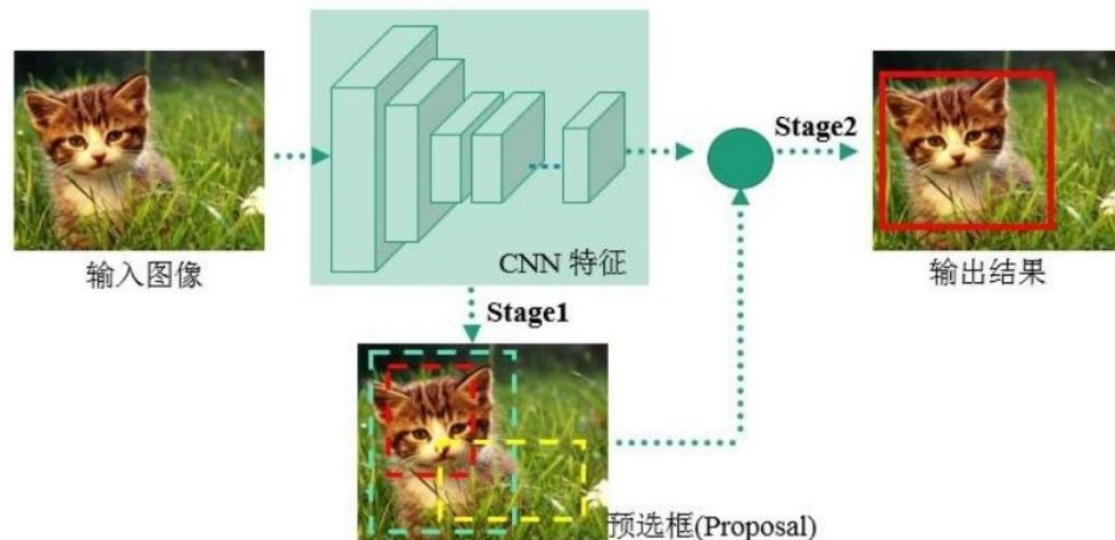
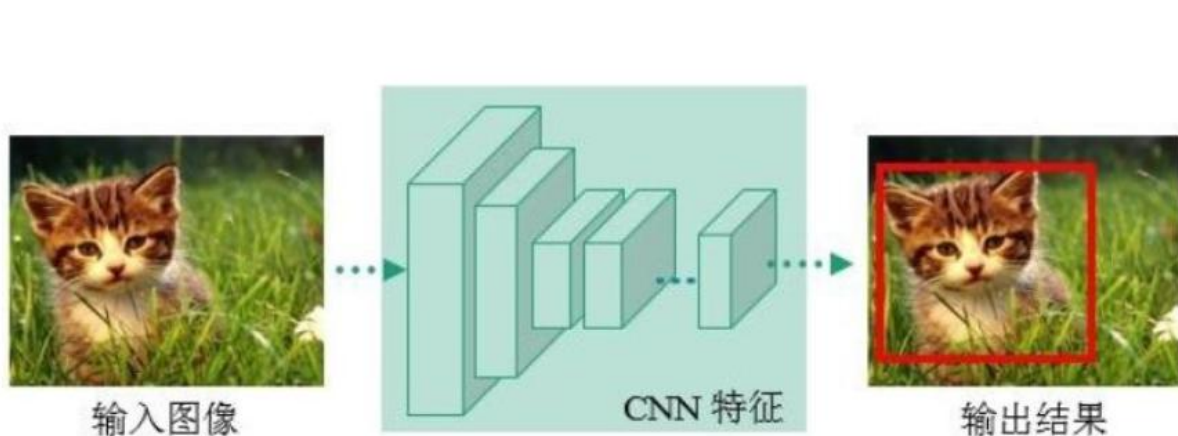


# YOLO系列

✓ 深度学习经典检测方法

✎ two-stage（两阶段）：Faster-rcnn Mask-Rcnn系列

✎ one-stage（单阶段）：YOLO系列



# YOLO系列

✓ one-stage:

✎ 最核心的优势：速度非常快，适合做实时检测任务！

✎ 但是缺点也是有的，效果通常情况下不会太好！

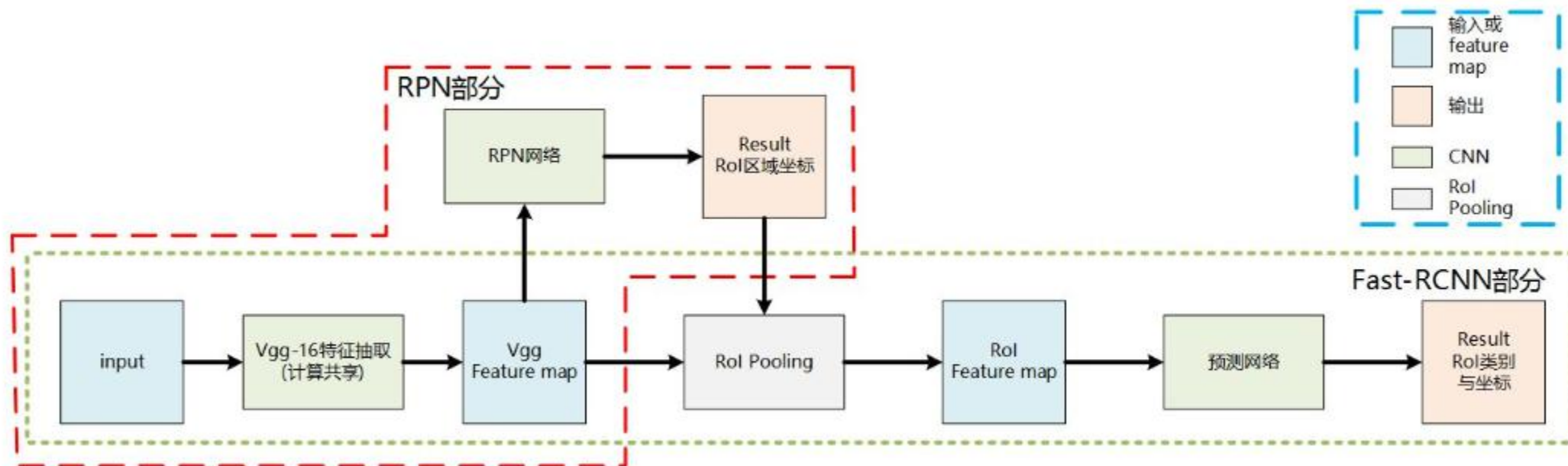
Model	Train	Test	mAP	FLOPS	FPS	Cfg	Weights
SSD300	COCO trainval	test-dev	41.2	-	46		<a href="#">link</a>
SSD500	COCO trainval	test-dev	46.5	-	19		<a href="#">link</a>
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40	<a href="#">cfg</a>	<a href="#">weights</a>
Tiny YOLO	COCO trainval	-	-	7.07 Bn	200	<a href="#">cfg</a>	<a href="#">weights</a>

# YOLO系列

✓ two-stage:

✎ 速度通常较慢（5FPS），但是效果通常还是不错的！

✎ 非常实用的通用框架MaskRcnn，建议熟悉下！



# YOLO系列

## ✓ 指标分析

📎 map指标：综合衡量检测效果；单看精度和recall不行吗？

📎 IOU:



$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$





# YOLO系列

## ✓ 指标分析

✎ 这几个哥们咱得认识：

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

已知条件：班级总人数100人，其中男生80人，女生20人。

目标：找出所有的女生。

结果：从班级中选择了50人，其中20人是女生，还错误的把30名男生挑选出来了。

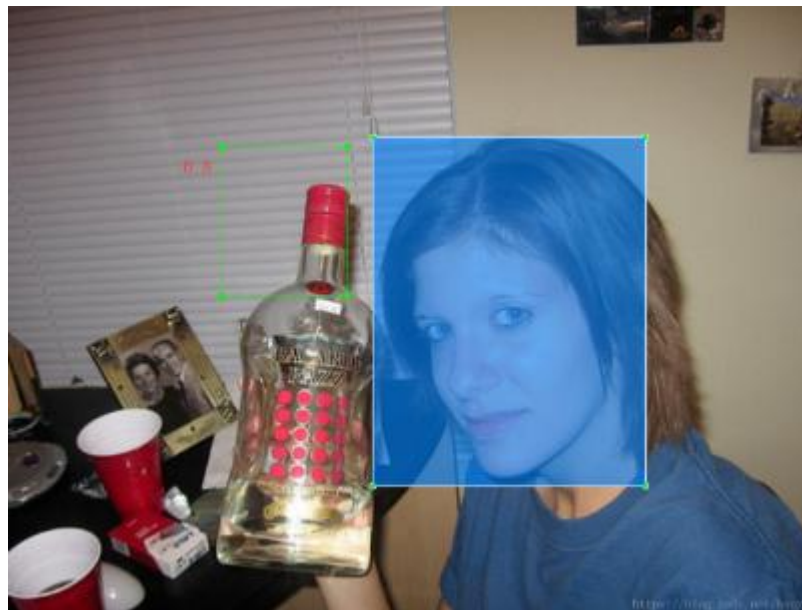
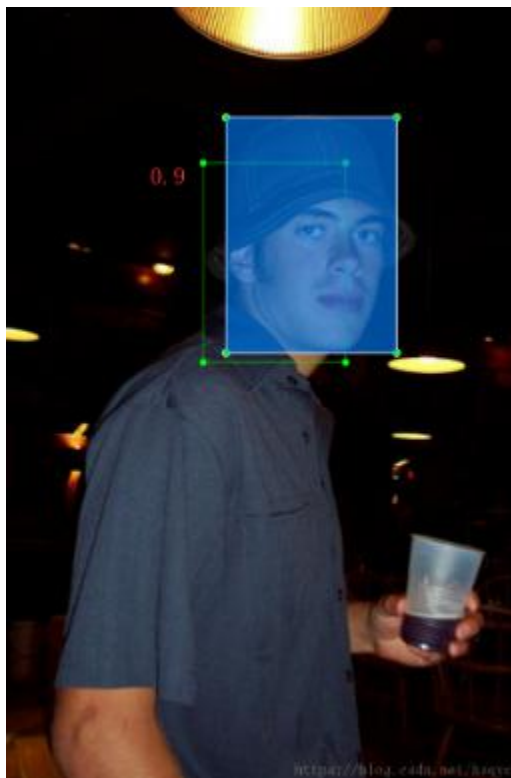
	相关( <b>Relevant</b> ), 正类	无关( <b>NonRelevant</b> ), 负类
被检索到 ( <b>Retrieved</b> )	true positives(TP 正类判定为正类,例子中就是正确的判定"这位是女生")	false positives(FP 负类判定为正类,"存伪",例子中就是分明是男生却判断为女生,当下伪娘横行,这个错常有人犯)
未被检索到 ( <b>Not Retrieved</b> )	false negatives(FN 正类判定为负类,"去真",例子中就是,分明是女生,这哥们却判断为男生--梁山伯同学犯的错就是这个)	true negatives(TN 负类判定为负类,也就是一个男生被判断为男生,像我这样的纯爷们一准儿就会在此处)

TP = 20; FP = 30; FN = 0; TN = 50;

# YOLO系列

## ✓ 指标分析

📎 检测任务中的精度和召回率分别代表什么？

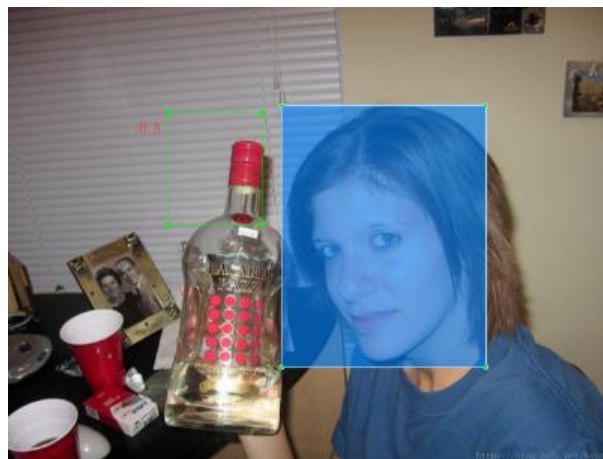
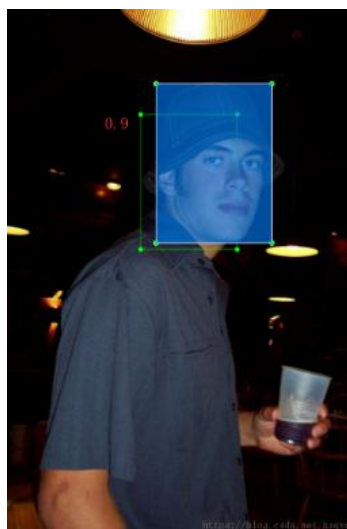


# YOLO系列

## ✓ 指标分析

✎ 基于置信度阈值来计算，例如分别计算0.9； 0.8； 0.7

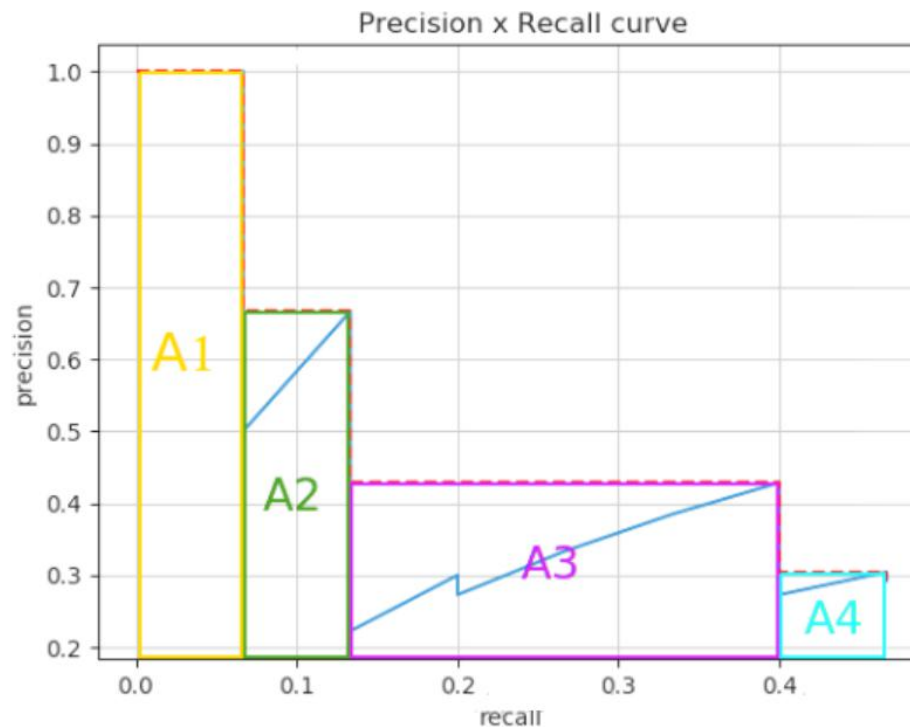
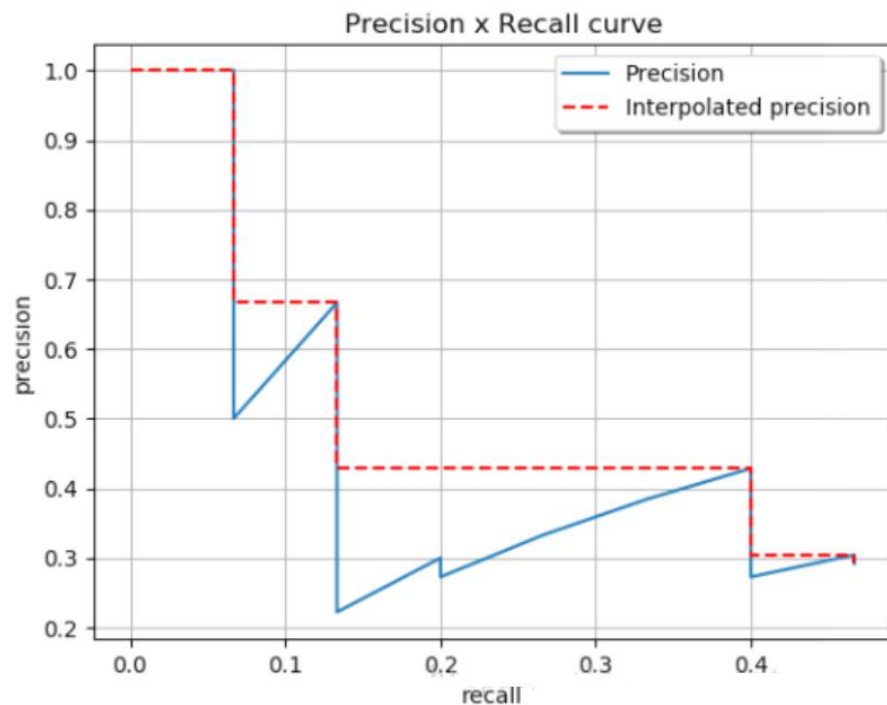
✎ 0.9时：  $TP + FP = 1$  ,  $TP = 1$  ;  $FN = 2$  ;  $Precision = 1/1$  ;  $Recall = 1/3$  ;



# YOLO系列

## ✓ 指标分析

✎ 如何计算AP呢？ 需要把所有阈值都考虑进来； MAP就是所有类别的平均





# YOLO系列

---

## ✓ YOLO-V1

✎ 经典的one-stage方法

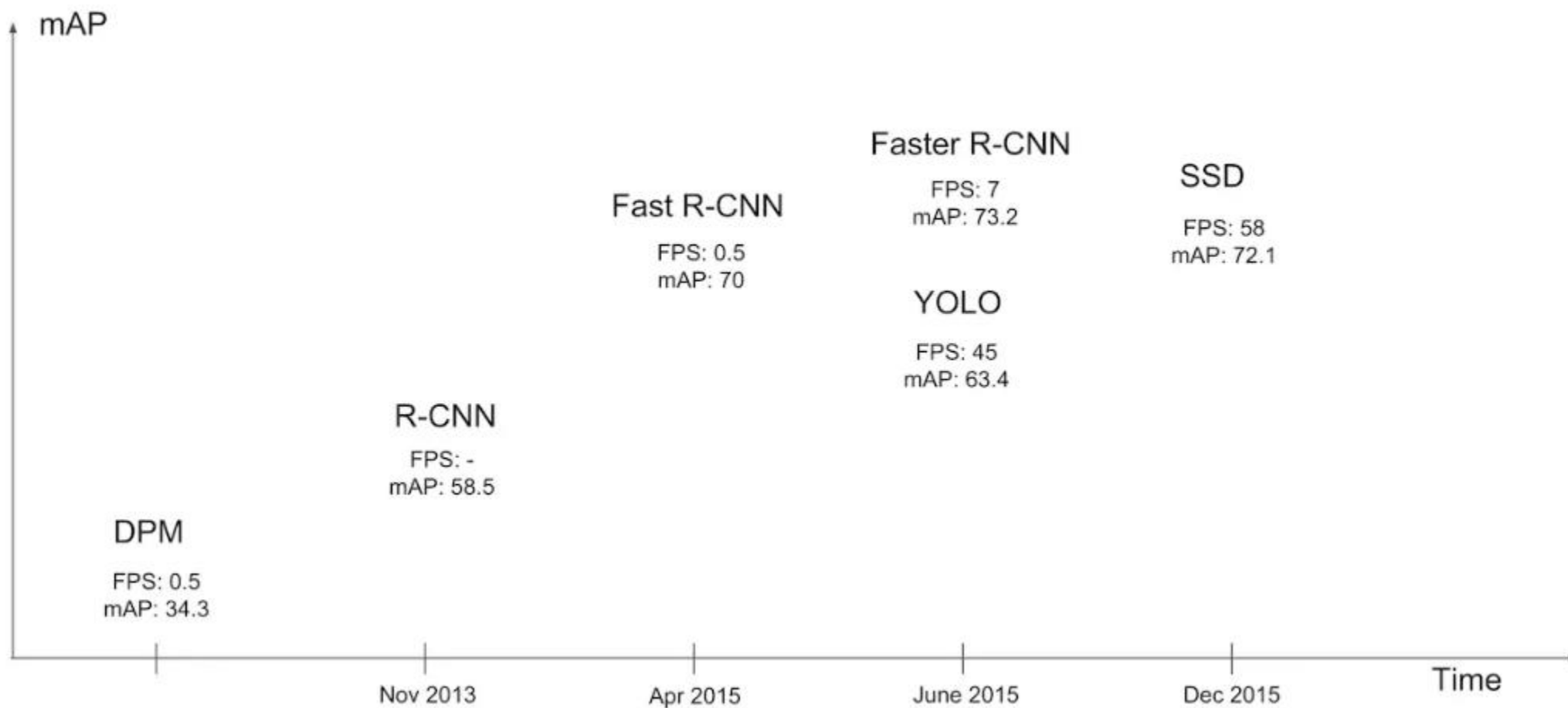
✎ You Only Look Once, 名字就已经说明了一切!

✎ 把检测问题转化成回归问题, 一个CNN就搞定了!

✎ 可以对视频进行实时检测, 应用领域非常广!

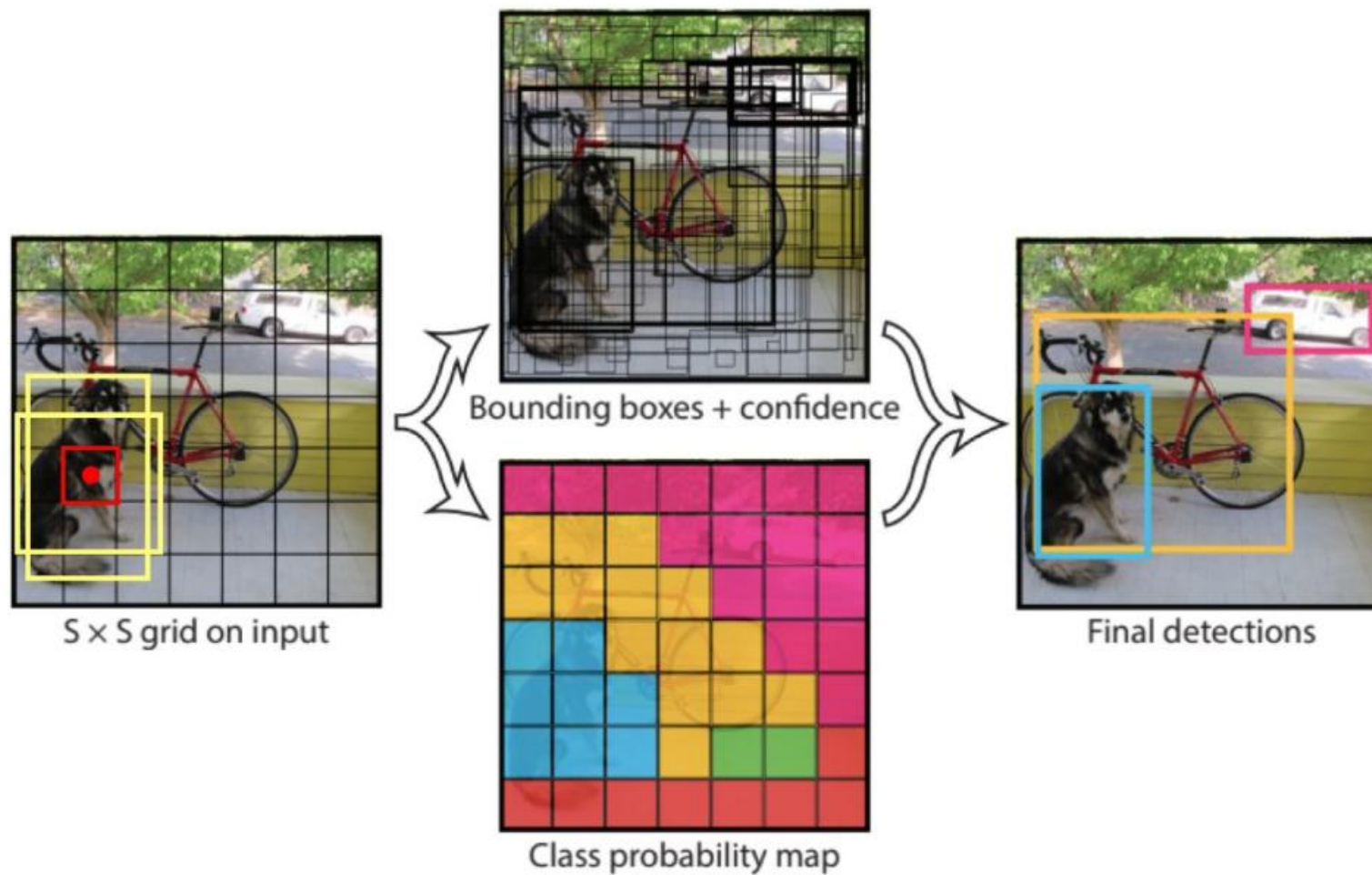
# YOLO系列

## ✓ YOLO-V1



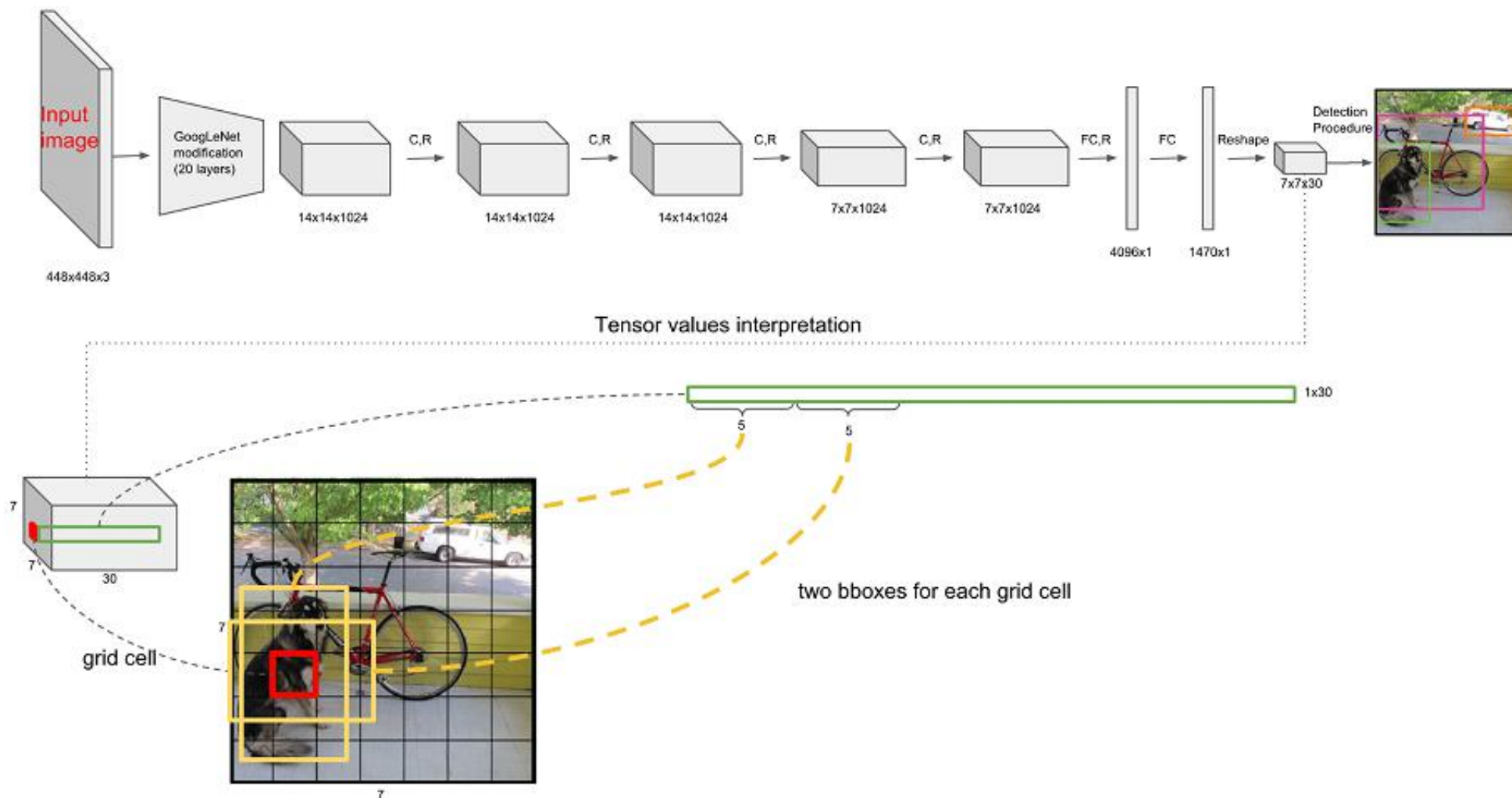
# YOLO系列

✓ 核心思想:



# YOLO系列

## ✓ 网络架构





# YOLO系列

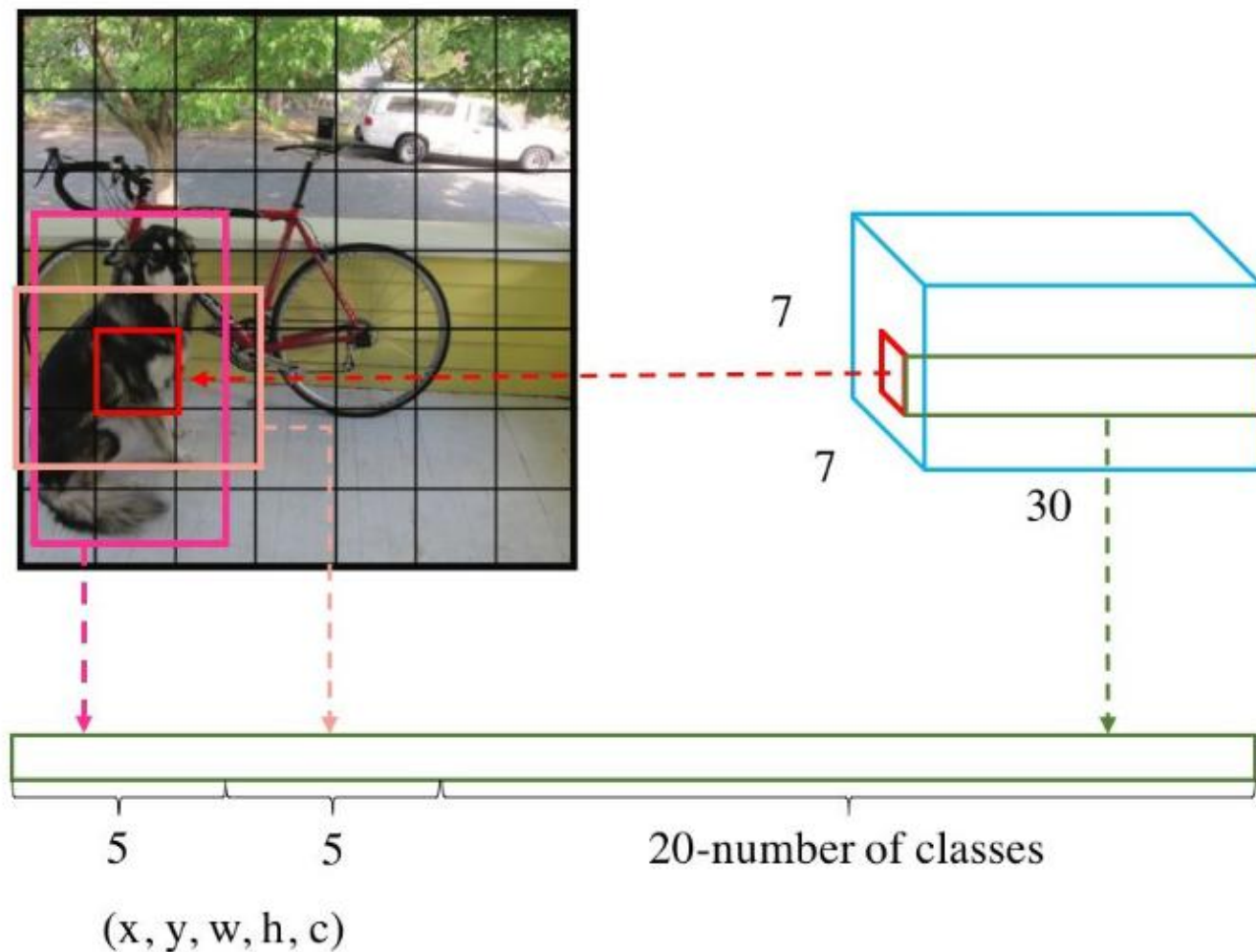
✓ 每个数字的含义:

✎  $10 = (X, Y, H, W, C) * B$  (2个)

✎ 当前数据集中有20个类别

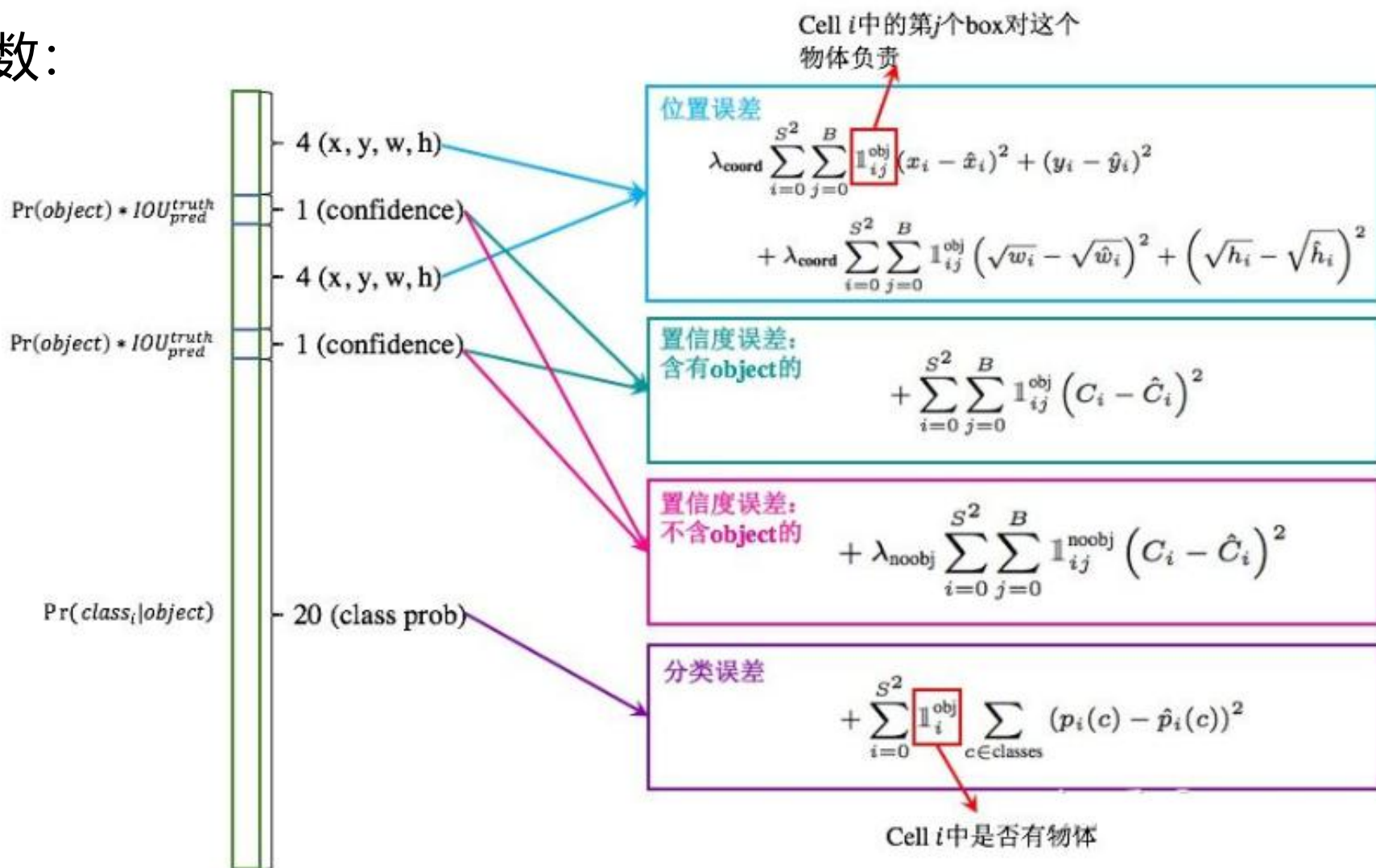
✎  $7*7$ 表示最终网格的大小

✎  $(S*S) * (B*5+C)$



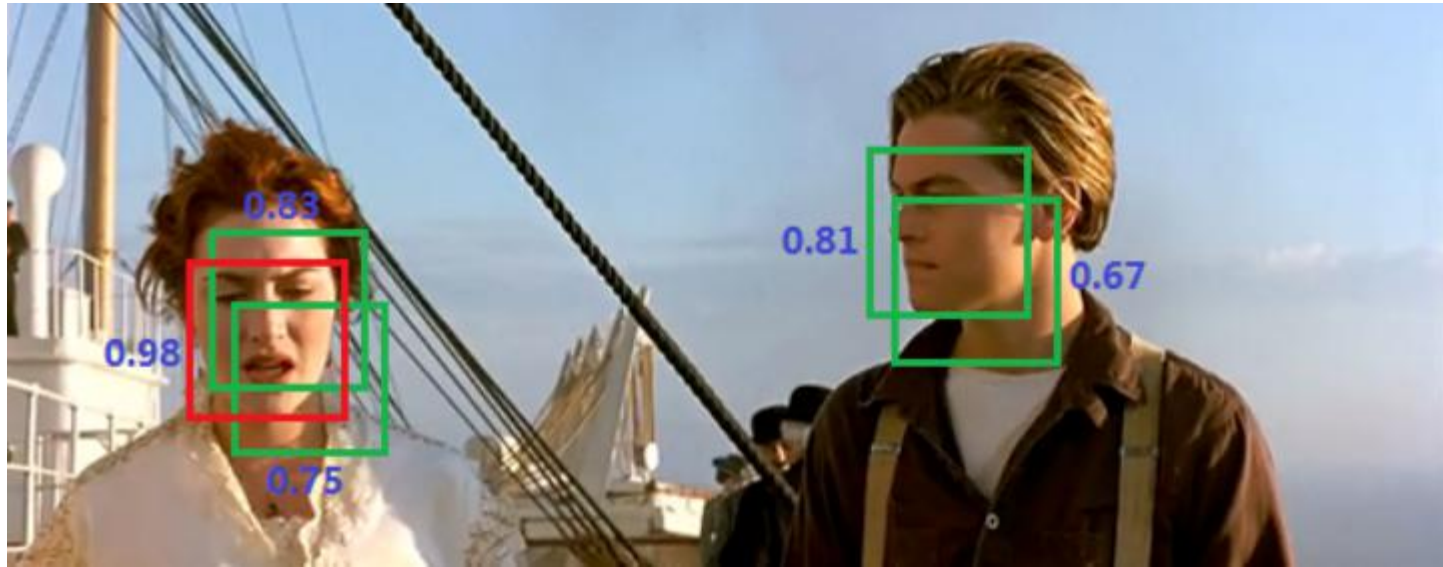
# YOLO系列

✓ 损失函数:



# YOLO系列

✓ NMS(非极大值抑制)



# YOLO系列

---

## ✓ YOLO-V1

✎ 优点：快速，简单！

✎ 问题1：每个Cell只预测一个类别，如果重叠无法解决

✎ 问题2：小物体检测效果一般，长宽比可选的但单一



# YOLO系列

✓ YOLO-V2

✎ 更快! 更强!

	YOLO								YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	<b>78.6</b>

# YOLO系列

---

## ✓ YOLO-V2-Batch Normalization

✎ V2版本舍弃Dropout，卷积后全部加入Batch Normalization

✎ 网络的每一层的输入都做了归一化，收敛相对更容易

✎ 经过Batch Normalization处理后的网络会提升2%的mAP

✎ 从现在的角度来看，Batch Normalization已经成网络必备处理

# YOLO系列

✓ YOLO-V2-更大的分辨率

✎ V1训练时用的是 $224 \times 224$ ，测试时使用 $448 \times 448$

✎ 可能导致模型水土不服，V2训练时额外又进行了10次 $448 \times 448$  的微调

✎ 使用高分辨率分类器后，YOLOv2的mAP提升了约4%



# YOLO系列

## ✓ YOLO-V2-网络结构

✎ DarkNet, 实际输入为 $416 \times 416$

✎ 没有FC层, 5次降采样, ( $13 \times 13$ )

✎  $1 \times 1$ 卷积节省了很多参数

Type	Filters	Size/Stride	Output
Convolutional	32	$3 \times 3$	$224 \times 224$
Maxpool		$2 \times 2/2$	$112 \times 112$
Convolutional	64	$3 \times 3$	$112 \times 112$
Maxpool		$2 \times 2/2$	$56 \times 56$
Convolutional	128	$3 \times 3$	$56 \times 56$
Convolutional	64	$1 \times 1$	$56 \times 56$
Convolutional	128	$3 \times 3$	$56 \times 56$
Maxpool		$2 \times 2/2$	$28 \times 28$
Convolutional	256	$3 \times 3$	$28 \times 28$
Convolutional	128	$1 \times 1$	$28 \times 28$
Convolutional	256	$3 \times 3$	$28 \times 28$
Maxpool		$2 \times 2/2$	$14 \times 14$
Convolutional	512	$3 \times 3$	$14 \times 14$
Convolutional	256	$1 \times 1$	$14 \times 14$
Convolutional	512	$3 \times 3$	$14 \times 14$
Convolutional	256	$1 \times 1$	$14 \times 14$
Convolutional	512	$3 \times 3$	$14 \times 14$
Maxpool		$2 \times 2/2$	$7 \times 7$
Convolutional	1024	$3 \times 3$	$7 \times 7$
Convolutional	512	$1 \times 1$	$7 \times 7$
Convolutional	1024	$3 \times 3$	$7 \times 7$
Convolutional	512	$1 \times 1$	$7 \times 7$
Convolutional	1024	$3 \times 3$	$7 \times 7$
Convolutional	1000	$1 \times 1$	$7 \times 7$
Avgpool		Global	1000
Softmax			

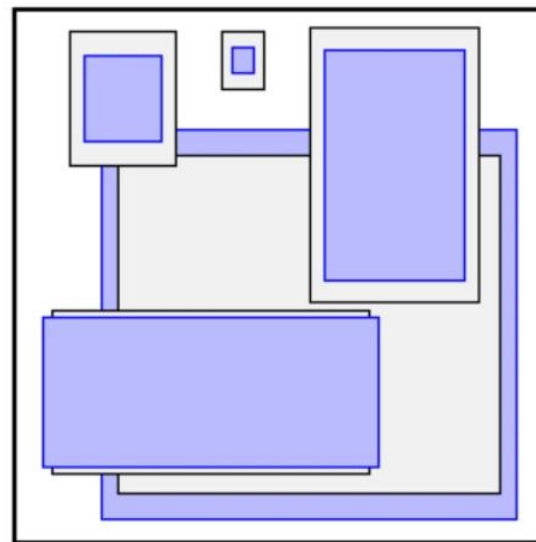
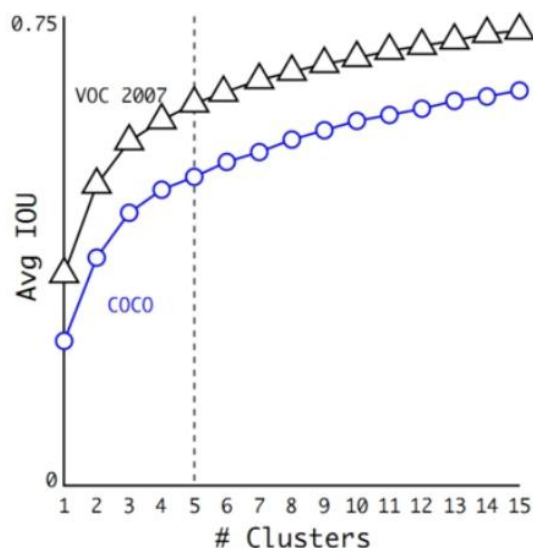


# YOLO系列

✓ YOLO-V2-聚类提取先验框

✎ faster-rcnn系列选择的先验比例都是常规的，但是不一定完全适合数据集

✎ K-means聚类中的距离： $d(box, centroids) = 1 - IOU(box, centroids)$



# YOLO系列

## ✓ YOLO-V2-Anchor Box

✎ 通过引入anchor boxes, 使得预测的box数量更多 ( $13*13*n$ )

✎ 跟faster-rcnn系列不同的是先验框并不是直接按照长宽固定比给定

without anchor	69.5 mAP	81% recall
with anchor	69.2 mAP	88% recall

# YOLO系列

## ✓ YOLO-V2-Directed Location Prediction

✎ bbox: 中心为 $(x_p, y_p)$ ; 宽和高为 $(w_p, h_p)$ , 则:

$$\begin{aligned}x &= x_p + w_p * tx \\ y &= y_p + h_p * ty\end{aligned}$$

✎  $tx=1$ ,则将bbox在x轴向右移动 $w_p$ ;  $tx=-1$ 则将其向左移动 $w_p$

✎ 这样会导致收敛问题, 模型不稳定, 尤其是刚开始进行训练的时候

✎ V2中并没有直接使用偏移量, 而是选择相对grid cell的偏移量

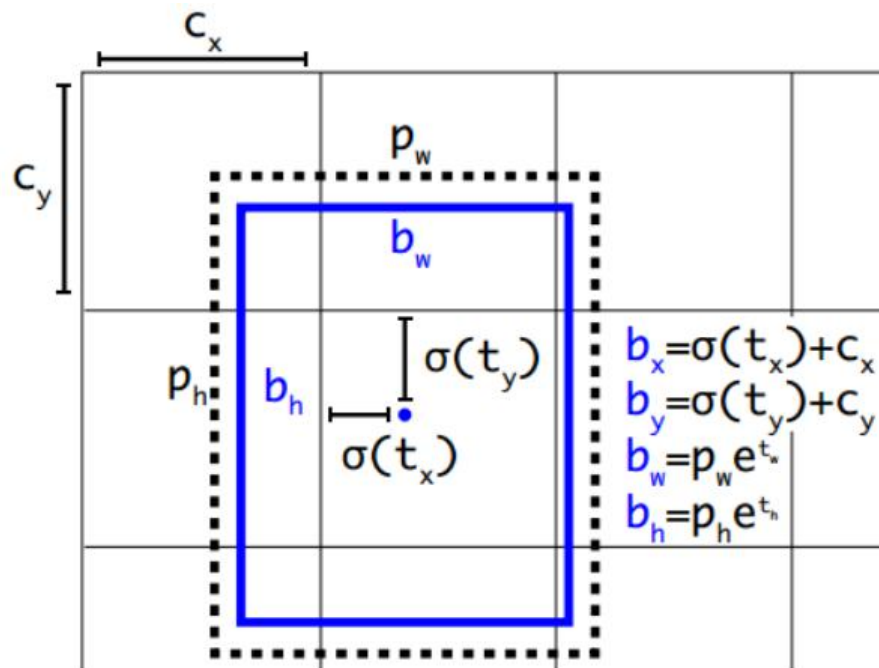
## ✓ YOLO-V2-Directed Location Prediction


$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

$$b_w = p_w e^{t_w}$$

$$b_h = p_h e^{t_h}$$



✎ 例如预测值 $(\sigma_x, \sigma_y, t_w, t_h) = (0.2, 0.1, 0.2, 0.32)$ , anchor框为 $p_w = 3.19275, p_h = 4.00944$

在特征图位置：

$$b_x = 0.2 + 1 = 1.2$$

$$b_y = 0.1 + 1 = 1.1$$

$$b_w = 3.19275 * e^{0.2} = 3.89963$$

$$b_h = 4.00944 * e^{0.32} = 5.52151$$

在原位置：

$$b_x = 1.2 * 32 = 38.4$$

$$b_y = 1.1 * 32 = 35.2$$

$$b_w = 3.89963 * 32 = 124.78$$

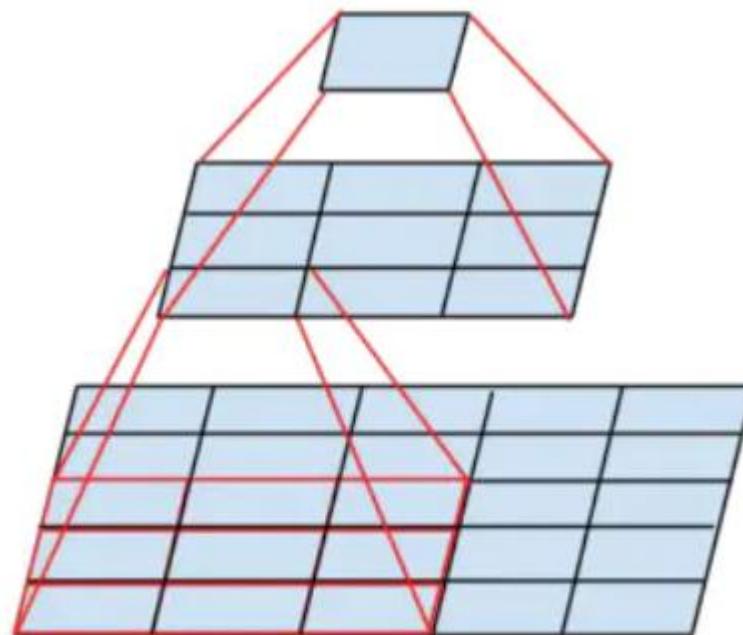
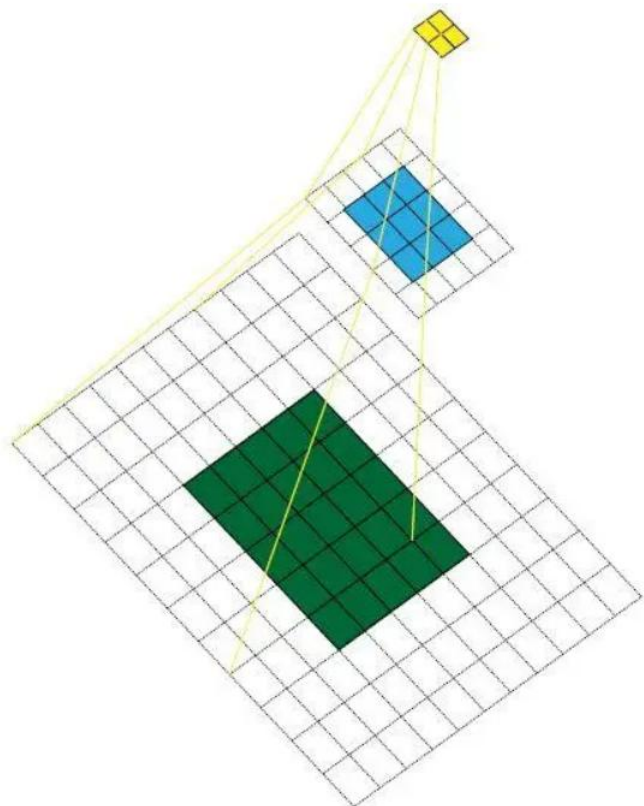
$$b_h = 5.52151 * 32 = 176.68$$



# YOLO系列

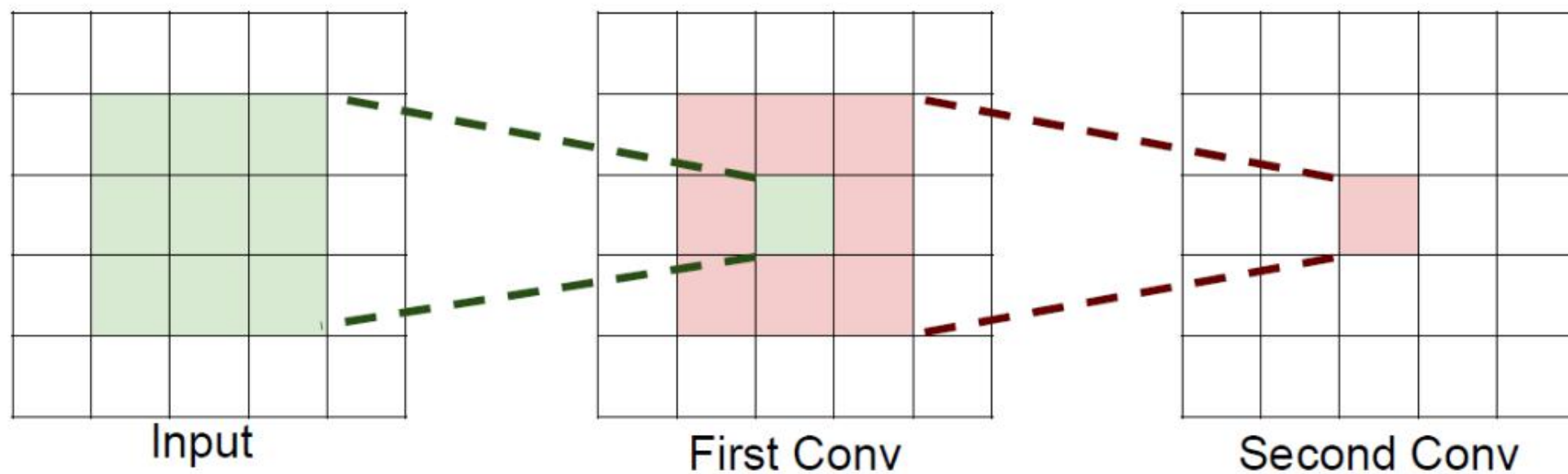
✓ 感受野

✎ 概述来说就是特征图上的点能看到原始图像多大区域



# YOLO系列

✓ 感受野:



✎ 如果堆叠3个 $3 \times 3$ 的卷积层，并且保持滑动窗口步长为1，其感受野就是 $7 \times 7$ 的了，这跟一个使用 $7 \times 7$ 卷积核的结果是一样的，那为什么非要堆叠3个小卷积呢？

# YOLO系列

## ✓ 感受野

✎ 假设输入大小都是 $h*w*c$ ，并且都使用 $c$ 个卷积核(得到 $c$ 个特征图)，可以来计算一下其各自所需参数：

一个 $7*7$ 卷积核所需参数：

$$= C \times (7 \times 7 \times C) = \mathbf{49 C^2}$$

3个 $3*3$ 卷积核所需参数：

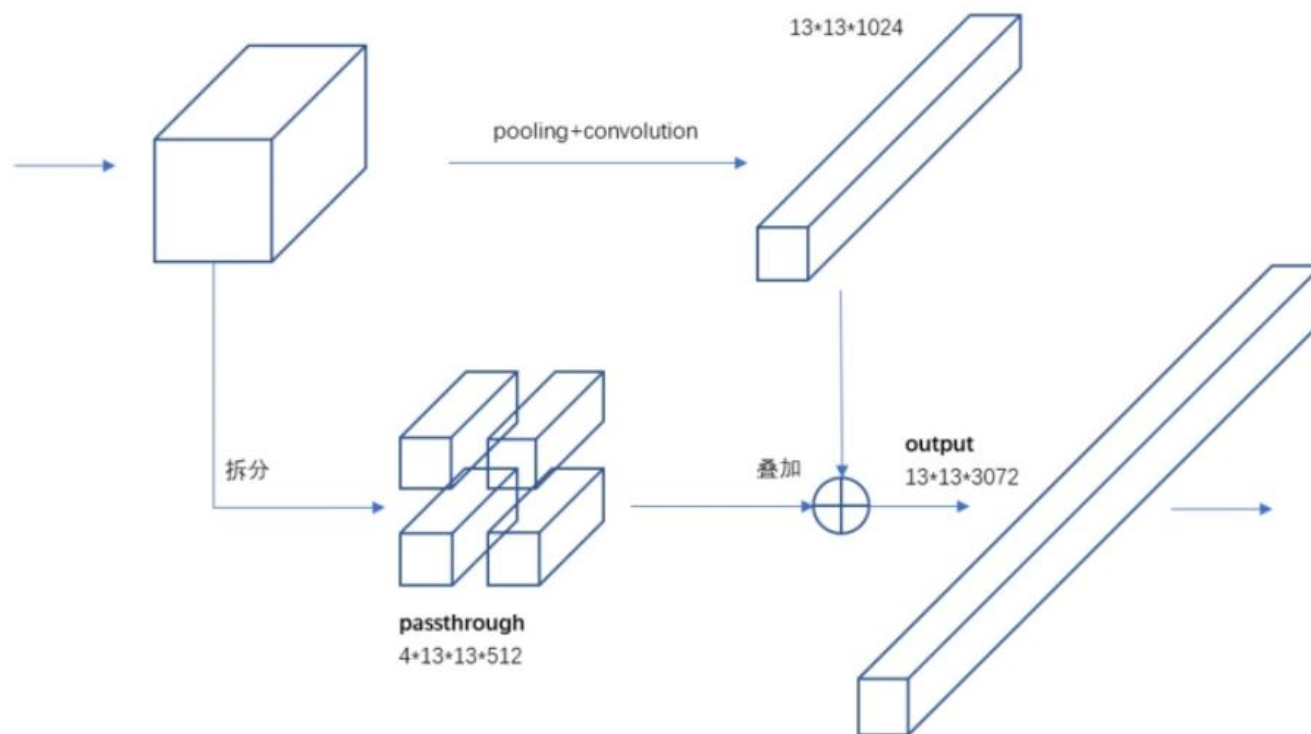
$$= 3 \times C \times (3 \times 3 \times C) = \mathbf{27 C^2}$$

✎ 很明显，堆叠小的卷积核所需的参数更少一些，并且卷积过程越多，特征提取也会越细致，加入的非线性变换也随着增多，还不会增大权重参数个数，这就是VGG网络的基本出发点，用小的卷积核来完成体特征提取操作。

# YOLO系列

## ✓ YOLO-V2-Fine-Grained Features

✎ 最后一层时感受野太大了，小目标可能丢失了，需融合之前的特征



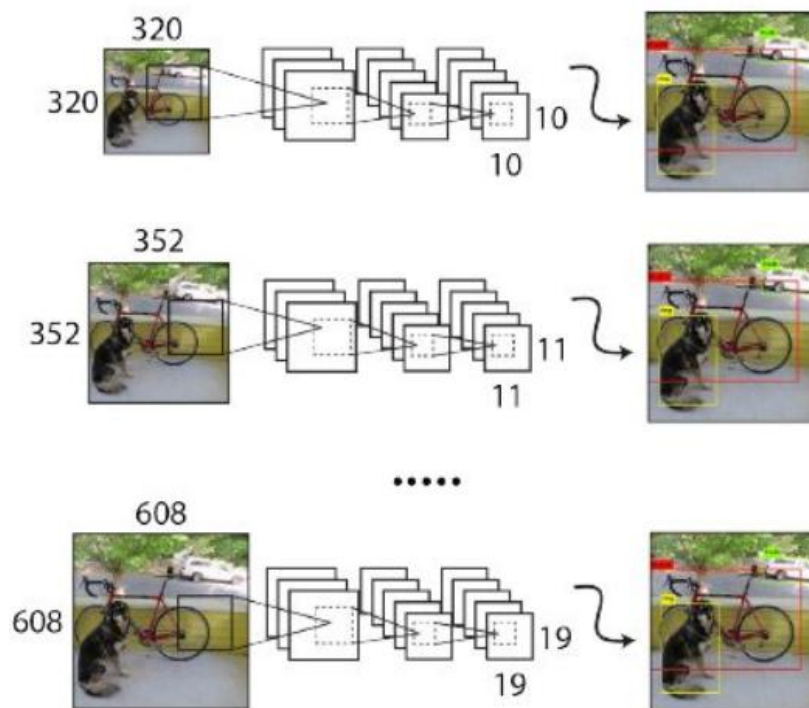
# YOLO系列

## ✓ YOLO-V2-Multi-Scale

📎 都是卷积操作可没人能限制我了！一定iterations之后改变输入图片大小

最小的图像尺寸为320 x 320

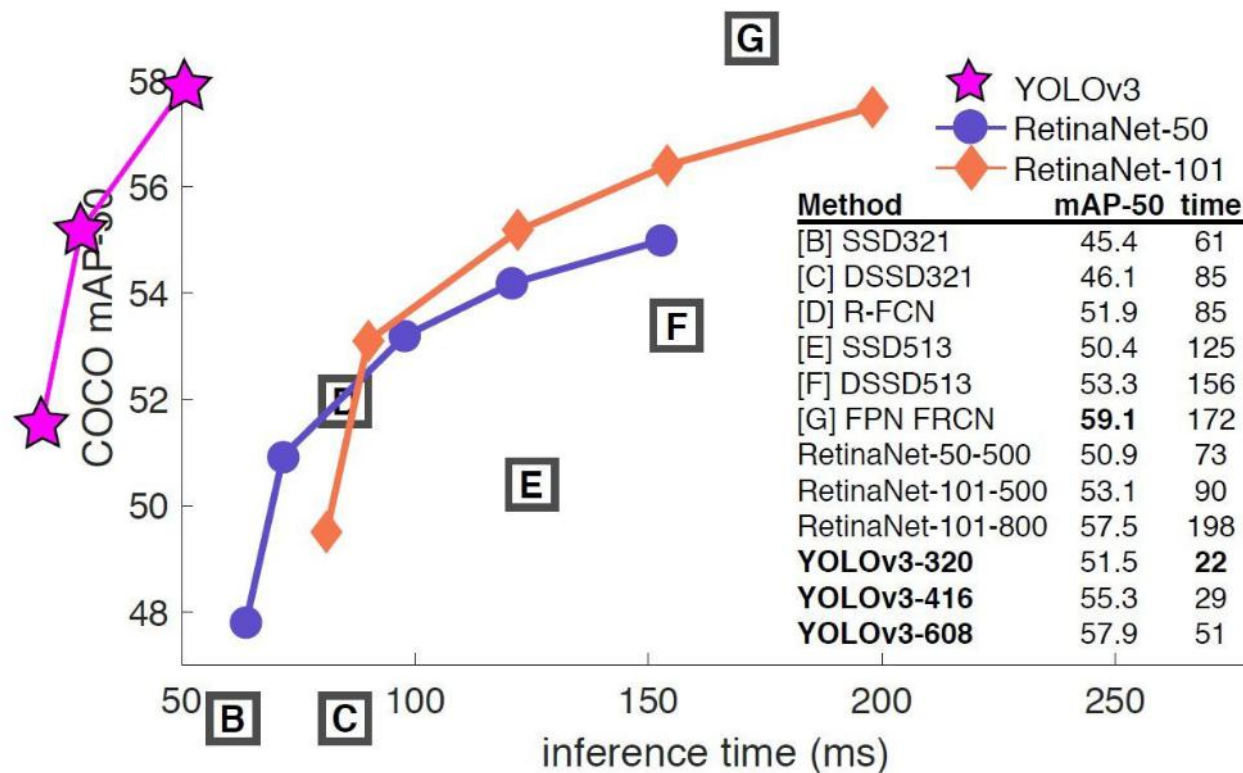
最大的图像尺寸为608 x 608



# YOLO系列

## ✓ YOLO-V3

📎 这张图讲道理真的过分了!!! 我不是针对谁, 在座的各位都是、、、





# YOLO系列

---

## ✓ YOLO-V3

✎ 终于到V3了，最大的改进就是网络结构，使其更适合小目标检测

✎ 特征做的更细致，融入多持续特征图信息来预测不同规格物体

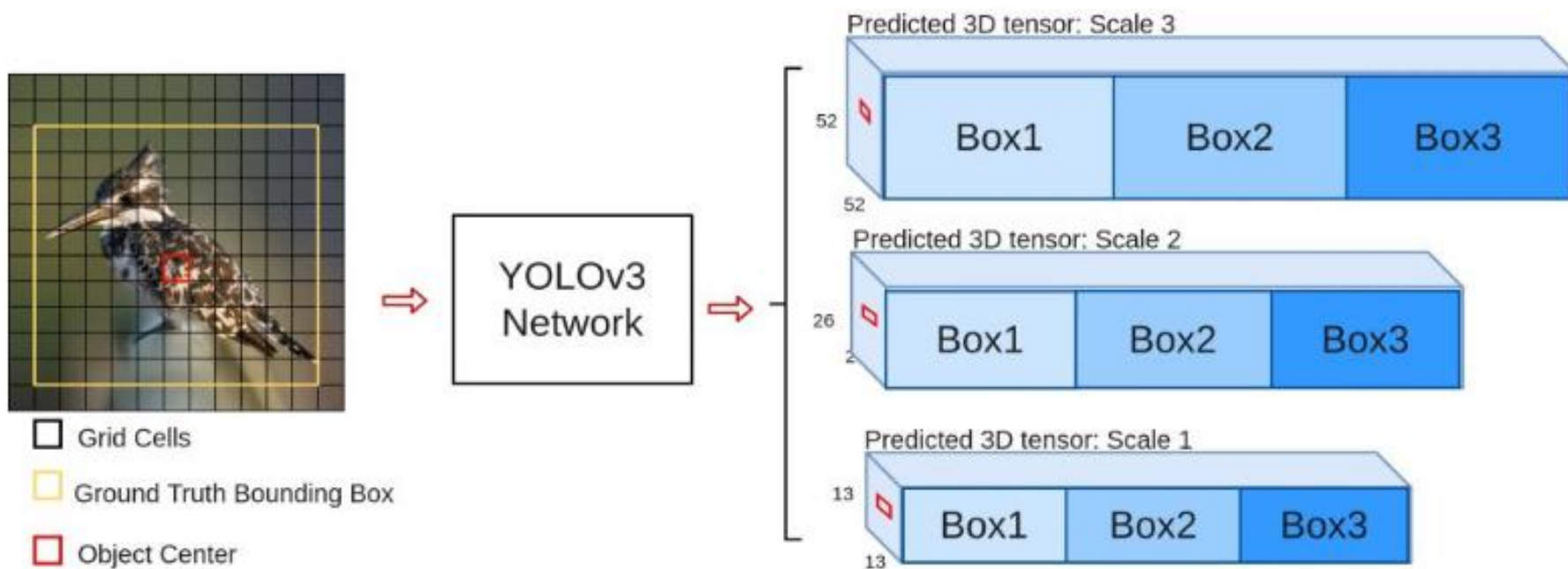
✎ 先验框更丰富了，3种scale，每种3个规格，一共9种

✎ softmax改进，预测多标签任务

# YOLO系列

✓ 多scale

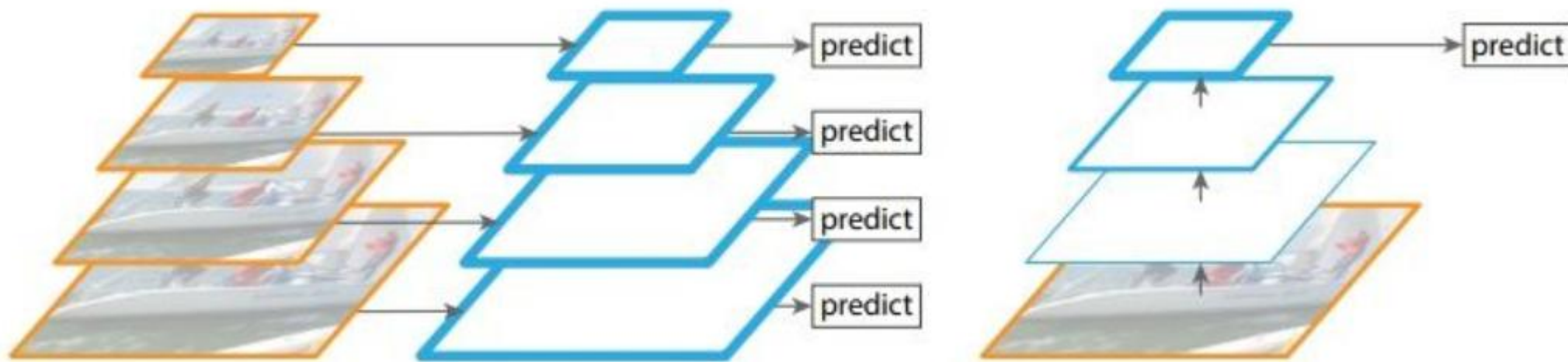
✎ 为了能检测到不同大小的物体，设计了3个scale



# YOLO系列

✓ scale变换经典方法

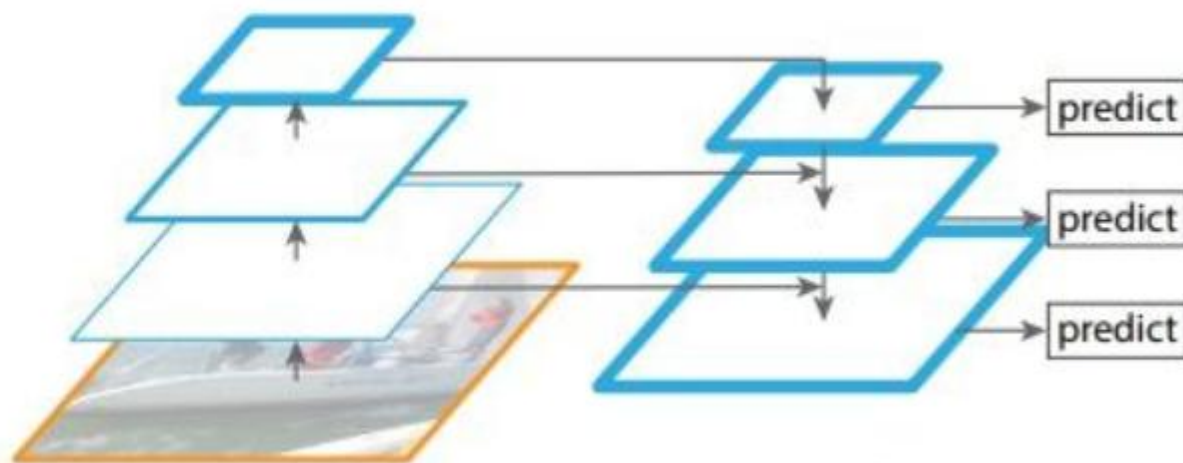
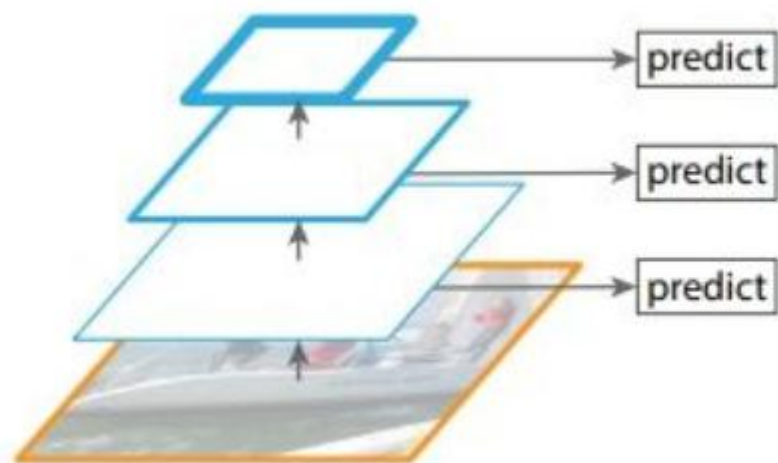
✎ 左图：图像金字塔；右图：单一的输入；



# YOLO系列

✓ scale变换经典方法

✎ 左图：对不同的特征图分别利用；右图：不同的特征图融合后进行预测；

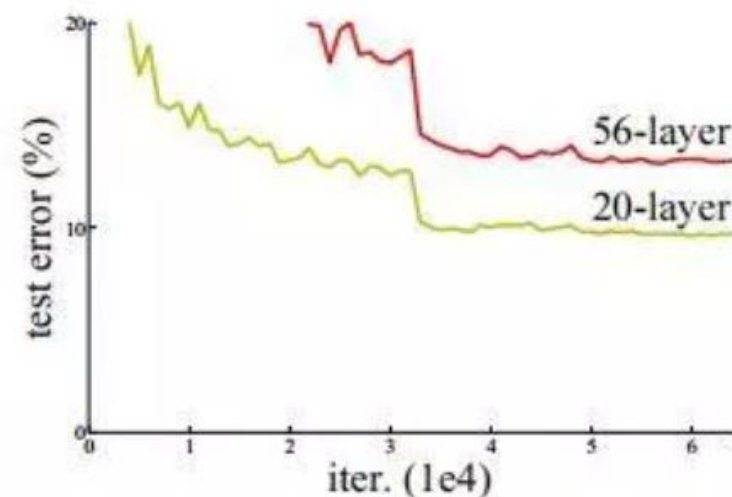
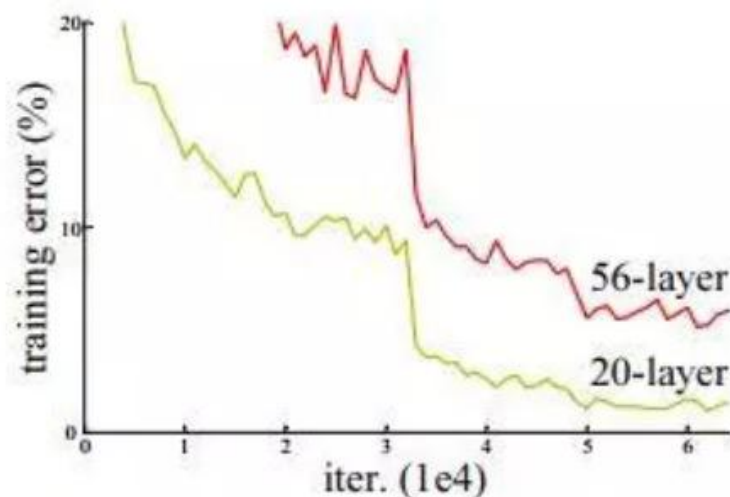
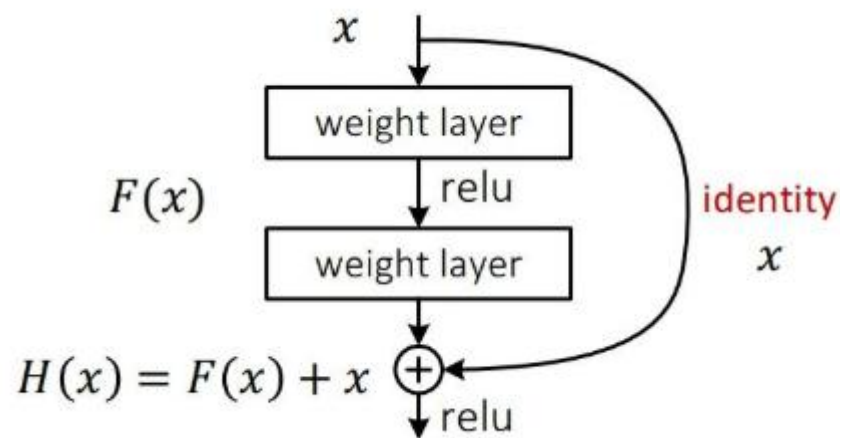


# YOLO系列

✓ 残差连接-为了更好的特征

✎ 从今天的角度来看，基本所有网络架构都用上了残差连接的方法

✎ V3中也用了resnet的思想，堆叠更多的层来进行特征提取



# YOLO系列

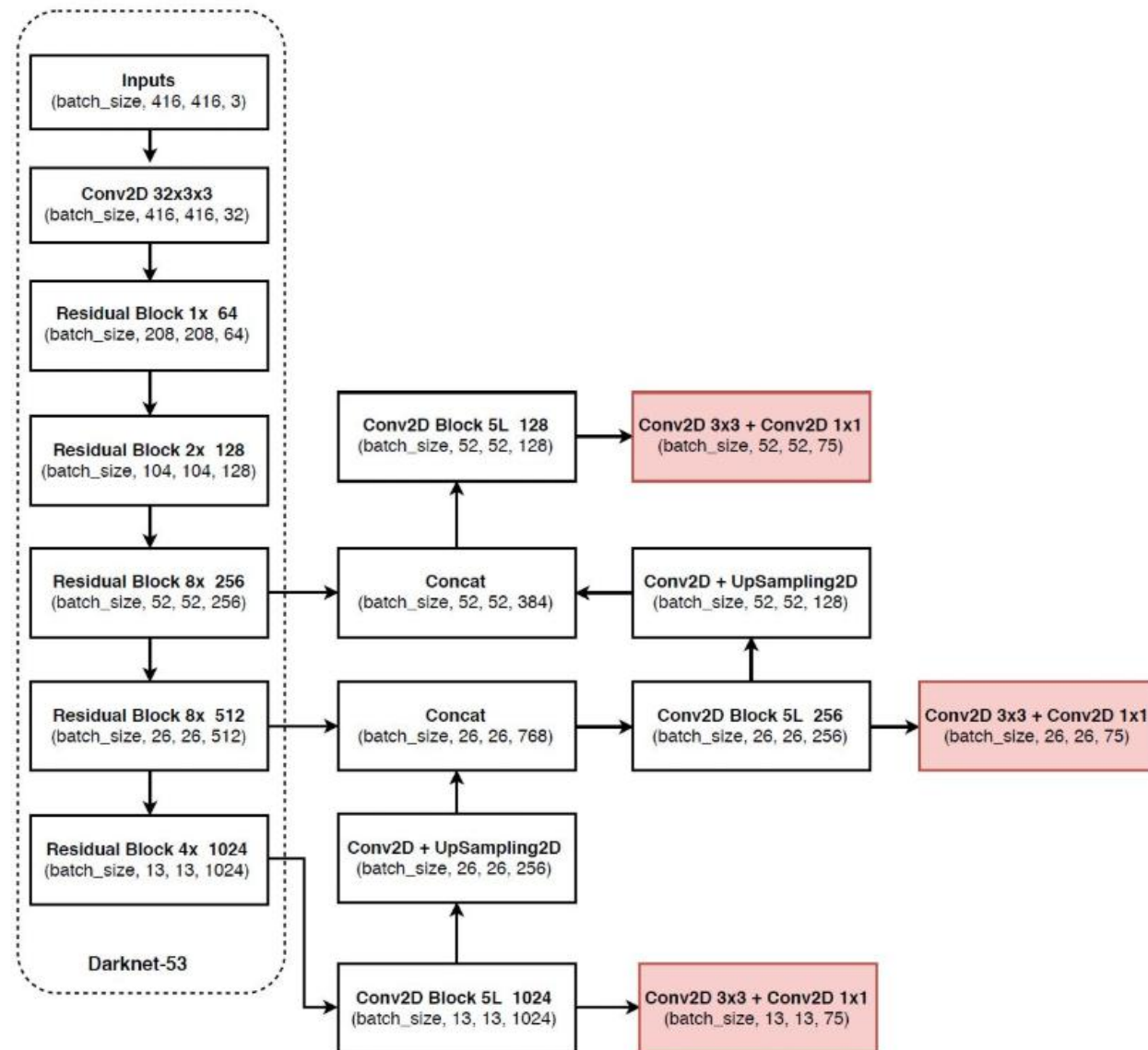
## ✓ 核心网络架构

✎ 没有池化和全连接层，全部卷积

✎ 下采样通过stride为2实现

✎ 3种scale，更多先验框

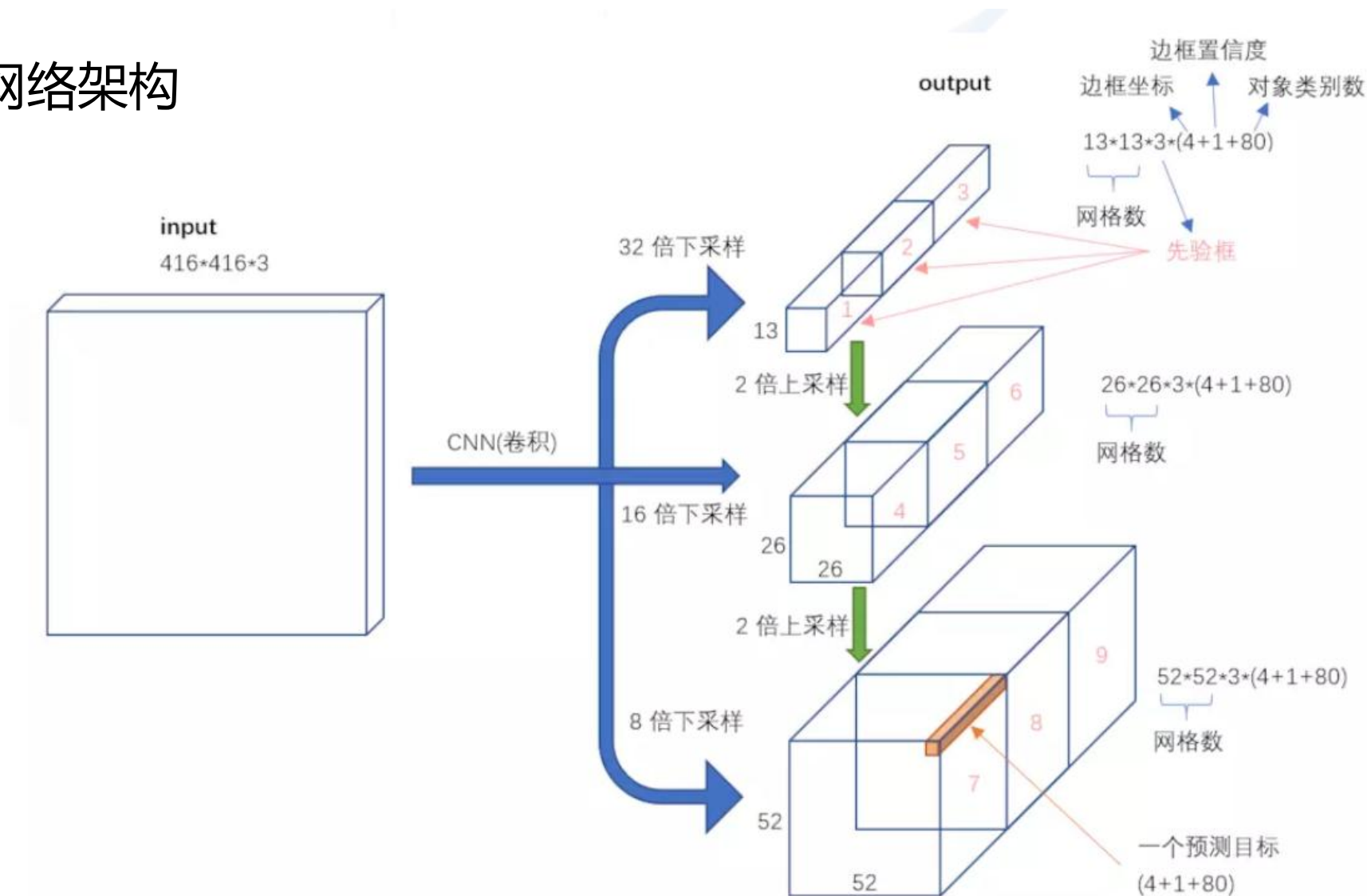
✎ 基本上当下经典做法全融入了





# YOLO系列

## ✓ 核心网络架构



# YOLO系列

## ✓ 先验框设计

✎ YOLO-V2中选了5个，这回更多了，一共有9种

✎ 13\*13特征图上：(116x90), (156x198), (373x326)

26\*26特征图上：(30x61), (62x45), (59x119)

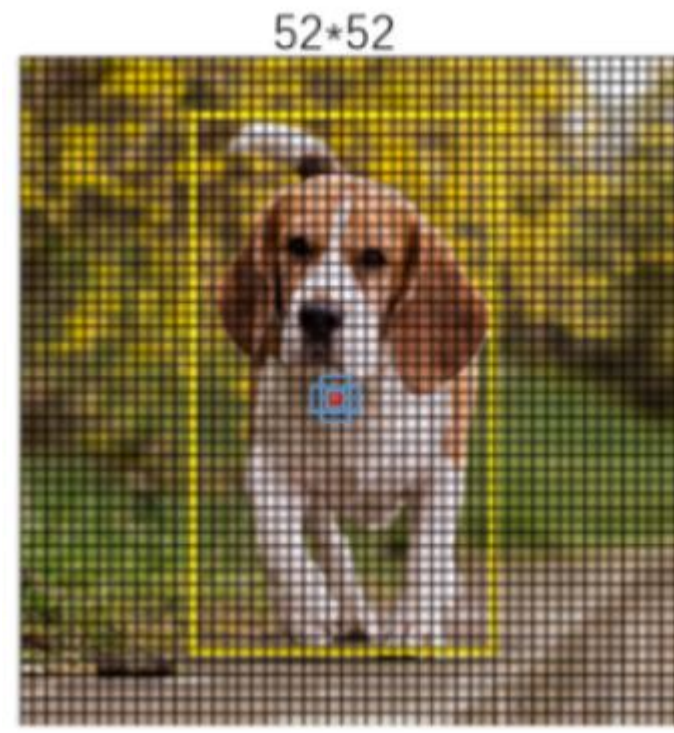
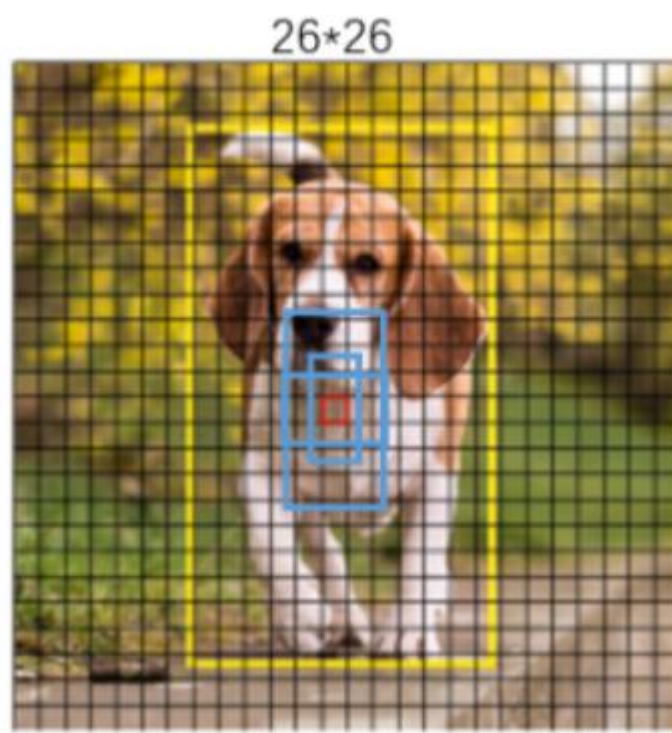
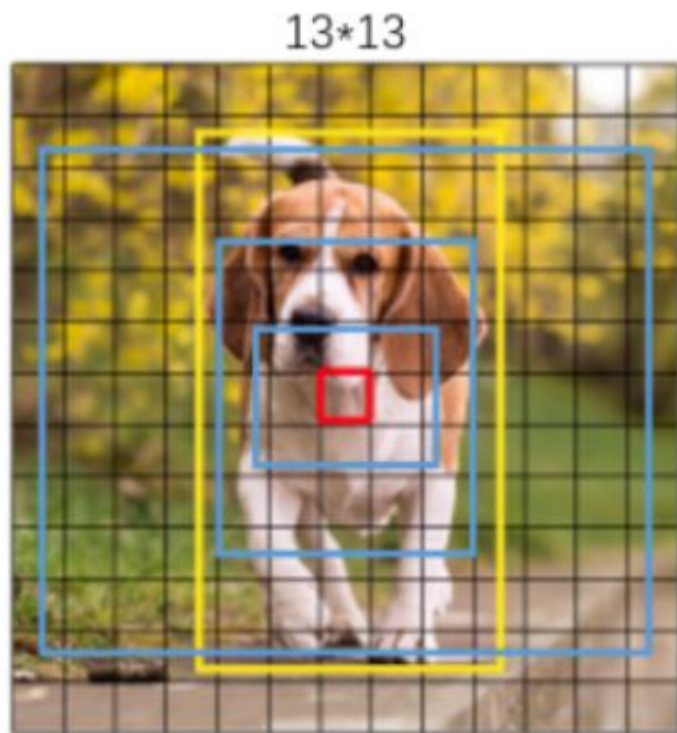
52\*52特征图上：(10x13), (16x30), (33x23)

特征图	13*13			26*26			52*52		
感受野	大			中			小		
先验框	(116x90)	(156x198)	(373x326)	(30x61)	(62x45)	(59x119)	(10x13)	(16x30)	(33x23)

# YOLO系列

✓ 先验框设计

📎 YOLO-V2中选了5个，这回更多了，一共有9种

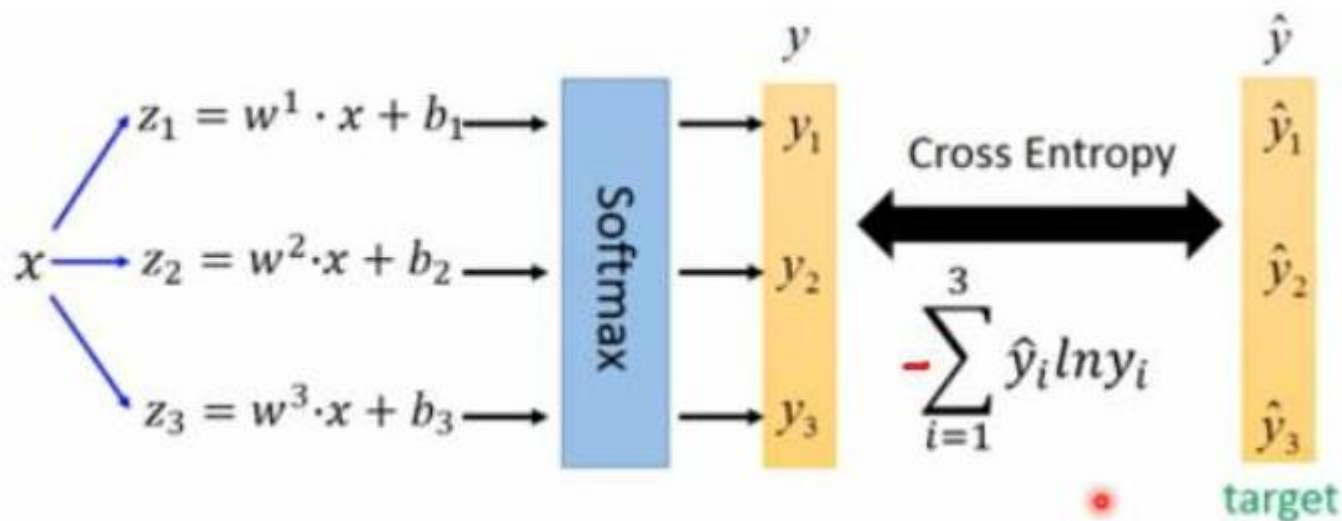


# YOLO系列

✓ softmax层替代

📎 物体检测任务中可能一个物体有多个标签

📎 logistic激活函数来完成，这样就能预测每一个类别是/不是



# YOLO系列-V4

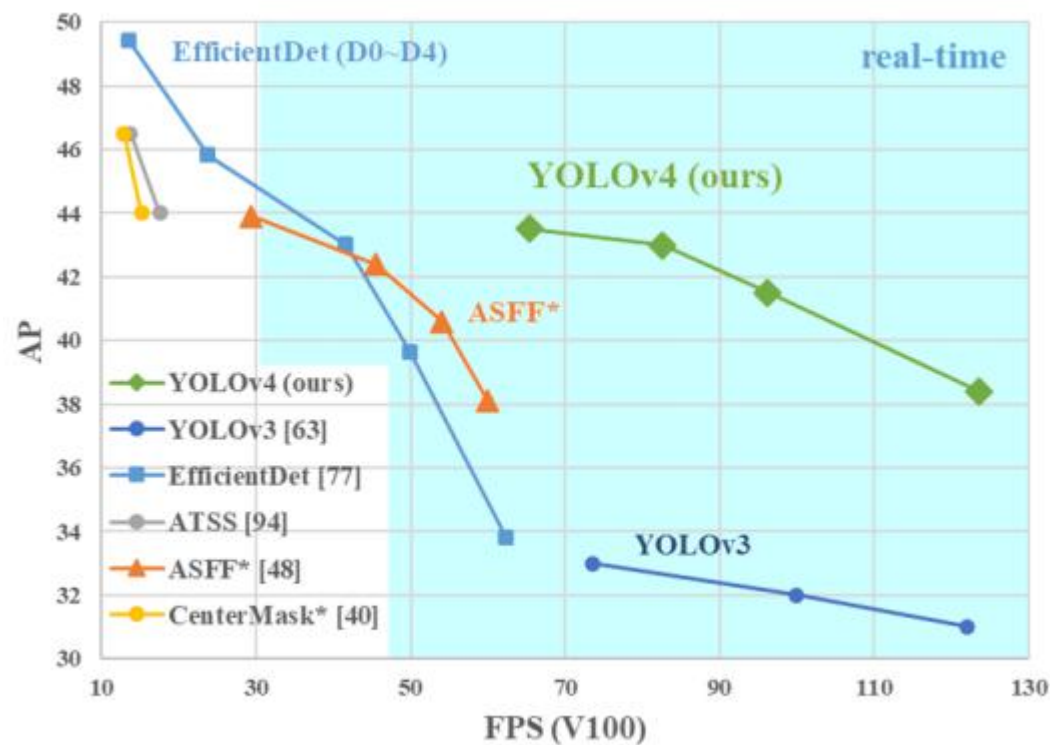
✓ 整体介绍 (Optimal Speed and Accuracy of Object Detection)

✍ 虽然作者换了，但精髓没变！

✍ 如果CV界有劳模奖，一定非他莫属！

✍ 整体看还是那个味，细还是他细！

✍ 江湖传闻最高的武功：嫁衣神功





# YOLO系列-V4

---

✓ V4贡献:

- ✎ 亲民政策，单GPU就能训练的非常好，接下来很多小模块都是这个出发点
- ✎ 两大核心方法，从数据层面和网络设计层面来进行改善
- ✎ 消融实验，感觉能做的都让他给做了，这工作量不轻
- ✎ 全部实验都是单GPU完成，不用太担心设备了



# YOLO系列-V4

---

## ✓ Bag of freebies(BOF)

✎ 只增加训练成本，但是能显著提高精度，并不影响推理速度

✎ 数据增强：调整亮度、对比度、色调、随机缩放、剪切、翻转、旋转





✎ 网络正则化的方法：Dropout、Dropblock等

✎ 类别不平衡，损失函数设计

# YOLO系列-V4

## ✓ Mosaic data augmentation

📎 方法很简单，参考CutMix然后四张图像拼接成一张进行训练

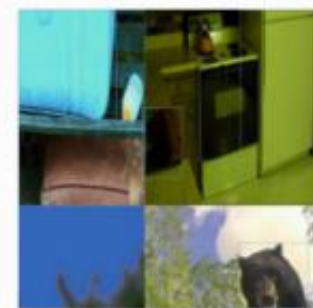
Image	ResNet-50	Mixup [48]	Cutout [3]	CutMix
				
Label	Dog 1.0	Dog 0.5 Cat 0.5	Dog 1.0	Dog 0.6 Cat 0.4
ImageNet Cls (%)	76.3 (+0.0)	77.4 (+1.1)	77.1 (+0.8)	<b>78.6</b> (+2.3)
ImageNet Loc (%)	46.3 (+0.0)	45.8 (-0.5)	46.7 (+0.4)	<b>47.3</b> (+1.0)
Pascal VOC Det (mAP)	75.6 (+0.0)	73.9 (-1.7)	75.1 (-0.5)	<b>76.7</b> (+1.1)



aug\_-319215602\_0\_-238783579.jpg



aug\_-1271888501\_0\_-749611674.jpg



aug\_1462167959\_0\_-1659206634.jpg



aug\_1474493600\_0\_-45389312.jpg



aug\_1715045541\_0\_603913529.jpg



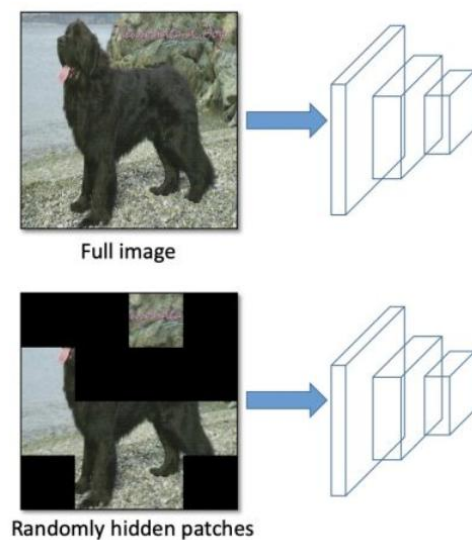
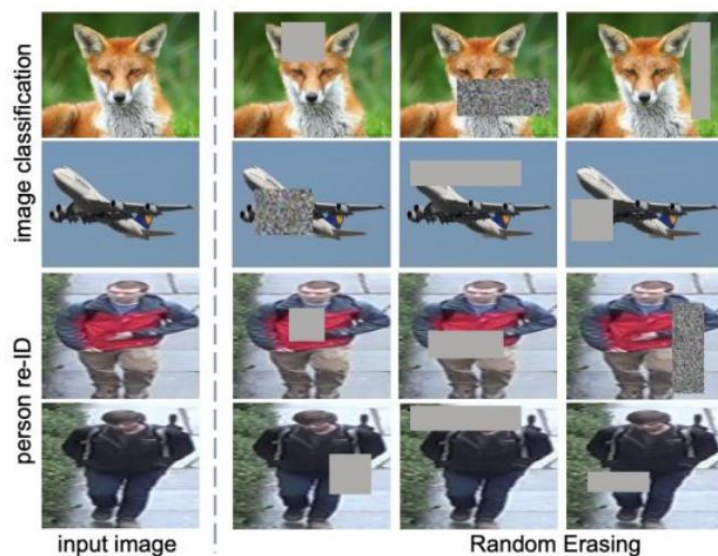
aug\_1779424844\_0\_-589696888.jpg

# YOLO系列-V4

## ✓ 数据增强

✎ Random Erase: 用随机值或训练集的平均像素值替换图像的区域

✎ Hide and Seek: 根据概率设置随机隐藏一些补丁



# YOLO系列-V4

✓ Self-adversarial-training(SAT)

✎ 通过引入噪音点来增加游戏难度

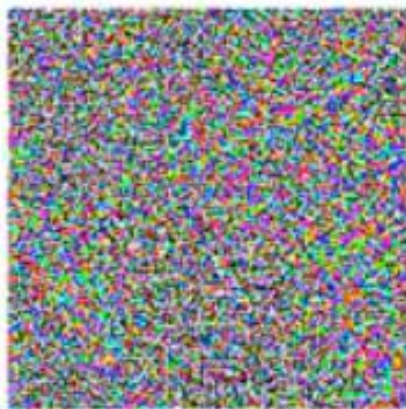


$x$

“panda”

57.7% confidence

+ .007 ×



$\text{sign}(\nabla_x J(\theta, x, y))$

“nematode”

8.2% confidence

=



$x +$

$\epsilon \text{sign}(\nabla_x J(\theta, x, y))$

“gibbon”

99.3 % confidence



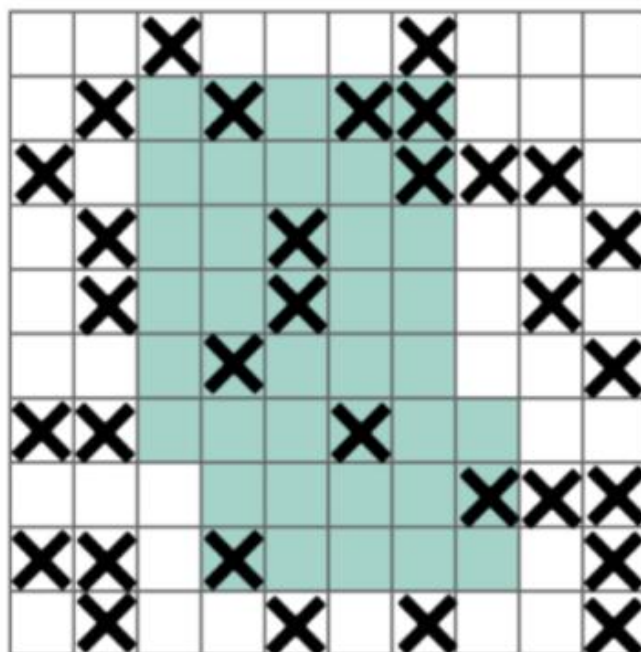
# YOLO系列-V4

## ✓ DropBlock

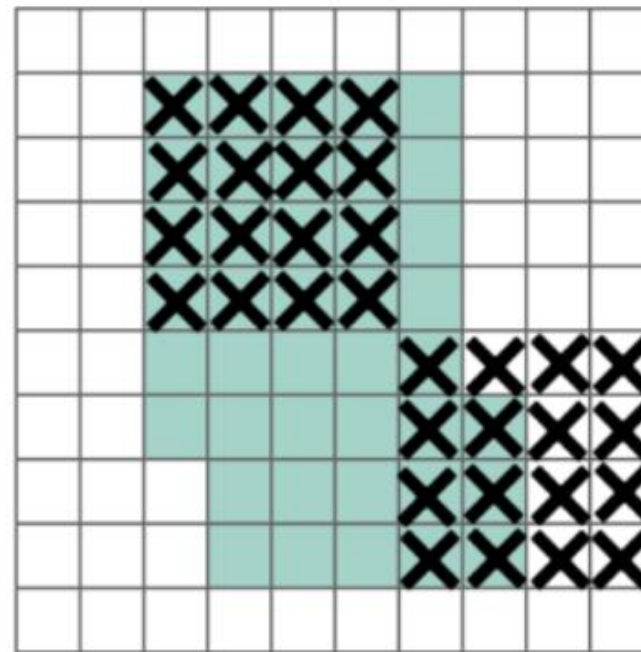
📌 之前的dropout是随机选择点(b)，现在吃掉一个区域



(a)



(b)



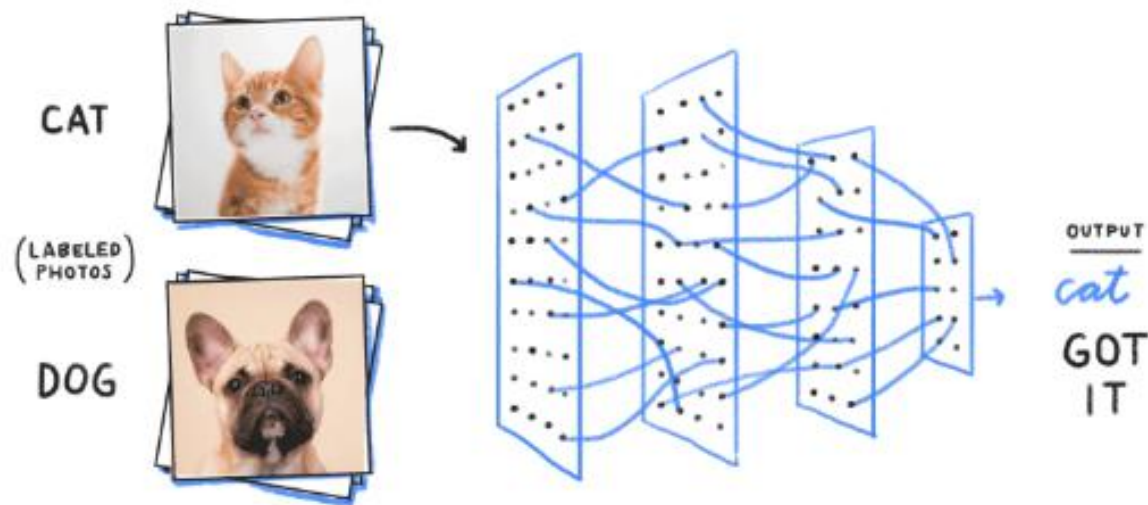
(c)

# YOLO系列-V4

## ✓ Label Smoothing

✎ 神经网络最大的缺点：自觉不错（过拟合），让它别太自信

✎ 例如原来标签为 (0,1) :  $[0, 1] \times (1 - 0.1) + 0.1/2 = [0.05, 0.95]$

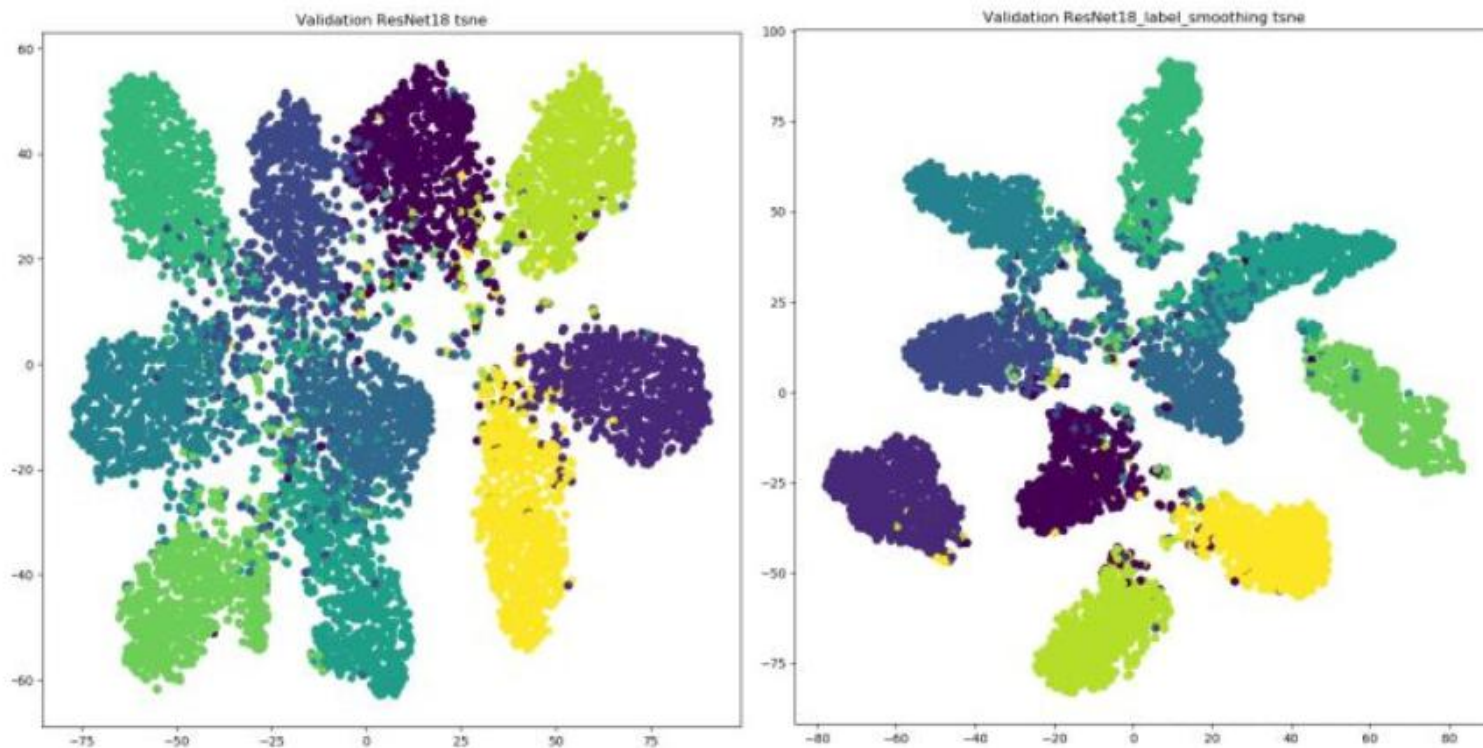




# YOLO系列-V4

## ✓ Label Smoothing

✎ 使用之后效果分析（右图）：簇内更紧密，簇间更分离



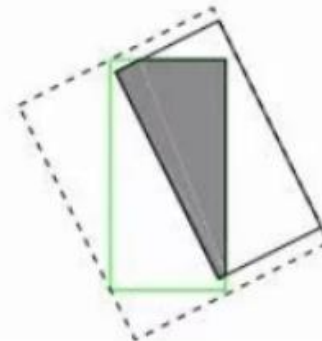
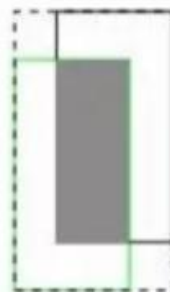
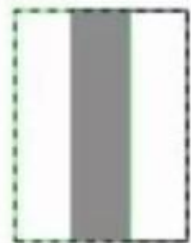
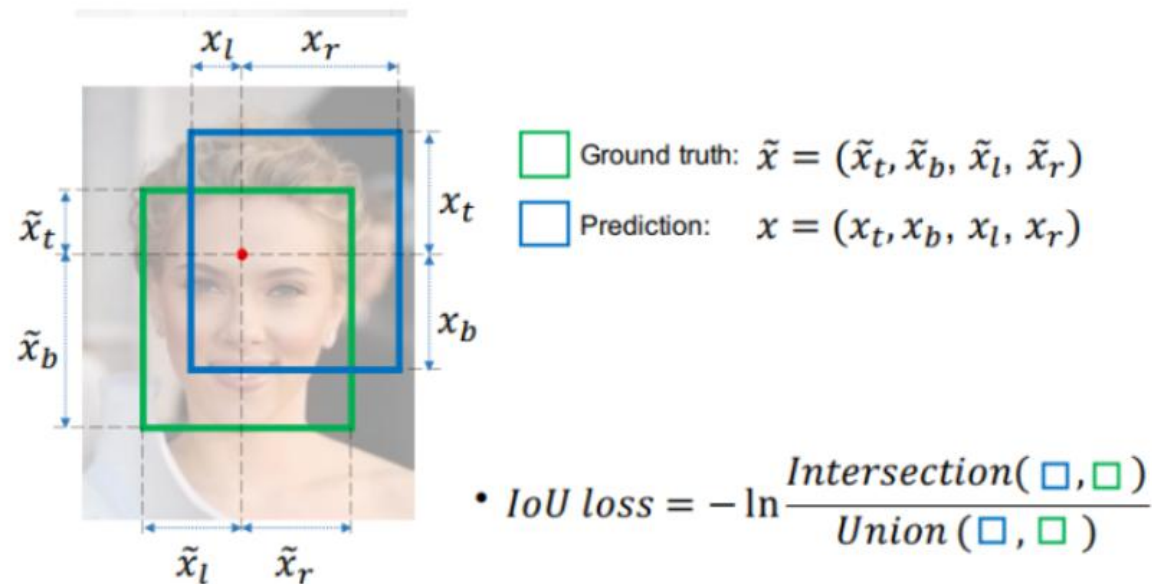
# YOLO系列-V4

✓ IOU损失

✎ IOU损失: (也经常1-IOU)

✎ 有哪些问题呢?

✎ 没有相交则IOU=0无法梯度计算, 相同的IOU却反映不出实际情况到底咋样



# YOLO系列-V4

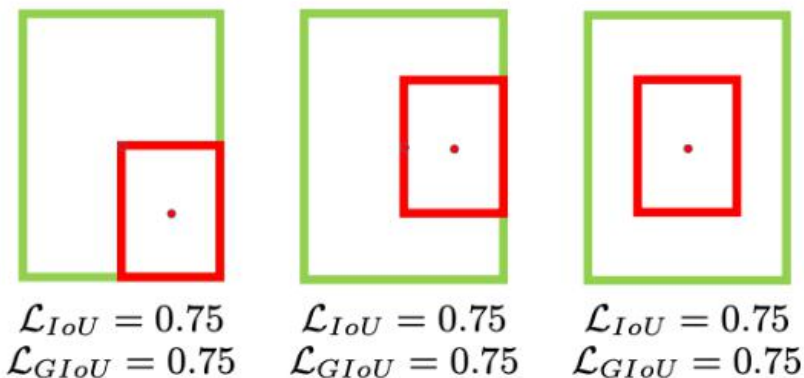
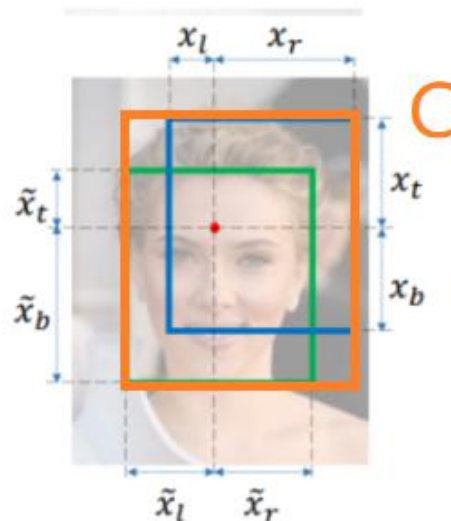
## ✓ GIOU损失

✎ 公式:  $\mathcal{L}_{GIOU} = 1 - IoU + \frac{|C - B \cup B^{gt}|}{|C|}$

✎ 引入了最小封闭形状C (C可以把A, B包含在内)

✎ 在不重叠情况下能让预测框尽可能朝着真实框前进

✎ 但是这种情况下又完了。。。



# YOLO系列-V4

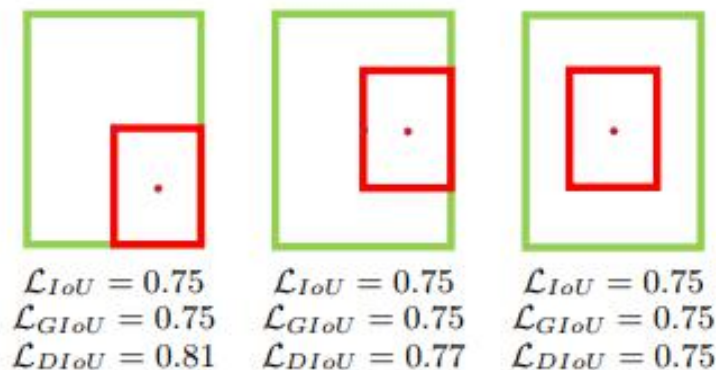
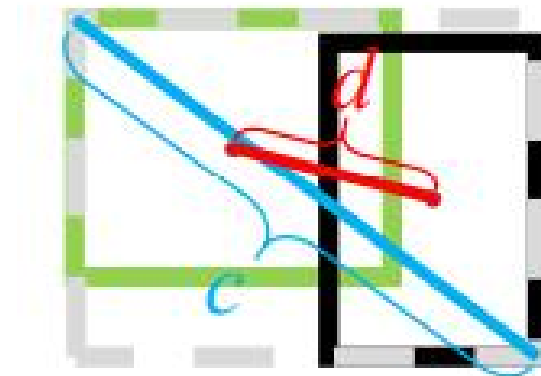
## ✓ DIOU损失

✎ 公式:  $\mathcal{L}_{DIOU} = 1 - IoU + \frac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2}$

✎ 其中分子计算预测框与真实框的中心点欧式距离d

✎ 分母是能覆盖预测框与真实框的最小BOX的对角线长度c

✎ 直接优化距离，速度更快，并解决GIOU问题



# YOLO系列-V4

## ✓ CIOU损失

✎ 公式:  $\mathcal{L}_{CIoU} = 1 - IoU + \frac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2} + \alpha v$

$$v = \frac{4}{\pi^2} \left( \arctan \frac{w^{gt}}{h^{gt}} - \arctan \frac{w}{h} \right)^2$$

$$\alpha = \frac{v}{(1 - IoU) + v}$$

✎ 损失函数必须考虑三个几何因素: 重叠面积, 中心点距离, 长宽比

✎ 其中 $\alpha$ 可以当做权重参数

# YOLO系列-V4

## ✓ DIOU-NMS

✎ 之前使用NMS来决定是否删除一个框，现在改用DIOU-NMS

✎ 公式: 
$$s_i = \begin{cases} s_i, & IoU - \mathcal{R}_{DIOU}(\mathcal{M}, B_i) < \varepsilon, \\ 0, & IoU - \mathcal{R}_{DIOU}(\mathcal{M}, B_i) \geq \varepsilon, \end{cases} \quad \mathcal{R}_{DIOU} = \frac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2}$$

✎ 不仅考虑了IoU的值,还考虑了两个Box中心点之间的距离

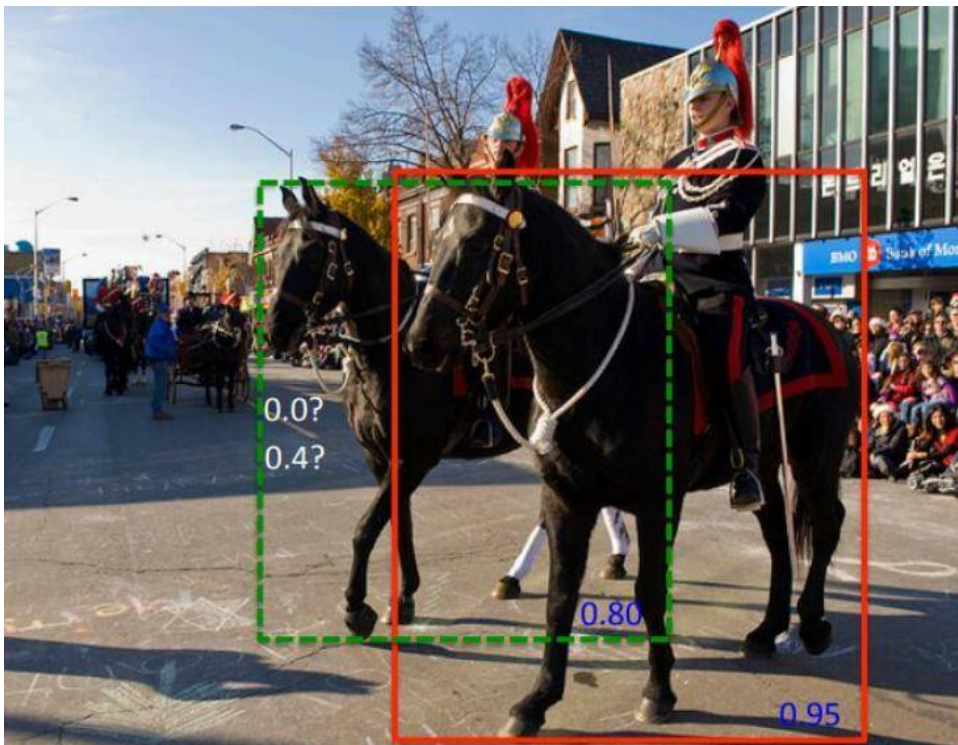
✎ 其中M表示高置信度候选框，Bi就是遍历各个框跟置信度高的重合情况



# YOLO系列-V4

## ✓ SOFT-NMS

📎 做人留一面日好相见，柔和一点的NMS，更改分数而且直接剔除



```
begin
   $\mathcal{D} \leftarrow \{\}$ 
  while  $\mathcal{B} \neq \text{empty}$  do
     $m \leftarrow \operatorname{argmax} \mathcal{S}$ 
     $\mathcal{M} \leftarrow b_m$ 
     $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{M}; \mathcal{B} \leftarrow \mathcal{B} - \mathcal{M}$ 
    for  $b_i$  in  $\mathcal{B}$  do
      if  $iou(\mathcal{M}, b_i) \geq N_t$  then
         $\mathcal{B} \leftarrow \mathcal{B} - b_i; \mathcal{S} \leftarrow \mathcal{S} - s_i$ 
      end
    end
     $s_i \leftarrow s_i f(iou(\mathcal{M}, b_i))$ 
  end
end
return  $\mathcal{D}, \mathcal{S}$ 
end
```

NMS

Soft-NMS

# YOLO系列-V4

---

✓ Bag of specials(BOS)

✎ 增加稍许推断代价，但可以提高模型精度的方法

✎ 网络细节部分加入了很多改进，引入了各种能让特征提取更好的方法

✎ 注意力机制，网络细节设计，特征金字塔等，你能想到的全有

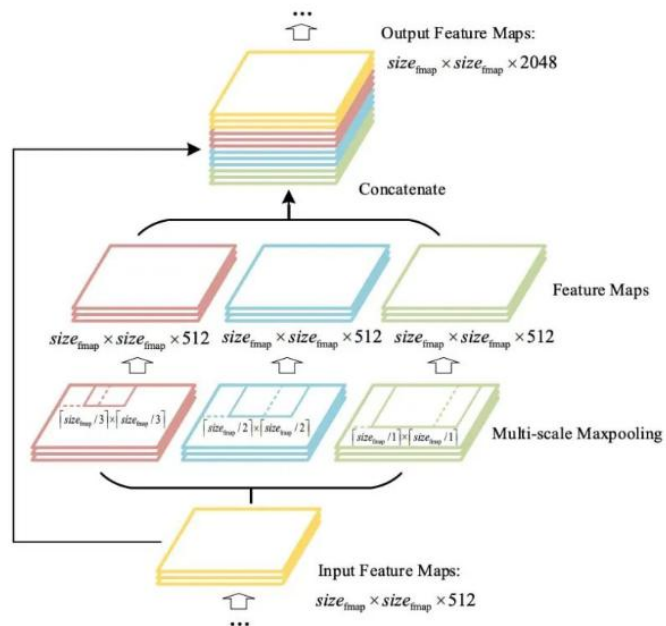
✎ 读折一篇相当于把今年来部分优秀的论文又过了一遍

# YOLO系列-V4

## ✓ SPPNet(Spatial Pyramid Pooling)

✎ V3中为了更好地满足不同输入大小，训练的时候要改变输入数据的大小

✎ SPP其实就是用最大池化来满足最终输入特征一致即可

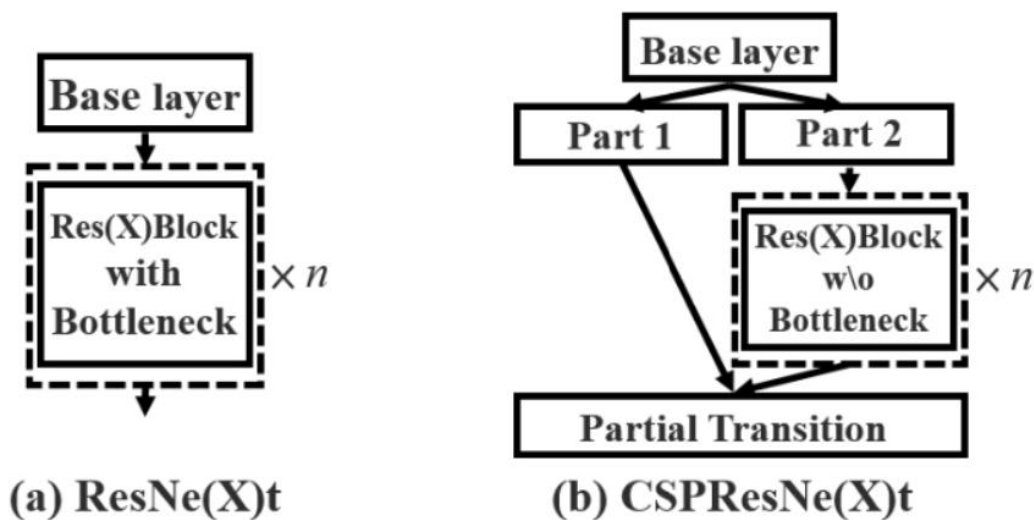


# YOLO系列-V4

✓ CSPNet (Cross Stage Partial Network)

✎ 每一个block按照特征图的channel维度拆分成两部分

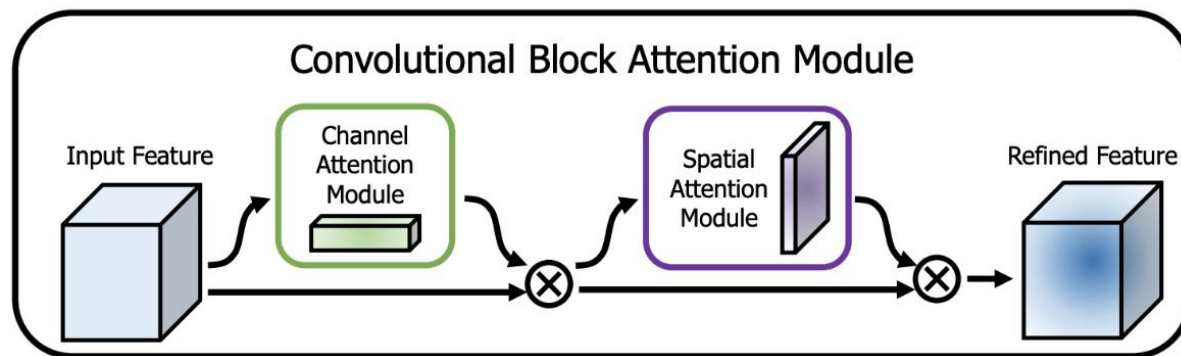
✎ 一份正常走网络，另一份直接concat到这个block的输出



# YOLO系列-V4

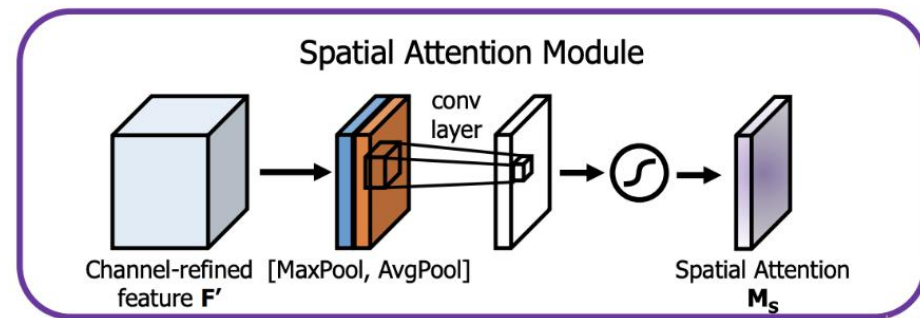
## ✓ CBAM

✎ 就是这个家伙：



✎ 其实就是加入了注意力机制，已经很常见了在各种论文中

✎ V4中用的是SAM，也就是空间的注意力机制

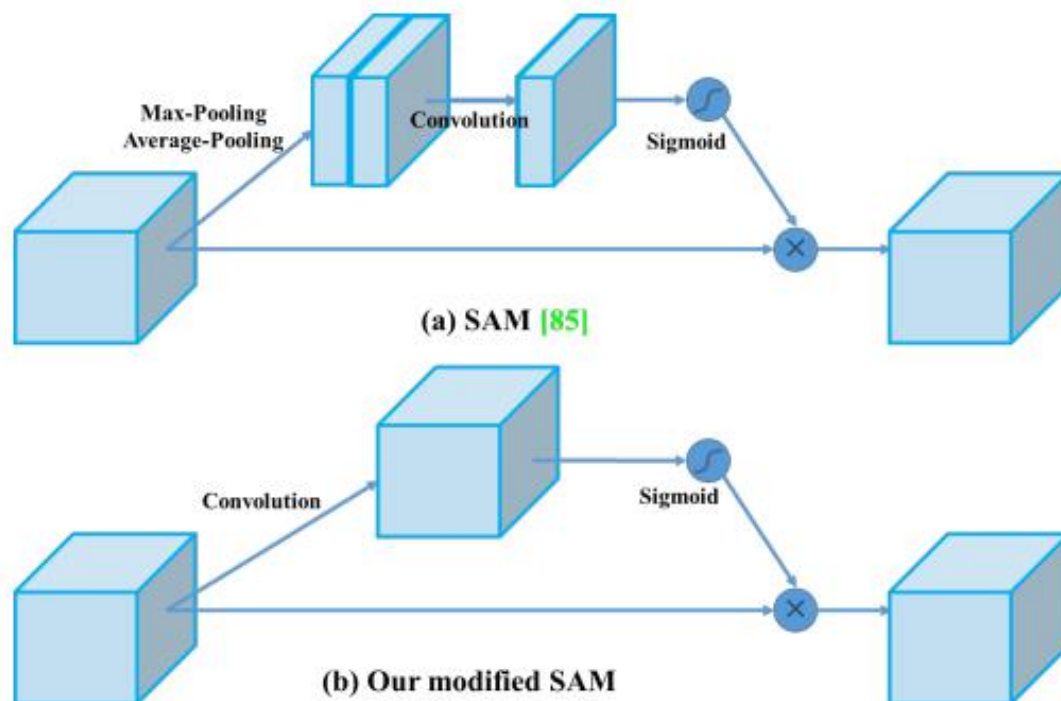


✎ 不光NLP,语音识别领域在搞attention，CV中也一样

# YOLO系列-V4

✓ YOLOV4中的Spatial attention module

✎ 一句话概述就是更简单了，速度相对能更快一点





# YOLO系列-V4

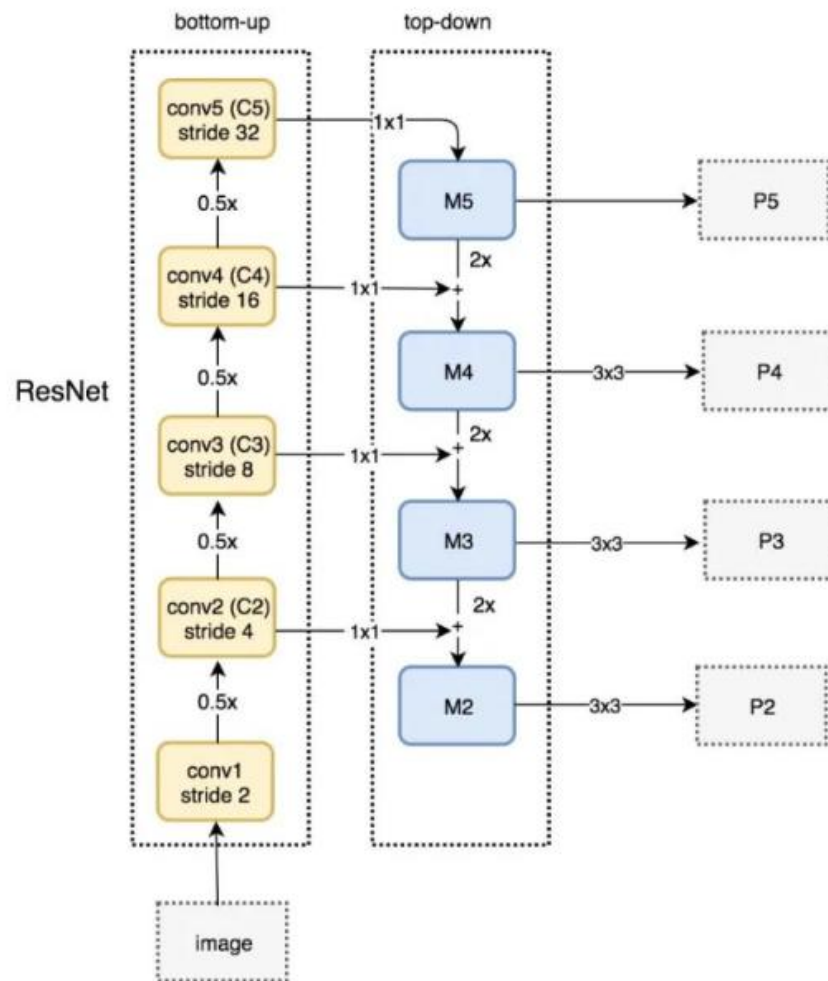
## ✓ PAN(Path Aggregation Network)

✍ 故事得先从FPN说起

✍ 自顶向下的模式，将高层特征传下来

✍ 好像只有一条路子，能不能来个双向的呢？

✍ 这就得轮到PAN登场了，思想也很简单

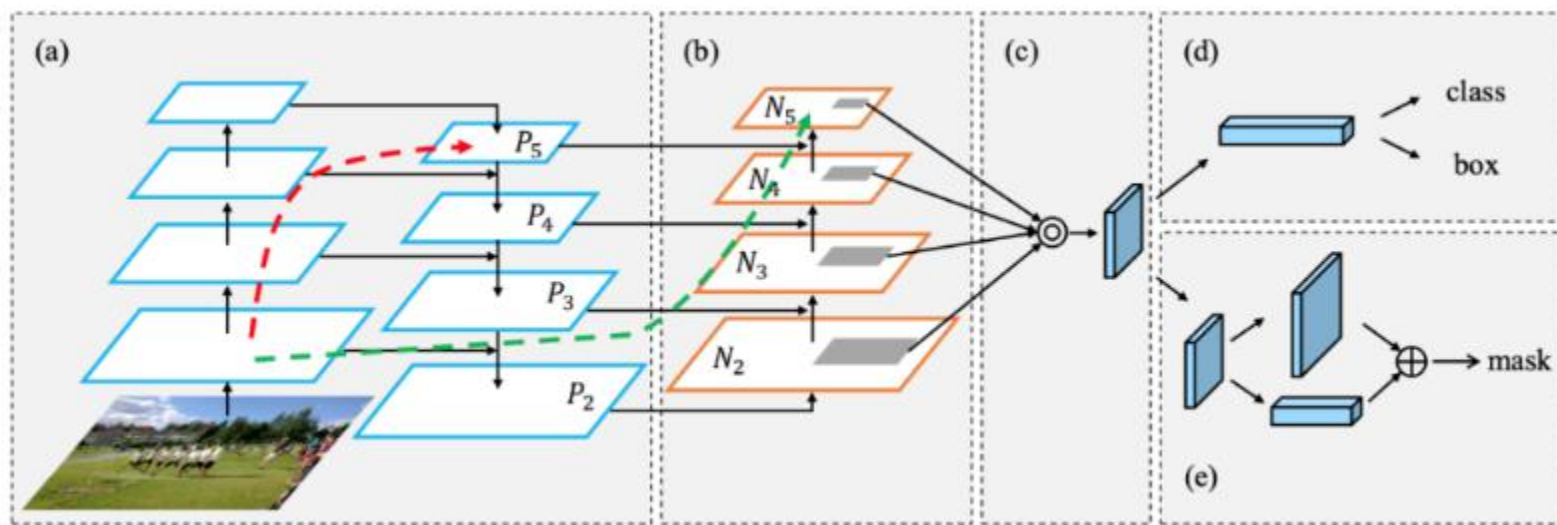


# YOLO系列-V4

✓ PAN(Path Aggregation Network)

✎ 引入了自底向上的路径，使得底层信息更容易传到顶部

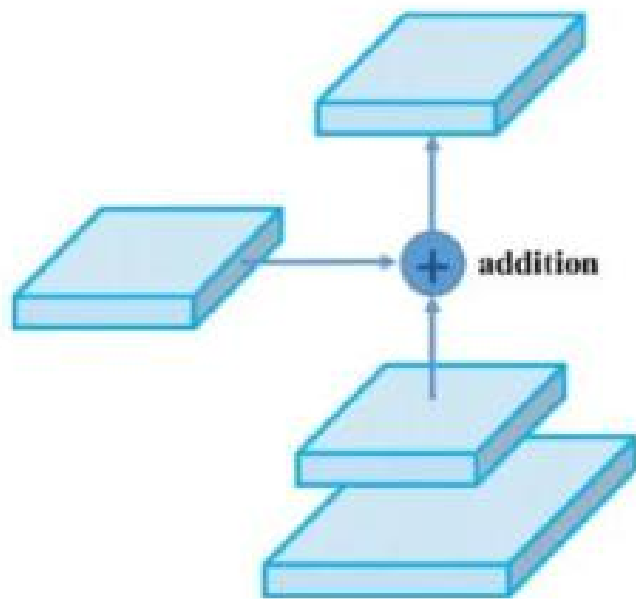
✎ 并且还是一个捷径，红色的没准走个100层(Resnet)，绿色的几层就到了



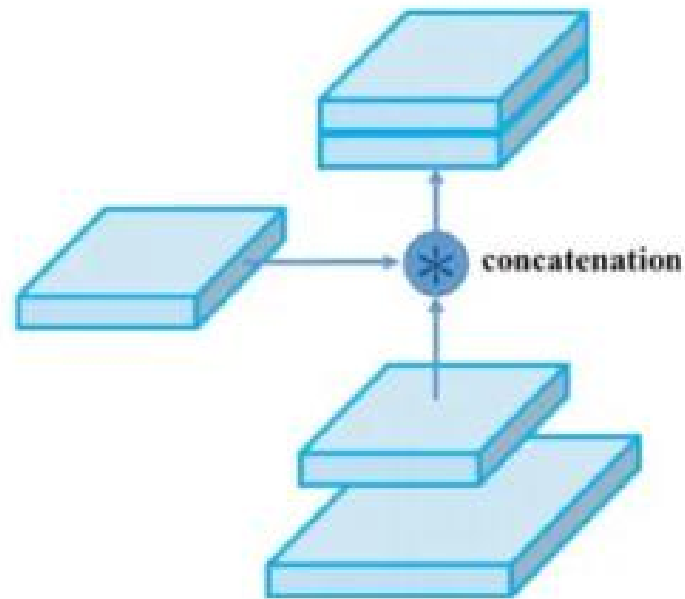
# YOLO系列-V4

✓ PAN(Path Aggregation Network)

✎ YOLOV4中并不是加法，而是拼接



(a) PAN [49]



(a) Our modified PAN

# YOLO系列-V4

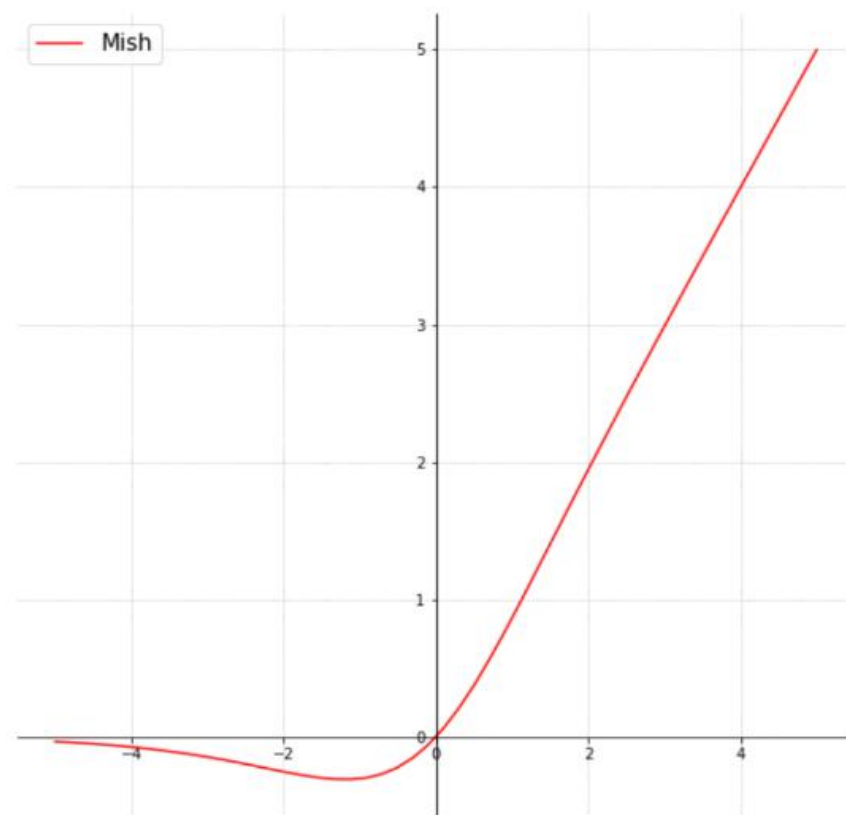
✓ Mish: (也许就是明日之星)

✎ 别一棒子全给打死, 给个改过自新的机会

✎ Relu有点太绝对了, Mish更符合实际

✎ 公式:  $f(x) = x \cdot \tanh(\ln(1 + e^x))$

✎ 但是计算量确实增加了, 效果会提升一点



# YOLO系列-V4

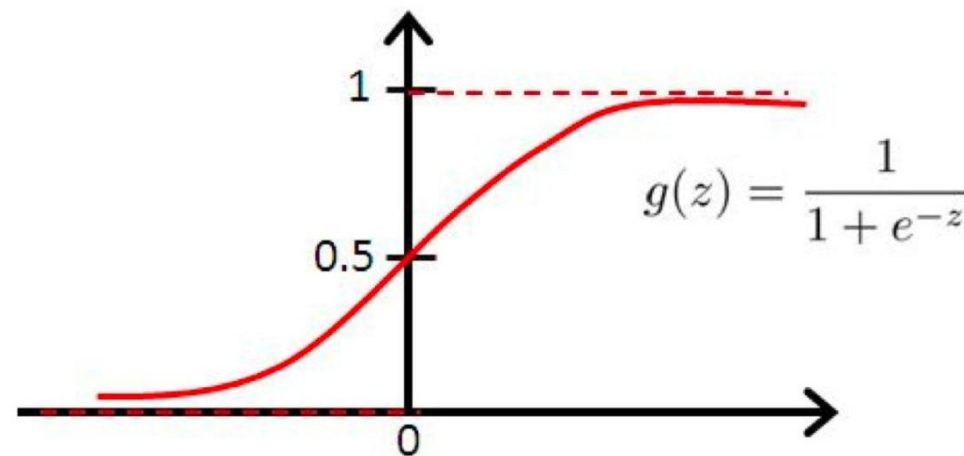
✓ eliminate grid sensitivity

✎ 比较好理解，坐标回归预测值都在0-1之间，如果在grid边界怎么表示？

✎ 此时就需要非常大的数值才可以达到边界

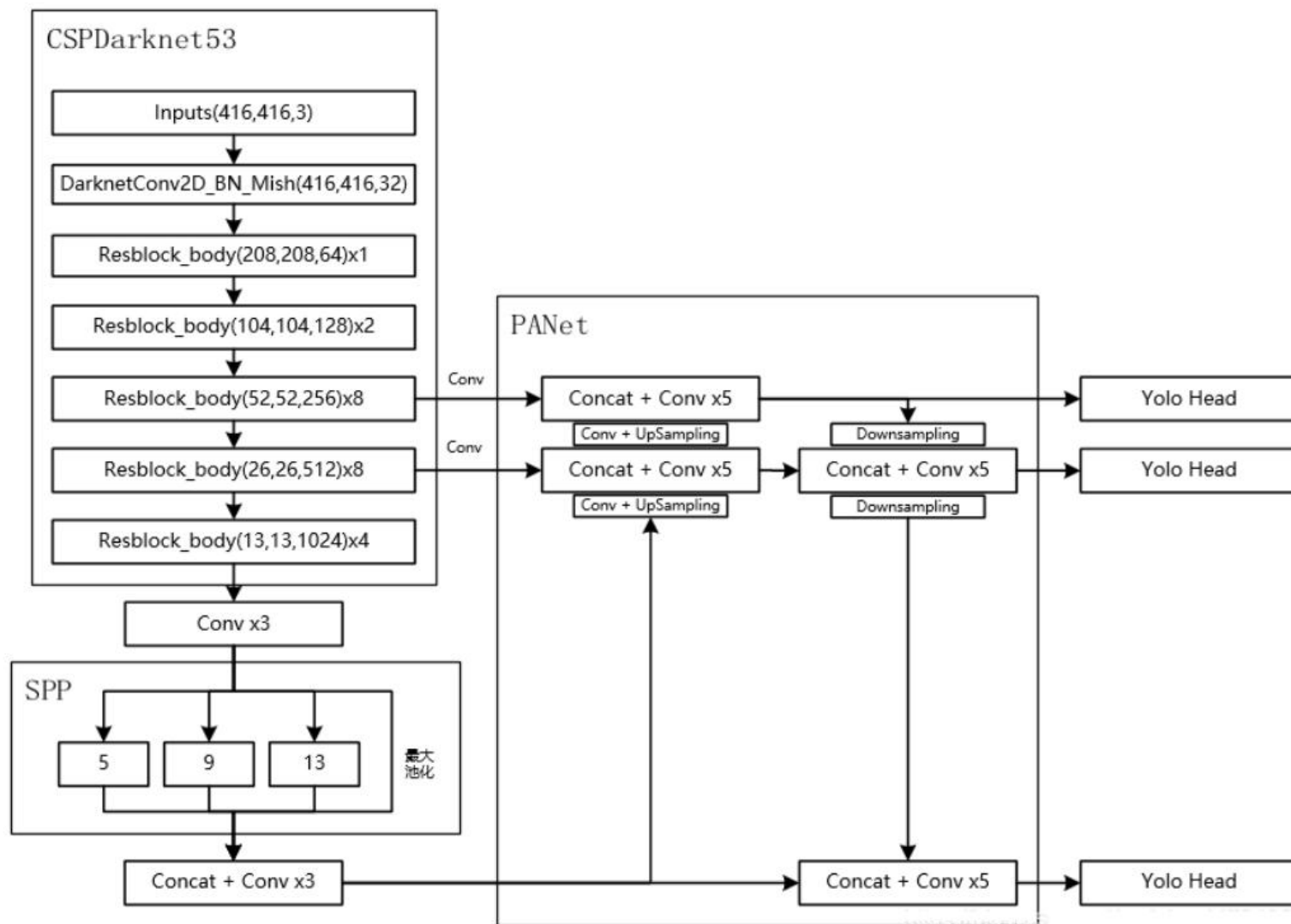
✎ 为了缓解这种情况可以在激活函数前加上一个系数（大于1的）：

$$b_y = \sigma(t_y) + c_y$$



# YOLO系列-V4

## ✓ 整体网络架构





# YOLOV5源码

## ✓ 可视化工具

- ✎ 1.配置好netron, 详情: <https://github.com/lutzroeder/netron>  
桌面版: <https://lutzroeder.github.io/netron/>
- ✎ 2.安装好onnx, `pip install onnx`即可
- ✎ 3.转换得到onnx文件, 脚本原始代码中已经给出
- ✎ 4.打开onnx文件进行可视化展示 (.pt文件展示效果不如onnx)

# YOLOV5源码

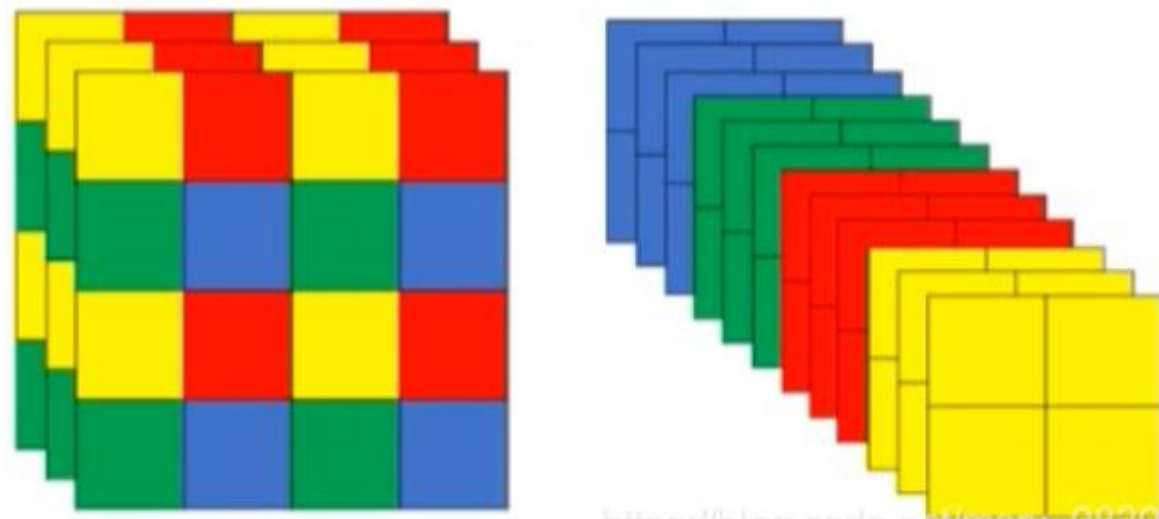
✓ Focus模块:

✎ 先分块，后拼接，再卷积

✎ 间隔的来完成分块任务

✎ 此时卷积输入的C就为12了

✎ 参考实验结果并不多，目的是为了加速，并不会增加AP



# YOLOV5源码

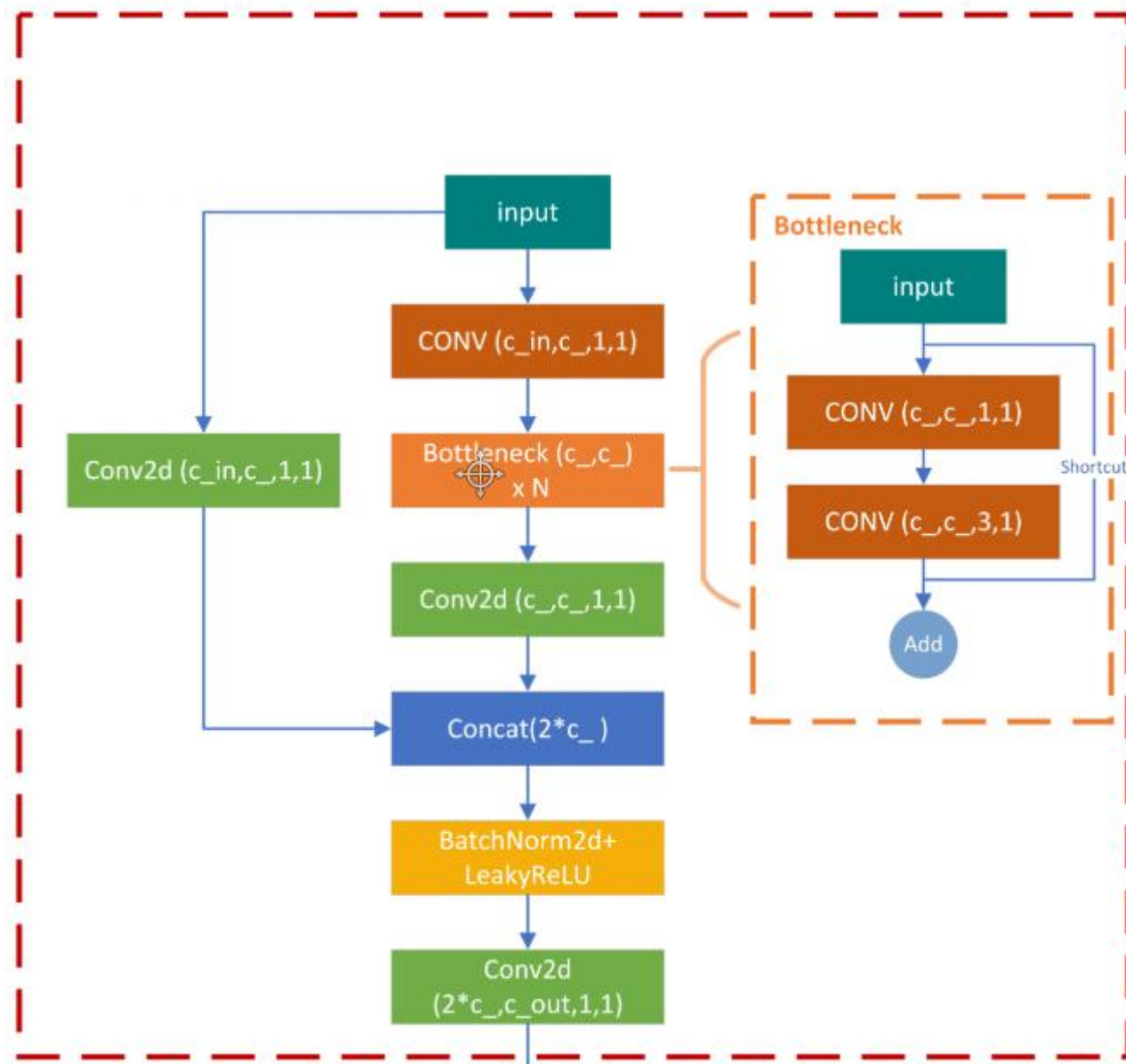
## ✓ BottleneckCSP

✎ 注意叠加个数

✎ 里面还包括了resnet模块

✎ 与V3版本类似，多了CSP

✎ 效果有一定提升



# YOLOV5源码

## ✓ PAN流程

