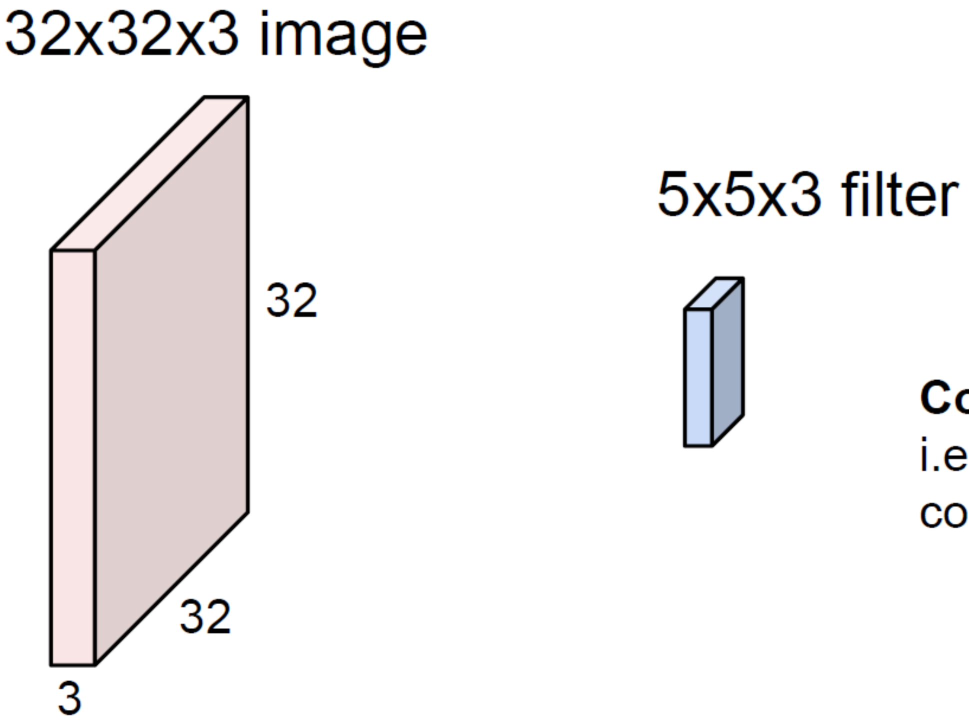
Convolution Layer

32x32x3 image



Convolution Layer
- From Stanford CS231n

Convolution Layer



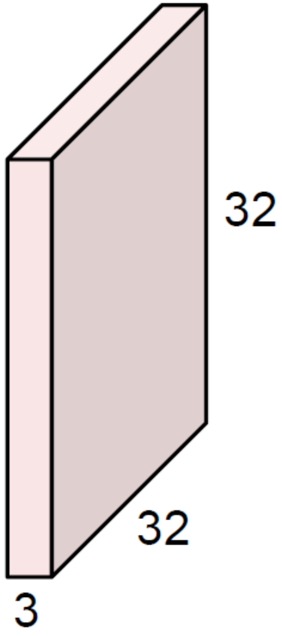
Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Convolution Layer
- From Stanford CS231n

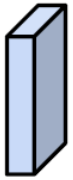
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



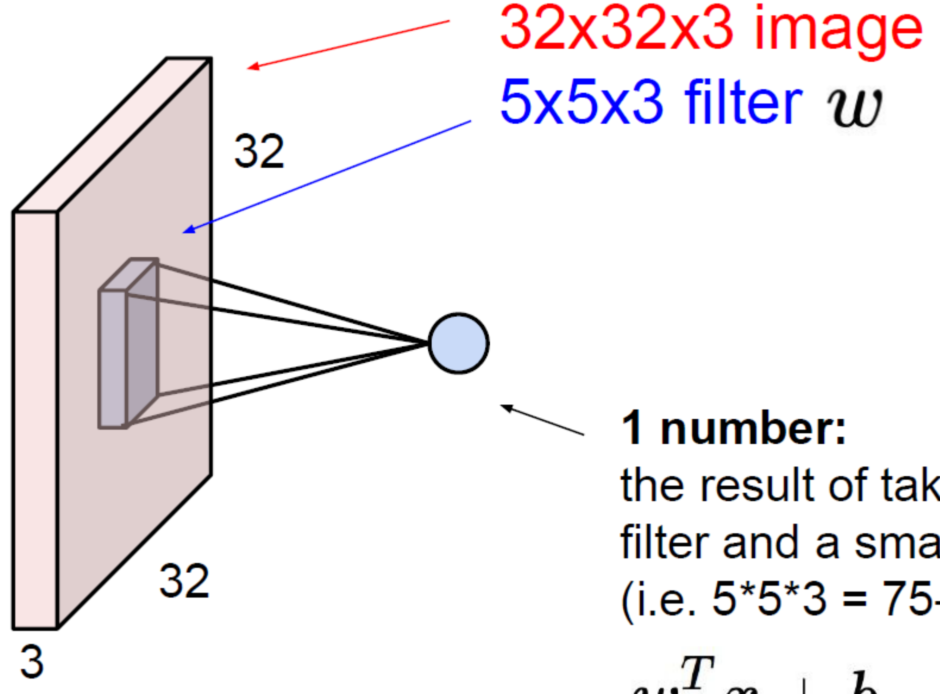
5x5x3 filter



Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

Convolution Layer
- From Stanford CS231n

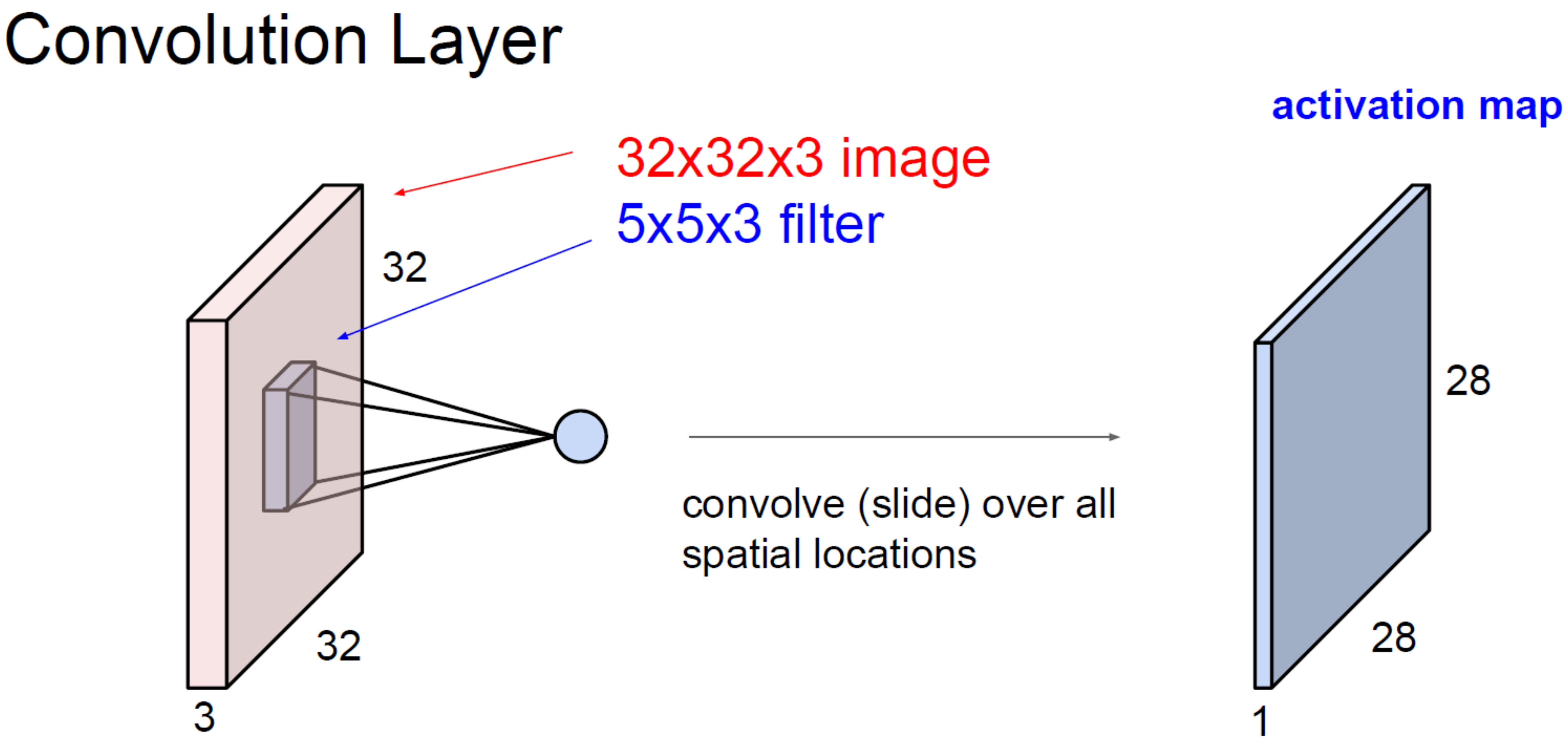
Convolution Layer



1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

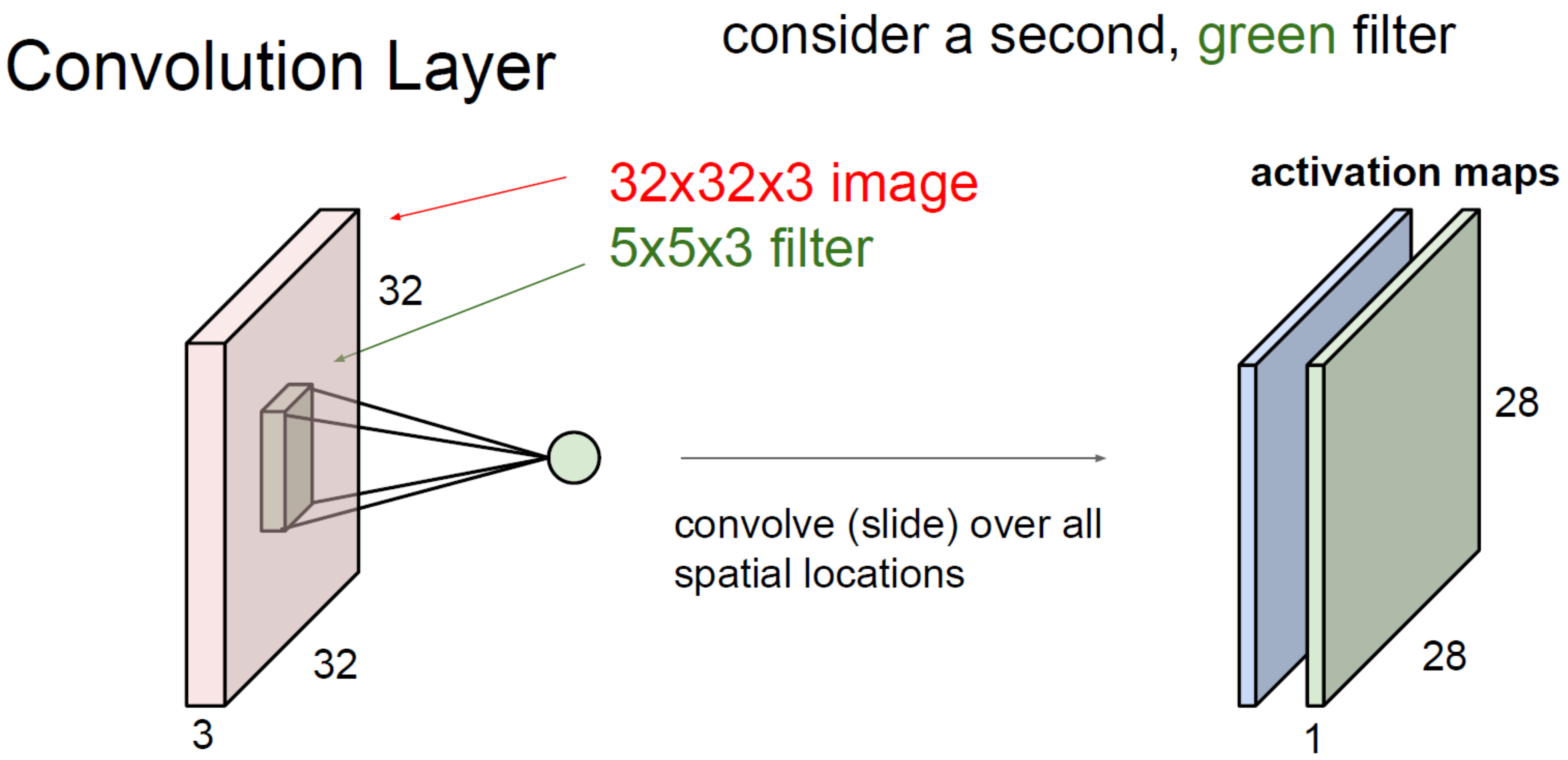
$$w^T x + b$$

Convolution Layer
- From Stanford CS231n



Slide Credit : Stanford CS231n

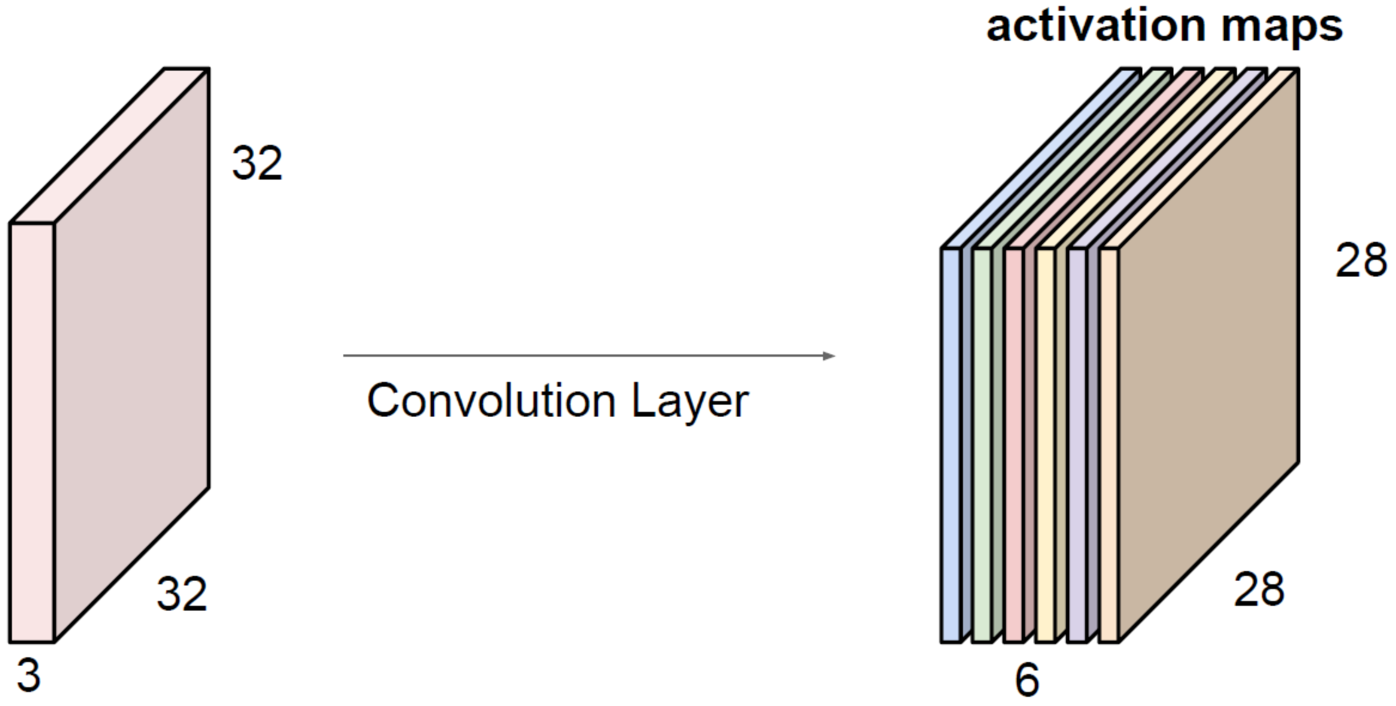
Convolution Layer
- From Stanford CS231n



Slide Credit : Stanford CS231n

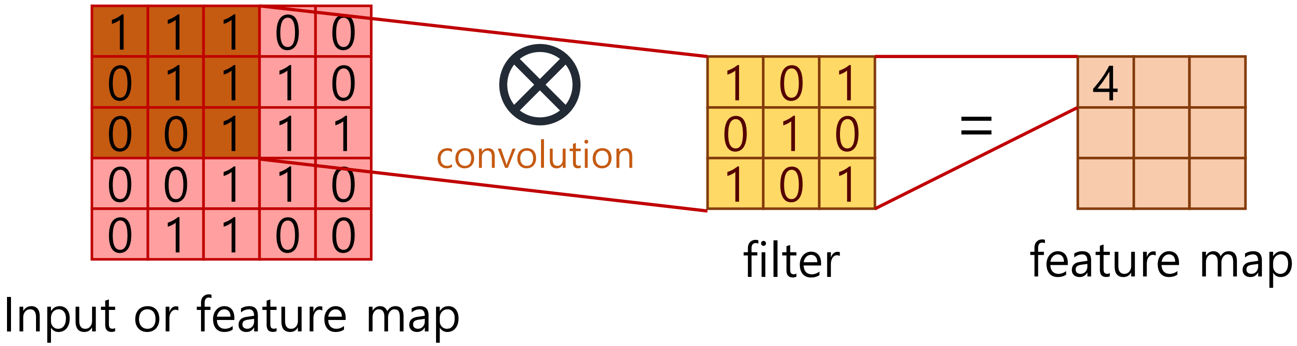
Convolution Layer
- From Stanford CS231n

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

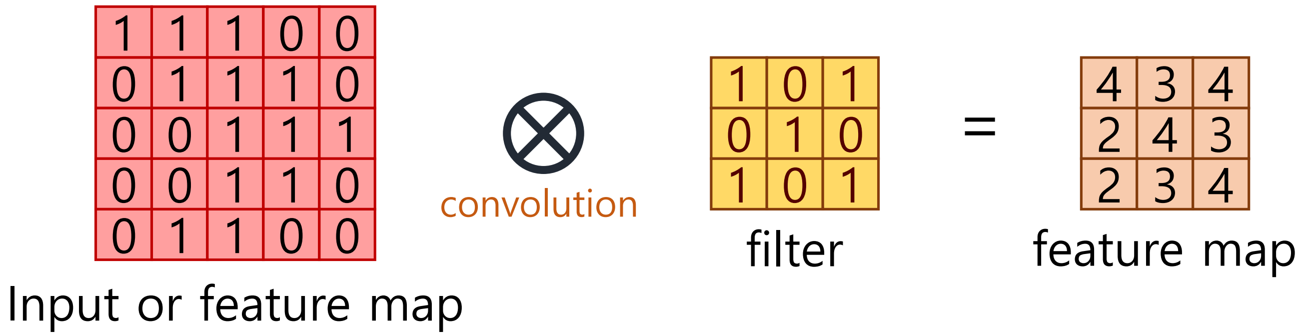


We stack these up to get a "new image" of size 28x28x6!

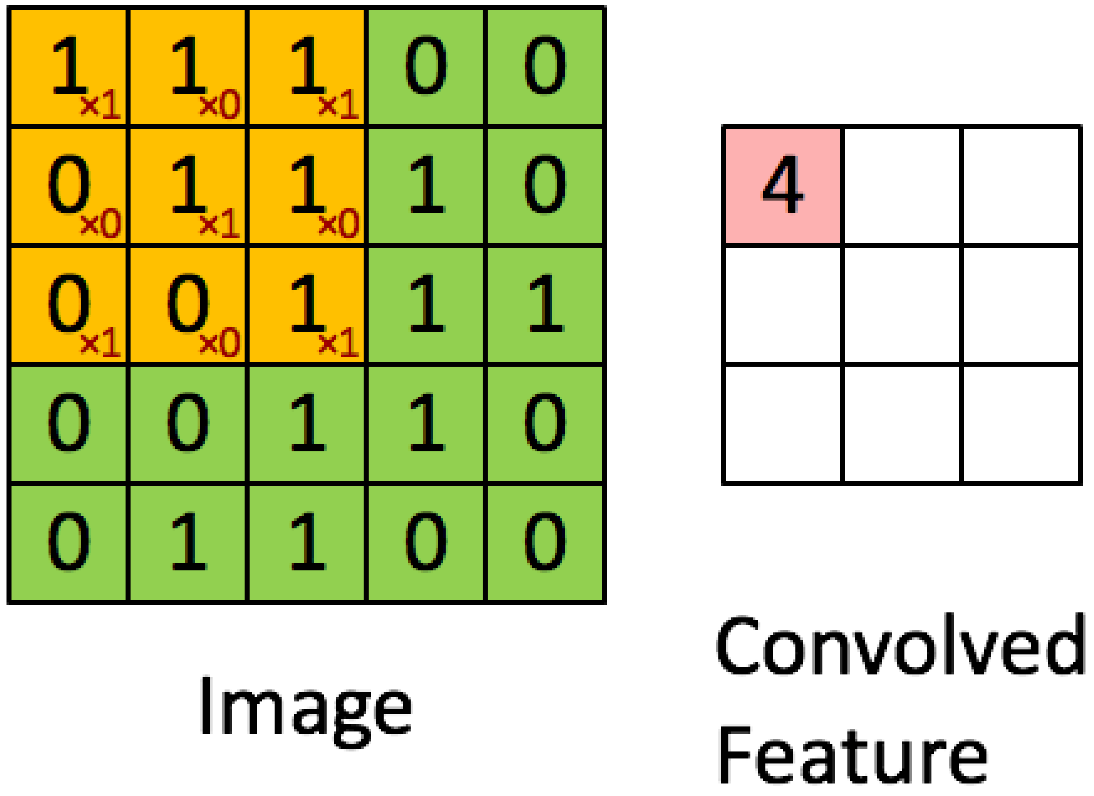
Convolution



- Convolution 연산 : 같은 위치에 있는 숫자끼리 곱한 후 모두 더함
  - 1X1 + 1X0 + 1X1 + 0X0 + 1X1 + 1X0 + 0X1 + 0X0 + 1X1 = 4
- Filter가 옆으로 이동 후 같은 연산 수행
- 옆으로 모두 이동한 이후에는 아래로 이동 후 같은 연산 수행



Convolution

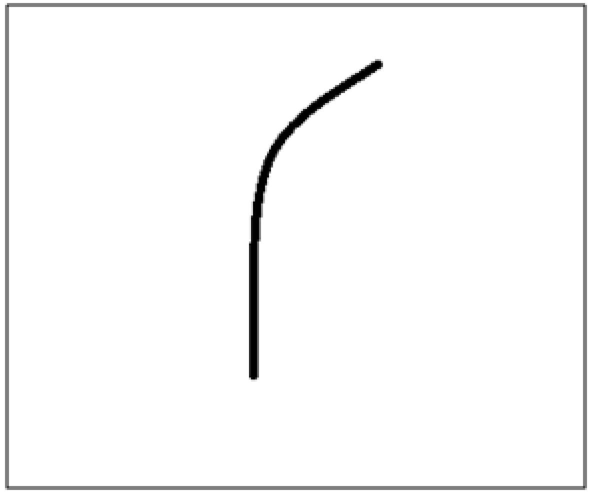


Credit : Leonardo's gitbook

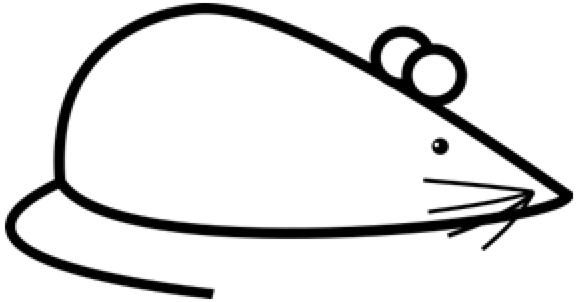
Feature Extractor

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

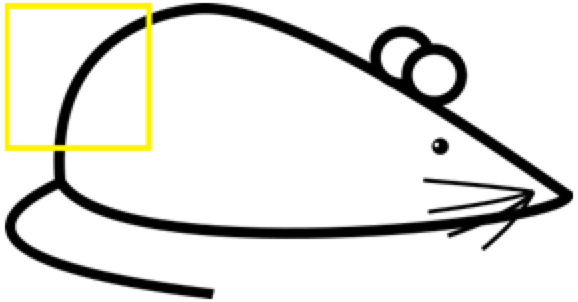
Pixel representation of filter



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

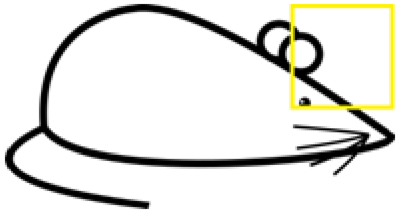
Pixel representation of the receptive field

*

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

*

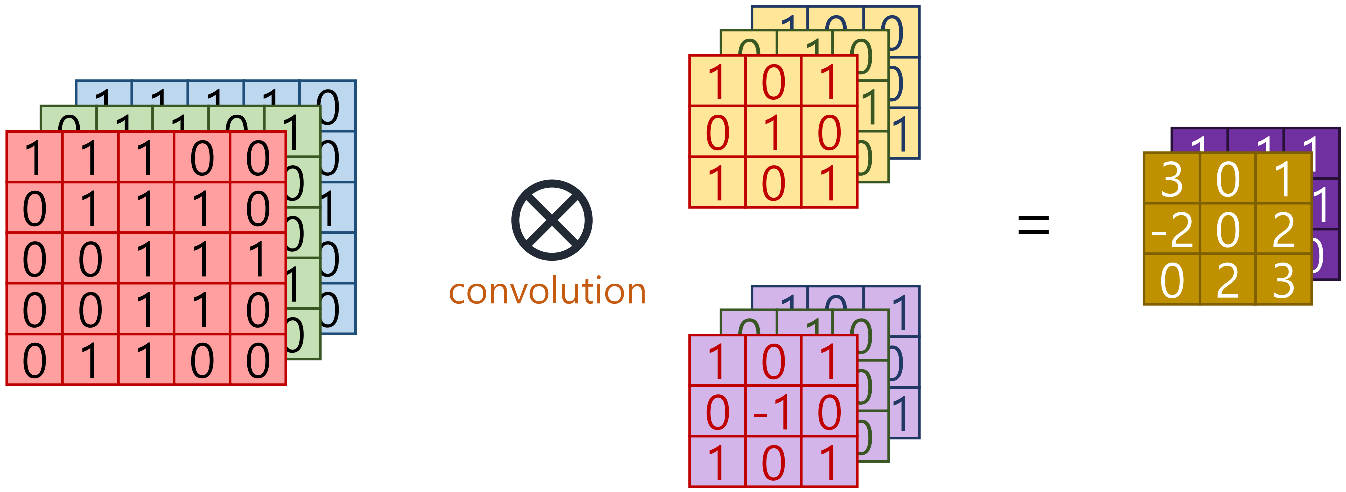
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0

Credit : Adit Deshpande's blog

Convolution
(Multi Channel, Many Filters)

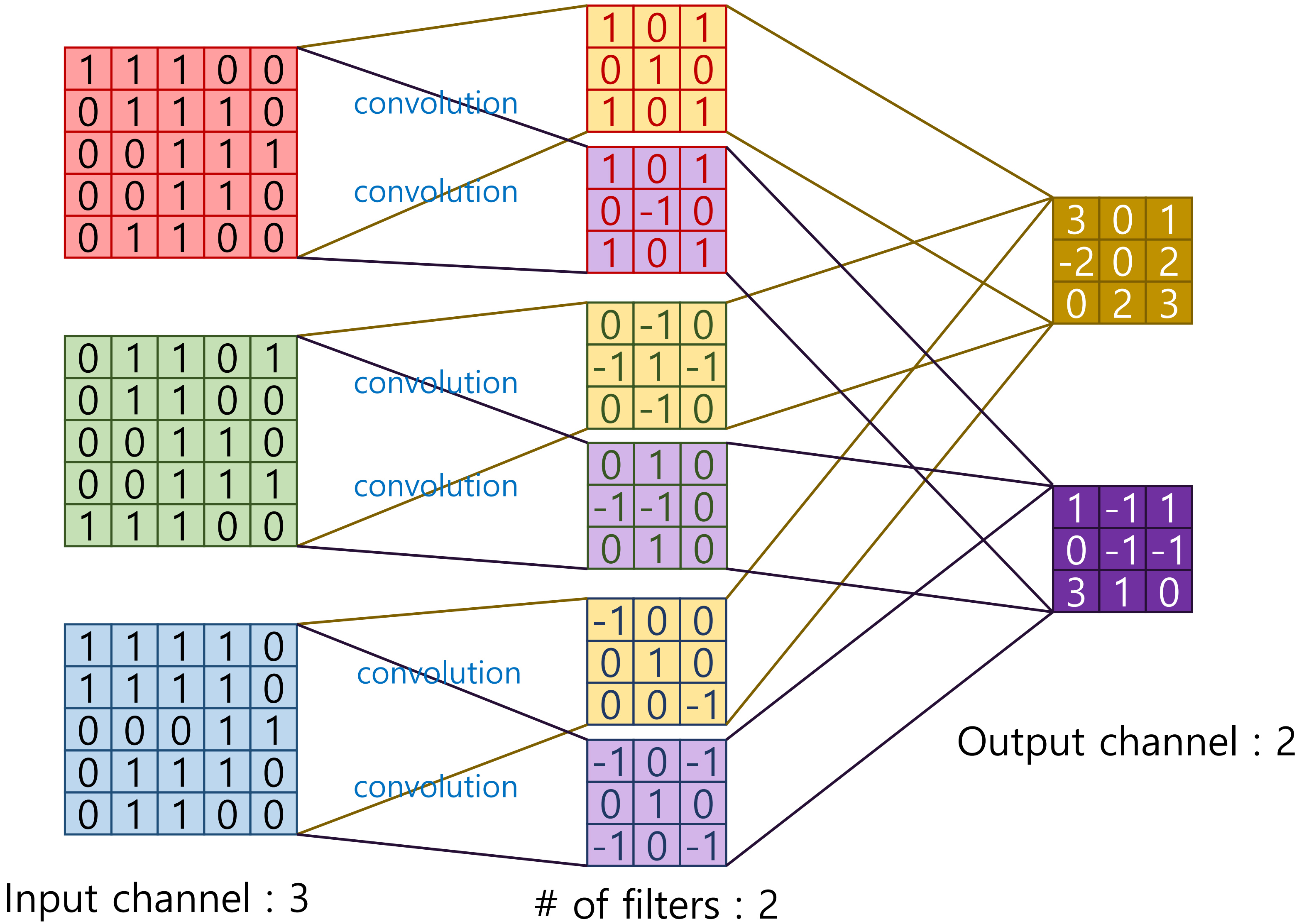


Input channel : 3

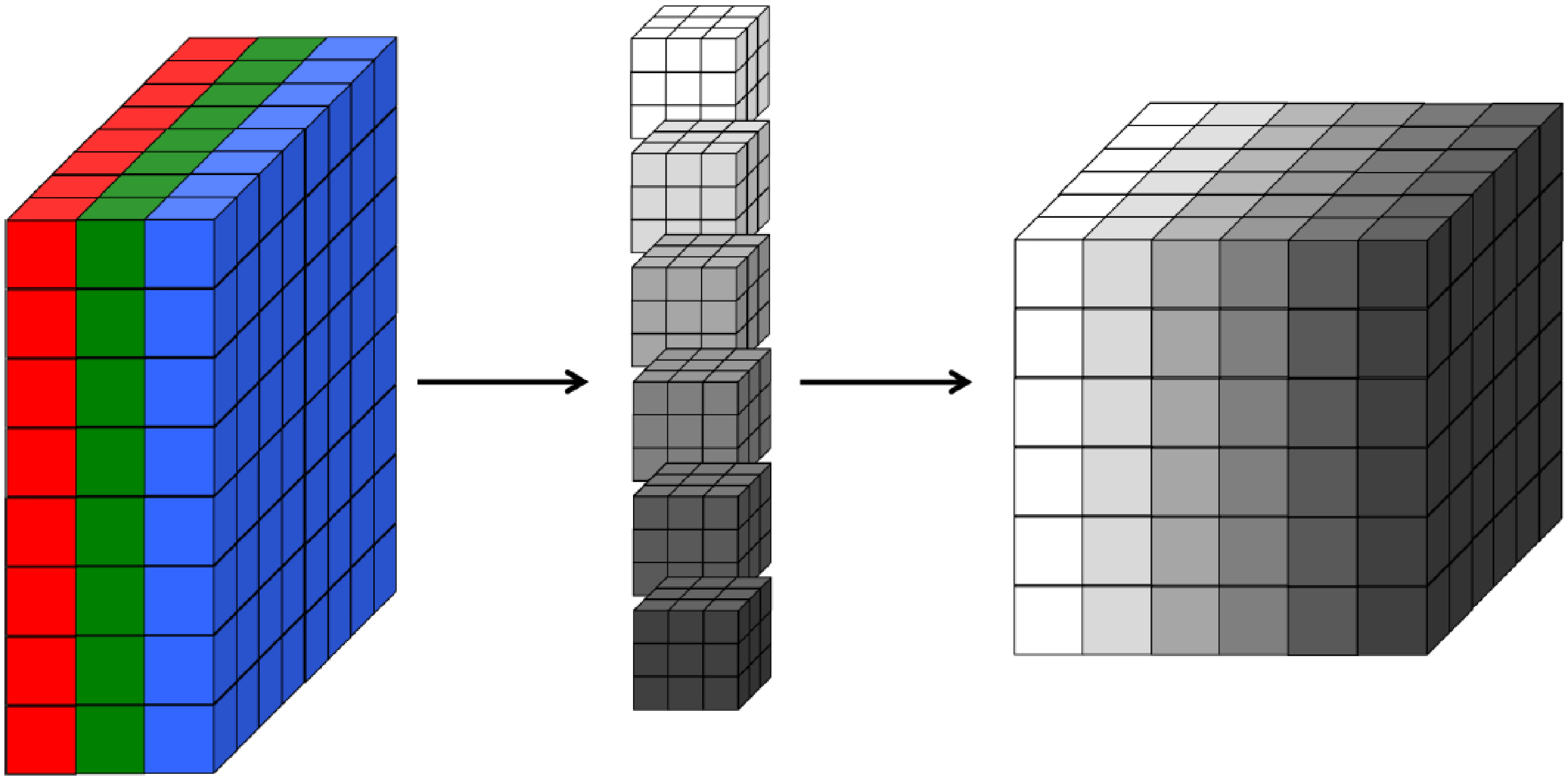
# of filters : 2

Output channel : 2

Convolution
(Multi Channel, Many Filters)



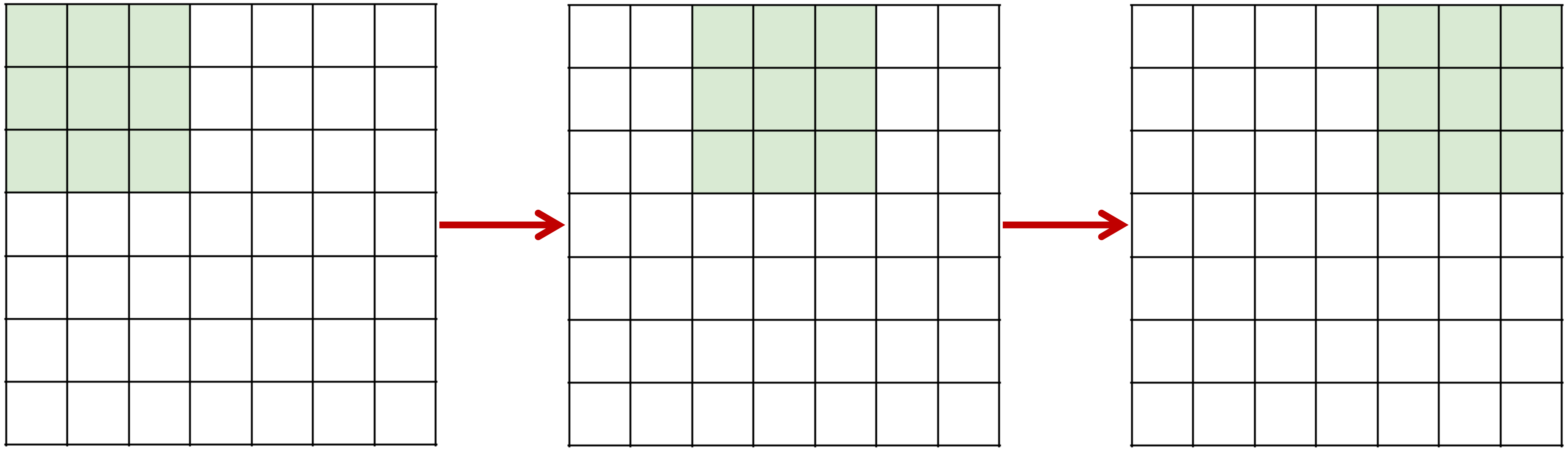
Visualization of a Convolution Layer



Picture Credit : fundamentals of deep learning by Nikhil

Options of Convolution

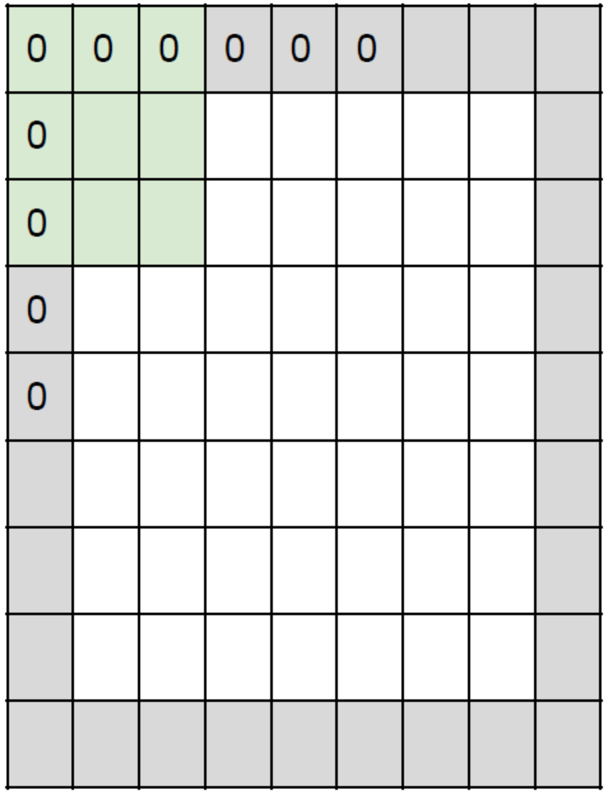
- Stride : filter가 한 번 convolution을 수행 한 후 옆으로(혹은 아래로) 얼마나 이동할 것인가
  - 예) 7x7 input, 3x3 convolution filter with stride 2 → 3x3 output!



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

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

tf.nn.conv2d

- Signature:

```
tf.nn.conv2d(input, filter, strides, padding,
use_cudnn_on_gpu=None, data_format=None, name=None)
```

- Docstring:

Computes a 2-D convolution given 4-D `input` and `filter` tensors.

Given an input tensor of shape

`[batch, in_height, in_width, in_channels]`

and a filter / kernel tensor of shape

`[filter_height, filter_width, in_channels, out_channels]’

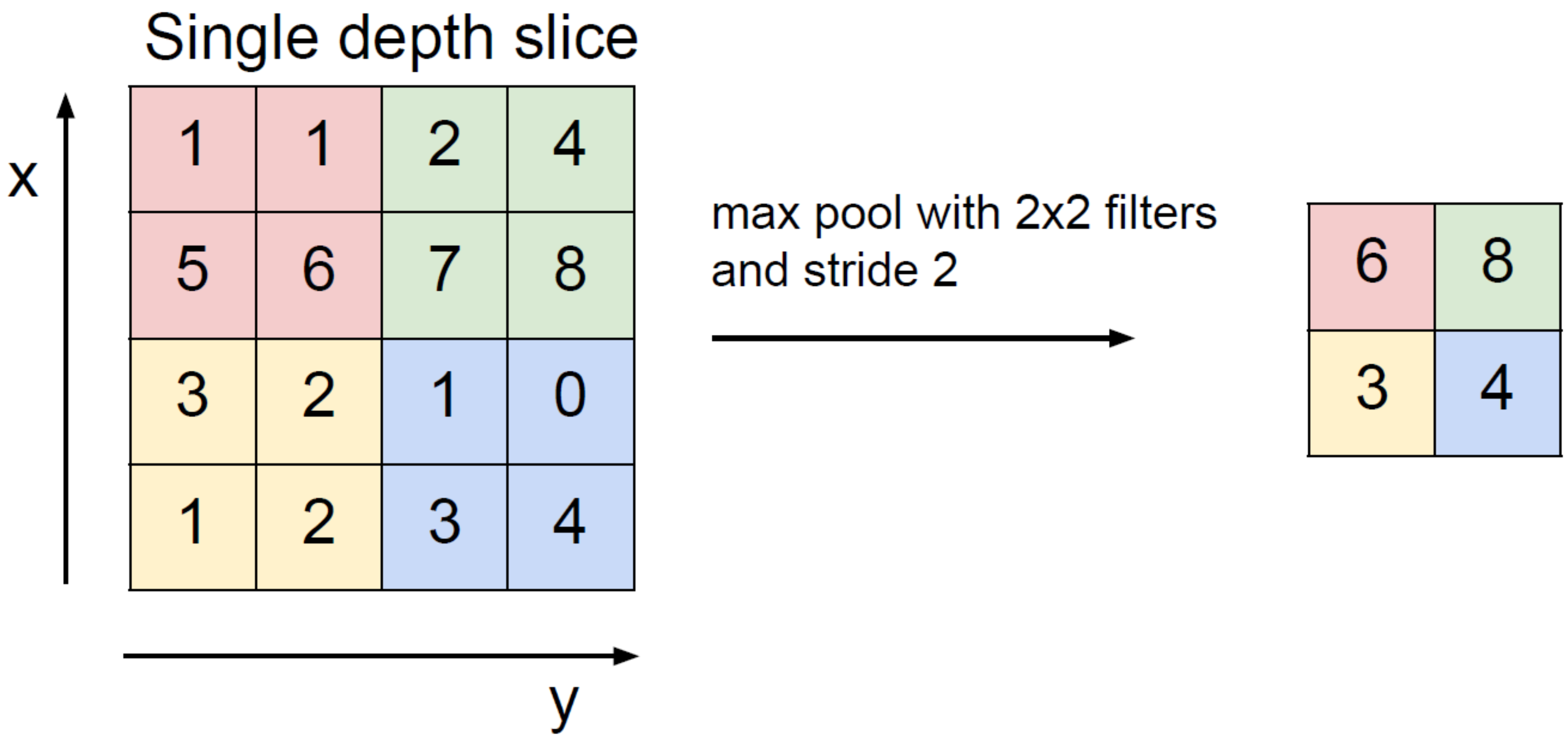
ReLU

$$\begin{bmatrix} 3 & 0 & 1 \\ -2 & 0 & 2 \\ 0 & 2 & 3 \end{bmatrix} \xrightarrow{\text{ReLU}} f \rightarrow \begin{bmatrix} 3 & 0 & 1 \\ 0 & 0 & 2 \\ 0 & 2 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -1 & 1 \\ 0 & -1 & -1 \\ 3 & 1 & 0 \end{bmatrix} \xrightarrow{\text{ReLU}} f \rightarrow \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 3 & 1 & 0 \end{bmatrix}$$

Pooling Layer

• Max pooling을 많이 사용함



2x2 Max Pooling with Stride=1

$$\begin{bmatrix} 3 & 0 & 1 \\ 0 & 0 & 2 \\ 0 & 2 & 3 \end{bmatrix} \xrightarrow{\text{max pooling}} \begin{bmatrix} 3 & 2 \\ 2 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 3 & 1 & 0 \end{bmatrix} \xrightarrow{\text{max pooling}} \begin{bmatrix} 1 & 1 \\ 3 & 1 \end{bmatrix}$$

tf.nn.max_pool

- Signature:

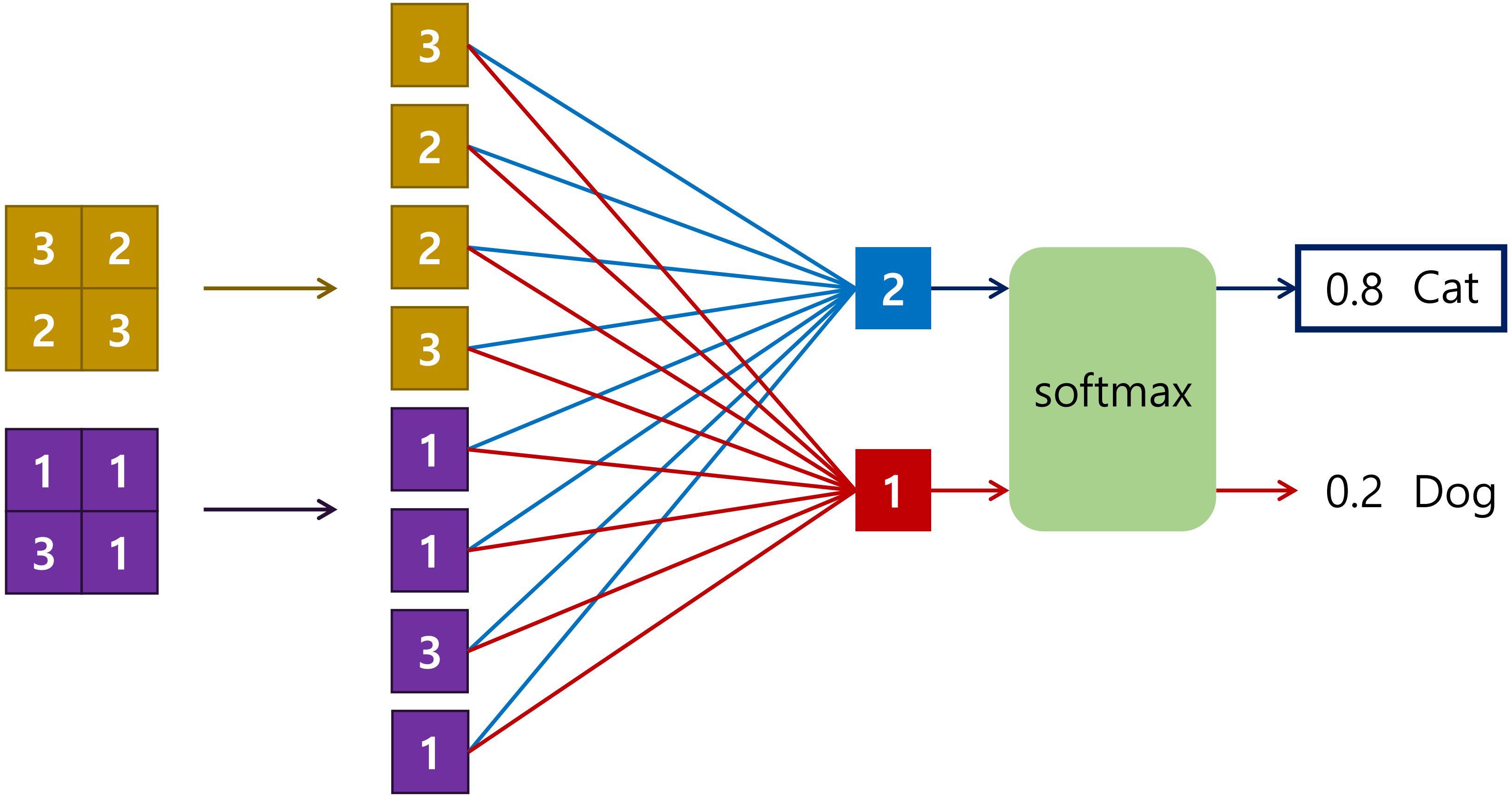
```
tf.nn.max_pool(value, ksize, strides, padding, data_format='NHWC', name=None)
```

- Docstring:

Performs the max pooling on the input.

- Args:
  - value: A 4-D `Tensor` with shape `[batch, height, width, channels]` and type `tf.float32`.
  - ksize: A list of ints that has length >= 4. The size of the window for each dimension of the input tensor.
  - strides: A list of ints that has length >= 4. The stride of the sliding window for each dimension of the input tensor.
  - padding: A string, either `'VALID'` or `'SAME'`. The padding algorithm.
  - data_format: A string. 'NHWC' and 'NCHW' are supported.
  - name: Optional name for the operation.

Fully-Connected Layer



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

CNN 사용을 위해 결정할 것들

- Input image size
- Layer 수 (convolution, fully connected 각각)
- Layer별 filter size
- Layer별 filter 수
- Batch size
- Optimizer
- Learning rate
- Regularization method 등등...