Introduction to C Programming

Lecture 14: basic pattern recognition II

Wenjin Wang
wangwj3@sustech.edu.cn

12-23-2022

Course syllabus

Nr.	Lecture	Date
1	Introduction	2022.9.9
2	Basics	2022.9.16
3	Decision and looping	2022.9.23
4	Array & string	2022.9.30
5	Functions	2022.10.9 (补)
6	Pointer	2022.10.14
7	Self-defined types	2022.10.21
8	I/O	2022.10.28

Nr.	Lecture	Date
9	Head files	2022.11.4
10	Review of lectures I	2022.11.25
11	Review of lectures II	2022.12.2
12	Review of lectures III	2022.12.9
13	AI in C programming	2022.12.16
14	AI in C programming	2022.12.23
15	Summary	2022.12.30

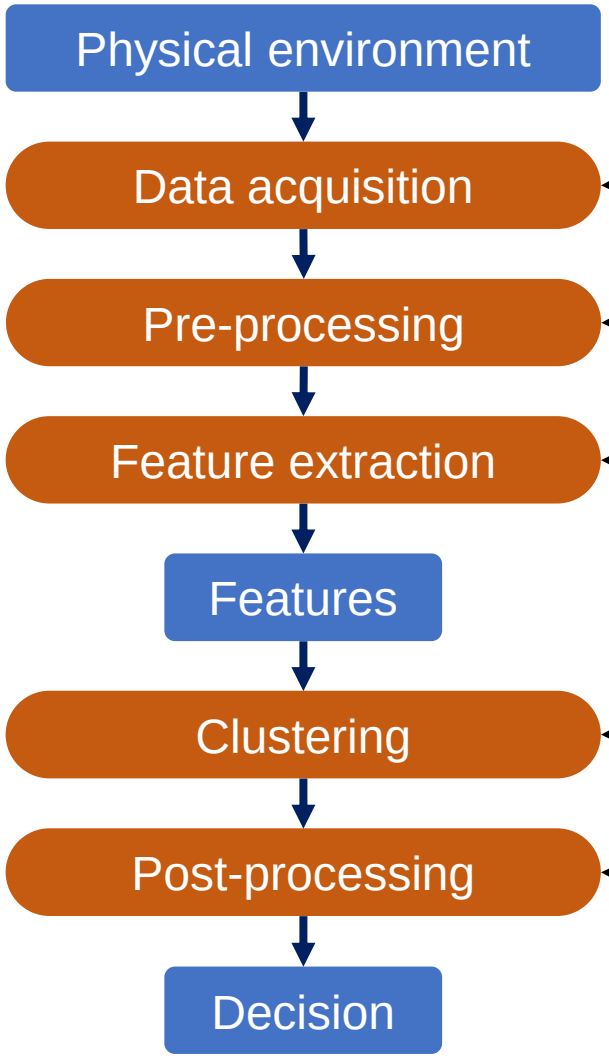
Objective of this lecture

You can play with basic AI algorithms!

Recap last lecture

- **Pattern recognition** is the key to make machine intelligent
- Two major types of pattern recognition algorithms: **clustering** and **classification**
- Two representative clustering algorithms are given: **Kmeans** and **DBscan**
- **Kmeans** needs to manually define number of clusters (the value of K)
- **DBscan** needs to input scanning radius and number of neighbours
- You know how to install **OpenCV** to visualize the Kmeans learning process

Unsupervised clustering (聚类)



Sensors: camera, microphone, electrodes, mattress, etc.

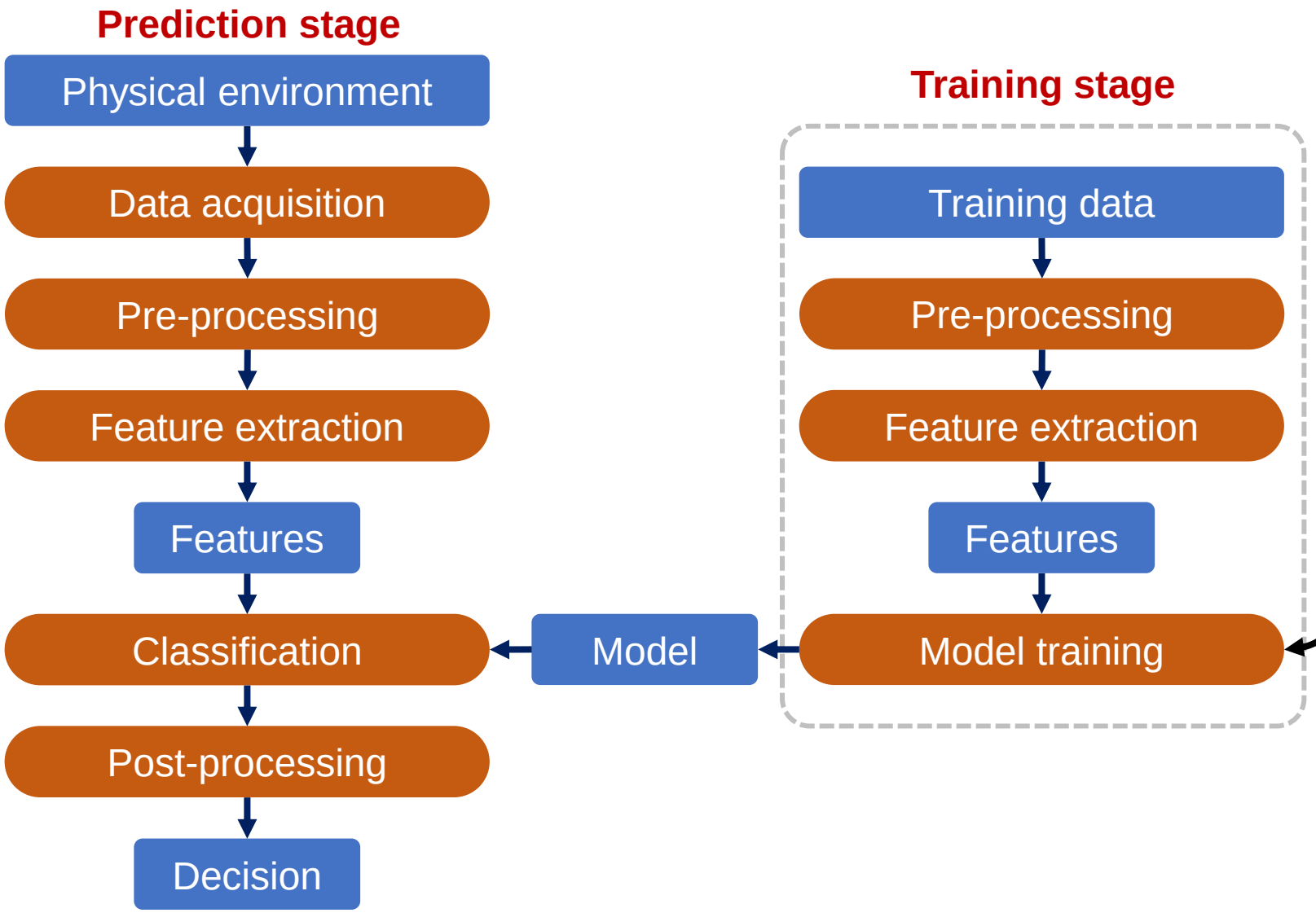
Remove noise from data, structure data format

Generate a representation to describe the data pattern
(e.g. color, shape, texture, frequency, amplitude, etc.)

Create clusters to separate data with different features
(data with similar patterns clustered together)

Evaluate the confidence, prune outliers to refine decision

Supervised classification (分类)



Model maps feature representations to decision

Content

1. Basic classification (KNN & perceptron)
2. Basics of CNN (deep learning!)
3. Applications of CNN

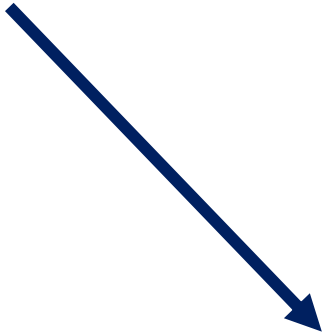
Content

1. **Basic classification (KNN & perceptron)**
2. **Basics of CNN (deep learning!)**
3. **Applications of CNN**

What is KNN?

K Nearest Neighbor

K个



最邻近点

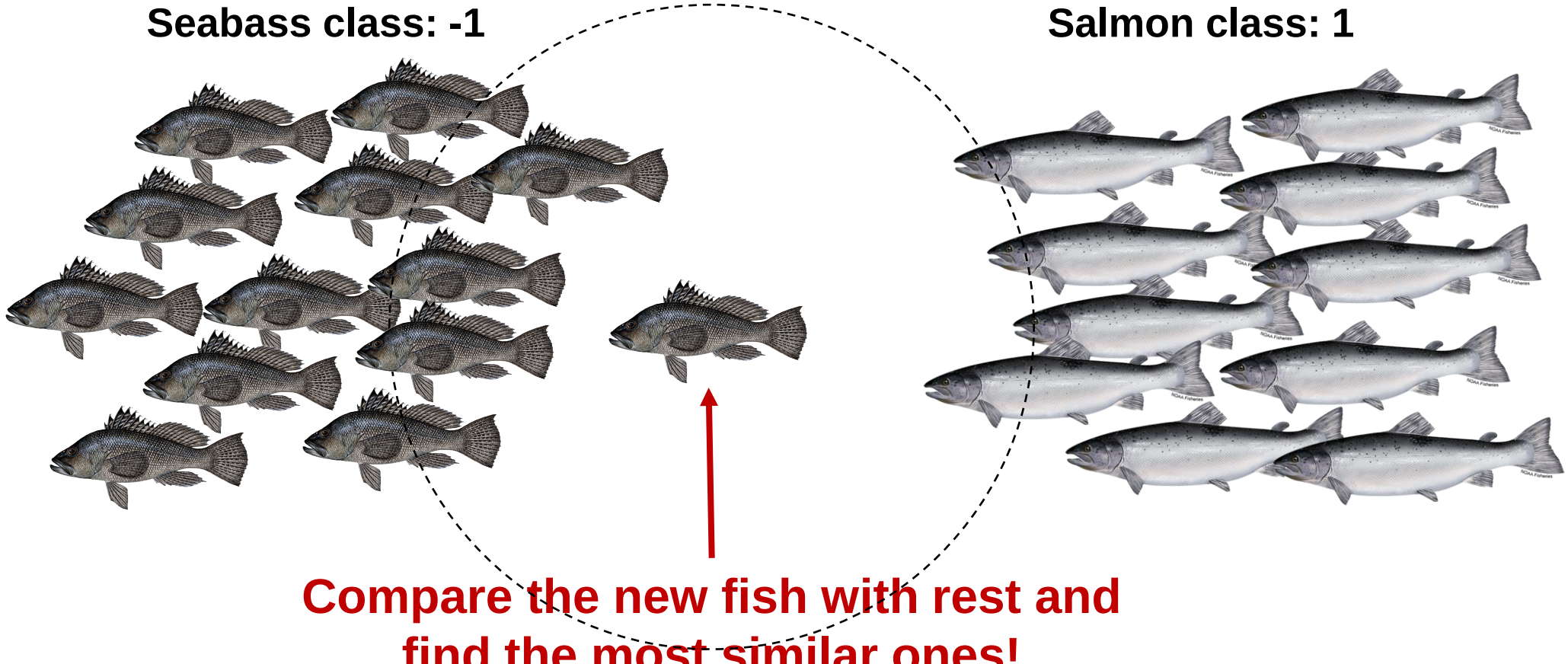
What is KNN?

K Nearest Neighbor

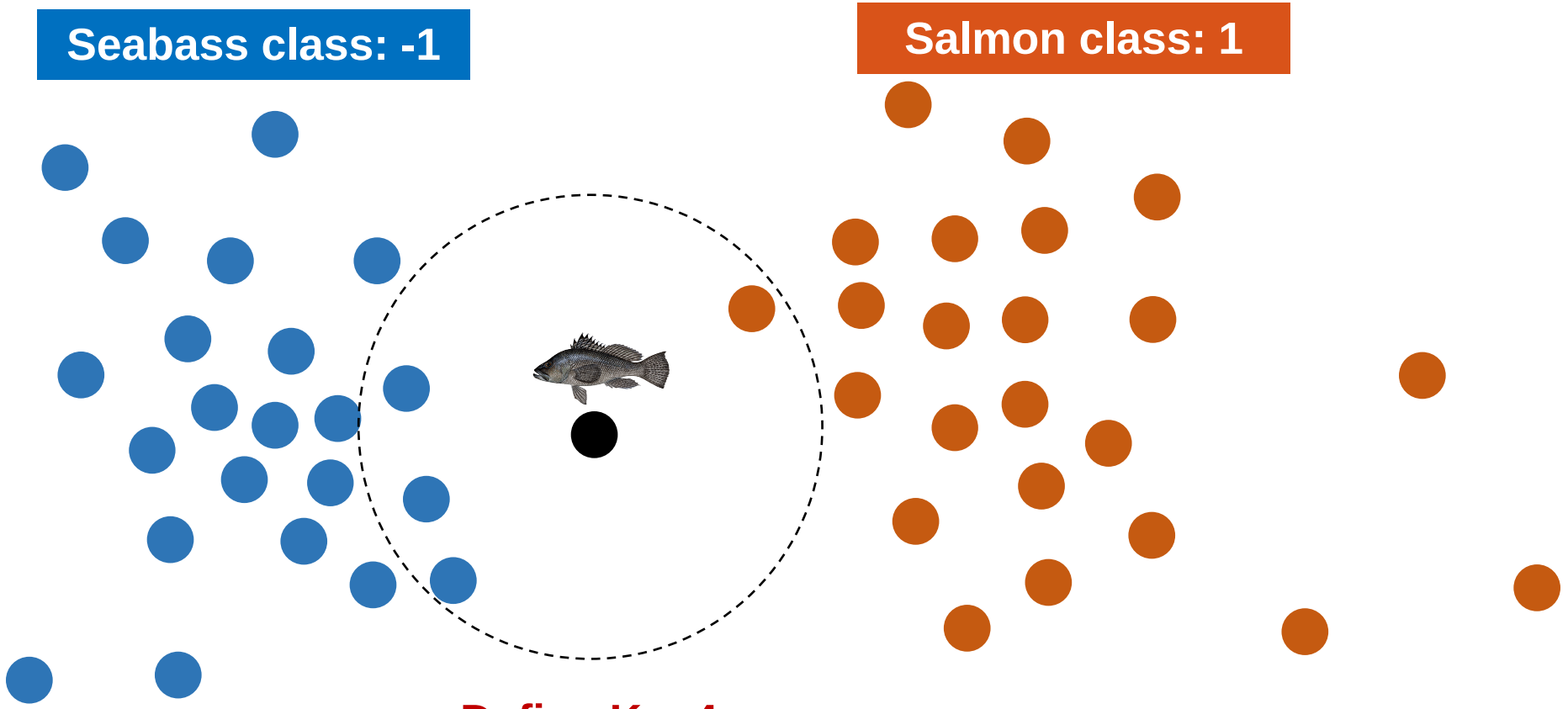
Majority of votes from neighborhood

What is KNN?

In our fish sorting case, seabass and salmon are collected as training samples

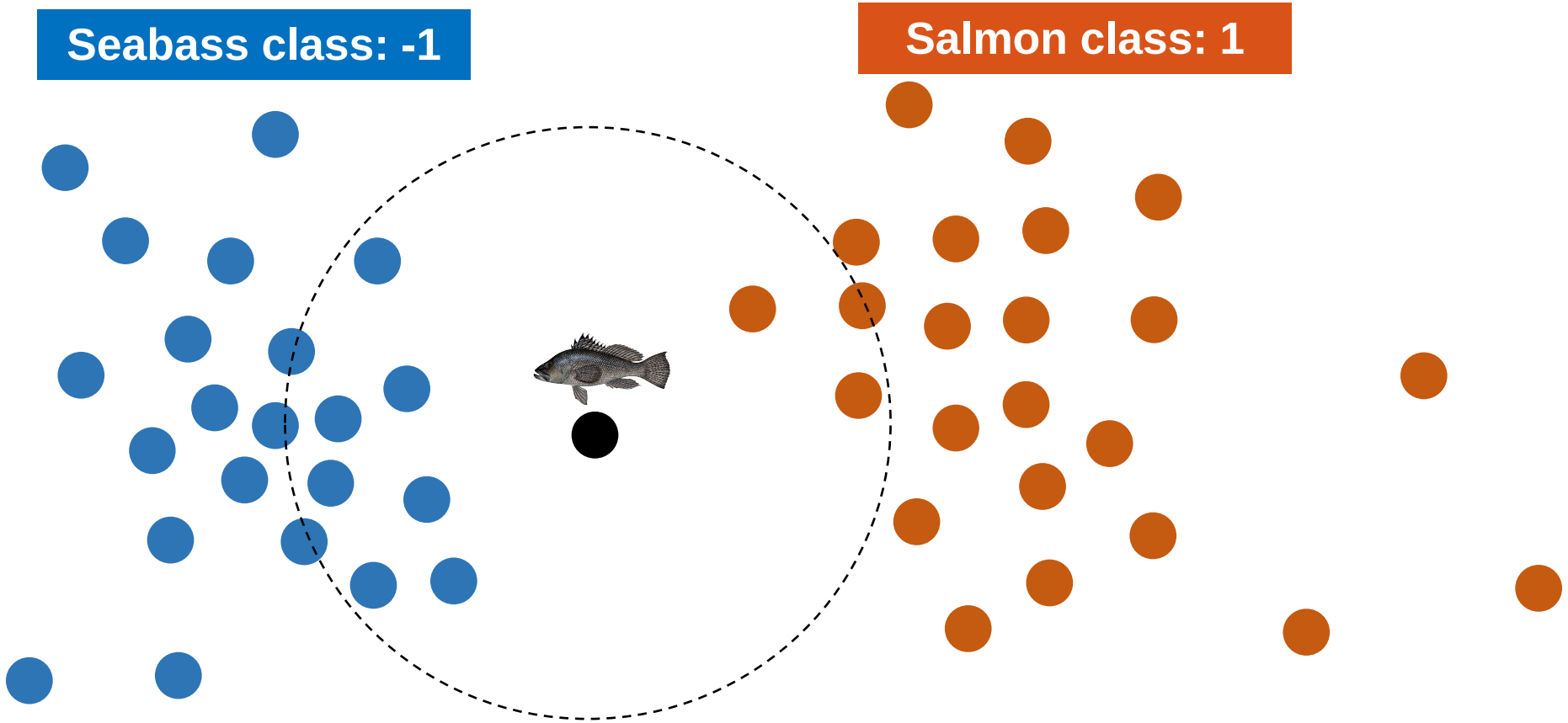


What is KNN?



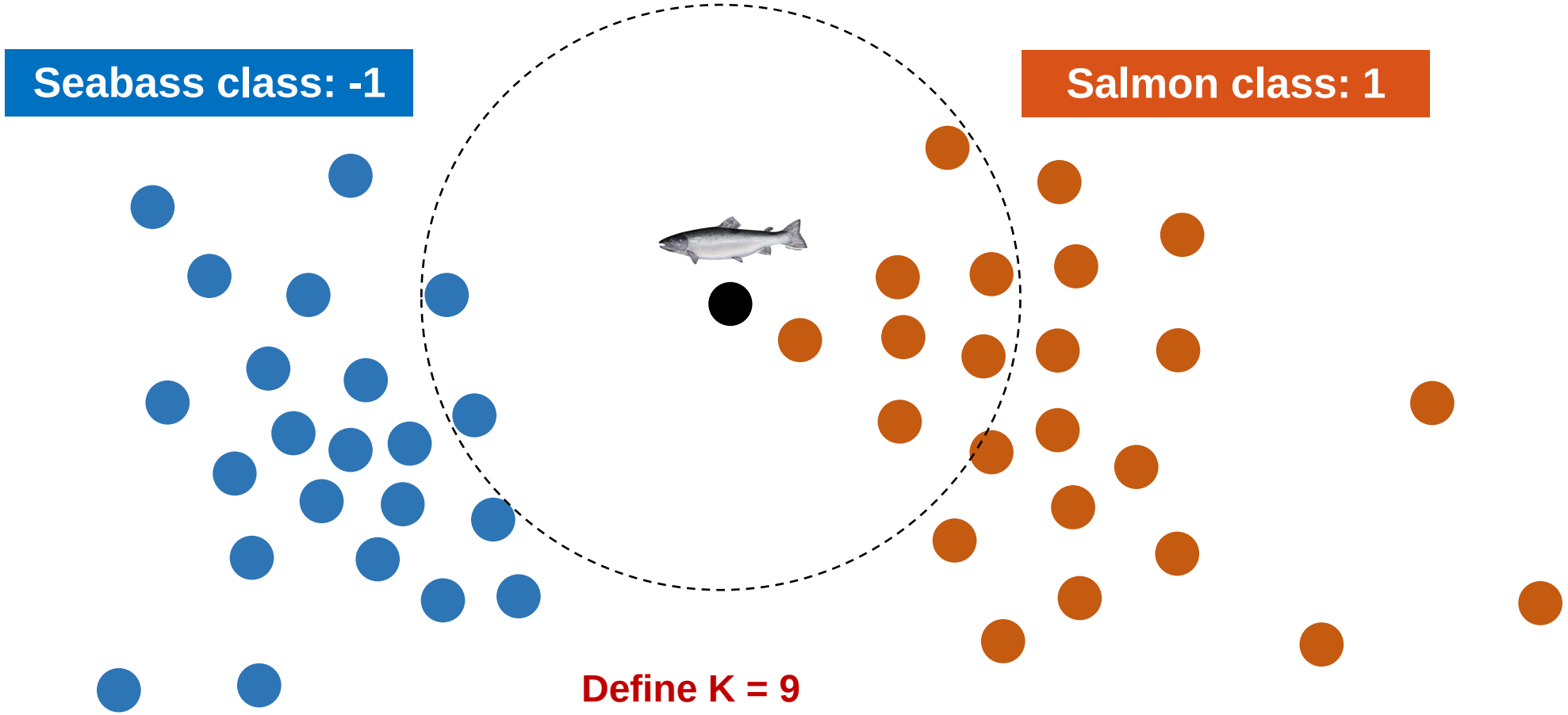
Define K = 4,
in 4 nearest neighbors, 3 are seabass, 1 is salmon

What is KNN?



Define K = 9,
in 9 nearest neighbors, 7 are seabass, 2 are salmon

What is KNN?



Define K = 9

in 9 nearest neighbors, 2 are seabass, 7 are salmon

KNN definition

KNN finds a number of K nearest neighbors in the space and use the majority of class to label the sample.

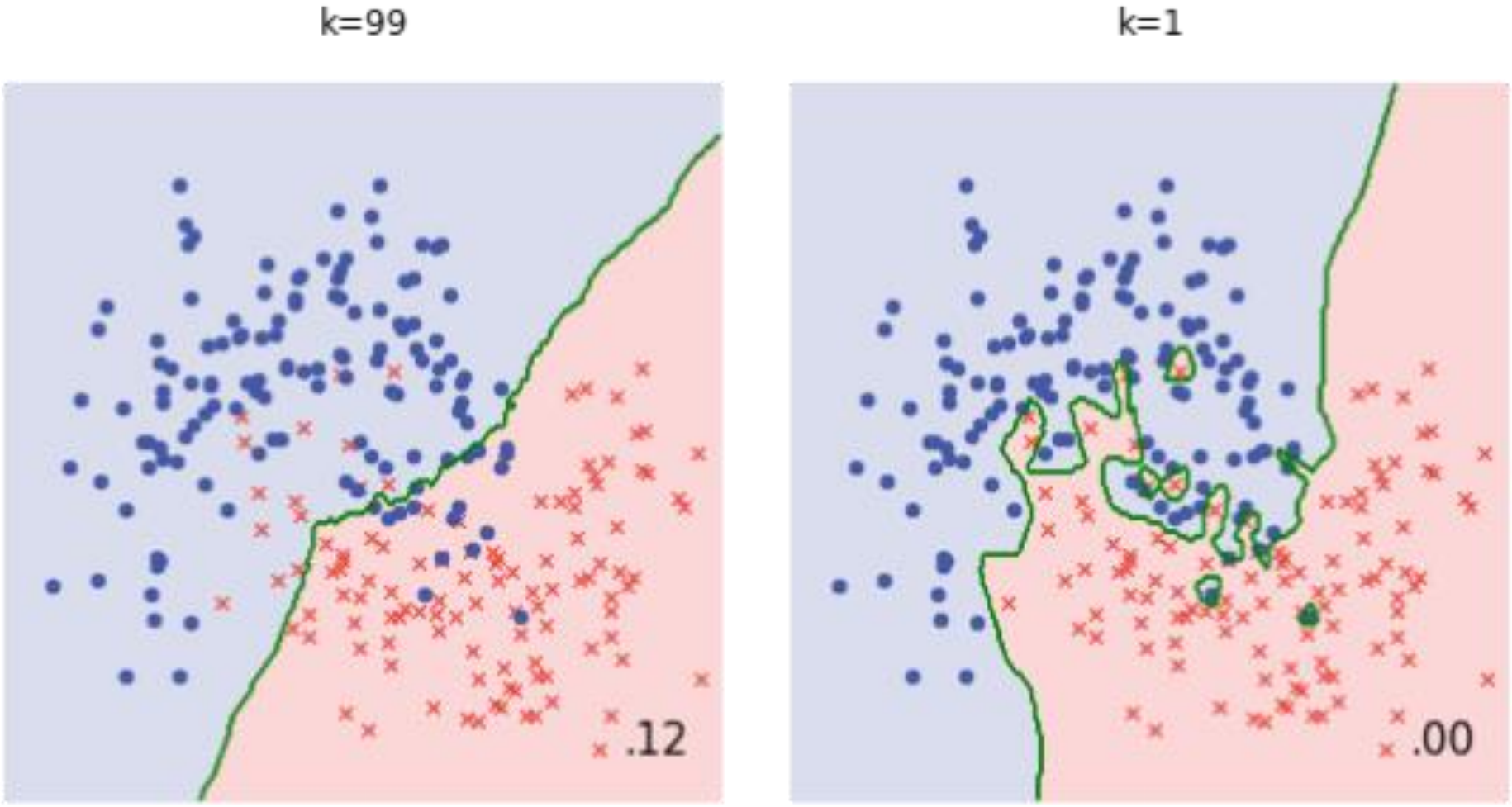
Input: samples

- Calculate the distance between sample and training samples
- Sort the distances and select the K nearest samples
- Use the majority class of K nearest samples to label the sample.

Output: class of sample

Effect of K

K is the ONLY parameter you need to define



Pros and cons of KNN

Pros

- Learning and implementation is extremely simple and intuitive
- Flexible decision boundaries (can be highly non-linear)

Cons

- Irrelevant or redundant features have negative impact
- Rely on the distance metric, cannot handle clusters with different densities

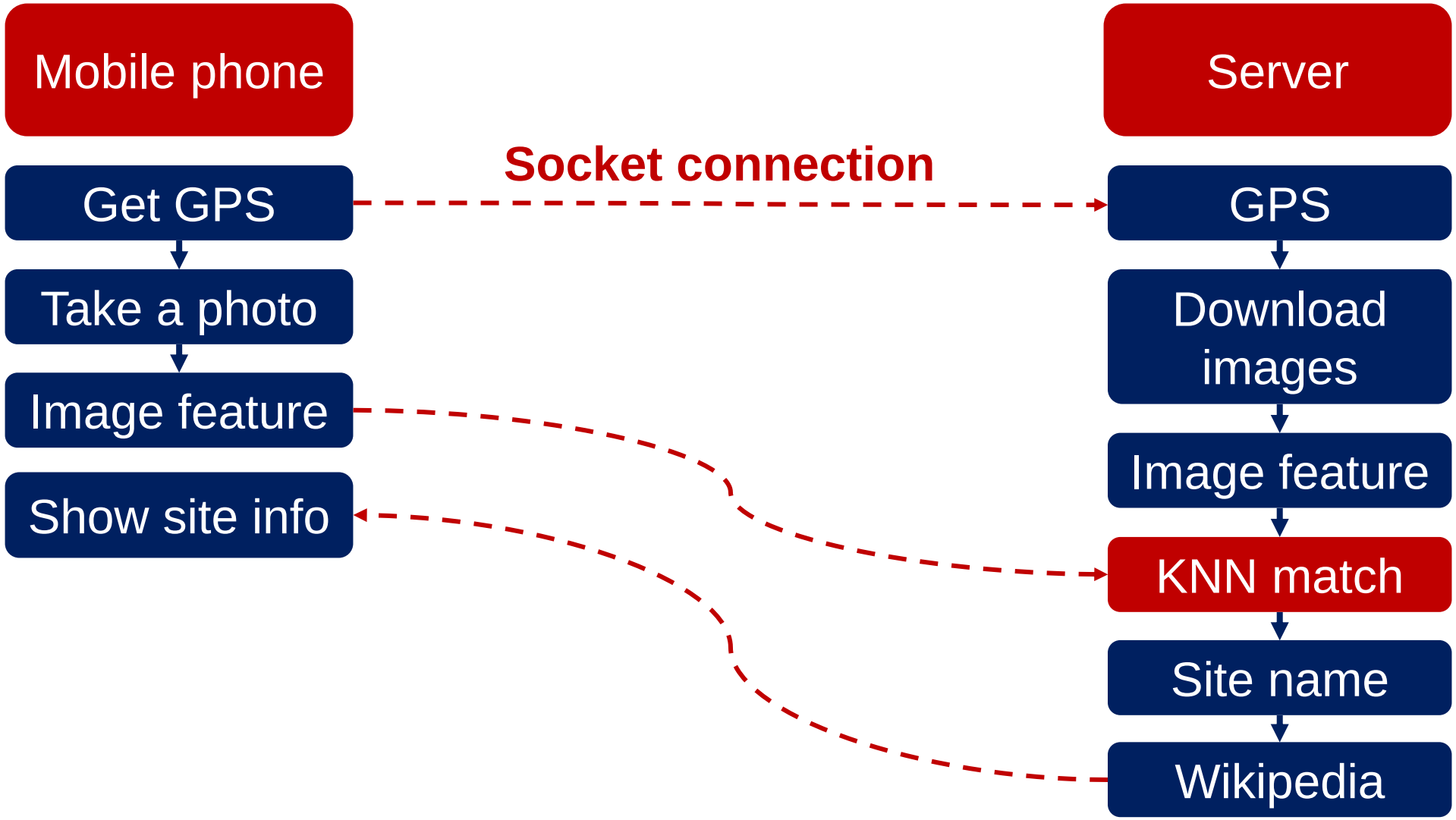
Applications of KNN

My MSc assignment 10 years ago at University of Amsterdam!

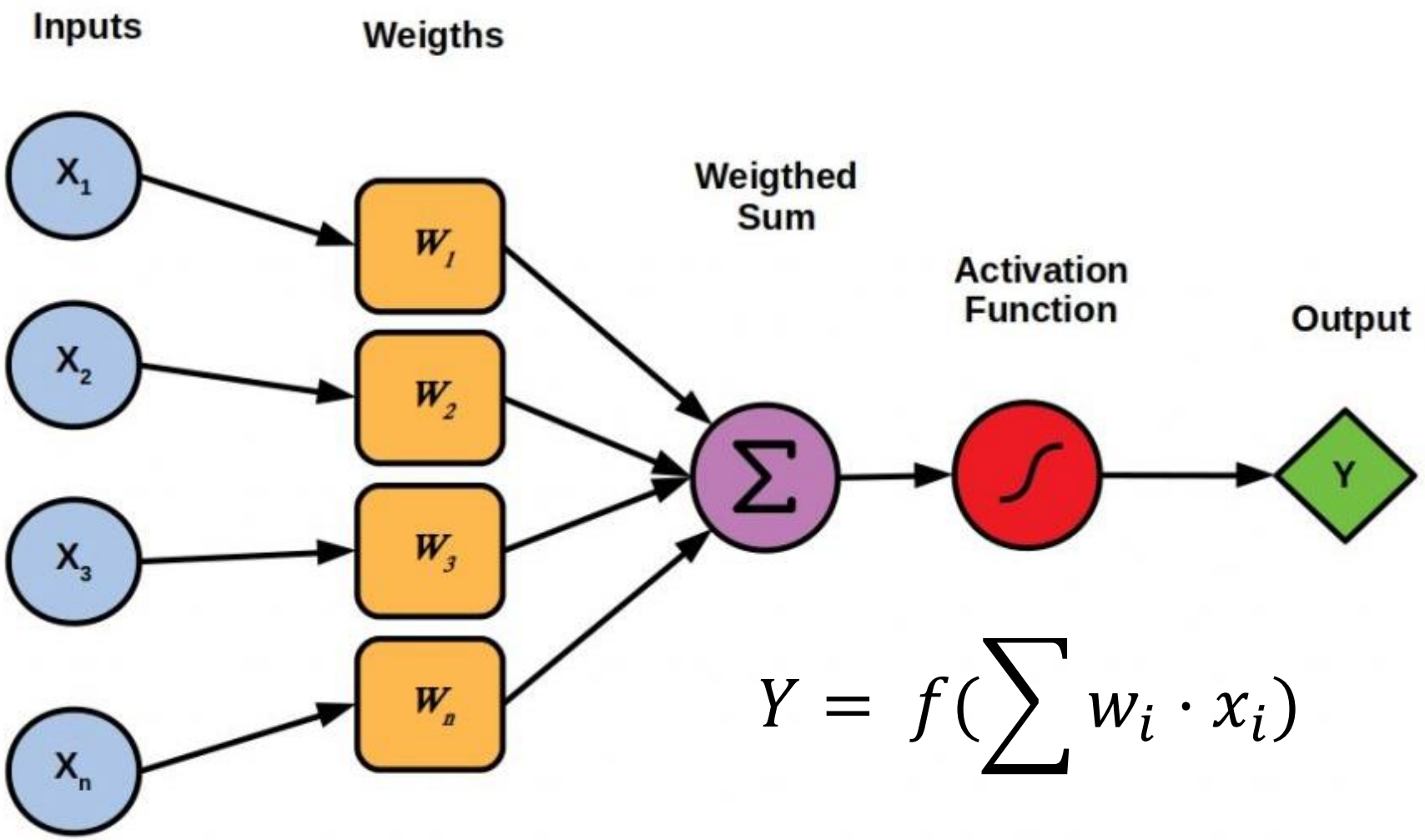
Develop an APP that search the information of an image taken by a phone!

KNN algorithm was implemented to search the image.

Applications of KNN

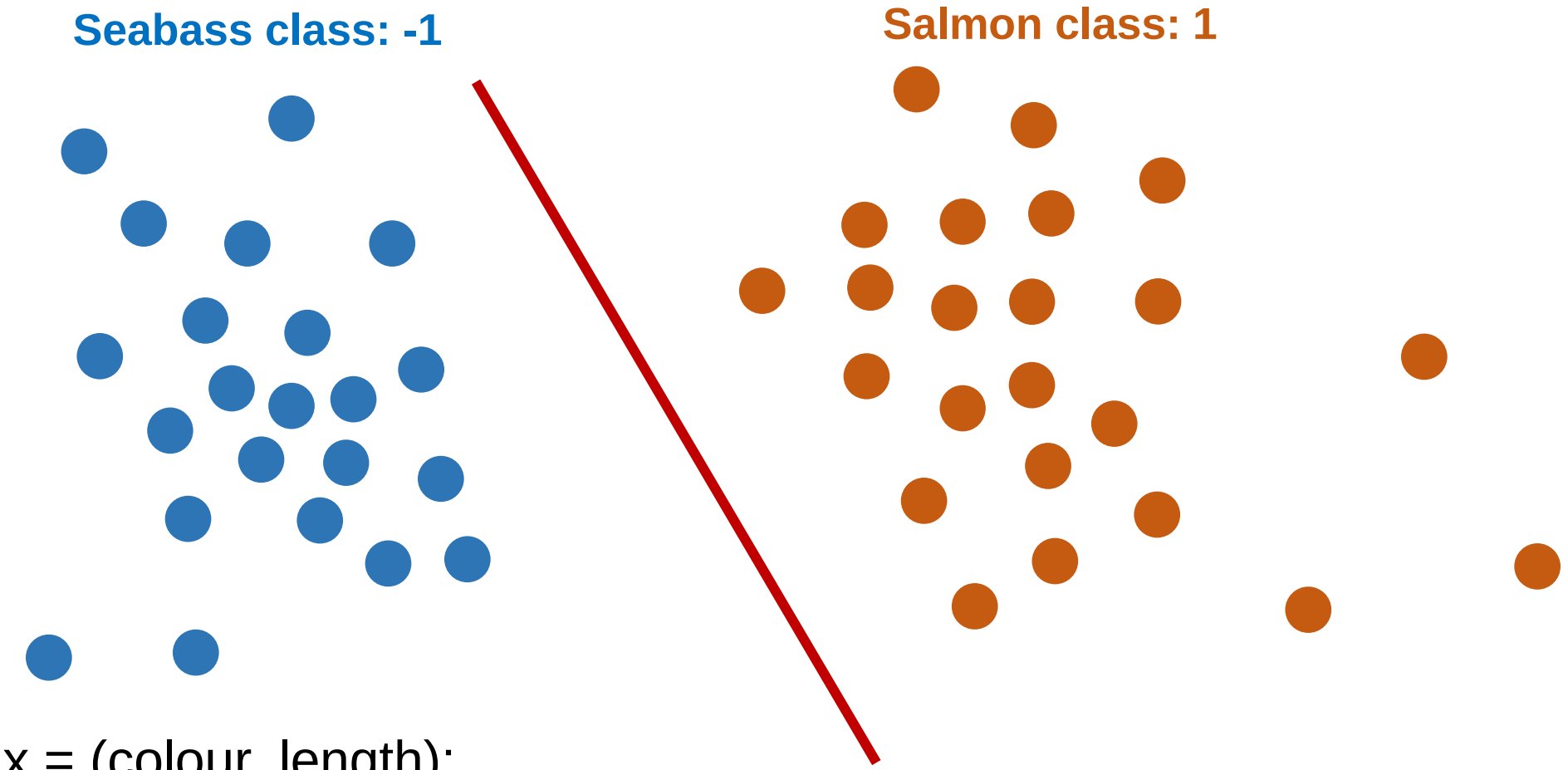


What is perceptron?



$$Y = f(\sum w_i \cdot x_i)$$

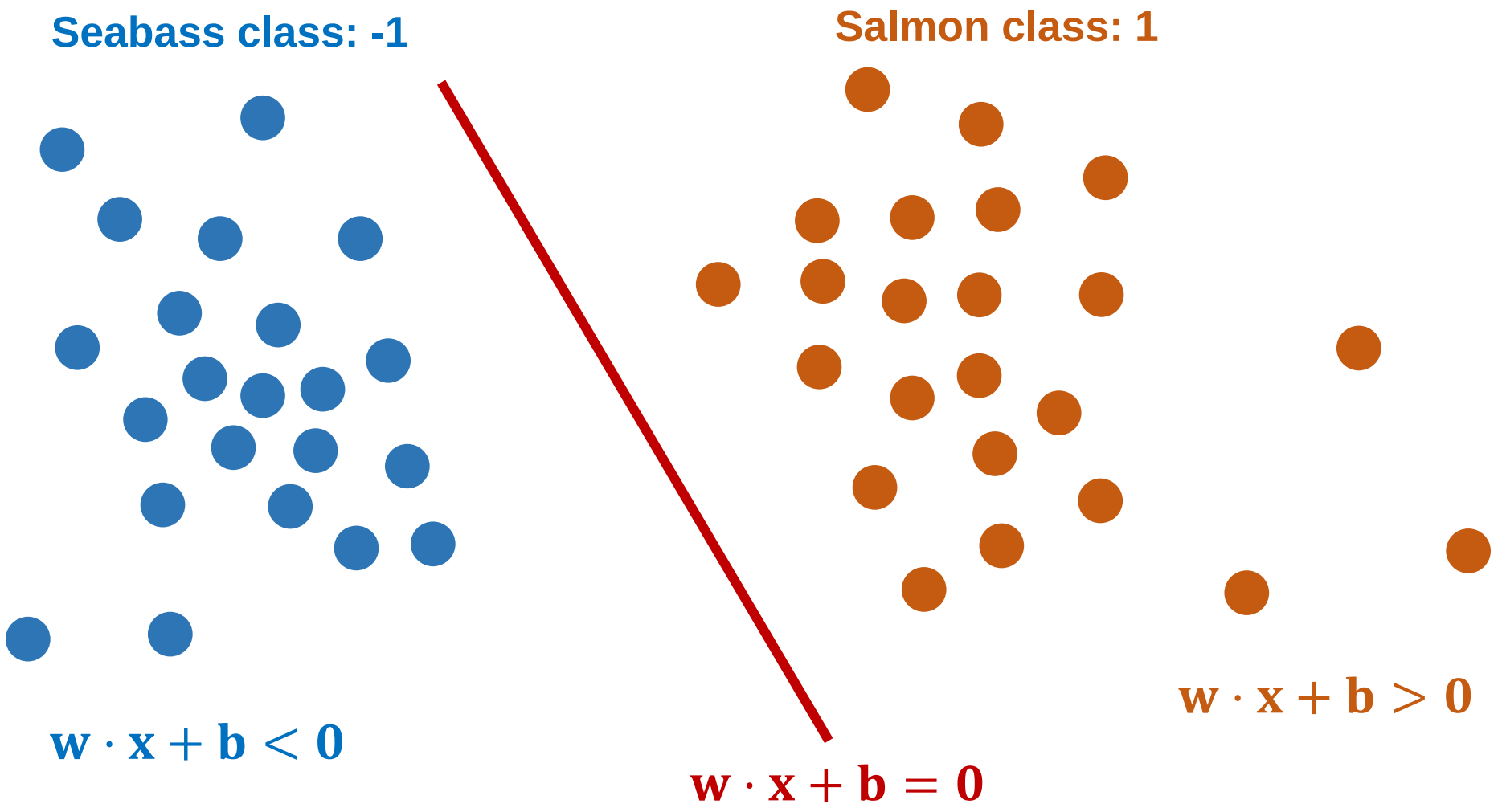
Problem definition in perceptron



x = (colour, length);
y = label; // -1 or 1

$w \cdot x + b = 0$

Problem definition in perceptron



Perceptron

Label:
y = {-1, 1}

Data:
x = (color, length)

$$f(x) = \text{sign}(w \cdot x + b)$$

Model coefficients

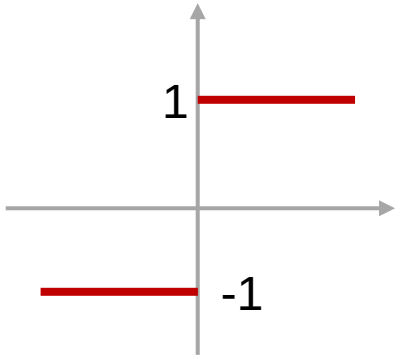
Perceptron

Label:
y = {-1, 1}

Data:
x = (color, length)

$$f(x) = \text{sign}(w \cdot x + b)$$

$$\textbf{sign}(\mathbf{x}) = \begin{cases} +\mathbf{1}, \boldsymbol{x} \geq \mathbf{0} \\ -\mathbf{1}, \boldsymbol{x} < \mathbf{0} \end{cases}$$



Perceptron

$$f(x) = \text{sign}(w \cdot x + b)$$

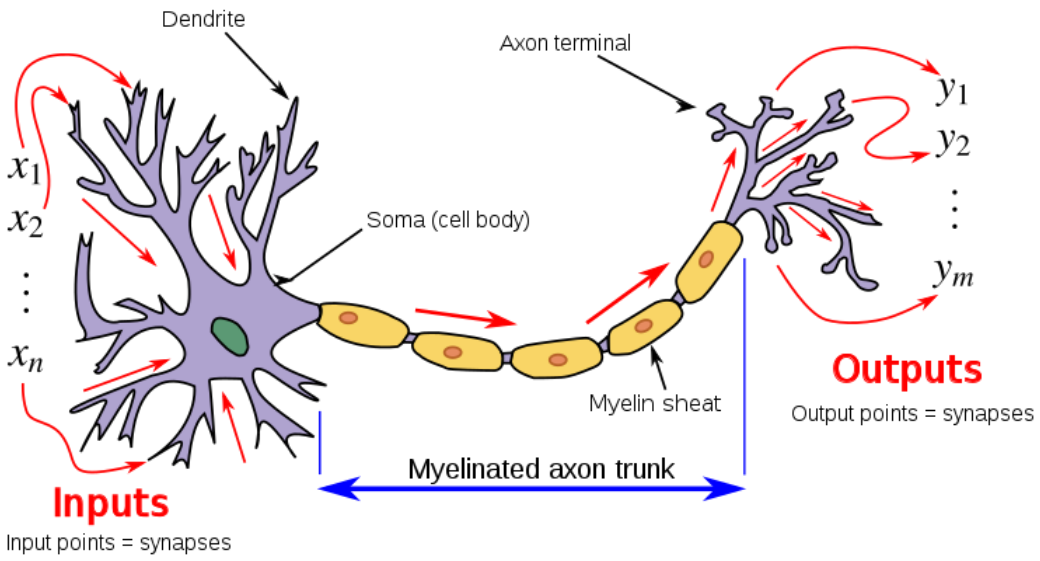
$$w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_n \cdot x_n + b \begin{cases} +1 \\ -1 \end{cases}$$

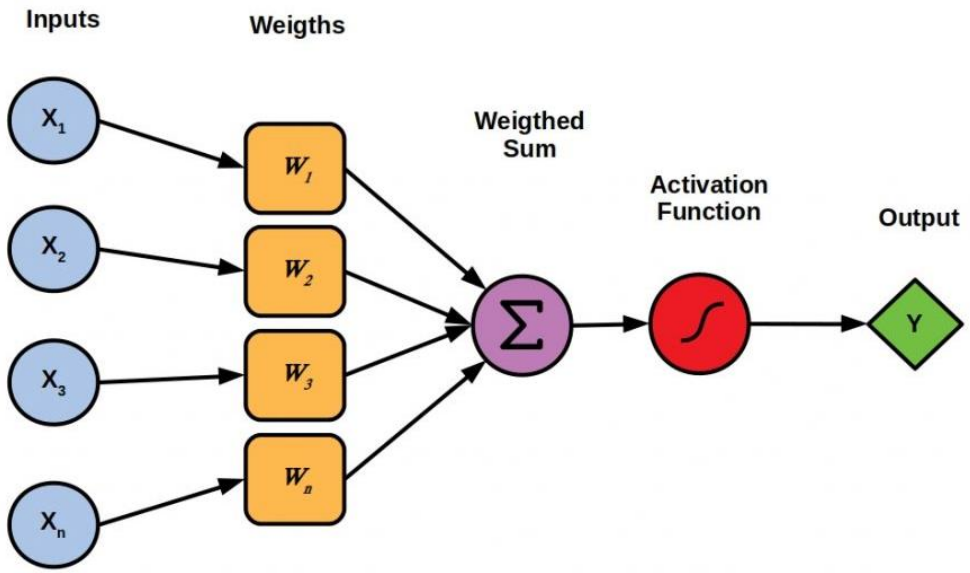
weights

bias

Perceptron

$$f(x) = \text{sign}(w \cdot x + b)$$





Perceptron

$$f(x) = \text{sign}(w \cdot x + b)$$

Decision

From sensor

How to get model parameters???

Train it!!!

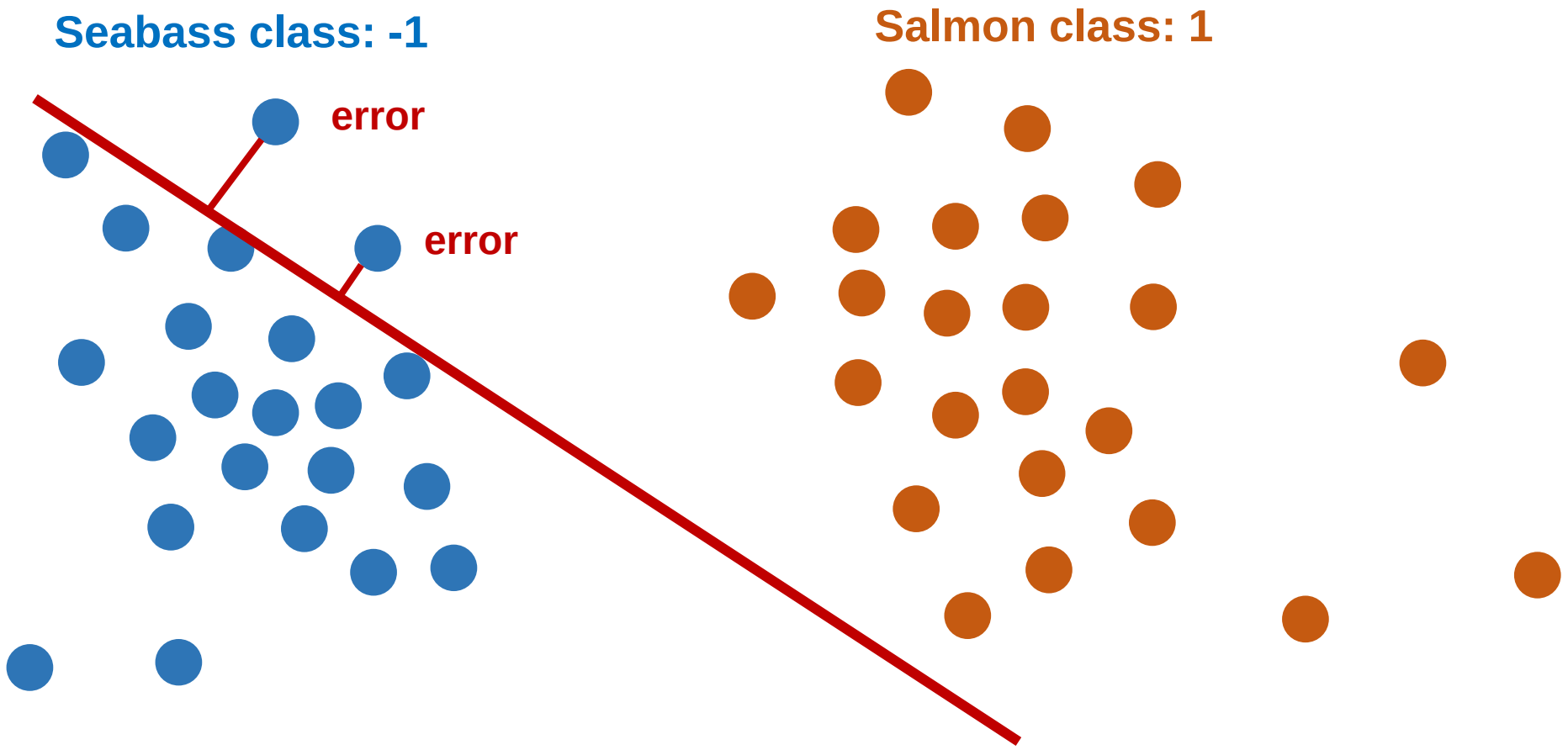
Perceptron

How to train a perceptron?

Minimize the error in classification!

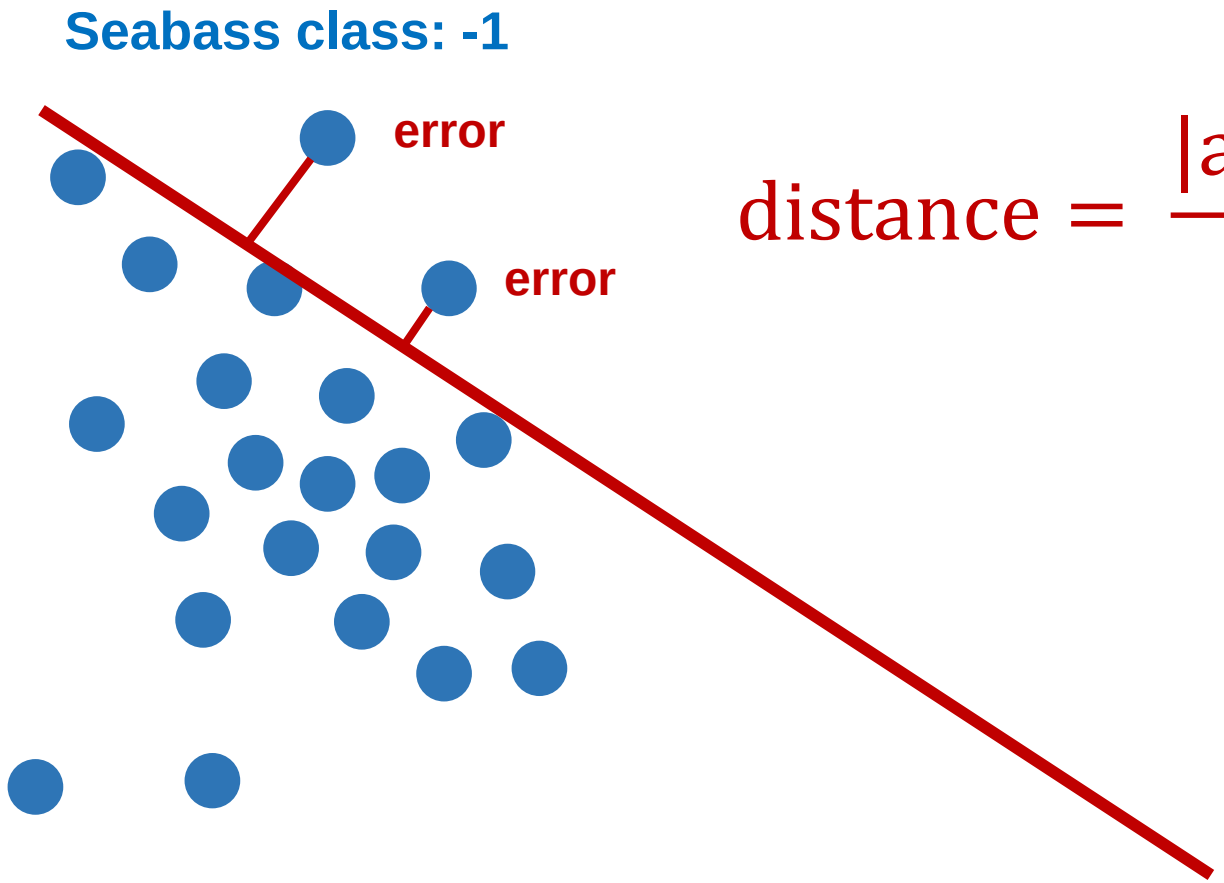
Perceptron

Minimize the error in classification



Perceptron

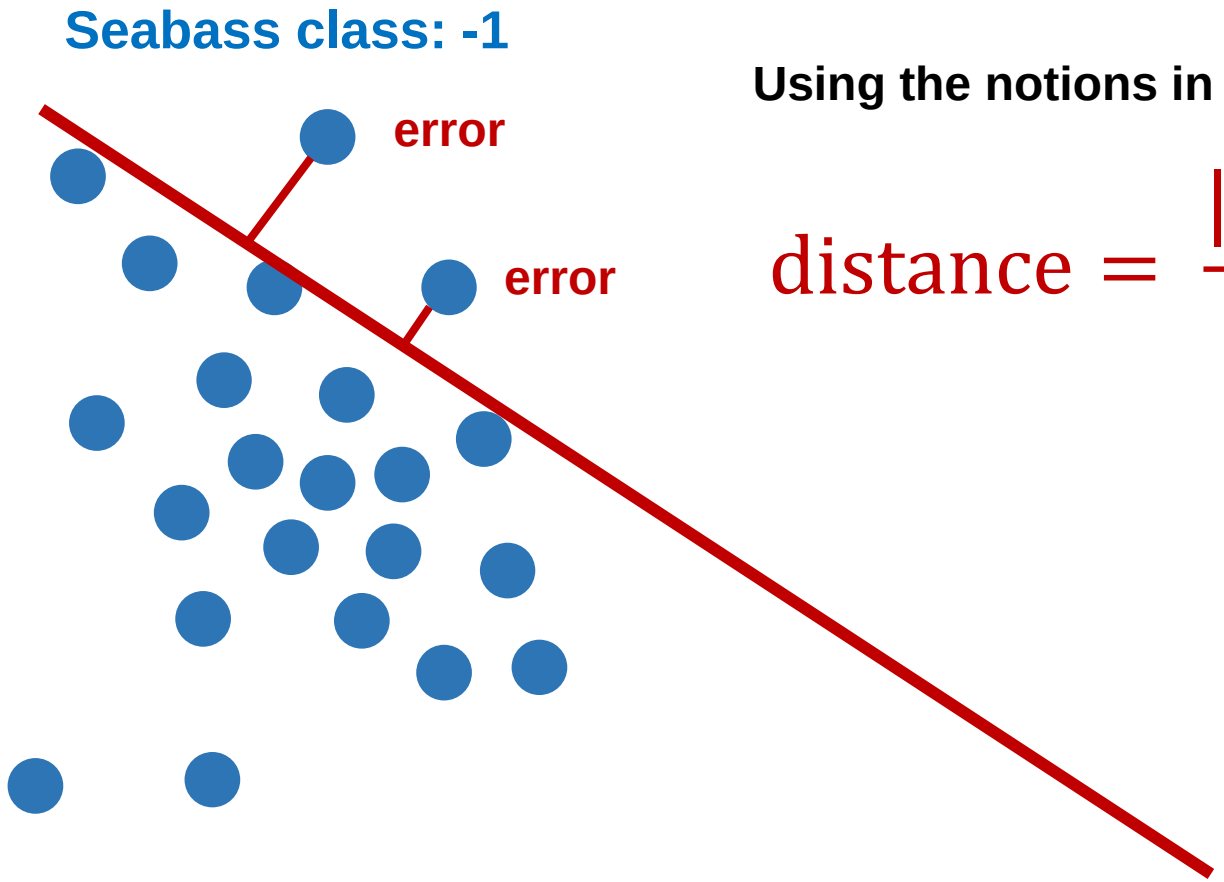
Minimize the error in classification



$$\text{distance} = \frac{|a \cdot x + b \cdot y + c|}{\sqrt{a^2 + b^2}}$$

Perceptron

Minimize the error in classification

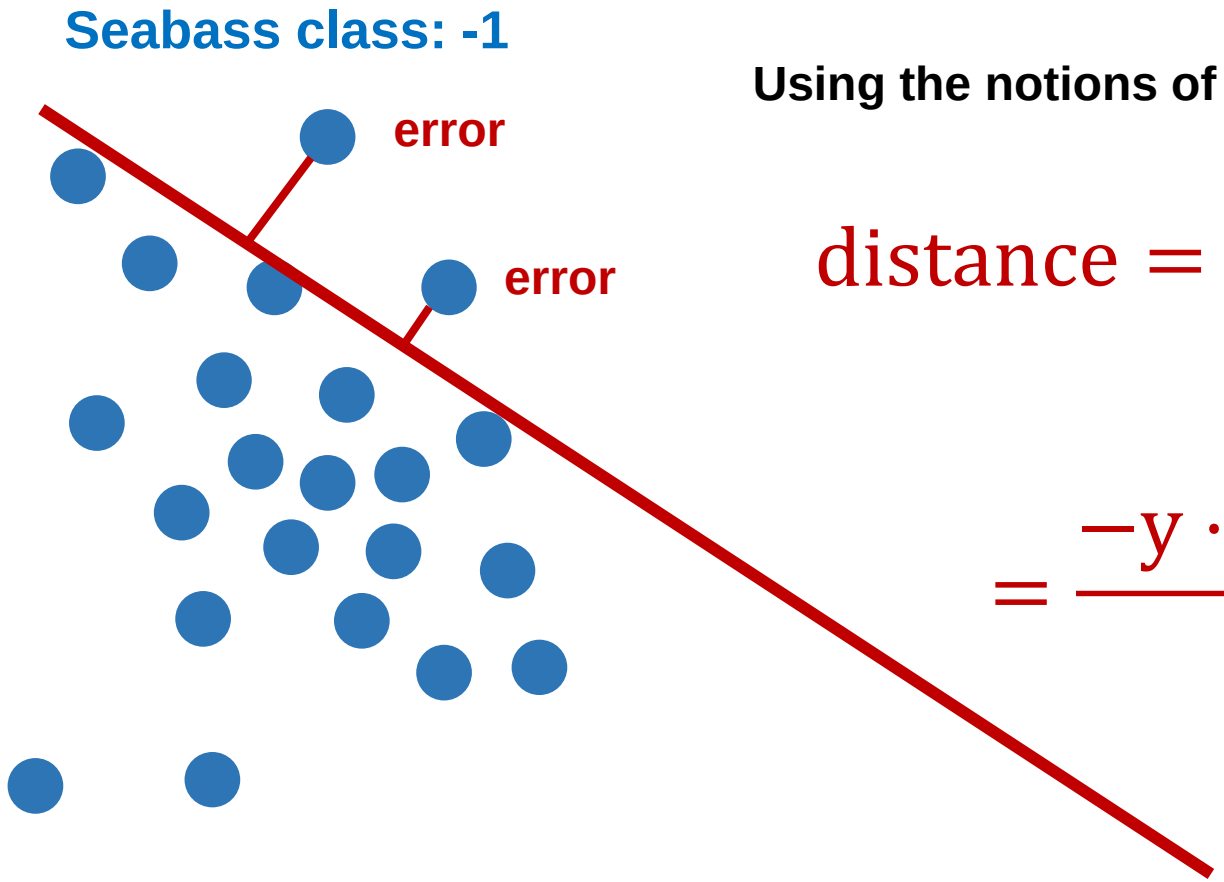


Using the notions in our task:

$$\text{distance} = \frac{|w_1 \cdot x_1 + w_2 \cdot x_2 + b|}{\sqrt{w_1^2 + w_2^2}}$$

Perceptron

Minimize the error in classification



Using the notions of our task:

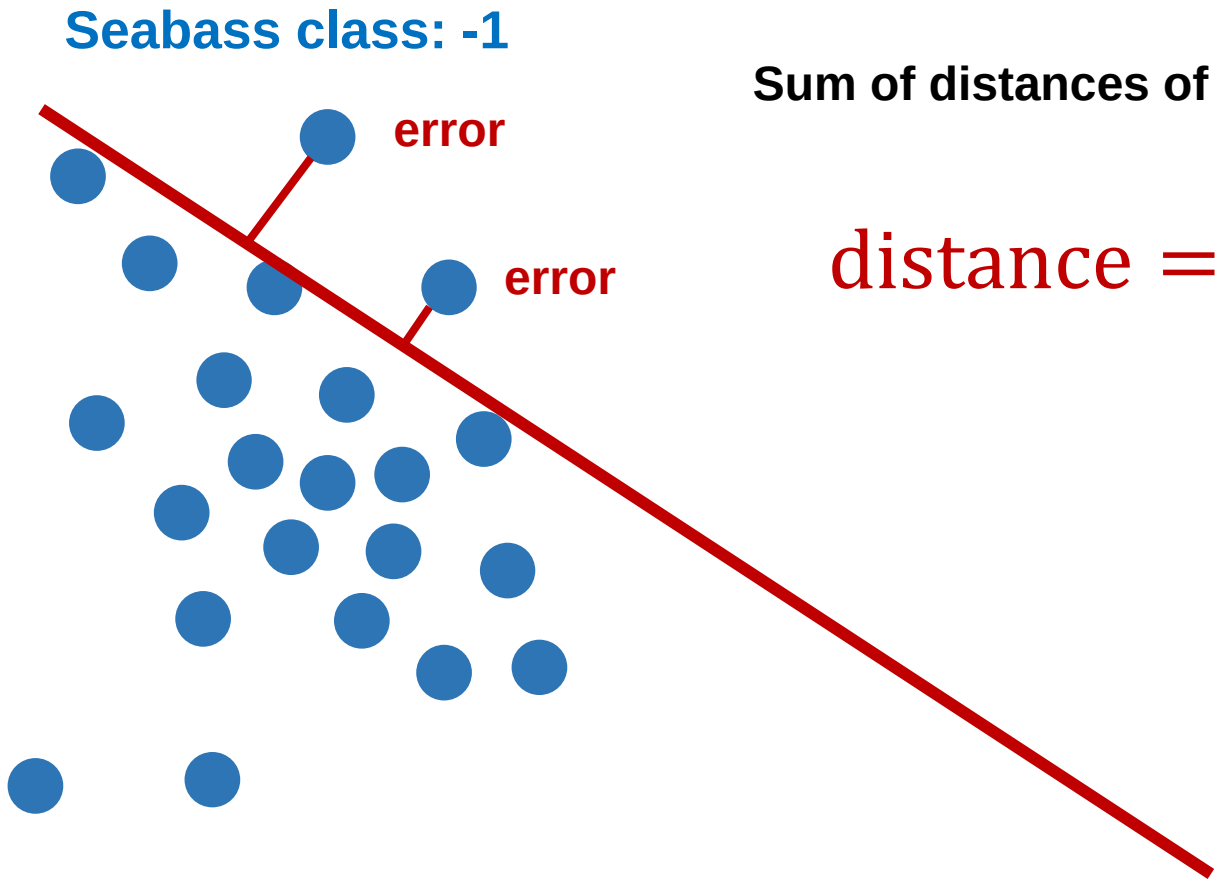
$$\text{distance} = \frac{|w \cdot x + b|}{||w||}$$

$$= \frac{-y \cdot (w \cdot x + b)}{||w||}$$

L2 norm

Perceptron

Minimize the error in classification



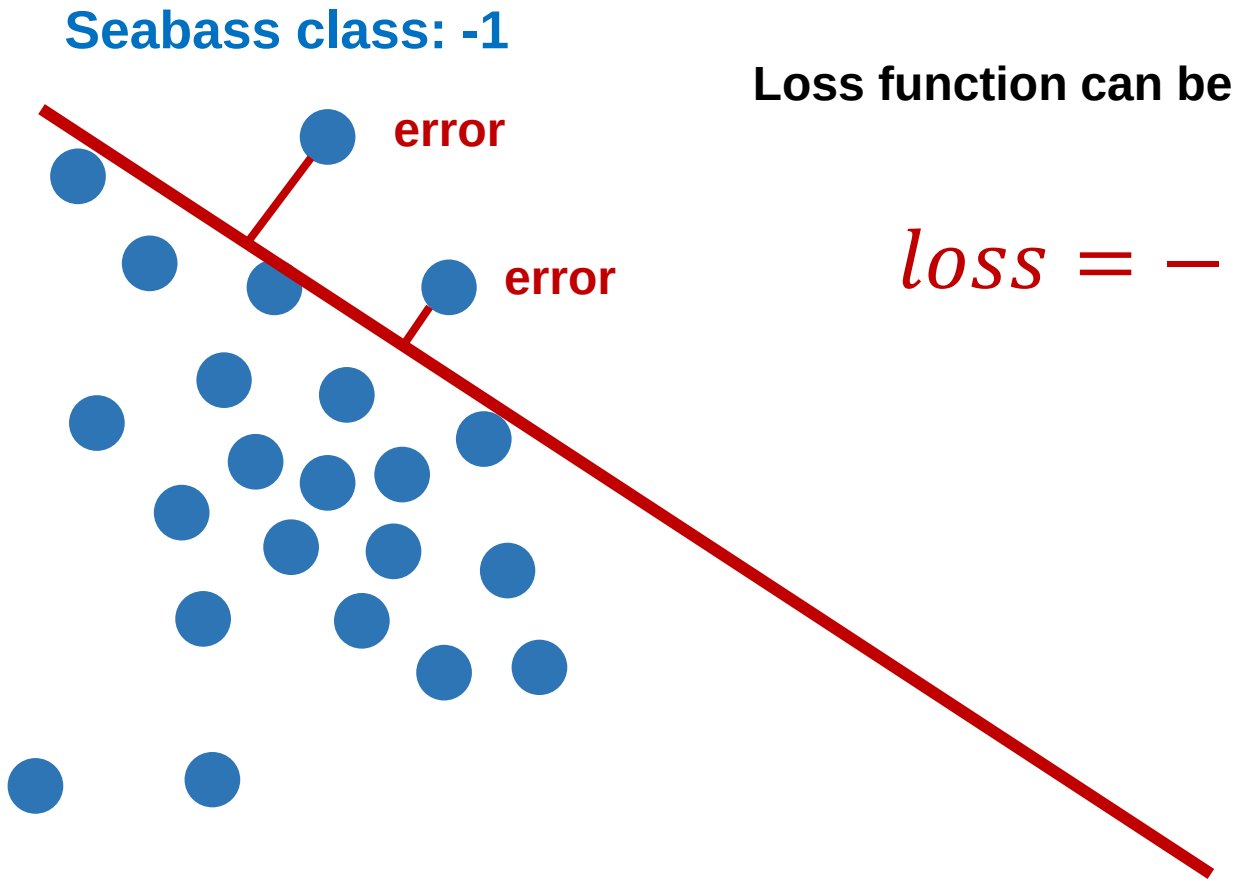
Sum of distances of all errors

$$\text{distance} = \sum \frac{-\text{y} \cdot (w \cdot x + b)}{||w||}$$

Norm is positive, no impact on results

Perceptron

Minimize the error in classification



Loss function can be defined as:

$$loss = -\sum y \cdot (w \cdot x + b)$$

Perceptron

The goal is to find **w and b** that minimize loss

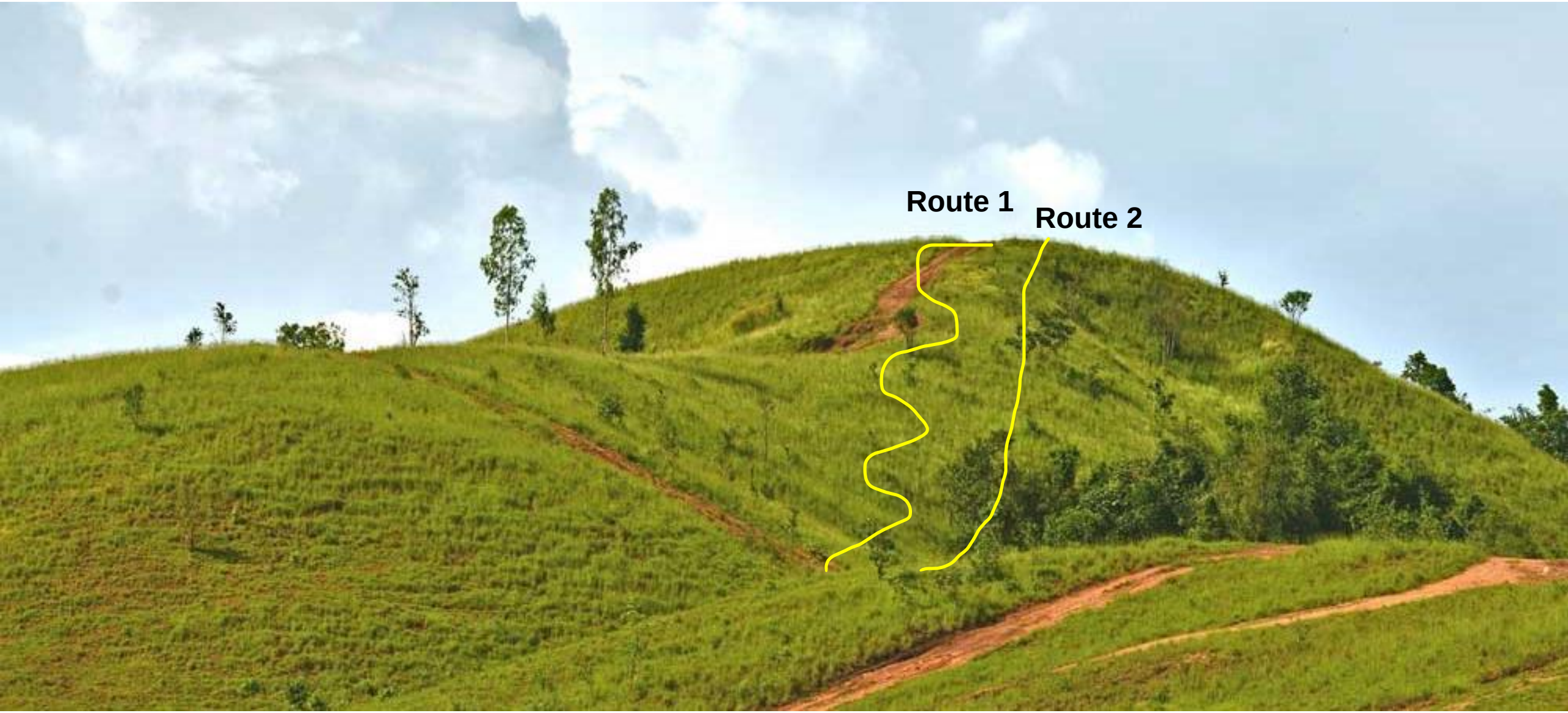
$$loss = -\sum y \cdot (w \cdot x + b)$$

损失函数

Training label

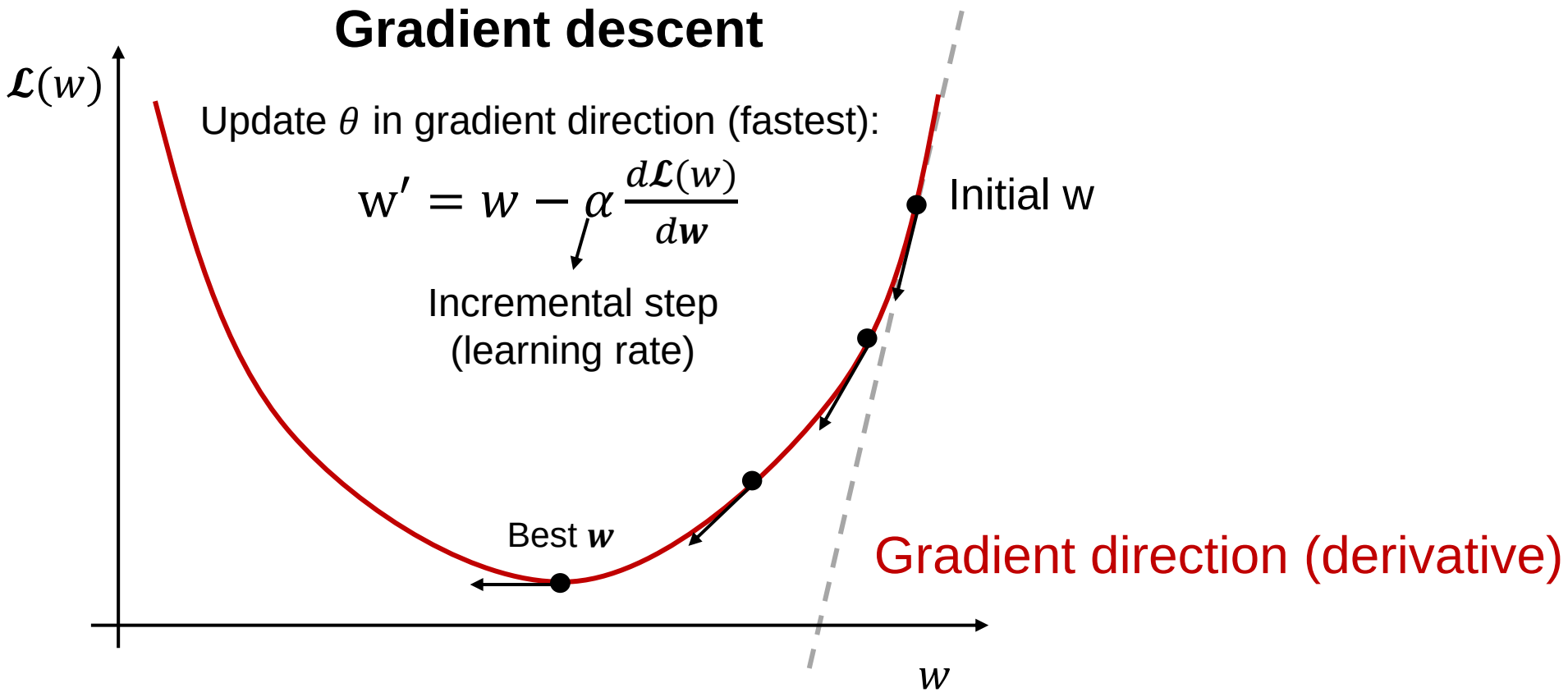
Training data

Gradient descent optimization



Gradient descent optimization

Update the coefficients along the gradient direction of loss function is the fastest path to achieve the minimum!!!



Perceptron

The gradient of loss is calculated by (partial) derivative

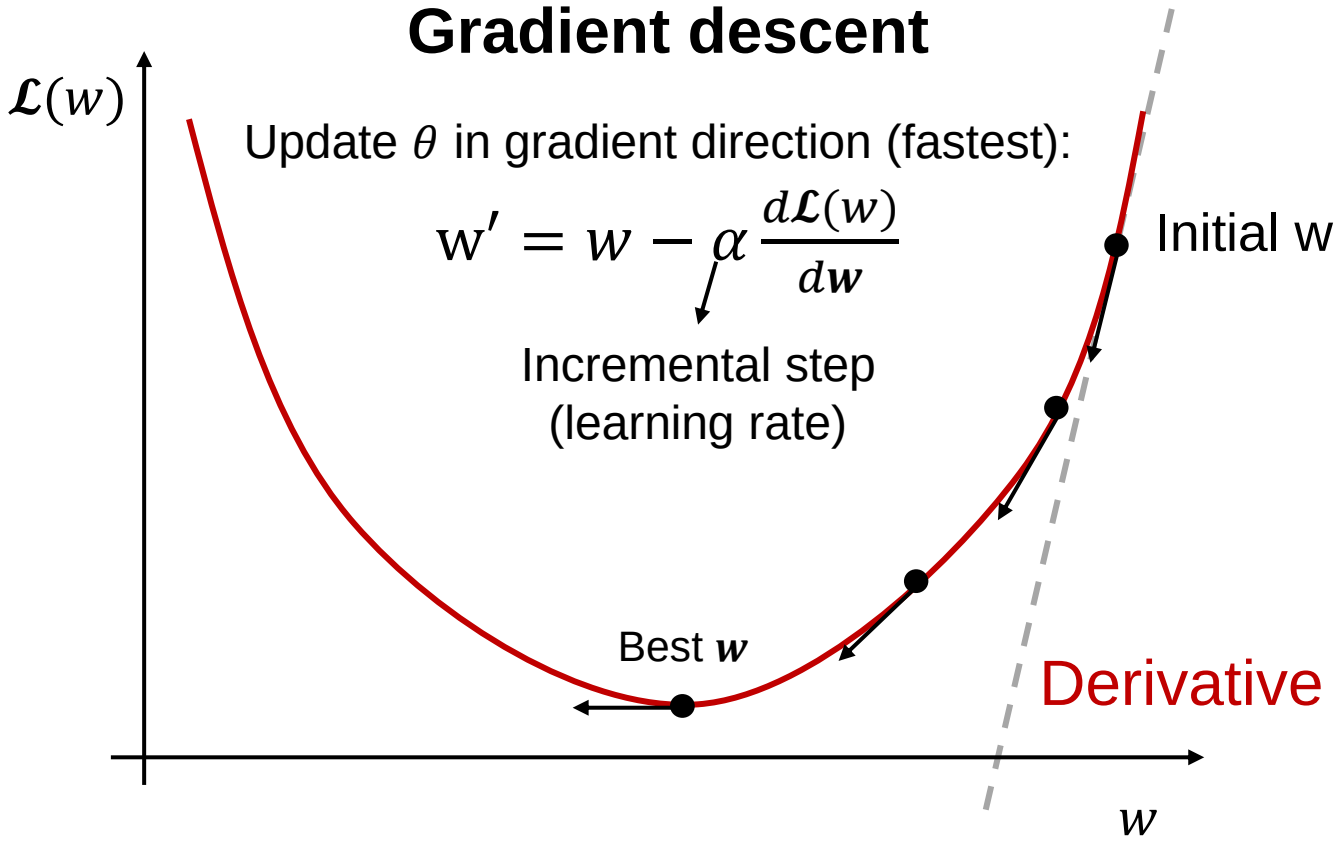
$$\boldsymbol{loss} = -\sum \boldsymbol{y} \cdot (\boldsymbol{w} \cdot \boldsymbol{x} + \boldsymbol{b})$$

$$\nabla_w L(w, b) = -\sum y_i \cdot x_i$$

$$\nabla_b L(w, b) = -\sum y_i$$

Perceptron

Update the coefficients step-wise along the gradient direction



$$w \leftarrow w + \alpha \cdot \nabla_{\mathrm{w}} \mathrm{L}(\mathrm{w}, \mathrm{b})$$

$$b \leftarrow b + \alpha \cdot \nabla_{\mathrm{b}} \mathrm{L}(\mathrm{w}, \mathrm{b})$$

α $(0 < \alpha < 1)$ learning rate

Pros and cons of perceptron

Pros:

- Simplicity
- Generally applicable for various tasks

Cons:

- Data must be linearly separable
- No guarantee on non-convex problem

Content

1. Basic classification (KNN & perceptron)
2. **Basics of CNN (deep learning!)**
3. Applications of CNN

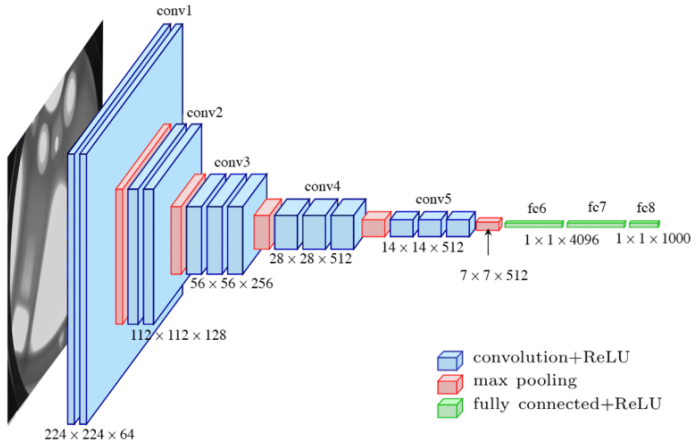
A basic introduction of CNN

CNN basics

(start with a concrete example)

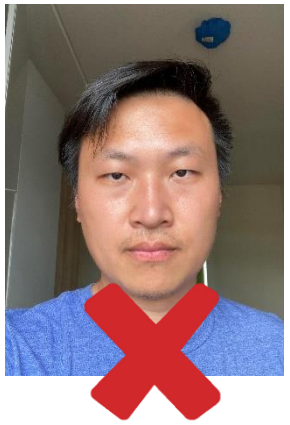
CNN applications

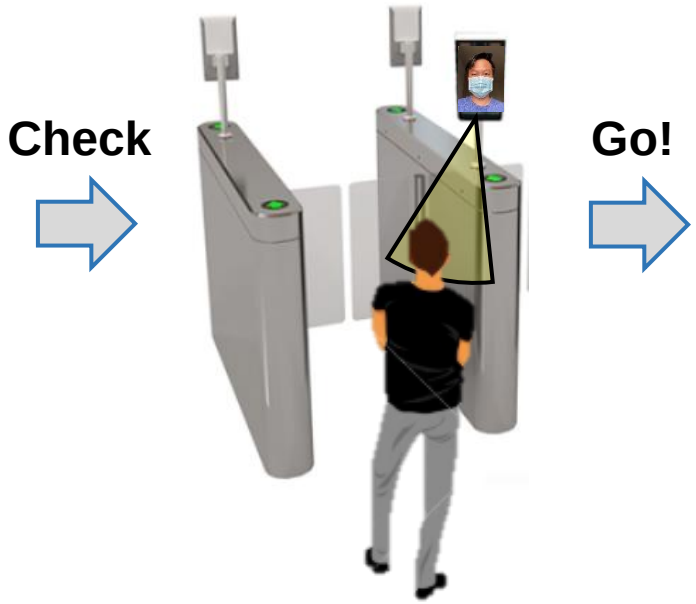
(general ideas in image and video processing)



Assuming the case of facial mask check

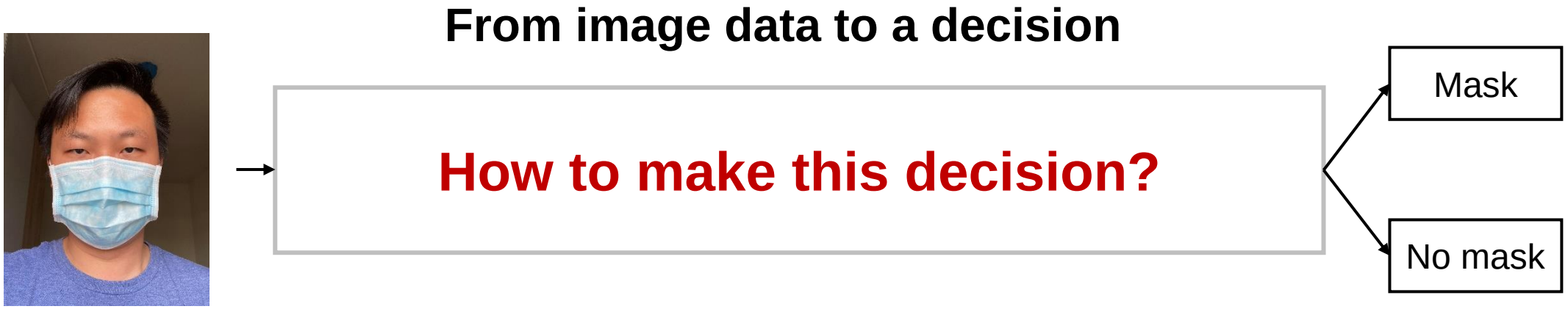
- You must wear a **face mask** to enter the school!!!





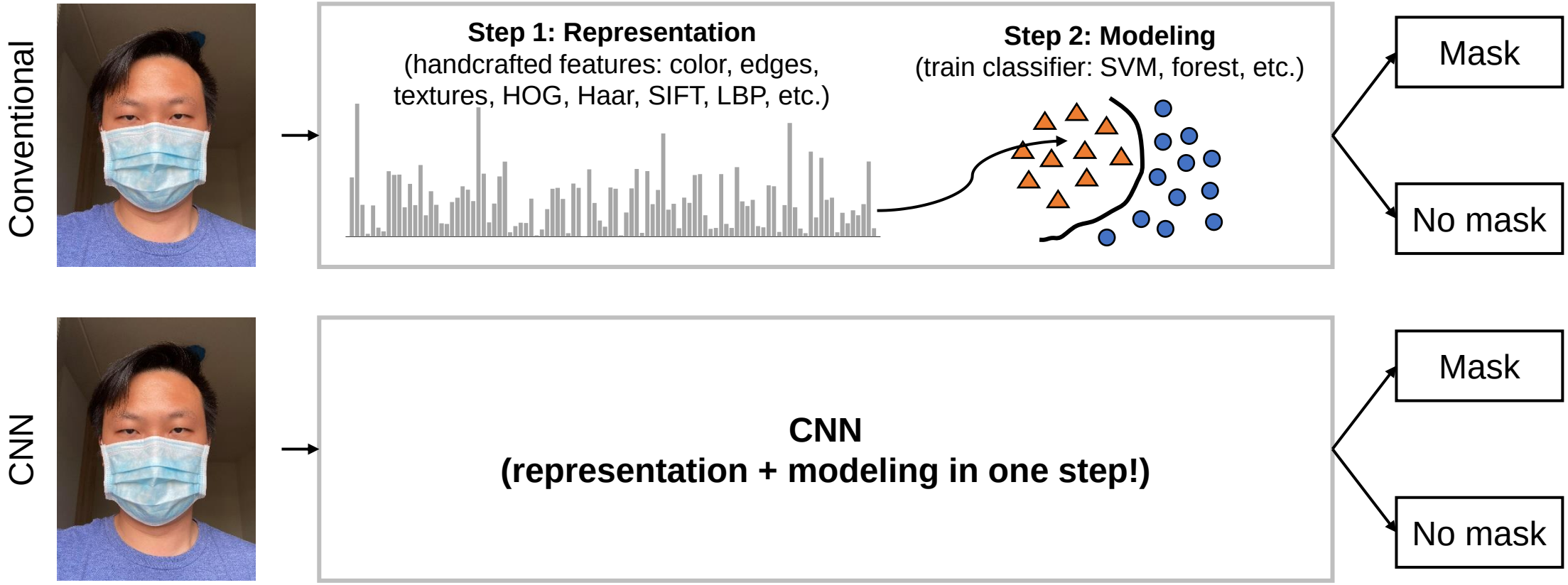
Goal

- Camera-based classification of faces with/without mask



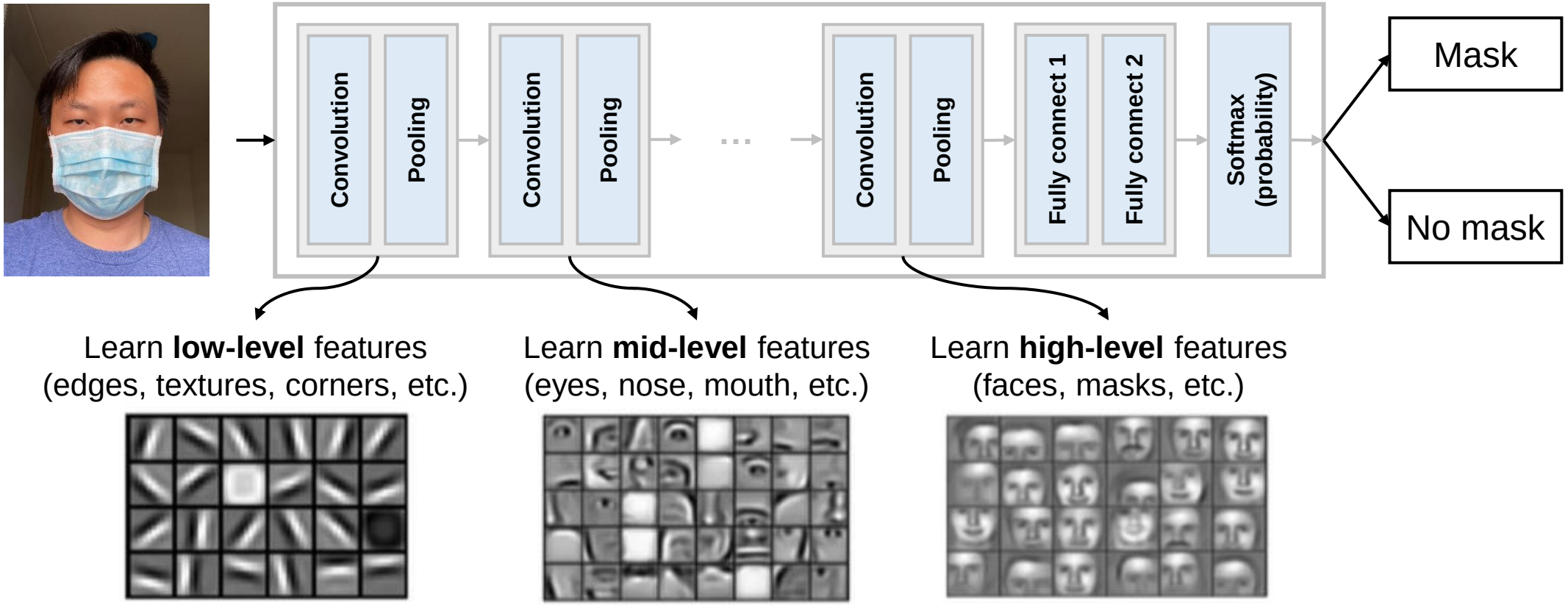
How?

- From conventional machine learning to CNN



What is CNN?

- Overview of a basic CNN architecture



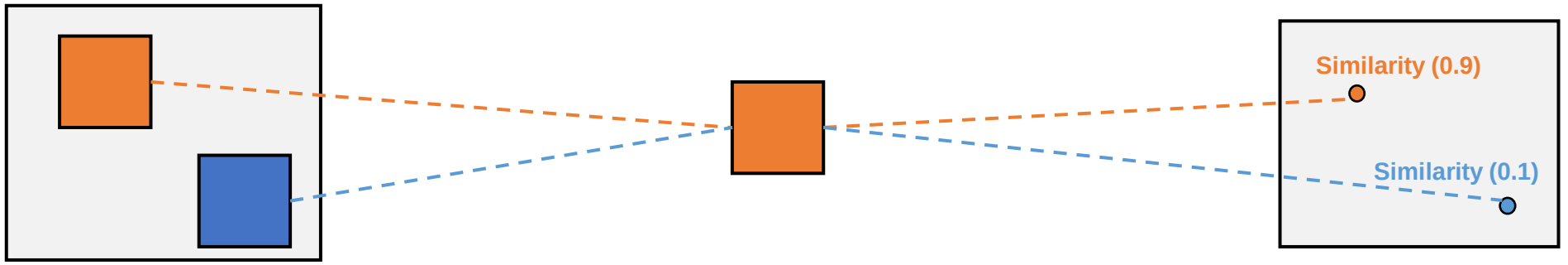
Overview of CNN basics

- Why convolution?
- Convolutional layer
- Pooling layer
- Fully-connected layer
- CNN training
- A minimum CNN example

Why convolution?

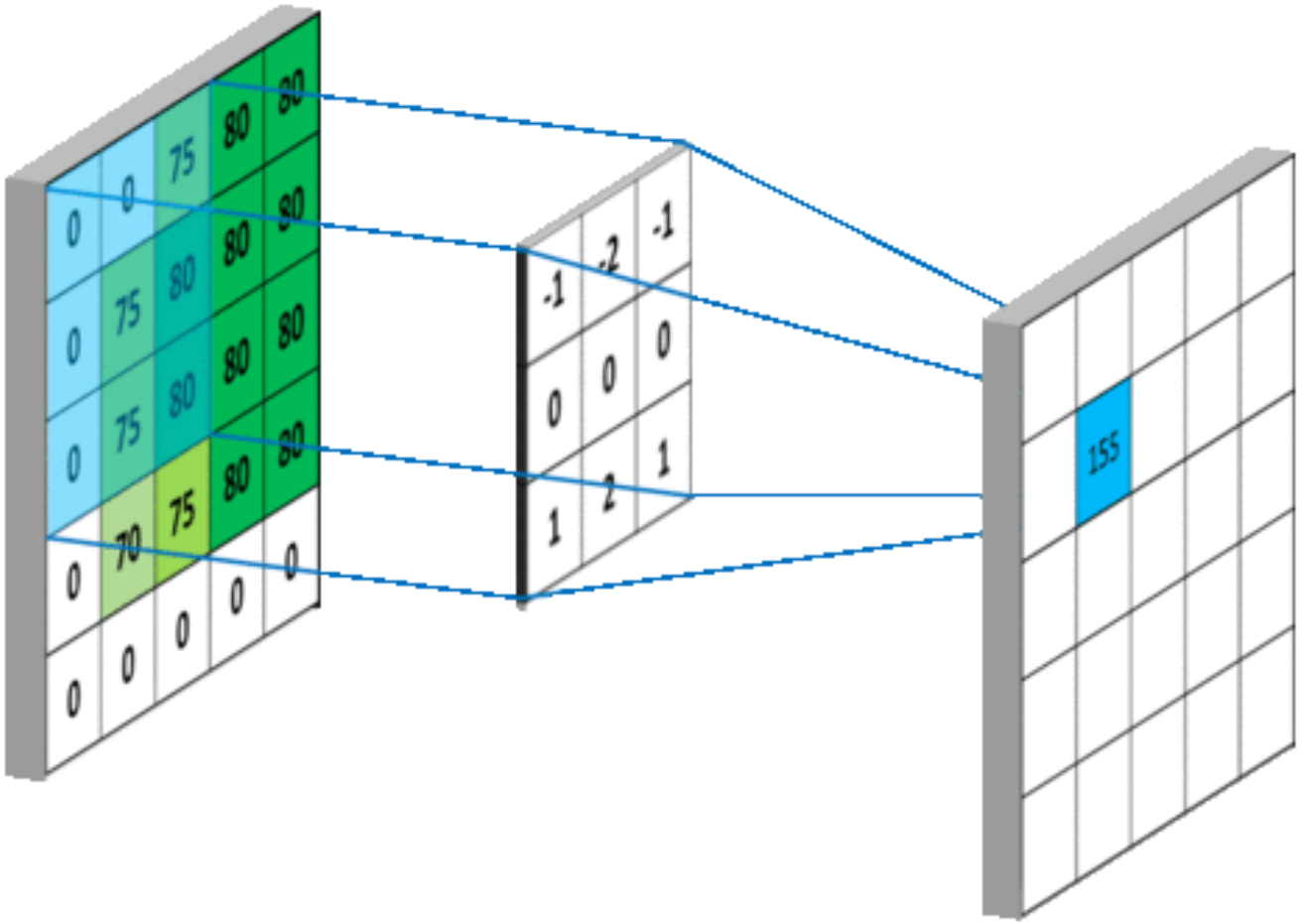
Why convolution?

- Convolution in CNN is actually "**cross correlation**" without kernel flipping!



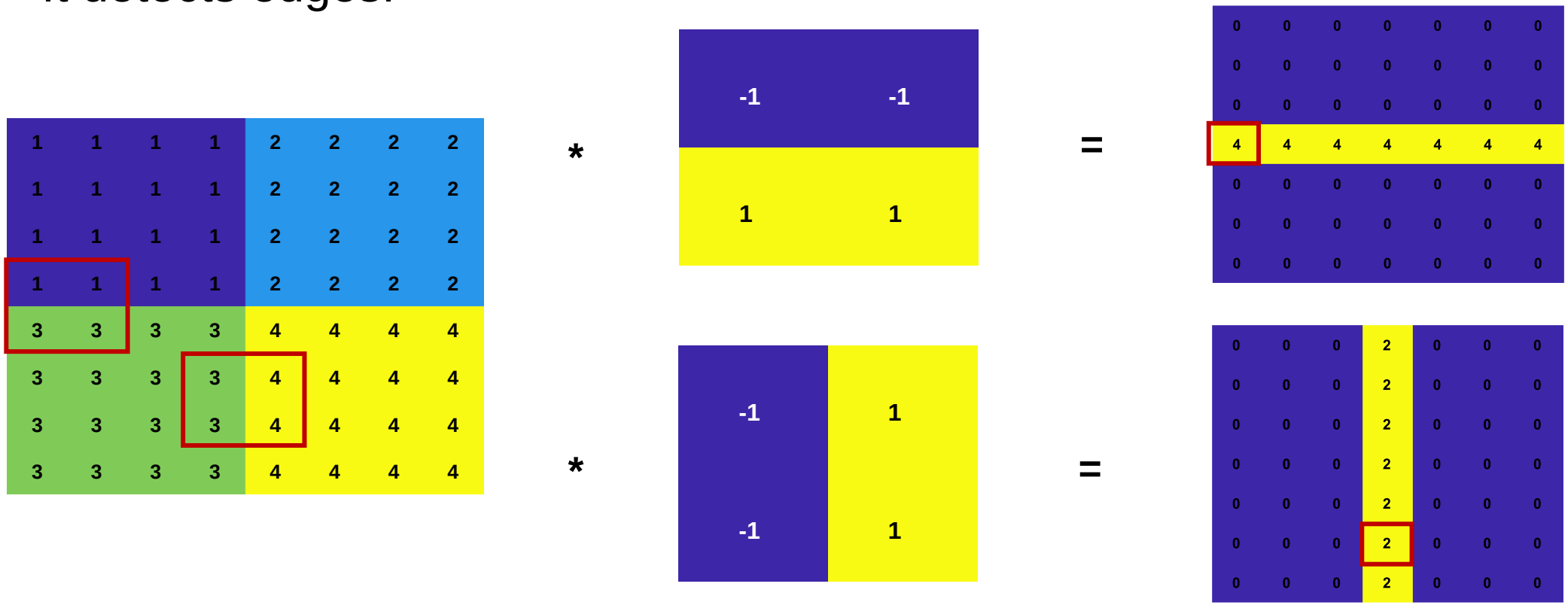
Essence of cross correlation: ***it measures the similarity (inner product) of data and knowledge at different shifted locations.*** *If data has a pattern similar to the knowledge, it will give a high response (correlated).*

Why convolution?



Why convolution?

- It detects edges!

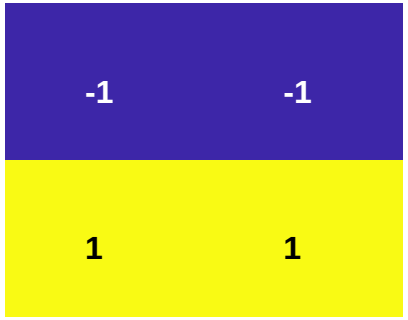


Input image (8x8) Kernels Convolved images

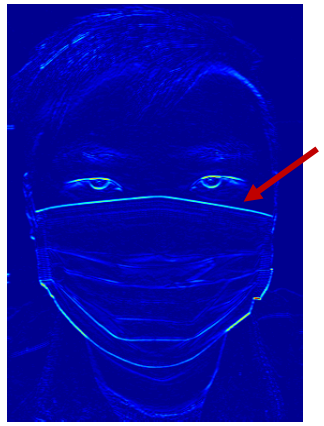
Why convolution?

- It detects edges!

*

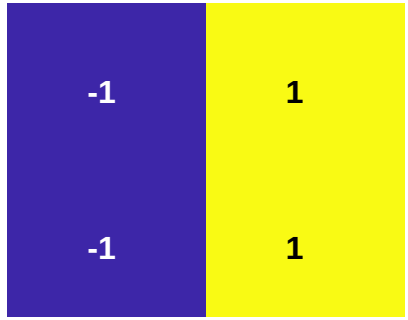


=



Horiz. edges

*



=

Verti. edges

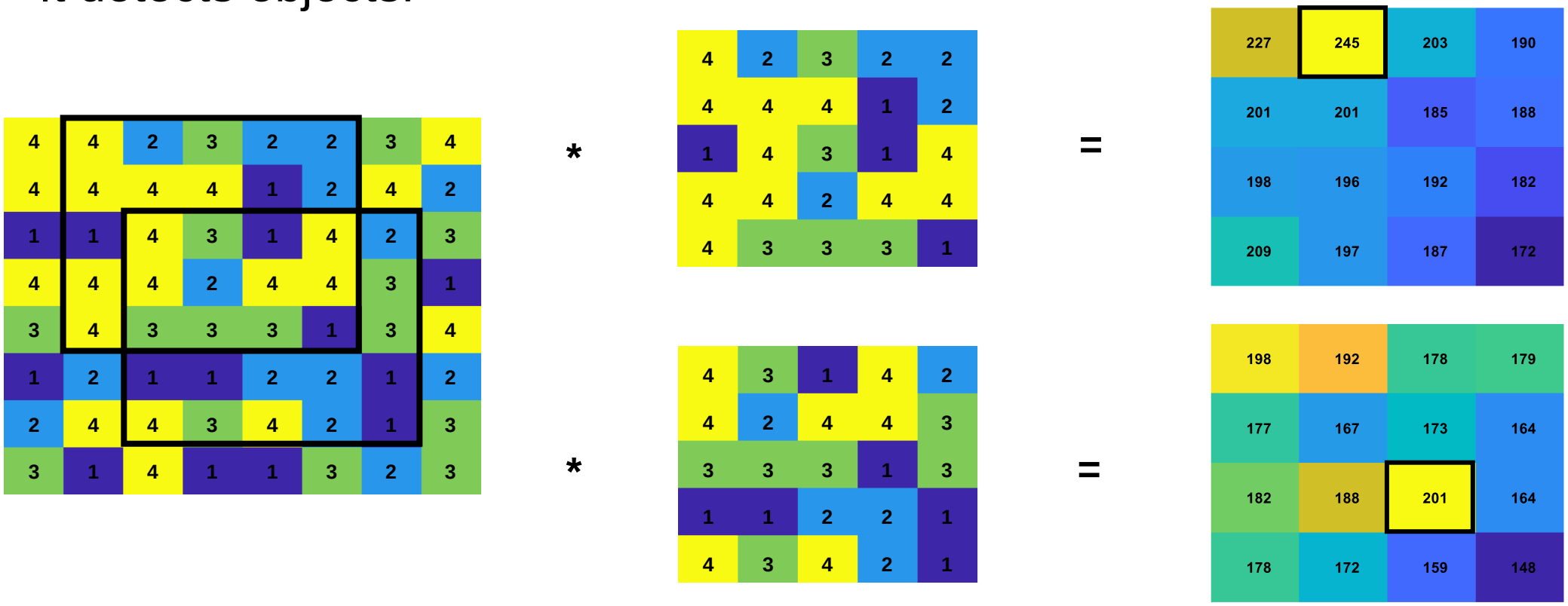
Input image

Kernels

Convolved images

Why convolution?

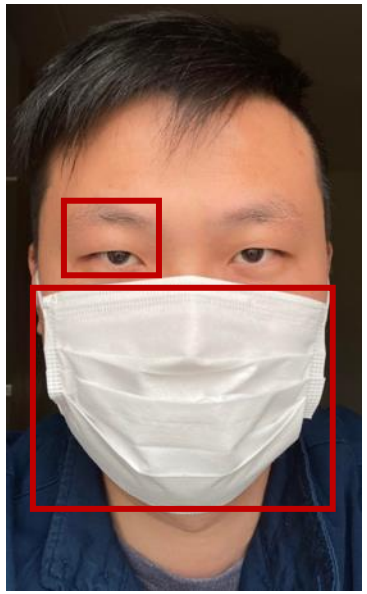
- It detects objects!



Input image (8x8) Kernels (larger) Convolved images

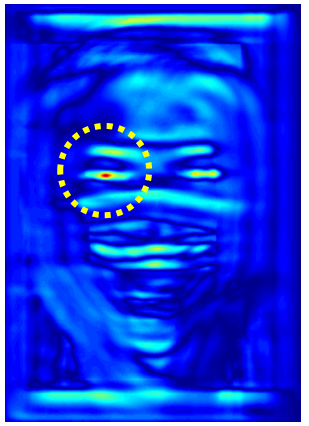
Why convolution?

- It detects objects!



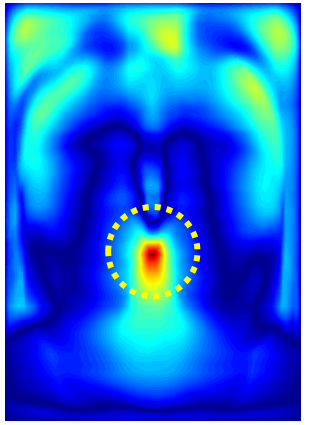
*

=



*

=



Input image

Kernels (larger)

Convolved images

Why convolution?

- It detects objects!

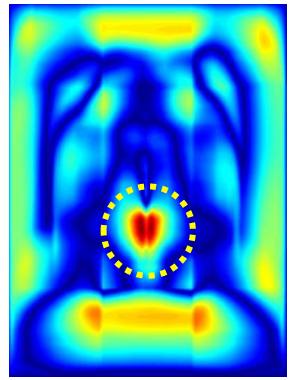
Input image

*

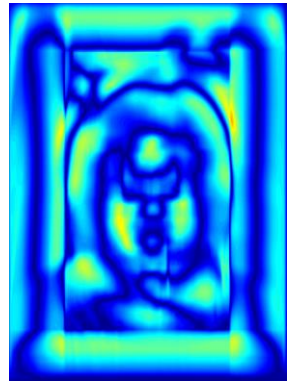
Kernel (learned)

*

=



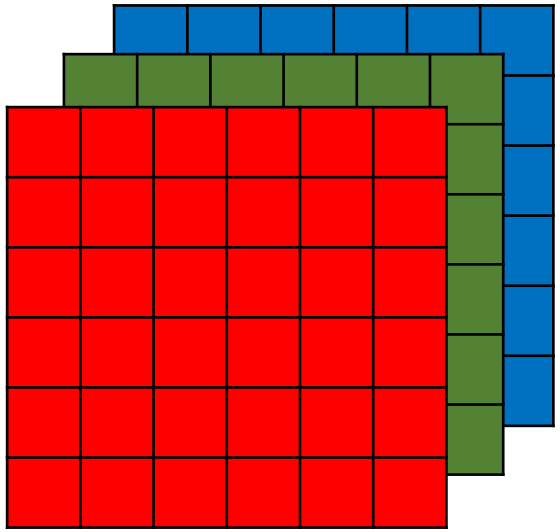
=



Convolved images

Why convolution?

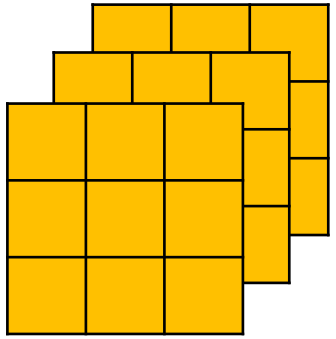
- A single kernel



$6 \times 6 \times 3$

Image size: $n \times n \times c$

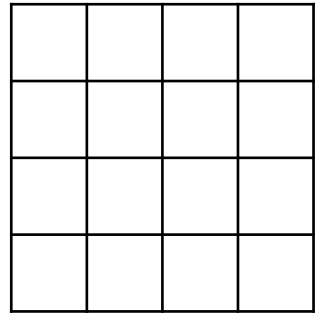
*



$3 \times 3 \times 3$

Kernel size: $k \times k \times c$
(stride (s) = 1)

=

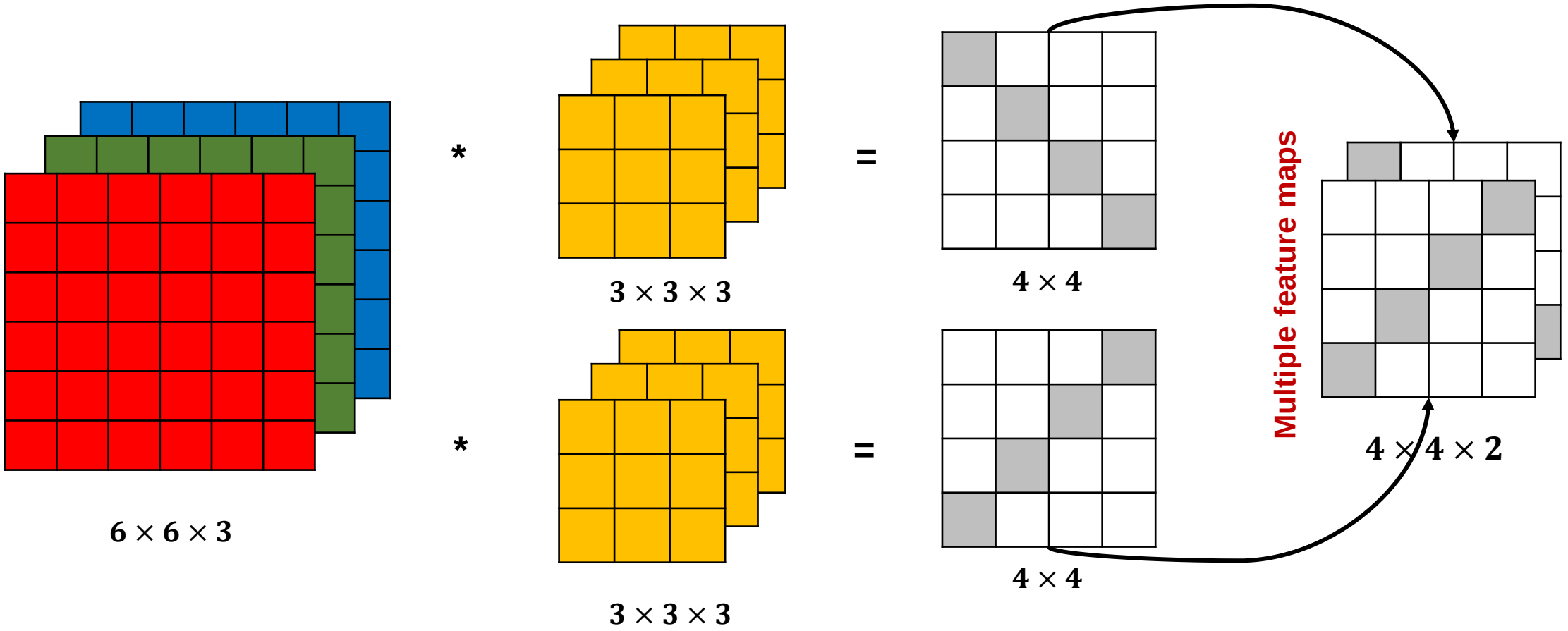


4×4

$$\left[\frac{n-k}{s}+1\right] \times \left[\frac{n-k}{s}+1\right]$$

Why convolution?

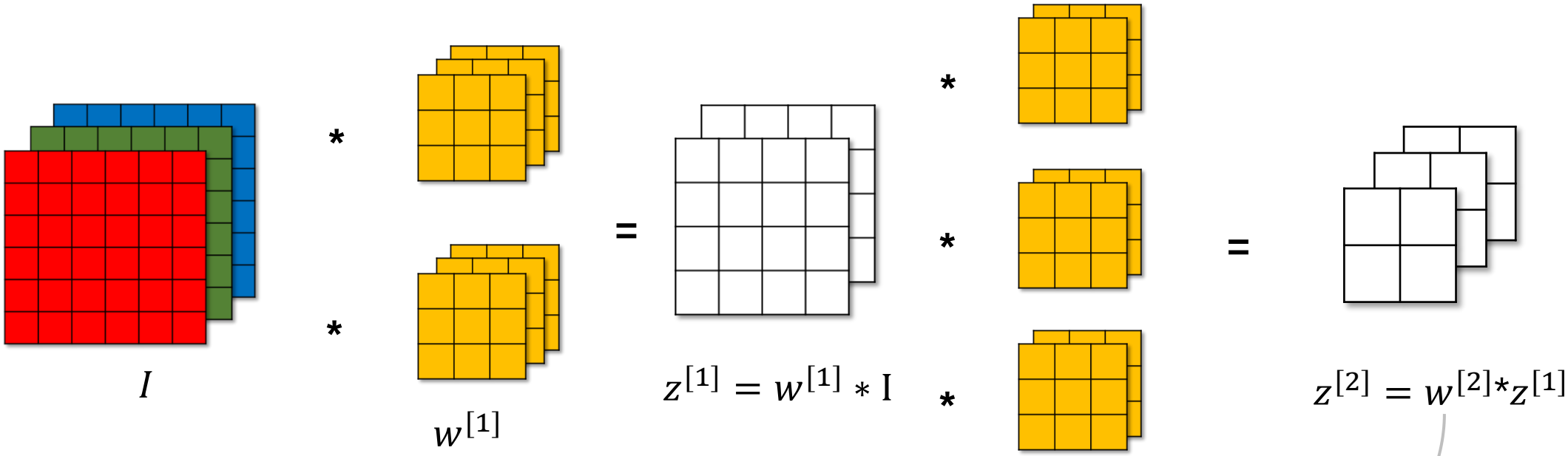
- Multiple kernels (one is not enough!)



Convolutional layer

Convolutional layer

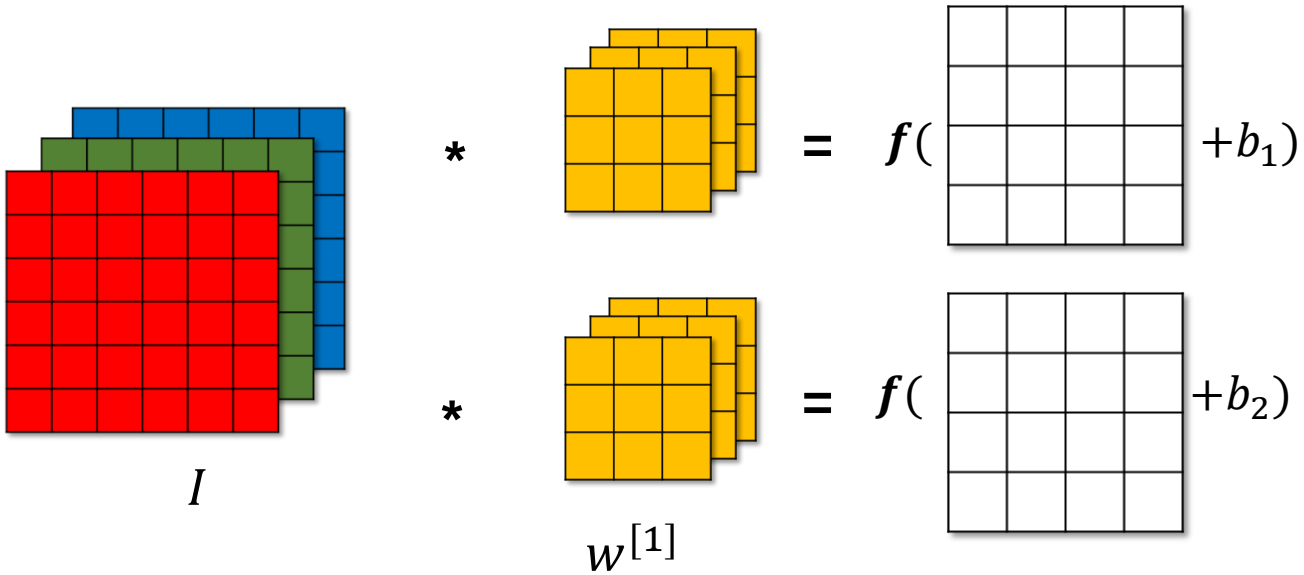
- Problem: linear operation between layers is redundant!



$$z^{[2]} = w^{[2]} * z^{[1]} = w^{[2]} * \left(w^{[1]} * I\right) = \left(\boldsymbol{w^{[1]} * w^{[2]}}\right) * I!$$

Convolutional layer

- Solution: add non-linear activation!



Non-linear activation (f):

$$z^{[1]} = f(w^{[1]} * \mathbf{I} + \mathbf{b}^{[1]})$$

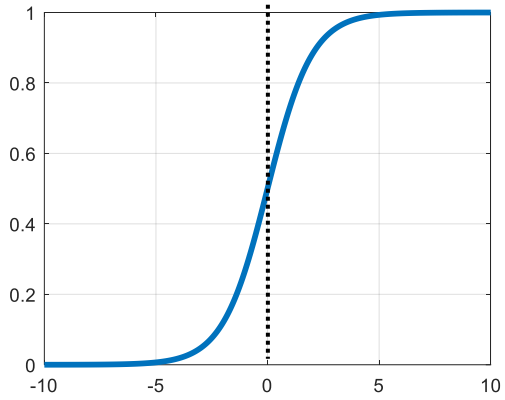
Kernel weights (scaling)

Bias (translation)

Convolutional layer

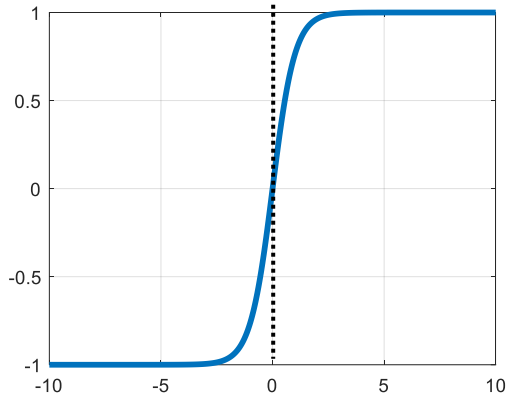
- What non-linear activations to use?

Sigmoid



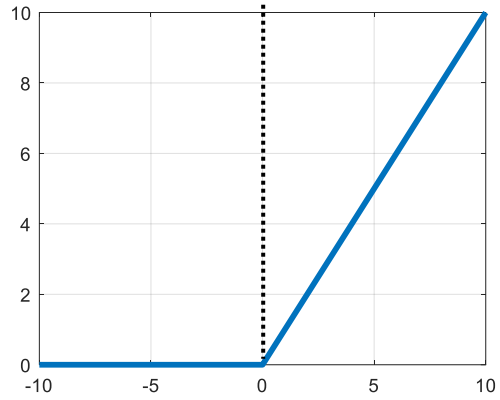
$$y = \frac{1}{1 + e^{-x}}$$

Tanh



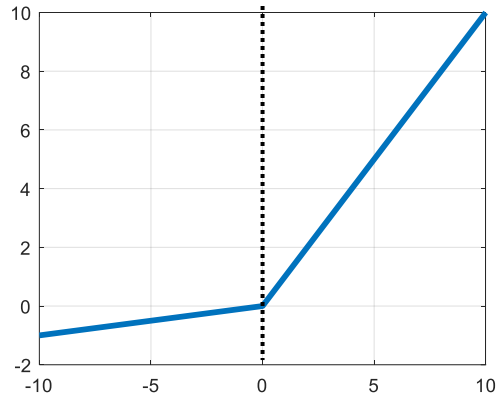
$$y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

ReLu



$$y = \max(0, x)$$

Leaky ReLu



$$y = \max(0.1x, x)$$

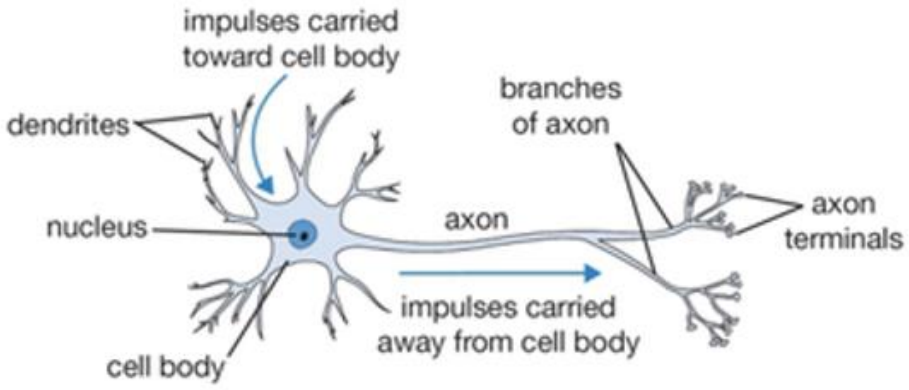
Gradient vanishing (dy <= 1)

ReLu is most popular, its derivate is 0 or 1, easy for back propagation training.

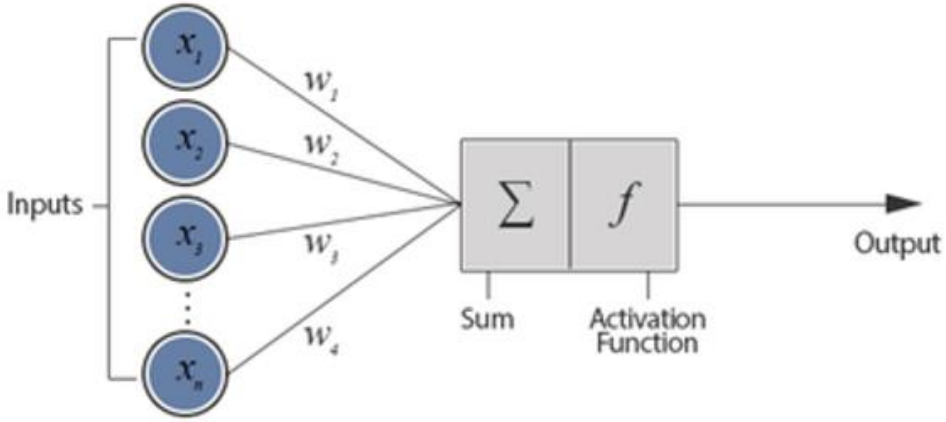
Convolutional layer

- Why non-linear activation works? Biological reasons?...

Biological neuron



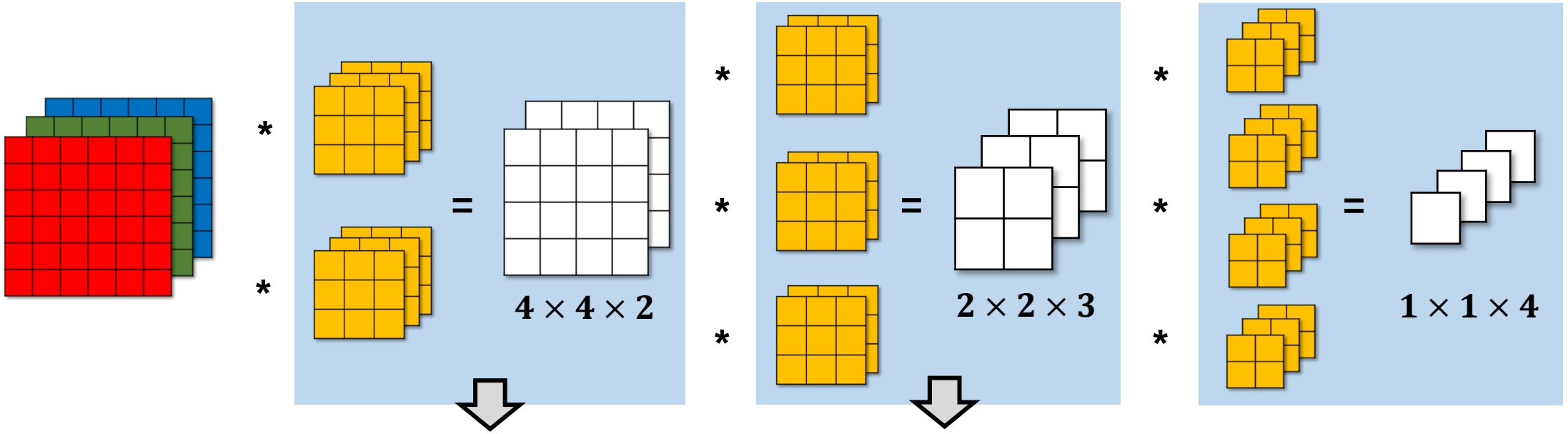
Artificial neuron

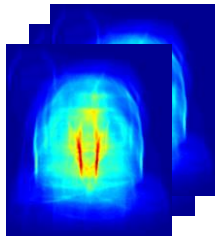


"Only when the stimulation is strong enough, it will activate the neuron and transmit the signal. Possibly because our biological system is not linearly reacting to all stimulus. Otherwise our body will be overwhelmed with all kinds of stimulus…"

Convolutional layer

- Multiple convolutional layers: why this is useful?



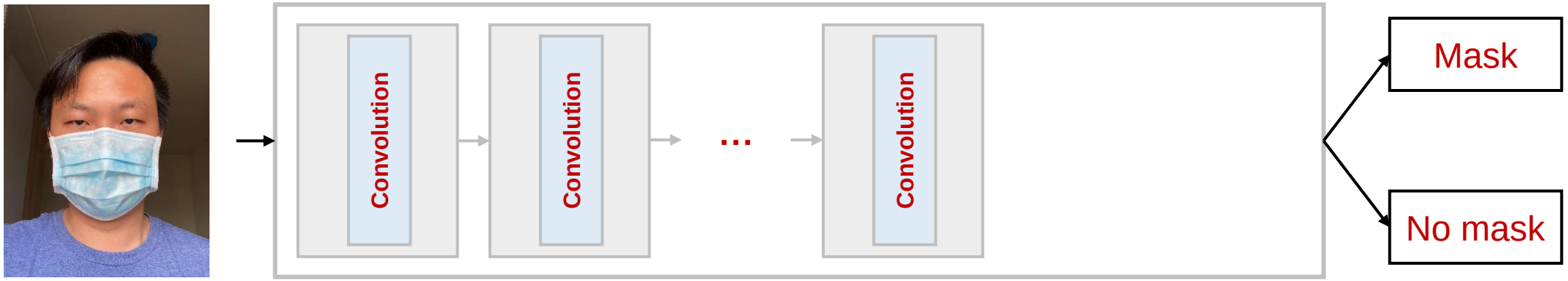


It's a multi-scale feature extraction:

- Different receptive fields (from local to global)
- Different contexts (from edges to objects)

Convolutional layer (in CNN architecture)

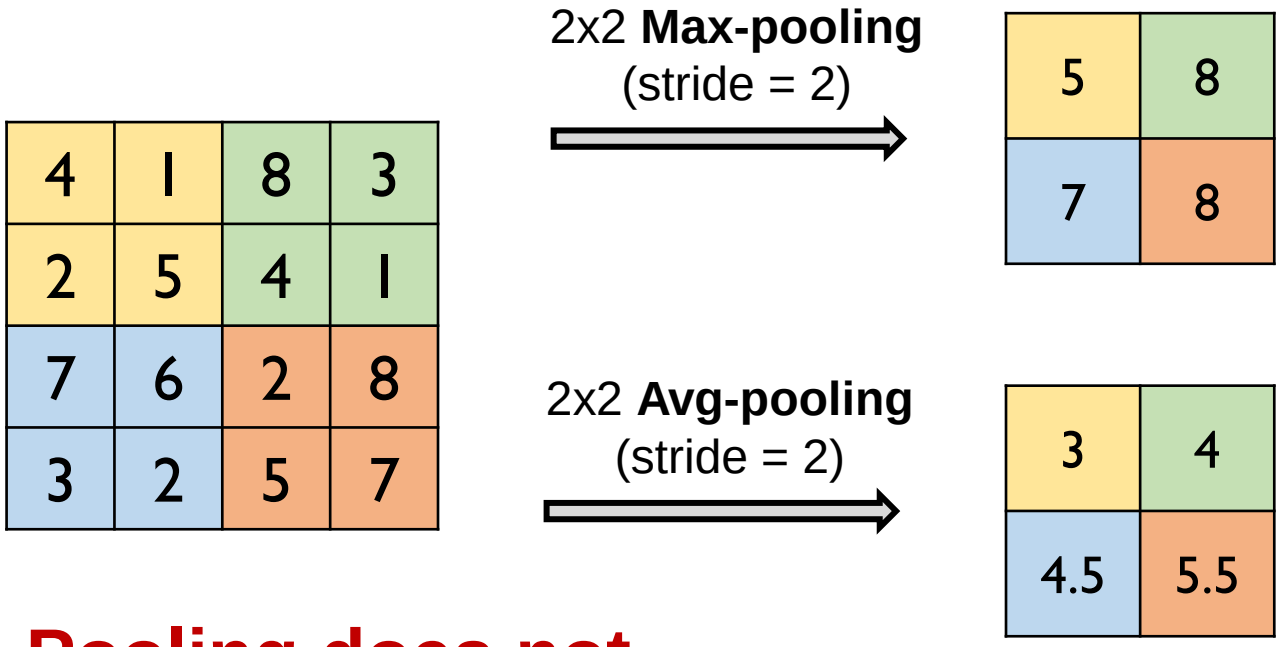
- Now we put **convolutional layers** in CNN



Pooling layer

Pooling layer

- Use spatial pooling to speed up down-sampling!



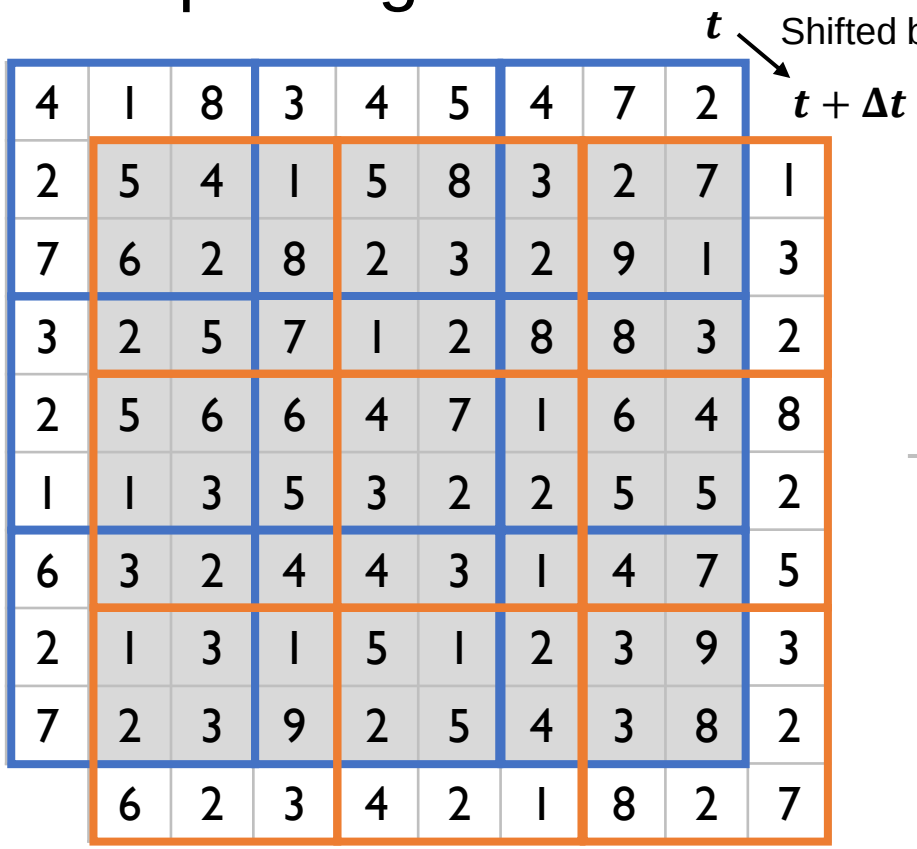
Max pooling is preferred in most cases:

- Highly non-linear
- Salient/high-frequency information (local maxima)
- Translation invariant

Pooling does not require parameters!

Pooling layer

- Max pooling



3x3 max pooling
(stride = 3)

t

8	8	9
6	7	8
7	9	9

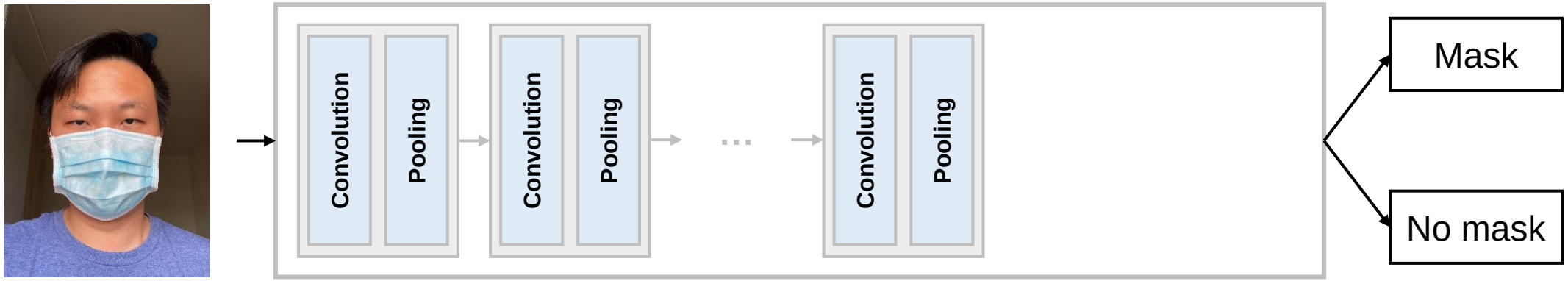
$t + \Delta t$

8	8	9
6	7	8
9	5	9

More or less translation invariant

Pooling layer (in CNN architecture)

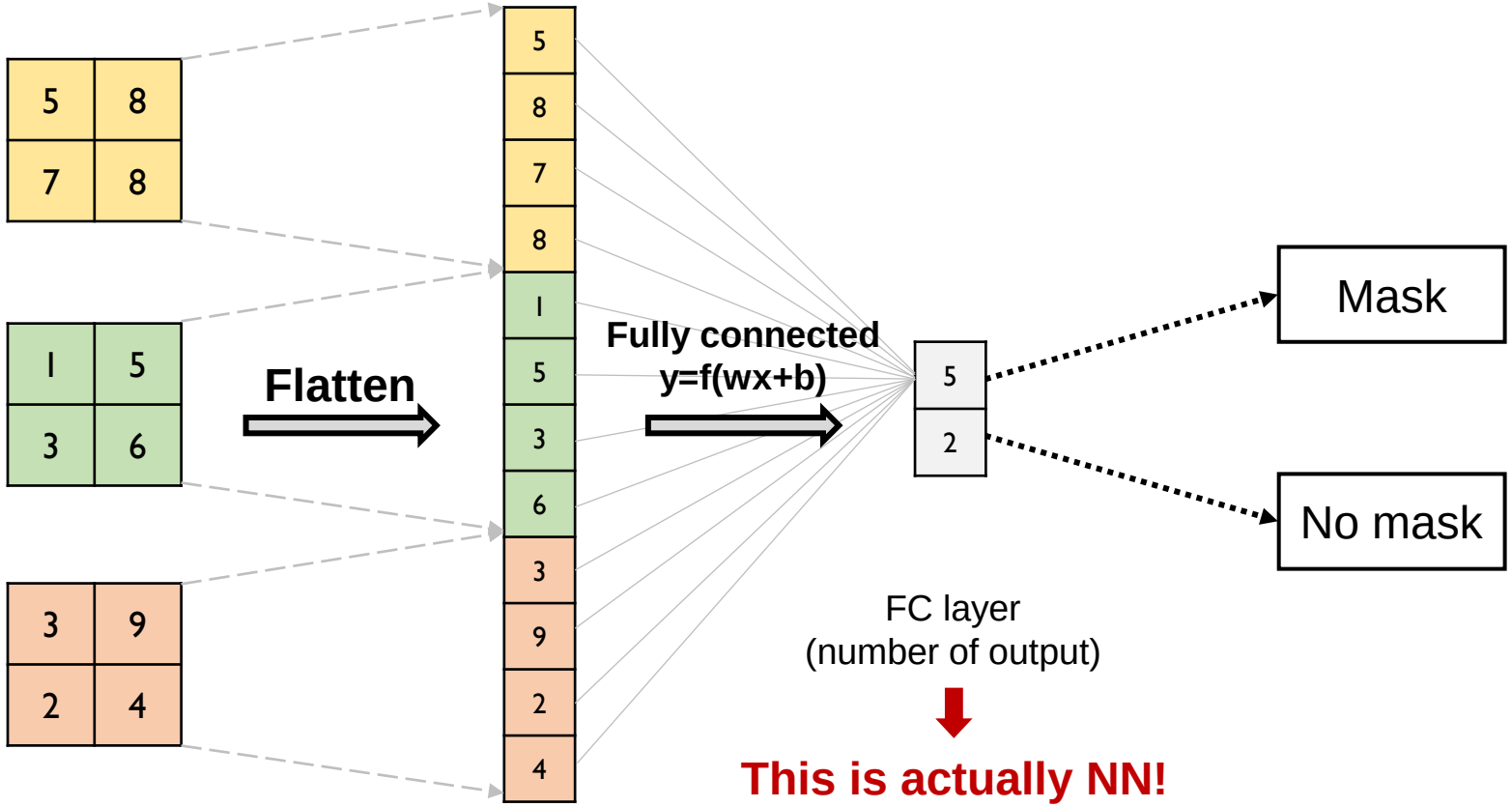
- Now we put **pooling layers** in CNN



Fully-connected layer

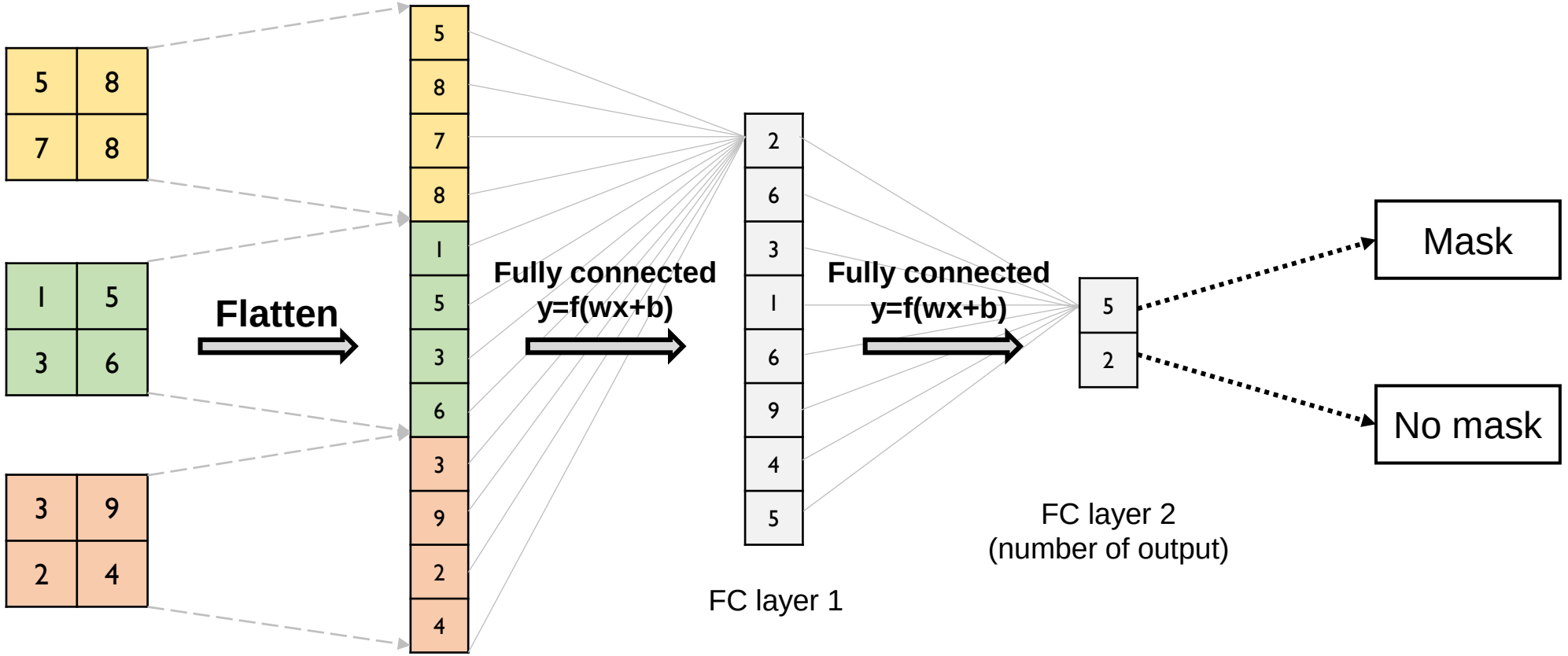
Fully-connected layer

- Convert local 2D maps to a global 1D vector (1 FC layer)



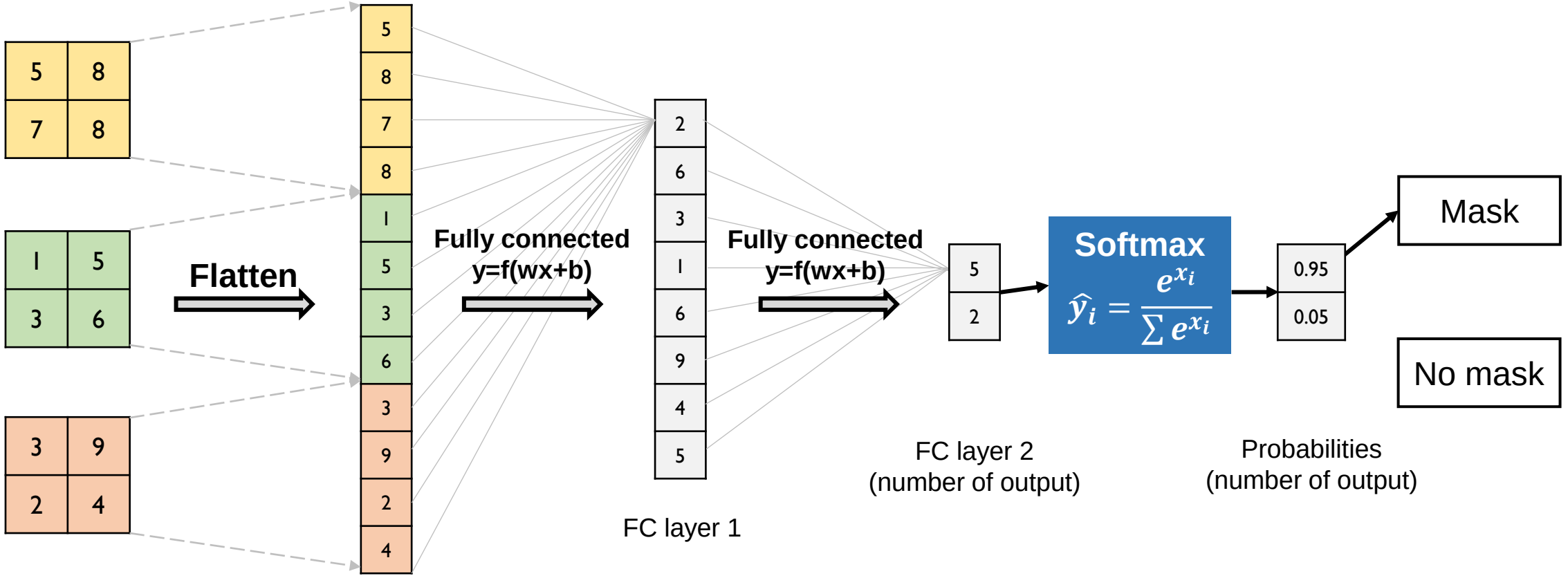
Fully-connected layer

- Convert local 2D maps to a global 1D vector (2 FC layers)



Fully-connected layer

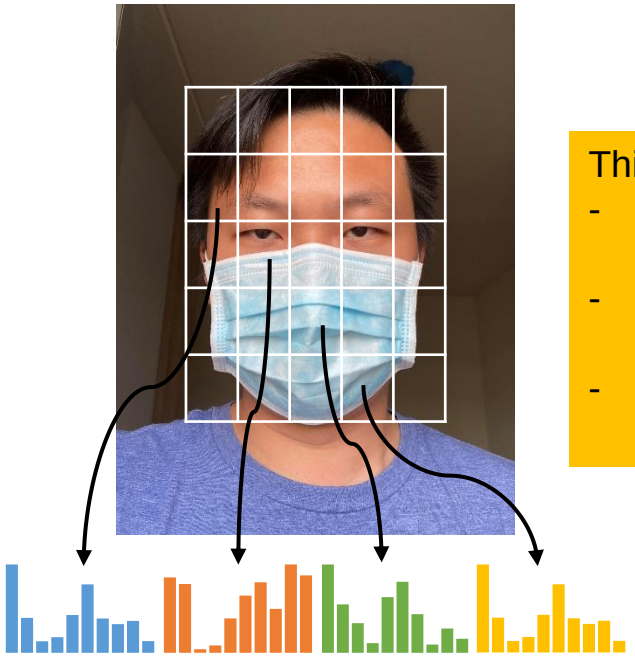
- Convert 1D feature vector to probabilities (Softmax)



Fully-connected layer

- Why fully-connected is useful?

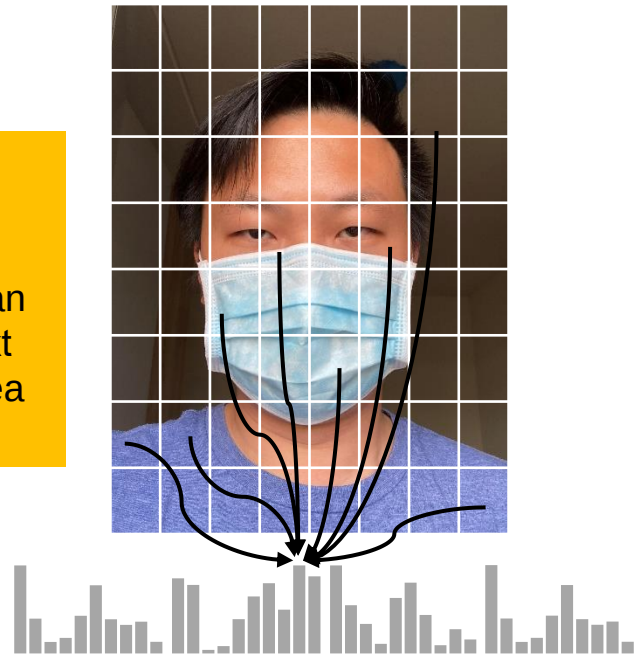
Conventional



Local features
(though features are concatenated, still local)

This is the powerful (terrible?) part of CNN:
- It may use global semantics to make a decision
- It does not even need to see a mask, but can infer a “mask” based on surrounding context
- It gives a global optimal decision, but no idea what is the measurement origin.

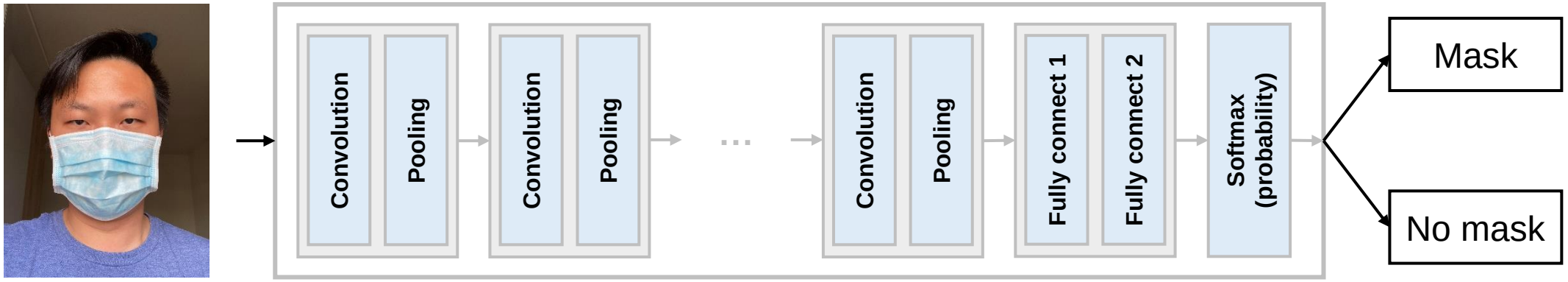
CNN



Fully-connected features
(features inside/outside object are fully exploited)

Fully-connected layer (in CNN architecture)

- Now we put **fully-connected layers** in CNN

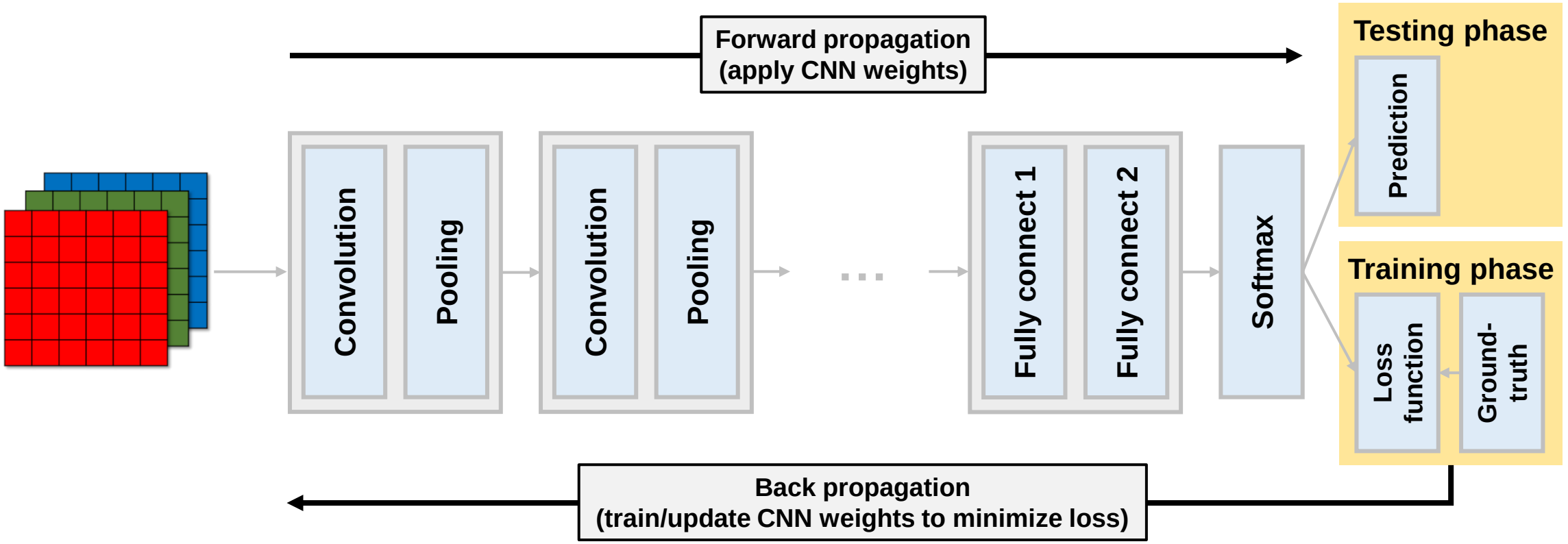


Application-wise it is more or less ready.

But you do not get the CNN weights for free!

How to get CNN weights?

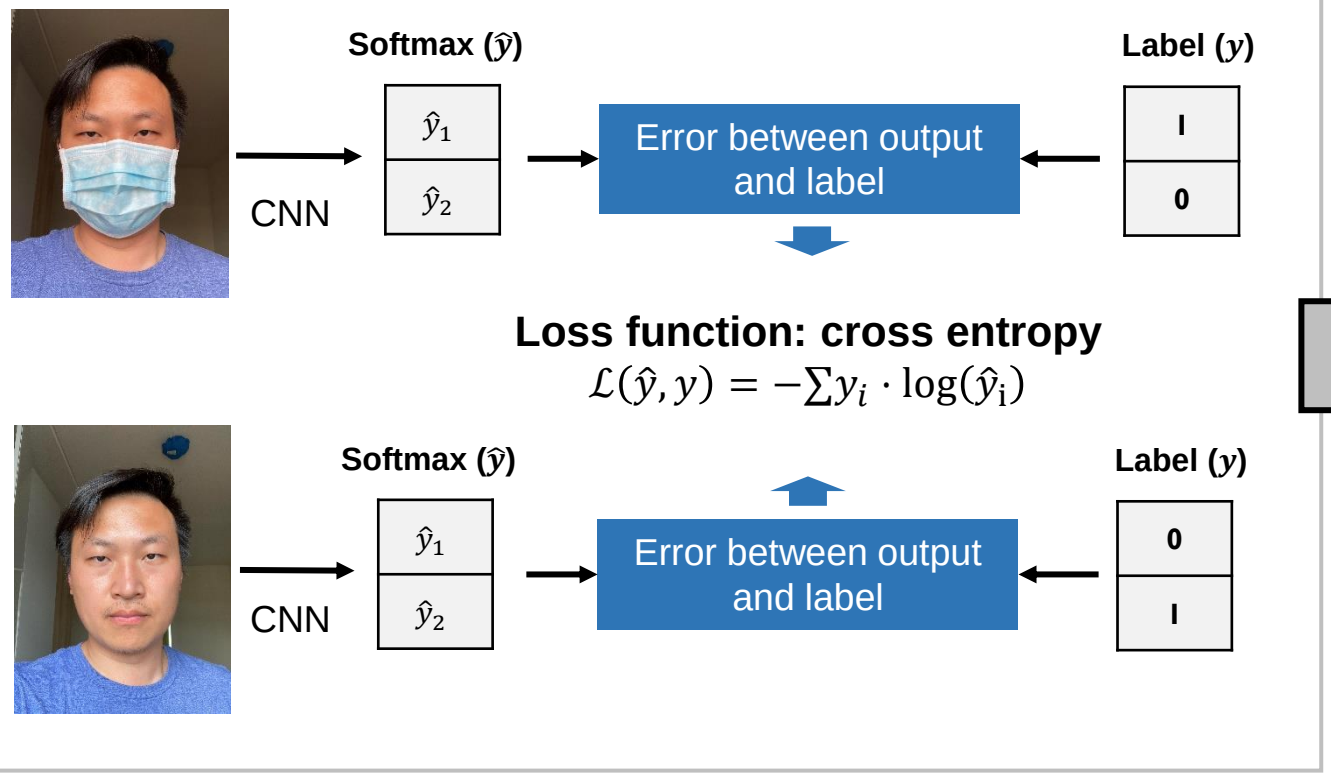
- Train it!!!

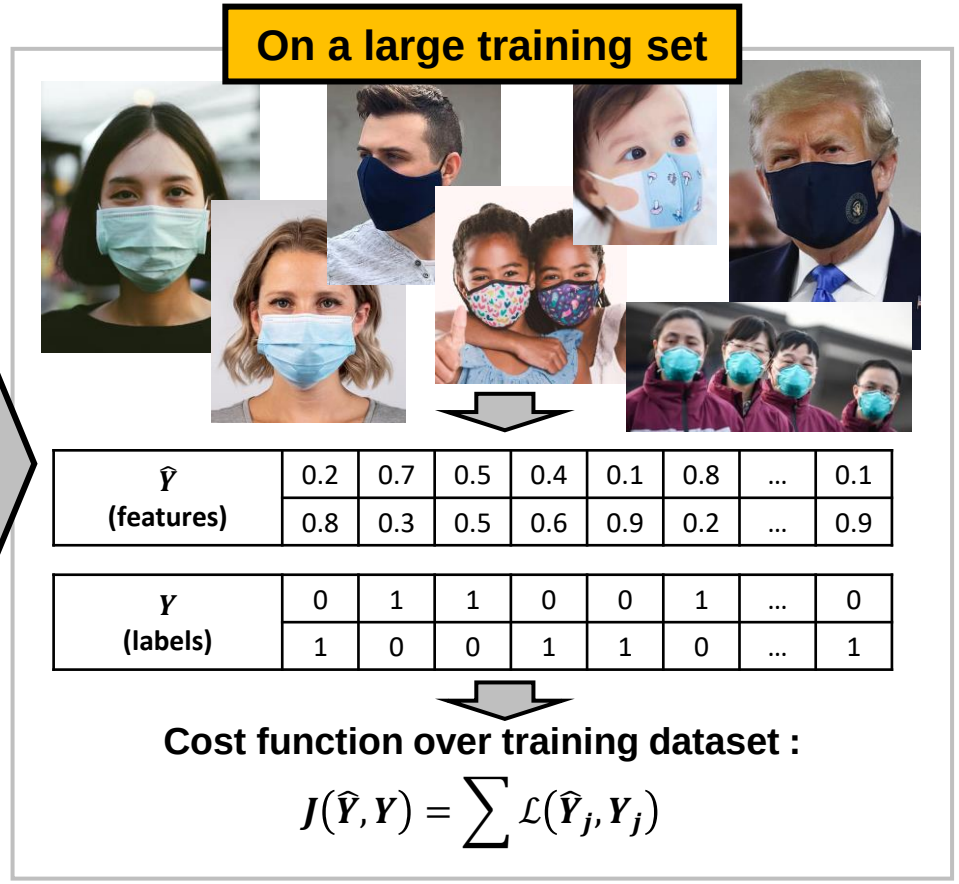


CNN training

Objective function

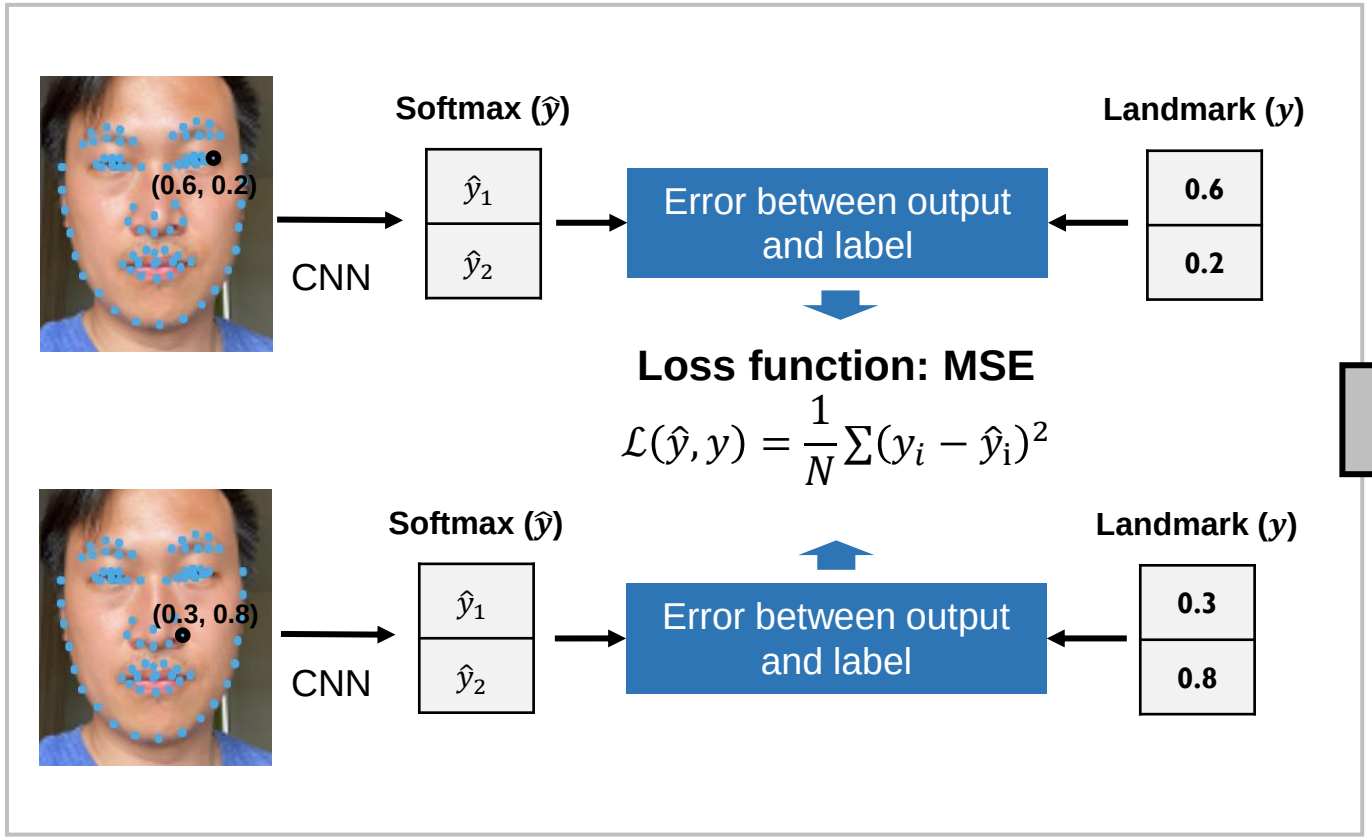
- First define an objective function (e.g. classification)

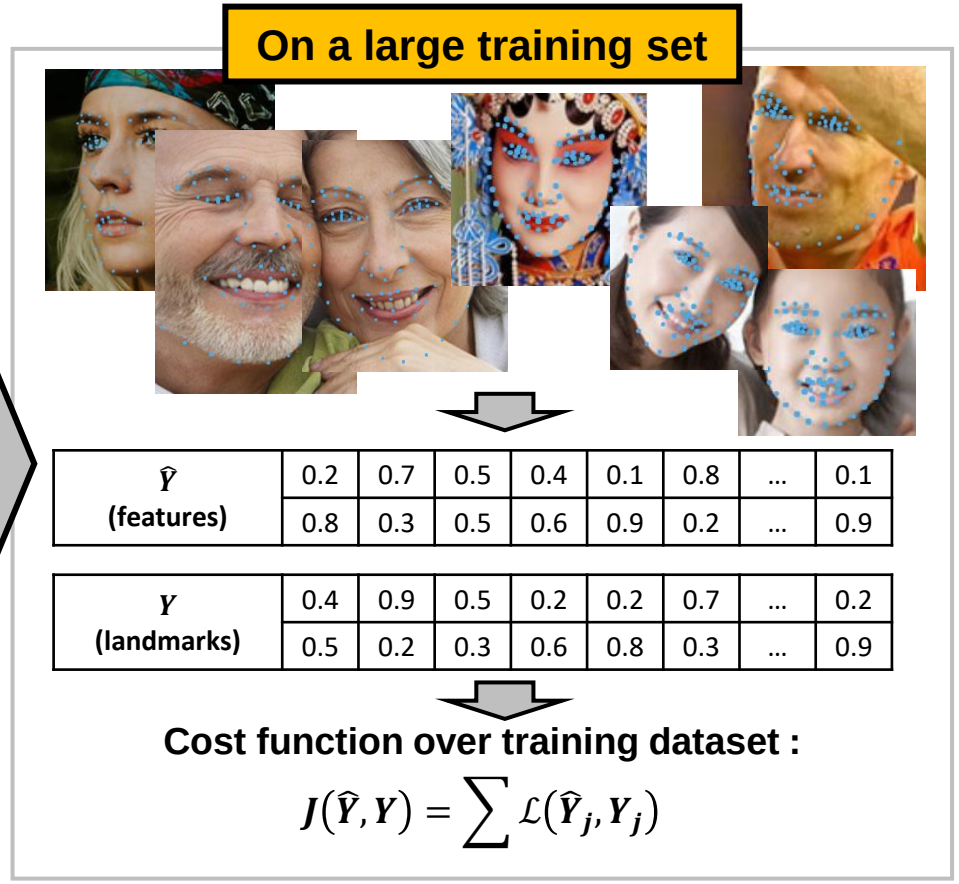




Objective function

- First define an objective function (e.g. regression)





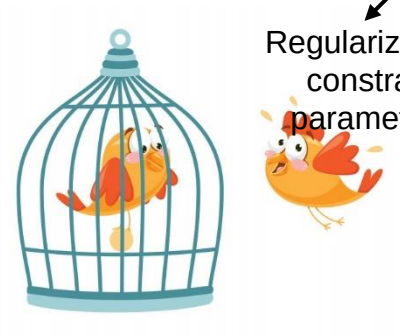
Gradient descent optimization

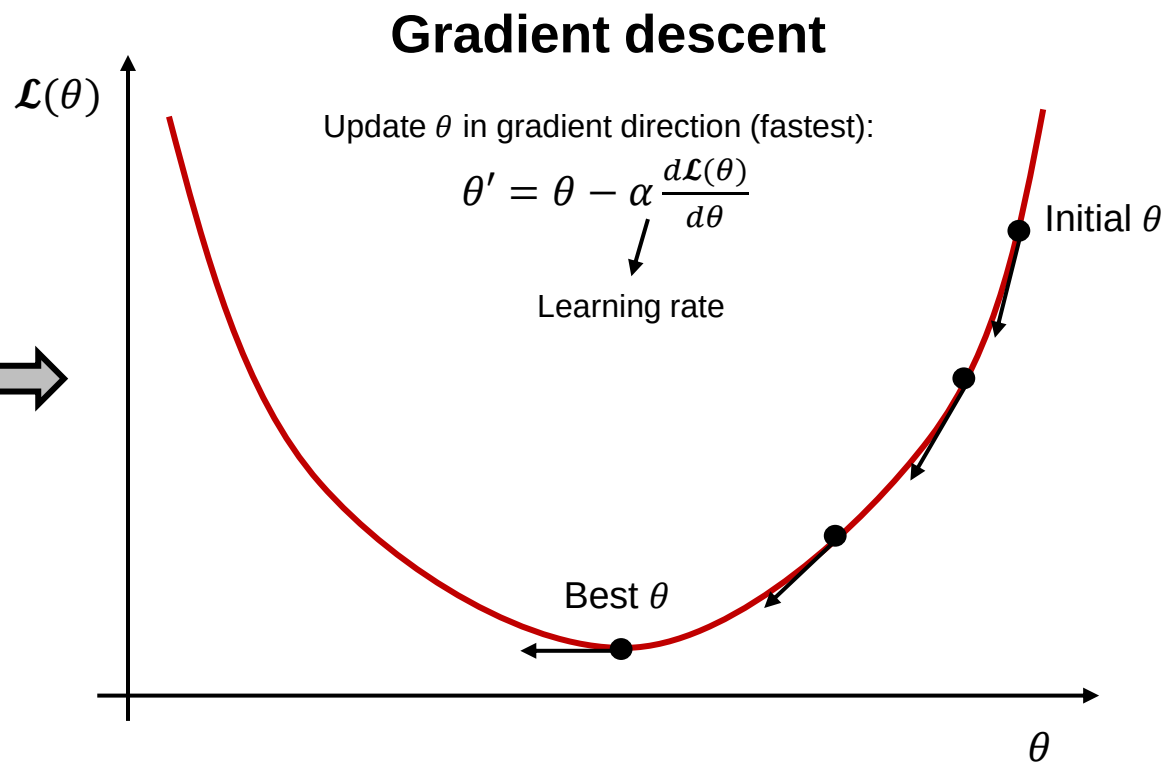
- Train CNN weights to minimize the loss towards objective!

- **CNN parameters**: $\theta = \{w, b\}$, w=weights, b=bias
- **Loss function**: $\mathcal{L}(\boldsymbol{f}(X, \theta), Y)$, X=data, Y=label

Optimize: $\min_{\theta} \sum \mathcal{L}(\boldsymbol{f}(X, \theta), Y) + \lambda \boldsymbol{R}(\theta)$

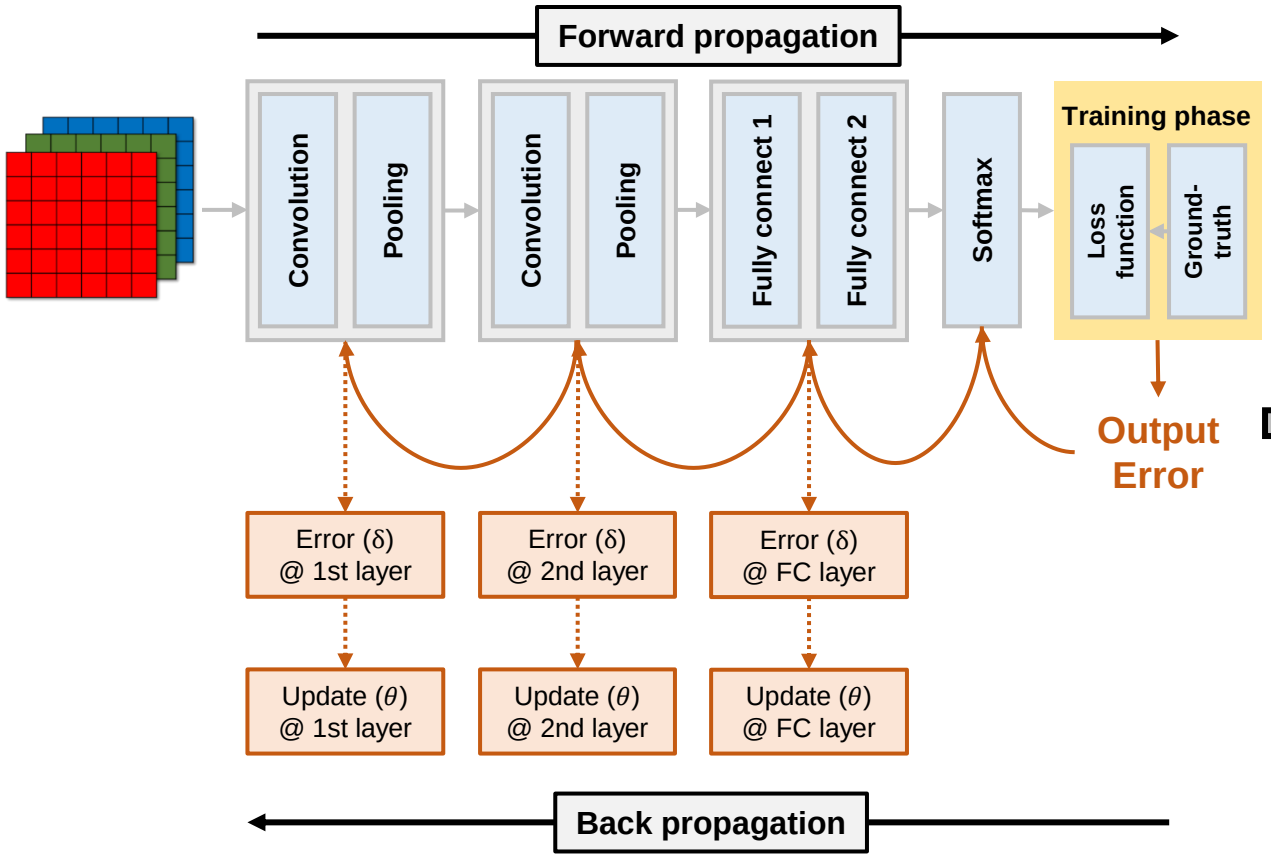
Regularization: constrain parameters





Back propagation

- Propagate error back to layers to update weights per layer.



Initialize CNN model (θ)

Do

- Forward propagation
- Back propagation
  - **Estimate output error**
  - **Propagate error back to each layer**
  - **Update θ per layer (gradient descent)**

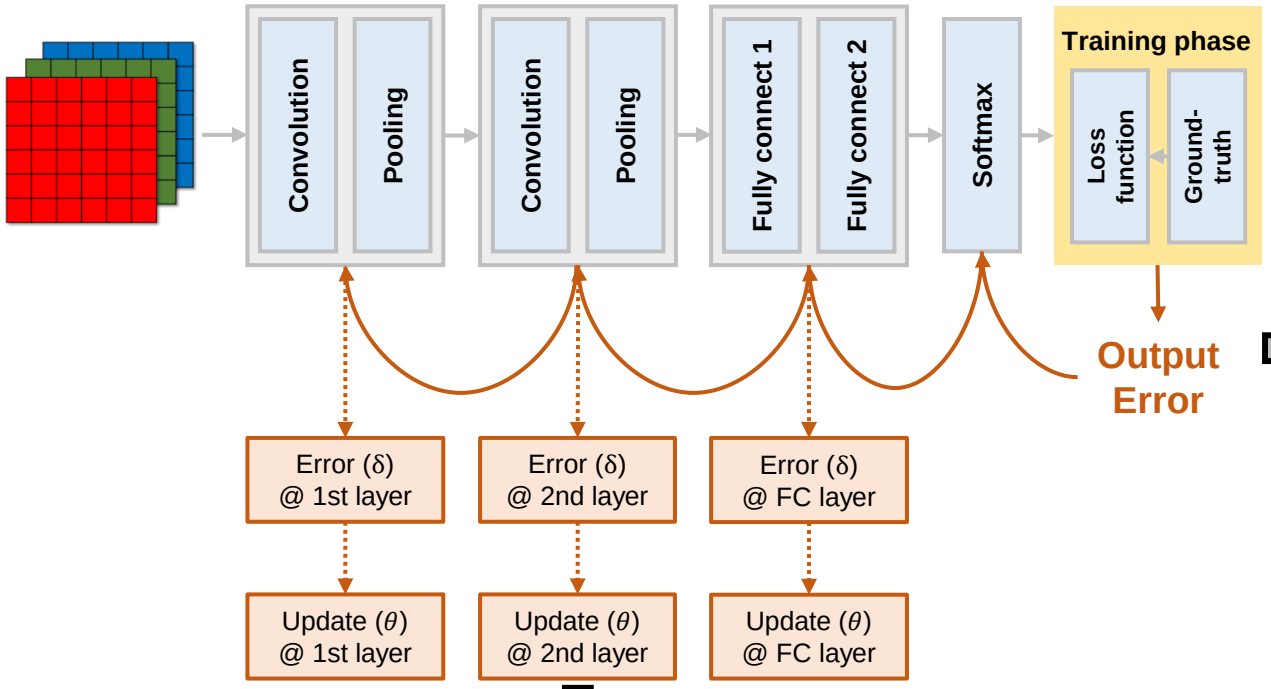
Till loss $\mathcal{L}(\theta)$ is small

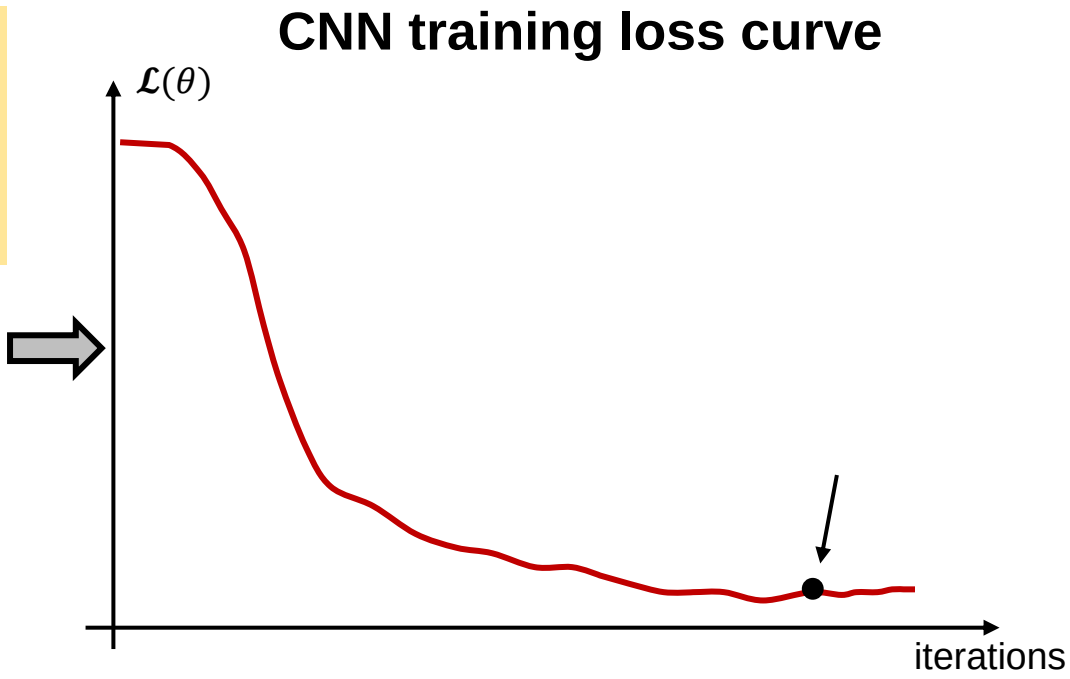
Output CNN model (θ)

Chain rule: $\frac{dL}{d_{input}} = \frac{dL}{d_{output}} \cdot \frac{d_{output}}{df(wx+b)} \cdot \frac{df(wx+b)}{d_{input}}$

Back propagation

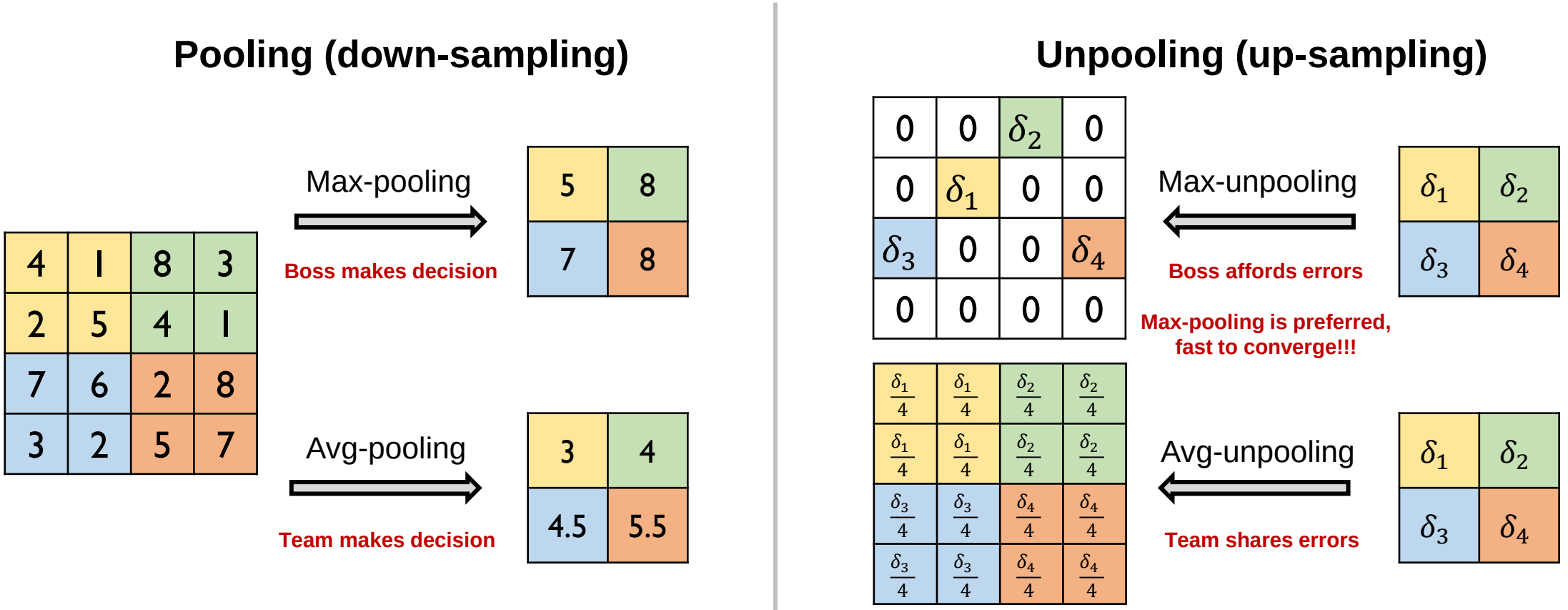
- Propagate error back to layers to update weights per layer.





Back propagation

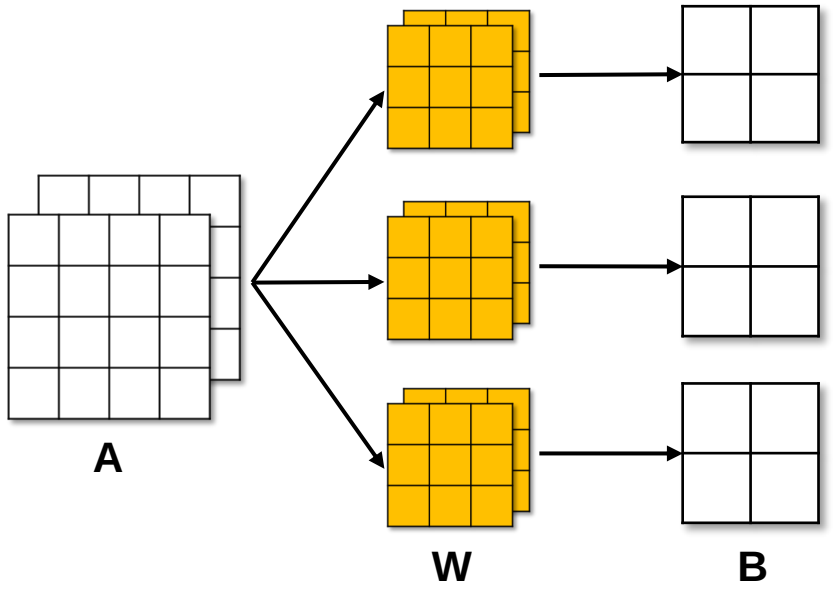
- Back propagation for pooling layer



Back propagation

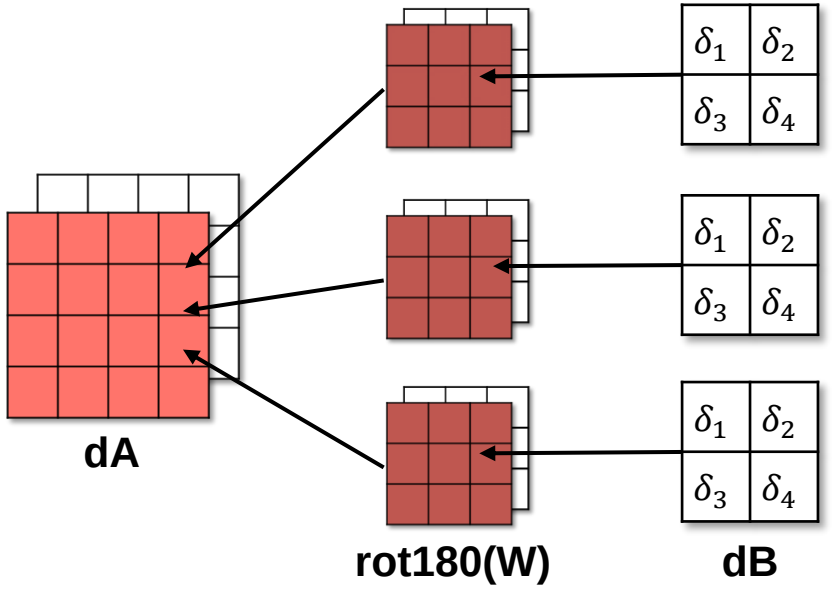
- Back propagation for convolution layer

Convolution



B = conv2(A, W, ‘valid’)

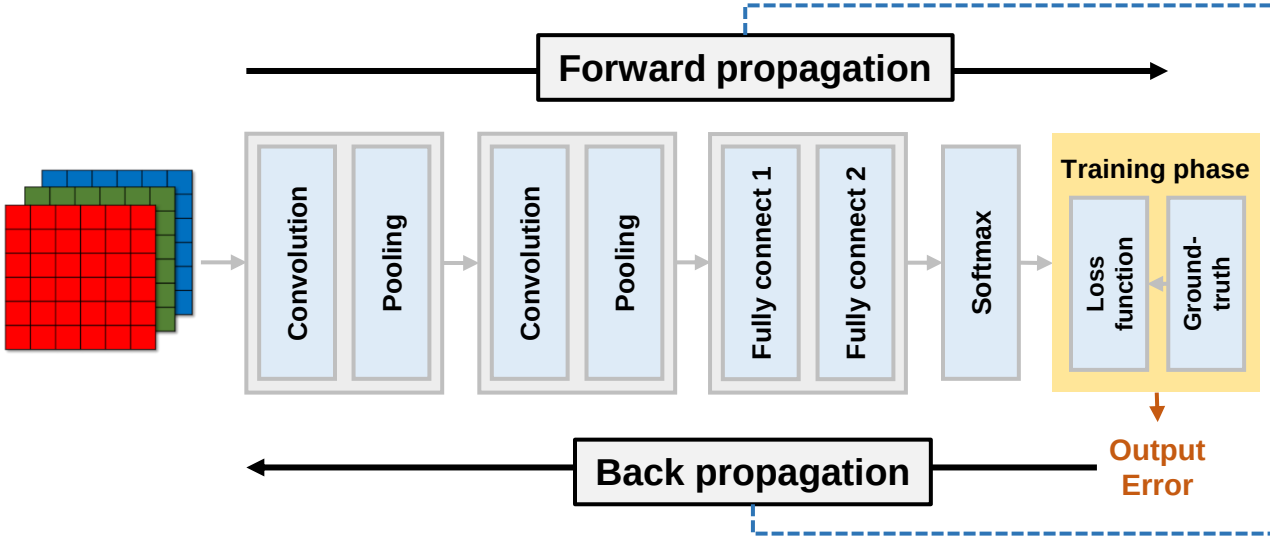
Deconvolution



dA = sum(conv2(dB, rot180(W), ‘full’))

Back propagation

- **6 core equations** you need to know for CNN!



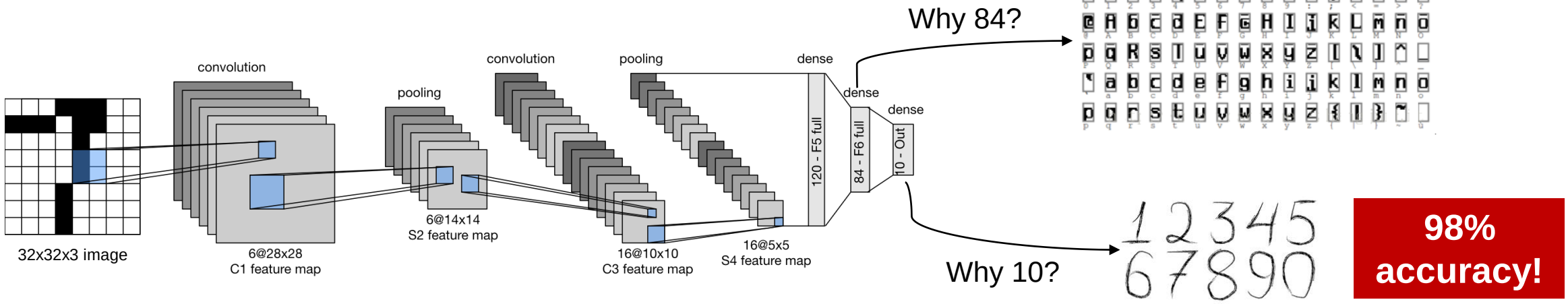
Forward propagation		
Activations (a)	Convolution	$a_l = \boldsymbol{f}(w * a_{l-1} + b)$
	Pooling	$a_l = \boldsymbol{pool}(a_{l-1})$
	Fully connected	$a_l = \boldsymbol{f}(w \cdot a_{l-1} + b)$

Back propagation		
Errors (δ)	Convolution	$\delta_l = \delta_{l+1} * \boldsymbol{rot180}(w_{l+1}) \odot \boldsymbol{f}'(w * a_l + b)$
	Pooling	$\delta_l = \boldsymbol{unpool}(\delta_{l+1})$
	Fully connected	$\delta_l = w_{l+1}^{\top} \cdot \delta_{l+1} \odot \boldsymbol{f}'(w * a_l + b)$

A minimal CNN example

LeNet-5

- LeCun et al. (1998) for recognizing handwritten digits



Layers	**1**	**2**		**3**		**3**			**4**
Type	Normalization	Convolution	Pooling (2x2)	Convolution	Pooling (2x2)	FC 1	FC 2	FC 2	Softmax
Input size	32x32	32x32	28x28x6	14x14x6	10x10x16	5x5x16	1x1x120	1x1x84	1x1x10
Kernel (weights and bias)	-	6@(5x5+1) = **456**	-	16@(5x5x6+1) = **2416**	-	120@(5x5x16+1) = **18120**	84@(1x1x120+1) = **10164**	10@(1x1x84) = **840**	-
Output size	32x32	28x28x6	14x14x6	10x10x16	5x5x16	1x1x120	1x1x84	1x1x10	1x1x10

My example

- Implementation from scratch

28x28x3 image

Demo 1: 2-layers CNN

Convolution (6@5x5) → Pooling (2x2 avg) → Convolution (12@5x5) → Pooling (2x2 max) → Fully connect (192) → Softmax (2 classes) → Mask / No mask

Class 1

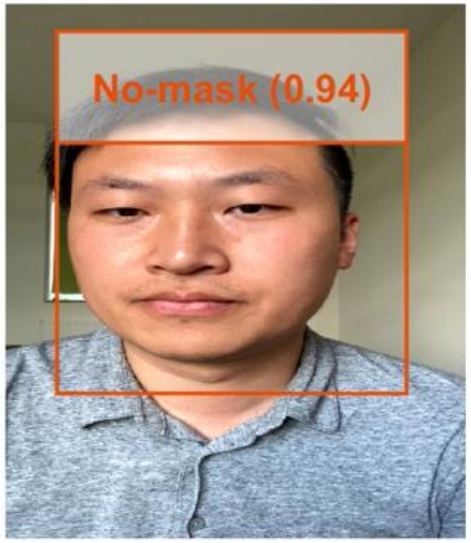
Class 2

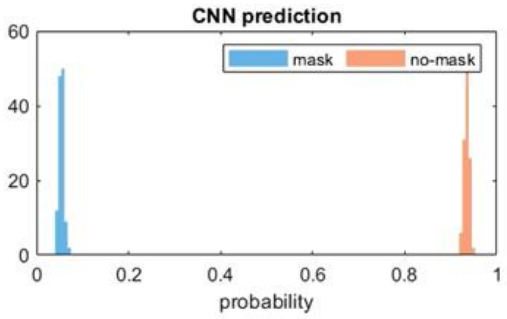
Demo 2: 1-layer CNN (slightly better)

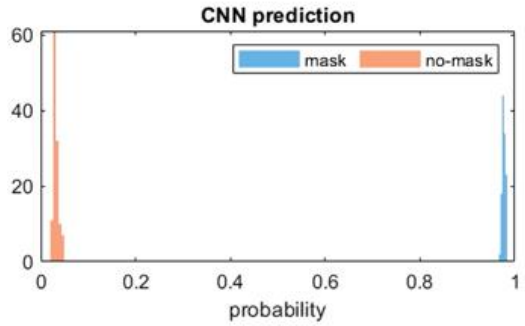
Convolution (6@9x9) → Pooling (4x4 avg) → Fully connect (150) → Softmax (2 classes) → Mask / No mask

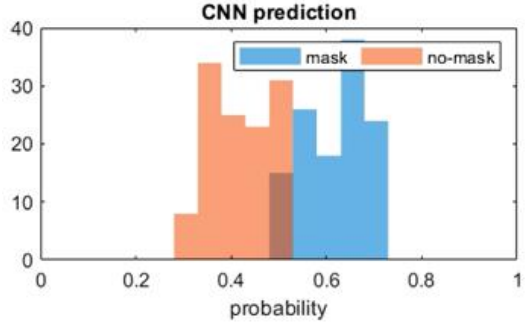
My example (live demo)

- Very simple, everyone can do

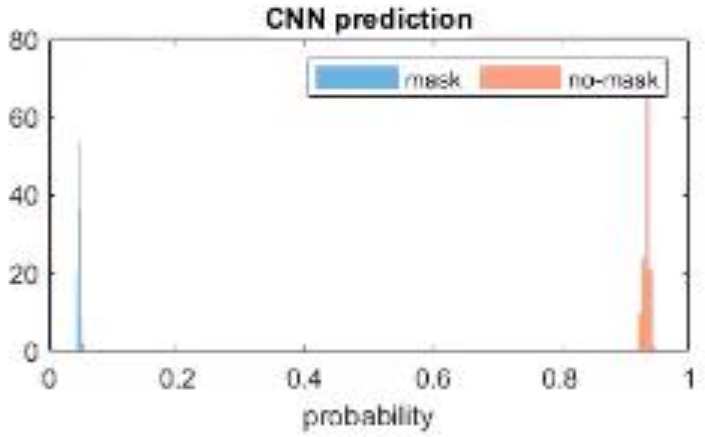









No-mask (0.93)


CNN prediction
mask
no-mask
80
60
40
20
0
0
0.2
0.4
0.6
0.8
1
probability

Why CNN works?

Our world is continuous and smooth at different locations and scales.

CNN pitfalls

Attacks for CNN

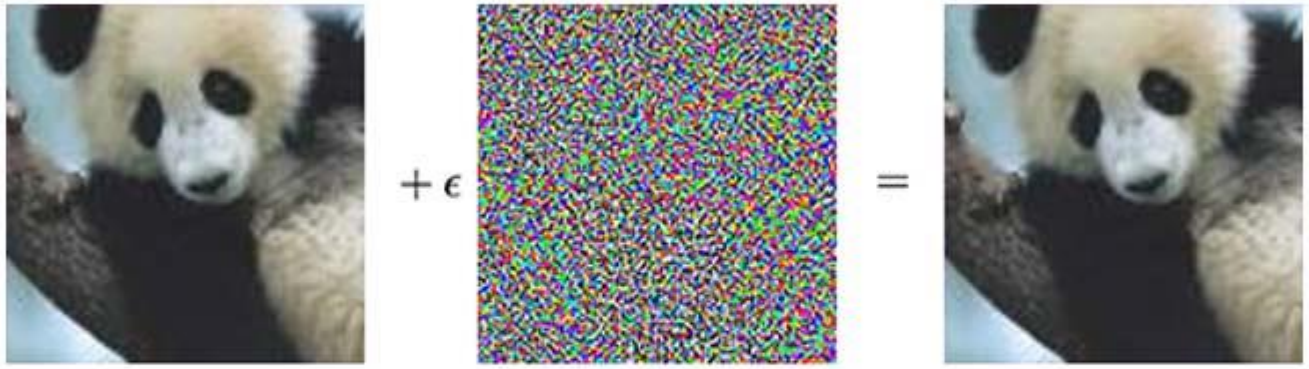
Cup(16.48%)
Soup Bowl(16.74%)

Bassinet(16.59%)
Paper Towel(16.21%)

Teapot(24.99%)
Joystick(37.39%)

Hamster(35.79%)
Nipple(42.36%)

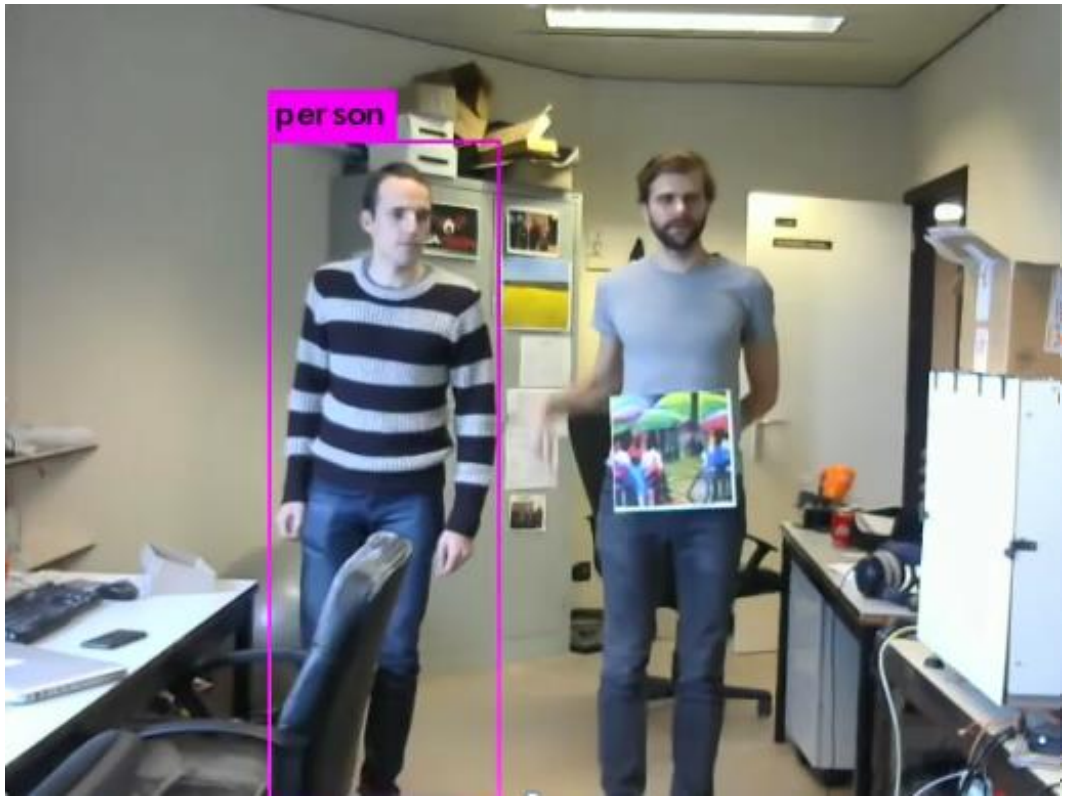
Just add one pixel or “invisible” noise perturbation can fool deep CNN!



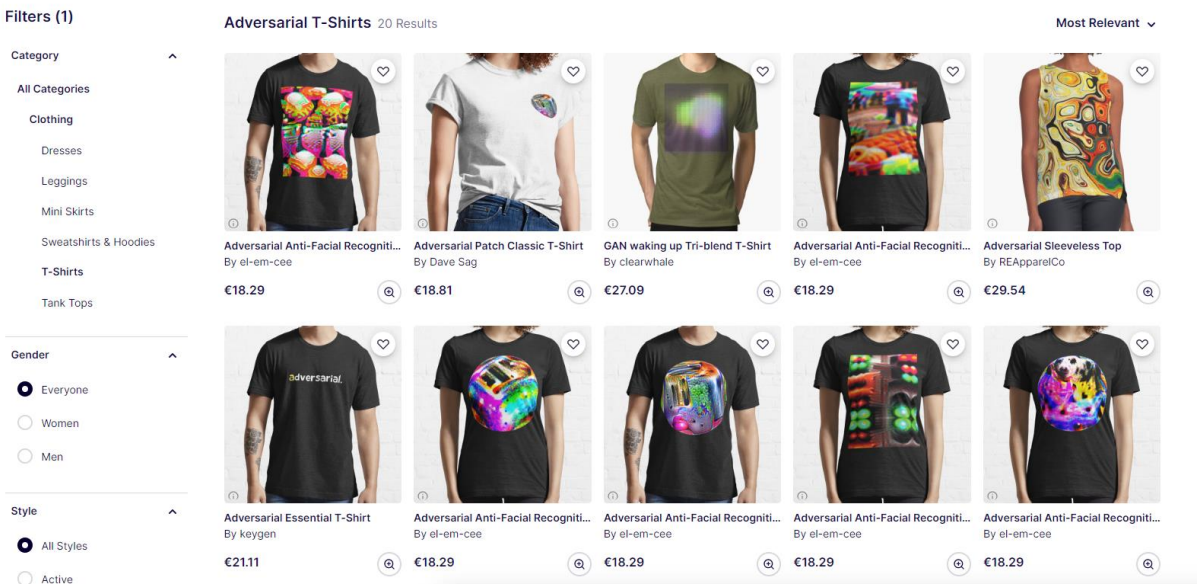
“panda”
57.7% confidence

“gibbon”
99.3% confidence

Attacks for CNN



Adversarial t-shirt, protect privacy?!



CNN remains a black box

- Though we have certain ways/tools to visualize and explain CNN retrospectively, it is not fully transparent and explainable, especially for training process
- No good guidelines to train CNN for a specific task (based on empirical settings or feelings, rule of thumb)
- Hurdle for some assignments, i.e. FDA clearance for clinical device
- …

CNN changes the way of research

Classical research style
(compete on ideas and domain knowledge)

Research becomes boring?!

CNN research style
(compete on data collection and GPU)

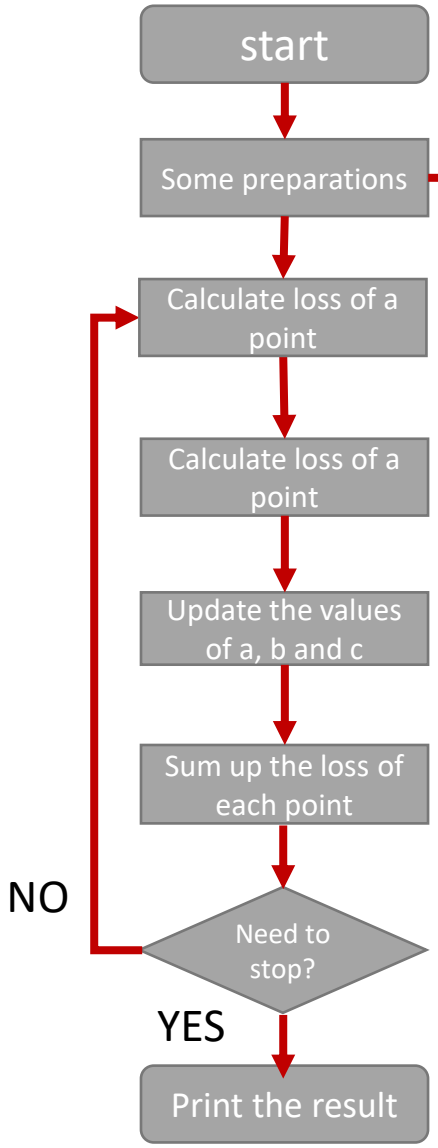
Any ideas about AI?

Assignment

Use perceptron to draw a straight line which can classify salmon and seabass based on length and color: I will provide you with a csv file which contains the length, color and label (-1 or 1) of 200 fishes. Use C to implement a perceptron and print the final equation of the straight line(a x + b y + c = 0)

a) The csv file has been uploaded on bb
b) If the label of a fish is 1, it means this is a salmon; if the label of a fish is -1, it means this is a seabass;
c) If you don't know how to read a csv file, you can refer to the answer of the previous assignment
d) If you want to plot the result in an image, you can install OpenCV.
e) The initial equation of a straight line is : 0 x + 0 y + 0 = 0
f) Use sign() as the activation function
g) The learning rate is 0.1

Assignment



Read the csv file and convert ASCII data into float

Set parameters (learning rate is 0.1; equation of a straight is 0 x + 0 y + 0 = 0; the number of fishes is 200; Maximum number of iterations is 100)

sum_loss = 0

Loss = label - sign(ax + by + c)

a = a + learning_rate * loss * x
b = b + learning_rate * loss * y
c = c + learning_rate * loss * 1

sum_loss = sum_loss + loss

If the sum_loss is equal to 0 or the number of iterations is equal to 100, break

Print ax + by + c = 0

Assignment

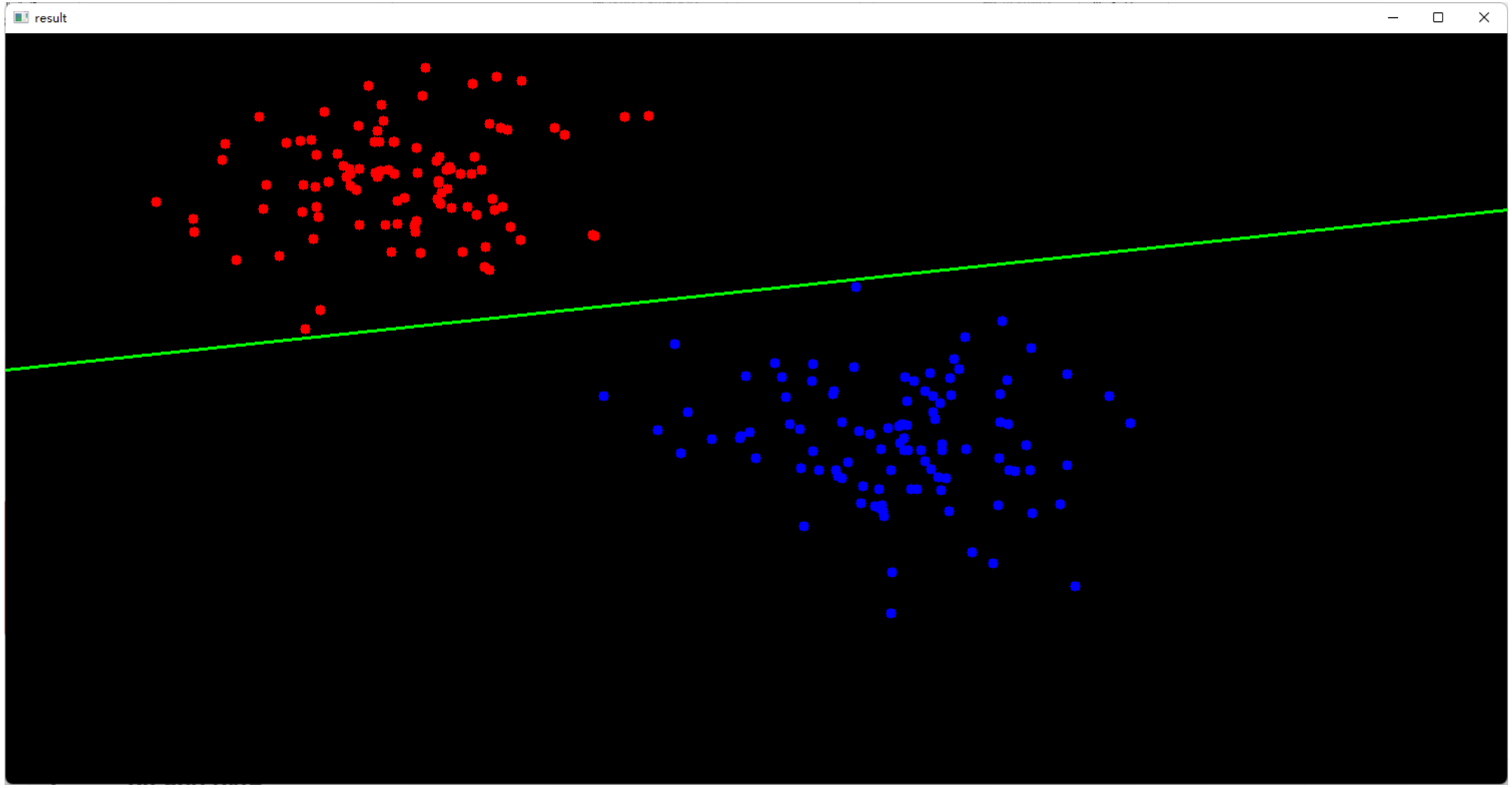
You can choose to use this function to plot the result

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <opencv2/opencv.hpp>

#define num_of_points 200
#define learning_rate 0.1


typedef struct
{
float x[2];
int label;
}Point;

Point points[num_of_points];
```

```
void show_result(Point* points, int num, float a, float b, float c)
{
cv::Mat result = cv::Mat(750, 1500, CV_8UC3, cv::Scalar(0, 0, 0));
for (int i = 0; i < num; i++)
{
cv::Point2f fish_point = cv::Point2f(points[i].x[0] * 50,
points[i].x[1] * 50);
if (points[i].label == 1)
cv::circle(result, fish_point, 5, cv::Scalar(255, 0, 0), -1);
else
cv::circle(result, fish_point, 5, cv::Scalar(0, 0, 255), -1);
}

cv::line(result, cv::Point2f(0,  -1 * c * 50 / b), cv::Point2f(1500,-1
* ( a * 1500 + c * 50) / b), cv::Scalar(0, 255, 0), 2);
std::cout << (float)c / b << "  " << (-1 * a * 1400 + c) / b;
imshow("result", result);

cv::waitKey(0);
}
```

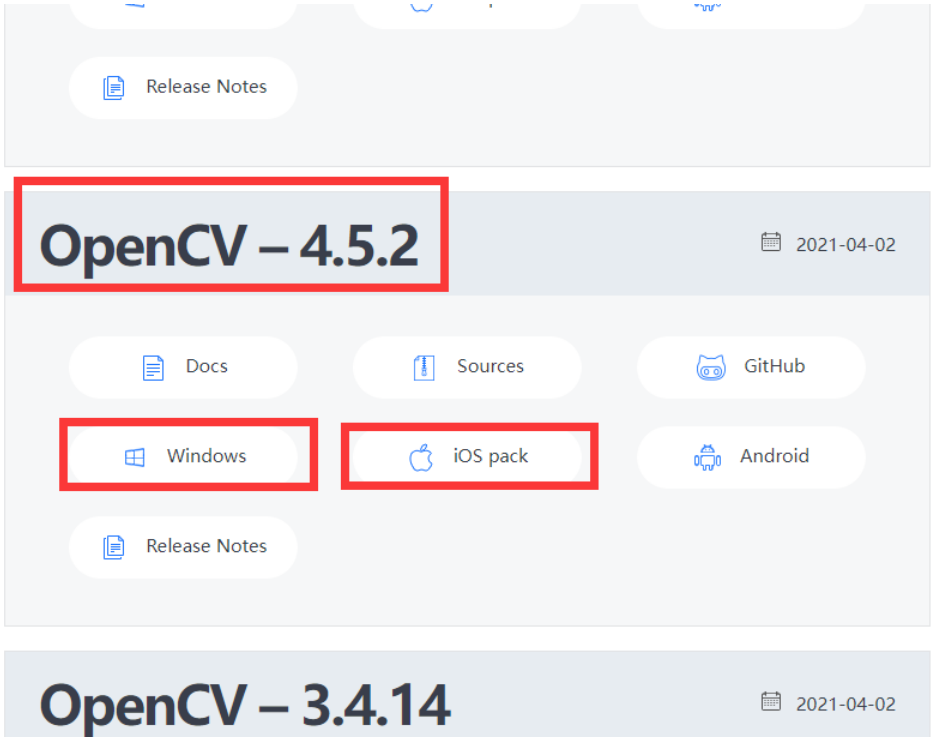
Assignment

The result looks like this



Install OpenCV on VS 2022

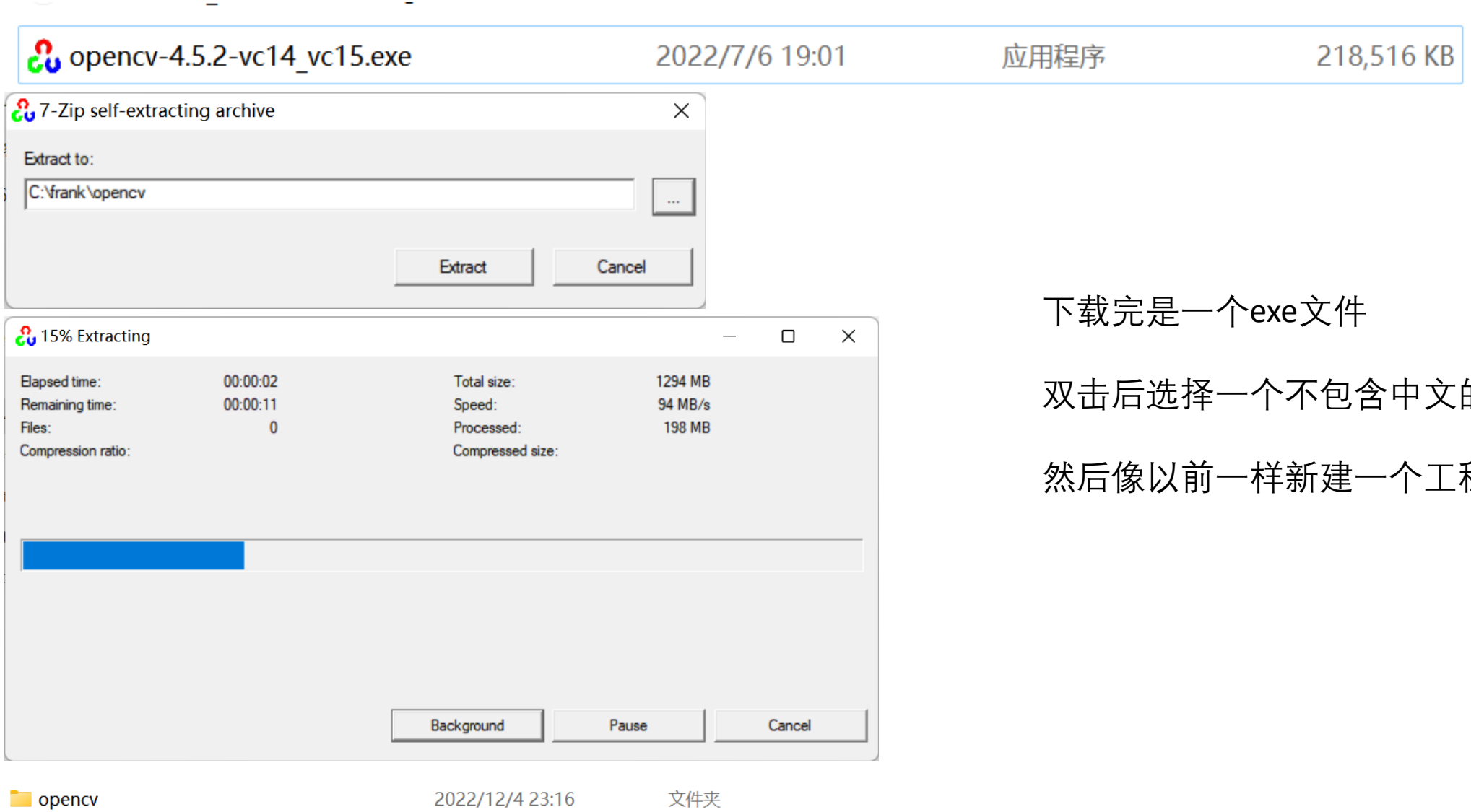
点击这个链接，到opencv官网下载 Releases - OpenCV



推荐4.5.2版本

根据自己的系统选择安装包

Install OpenCV on VS 2022

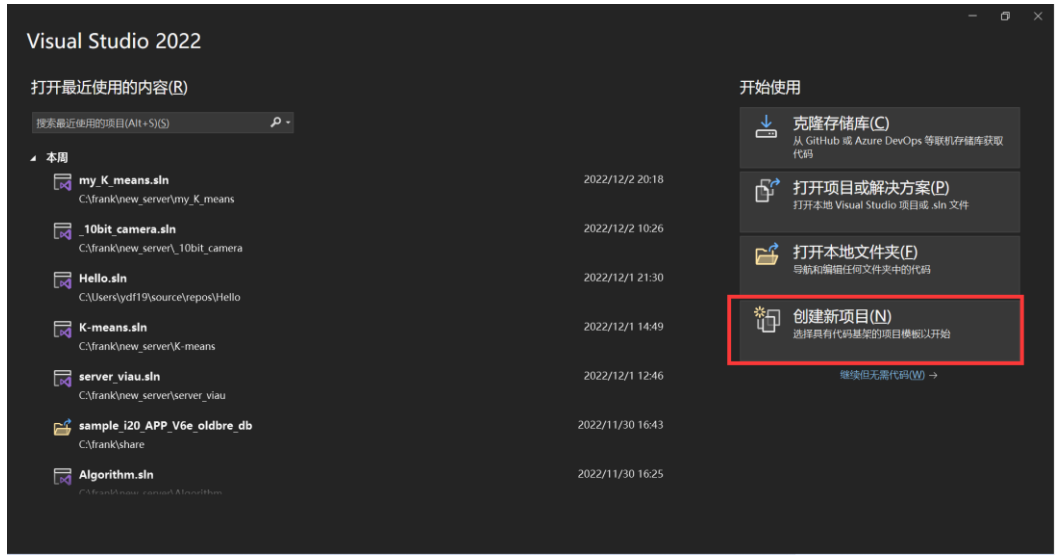


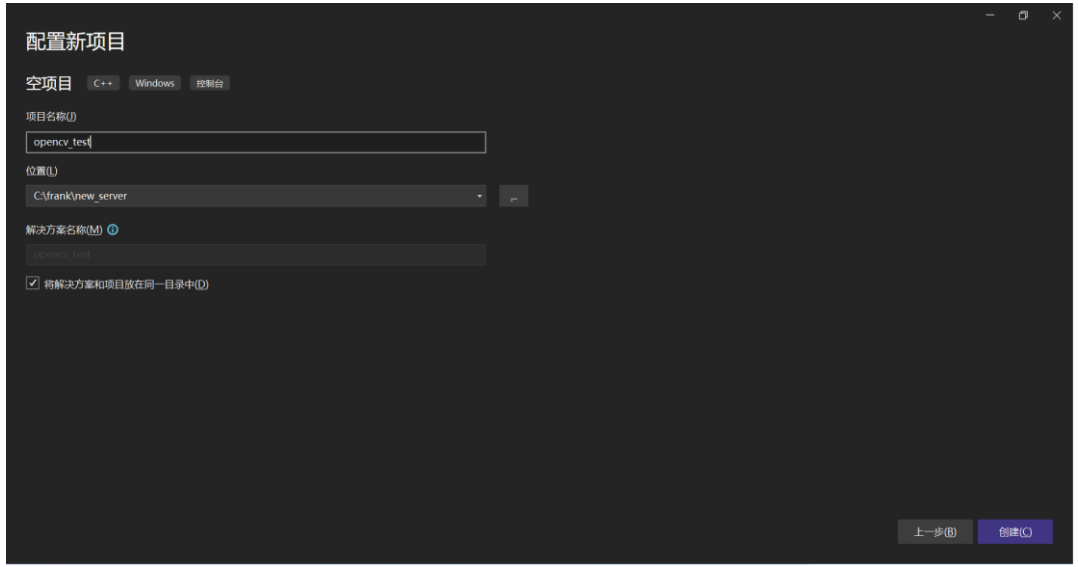
下载完是一个exe文件

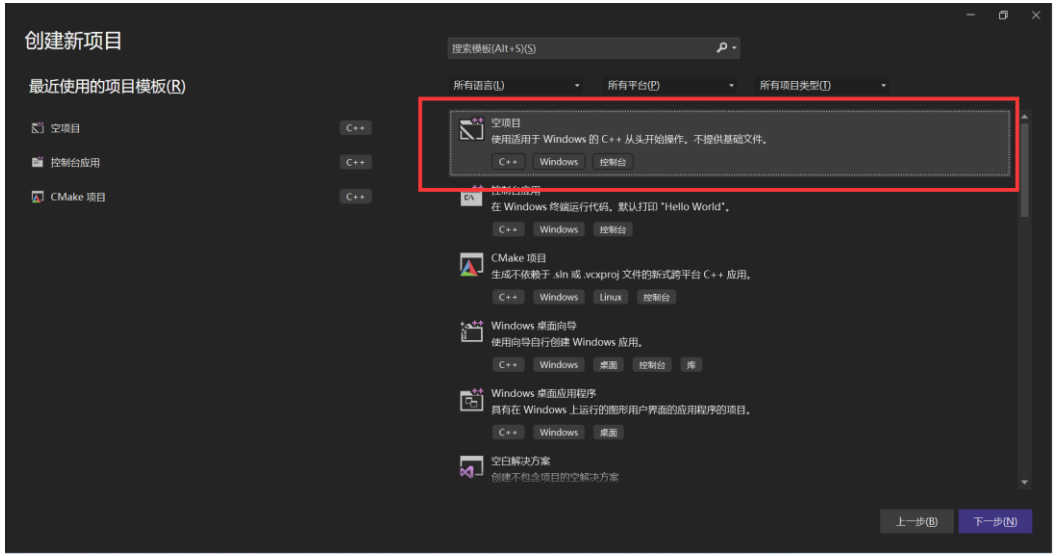
双击后选择一个不包含中文的路径

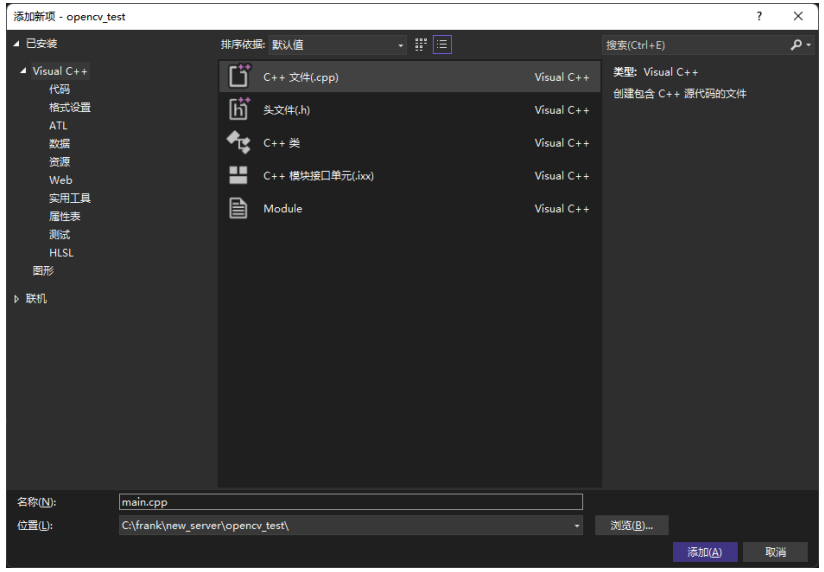
然后像以前一样新建一个工程

Install OpenCV on VS 2022

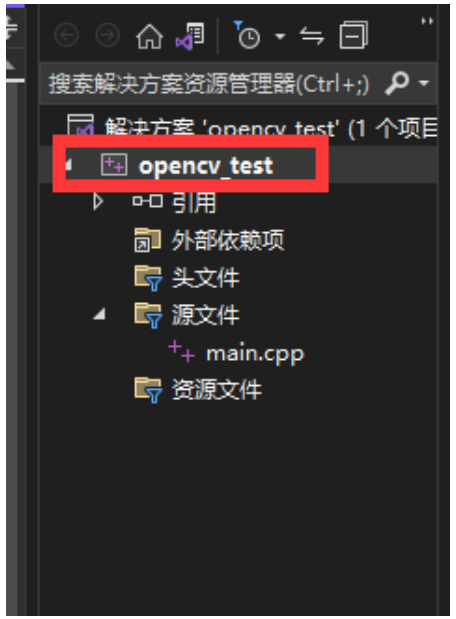






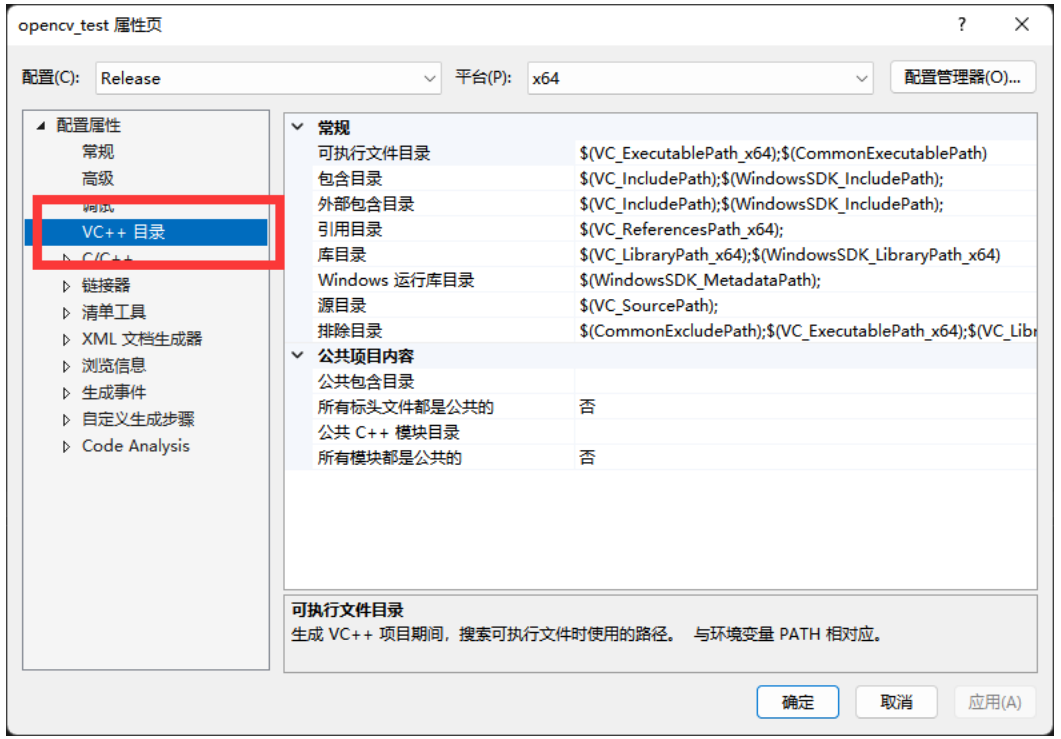


Install OpenCV on VS 2022



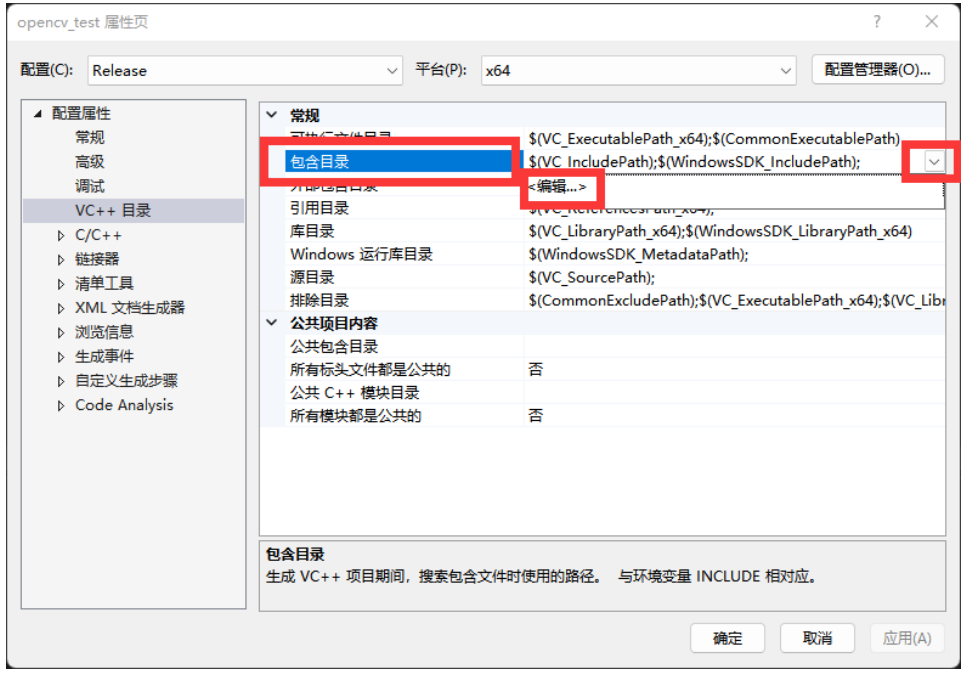
右键新建的项目

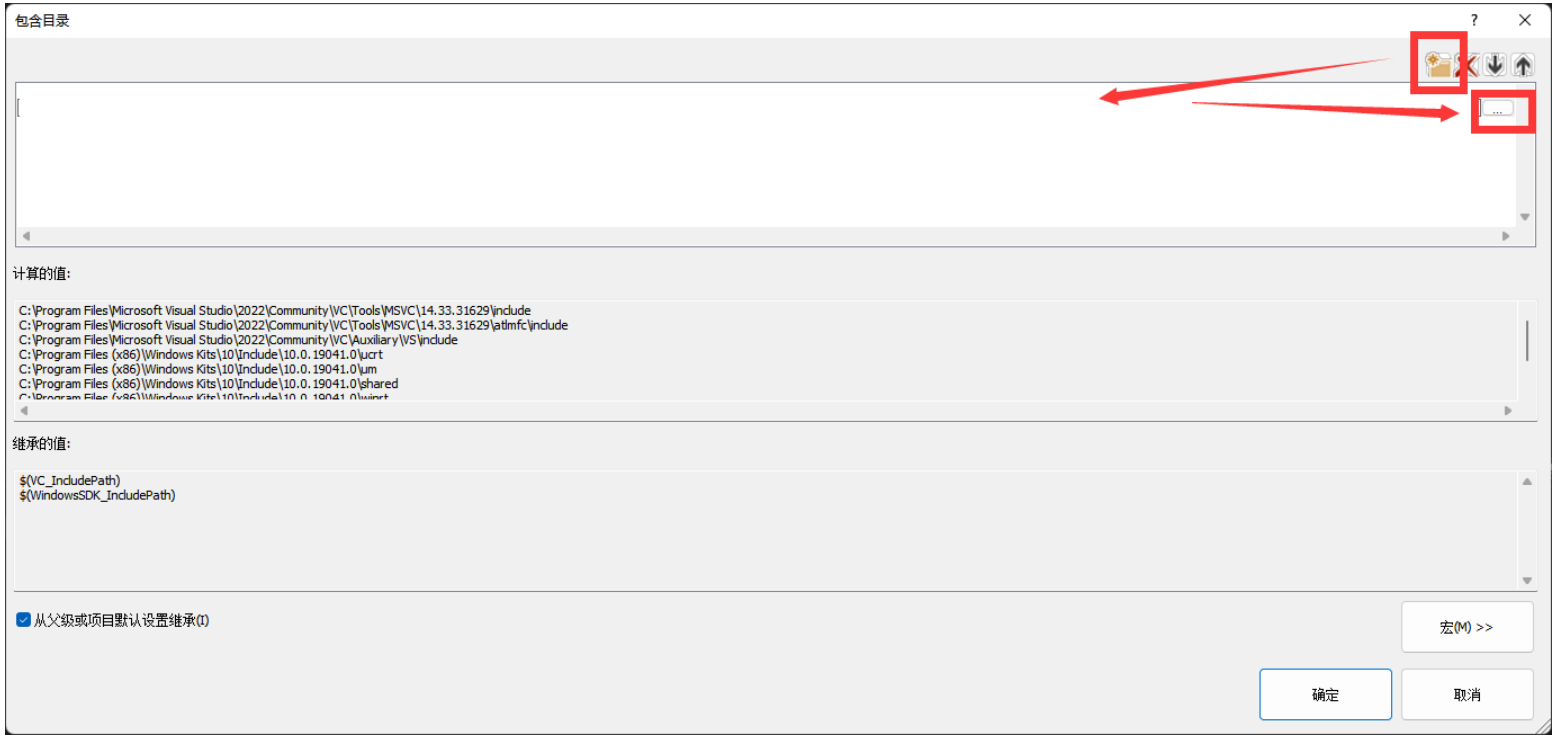
点击属性



点击VC++目录

Install OpenCV on VS 2022

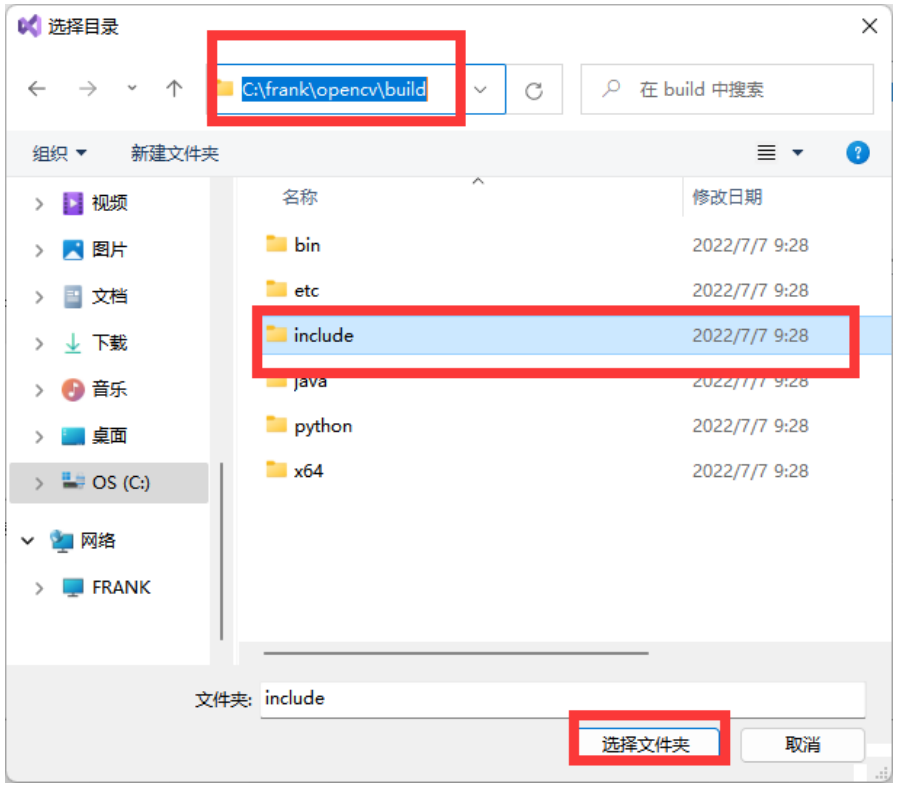


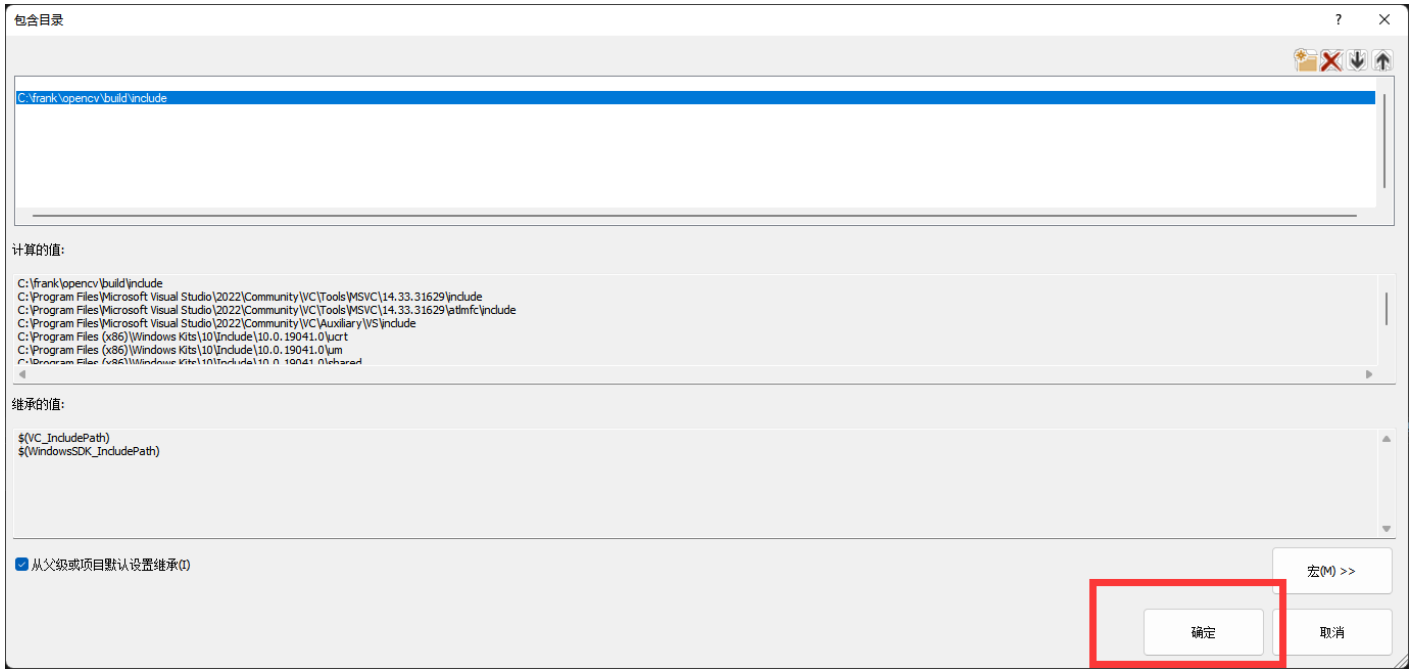


点击包含目录 点击右边的⬇ 点击编辑

依次点击

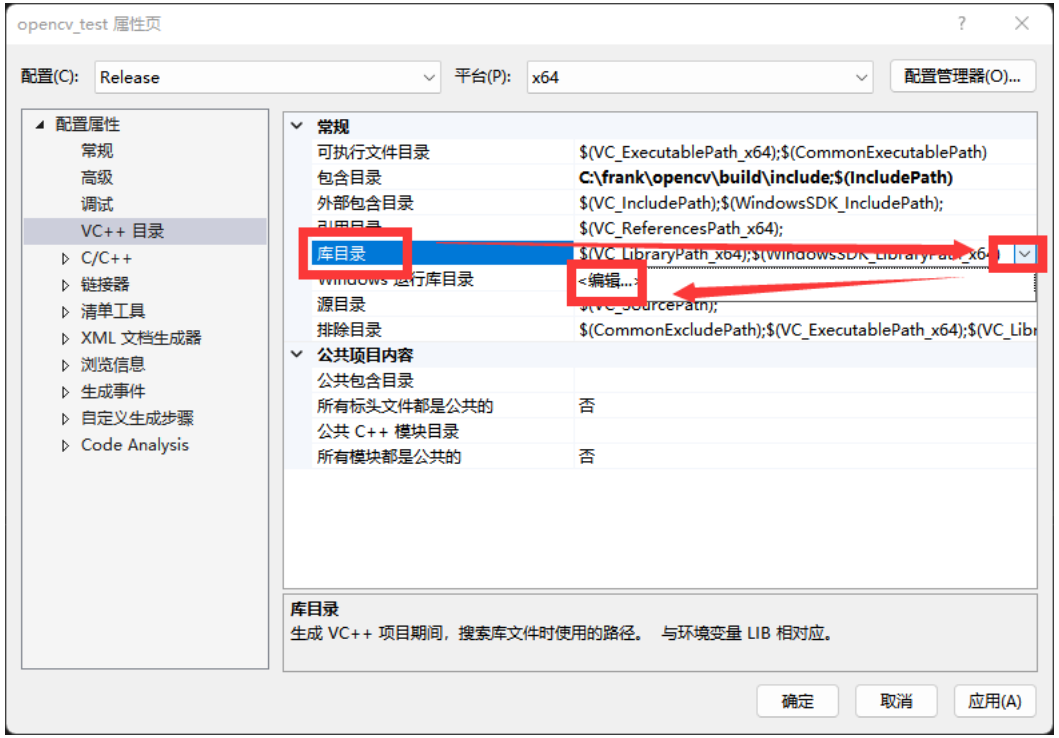
Install OpenCV on VS 2022





选择你安装opencv的路径 opencv/build/include 然后确定

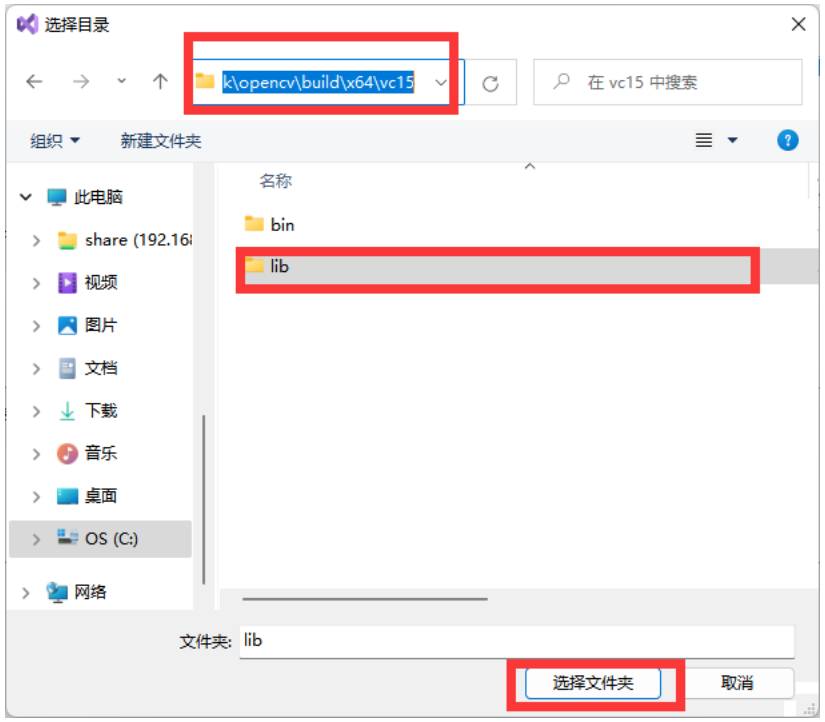
Install OpenCV on VS 2022

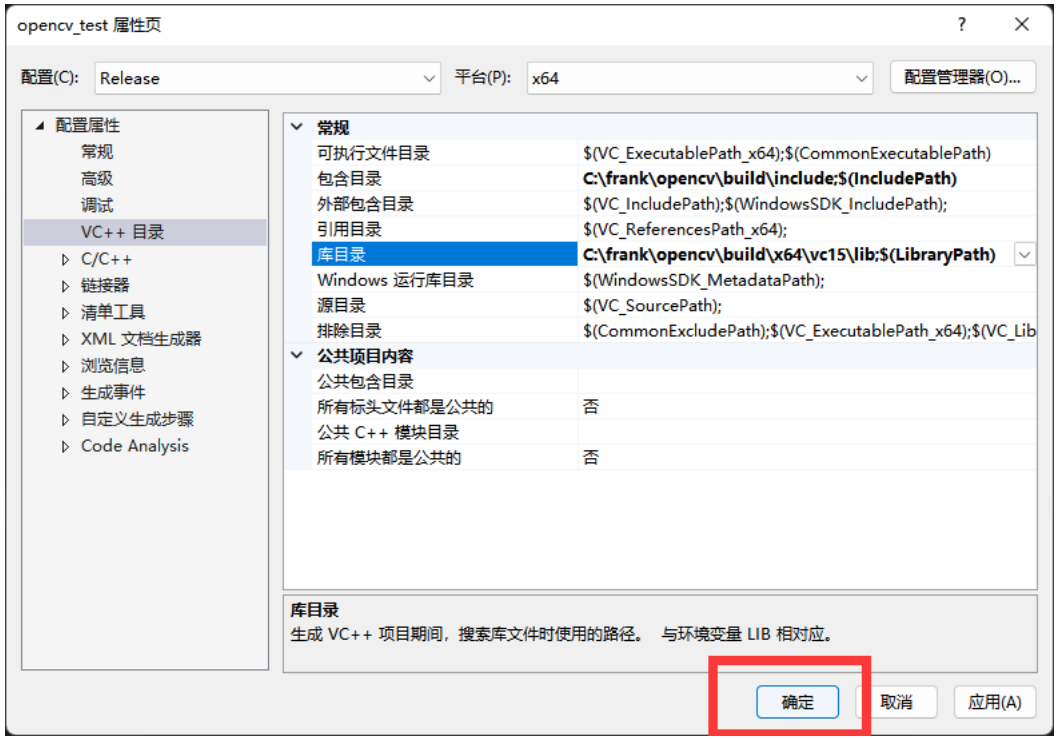


点击库目录 点击右边的⬇ 点击编辑

依次点击

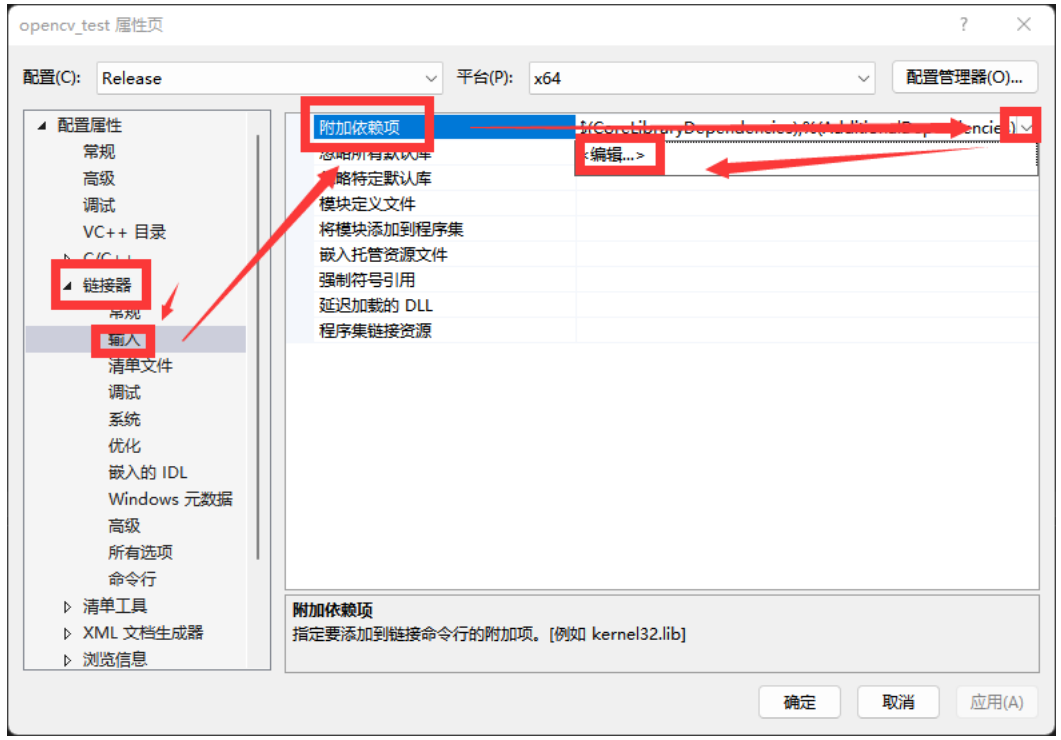
Install OpenCV on VS 2022

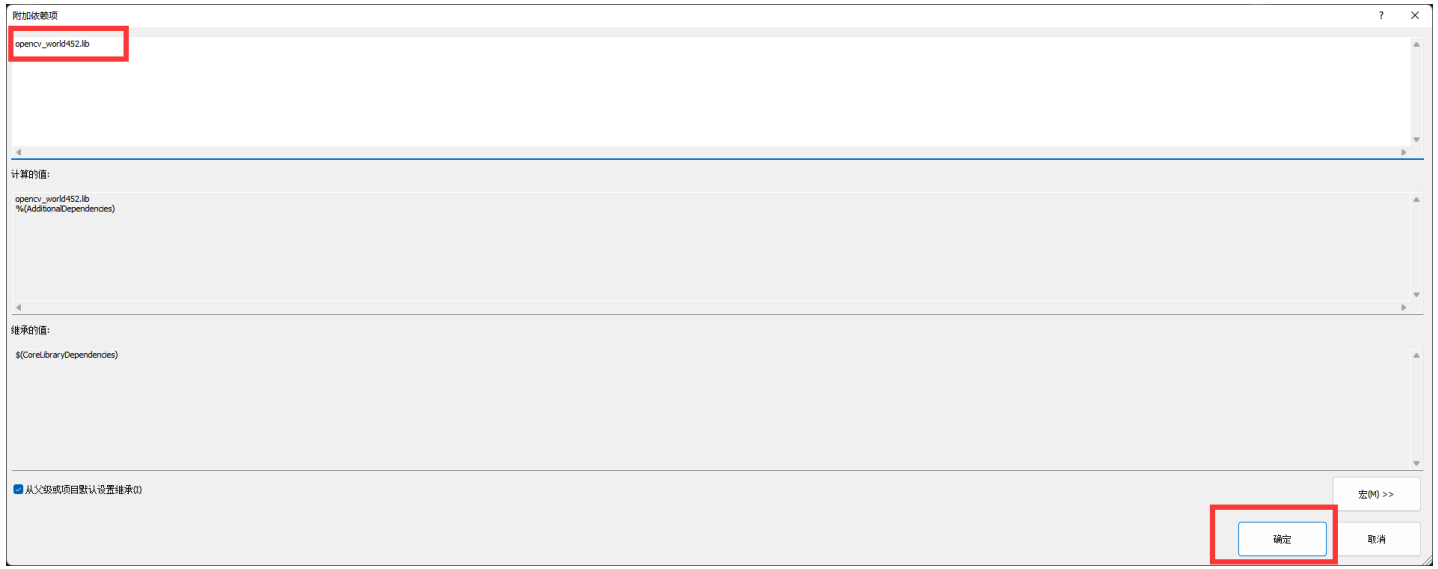




选择你安装opencv的路径 opencv/build/x64/vc15/lib 然后确定

Install OpenCV on VS 2022

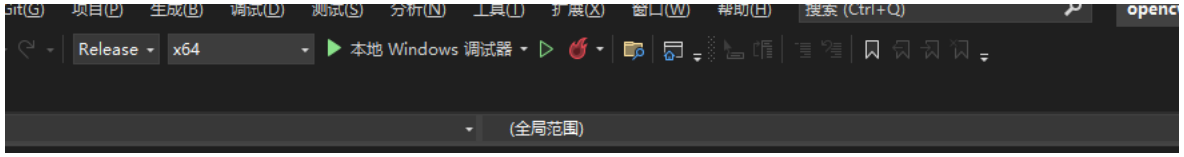




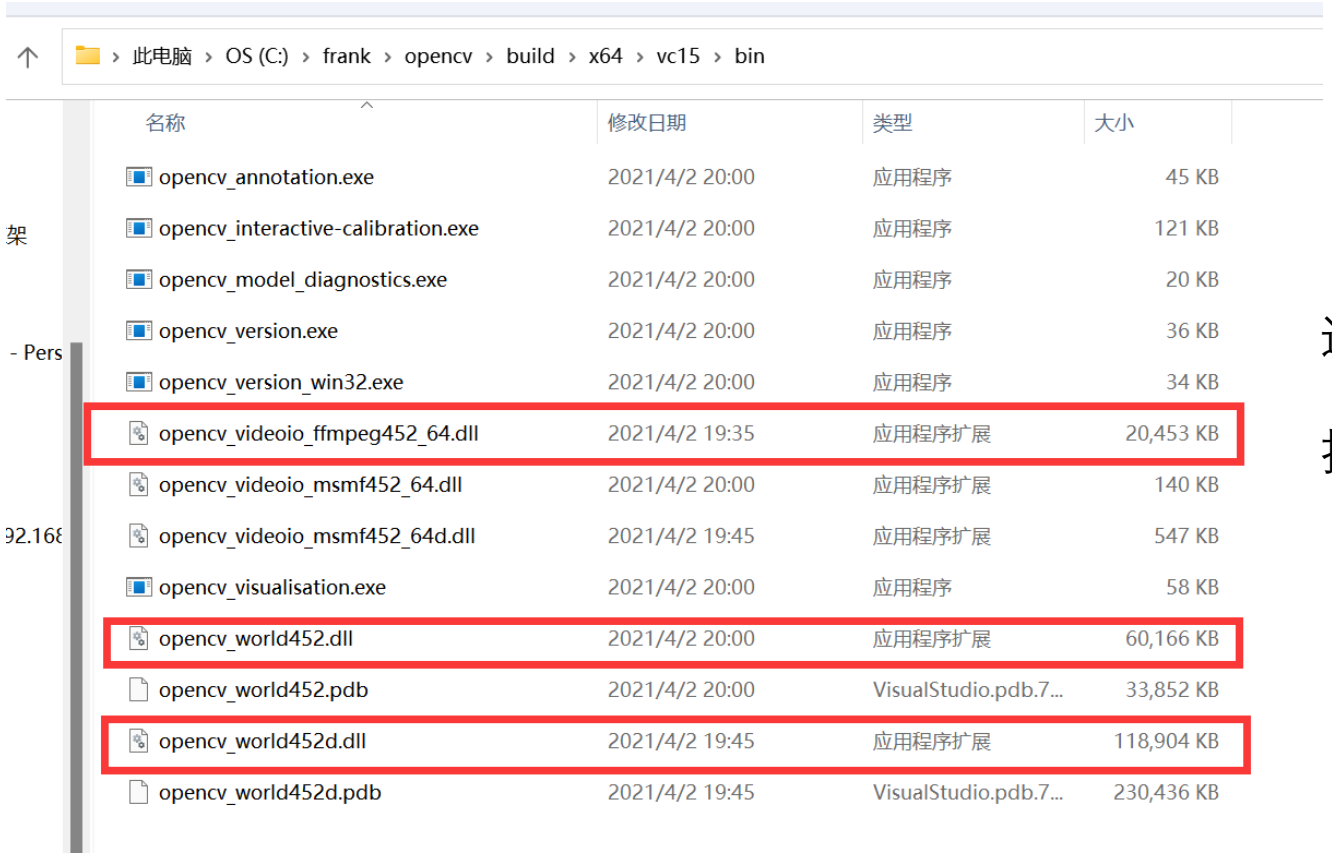
选择 链接器 -> 输入 -> 附加依赖项 -> ⬇ -> 编辑

手动输入 opencv_world452.lib 然后确定

Install OpenCV on VS 2022



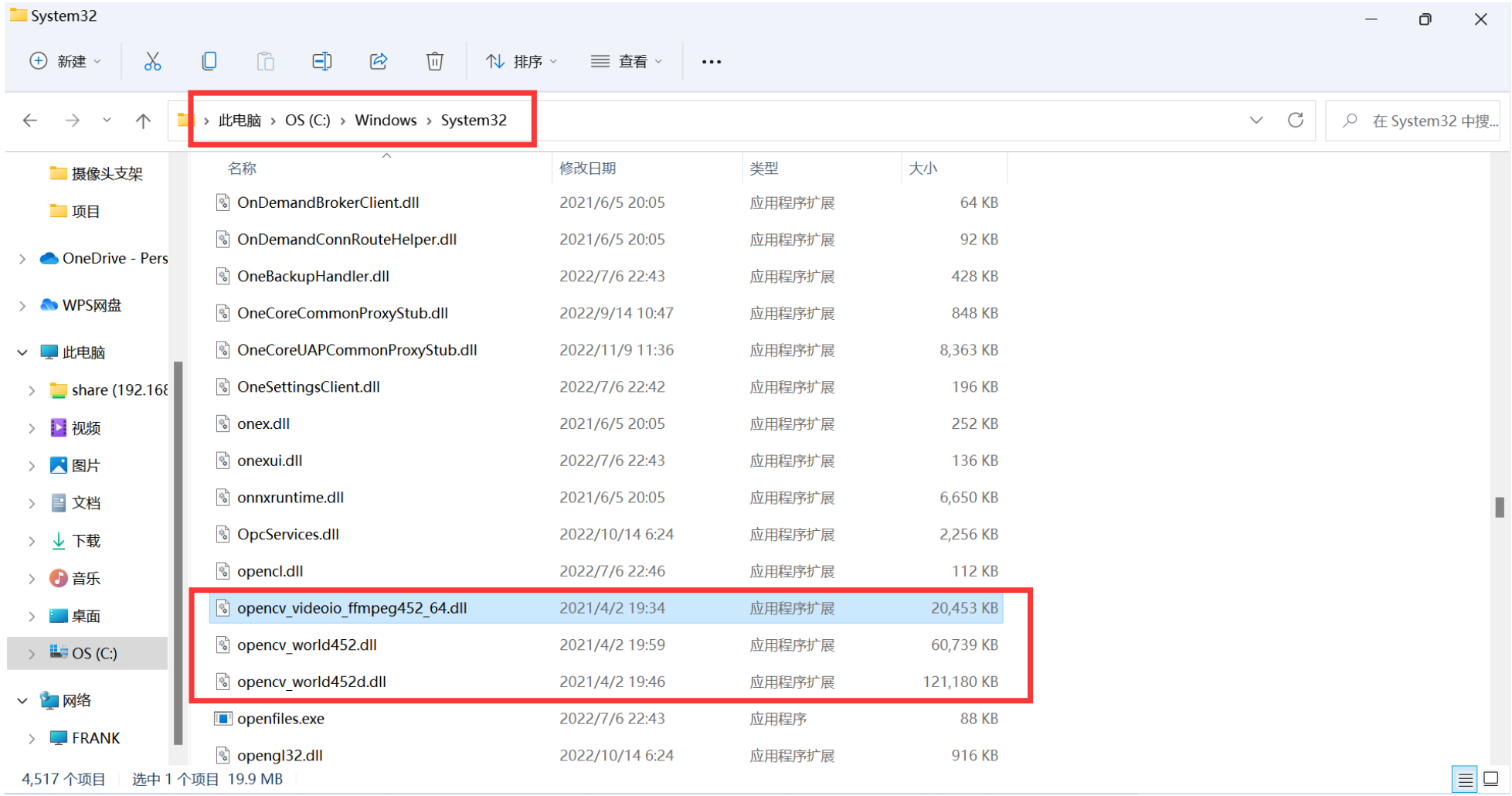
这里改成release x64



进入到opencv/build/x64/vc15/bin

把这三个 .dll 文件复制到C：//Windows/system32下

Install OpenCV on VS 2022



就像这样

Install OpenCV on VS 2022

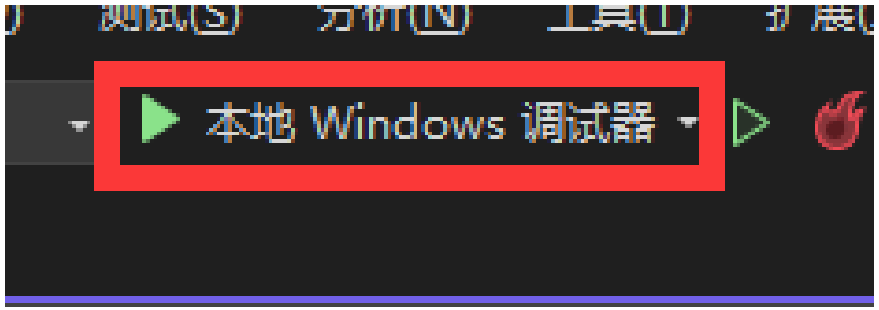
```cpp
#include <iostream>
#include <opencv2/opencv.hpp>
using namespace std;
using namespace cv;
int main()
{
    // 读取图片（使用图片的绝对路径，参考自己的图所在目录）
    Mat srcImg = imread("C://Users//ydf19//Desktop//lenna.jpg");
    if (srcImg.empty()) {
        cout << "could not load image..." << endl;
        return -1;
    }

    imshow("Test opencv setup", srcImg);
    // 显示灰度图
    Mat Gray;
    cvtColor(srcImg, Gray, 6);
    imshow("Gray", Gray);
    // 等待任意按键按下，不添加此语句图片会一闪而过
    waitKey(0);

    cout << "Hello,world." << endl;
    return 0;
}
```

写几行代码测试一下

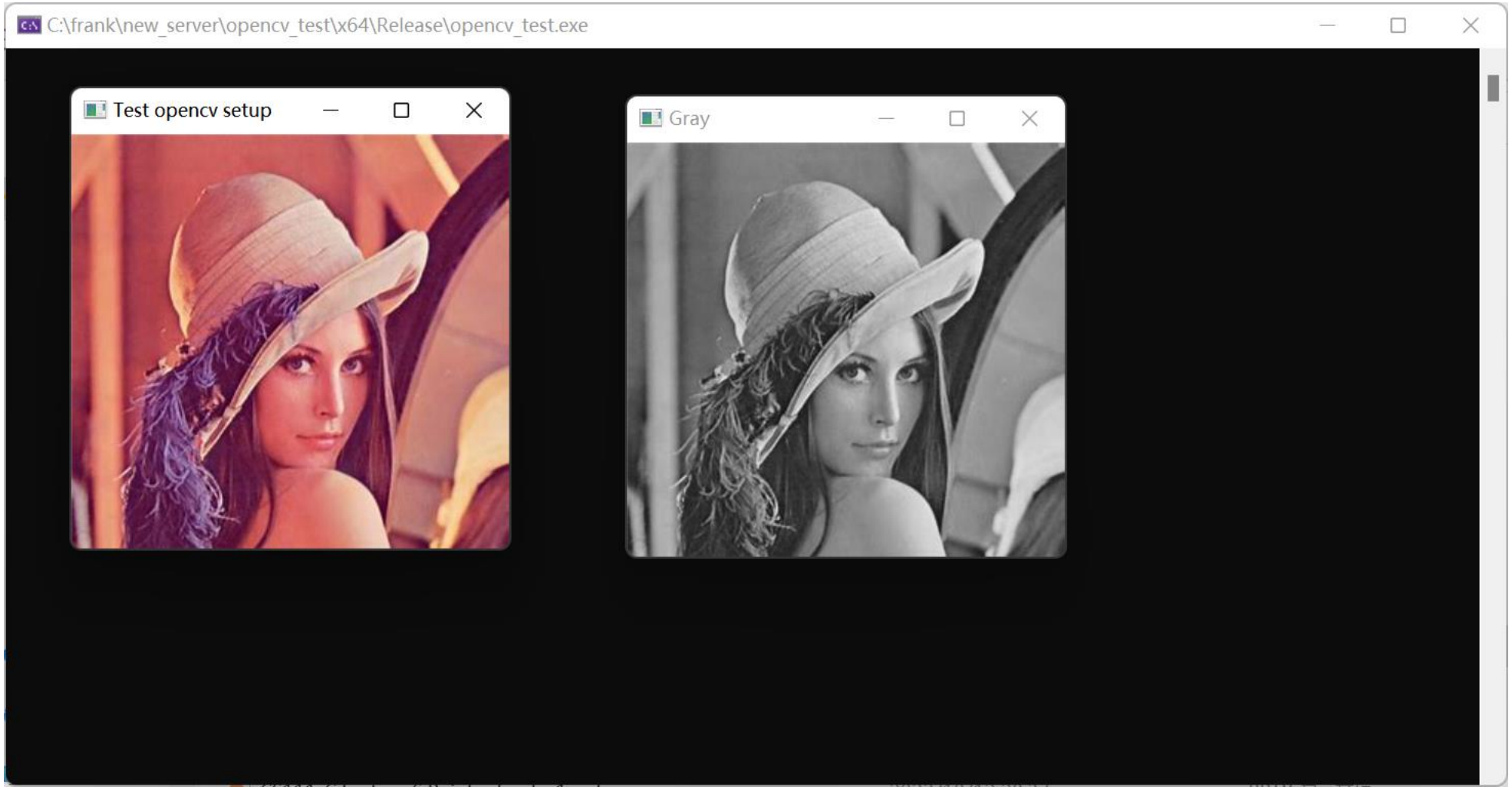
在桌面放一张图片

然后粘贴这些代码

运行



Install OpenCV on VS 2022



成功了

其他系统和编译器的安装可以参考以下链接

Vscode & macOS	https://blog.csdn.net/weixin_43562948/article/details/103956901
Vscode & windows	https://blog.csdn.net/Avrilzyx/article/details/107036375
DEV c++ & windows	https://blog.csdn.net/wadefelix/article/details/1334515