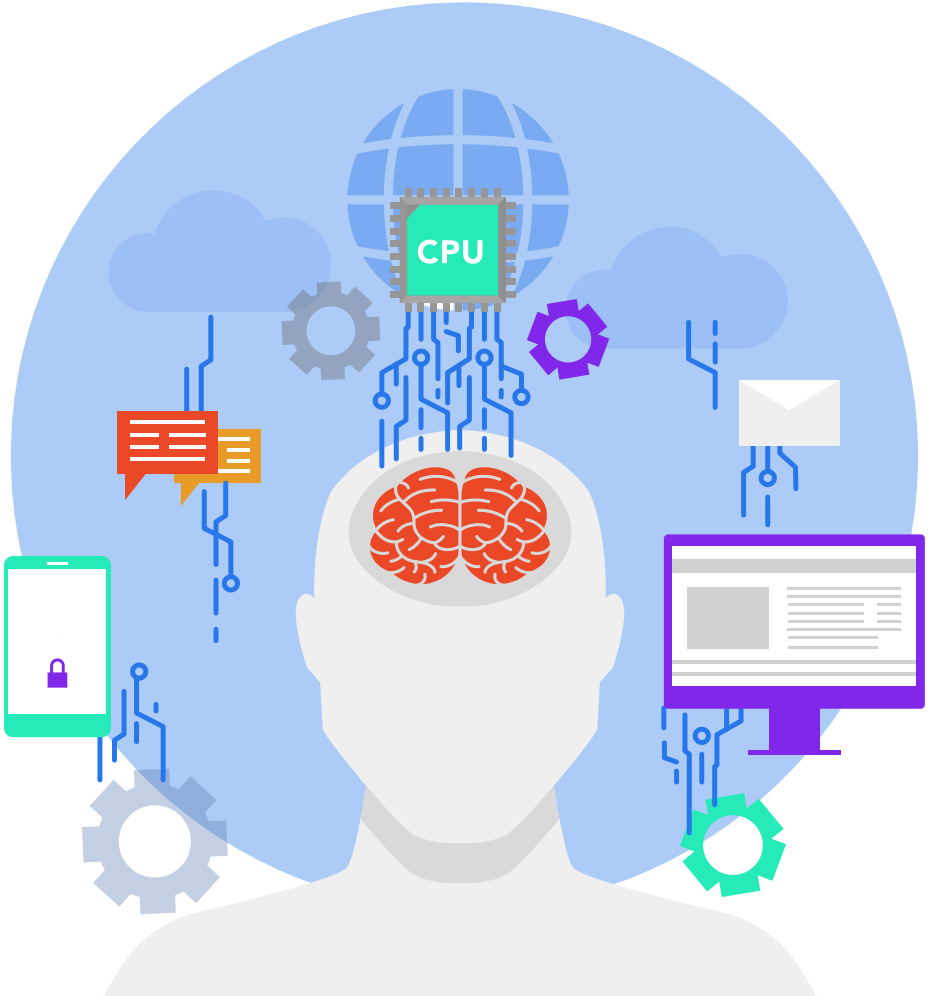
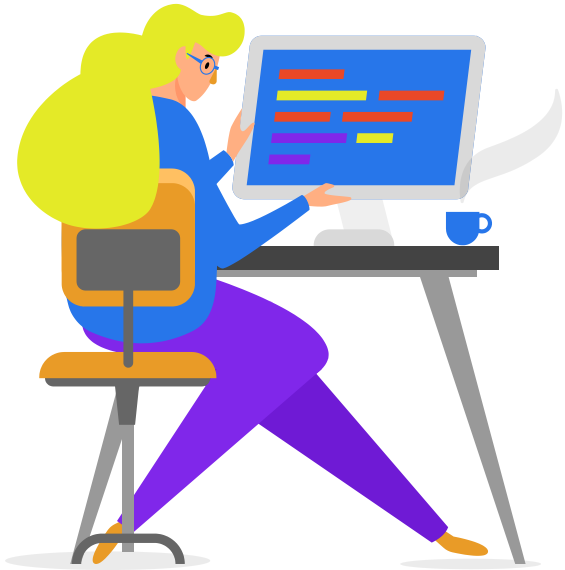


Deep Learning for Computer Vision

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

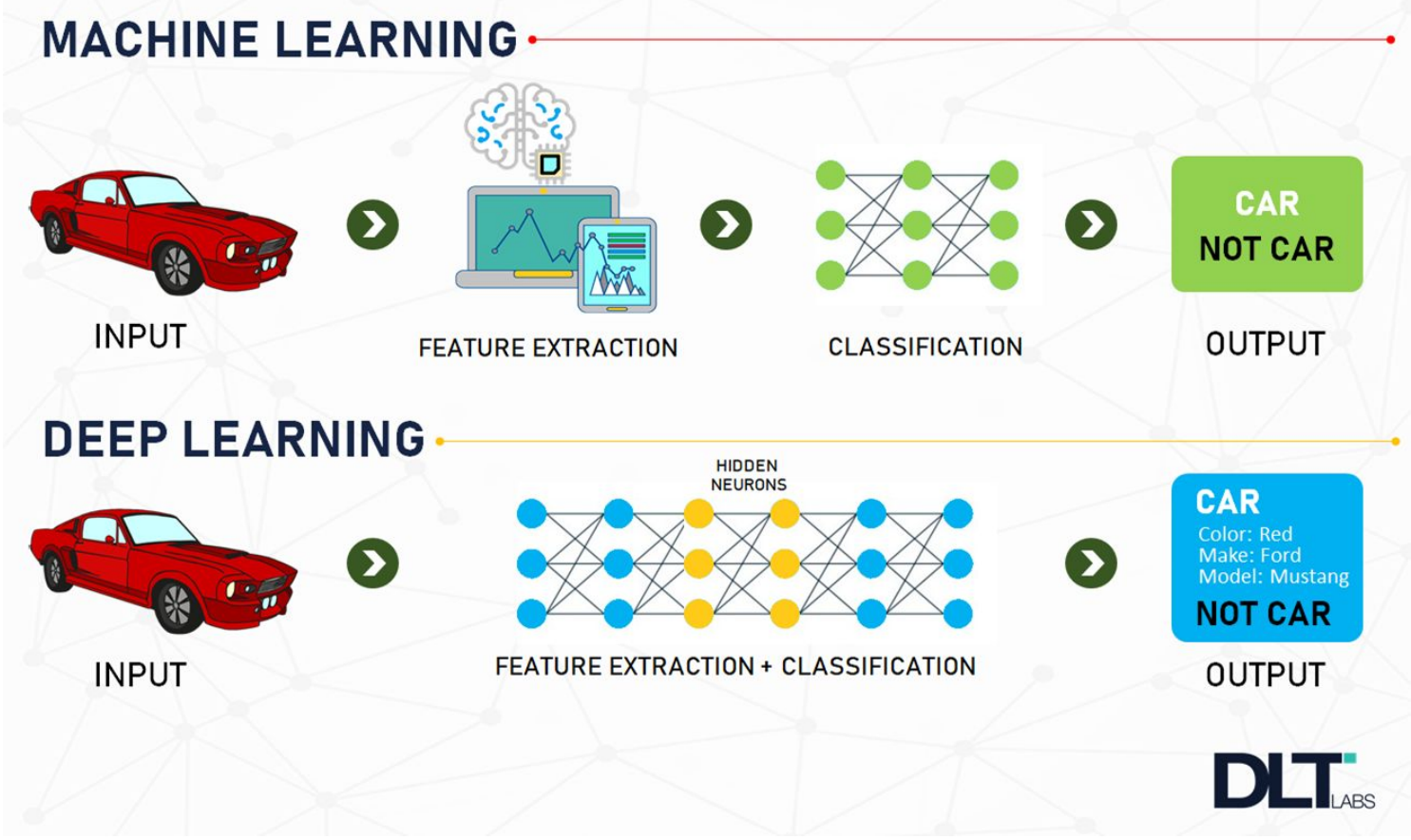
04 **Image Classification**
Bài toán phân loại ảnh

05 **Object detection**
Bài toán định vị đối tượng

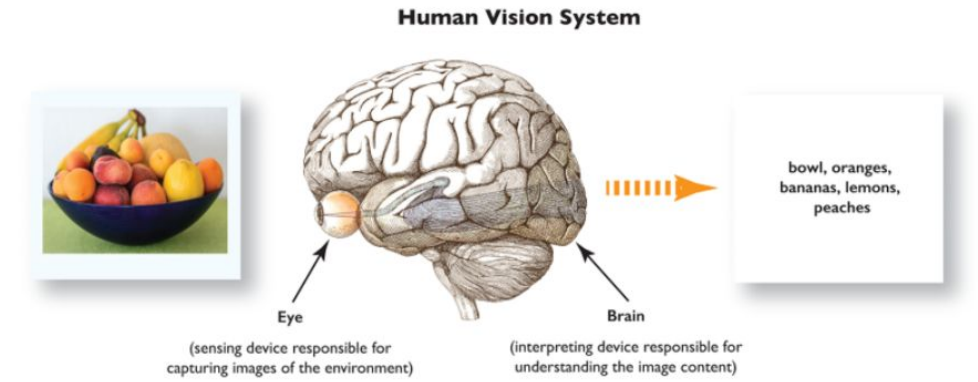
06 **Other topics**
Image segmentation, GANs

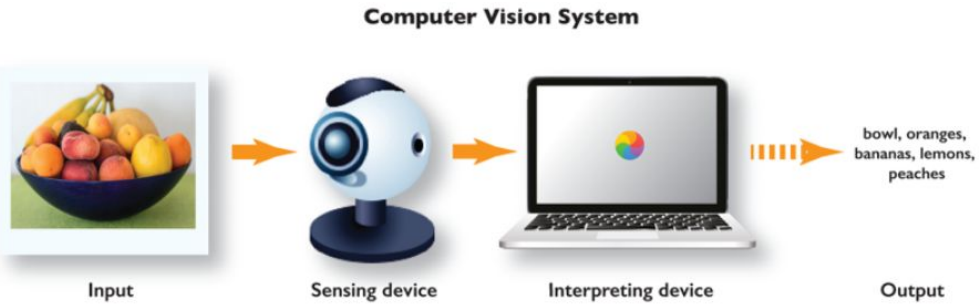
Tổng quan về Deep Learning

Tổng quan về Deep Learning

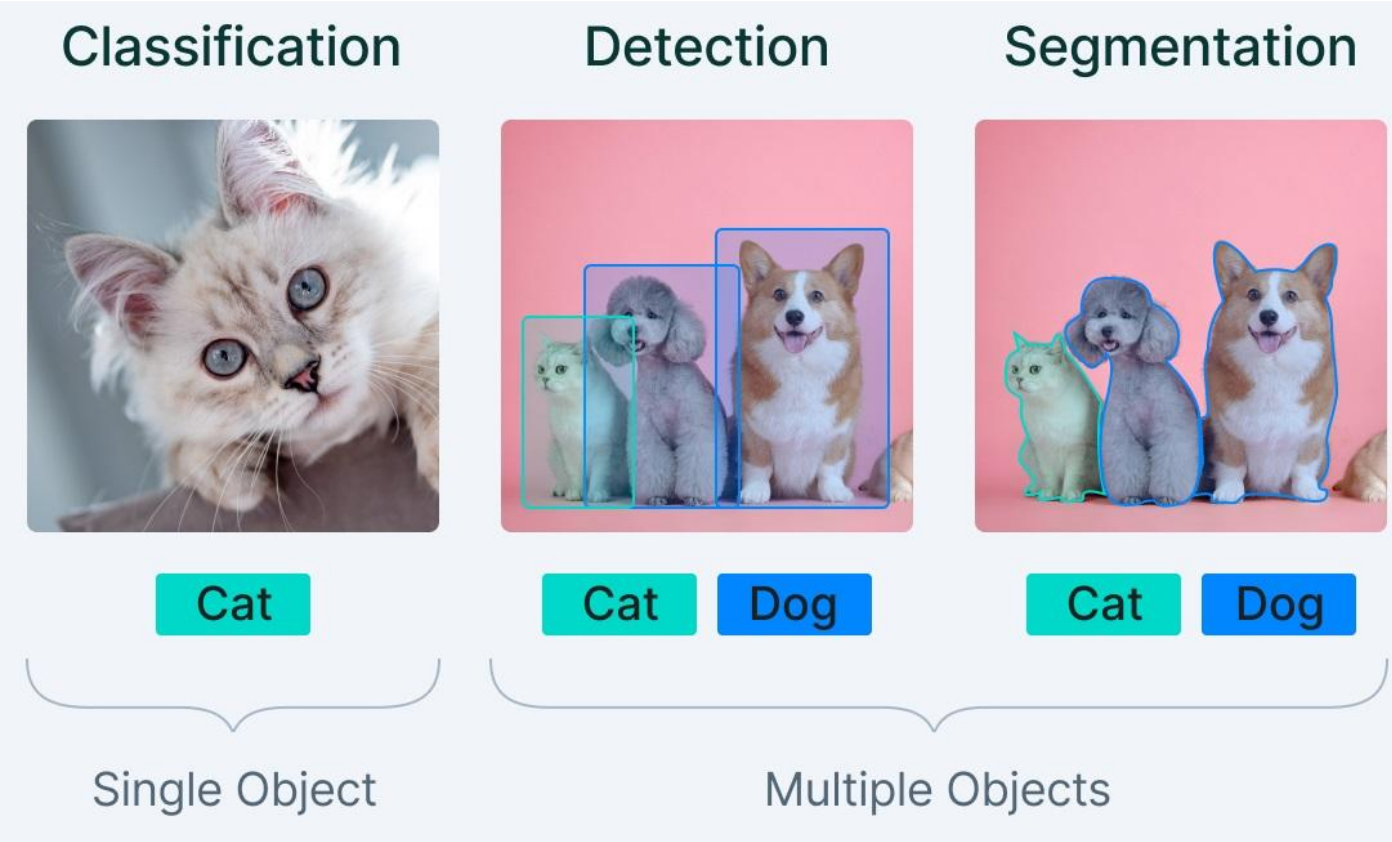


Deep Learning for Computer Vision

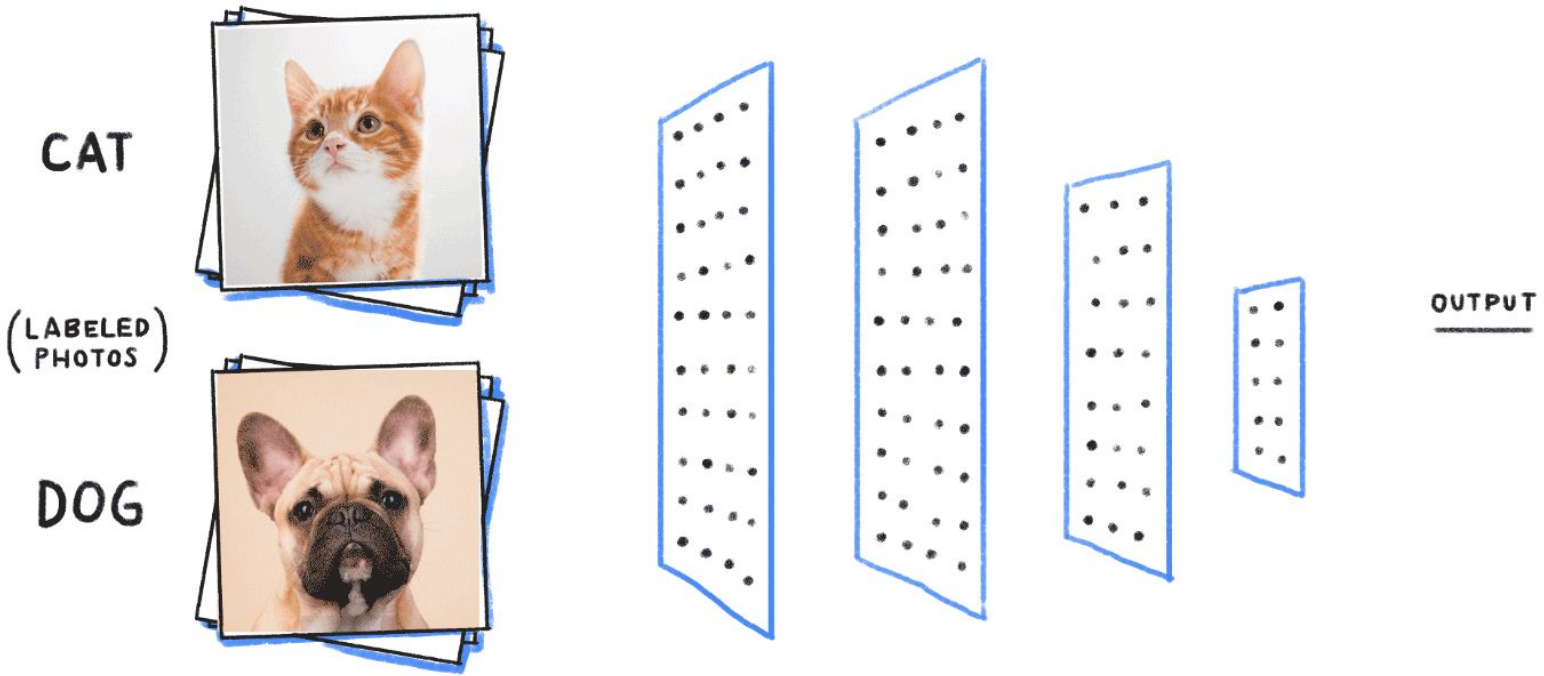




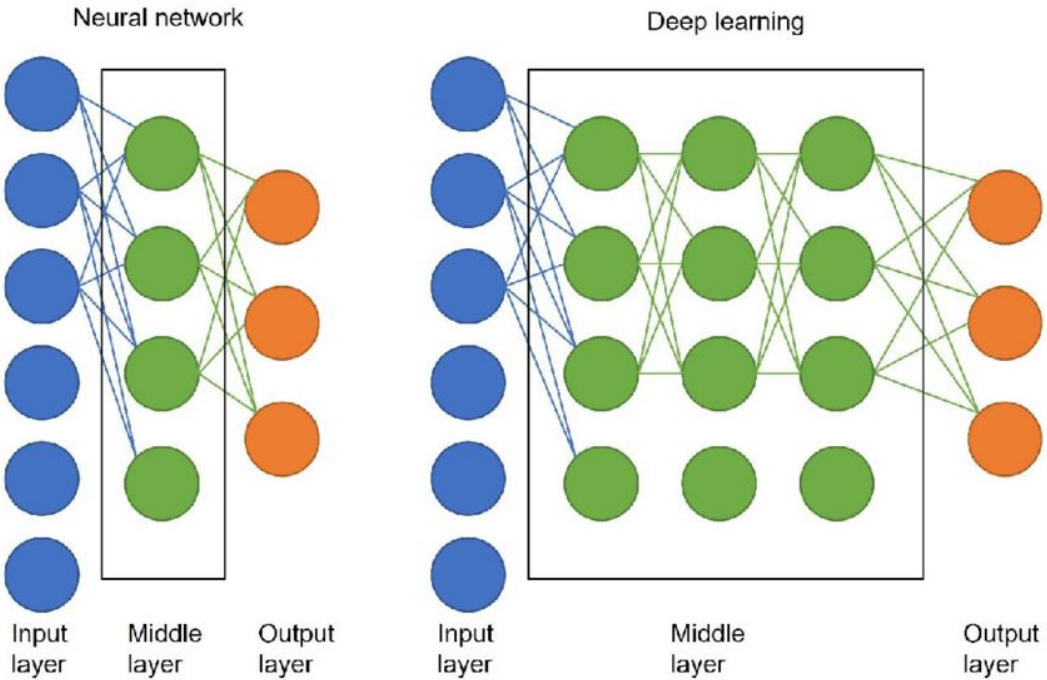
Deep Learning for Computer Vision



Deep Learning for Computer Vision



Neural Network vs Deep Learning



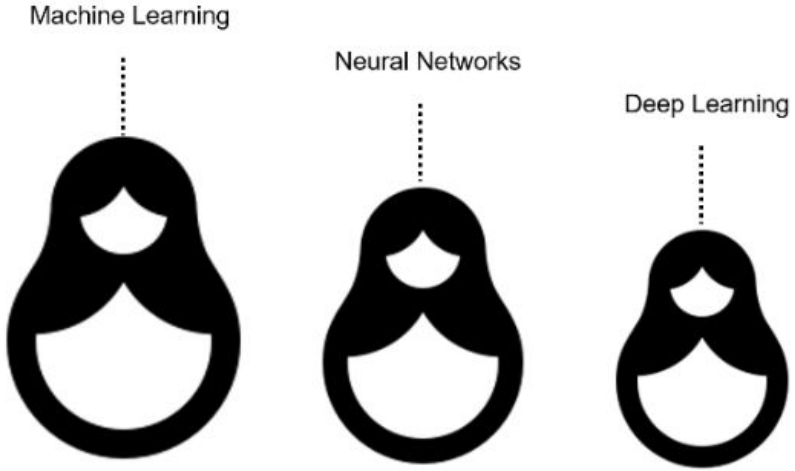
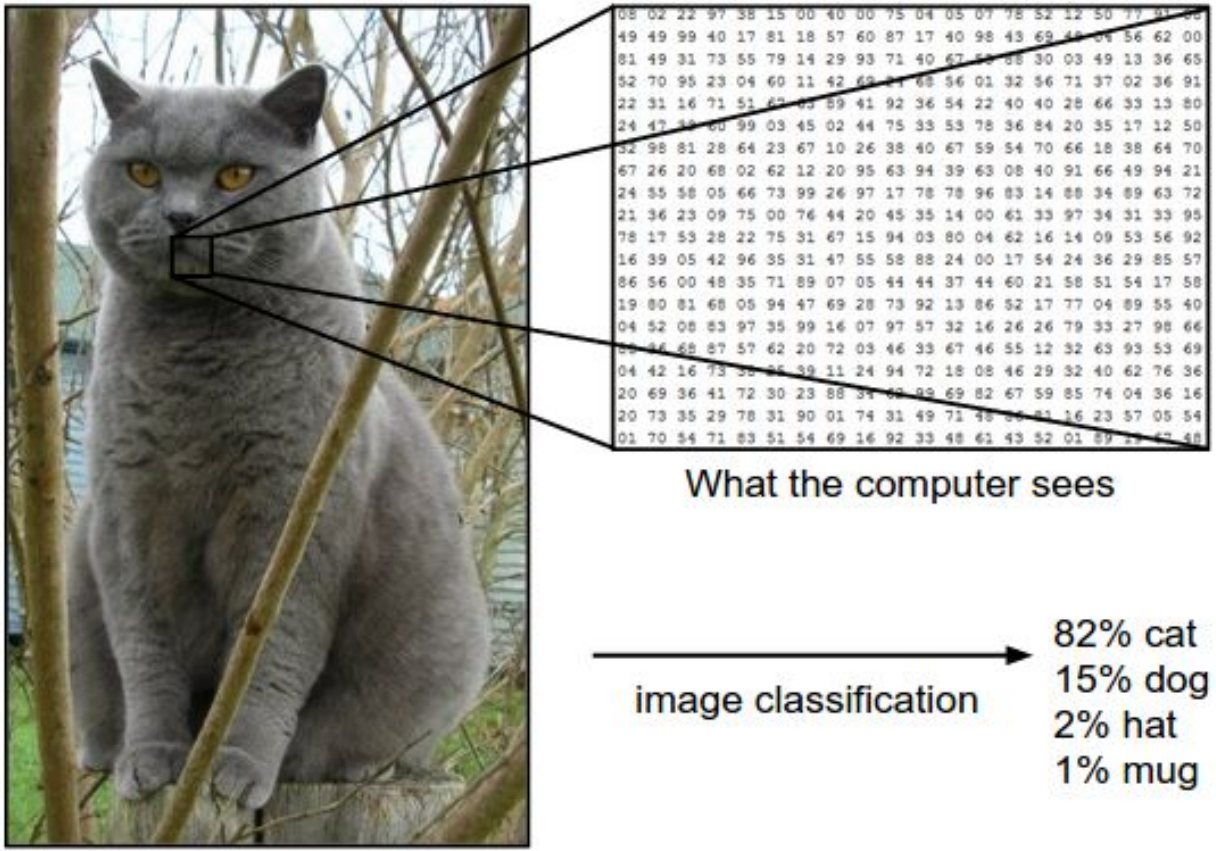
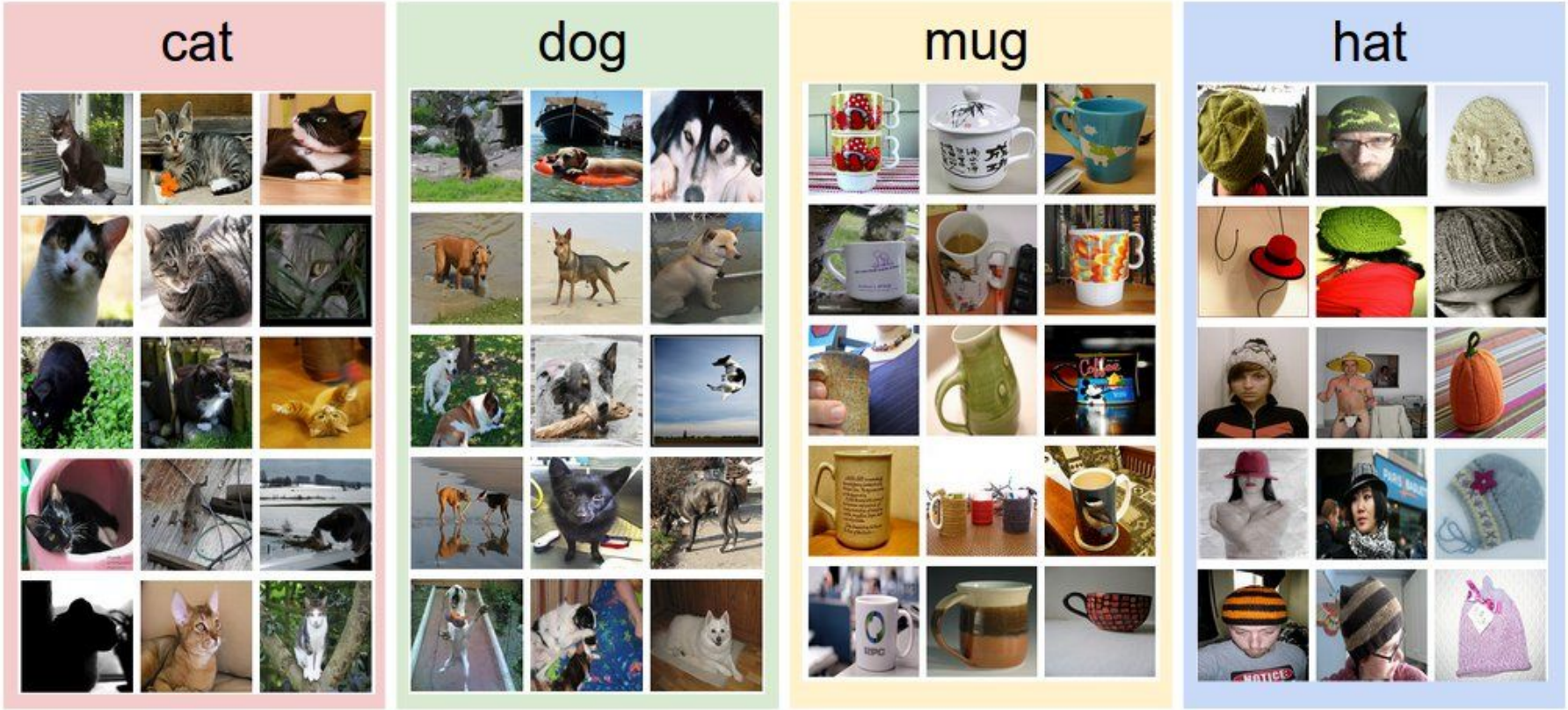


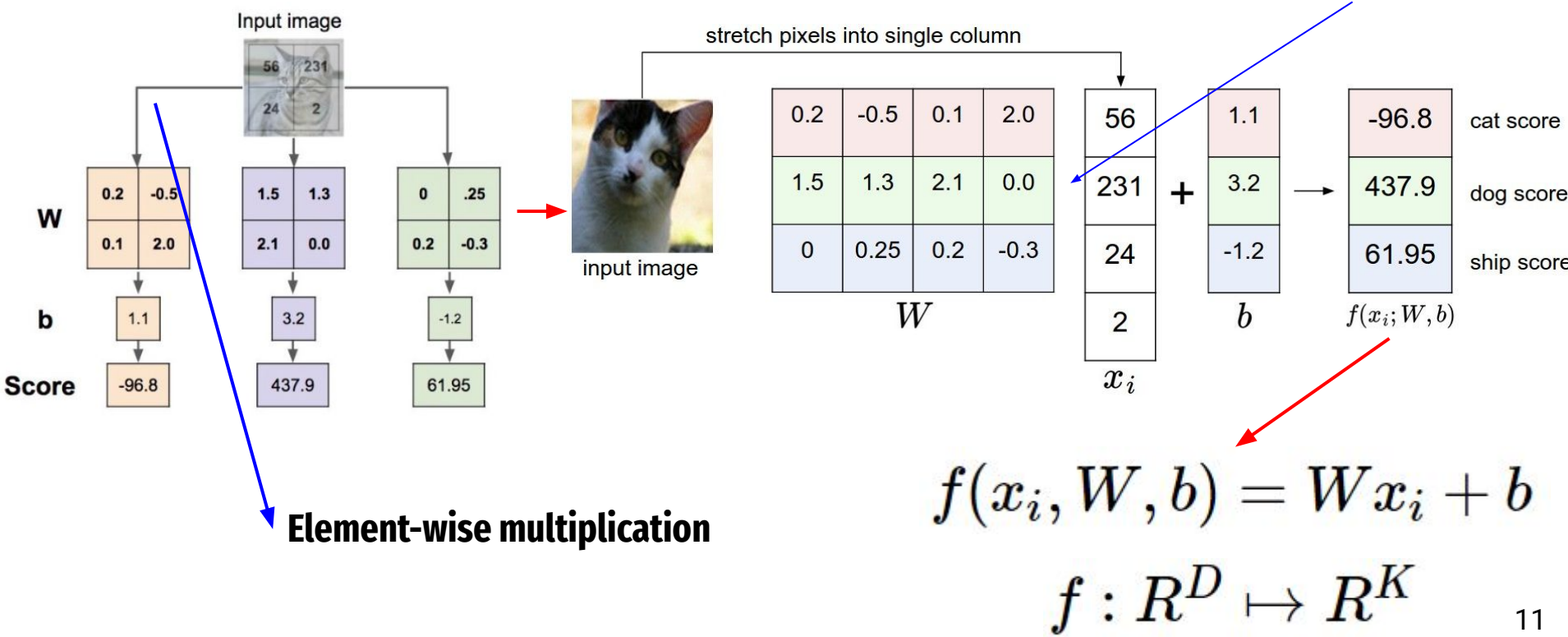
Image Classification



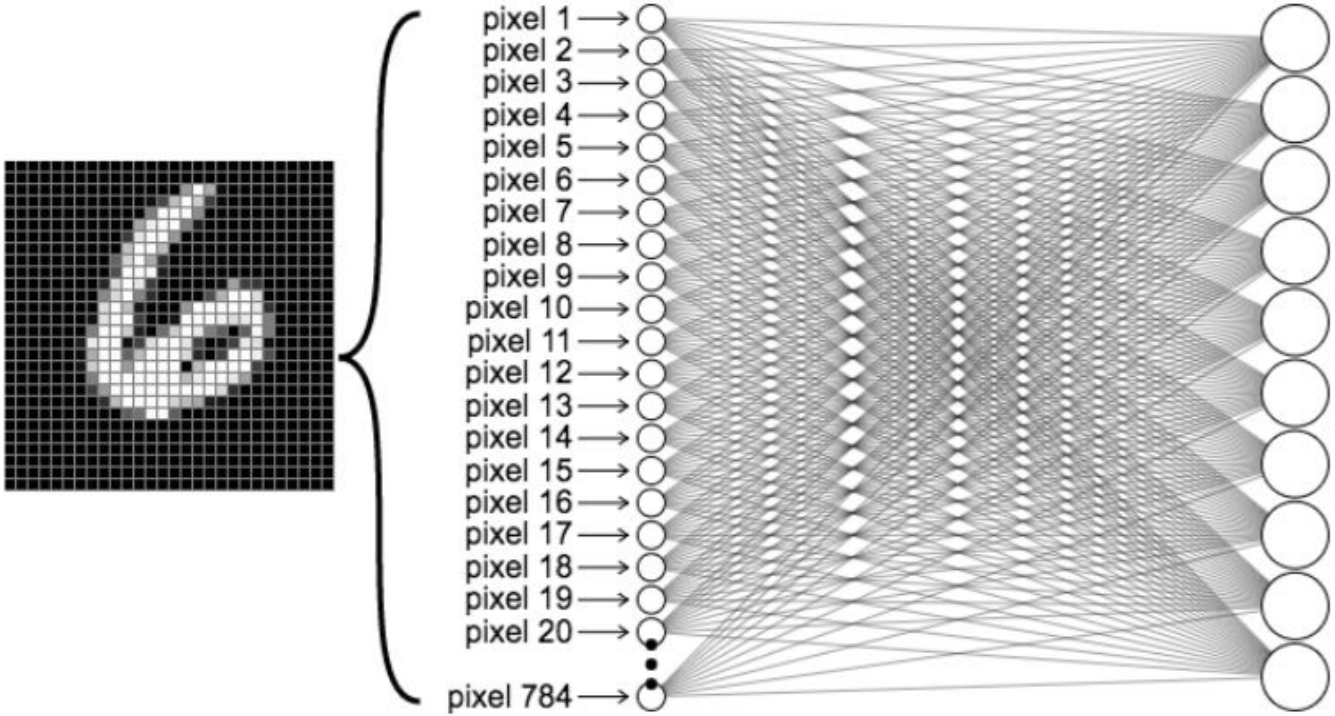
Dataset



Linear Classifier - score function

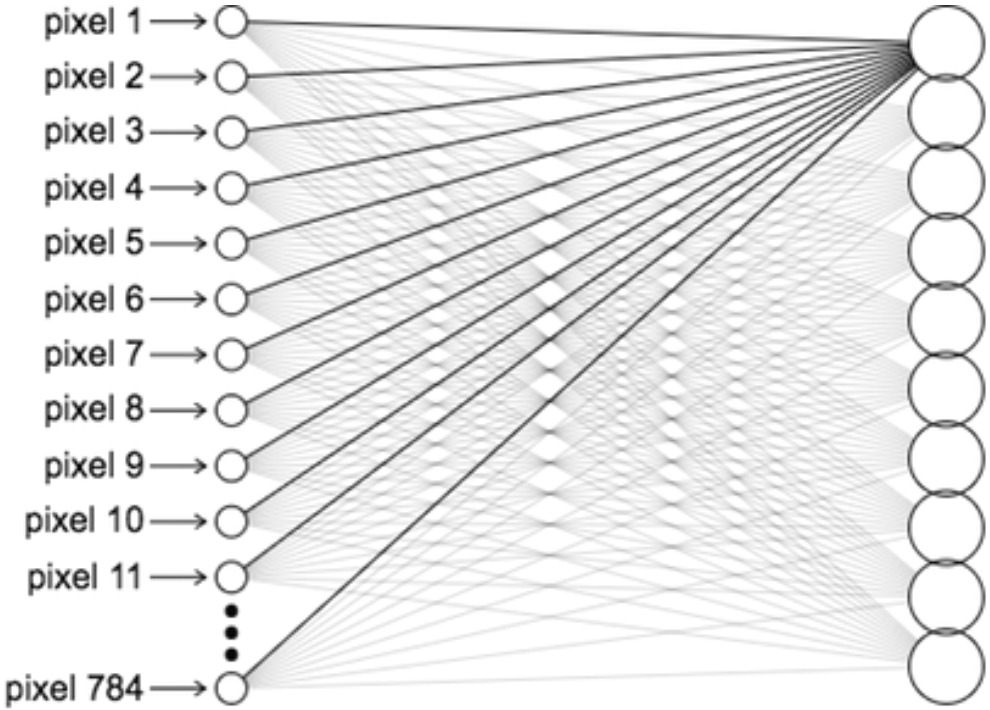


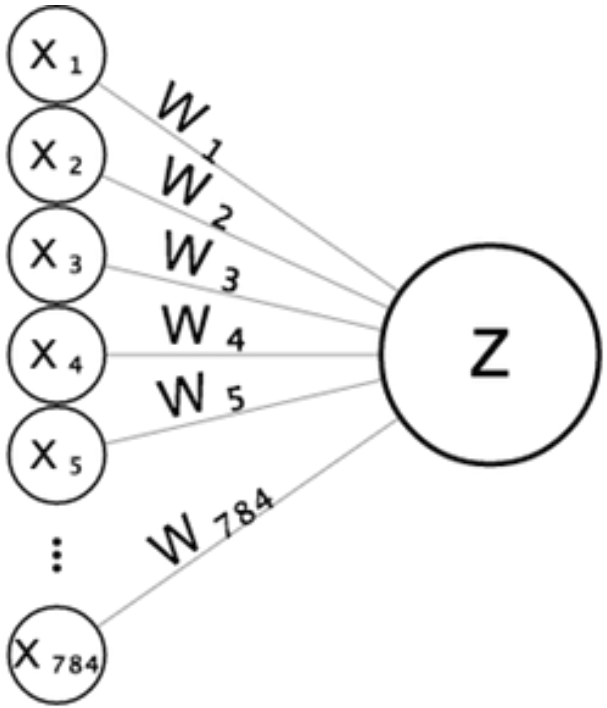
Linear Classifier explanation



MNIST dataset

Linear Classifier - Weight visualization

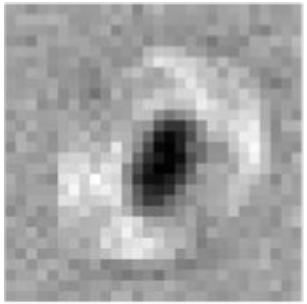




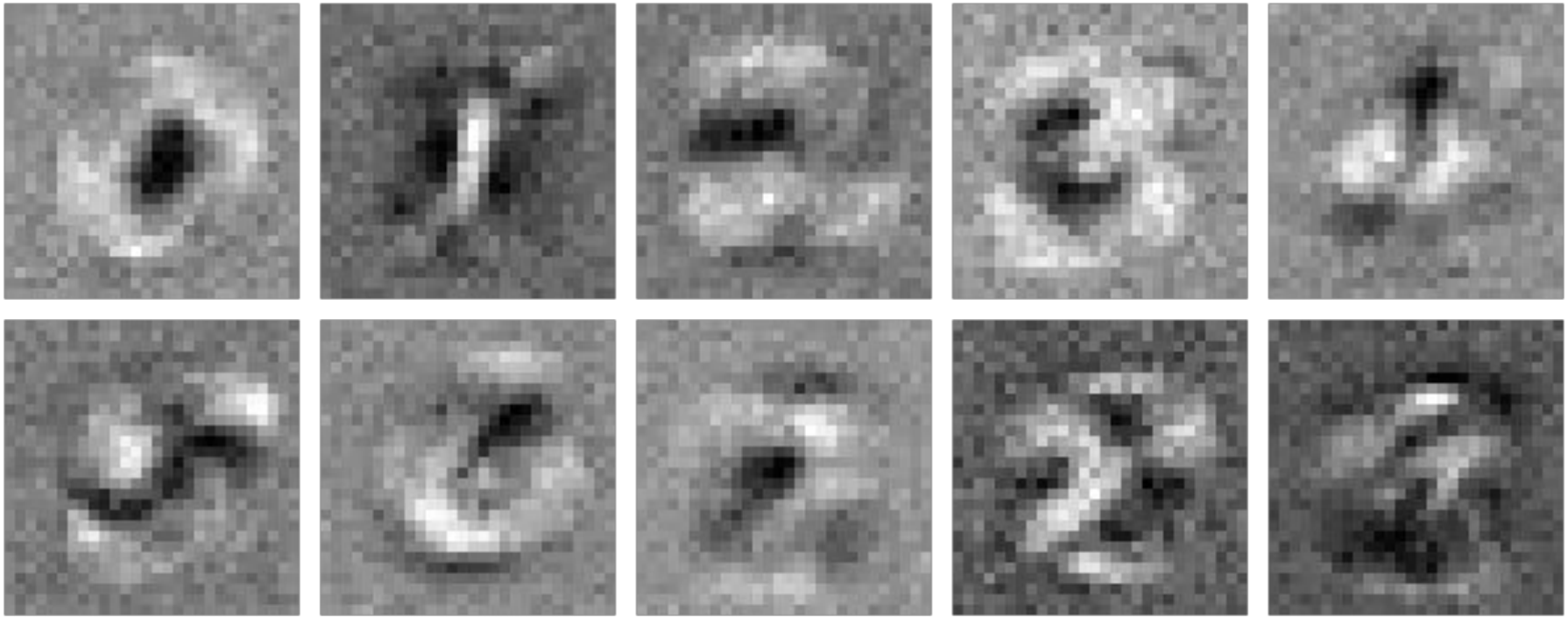
Linear Classifier - Weight visualization

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

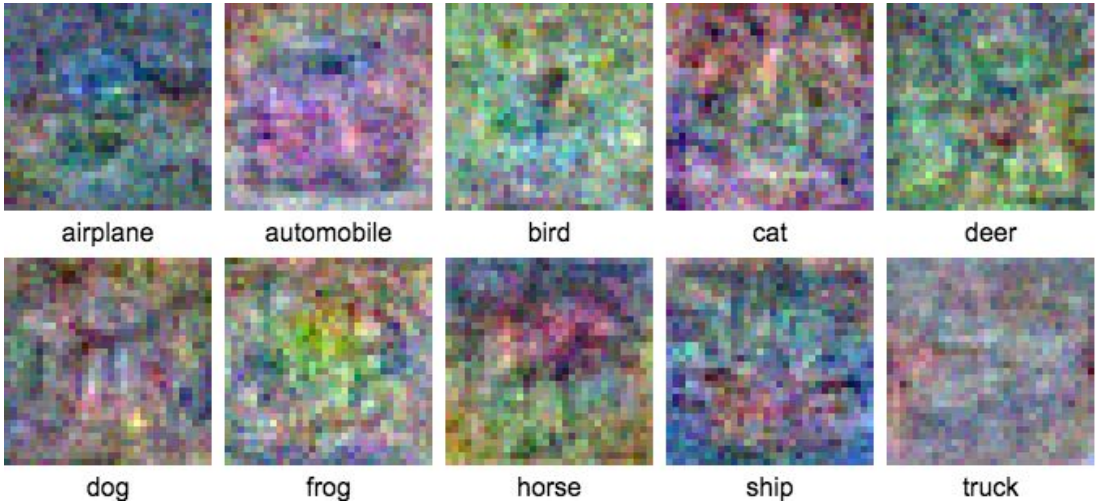
$$\begin{bmatrix} W_1 & W_2 & W_3 & \cdots & W_{28} \\ W_{29} & W_{30} & W_{31} & & W_{56} \\ W_{57} & W_{58} & W_{59} & & W_{84} \\ \vdots & & & \ddots & \\ W_{757} & W_{758} & W_{759} & & W_{784} \end{bmatrix}$$



Linear Classifier - Weight visualization

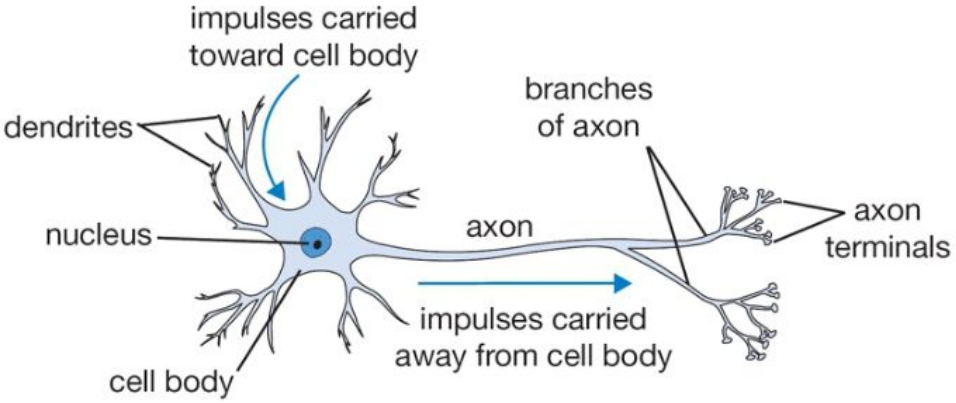


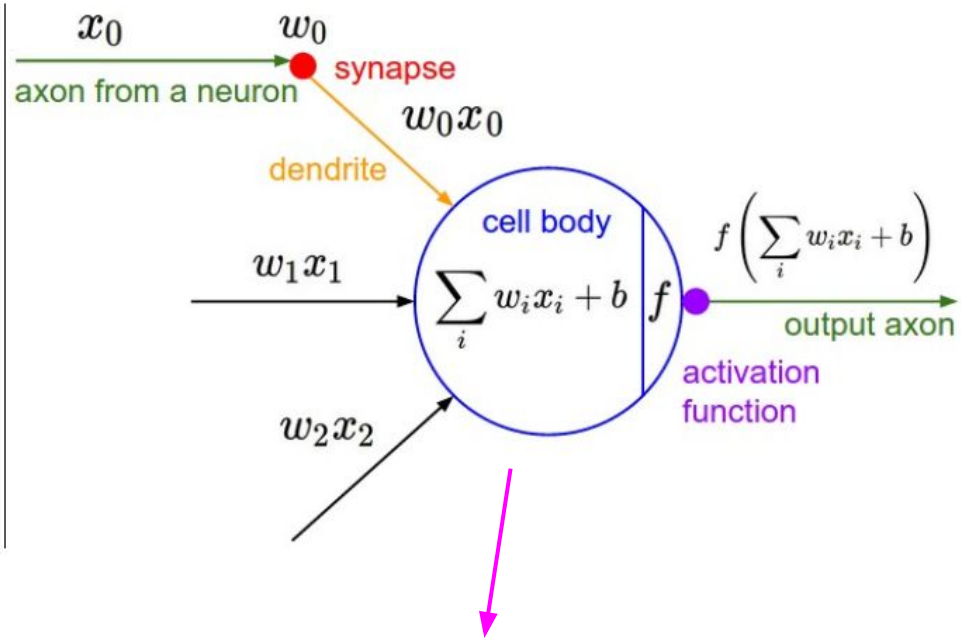
Linear Classifier - Weight visualization



[CIFAR dataset]

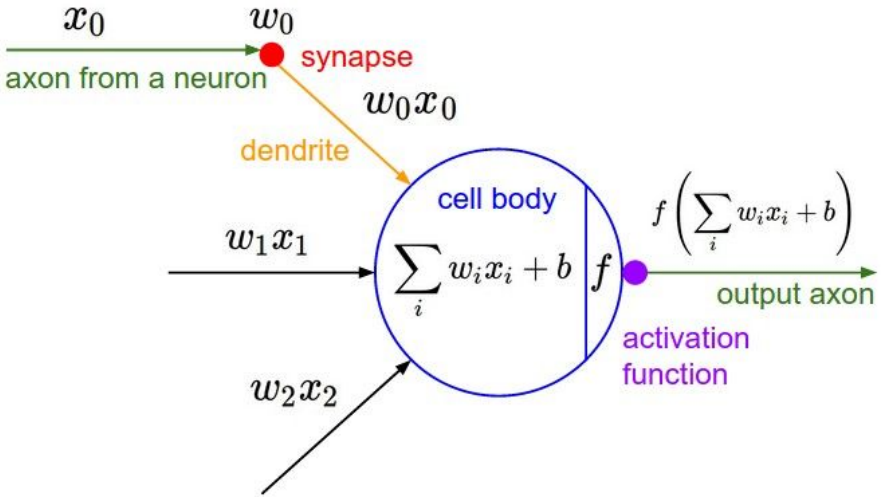
Biological Neuron vs Artificial Neuron

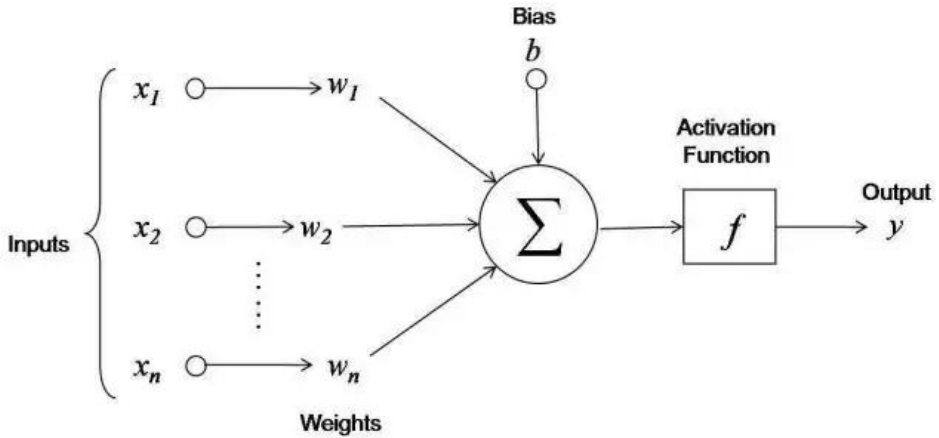




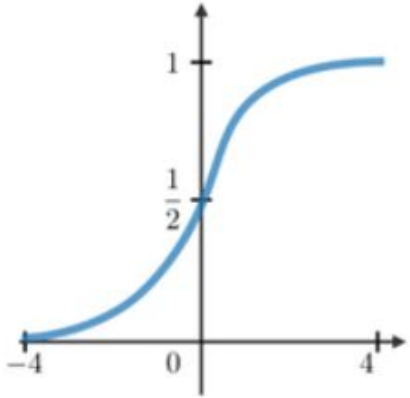
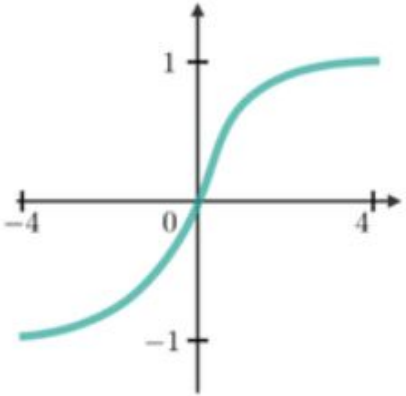
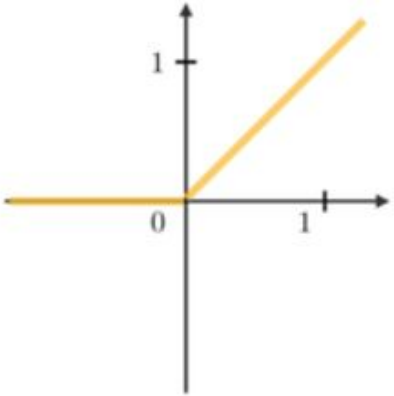
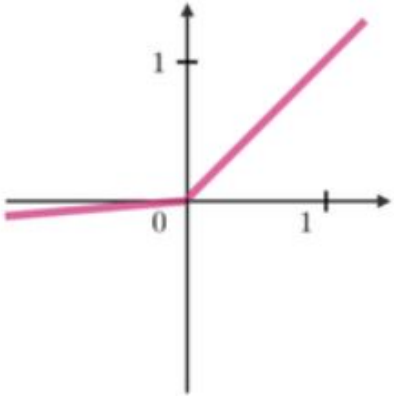
Neuron = linear classifier + activation function

From linear classifier to neuron

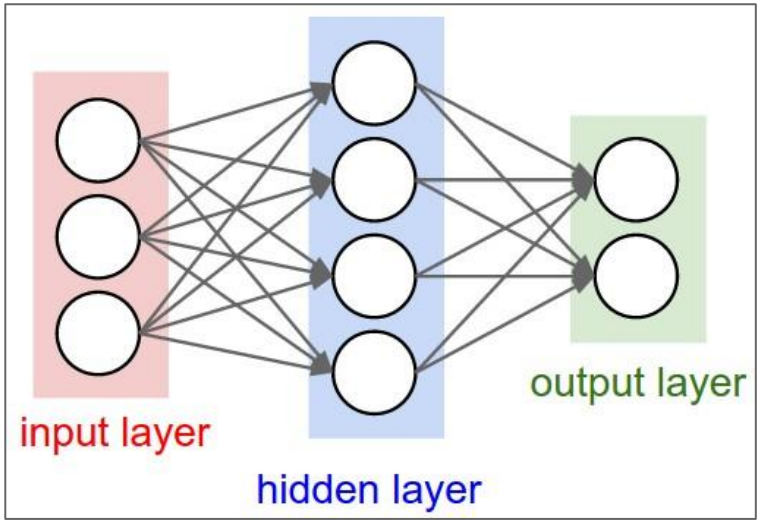


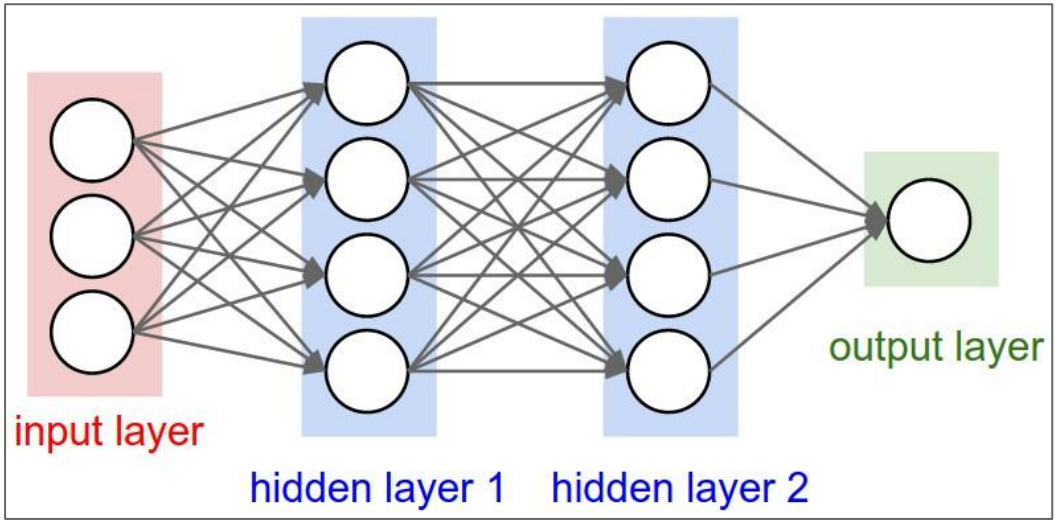


Activation function

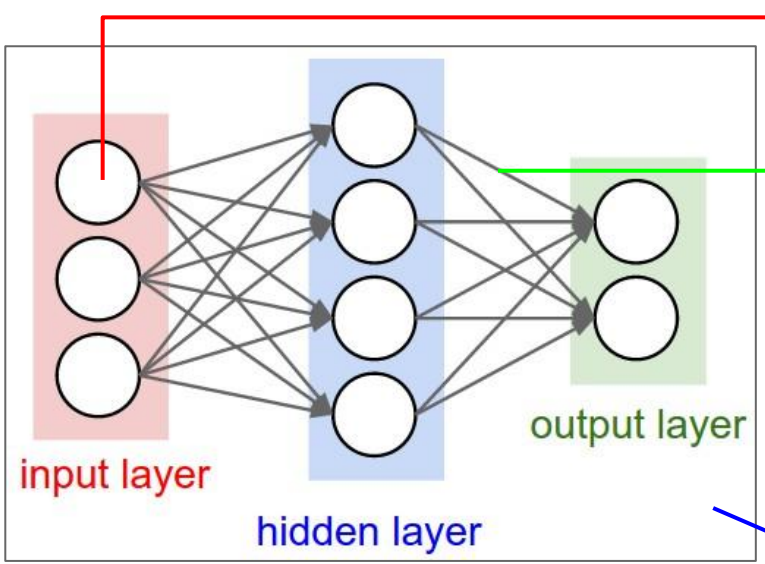
Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

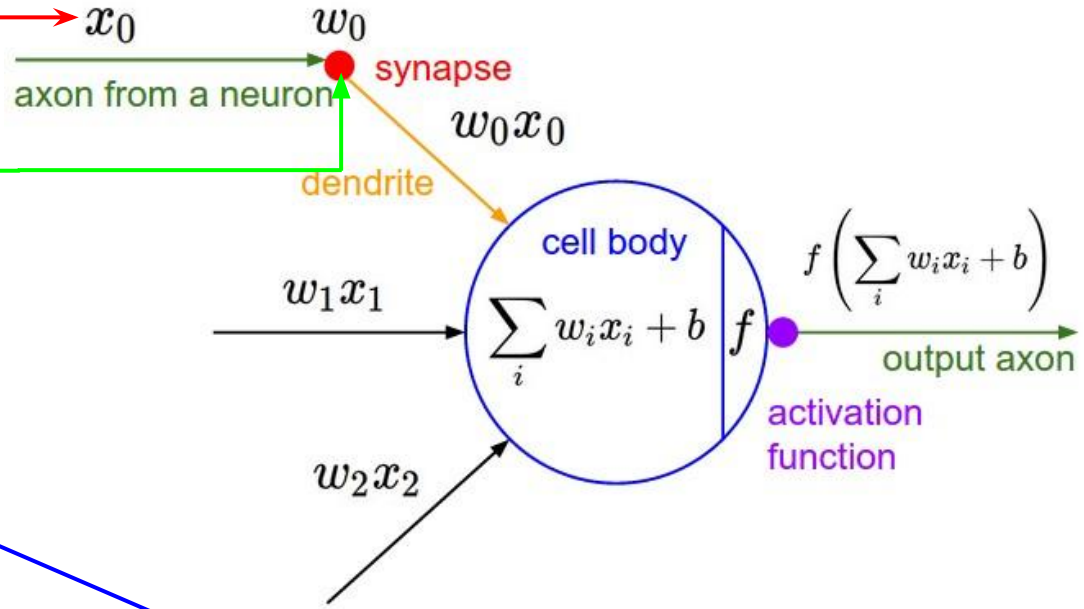
Neural Network Architecture





Neural Network





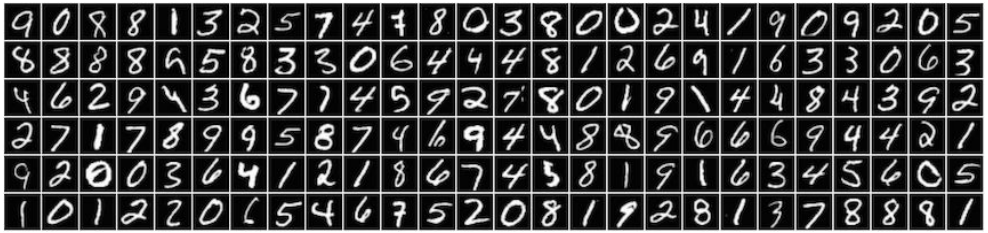
Forward propagation

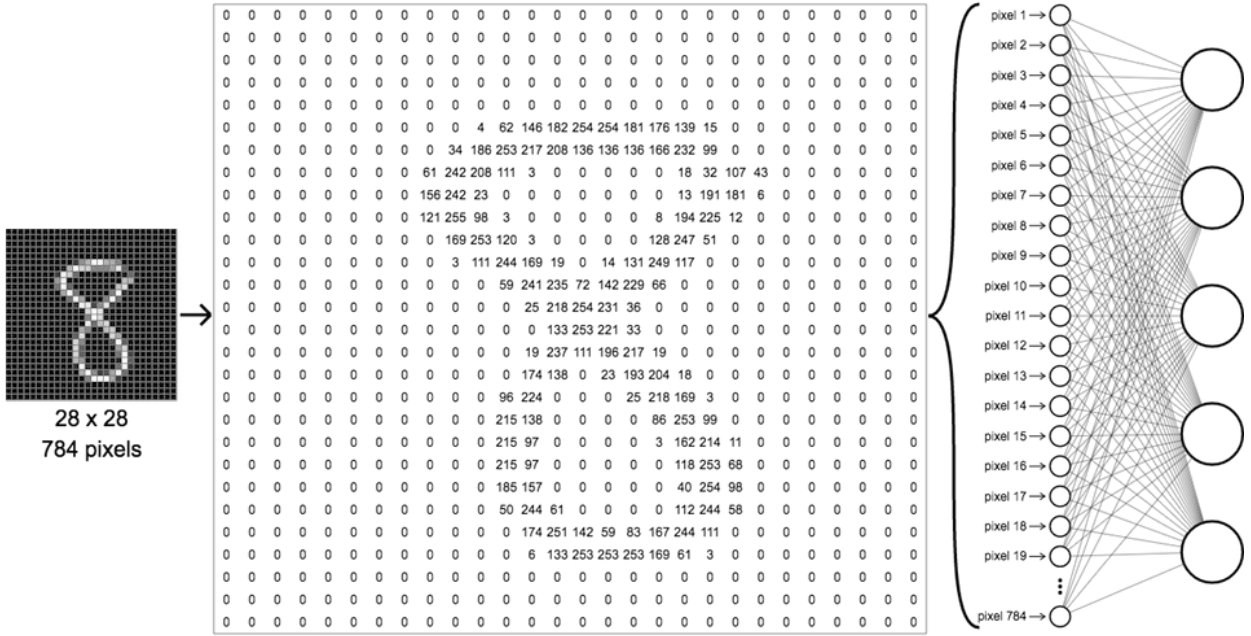
Forward pass

Feedforward neural network

Demo

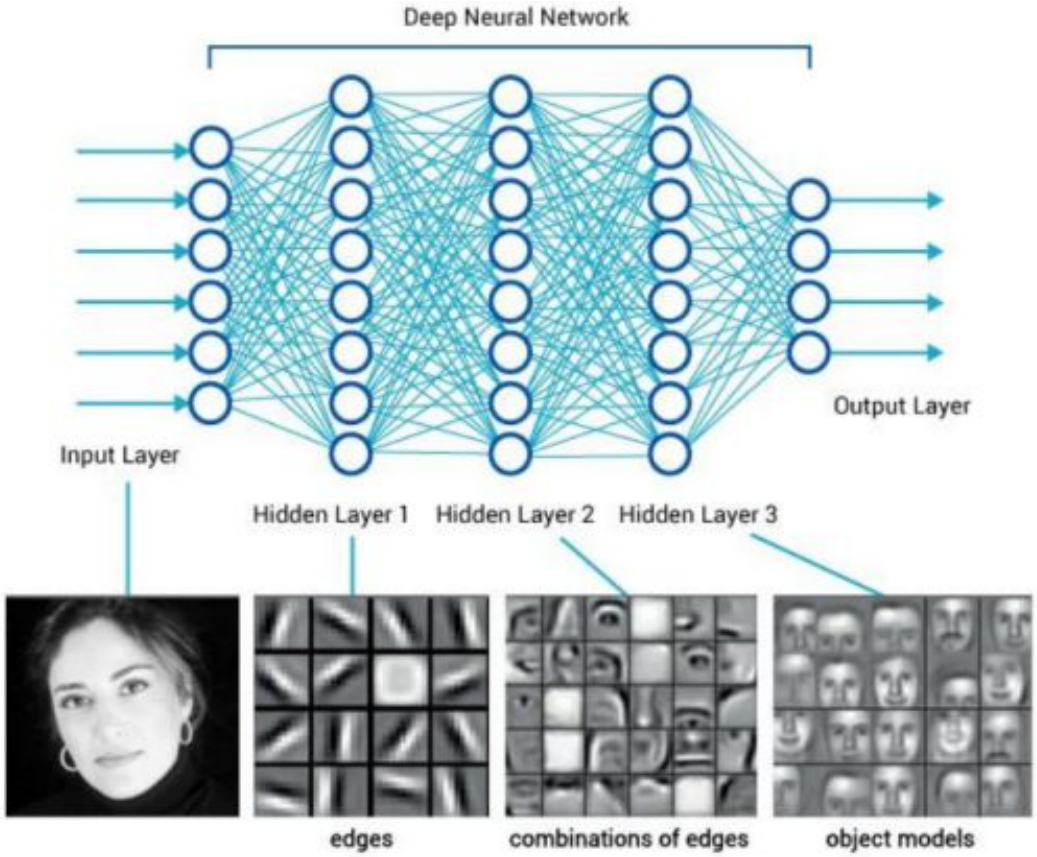
Complete example of Neural Network





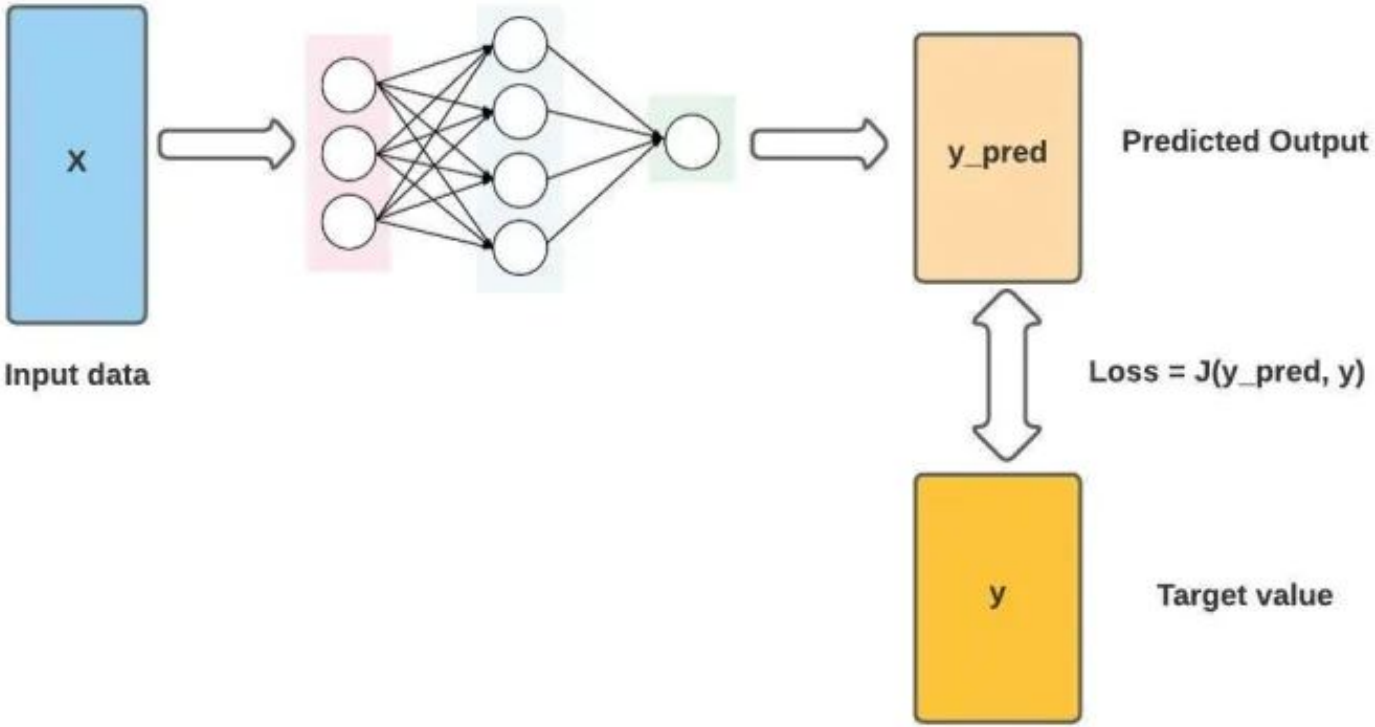
[Demo]

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

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

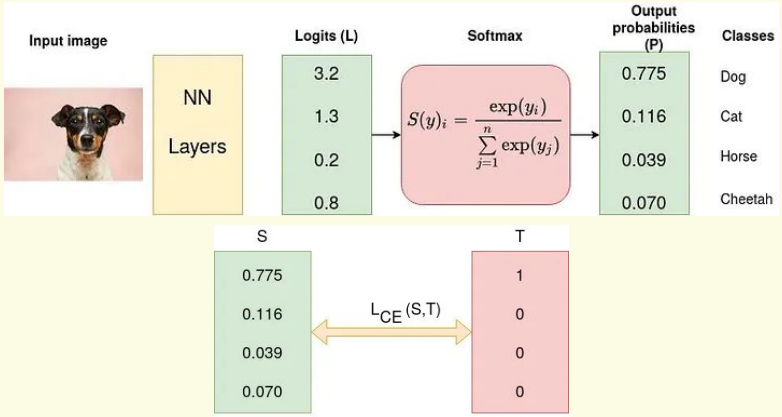
Huber Loss

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

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

Classification

Cross-Entropy Loss



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

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

```python
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

```python
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

```python
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

$$f(x,y)$$

$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{bmatrix}^T = \textit{gradient}$$

$$= \frac{\partial f}{\partial x} i + \frac{\partial f}{\partial y} j \quad \textbf{(in two dimensions)}$$

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

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

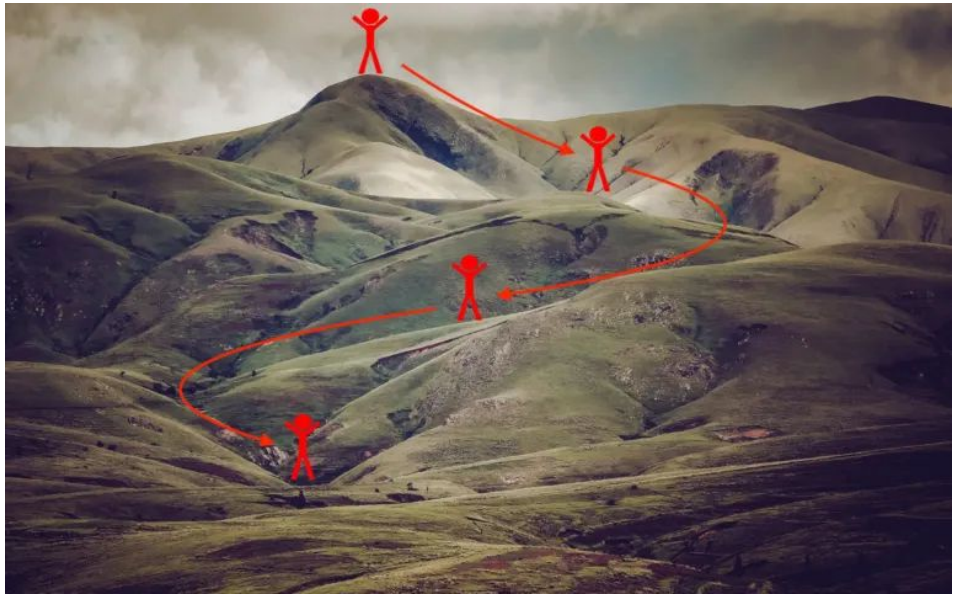
and

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

- 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.

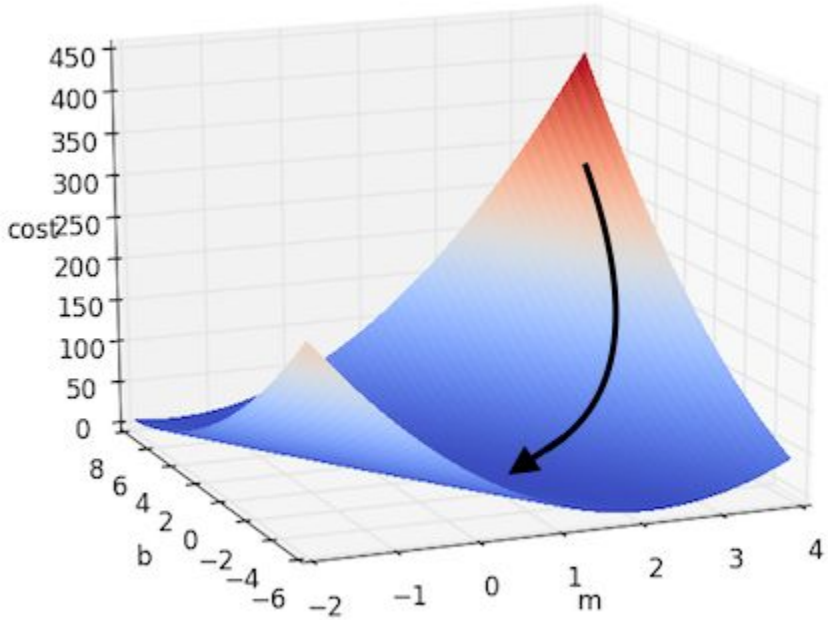
Optimization function

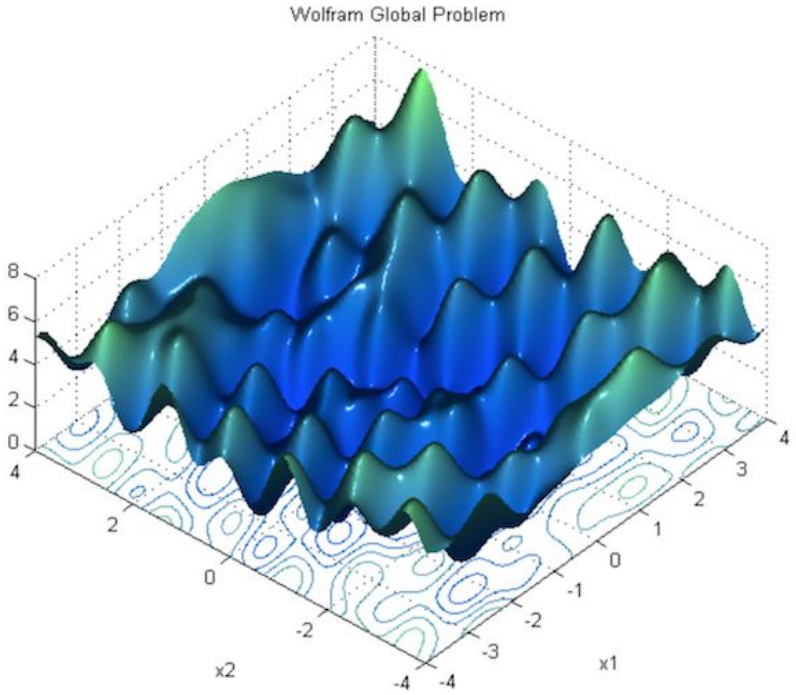
Gradient Descent



Optimization function

Gradient Descent in Linear Classifier vs Neural network





Gradient Descent: Different versions

Batch Gradient Descent

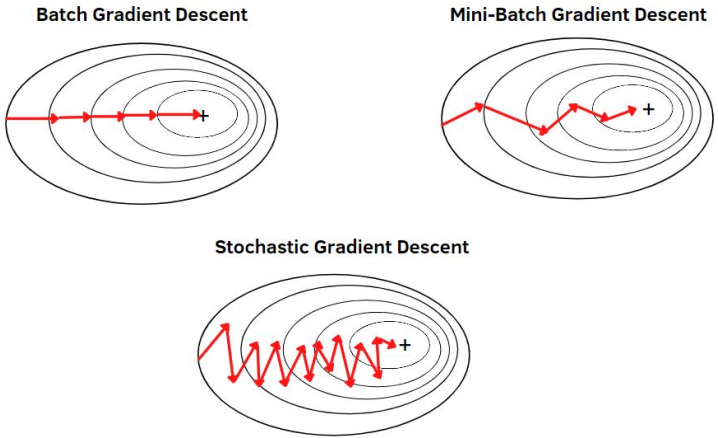
Tất cả datapoints được đưa vào mô hình cùng 1 lúc để tính gradient

Stochastic Gradient Descent

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

(Stochastic) Mini-batch Gradient Descent

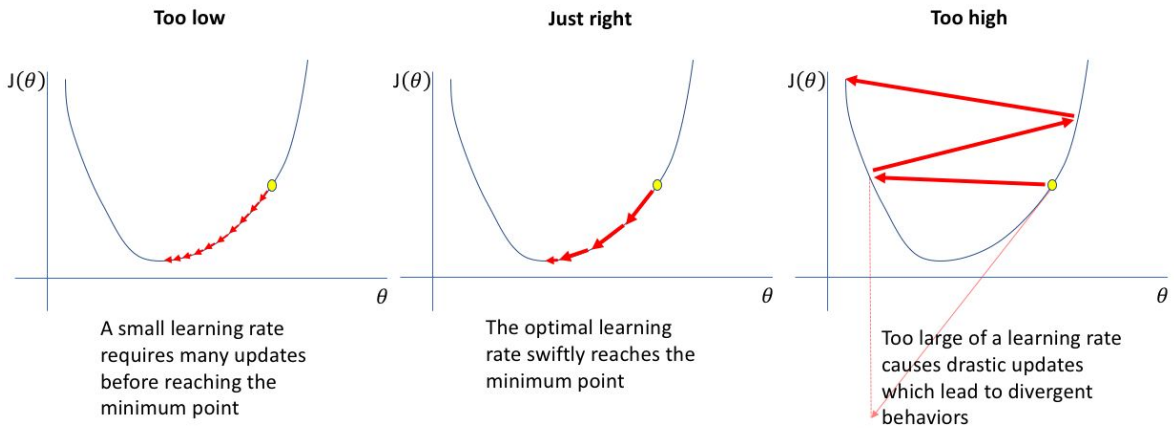
N datapoint sẽ được đưa vào mô hình cùng lúc để tính gradient



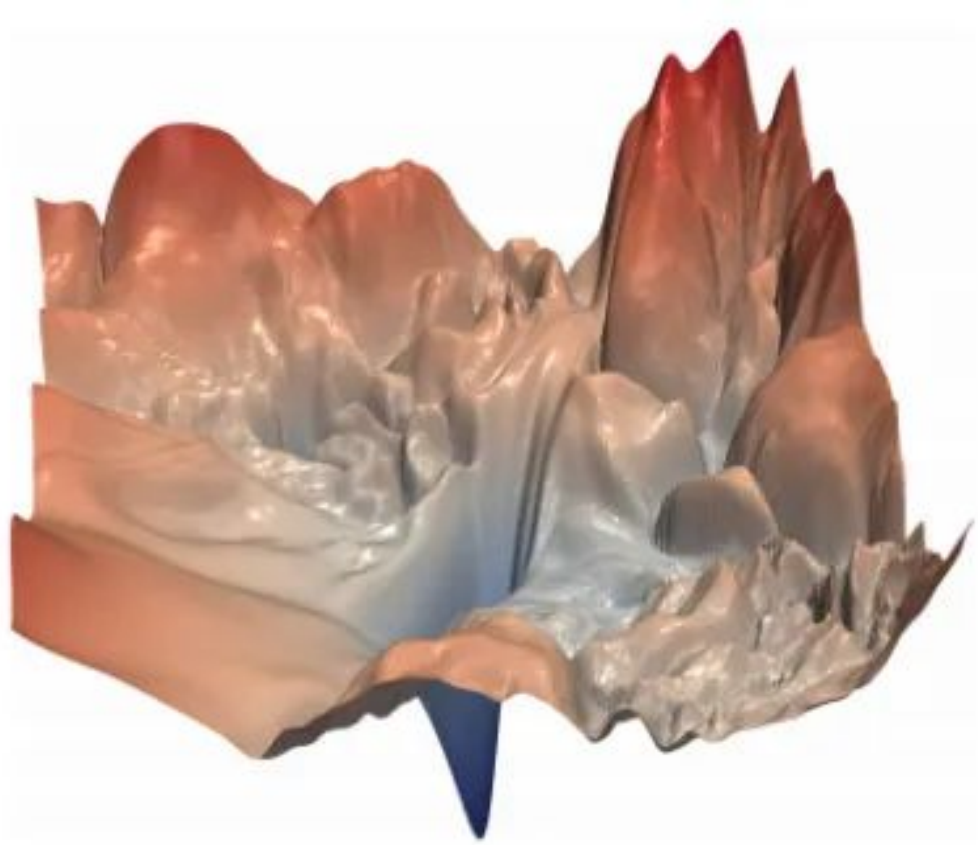
How parameters are updated?

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

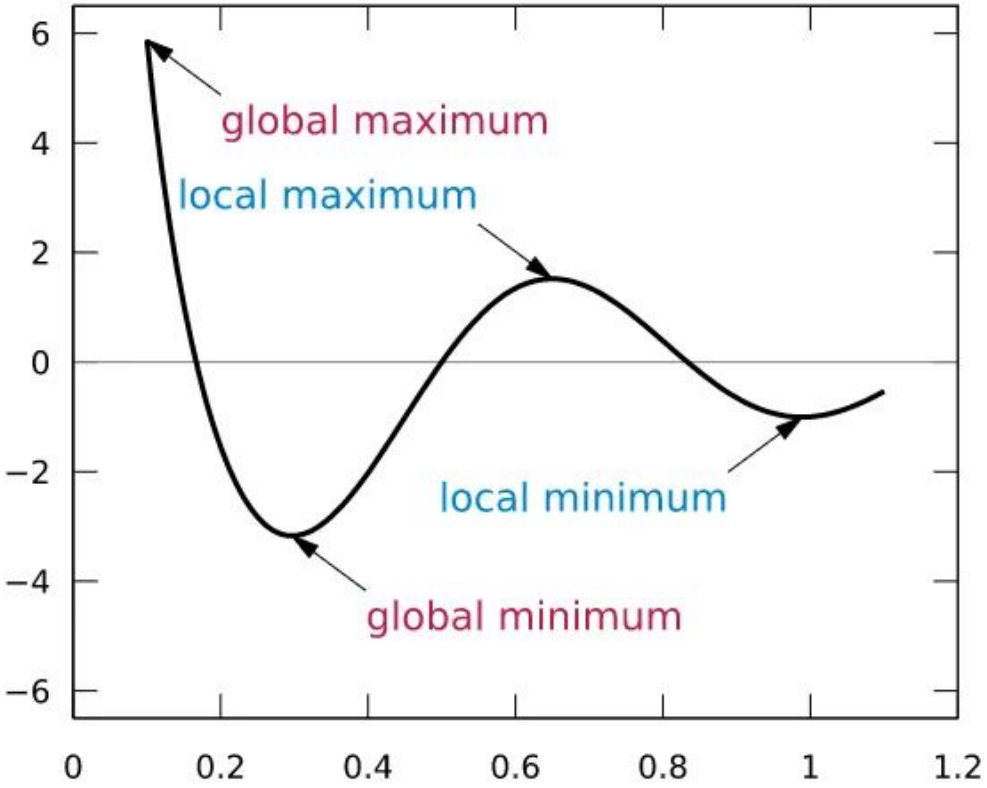
W_{new}	Updated weight
W_{old}	Old weight
η	Learning rate (η>0)
$\frac{\partial L}{\partial W_{old}}$	Partial derivative of L with respect to old weight



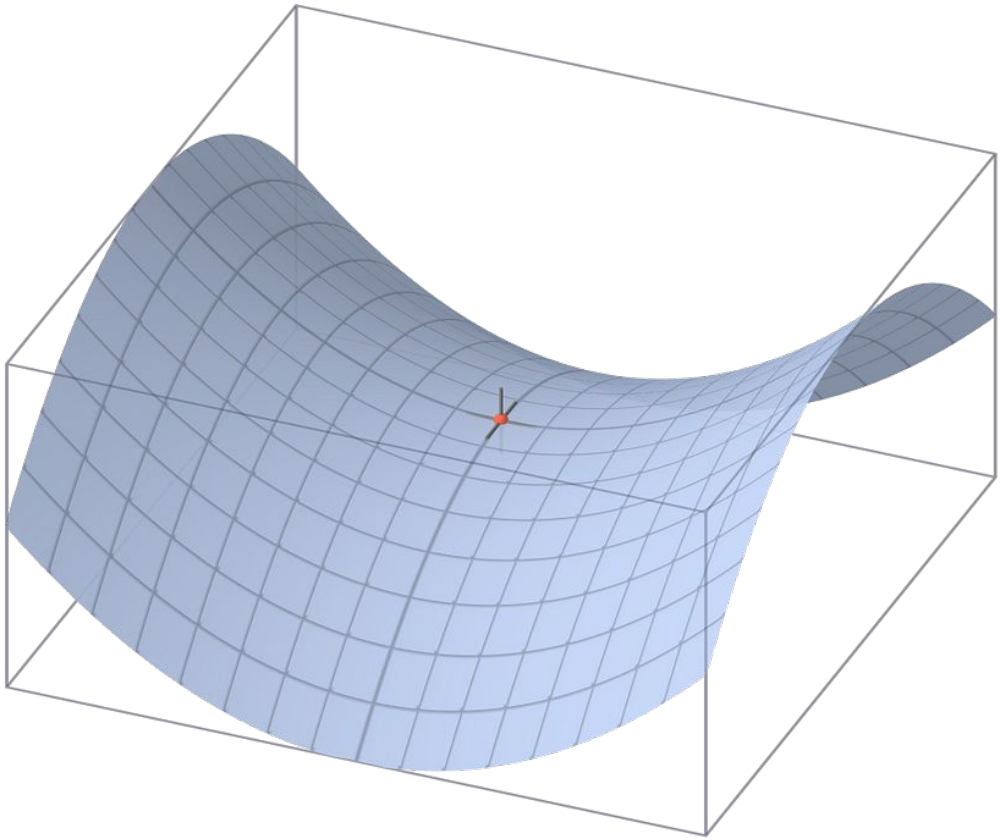
Challenge with Gradient Descent: Loss function



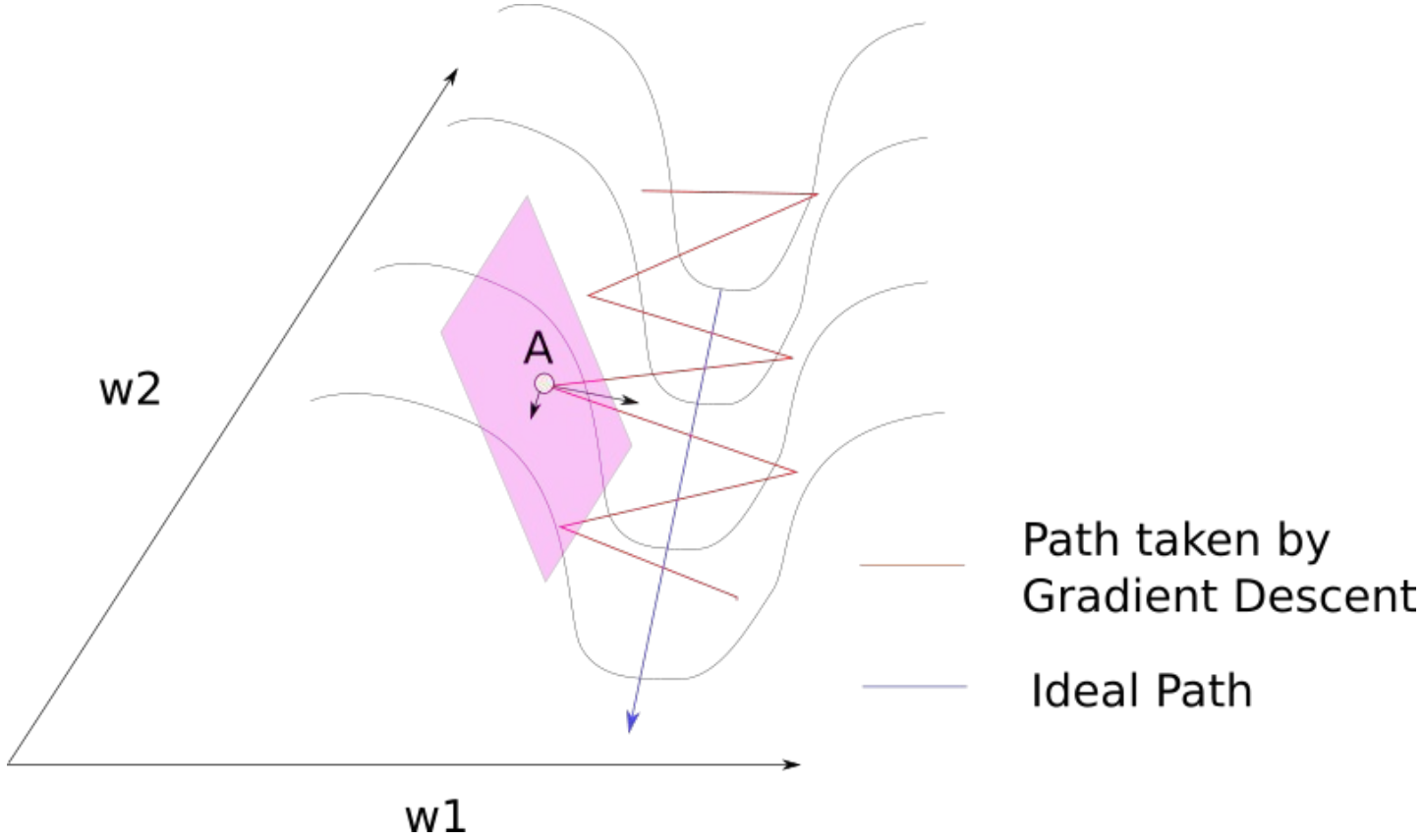
Challenge with Gradient Descent: Local minima



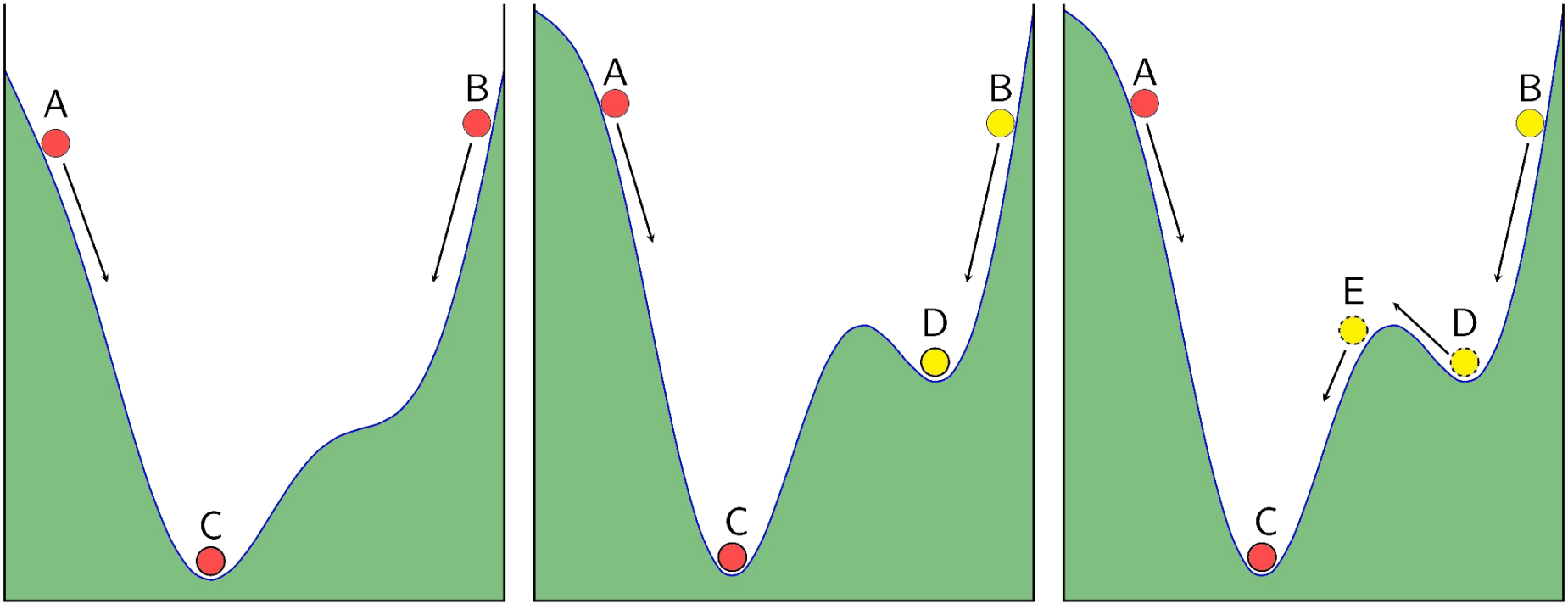
Challenge with Gradient Descent: Saddle points



Challenge with Gradient Descent: Ravines



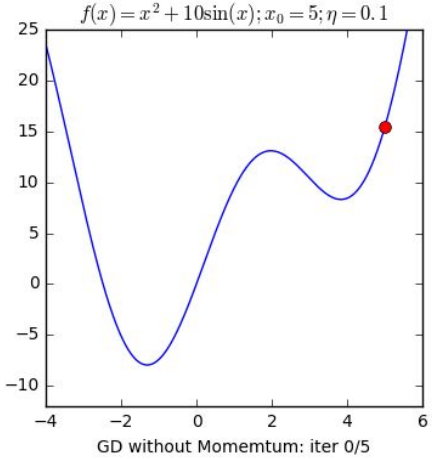
Momentum



Momentum

Gradient Descent Update Rule

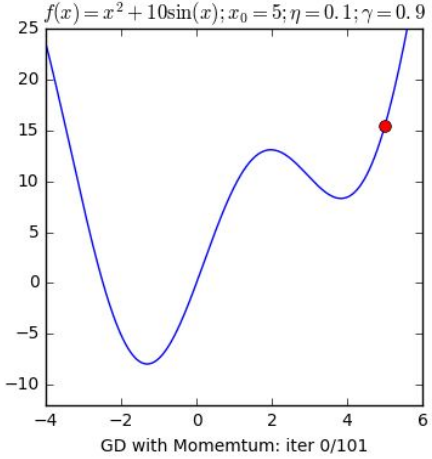
$$w_{t+1} = w_t - \eta \nabla w_t$$



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

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

$\mathbb{R}^{d \times d}$

$$G_t = \begin{bmatrix} \cdots & & \cdots & \\ & \ddots & & \\ \cdots & & \cdots & \\ & & & \ddots \end{bmatrix}$$

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

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

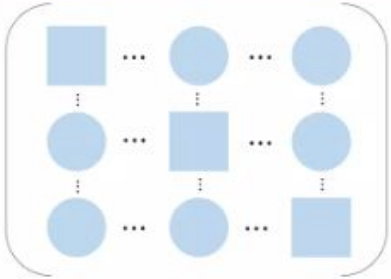
Adaptive Gradient Descent (2)

SGD

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

$\mathbb{R}^{d \times d}$

$$G_t =$$



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

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

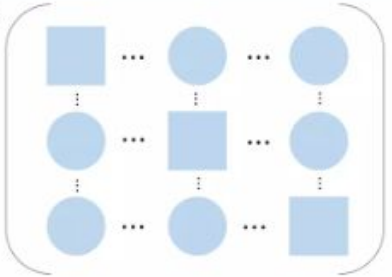
Adaptive Gradient Descent (3)

SGD

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

$\mathbb{R}^{d \times d}$

$G_t =$



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

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

$\mathbb{R}^{d \times d}$

$$G_t = \begin{pmatrix} \square & \cdots & \bigcirc & \cdots & \bigcirc \\ \vdots & & \vdots & & \vdots \\ \bigcirc & \cdots & \square & \cdots & \bigcirc \\ \vdots & & \vdots & & \vdots \\ \bigcirc & \cdots & \bigcirc & \cdots & \square \end{pmatrix}$$

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

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

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

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

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

BackPropagation

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

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

Inside function

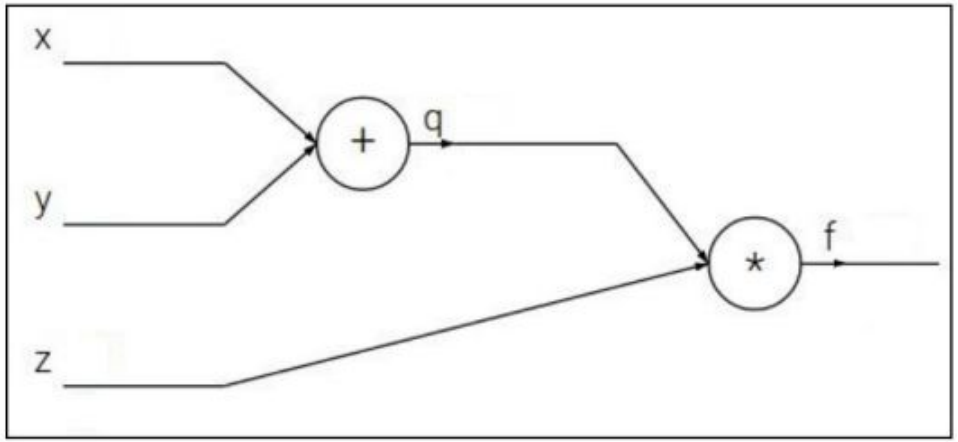
$$\frac{dy}{dx}=10\left(12x^2-10x+6\right)\left(4x^3-5x^2+6x-7\right)^9$$

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

Backpropagation

Backpropagation: a simple example

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

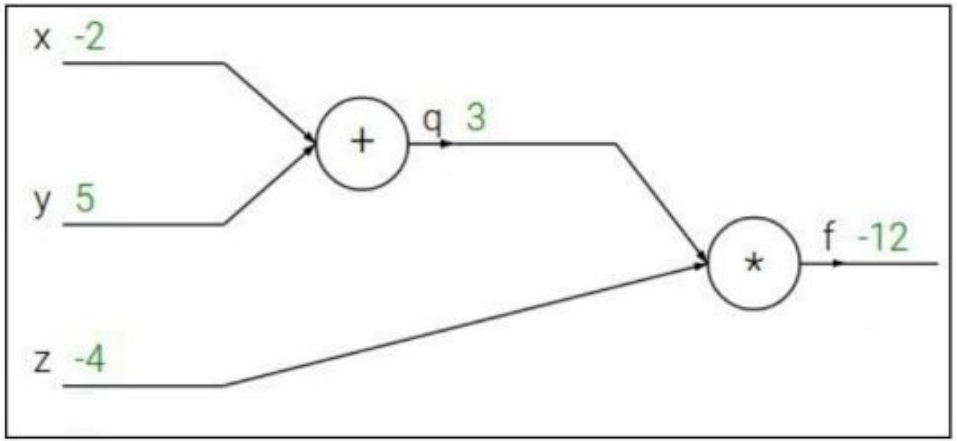


BackPropagation - example

Backpropagation: a simple example

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

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



forward

backward

BackPropagation - example

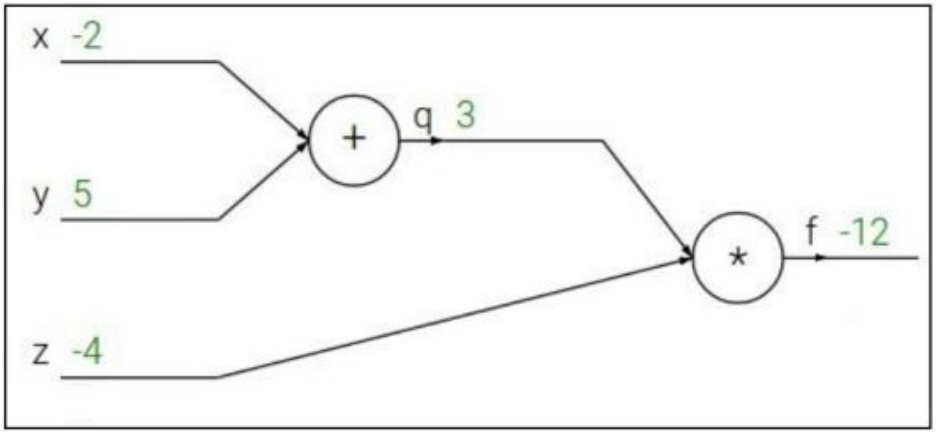
Backpropagation: a simple example

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

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

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

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



forward

backward

BackPropagation - example

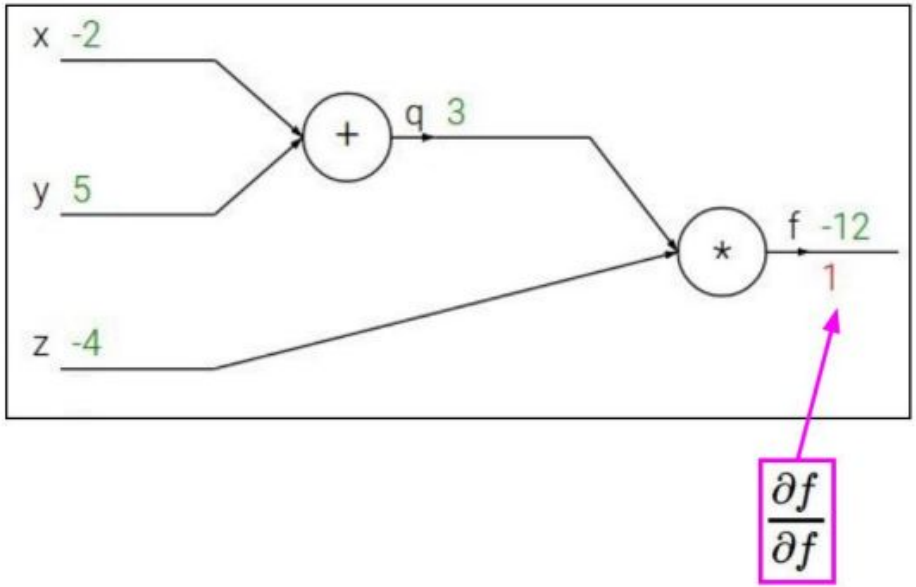
Backpropagation: a simple example

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

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

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

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



forward

backward

BackPropagation - example

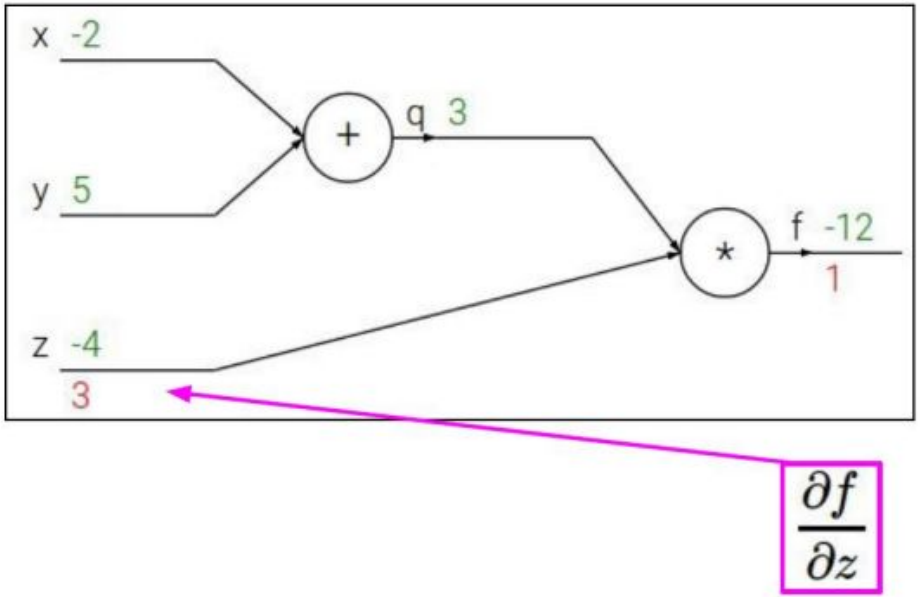
Backpropagation: a simple example

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

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

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

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



forward

backward

BackPropagation - example

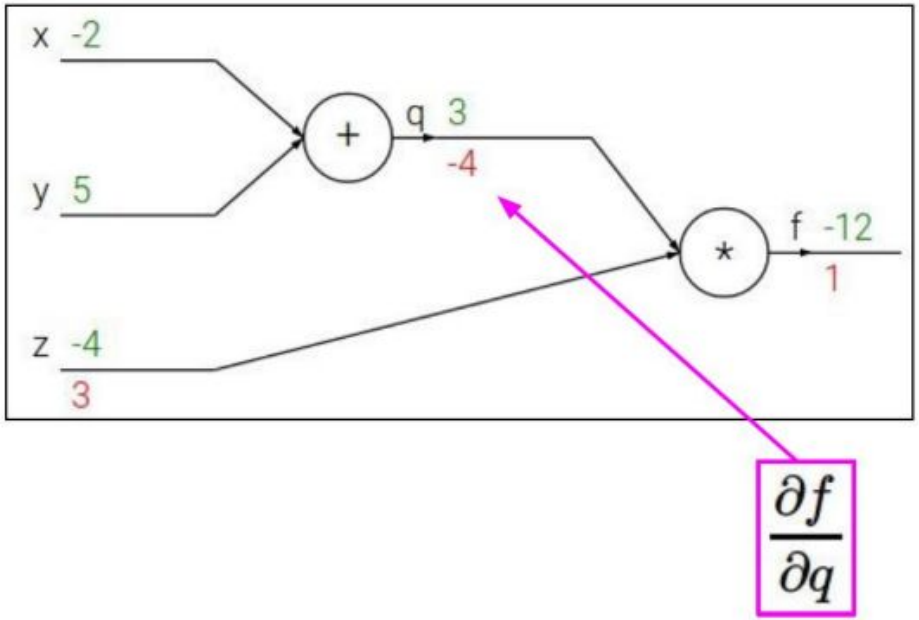
Backpropagation: a simple example

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

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

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

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



forward

backward

BackPropagation - example

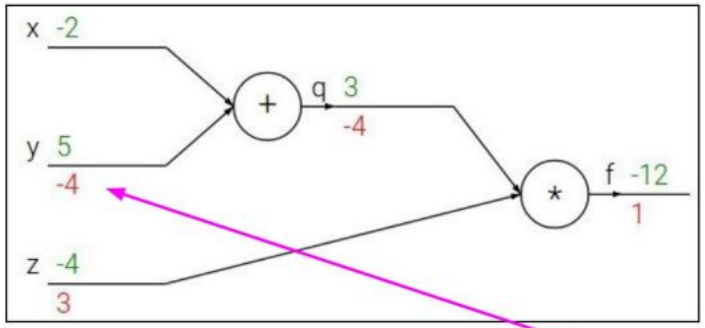
Backpropagation: a simple example

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

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

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

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



Chain rule:

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

Upstream gradient

Local gradient

forward

backward

BackPropagation - example

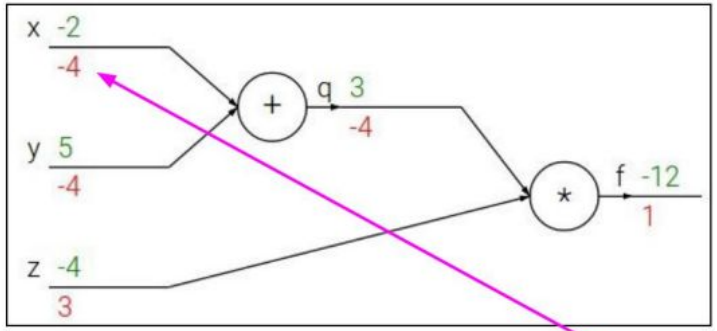
Backpropagation: a simple example

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

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

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

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



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

Chain rule:

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

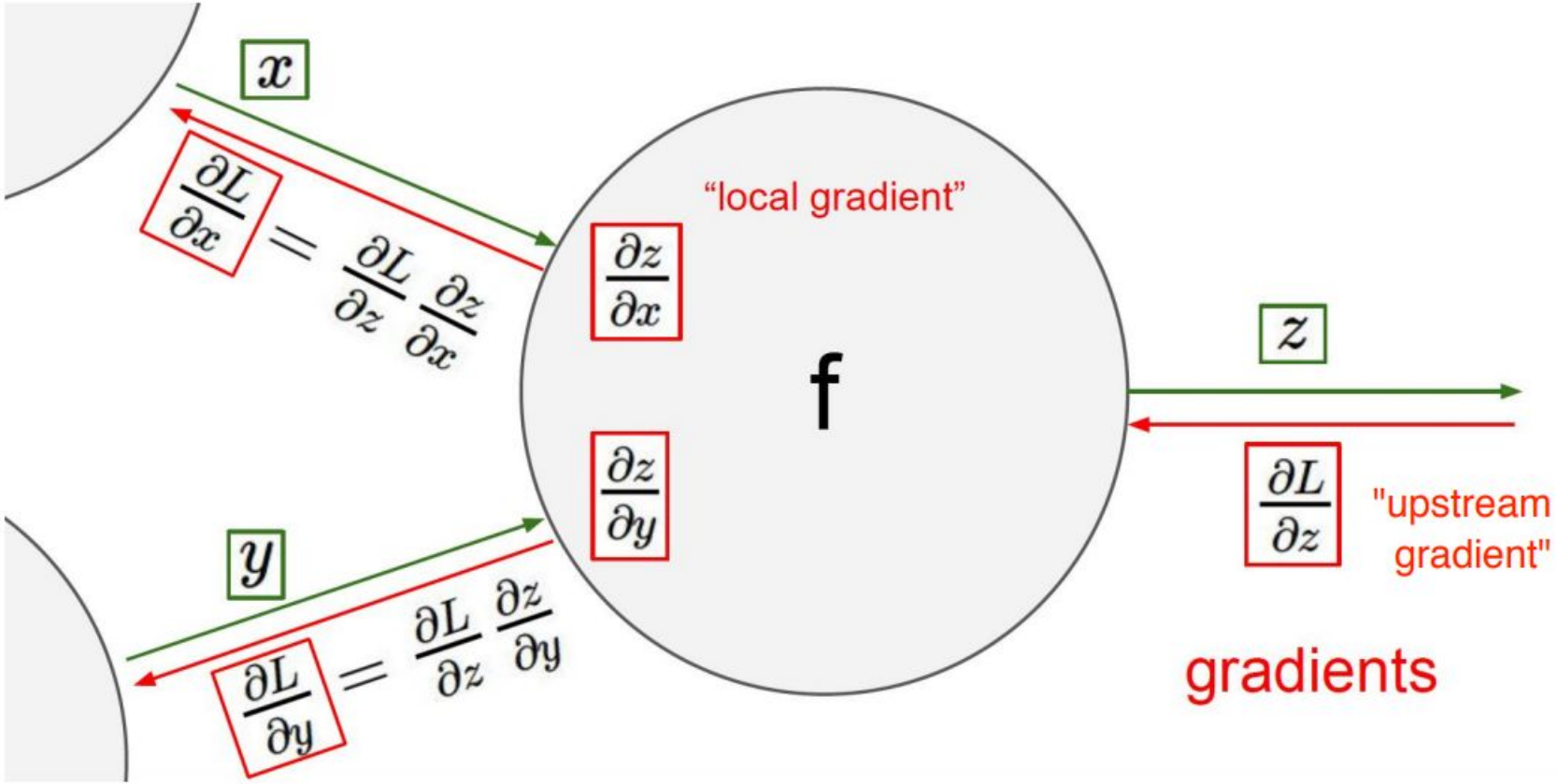
Upstream gradient

Local gradient

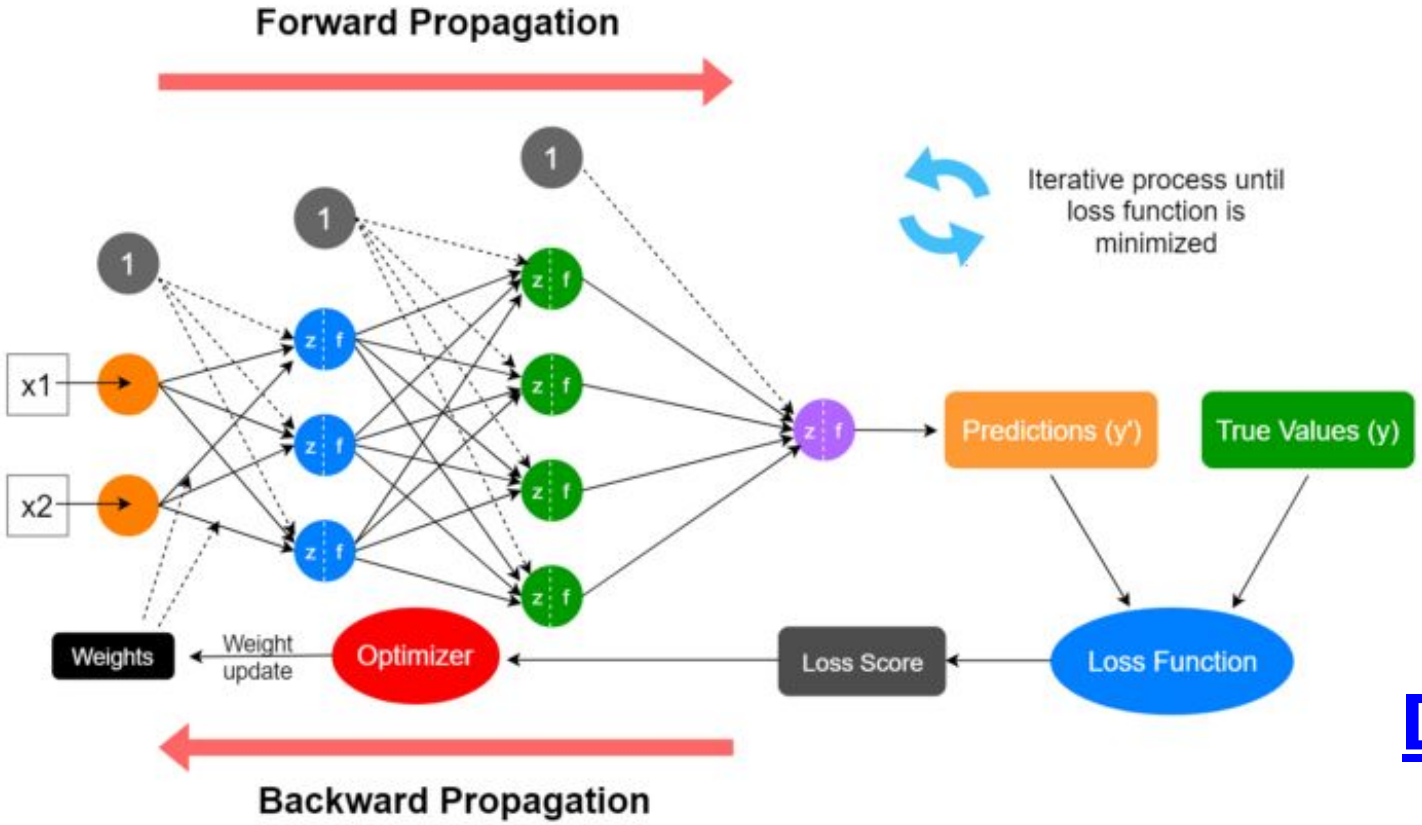
forward

backward

BackPropagation - explanation

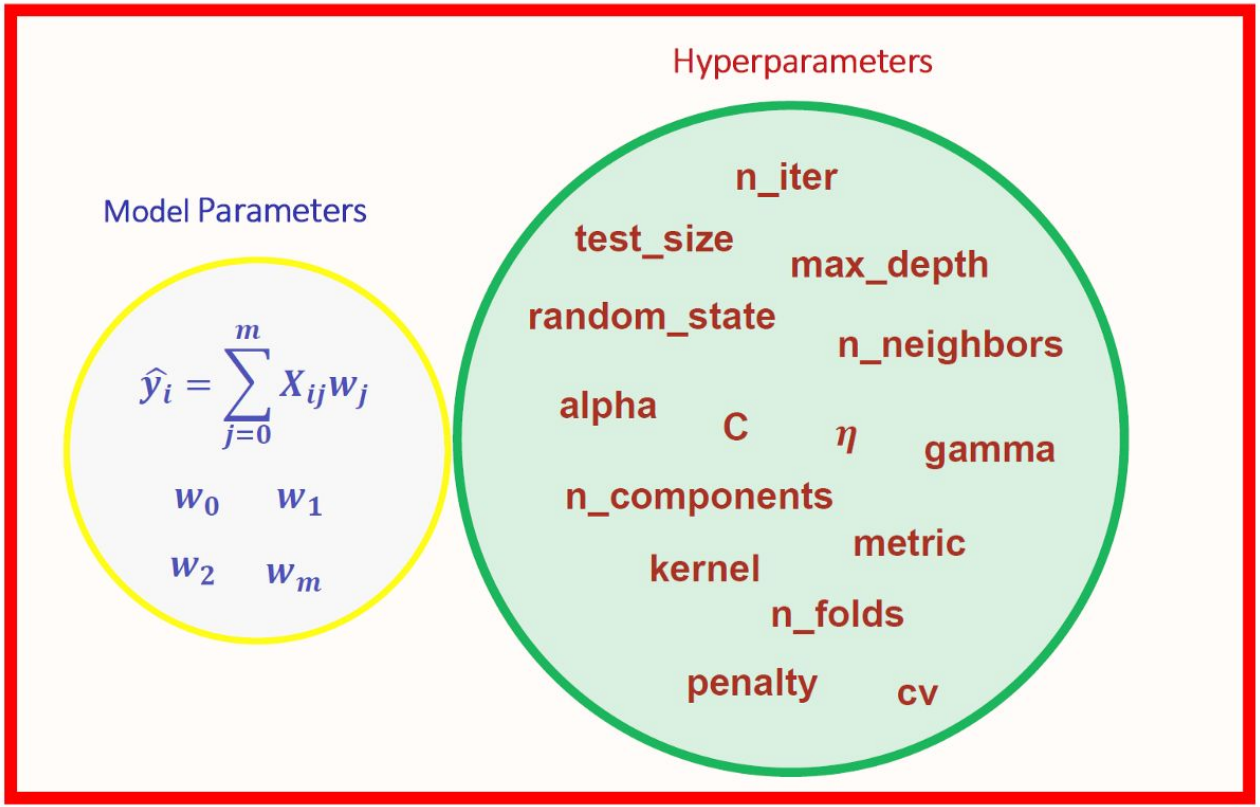


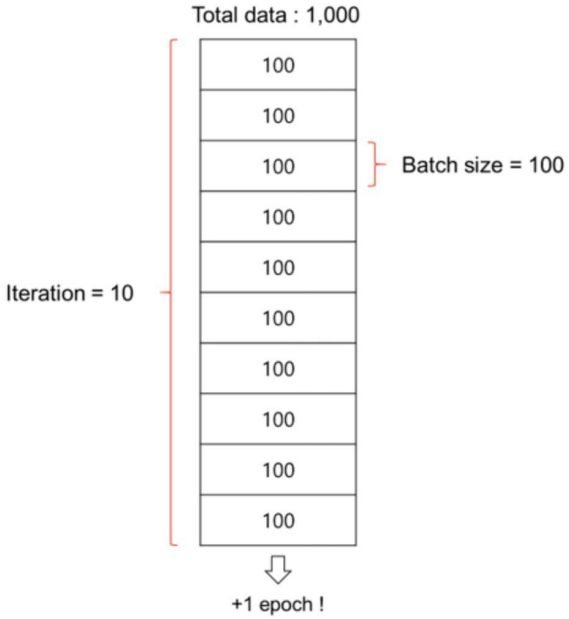
Neural network's learning process



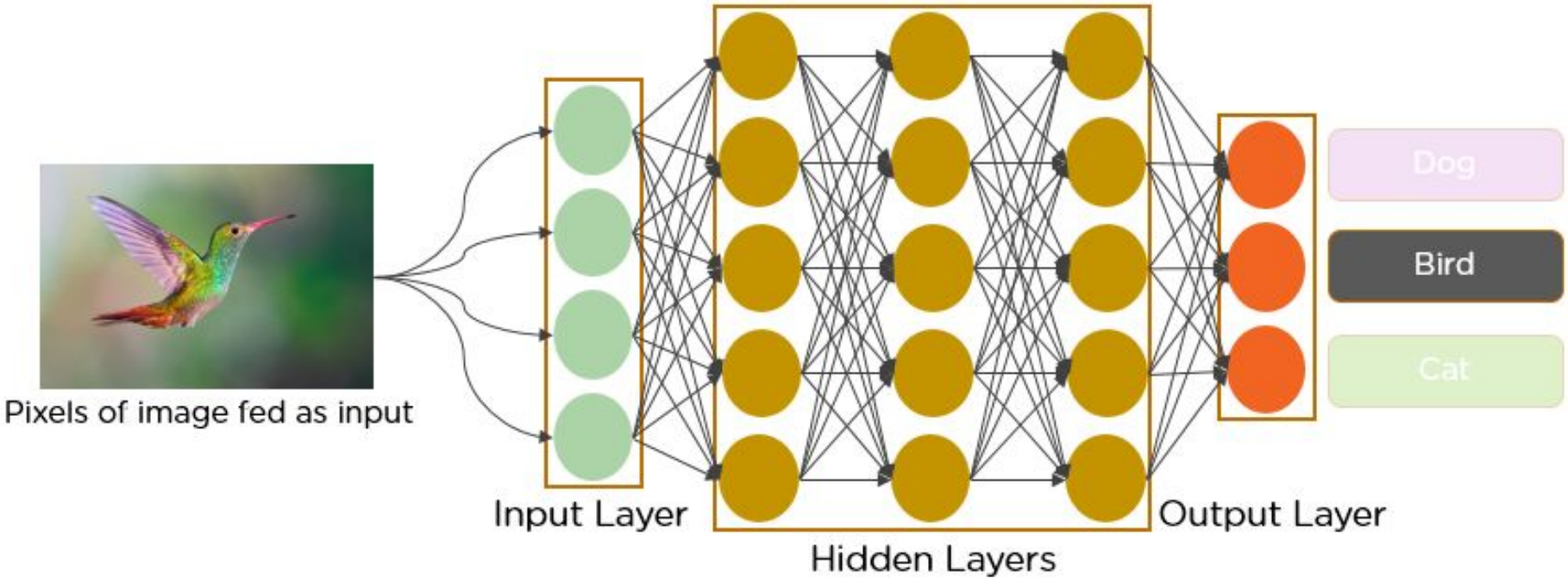
Demo

Parameters vs Hyperparameters

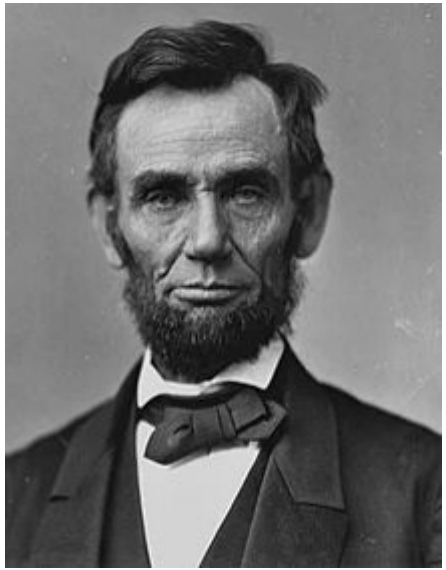


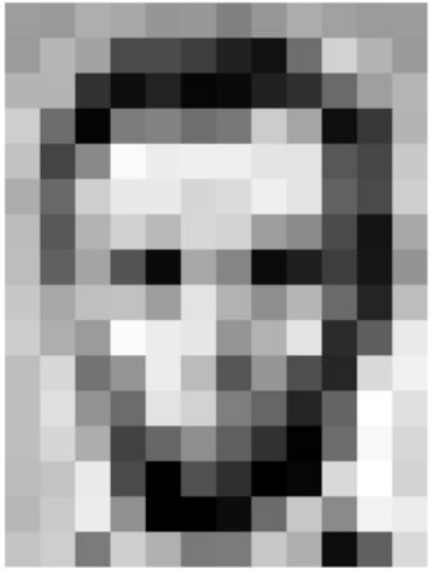


Neural Network



Gray image is represented in Computer Vision

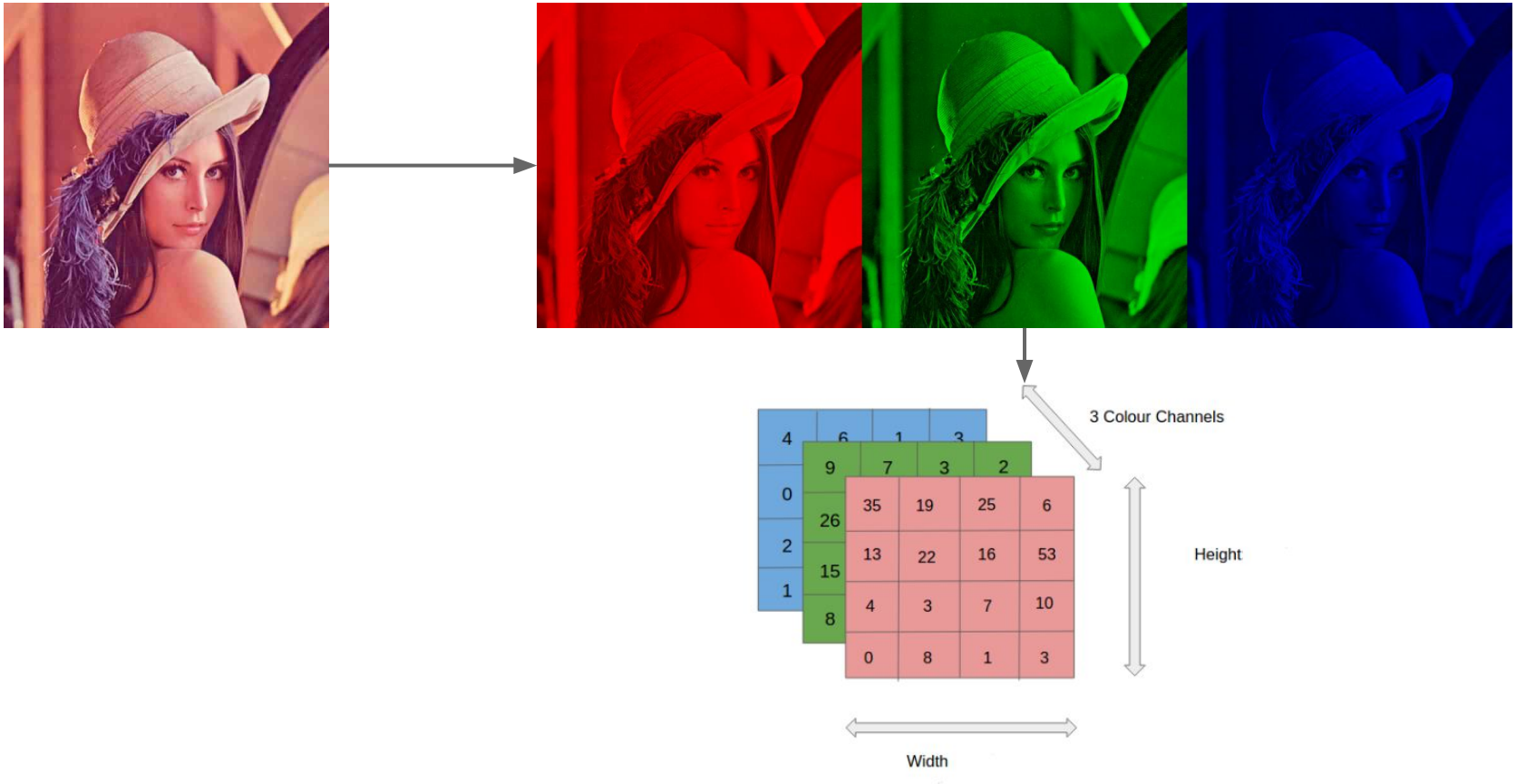




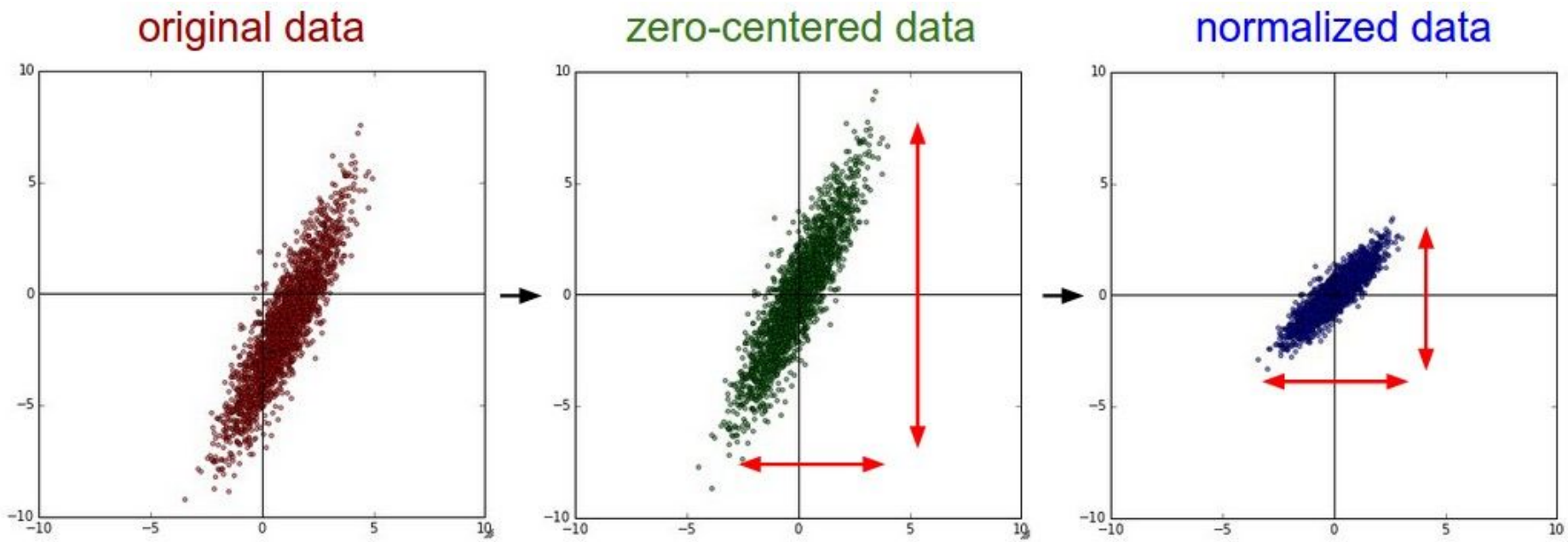
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	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
190	224	147	108	227	210	127	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
196	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	239	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	182	106	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
190	224	147	108	227	210	127	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
196	206	123	207	177	121	123	200	175	13	96	218

Color image is represented in Computer Vision



Data preprocessing



Train a (Convolutional) Neural Network in Pytorch

Step 1 01

Setup a dataset

Xây dựng 1 dataset từ tập ảnh

Step 2 02

Define a model

Định nghĩa mô hình

Step 3 03

Define a loss function

Chọn/tự xây dựng 1 hay nhiều hàm loss

Step 4 04

Define an optimizer

Chọn 1 optimizer

Step 5 05

Train

Huấn luyện mô hình với training data

Step 6 06

Validate

Đánh giá trong quá trình huấn luyện với validation data

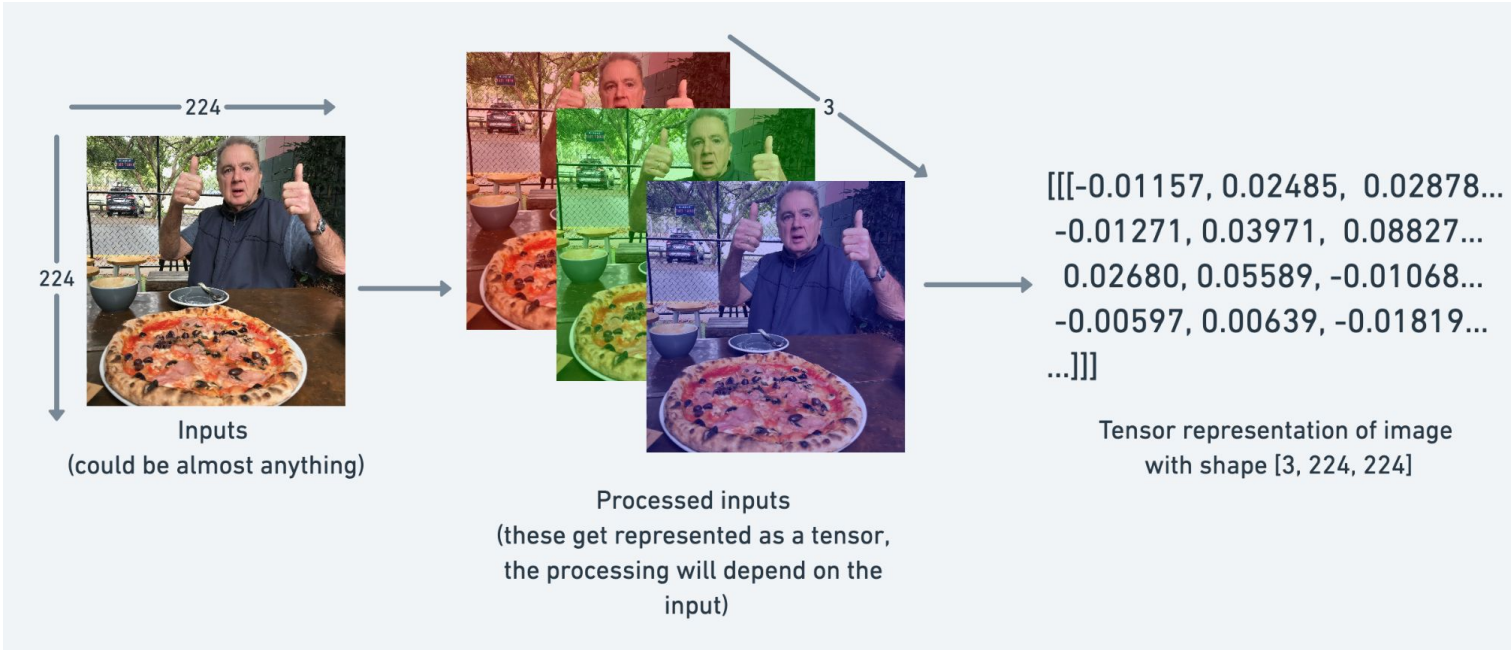
Step 7 07

Test

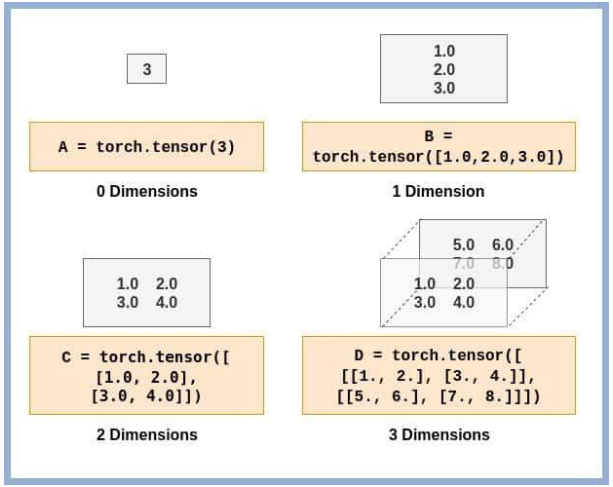
Đánh giá mô hình cuối cùng với test data

Pytorch: tensor

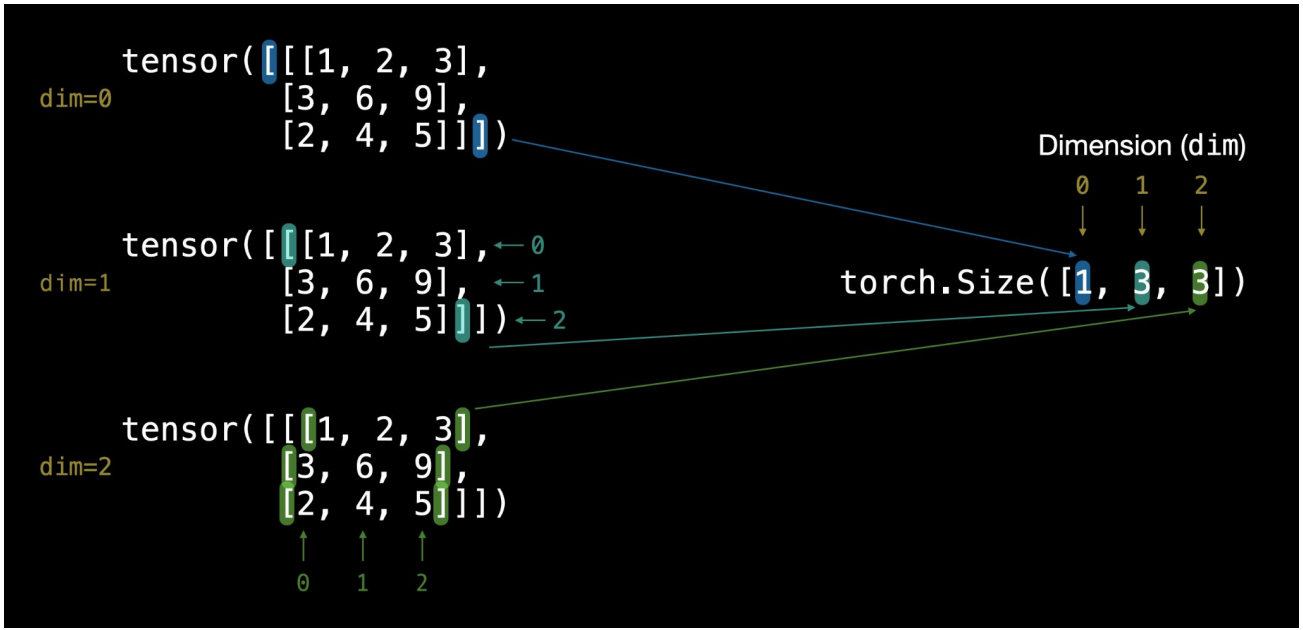
Tensor in Pytorch is equivalent to Array in Numpy



Pytorch: tensor



Pytorch: tensor




```python
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)
```

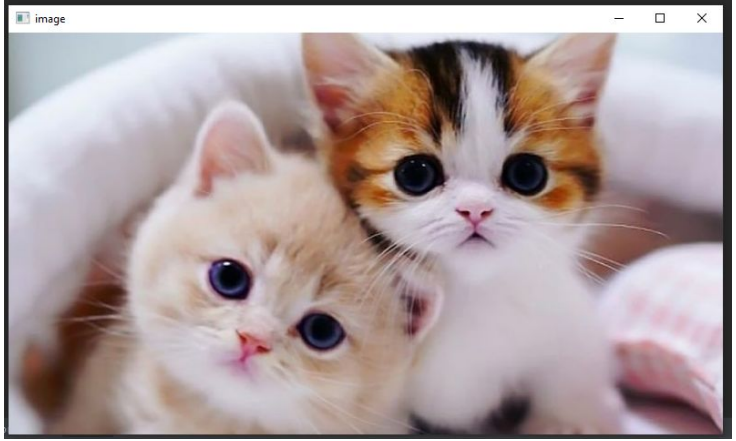
example

```
C:\Users\Viet\anaconda3\envs\basic\pyt
torch.Size([1, 3, 3])
3
torch.int64
cpu
```

Pytorch: Image to tensor

```python
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
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
```

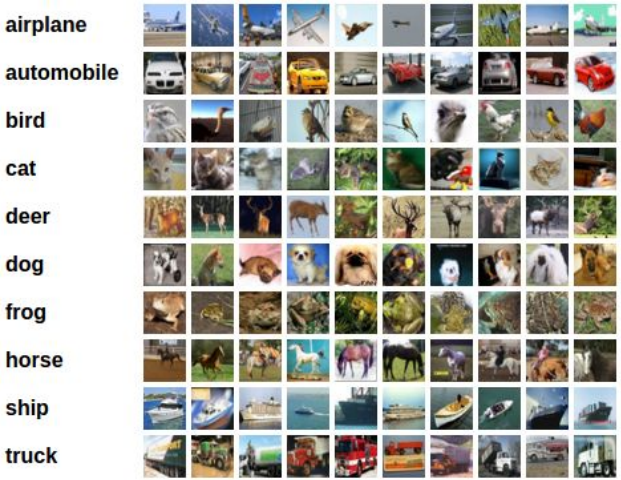



```python
from PIL import Image
from torchvision.transforms import ToTensor

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

Pytorch: Datasets

Built-in datasets



CIFAR dataset

```
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)
```

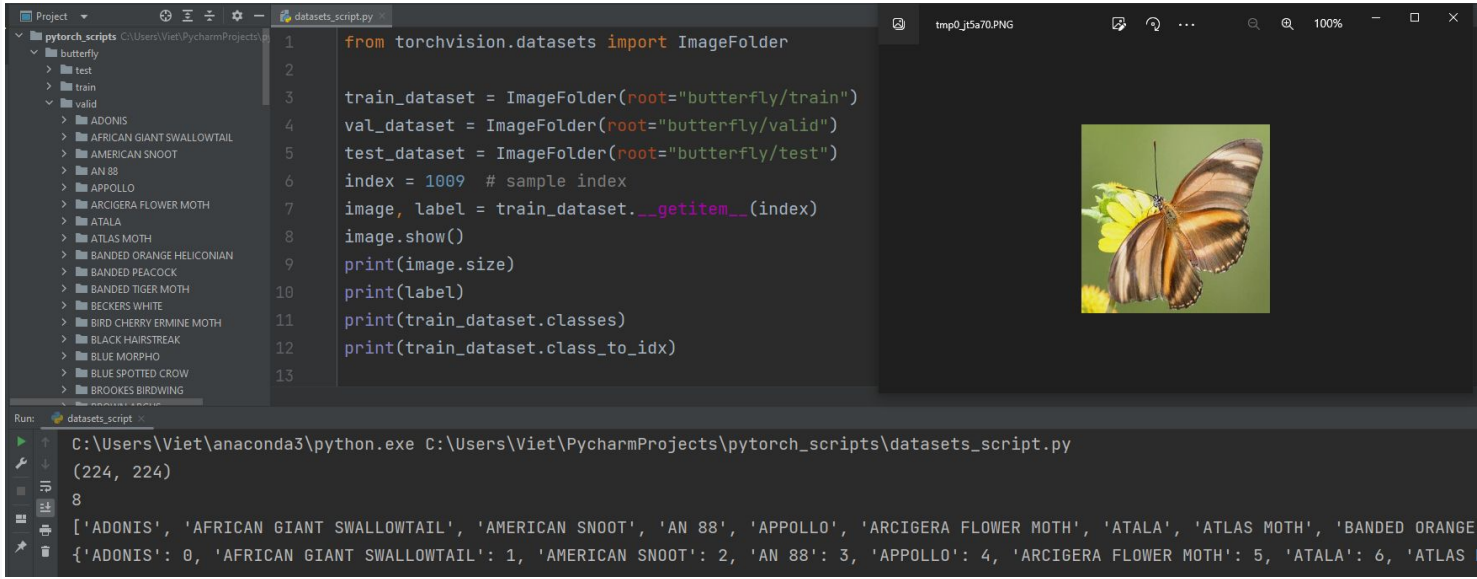
```
C:\Users\Viet\anaconda3\python.exe C:\Users\Viet\PycharmProjects\p
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.g
100%|██████████| 170498071/170498071 [00:52<00:00, 3272326.26it/s]
Extracting ./cifar\cifar-10-python.tar.gz to ./cifar
Files already downloaded and verified
(32, 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]

```
train
    class1
        1.jpg
        2.jpg
    class2
        1.jpg
        2.jpg
valid
    class1
        1.jpg
        2.jpg
    class2
        1.jpg
        2.jpg
```



Pytorch: Built-in datasets

Pytorch all built-in datasets

```
dataset = torchvision.datasets.ImageNet('path/to/imagenet_root/')
dataloader = torch.utils.data.DataLoader(imagenet_data,
                                         batch_size=4,
                                         shuffle=True,
                                         num_workers=args.nThreads)
```

Pytorch: Datasets

Requirements for a custom dataset implementation:

a subclass of torch.utils.data.Dataset

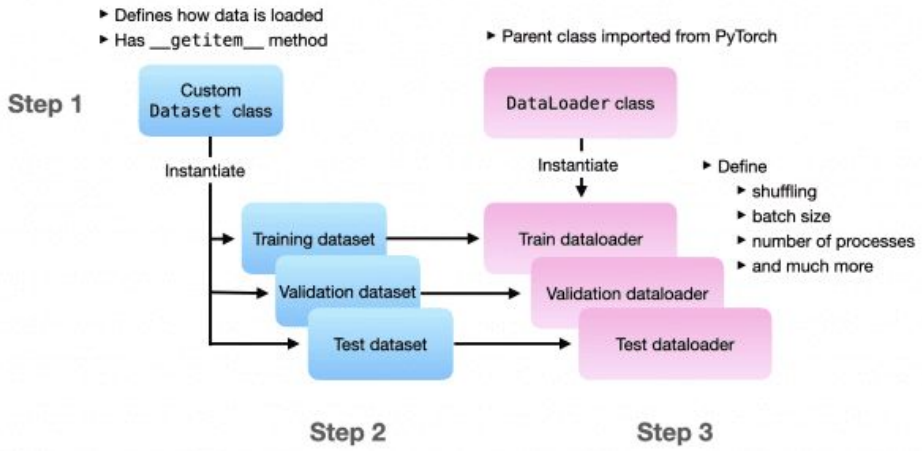
has the __len__ method implemented

has the __getitem__ method implemented

repo

```
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




```python
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

```python
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>)
```

```python
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>)
```

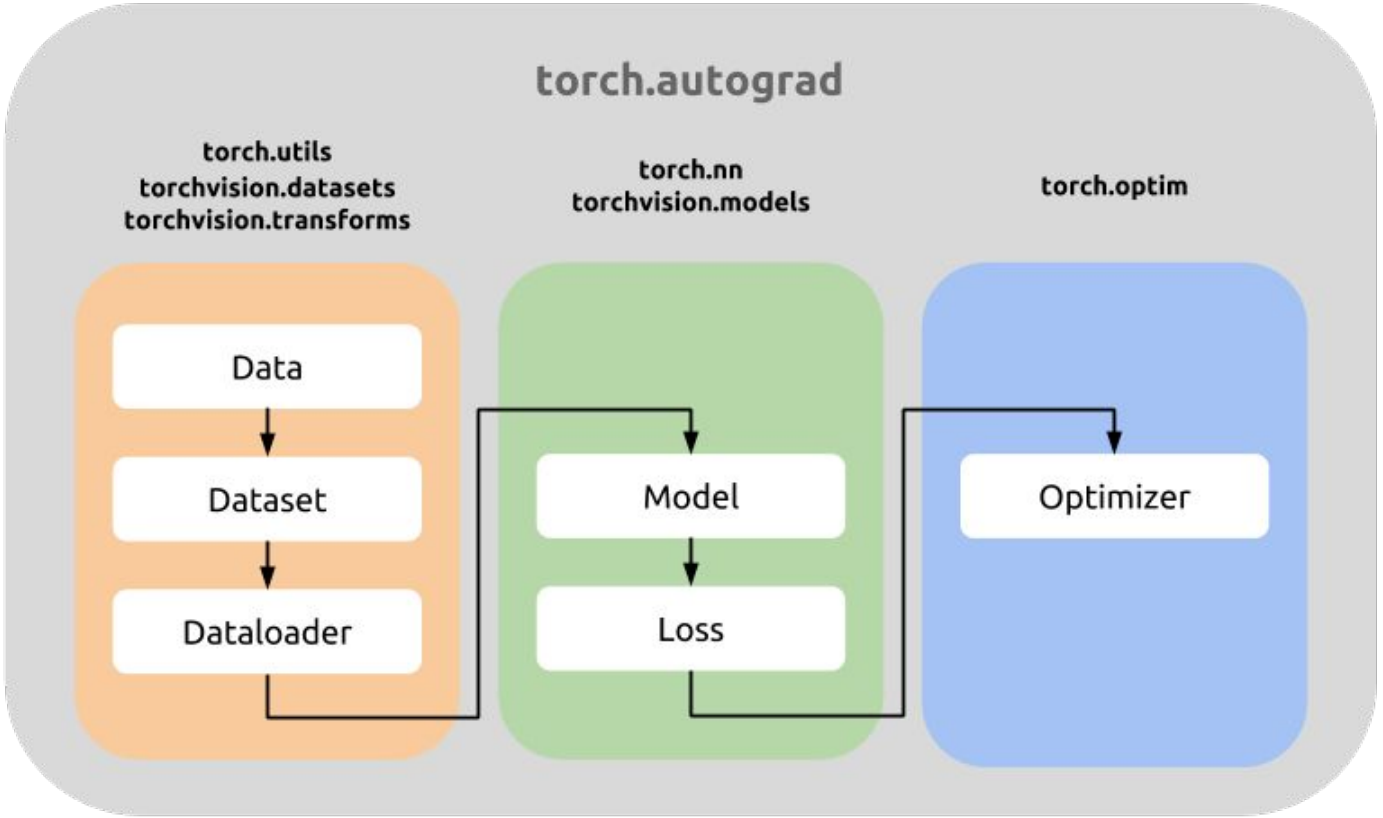
Step 3: Define a loss function and an optimizer

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

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

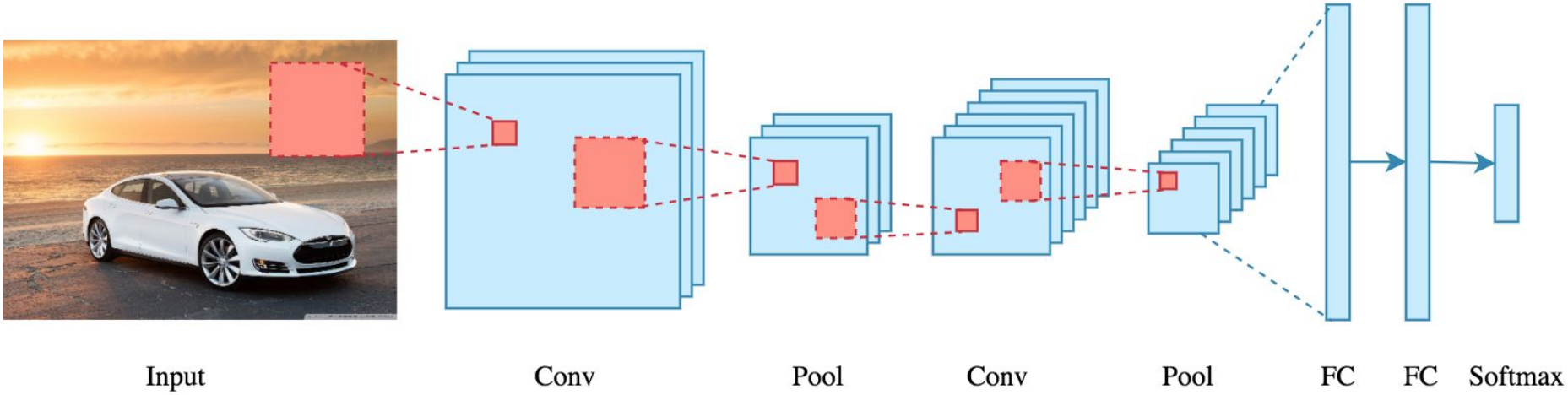
Step 4: Train the network

```
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



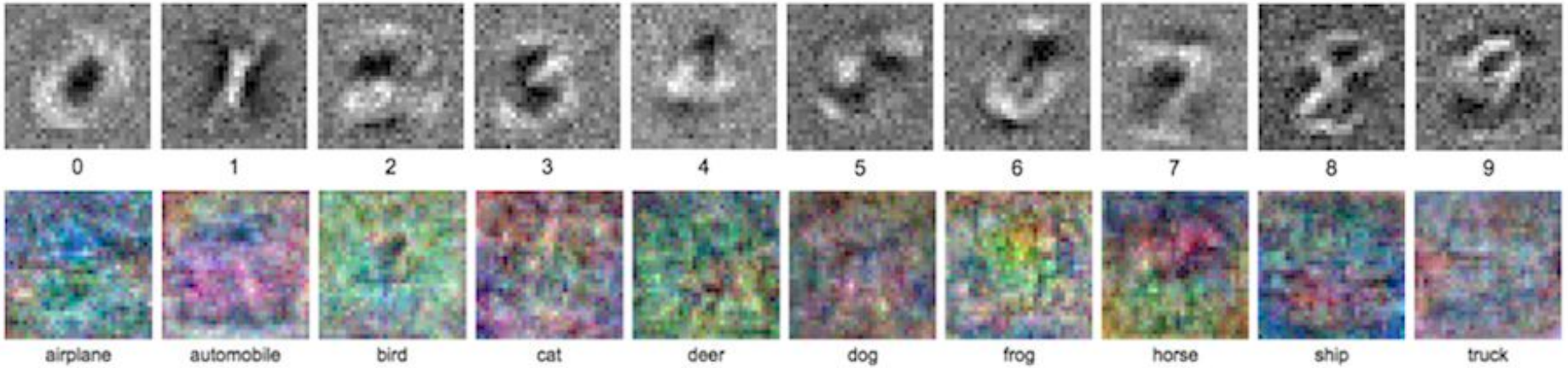
Problems in Neural Network

Variations

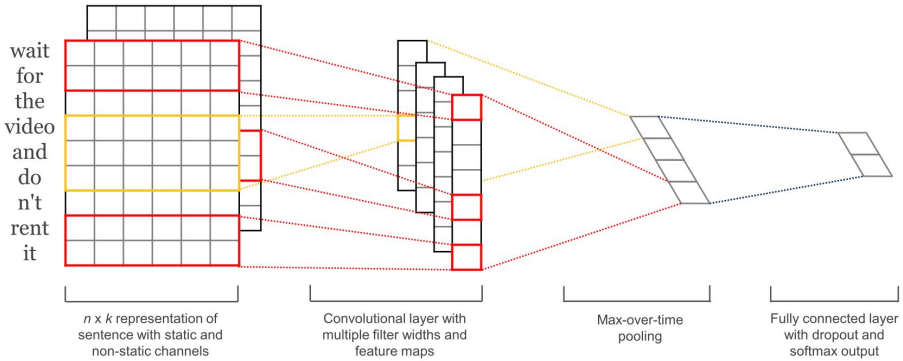
- Position
- Orientation
- Scale

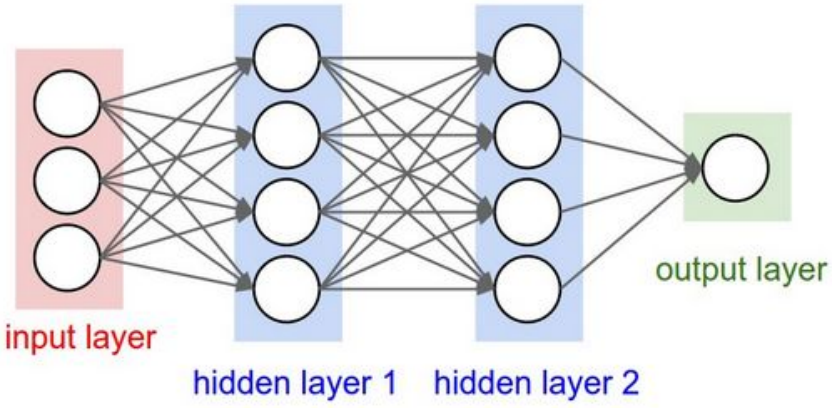
Scalable

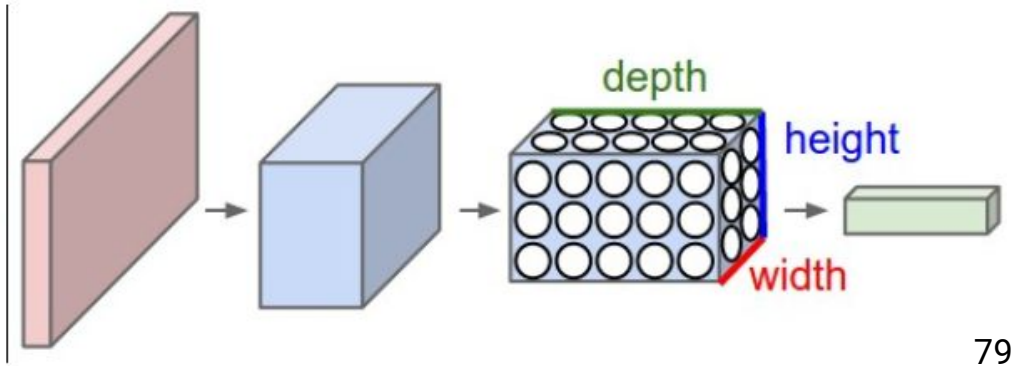
- Memory
- Calculation



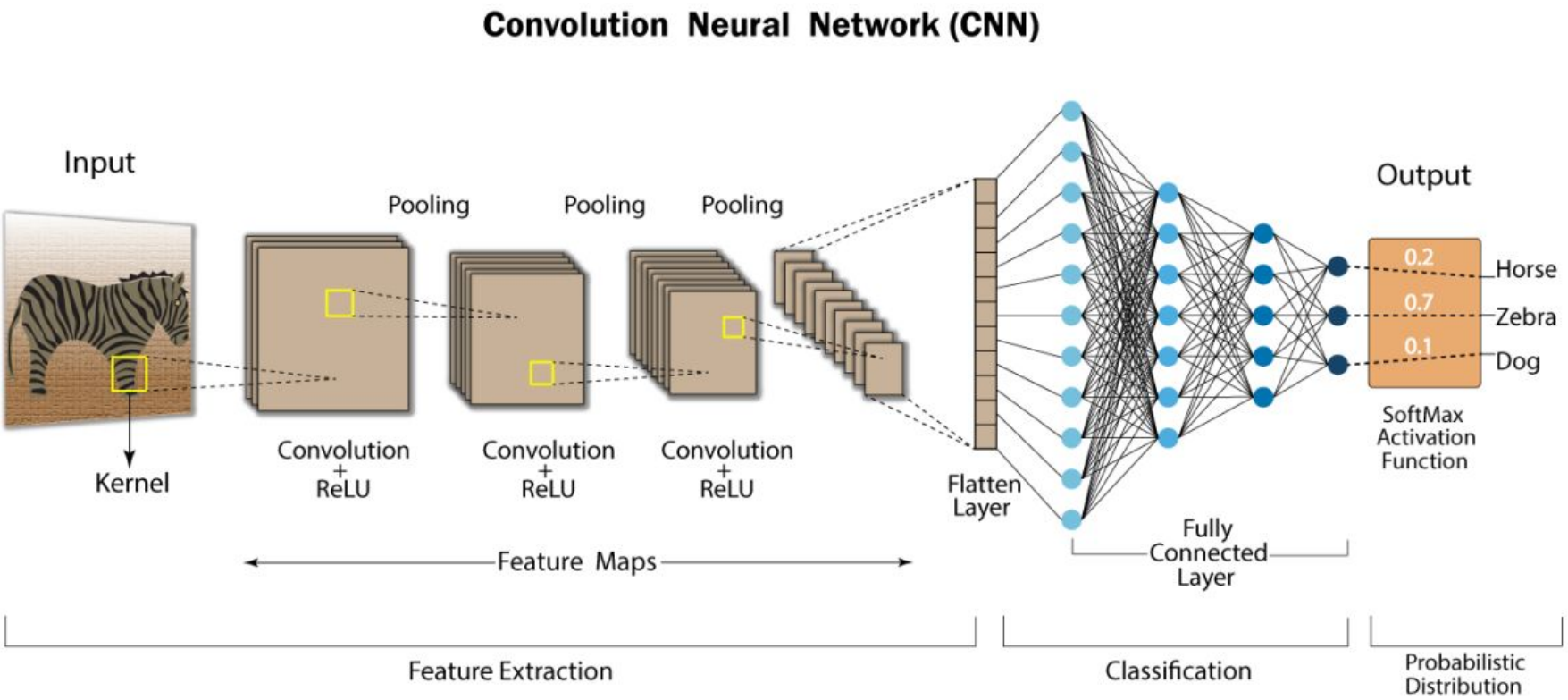
Convolutional Neural Network



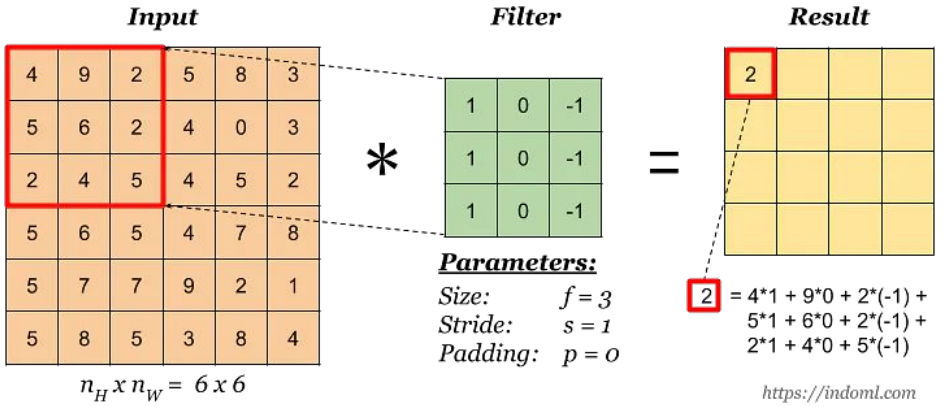


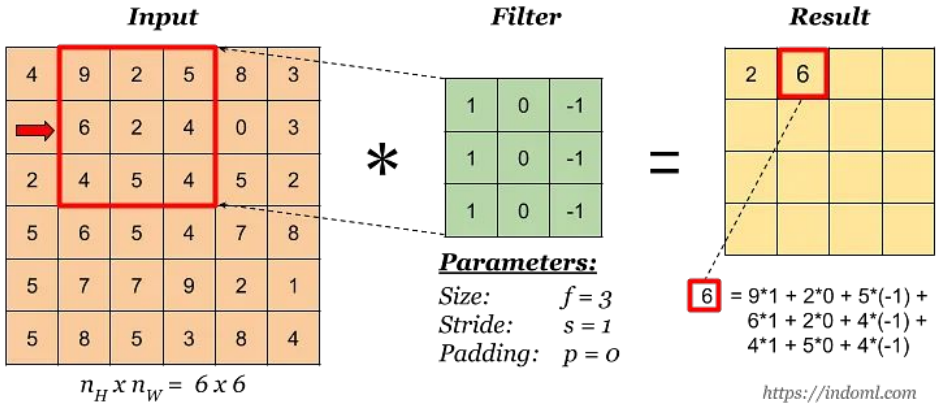


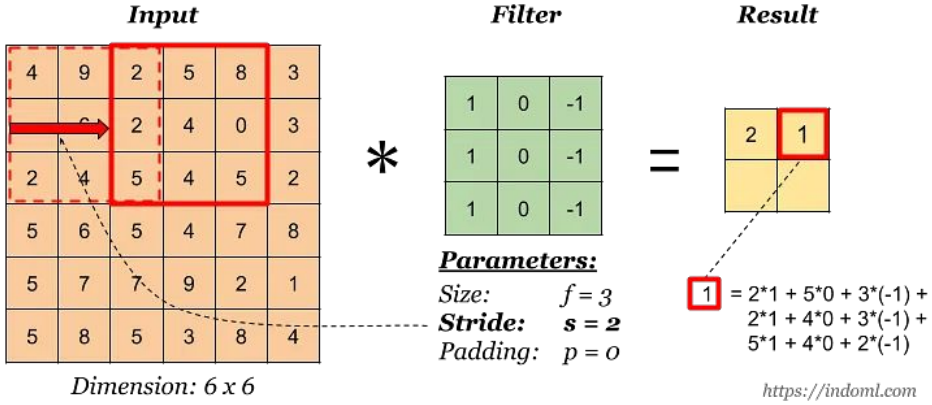
Convolutional Neural Network

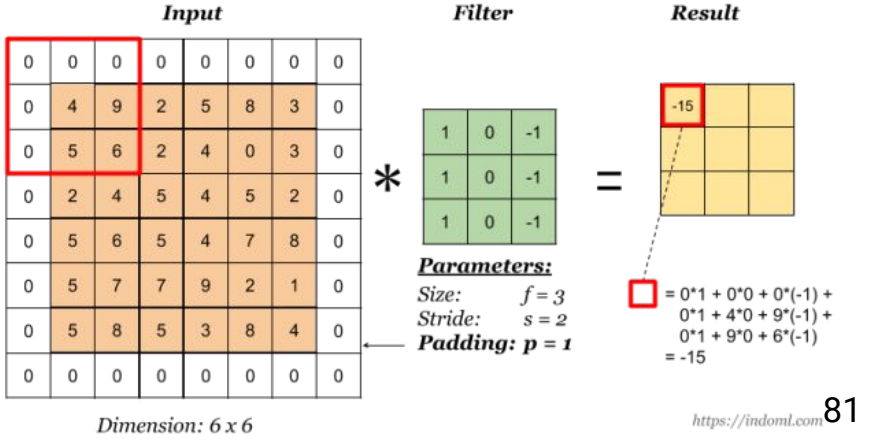


Convolutional operation

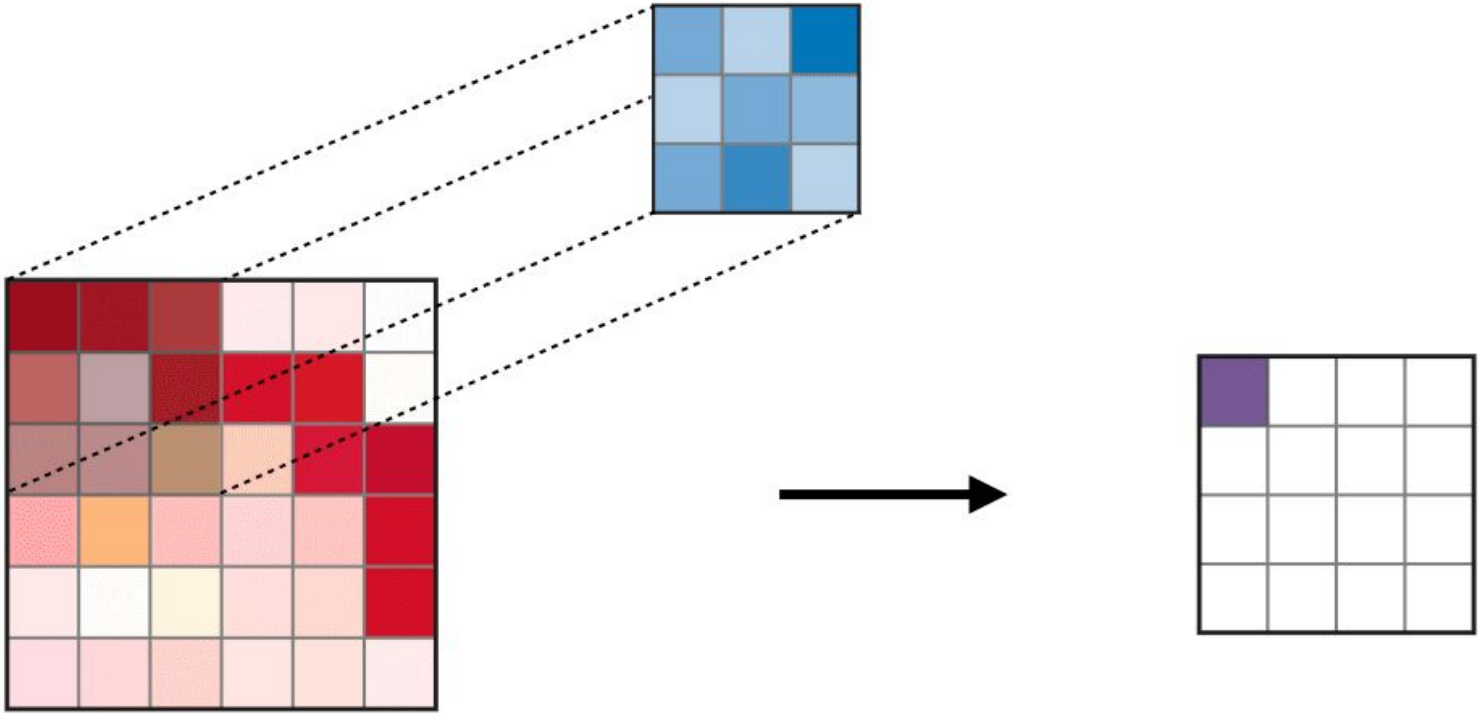




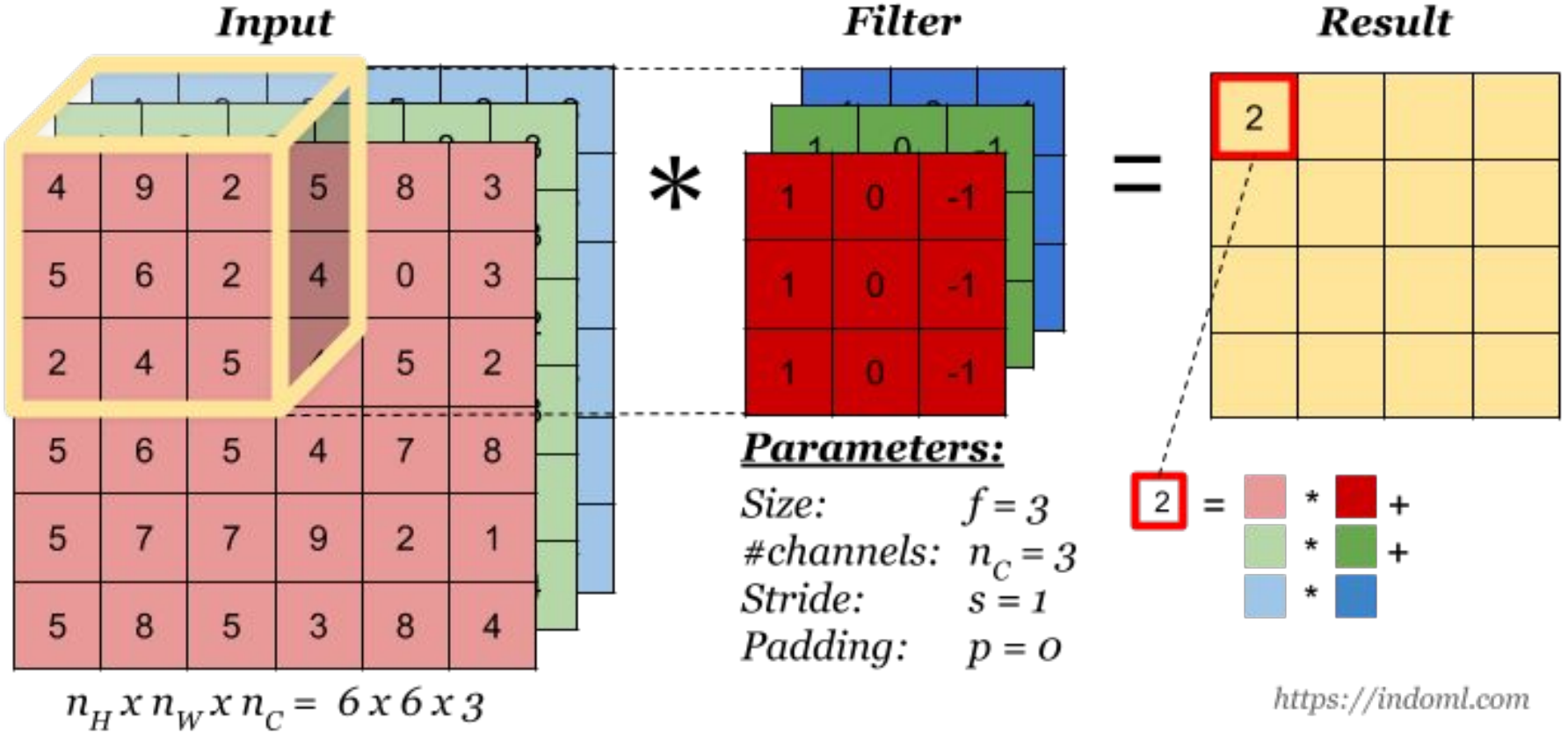




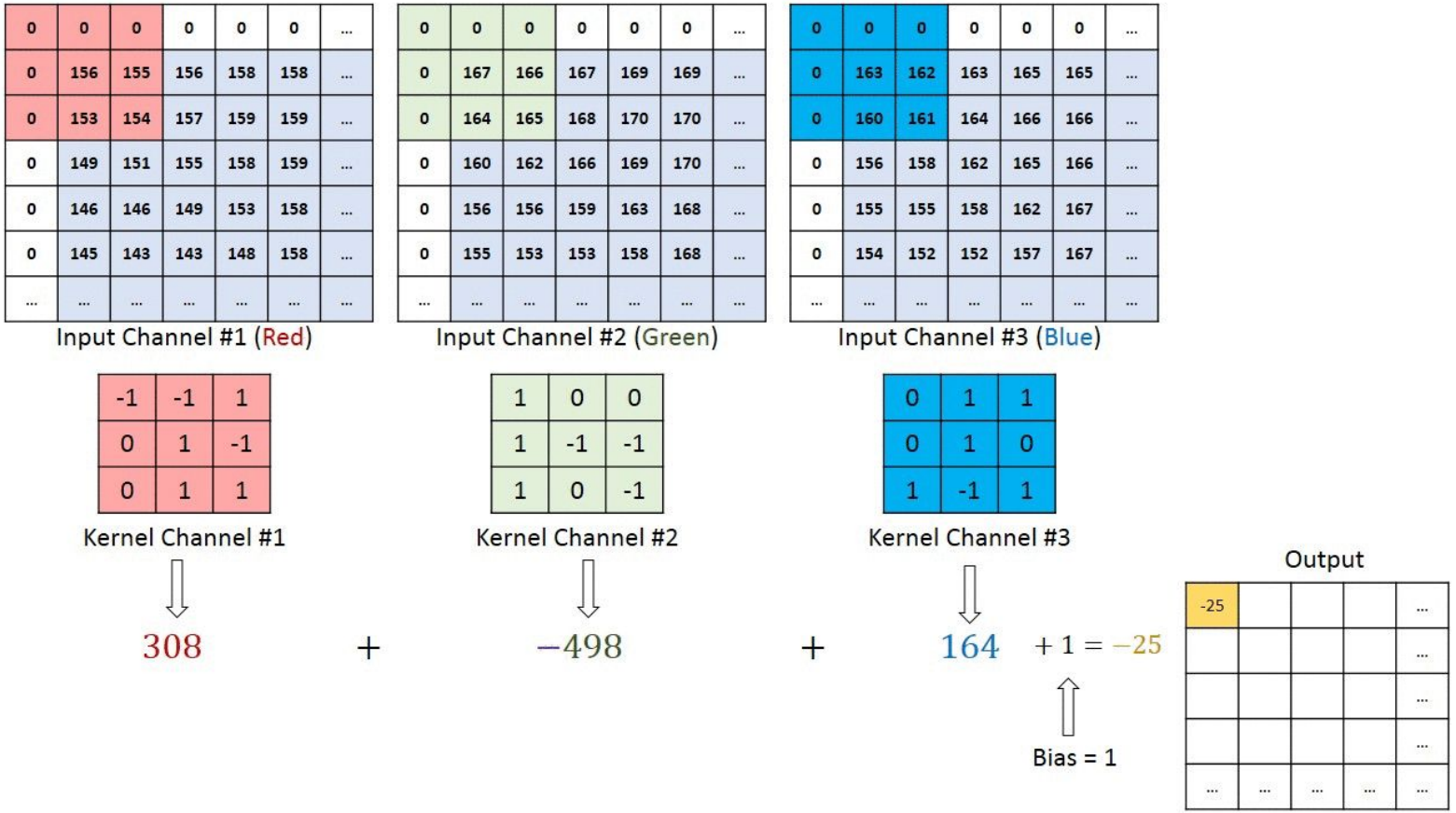
Convolutional operation



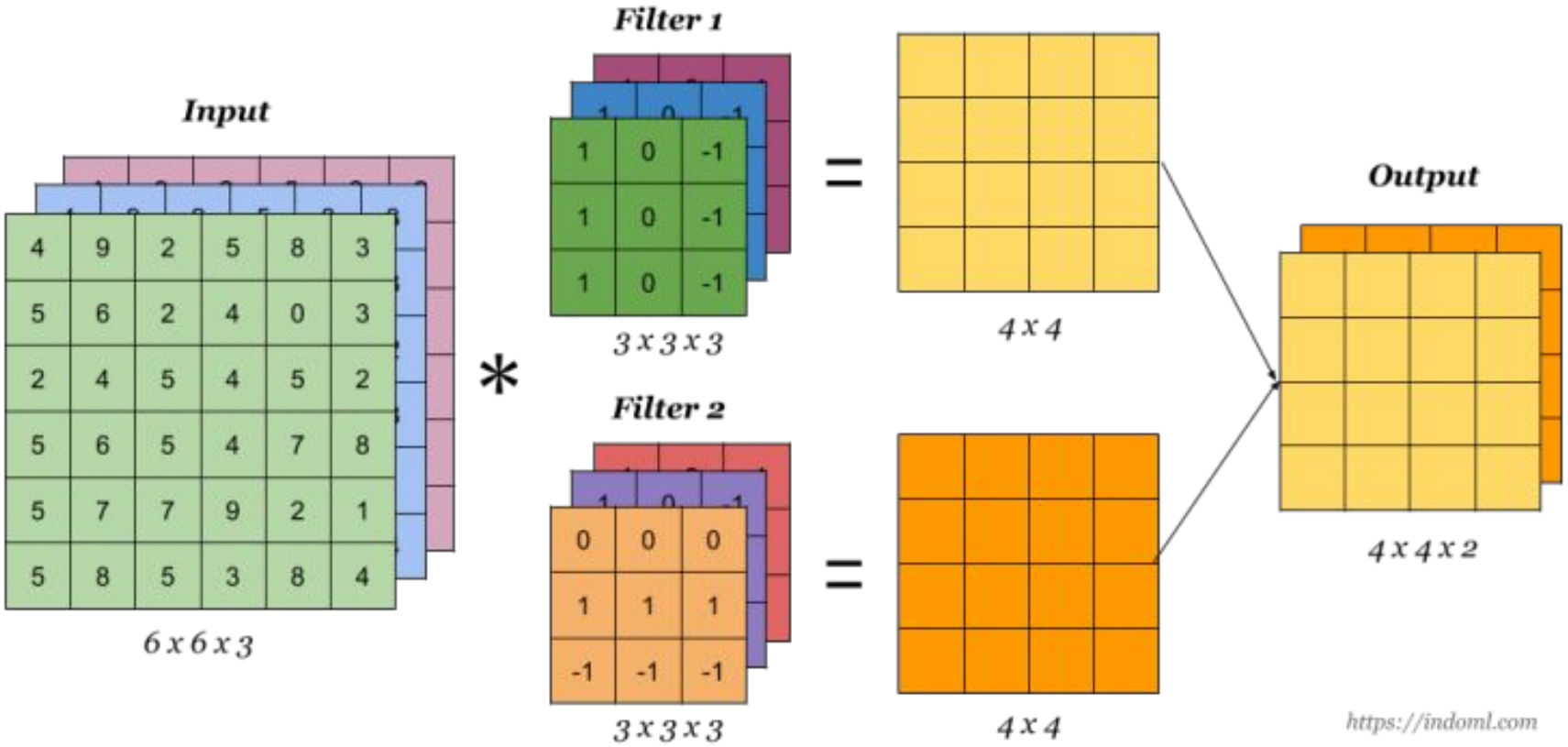
Convolutional operation on Volume



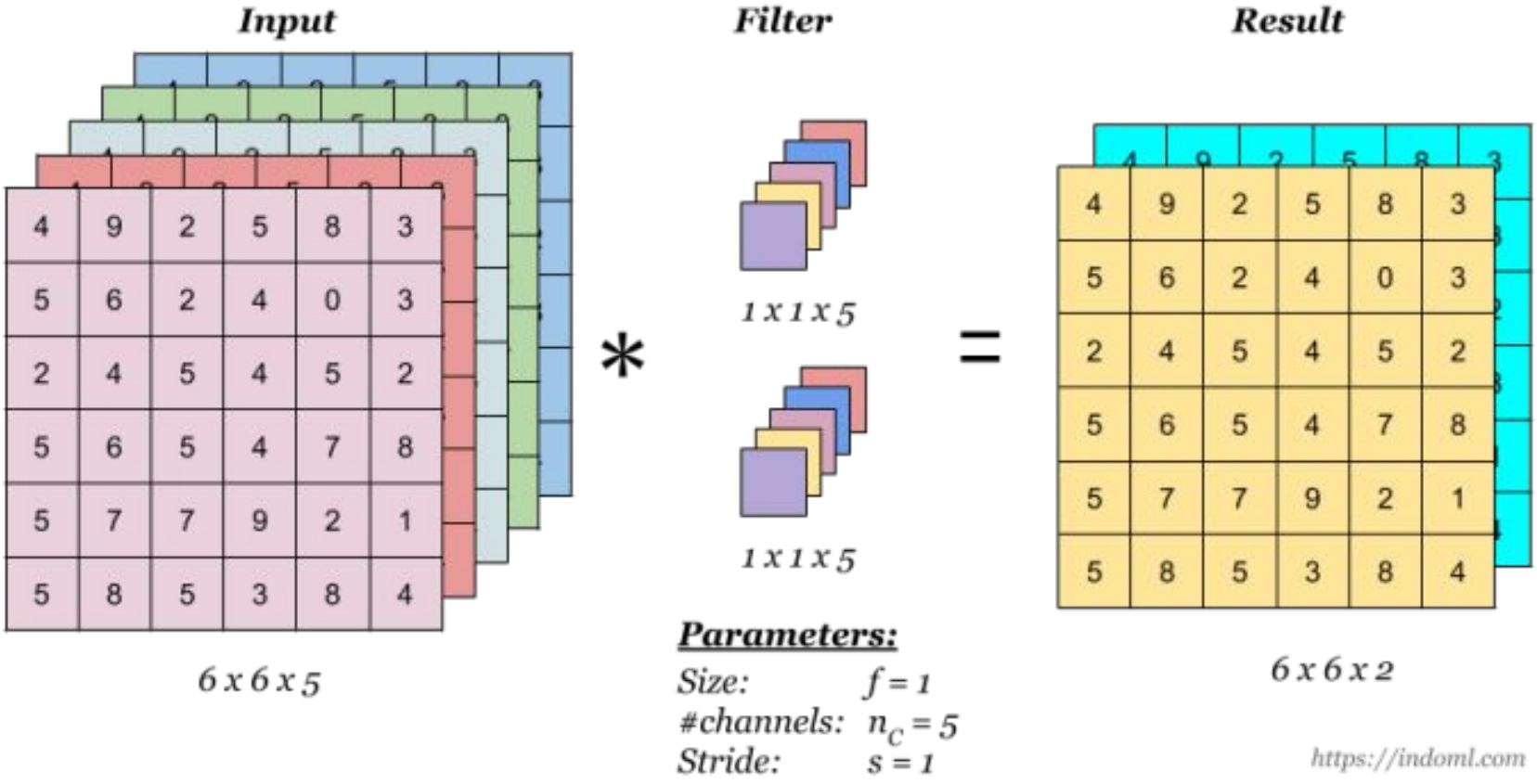
Convolutional operation on Volume



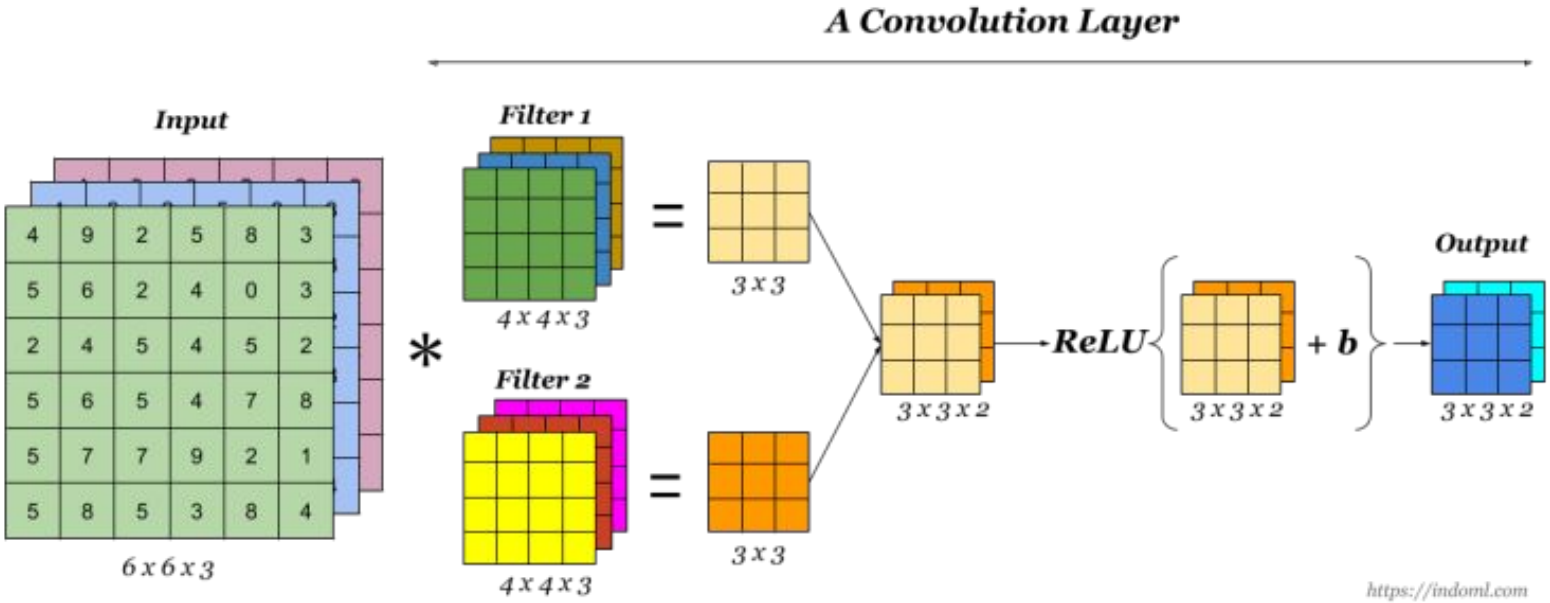
Convolutional operation with multiple Filters/Kernels



1x1 Convolutional operation



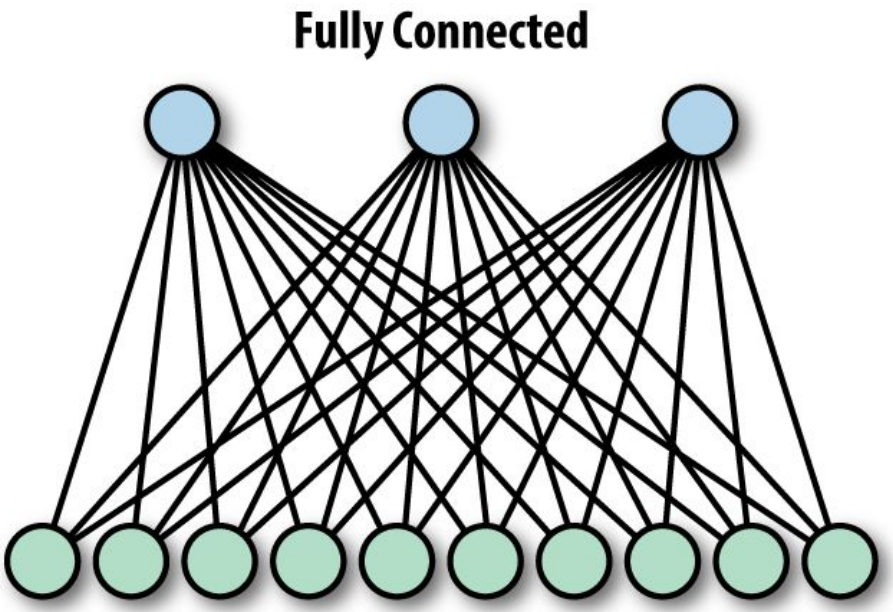
Convolutional layer

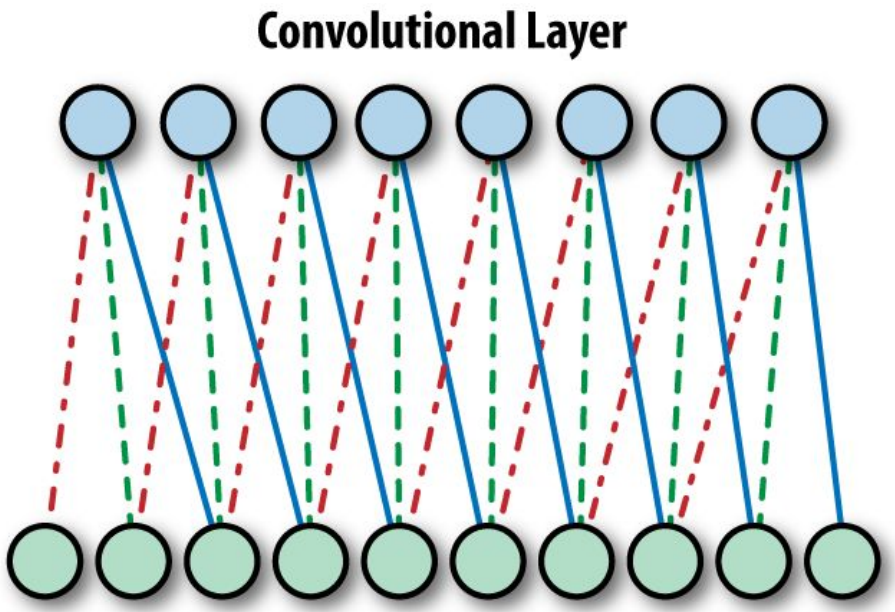


$$\text{output size} = \left[\frac{(\text{input size}) + 2*\text{padding} - (\text{kernel size} - 1) - 1}{\text{stride}} + 1 \right]$$

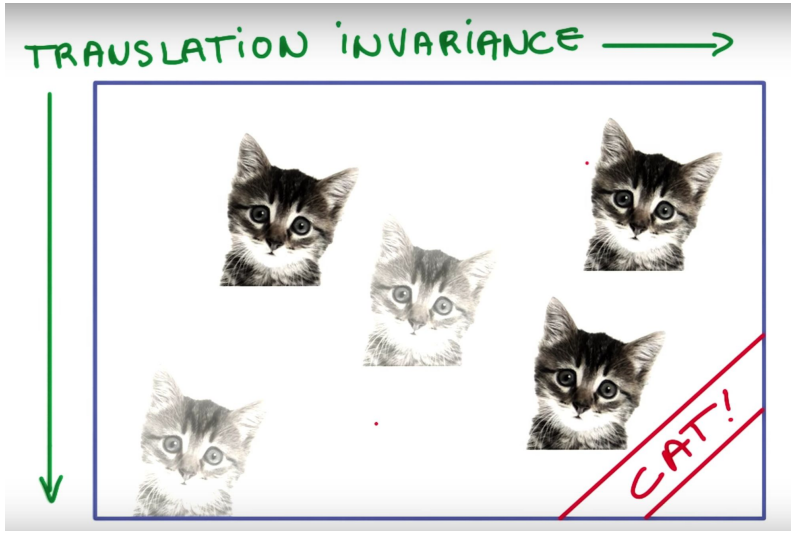
Convolutional layer's advantage

Local Connectivity Parameter Sharing

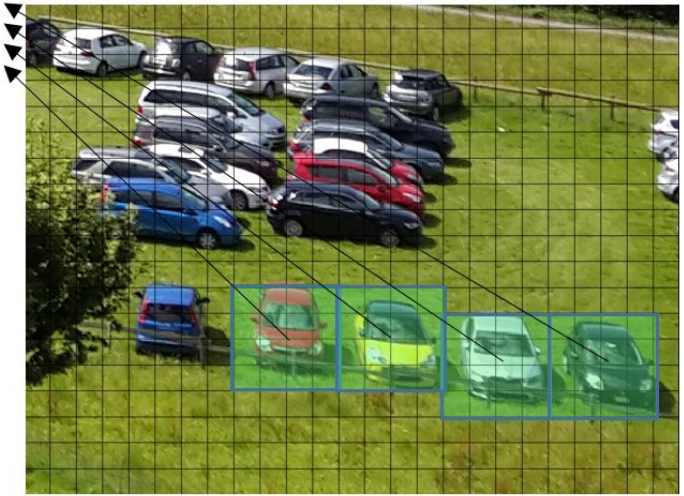




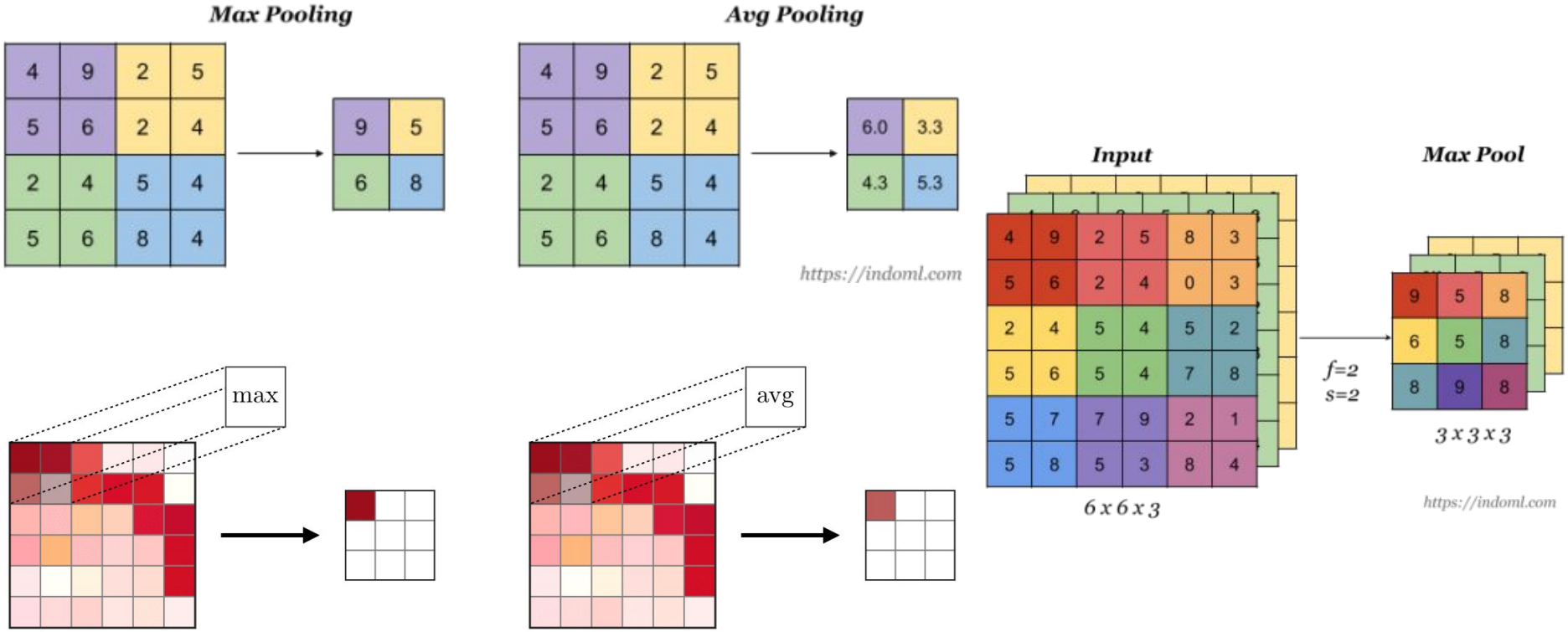
Convolutional layer's advantage



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}

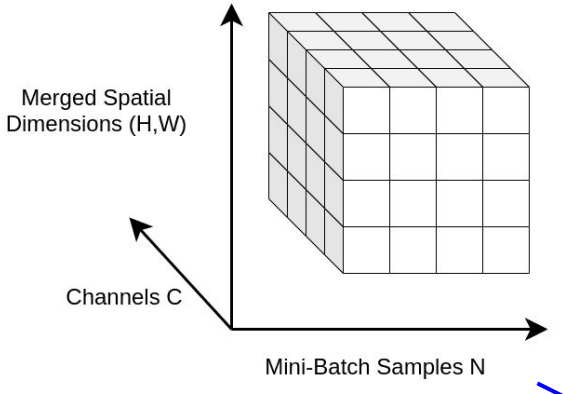


Pooling layer



Internal Covariate Shift

Batch Normalization layer

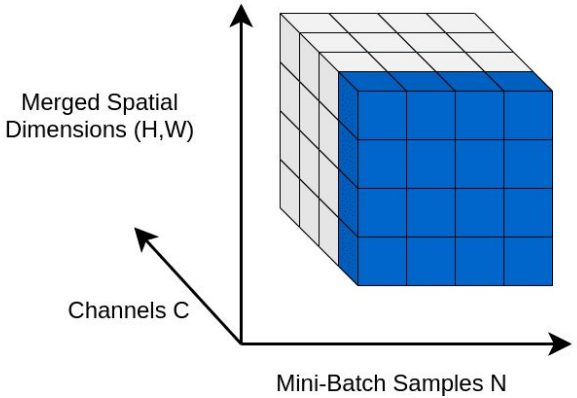


$$BN(x) = \gamma(\frac{x - \mu(x)}{\sigma(x)}) + \beta$$

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

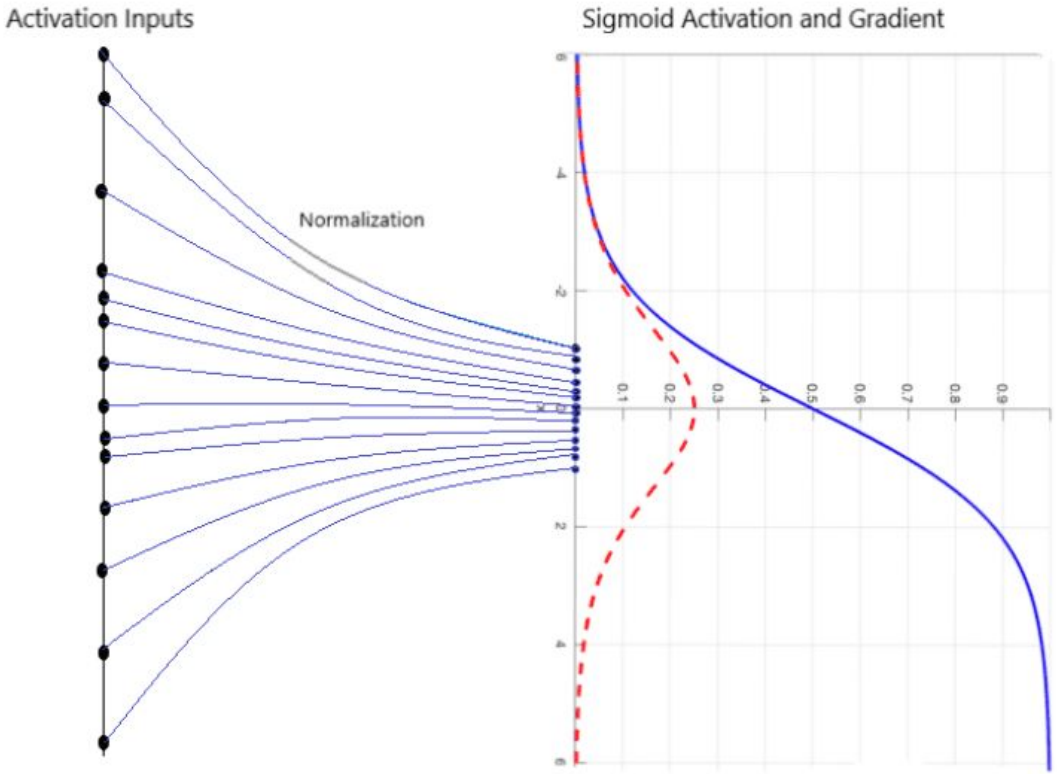
$$\sigma_c(x) = \sqrt{\frac{1}{NHW}\sum_{n=1}^{N}\sum_{h=1}^{H}\sum_{w=1}^{W}(x_{nchw} - \mu_c(x))^2}$$

Batch Norm

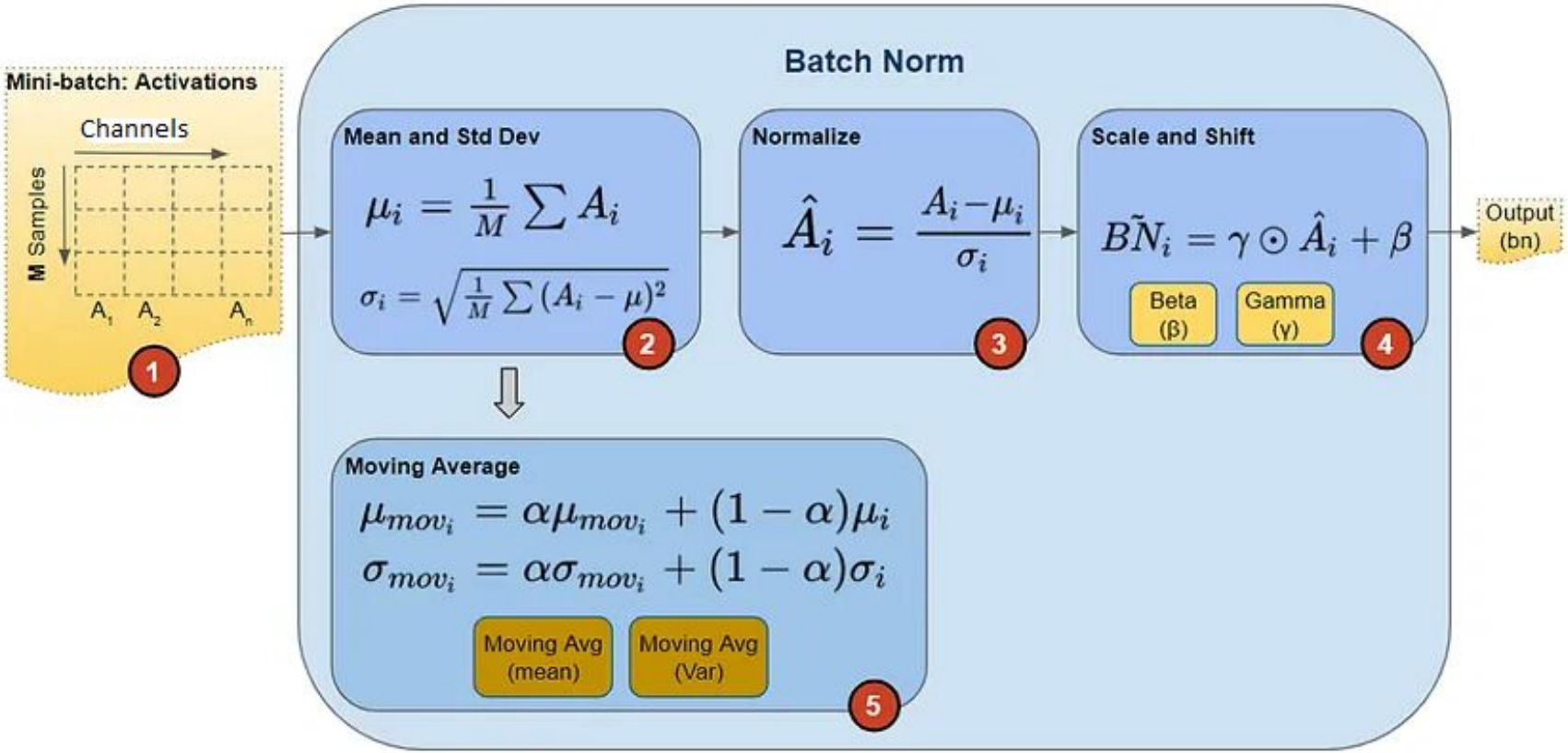


[Batch Normalization]

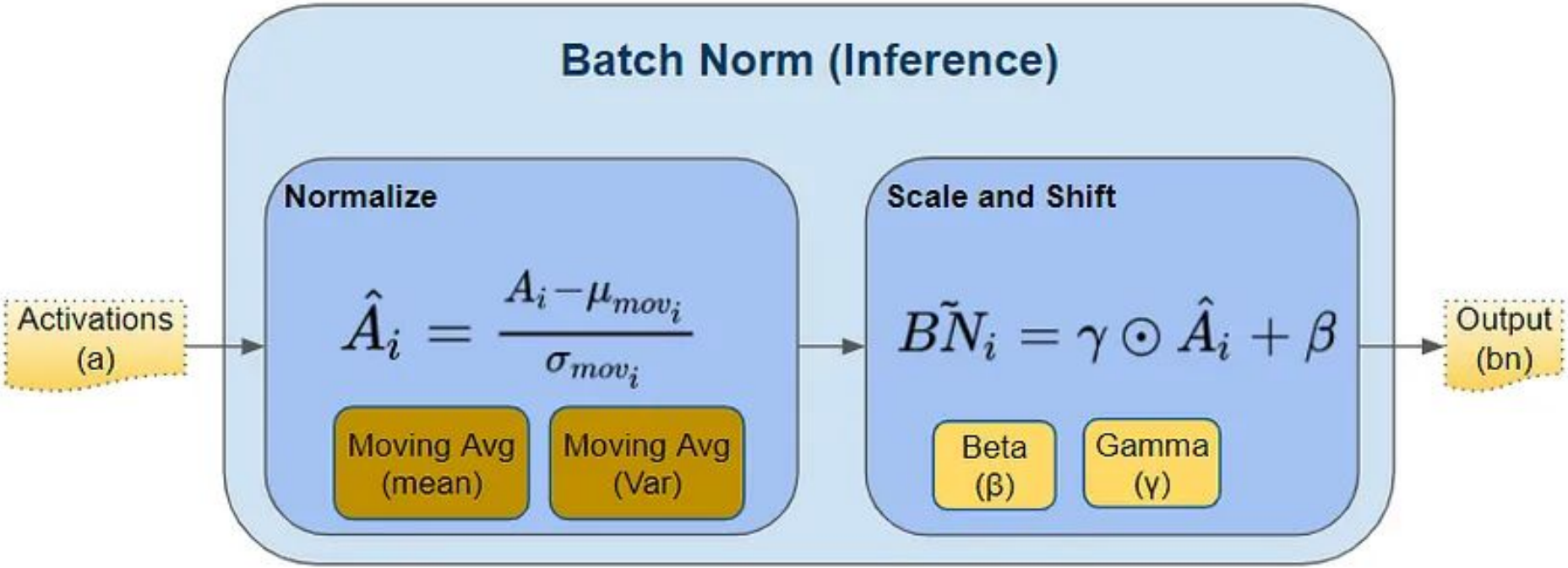
Why Normalization helps?



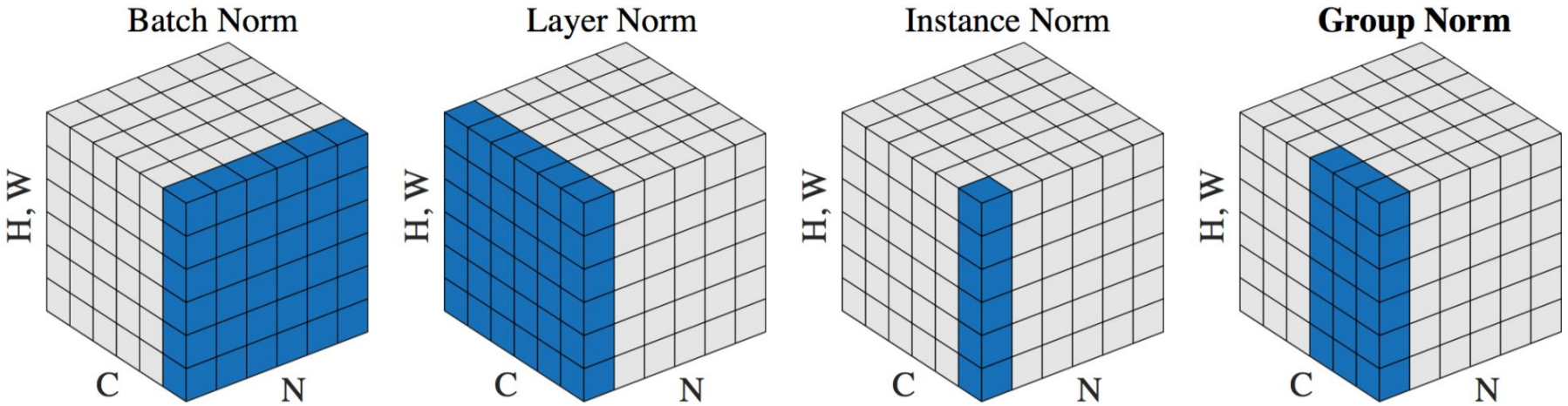
Batch Normalization: Training phase



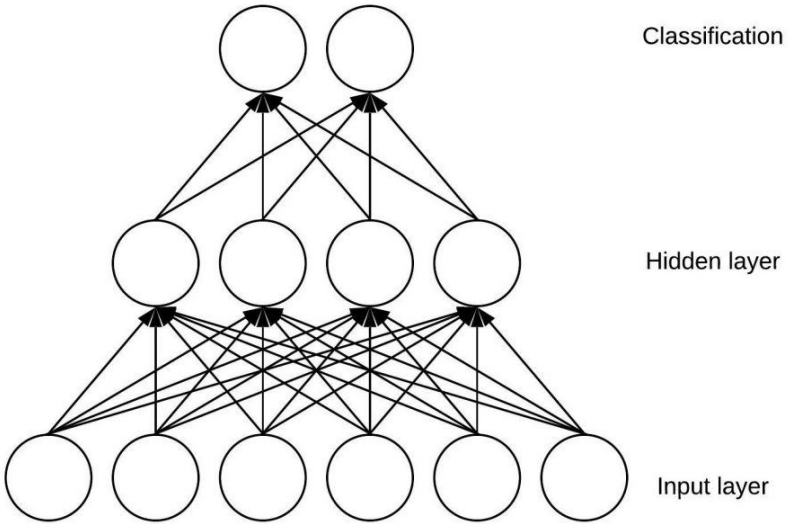
Batch Normalization: Inference phase



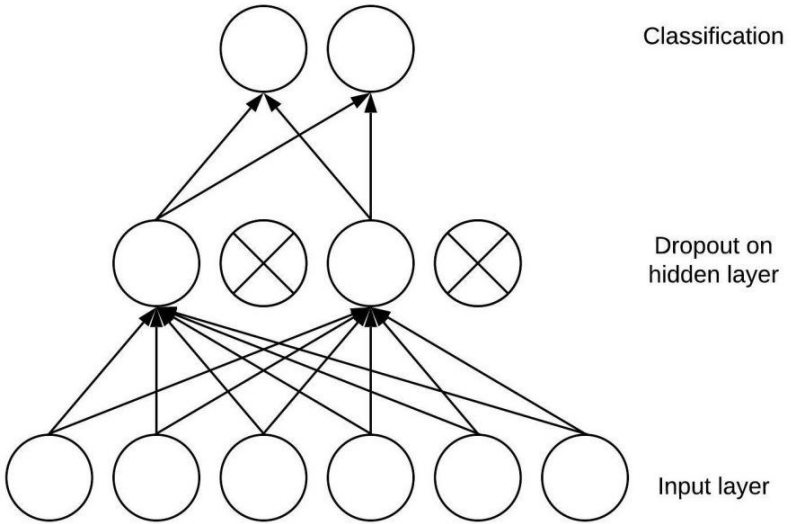
Normalization layer



Dropout layer



Without Dropout

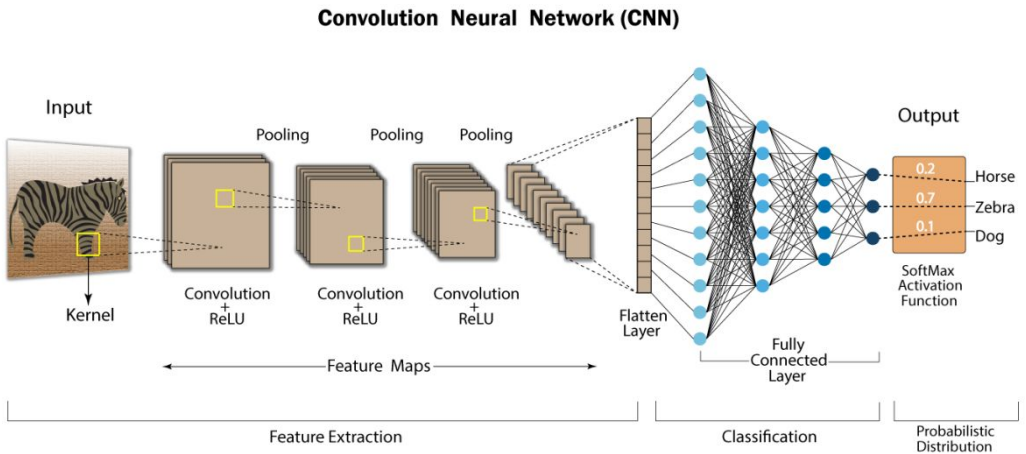


With Dropout

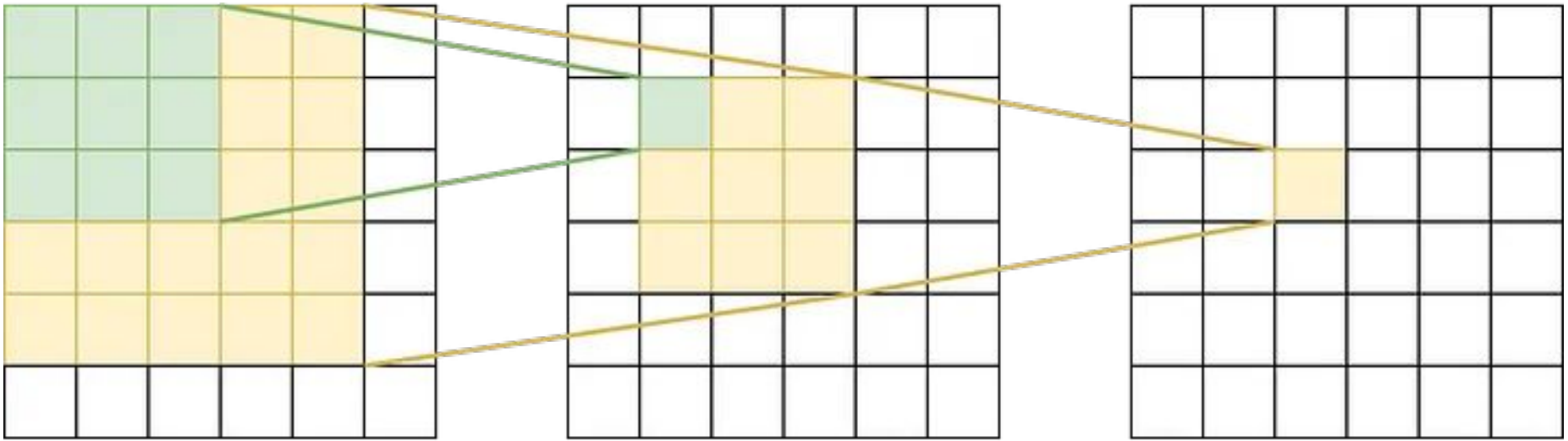
Common CNN's layer pattern (part 1)

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

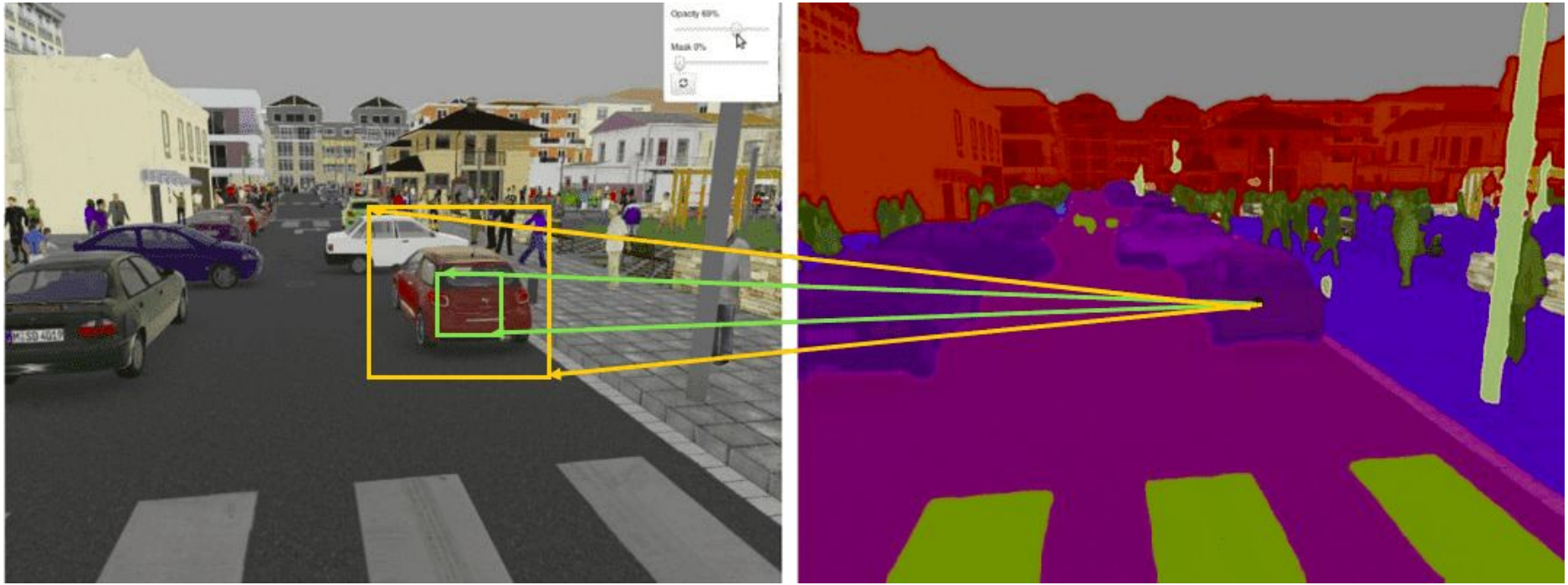
1. **Một vài block của Conv->(BatchNorm)->ReLU**
2. **Max-pooling**
3. **Lặp lại bước 1 và 2 cho đến khi feature map đủ bé**
4. **Flatten (làm phẳng) feature map**
5. **Một (vài) fully connected layer**



Receptive Field



Receptive Field



Receptive Field

How to increase Receptive field

Add more Conv layers

Make the network deeper

Add pooling layers

sub-sampling

Use Dilated Conv

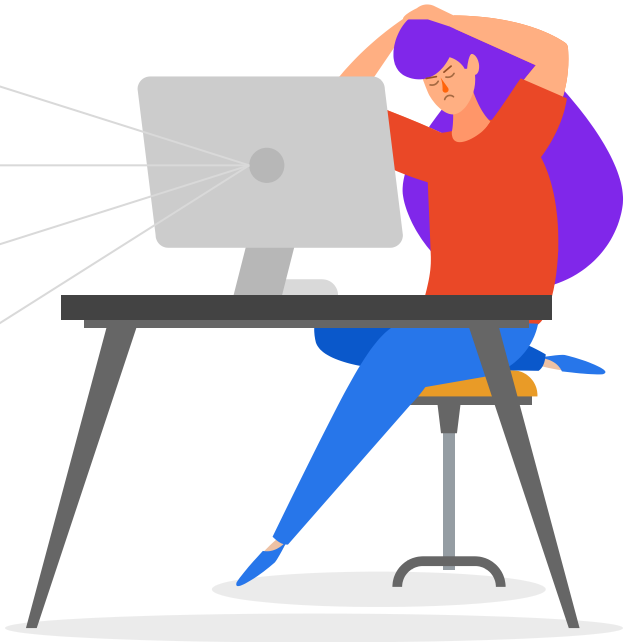
Use Depth-wise Conv

01

02

03

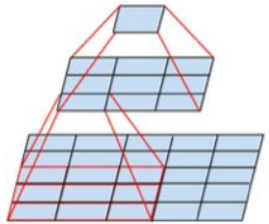
04



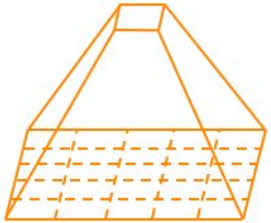
Common CNN's layer pattern (part 2)

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*3*3	5*5
Number of non-linear layers	2	1

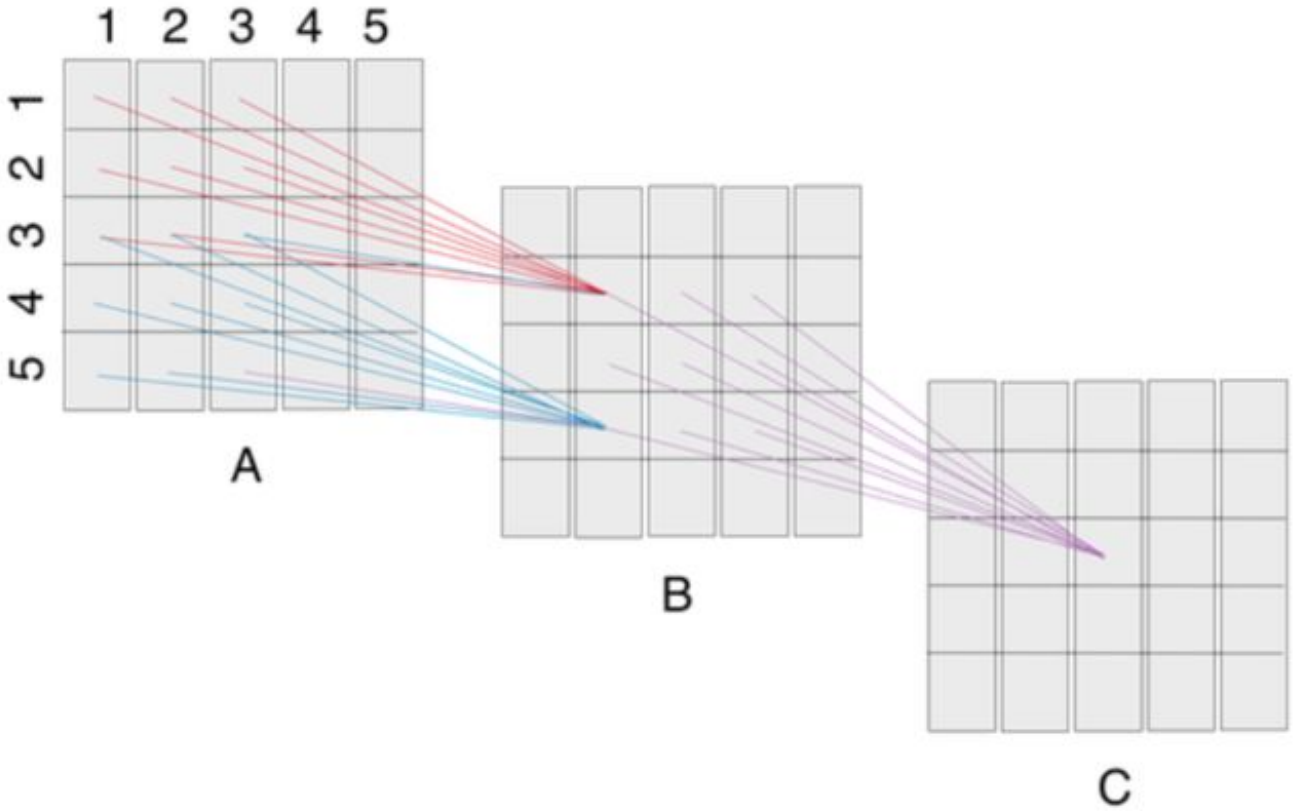


two successive
3x3 convolutions

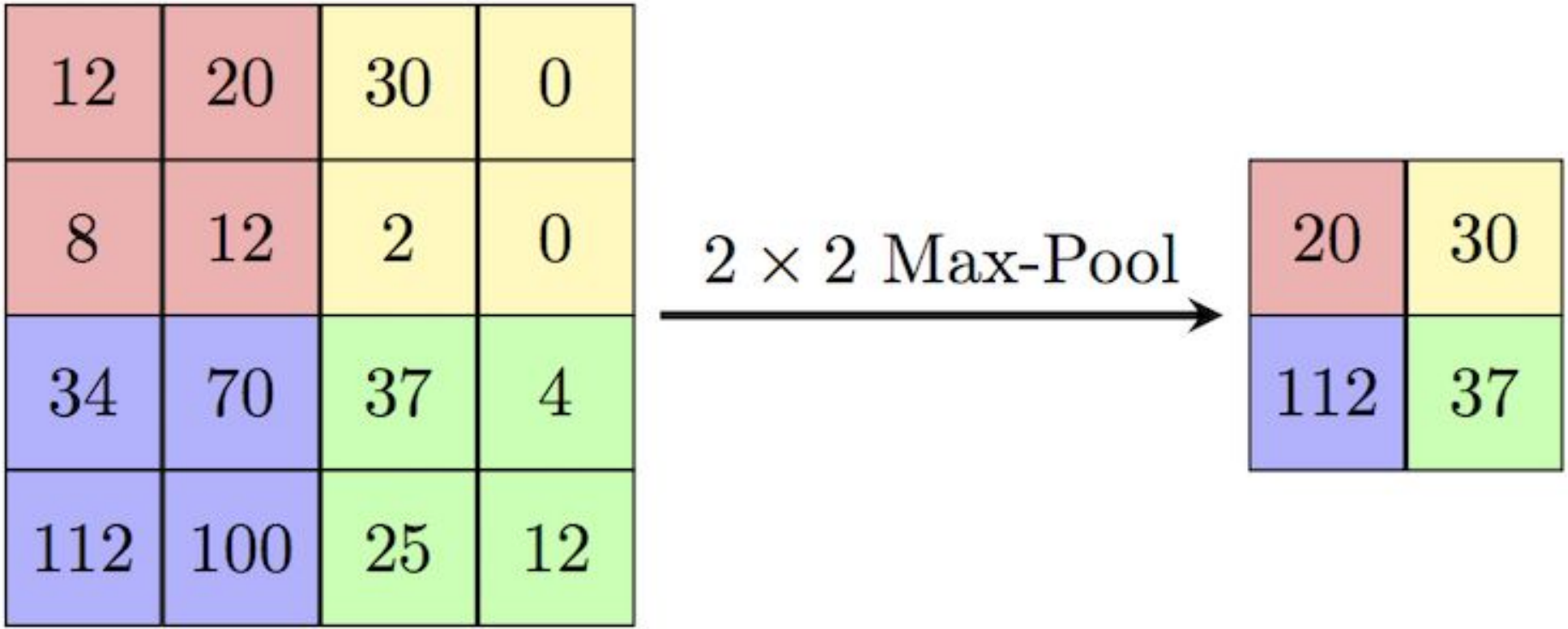


5x5 convolution

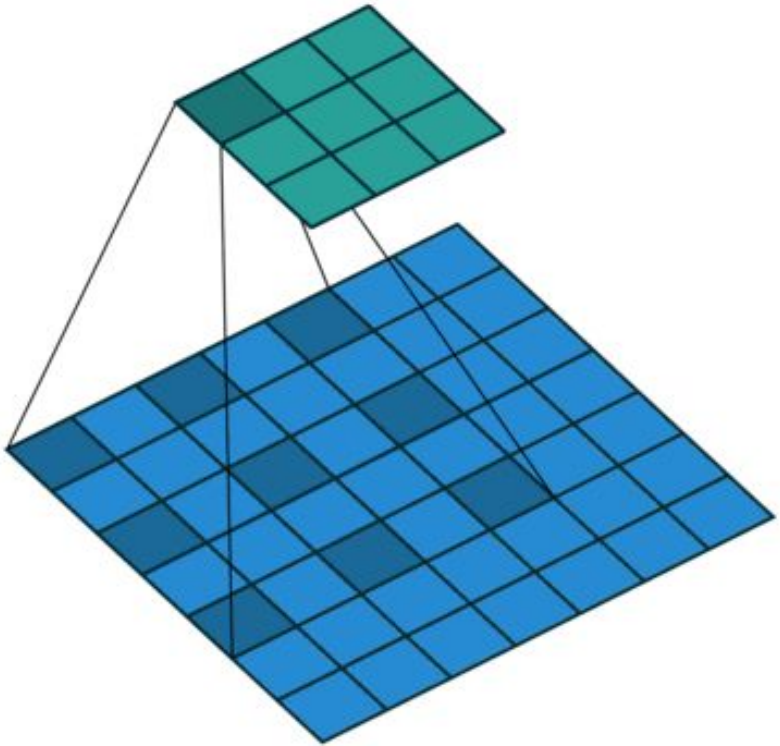
Effective Receptive Field



Use Pooling layer



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)
```

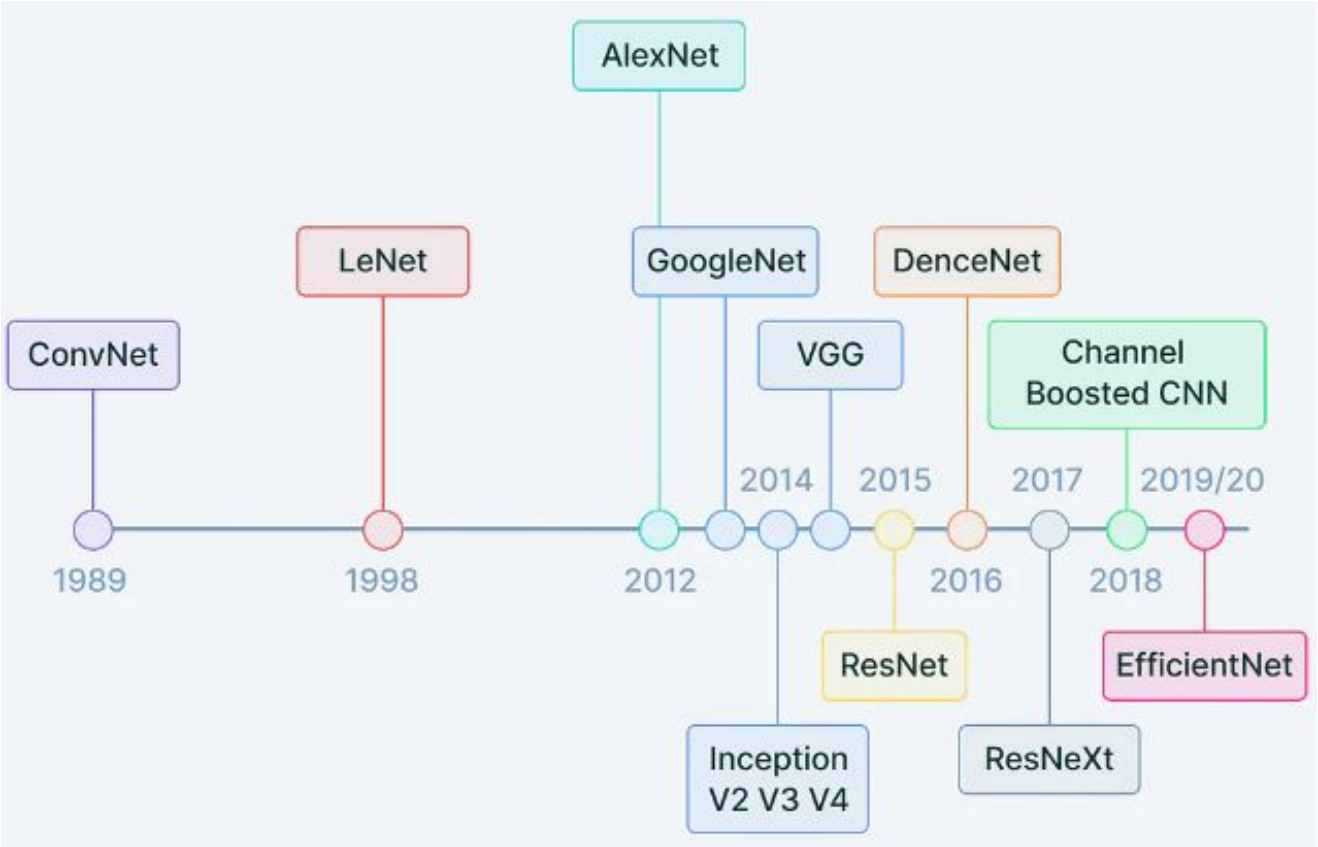
Common CNN's layer pattern (part 3)

Làm sao để thiết kế CNN architecture?

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

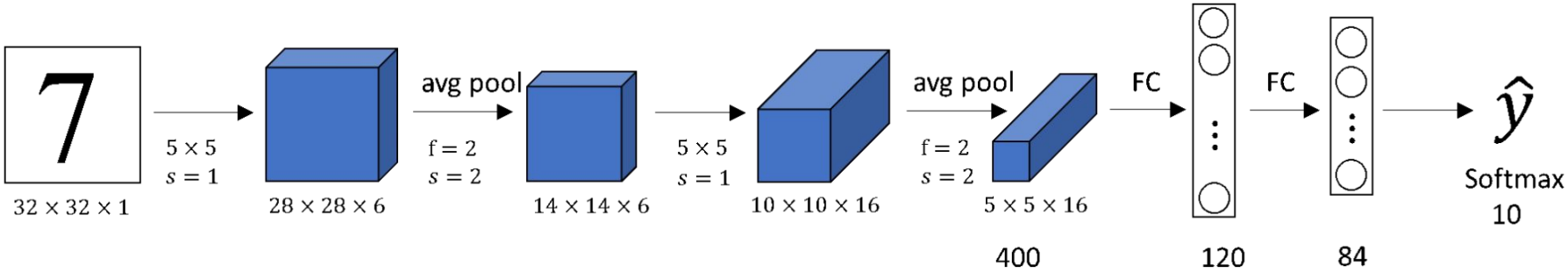
- **Mô hình nên có bao nhiêu layer?**
- **Mỗi layer nên có parameter như thế nào?**
  - **Convolutional có bao nhiêu channel? Kernel size là bao nhiêu?**
  - **Nên dùng activation function gì?**
  - **Pooling nên có kernel size là bao nhiêu?**
  - **Nên flatten feature map ở đâu?**
  - **Nên có bao nhiêu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

Common CNN architectures



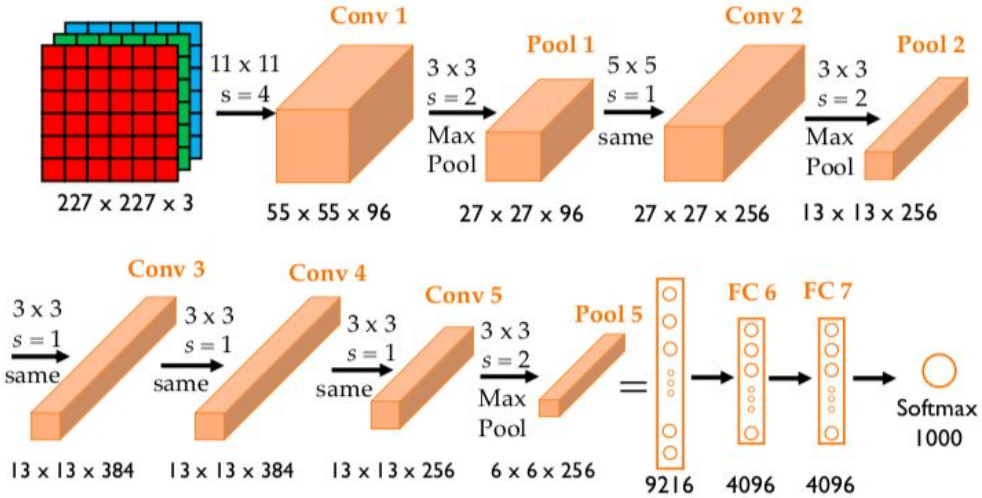
ImageNet Large Scale Visual Recognition Challenge

LeNet (1998)



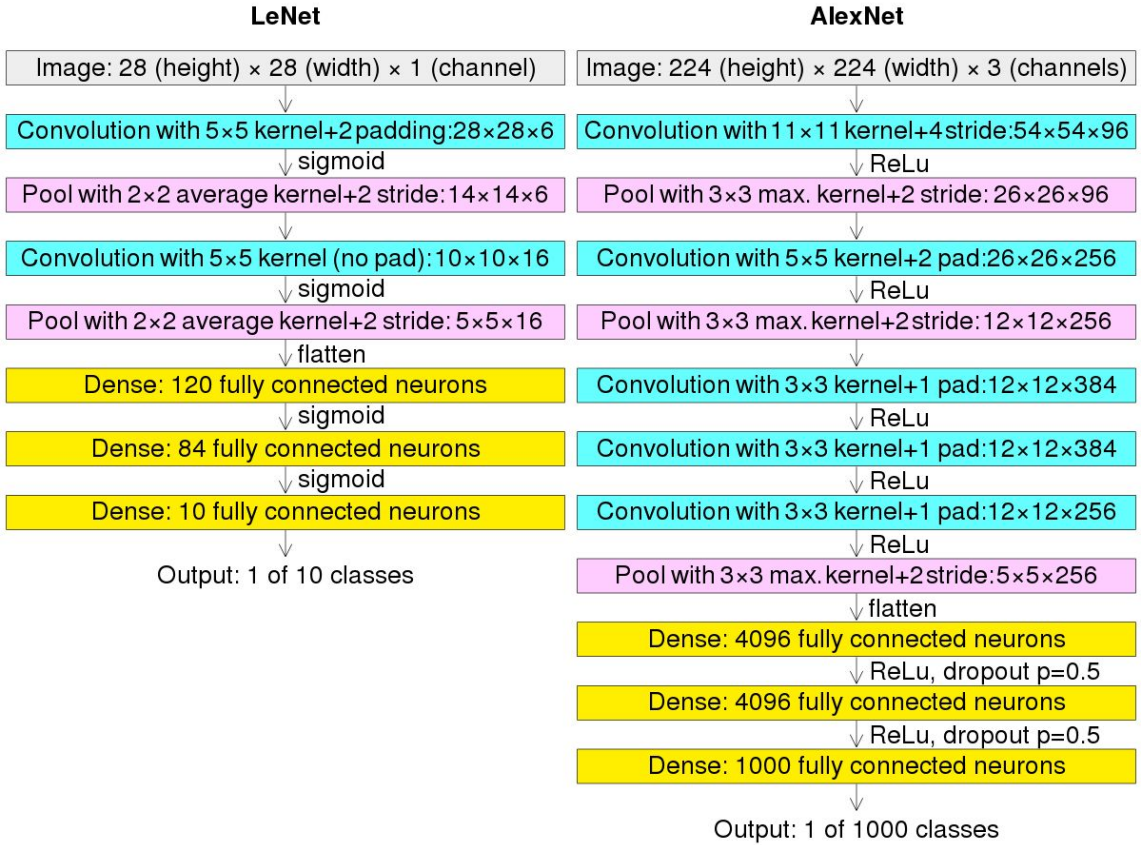
Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

AlexNet (2012)

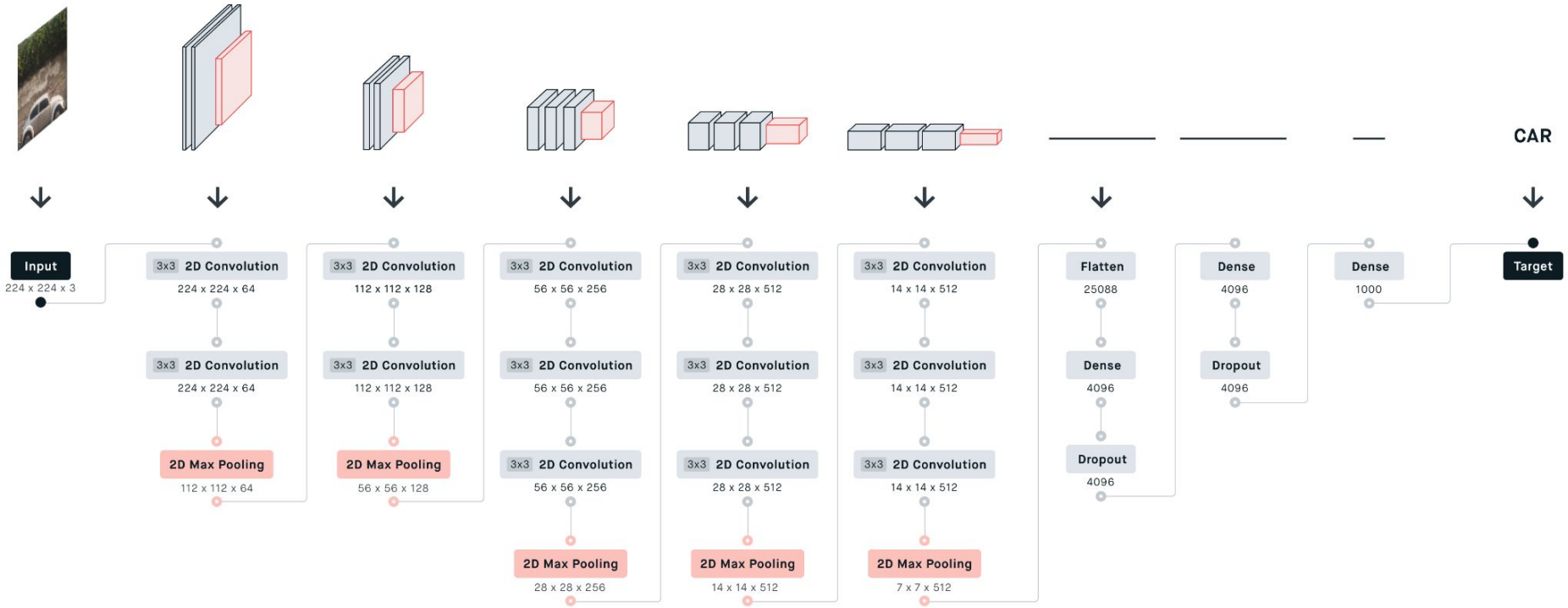


Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	227x227x3	-	-	-
1	Convolution	96	55 x 55 x 96	11x11	4	relu
	Max Pooling	96	27 x 27 x 96	3x3	2	relu
2	Convolution	256	27 x 27 x 256	5x5	1	relu
	Max Pooling	256	13 x 13 x 256	3x3	2	relu
3	Convolution	384	13 x 13 x 384	3x3	1	relu
4	Convolution	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

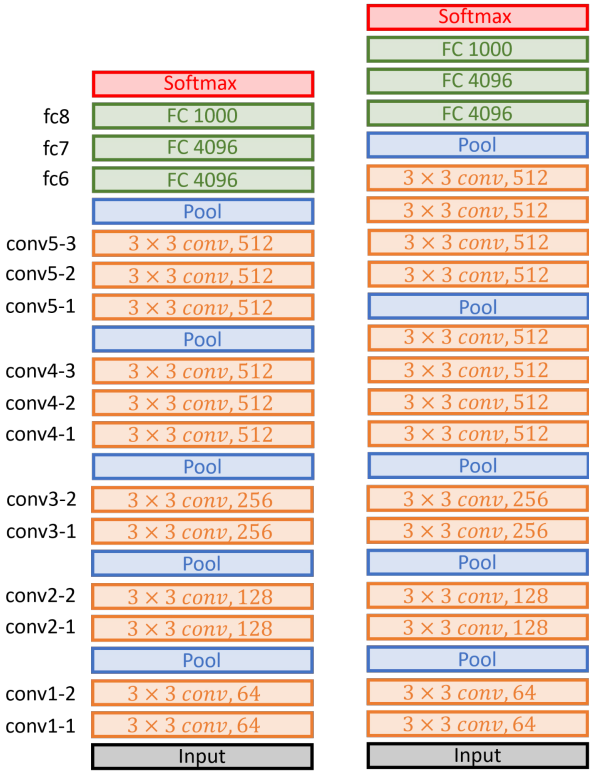
LeNet vs AlexNet

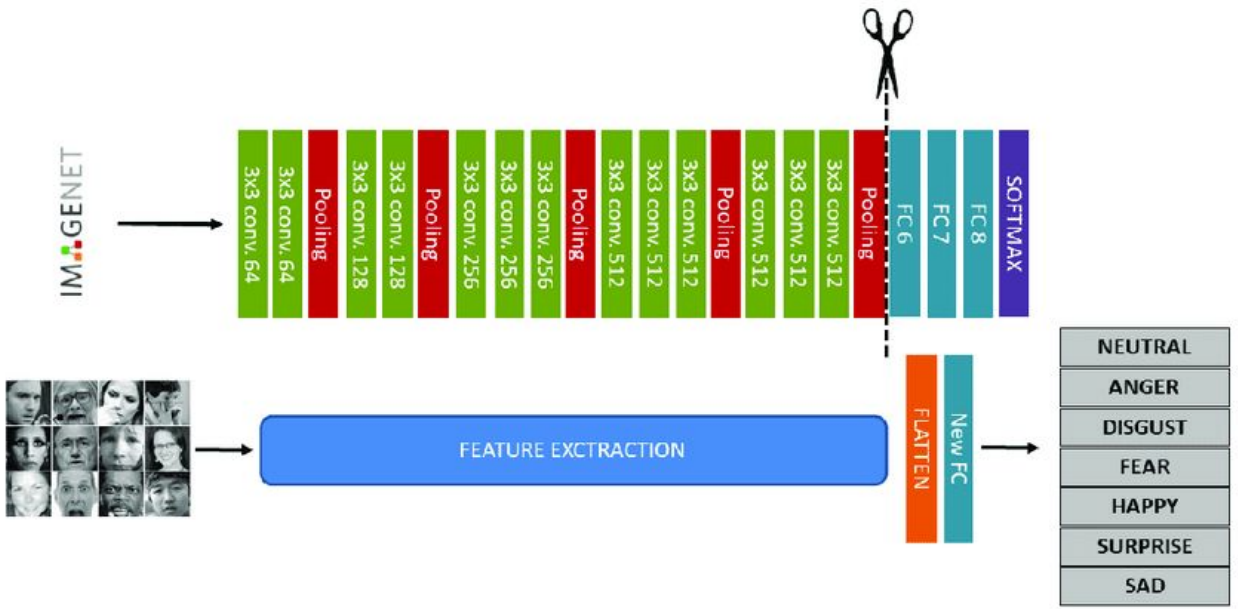


VGGNet (2014)

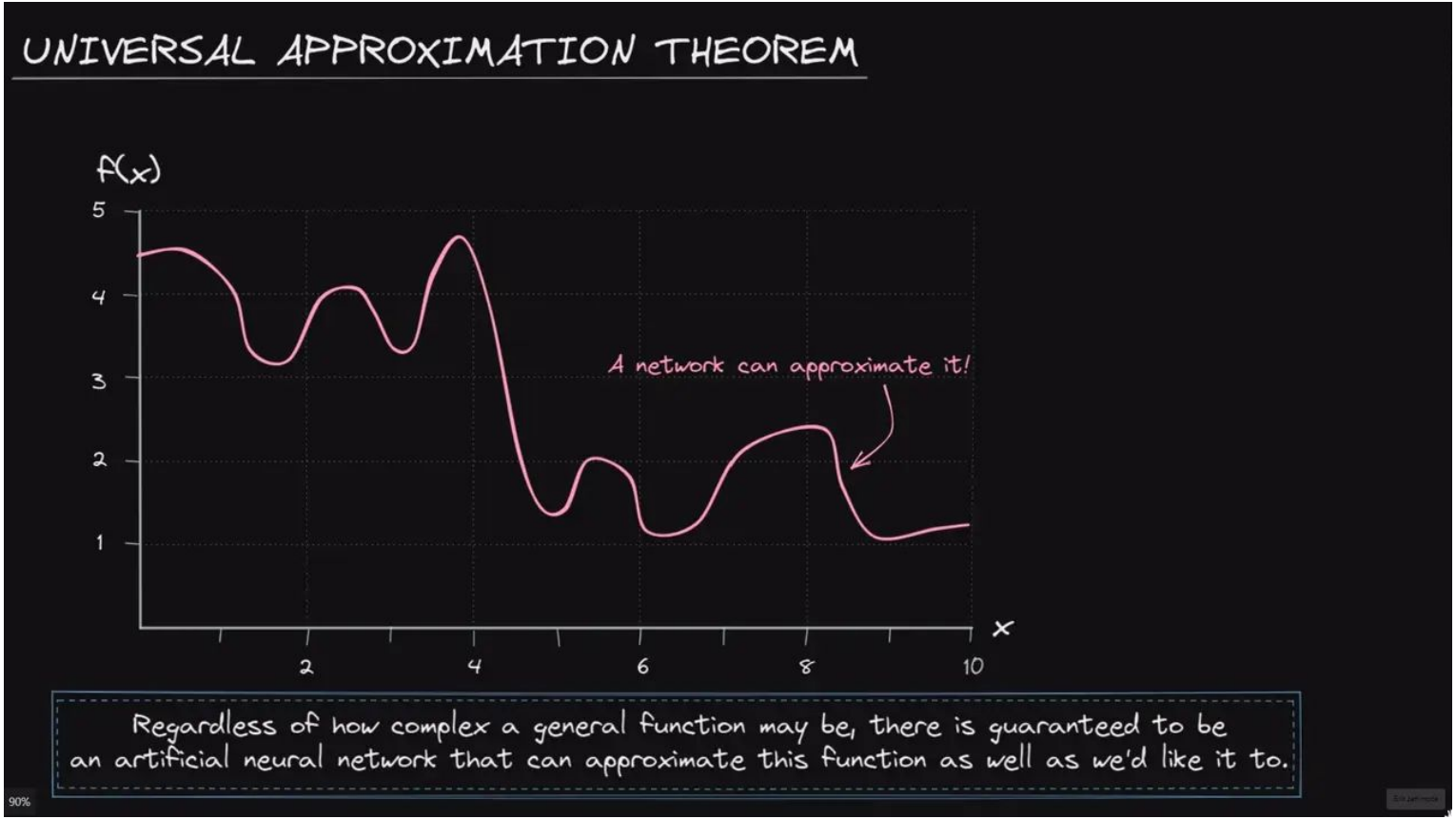

Input
224 x 224 x 3
3x3 2D Convolution
224 x 224 x 64
3x3 2D Convolution
224 x 224 x 64
2D Max Pooling
112 x 112 x 64
3x3 2D Convolution
112 x 112 x 128
3x3 2D Convolution
112 x 112 x 128
2D Max Pooling
56 x 56 x 128
3x3 2D Convolution
56 x 56 x 256
3x3 2D Convolution
56 x 56 x 256
3x3 2D Convolution
56 x 56 x 256
2D Max Pooling
28 x 28 x 256
3x3 2D Convolution
28 x 28 x 512
3x3 2D Convolution
28 x 28 x 512
3x3 2D Convolution
28 x 28 x 512
2D Max Pooling
14 x 14 x 512
3x3 2D Convolution
14 x 14 x 512
3x3 2D Convolution
14 x 14 x 512
3x3 2D Convolution
14 x 14 x 512
2D Max Pooling
7 x 7 x 512
Flatten
25088
Dense
4096
Dropout
4096
Dense
4096
Dropout
4096
Dense
1000
CAR
Target

VGGNet (2014)

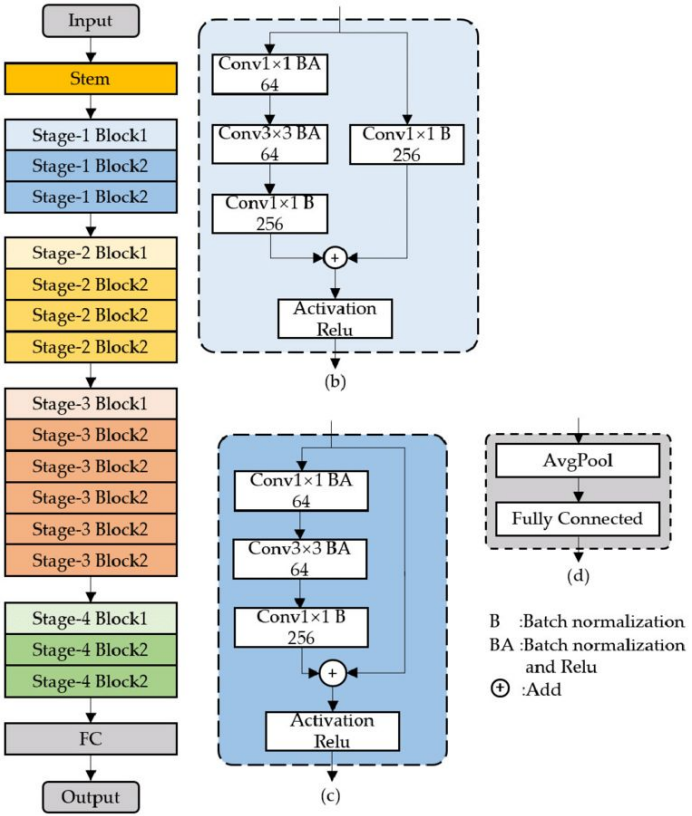

Softmax
fc8
FC 1000
fc7
FC 4096
fc6
FC 4096
Pool
conv5-3
3 × 3 conv, 512
conv5-2
3 × 3 conv, 512
conv5-1
3 × 3 conv, 512
Pool
conv4-3
3 × 3 conv, 512
conv4-2
3 × 3 conv, 512
conv4-1
3 × 3 conv, 512
Pool
conv3-2
3 × 3 conv, 256
conv3-1
3 × 3 conv, 256
Pool
conv2-2
3 × 3 conv, 128
conv2-1
3 × 3 conv, 128
Pool
conv1-2
3 × 3 conv, 64
conv1-1
3 × 3 conv, 64
Input
VGG16
Softmax
FC 1000
FC 4096
FC 4096
Pool
3 × 3 conv, 512
3 × 3 conv, 512
3 × 3 conv, 512
3 × 3 conv, 512
Pool
3 × 3 conv, 512
3 × 3 conv, 512
3 × 3 conv, 512
3 × 3 conv, 512
Pool
3 × 3 conv, 256
3 × 3 conv, 256
Pool
3 × 3 conv, 128
3 × 3 conv, 128
Pool
3 × 3 conv, 64
3 × 3 conv, 64
Input
VGG19


IMAGENET
3x3 conv. 64
3x3 conv. 64
Pooling
3x3 conv. 128
3x3 conv. 128
Pooling
3x3 conv. 256
3x3 conv. 256
3x3 conv. 256
Pooling
3x3 conv. 512
3x3 conv. 512
3x3 conv. 512
Pooling
3x3 conv. 512
3x3 conv. 512
3x3 conv. 512
Pooling
FC 6
FC 7
FC 8
SOFTMAX
FEATURE EXCTRACTION
FLATTEN
New FC
NEUTRAL
ANGER
DISGUST
FEAR
HAPPY
SURPRISE
SAD

Universal Approximation Theorem



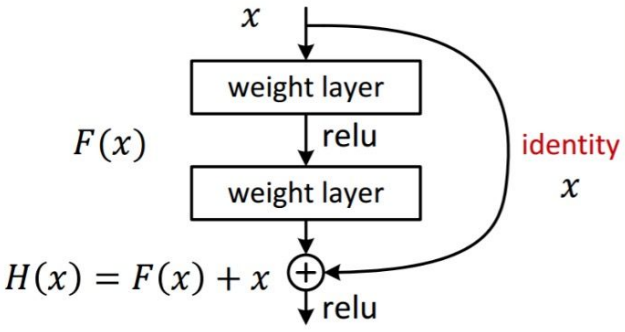
ResNet (2015)



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	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 $H(x) = F(x) + x$

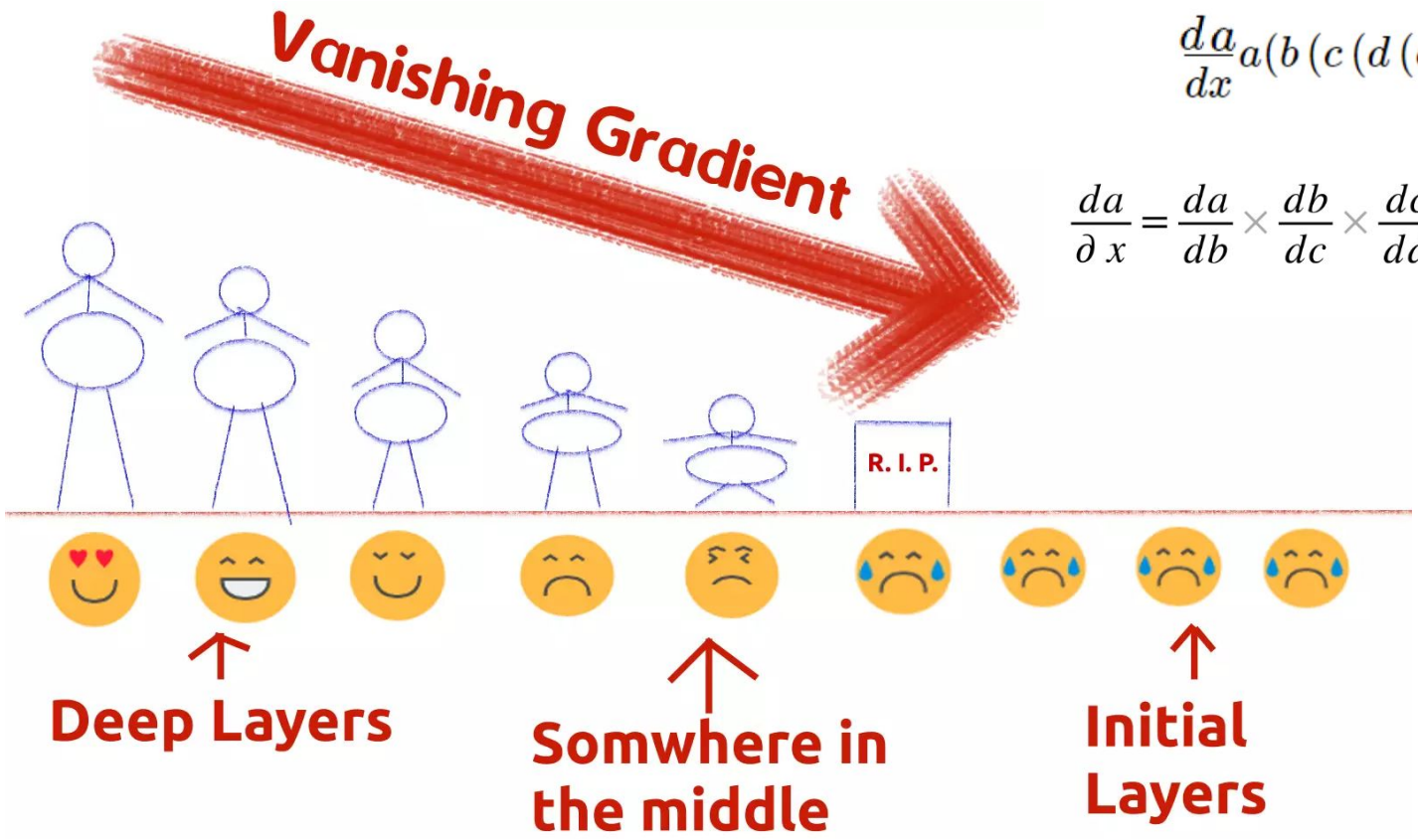
$F(x)$ is obtained from x as follows.

$$x \rightarrow weight_1 \rightarrow ReLU \rightarrow weight_2$$

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

$$F(x) + x \rightarrow ReLU$$

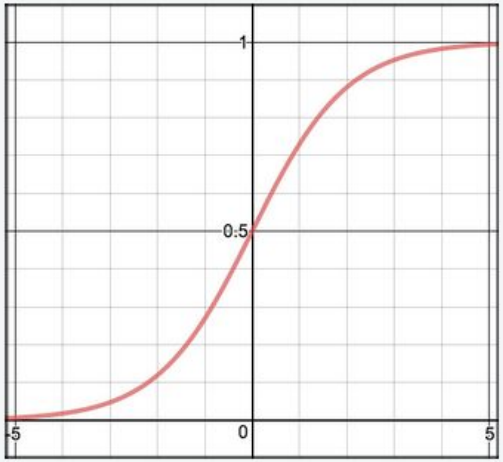
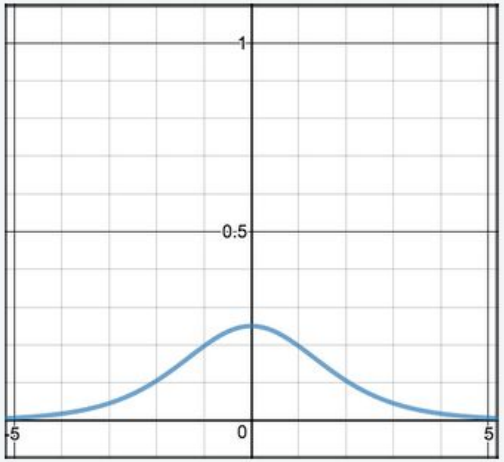
Problems in Neural Network: Vanishing gradients



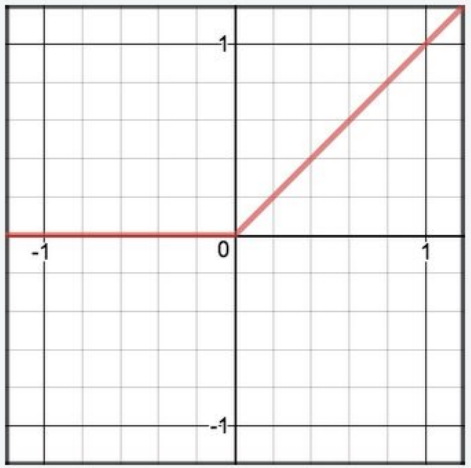
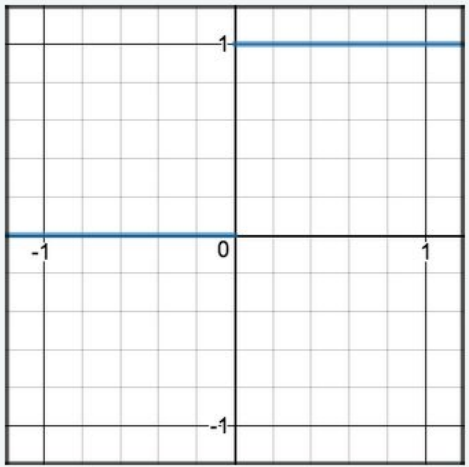
$$\frac{da}{dx} a(b(c(d(e(f(g(x)))))))$$

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

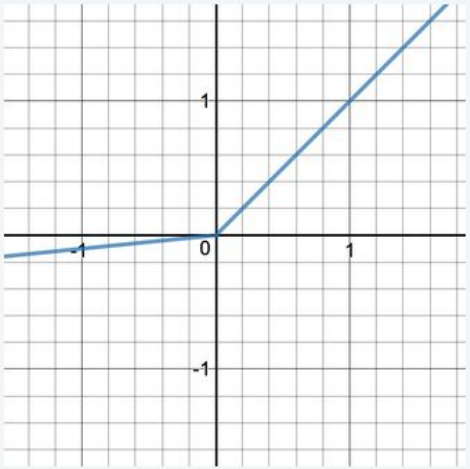
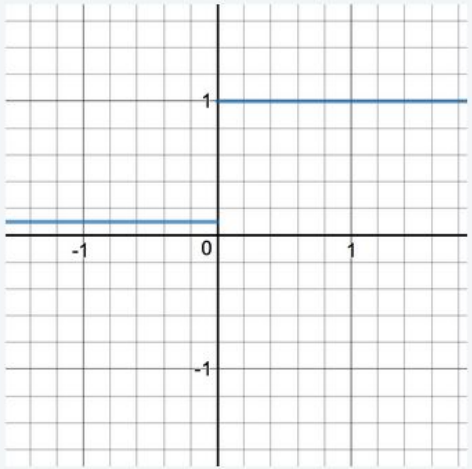
Sigmoid vs Vanishing gradients

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

ReLU vs Vanishing gradients

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

Leaky ReLU vs Vanishing gradients

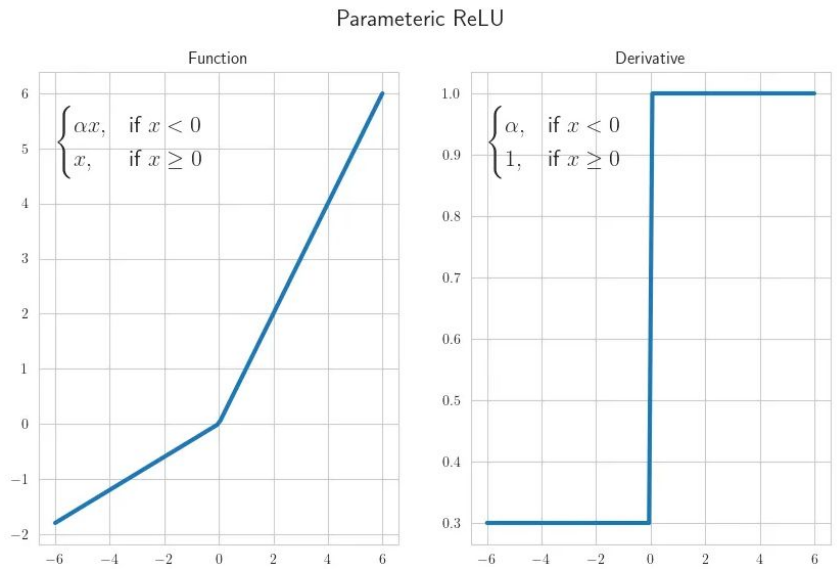
Function	Derivative
$R(z) = \begin{Bmatrix} z & z > 0 \\ \alpha z & z <= 0 \end{Bmatrix}$	$R'(z) = \begin{Bmatrix} 1 & z > 0 \\ \alpha & z < 0 \end{Bmatrix}$
	

```
def leakyrelu(z, alpha):
        return max(alpha * z, z)
```

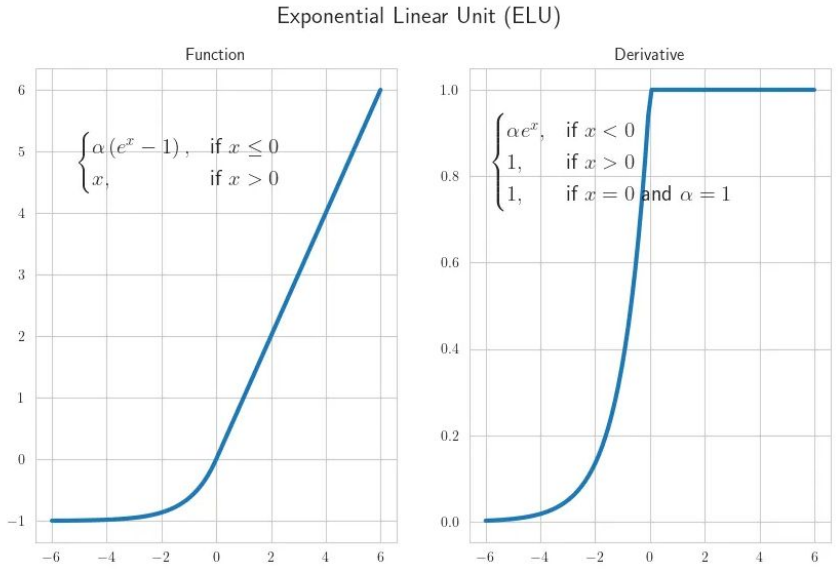
```
def leakyrelu_prime(z, alpha):
        return 1 if z > 0 else alpha
```

Các biến thể khác của ReLU

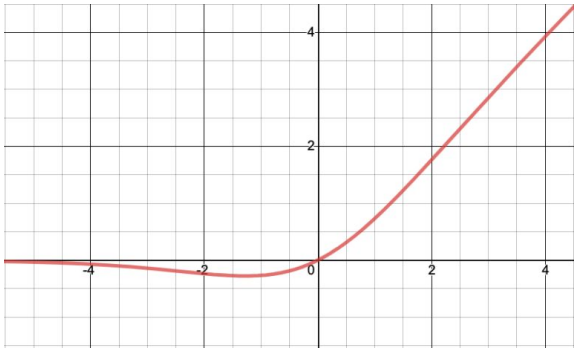
Parametric ReLU

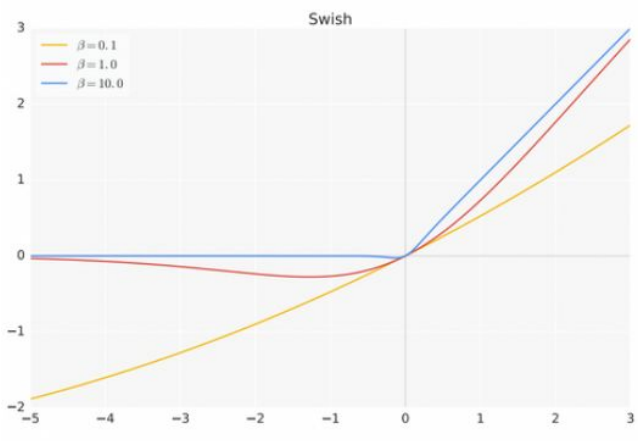


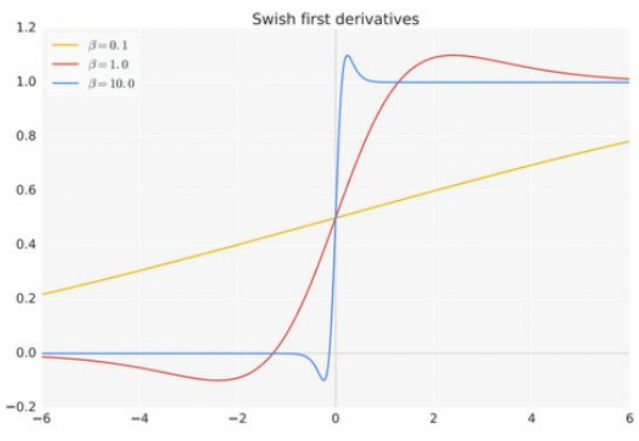
Exponential Linear Unit (ELU)



SWISH


4
2
-4
-2
0
2
4


Swish
$\beta = 0.1$
$\beta = 1.0$
$\beta = 10.0$
3
2
1
0
−1
−2
−5
−4
−3
−2
−1
0
1
2
3


Swish first derivatives
$\beta = 0.1$
$\beta = 1.0$
$\beta = 10.0$
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−6
−4
−2
0
2
4
6

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

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

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

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

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

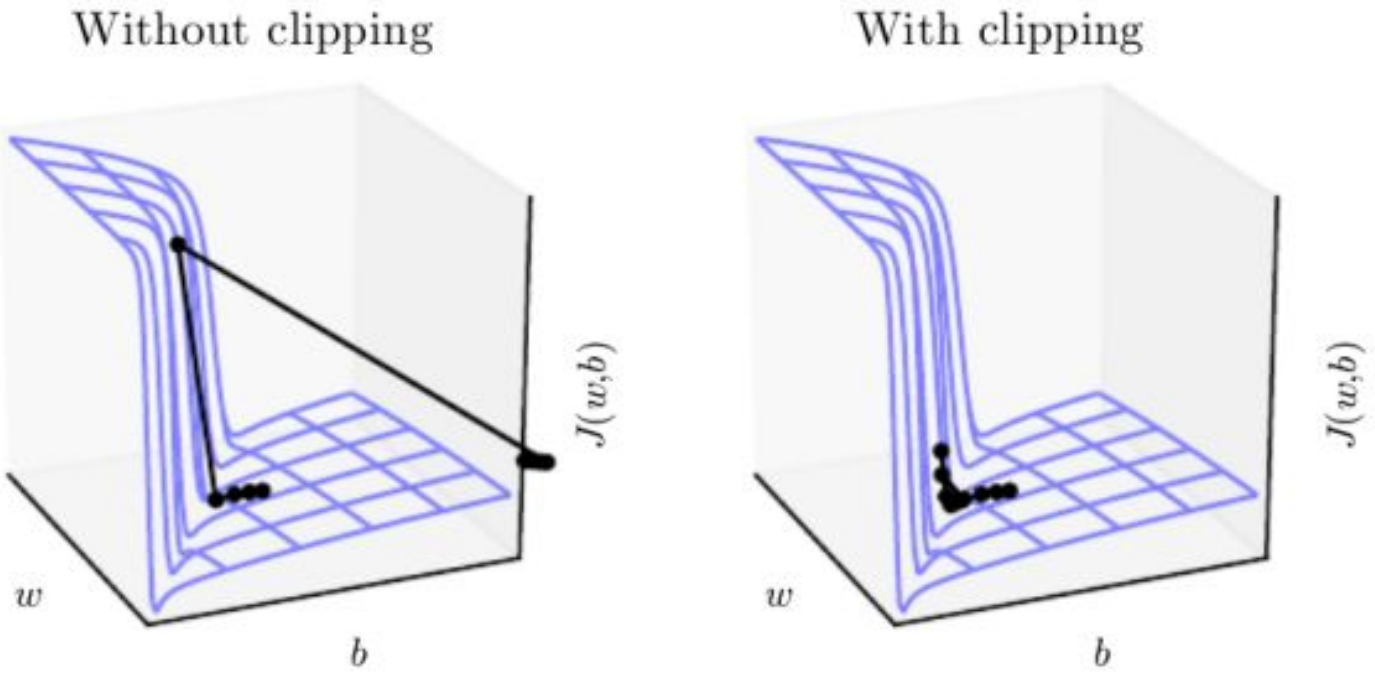
    $$W \sim N(0, Var(W))$$

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

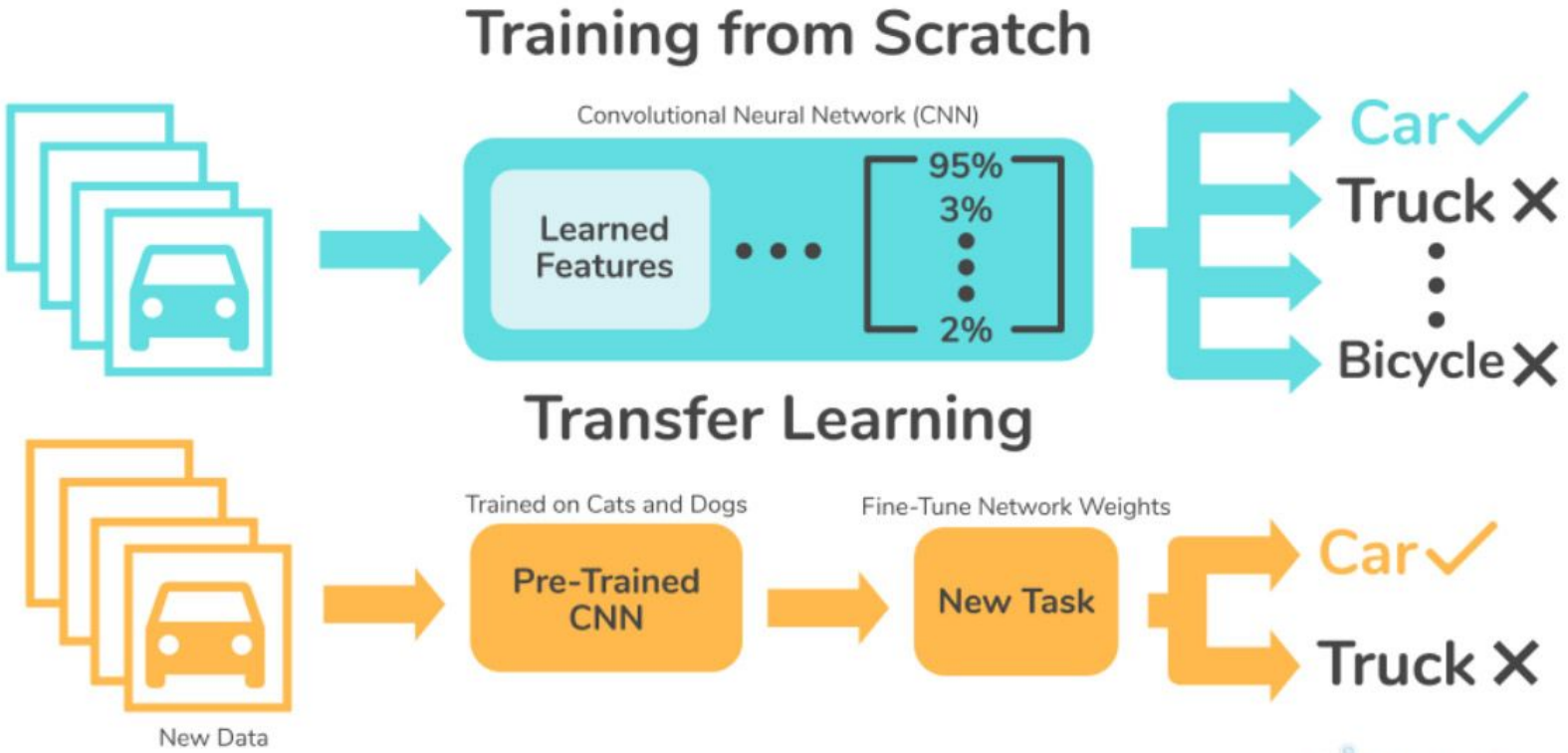
  - He Uniform initialization

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

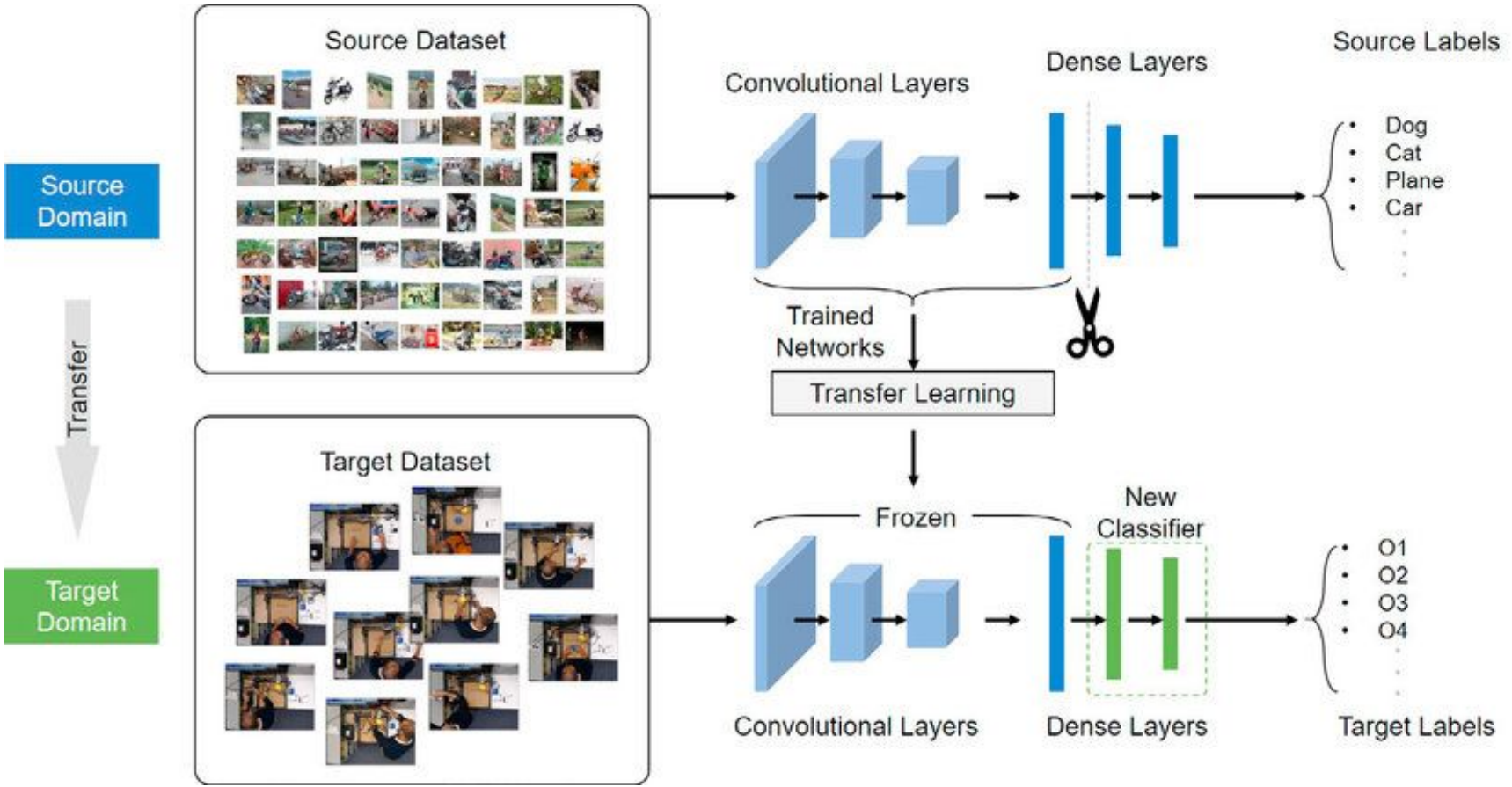
Gradient clipping



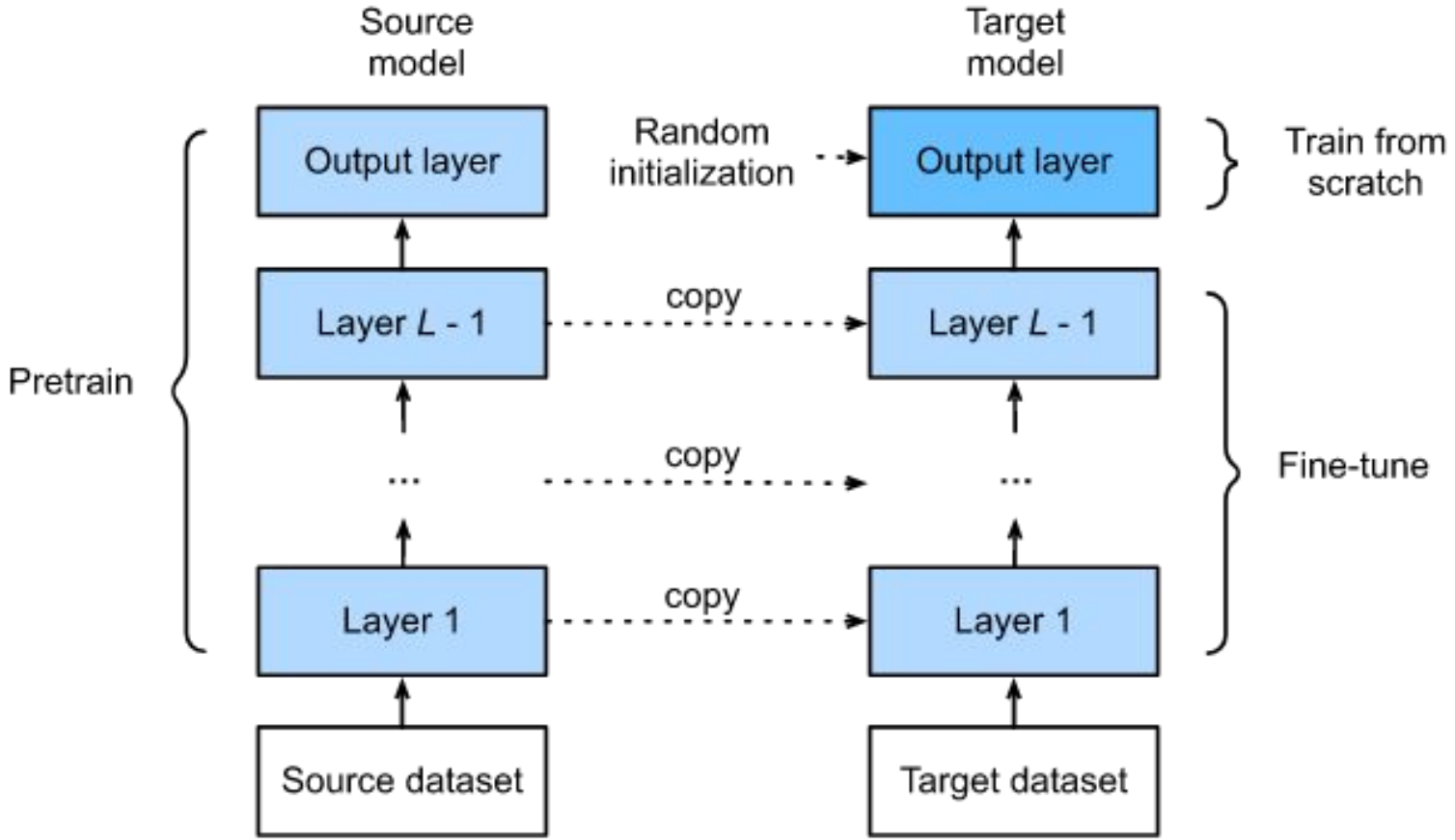
Transfer Learning



Transfer Learning: Use CNN as fixed feature extractor



Transfer Learning: Fine-tune CNN

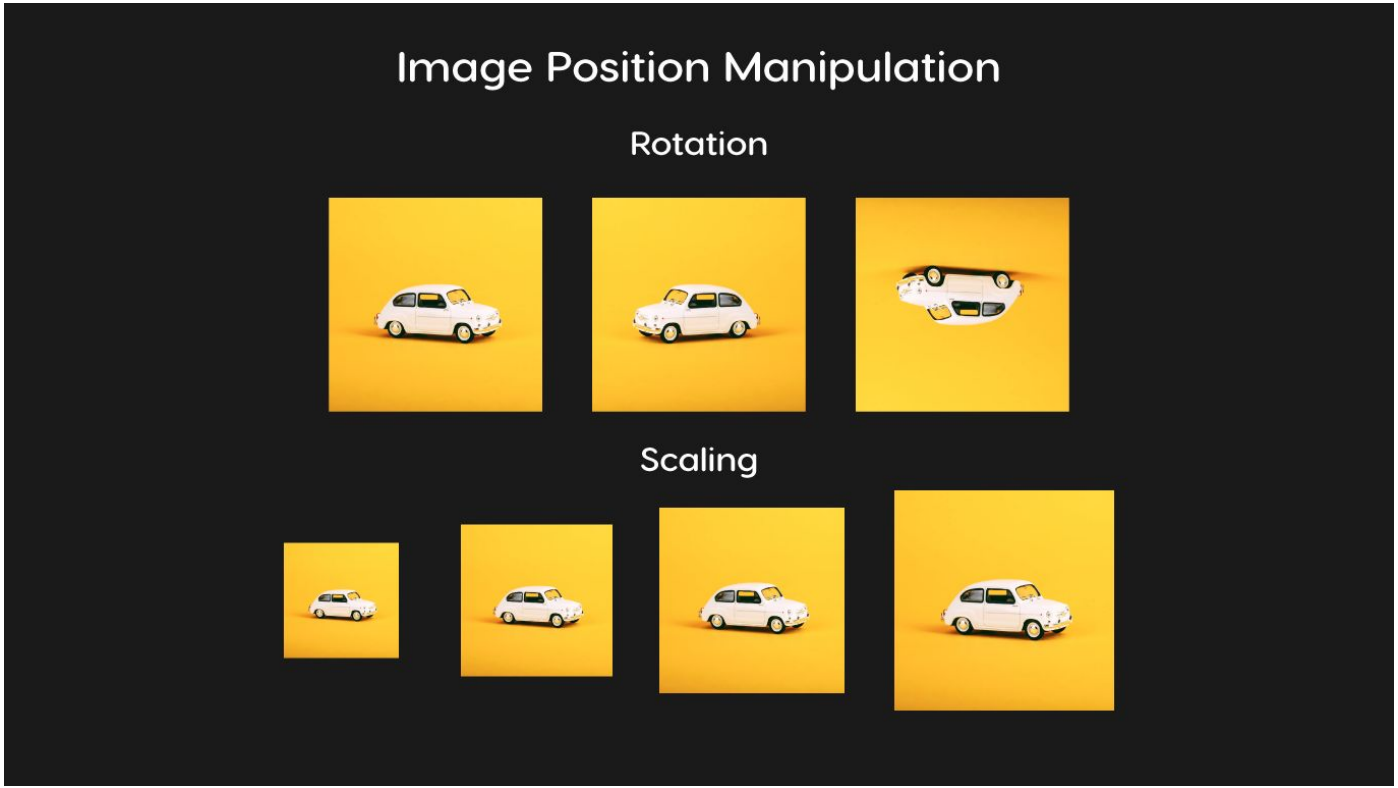


Transfer Learning: How to choose approach

	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



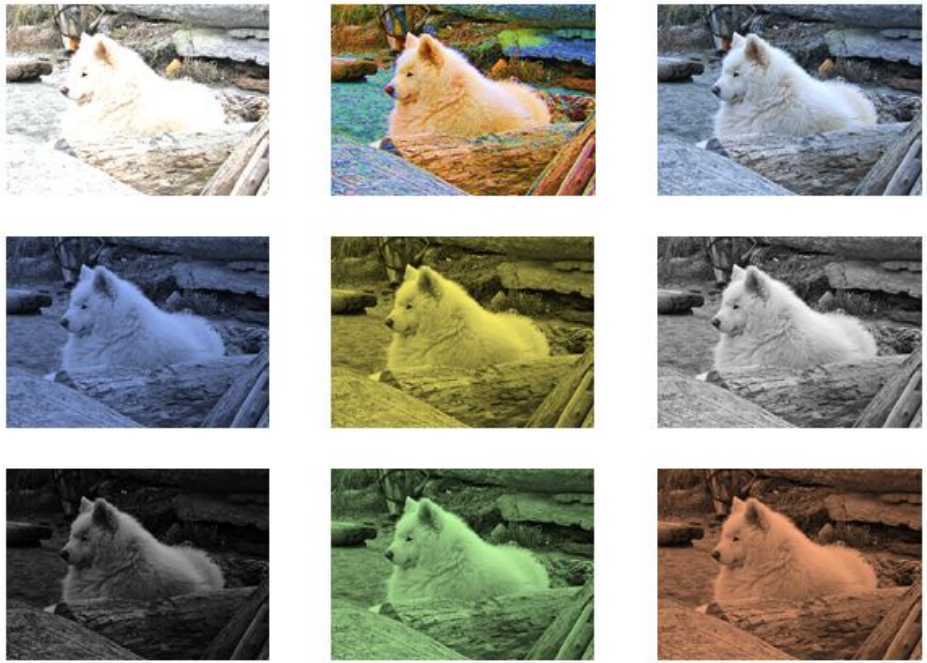
Another name: Geometric transformations

Data augmentation: Color manipulation

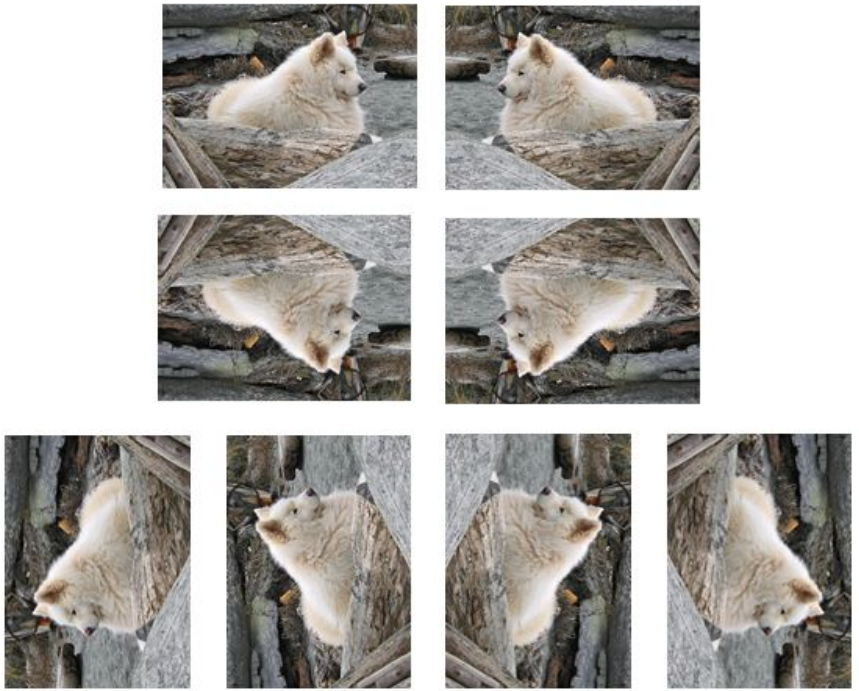
Another name: Color distortion

Data augmentation: Example

Brightness, Contrast, and Color Changes

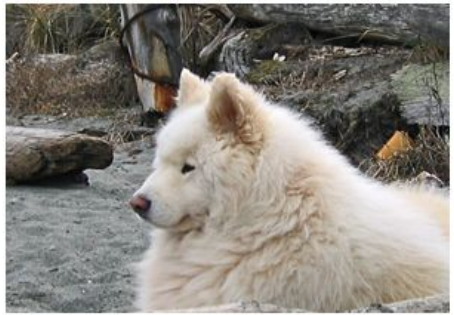


Horizontal and Vertical Flips



Data augmentation: Example

Random Crops




Deformations

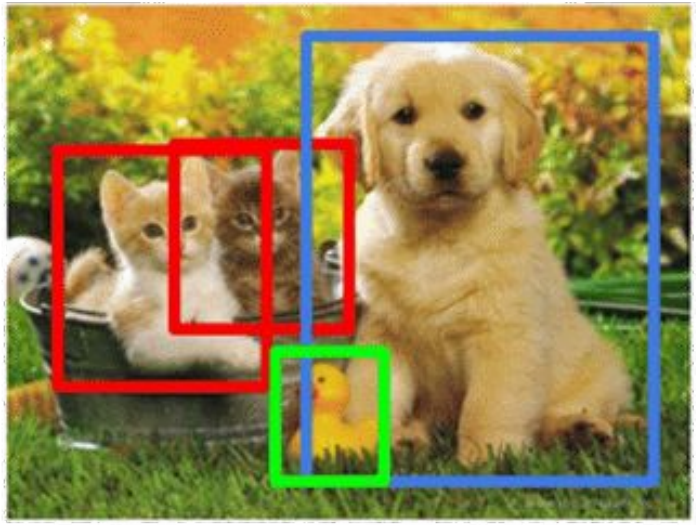
Object Detection

Image Classification vs Object Detection

Classification

CAT

Object Detection



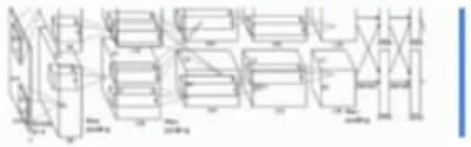
CAT, DOG, DUCK

Object Detection

Input: a single RGB image

Output: a set of detected objects

Object Detection - Single Object



What?

Correct label: Cat

Class Scores:
- Cat: 0.9
- Dog: 0.01
- Car: 0.05

FC: 4096 to 1000

Softmax loss

Where?

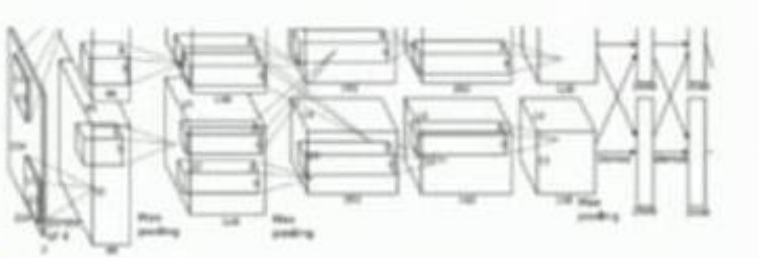
FC: 4096 to 4

Box coordinates:
- x, y, w, h

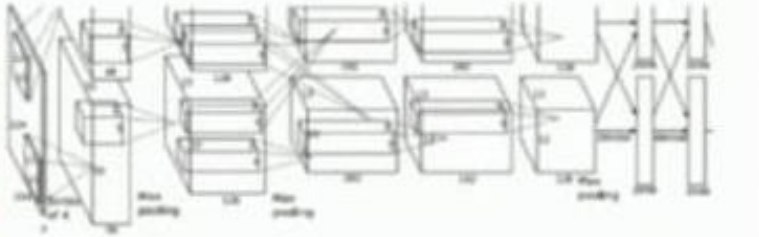
L2 loss

Corect box coordinates:
- x', y', w', h'

Object Detection - Multiple Objects



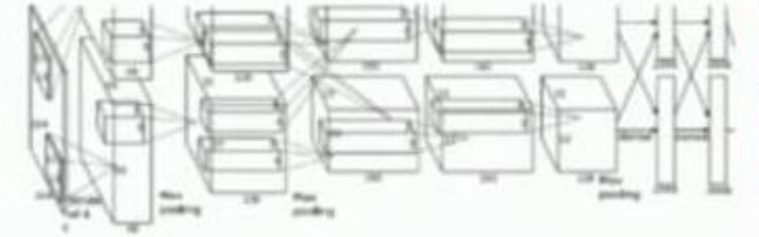
CAT: (x, y, w, h)



DOG: (x, y, w, h)

DOG: (x, y, w, h)

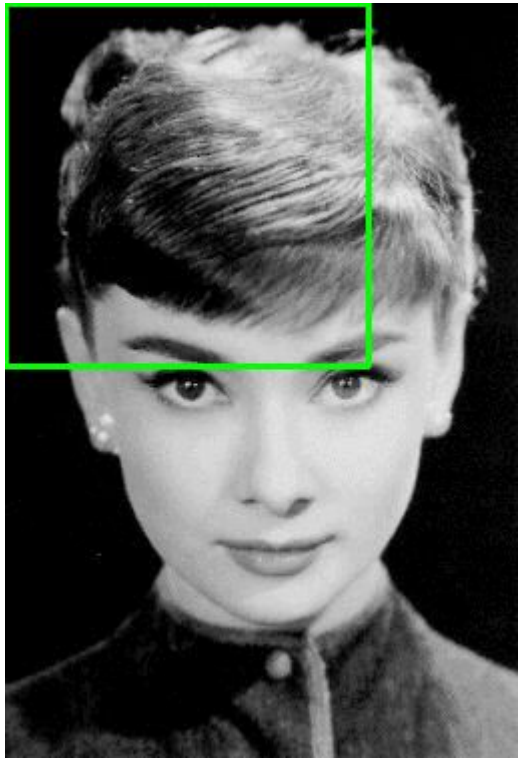
CAT: (x, y, w, h)



DUCK: (x, y, w, h)

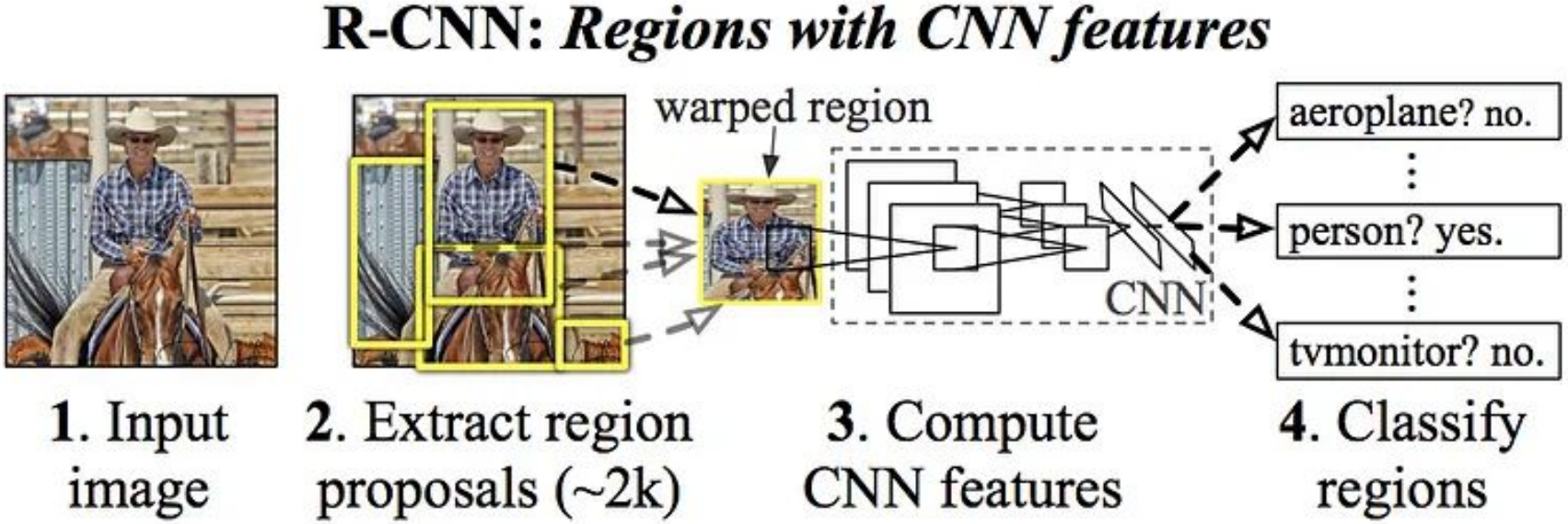
DUCK: (x, y, w, h)

Sliding Windows

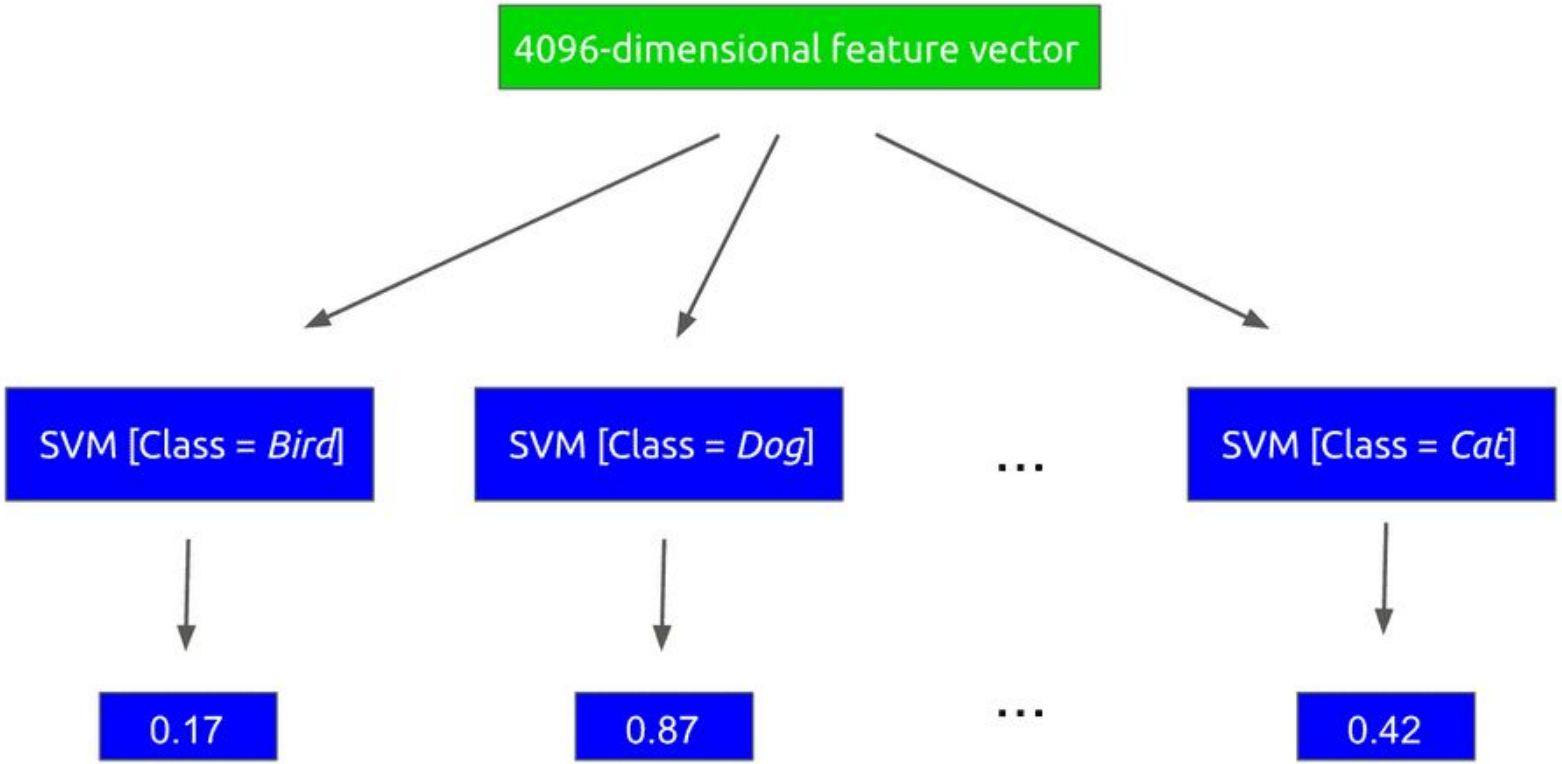


Selective Search

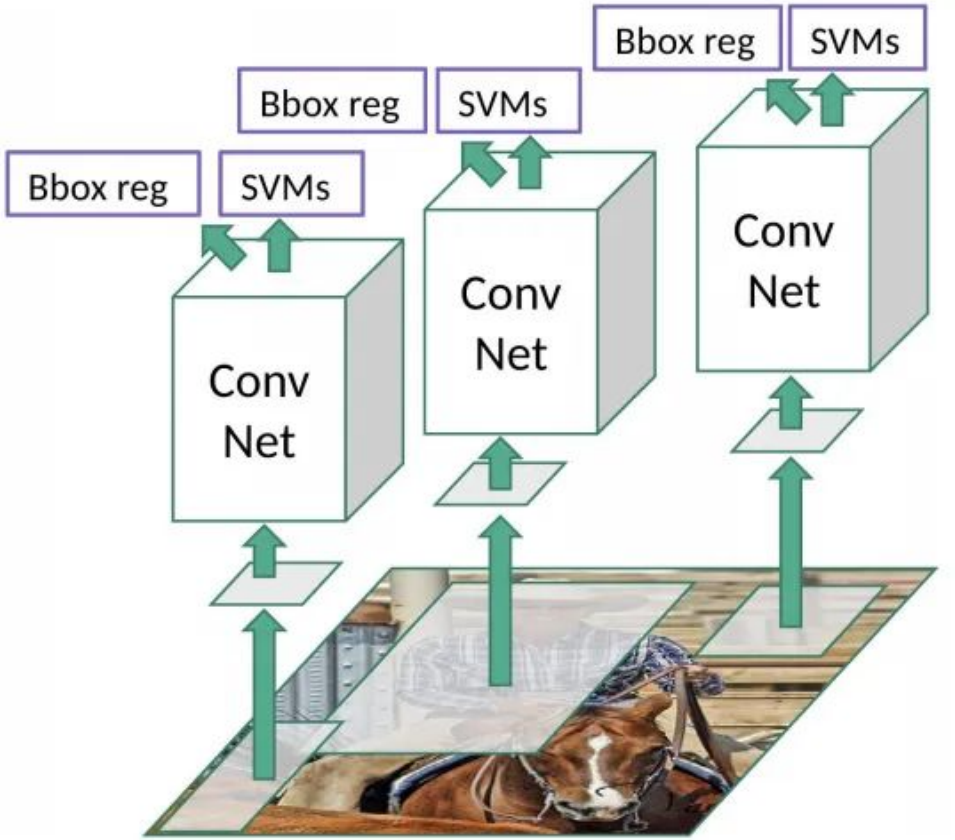
R-CNN -> Fast R-CNN -> Faster R-CNN



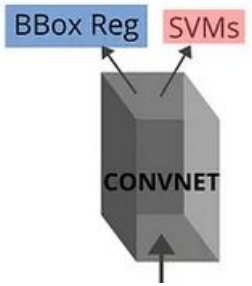
R-CNN -> Fast R-CNN -> Faster R-CNN

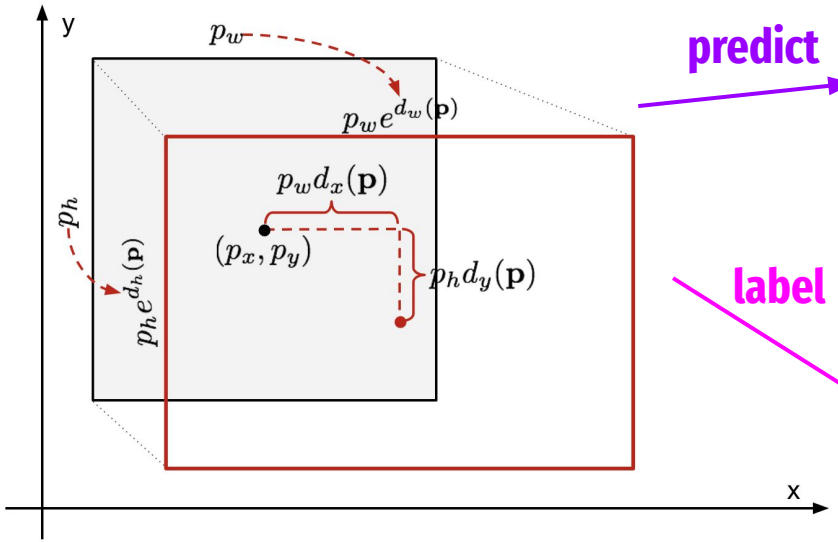


R-CNN -> Fast R-CNN -> Faster R-CNN



Bounding box regression





predict

$$\hat{g}_x = p_w d_x(\mathbf{p}) + p_x$$
$$\hat{g}_y = p_h d_y(\mathbf{p}) + p_y$$
$$\hat{g}_w = p_w \exp(d_w(\mathbf{p}))$$
$$\hat{g}_h = p_h \exp(d_h(\mathbf{p}))$$

label

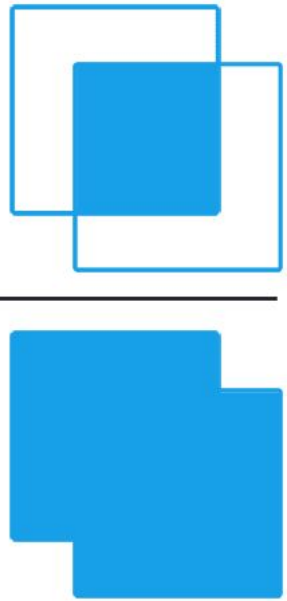
$$t_x = (g_x - p_x)/p_w$$
$$t_y = (g_y - p_y)/p_h$$
$$t_w = \log(g_w/p_w)$$
$$t_h = \log(g_h/p_h)$$

p: region proposals (anchors)
g: actual bbox
ĝ: predicted bbox

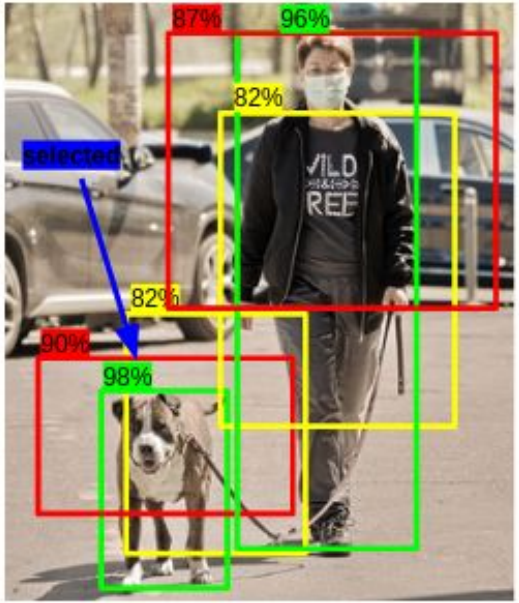
$$\mathcal{L}_{\text{reg}} = \sum_{i \in \{x,y,w,h\}} (t_i - d_i(\mathbf{p}))^2 + \lambda \|\mathbf{w}\|^2$$

Intersection over Union

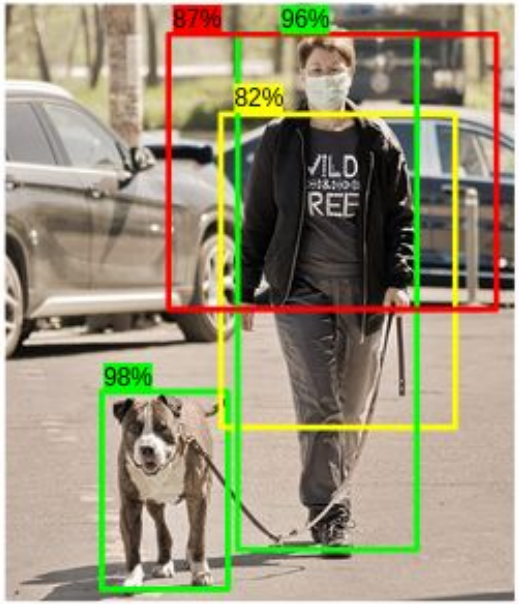
$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



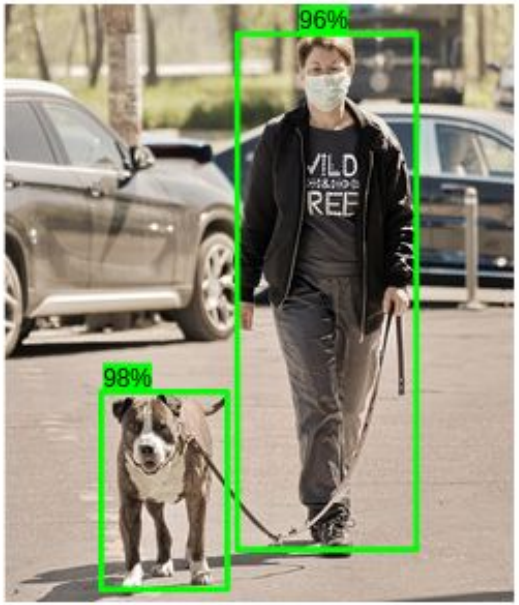
Non-maximum Suppression



Step 1: Selecting Bounding box with highest score

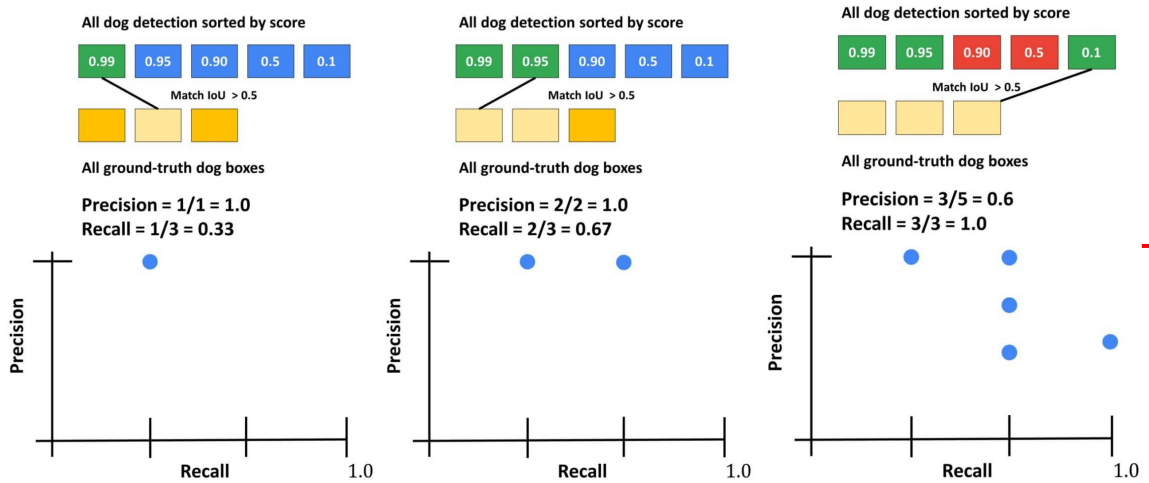


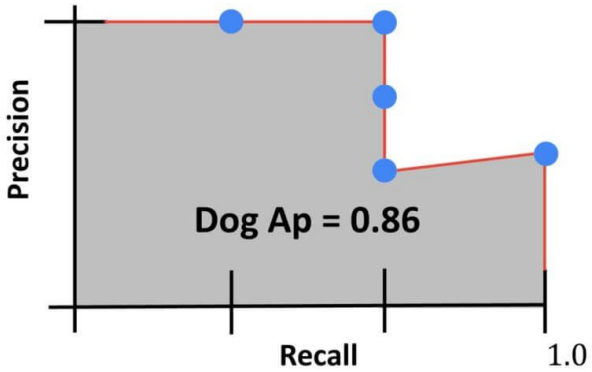
Step 3: Delete Bounding box with high overlap



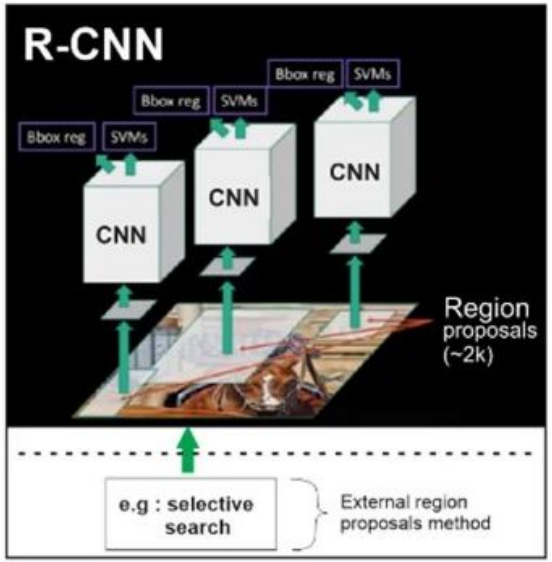
Step 5: Final Output

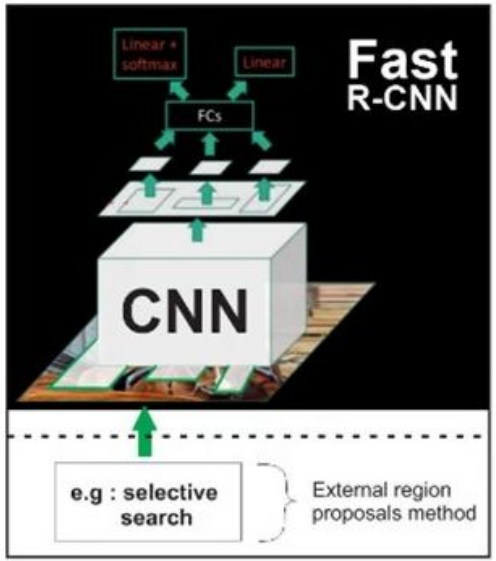
Mean Average Precision





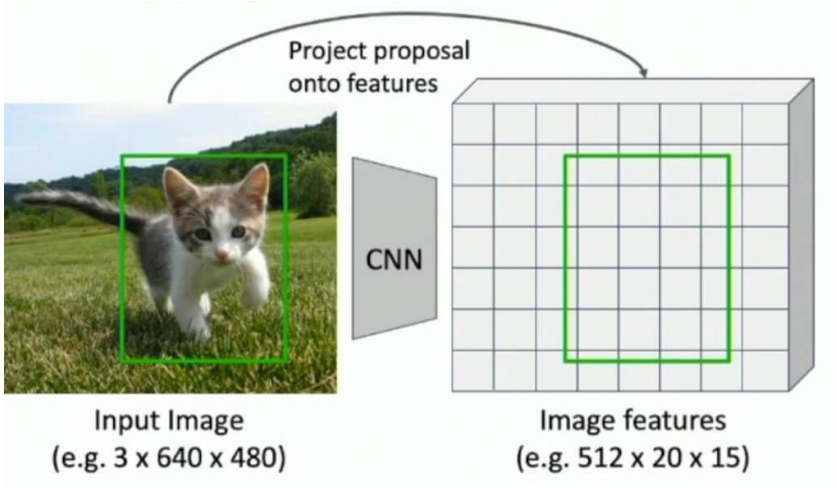
R-CNN -> Fast R-CNN -> Faster R-CNN

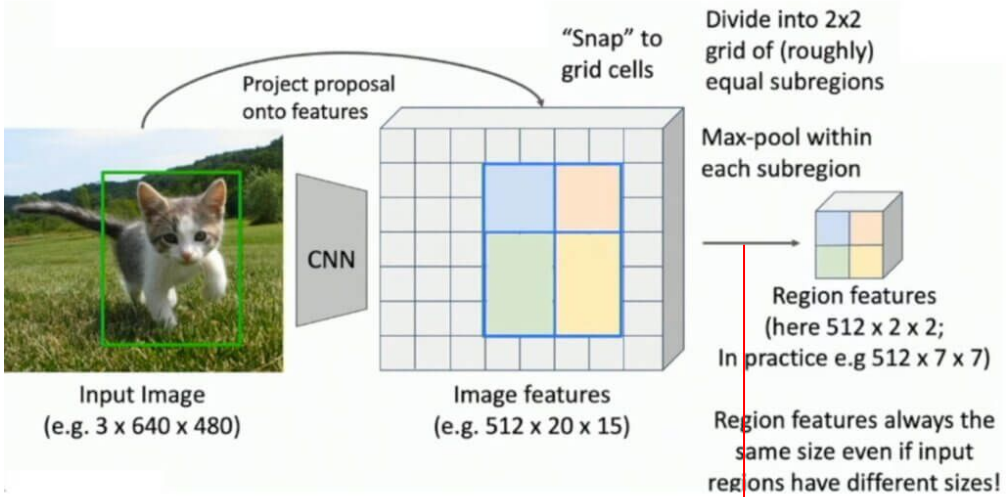




	R-CNN	Fast R-CNN
Test time per image	50 seconds	2 seconds
Speed-up	1x	25x

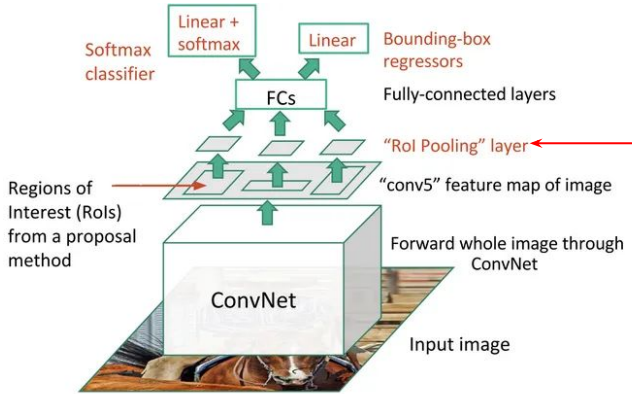
RoI Pooling



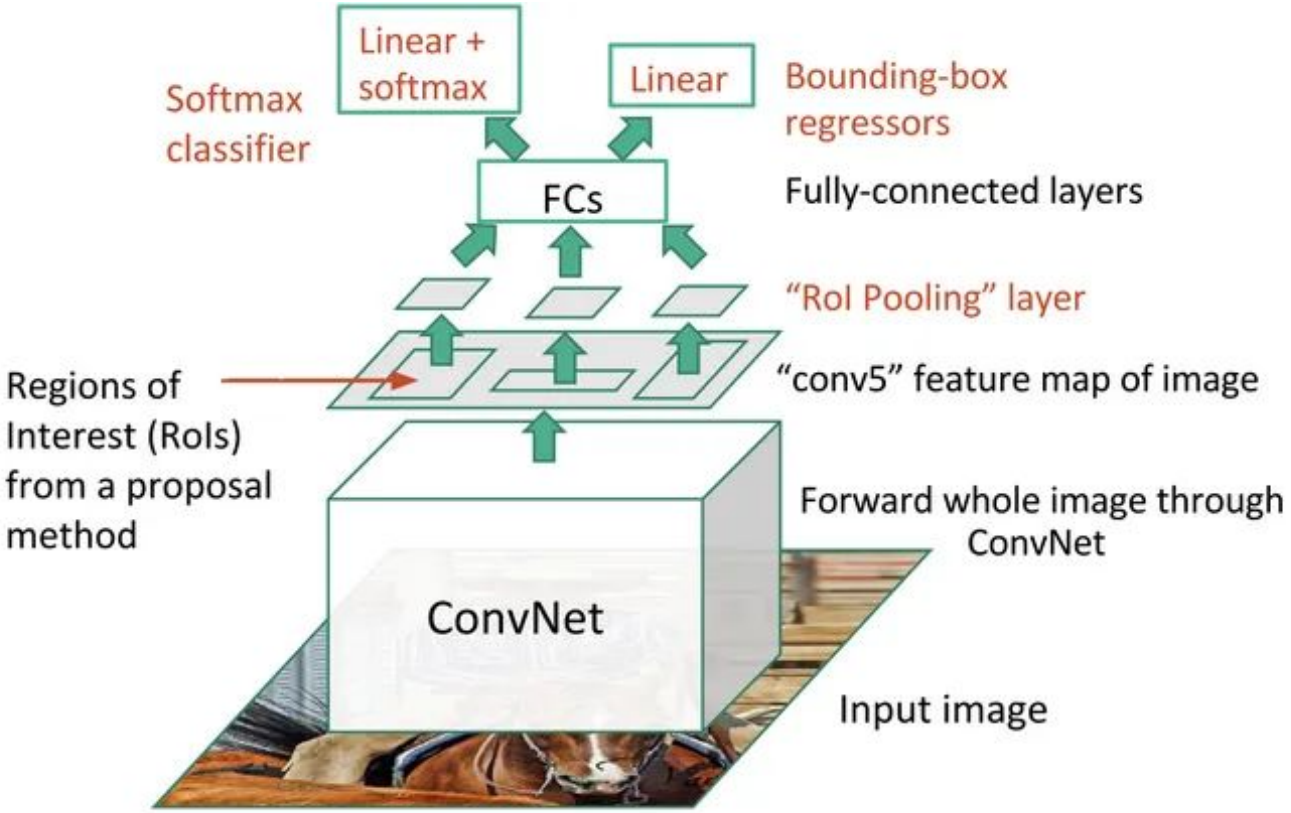


8 x 8

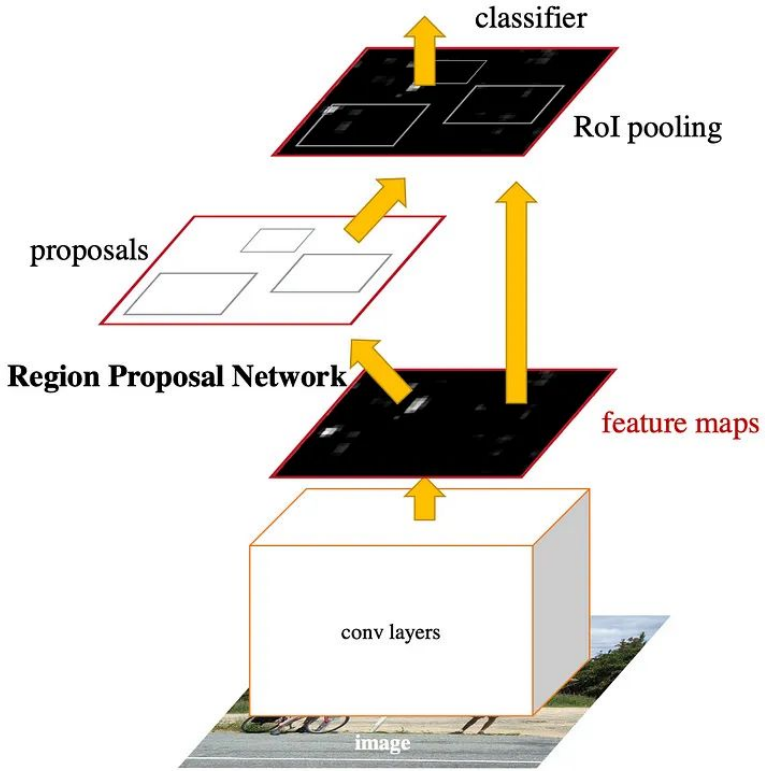
0.88	0.44	0.16	0.14	0.37	0.77	0.96	0.27
0.19	0.45	0.16	0.57	0.63	0.29	0.71	0.70
0.66	0.36	0.64	0.82	0.54	0.73	0.59	0.25
0.85	0.24	0.84	0.76	0.29	0.75	0.62	0.24
0.32	0.74	0.39	0.31	0.34	0.03	0.33	0.48
0.20	0.69	0.13	0.16	0.73	0.65	0.96	0.32
0.19	0.14	0.86	0.09	0.88	0.07	0.01	0.48
0.83	0.24	0.04	0.97	0.34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



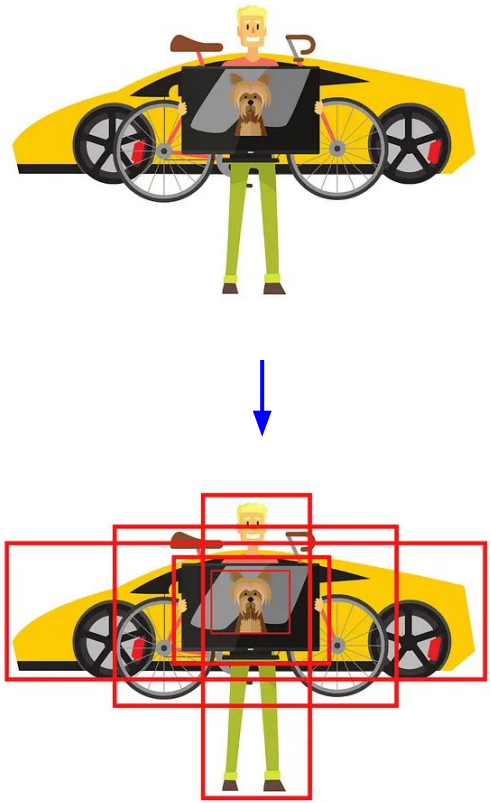
R-CNN -> Fast R-CNN -> Faster R-CNN

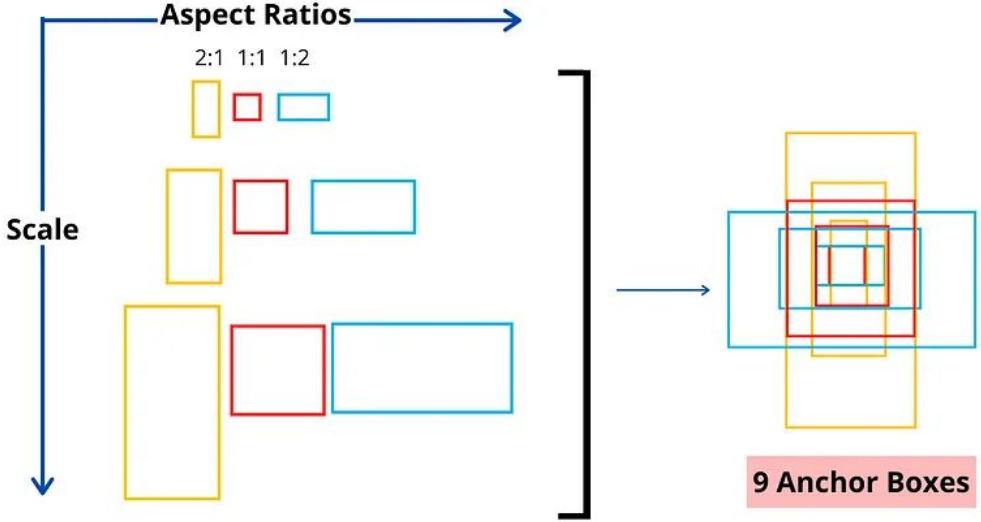


R-CNN -> Fast R-CNN -> Faster R-CNN

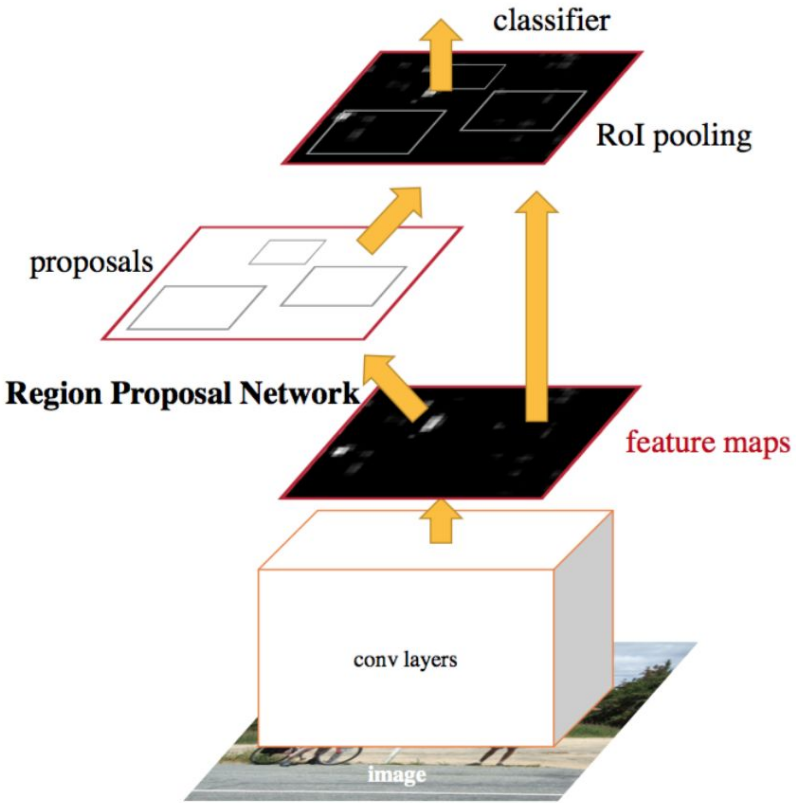


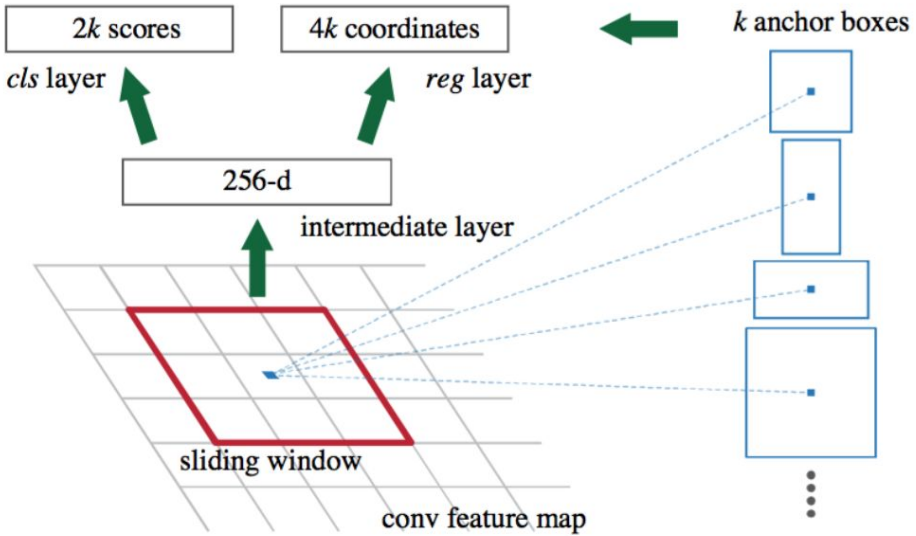
Anchors





Region Proposal Network (RPN)



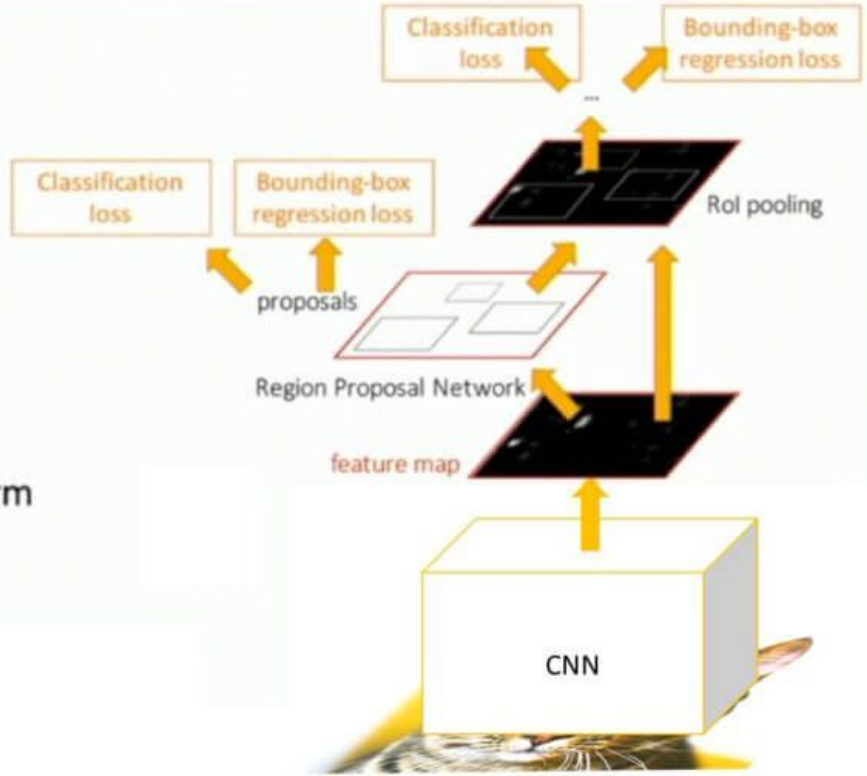


Faster R-CNN losses

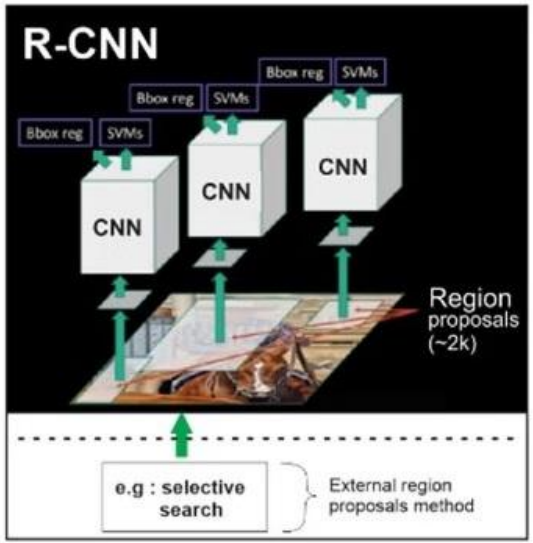
Faster R-CNN: Learnable region proposals

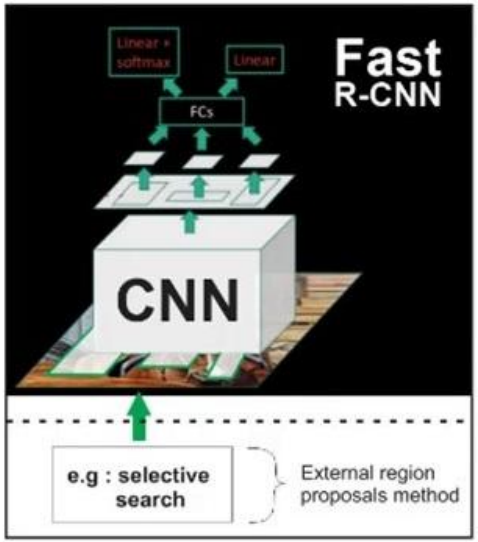
Jointly train with 4 losses:

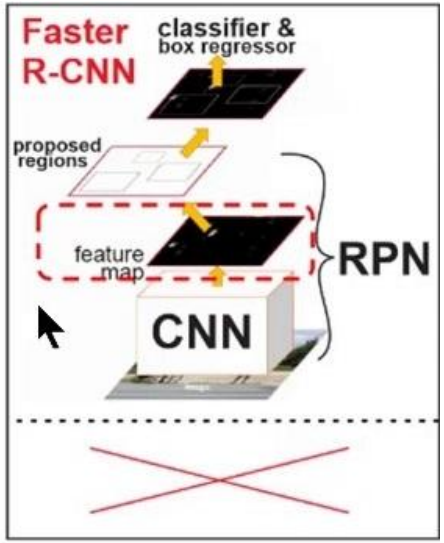
1. **RPN classification**: anchor box is object / not an object
2. **RPN regression**: predict transform from anchor box to proposal box
3. **Object classification**: classify proposals as background / object class
4. **Object regression**: predict transform from proposal box to object box



Comparison



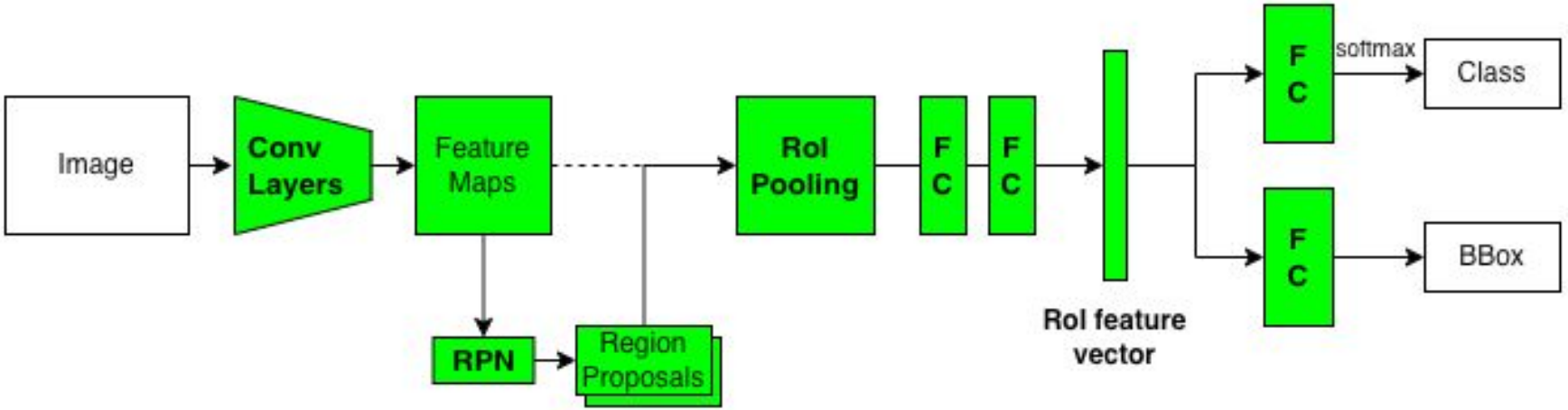


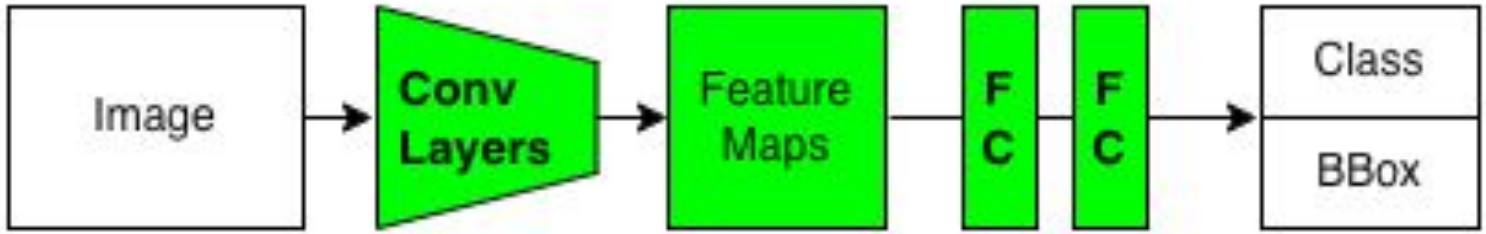


	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image	50 seconds	2 seconds	0.2 seconds
Speed-up	1x	25x	250x

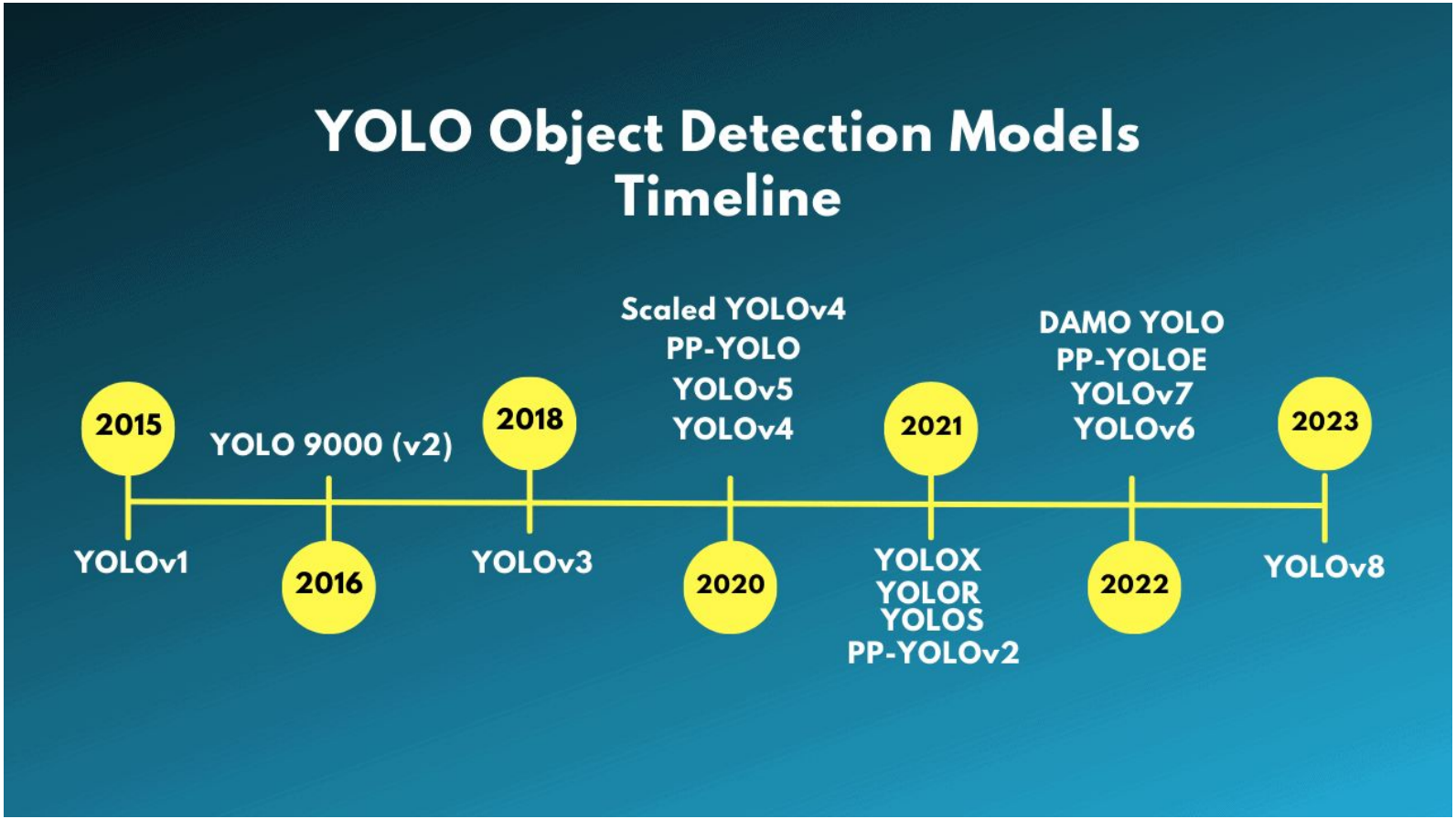
You Only Look Once

Faster R-CNN vs YOLO

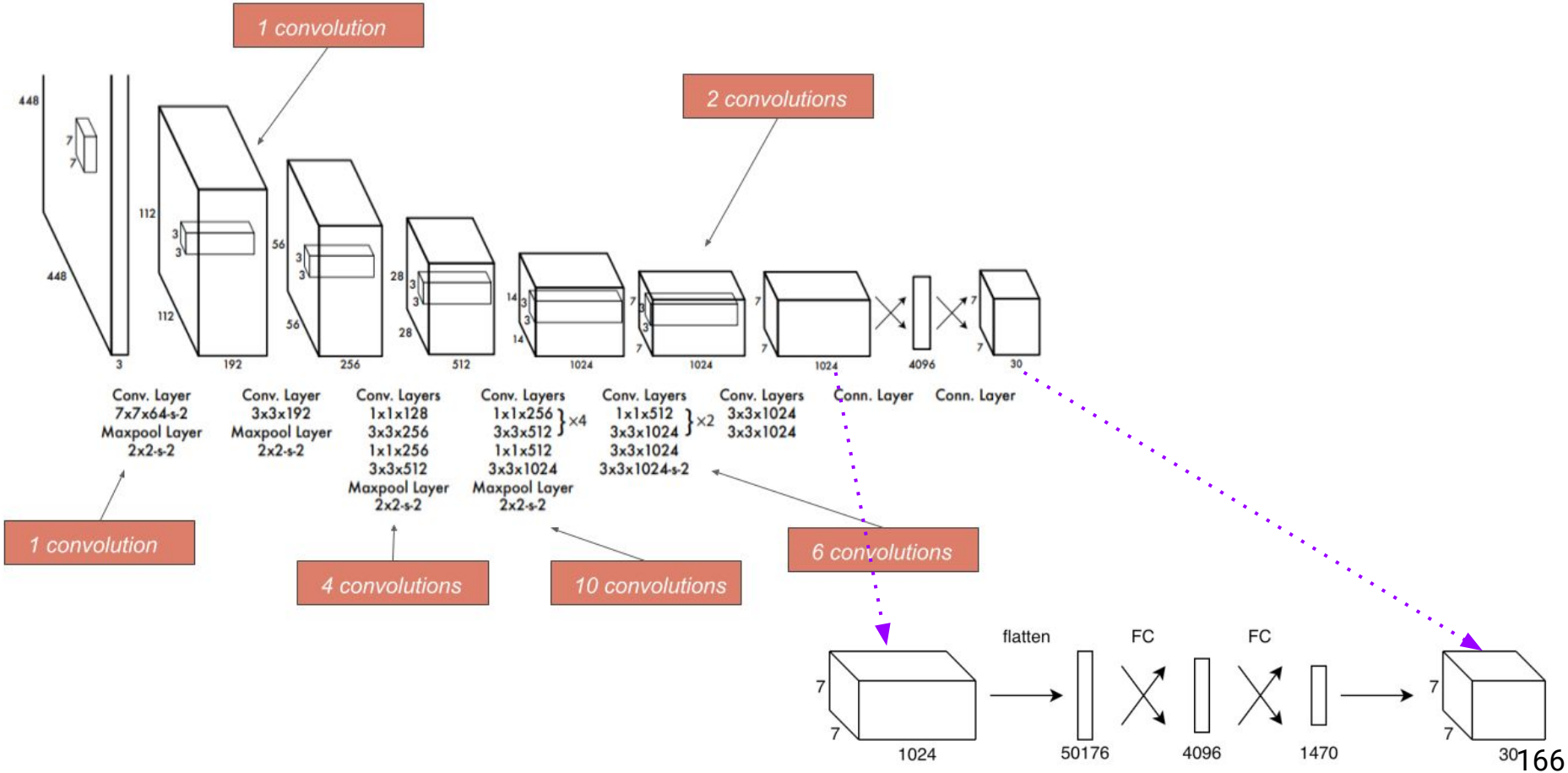




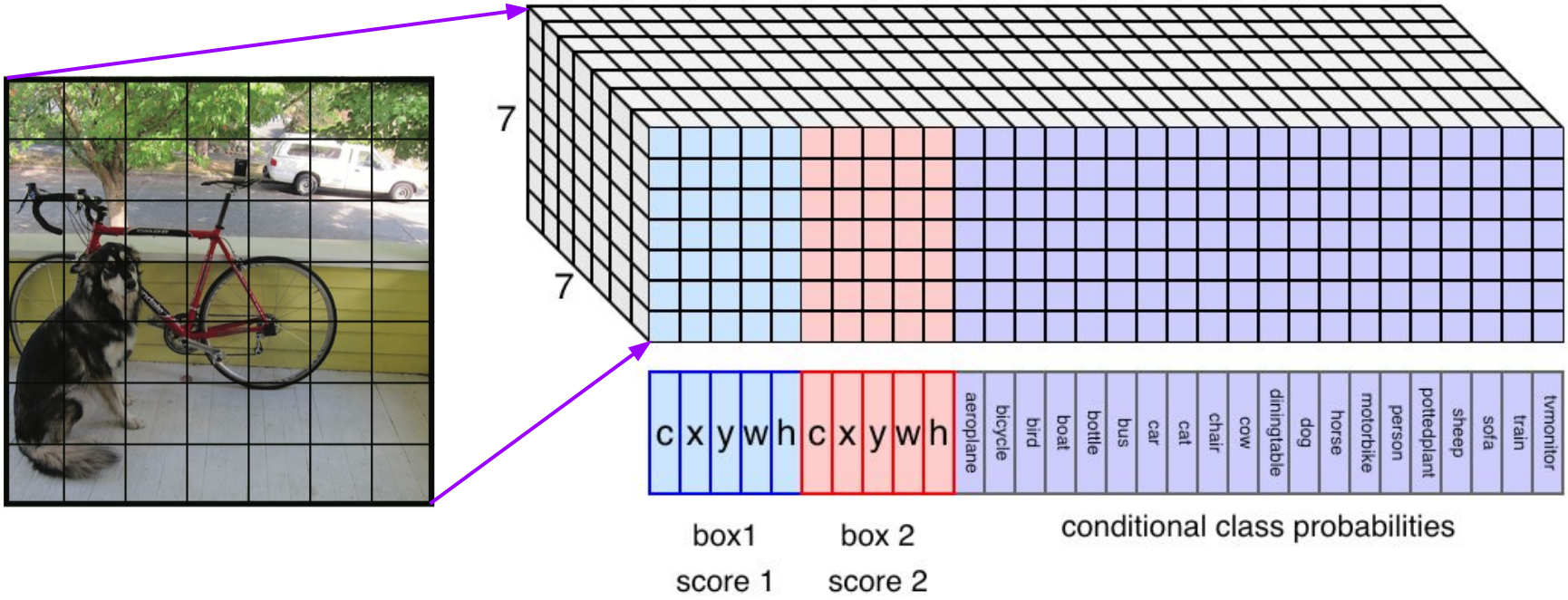
YOLO


YOLO Object Detection Models Timeline
2015
YOLOv1
YOLO 9000 (v2)
2016
2018
YOLOv3
Scaled YOLOv4
PP-YOLO
YOLOv5
YOLOv4
2020
2021
YOLOX
YOLOR
YOLOS
PP-YOLOv2
DAMO YOLO
PP-YOLOE
YOLOv7
YOLOv6
2022
2023
YOLOv8

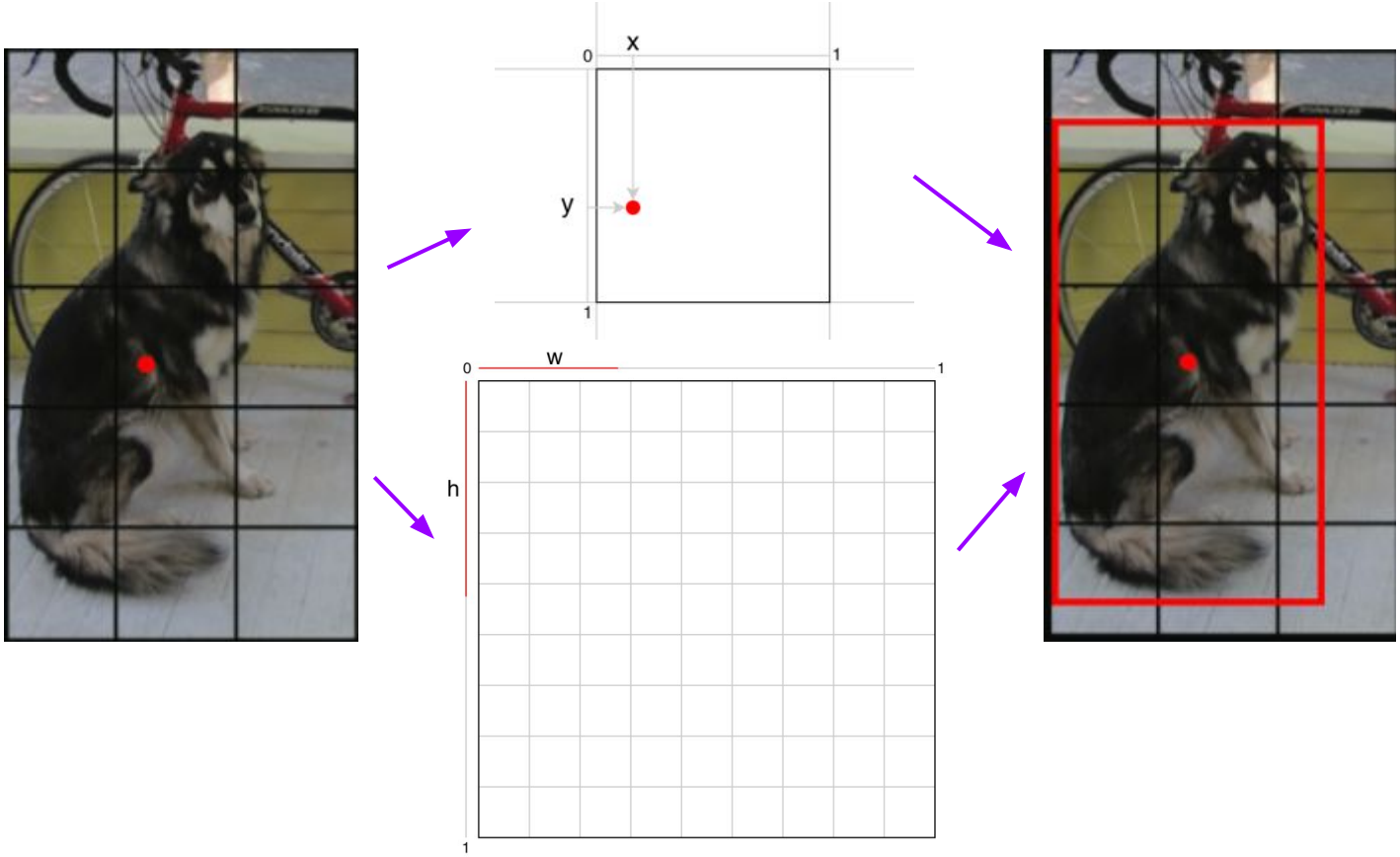
YOLO v1


1 convolution
2 convolutions
448
448
7
7
3
112
112
192
56
56
256
28
28
512
14
14
1024
7
7
1024
7
7
1024
4096
7
7
30
Conv. Layer
7x7x64-s-2
Maxpool Layer
2x2-s-2
Conv. Layer
3x3x192
Maxpool Layer
2x2-s-2
Conv. Layers
1x1x128
3x3x256
1x1x256
3x3x512
Maxpool Layer
2x2-s-2
Conv. Layers
1x1x256
3x3x512
×4
1x1x512
3x3x1024
Maxpool Layer
2x2-s-2
Conv. Layers
1x1x512
3x3x1024
×2
3x3x1024
3x3x1024-s-2
Conv. Layers
3x3x1024
3x3x1024
Conn. Layer
Conn. Layer
1 convolution
4 convolutions
10 convolutions
6 convolutions
flatten
FC
FC
7
7
1024
50176
4096
1470
7
7
30

YOLO v1: feature map

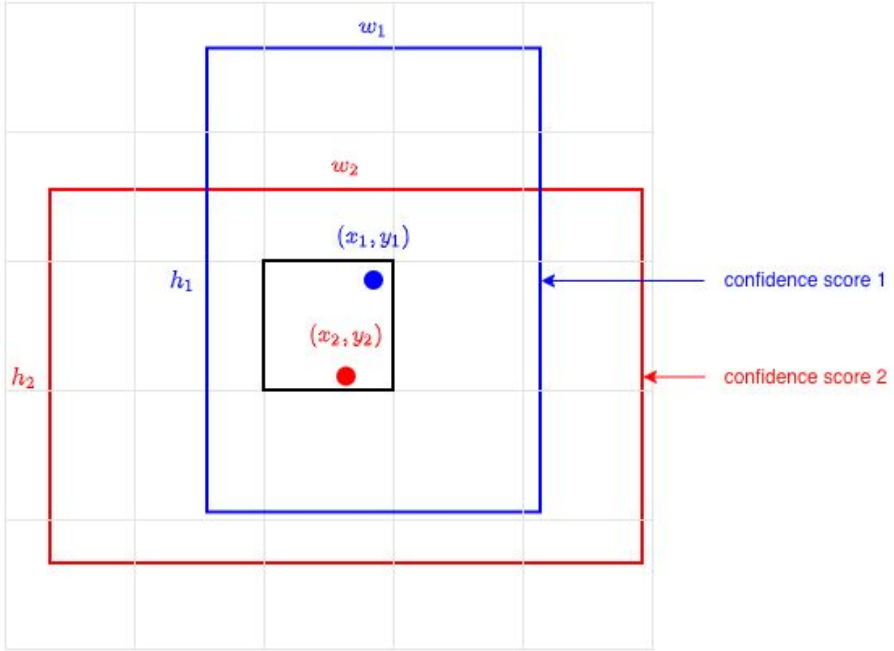


YOLO v1: Bounding box

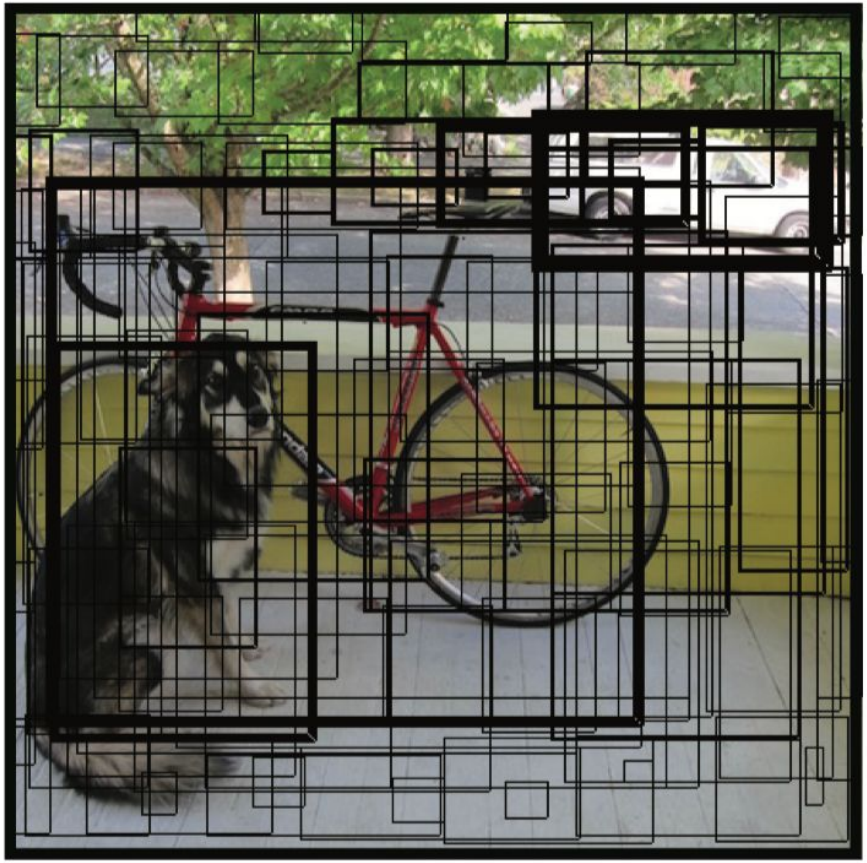


YOLO v1: Confidence score

$$\text{confidence\_score} = \Pr(\text{Object}) \times \text{IoU}^{\text{truth}}_{\text{pred}}$$

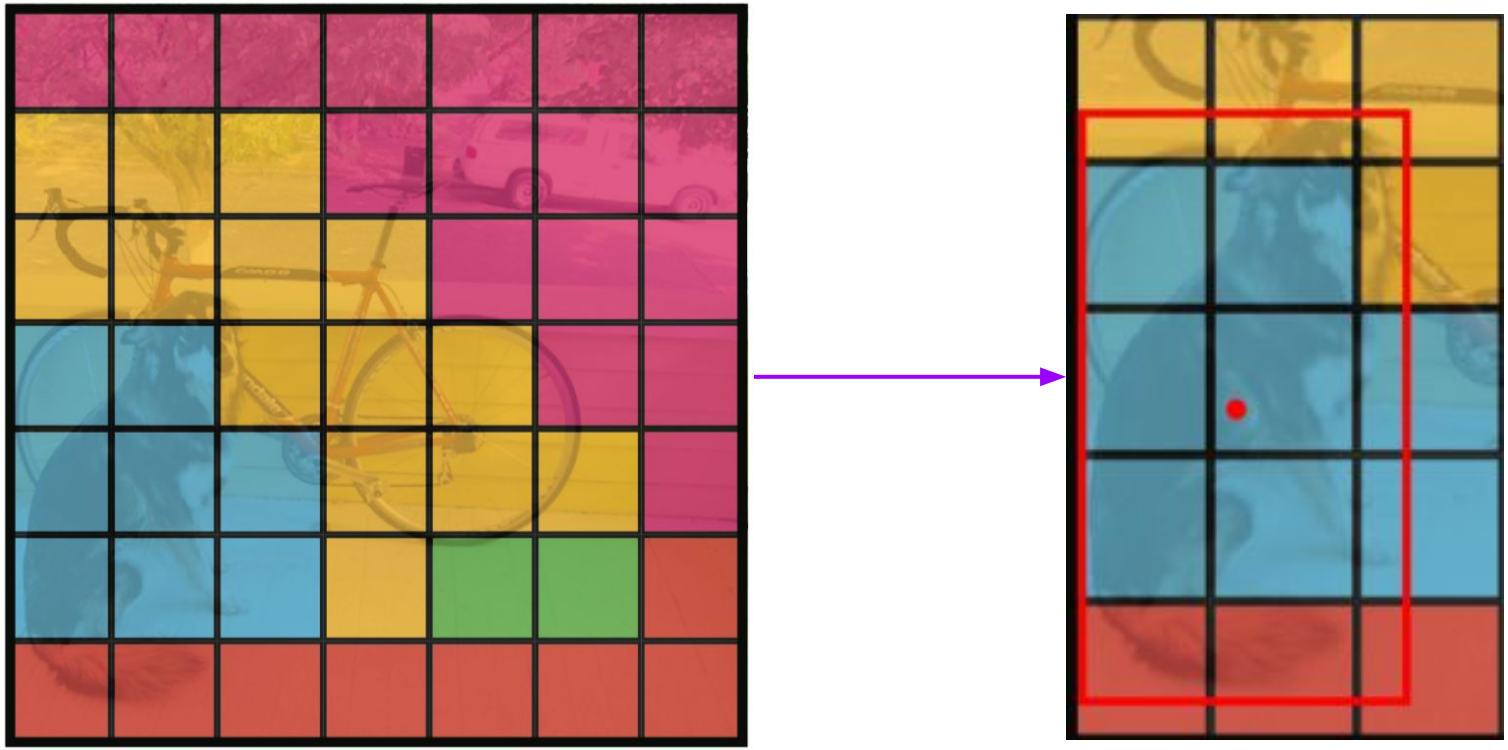


YOLO v1: Bounding box + Confidence score

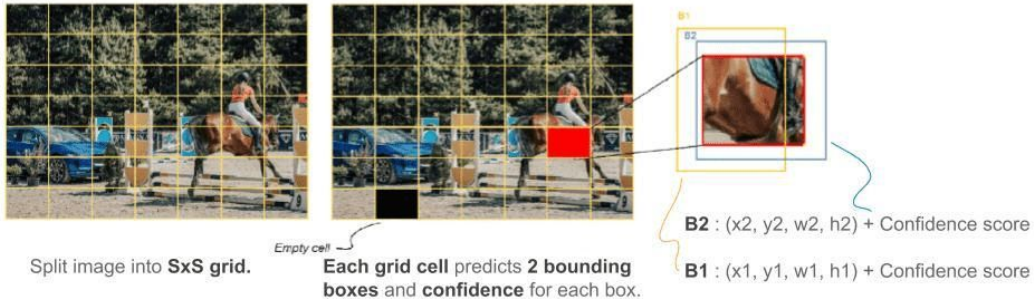


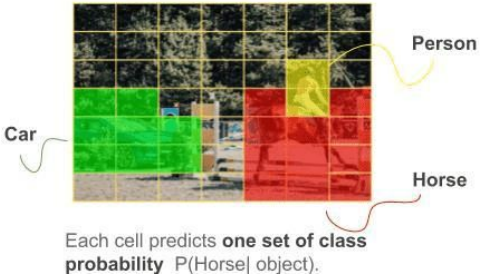
YOLO v1: Class confidence score

$$\begin{aligned}\text{class\_confidence\_score} &= \Pr(\text{Class}_i|\text{Object}) \times \Pr(\text{Object}) \times \text{IoU}^{\text{truth}}_{\text{pred}} \\ &= \Pr(\text{Class}_i) \times \text{IoU}^{\text{truth}}_{\text{pred}}\end{aligned}$$

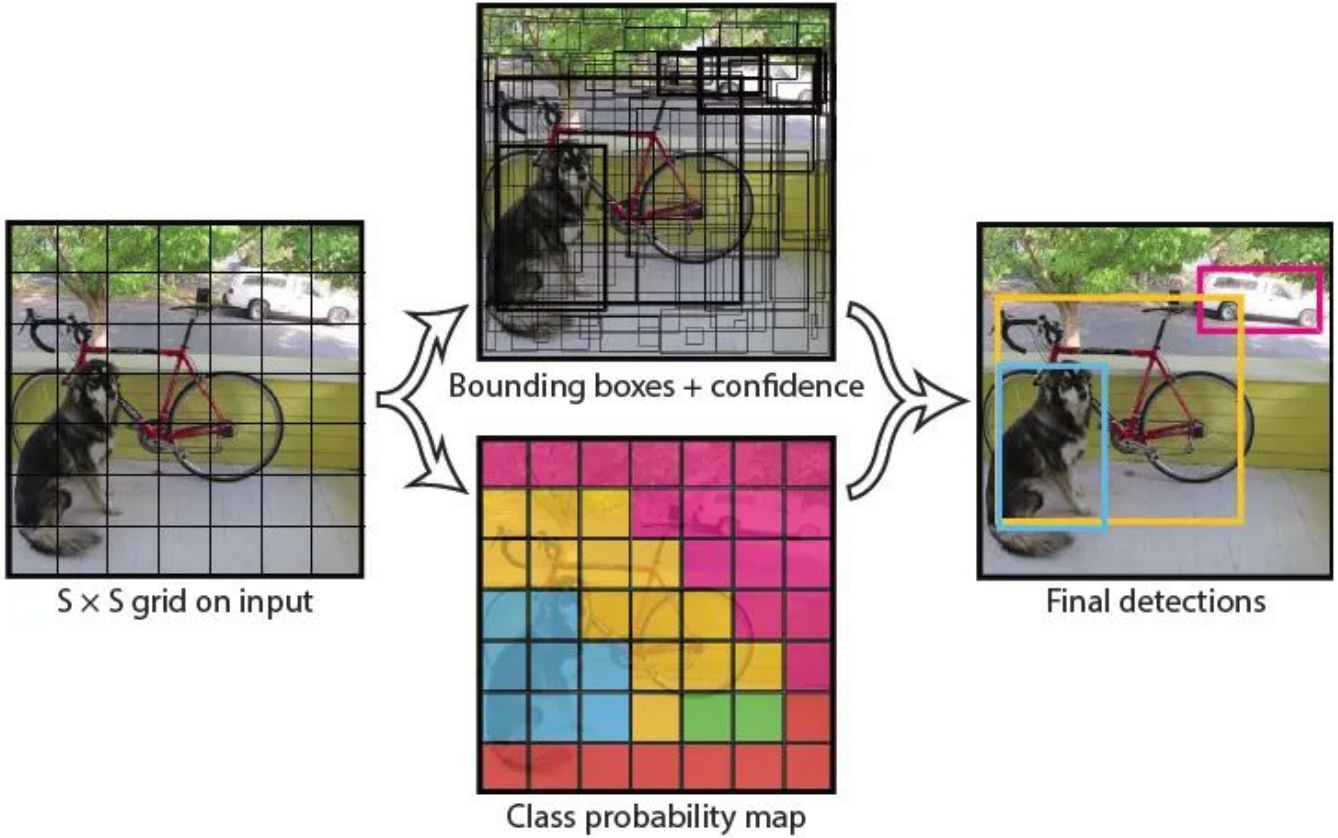


YOLO v1: Put all together

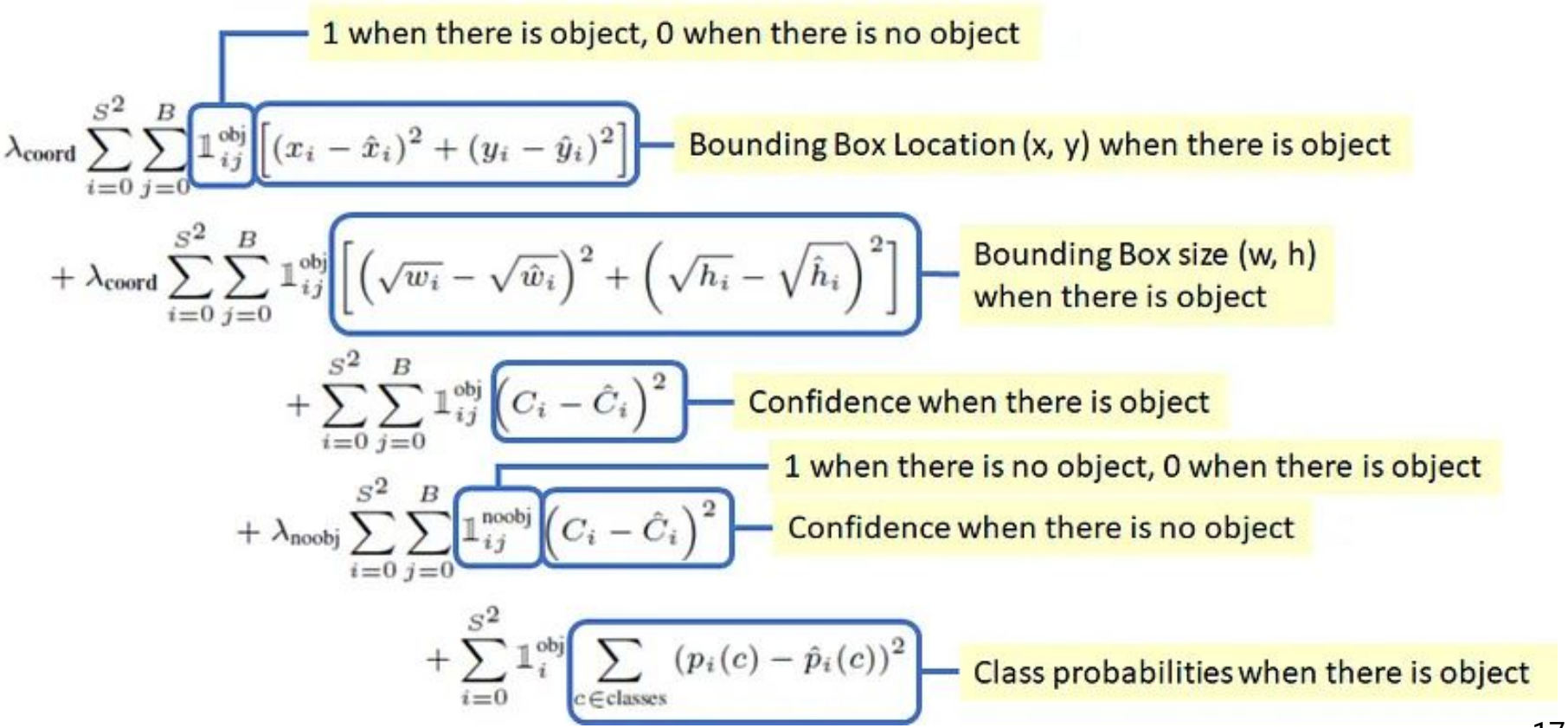




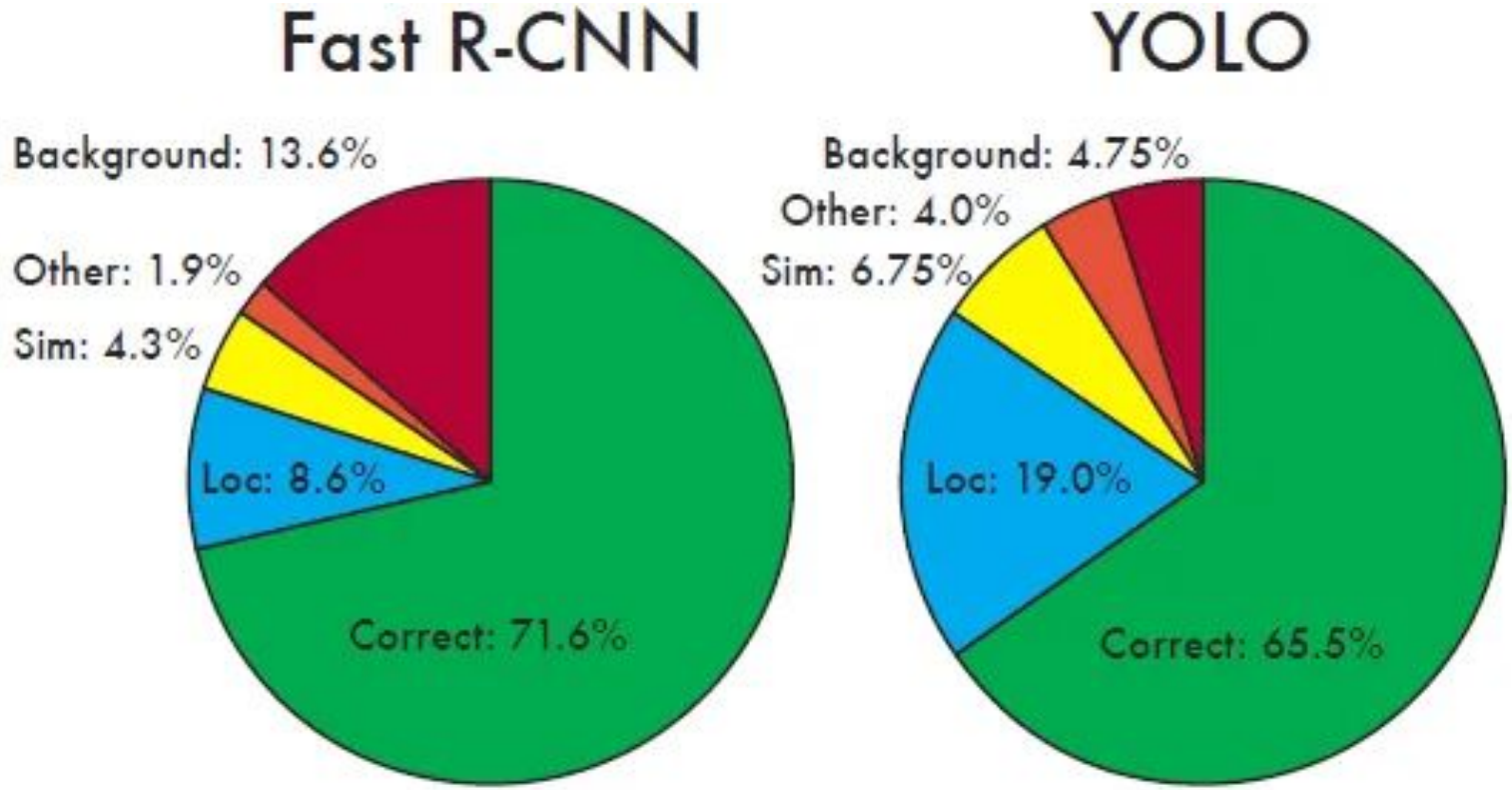
YOLO v1: Post-processing



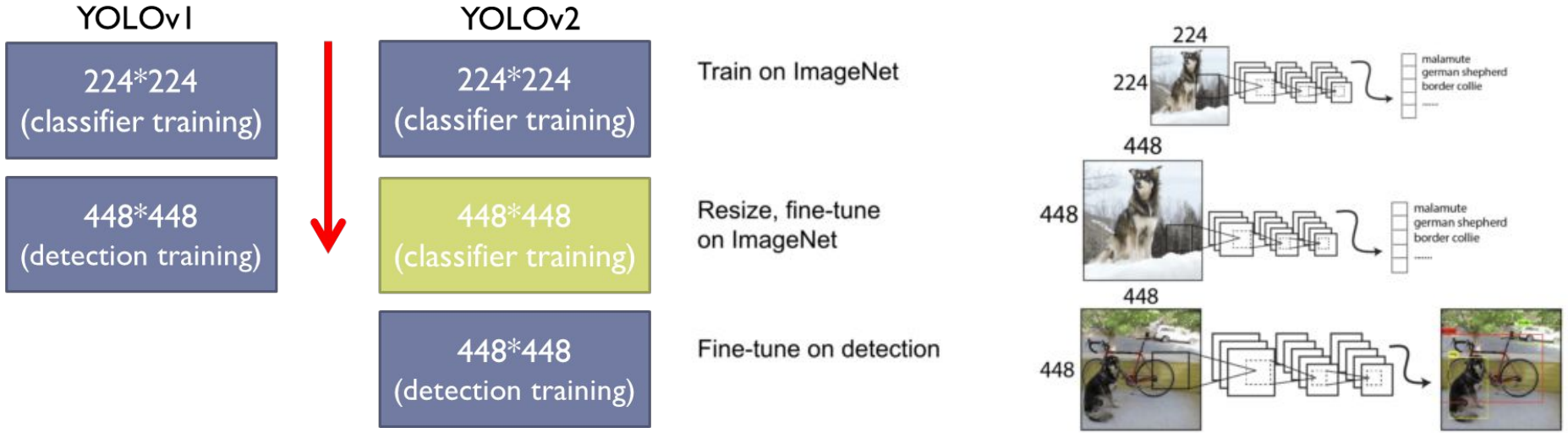
YOLO v1: Loss function



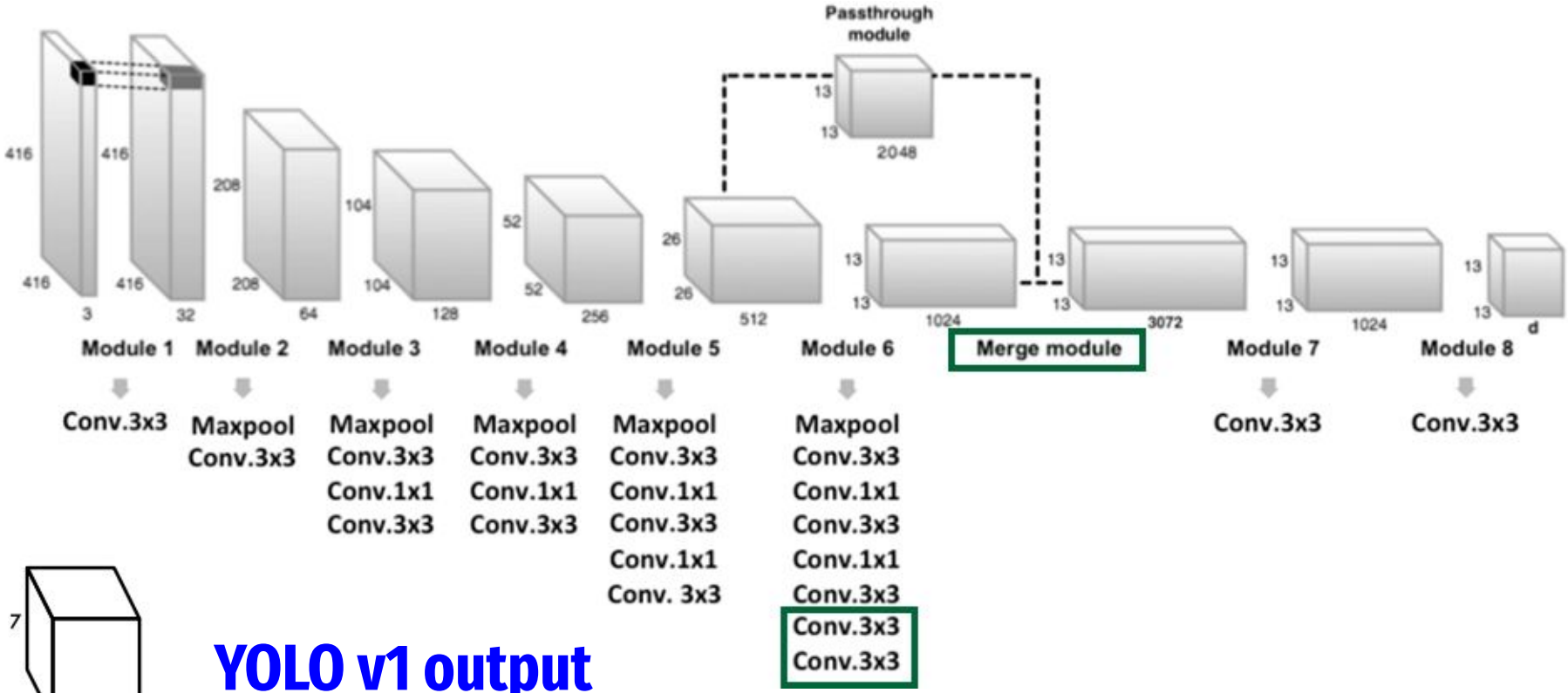
YOLO v1: Evaluation



YOLO v2: Adjustments from Yolov1

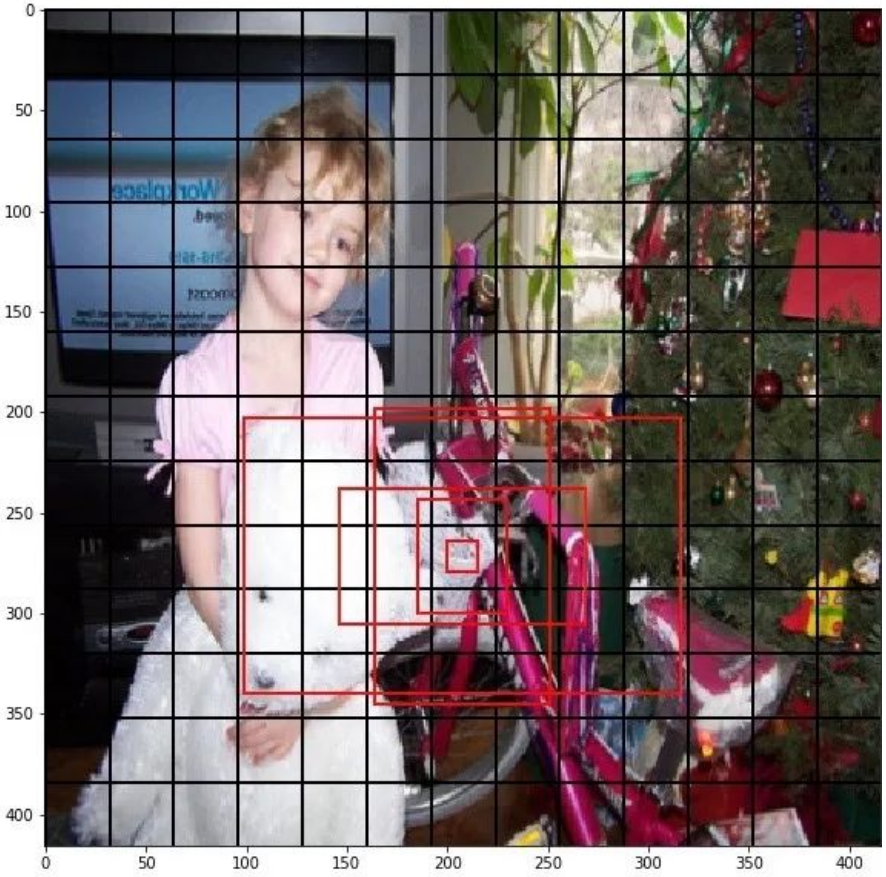


YOLO v2: Adjustments from Yolov1

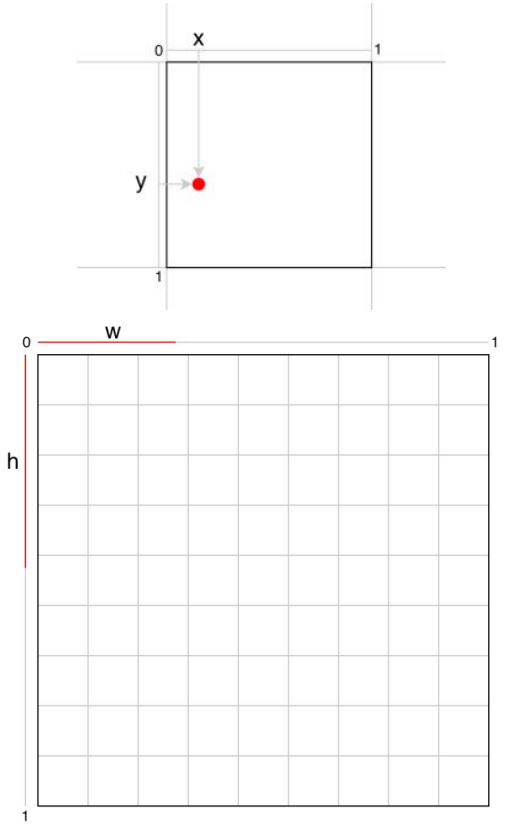


YOLO v1 output

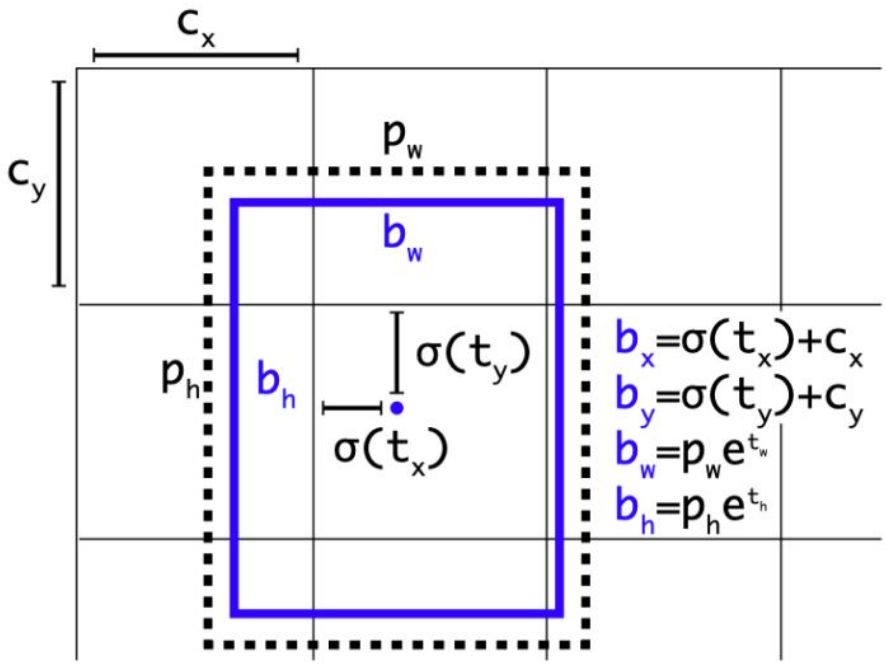
YOLO v2: Adjustments from Yolov1



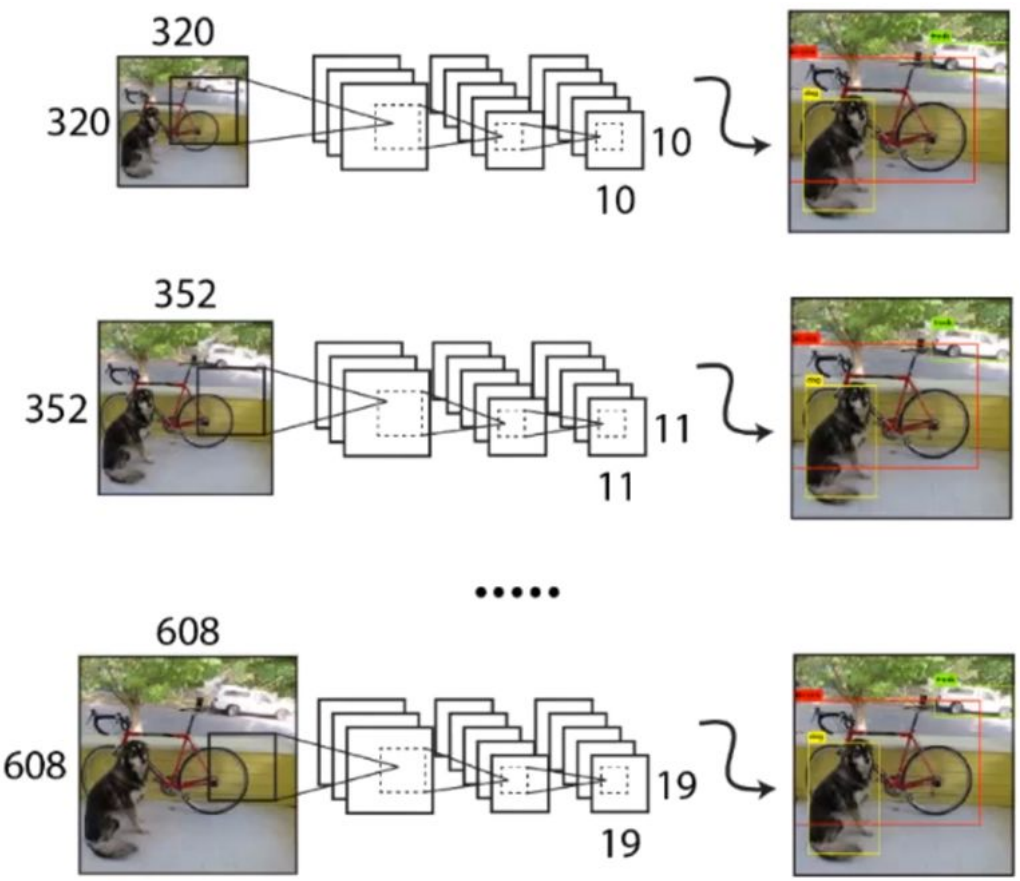
YOLO v2: Adjustments from Yolov1



VS



YOLO v2: Adjustments from Yolov1



320/32=10

352/32=11

608/32=19

From YOLO v2 to YOLO v3

YOLOV2

Type	Filters	Size	Output
Convolutional	32	3 x 3	224 x 224
Maxpool		2 x 2 / 2	112 x 112
Convolutional	64	3 x 3	112 x 112
Maxpool		2 x 2 / 2	56 x 56
Convolutional	128	3 x 3	56 x 56
Convolutional	64	1 x 1	56 x 56
Convolutional	128	3 X 3	56 X 56
Maxpool		2 x 2 / 2	28 X 28
Convolutional	256	3 x 3	28 X 28
Convolutional	128	1 x 1	28 X 28
Convolutional	256	3 X 3	28 X 28
Maxpool		2 x 2 / 2	14 X 14
Convolutional	512	3 x 3	14 X 14
Convolutional	256	1 x 1	14 X 14
Convolutional	512	3 X 3	14 X 14
		2 x 2 / 2	7 X 7
Convolutional	1024	3 x 3	7 X 7
Convolutional	512	1 x 1	7 X 7
Convolutional	1024	3 X 3	7 X 7
Convolutional	512	1 x 1	7 X 7
Convolutional	1024	3 X 3	7 X 7
Convolutional	1000	1 X 1	7 X 7
Avgpool		Global	1000

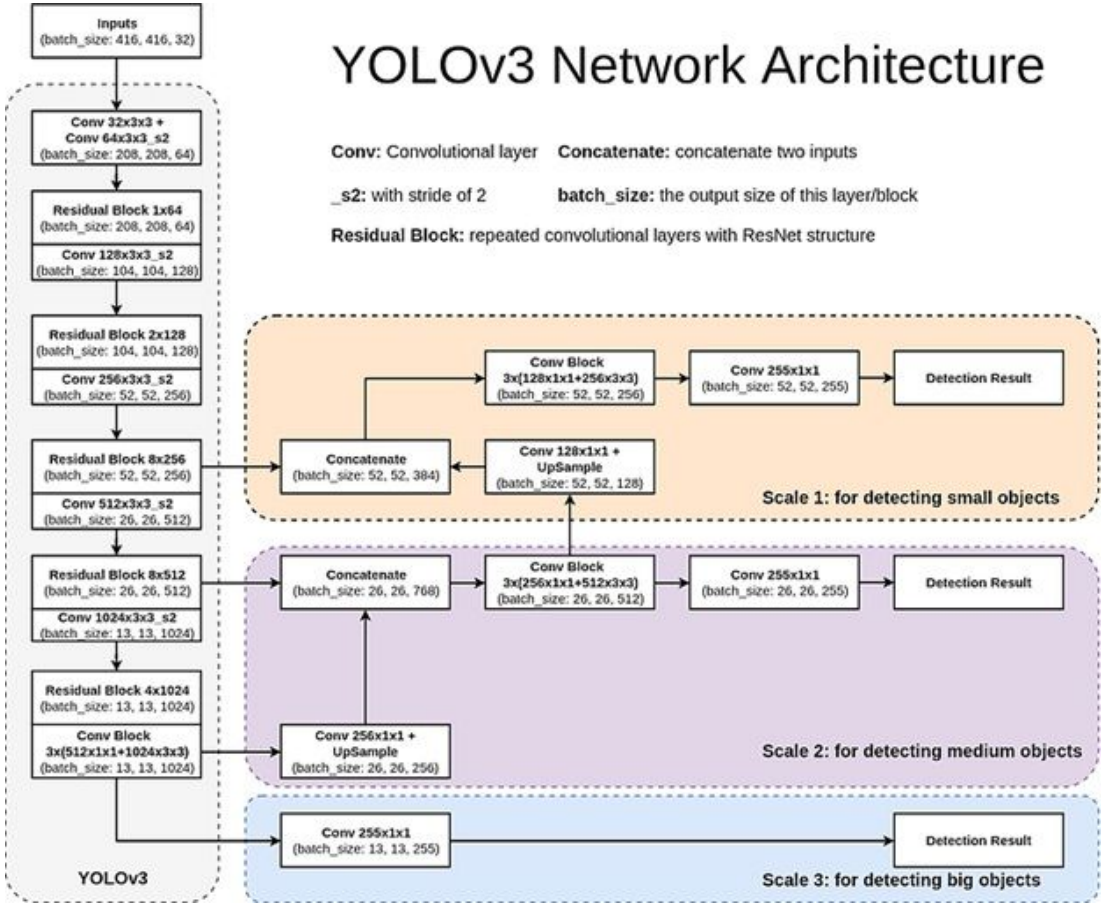
Darknet-19

YOLOV3

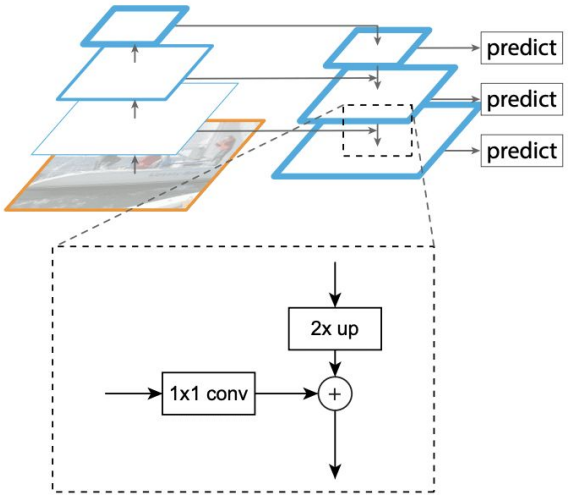
	Type	Filters	Size	Output
	Convolutional	32	3 x 3	256 x 256
	Convolutional	64	3 x 3 / 2	128 x 128
1 x	Convolutional	32	1 x 1	
	Convolutional	64	3 X 3	
	Residual			128 X 128
	Convolutional	128	3 x 3 / 2	64 x 64
2 x	Convolutional	64	1 x 1	
	Convolutional	128	3 x 3	
	Residual			64 X 64
	Convolutional	256	3 x 3 / 2	32 x 32
8 x	Convolutional	128	1 x 1	
	Convolutional	256	3 x 3	
	Residual			32 x 32
	Convolutional	512	3 x 3 / 2	32 x 32
8 x	Convolutional	256	1 x 1	
	Convolutional	512	3 x 3	
	Residual			16 x 16
	Convolutional	1024	3 x 3 / 2	32 x 32
4 x	Convolutional	512	1 x 1	
	Convolutional	1024	3 x 3	
	Residual			8 x 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

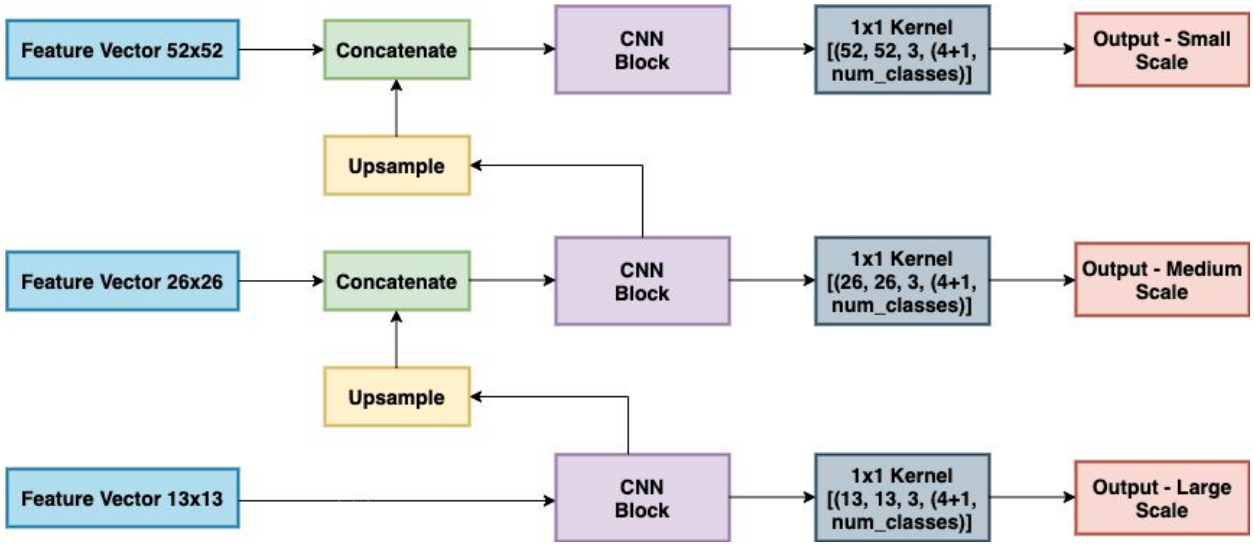
Darknet-53

YOLO v3: Residual block and skip connection



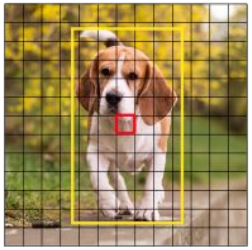
YOLO v3: Feature Pyramid Network





YOLO v3: Different scales

Prediction Feature Maps at different Scales

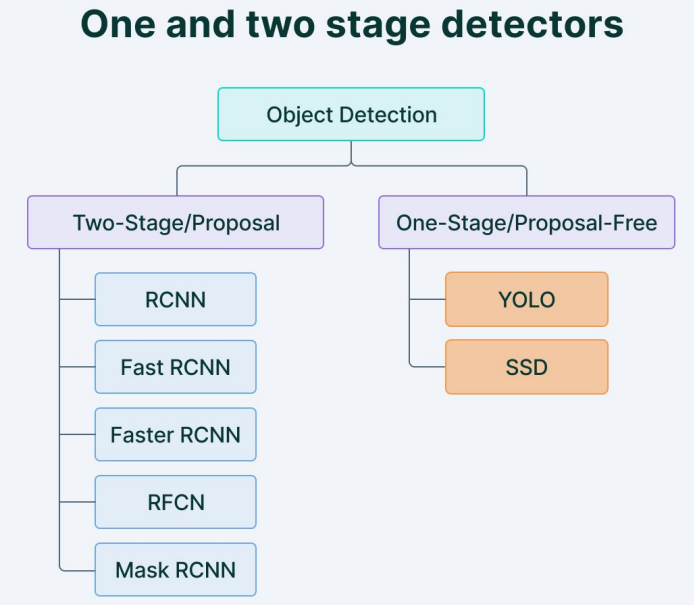


13 x 13

26 x 26

52 x 52

One-stage vs two-stage object detector



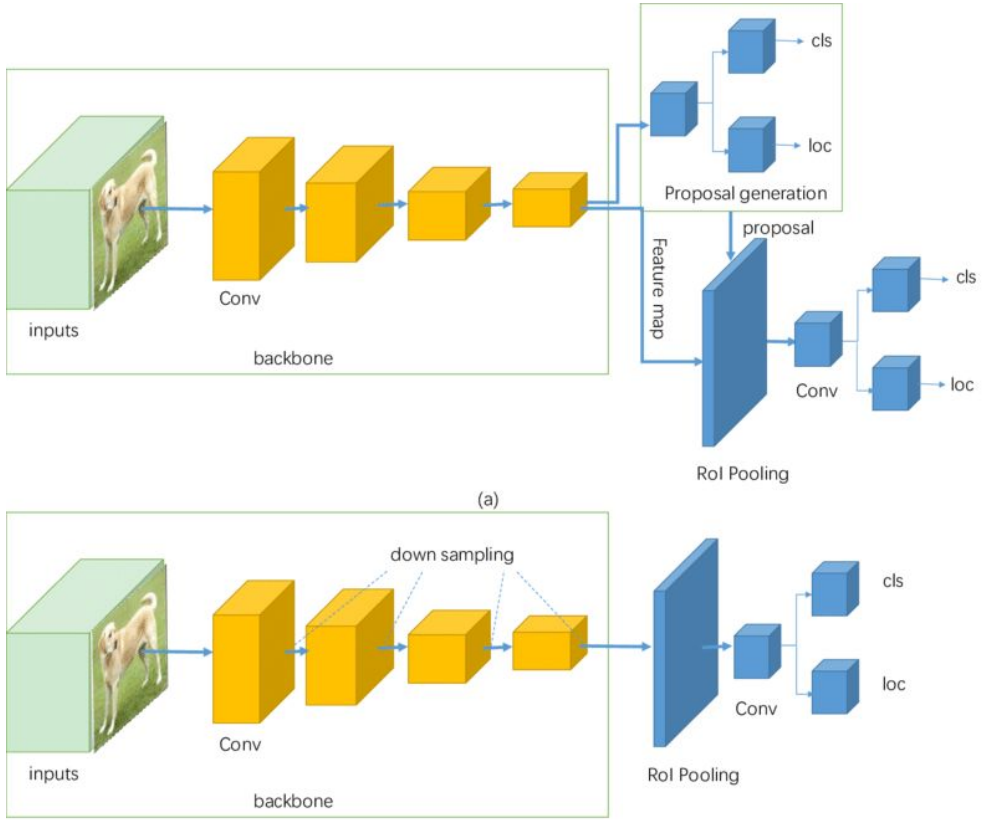
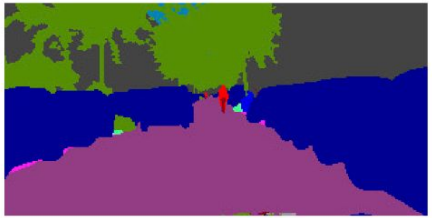


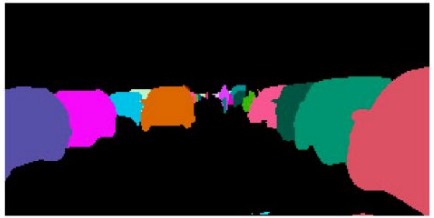
Image Segmentation

Types of Image segmentation tasks

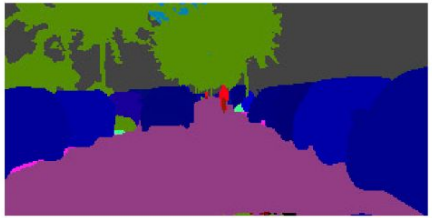
(a) Image



(b) Semantic Segmentation



(c) Instance Segmentation



(d) Panoptic Segmentation

Input

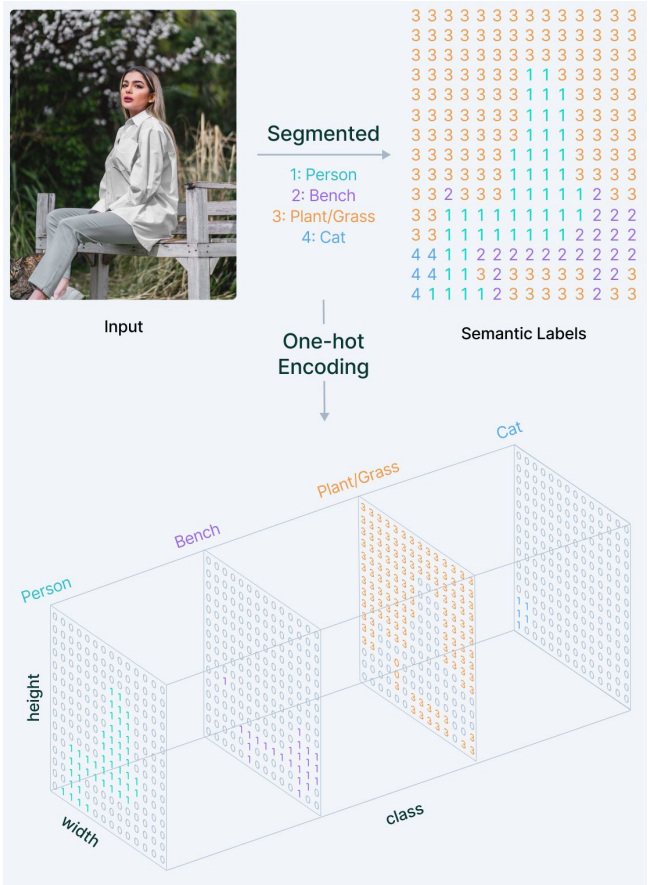
segmented

1: Person
2: Purse
3: Plants/Grass
4: Sidewalk
5: Building/Structures

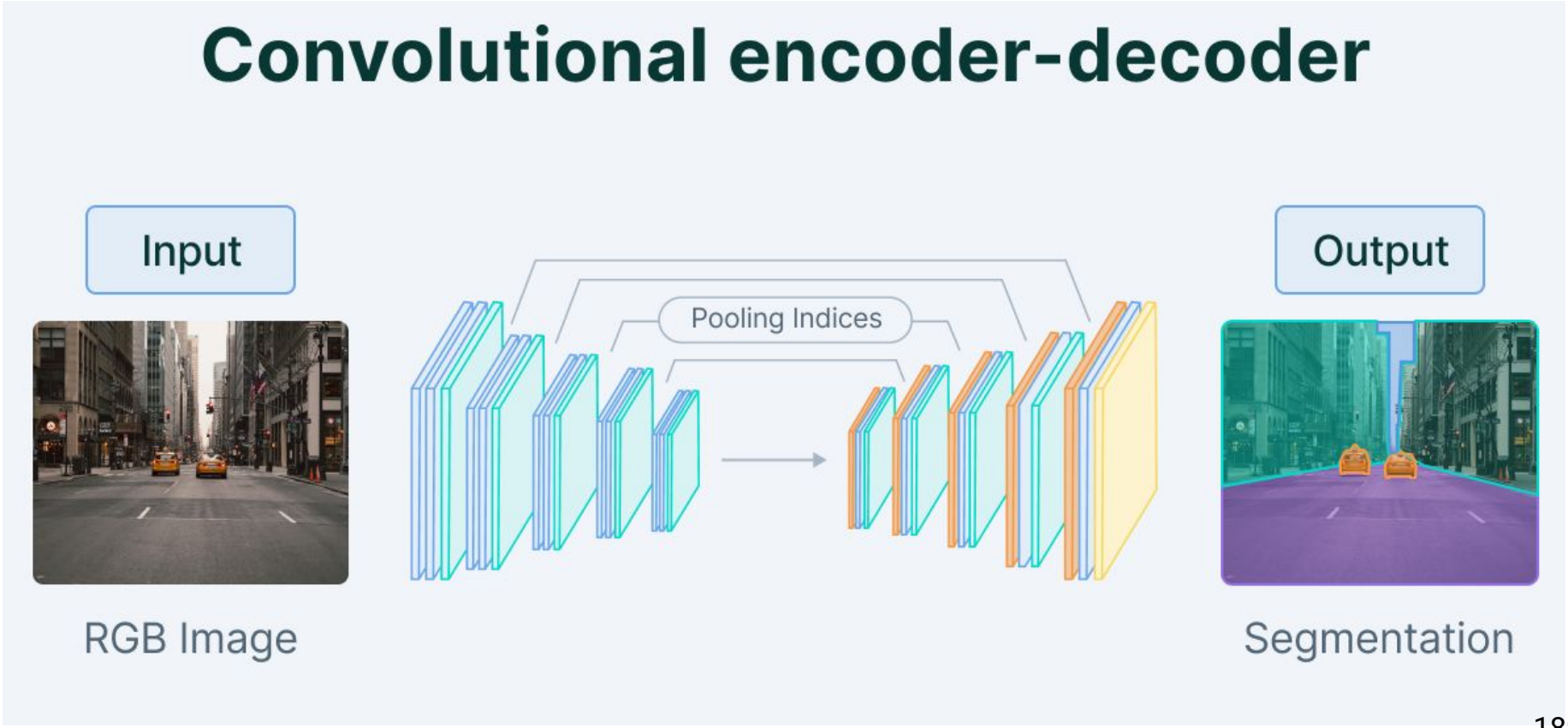
3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	1	1	1	1	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	1	1	3	3	3	5	5	5	5	5	5	5
5	5	3	3	3	3	1	1	3	3	5	5	5	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	4	4	4	4	4	5	5	5
4	4	4	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	4	4	4	4	4	4	4

Semantic Labels

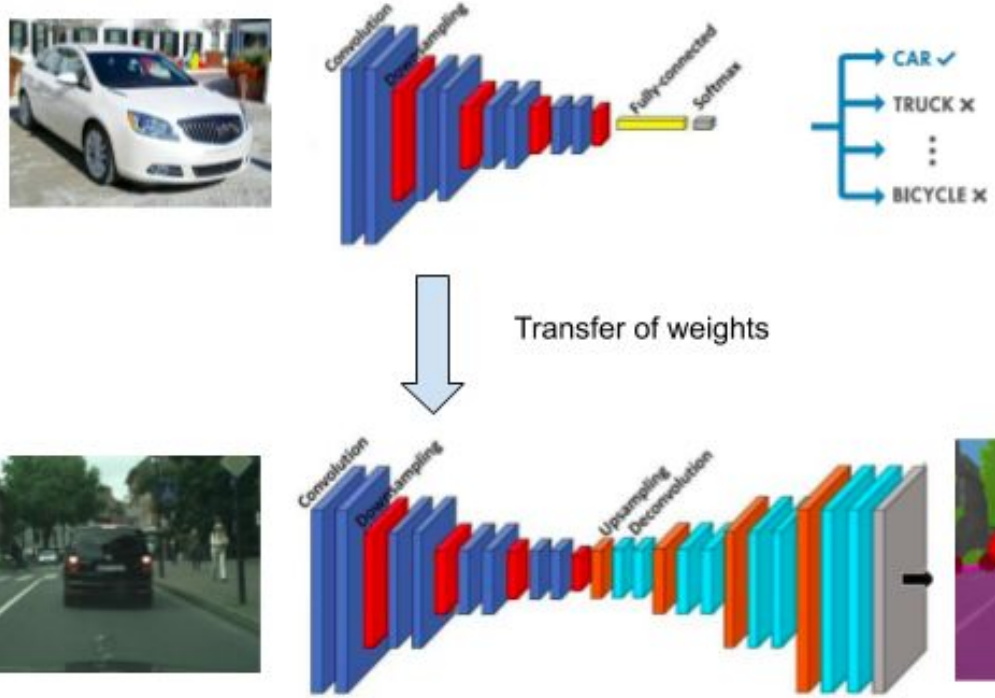
Annotations



Basic structure

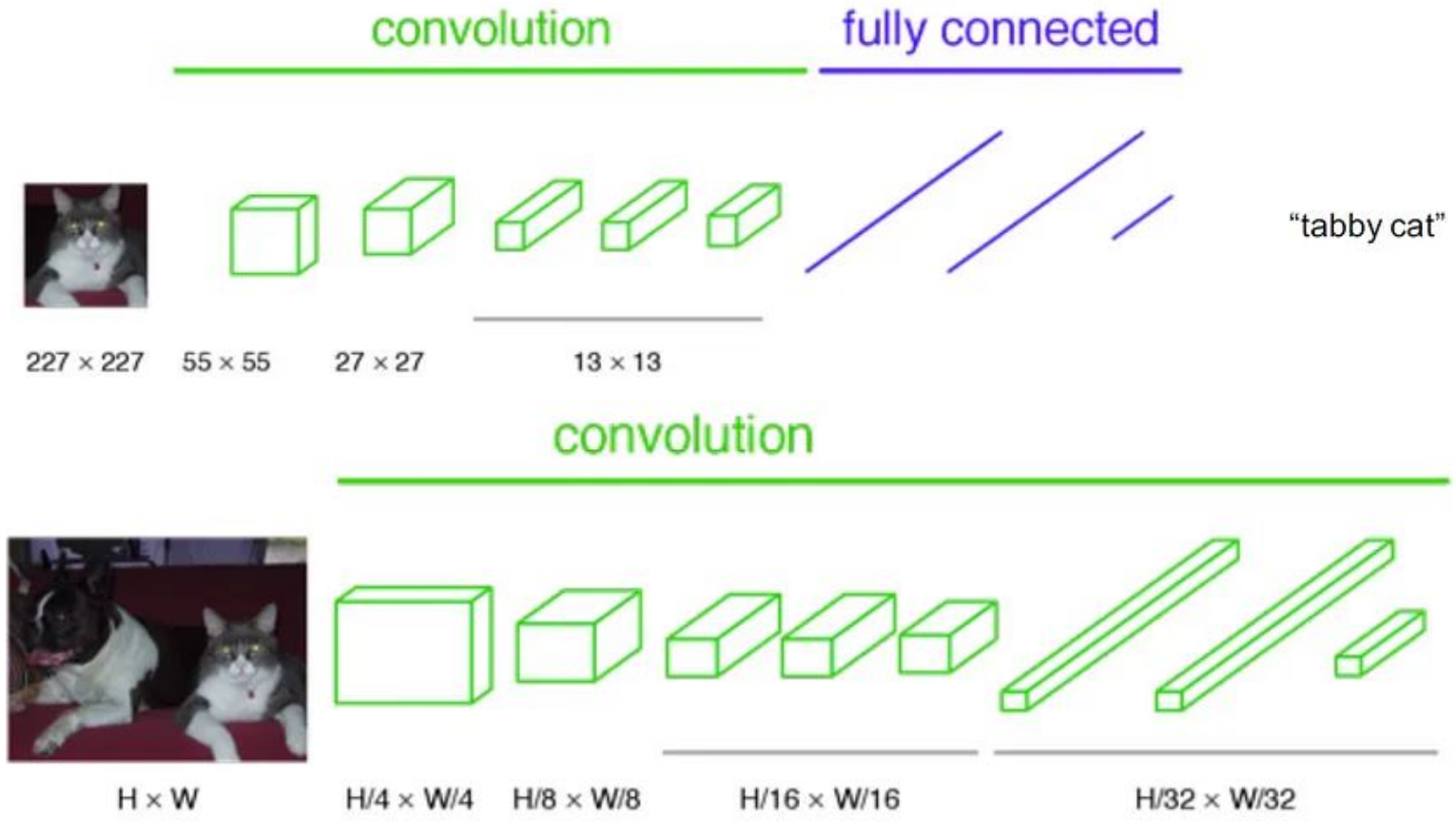


Transfer learning

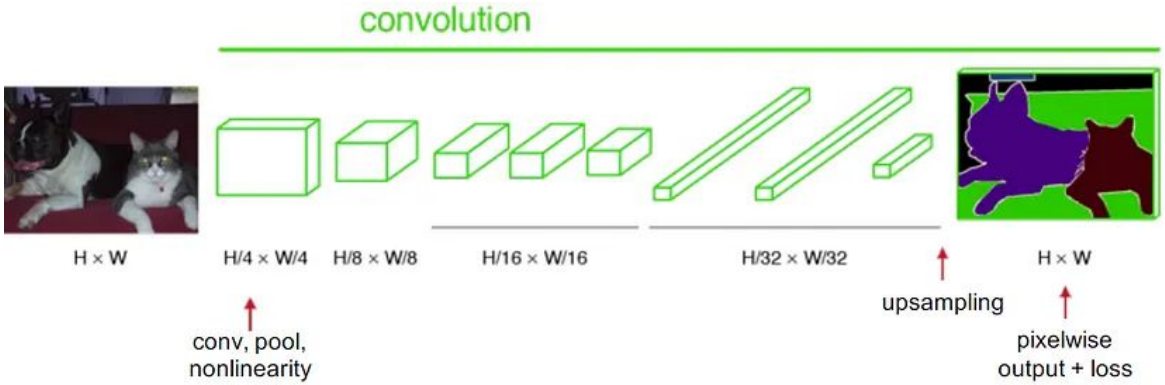


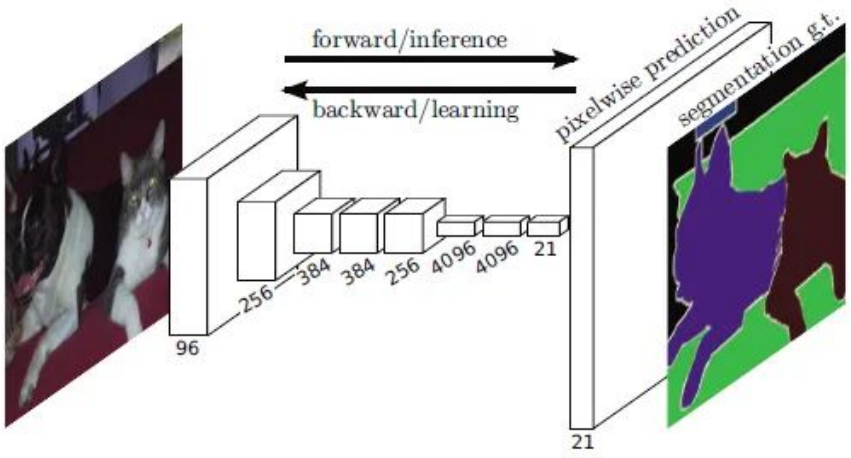
Fully Convolutional Network

FCN: From Image Classification...

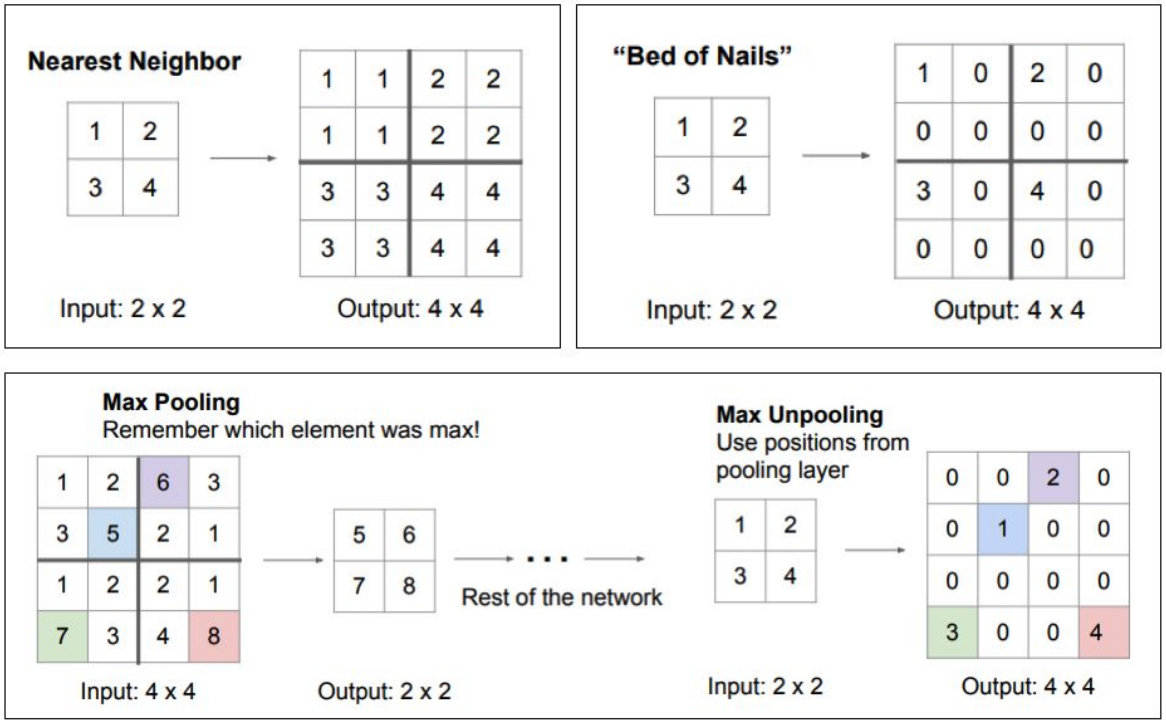


FCN: ...to Semantic Segmentation

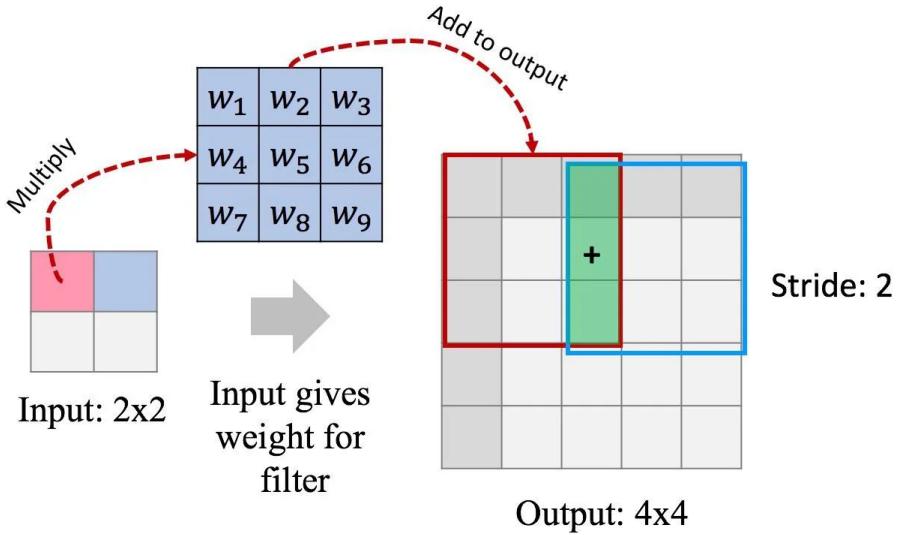


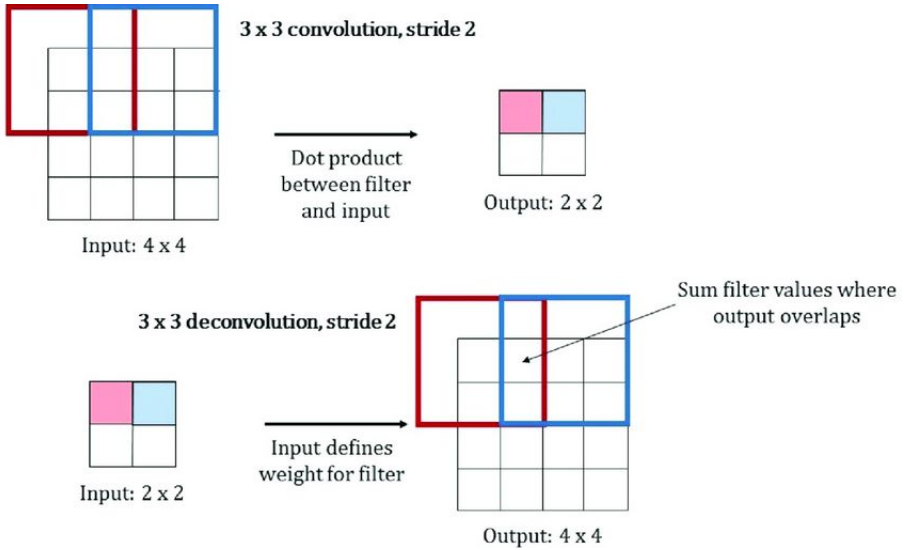


Upsampling/Unpooling methods

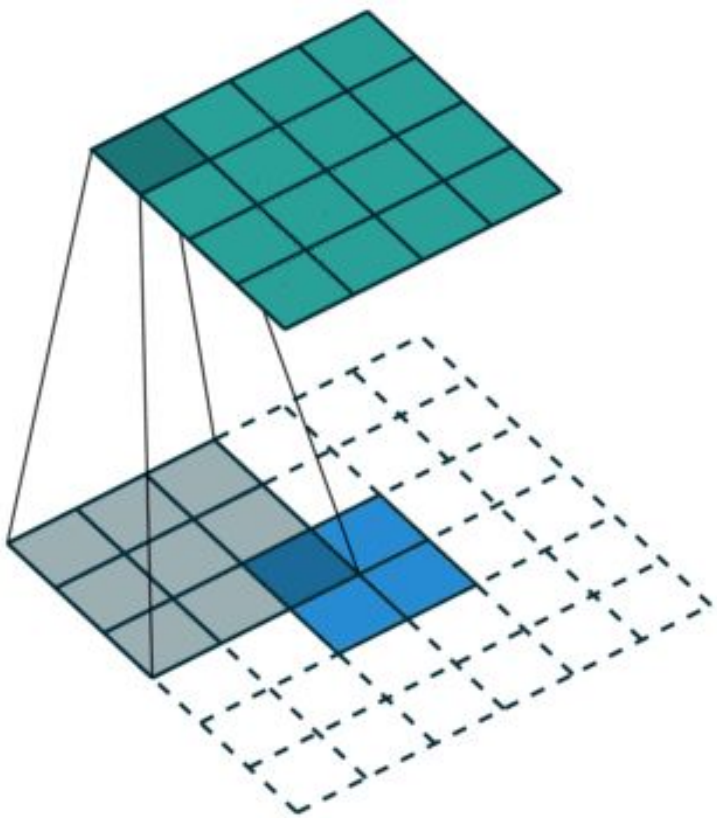


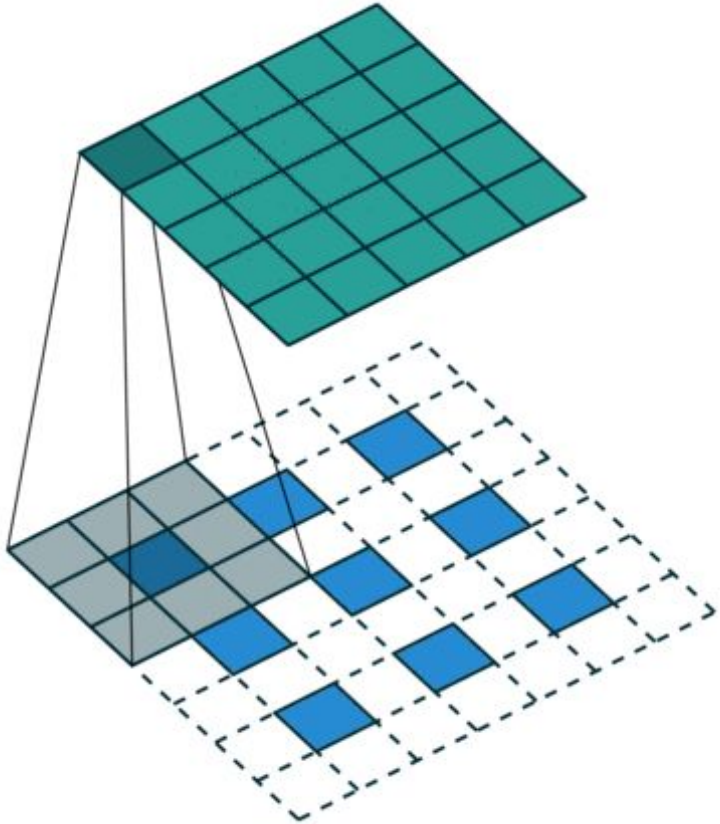
Transposed Convolution (Deconvolution)



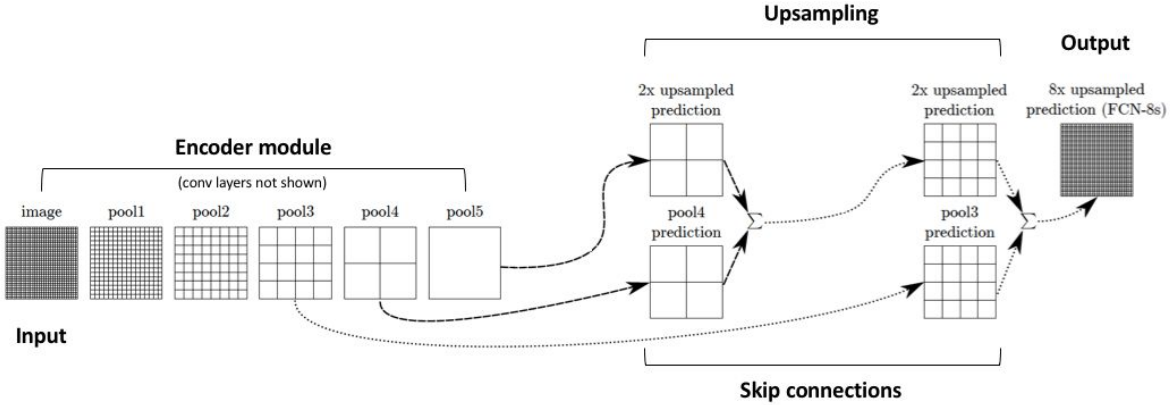


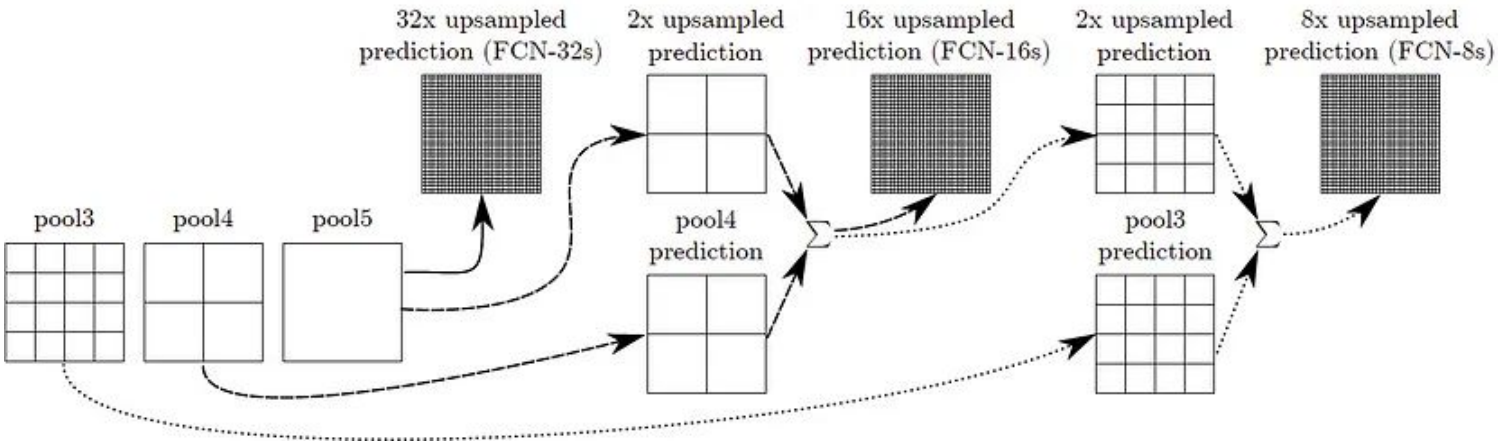
Transposed Convolution (Deconvolution)



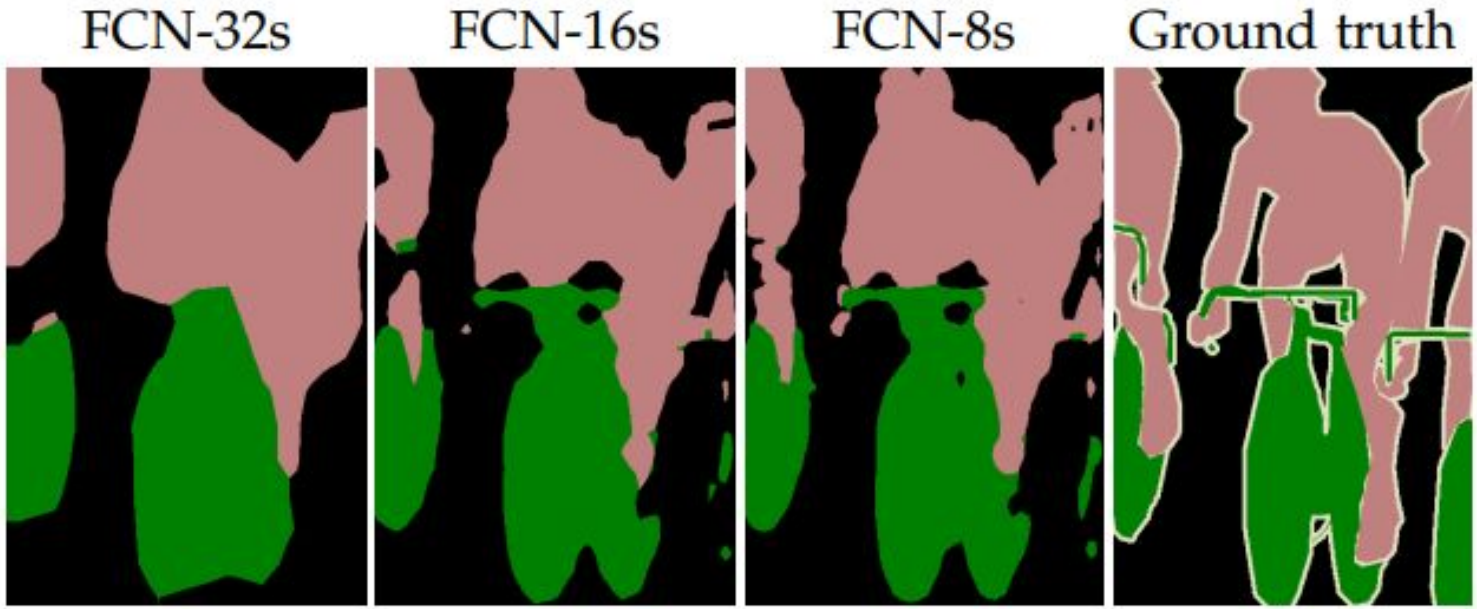


Skip connection



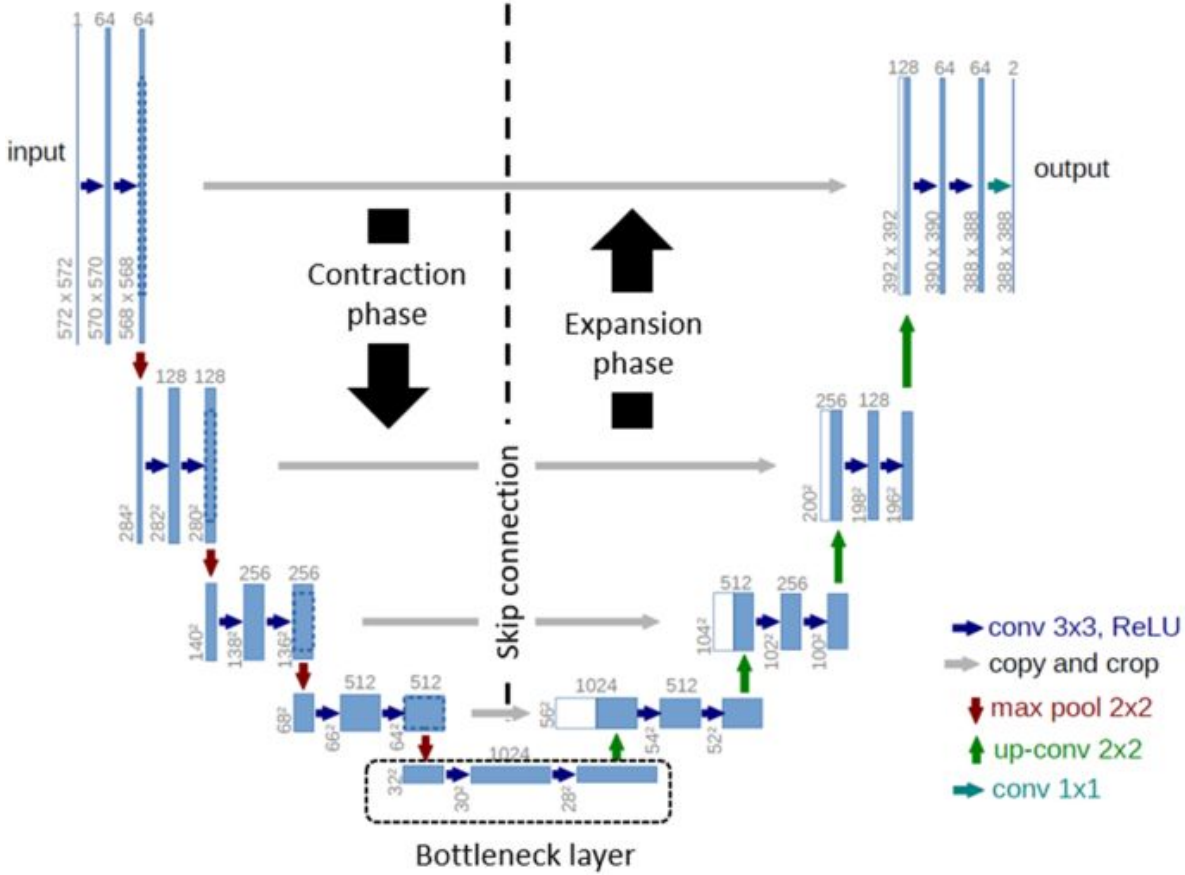


Result



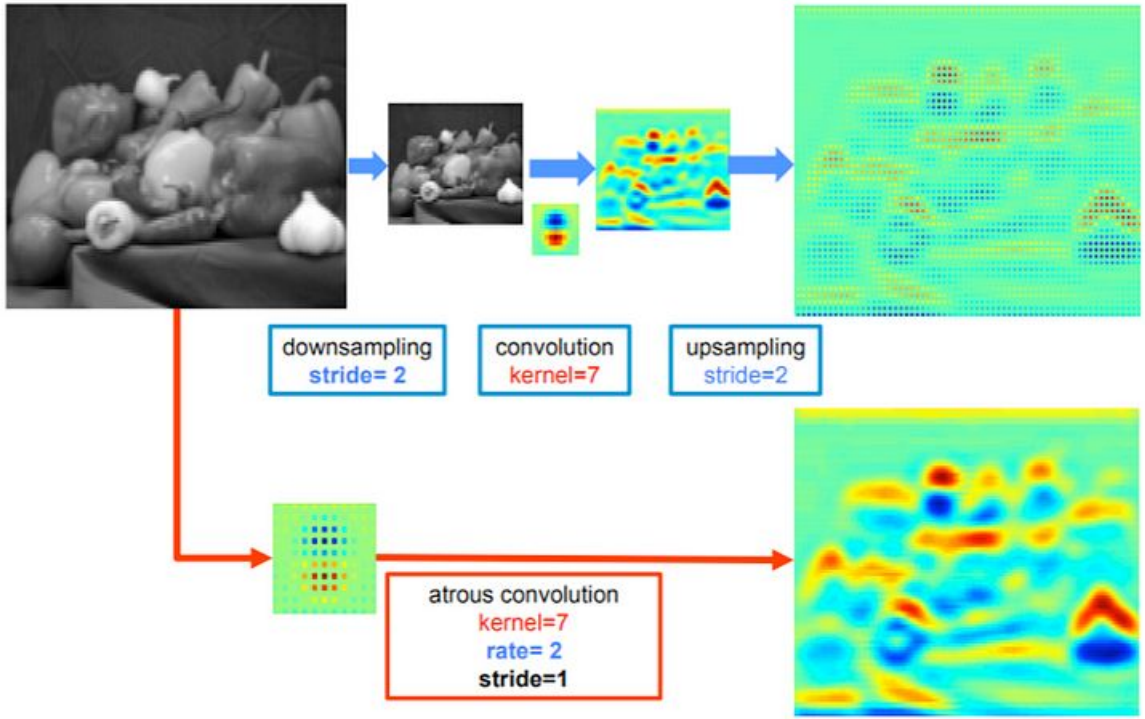
Refining fully convolutional networks by fusing information from layers with different strides improves spatial detail. The first three images show the output from our 32, 16, and 8 pixel stride nets

U-Net

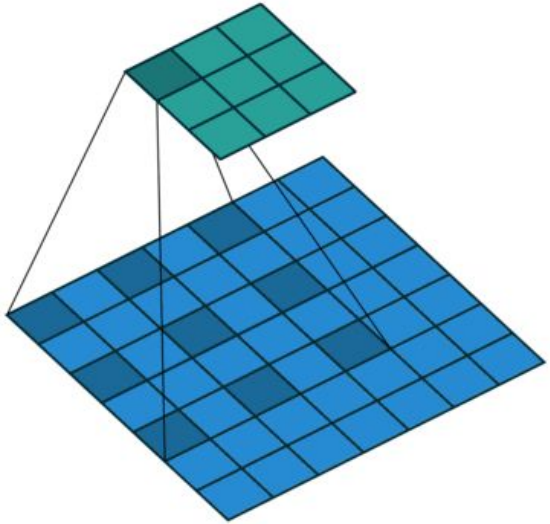

input
output
Contraction phase
Expansion phase
Skip connection
Bottleneck layer
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

Deeplab

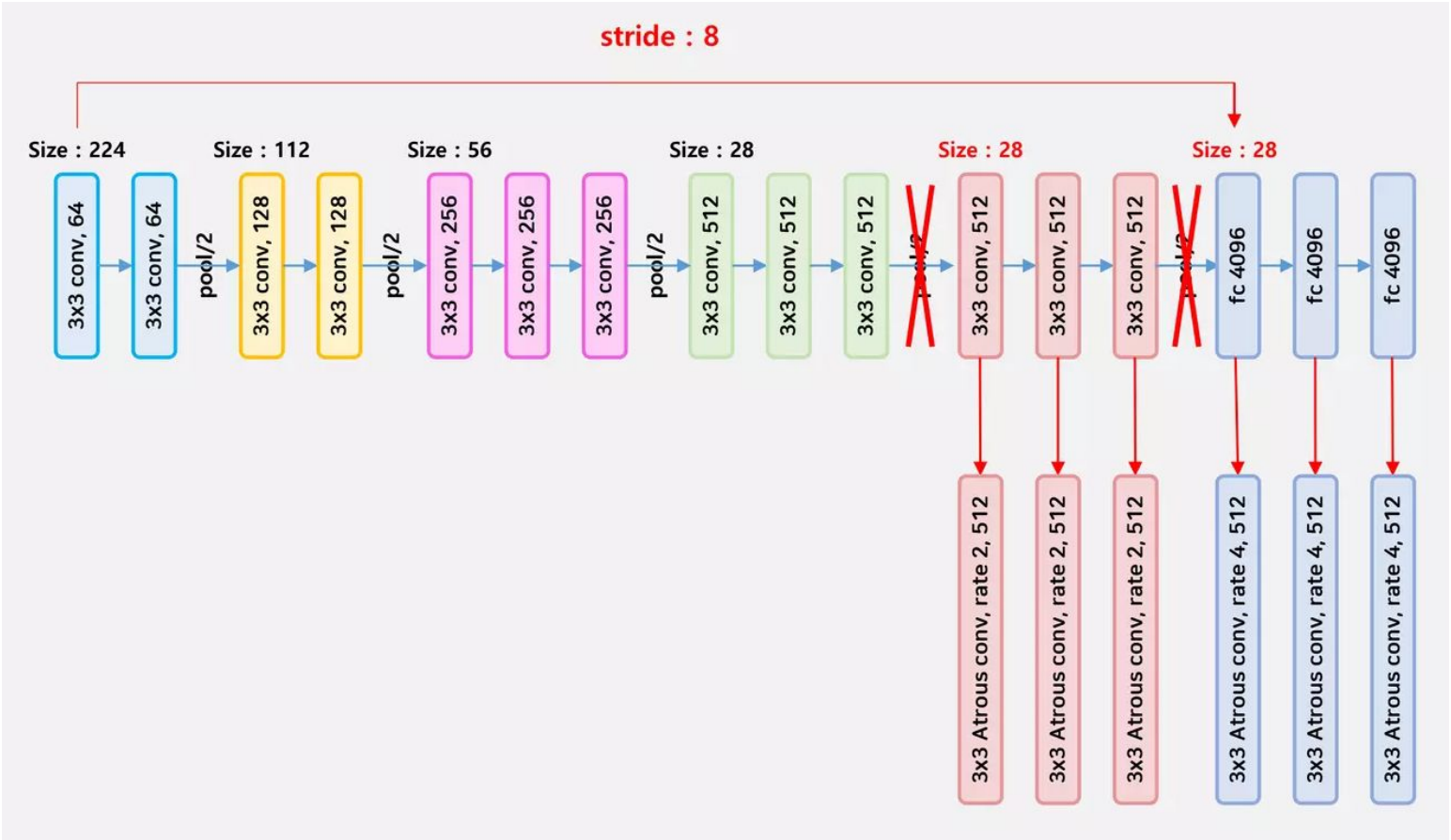
Deeplab v1: DCNN



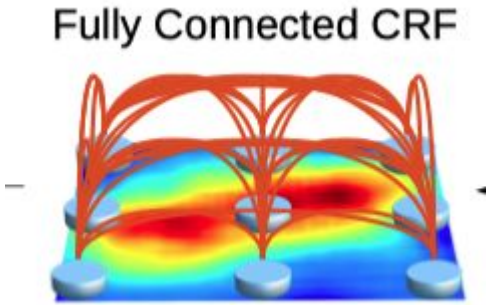
Dilated/Atrous Convolution

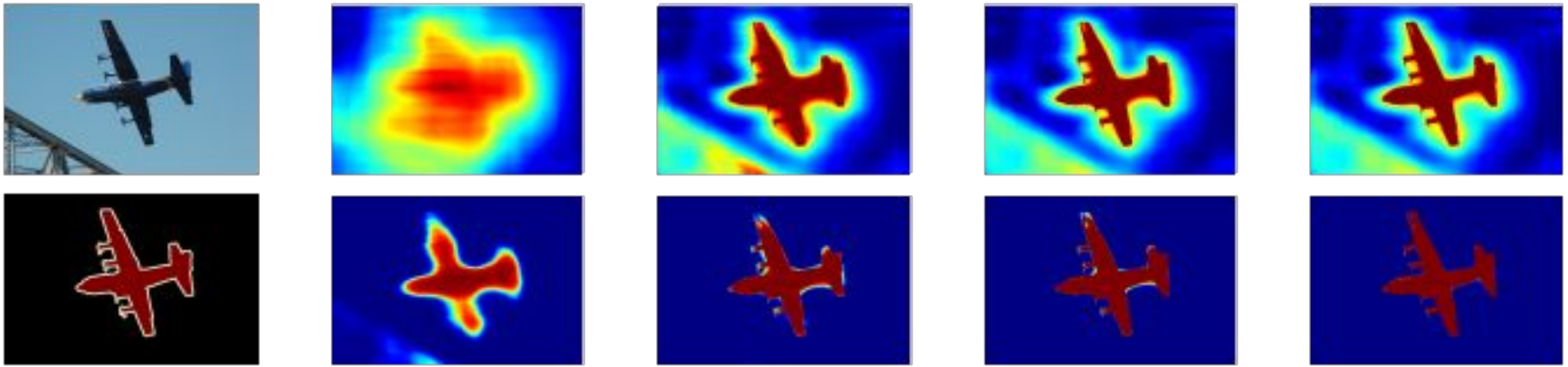


Deeplab v1: DCNN



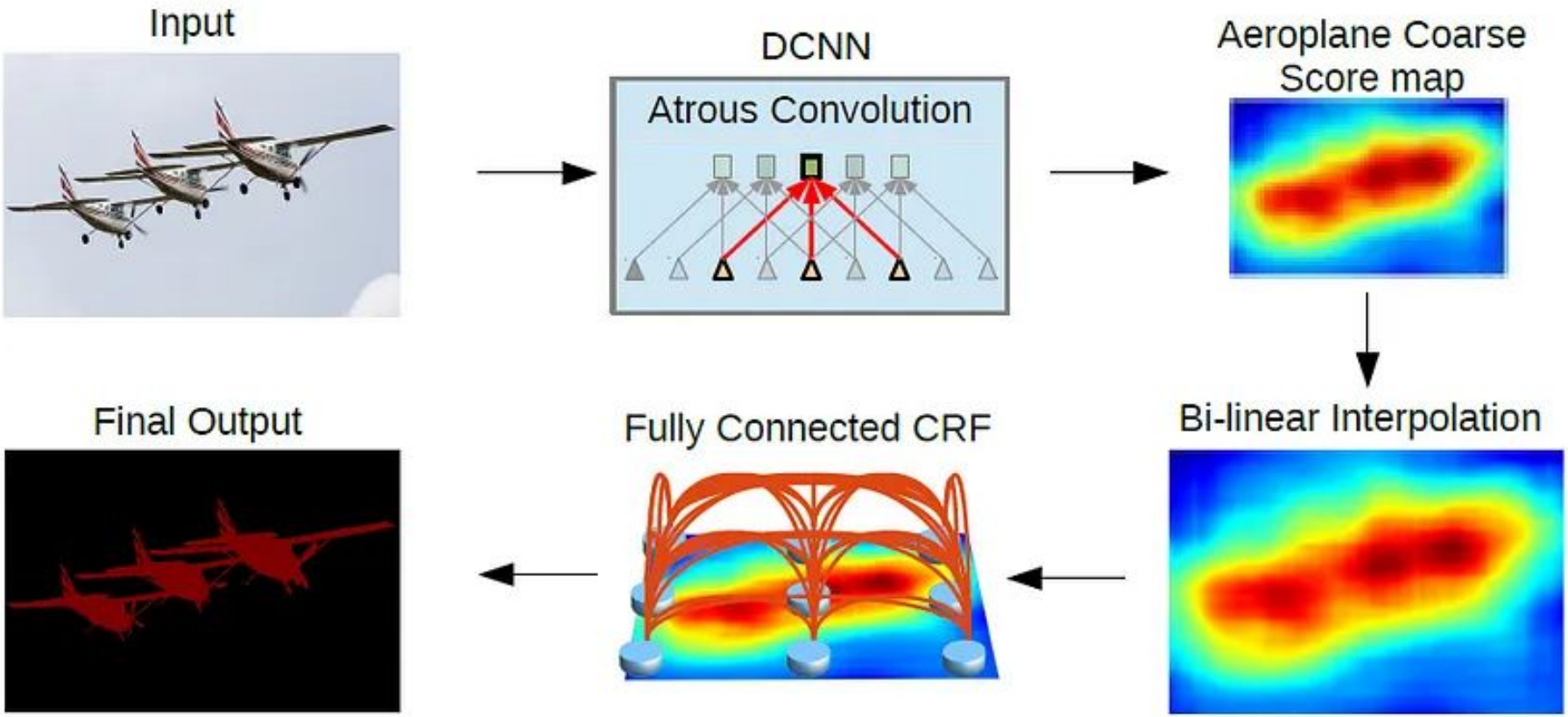
Deeplab v1: Fully Connected Conditional Random Fields



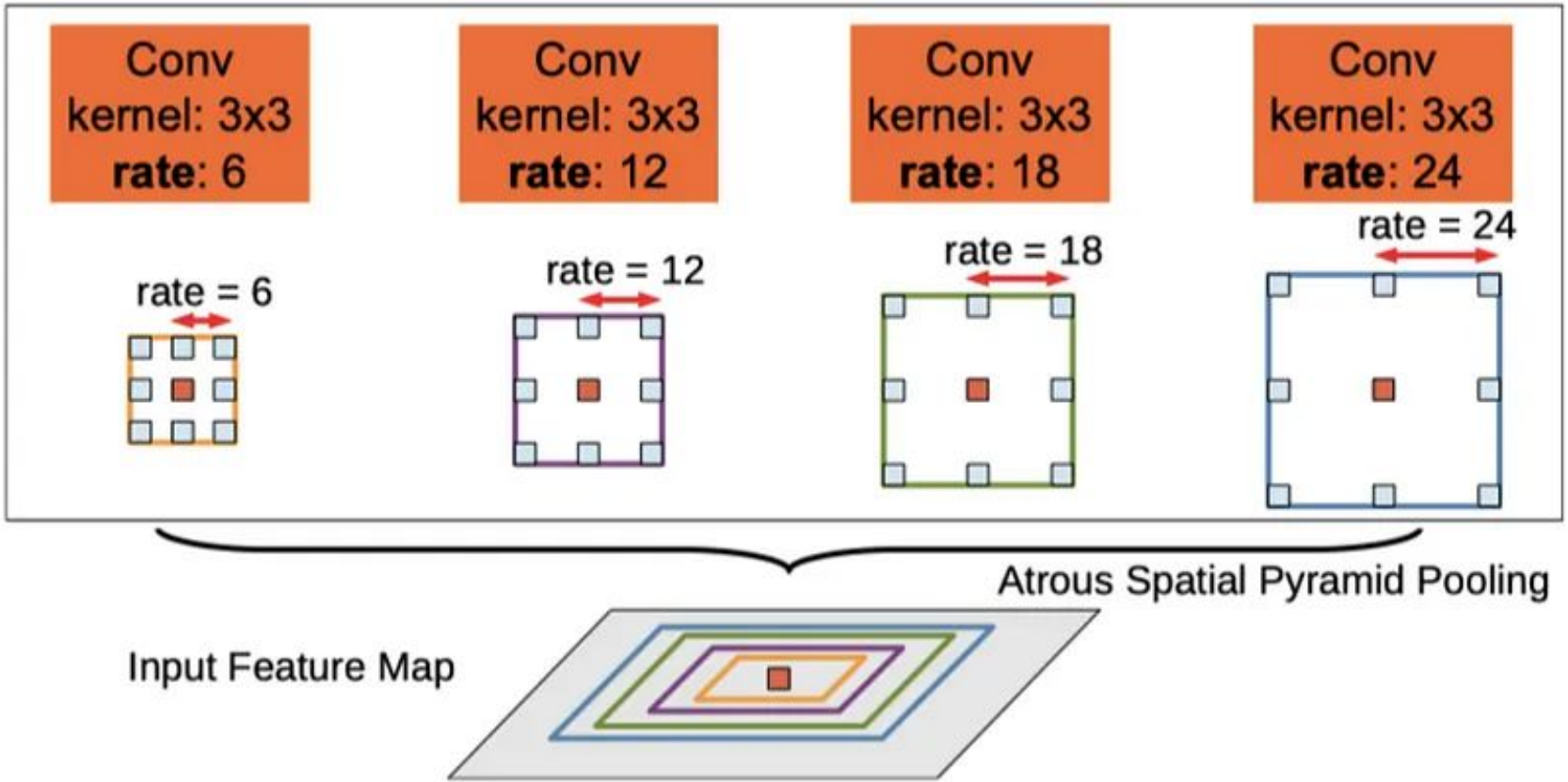


Image/G.T. | DCNN output | CRF Iteration 1 | CRF Iteration 2 | CRF Iteration 10

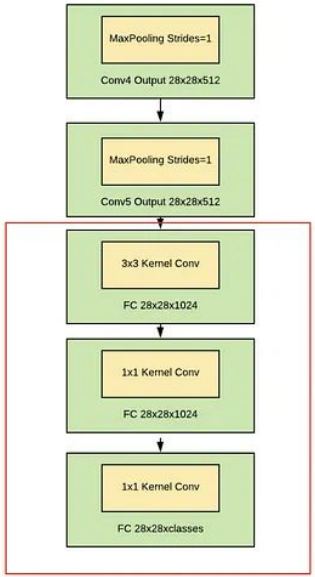
Deeplab v1: Pipeline



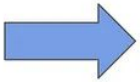
Deeplab v2: Atrous Spatial Pyramid Pooling

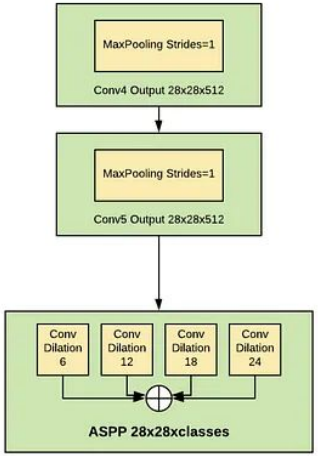


Deeplab v2: Difference from v1

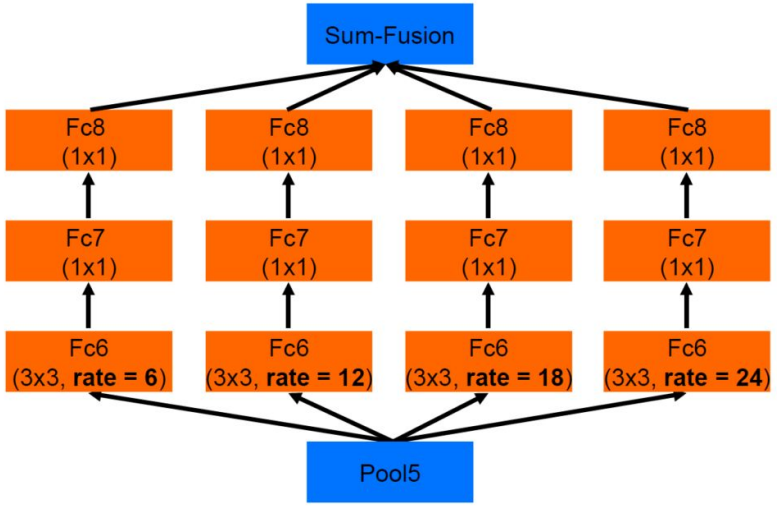


DeepLabV1

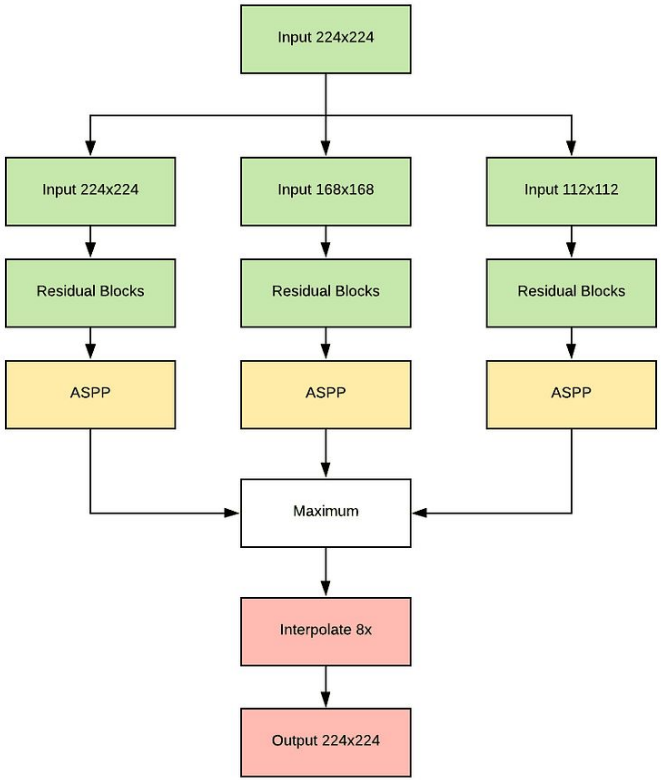




DeepLabV2

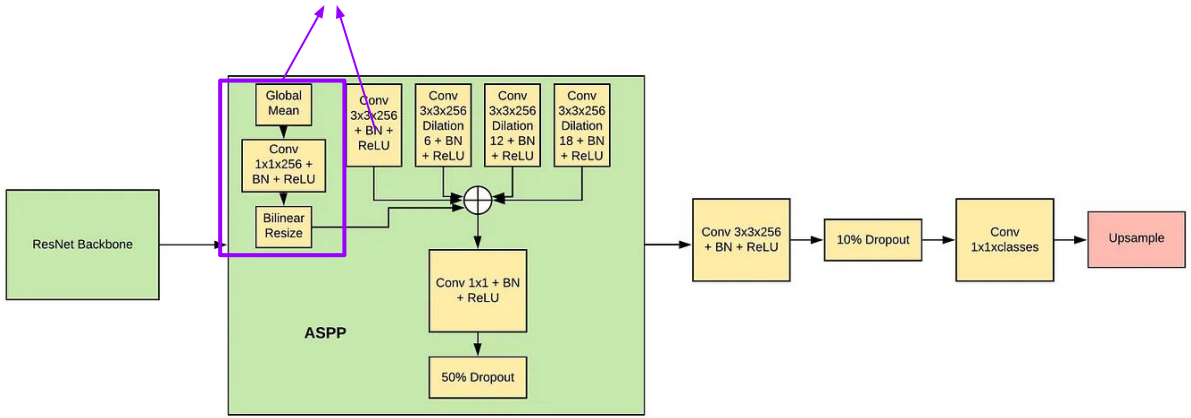


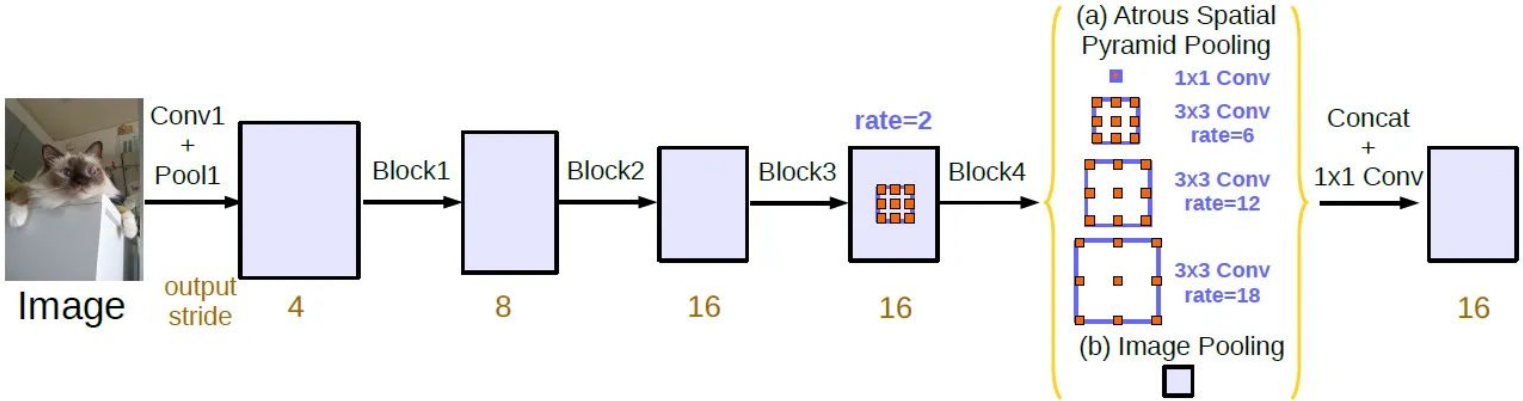
Deeplab v2: Difference from v1



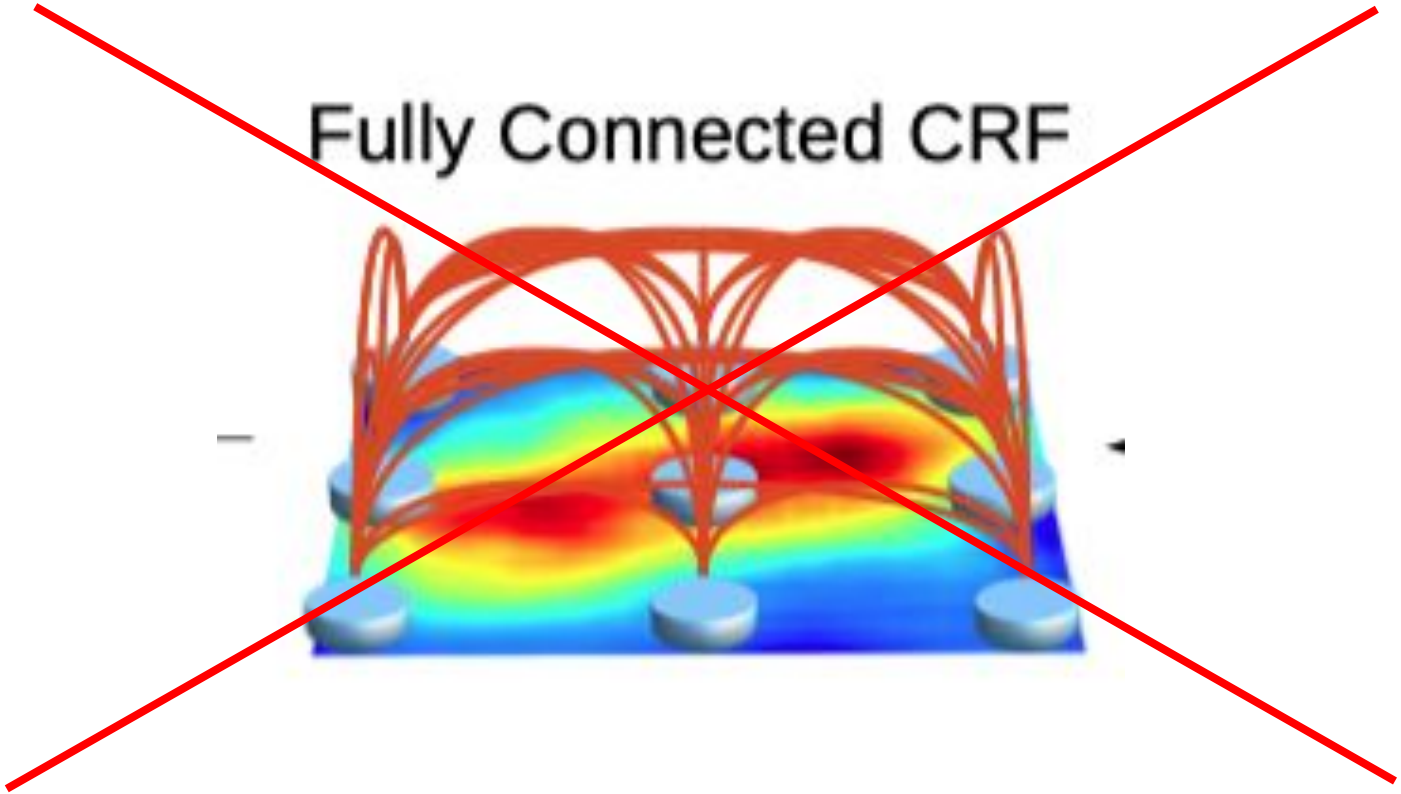
Deeplab v3: Difference from v2

Differences from Deeplab v2



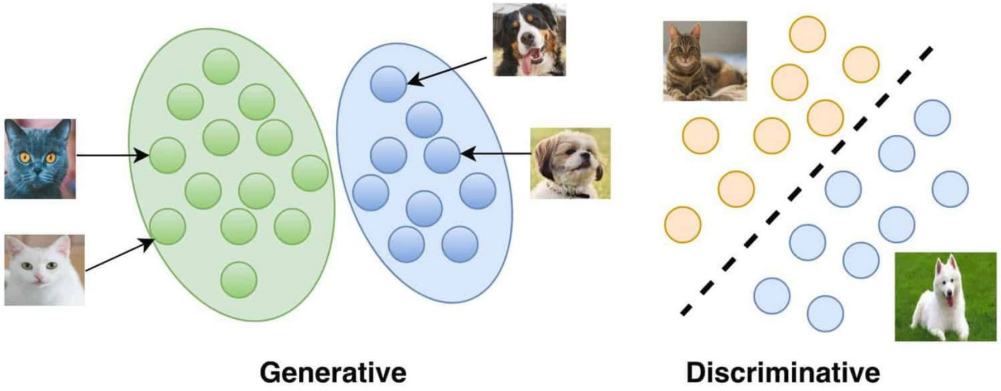


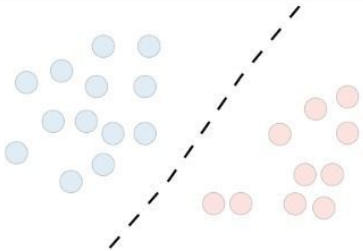
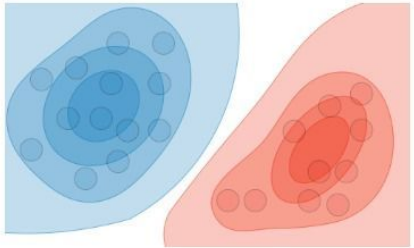
Deeplab v3: Difference from v2



Generative adversarial networks

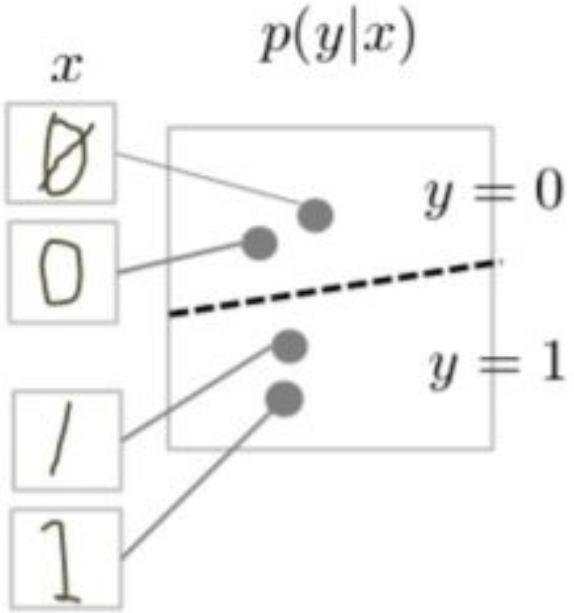
Generative model vs Discriminative model



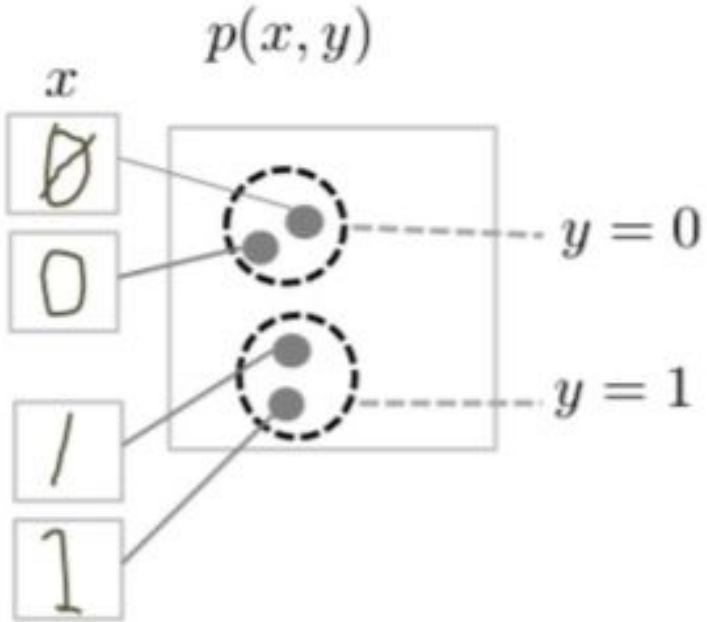
	Discriminative model	Generative model
Goal	Directly estimate $P(y\|x)$	Estimate $P(x\|y)$ to then deduce $P(y\|x)$
What's learned	Decision boundary	Probability distributions of the data
Illustration		
Examples	Regressions, SVMs	GDA, Naive Bayes

Which task is harder?

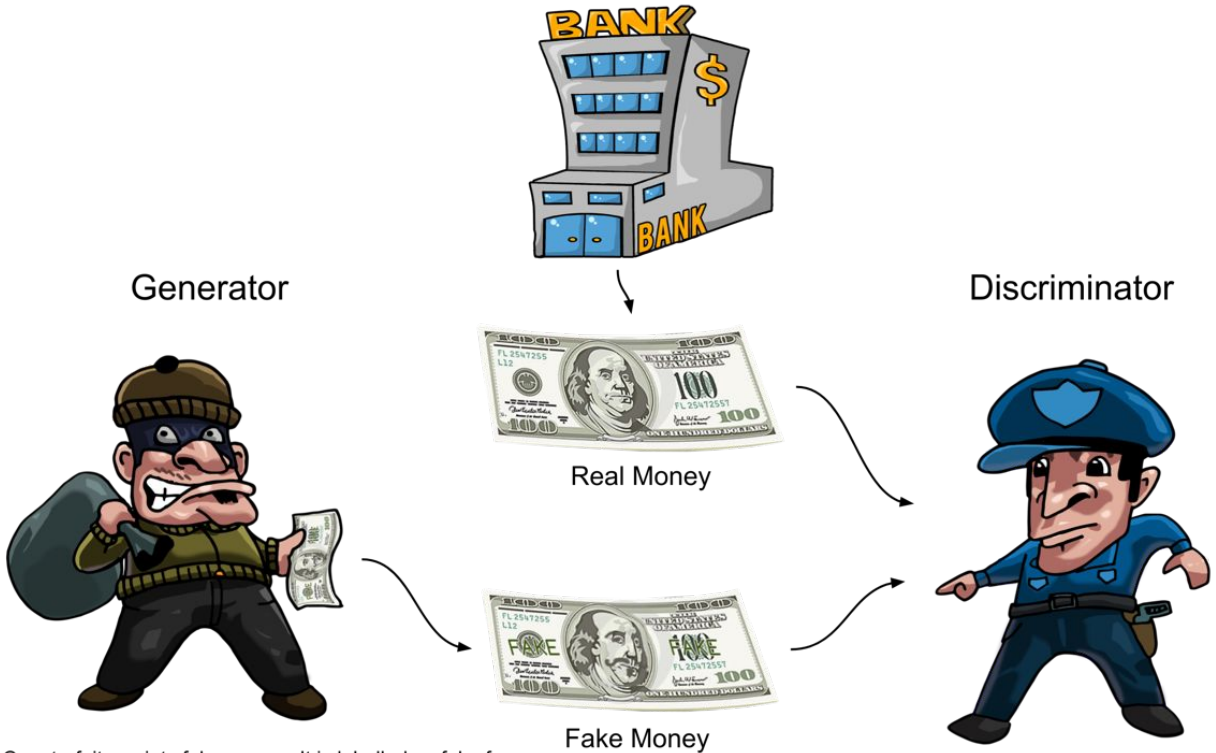
- Discriminative Model



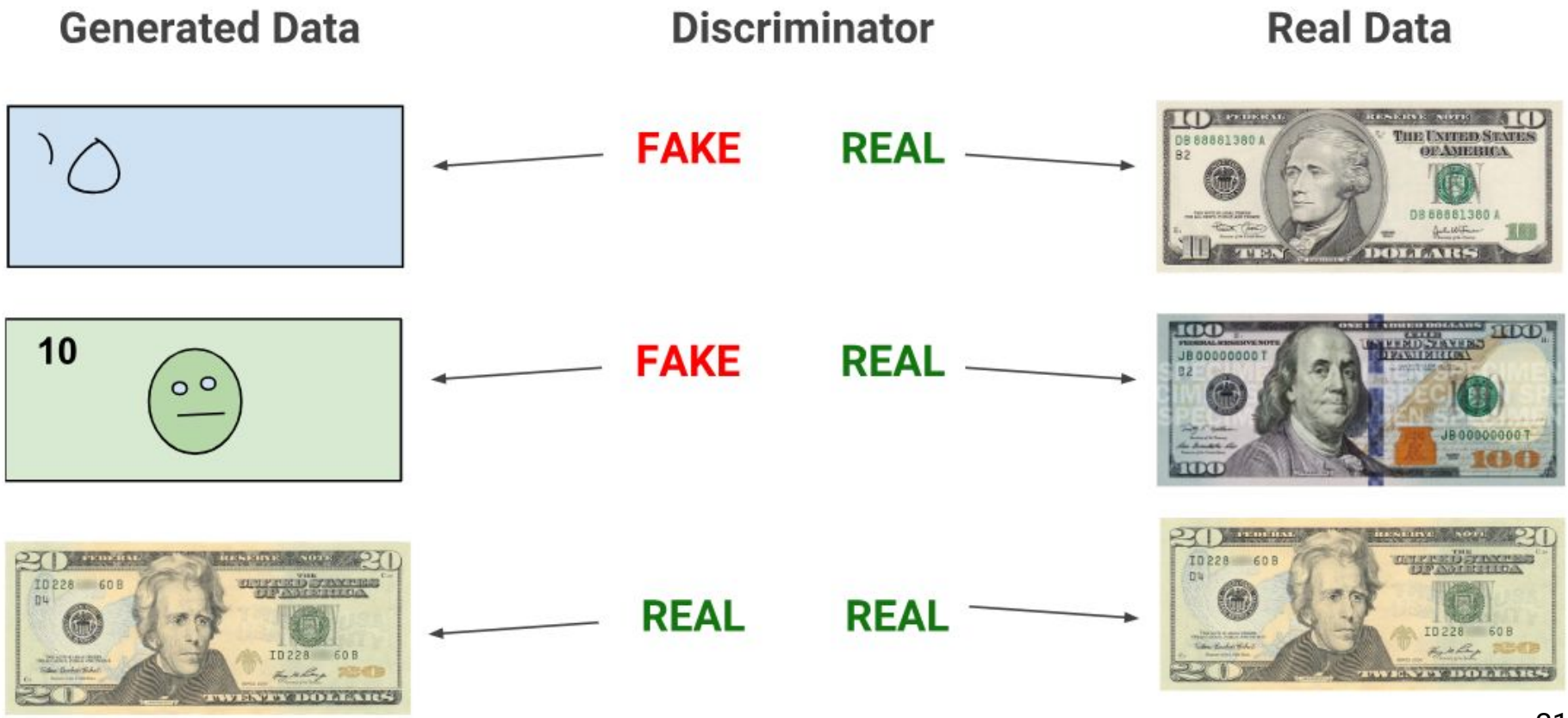
- Generative Model



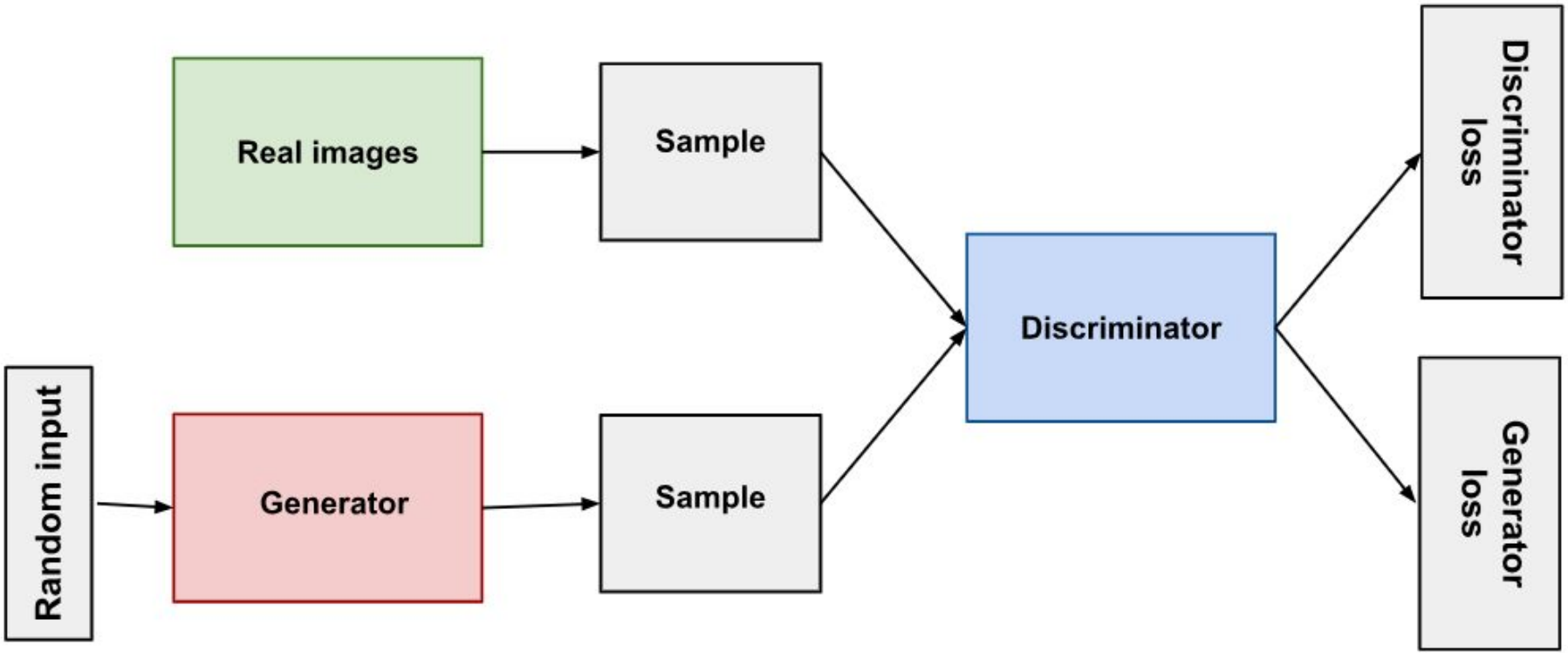
Overview of GAN structure



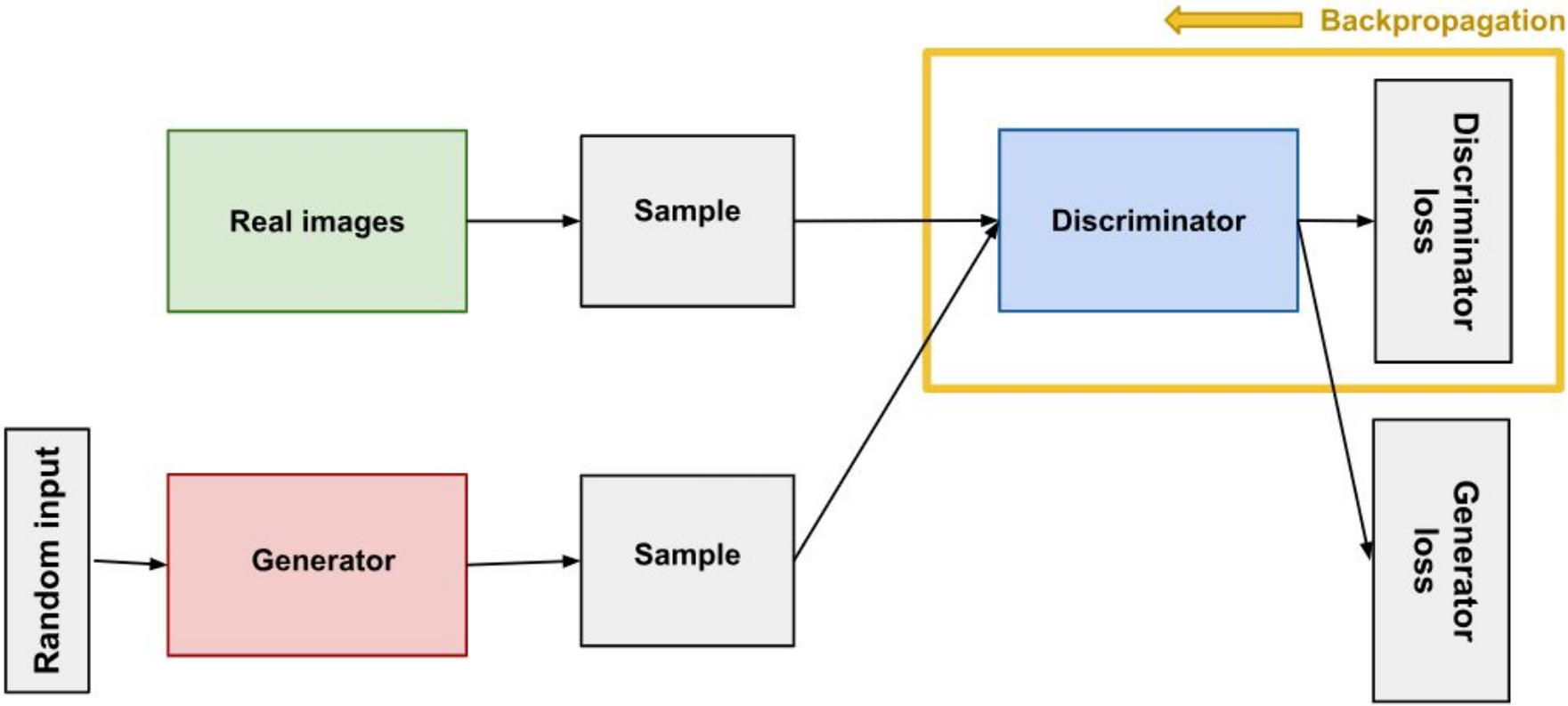
Overview of GAN structure



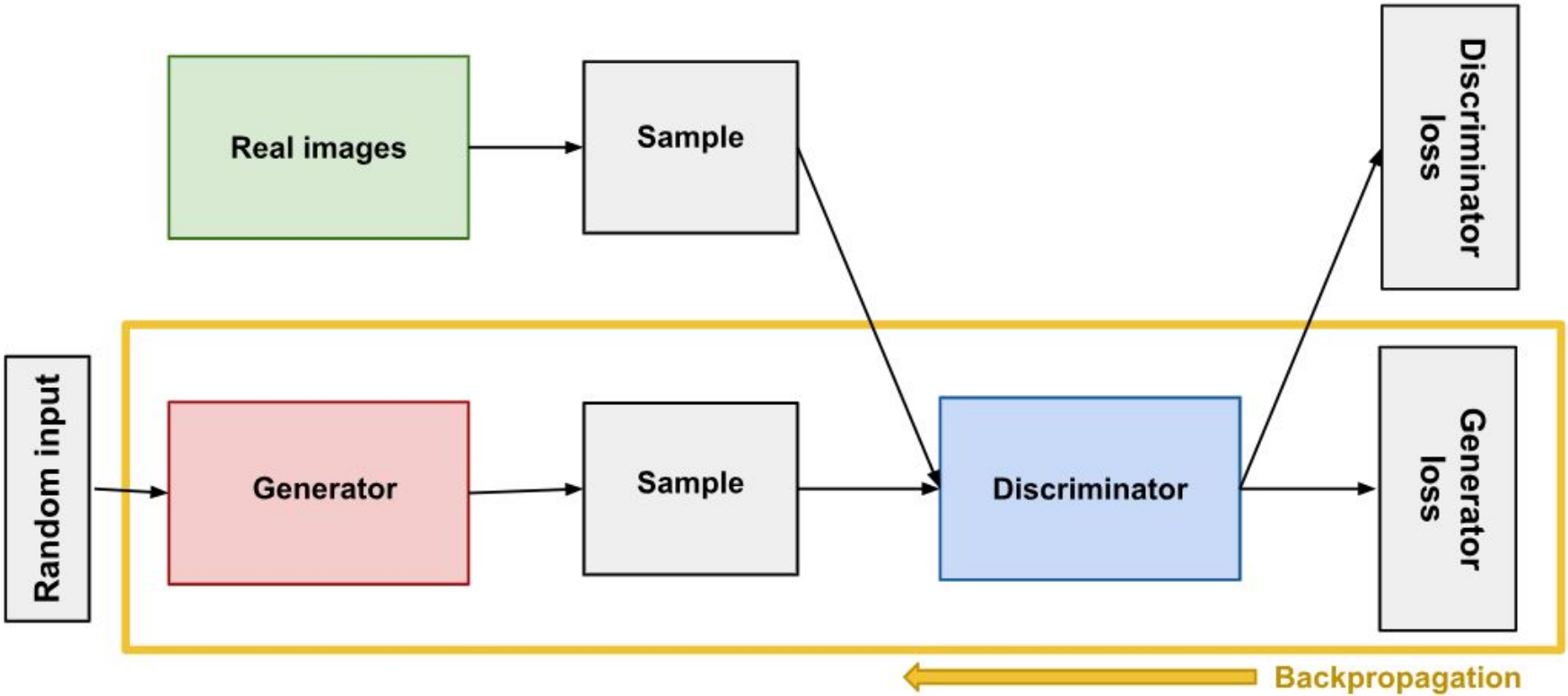
Overview of GAN structure



Discriminator's training



Generator's training

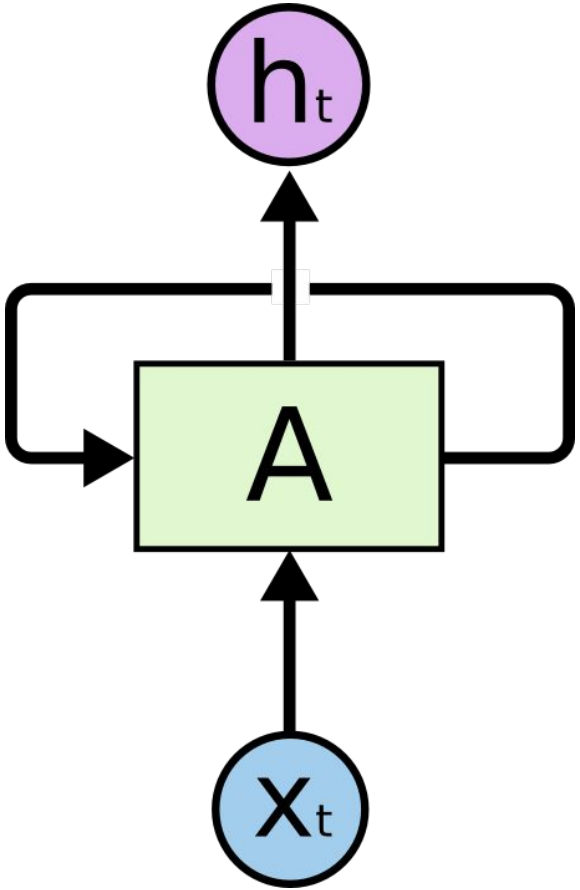


GAN's training

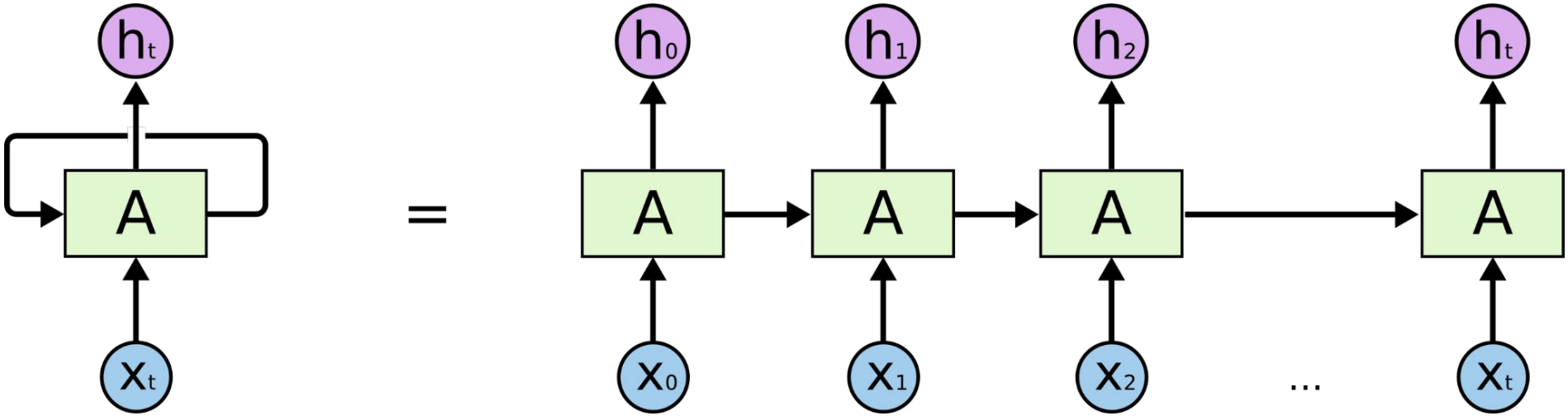
- Discriminator được huấn luyện trong 1 hoặc 1 vài epochs. Trong giai đoạn này generator được giữ cố định
- Generator được huấn luyện trong 1 hoặc 1 vài epochs. Trong giai đoạn này discriminator được giữ cố định
- Lặp lại 2 bước trên

Recurrent Neural Network

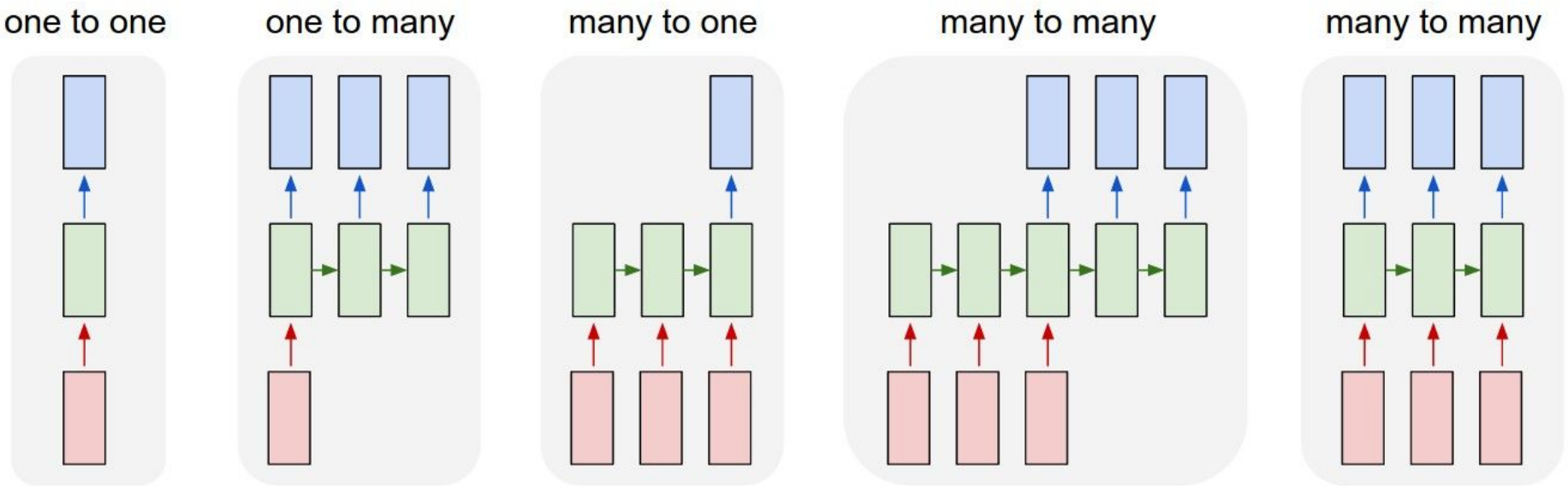
Recurrent Neural Network



Recurrent Neural Network

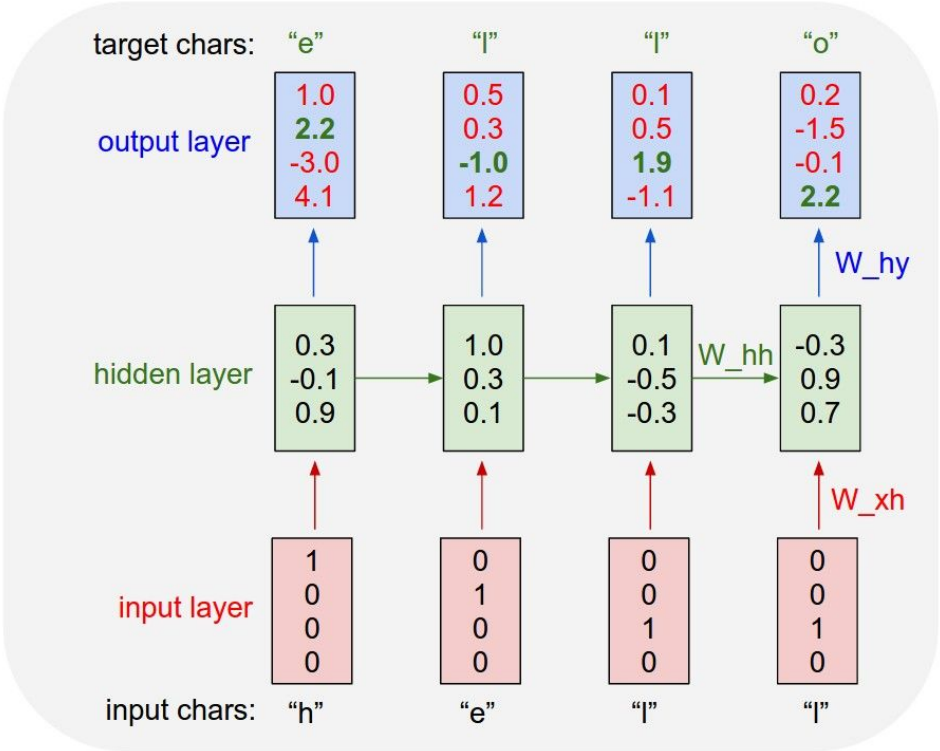


Recurrent Neural Network



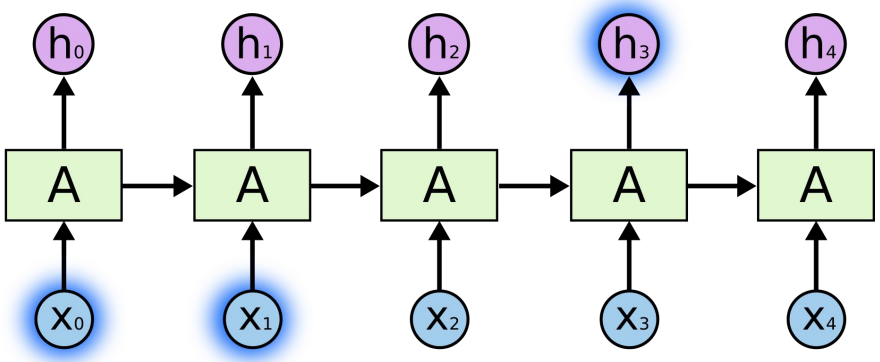
Example

```
rnn = RNN()
y = rnn.step(x) # x is an input vector, y is the RNN's output vector
```

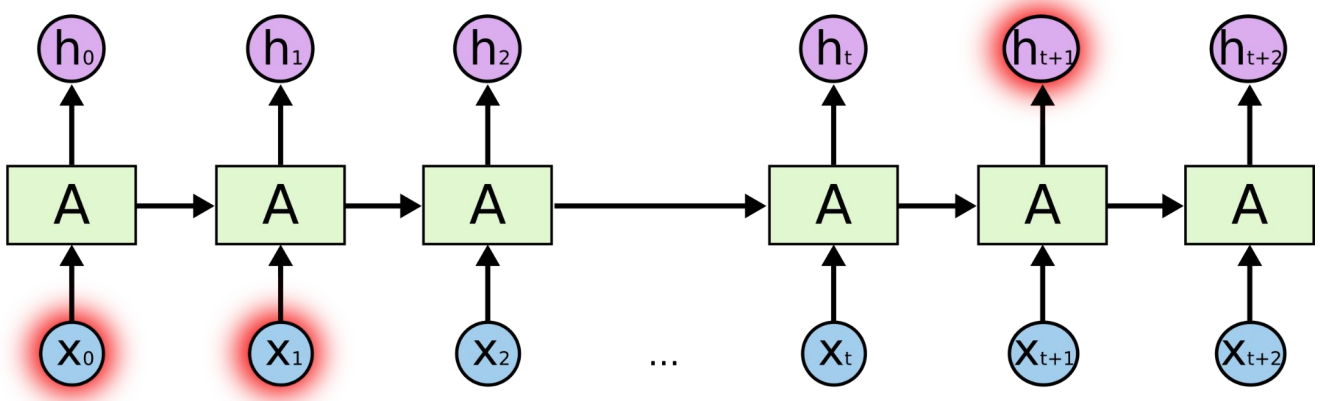



```
class RNN:
  # ...
  def step(self, x):
    # update the hidden state
    self.h = np.tanh(np.dot(self.W_hh, self.h) + np.dot(self.W_xh, x))
    # compute the output vector
    y = np.dot(self.W_hy, self.h)
    return y
```

Problem of RNN

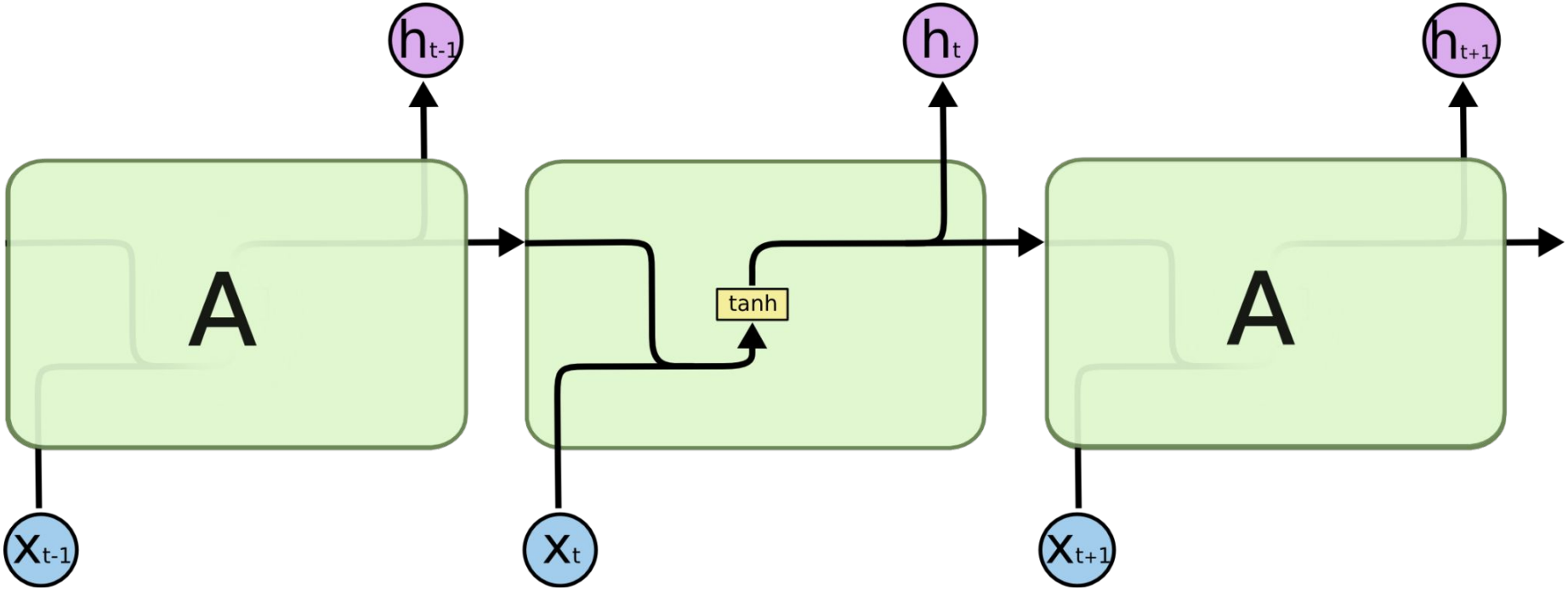


The clouds are in the **sky**

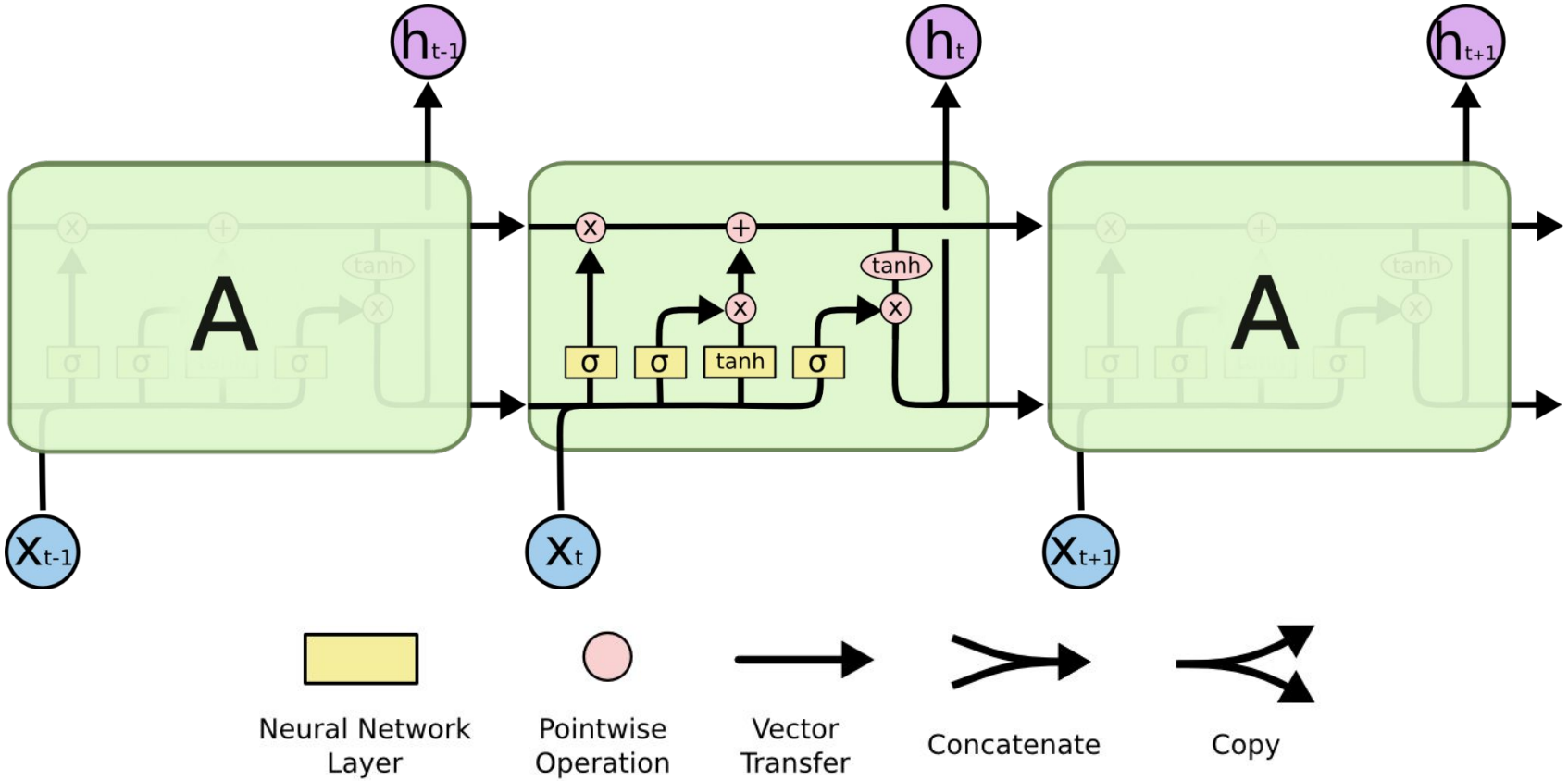


I was born in Vietnam... I could speak **Vietnamese** fluently

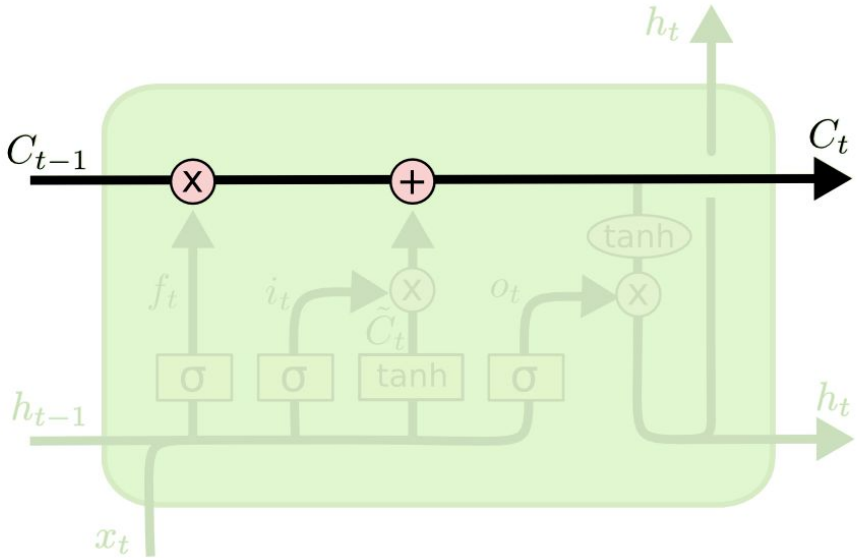
Standard RNN

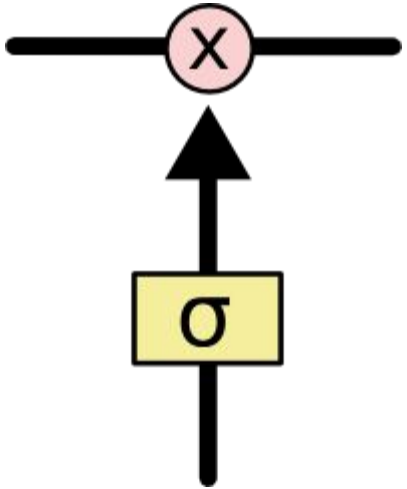


Long Short Term Memory

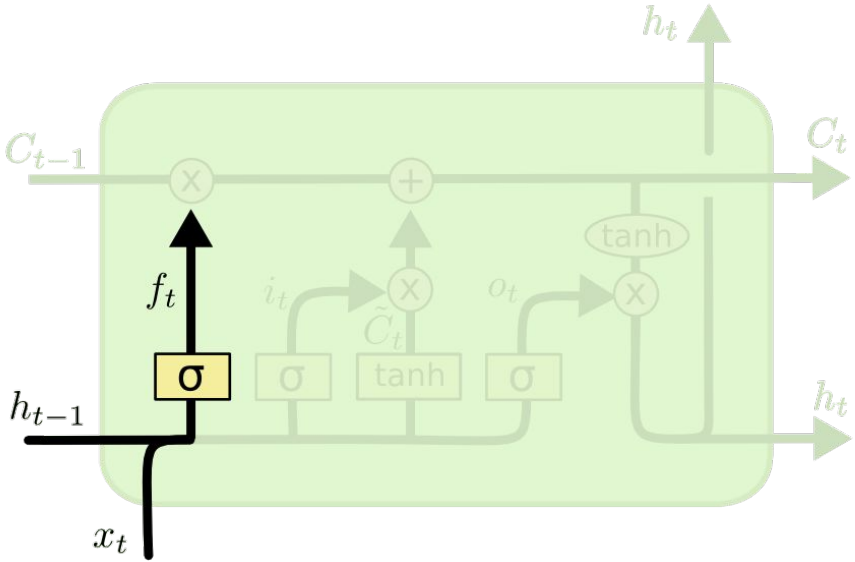


Long Short Term Memory: Idea



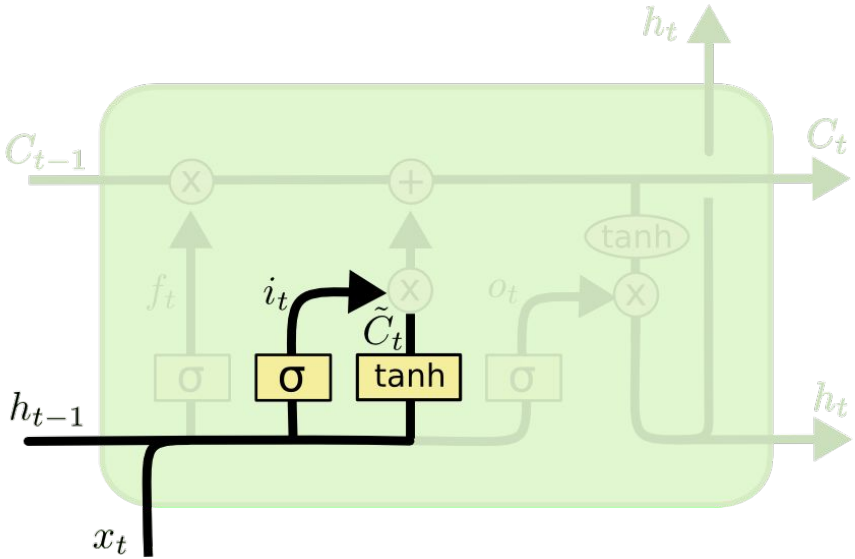


LSTM: Forget gate layer



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

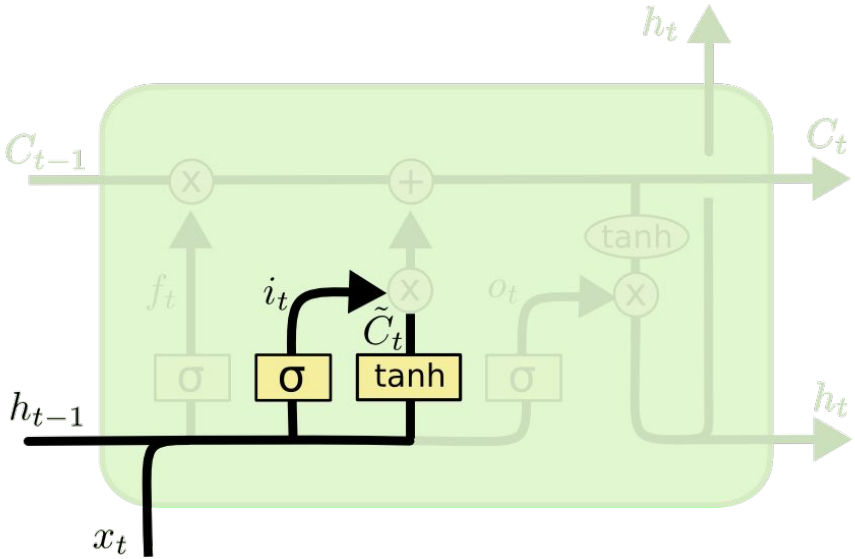
LSTM: Input gate layer



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

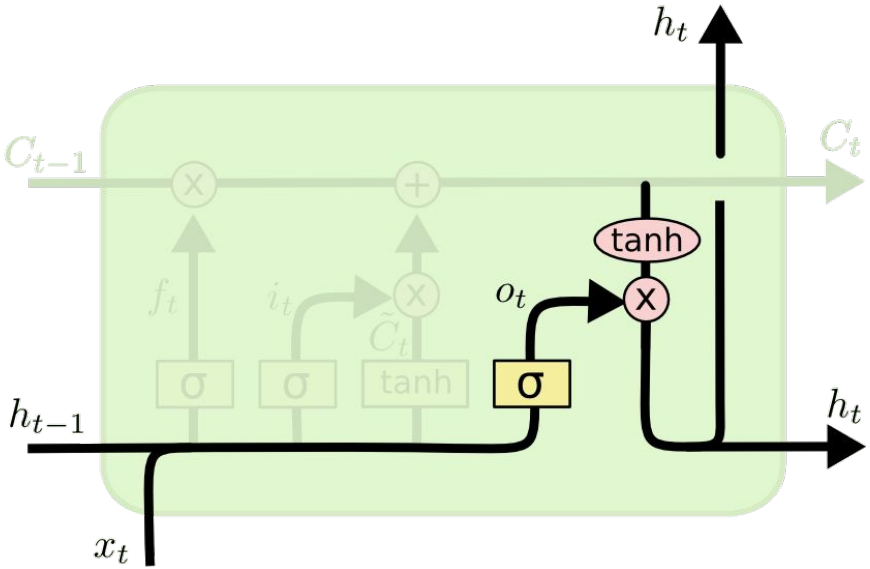
LSTM: New state



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

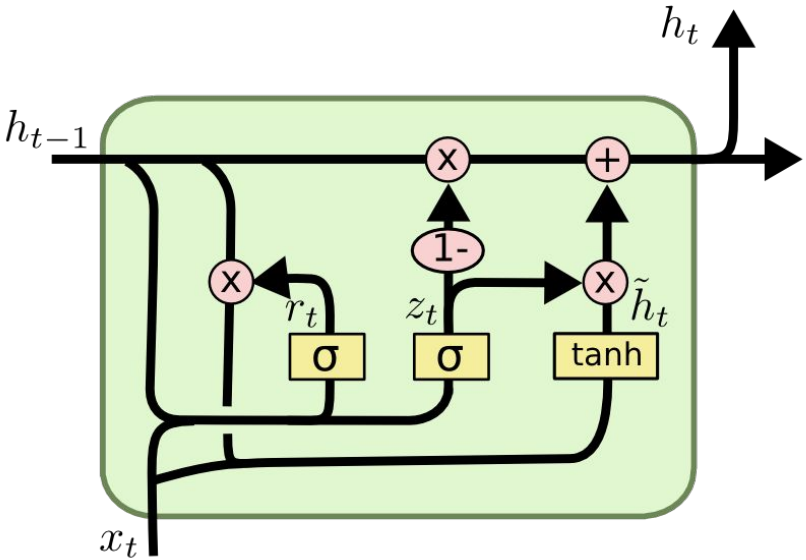
LSTM: Output



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Gated Recurrent Unit: GRU



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

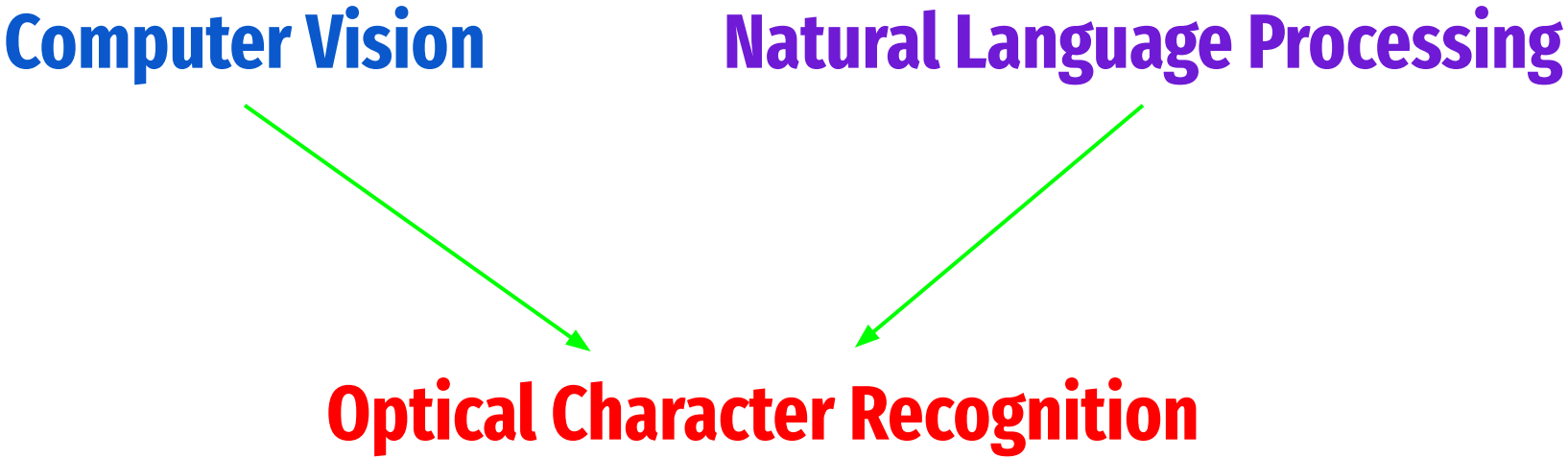
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

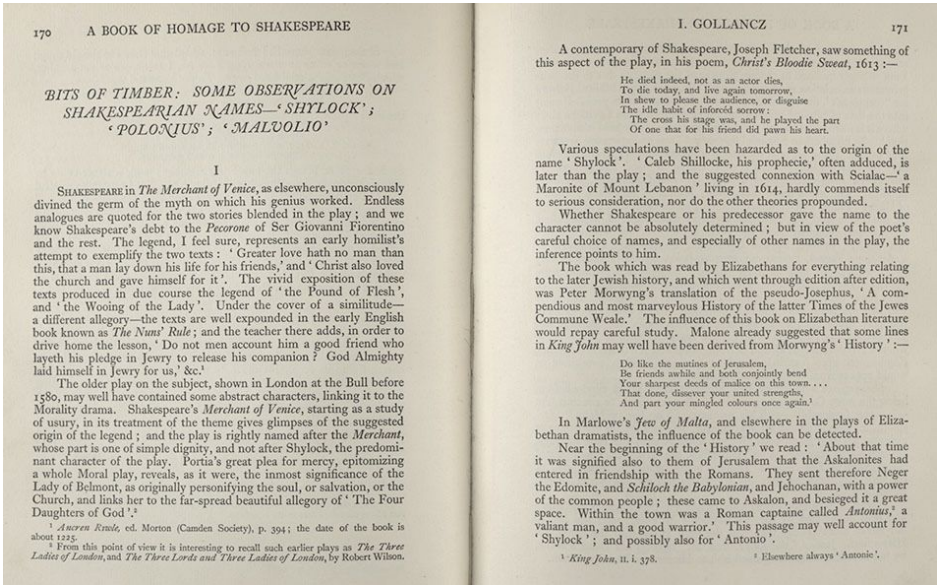
Optical character recognition

Example

Challenges


Computer Vision
Natural Language Processing
Optical Character Recognition

Challenges



170 A BOOK OF HOMAGE TO SHAKESPEARE

BITS OF TIMBER: SOME OBSERVATIONS ON SHAKESPEARIAN NAMES—'SHYLOCK'; 'POLONIUS'; 'MALVOLIO'

I

SHAKESPEARE in *The Merchant of Venice*, as elsewhere, unconsciously divined the germ of the myth on which his genius worked. Endless analogues are quoted for the two stories blended in the play; and we know Shakespeare's debt to the *Pecorone* of Ser Giovanni Fiorentino and the rest. The legend, I feel sure, represents an early homilist's attempt to exemplify the two texts: 'Greater love hath no man than this, that a man lay down his life for his friends,' and 'Christ also loved the church and gave himself for it'. The vivid exposition of these texts produced in due course the legend of 'the Pound of Flesh', and 'the Wooing of the Lady'. Under the cover of a similitude—a different allegory—the texts are well expounded in the early English book known as *The Nuns' Rule*; and the teacher there adds, in order to drive home the lesson, 'Do not men account him a good friend who layeth his pledge in Jewry to release his companion? God Almighty laid himself in Jewry for us,' &c.[1]

The older play on the subject, shown in London at the Bull before 1580, may well have contained some abstract characters, linking it to the Morality drama. Shakespeare's *Merchant of Venice*, starting as a study of usury, in its treatment of the theme gives glimpses of the suggested origin of the legend; and the play is rightly named after the *Merchant*, whose part is one of simple dignity, and not after Shylock, the predominant character of the play. Portia's great plea for mercy, epitomizing a whole Moral play, reveals, as it were, the inmost significance of the Lady of Belmont, as originally personifying the soul, or salvation, or the Church, and links her to the far-spread beautiful allegory of 'The Four Daughters of God'.[2]

[1] *Ancren Riwle*, ed. Morton (Camden Society), p. 394; the date of the book is about 1225.

[2] From this point of view it is interesting to recall such earlier plays as *The Three Ladies of London*, and *The Three Lords and Three Ladies of London*, by Robert Wilson.

I. GOLLANCZ 171

A contemporary of Shakespeare, Joseph Fletcher, saw something of this aspect of the play, in his poem, *Christ's Bloodie Sweat*, 1613:—

> He died indeed, not as an actor dies,
> To die today, and live again tomorrow,
> In shew to please the audience, or disguise
> The idle habit of inforcèd sorrow:
> The cross his stage was, and he played the part
> Of one that for his friend did pawn his heart.

Various speculations have been hazarded as to the origin of the name 'Shylock'. 'Caleb Shillocke, his prophecie,' often adduced, is later than the play; and the suggested connexion with Scialac—'a Maronite of Mount Lebanon' living in 1614, hardly commends itself to serious consideration, nor do the other theories propounded.

Whether Shakespeare or his predecessor gave the name to the character cannot be absolutely determined; but in view of the poet's careful choice of names, and especially of other names in the play, the inference points to him.

The book which was read by Elizabethans for everything relating to the later Jewish history, and which went through edition after edition, was Peter Morwyng's translation of the pseudo-Josephus, 'A compendious and most marveylous History of the latter Times of the Jewes Commune Weale.' The influence of this book on Elizabethan literature would repay careful study. Malone already suggested that some lines in *King John* may well have been derived from Morwyng's 'History':—

> Do like the mutines of Jerusalem,
> Be friends awhile and both conjointly bend
> Your sharpest deeds of malice on this town....
> That done, dissever your united strengths,
> And part your mingled colours once again.[1]

In Marlowe's *Jew of Malta*, and elsewhere in the plays of Elizabethan dramatists, the influence of the book can be detected.

Near the beginning of the 'History' we read: 'About that time it was signified also to them of Jerusalem that the Askalonites had entered in friendship with the Romans. They sent therefore Neger the Edomite, and *Schiloch the Babylonian*, and Jehochanan, with a power of the common people; these came to Askalon, and besieged it a great space. Within the town was a Roman captaine called *Antonius*,[2] a valiant man, and a good warrior.' This passage may well account for 'Shylock'; and possibly also for 'Antonio'.

[1] *King John*, II. i. 378.

[2] Elsewhere always 'Antonie'.

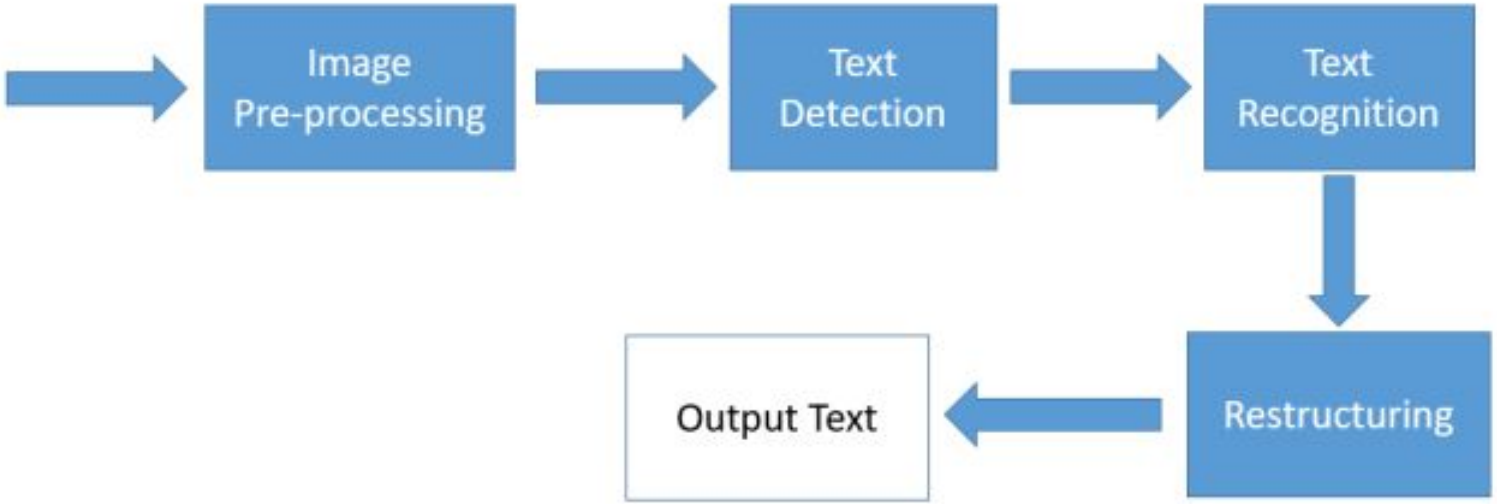
Datasets

Street View House Number dataset

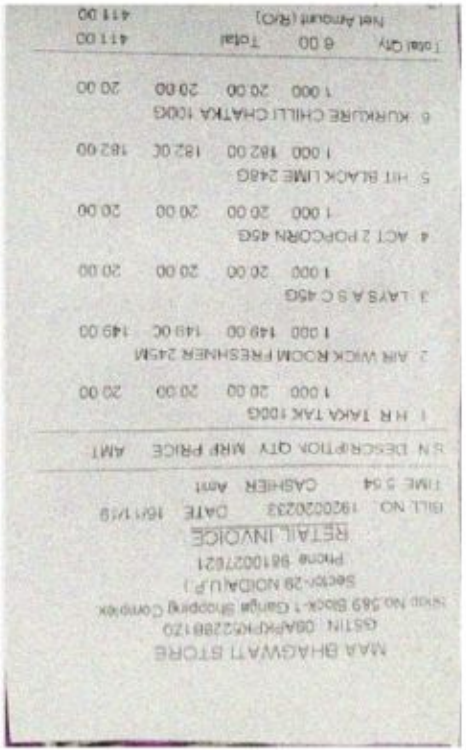
Scene Text dataset

OCR general pipeline

Input image



Step 1: Image pre-processing



Input image

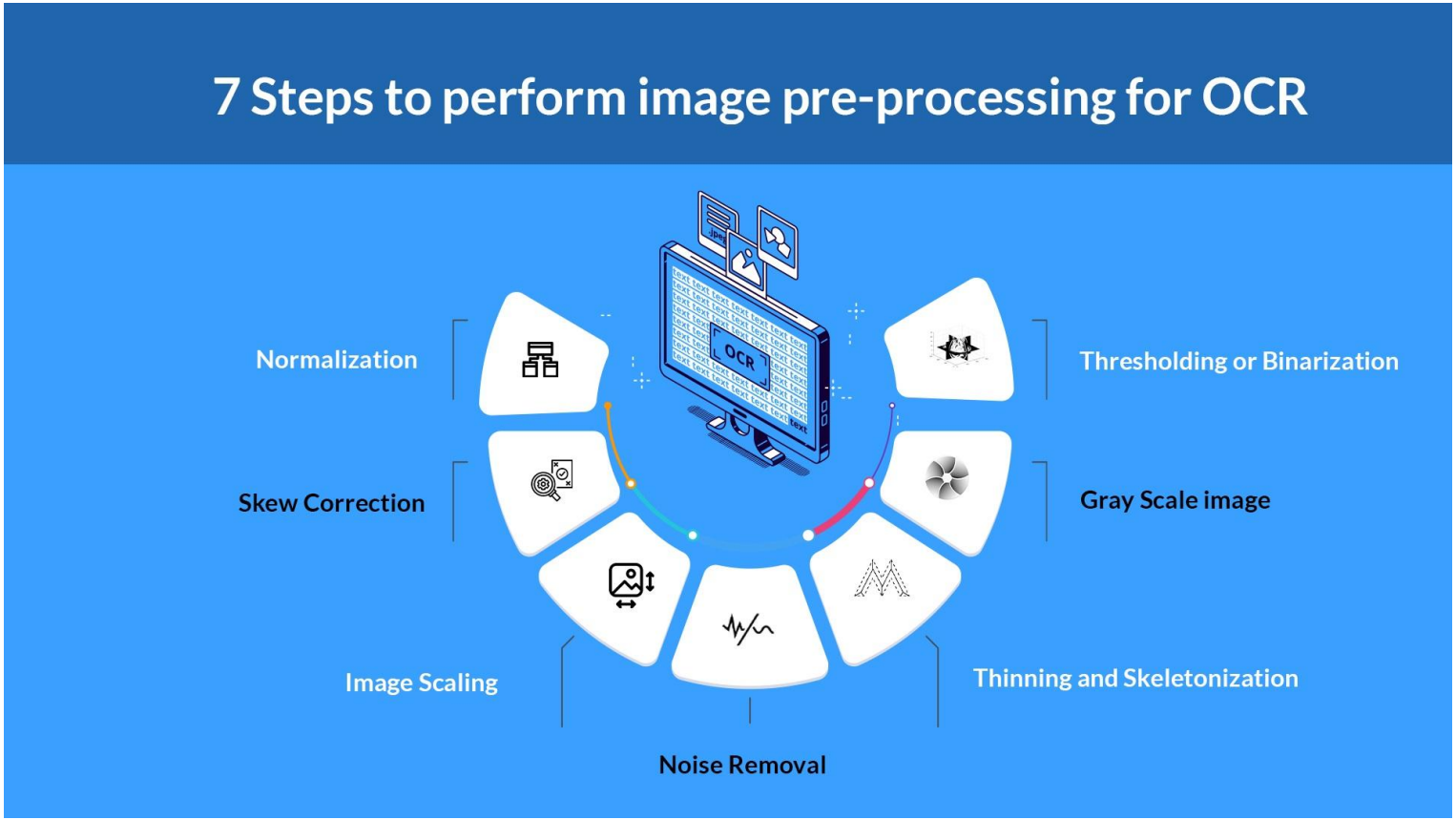
Noise and

rotation

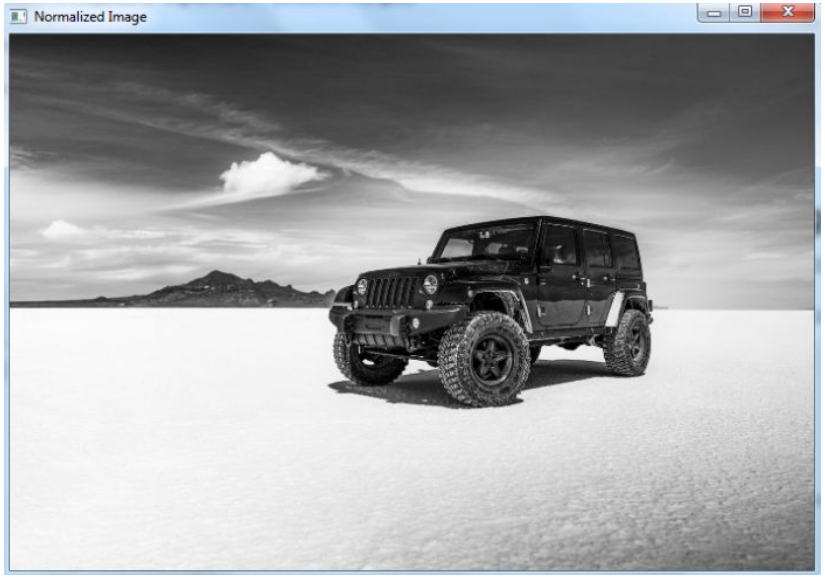
correction

Corrected image

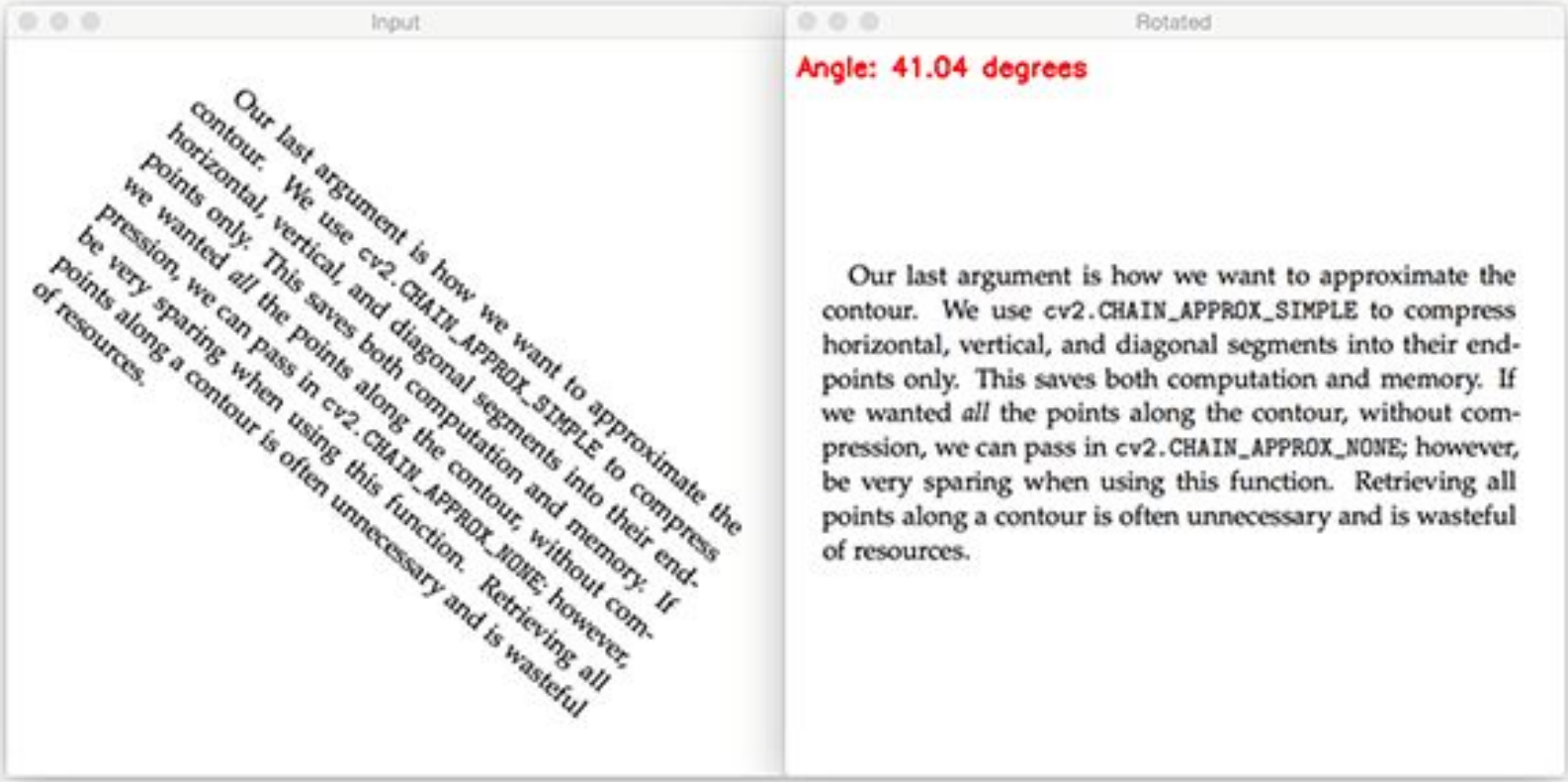
Step 1: Image pre-processing



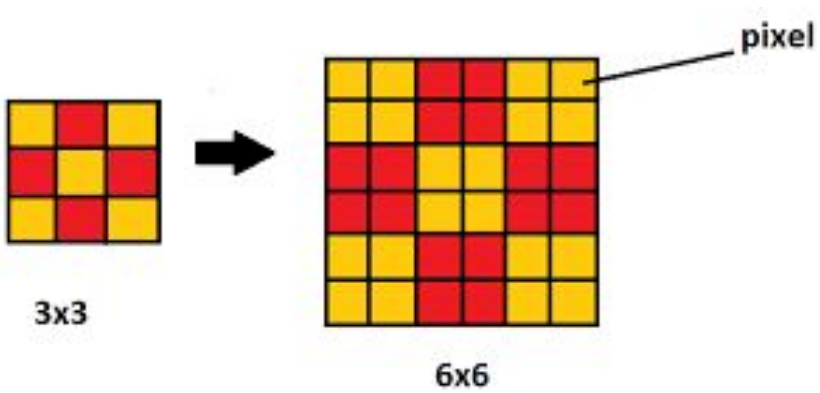
Step 1.1: Normalization

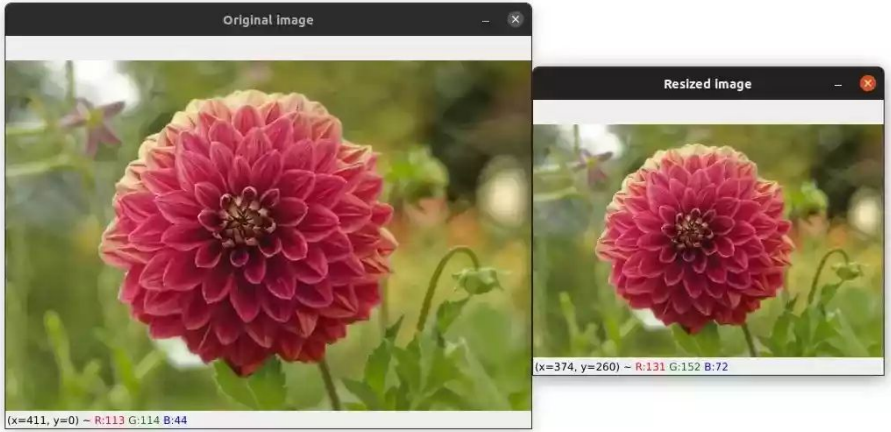


Step 1.2: Skew correction

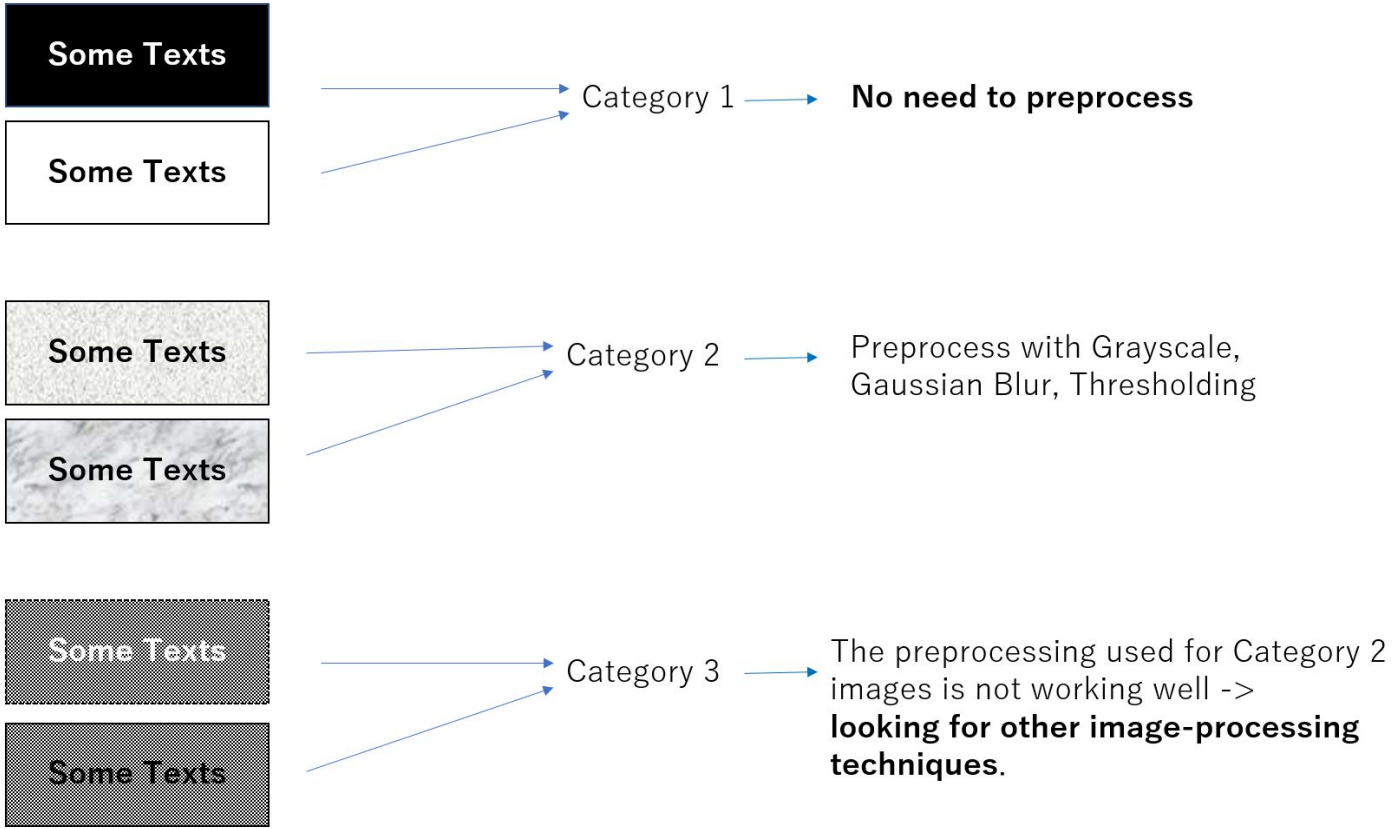


Step 1.3: Image scaling





Step 1.4: Noise Removal



Step 1.5: Thinning and Skeletonization

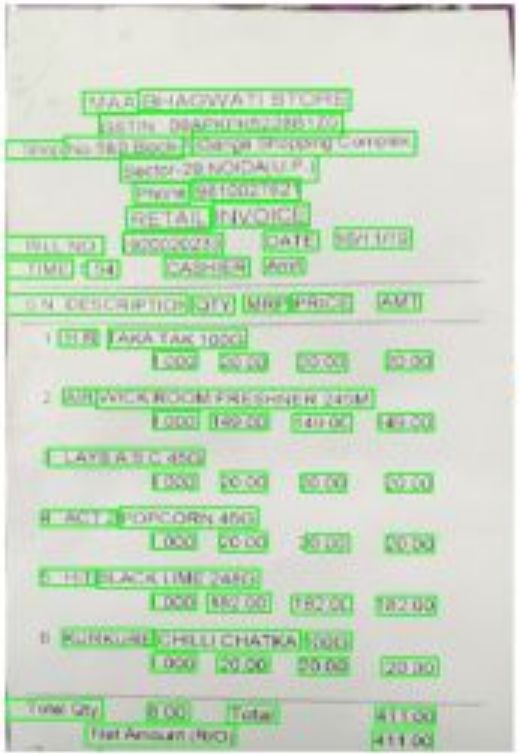
Step 1.6: Gray scale image

Step 1.7: Thresholding or Binarization

Step 2: Text detection

Text Detection

Step 3: Text recognition



Text Recognition

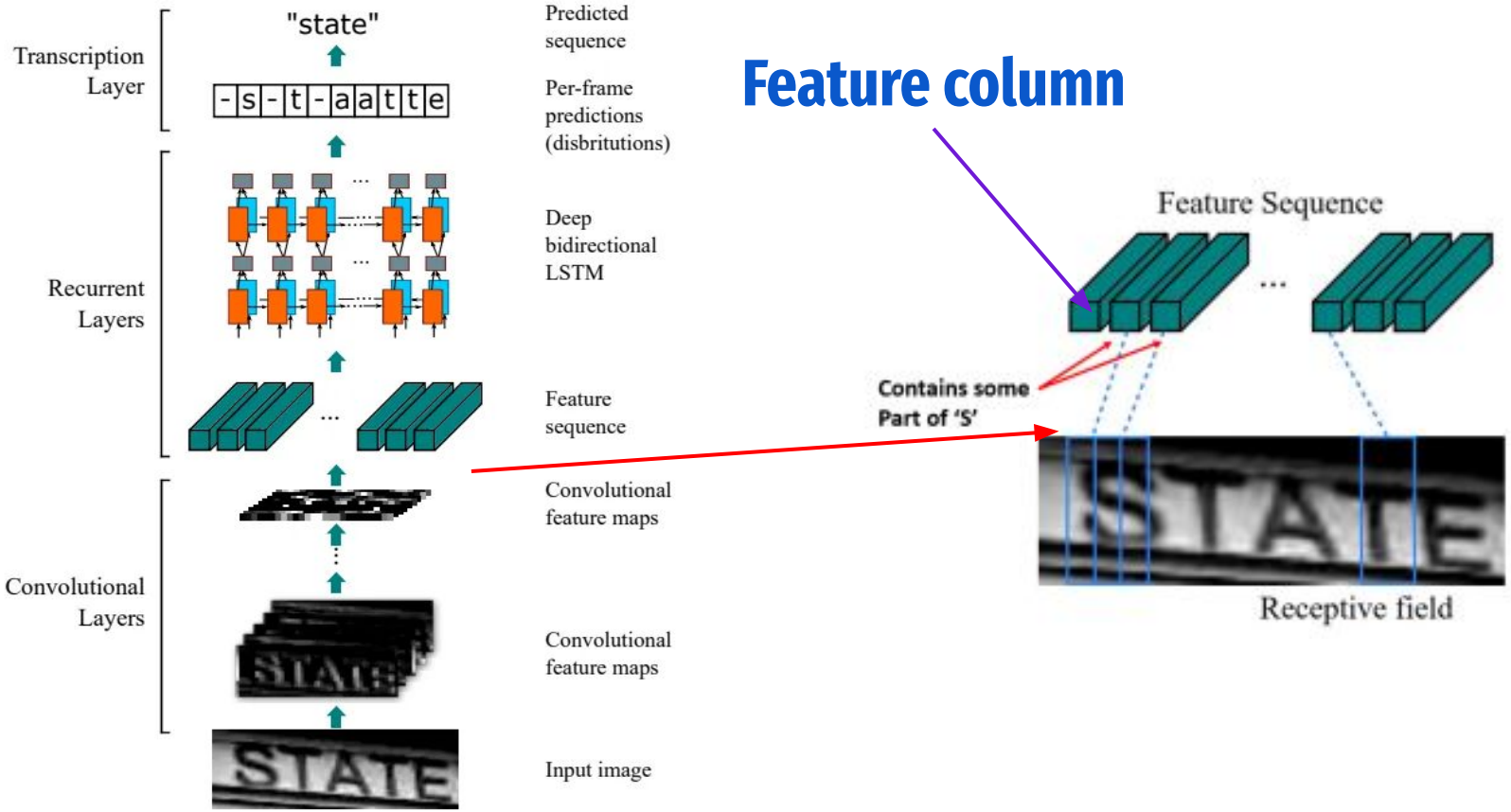
*progressive gan results model summary_info - Notepad

File Edit Format View Help

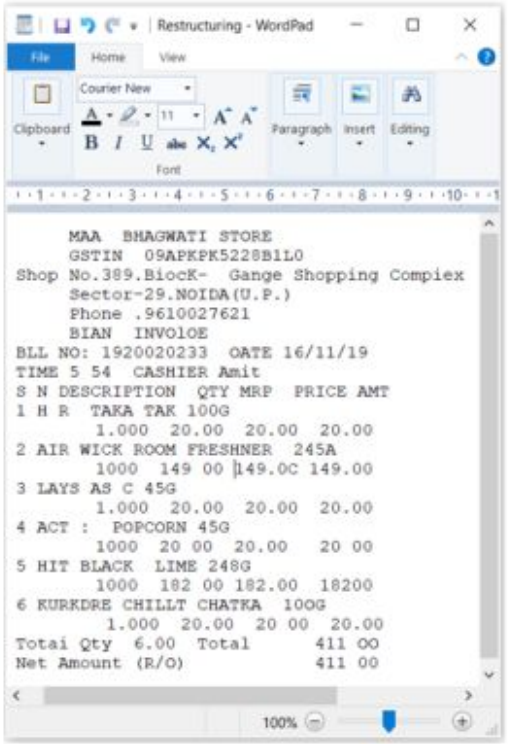
Segment	Coordiantes	Text
1_0.png	81 89 49 28	MAA
1_1.png	133 89 115 28	BHAGWATI
1_2.png	257 89 77 28	STORE
2_0.png	91 119 63 23	GSTIN
3_0.png	159 114 167 25	09APKPK5228B1L0
4_0.png	17 160 175 25	Shop No.5&9.BlocK-
5_0.png	178 155 217 27	Gariga Shopping Complex
6_0.png	114 160 197 29	Sector-29.NOIDA(U.P.)
7_0.png	121 184 162 29	Phone .9810027621
8_0.png	122 216 83 27	BIAN
9_0.png	203 212 96 25	INVOlOE
10_0.png	0 247 96 21	BLL NO:
11_0.png	112 243 107 24	1920020233
12_0.png	252 241 56 22	DATE
13_0.png	320 239 76 22	16/11/19
14_0.png	0 272 64 21	[fIME
15_0.png	72 273 35 20	5 54
16_0.png	156 268 87 24	CASHIER
17_0.png	252 270 44 18	Amit
18_0.png	0 311 50 19	[S N
19_0.png	59 309 120 22	DESCRIPTION
19_1.png	183 309 41 22	QTY
20_0.png	234 309 44 22	MRP
20_1.png	279 309 58 22	PRICE
21_0.png	363 309 48 19	AMT

Ln 18, Col 27 100% Windows (CRLF) UTF-8

Convolutional-recurrent neural network



Step 4: Restructuring



Restructured text

Original Image