

目标检测

概览

1. 目标检测简介；
2. 目标检测常用评价指标与工具；
3. 单目标分类与定位算法；
4. 使用CNN进行目标检测；
5. YOLO算法；
6. SSD算法。

1. 目标检测简介

目标检测

目标检测（Object Detection） 是计算机视觉与图像处理技术，涉及检测数字图像、视频中的对象（例如人、动物、人脸等）。可用于图像搜索、安防等多种领域。

分类、定位与目标检测



分类 → 哈士奇

单目标

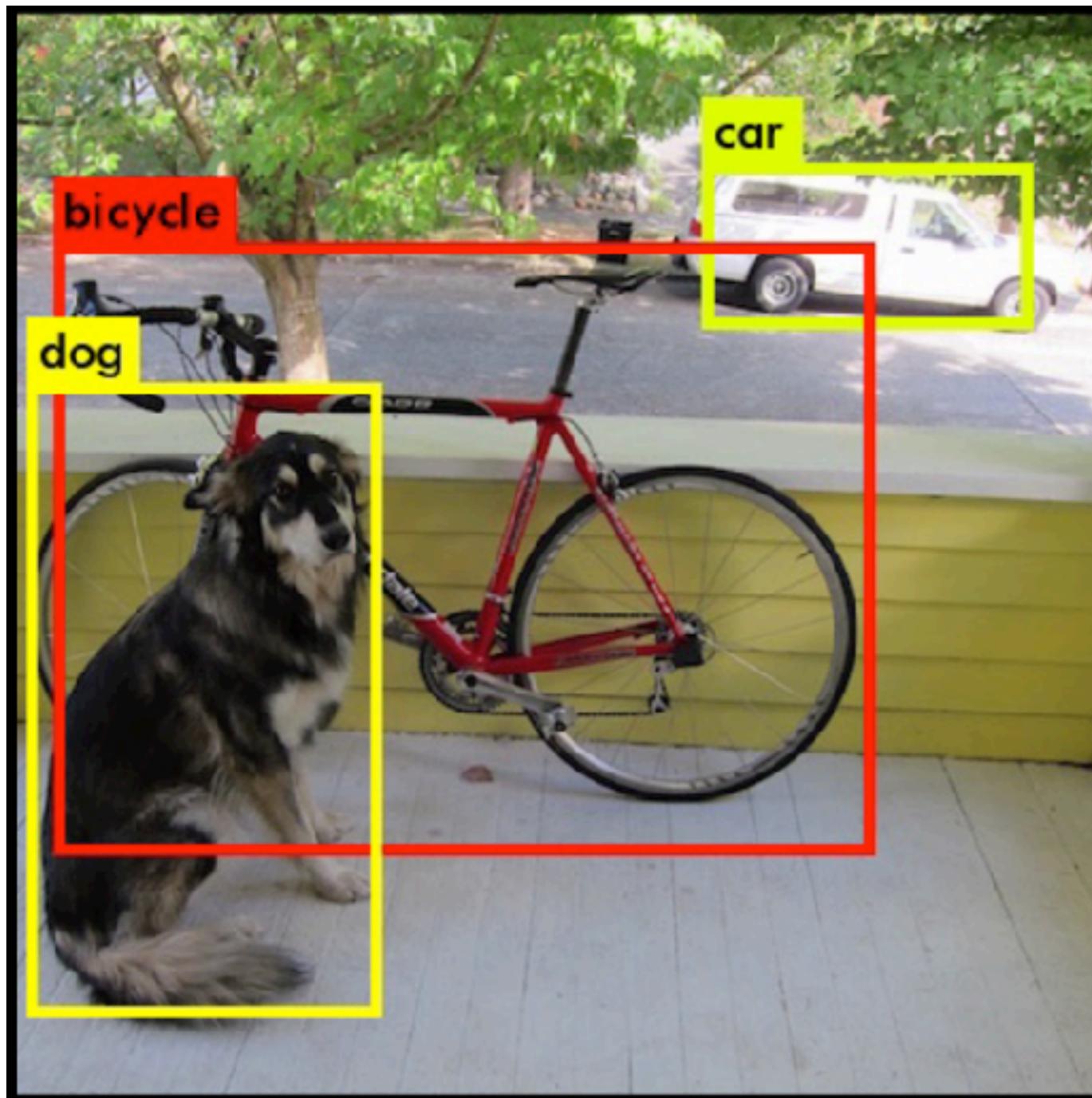
定位 → (x, y, w, h)



目标检测 → 哈士奇 (x_1, y_1, w_2, h_1)
哈士奇 (x_2, y_2, w_2, h_2)
.....

多目标

目标检测用途



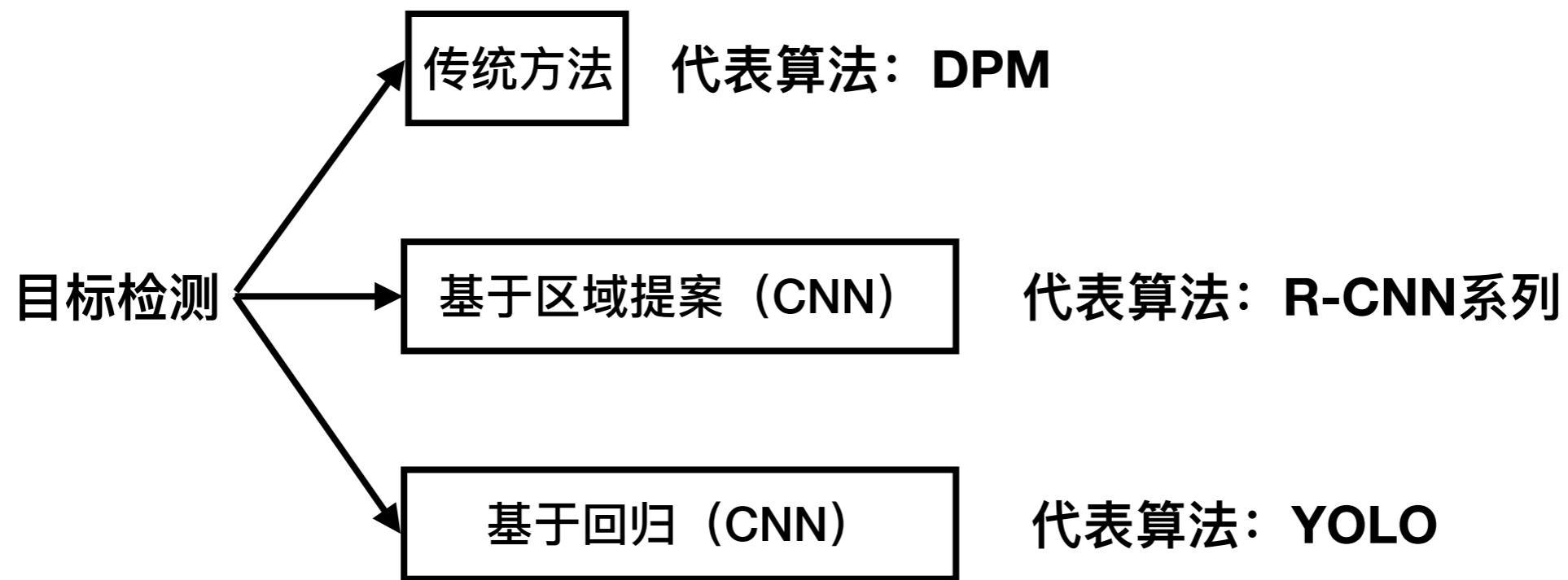
目标检测用途



目标检测近期发展历程

- 2008年，DPM（Deformable Part Model）算法被提出，DPM算法利用特征提取器提取特征对目标进行提取与识别，是经典的传统图像目标检测算法；
- 2014年前后，RCNN（Regions with Convolutional Neural Network Features）诞生，其在性能上大幅领先传统方法，拉开了使用深度学习解决目标检测问题的帷幕；
- 2015年，Faster RCNN的诞生极大的提高了模型检测速度与性能；
- 2016年前后，YOLO模型利用端到端训练与预测的方法实现了一个模型、一次训练完成目标检测的任务；之后不久，诞生了SSD算法，继承了Faster RCNN与YOLO的优点，不仅使构建与训练模型变得简单，而且极大的提高了模型性能与检测速度。

目标检测发展分支



目标检测算法中，更快、更准确这两个指标往往较难平衡，例如传统方法速度较快，但往往检测结果不理想，RCNN系列极大的提高了检测性能，但其速度较慢，应用场景受限。基于回归的CNN模型构建的目标检测算法构建简单、速度快且有较高的性能，是目标检测领域发展的重要方向。

2. 目标检测常用 评价指标与工具

常见名词解释

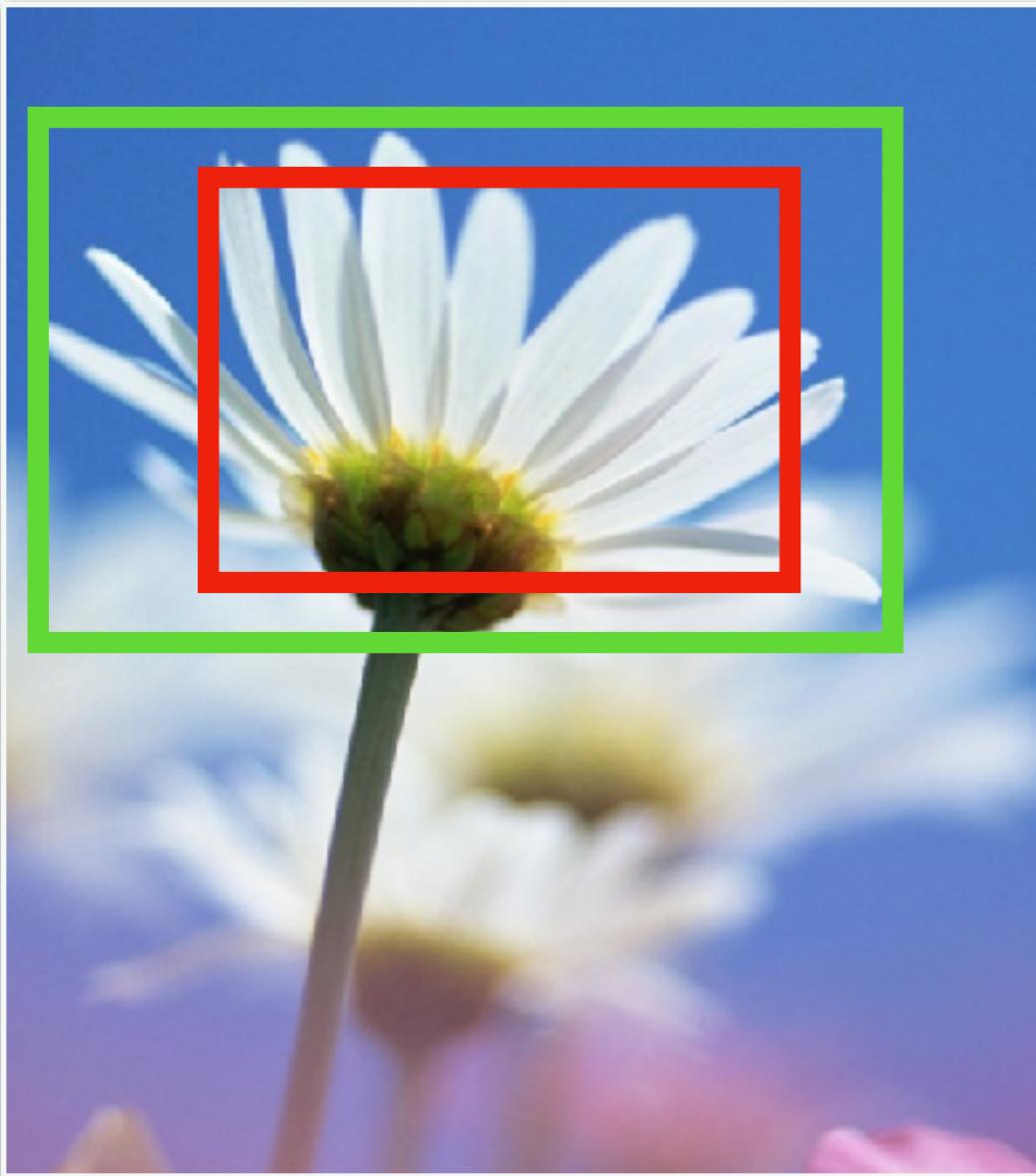
- **区域提案 (Region Proposal, RP)** 方法：从被检测图片中以某种规则筛选出的可能存在目标的区域的方法。
- **边界框 (Bounding Box, BBox)** : 图像中物体的边框（通常指模型预测的边框），一般包含坐标值与宽高等信息。
- **参考标准 (Ground Truth)** : 数据集中标注的结果，这里常用来表示标注的边框与框中物体类别等信息。

目标检测评价指标

1. 边框重合度 (**Intersection Over Union, IOU**) , 预测的边框与标注的边框重合度的评价指标。
2. mAP (**mean Average-Precision**) , 预测框内物体类别准确度的衡量指标 (有时也可使用使用错误率、AUC均值等指标衡量) 。

2.1 IOU

IOU



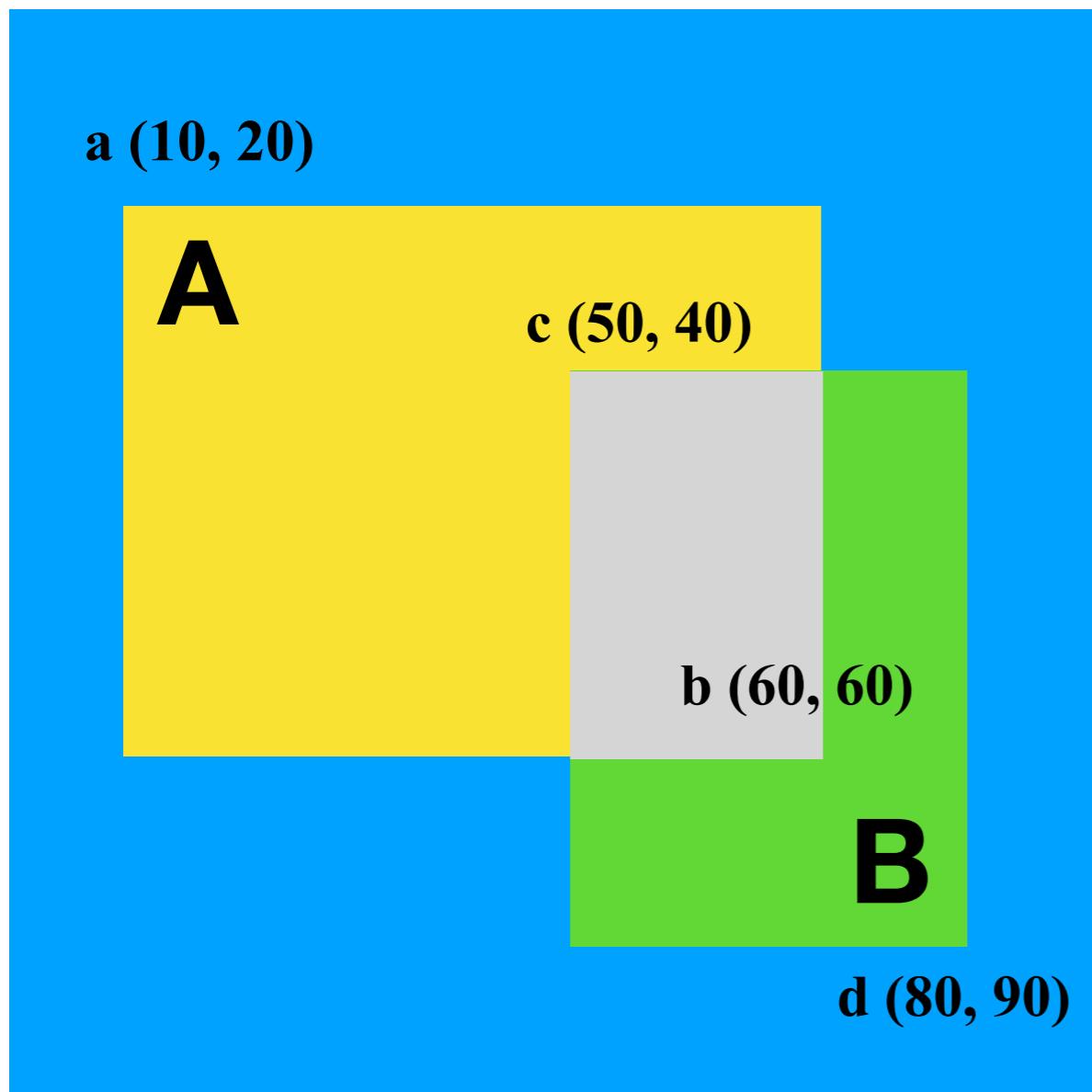
若GroundTruth窗口面积为A,
算法预测的BoundingBox面积为B,

$$\text{则 } \text{IOU} = \frac{A \cap B}{A \cup B}$$

当IOU = 0时， 表示不重叠；
当IOU = 1时， 表示完全重叠；
通常设置一个阈值， 低于此阈值
则表示没有检测到目标。

绿色框表示标记， 红色框表示预测结果。

实例一：当两个区域有交叉时



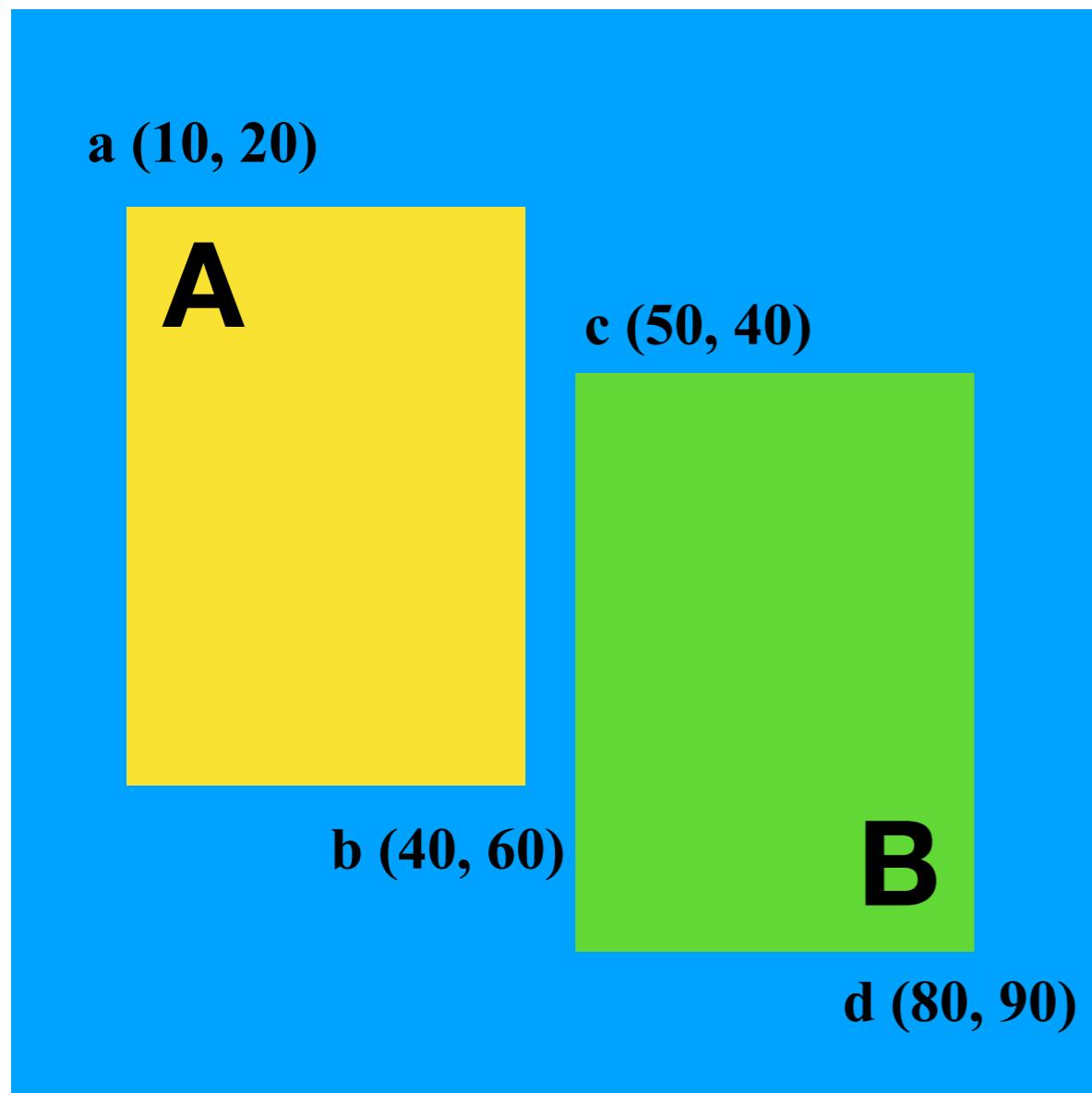
问题：背景是一张100*100大小的图片，A区域左上角坐标a为(10, 20)，右下角坐标b为(60, 60)；B区域左上角坐标c为(50, 40)，右下角坐标b为(80, 90)，则IOU值为多少？如何计算？

求解步骤：

a、c坐标比大
b、d坐标比小

1. 求交叉区域左上角与又下角坐标；
2. 根据交叉区域坐标求出交叉区域宽高；
3. 计算A、B两个区域的交集与并集面积；
4. 求得IOU值。

实例二：当两个区域无交叉时



问题：背景是一张 $100*100$ 大小的图片，A区域左上角坐标a为 $(10, 20)$ ，右下角坐标b为 $(40, 60)$ ；B区域左上角坐标c为 $(50, 40)$ ，右下角坐标b为 $(80, 90)$ ，则IOU值为多少？如何计算？

当A、B区域无交集时，a、c之间的最大值得到的坐标，b、d之间最小值得到的坐标无法组成一个矩形区域。

求IOU值的一般步骤

1. 比较两个窗口区域左上角坐标，分别取横纵坐标最大值，组成新坐标点e；
2. 比较两个窗口区域右下角坐标，分别取横纵坐标最小值，组成新坐标点f；
3. 使用点f的横纵坐标分别减去e的横纵坐标得到差值；
4. 若两个差值中存在小于等于0的值，则表示两个区域交集面积为0，否则交集面积等于差值的积，即为两个区域的交集；
5. 求得两个区域的并集 = 两个区域的面积之和 - 两个区域的交集；
6. 根据交并集求得IOU值。

使用TensorFlow实现IOU计算

```
def calc_iou(box1, box2):
    left_up = tf.maximum(box1[:2], box2[:2])
    right_down = tf.minimum(box1[2:], box2[2:])
    intersection = tf.maximum(0., right_down - left_up)
    inter_square = intersection[0] * intersection[1]

    box1_square = (box1[2] - box1[0]) * (box1[3] - box1[1])
    box2_square = (box2[2] - box2[0]) * (box2[3] - box2[1])

    iou = inter_square / (box1_square + box2_square - inter_square + 1e-8)

    return iou

with tf.Session() as sess:
    iou = calc_iou([10., 20., 60., 60.], [50., 40., 80., 90.])
    print(sess.run(iou))
    iou = calc_iou([10., 20., 40., 60.], [50., 40., 80., 90.])
    print(sess.run(iou))
```

0.0606061
0.0

IOU优缺点

- 计算简便，是目前使用较多的方法；
- 对于不同大小、形状的物体，使用IOU值计算预测框与标记框之间的重合度可能是不合理的。

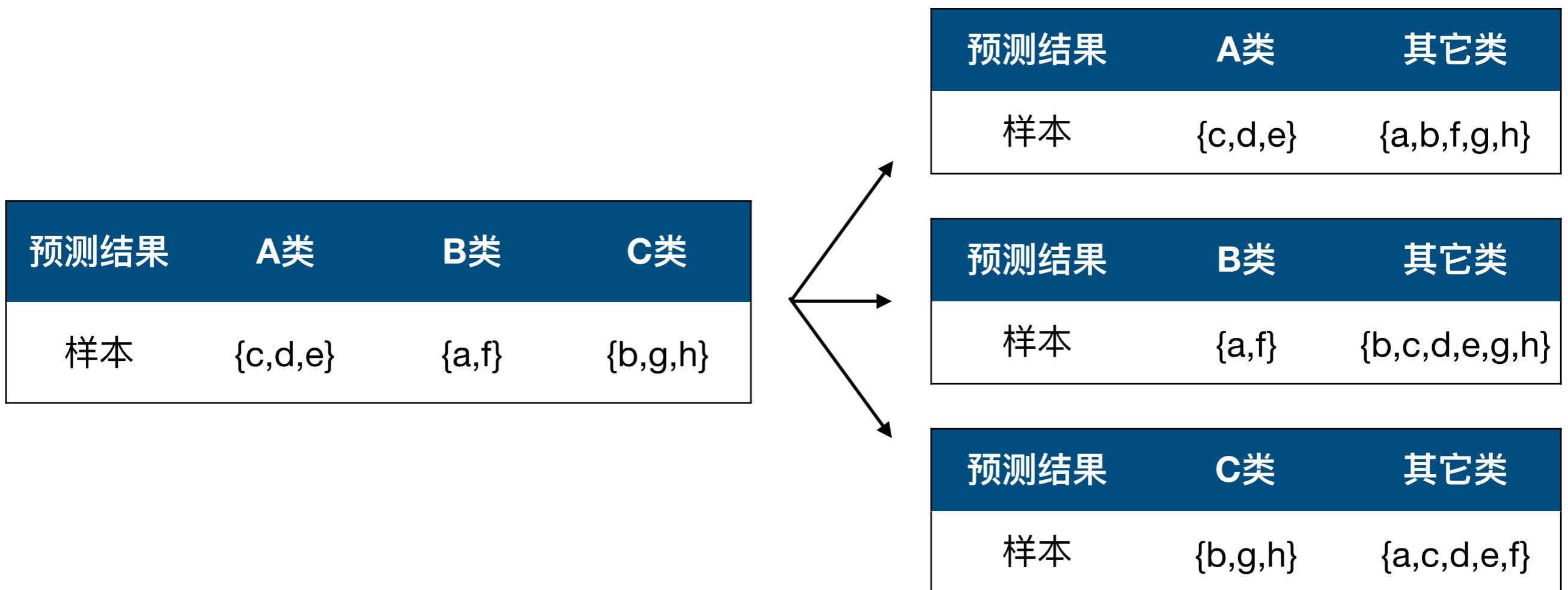
练习

- 改进上述使用TensorFlow计算两个box的IOU值的实现，使其可以同时完成任意多对box计算IOU值。

2.2 mAP

mAP

mAP (mean Average-Precision) 应用在多类别分类情况下，即
将多类别分类结果转化为多个2分类结果，分别求得每个2分类的
AP值，最后再求多个AP的均值即可。



查准率与查全率

混淆矩阵

真实情况	预测结果	
	正例	反例
正例	TP (真正例)	FN (假反例)
反例	FP (假正例)	TN (真反例)

查准率 P 与查全率 R 分别定义为

$$P = \frac{TP}{TP + FP},$$

$$R = \frac{TP}{TP + FN}.$$

一般来说，查准率高时，查全率低；查全率高时，查准率低，只有在简单的任务中，才能使得查准率与查全率都很高。

分类阈值选取

思考：在神经网络中，二分类问题对应的模型输出层一般是0-1之间的概率值，默认的我们使用0.5作为阈值，有什么问题？

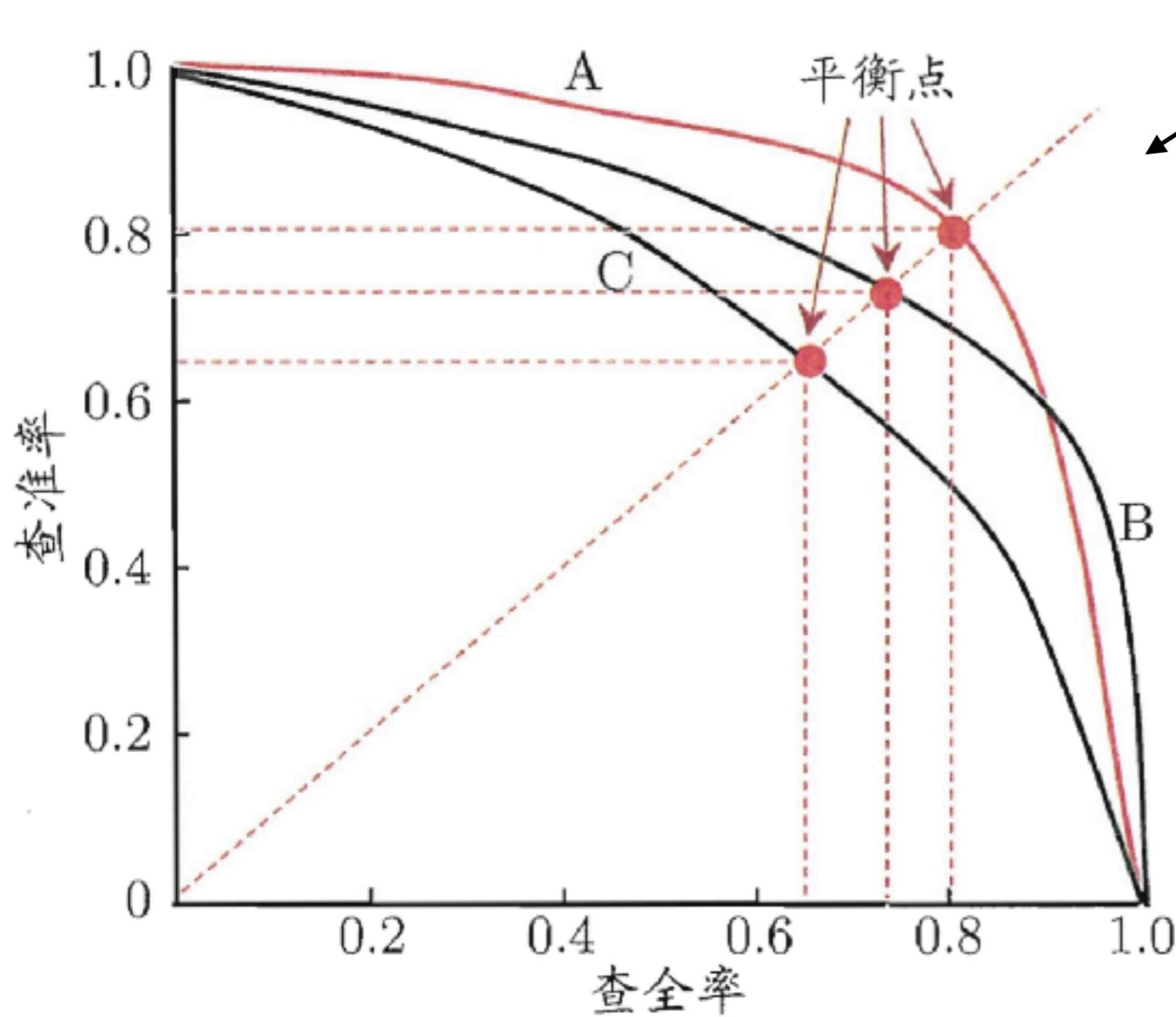
标签	模型输出
1	0.998
1	0.8
1	0.72
0	0.54
0	0.52
1	0.49
0	0.3
0	0.22
0	0.09

选取不同的分类阈值，可以得到不同的结果，一般的使用0.5作为阈值，并非最好的，也无法准确的衡量模型的性能，这时通常有两种做法：

1. 选取查准率与查全率的平衡点时的分类阈值作为衡量模型性能的分类阈值；
2. 使用P-R曲线的下面积AP作为衡量模型性能值。

使用哪个值作为阈值？

P-R曲线

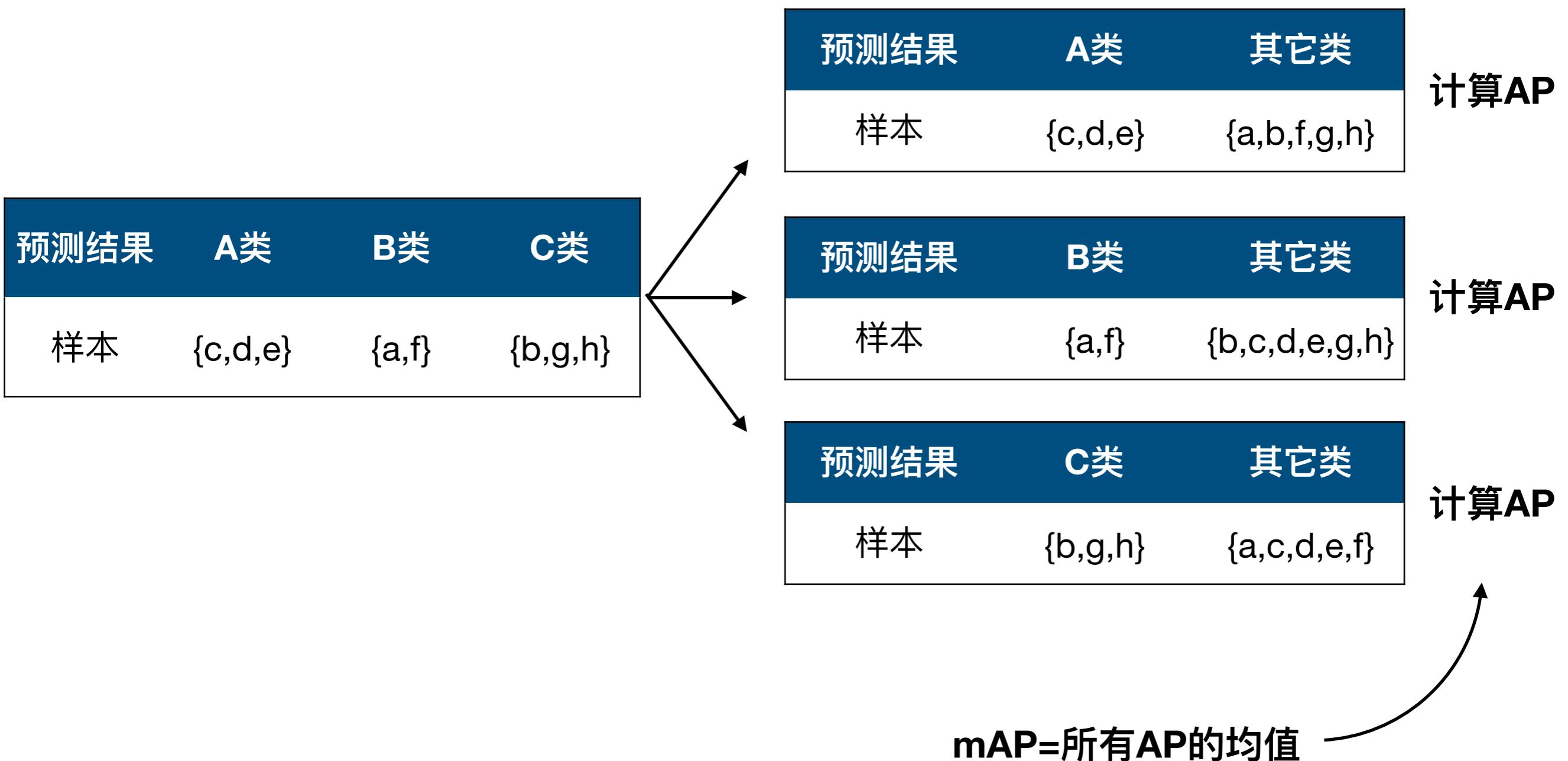


将样本按照结果排序，分别计算不同阈值下的查准率与查全率，绘制P-R曲线。

1. 使用平衡点衡量的方法计算简单但不够准确。
2. 使用P-R曲线下面积AP计算较为繁琐，但结果准确。
3. (了解) 介于两者之间的方案是F1值，即查准率与查全率的调和平均。

P-R曲线与平衡点示意图

mAP



计算AP示例

在二分类任务中，模型预测结果的如下：

样本	标记	预测结果
0	0	0.2
1	1	0.4
2	1	0.5
3	0	0.52
4	1	0.9

分类边界 (大于边界值则为正类)	混淆矩阵		查准率	查全率
0	3	0	3/5	3/3
0.2	2	0		
0.4	3	0	3/4	3/3
0.5	1	1		
0.52	2	1	2/3	2/3
0.9	1	1		
	1	2	1/2	1/3
	1	1		
	1	2	1/1	1/3
	0	2		
	0	3	1 (规定为1)	0/3
	0	2		

计算AP示例

A = (1, 0)

B = $\left(1, \frac{3}{4}\right)$

$\rightarrow (1, 0.75)$

C = $\left(\frac{2}{3}, \frac{2}{3}\right)$

$\rightarrow (0.67, 0.67)$

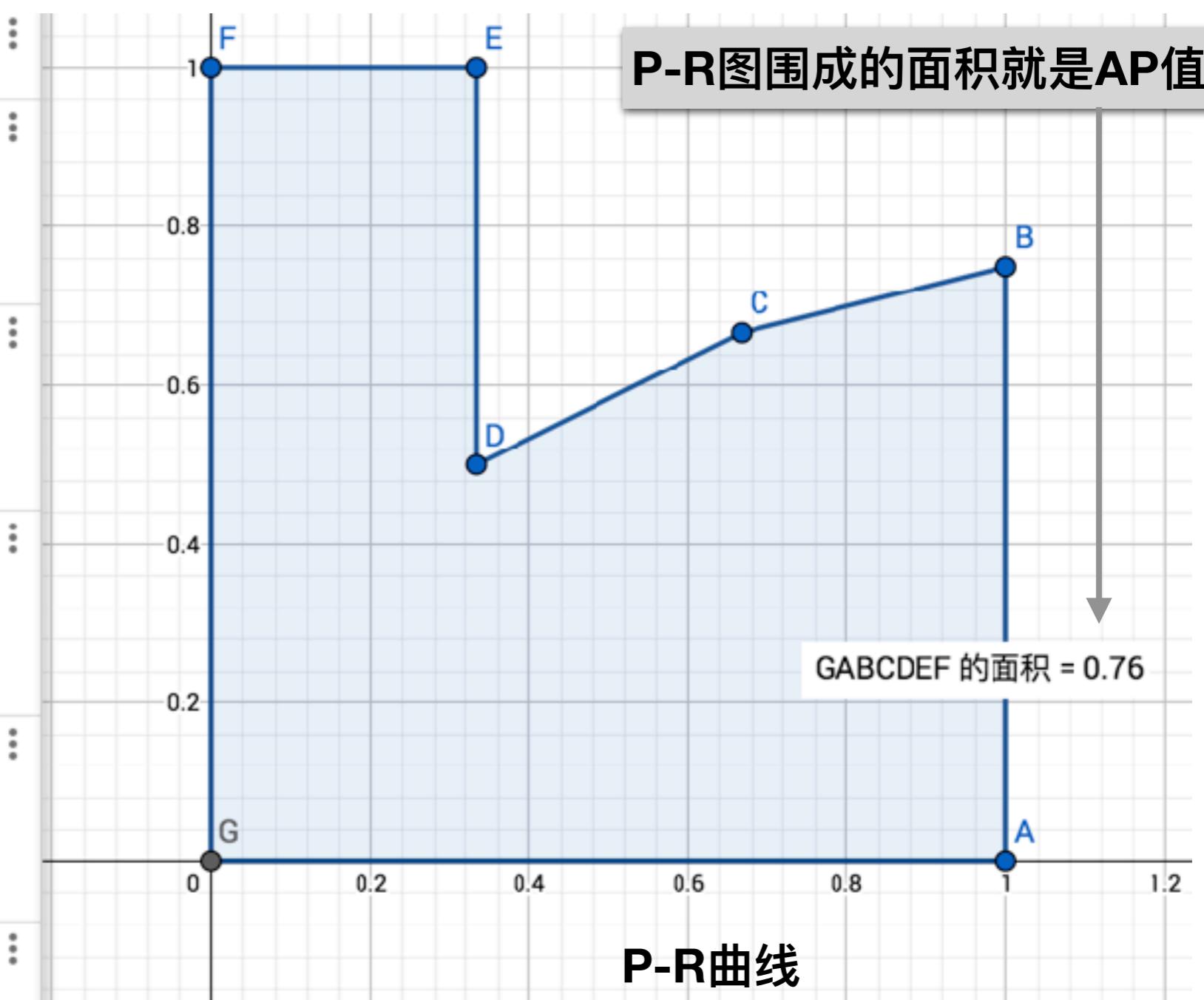
D = $\left(\frac{1}{3}, \frac{1}{2}\right)$

$\rightarrow (0.33, 0.5)$

E = $\left(\frac{1}{3}, 1\right)$

$\rightarrow (0.33, 1)$

F = (0, 1)



计算AP示例

使用sklearn下的metrics包中的average_precision_score方法
快捷计算AP：

```
import sklearn.metrics as metrics

metrics.average_precision_score(
    [0, 1, 1, 0, 1],
    [0.2, 0.4, 0.5, 0.52, 0.9])
```

```
0.7638888888888884
```

练习

训练一个模型完成3分类任务，此模型预测结果如下图，要求根据预测结果和标记值，利用sklearn相关模块，求得mAP值。

样本	标记	预测结果		
		A类	B类	C类
0	2	0.2	0.1	0.7
1	0	0.4	0.5	0.1
2	2	0.1	0.15	0.75
3	0	0.3	0.4	0.3
4	1	0.6	0.3	0.1
5	1	0.25	0.65	0.1
6	0	0.9	0.05	0.05
7	1	0.35	0.45	0.2

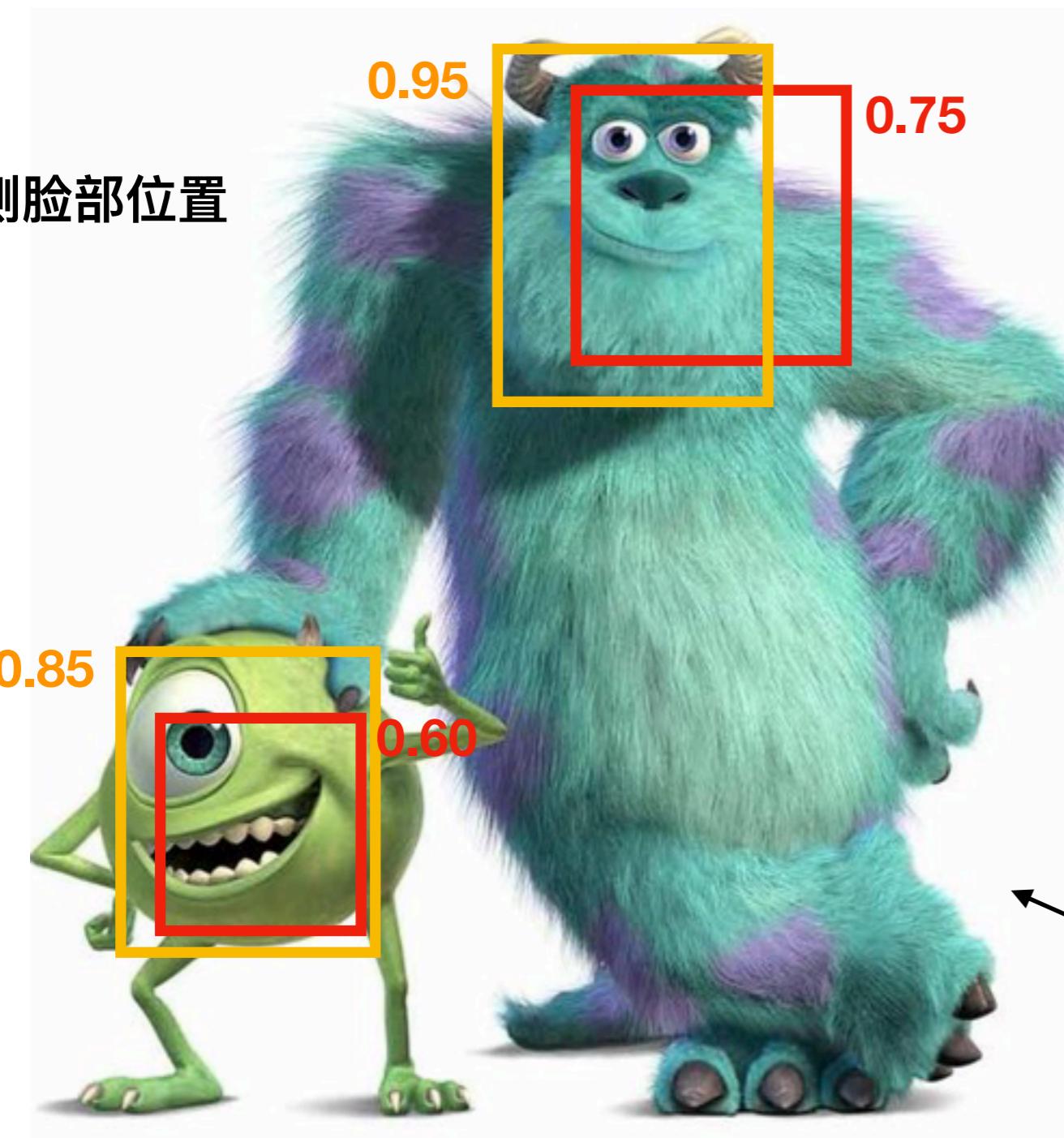
2.3 NMS

NMS

非极大值抑制（Non Maximum Suppression, NMS） 是一种搜索局部最大值的方法，在目标检测任务当中常被用来去除重叠度较高但置信度较低的候选窗口，提高检测精度与速度。

NMS的用途

检测脸部位置

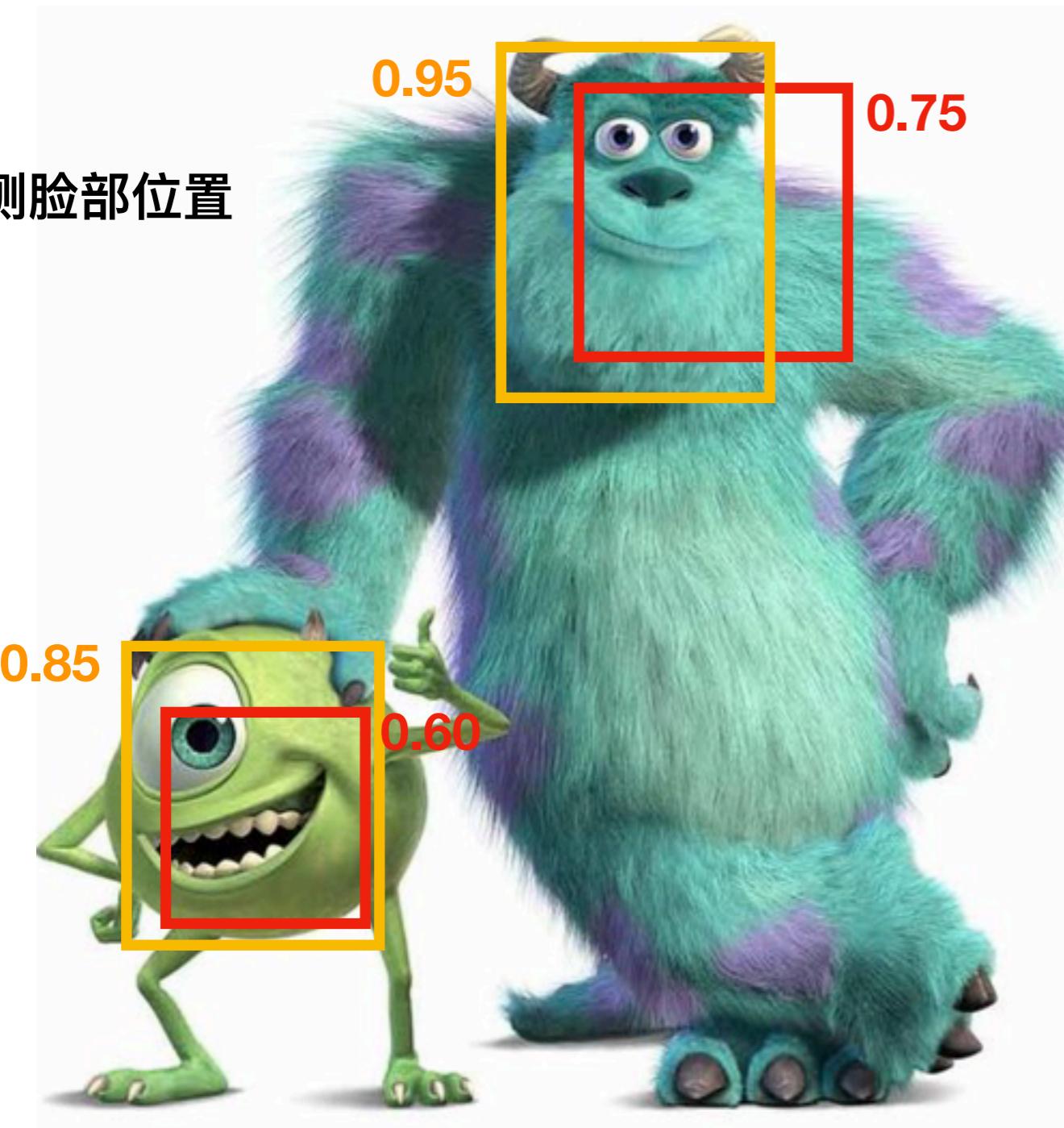


在执行目标检测任务时，往往一个目标周围会有很多候选区域，每一个候选区域会包含相应的分类得分或者目标存在置信度，而我们可以根据这些信息利用NMS方法去除冗余窗口。

这里我们希望去除红色的候选框

NMS执行过程

检测脸部位置



第一步：将候选框的得分排序，如下

0.95 0.85 0.75 0.6

选择得分最高的候选框，这里是0.95，并作为保留下来的候选框之一。

第二步：遍历其它候选框，并计算与当前所选候选框的IOU值，删除IOU值大于阈值（例如0.7）的候选框。

第三步：将未删除与未确定的剩余候选框作为新的候选框集合，执行第一步直到所有框被删除或者保留。

多类别NMS执行过程

多类别NMS算法，就是分别对每一个类别执行单类别NMS算法，具体来讲就是首先根据每一个框的分类结果确定当前框的所属类别，其次分别对每一个类别的框运行单类别NMS算法即可。

NMS算法的优缺点

- 思路简单，实现方便、快捷、高效，是常用的选择窗口去重方法；
- 在执行NMS前，往往需要先去除置信度很低的窗口；
- 当两个或多个目标重叠度很高时，NMS算法会删除置信度较低的选择框，从而降低目标的召回率。改进方法如soft-NMS。

NMS算法实现

第一步：实现IOU计算（此处使用Numpy实现IOU计算）

```
def calc_iou(box1, box2):
    left_up = np.maximum(box1[: 2], box2[: 2])
    right_down = np.minimum(box1[2: 4], box2[2: 4])
    intersection = np.maximum(0., right_down - left_up)
    inter_square = intersection[0] * intersection[1]

    box1_square = (box1[2] - box1[0]) * (box1[3] - box1[1])
    box2_square = (box2[2] - box2[0]) * (box2[3] - box2[1])

    iou = inter_square / (box1_square + box2_square - inter_square)

    return iou
```

NMS算法实现

第二步：实现NMS算法（单一类别）

```
def nms(boxes, threshold=0.7):
    boxes.sort(key=lambda boxes: boxes[4], reverse=True)
    res_boxes = []
    while len(boxes) != 0:
        res_boxes.append(boxes[0])
        del boxes[0]

        keep_num = 0
        for i in range(len(boxes)):
            iou = calc_iou(res_boxes[-1][0: 4], boxes[keep_num][0: 4])
            if iou > threshold:
                del boxes[keep_num]
            else:
                keep_num += 1

    return res_boxes
```

NMS算法实现

第三步：测试NMS算法

```
boxes = [[0, 0, 50, 50, 0.9],  
         [0, 0, 60, 50, 0.85],  
         [40, 40, 70, 70, 0.4],  
         [35, 45, 70, 80, 0.7]]
```

```
res_boxes = nms(boxes, 0.5)  
print(res_boxes)
```

```
[[0, 0, 50, 50, 0.9], [35, 45, 70, 80, 0.7]]
```

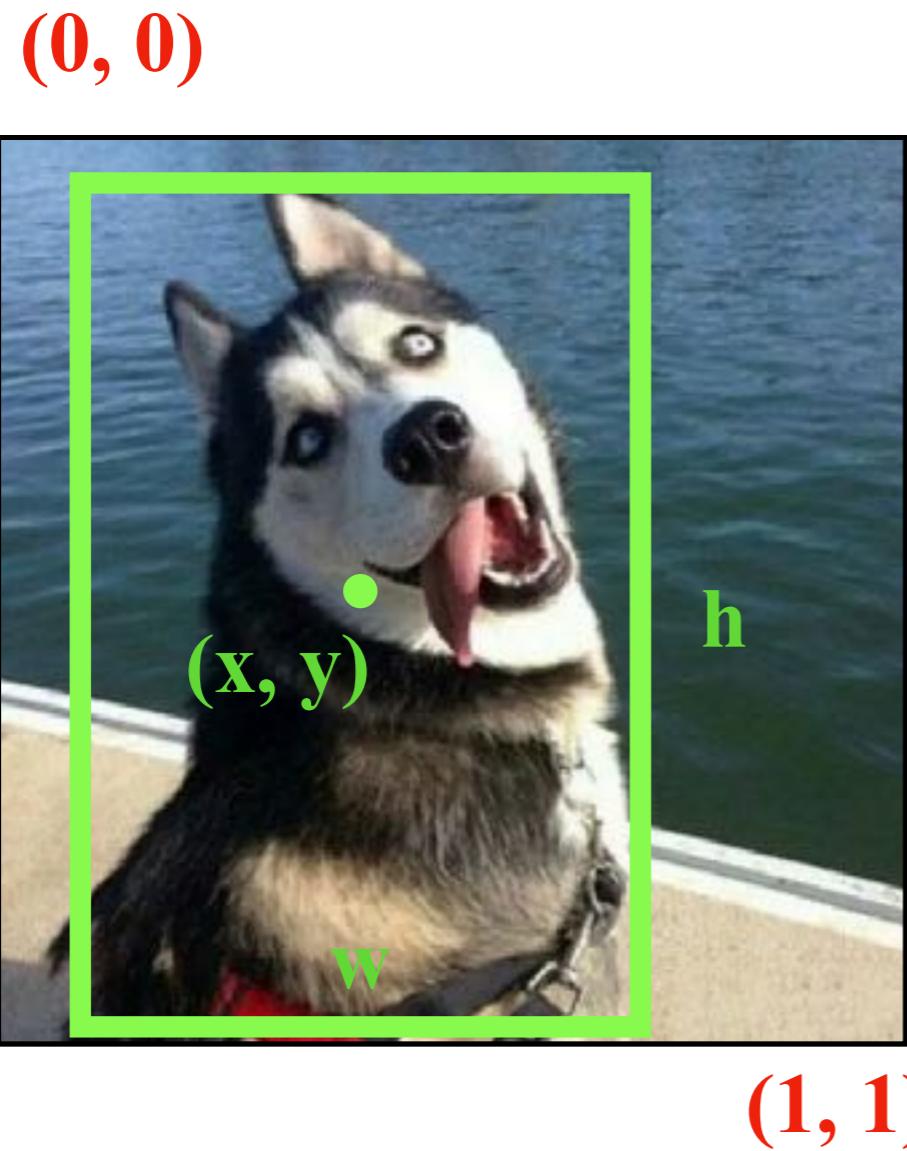
练习

- 参考上述单类别NMS算法的实现，实现2类别NMS算法，数据集如下：

box坐标	正类得分	负类得分
[0, 0, 50, 50, 0.9, 0.1]		
[0, 0, 60, 50, 0.85, 0.15]		
[40, 40, 70, 70, 0.4, 0.6]		
[35, 45, 70, 80, 0.2, 0.8]		
[40, 42, 68, 70, 0.3, 0.7]		

3. 单目标分类 与定位算法

输入定义



通常，我们将图片的宽高由size转化成为0-1之间的值，即图片左上角的坐标为 $(0, 0)$ ，图片右下角的坐标为 $(1, 1)$ 。

图片中的目标使用四个数值进行描述即目标左上角的坐标 (x, y) ，以及目标的宽w、高h。

目标定位

目标定位 (Target Localization) 任务中，输入图片可能包含一个目标或者不包含目标。要求当图片中包含目标时，输出目标类别与目标位置、大小；当图片中不包含目标时，输出图片为背景图片。



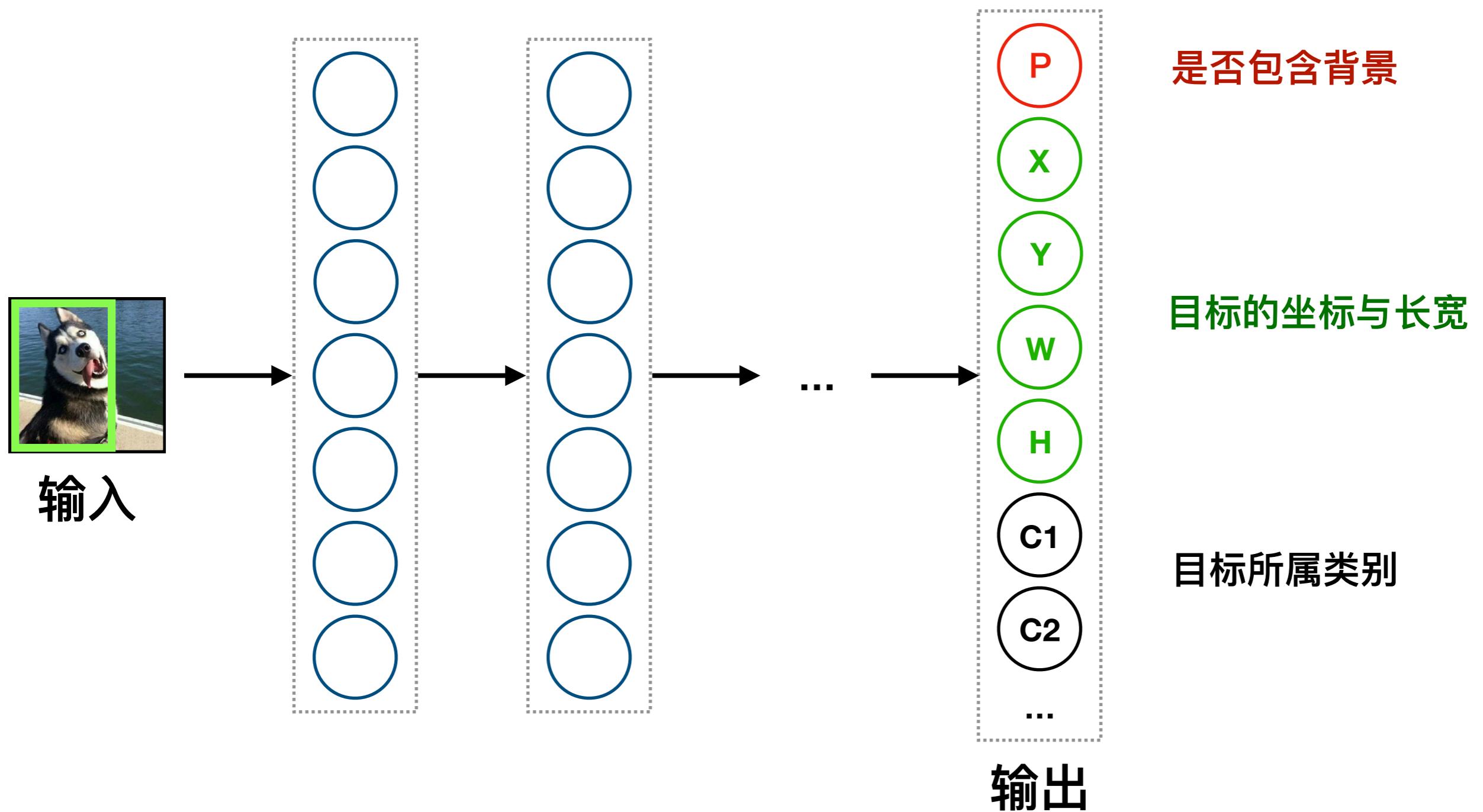
a



b

a图图片包含目标，检测时需要输出目标类别与位置；b图不包含目标。

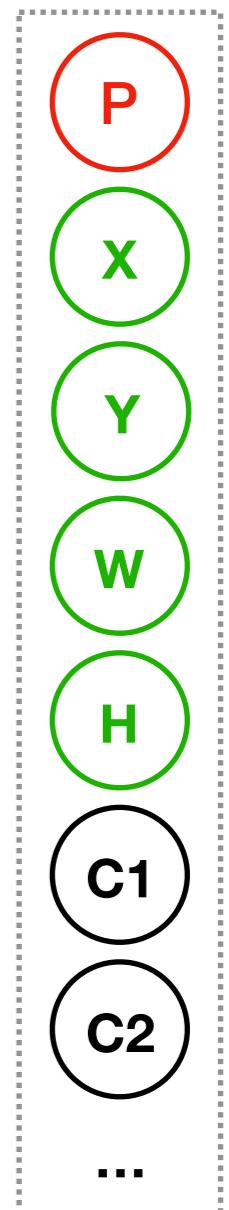
模型结构



模型的输出层

模型的输出层包括三部分内容，分别是：

1. 图片中包含目标的可能性；
 2. 目标的坐标与宽高；
 3. 目标所属类别。
- 当输出层中的P小于某个阈值时，我们认为此图片不包含目标，并且我们不关心其它神经元的输出；
 - 当输出层中的P大于某个阈值时，我们认为此图片包含目标，这时其它神经元的输出是有用的。



模型的输出层

目标定位代价函数

当图片中包含目标，即 $P=1$ 时：

$$J = \text{BgError} + \text{CoordError} + \text{ClassError}$$

此处误差衡量可以使用均方误差代价函数， BgError 与 ClassError 也可以使用交叉熵代价函数。

当图片中不包含目标，即 $P=0$ 时： $J = \text{BgError}$

TensorFlow实现代价函数

```
import tensorflow as tf

# 模拟模型输出与数据标记
outputs = tf.random_uniform([2, 8], 0, 1, dtype=tf.float32)
labels = tf.constant([[0, 0, 0, 0, 0, 0, 0, 0],
                      [1, 0.2, 0.3, 0.3, 0.8, 0, 0, 1]], dtype=tf.float32)

label_confidence = labels[:, 0: 1]
# 根据label_confidence生成掩码
mask = tf.concat([tf.ones_like(labels)[:, 0: 1],
                  (tf.ones_like(labels) * label_confidence)[:, 1: ]], 1)
# 对模型输出进行过滤, 使label_confidence==0的模型输出除了置信度以外的值设为0
outputs = outputs * mask

# 此处3部分代价均使用均方误差代价函数
cost = tf.reduce_mean(tf.square(labels - outputs))
```

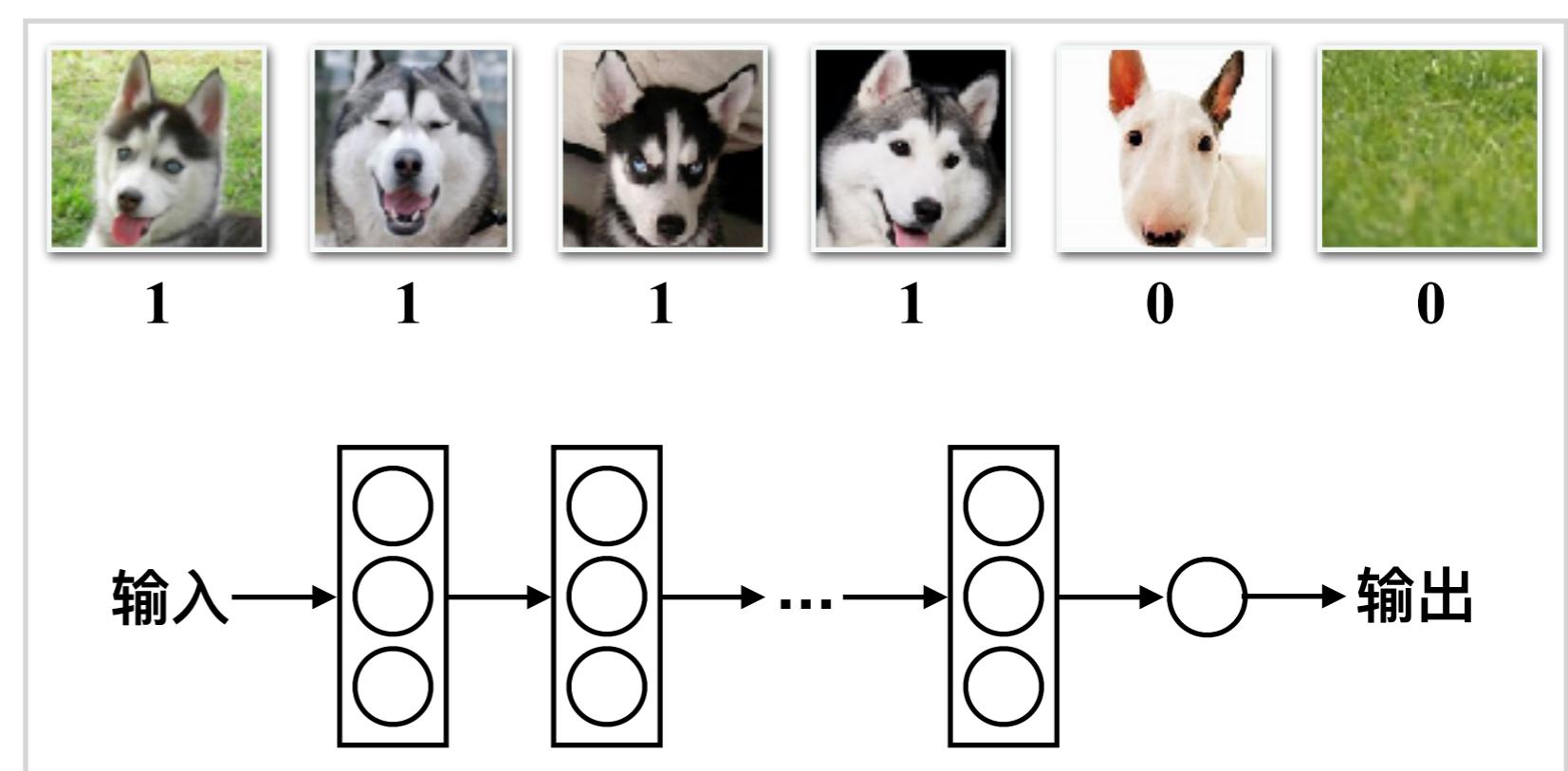
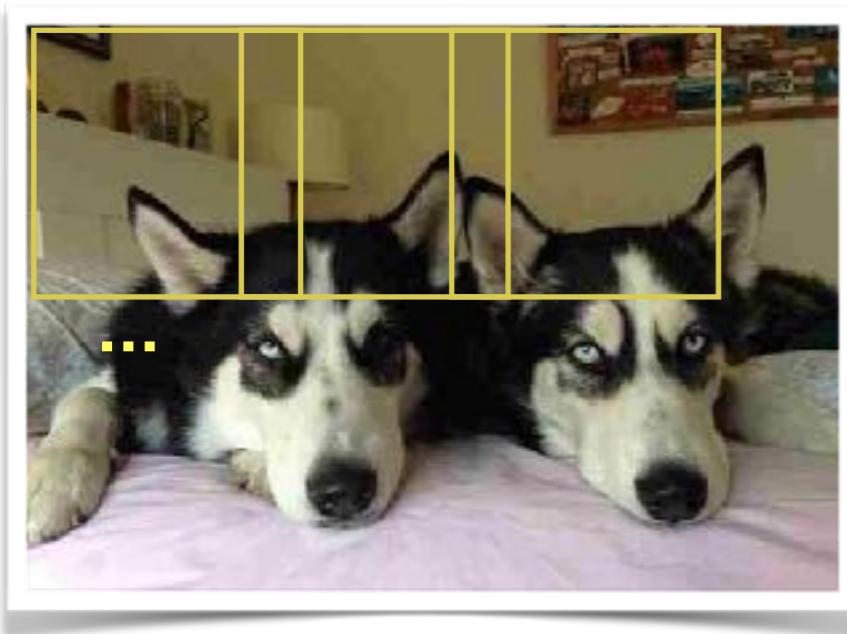
4. 使用CNN进 行目标检测

4.1 滑窗法

滑窗法

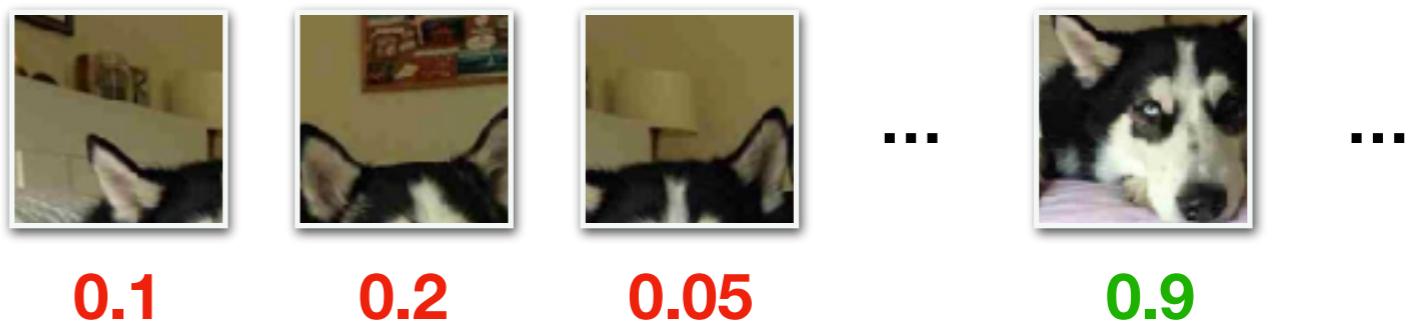
滑窗（Sliding Window） 法是利用已经训练好的分类器，在图像上按照一定间隔和不同大小窗口提取子图进行识别的方法。滑窗法是一种朴素、简单但效率低下的目标检测算法。

使用分类CNN执行滑窗法



步骤：

1. 训练分类的CNN模型；
2. 利用滑窗提取进行识别。



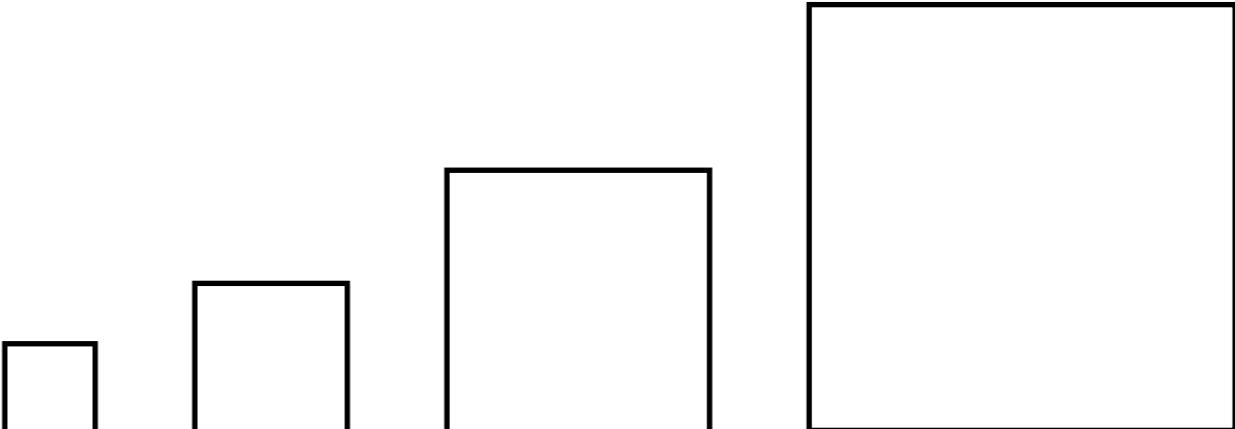
训练分类CNN的注意事项

- 训练CNN的输入图片中，尽量使得目标出现在中心位置；
- 确保不同类别的数据比例不出现严重失衡；
- 当只有一种类别时，此CNN可用二分类（或目标定位方法）任务完成，即目标、背景两种类别；
- 当有n种类别时，此CNN可用n+1分类任务（或目标定位方法）完成，多出的一种为背景类别。

滑窗操作的注意事项

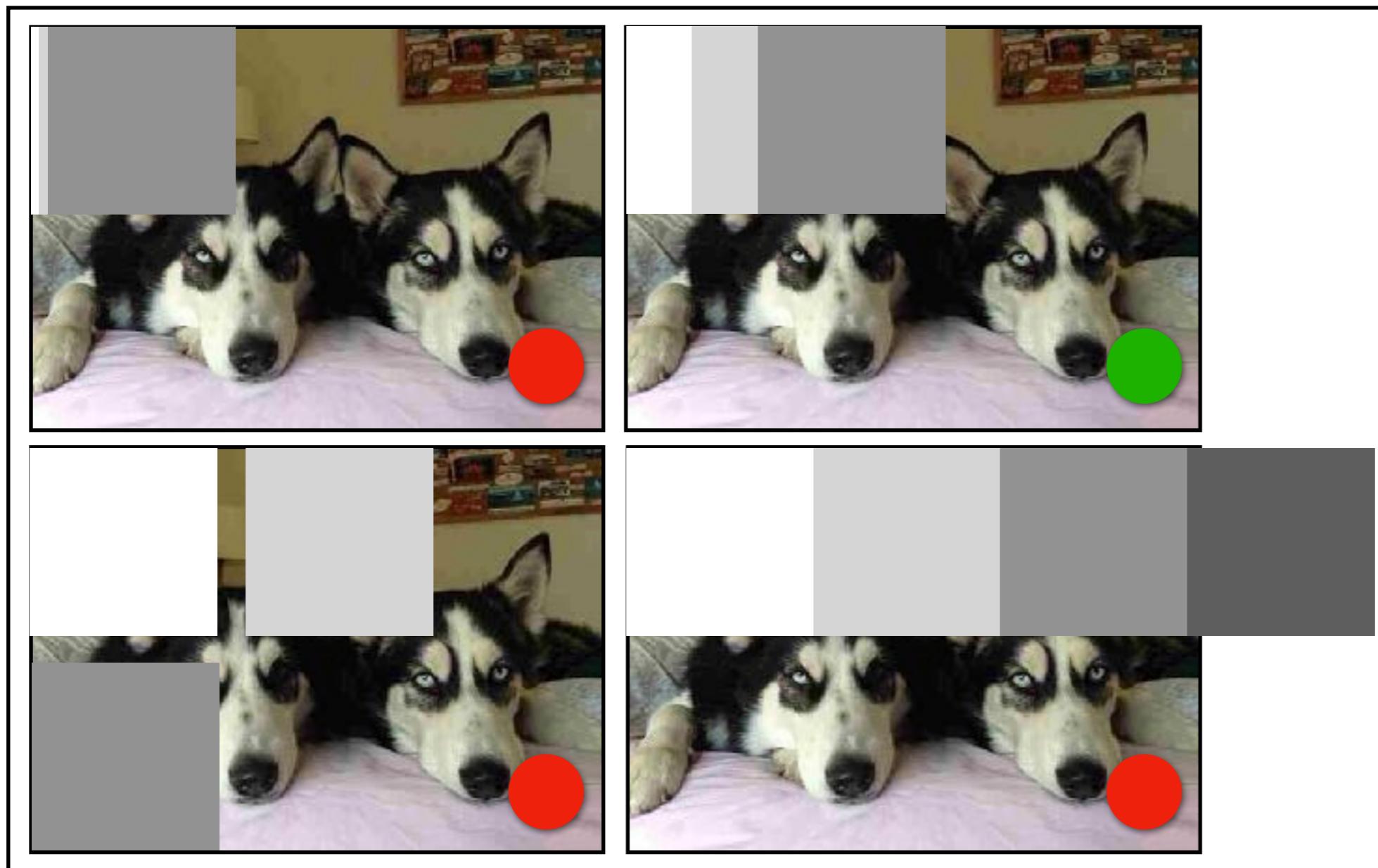
- 需要选取不同的大小的滑窗进行区域提案；
- 不同大小的滑窗提取到的不同大小的子图，可进行缩放后放入CNN进行识别；
- 滑窗步长越小，提取到目标的几率越大，但计算量更高；
- 滑窗形状的比例通常是固定的。

选择滑窗大小



1. 通常选择不同大小的滑窗提作为候选区域的窗口，以此满足提取不同大小的目标的需求；
2. 窗口的比例通常是固定的；
3. 不同大小滑窗提取到不同大小的子图，需要缩放之后使用CNN进行分类。

选择滑窗步伐



1. 随着滑窗步伐的减小，计算量成指数倍增加，通常不使用太小的步伐。
2. 通常，滑窗的步长小于滑窗大小，并且不会使滑窗超出图片边界。

使用滑窗法进行预测

1. 对输入图片使用不同大小的窗口提取子图；
2. 将这些子图全部送入CNN模型进行结果预测；
3. 使用非极大值抑制法去除重叠度较大的窗口。

滑窗法小结

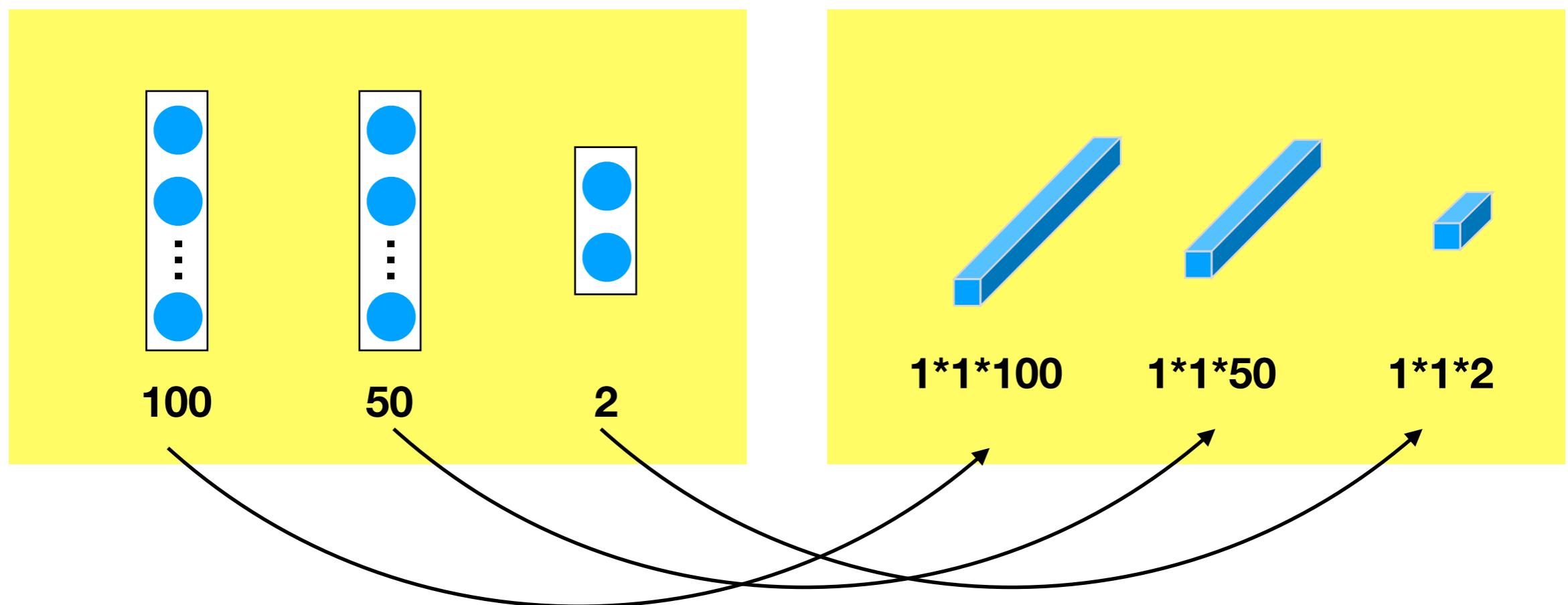
- 滑窗法即在图片上使用滑窗提取候选区域，再使用已经训练好的分类器对每一个候选区域进行分类；
- 滑窗法的计算量与滑窗的大小、数量、滑动步长、分类模型等有关系；
- 通常滑窗法使用线性分类器作为分类模型，而不直接使用CNN；
- 模型预测时，需要使用NMS消除冗余窗口。

4.2 改进滑窗法 中的CNN

4.2.1 改进CNN

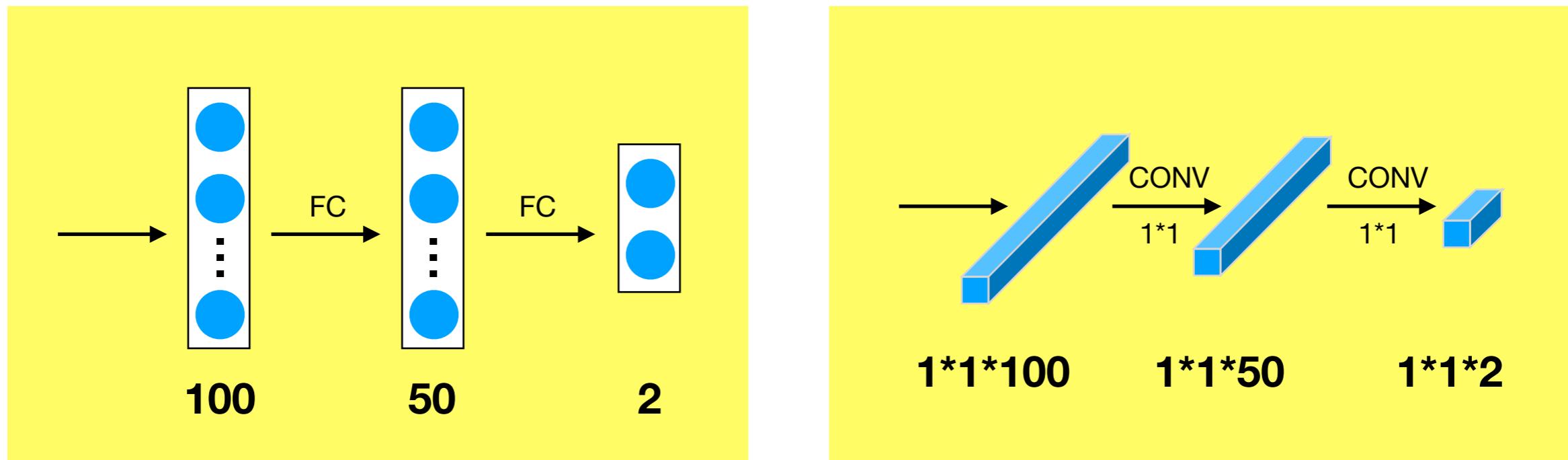
将全连接层转化为卷积层

1. 将全连接层神经元转化为等价的卷积层特征图。



将全连接层转化为卷积层

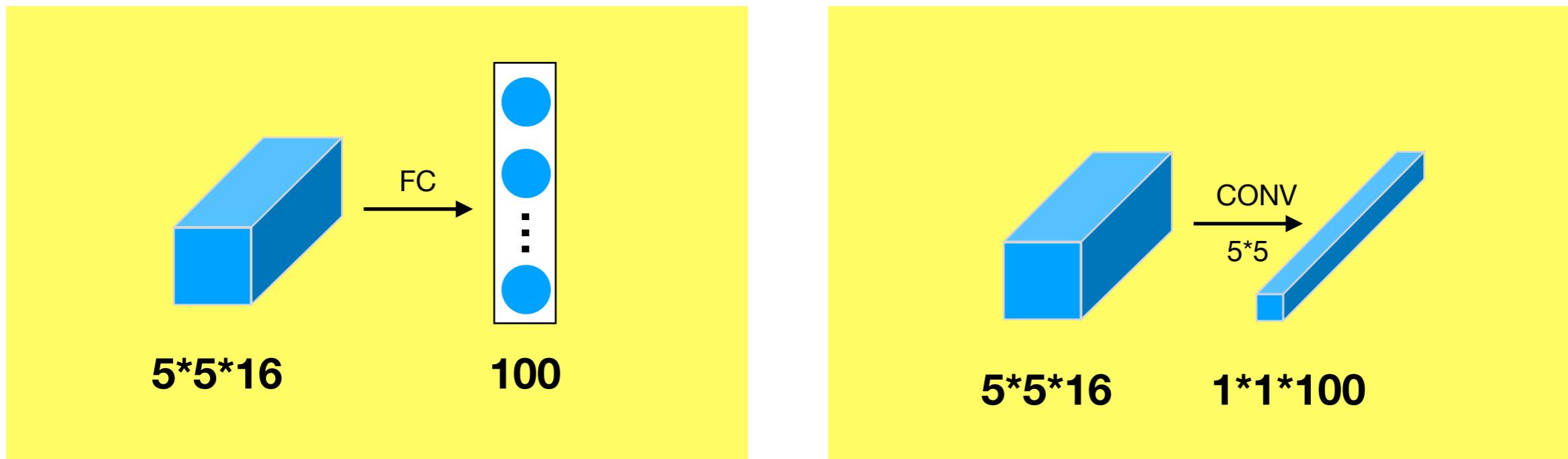
2. 将相连的全连接与全连接层转化为卷积层与卷积层



等价转化：全连接层的神经元可以转化成 1×1 大小的特征图，全连接之后再接全连接层等价于 1×1 大小的特征图之后再进行 1×1 卷积。

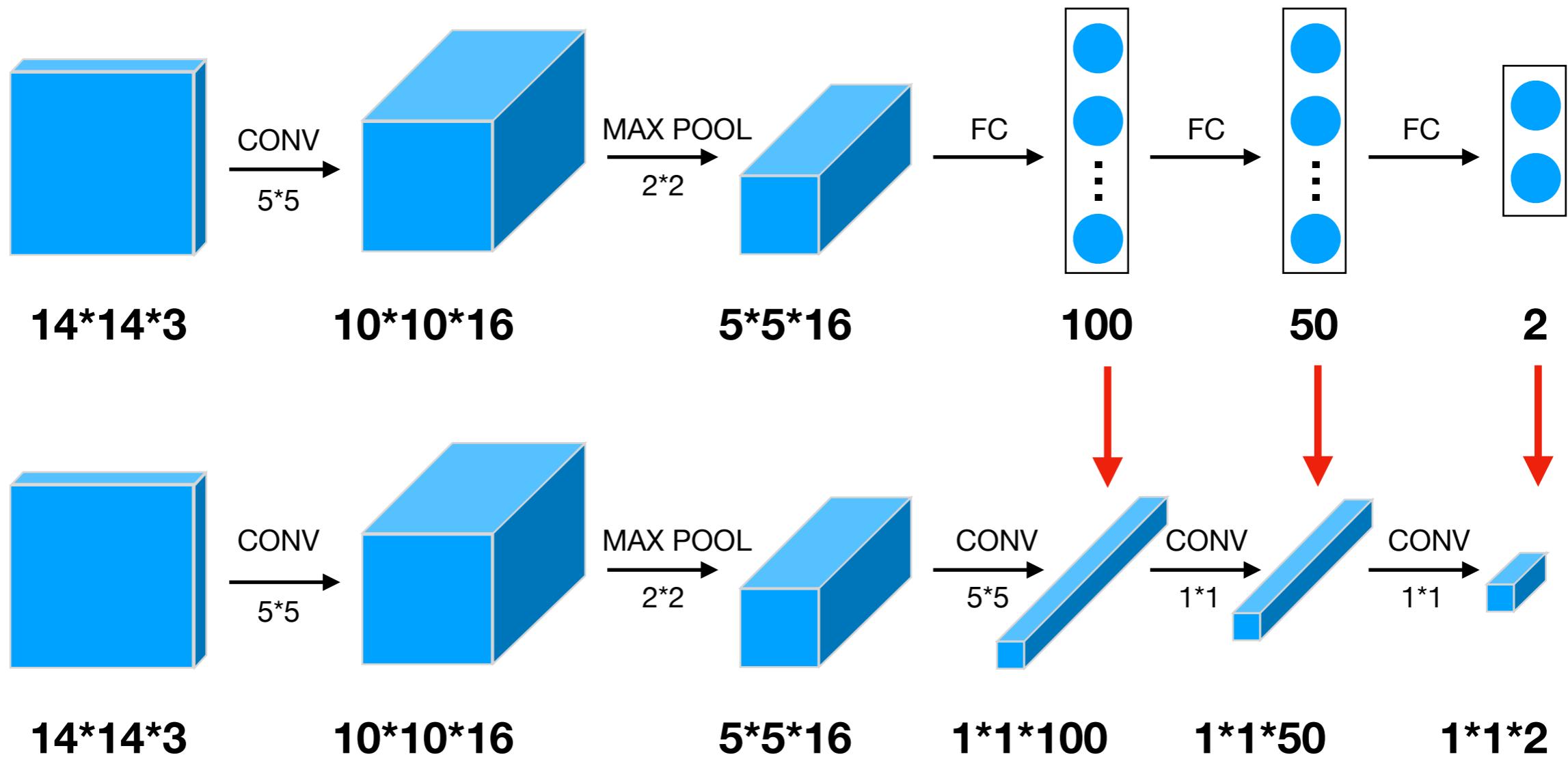
将全连接层转化为卷积层

3. 将与卷积层相连的全连接层转化卷积层。



等价转化：全连接层与卷积特征图相连，即全连接层的每一个神经元分别于卷积特征图的每一个激活值连接。其等价于卷积之后接与输入特征图大小相同的卷积核进行卷积的过程。

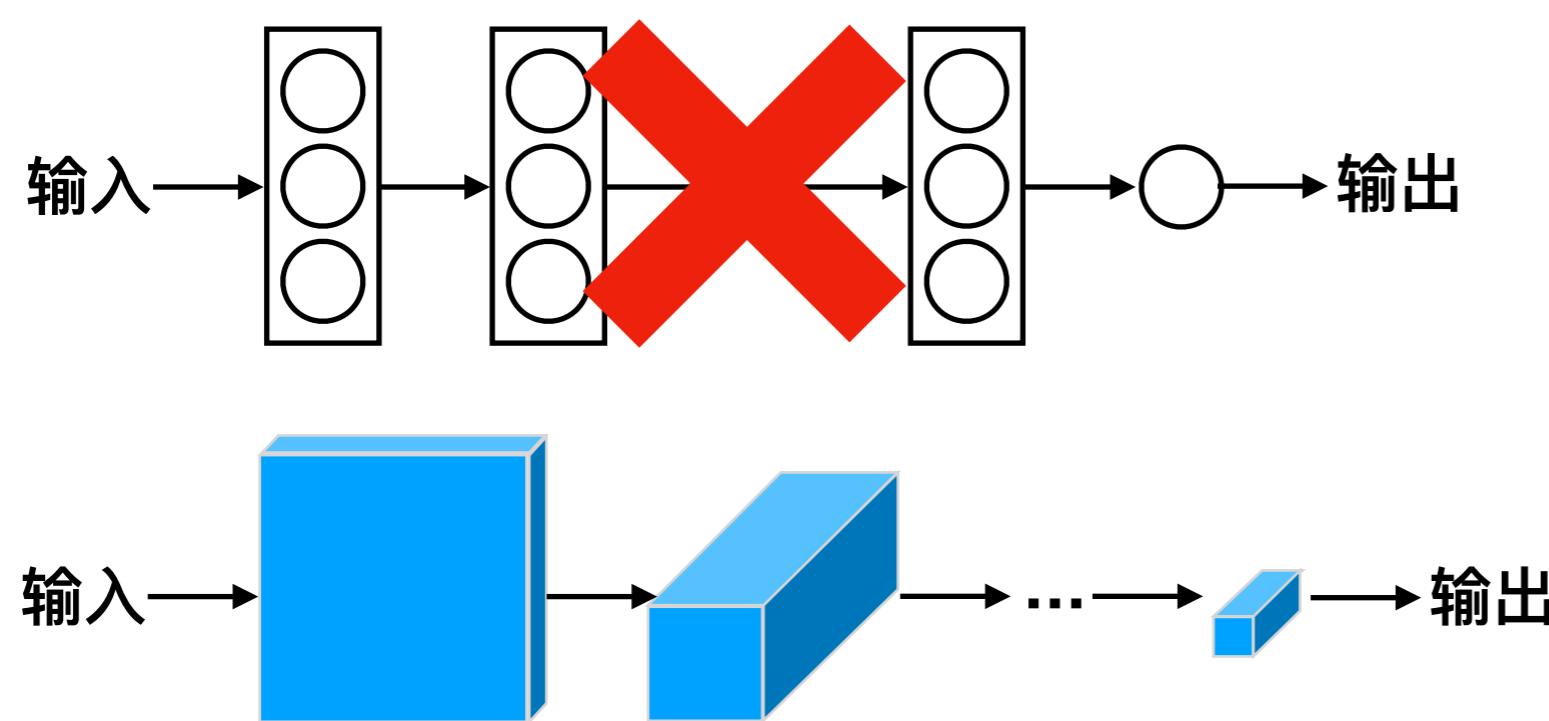
将NN转化为“全卷积”网络



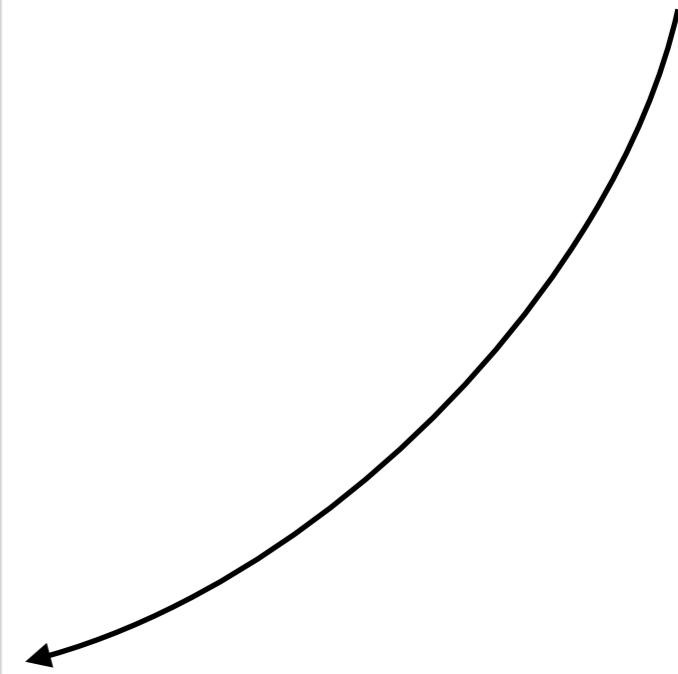
两个模型在数学上等价的。第二个模型的输出层shape变换之后，可认为两个模型完全一致。

4.2.2 使用改进CNN执行滑窗法

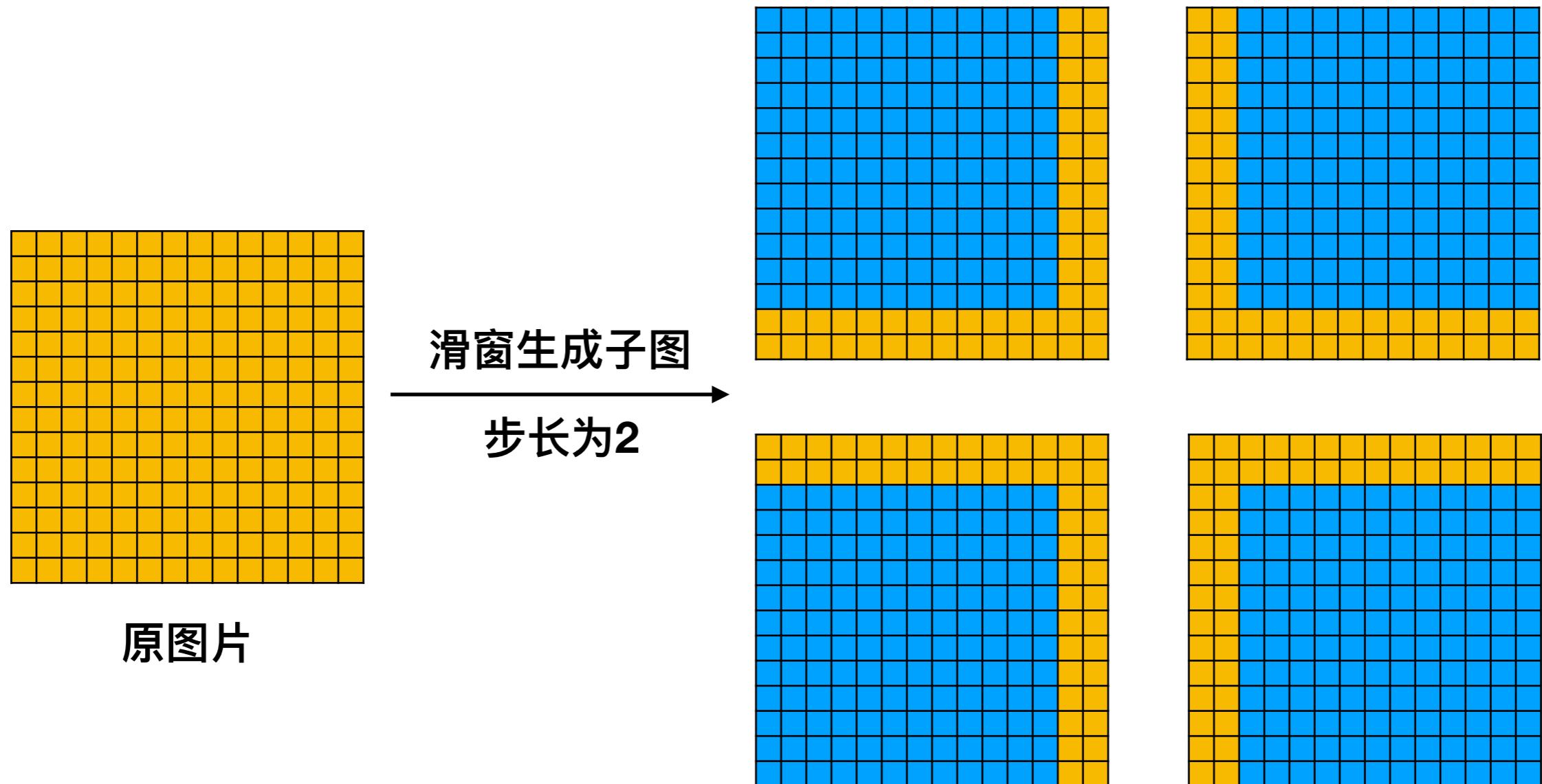
使用“全卷积”网络作为分类网络



训练时：使用上述得到的“全卷积”神经网络代替传统网络作为分类模型，进行训练。

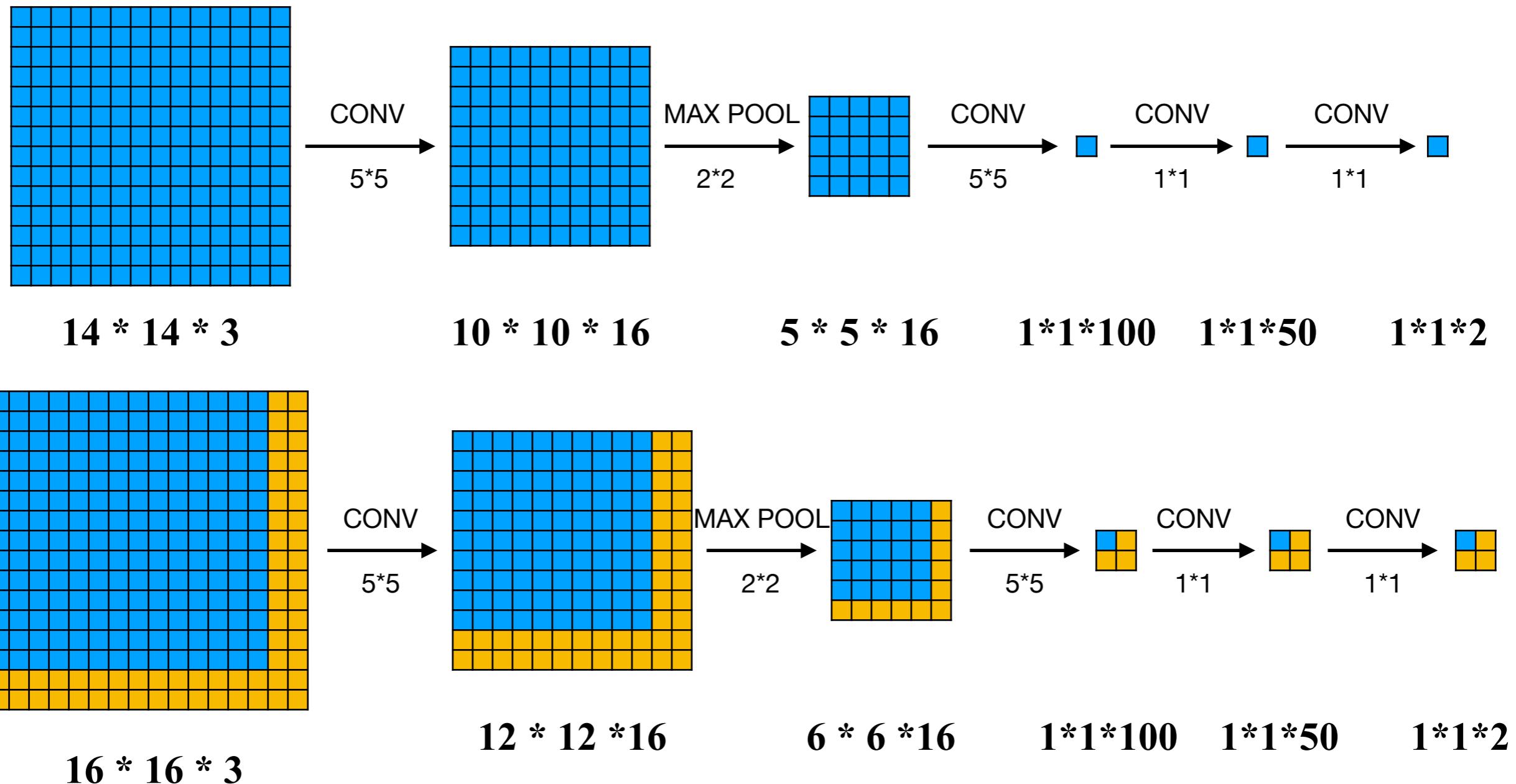


使用“全卷积”网络进行预测



传统的滑窗法，需要将图片的所有子图分别送入分类模型，判断是否为背景。

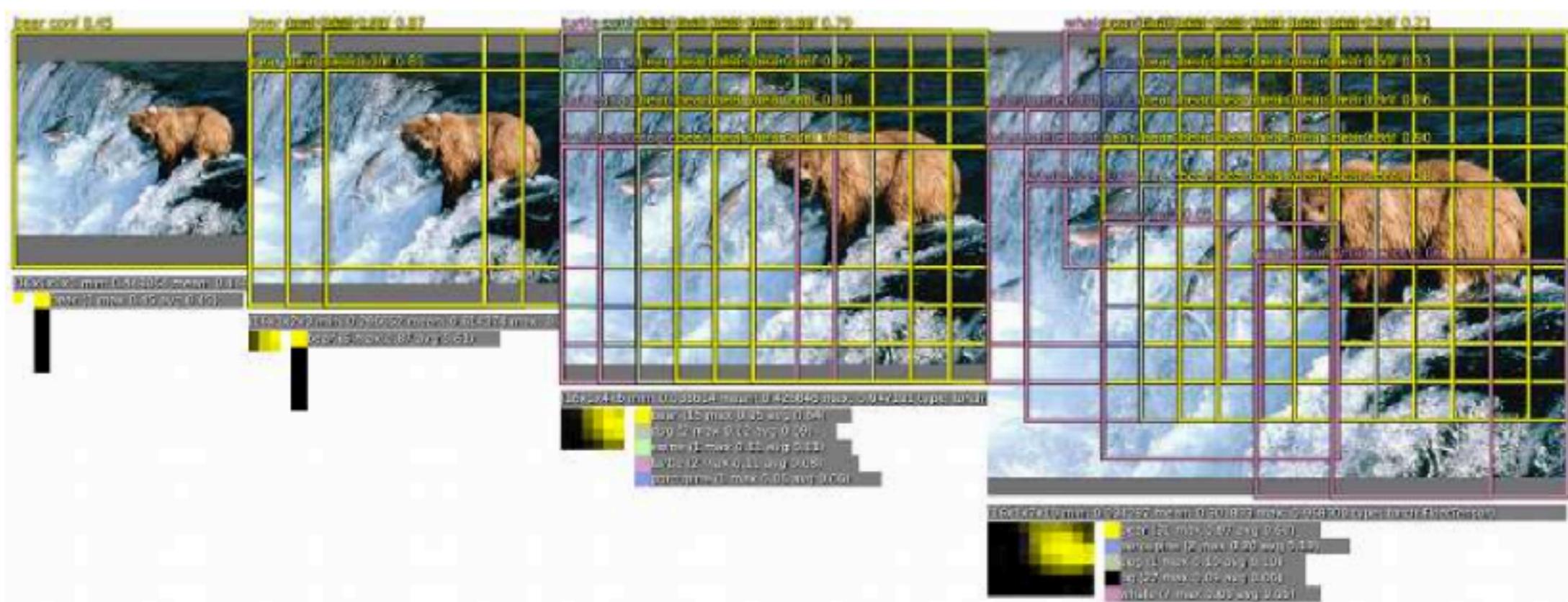
使用“全卷积”网络进行预测



将原图直接输入分类所用的“全卷机”网络进行预测，即可得到不同滑窗对应的分类结果。

思考：滑窗法往往需要使用不同大小的窗口进行滑动，此处相当于使用了固定大小的窗口进行滑动，改如何解决此问题？

答：对输入图片进行缩放。



使用“全卷积”网络的特点

- 训练时，训练分类模型，预测时将整张图片送入分类的“全卷机”网络进行预测，其代表模型是OverFeat；
- 网络输出层的特征图的每一个区域都代表一个预测结果；
- 只需要一次前向传播即可完成一类滑窗口大小对应的所有子图预测；
- 需要对原图进行不同大小的缩放来满足不同大小的滑窗对候选区域的提取；
- 在预测时，往往回得到大量的候选窗口，需要使用NMS算法去除重复窗口。

4.3 其它改进

RCNN的改进

- RCNN使用Selective Search的方法进行区域提案，即使用分割算法生成小区域，再使用超像素合并的方法使之成为大区域，最终将这些区域作为候选区域。与滑窗法相比，生成的候选窗口数量大大降低（通常为2000个左右）。
- RCNN没有使用“全卷积”网络的方法减少前向传播的次数，所以RCNN速度很慢，但比简单使用CNN滑窗法提高了很多。

SPPNet的改进

- SPPNet类似于OverFeat使用卷积层提取特征，并利用RCNN中提出的Selective Search方法选择候选区域并映射到最后一个卷积层的特征图得到ROI区域，再使用简单的分类网络对ROI进行分类；
- SPPNet提出空间金字塔池化（Spatial Pyramid Pooling, SPP）使得ROI区域转化为固定大小的维度，便于送入分类模型。

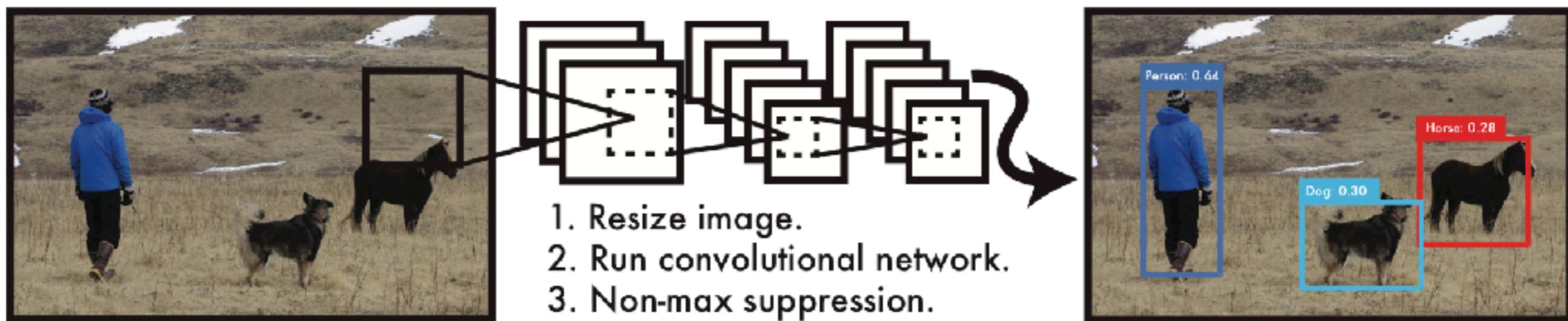
5. YOLO

3.1 YOLO简介

YOLO简介

YOLO (You Only Look Once) 是一种端到端的目标检测算法，即只需要一个模型，运行一次前向传播，就能完成目标定位与目标分类的任务。YOLO拥有很好的性能，可以达到每秒45帧的速度实时处理图像。较小版本的Fast YOLO更是达到了每秒155帧的速度，虽然误差也高了一些。

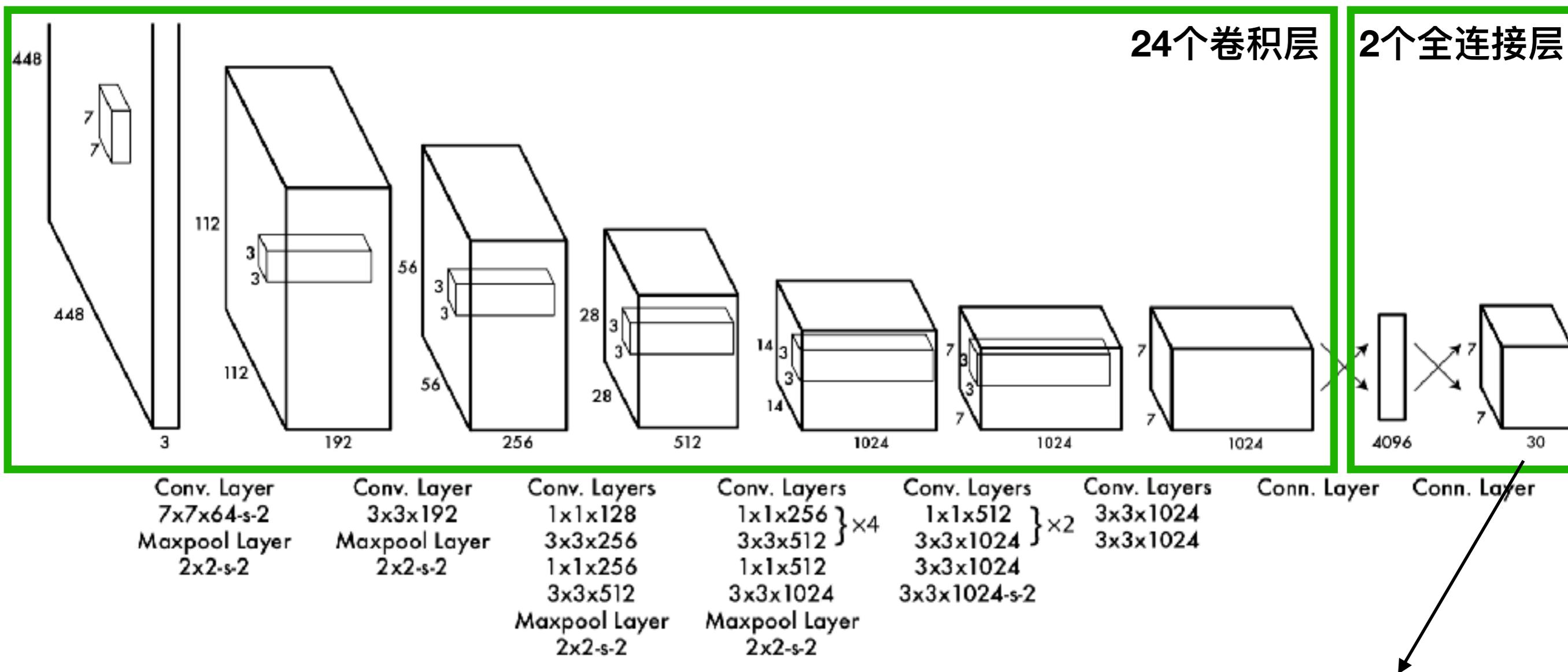
YOLO简介



YOLO使用流程：

1. 将输入图片resize为448*448大小；
2. 在图像上运行单个卷积网络；
3. 根据模型输出的置信度利用NMS算法对检测结果进行过滤。

YOLO模型结构概览



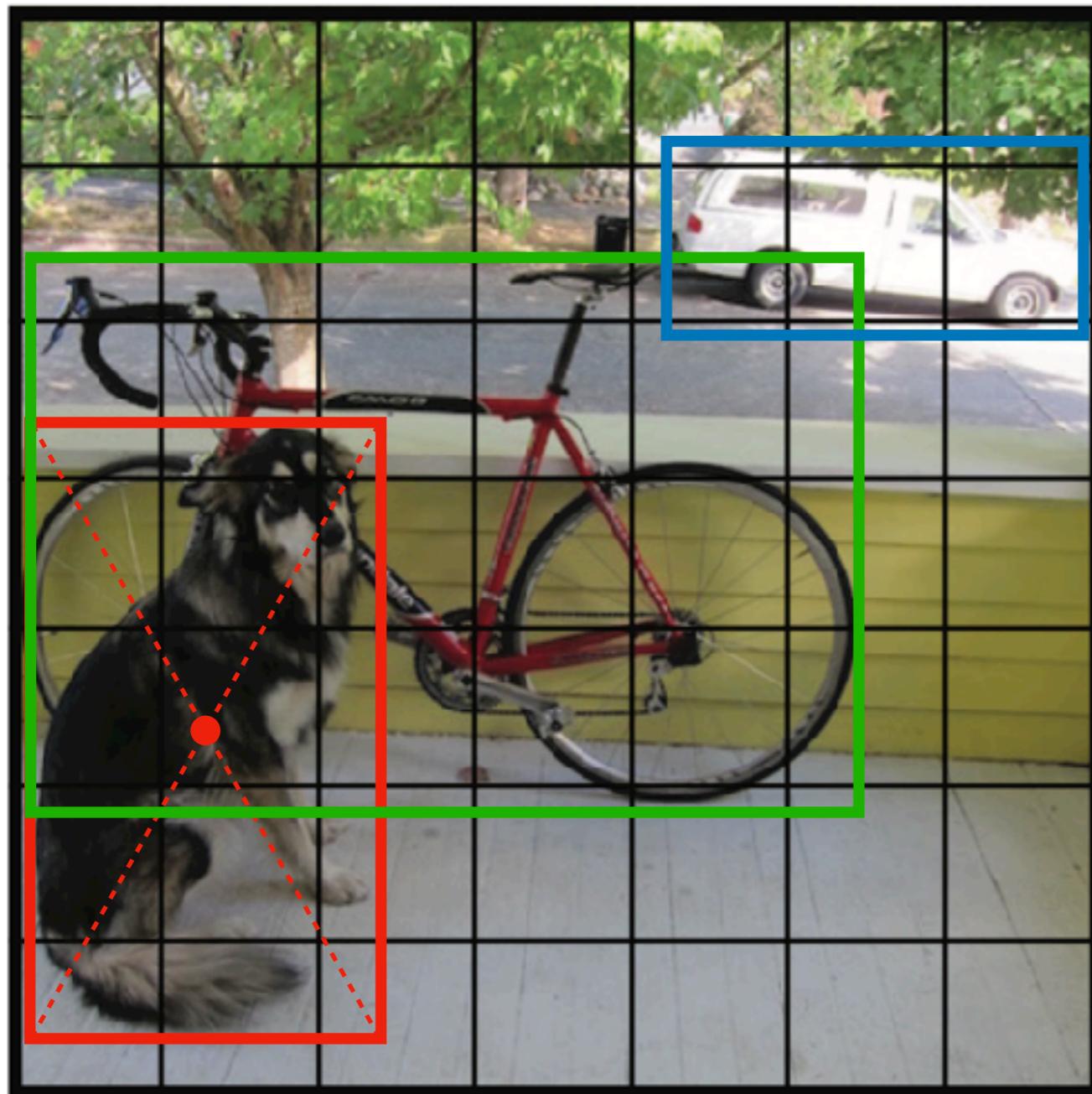
输出为 $7 \times 7 \times 30$ 个结果，即预测图片 7×7 个位置中每一个位置是否有目标以及目标的具体位置与类别。

YOLO与分类CNN相比

- 从整体结构上看，YOLO就是CNN的一种具体实现；
- YOLO与分类CNN都是输入一张图片；
- YOLO的输出层虽然是全连接层，但reshape成为了4阶张量，用于输出多个目标的信息；
- YOLO的代价函数更复杂，包含了置信度、分类、坐标3部分代价。

3.2 模型要义

输入与输出表示关系



$S \times S$ grid on input

1. 将输入图像分成 $S \times S$ 个格子；
2. 每个格子最多负责检测一个目标；
3. 物体的中心位置所在的格子负责检测目标。

论文中： $S=7$

思考

- 当 $S=7$ 时，最多可检测多少个目标，当 $S=19$ 时呢？

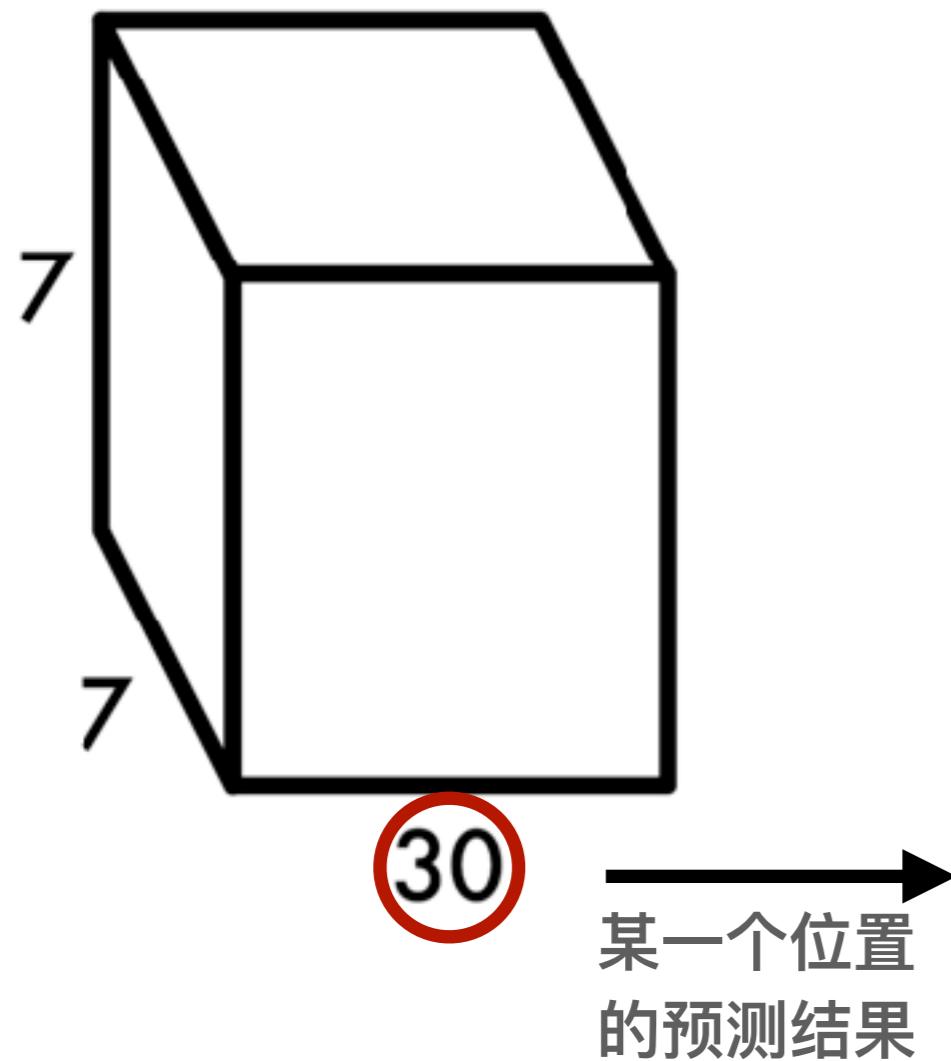
49个、361个

- 当两个或多个物体中心位置落入一个格子时，如何检测？

只能检测其中的一个目标，另一个无法检测。

YOLO输出层

YOLO模型输出层



- yolo共输出 $S \times S$ 个位置中对象的信息；
- 每个位置预测对应的C个类别的条件概率；
- 每个位置预测B个BBox；
- 每个BBox对应5个输出值，包括BBox的坐标与大小 (x, y, w, h) 以及置信度。



格子输出的C个类别的定义

每个格子输出C个类别（论文中C=20），代表了格子中存在目标的条件下目标分别是每个类别的条件概率，即每个格子输出类别定义为： $P(\text{Class}_i | \text{object})$ 。

BBox的定义

通常，BBox包含中心位置坐标与宽高信息，为了表述方便，此处将BBox对应的置信度也认为是BBox的属性。每个格子对应B个BBox（论文中B=2），**每个BBox的定义如下：**

x：中心位置坐标距离所在格子左边的距离，值域为[0, 1]；

y：中心位置坐标距离所在格子上边的距离，值域为[0, 1]；

w：目标的宽，值域为[0, 1]；

h：目标的高，值域为[0, 1]；

confidence：置信度，值域为[0, 1]。

置信度定义

每个BBox包含一个置信度，反映了当前BBox包含目标的概率和BBox预测的准确度，所以置信度定义为：

$$\text{confidence} = P(\text{object}) * \text{IOU}$$



可以看到，置信度越大，对BBox包含目标且预测准确的信心越大；反之，置信度越小，对BBox包含目标且预测准确的信心越小。

利用置信度过滤无效输出

根据：

格子输出类别的条件概率： $P(\text{Class}_i | \text{object})$

每个BBox的置信度： $\text{confidence} = P(\text{object}) * \text{IOU}$

可以得到：

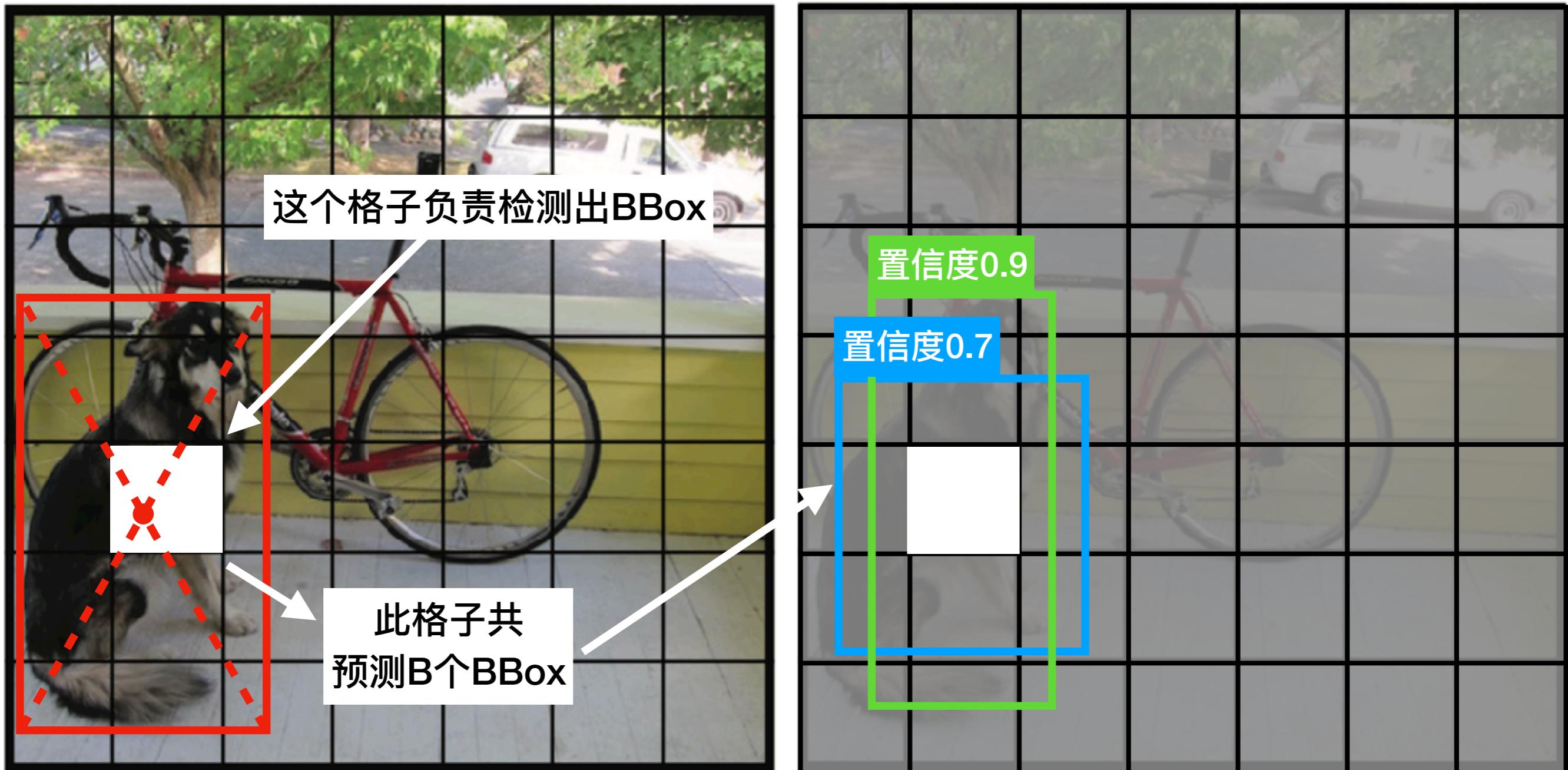
每一个BBox中：①包含某一类目标的概率 ②此BBox预测为某个类别概率：

$$P(\text{Class}_i | \text{object}) * P(\text{object}) * \text{IOU} = P(\text{Class}_i) * \text{IOU}$$

其小于某个阈值时（论文中为0.2），我们认为其不包含此类别目标。

YOLO输入与输出 对应关系

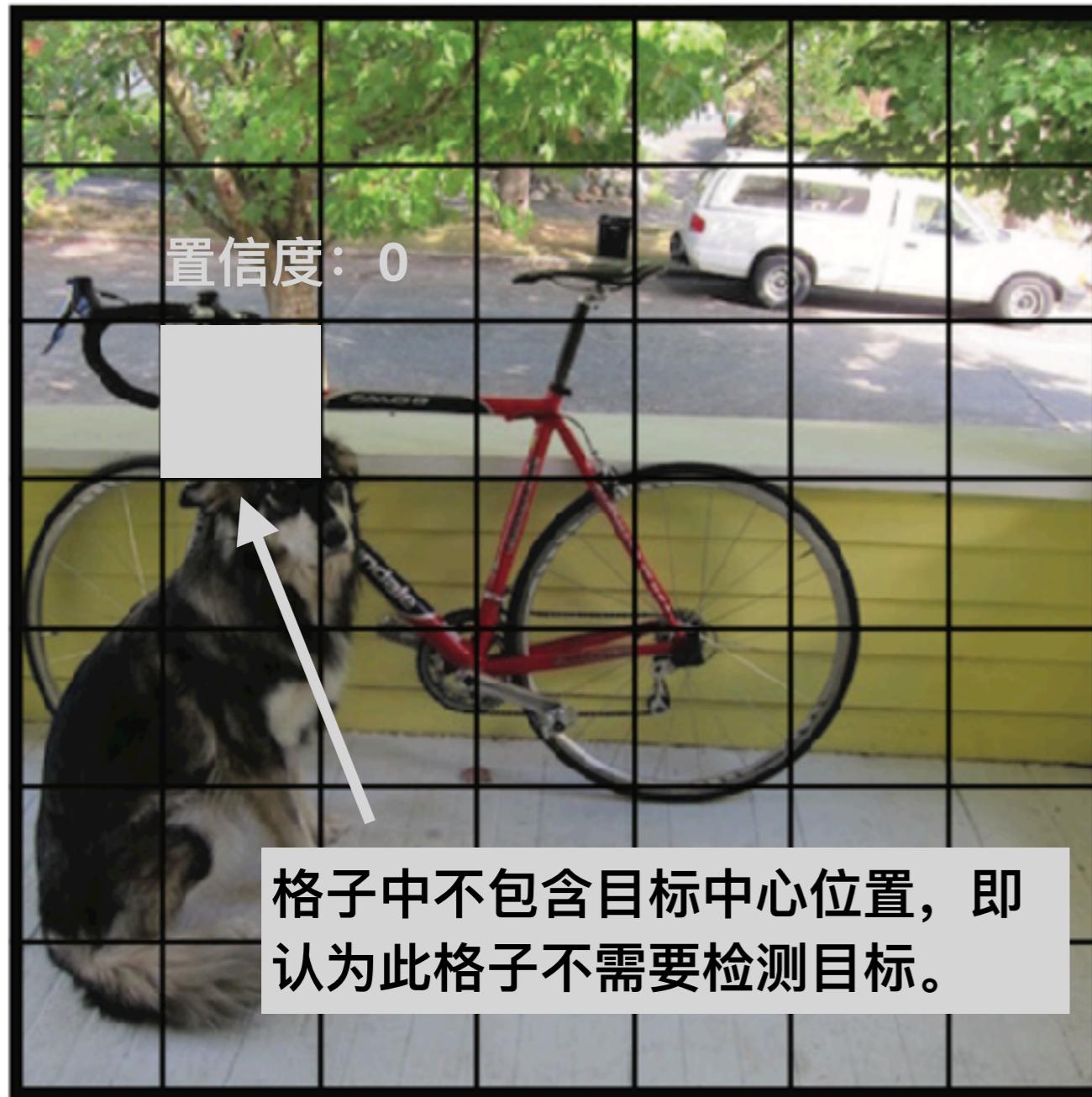
对于包含目标的格子



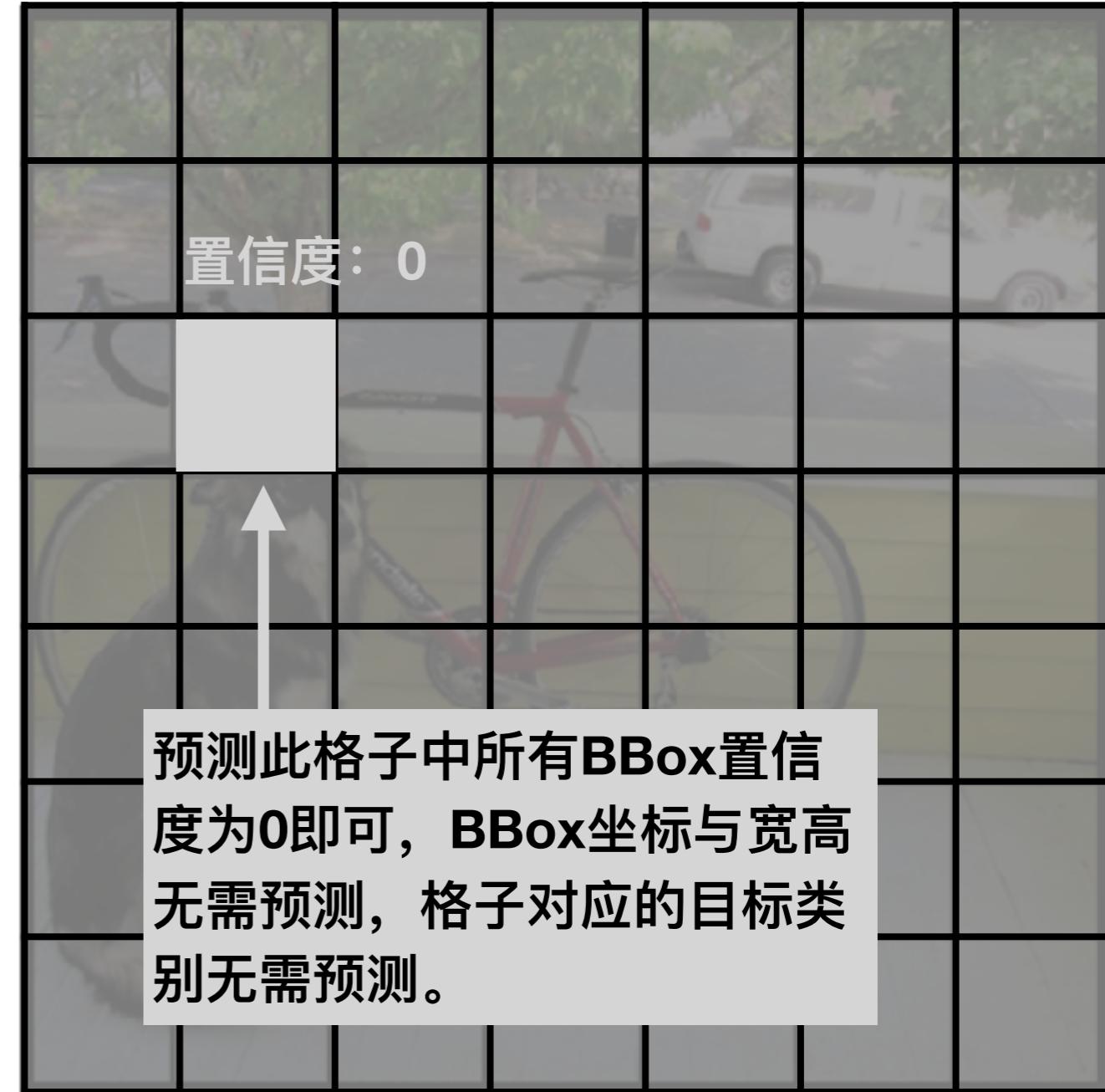
输入数据：确定了哪个格子负责检测目标

模型输出：BBox根据置信度与类别条件概率确定

对于不包含目标的格子



输入层



输出层

3.3 代价函数

代价函数

$$\text{loss} = \text{CoordError} + \text{IoUError} + \text{ClassesError}$$

坐标误差

置信度误差

分类误差

坐标误差: bbox中心点坐标误差与bbox宽、高误差；

置信度误差: 预测置信度与IOU之间的误差；

分类误差: 格子预测类别的误差。

坐标误差

坐标误差包括两部分：中心点坐标误差与BBox宽高误差。

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

注意：

1. 使用均方误差误差代价函数；
2. 只对包含目标的BBox计算代价（一个格子中可能包含多个BBox，此时只计算IOU值最大的BBox代价）。
3. 宽、高开平方后再计算误差，减缓大的BBox产生较大代价的影响。

置信度误差

我们期望预测的置信度反映了：

- ① 是否包含目标（当BBox中不存在目标时，期望置信度为0）；
- ② 物体位置的准确性（当BBox中存在目标时，期望置信度为IOU值）。

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

在YOLO所使用的VOC数据集中，每张图片对应的目标较少，通常仅有不超过5的目标，所以包含目标与否的窗口数量不成比例，所以通常会加入一个超参数来调整二者之间对loss的贡献，如下：

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

分类误差

分类误差使用均方误差代价函数，只计算包含目标的格子输出类别的代价：

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

注意：YOLO所使用的VOC数据集是20分类任务，所以分类误差所占的比重较大，也需要调整其占总代价的比例，论文中，作者调节的是坐标误差而没有调节分类误差所占笔记。

完整代价函数

loss = CoordError + IouError + ClassesError

$$\begin{aligned} &= \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{\text{obj}} \left[\left(x_i - \hat{x}_i \right)^2 + \left(y_i - \hat{y}_i \right)^2 \right] + \\ &\quad \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] + \\ &\quad \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 + \\ &\quad \sum_{i=0}^{S^2} 1_{ij}^{\text{obj}} \sum_{\hat{c} \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \end{aligned}$$

3.4 模型训练

YOLO模型训练

1. 预训练。使用 ImageNet 1000分类数据训练YOLO模型的前20个卷积层（训练时增加平均池化、全连接层，模型输入图片的大小为 $224 * 224$ ）。
2. YOLO训练。使用预训练中前20个卷积层网络参数初始化 YOLO模型前20个卷积层的网络参数（输入图片大小为 $448 * 448$ ）。

3.5 模型预测

模型预测

1. 将输入图片resize到 448 * 448 大小；
2. 将图片输入模型得到输出；
3. 利用NMS算法去除重复窗口（思考：训练时只训练某一个格子检测一个对象，那么测试时为什么仍然需要NMS去重）；

3.6 模型效果

速度快、性能较高

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	155
YOLO	2007+2012	63.4	45
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

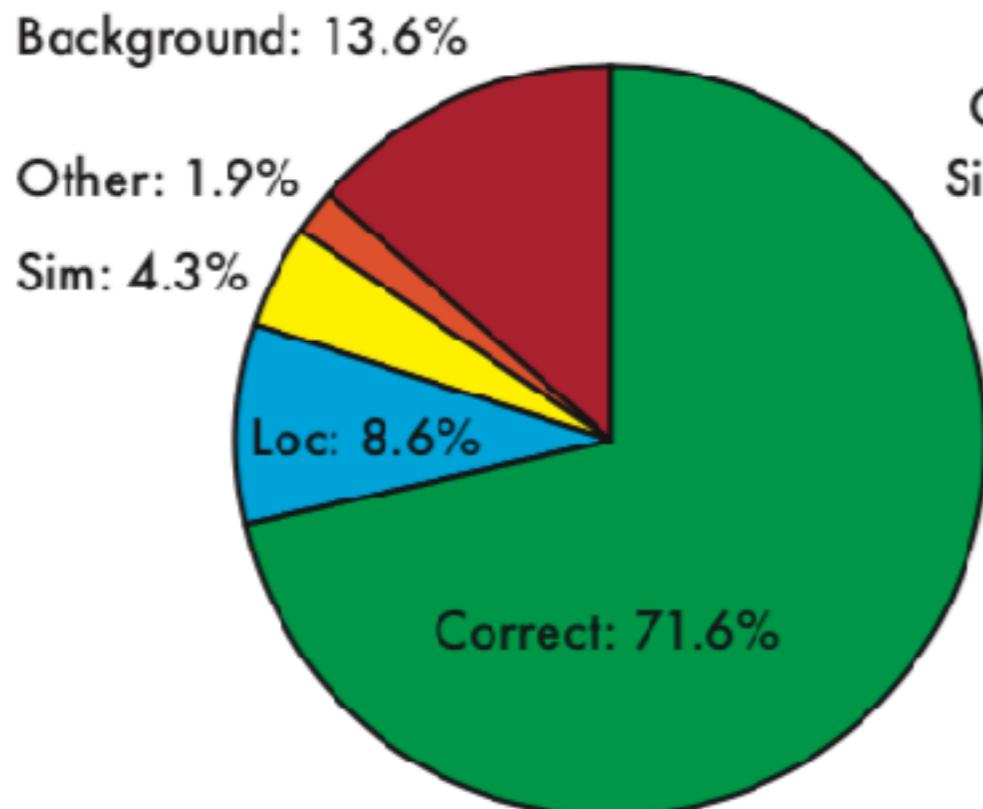
实时检测系统在PASCAL VOC2007数据下的对比，可以看到：

在实时系统中Fast YOLO是速度最快的，YOLO是检测性能最好的系统。

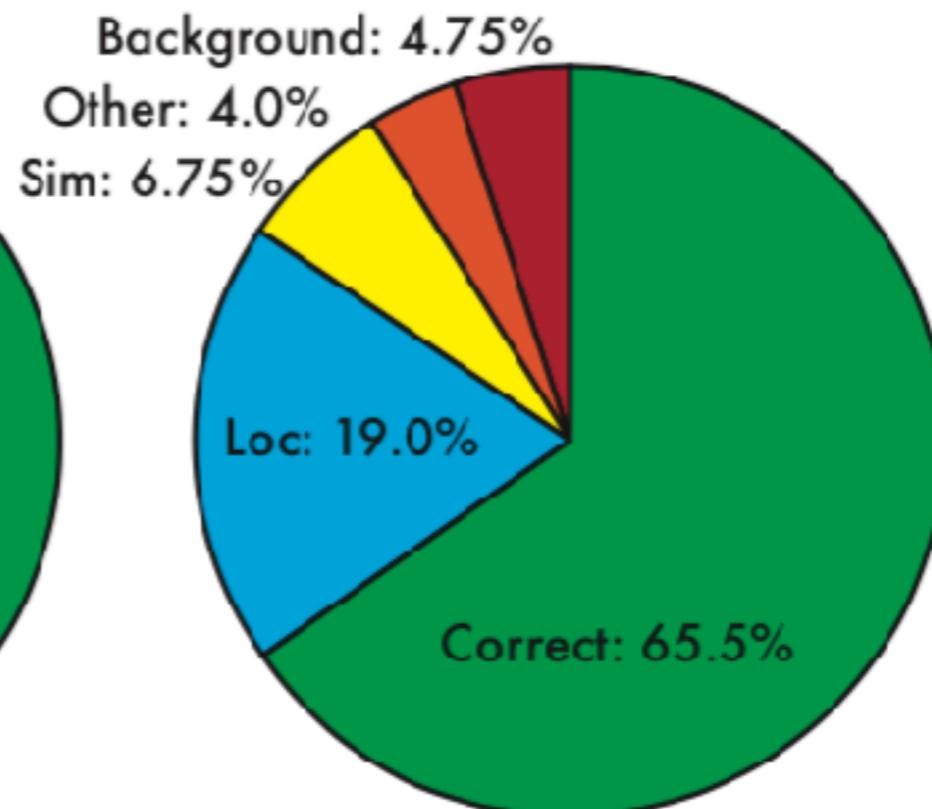
非实时系统中YOLO与Fast R-CNN的性能差距并不是很大，但YOLO的检测速度要快很多。

背景误检率低

Fast R-CNN

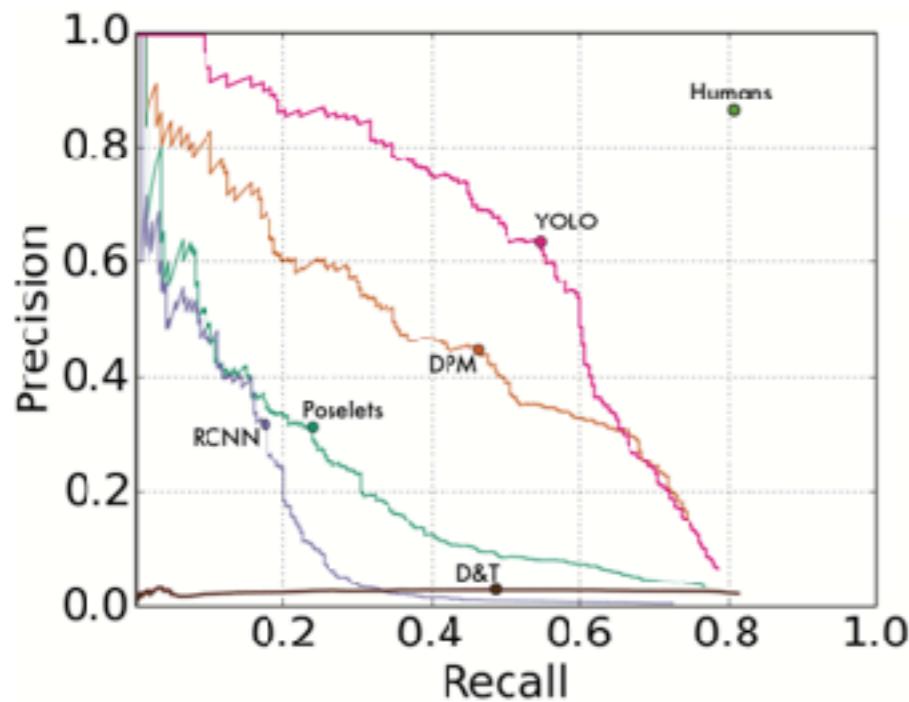


YOLO



与Fast R-CNN相比，YOLO的背景误检率更低，这主要是因为YOLO端到端的模型可以更好的利用上下文信息。与Fast R-CNN相比，YOLO的定位准确率更低，这主要是由于全连接层影响了精度。

更好的“适应能力”

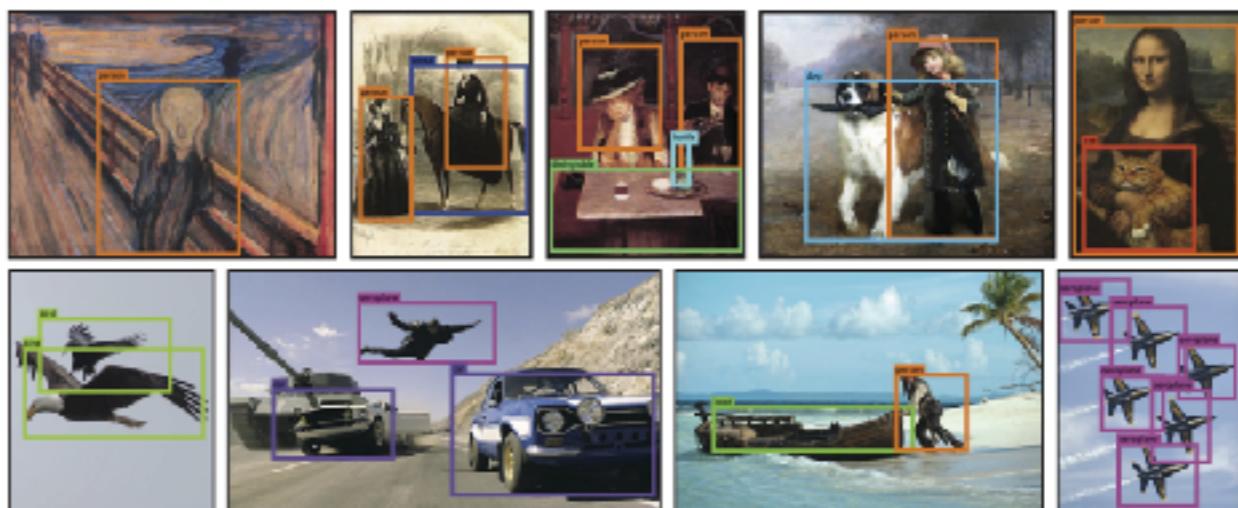


(a) Picasso Dataset precision-recall curves.

	VOC 2007 AP	Picasso		People-Art AP
	AP	Best F_1		
YOLO	59.2	53.3	0.590	45
R-CNN	54.2	10.4	0.226	26
DPM	43.2	37.8	0.458	32
Poselets [2]	36.5	17.8	0.271	
D&T [4]	-	1.9	0.051	

(b) Quantitative results on the VOC 2007, Picasso, and People-Art Datasets.
The Picasso Dataset evaluates on both AP and best F_1 score.

YOLO在艺术作品等数据集
中表现更好。



YOLO优缺点

- 速度快。在Titan X GPU上达到了每秒45帧的检测速度，这意味着可以处理实时视屏流；
- 可以利用上下文信息。与fast RCNN等基于区域提案模型的相比，YOLO可以利用图像的全局信息，背景误检率低；
- YOLO具有更好的泛化能力。相比于DPM和RCNN等方法，YOLO表现出了更好的可迁移特性，这是由于其拥有更好的表示造成的。
- YOLO的检测精度相比（当时）最先进的检测系统要落后一些，尤其对于较小的目标。
- YOLO目标定位的精度相比（当时）最先进的检测系统要落后一些。

TensorFlow实现

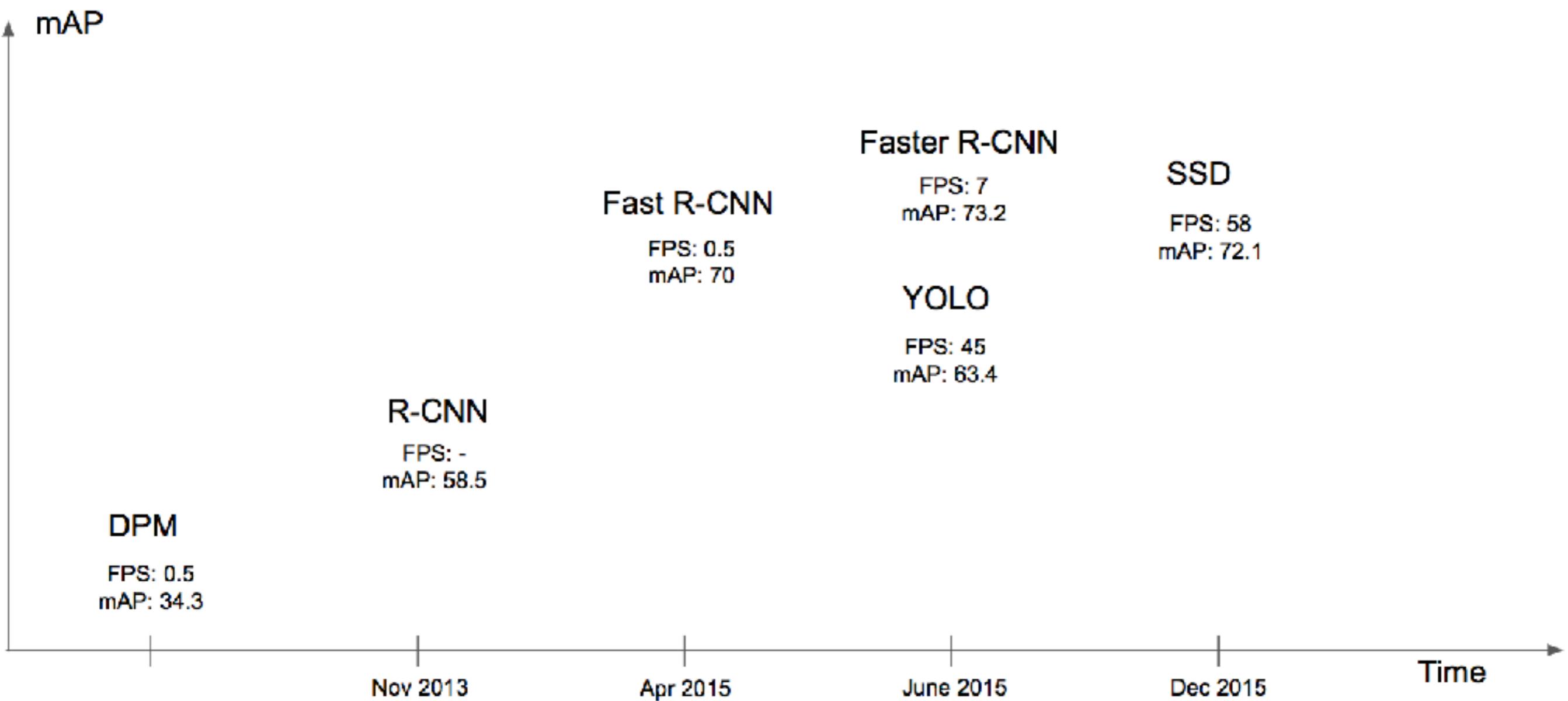
6. SSD

6.1 SSD简介

SSD简介

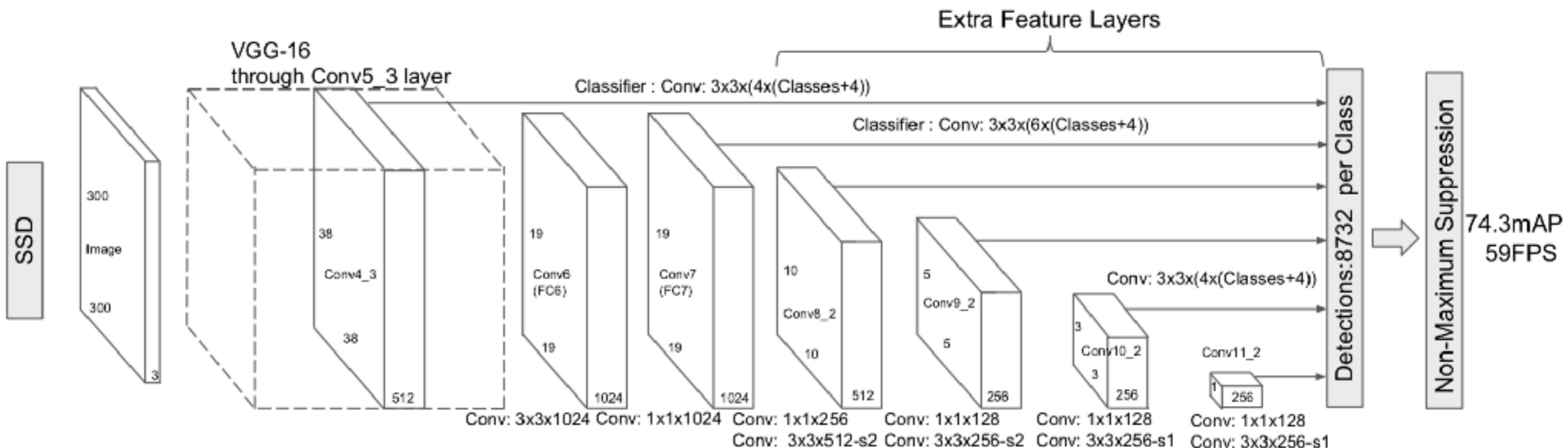
SSD (Single Shot multibox Detector) 是目标检测算法中的集大成者。其借鉴了YOLO算法中基于格子预测目标的端到端模型的思想与Faster R-CNN算法中使用anchor box的思想，并使用多层级特征图进行目标检测，使得模型在保持很高检测精度的条件下达到了实时的检测速度，是目前最好的目标检测算法之一。

SSD性能



在Pascal VOC2007+2012上训练，在Pascal VOC 2007测试的结果。

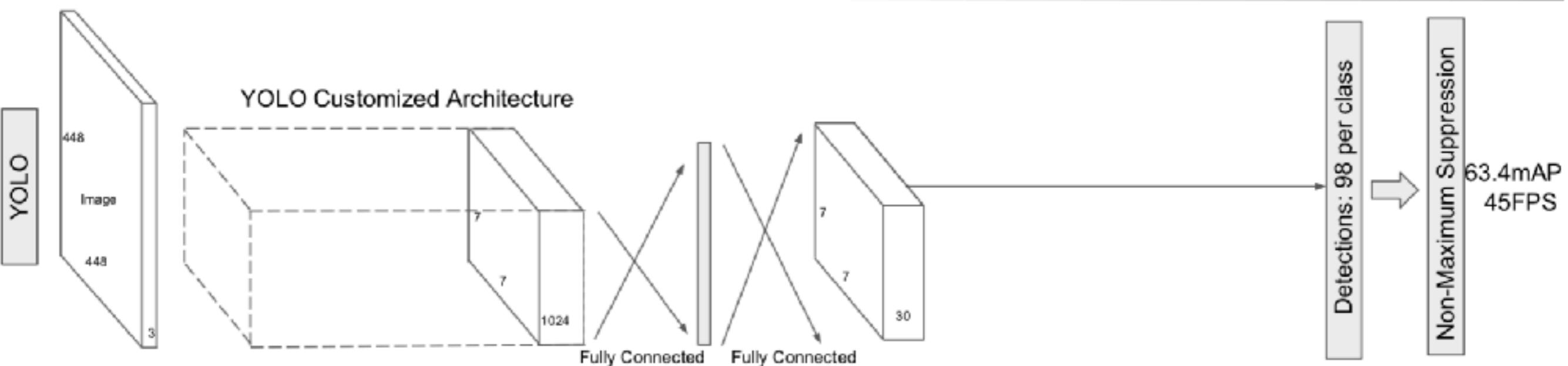
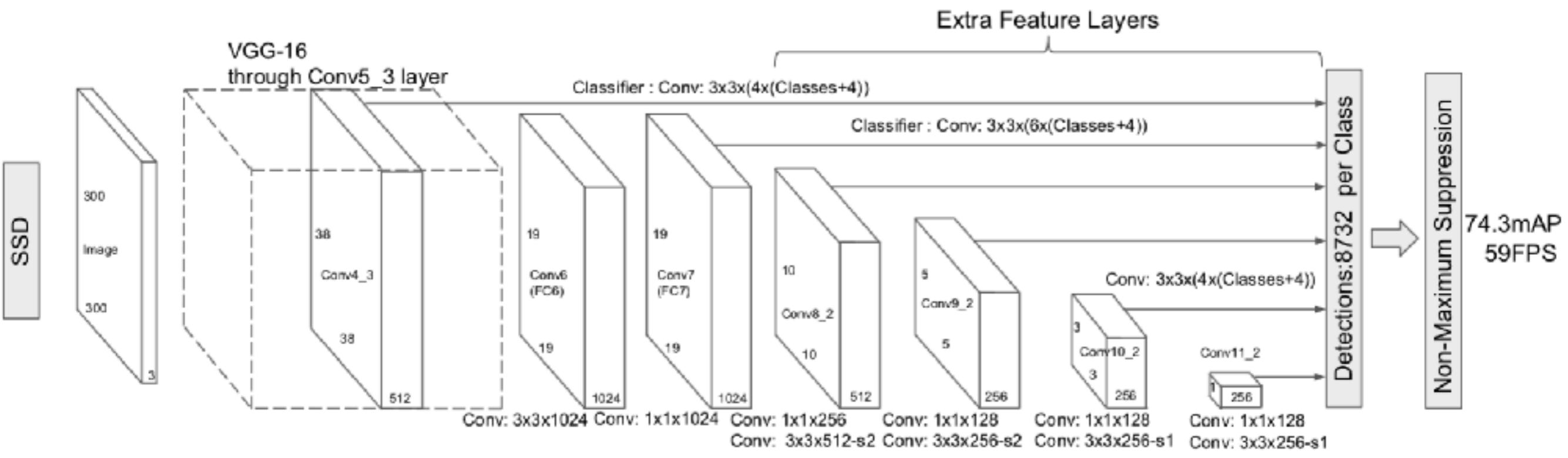
SSD模型总体结构



特点：

1. SSD基于VGG16构建而来，使用了Conv5_3之前的层；
2. SSD模型只使用了卷积层作为参数层，而没有使用全连接等其他类型的参数层；
3. SSD使用了不同的层输出结果；
4. 模型简洁、速度快、端到端构建与训练。

YOLO与SSD对比



6.2 SSD模型运行方式

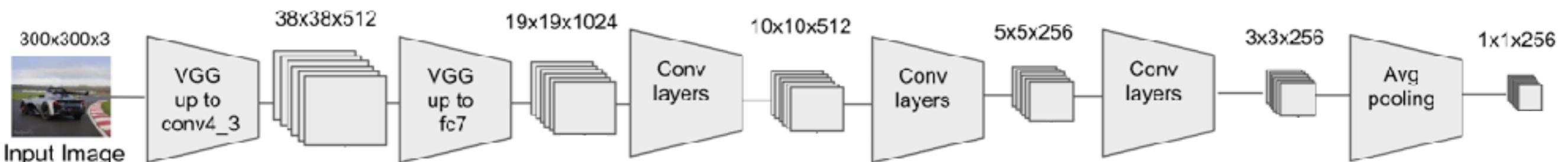
SSD架构：主干结构

300x300x3

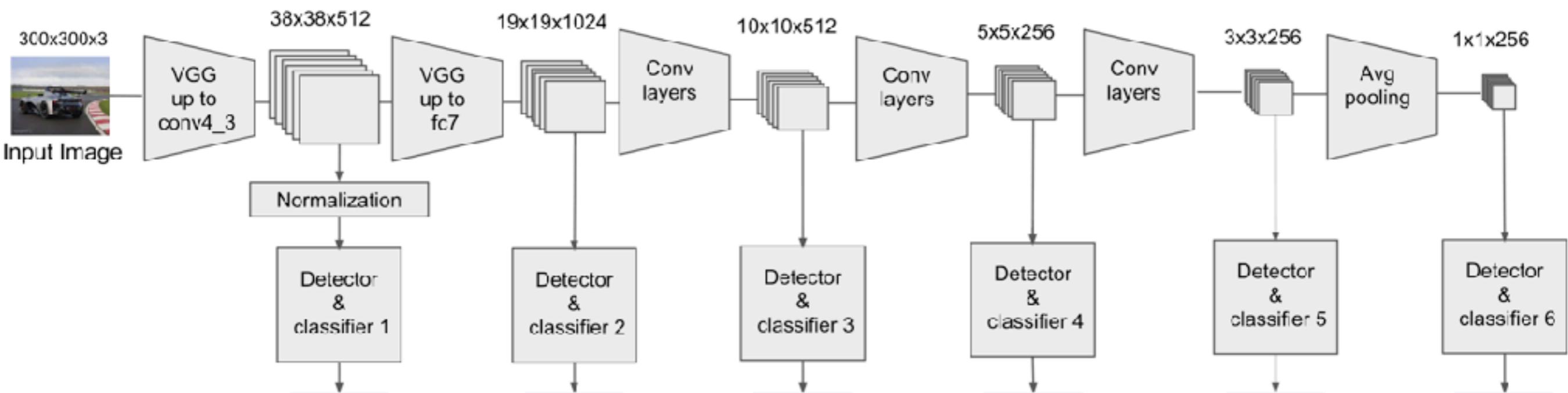


Input Image

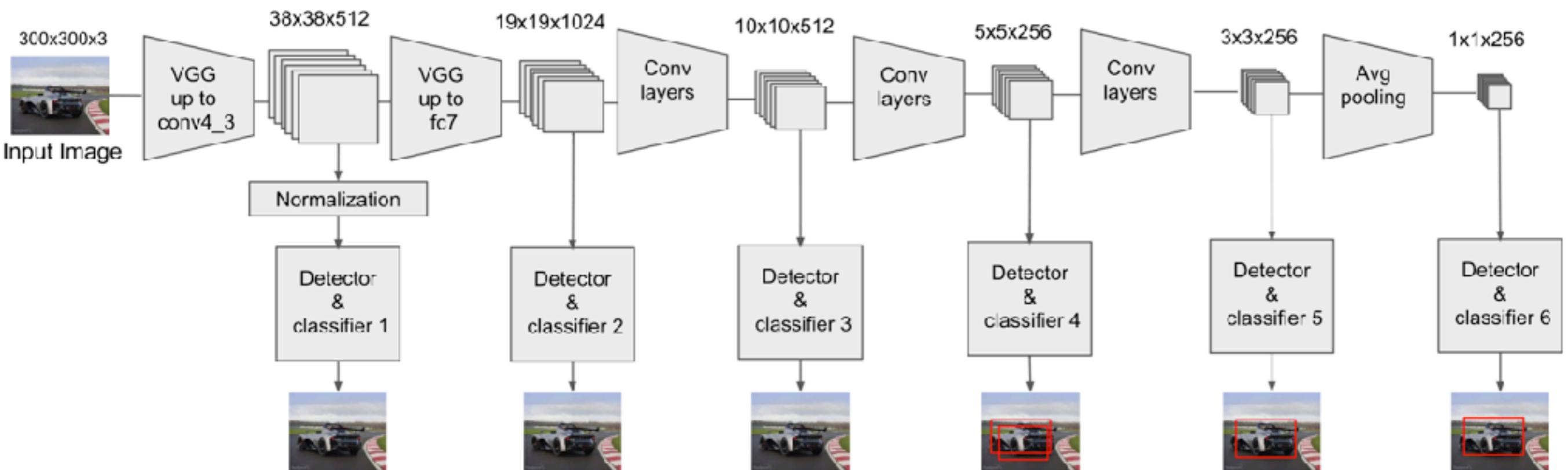
SSD架构：主干结构



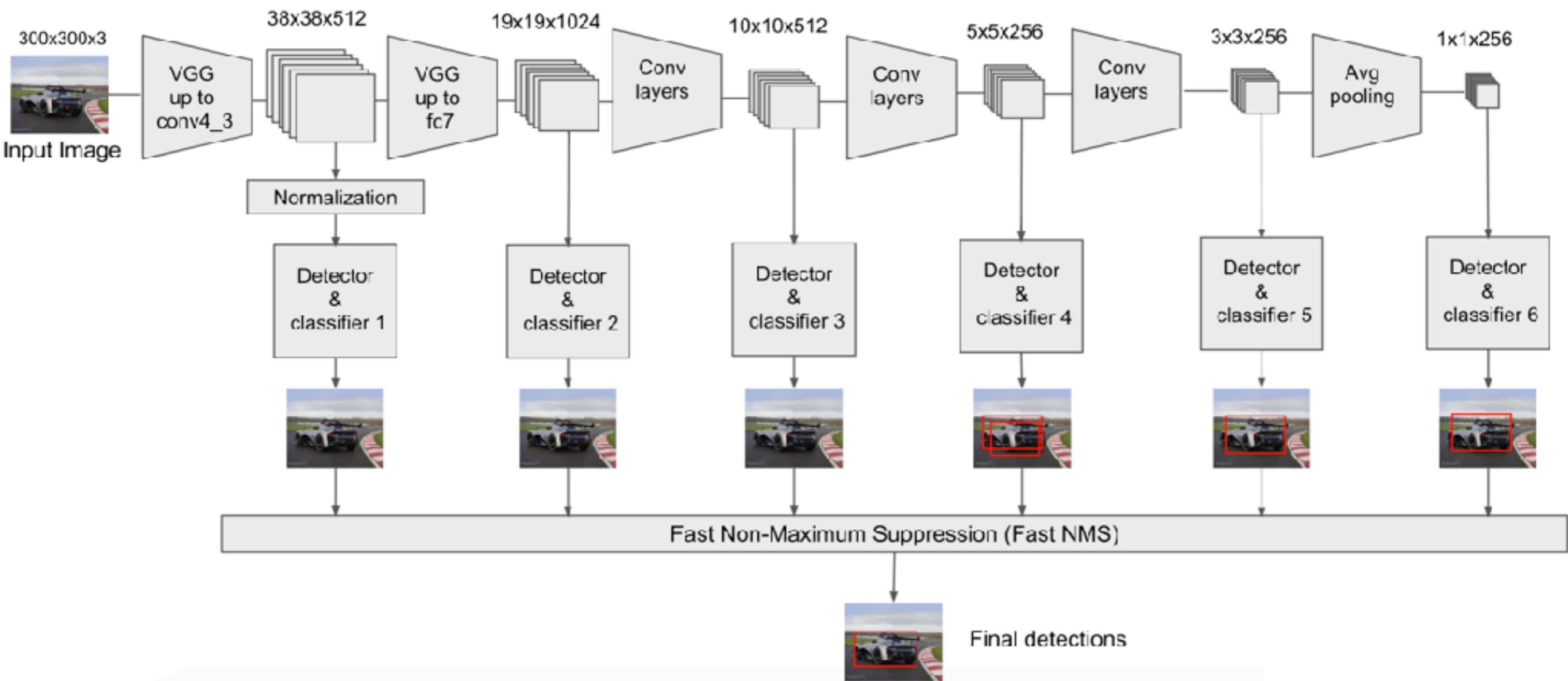
SSD架构：主干结构



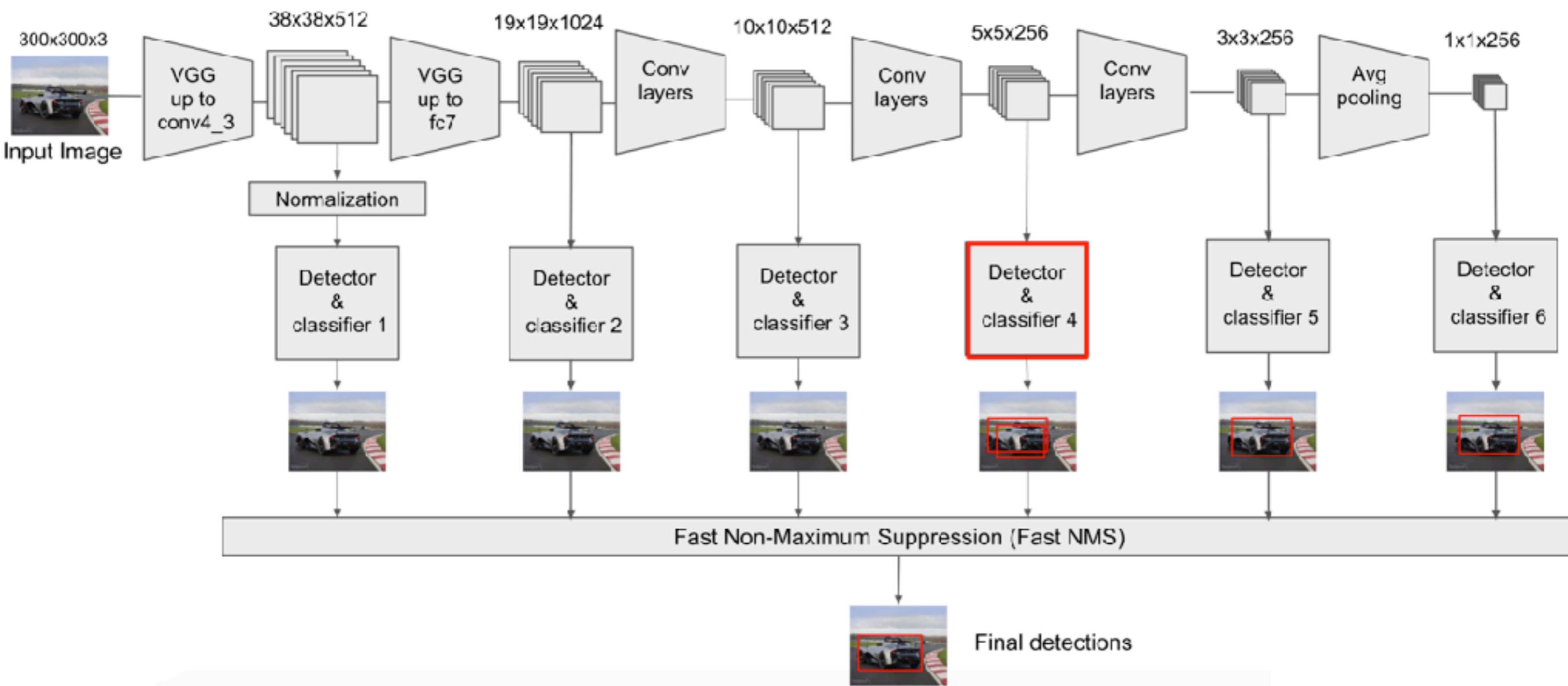
SSD架构：主干结构



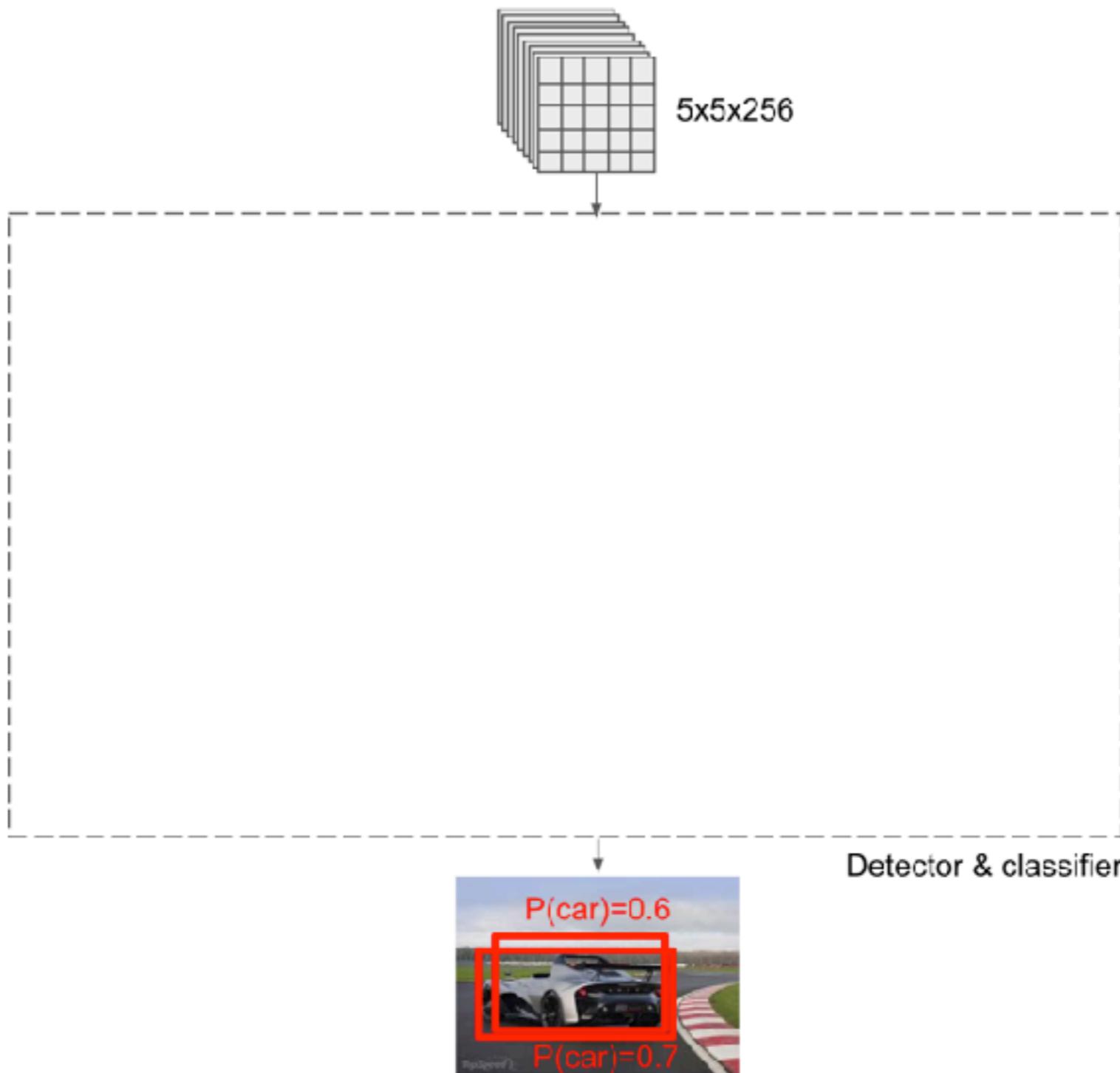
SSD架构：主干结构



SSD架构：主干结构



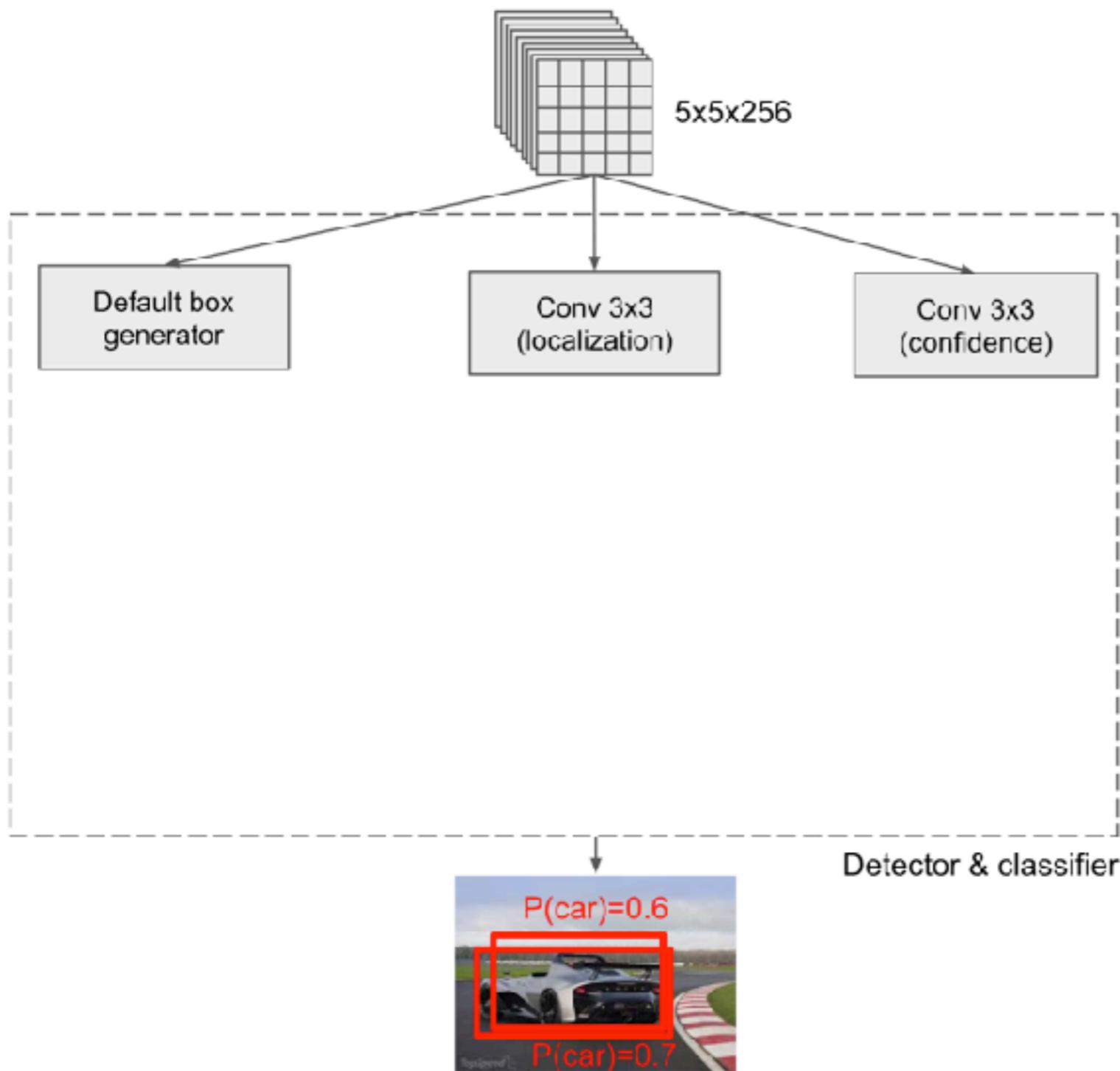
SSD架构：目标检测



参数：

- 输入图片大小 ($300 * 300$)
- 特征图大小 ($5 * 5 * 256$)
- default box数量为3 (每个特征对应三个default box)

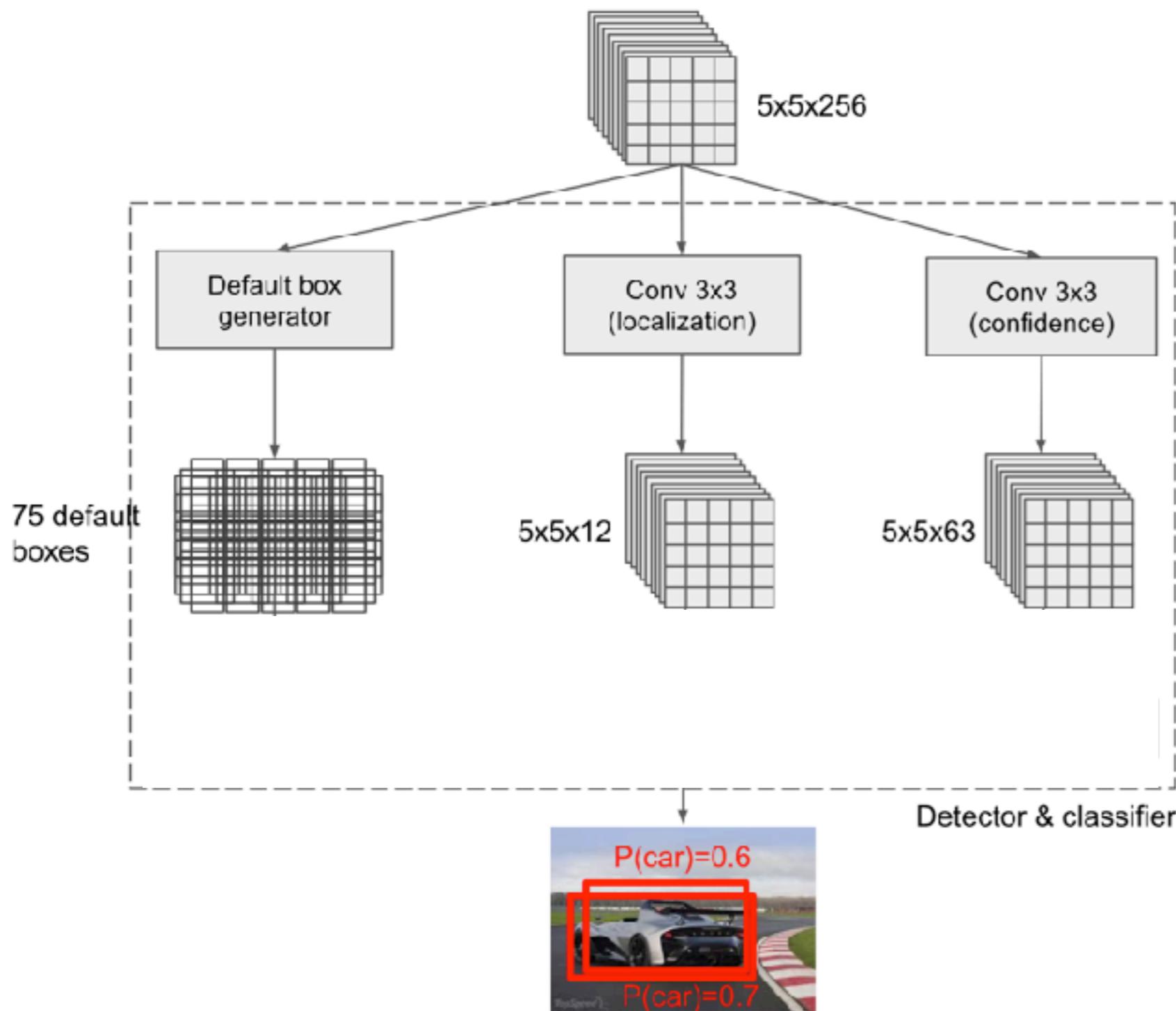
SSD架构：目标检测



参数：

- 输入图片大小 ($300 * 300$)
- 特征图大小 ($5 * 5 * 256$)
- default box数量为3 (每个特征对应三个default box)

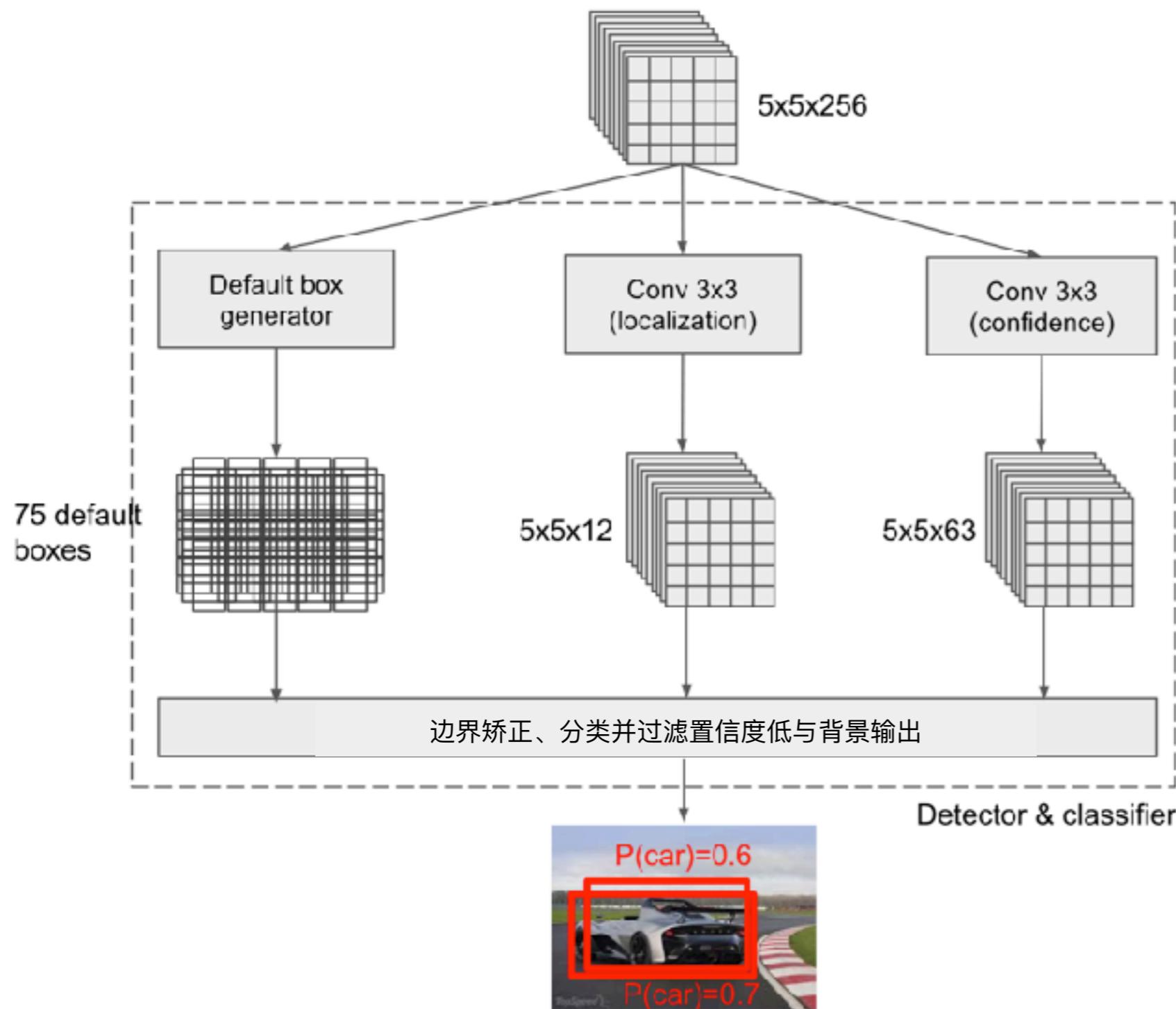
SSD架构：目标检测



参数：

- 输入图片大小 ($300 * 300$)
- 特征图大小 ($5 * 5 * 256$)
- default box数量为3 (每个特征对应三个default box)

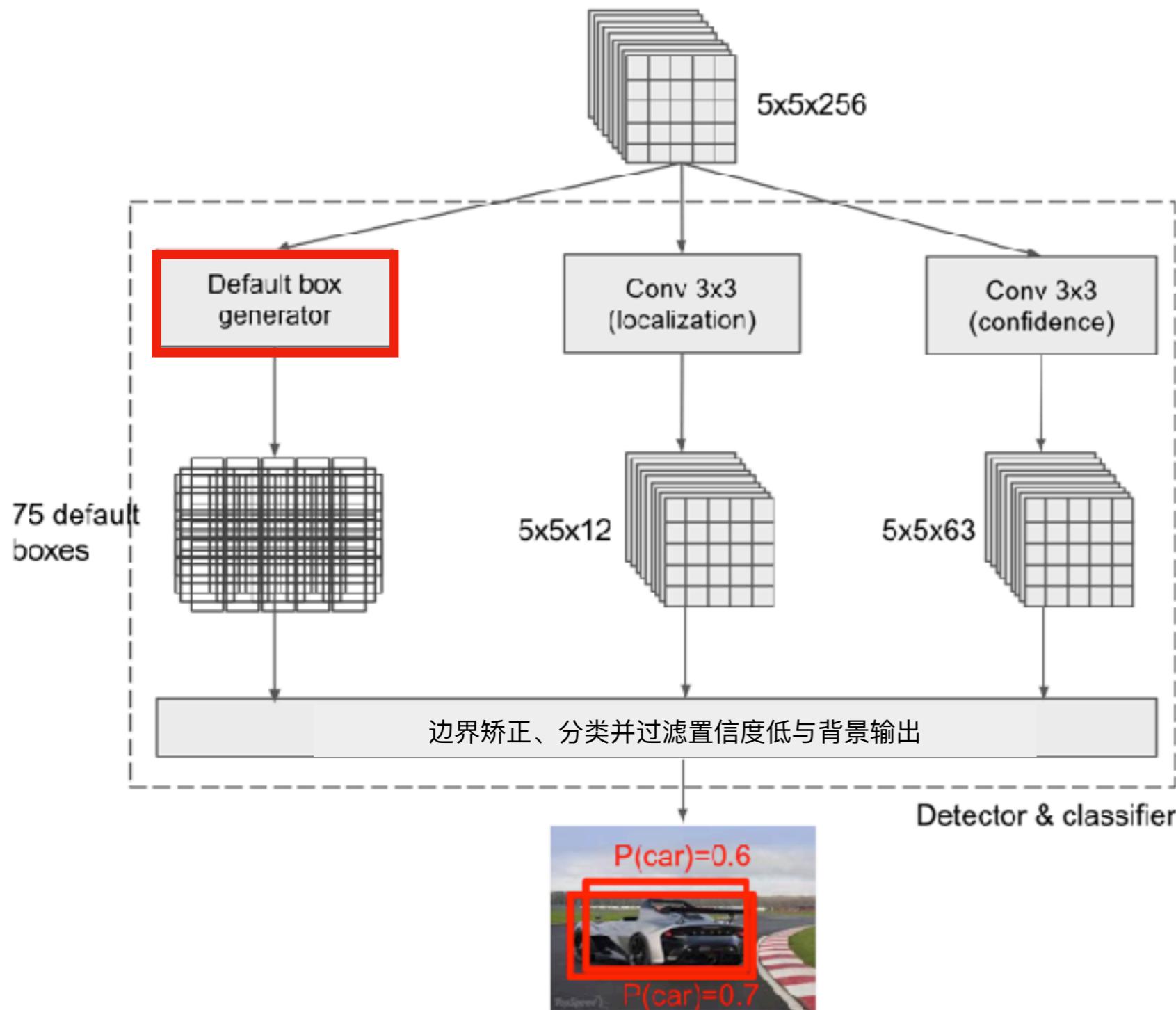
SSD架构：目标检测



参数：

- 输入图片大小 ($300 * 300$)
- 特征图大小 ($5 * 5 * 256$)
- default box数量为3 (每个特征对应三个default box)

SSD架构：目标检测



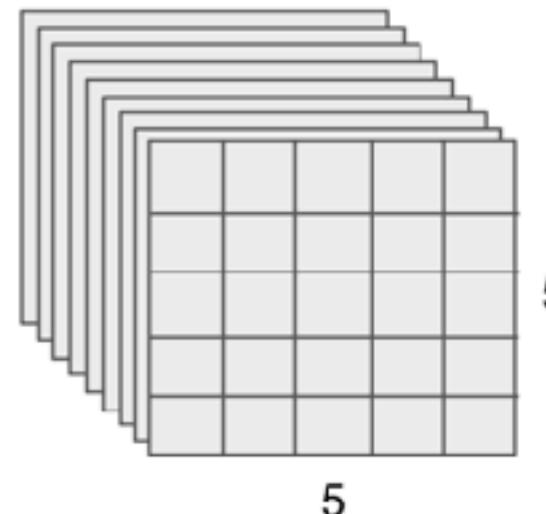
参数：

- 输入图片大小 ($300 * 300$)
- 特征图大小 ($5 * 5 * 256$)
- default box数量为3 (每个特征对应三个default box)

生成default box



Input Image

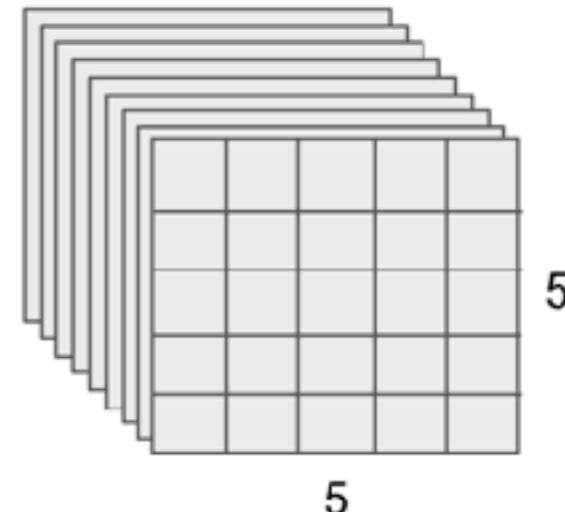
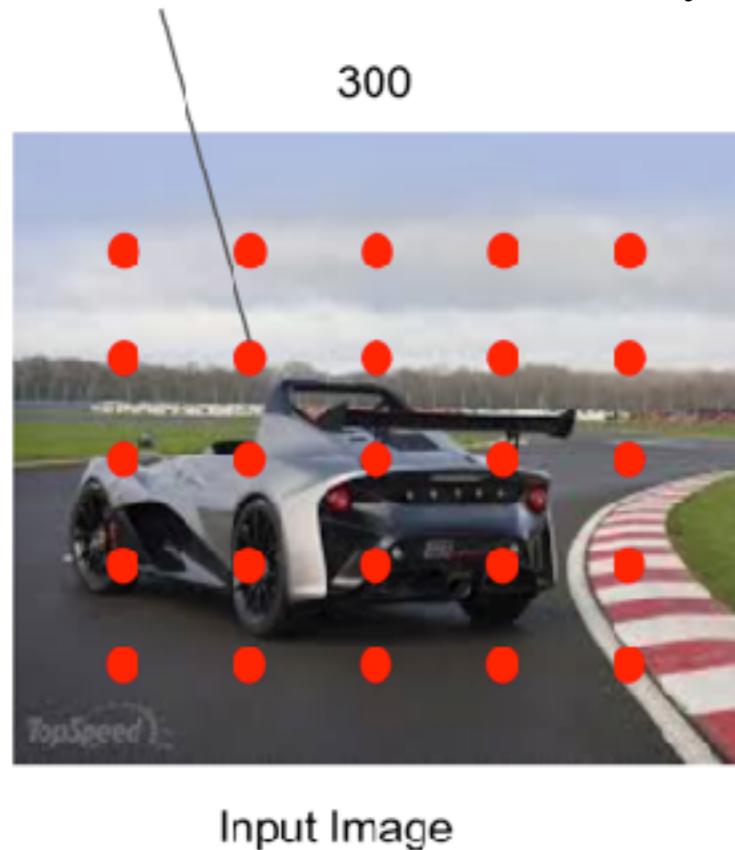


参数：

- 输入图片大小 ($300 * 300$)
- 特征图大小 ($5 * 5$)
- default box数量为3 (每个特征对应三个default box)
- $\text{min_size}=168$
- $\text{aspect_ratio}=2$

生成default box

生成default box的中心位置 (xc, yc)



参数：

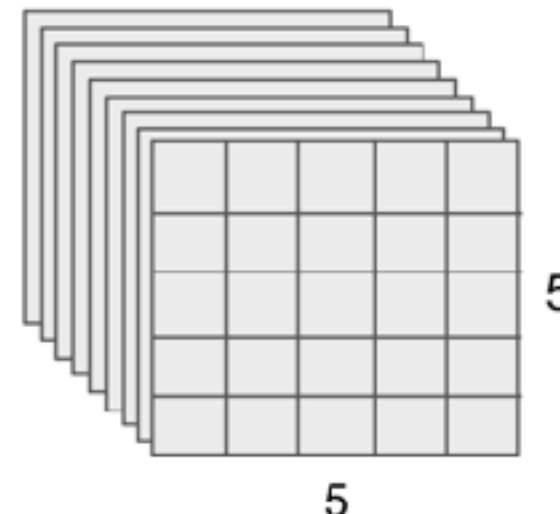
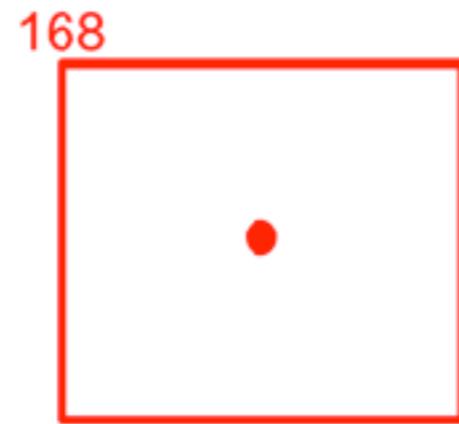
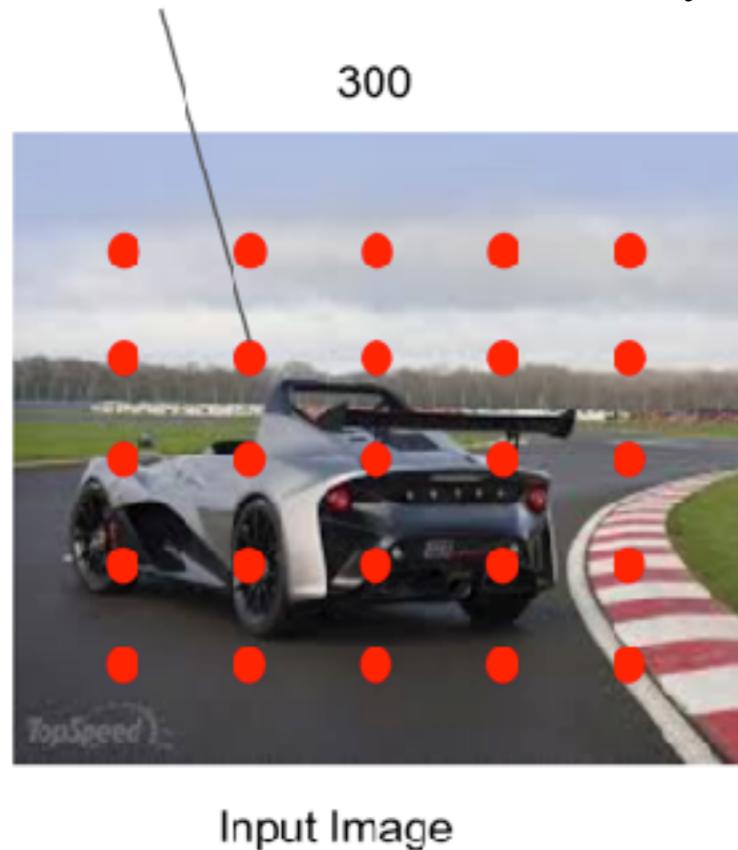
- 输入图片大小 ($300 * 300$)
- 特征图大小 ($5 * 5$)
- default box数量为3 (每个特征对应三个default box)
- $\text{min_size}=168$
- $\text{aspect_ratio}=2$

default box构成：

- xc , default box中心位置相对于所在cell的x
- yc , default box中心位置相对于所在cell的y
- w , default box宽
- h , default box高

生成default box

生成default box的中心位置 (xc, yc)



参数：

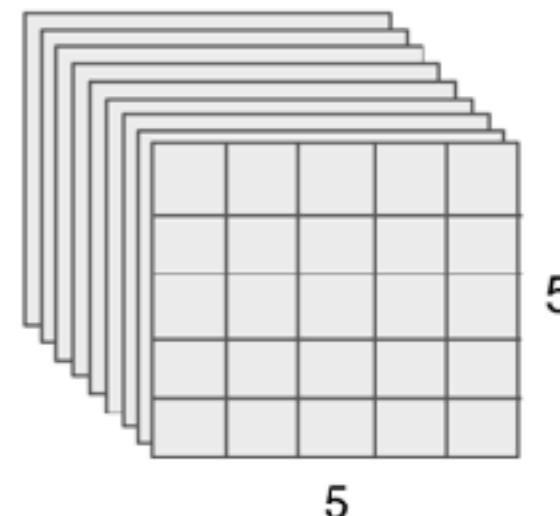
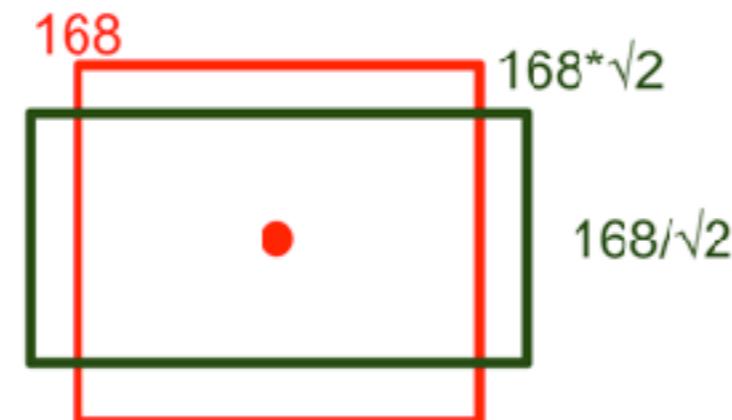
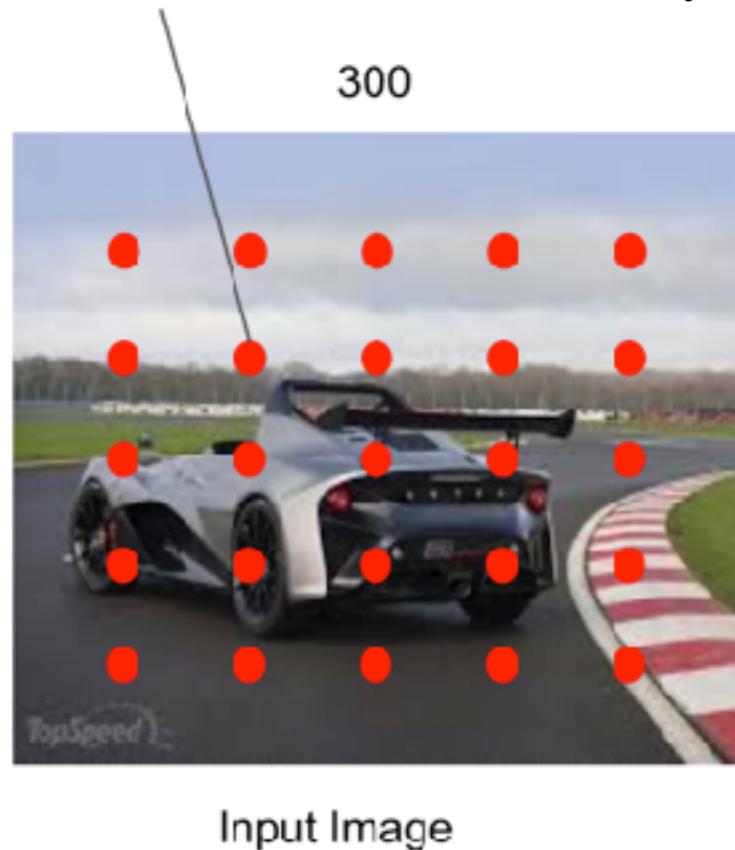
- 输入图片大小 ($300 * 300$)
- 特征图大小 ($5 * 5$)
- default box数量为3 (每个特征对应三个default box)
- $\text{min_size}=168$
- $\text{aspect_ratio}=2$

default box构成：

- xc , default box中心位置相对于所在cell的x
- yc , default box中心位置相对于所在cell的y
- w , default box宽
- h , default box高

生成default box

生成default box的中心位置 (xc, yc)



参数：

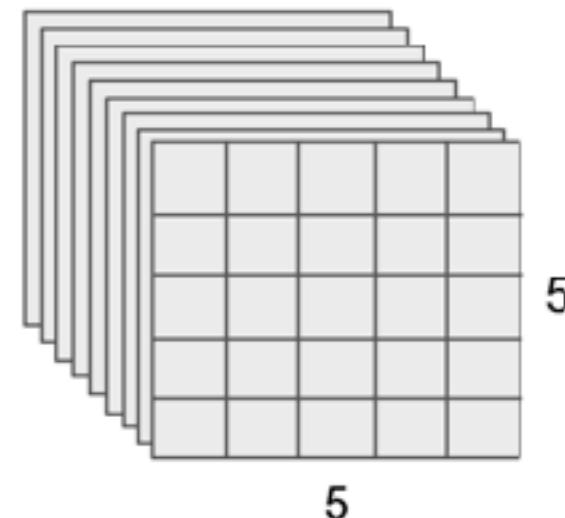
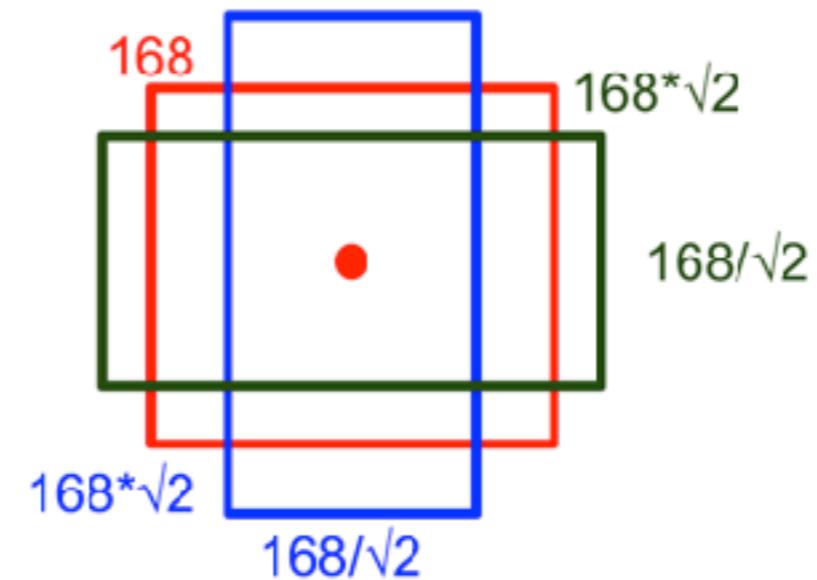
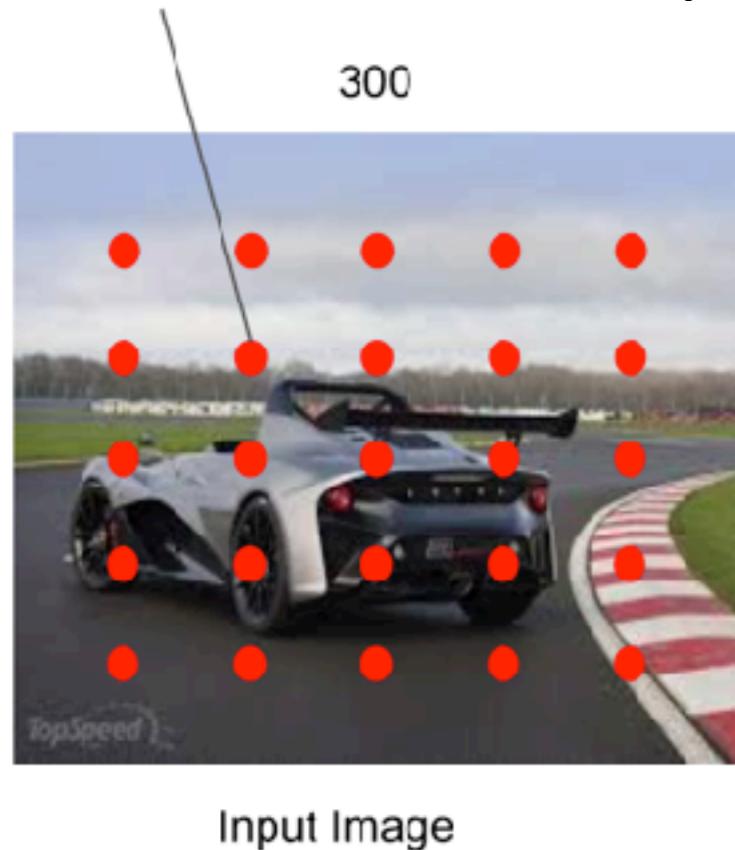
- 输入图片大小 $(300 * 300)$
- 特征图大小 $(5 * 5)$
- default box数量为3 (每个特征对应三个default box)
- $\text{min_size}=168$
- $\text{aspect_ratio}=2$

default box构成：

- xc , default box中心位置相对于所在cell的x
- yc , default box中心位置相对于所在cell的y
- w , default box宽
- h , default box高

生成default box

生成default box的中心位置 (x_c, y_c)



参数：

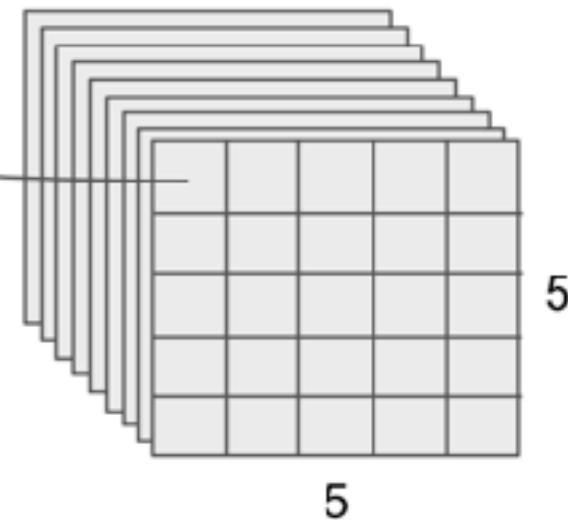
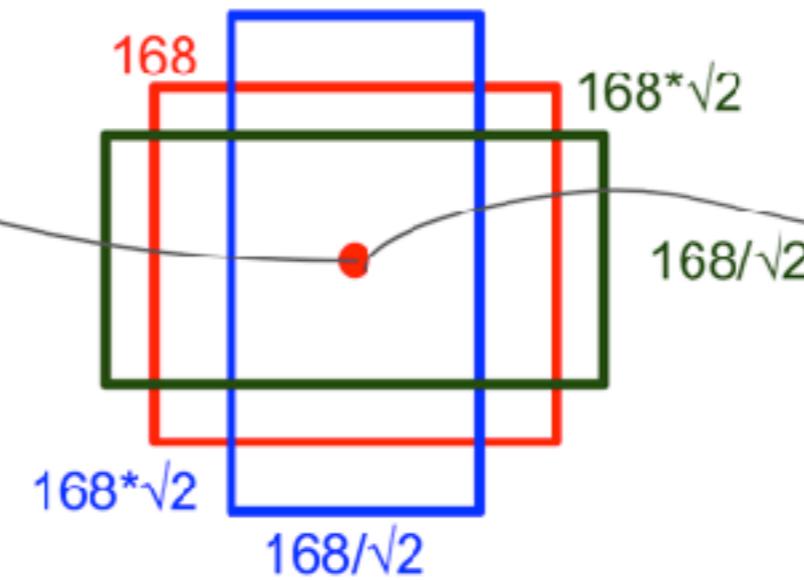
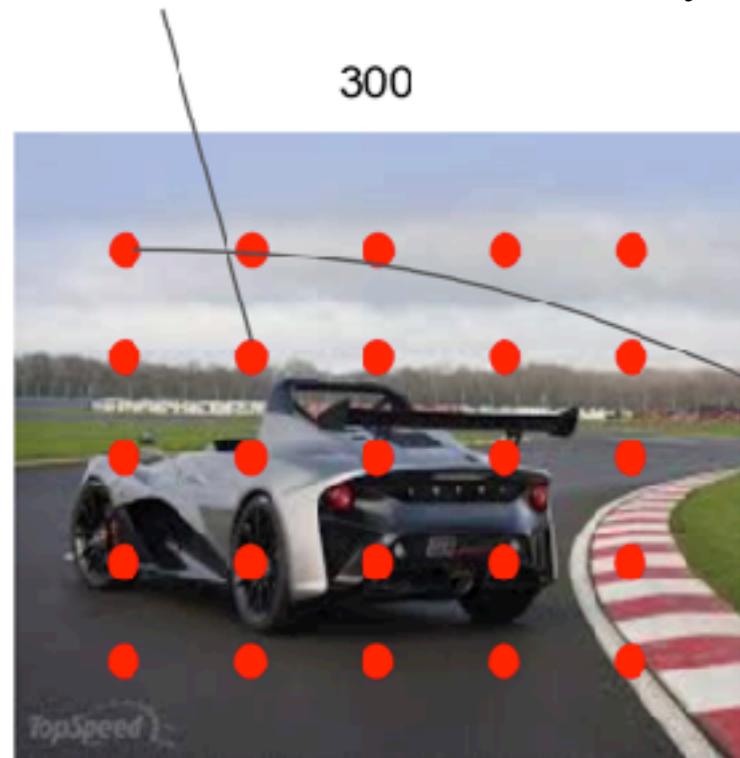
- 输入图片大小 ($300 * 300$)
- 特征图大小 ($5 * 5$)
- default box数量为3 (每个特征对应三个default box)
- $\text{min_size}=168$
- $\text{aspect_ratio}=2$

default box构成：

- x_c , default box中心位置相对于所在cell的x
- y_c , default box中心位置相对于所在cell的y
- w , default box宽
- h , default box高

生成default box

生成default box的中心位置 (xc, yc)



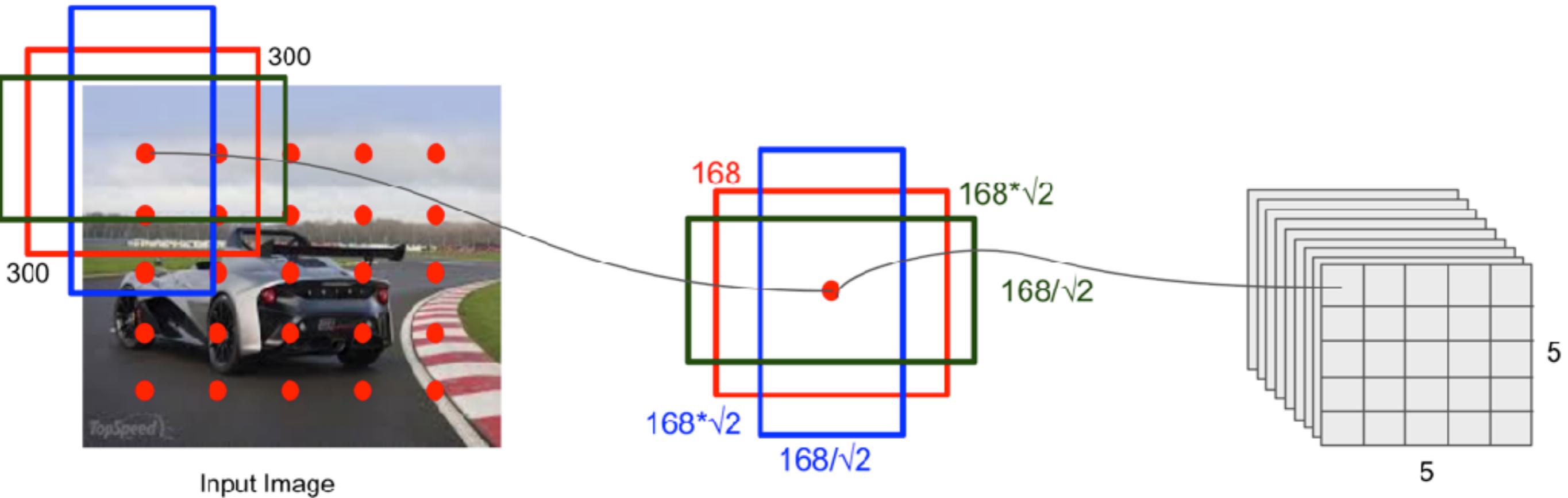
参数：

- 输入图片大小 ($300 * 300$)
- 特征图大小 ($5 * 5$)
- default box数量为3 (每个特征对应三个default box)
- $\text{min_size}=168$
- $\text{aspect_ratio}=2$

default box构成：

- xc , default box中心位置相对于所在cell的x
- yc , default box中心位置相对于所在cell的y
- w , default box宽
- h , default box高

生成default box



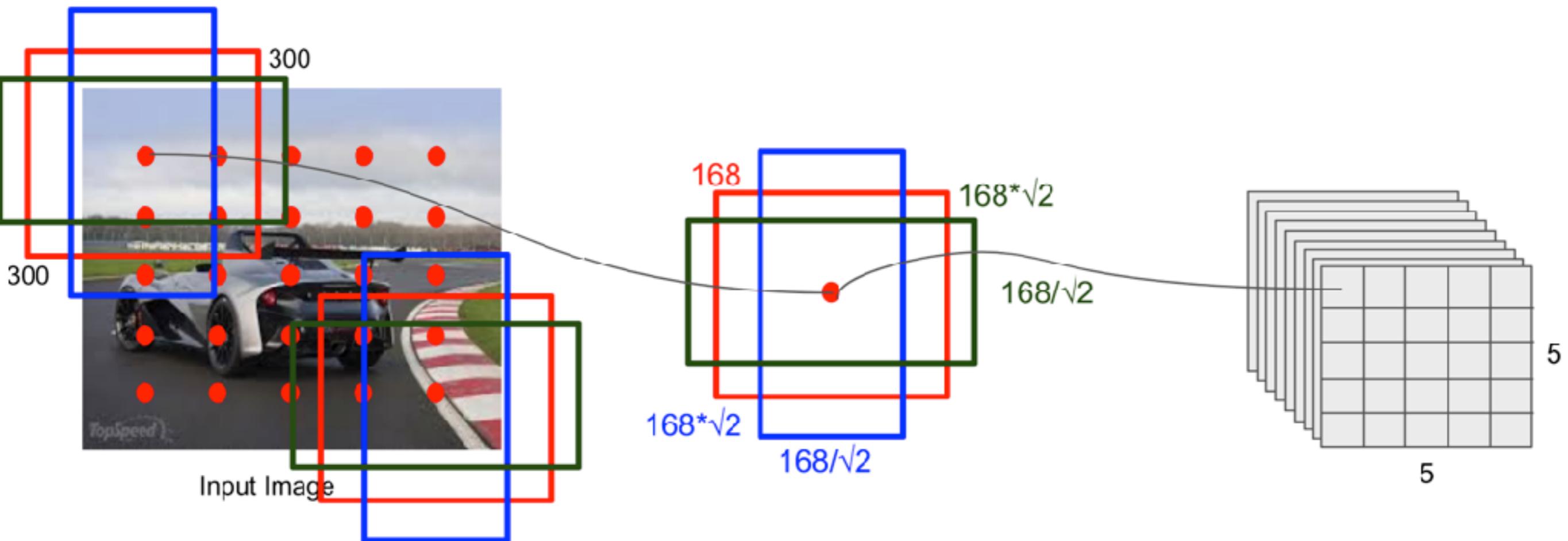
参数：

- 输入图片大小 ($300 * 300$)
- 特征图大小 ($5 * 5$)
- default box数量为3 (每个特征对应三个default box)
- $\text{min_size}=168$
- $\text{aspect_ratio}=2$

default box构成：

- xc , default box中心位置相对于所在cell的x
- yc , default box中心位置相对于所在cell的y
- w , default box宽
- h , default box高

生成default box



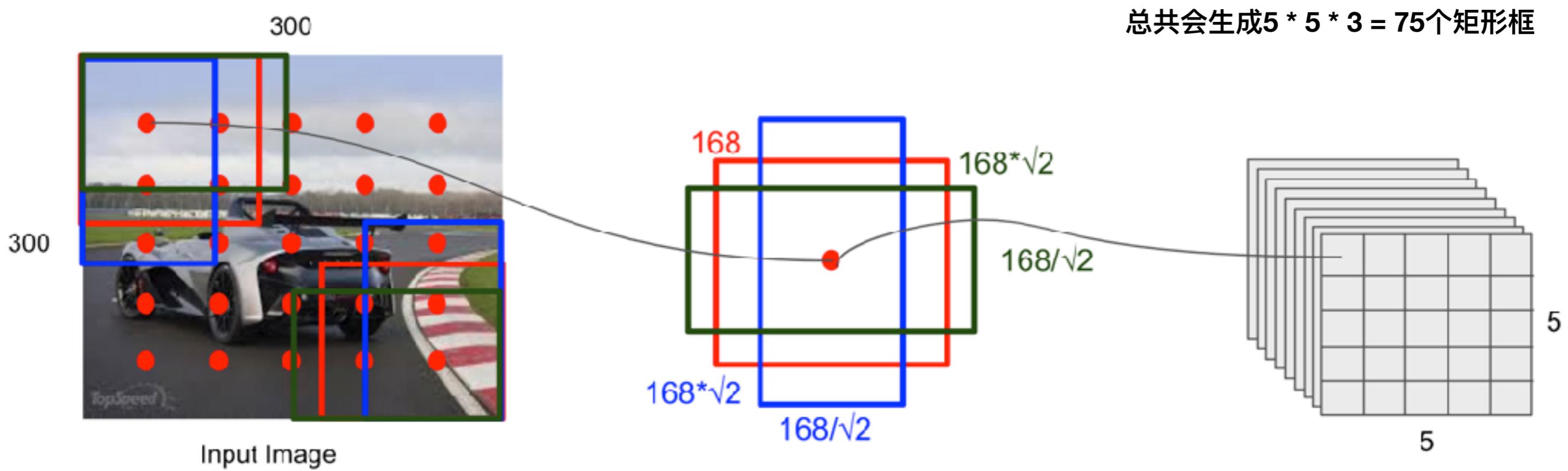
参数：

- 输入图片大小 ($300 * 300$)
- 特征图大小 ($5 * 5$)
- default box数量为3 (每个特征对应三个default box)
- $\text{min_size}=168$
- $\text{aspect_ratio}=2$

default box构成：

- xc , default box中心位置相对于所在cell的x
- yc , default box中心位置相对于所在cell的y
- w , default box宽
- h , default box高

生成default box



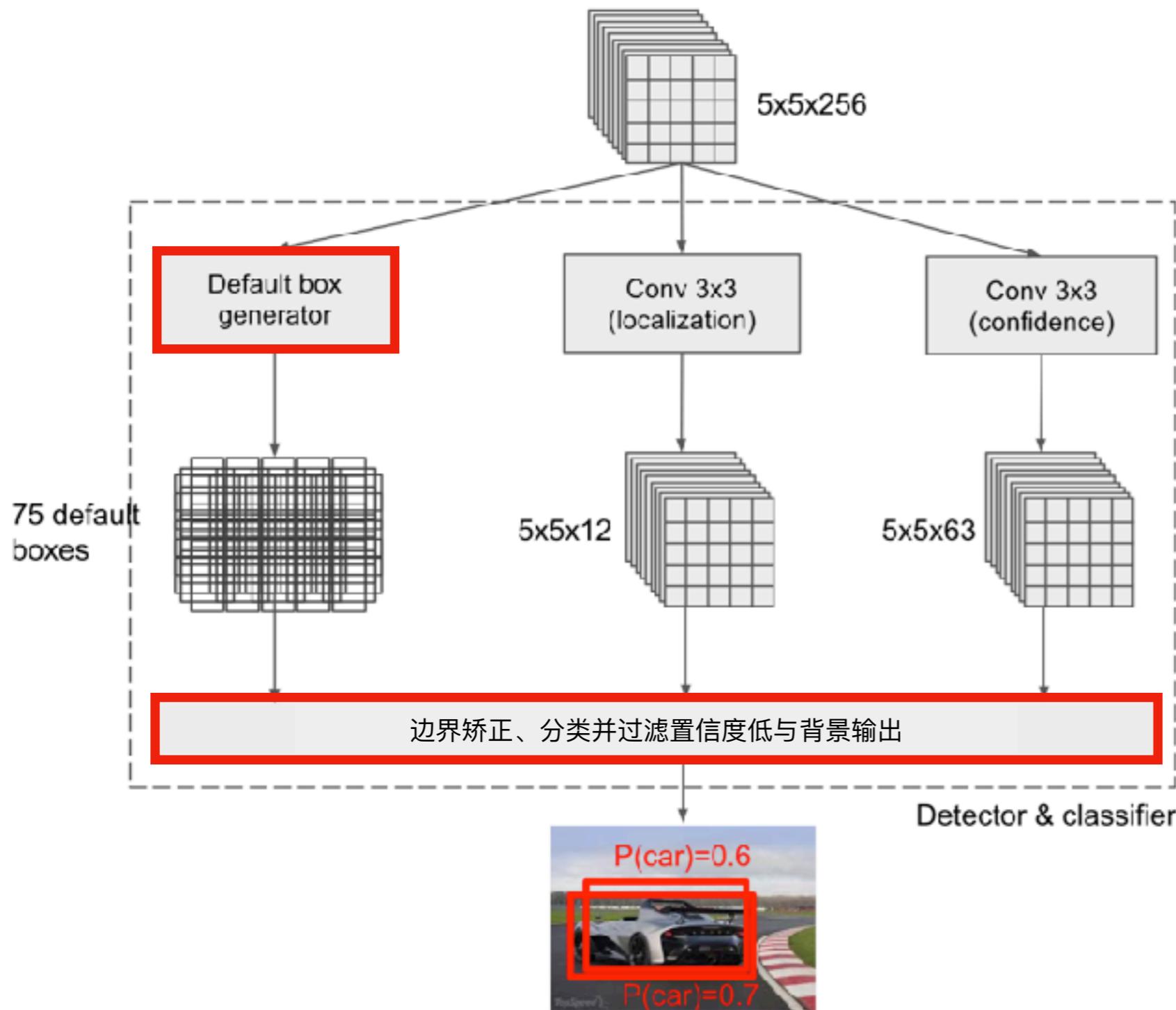
参数：

- 输入图片大小 ($300 * 300$)
- 特征图大小 ($5 * 5$)
- default box数量为3 (每个特征对应三个default box)
- $\text{min_size}=168$
- $\text{aspect_ratio}=2$

default box构成：

- xc , default box中心位置相对于所在cell的x
- yc , default box中心位置相对于所在cell的y
- w , default box宽
- h , default box高

SSD架构：目标检测

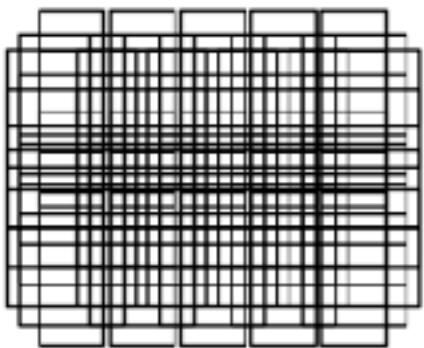


参数：

- 输入图片大小 ($300 * 300$)
- 特征图大小 ($5 * 5 * 256$)
- default box数量为3 (每个特征对应三个default box)

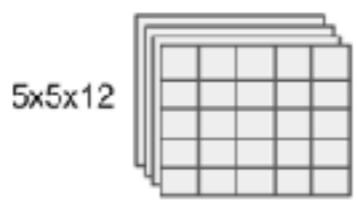
BBox边界矫正与分类

Default boxes

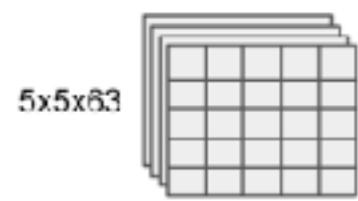


总共 $5 \times 5 \times 3 = 75$ 个default box

Localization

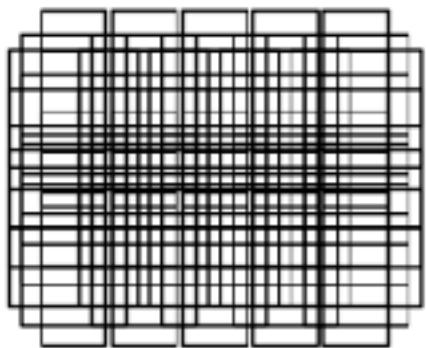


Confidence



BBox边界矫正与分类

Default boxes



总共 $5 \times 5 \times 3 = 75$ 个default box

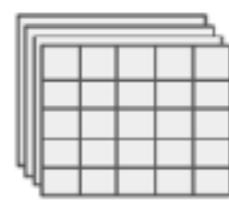
Localization

$5 \times 5 \times 12$



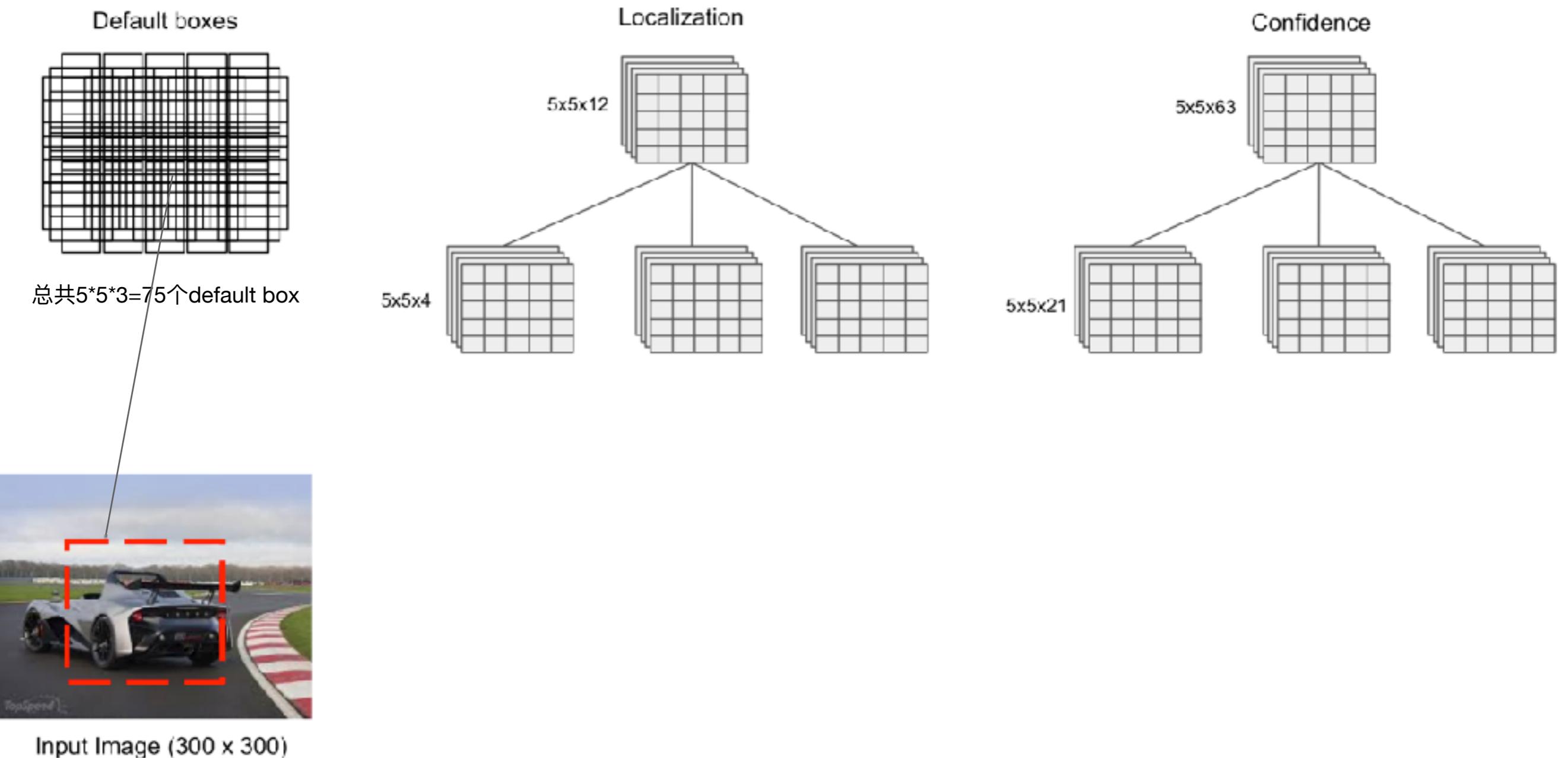
Confidence

$5 \times 5 \times 63$

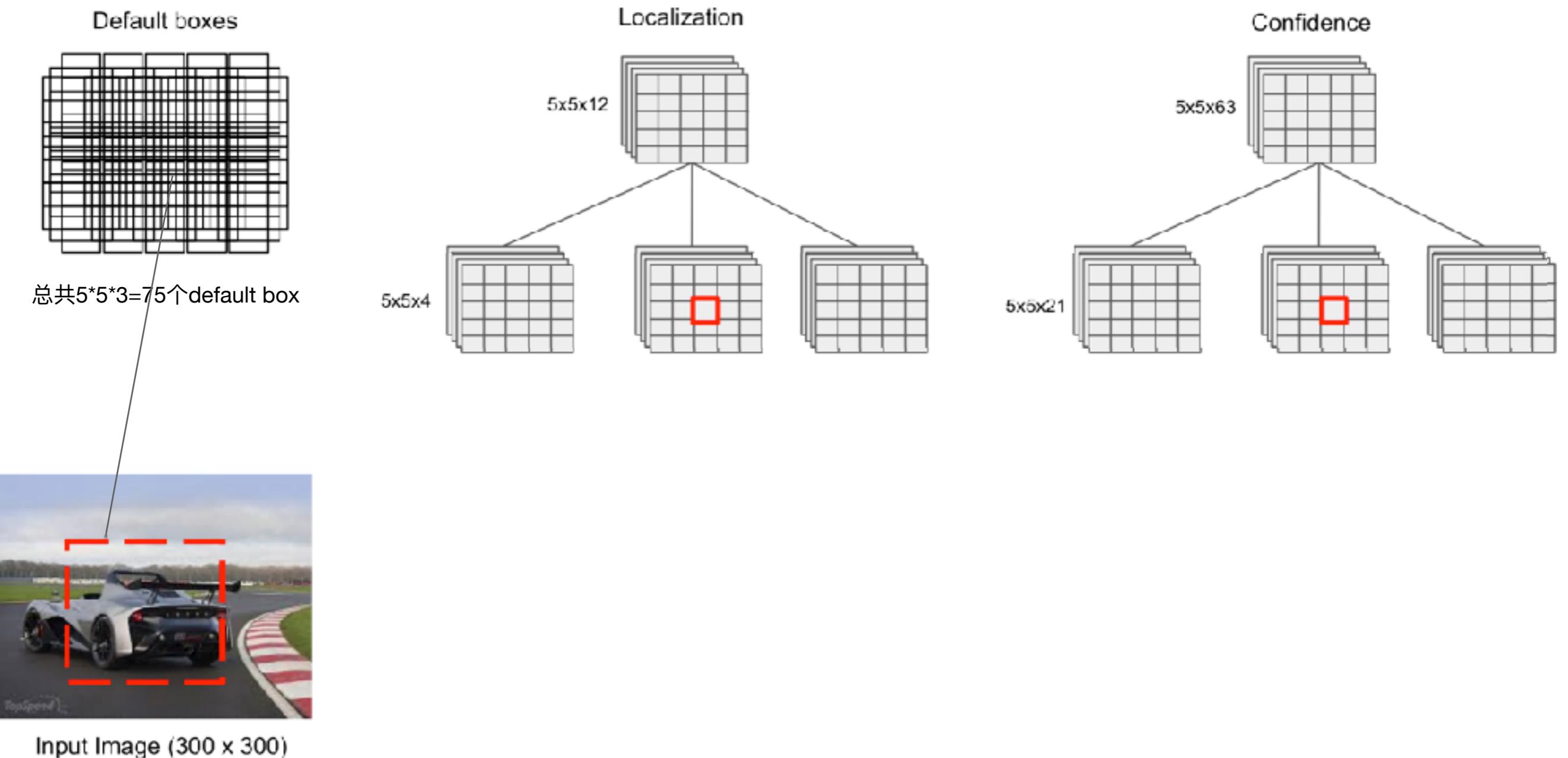


Input Image (300 x 300)

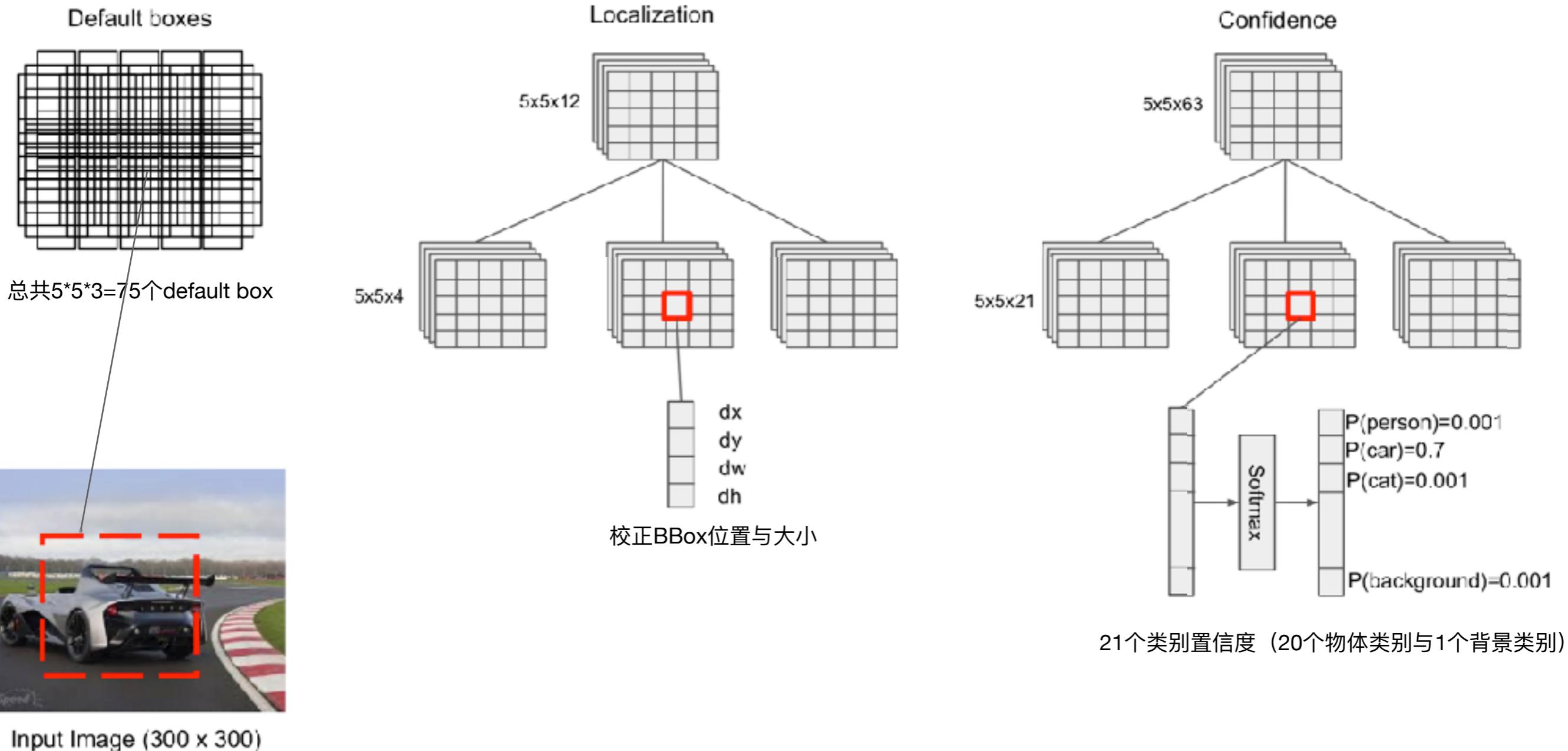
BBox边界矫正与分类



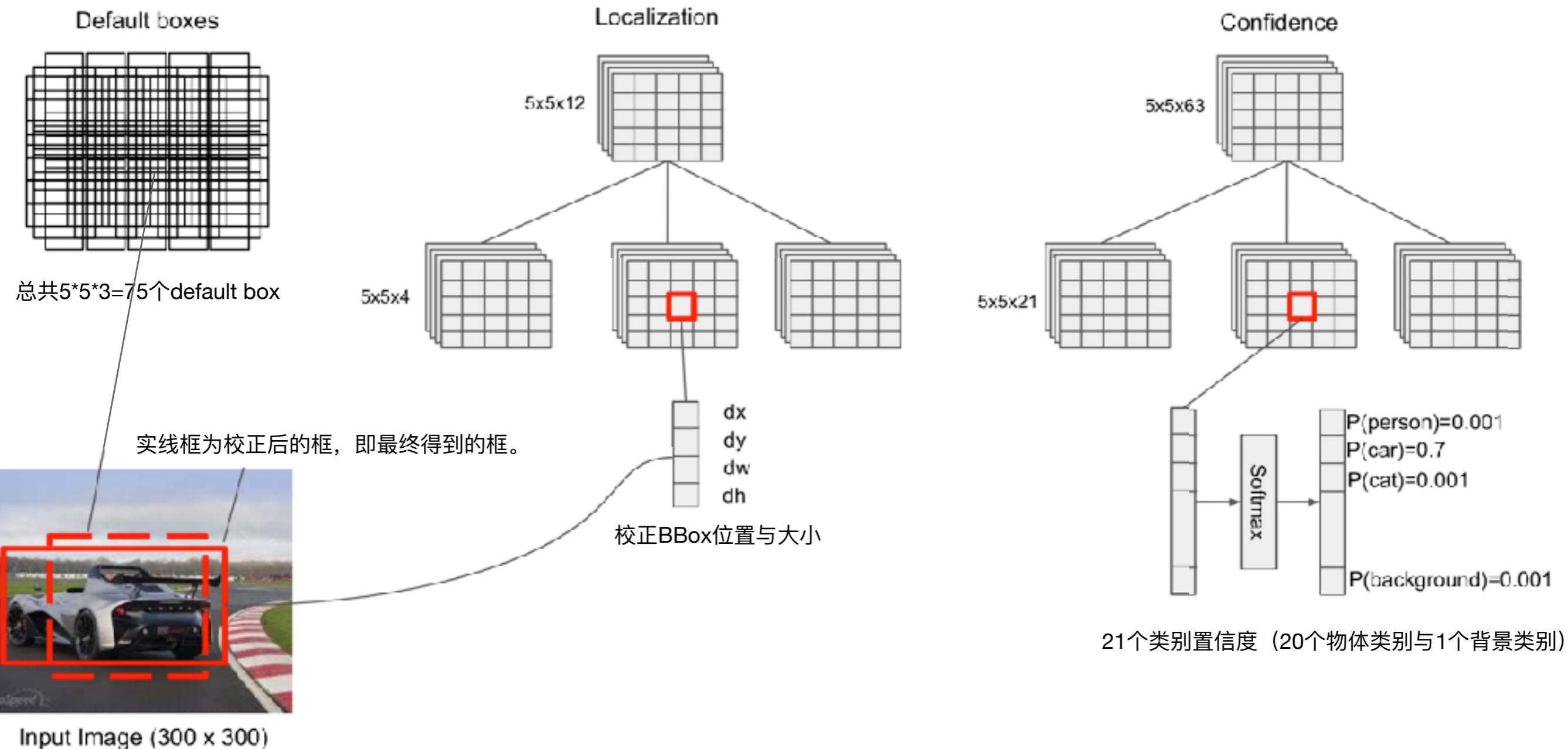
BBox边界矫正与分类



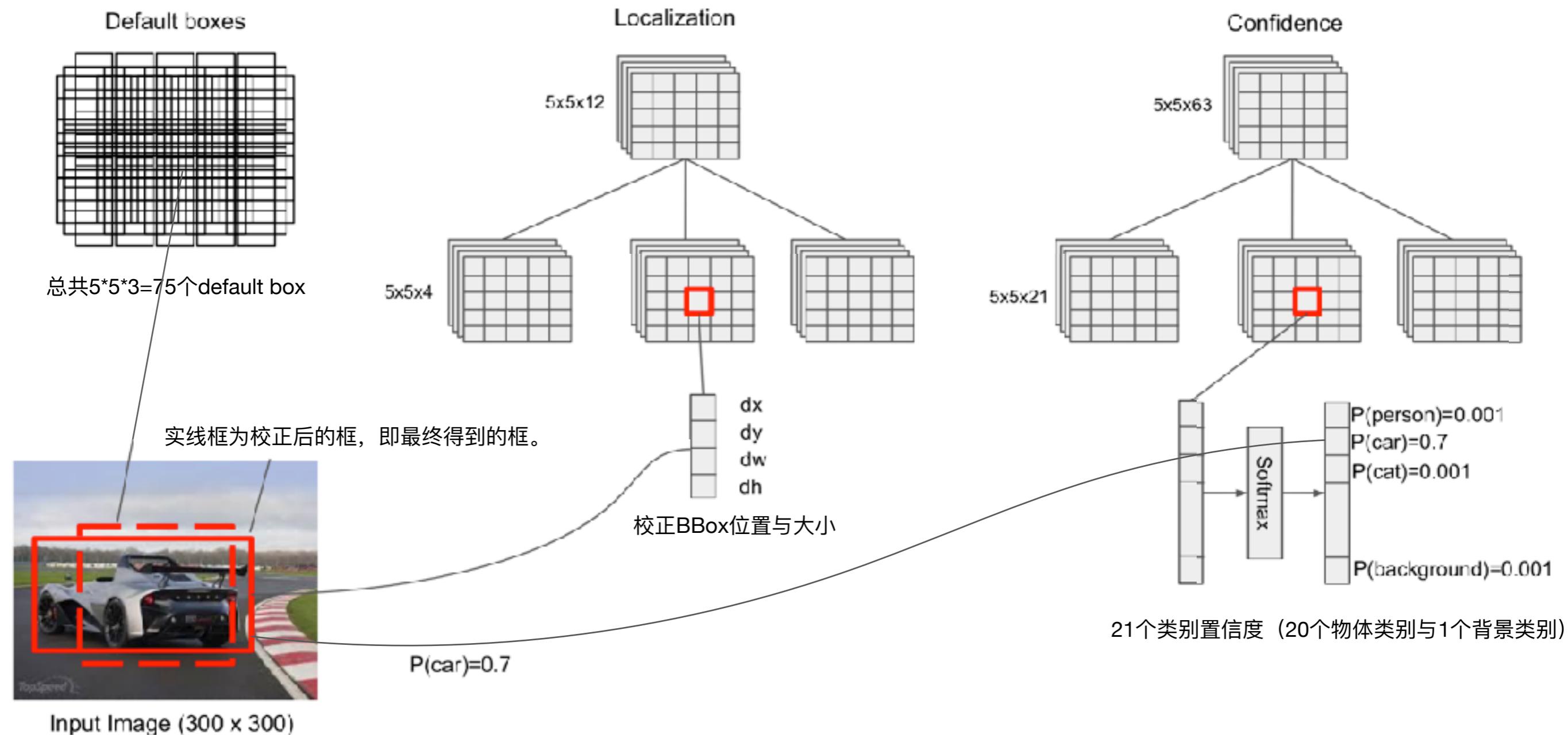
BBox边界矫正与分类



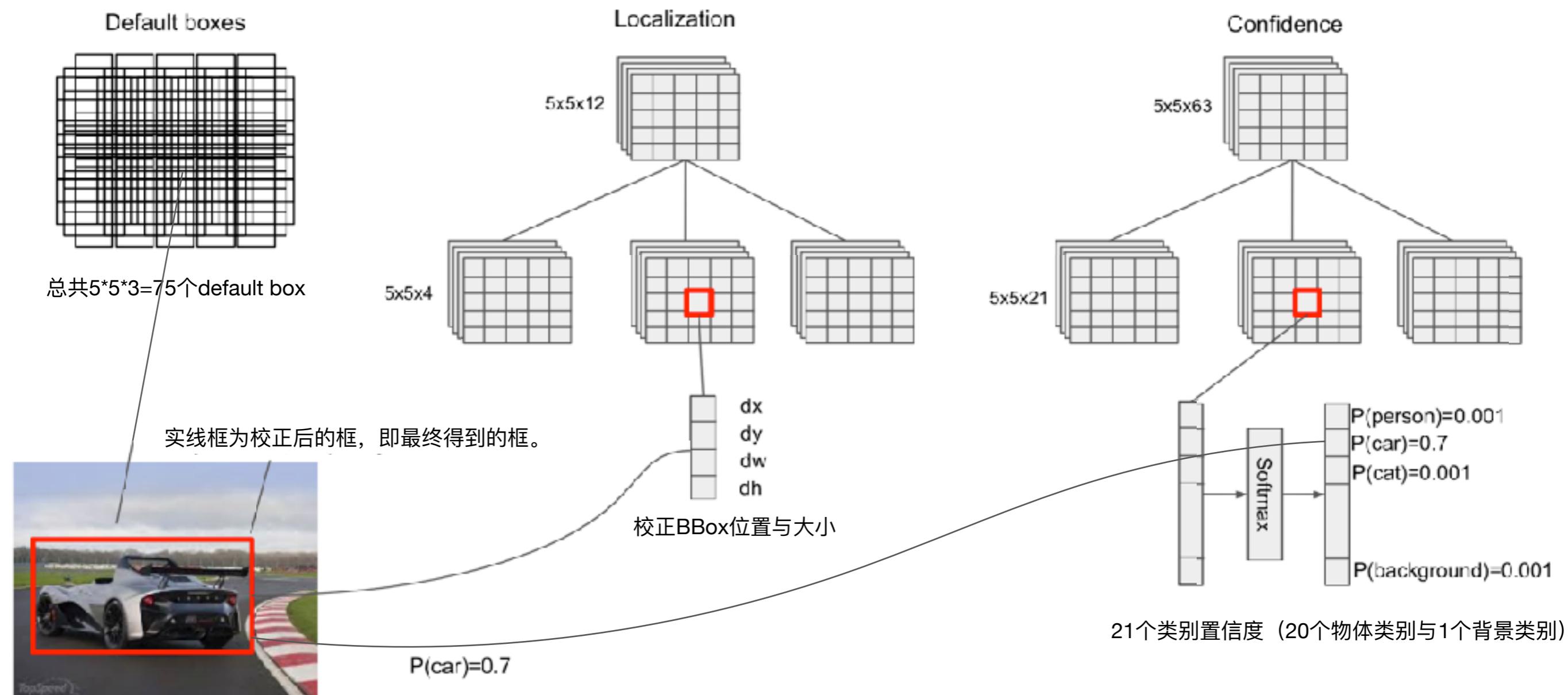
BBox边界矫正与分类



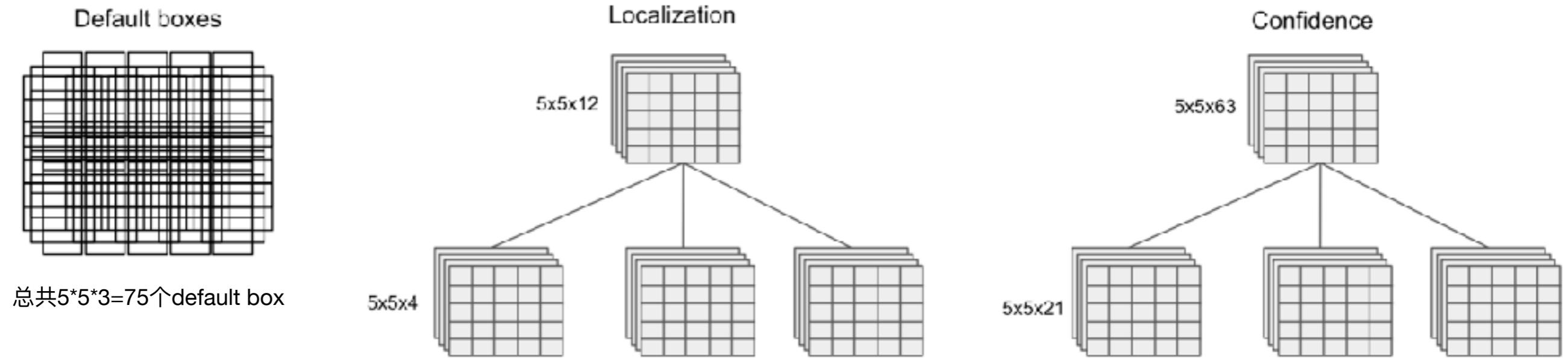
BBox边界矫正与分类



BBox边界矫正与分类

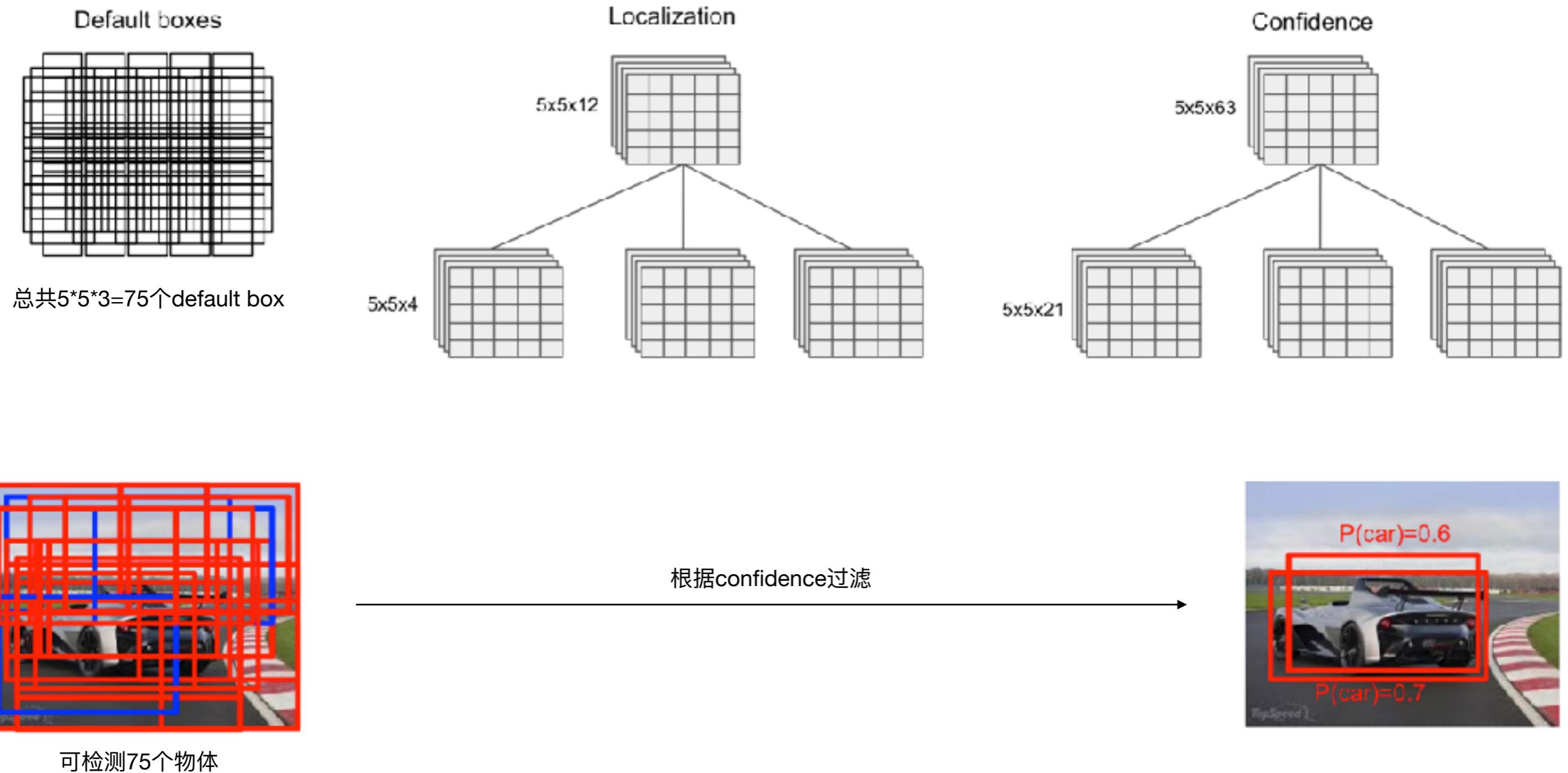


输出过滤

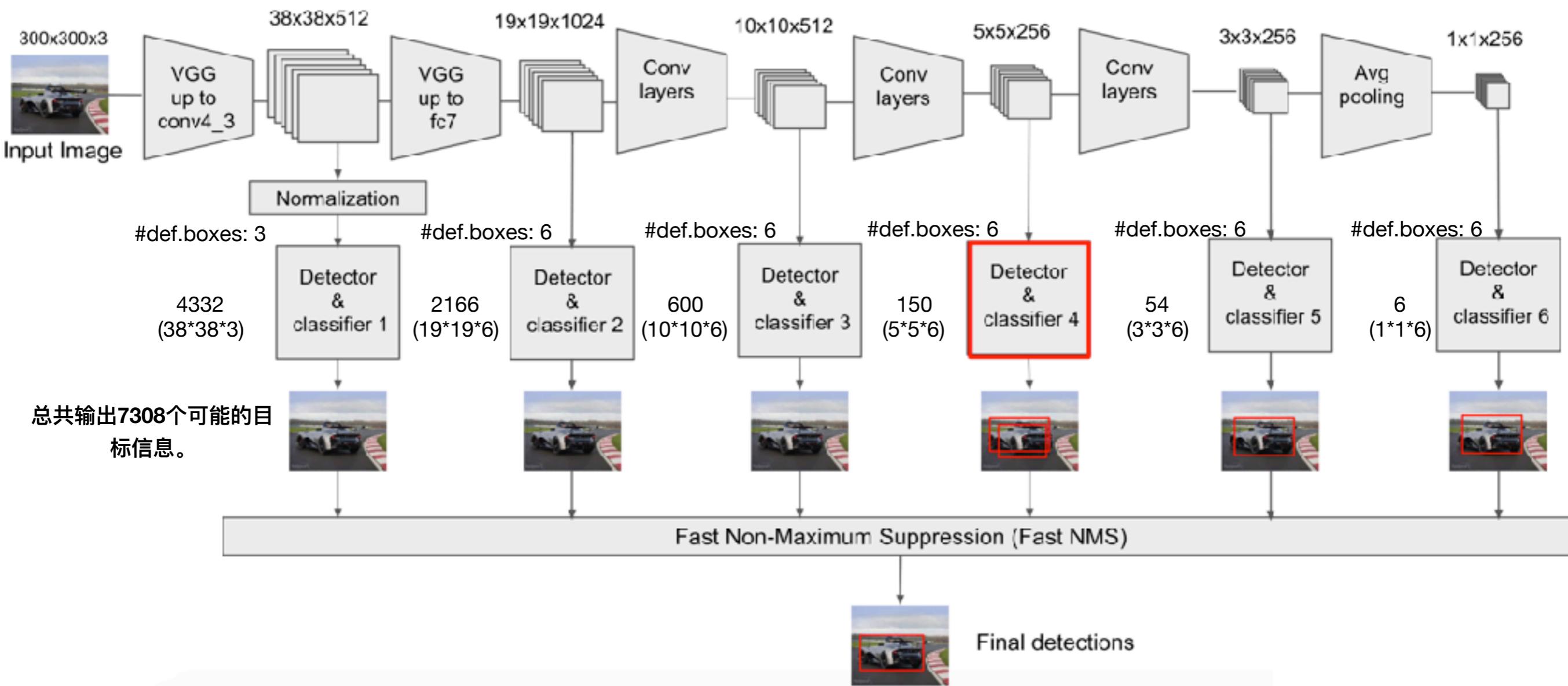


可检测75个物体

输出过滤

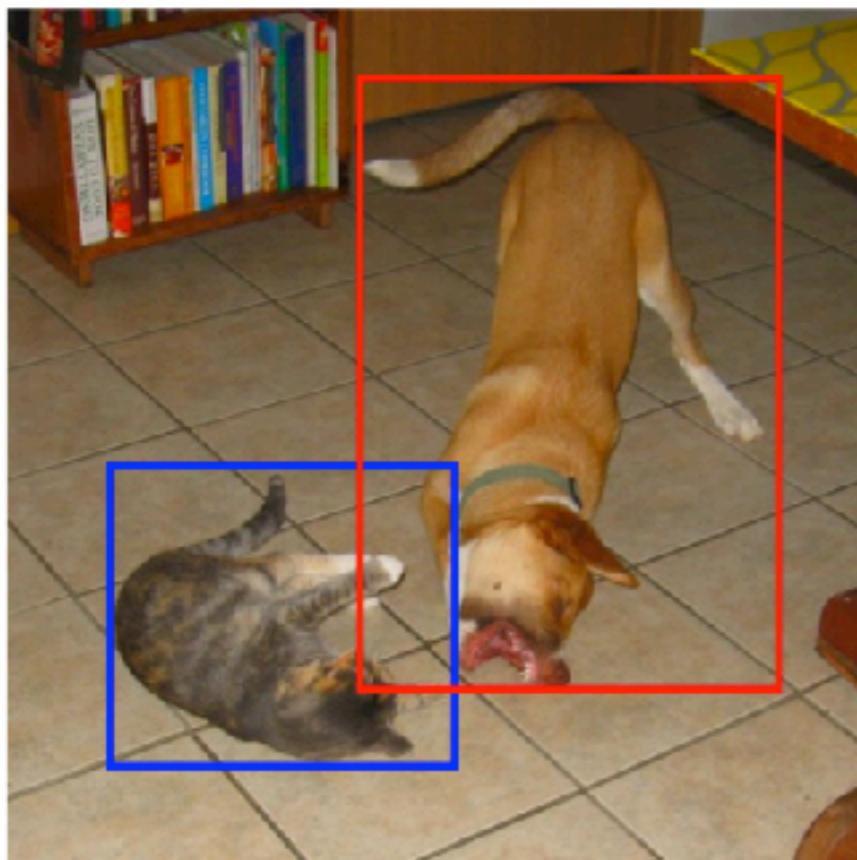


SSD架构

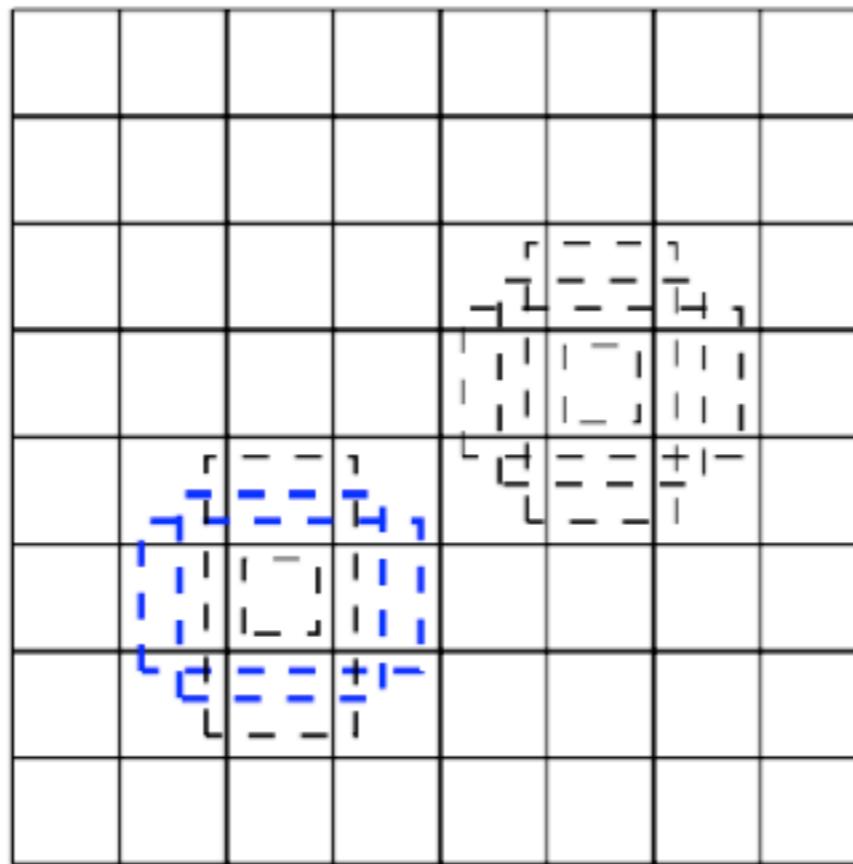


6.3 SSD训练

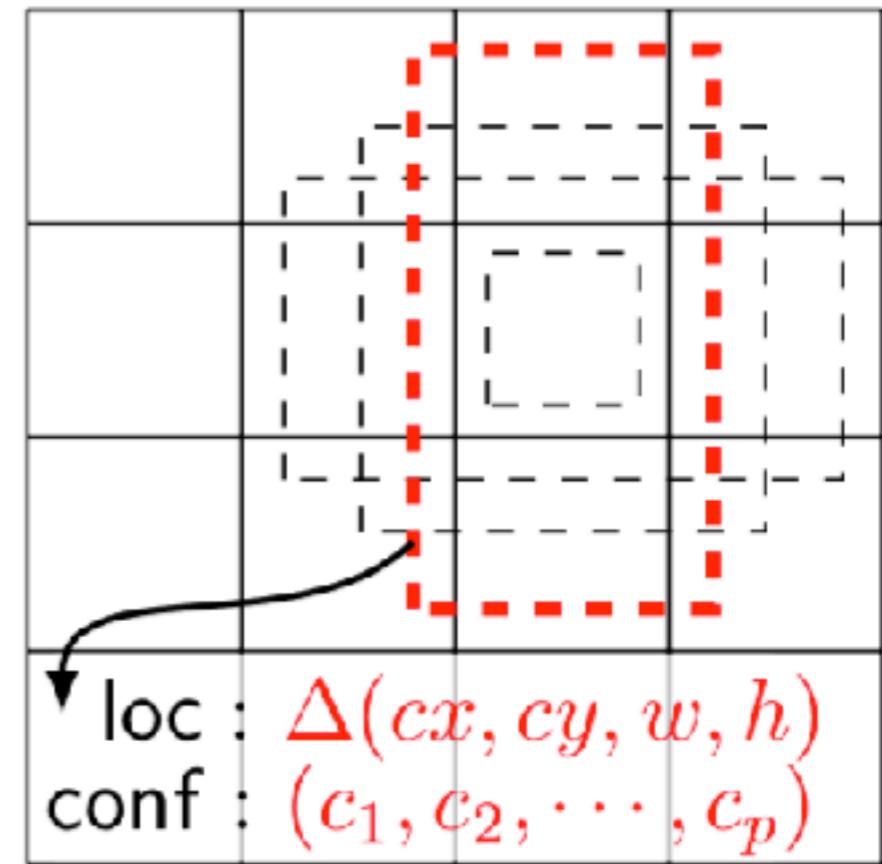
正负样本



(a) Image with GT boxes



(b) 8×8 feature map



loc : $\Delta(cx, cy, w, h)$
conf : (c_1, c_2, \dots, c_p)

(c) 4×4 feature map

图(a)表示输入图像与**ground truth boxes**, 图(b)表示 8×8 特征图, 图(c)表示 4×4 特征图。共有两个**default box**与猫匹配, 有一个**default box**与狗匹配。SSD与YOLO等模型不同的地方在于 SSD把所有**default boxes**与**GT boxes**的IOU值大于某个阈值的**default boxes**均作为正类样本, 其它的作为负类样本。

正负样本

1. 首先寻找与每一个ground truth box有最大IOU值的default box，保证每一个ground truth box都有对应的default box；
2. 其次在所有剩余的default box中，分别与每一个ground truth box计算IOU值，把大于阈值（论文中为0.5）的default box挑选出来；
3. 由1、2步骤挑选得到的default box都认为是正样本（包含目标的 default box），其它样本都是负样本（不包含目标的default box）；
4. 实际中由于负类样本数量往往远大于正样本数量，所以作者提出根据 confidence loss 对负样本由大到小进行排序，选择其中一部分较大 loss 的负样本（论文中最大数量为正类样本的3倍）。

代价函数

代价包括定位误差与分类误差（置信度误差）两部分函数：

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g))$$

其中 N 指正类样本数量， α 用来调节定位误差所占比例， α 默认为1。

代价函数

分类误差如下：

$$L_{conf}(x, c) = - \sum_{i \in Pos}^N x_{ij}^p \log(\hat{c}_i^p) - \sum_{i \in Neg} \log(\hat{c}_i^0)$$

其中 $x_{ij}^p = \{1, 0\}$ $\hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$

代价函数

定位误差如下：

$$L_{loc}(x, l, g) = \sum_{i \in Pos}^N \sum_{m \in \{cx, cy, w, h\}} x_{ij}^k \text{smooth}_{L1}(l_i^m - \hat{g}_j^m)$$

其中 $\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \quad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$

$$\hat{g}_j^w = \log\left(\frac{g_j^w}{d_i^w}\right) \quad \hat{g}_j^h = \log\left(\frac{g_j^h}{d_i^h}\right)$$

$$\text{smooth}_{L1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases}$$

6.4 SSD细节

Default Box生成

每个特征图预测的“基准”default box（相对于原图大小）的缩放比例：

$$s_k = s_{\min} + \frac{s_{\max} - s_{\min}}{m - 1}(k - 1), \quad k \in [1, m]$$

m表示进行预测的特征图数量（论文中为6），默认的S_max=0.9，
S_min=0.2。

可以看到：特征图越小，对应的default box越大。

Default Box生成

每个特征图预测的default box相对于“基准”default box的比例：

$$w_k^a = s_k \sqrt{a_r}$$

$$h_k^a = s_k / \sqrt{a_r}$$

其中： $a_r \in \{1, 2, 3, \frac{1}{2}, \frac{1}{3}\}$

当 $a_r = 1$ 时，我们增加一种情况： $s'_k = \sqrt{s_k s_{k+1}}$

最终，每个特征对应6个default box（作者并没有按照这个规则生成default box）。

Default Box中心位置

第k个特征图第i行第j列中心位置: $(\frac{i+0.5}{|f_k|}, \frac{j+0.5}{|f_k|})$

其中: $i, j \in [0, |f_k|)$

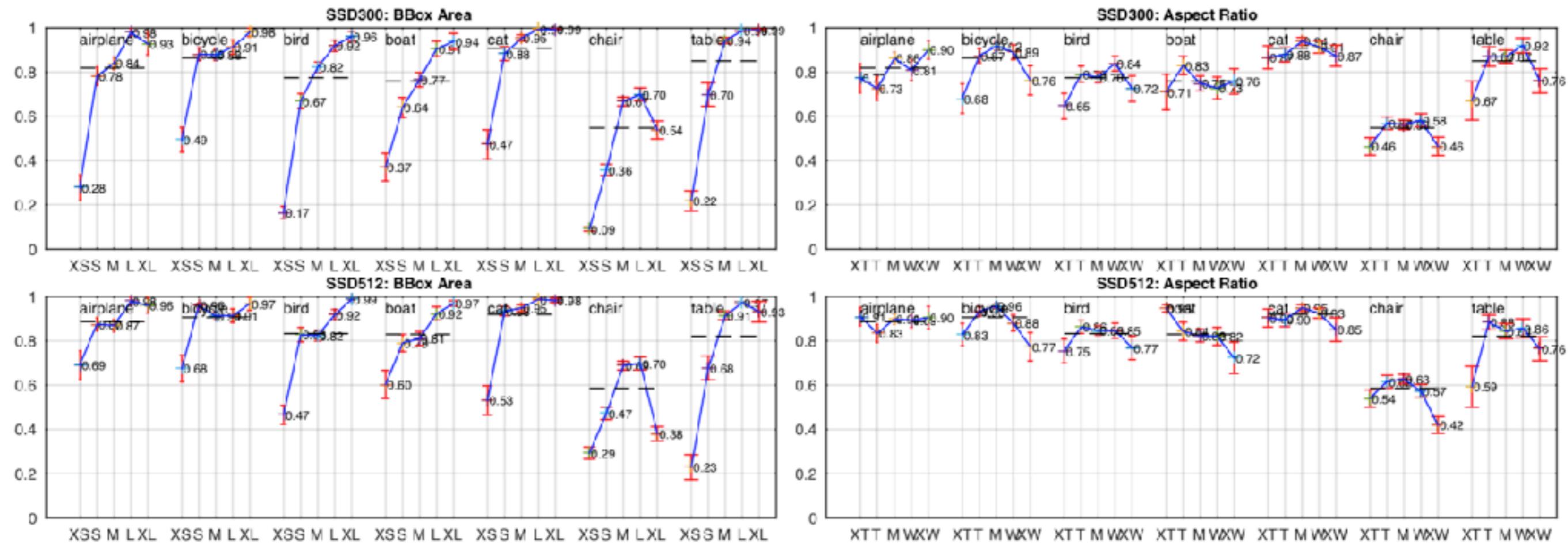
6.5 SSD效果

SSD性能

Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast [6]	07	66.9	74.5	78.3	69.2	53.2	36.6	77.3	78.2	82.0	40.7	72.7	67.9	79.6	79.2	73.0	69.0	30.1	65.4	70.2	75.8	65.8
Fast [6]	07+12	70.0	77.0	78.1	69.3	59.4	38.3	81.6	78.6	86.7	42.8	78.8	68.9	84.7	82.0	76.6	69.9	31.8	70.1	74.8	80.4	70.4
Faster [2]	07	69.9	70.0	80.6	70.1	57.3	49.9	78.2	80.4	82.0	52.2	75.3	67.2	80.3	79.8	75.0	76.3	39.1	68.3	67.3	81.1	67.6
Faster [2]	07+12	73.2	76.5	79.0	70.9	65.5	52.1	83.1	84.7	86.4	52.0	81.9	65.7	84.8	84.6	77.5	76.7	38.8	73.6	73.9	83.0	72.6
Faster [2]	07+12+COCO	78.8	84.3	82.0	77.7	68.9	65.7	88.1	88.4	88.9	63.6	86.3	70.8	85.9	87.6	80.1	82.3	53.6	80.4	75.8	86.6	78.9
SSD300	07	68.0	73.4	77.5	64.1	59.0	38.9	75.2	80.8	78.5	46.0	67.8	69.2	76.6	82.1	77.0	72.5	41.2	64.2	69.1	78.0	68.5
SSD300	07+12	74.3	75.5	80.2	72.3	66.3	47.6	83.0	84.2	86.1	54.7	78.3	73.9	84.5	85.3	82.6	76.2	48.6	73.9	76.0	83.4	74.0
SSD300	07+12+COCO	79.6	80.9	86.3	79.0	76.2	57.6	87.3	88.2	88.6	60.5	85.4	76.7	87.5	89.2	84.5	81.4	55.0	81.9	81.5	85.9	78.9
SSD512	07	71.6	75.1	81.4	69.8	60.8	46.3	82.6	84.7	84.1	48.5	75.0	67.4	82.3	83.9	79.4	76.6	44.9	69.9	69.1	78.1	71.8
SSD512	07+12	76.8	82.4	84.7	78.4	73.8	53.2	86.2	87.5	86.0	57.8	83.1	70.2	84.9	85.2	83.9	79.7	50.3	77.9	73.9	82.5	75.3
SSD512	07+12+COCO	81.6	86.6	88.3	82.4	76.0	66.3	88.6	88.9	89.1	65.1	88.4	73.6	86.5	88.9	85.3	84.6	59.1	85.0	80.4	87.4	81.2

在PASCAL VOC2007 测试集上的结果，可以看到SSD性能是最好的。Fast和Faster R-CNN输入图像最小尺寸为600， SSD为300余512，可以看到输入图像越大，效果越好。

SSD性能



SSD对较小的物体检测性能较差，这是由于小物体在高层中可能没有保留下信息。但也可以看到随着输入图片大小的增大，模型对小物体的检测性能得到提高。

SSD优缺点

- 检测速度快、性能优，几乎可以媲美目前检测性能最好的Faster R-CNN模型；
- 利用金字塔特征图，可以检测到不同大小的目标；
- 需要根据经验手动设置default box的min_size、aspect_ratio等信息，较为繁琐。
- 对小目标的召回率较低，这可能是由于低层的特征图对特征提取不够充分而致。

THANKS