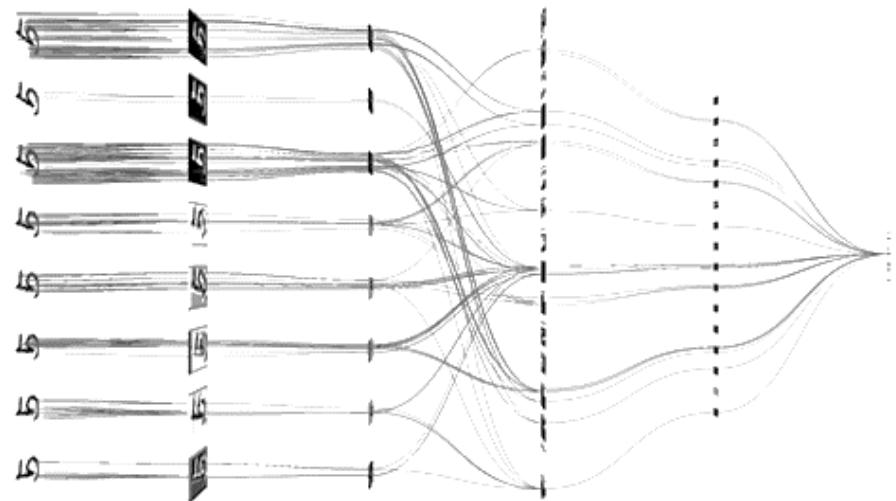


# Convolutional Neural Network



Fast Campus  
Start Deep Learning with Tensorflow

# Image Classification

- A core task in Computer Vision



This image by Nikita is  
licensed under CC-BY 2.0

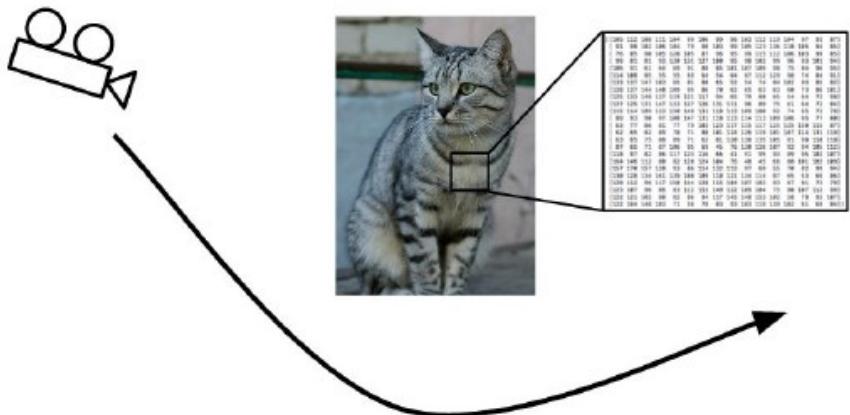
(assume given set of discrete labels)  
{dog, cat, truck, plane, ...}



cat

# 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

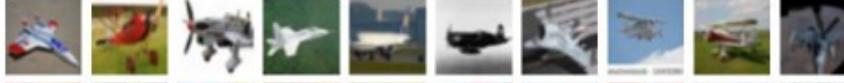
# Example Dataset: CIFAR10

**10** classes

**50,000** training images

**10,000** testing images

**airplane**



**automobile**



**bird**



**cat**



**deer**



**dog**



**frog**



**horse**



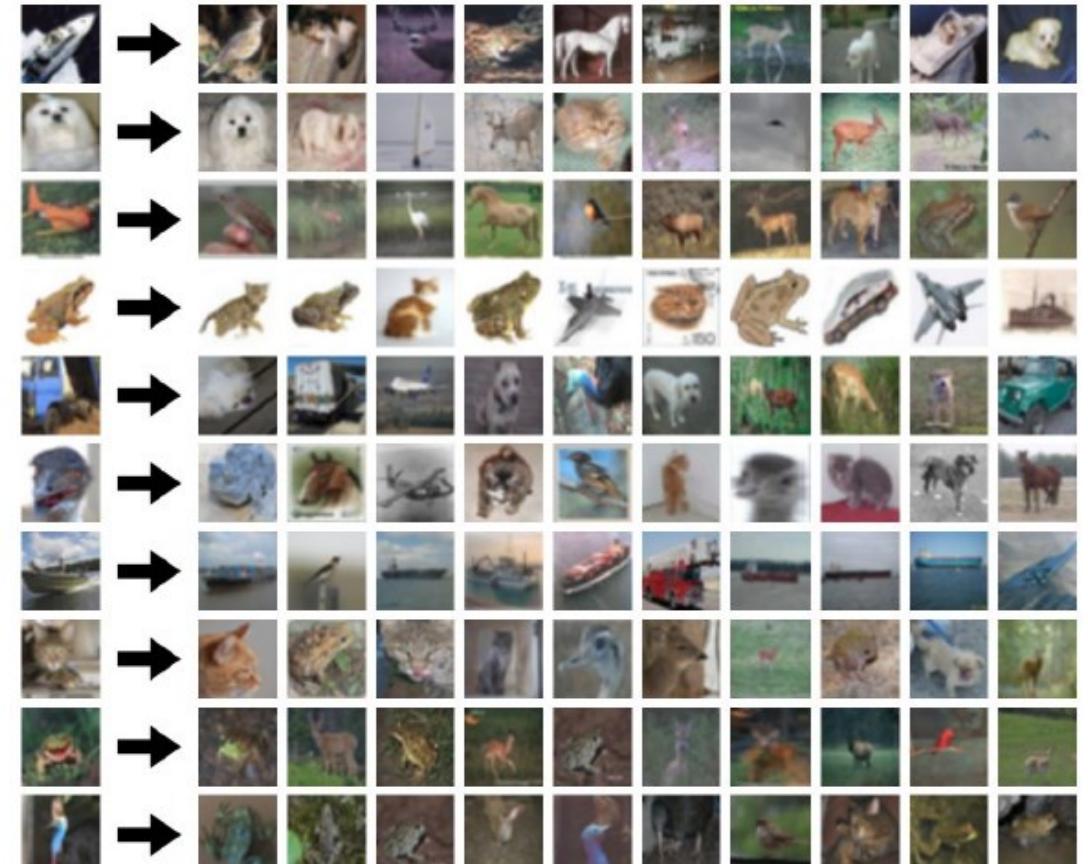
**ship**



**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

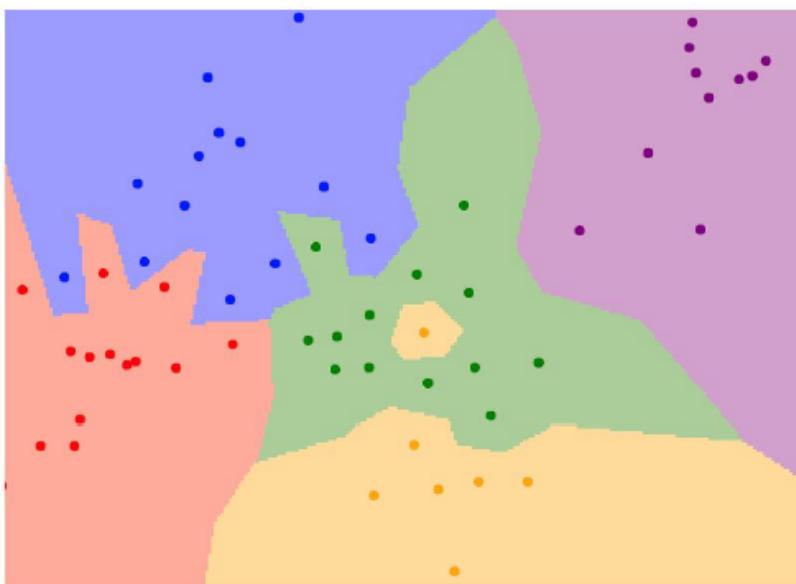
46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

=

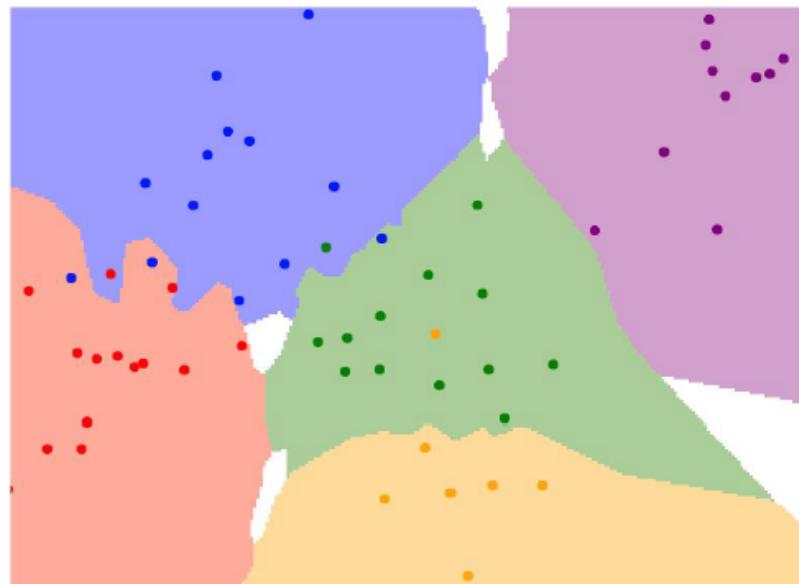
add → 456

# K-Nearest Neighbors

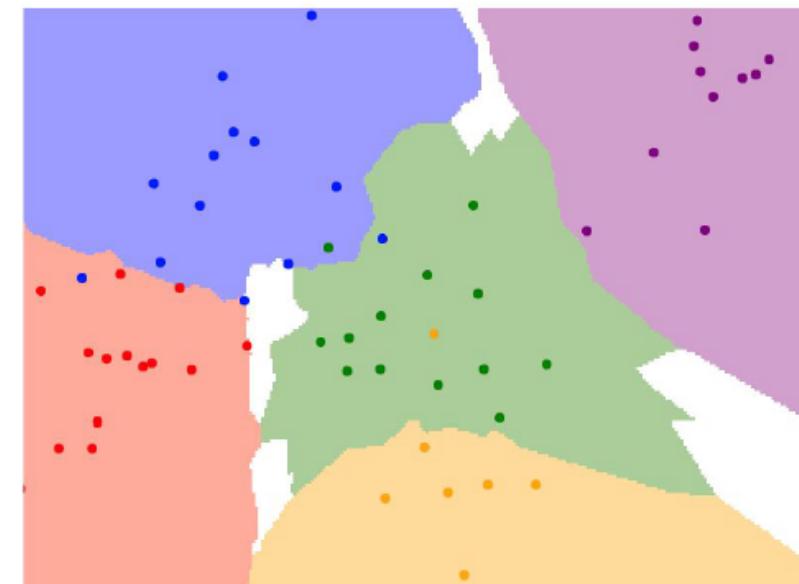
Instead of copying label from nearest neighbor,  
take **majority vote** from K closest points



$K = 1$



$K = 3$

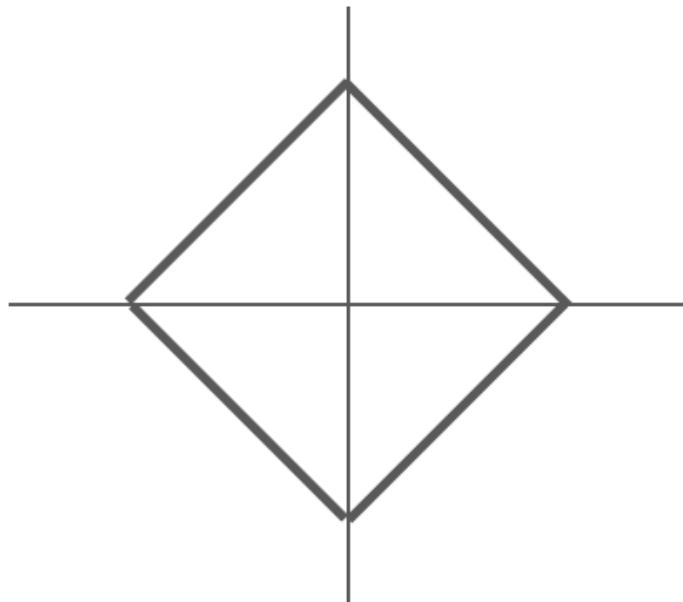


$K = 5$

# K-Nearest Neighbors: Distance Metric

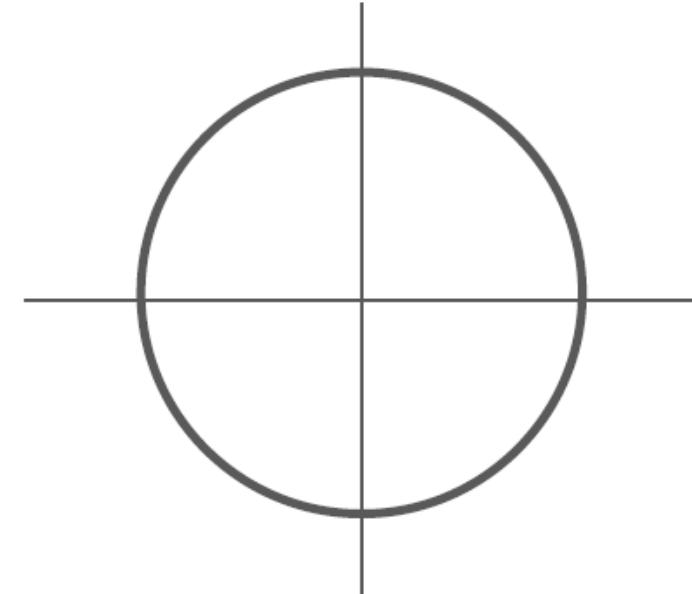
L1 (Manhattan) distance

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



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

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

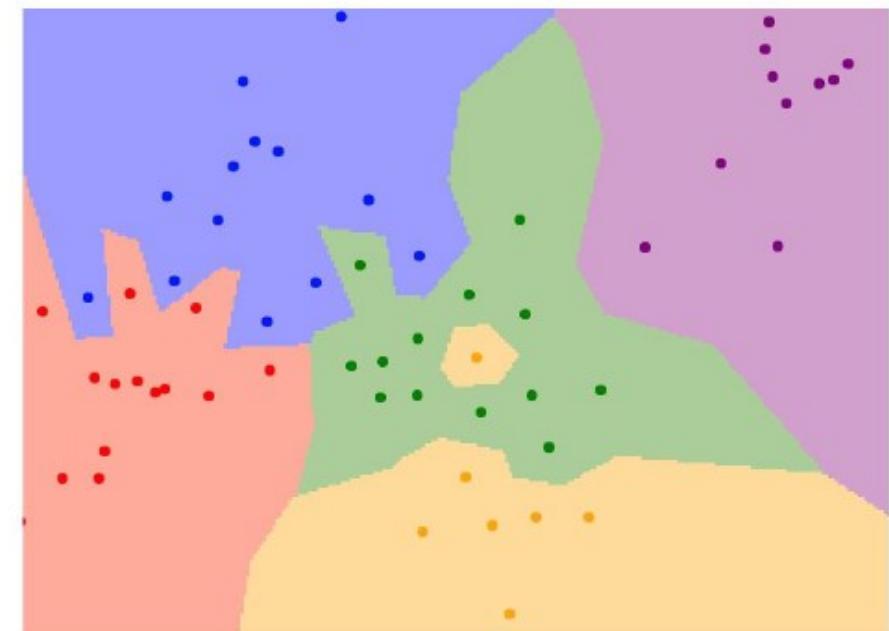


$K = 1$

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

L2 (Euclidean) distance

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



$K = 1$

# Hyperparameters

What is the best value of **k** to use?

What is the best **distance** to use?

These are **hyperparameters**: choices about the algorithm that we set rather than learn

Very problem-dependent.

Must try them all out and see what works best.

# K-fold Cross-Validation

Your Dataset

**Idea #4: Cross-Validation:** Split data into **folds**,  
try each fold as validation and average the results

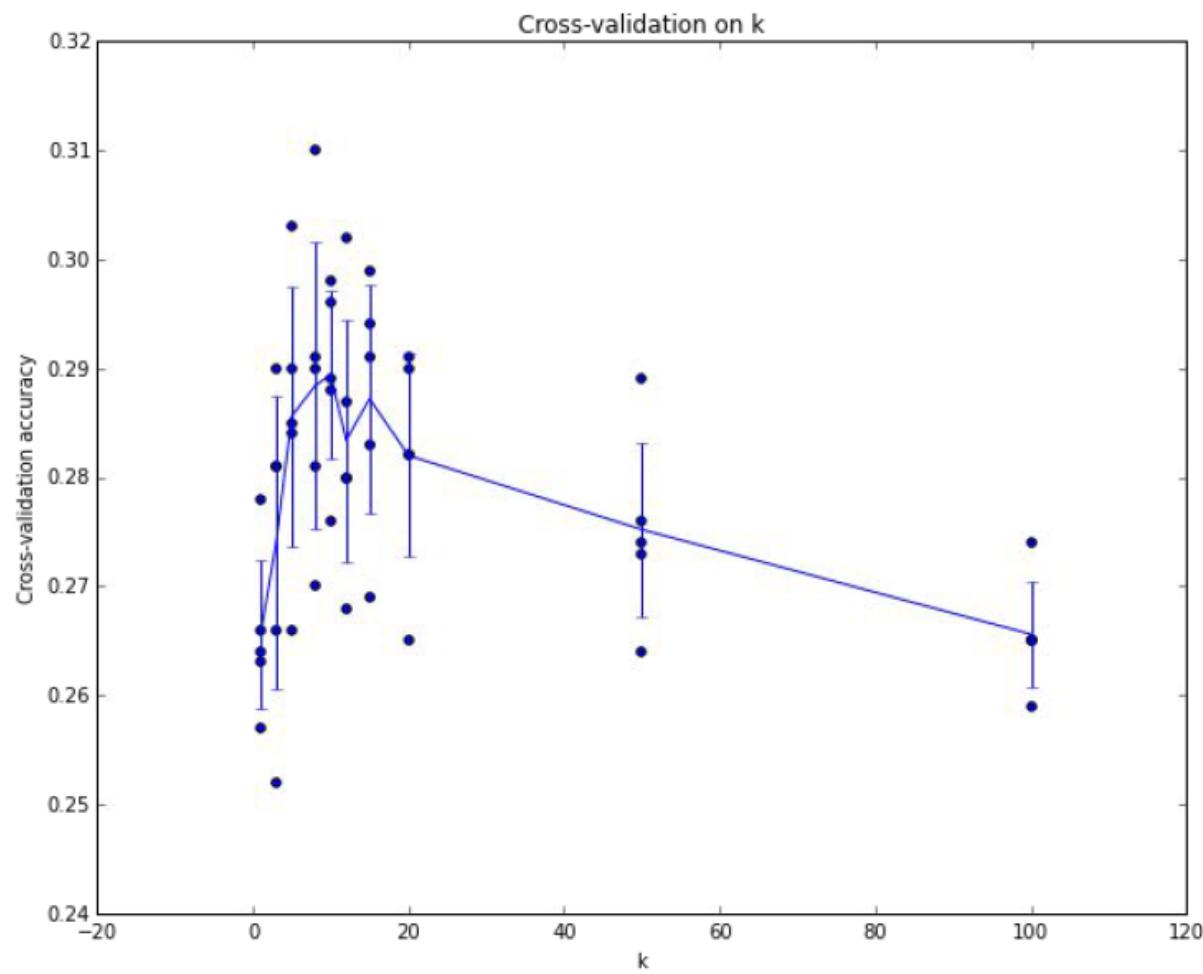
fold 1	fold 2	fold 3	fold 4	fold 5	test
--------	--------	--------	--------	--------	------

fold 1	fold 2	fold 3	fold 4	fold 5	test
--------	--------	--------	--------	--------	------

fold 1	fold 2	fold 3	fold 4	fold 5	test
--------	--------	--------	--------	--------	------

Useful for small datasets, but not used too frequently in deep learning

# Setting Hyperparameters



Example of  
5-fold cross-validation  
for the value of **k**.

Each point: single  
outcome.

The line goes  
through the mean, bars  
indicated standard  
deviation

(Seems that  $k \approx 7$  works best  
for this data)

# 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)

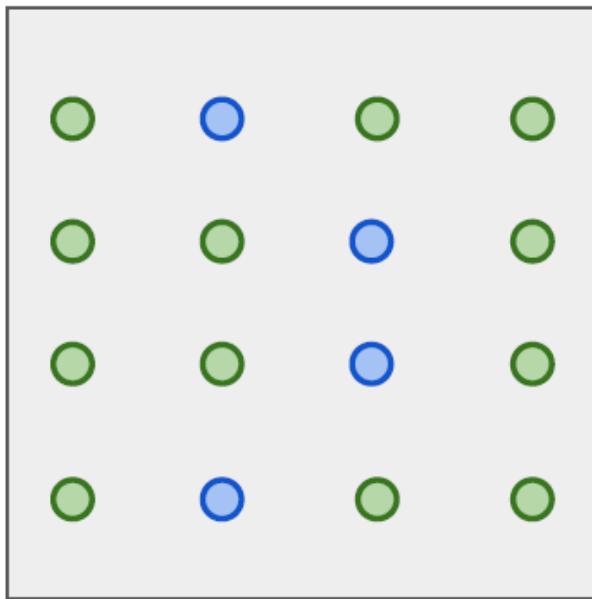
# Curse of Dimensionality

- Curse of dimensionality

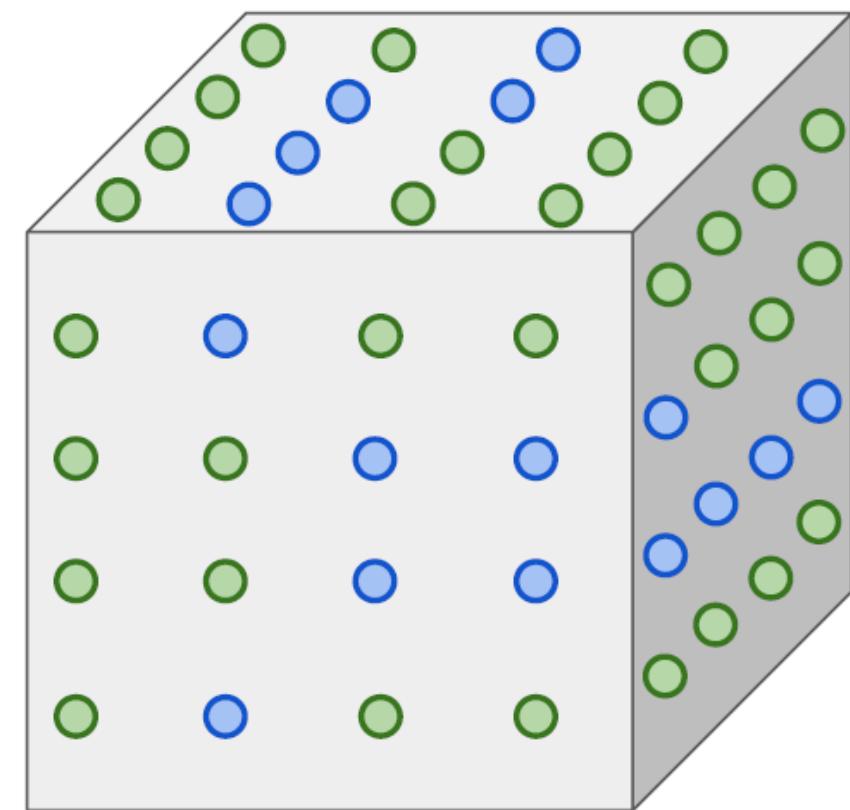
Dimensions = 1  
Points = 4



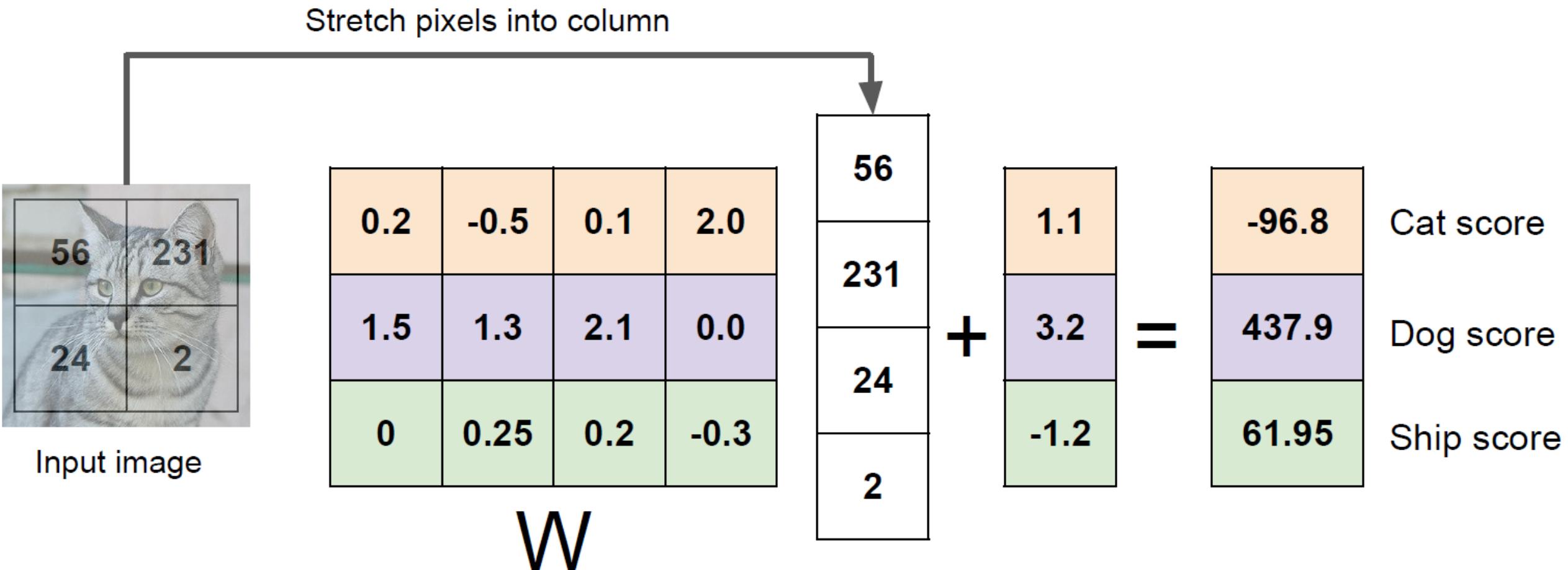
Dimensions = 2  
Points =  $4^2$



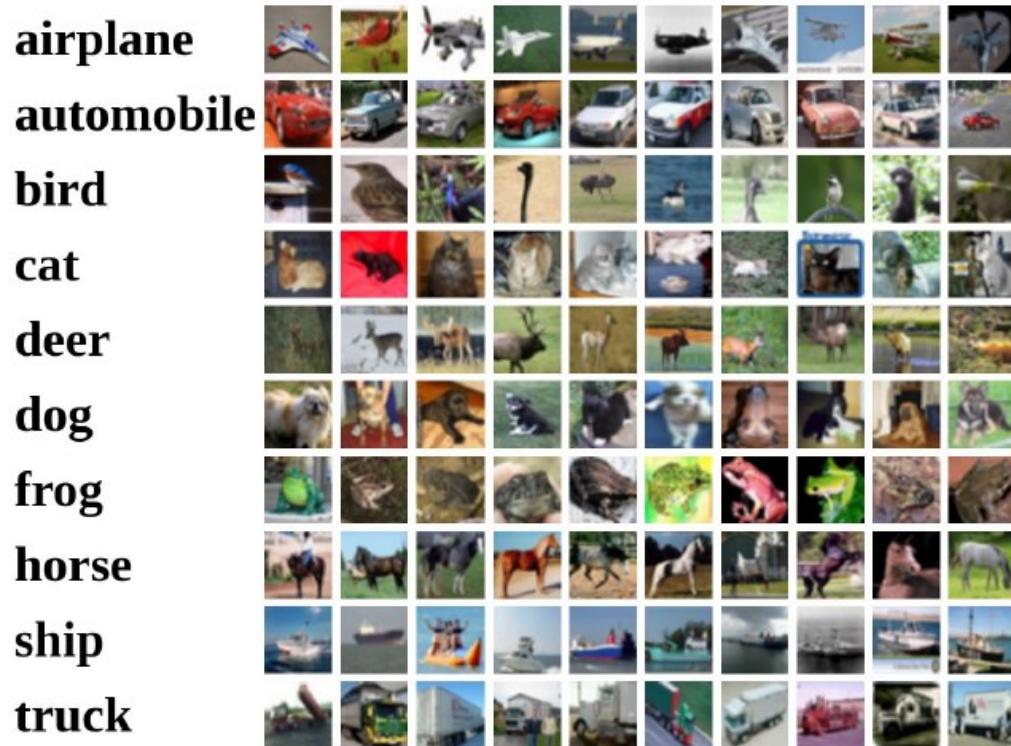
Dimensions = 3  
Points =  $4^3$



# Linear Classifier



# Interpreting a Linear Classifier

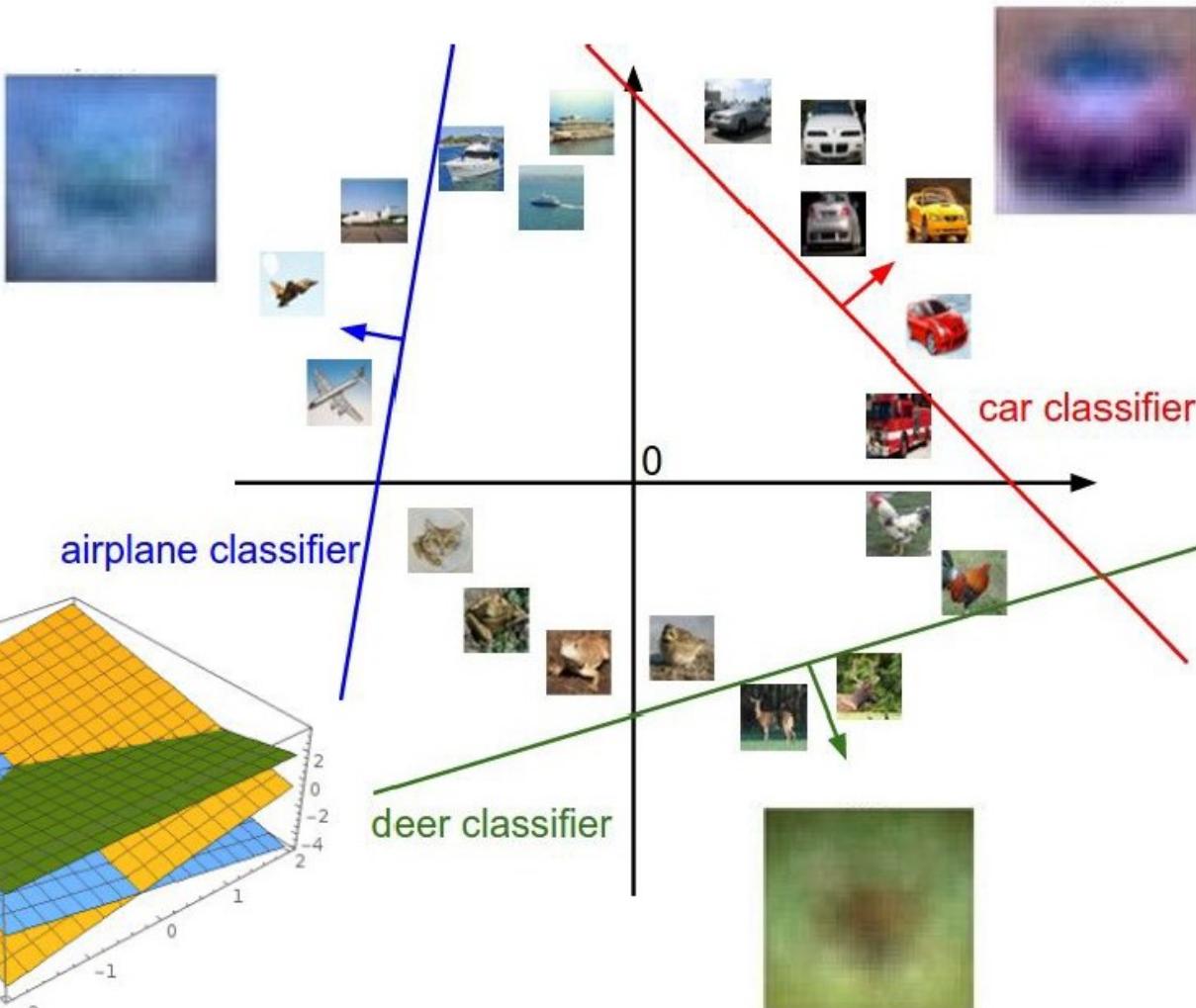


$$f(x, W) = Wx + b$$

Example trained weights  
of a linear classifier  
trained on CIFAR-10:



# Interpreting a Linear Classifier



$$f(x, W) = Wx + b$$



Array of **32x32x3** numbers  
(3072 numbers total)

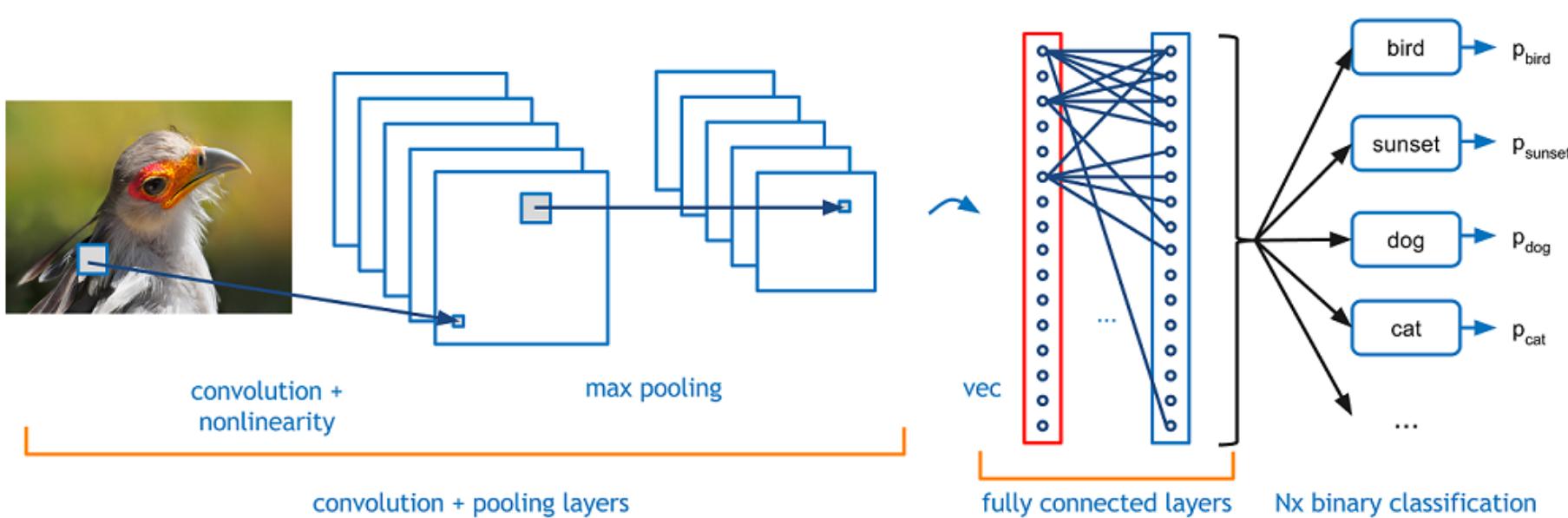
# Convolutional Neural Network



What We See

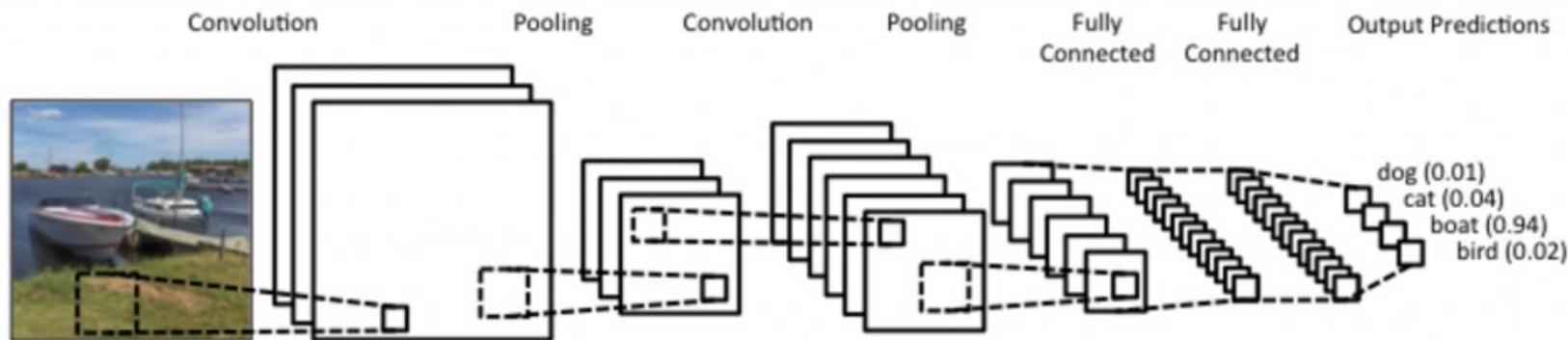
08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08  
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00  
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65  
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91  
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80  
24 47 32 60 99 03 45 02 44 75 33 53 78 36 80 20 35 17 12 50  
32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70  
67 26 20 68 02 62 12 20 99 63 94 39 63 08 40 91 66 49 94 21  
24 55 58 05 66 73 99 24 97 17 78 78 96 83 14 88 34 89 63 72  
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95  
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92  
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57  
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58  
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40  
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66  
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69  
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36  
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16  
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54  
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48

What Computers See



# Convolutional Neural Network

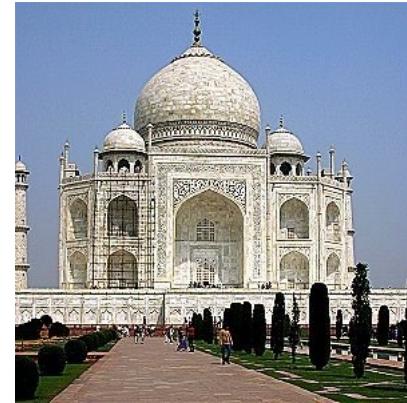
- 이미지 인식에 가장 널리 사용됨
- 일반적으로 convolution layer, pooling layer, fully-connected layer로 구성
- Parameter(weight) sharing
- Convolution과 pooling layer는 feature를 추출하고 fully-connected layer는 어떤 class에 속하는지 판단하는 역할을 수행



# Convolution Filters(Hand Crafted)



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



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



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



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



# Let's Try!

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

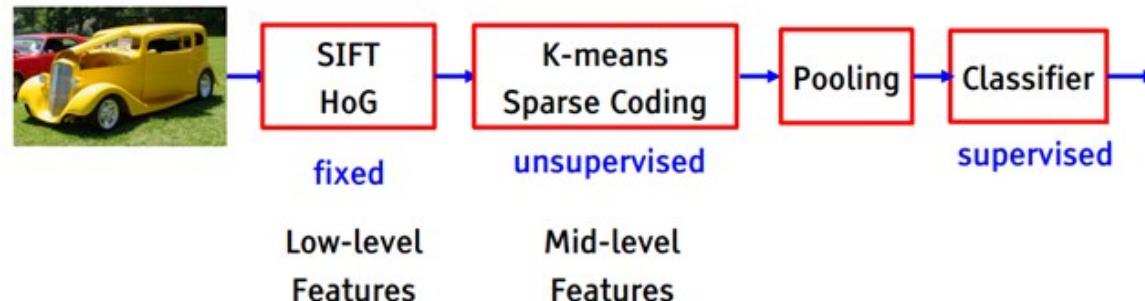
0	-1	0
-1	5	-1
0	-1	0

sharpen ▾

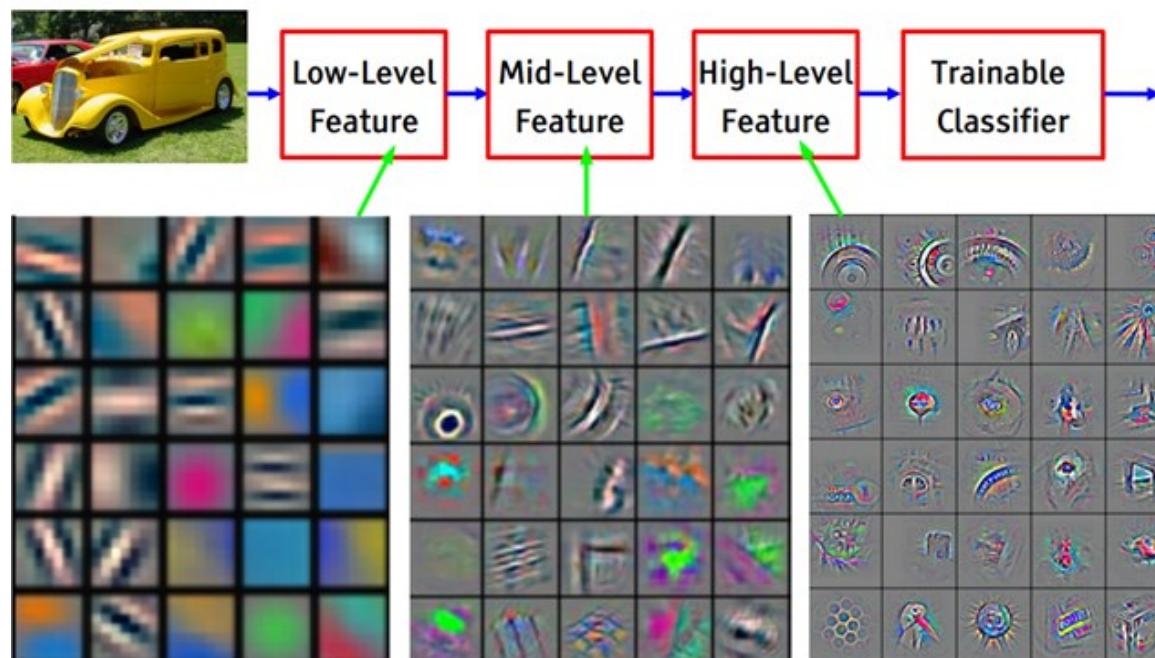


# Before and After

Object recognition 2006-2012

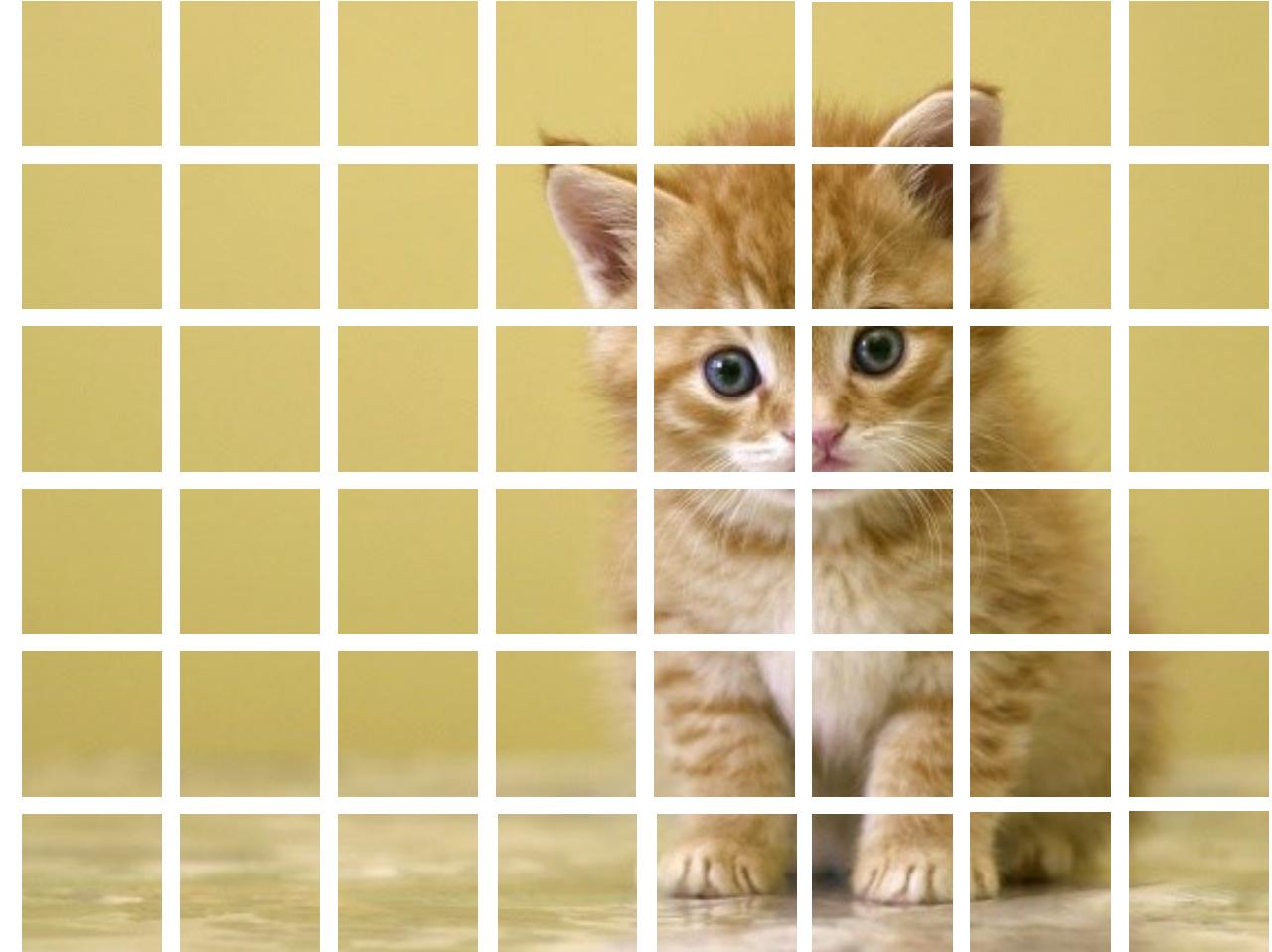


State of the art object recognition using CNNs



# CNN 동작원리

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

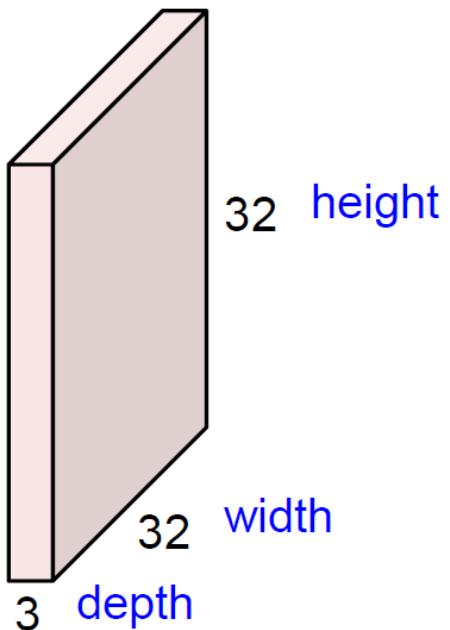


# Convolution Layer

- From Stanford CS231n

## Convolution Layer

32x32x3 image

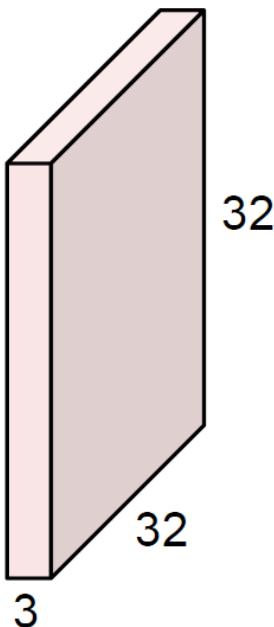


# Convolution Layer

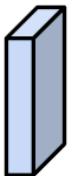
- From Stanford CS231n

## Convolution Layer

32x32x3 image



5x5x3 filter

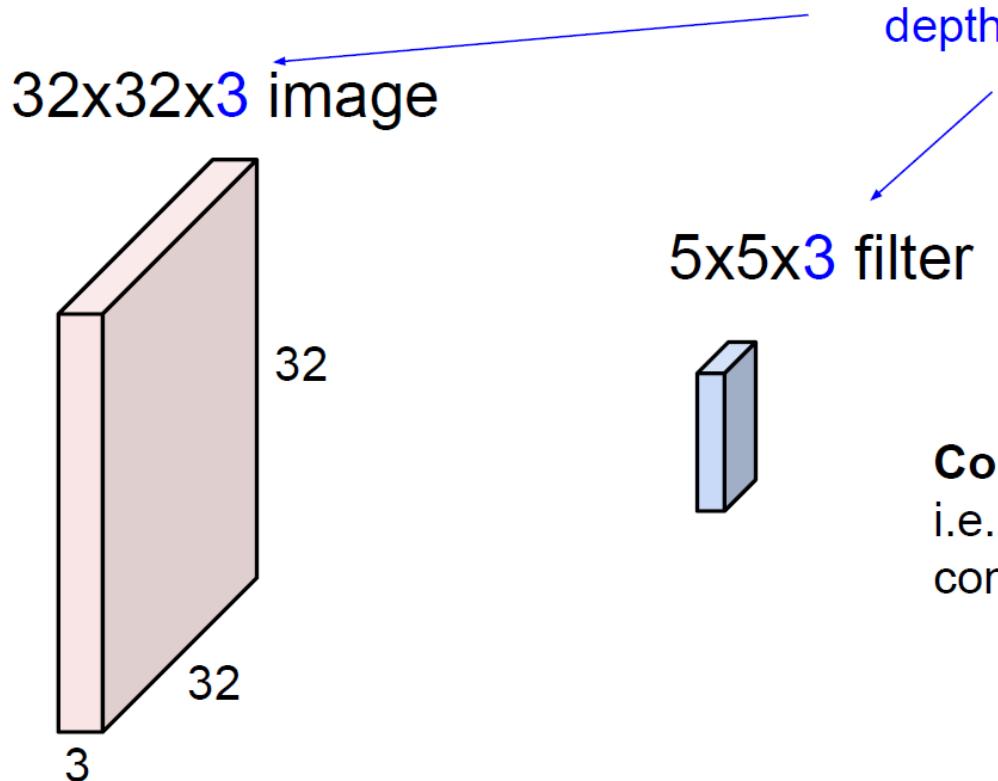


**Convolve** the filter with the image  
i.e. “slide over the image spatially,  
computing dot products”

# Convolution Layer

- From Stanford CS231n

## Convolution Layer



Filters always extend the full depth of the input volume

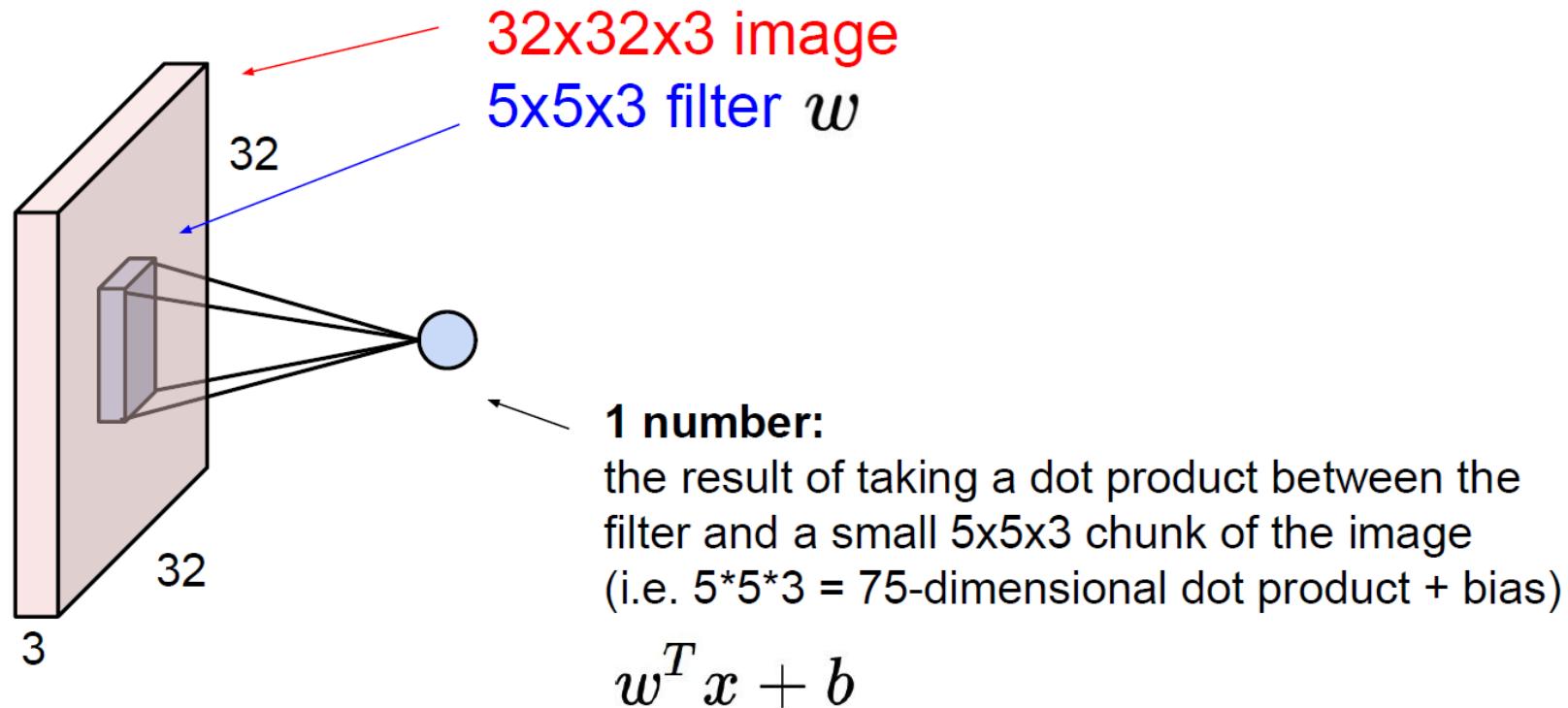
5x5x3 filter

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

# Convolution Layer

- From Stanford CS231n

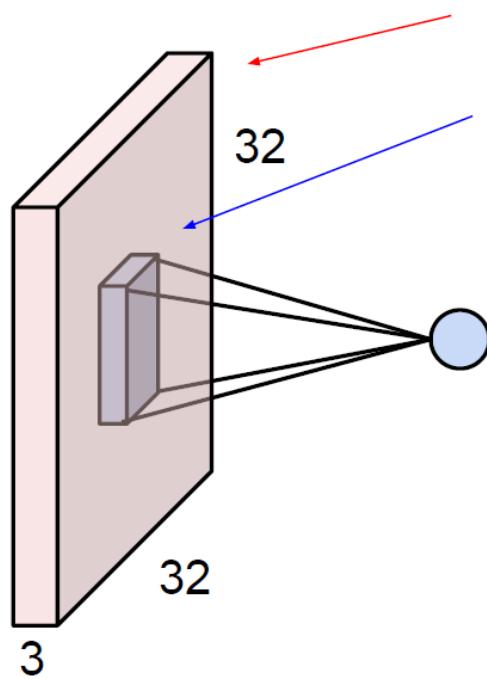
## Convolution Layer



# Convolution Layer

- From Stanford CS231n

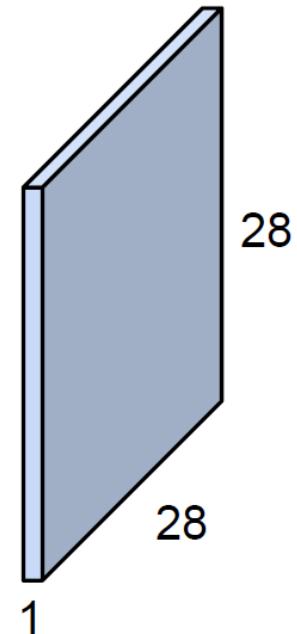
## Convolution Layer



32x32x3 image  
5x5x3 filter

convolve (slide) over all  
spatial locations

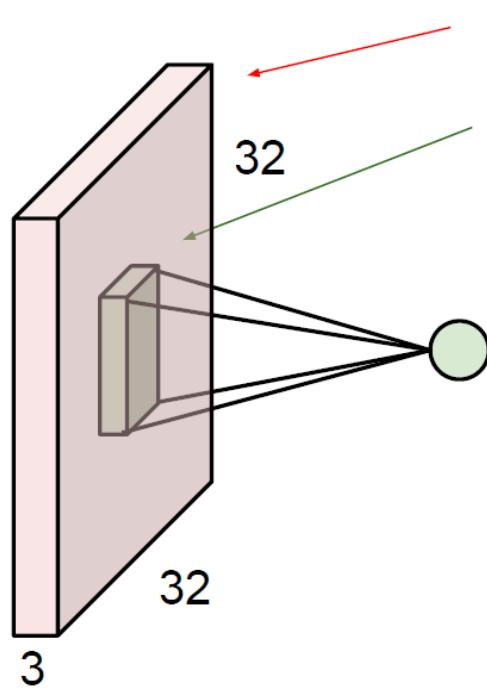
activation map



# Convolution Layer

- From Stanford CS231n

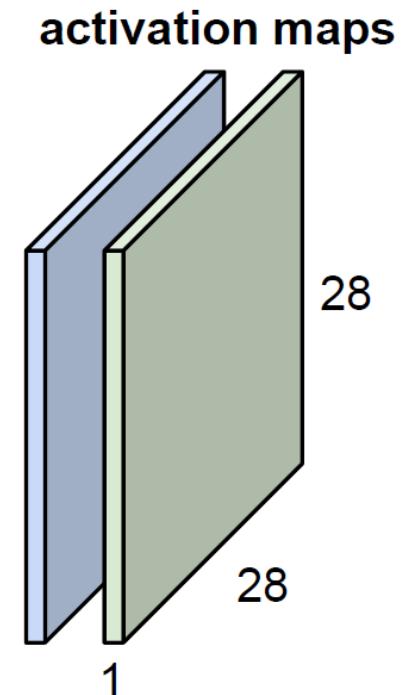
## Convolution Layer



32x32x3 image  
5x5x3 filter

convolve (slide) over all  
spatial locations

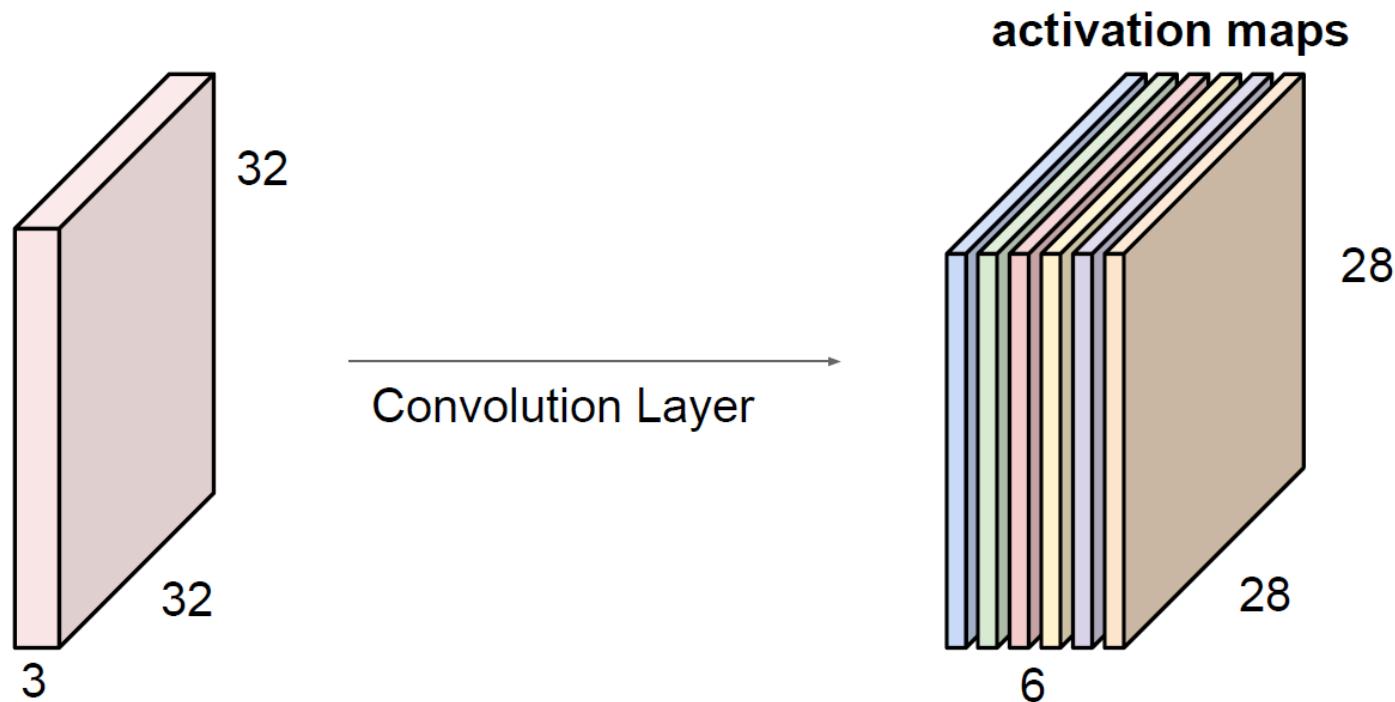
consider a second, green filter



# Convolution Layer

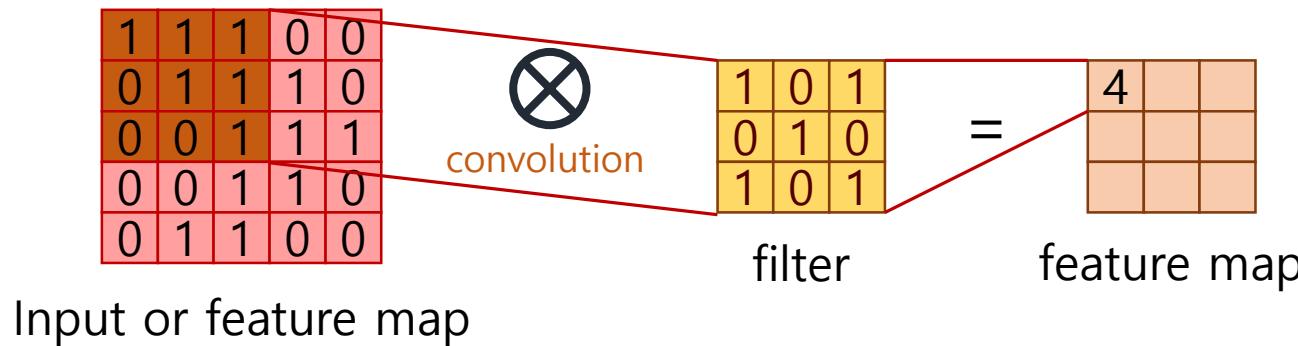
- From Stanford CS231n

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

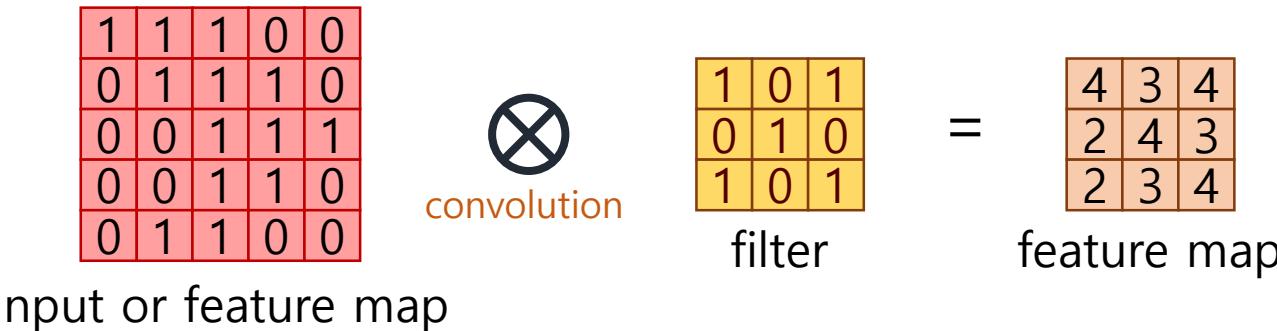


We stack these up to get a “new image” of size  $28 \times 28 \times 6$ !

# Convolution



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



# Convolution

1 <small>x1</small>	1 <small>x0</small>	1 <small>x1</small>	0	0
0 <small>x0</small>	1 <small>x1</small>	1 <small>x0</small>	1	0
0 <small>x1</small>	0 <small>x0</small>	1 <small>x1</small>	1	1
0	0	1	1	0
0	1	1	0	0

Image

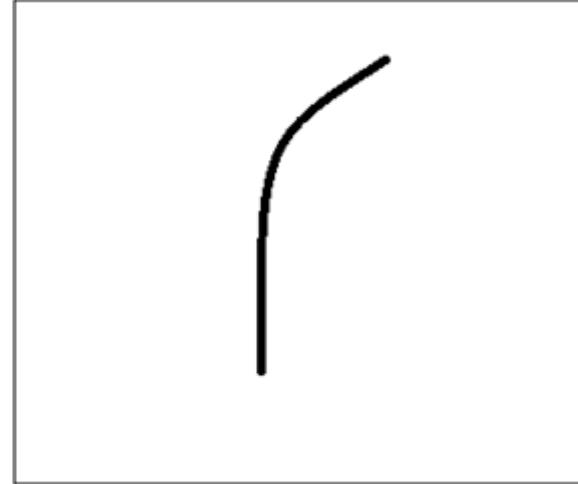
4		

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
0	0	0	0	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)

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

⊗  
convolution

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

0	1	0	0	0	0
1	0	1	0	0	0
0	-1	0	0	0	1

Input channel : 3

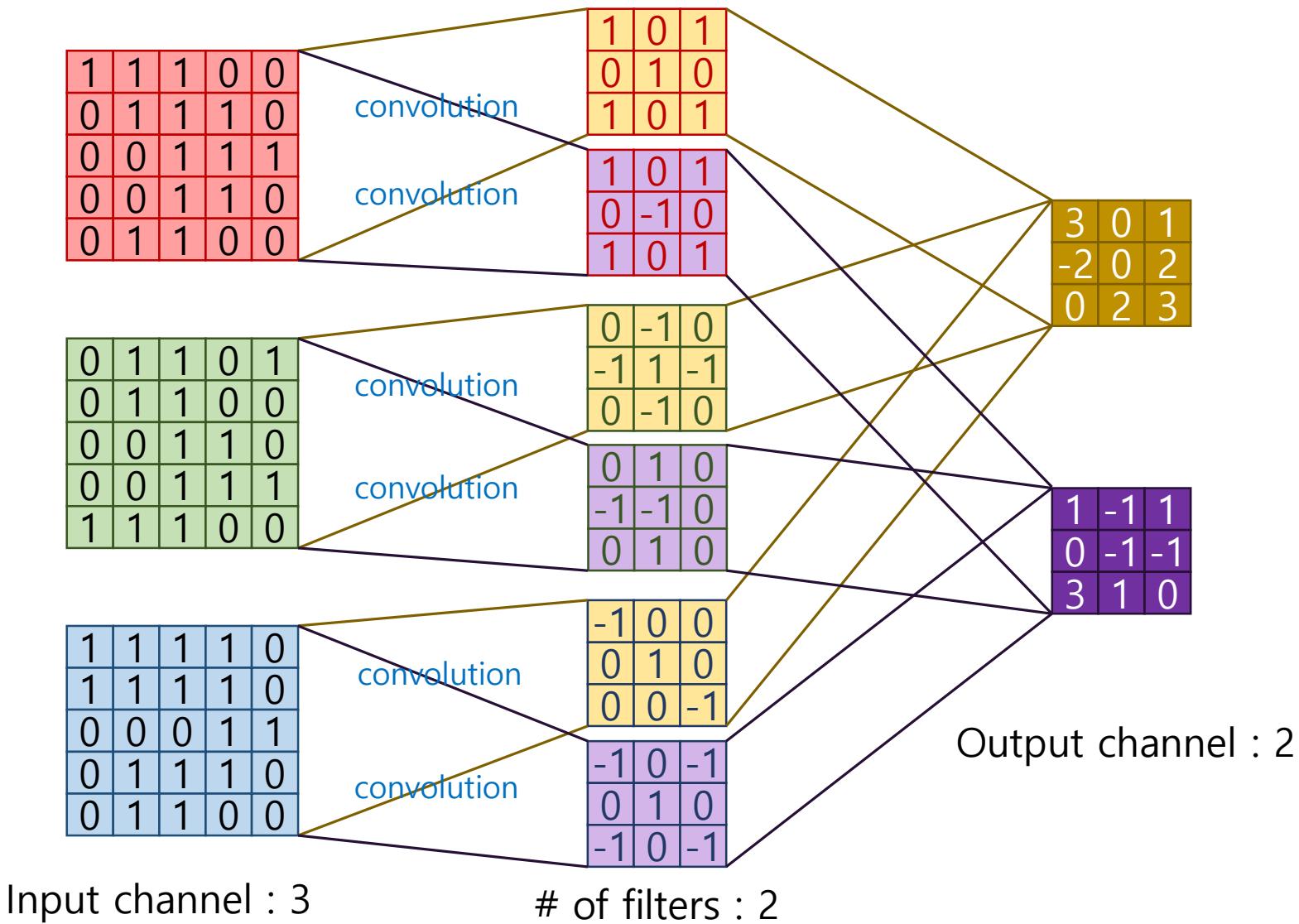
# of filters : 2

=

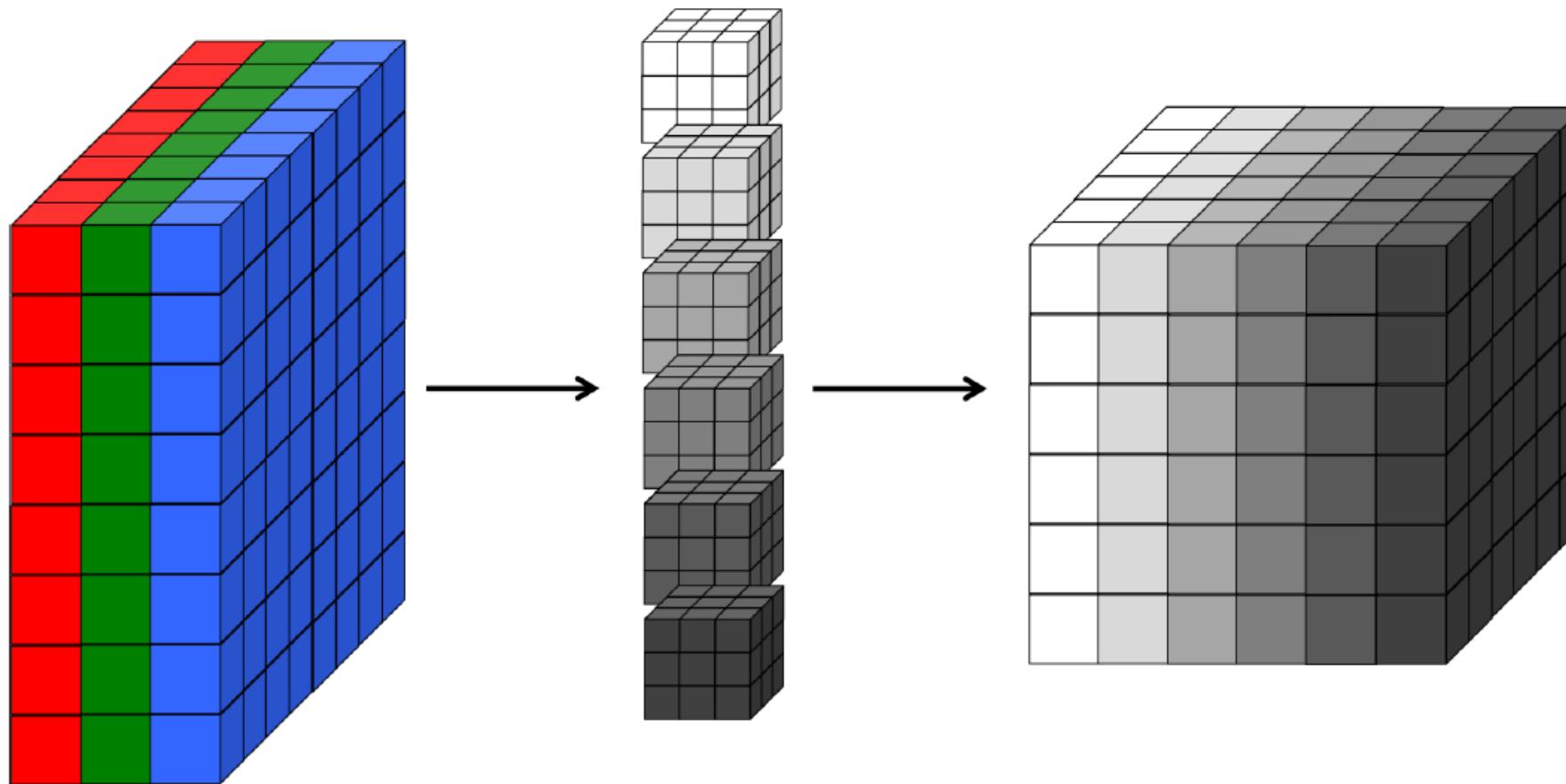
3	1	1	1	1	1
-2	0	2	2	0	1
0	2	3	0	0	0

Output channel : 2

# Convolution (Multi Channel, Many Filters)

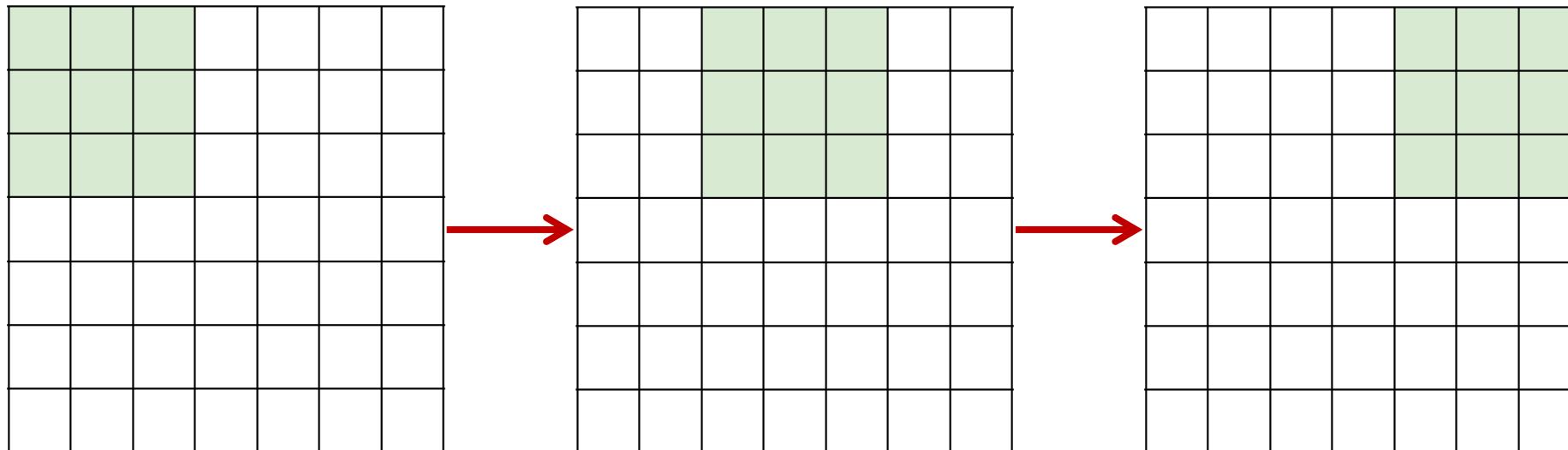


# Visualization of a Convolution Layer



# 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  $F \times F$ , 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

# 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.

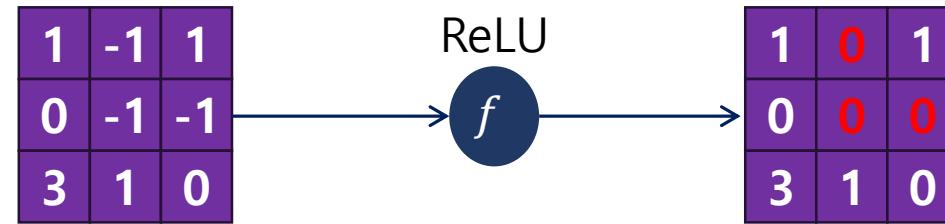
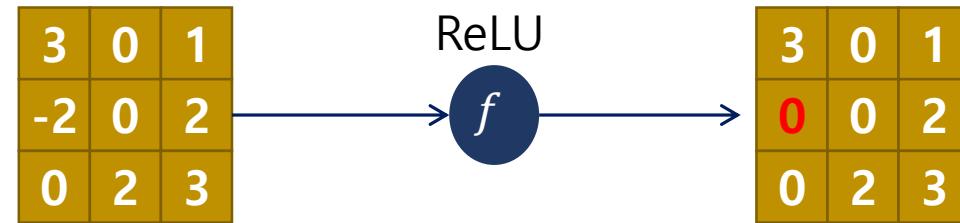
Given an input tensor of shape

'[batch, in\_height, in\_width, in\_channels]'

and a filter / kernel tensor of shape

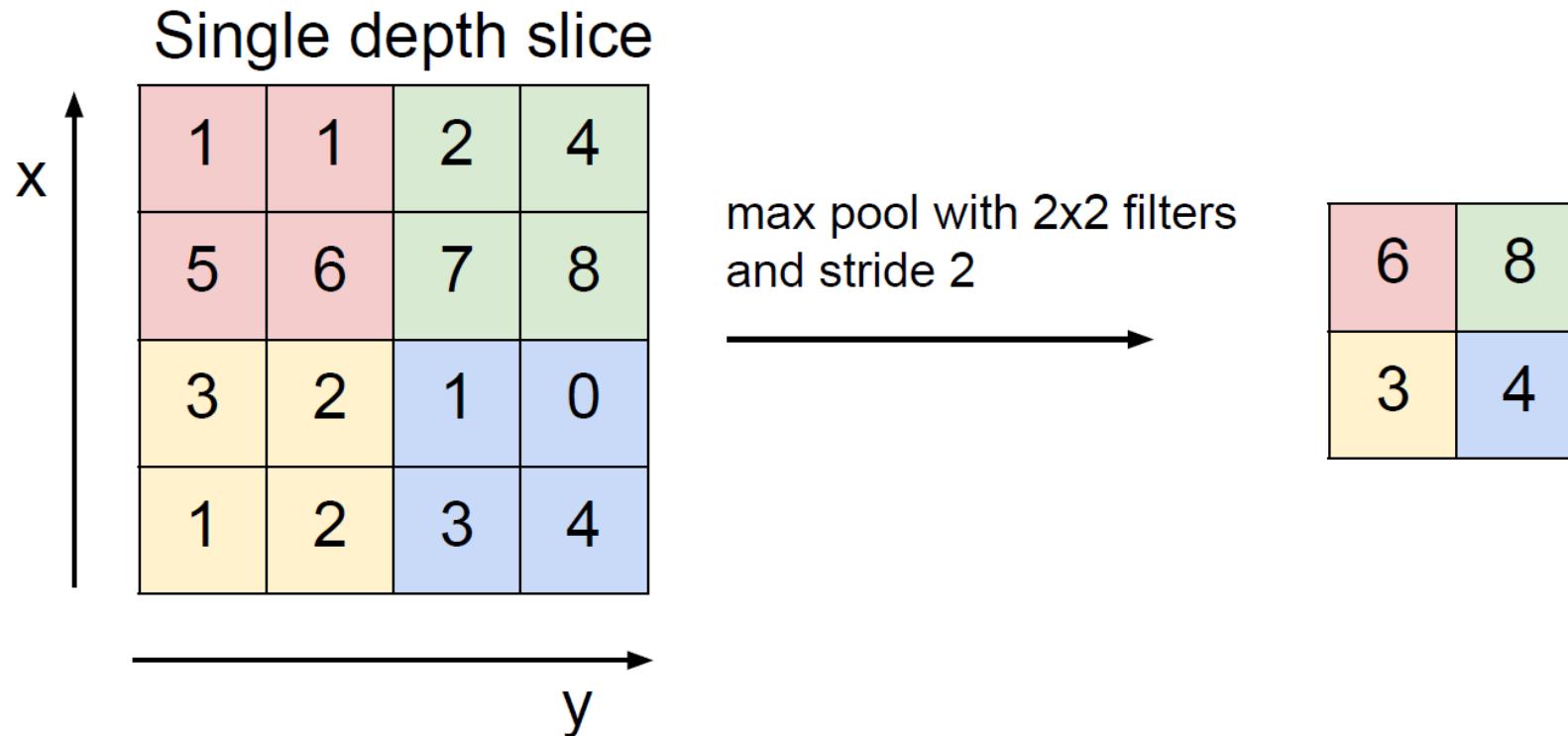
'[filter\_height, filter\_width, in\_channels, out\_channels]'

# ReLU

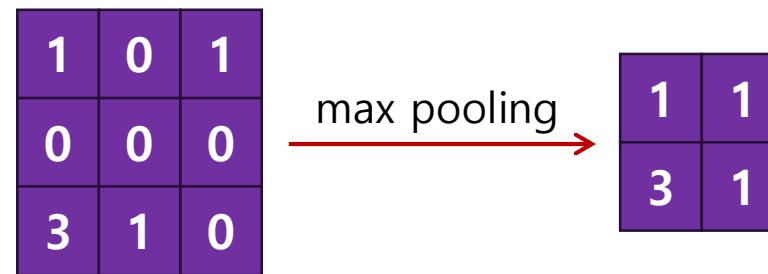
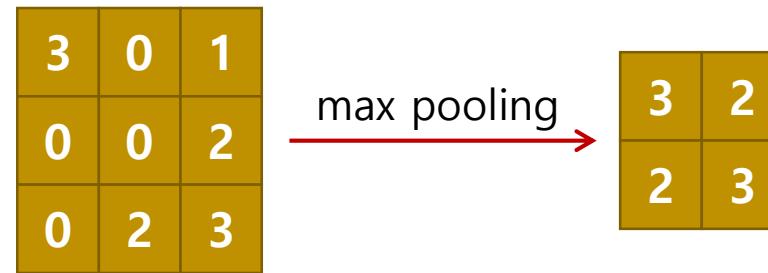


# Pooling Layer

- Max pooling을 많이 사용함



# 2x2 Max Pooling with Stride=1



# tf.nn.max\_pool

- **Signature:**

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

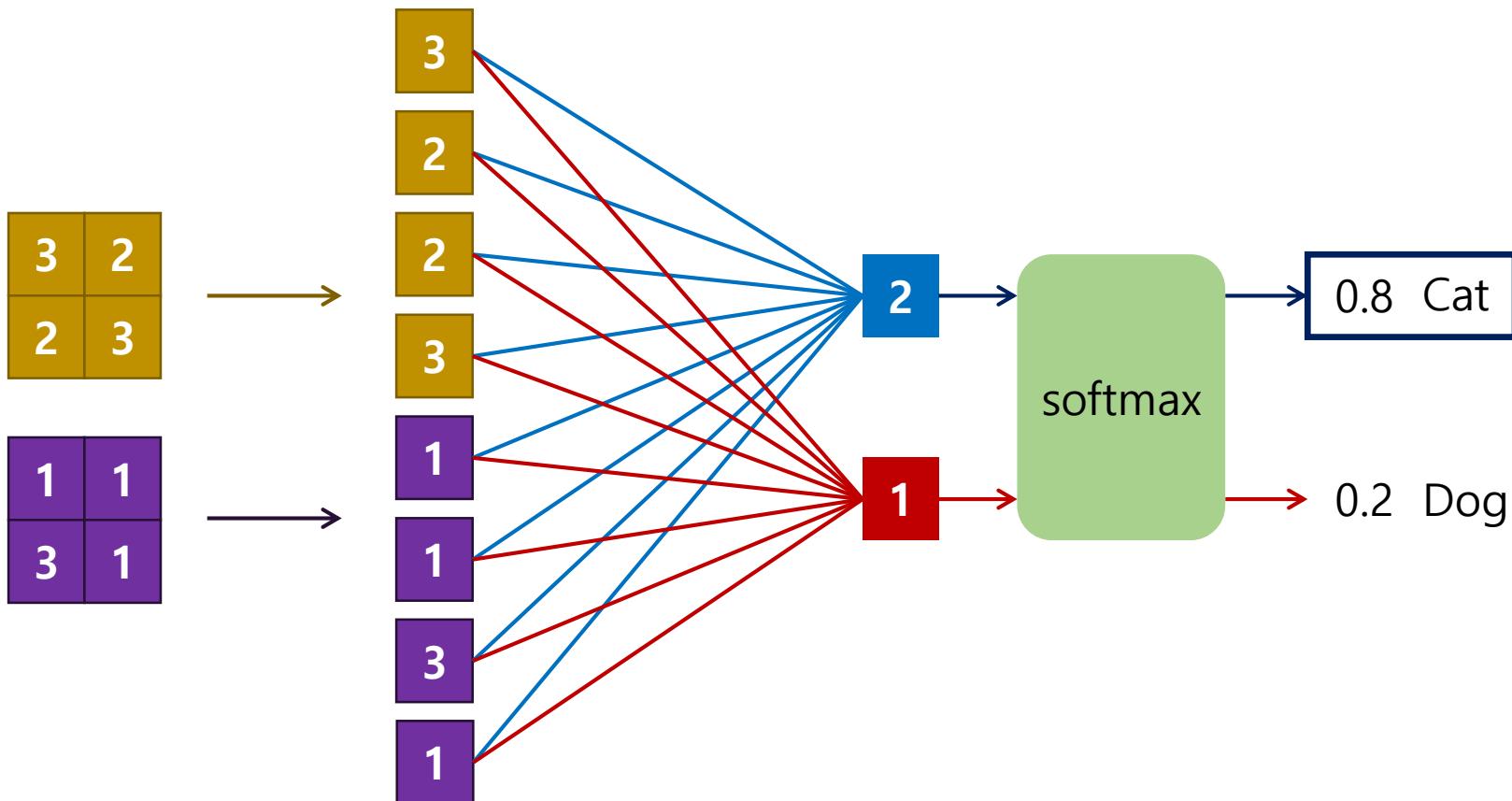
- **Docstring:**

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  $\geq 4$ . The size of the window for each dimension of the input tensor.
- strides: A list of ints that has length  $\geq 4$ . 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...