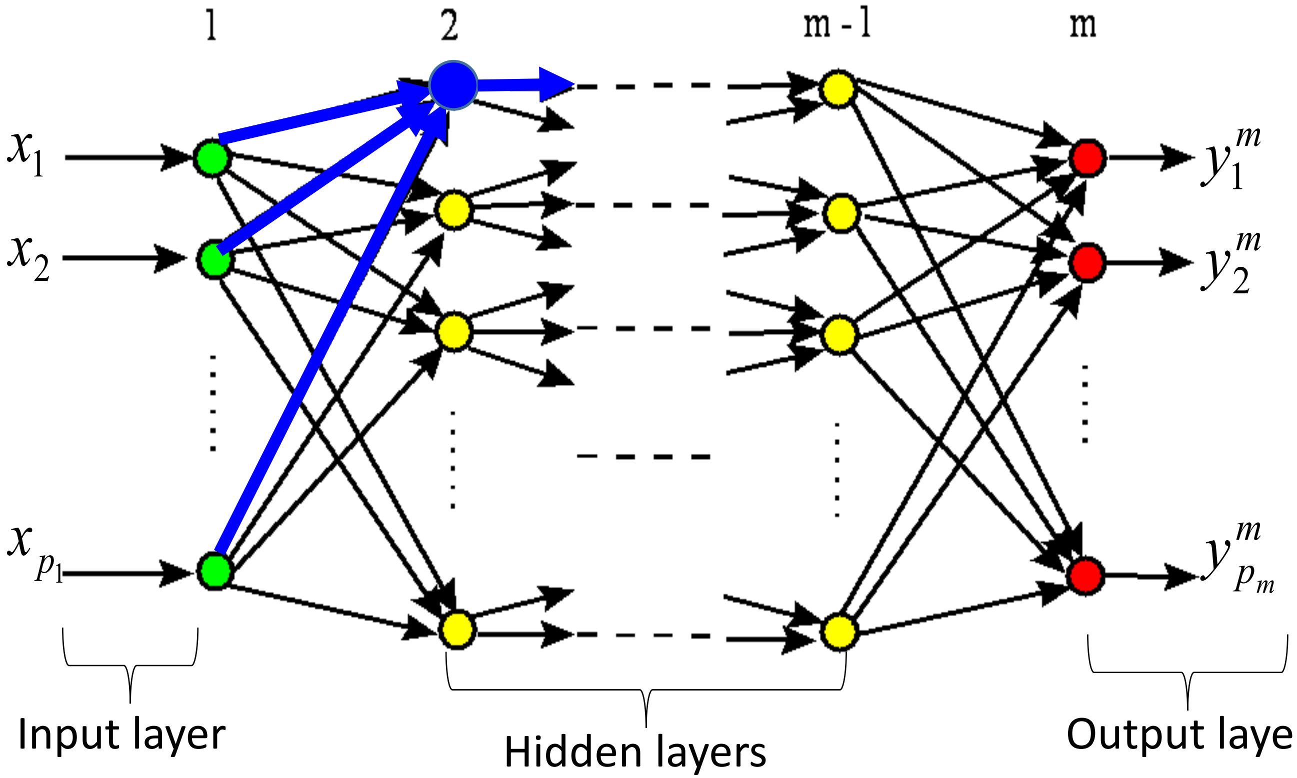
05
Neural Networks
Convolutional Neural Networks (CNNs)

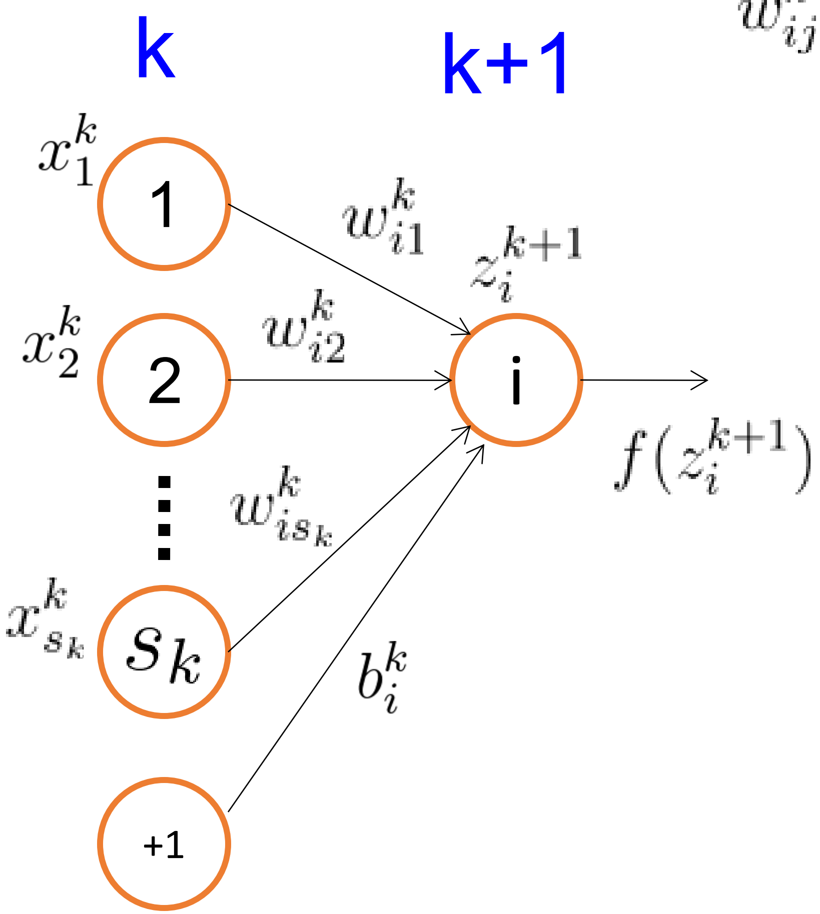
Review: fully-connected (FC) neural network



Vector representation:

$$X = [x_1 \quad x_2 \quad \cdots \quad x_{p_1}]^T \qquad Y = [y_1^m \quad y_2^m \quad \cdots \quad y_{p_m}^m]^T$$

Review: input-output mapping



w_{ij}^k : the weight associated with the edge linking the j-th neuron In layer k and the i-th neuron in layer k+1

$$z_i^{k+1} = \sum_{j=1}^{s_k} w_{ij}^k x_{s_k}^k + b_i^k$$

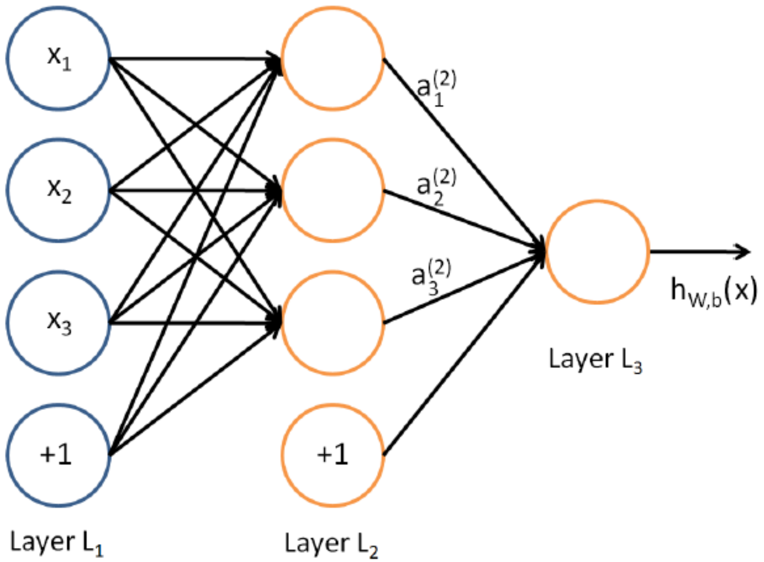
$$a_i^{k+1} = f(z_i^{k+1}) = \frac{1}{1+\exp(-z_i^{k+1})}$$

$$f'(z_i^{k+1}) = f(z_i^{k+1})(1 - f(z_i^{k+1}))$$

$$\forall i = 1, 2, \ldots, s_{k+1}$$

$$\forall k = 1, 2, \ldots, n_l$$

Review: example



$$a_1^{(2)} = f(W_{11}^{(1)} x_1 + W_{12}^{(1)} x_2 + W_{13}^{(1)} x_3 + b_1^{(1)})$$
$$a_2^{(2)} = f(W_{21}^{(1)} x_1 + W_{22}^{(1)} x_2 + W_{23}^{(1)} x_3 + b_2^{(1)})$$
$$a_3^{(2)} = f(W_{31}^{(1)} x_1 + W_{32}^{(1)} x_2 + W_{33}^{(1)} x_3 + b_3^{(1)})$$
$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)} a_1^{(2)} + W_{12}^{(2)} a_2^{(2)} + W_{13}^{(2)} a_3^{(2)} + b_1^{(2)})$$

Review: gradient based optimization

Calculate gradient on each training instance:

1) Forward propagation: calculate outputs from 1st layer to last layer
2) Backward propagation: calculate errors δ from last layer to 1st layer:

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \; \frac{1}{2} \left\|y - h_{W,b}(x)\right\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)}), \;\; \forall l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$$

If Sigmoid activation is used: $f'(z_i^{(l)}) = f(z_i^{(l)})(1 - f(z_i^{(l)}))$

3) Calculate gradients:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y) = \delta_i^{(l+1)}.$$

Review: back propagation algorithm

（1）**Initialize W and b randomly**

（2）**Set** $\Delta W^{(l)}, \Delta b^{(l)}$ **to 0** $\forall l = 1, 2, 3, \ldots, n_l - 1$

（3）**For** $i = 1$ **to** m (the number of samples),

a): **Calculate** $\nabla_{W^{(l)}} J(W, b; x, y)$， $\nabla_{b^{(l)}} J(W, b; x, y)$

b): **let** $\Delta W^{(l)} := \Delta W^{(l)} + \nabla_{W^{(l)}} J(W, b; x, y)$

c): **let** $\Delta b^{(l)} := \Delta b^{(l)} + \nabla_{b^{(l)}} J(W, b; x, y)$

（4）**Update W and b:**

$$W^{(l)} = W^{(l)} - \alpha \left[\left(\frac{1}{m} \Delta W^{(l)} \right) + \lambda W^{(l)} \right]$$

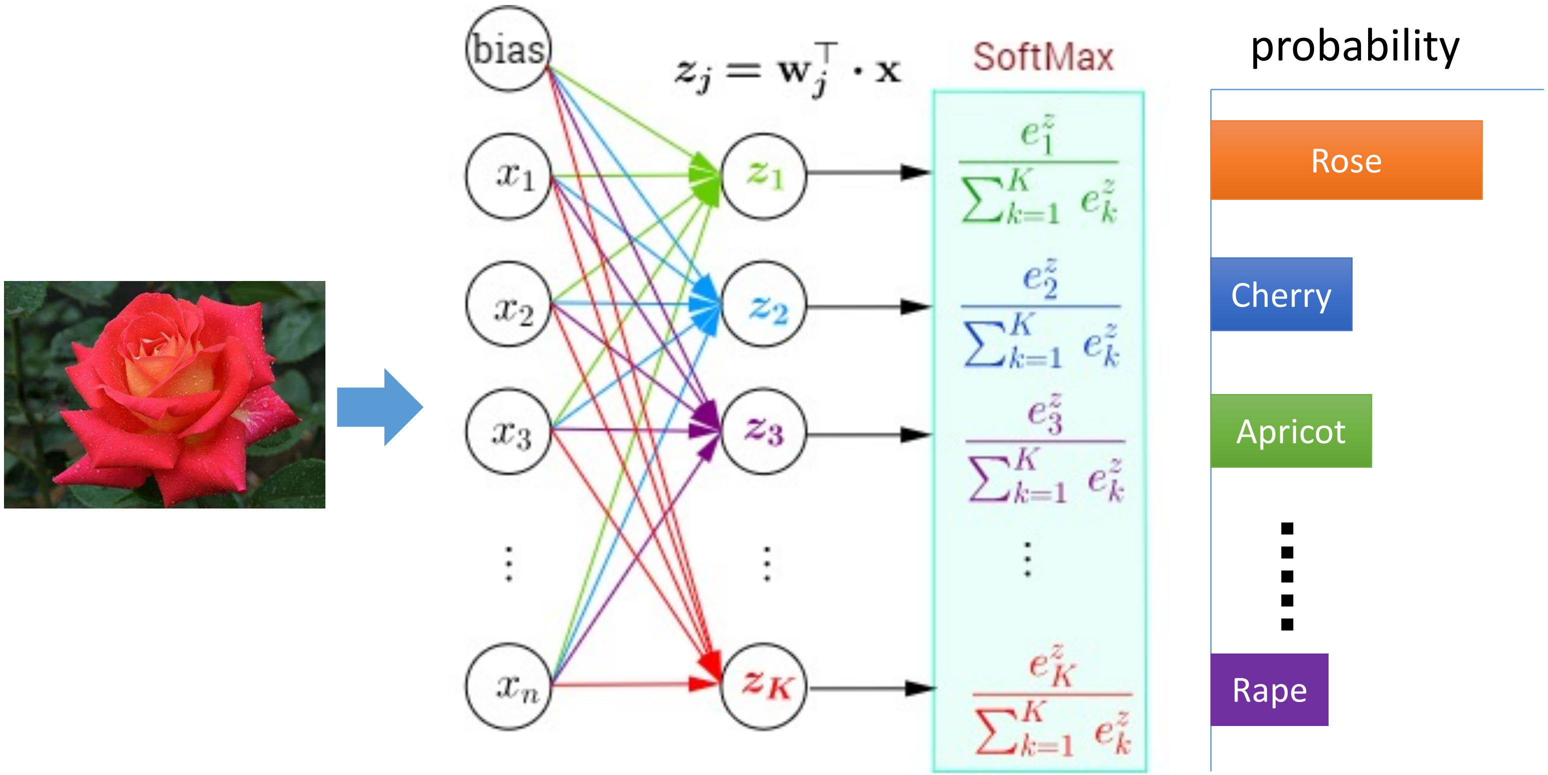
$$b^{(l)} = b^{(l)} - \alpha \left[\frac{1}{m} \Delta b^{(l)} \right]$$

Roadmap

- Fully-connected neural network
- Convolutional neural network (CNN)
- Popular CNN architectures

Image classification task

Fully-connected neural network solution



The issue of FC neural network

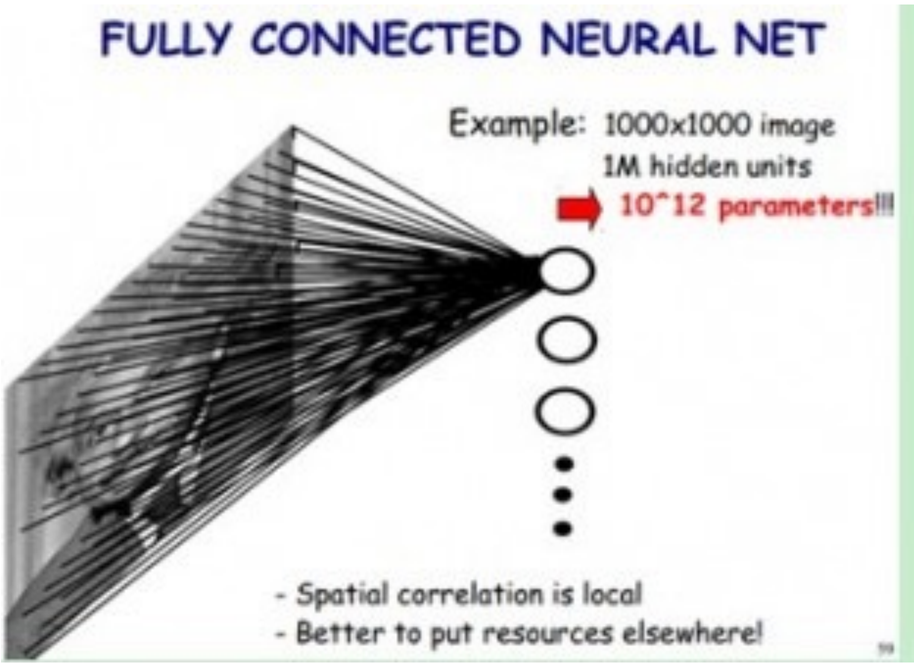
- Issue: too many neurons (weights)
  - Think about the flower classification task
  - Insight: Human eyes perceives from local to global

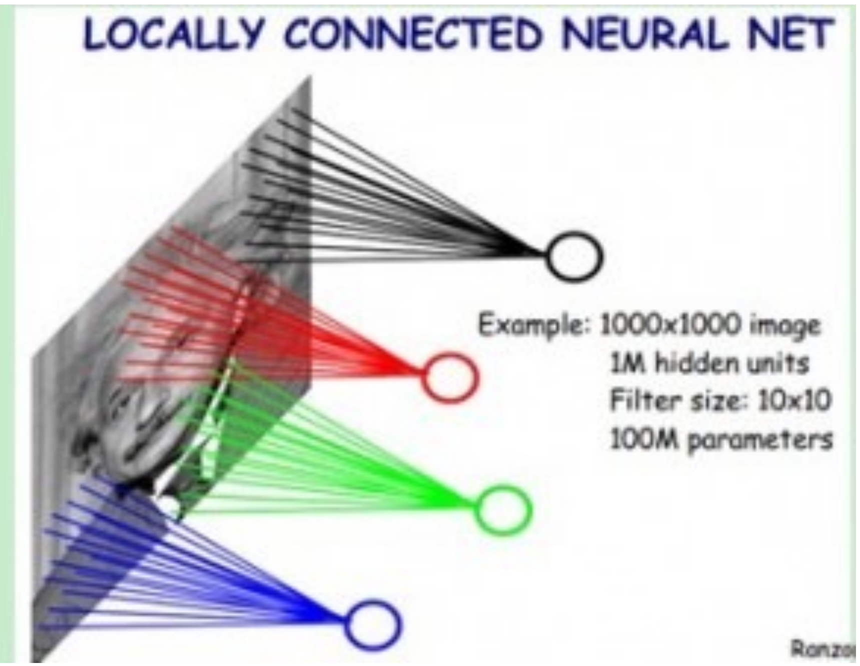
- Fix: local perceiving, weights sharing
  - Two assumptions
    - Proximate pixels are correlated, while distant pixels are independent
    - Different local regions preserves identical statistical properties in images

Fix issue: local perception

Insight: proximate pixels are correlated, while faraway pixels are independent

Fix: replace full connection by local connection

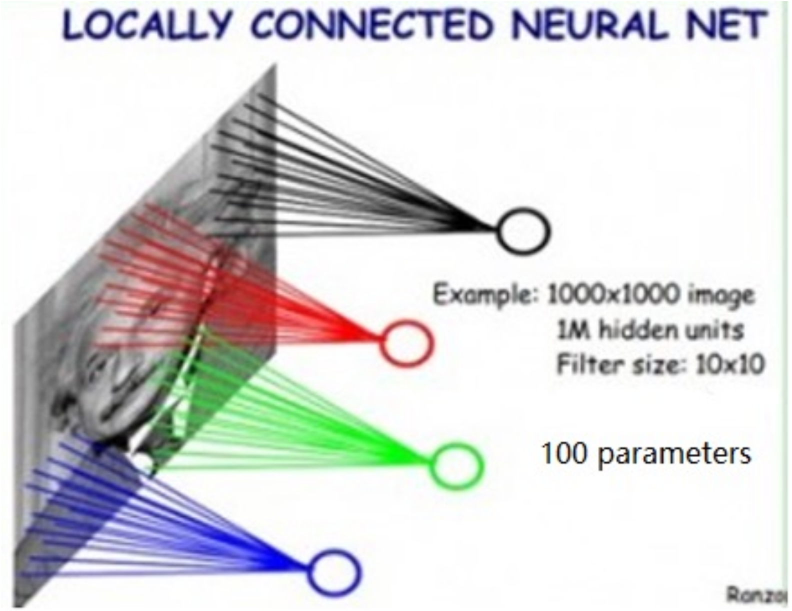




Fix issue: sharing weights

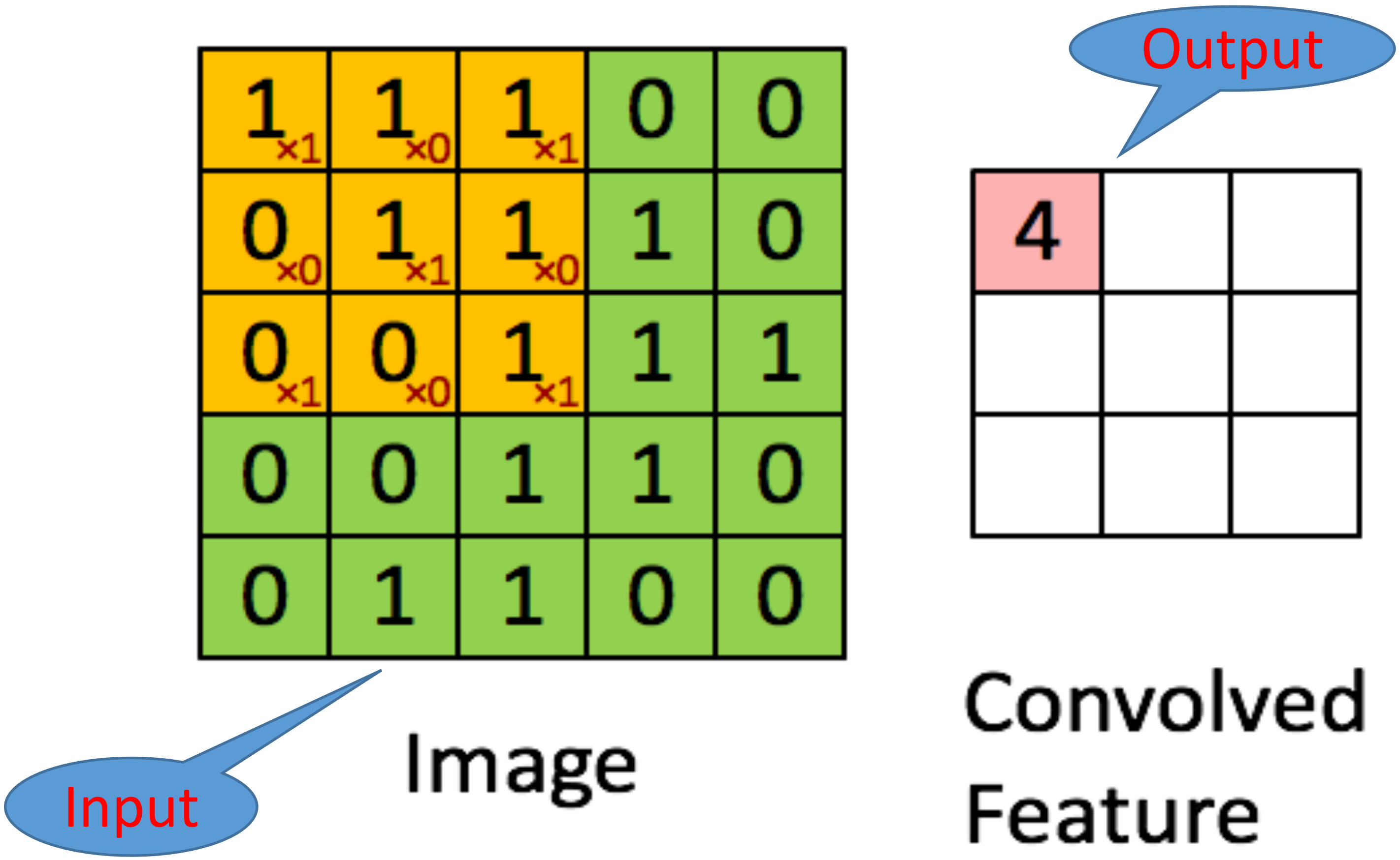
Insight: different local regions preserves identical statistical properties in images

Fix: different filters/kernels share a group of weights



Convolution

Local perception + Weight sharing = Convolution

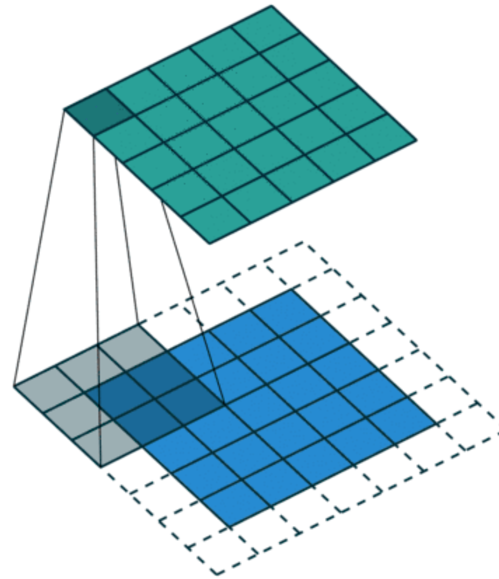


Convolution

- Filter size: k x k (3x3)
- Stride: s (1)
- Padding: p (1)
- w_{in}=5, h_{in}=5
- w_{out}=5, h_{out}=5

$$w_{out} = \lfloor (w_{in} - k + 2 \times p)/s + 1 \rfloor$$

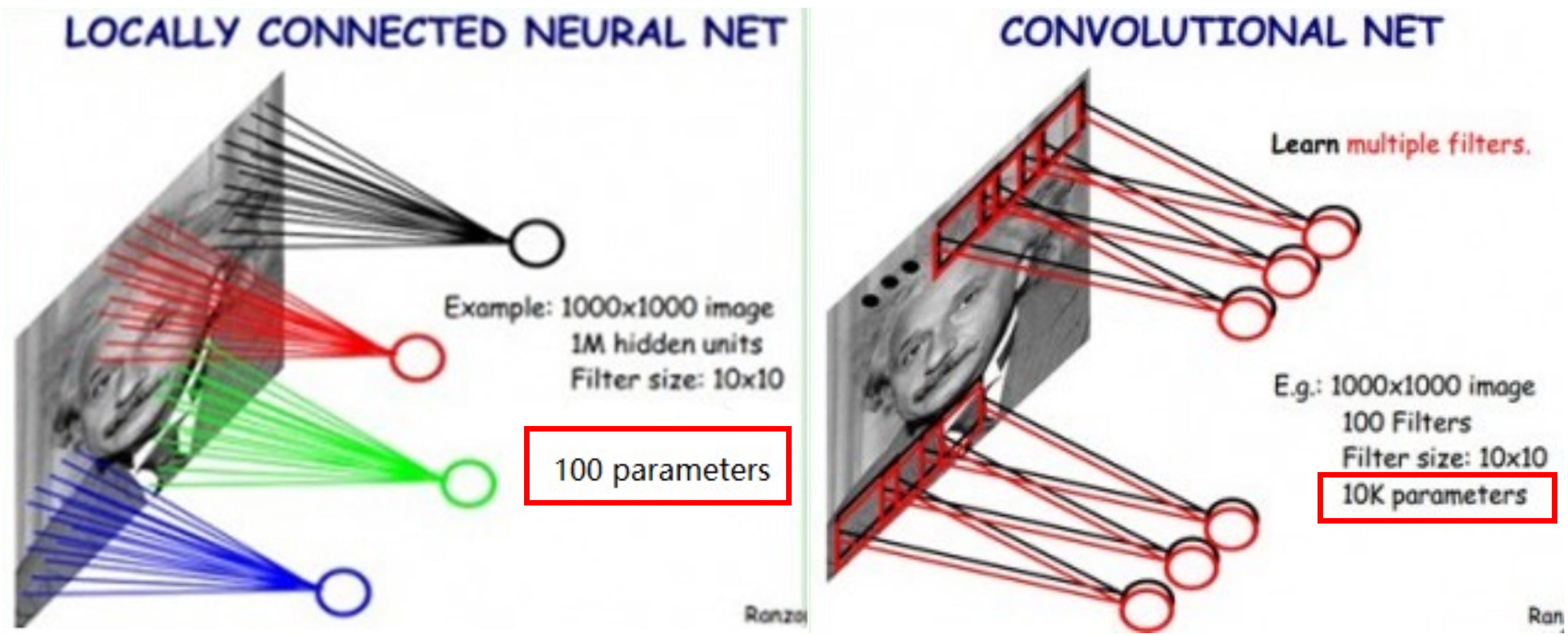
$$h_{out} = \lfloor (h_{in} - k + 2 \times p)/s + 1 \rfloor$$



Convolution with multiple filters

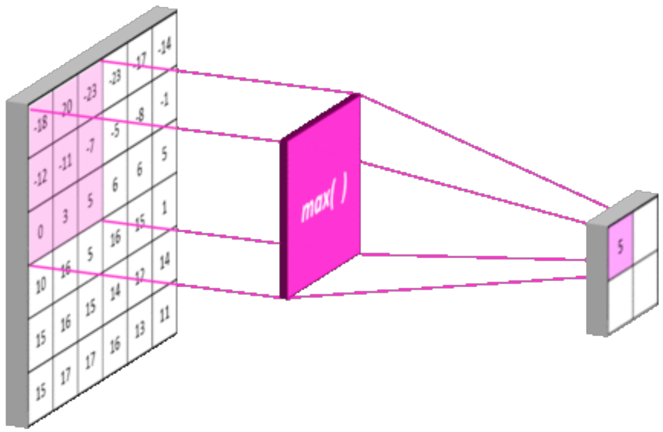
Problem: only one kernel is too weak in feature learning

Fix: convolution with multiple kernels



Feature map and pooling

Max-pooling: max(a,b,c…)   Average-pooling: sum(a,b,c…)/N



- Why pooling?
  - Another way to reduce the number of weights
  - Increase receptive field
  - Remove noise

Definition: CNN Receptive Field

The **receptive field** is defined as the region in the input space that a particular **CNN**'s feature is looking at (i.e., be affected by). A **receptive field** of a feature can be described by its center location and its size.

Why CNN for Image

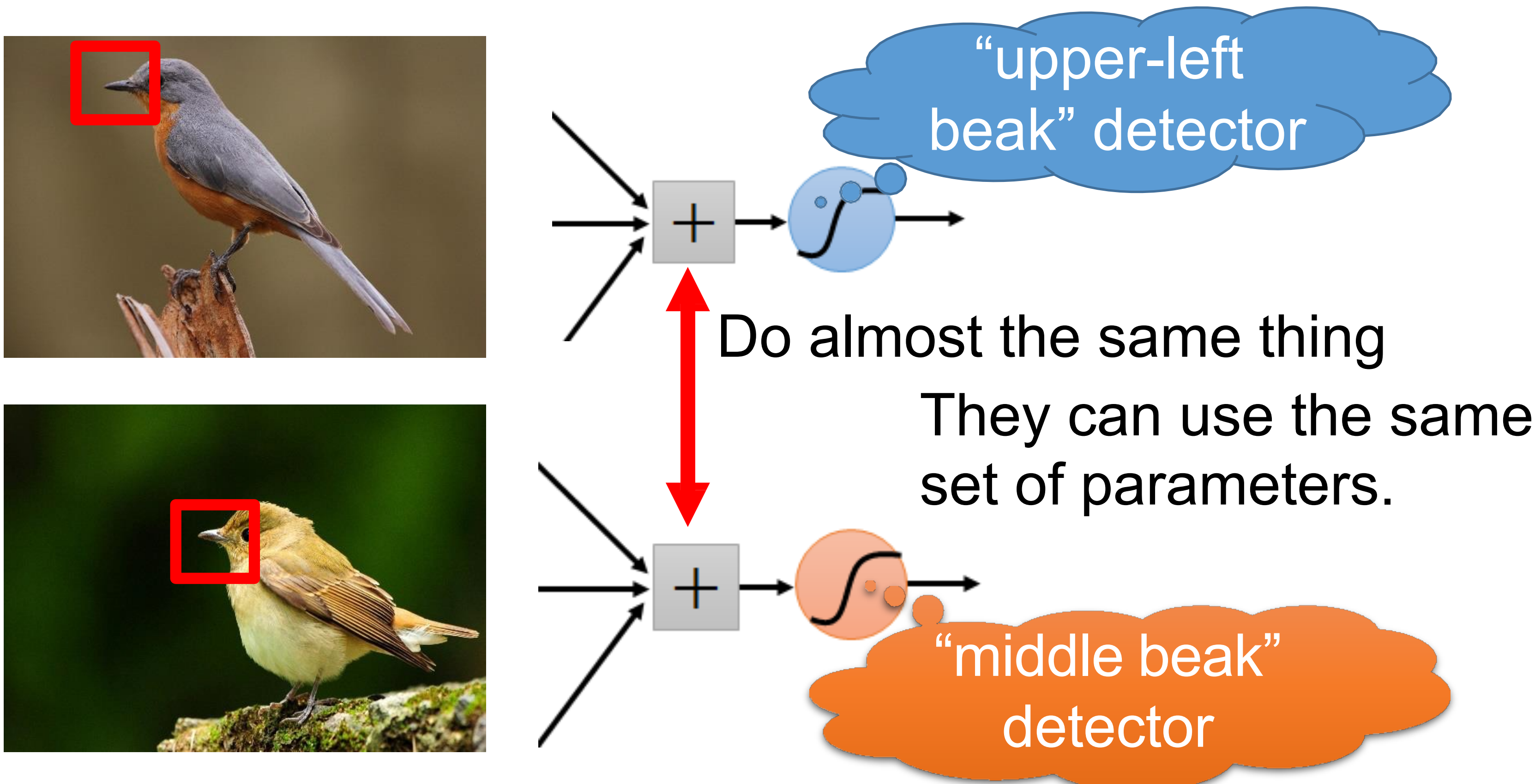
- Some patterns are much smaller than the whole image

A neuron does not have to see the whole image to discover the pattern.

Connecting to small region with less

Why CNN for Image

- The same patterns appear in different regions.



Why CNN for Image

- Subsampling the pixels will not change the object

bird

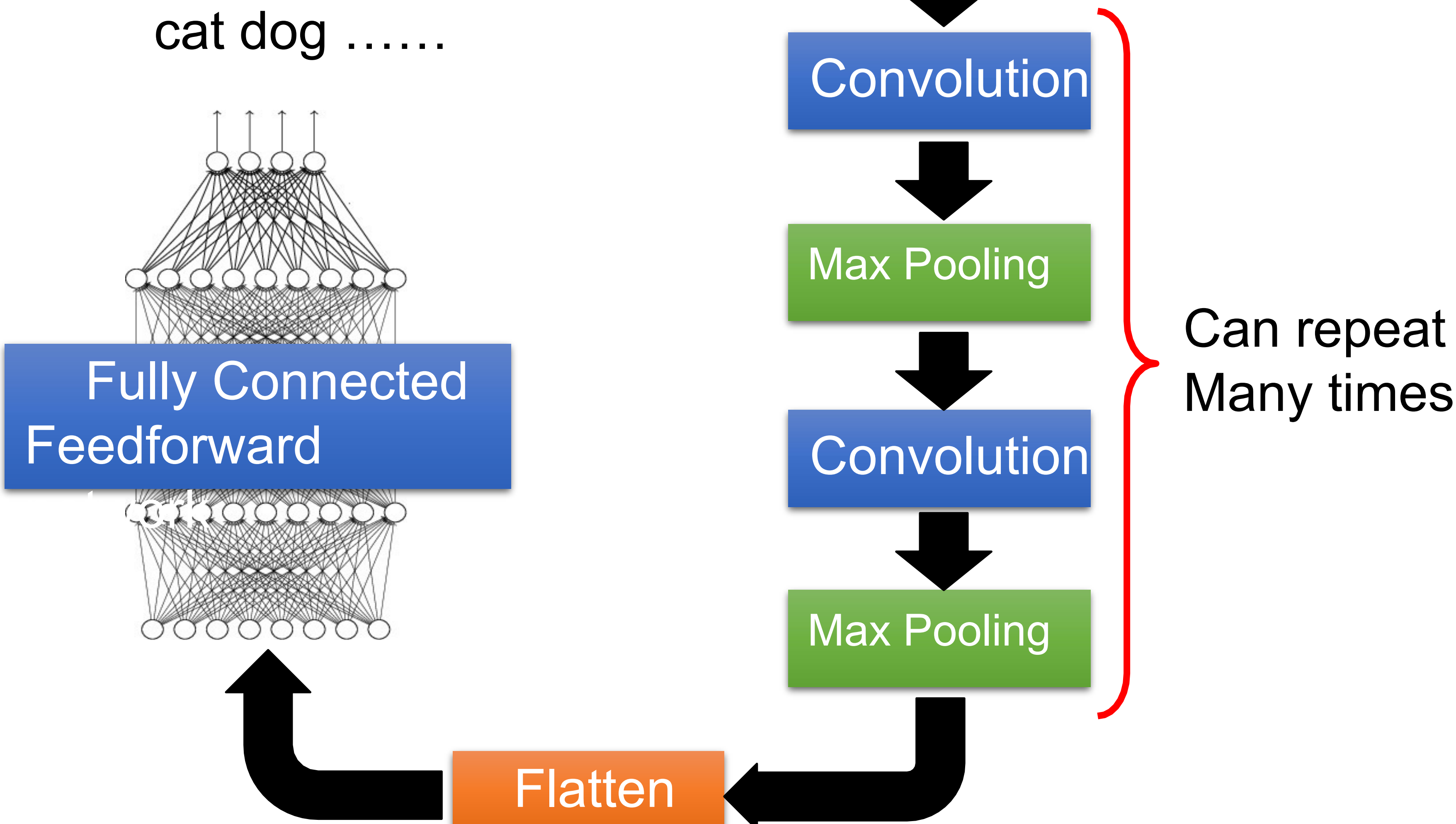
subsampling

bird

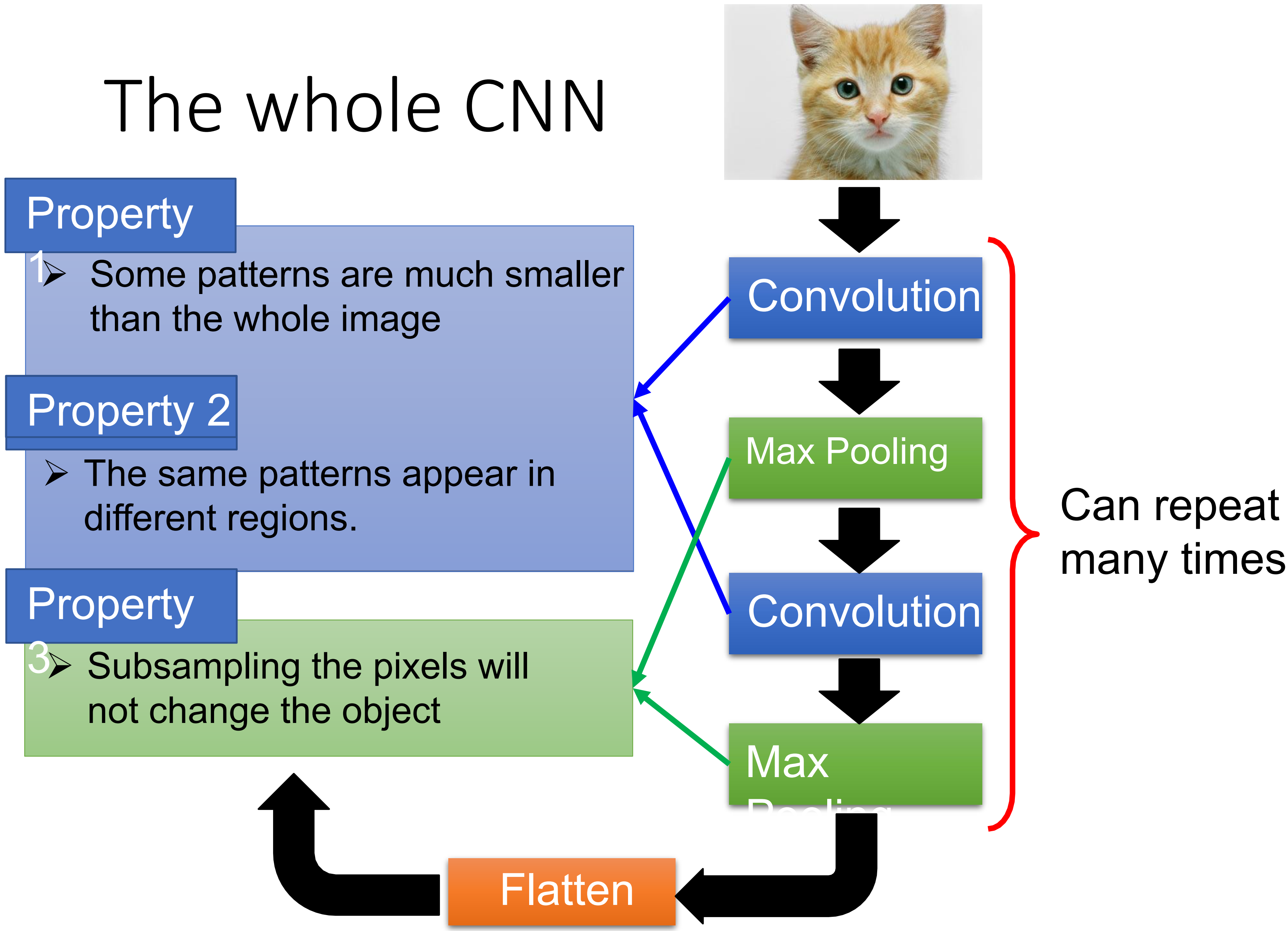
We can subsample the pixels to make image smaller

Less parameters for the network to process the image

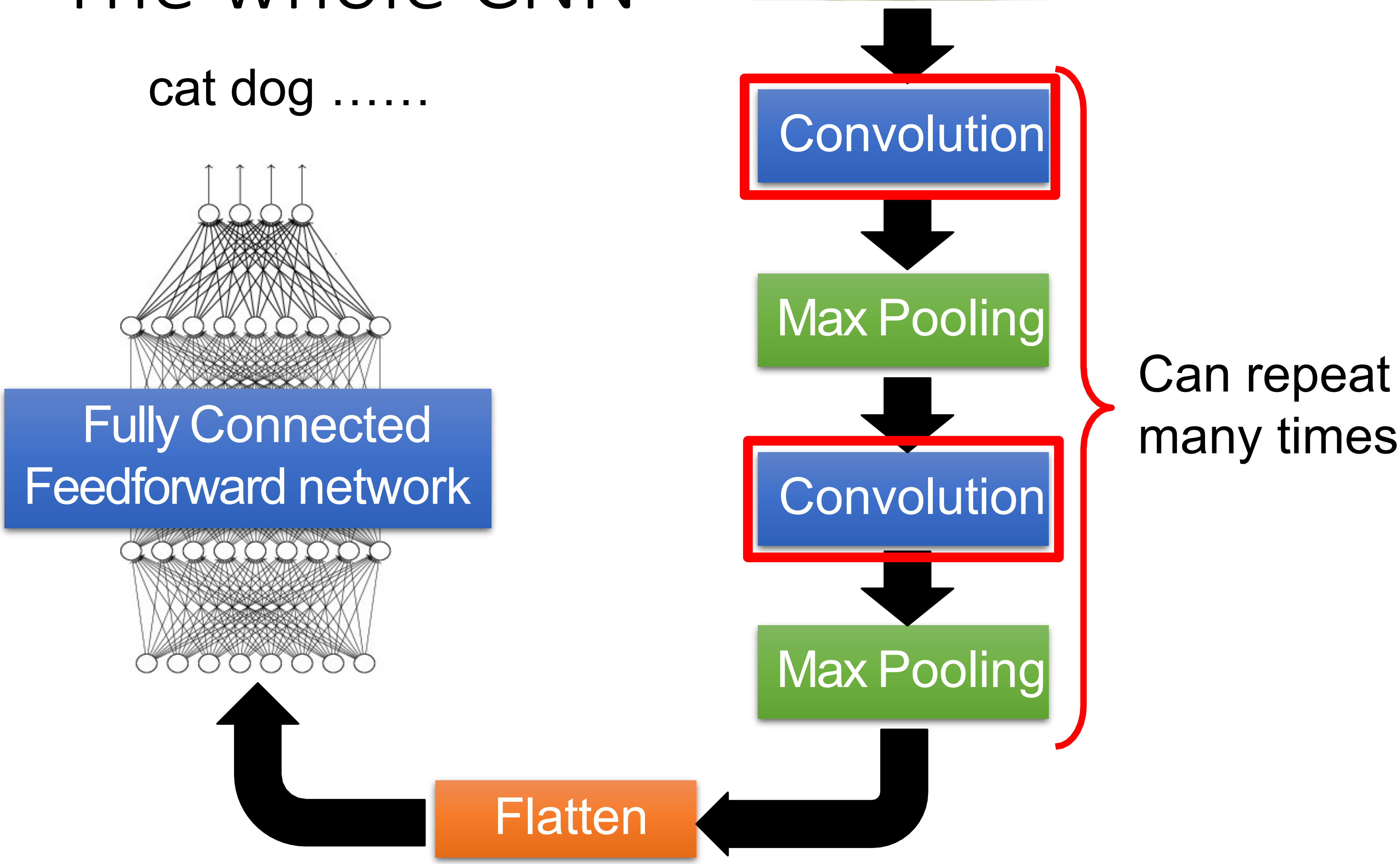
The whole CNN



The whole CNN



The whole CNN



CNN – Convolution

Those are the network parameters to be learned.

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

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

Matrix

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

Matrix

⋮ ⋮

Property 1

Each filter detects a small pattern (3 x 3).

CNN – Convolution

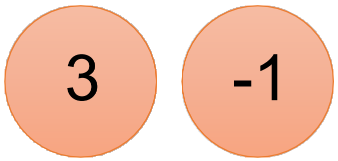
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

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

6 x 6 image



CNN – Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

If stride=2

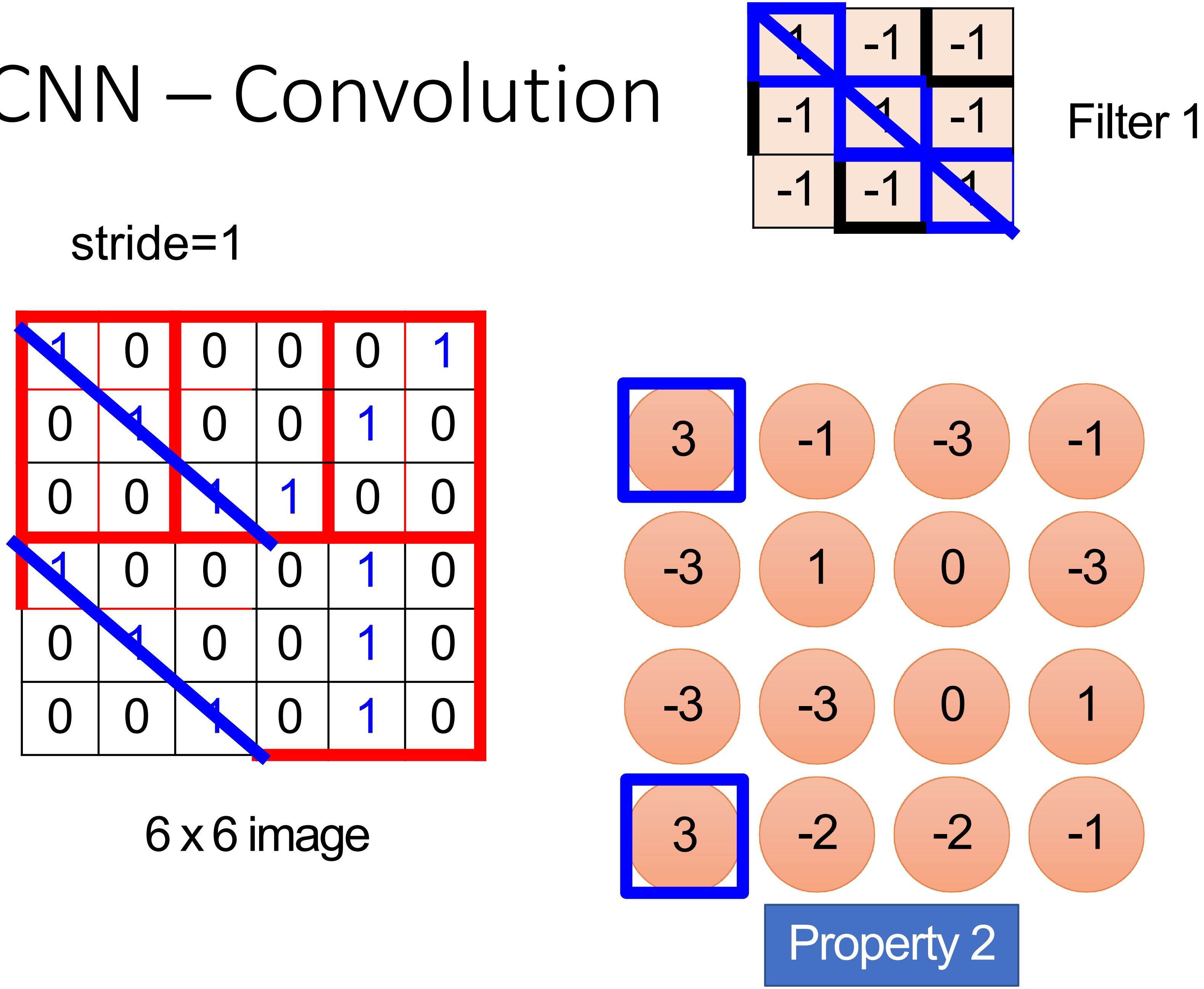
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3 -3

We set stride=1 below

CNN – Convolution



CNN – Convolution

-1	1	-1
-1	1	-1
-1	1	-1

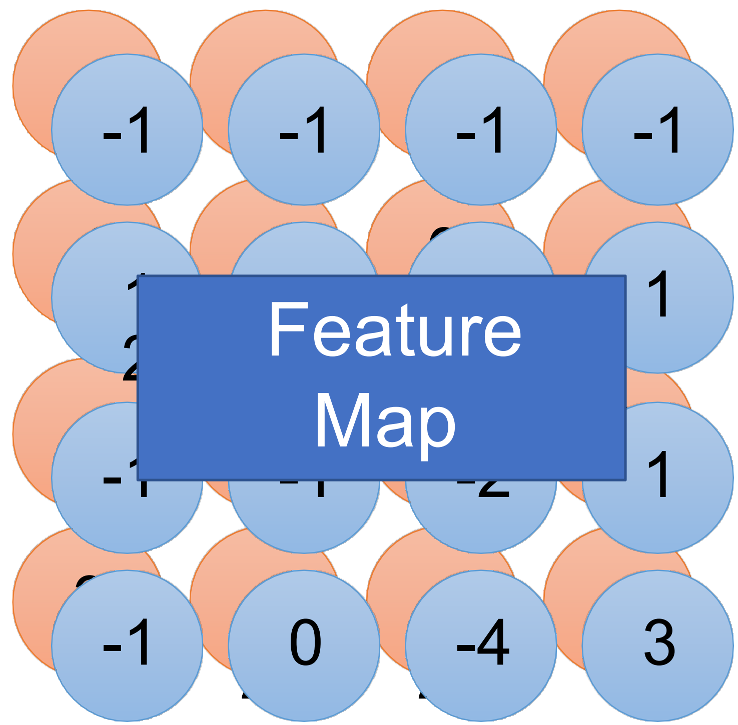
Filter 2

stride=1

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

6 x 6 image

Do the same process for every filter



4 x 4 image

CNN – Colorful image

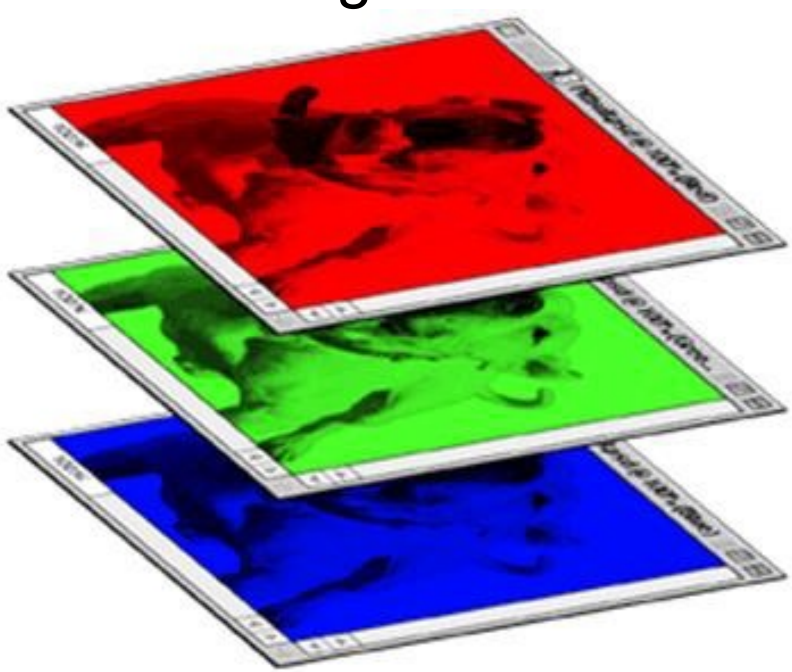
Filter 1

1	-1	-1
-1	1	-1
-1	-1	1

Filter 2

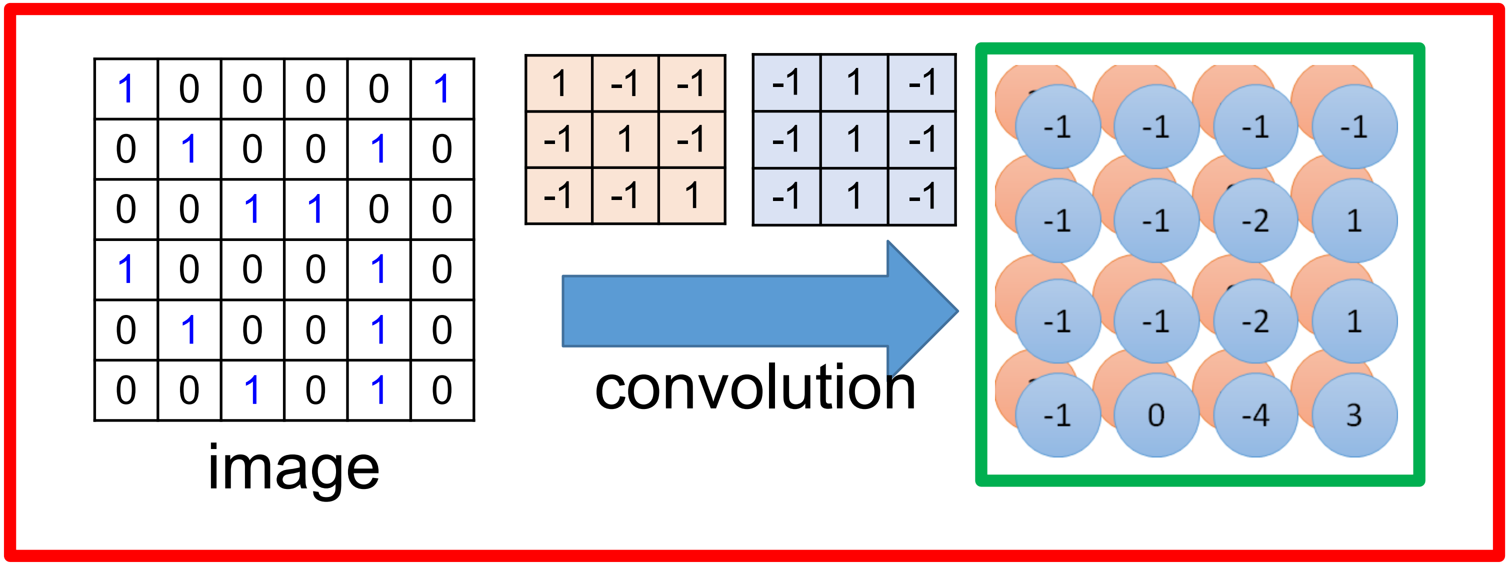
-1	1	-1
-1	1	-1
-1	1	-1

Colorful image

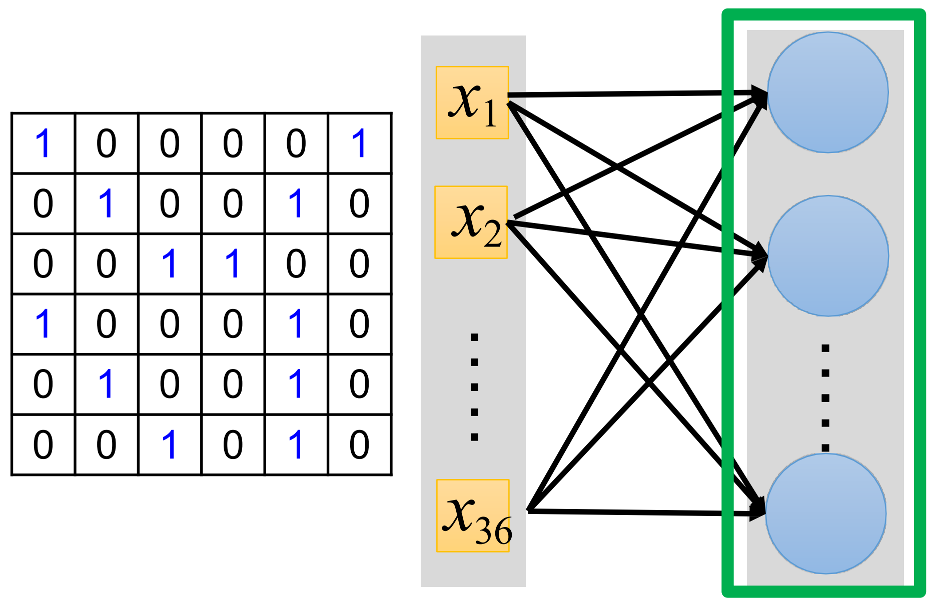


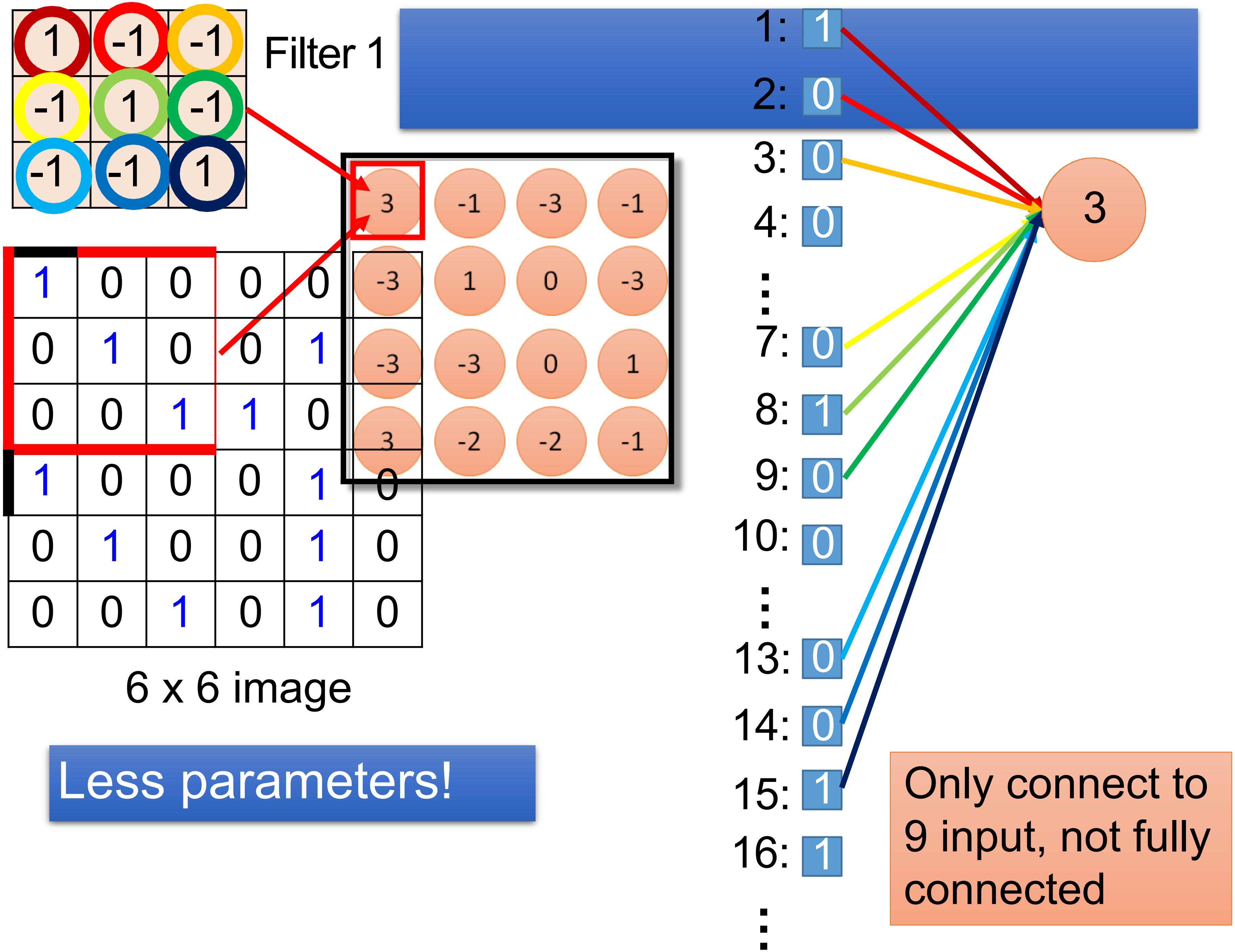
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

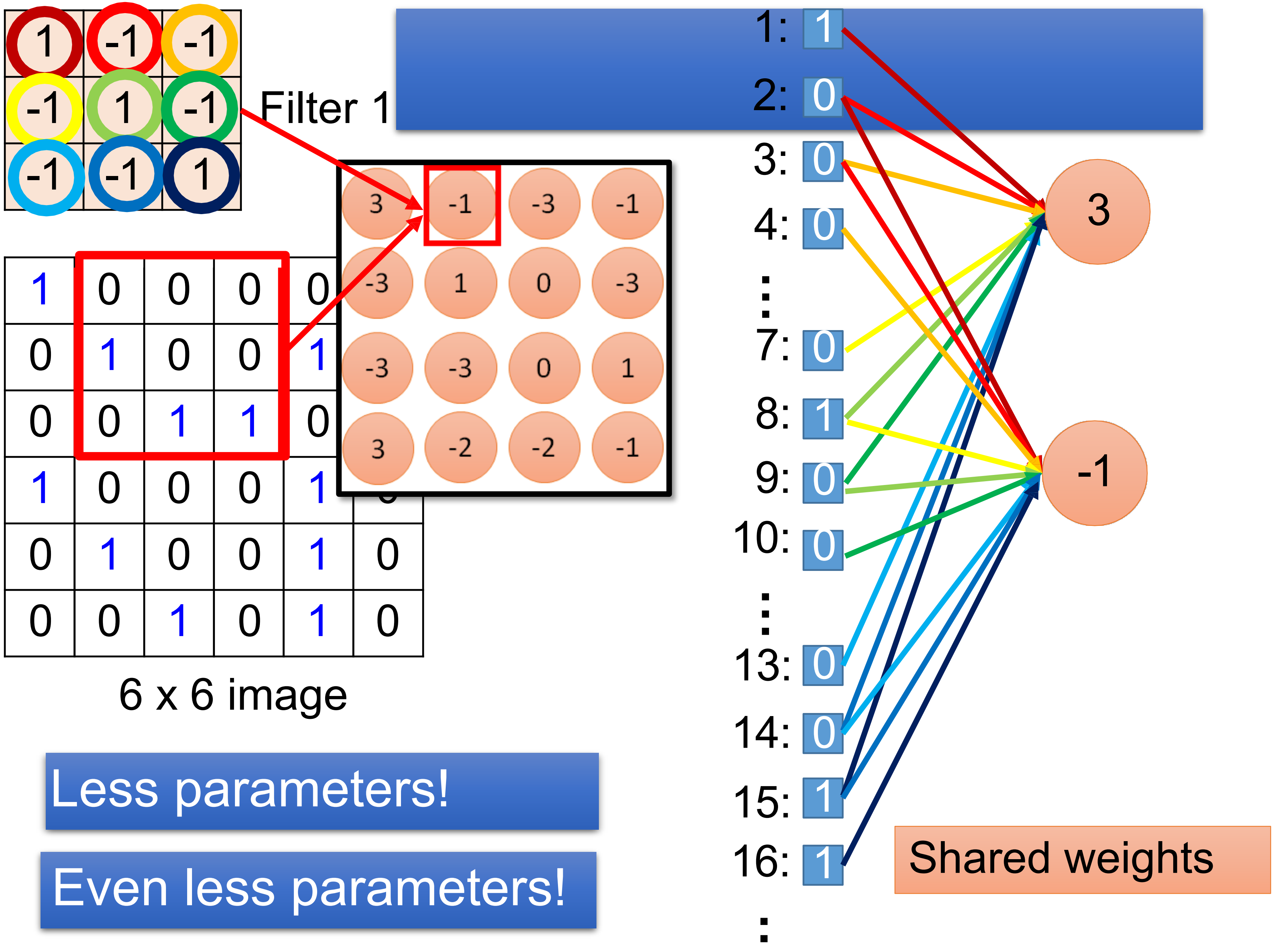
Convolution v.s. Fully Connected



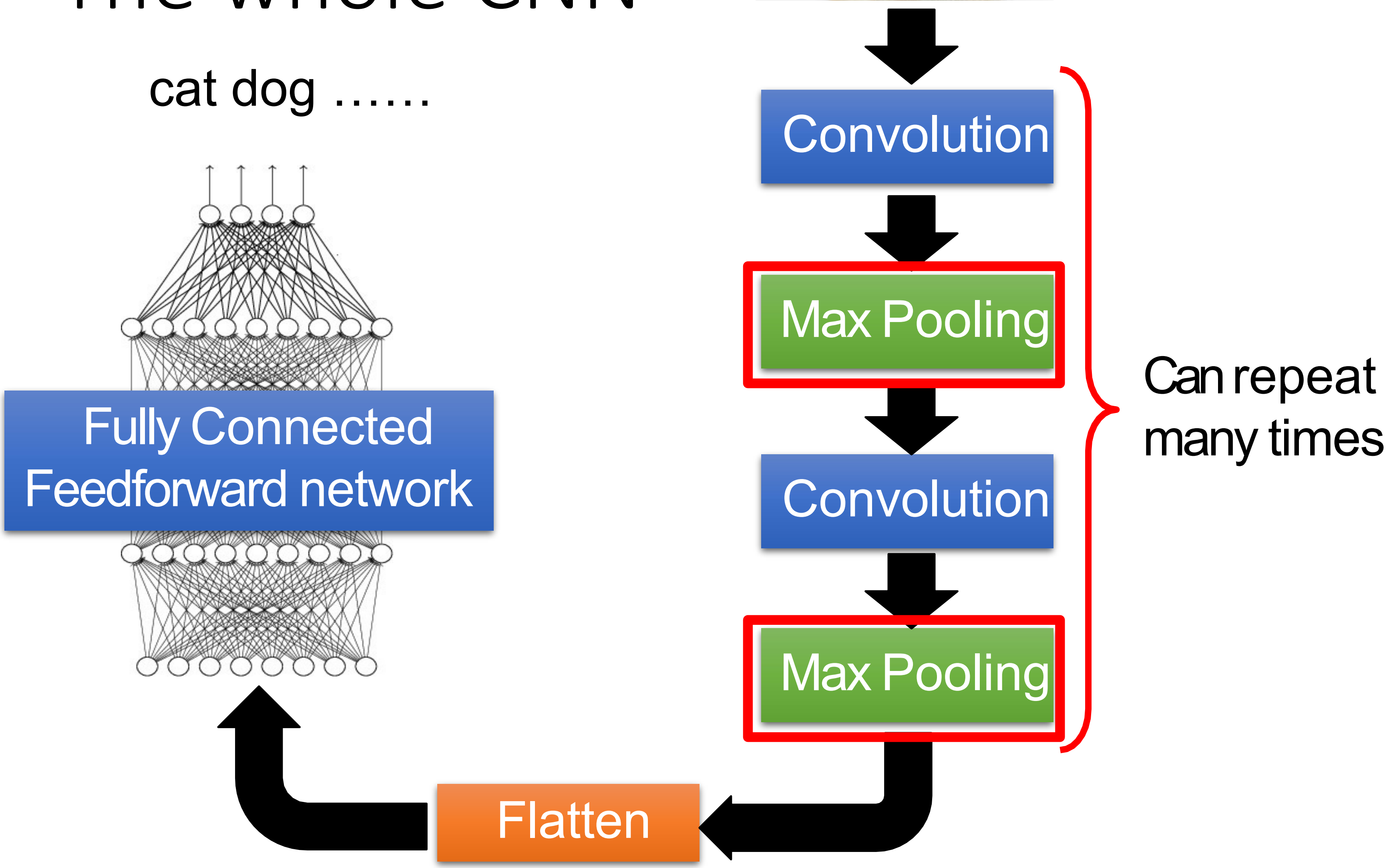
Fully-connected




Filter 1
1 -1 -1
-1 1 -1
-1 -1 1
3 -1 -3 -1
-3 1 0 -3
-3 -3 0 1
3 -2 -2 -1
1 0 0 0 0 1
0 1 0 0 1 0
0 0 1 1 0 0
1 0 0 0 1 0
0 1 0 0 1 0
0 0 1 0 1 0
6 x 6 image
Less parameters!
1: 1
2: 0
3: 0
4: 0
7: 0
8: 1
9: 0
10: 0
13: 0
14: 0
15: 1
16: 1
3
Only connect to 9 input, not fully connected


Filter 1
6 x 6 image
Less parameters!
Even less parameters!
Shared weights
1: 1
2: 0
3: 0
4: 0
7: 0
8: 1
9: 0
10: 0
13: 0
14: 0
15: 1
16: 1
3
-1

The whole CNN



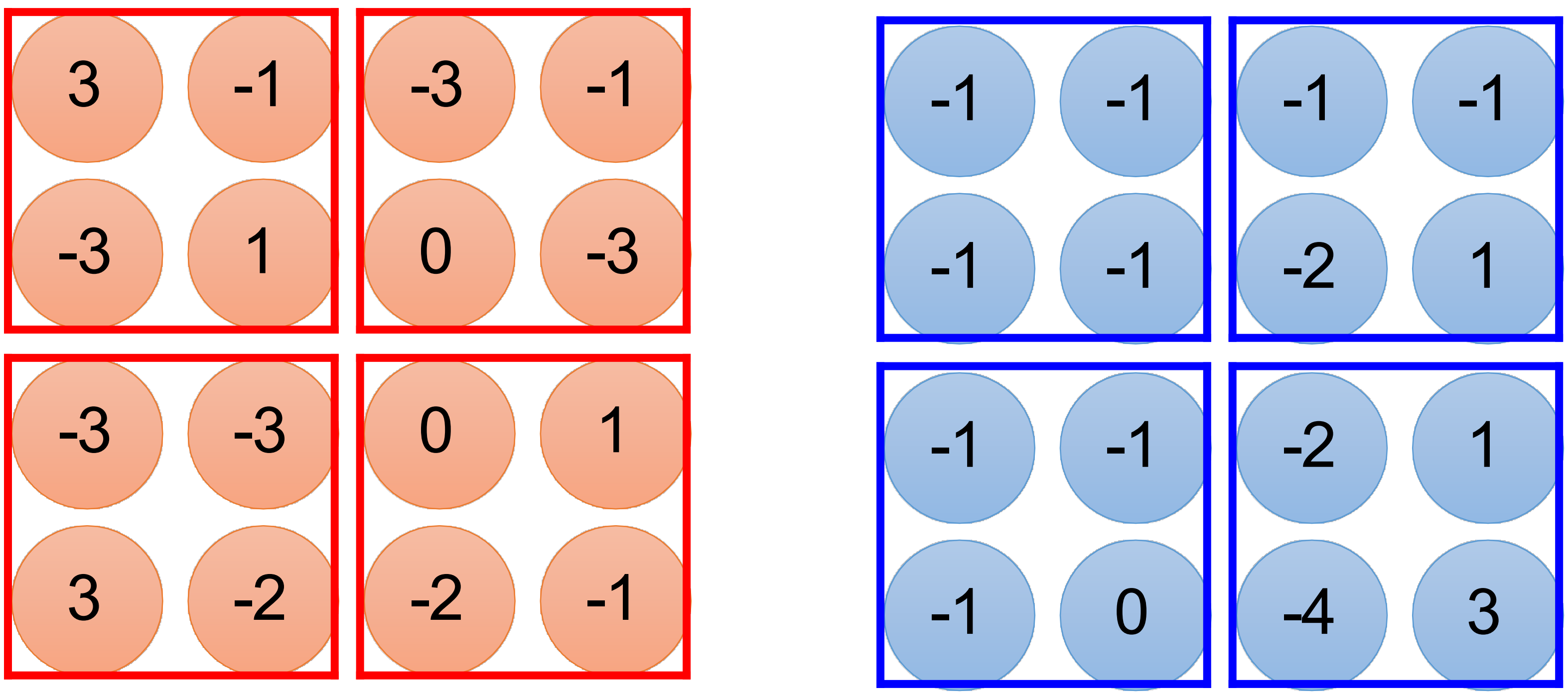
CNN – Max Pooling

1	-1	-1
-1	1	-1
-1	-1	1

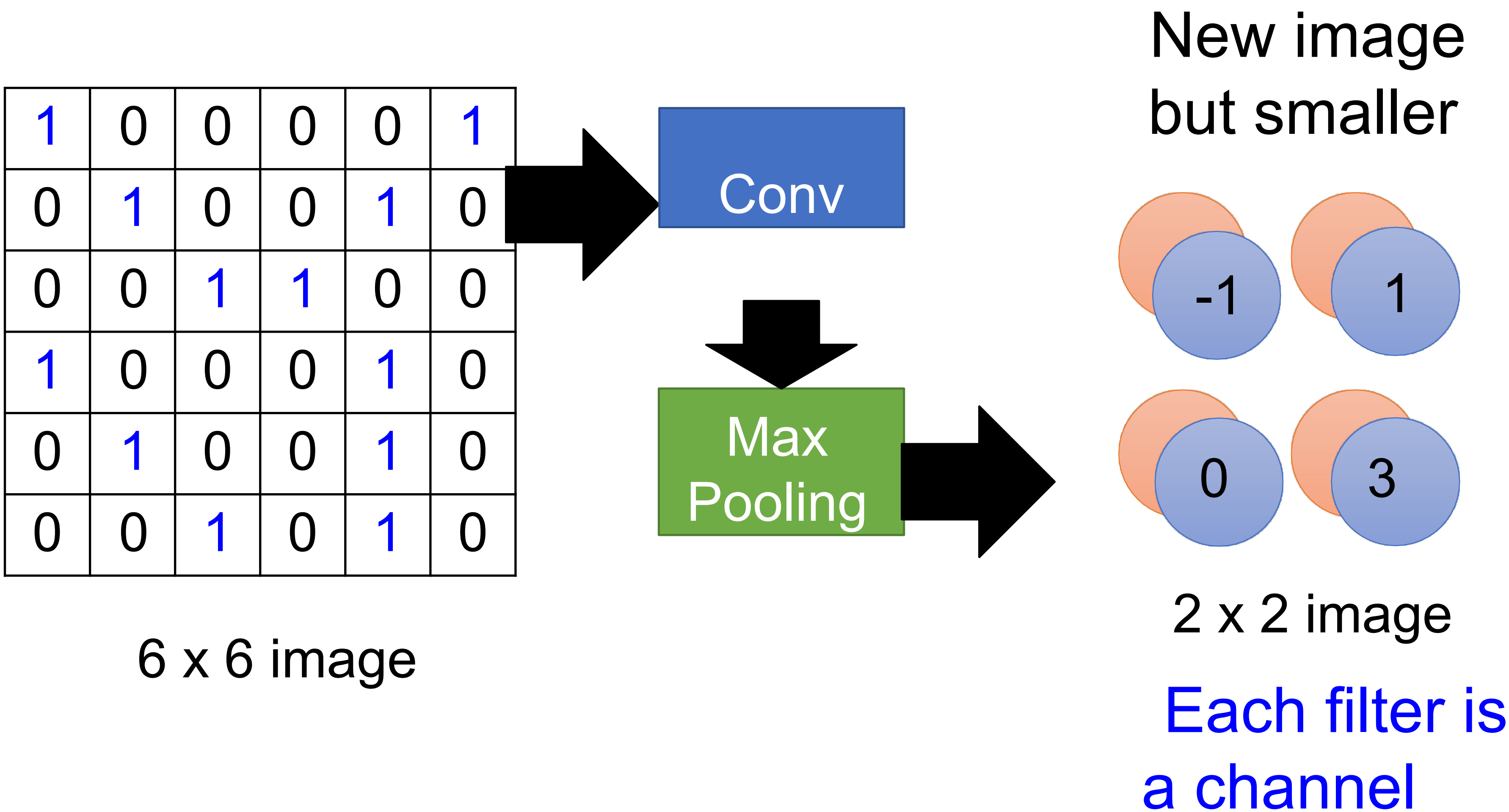
Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

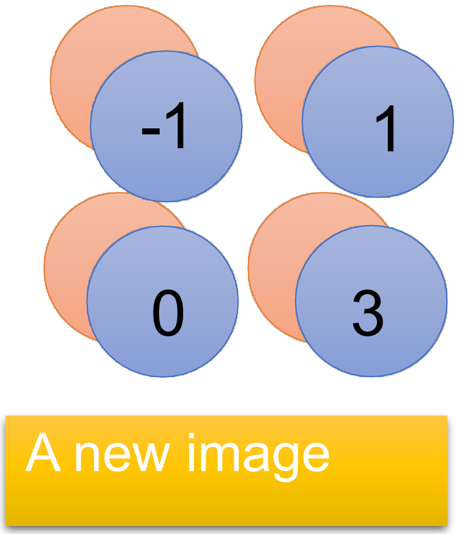
Filter 2



CNN – Max Pooling

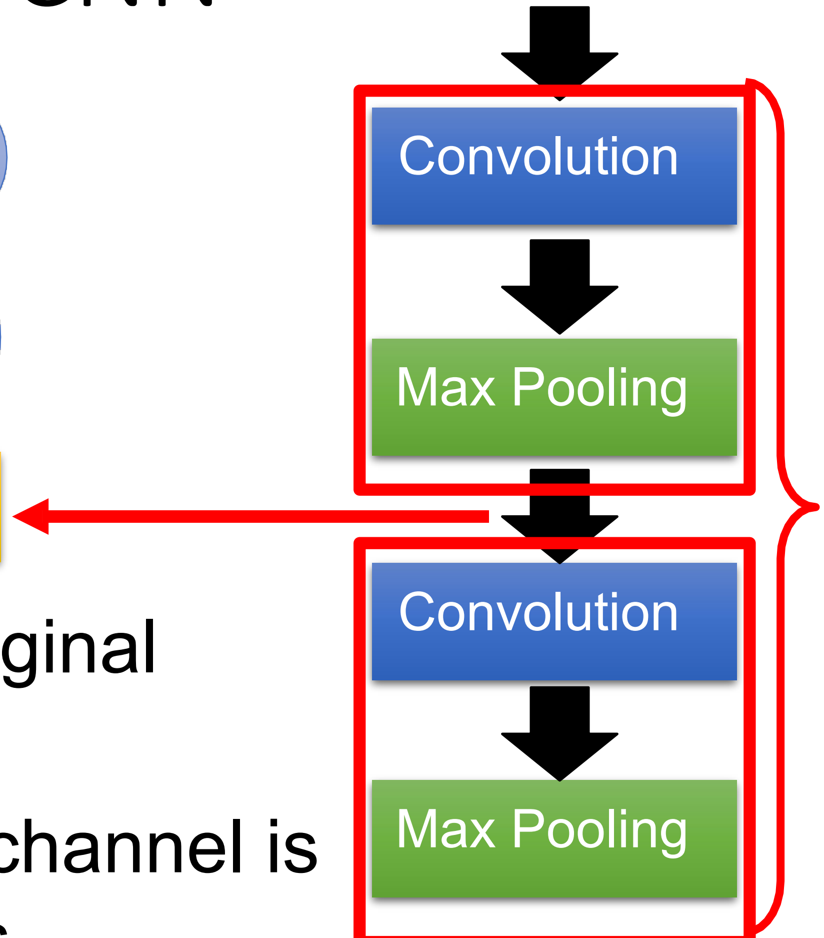


The whole CNN



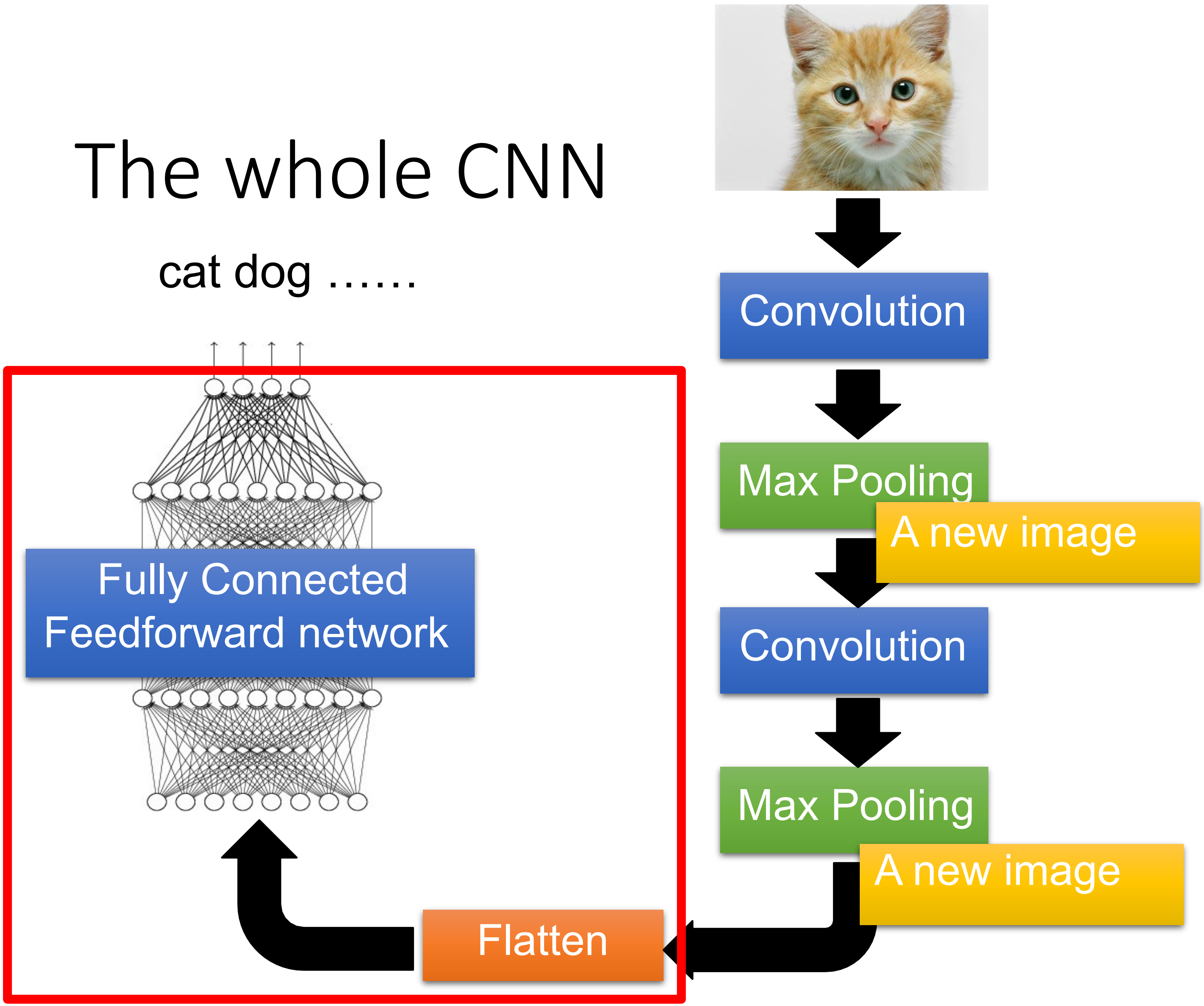
Smaller than the original image

The number of the channel is the number of filters

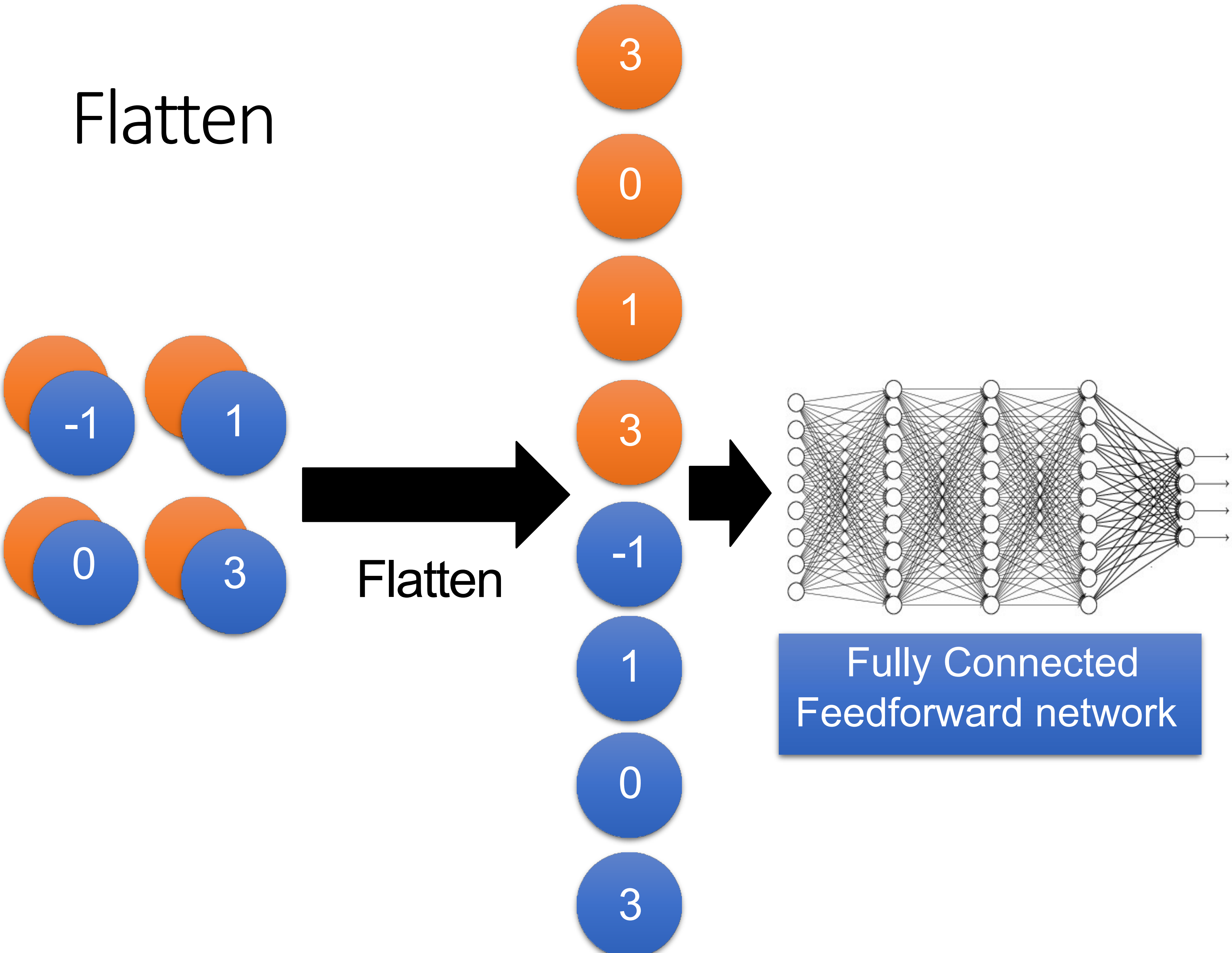


Can repeat many times

The whole CNN



Flatten

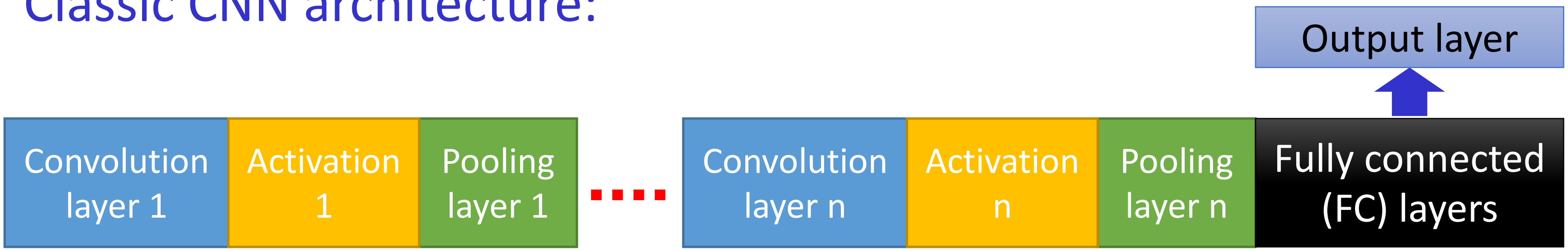


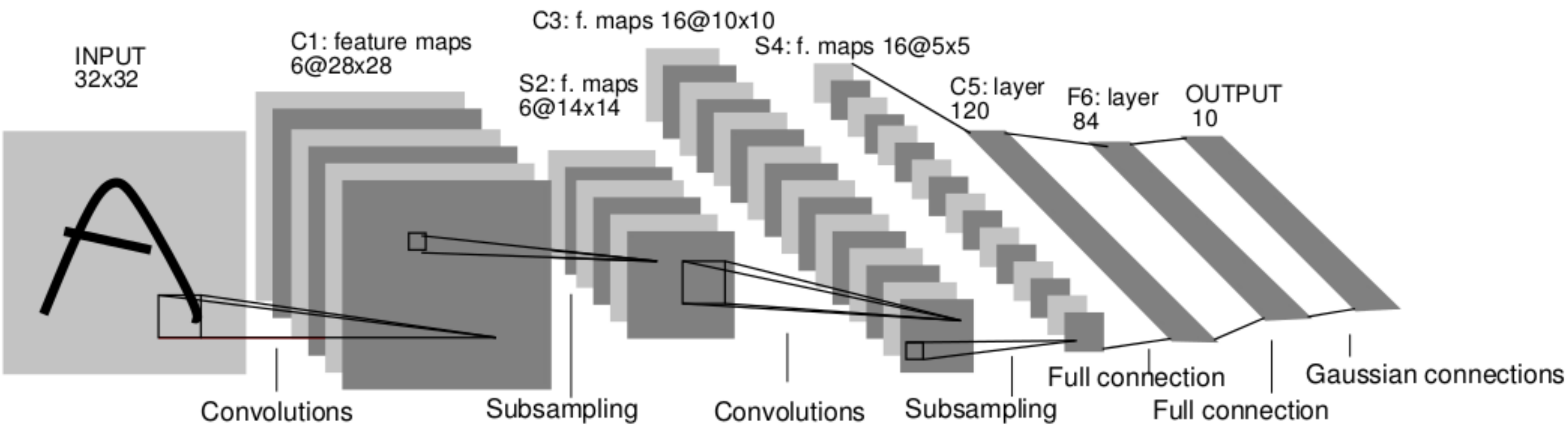
-1
1
0
3
Flatten
3
0
1
3
-1
1
0
3
Fully Connected
Feedforward network

CNN architecture as a whole

$$w_{out} = \lfloor (w_{in} - k + 2 \times p)/s + 1 \rfloor$$

Classic CNN architecture:





Question: how many parameters (weights) within the network?

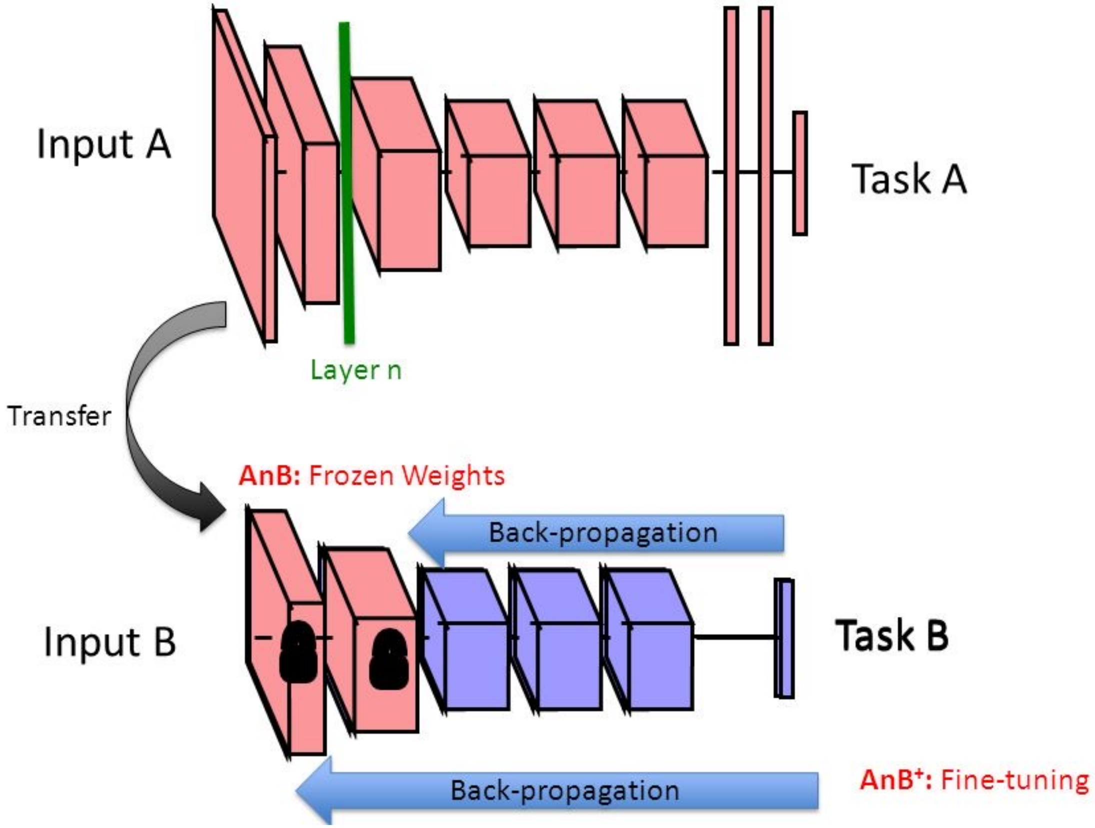
Suppose stride = 1, padding = 0, and do not use bias parameters

Train CNN

- The main computation of CNN training is calculating errors (recall back propagation)
- Three conditions:
  - If FC layer: the same as fully-connected network
  - If convolutional layer: see UFLDL
  - If pooling layer: UFLDL
- Open CNN library saves: PyTorch, Caffe, Tensorflow etc.

Train CNN with transfer-learning

Transfer weights from task A to task B



Tips for transfer learning

- Insight: features become more and more specific to the task from 1st layer to last layer
- Transfer learning strategies:

B size vs. A size	B task vs. A task	Transfer learning
small	similar	Just fine-tune FC layers
small	distinct	Train SVM classifier with features from beginning layers
large	similar	Fine-tune all layers
large	distinct	Train from scratch or fine-tune all layers

- Using small learning rates for transfer learning

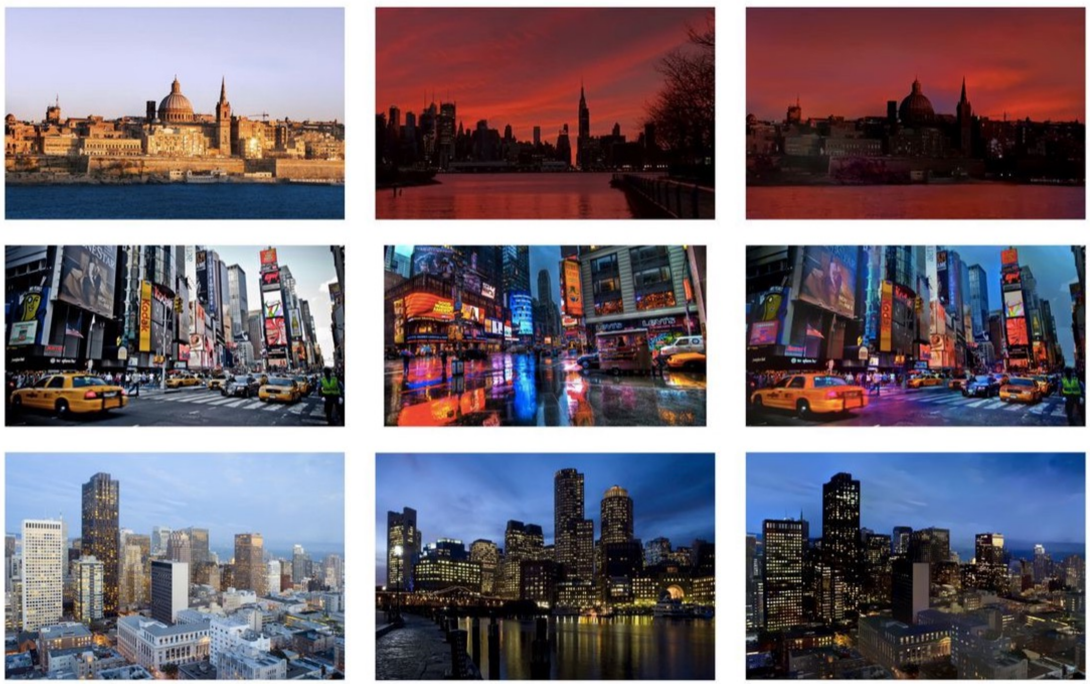
What if A and B images have different sizes?

- Solution 1: scale B images to the same size of A
- Solution 2: no scaling, but to modify the sizes of stride and/or pooling (without modifying the sizes of convolutional kernels)

Transfer learning with data augmentation

Data augmentation: random scaling, rotating, horizontal flipping, RGB jittering, style transfer with GAN

Useful when training data is limited

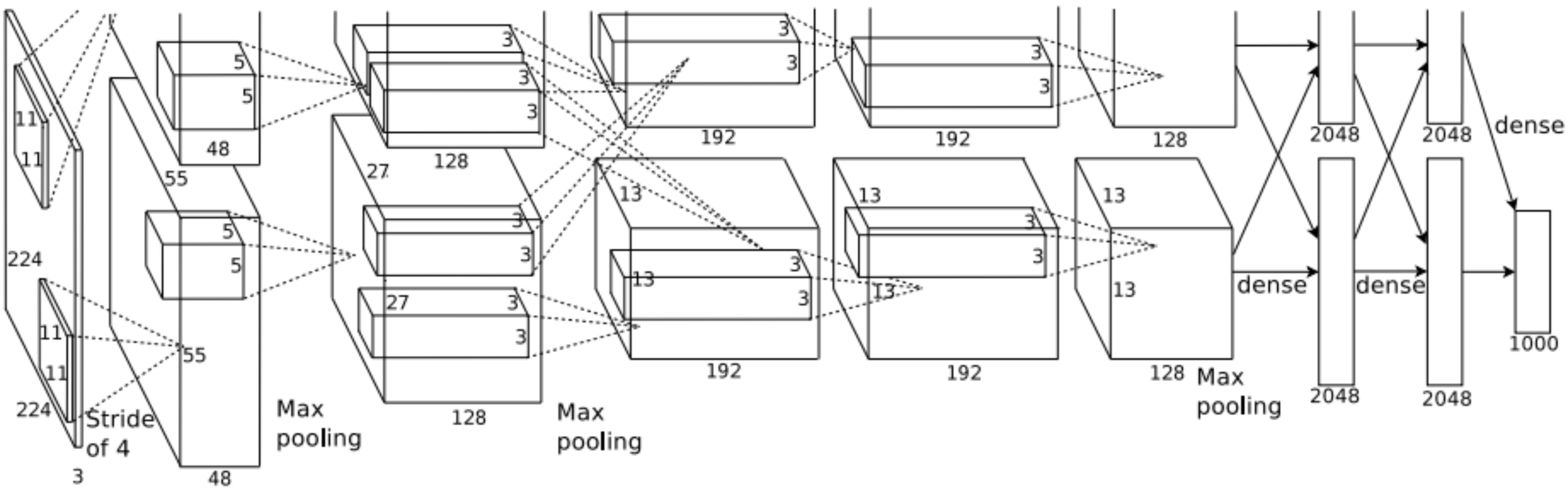


Original photo Reference photo Result

Roadmap

- Fully-connected neural network
- Convolutional neural network (CNN)
- Popular CNN architectures

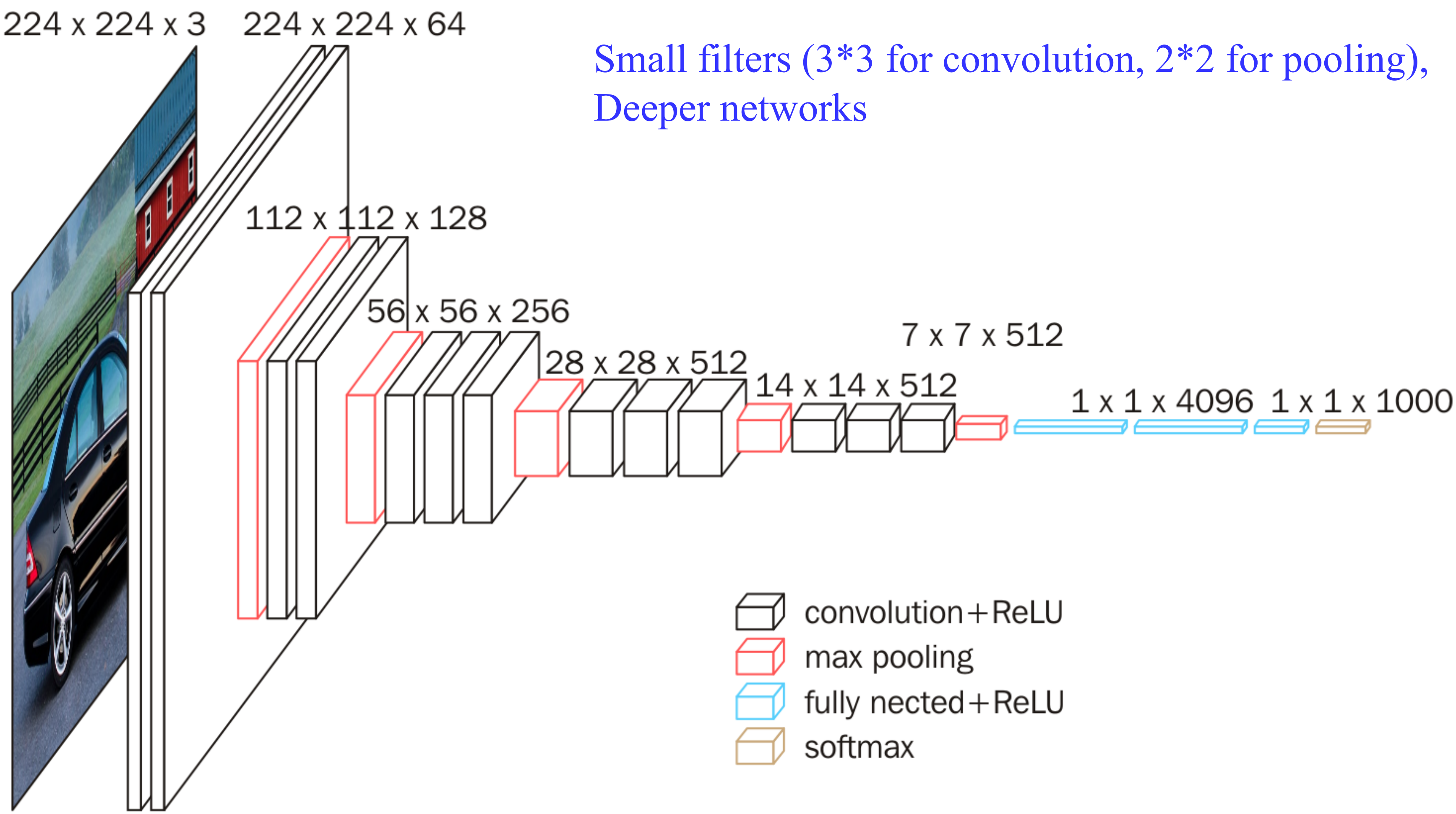
AlexNet



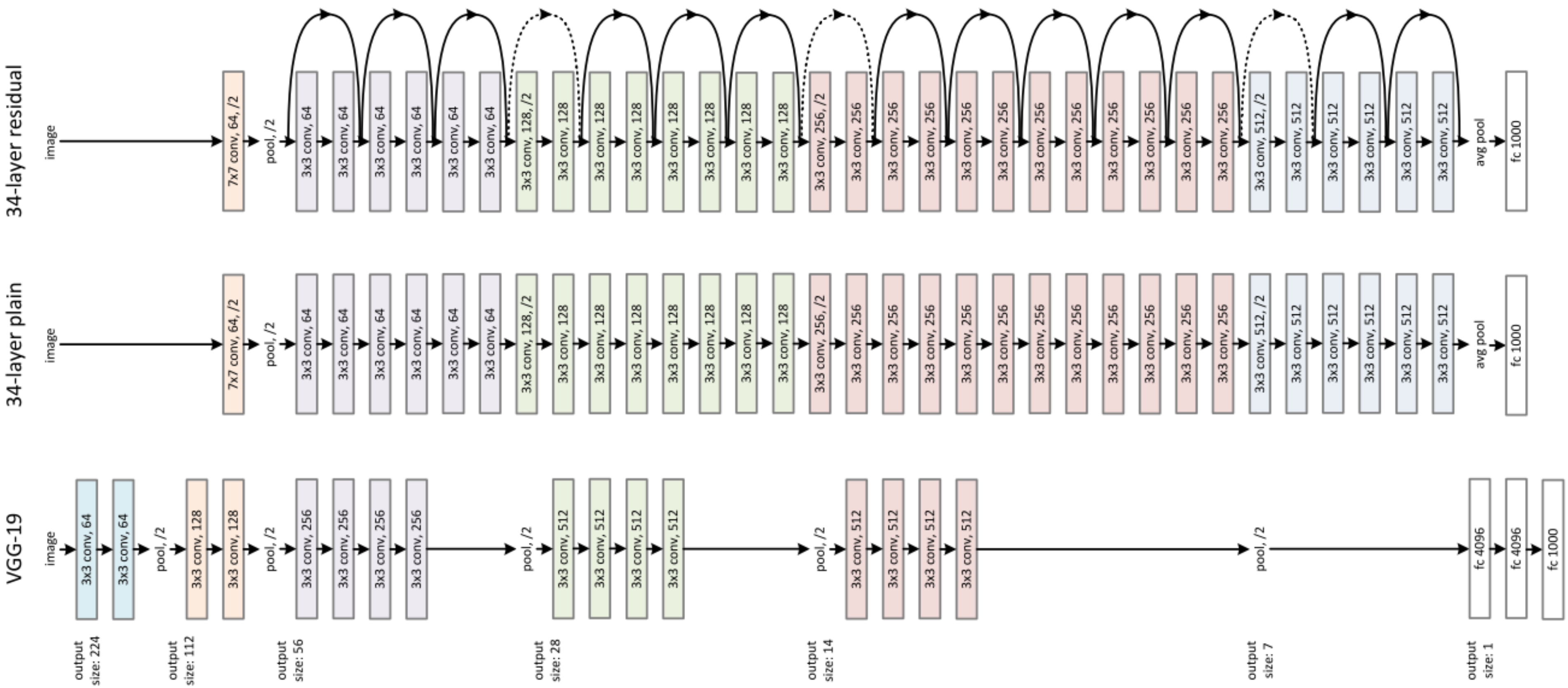
By Alex Krizhevsky

VGG 16

Small filters (3*3 for convolution, 2*2 for pooling),
Deeper networks



ResNet



Resnet vs. plain and VGG 19

Summary

- CNN can tackle images effectively and efficiently
- Train CNN is a breeze with open libraries
- Transfer learning is important to deal with small training sets

Acknowledgement

Reference and thanks to:

- **Sandford University CS221 Course:**

Artificial Intelligence: Principles and Techniques

https://stanford-cs221.github.io/autumn2022/

National Taiwan University ML2020 Course:

Machine Learning

https://speech.ee.ntu.edu.tw/~hylee/ml/2020-spring.php