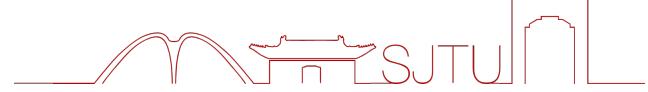




上海交通大学
SHANGHAI JIAO TONG UNIVERSITY



目标检测算法

学生创新中心 AI LAB

肖雄子彦

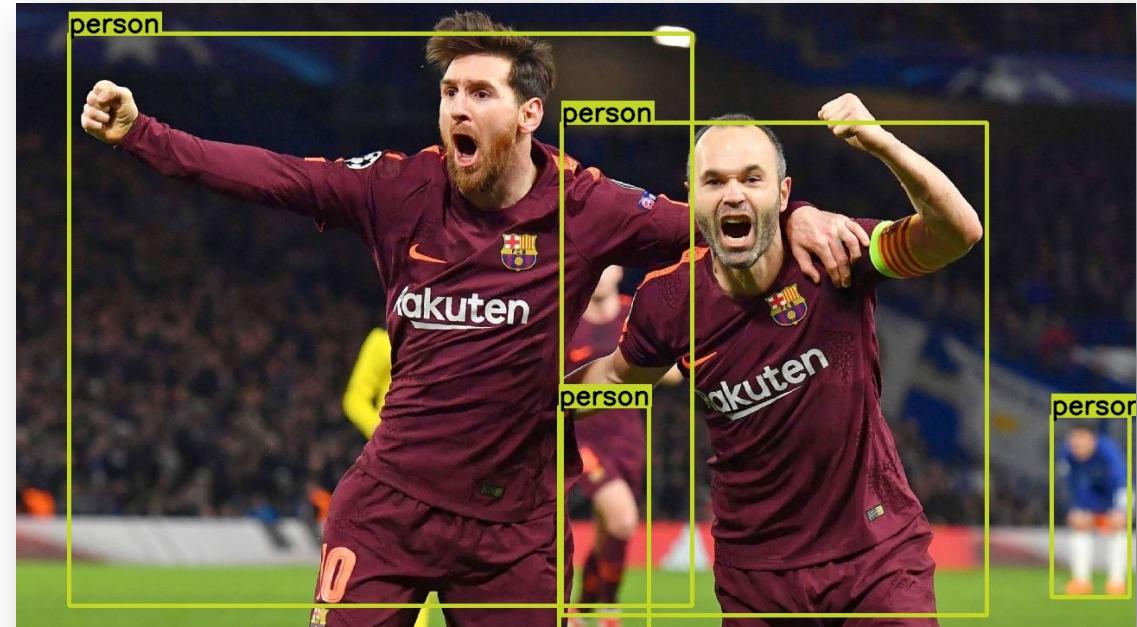
饮水思源 · 爱国荣校

Object Detection

什么是目标检测 (object detection) ?



- 输入: 图片
- 输出: 物体类别、位置坐标
- 主要评估指标: IOU



Object Detection

目标检测的滑动窗口原理：将检测问题转化为了图像分类问题。



- 采用不同大小和宽高比的窗口
- 以一定的步长，在整张图片上滑动
- 窗口选定的区域做图像分类，从而实现检测任务

Problem

候选框过多，分类器预测计算量大；
为了保证速度，分类器不能太复杂。

Solution

减少要分类的候选框



最有可能包含目标的候选框

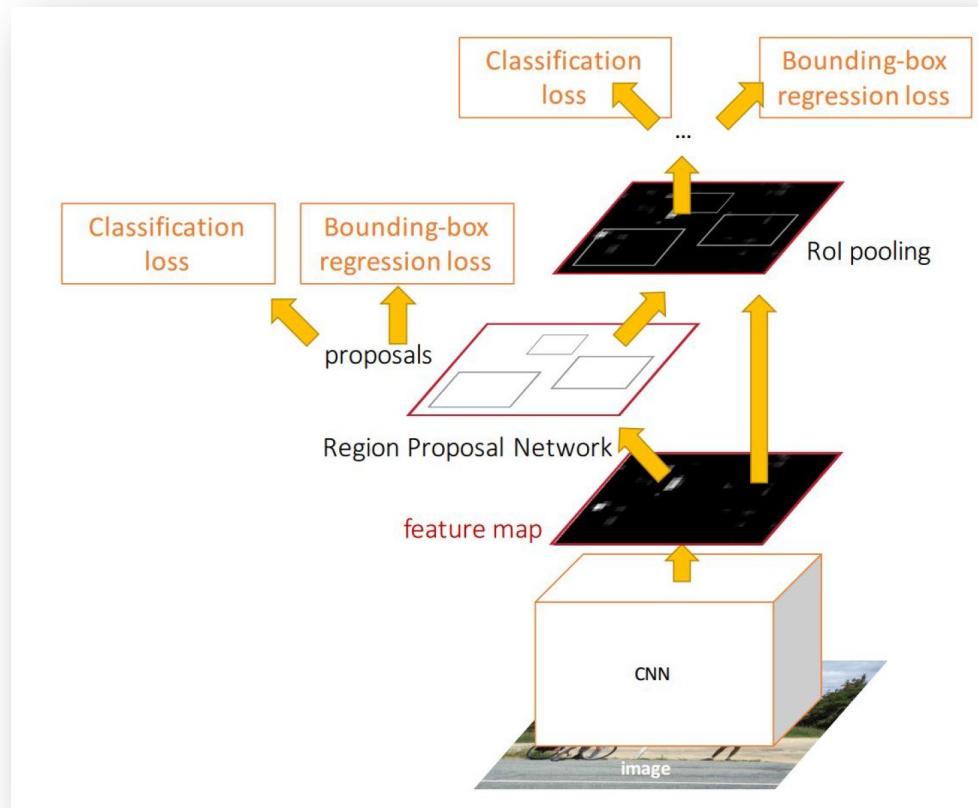
Region Proposal

Selective Search、RPN



2 stage Detection: Faster R-CNN

基于Region Proposal的R-CNN系算法
(R-CNN, Fast R-CNN, Faster R-CNN)



Faster R-CNN

Classification
&
Regression



ROI pooling (池化到相同的shape)

Region
Proposal
Network

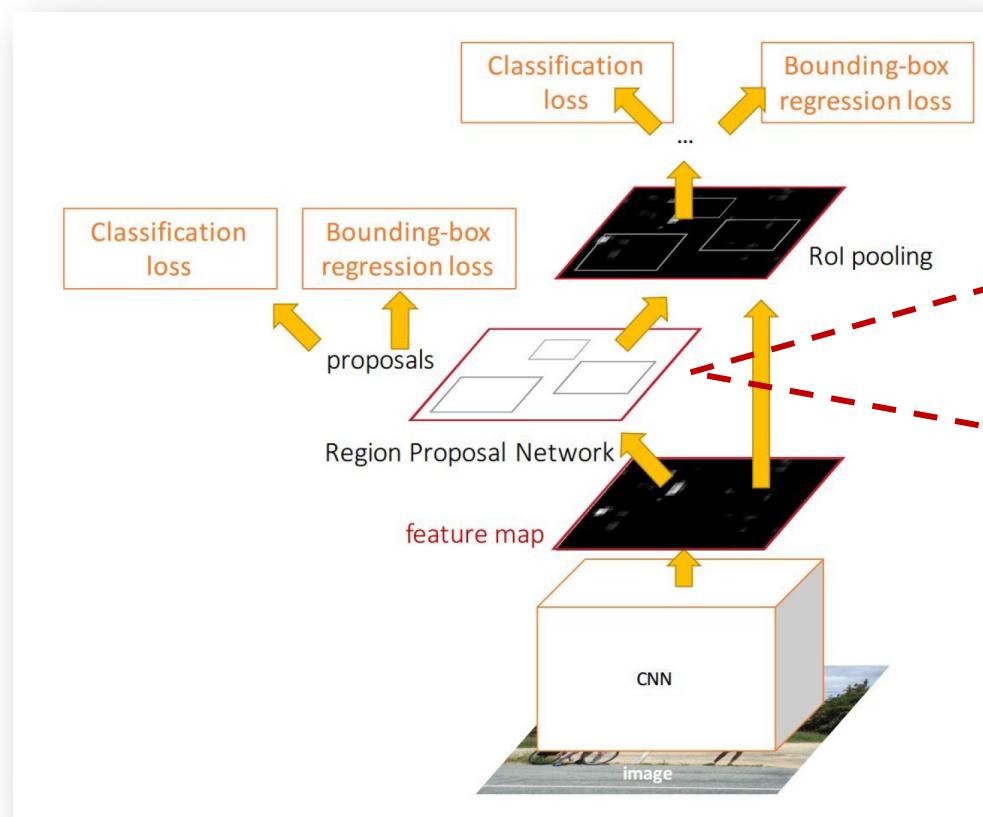
Two-stage

- 准确度高
- 算法运行耗时长

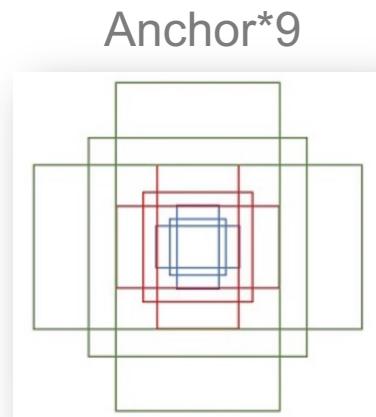
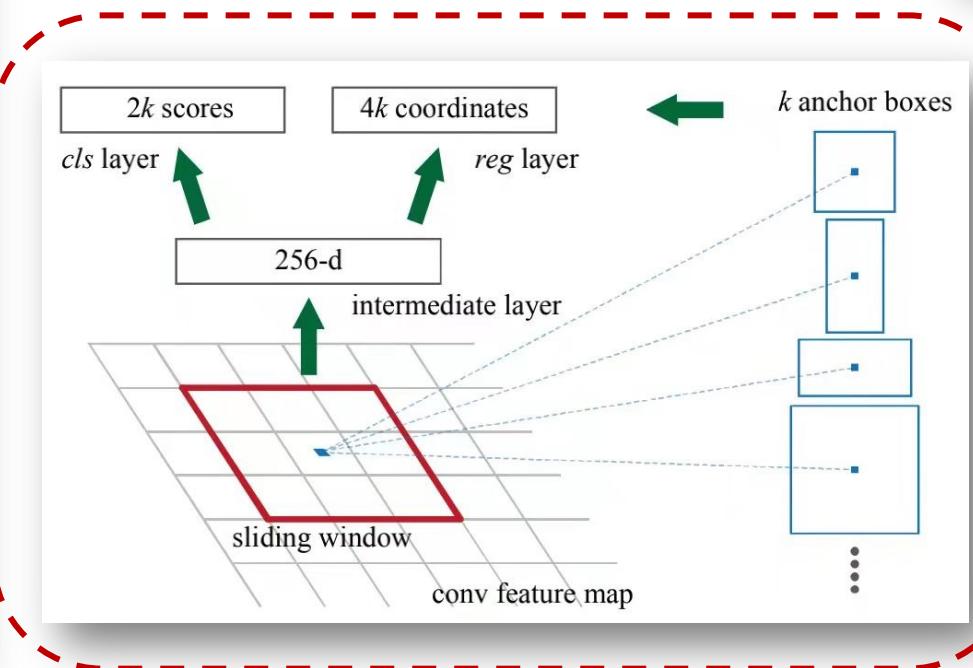
这一步不判定物体所属类别，
而是做二分类（前景 or 背景）
设定一个threshold提取有可能
包含目标的候选框。

2 stage Detection: Faster R-CNN

Region Proposal Network



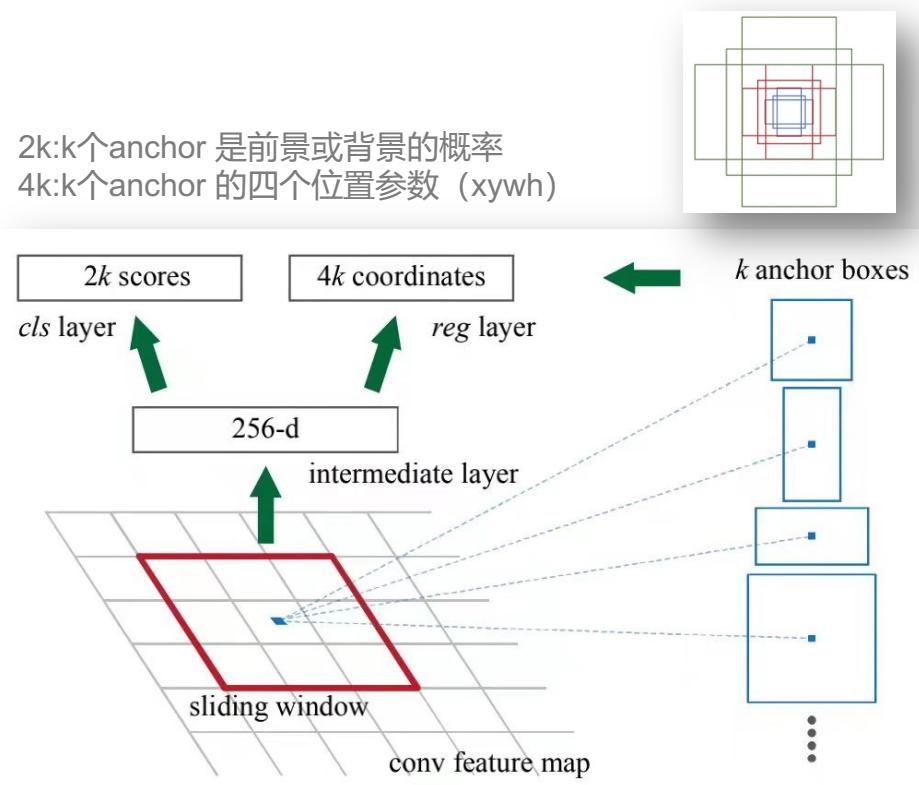
2k:k个anchor 是前景或背景的概率
4k:k个anchor 的四个位置参数 (xywh)



Faster R-CNN

2 stage Detection: Faster R-CNN

Training RPN



- 剔除超出边界的Anchor
- Anchor通过回归生成候选框
- 对候选框采取NMS剔除部分候选框
- 选取一定数量的正样本和负样本用于训练 (原文128个)
正样本: 与Ground Truth的IOU>0.7或IOU最大的候选框
负样本: 与Ground Truth的IOU<0.3的候选框
- RPN损失函数 (加权平均)

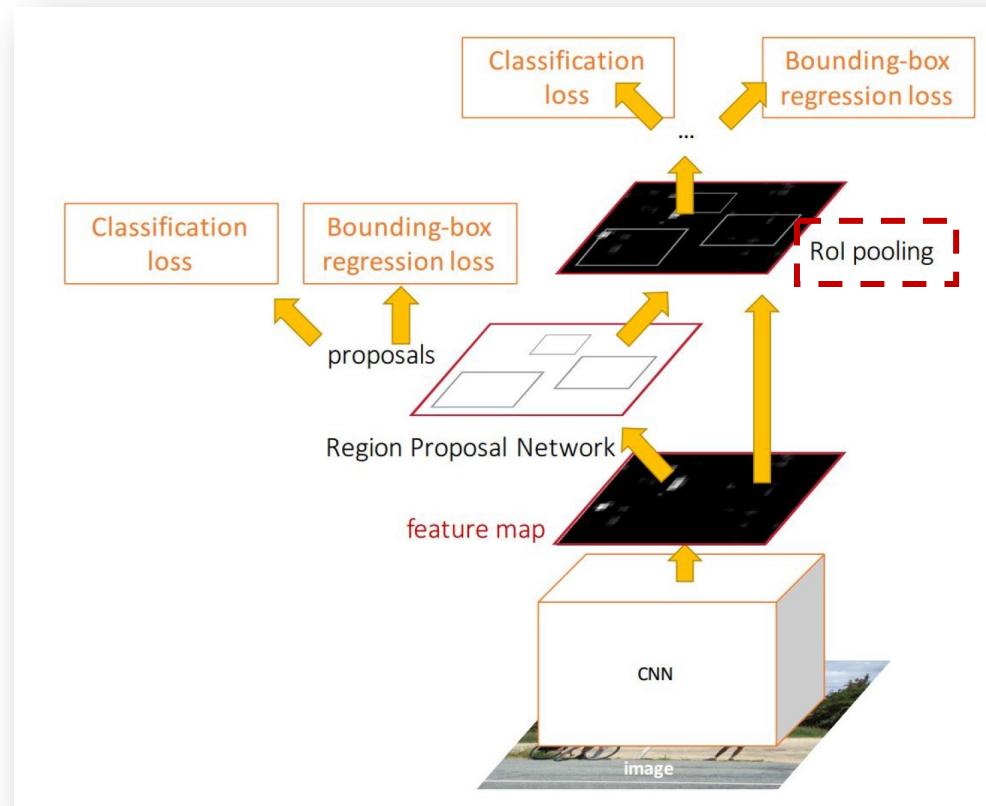
$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$$

分类损失 边界框回归损失

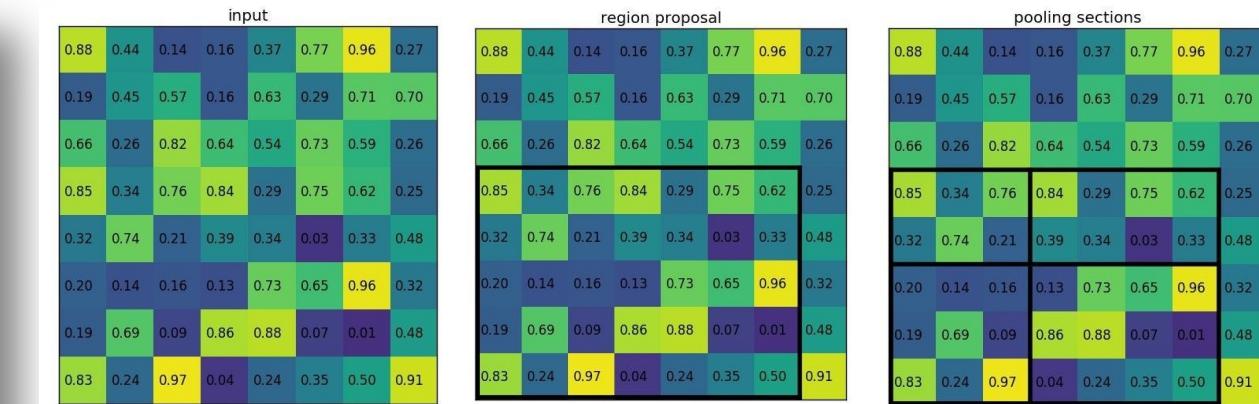
2 stage Detection: Faster R-CNN

Region of Interest (ROI) Pooling

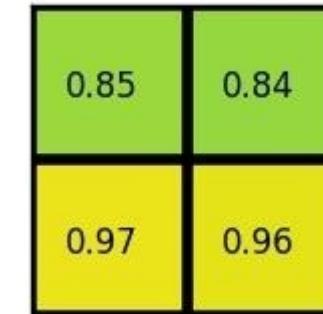
把每个proposal特征都统一为相同的feature map



Faster R-CNN

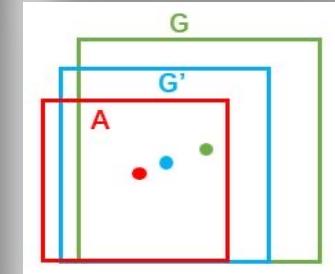
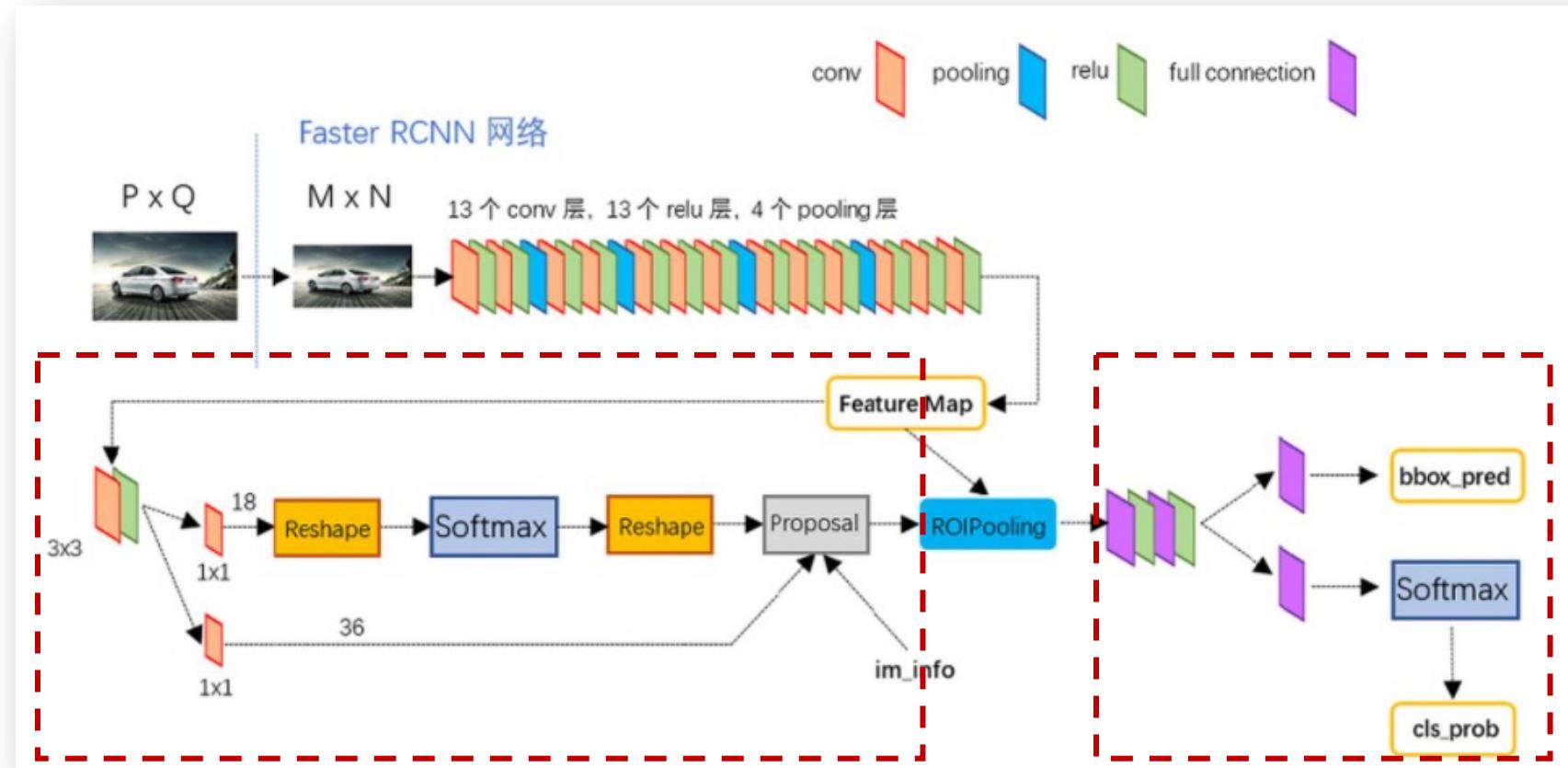


特征矩阵通过ROI pooling
层缩放到 7×7 大小的特征图
并展平送入后面的FC层。



2 stage Detection: Faster R-CNN

整体网络结构

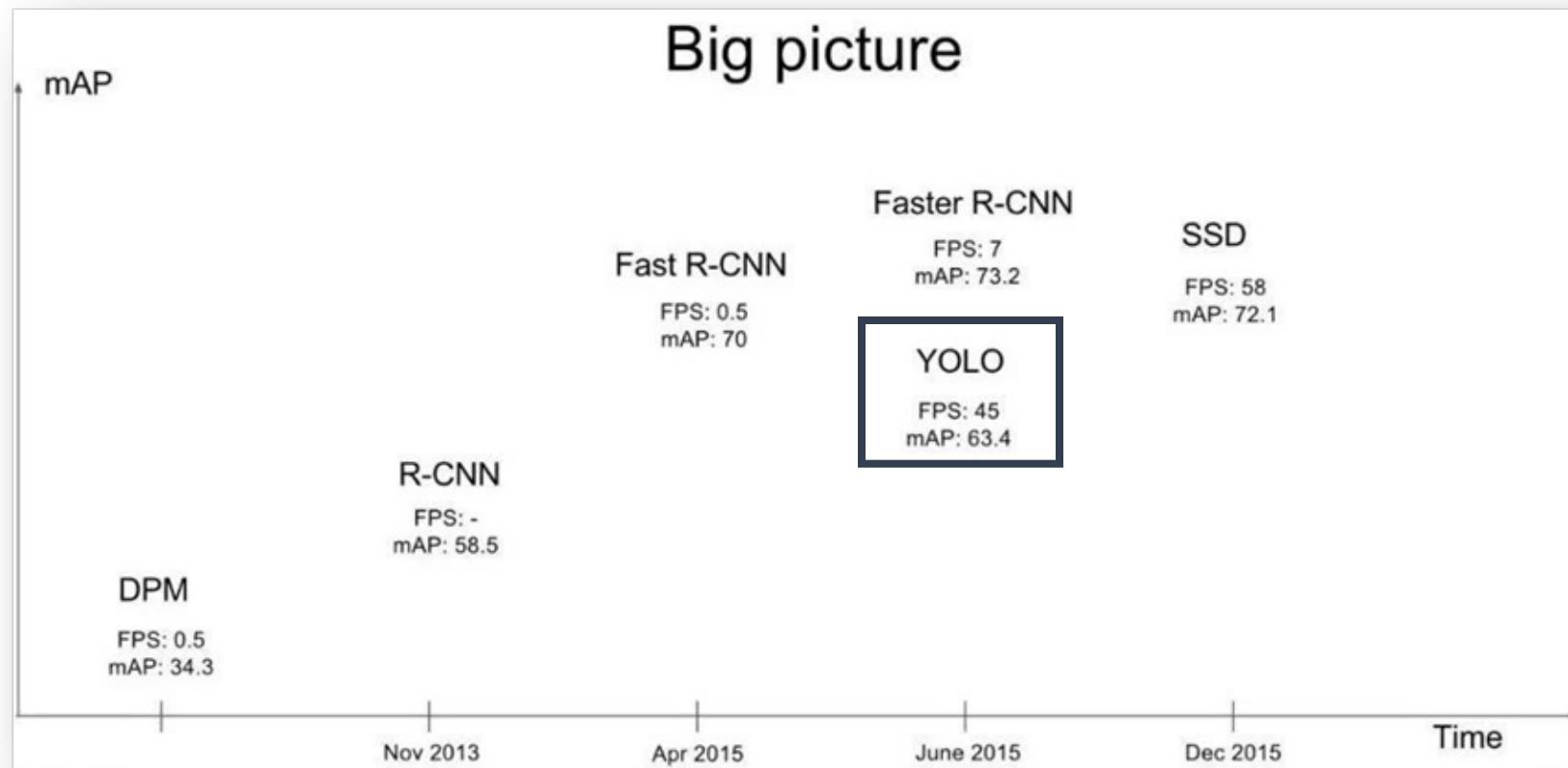


全连接网络：

- 更精确预测框 (bounding box)
- 类别概率

RPN网络：筛选候选框；初步回归预测框

1 stage Detection: YOLO



One-stage

- 速度快
- 准确率不足

- 分类
 - 边框预测
- 同时进行

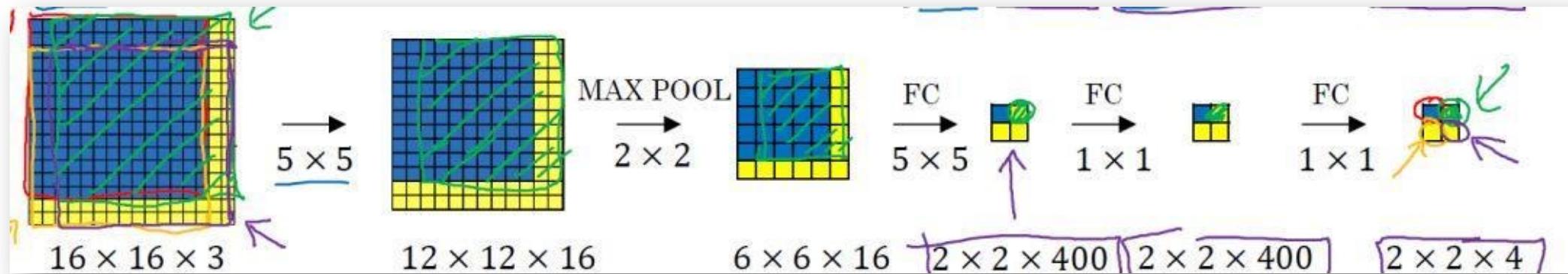
You Only Look Once: Unified, Real-Time Object Detection

仅使用一个CNN网络，利用这个统一的框架，提供端到端的实时预测，速度快。

YOLO 算法原理

YOLO思路：

滑动窗口非常耗时，而CNN在卷积运算的过程中就形成了滑动窗口的选择



2x2图上的每个元素都和原图位置是一一对应的
可以想象成经过了 14×14 的kernel，步长为2的卷积得到

一次CNN计算就可以实现窗口的滑动、子区域的分类预测 (output channel)

CNN卷积操作的特点：图片信息的空间相对位置不变

YOLO 算法原理

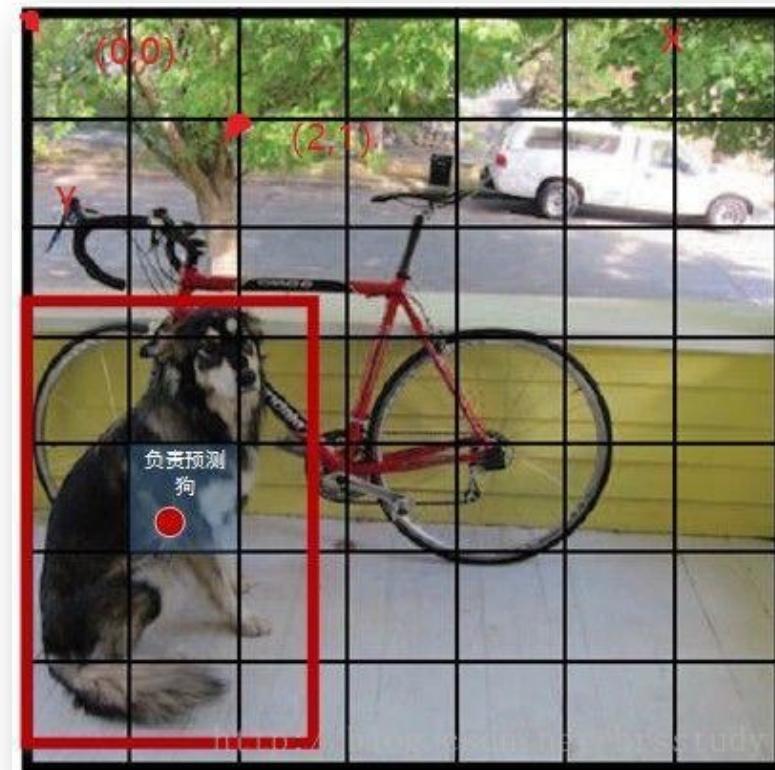
YOLO在一次CNN计算中，直接从原始图像上预测
物体类别(class probabilities)和位置边界框(bounding boxes)

Step1 划分单元格

- YOLO将输入的图分割成 $S \times S$ 网格
- 如某个object的中心落在这个单元格中，
则该单元格就负责预测这个object

Step2 每个单元格预测

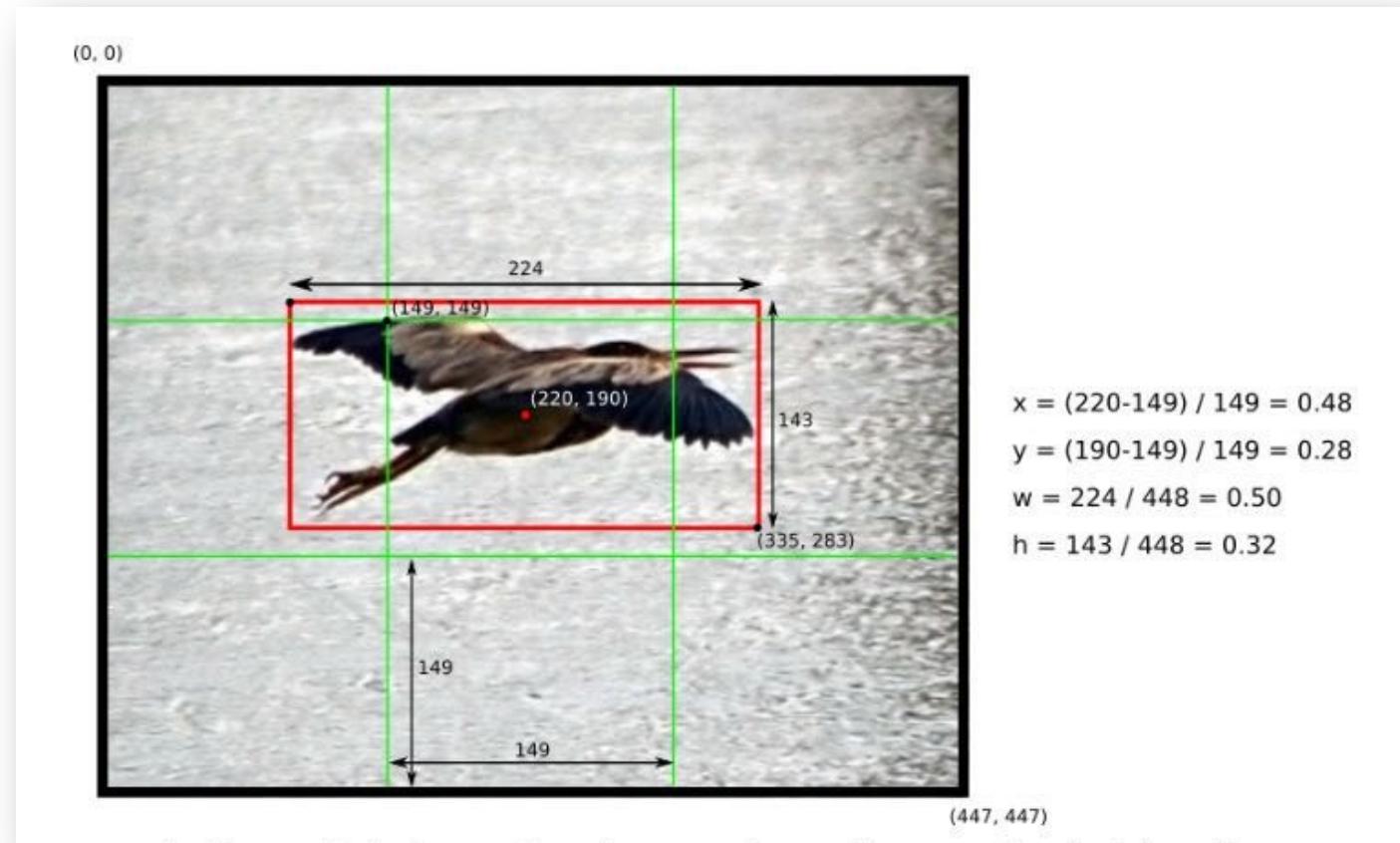
- Bounding Box：边界框的坐标 $x\backslash y\backslash w\backslash h$
- Confidence：每个边界框的置信度
- Class：图像包含的类别数，如20类



Ground Truth

YOLO 算法原理

| 单元格任务1：每个单元格预测B个边界框（bounding box）



边界框
(bounding box)

回归

(x,y)
中心点坐标
(w,h)
宽度和高度
confidence

YOLO 算法原理

| 单元格任务2：每个bbox都有一个置信度 (confidence score)

$$\text{Confidence} = P_r(\text{Object}) * \text{IOU}_{pred}^{truth}$$

0-1之间的值

边界框含有目标的可能性

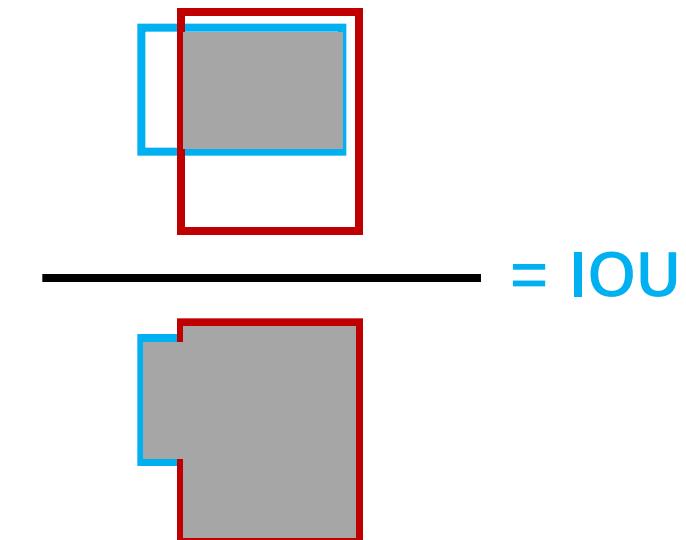
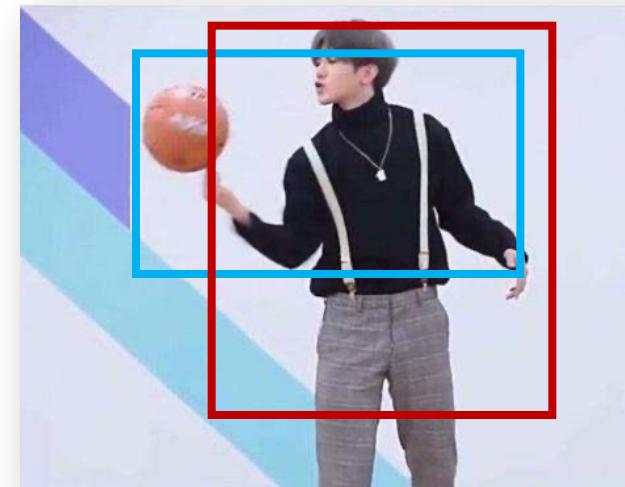
- 当这个cell不存在物体时

$$P_r(\text{Object}) = 0$$

- 而这个cell存在物体时

$$P_r(\text{Object}) = 1$$

预测框与实际框的交并比， IOU (intersection over union)



YOLO 算法原理

单元格任务3：分类 (classification)

- 每一个单元格还要预测出C个类别概率

$$Pr(class_i|object)$$

在YOLO里, 每个单元格预测一组类别概率, 而不是每个边界框预测一组类别概率

这些概率值是在各个边界框置信度下的条件概率

- 测试阶段, 需要计算各个边界框类别置信度 (class-specific confidence scores) :

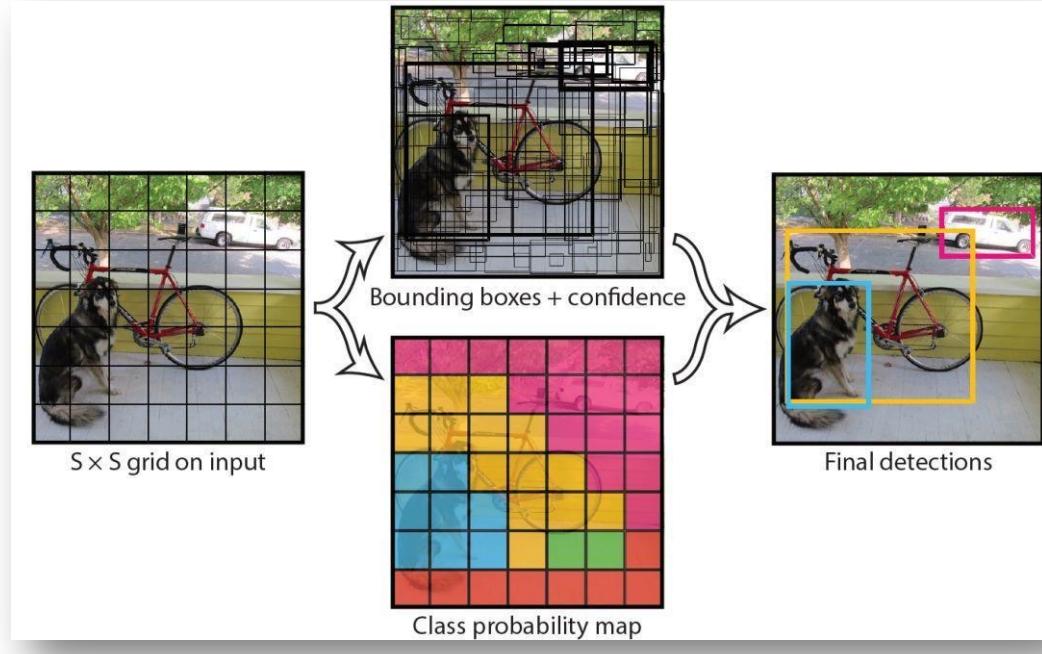
$$Pr(class_i|object) * \underbrace{Pr(object)}_{confidence\ score} * IOU_{pred}^{truth} = Pr(class_i) * IOU_{pred}^{truth}$$

用于预测时边界框的筛选

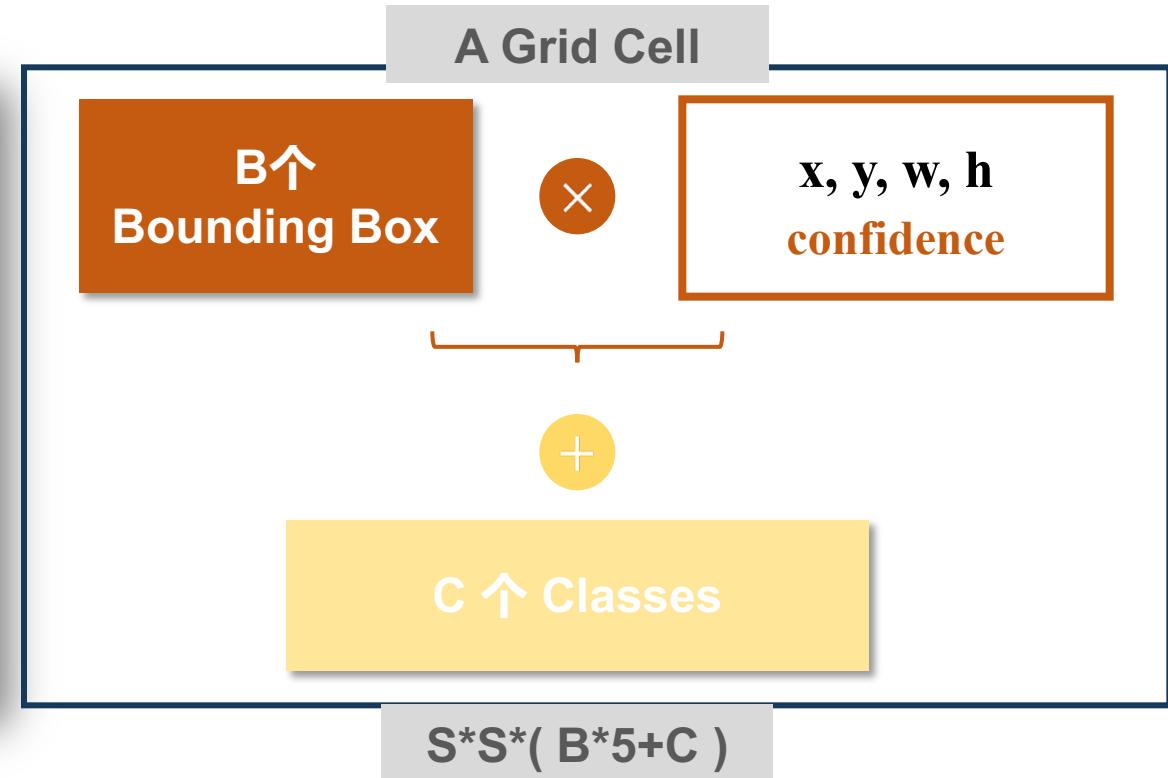


YOLO 算法原理

最终输出的张量size：

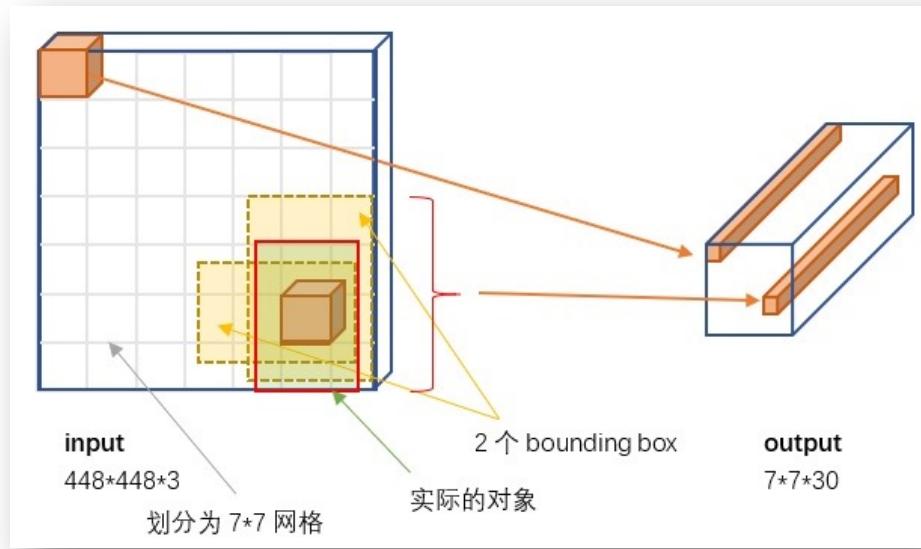


$$S = 7 \quad B = 2 \quad C = 20$$

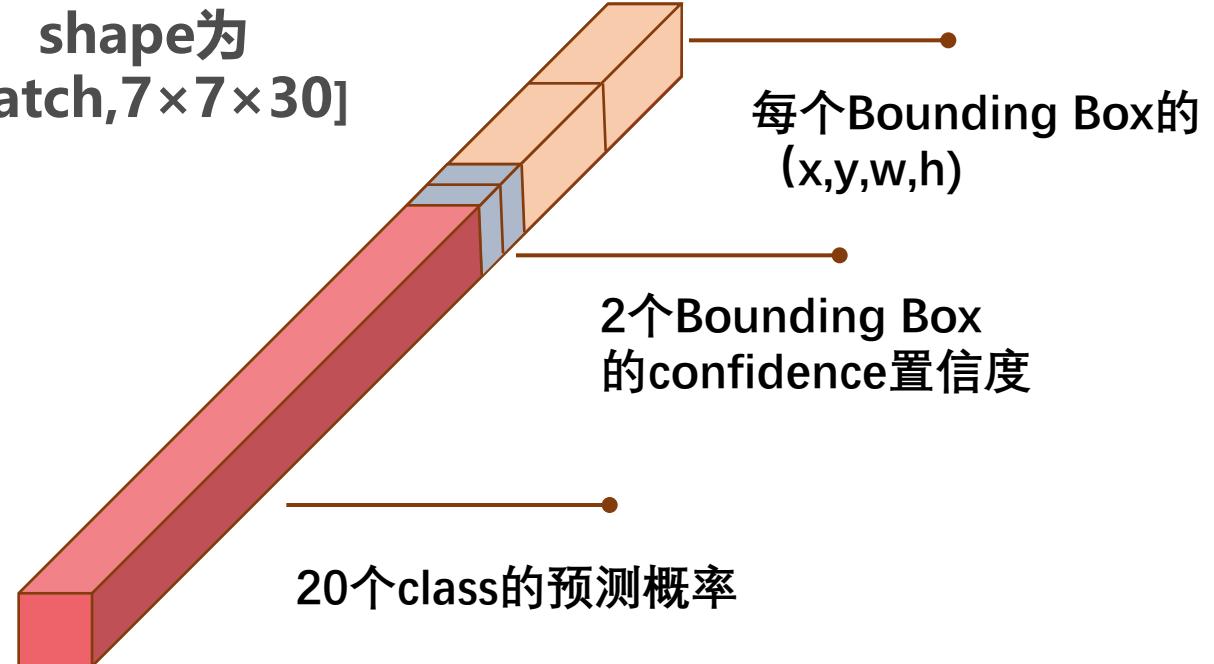


那么，最终预测值张量大小为： $7 * 7 * (2 * 5 + 20) = 7 * 7 * 30$

YOLO 算法原理



shape为
[batch,7×7×30]



类别概率

$$P[:, 0:7 * 7 * 20]$$

置信度

$$P[:, 7 * 7 * 20:7 * 7 * (20 + 2)]$$

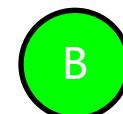
边框参数

$$P[:, 7 * 7 * (20 + 2) :]$$

在YOLO算法中，什么叫含有object（目标）的cell（网格）？



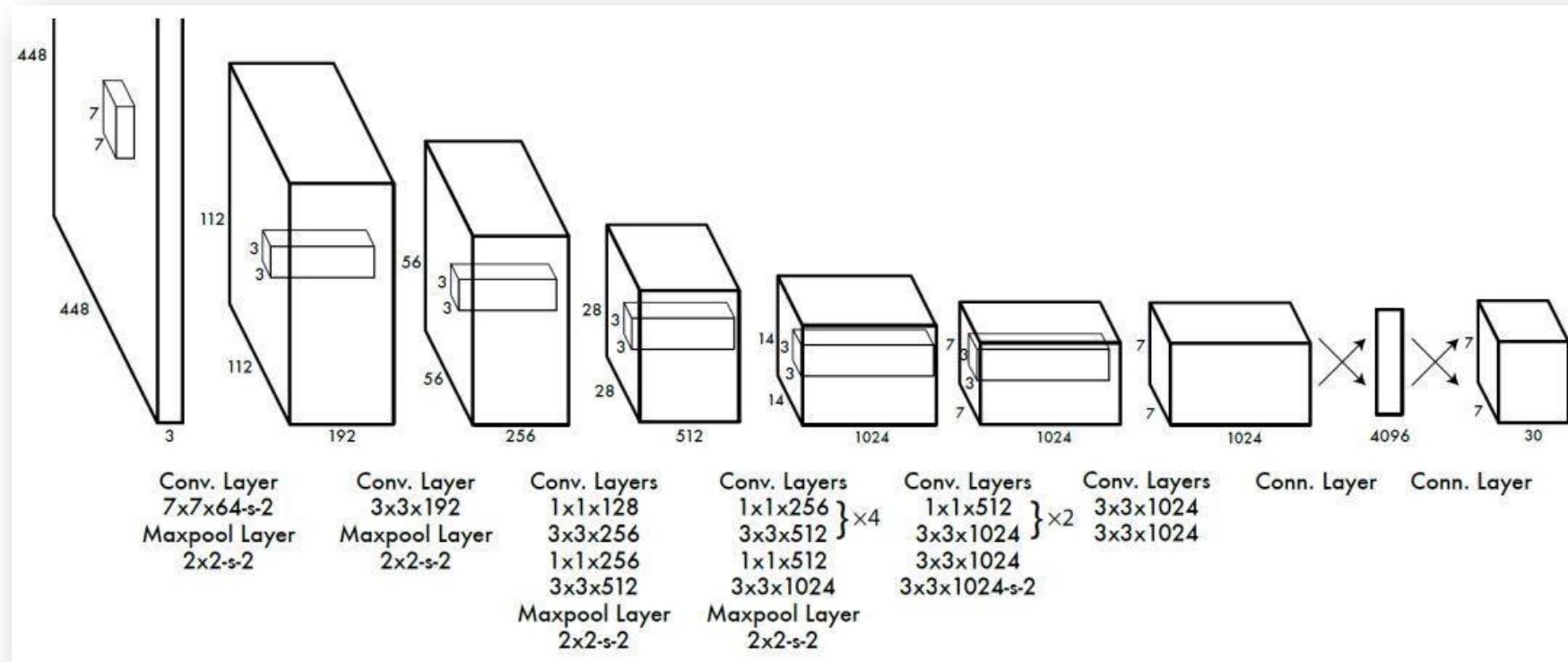
Object所覆盖的网格



Object的中心点落入的那个网格

提交

YOLO 网络设计



- 采用卷积网络来提取特征，然后使用全连接层来得到预测值。
- 网络结构参考GoLeNet模型，包含24个卷积层和2个全连接层。
- 对于卷积层和全连接层，采用Leaky ReLU激活函数 $\max(x, 0.1x)$ 。

YOLO 损失函数

Yolo算法将目标检测看成回归问题，所以采用的是均方差损失函数。
但是对不同的部分采用了不同的权重。

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

同样的偏移对大bbox影响较小，对小bbox影响较大

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

判断第i个网格中的第j个
box是否负责这个object

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

含object的box的
confidence预测

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

不含object的box的
confidence预测

判断是否有object中
心落在网格i中

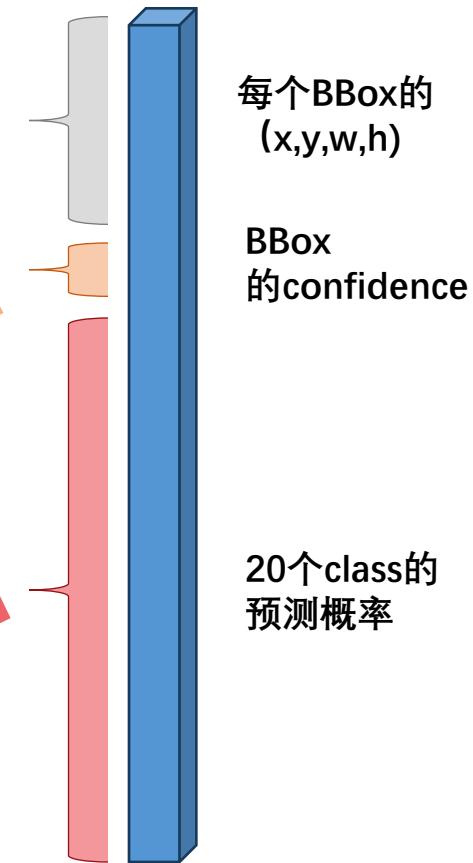
$$+ \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

定位误差

坐标预测

置信度误差

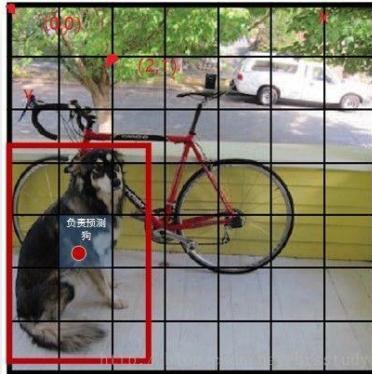
类别预测



YOLO 损失函数

含有Object的BBox

- 存在目标(中心落入该cell)
- Cell中这个BBox的IOU最大



剩下的全是不含 Object的BBox

- 存在目标，但IOU小
- 不存在目标的BBox

✓

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

坐标预测

✓

判断第i个网格中的第j个
box是否负责这个object

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

含object的box的
confidence预测

✓

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

不含object的box的
confidence预测

✓

判断是否有object中
心落在网格i中

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

类别预测

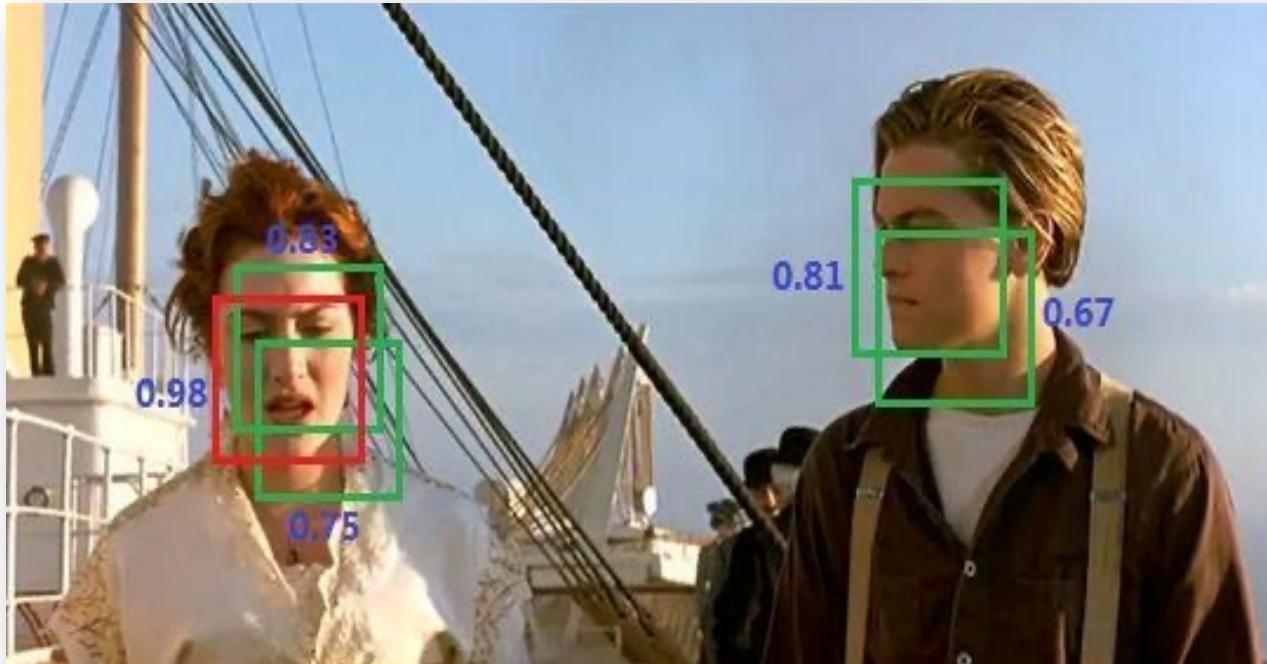
✓

测试阶段框太多? NMS

非极大值抑制算法 (non maximum suppression, NMS)
主要解决一个目标被多次检测的问题

$$P_r(class_i|object) * P_r(object) * \text{IOU}_{pred}^{truth} = P_r(class_i) * \text{IOU}_{pred}^{truth}$$

confidence score



- ① 找到类别置信度最大的边界框
- ② 计算其与剩余框的IOU
- ③ 如果IOU大于阈值（重合度过高），那么就将该框剔除
- ④ 寻找类别置信度第二高的框，重复上述过程，直到处理完所有的检测框。

YOLO 算法

优点

- YOLO检测速度快，能够实时处理流媒体视频。
- 较RCNN/Fast-RCNN等算法相比，背景误检数量少了一半。
(因为YOLO是对输入图像进行全局处理，与滑动窗口/区域提取方式相比，能够有效获取上下文信息)
- YOLO泛化能力强，模型鲁棒性高。

缺点

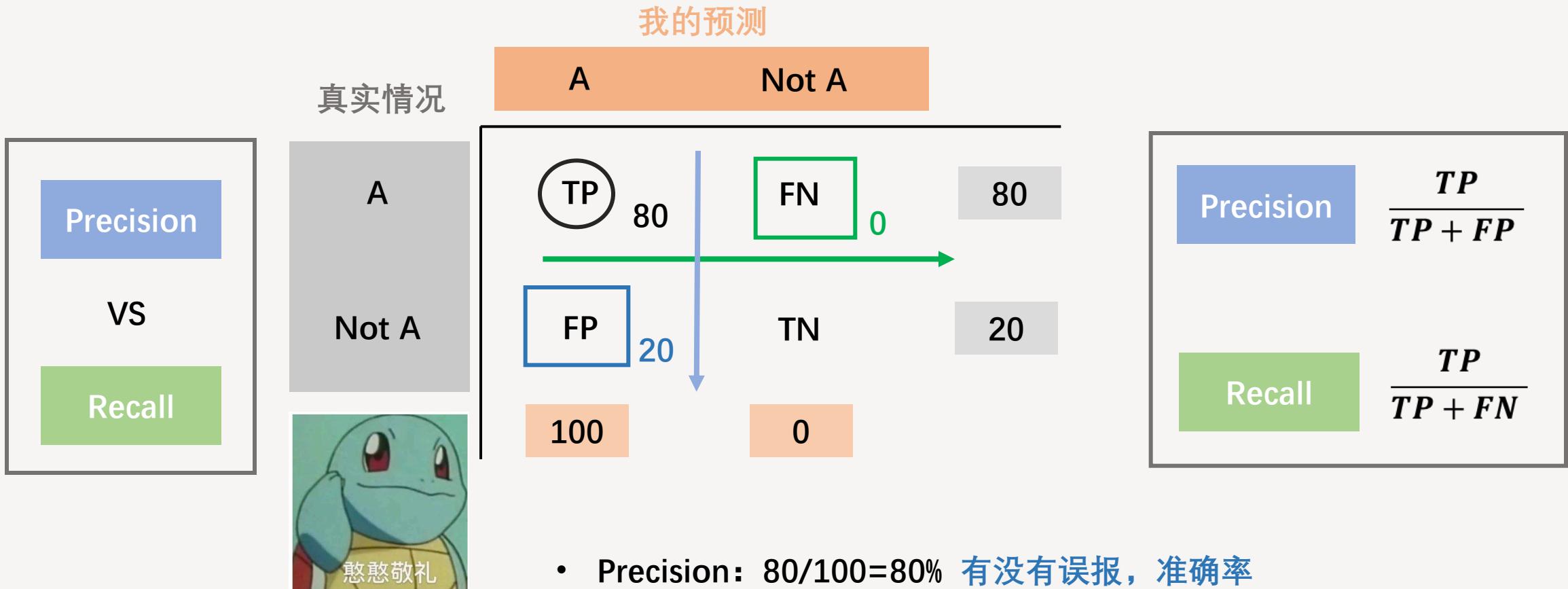
- 物体定位精准性较差 (损失函数的问题；正负样本比例失衡…)
- 对小目标、密集出现的物体(鸟群)检测较差。
- 当一个小格中出现多个小物体、或一个小格中出现多个不同物体时效果欠佳。



关于评估方法：mAP

- mAP: Mean Average Precision

Precision和Recall往往是trade off的

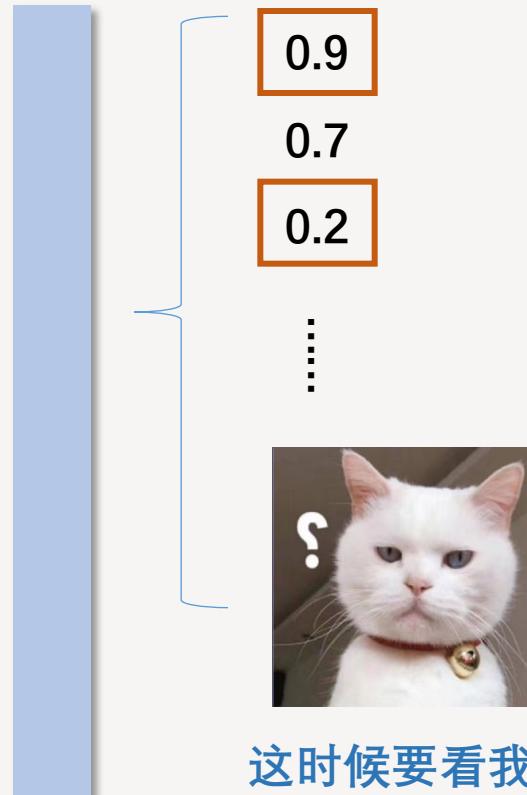


关于mAP

- 检测问题中怎么算TP? $IOU > 0.5$

Confidence score

Bounding Box



这时候要看我们
设置的threshold

设置不同的threshold会得到不同的precision和recall

- 如果threshold太高，例如设为0.9
筛选非常严格，这时候一旦认为是猫的基本都是猫

Precision



但对于score较低(如0.7)，但确实是猫的样本就会漏掉

Recall



- 如果threshold太低，不是猫也都算猫，很全但不准。
precision就会很低，recall就会很高

Precision



Recall



Precision和Recall不是一个绝对的东西，
而是相对threshold改变的。

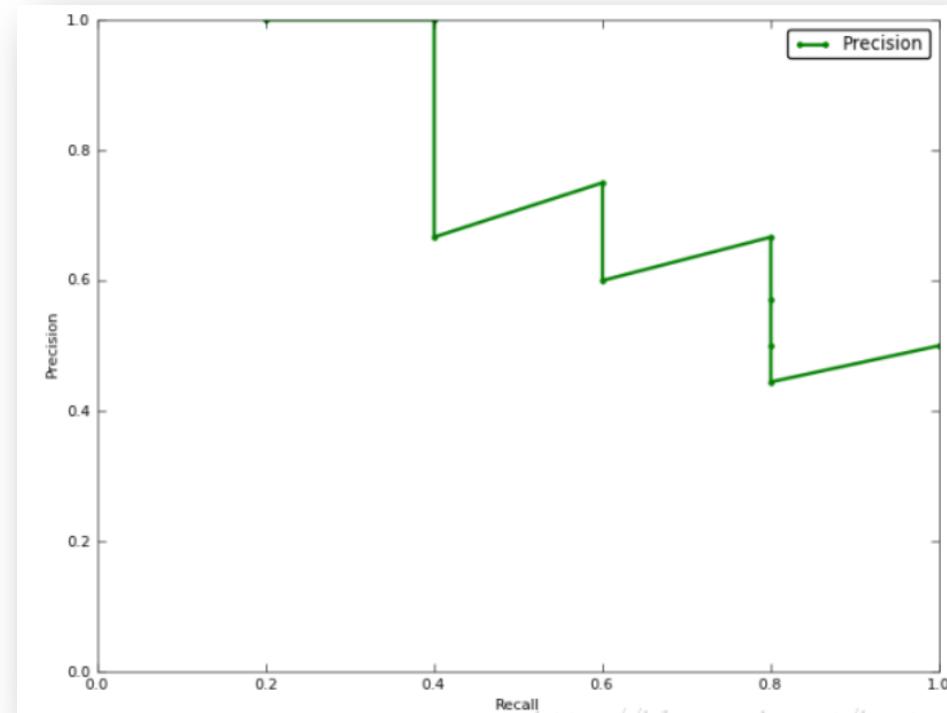
关于mAP

因此，单用precision/recall来作为标准判断不太合适，需要两个一起动态考虑

- 以猫这个分类为例

每个不同的threshold，
都可以取到一组 (precision, recall)
因此调整threshold得到一条
precision-recall curve

猫这个类的Average Precision
即curve下的面积



每个类别都分别计算precision和recall，得到PR曲线，求均值就是mAP.

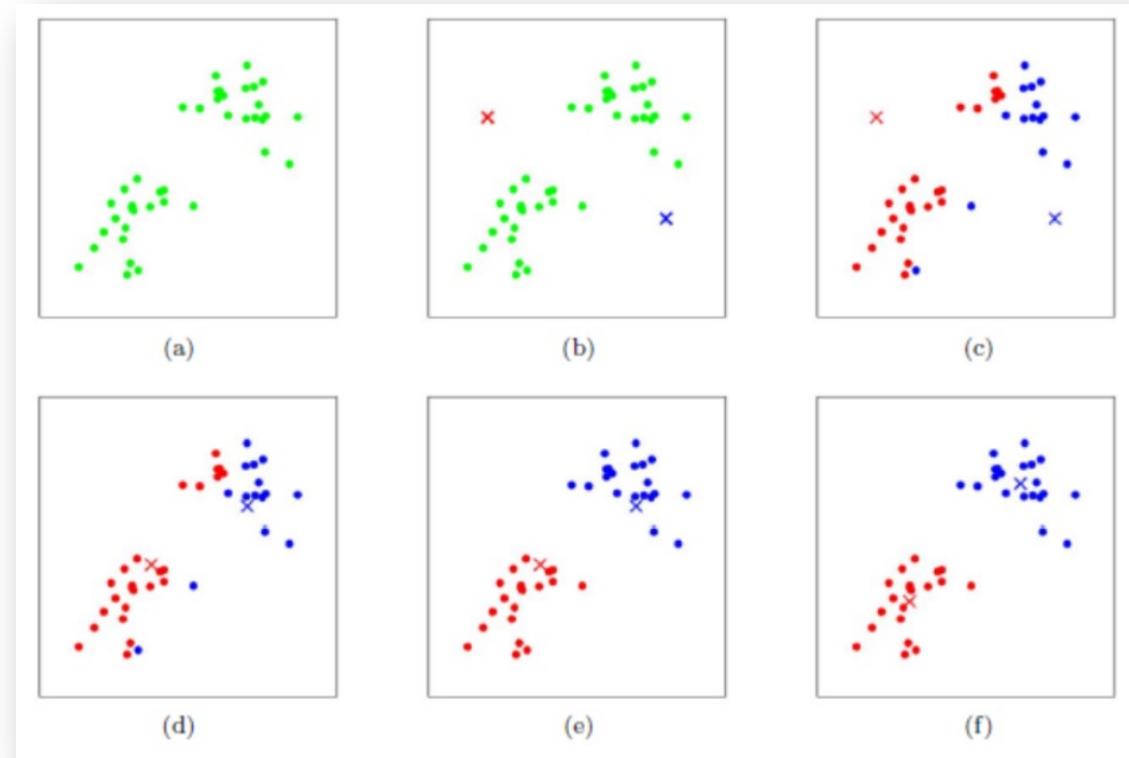
YOLO 改进：采用先验框Anchor box

Anchor box最初是由Faster RCNN引入

使用K-means方法从训练集的所有Ground Truth Box中统计最常出现的几个box形状、尺寸

K-means方法

BBox在训练的时候只需要在Anchor box基础上位移缩放，加速收敛



YOLO 改进：采用先验框Anchor box

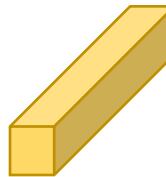
通过对训练样本的Ground Truth Box的聚类，
选出了3*3个具有代表性形状的宽和高。

(373x326) (156x198) (116x90)

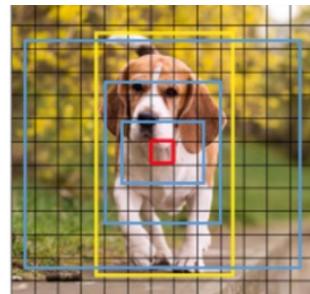
(59x119) (62x45) (30x61)

(33x23) (16x30) (10x13)

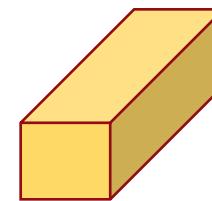
全局



13*13*255



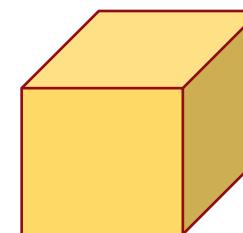
大感受野



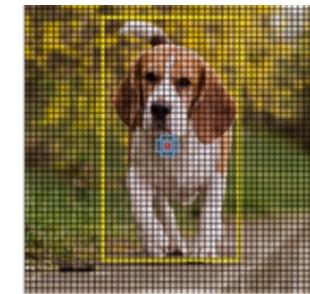
26*26*255



中感受野



52*52*255



小感受野

局部

Bounding box 预测

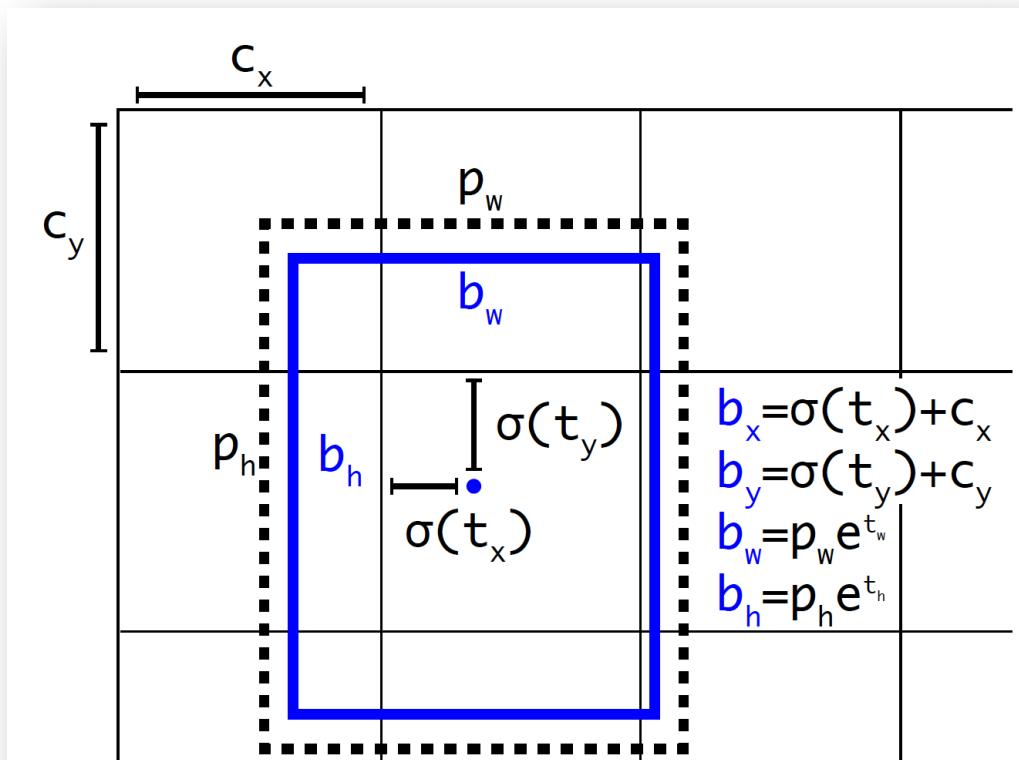


Figure 2. **Bounding boxes with dimension priors and location prediction.** We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function. This figure blatantly self-plagiarized from [15].
https://blog.csdn.net/qq_41994006

Sigmoid函数 $\frac{1}{1+e^{-x}}$ 将 t_x 映射到0~1范围内

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \end{aligned}$$

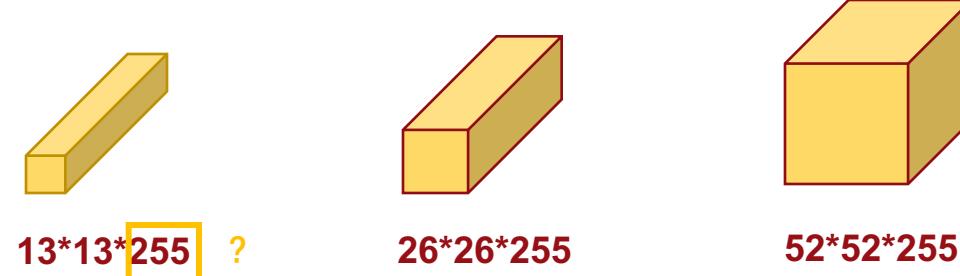
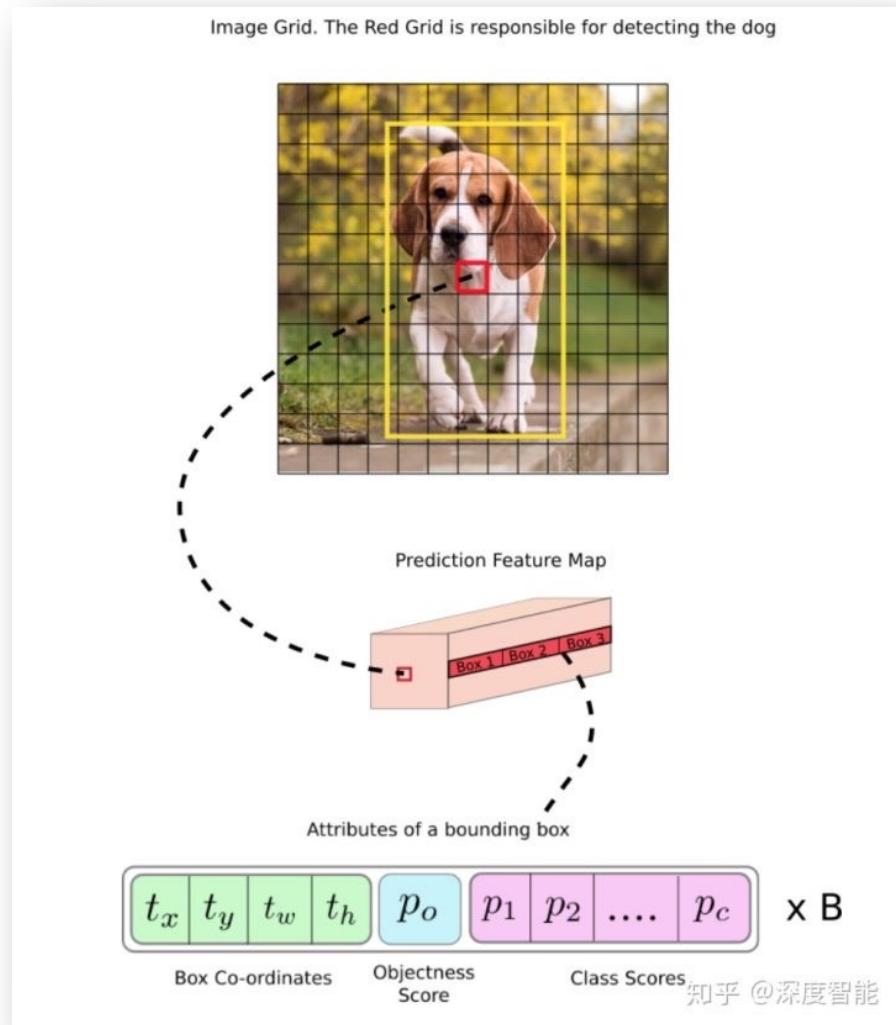
网格的左上角坐标
每个cell单位是1

用来控制
bbox缩放

t_x, t_y, t_w, t_h 是要学习的参数

Anchor的宽高

YOLO v3 参数



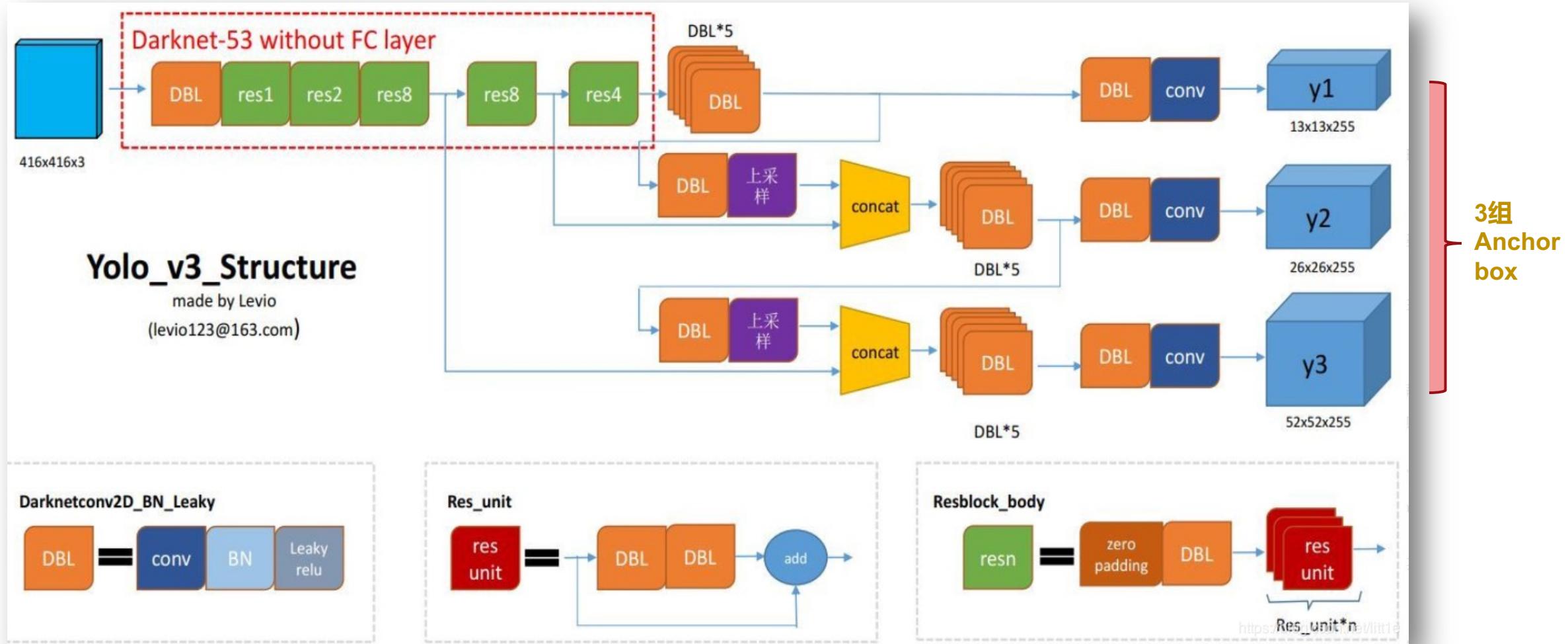
每个网格单元预测3个box

- 每个box有($x, y, w, h, confidence$)五个基本参数，
- 每个box要预测80个类别的概率

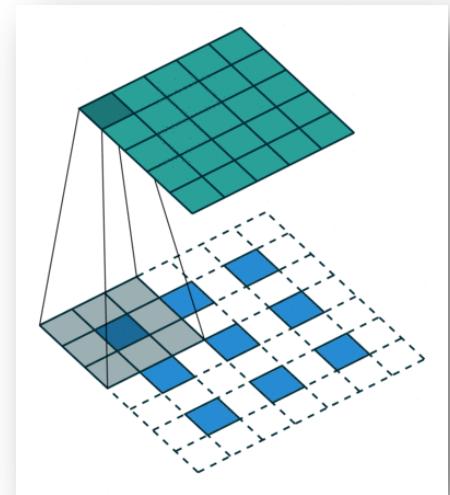
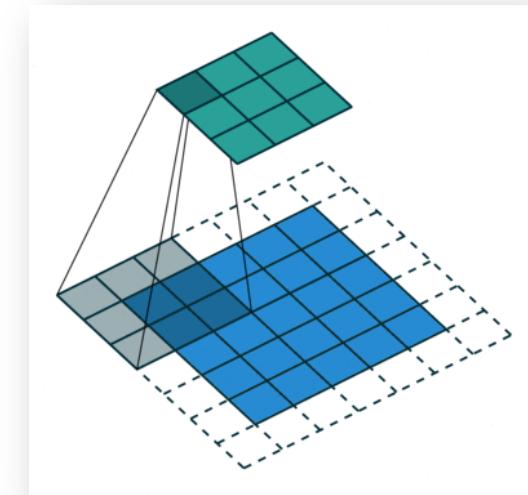
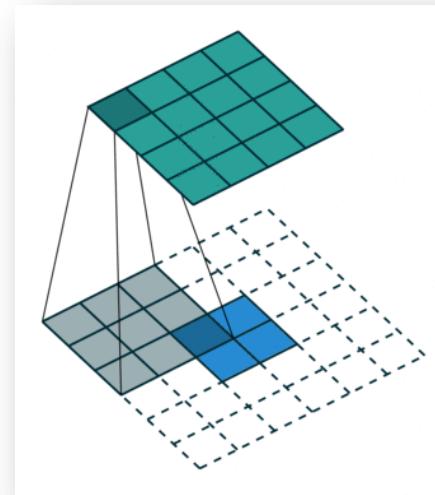
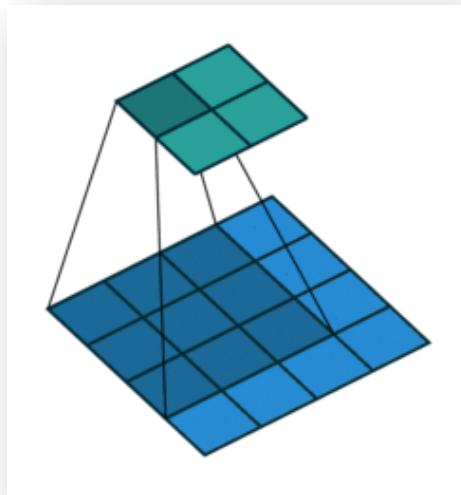
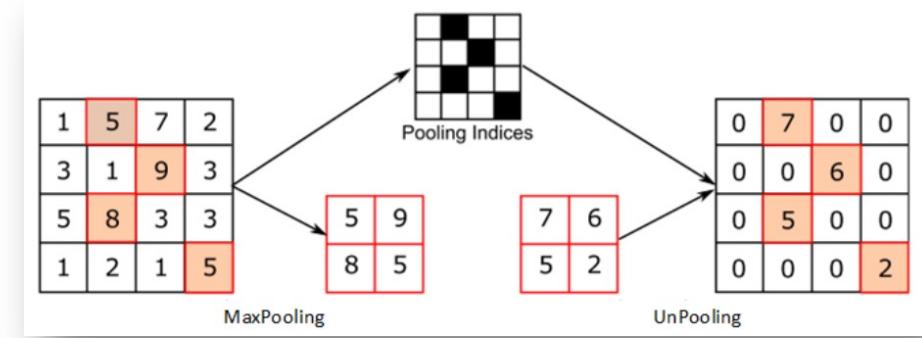
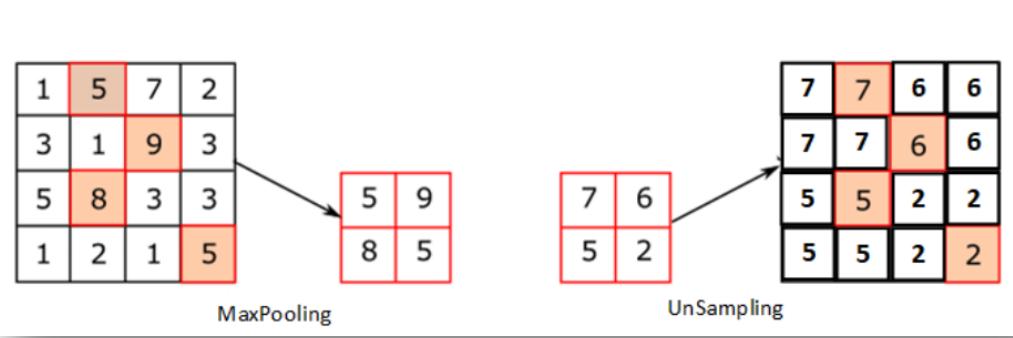
所以每个单元格需要预测的参数量：

$$3 * (5 + 80) = 255$$

YOLOv3 多尺度检测



上采样 \ 上池化



https://github.com/vdumoulin/conv_arithmetic

Backbone

- 整个v3结构里面，没有池化层和全连接层。尺寸变换是通过改变卷积核的步长来实现。
- 比如stride=(2, 2)，等于将图像边长缩小了一半(即面积缩小到原来的1/4)。

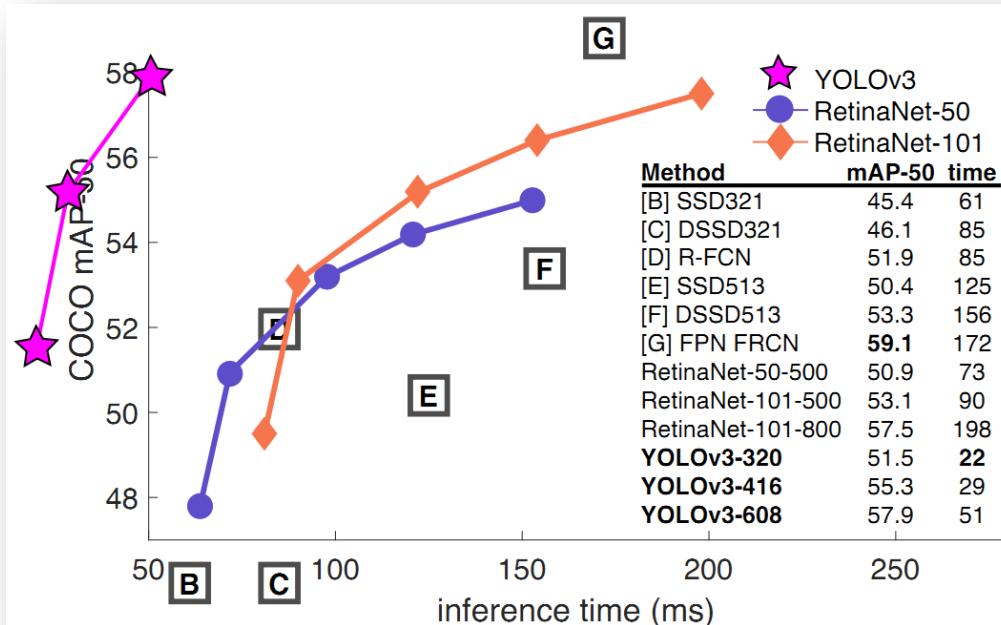
Yolo v2 base net				
Type	Filters	Size/Stride	Output	
Convolutional	32	3 × 3	224 × 224	
Maxpool		2 × 2 / 2	112 × 112	
Convolutional	64	3 × 3	112 × 112	
Maxpool		2 × 2 / 2	56 × 56	
Convolutional	128	3 × 3	56 × 56	
Convolutional	64	1 × 1	56 × 56	
Convolutional	128	3 × 3	56 × 56	
Maxpool		2 × 2 / 2	28 × 28	
Convolutional	256	3 × 3	28 × 28	
Convolutional	128	1 × 1	28 × 28	
Convolutional	256	3 × 3	28 × 28	
Maxpool		2 × 2 / 2	14 × 14	
Convolutional	512	3 × 3	14 × 14	
Convolutional	256	1 × 1	14 × 14	
Convolutional	512	3 × 3	14 × 14	
Convolutional	256	1 × 1	14 × 14	
Convolutional	512	3 × 3	14 × 14	
Maxpool		2 × 2 / 2	7 × 7	
Convolutional	1024	3 × 3	7 × 7	
Convolutional	512	1 × 1	7 × 7	
Convolutional	1024	3 × 3	7 × 7	
Convolutional	512	1 × 1	7 × 7	
Convolutional	1024	3 × 3	7 × 7	
Convolutional	1000	1 × 1	7 × 7	
	Avgpool		Global	1000
	Softmax			
Yolo v3 base net				
Type	Filters	Size	Output	
Convolutional	32	3 × 3	256 × 256	
Convolutional	64	3 × 3 / 2	128 × 128	
1x	Convolutional	32	1 × 1	
Convolutional	64	3 × 3	128 × 128	
Residual				
Convolutional	128	3 × 3 / 2	64 × 64	
2x	Convolutional	64	1 × 1	
Convolutional	128	3 × 3	64 × 64	
Residual				
Convolutional	256	3 × 3 / 2	32 × 32	
8x	Convolutional	128	1 × 1	
Convolutional	256	3 × 3	32 × 32	
Residual				
Convolutional	512	3 × 3 / 2	16 × 16	
8x	Convolutional	256	1 × 1	
Convolutional	512	3 × 3	16 × 16	
Residual				
Convolutional	1024	3 × 3 / 2	8 × 8	
4x	Convolutional	512	1 × 1	
Convolutional	1024	3 × 3	8 × 8	
Residual				
Avgpool		Global		
Connected			1000	
Softmax				



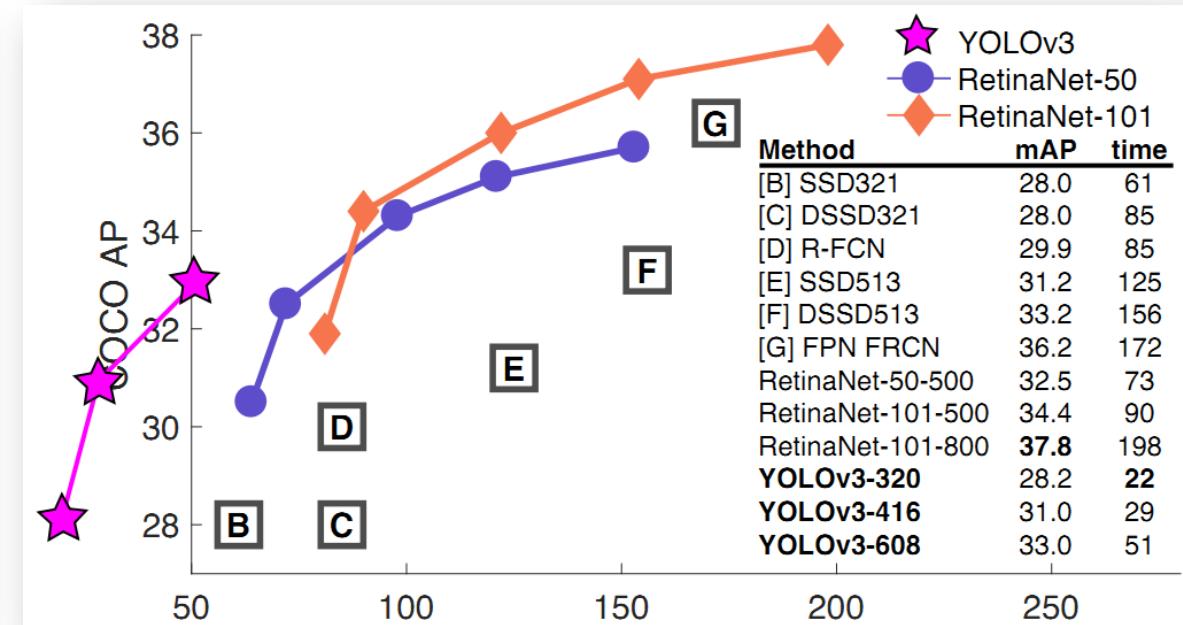
YOLO v3损失函数、性能表现

Loss Function

- YOLO v3为多标签二分类，输出由softmax改成sigmoid
(YOLO各类别互斥，YOLO v3不必互斥)
- Coding中，类别损失实际用的交叉熵



AP在IOU=0.5标准上效果不错



但在IOU=0.5-0.95评估标准上效果一般