Deep Learning IC Design

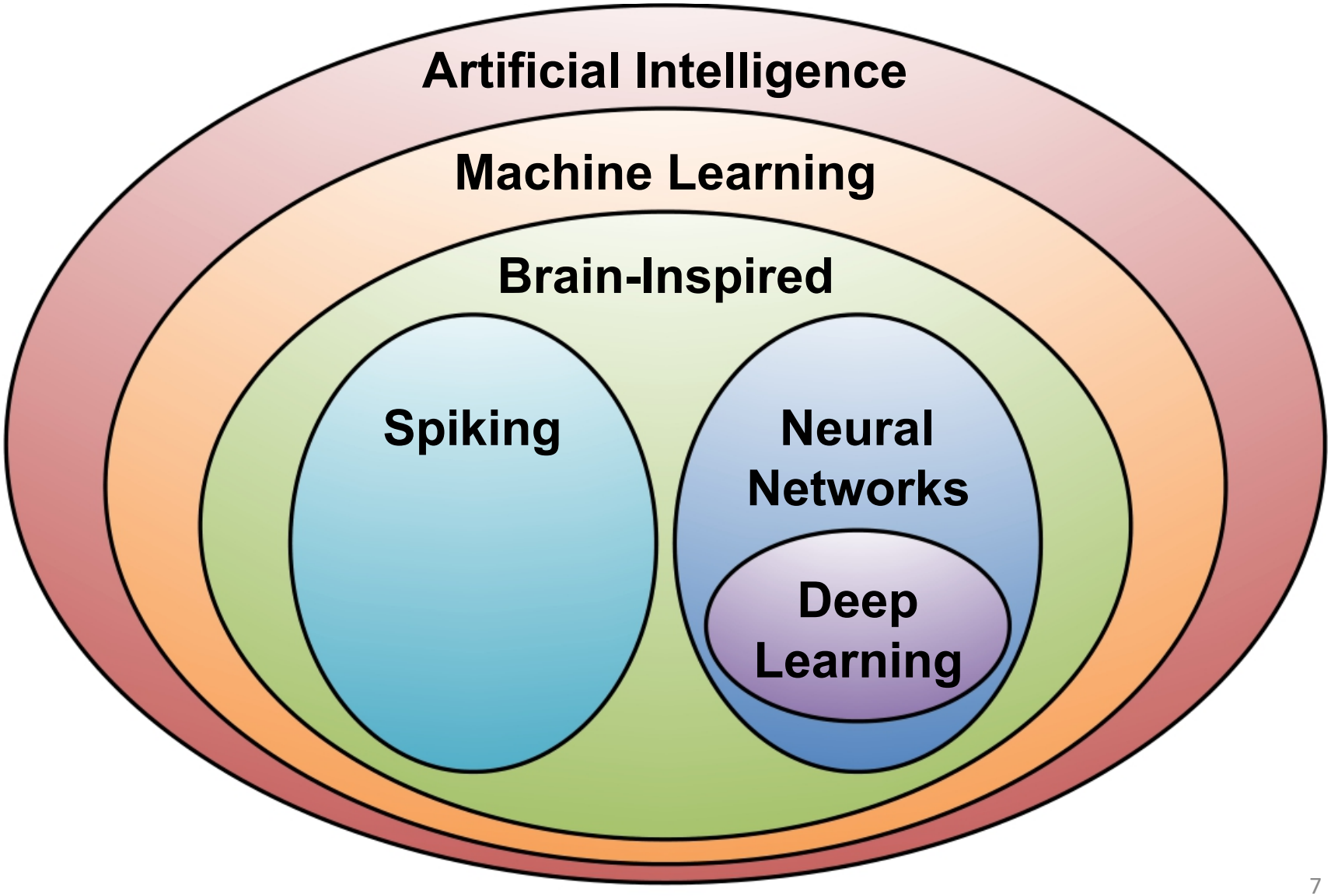
Outline

- **Overview of Deep Neural Networks**
- **DNN Development Resources**
- **Survey of DNN Hardware**
- **DNN Accelerators**
- **DNN Model and Hardware Co-Design**

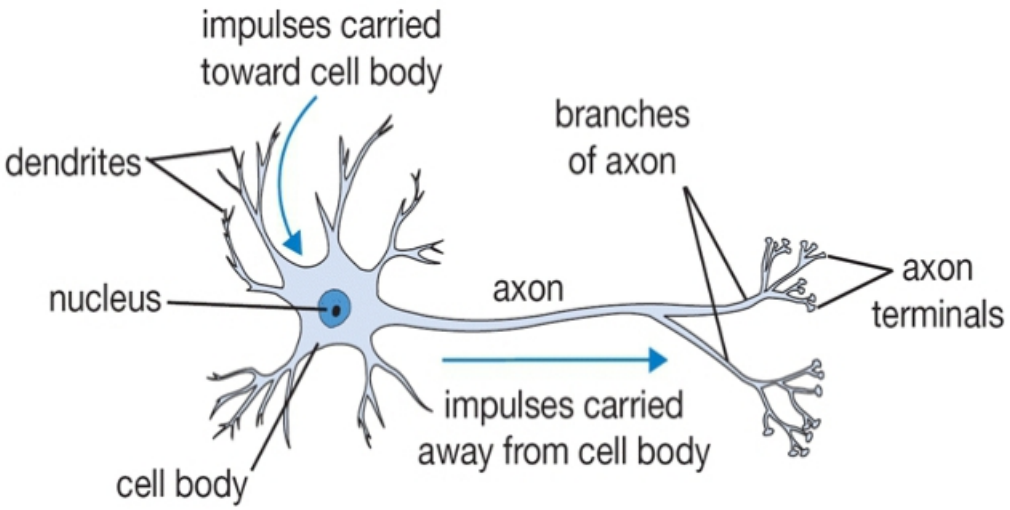
Slides are modified from ISCA 2017 tutorial by Vivienne Sze
Website: http://eyeriss.mit.edu/tutorial.html

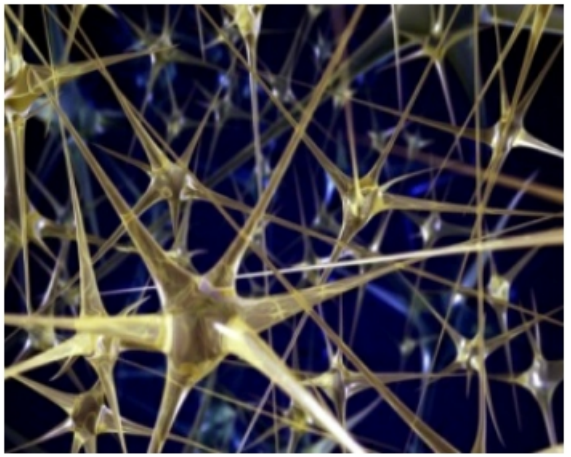
Background of Deep Neural Networks

Deep Learning



How Does the Brain Work?





- The basic computational unit of the brain is a **neuron**
  →86B neurons in the brain
- Neurons are connected with nearly **10^{14} – 10^{15} synapses**
- Neurons receive input signal from **dendrites** and produce output signal along **axon**, which interact with the dendrites of other neurons via **synaptic weights**
- Synaptic weights – learnable & control influence strength

Image Source: Stanford

Neural Networks: Weighted Sum

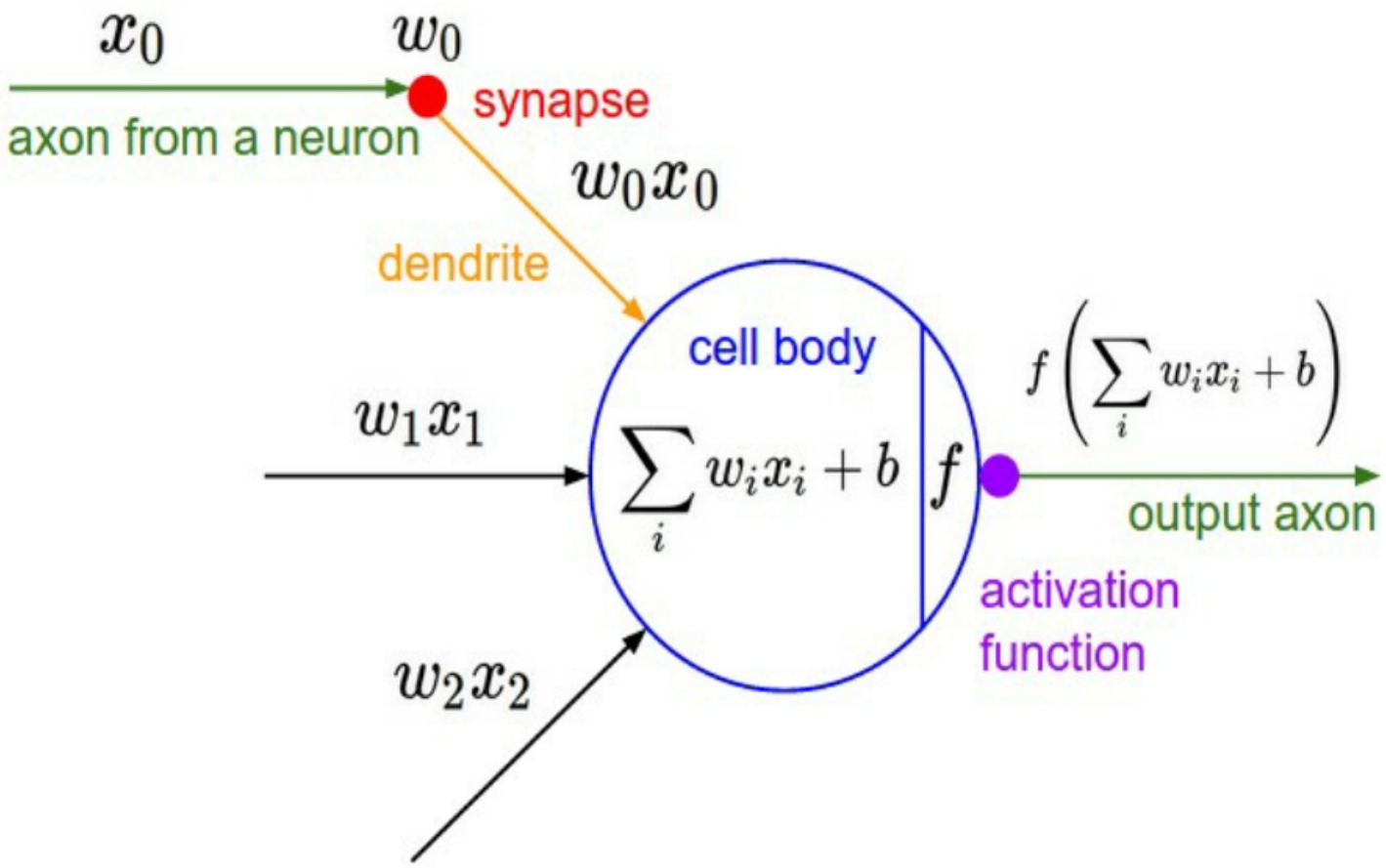


Image Source: Stanford

Many Weighted Sums

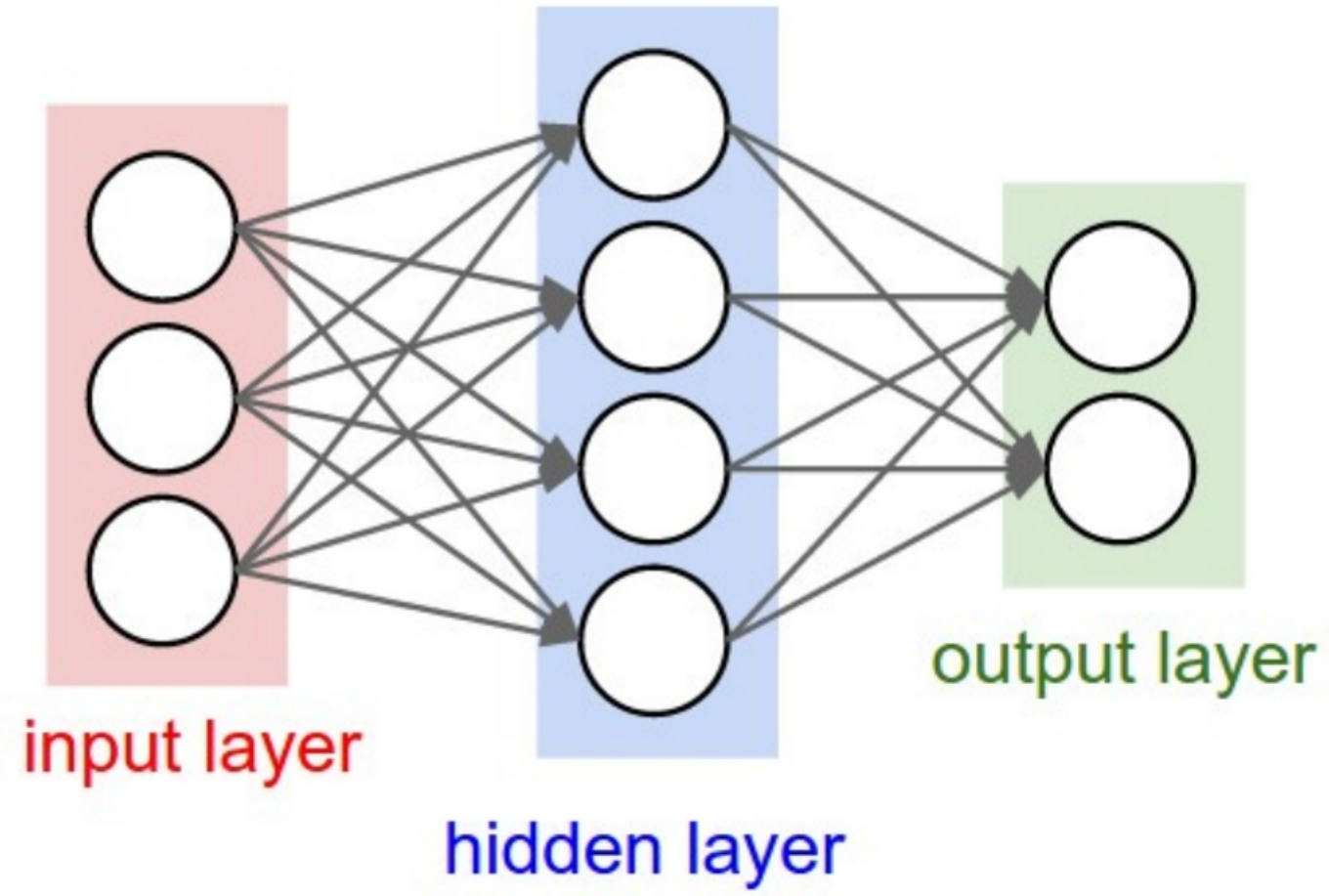


Image Source: Stanford

What is Deep Learning?

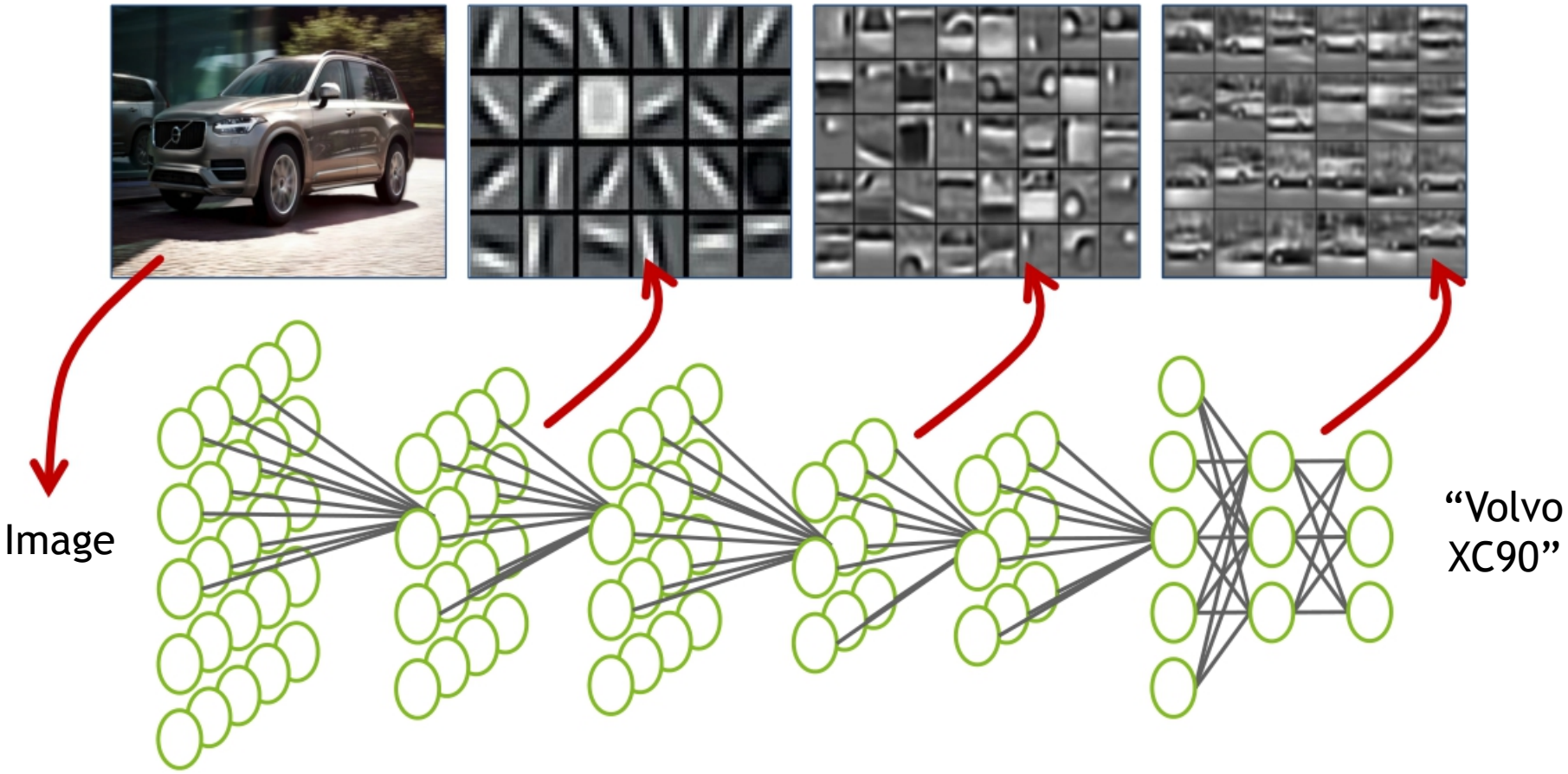


Image Source: [Lee et al., Comm. ACM 2011]

Why is Deep Learning Hot Now?

Big Data Availability

GPU Acceleration

New ML Techniques

facebook **350M** images uploaded per day

Walmart **2.5 Petabytes** of customer data hourly

YouTube **300 hours** of video uploaded every minute

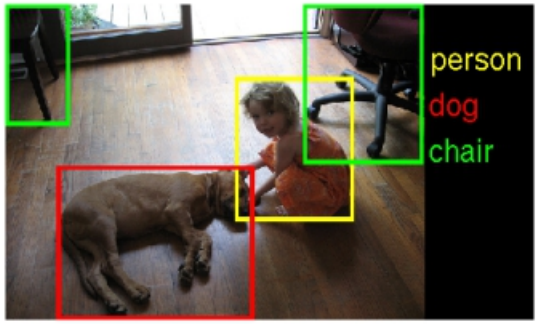
ImageNet Challenge

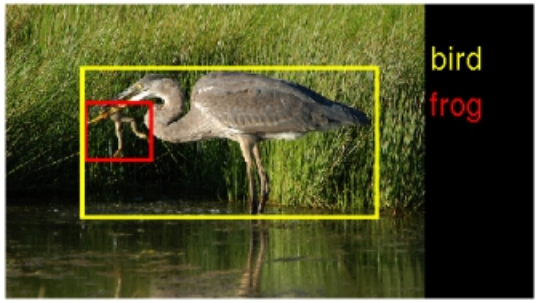
Image Classification Task:

1.2M training images • 1000 object categories

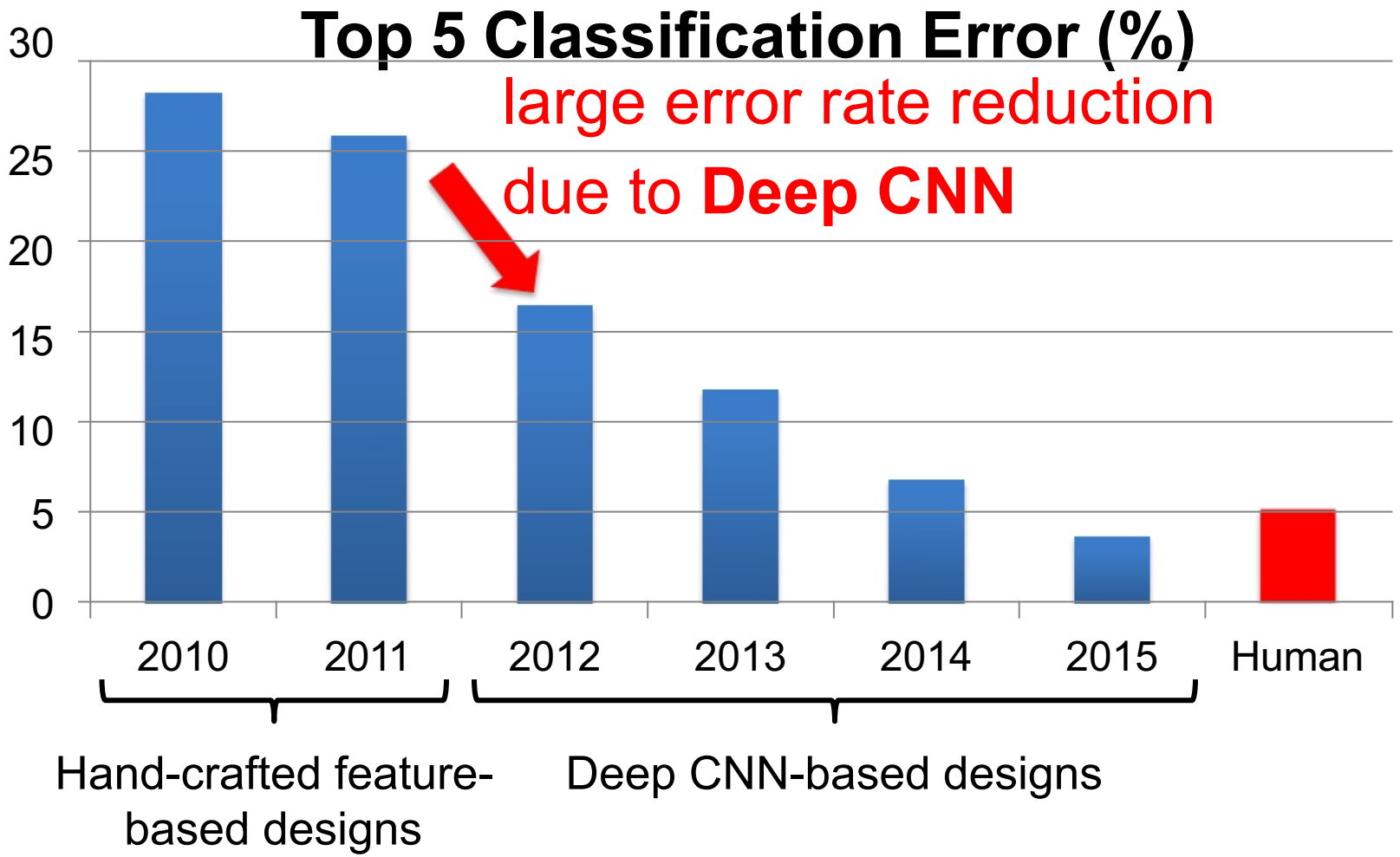
Object Detection Task:

456k training images • *200 object categories*





ImageNet: Image Classification Task



[Russakovsky et al., IJCV 2015]

Established Applications

- **Image**
  - Classification: image to object class
  - Recognition: same as classification (except for faces)
  - Detection: assigning bounding boxes to objects
  - Segmentation: assigning object class to every pixel

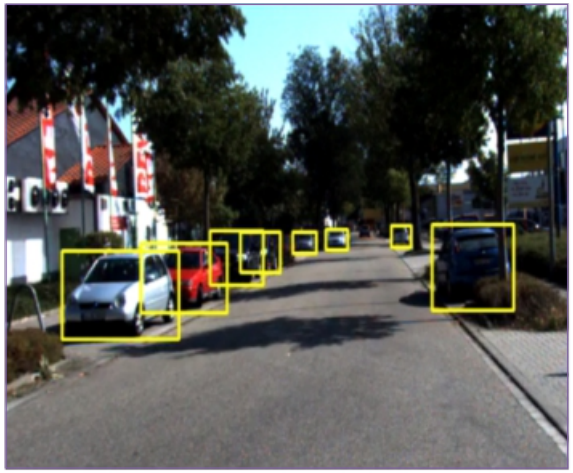
- **Speech & Language**
  - Speech Recognition: audio to text
  - Translation
  - Natural Language Processing: text to meani
  - Audio Generation: text to audio

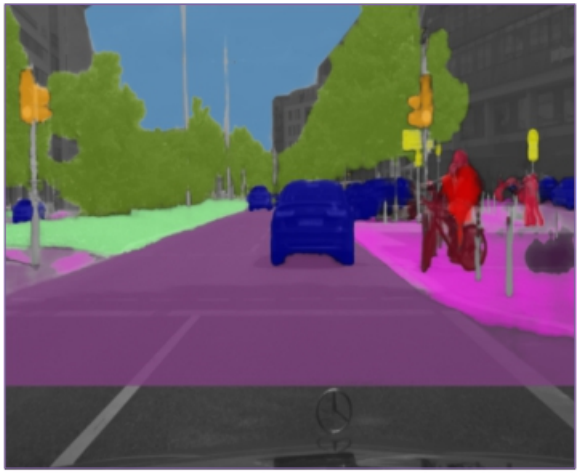
- **Games**

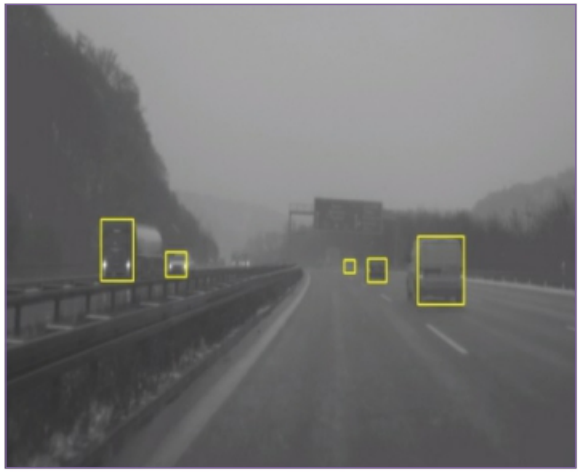
Deep Learning on Games

Google DeepMind AlphaGo

Deep Learning for Self-driving Cars





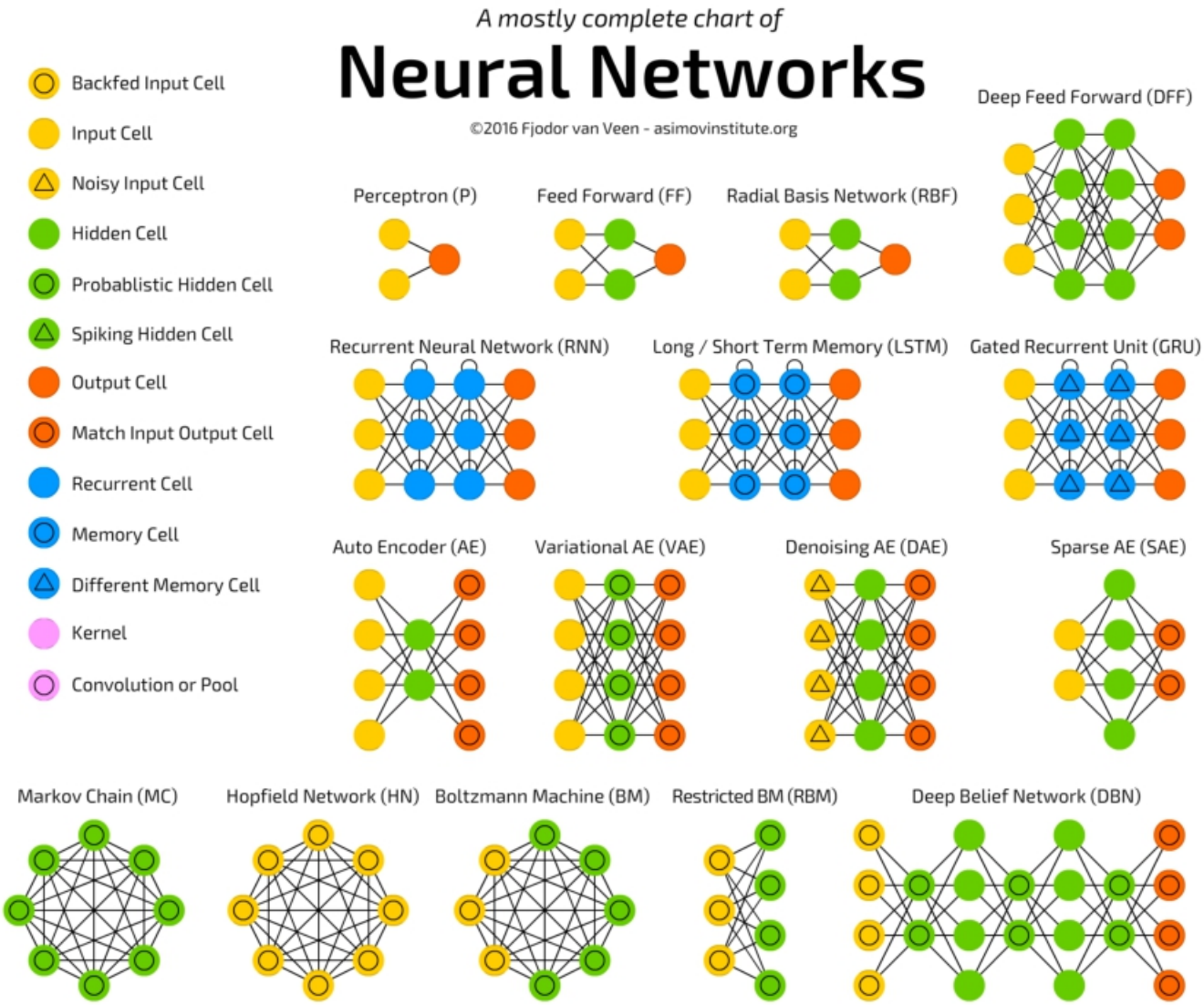


Overview of Deep Neural Networks

DNN Timeline

- **1940s: Neural networks were proposed**
- **1960s: Deep neural networks were proposed**
- **1989: Neural network for recognizing digits (LeNet)**
- **1990s: Hardware for shallow neural nets**
  - Example: Intel ETANN (1992)
- **2011: Breakthrough DNN-based speech recognition**
  - Microsoft real-time speech translation
- **2012: DNNs for vision supplanting traditional ML**
  - AlexNet for image classification
- **2014+: Rise of DNN accelerator research**
  - Examples: Neuflow, DianNao, etc.

So Many Neural Networks!



http://www.asimovinstitute.org/neural-network-zoo/

DNN Terminology 101

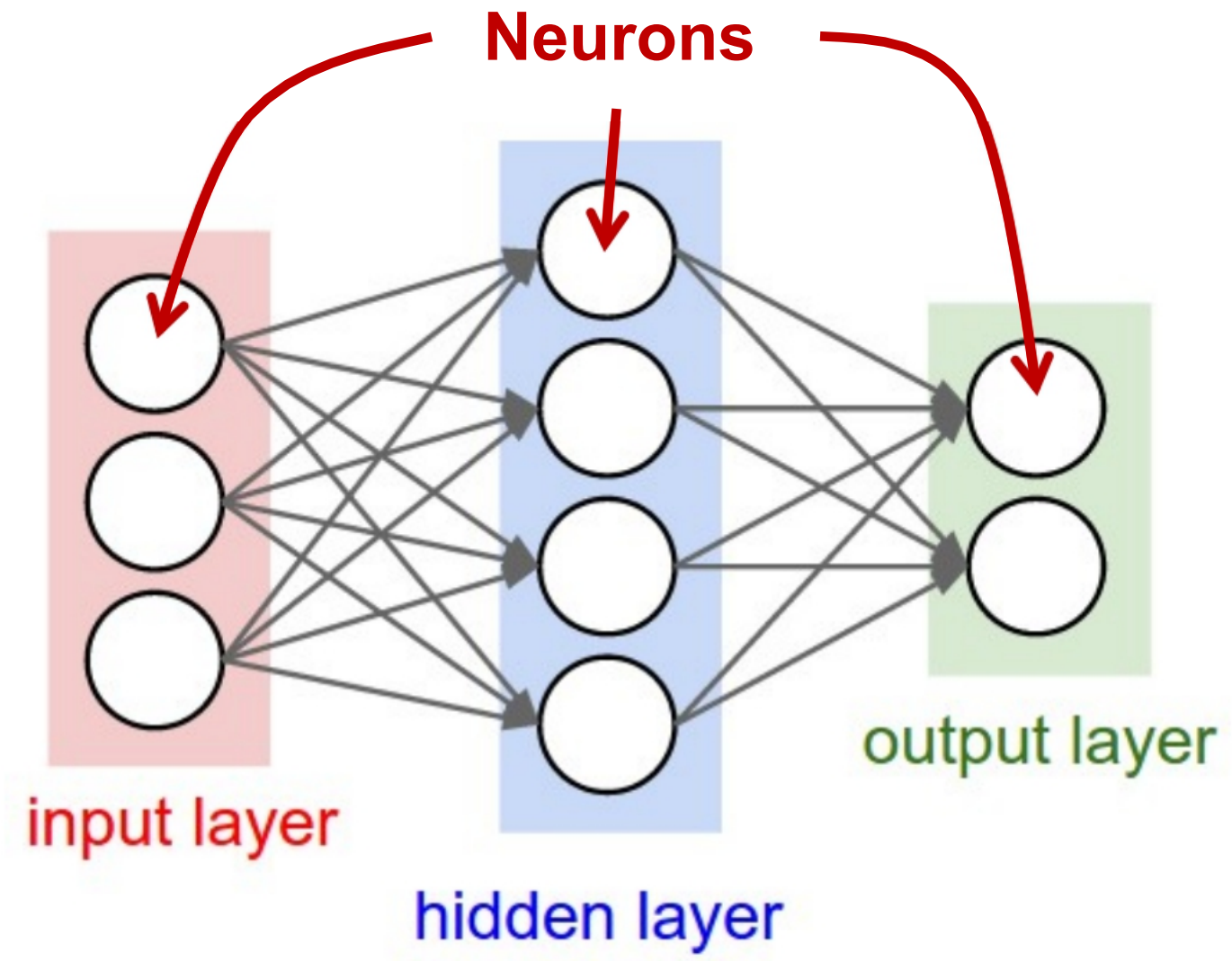


Image Source: Stanford

DNN Terminology 101

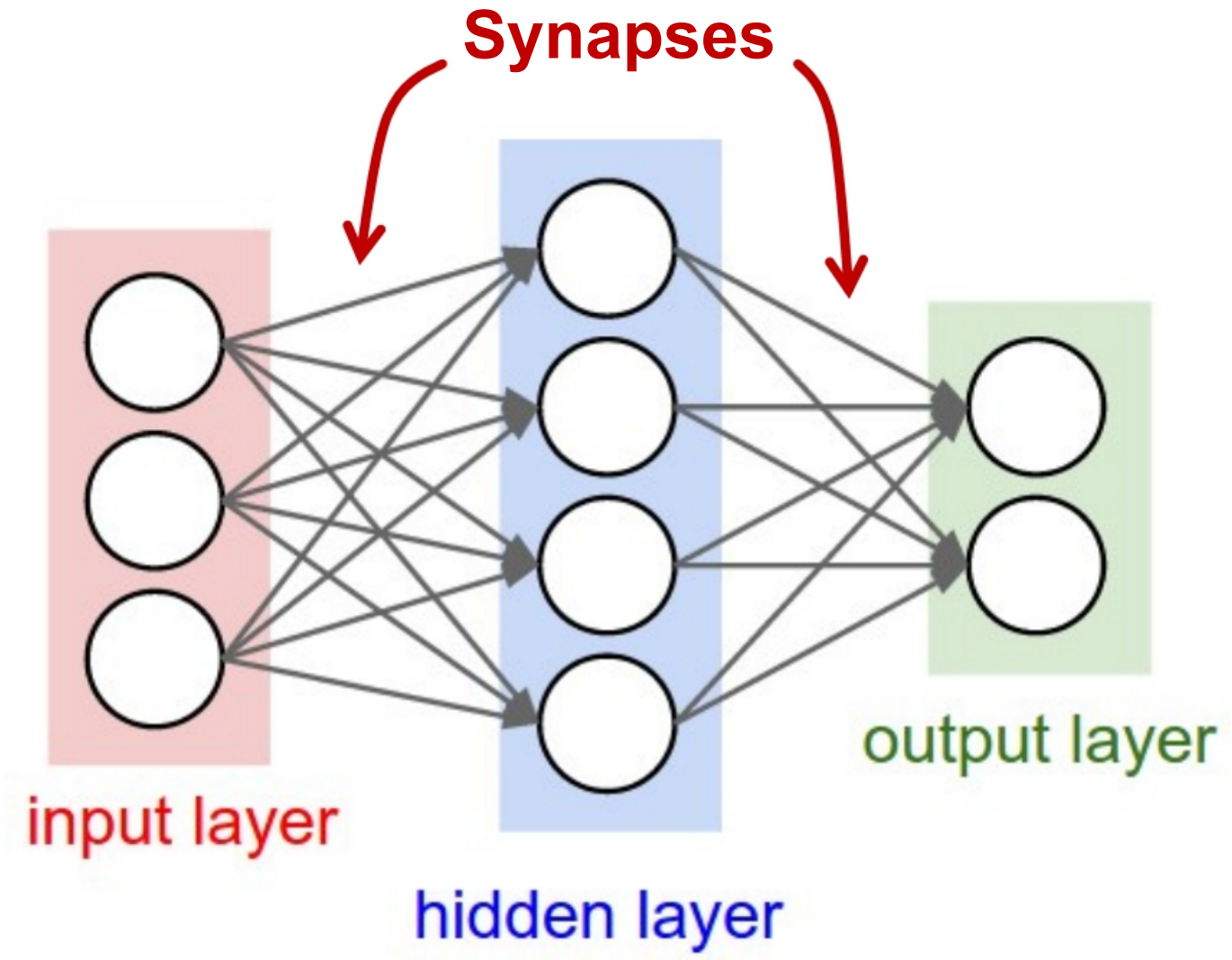


Image Source: Stanford

DNN Terminology 101

Each synapse has a **weight** for neuron **activation**

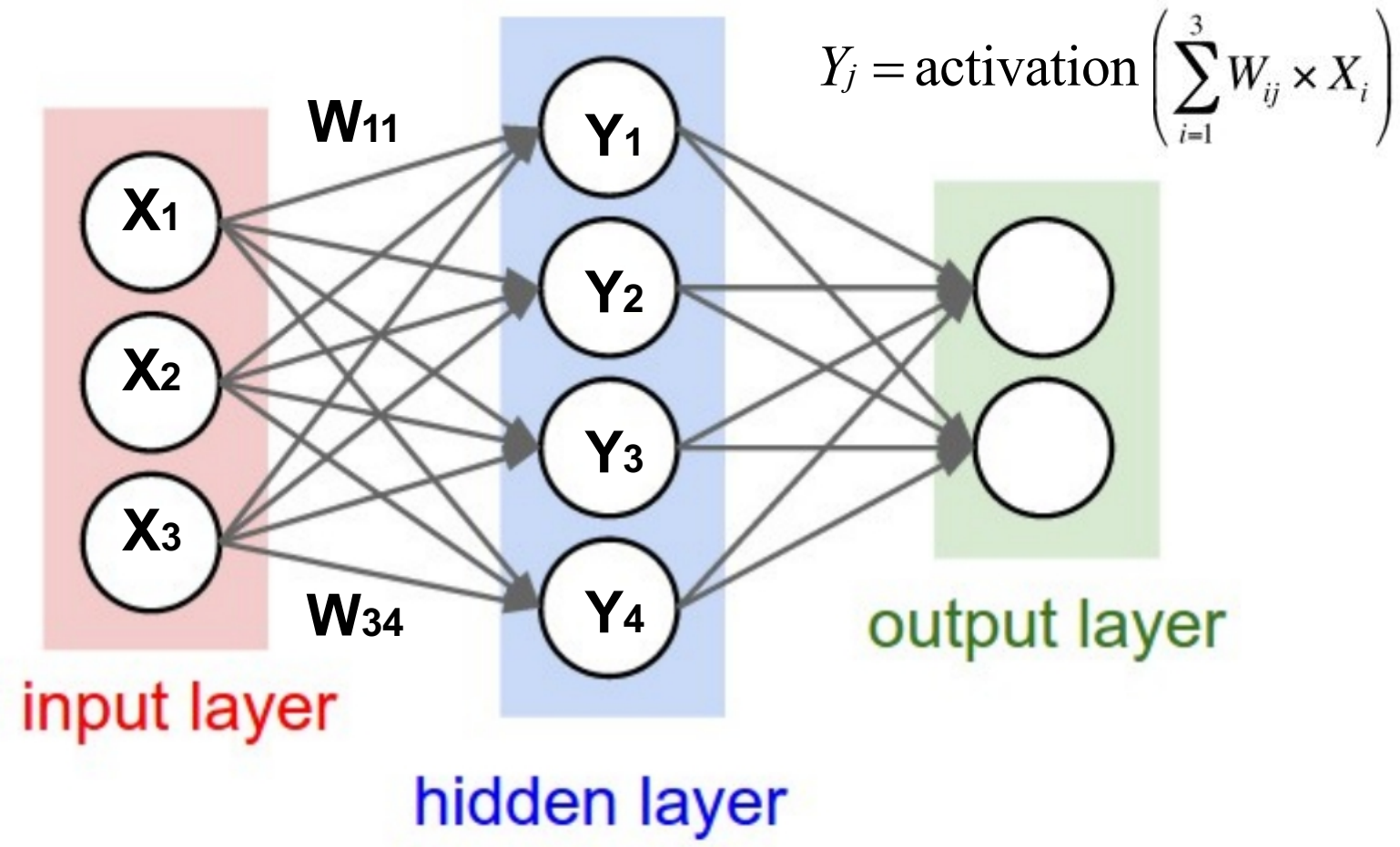


Image Source: Stanford

DNN Terminology 101

Weight Sharing: multiple synapses use the **same weight value**

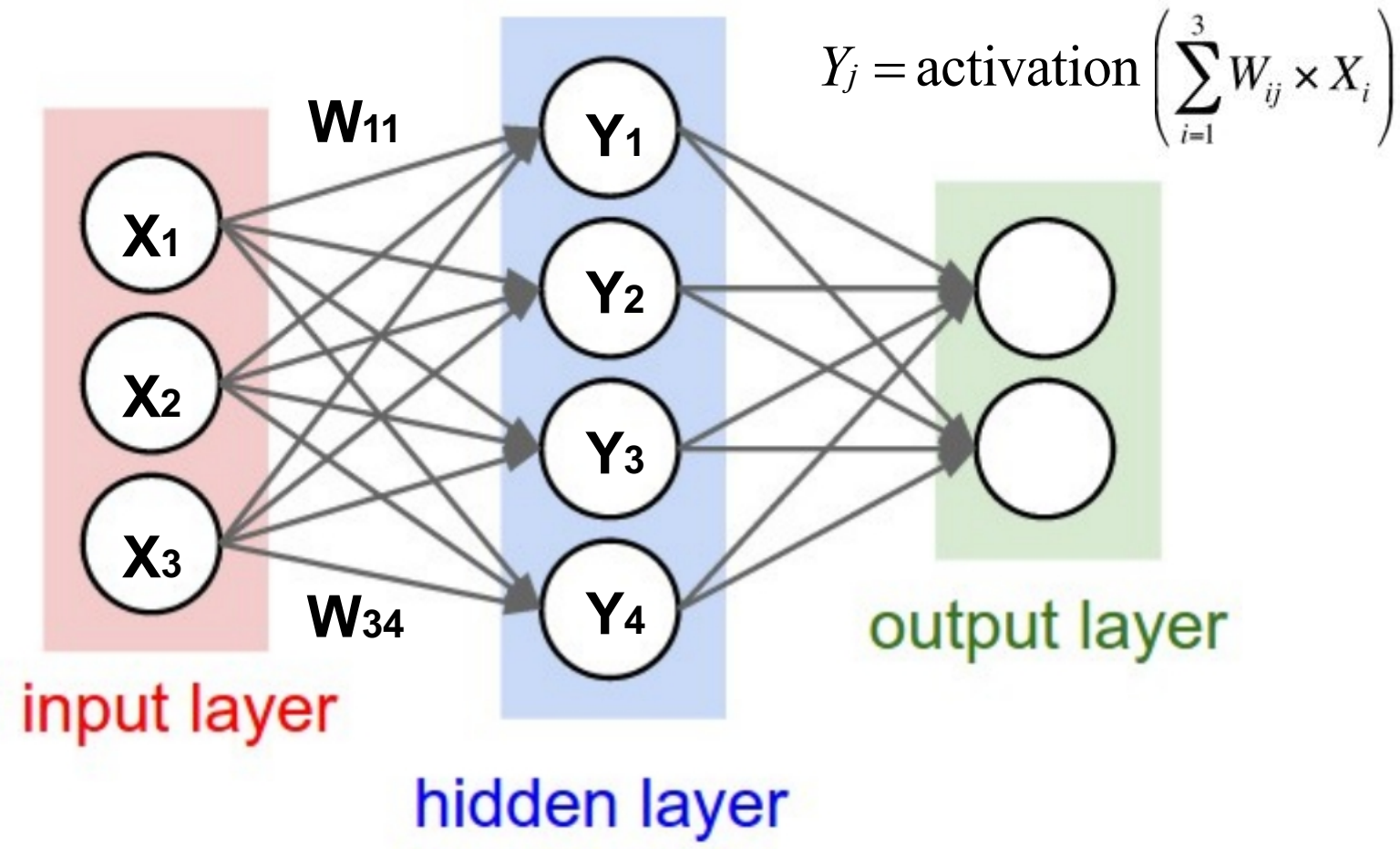


Image Source: Stanford

DNN Terminology 101

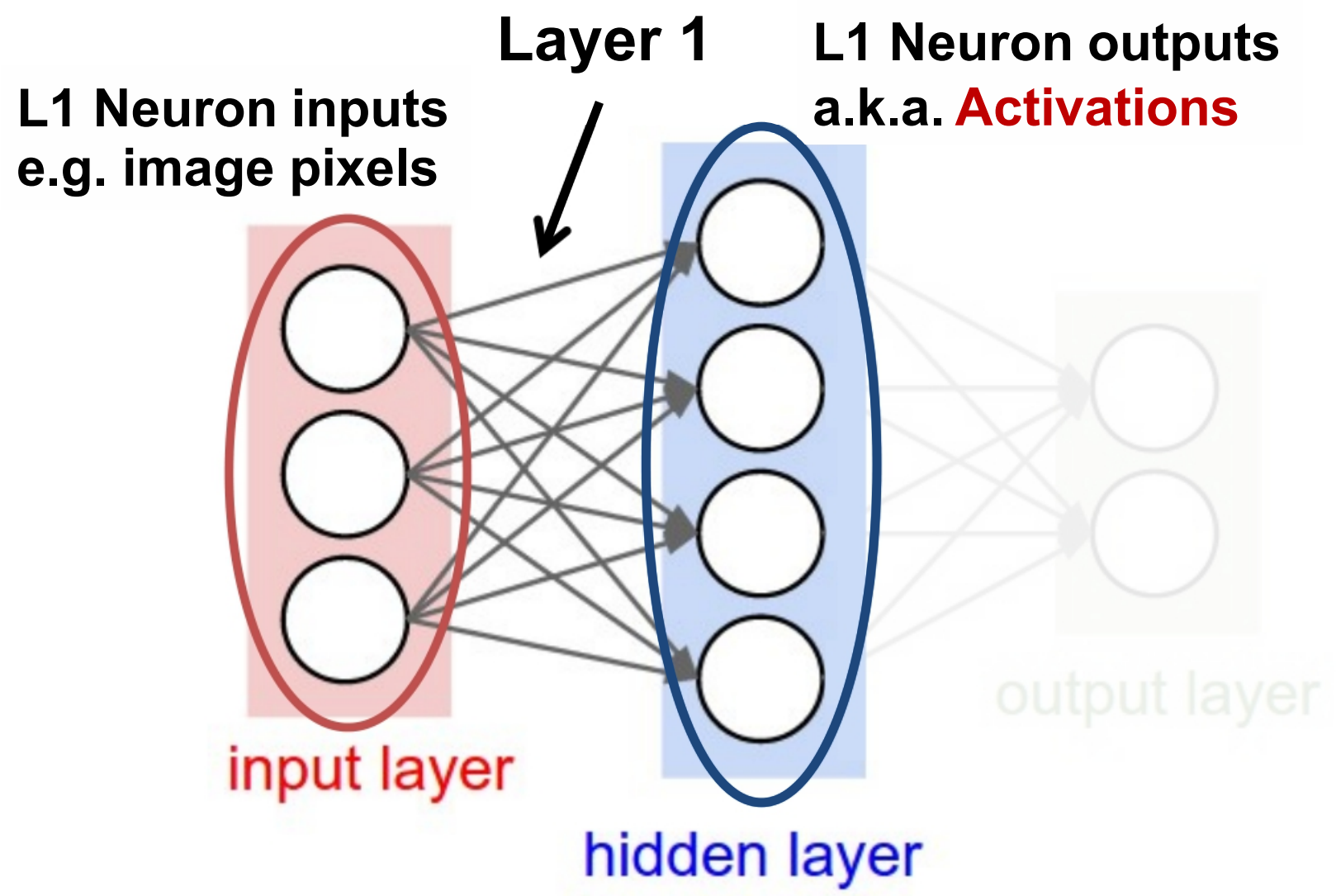


Image Source: Stanford

DNN Terminology 101

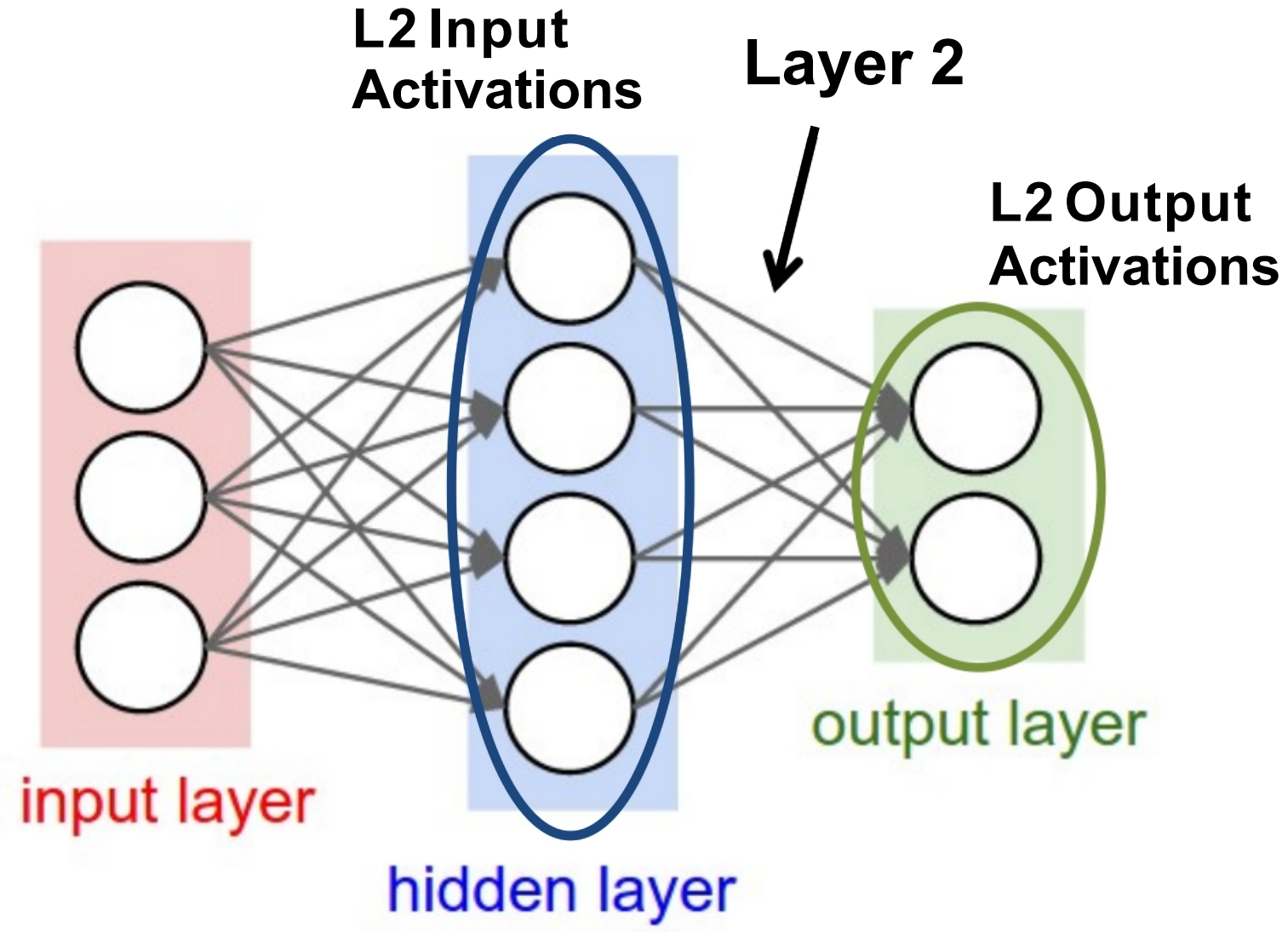


Image Source: Stanford

DNN Terminology 101

Fully-Connected: all i/p neurons connected to all o/p neurons

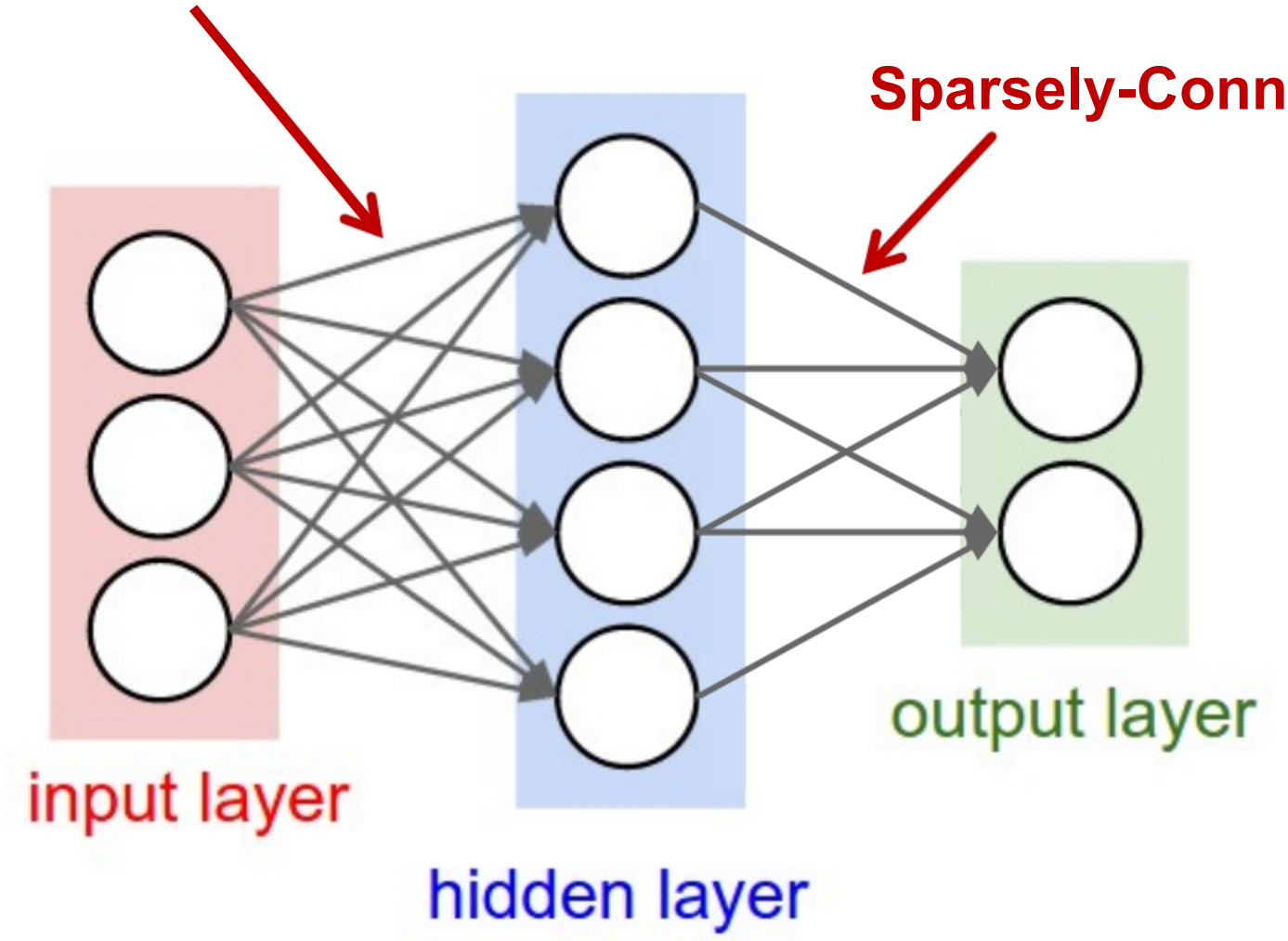


Image Source: Stanford

DNN Terminology 101

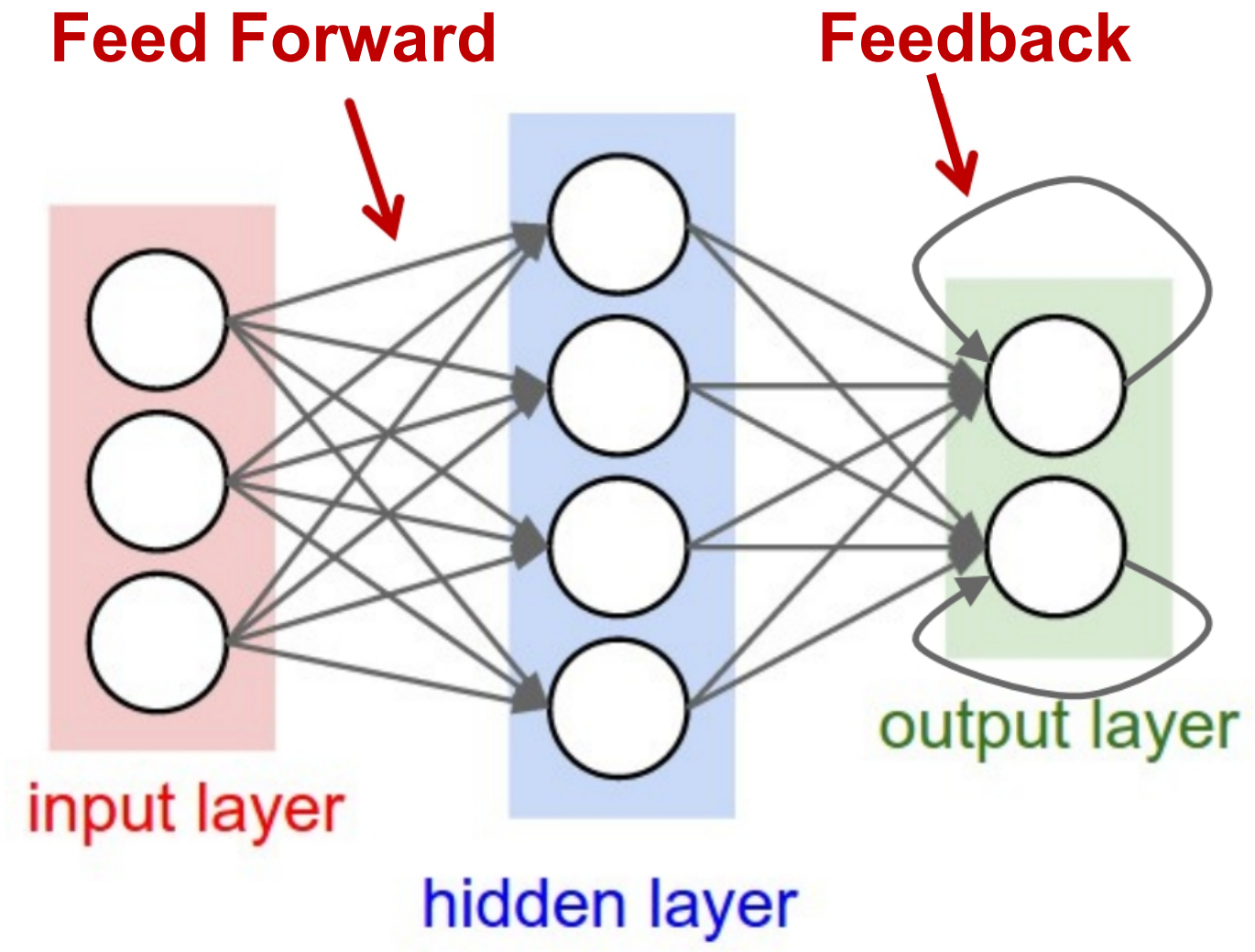


Image Source: Stanford

Popular Types of DNNs

- **Fully-Connected NN**
  - feed forward, a.k.a. multilayer perceptron (MLP)
- **Convolutional NN (CNN)**
  - feed forward, sparsely-connected w/ weight sharing
- **Recurrent NN (RNN)**
  - feedback
- **Long Short-Term Memory (LSTM)**
  - feedback + storage

Inference vs. Training

- **Training:** Determine weights
  - **Supervised:**
    - Training set has inputs and outputs, i.e., labeled
  - **Unsupervised:**
    - Training set is unlabeled
  - **Semi-supervised:**
    - Training set is partially labeled
  - **Reinforcement:**
    - Output assessed via rewards and punishments

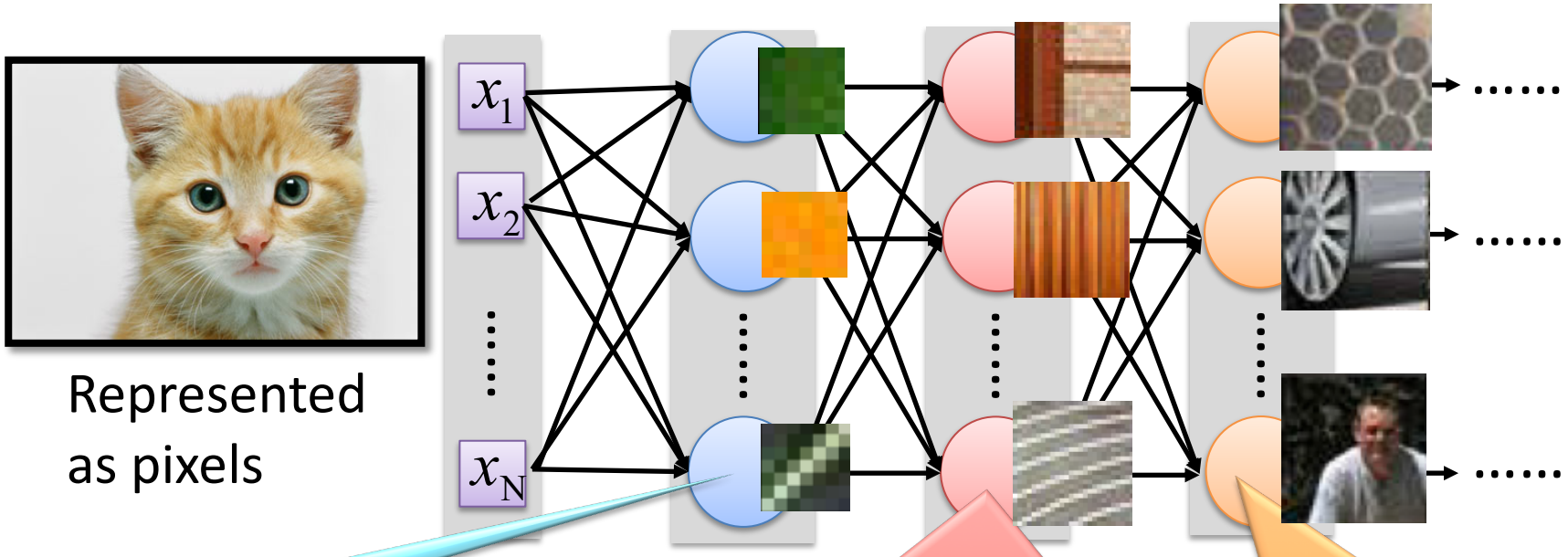
- **Inference:** Apply weights to determine output

Convolutional Neural Network

Slides are modified from Prof. Hung-yi Lee's CNN tutorial
Available at
http://speech.ee.ntu.edu.tw/~tlkagk/

Why CNN for Image?

[Zeiler, M. D., *ECCV 2014*]



Represented as pixels

The most basic classifiers

Use 1st layer as module to build classifiers

Use 2nd layer as module .....

Can the network be simplified by considering the properties of images?

Why CNN for Image

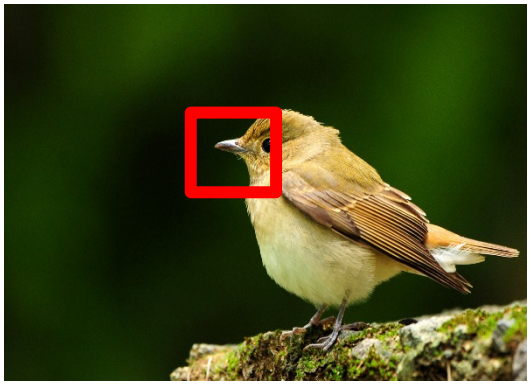
- Some patterns are much smaller than the whole image

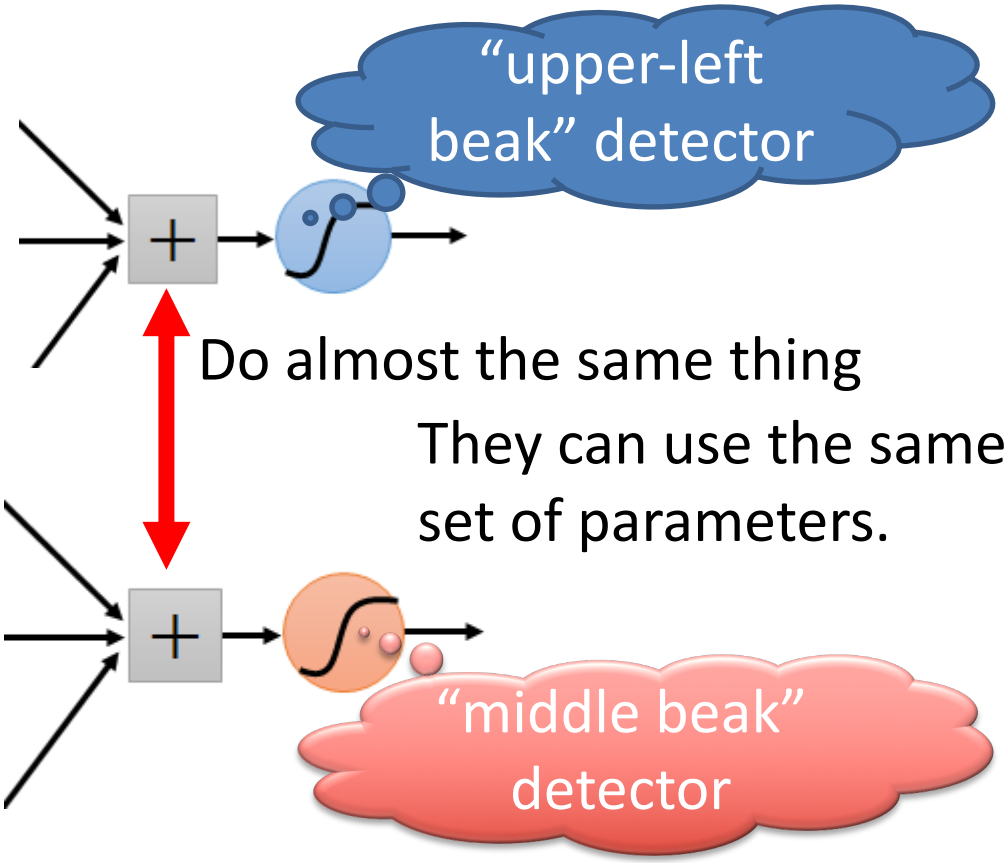
A neuron does not have to see the whole image to discover the pattern.

Connecting to small region with less parameters

Why CNN for Image

- The same patterns appear in different regions.





Why CNN for Image

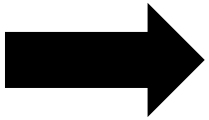
- Subsampling the pixels will not change the object

bird

subsampling

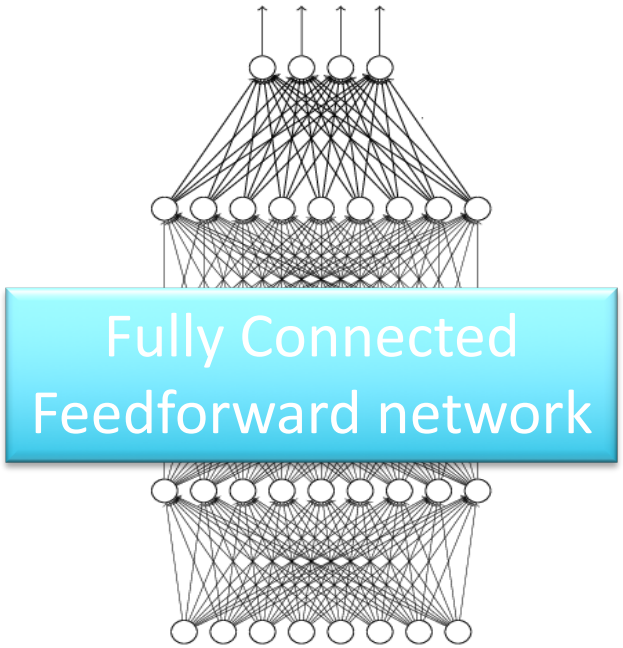
bird

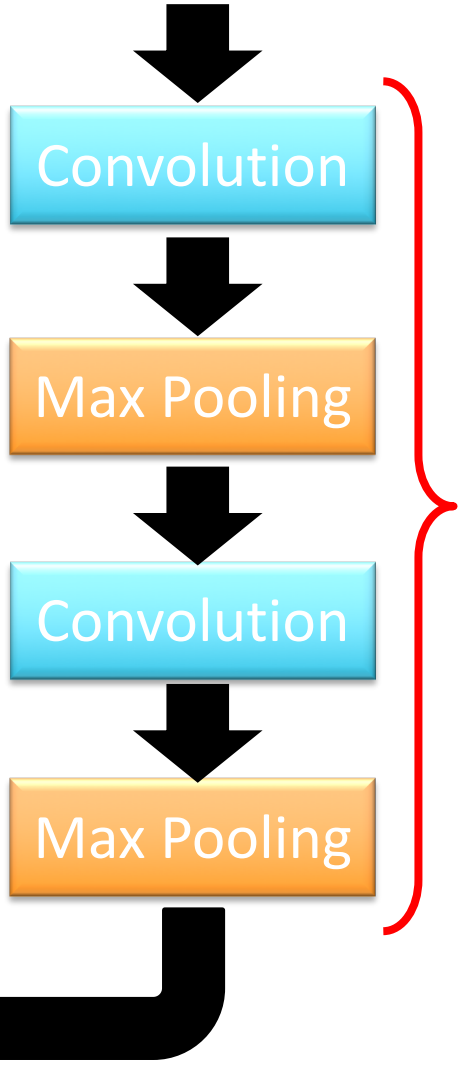
We can subsample the pixels to make image smaller



Less parameters for the network to process the image

The whole CNN





The whole CNN

Property 1

- Some patterns are much smaller than the whole image

Property 2

- The same patterns appear in different regions.

Property 3

- Subsampling the pixels will not change the object

Convolution

Max Pooling

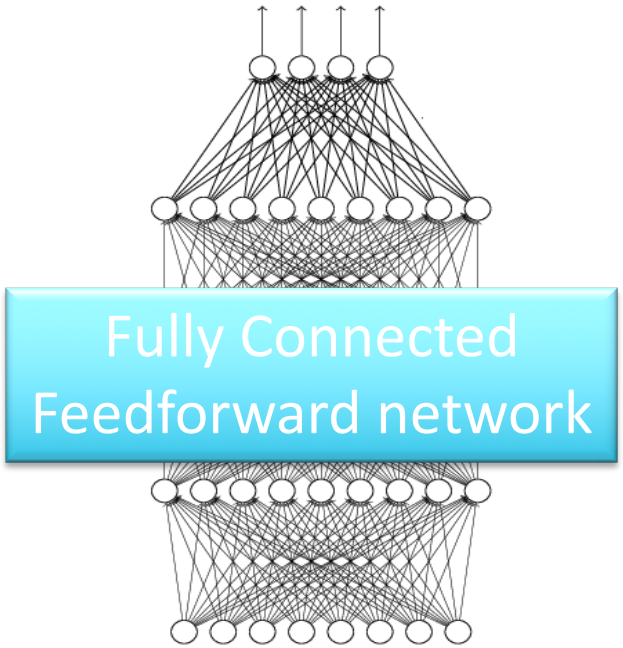
Convolution

Max Pooling

Can repeat many times

Flatten

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

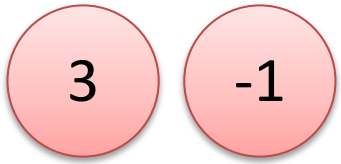
1	-1	-1
-1	1	-1
-1	-1	1

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

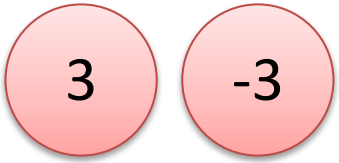
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

If stride=2

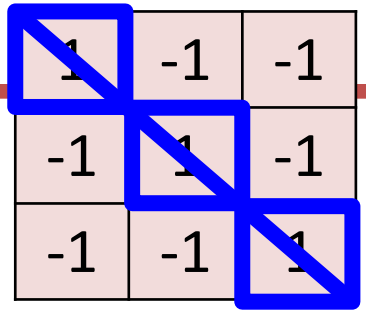
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image



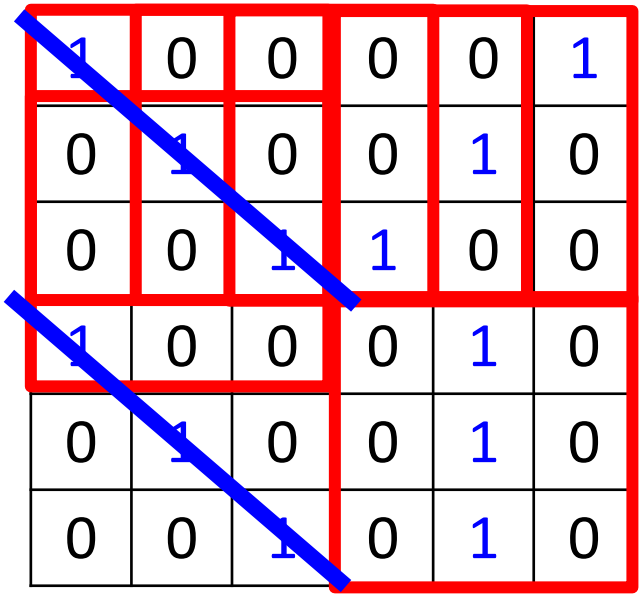
We set stride=1 below

CNN – Convolution

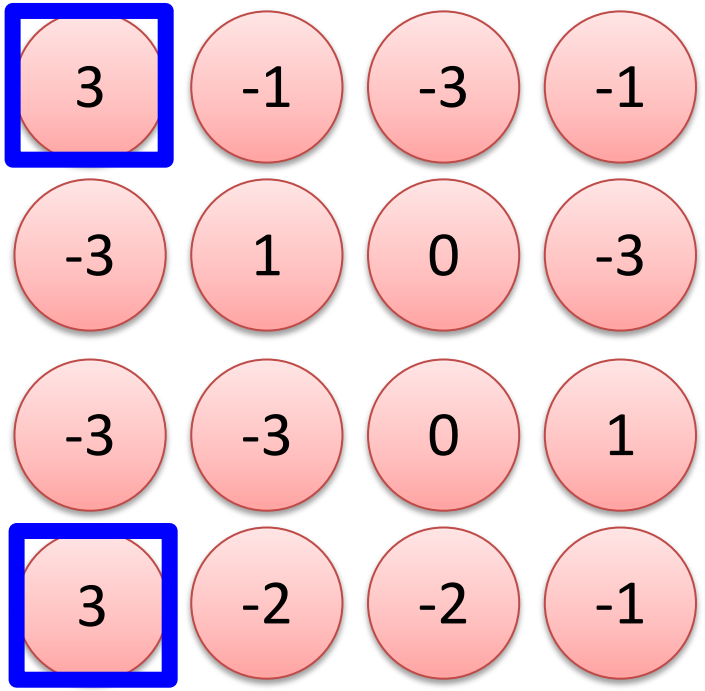


Filter 1

stride=1



6 x 6 image



Property 2

CNN – Convolution

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

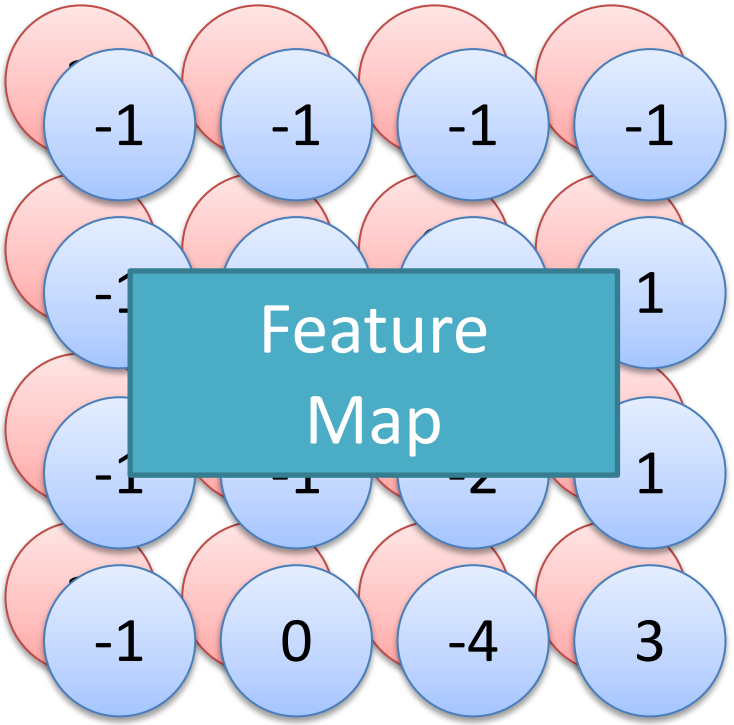
Filter 2

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

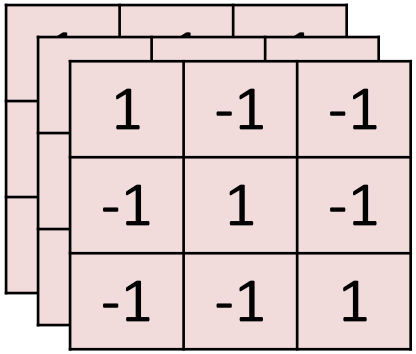
6 x 6 image

Do the same process for every filter

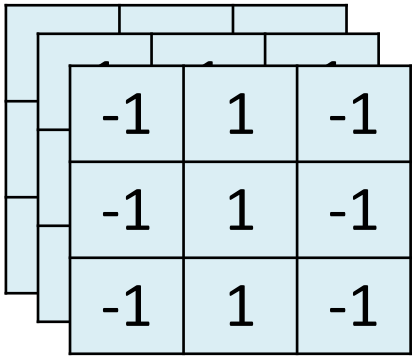


4 x 4 image

CNN – Colorful image

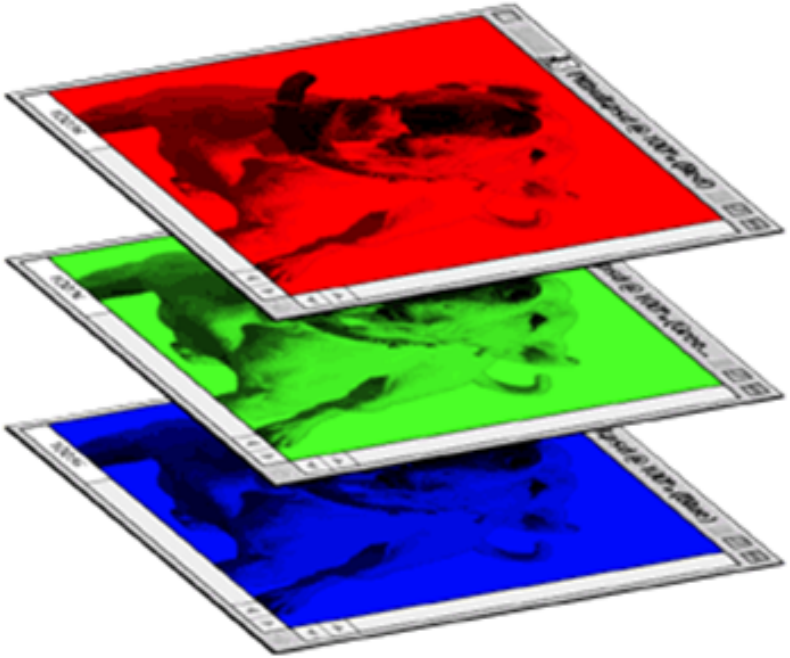


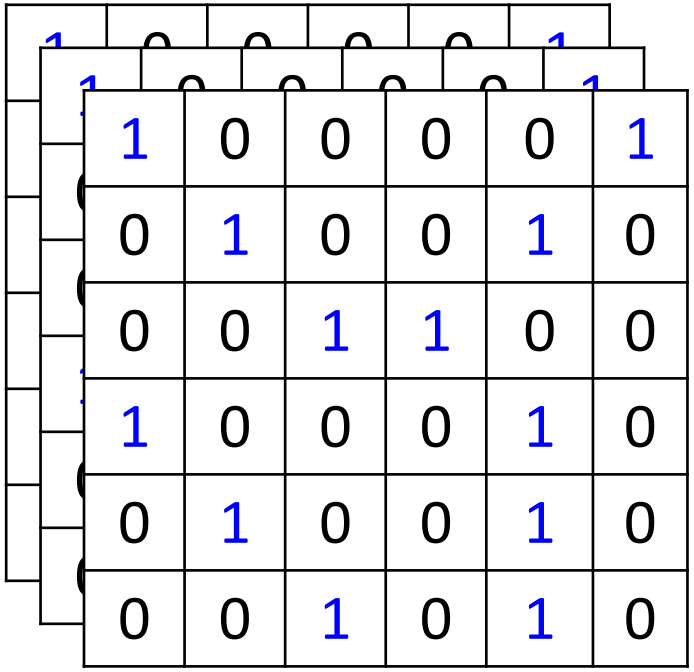
Filter 1



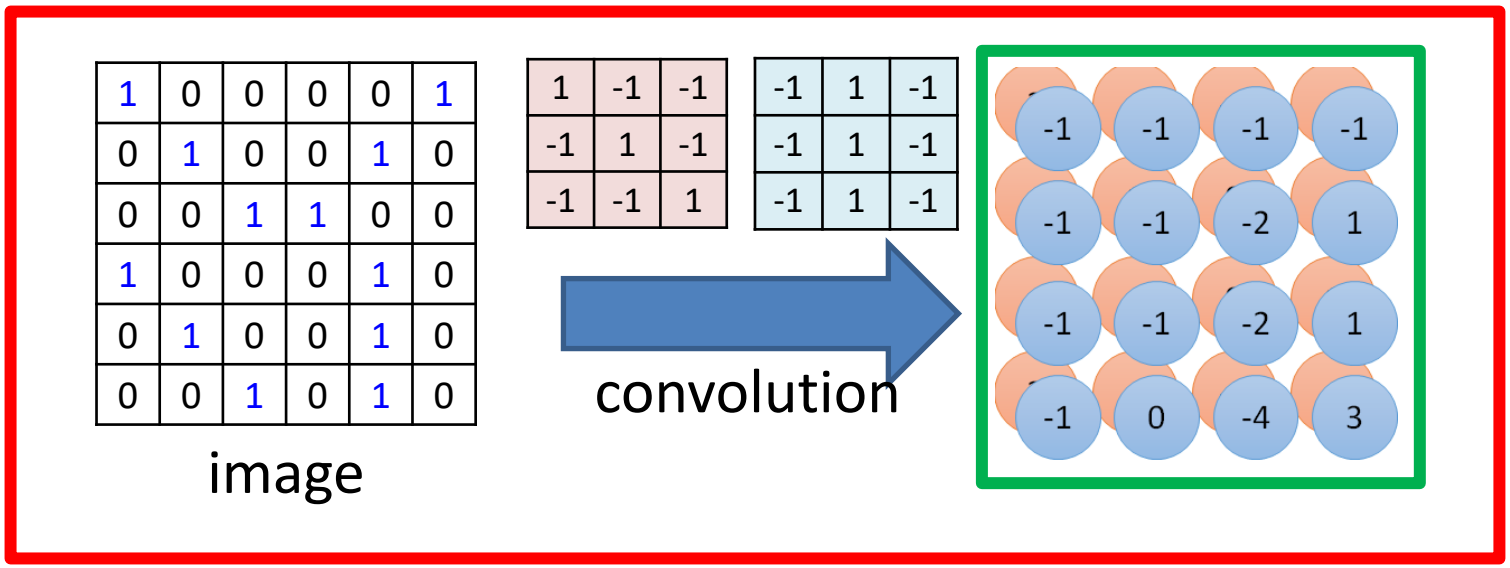
Filter 2

Colorful image



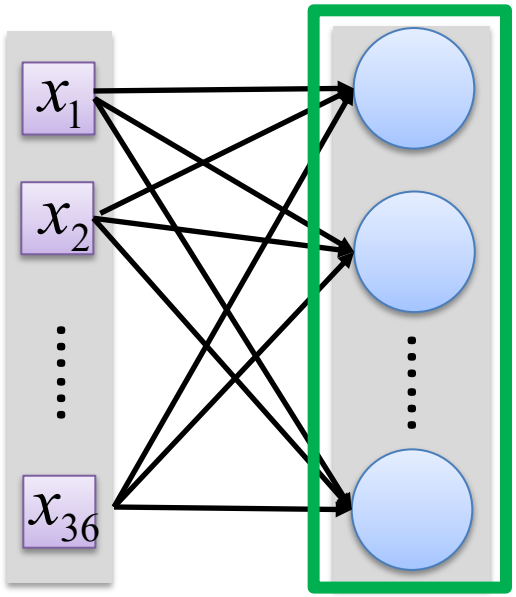


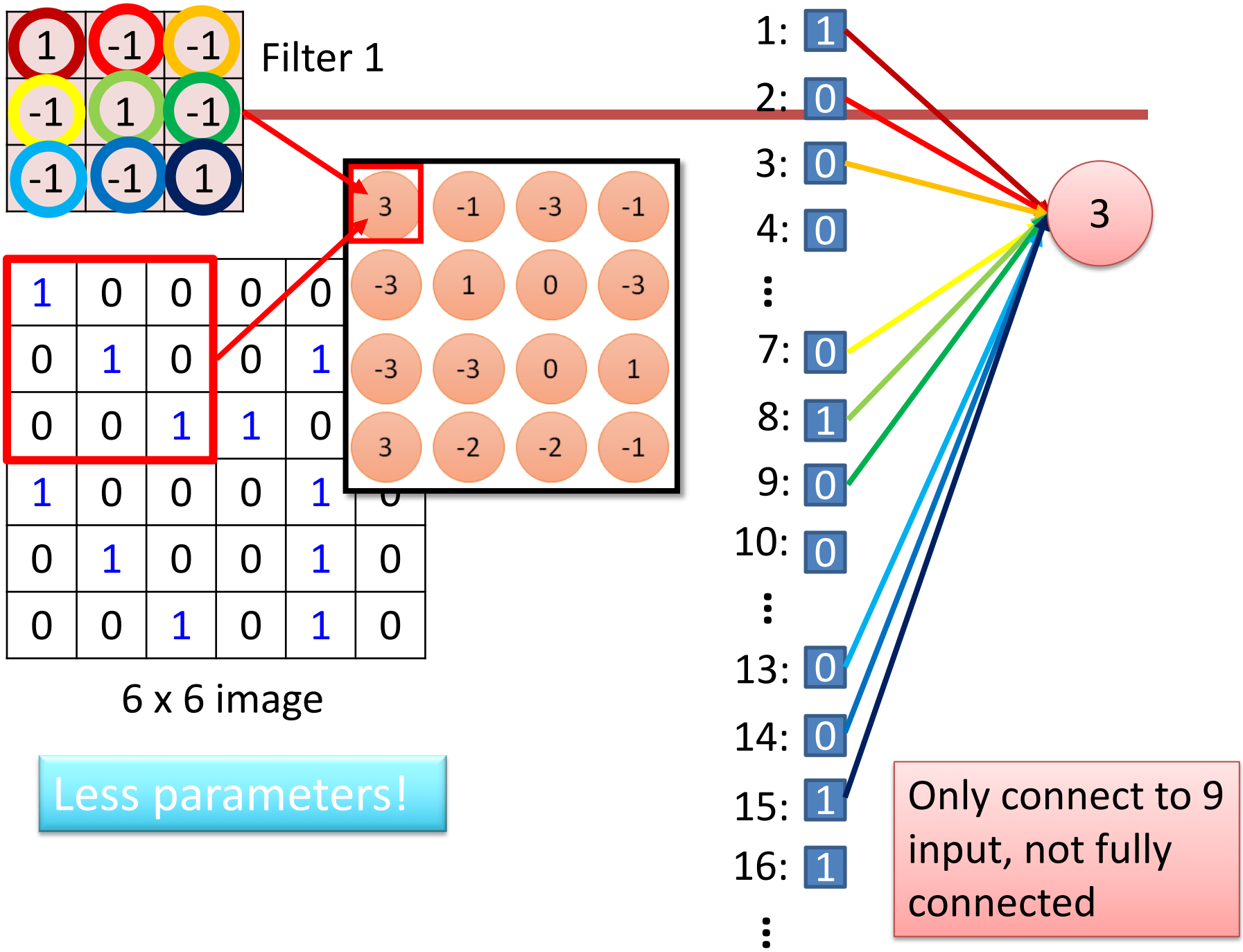
Convolution v.s. Fully Connected

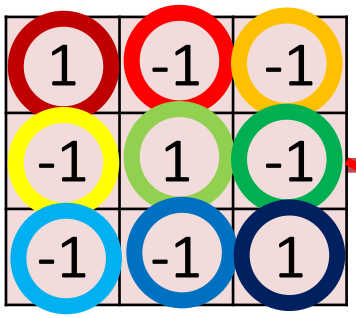


Fully-connected

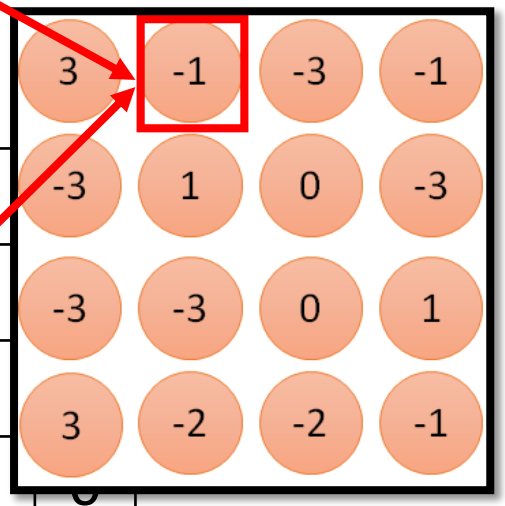
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0




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

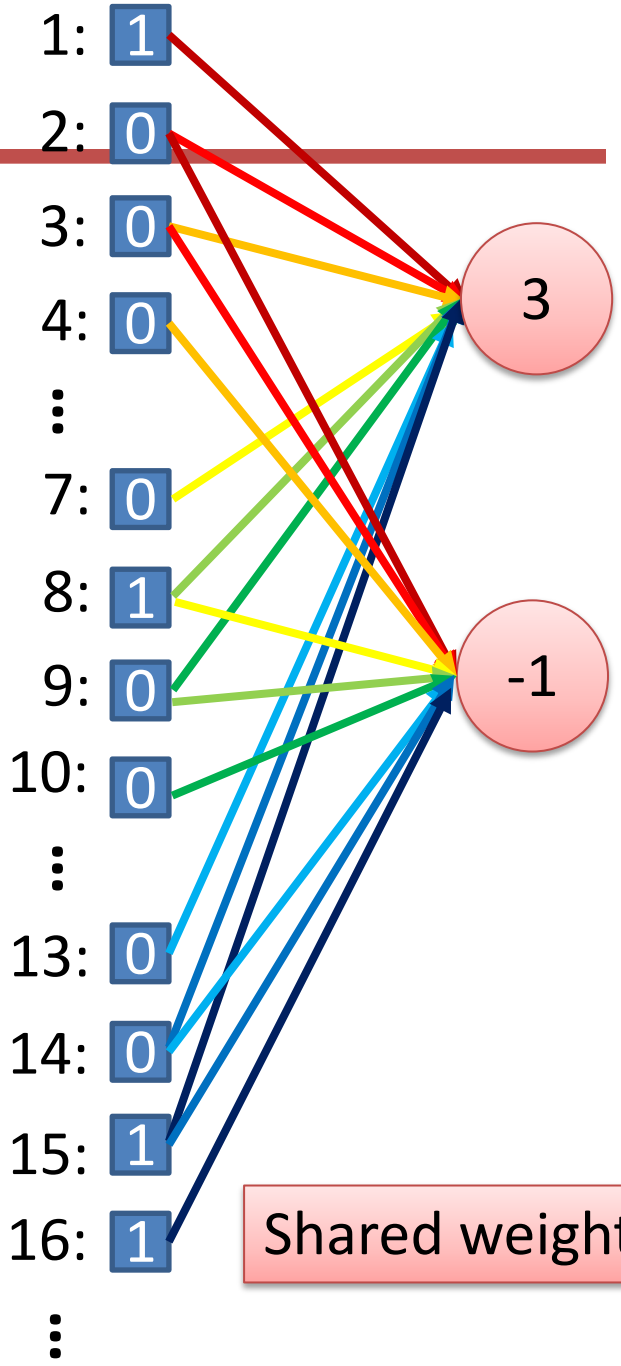


Filter 1



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

6 x 6 image



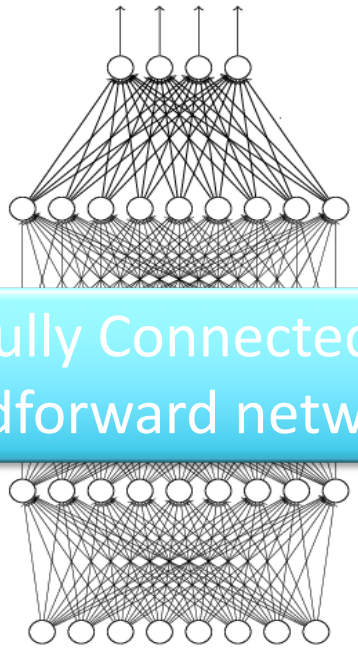
Less parameters!

Even less parameters!

Shared weights

The whole CNN

cat dog ……



Fully Connected Feedforward network

Convolution

Max Pooling

Convolution

Max Pooling

Can repeat many times

Flatten

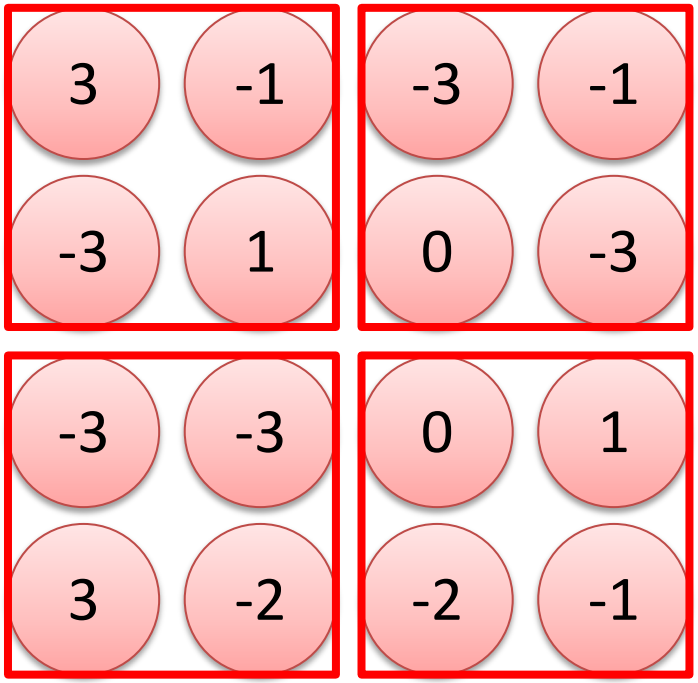
CNN – Max Pooling

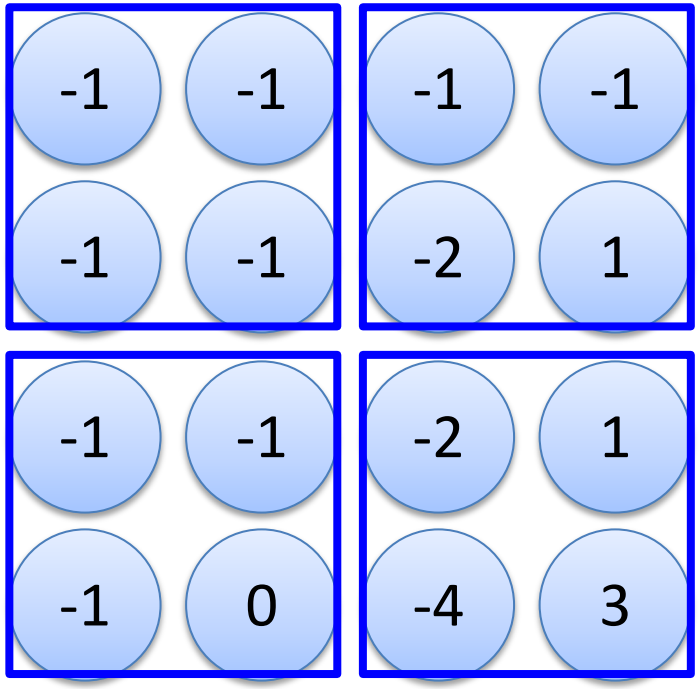
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

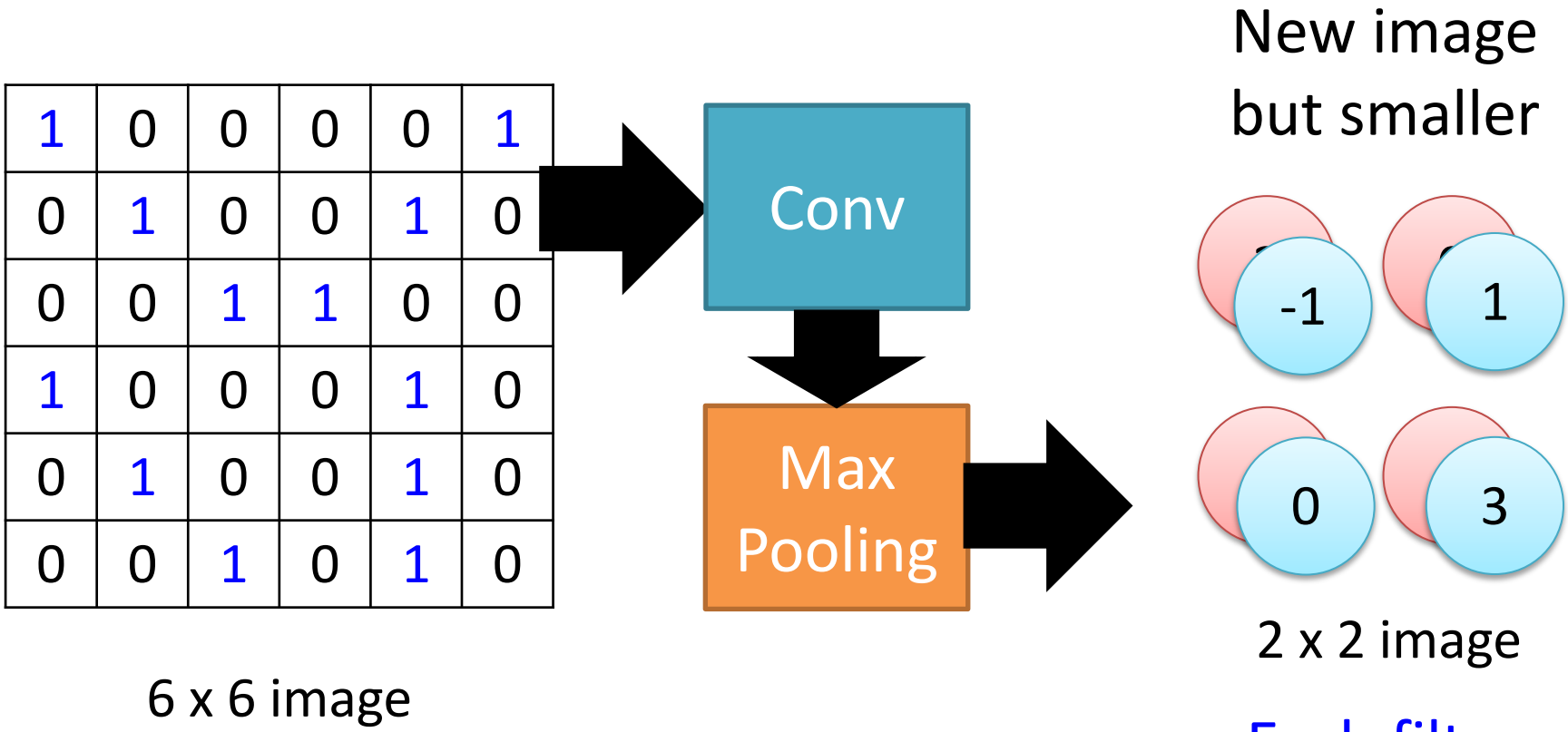
-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

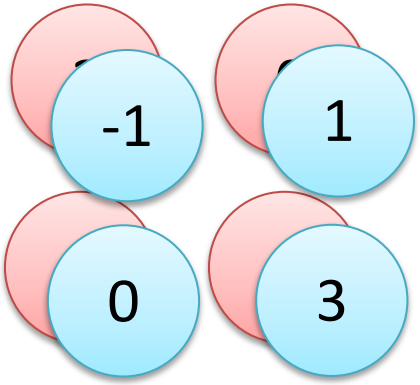




CNN – Max Pooling



The whole CNN



A new image

Convolution

Max Pooling

Convolution

Max Pooling

Can repeat many times

Smaller than the original image

The number of the channel is the number of filters

The whole CNN

cat dog ……

Convolution

Max Pooling

A new image

Convolution

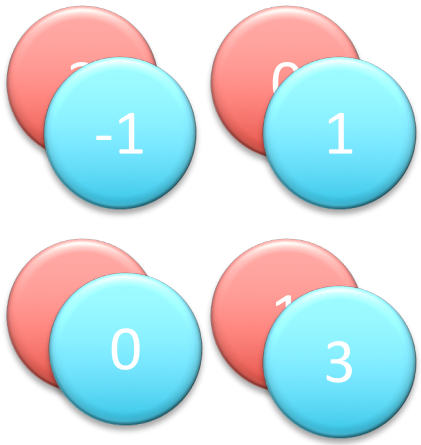
Max Pooling

A new image

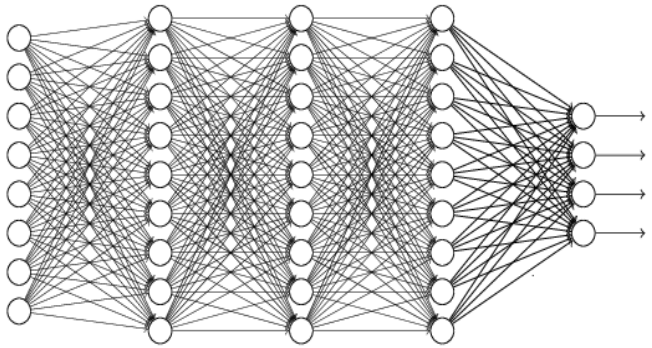
Flatten

Fully Connected Feedforward network

Flatten

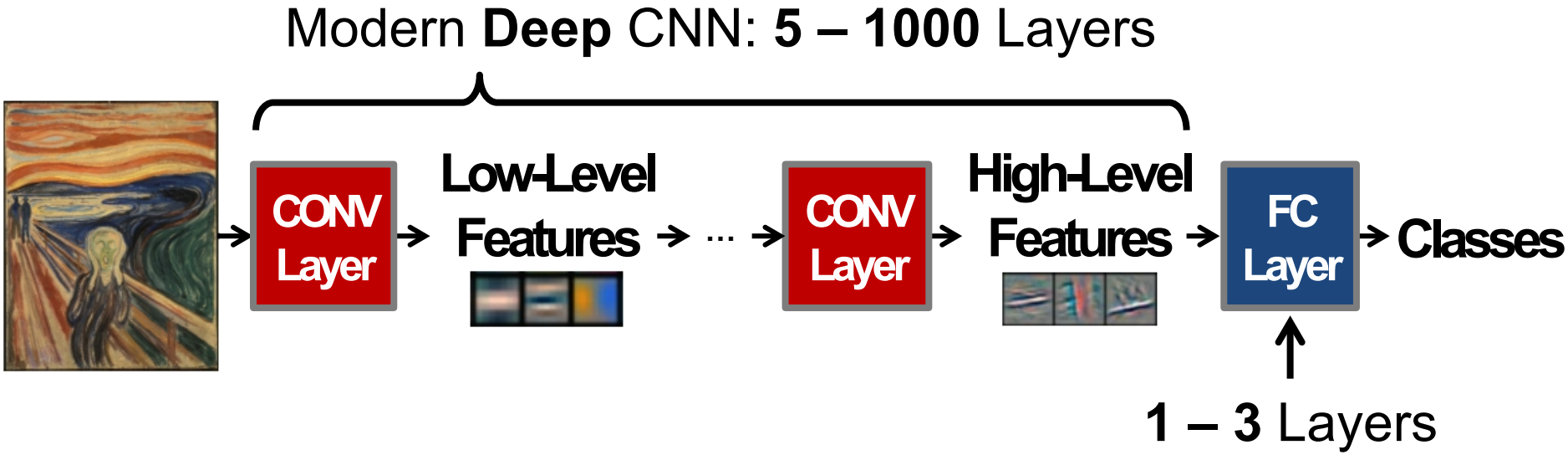


Flatten

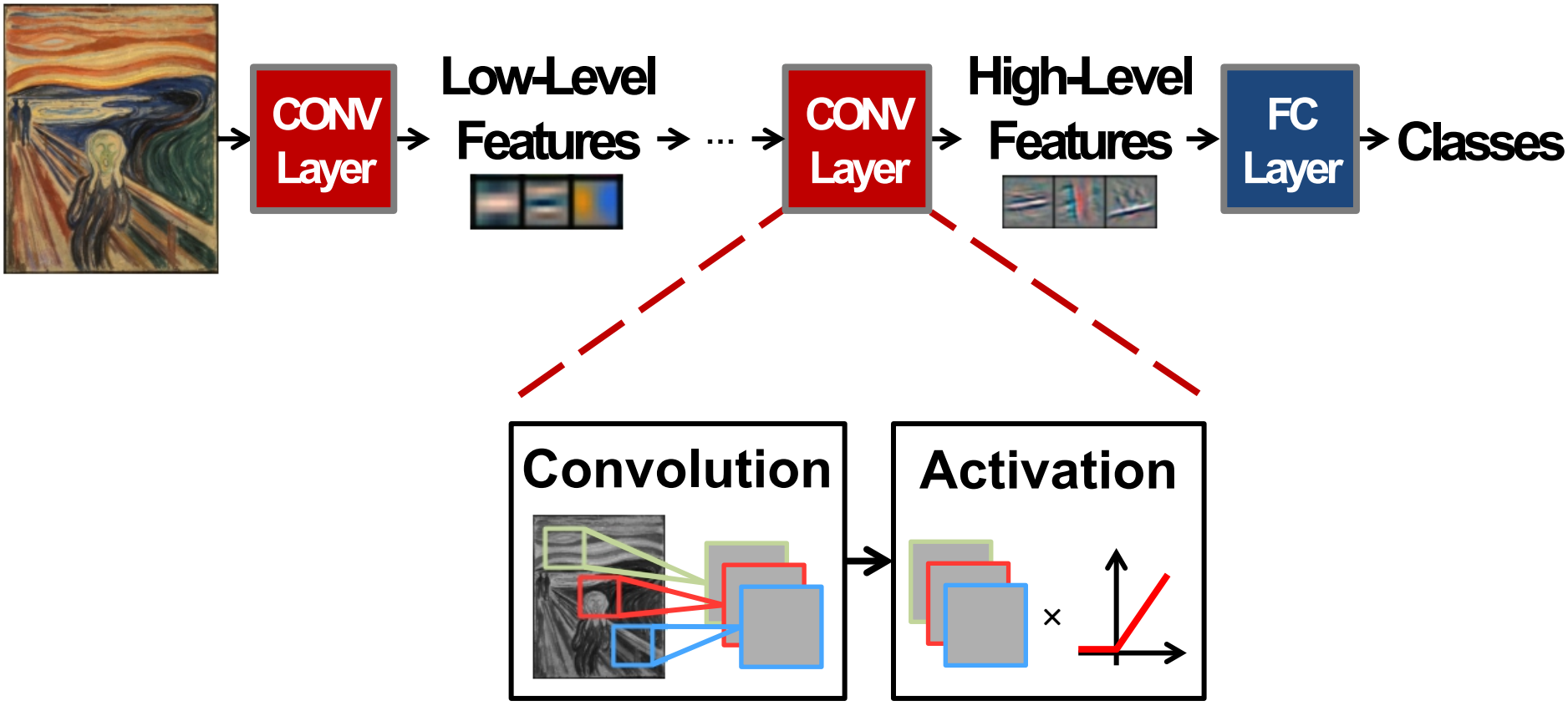


Fully Connected Feedforward network

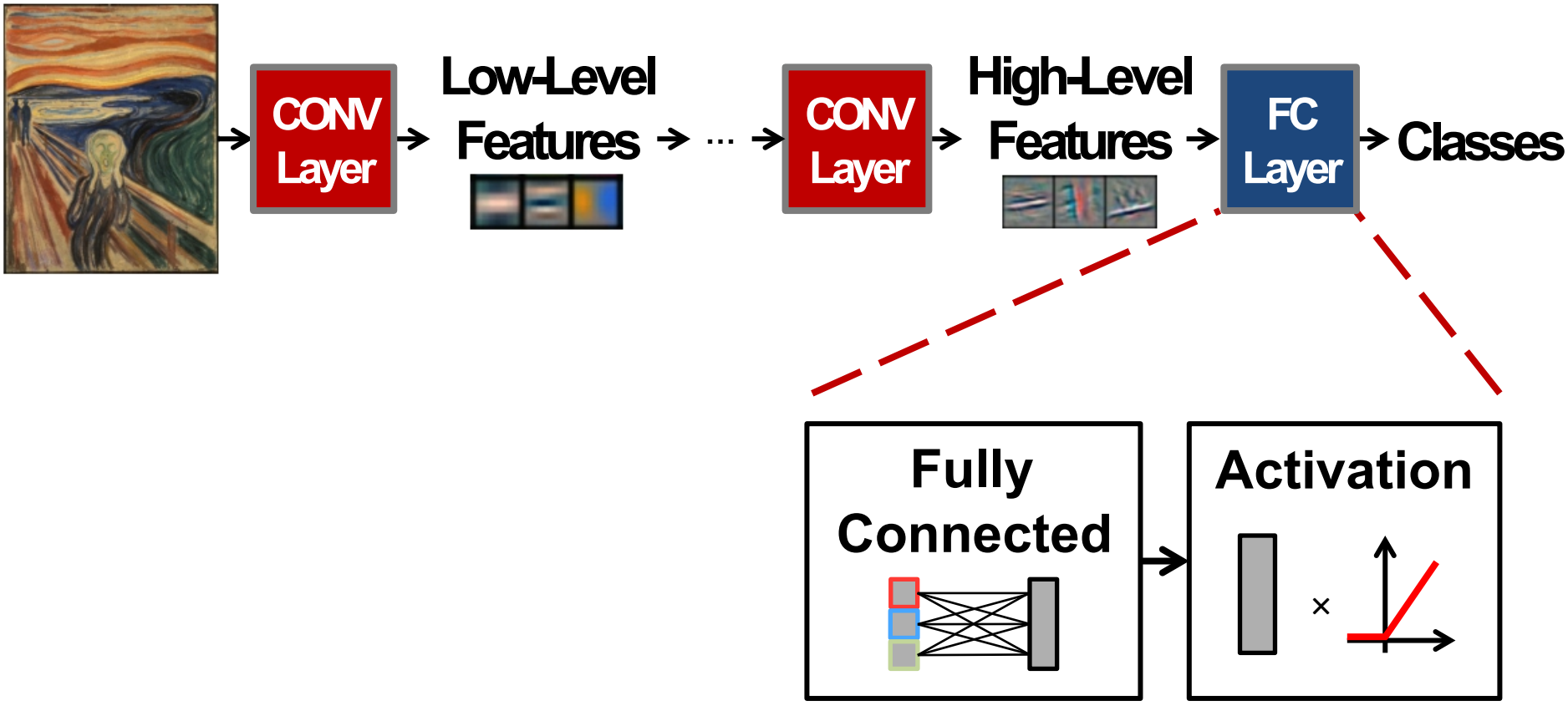
Deep Convolutional Neural Networks



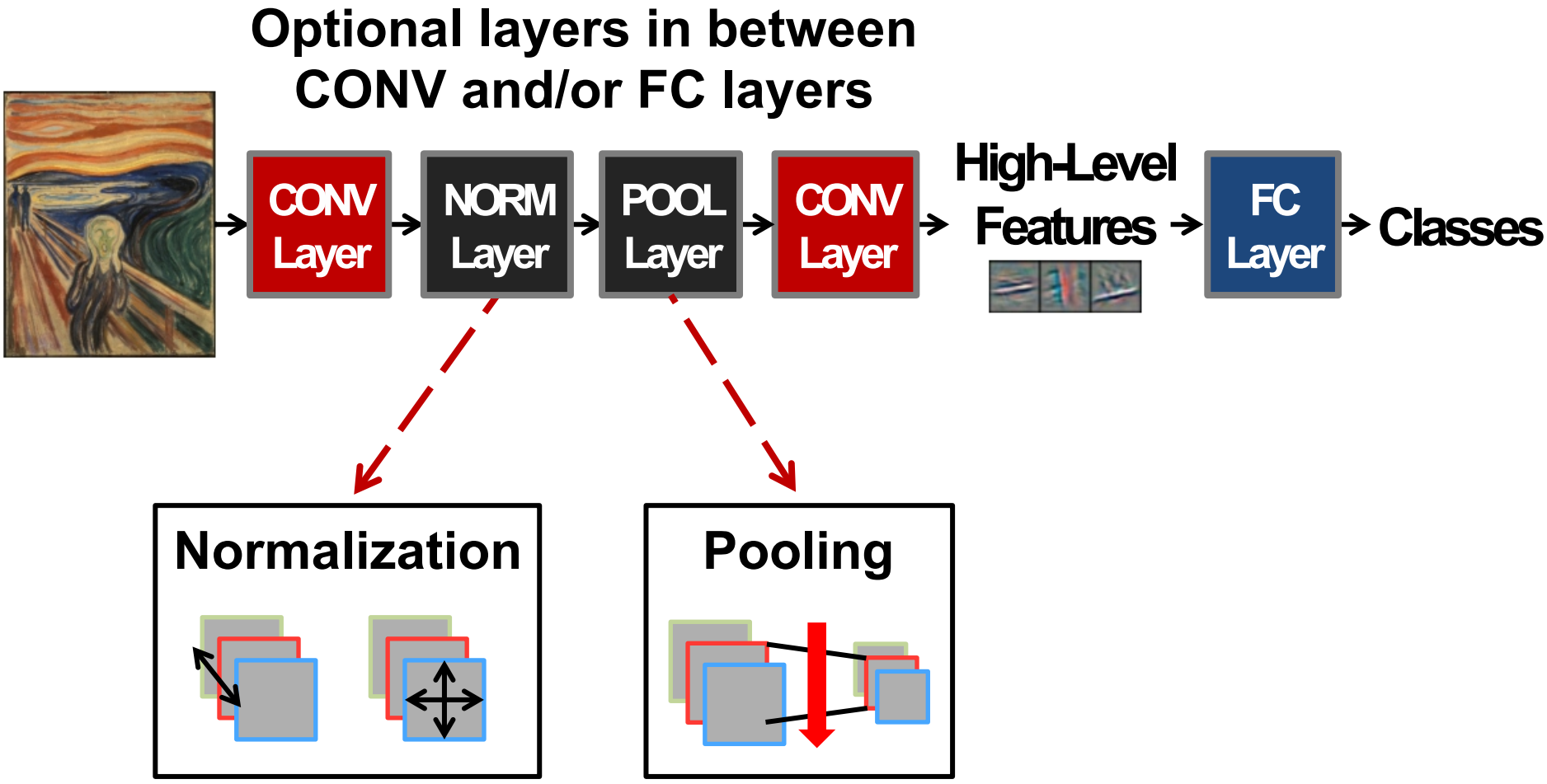
Deep Convolutional Neural Networks



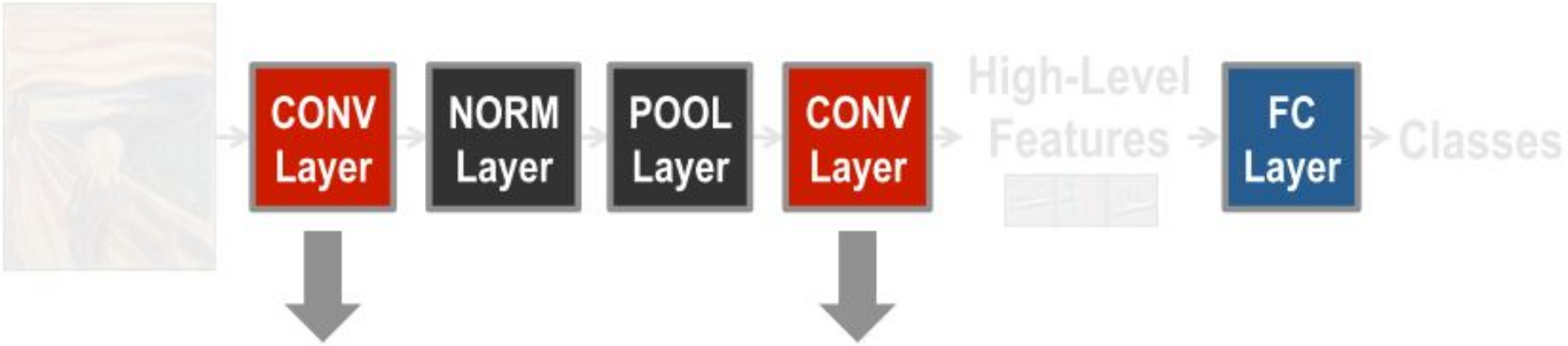
Deep Convolutional Neural Networks

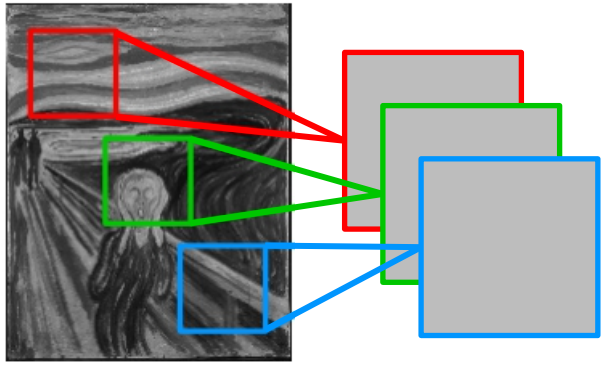


Deep Convolutional Neural Networks



Deep Convolutional Neural Networks





Convolutions account for more than 90% of overall computation, dominating **runtime** and **energy consumption**

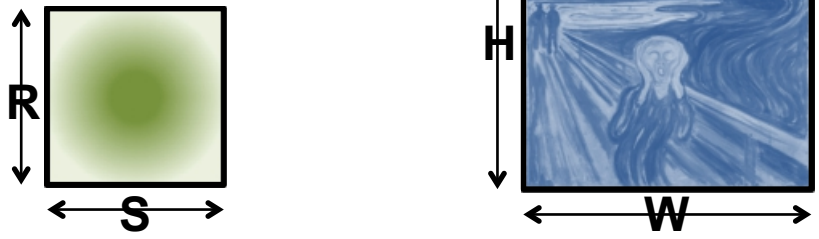
CNN Decoder Ring

- N – Number of input fmaps/output fmaps (batch size)
- C – Number of 2-D input fmaps /filters (channels)
- H – Height of input fmap (activations)
- W – Width of input fmap (activations)
- R – Height of 2-D filter (weights)
- S – Width of 2-D filter (weights)
- M – Number of 2-D output fmaps (channels)
- E – Height of output fmap (activations)
- F – Width of output fmap (activations)

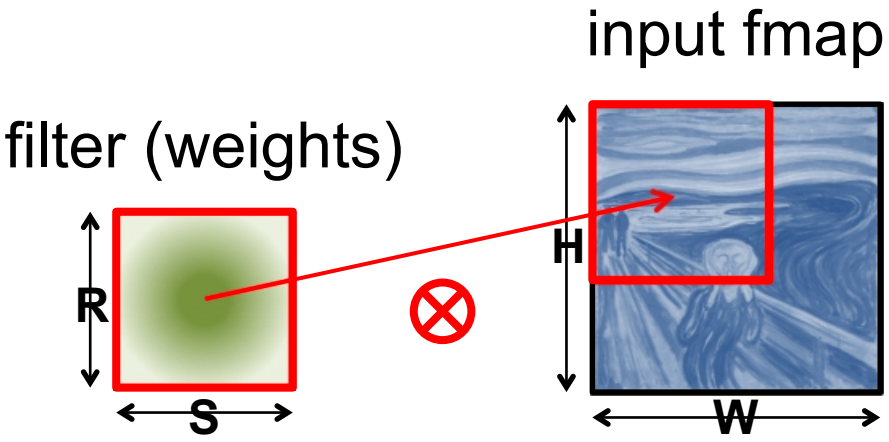
Convolution (CONV) Layer

a plane of input activations
a.k.a. **input feature map (fmap)**

filter (weights)

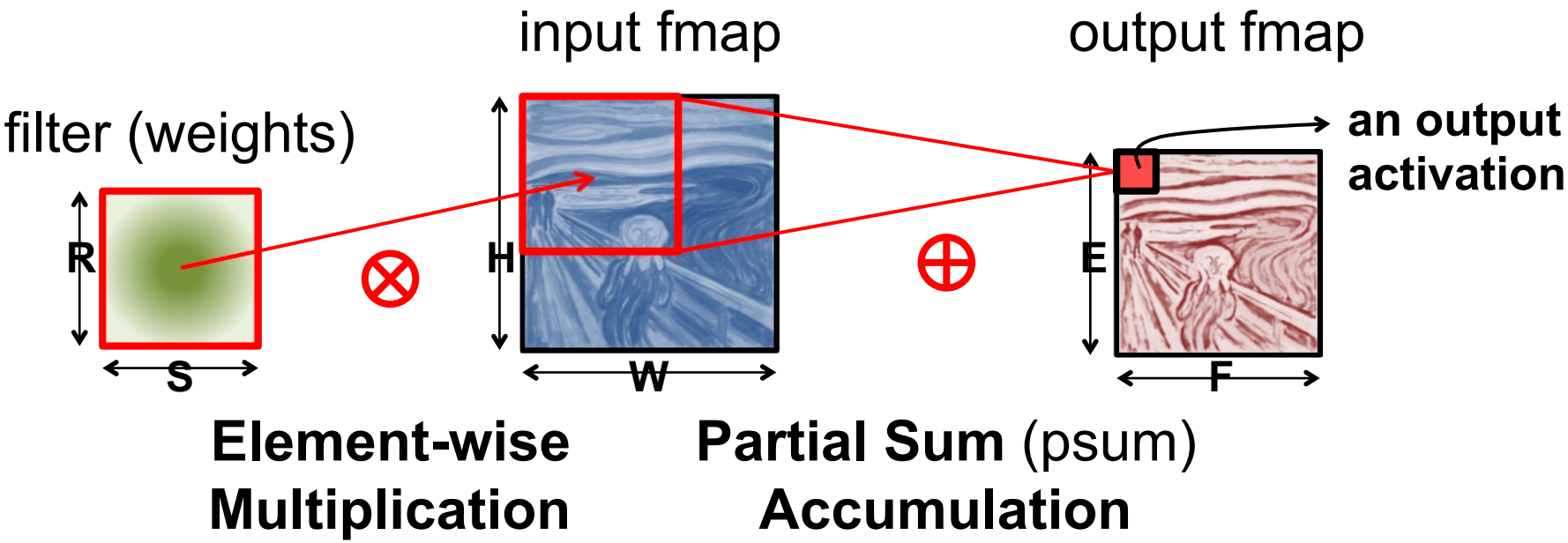


Convolution (CONV) Layer

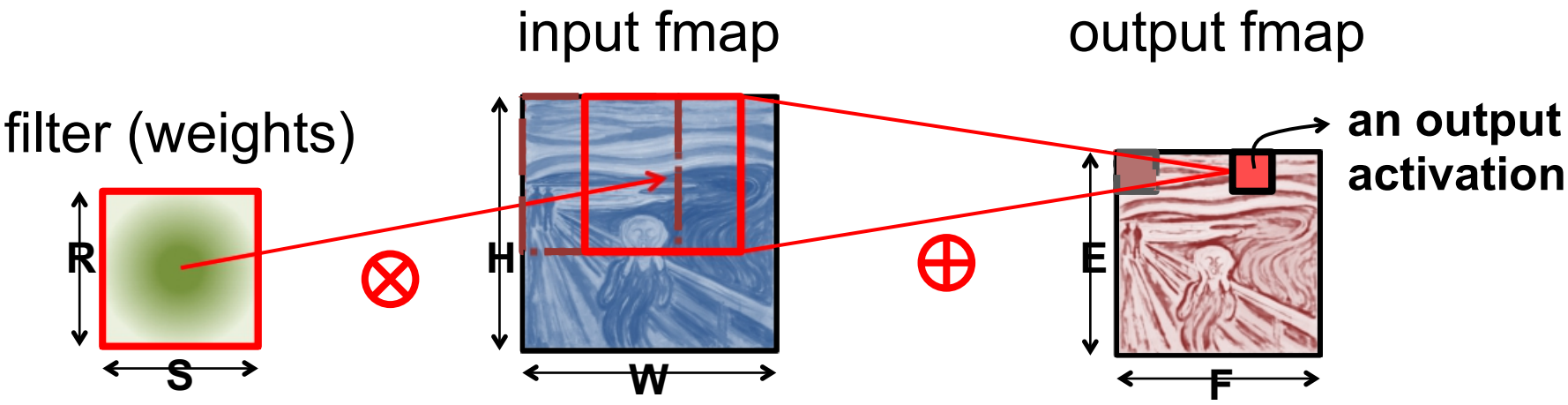


Element-wise Multiplication

Convolution (CONV) Layer

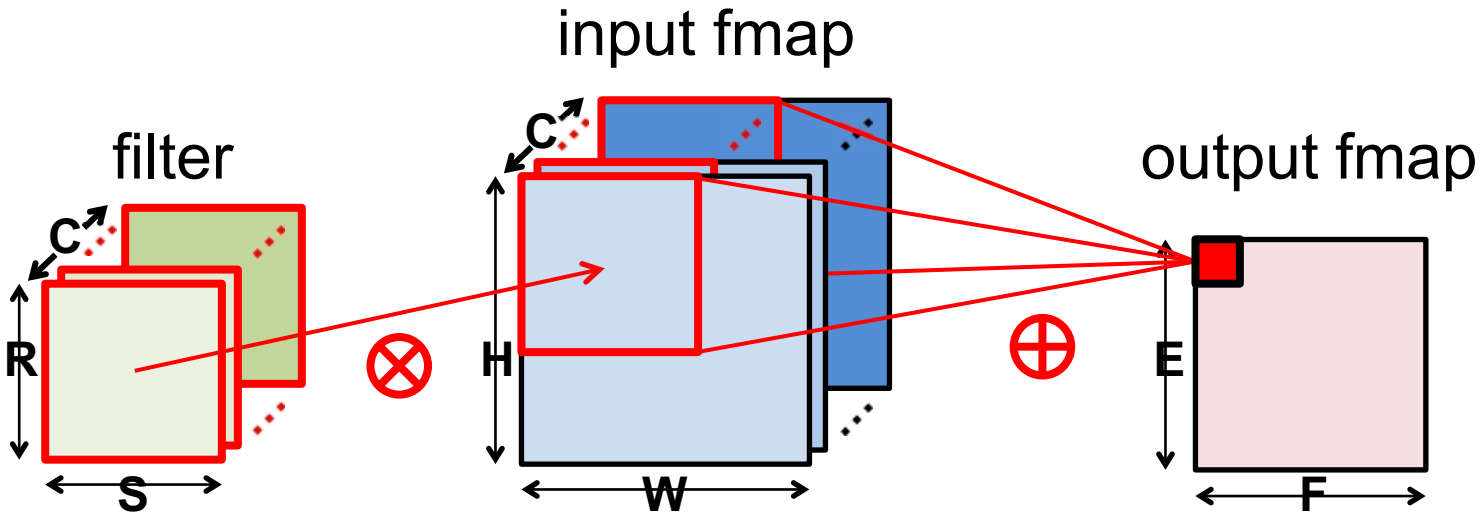


Convolution (CONV) Layer



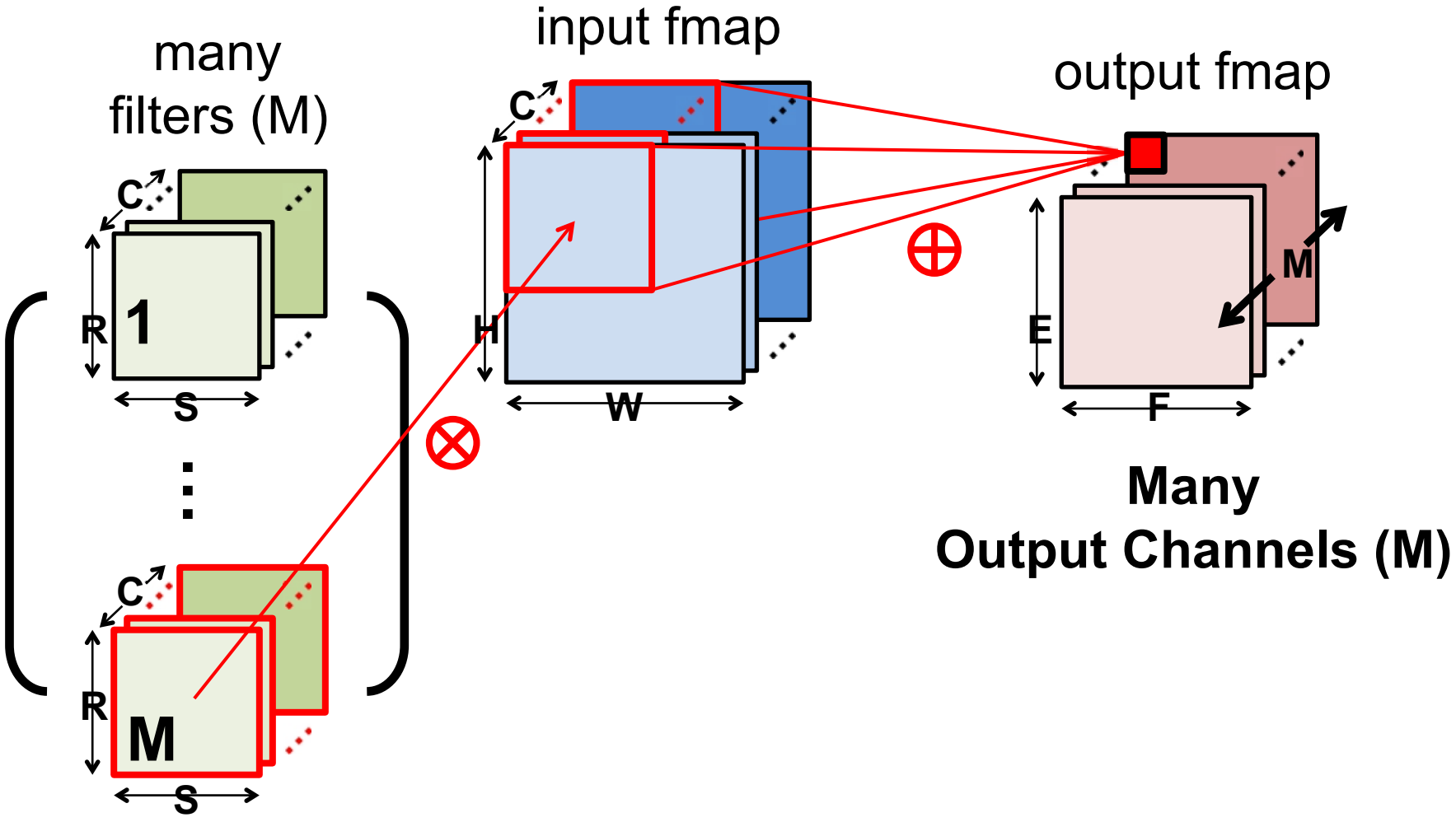
Sliding Window Processing

Convolution (CONV) Layer

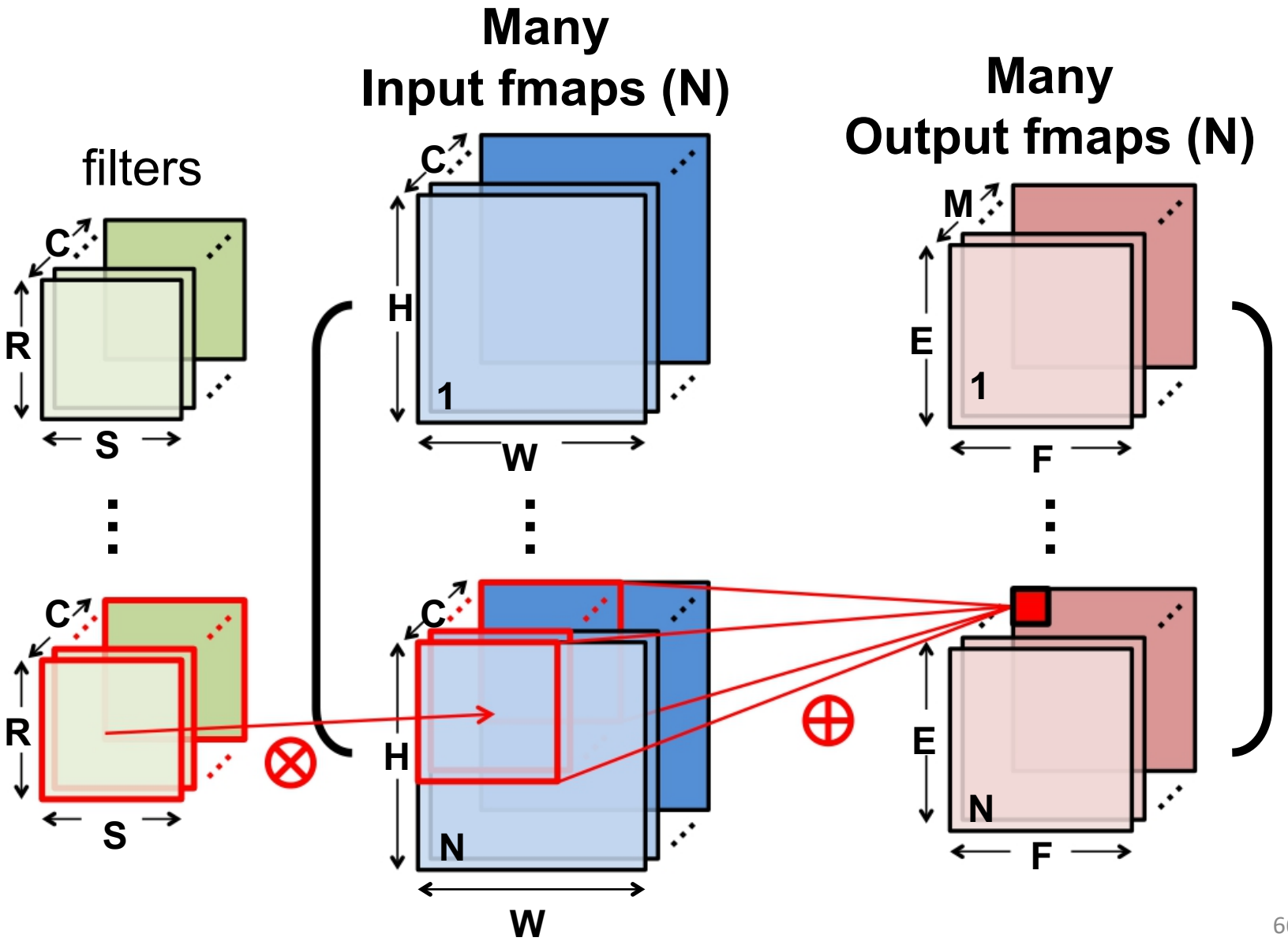


Many Input Channels (C)

Convolution (CONV) Layer



Convolution (CONV) Layer



CNN Decoder Ring

- N – Number of input fmaps/output fmaps (batch size)
- C – Number of 2-D input fmaps /filters (channels)
- H – Height of input fmap (activations)
- W – Width of input fmap (activations)
- R – Height of 2-D filter (weights)
- S – Width of 2-D filter (weights)
- M – Number of 2-D output fmaps (channels)
- E – Height of output fmap (activations)
- F – Width of output fmap (activations)

CONV Layer Tensor Computation

Output fmaps (O)

Input fmaps (I)

Biases (B)

Filter weights (W)

$$\mathbf{O}[n][m][x][y] = \text{Activation}(\mathbf{B}[m] + \sum_{i=0}^{R-1}\sum_{j=0}^{S-1}\sum_{k=0}^{C-1} \mathbf{I}[n][k][Ux+i][Uy+j] \times \mathbf{W}[m][k][i][j]),$$

$$0 \leq n < N, 0 \leq m < M, 0 \leq y < E, 0 \leq x < F,$$

$$E = (H - R + U)/U, F = (W - S + U)/U.$$

Shape Parameter	Description
N	fmap batch size
M	# of filters / # of output fmap channels
C	# of input fmap/filter channels
H/W	input fmap height/width
R/S	filter height/width
E/F	output fmap height/width
U	convolution stride

CONV Layer Implementation

Naïve 7-layer for-loop implementation:

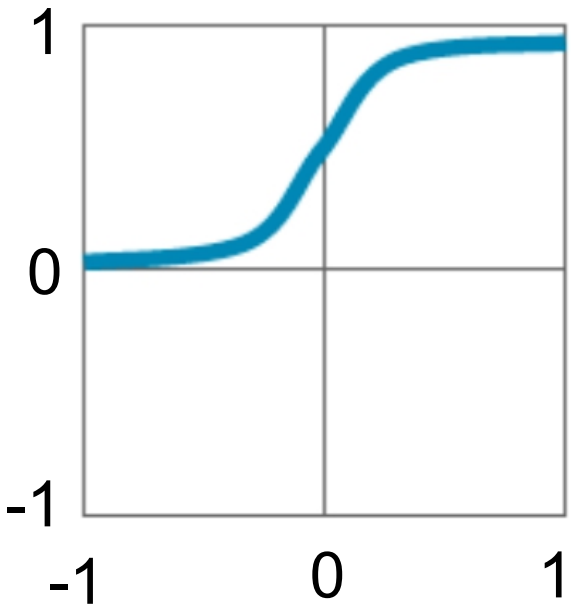
```
for (n=0; n<N; n++) {
    for (m=0; m<M; m++) {
        for (x=0; x<F; x++) {
            for (y=0; y<E; y++) {
                O[n][m][x][y] = B[m];
                for (i=0; i<R; i++) {
                    for (j=0; j<S; j++) {
                        for (k=0; k<C; k++)
                {           O[n][m][x][y] += I[n][k][Ux+i][Uy+j] × W[m][k][i][j];
                        }
                    }
                }

                O[n][m][x][y] = Activation(O[n][m][x][y]);
            }
        }
    }
}
```

for each output fmap value

convolve a window and apply activation

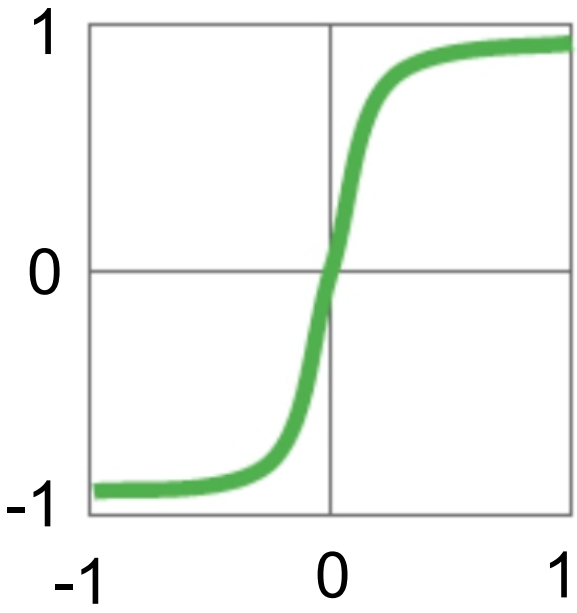
Traditional Activation Functions

Sigmoid



$$y=1/(1+e^{-x})$$

Hyperbolic Tangent

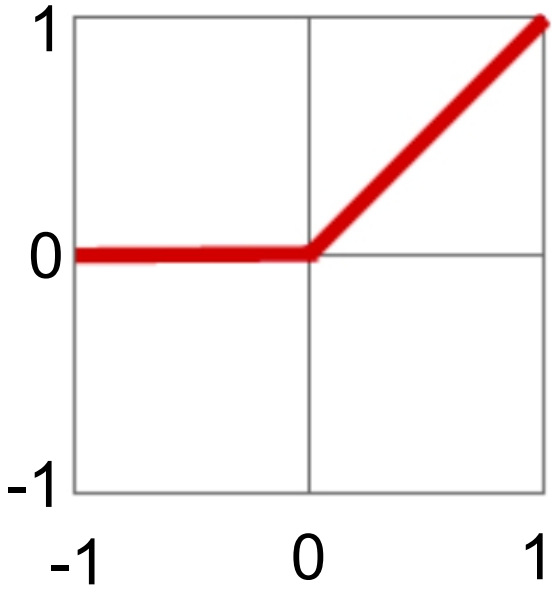


$$y=(e^{x}-e^{-x})/(e^{x}+e^{-x})$$

Image Source: Caffe Tutorial

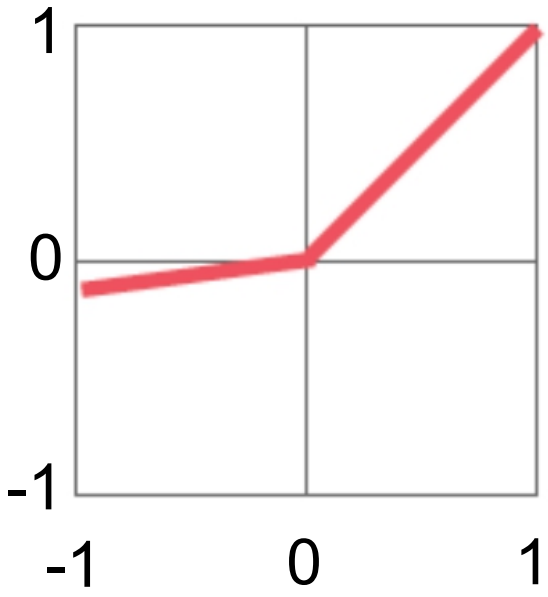
Modern Activation Functions

Rectified Linear Unit (ReLU)



$y = \max(0, x)$

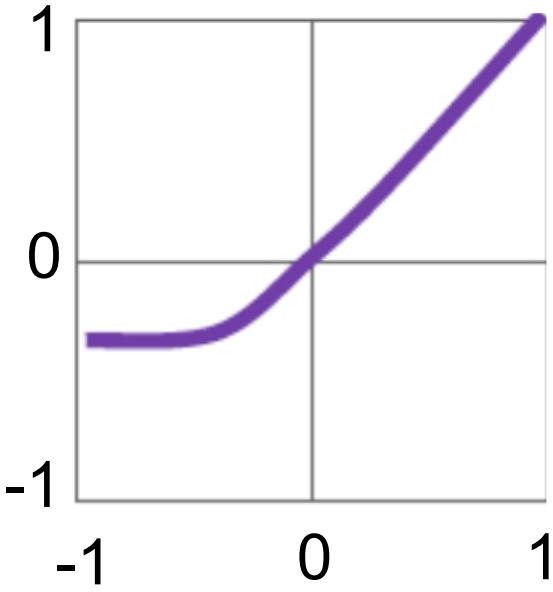
Leaky ReLU



$y = \max(\alpha x, x)$

α = small const. (e.g. 0.1)

Exponential LU

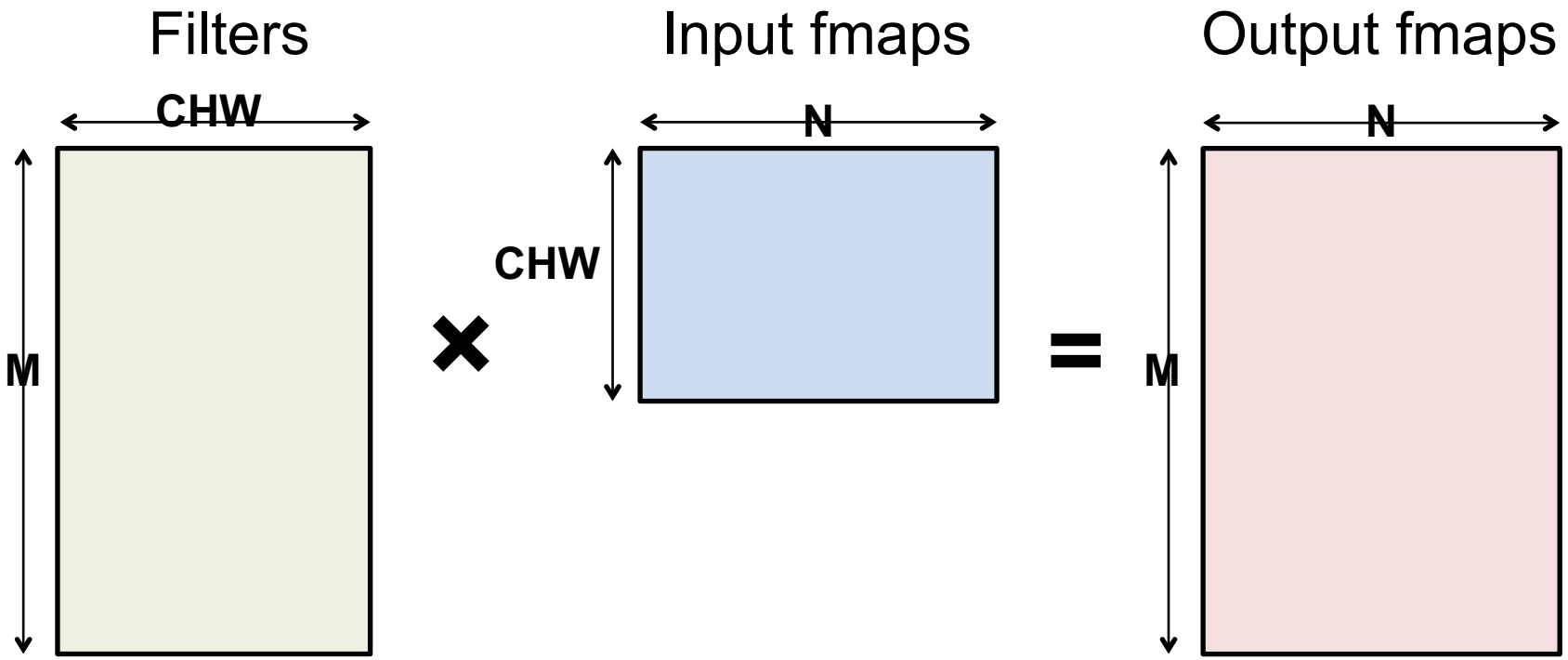


$$y = \begin{cases} x, & x \geq 0 \\ \alpha(e^x - 1), & x < 0 \end{cases}$$

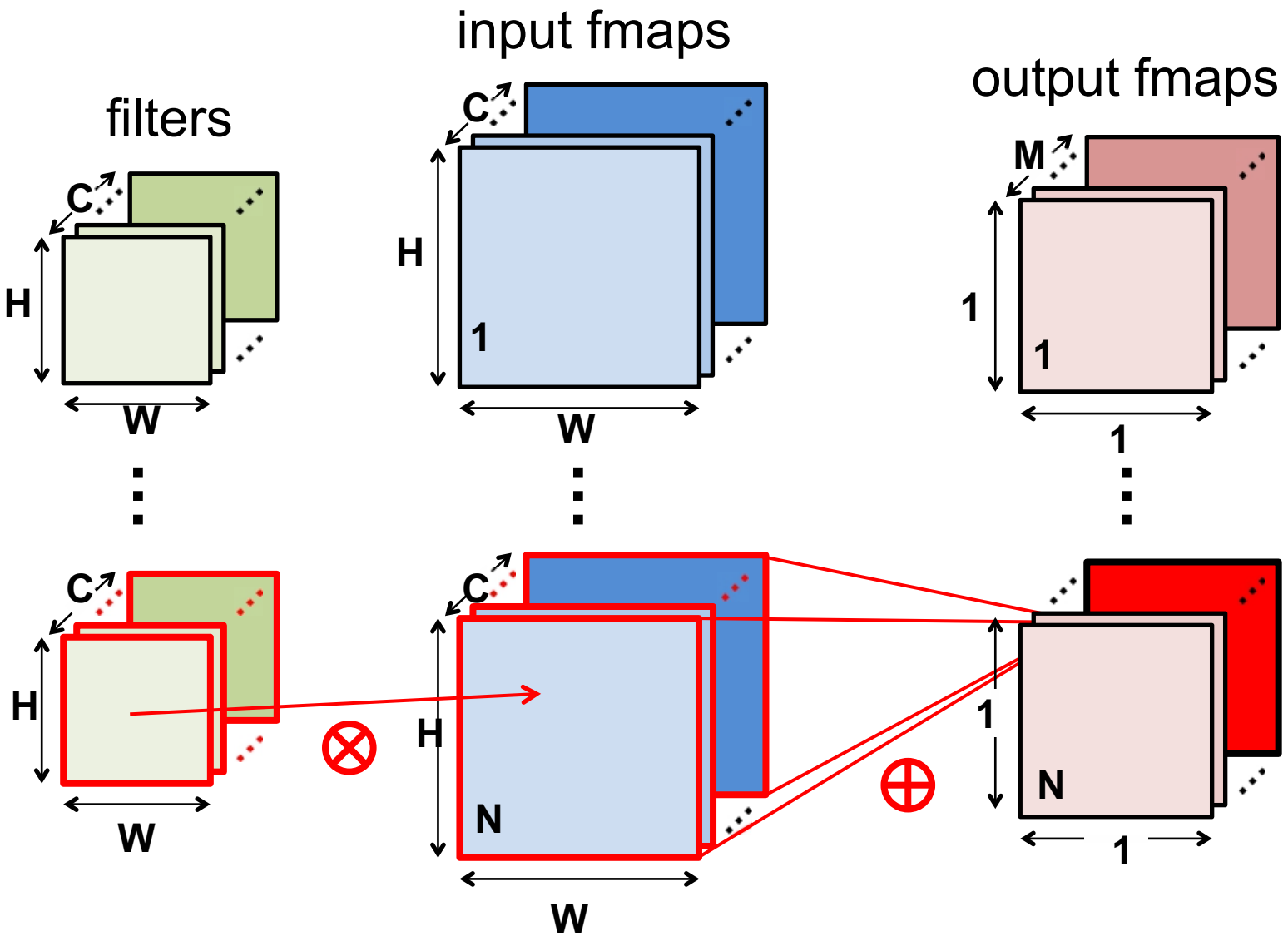
Image Source: Caffe Tutorial

Fully-Connected (FC) Layer

- Height and width of output fmaps are 1 (E = F = 1)
- Filters as large as input fmaps (R = H, S = W)
- Implementation: **Matrix Multiplication**

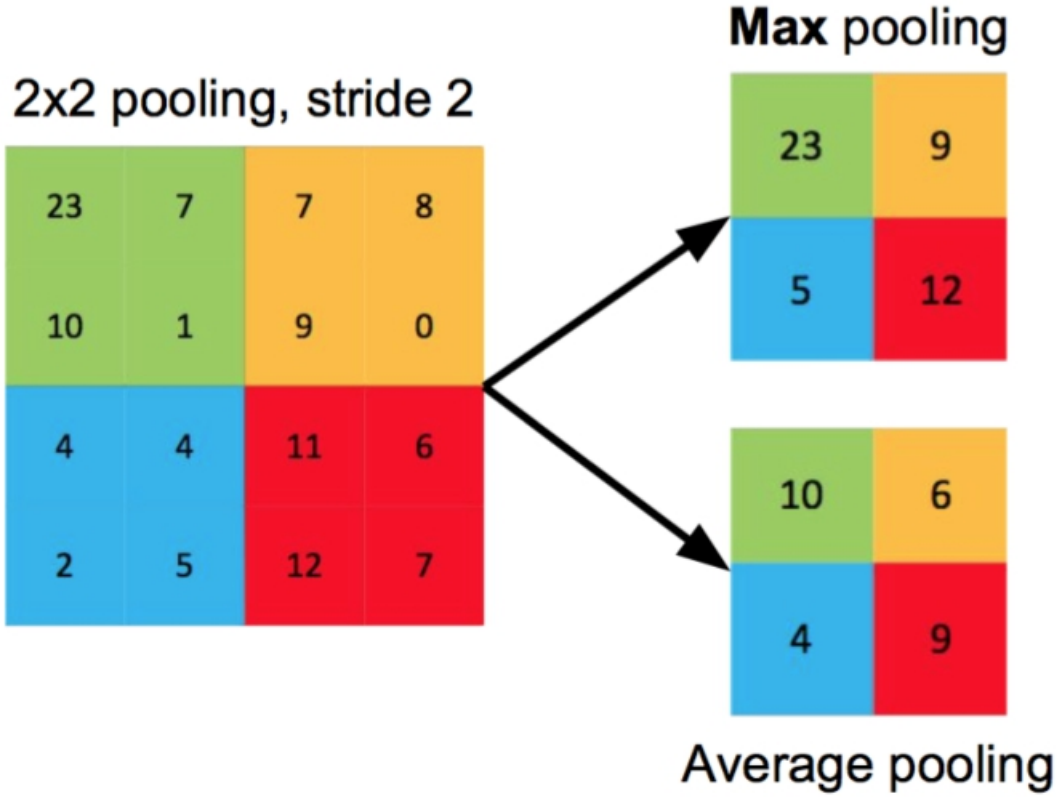


FC Layer – from CONV Layer POV



Pooling (POOL) Layer

- Reduce resolution of each channel independently
- Overlapping or non-overlapping →depending on stride



Increases translation-invariance and noise-resilience

Image Source: Caffe Tutorial

POOL Layer Implementation

Naïve 6-layer for-loop max-pooling implementation:

```
for (n=0; n<N; n++) {
    for (m=0; m<M; m++) {
        for (x=0; x<F; x++) {
            for (y=0; y<E; y++) {

                max = -Inf;
                for (i=0; i<R; i++) {
                    for (j=0; j<S; j++) {
                        if (I[n][m][Ux+i][Uy+j] > max)
                    {
                            max = I[n][m][Ux+i][Uy+j];
                    }   }

                }

                O[n][m][x][y] = max;
            }
        }
    }
}
```

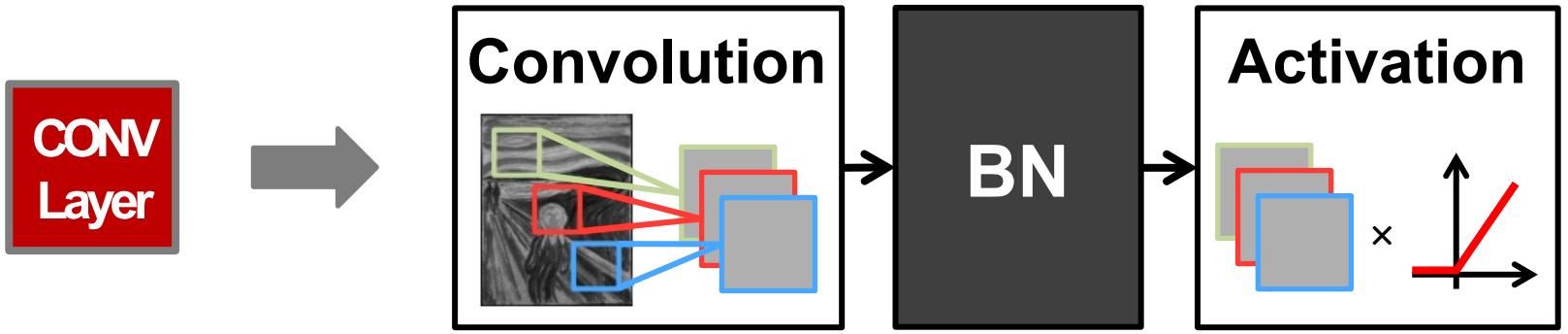
for each pooled value

find the max with in a window

Normalization (NORM) Layer

- **Batch Normalization (BN)**
  - Normalize activations towards mean=0 and std. dev.=1 based on the statistics of the training dataset –

put **in between** **CONV/FC** and **Activation function**

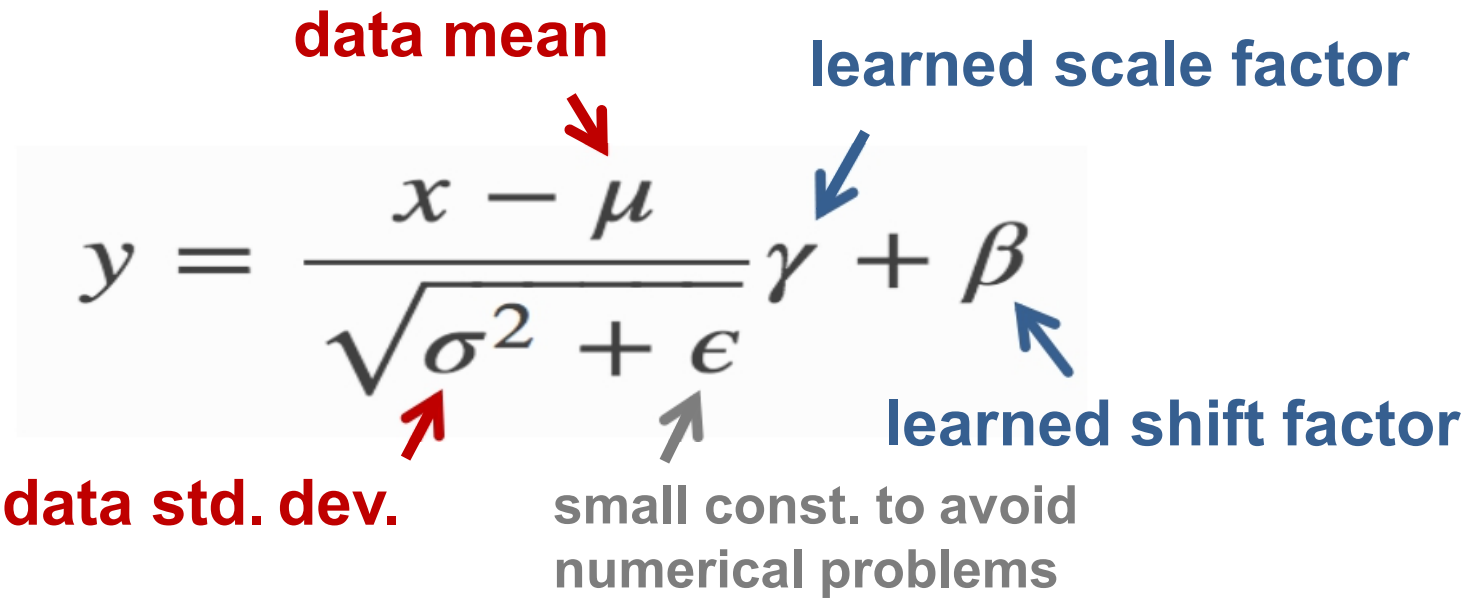


Believed to be key to getting high accuracy and faster training on very deep neural networks.

[Ioffe et al., ICML 2015]

BN Layer Implementation

- The normalized value is further scaled and shifted, the parameters of which are learned from training



Relevant Components for Tutorial

- Typical operations that we will discuss:
  - Convolution (CONV)
  - Fully-Connected (FC)
  - Max Pooling
  - ReLU

Backup Slides