西北大学

智能信息系统综合实践

实验报告

题　　目：　目标检测

年　　级：　2020 级

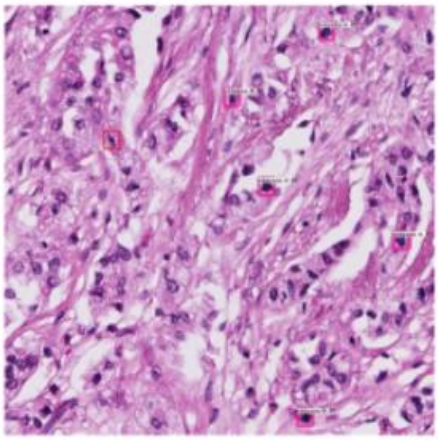
专　　业：　软件工程

学　　号：　2020118035

姓　　名：　王凯

一、题目

有丝分裂细胞检测



提供数据集
训练集：313张图像
测试集：80张图像
标注格式：.xml，可再提供xml转为txt/csv的代码
（这三种格式几乎支持开源你的目标检测程序）

代码：
提供Faster RCNN代码（包含数据预处理、训练、测试、指标计算），tensorflow

目标：
请在测试集中达到更好的结果
AP，recall

二、解题步骤

【算法分析】

1. 目标检测任务是找出图像中所有感兴趣的目标（物体），确定它的类别和位置，是计算机视觉领域的核心问题之一。基于深度学习的目标检测算法主要分为两类：Two stage 和 One stage。前者先进行区域生成，再通过卷积神经网络进行样本分类，常见的算法有 R-CNN、SPP-Net、Fast R-CNN、Faster R-CNN 和 R-FCN 等；而后者不用区域生成，直接在网络中提取特征来预测物体分类和位置，常见的算法有 YOLO 系列、SSD 等。

2. 在本实验中我选择使用 Faster R-CNN 模型。Fast R-CNN 中使用 Selective Search 进行区域生成，耗时较大，而 Faster R-CNN 使用 RPN（建议区域生成）网络进行改进，大大减少区域生成的时间。

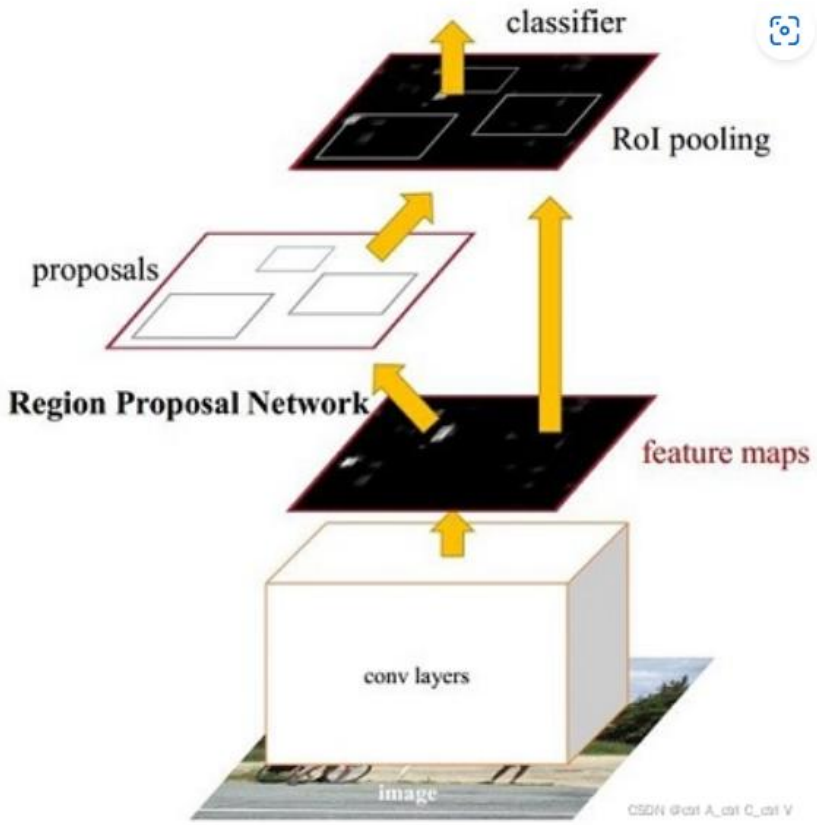
3. 算法流程：

（1）将图像输入网络得到相应的特征图。

（2）使用 RPN 网络生成候选框，将 RPN 生成的候选框投影到特征图上获得 ROI 区域的特征矩阵。

（3)将每个 ROI 区域的特征矩阵通过 ROI pooling 层缩放到 7x7 大小的特征图，接着将特征图展平为 vector，之后通过一系列全连接层得到预测结果。

4. Faster RCNN 的基本结构：



5. Faster RCNN 的基本架构

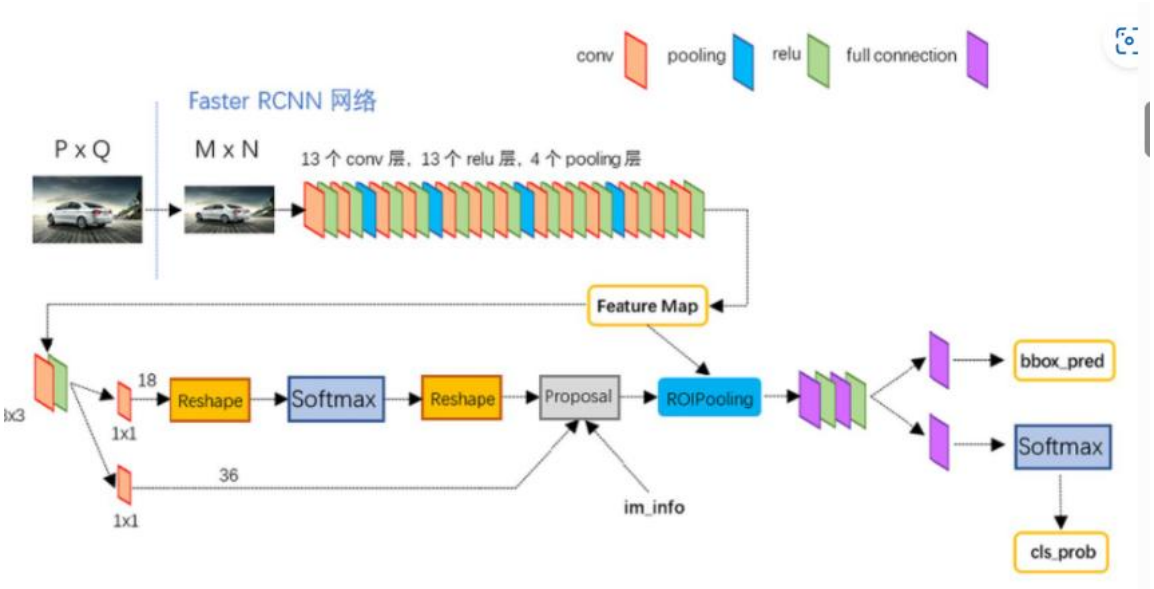
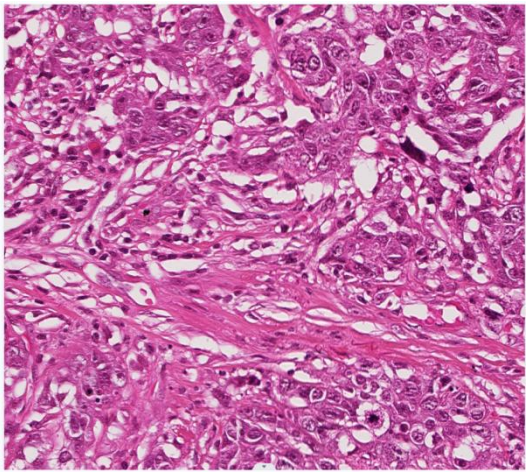


图2 faster_rcnn_test.pt网络结构 （pascal_voc/VGG16/faster_rcnn_alt_opt/faster_rcnn_test.pt)

【数据集分析】

1. 本次实验所给数据集为有丝分裂细胞数据集，包含训练集和测试集，训练集中有 313 个样本，测试集中有 80 个样本，每个样本包含一个 img 图像文件和一个 xml 注释文件。



```
<object>
 <name>mitosis</name>
 <pose>Unspecified</pose>
 <truncated>0</truncated>
 <difficult>0</difficult>
 <bndbox>
  <xmin>834</xmin>
  <ymin>938</ymin>
  <xmax>894</xmax>
  <ymax>998</ymax>
 </bndbox>
</object>
```

2. 每个图像尺寸为 3*1663*1485，其中含有多个细胞，本次实验目标就是识别其中的有丝分裂细胞。

3. xml 注释文件是对应图像中标注物体的信息，包括边界框位置、类别标签等，因为本次实验任务是识别图像中的有丝分裂细胞，目标物体只有一类，所以我们主要关注其中的<bndbox>标签，它里边的四个属性构成了有丝分裂细胞的边界框。

【解题思路】（代码有点多，只显示函数申明和部分关键代码）

1. 自定义一个 CellDataset 类，继承 Dataset 类，修改其中的__len__和__getitem__方法，用来表示有丝分裂细胞数据集。

```
class CellDataset(torch.utils.data.Dataset):
    def __init__(self, root, phase):
        self.root = root
        self.phase = phase
        self.images = []
        self.annotations = []
```

```
    def __len__(self):
        return len(self.images)
```

```
    def __getitem__(self, idx):
        # 读取图像和注释文件
        image_path = self.images[idx]
        annotation_path = self.annotations[idx]
        image = Image.open(image_path)
        # 将图像转换为 PyTorch 张量
        transform = transforms.ToTensor()
        image = transform(image)
```

2. 定义模型训练的超参数，因为**本次实验中目标物体只有一类，所以设计的 num_classes=2，对应 mitosis 和背景**。使用自定义的 CellDataset 表示细胞数据集，然后使用创建对应的 train_loader 和 test_loader，对训练集和测试集批量处理。创建 Faster RCNN 模型和 SGD 优化器并对模型进行训练，本实验使用 torchvision.models.detection 中预训练的 fasterrcnn_resnet50_fpn 模型，减少训练时间，且能减弱数据集较小对模型性能的影响。（模型只训练了一次，训练了 50 个 epoch，并未进行参数调优，故模型性能一般，后续得到的指标并不高）

```
# 定义一些超参数
dataset_root = "E:/Data/object detection/data"
num_classes = 2 # mitosis和背景
batch_size = 1
num_epochs = 10
lr = 0.005

# 加载数据集
train_dataset = CellDataset(dataset_root, "train")
test_dataset = CellDataset(dataset_root, "test")
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=Fa
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=Fals

# 创建模型和优化器
model = get_model(num_classes)
optimizer = torch.optim.SGD(model.parameters(), lr=lr, momentum=0.9, weigh

# 设置设备
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
```

```
# 训练模型
start_time = time.time()
num_epochs = 50
for epoch in range(num_epochs):
    model.train()
    for images, targets in train_loader:
        images = list(image.to(device) for image in images)
        targets = [{k: v.to(device) for k, v in t.items()} for t in targe

        optimizer.zero_grad()
        loss_dict = model(images, targets)
        losses = sum(loss for loss in loss_dict.values())
        losses.backward()
        optimizer.step()
    print(f'Epoch {epoch + 1}, Loss: {losses.item()}')
end_time = time.time()

torch.save(model.state_dict(), "cell_detection_model_2.pth")
```

3. 利用训练模型对单张图片 H03_00Ad.png 进行预测并绘制边框。使用 cv2.imread()对应路径下的读取图像文件，使用训练的模型对图片进行预测得到对应的 output，该输出包含对该图片预测的物体对应的的边界框 boxes、置信度 scores 以及标签 labels。使用 output[‘boxes’]得到所有边界框，使用 cv2.rectangle 绘制那些 score>0.5 的边界框（score 大于 0.5 的视为正样本，反之则为负样本），并输出标签和置信度，最后保存含边界框的图像。

```
# 对图像进行预测，并绘制边界框，显示类别和置信度
def predict_draw_boxes(path, name):
```

```
        boxes = output['boxes'].cpu().detach().numpy()
        scores = output['scores'].cpu().detach().numpy()
        labels = output['labels'].cpu().detach().numpy()
```

```
        for j in range(len(boxes)):
            box = boxes[j]
            score = scores[j]
            label = labels[j]

            if score < 0.5:
                continue

            x1, y1, x2, y2 = box
            cv2.rectangle(image, (int(x1), int(y1)), (int(x2), int(y2)), (

            label_str = f"mitosis,{score:.2f}"
            cv2.putText(image, label_str, (int(x1), int(y1) - 10), cv2.FON

        cv2.imwrite(f"{name}_with_rectangle.jpg", image)
```

```
predict_draw_boxes("E:/Data/object detection/data/test/img/H03_00Ad.png",
```

```
<bndbox>
 <xmin>1320</xmin>
 <ymin>1266</ymin>
 <xmax>1380</xmax>
 <ymax>1326</ymax>
</bndbox>
</object>
```

```
<bndbox>
 <xmin>178</xmin>
 <ymin>1258</ymin>
 <xmax>238</xmax>
 <ymax>1318</ymax>
</bndbox>
```

```
<bndbox>
 <xmin>1510</xmin>
 <ymin>188</ymin>
 <xmax>1570</xmax>
 <ymax>248</ymax>
</bndbox>
```

图 1 H03_00Ad.png 的真实边界框

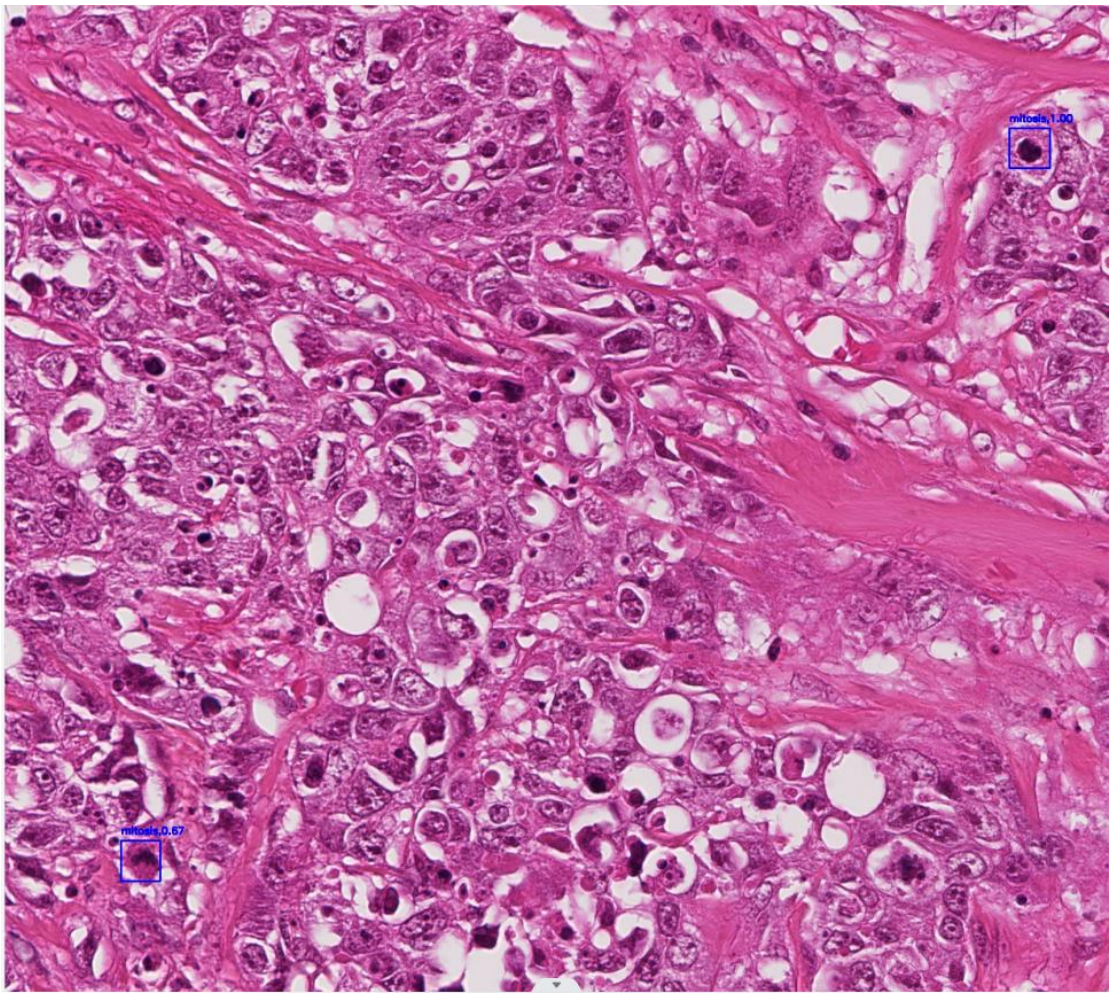


图 2 H03_00Ad.png 的预测结果

对比图 1 和图 2，我们可以看出三个真实边界框，模型预测出了后两个边界框，且位置较准确，第一个边界框模型没有预测出，这可能是因为数据集太小，模型训练不够充分造成的。

4. 计算 Precision 和 recall，对模型进行评估。使用训练的模型对测试集进行

预测，设置交并比阈值 iou_threshold 和置信度阈值 score_threshold，图片预测结果中 score >= score_threshold 的边界框视为正例，反之为负例。因为**precision 是预测的正例中真正的正例的占比，而 recall 为真正的正例中预测出的正例的占比，二者一个从预测结果出发，一个从真实值出发**，故从两个角度分别计算：

对每个预测框，求它与所有真实框的交并比，取其中的最大值为 max_iou。按下述规则计算 precision=TP/(TP+FP)：

(1) 若 score >= score_threshold and max_iou >= iou_threshold，则该预测框为真阳 TP；

(2) 若 score >= score_threshold and max_iou < iou_threshold，则该预测框为假阳 FP.

对每个真实框，求它与所有预测框的交并比，取其中的最大值为 max_iou。按下述规则计算 recall=TP/(TP+FN)：

(3) 若 score >= score_threshold and max_iou >= iou_threshold，则该预测框为真阳 TP；

(4) 若 score < score_threshold and max_iou >= iou_threshold，则该预测框为假阳 FN.

```
# 对测试集进行预测
preds = []
targs = []
model.eval()
i = 0
with torch.no_grad():
    for images, targets in test_loader:
        if i==4:
            break
        images = [image.to(device) for image in images]
        targets = [{k: v.to(device) for k, v in t.items()} for t in targ
        outputs = model(images)
        preds.append(outputs[0])
        targs.append(targets[0])
```

```python
def calculate_iou(boxes1, boxes2):
    """
    计算两组框之间的IoU值

    参数:
    - boxes1: 第一组框，形状为(N, 4)，其中N表示框的数量，每个框由(xmin, ymi
    - boxes2: 第二组框，形状为(M, 4)，其中M表示框的数量，每个框由(xmin, ymi

    返回:
    - ious: IoU值，形状为(N, M)，表示第一组框和第二组框之间的IoU值
    """
    # 计算两组框的面积
    area_boxes1 = (boxes1[:, 2] - boxes1[:, 0]) * (boxes1[:, 3] - boxes1[:
    area_boxes2 = (boxes2[:, 2] - boxes2[:, 0]) * (boxes2[:, 3] - boxes2[:

    # 计算两组框的交集的坐标
    x_min = torch.max(boxes1[:, 0].unsqueeze(1), boxes2[:, 0].unsqueeze(0))
    y_min = torch.max(boxes1[:, 1].unsqueeze(1), boxes2[:, 1].unsqueeze(0))
    x_max = torch.min(boxes1[:, 2].unsqueeze(1), boxes2[:, 2].unsqueeze(0))
    y_max = torch.min(boxes1[:, 3].unsqueeze(1), boxes2[:, 3].unsqueeze(0))

    # 计算交集的面积
    intersection_area = torch.clamp(x_max - x_min, min=0) * torch.clamp(y_

    # 计算并集的面积
    union_area = area_boxes1.unsqueeze(1) + area_boxes2.unsqueeze(0) - int

    # 计算IoU值
    ious = intersection_area / union_area
    return ious.cpu()
```

```python
def compute_precision_recall(predictions, targets, score_threshold=0.5, io
    """
    计算查准率和召回率

    参数:
    - predictions: 预测结果，包括检测框坐标、类别标签和置信度
    - targets: 标注信息，包括真实框坐标和类别标签
    - score_threshold: 得分阈值，大于或等于阈值，预测为正样本，反之为负样本
    - iou_threshold: IoU（Intersection over Union）阈值，用于判断检测结果是

    返回:
    - precision: 查准率
    - recall: 召回率
    """
```

```
    # 遍历每个预测结果，计算precision
    for prediction,target in zip(predictions,targets):
        predicted_boxes = prediction['boxes']  # 预测框坐标
        scores = prediction['scores']
        ious = calculate_iou(predicted_boxes, target['boxes'])
        # 根据score_threshold和iou_threshold计算TP，FP，FN
        for j in range(len(predicted_boxes)):
            # 预测为正样本，实际为正样本，TP++
            if np.max(ious[j].numpy()) >= iou_threshold and scores[j] >=
                true_positives += 1
            # 预测为正样本，实际为负样本，FP++
            elif np.max(ious[j].numpy()) < iou_threshold and scores[j] >=
                false_positives += 1
```

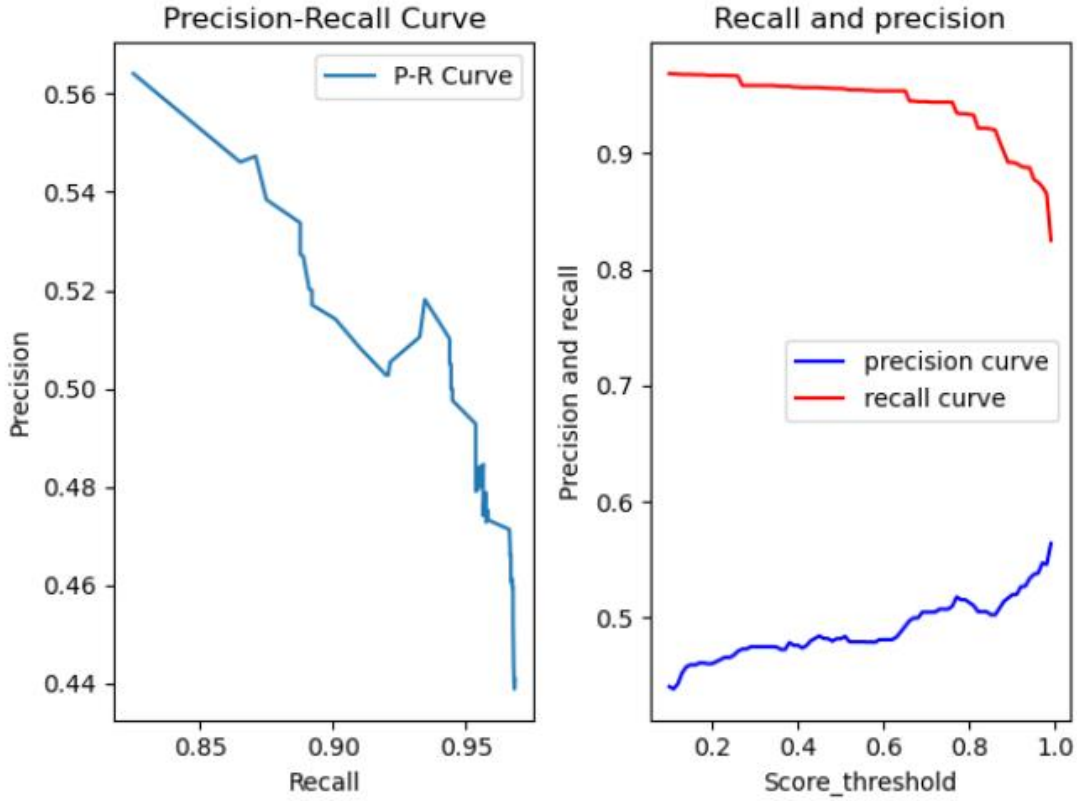
```
    # 遍历每个真实框，计算recall
    for prediction,target in zip(predictions,targets):
        true_boxes = target['boxes']  # 预测框坐标
        scores = prediction['scores']
        ious = calculate_iou(true_boxes, prediction['boxes'])
        # 根据score_threshold和iou_threshold计算TP，FP，FN
        for j in range(len(true_boxes)):
            # 预测为正样本，实际为正样本，TP++
            a = ious[j].numpy()
            if a.size==0:
                continue
            indice = np.argmax(a)
            if a[indice] >= iou_threshold and scores[indice] >= score_thre
                true_positives += 1
            # 预测为负样本，实际为正样本
            elif a[indice] < iou_threshold and scores[indice] < score_thre
                false_negatives += 1
```

5. 创建一组从0.1到1,间隔为0.01的数据作为置信度阈值 score_thresholds，默认交并比阈值 iou_threshold 为0.5,利用 compute_precision_recall()计算在不同置信度阈值下的 precision 和 recall，绘制查准率曲线、查全率曲线以及查准率-查全率曲线。

```
def plot_pr_curves(preds, targs):
    """
    计算不同得分阈值对应的P和R值，绘制查准率曲线、查全率率曲线和查准率-查全

    参数：
    - preds: 是对每个图片的预测值列表，包括边框，得分，标签
    - targs: 是每个图片的真实值列表

    返回：
    - pres: 查准率列表
    - recs: 查全率列表
    """
    # 得分阈值列表
    score_thresholds = np.arange(0.1, 1, step=0.01).tolist()
```

```
    # 调整得分阈值，计算对应查准率和查全率
    pres = []
    recs = []
    for score_threshold in score_thresholds:
        pre, rec = compute_precision_recall(preds, targs, score_threshold,
        pres.append(pre)
        recs.append(rec)

    # 创建画布
    fig, axes = plt.subplots(1,2)

    # 绘制P-R曲线
    axes[0].plot(recs, pres, label='P-R Curve')

    # 设置图例、坐标轴名称和标题
    axes[0].set_xlabel('Recall')
    axes[0].set_ylabel('Precision')
    axes[0].set_title('Precision-Recall Curve')
    axes[0].legend()

    # 绘制查准率曲线和查全率曲线
    axes[1].plot(score_thresholds, pres, 'b-', label='precision curve')
    axes[1].plot(score_thresholds, recs, 'r-', label='recall curve')
    axes[1].set_title('Recall and precision')
    axes[1].set_xlabel('Score_threshold')
    axes[1].set_ylabel('Precision and recall')
```

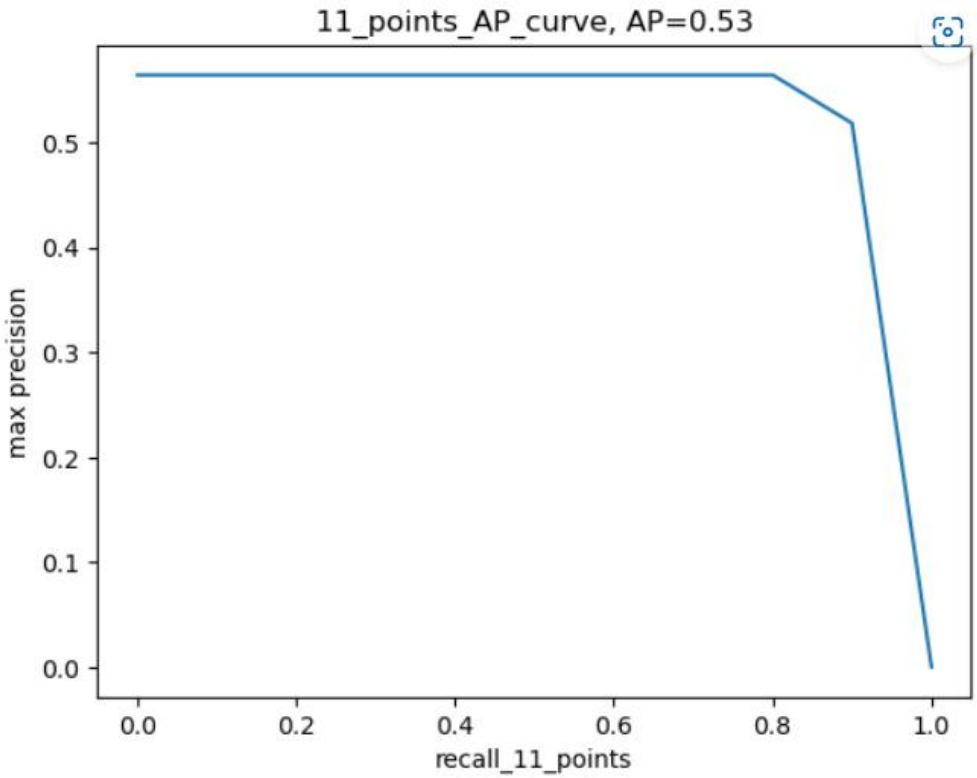
```
pres, recs = plot_pr_curves(preds, targs)
```



根据上图，我们可以观察到，**recall 和 precision 大体上呈反比，这是因为要想 recall 高，就需要增加预测框或者提升模型性能，而预测框增多，precision 自然会降低，同理 precision 高，recall 也会降低。随着 score_threshold 阈值的增大，预测结果中的正例减小，recall 变小，而预测的正例中真正的正例增多，precision 增大**。模型的 recall 较大，为 0.9，而 precision 只有 0.5，这可能是因为模型训练不够充分，预测的边界框数量太多，囊括了真实框，故 recall 大，但是 precision 却比较小。

6. 调整阈值得到的 precision 列表和 recall 列表计算 AP。这里我们采用 11 点法计算 AP，设置从 0 到 1 间隔为 0.1 的 11 各点作为 recall 阈值，对于每个 recall 阈值 recall_threshold，取 recall 列表中大于等于该阈值的 racall 中 precision 的最大值。绘制 recall_threshold 和对应 max_precision 曲线，利用 torch.trapz()求曲线和坐标轴围成的面积，即为 AP.

```
# 计算ap值
def compute_ap(precisions, recalls):
    pres = np.array(precisions)
    recs = np.array(recalls)

    rec_thresholds = np.linspace(0, 1, 11)
    pres_list = []

    for rec_threshold in rec_thresholds:
        indices = pres[np.where(recs>=rec_threshold)]
        if indices.size==0:
            pres_list.append(0)
        else:
            pres_list.append(np.max(indices))

    plt.plot(rec_thresholds, pres_list)
    ap = torch.trapz(torch.tensor(pres_list), torch.tensor(rec_thresholds))
    plt.title(f"11_points_AP_curve, AP={ap:.2f}")
    plt.xlabel('recall_11_points')
    plt.ylabel('max precision')
```



计算可得该模型的 AP 值为 0.53，并不高，这可能是因为数据集较小，且模型只训练了一次，并未进行参数调优。

三、总结

1. 训练所得模型的 precision 和 AP 值都不高，这可能是因为训练集较小，只有 313 张图，且模型只训练了一次，并未进行参数调优，导致模型预测的边界框太多，precision 较低，从而使得计算的 AP 也比较小。

2. 在计算 recall 和 precision 时，要注意 precision 是预测的正例中真正的正例的占比，而 recall 为真正的正例中预测出的正例的占比，二者一个从预测结果出发，一个从真实值出发。

3. 模型的 recall 和 precision 大体上呈反比，这是因为要想 recall 高，就需要增加预测框或者提升模型性能，而预测框增多，precision 自然会降低，同理 precision 高，recall 也会降低。随着 score_threshold 阈值的增大，预测结果中的正例减小，recall 变小，而预测的正例中真正的正例增多，precision 增大。