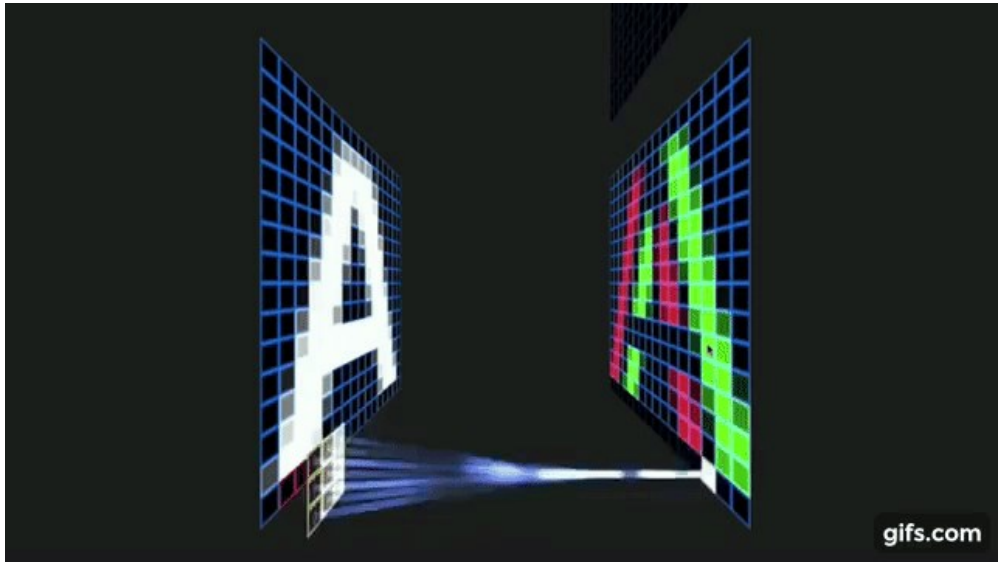
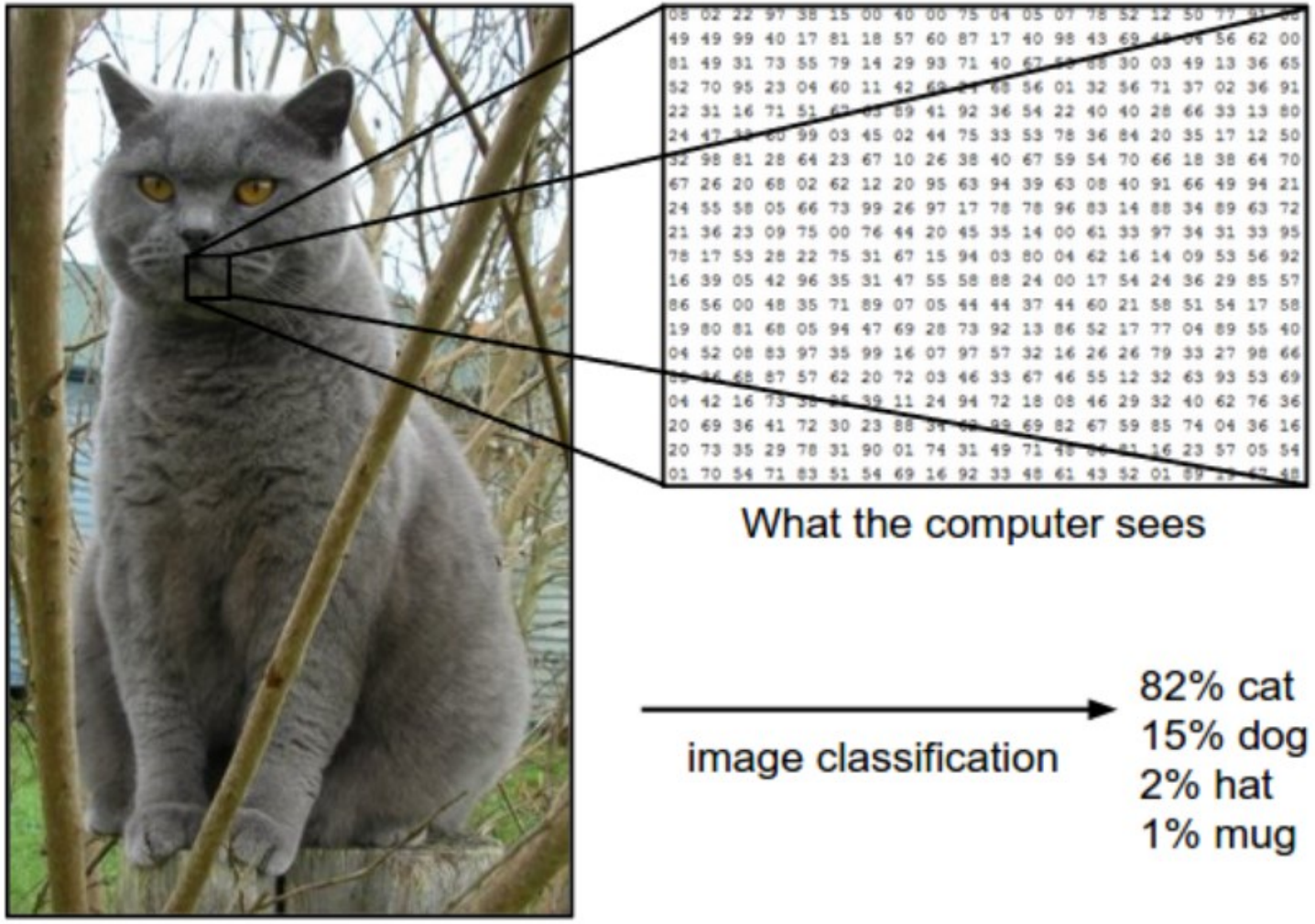
Convolutional Neural Network



Fast Campus
Start Deep Learning with TensorFlow

Image Classification

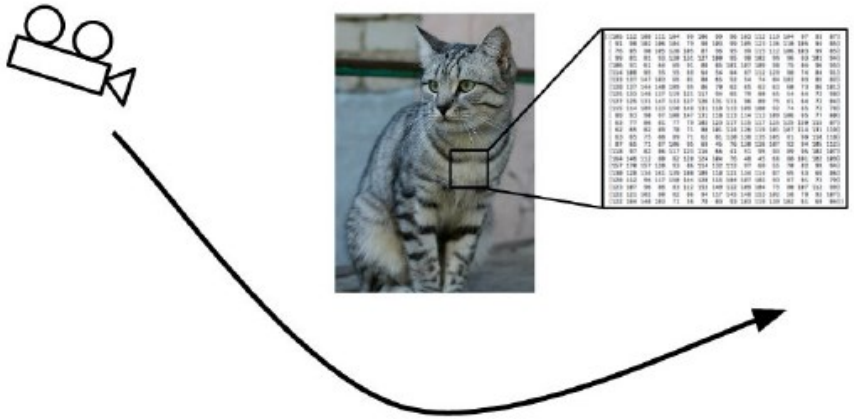
- A core task in Computer Vision



Slide Credit : Stanford CS231n

Challenges of Recognition

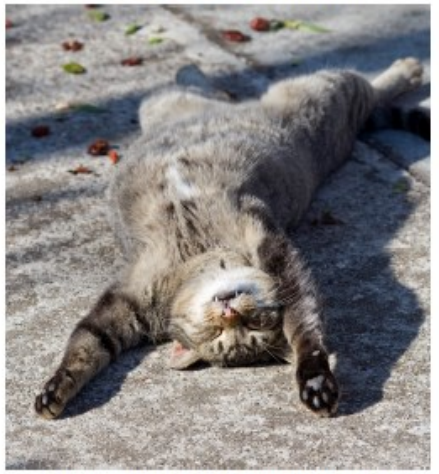
Viewpoint



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

An Image Classifier

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

Unlike e.g. sorting a list of numbers,

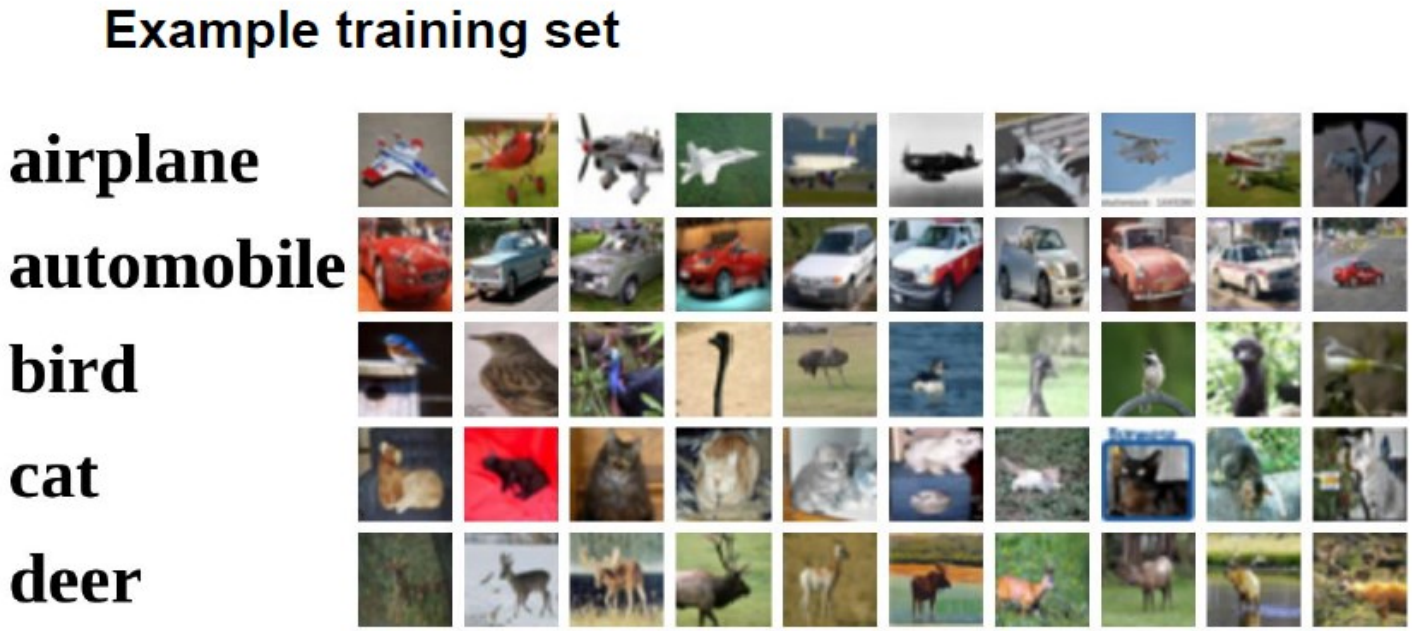
no obvious way to hard-code the algorithm for recognizing a cat, or other classes.

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

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

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



First Classifier – Nearest Neighbor

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

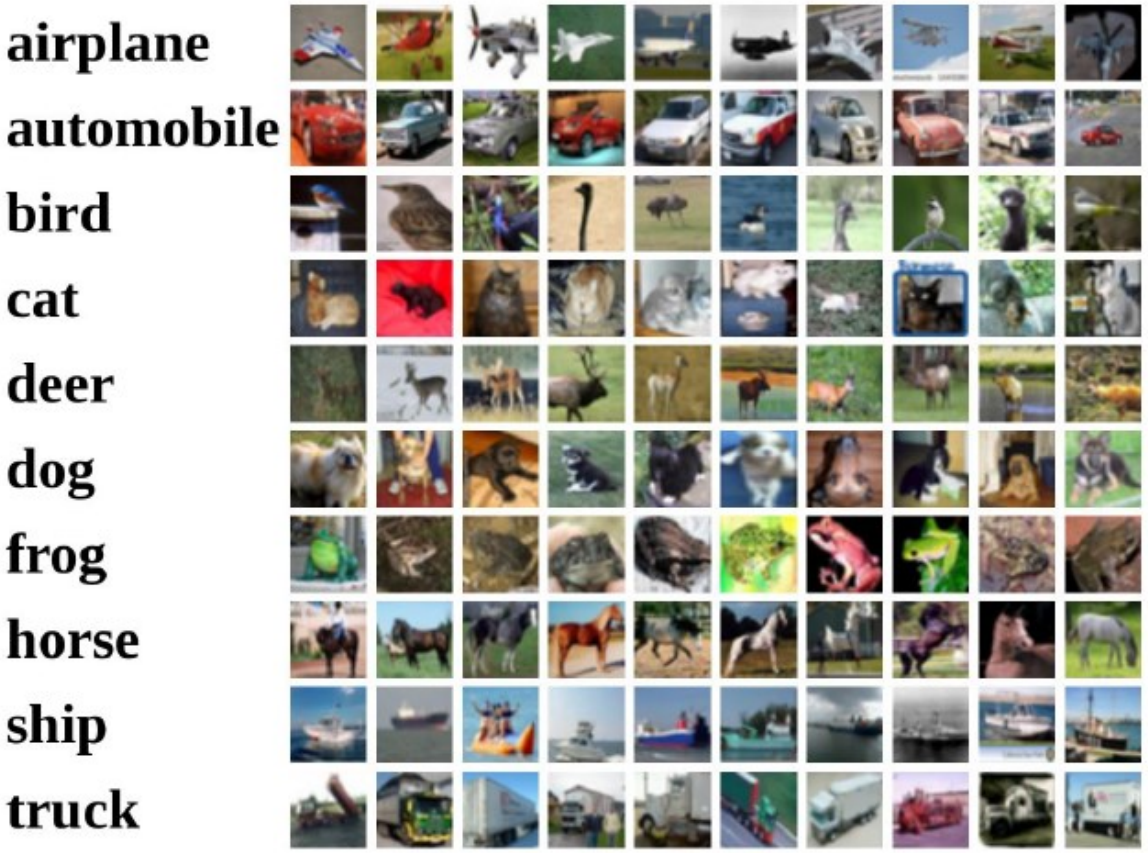
Memorize all data and labels

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

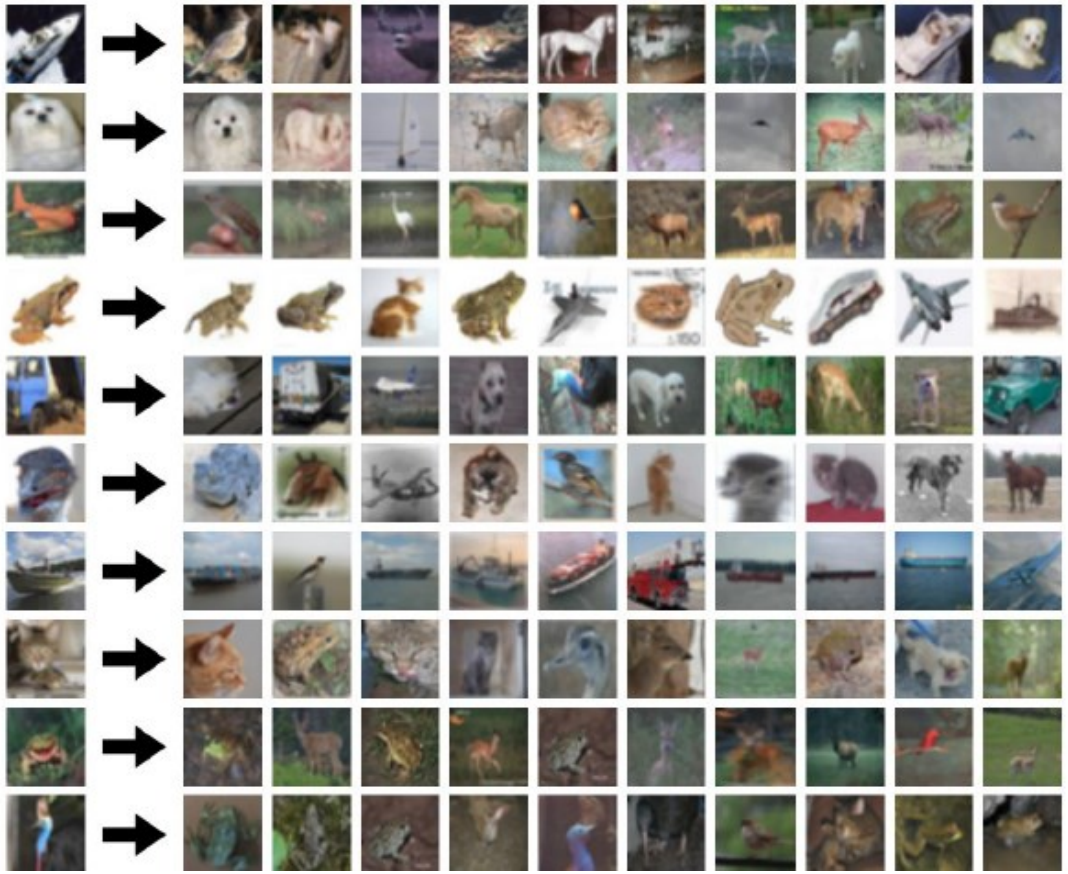
Predict the label of the most similar training image

Example Dataset: CIFAR10

10 classes
50,000 training images
10,000 testing images

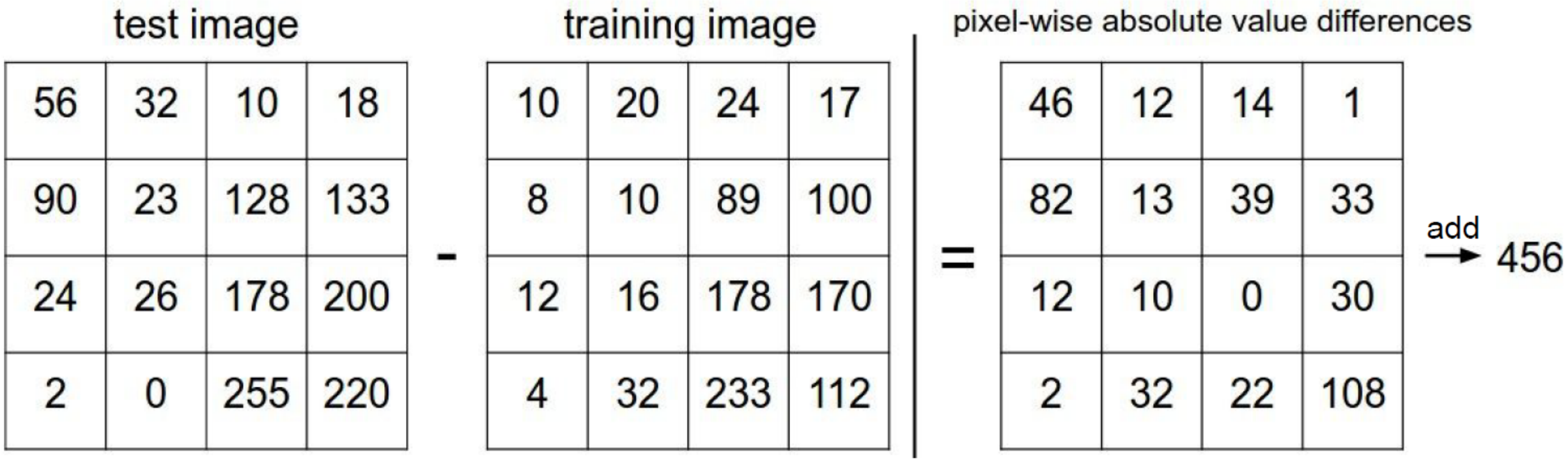


Test images and nearest neighbors



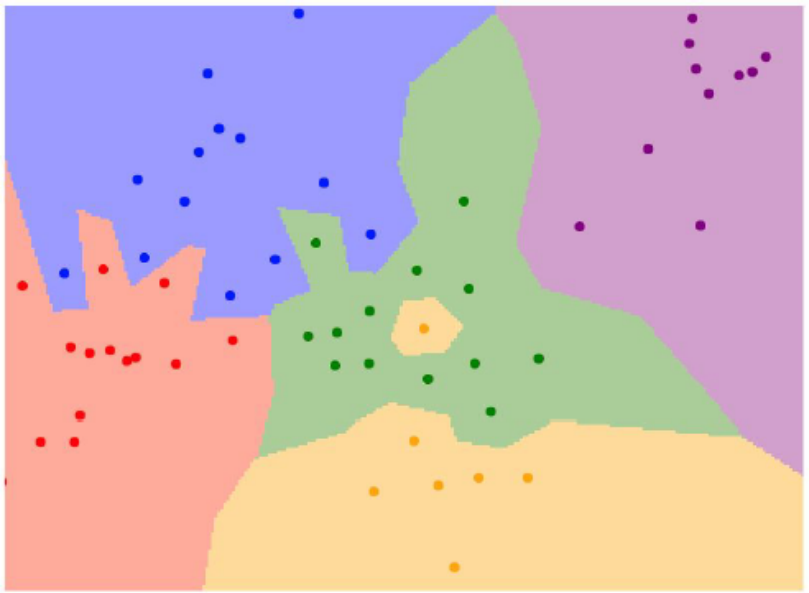
Distance Metric

L1 distance: $$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

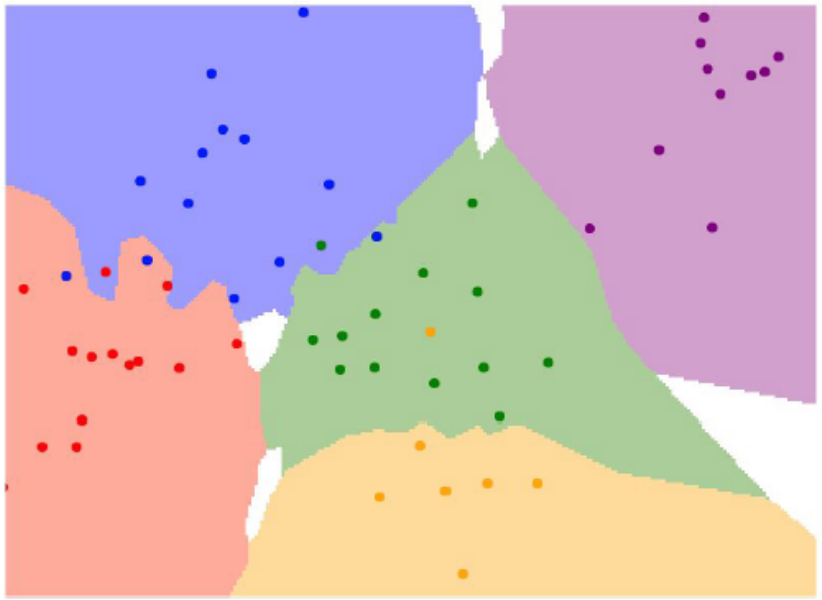


K-Nearest Neighbors

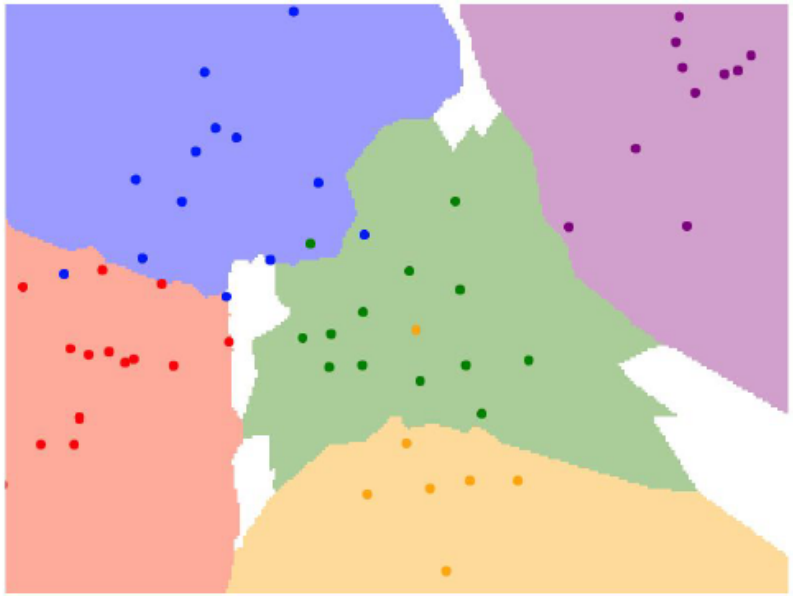
Instead of copying label from nearest neighbor,
take **majority vote** from K closest points



K = 1



K = 3

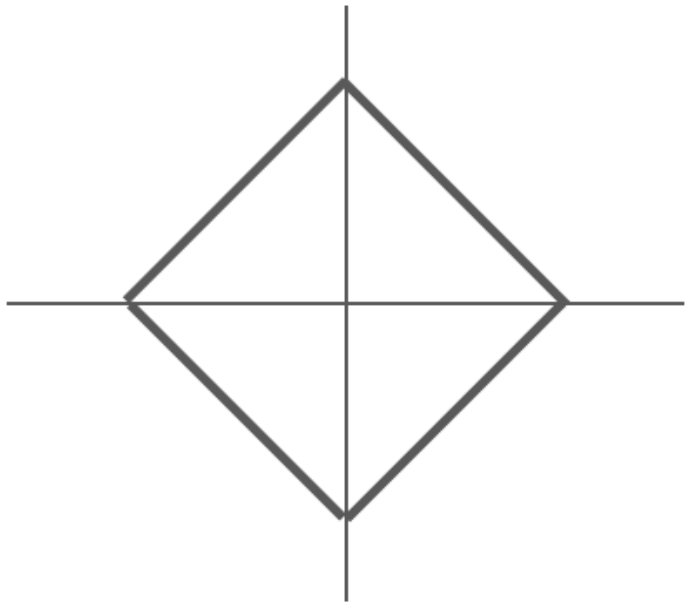


K = 5

K-Nearest Neighbors: Distance Metric

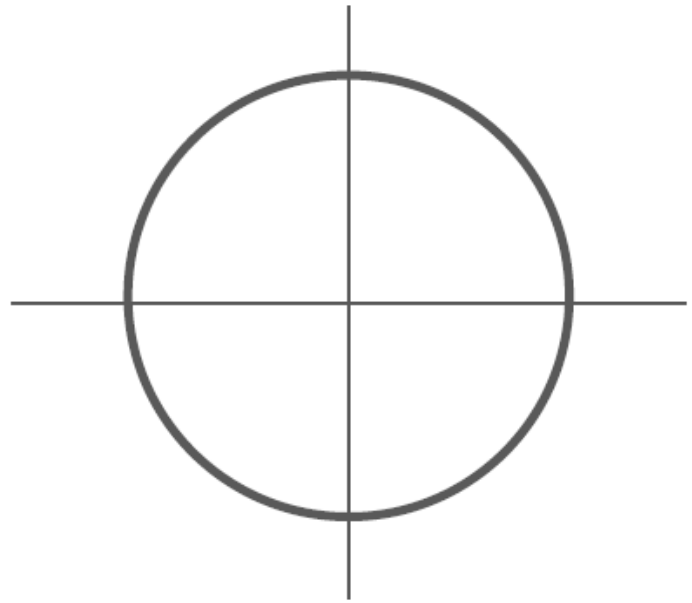
L1 (Manhattan) distance

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



L2 (Euclidean) distance

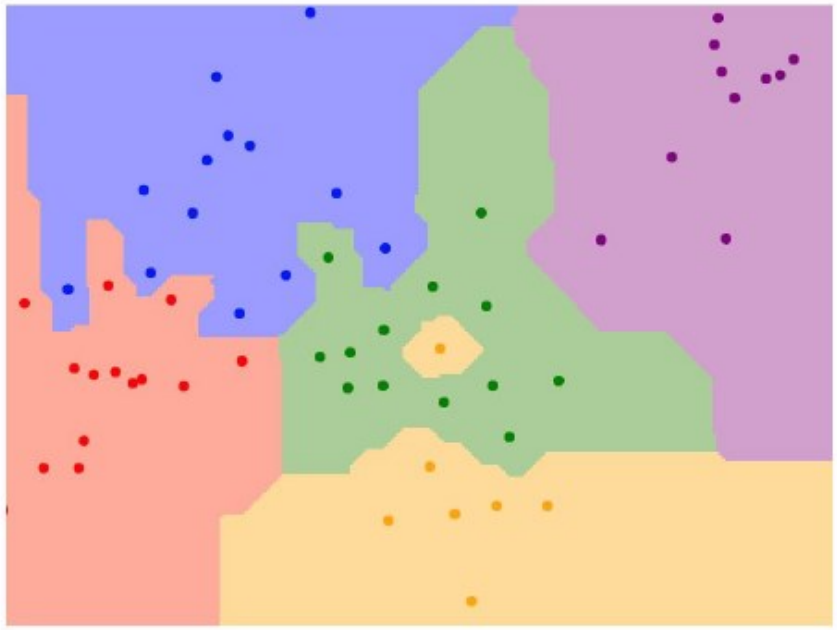
$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$



K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

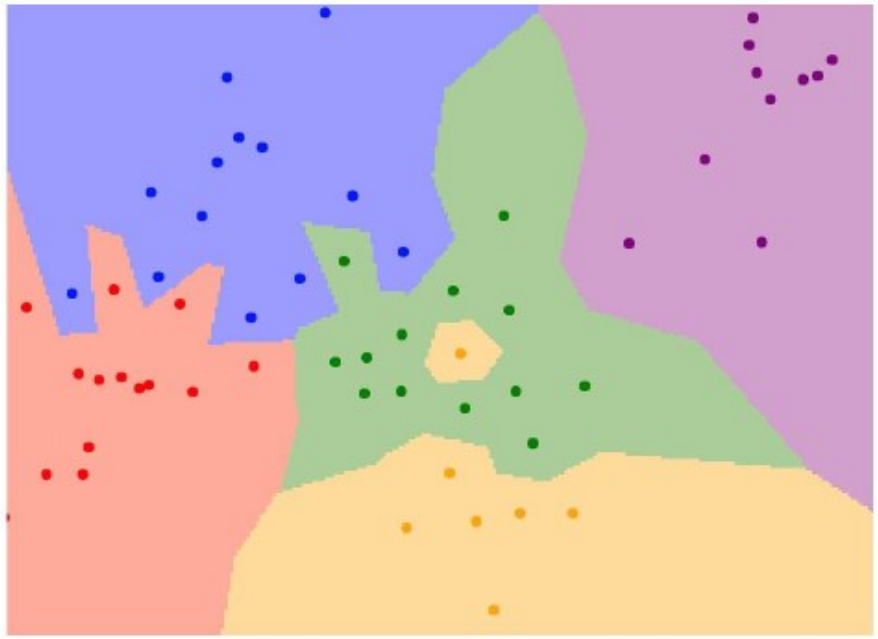
$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



K = 1

L2 (Euclidean) distance

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



K = 1

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

K-Nearest Neighbor on Images NEVER Used

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

Original

Boxed

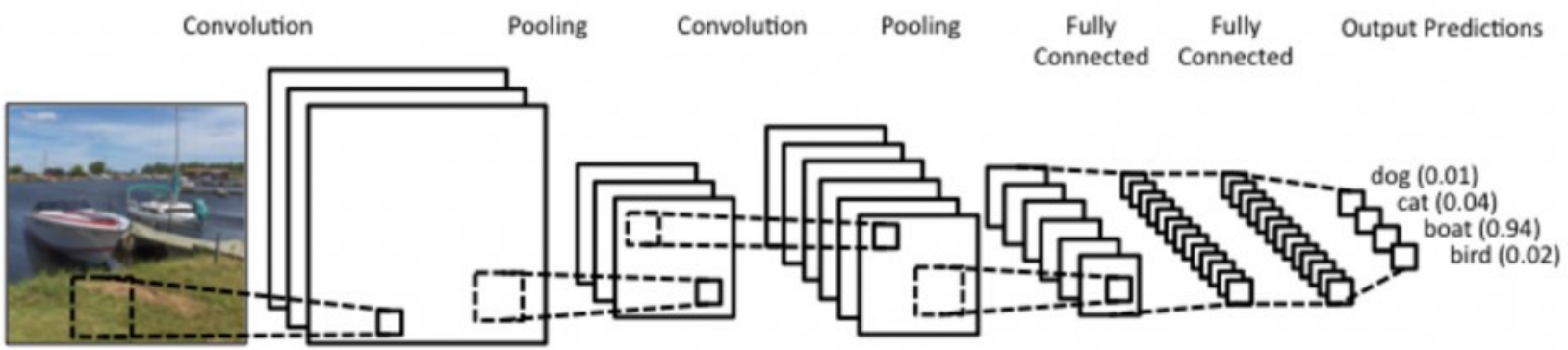
Shifted

Tinted

(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)

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

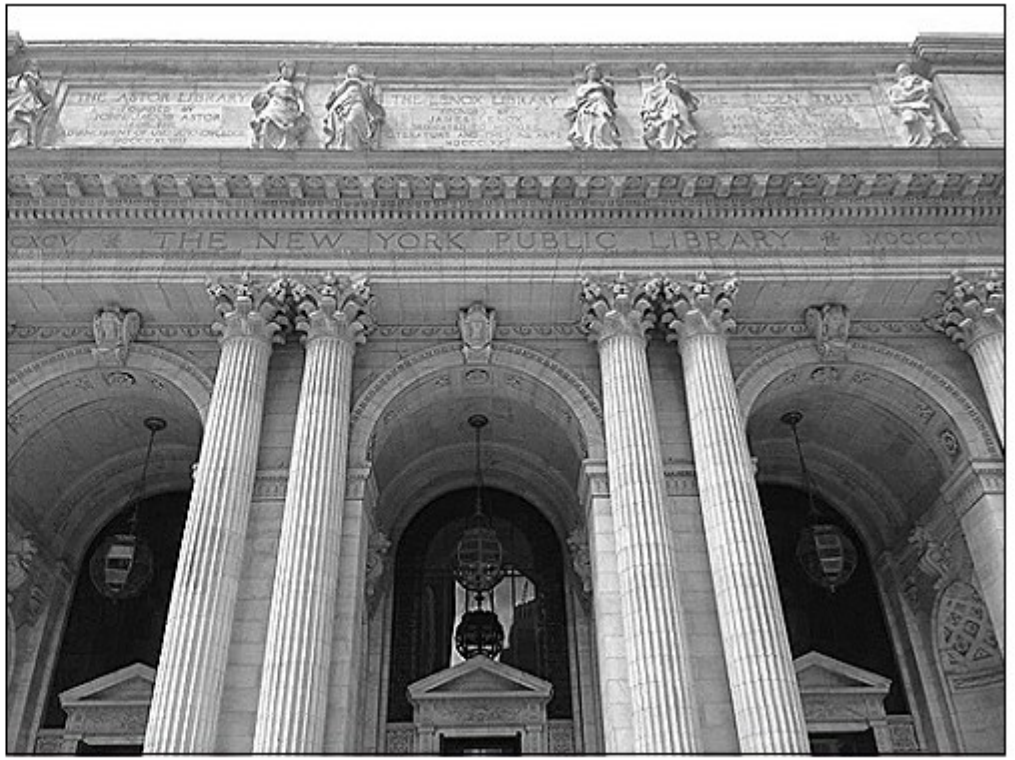
	-2	-1	0	
	-1	1	1	
	0	1	2	

Let's Try!

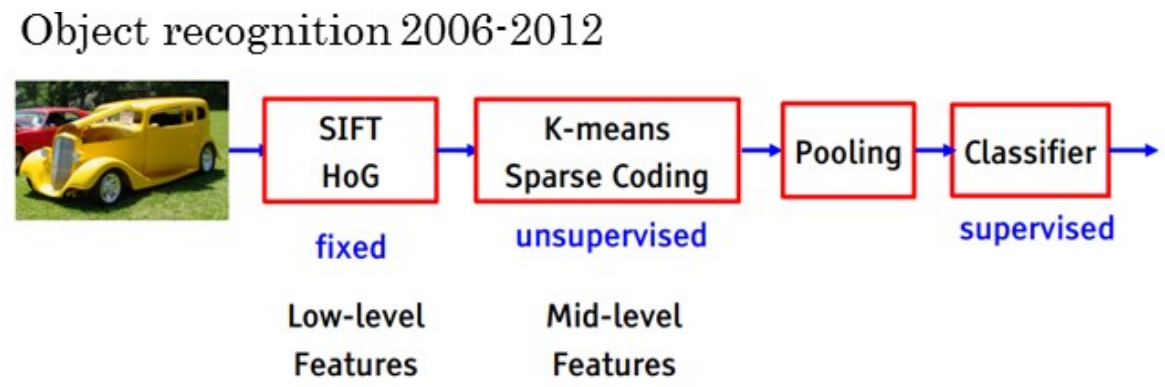
- http://setosa.io/ev/image-kernels/

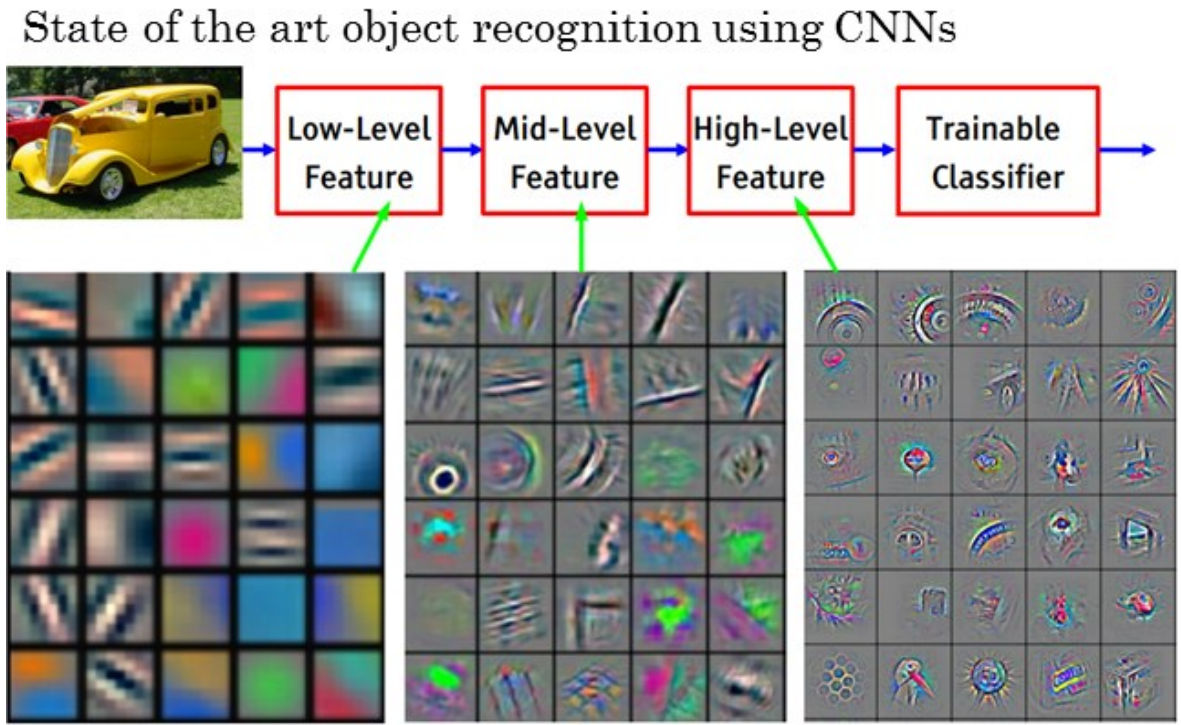
0	-1	0
-1	5	-1
0	-1	0

sharpen



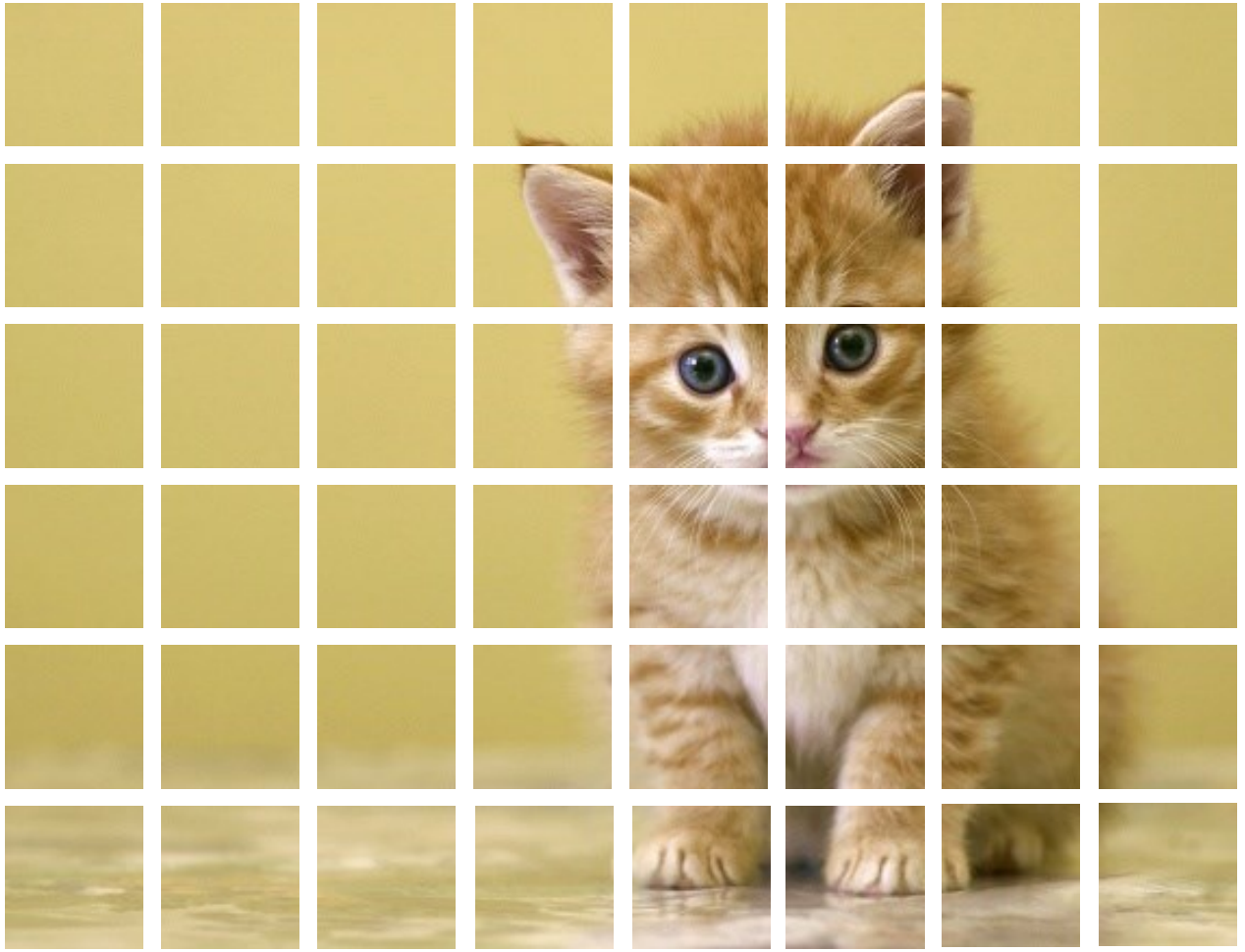
Before and After



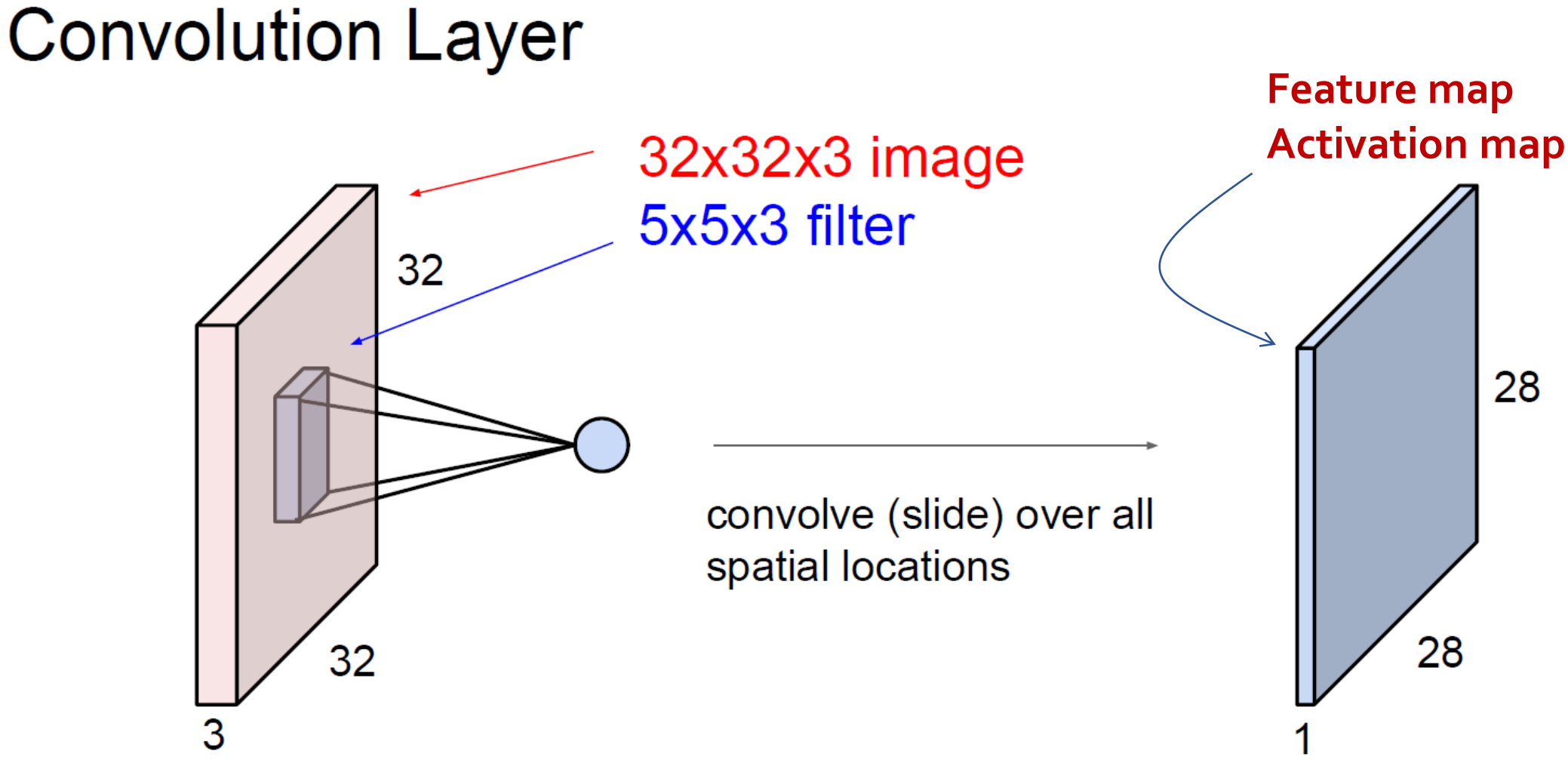


CNN 동작원리

- 이미지를 작은 tile로 나누고, 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

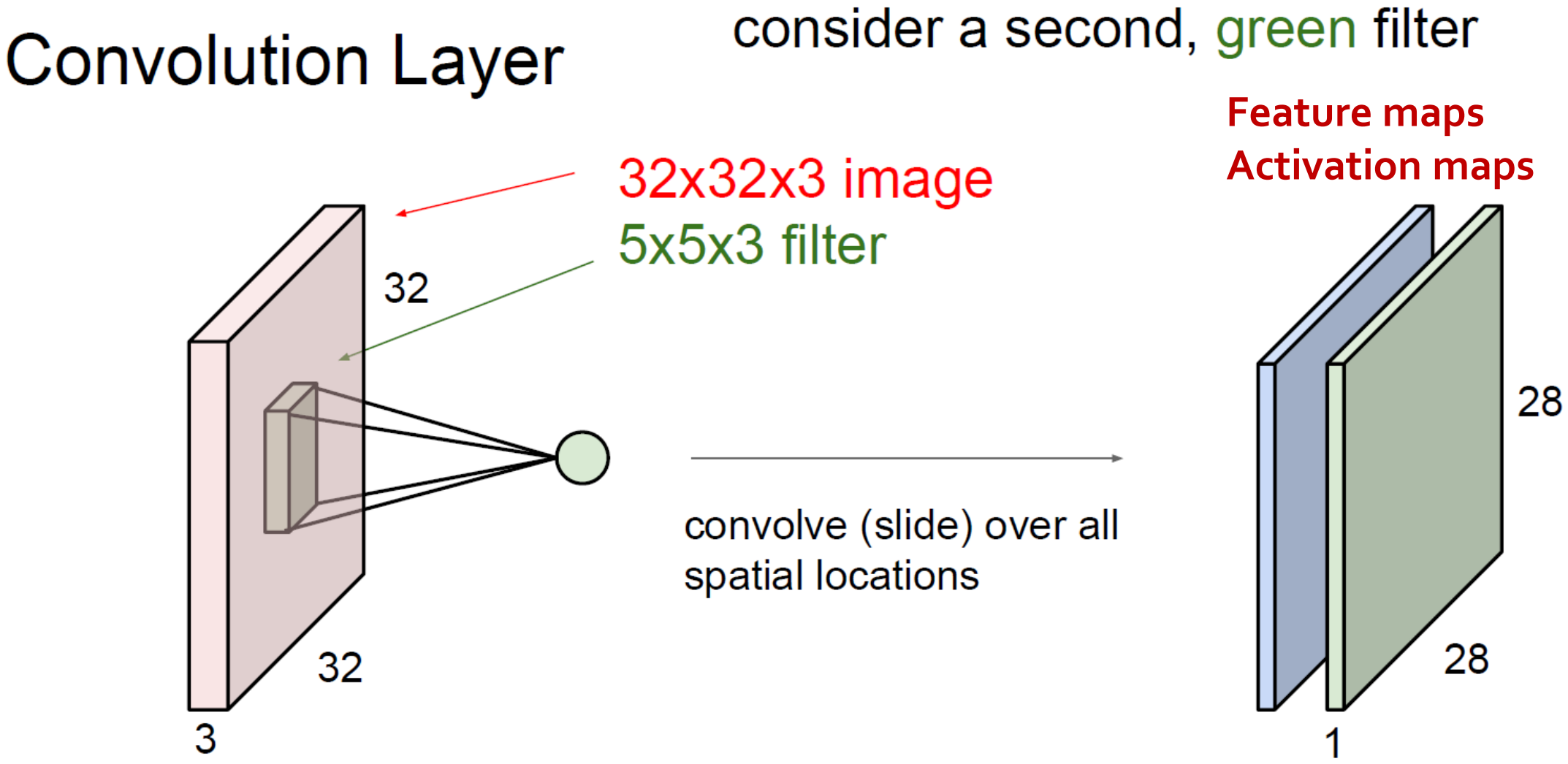


2D Convolution Layer



Slide Credit : Stanford CS231n

2D Convolution Layer



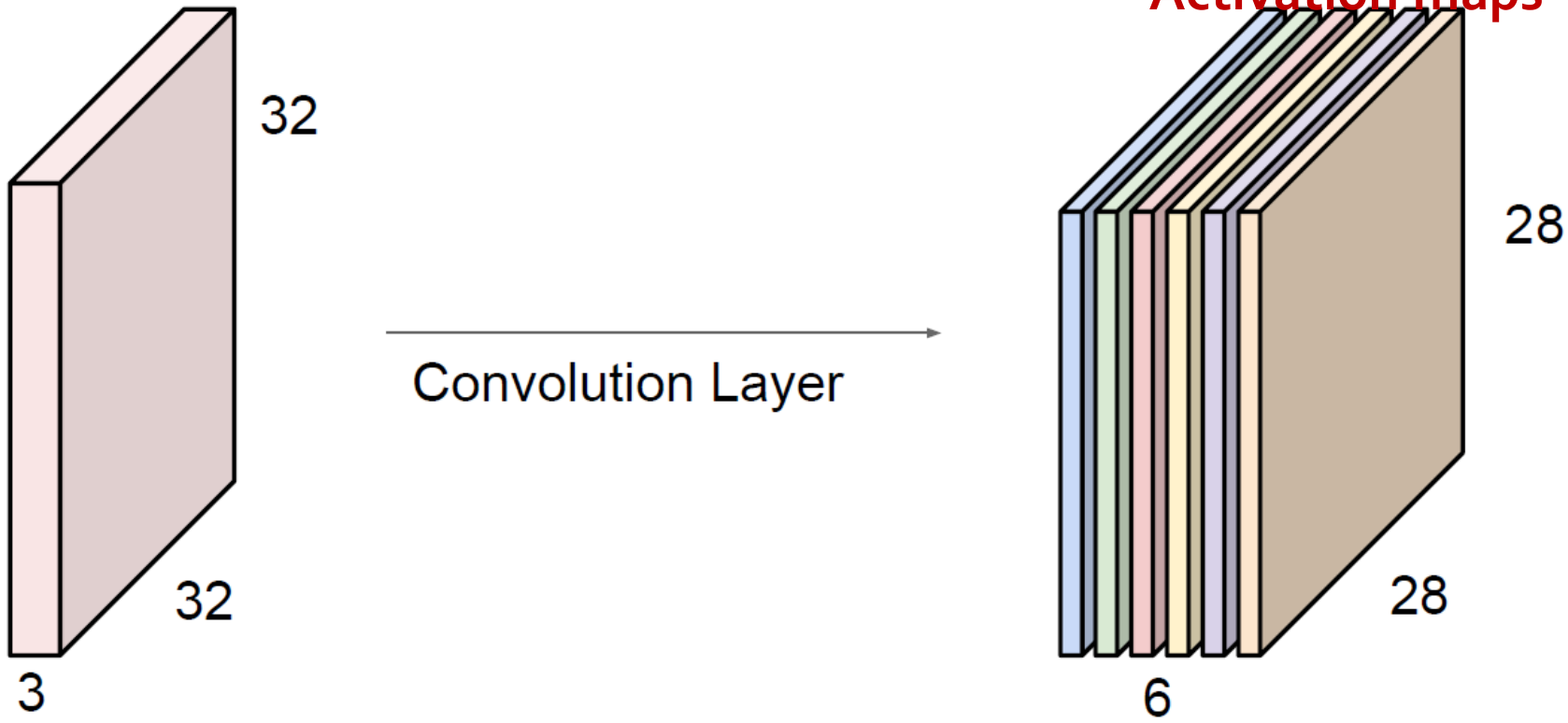
Slide Credit : Stanford CS231n

2D Convolution Layer

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

Feature maps

Activation maps

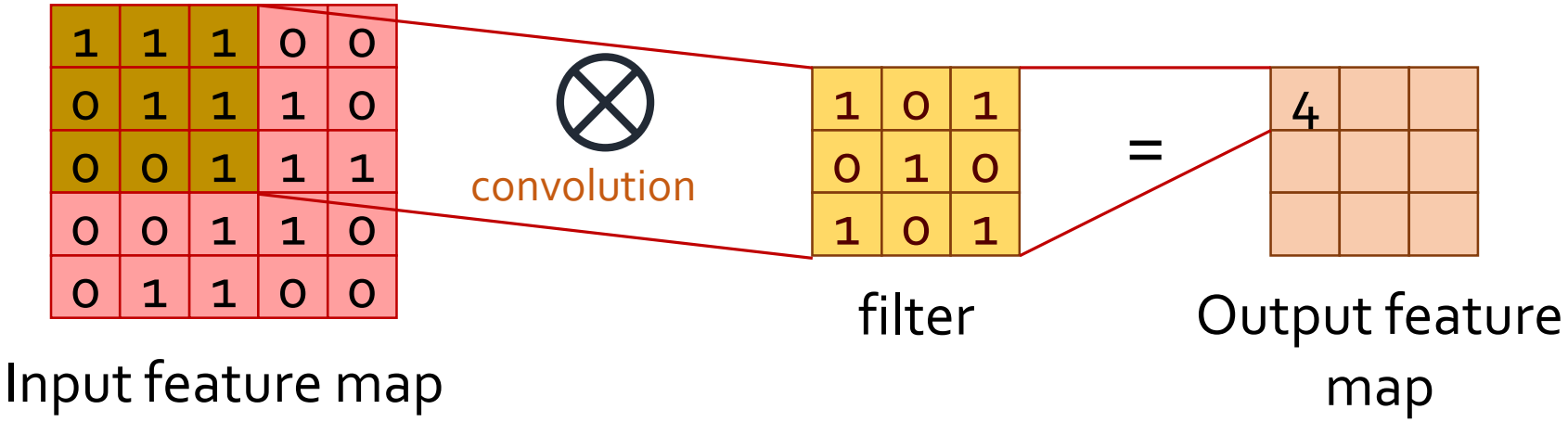


We stack these up to get a "new image" of size 28x28x6!

Slide Credit : Stanford CS231n

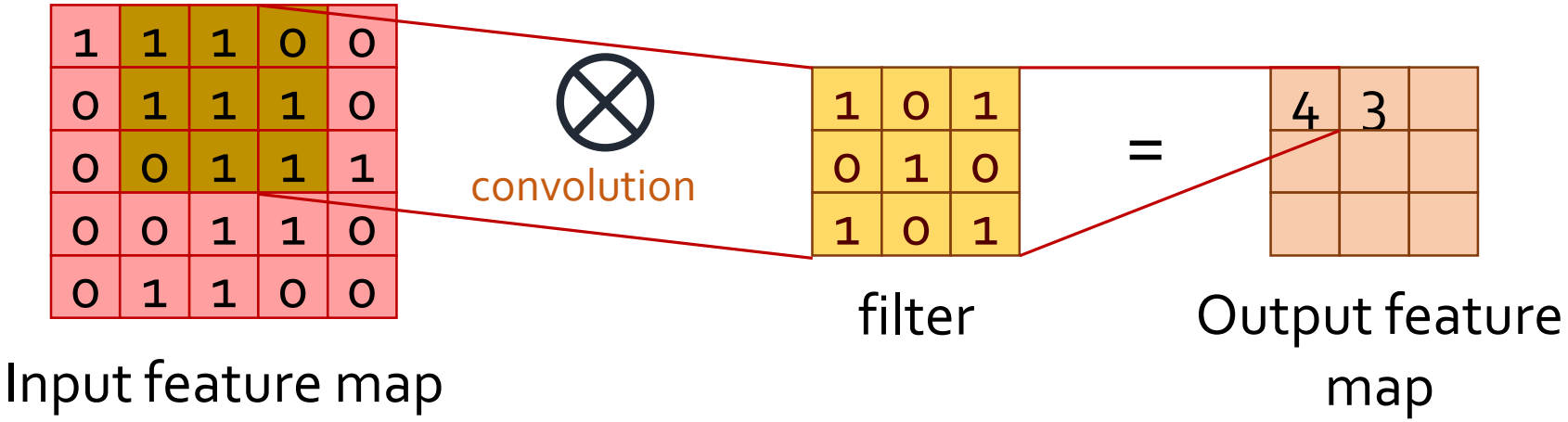
2D Convolution Layer – Computation

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



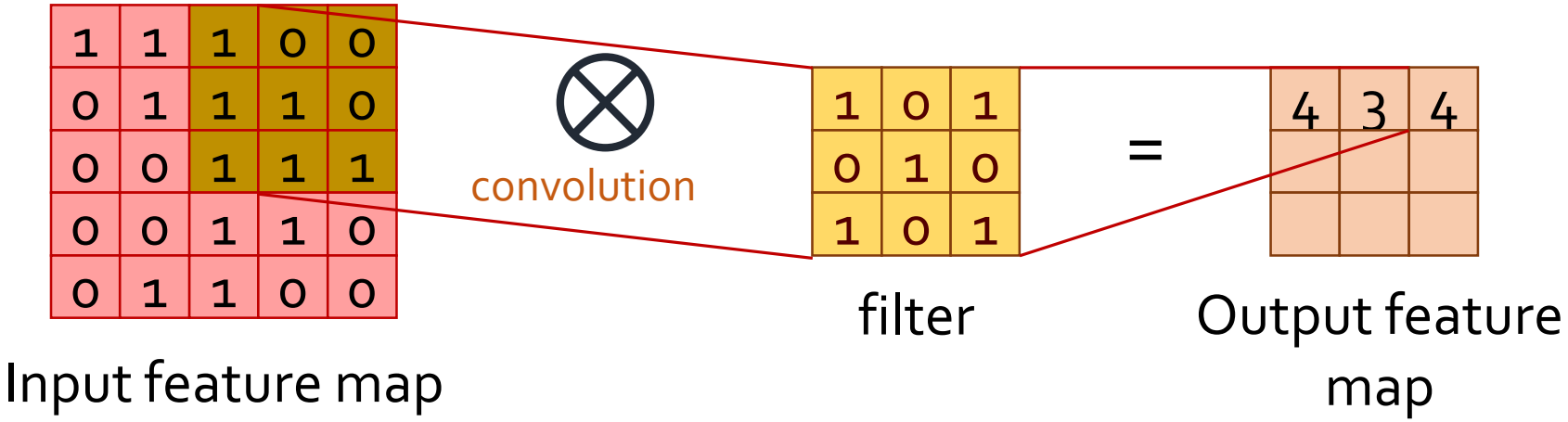
2D Convolution Layer – Computation

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



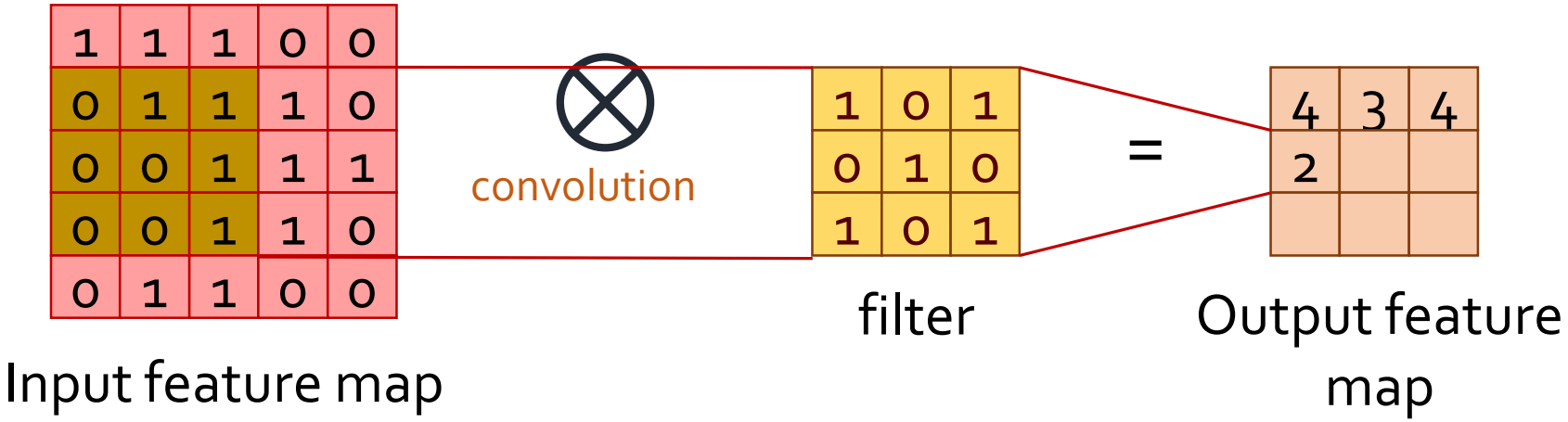
2D Convolution Layer – Computation

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



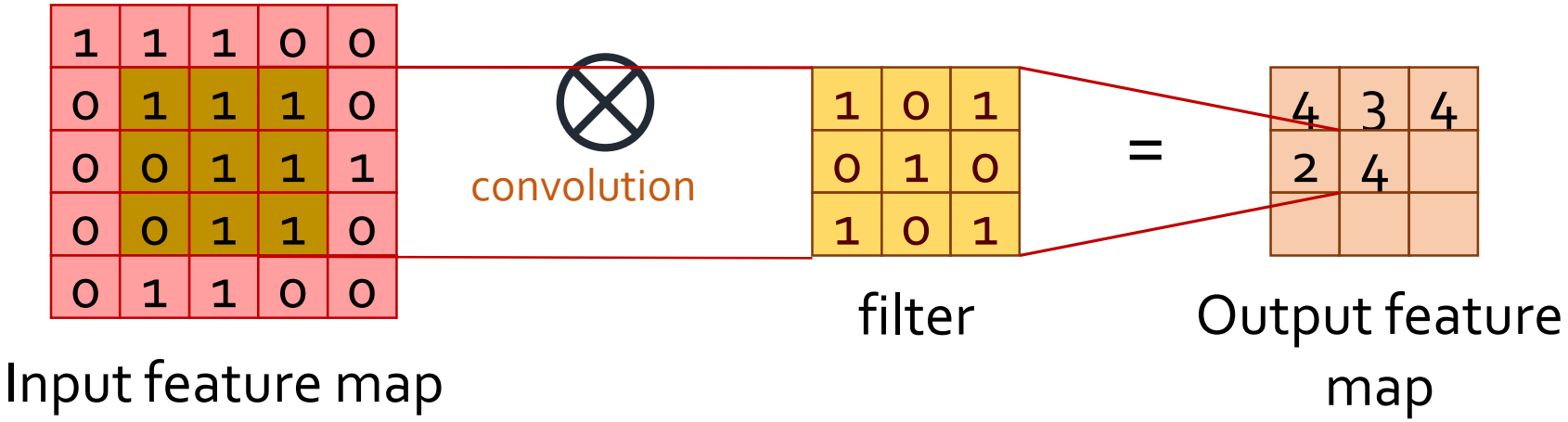
2D Convolution Layer – Computation

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



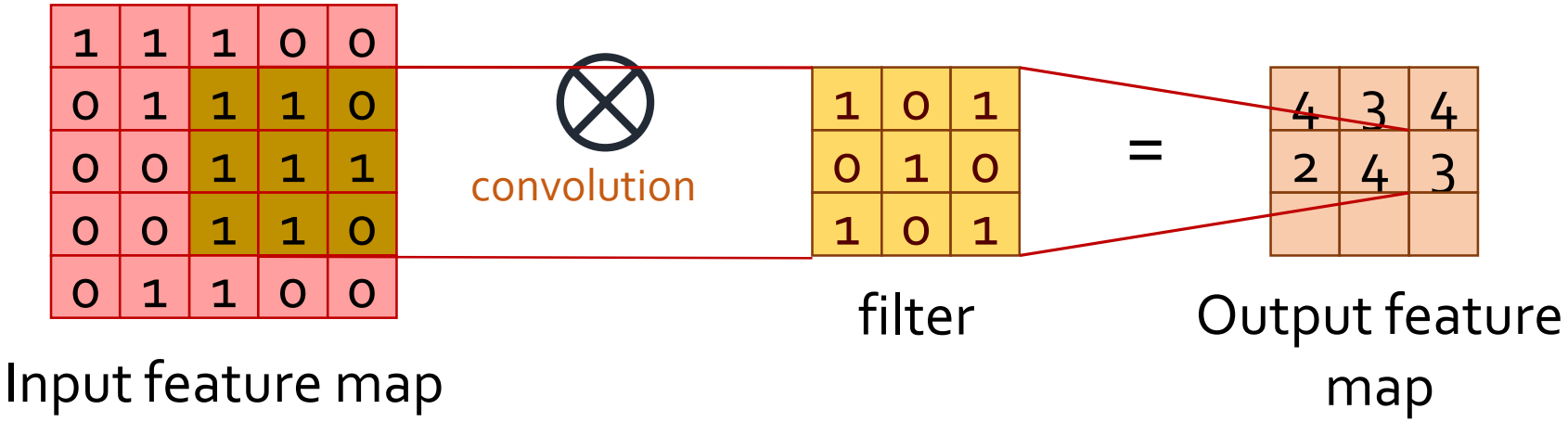
2D Convolution Layer – Computation

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



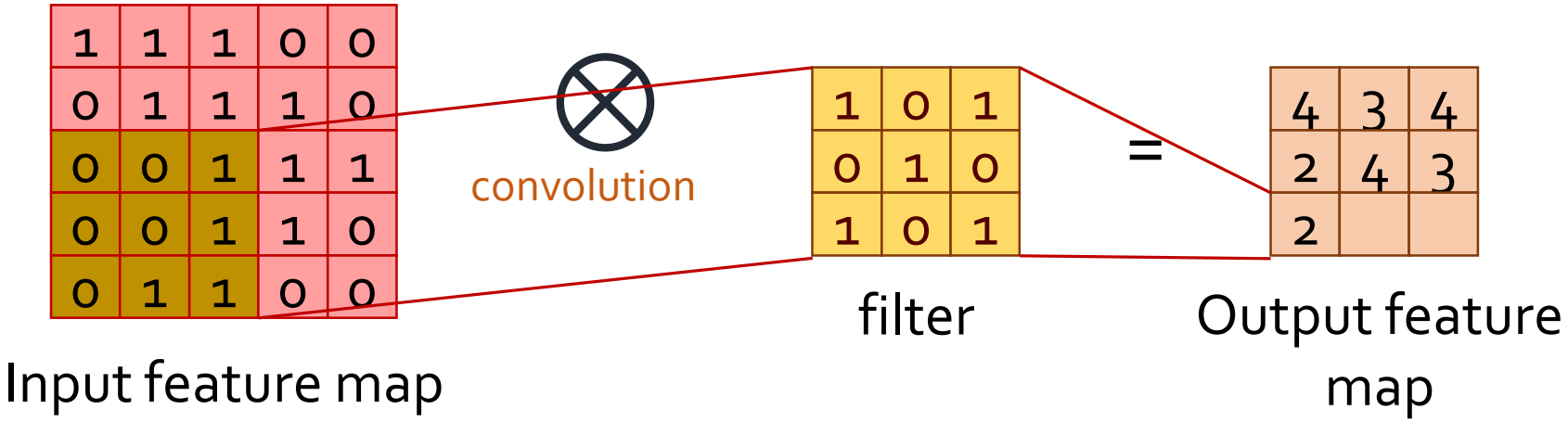
2D Convolution Layer – Computation

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



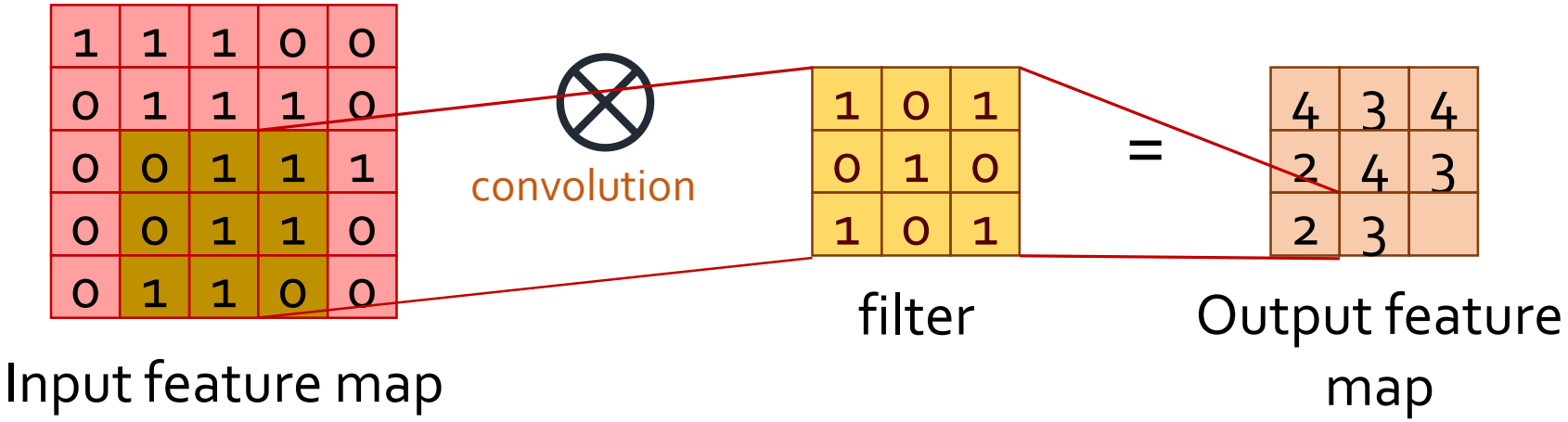
2D Convolution Layer – Computation

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



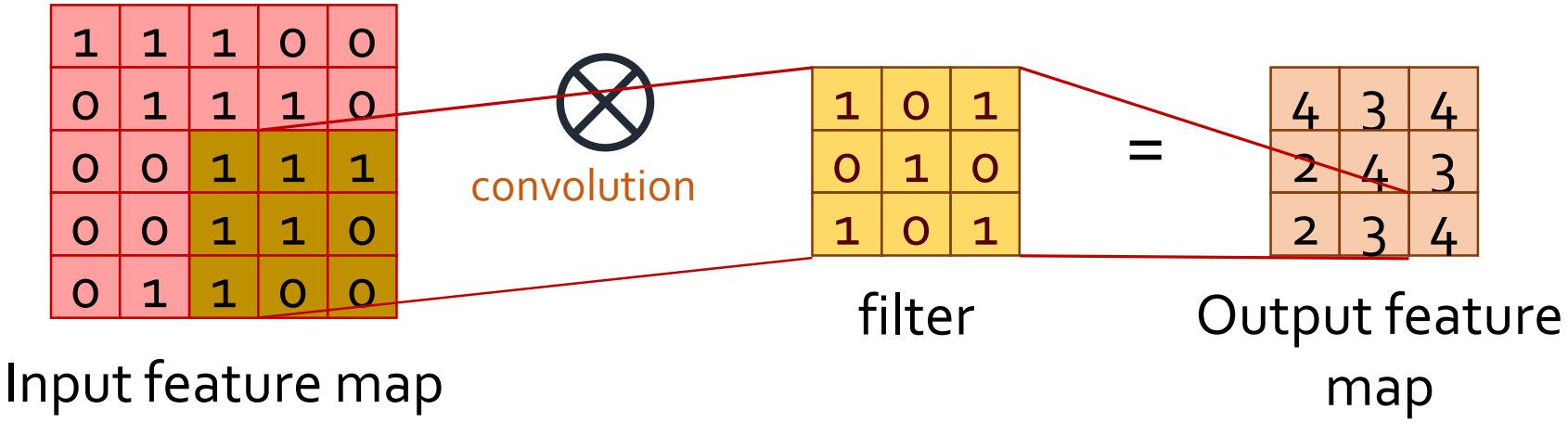
2D Convolution Layer – Computation

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

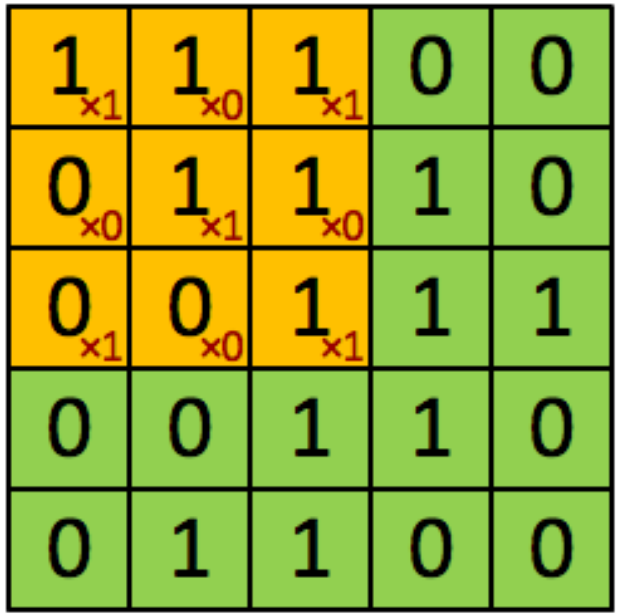


2D Convolution Layer – Computation

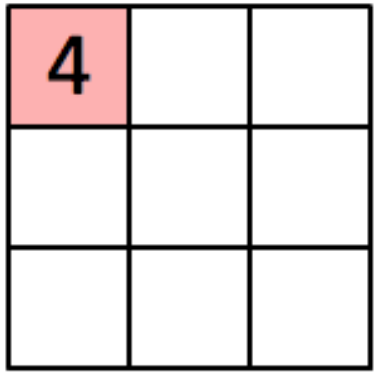
- 1X1 + 1X0 + 1X1 + 1X0 + 1X1 + 0X0 + 1X1 + 0X0 + 0X1 = 4



2D Convolution Layer – Computation



Image

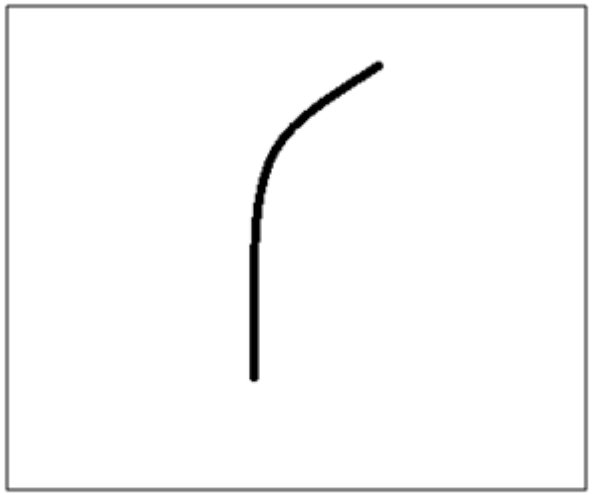


Convolved Feature

Feature Extractor

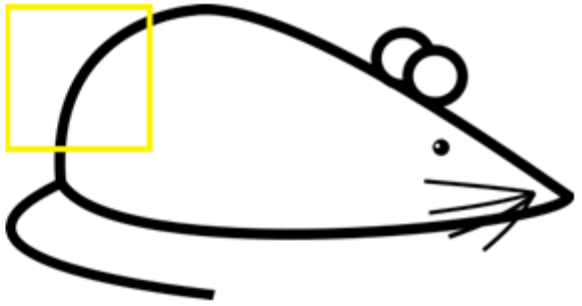
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

Original image



Visualization of the filter on the image

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

*

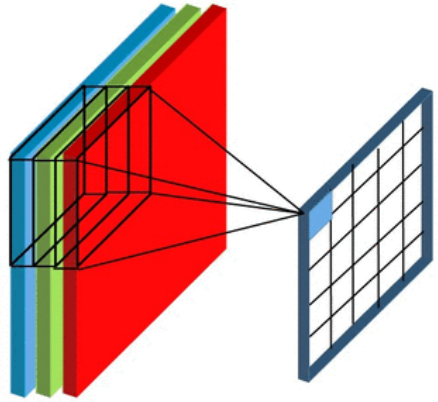
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

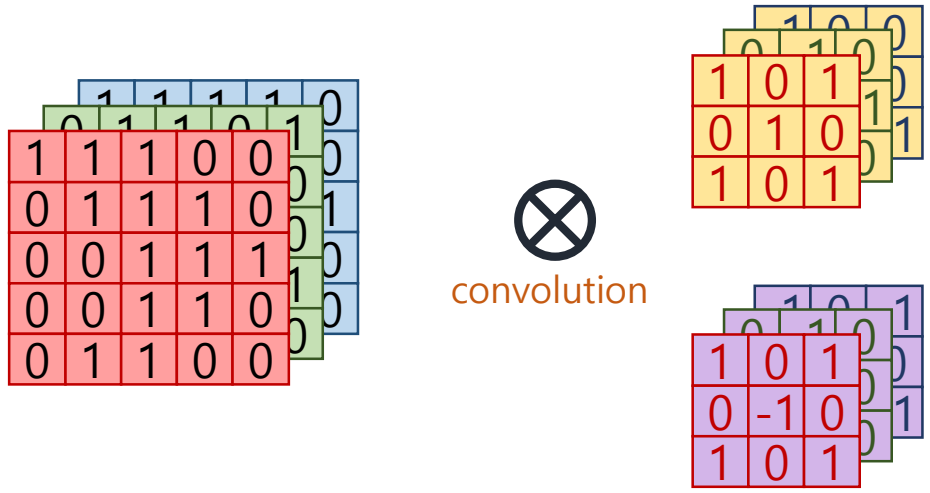
Pixel representation of filter

Multiplication and Summation = 0

Credit : Adit Deshpande's blog

Convolution
(Multi Channel, Many Filters)



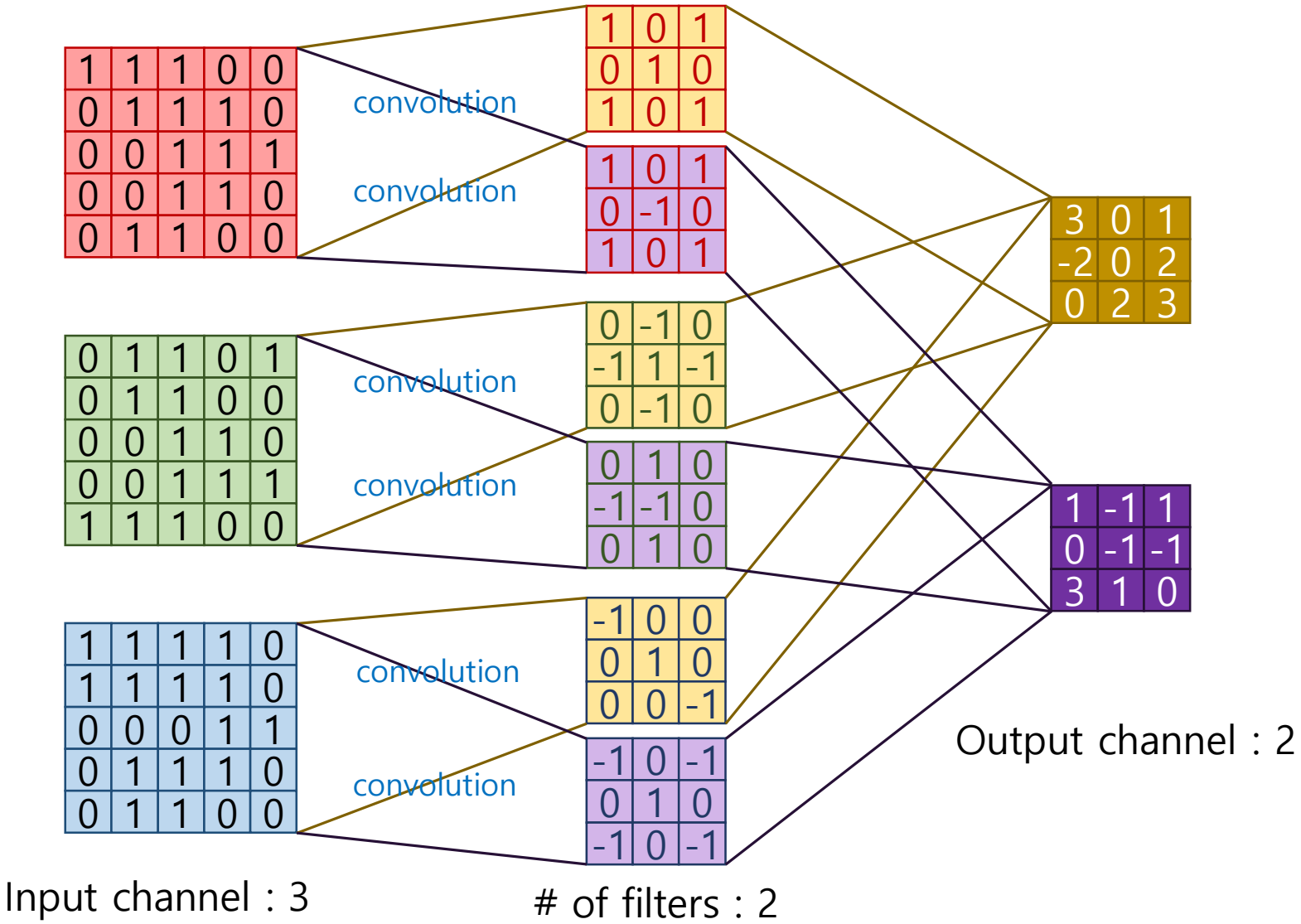


Input channel : 3

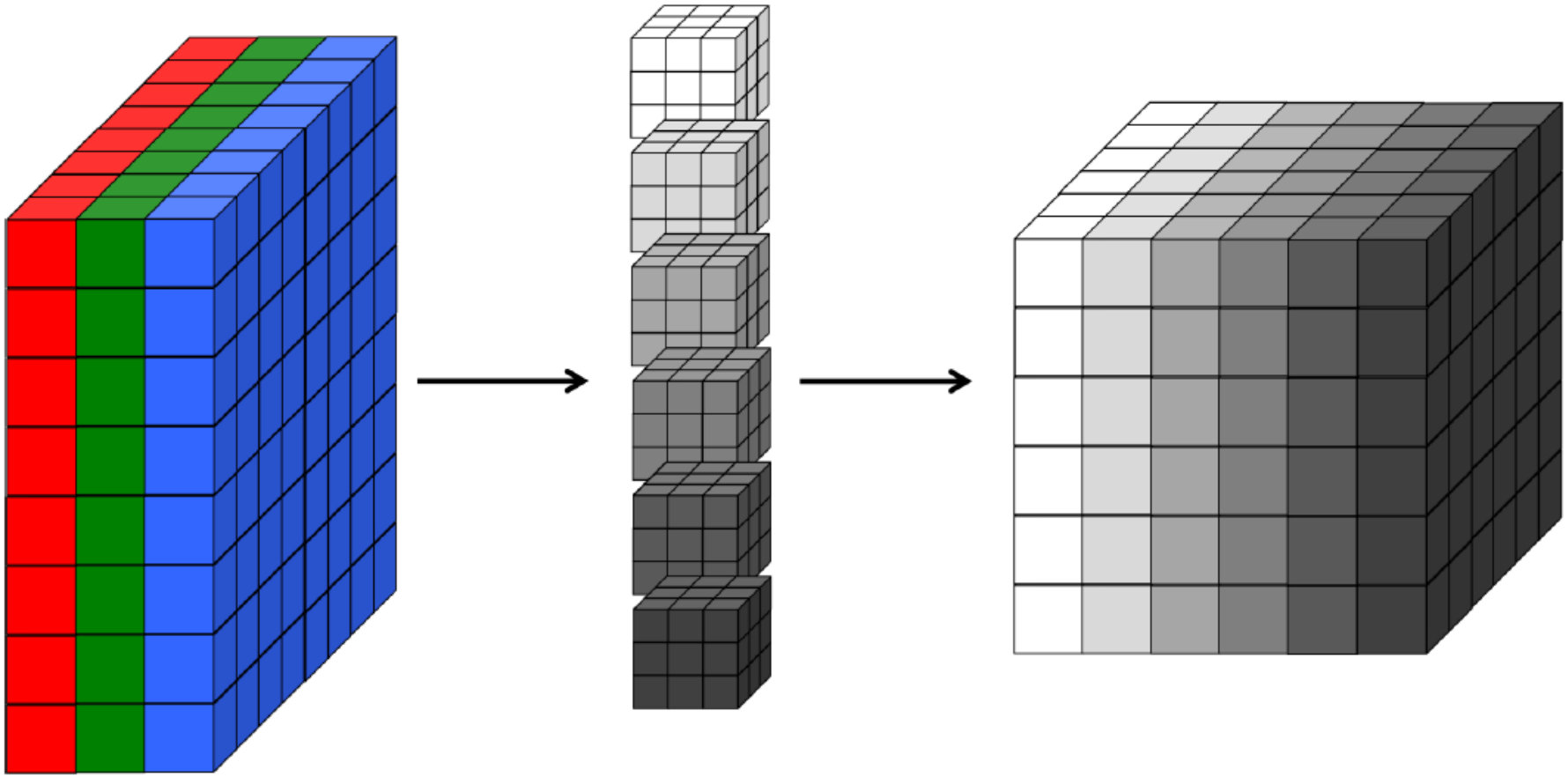
# of filters : 2

Output channel : 2

Convolution
(Multi Channel, Many Filters)

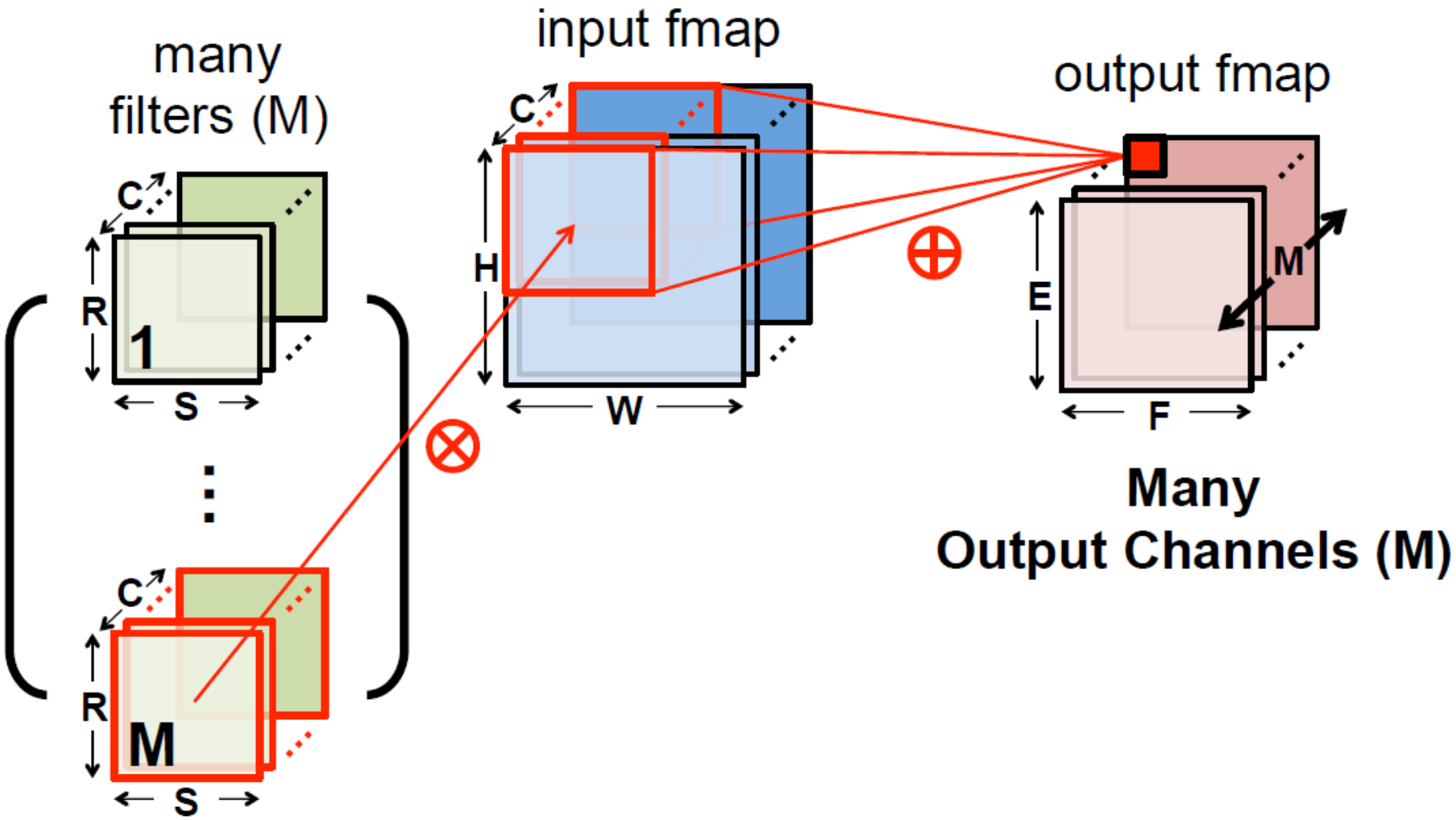


Visualization of a Convolution Layer



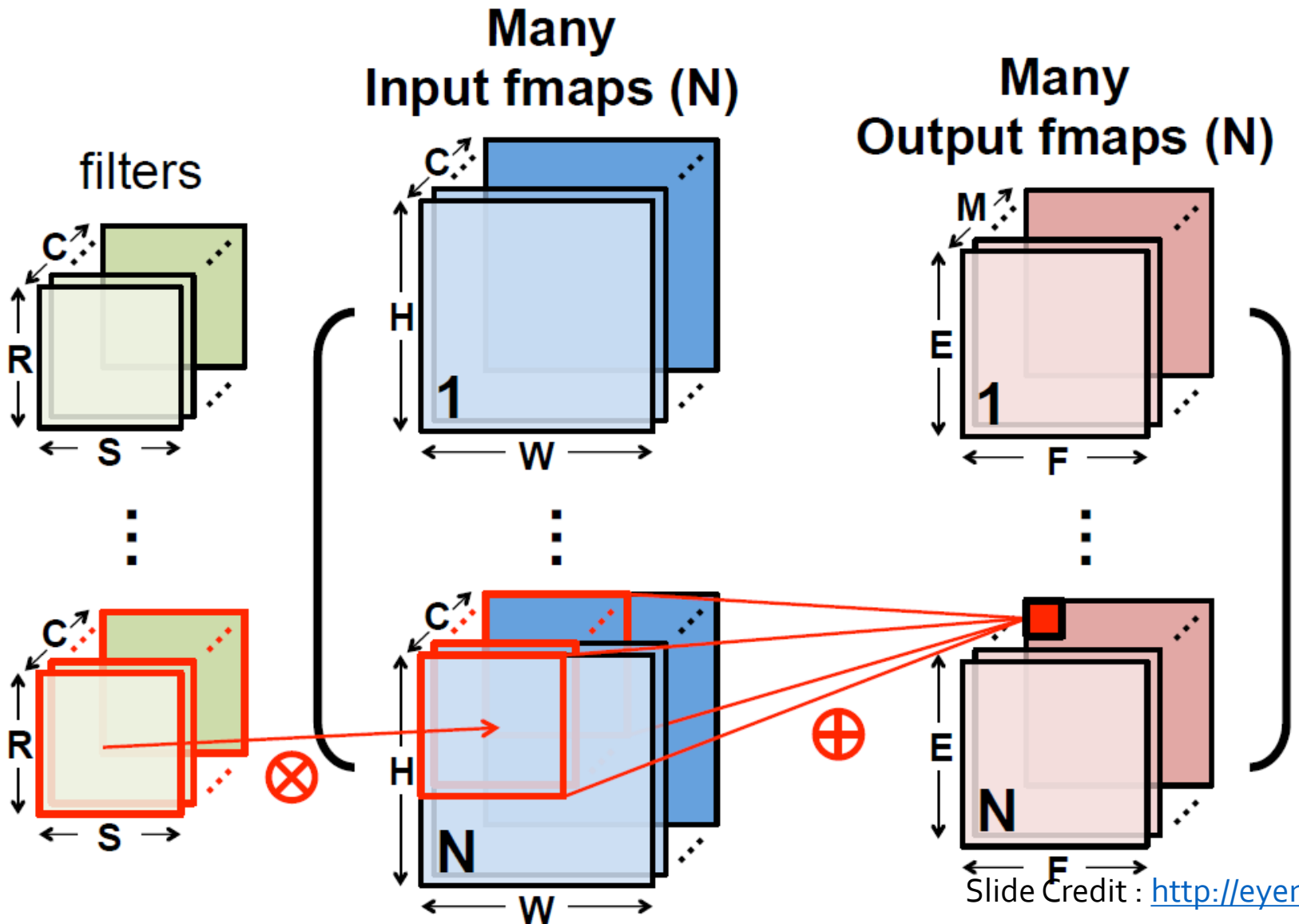
Picture Credit : fundamentals of deep learning by Nikhil

2D Convolution Layer



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

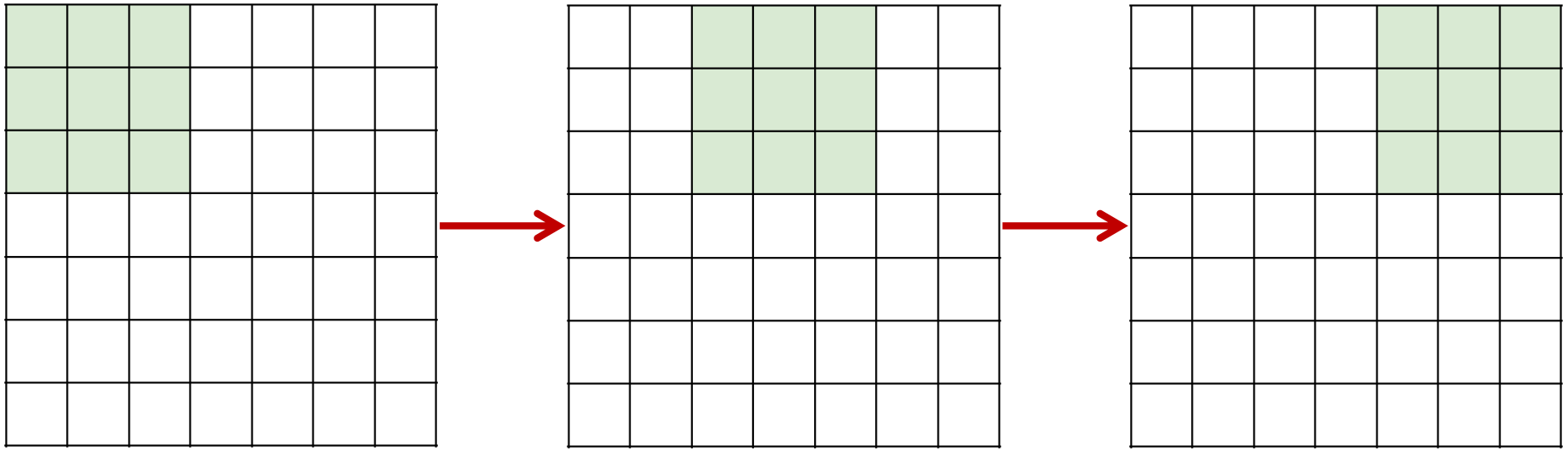
2D Convolution Layer – 4D Tensors



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

Options of Convolution

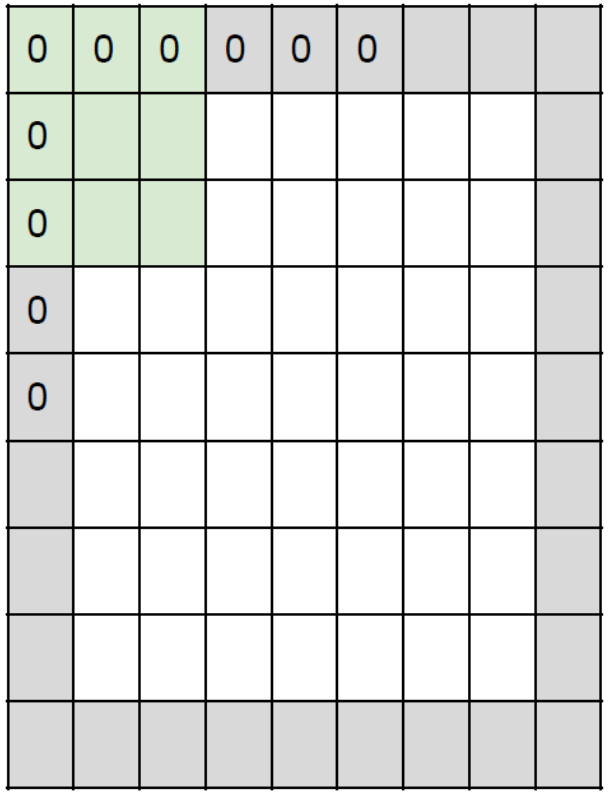
- Stride : filter가 한 번 convolution을 수행 한 후 옆으로(혹은 아래로) 얼마나 이동할 것인가
  - 예) 7x7 input, 3x3 convolution filter with stride 2 → 3x3 output!



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

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

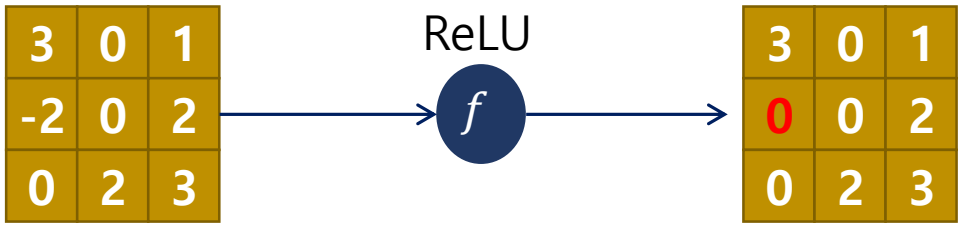
Input이 $W_i \times H_i \times C_i$ 이고,
$F \times F$ filter를 K개 사용하고,
stride는 S,
zero padding은 P 만큼 했을 경우,
output feature map size($W_o \times H_o \times C_o$)는,

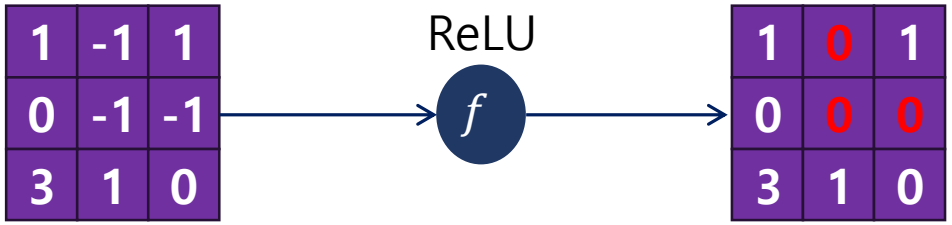
$$W_o = \frac{(W_i - F + 2P)}{S} + 1$$

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

$$C_o = K$$

ReLU





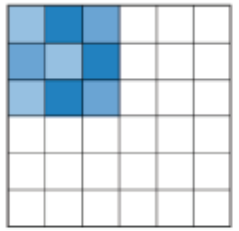
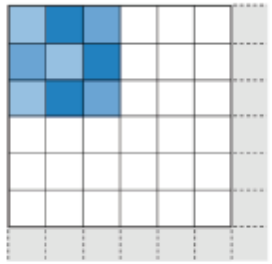
tf.keras.layers.Conv2D

```
__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

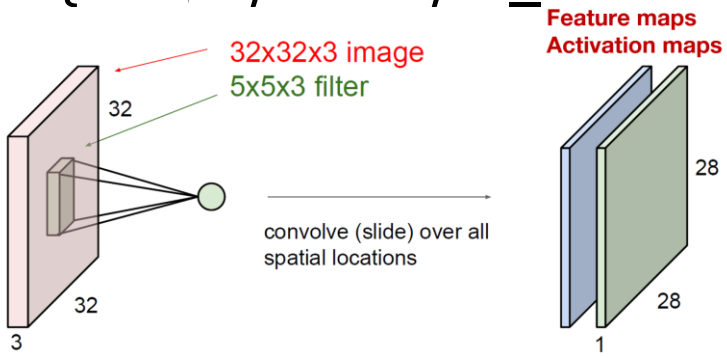
	Valid	Same
Value	$P = 0$	$P_{\text{start}} = \left\lfloor \frac{S\lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rfloor$ $P_{\text{end}} = \left\lceil \frac{S\lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rceil$
Illustration		
Purpose	- No padding - Drops last convolution if dimensions do not match	- Padding such that feature map size has size $\left\lceil \frac{I}{S} \right\rceil$ - Output size is mathematically convenient - Also called 'half' padding

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.

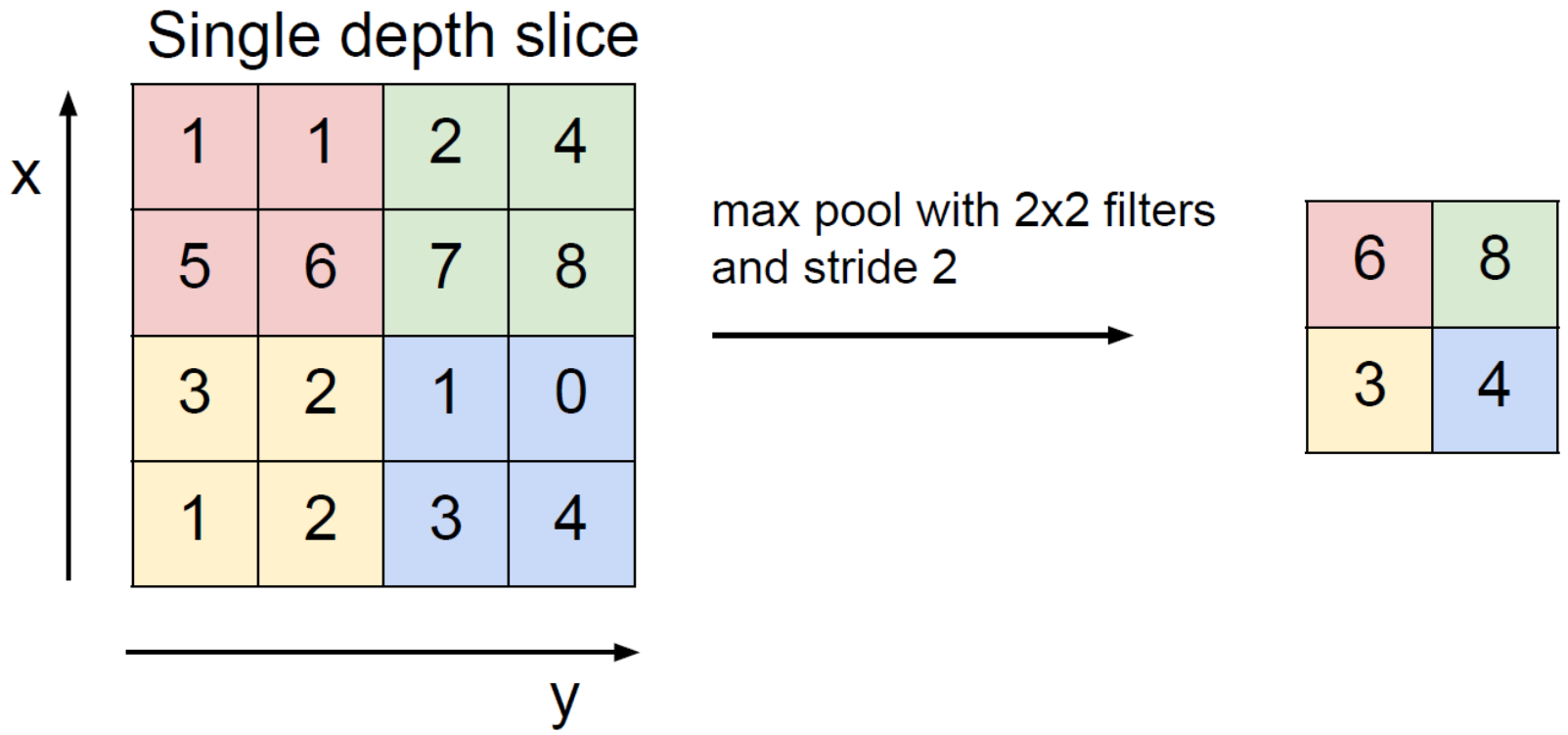
kernel dimension : {height, width, in_channel, out_channel}

Ex) {5, 5, 3, 2}



Pooling Layer

- Max pooling or Average Pooling



2x2 Max Pooling with Stride=1

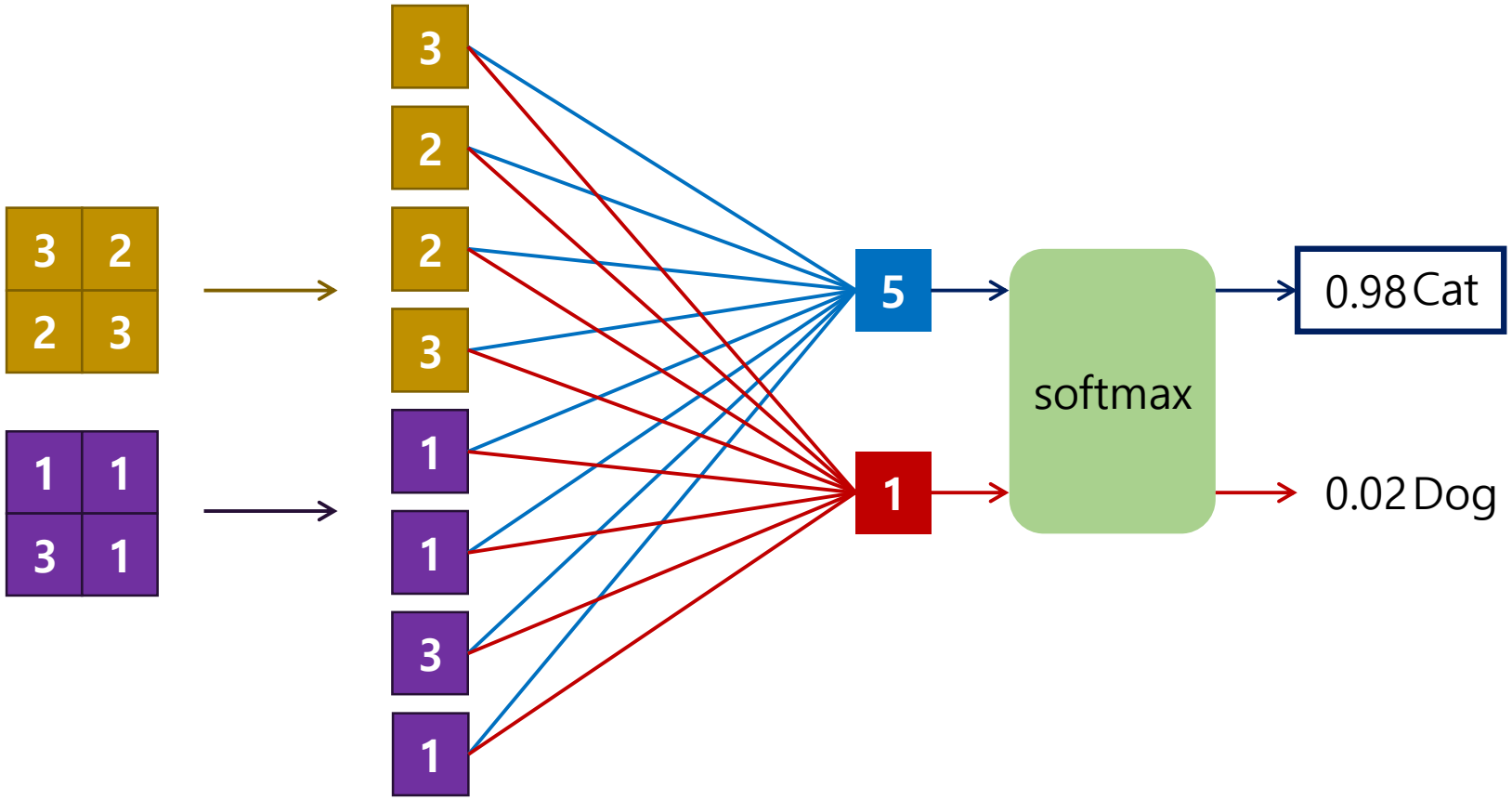
tf.keras.layers.MaxPool2D

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

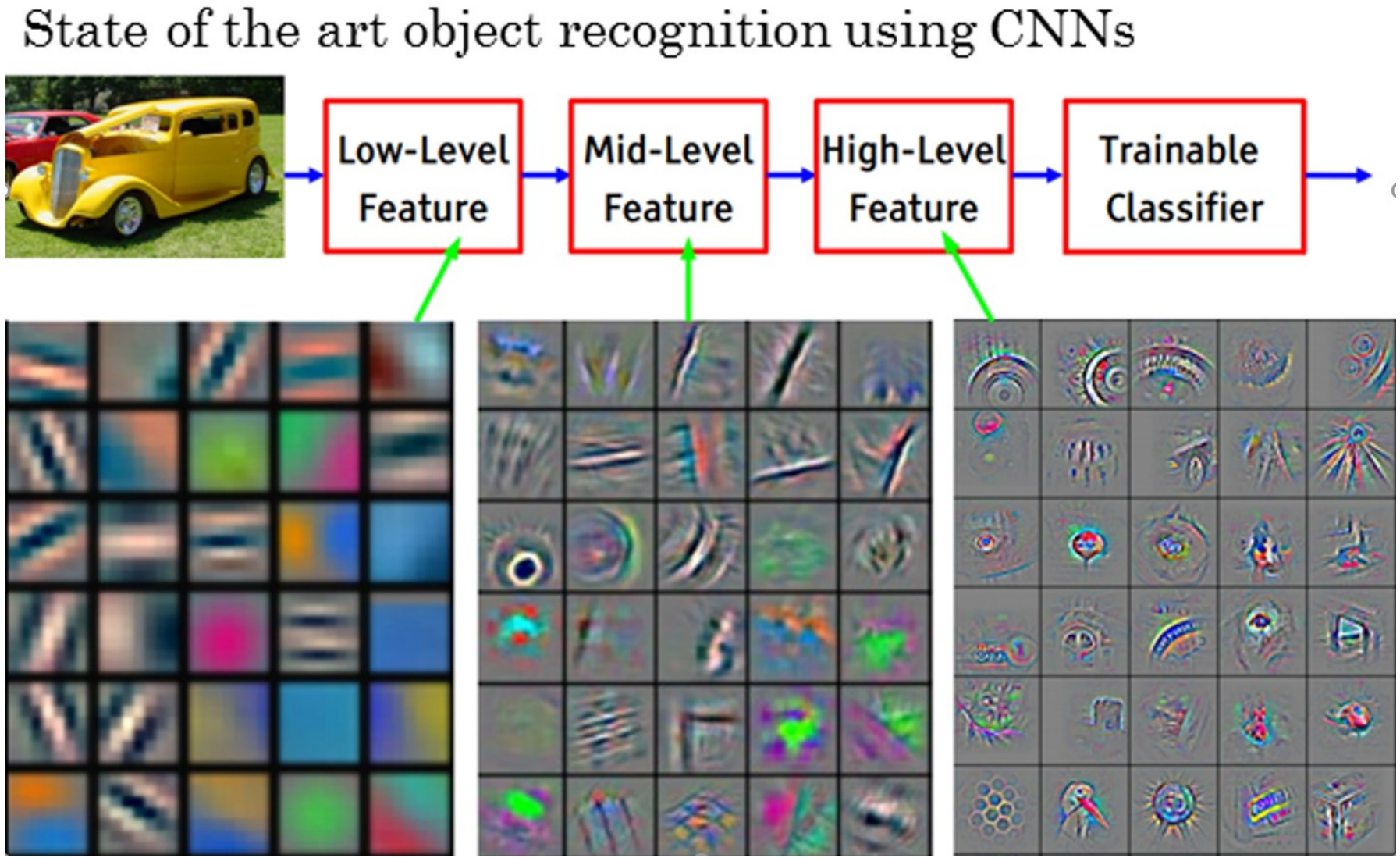
tf.keras.layers.MaxPool2D

- `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



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