

Lecture 5

Object Classification & Detection from a View of Deep Learning

李亦宁

2022年3月



目录

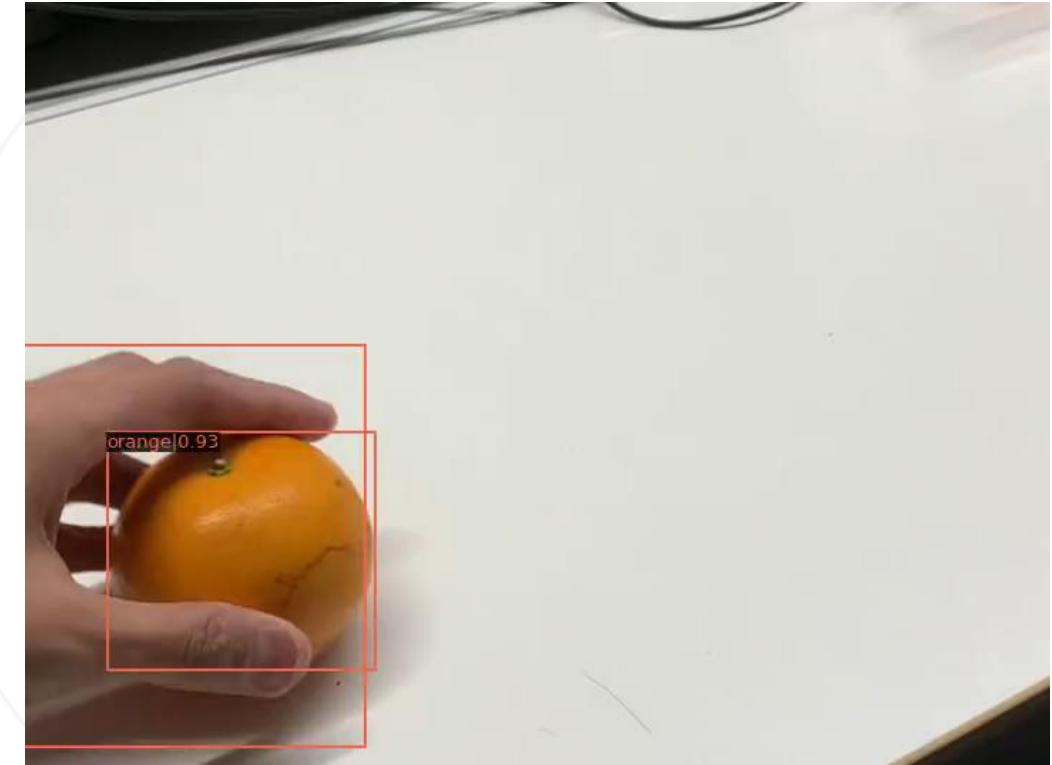
图像分类：

图像中通常只有一个（主要）物体
只需要进行类别预测



目标检测：

图像中有不定数目的物体
分类同时还需要定位物体

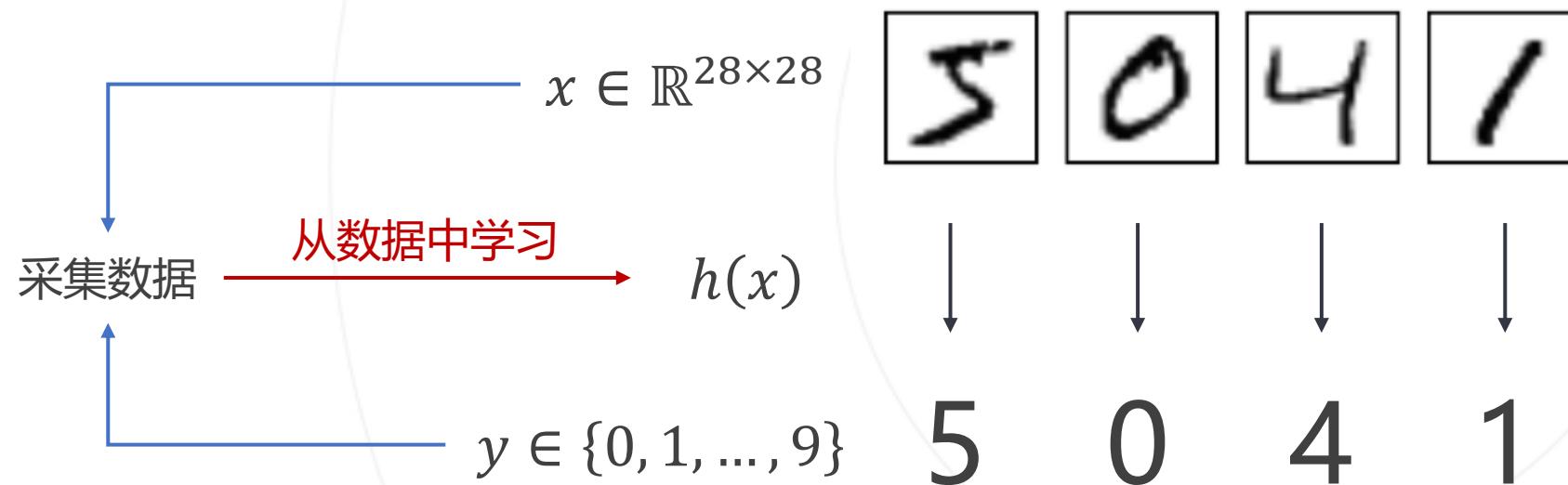


机器学习视角下的图像分类

- 从数据集 $\mathcal{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ 中学习一个分类函数

$$y = h(x) \text{ 或 } P(y|x) = h(x)$$

- 其中, $x \in \mathbb{R}^N$ 为输入模型的数据, $y \in \mathcal{Y}$ 为模型预测的类别, $P(y|x)$ 为模型预测数据属于不同类别的概率。 $h(x)$ 也称为**分类器**。



1. 构建函数族 \mathcal{H} 包含全部可能的分类器函数 $h(x)$

- 例：线性分类器 $\mathcal{H} = \{\sigma(\theta^T x + \theta_0) \mid \theta \in \mathbb{R}^N, \theta_0 \in \mathbb{R}\}$
- 例：多层感知器 $\mathcal{H} = \{\sigma(\theta_2^T \sigma(\theta_1^T x + \theta_{10}) + \theta_{20}) \mid \theta_1 \in \mathbb{R}^{N \times M}, \theta_2 \in \mathbb{R}^M, \theta_{10}, \theta_{20} \in \mathbb{R}\}$
- 例：参数化函数族 $\mathcal{H} = \{h(x; \theta) \mid \theta \in \Theta\}$, 对于每一个 θ , $h(x; \theta)$ 是一个分类器

2. 在 \mathcal{H} 中搜索最好的分类器

- 怎么描述 “好” -> 错误率越低越好 -> 损失函数=错误率
- 怎么找到这个最好的 -> 调整 θ 降低损失函数 -> 最优化问题

1. 函数族 $\mathcal{H} = \{h(x; \theta) \mid \theta \in \Theta\}$ 应该是什么样的?

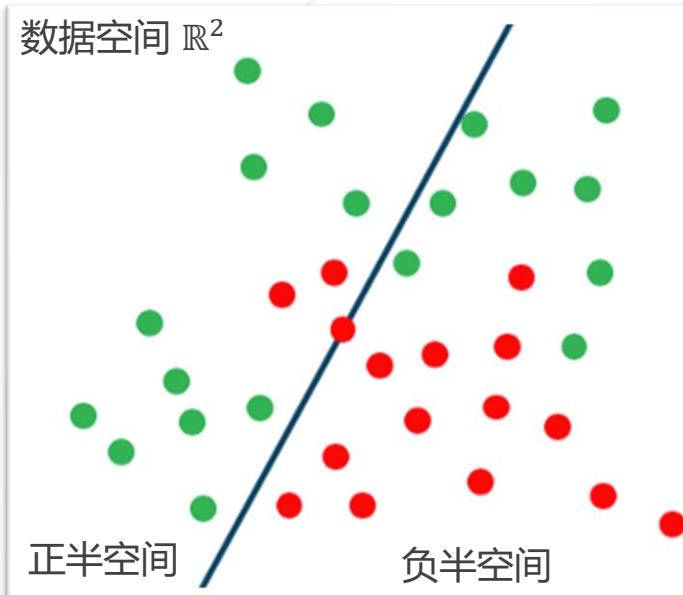
- (确定结构的) 卷积神经网络

2. 怎么衡量分类模型的好坏?

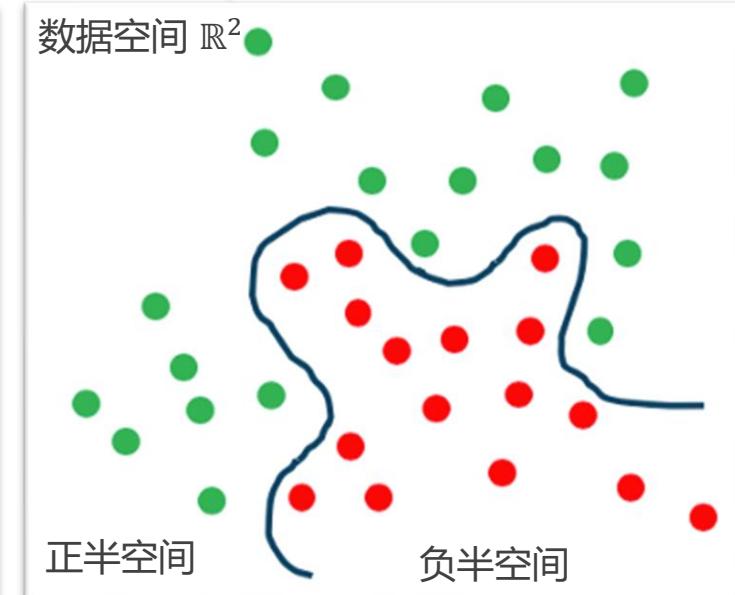
- 交叉熵损失函数

3. 怎么在 \mathcal{H} 中搜索最佳模型?

- 随机梯度下降为主
- 各种经验策略辅助



线性分类函数



非线性分类函数

目标：寻找损失函数曲面 $L(\theta)$ 的谷点

Step 1：随机选取找一个起始点 $\theta^{(0)}$

Step 2：寻找下降最快的方向，即负梯度

方向 $\nabla_{\theta} L(\theta^{(0)})$ ，并前进一步

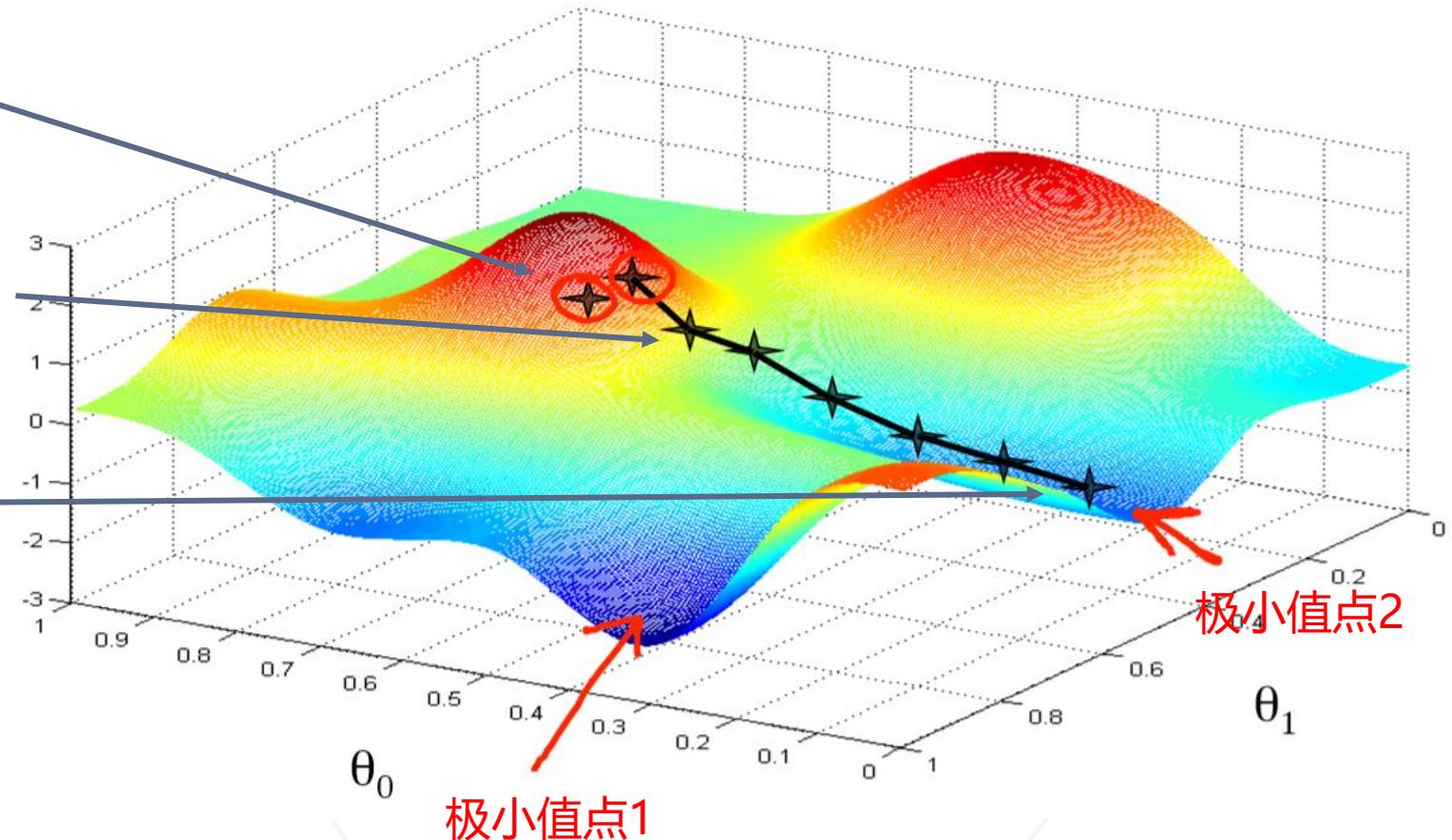
$$\theta^{(1)} = \theta^{(0)} - \eta \nabla_{\theta} L(\theta^{(0)})$$

学习率

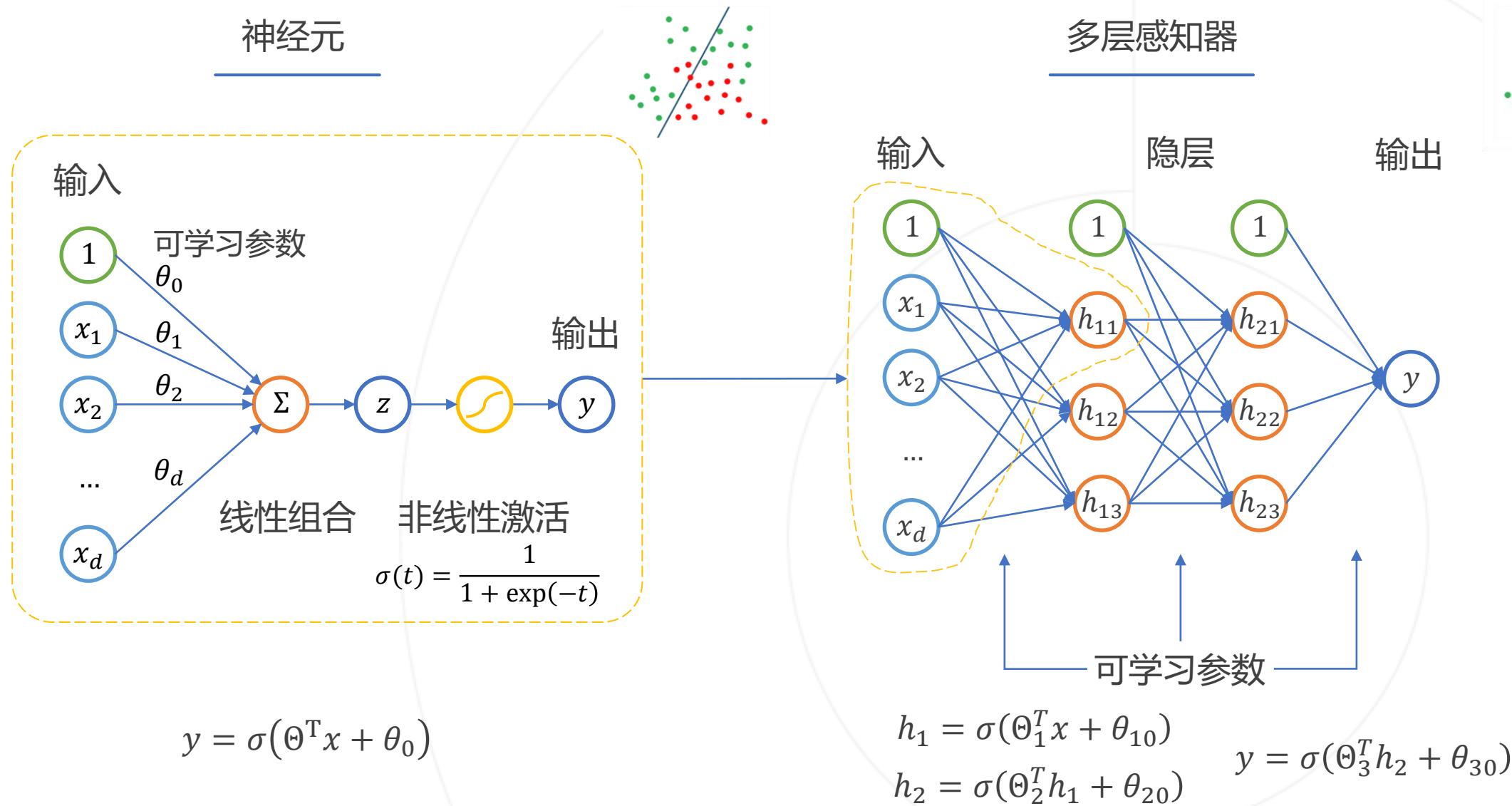
Step 3：重复直到不能再下降

凸函数：保证找到最小值点

非凸函数：不能保证，和起始位置、每步的步长有关系

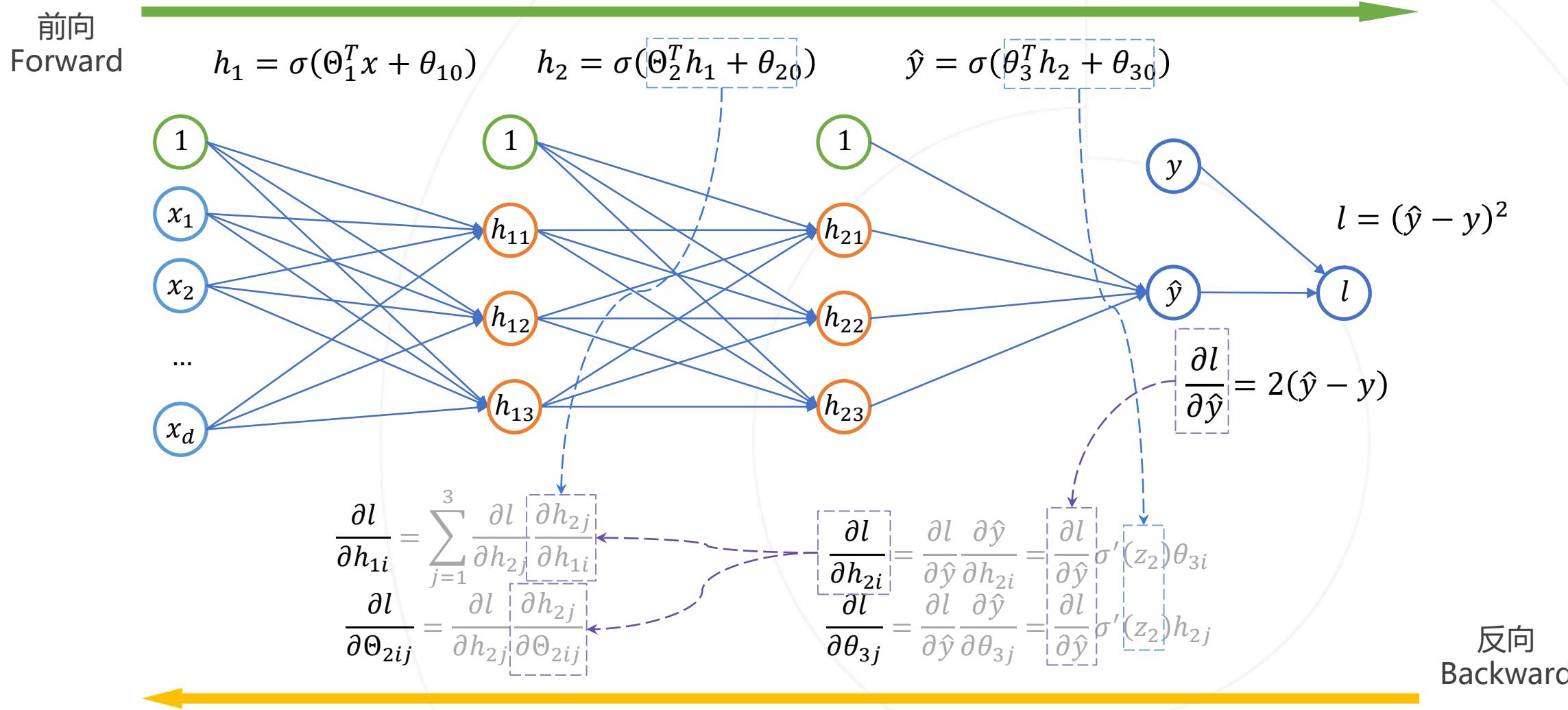


多层感知器



1. 确定结构（层数、每层单元数）的多层感知器构成了一个分类器族 \mathcal{H} ，不同的连接权重对应了族内不同的分类器
 2. 基于梯度下降寻找最优参数，进而得到最优（准确率最高）的网络
- 如何计算损失函数对于网络参数的梯度？ \rightarrow 反向传播算法

➤ 目标：计算损失函数 l 对于所有隐层参数 Θ 的梯度



- 在神经元数目足够的情况下，单隐层的神经网络可以逼近任意连续函数 (1990s~2000s)
- 所需的神经元的数目随数据维度指数级增长
- 图像分类 \neq 拟合**任意复杂的分类函数**
- 从实用角度讲，需要根据图像的特点设计更高效的模型

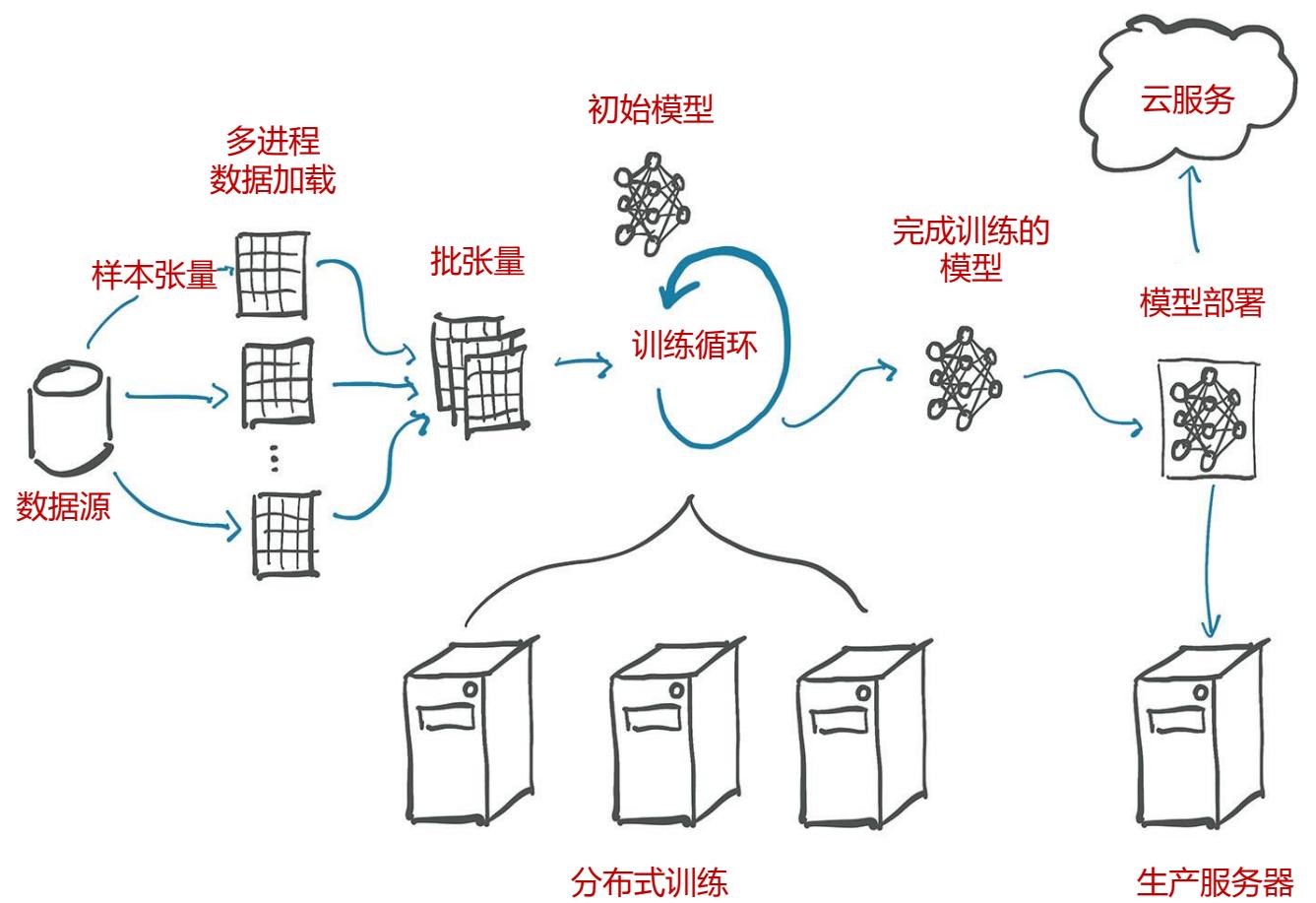
深度学习框架 PyTorch 简介

PYTORCH

PyTorch 是由 Facebook 开发的开源深度学习框架。

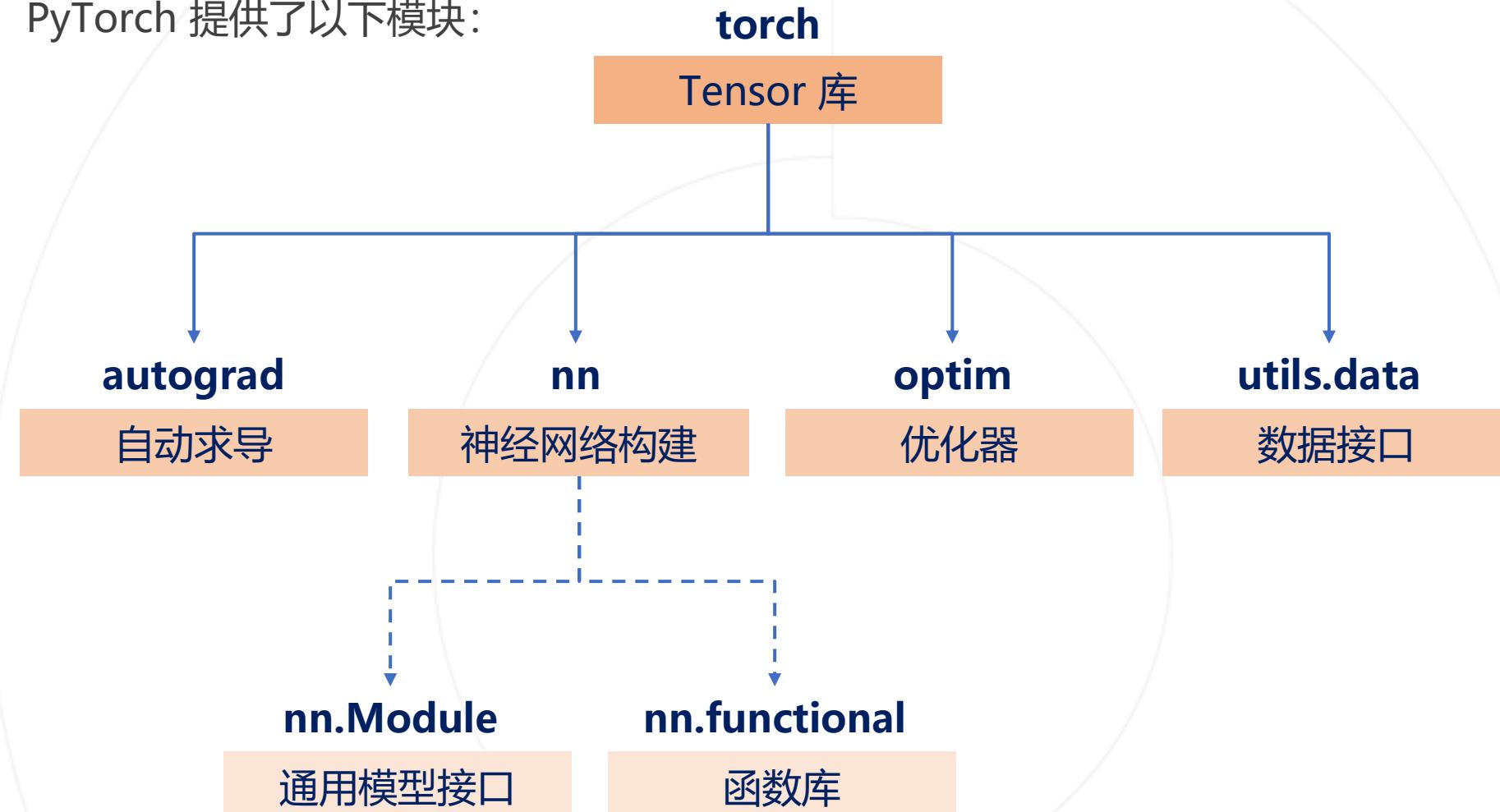
PyTorch 提供了完整的工具链用于构建、训练和部署深度学习模型。

<https://pytorch.org/>



使用 PyTorch 进行模型生产的流程

针对深度学习模型构建与训练，PyTorch 提供了以下模块：



torch 库

多维数组的数据结构
Tensor

多维数组的运算

多计算后端支持

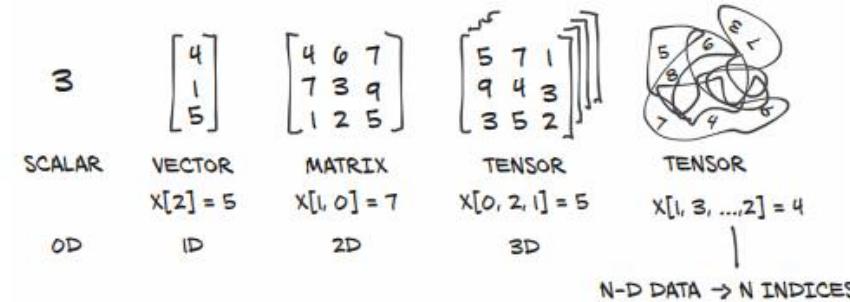
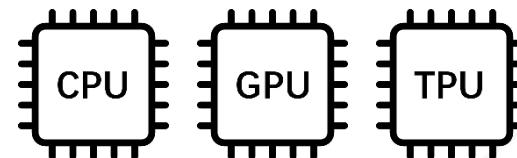


Diagram illustrating element-wise addition of two 2x2 tensors:

data ones =
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix}$$

Diagram illustrating element-wise addition of a 2x3 tensor and a 2x2 row vector:

data ones_row = data ones_row =
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 4 & 5 \\ 6 & 7 \end{bmatrix}$$



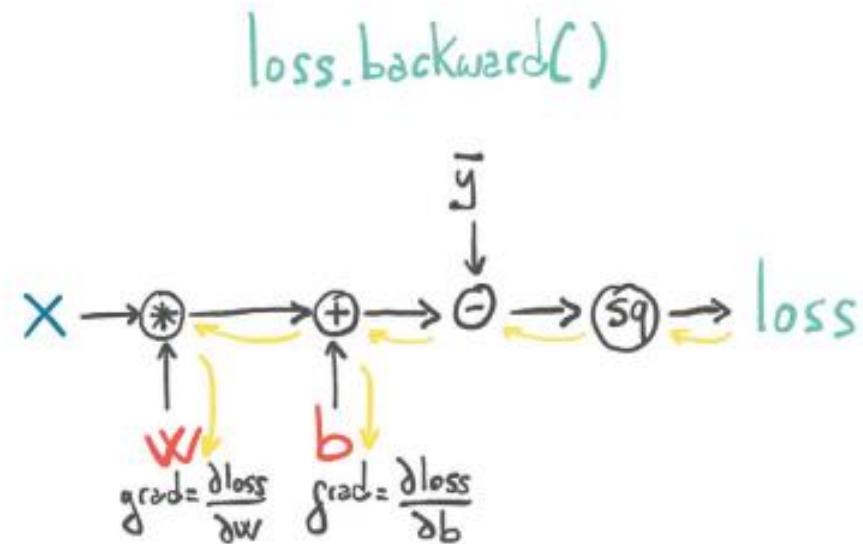
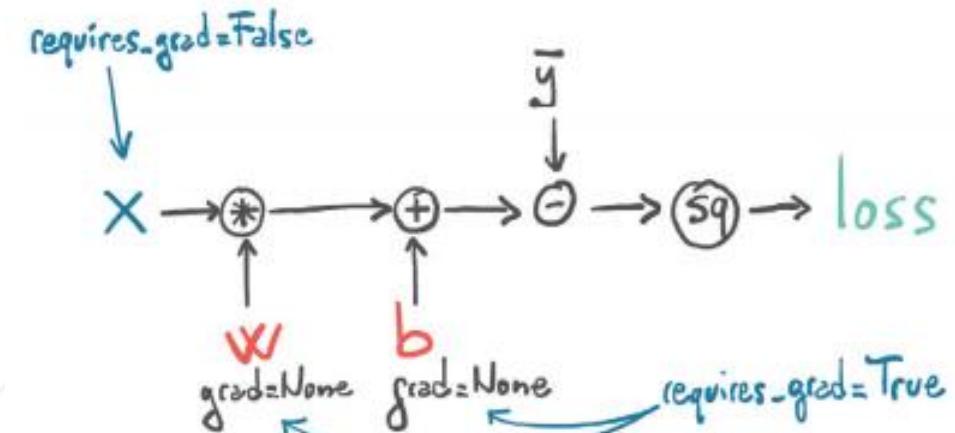
例：梯度下降求解线性回归：

$$L = \frac{1}{2} \sum_{i=1}^N (y_i - (w^T x_i + b))^2$$

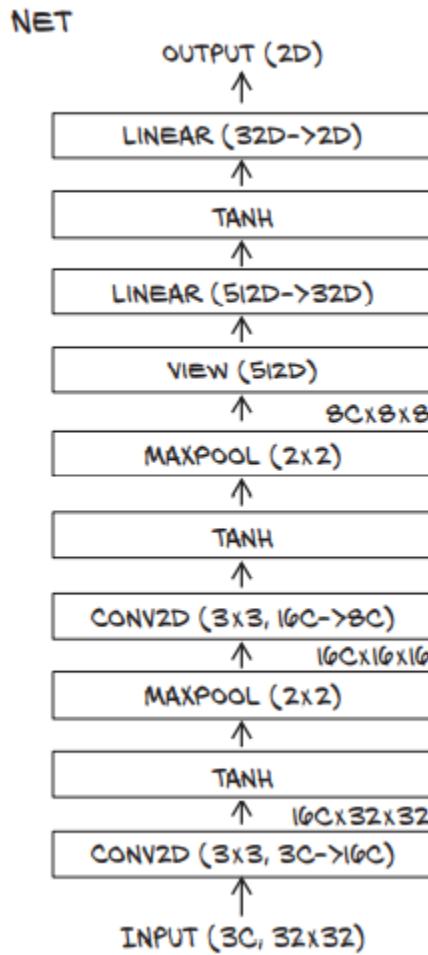
借助 PyTorch 自动求导：

```
loss = 0.5 * (Y - X.T @ w - b) ** 2
loss = loss.sum()
loss.backward()
```

loss 对 w 和 b 的导数存储在 w.grad 和 b.grad 中



torch.nn.Module 定义了神经网络模型的抽象接口



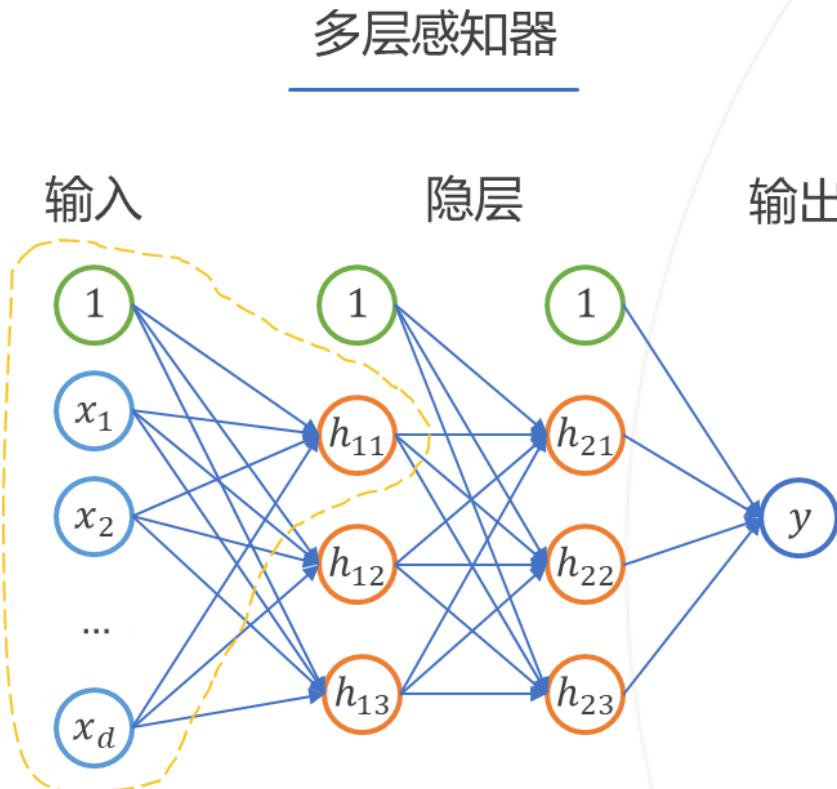
```

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
        self.tanh1 = nn.Tanh()
        self.pool1 = nn.MaxPool2d(kernel_size=(2,2))
        self.conv2 = nn.Conv2d(16, 8, kernel_size=3, padding=1)
        self.tanh2 = nn.Tanh()
        self.pool2 = nn.MaxPool2d(kernel_size=(2,2))
        self.fc1 = nn.Linear(8*8*8, 32)
        self.tanh3 = nn.Tanh()
        self.fc2 = nn.Linear(32, 2)

    def forward(self, x):
        out = self.pool1(self.tanh1(self.conv1(x)))
        out = self.pool2(self.tanh2(self.conv2(out)))
        out = out.view(-1, 8*8*8)
        out = self.tanh3(self.fc1(out))
        out = self.fc2(out)
        return out
  
```

可调用对象 Callable object
`net = Net()`
`pred = net(img_batch)`

torch.nn.Module 定义了神经网络模型的抽象接口



```
class MLP(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.linear1 = nn.Linear(10, 3)
        self.tanh1 = nn.Tanh()
        self.linear2 = nn.Linear(3, 3)
        self.tanh2 = nn.Tanh()
        self.linear3 = nn.Linear(3, 1)

    def forward(self, x):
        out = self.linear1(x)
        out = self.tanh1(out)
        out = self.linear2(out)
        out = self.tanh2(out)
        out = self.linear3(out)
        return out
```

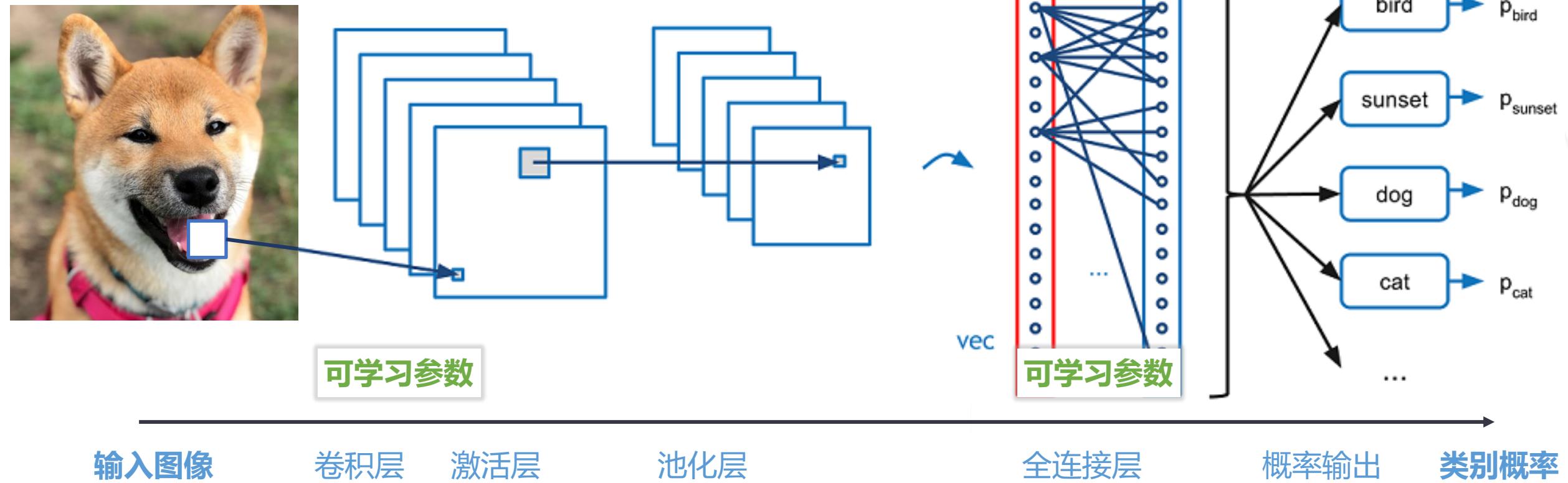
- 模型和其中的层通常都是 torch.nn.Module 的实例
- Torch.nn.Module 是可调用对象 (Callable Object)

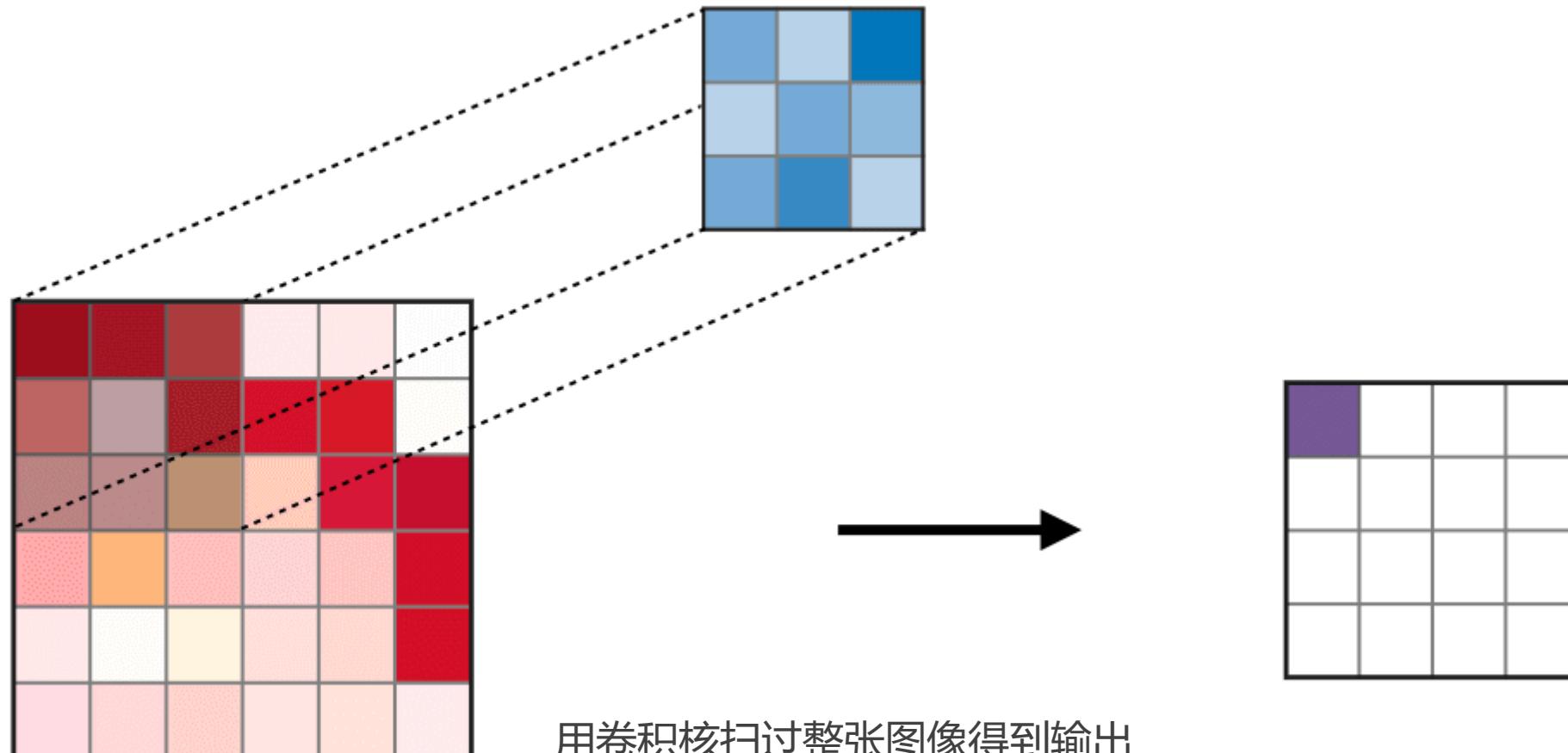
```
model = MLP()
pred = model(data)
```

卷积神经网络

卷积神经网络的整体结构

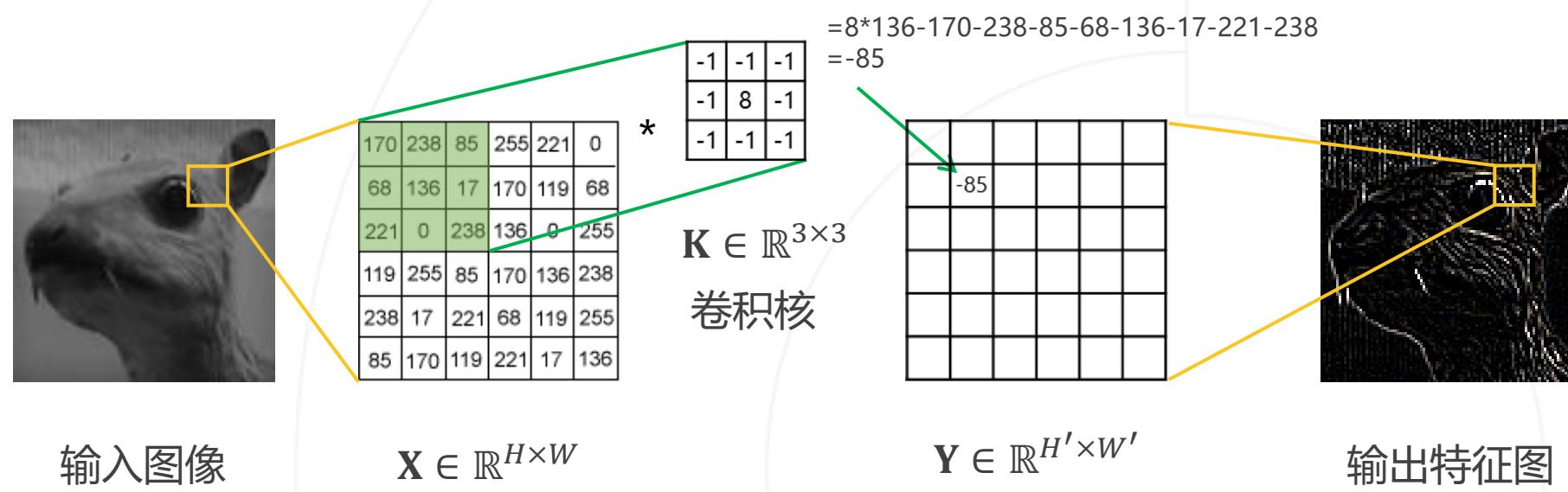
OpenMMLab





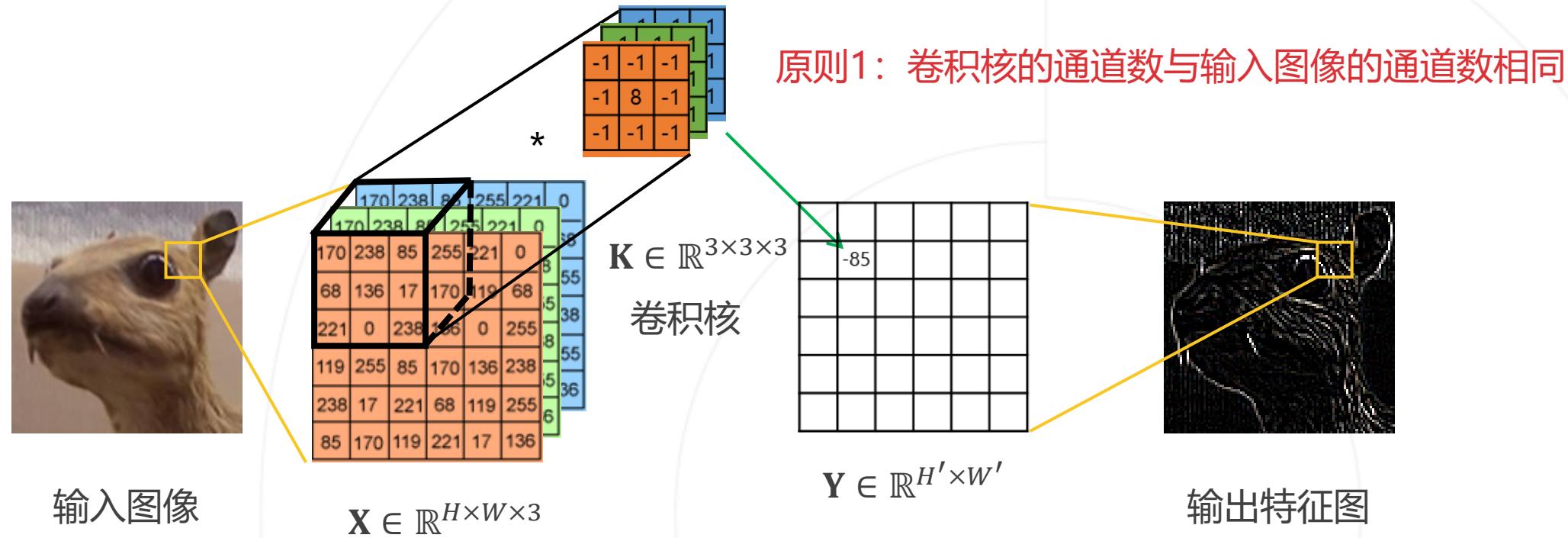
用卷积核扫过整张图像得到输出

$$Y[h, w] = \sum_{i=-1}^1 \sum_{j=-1}^1 X[h + i, w + j] K[i, j]$$



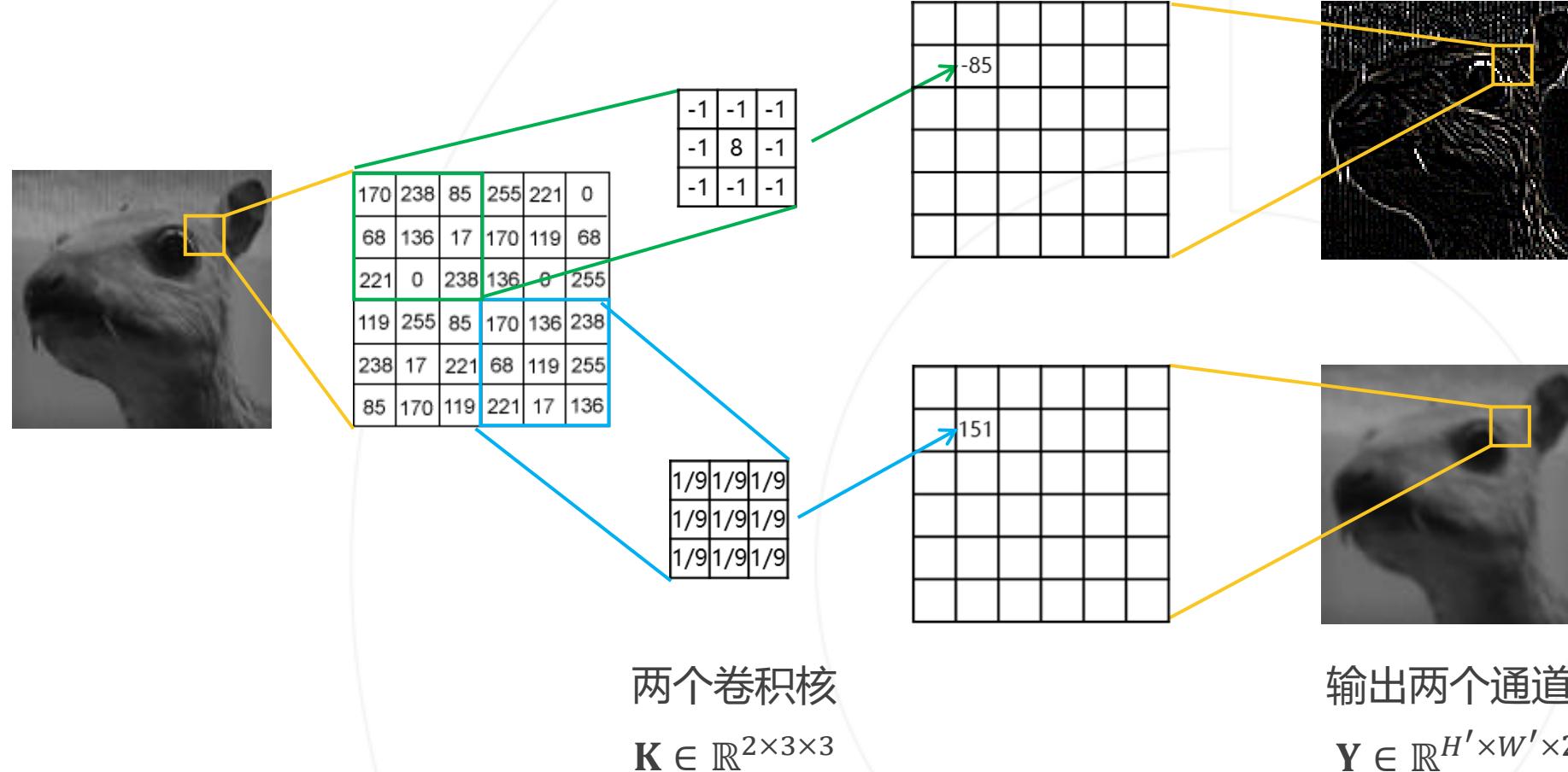
用卷积核扫过整张图像得到输出

$$\mathbf{Y}[h, w] = \sum_{i=-1}^1 \sum_{j=-1}^1 \mathbf{X}[h + i, w + j] \mathbf{K}[i, j]$$



$$Y[h, w] = \sum_{i=-1}^1 \sum_{j=-1}^1 \sum_{k=1}^3 X[h + i, w + j, k] K[i, j, k]$$

使用多个卷积核

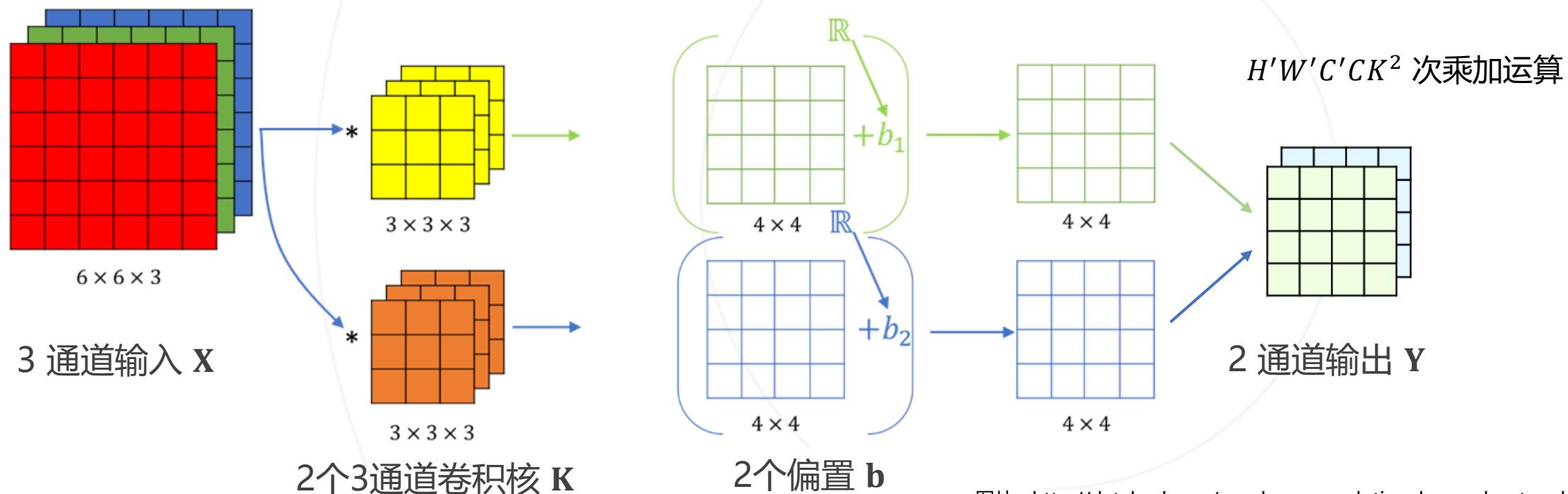


原则2：输出特征图的通道数与卷积核的个数相同

卷积层通过卷积运算，将 C 通道的特征图 $\mathbf{X} \in \mathbb{R}^{H \times W \times C}$ 变换为 C' 通道的特征图 $\mathbf{Y} \in \mathbb{R}^{H' \times W' \times C'}$ ，计算公式为：

$$\mathbf{Y}[h, w, c] = \sum_{i=1}^K \sum_{j=1}^K \sum_{k=1}^C \mathbf{X}[h+i, w+j, k] \mathbf{K}_c[i, j, k] + \mathbf{b}[c]$$

其中， $\mathbf{K} \in \mathbb{R}^{C' \times K \times K \times C}$ 为 C' 个 C 通道的卷积核， $\mathbf{b} \in \mathbb{R}^{C'}$ 为每个输出通道对应的偏置，二者均为可学习的参数。



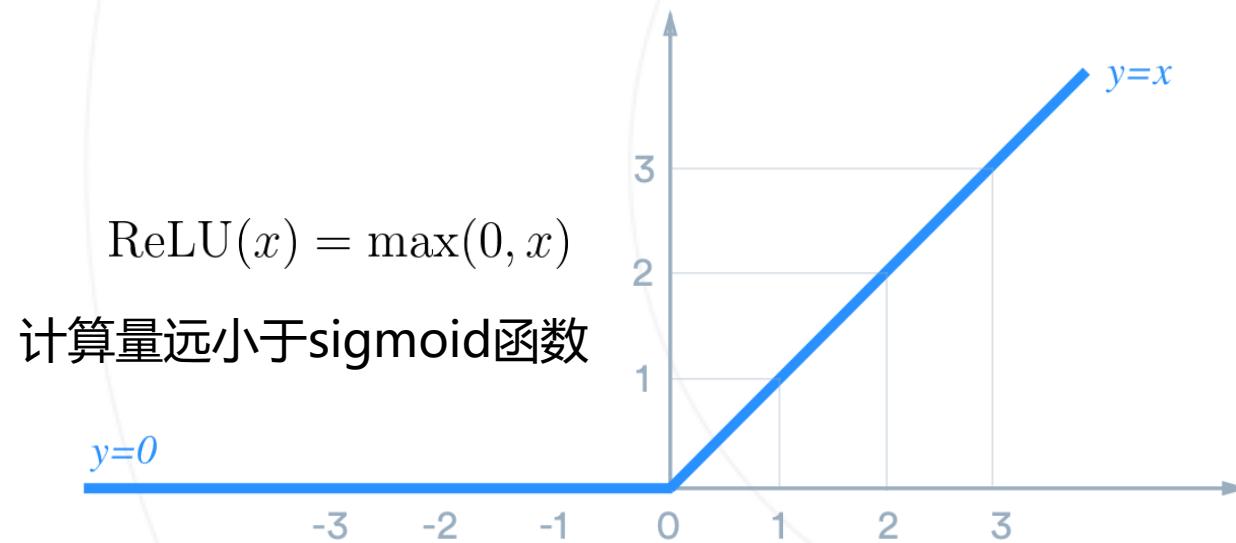
```
# Functional API

```

- 激活层基于一个非线性函数 $\sigma: \mathbb{R} \rightarrow \mathbb{R}$ 对输入特征图 $\mathbf{X} \in \mathbb{R}^{H \times W \times C}$ 进行逐元素的变换：

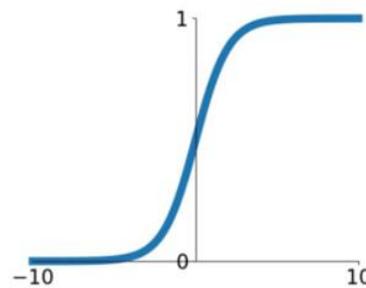
$$\mathbf{Y}[h, w, c] = \sigma(\mathbf{X}[h, w, c])$$

- 激活层通常不包含可学习参数。
- ReLU (Rectified Linear Unit) 是卷积网络中最常用的非线性激活函数。



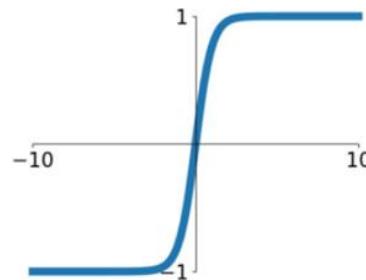
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



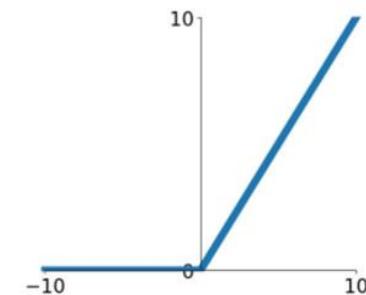
tanh

$$\tanh(x)$$



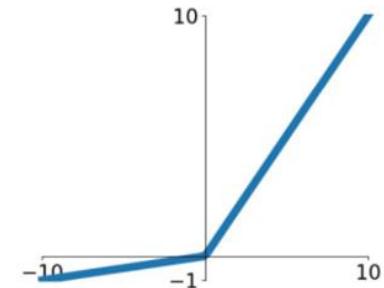
ReLU

$$\max(0, x)$$



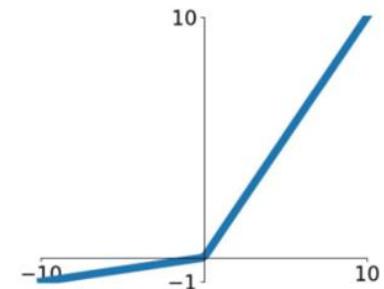
Leaky ReLU

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



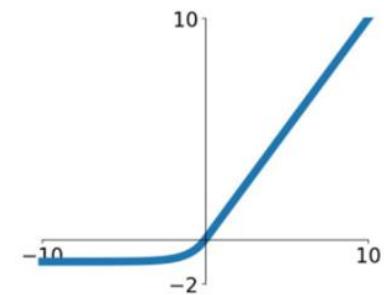
PReLU

$$\max(ax, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



```
x = torch.randn(3, 3)
```

```
# Functional API
```

```
y = F.relu(x)
```

```
# Class API
```

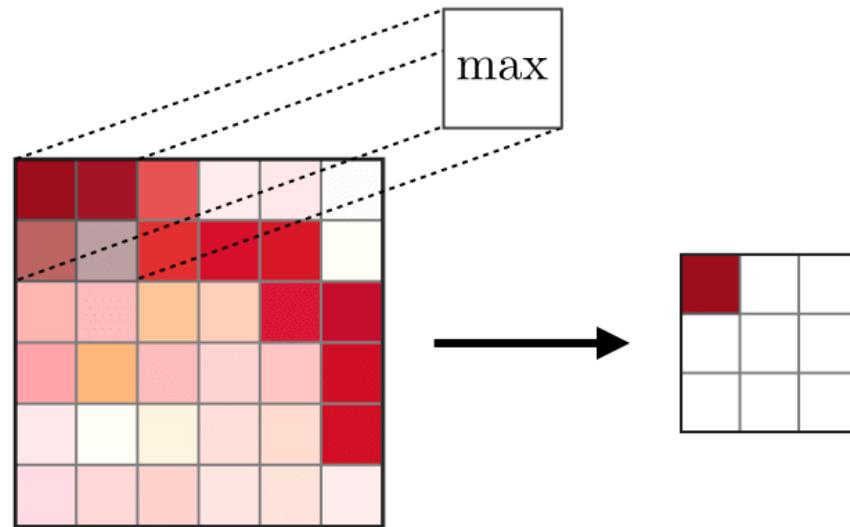
```
relu = nn.ReLU()
```

```
y = relu(x)
```

池化层在特征图的局部区域内计算最大值或平均值，从而降低特征图分辨率，节省计算量，提高特征的空间鲁棒性。

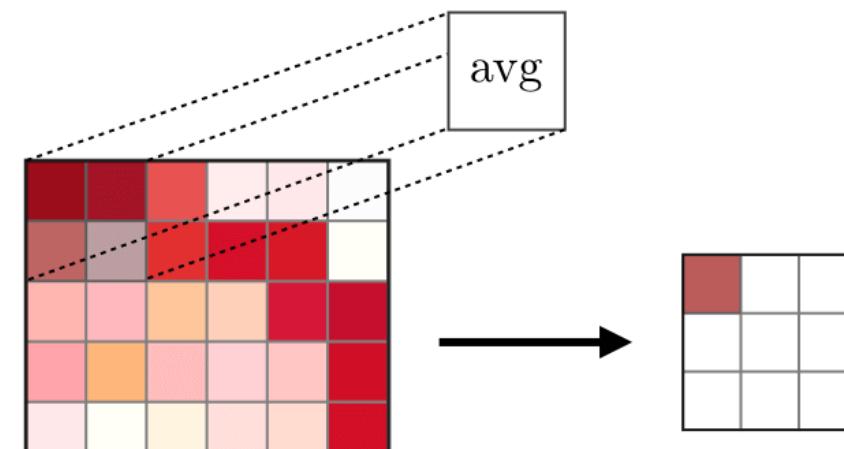
最大池化

$$\mathbf{Y}[h, w, c] = \max_{1 \leq i, j \leq k} \mathbf{X}[sh + i, sw + j, c]$$

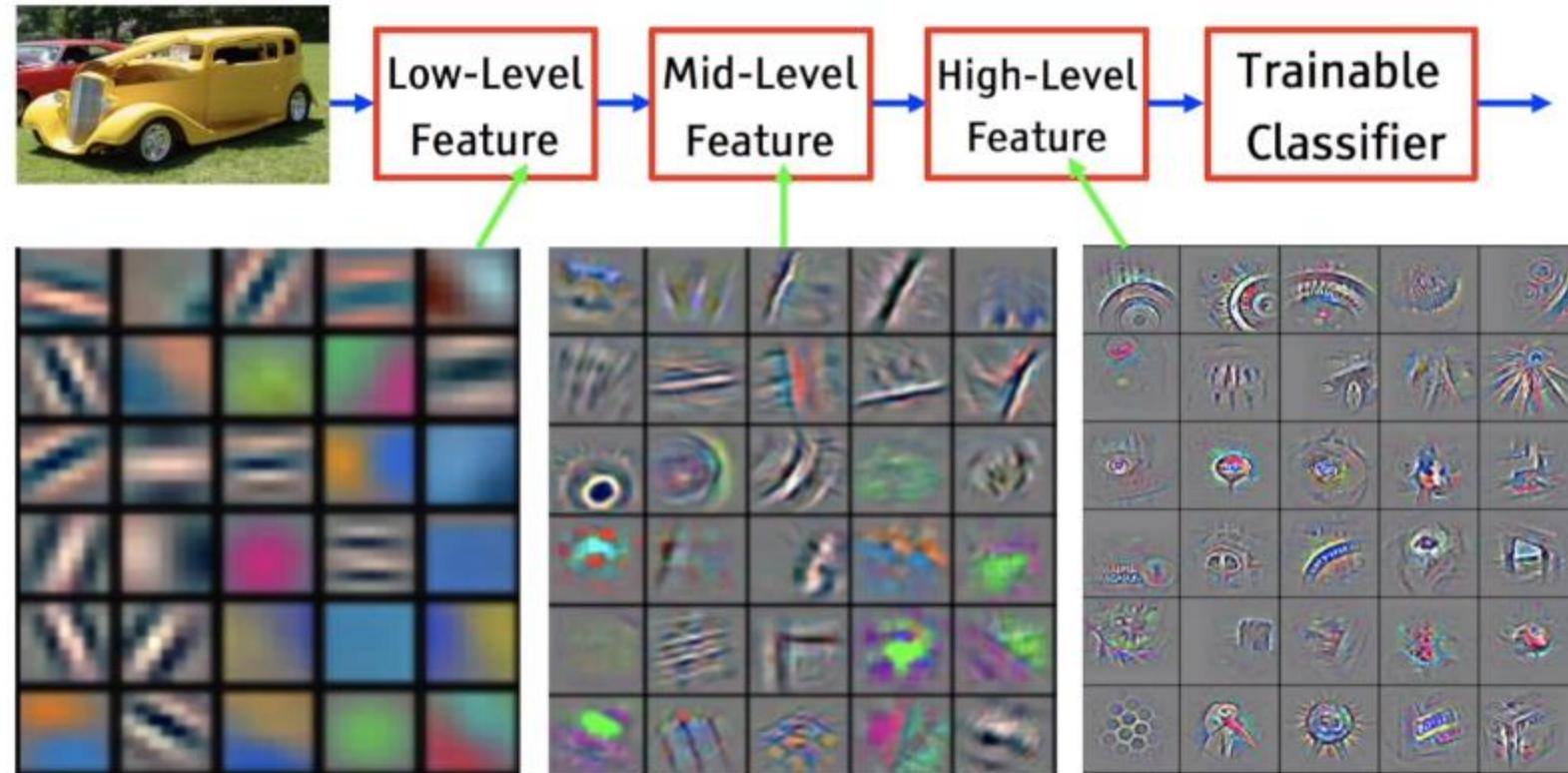


平均池化

$$\mathbf{Y}[h, w, c] = \frac{1}{k^2} \sum_{1 \leq i, j \leq k} \mathbf{X}[sh + i, sw + j, c]$$



池化层使深层神经元具有更大的感受野（获取原图信息的空间范围），从而帮助深层特征提取提取高级语义信息。



```
x = torch.randn(1, 1, 4, 4)

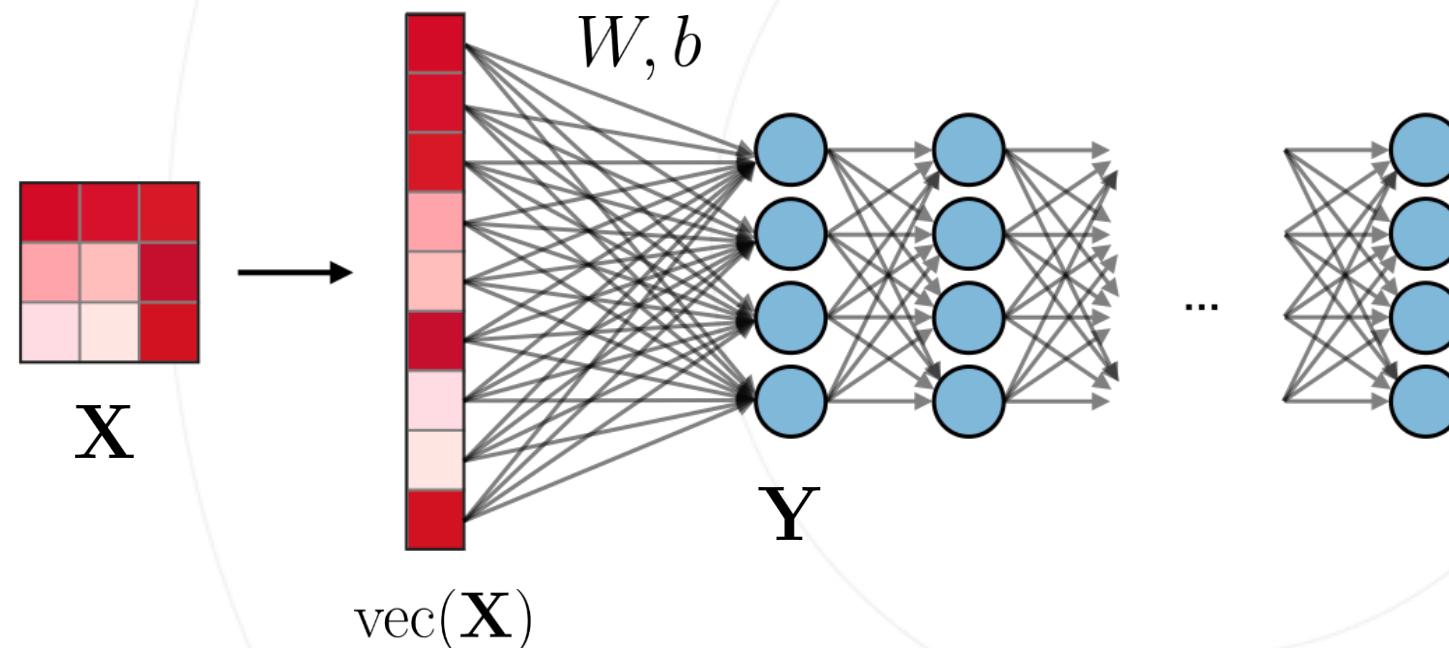
# Functional API
y = F.max_pool2d(x, kernel_size=2, stride=2)

# Class API
pool = nn.MaxPool2d(kernel_size=(2, 2), stride=2)
y = pool(x)
```

- 全连接层通过矩阵乘法将输入特征 $\mathbf{X} \in \mathbb{R}^N$ 映射为输出特征 $\mathbf{Y} \in \mathbb{R}^M$

$$\mathbf{Y} = W\mathbf{X} + b \quad \text{或} \quad \mathbf{Y} = W\text{vec}(\mathbf{X}) + b$$

- 全连接层包含可学习参数 $W \in \mathbb{R}^{M \times N}, b \in \mathbb{R}^M$ 。
- 多层感知器中的层就是全连接层。



```
# Vectorize
fmap = torch.randn(1, 2, 2, 2)
x = fmap.view(-1, 8)

# Functional API
weight = torch.randn(6, 8)
bias = torch.randn(6)
y = F.linear(x, weight, bias)

# Class API
linear = nn.Linear(8, 6) # in_features=8, out_features=6
y = linear(x)
```

- 概率输出层 $S: \mathbb{R}^K \rightarrow [0,1]^K$ 将任意特征向量转换为概率分布向量

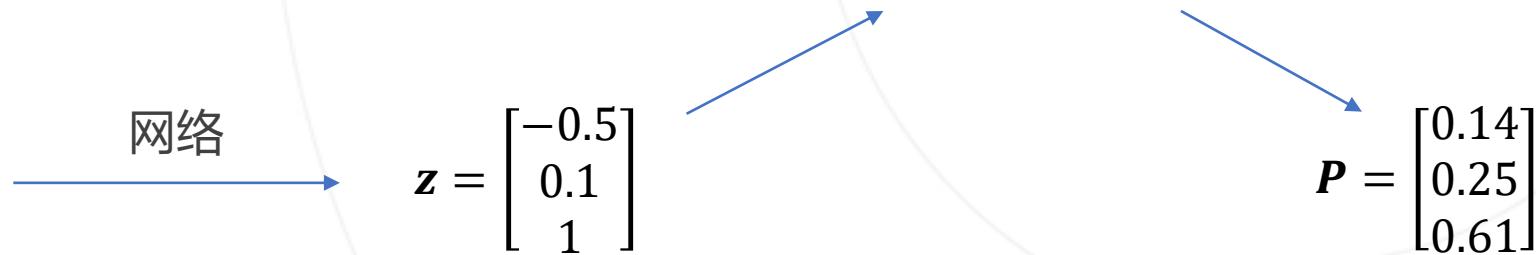
$$P(y|x) = S(h(x; \Theta))$$

- 二分类，全连接层输出标量 $z \in \mathbb{R}$ ，通过 Sigmoid 函数计算正类的概率

$$P(y = 1|x) = \frac{1}{1 + \exp(-z)}$$

- 多分类，全连接层输出 K 维向量 $z \in \mathbb{R}^K$ ，通过 Softmax 函数计算 K 类的概率

$$P(y = k|x) = \frac{\exp z_k}{\sum_{i=1}^K \exp(z_k)}$$



PyTorch 实现 Softmax



```
z = torch.rand(3)
```

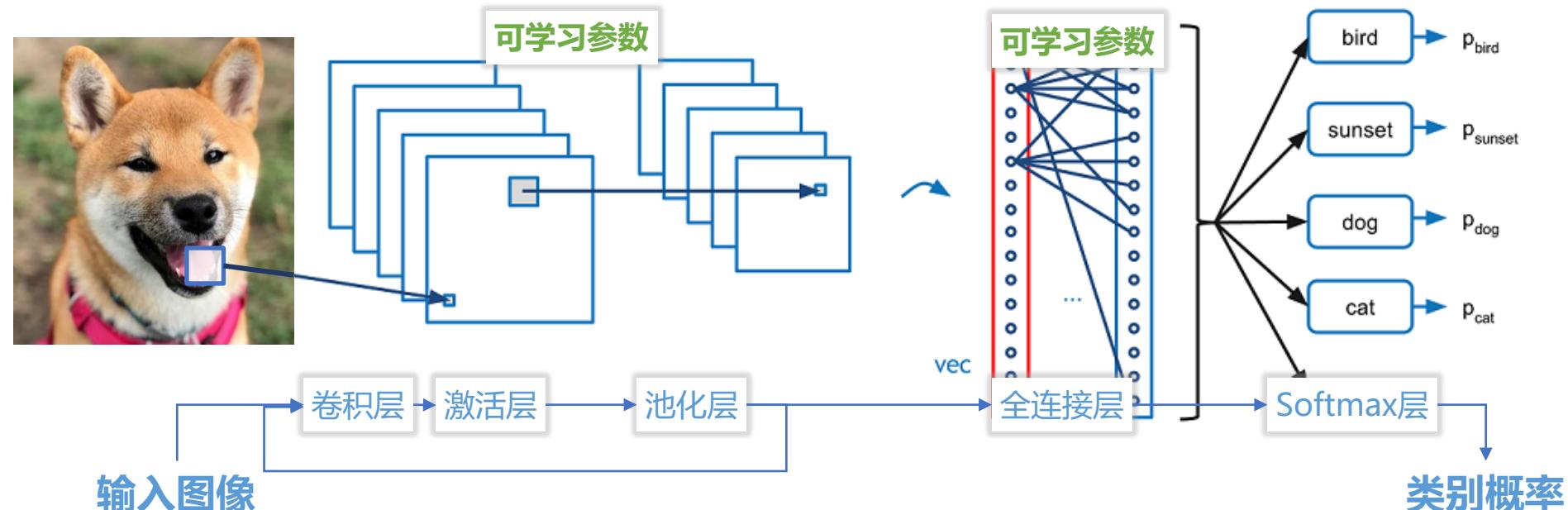
```
# Functional API
```

```
p = F.softmax(z)
```

```
# Class API
```

```
s = nn.Softmax()
```

```
p = s(z)
```



一个端到端学习的分类器 $P(y|x) = h(x; \Theta)$

学习一个好的图像特征

视觉任务的初衷

简单的分类器完成分类

稳健表达图像内容，不受像素外观影响

图像分类模型设计

—— 构建“好”的函数族

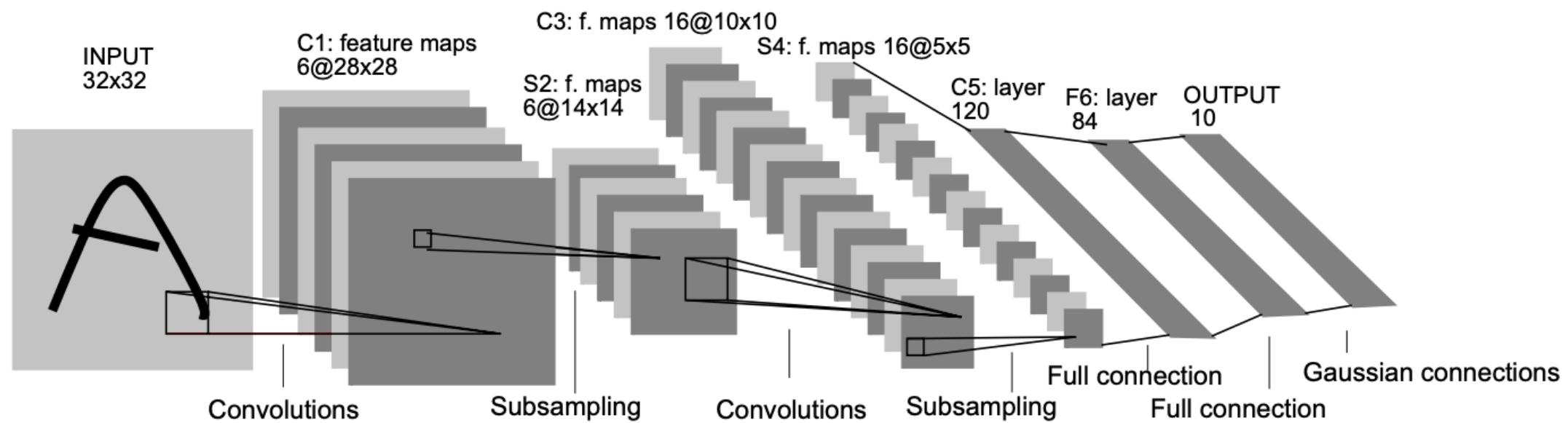
图像分类网络通常可以分为主干网络和分类器两部分



最早的卷积神经网络之一。

LeNet-5 由 7 层网络组成，其中 2 层为卷积层，2 层为下采样层，3 层为全连接层。

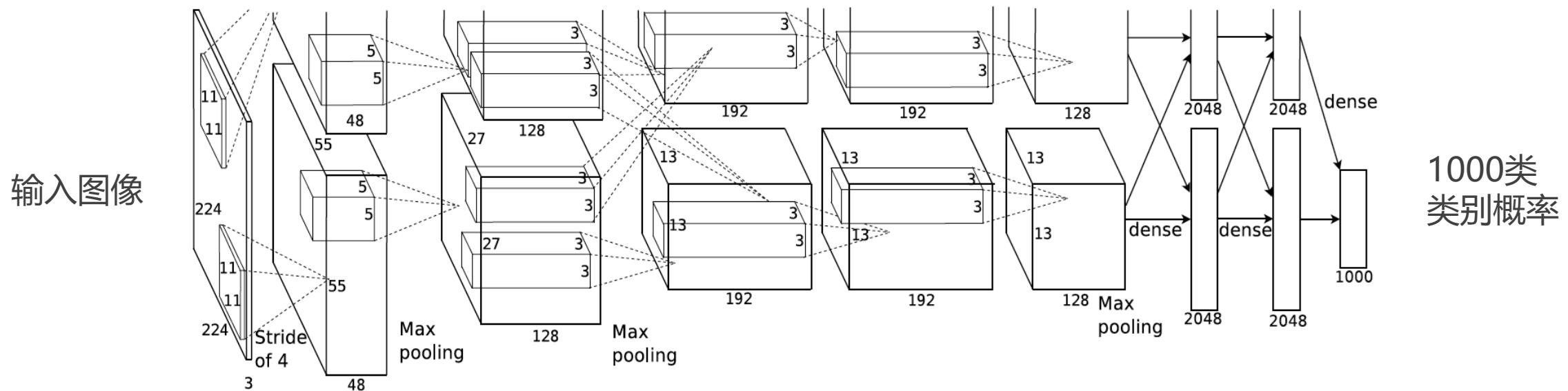
60K 个可学习参数。



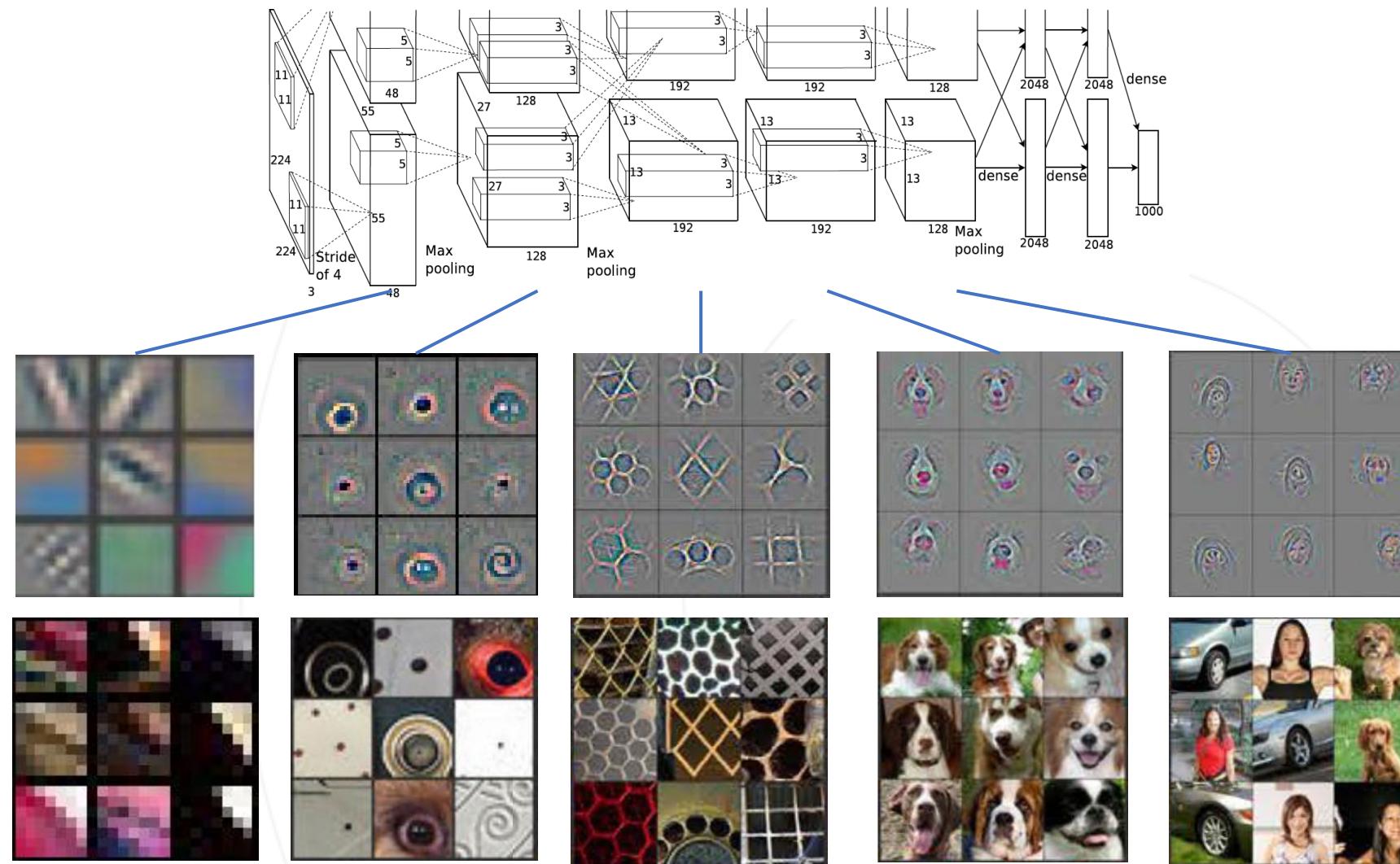
第一个成功应用于大规模图像分类的卷积神经网络模型。

8 个可学习层，5 个卷积层，3 个全连接层，共有 60M 个可学习参数。

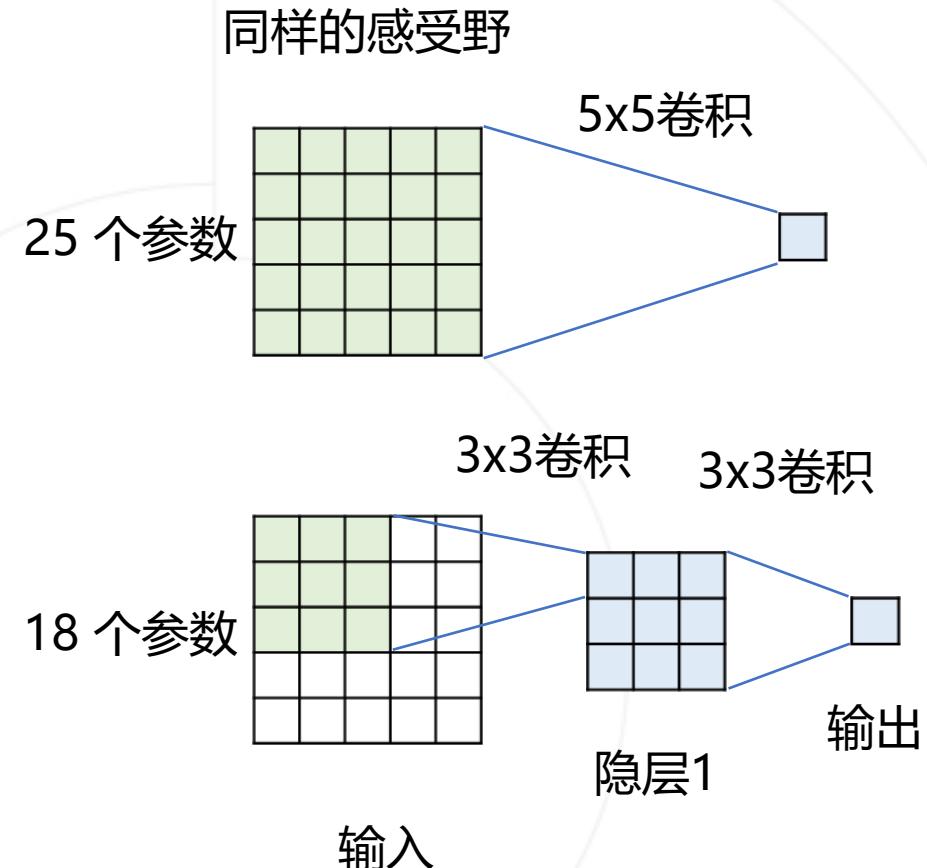
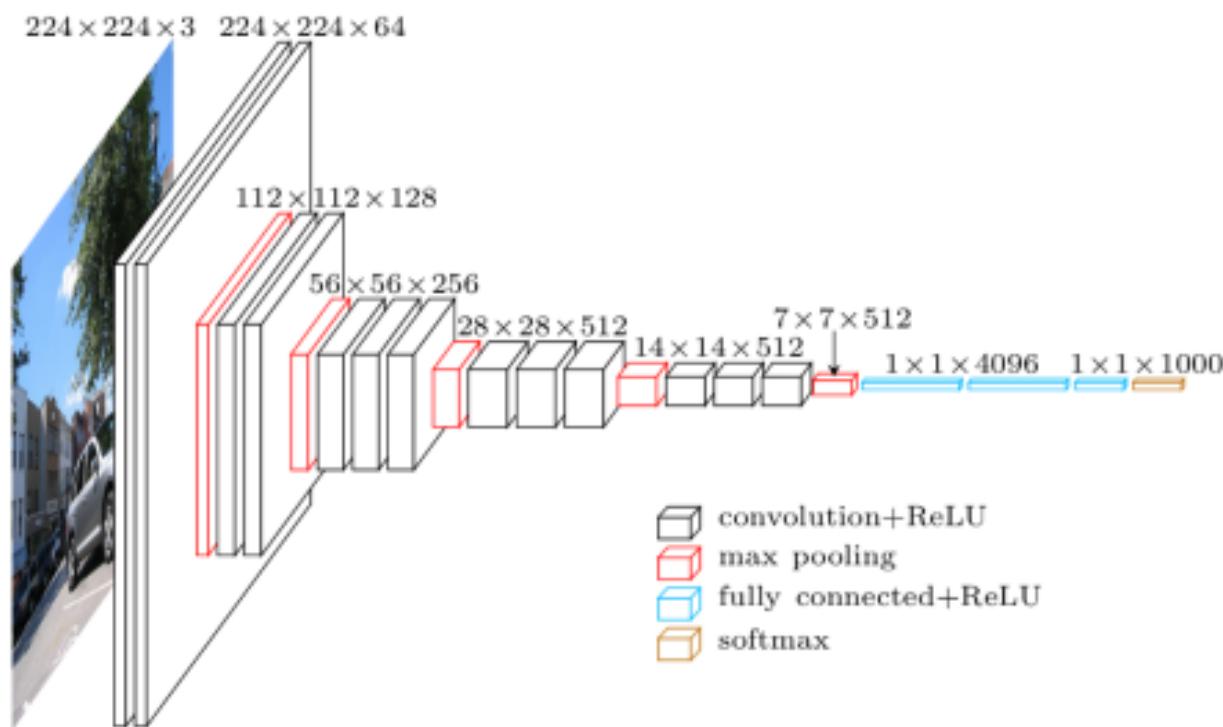
使用 2 个 NVIDIA GTX 580 GPU 训练了 1 周。



卷积网络中的多层次特征



- 只使用 3×3 的卷积核
- 层数增多：16~19层
- 边界填充1像素，维持空间分辨率
- 倍增通道数的同时减半空间分辨率

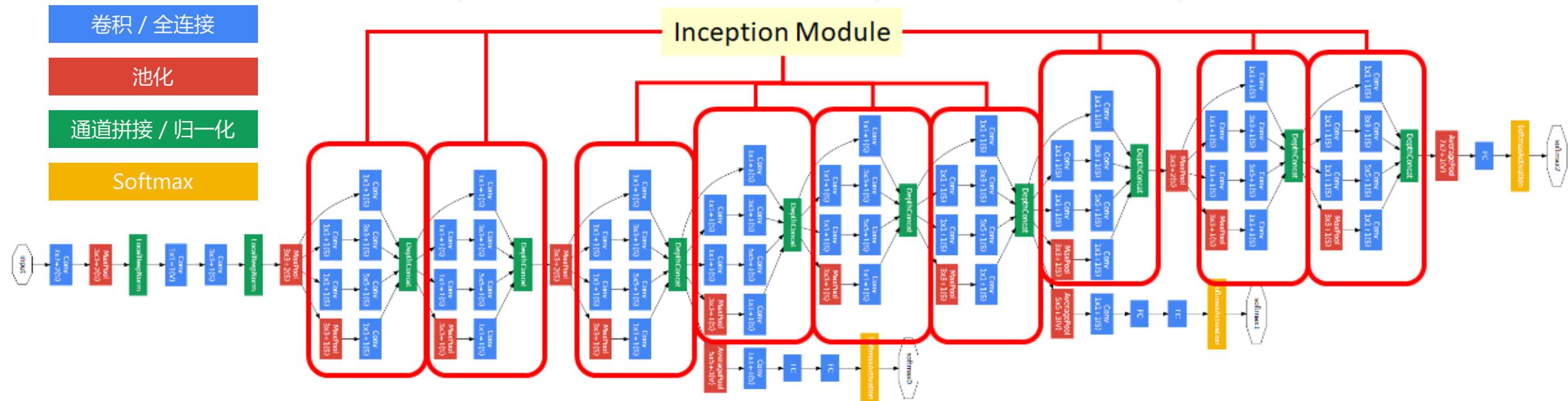


- VGG-16 有 138M 权重参数，其中超过 100M 来自全连接层。

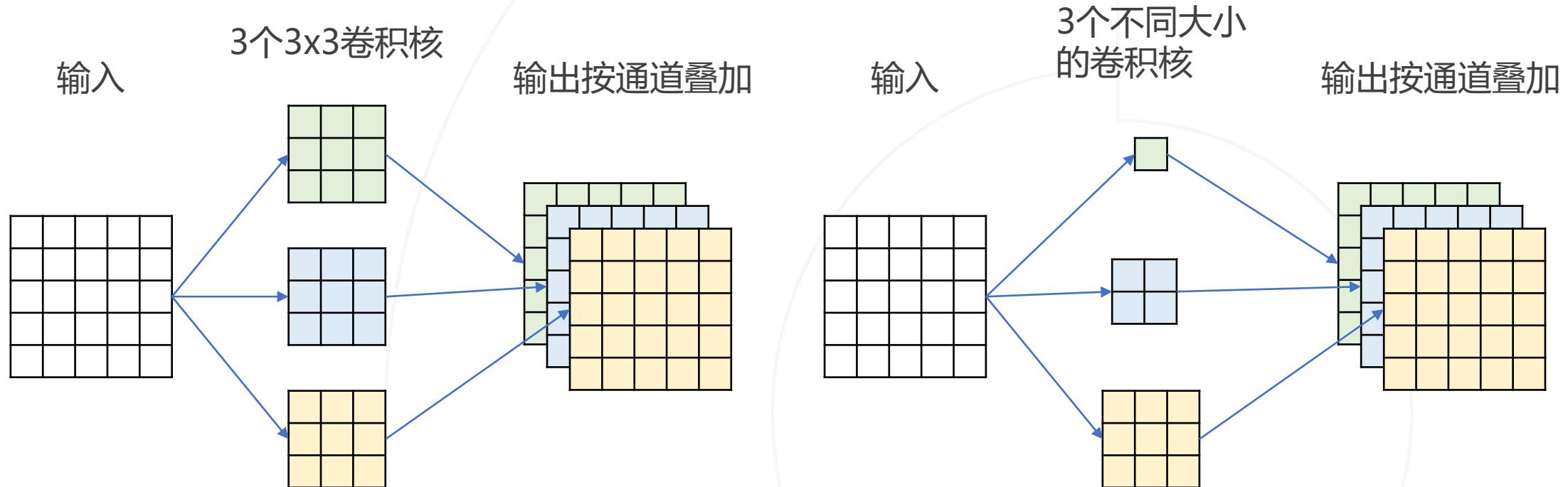
VGG16 - Structural Details													
#	Input Image			output			Layer	Stride	Kernel	in	out	Param	
1	224	224	3	224	224	64	conv3-64	1	3	3	3	64	1792
2	224	224	64	224	224	64	conv3064	1	3	3	64	64	36928
	224	224	64	112	112	64	maxpool	2	2	2	64	64	0
3	112	112	64	112	112	128	conv3-128	1	3	3	64	128	73856
4	112	112	128	112	112	128	conv3-128	1	3	3	128	128	147584
	112	112	128	56	56	128	maxpool	2	2	2	128	128	65664
5	56	56	128	56	56	256	conv3-256	1	3	3	128	256	295168
6	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
7	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
	56	56	256	28	28	256	maxpool	2	2	2	256	256	0
8	28	28	256	28	28	512	conv3-512	1	3	3	256	512	1180160
9	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
10	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
	28	28	512	14	14	512	maxpool	2	2	2	512	512	0
11	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
12	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
13	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
	14	14	512	7	7	512	maxpool	2	2	2	512	512	0
14	1	1	25088	1	1	4096	fc		1	1	25088	4096	102764544
15	1	1	4096	1	1	4096	fc		1	1	4096	4096	16781312
16	1	1	4096	1	1	1000	fc		1	1	4096	1000	4097000
Total											138,423,208		

GoogLeNet (Inception v1, 2014)

- 使用 Inception 模块，在同一层使用不同尺寸的卷积核
- 22 个可学习层，仅 7M 权重参数

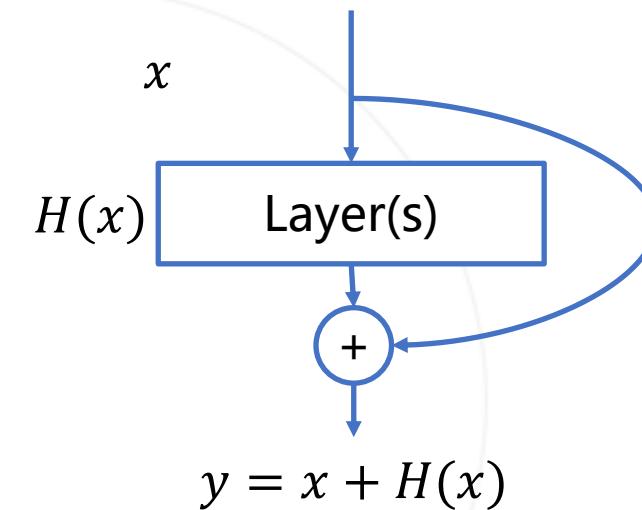
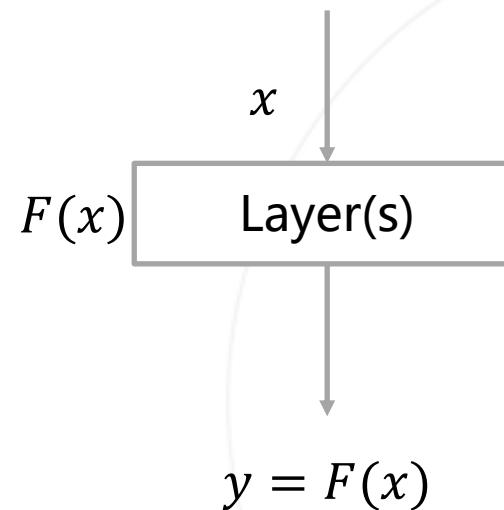


使用不同大小的卷积核

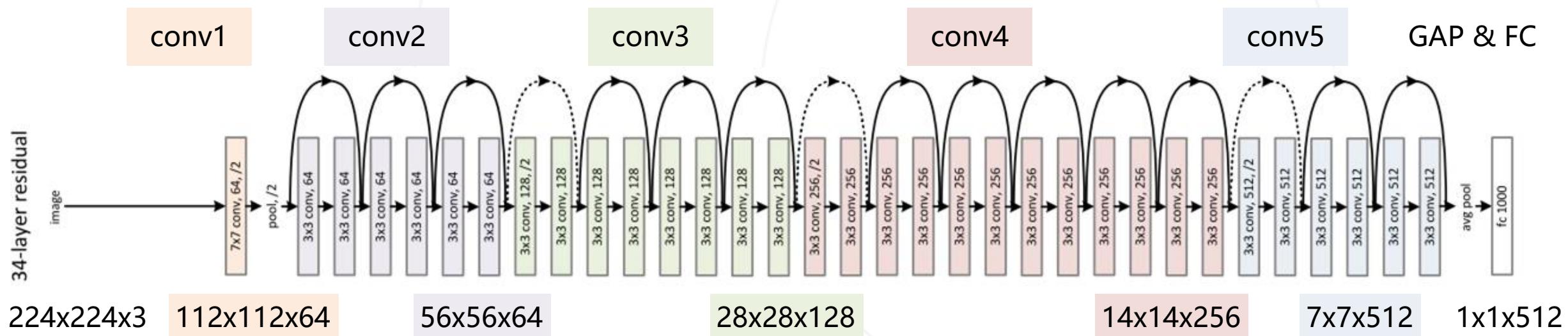


- 迄今影响力最大、使用最广泛的模型结构之一，获得 CVPR 2016 最佳论文奖。
- 残差学习：**引入了跨层连接，将网络深度提高到了上百层。

$$y = x + H(x)$$

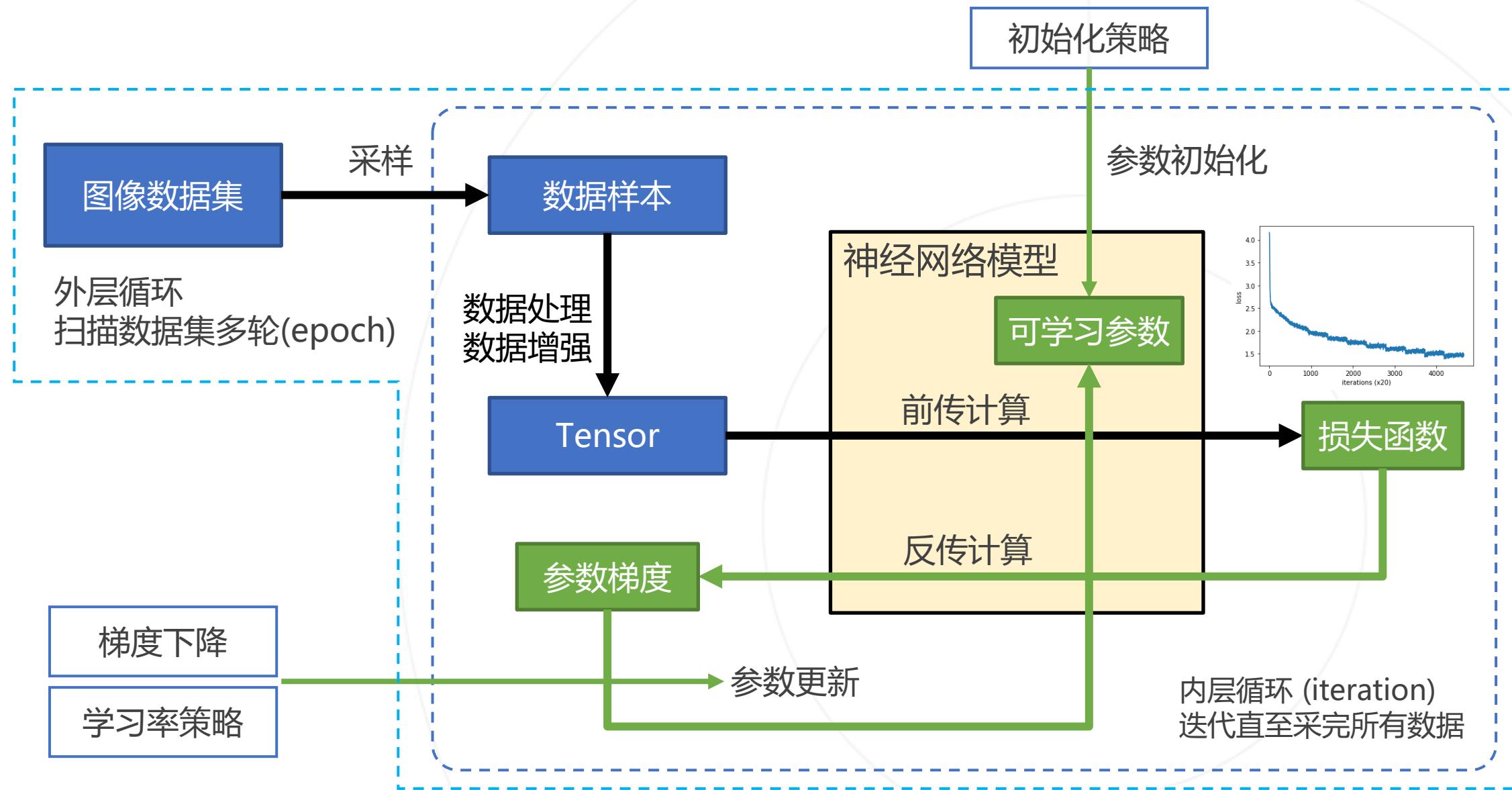


- 5 级，每级包含若干残差模块。不同残差模块个数 → 不同 ResNet 结构。
- 每级输出分辨率减半，通道倍增
- 全局平均池化压缩空间维度
- 单层全连接层产生类别概率



图像分类模型训练

—— 在函数族中寻找“好”的函数



对于预测类别概率 $P \in [0,1]^K$ 和真值 y ，定义交叉熵损失为：

$$l(P, y) = -\log P_y$$

P_y 的减函数， $P_y = 1$ 时取最小

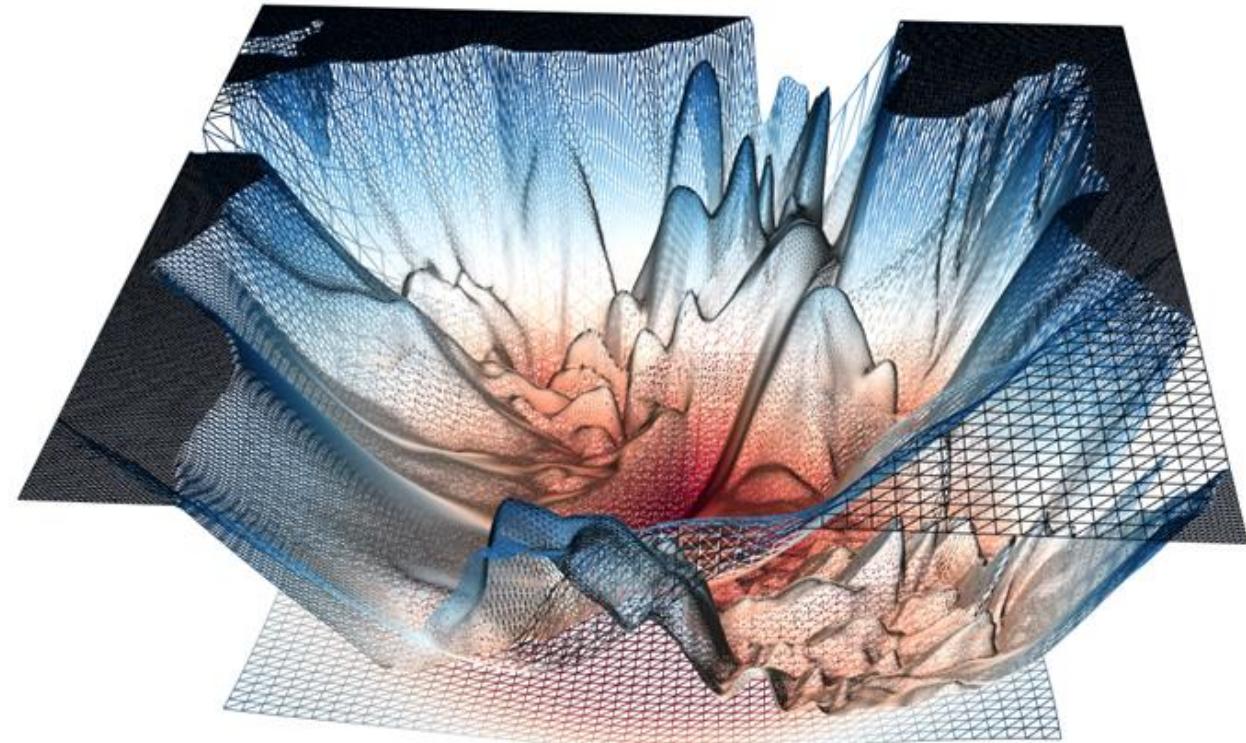
此时 $P = [0, 0, \dots, 1, \dots, 0]$ ，中只有第 y 项为 1， $l(P, y) = 0$

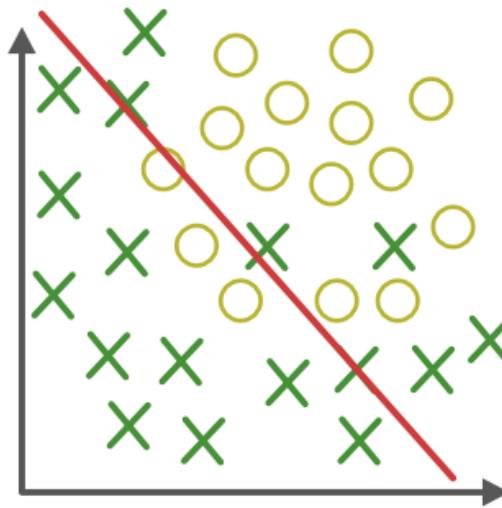
1. 数据量大、计算量大

➤ 例：在 ImageNet 数据集上训练 AlexNet， $N \approx 1\ 000\ 000$, $\dim(\Theta) \approx 60\ 000\ 000$

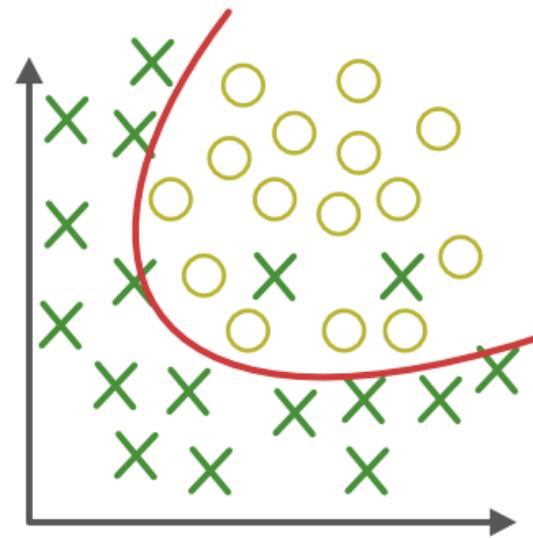
2. 损失函数高度复杂：高维、非凸

3. 模型复杂度高，容易过拟合

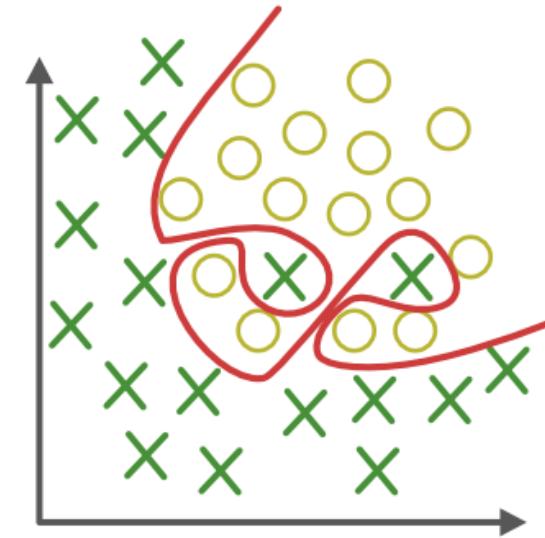




欠拟合



拟合



过拟合

没有捕捉数据中的规律
不能准确预测未来数据
模型过于简单

捕捉到数据中的规律
可以准确预测未来数据

过度拟合到数据中的**噪声**
不能准确预测未来数据
模型过于复杂、且数据不够

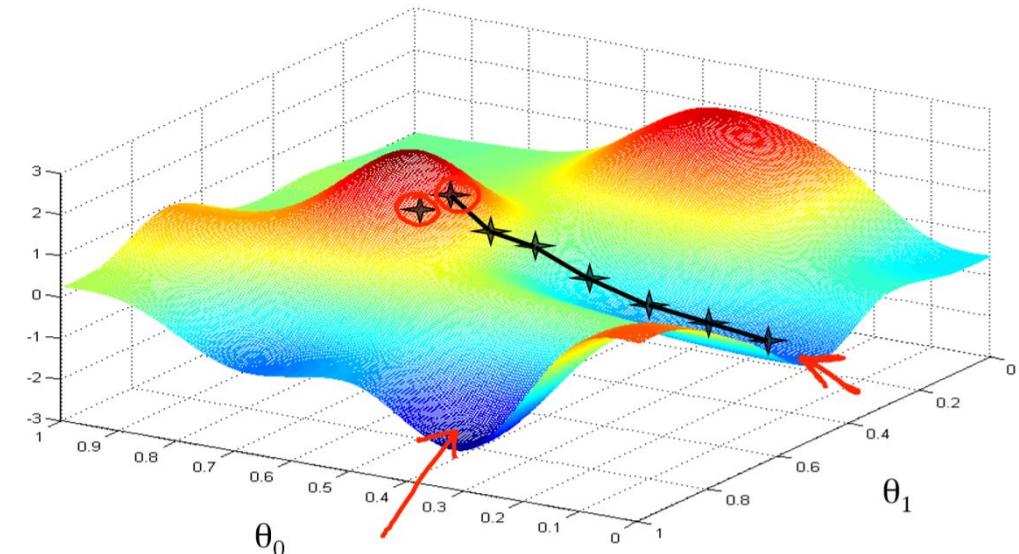
- 给定数据集 $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ ，模型 $h(x; \Theta)$ 和损失函数 $l(P, y)$ ，

$$L(\Theta | \mathcal{D}) := \frac{1}{N} \sum_{i=1}^N l(h(x_i; \Theta), y_i)$$

很大

- 随机初始化参数 $\Theta^{(0)}$
- 迭代直至收敛：
 - 前向+反向传播计算梯度 $\nabla_{\Theta} L(\Theta^{(t-1)})$
 - 参数更新 $\Theta^{(t)} = \Theta^{(t-1)} - [\eta] \nabla_{\Theta} L(\Theta^{(t-1)})$

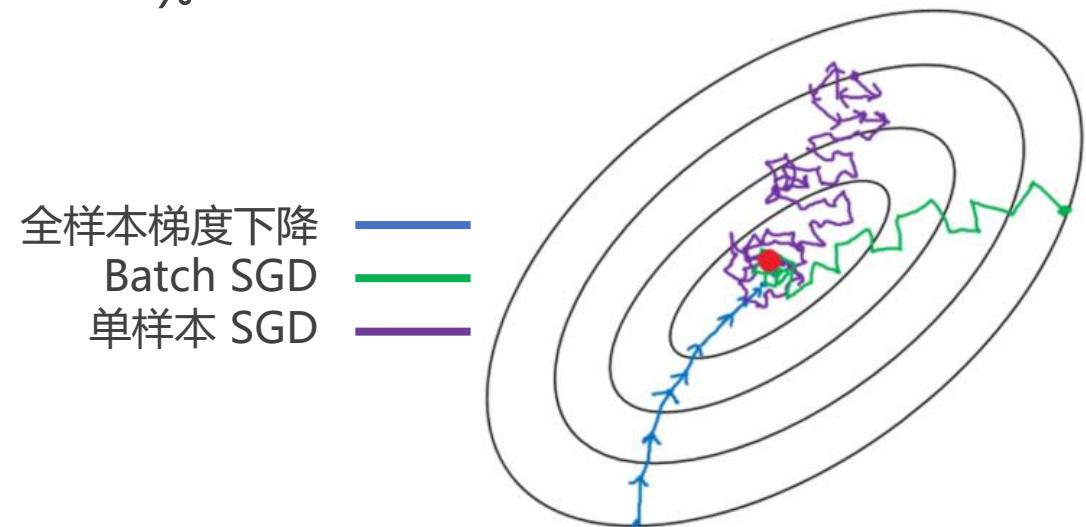
学习率



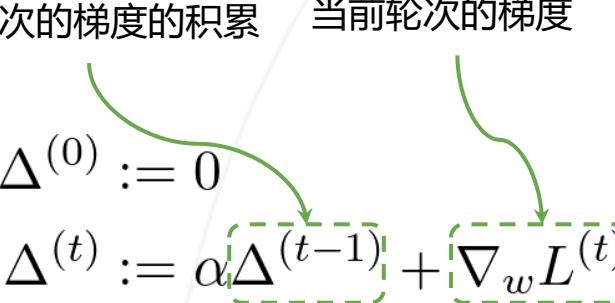
- 每次迭代随机选取指标集 $\mathcal{B}^{(t)} = \{i_1, i_2, \dots, i_B\}$ 近似计算损失函数的梯度

$$L_{\mathcal{B}}(\Theta | \mathcal{D}) = \frac{1}{B} \sum_{i=1}^B l(h(x_{i_j}; \theta), y_{i_j}) \approx \frac{1}{N} \sum_{i=1}^N l(h(x_i; \theta), y_i)$$

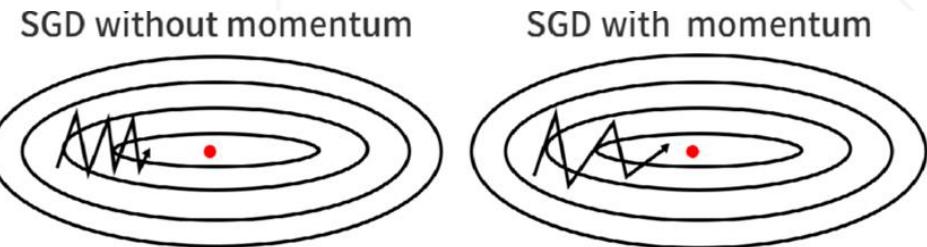
- 对应的样本子集 $D_B = \{x_{i_j}, y_{i_j}\}_{j=1}^B \subset D$ 称为批 (Mini Batch)。



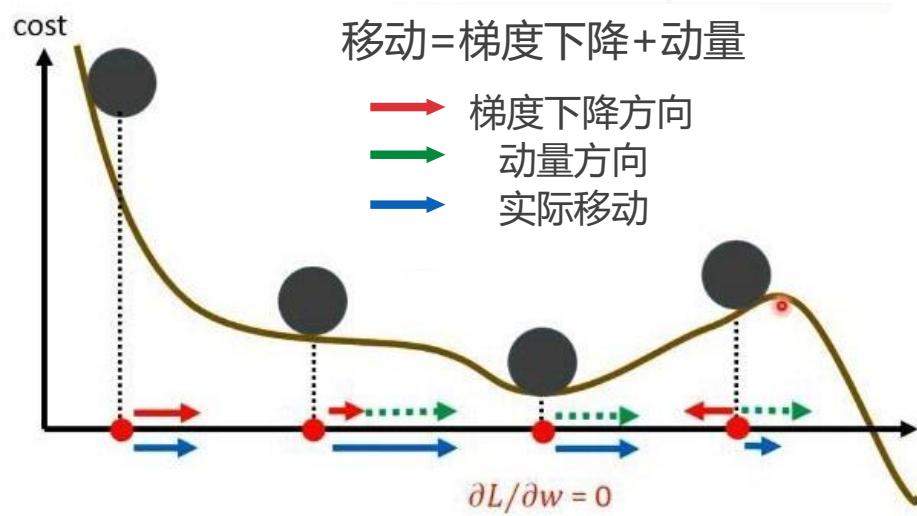
- 将前次的移动延续到本次

以往轮次的梯度的积累 $\Delta^{(0)} := 0$ 指数滑动平均 $\Delta^{(t)} := \alpha \Delta^{(t-1)} + \nabla_w L^{(t)}$ 按照积累值移动 $w^{(t)} := w^{(t-1)} - \eta \Delta^{(t)}$	当前轮次的梯度 
--	---

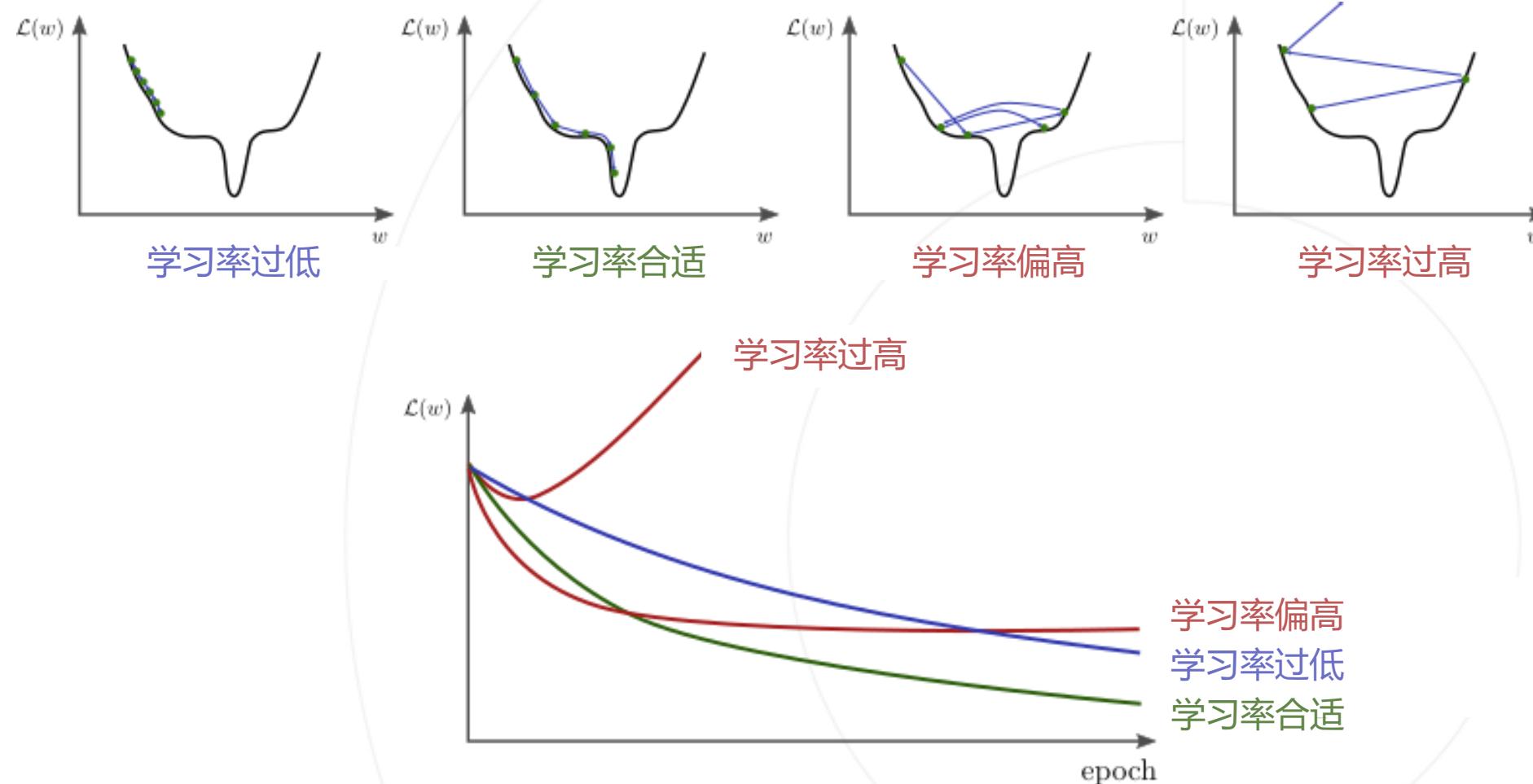
- 缓解随机梯度下降的波动



- 帮助逃离局部极小值和鞍点

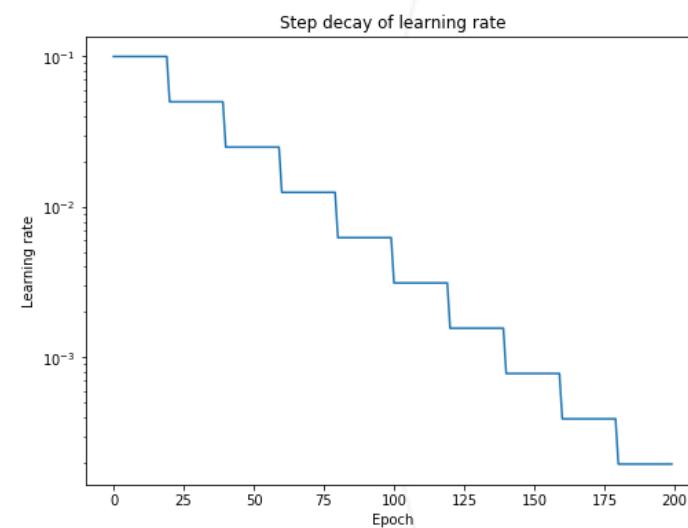


学习率对训练的影响

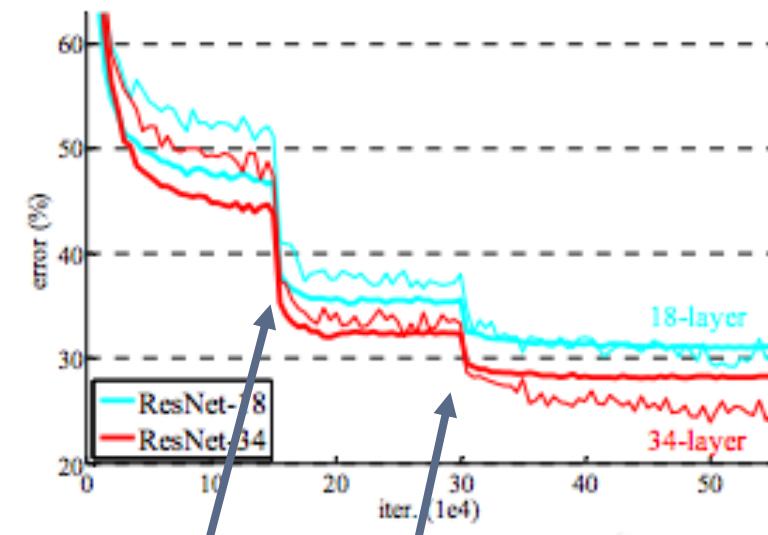


在训练初始阶段使用较大的学习率，损失函数稳定后下降学习率：

- 按步长下降
- 按比例下降 $\eta(t) = \eta_0 e^{-kt}$
- 按倒数下降 $\eta(t) = \frac{\eta_0}{\epsilon+t}$



每 20 轮降低学习率至 1/3



降低学习率后损失函数继续下降

针对卷积层和全连接层，初始化连接权重 W 和偏置 b

➤ 随机初始化

1. 朴素方法：依照均匀分布或高斯分布

$$W_{ij}, b_i \sim U(-a, a) \text{ or } W_{ij}, b_i \sim \mathcal{N}(0, \sigma^2) \quad a = \sqrt{3}\sigma \Rightarrow \text{方差相同}$$

2. Xavier 方法 (2010)：前传时维持激活值的方差，反传维持梯度的方差

$$a = \sqrt{6/(\text{fan}_{\text{in}} + \text{fan}_{\text{out}})} \text{ or } \sigma = \sqrt{2/(\text{fan}_{\text{in}} + \text{fan}_{\text{out}})}$$

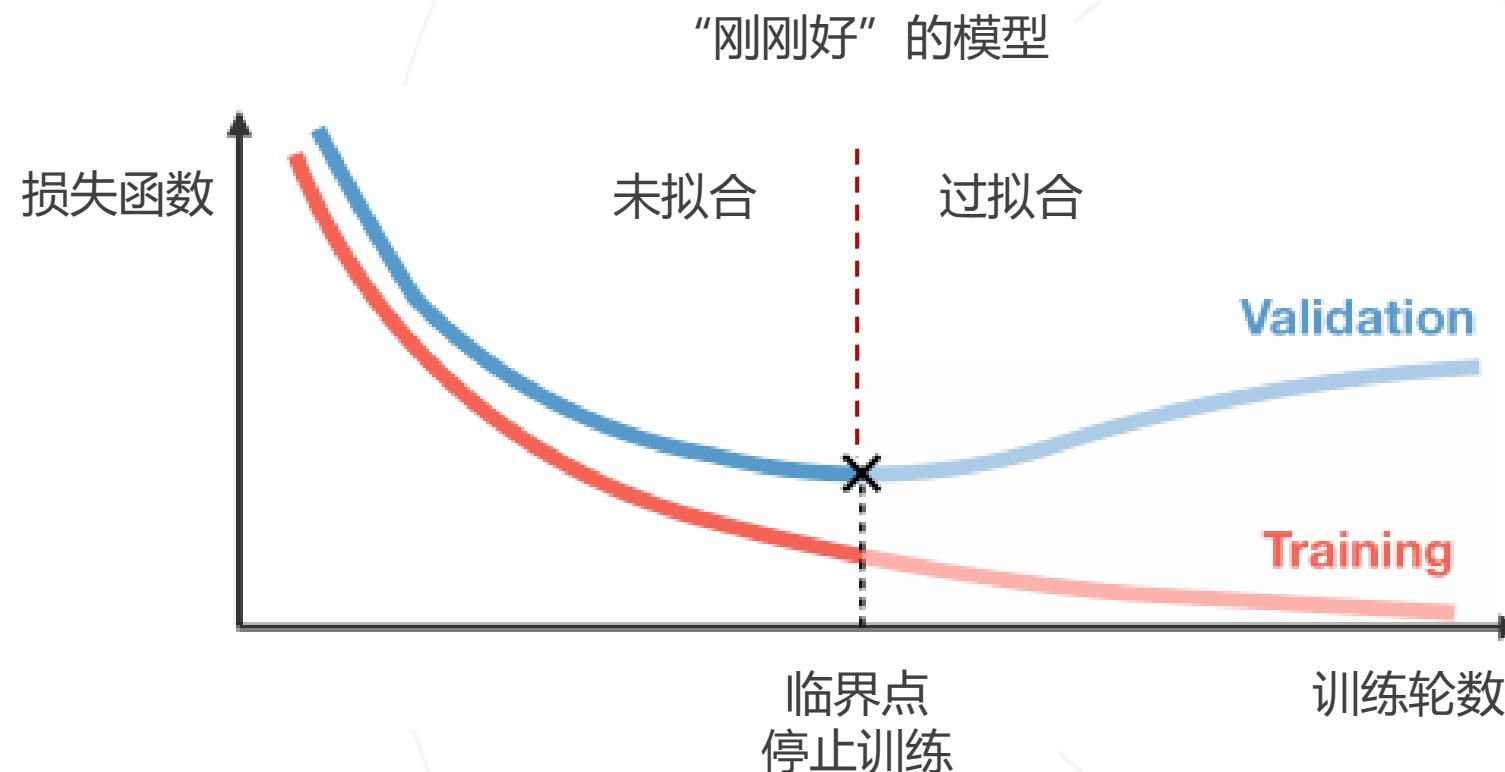
3. Kaiming 方法 (2015)：同上，但针对 ReLU 激活函数

$$a = \sqrt{6/n} \text{ or } \sigma = \sqrt{2/n} , \text{ here } n = \text{fan}_{\text{in}} \text{ or } \text{fan}_{\text{out}}$$

- 用训练好的模型（通常基于ImageNet数据集）进行权重初始化
 - 替换预训练模型的分类头，进行微调训练 (finetune)

将训练数据集划分为训练集和验证集，在训练集上训练，周期性在验证集上测试。

当验证集的 loss 不降反升时，停止训练，防止过拟合。



- 在损失函数中引入正则化项，以鼓励训练出相对简单的模型：

$$R(\Theta) = L(\Theta | \mathcal{D}) + \frac{1}{2} \lambda \|\Theta\|_2^2$$

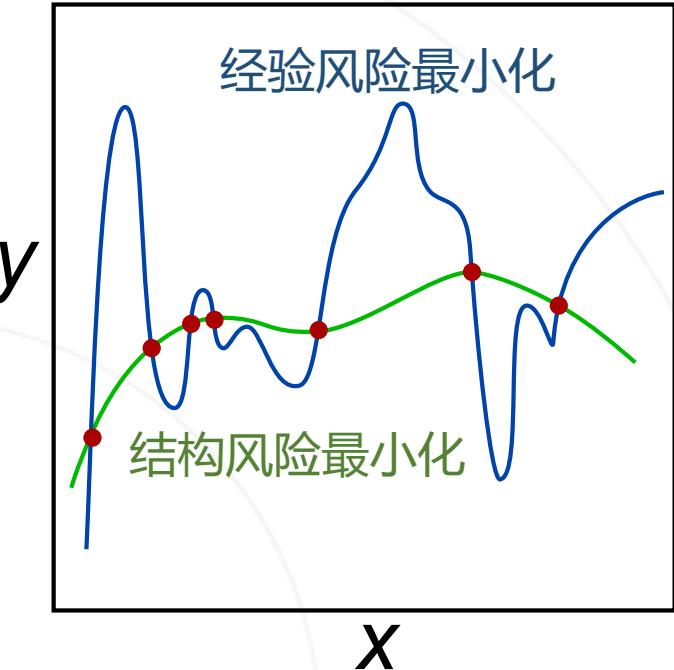
- 结构风险的梯度：

$$\nabla_{\Theta} R = \nabla_{\Theta} L + \lambda \Theta$$

- 梯度更新策略：

$$\begin{aligned}
 \Theta^{(t)} &= \Theta^{(t-1)} - \eta \nabla_{\Theta} R(\Theta^{(t-1)}) \\
 &= \Theta^{(t-1)} - \eta \nabla_{\Theta} L(\Theta^{(t-1)}) - \lambda \eta \Theta^{(t-1)} \\
 &= \boxed{(1 - \lambda \eta)} \Theta^{(t-1)} - \boxed{\eta \nabla_{\Theta} L(\Theta^{(t-1)})}
 \end{aligned}$$

权重衰减 常规梯度下降



- 泛化性好的模型 \leftarrow 大量多样化的数据
- 数据的采集标注是有成本的
- 数据增广**
- 利用简单的随机变换，从一张图片扩充出多张图片

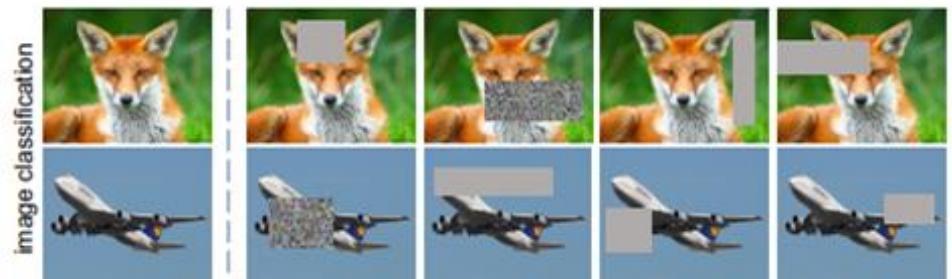
几何变换



色彩变换

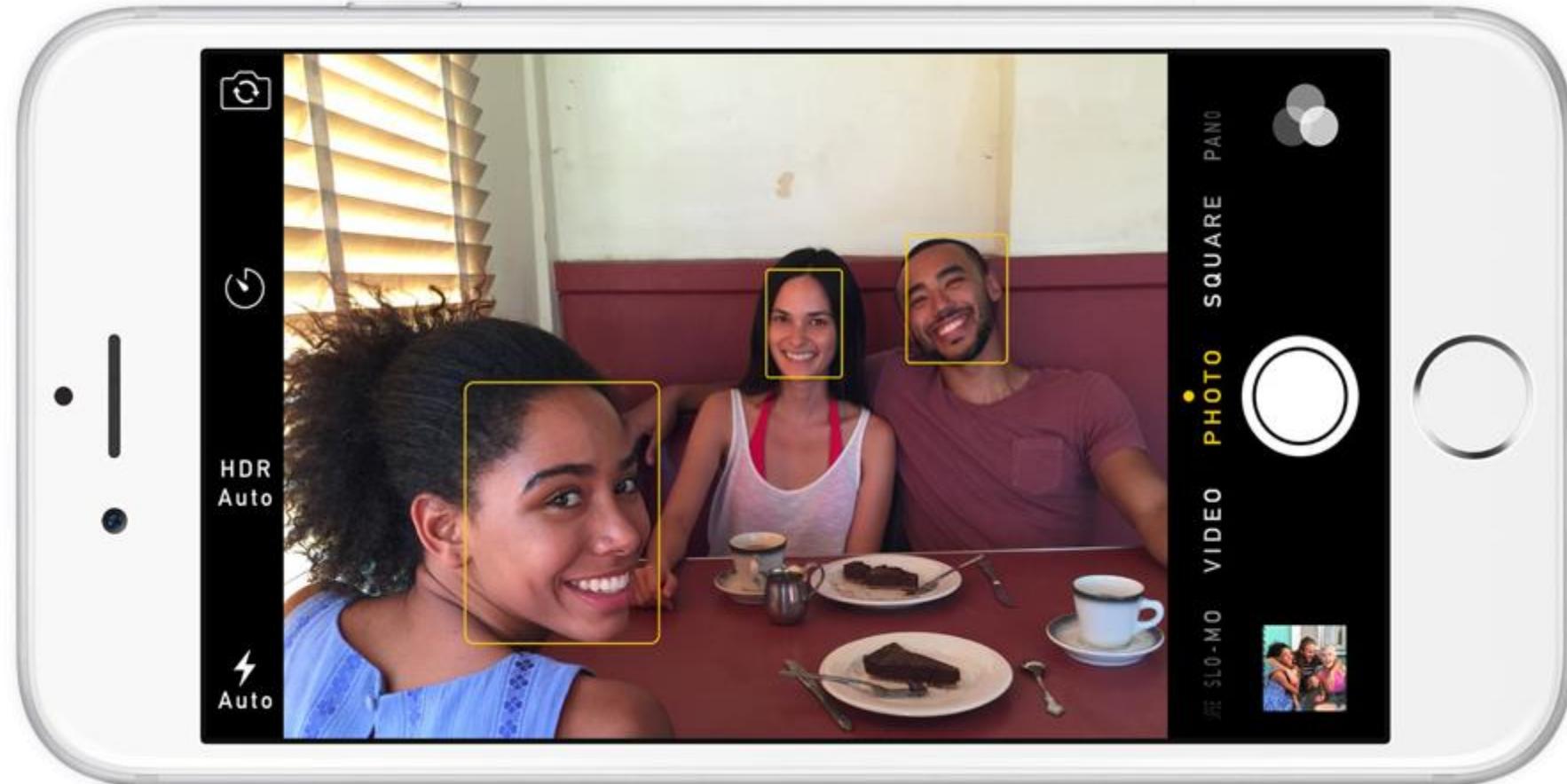


随机遮挡



- 随机梯度下降为主，各种经验策略辅助
- 损失函数高度不规则，非凸
 - 权重初始化：kaiming init，预训练模型
 - 优化器改进：动量 SGD
 - 学习率策略：学习率退火
- 防止过拟合
 - 数据增广、早停

目标检测的基本思路



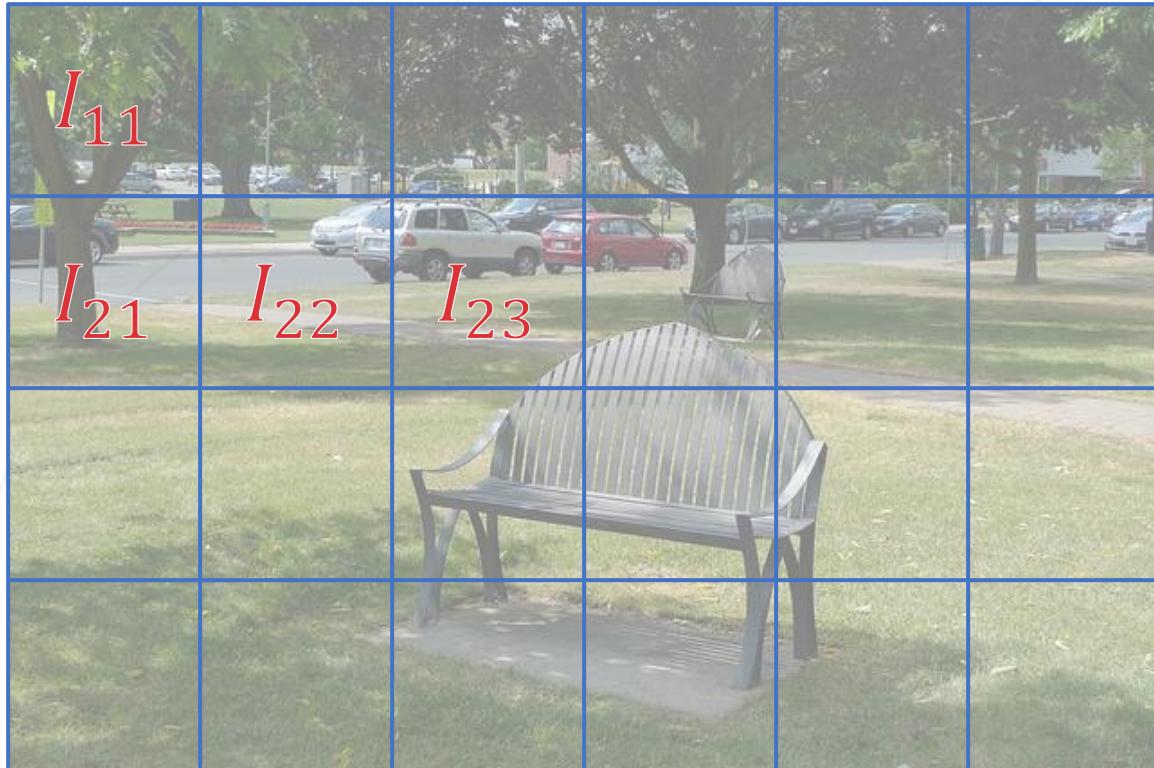


假设已经具备了图像分类算法 $C(I)$

1. 将图像切分成多块
2. 用分类算法 $C(I_{ij})$ 对每个图像块进行预测
3. 检测结果 = (分类结果, 图像块位置)

问题：

过于粗糙，无法检测分块边界上的物体



物体区域

类别: $C(I_{23}) = \text{car}$

位置: $L(I_{23}) = (3w, 2h)$

背景区域

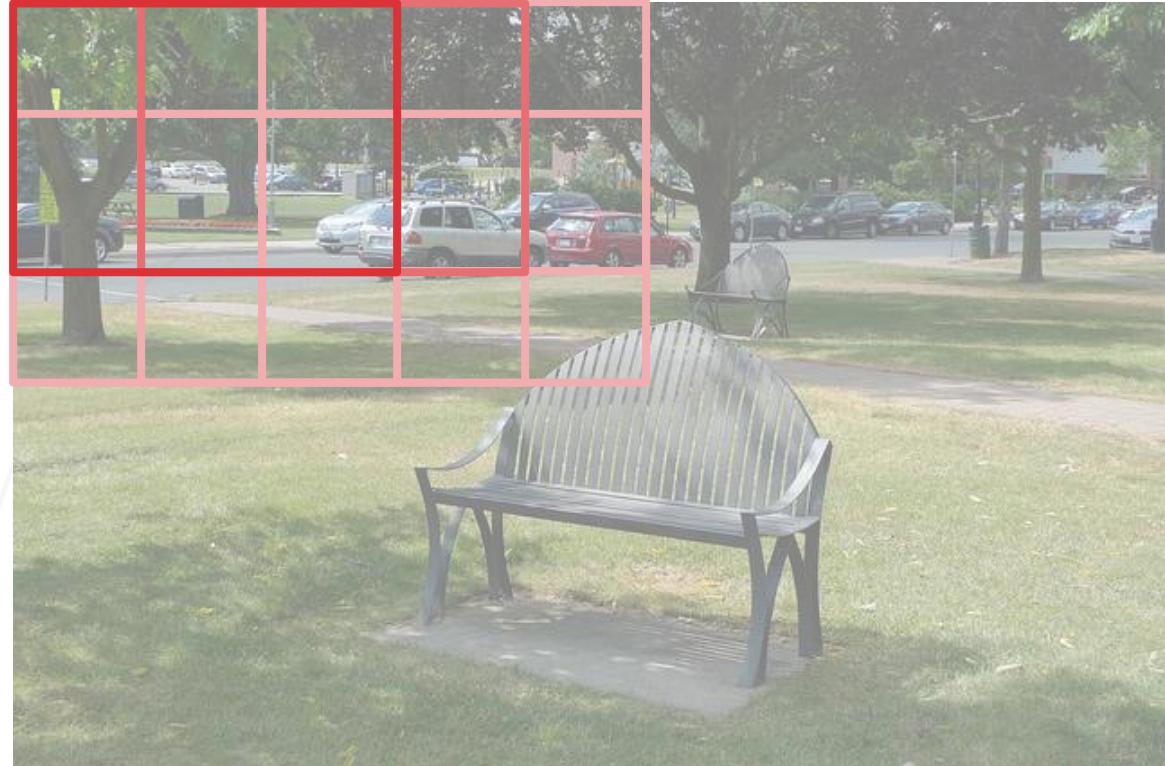
类别: $C(I_{22}) = \text{background}$

位置: 不关心

问题：图像分块过于粗糙，无法检测分块边界上的物体

改进：

1. 使用重叠的窗口，覆盖更多可能出现物体的位置
2. 用分类算法 $C(I)$ 检测每个图像块
3. 检测结果=(分类结果, 滑窗位置)



? 问题：滑窗边界与物体精确边界有偏差

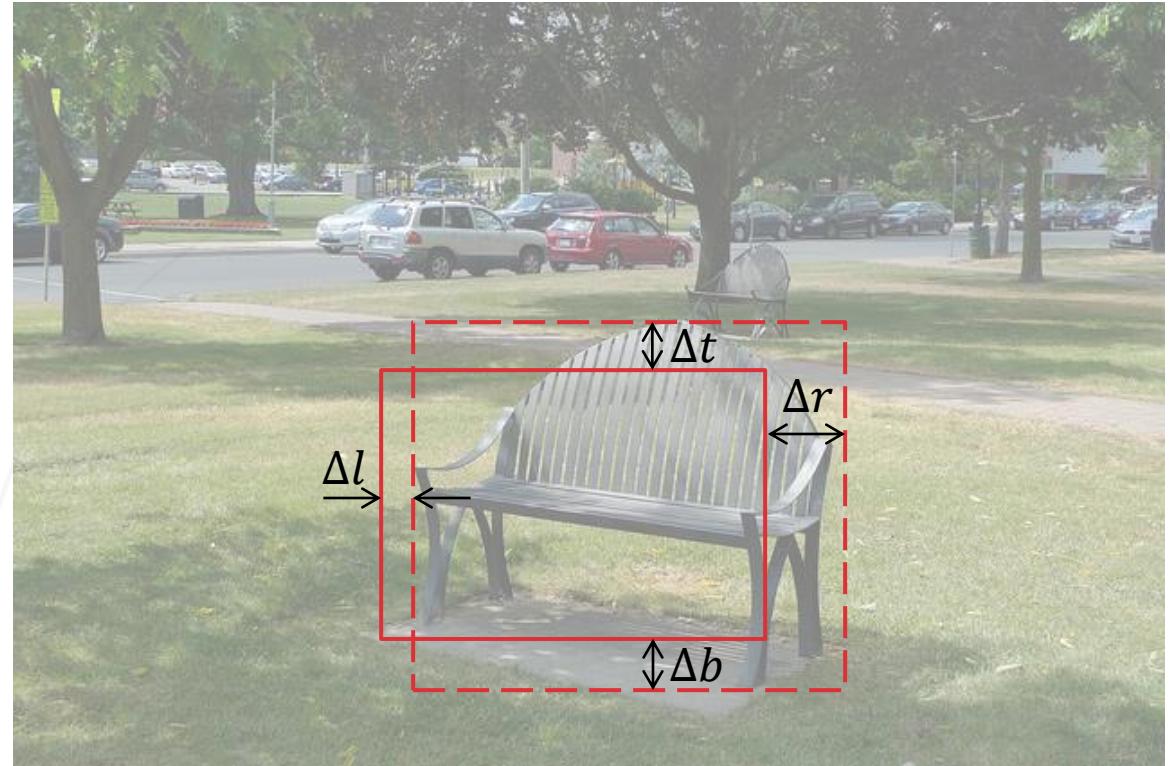
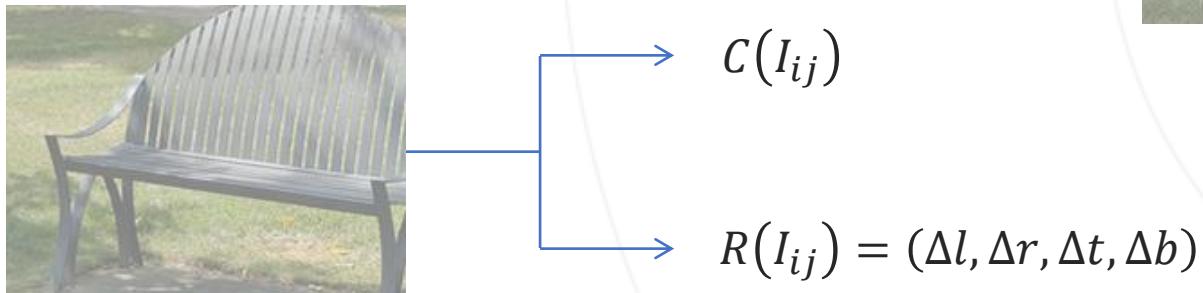
改进：分类的同时预测物体的精确位置
→ 边界框回归

分类模型 $C(I_{ij})$ 预测物体类别

回归模型 $R(I_{ij})$ 预测物体精确位置相对于窗的偏差

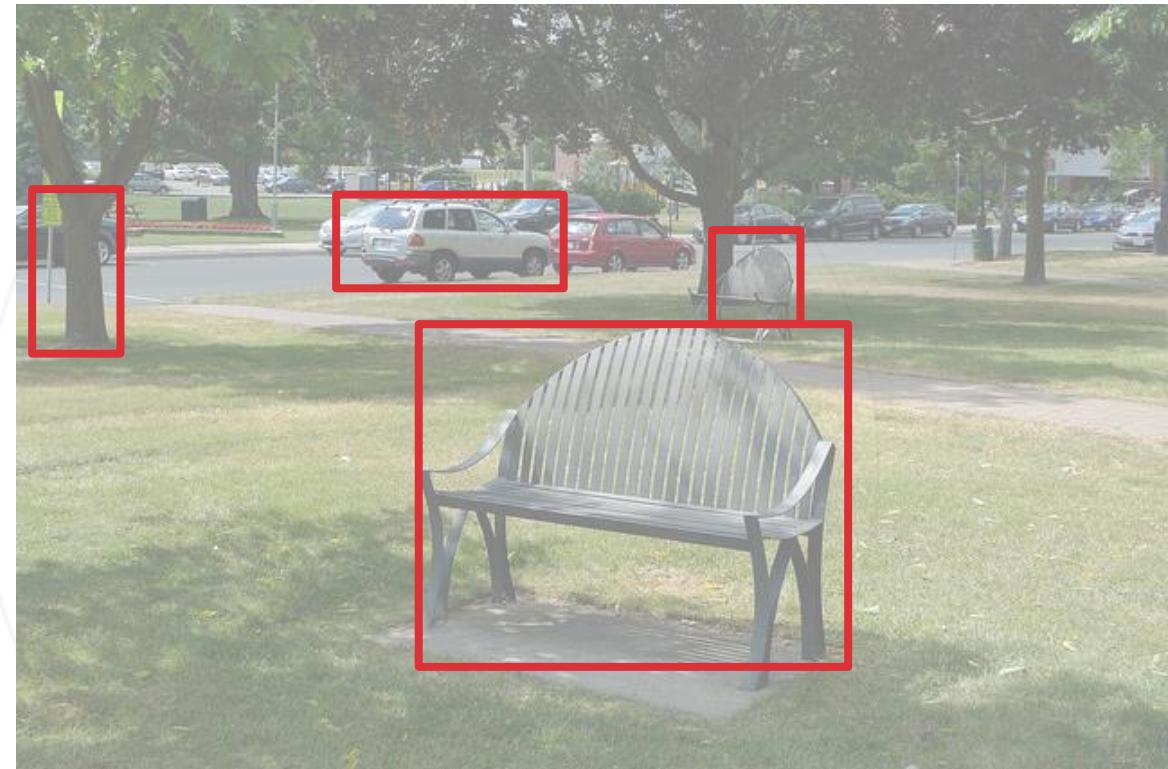
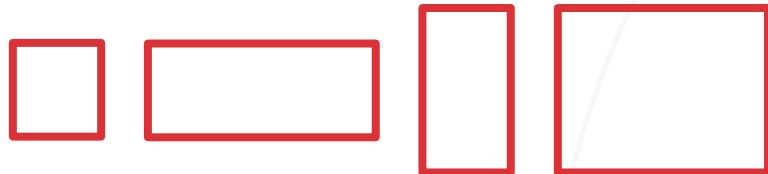
检测结果 = $(C(I_{ij}), \text{窗位置} + R(I_{ij}))$

多任务学习 (multi-task learning)



? 问题：不同物体大小不同，长宽比不同

❑ 改进：使用大小、长宽比不同的滑窗



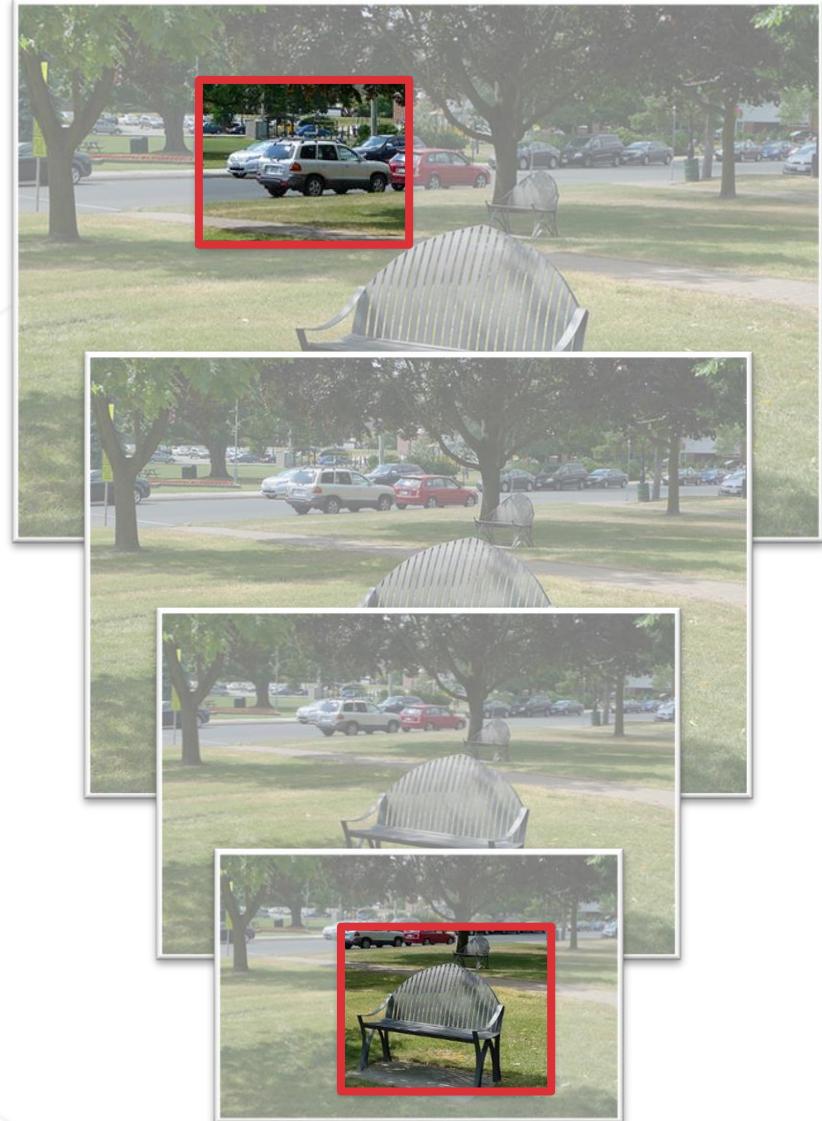
？ **问题：**不同物体大小不同，长宽比不同

❑ **解决方案：**将图像缩放到不同大小，构建图像金字塔 (Image Pyramid)。

相同大小的窗口在不同尺寸的图象上可以检测不同尺寸的物体。

图像逐级缩小

可检测的物体逐级放大



滑窗 = 空间上的密集预测

考虑 800x600 的图像，使用 80x60 的窗，步长 10 像素滑动

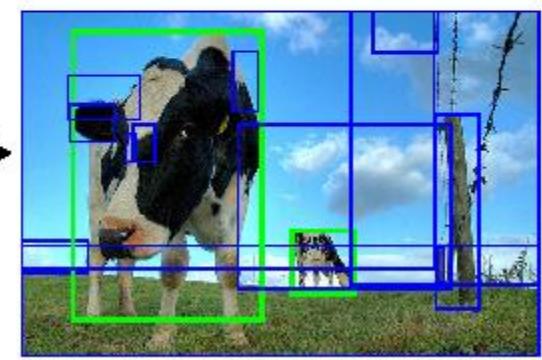
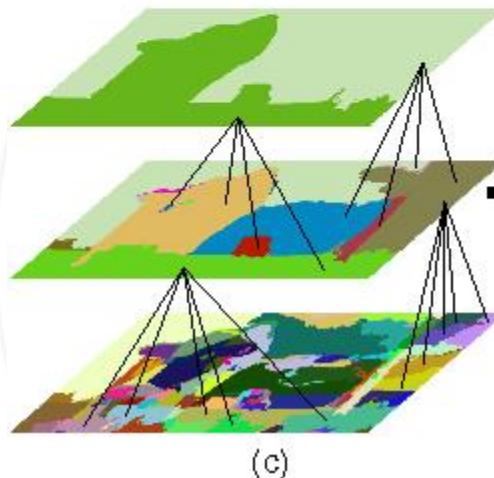
需要在 4800 个窗上进行分类预测

- × 每个位置5个尺度的窗
- × 每个尺度3个长宽比
- ≈ 检测一张图像需要完成数万次图像分类预测

难以满足实时检测的需求

分析：

大量窗口都落在不包含物体的边界区域。可以先通过简单快速的方法找出可能含物体的区域。



Selective Search 算法：

使用贪心算法，将空间相邻且特征相似的图像块逐步合并到一起，形成可能包含物体的区域，称为提议区域或提议框。

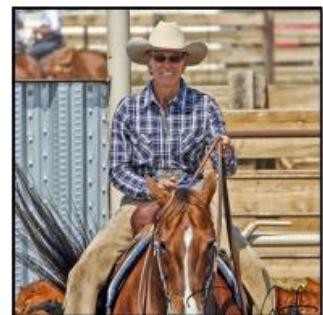
两阶段目标检测算法



输入图片

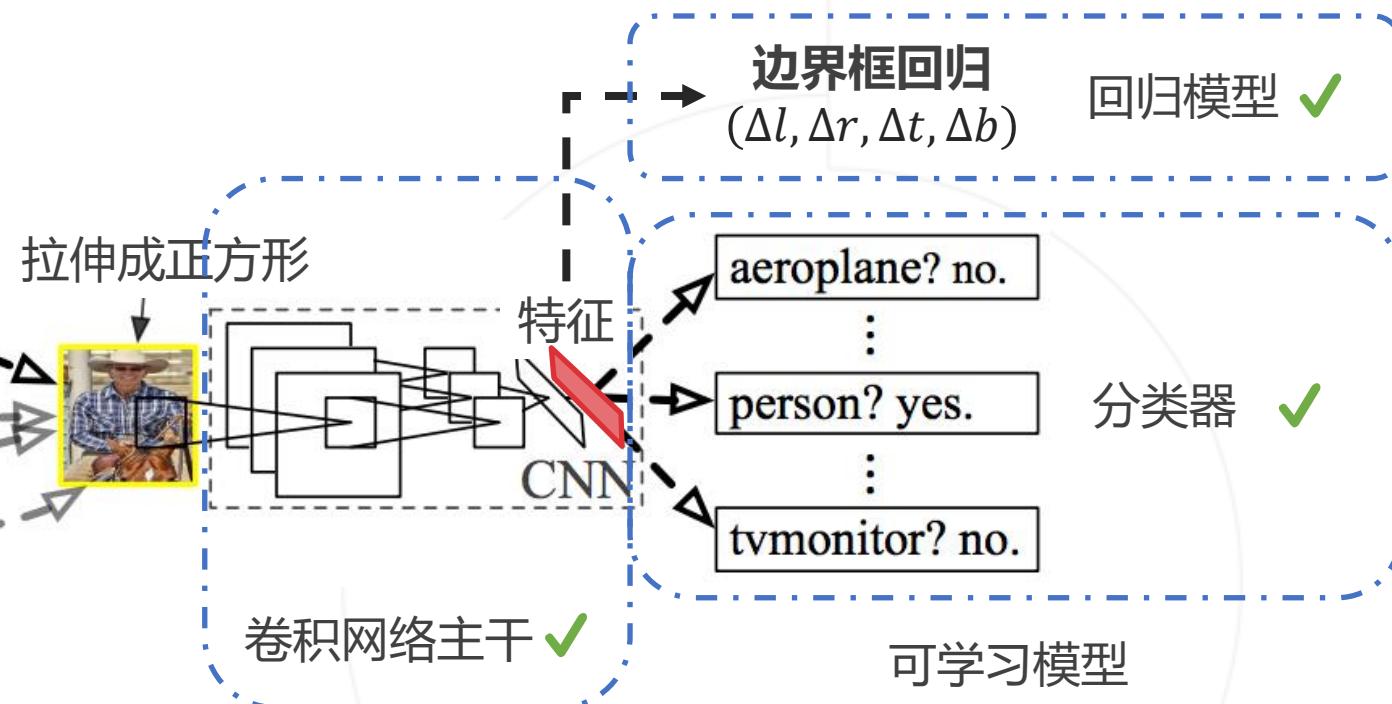
R-CNN 算法包含两个步骤，因此也称为两阶段方法

模型中的哪些模块需要学习？



区域提议 ×

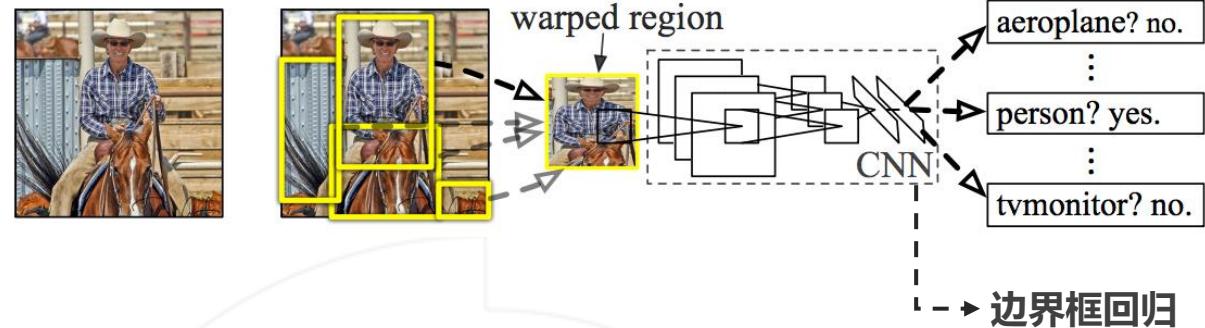
固定算法
无可学习参数



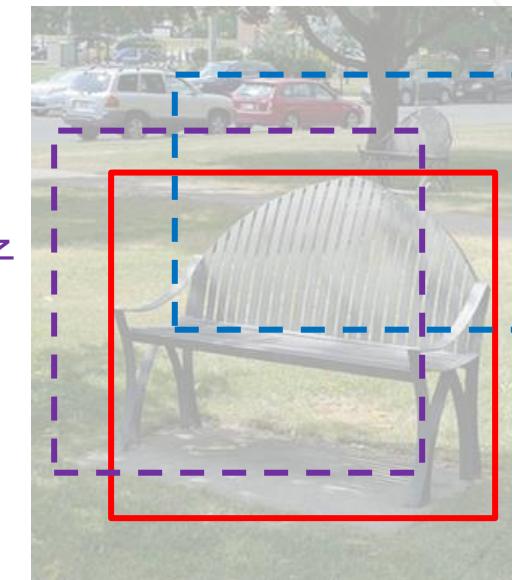
如何生成分类和回归的目标值？

对于提议框 P 和标注框 B

- 如果二者重叠较大
分类目标值 := B 的类别标注
回归目标值 := 编码后的偏差值
- 如果二者重叠较小
分类目标值 := 背景
回归分支不计算 LOSS



提议框 P1
重叠较大
类别:=椅子



提议框 P2
重叠较小
类别:=背景

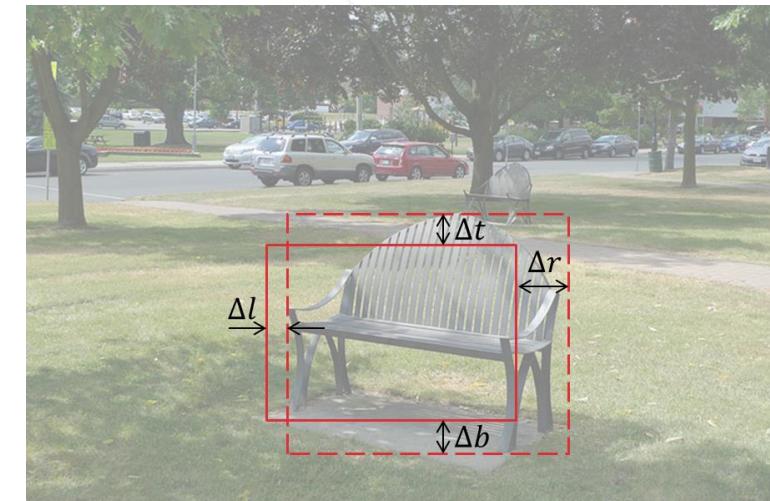
标注框 B
类别=椅子

为使回归模型更易于学习，通常使用如下编码策略对边界框偏移量进行编码：

对于提议框 $P = (p_x, p_y, p_h, p_w)$ 和真值框 $B = (b_x, b_y, b_h, b_w)$

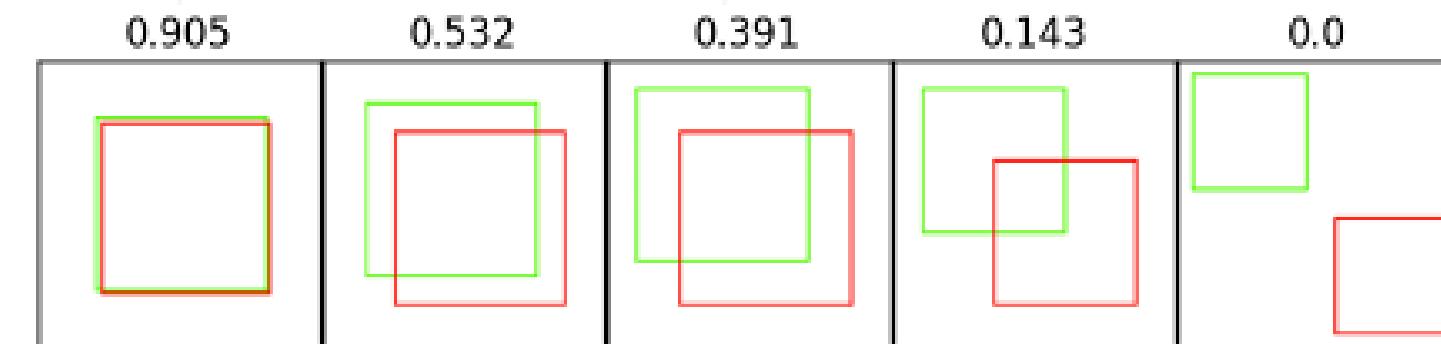
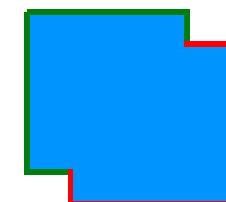
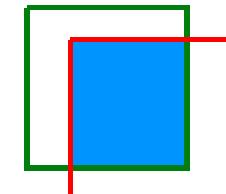
边界框的偏移量，即回归模型的预测目标为：

$$(t_x, t_y, t_h, t_w) = \left(\underbrace{\frac{b_x - p_x}{p_w}, \frac{b_y - p_y}{p_h}}_{\text{位置偏差基于边长归一化}}, \underbrace{\log\left(\frac{b_w}{p_w}\right), \log\left(\frac{b_h}{p_h}\right)}_{\text{尺度比例的对数}} \right)$$



交并比 IoU 定义为两个矩形框交集面积与并集面积的比值，是对两个矩形框重合度的衡量。

$$IOU = \frac{\text{area of overlap}}{\text{area of union}}$$



图片来源：

http://ronny.rest/tutorials/module/localization_001/iou/

<https://supervise.ly/explore/plugins/mean-average-precision-m-ap-75277/overview>

检测算法有时会针对单个物体给出多个相近的检测框。

在所有重叠框中，只需要保留置信度最高的。

非极大值抑制 (NMS) 算法：

输入：检测器产生的一系列检测框 $P = \{P_1, \dots, P_n\}$ 及对应的置信度

$s = \{s_1, \dots, s_n\}$, IoU 阈值 t

步骤：

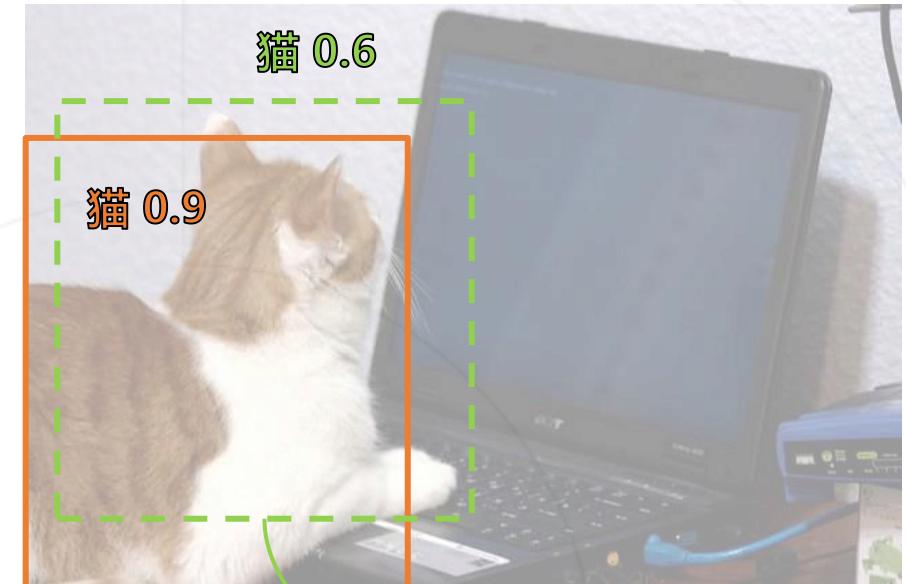
1. 初始化结果集 $R = \emptyset$

2. 重复直至 P 为空集

① 找出 P 中置信度最大的框 P_i 并加入 R

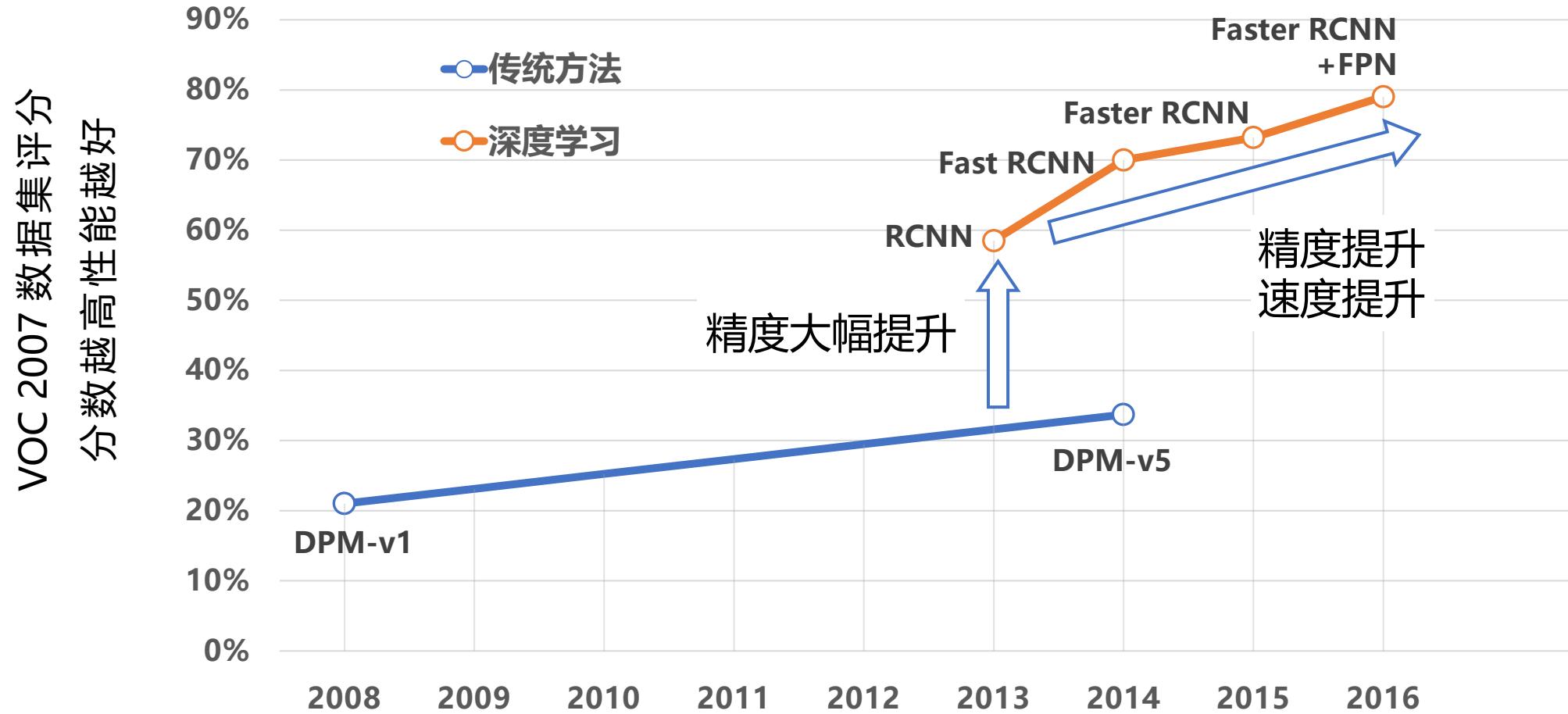
② 从 P 中删除 P_i 以及与 P_i 交并比大于 t 的框

输出：结果集 R



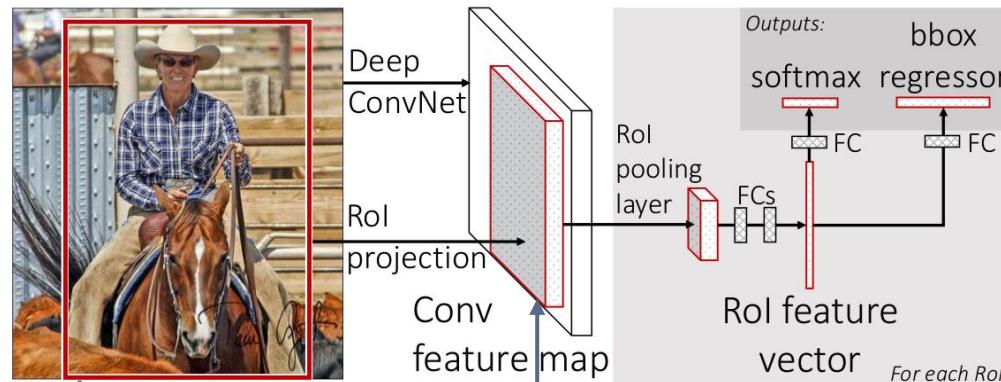
删除绿色框，因为它和置信度更高的橙色框重叠。

- 卷积网络使用 AlexNet
- 使用 Selective Search 产生提议框
- 非端到端训练，先训练卷积网络的主干部分，再基于卷积网络的特征训练 SVM 分类器
- 非多任务学习，先训练分类模型，再边界框回归单独训练
- 根据配置不同，单图推理几秒~几十秒



两阶段方法的演进 (2014 - 2017)

Fast RCNN 2014



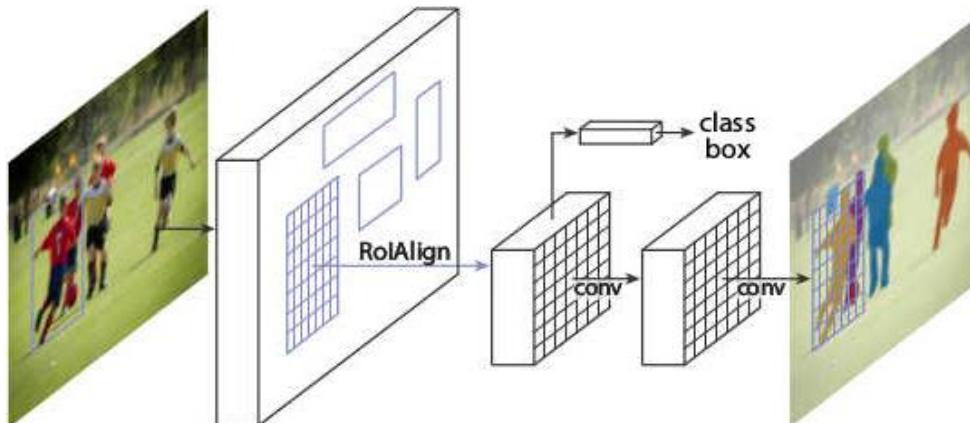
提出了 ROI Pooling 方法，把区域从图像移动到特征图上，大幅降低了计算量。

Faster RCNN 2015



提出了 RPN 网络，用于替换传统方法，产生区域提议，进一步提高效率。

Mask RCNN 2017

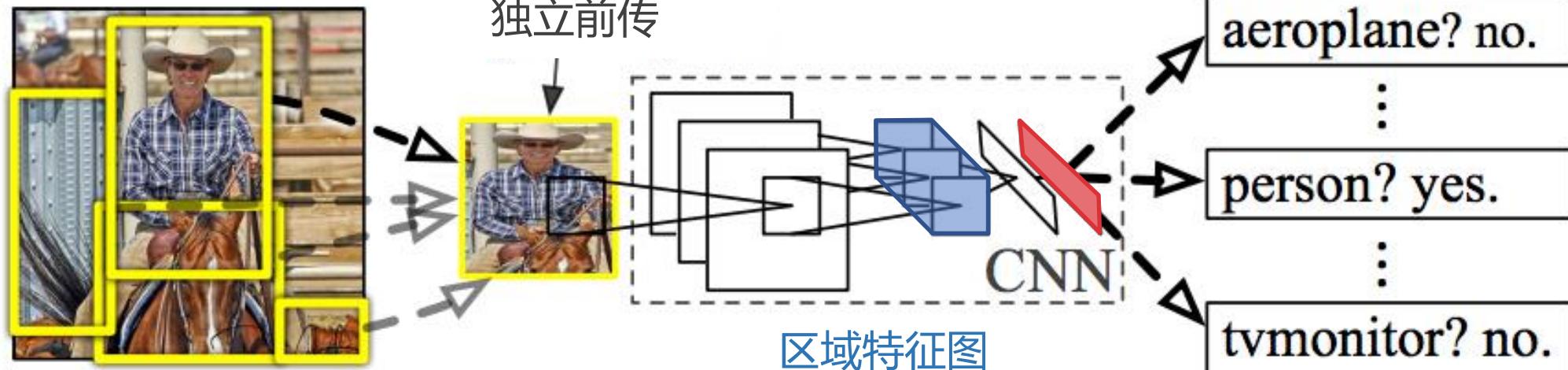


提出了 ROI Align 算法升级替换 ROI Pooling。
加入用于实例分割的分支。

问题： 数千个提议框

数千次全图CNN前传

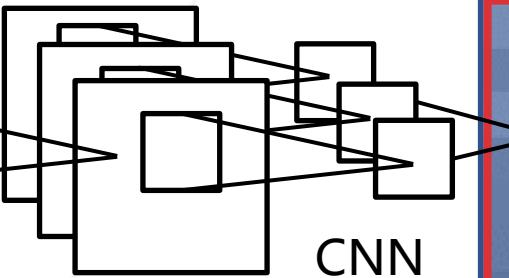
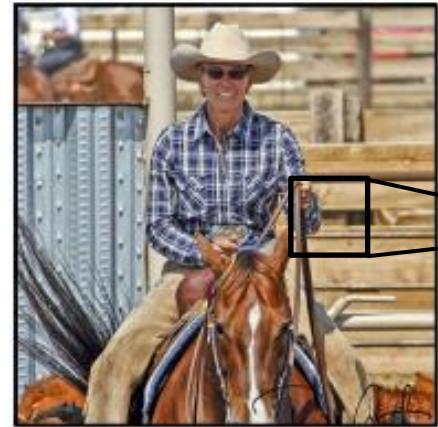
慢！



分析： 提议框大量重叠

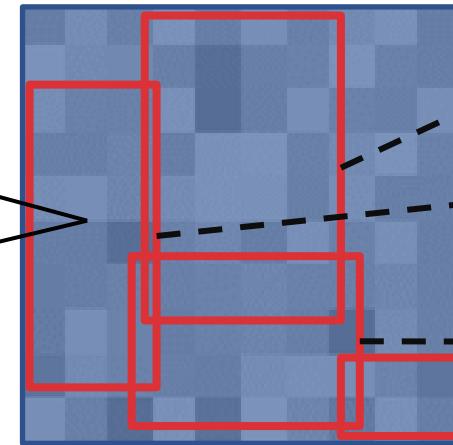
重叠区域大量重复卷积计算

改进思路：整张图片输入主干网络，共享卷积计算，再在特征图上完成裁框



全图输入卷积网络

全图特征图



单次前传

类别预测 + 边界框回归
类别预测 + 边界框回归
类别预测 + 边界框回归

在特征图上完成裁框

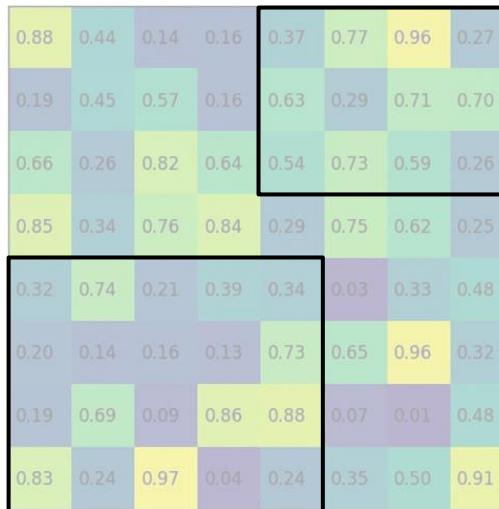
基于区域特征图的**轻量**预测

不同框重叠的部分共享计算

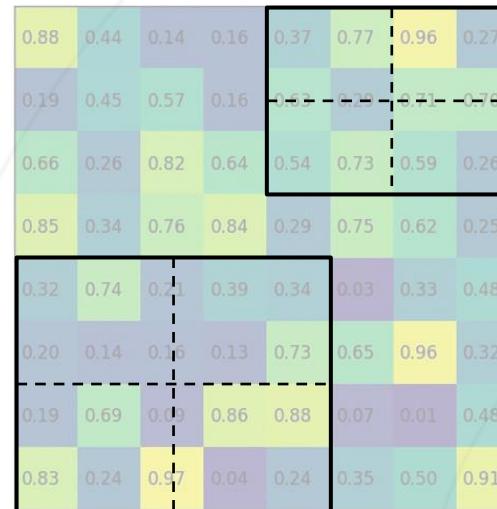
问题 1：如何将提议框映射到特征图上？→ 按照几何比例缩放

问题 2：如何将提议框内不定尺寸的特征图变成固定尺寸？→ RoI Pooling

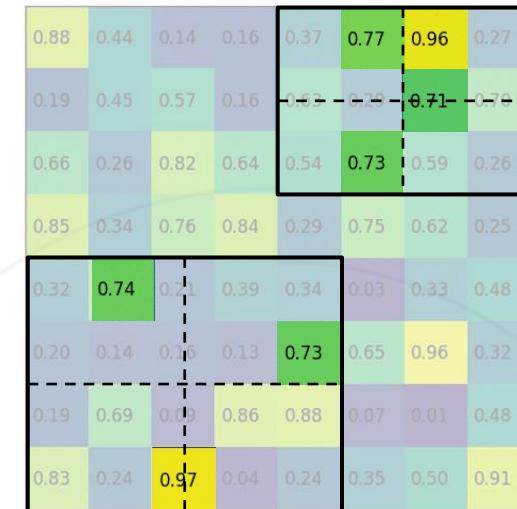
全图特征图与提议框



切分提议区域



各区域最大值



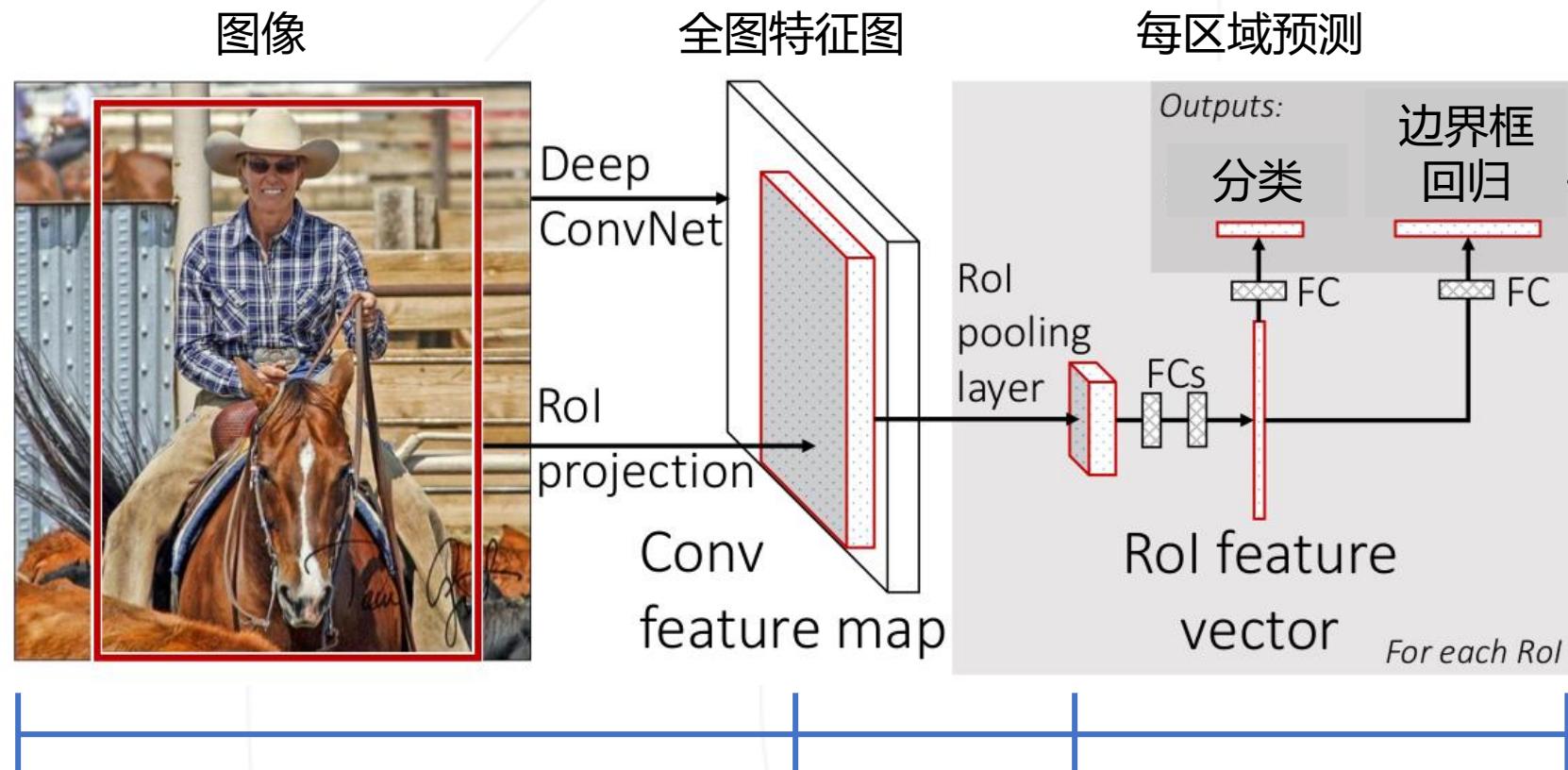
池化结果

0.77	0.96
0.73	0.71
0.74	0.73
0.97	0.97

做法：将提议区域切分成固定数目的格子（上图中 2×2 ，实际常用 7×7 ），对于每个格子：

- 如果格子边界不在整数坐标，则膨胀至整数坐标
- 通过 Max Pooling 得到格子的输出特征

作用：将任意尺寸的提议区域映射至固定尺寸的特征图，同时保留图像特征



1. 将原图上的提议框按照几何比例缩放到特征图上

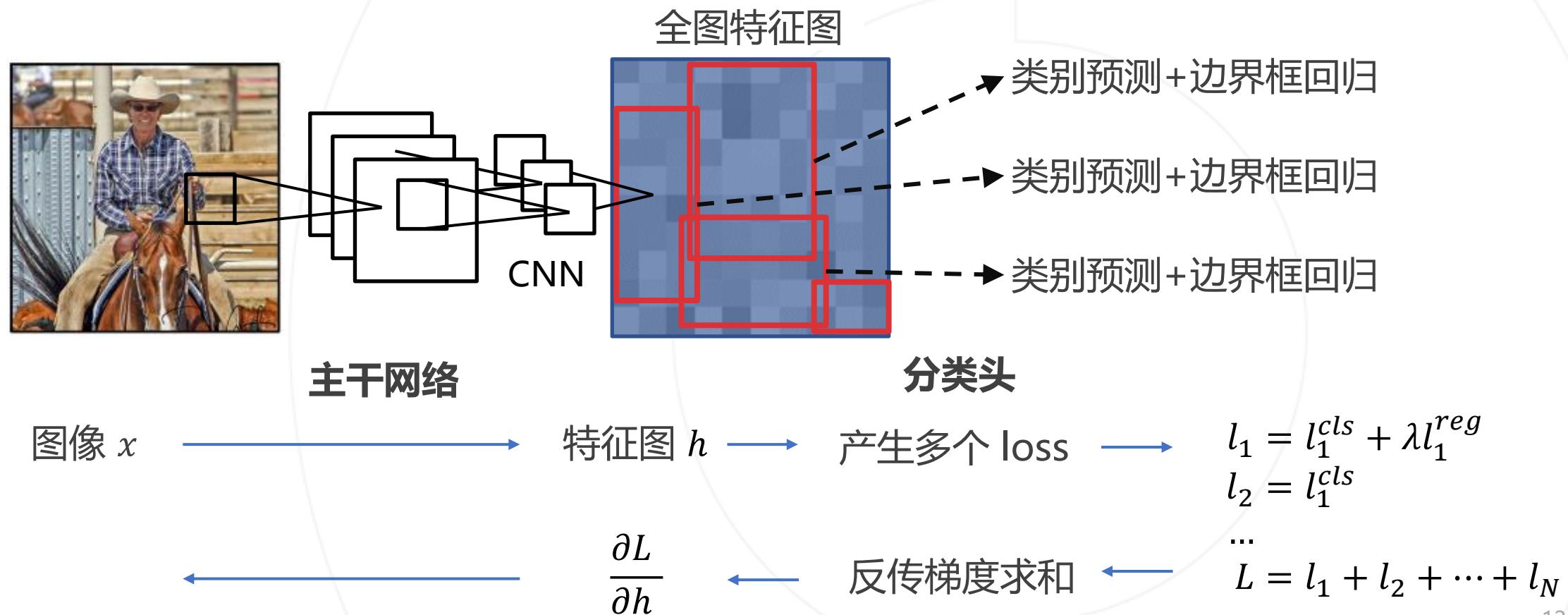
2. 利用 RoI Pooling，将不同尺寸的特征图变换到相同大小

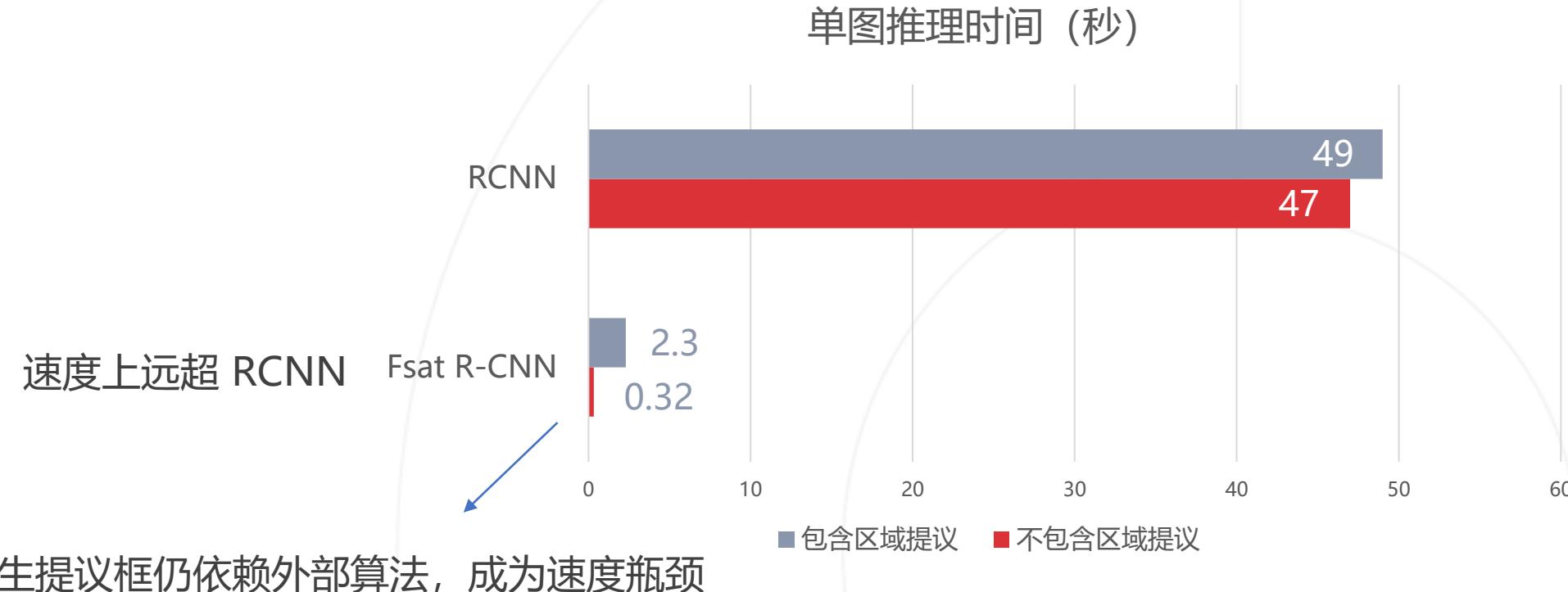
3. 基于相同尺寸的特征图进行分类和回归

$\text{RoI} = \text{Region of Interest}$
提议框内的图像区域

与 RCNN 相同，比较真值框与提议框的 IoU，确定提议框的分类目标和回归目标

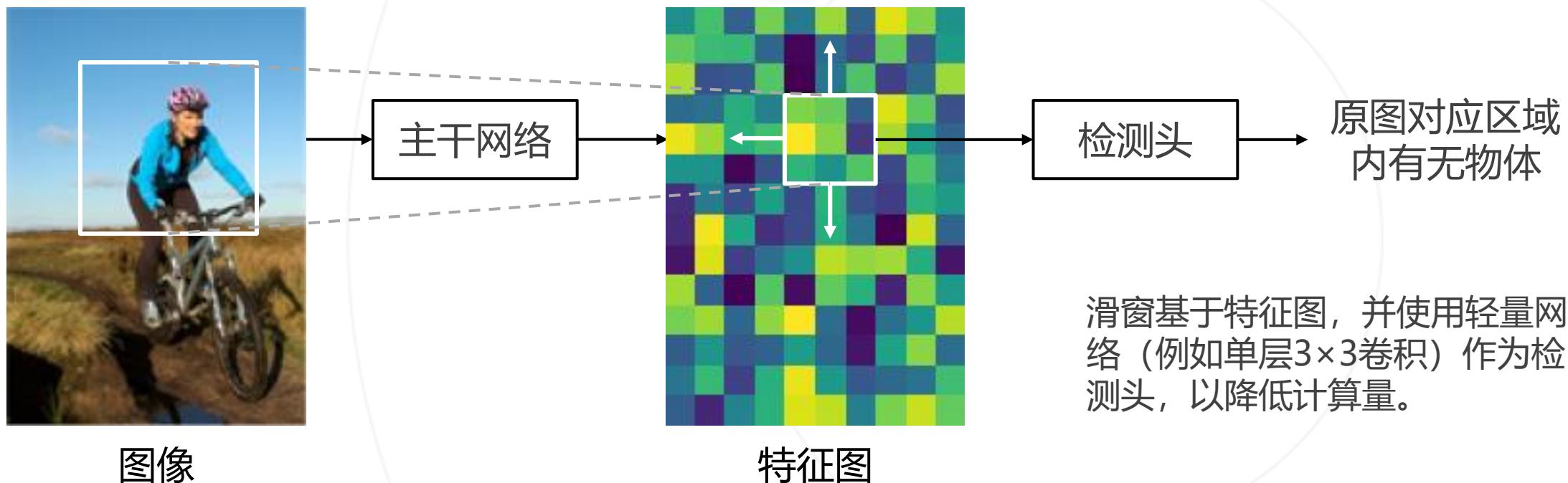
通过 RoI Pooling 的前传和反传实现端到端训练





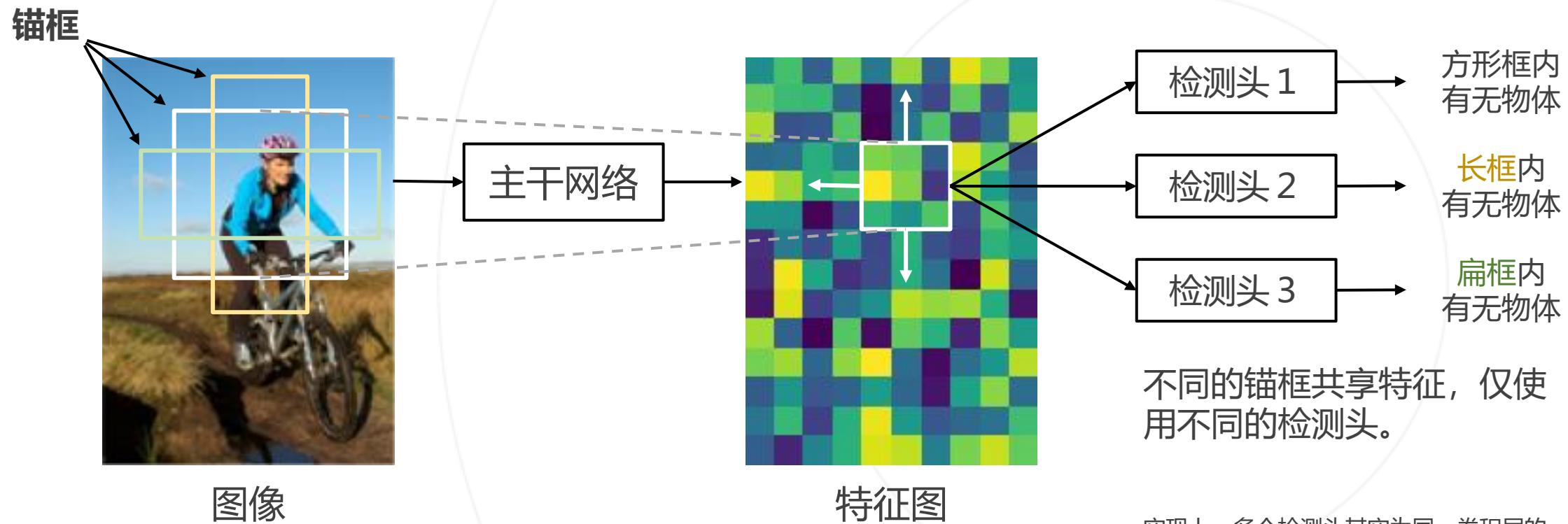
解决方案：使用卷积网络产生提议框，并与检测器共享主干网络结构

- 区域提议 = 在图中找到有物体的框 → 单类别的物体检测问题
- 采用滑窗的思路，进行空间密集预测



如何处理不同大小的物体？

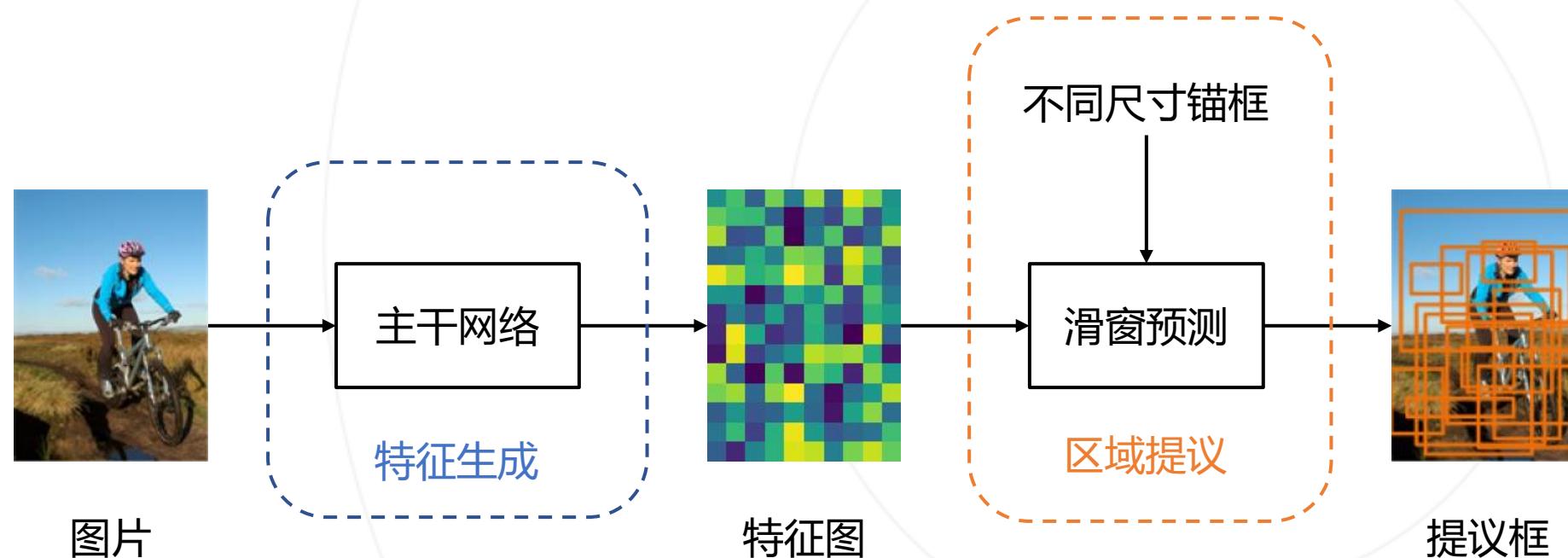
在原图上设置不同大小的假想框，用不同的检测头检测对应框中是否出现物体，称为锚框。

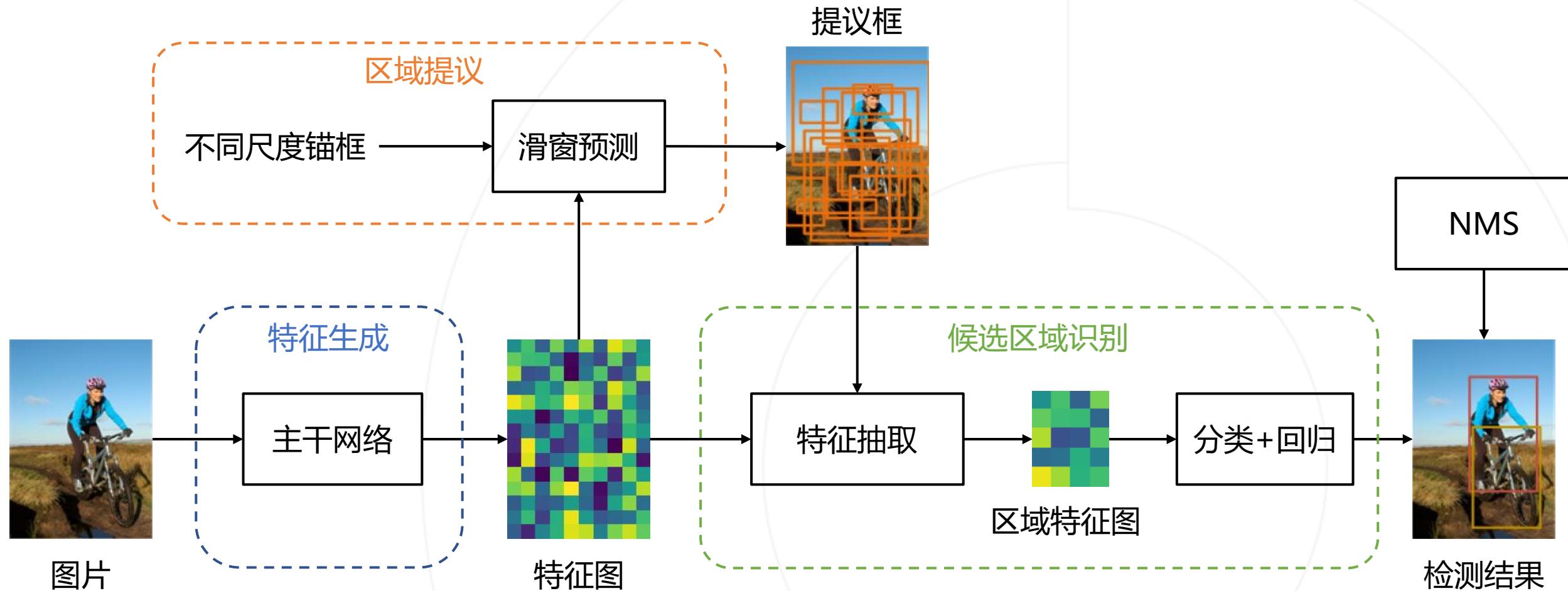


实现上，多个检测头其实为同一卷积层的不同通道。

推理：

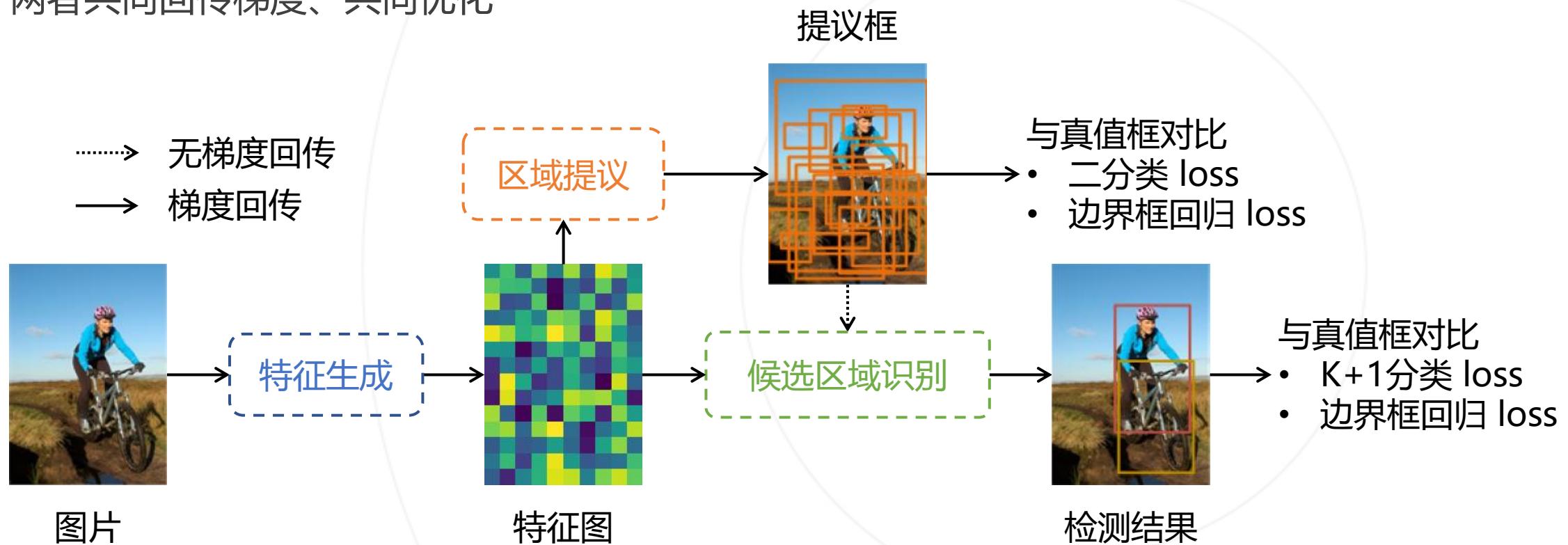
1. 主干网络产生特征图
2. 不同的检测头产生区域提议（分类+边界框回归）
3. 集合所有提议框，做 NMS





RPN 与 Fast RCNN 共享主干网络

- Faster RCNN = RPN + Fast RCNN
- 每部分产生分类和回归 loss
- 两者共同回传梯度、共同优化



到此为止，模型基于单级特征图进行预测，通常是主干网络最后一层或倒数第二层。

问题：

高层次特征空间降采样率较大 → 小物体信息丢失

解决思路：

- 基于低一层特征图预测 → 低层特征语义信息薄弱
- 图像金字塔 → 低效率



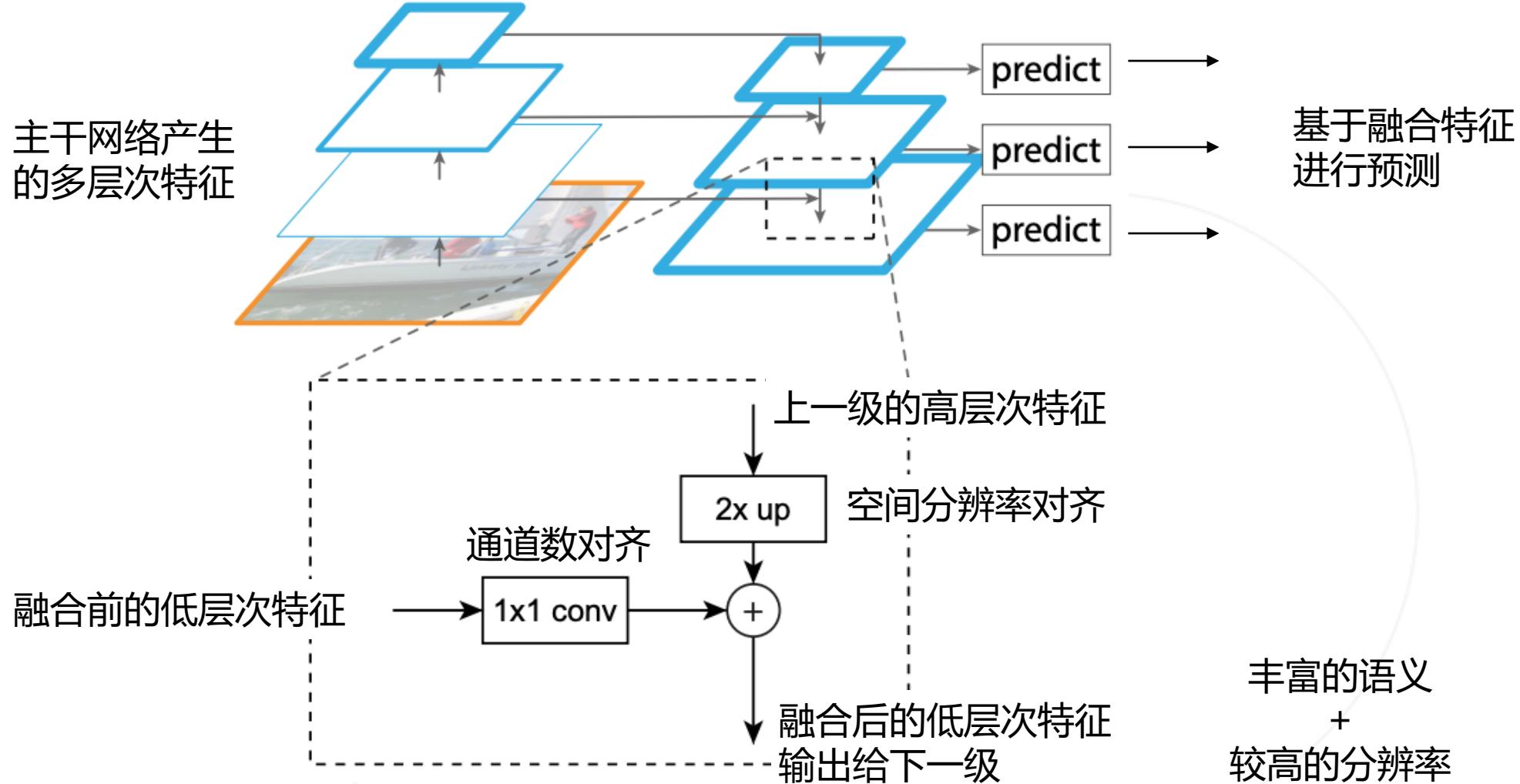
融合高+低层次的特征



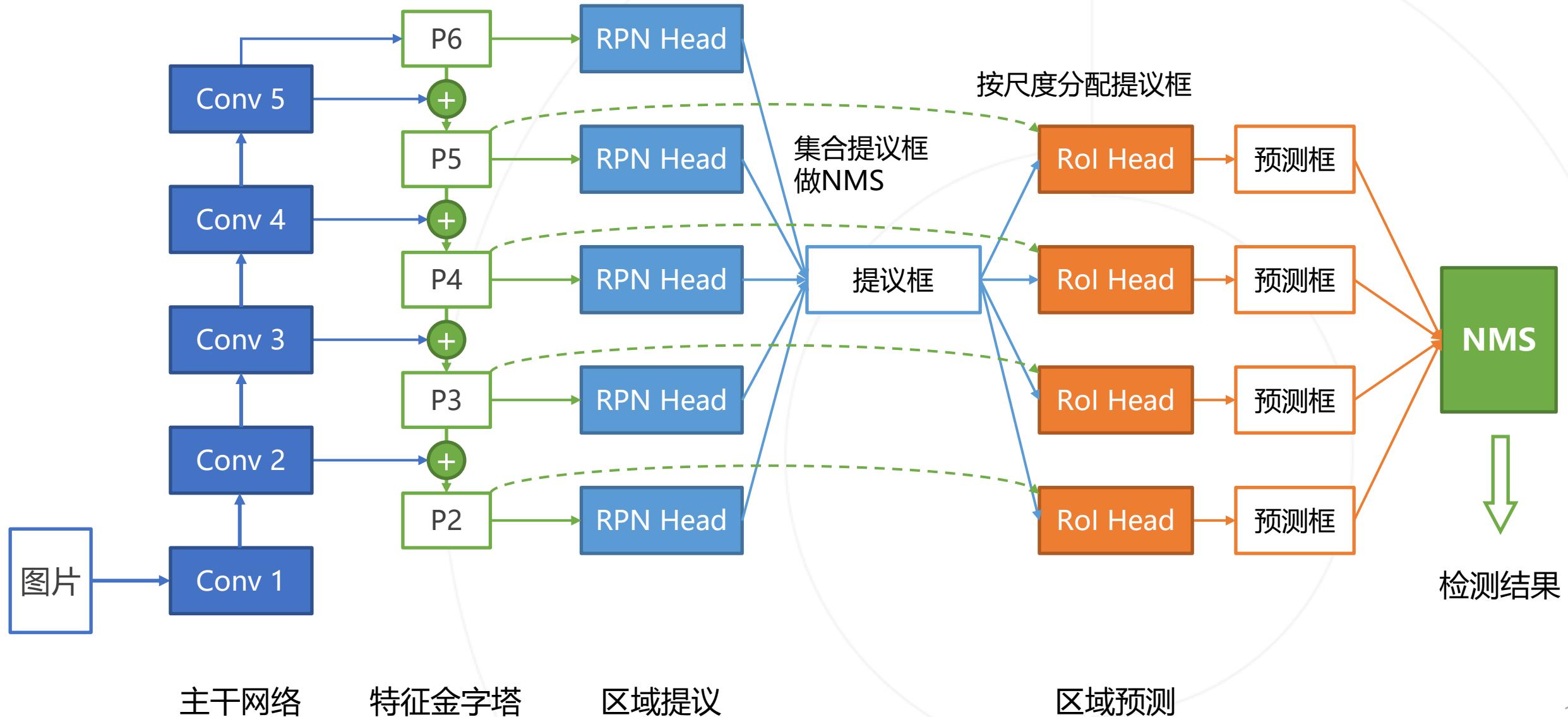
特征金字塔网络 FPN

特征金字塔网络 Feature Pyramid Network

OpenMMLab

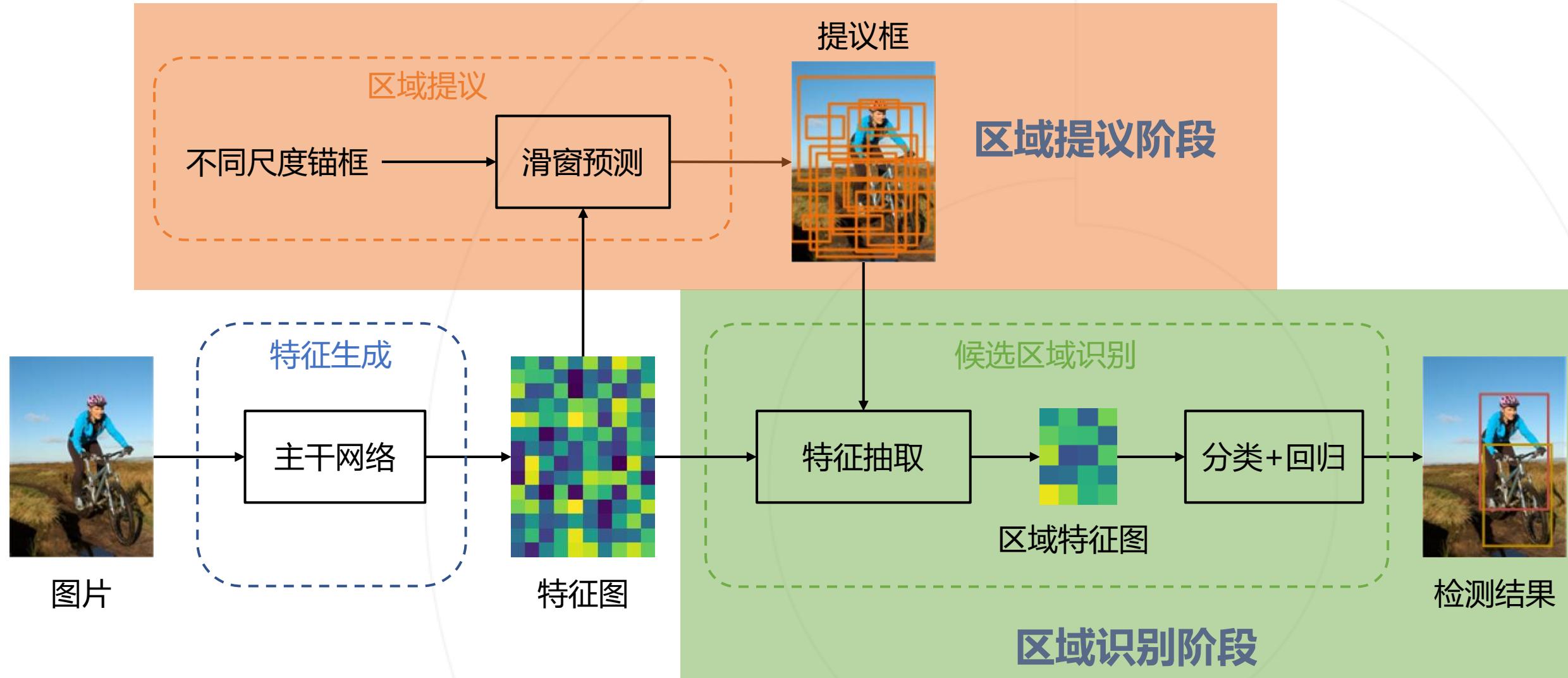


在 Faster RCNN 模型中使用 FPN



单阶段目标检测算法

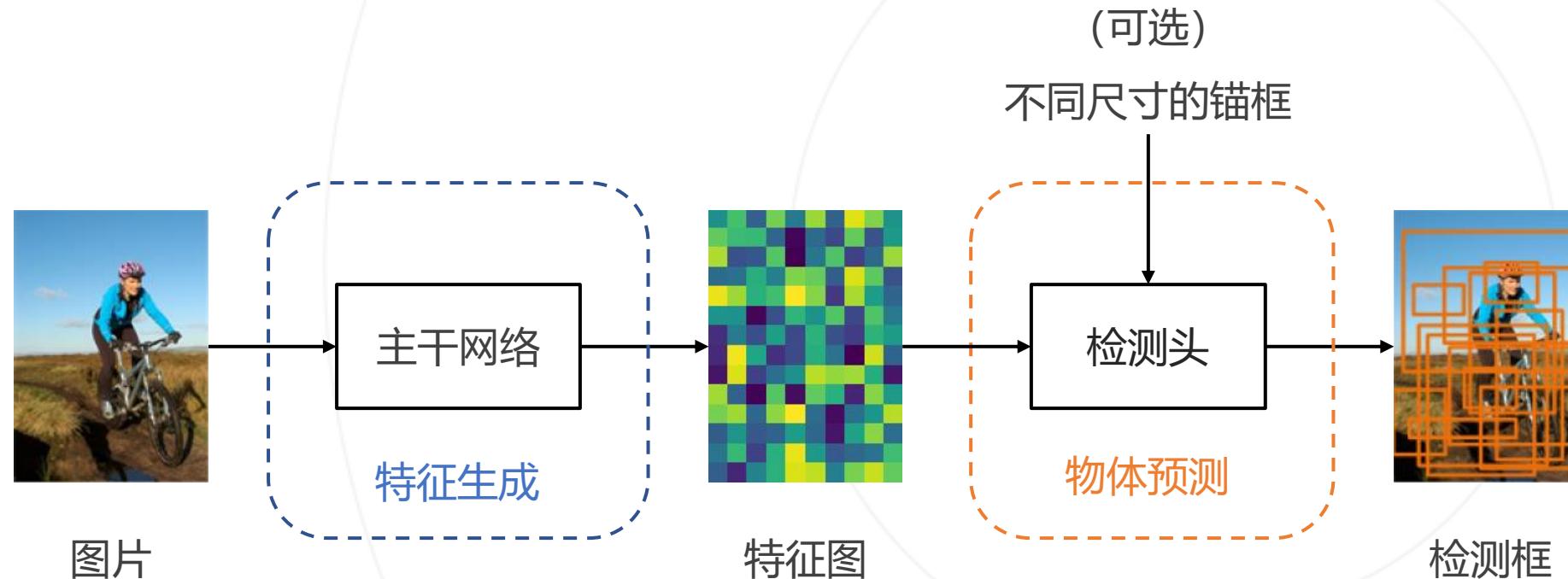
Single-staged Methods

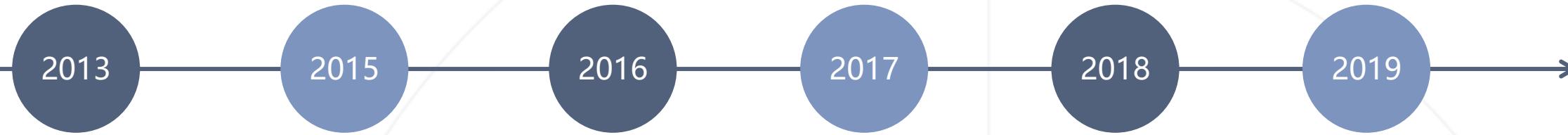


特点：没有区域提议阶段，直接基于特征图产生类别和边界框预测

优点：推理速度快，结构简单，易于在不同设备上部署

缺点：多数情况下性能不如两阶段算法





两阶段算法

单阶段算法

无锚框算法

基于特征图直接回归边界框



基于多尺度特征图与锚框

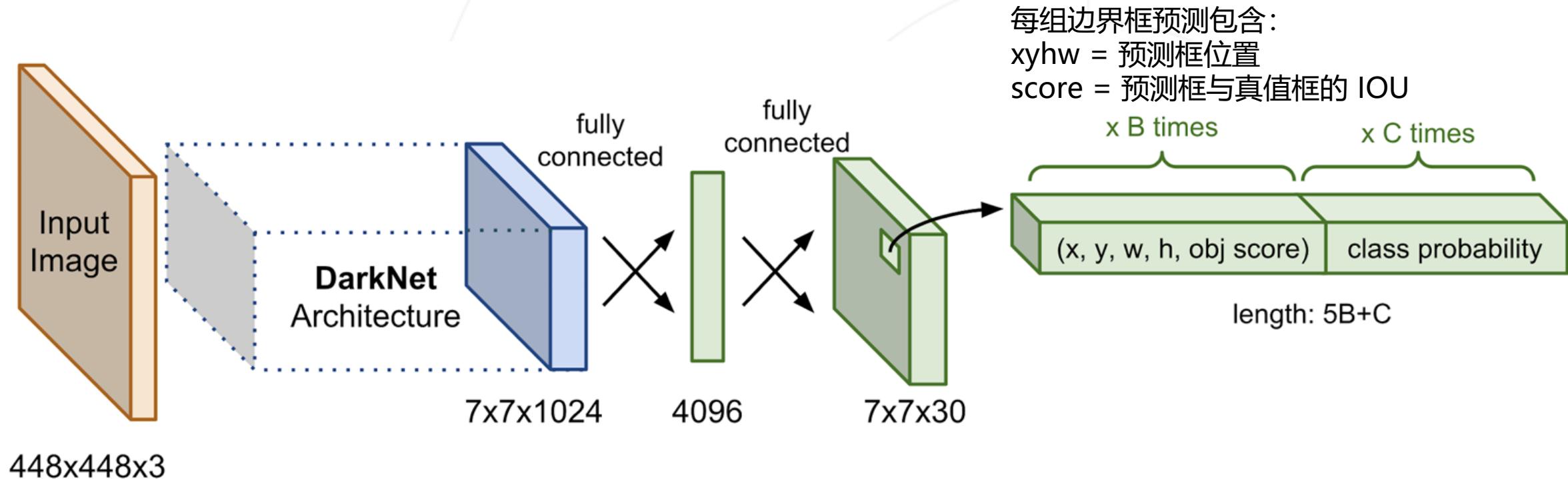


提出 Focal Loss

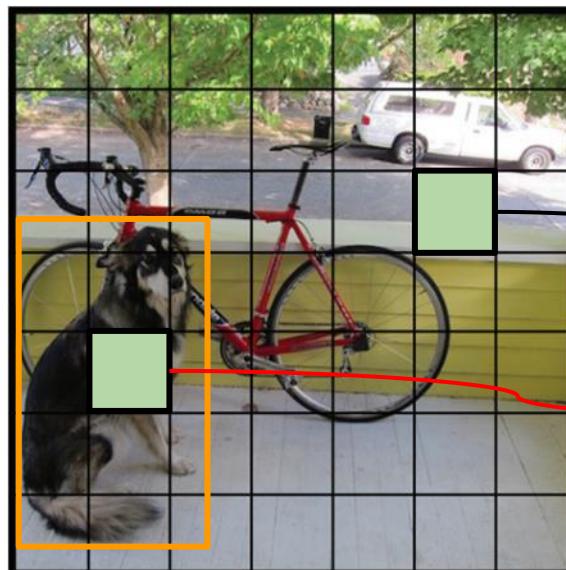


多尺度特征图直接回归

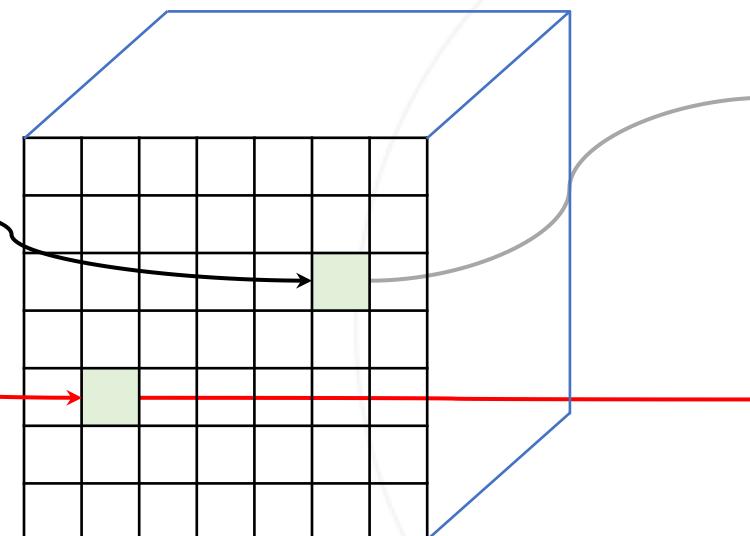
- 最早的单阶段算法之一
- DarkNet 结构的主干网络产生特征图
- 全连接层产生全部空间位置的预测结果，每个位置包含 C 维分类概率和 B 组边界框预测



- 将原图切分成 $S \times S$ 大小的格子，对应预测图上 $S \times S$ 个像素位置。
- 如果原图上某个物体的中心落于某个格子内，则对应位置的预测值应给出该物体的类别和边界框的位置（基于格子边界的偏移量）。
- 其余位置应预测为背景类别，不关心边界框预测结果。



将图像划分为 7×7 的格子



7×7 分辨率的预测图
每个位置 $5B+C$ 个通道
 $= B$ 个框 + C 个类别的预测

类别预测 = 背景
边界框不计算 loss

类别预测 = 狗
边界框预测：
 X, Y = 相对于格子边界的偏移量
 W, H = 窗大小 / 图像大小
分数 = 预测框与真值框的交并比

多任务学习：回归、分类共同计入损失函数，通过 λ 控制权重

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

边界框需要产生**物体**预测时，
计算边界框坐标的回归损失

边界框需要产生**类别**（物体/背景）预测时，
计算边界框置信度的回归损失

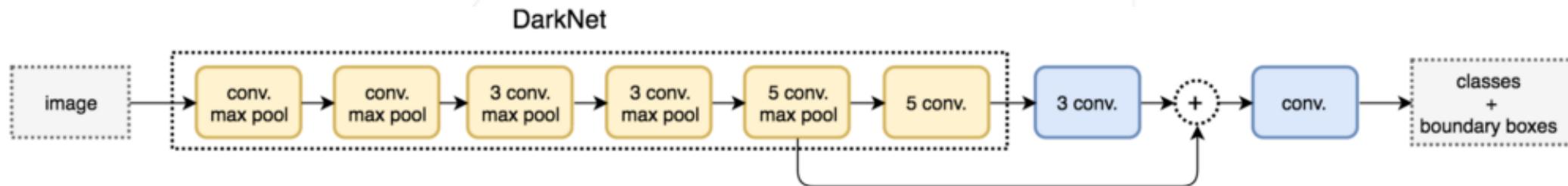
边界框需要产生**物体**预测时，
计算C个类别概率的回归损失

- 快！在Pascal VOC 数据集上，使用自己设计的 DarkNet 结构可以达到实时速度，使用相同的 VGG 可以达到 3 倍于 Faster RCNN 的速度
- 不依赖锚框，直接回归边界框

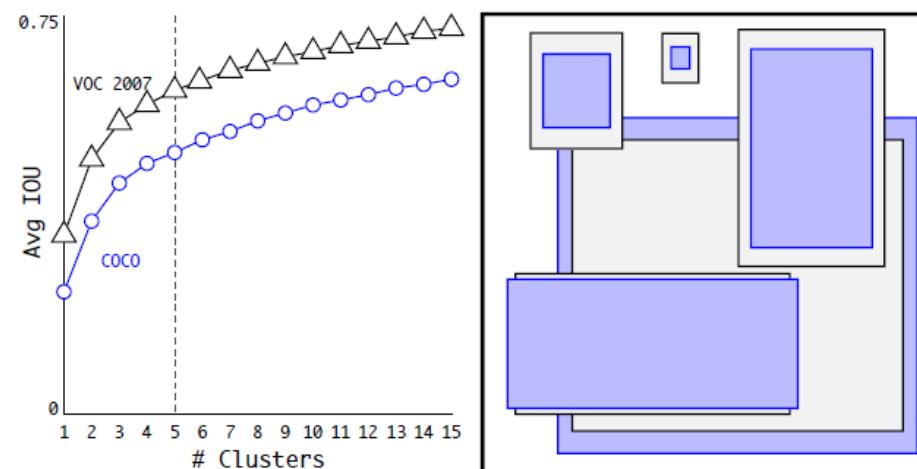
检测算法	主干网络	训练集	检测精度 (mAP)	检测速度 (FPS)
Fast YOLO	9层 DarkNet	VOC 2007 + 2012	52.7	155
YOLO	24层 DarkNet	VOC 2007 + 2012	63.4	45
Faster R-CNN VGG16	VGG 16	VOC 2007 + 2012	73.2	7
YOLO VGG 16	VGG 16	VOC 2007 + 2012	66.4	21

- 由于每个格子只能预测 1 个物体，因此对重叠物体、尤其是大量重叠的小物体容易产生漏检
- 直接回归边界框有难度，回归误差较大

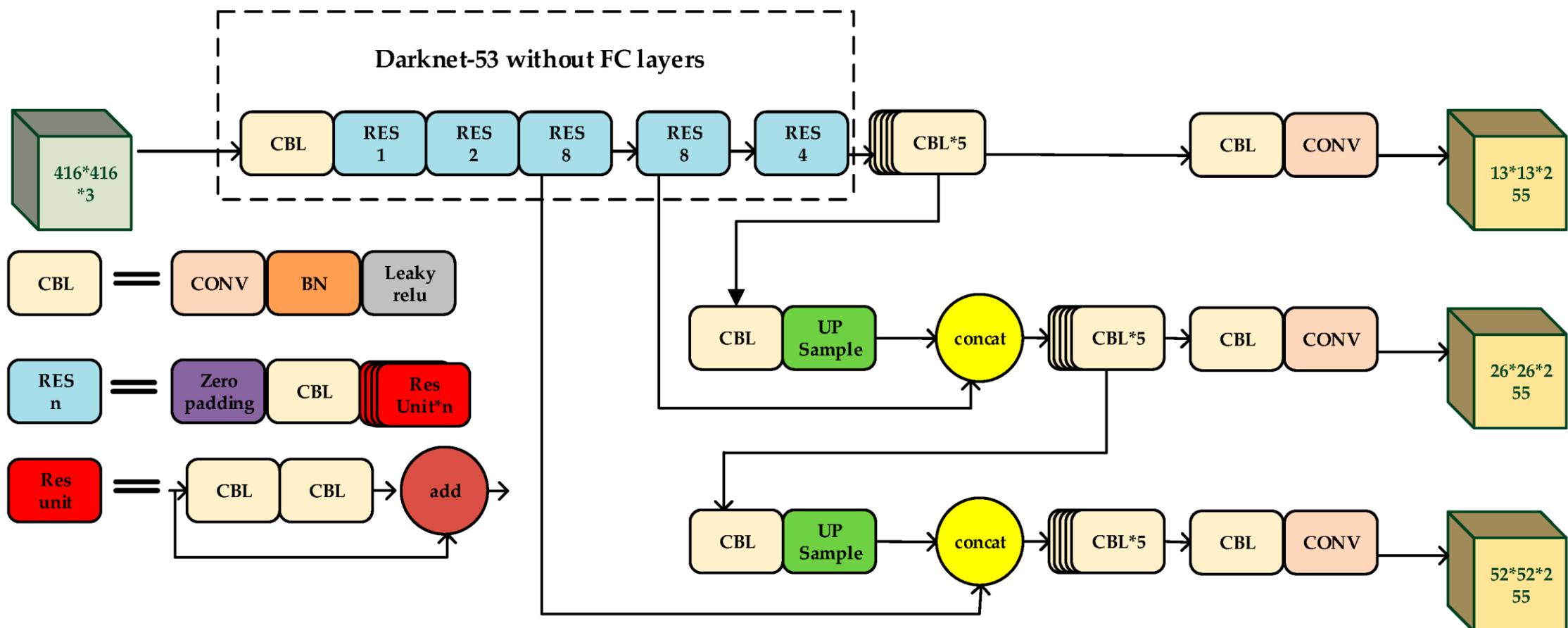
- 新的 DarkNet-19 结构的主干网络，加入 BN



- 加入锚框，并使用聚类方法设定锚框尺寸



- 主干网络加入残差结构 → DarkNet 53
- 加入类 FPN 结构，基于多尺度特征图预测



无锚框目标检测算法

Anchor-free Methods

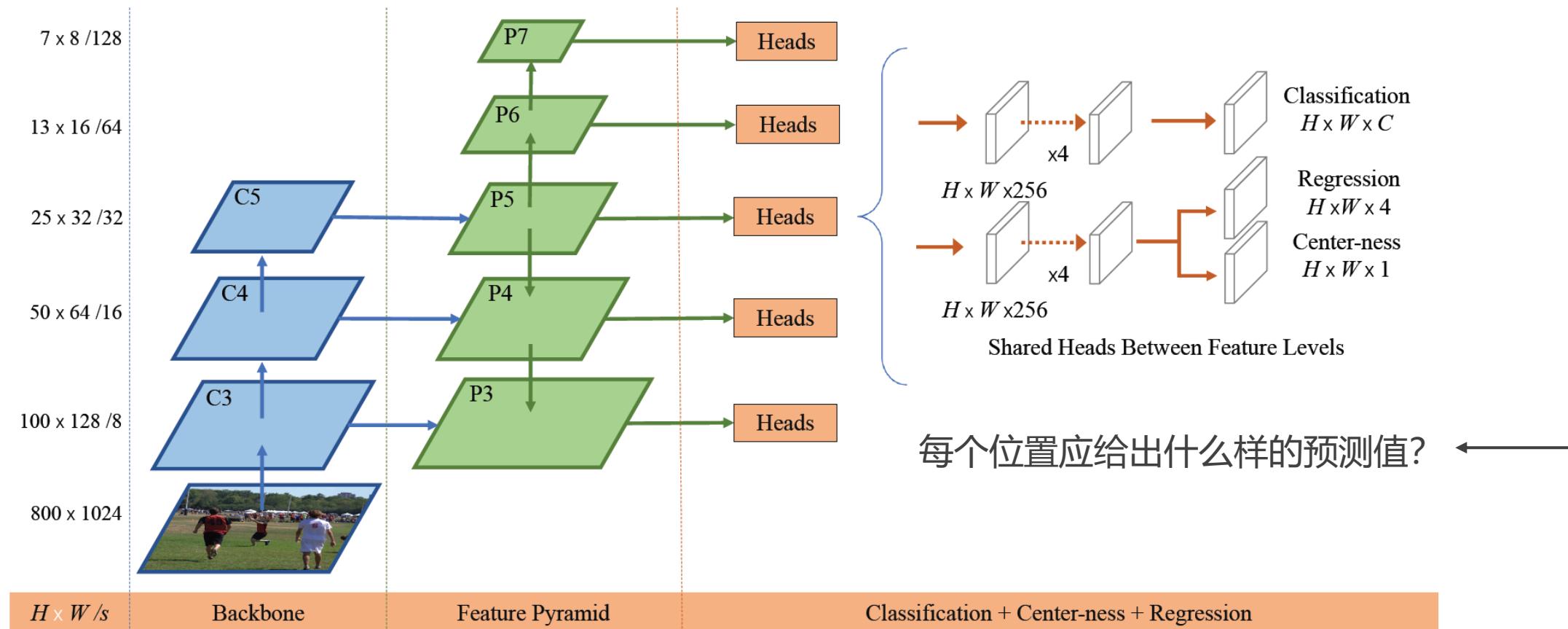
锚框

- Faster RCNN、YOLO v2 / v3、RetinaNet 都是基于锚框的检测算法
- 锚框引入了大量超参数（如大小、长宽比、数量、IoU 阈值等），需要人工调整
- 锚框涉及大量复杂计算，如锚框与真值框之间的 IoU 以及锚框与真值框的匹配等

无锚框

- 无锚框算法尝试直接基于特征学习边界框的位置，不依赖锚框，从而减降低模型复杂度
- YOLO v1 是无锚框算法，但由于提出时间较早，相关技术并不完善，性能不如基于锚框的算法

- 主干网络与特征金字塔产生多尺度特征图
- 检测头在多级特征图上完成预测，对于每个位置，预测**类别、边界框位置和中心度**三组数值



基本规则

- 如果特征图上某个位置（在原图上对应的位置）位于某个物体的**边界框内部**，则该位置的预测值应给出该物体的：

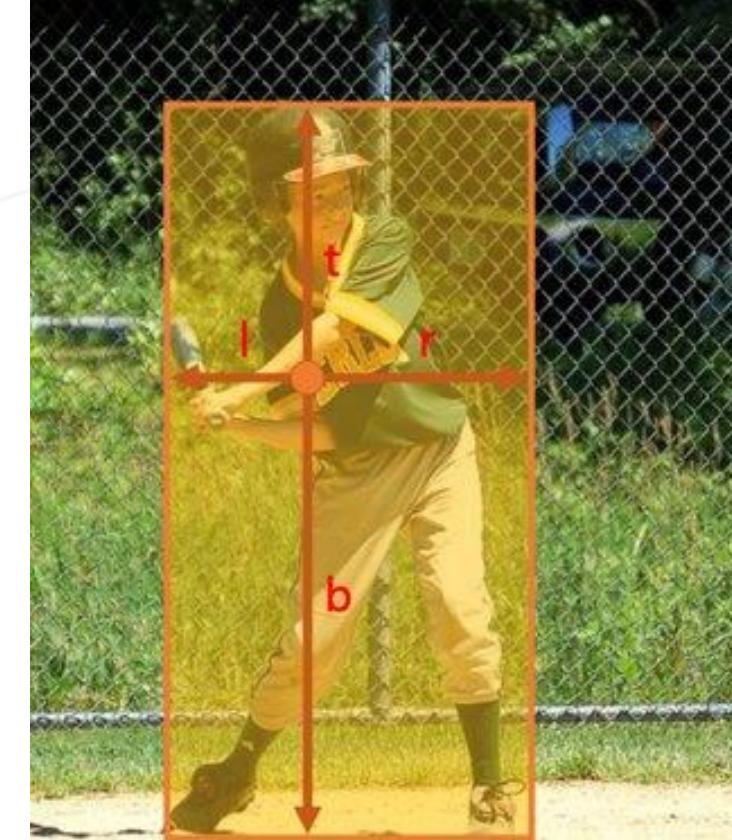
- 类别
- 边界框相对于该**位置**的偏移量
- 中心度，用于衡量预测框的优劣

对比：锚框算法基于 IoU 判定正负样本，FCOS 有更多正样本

- 如果某个位置不在任何物体的边界框内部，分类为背景。

对比：锚框算法给出基于锚框的偏移量

类似 YOLO 的 score



回顾

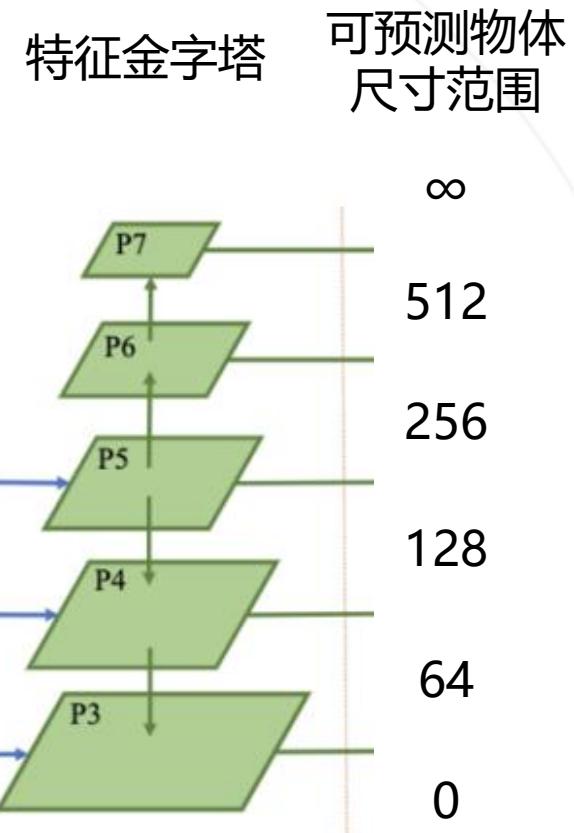
- YOLO 直接回归边界框，但不能处理重叠物体
- 基于锚框的算法可以基于不同的锚框，产生不同目标物体的预测

FCOS 如何处理重叠问题？

重叠的物体尺度通常不同，由不同尺度的特征图给出对应目标物体的预测：

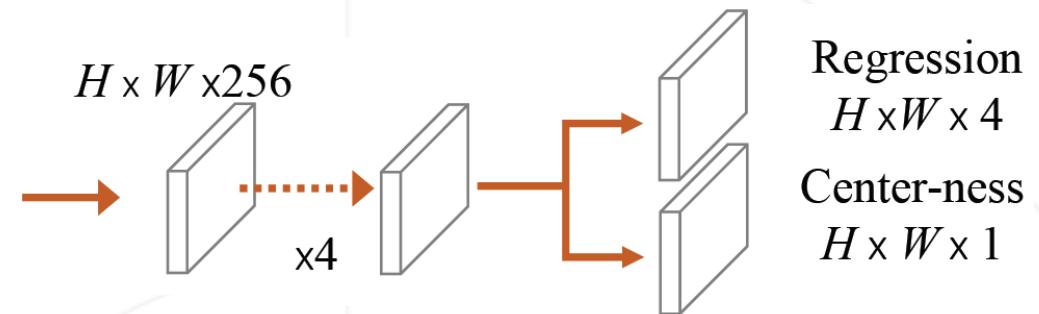
- 小物体的预测基于低层次特征图给出
- 大物体的预测基于高层次特征图给出

FCOS 的多尺度匹配策略可以很大程度解决重叠问题。如果同层特征图还有重叠，预测重叠物体中较小的那个



FCOS 将所有框内的位置归类为正样本：

- 有更多正样本用于训练
- 但也会让分类器更容易在物体周围产生低质量预测框



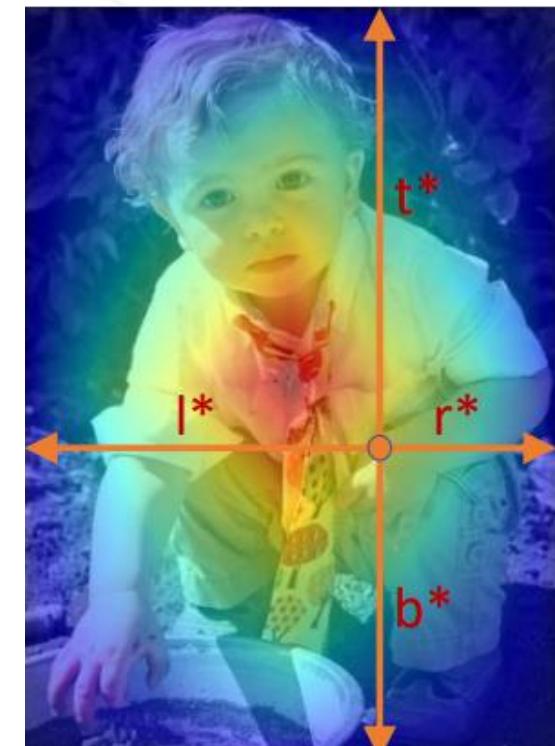
解决方案：

- 每个位置额外预测一个**中心度值**，用于该预测框的优劣
- 推理阶段：置信度 = $\sqrt{\text{中心度} \times \text{分类概率}}$

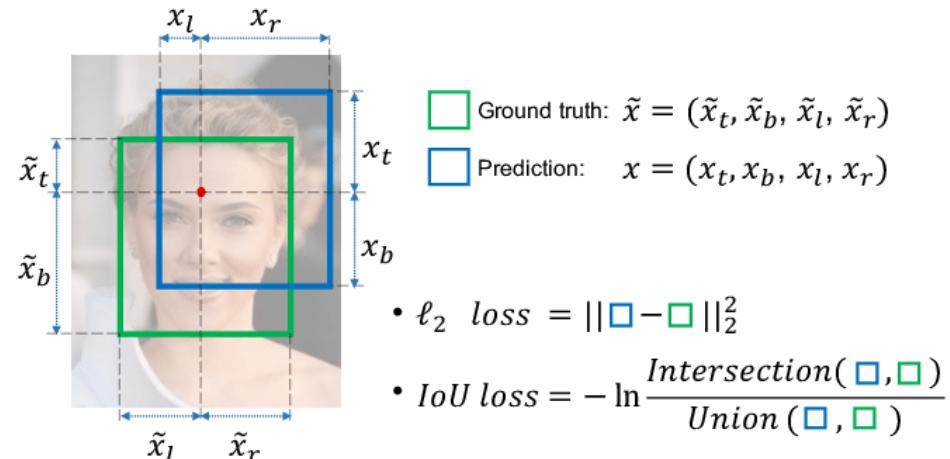
中心度定义：

$$\text{centerness}^* = \sqrt{\frac{\min(l^*, r^*)}{\max(l^*, r^*)} \times \frac{\min(t^*, b^*)}{\max(t^*, b^*)}}.$$

几何意义：位置越靠近边界框中心越接近 1，越靠近边界越接近 0



- 位置:** 每个位置的分类损失 $L_{cls}(\mathbf{p}_{x,y}, c_{x,y}^*)$ ，使用 Focal loss
- 边界框回归:** 对于前景样本的边界框回归损失 $L_{reg}(\mathbf{t}_{x,y}, \mathbf{t}_{x,y}^*)$ ，使用 IoU loss



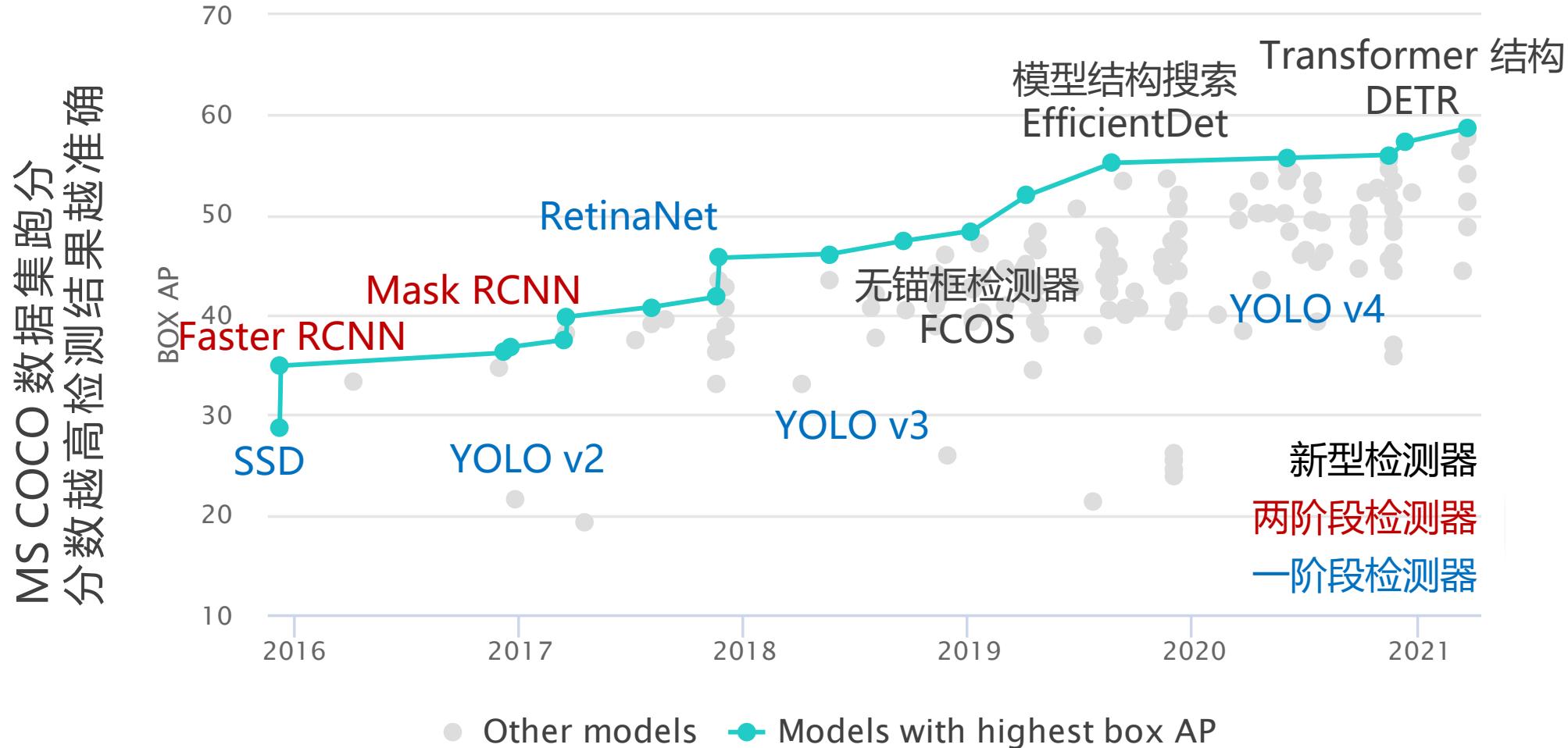
- 中心度:** 对于前景样本的 Center-ness 值，因为介于 0-1 之间，使用 BCE loss
- 三者求和共同优化

x, y	位置	$\mathbf{p}_{x,y}$	类别概率预测
		$c_{x,y}^*$	类别真值

$\mathbf{t}_{x,y} = (l, t, r, b)$	边界框预测值
$\mathbf{t}_{x,y}^* = (l^*, t^*, r^*, b^*)$	边界框真值

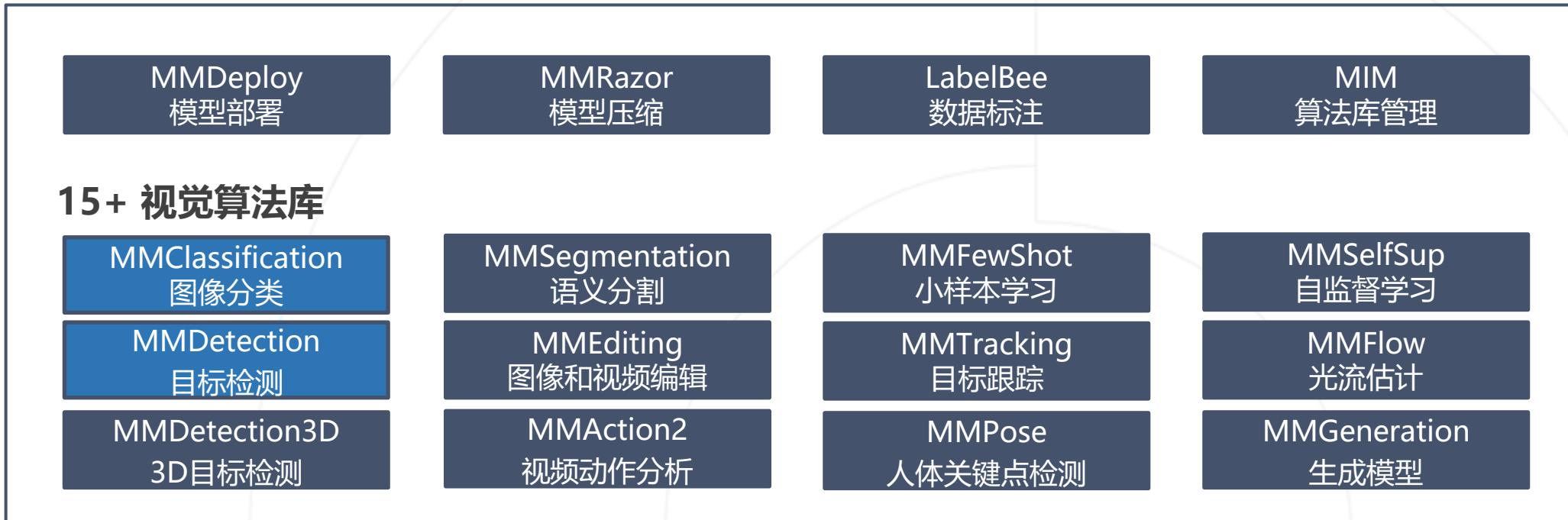
完全重合，IoU loss = 0
完全不重合，IoU loss = ∞

与L2相比，大小物体产生同等大小的loss



OpenMMLab 简介

算法框架



视觉基础库 MMCV

训练框架

 PyTorch

社区运营 – 影响力



用户遍及全球**110**个国家和地区



- 2018-10 发布
- 2019-07 v1.0
- 2020-05 v2.0

任务支持

目标检测

实例分割

覆盖广泛

~380 个
预训练模型

~60 篇
论文复现

常用学术数据集

算法丰富

两阶段检测器

一阶段检测器

级联检测器

无锚点检测器

Transformer

使用方便

训练工具

测试工具

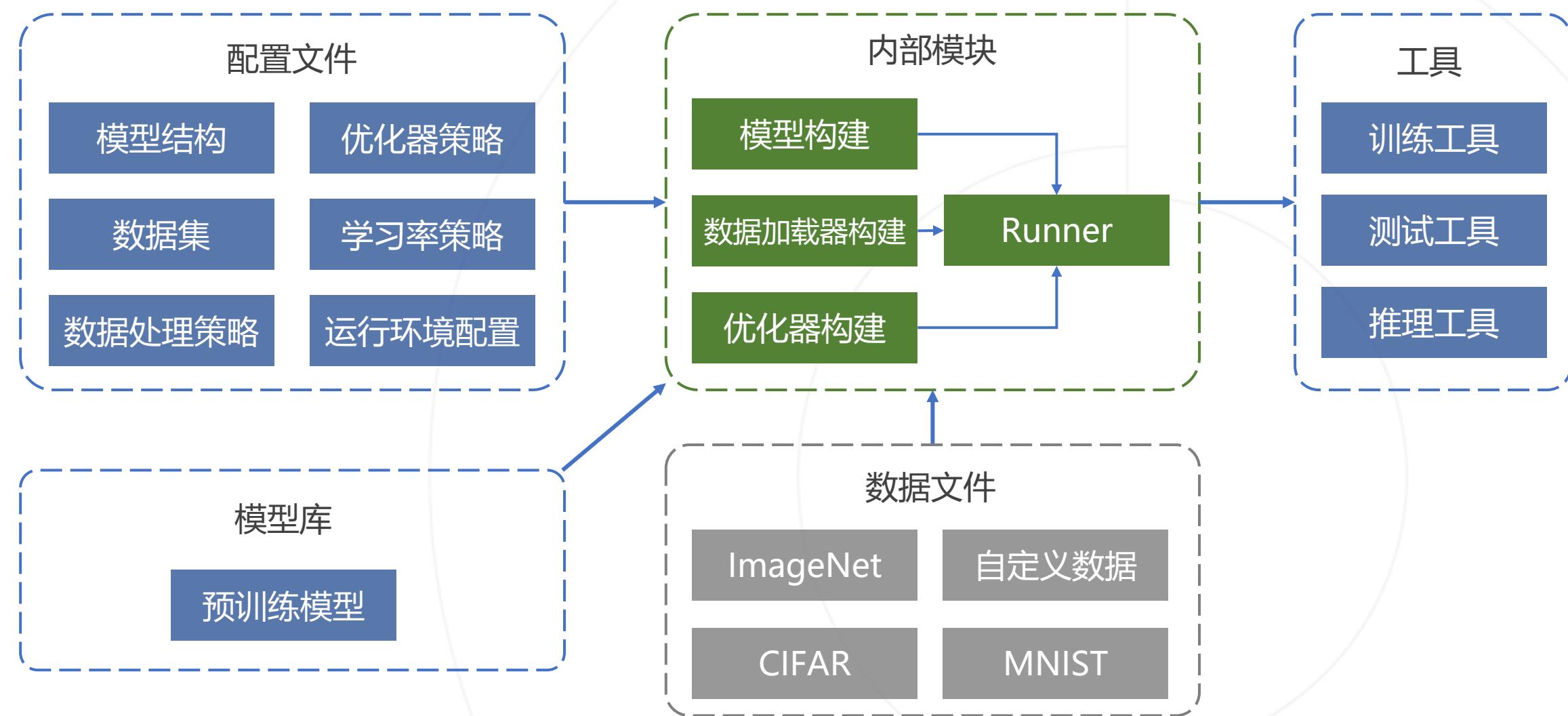
推理 API

open-mmlab / mm detection



<https://github.com/open-mmlab/mmdetection>

...	
apis	训练、推理、测试的高层次 API
core	anchor、bbox、mask 等模块的实现
datasets	数据集支持、数据预处理与数据增强
models	检测模型的实现
utils 辅助工具	
__init__.py	
version.py	



谢谢大家

谢谢大家