一，anchors 理解

所谓 `anchors`，实际上就是一组由 generate_anchors.py 生成的矩形框。其中每行的4个值 `(x1,y1,x2,y2)` 表矩形左上和右下角点坐标。9 个矩形共有 3 种形状，长宽比为大约为 `{1:1, 1:2, 2:1}` 三种, 实际上通过anchors就引入了检测中常用到的多尺度方法。`generate_anchors.py` 的代码如下:

```
import numpy as np
import six
from six import __init__  # 兼容python2和python3模块


def generate_anchor_base(base_size=16, ratios=[0.5, 1, 2],
                         anchor_scales=[8, 16, 32]):
    """Generate anchor base windows by enumerating aspect ratio and scales.

    Generate anchors that are scaled and modified to the given aspect ratios.
    Area of a scaled anchor is preserved when modifying to the given aspect
    ratio.

    :obj:`R = len(ratios) * len(anchor_scales)` anchors are generated by this
    function.
    The :obj:`i * len(anchor_scales) + j` th anchor corresponds to an anchor
    generated by :obj:`ratios[i]` and :obj:`anchor_scales[j]`.

    For example, if the scale is :math:`8` and the ratio is :math:`0.25`,
    the width and the height of the base window will be stretched by :math:`8`.
    For modifying the anchor to the given aspect ratio,
    the height is halved and the width is doubled.

    Args:
        base_size (number): The width and the height of the reference window.
        ratios (list of floats): This is ratios of width to height of
            the anchors.
        anchor_scales (list of numbers): This is areas of anchors.
            Those areas will be the product of the square of an element in
            :obj:`anchor_scales` and the original area of the reference
            window.

    Returns:
        ~numpy.ndarray:
        An array of shape :math:`(R, 4)`.
        Each element is a set of coordinates of a bounding box.
        The second axis corresponds to
        :math:`(x_{min}, y_{min}, x_{max}, y_{max})` of a bounding box.

    """
    import numpy as np
    py = base_size / 2.
    px = base_size / 2.

    anchor_base = np.zeros((len(ratios) * len(anchor_scales), 4),
                           dtype=np.float32)
    for i in six.moves.range(len(ratios)):
        for j in six.moves.range(len(anchor_scales)):
            h = base_size * anchor_scales[j] * np.sqrt(ratios[i])
            w = base_size * anchor_scales[j] * np.sqrt(1. / ratios[i])
```

```
            index = i * len(anchor_scales) + j
            anchor_base[index, 0] = px - w / 2.
            anchor_base[index, 1] = py - h / 2.

            anchor_base[index, 2] = px + h / 2.
            anchor_base[index, 3] = py + w / 2.
    return anchor_base


# test
if __name__ == "__main__":
    bbox_list = generate_anchor_base()
    print(bbox_list)
```

程序运行输出如下：

> [[-82.50967   -37.254833   53.254833   98.50967]
> [-173.01933   -82.50967   98.50967   189.01933]
> [-354.03867 -173.01933   189.01933   370.03867]
> [-56.   -56.   72.   72.   ]
> [-120.   -120.   136.   136.   ]
> [-248.   -248.   264.   264.   ]
> [-37.254833 -82.50967   98.50967   53.254833]
> [-82.50967 -173.01933   189.01933   98.50967]
> [-173.01933 -354.03867   370.03867   189.01933]]

二，交并比IOU

交并比（Intersection-over-Union，IoU），目标检测中使用的一个概念，是产生的候选框（candidate bound）与原标记框（ground truth bound）的交叠率，即它们的交集与并集的比值。最理想情况是完全重叠，即比值为1。计算公式如下：

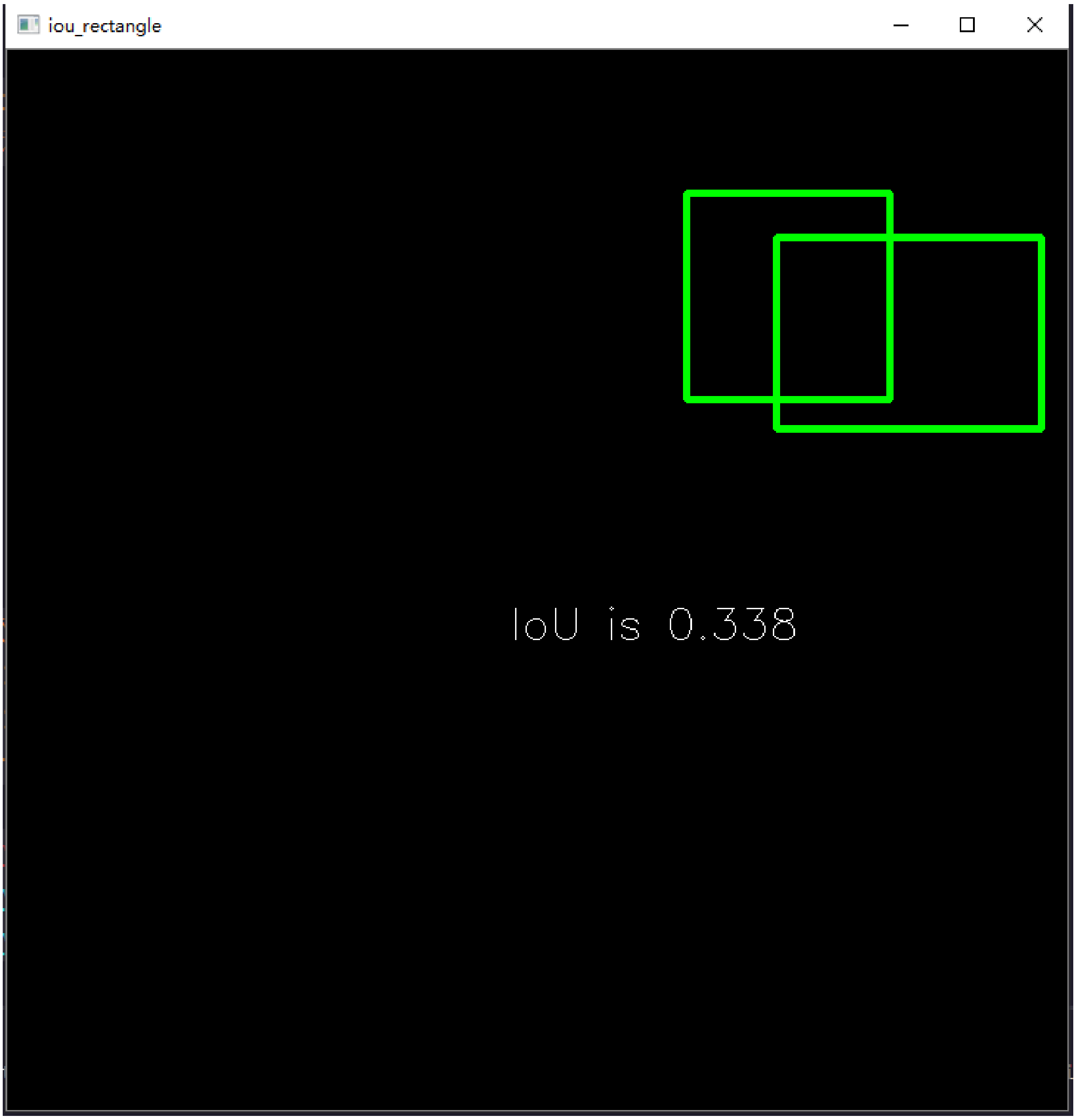
$$IoU = \frac{area(C) \cap area(G)}{area(C) \cup area(G)}$$

代码实现如下：

```
# _*_ coding:utf-8 _*_
# 计算iou

"""
bbox的数据结构为(xmin,ymin,xmax,ymax)--(x1,y1,x2,y2),
每个bounding box的左上角和右下角的坐标
输入：
    bbox1, bbox2: Single numpy bounding box, Shape: [4]
输出：
    iou值
"""
import numpy as np
import cv2

def iou(bbox1, bbox2):
    """
    计算两个bbox(两框的交并比)的iou值
    :param bbox1: (x1,y1,x2,y2), type: ndarray or list
    :param bbox2: (x1,y1,x2,y2), type: ndarray or list
    :return: iou, type float
    """
    if type(bbox1) or type(bbox2) != 'ndarray':
```

```
        bbox1 = np.array(bbox1)
        bbox2 = np.array(bbox2)

    assert bbox1.size == 4 and bbox2.size == 4, "bounding box coordinate size must be 4"
    xx1 = np.max((bbox1[0], bbox2[0]))
    yy1 = np.max((bbox1[1], bbox1[1]))
    xx2 = np.min((bbox1[2], bbox2[2]))
    yy2 = np.min((bbox1[3], bbox2[3]))
    bwidth = xx2 - xx1
    bheight = yy2 - yy1
    area = bwidth * bheight  # 求两个矩形框的交集
    union = (bbox1[2] - bbox1[0])*(bbox1[3] - bbox1[1]) + (bbox2[2] - bbox2[0])*(bbox2[3] - bbox2[1]) - area  # 求两个矩形框的并集
    iou = area / union

    return iou


if __name__=='__main__':
    rect1 = (461, 97, 599, 237)
    # (top, left, bottom, right)
    rect2 = (522, 127, 702, 257)
    iou_ret = round(iou(rect1, rect2), 3) # 保留3位小数
    print(iou_ret)

    # Create a black image
    img=np.zeros((720,720,3), np.uint8)
    cv2.namedWindow('iou_rectangle')
    """
    cv2.rectangle 的 pt1 和 pt2 参数分别代表矩形的左上角和右下角两个点,
    coordinates for the bounding box vertices need to be integers if they are in a tuple,
    and they need to be in the order of (left, top) and (right, bottom).
    Or, equivalently, (xmin, ymin) and (xmax, ymax).
    """
    cv2.rectangle(img,(461, 97),(599, 237),(0,255,0),3)
    cv2.rectangle(img,(522, 127),(702, 257),(0,255,0),3)
    font  = cv2.FONT_HERSHEY_SIMPLEX
    cv2.putText(img, 'IoU is ' + str(iou_ret), (341,400), font, 1,(255,255,255),1)
    cv2.imshow('iou_rectangle', img)
    cv2.waitKey(0)
```

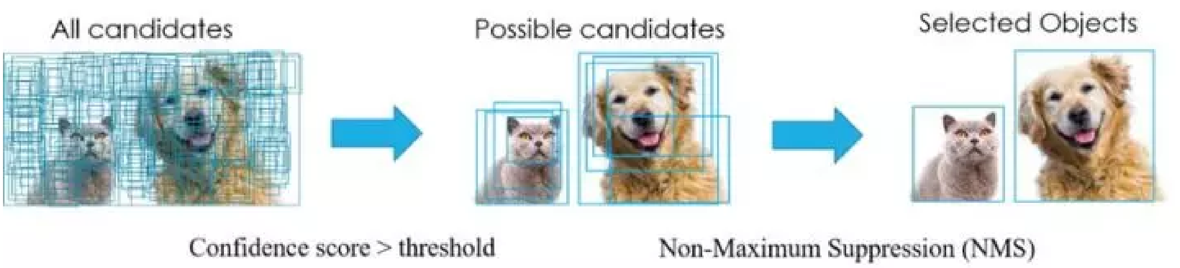
代码输出结果如下所示：



三，NMS 算法

NMS介绍

在目标检测中，常会利用非极大值抑制算法(NMS，non maximum suppression)对生成的大量候选框进行后处理，去除冗余的候选框，得到最佳检测框，以加快目标检测的效率。其本质思想是其思想是搜素局部最大值，抑制非极大值。非极大值抑制，在计算机视觉任务中得到了广泛的应用，例如边缘检测、人脸检测、目标检测（DPM，YOLO，SSD，Faster R-CNN）等。即如下图所示实现效果，消除多余的候选框，找到最佳的 `bbox`。`NMS过程` 如下图所示：



以上图为例，每个选出来的 `Bounding Box` 检测框（既BBox）用 `（x,y,h,w, confidence score，Pdog,Pcat）` 表示，`confidence score` 表示 `background` 和 `foreground` 的置信度得分，取值范围 `[0,1]`。Pdog, Pcat分布代表类别是狗和猫的概率。如果是 100 类的目标检测模型，`BBox` 输出向量为 `5+100=105`。

NMS算法

`NMS` 主要就是通过迭代的形式，不断地以最大得分的框去与其他框做 `IoU` 操作，并过滤那些 `IoU` 较大的框。

其实现的思想主要是将各个框的置信度进行排序，然后选择其中置信度最高的框 A，将其作为标准选择其他框，同时设置一个阈值，当其他框B与A的重合程度超过阈值就将 B 舍弃掉，然后在剩余的框中选择置信度最大的框，重复上述操作。算法过程如下：

1. 根据候选框类别分类概率排序：`F>E>D>C>B>A`，并标记最大概率的矩形框F作为标准框。
2. 分别判断 `A~E` 与 `F` 的重叠度 `IOU` (两框的交并比)是否大于某个设定的阈值，假设 `B、D` 与 `F` 的重叠度超过阈值，那么就扔掉 `B、D`；
3. 从剩下的矩形框 `A、C、E`中，选择概率最大的 `E`，标记为要保留下来的，然后判读 `E` 与 `A、C` 的重叠度，扔掉重叠度超过设定阈值的矩形框；
4. 对剩下的 `bbox`，循环执行(2)和(3)直到所有的 `bbox` 均满足要求（即不能再移除 `bbox`）

nms的python代码如下：

```
import numpy as np

def py_nms(dets, thresh):
    """Pure Python NMS baseline.注意，这里的计算都是在矩阵层面上计算的
    greedily select boxes with high confidence and overlap with current maximum <=
thresh
    rule out overlap >= thresh
    :param dets: [[x1, y1, x2, y2 score],] # ndarray, shape(-1,5)
    :param thresh: retain overlap < thresh
    :return: indexes to keep
    """
    # x1、y1、x2、y2、以及score赋值
    x1 = dets[:, 0]
    y1 = dets[:, 1]
    x2 = dets[:, 2]
    y2 = dets[:, 3]

    # 计算每一个候选框的面积, 纯矩阵加和乘法运算,为何加1?
    areas = (x2 - x1 + 1) * (y2 - y1 + 1)
    # order是将confidence降序排序后得到的矩阵索引
    order = np.argsort(dets[:, 4])[::-1]
    keep = []
    while order.size > 0:
        i = order[0]
        keep.append(i)
        # 计算当前概率最大矩形框与其他矩形框的相交框的坐标，会用到numpy的broadcast机制，得到的是向量
        xx1 = np.maximum(x1[i], x1[order[1:]])
        yy1 = np.maximum(y1[i], y1[order[1:]])
        xx2 = np.minimum(x2[i], x2[order[1:]])
        yy2 = np.minimum(y2[i], y2[order[1:]])

        # 计算相交框的面积,注意矩形框不相交时w或h算出来会是负数，用0代替
        w = np.maximum(0.0, xx2 - xx1 + 1)
        h = np.maximum(0.0, yy2 - yy1 + 1)
        inter = w * h
        # 计算重叠度IOU：重叠面积/（面积1+面积2-重叠面积）
        iou = inter / (areas[i] + areas[order[1:]] - inter)
        # 找到重叠度不高于阈值的矩形框索引
        inds = np.where(iou < thresh)[0]
        # 将order序列更新，由于前面得到的矩形框索引要比矩形框在原order序列中的索引小1，所以要把这个1
加回来
        order = order[inds + 1]
    return keep
```

```
# test
if __name__ == "__main__":
    dets = np.array([[30, 20, 230, 200, 1],
                     [50, 50, 260, 220, 0.9],
                     [210, 30, 420, 5, 0.8],
                     [430, 280, 460, 360, 0.7]])
    thresh = 0.35
    keep_dets = py_nms(dets, thresh)
    print(keep_dets)
    print(dets[keep_dets])
```

程序输出如下：

> [0, 2, 3]
> [[30. 20. 230. 200. 1.]
> [210. 30. 420. 5. 0.8]
> [430. 280. 460. 360. 0.7]]

另一个版本的 nms 的 python 代码如下：

```
from __future__ import print_function
import numpy as np
import time

def intersect(box_a, box_b):
    max_xy = np.minimum(box_a[:, 2:], box_b[2:])
    min_xy = np.maximum(box_a[:, :2], box_b[:2])
    inter = np.clip((max_xy - min_xy), a_min=0, a_max=np.inf)
    return inter[:, 0] * inter[:, 1]

def get_iou(box_a, box_b):
    """Compute the jaccard overlap of two sets of boxes.  The jaccard overlap
    is simply the intersection over union of two boxes.
    E.g.:
        A ∩ B / A ∪ B = A ∩ B / (area(A) + area(B) - A ∩ B)
        The box should be [x1,y1,x2,y2]
    Args:
        box_a: Single numpy bounding box, Shape: [4] or Multiple bounding boxes, Shape:
[num_boxes,4]
        box_b: Single numpy bounding box, Shape: [4]
    Return:
        jaccard overlap: Shape: [box_a.shape[0], box_a.shape[1]]
    """
    if box_a.ndim==1:
        box_a=box_a.reshape([1,-1])
    inter = intersect(box_a, box_b)
    area_a = ((box_a[:, 2]-box_a[:, 0]) *
              (box_a[:, 3]-box_a[:, 1]))  # [A,B]
    area_b = ((box_b[2]-box_b[0]) *
              (box_b[3]-box_b[1]))  # [A,B]
    union = area_a + area_b - inter
    return inter / union  # [A,B]

def nms(bboxs,scores,thresh):
    """
    The box should be [x1,y1,x2,y2]
    :param bboxs: multiple bounding boxes, Shape: [num_boxes,4]
    :param scores: The score for the corresponding box
    :return: keep inds
```

```
    """
    if len(bboxs)==0:
        return []
    order=scores.argsort()[::-1]
    keep=[]
    while order.size>0:
        i=order[0]
        keep.append(i)
        ious=get_iou(bboxs[order],bboxs[i])
        order=order[ious<=thresh]
    return keep
```

四，Soft NMS算法

Soft NMS算法是对NMS算法的改进，是发表在ICCV2017的文章中提出的。`NMS`算法存在一个问题是可能会把一些目标框给过滤掉，从而导致目标的`recall`指标比较低。原来的NMS可以描述如下：将IOU大于阈值的窗口的得分全部置为0，计算公式如下：

$$s_i = \begin{cases} s_i, & \text{iou}(\mathcal{M}, b_i) < N_t \\ 0, & \text{iou}(\mathcal{M}, b_i) \geq N_t \end{cases},$$

文章的改进有两种形式，一种是`线性加权`的。设 si 为第 i 个 box 的 score, 则在应用 SoftNMS 时各个 box score 的计算公式如下：

$$s_i = \begin{cases} s_i, & \text{iou}(\mathcal{M}, b_i) < N_t \\ s_i(1 - \text{iou}(\mathcal{M}, b_i)), & \text{iou}(\mathcal{M}, b_i) \geq N_t \end{cases},$$

另一种是`高斯加权`的，高斯惩罚系数(与上面的线性截断惩罚不同的是, 高斯惩罚会对其他所有的 box 作用)，计算公式图如下：

$$s_i = s_i e^{-\frac{\text{iou}(\mathcal{M}, b_i)^2}{\sigma}}, \forall b_i \notin \mathcal{D}$$

注意，这两种形式，思想都是M为当前得分最高框，b_i为待处理框，b_i和M的IOU越大，bbox的得分s_i就下降的越厉害(N_t为给定阈值)。更多细节可以参考原论文
soft nms 的 python 代码如下：

```
def soft_nms(dets, thresh, type='gaussian'):
    x1 = dets[:, 0]
    y1 = dets[:, 1]
    x2 = dets[:, 2]
    y2 = dets[:, 3]
    scores = dets[:, 4]

    areas = (x2 - x1 + 1) * (y2 - y1 + 1)
    order = scores.argsort()[::-1]
    scores = scores[order]

    keep = []
    while order.size > 0:
        i = order[0]
        dets[i, 4] = scores[0]
```

```
        keep.append(i)

        xx1 = np.maximum(x1[i], x1[order[1:]])
        yy1 = np.maximum(y1[i], y1[order[1:]])
        xx2 = np.minimum(x2[i], x2[order[1:]])
        yy2 = np.minimum(y2[i], y2[order[1:]])

        w = np.maximum(0.0, xx2 - xx1 + 1)
        h = np.maximum(0.0, yy2 - yy1 + 1)
        inter = w * h
        # 计算重叠度IOU：重叠面积/（面积1+面积2-重叠面积）
        ovr = inter / (areas[i] + areas[order[1:]] - inter)

        order = order[1:]
        scores = scores[1:]
        if type == 'linear':
            inds = np.where(ovr >= thresh)[0]
            scores[inds] *= (1 - ovr[inds])
        else:
            scores *= np.exp(- ovr ** 2 / thresh)
        inds = np.where(scores > 1e-3)[0]
        order = order[inds]
        scores = scores[inds]

        tmp = scores.argsort()[::-1]
        order = order[tmp]
        scores = scores[tmp]
    return keep
```

五，目标检测领域中的数据不均衡问题

参考此论文Imbalance Problems in Object Detection，我的理解之后补充。

参考资料

- NMS介绍
- Faster RCNN 源码解读(2) -- NMS(非极大抑制)