

Machine Learning, Spring 2020

Deep Learning Part III

Instructor: Prof. Yi Fang

yfang@nyu.edu

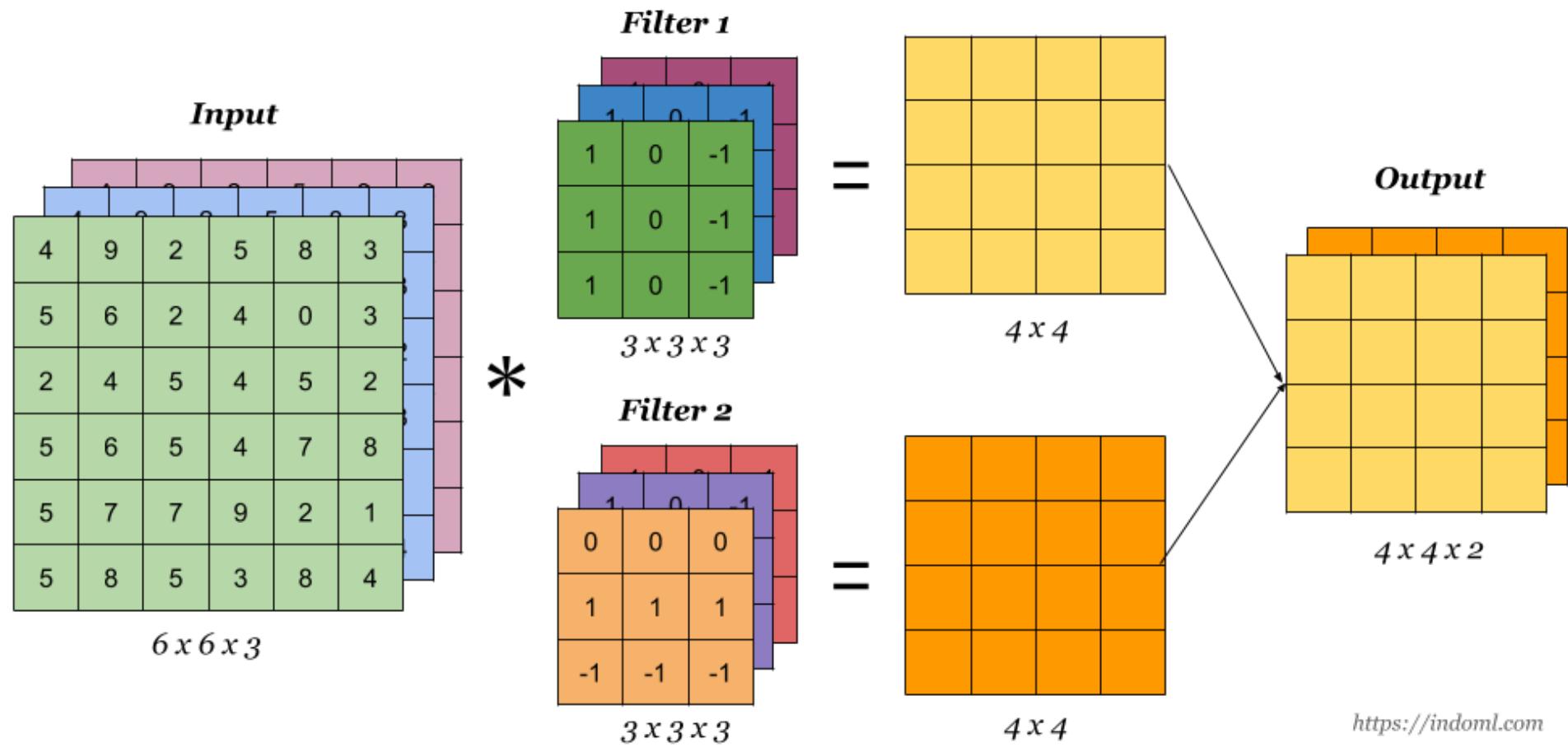
Reading Assignment: Chapter 10,11,12 & 13

Python tutorial: <http://learnpython.org/>

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

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

Convolution Neural Network



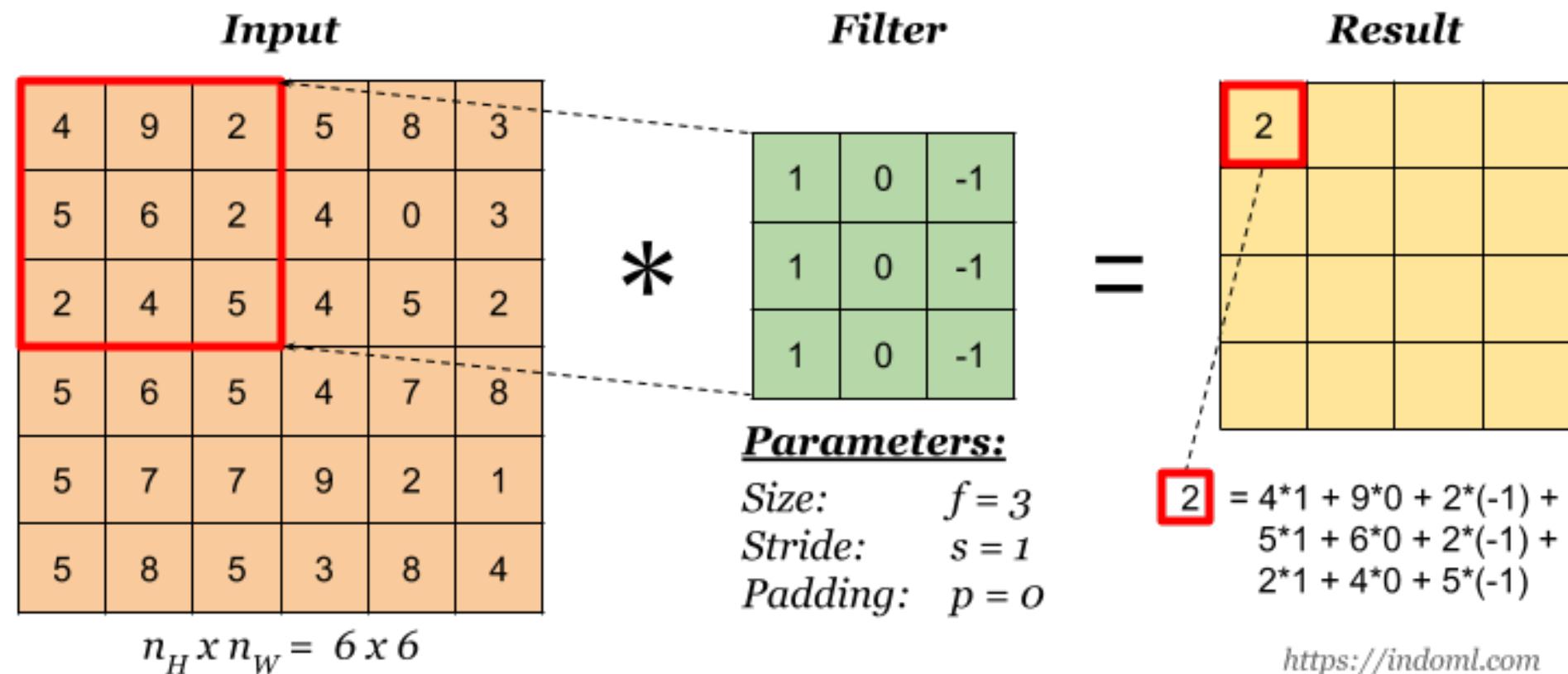
<https://indoml.com>

CNN Architectures

- Basics of CNN
- Classic Networks

CNN Basics

Basic Convolution Operation



Input

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

$$n_H \times n_W = 6 \times 6$$

Filter

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

*

Result

2	6		

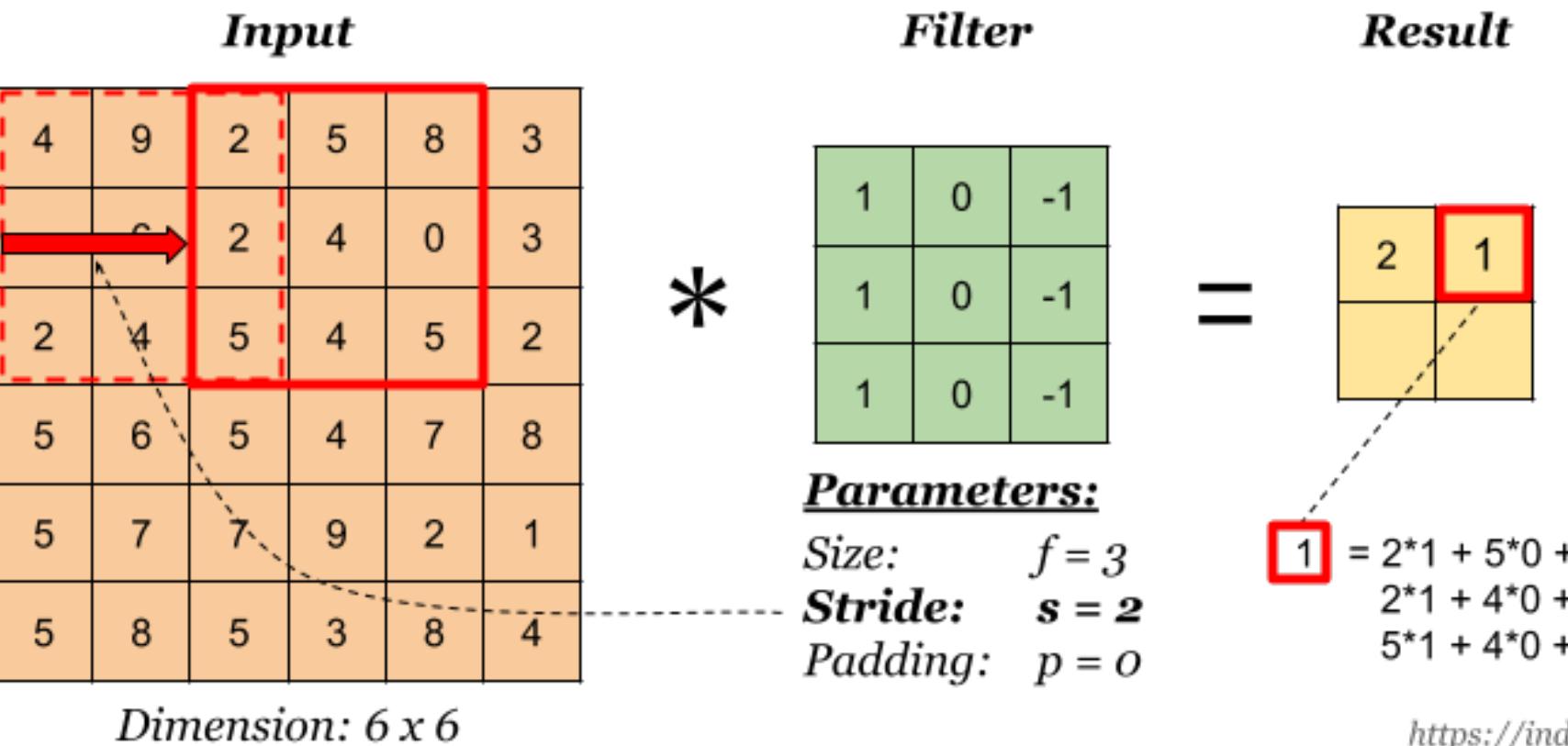
=

Parameters:Size: $f = 3$ Stride: $s = 1$ Padding: $p = 0$

$\boxed{6} = 9*1 + 2*0 + 5*(-1) +$
 $6*1 + 2*0 + 4*(-1) +$
 $4*1 + 5*0 + 4*(-1)$

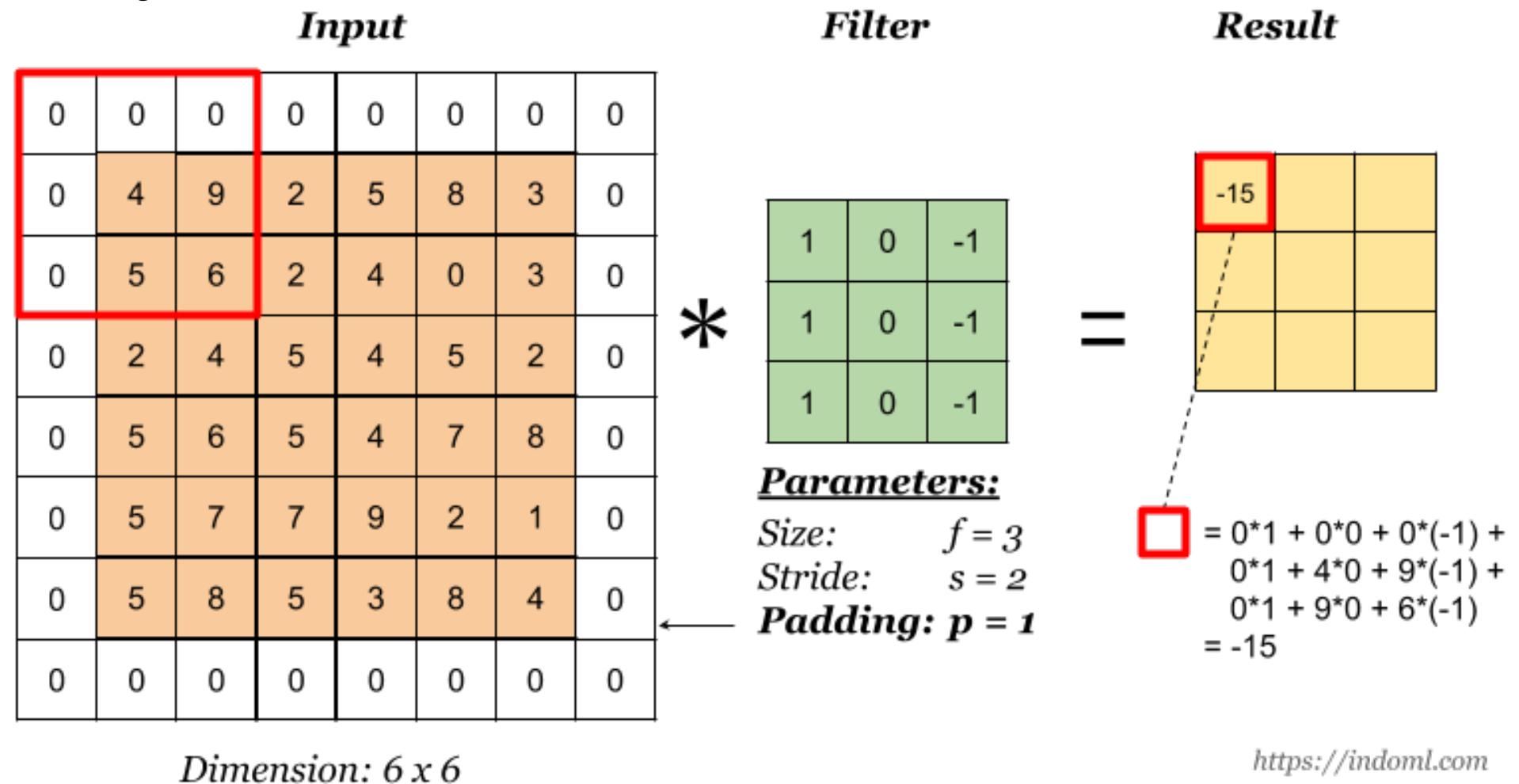
<https://indoml.com>

Stride:

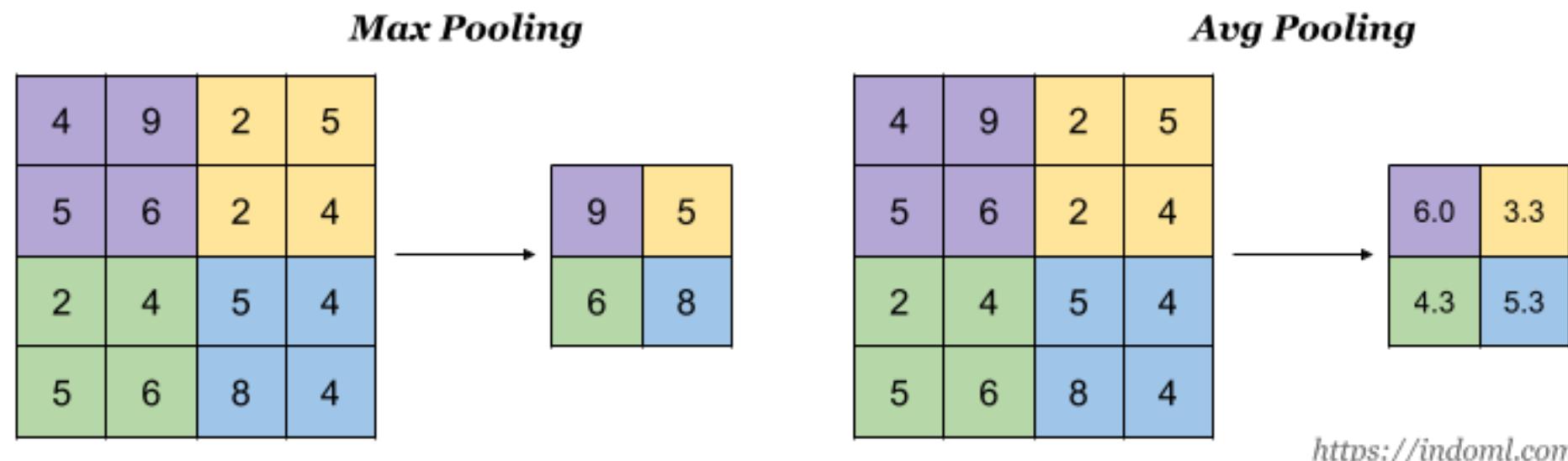


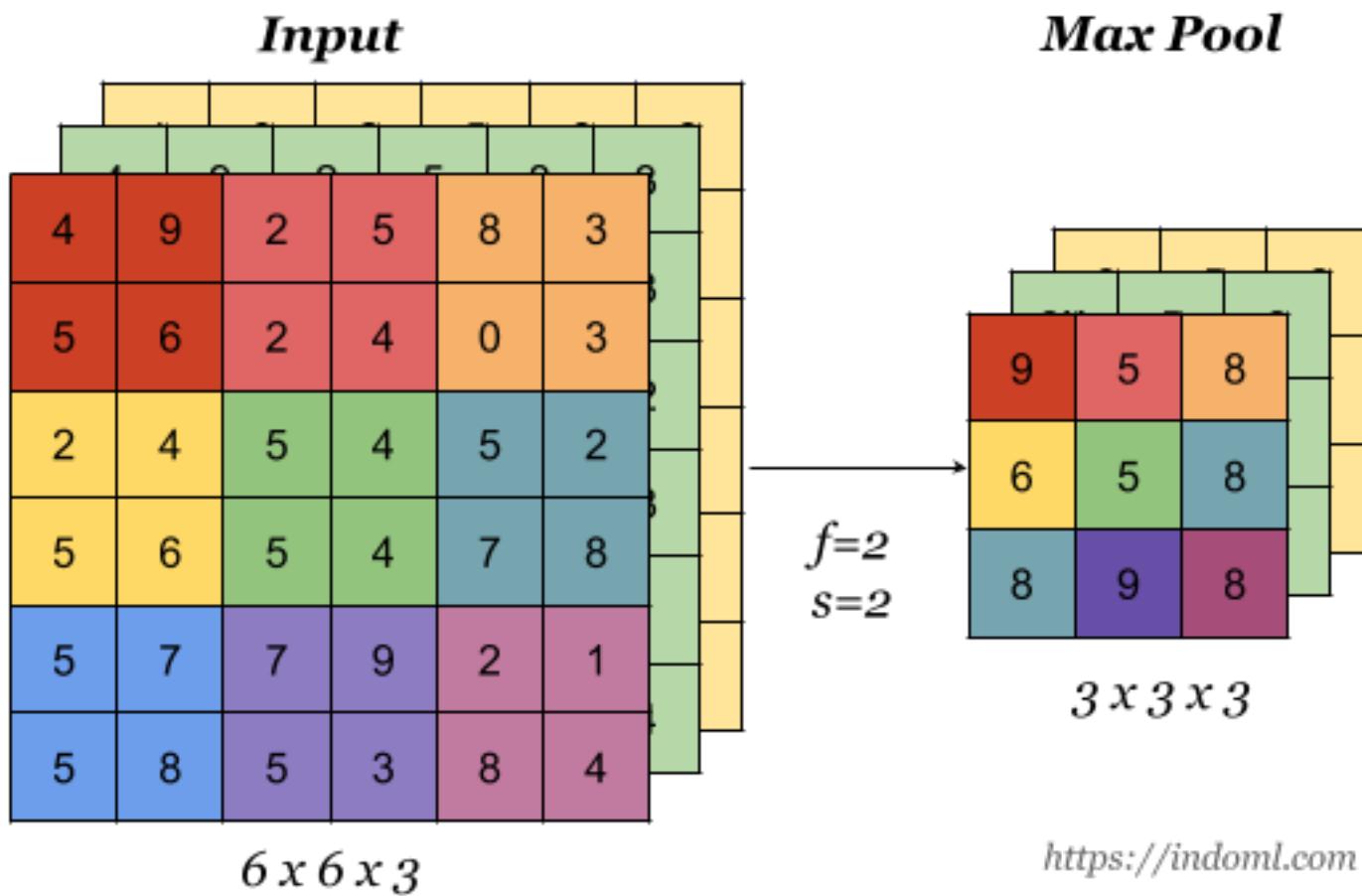
<https://indoml.com>

Padding:



Pooling layer:





Softmax layer:

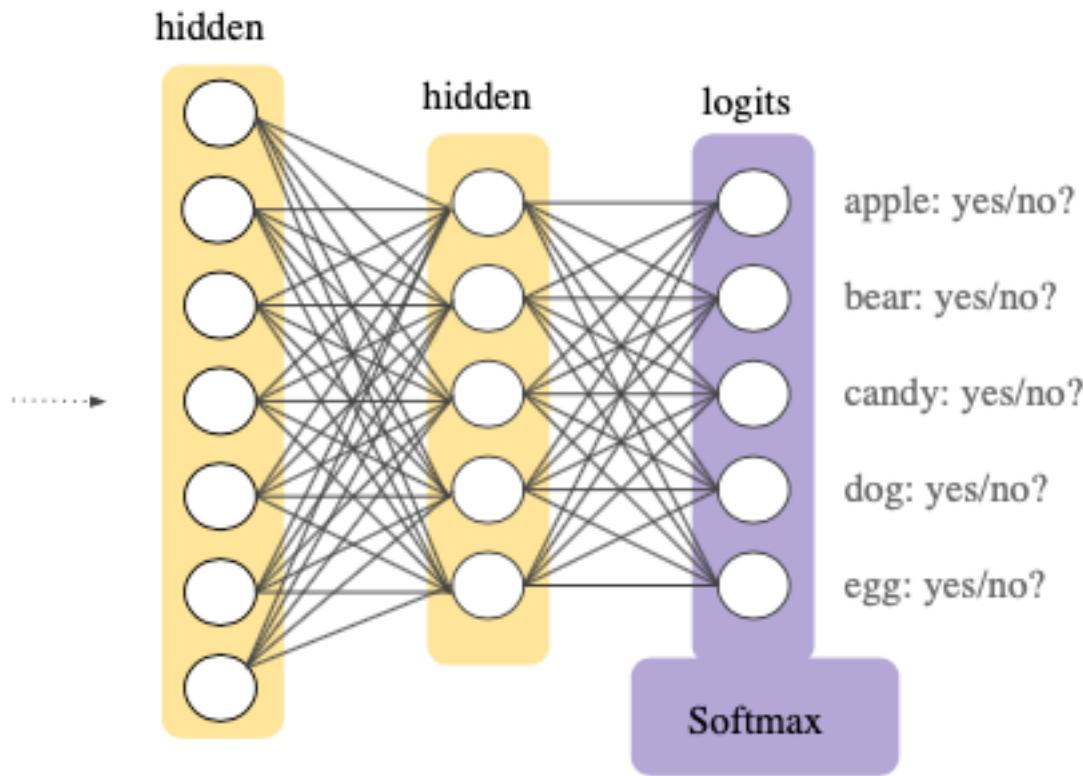
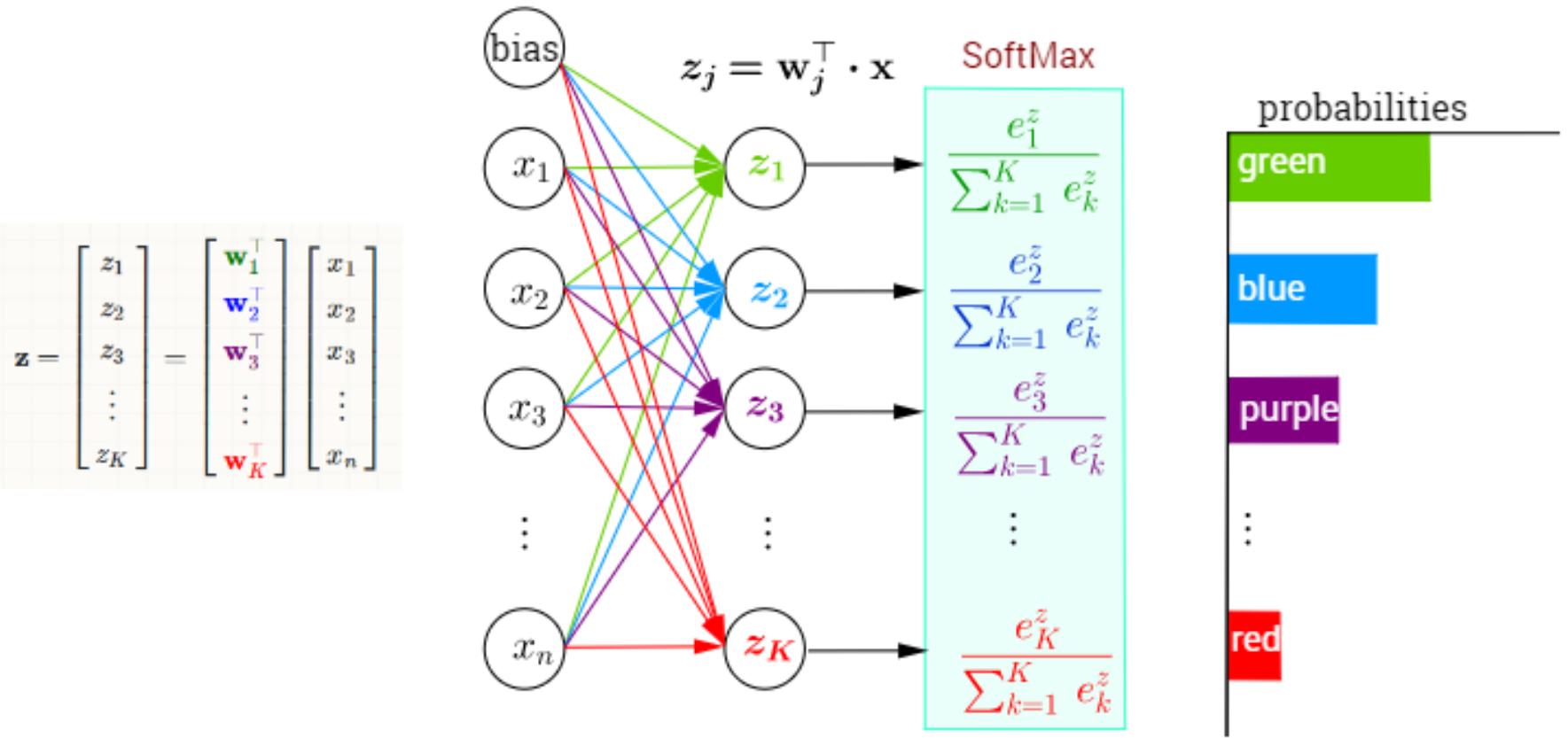
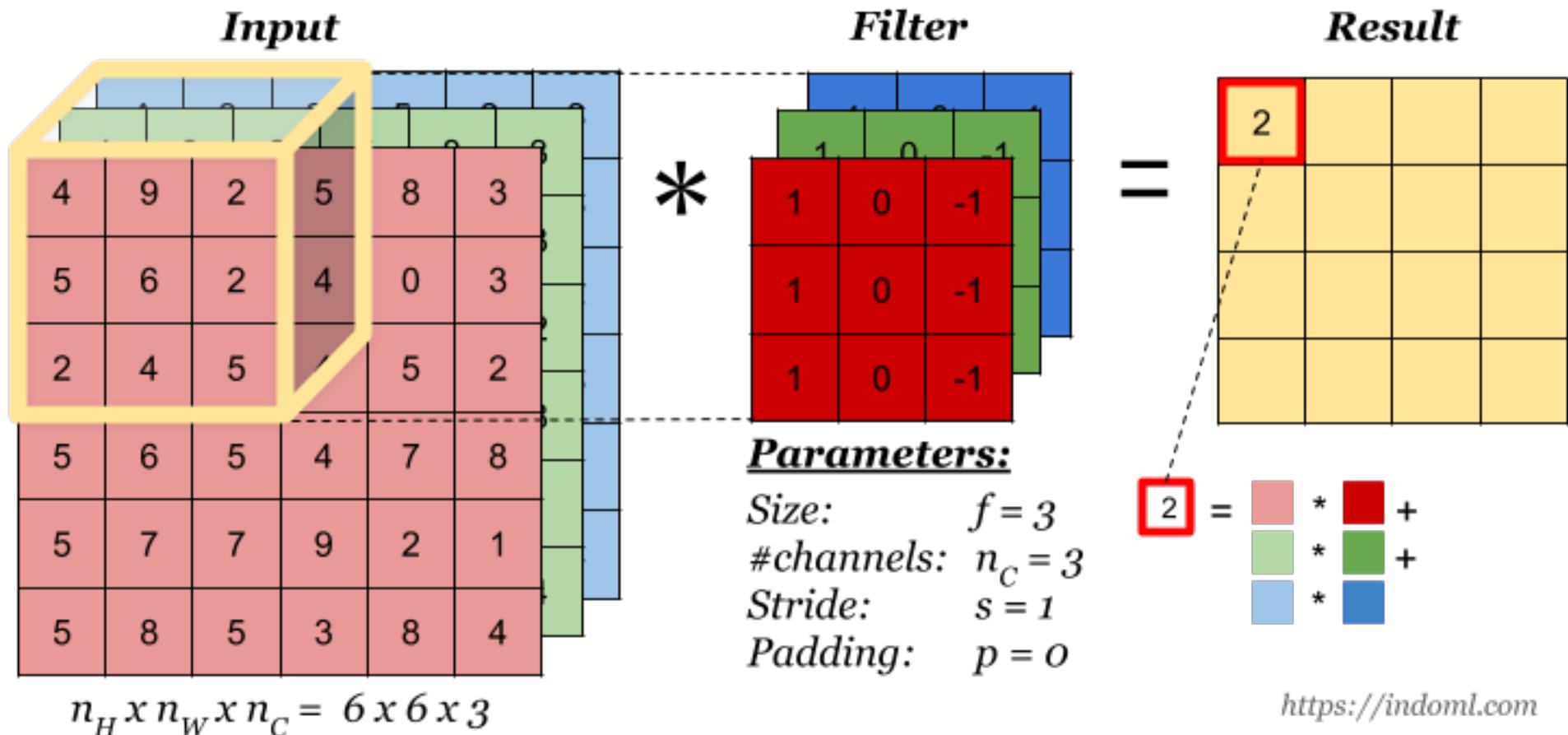


Image source: <https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax>



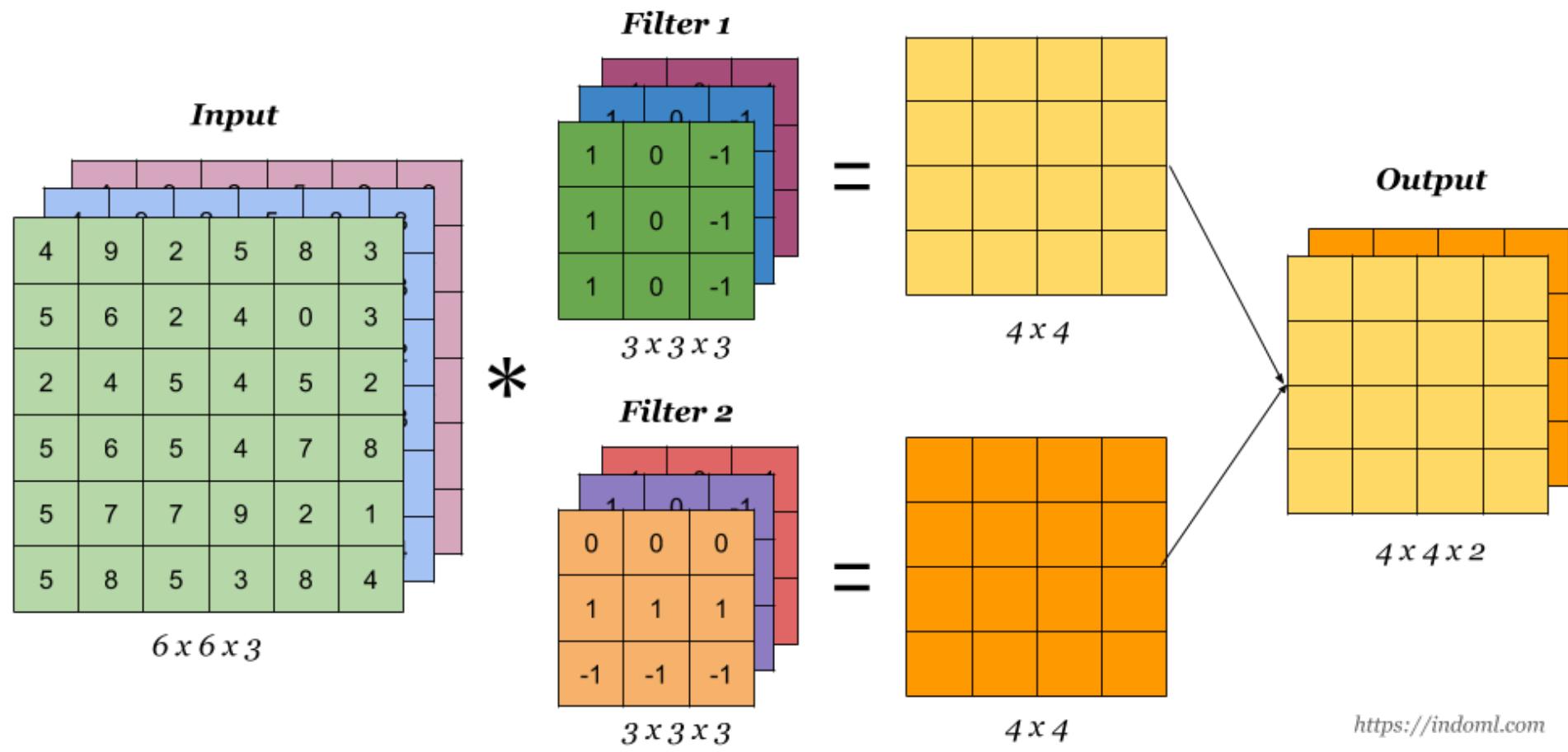
Source: <https://stats.stackexchange.com/questions/273465/neural-network-softmax-activation>

CNN on Volume (Tensors)



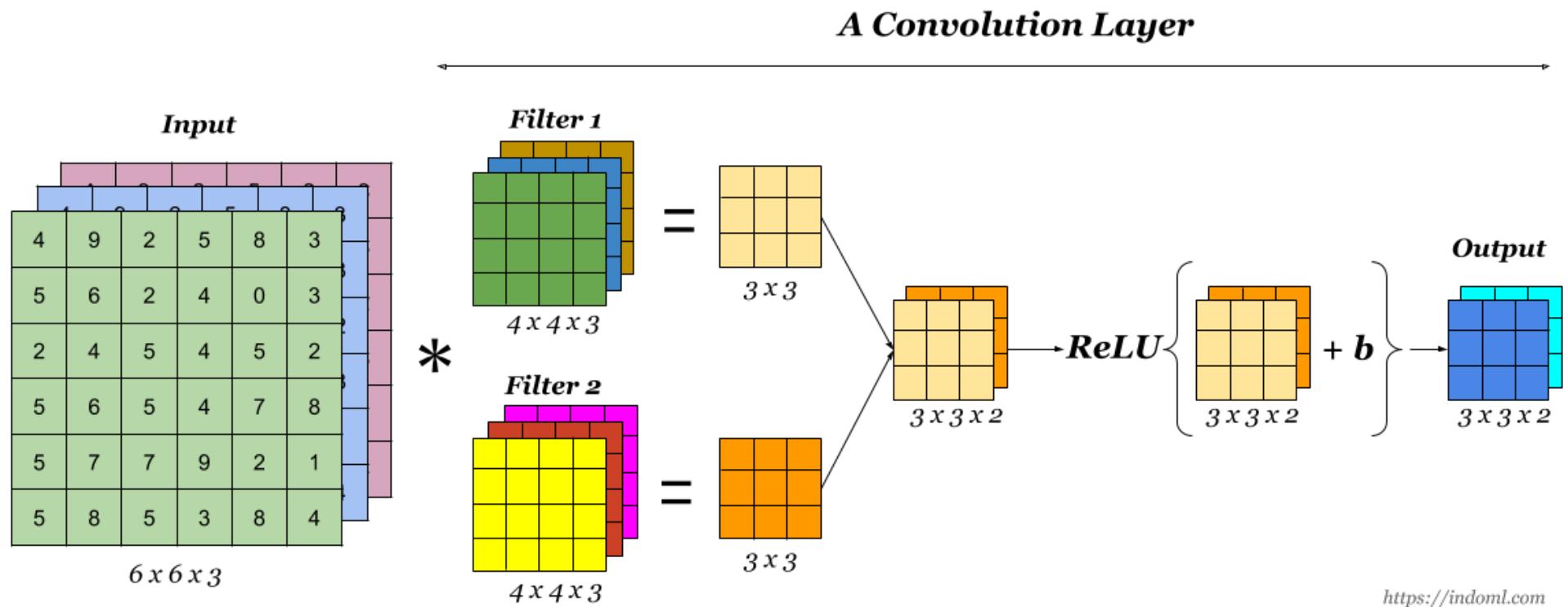
<https://indoml.com>

Multiple Filters

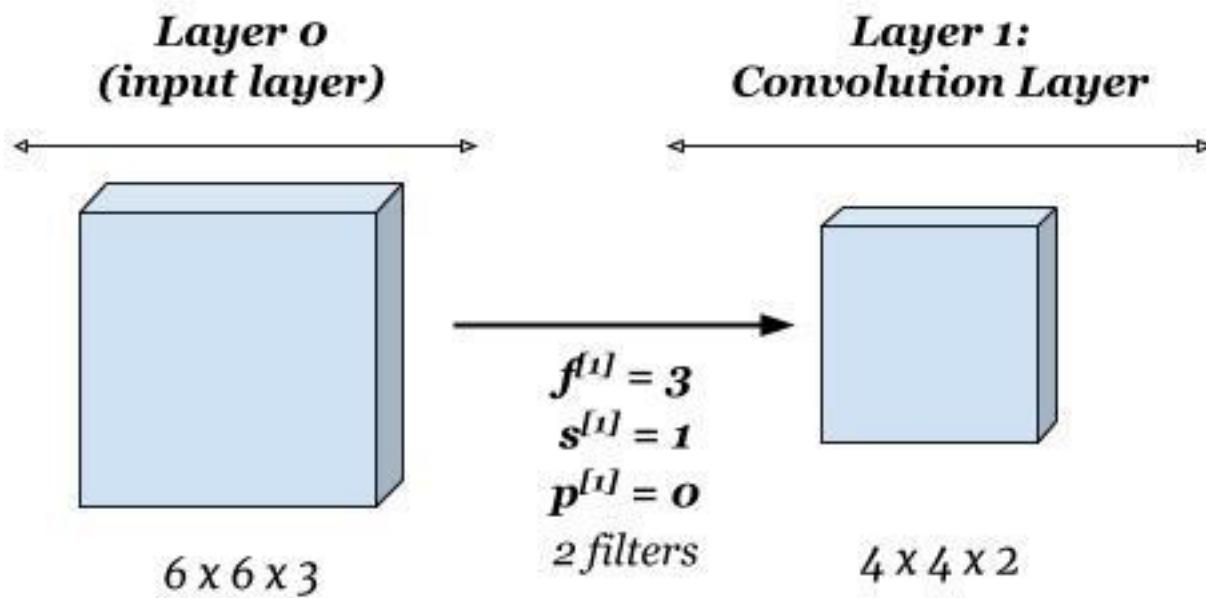


<https://indoml.com>

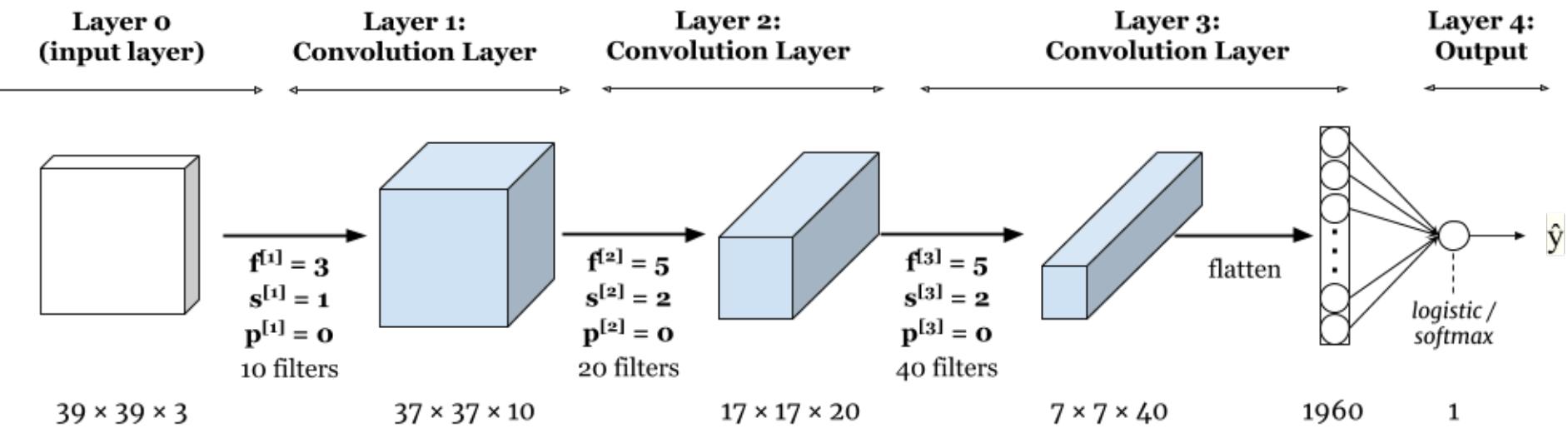
One Layer Representation



Shorthand Representation

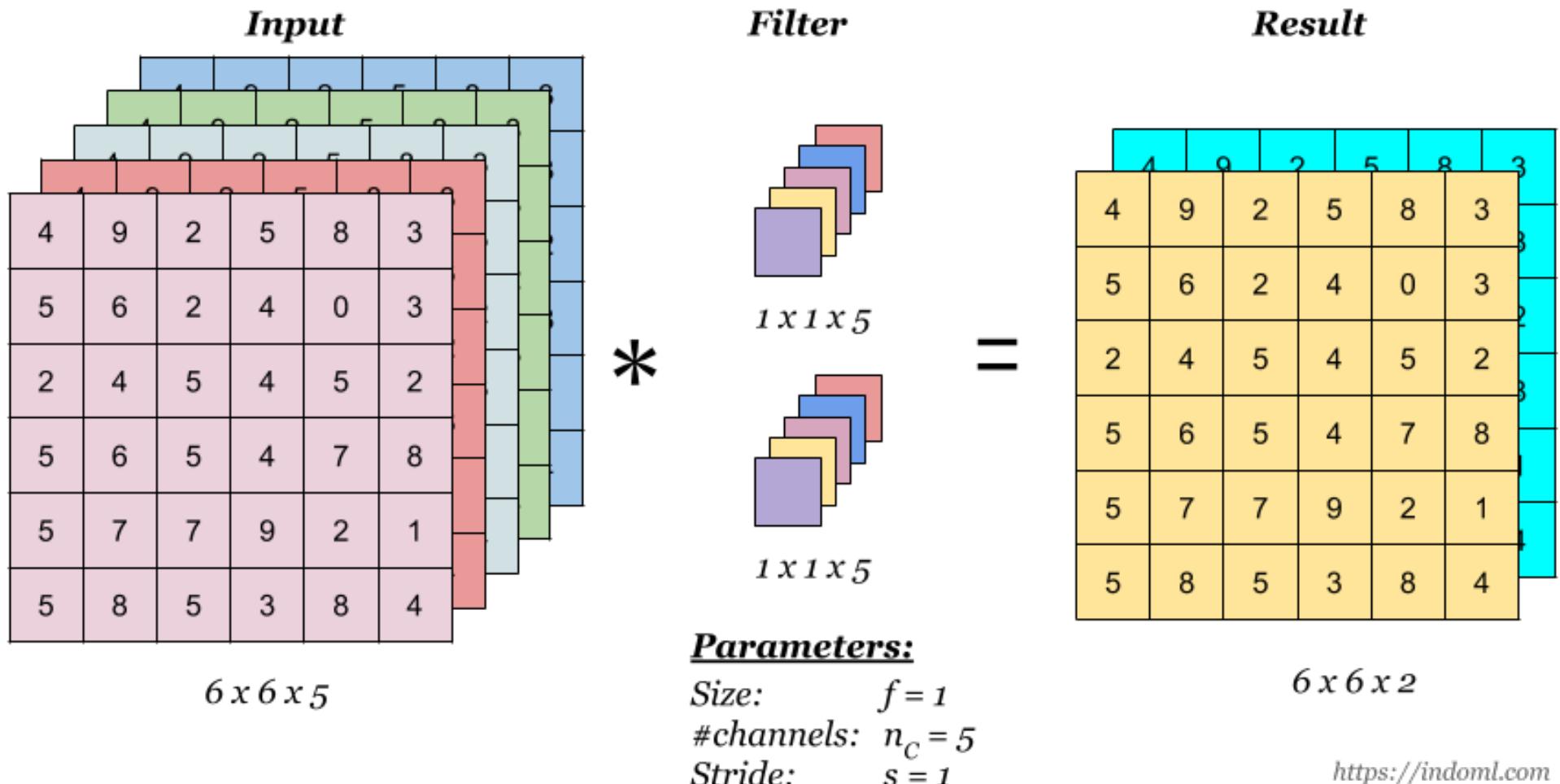


Sample Network Structure



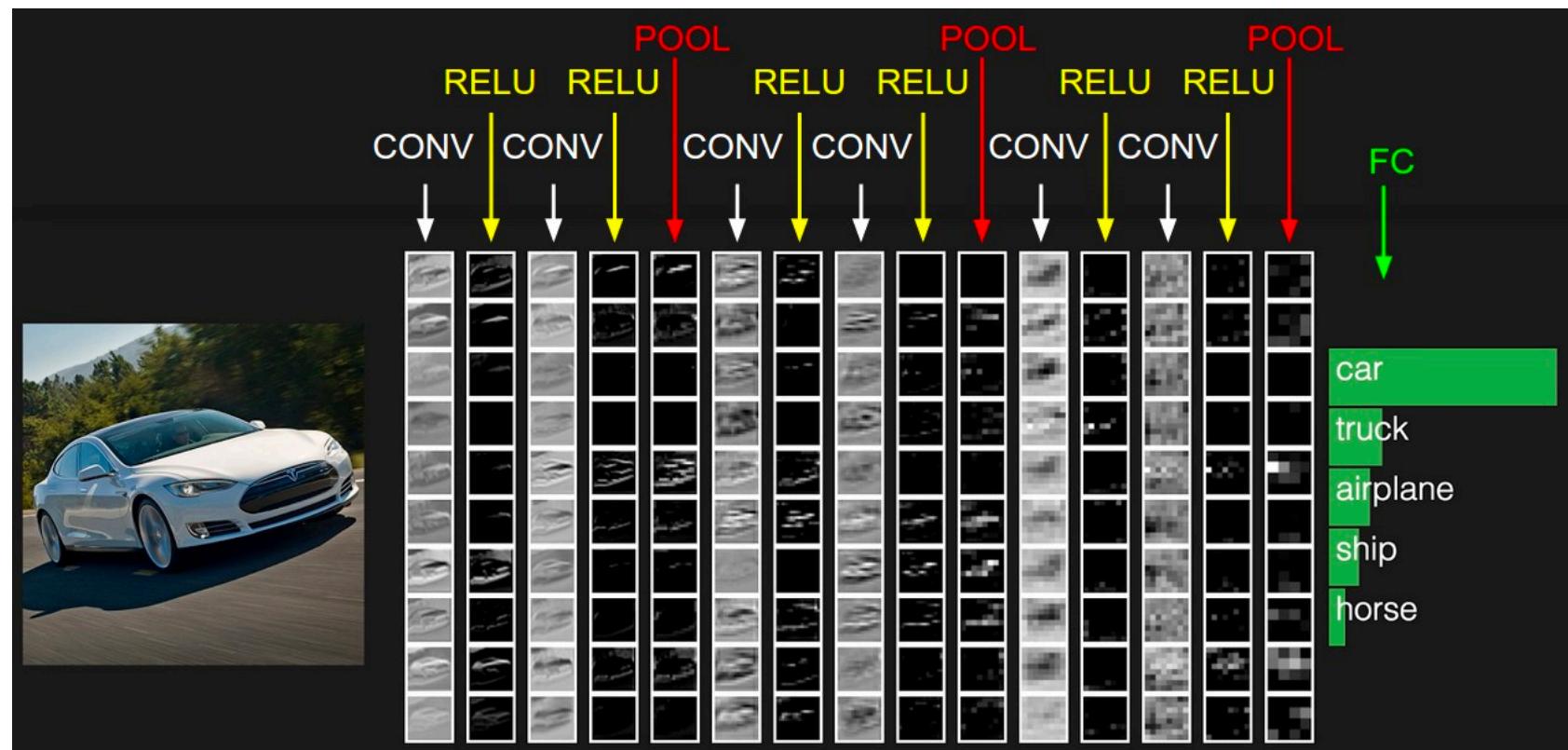
<https://indoml.com>

1*1 Convolutions



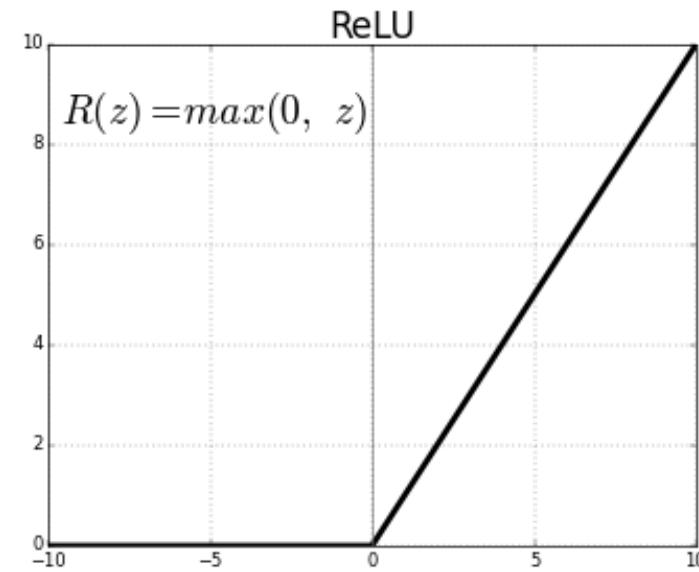
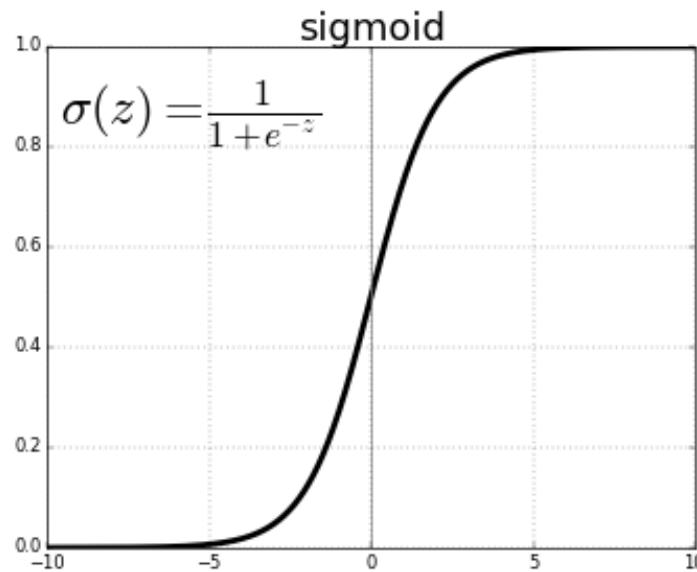
Convolution Neural Network

- An example of ConvNet architecture.



Activation Functions

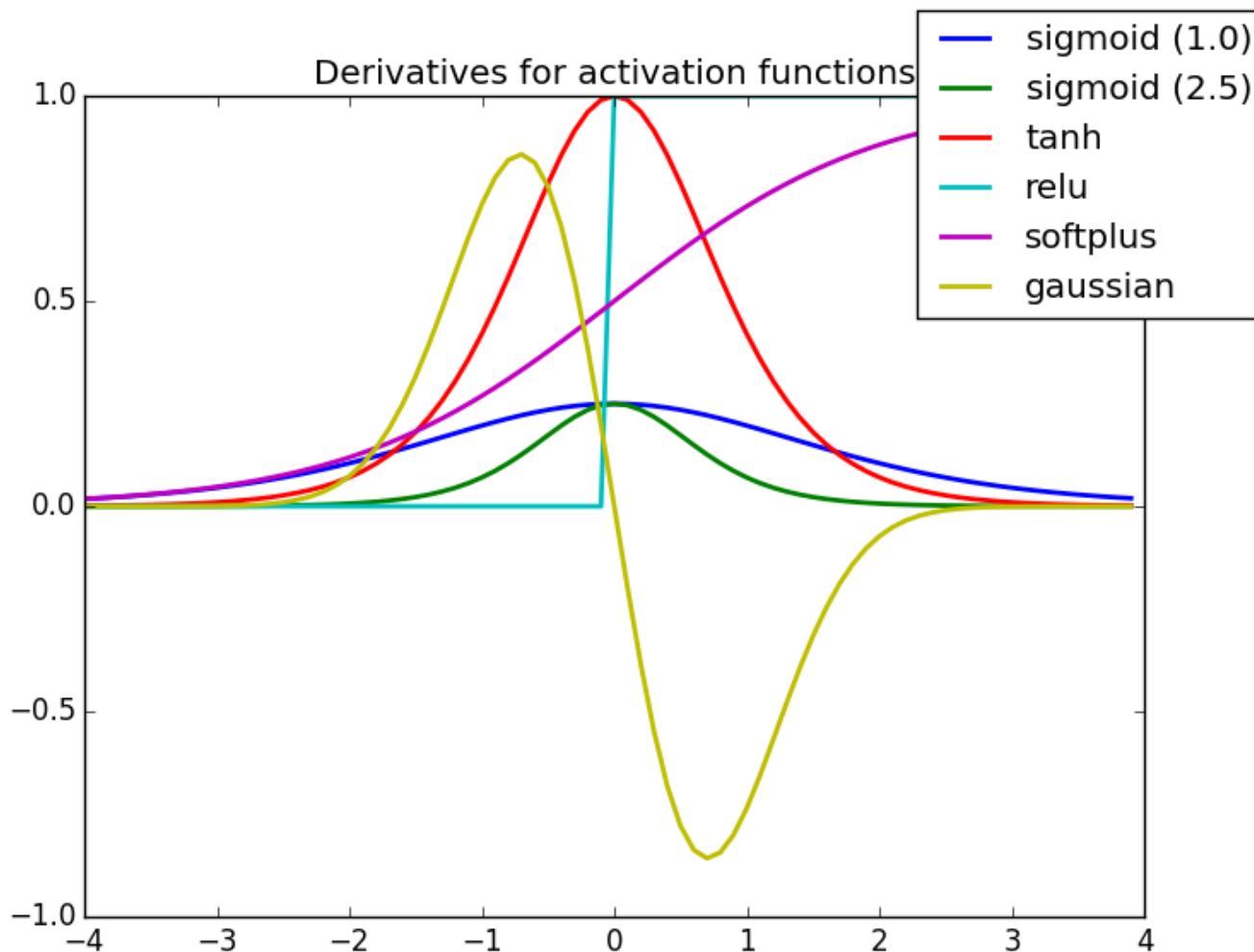
- Sigmoid Function
- ReLU Function
- More



Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a. k. a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
Arctan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU) [2]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parametric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Source: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>

Derivatives of Activation Functions

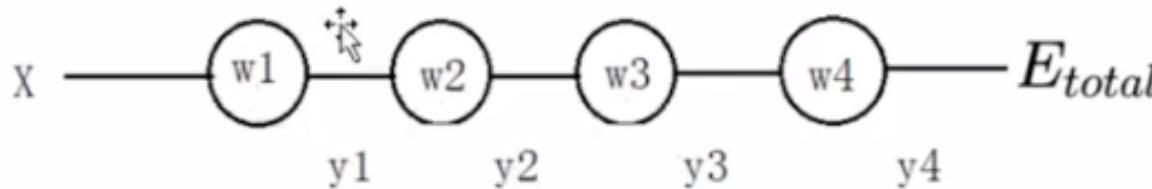


Source: <https://www.picswe.com/pics/relu-activation-7d.html>

Gradient Vanishing and Exploding

$$w_1^+ = w_1 - \eta \frac{\partial E_{total}}{\partial w_1}$$

$$y_i = \sigma(z_i) = \sigma(w_i x_i + b_i)$$



$$\begin{aligned}\frac{\partial E_{total}}{\partial w_1} &= \frac{\partial E_{total}}{\partial y_4} \frac{\partial y_4}{\partial z_4} \frac{\partial z_4}{\partial x_4} \frac{\partial x_4}{\partial z_3} \frac{\partial z_3}{\partial x_3} \frac{\partial x_3}{\partial z_2} \frac{\partial z_2}{\partial x_2} \frac{\partial x_2}{\partial z_1} \frac{\partial z_1}{\partial w_1} \\ &= \frac{\partial E_{total}}{\partial y_4} \sigma'(z_4) w_4 \sigma'(z_3) w_3 \sigma'(z_2) w_2 \sigma'(z_1) x_1\end{aligned}$$

Batch Normalization

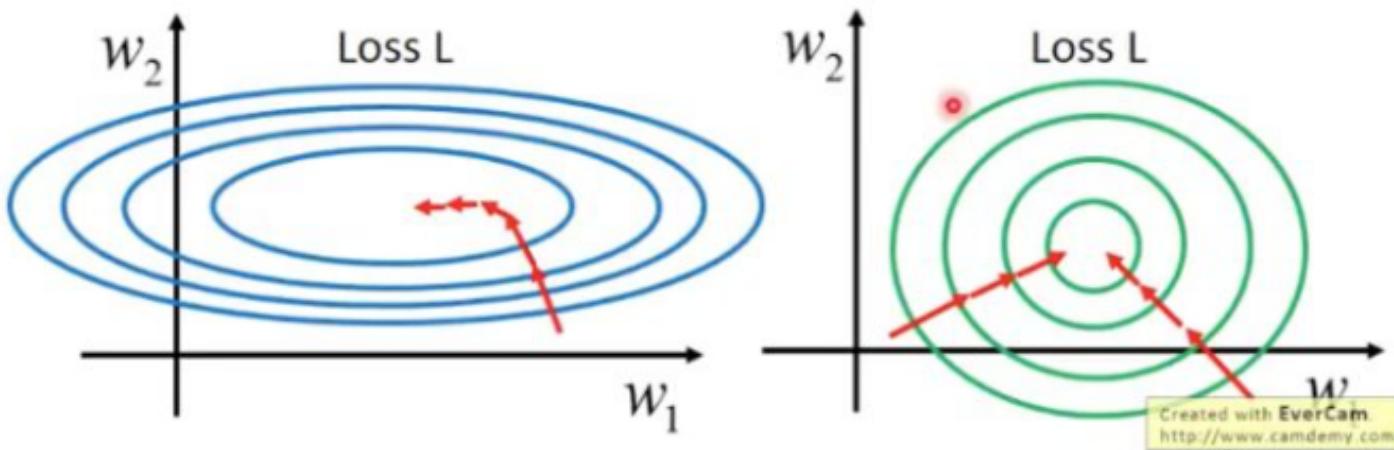
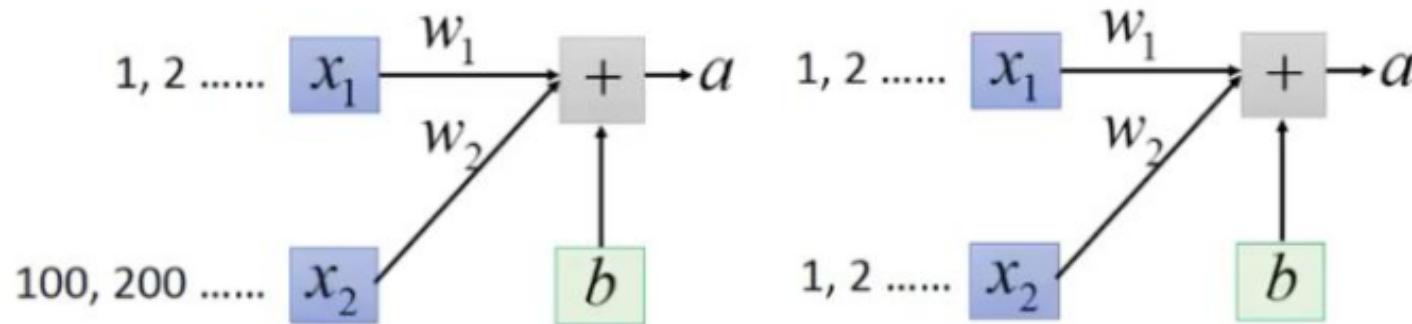
- Batch normalization speeds up the training by setting high learning rate
- Batch normalization prevents the gradients from getting too small or too large by normalizing data across each batch, as the name suggests. It also acts as a regularization method, similar to dropout.

Cited: <https://zhuanlan.zhihu.com/p/34480619> and Prof. Lee from TW Univ.

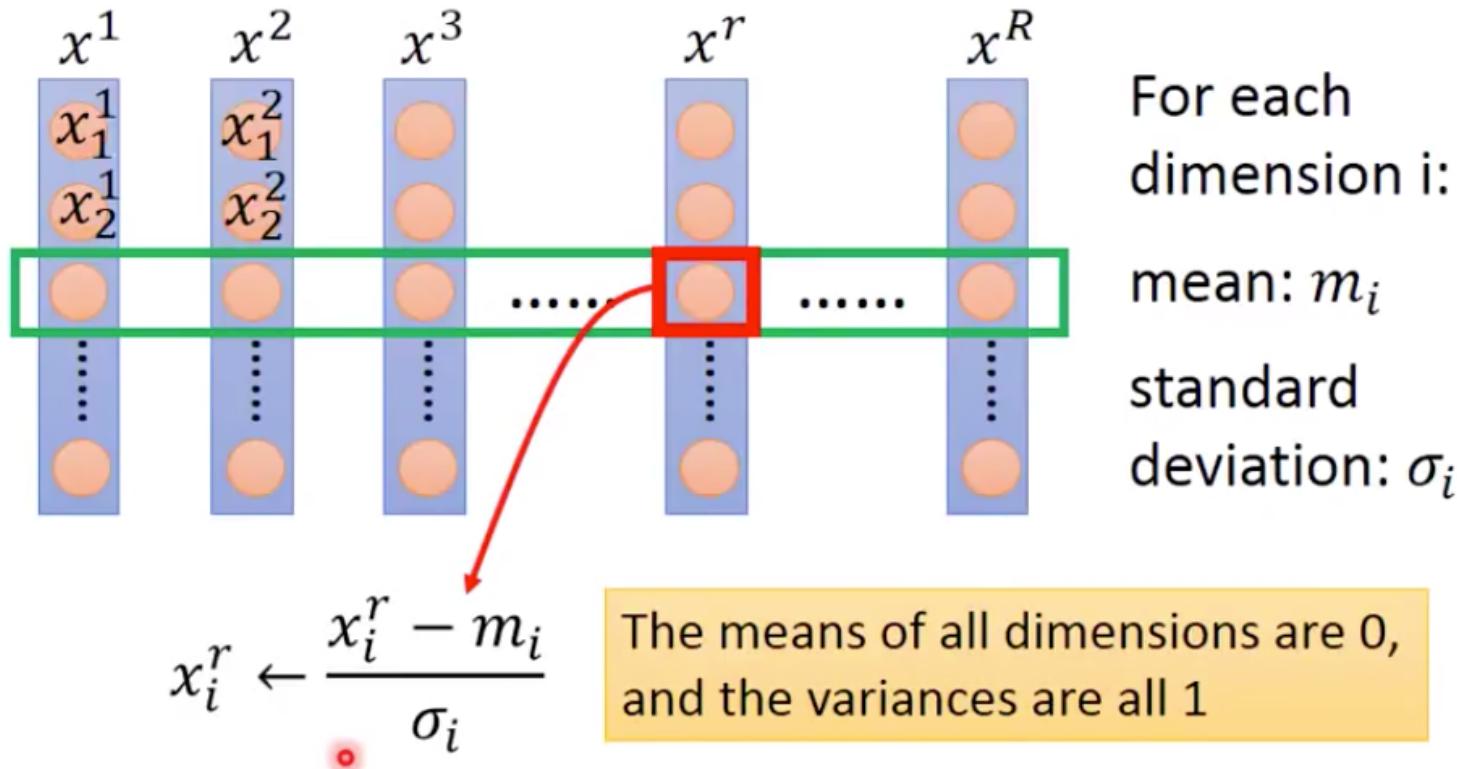
Feature Scaling

Feature Scaling

Make different features have the same scaling



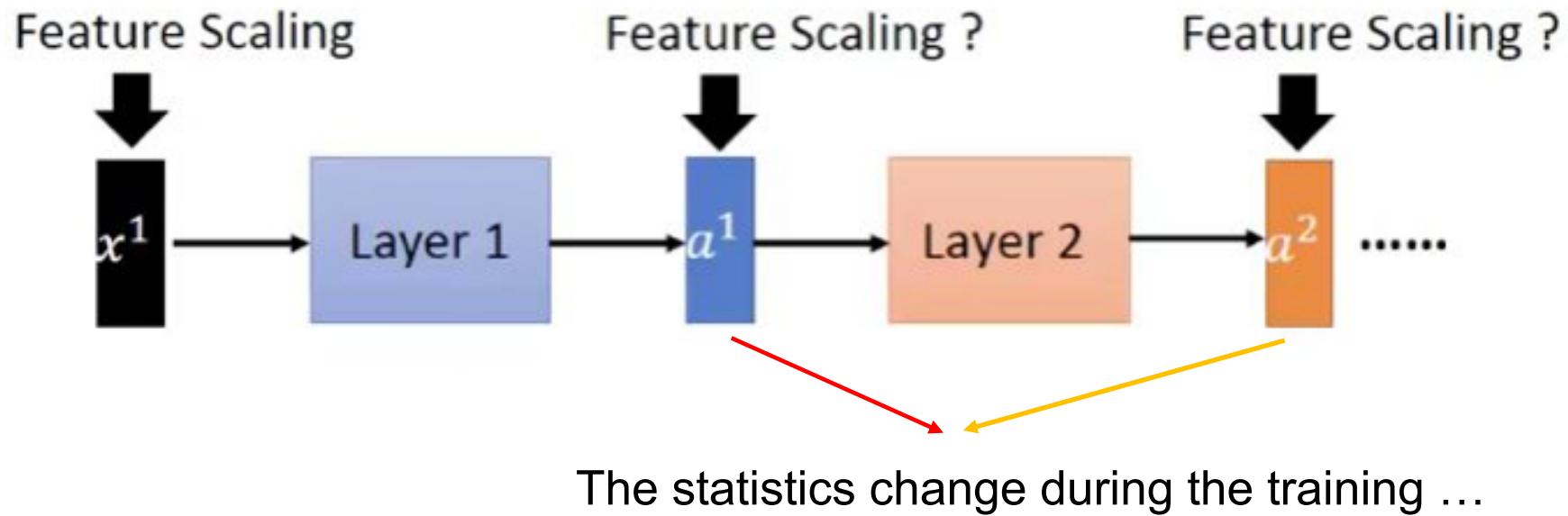
Feature Scaling



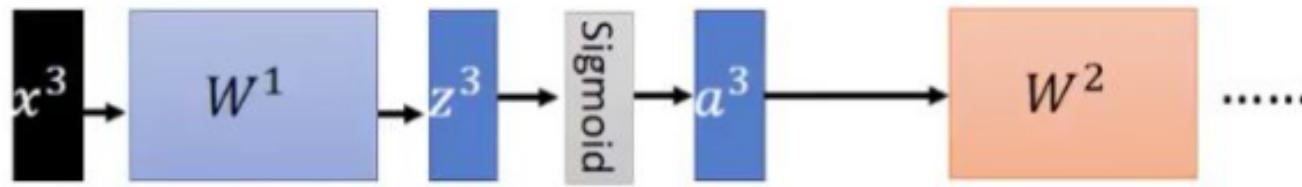
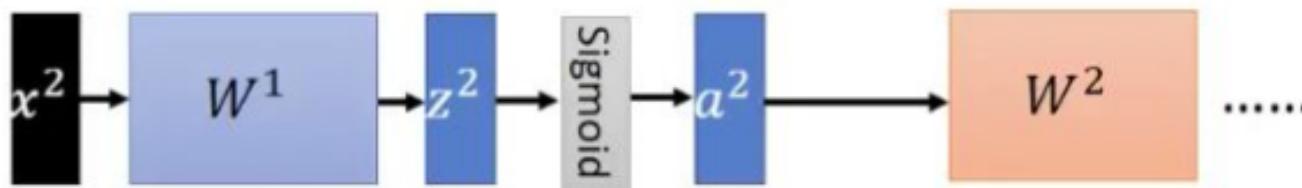
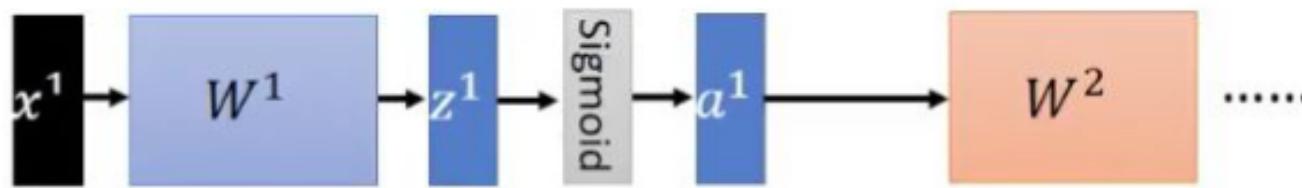
In general, gradient descent converges much faster with feature scaling than without it.

Scaling at Hidden Layer

- Internal Covariate Shift



Batch of Data

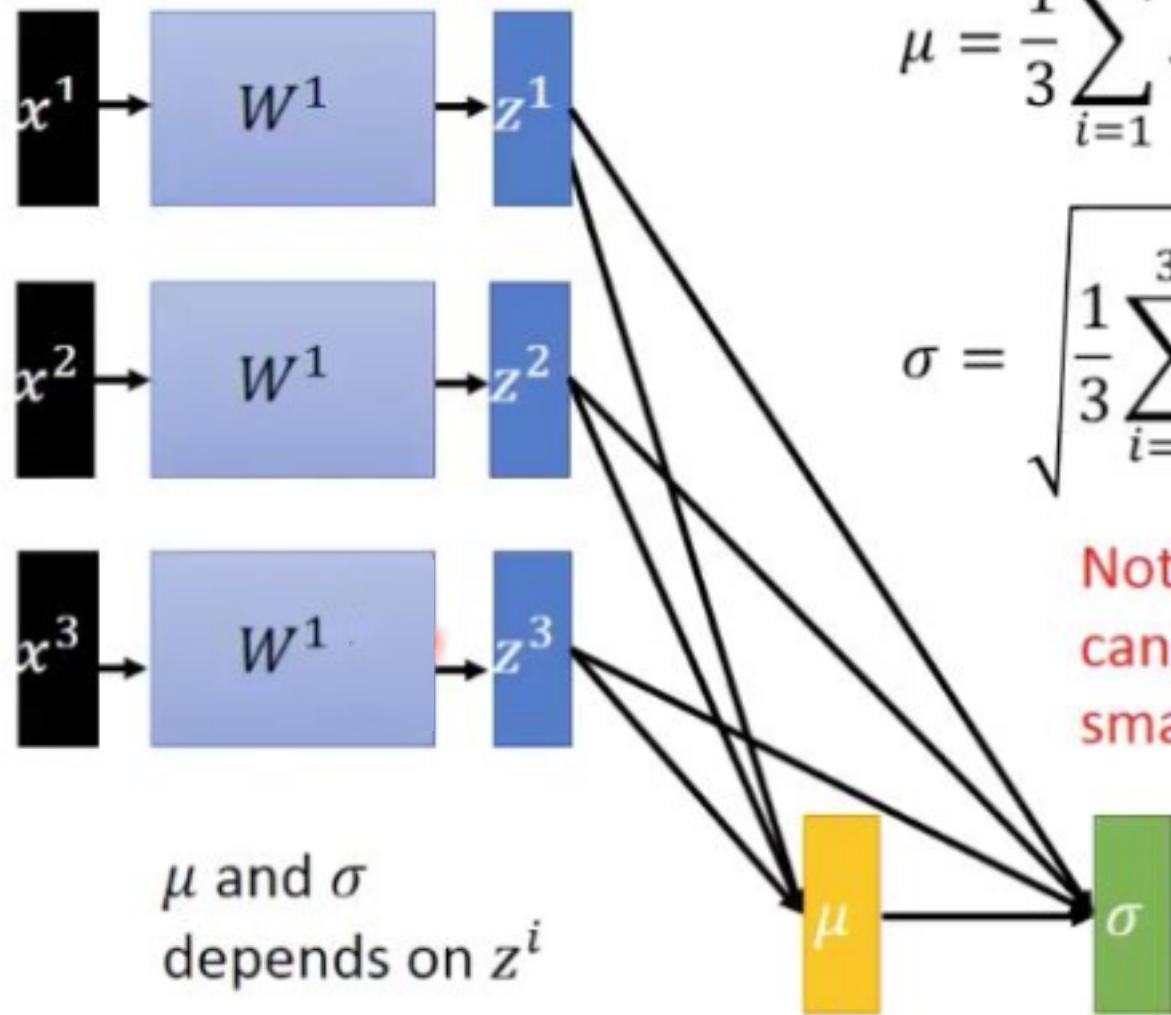


Batch

$$\begin{matrix} z^1 & z^2 & z^3 \end{matrix} = \begin{matrix} W^1 \\ x^1 & x^2 & x^3 \end{matrix}$$

Created with EverCam.

Batch normalization



$$\mu = \frac{1}{3} \sum_{i=1}^3 z^i$$

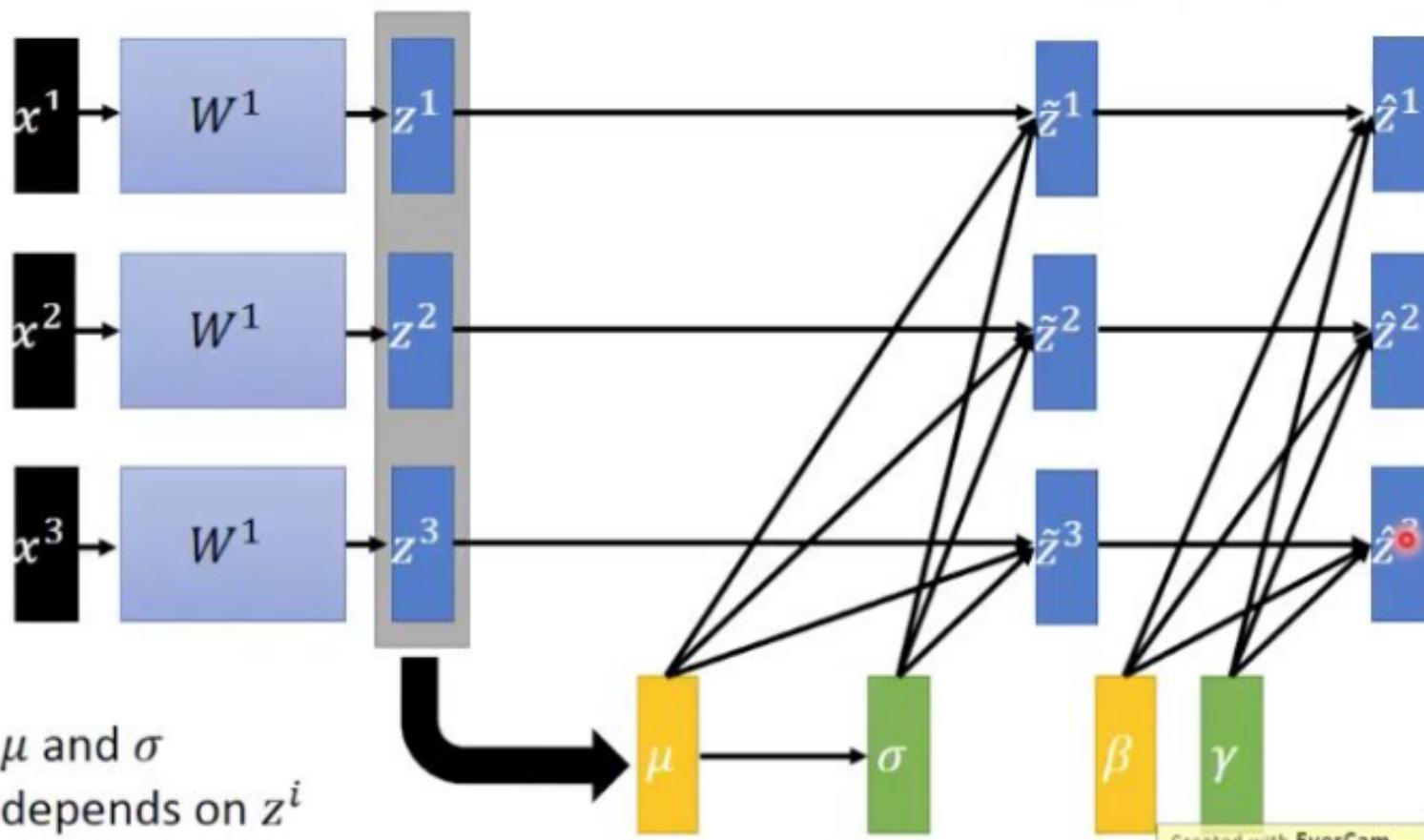
$$\sigma = \sqrt{\frac{1}{3} \sum_{i=1}^3 (z^i - \mu)^2}$$

Note: Batch normalization
cannot be applied on
small batch.

Batch normalization

$$\tilde{z}^i = \frac{z^i - \mu}{\sigma}$$

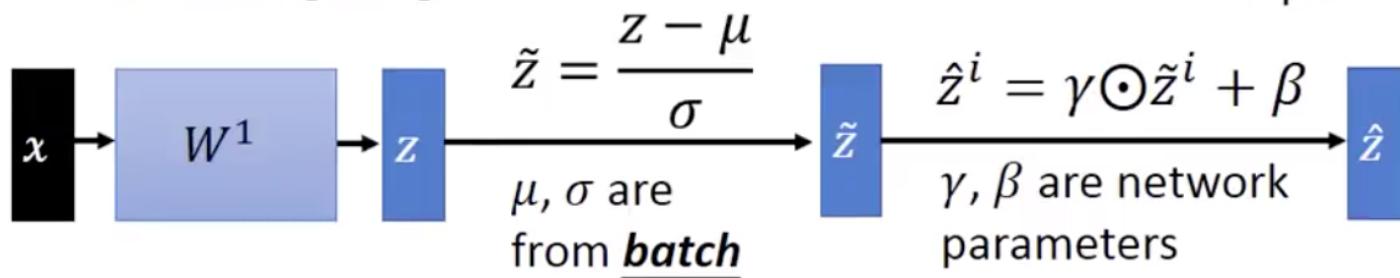
$$\hat{z}^i = \gamma \odot \tilde{z}^i + \beta$$



BN at Testing

Batch normalization

- At testing stage:



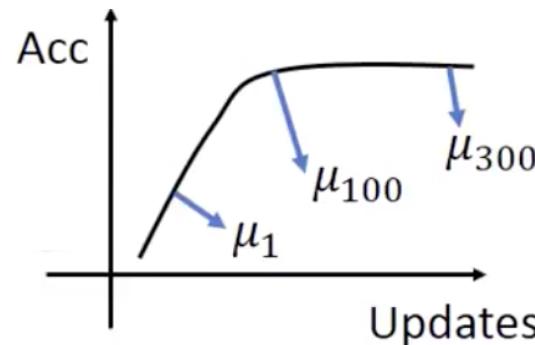
We do not have **batch** at testing stage.

Ideal solution:

Computing μ and σ using the whole training dataset.

Practical solution:

Computing the moving average of μ and σ of the batches during training.



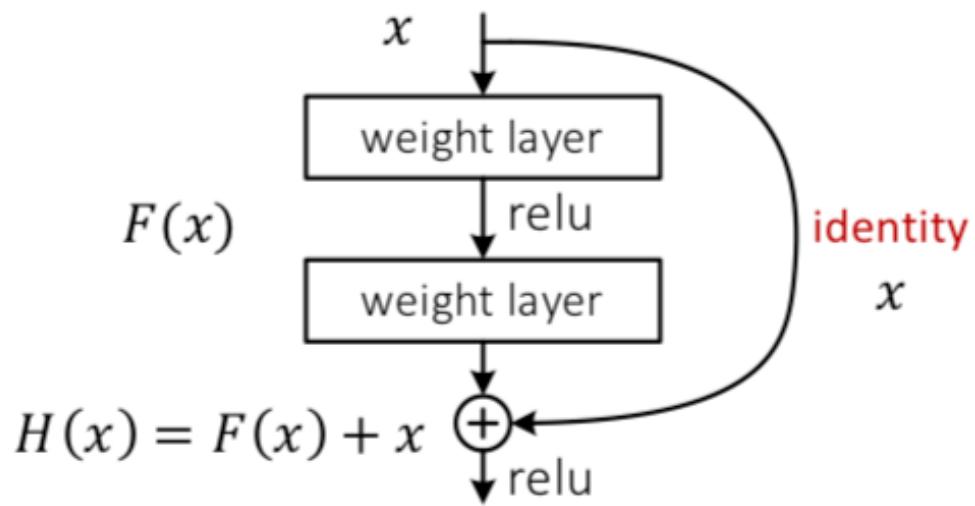
BN: Benefits

- Reduce the covariate shift, speed up the training
- Reduce the vanishing/exploding gradients
- Less affected by weights initialization

Residual Training

Deep Residual Learning

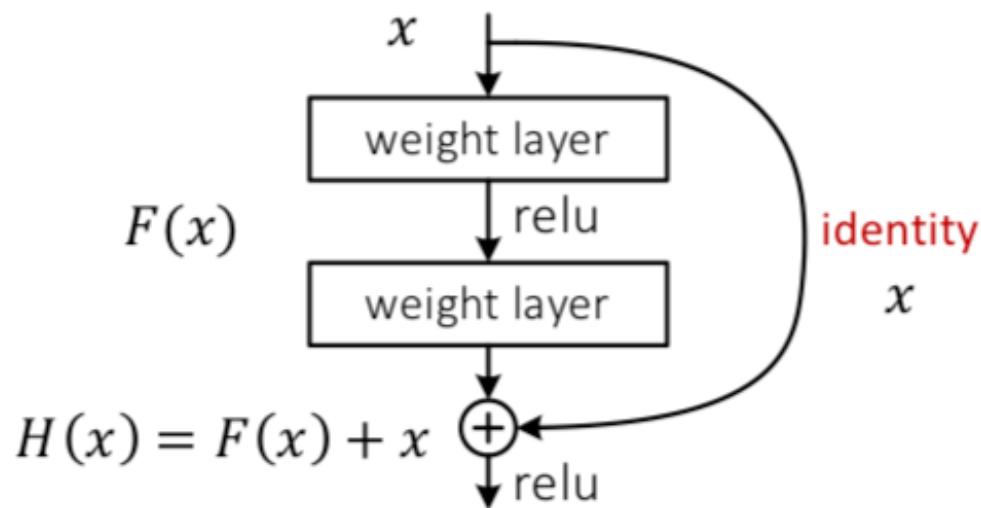
- **Residual** net



$H(x)$ is any desired mapping,
hope the 2 weight layers fit $H(x)$
hope the 2 weight layers fit $F(x)$
let $H(x) = F(x) + x$

Deep Residual Learning

- $F(x)$ is a **residual mapping w.r.t. identity**



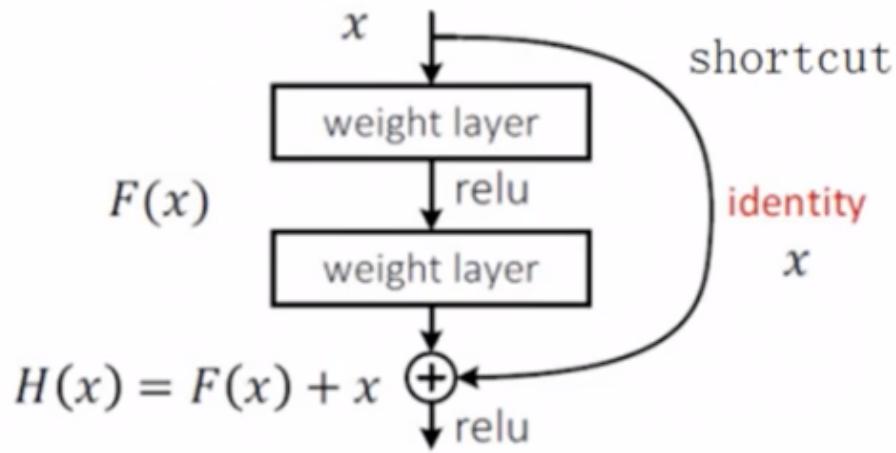
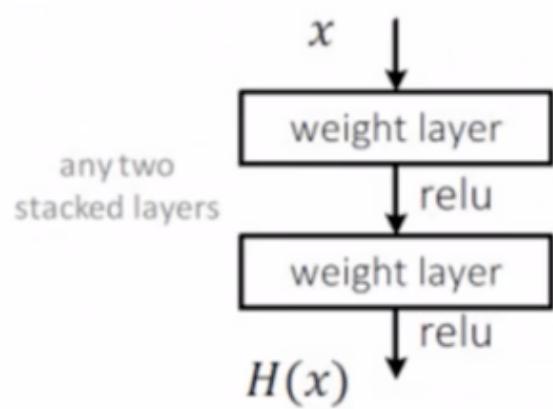
- If identity were optimal, easy to set weights as 0
- If optimal mapping is closer to identity, easier to find small fluctuations

Network Design

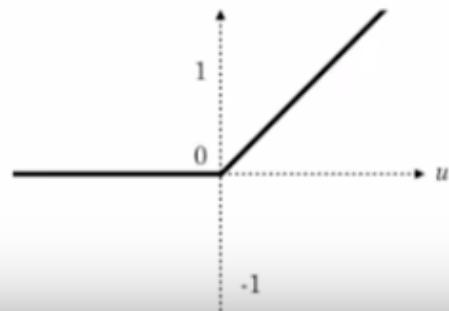
plain net



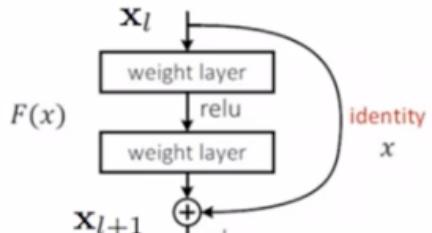
More zeros at ReLU



$$f(u) = \max(0, u)$$



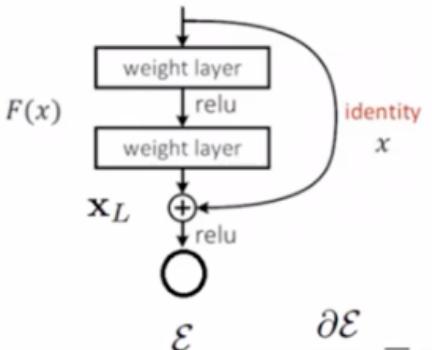
Less Gradient Vanishing



$$\mathbf{x}_{l+1} = \mathbf{x}_l + \mathcal{F}(\mathbf{x}_l, \mathcal{W}_l)$$

a term of $\frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}$ that propagates information directly without concerning any weight layers, and another term of $\frac{\partial \mathcal{E}}{\partial \mathbf{x}_L} \left(\frac{\partial}{\partial \mathbf{x}_l} \sum_{i=l}^{L-1} \mathcal{F} \right)$ that propagate through the weight layers. The additive term of $\frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}$ ensures that information is directly propagated back to *any shallower unit* l .

$$\mathbf{x}_{l+2} = \mathbf{x}_{l+1} + \mathcal{F}(\mathbf{x}_{l+1}, \mathcal{W}_{l+1}) = \mathbf{x}_l + \mathcal{F}(\mathbf{x}_l, \mathcal{W}_l) + \mathcal{F}(\mathbf{x}_{l+1}, \mathcal{W}_{l+1}).$$



$$\mathbf{x}_L \xrightarrow{\text{residual}} \mathbf{x}_l + \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x}_i, \mathcal{W}_i)$$

This “1” will back propagate the gradient at deeper layer

$$\frac{\partial \mathcal{E}}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L} \frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L} \left(1 + \frac{\partial}{\partial \mathbf{x}_l} \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x}_i, \mathcal{W}_i) \right)$$

Benefits

- Speed up the training
- Deeper network with reduced gradient vanishing

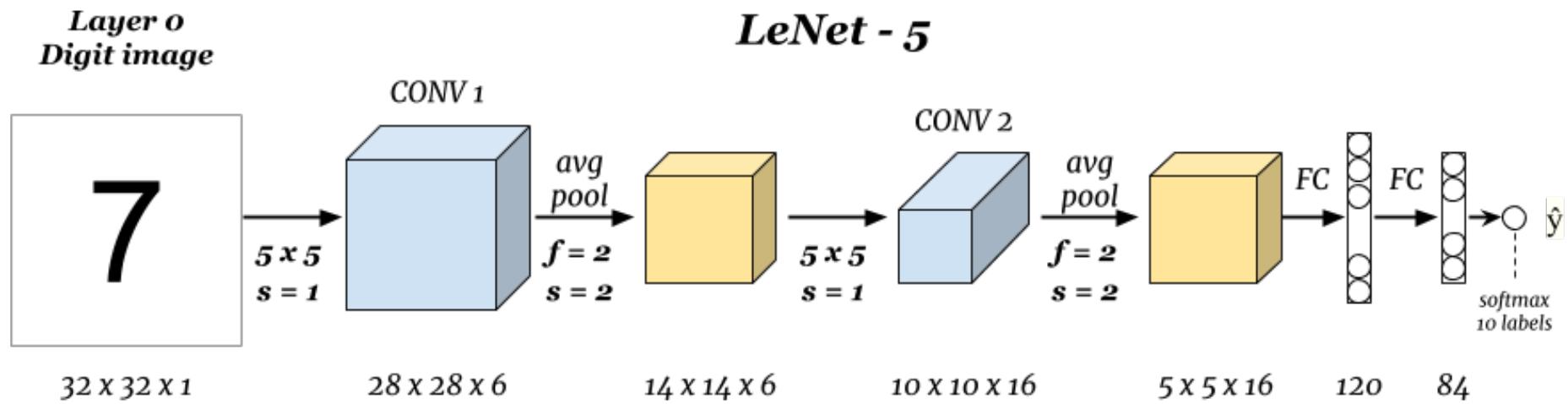
References

- 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/>

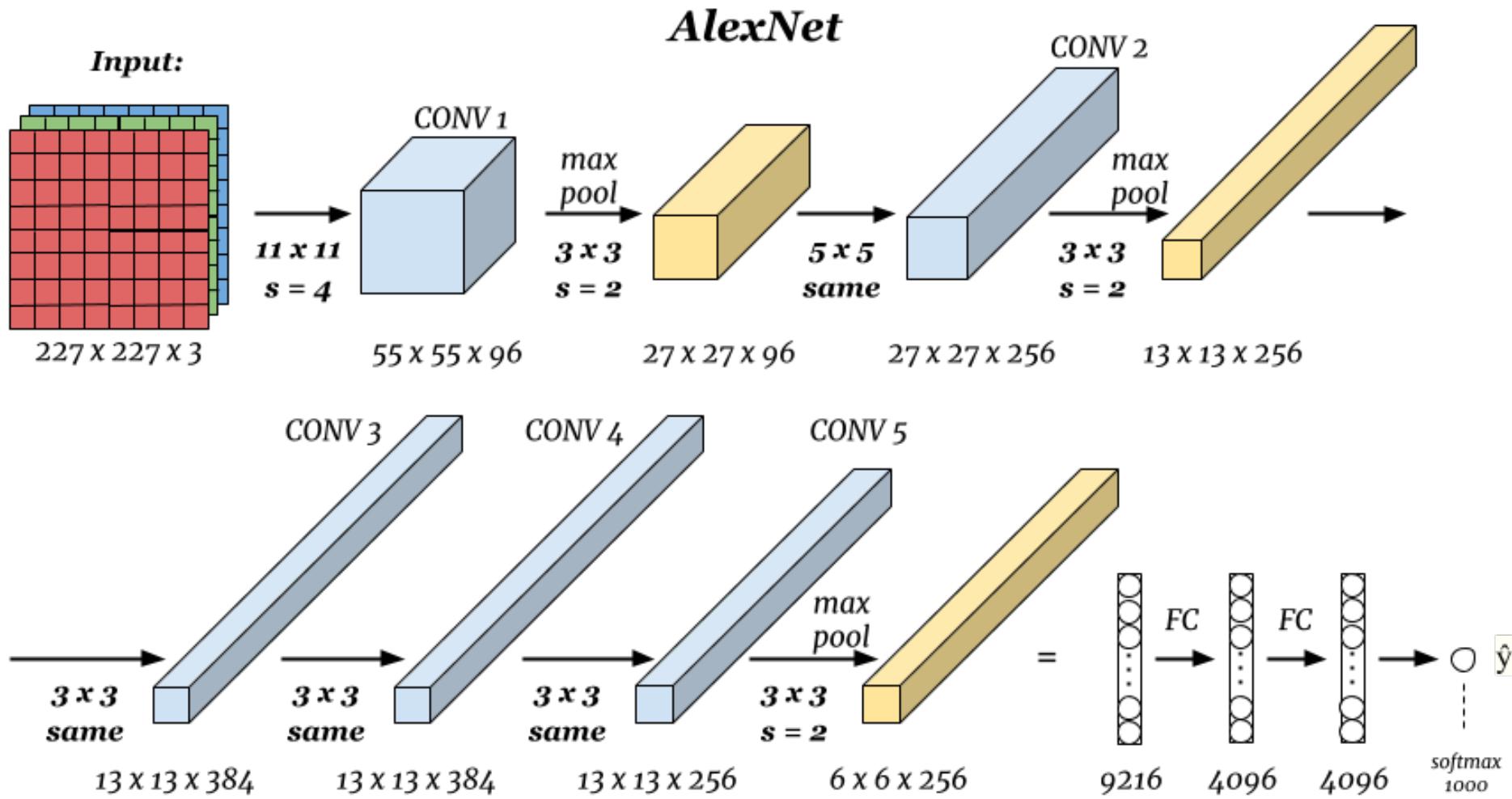
Classic Networks

- LeNet
- AlexNet
- VGG Net
- ResNet
- GooLeNet

LeNet

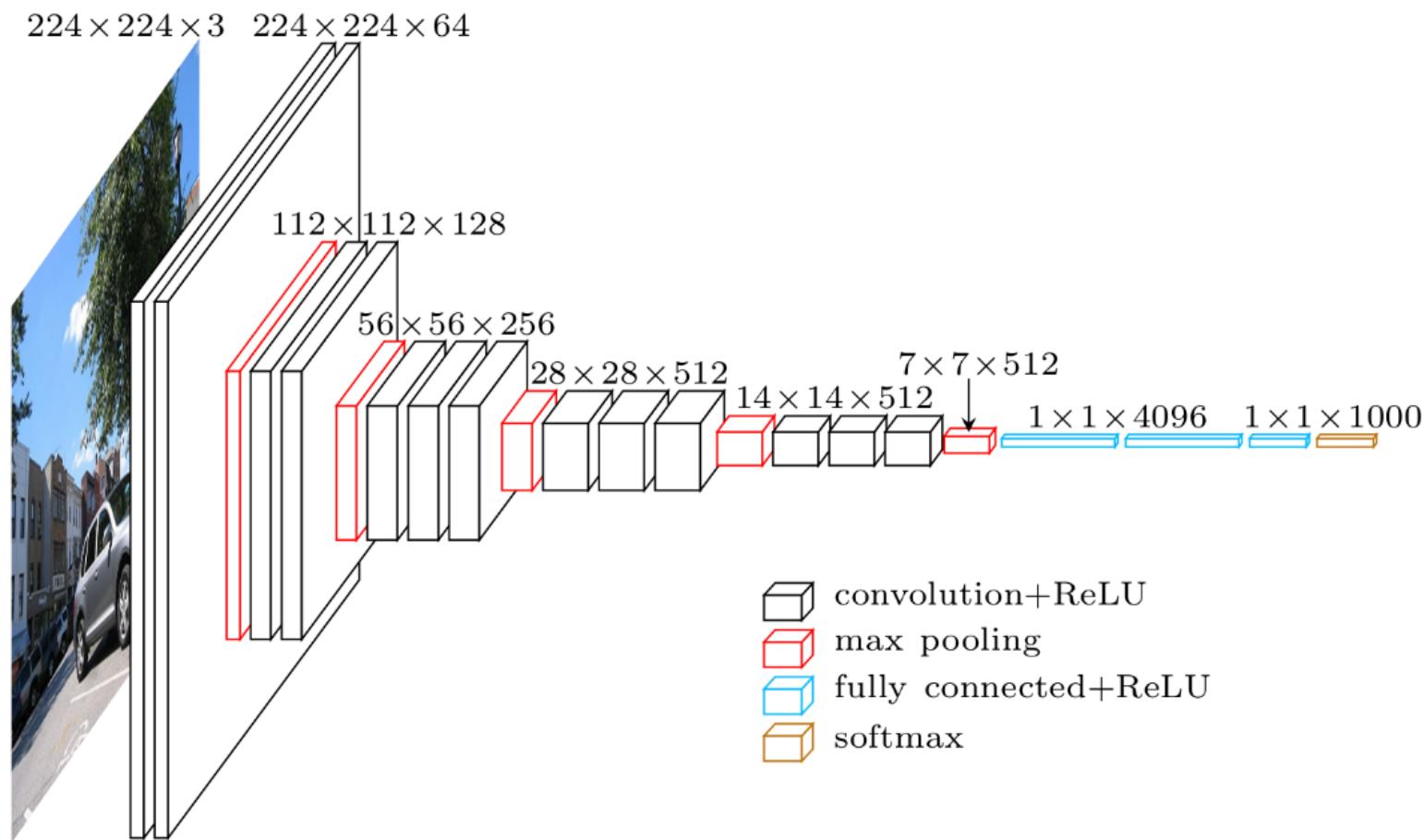


AlexNet

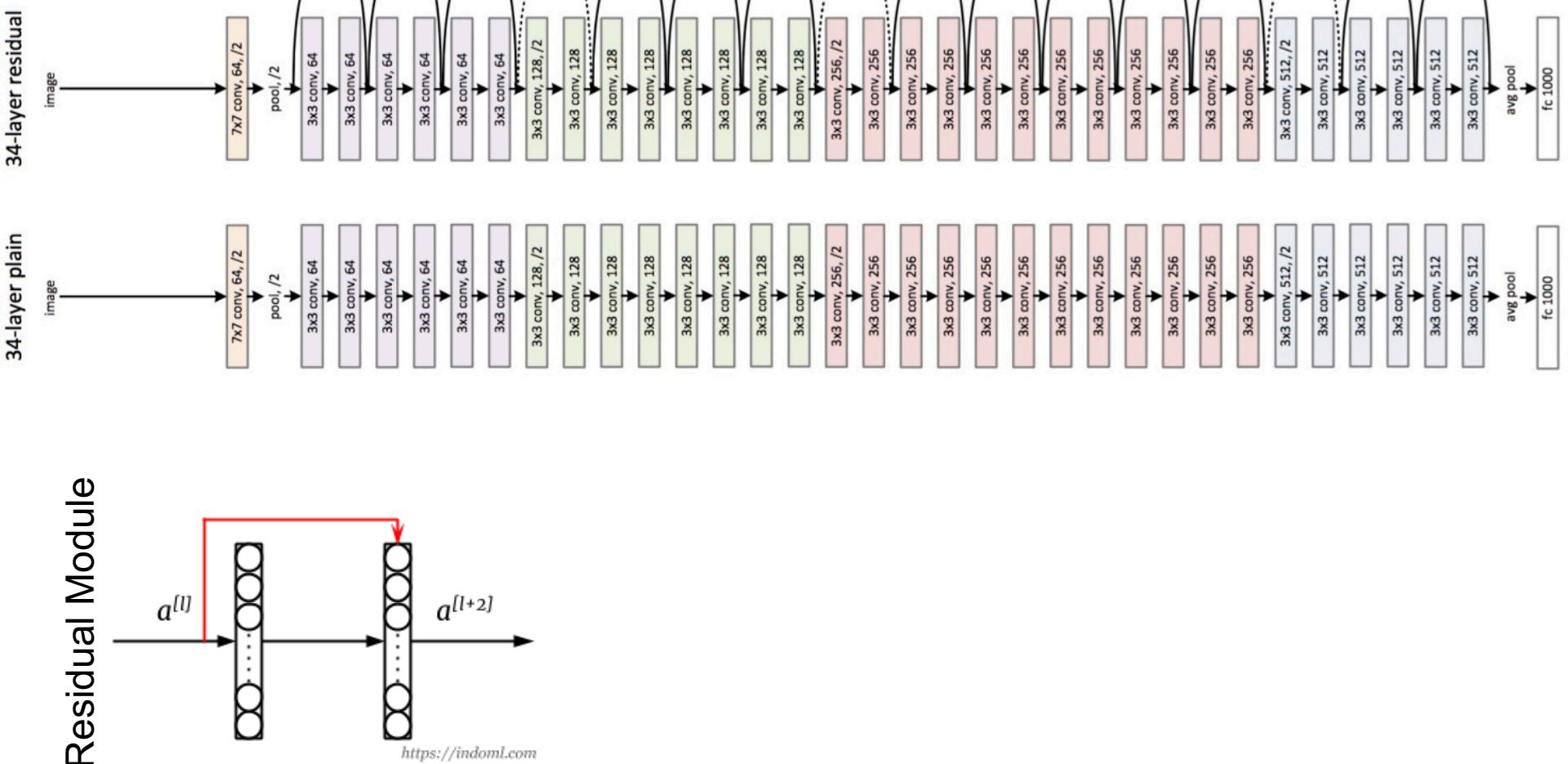


<https://indoml.com>

VGG-16

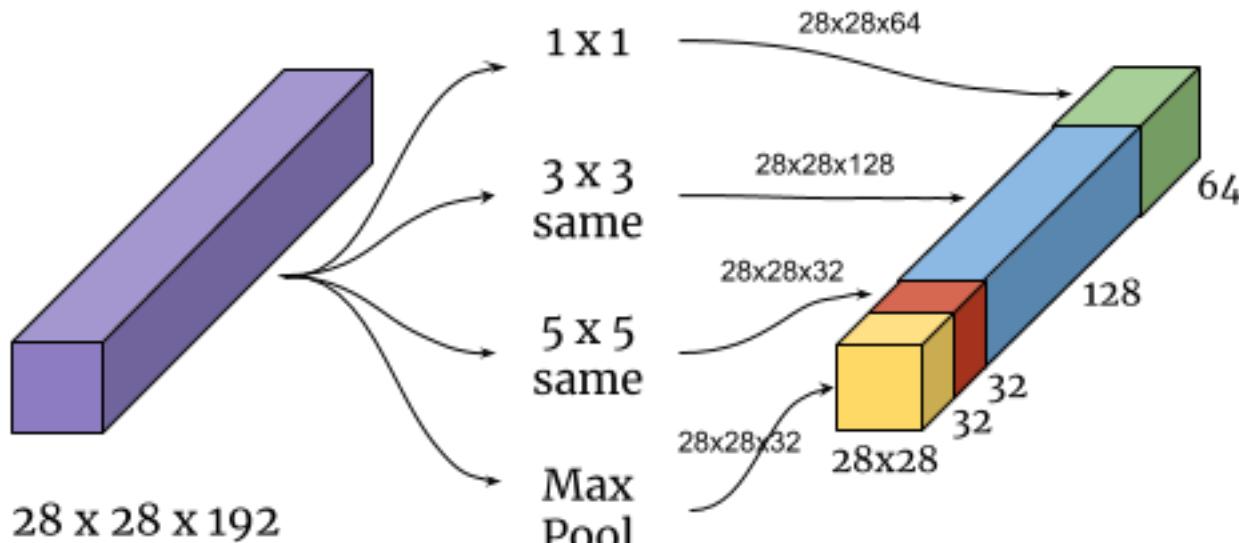


ResNet

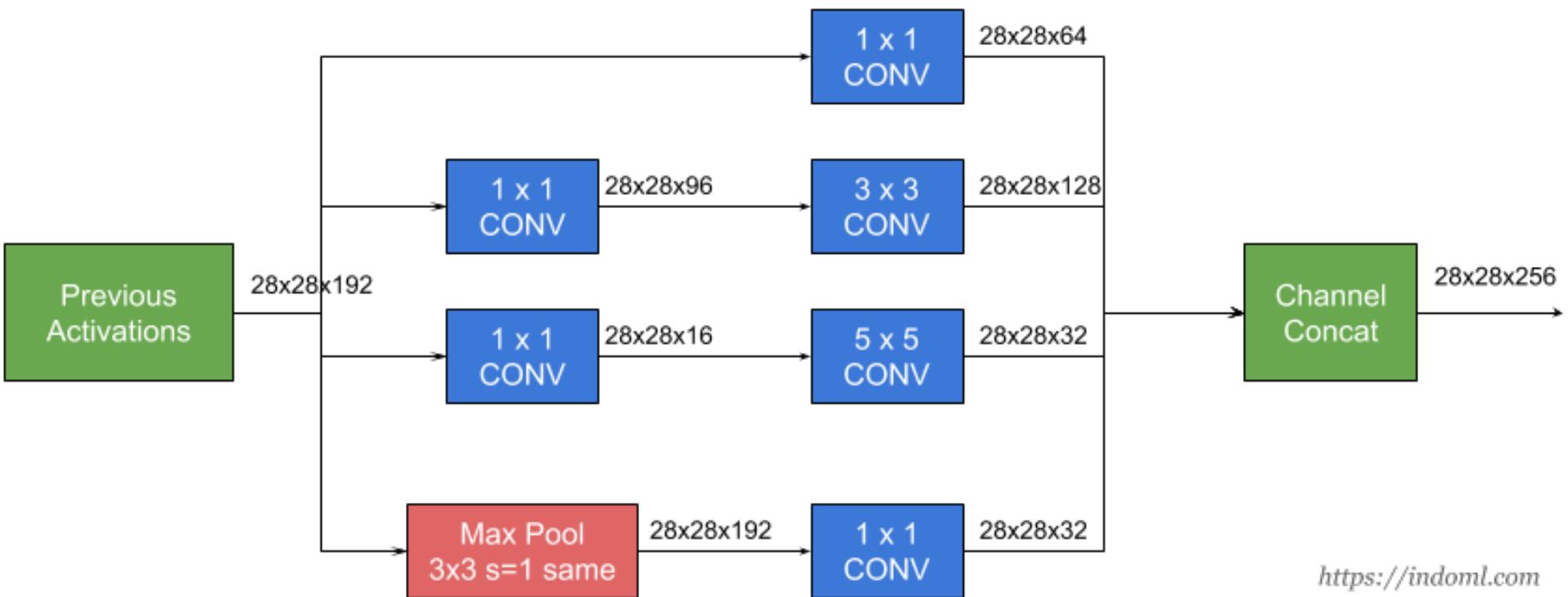


GoogLeNet

Inception Module

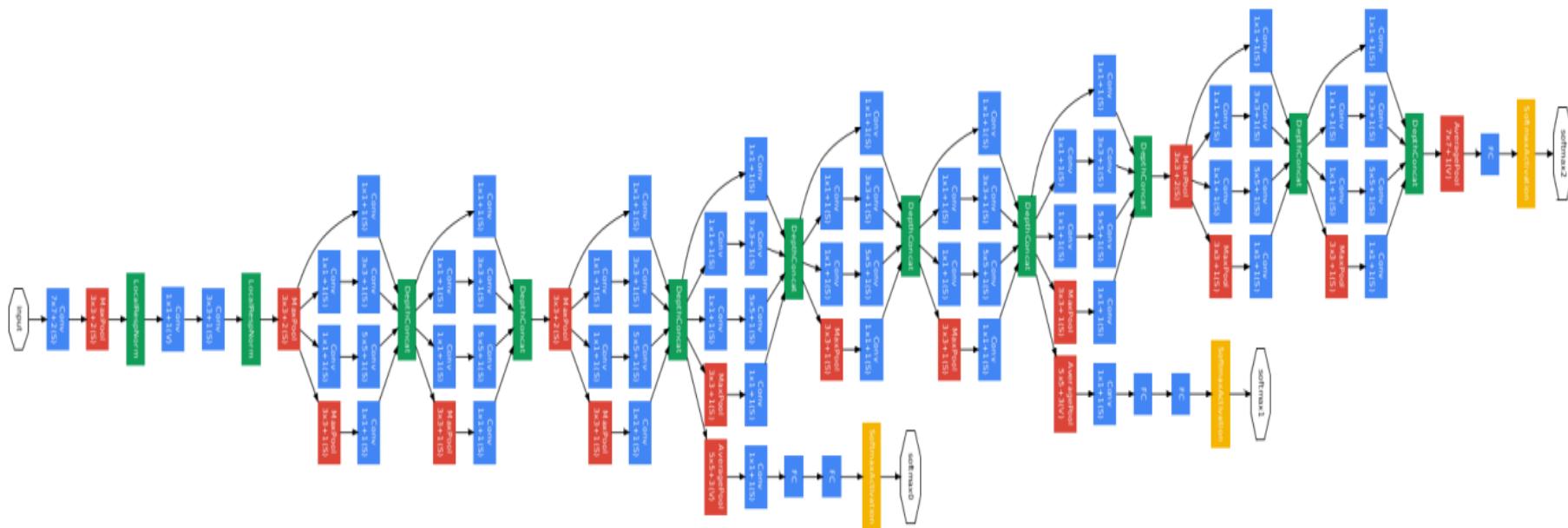


<https://indoml.com>



<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%	