目标检测

李乡儒

华南师范大学计算机学院

2022 年 5 月 6 日

華南師範大學
SOUTH CHINA NORMAL UNIVERSITY

目录 I

内容概要

- 什么是目标检测
- 如何表示
- 基准数据集
- 评价指标
- 检测方法
- You Only Look Once
- Fast R-CNN

目录 I

目标检测

- 是什么?
- 在哪里?
- 有哪些?

“是什么”意味着需要判断出找出来的目标是什么, 也就是对目标的类别做判断, 是人还是狗或者是车

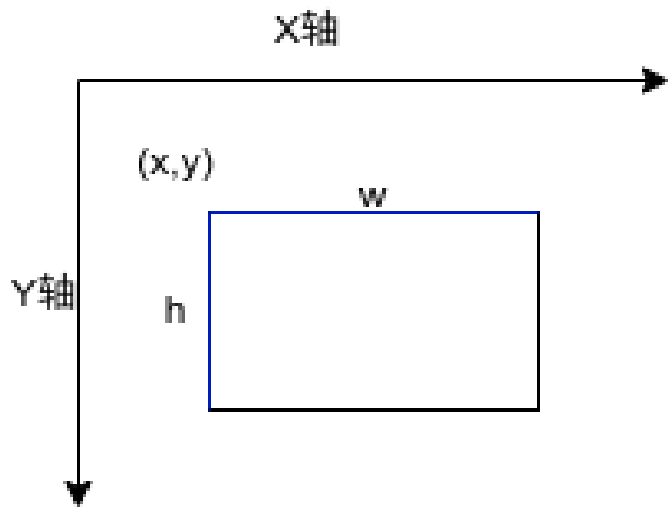
“在哪里”需要指出找到的目标在图像的那个地方, 范围是哪里

“有哪些”意味着需要将图像当中所有的感兴趣物体 (预先定义的类别) 找出来

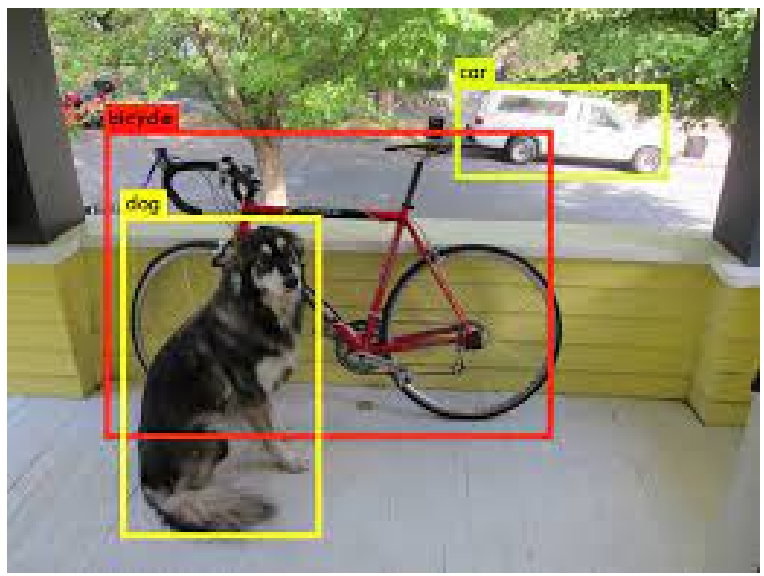
如何表示

在计算机当中需要用数字来表达 “是什么”、“在哪里”、“在哪里”

- 整数 (one-hot 向量) 表示类别
- 矩形框表示位置 (四元组 $[x, y, w, h]$)
- 堆叠所有目标组成数组 $n \times 5$



(a) box



(b) Object Detection

模型表示

知道了如何用数字表达检测结果, 那么如何让网络模型输出这些数据?

- 分类表达

  对于分类, 已经很熟悉了,Softmax 计算类别概率就能得到类别

- 预测框表达

  预测框其实是一个四维向量, 很容易就能想到用回归来拟合

- 所有物体表达

  所有物体实际上是一个矩阵, 是不是也可以用回归来拟合?

模型表示

实际上让神经网络输出所有物体的预测信息是很困难的, 因为每张图片当中的物体数量都是不确定的, 而神经网络无法根据内容调整输出的数量.

- “广撒网, 多敛鱼, 择优而从之”

  广撒网: 生成过量的预测框

  多敛鱼: 尽可能覆盖所有的目标

  择优而从之: 询问模型是否存在目标

通过这种策略, 目标检测就变成了“在不在、是什么”的问题

目录 I

目标检测公共数据集

Pascal VOC，ILSVRC，MS-COCO，和 Open Images 数据集是目标检测使用最多的四大公共数据集

PASCAL VOC： http://host.robots.ox.ac.uk/pascal/VOC/

ILSVRC： https://www.image-net.org/challenges/LSVRC/

MS-COCO： https://cocodataset.org/#home

Open Images： https://storage.googleapis.com/openimages/web/index.html

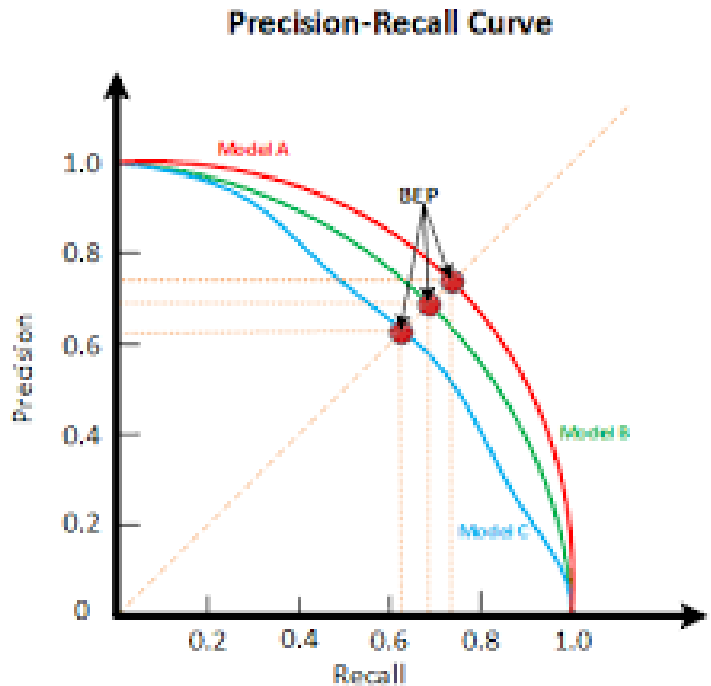
本课程使用 COCO-2017 数据集作为示例.

评价指标

目标检测的目标是三个方面，那评价指标自然要考虑这三个内容.

- 分类指标: 准确率/精度/召回率/F1 值等
- 目标框匹配:IoU
- 检测是否完全: 召回率/精度

最主要的一个指标是 mAP(mean Average Precision), 其中 m 是对类别取平,AP 是 Precision-Recall 曲线下面积.
运算过程如下：
对于每个类别:
　　对于每张图片:
　　　　求 Precision 和 Recall
　　　　将点 (r,p) 记录到曲线
　　求曲线下面积
求平均面积



目录 I

目标检测有两个主要流派:Anchor-based 与 Anchor-Free, 两类算法的主要区别在于算法当中是否有预设锚框

- Anchor-based:YOLOv3、SSD、Faster R-CNN
- Anchor-Free:YOLOv1、CornerNet、CenterNet、FCOS

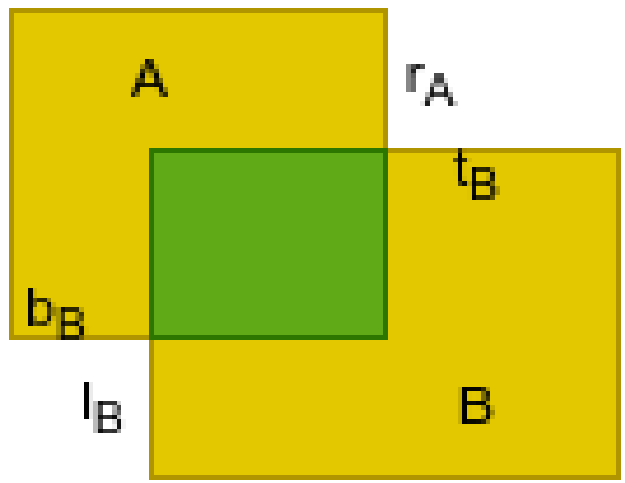
另外一个划分是: 一阶段和二阶段算法, 阶段主要是指从输入到结果之间网络模型的运算次数

- 一阶段:YOLO、SSD、CornerNet、CenterNet、FCOS
- 二阶段:R-CNN、Faster R-CNN

在后面的算法当中将介绍二阶段 Anchor-based 算法 Faster R-CNN 和一阶段 Anchor-Free 算法 YOLOv1

基础知识

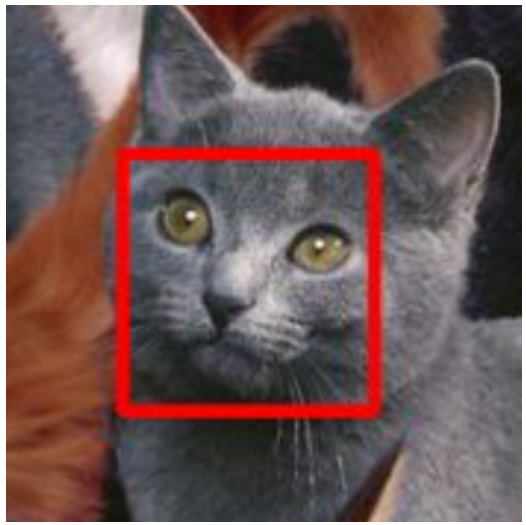
- 矩形框的 3 种表示矩形框的表示形式有:
  $[x, y, w, h]$
  $[x_1, y_1, x_2, y_2]([left, top, right, bottom])$
  $[cx, cy, w, h]$
- 交并比 (IoU) 交并比是指两个矩形的交集与并集的面积之比
  $IoU = \frac{|A \cap B|}{|A \cup B|}$, 实现是采用第二种矩形表示进行实现
  $A[l_A, t_A, r_A, b_A], B[l_B, t_B, r_B, b_B]$:
  $S_A = (r_A - l_A) \times (b_A - t_A)$
  $S_B = (r_B - l_B) \times (b_B - t_B)$
  $W_{AB} = (min(r_A, r_B) - max(l_A, l_B))$
  $H_{AB} = (min(b_A, b_B) - max(t_A, t_B))$
  if $W_{AB} < 0$ or $H_{AB} < 0$ then $IoU = 0$
  else $IoU = \frac{W_{AB} \times H_{AB}}{S_A + S_B - W_{AB} \times H_{AB}}$



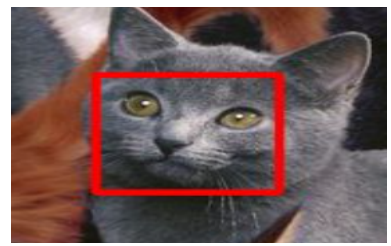
```
def boxIou(boxes1: Tensor, boxes2: Tensor)->Tensor:
    # boxes1:(N,4) boxes2:(M,4) use 'xyxy'
    area1=(boxes1[:,2]-boxes1[:,0])*(boxes1[:,3]-boxes1[:,1])
    area2=(boxes1[:,2]-boxes2[:,0])*(boxes1[:,3]-boxes2[:,1])
    lt=torch.max(boxes1[:,None,:2],boxes2[:,:2]) # [N,M,2]
    rb=torch.min(boxes1[:,None,2:],boxes2[:,2:]) # [N,M,2]
    wh=(rb-lt).clamp (min=0)
    inter=wh[...,0]*wh[...,1] # [N,M]
    union=area1[:,None]+area2-inter
    return inter/union
```

矩形框 resize

在实际操作当中需要对输入的图像进行 resize 操作, 但是这会导致矩形框的偏移, 因此需要对矩形框进行相应变化



(c) 源图像



(d) 目标图像

图像宽高改变后, 坐标点 (x, y) 和矩形框的宽高 (w, h) 按比例变化, 源宽高为 (w_o, h_o), 目标宽高为 (w_t, h_t):

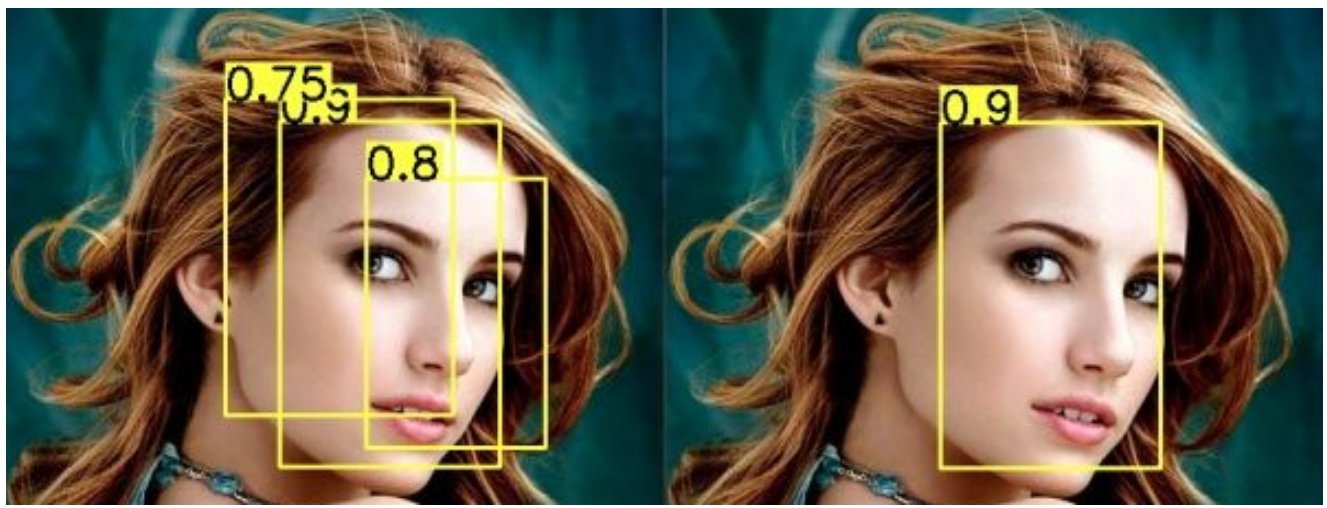
$$x' = \frac{xw_t}{w_o}, y' = \frac{yh_t}{h_o}$$

```
@no_grad()
def resizeBox(
        orgsize:Tuple[int,int],
        tarsize:Tuple[int,int],
        boxes:Tensor
        )->Tensor:
        orgsize,tarsize=Tensor(orgsize),Tensor(tarsize)
        matrix=torch.diag(tarsize/orgsize)
        return torch.cat([boxes[:,:2]@matrix,boxes[:,:2]@matrix],-1)
```

非极大值抑制

非极大值抑制 (NMS) 主要用来去除重复的预测框，其输入是预测框数组与对应的置信度数组，算法过程如下:

按置信的从大到小排序
While 数组非空:
    取出置信度最大的预测框 B
    B 放入结果集
    对于剩余数组当中每一个预测框 b:
        if IoU(B,b)> 阈值:
            从数组中删除 b

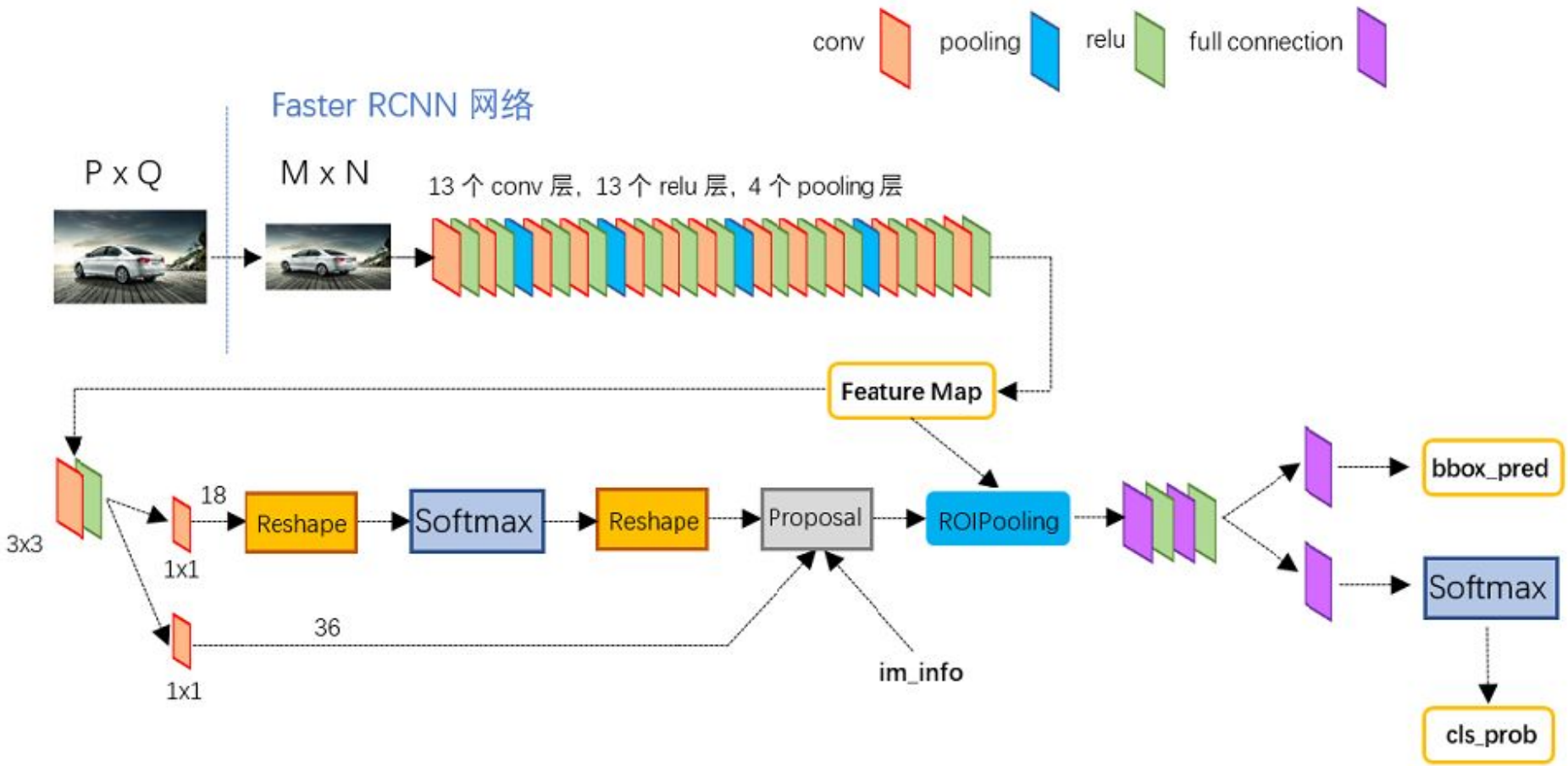


```
@no_grad()
def nms(boxes:Tensor,score:Tensor, threshold:float)->Tensor:
    # boxes:(N,4) score:(N)
    _,indexes=score.sort(descending=True)
    result=[]
    while 0 not in indexes.shape:
        result.append(indexes[0])
        temp_boxes=boxes[indexes]
        ious=boxIou(temp_boxes[:1],temp_boxes[1:])[0]
        indexes=indexes[1:][ious<=threshold]
    return torch.stack(result)
```

目录 I

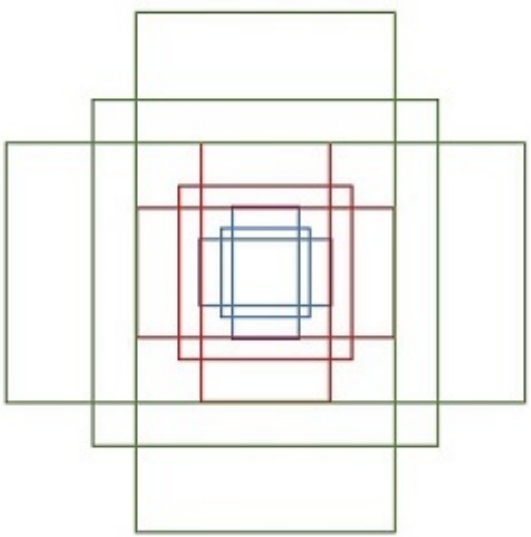
Faster R-CNN

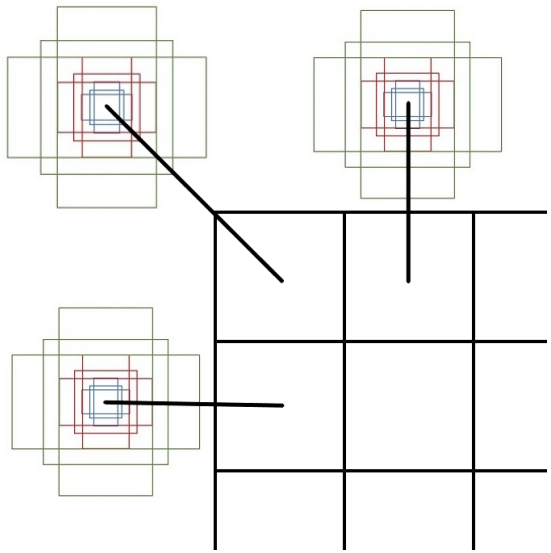
Faster R-CNN 是一个较为成熟的二阶段 Anchor-based 目标检测算法

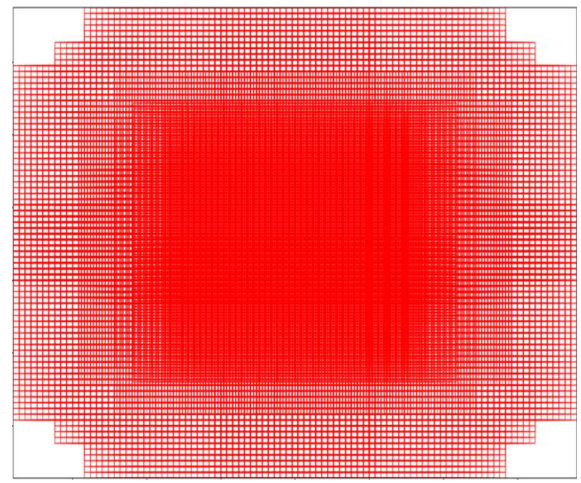


预设锚框

Faster R-CNN 中设置了 3 种比例的锚框,1:1,1:2,2:1. 并且使用了多种尺寸,$64^2, 128^2, 256^2, 512^2 \cdots$







锚框平铺

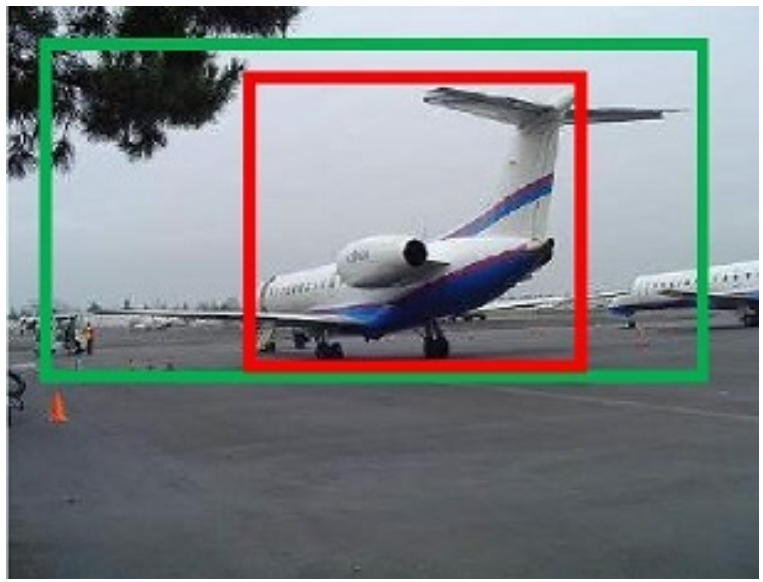
将图片分割为 $P \times Q$ 的网格, 在每一个网格上应用预设锚框, 则锚框数量一共有 $3 \times P \times Q \times S$, 其中 S 为应用的尺寸的数量.

```
def tileAnchors(
    anchors:Tensor,
    gridsize:Tuple[int,int],
    imagesize:Tuple[int,int]
    )->Tensor:
    baselength=Tensor(imagesize)/Tensor(gridsize)
    basepos=torch.stack(torch.meshgrid(
        torch.arange(baselength[0]/2,imagesize[0],baselength[1]),
        torch.arange(baselength[1]/2,imagesize[1],baselength[1]),
        indexing='xy'),dim=-1) # (H,W,2)
    baseanc=torch.zeros((*basepos.shape[:2],anchors.shape[0],4))
    baseanc[...,:2]=basepos.unsqueeze(-2)-anchors/2
    baseanc[...,2:]=basepos.unsqueeze(-2)+anchors/2
    return baseanc
```

边框回归

在将锚框平铺到整张图像后, 预设锚框基本上覆盖了所有目标, 但是还不够精确, 而边框回归让预测框更加准确.Faster R-CNN 使用了如下形式的变换将锚框 A 映射到预测框 G:

$$
\begin{aligned}
G_x &= A_w d_x(A) + A_x \\
G_y &= A_g d_y(A) + A_y \\
G_w &= A_w exp(d_w(A)) \\
G_h &= A_h exp(d_h(A))
\end{aligned}
$$



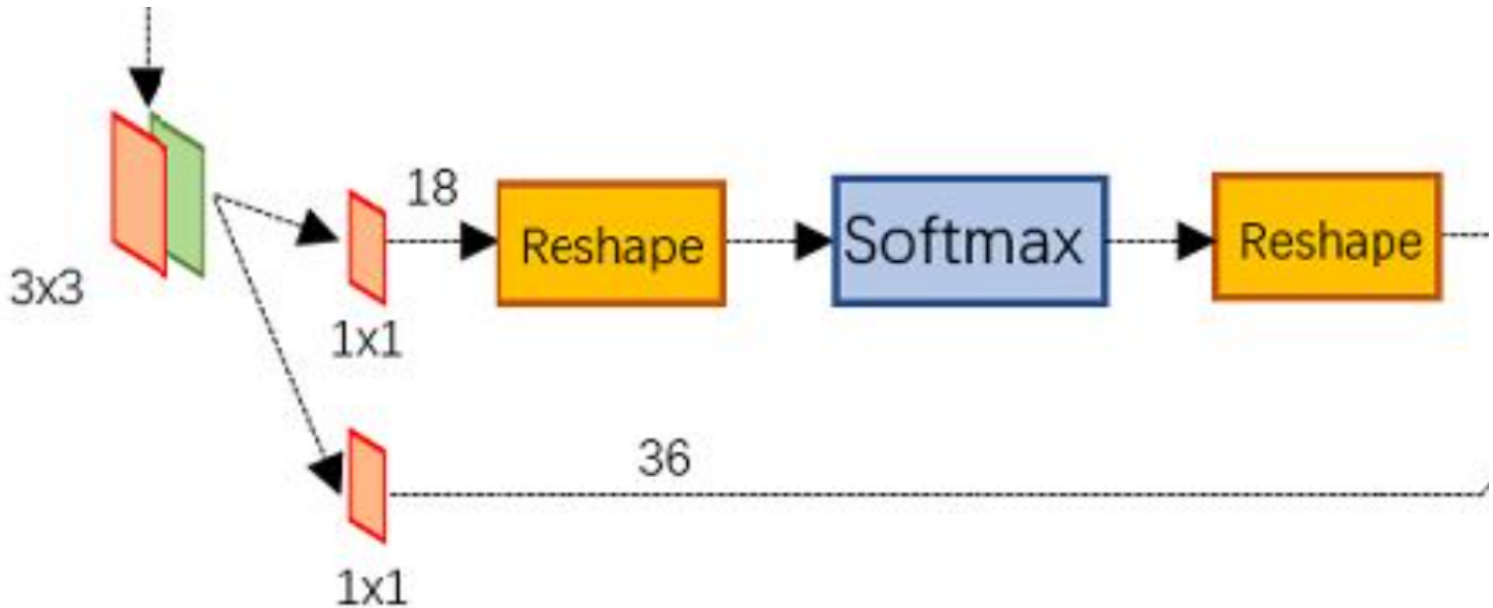
因此对预测框的校准问题就变成了对 $d_x(A)d_y(A)d_w(A)d_y(A)$ 的回归问题

```
def calculatedxywh(anchors:Tensor,boxes:Tensor)->Tensor:
    # anchors:(...,4) boxes:(...,4)
    dxywh=torch.zeros_like(anchors)
    dxywh[...,:2]=(boxes[...,:2]-anchors[...,:2])/anchors[...,2:]
    dxywh[...,2:]=torch.log(boxes[...,2:]/anchors[...,2:])
    dxywh=dxywh.nan_to_num(0)
    return dxywh

def applydxywh(anchors:Tensor,dxywh:Tensor)->Tensor:
    # anchors:(...,4) dxywh:(...,4)
    matrix=torch.zeros(*dxywh.shape,4)
    matrix[...,:2,2:]=dxywh[...,:2].diag_embed()
    matrix[...,:2,:2]=1-matrix[...,:2,2:]
    matrix[...,2:,2:]=torch.exp(dxywh[...,2:]).diag_embed()
    matrix[...,2:,:2]=1-matrix[...,2:,2:]
    anchors=(matrix@anchors.unsqueeze(-1)).squeeze(-1)
    return anchors
```

RPN 层

RPN 层的主要工作是预测预设框内是否有目标物体以及预测从预设框到预测框的变换 d_{xywh}



其中预设框内是否有目标物体是一个二分类问题, 使用通道为 $2K$ 的 1×1 卷积实现并且在 reshape 后经过 Softmax 运算, 预测变换使用的是通道为 $4K$ 的 1×1 卷积

Proposal 层

Proposal Layer 的工作是根据 RPN 层的预测信息挑选出合适的预测框, 算法如下:

输入: 预设框内有物体的置信度, 预设框, 变换的回归估计

1 根据置信度从大到小排序, 选出置信度最高的 N 个预测框
2 使用变换的回归估计对预设框进行修正
3 修正超出图像边界的预测框
4 剔除过小的预测框
5 对剩余预测框执行非极大值抑制 (NMS)

到这一步已经完成了” 在哪里” 和” 有哪些” 的任务, 后面的工作就是分类以及对结果的轻微修正

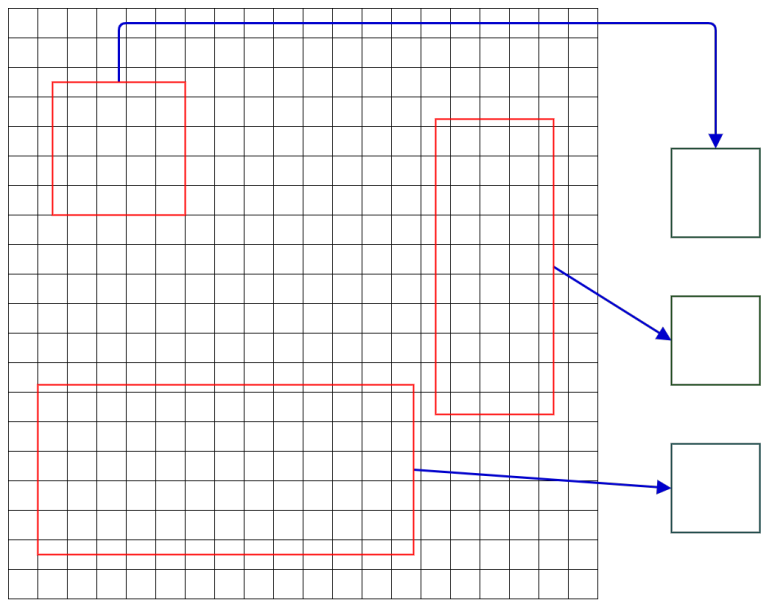
```
def proposal(anchs:Tensor,boxreg:Tensor,score:Tensor,N:int,
    imagesize:Tuple[int,int],minsize:int,threshold:float)->Tensor:
    # anchors:(N,H,W,K,4) boxreg:(N,H,W,K,4) score:N,H,W,K
    idx=score>0.5
    anchs,boxreg,score=applyIndex(idx,anchs,boxreg,score)
    if score.shape[0]>N:
        idx=score.topk(N).indices
        anchs,boxreg,score=applyIndex(idx,anchs,boxreg,score)
    boxes=applydxywh(anchors,boxreg)
    boxes=clip_boxes_to_image(boxes,imagesize)
    idx=remove_small_boxes(boxes,minsize)
    boxes,score=applyIndex(idx,boxes,score)
    idx=nms(boxes,minsize,threshold)
    return boxes[idx].clone()
```

ROI Pooling

Region of interest pooling 是对选定的区域进行采样的过程, 在得到预测框后需要将区域的图像输入 CNN 分类器当中, 这要求这些区域的分辨率保持一致.

首先,Faster R-CNN 是在下采样后的特征图上进行采样的, 因此需要把预测框转换为特征图的尺寸.

接下来, 对预测框进行取整来对特征图进行切片并将进行池化, 使特征图切片的大小统一. 这里使用自适应最大/平均池化来代替原论文的复杂池化过程.

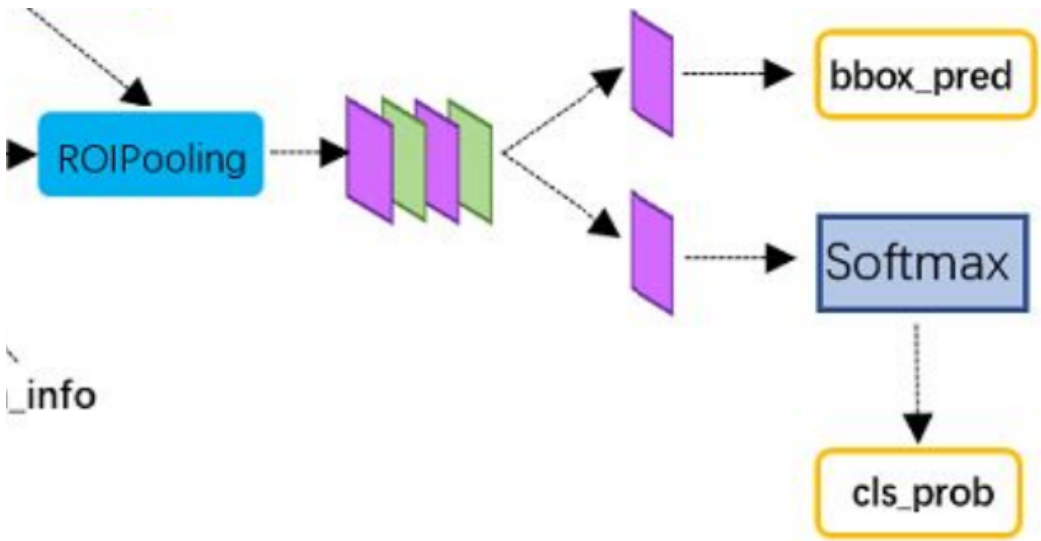


```
def roiPooling(
        feamap:Tensor,
        boxes:Tensor,
        imgsize:Tuple[int,int],
        outsize:Tuple[int,int]):
        # feamap (1,C,H,W) boxes:(N,4)
        h,w=feamap.shape[-2:]
        boxes=resizeBox(imgsize,(w,h),boxes)
        boxes=boxes.floor()
        roiarea=torch.cat([F.adaptive_avg_pool2d(
                feamap[...,box[1]:box[3],box[0]:box[2]],outsize)
                for box in boxes])
        return roiarea
```

第二阶段

第二阶段的任务是一个简单的分类 + 回归任务.

- 输入:ROI 区域特征图 ([M,C,H,W])
- 网络: 卷积网络
- 输出 1: 基于第一次预测框的更精确的预测框回归
- 输出 2:ROI 区域物体类别



训练过程

Faster R-CNN 的特征提取器使用预训练的卷积网络 (resnet18、vgg16 等)

1 固定特征提取器, 训练 RPN 层

2 固定特征提取器和 RPN, 训练分割部分

3 整体训练

在两个阶段的训练当中都有回归任务和分类任务, 回归任务使用的是 L1 Smooth Loss, 分类使用 CrossEntrpyLoss

$$l = \begin{cases} 0.5(y - \hat{y})^2/\beta, & \text{if } |y - \hat{y}| < \beta \\ |y - \hat{y}| - 0.5\beta, & \text{otherwise} \end{cases}$$

数据处理

在实际训练时, 需要将原始数据格式转换成模型需要的输入或标签, 这里以 COCO 数据集为例说明需要做哪些处理:

1 获取图片、定位框、目标分类标签

2 将图片 resize 到指定大小方便训练

3 将定位框转换到新的坐标系

4 生成预设框内有/无物体的分类标签

5 生成预设框到定位框的变换 dxywh

其中生成的两种数据仅训练 RPN 层时使用, 定位框与分类标签在训练第二阶段网络是使用.

```
class FasterRCNNCOCODataset(CocoDetection):
    def __init__(self,
        root:str,
        annFile:str,
        gridsize:Tuple[int,int],
        imagesize:Tuple[int,int],
        imagetransformers:Optional[Callable],
        outputItems:List[str]):
        super().__init__(root,annFile)
        self.anchorsWH=anchors
        self.anchors=tileAnchors(anchors,anchors
    def getPositiveNegativeRegression(self,boxes:Tensor):
        ious=boxIou(self.anchors.view(-1,4),
            box_convert(boxes,'xywh','xyxy'))
        ious=ious.reshape(*self.anchors.shape[:3],boxes.shape[0])
        maxiou,indexes=ious.max(-1)
        currentgt=boxes[indexes]
    return torch.stack([(maxiou>0.7).long(),(maxiou<0.3).long()]),
            calculatedxywh(self.anchors,currentgt)
```

```
def __getitem__(self,index:int):
    image,target=super().__getitem__()
    image=TF.to_tensor(image)
    h,w=image.shape[-2:]
    image=TF.resize(image,(self.imagesize[1],self.imagesize[0]))
    if self.imagetransformers:
        image=self.imagetransformers(image)
    bboxes=Tensor([_['bbox'] for _ in target])
    bboxes=resizeBox((w,h),self.imagesize,bboxes)
    category_ids=Tensor([_['category_id'] for _ in target])
    if 'positive_negative' in self.outputItems:
        positive_negative,regression=
            self.getPositiveNegativeRegression(bboxes)
    result=[]
    for _ in self.outputItems:
        result.append(eval(_))
    return result
```