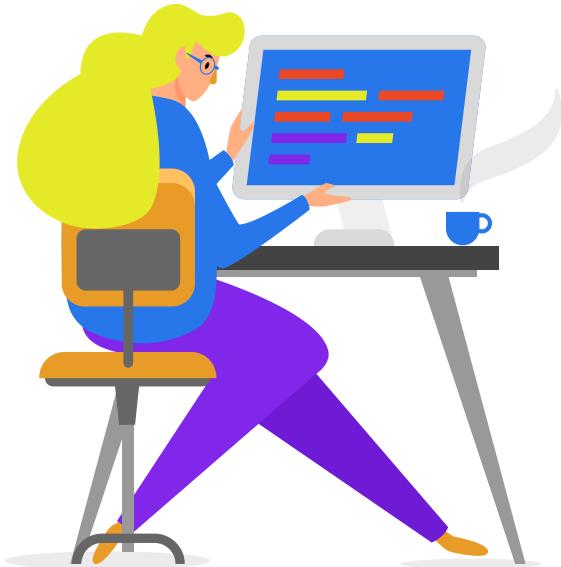


Deep Learning Computer Vision NLP

Thang Nguyen
Viet Nguyen

youtube.com/@vietnh1009

Nội dung



- 01 **Deep Learning**
Tổng quan về Deep Learning
- 02 **Neural Network**
Cấu trúc của Neural network
- 03 **Convolutional NN**
Cấu trúc của CNN
- 04 **Image Classification**
Bài toán phân loại ảnh
- 05 **Object detection**
Bài toán định vị đối tượng
- 06 **Other topics**
Image segmentation,
GANs

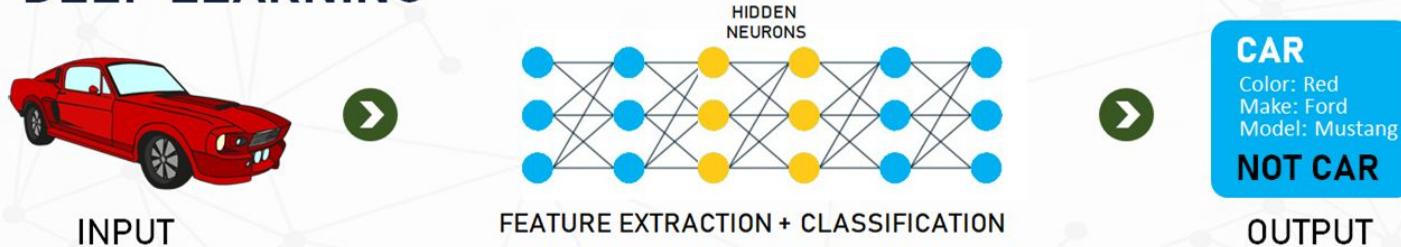
Tổng quan về Deep Learning

Tổng quan về Deep Learning

MACHINE LEARNING

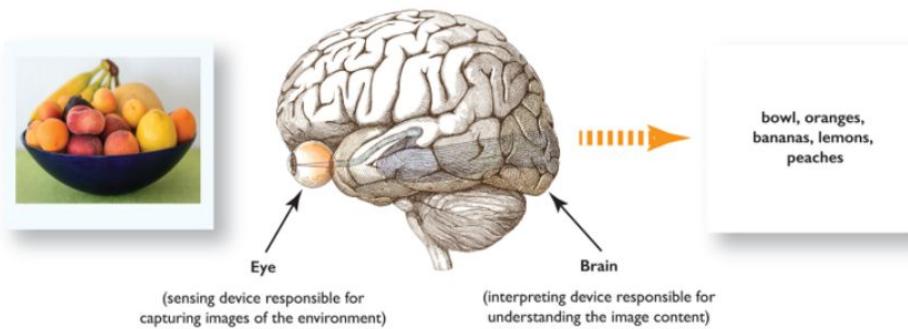


DEEP LEARNING

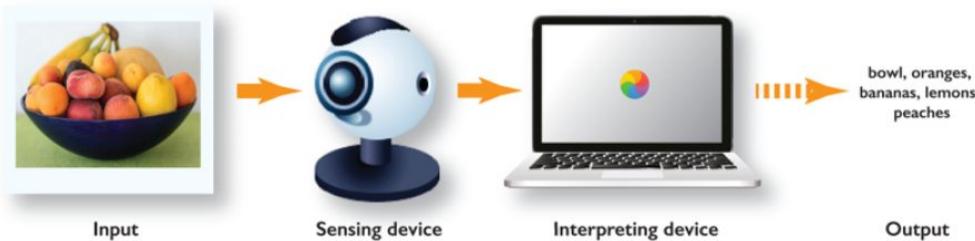


Deep Learning for Computer Vision

Human Vision System



Computer Vision System



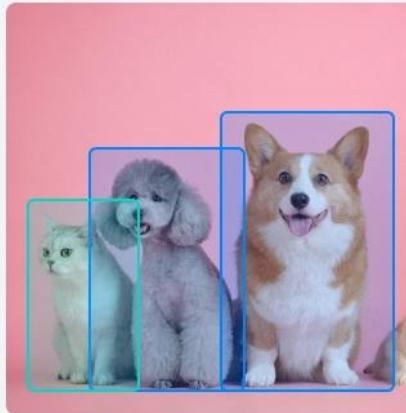
Deep Learning for Computer Vision

Classification



Cat

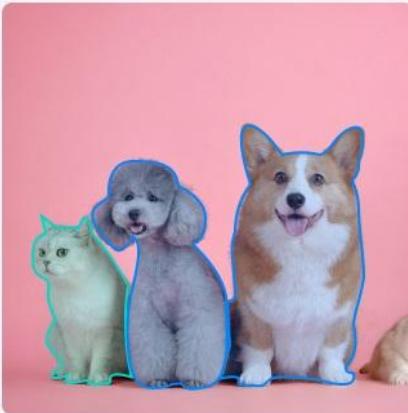
Detection



Cat

Dog

Segmentation



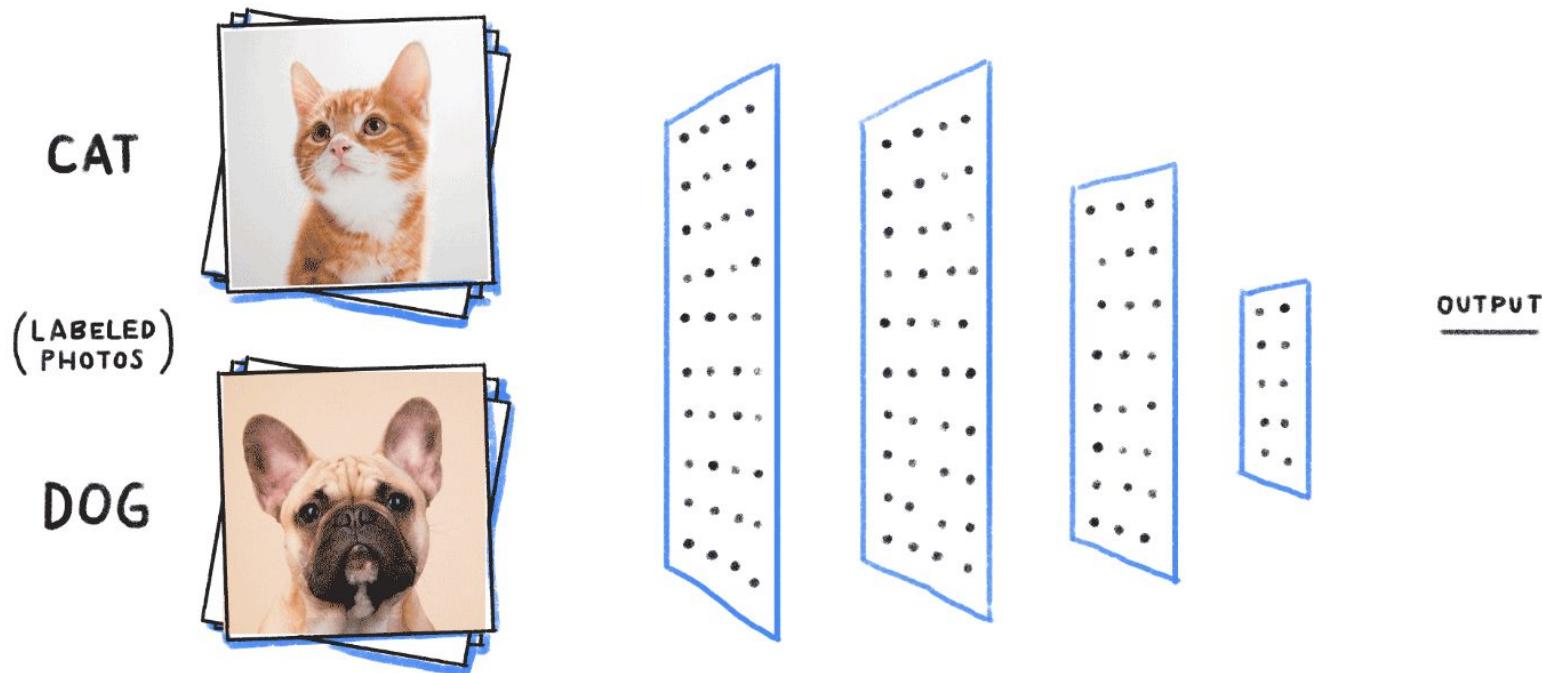
Cat

Dog

Single Object

Multiple Objects

Deep Learning for Computer Vision



Neural Network vs Deep Learning

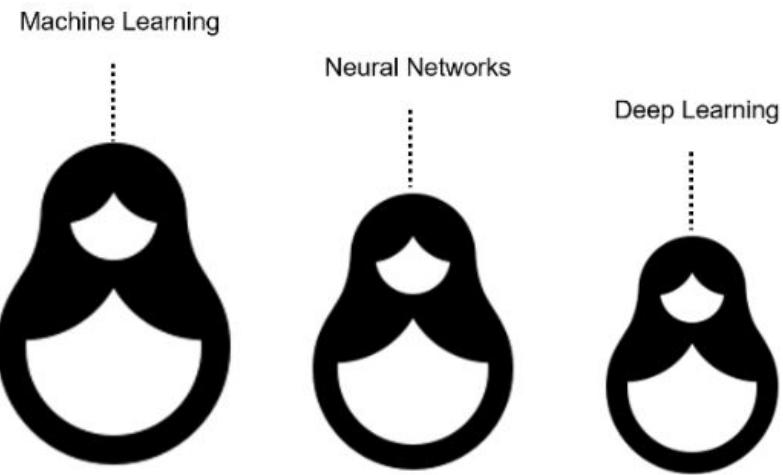
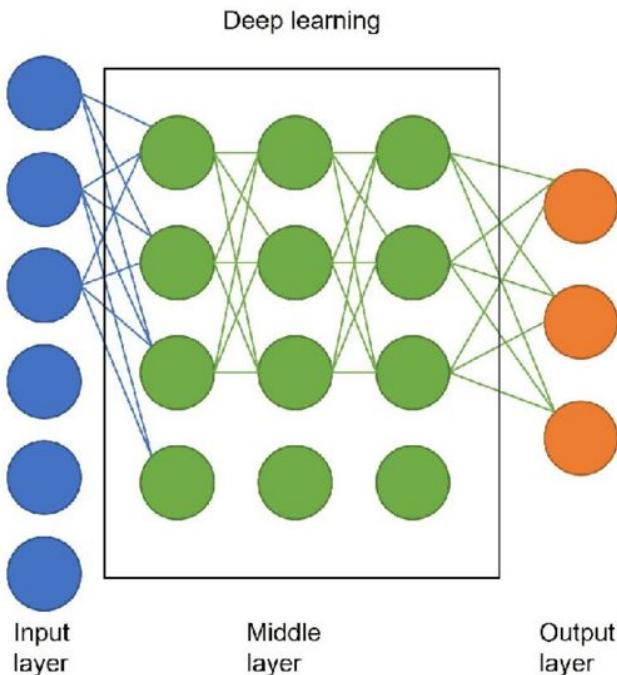
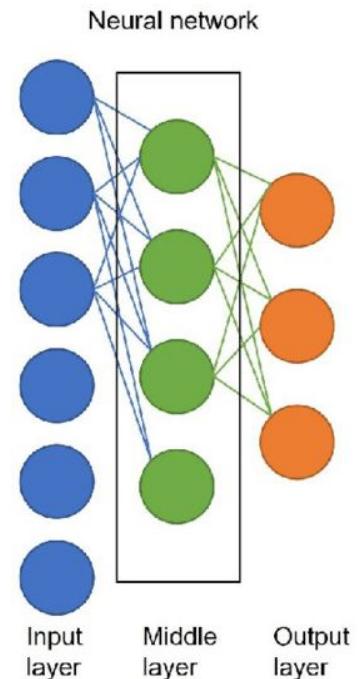


Image Classification



08	02	22	97	58	15	00	40	00	75	04	05	01	78	52	12	50	77	91	14
49	49	99	40	17	81	18	57	60	87	17	40	98	43	69	49	66	56	62	00
81	49	31	73	55	79	14	29	93	71	40	67	51	66	30	03	49	13	36	65
52	70	95	23	04	60	11	42	62	51	68	56	01	32	56	71	37	02	36	91
22	31	16	71	51	62	03	59	41	92	36	54	22	40	40	28	66	33	13	80
24	47	14	60	99	03	45	02	44	75	33	53	78	36	84	20	35	17	12	50
32	98	81	28	64	23	67	10	26	38	40	67	59	54	70	66	18	38	64	70
67	26	20	68	02	62	12	20	95	63	94	39	63	08	40	91	66	49	94	21
24	55	58	05	66	73	99	26	97	17	78	78	96	83	14	88	34	89	63	72
21	36	23	09	75	00	76	44	20	45	35	14	00	61	33	97	34	31	33	95
78	17	53	28	22	75	31	67	15	94	03	80	04	62	16	14	09	53	56	92
16	39	05	42	96	35	31	47	55	58	88	24	00	17	54	24	36	29	85	57
86	56	00	48	35	71	89	07	05	44	44	37	44	60	21	58	51	54	17	58
19	80	81	68	05	94	47	69	28	73	92	13	86	52	17	77	04	89	55	40
04	52	08	63	97	35	99	16	07	97	57	32	16	26	26	79	33	27	98	66
06	44	68	87	57	62	20	72	03	46	33	67	46	55	12	32	63	93	53	69
04	42	16	73	58	25	39	11	24	94	72	18	08	46	29	32	40	62	76	36
20	69	36	41	72	30	23	88	31	63	99	69	82	67	59	85	74	04	36	16
20	73	35	29	78	31	90	01	74	31	49	71	48	55	81	16	23	57	05	54
01	70	54	71	83	51	54	69	16	92	33	48	61	43	52	01	89	23	47	46

What the computer sees

image classification

82% cat
15% dog
2% hat
1% mug

Dataset

cat



dog



mug

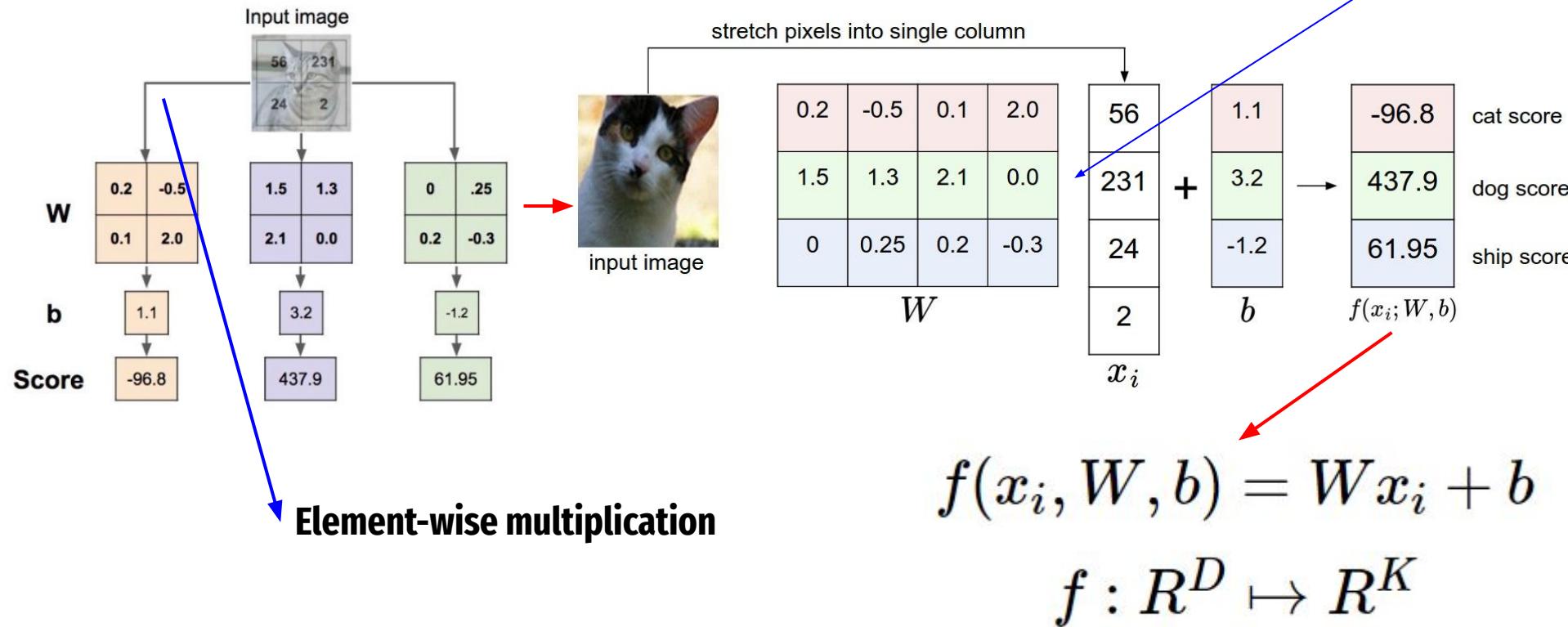


hat

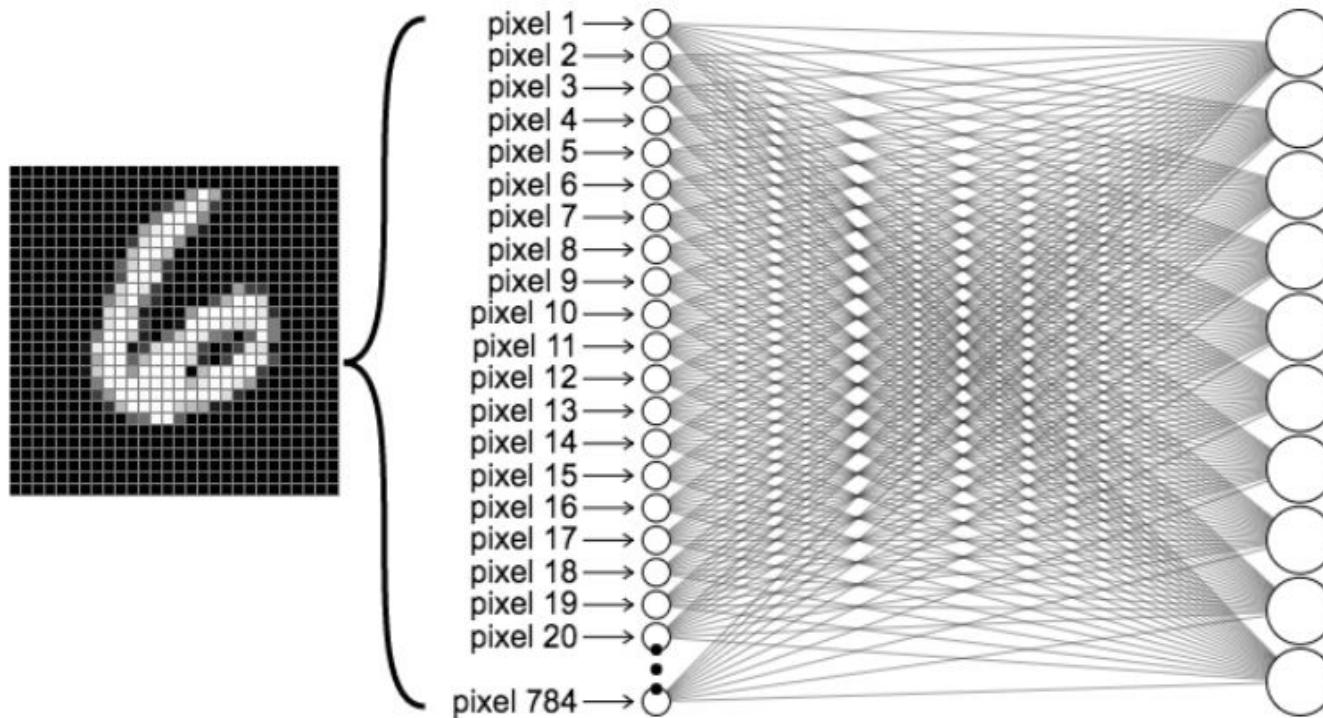


Linear Classifier - score function

Matrix multiplication

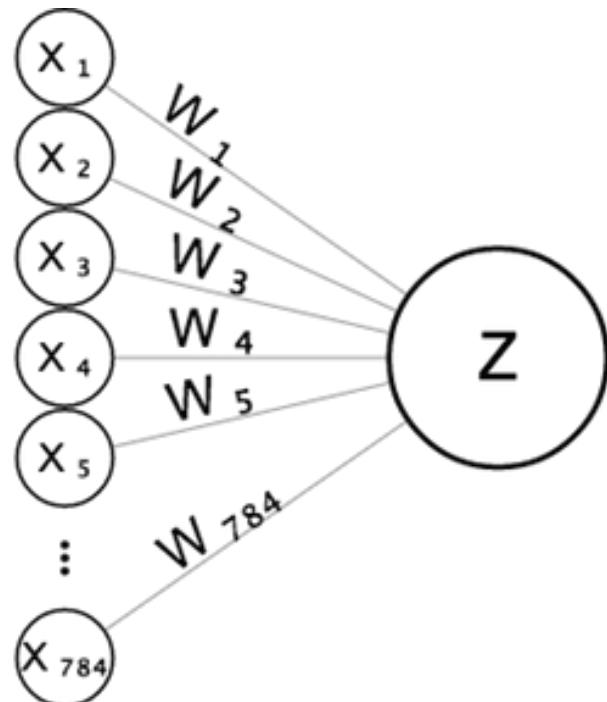
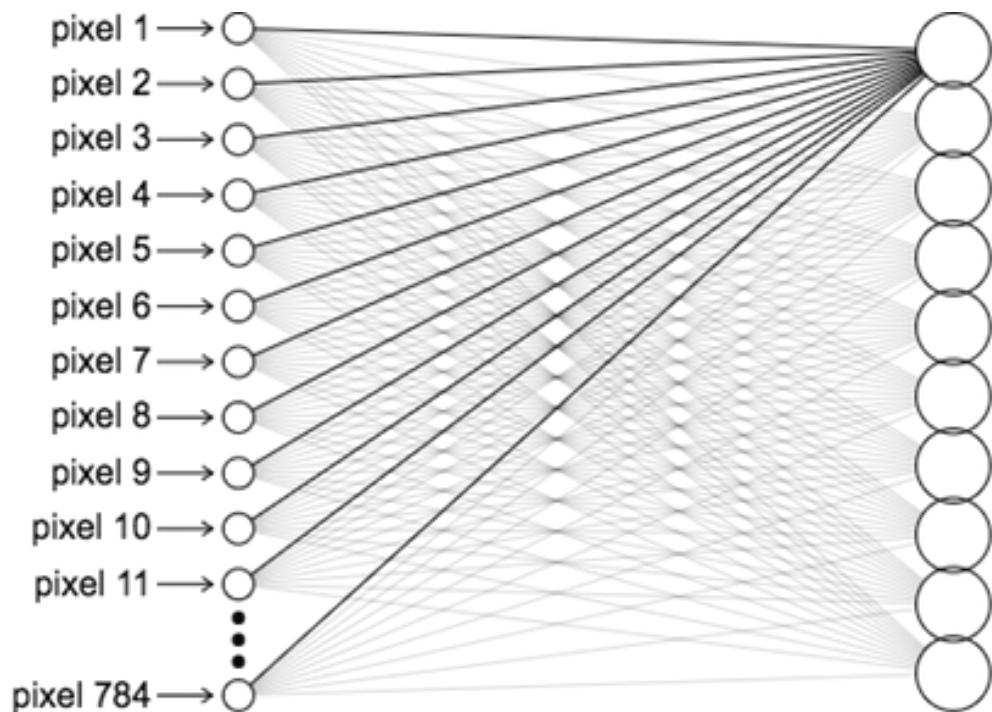


Linear Classifier explanation



[MNIST dataset](#)

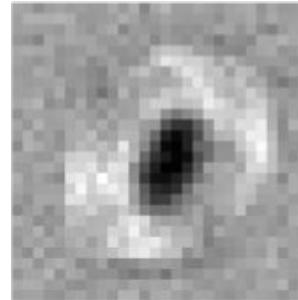
Linear Classifier - Weight visualization



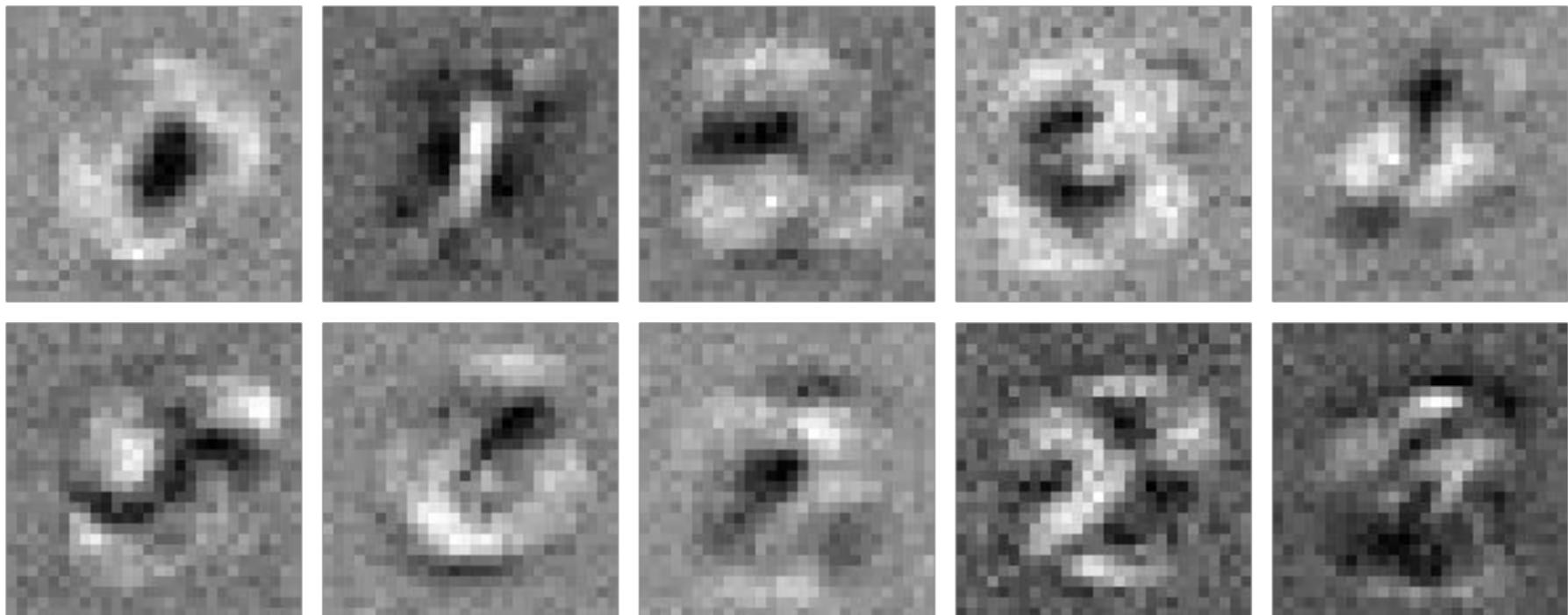
Linear Classifier - Weight visualization

$$\begin{matrix} X_1 & X_2 & X_3 & \dots & X_{28} \\ X_{29} & X_{30} & X_{31} & & X_{56} \\ X_{57} & X_{58} & X_{59} & & X_{84} \\ \vdots & & \ddots & & \vdots \\ X_{757} & X_{758} & X_{759} & & X_{784} \end{matrix} \circ \begin{matrix} W_1 & W_2 & W_3 & \dots & W_{28} \\ W_{29} & W_{30} & W_{31} & & W_{56} \\ W_{57} & W_{58} & W_{59} & & W_{84} \\ \vdots & & \ddots & & \vdots \\ W_{757} & W_{758} & W_{759} & & W_{784} \end{matrix} = Z$$

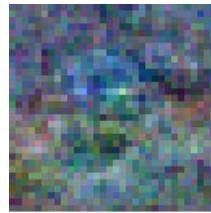
$$\begin{matrix} W_1 & W_2 & W_3 & \dots & W_{28} \\ W_{29} & W_{30} & W_{31} & & W_{56} \\ W_{57} & W_{58} & W_{59} & & W_{84} \\ \vdots & & \ddots & & \vdots \\ W_{757} & W_{758} & W_{759} & & W_{784} \end{matrix}$$



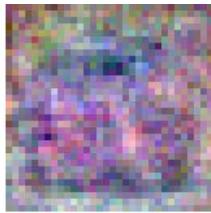
Linear Classifier - Weight visualization



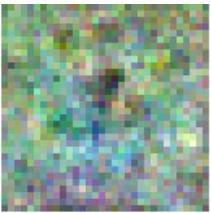
Linear Classifier - Weight visualization



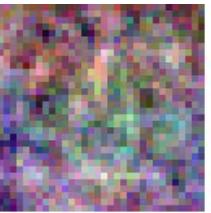
airplane



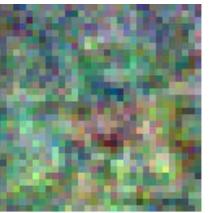
automobile



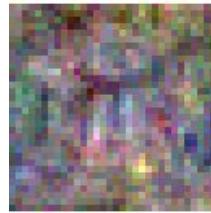
bird



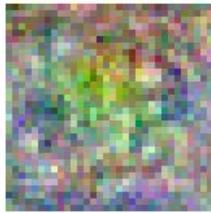
cat



deer



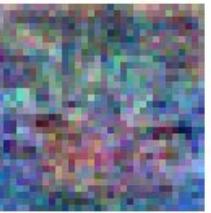
dog



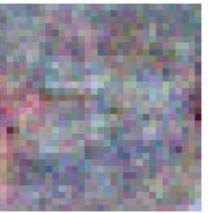
frog



horse



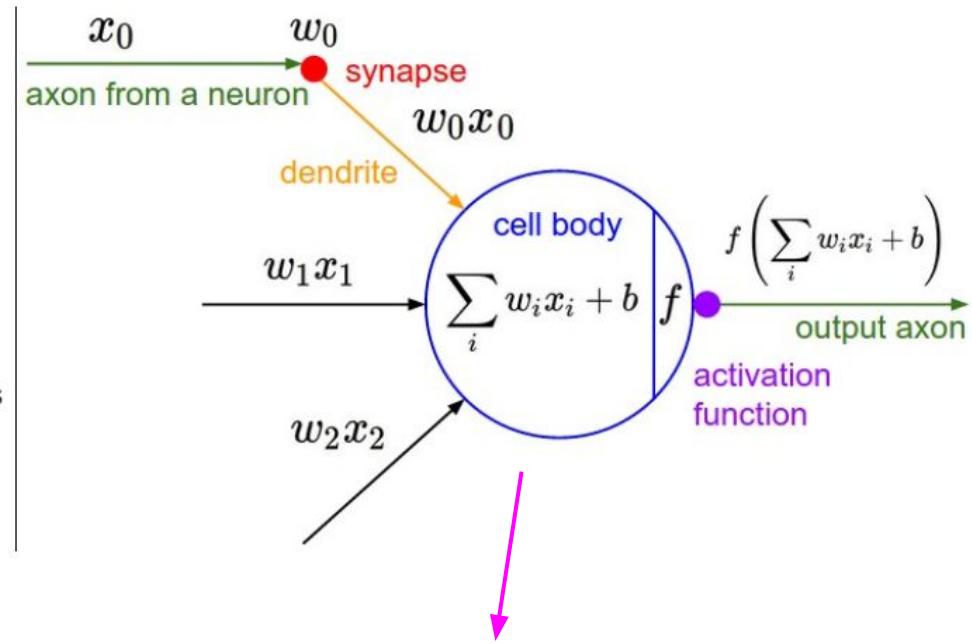
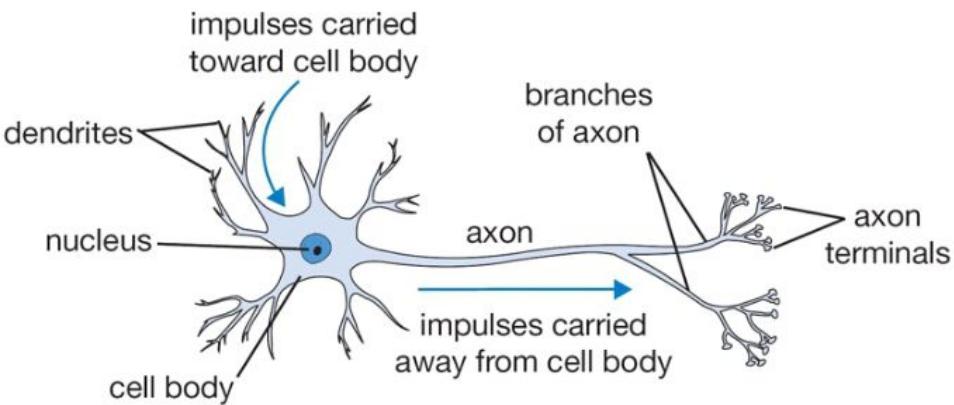
ship



truck

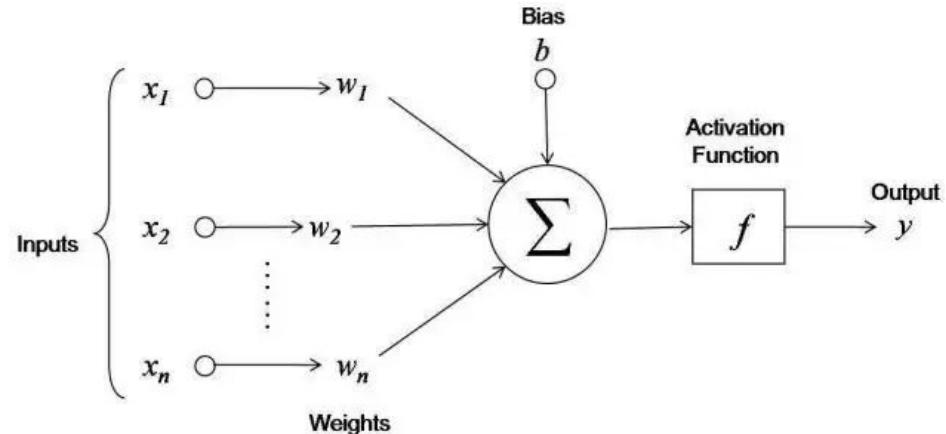
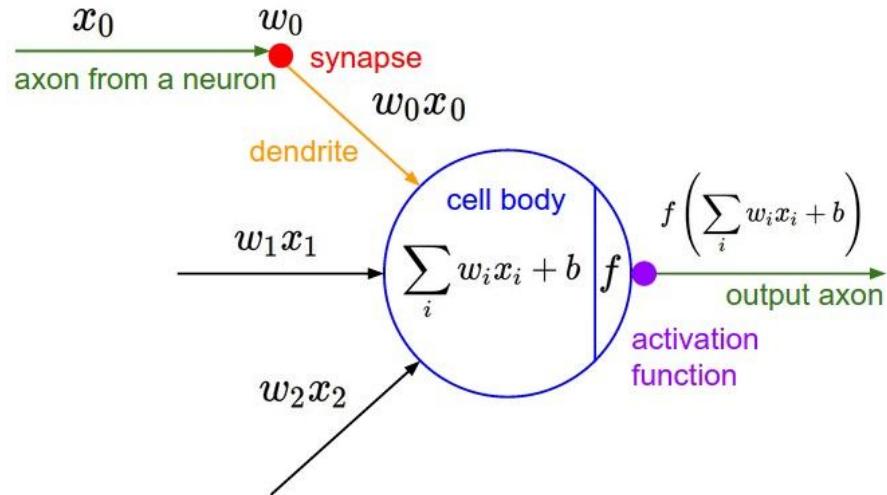
[CIFAR dataset](#)

Biological Neuron vs Artificial Neuron

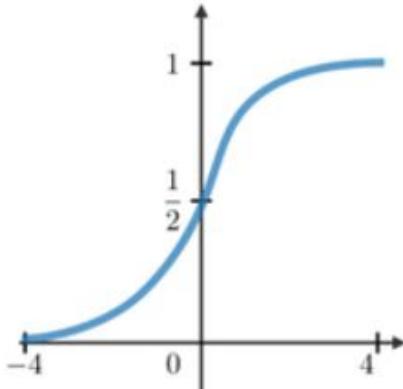
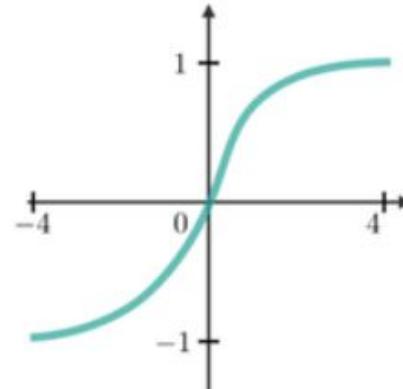
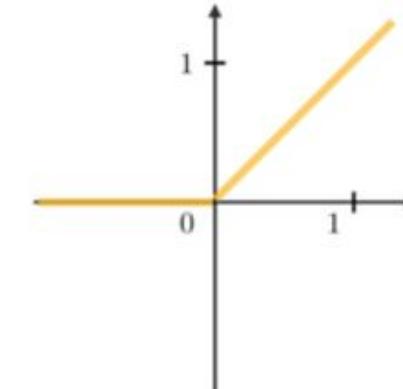
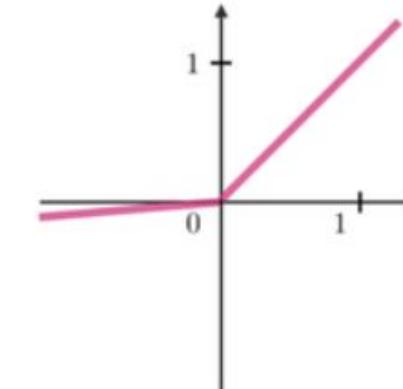


Neuron = linear classifier + activation function

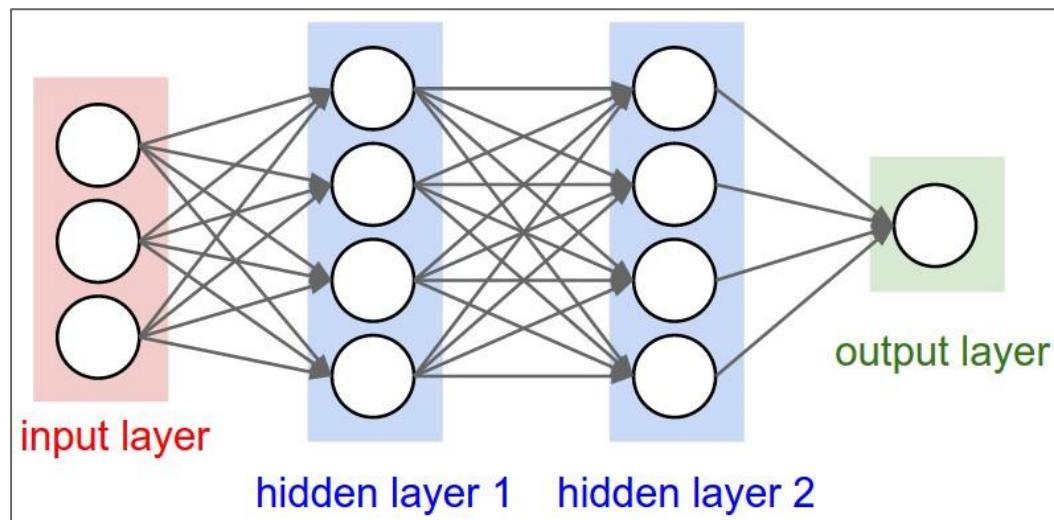
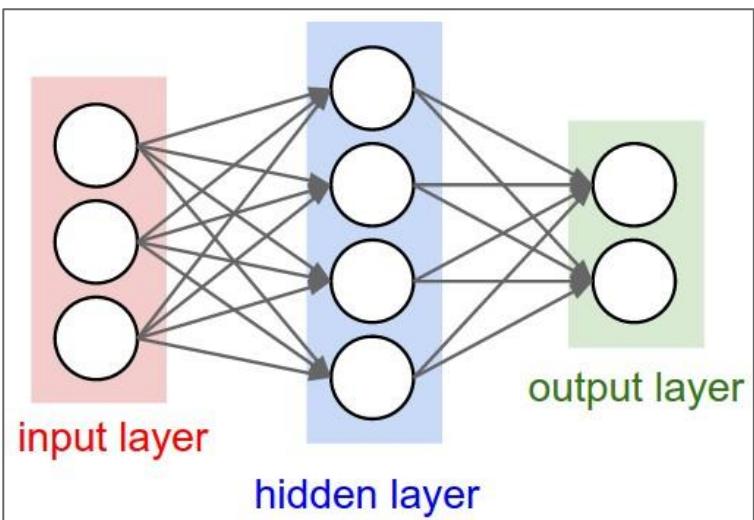
From linear classifier to neuron



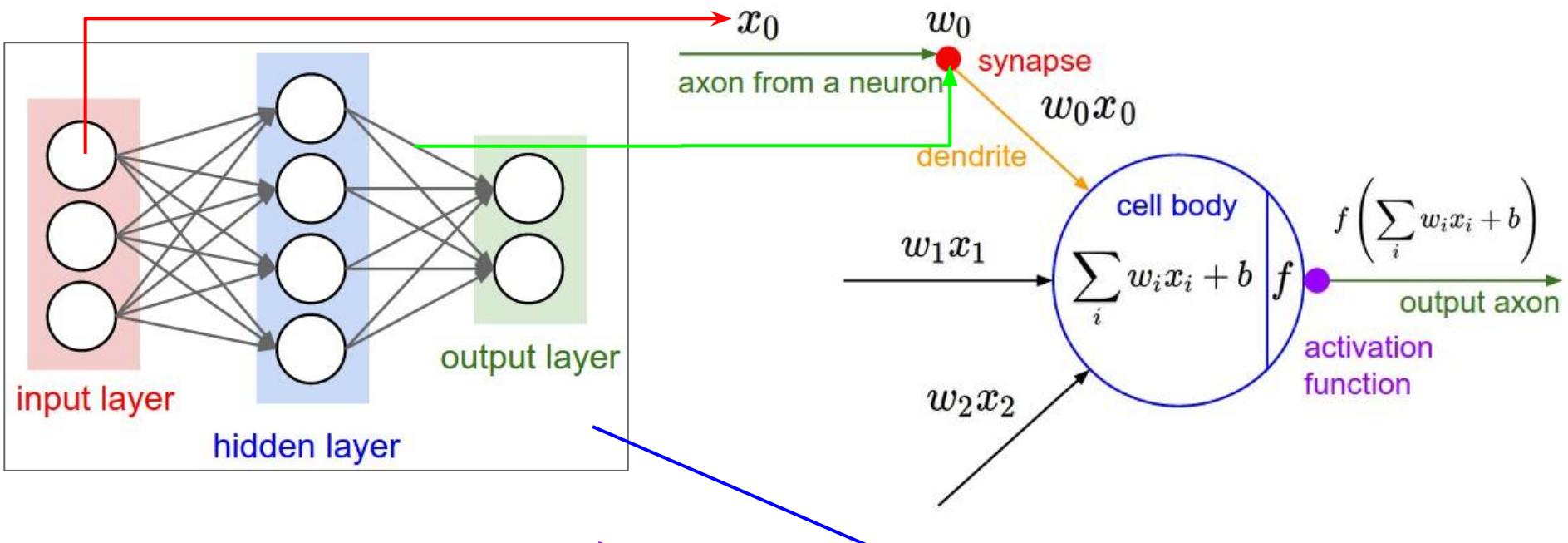
Activation function

Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1 + e^{-z}}$ 	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ 	$g(z) = \max(0, z)$ 	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$ 

Neural Network Architecture



Neural Network

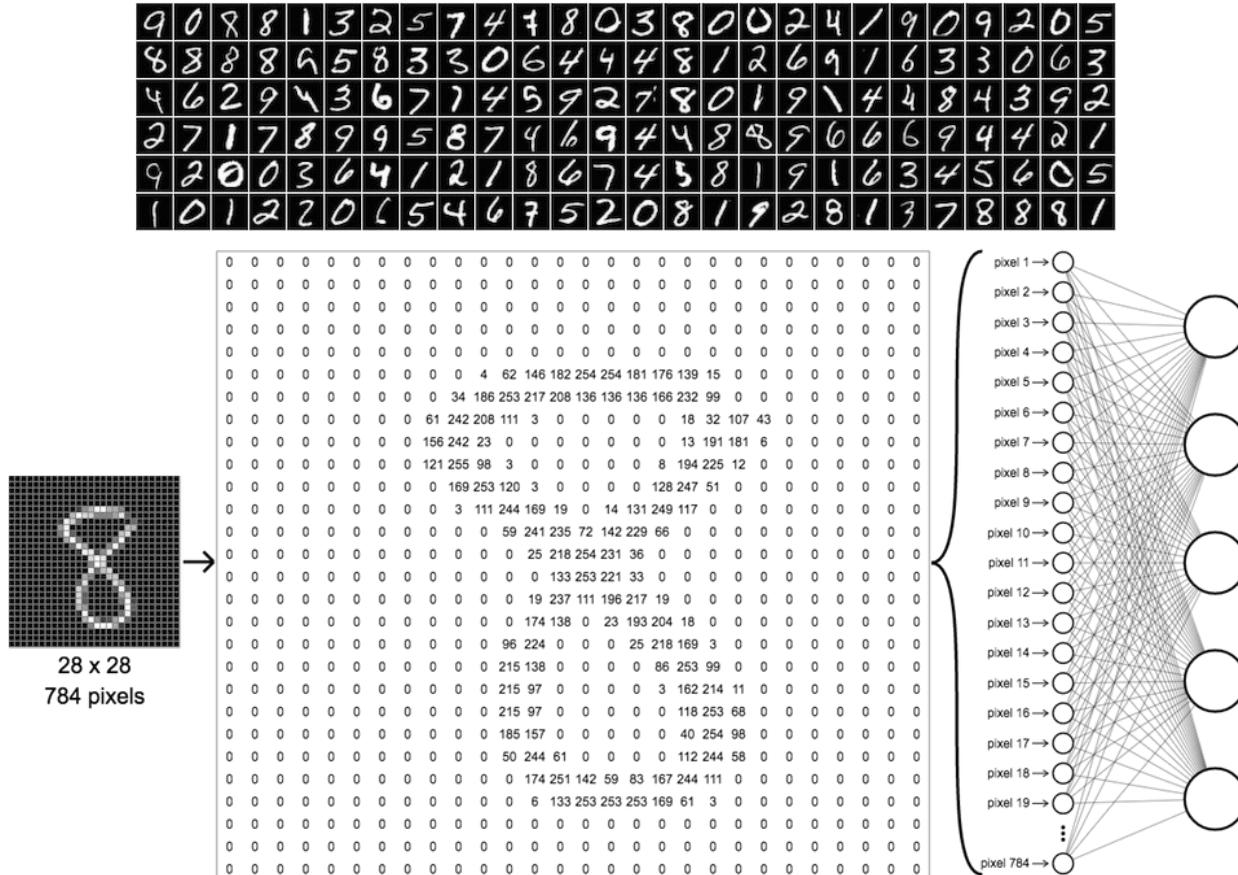


Forward propagation
Forward pass

Feedforward neural network

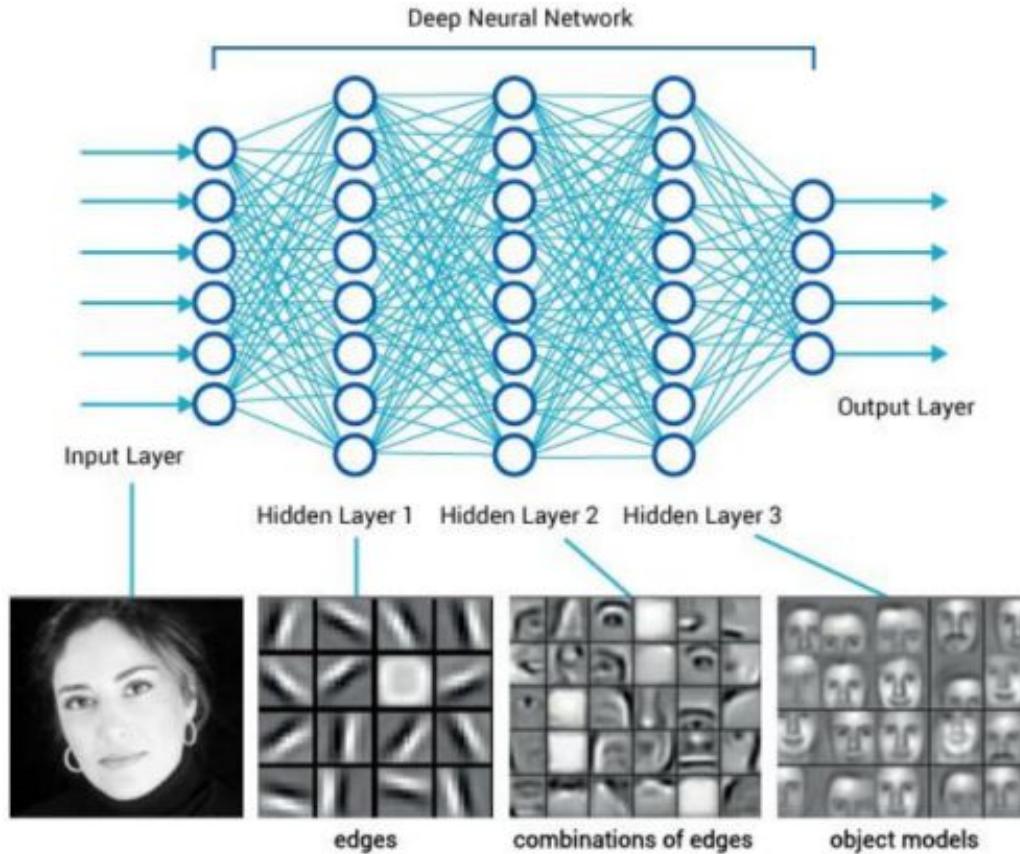
Demo

Complete example of Neural Network



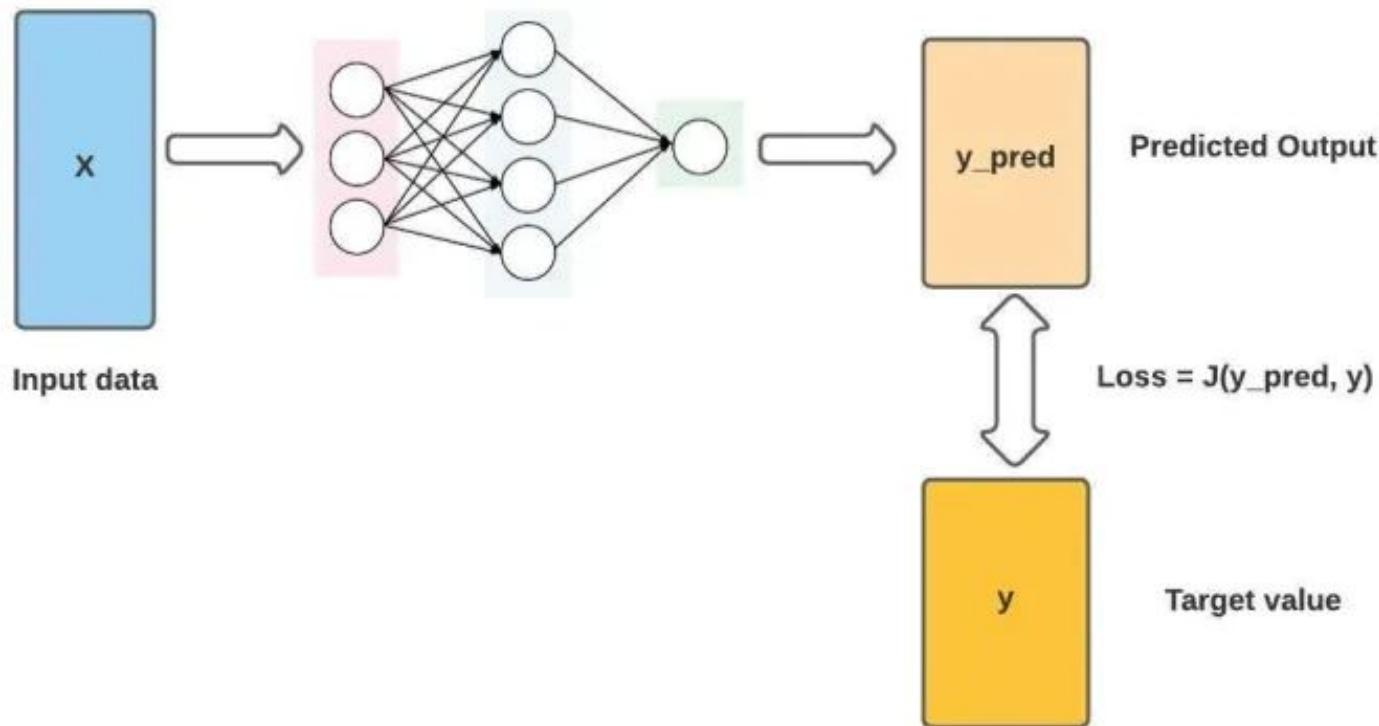
Demo

Deep Neural Network



Demo

Loss function



Loss function

Regression

Mean Absolute Error Loss

$$MAE = \frac{1}{n} \sum_{i=1}^n |y^{(i)} - \hat{y}^{(i)}|$$

Mean Squared Error Loss

$$MSE = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2$$

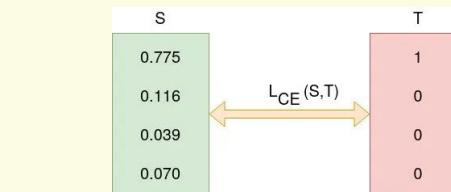
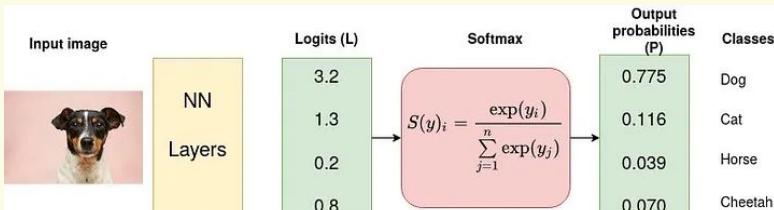
Huber Loss

$$Huber = \frac{1}{n} \sum_{i=1}^n \frac{1}{2} (y_i - \hat{y}_i)^2 \quad |y_i - \hat{y}_i| \leq \delta$$

$$Huber = \frac{1}{n} \sum_{i=1}^n \delta \left(|y_i - \hat{y}_i| - \frac{1}{2} \delta \right) \quad |y_i - \hat{y}_i| > \delta$$

Classification

Cross-Entropy Loss



$$L_{CE} = - \sum_{i=1}^n t_i \log(p_i), \text{ for } n \text{ classes,}$$

where t_i is the truth label and p_i is the Softmax probability for the i^{th} class.

Loss function

Mean Absolute Error Loss

```
import torch
import torch.nn as nn

input = torch.randn(3, 5, requires_grad=True)
target = torch.randn(3, 5)

mae_loss = nn.L1Loss()
output = mae_loss(input, target)
output.backward()

print('input: ', input)
print('target: ', target)
print('output: ', output)
```

```
||||||||||||||||| OUTPUT |||||||||||||||||||||
```

```
input:  tensor([[ 0.2423,  2.0117, -0.0648, -0.0672, -0.1567],
               [-0.2198, -1.4090,  1.3972, -0.7907, -1.0242],
               [ 0.6674, -0.2657, -0.9298,  1.0873,  1.6587]], requires_grad=True)
target:  tensor([[[-0.7271, -0.6048,  1.7069, -1.5939,  0.1023],
                 [-0.7733, -0.7241,  0.3062,  0.9830,  0.4515],
                 [-0.4787,  1.3675, -0.7110,  2.0257, -0.9578]]])
output:  tensor(1.2850, grad_fn=<L1LossBackward>)
```

Loss function

Mean Squared Error Loss

```
import torch
import torch.nn as nn

input = torch.randn(3, 5, requires_grad=True)
target = torch.randn(3, 5)
mse_loss = nn.MSELoss()
output = mse_loss(input, target)
output.backward()

print('input: ', input)
print('target: ', target)
print('output: ', output)
```

||||||||||||||||| OUTPUT |||||||||||||||||||||

```
input:  tensor([[ 0.3177,   1.1312,  -0.8966,  -0.0772,   2.2488],
               [ 0.2391,   0.1840,  -1.2232,   0.2017,   0.9083],
               [-0.0057,  -3.0228,   0.0529,   0.4084,  -0.0084]], requires_grad=True)
target:  tensor([[ 0.2767,   0.0823,   1.0074,   0.6112,  -0.1848],
                [ 2.6384,  -1.4199,   1.2608,   1.8084,   0.6511],
                [ 0.2333,  -0.9921,   1.5340,   0.3703,  -0.5324]])
output:  tensor(2.3280, grad_fn=<MselossBackward>)
```

Loss function

Cross-Entropy Loss

```
import torch
import torch.nn as nn

input = torch.randn(3, 5, requires_grad=True)
target = torch.empty(3, dtype=torch.long).random_(5)

cross_entropy_loss = nn.CrossEntropyLoss()
output = cross_entropy_loss(input, target)
output.backward()

print('input: ', input)
print('target: ', target)
print('output: ', output)
```

```
||||||| OUTPUT |||||  
  
input:  tensor([[ 0.1639, -1.2095,  0.0496,  1.1746,  0.9474],  
                [ 1.0429,  1.3255, -1.2967,  0.2183,  0.3562],  
                [-0.1680,  0.2891,  1.9272,  2.2542,  0.1844]], requires_grad=True)  
target:  tensor([4, 0, 3])  
output:  tensor(1.0393, grad_fn=<NllLossBackward>)
```

Gradient

$f(x,y)$

$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{bmatrix}^T = \text{gradient}$$
$$= \frac{\partial f}{\partial x} \vec{i} + \frac{\partial f}{\partial y} \vec{j} \quad (\text{in two dimensions})$$

The **gradient** of a function represents the rate of change of a function, it is defined as

$$\nabla f(x,y) = \langle f_x, f_y \rangle = \frac{\partial f}{\partial x} \vec{i} + \frac{\partial f}{\partial y} \vec{j}$$

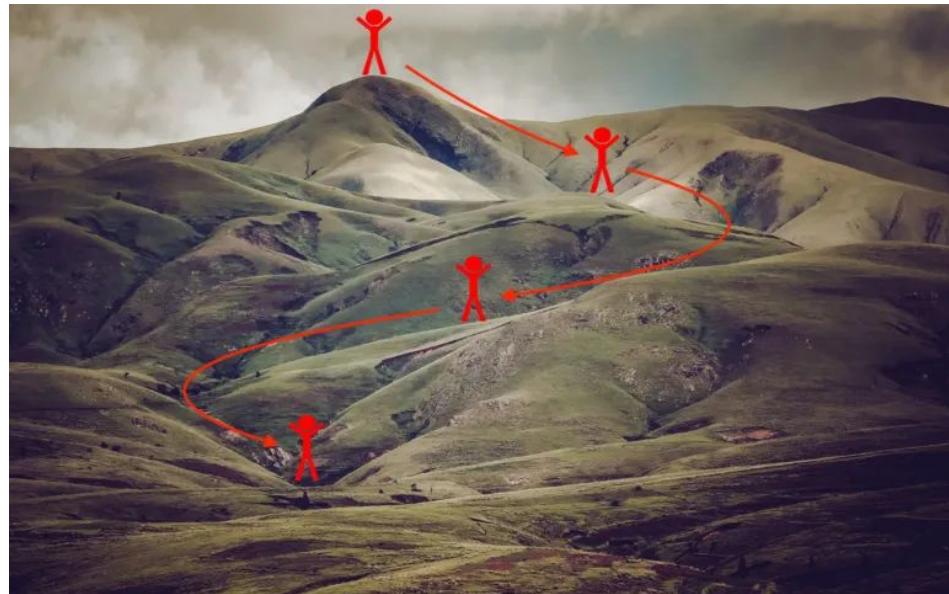
and

$$\nabla f(x,y,z) = \langle f_x, f_y, f_z \rangle = \frac{\partial f}{\partial x} \vec{i} + \frac{\partial f}{\partial y} \vec{j} + \frac{\partial f}{\partial z} \vec{k}$$

- It is a vector!
- The gradient vector points in the **direction of maximum increase**, and the magnitude of this vector represents the **maximum rate of change**.

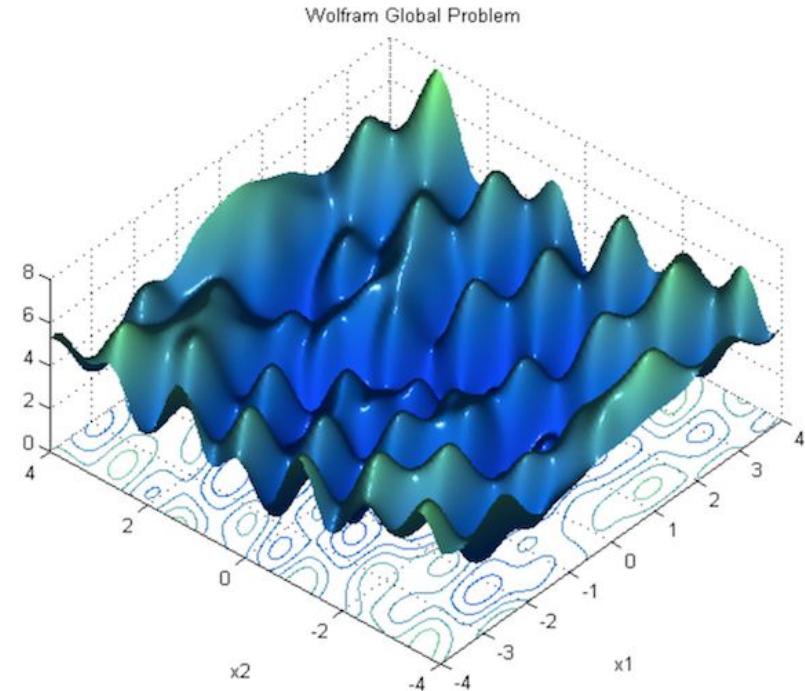
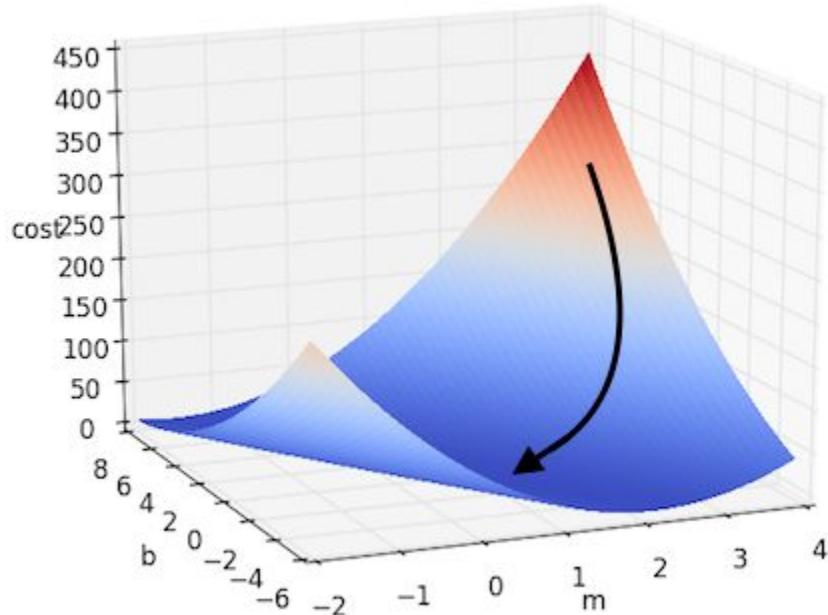
Optimization function

Gradient Descent



Optimization function

Gradient Descent in Linear Classifier vs Neural network



Gradient Descent: Different versions

Batch Gradient Descent

Tất cả datapoints
được đưa vào mô hình
cùng 1 lúc để tính
gradient

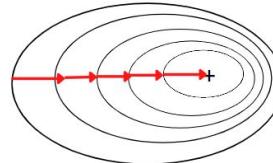
Stochastic Gradient Descent

Từng datapoint một sẽ
được đưa vào mô hình để
tính gradient

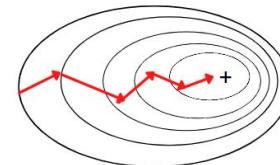
(Stochastic) Mini-batch Gradient Descent

N datapoint sẽ được đưa
vào mô hình cùng lúc để
tính gradient

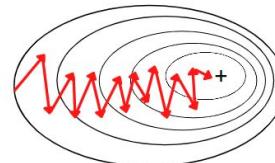
Batch Gradient Descent



Mini-Batch Gradient Descent



Stochastic Gradient Descent



How parameters are updated?

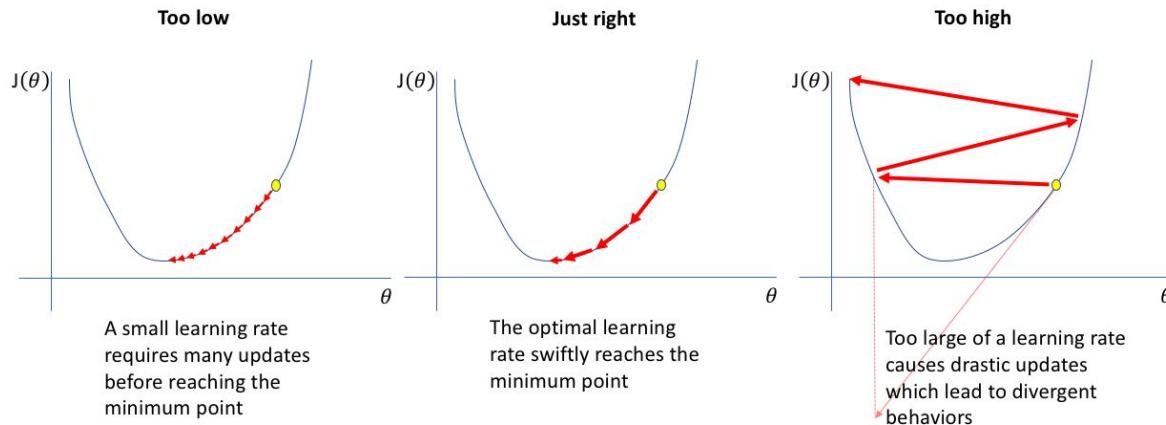
$$W_{\text{new}} = W_{\text{old}} - \eta \frac{\partial L}{\partial W_{\text{old}}}$$

W_{new} Updated weight

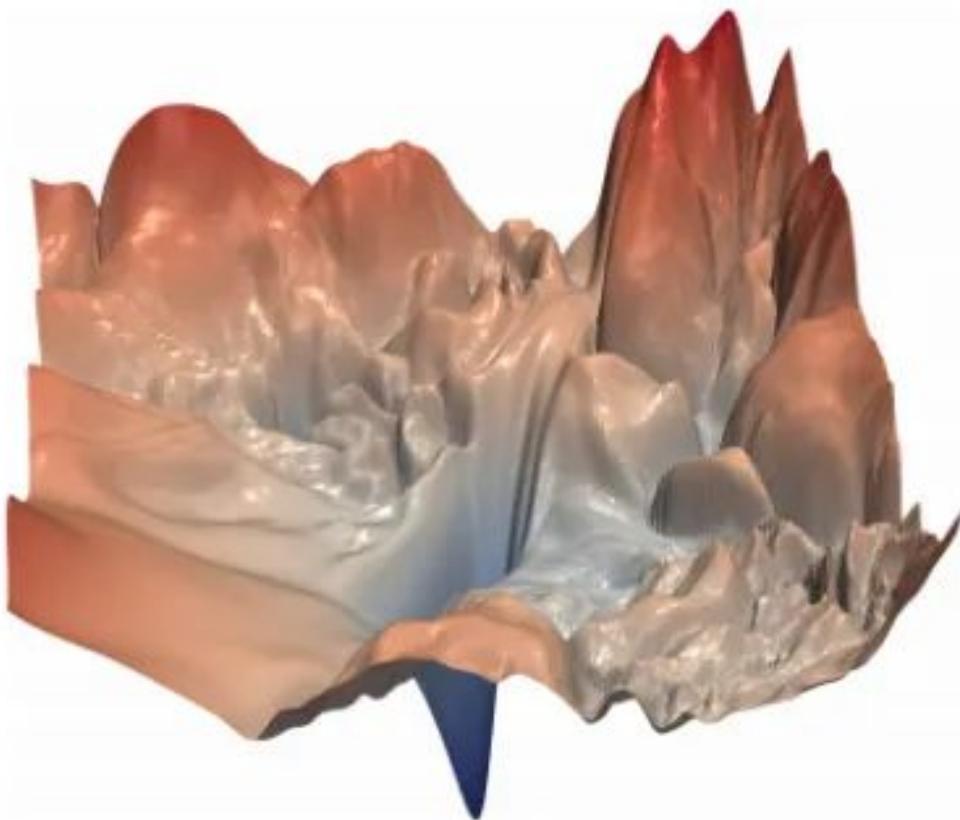
W_{old} Old weight

η Learning rate ($\eta > 0$)

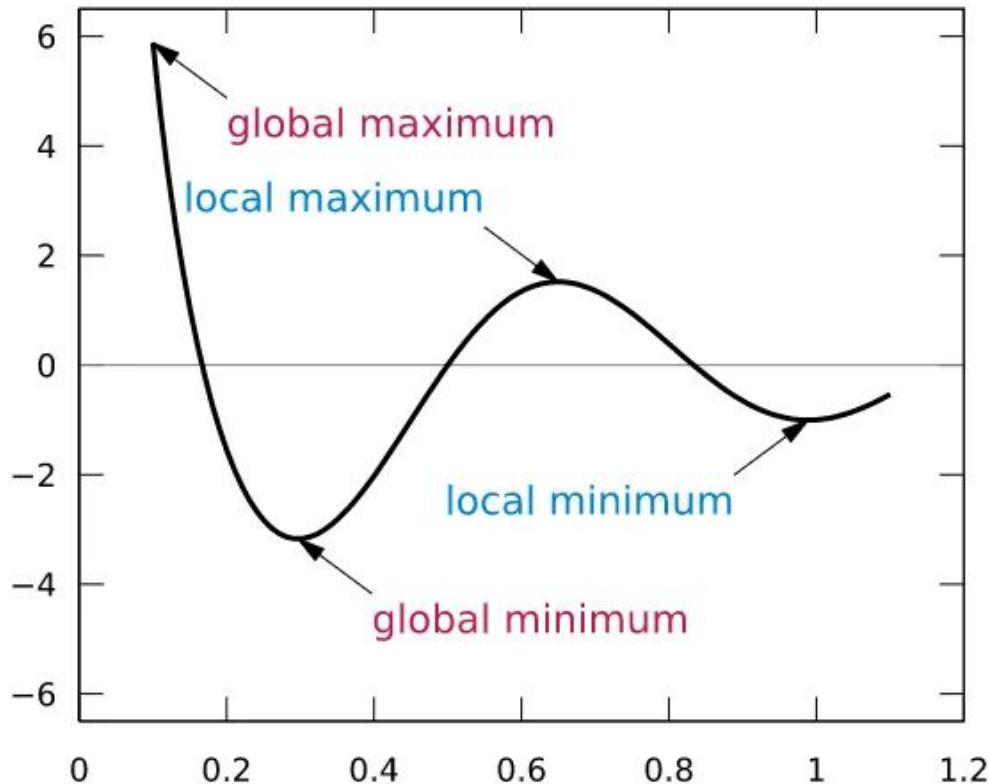
$\frac{\partial L}{\partial W_{\text{old}}}$ Partial derivative of L with respect to old weight



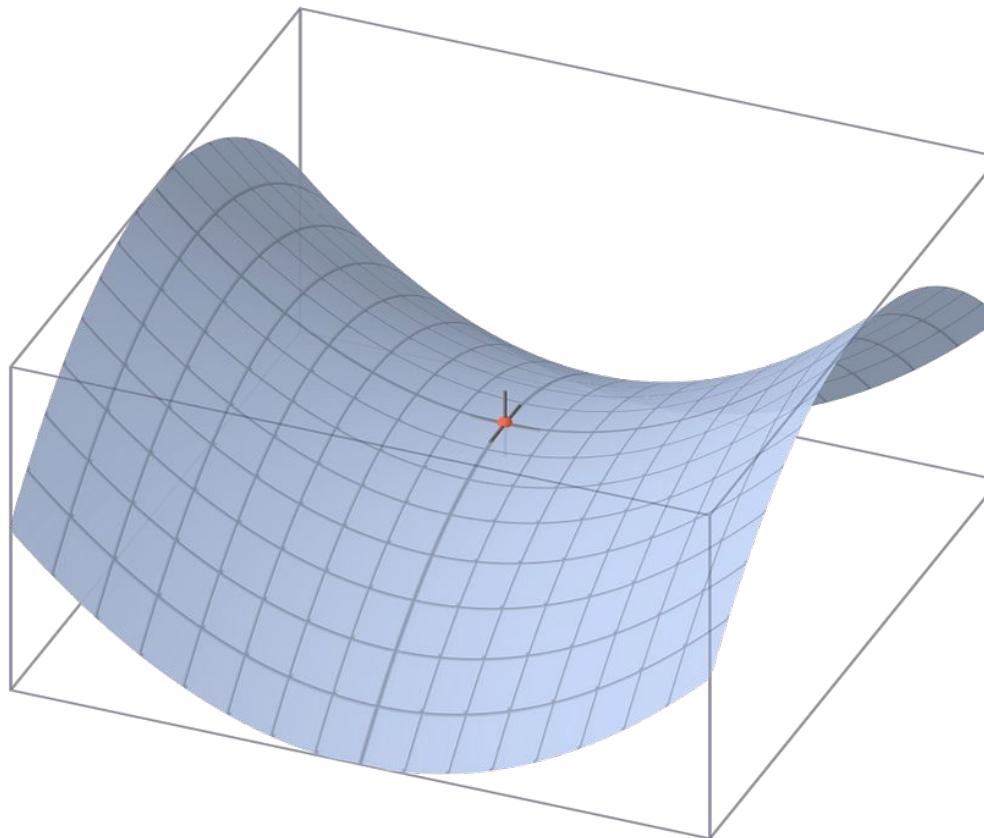
Challenge with Gradient Descent: Loss function



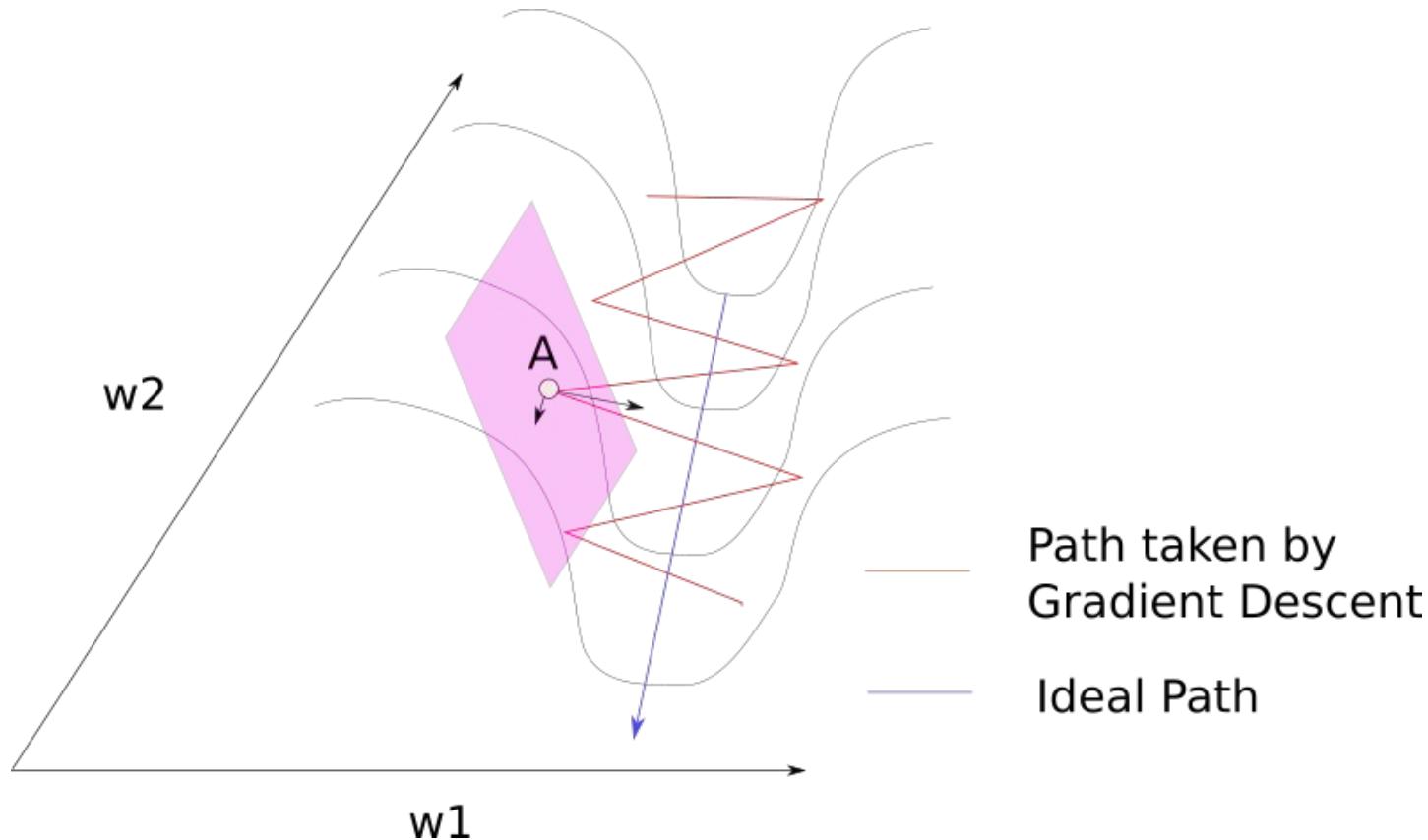
Challenge with Gradient Descent: Local minima



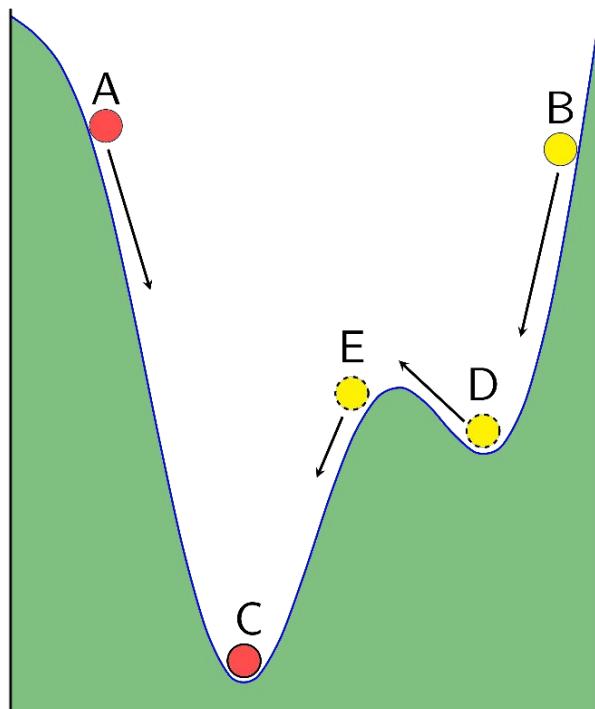
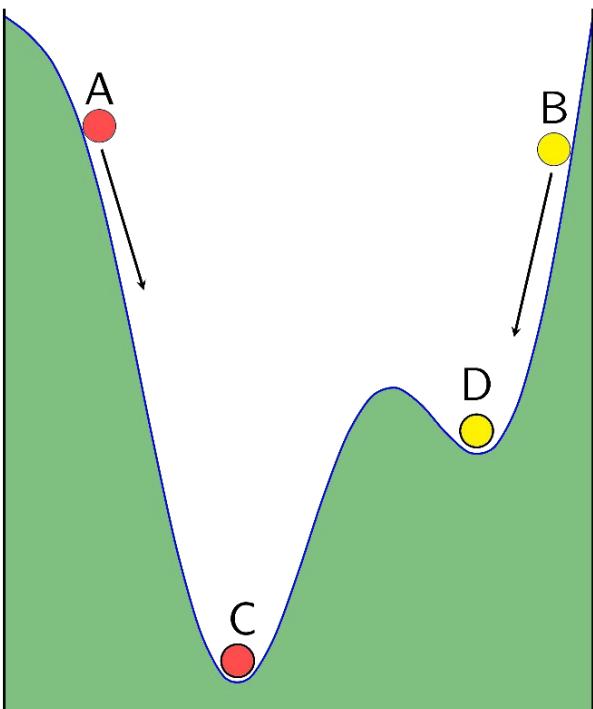
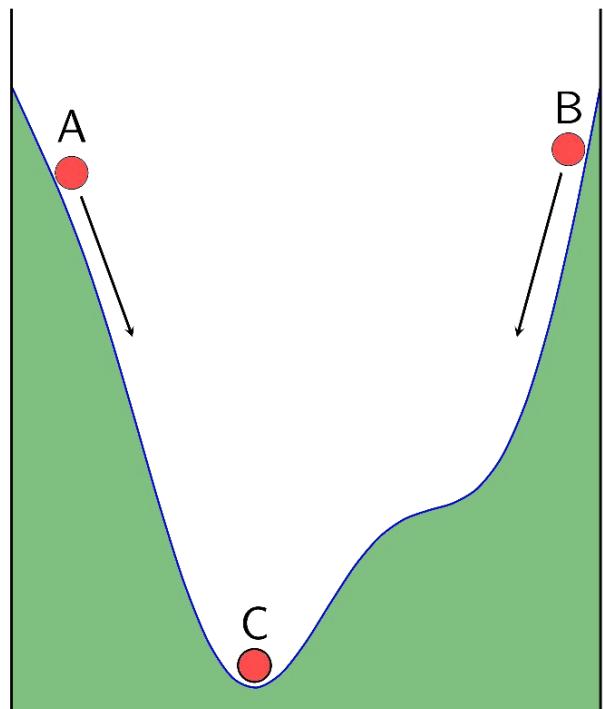
Challenge with Gradient Descent: Saddle points



Challenge with Gradient Descent: Ravines



Momentum



Momentum

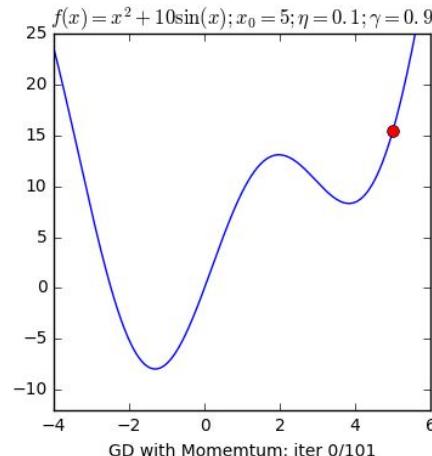
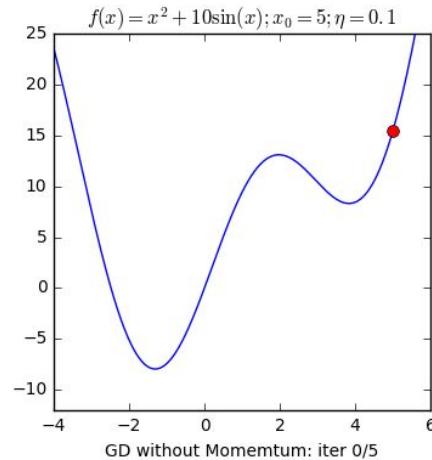
Gradient Descent Update Rule

$$w_{t+1} = w_t - \eta \nabla w_t$$

Momentum based Gradient Descent Update Rule

$$v_t = \gamma * v_{t-1} + \eta \nabla w_t$$

$$w_{t+1} = w_t - v_t$$



Adaptive Gradient Descent (1)

SGD

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$$



$$G_t = \begin{pmatrix} \mathbb{R}^{d \times d} \\ \vdots & \cdots & \vdots & \cdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

Adagrad

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

$$g_{t,i} = \nabla_{\theta} J(\theta_i)$$

Vectorize

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t.$$

Adaptive Gradient Descent (2)

SGD

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$$

$$G_t = \begin{pmatrix} \vdots & \cdots & \vdots & \cdots \\ \text{---} & \text{---} & \text{---} & \text{---} \\ \text{---} & \text{---} & \text{---} & \text{---} \\ \vdots & \cdots & \vdots & \cdots \\ \text{---} & \text{---} & \text{---} & \text{---} \\ \text{---} & \text{---} & \text{---} & \text{---} \end{pmatrix} \in \mathbb{R}^{d \times d}$$



Adagrad modifies the general learning rate η based on the past gradients that have been computed for θ_i

Adagrad

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

$$g_{t,i} = \nabla_{\theta} J(\theta_i)$$

Vectorize

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t.$$

Adaptive Gradient Descent (3)

SGD

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$$

$$G_t = \begin{pmatrix} \mathbb{R}^{d \times d} \\ \vdots & \cdots & \vdots & \cdots \end{pmatrix}$$



G_t is a diagonal matrix where each diagonal element (i,i) is the sum of the squares of the gradients θ_i up to time step t .

Adagrad

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

$$g_{t,i} = \nabla_{\theta} J(\theta_i)$$

Vectorize

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t.$$

Adaptive Gradient Descent (4)

SGD

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$$

$$G_t = \begin{pmatrix} & \cdots & \cdots & \cdots \\ \vdots & \ddots & \ddots & \vdots \\ & \cdots & \cdots & \ddots \\ \vdots & \ddots & \ddots & \vdots \\ & \cdots & \cdots & \ddots \end{pmatrix} \in \mathbb{R}^{d \times d}$$



ϵ is a smoothing term that avoids division by zero (usually on the order of $1e - 8$).

Adagrad

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

$$g_{t,i} = \nabla_{\theta} J(\theta_i)$$

Vectorize

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t.$$

Adaptive Gradient Descent (4)

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \varepsilon}} \cdot g_t$$

G_t is sum of the squares of the past gradients w.r.t. to all parameters θ

Root Mean Square Propagation

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{(1 - \gamma)g_{t-1}^2 + \gamma g_t^2 + \varepsilon}} \cdot g_t$$

BackPropagation

Chain Rule

If f and g are both differentiable and $F(x)$ is the composite function defined by $F(x) = f(g(x))$ then F is differentiable and F' is given by the product

$$F'(x) = f'(g(x)) g'(x)$$

Differentiate
outer function

Differentiate
inner function

$$\frac{da}{dx} a(b(c(d(e(f(g(x)))))))$$

$$\frac{\partial a}{\partial x} = \frac{\partial a}{\partial b} \times \frac{\partial b}{\partial c} \times \frac{\partial c}{\partial d} \times \frac{\partial d}{\partial e} \times \frac{\partial e}{\partial f} \times \frac{\partial f}{\partial g} \times \frac{\partial g}{\partial x}$$

BackPropagation

$$\frac{d}{dx} \left[(4x^3 - 5x^2 + 6x - 7)^{10} \right]$$

$$\frac{d}{dx} \left[\underbrace{(4x^3 - 5x^2 + 6x - 7)^{10}}_{\text{outside function}} \right] = 10(4x^3 - 5x^2 + 6x - 7)^9 \cdot (12x^2 - 10x + 6)$$

Inside function

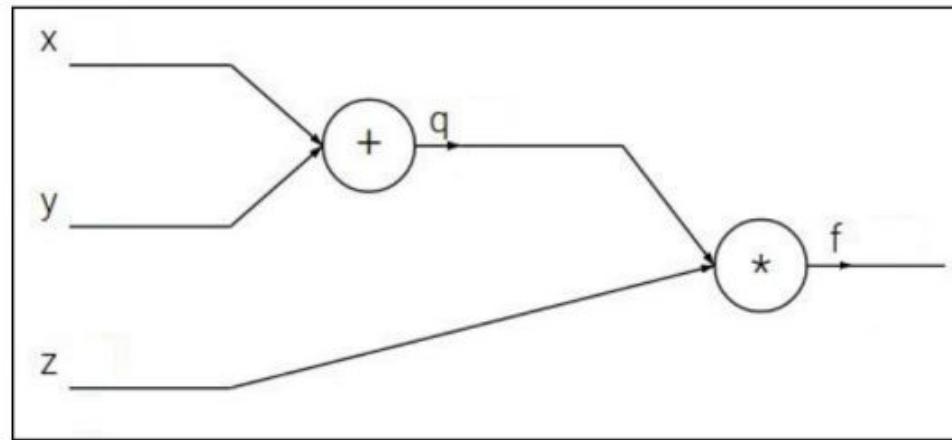
$$\frac{dy}{dx} = 10(12x^2 - 10x + 6)(4x^3 - 5x^2 + 6x - 7)^9$$

$$\frac{dy}{dx} = 20(6x^2 - 5x + 3)(4x^3 - 5x^2 + 6x - 7)^9$$

Backpropagation

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

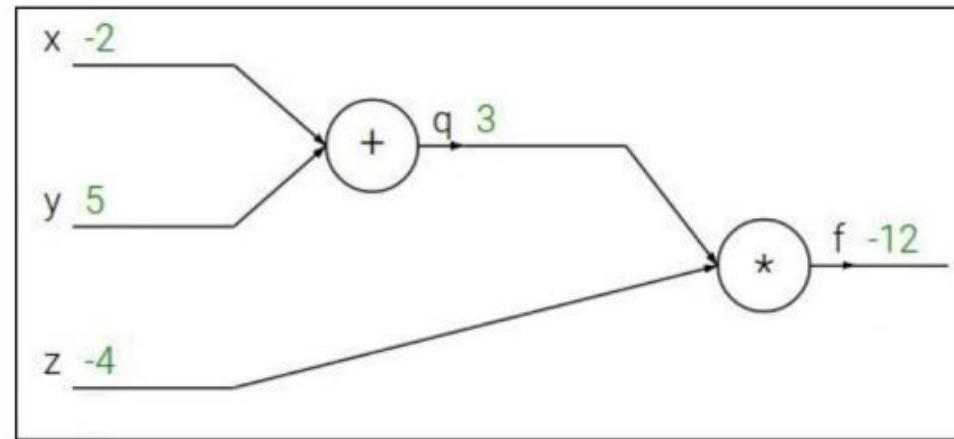


BackPropagation - example

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. $x = -2$, $y = 5$, $z = -4$



forward

backward

BackPropagation - example

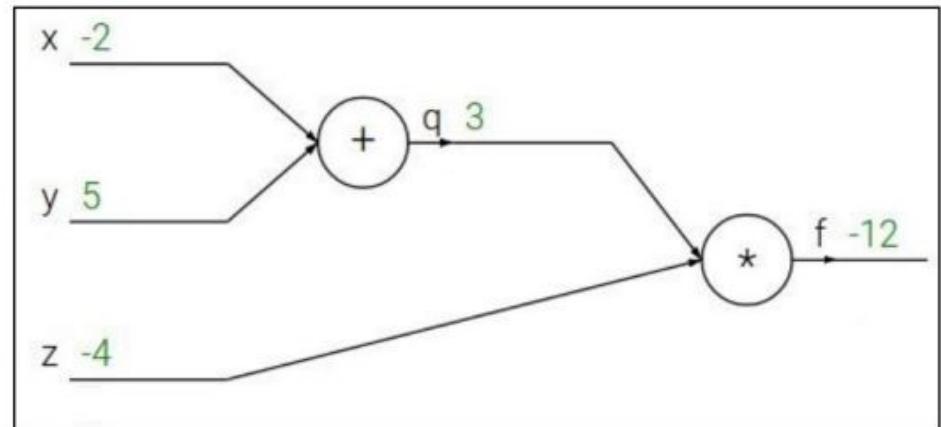
Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. $x = -2$, $y = 5$, $z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



forward

backward

BackPropagation - example

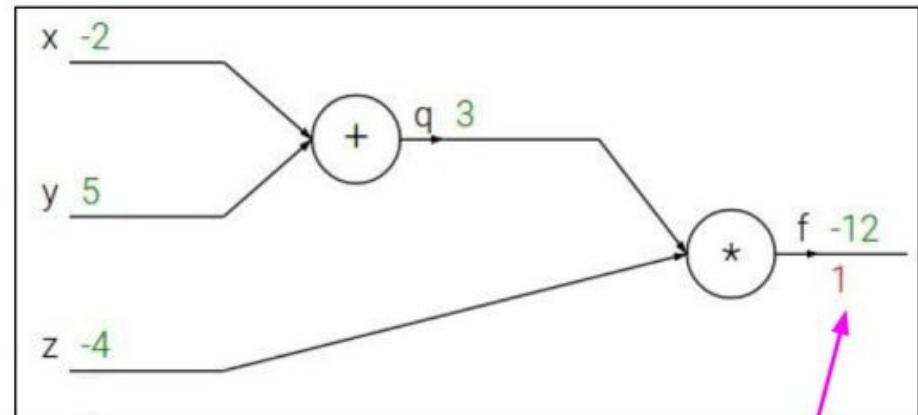
Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. $x = -2$, $y = 5$, $z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



$$\frac{\partial f}{\partial f}$$

forward

backward

BackPropagation - example

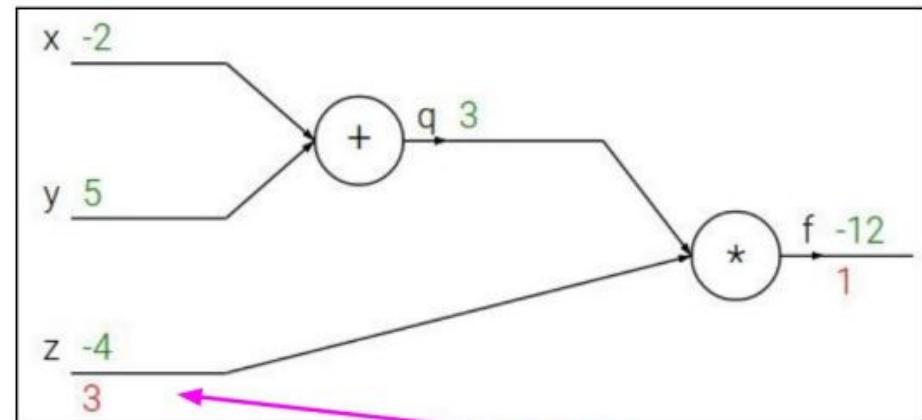
Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



$$\frac{\partial f}{\partial z}$$

forward

backward

BackPropagation - example

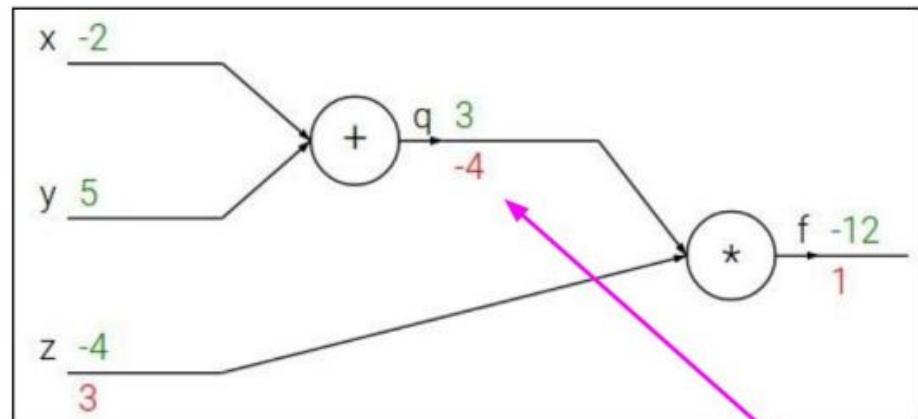
Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. $x = -2$, $y = 5$, $z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



$$\frac{\partial f}{\partial q}$$

forward

backward

BackPropagation - example

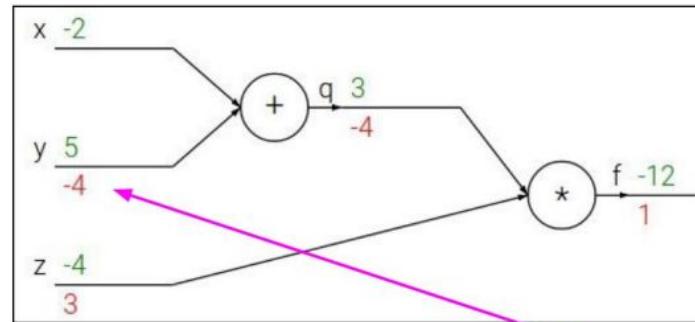
Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. $x = -2$, $y = 5$, $z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

Upstream
gradient

Local
gradient

$$\frac{\partial f}{\partial y}$$

forward

backward

BackPropagation - example

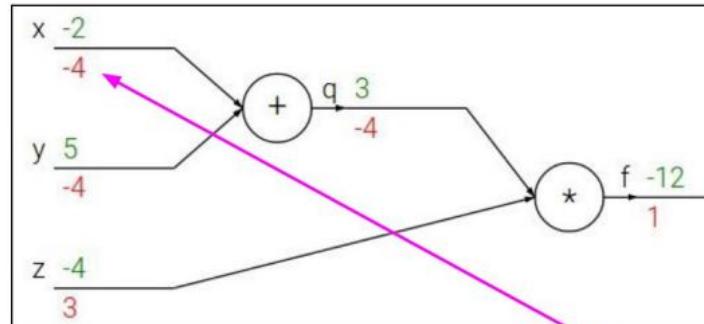
Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



Chain rule:

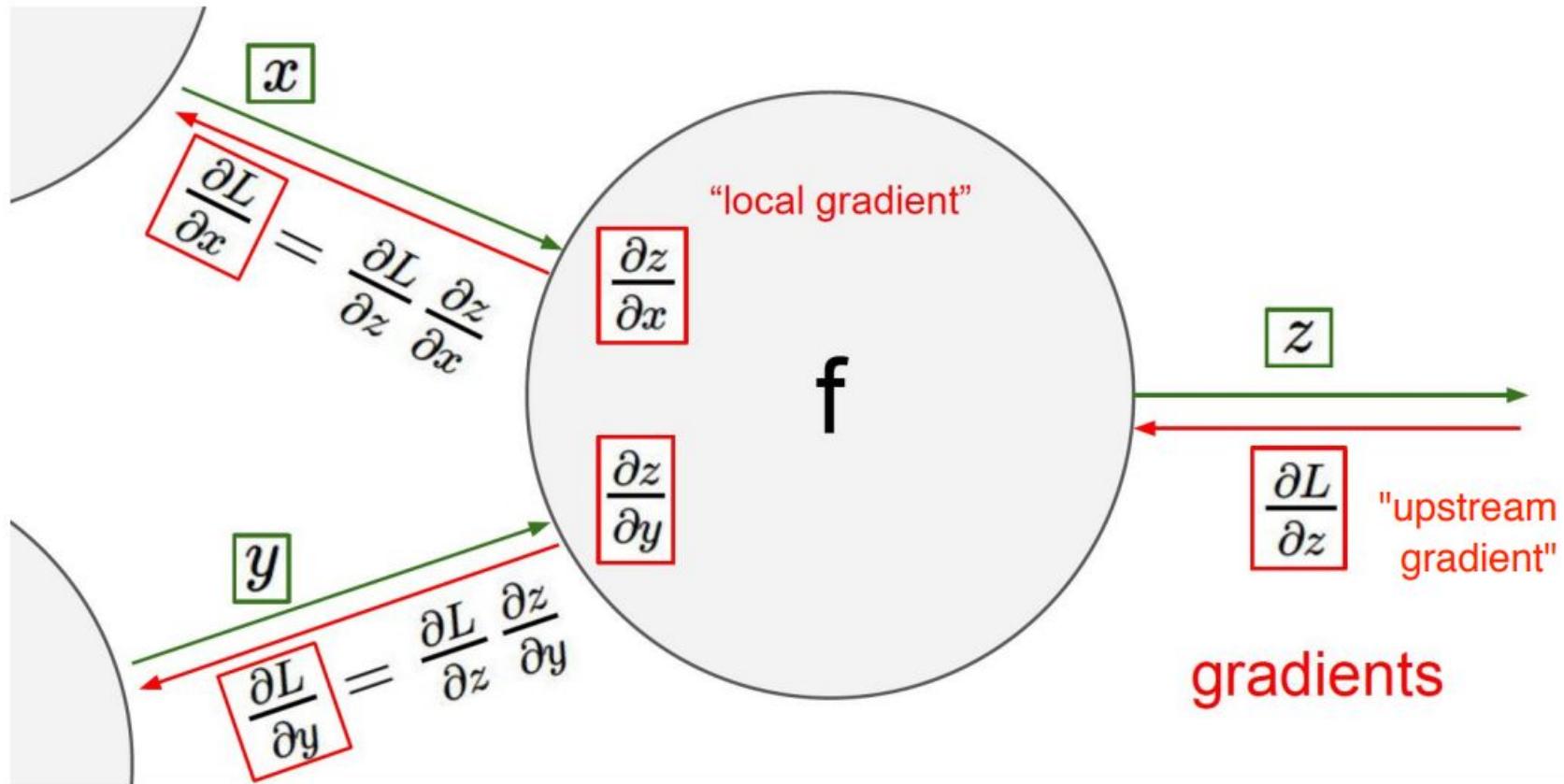
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

Upstream gradient Local gradient

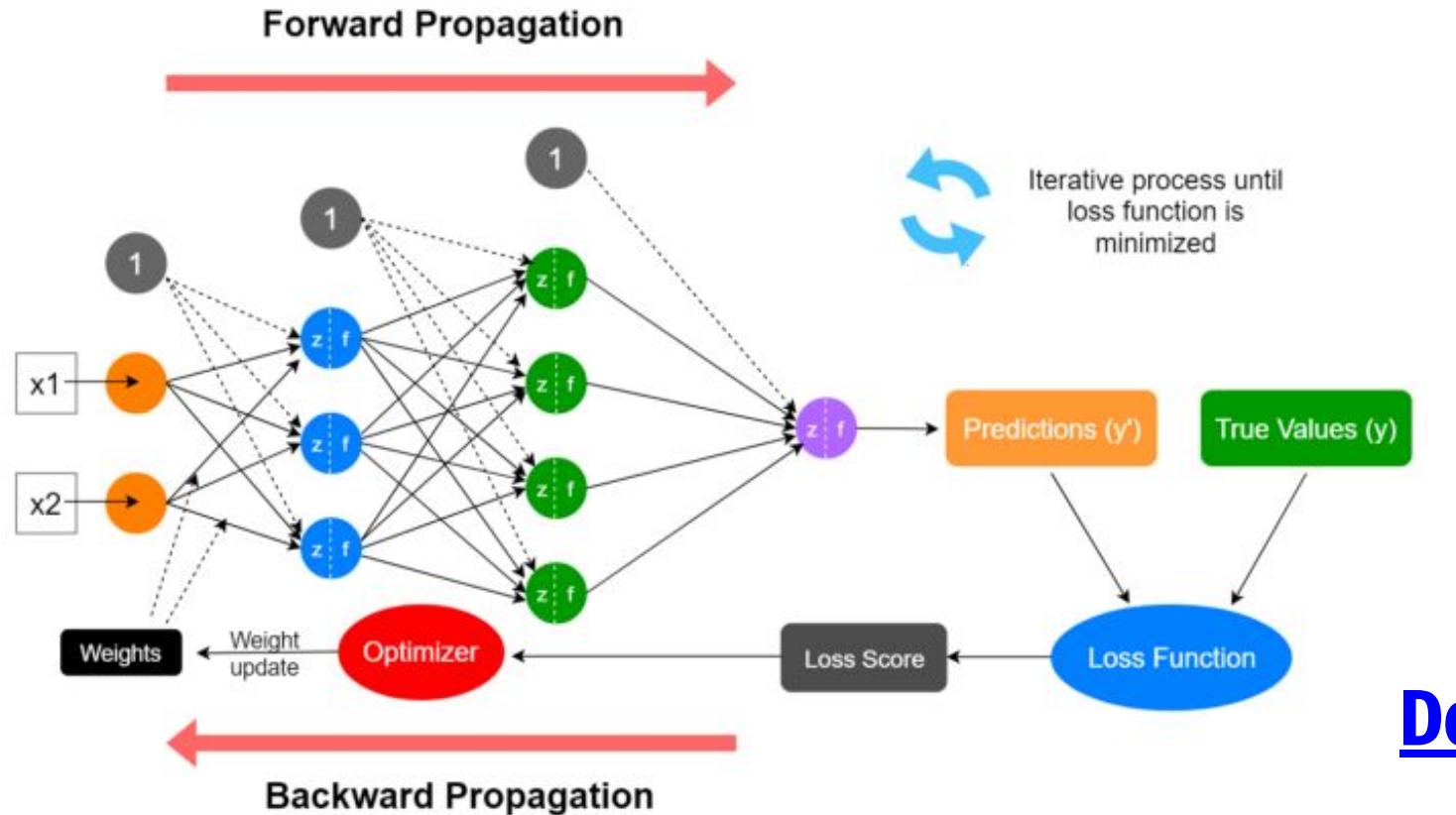
forward

backward

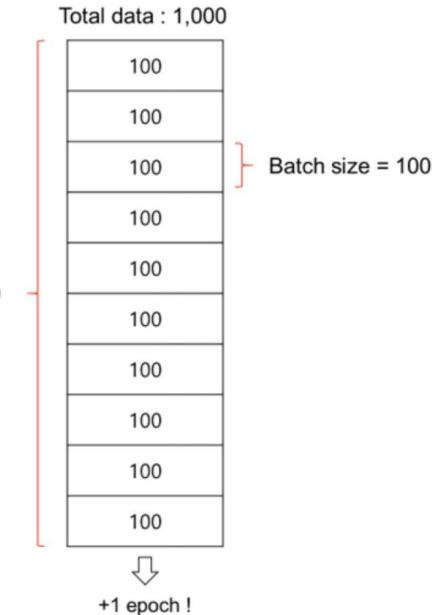
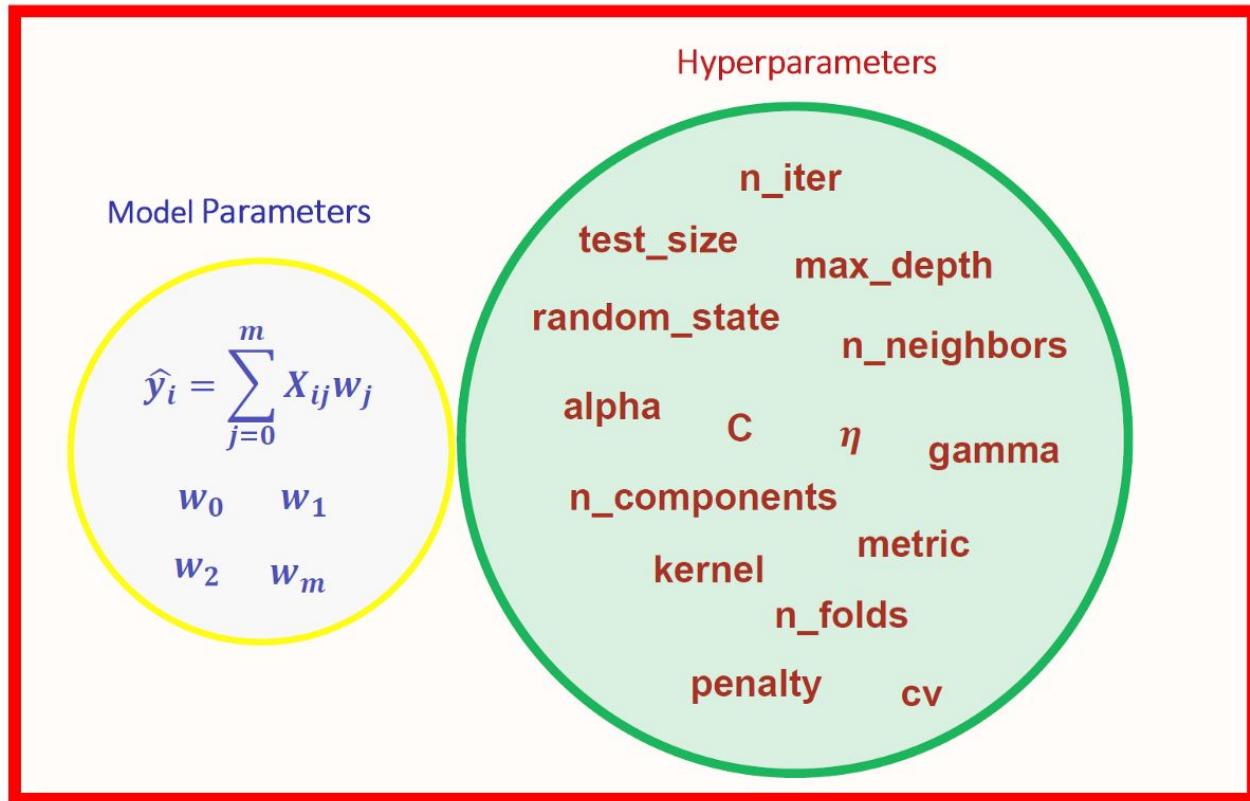
BackPropagation - explanation



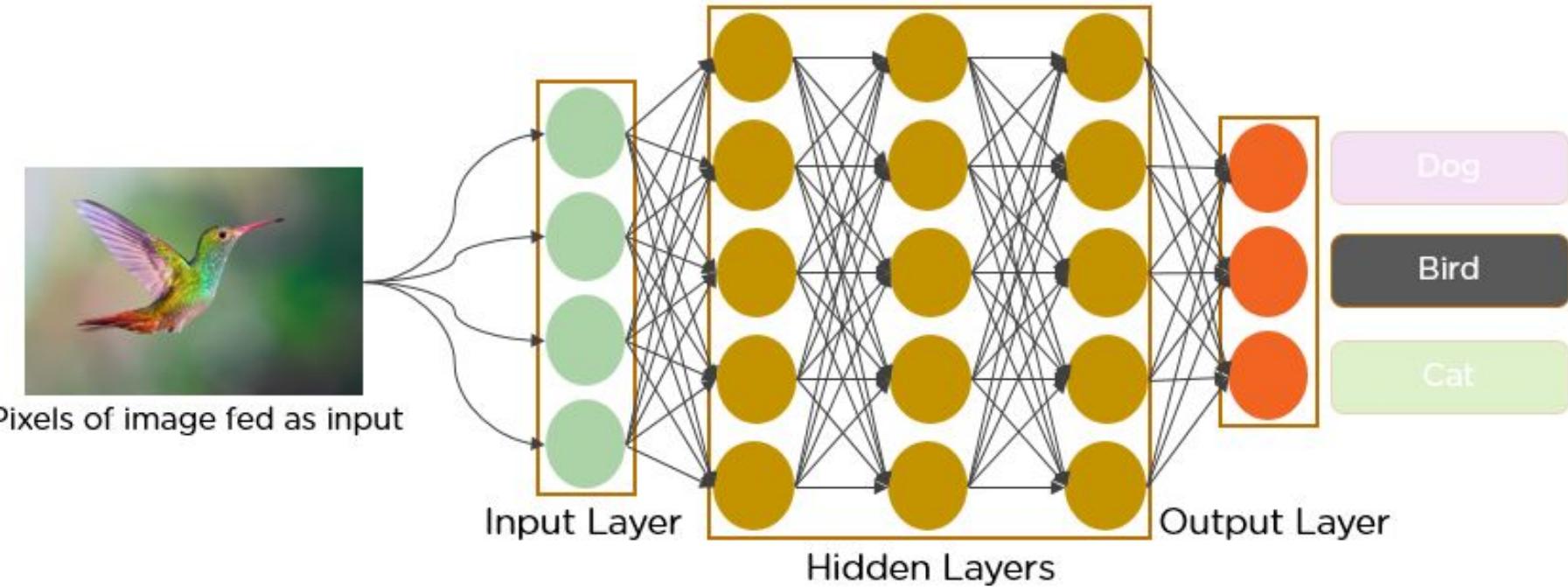
Neural network's learning process



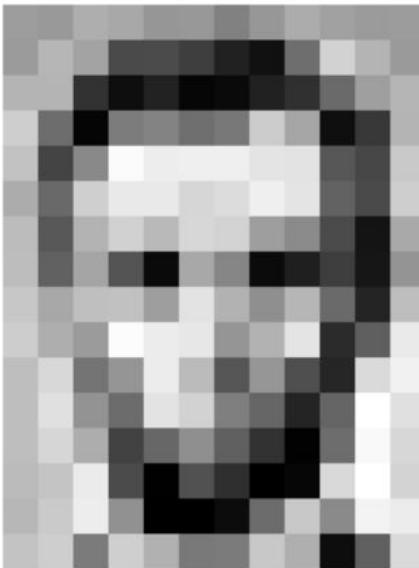
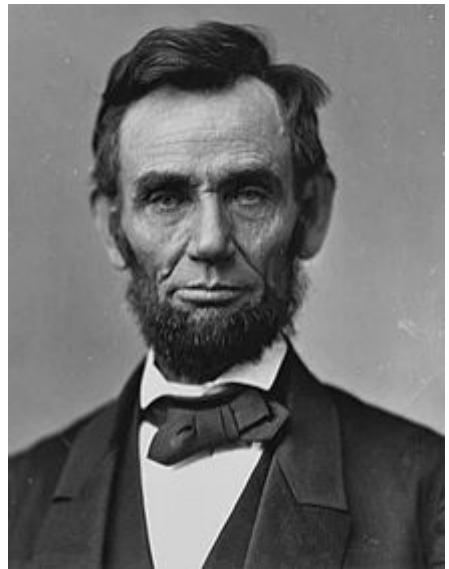
Parameters vs Hyperparameters



Neural Network



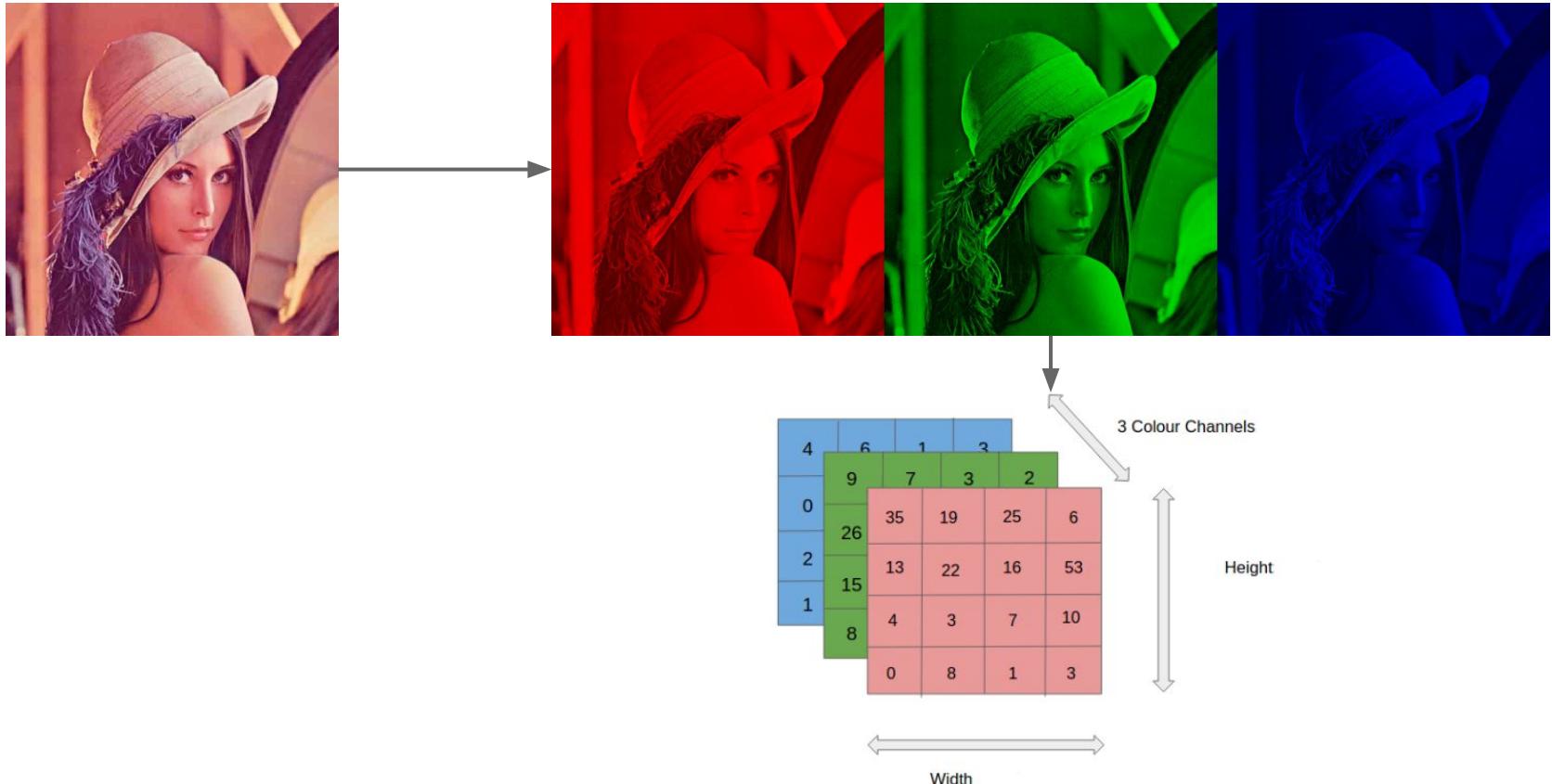
Gray image is represented in Computer Vision



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	299	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	162	105	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
180	224	147	108	237	210	137	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

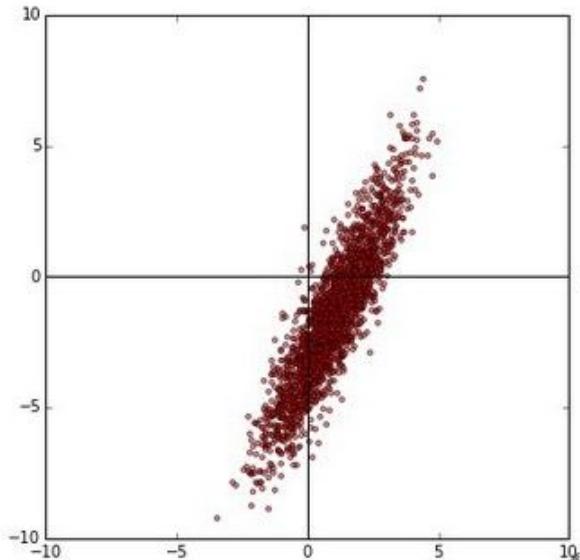
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	299	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	162	105	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
180	224	147	108	237	210	137	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	211
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Color image is represented in Computer Vision

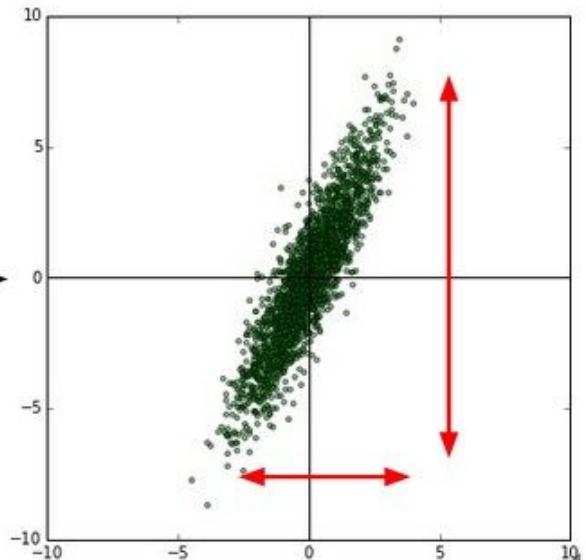


Data preprocessing

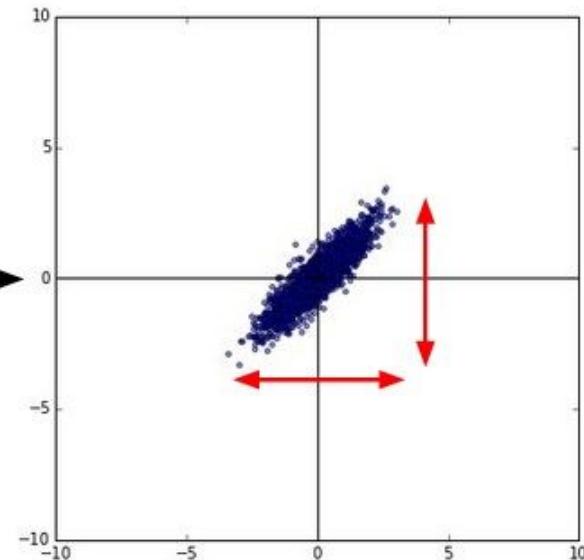
original data



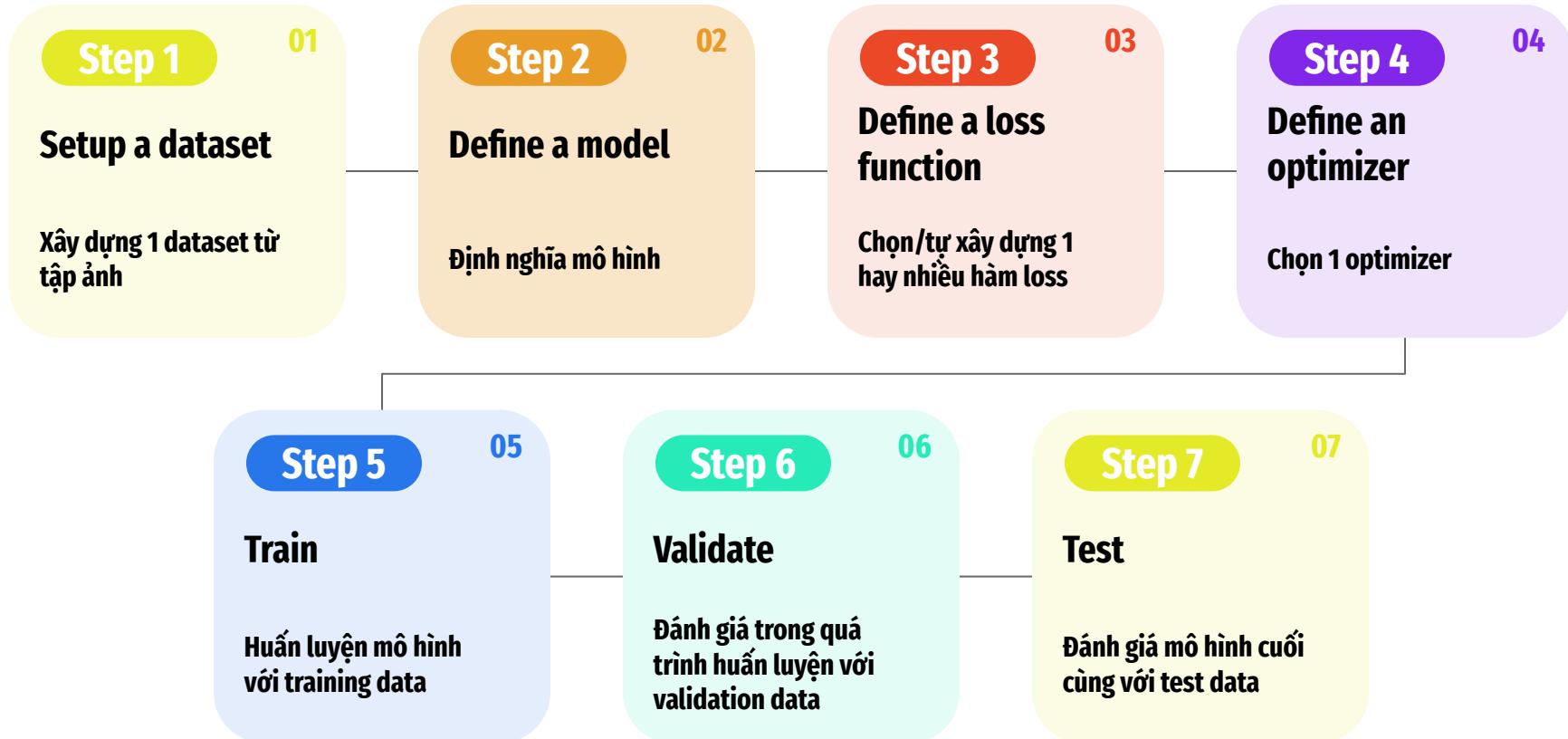
zero-centered data



normalized data

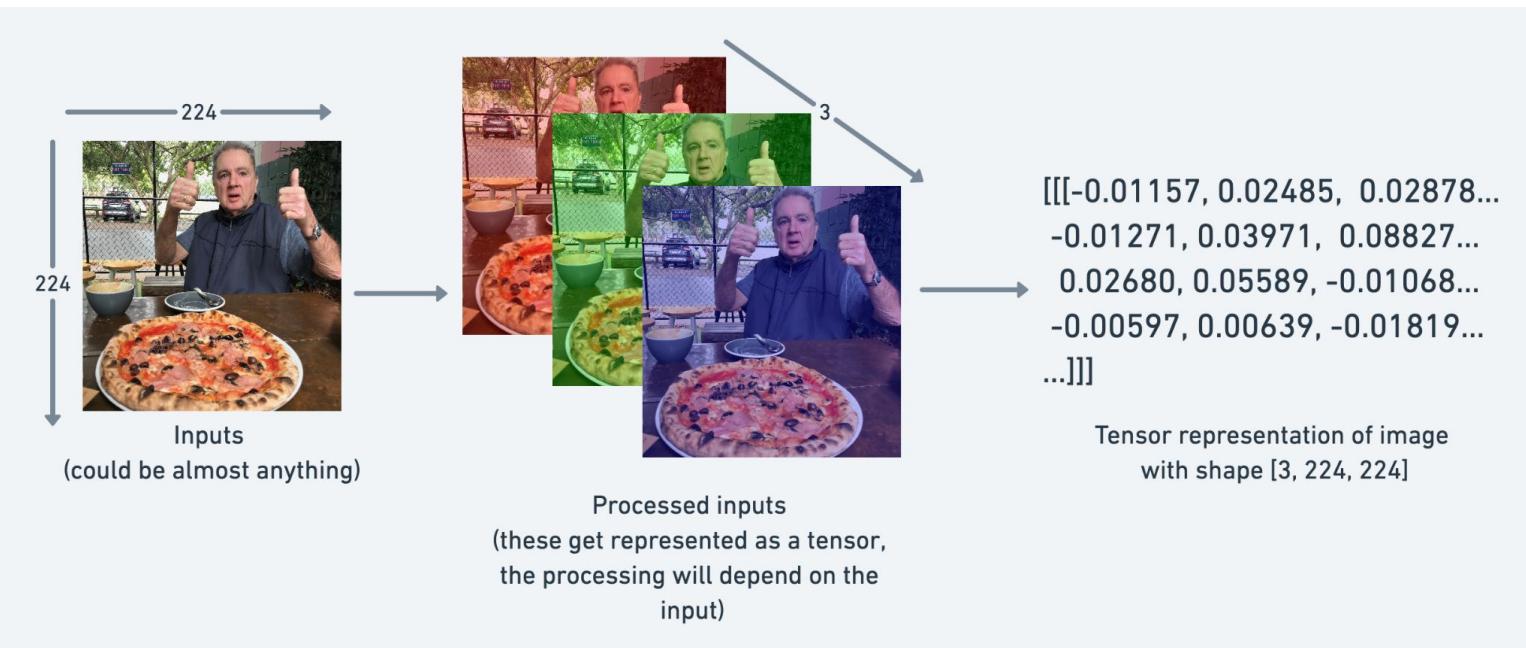


Train a (Convolutional) Neural Network in Pytorch

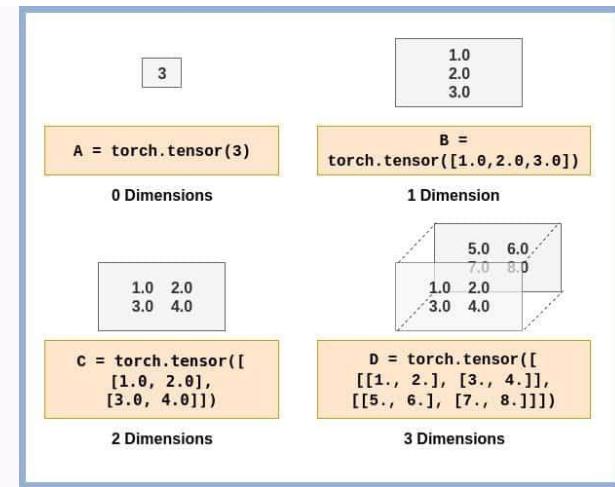


Pytorch: tensor

Tensor in Pytorch is equivalent to Array in Numpy



Pytorch: tensor



Pytorch: tensor

```
tensor([[1, 2, 3],  
       [3, 6, 9],  
       [2, 4, 5]])  
  
dim=0  
  
tensor([[1, 2, 3],  
       [3, 6, 9],  
       [2, 4, 5]])  
  
dim=1  
  
tensor([[1, 2, 3],  
       [3, 6, 9],  
       [2, 4, 5]])  
  
dim=2
```

Dimension (dim)
0 1 2
↓ ↓ ↓
torch.Size([1, 3, 3])

```
import torch  
  
tensor = torch.tensor([[1, 2, 3],  
                      [3, 6, 9],  
                      [2, 4, 5]])  
  
print(tensor.shape)  
print(tensor.ndim)  
print(tensor.dtype)  
print(tensor.device)  
  
example x  
C:\Users\Viet\anaconda3\envs\basic\py  
torch.Size([1, 3, 3])  
3  
torch.int64  
cpu
```

Pytorch: Image to tensor

```
import cv2
import torch

image = cv2.imread("cat.jpg")
cv2.imshow("image", image)
image = torch.from_numpy(image)
print("Image's shape: {}".format(image.shape))
print("Image's number of dimensions: {}".format(image.ndim))
cv2.waitKey(0)

pytorch_tutorials x
C:\Users\Viet\anaconda3\envs\basic\python.exe C:\Users\Viet\Pycha
Image's shape: torch.Size([441, 784, 3])
Image's number of dimensions: 3


```

```
from PIL import Image
from torchvision.transforms import ToTensor

image = Image.open("cat.jpg")
image.show()
transform = ToTensor()
image = transform(image)
print("Image's shape: {}".format(image.shape))
print("Image's number of dimensions: {}".format(image.ndim))

pytorch_tutorials x
C:\Users\Viet\anaconda3\envs\basic\python.exe C:\Users\Viet\Pycha
Image's shape: torch.Size([3, 441, 784])
Image's number of dimensions: 3


```

Pytorch: Datasets

Built-in datasets

airplane



automobile



bird



cat



deer



dog



frog



horse



ship



truck



CIFAR dataset

The screenshot shows a PyCharm IDE interface. On the left, there's a file tree with 'pytorch_scripts' containing 'cifar' and 'cifar-10-batches-py'. The 'cifar-10-batches-py' folder contains files like 'batch.meta', 'data_batch_1', 'data_batch_2', 'data_batch_3', 'data_batch_4', 'data_batch_5', 'readme.html', and 'test_batch'. Below it is 'cifar-10-python.tar.gz' and 'builtin_dataset.py'. The main code editor window displays the following Python script:

```
from torchvision.datasets import CIFAR10

train_dataset = CIFAR10(root='./cifar', train=True, download=True)
test_dataset = CIFAR10(root='./cifar', train=False, download=True)
index = 1009 # sample index
image, label = train_dataset.__getitem__(index)
image.show()
print(image.size)
print(label)
print(train_dataset.classes)
print(train_dataset.class_to_idx)
```

The 'Run' tab at the bottom shows the command: C:\Users\Viet\anaconda3\python.exe C:\Users\Viet\PycharmProjects\p. It also shows the progress of downloading the CIFAR-10 dataset from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz, with a speed of 3272326.26it/s. The output window shows the classes and class-to-index mapping for the dataset.

Run: C:\Users\Viet\anaconda3\python.exe C:\Users\Viet\PycharmProjects\p
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
100% [██████████] 170498071 / 170498071 [00:52<00:00, 3272326.26it/s]
Extracting ./cifar\cifar-10-python.tar.gz to ./cifar
Files already downloaded and verified
(32, 32)
2
['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
{'airplane': 0, 'automobile': 1, 'bird': 2, 'cat': 3, 'deer': 4, 'dog': 5, 'frog': 6, 'horse': 7, 'ship': 8, 'truck': 9}

Pytorch: Datasets

ImageFolder

Butterfly dataset

The screenshot shows a PyCharm interface with a file tree on the left and a code editor on the right.

File Tree:

```
1.   |   train
2.   |       |   class1
3.   |       |       |   1.jpg
4.   |       |       |   2.jpg
5.   |       |   class2
6.   |       |       |   1.jpg
7.   |       |       |   2.jpg
8.   |   valid
9.   |       |   class1
10.  |       |       |   1.jpg
11.  |       |       |   2.jpg
12.  |       |   class2
13.  |       |       |   1.jpg
14.  |       |       |   2.jpg
```

Code Editor (datasets_script.py):

```
1. from torchvision.datasets import ImageFolder
2.
3. train_dataset = ImageFolder(root="butterfly/train")
4. val_dataset = ImageFolder(root="butterfly/valid")
5. test_dataset = ImageFolder(root="butterfly/test")
6.
7. index = 1009 # sample index
8. image, label = train_dataset.__getitem__(index)
9. image.show()
10. print(image.size)
11. print(label)
12. print(train_dataset.classes)
13. print(train_dataset.class_to_idx)
```

Run Output:

```
C:\Users\Viet\anaconda3\python.exe C:\Users\Viet\PycharmProjects\pytorch_scripts\datasets_script.py
(224, 224)
8
['ADONIS', 'AFRICAN GIANT SWALLOWTAIL', 'AMERICAN SNOOT', 'AN 88', 'APOLLO', 'ARCIGERA FLOWER MOTH', 'ATALA', 'ATLAS MOTH', 'BANDED ORANGE HELICONIAN', 'BANDED PEACOCK', 'BANDED TIGER MOTH', 'BECKERS WHITE', 'BIRD CHERRY ERMINE MOTH', 'BLACK HAIRSTREAK', 'BLUE MORPHO', 'BLUE SPOTTED CROW', 'BROOKES BIRDWING']
{'ADONIS': 0, 'AFRICAN GIANT SWALLOWTAIL': 1, 'AMERICAN SNOOT': 2, 'AN 88': 3, 'APOLLO': 4, 'ARCIGERA FLOWER MOTH': 5, 'ATALA': 6, 'ATLAS MOTH': 7}
```

Image Preview: A close-up photograph of a butterfly with brown and orange patterns on its wings, resting on a yellow flower.

Pytorch: Built-in datasets

Pytorch all built-in datasets

Pytorch: Datasets

Requirements for a custom dataset implementation:

a subclass of
torch.utils.data.Dataset

has the `__len__`
method implemented

has the `__getitem__`
method implemented

```
class TinyData(Dataset):
    def __init__(self, setname):
        """Tiny Dataset for 32 x 32 images for color classification.
        Variables:
        <setname> can be any of: 'train' to specify the training set
                                    'val' to specify the validation set
                                    'test' to specify the test set"""
        self.setname = setname
        assert setname in ['train', 'val', 'test']

    #Define dataset
    overall_dataset_dir = os.path.join(os.path.join(os.getcwd(), 'load_dataset'), 'tiny_data')
    self.selected_dataset_dir = os.path.join(overall_dataset_dir, setname)

    #E.g. self.all_filenames = ['006.png', '007.png', '008.png'] when setname=='val'
    self.all_filenames = os.listdir(self.selected_dataset_dir)
    self.all_labels = pd.read_csv(os.path.join(overall_dataset_dir, 'tiny_labels.csv'), header=0, index_col=0)
    self.label_meanings = self.all_labels.columns.values.tolist()

    def __len__(self):
        """Return the total number of examples in this split, e.g. if
        self.setname=='train' then return the total number of examples
        in the training set"""
        return len(self.all_filenames)

    def __getitem__(self, idx):
        """Return the example at index [idx]. The example is a dict with keys
        'data' (value: Tensor for an RGB image) and 'label' (value: multi-hot
        vector as Torch tensor of gr truth class labels)."""
        selected_filename = self.all_filenames[idx]
        imagepil = PIL.Image.open(os.path.join(self.selected_dataset_dir, selected_filename)).convert('RGB')

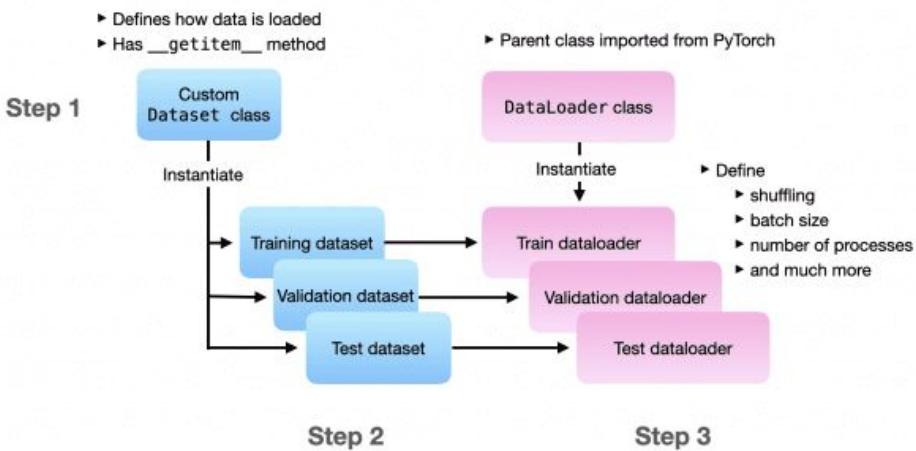
        #convert image to Tensor and normalize
        image = utils.to_tensor_and_normalize(imagepil)

        #load label
        label = torch.Tensor(self.all_labels.loc[selected_filename, :].values)

        sample = {'data':image, #preprocessed image, for input into NN
                  'label':label,
                  'img_idx':idx}
        return sample
```



Step 1: Setup a dataset



```
...  
from torchvision.datasets import CIFAR10  
from torchvision.transforms import ToTensor  
from torch.utils.data import DataLoader  
  
if __name__ == '__main__':  
    training_data = CIFAR10(  
        root='./cifar',  
        train=True,  
        download=True,  
        transform=ToTensor()  
    )  
    test_data = CIFAR10(  
        root='./cifar',  
        train=False,  
        download=True,  
        transform=ToTensor()  
    )  
    train_dataloader = DataLoader(  
        dataset=training_data,  
        batch_size=16,  
        num_workers=8,  
        drop_last=True,  
        shuffle=True  
    )  
    test_dataloader = DataLoader(  
        dataset=test_data,  
        batch_size=4,  
        num_workers=8,  
        drop_last=False,  
        shuffle=False,  
    )  
  
    for images, labels in train_dataloader:  
        print(images.shape, labels.shape) # torch.Size([16, 3, 32, 32]) torch.Size([16])
```

Step 2: Define a Neural Network

```
...  
  
import torch  
import torch.nn as nn  
  
class SimpleNeuralNetwork(nn.Module):  
    def __init__(self, num_classes=10):  
        super().__init__()  
        self.flatten = nn.Flatten()  
        self.layer_1 = nn.Sequential(  
            nn.Linear(3 * 32 * 32, 512),  
            nn.ReLU()  
        )  
        self.layer_2 = nn.Sequential(  
            nn.Linear(512, 1024),  
            nn.ReLU(),  
        )  
        self.layer_3 = nn.Sequential(  
            nn.Linear(1024, 512),  
            nn.ReLU(),  
        )  
        self.layer_4 = nn.Linear(512, num_classes)  
  
    def forward(self, x):  
        x = self.flatten(x)  
        x = self.layer_1(x)  
        x = self.layer_2(x)  
        x = self.layer_3(x)  
        x = self.layer_4(x)  
        return x  
  
if __name__ == '__main__':  
    random_image = torch.rand(1, 3, 32, 32) # [B,C,H,W]  
    model = SimpleNeuralNetwork() # Define a neural network  
    predictions = model(random_image) # Forward pass  
    print(predictions)  
    # tensor([[ 0.0320, -0.0539, -0.0715,  0.0101,  0.0413, -0.0189,  0.0116, -0.0245,  
    #           0.0019,  0.0408]], grad_fn=<AddmmBackward0>)
```

```
...  
  
import torch  
import torch.nn as nn  
  
class SimpleNeuralNetwork(nn.Module):  
    def __init__(self, num_classes=10):  
        super().__init__()  
        self.flatten = nn.Flatten()  
        self.layers = nn.Sequential(  
            nn.Linear(3 * 32 * 32, 512),  
            nn.ReLU(),  
            nn.Linear(512, 1024),  
            nn.ReLU(),  
            nn.Linear(1024, 512),  
            nn.ReLU(),  
            nn.Linear(512, num_classes),  
        )  
  
    def forward(self, x):  
        x = self.flatten(x)  
        x = self.layers(x)  
        return x  
  
if __name__ == '__main__':  
    random_image = torch.rand(1, 3, 32, 32) # [B,C,H,W]  
    model = SimpleNeuralNetwork() # Define a neural network  
    predictions = model(random_image) # Forward pass  
    print(predictions)  
    # tensor([[ 0.0320, -0.0539, -0.0715,  0.0101,  0.0413, -0.0189,  0.0116, -0.0245,  
    #           0.0019,  0.0408]], grad_fn=<AddmmBackward0>)
```

Step 3: Define a loss function and an optimizer

```
import torch.nn as nn
from torch.optim import SGD

criterion = nn.CrossEntropyLoss()
optimizer = SGD(model.parameters(), lr=0.001, momentum=0.9)
```

Step 4: Train the network

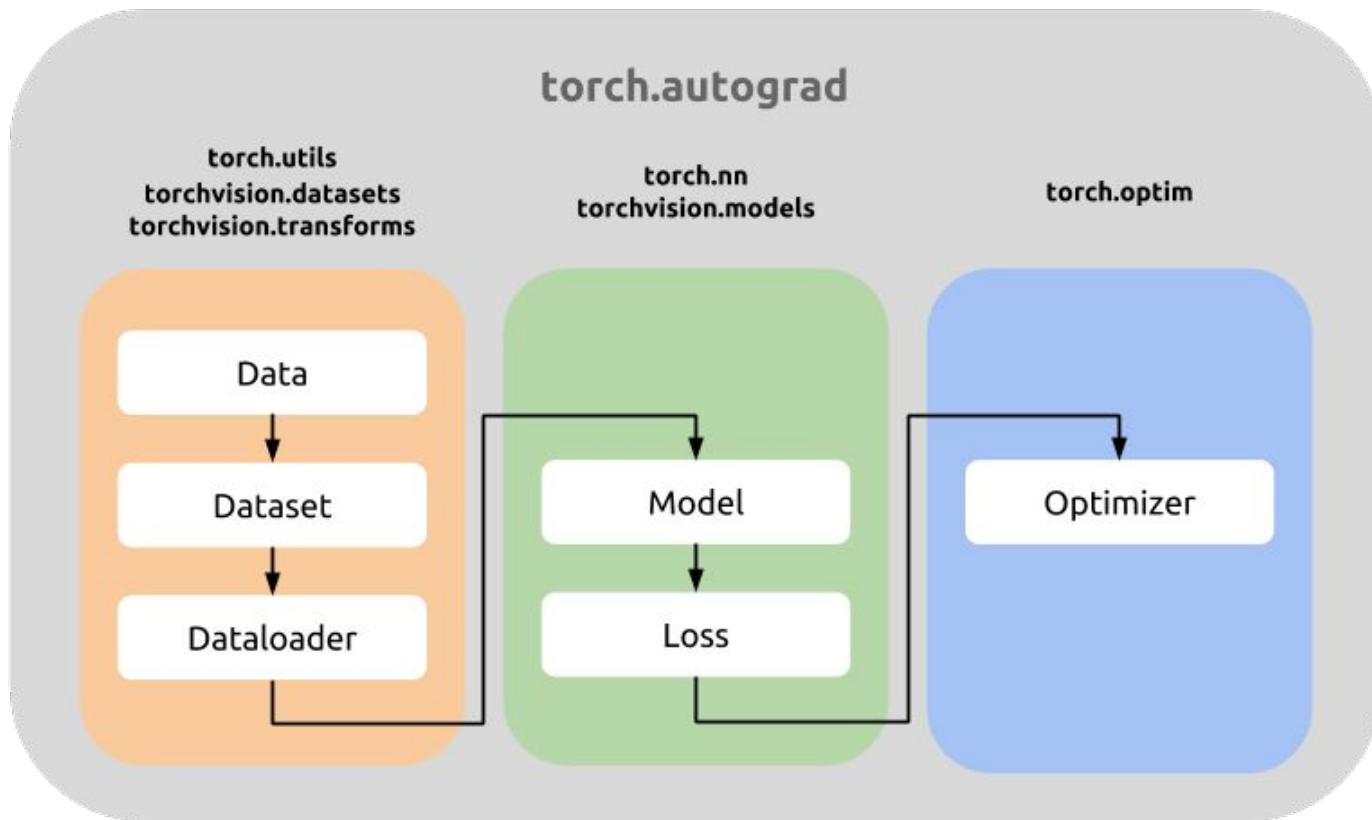
```
batch_size = 128
num_epochs = 100

transform = Compose([
    ToTensor(),
    Normalize((0.49139968, 0.48215841, 0.44653091), (0.24703223, 0.24348513, 0.26158784))
])
train_set = CIFARDataset(root='./data', train=True, transform=transform)
train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True, num_workers=0)
test_set = CIFARDataset(root='./data', train=False, transform=transform)
test_loader = DataLoader(test_set, batch_size=batch_size, shuffle=False, num_workers=0)
model = SimpleNet()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
num_iters = len(train_loader)

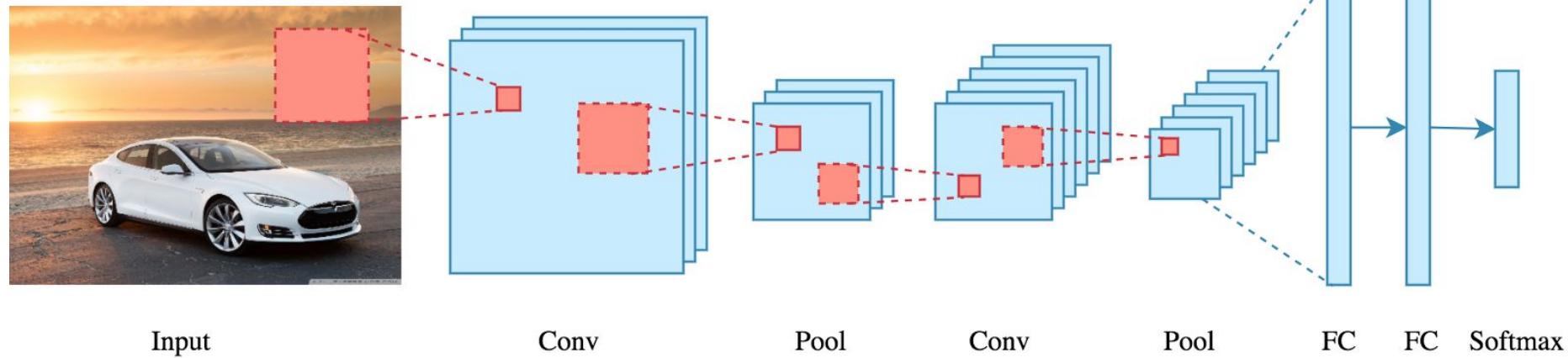
for epoch in range(num_epochs):
    model.train()
    # Training step
    for i, (images, labels) in enumerate(train_loader):
        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Step 4: Train the network



Convolutional Neural Network



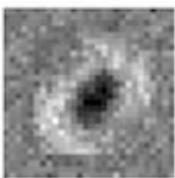
Problems in Neural Network

Variations

- Position
- Orientation
- Scale

Scalable

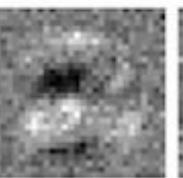
- Memory
- Calculation



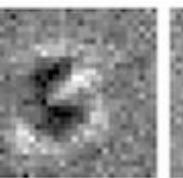
0



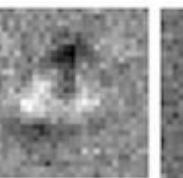
1



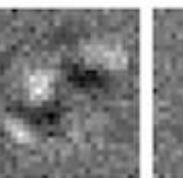
2



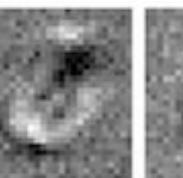
3



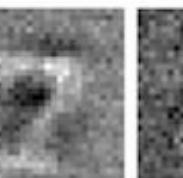
4



5



6



7



8



9



airplane



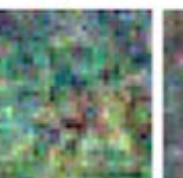
automobile



bird



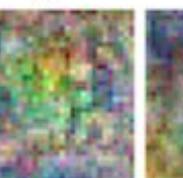
cat



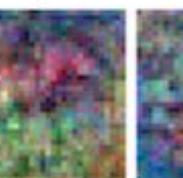
deer



dog



frog



horse

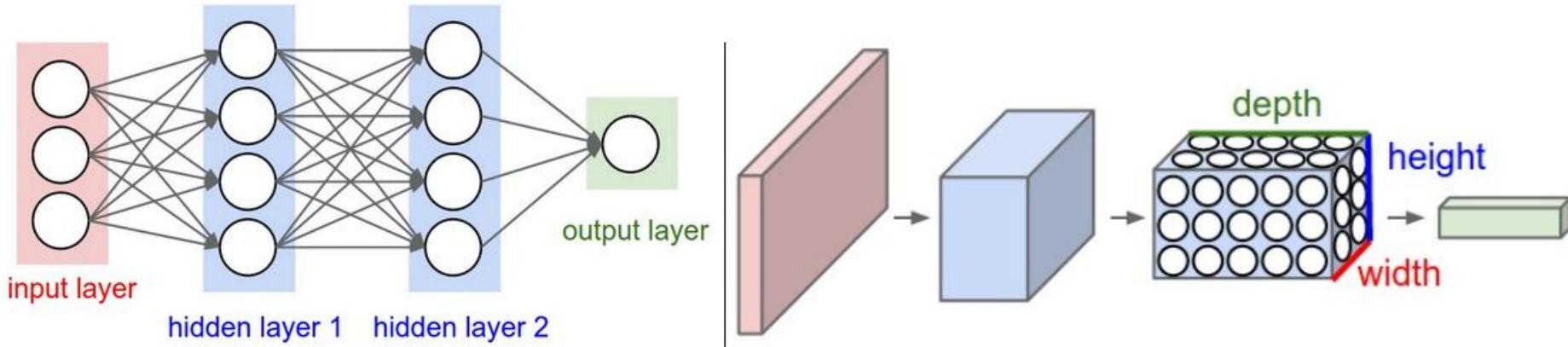
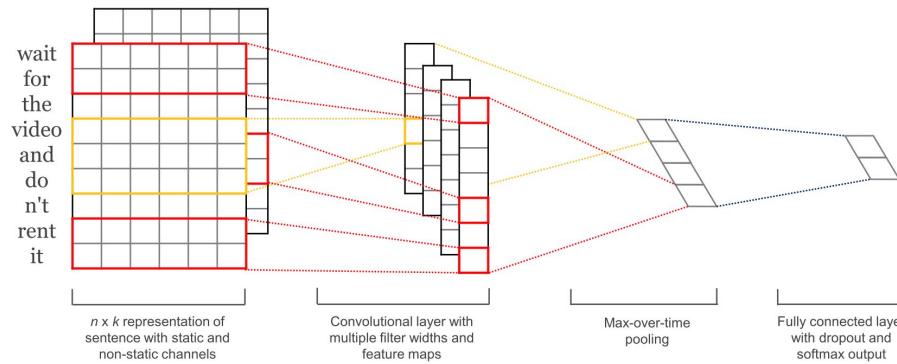


ship



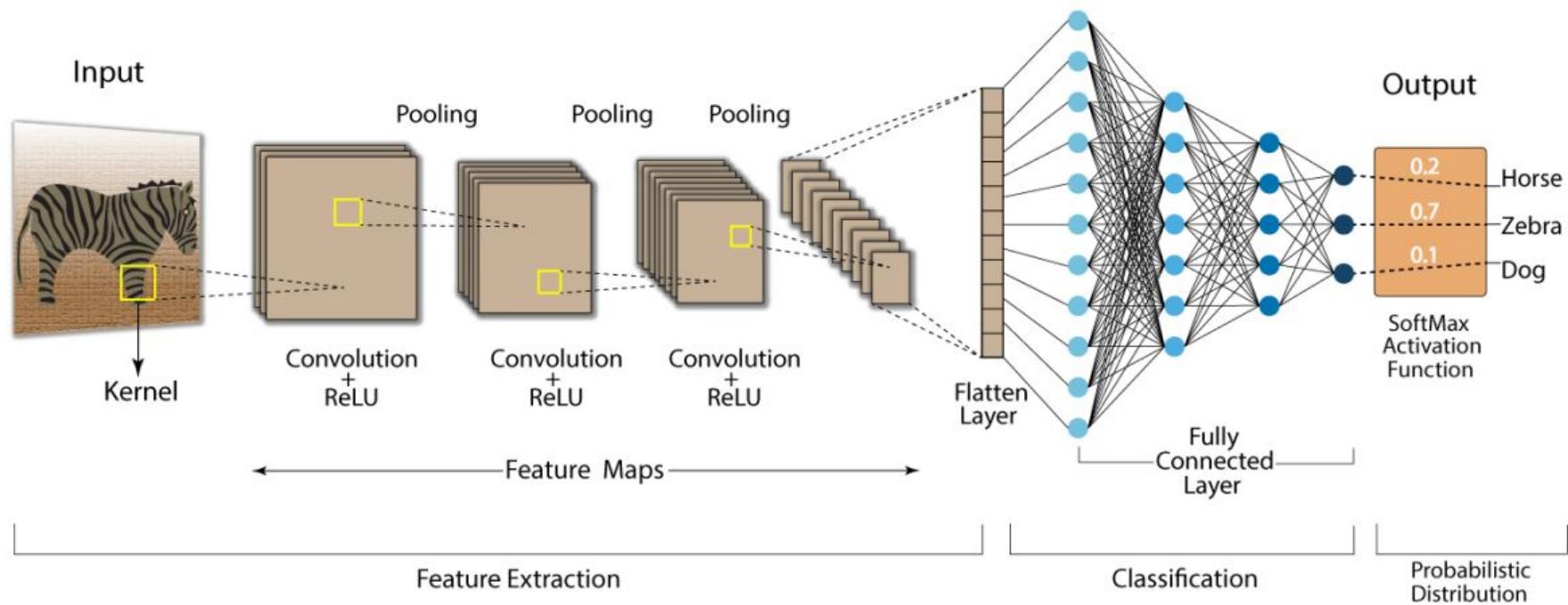
truck

Convolutional Neural Network



Convolutional Neural Network

Convolution Neural Network (CNN)



Convolutional operation

Input								
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			

$n_H \times n_W = 6 \times 6$

Filter		
1	0	-1
1	0	-1

*

Result		
2		

=

2		

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

$$2 = 4*1 + 9*0 + 2*(-1) + 5*1 + 6*0 + 2*(-1) + 2*1 + 4*0 + 5*(-1)$$

<https://indoml.com>

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

*

Result								
2	6							

=

6		

<https://indoml.com>

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			

Dimension: 6×6

Filter		
1	0	-1
1	0	-1

*

Result		
2	1	

=

1		

Parameters:
Size: $f = 3$
Stride: $s = 2$
Padding: $p = 1$

$$1 = 2*1 + 5*0 + 3*(-1) + 2*1 + 4*0 + 3*(-1) + 5*1 + 4*0 + 2*(-1)$$

<https://indoml.com>

Input								
0	0	0	0	0	0	0	0	0
0	4	9	2	5	8	3	0	0
0	5	6	2	4	0	3	0	0
0	2	4	5	4	5	2	0	0
0	5	6	5	4	7	8	0	0
0	5	7	7	9	2	1	0	0
0	5	8	5	3	8	4	0	0
0	0	0	0	0	0	0	0	0

Dimension: 6×6

Filter		
1	0	-1
1	0	-1

*

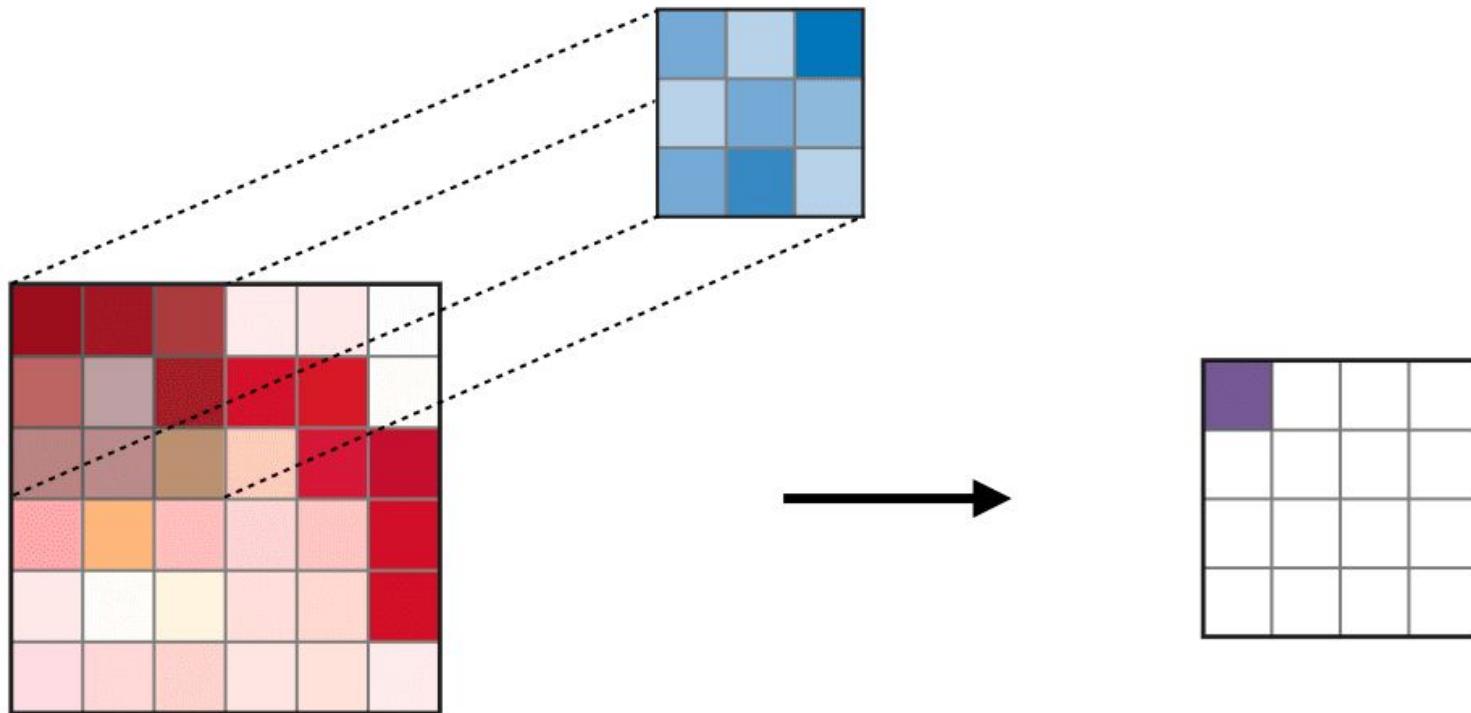
Result		
-15		

=

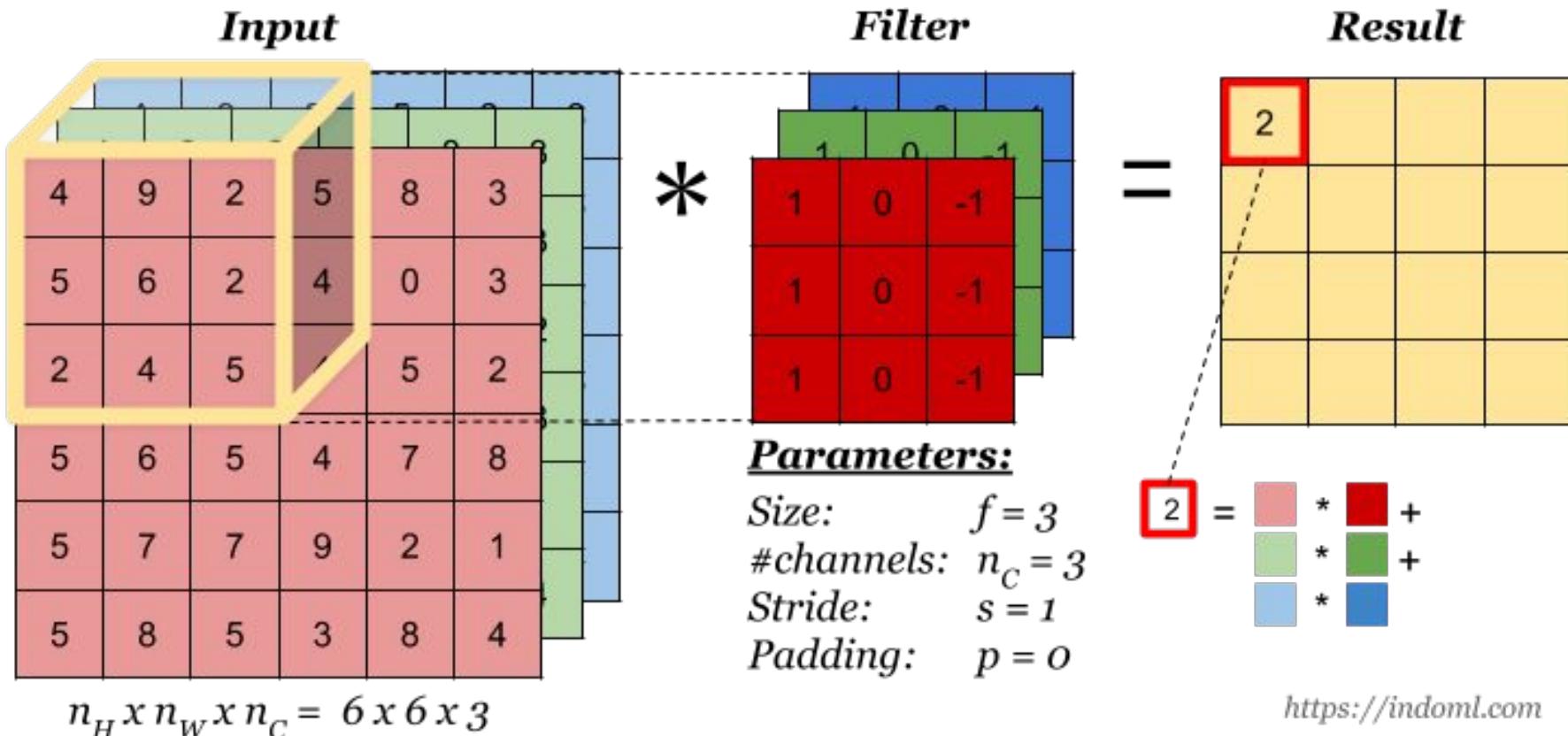
$$\boxed{-15} = 0*1 + 0*0 + 0*(-1) + 0*1 + 4*0 + 9*(-1) + 0*1 + 9*0 + 6*(-1) = -15$$

<https://indoml.com>

Convolutional operation



Convolutional operation on Volume



Convolutional operation on Volume

0	0	0	0	0	0	0	...
0	156	155	156	158	158	158	...
0	153	154	157	159	159	159	...
0	149	151	155	158	159	159	...
0	146	146	149	153	158	158	...
0	145	143	143	148	158	158	...
...

Input Channel #1 (Red)

0	0	0	0	0	0	0	...
0	167	166	167	169	169	169	...
0	164	165	168	170	170	170	...
0	160	162	166	169	170	170	...
0	156	156	159	163	168	168	...
0	155	153	153	158	168	168	...
...

Input Channel #2 (Green)

0	0	0	0	0	0	0	...
0	163	162	163	165	165	165	...
0	160	161	164	166	166	166	...
0	156	158	162	165	166	166	...
0	155	155	158	162	167	167	...
0	154	152	152	157	167	167	...
...

Input Channel #3 (Blue)

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

Kernel Channel #1



308

+

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

Kernel Channel #2



-498

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



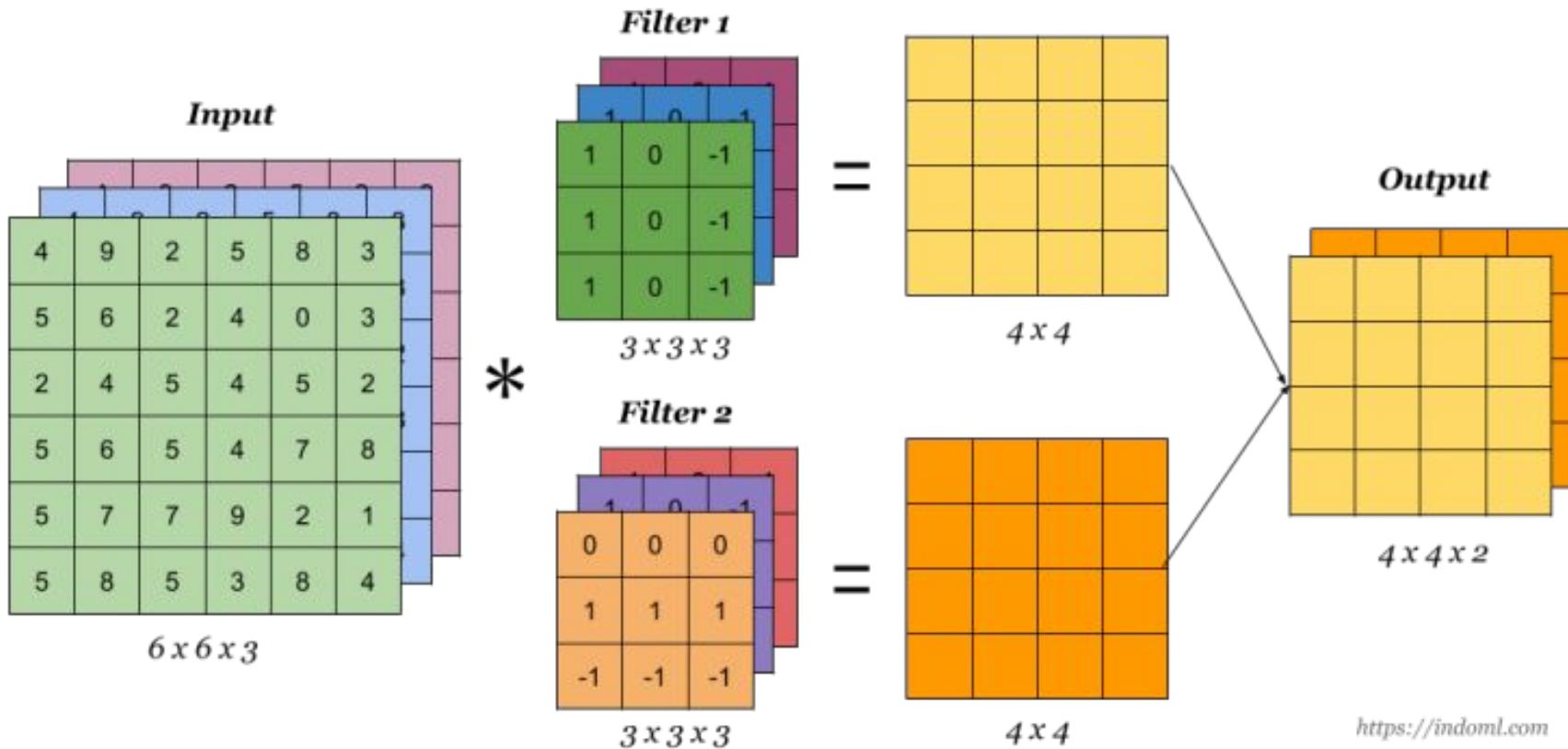
164 + 1 = -25

Bias = 1

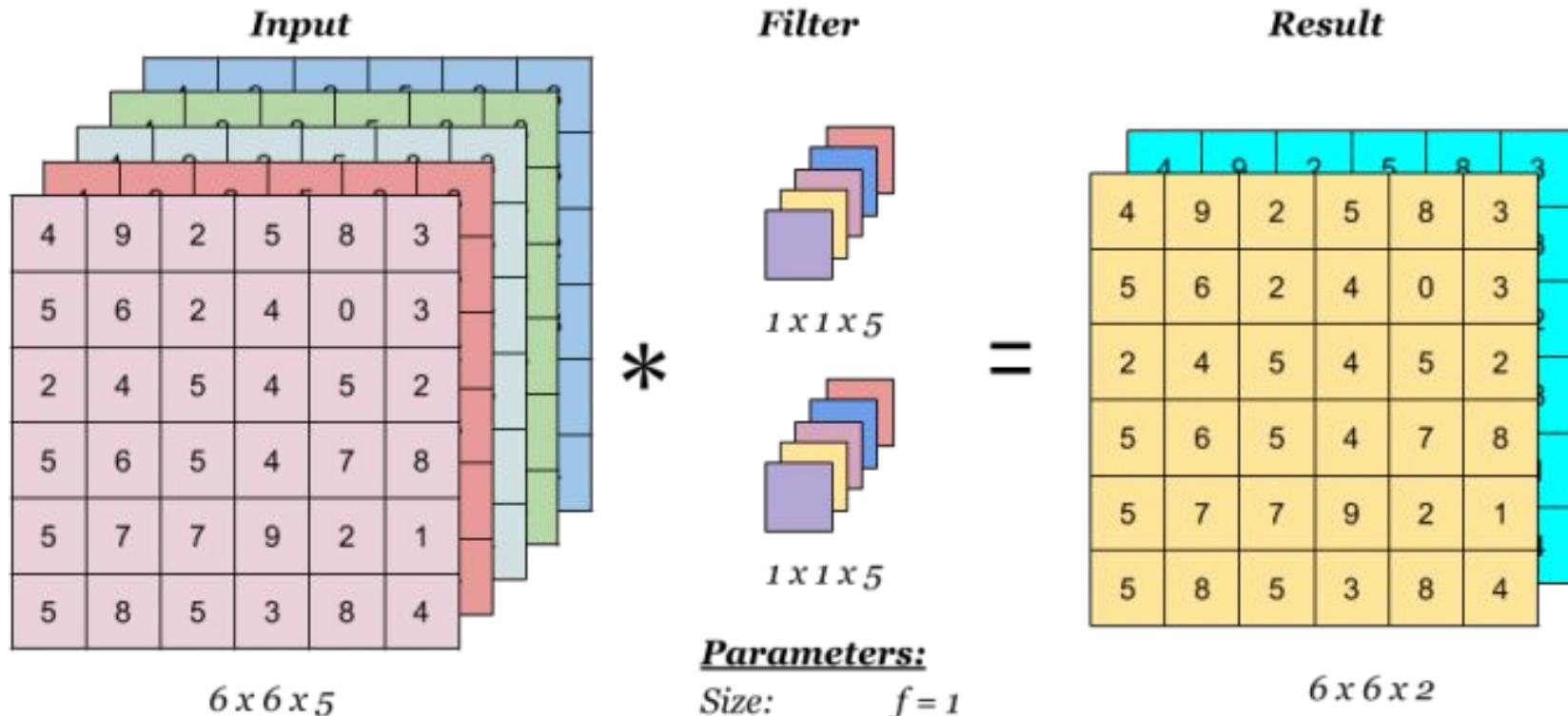
-25					...
					...
					...
					...
					...

Output

Convolutional operation with multiple Filters/Kernels

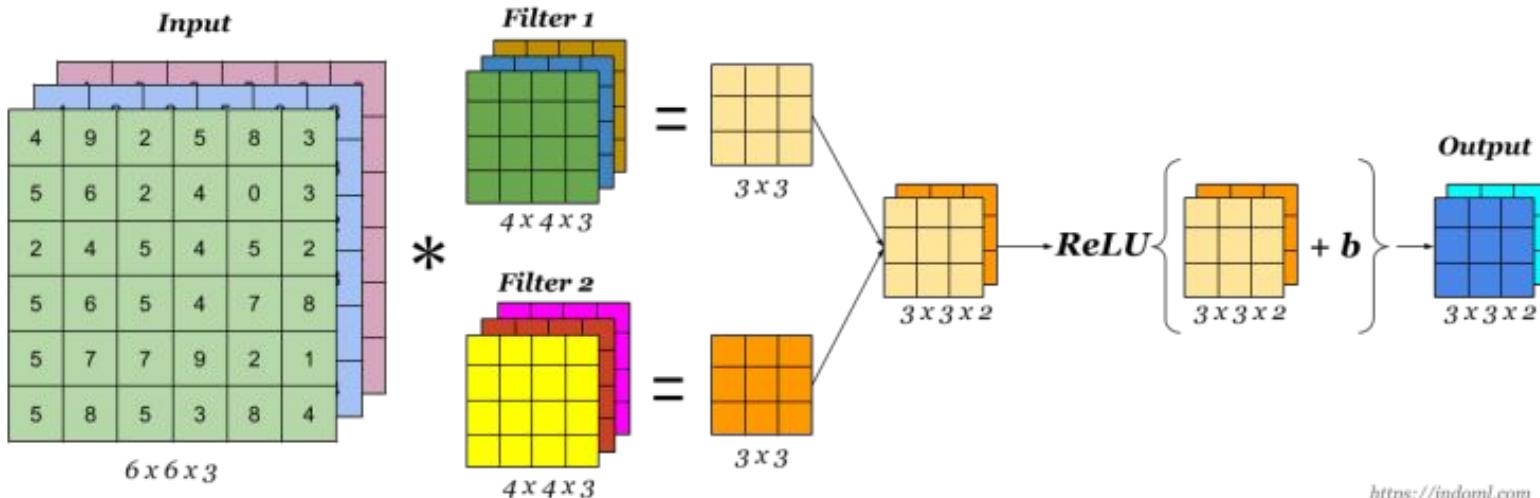


1x1 Convolutional operation



Convolutional layer

A Convolution Layer

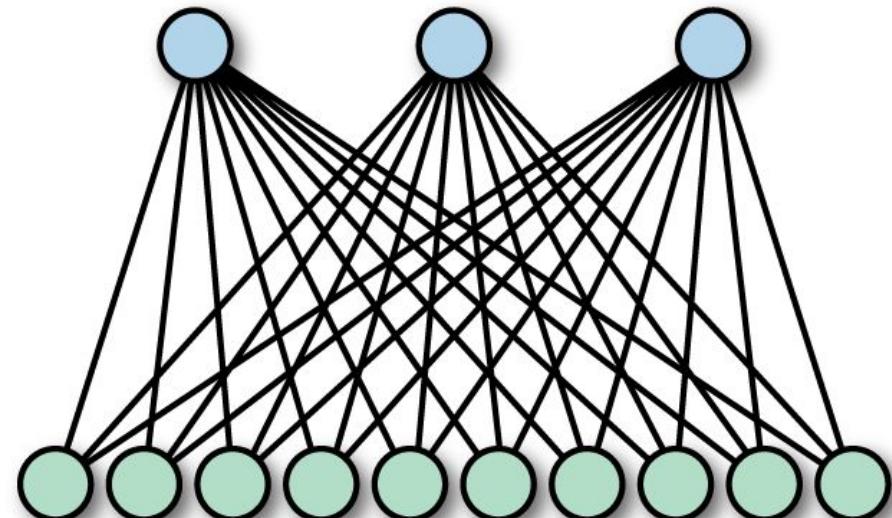


output size =

$$\frac{(\text{input size}) + 2 * \text{padding} - (\text{kernel size} - 1) - 1}{\text{stride}} + 1$$

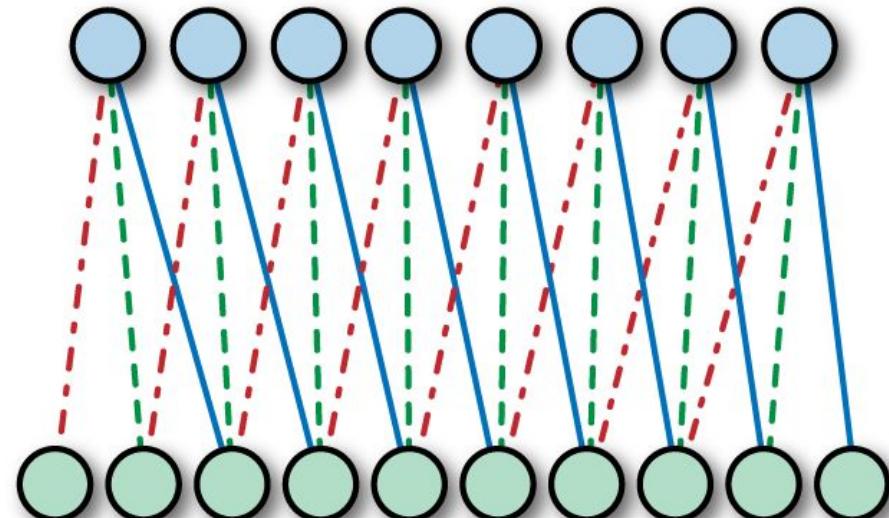
Convolutional layer's advantage

Fully Connected



Local Connectivity

Convolutional Layer



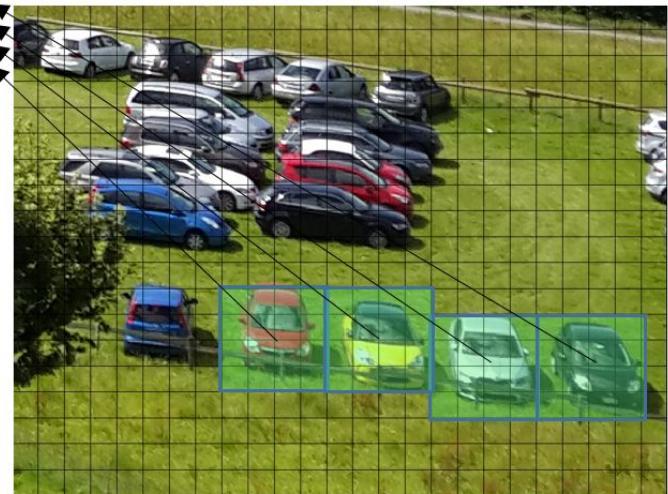
Parameter Sharing

Convolutional layer's advantage

TRANSLATION INVARIANCE →

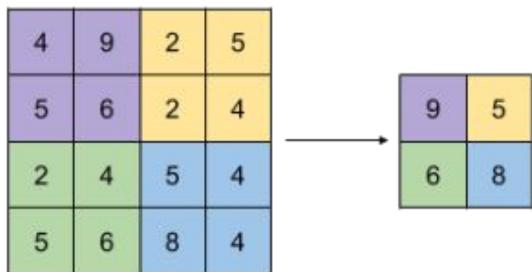


w_{11}	w_{12}	w_{13}	w_{14}
w_{21}	w_{22}	w_{23}	w_{24}
w_{31}	w_{32}	w_{33}	w_{34}
w_{41}	w_{42}	w_{43}	w_{44}

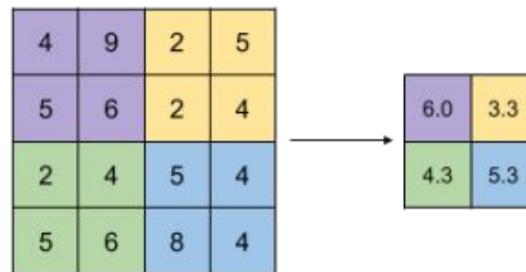


Pooling layer

Max Pooling

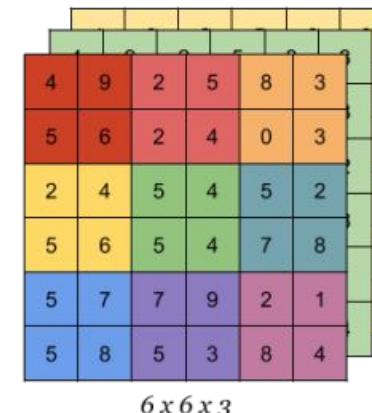


Avg Pooling

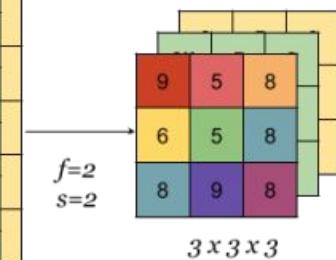


<https://indoml.com>

Input

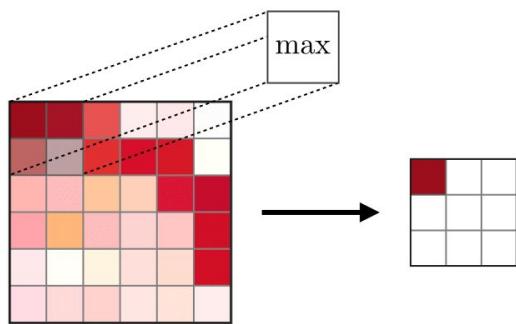


Max Pool

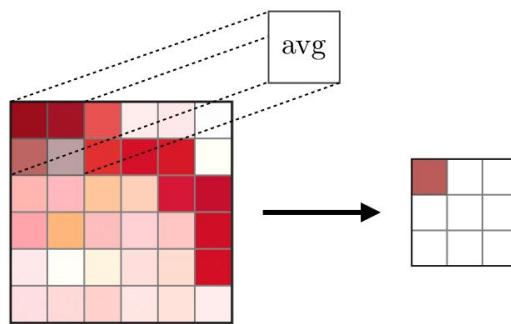


<https://indoml.com>

max



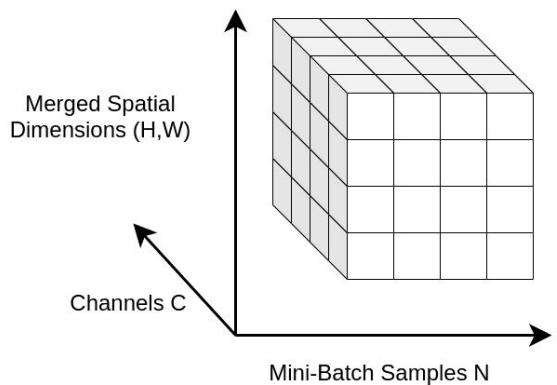
avg



Internal Covariate Shift



Batch Normalization layer

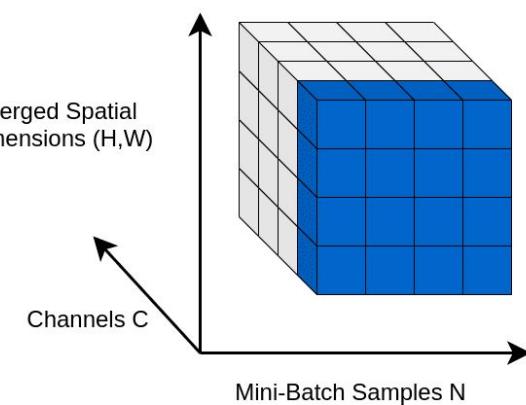


$$BN(x) = \gamma \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \beta$$

$$\mu_c(x) = \frac{1}{NHW} \sum_{n=1}^N \sum_{h=1}^H \sum_{w=1}^W x_{nchw}$$

$$\sigma_c(x) = \sqrt{\frac{1}{NHW} \sum_{n=1}^N \sum_{h=1}^H \sum_{w=1}^W (x_{nchw} - \mu_c(x))^2}$$

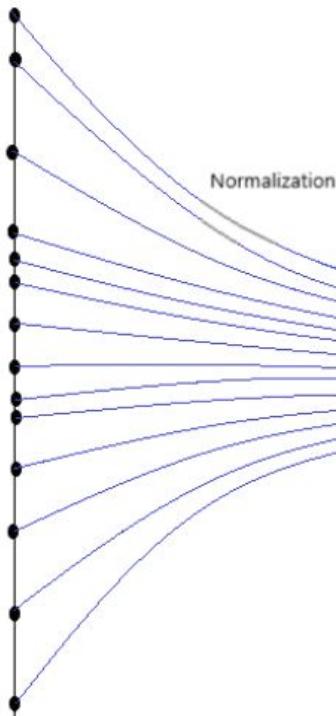
Batch Norm



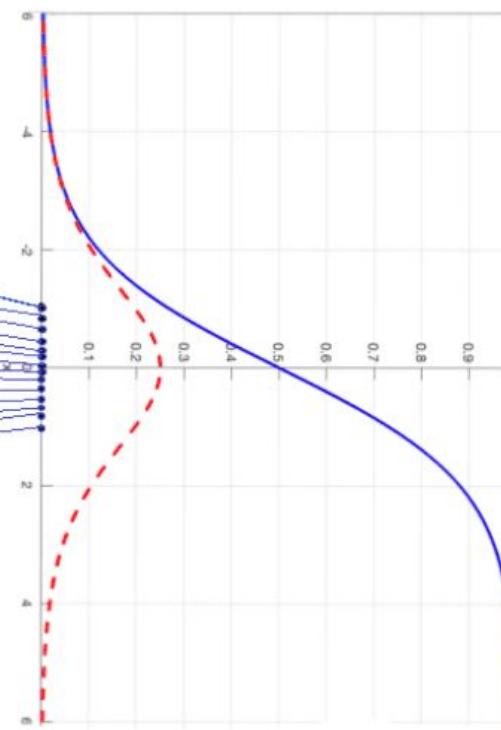
Batch Normalization

Why Normalization helps?

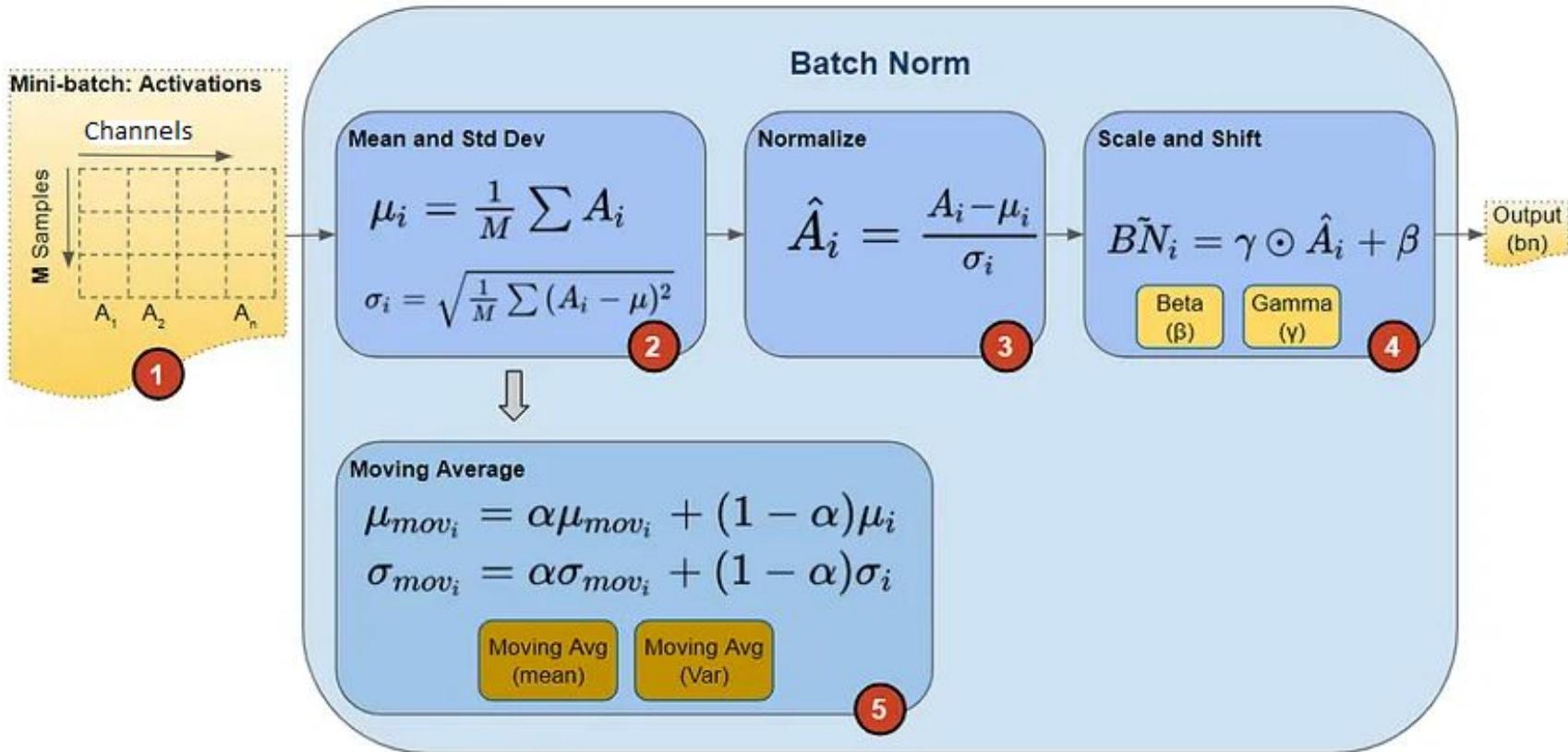
Activation Inputs



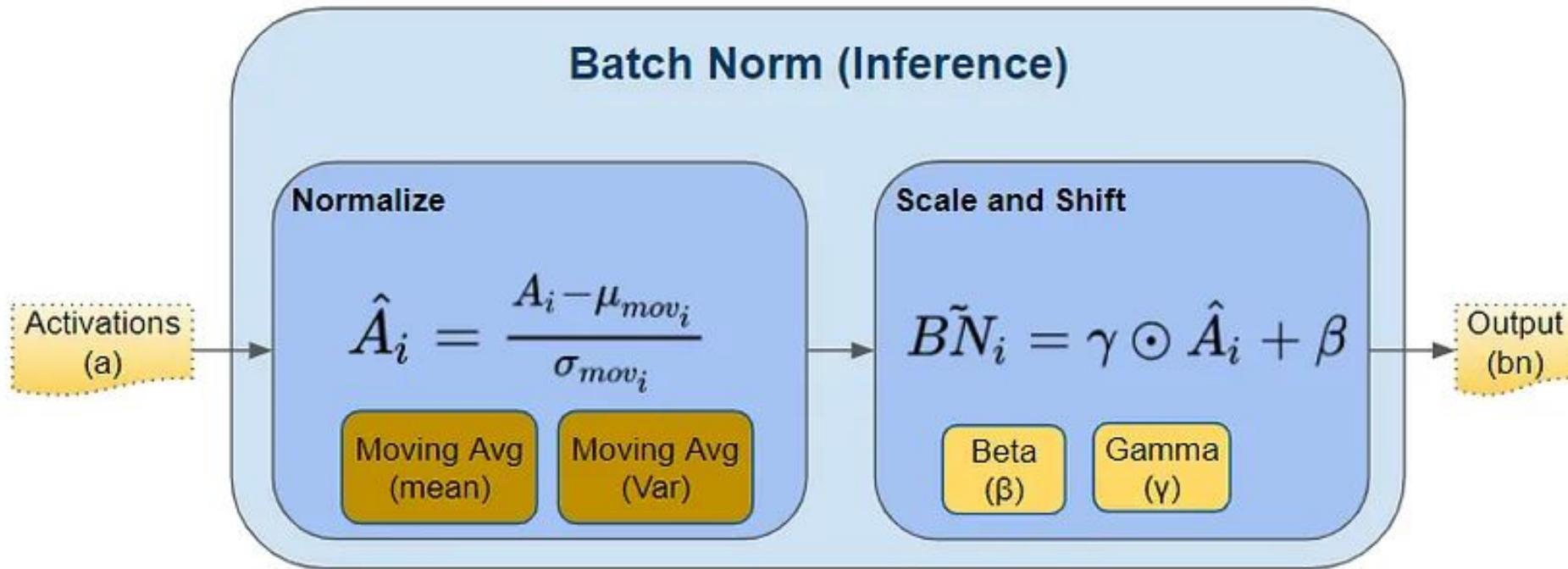
Sigmoid Activation and Gradient



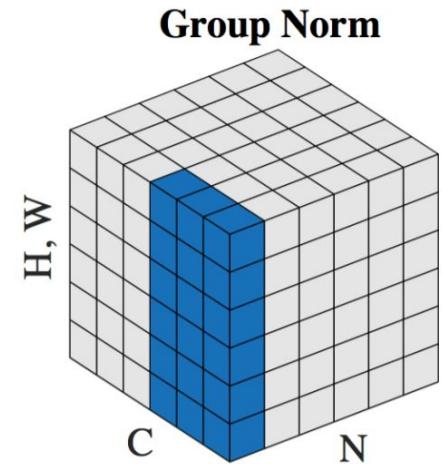
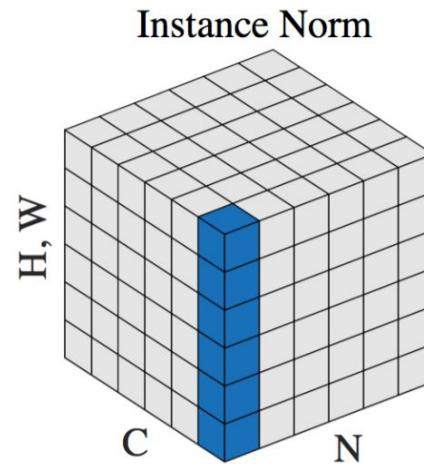
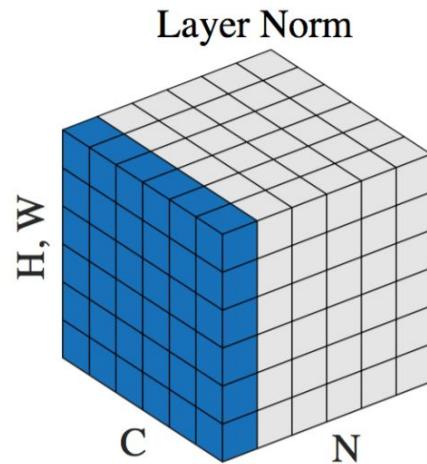
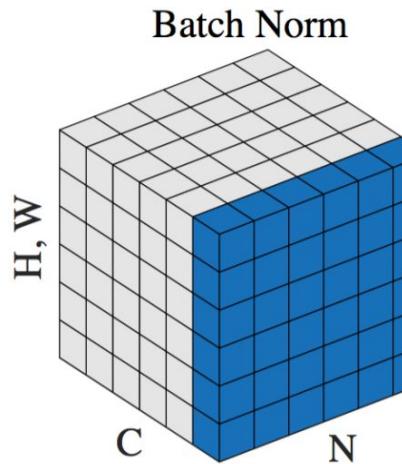
Batch Normalization: Training phase



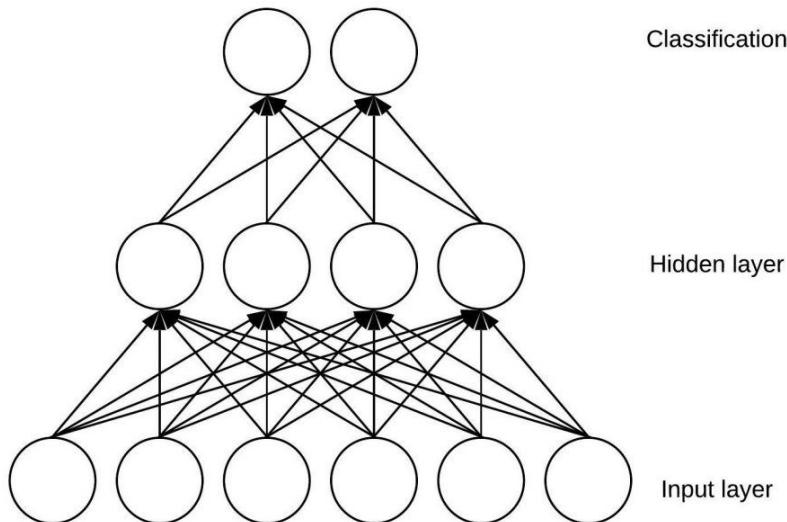
Batch Normalization: Inference phase



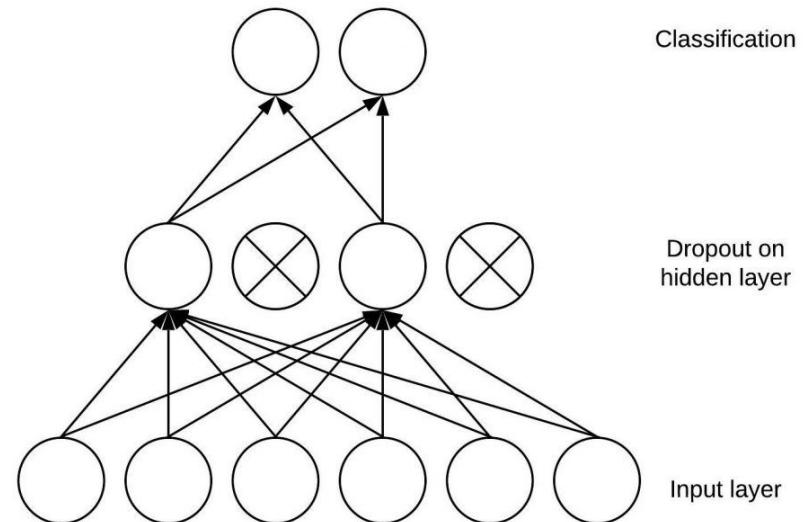
Normalization layer



Dropout layer



Without Dropout

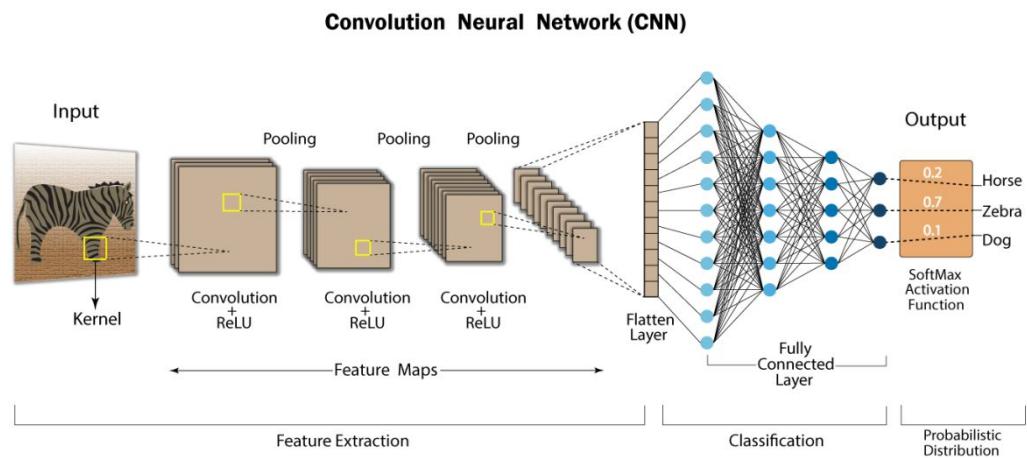


With Dropout

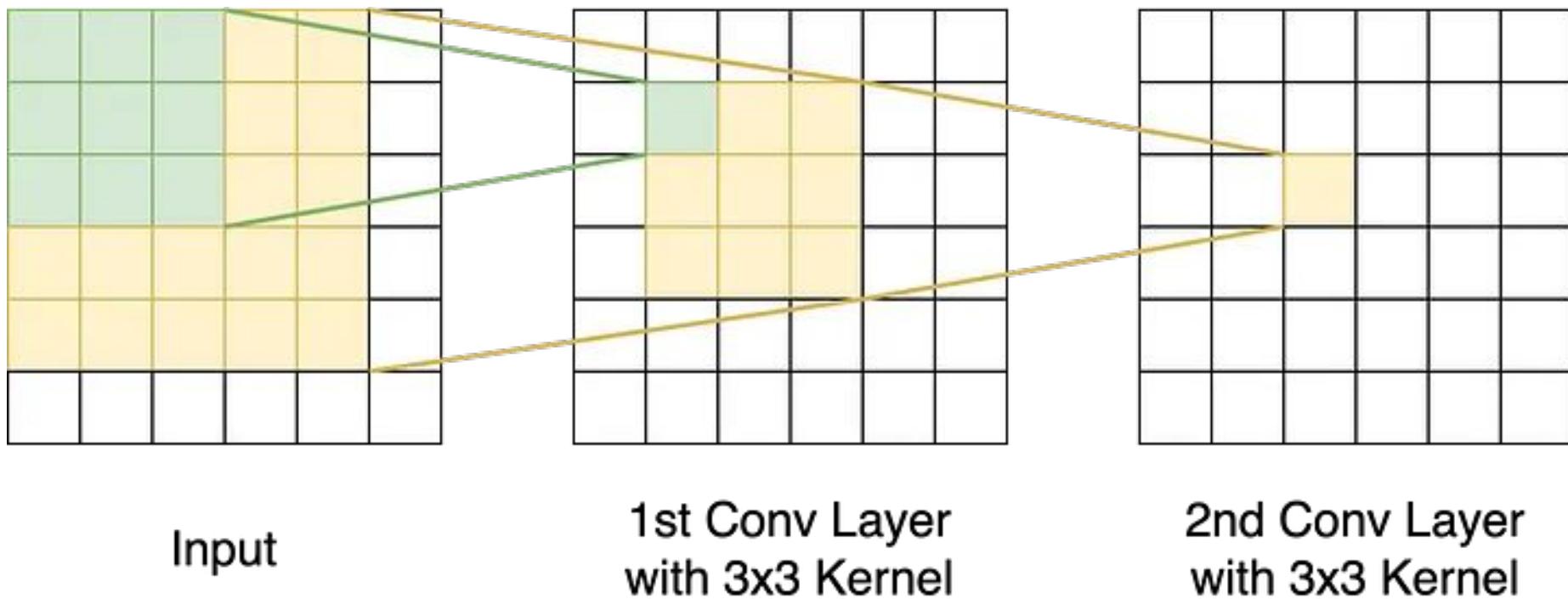
Common CNN's layer pattern (part 1)

Hình thức phổ biến nhất của CNN: Các layer lần lượt được sắp xếp theo thứ tự sau:

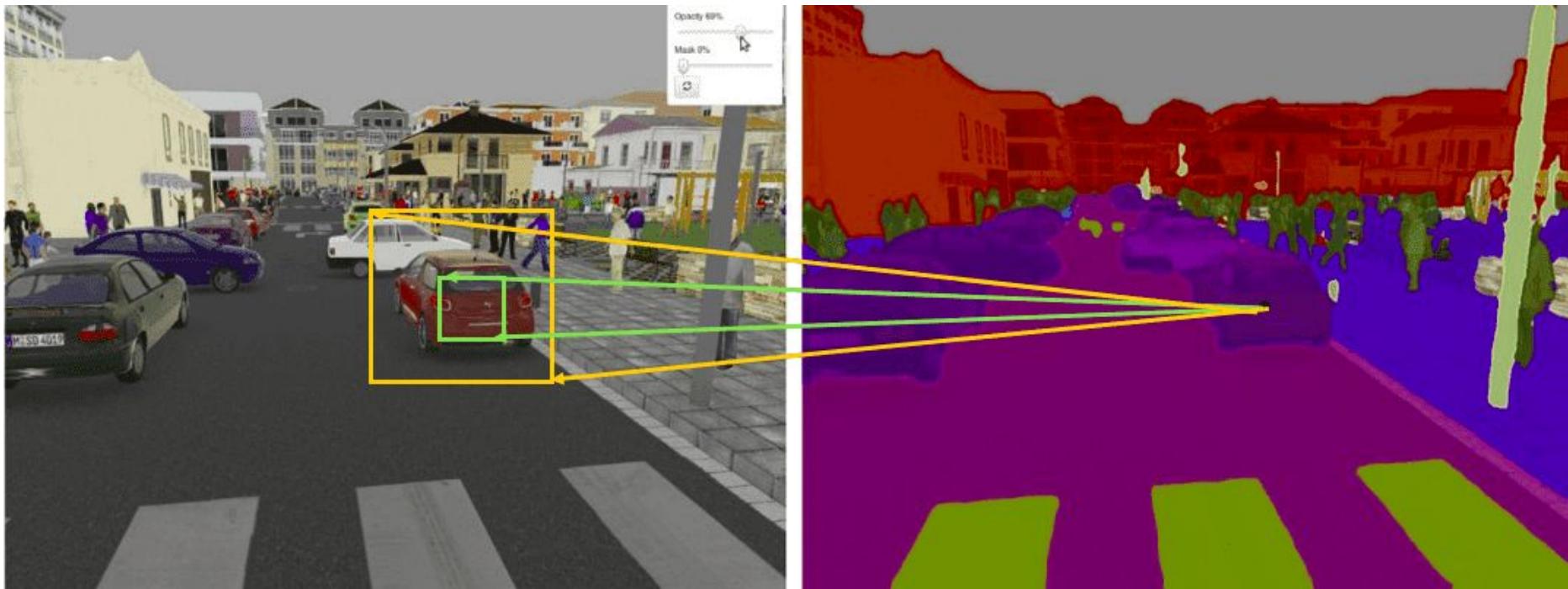
1. Một vài block của Conv->(BatchNorm)->ReLU
2. Max-pooling
3. Lặp lại bước 1 và 2 cho đến khi feature map đủ bé
4. Flatten (làm phẳng) feature map
5. Một (vài) fully connected layer



Receptive Field



Receptive Field



Receptive Field

Add more Conv layers

Make the network deeper

Add pooling layers

sub-sampling

Use Dilated Conv

Use Depth-wise Conv

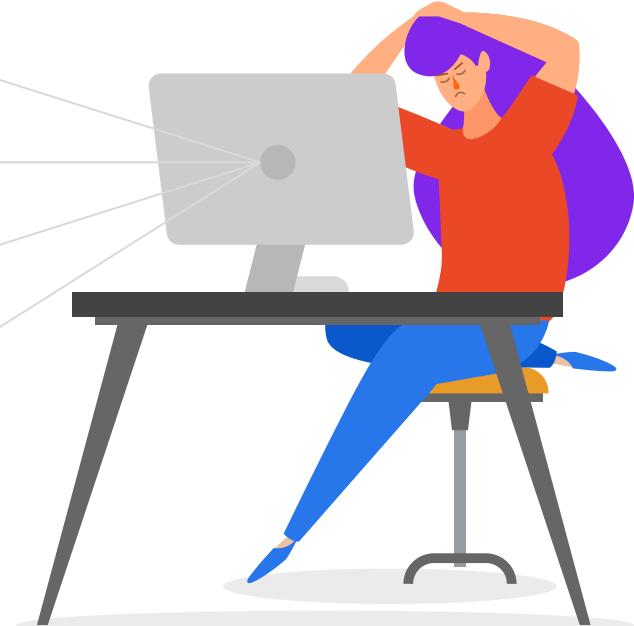
01

02

03

04

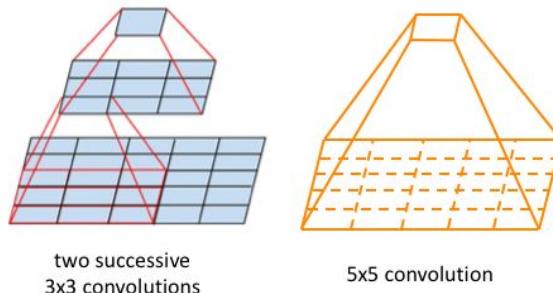
How to increase Receptive field



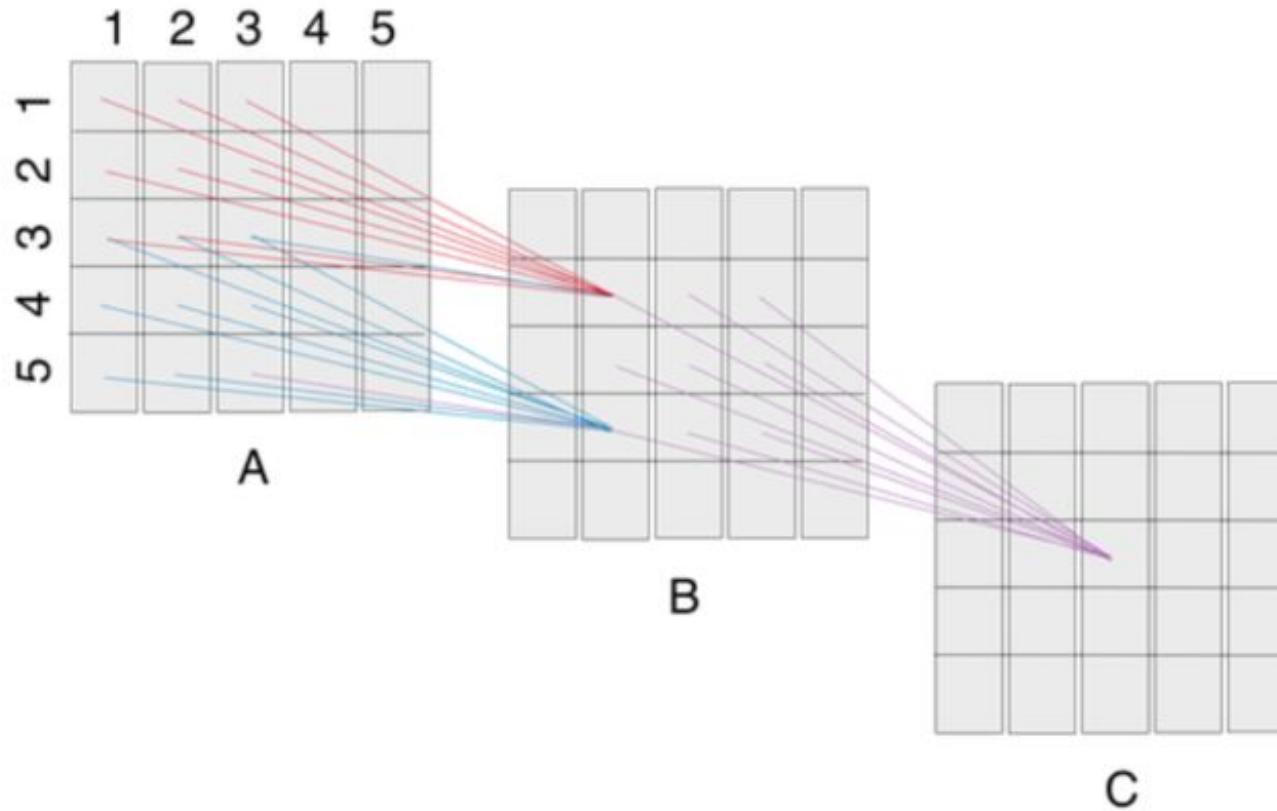
Common CNN's layer pattern (part 2)

Nhiều Conv layer với filter/kernel bé tốt hơn 1 Conv layer với filter/kernel lớn

	2 3x3-kernel Conv layers	1 5x5 kernel Conv layer
Receptive field of last layer		5x5
Number of parameters	$2 \cdot 3 \cdot 3$	$5 \cdot 5$
Number of non-linear layers	2	1



Effective Receptive Field



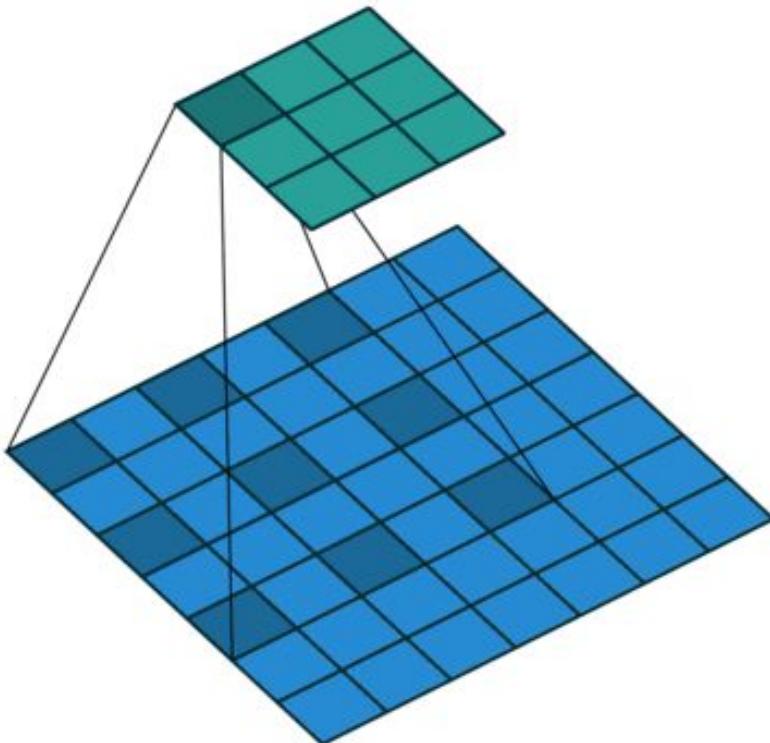
Use Pooling layer

12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

2×2 Max-Pool

20	30
112	37

Dilated/Atrous Convolutions



Common CNN's layer pattern (part 3)

Làm sao để thiết kế CNN architecture?

Rule 1: Don't be a hero. Use whatever works best one ImageNet

TORCHVISION.MODELS

The models subpackage contains definitions of models for addressing different tasks, including: image classification, pixelwise semantic segmentation, object detection, instance segmentation, person keypoint detection and video classification.

Classification

The models subpackage contains definitions for the following model architectures for image classification:

- AlexNet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3
- GoogLeNet
- ShuffleNet v2
- MobileNet v2
- ResNeXt
- Wide ResNet
- MNASNet

We provide pre-trained models, using the PyTorch `torch.utils.model_zoo`. These can be constructed by passing `pretrained=True`:

```
import torchvision.models as models
resnet18 = models.resnet18(pretrained=True)
alexnet = models.alexnet(pretrained=True)
squeezenet = models.squeezeNet1_0(pretrained=True)
vgg16 = models.vgg16(pretrained=True)
densenet = models.densenet161(pretrained=True)
inception = models.inception_v3(pretrained=True)
googlenet = models.googlenet(pretrained=True)
shufflenet = models.shufflenet_v2_x1_0(pretrained=True)
mobilenet = models.mobilenet_v2(pretrained=True)
resnext50_32x4d = models.resnext50_32x4d(pretrained=True)
wide_resnet50_2 = models.wide_resnet50_2(pretrained=True)
mnasnet = models.mnasnet1_0(pretrained=True)
```

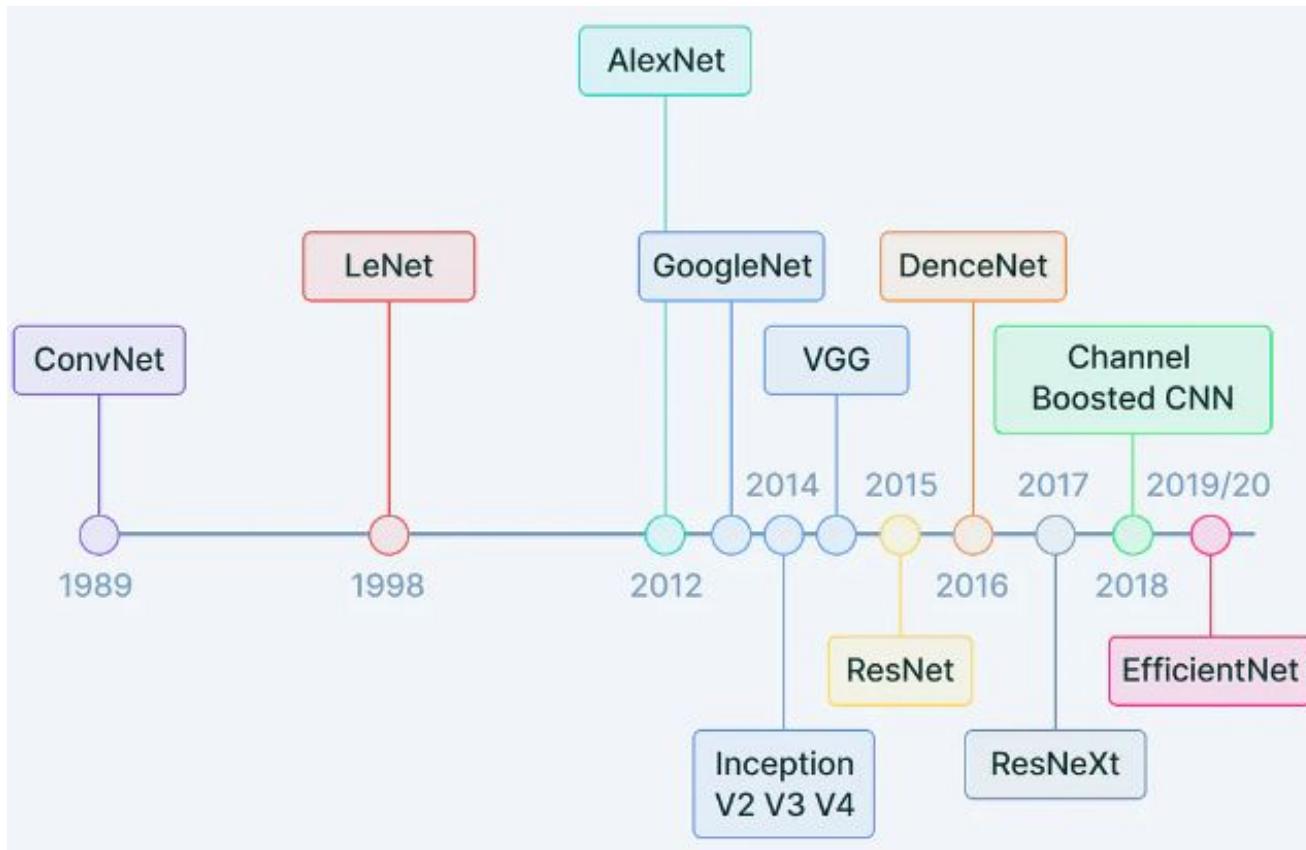
Common CNN's layer pattern (part 3)

Làm sao để thiết kế CNN architecture?

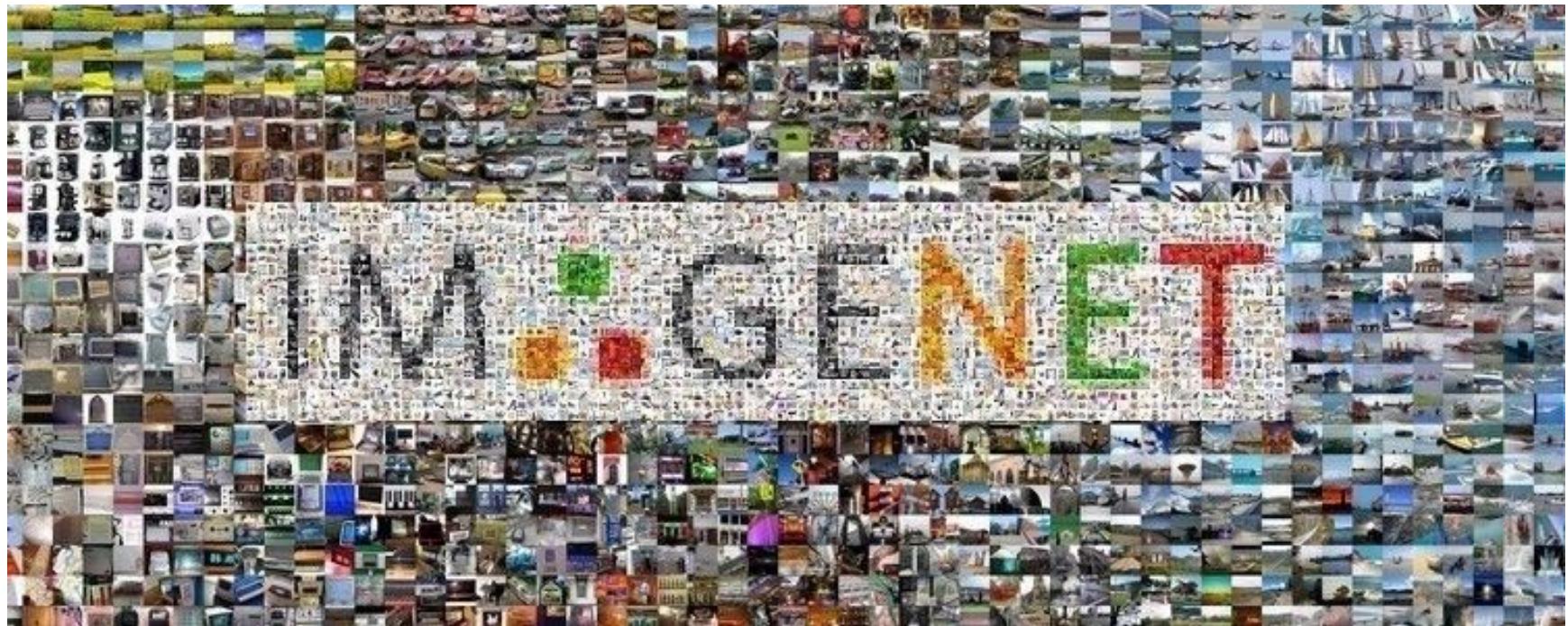
Rule 2: If rule 1 does not work, you could consider the following question

- Mô hình nên có bao nhiêu layer?
- Mỗi layer nên có parameter như thế nào?
 - Convolutional có bao nhiêu channel? Kernel size là bao nhiêu?
 - Nên dùng activation function gì?
 - Pooling nên có kernel size là bao nhiêu?
 - Nên flatten feature map ở đâu?
 - Nên có bao nhiêu fully connected layer?
 - Có nên dùng Batchnorm, Dropout, ...?

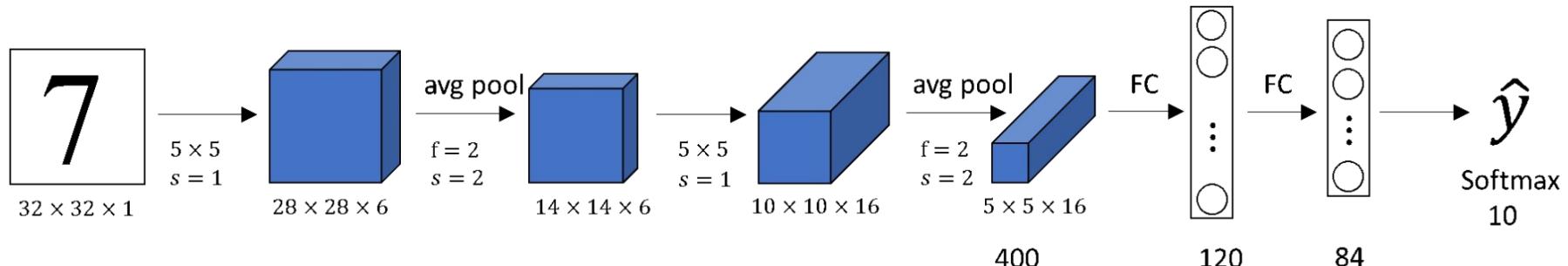
Common CNN architectures



ImageNet Large Scale Visual Recognition Challenge

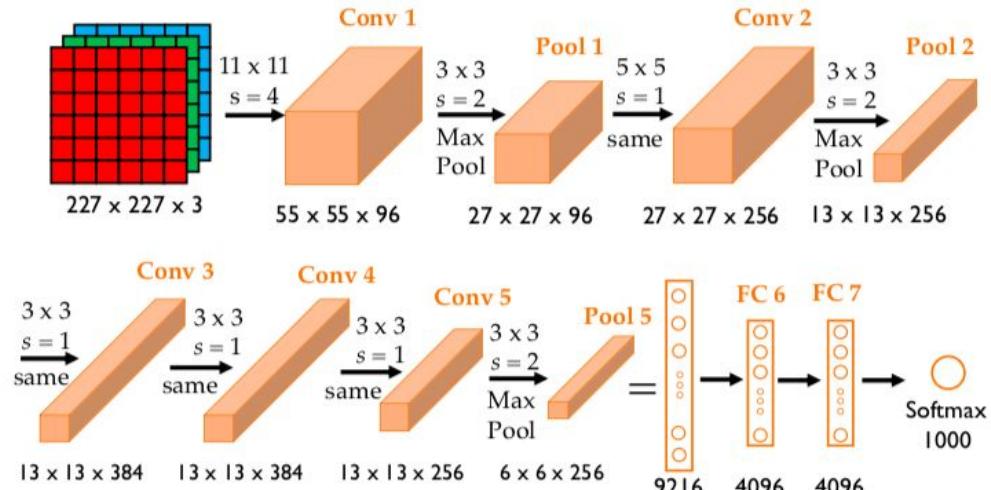


LeNet (1998)



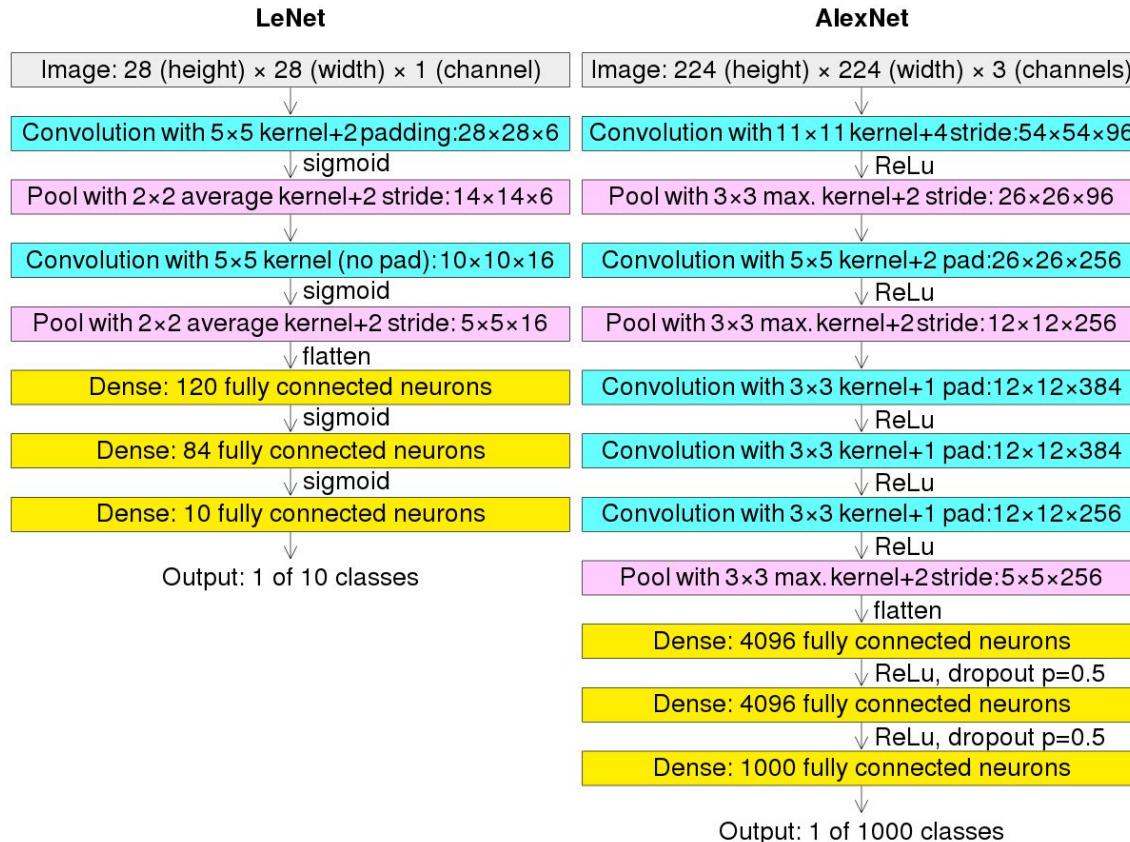
Layer	Feature Map	Size	Kernel Size	Stride	Activation	
Input	1	32x32	-	-	-	
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

AlexNet (2012)

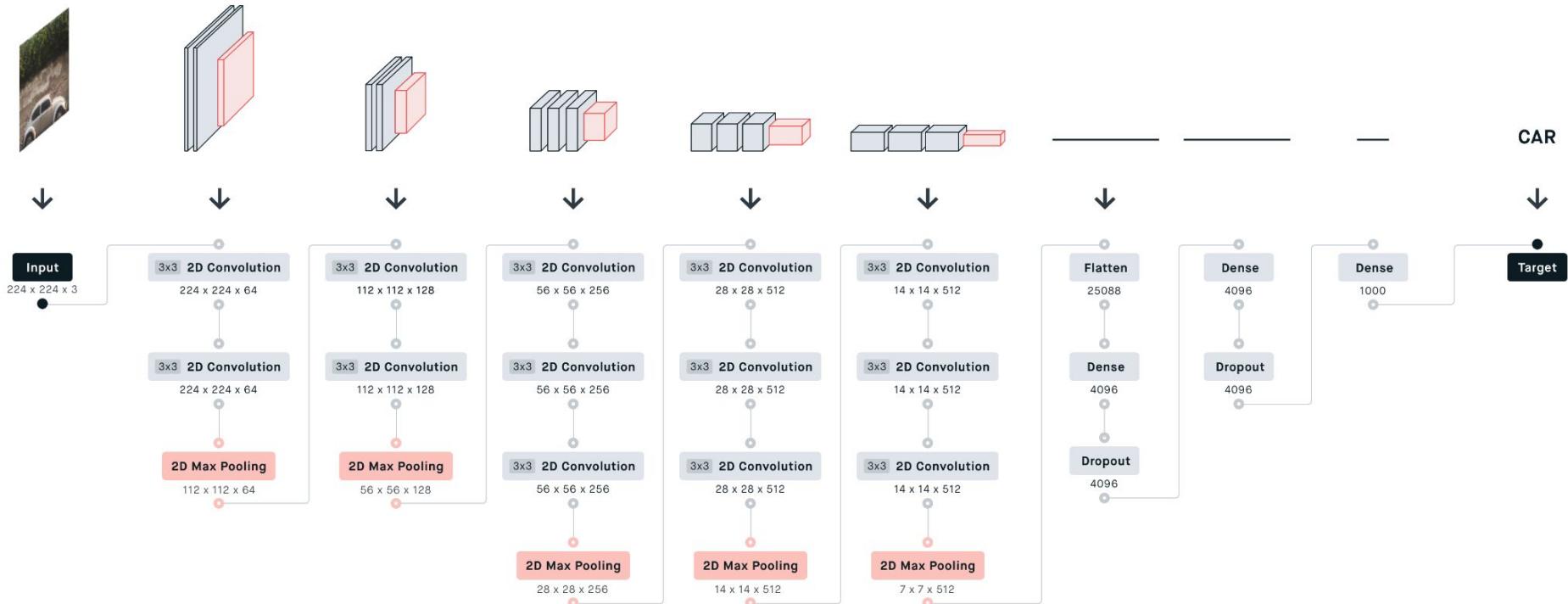


Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	227x227x3	-	-	-
1	Convolution	96	55 x 55 x 96	11x11	4	relu
	Max Pooling	96	27 x 27 x 96	3x3	2	relu
2	Convolution	256	27 x 27 x 256	5x5	1	relu
	Max Pooling	256	13 x 13 x 256	3x3	2	relu
3	Convolution	384	13 x 13 x 384	3x3	1	relu
4	Convolution	384	13 x 13 x 384	3x3	1	relu
5	Convolution	256	13 x 13 x 256	3x3	1	relu
	Max Pooling	256	6 x 6 x 256	3x3	2	relu
		-	9216	-	-	
7	FC	-	4096	-	-	relu
8	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

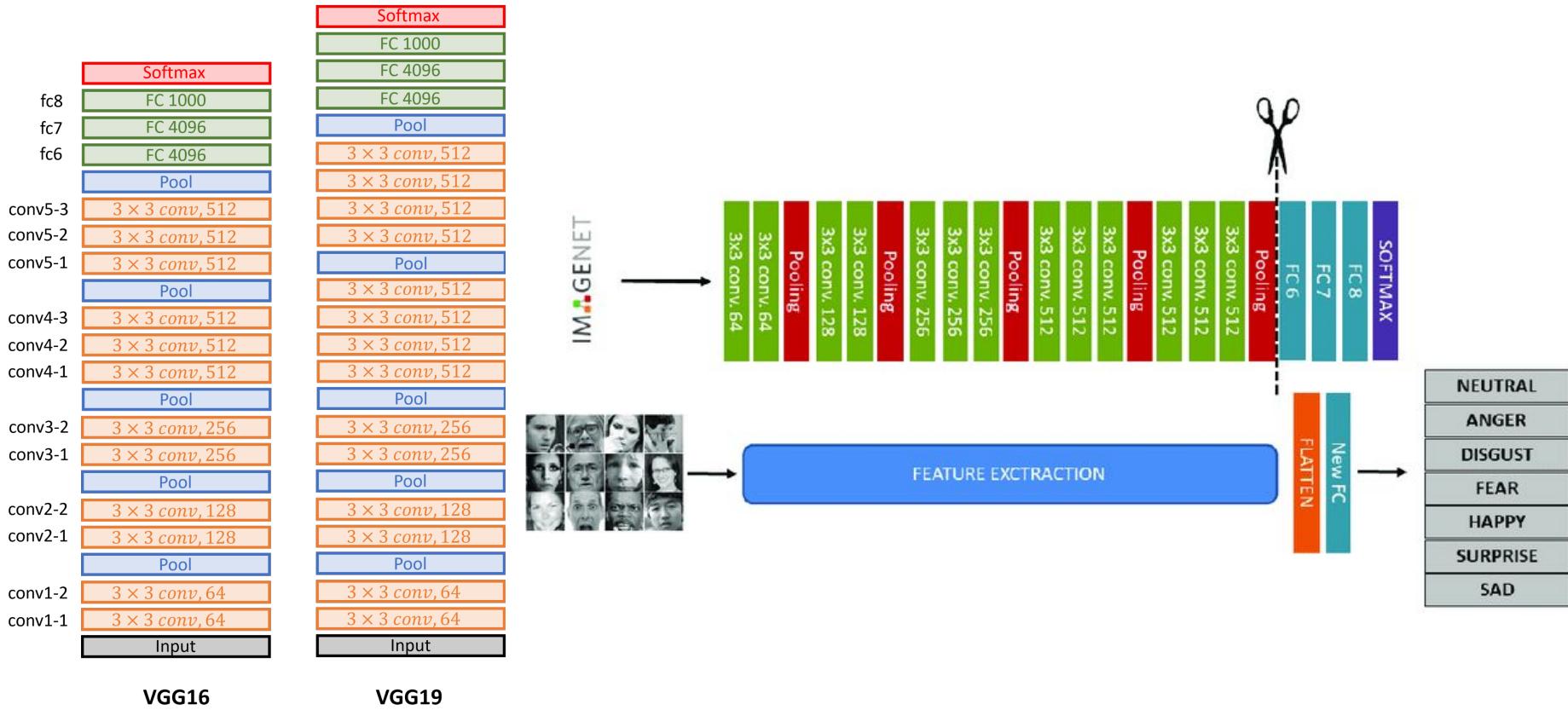
LeNet vs AlexNet



VGGNet (2014)

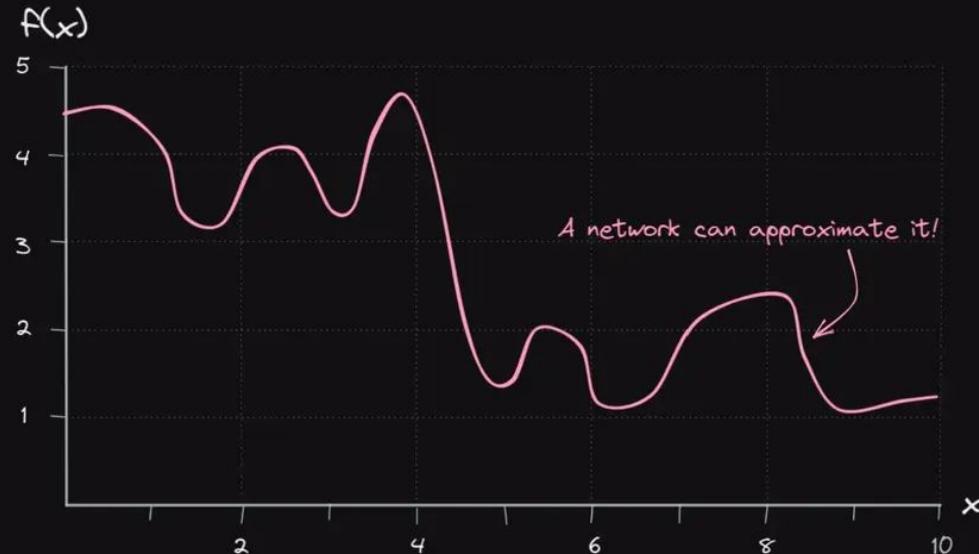


VGGNet (2014)



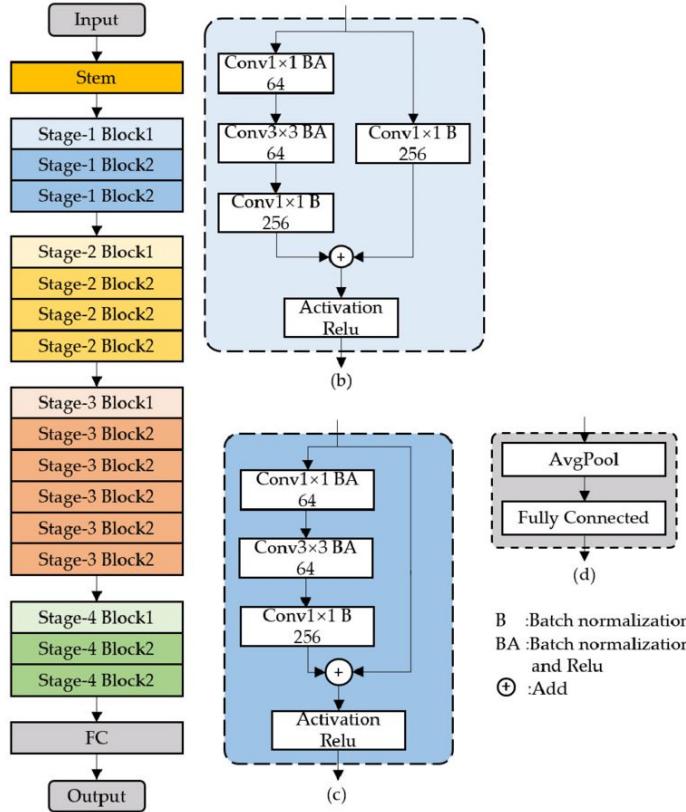
Universal Approximation Theorem

UNIVERSAL APPROXIMATION THEOREM



Regardless of how complex a general function may be, there is guaranteed to be an artificial neural network that can approximate this function as well as we'd like it to.

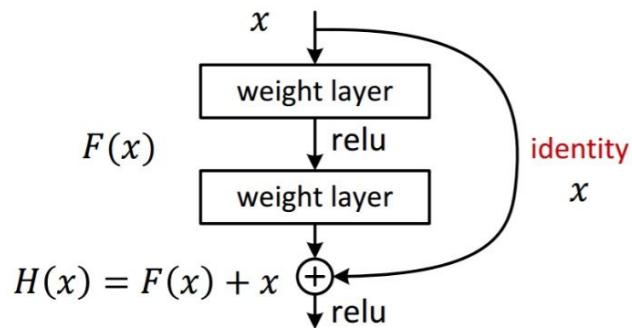
ResNet (2015)



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112			7×7, 64, stride 2		
conv2_x	56×56			3×3 max pool, stride 2		
conv3_x	28×28	$\left[\begin{array}{l} 3 \times 3, 64 \\ 3 \times 3, 64 \end{array} \right] \times 2$	$\left[\begin{array}{l} 3 \times 3, 64 \\ 3 \times 3, 64 \end{array} \right] \times 3$	$\left[\begin{array}{l} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{array} \right] \times 3$	$\left[\begin{array}{l} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{array} \right] \times 3$	$\left[\begin{array}{l} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{array} \right] \times 3$
conv4_x	14×14	$\left[\begin{array}{l} 3 \times 3, 128 \\ 3 \times 3, 128 \end{array} \right] \times 2$	$\left[\begin{array}{l} 3 \times 3, 128 \\ 3 \times 3, 128 \end{array} \right] \times 4$	$\left[\begin{array}{l} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{array} \right] \times 4$	$\left[\begin{array}{l} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{array} \right] \times 4$	$\left[\begin{array}{l} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{array} \right] \times 8$
conv5_x	7×7	$\left[\begin{array}{l} 3 \times 3, 256 \\ 3 \times 3, 256 \end{array} \right] \times 2$	$\left[\begin{array}{l} 3 \times 3, 256 \\ 3 \times 3, 256 \end{array} \right] \times 6$	$\left[\begin{array}{l} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{array} \right] \times 6$	$\left[\begin{array}{l} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{array} \right] \times 23$	$\left[\begin{array}{l} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{array} \right] \times 36$
	1×1			average pool, 1000-d fc, softmax		
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ResNet (2015)

- 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 $H(x) = F(x) + x$

$F(x)$ is obtained from x as follows.

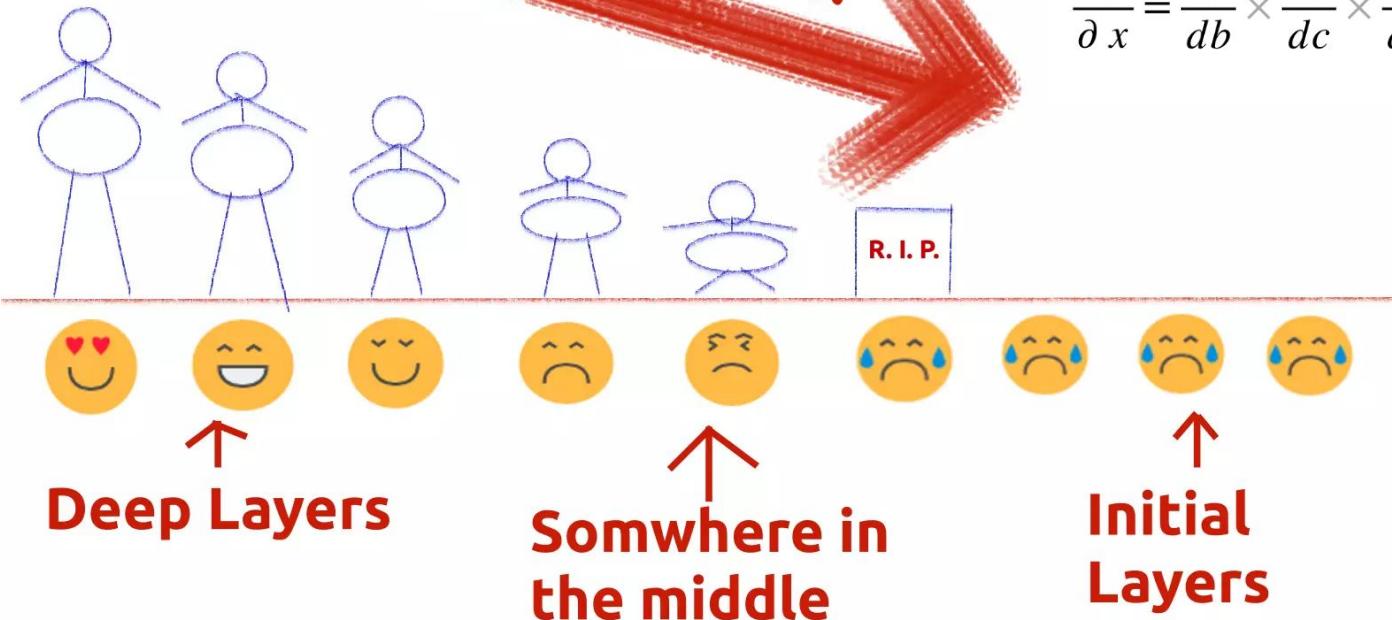
$x \rightarrow \text{weight}_1 \rightarrow \text{ReLU} \rightarrow \text{weight}_2$

$H(x)$ is obtained from $F(x)$ as follows.

$F(x) + x \rightarrow \text{ReLU}$

Problems in Neural Network: Vanishing gradients

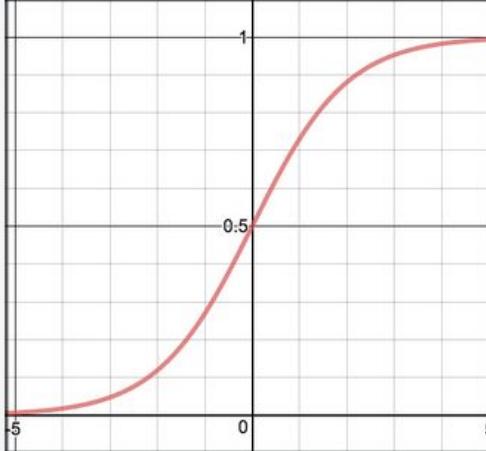
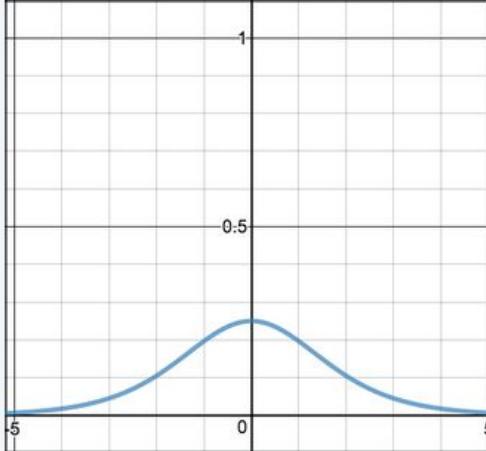
Vanishing Gradient



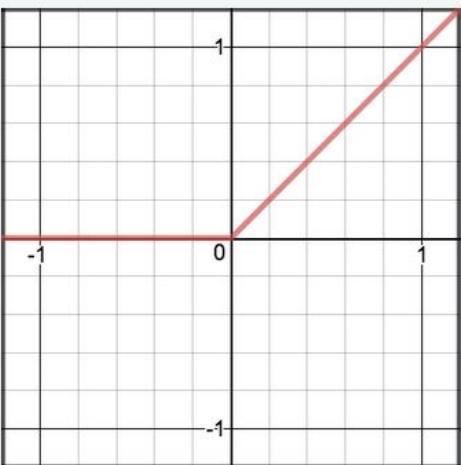
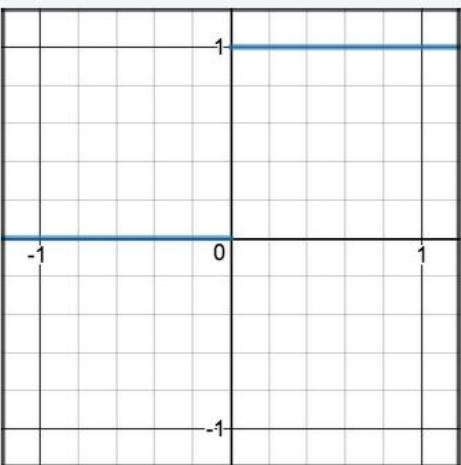
$$\frac{da}{dx} = a(b(c(d(e(f(g(x)))))))$$

$$\frac{da}{\partial x} = \frac{da}{db} \times \frac{db}{dc} \times \frac{dc}{dd} \times \frac{dd}{de} \times \frac{de}{df} \times \frac{df}{dg} \times \frac{dg}{dx}$$

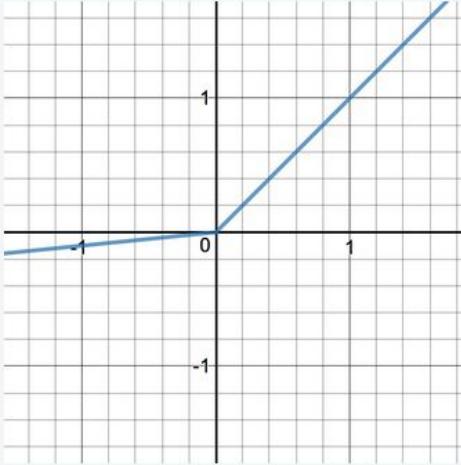
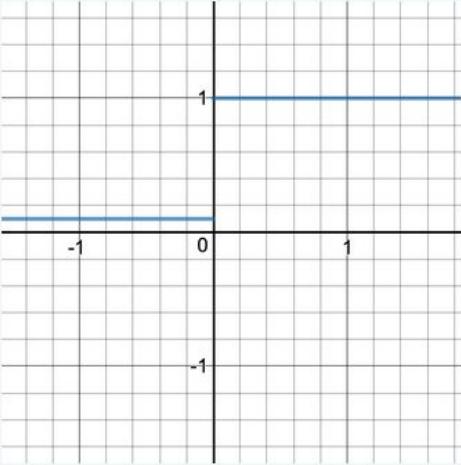
Sigmoid vs Vanishing gradients

Function	Derivative
$S(z) = \frac{1}{1 + e^{-z}}$	$S'(z) = S(z) \cdot (1 - S(z))$
	
<pre>def sigmoid(z): return 1.0 / (1 + np.exp(-z))</pre>	<pre>def sigmoid_prime(z): return sigmoid(z) * (1-sigmoid(z))</pre>

ReLU vs Vanishing gradients

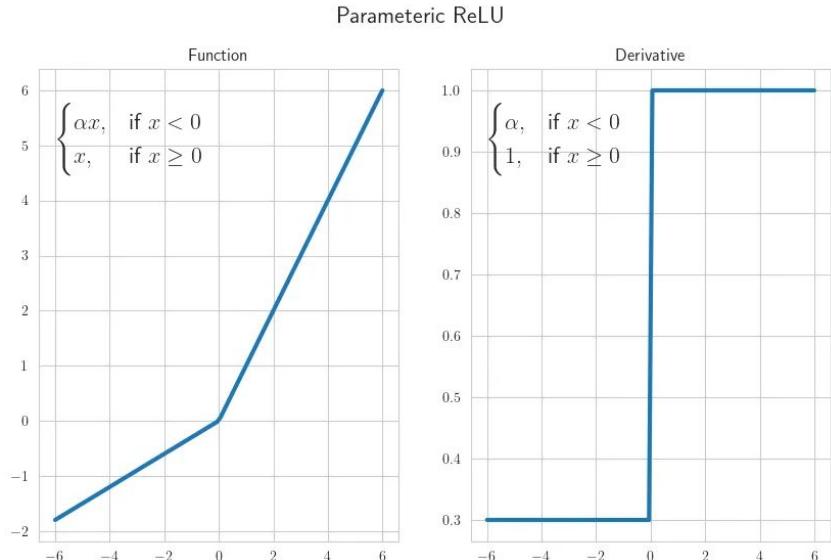
Function	Derivative
$R(z) = \begin{cases} z & z > 0 \\ 0 & z \leq 0 \end{cases}$	$R'(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$
	
<pre>def relu(z): return max(0, z)</pre>	<pre>def relu_prime(z): return 1 if z > 0 else 0</pre>

Leaky ReLU vs Vanishing gradients

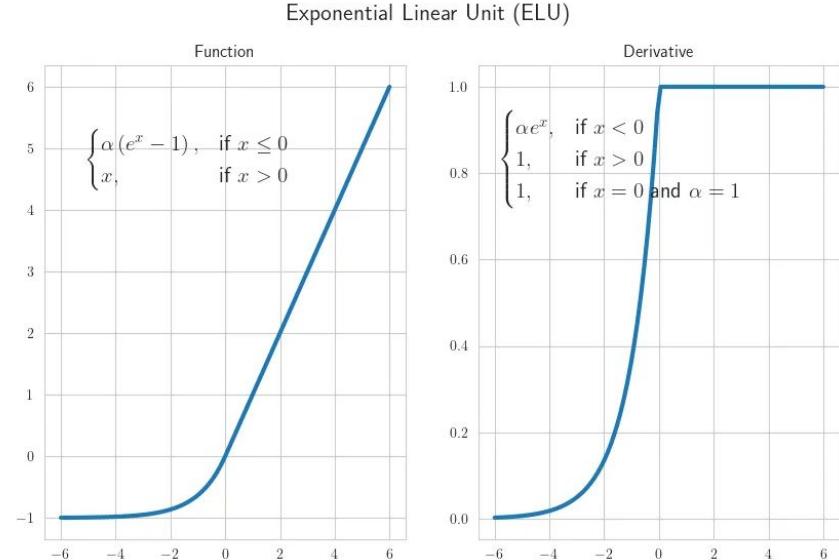
Function	Derivative
$R(z) = \begin{cases} z & z > 0 \\ \alpha z & z \leq 0 \end{cases}$	$R'(z) = \begin{cases} 1 & z > 0 \\ \alpha & z \leq 0 \end{cases}$
	
<pre>def leakyrelu(z, alpha): return max(alpha * z, z)</pre>	<pre>def leakyrelu_prime(z, alpha): return 1 if z > 0 else alpha</pre>

Các biến thể khác của ReLU

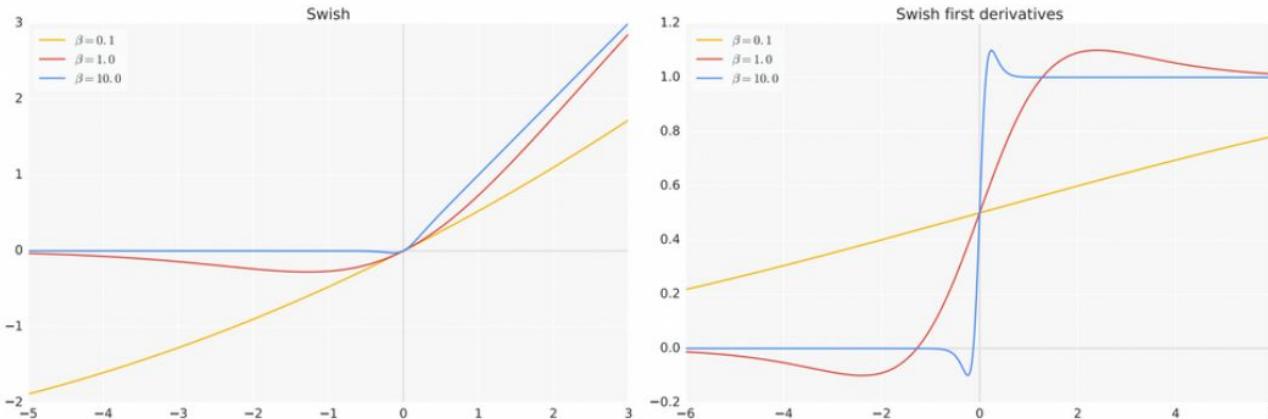
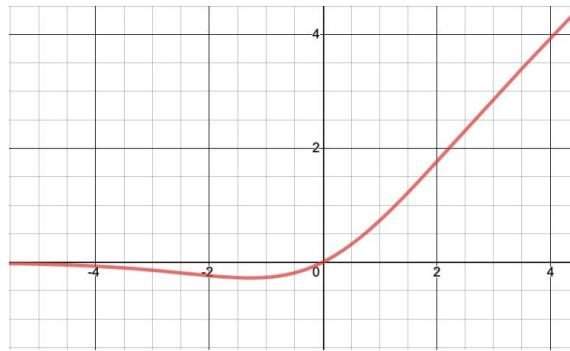
Parametric ReLU



Exponential Linear Unit (ELU)



SWISH



Problems in Neural Network: Exploding gradients

Ngược lại với vanishing gradient

Dấu hiệu

- Model's parameter tăng rất nhanh (exponential growth)
- Model's parameter có thể thành NaN trong quá trình training
- Loss trở thành NaN trong quá trình training
- Loss lớn dù train lâu

Nguyên nhân

- Improper initial weights => large weight update
- Poor initial learning rate
- Poor loss function => large loss value => large weight update

Solutions for Vanishing/Exploding gradients

- Proper activation functions, like ReLU and its derivative
- Batch Normalization
- Regularization
- Proper weight initialization
- Gradient clipping (against exploding gradients)
 - By values
 - By norm

Weight initialization

- Performance của CNN/NN phụ thuộc vào weight initialization
- Các lợi ích của good weight initialization:
 - Reproducible model
 - Model converge faster
 - Tránh được vanishing/exploding gradients
- Các cách phổ biến:
 - Zero/Constant/Random initialization

Weight initialization: Zero/Constant initialization

- Tất cả các parameter đều được khởi tạo là 0 hoặc 1 giá trị cố định
- Tất cả các node sẽ học theo cùng 1 cách/1 kiểu => fail to break symmetry
- Luôn dẫn đến kết quả tồi



Weight initialization: Random initialization

- Giải quyết được vấn đề break the symmetry
- Có thể dẫn đến vấn đề mới: **Vanishing/exploding gradients**
- Random initialization from a distribution:
 - Random Normal
 - Random Uniform

Weight initialization: Xavier/Glorot initialization

- Được áp dụng cho các layers (fully connected layers, Conv layers) mà sử dụng activation function là sigmoid hay tanh
- Có 2 phiên bản:

Uniform Xavier initialization: draw each weight, w , from a random uniform distribution

$$\text{in } [-x, x] \text{ for } x = \sqrt{\frac{6}{\text{inputs} + \text{outputs}}}$$

Normal Xavier initialization: draw each weight, w , from a normal distribution with a mean

of 0, and a standard deviation $\sigma = \sqrt{\frac{2}{\text{inputs} + \text{outputs}}}$

Weight initialization: He/Kaiming initialization

- Được áp dụng cho các layers (fully connected layers, Conv layers) mà sử dụng activation function là **ReLU** và các biến thể của nó
- Có 2 phiên bản:
 - He Normal initialization

$$W \sim N(0, Var(W))$$

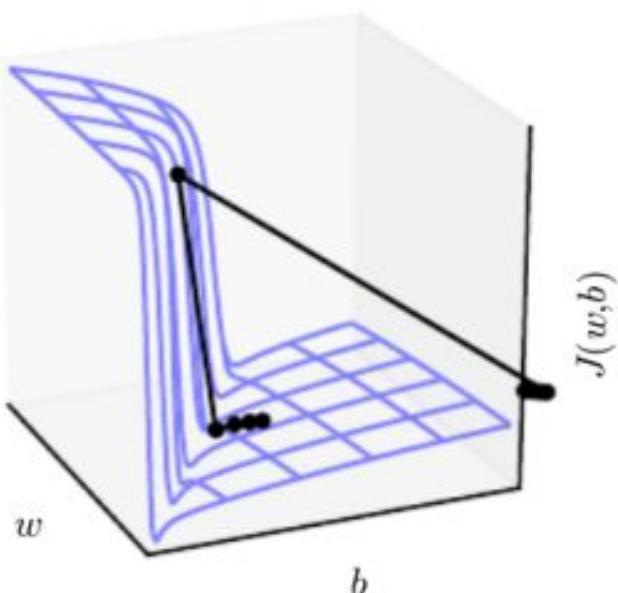
$$Var(W) = \sqrt{\frac{2}{n_{in}}}$$

- He Uniform initialization

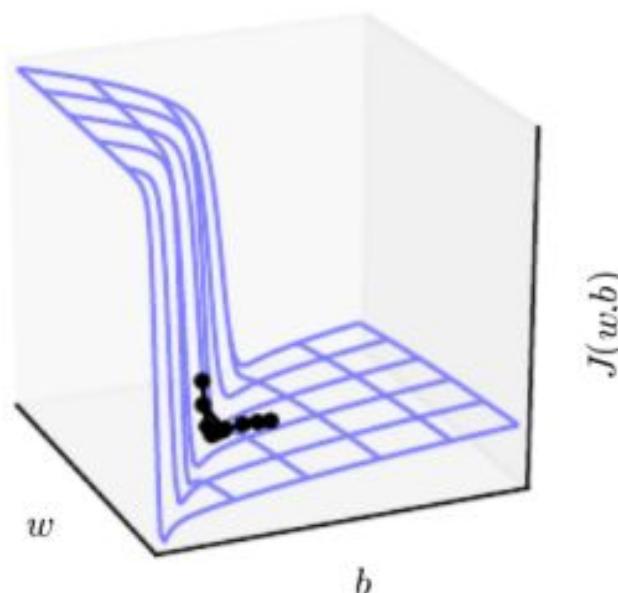
$$W \sim U\left(-\sqrt{\frac{6}{n_{in}}}, +\sqrt{\frac{6}{n_{in}}}\right)$$

Gradient clipping

Without clipping

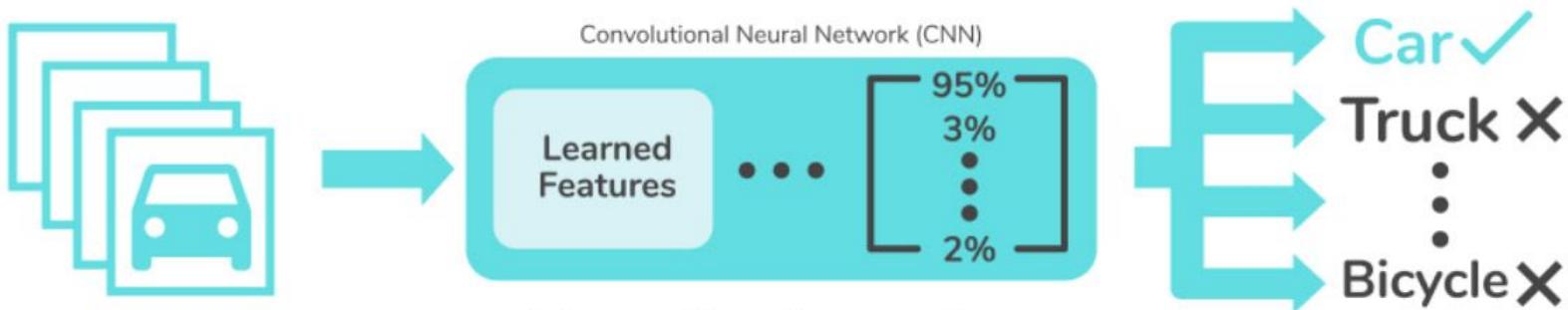


With clipping



Transfer Learning

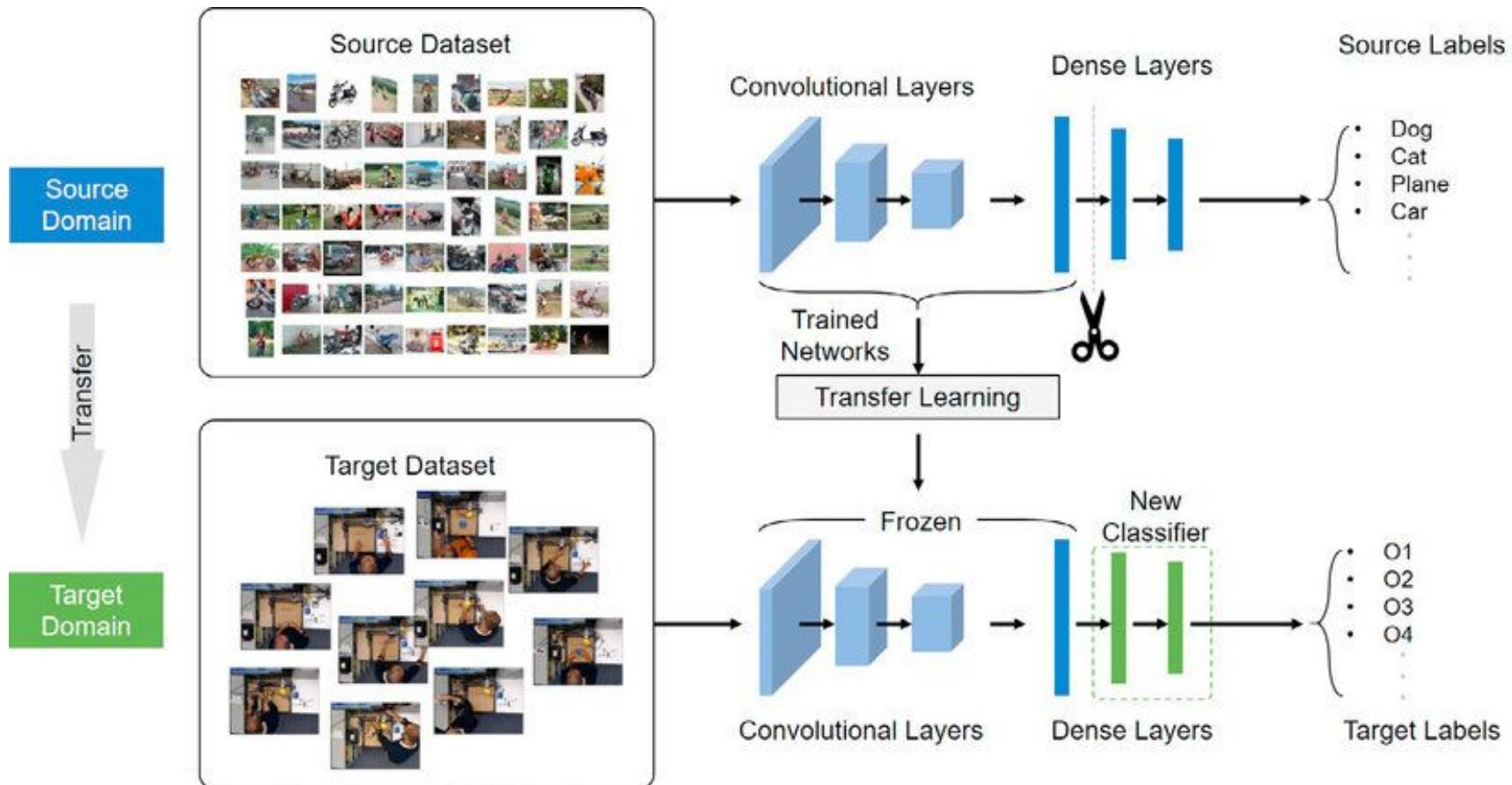
Training from Scratch



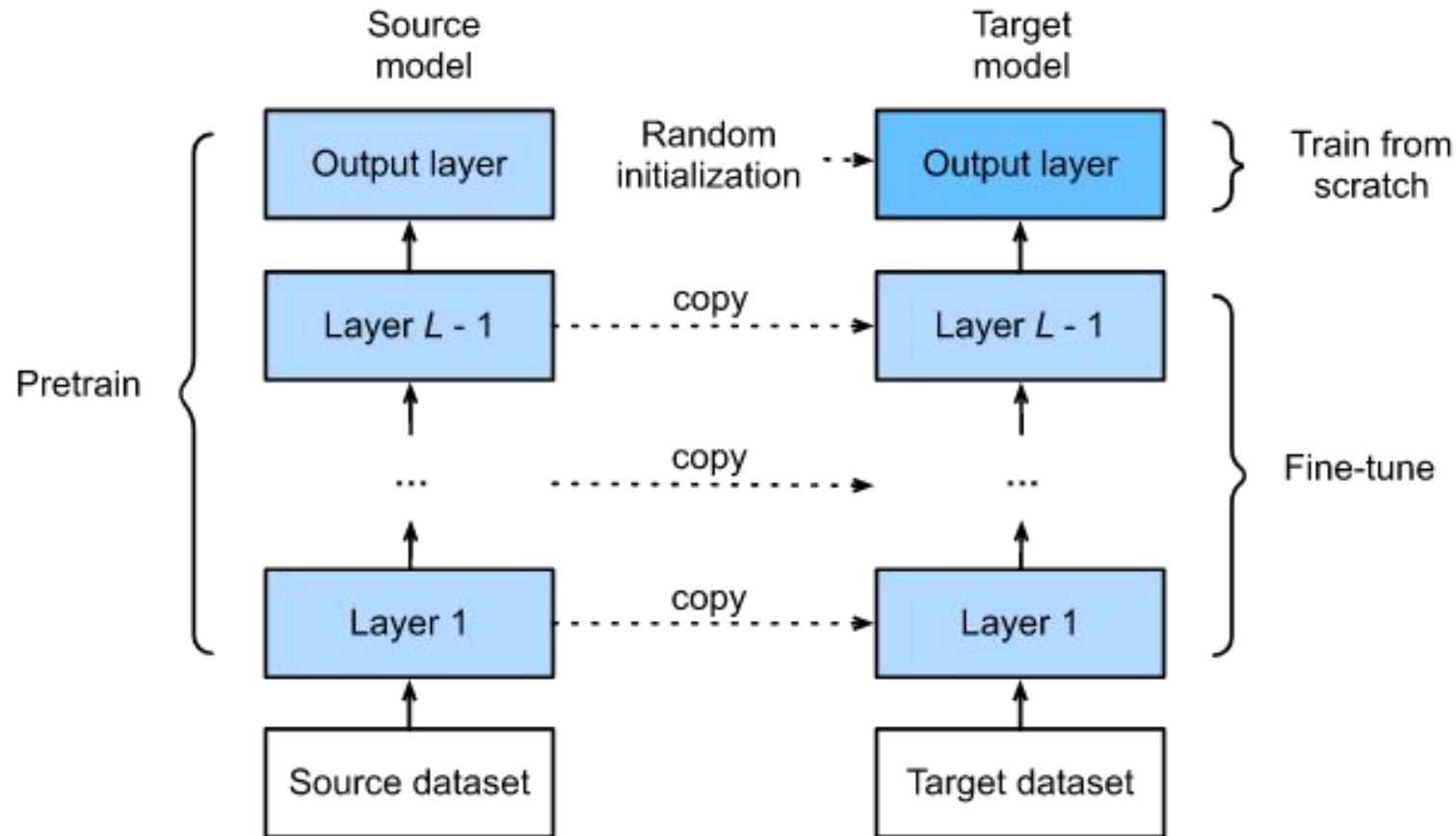
Transfer Learning



Transfer Learning: Use CNN as fixed feature extractor



Transfer Learning: Fine-tune CNN



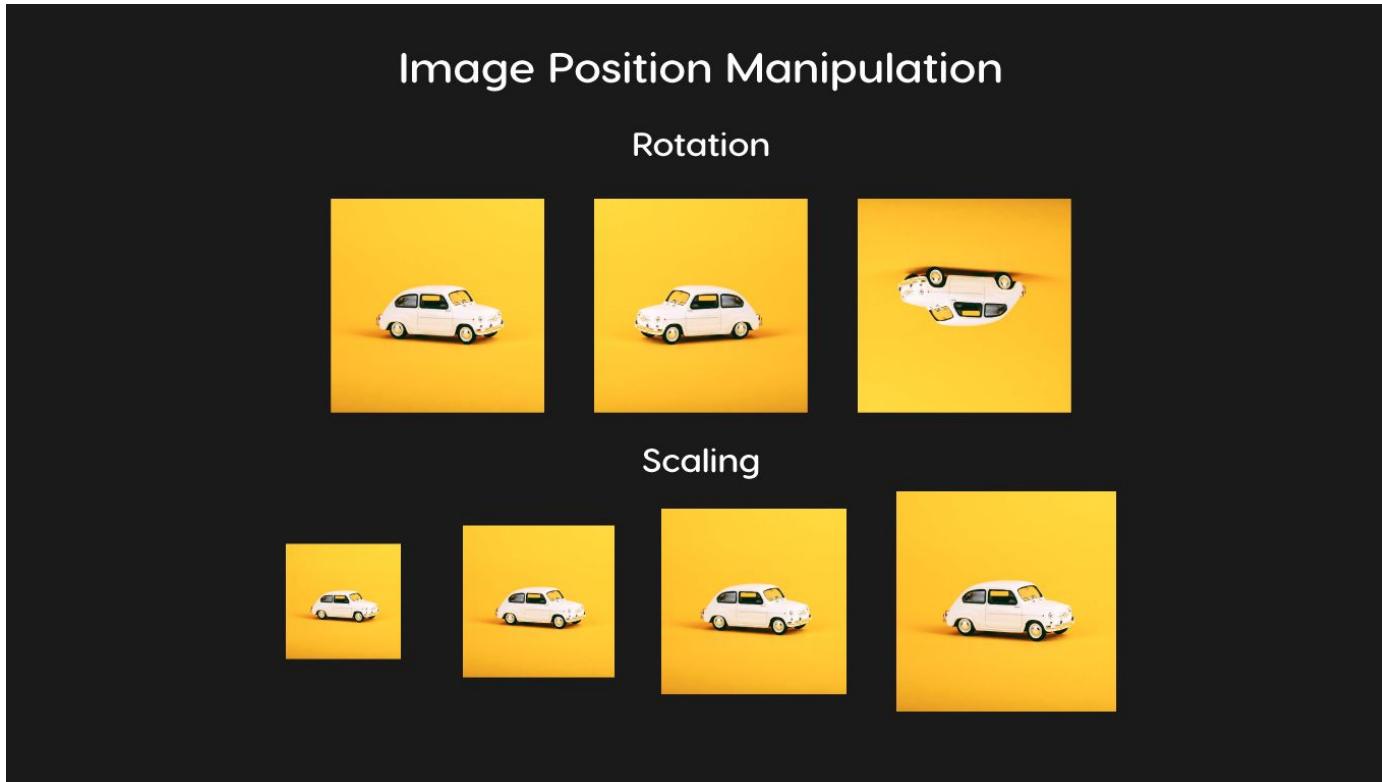
Transfer Learning: How to choose approach

	Similar dataset	Different dataset
Small dataset	Transfer learning: highest level features + classifier	Transfer learning: lower level features + classifier
Large dataset	Fine-tune*	Fine-tune*

Data augmentation

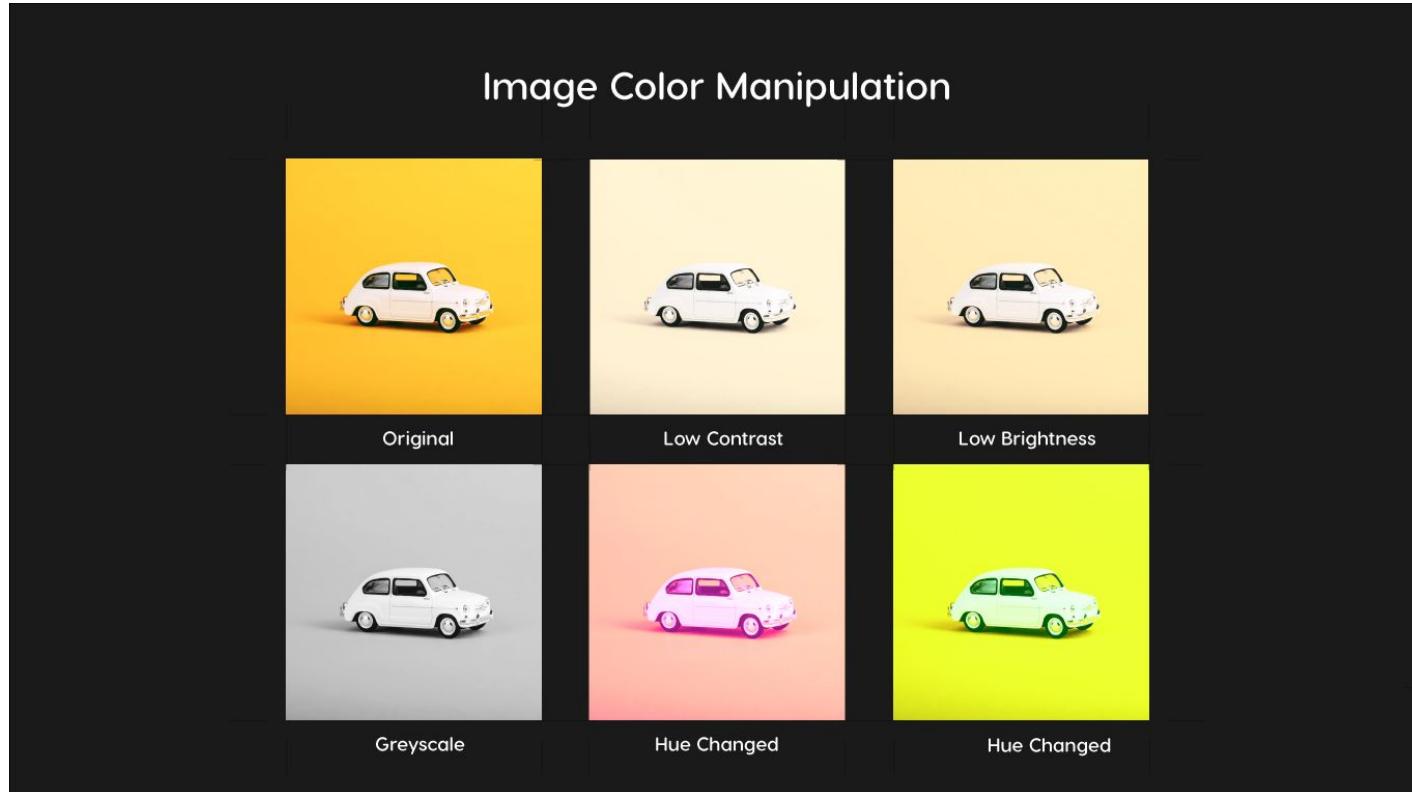


Data augmentation: Position manipulation



Another name: Geometric transformations

Data augmentation: Color manipulation



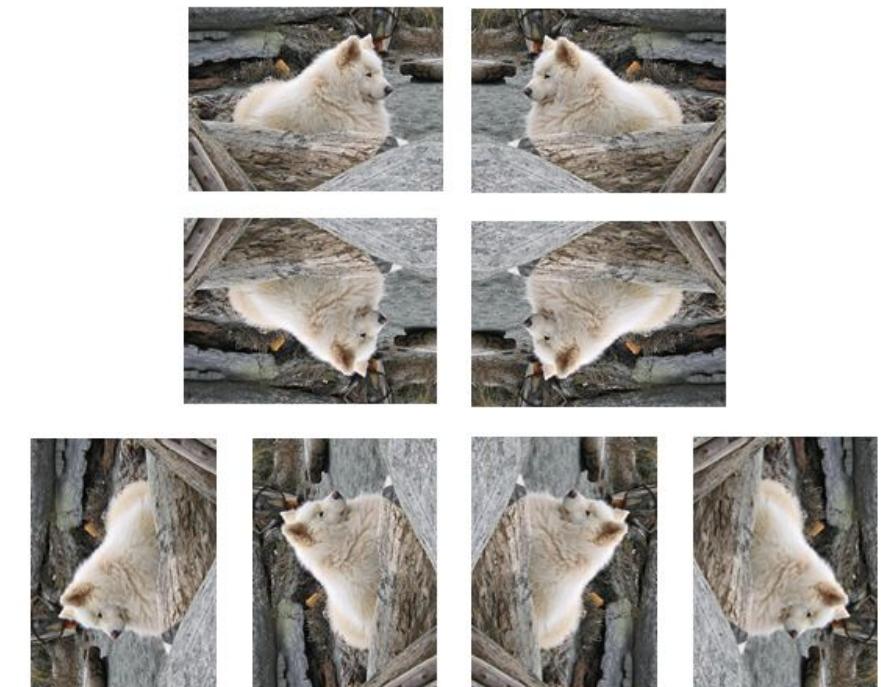
Another name: Color distortion

Data augmentation: Example

Brightness, Contrast, and Color Changes



Horizontal and Vertical Flips



Data augmentation: Example

Random Crops



Deformations



Object Detection

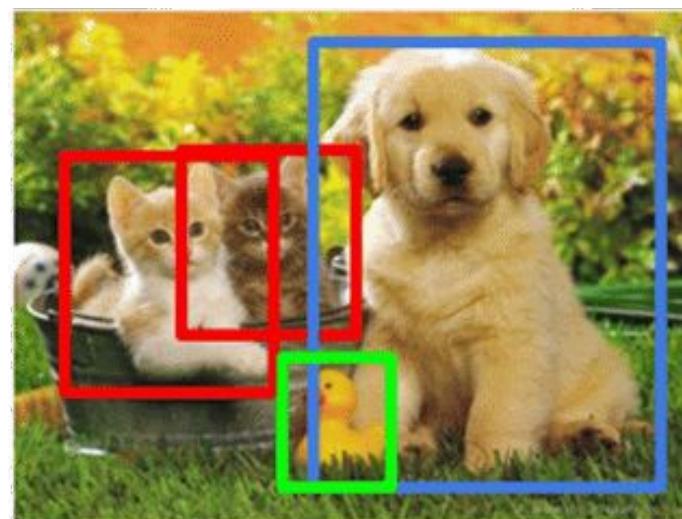
Image Classification vs Object Detection

Classification



CAT

Object Detection



CAT, DOG, DUCK

Object Detection

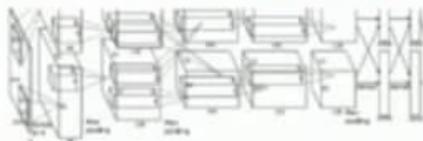
Input: a single RGB image



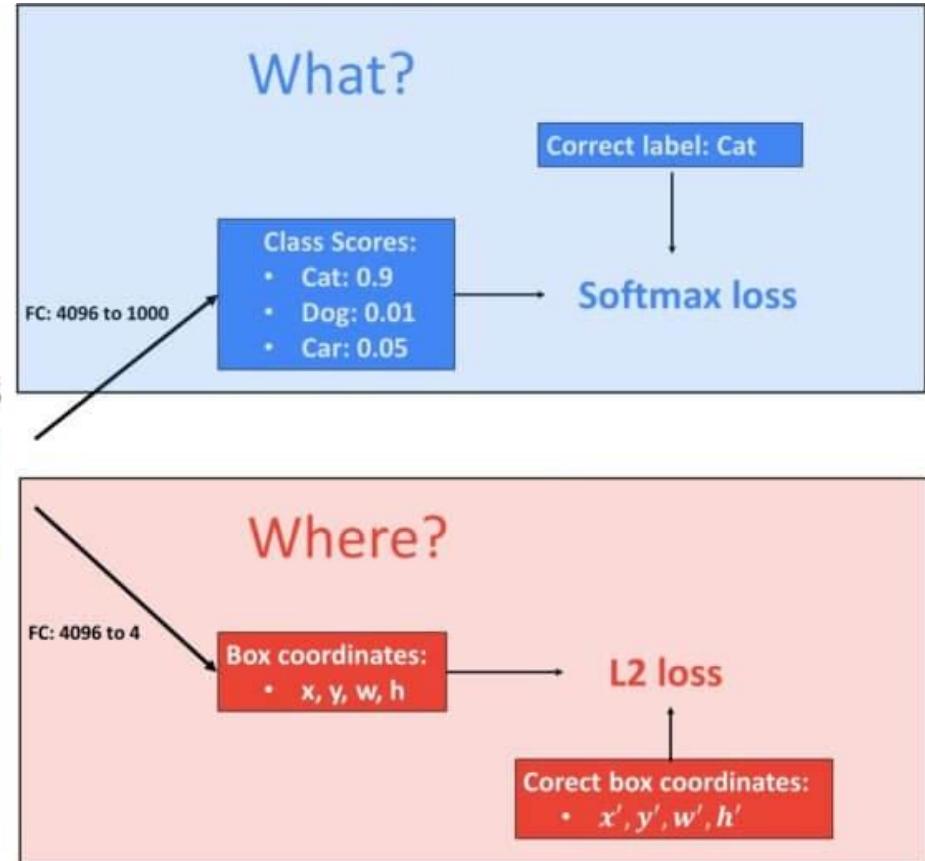
Output: a set of detected objects



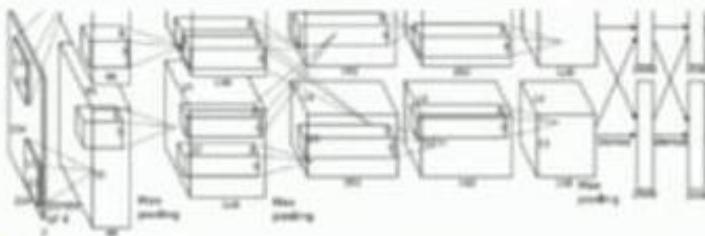
Object Detection - Single Object



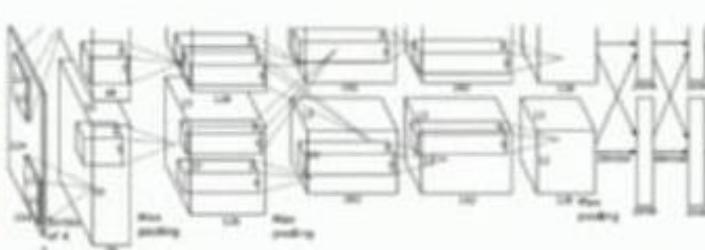
Vector: 4096



Object Detection - Multiple Objects



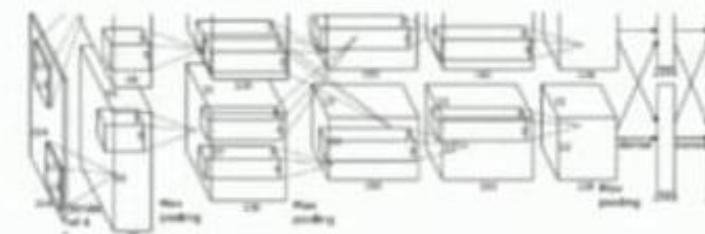
CAT: (x, y, w, h)



DOG: (x, y, w, h)

DOG: (x, y, w, h)

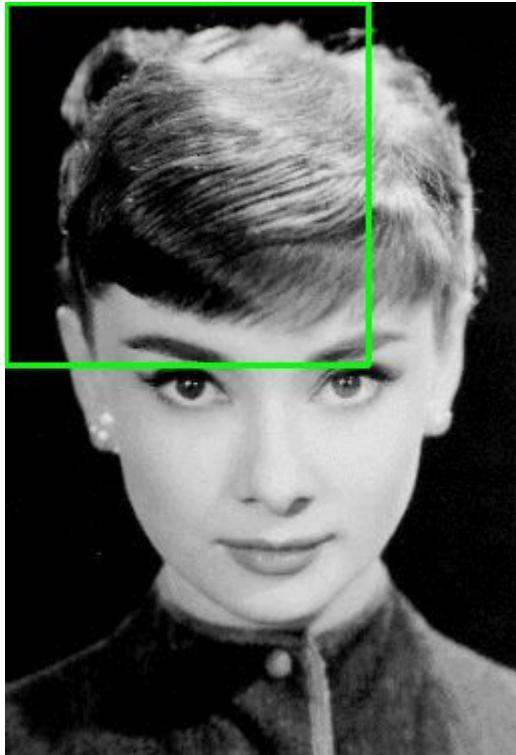
CAT: (x, y, w, h)



DUCK: (x, y, w, h)

DUCK: (x, y, w, h)

Sliding Windows

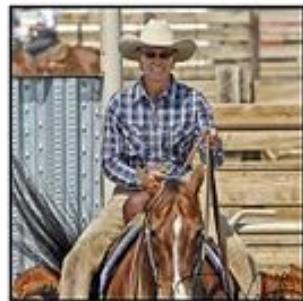


Selective Search

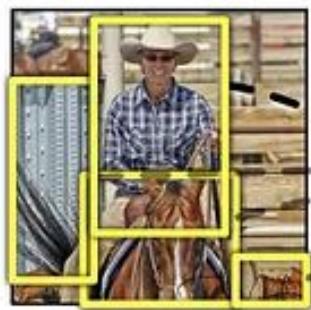


R-CNN \rightarrow Fast R-CNN \rightarrow Faster R-CNN

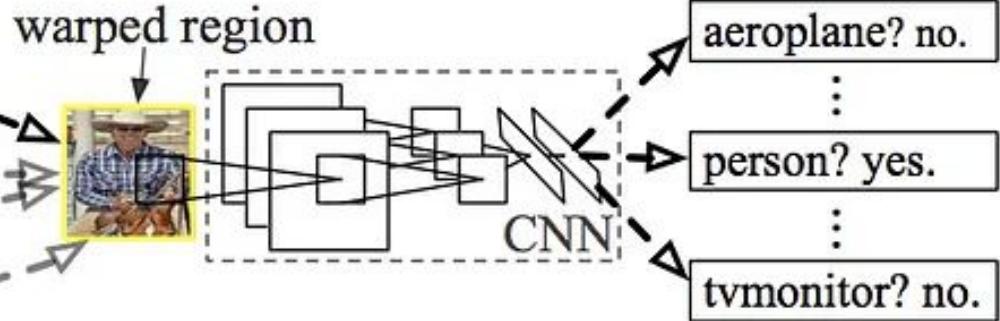
R-CNN: *Regions with CNN features*



1. Input image



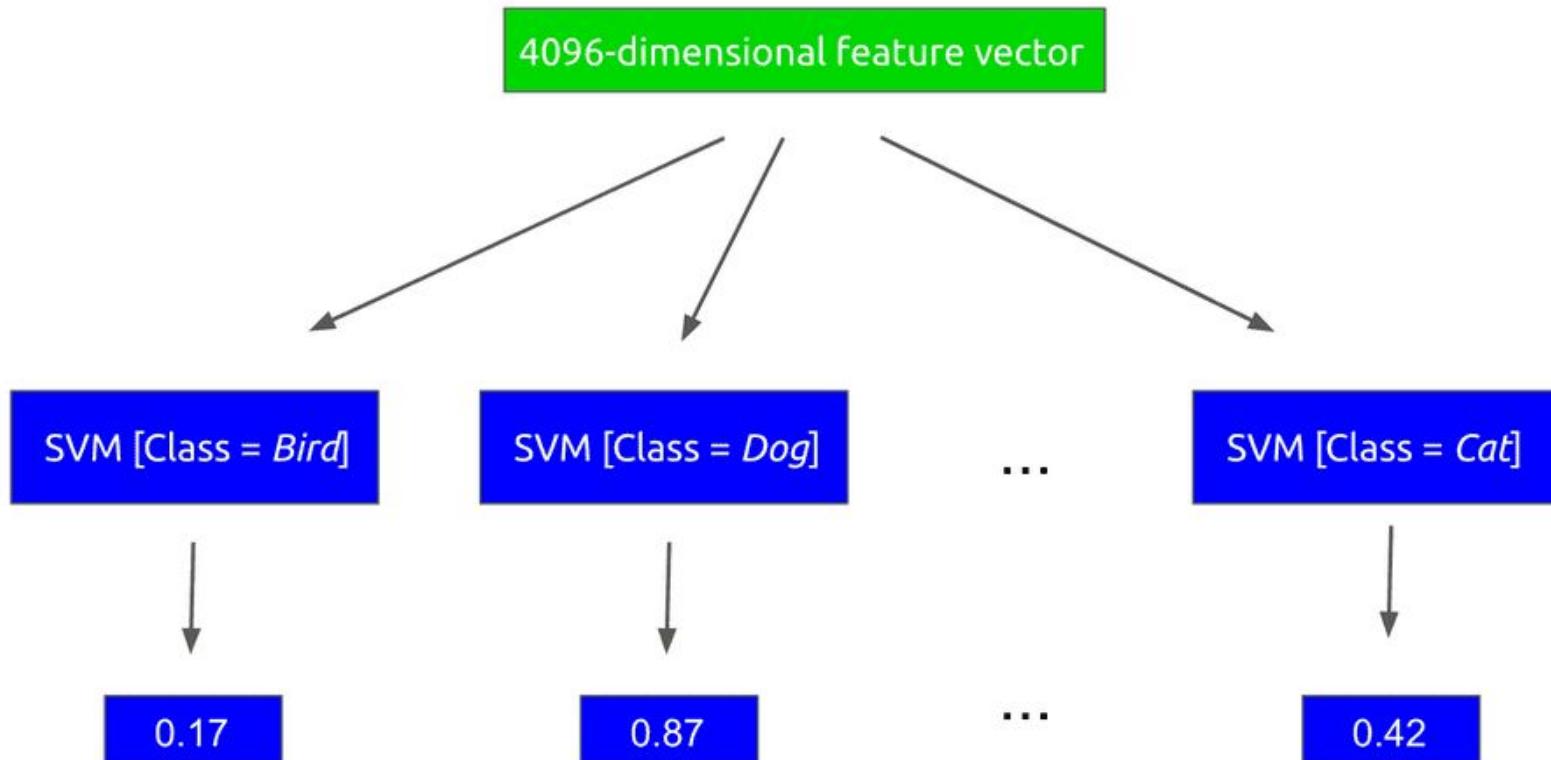
2. Extract region proposals ($\sim 2k$)



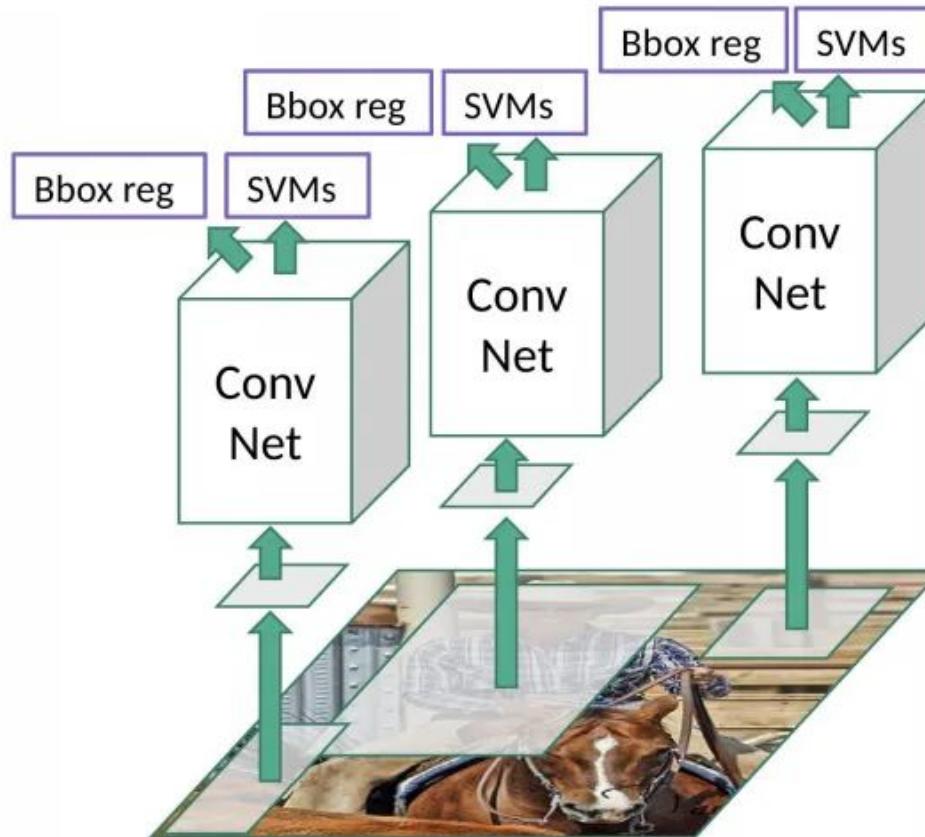
3. Compute CNN features

4. Classify regions

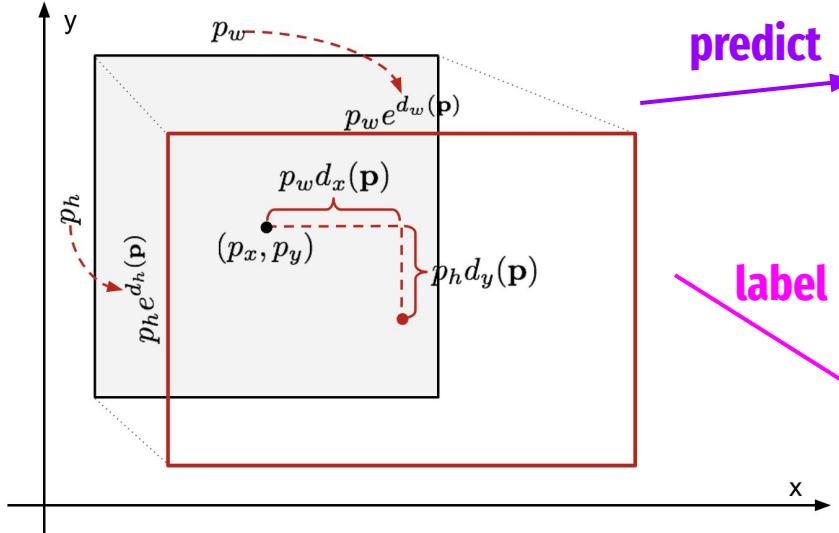
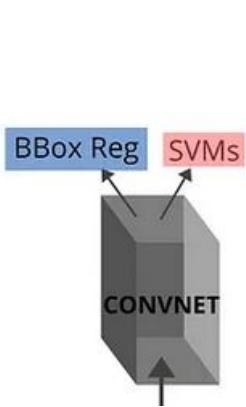
R-CNN \rightarrow Fast R-CNN \rightarrow Faster R-CNN



R-CNN → Fast R-CNN → Faster R-CNN



Bounding box regression



$$\hat{g}_x = p_w d_x(\mathbf{p}) + p_x$$

$$\hat{g}_y = p_h d_y(\mathbf{p}) + p_y$$

$$\hat{g}_w = p_w \exp(d_w(\mathbf{p}))$$

$$\hat{g}_h = p_h \exp(d_h(\mathbf{p}))$$

$$t_x = (g_x - p_x)/p_w$$

$$t_y = (g_y - p_y)/p_h$$

$$t_w = \log(g_w/p_w)$$

$$t_h = \log(g_h/p_h)$$

p: region proposals (anchors)

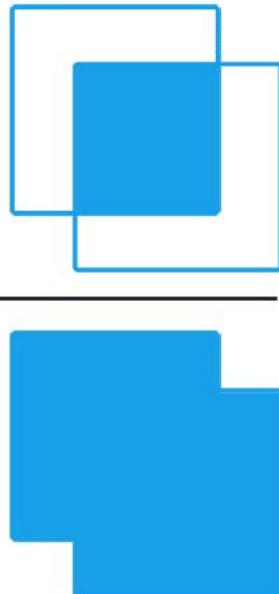
g: actual bbox

ĝ: predicted bbox

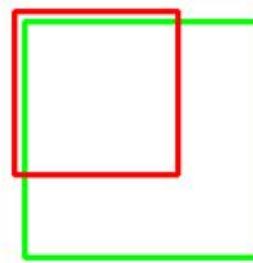
$$\mathcal{L}_{\text{reg}} = \sum_{i \in \{x, y, w, h\}} (t_i - d_i(\mathbf{p}))^2 + \lambda \|\mathbf{w}\|^2$$

Intersection over Union

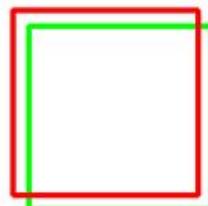
$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



IoU: 0.4034

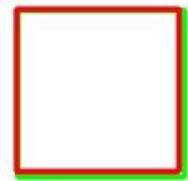


IoU: 0.7330



Poor

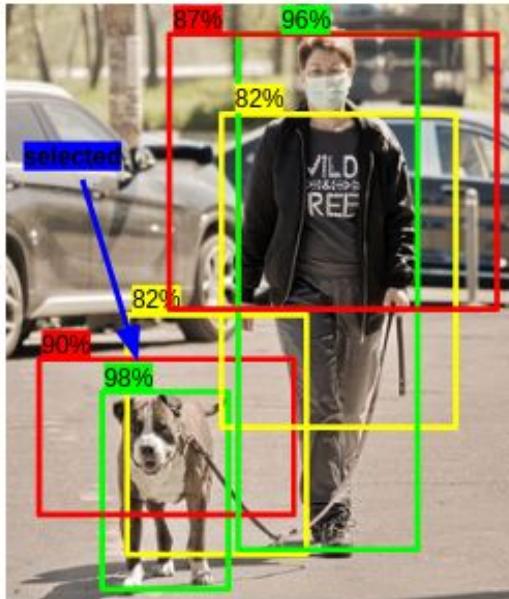
IoU: 0.9264



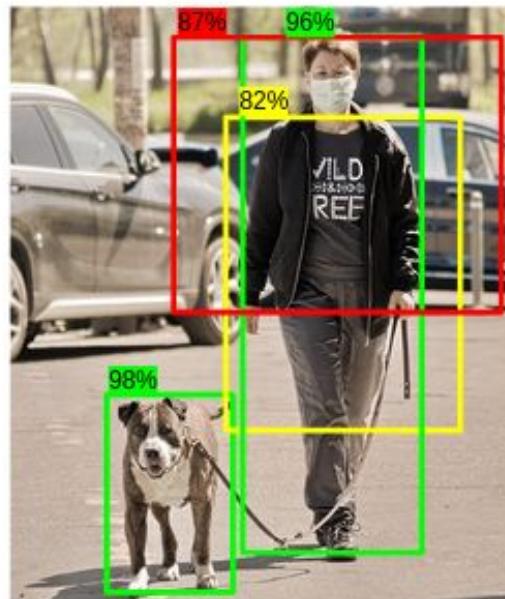
Good

Excellent

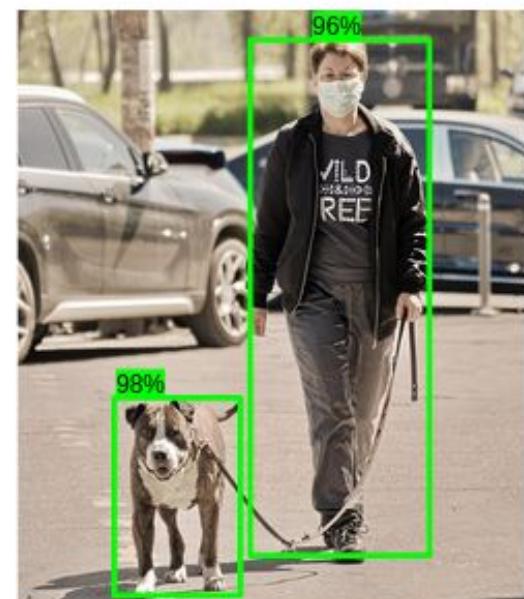
Non-maximum Suppression



Step 1: Selecting Bounding box with highest score



Step 3: Delete Bounding box with high overlap



Step 5: Final Output

Mean Average Precision



All ground-truth dog boxes

Precision = $1/1 = 1.0$

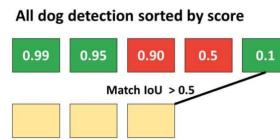
Recall = $1/3 = 0.33$



All ground-truth dog boxes

Precision = $2/2 = 1.0$

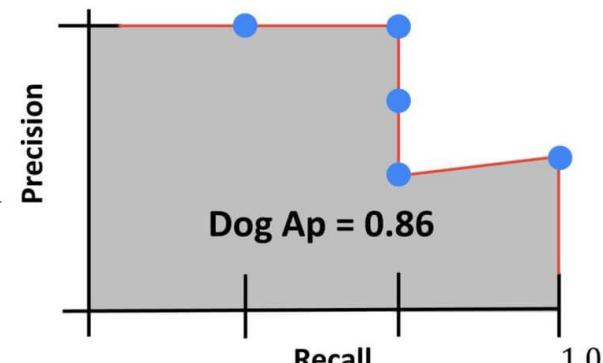
Recall = $2/3 = 0.67$



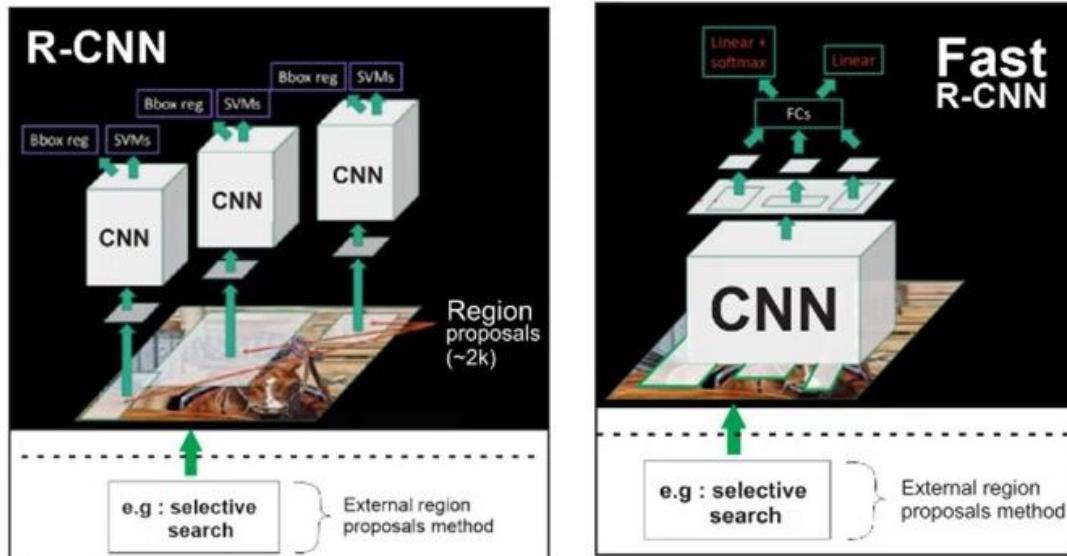
All ground-truth dog boxes

Precision = $3/5 = 0.6$

Recall = $3/3 = 1.0$

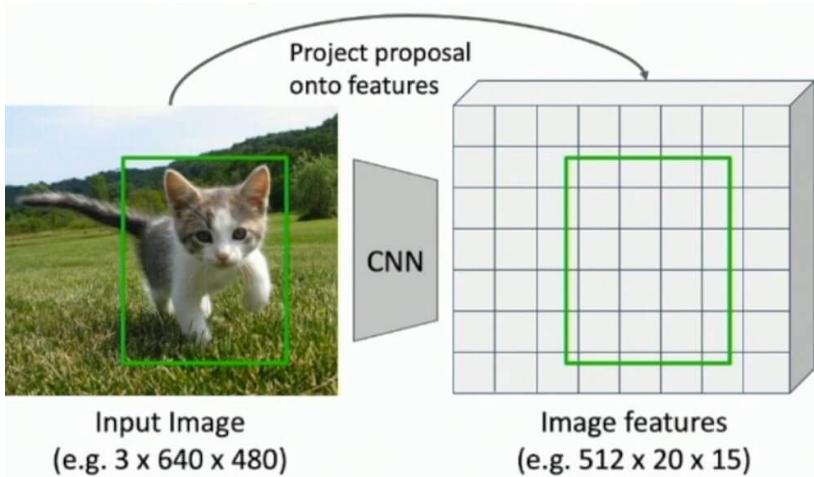


R-CNN → Fast R-CNN → Faster R-CNN

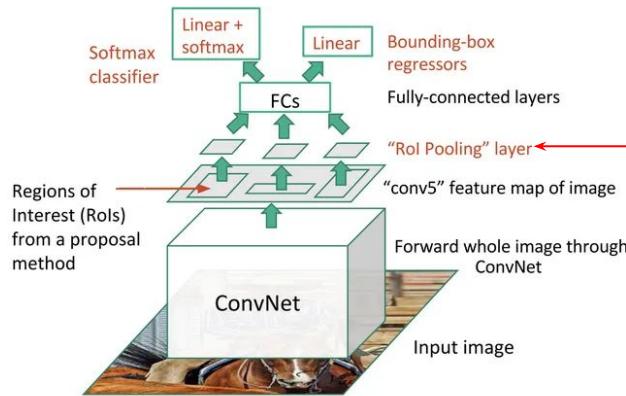
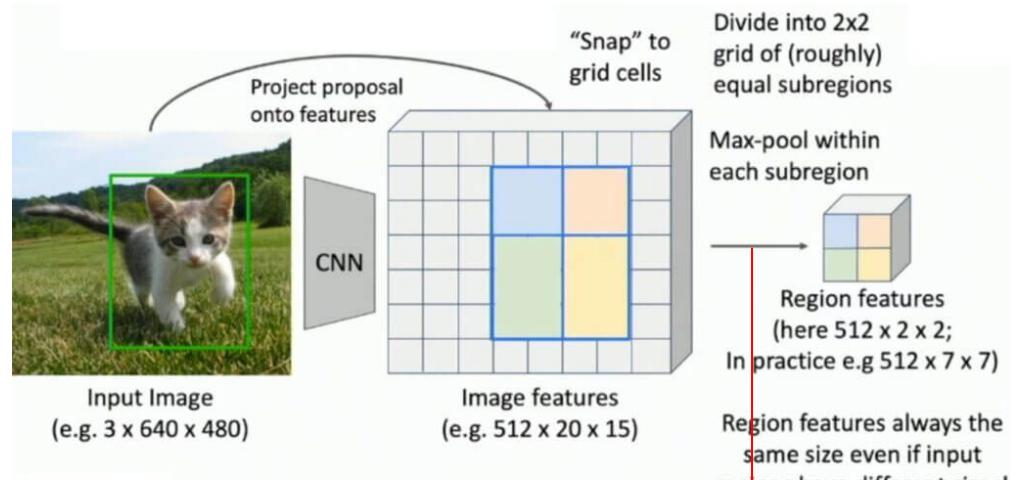


	R-CNN	Fast R-CNN
Test time per image	50 seconds	2 seconds
Speed-up	1x	25x

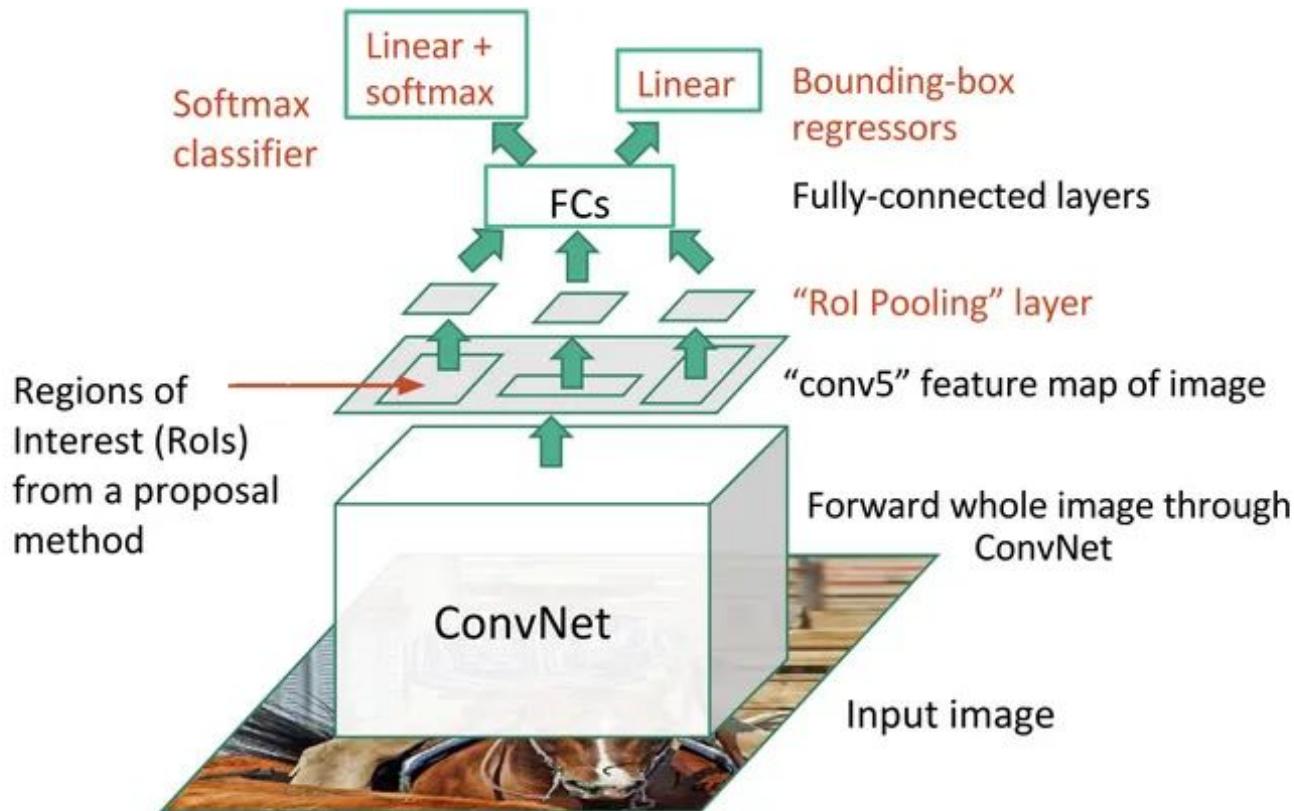
RoI Pooling



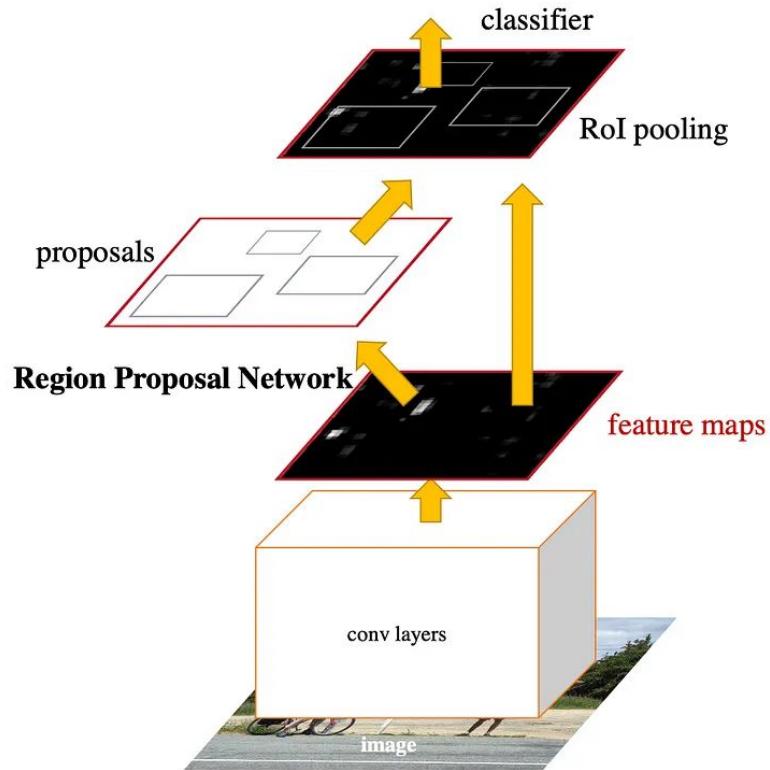
8 x 8									
0.88	0.44	0.16	0.14	0.37	0.77	0.96	0.27		
0.19	0.45	0.16	0.57	0.63	0.29	0.71	0.70		
0.66	0.36	0.64	0.82	0.54	0.73	0.59	0.25		
0.85	0.24	0.84	0.76	0.29	0.75	0.62	0.24		
0.32	0.74	0.39	0.31	0.34	0.03	0.33	0.48		
0.20	0.69	0.13	0.16	0.73	0.65	0.96	0.32		
0.19	0.14	0.86	0.09	0.88	0.07	0.01	0.48		
0.83	0.24	0.04	0.97	0.34	0.35	0.50	0.91		



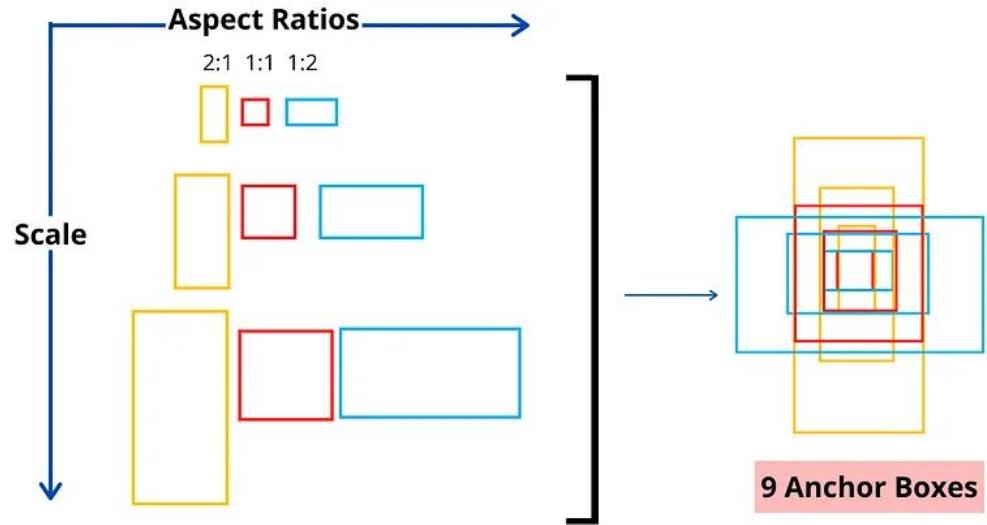
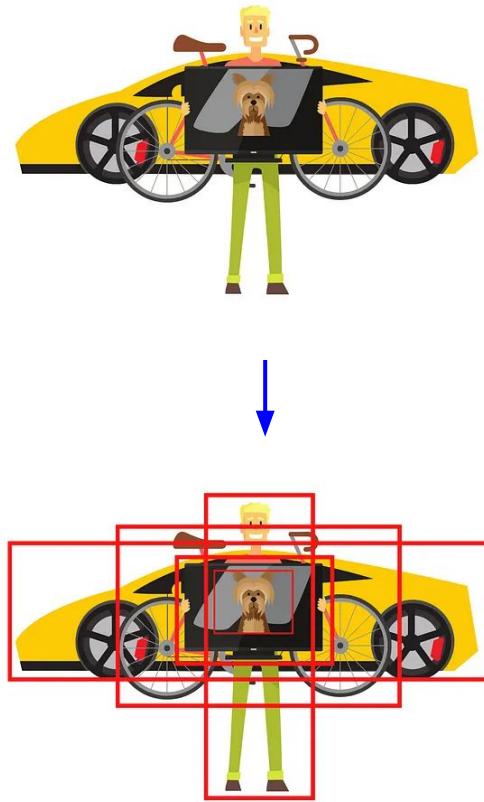
R-CNN \rightarrow Fast R-CNN \rightarrow Faster R-CNN



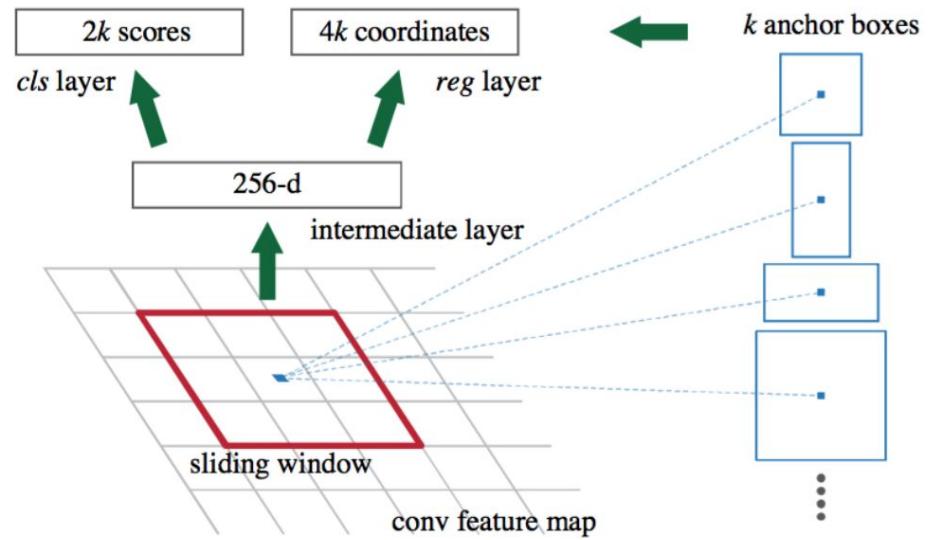
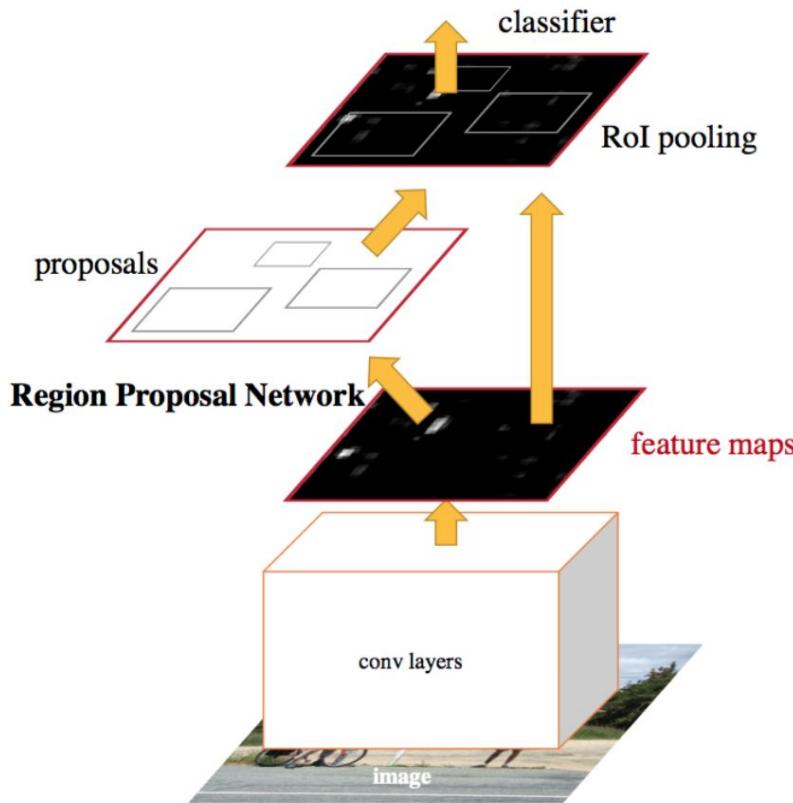
R-CNN \rightarrow Fast R-CNN \rightarrow Faster R-CNN



Anchors



Region Proposal Network (RPN)



Faster R-CNN losses

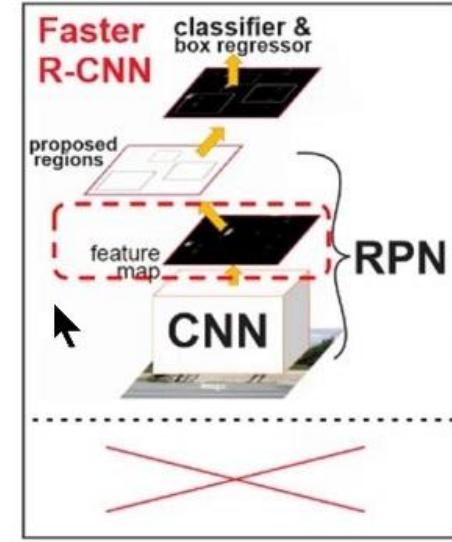
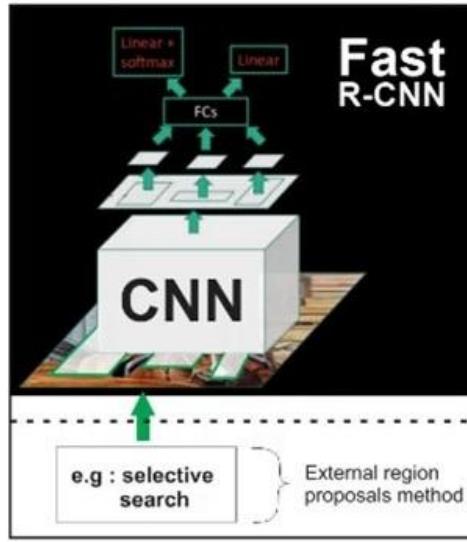
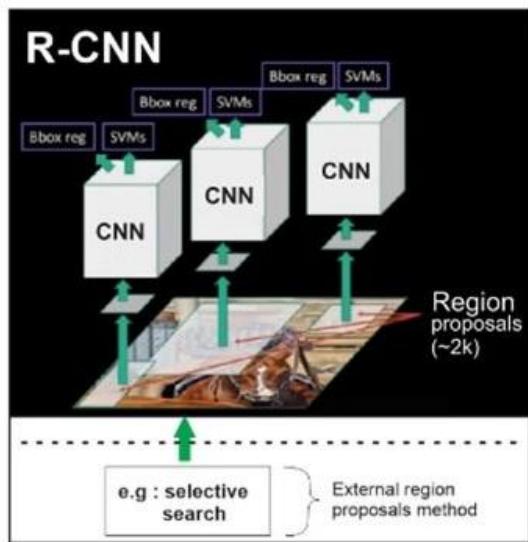
Faster R-CNN: Learnable region proposals

Jointly train with 4 losses:

1. **RPN classification:** anchor box is object / not an object
2. **RPN regression:** predict transform from anchor box to proposal box
3. **Object classification:** classify proposals as background / object class
4. **Object regression:** predict transform from proposal box to object box



Comparison



	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image	50 seconds	2 seconds	0.2 seconds
Speed-up	1x	25x	250x