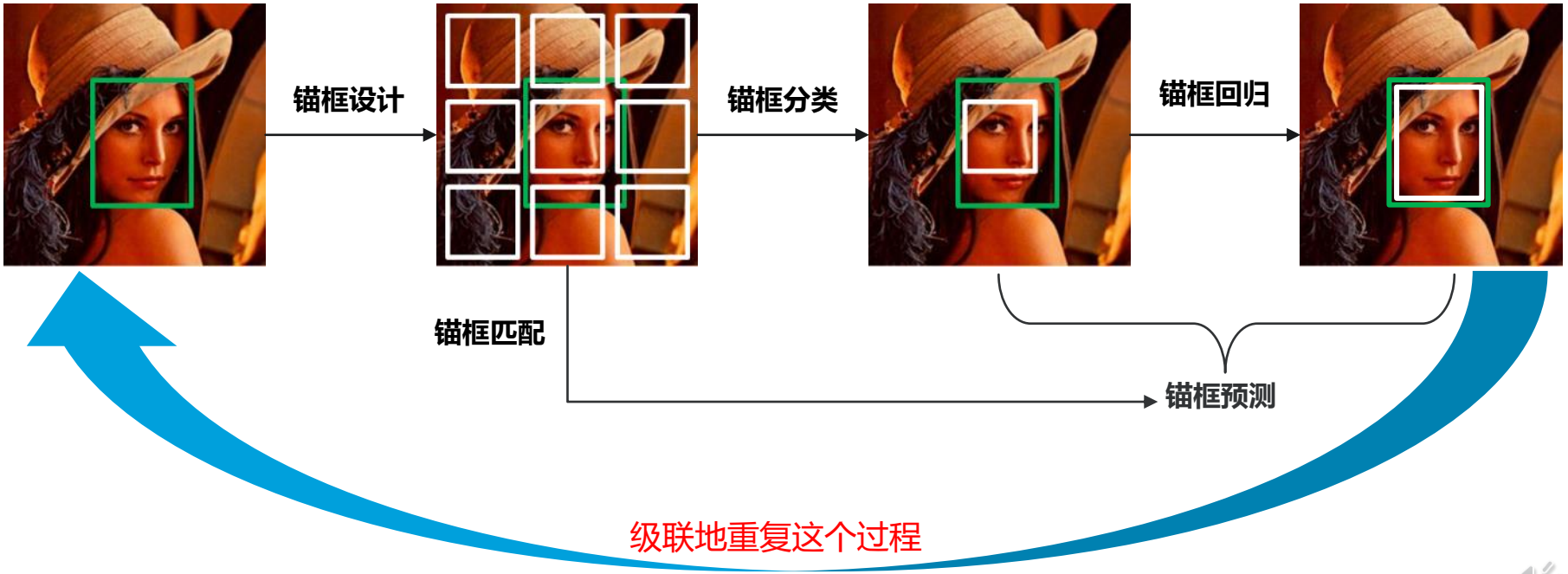
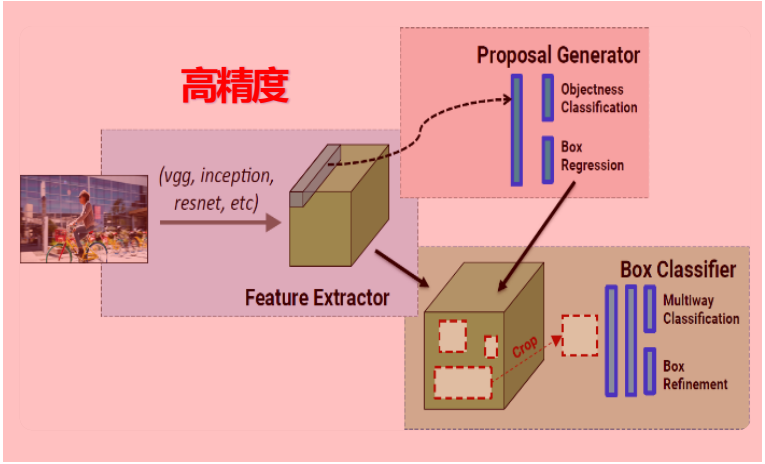
内容回顾：基于锚框的检测算法

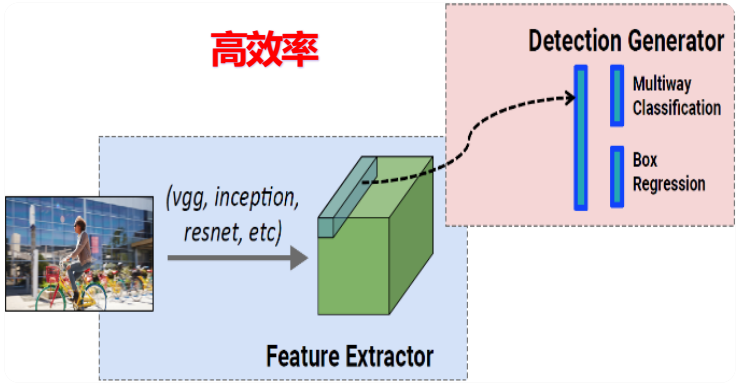
锚框机制是该类物体检测算法的核心



内容回顾：多阶段法

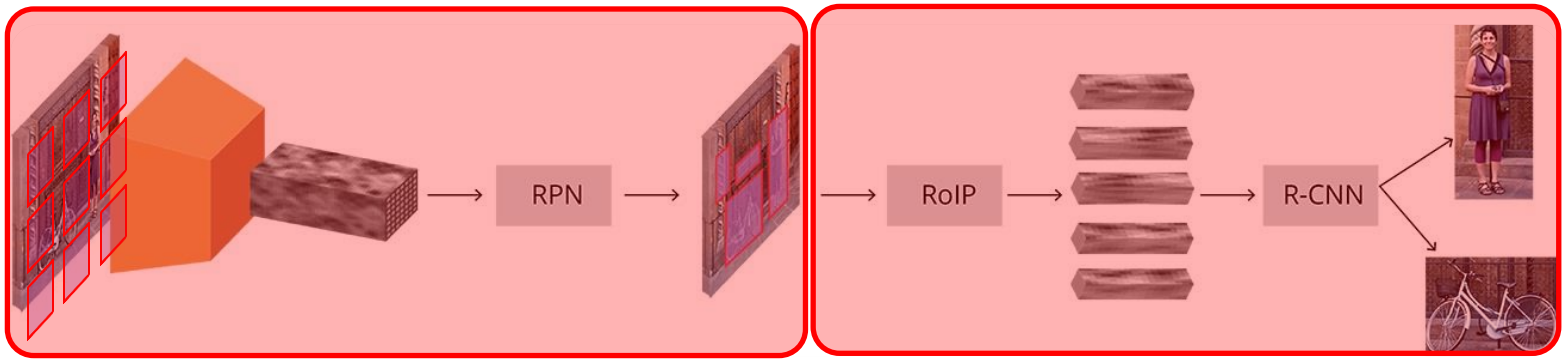


多阶段法



单阶段法

内容回顾：多阶段法Faster R-CNN



Faster R-CNN中RPN步骤：

① 整张图传入VGG16或ResNet提取特征

② 选择下采样倍数为16的特征层作为检测层

③ 根据检测层预设一系列大小和比例的锚框（9个）

④ 对锚框进行二分类和回归得到若干候选区域

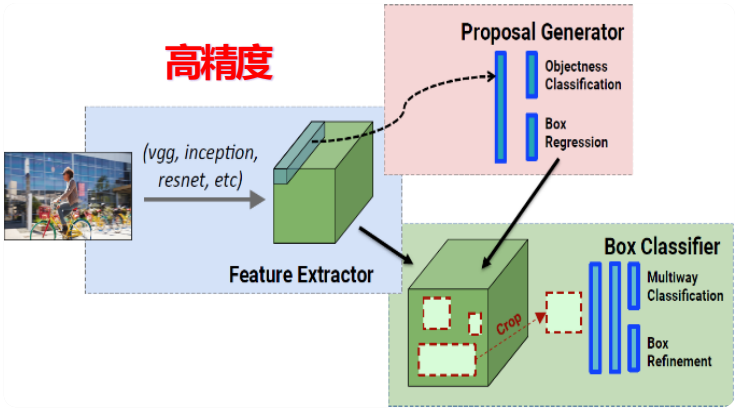
Faster R-CNN中Fast R-CNN步骤：

① 利用RoIPooling在检测层的特征上提取每个候选区域对应的特征

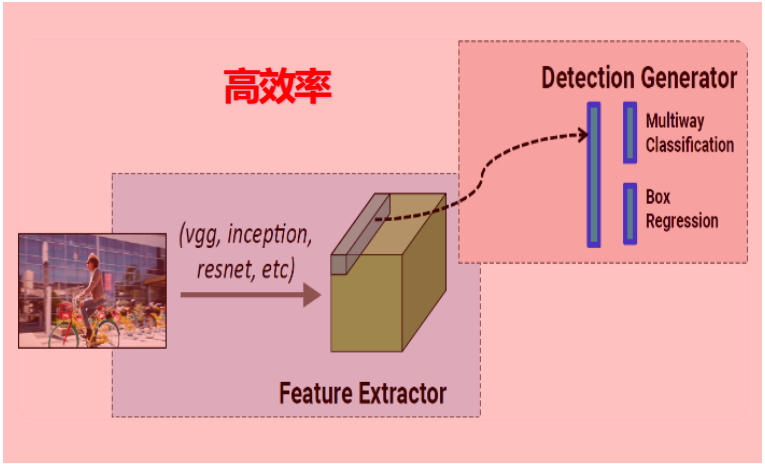
② 输入CNN/FC子网络来增强候选区域的特征

③ 对候选区域进行多分类和回归得到检测结果

基于锚框的物体检测

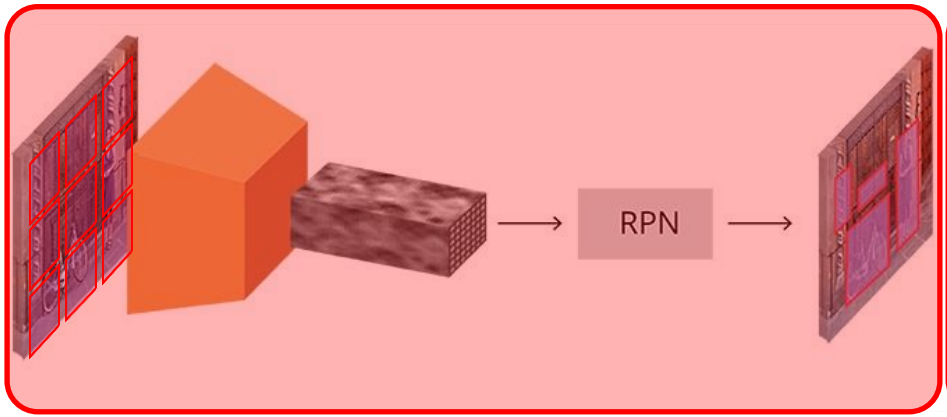


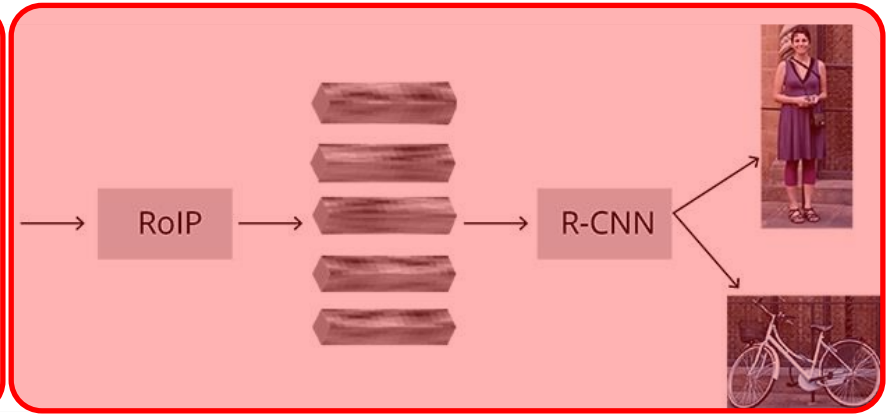
多阶段法



单阶段法

基于锚框的单阶段检测算法





Faster R-CNN中RPN步骤：

① 整张图传入VGG16或ResNet提取特征

② 选择下采样倍数为16的特征层作为检测层

③ 根据检测层预设一系列大小和比例的锚框（9个）

④ 对锚框进行二分类和回归得到若干候选区域

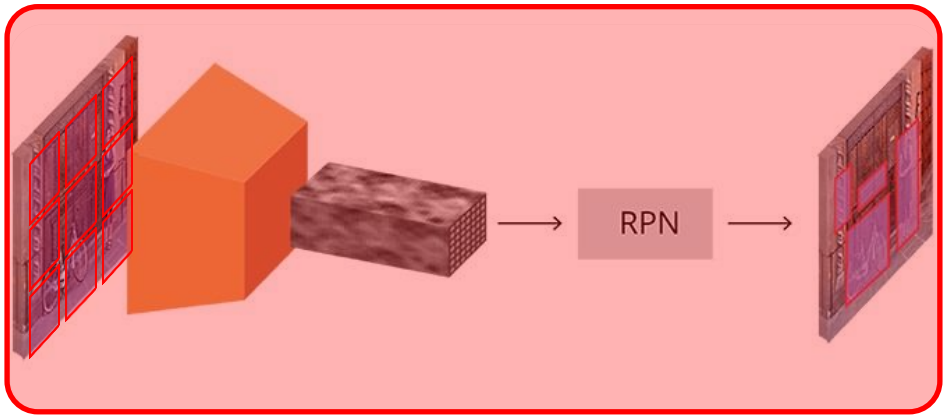
Faster R-CNN中Fast R-CNN步骤：

① 利用RoIPooling在检测层的特征上提取每个候选区域对应的特征

② 输入CNN/FC子网络来增强候选区域的特征

③ 对候选区域进行多分类和回归得到检测结果

基于锚框的单阶段检测算法



Faster R-CNN中RPN步骤：

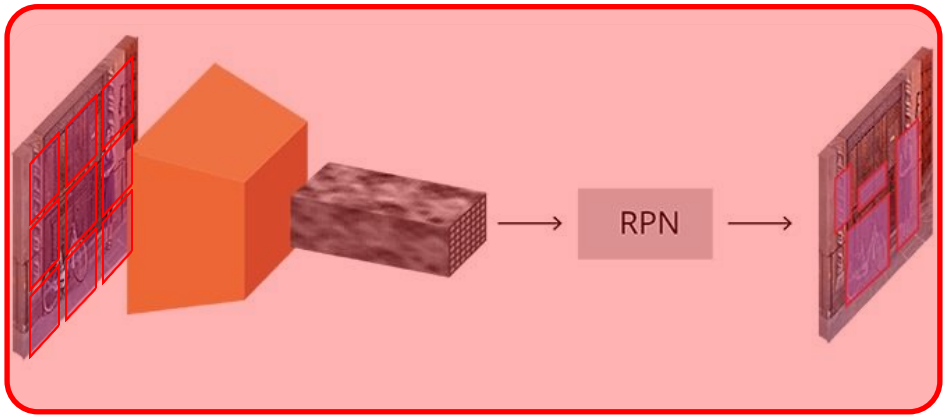
① 整张图传入VGG16或ResNet提取特征

② 选择下采样倍数为16的特征层作为检测层

③ 根据检测层预设一系列大小和比例的锚框（9个）

④ 对锚框进行二分类和回归得到若干候选区域

基于锚框的单阶段检测算法



Faster R-CNN中RPN步骤：

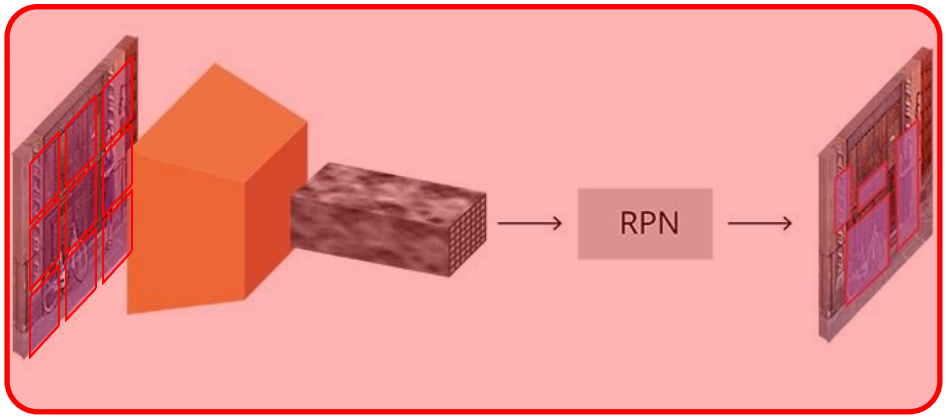
① 整张图传入VGG16或ResNet提取特征

② 选择下采样倍数为16的特征层作为检测层

③ 根据检测层预设一系列大小和比例的锚框（9个）

④ 对锚框进行~~二分类~~ **多分类**和回归得到~~候选区域~~ **检测结果**

基于锚框的单阶段检测算法



Faster R-CNN中RPN步骤：

① 整张图传入VGG16或ResNet提取特征

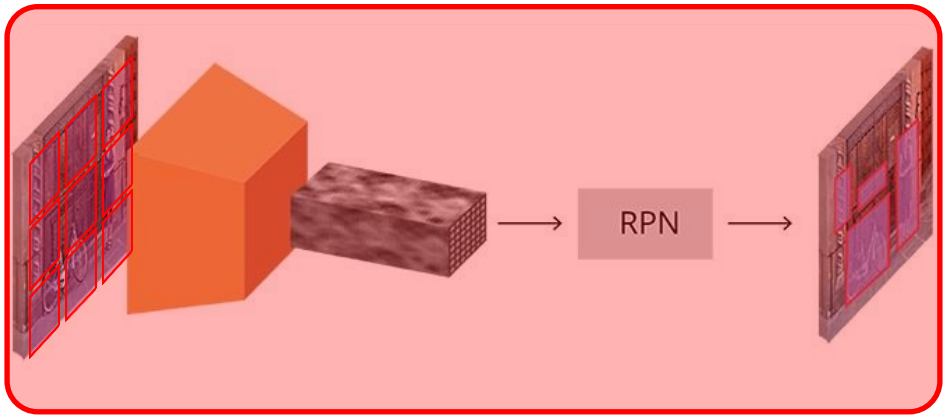
② 选择下采样倍数为16的特征层作为检测层

③ 根据检测层预设一系列大小和比例的锚框（9个）

④ 对锚框进行~~二分类~~ **多分类**和回归得到~~候选区域~~ **检测结果**

基于锚框的单阶段检测算法流程

基于锚框的单阶段检测算法



Faster R-CNN中RPN步骤：

① 整张图传入VGG16或ResNet提取特征

② 选择下采样倍数为16的特征层作为检测层

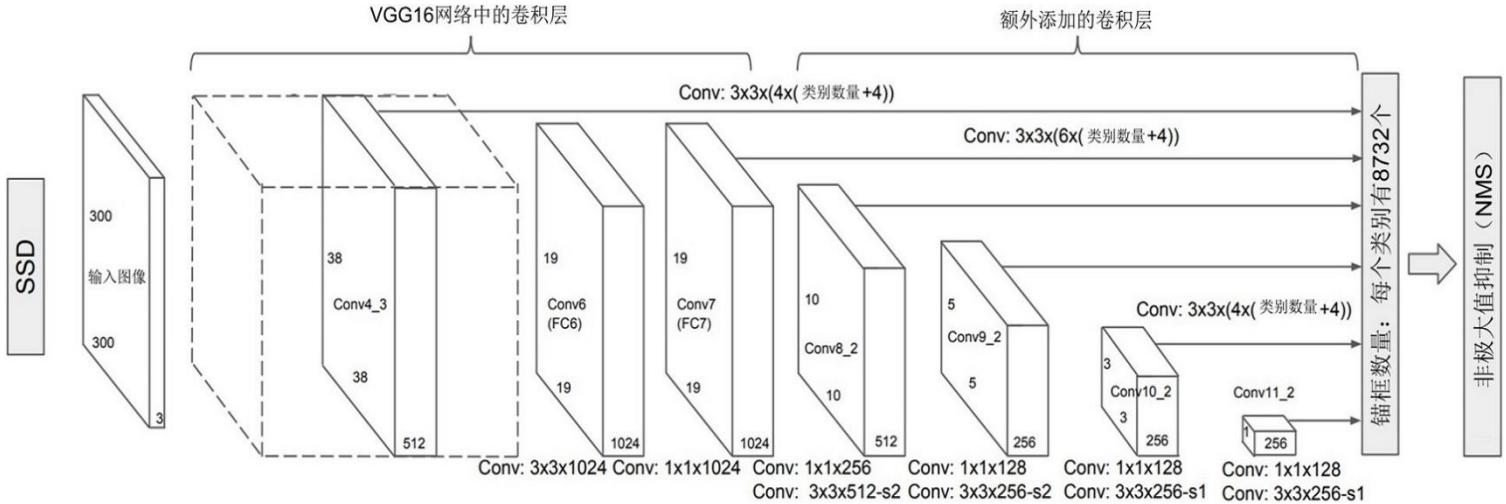
③ 根据检测层预设一系列大小和比例的锚框（9个）

④ 对锚框进行~~二分类~~ **多分类**和回归得到~~候选区域~~ **检测结果**

SSD **RetinaNet**

基于锚框的单阶段检测算法流程

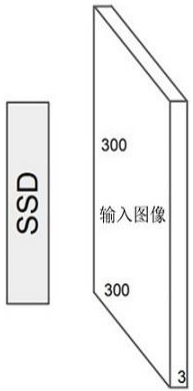
基于锚框的单阶段检测算法：SSD (Single Shot MultiBox Detector)



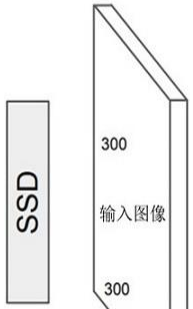
- 输入图像：经过丰富的数据增广，得到300x300或512x512的输入图像
- 基础网络：ImageNet预训练的VGG16网络 + 一些额外的卷积层
- 多检测层：选择6个特征层作为检测层，每层关联不同的锚框
- 难负样本挖掘：利用误差损失值的大小，选择出3倍正样本的负样本参与训练

SSD: Single Shot MultiBox Detector, ECCV 2016

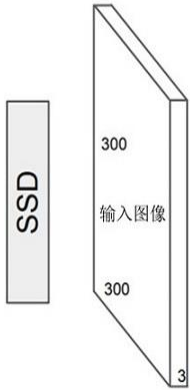
SSD检测算法：输入图像

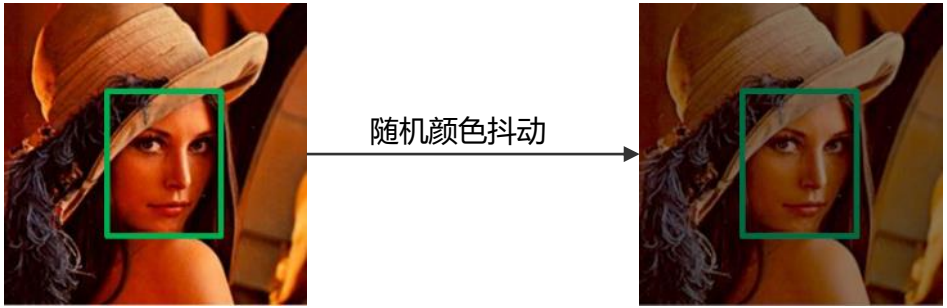


SSD检测算法：输入图像

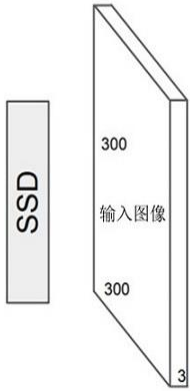


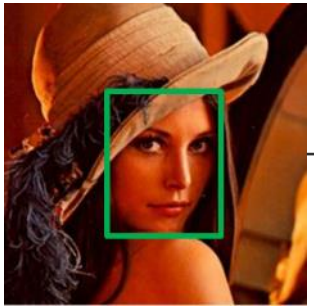
SSD检测算法：输入图像





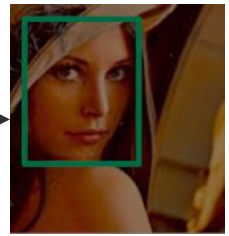
SSD检测算法：输入图像



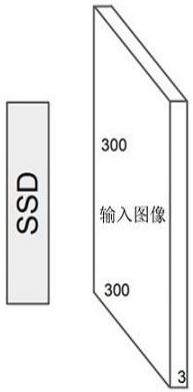


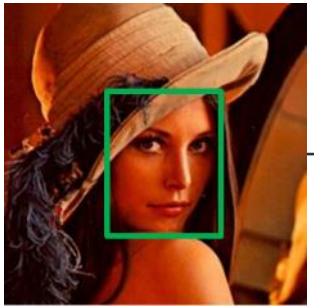
随机颜色抖动

随机裁剪



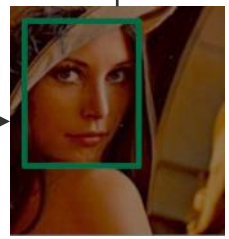
SSD检测算法：输入图像



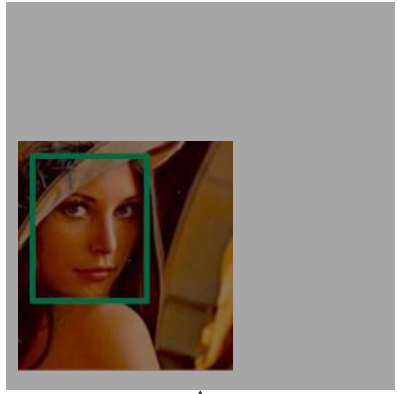


随机颜色抖动

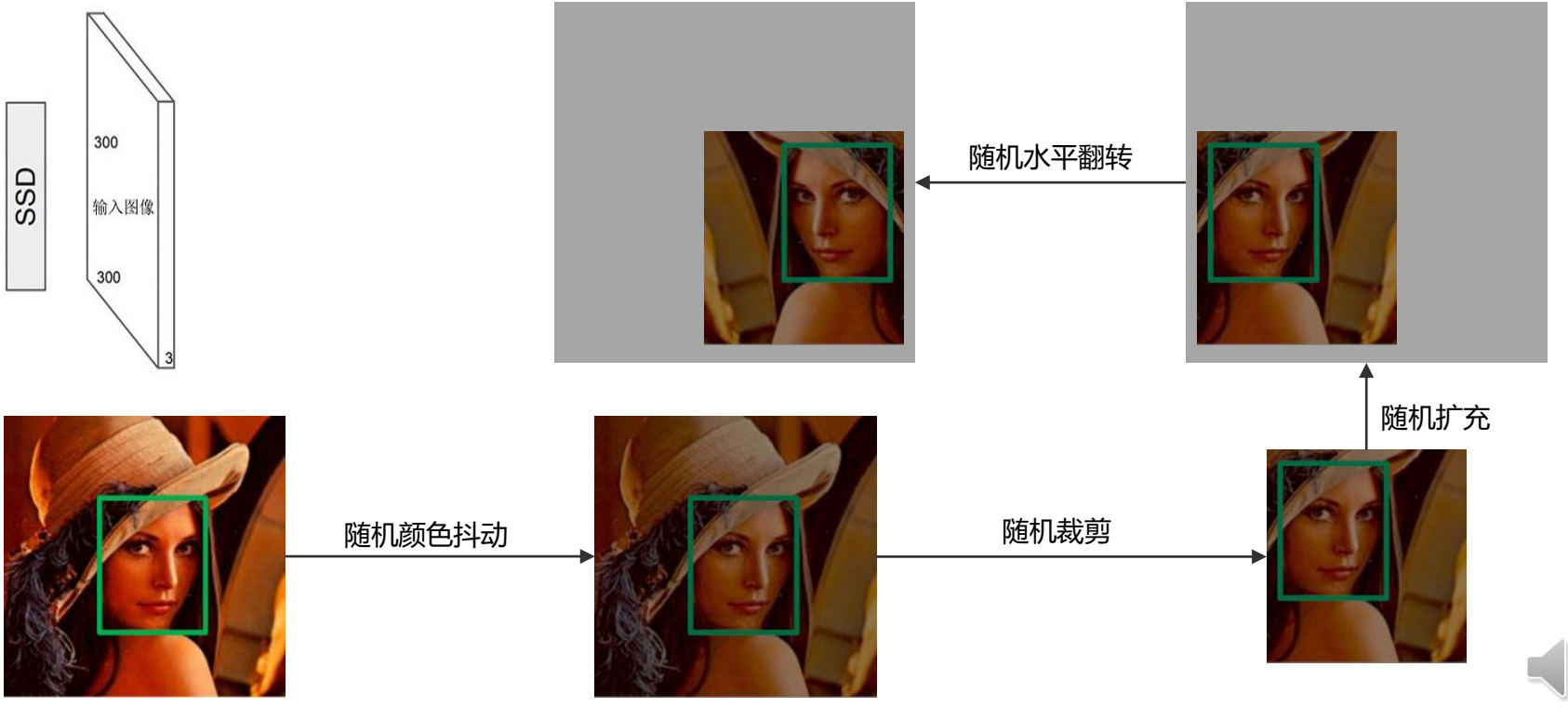
随机裁剪



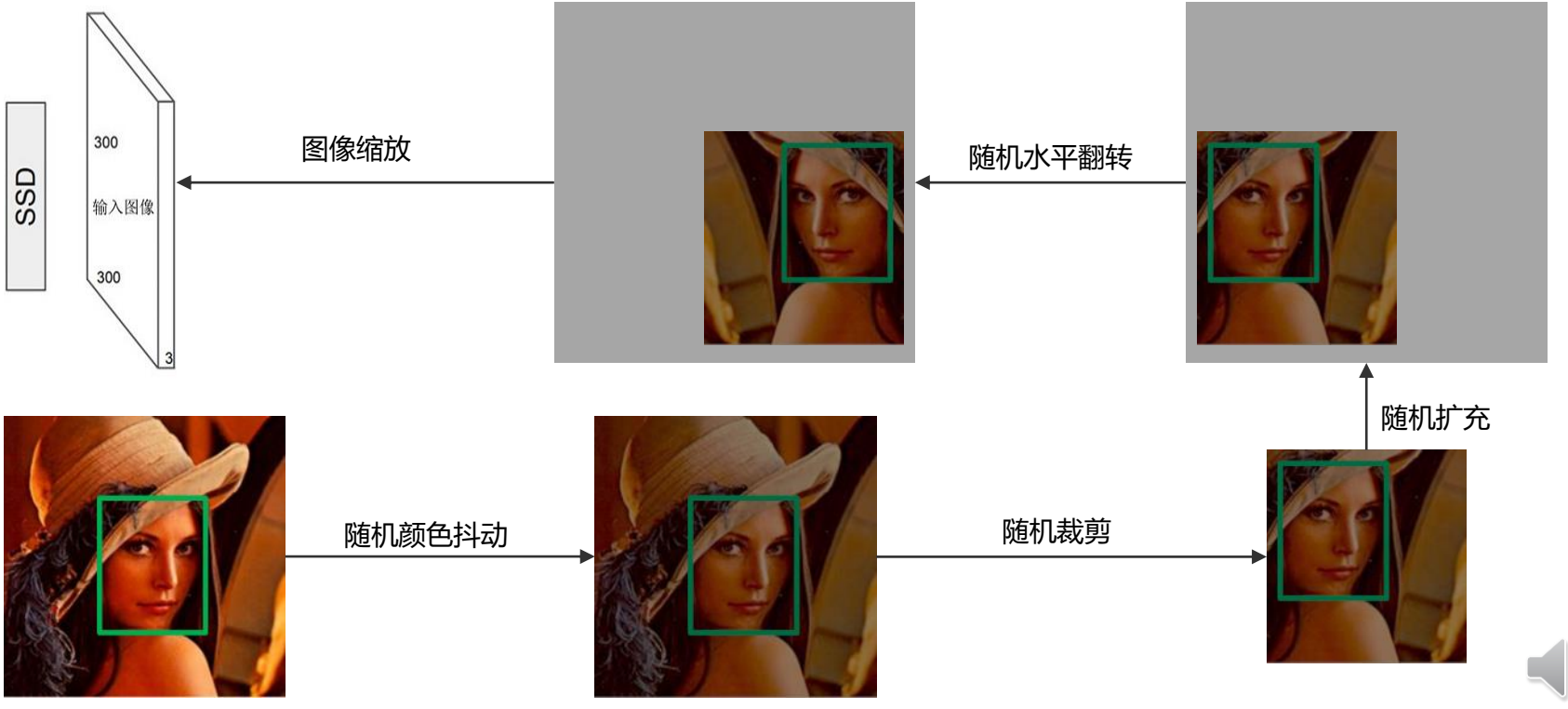
随机扩充



SSD检测算法：输入图像

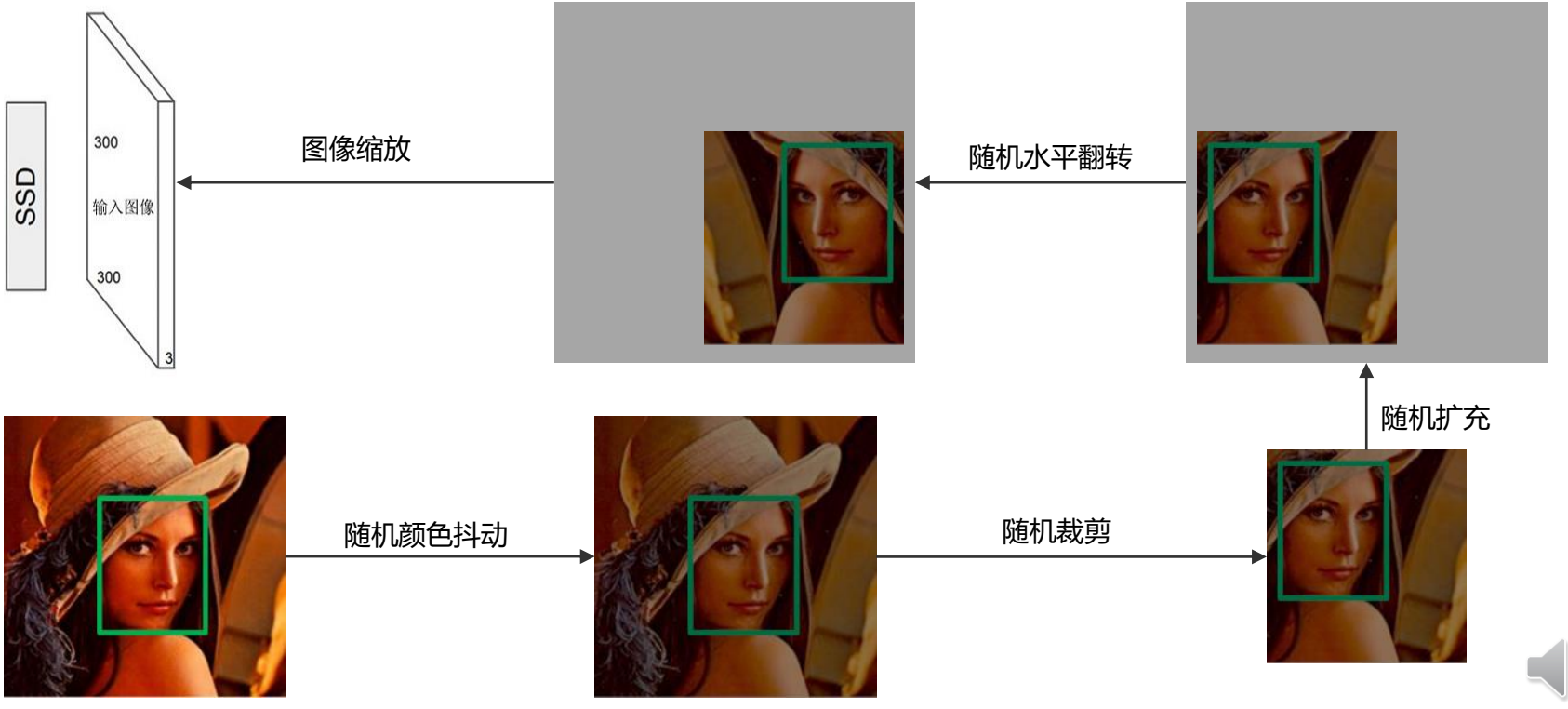


SSD检测算法：输入图像

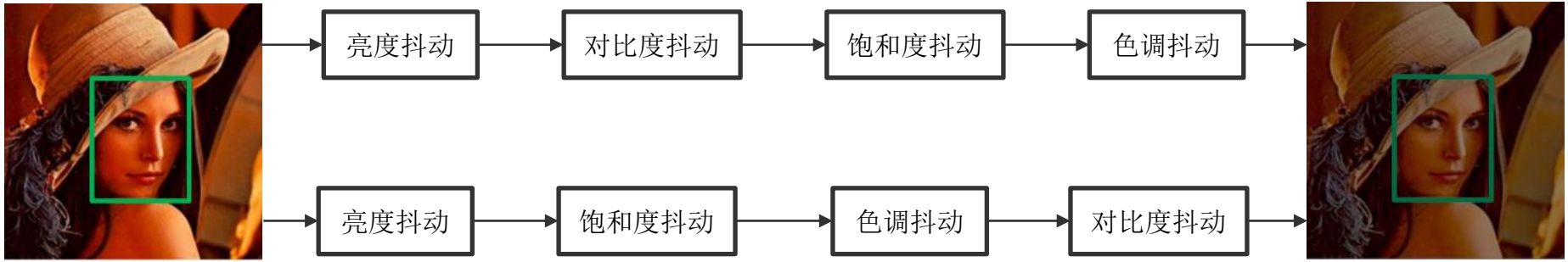


SSD检测算法：输入图像

测试阶段，只对输入图像进行图像缩放操作，其他数据增广操作不执行



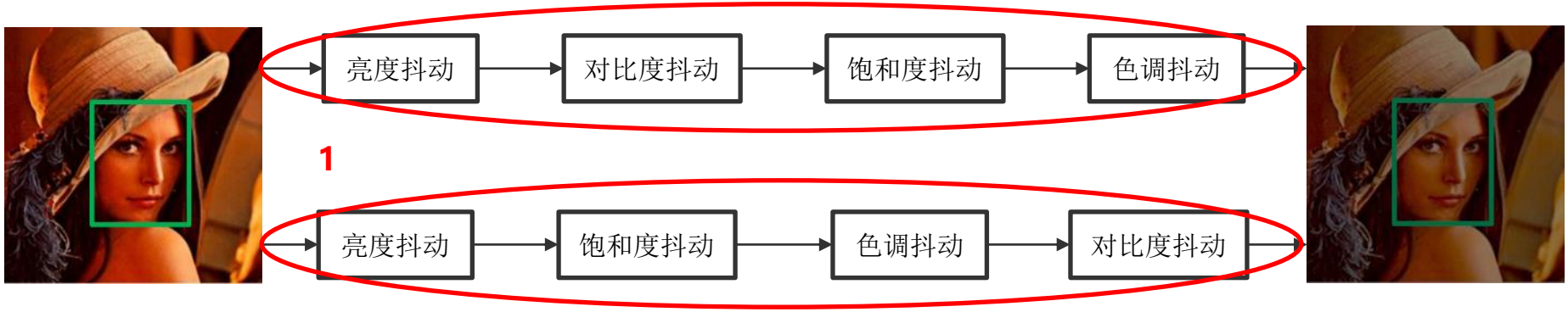
SSD检测算法：输入图像的随机颜色抖动



- **亮度抖动**：RGB图像 + random.uniform(-32, 32)
- **对比度抖动**：RGB图像 * random.uniform(0.5, 1.5)
- **饱和度抖动**：HSV图像的S通道 * random.uniform(0.5, 1.5)
- **色调抖动**：HSV图像的H通道 + random.randint(-18, 18)

注：HSV颜色空间, H->色调, S->饱和度, V->明度

SSD检测算法：输入图像的随机颜色抖动



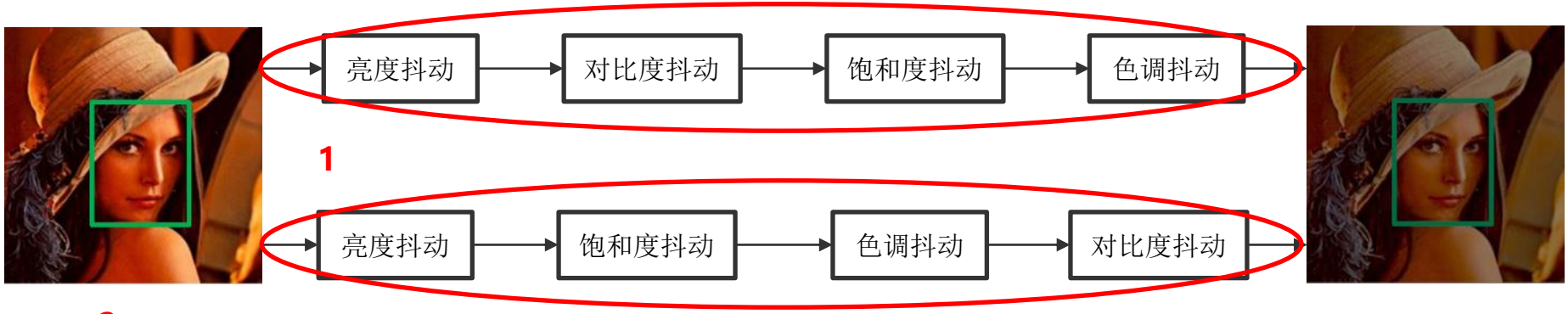
- **亮度抖动**：RGB图像 + random.uniform(-32, 32)
- **对比度抖动**：RGB图像 * random.uniform(0.5, 1.5)
- **饱和度抖动**：HSV图像的S通道 * random.uniform(0.5, 1.5)
- **色调抖动**：HSV图像的H通道 + random.randint(-18, 18)

- 随机的3层含义

1. 从设计的两条固定路线中，以1/2的概率随机选一条

注：HSV颜色空间, H->色调, S->饱和度, V->明度

SSD检测算法：输入图像的随机颜色抖动



- **亮度抖动**：RGB图像 + random.uniform(-32, 32)
- **对比度抖动**：RGB图像 * random.uniform(0.5, 1.5)
- **饱和度抖动**：HSV图像的S通道 * random.uniform(0.5, 1.5)
- **色调抖动**：HSV图像的H通道 + random.randint(-18, 18)

- 随机的3层含义

1. 从设计的两条固定路线中，以1/2的概率随机选一条
2. 每个抖动操作以1/2的概率执行

注：HSV颜色空间, H->色调, S->饱和度, V->明度

SSD检测算法：输入图像的随机颜色抖动

亮度抖动 → 对比度抖动 → 饱和度抖动 → 色调抖动

1

亮度抖动 → 饱和度抖动 → 色调抖动 → 对比度抖动

2

3

- **亮度抖动**：RGB图像 + random.uniform(-32, 32)
- **对比度抖动**：RGB图像 * random.uniform(0.5, 1.5)
- **饱和度抖动**：HSV图像的S通道 * random.uniform(0.5, 1.5)
- **色调抖动**：HSV图像的H通道 + random.randint(-18, 18)

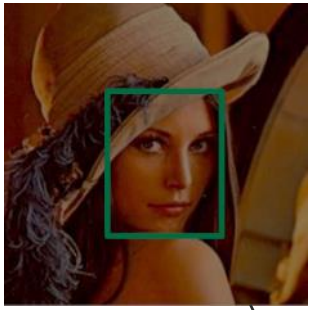
- 随机的3层含义

1. 从设计的两条固定路线中，以1/2的概率随机选一条
2. 每个抖动操作以1/2的概率执行
3. 每个抖动操作中的参数都是随机生成的

注：HSV颜色空间, H->色调, S->饱和度, V->明度

SSD检测算法：输入图像的随机裁剪

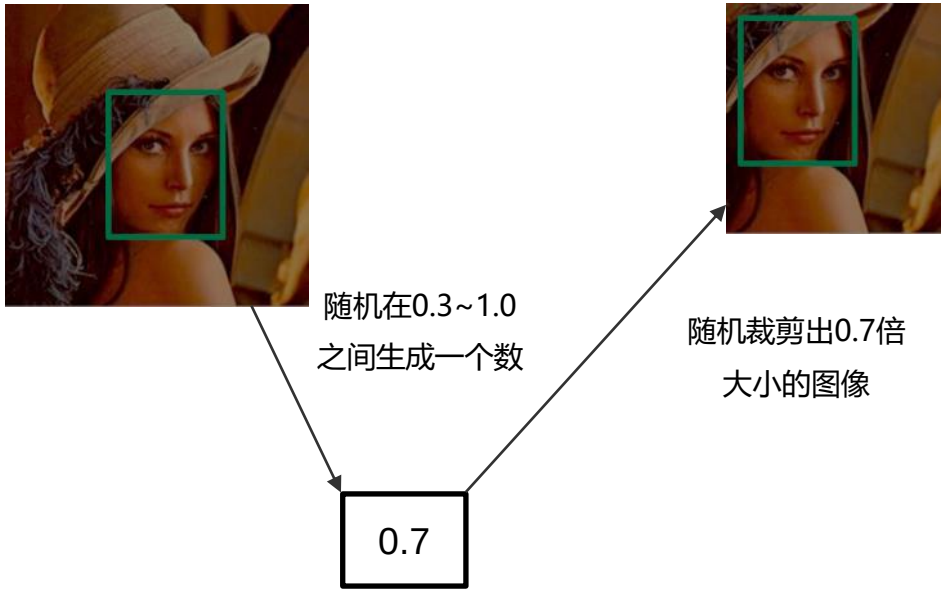
SSD检测算法：输入图像的随机裁剪



随机在0.3~1.0
之间生成一个数

0.7

SSD检测算法：输入图像的随机裁剪



SSD检测算法：输入图像的随机裁剪

随机在0.3~1.0
之间生成一个数

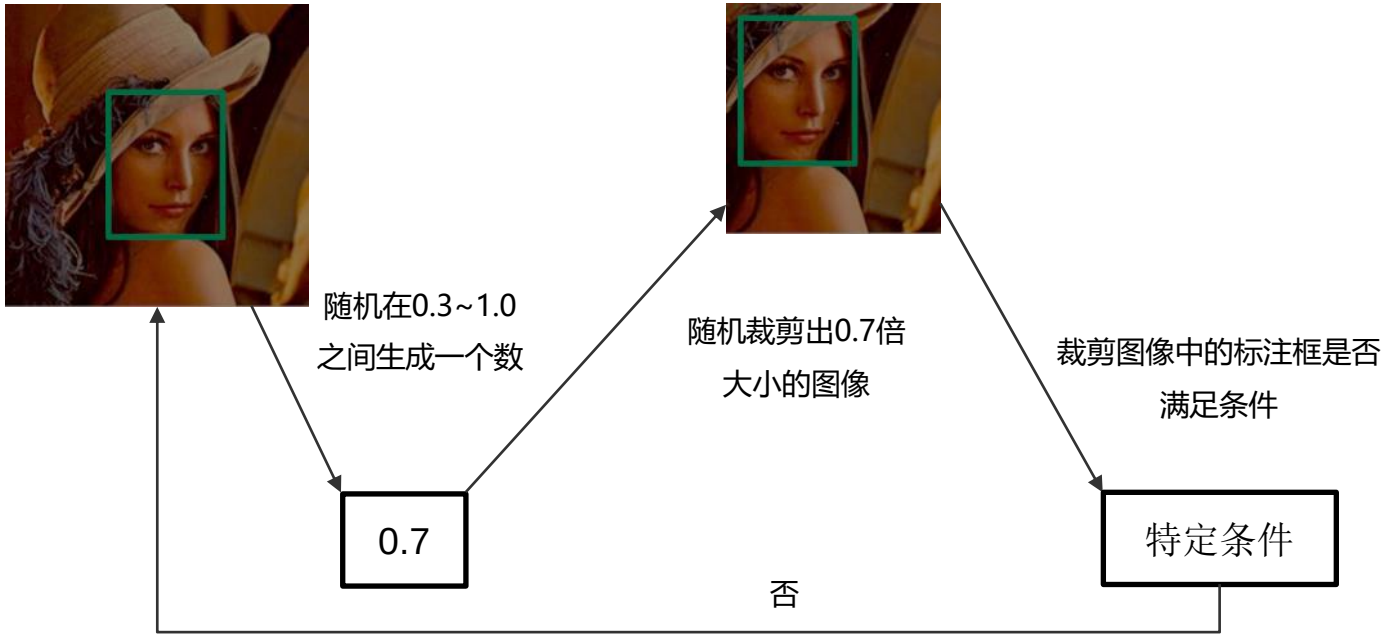
0.7

随机裁剪出0.7倍
大小的图像

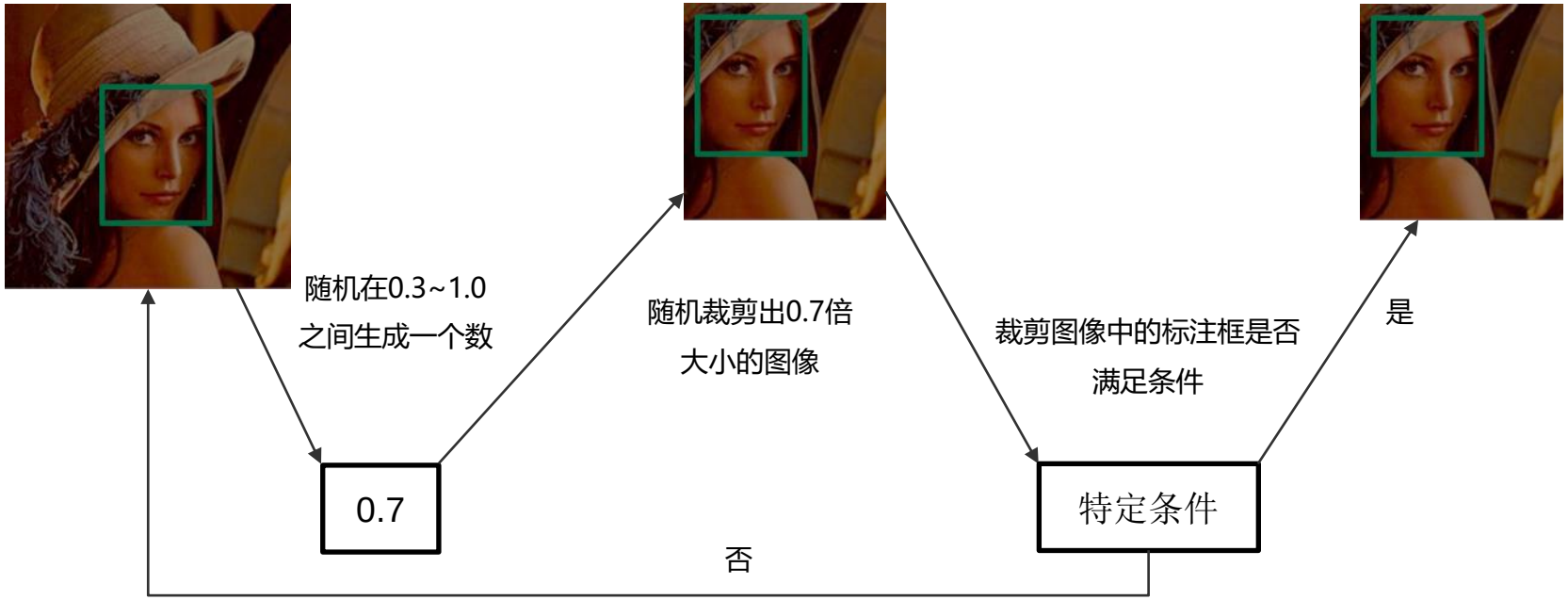
裁剪图像中的标注框是否
满足条件

特定条件

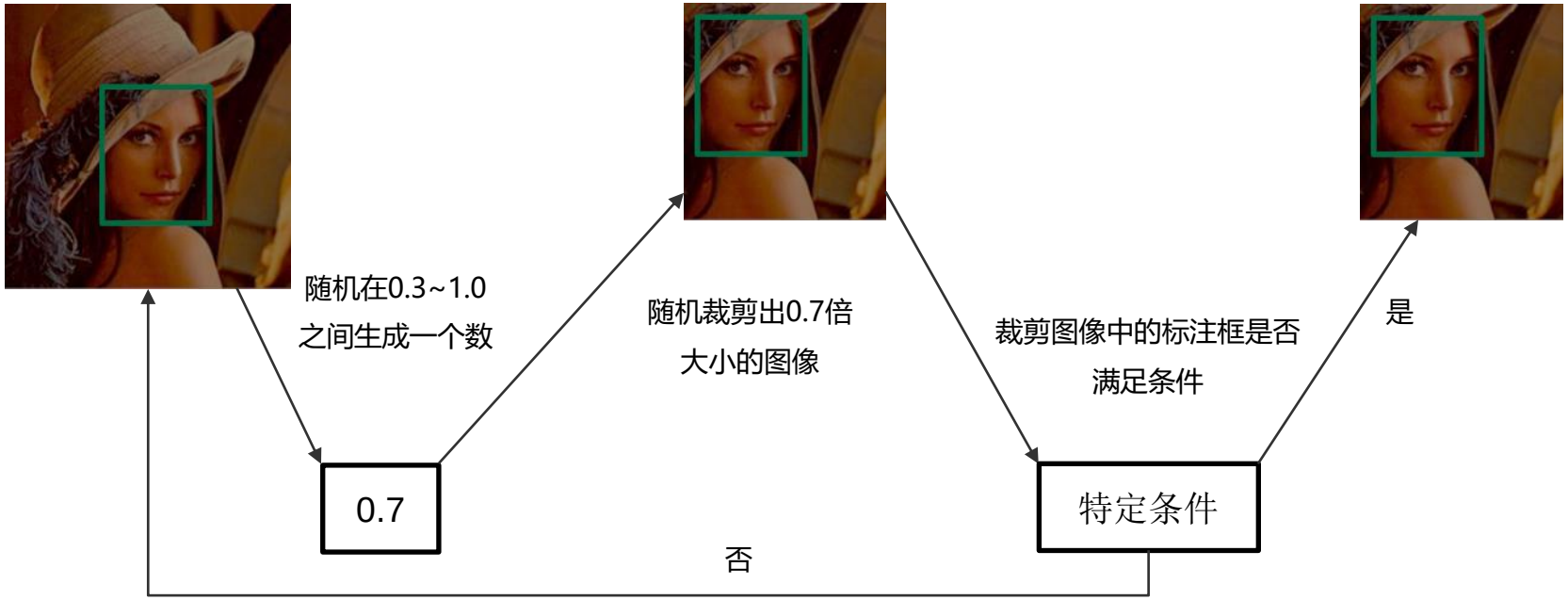
SSD检测算法：输入图像的随机裁剪



SSD检测算法：输入图像的随机裁剪

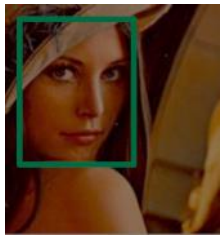


SSD检测算法：输入图像的随机裁剪



- 随机裁剪的数据增广操作以1/2的概率执行

SSD检测算法：输入图像的随机扩充

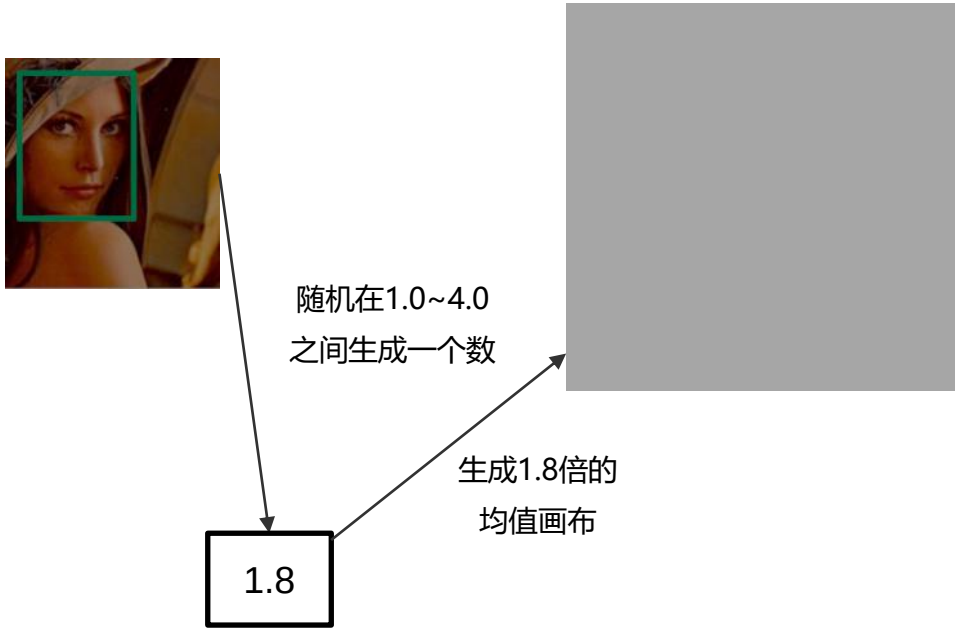


SSD检测算法：输入图像的随机扩充

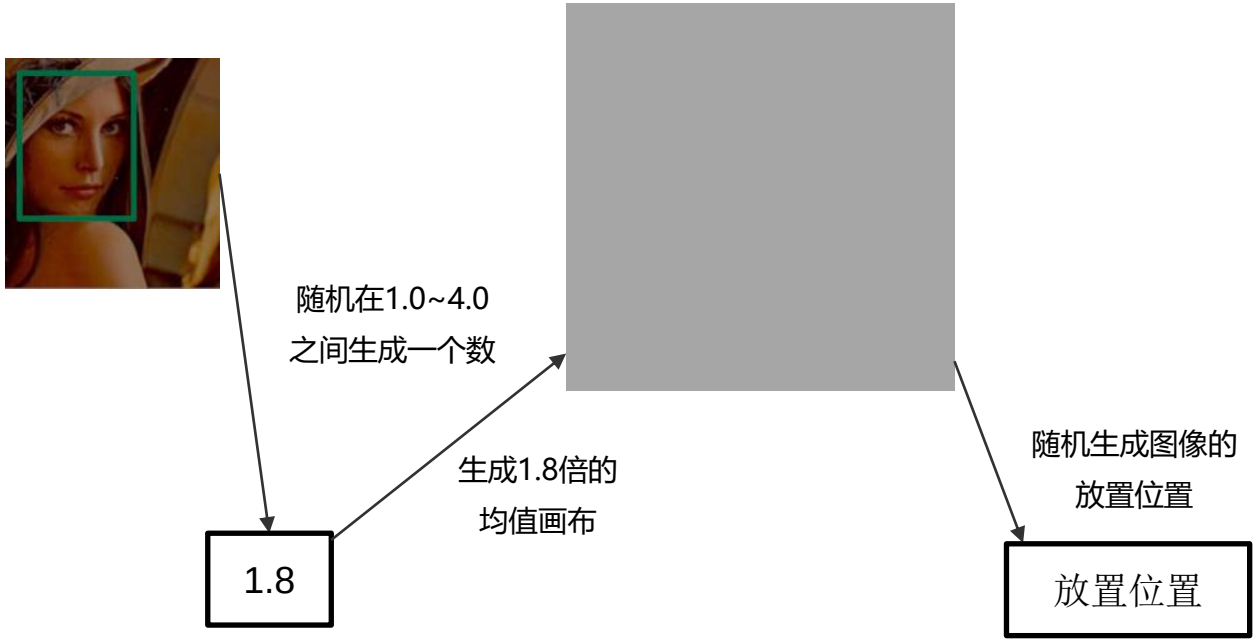
随机在1.0~4.0
之间生成一个数

1.8

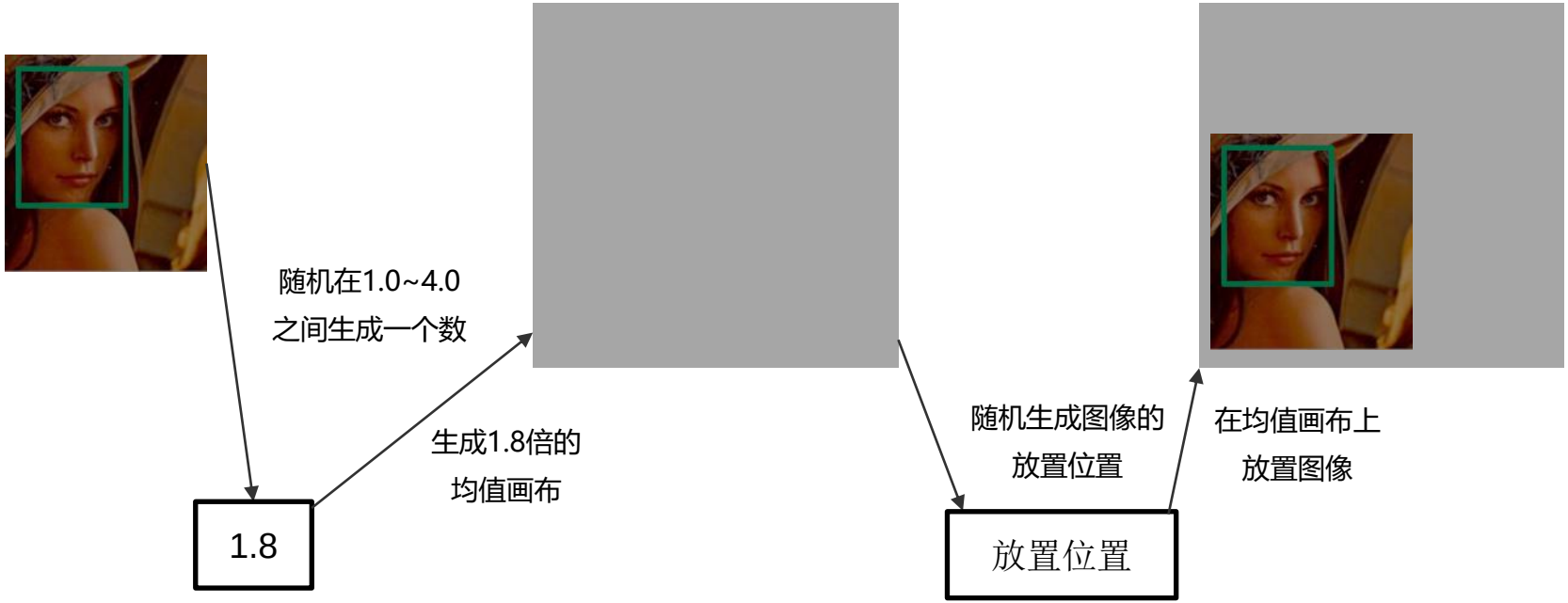
SSD检测算法：输入图像的随机扩充



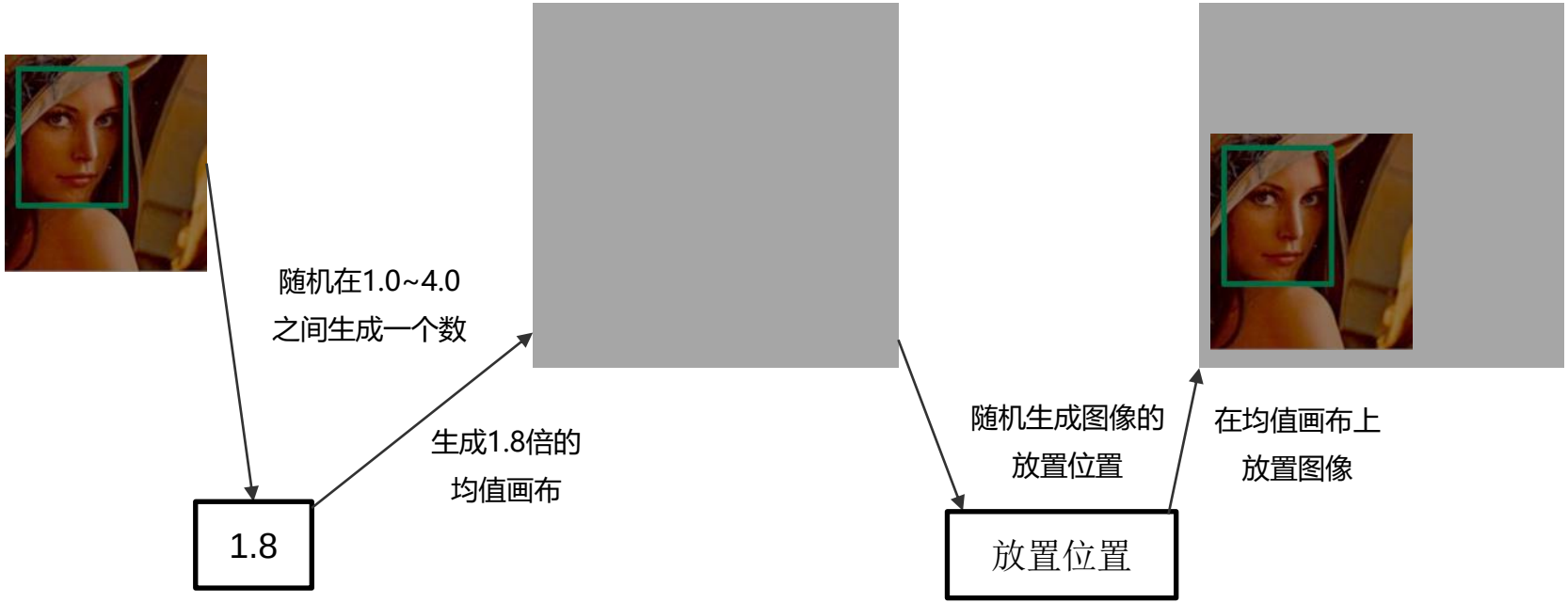
SSD检测算法：输入图像的随机扩充



SSD检测算法：输入图像的随机扩充



SSD检测算法：输入图像的随机扩充

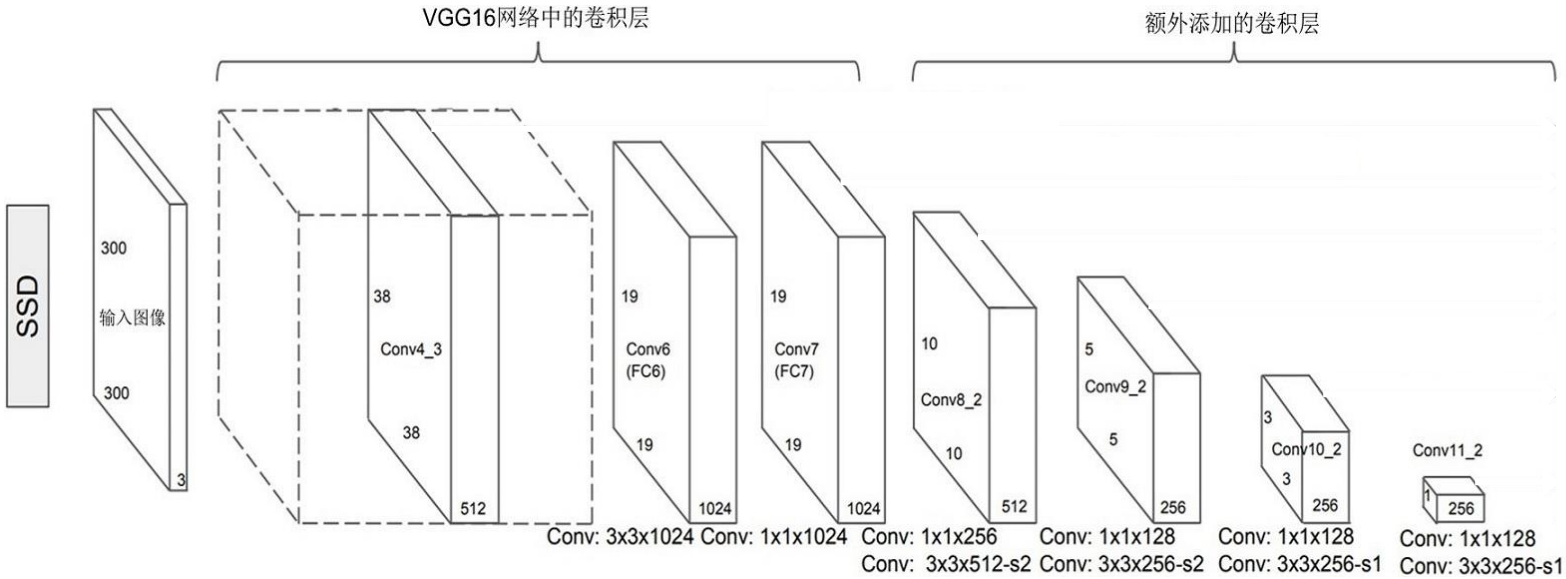


- 随机扩充的数据增广操作以1/2的概率执行

SSD检测算法：输入图像的随机水平翻转和缩放

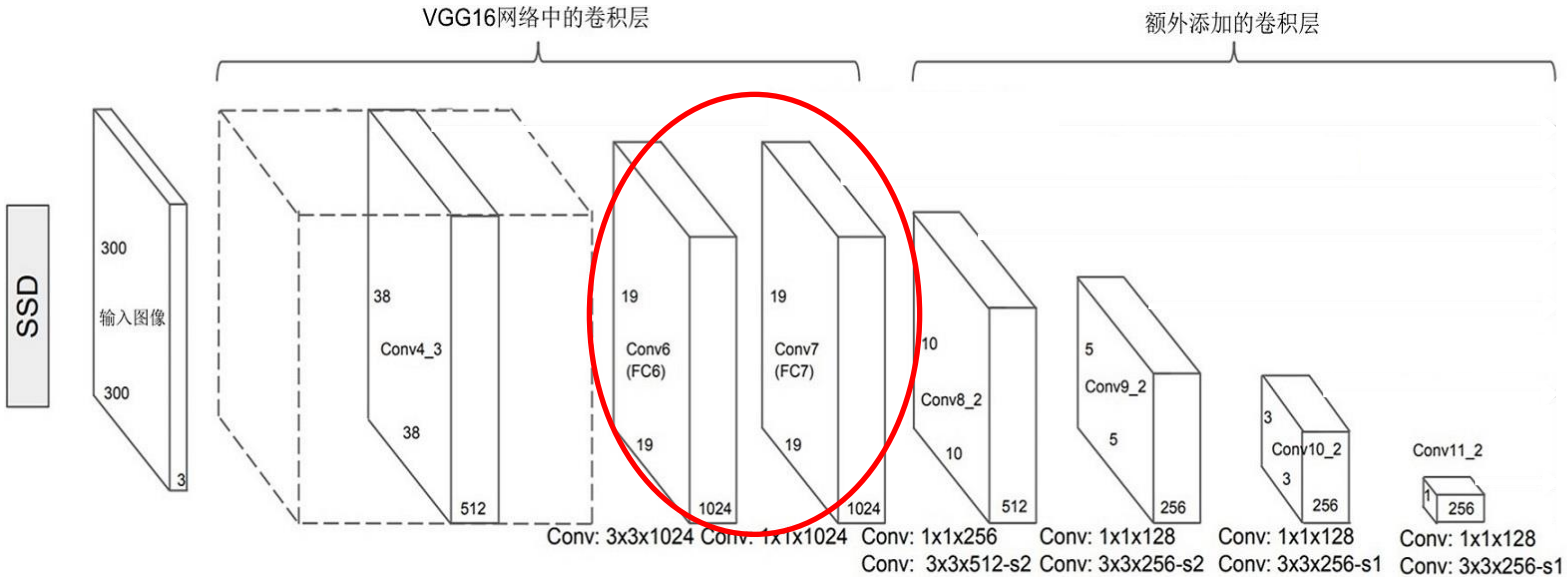
- 随机水平翻转：以1/2的概率对图像进行水平方向的翻转
- 图像缩放：把任意大小的图像缩放到300x300大小（**物体比例会改变**）

SSD检测算法：基础网络



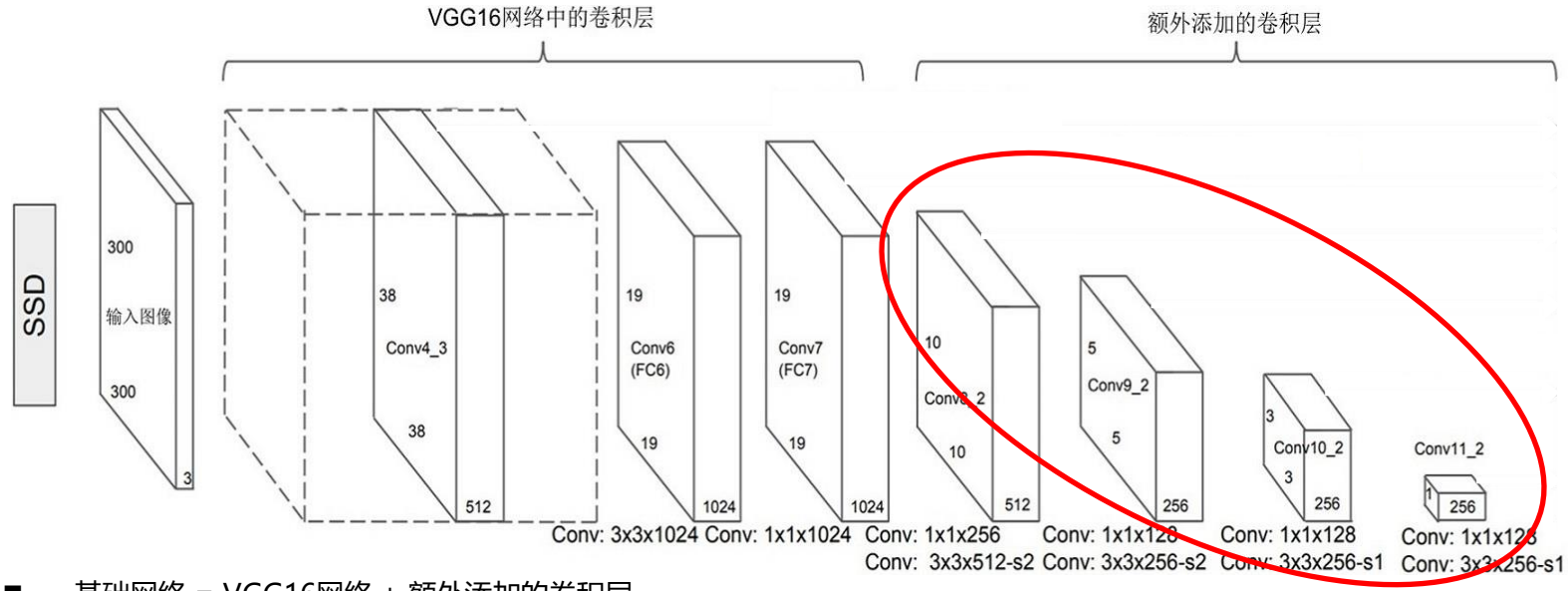
- 基础网络 = VGG16网络 + 额外添加的卷积层

SSD检测算法：基础网络



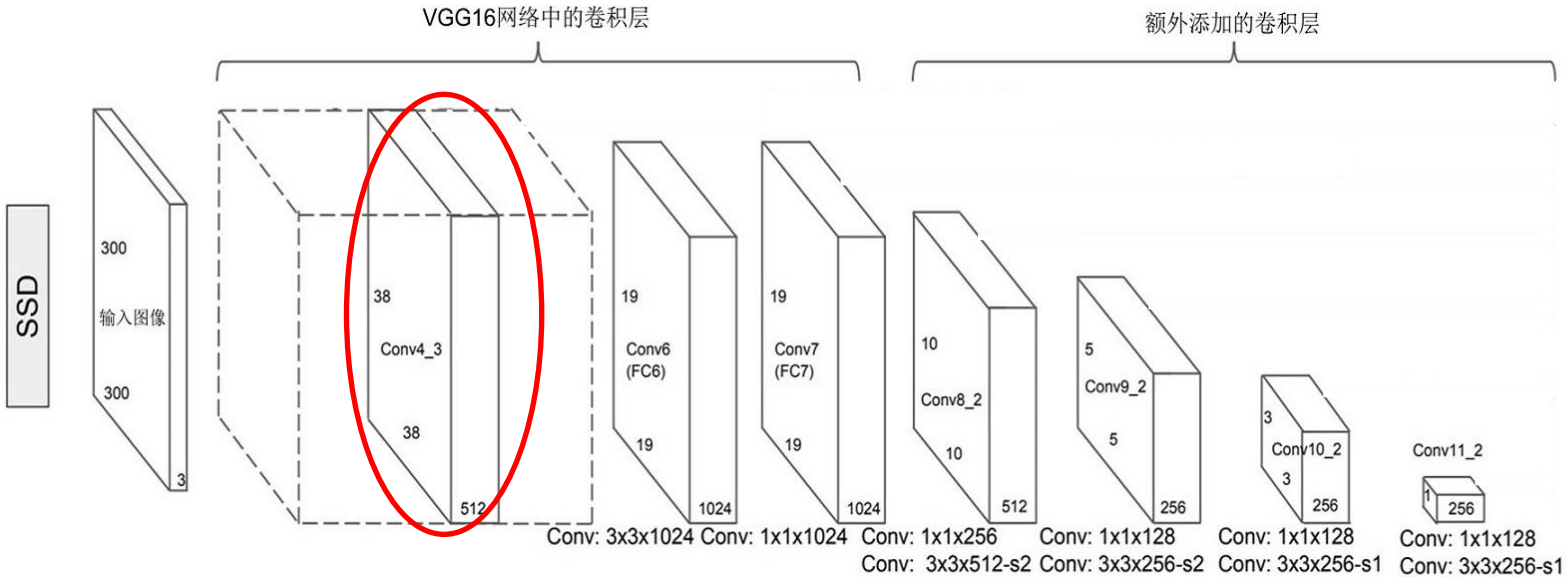
- 基础网络 = VGG16网络 + 额外添加的卷积层
- VGG16中的全连接层FC6和FC7通过权重采样变成卷积层Conv6和Conv7

SSD检测算法：基础网络



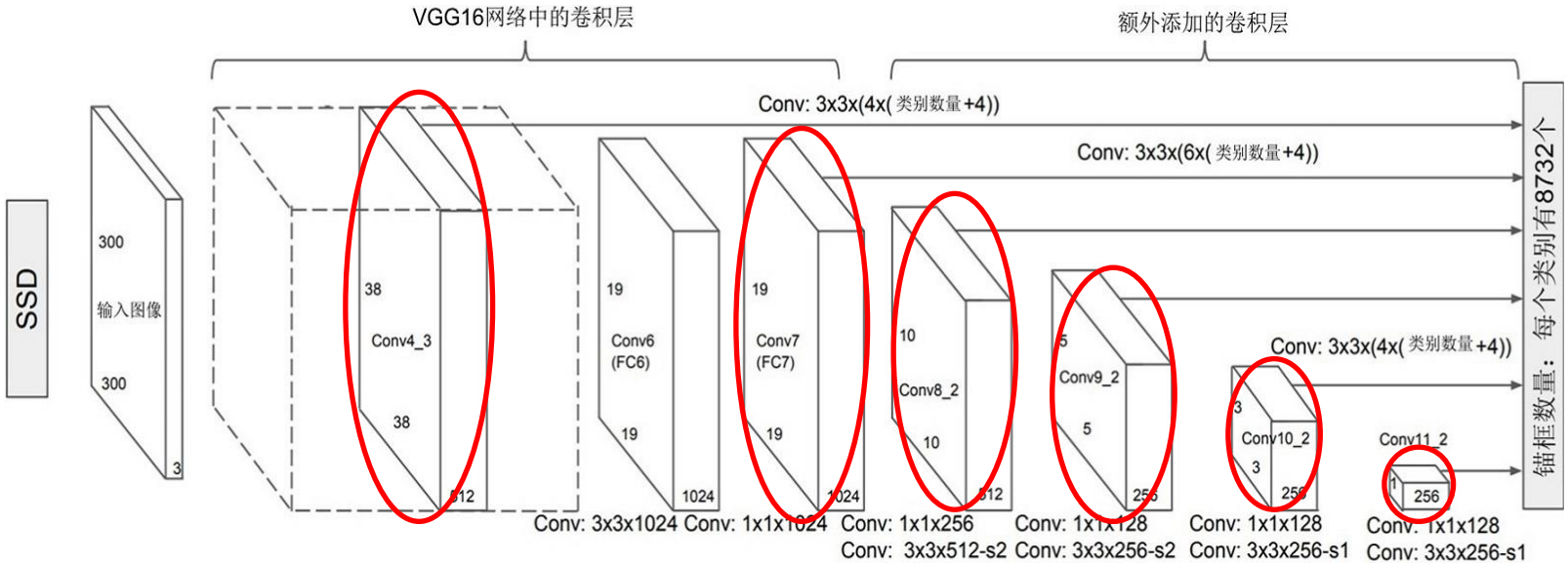
- 基础网络 = VGG16网络 + 额外添加的卷积层
- VGG16中的全连接层FC6和FC7通过权重采样变成卷积层Conv6和Conv7
- 额外添加了8个卷积层，每2个卷积层为1组，有着相同的下采样倍数

SSD检测算法：基础网络



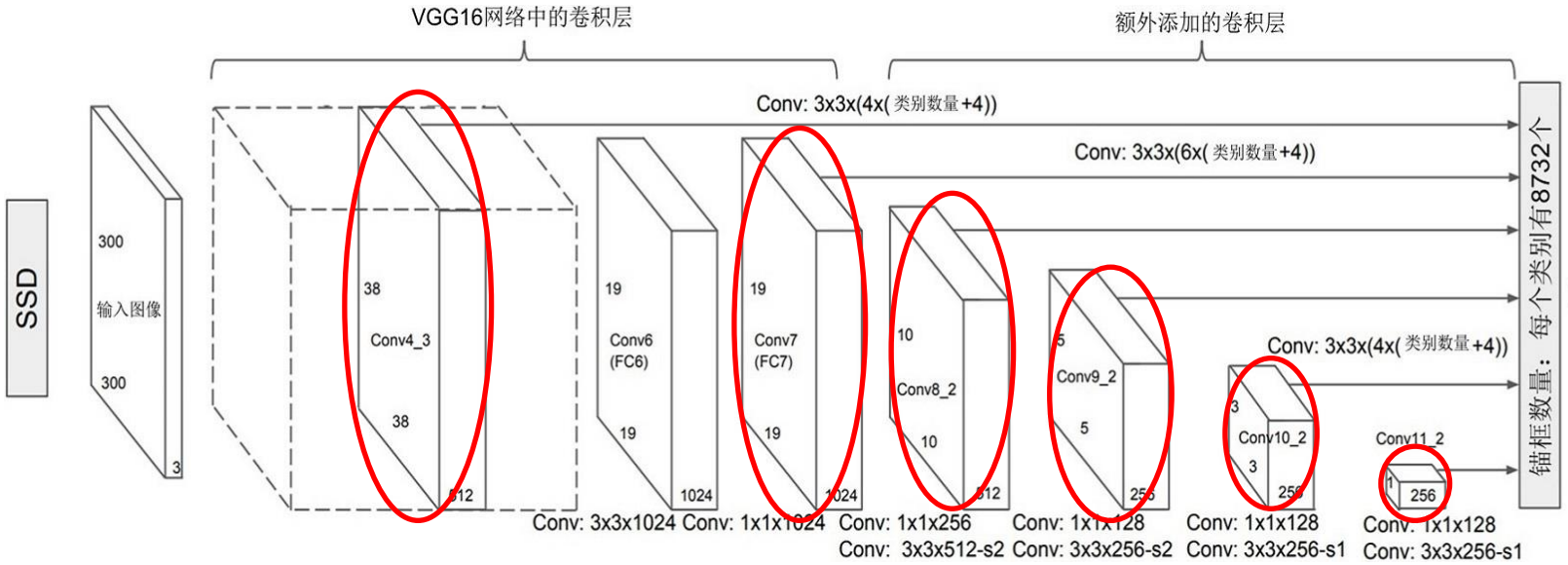
- 基础网络 = VGG16网络 + 额外添加的卷积层
- VGG16中的全连接层FC6和FC7通过权重采样变成卷积层Conv6和Conv7
- 额外添加了8个卷积层，每2个卷积层为1组，有着相同的下采样倍数
- VGG16中的Conv4_3卷积层的幅值太大，使用归一化操作把幅值变成20，并反传学习该参数

SSD检测算法：多检测层



- 6个检测层：Conv4_3、Conv_7、 Conv8_2、Conv9_2、Conv10_2、Conv11_2

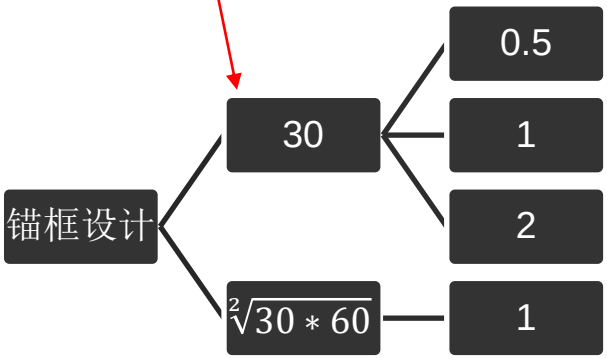
SSD检测算法：多检测层

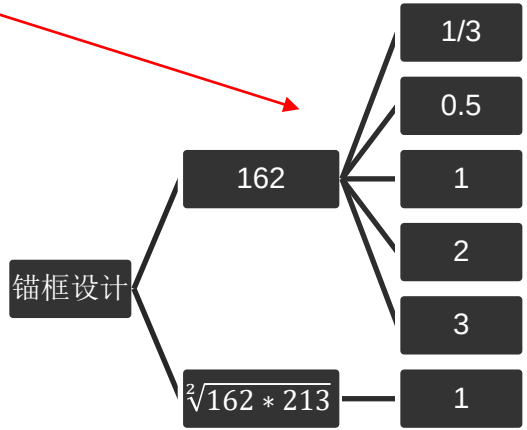


- 6个检测层：Conv4_3、Conv_7、 Conv8_2、Conv9_2、Conv10_2、Conv11_2
- 下采样倍数： 8、 16、 32、 64、 128、 256
- 锚框大小： [30, 60]、[60, 111]、[111, 162]、[162, 213]、[213, 264]、[264, 315]
- 锚框比例： [0.5,1,2]、[$\frac{1}{3}$,0.5,1,2,3]、[$\frac{1}{3}$,0.5,1,2,3]、[$\frac{1}{3}$,0.5,1,2,3]、 [0.5,1,2]、 [0.5,1,2]

SSD检测算法：多检测层的锚框设计

- 6个检测层：Conv4_3、Conv_7、 Conv8_2、Conv9_2、Conv10_2、Conv11_2
- 下采样倍数： 8 、 16、 32、 64、 128、 256
- 锚框大小： [30, 60]、[60, 111]、[111, 162]、[162, 213]、[213, 264]、[264, 315]
- 锚框比例： [0.5,1,2]、 $[\frac{1}{3},0.5,1,2,3]$、 $[\frac{1}{3},0.5,1,2,3]$、 $[\frac{1}{3},0.5,1,2,3]$、 [0.5,1,2]、 [0.5,1,2]

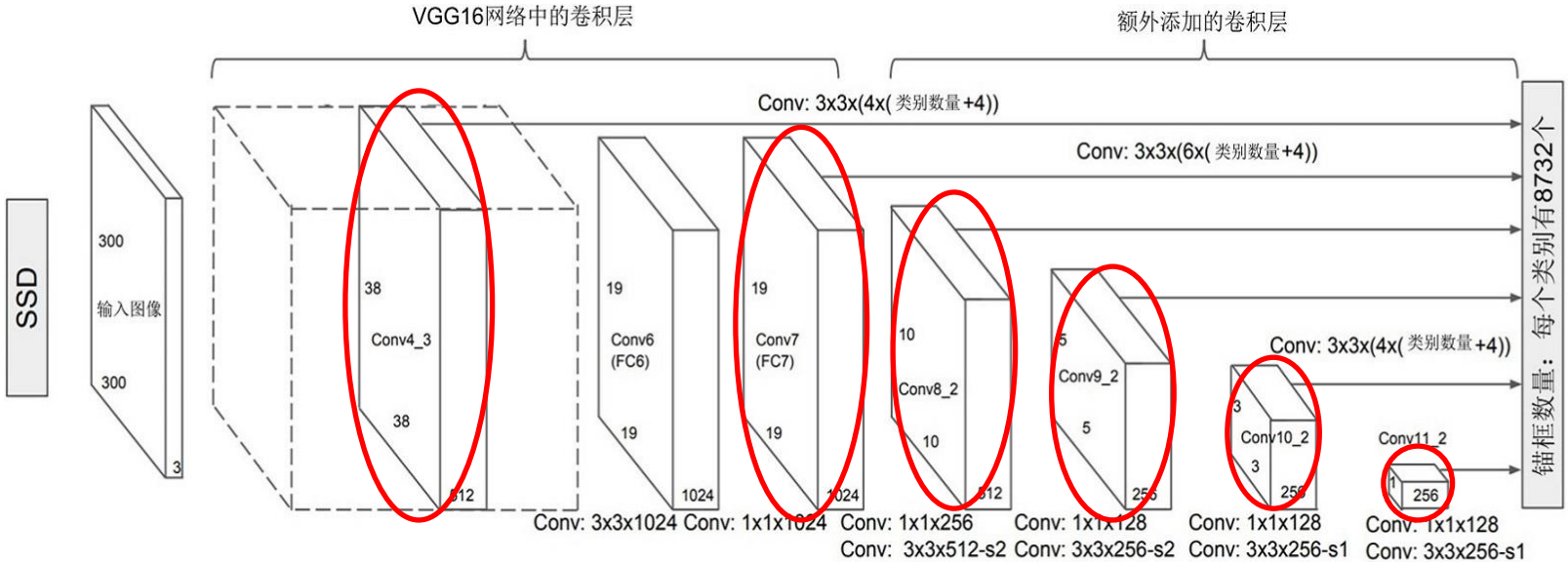




* SSD中生成锚框的规则：每个检测层的锚框大小都有两个尺度，即[S_{min}, S_{max}]

S_{min}生成与所有锚框比例结合生成锚框，S_{max}只跟1:1的锚框比例结合生成锚框

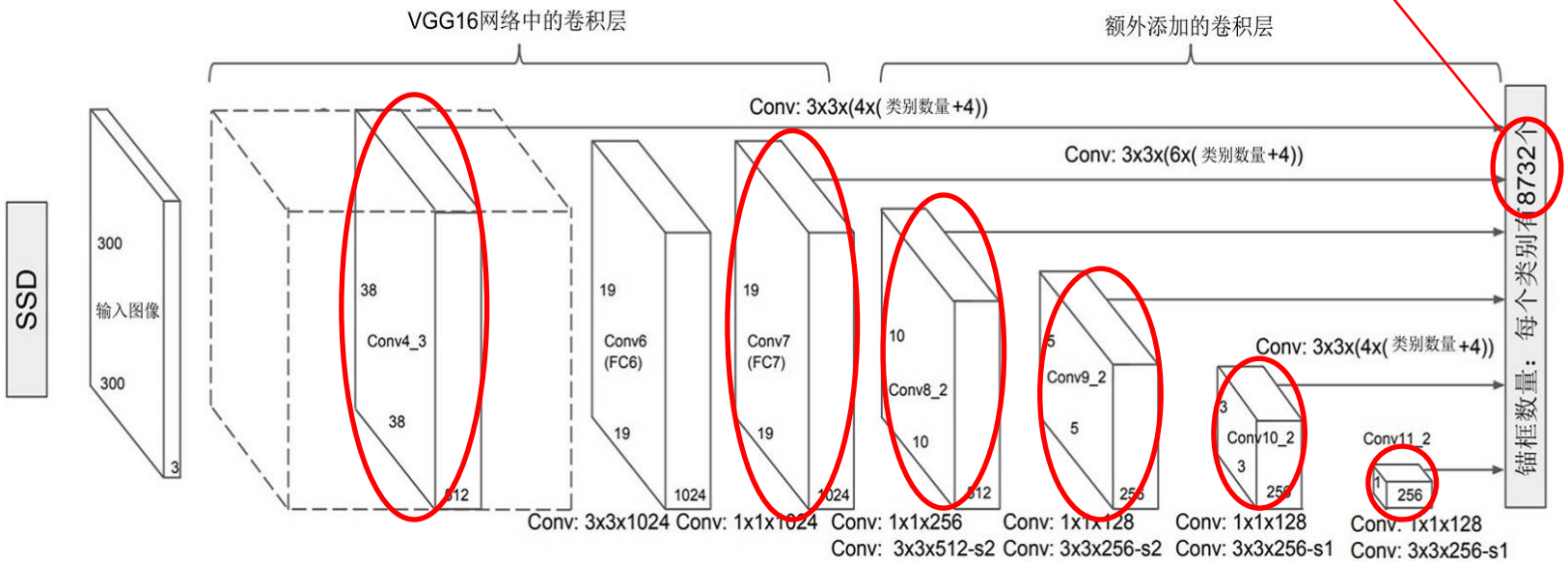
SSD检测算法：多检测层



- 6个检测层：Conv4_3、Conv_7、 Conv8_2、Conv9_2、Conv10_2、Conv11_2
- 下采样倍数： 8、 16、 32、 64、 128、 256
- 锚框大小： [30, 60]、[60, 111]、[111, 162]、[162, 213]、[213, 264]、[264, 315]
- 锚框比例： [0.5,1,2]、[$\frac{1}{3}$,0.5,1,2,3]、[$\frac{1}{3}$,0.5,1,2,3]、[$\frac{1}{3}$,0.5,1,2,3]、 [0.5,1,2]、 [0.5,1,2]
- 锚框个数： 4、 6、 6、 6、 4、 4

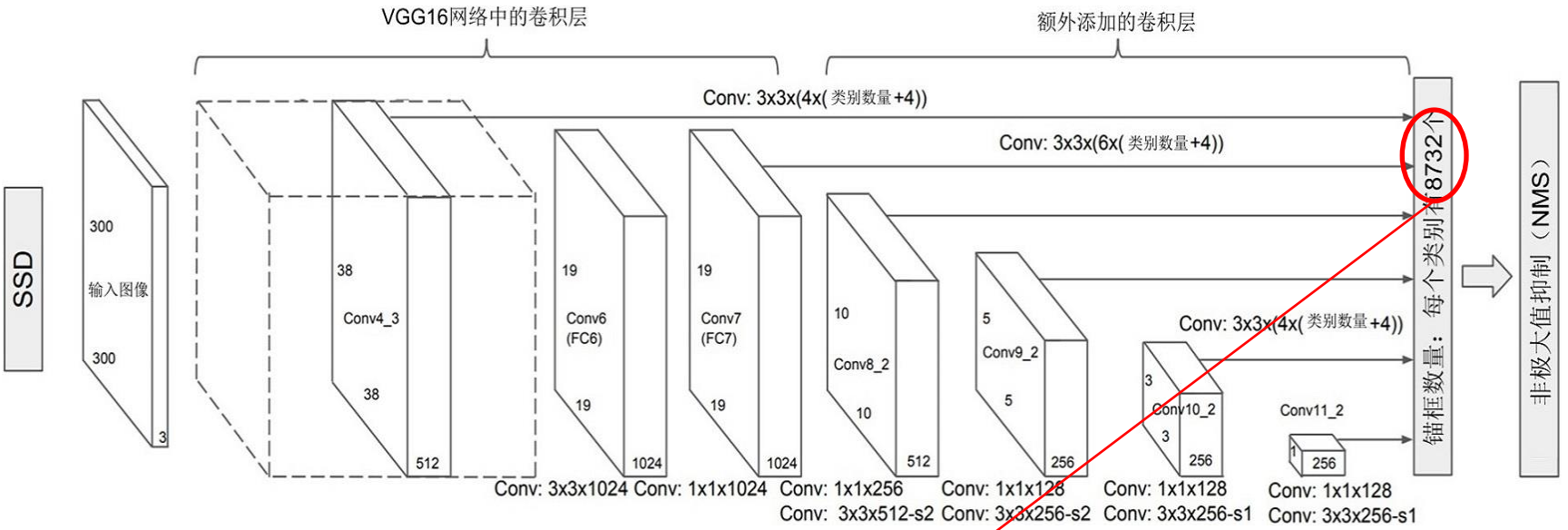
SSD检测算法：多检测层

38x38x4+19x19x6+10x10x6+5x5x6+3x3x4+1x1x4 = 8732



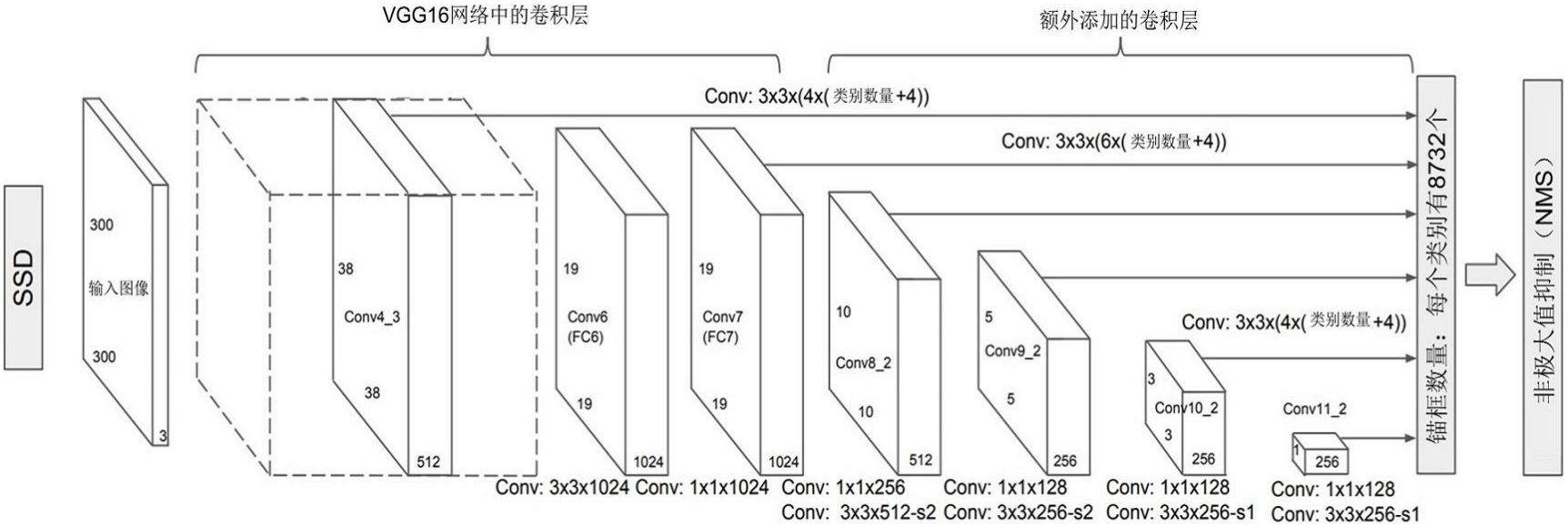
- 6个检测层：Conv4_3、Conv_7、 Conv8_2、Conv9_2、Conv10_2、Conv11_2
- 下采样倍数： 8、 16、 32、 64、 128、 256
- 锚框大小： [30, 60]、[60, 111]、[111, 162]、[162, 213]、[213, 264]、[264, 315]
- 锚框比例： [0.5,1,2]、[1,0.5,1,2,3]、[1,0.5,1,2,3]、[1,0.5,1,2,3]、 [0.5,1,2]、 [0.5,1,2]
- 锚框个数： 4、 6、 6、 6、 4、 4

SSD检测算法：难负样本挖掘

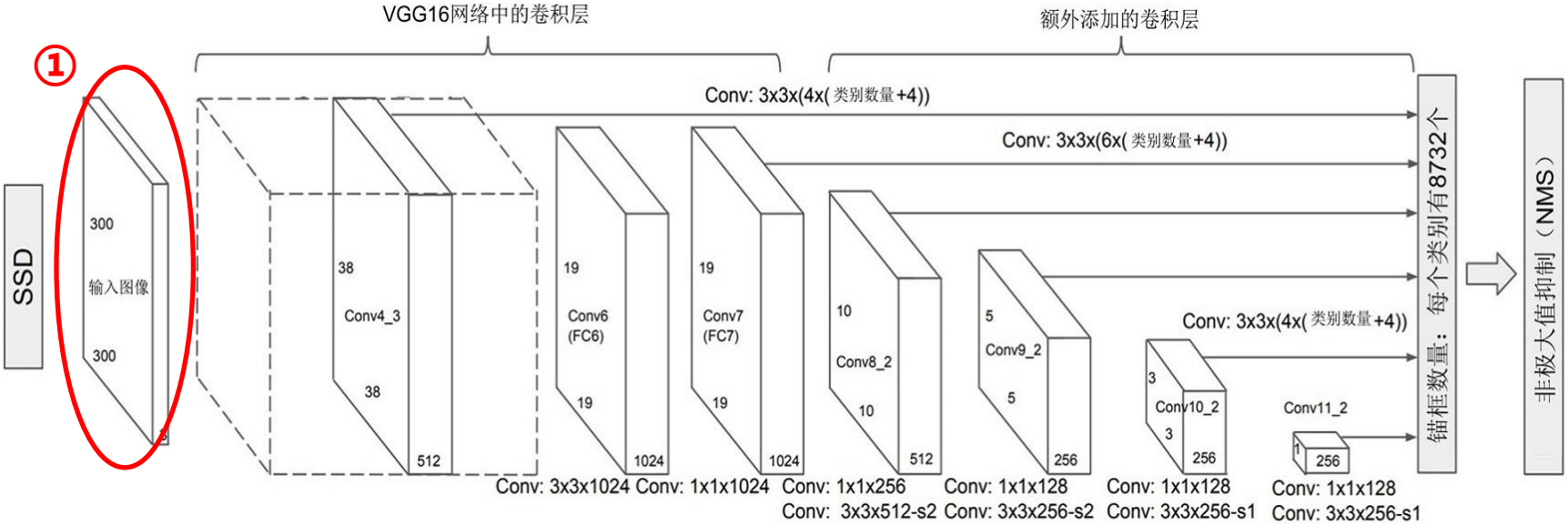


- 锚框匹配：①正样本：最佳匹配或IoU≥0.5；②负样本：IoU < 0.5
- 匹配结果：把锚框划分为正负样本之后，一般只有几十个锚框是正样本，其他都是负样本
- 比例失衡：正样本/负样本 = 约50个正样本/约8700个负样本 ≈ 1/174
- 导致问题：极度不平衡的正负样本比例，让网络训练朝着错误的方向优化
- RPN的解决方案：正样本保持不变，随机选择一小部分负样本出来，其他负样本忽略
- SSD的解决方案：正样本保持不变，根据分类误差损失值对负样本进行降序排序，选出正样本数量的3倍，其他忽略

SSD检测算法总结：主要贡献

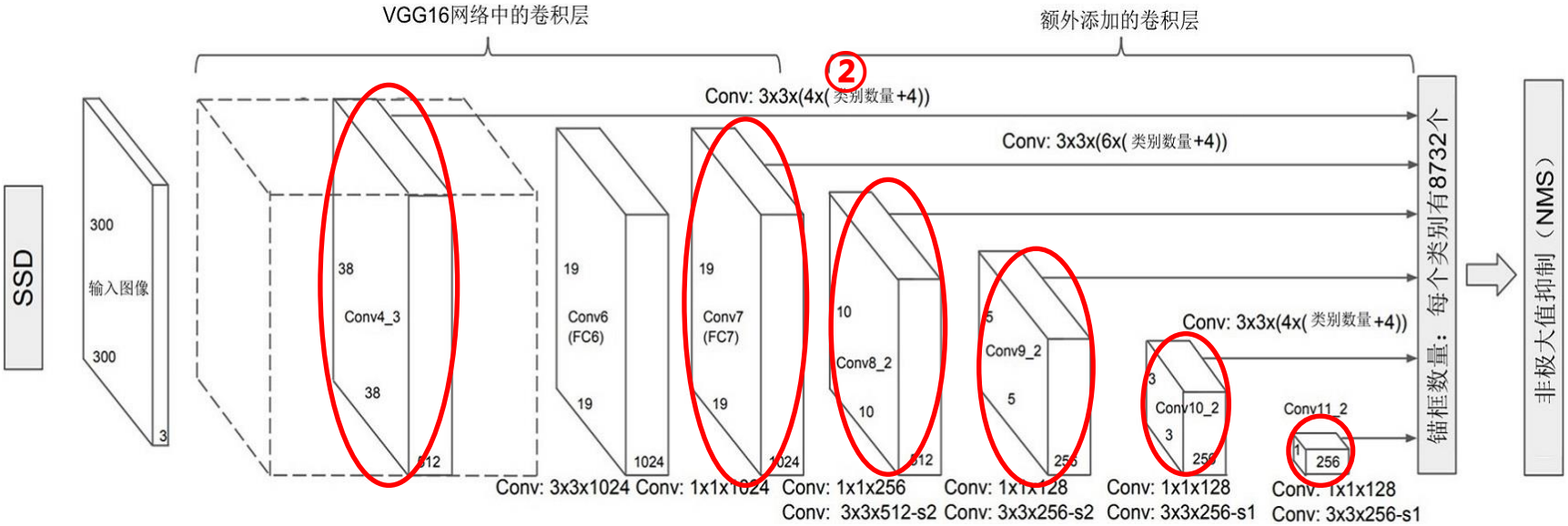


SSD检测算法总结：主要贡献



① 数据增广：RPN只是用了水平翻转和静态多尺度训练，而SSD设计了随机性更强、更丰富的数据增广，例如颜色抖动、随机裁剪+扩充（动态多尺度训练）

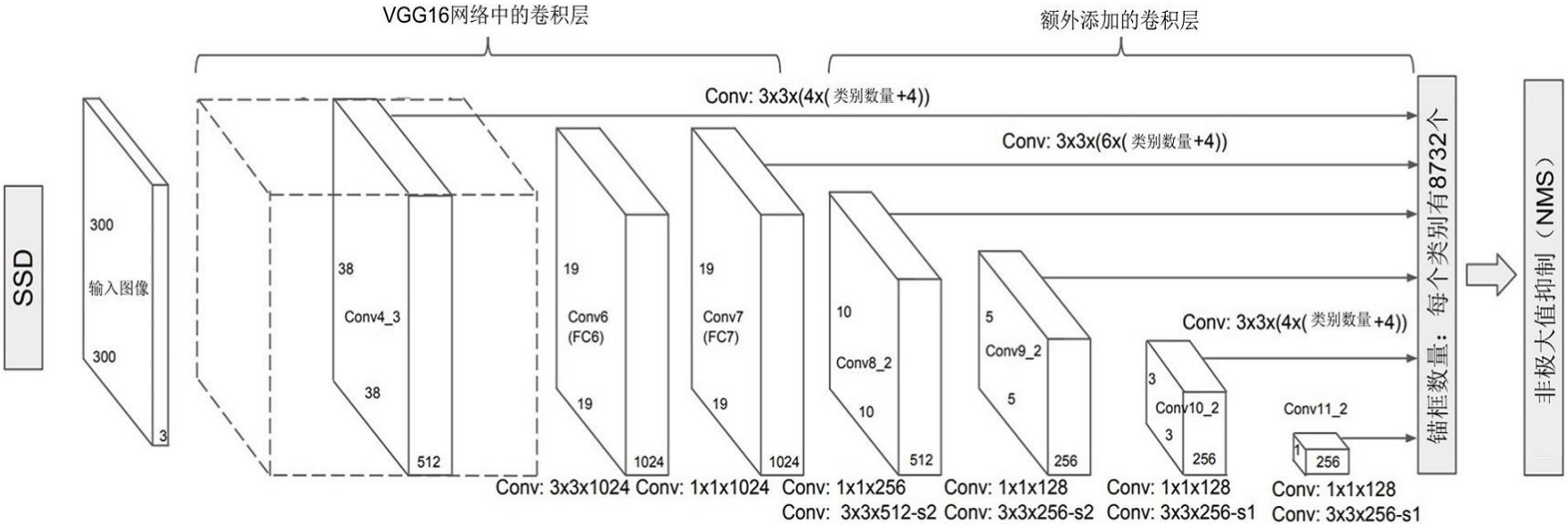
SSD检测算法总结：主要贡献



① 数据增广：RPN只是用了水平翻转和静态多尺度训练，而SSD设计了随机性更强、更丰富的数据增广，例如颜色抖动、随机裁剪+扩充（动态多尺度训练）

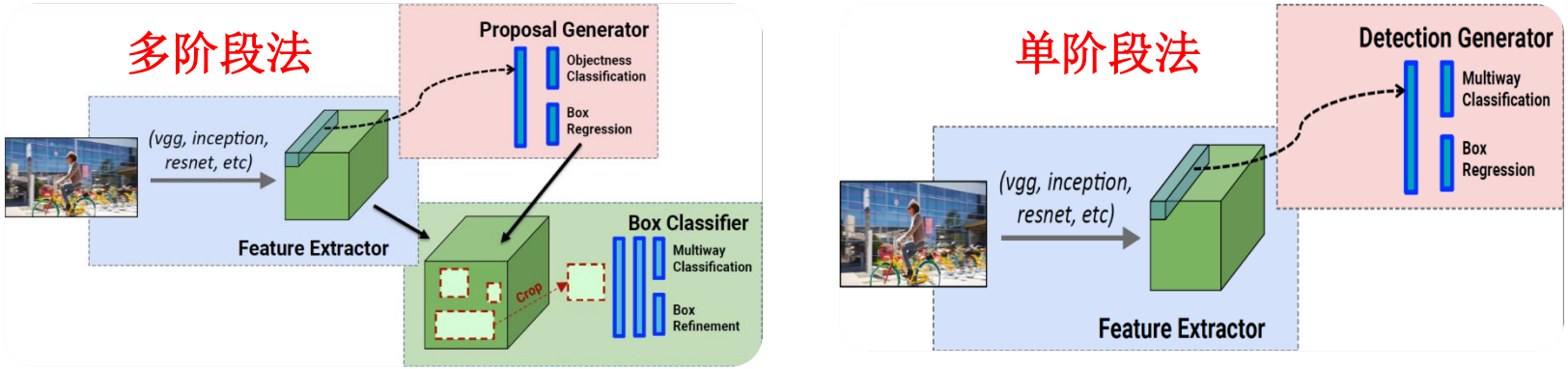
② 多检测层：RPN是在单个检测层上关联锚框进行检测，而SSD在多个检测层上关联适当的锚框进行检测，后续算法基本都采用多检测层这种设计

SSD检测算法总结：集大成者



- 单阶段检测算法的集大成者，保持几十FPS的速度，精度跟多阶段检测算法差不多
- 所有代码都是基于Caffe用C++实现，方便工程部署，很多实际产品在使用
- 后续的单阶段检测算法大多都是基于SSD进行改进的

基于锚框的单阶段检测算法：RetinaNet



两类检测算法的对比

- 多阶段法：高精度，但速度较慢
- 单阶段法：速度快，但是准确率不如前者
- 单阶段法精度差的主要原因之一：**正负样本数量的极度不均衡**

Focal Loss for Dense Object Detection, **Best Student Paper Award**, ICCV 2017

基于锚框的单阶段检测算法：RetinaNet

- SSD使用**难负样本挖掘**来解决正负样本比例极度不平衡的问题，而难负样本挖掘有两个问题：

① 样本利用不充分：只使用了一小部分较难的负样本，大部分负样本都没有使用

② 挖掘难以控制：利用正样本数量的3倍来挖负样本，有时挖太多，有时挖太少

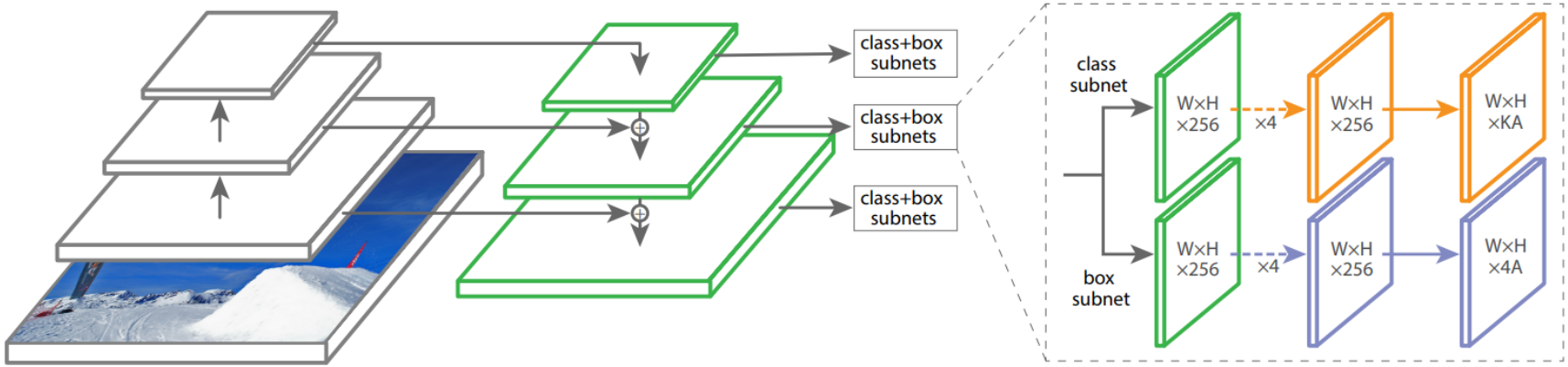
- RetinaNet通过修改标准交叉熵损失，提出了**focal loss损失函数**：

① 通过减少易分类样本的权重，使得模型在训练时更专注于难分类的样本，从而充分地利用所有样本

② 使得单阶段法检测器可以达到多阶段法检测器准确率，同时不影响原有的速度

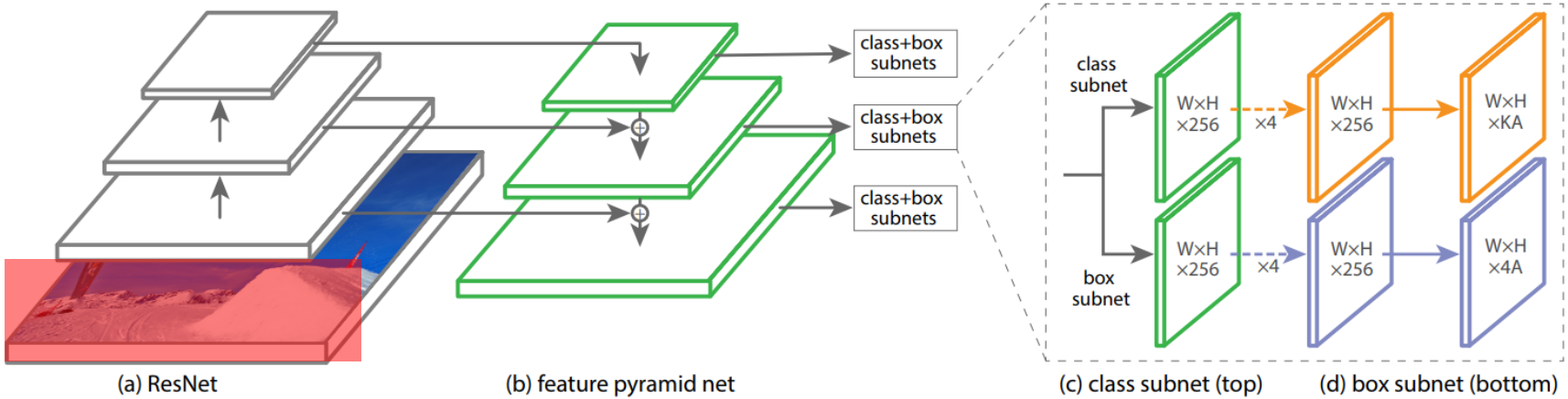
Focal Loss for Dense Object Detection, **Best Student Paper Award**, ICCV 2017

RetinaNet检测算法：整体框架



(a) ResNet (b) feature pyramid net (c) class subnet (top) (d) box subnet (bottom)

RetinaNet检测算法：输入图像



■ RetinaNet对输入图像的操作：

① 随机水平翻转

② 单尺度训练：图像短边等比例缩放至800，且长边不超过1333

③ 多尺度训练：图像短边等比例缩放至[640, 672, 704, 736, 768, 800]，且长边不超过1333

■ SSD对输入图像的操作：

① 颜色抖动

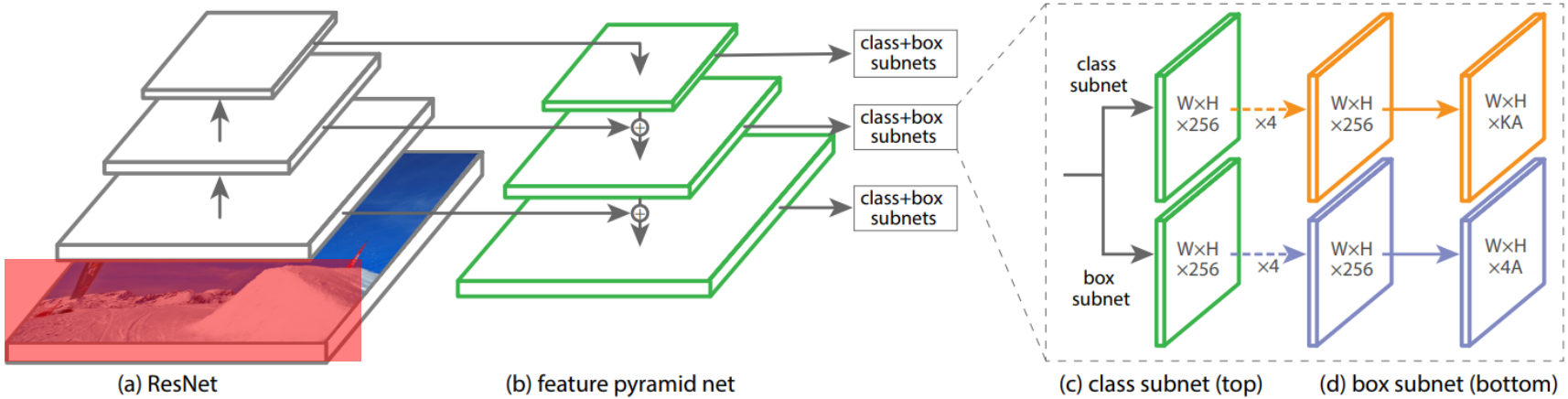
② 随机裁剪

③ 随机扩充

④ 随机水平翻转

⑤ 不等比例地缩放至300x300或512x512

RetinaNet检测算法：输入图像



■ RetinaNet对输入图像的操作 VS ■ SSD对输入图像的操作：

① 输入大小：RetinaNet是~800x1333，而SSD是300x300或512x512

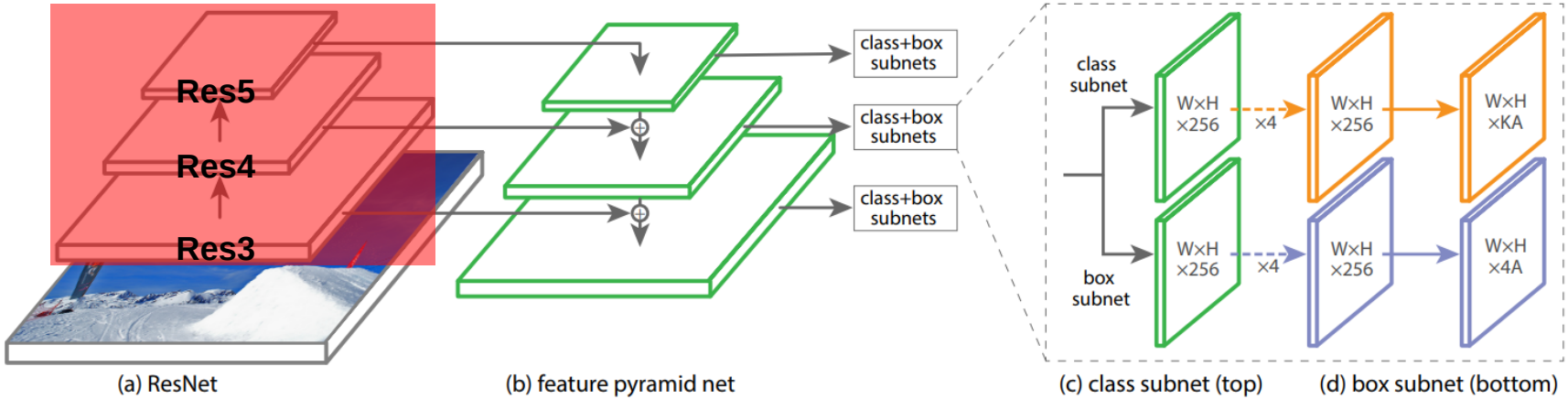
② 物体大小：RetinaNet输入大，物体整体较大，检测难度低，而SSD则相反

③ 多尺度方式：RetinaNet是静态多尺度 (6选1)，而SSD是动态多尺度 (随机扩充+裁剪)

④ 颜色抖动： RetinaNet没有使用，而SSD使用了，在某些任务某些数据上有一定效果

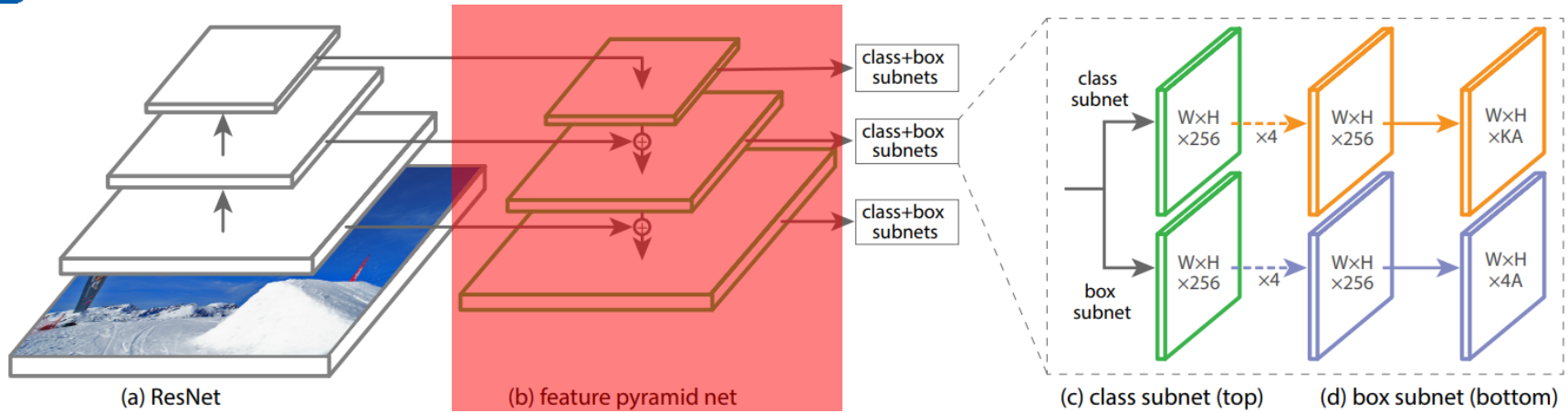
⑤ 迭代次数：数据增广越多，迭代次数就要越多，一般是数据增广+1，迭代次数x2

RetinaNet检测算法：基础网络



- ResNet50/101
- 所有的ResNet网络，都有5个模块组成
- 5个模块：Res1、 Res2、 Res3、 Res4、 Res5
- 下采样率：2^1=2、2^2=4、 2^3=8、 2^4=16、2^5=32
- RetinaNet选取3个模块来作为初始的检测层
- Res3、 Res4、 Res5

RetinaNet检测算法：特征金字塔



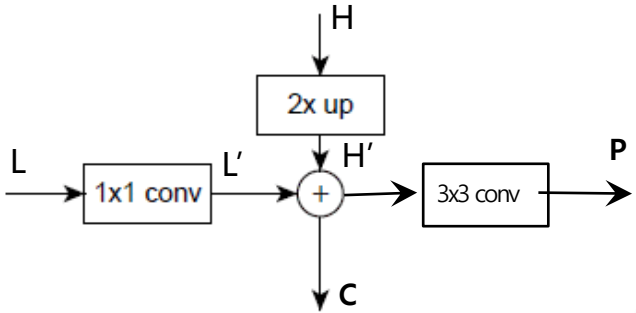
- 利用特征金字塔强化检测层的特征：

① 低层特征L经过1x1卷积，得到L′

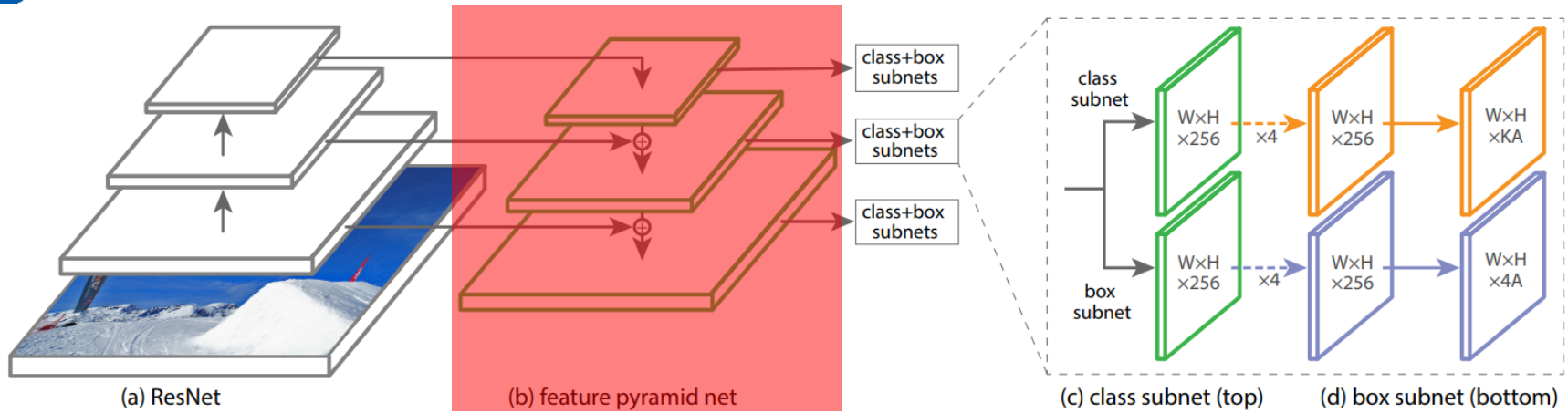
② 高层特征H经过上采样，得到H′

③ 特征融合C = L′ + H′

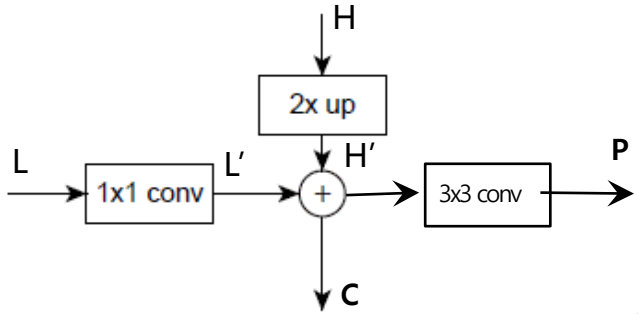
④ 融合特征经过3x3卷积，得到P



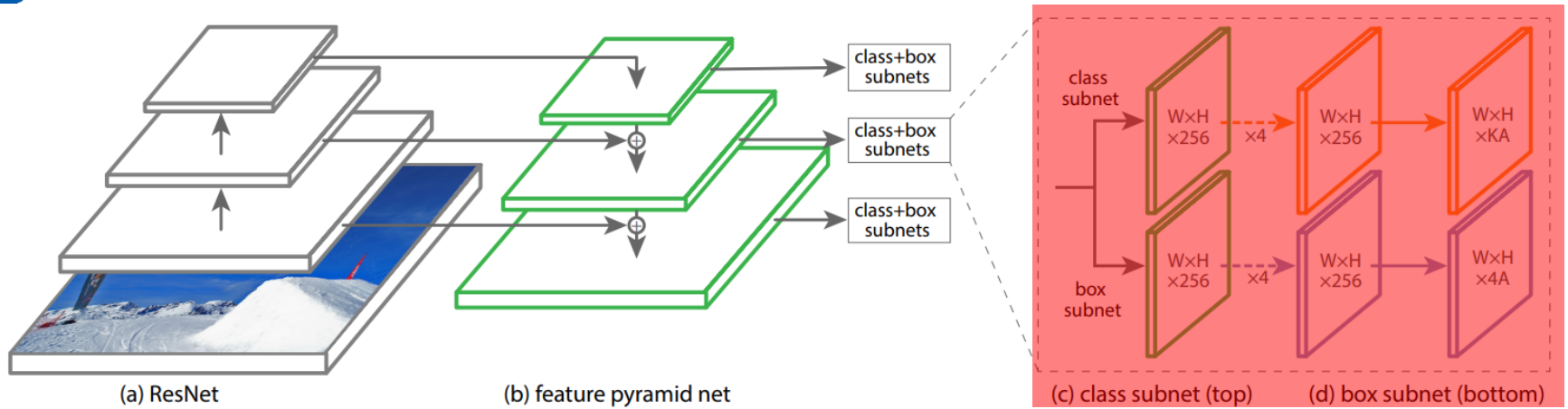
RetinaNet检测算法：特征金字塔



- C是下一轮融合的高层输入，P是用于检测的特征
- 上采样用的是最近邻插值
- 卷积层后不跟BN、ReLU
- 利用FPN对初始的检测层进行强化得到P3，P4，P5
- 再从P5后面使用下采样率为2的卷积层生成P6，P7



RetinaNet检测算法：检测子网络



- 检测子网络变深：

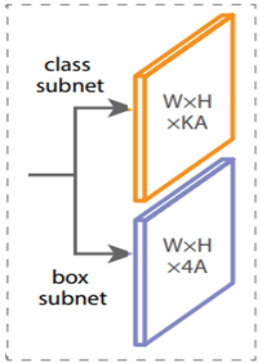
① 分类分支加了4个卷积层

② 回归分支加了4个卷积层

- 检测子网络共享

① SSD中，每个检测层都有一个检测头

② RetinaNet中，所有检测层共用一个检测头



SSD的检测子网络

RetinaNet检测算法：Focal Loss

- 二分类的交叉熵 (cross entropy, CE) 损失函数：

$$\mathrm{CE}(p, y) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1-p) & \text{otherwise.} \end{cases}$$

RetinaNet检测算法：Focal Loss

- 二分类的交叉熵 (cross entropy, CE) 损失函数：

$$\mathrm{CE}(p, y) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1-p) & \text{otherwise.} \end{cases}$$

- 定义P_t为：

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{otherwise} \end{cases}$$

RetinaNet检测算法：Focal Loss

- 二分类的交叉熵 (cross entropy, CE) 损失函数：

$$\mathrm{CE}(p, y) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1-p) & \text{otherwise.} \end{cases}$$

- 定义P_t为：

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{otherwise} \end{cases}$$

- 则可以简化为：

$$\mathrm{CE}(p, y) = \mathrm{CE}(p_t) = -\log(p_t)$$

RetinaNet检测算法：Focal Loss

- 二分类的交叉熵 (cross entropy, CE) 损失函数：

$$\mathrm{CE}(p, y) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1-p) & \text{otherwise.} \end{cases}$$

- 定义P_t为：

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{otherwise} \end{cases}$$

- 则可以简化为：

$$\mathrm{CE}(p, y) = \mathrm{CE}(p_t) = -\log(p_t)$$

P_t 越接近于1，表示分类分的越正确

RetinaNet检测算法：Focal Loss

- 二分类的交叉熵 (cross entropy, CE) 损失函数：

$$\mathrm{CE}(p, y) = \mathrm{CE}(p_t) = -\log(p_t)$$

- Focal Loss为：

$$\mathrm{FL}(p_t) = -(1 - p_t)^{\gamma} \log(p_t).$$

RetinaNet检测算法：Focal Loss

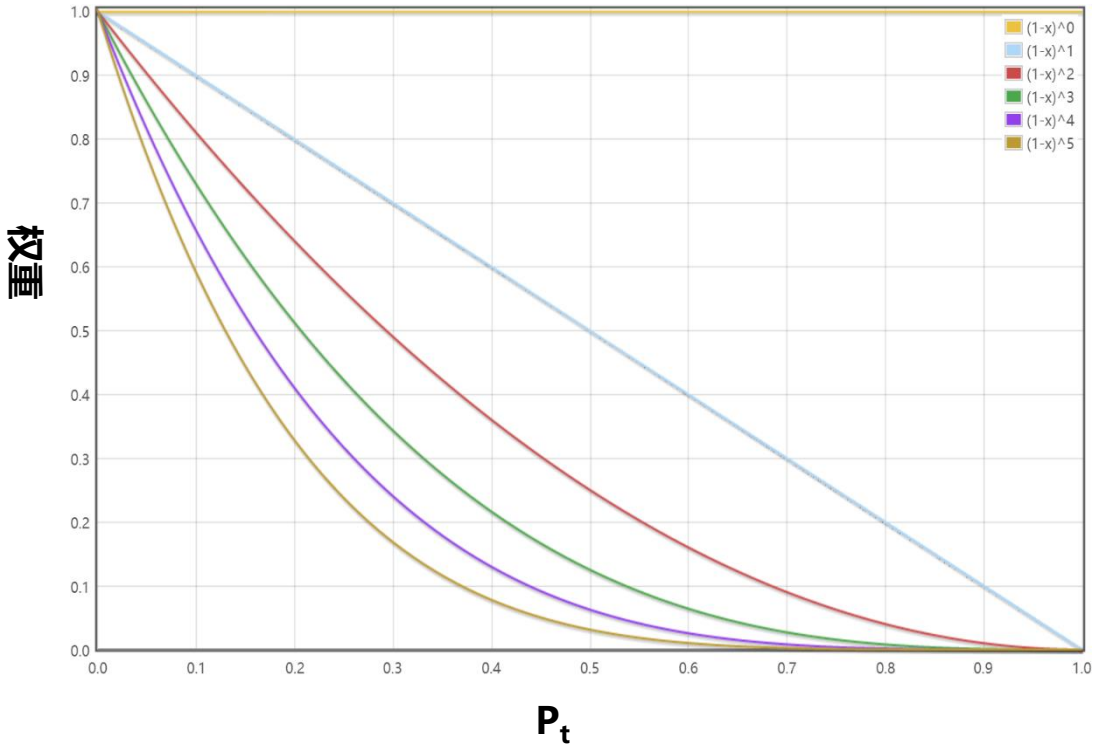
- 二分类的交叉熵 (cross entropy, CE) 损失函数：

$$\mathrm{CE}(p, y) = \mathrm{CE}(p_t) = -\log(p_t)$$

- Focal Loss为：

$$\mathrm{FL}(p_t) = -(1 - p_t)^{\gamma} \log(p_t).$$

RetinaNet检测算法：Focal Loss



$$(1-p_t)^{\gamma}$$

$$\gamma = 0,1,2,3,4,5$$

P_t 越接近于1，

权重越小

RetinaNet检测算法：Focal Loss

- 二分类的交叉熵 (cross entropy, CE) 损失函数：

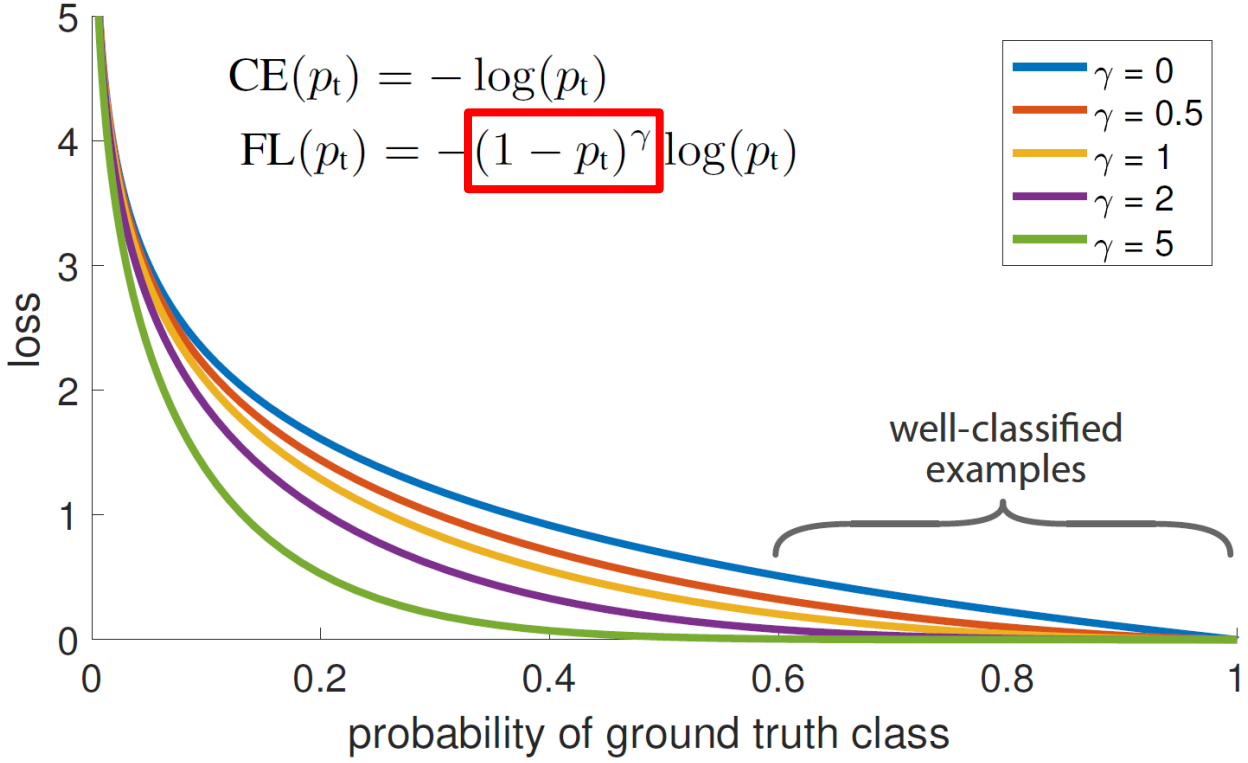
$$\mathrm{CE}(p, y) = \mathrm{CE}(p_t) = -\log(p_t)$$

- Focal Loss为：

$$\mathrm{FL}(p_t) = -(1 - p_t)^{\gamma} \log(p_t)$$

- **P_t 越接近于1，表示分类分的越正确**
- **分类分的越正确，加权的权重越小**

RetinaNet检测算法：Focal Loss



RetinaNet检测算法：Focal Loss和难样本挖掘

$$\mathrm{CE}(p_t) = -\log(p_t)$$
$$\mathrm{FL}(p_t) = -(1-p_t)^{\gamma}\log(p_t)$$

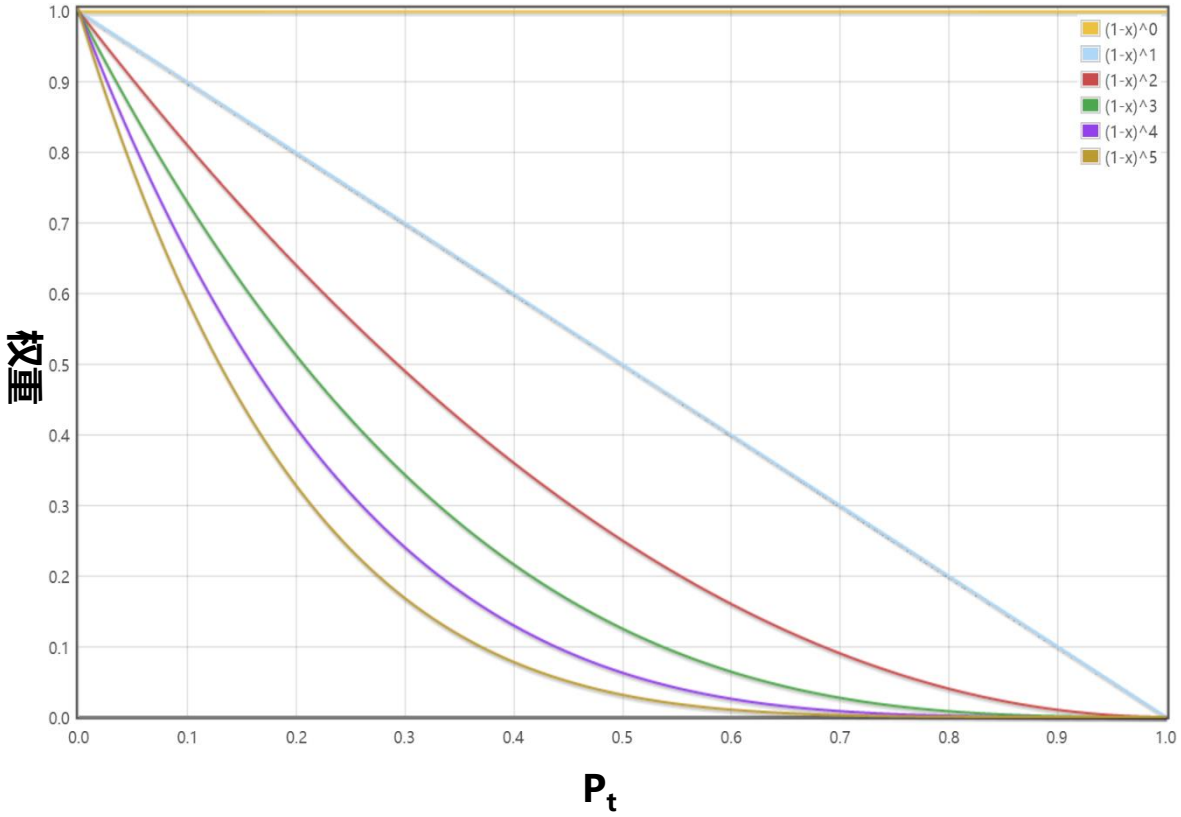
难样本挖掘

- 难样本挖掘是Hard Weight（硬加权）
- 选中的样本，权重为1
- 滤掉的样本，权重为0
- 挖掘固定比例或数量的样本
- 不能充分利用所有的样本

Focal Loss

- Focal Loss是Soft Weight（软加权）
- 分类分的越正确，权重越低
- 分类分的越错误，权重相对越高
- 所有的样本都参与训练
- 能够充分利用所有的样本

RetinaNet检测算法：Focal Loss和难样本挖掘



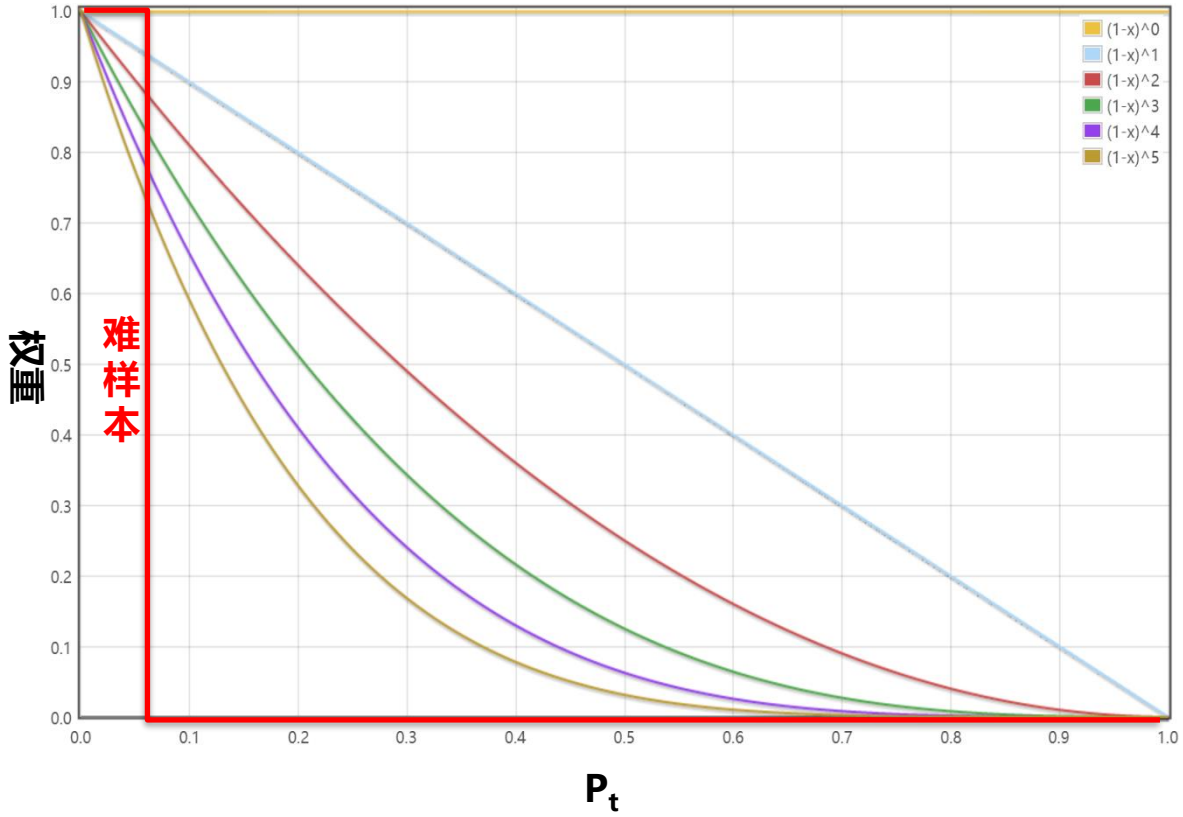
难样本挖掘

- 硬加权
- 选中的样本，权重为1
- 滤掉的样本，权重为0
- 挖掘固定比例或数量的样本
- 不能充分利用所有的样本

Focal Loss

- 软加权
- 分类越正确，权重越低
- 分类越错误，权重相对越高
- 所有的样本都参与训练
- 能够充分利用所有的样本

RetinaNet检测算法：Focal Loss和难样本挖掘



难样本挖掘

- 硬加权
- 选中的样本，权重为1
- 滤掉的样本，权重为0
- 挖掘固定比例或数量的样本
- 不能充分利用所有的样本

Focal Loss

- 软加权
- 分类越正确，权重越低
- 分类越错误，权重相对越高
- 所有的样本都参与训练
- 能够充分利用所有的样本

RetinaNet检测算法：模型初始化

- ResNet基础网络是用ImageNet预训练的模型进行初始化
- 所有新添加的卷积层都是用σ = 0.01的高斯函数来随机初始化权重w，除了分类子网络中的最后一个卷积层的偏置b，其他卷积层的偏置都初始化为0
- 分类子网络中的最后一个卷积层的偏置b，用下面这个公式进行初始化：

$$b = -\log((1-\pi)/\pi)$$

- 其中π = 0.01，这意味着在训练开始时，每个锚框的前景得分大约是~0.01，目的是降低损失函数的初始值

RetinaNet检测算法：分类子网络初始化

RetinaNet检测算法：分类子网络初始化

x —最后一个卷积层→ y —Sigmoid函数→ p

RetinaNet检测算法：分类子网络初始化

$$x \xrightarrow{\text{最后一个卷积层}} y \xrightarrow{\text{Sigmoid函数}} p$$

$$y = wx + b$$

$$p = \frac{1}{1 + e^{-y}} = \frac{1}{1 + e^{-(wx+b)}}$$

RetinaNet检测算法：分类子网络初始化

$$x \xrightarrow{\text{最后一个卷积层}} y \xrightarrow{\text{Sigmoid函数}} p$$

$$y = wx + b$$

$$p = \frac{1}{1 + e^{-y}} = \frac{1}{1 + e^{-(wx+b)}}$$

1. 权重w用σ=0.01的Gaussian函数初始化，偏置b初始化为0，那么p为：

$$p = \frac{1}{1 + e^{-((w \approx 0)x+0)}} = 0.5$$

RetinaNet检测算法：分类子网络初始化

$$x \xrightarrow{\text{最后一个卷积层}} y \xrightarrow{\text{Sigmoid函数}} p$$

$$y = wx + b$$

$$p = \frac{1}{1 + e^{-y}} = \frac{1}{1 + e^{-(wx+b)}}$$

1. 权重w用σ=0.01的Gaussian函数初始化，偏置b初始化为0，那么p为：

$$p = \frac{1}{1 + e^{-((w \approx 0)x+0)}} = 0.5$$

2. 权重w用σ=0.01的Gaussian函数初始化，偏置b初始化为$-\log((1-\pi)/\pi)$，那么p为：

$$p = \frac{1}{1 + e^{-((w \approx 0)x - \log((1-\pi)/\pi))}} = \pi = 0.01$$

RetinaNet检测算法：分类子网络初始化

P从0.5到0.01的作用

RetinaNet检测算法：分类子网络初始化

P从0.5到0.01的作用

- P=0.5时

$$CE(p,y)=\begin{cases}-\log(p)=-\log(0.5)=0.693, & \text{正样本}\\ -\log(1-p)=-\log(0.5)=0.693, & \text{负样本}\end{cases}$$

RetinaNet检测算法：分类子网络初始化

P从0.5到0.01的作用

- P=0.5时

$$CE(p,y)=\begin{cases}-\log(p)=-\log(0.5)=0.693, & \text{正样本}\\ -\log(1-p)=-\log(0.5)=0.693, & \text{负样本}\end{cases}$$

- P=0.01时

$$CE(p,y)=\begin{cases}-\log(p)=-\log(0.01)=4.605, & \text{正样本}\\ -\log(1-p)=-\log(0.99)=0.010, & \text{负样本}\end{cases}$$

RetinaNet检测算法：分类子网络初始化

P从0.5到0.01的作用

- P=0.5时

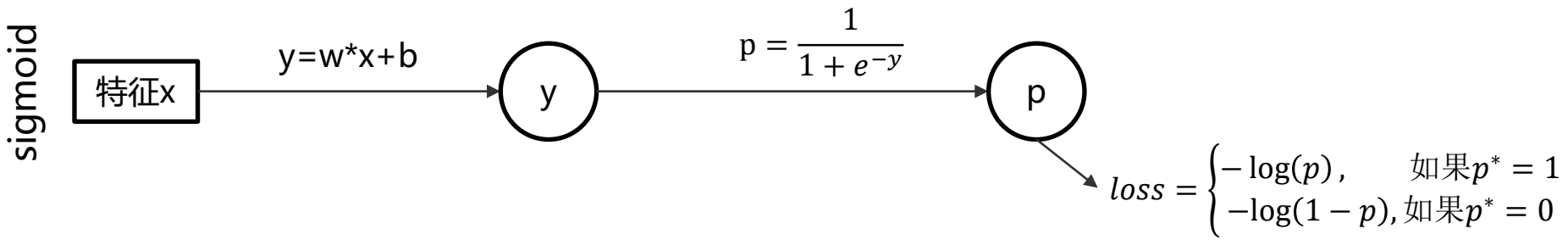
$$CE(p,y)=\begin{cases}-\log(p)=-\log(0.5)=0.693, & \text{正样本}\\ -\log(1-p)=-\log(0.5)=0.693, & \text{负样本}\end{cases}$$

- P=0.01时

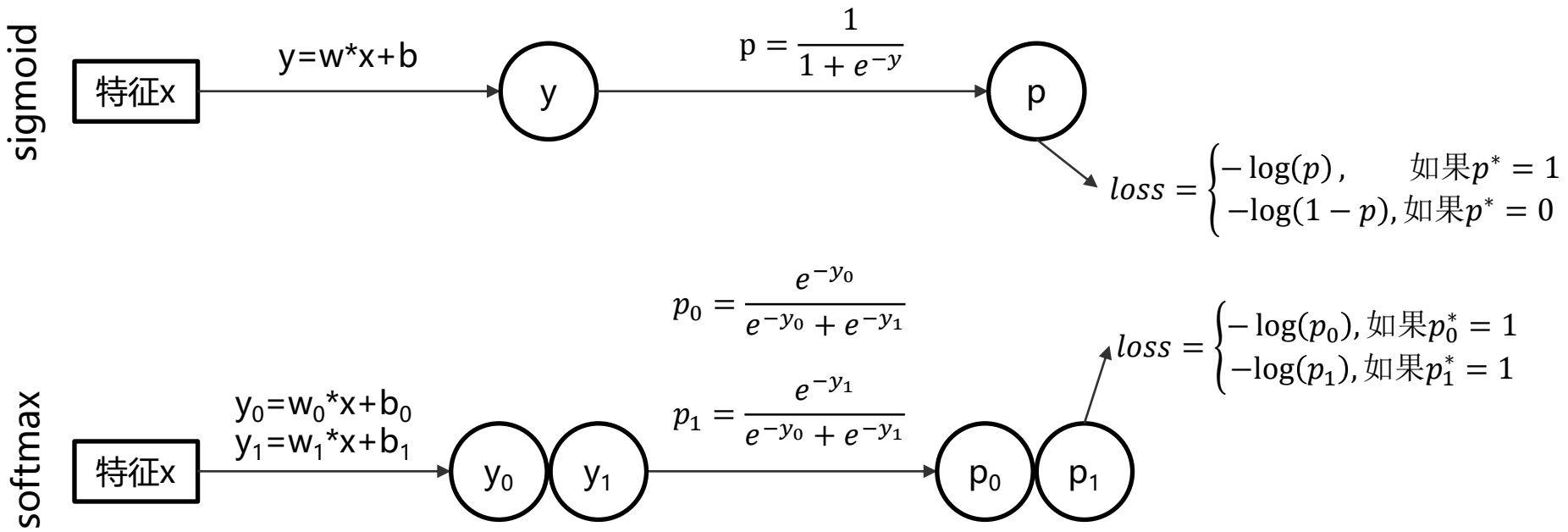
$$CE(p,y)=\begin{cases}-\log(p)=-\log(0.01)=4.605, & \text{正样本}\\ -\log(1-p)=-\log(0.99)=0.010, & \text{负样本}\end{cases}$$

- 当正负样本为1:10时，loss从0.693+0.693*10=7.623变为4.605+0.010*10=4.705，减小1.62倍
- 当正负样本为1:100时，loss从0.693+0.693*100=69.993变为4.605+0.010*100=5.605，减小12.49倍
- 当正负样本为1:1000时，loss从0.693+0.693*1000=693.693变为4.605+0.010*1000=14.605，减小47.50倍
- 当正负样本为1:10000时，loss从0.693+0.693*10000=6930.693变为4.605+0.010*10000=104.605，减小66.26倍

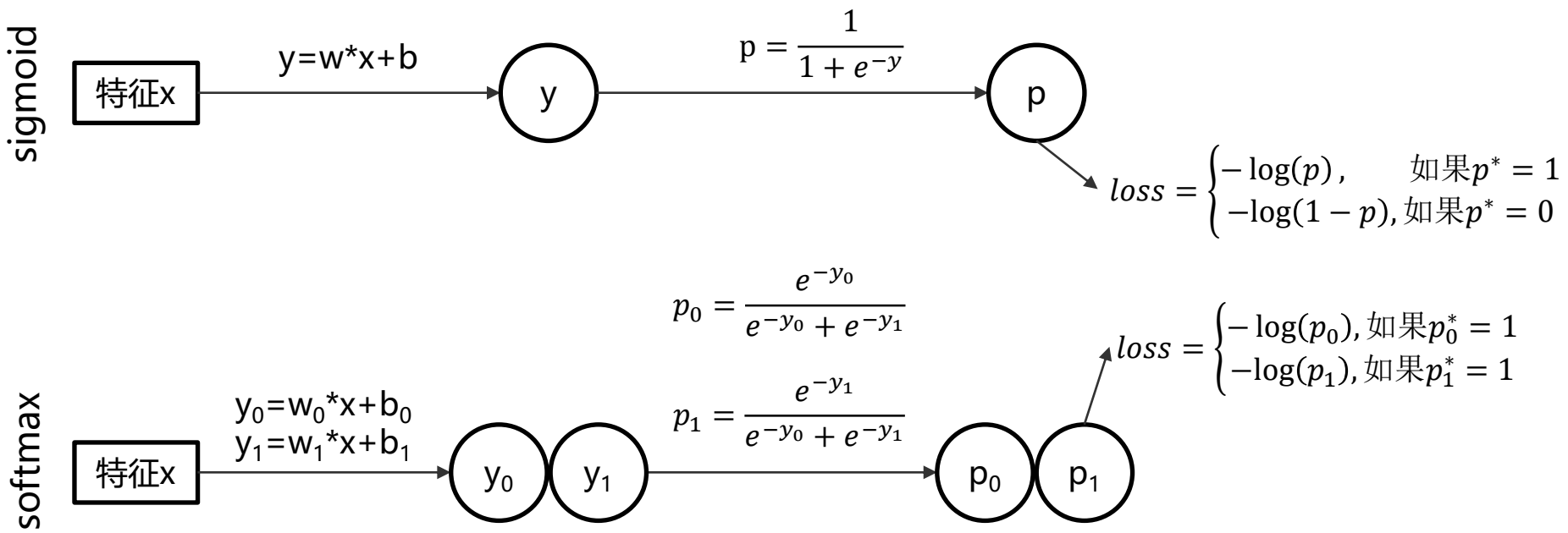
RetinaNet检测算法：Sigmoid和Softmax做二分类



RetinaNet检测算法：Sigmoid和Softmax做二分类

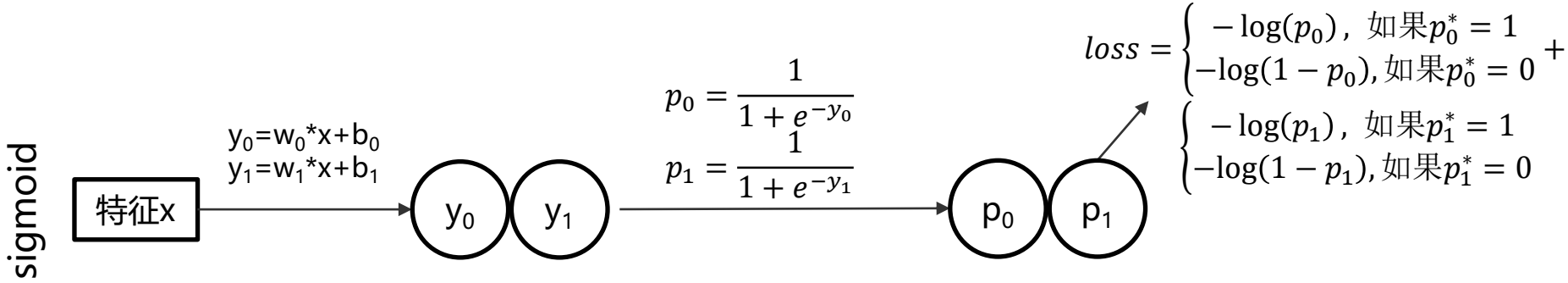


RetinaNet检测算法：Sigmoid和Softmax做二分类

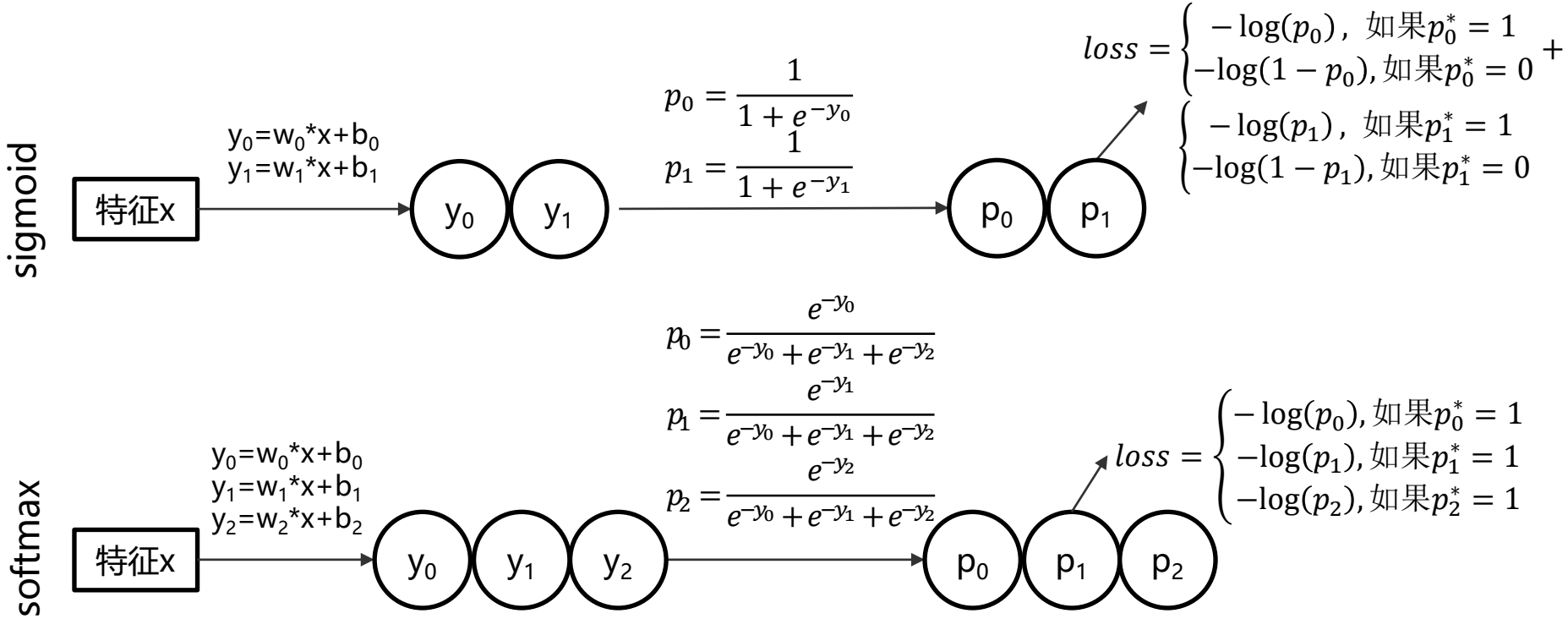


二分类任务时，softmax跟sigmoid等价

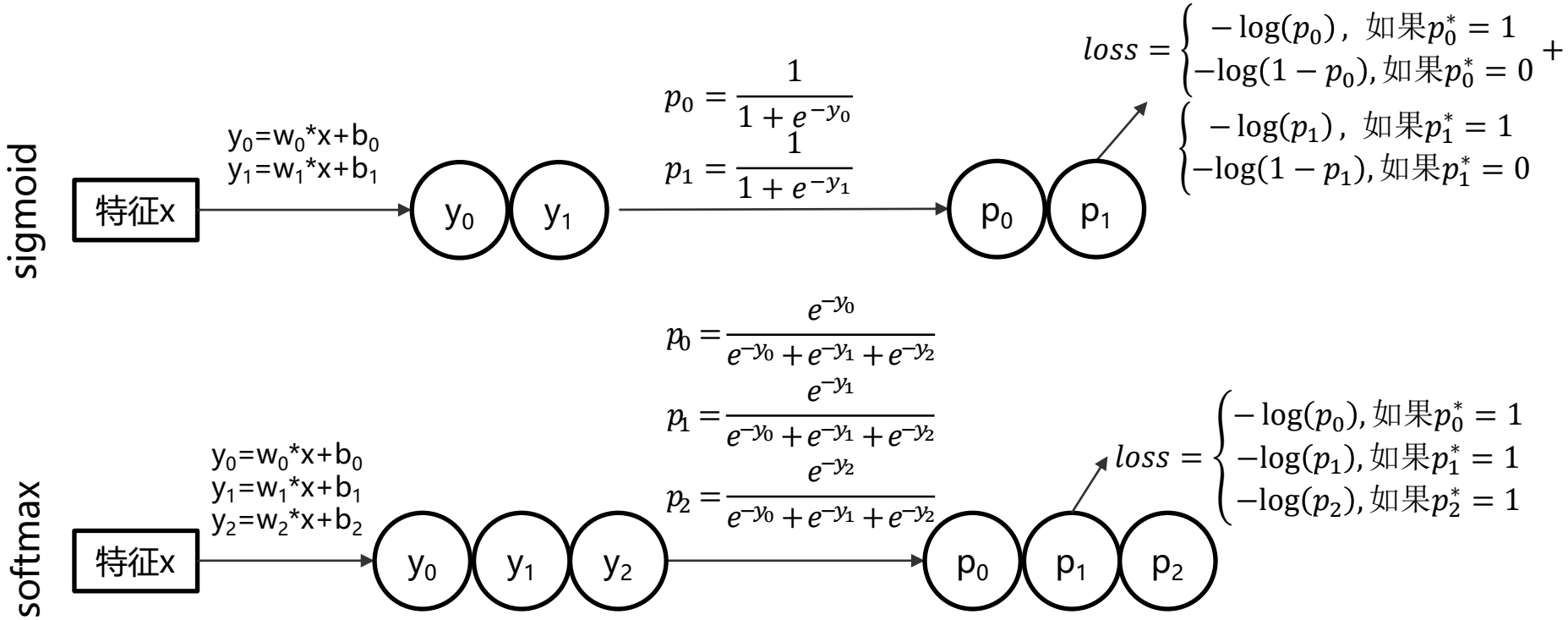
RetinaNet检测算法：Sigmoid和Softmax做多分类



RetinaNet检测算法：Sigmoid和Softmax做多分类



RetinaNet检测算法：Sigmoid和Softmax做多分类



多（三）分类任务时，softmax跟sigmoid不同

RetinaNet检测算法：Sigmoid和Softmax对比

	Sigmoid	Softmax	异同
检测二分类	1. 最后一层输入是1维度的x 2. 正样本的概率$p_1 = \frac{1}{1+e^{-x}}$ 3. 负样本的概率$p_0 = 1 - p_1$	1. 最后一层输入是2维度的x_0, x_1 2. 正样本的概率$p_1 = \frac{e^{x_1}}{e^{x_0}+e^{x_1}}$ 3. 负样本的概率$p_0 = \frac{e^{x_0}}{e^{x_0}+e^{x_1}}$	1. 换算$x = \frac{x_1}{x_0}$ 2. 二分类时，sigmoid loss与softmax loss等价，只是softmax会多一个维度
检测N分类	1. 最后一层输入是N维度的x_0, x_1, ..., x_{N-1} 2. 属于某一类的概率$p_n = \frac{1}{1+e^{-x_n}}$ 3. 不属于某一类的概率$p'_n = 1 - p_n$	1. 最后一层输入是N+1维度的x_0, x_1, ..., x_{N-1}, x_N 2. 属于某一类的概率$p_n = \frac{e^{x_n}}{e^{x_0}+e^{x_1}+\cdots+e^{x_n}+\cdots e^{x_N}}$ 3. 不属于某一类的概率$p'_n = 1 - p_n = \frac{e^{x_0}+e^{x_1}+\cdots+e^{x_{n-1}}+e^{x_{n+1}}+\cdots e^{x_N}}{e^{x_0}+e^{x_1}+\cdots+e^{x_n}+\cdots e^{x_N}}$	1. Softmax有类间归一化操作，故有显性的互斥性，即属于某一类，就不能属于其他类 2. Sigmoid没有类间归一化操作，故没有显性的互斥性 3. Softmax会比Sigmoid多一个维度