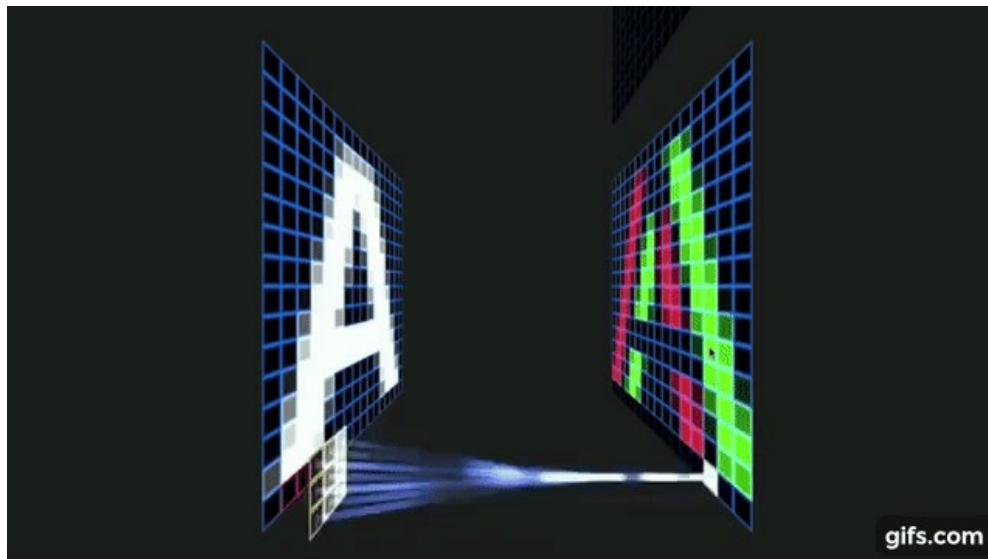


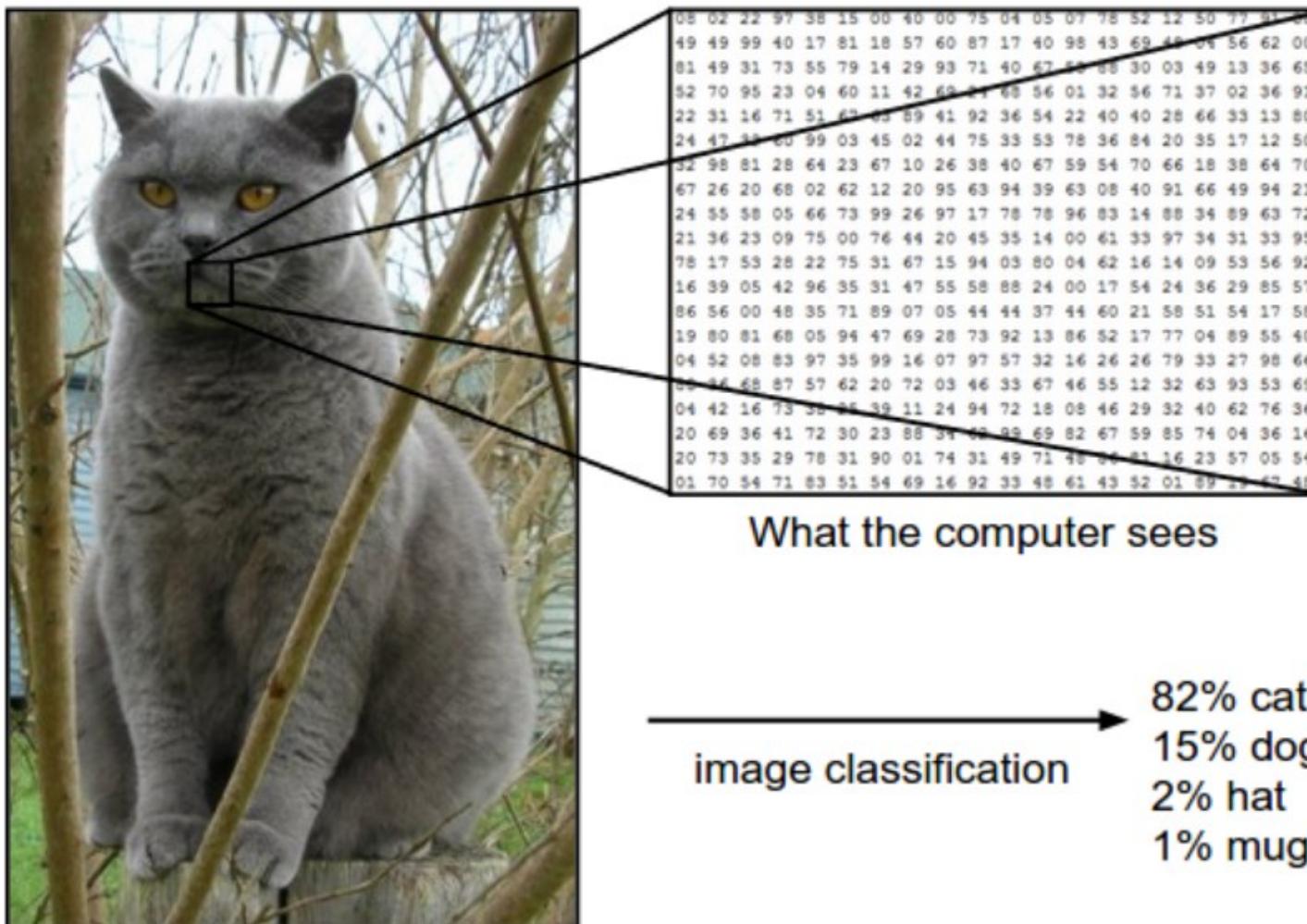
# Convolutional Neural Network



Fast Campus  
Start Deep Learning with TensorFlow

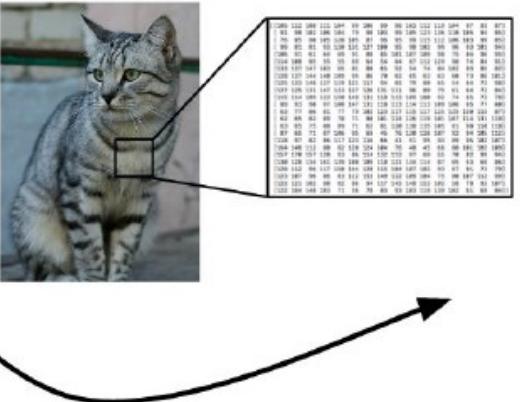
# Image Classification

- A core task in Computer Vision



# Challenges of Recognition

Viewpoint

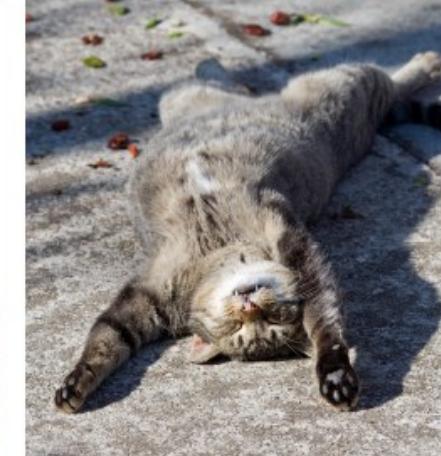


Illumination



[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](#)

# An Image Classifier

```
def classify_image(image):  
    # Some magic here?  
    return class_label
```

Unlike e.g. sorting a list of numbers,  
**no obvious way** to hard-code the algorithm for  
recognizing a cat, or other classes.

# Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

Example training set

```
def train(images, labels):  
    # Machine learning!  
    return model
```

```
def predict(model, test_images):  
    # Use model to predict labels  
    return test_labels
```

**airplane**



**automobile**



**bird**



**cat**



**deer**



# First Classifier – Nearest Neighbor

```
def train(images, labels):  
    # Machine learning!  
    return model
```



Memorize all data and labels

```
def predict(model, test_images):  
    # Use model to predict labels  
    return test_labels
```



Predict the label of the most similar training image

# Example Dataset: CIFAR10

**10** classes

**50,000** training images

**10,000** testing images

**airplane**



**automobile**



**bird**



**cat**



**deer**



**dog**



**frog**



**horse**



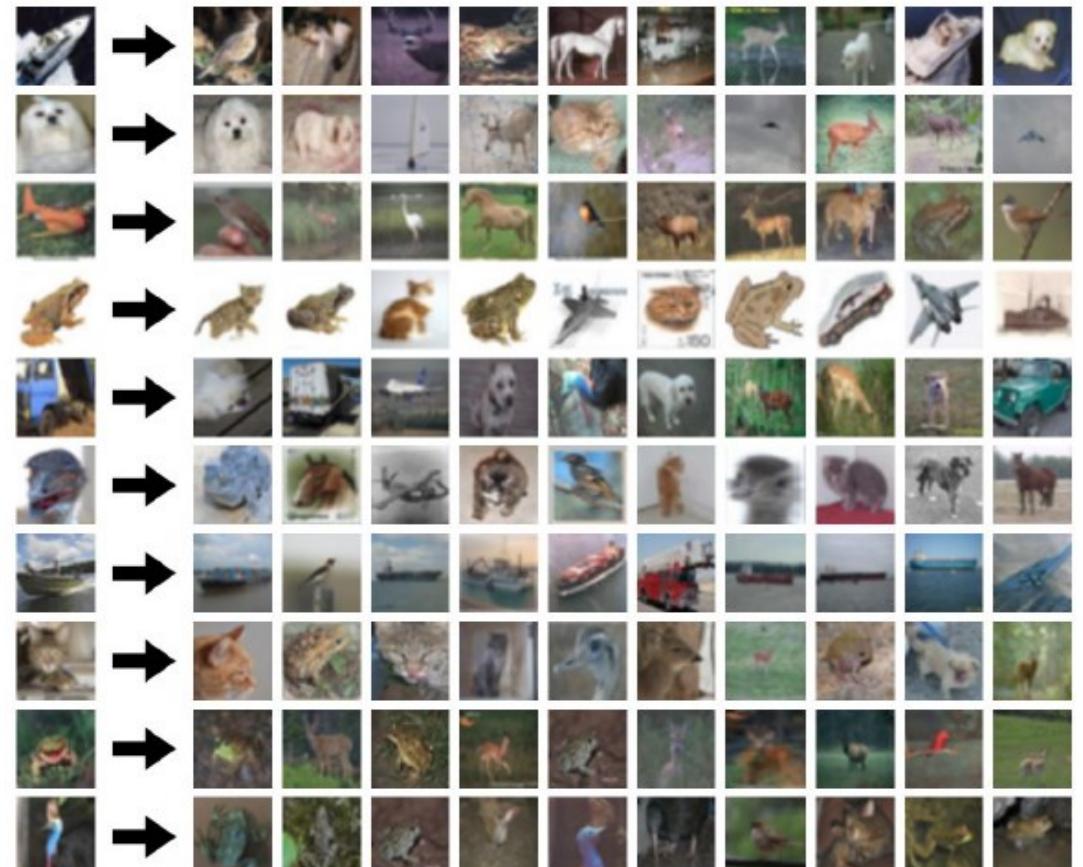
**ship**



**truck**



Test images and nearest neighbors



# Distance Metric

L1 distance:

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

test image

56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

training image

10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

-

pixel-wise absolute value differences

46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

=

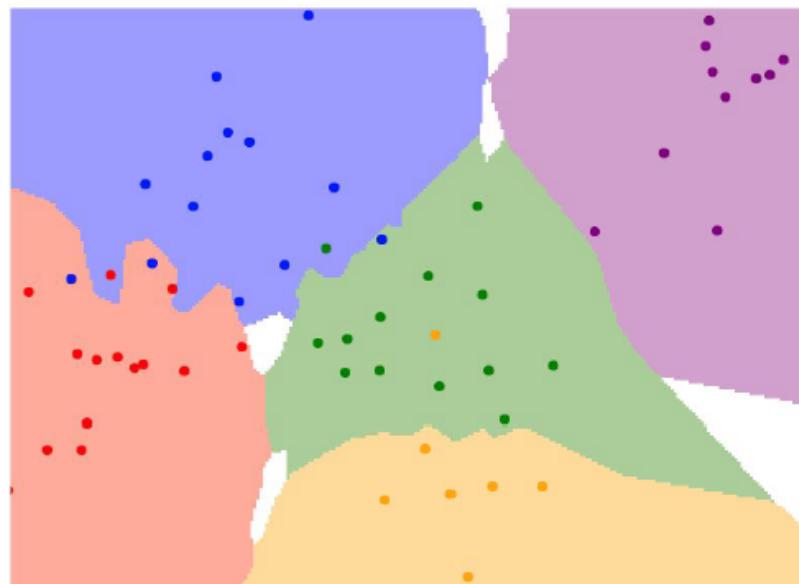
add → 456

# K-Nearest Neighbors

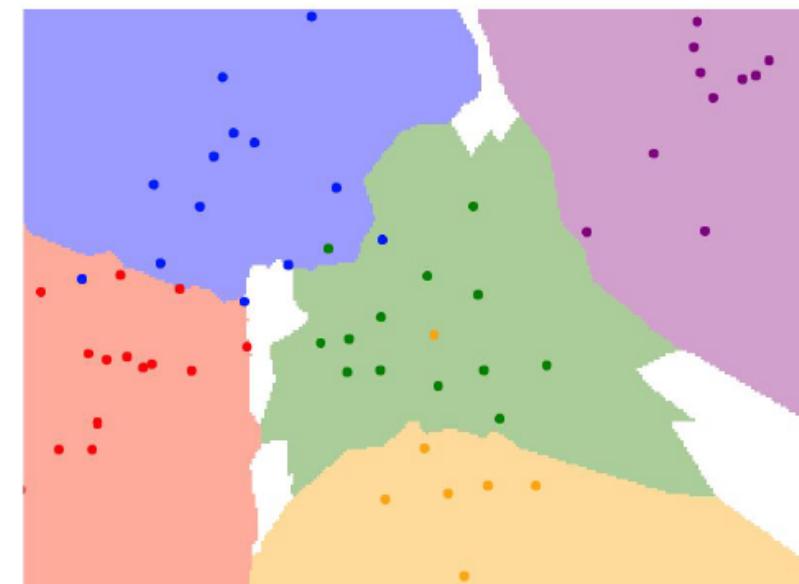
Instead of copying label from nearest neighbor,  
take **majority vote** from K closest points



$K = 1$



$K = 3$

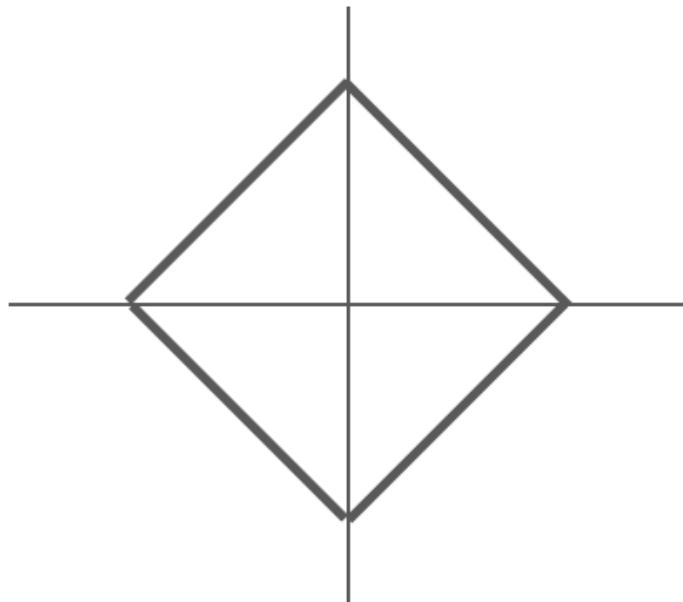


$K = 5$

# K-Nearest Neighbors: Distance Metric

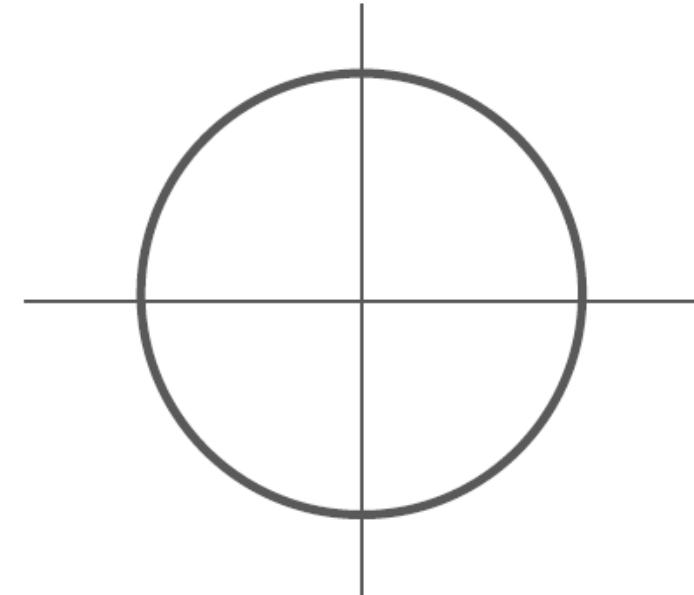
L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$



# K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

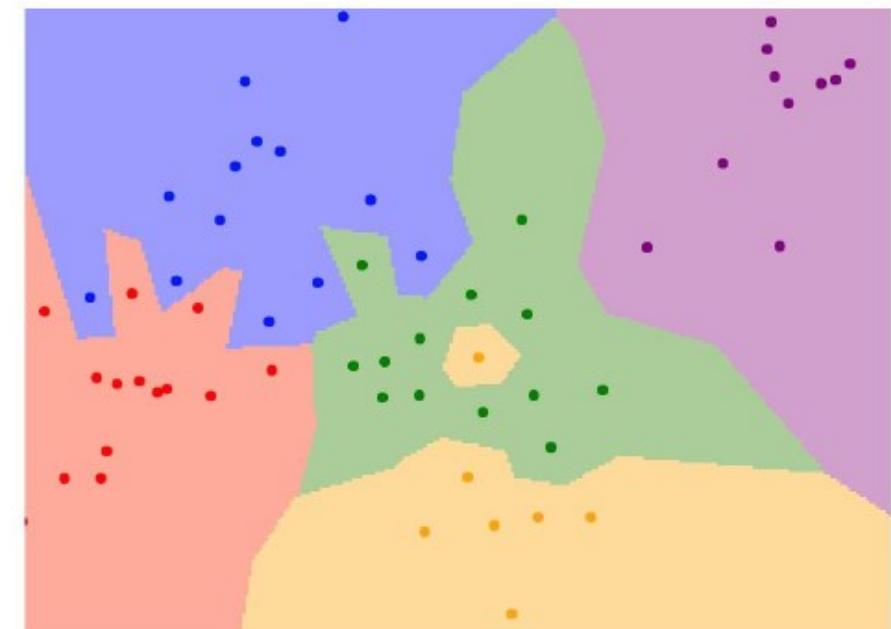


$K = 1$

<http://vision.stanford.edu/teaching/cs231n-demos/knn/>

L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$



$K = 1$

# K-Nearest Neighbor on Images NEVER Used

- Very slow at test time
- Distance metrics on pixels are not informative

Original



Boxed



Shifted



Tinted

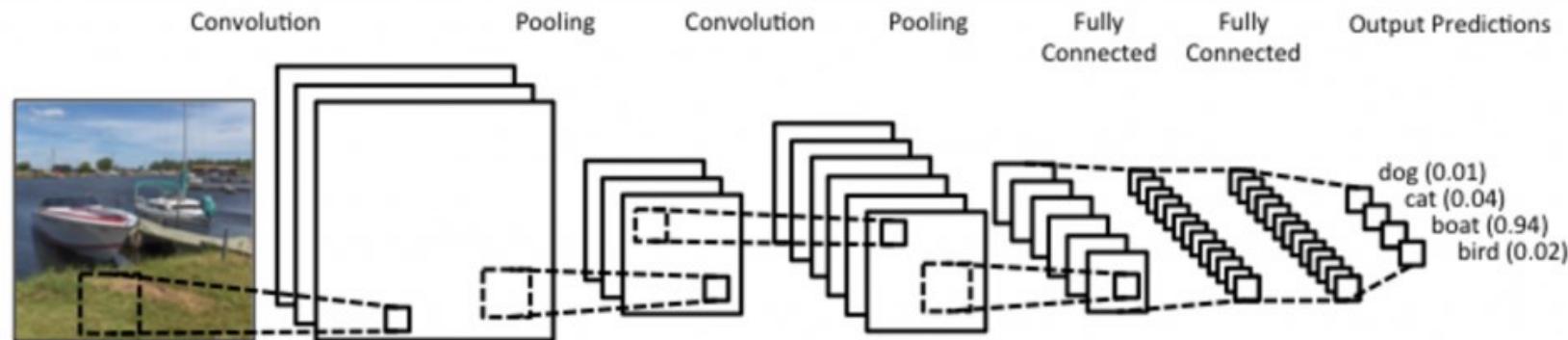


[Original image is  
CC0 public domain](#)

(all 3 images have same L2 distance to the one on the left)

# Convolutional Neural Network

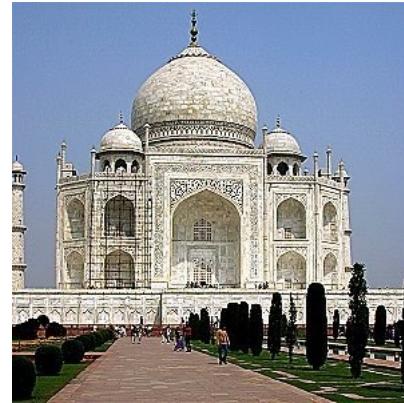
- Most widely used for image classification.
- Generally, it consists of convolution layer, pooling layer and fully-connected layer.
- Convolution, Pooling layer – feature extraction
- Fully-connected layer – classification



# Convolution Filters(Hand Crafted)



$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 5 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$



$$\begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix}$$



$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$



$$\begin{matrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{matrix}$$



# Let's Try!

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

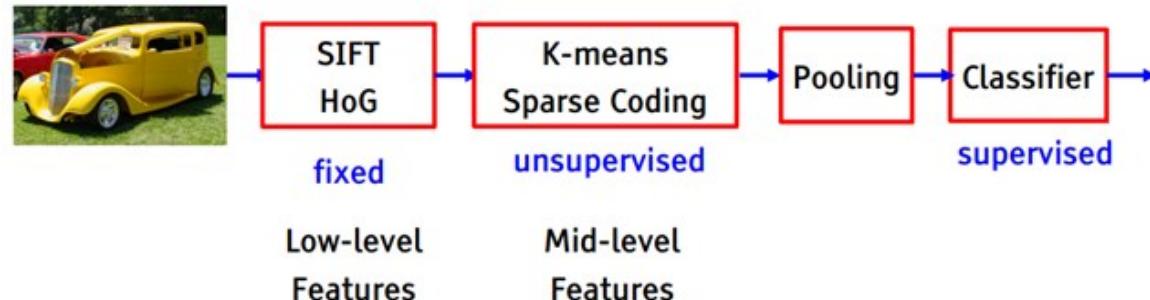
0	-1	0
-1	5	-1
0	-1	0

sharpen ▾

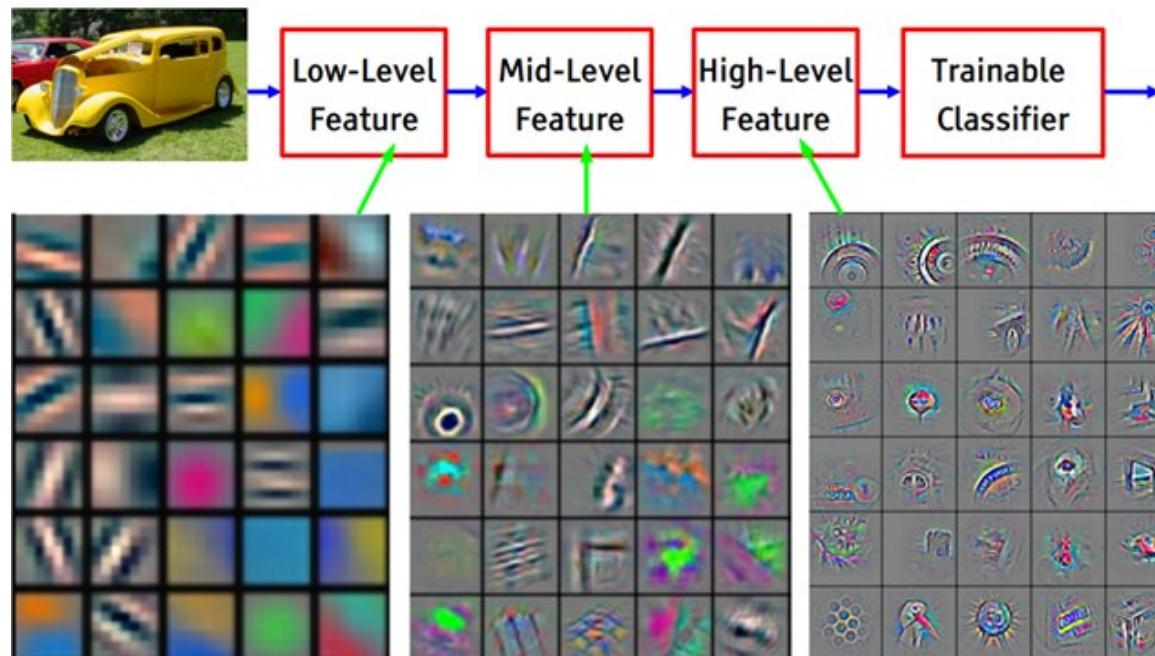


# Before and After

Object recognition 2006-2012

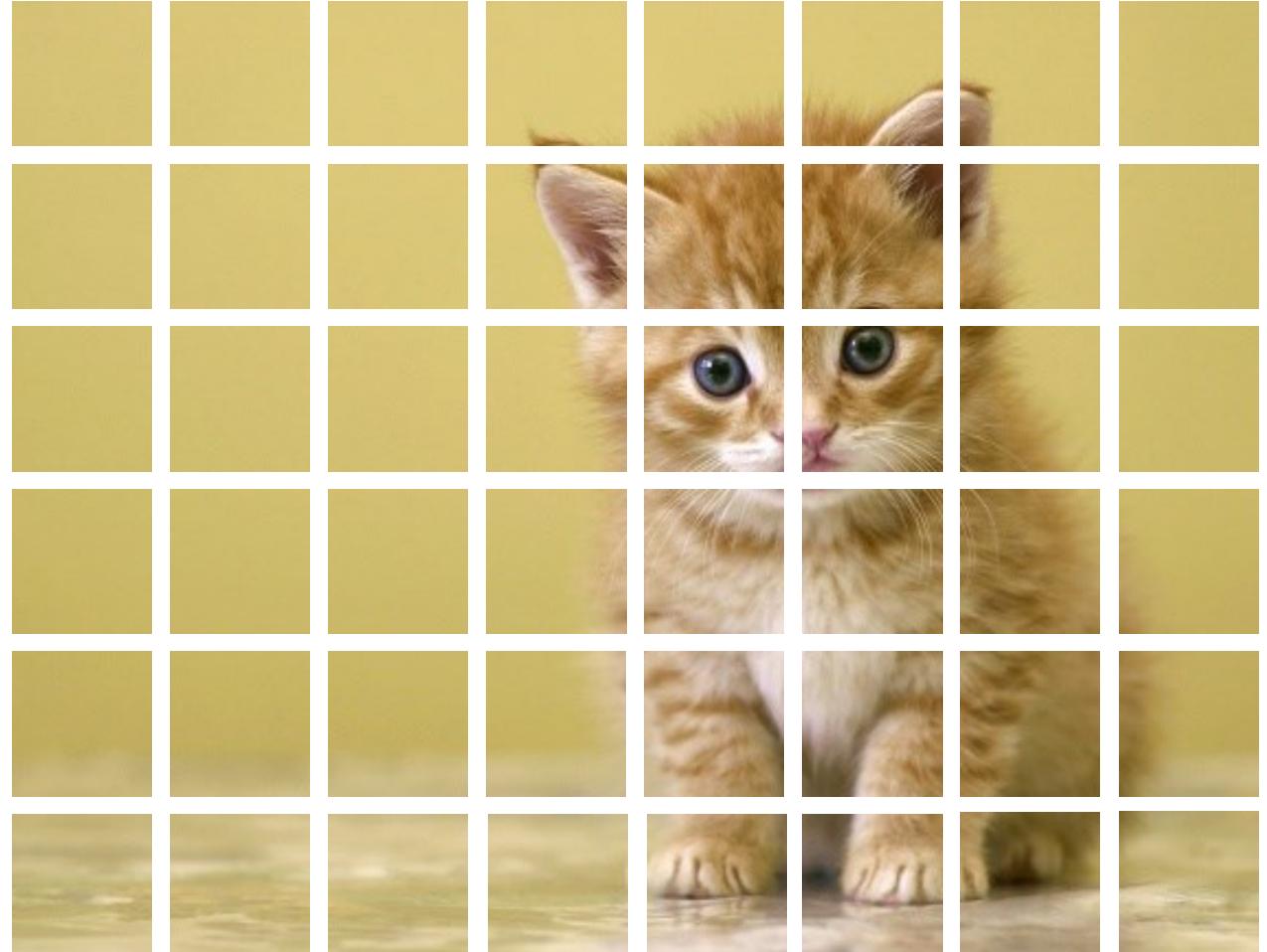


State of the art object recognition using CNNs



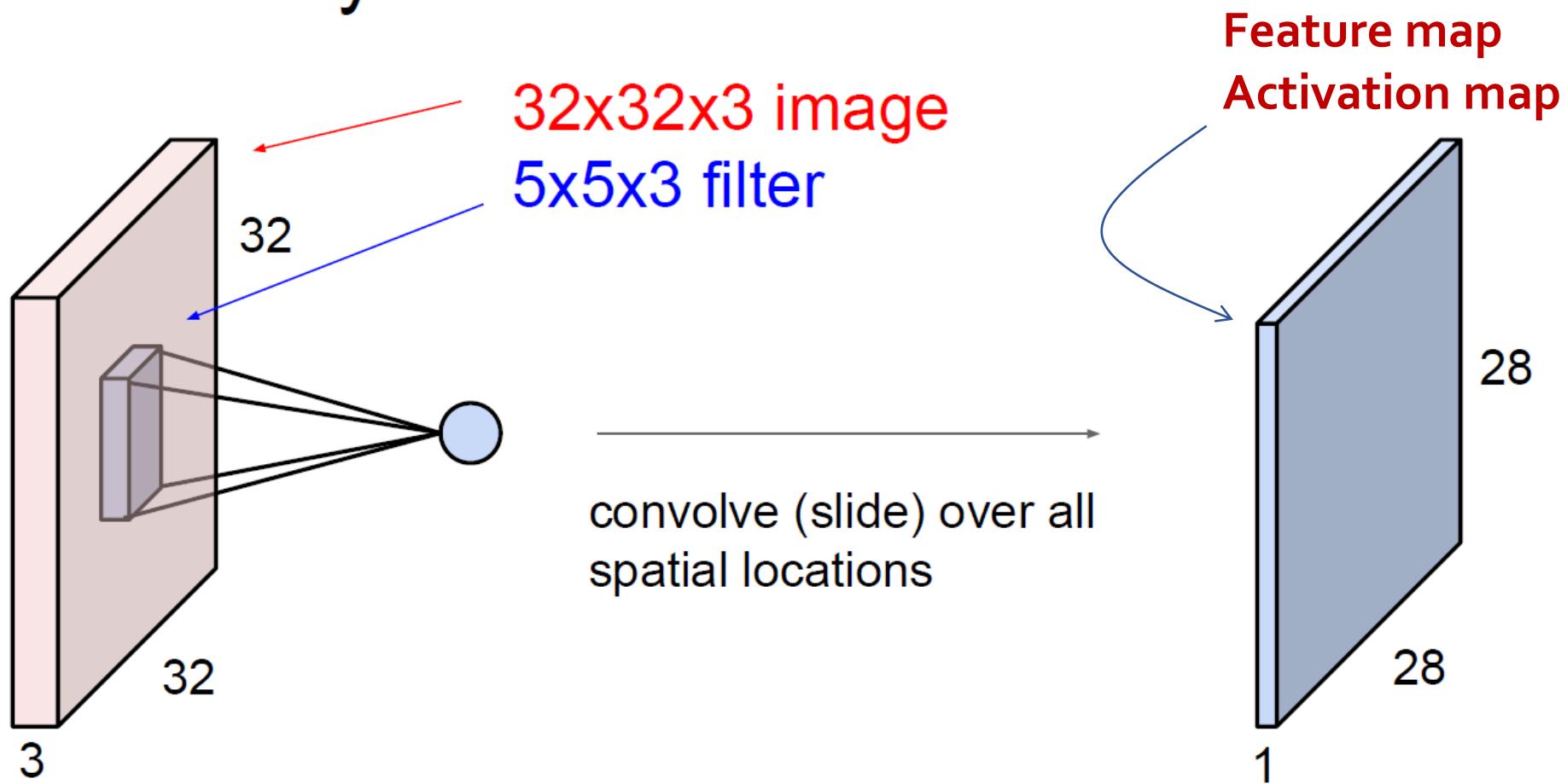
# CNN 동작원리

- 이미지를 작은 tile로 나누고, convolution filter를 통해 tile에서 특정 feature를 추출(예: 귀)
- Filter가 다음 tile로 이동하면서 같은 방법으로 feature를 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



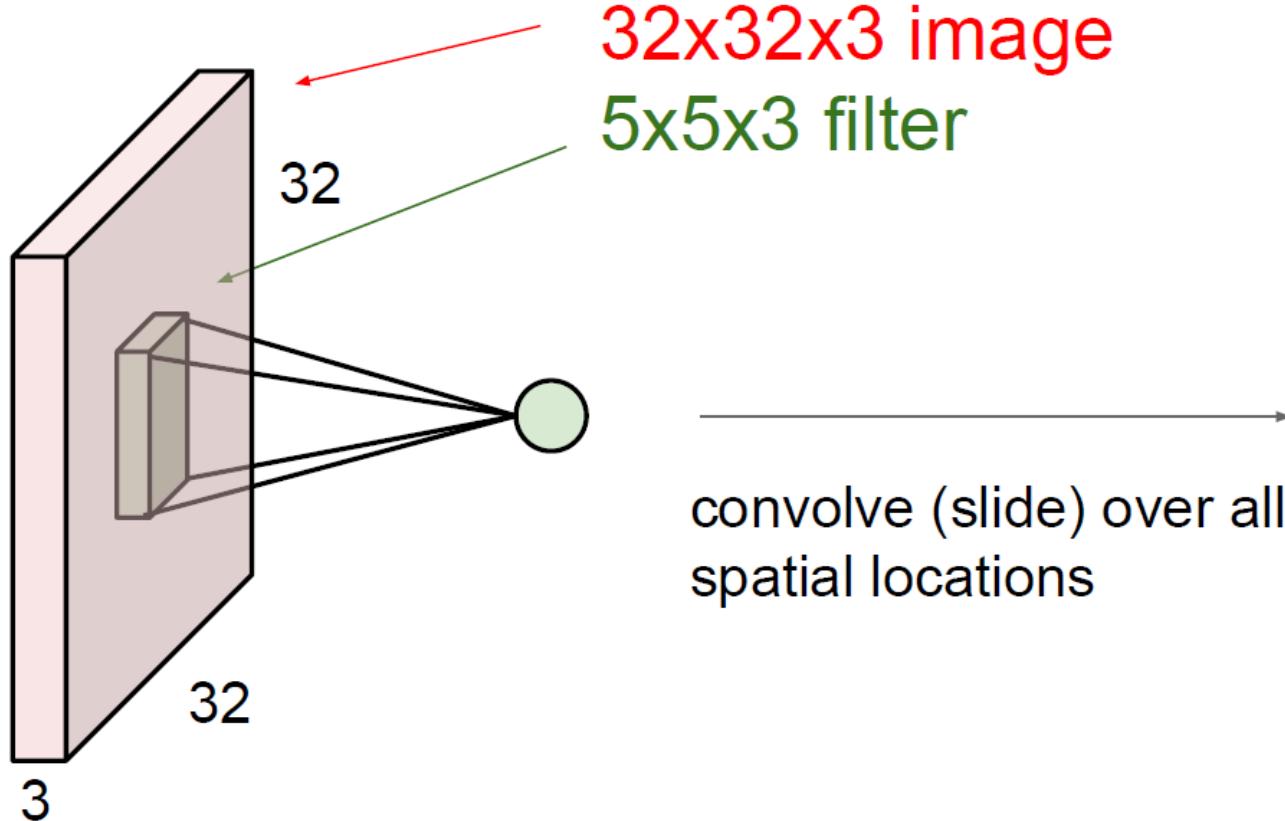
# 2D Convolution Layer

## Convolution Layer



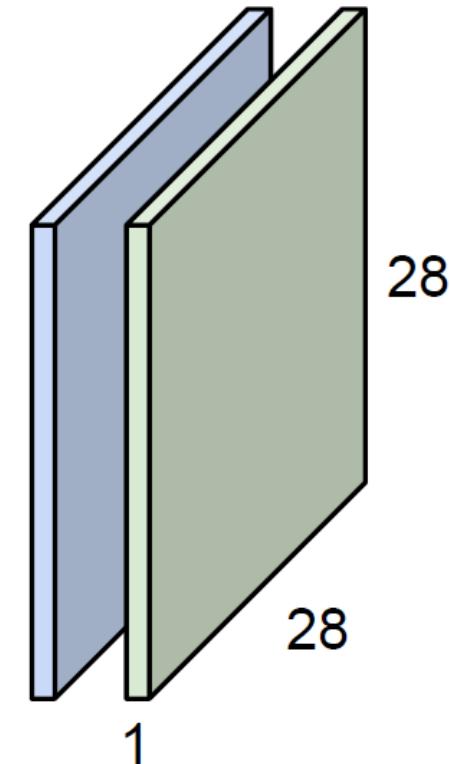
# 2D Convolution Layer

## Convolution Layer



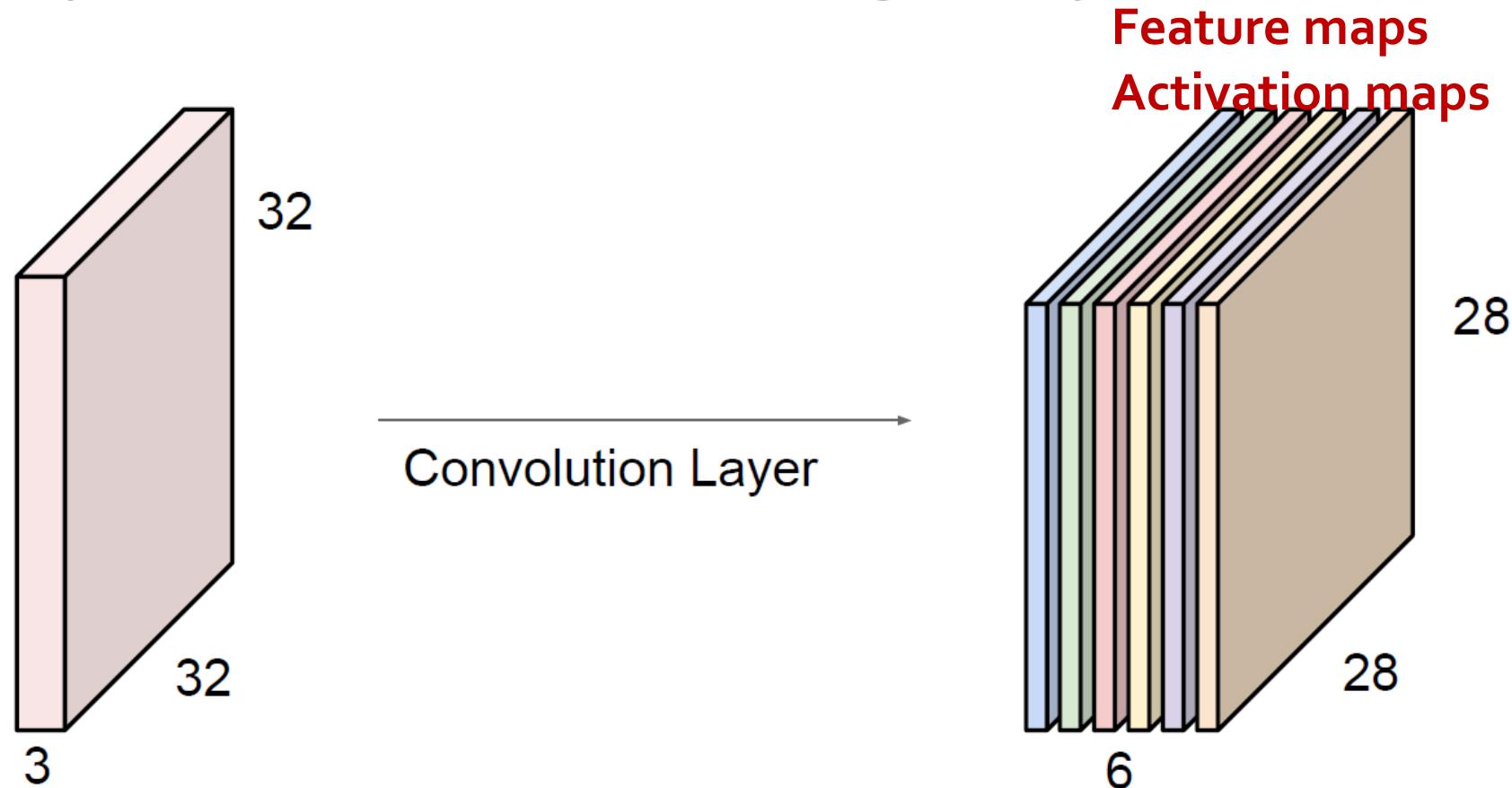
consider a second, green filter

**Feature maps**  
**Activation maps**



# 2D Convolution Layer

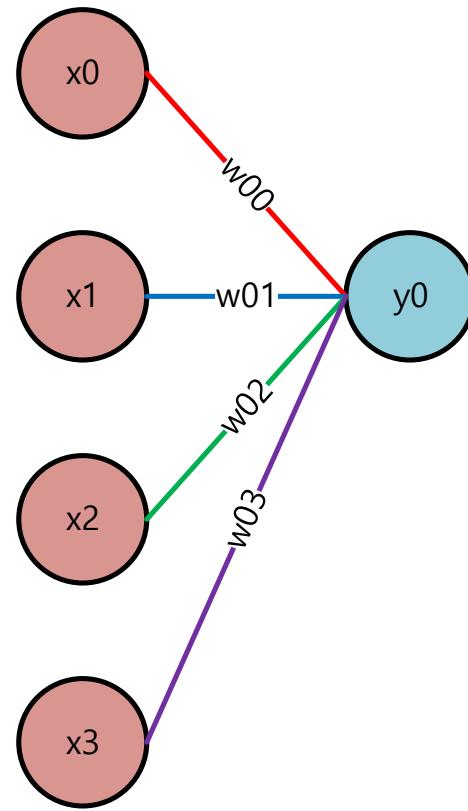
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

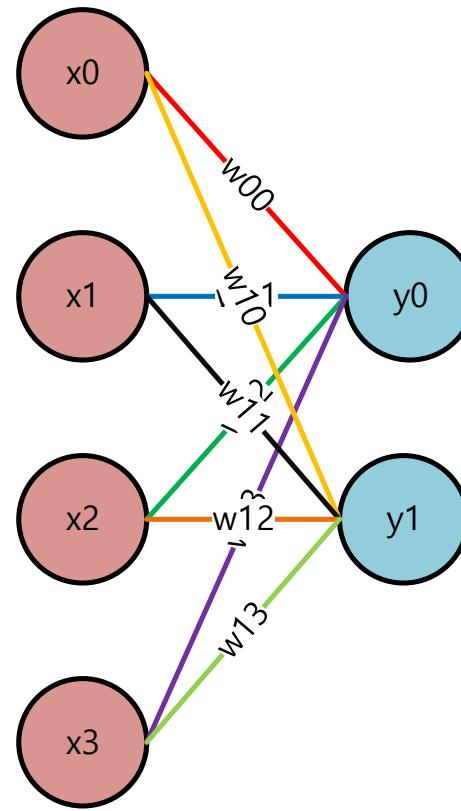
# Dense Layer vs 1-D Convolution Layer

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



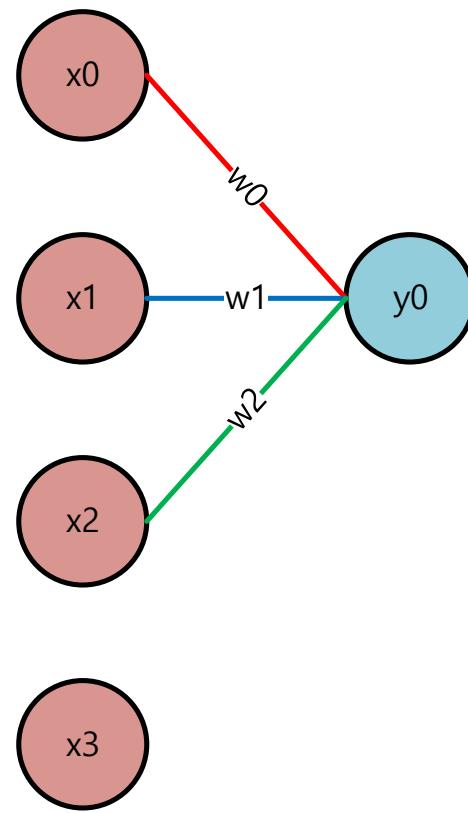
# Dense Layer vs 1-D Convolution Layer

- Dense Layer(Fully Connected Layer)
  - $y_0 = x_0 \cdot w_{00} + x_1 \cdot w_{01} + x_2 \cdot w_{02} + x_3 \cdot w_{03}$
  - $y_1 = x_0 \cdot w_{10} + x_1 \cdot w_{11} + x_2 \cdot w_{12} + x_3 \cdot w_{13}$



# Dense Layer vs 1-D Convolution Layer

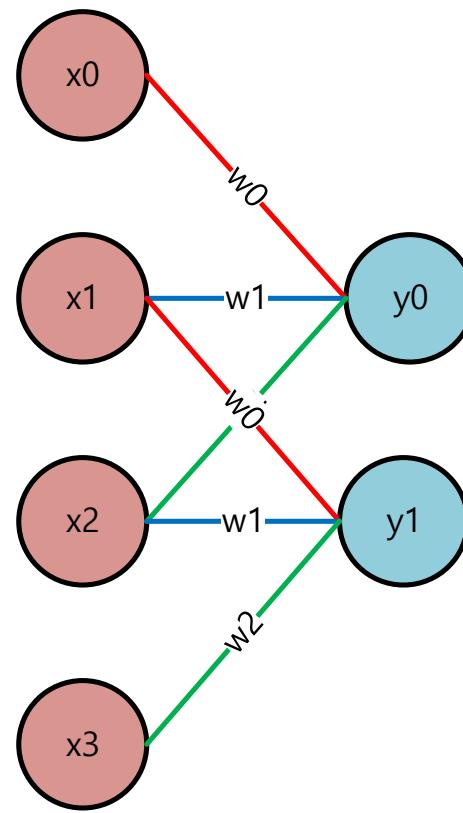
- 1-D Convolution Layer
  - $y0 = x0 \cdot w0 + x1 \cdot w1 + x2 \cdot w2$



# Dense Layer vs 1-D Convolution Layer

- 1-D Convolution Layer

- $y_0 = x_0 \cdot w_0 + x_1 \cdot w_1 + x_2 \cdot w_2$
- $y_0 = x_1 \cdot w_0 + x_2 \cdot w_1 + x_3 \cdot w_2$

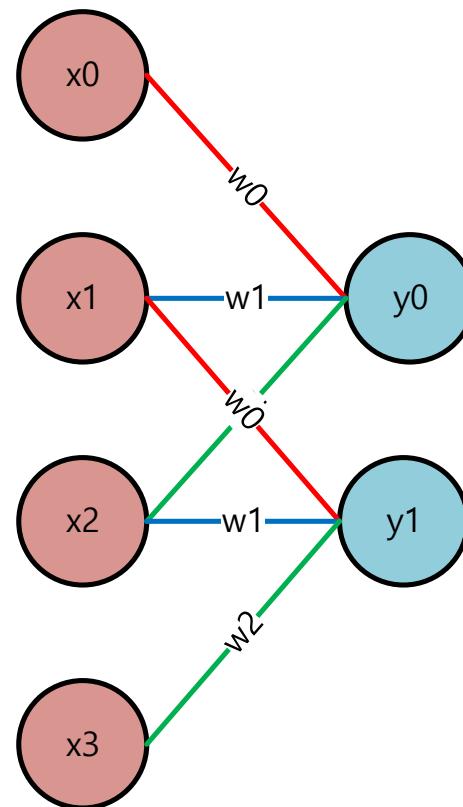


# Dense Layer vs 1-D Convolution Layer

- 1-D Convolution Layer

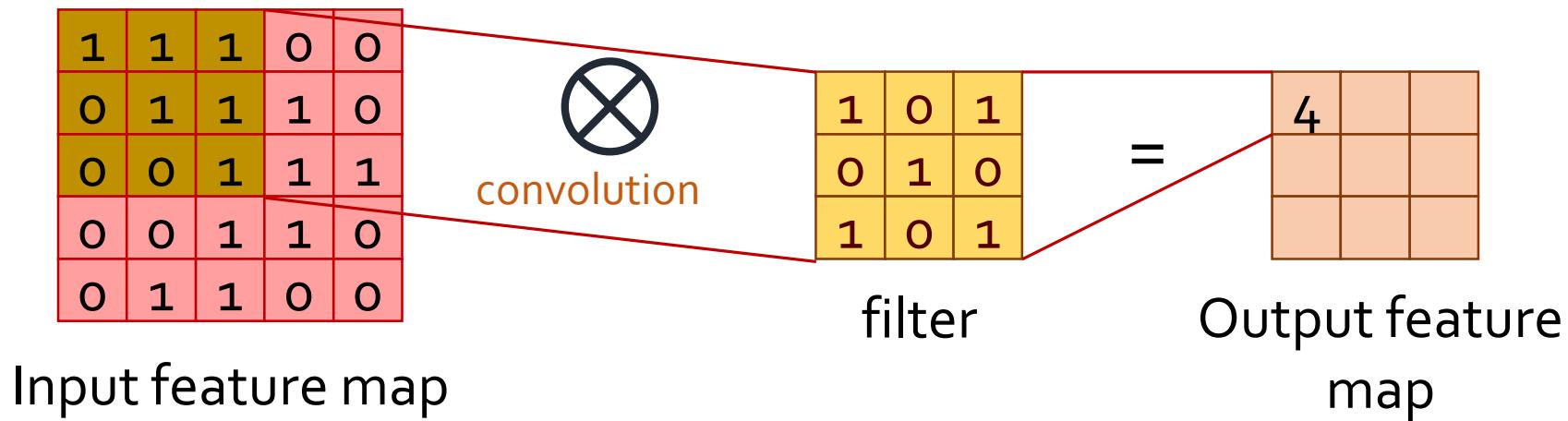
- $y_0 = x_0 \cdot w_0 + x_1 \cdot w_1 + x_2 \cdot w_2$
- $y_0 = x_1 \cdot w_0 + x_2 \cdot w_1 + x_3 \cdot w_2$

Weight sharing  
&  
Locally connected



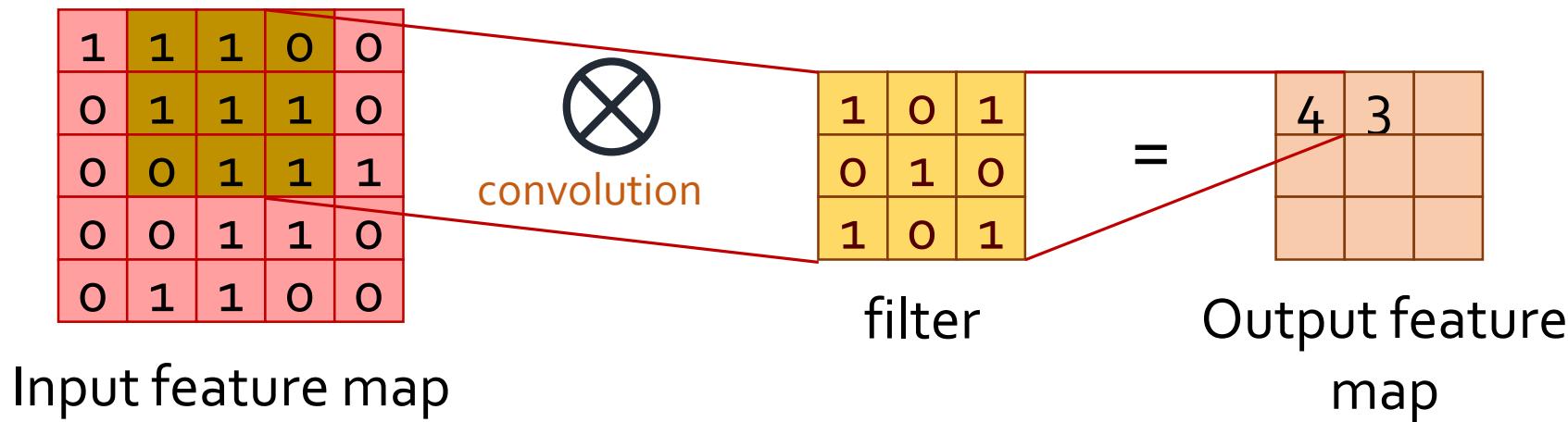
# 2D Convolution Layer – Computation

- $1 \times 1 + 1 \times 0 + 1 \times 1 + 0 \times 0 + 1 \times 1 + 1 \times 0 + 0 \times 1 + 0 \times 0 + 1 \times 1 = 4$



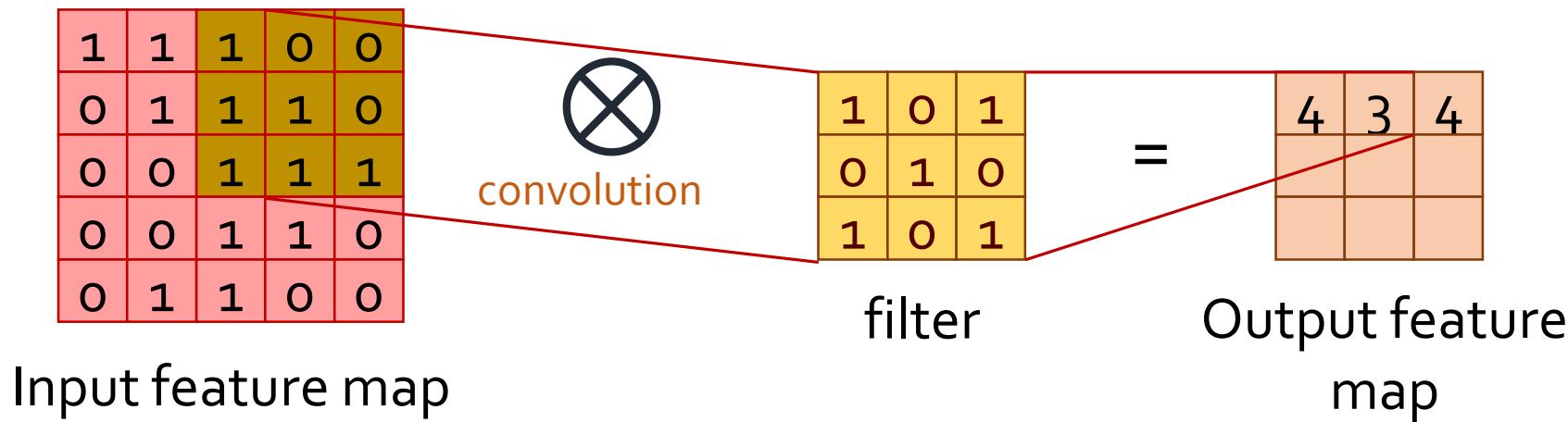
# 2D Convolution Layer – Computation

- $1 \times 1 + 1 \times 0 + 0 \times 1 + 1 \times 0 + 1 \times 1 + 1 \times 0 + 0 \times 1 + 1 \times 0 + 1 \times 1 = 3$



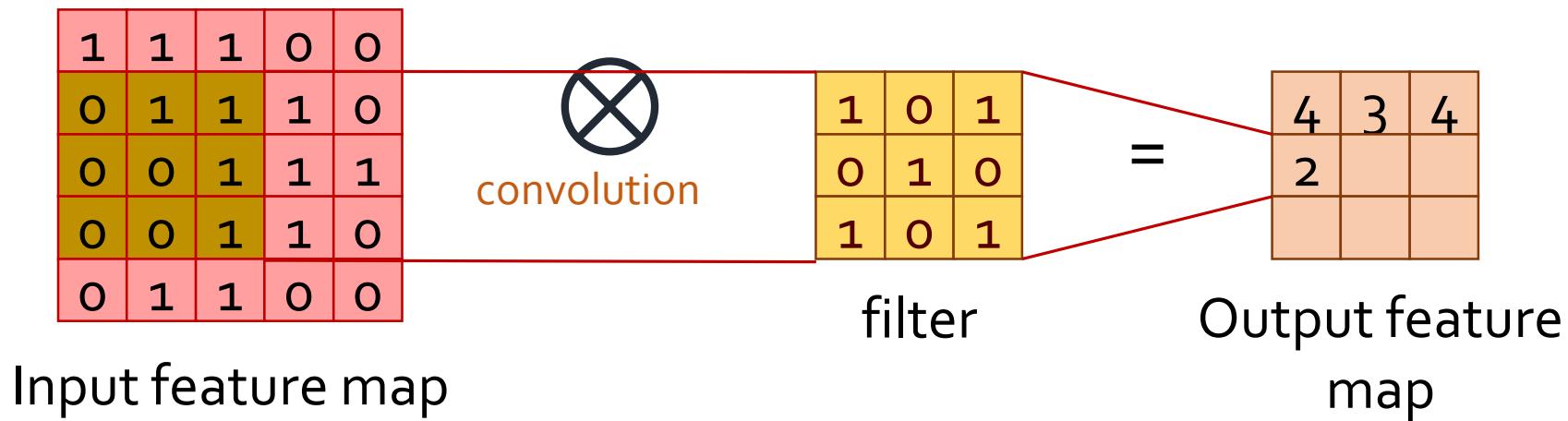
# 2D Convolution Layer – Computation

- $1 \times 1 + 0 \times 0 + 0 \times 1 + 1 \times 0 + 1 \times 1 + 0 \times 0 + 1 \times 1 + 1 \times 0 + 1 \times 1 = 4$



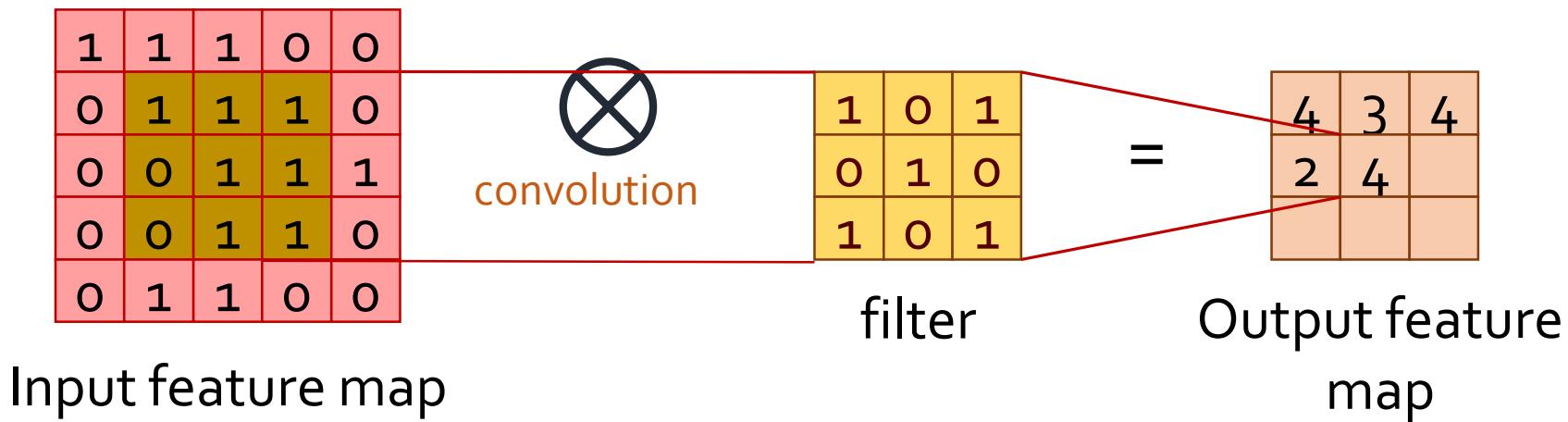
# 2D Convolution Layer – Computation

- $0 \times 1 + 1 \times 0 + 1 \times 1 + 0 \times 0 + 0 \times 1 + 1 \times 0 + 0 \times 1 + 0 \times 0 + 1 \times 1 = 2$



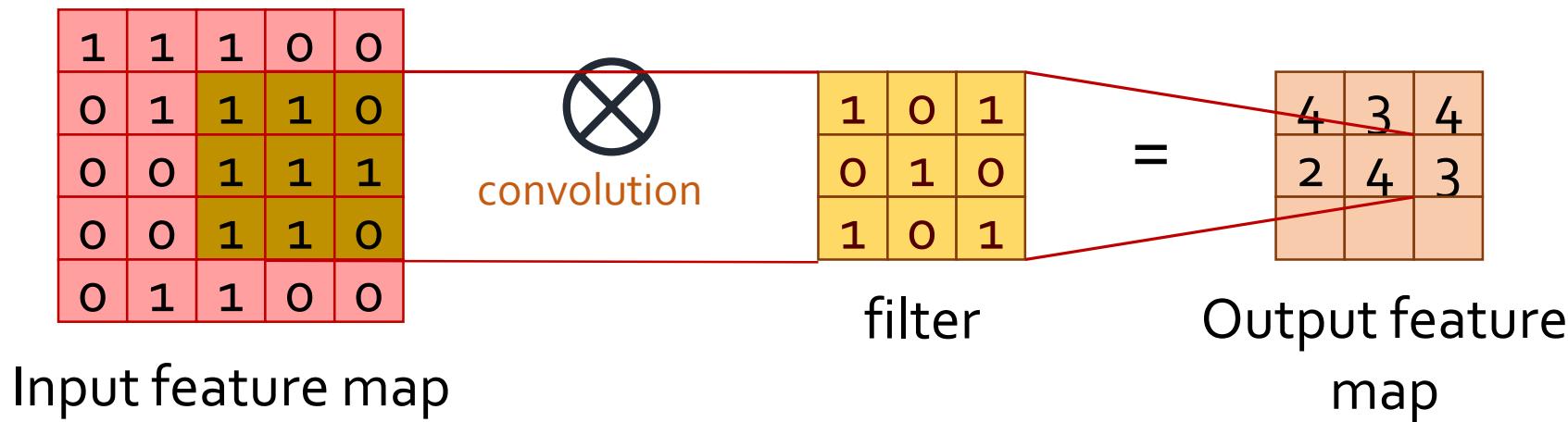
# 2D Convolution Layer – Computation

- $1 \times 1 + 1 \times 0 + 1 \times 1 + 0 \times 0 + 1 \times 1 + 1 \times 0 + 0 \times 1 + 1 \times 0 + 1 \times 1 = 4$



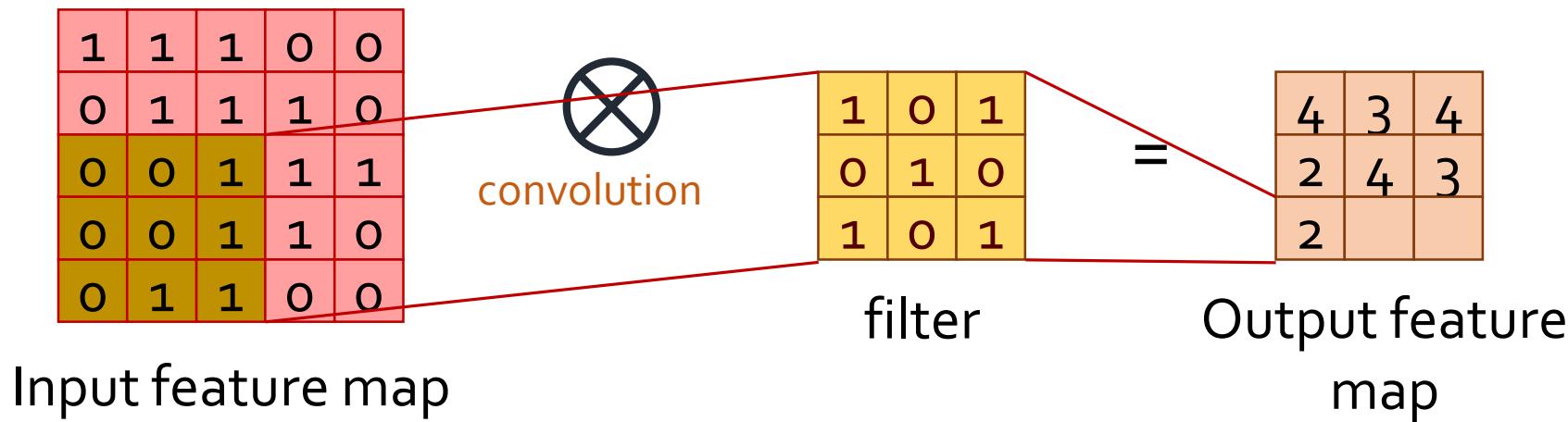
# 2D Convolution Layer – Computation

- $1 \times 1 + 1 \times 0 + 0 \times 1 + 1 \times 0 + 1 \times 1 + 1 \times 0 + 1 \times 1 + 1 \times 0 + 0 \times 1 = 3$



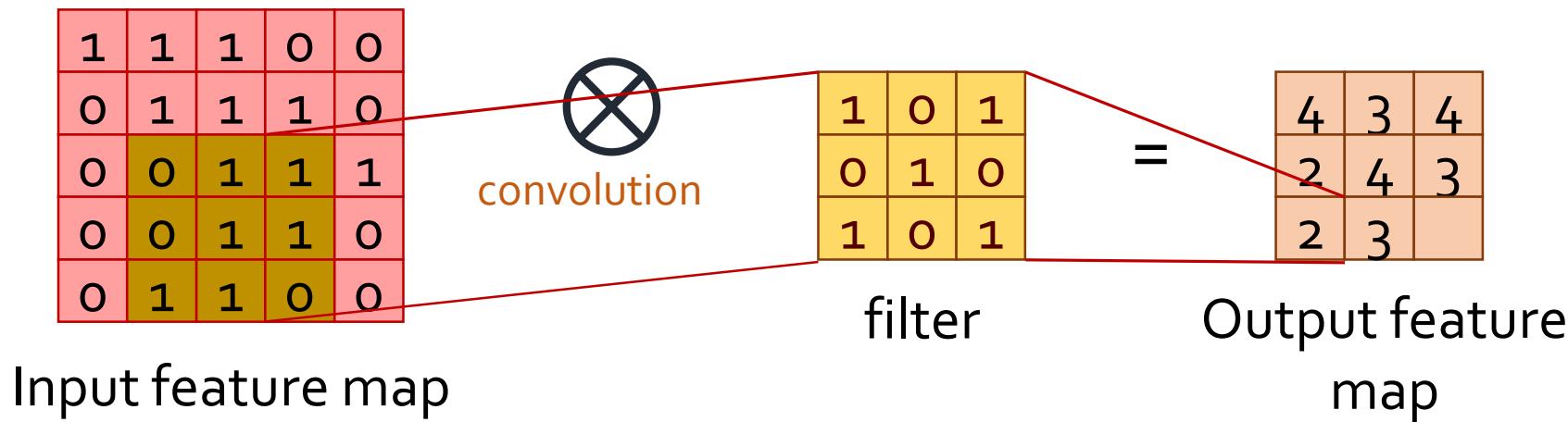
# 2D Convolution Layer – Computation

- $0 \times 1 + 0 \times 0 + 1 \times 1 + 0 \times 0 + 0 \times 1 + 1 \times 0 + 0 \times 1 + 1 \times 0 + 1 \times 1 = 2$



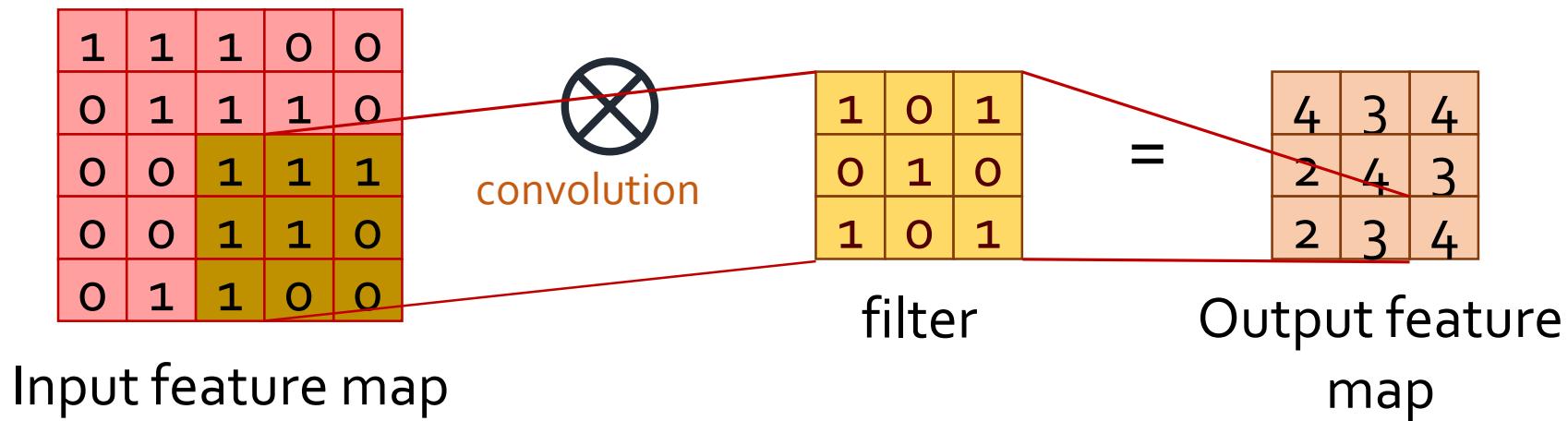
# 2D Convolution Layer – Computation

- $0 \times 1 + 1 \times 0 + 1 \times 1 + 0 \times 0 + 1 \times 1 + 1 \times 0 + 1 \times 1 + 1 \times 0 + 0 \times 1 = 3$



# 2D Convolution Layer – Computation

- $1 \times 1 + 1 \times 0 + 1 \times 1 + 1 \times 0 + 1 \times 1 + 0 \times 0 + 1 \times 1 + 0 \times 0 + 0 \times 1 = 4$



# 2D Convolution Layer – Computation

1 <small>×1</small>	1 <small>×0</small>	1 <small>×1</small>	0	0
0 <small>×0</small>	1 <small>×1</small>	1 <small>×0</small>	1	0
0 <small>×1</small>	0 <small>×0</small>	1 <small>×1</small>	1	1
0	0	1	1	0
0	1	1	0	0

Image

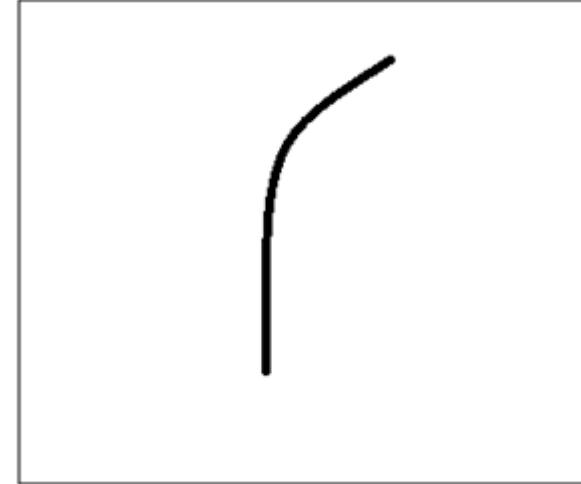
4		

Convolved  
Feature

# 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

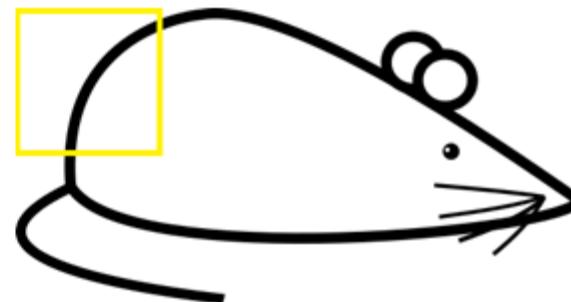
Pixel representation of filter



Visualization of a curve detector filter



Original image



Visualization of the filter on the image

# 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	
0	0	0	50	50	0	0	

Pixel representation of the receptive field

\*

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

Pixel representation of filter

$$\text{Multiplication and Summation} = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 \text{ (A large number!)}$$



Visualization of the filter on the image

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

Pixel representation of receptive field

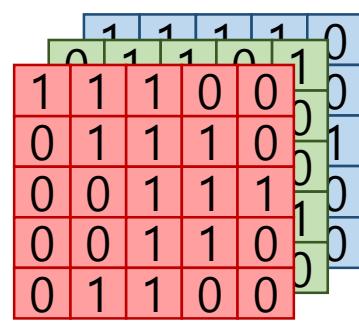
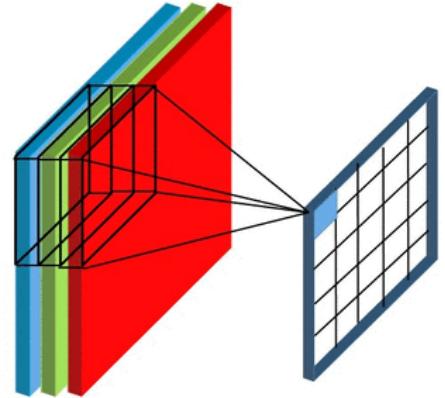
\*

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

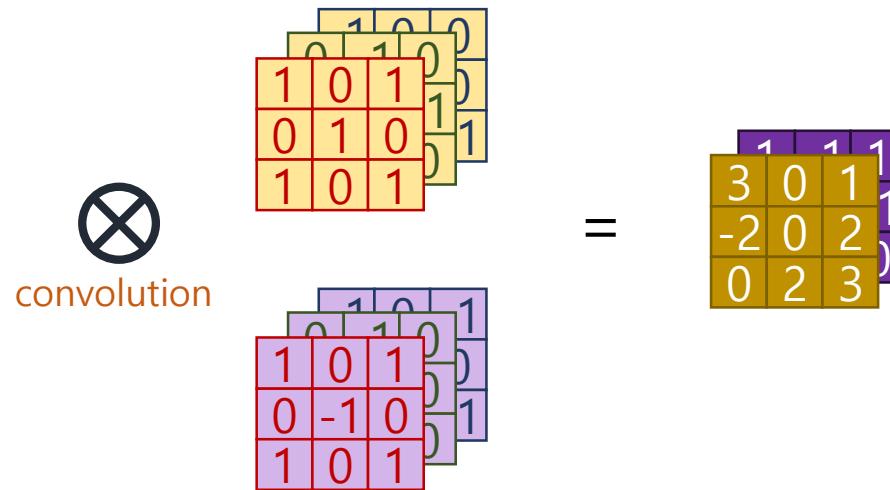
Pixel representation of filter

$$\text{Multiplication and Summation} = 0$$

# Convolution (Multi Channel, Many Filters)



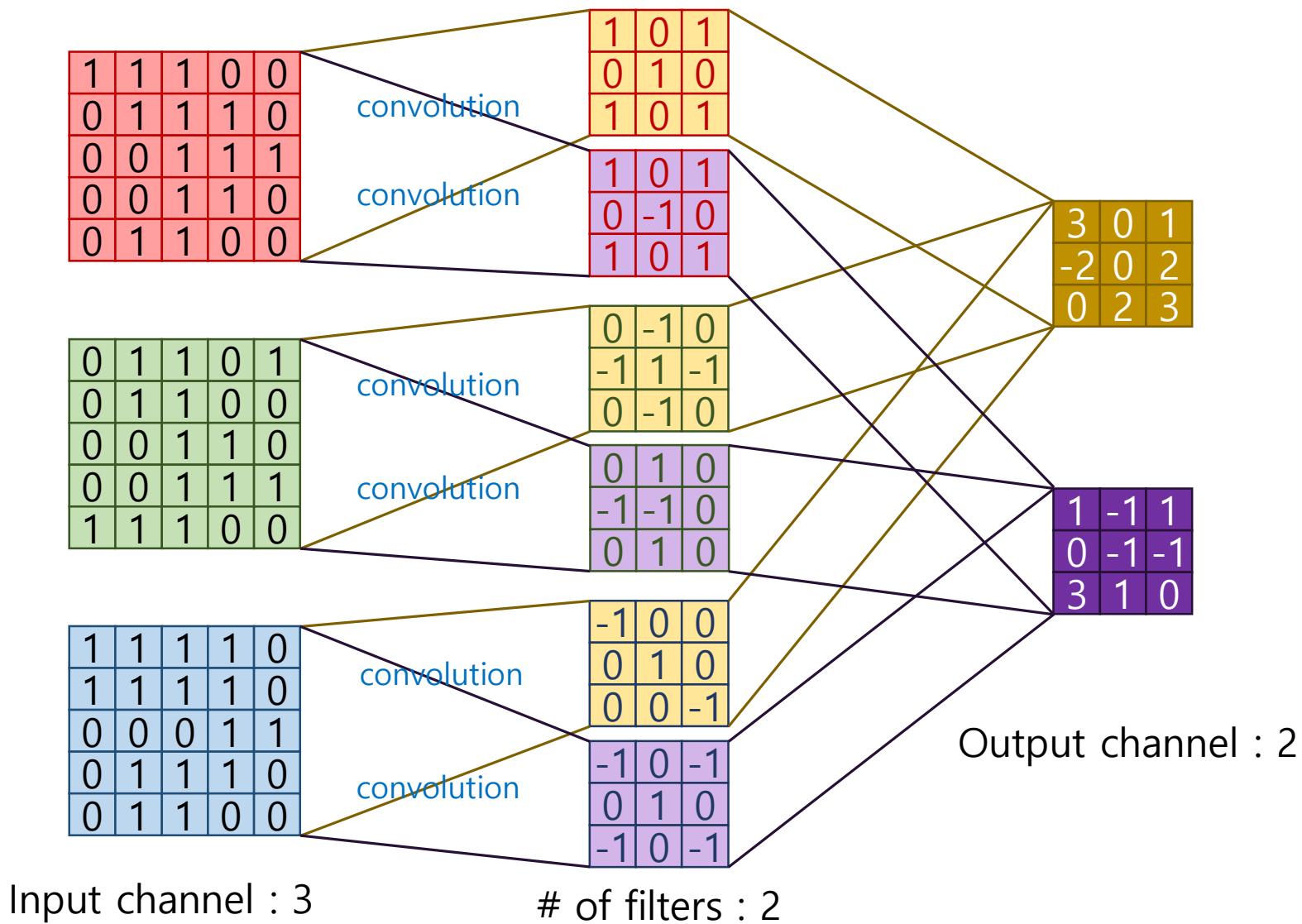
Input channel : 3



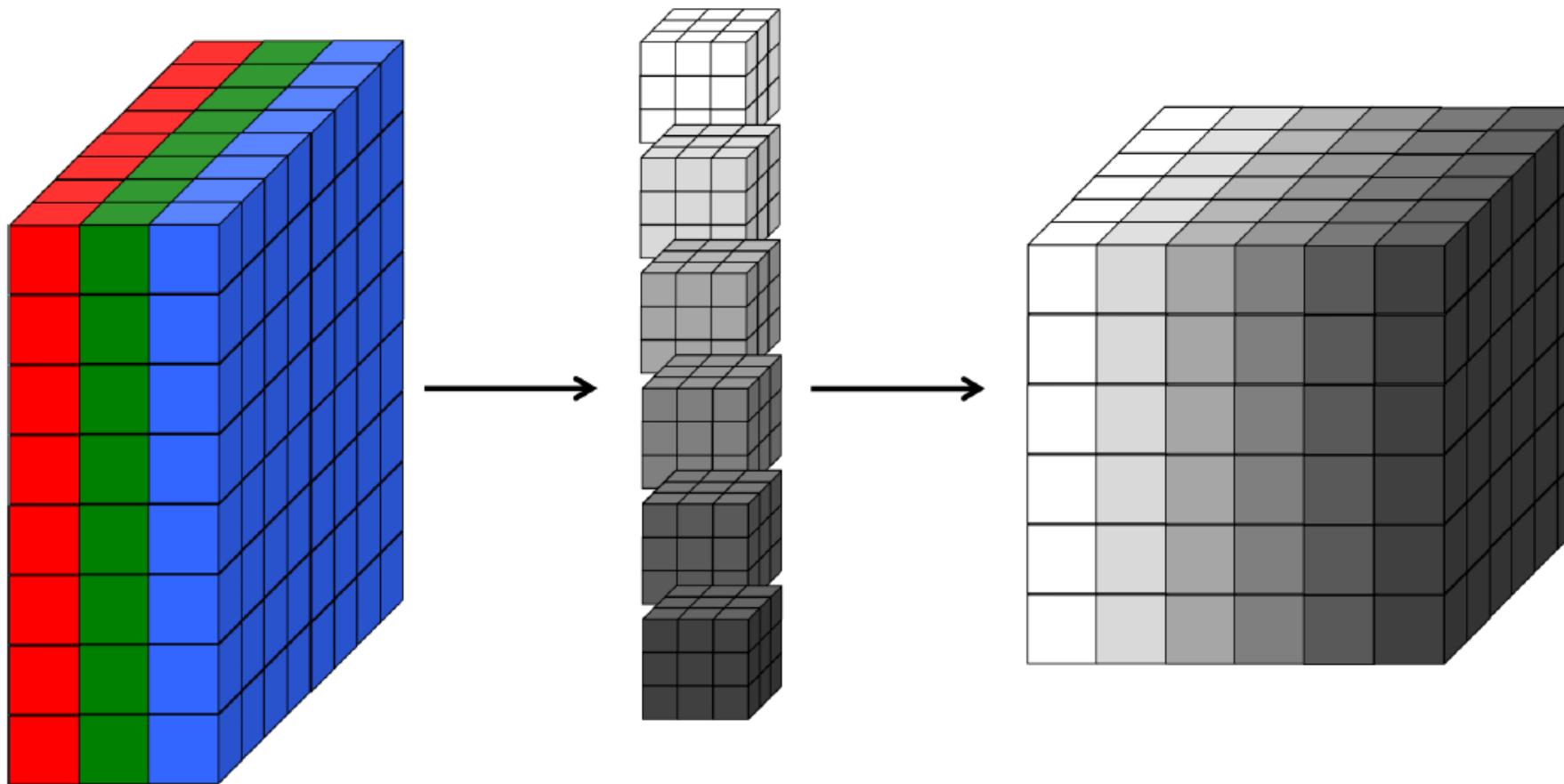
# of filters : 2

Output channel : 2

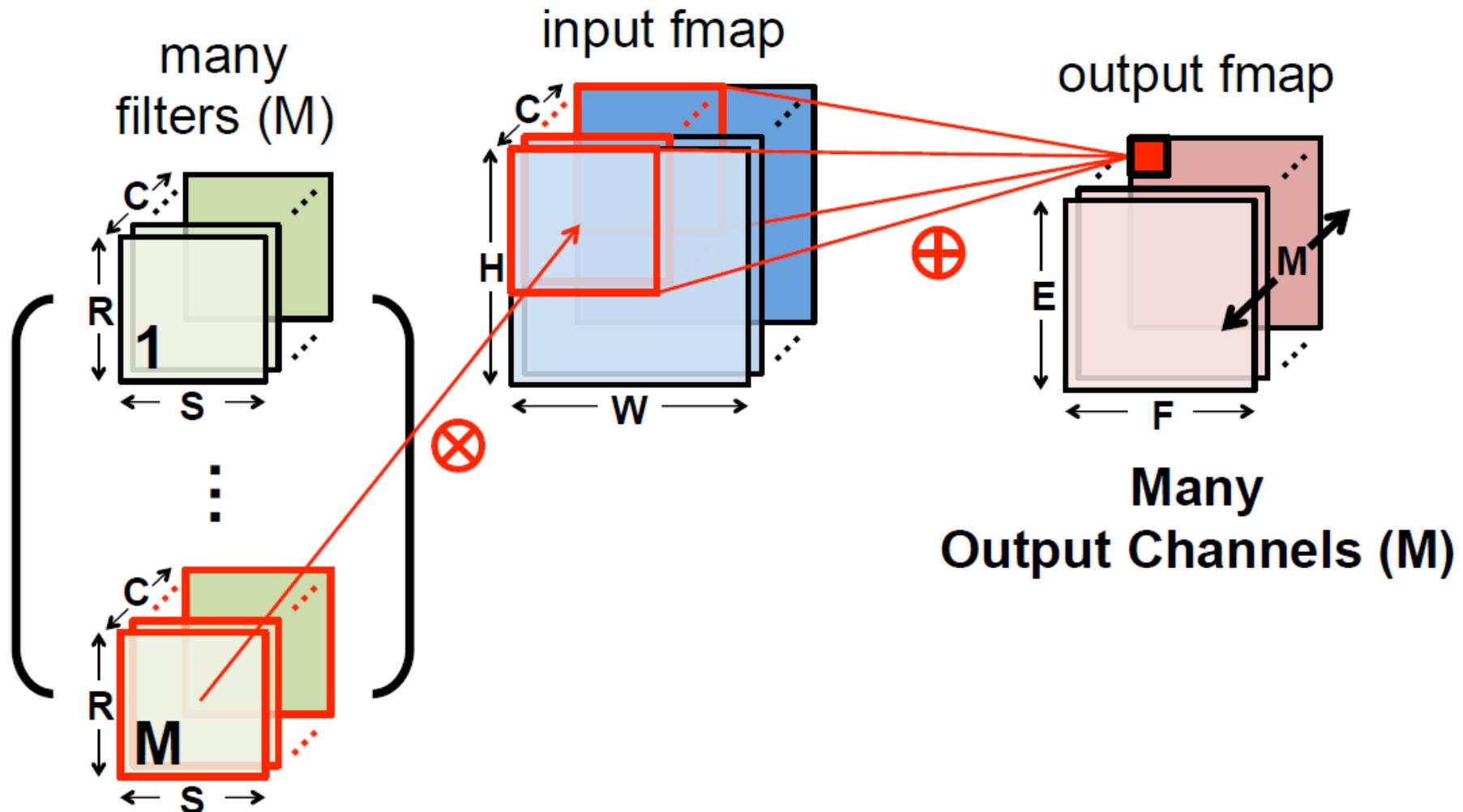
# Convolution (Multi Channel, Many Filters)



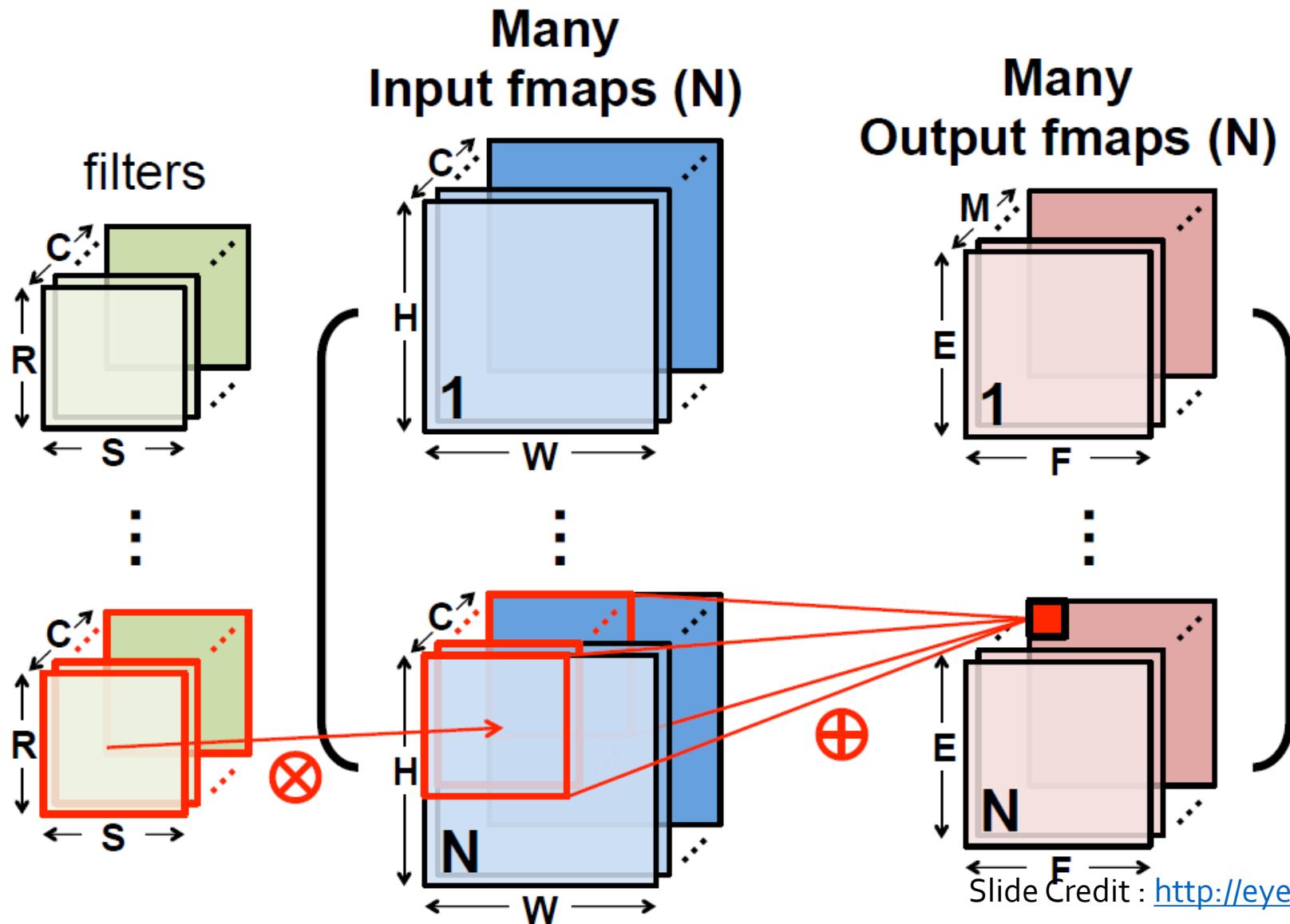
# Visualization of a Convolution Layer



# 2D Convolution Layer

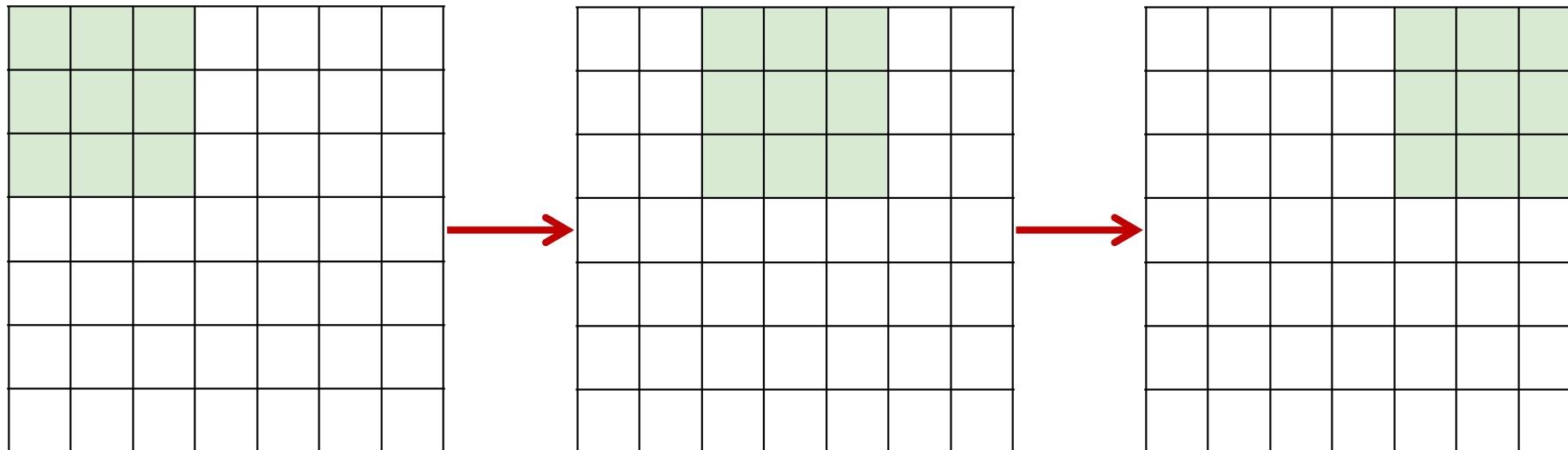


# 2D Convolution Layer – 4D Tensors



# 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!



# Options of Convolution

- Zero Padding

0	0	0	0	0	0			
0								
0								
0								
0								

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.  $F = 3 \Rightarrow$  zero pad with 1

$F = 5 \Rightarrow$  zero pad with 2

$F = 7 \Rightarrow$  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

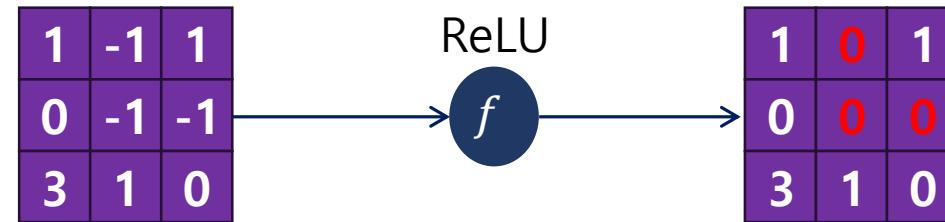
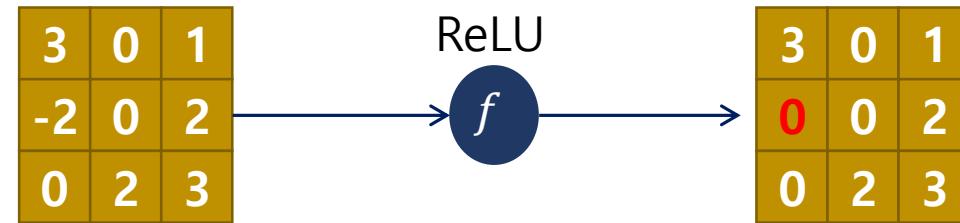
*Input O/  $W_i \times H_i \times C_i$  O/고,  
F × F filter를 K 개 사용하고,  
stride 는 S,  
zero padding 은 P 만큼 했을 경우,  
output feature map size( $W_o \times H_o \times C_o$ )는,*

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

- Answer

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

# ReLU



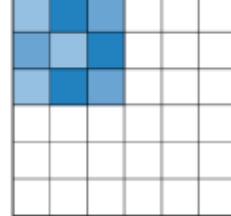
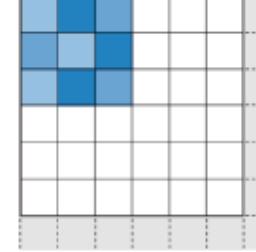
# tf.keras.layers.Conv2D

```
__init__(  
    filters,  
    kernel_size,  
    strides=(1, 1),  
    padding='valid',  
    data_format=None,  
    dilation_rate=(1, 1),  
    activation=None,  
    use_bias=True,  
    kernel_initializer='glorot_uniform',  
    bias_initializer='zeros',  
    kernel_regularizer=None,  
    bias_regularizer=None,  
    activity_regularizer=None,  
    kernel_constraint=None,  
    bias_constraint=None,  
    **kwargs  
)
```

# tf.keras.layers.Conv2D

- `filters` : Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- `kernel_size` : An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- `strides` : An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
- `padding` : one of `"valid"` or `"same"` (case-insensitive).
- `data_format` : A string, one of `channels_last` (default) or `channels_first`. The ordering of the dimensions in the inputs. `channels_last` corresponds to inputs with shape `(batch, height, width, channels)` while `channels_first` corresponds to inputs with shape `(batch, channels, height, width)`. It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "channels\_last".

# Padding – SAME vs VALID

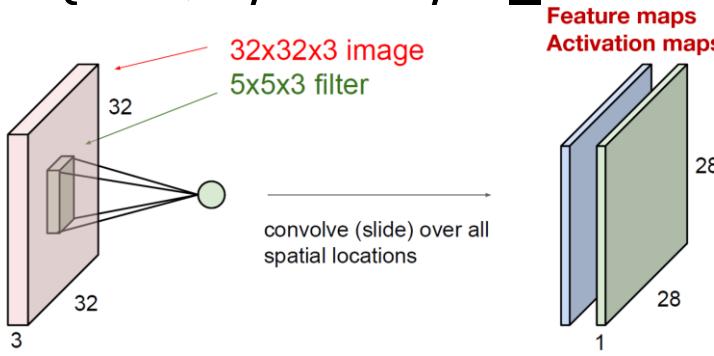
	Valid	Same
Value	$P = 0$	$P_{\text{start}} = \left\lceil \frac{S \lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rceil$ $P_{\text{end}} = \left\lceil \frac{S \lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rceil$
Illustration		
Purpose	<ul style="list-style-type: none"><li>- No padding</li><li>- Drops last convolution if dimensions do not match</li></ul>	<ul style="list-style-type: none"><li>- Padding such that feature map size has size <math>\left\lceil \frac{I}{S} \right\rceil</math></li><li>- Output size is mathematically convenient</li><li>- Also called 'half' padding</li></ul>

# tf.keras.layers.Conv2D

- `activation` : Activation function to use. If you don't specify anything, no activation is applied (ie. "linear" activation: `a(x) = x`).
- `use_bias` : Boolean, whether the layer uses a bias vector.
- `kernel_initializer` : Initializer for the `kernel` weights matrix.
- `bias_initializer` : Initializer for the bias vector.
- `kernel_regularizer` : Regularizer function applied to the `kernel` weights matrix.
- `bias_regularizer` : Regularizer function applied to the bias vector.

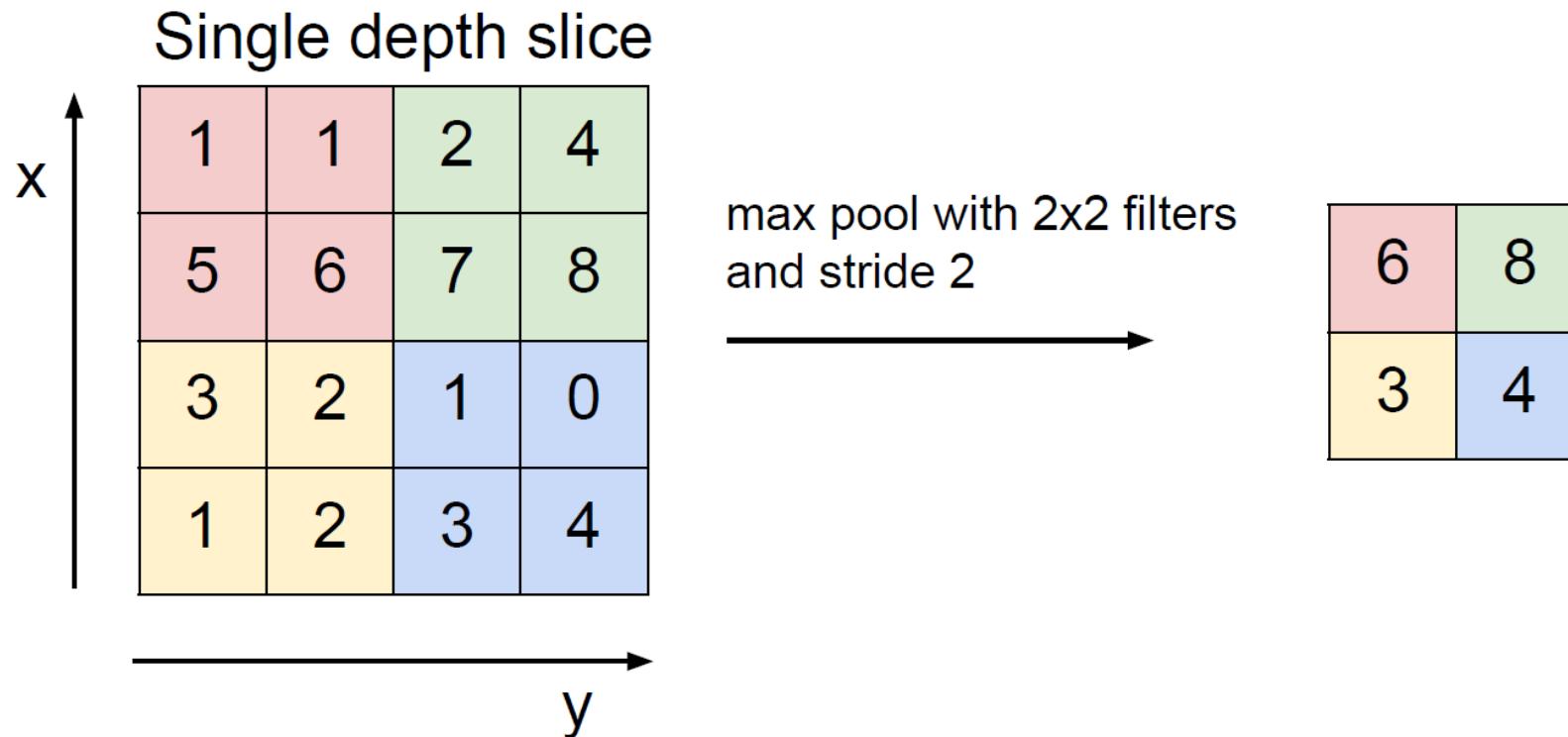
kernel dimension : {height, width, in\_channel, out\_channel}

Ex) {5, 5, 3, 2}

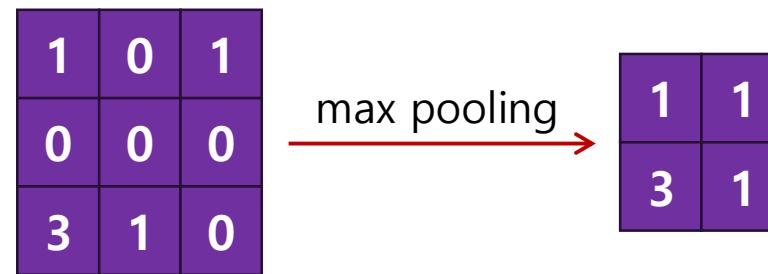
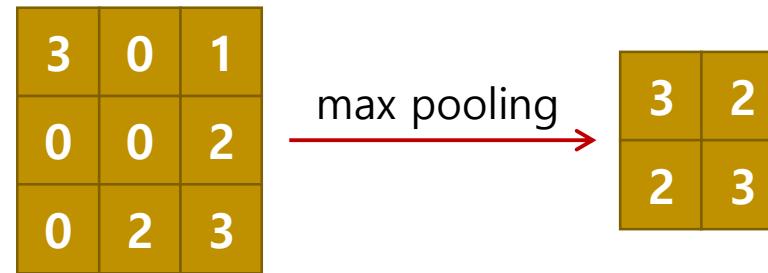


# Pooling Layer

- Max pooling or Average Pooling



# 2x2 Max Pooling with Stride=1



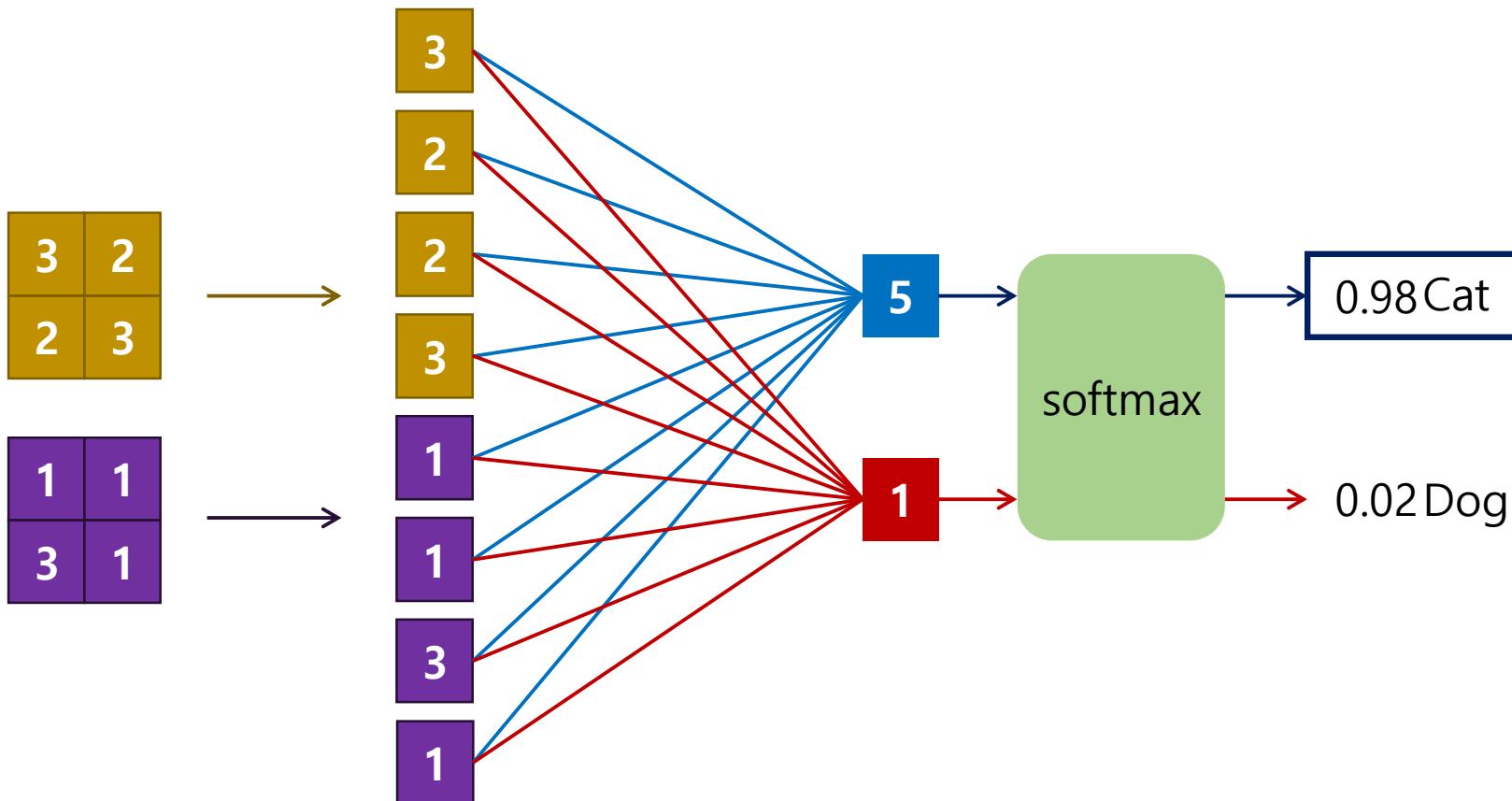
# tf.keras.layers.MaxPool2D

```
__init__(  
    pool_size=(2, 2),  
    strides=None,  
    padding='valid',  
    data_format=None,  
    **kwargs  
)
```

# tf.keras.layers.MaxPool2D

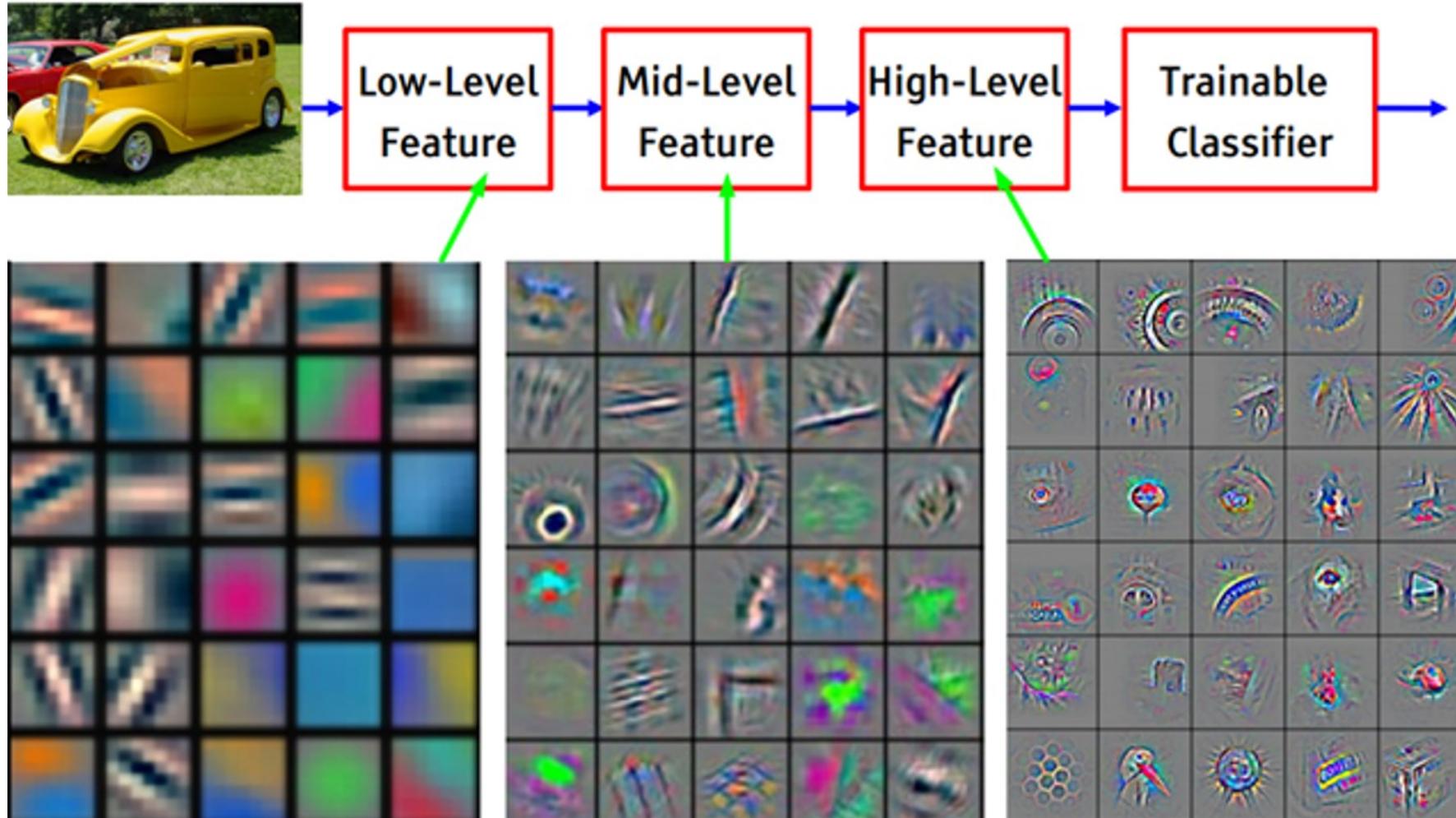
- `pool_size` : integer or tuple of 2 integers, factors by which to downscale (vertical, horizontal). (2, 2) will halve the input in both spatial dimension. If only one integer is specified, the same window length will be used for both dimensions.
- `strides` : Integer, tuple of 2 integers, or None. Strides values. If None, it will default to `pool_size`.
- `padding` : One of "valid" or "same" (case-insensitive).
- `data_format` : A string, one of `channels_last` (default) or `channels_first`. The ordering of the dimensions in the inputs. `channels_last` corresponds to inputs with shape `(batch, height, width, channels)` while `channels_first` corresponds to inputs with shape `(batch, channels, height, width)`. It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "channels\_last".

# Fully-Connected Layer



# Convolutional Neural Network

State of the art object recognition using CNNs



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