Deep Learning Part I

Instructor: Prof. Yi Fang

yfang@nyu.edu

Python tutorial: http://learnpython.org/

TensorFlow tutorial: https://www.tensorflow.org/tutorials/

PyTorch tutorial: https://pytorch.org/tutorials/

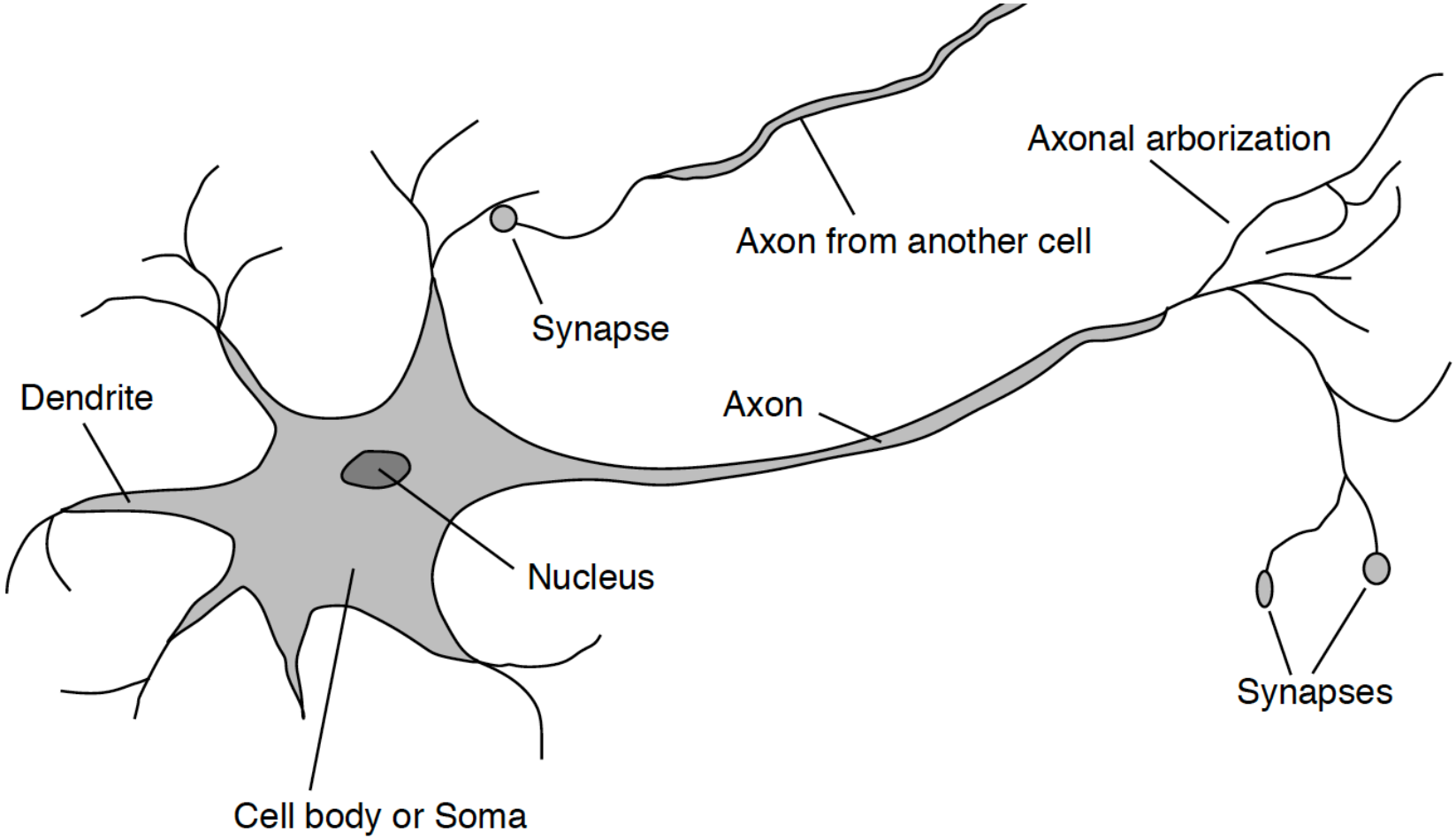
Acknowledge: The slides are partially referred to coursera online machine learning course by Prof. Andrew Ng, and NYU machine learning course. All copyrights owned by original authors. (i.e. http://u.cs.biu.ac.il/~89-570/)

Overview

- Human Brain
- Neural Network
- Perceptrons
- Multilayer Perceptrons
- Convolution Neural Network
  - Overview architecture
  - Convolution operation
  - Convolution layer
  - Pooling layer
  - Fully connection layer

Human Brain

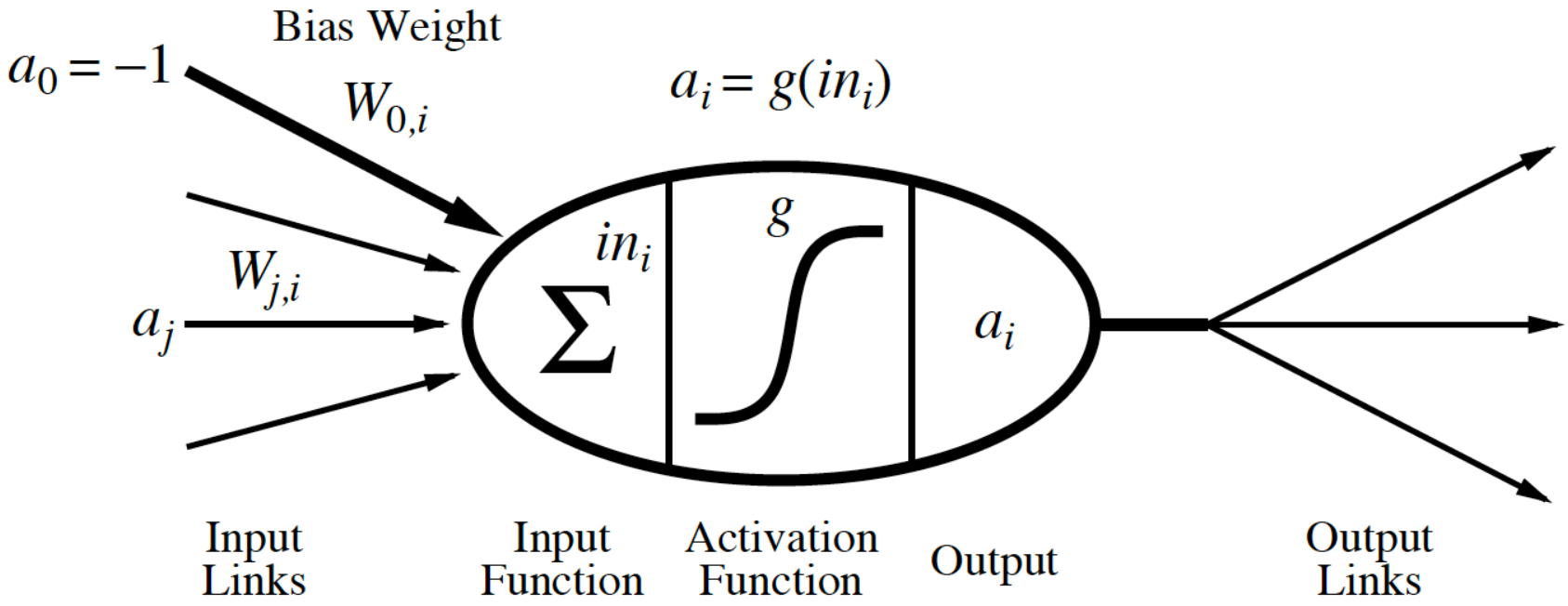
- 10^{11} neurons of > 20 types, 10^{14} synapses, 1ms-10ms cycle time
- Signals are noisy “spike trains” of electrical potential



Neurons

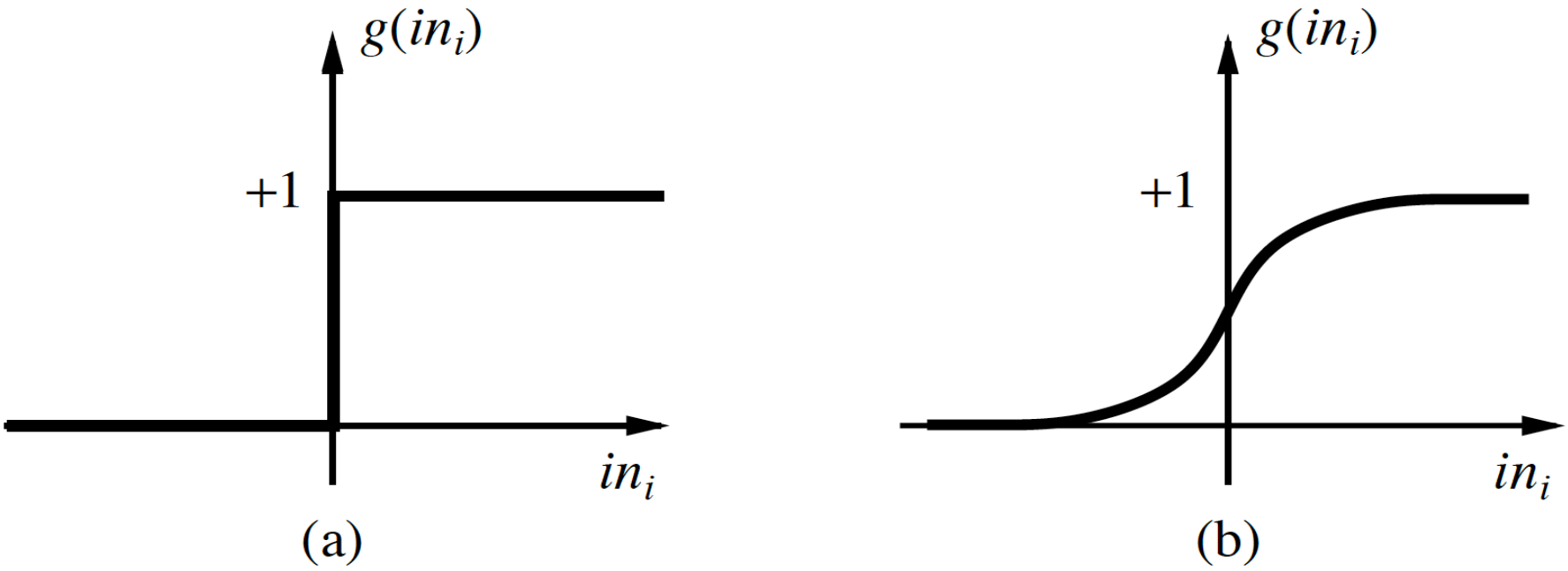
- A gross oversimplication of real neurons

$$a_i \leftarrow g(in_i) = g\left(\sum_j W_{j,i} a_j\right)$$



Human Brain

- Activation function



(a) is a step function or threshold function

(b) is a sigmoid function $1/(1+e^{-x})$

Changing the bias weight $W_{0,i}$ moves the threshold location

Network structure

- Feed-forward Networks
  - Single-layer perceptrons
  - Multi-layer perceptrons

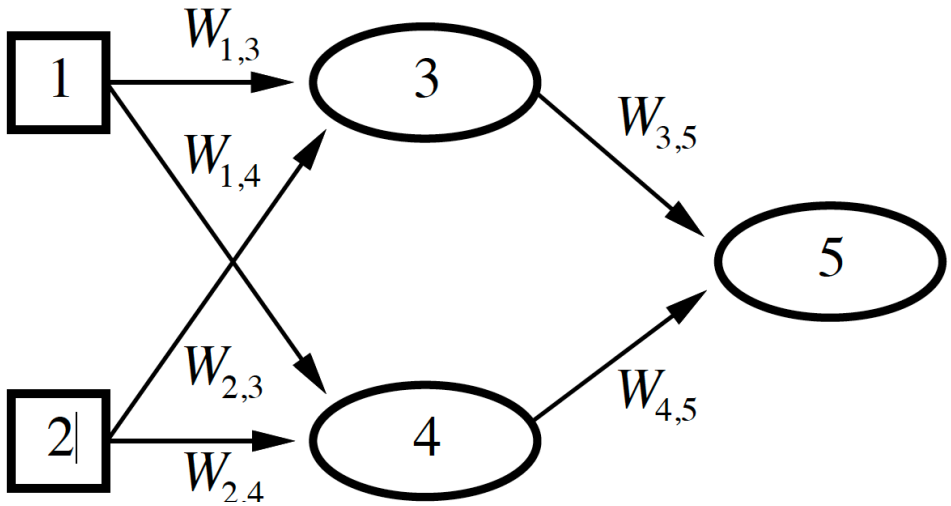
  Feed-forward networks implement functions which have no internal state
- Recurrent Network
  - Hopfield networks
  - Boltzmann machines

  Recurrent neural nets have directed cycles with delays and have internal state (like ip-ops)

Feed-forward Networks

- Feed-forward network = a parameterized family of nonlinear functions
- Adjusting weights changes the function

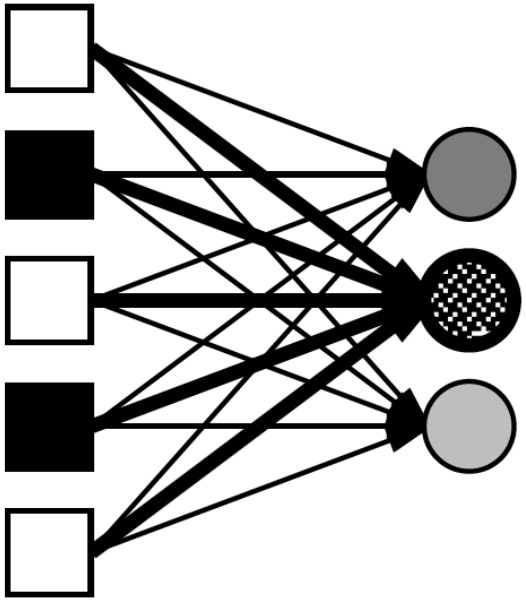
An easy example:

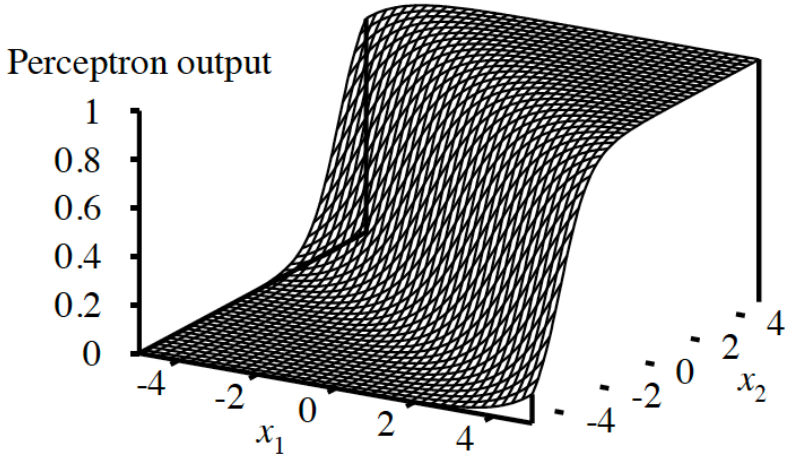


$$a_5 = g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4)$$
$$= g(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2))$$

Single-layer perceptrons

- Output units all operate separately an no shared weights
- Adjusting weights moves the location, orientation, and steepness of cliff





Perceptron learning

Learn by adjusting weights to reduce error on training set

The squared error for an example with input $\mathbf{x}$ and true output y is

$$E = \frac{1}{2}Err^2 \equiv \frac{1}{2}(y - h_{\mathbf{W}}(\mathbf{x}))^2 \ ,$$

Perform optimization search by gradient descent:

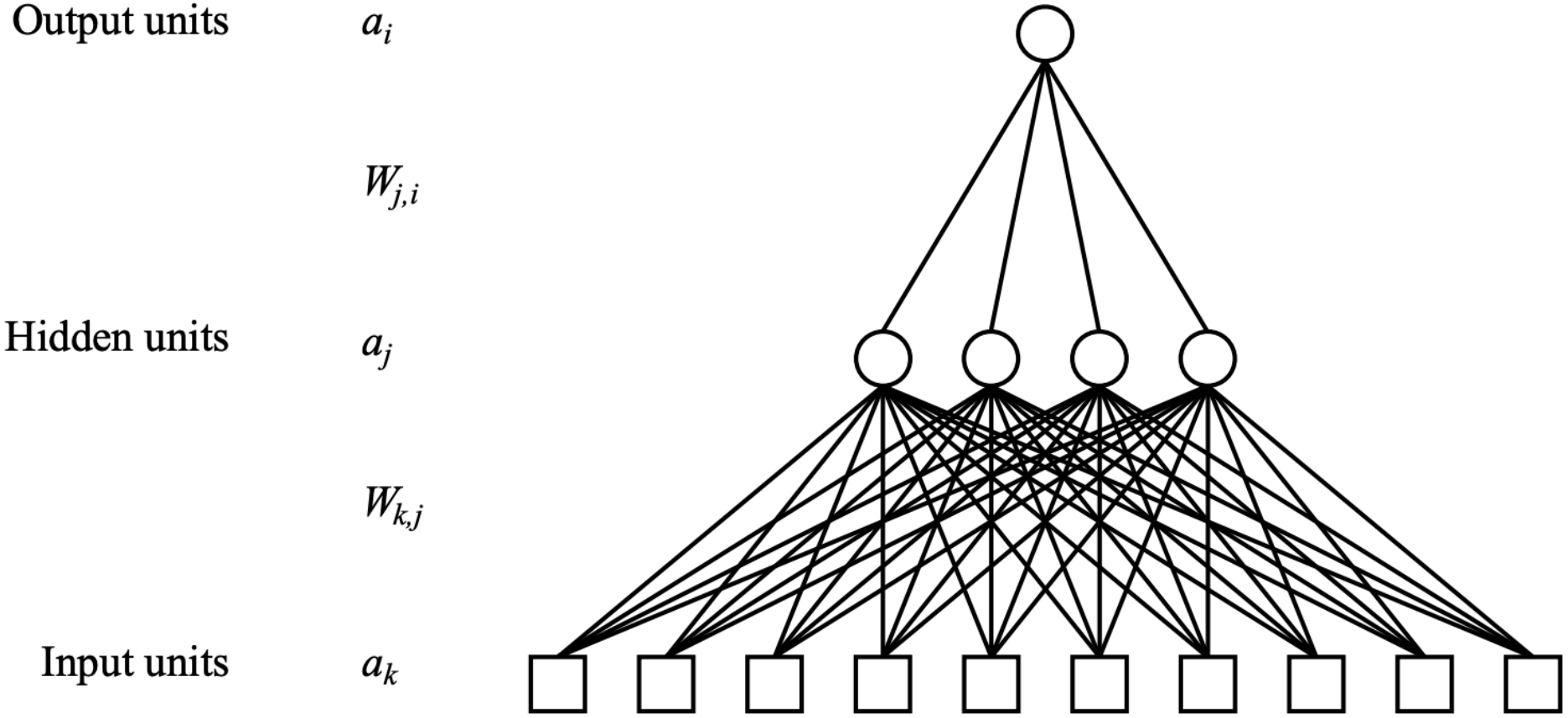
$$\begin{aligned} \frac{\partial E}{\partial W_j} &= Err \times \frac{\partial Err}{\partial W_j} = Err \times \frac{\partial}{\partial W_j}\left(y - g(\Sigma_{j=0}^{n} W_j x_j)\right) \\ &= -Err \times g'(in) \times x_j \end{aligned}$$

Simple weight update rule:

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$$

Multilayer perceptrons

- Layers are usually fully connected
- Numbers of hidden units typically chosen by hand



Back-propagation learning

Output layer: same as for single-layer perceptron,

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$

where $\Delta_i = Err_i \times g'(in_i)$

Hidden layer: **back-propagate** the error from the output layer:

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i \ .$$

Update rule for weights in hidden layer:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j \ .$$

Back-propagation example

- Loss is defined as:

$$E = \frac{1}{2}\sum_i (y_i - a_i)^2 \ ,$$

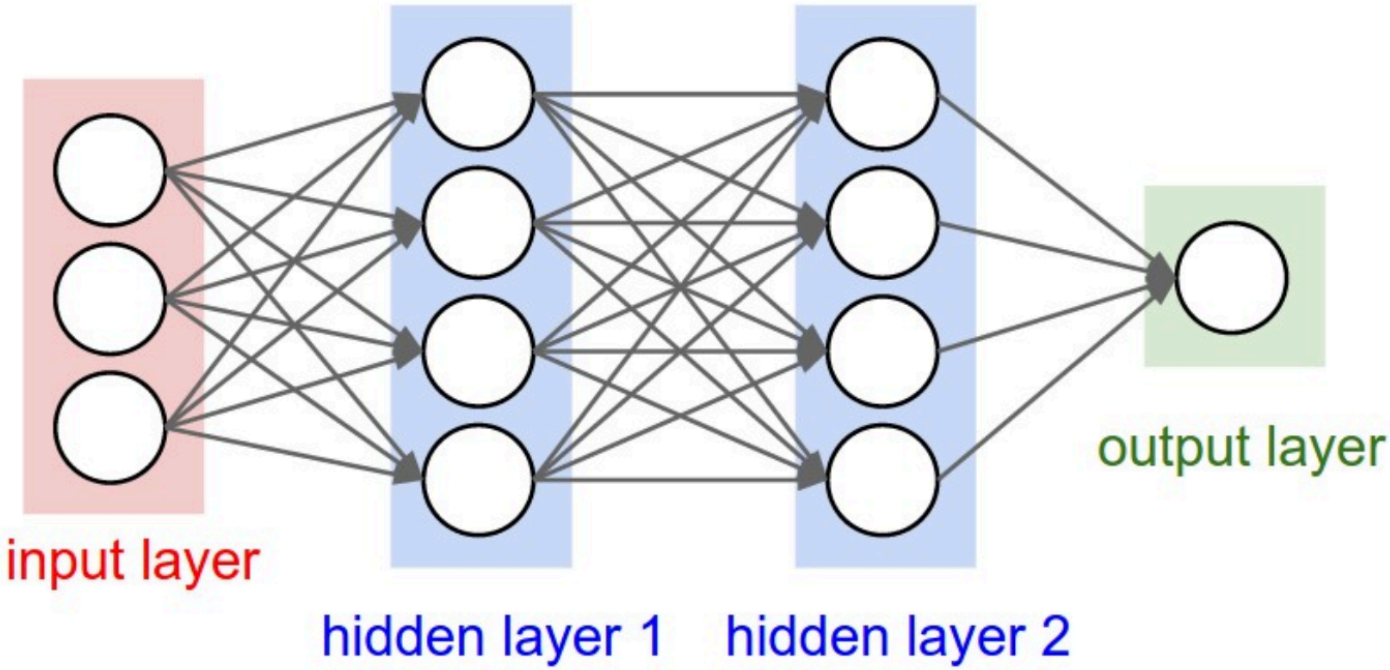
- where the sum is over the nodes in the output layer

$$\begin{aligned}
\frac{\partial E}{\partial W_{j,i}} &= -(y_i - a_i)\frac{\partial a_i}{\partial W_{j,i}} = -(y_i - a_i)\frac{\partial g(in_i)}{\partial W_{j,i}} \\
&= -(y_i - a_i)g'(in_i)\frac{\partial in_i}{\partial W_{j,i}} = -(y_i - a_i)g'(in_i)\frac{\partial}{\partial W_{j,i}}\left(\sum_j W_{j,i}a_j\right) \\
&= -(y_i - a_i)g'(in_i)a_j = -a_j\Delta_i
\end{aligned}$$

Convolution Neural Network (CNN)

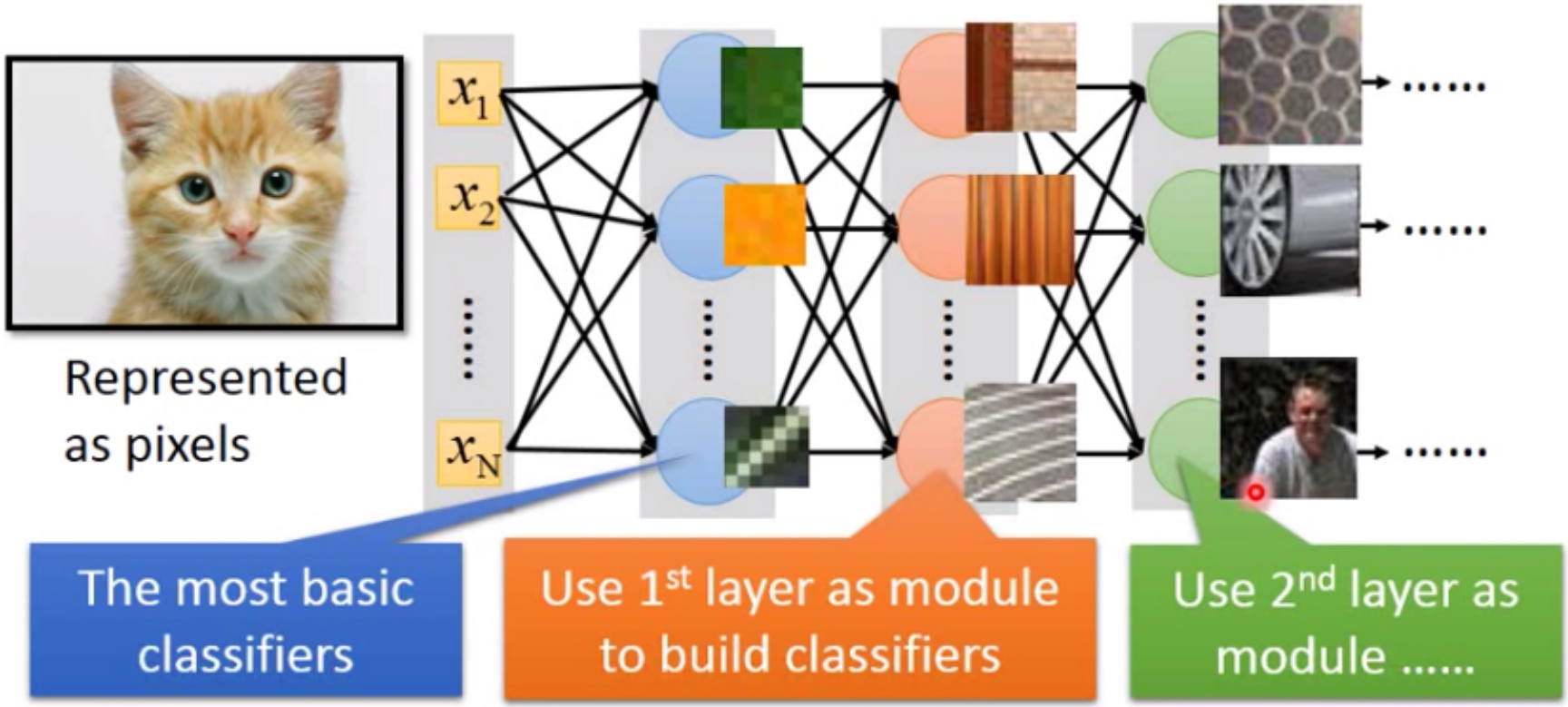
Why CNN?

- A regular 3-layer Neural Network.



Why CNN for Image?

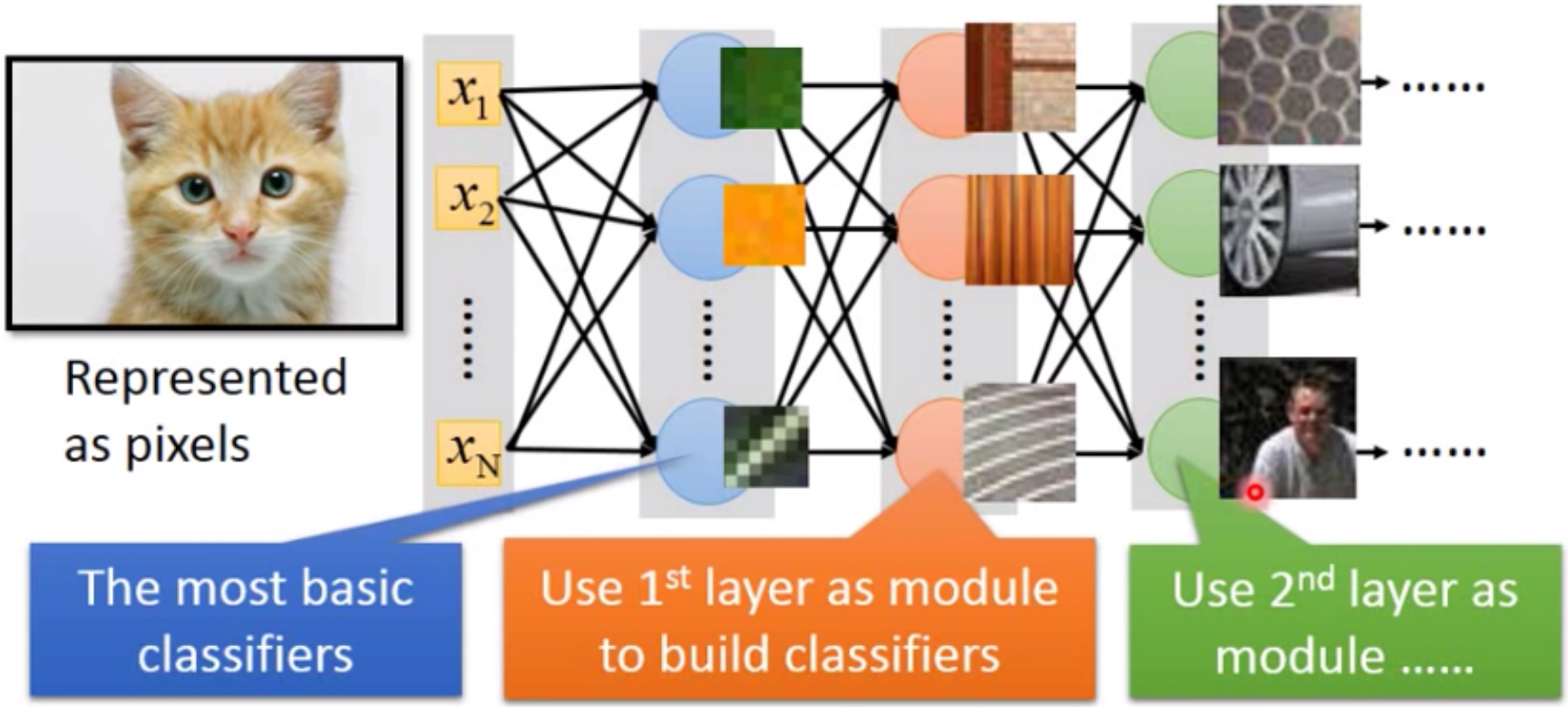
[Zeiler, M. D., *ECCV 2014*]



Redundant Parameters?

Why CNN for Image?

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



Can the Network be simplified?

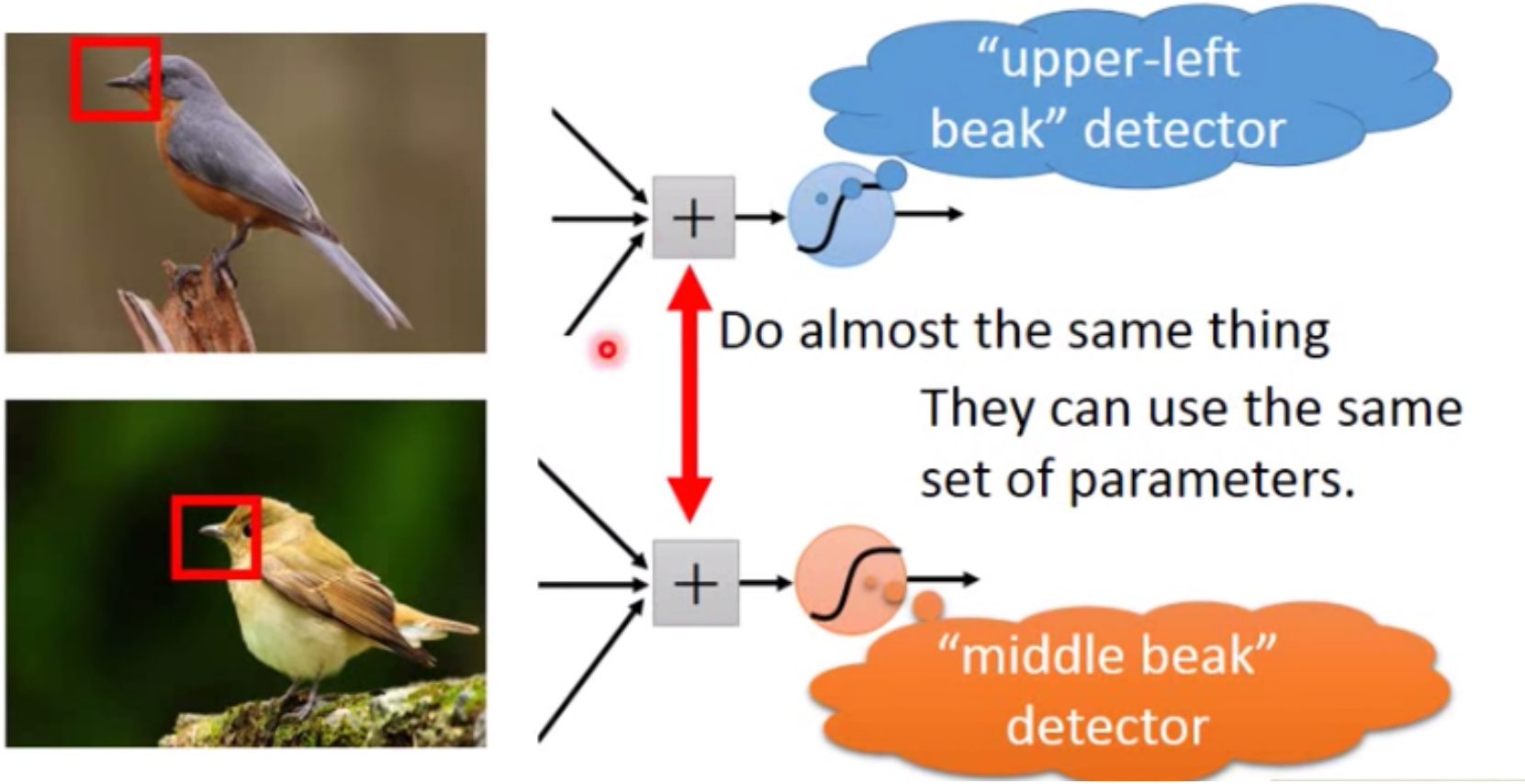
- Focus on local region
- Insensitive to location
- Sub-sample operator

Focus on local region

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

Connecting to small region with less parameters

Insensitive to location



Sub-sampling

- 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

Property 1

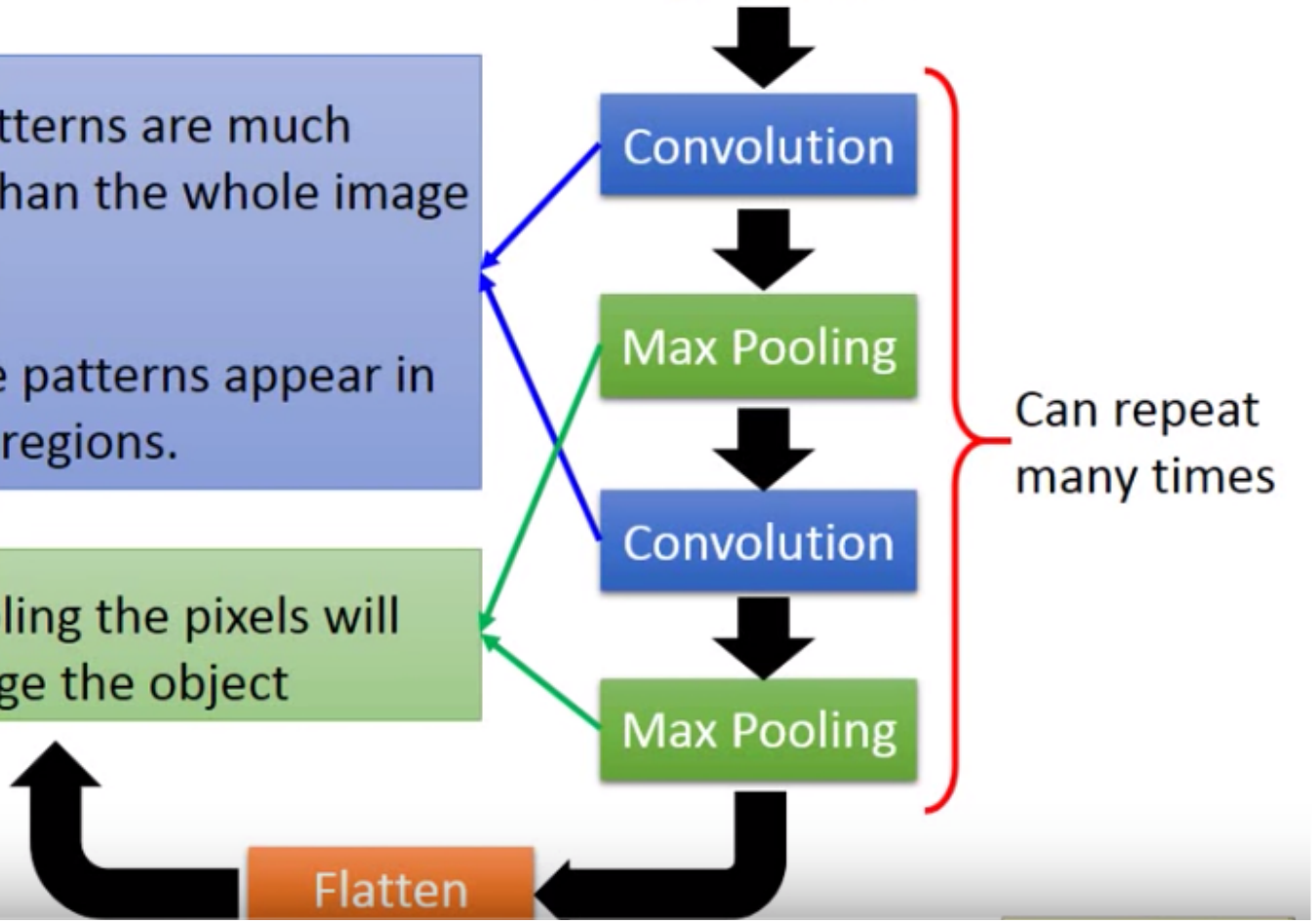
- Some patterns are much smaller than the whole image

Property 2

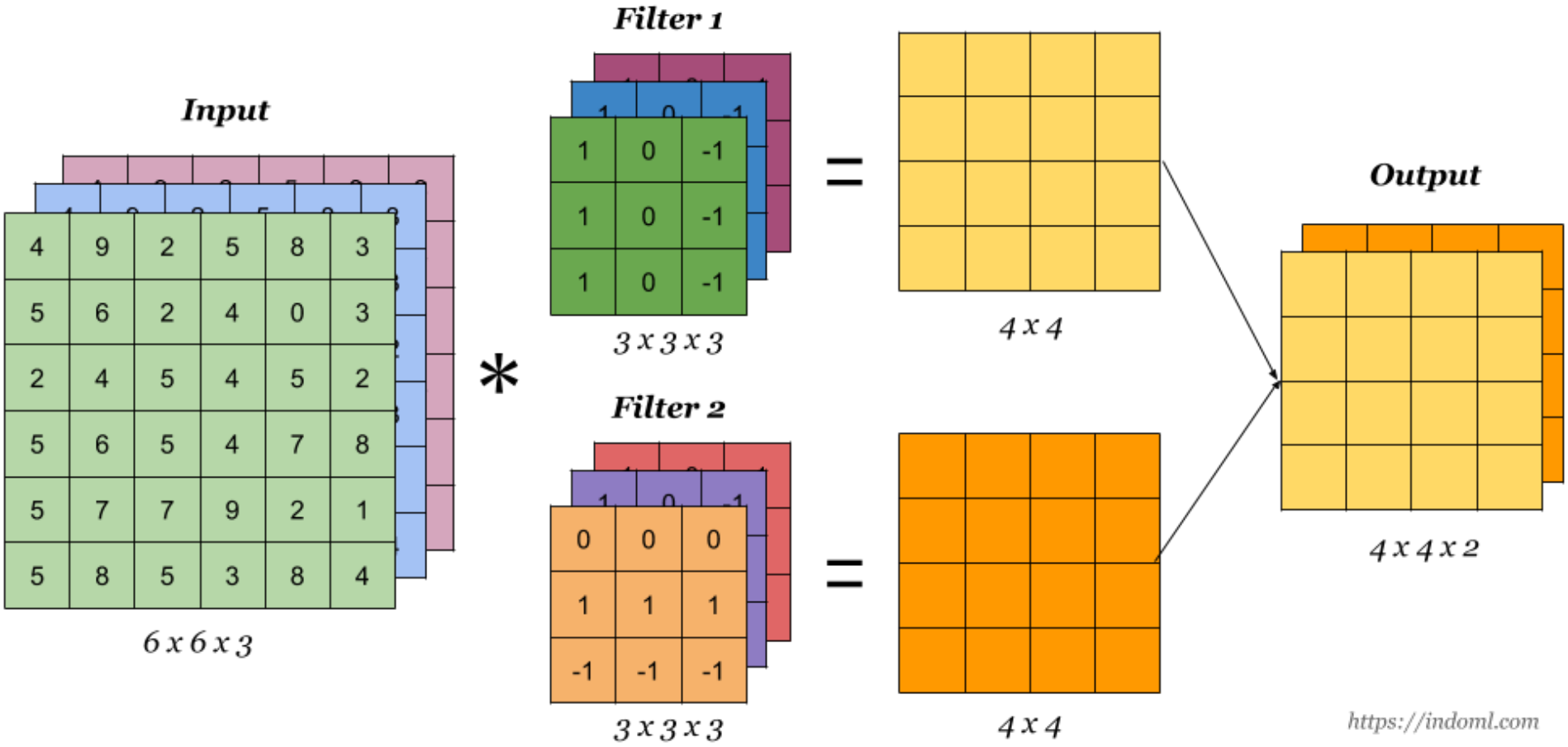
- The same patterns appear in different regions.

Property 3

- Subsampling the pixels will not change the object



Convolution Neural Network

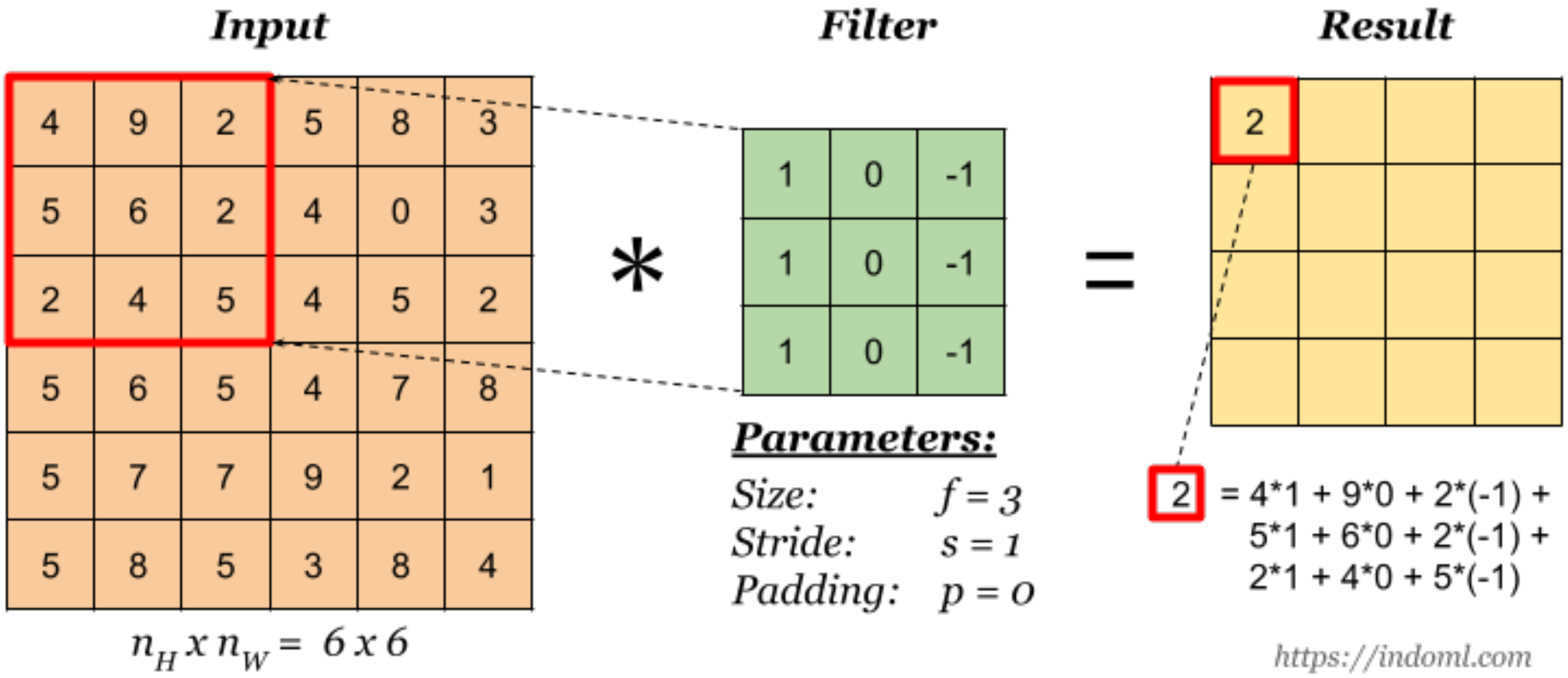


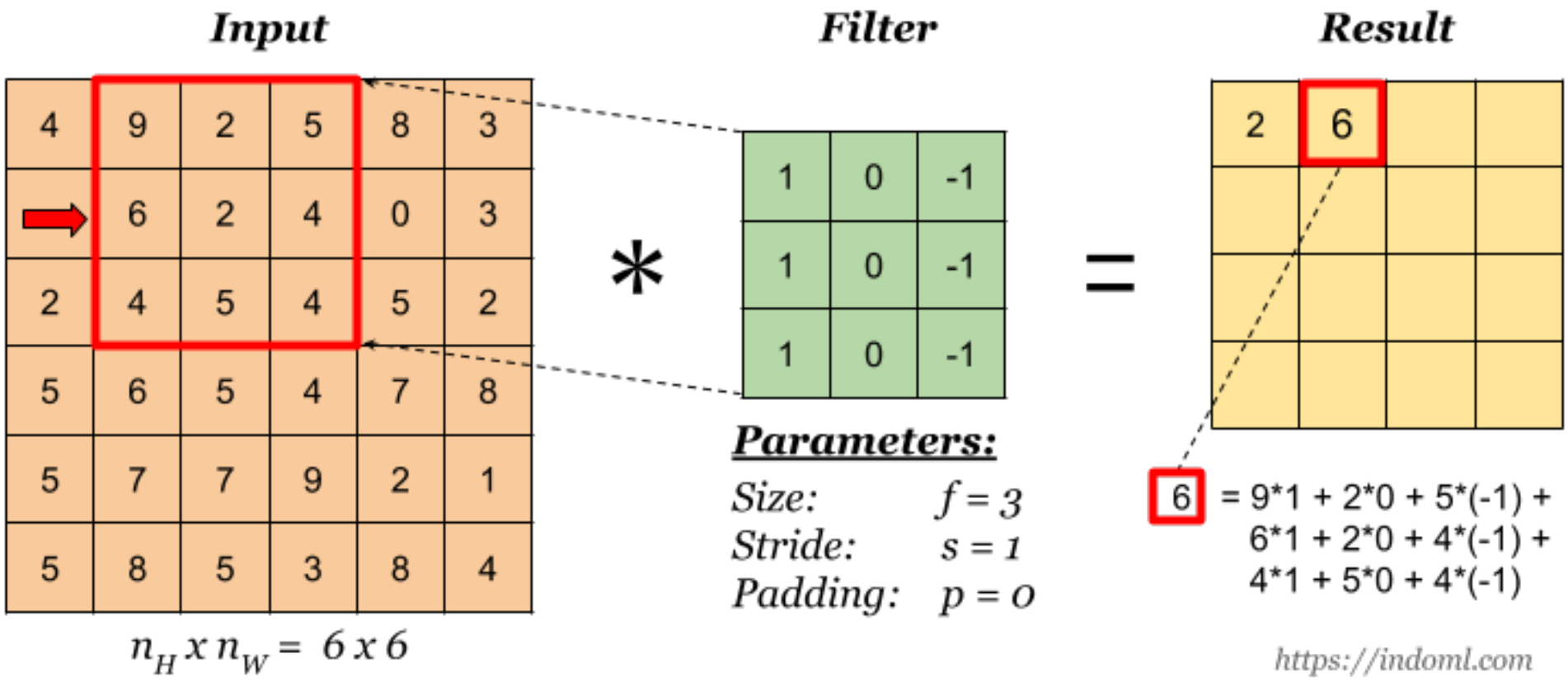
CNN Architectures

- Basics of CNN
- Classic Networks

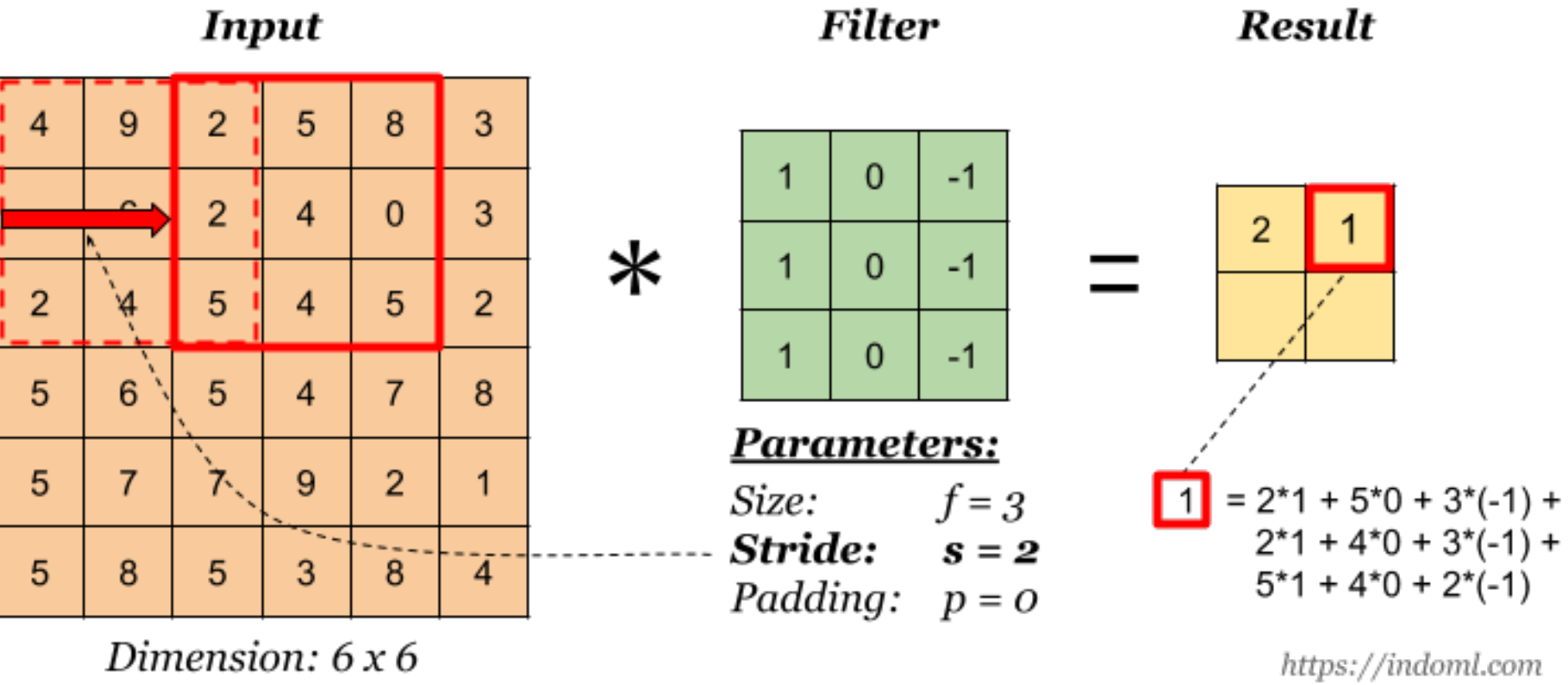
CNN Basics

Basic Convolution Operation

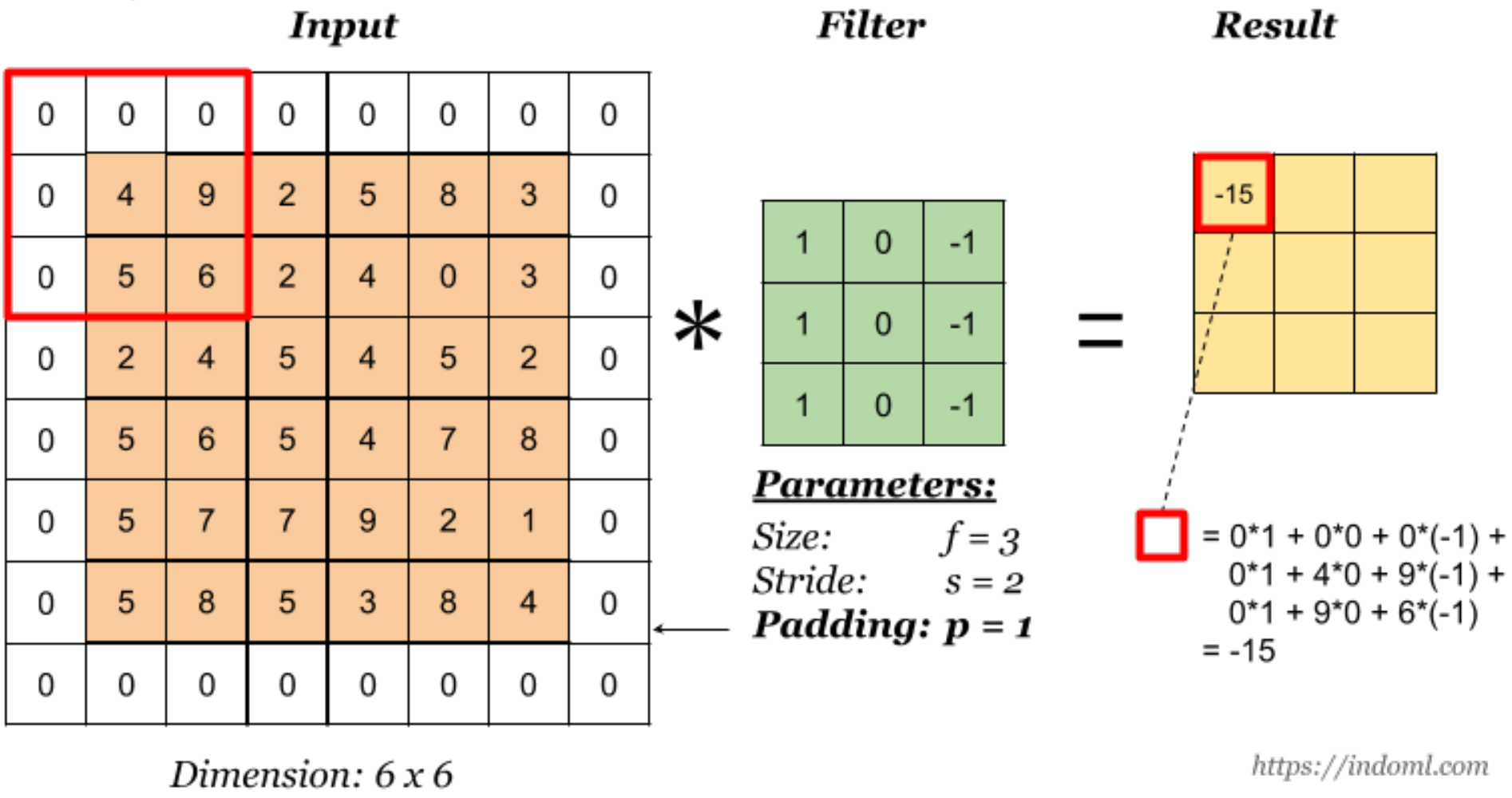



Input
Filter
Result
4 9 2 5 8 3
6 2 4 0 3
2 4 5 4 5 2
5 6 5 4 7 8
5 7 7 9 2 1
5 8 5 3 8 4
1 0 -1
1 0 -1
1 0 -1
*
=
2 6
Parameters:
Size: f = 3
Stride: s = 1
Padding: p = 0
6 = 9*1 + 2*0 + 5*(-1) +
6*1 + 2*0 + 4*(-1) +
4*1 + 5*0 + 4*(-1)
n_H x n_W = 6 x 6
https://indoml.com

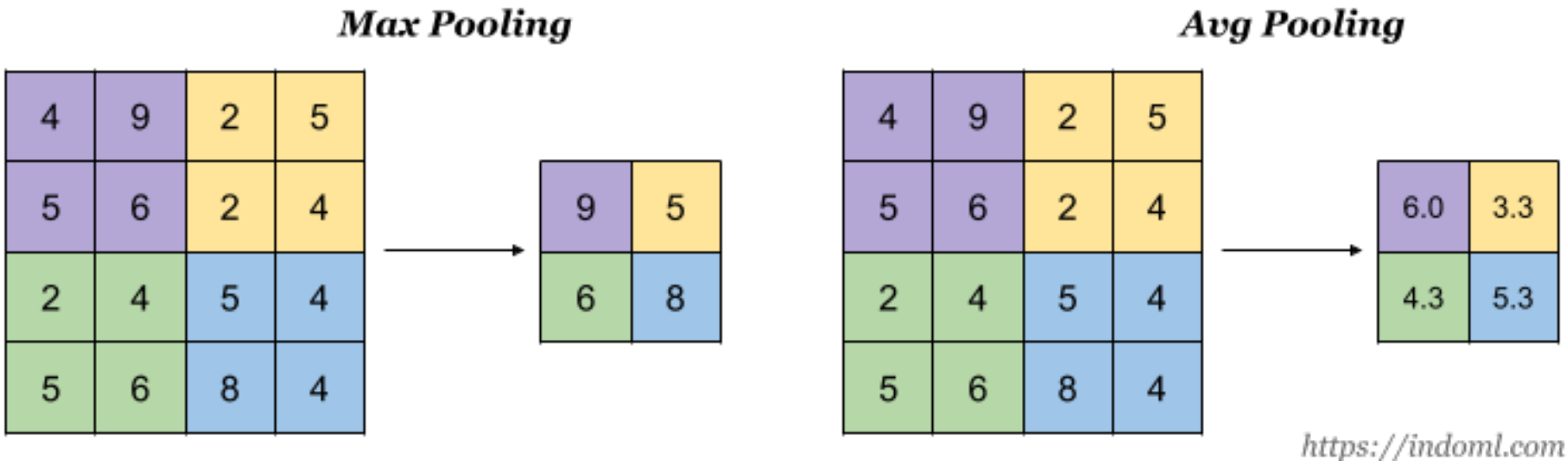
Stride:

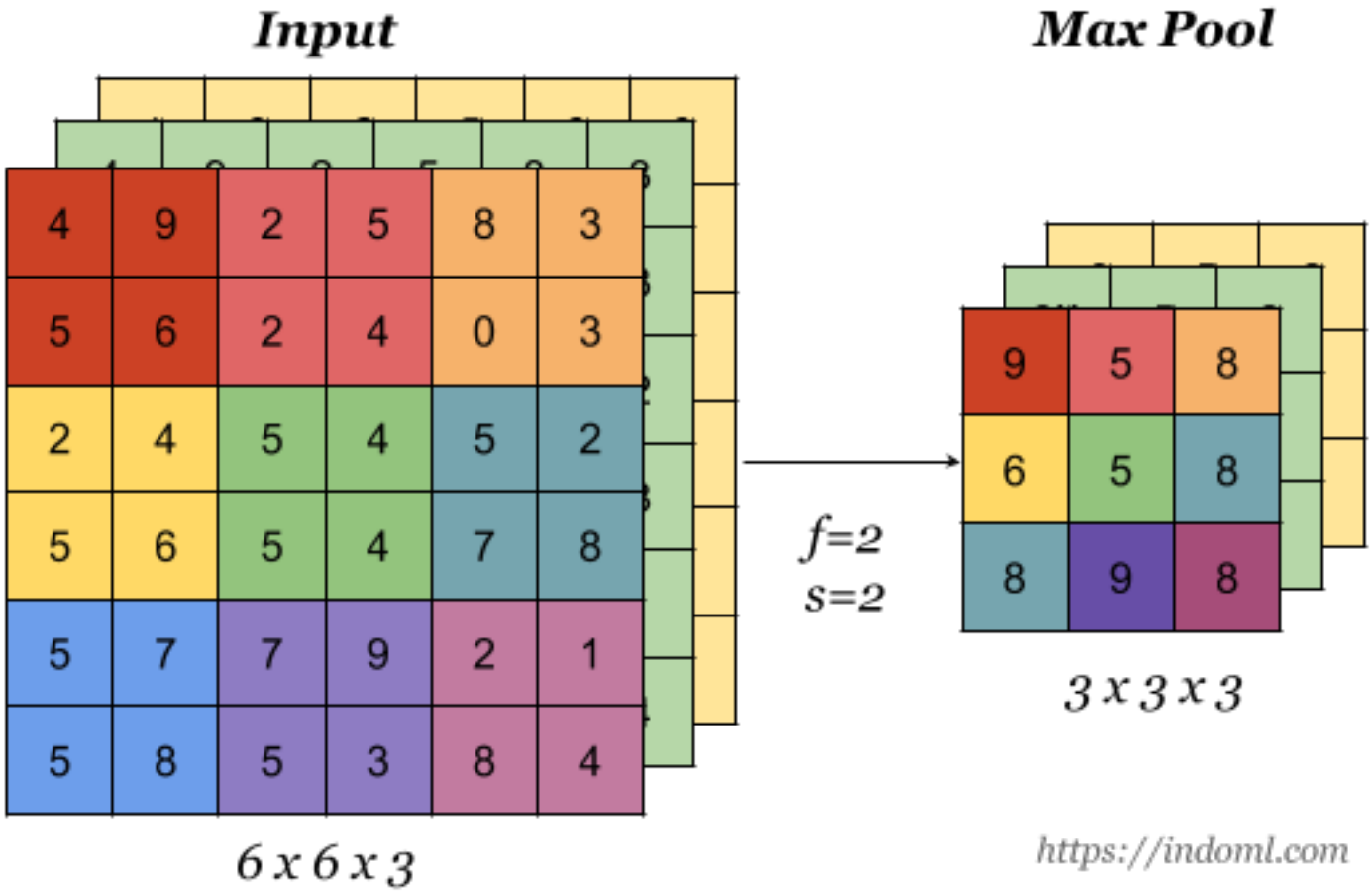

Input
Filter
Result
4 9 2 5 8 3
2 4 0 3
2 4 5 4 5 2
5 6 5 4 7 8
5 7 7 9 2 1
5 8 5 3 8 4
*
1 0 -1
1 0 -1
1 0 -1
=
2 1
Parameters:
Size: f = 3
Stride: s = 2
Padding: p = 0
1 = 2*1 + 5*0 + 3*(-1) +
2*1 + 4*0 + 3*(-1) +
5*1 + 4*0 + 2*(-1)
Dimension: 6 x 6
https://indoml.com

Padding:


Input
Filter
Result
0 0 0 0 0 0 0 0
0 4 9 2 5 8 3 0
0 5 6 2 4 0 3 0
0 2 4 5 4 5 2 0
0 5 6 5 4 7 8 0
0 5 7 7 9 2 1 0
0 5 8 5 3 8 4 0
0 0 0 0 0 0 0 0
*
1 0 -1
1 0 -1
1 0 -1
=
-15
Parameters:
Size: f = 3
Stride: s = 2
Padding: p = 1
= 0*1 + 0*0 + 0*(-1) +
0*1 + 4*0 + 9*(-1) +
0*1 + 9*0 + 6*(-1)
= -15
Dimension: 6 x 6
https://indoml.com

Pooling layer:




Input
Max Pool
4 9 2 5 8 3
5 6 2 4 0 3
2 4 5 4 5 2
5 6 5 4 7 8
5 7 7 9 2 1
5 8 5 3 8 4
6 x 6 x 3
f=2
s=2
9 5 8
6 5 8
8 9 8
3 x 3 x 3
https://indoml.com

Softmax layer:

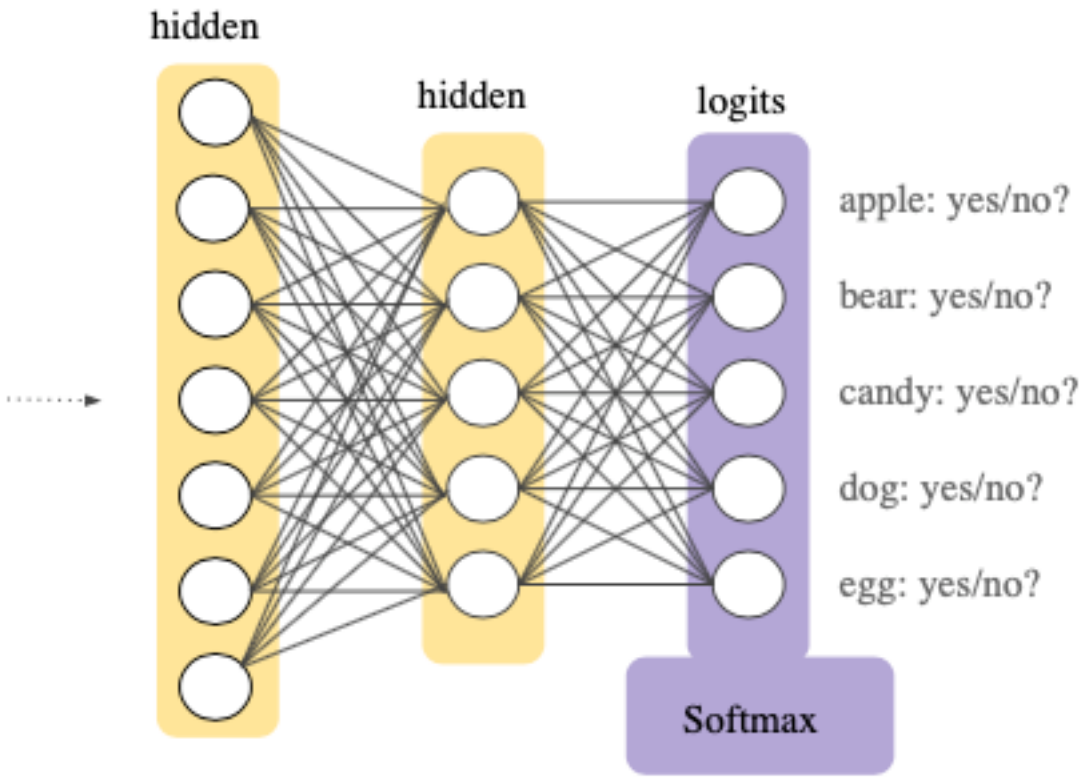
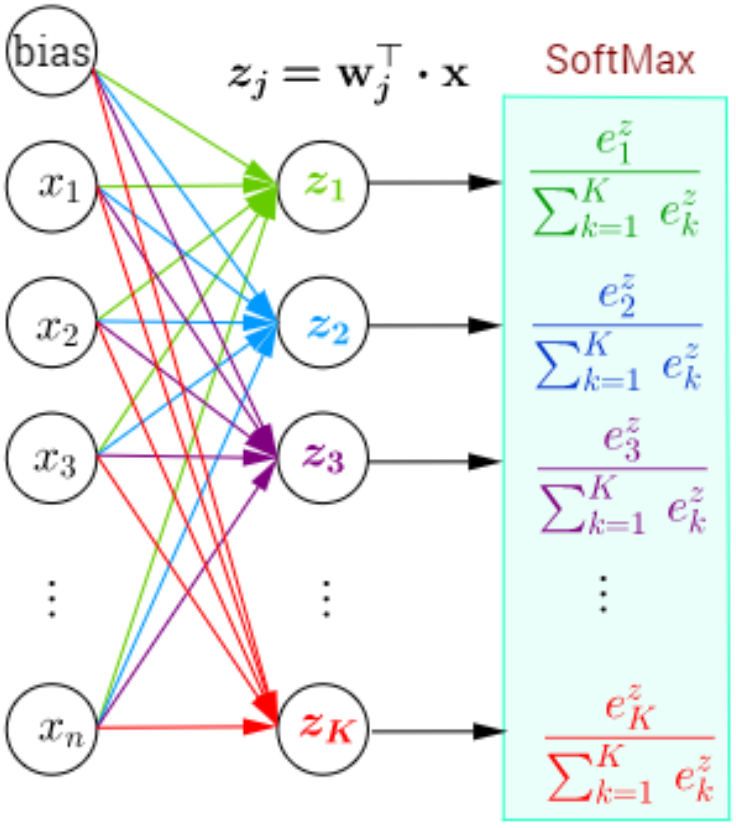


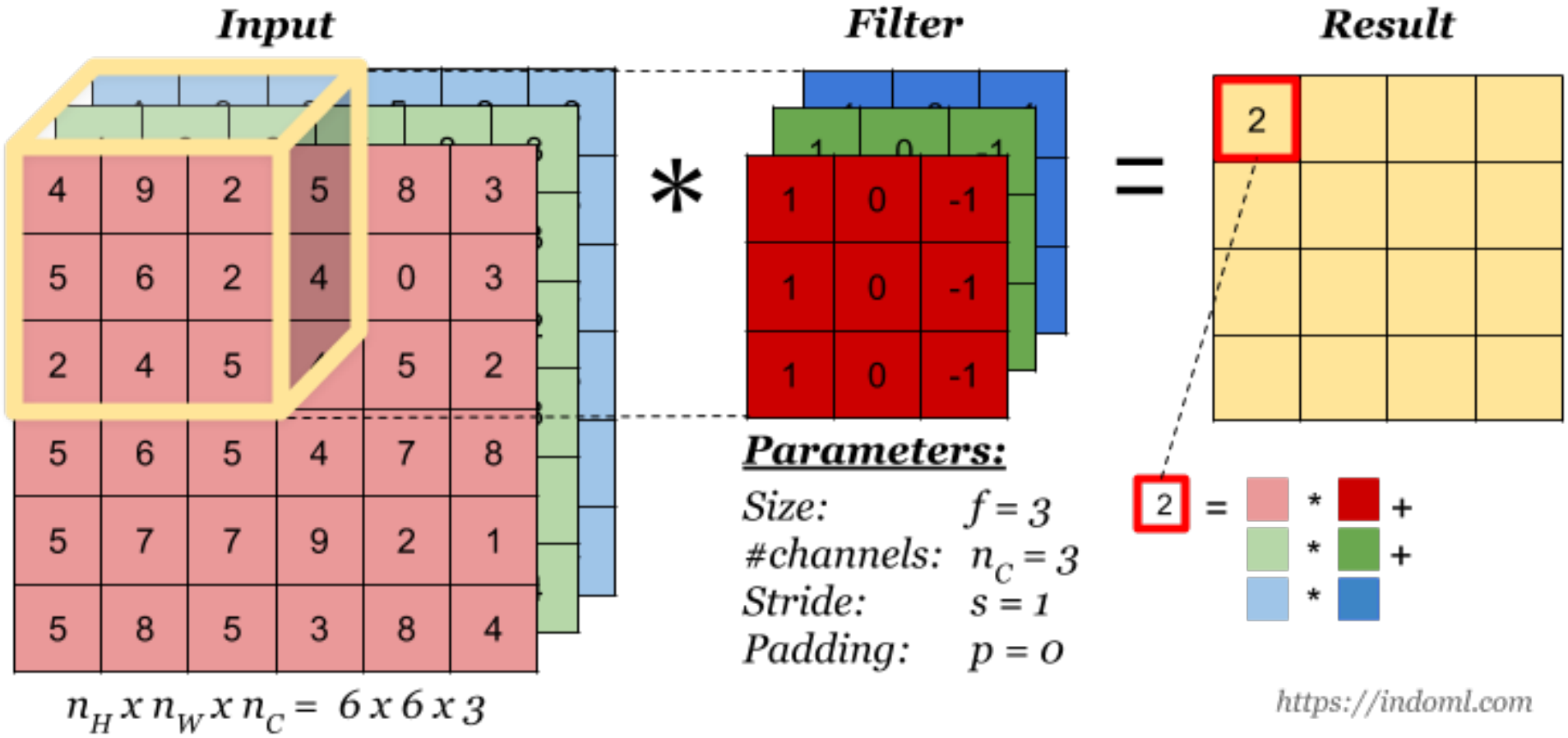
Image source: https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

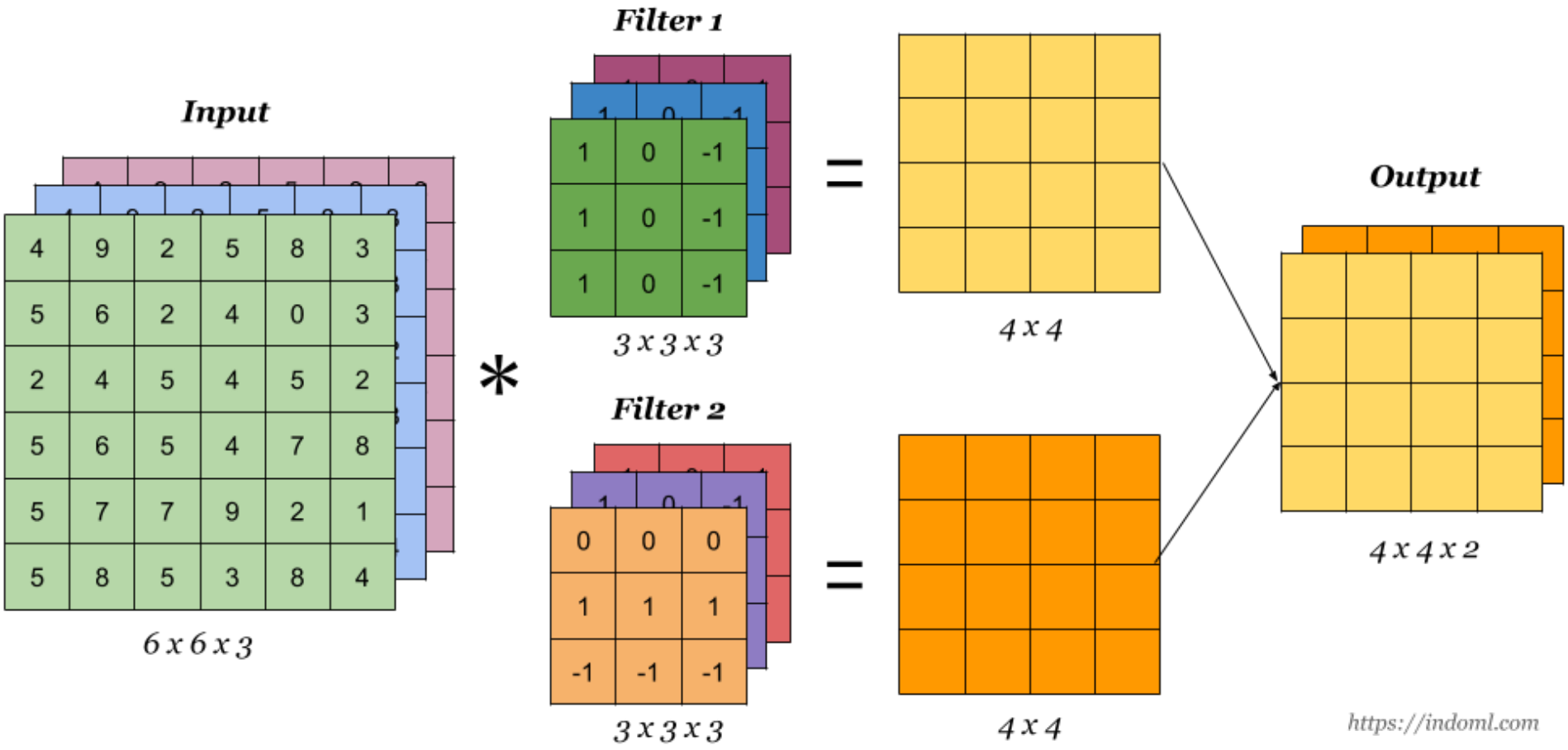


Source: https://stats.stackexchange.com/questions/273465/neural-network-softmax-activation

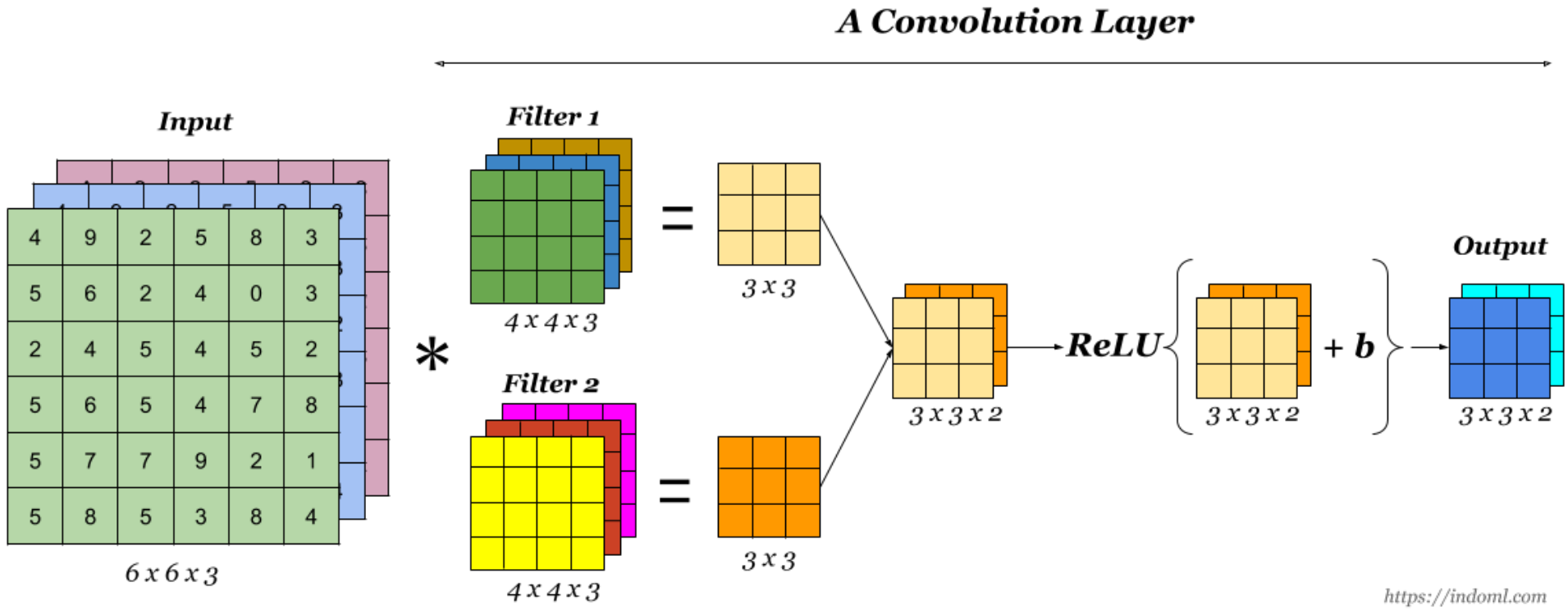
CNN on Volume (Tensors)



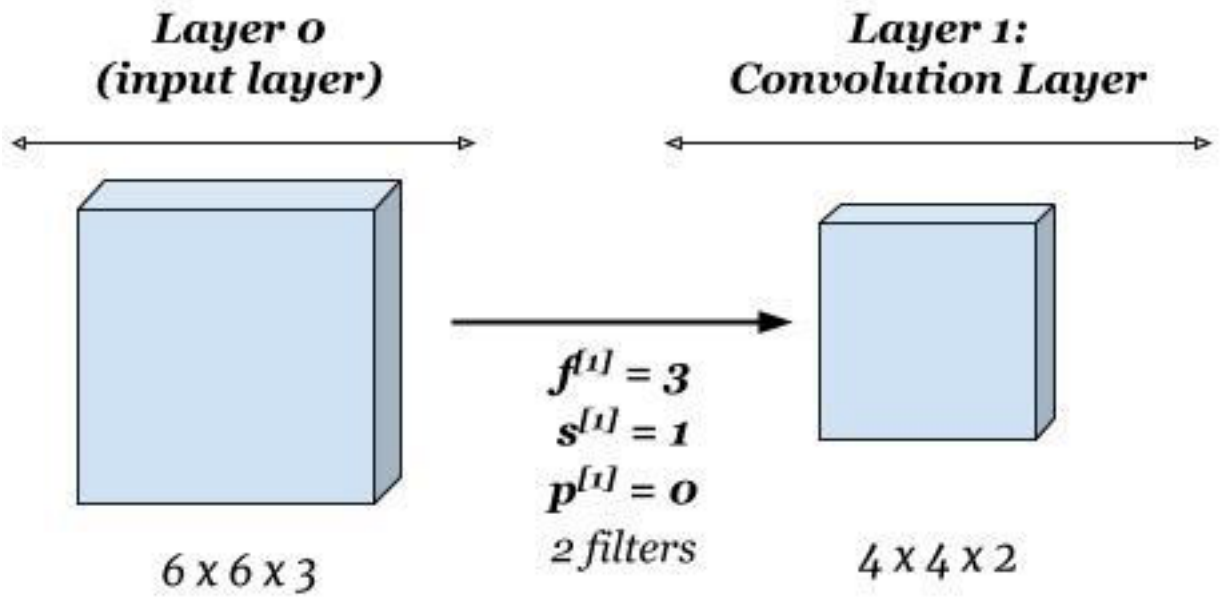
Multiple Filters



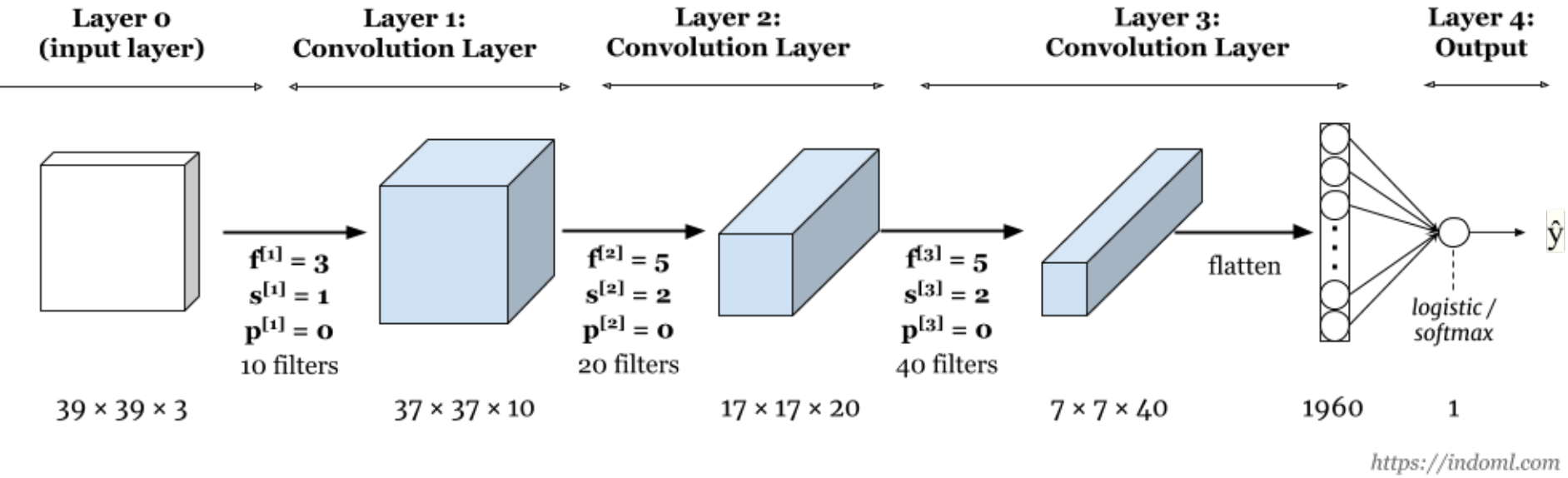
One Layer Representation



Shorthand Representation

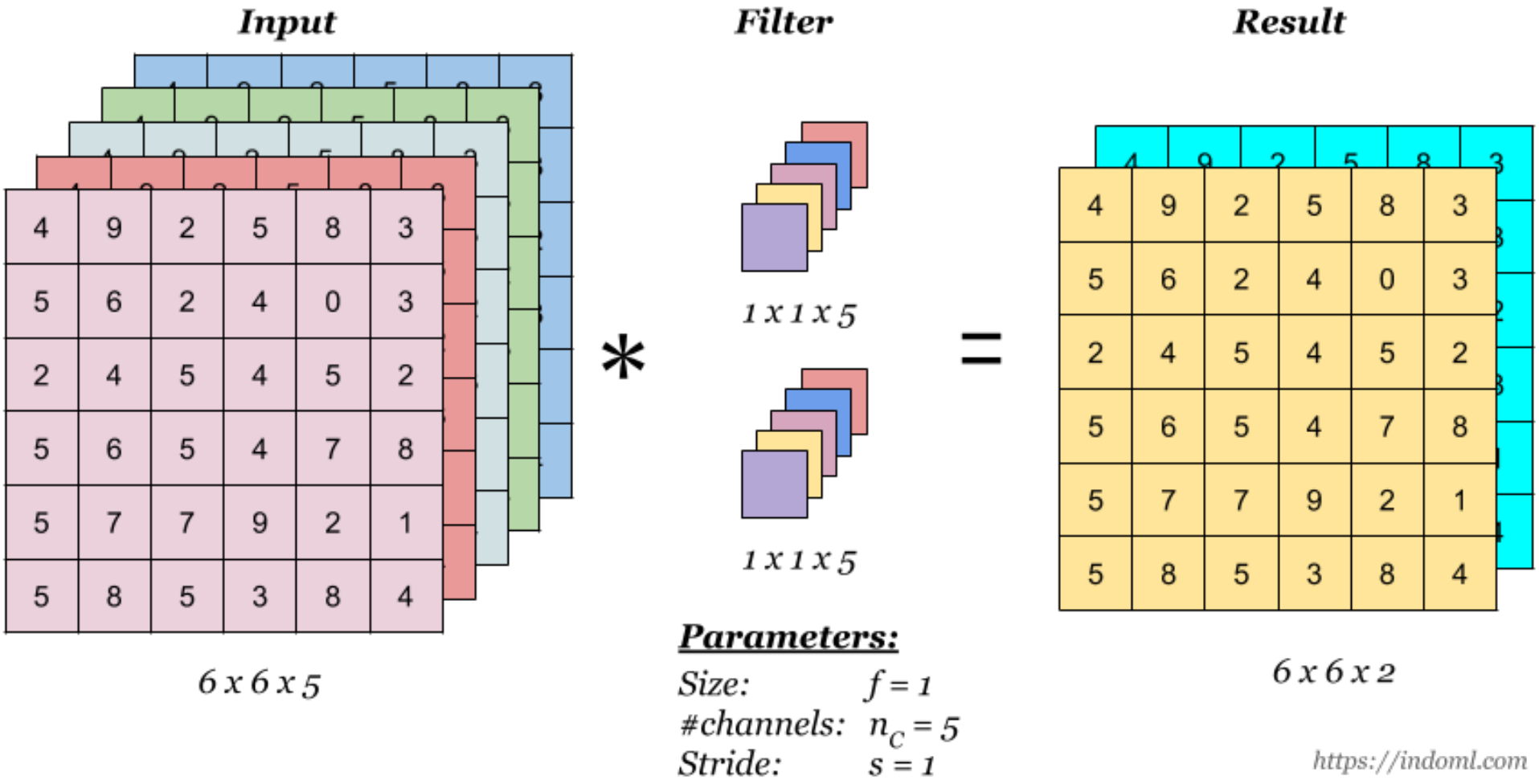


Sample Network Structure

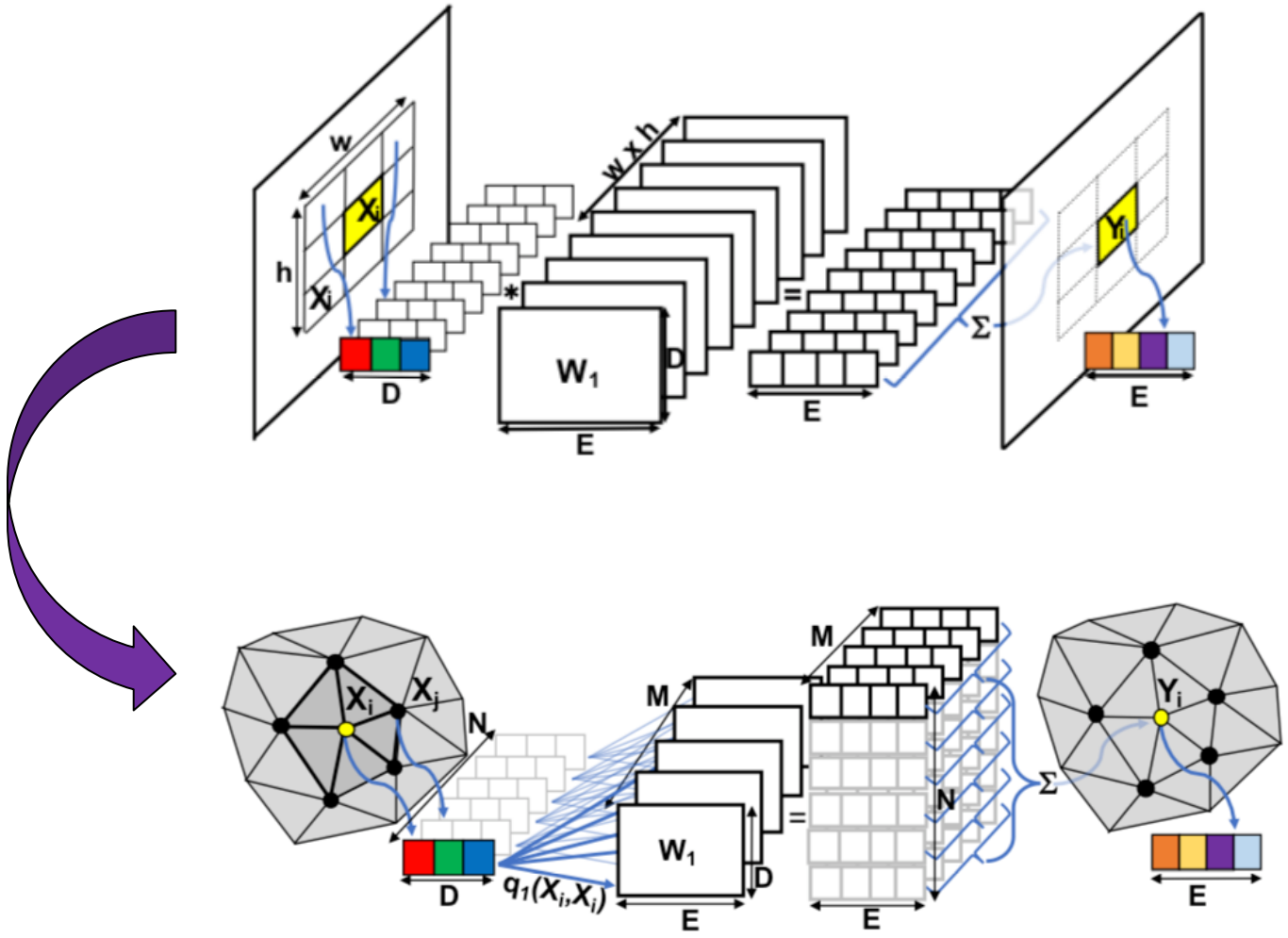


https://indoml.com

1*1 Convolutions



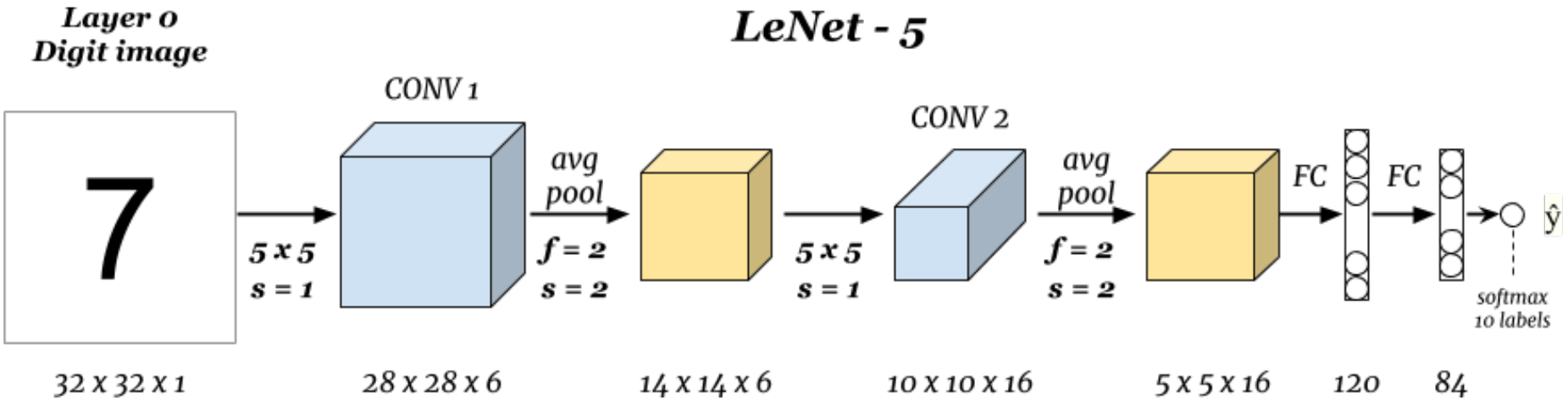
Generalize the CNN



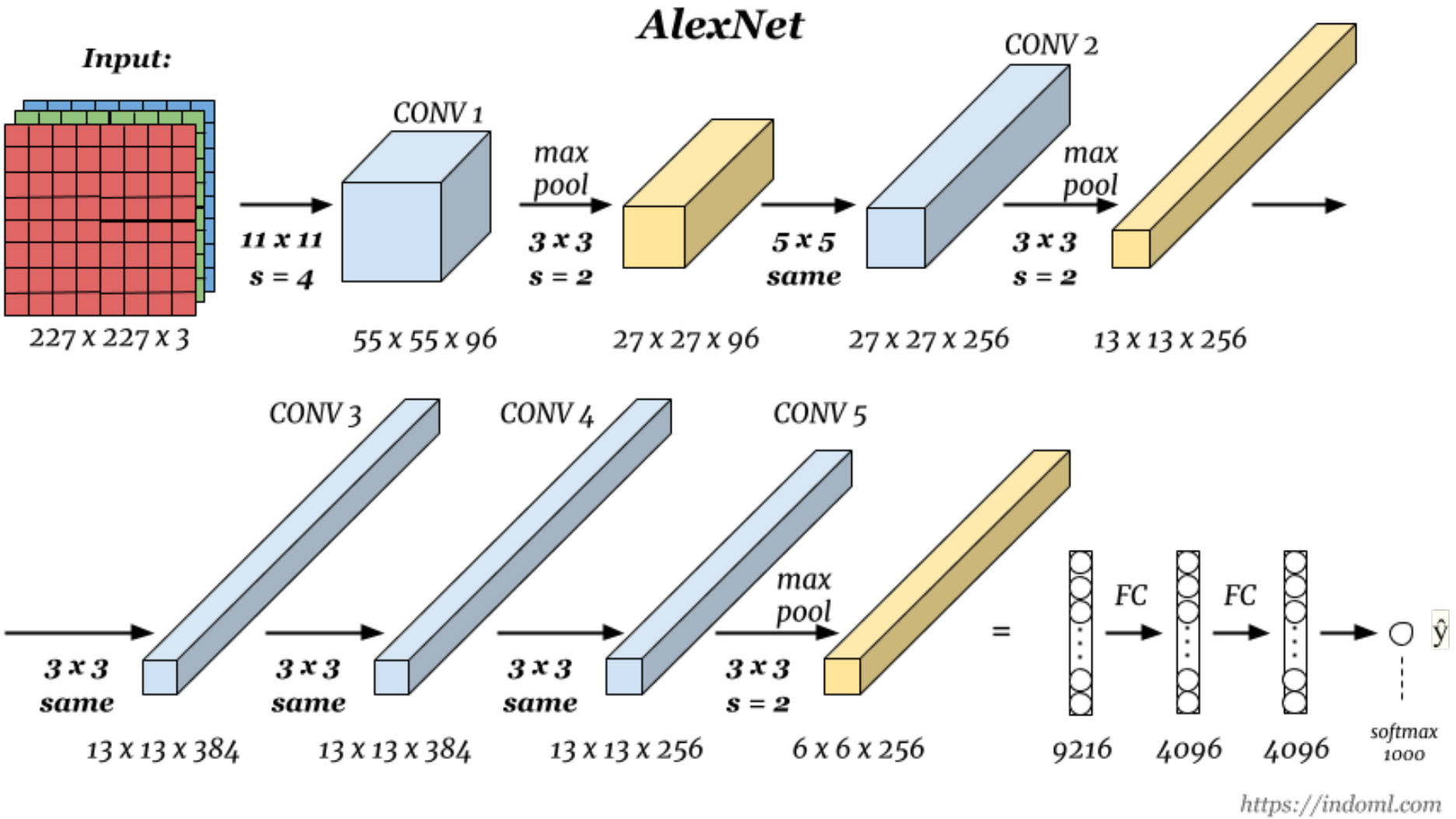
Classic Networks

- LeNet
- AlexNet
- VGG Net
- ResNet
- GooLeNet

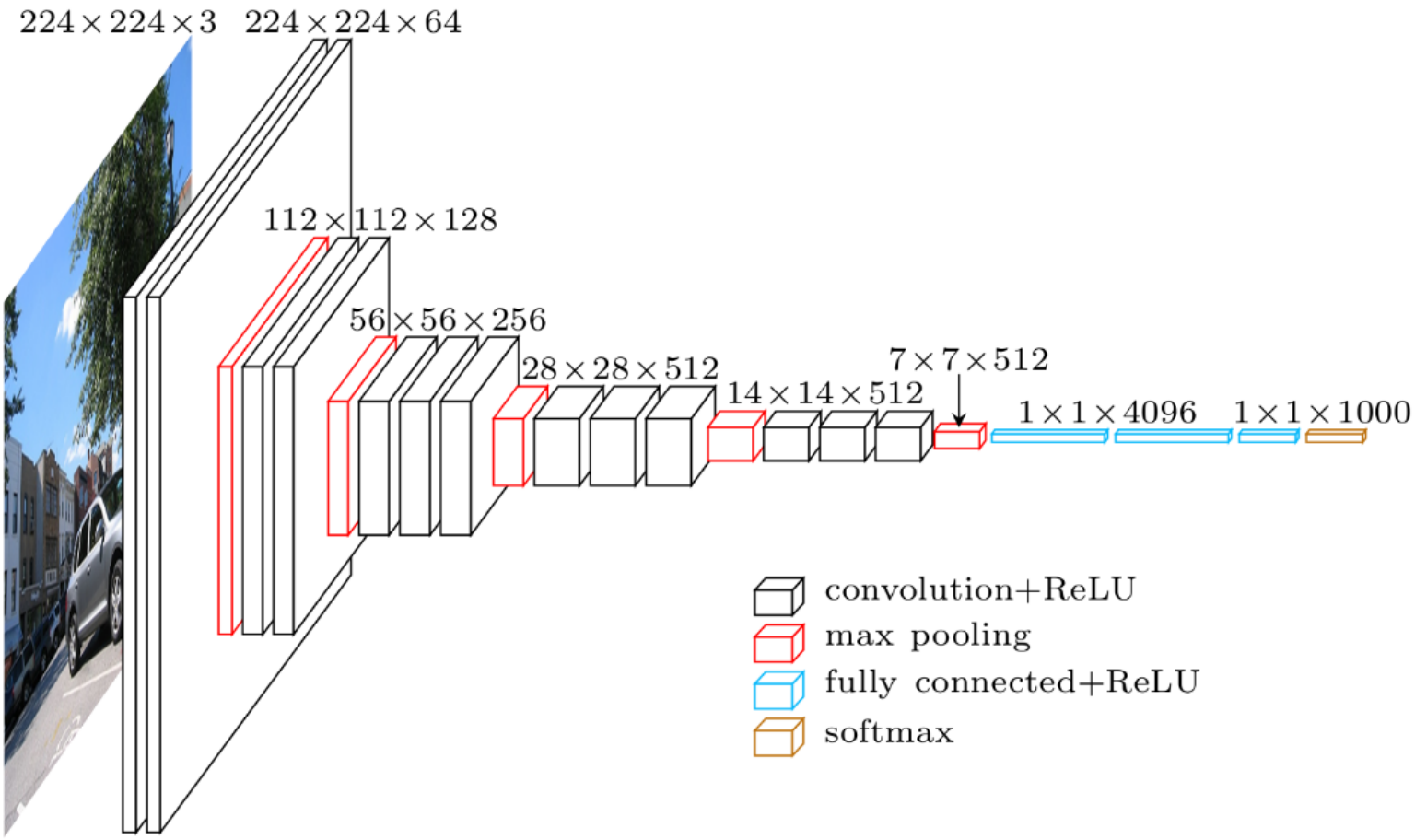
LeNet


Layer 0
Digit image
LeNet - 5
7
CONV 1
5 x 5
s = 1
avg
pool
f = 2
s = 2
CONV 2
5 x 5
s = 1
avg
pool
f = 2
s = 2
FC
FC
softmax
10 labels
ŷ
32 x 32 x 1
28 x 28 x 6
14 x 14 x 6
10 x 10 x 16
5 x 5 x 16
120
84

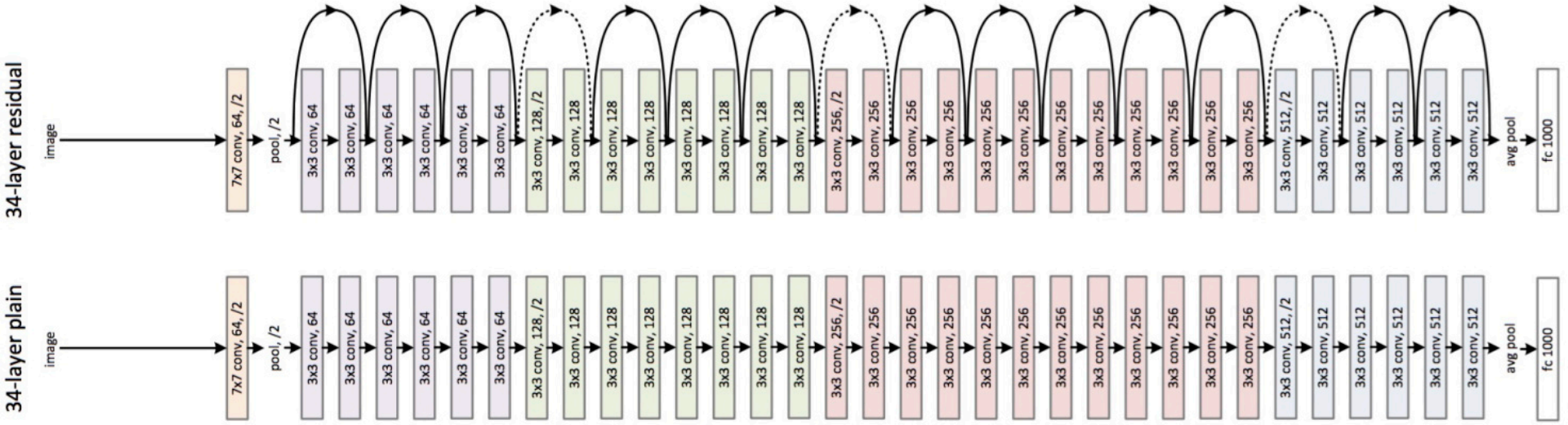
AlexNet


AlexNet
Input:
227 x 227 x 3
11 x 11
s = 4
CONV 1
55 x 55 x 96
max
pool
3 x 3
s = 2
27 x 27 x 96
5 x 5
same
CONV 2
27 x 27 x 256
max
pool
3 x 3
s = 2
13 x 13 x 256
3 x 3
same
CONV 3
13 x 13 x 384
3 x 3
same
CONV 4
13 x 13 x 384
3 x 3
same
CONV 5
13 x 13 x 256
max
pool
3 x 3
s = 2
6 x 6 x 256
=
9216
FC
4096
FC
4096
softmax
1000
ŷ
https://indoml.com

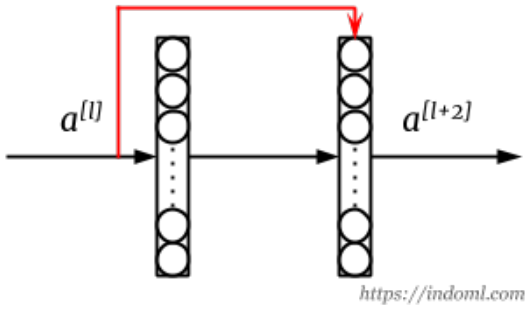
VGG-16


224 × 224 × 3
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
convolution+ReLU
max pooling
fully connected+ReLU
softmax

ResNet

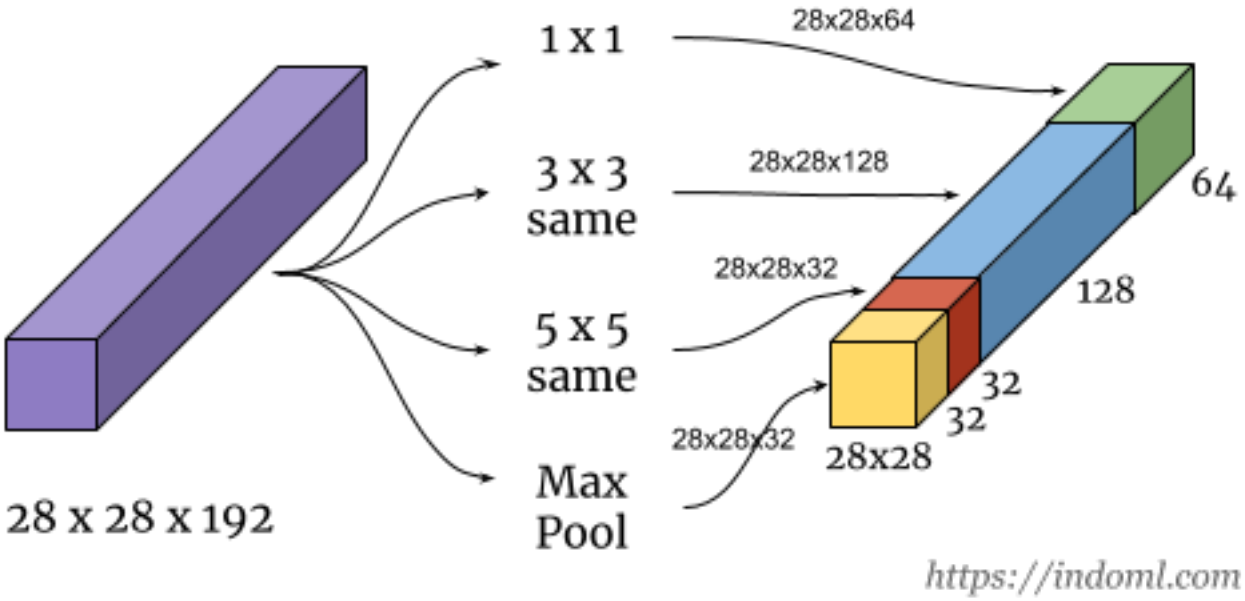


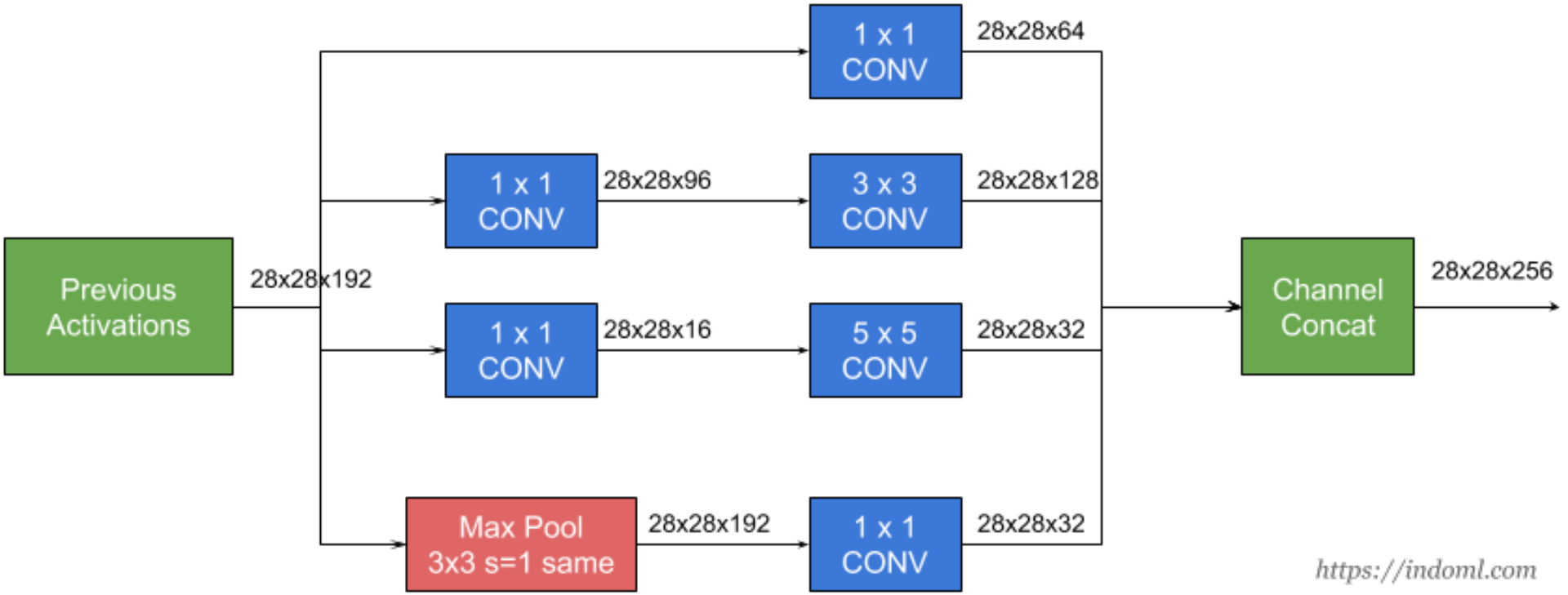
Residual Module



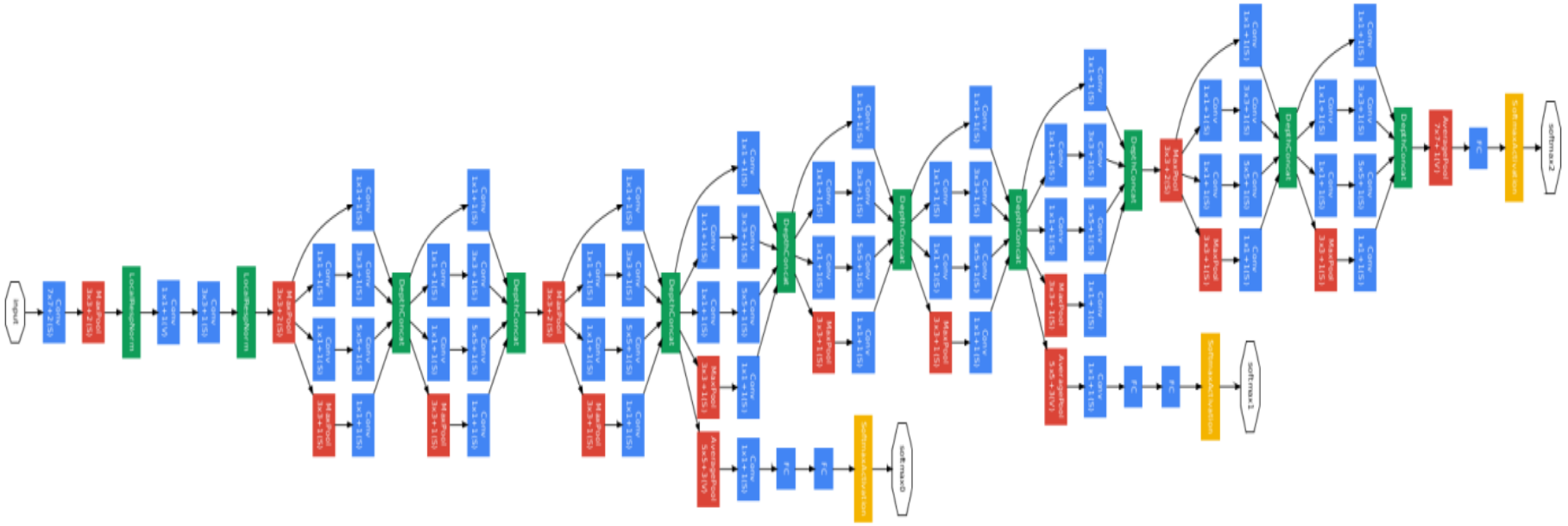
GoogLeNet

Inception Module




Previous Activations
28x28x192
1 x 1 CONV
28x28x64
1 x 1 CONV
28x28x96
3 x 3 CONV
28x28x128
1 x 1 CONV
28x28x16
5 x 5 CONV
28x28x32
Max Pool 3x3 s=1 same
28x28x192
1 x 1 CONV
28x28x32
Channel Concat
28x28x256
https://indoml.com

GoogLeNet:



Summary of Networks

Year	CNN	Developed by	Place	Top-5 error rate	No. of parameters
1998	LeNet(8)	Yann LeCun et al			60 thousand
2012	AlexNet(7)	Alex Krizhevsky, Geoffrey Hinton, Ilya Sutskever	1st	15.3%	60 million
2014	GoogLeNet(19)	Google	1st	6.67%	4 million
2014	VGG Net(16)	Simonyan, Zisserman	2nd	7.3%	138 million
2015	ResNet(152)	Kaiming He	1st	3.6%	

References

- CNN: ML Lecture 10: Convolutional Neural Network by Hongyi Li (youtube)
- Chapter 20 from Berkeley Artificial Intelligence: A Modern Approach (link: http://aima.eecs.berkeley.edu/)
- Module 2: Convolutional Neural Networks from course notes of Stanford CS231n (link: http://cs231n.github.io/convolutional-networks/)
- Full connection blog (link: https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-4-full-connection)
- http://aima.eecs.berkeley.edu/
- http://cs231n.github.io/convolutional-networks/