**智能信息系统综合实践**

**实验报告**

|  |  |
| --- | --- |
| **题 目：** | 目标检测 |
| **年 级：** | **2021** |
| **专 业：** | **软件工程** |
| **学 号：** | **2021117405** |
| **姓 名：** | **孙潇桐** |

目录

[1. 题目 3](#_Toc164629696)

[2. 解题步骤 3](#_Toc164629697)

[2.1. 训练Faster RCNN 3](#_Toc164629698)

[2.1.1. 关于Faster RCNN 3](#_Toc164629699)

[2.2. 对最佳模型的分析 6](#_Toc164629700)

[2.2.1. 关于最佳模型的RP曲线 6](#_Toc164629701)

[2.2.2. 在测试集上的一些结果 6](#_Toc164629702)

[3. 总结 8](#_Toc164629703)

[4. 附件 9](#_Toc164629704)

[4.1. 完整代码 9](#_Toc164629705)

[4.1.1. 导入训练数据并记录每一个epoch的信息：train.py 9](#_Toc164629706)

[4.1.2. 定义每个epoch具体如何训练：engine.py 12](#_Toc164629707)

[4.1.3. 继承torch的dataloader类以导入数据：data\_utils.py 16](#_Toc164629708)

[4.1.4. 计算模型Precision，Recall和AP的评估工具：test.py 23](#_Toc164629709)

[4.1.5. 配置文件（记录类名，学习率，数据集位置）：config.py 27](#_Toc164629710)

[4.2. 参考资料 29](#_Toc164629711)

# 题目

在提供的数据集icpr2014上训练Faster RCNN模型，随后在测试集上对模型进行测试。使得最终的模型AP和Recall尽可能的高。

# 解题步骤

## 训练Faster RCNN

### 关于Faster RCNN

我本次实验使用的模型为torchvision库提供的FasterRCNN\_ResNet50\_FPN，这是一个使用FPN优化的Faster RCNN模型，跟老师给的代码中使用的模型差不多。

Faster RCNN[1]是在之前的RCNN和Fast RCNN模型的基础上进行改进而来的。主要包括两个模块:

* 区域proposals网络(Region Proposal Network, RPN)：

RPN是一个全卷积网络，用于高效地生成候选目标边界框。这个模块接受一整张图像的输入，在图像的每个位置上滑动一个小窗口，并为每个窗口预测其是否包含了一个目标物体以及这个目标物体的大致位置。

* 目标检测网络：

检测网络负责在RPN生成的proposals中，准确地检测目标物体的类别和精确边界框位置。它首先使用RoIPooling层从卷积特征中提取固定大小的特征图，然后将其输入到两个全连接层中，一个分支用于目标分类，另一个分支用于边界框回归。

FPN优化是对Faster RCNN的一个非常重要的优化，现在已经应用于很多模型，torchvision中已经没有不使用FPN的Faster RCNN模型。在传统的Faster RCNN中，使用卷积网络的最终特征图来预测目标建议区域和目标分类回归。但最终特征图的分辨率较低，对小目标的检测效果不佳。为了解决这个问题，FPN利用不同级别的特征图，构建一个包含多尺度特征金字塔的特征层级，提高小目标的检测能力。通过逐层采样的方式，向上层提供更加丰富的信息，达到增加目标精度的目的。

* + 1. 开始训练

这次我使用的模型是经过预训练的，只需要再经过简单的训练之后就可以完成指定的任务了。我的超参数是120个epoch，0.001的学习率。因为这次的分类任务比较简单，为了防止过拟合，我同时保存了最新模型和最佳模型。我是在完成每一个epoch之后对模型在测试集上的AP进行一个计算，然后和之前的到的最佳模型比较之后判断是否覆盖最佳的模型。

在训练的过程中我保存了下面这些数据，并绘制了训练时的图像：

1. train\_loss\_list: 这个列表保存了每一个训练batch的总损失值(total loss)。

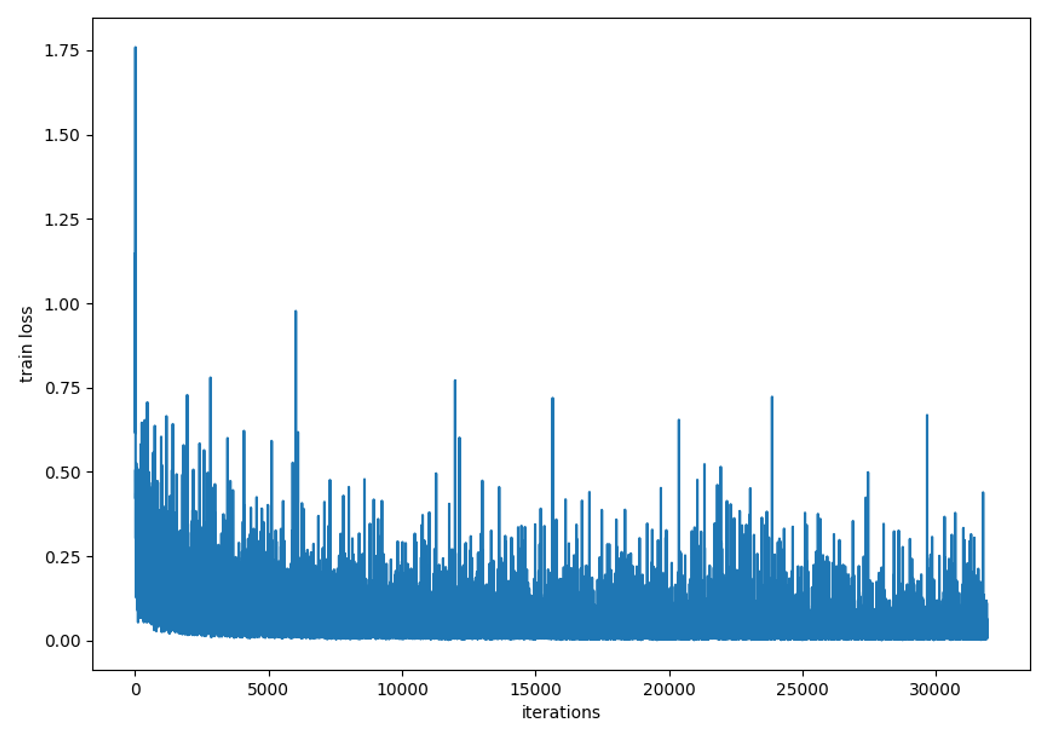


Figure 1 迭代次数的损失

1. loss\_cls\_list: 这个列表保存了每一个训练batch的分类损失(classification loss)。

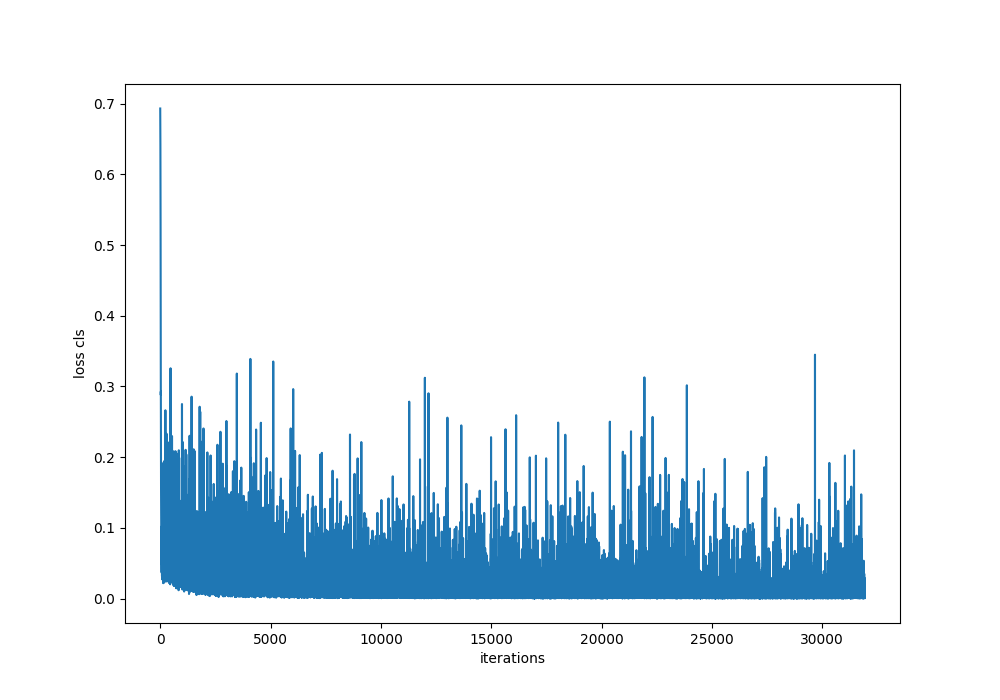


Figure 2 分类损失

1. loss\_box\_reg\_list: 这个列表保存了每一个训练batch的边界框回归损失(bounding box regression loss)。边界框回归损失衡量了预测的边界框与真实边界框之间的差异。

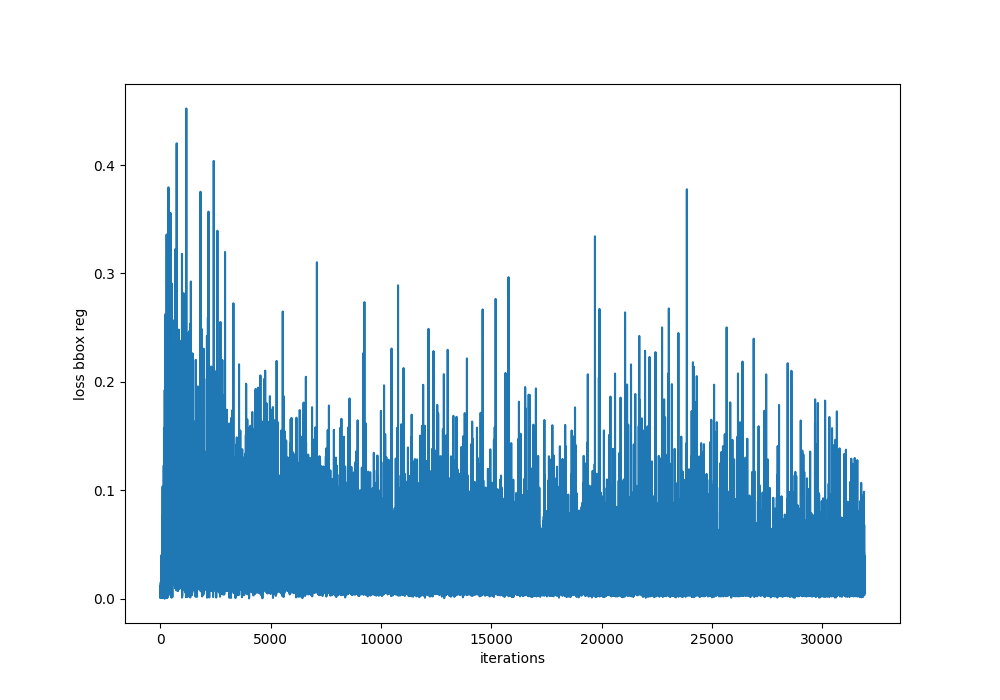


Figure 3 边界框回归损失

1. loss\_objectness\_list: 这个列表保存了每一个训练batch的**前景/背景**分类损失(objectness loss)。这部分损失来自于上面提到的RPN中判断是否含有目标物体的分类任务。

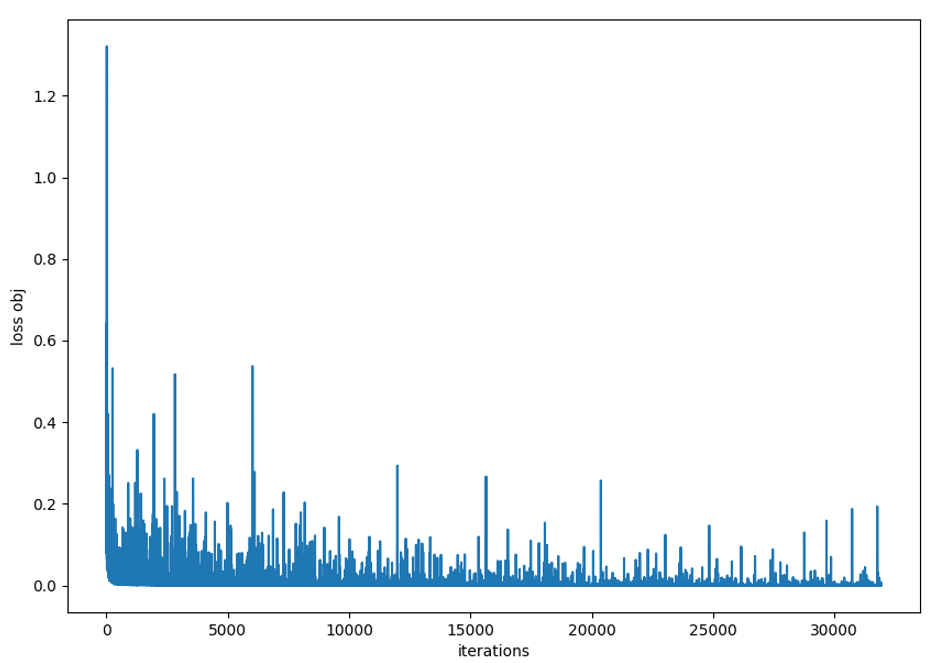


Figure 4 前景/背景分类损失

1. loss\_rpn\_bbox\_list: 这个列表保存了每一个训练batch的 RPN 网络预测的候选区域框与真实边界框之间的回归损失值。

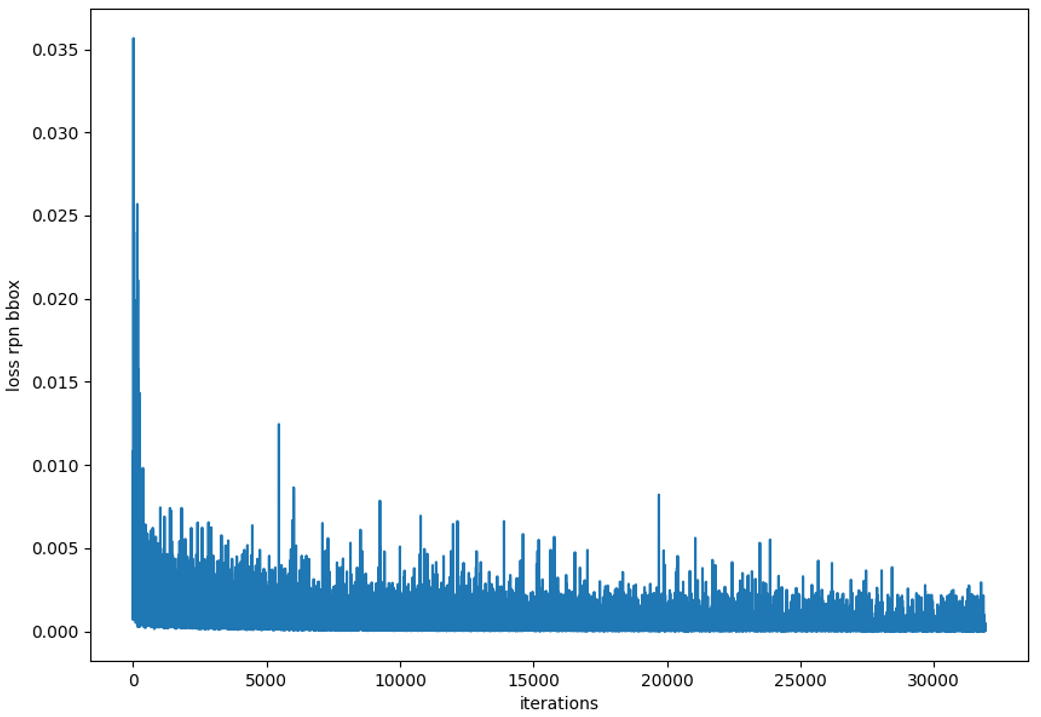


Figure 5 候选框/真实框损失

1. mAP\_list: 这个列表保存了在验证集上计算得到的每一个epoch的平均精度(mean Average Precision, mAP)。

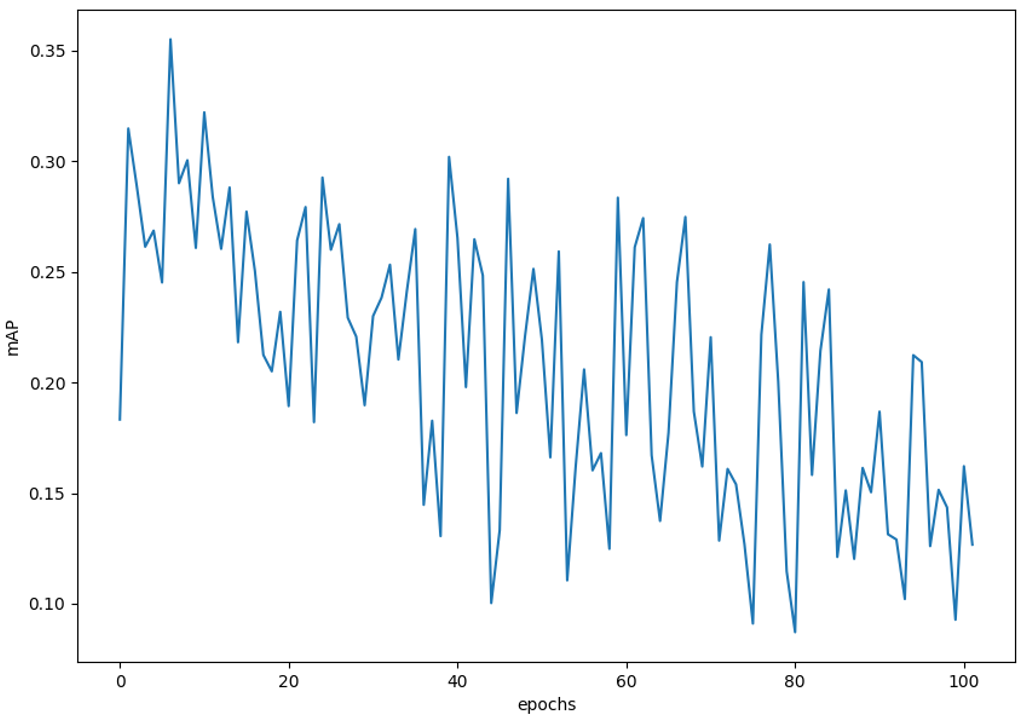


Figure 6 训练过程中的mAP

* + 1. 对结果的总结

可以从图6的mAP变化曲线看出来，最佳的模型大概在第六个epoch出现。后面很明显出线了过拟合的情况，mAP稳步降低。由此可见这个分类比较简单，下面我就展示一些我的分类成果。

## 对最佳模型的分析

### 关于最佳模型的RP曲线

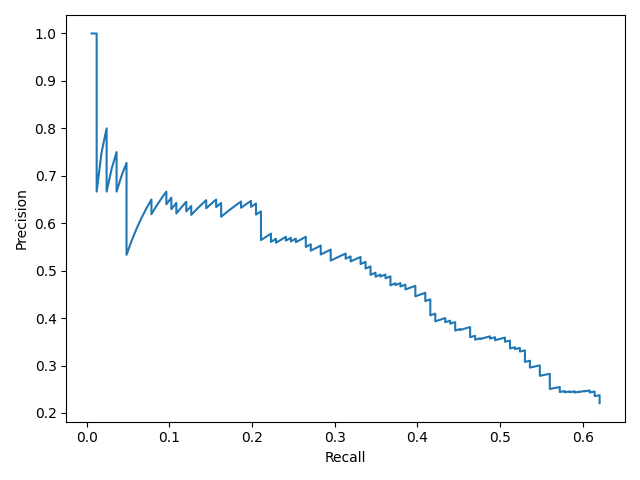


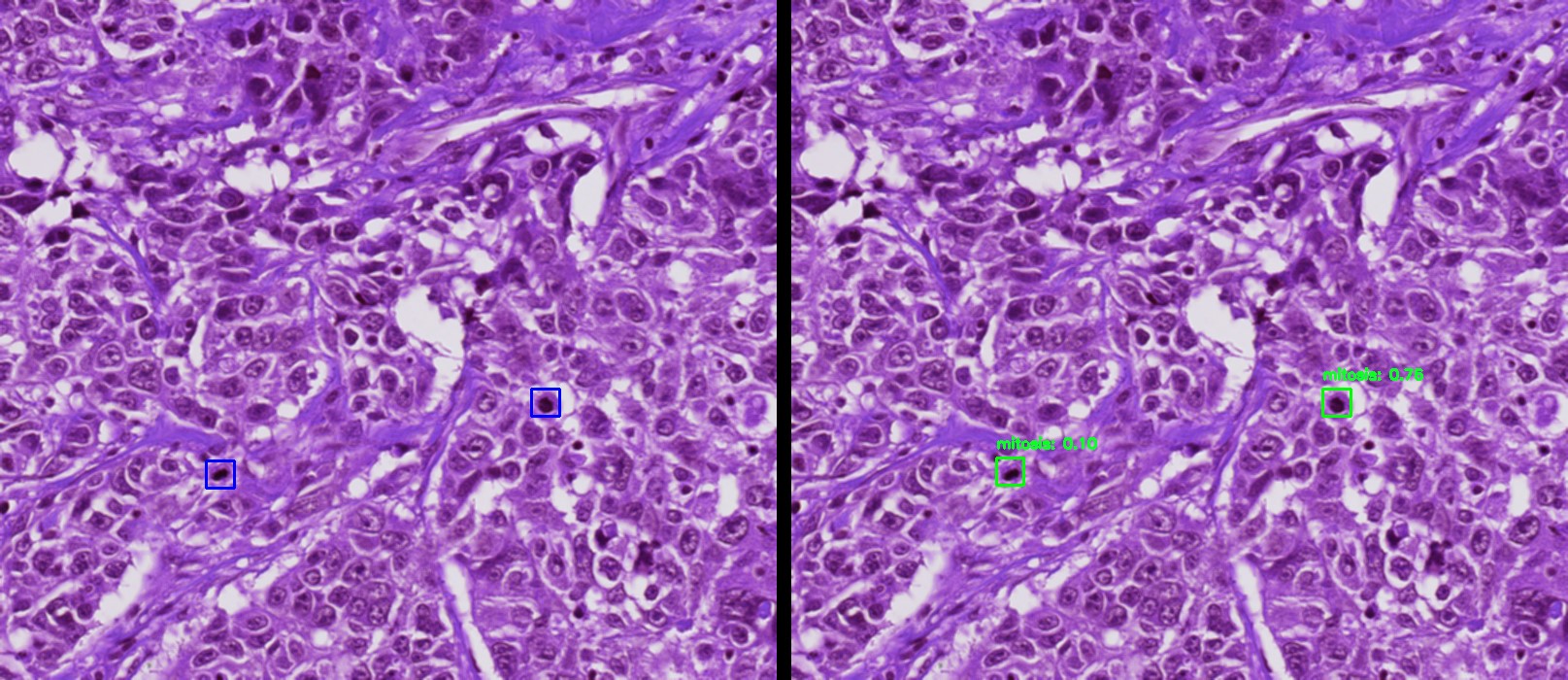
Figure 7 最佳模型的PR曲线

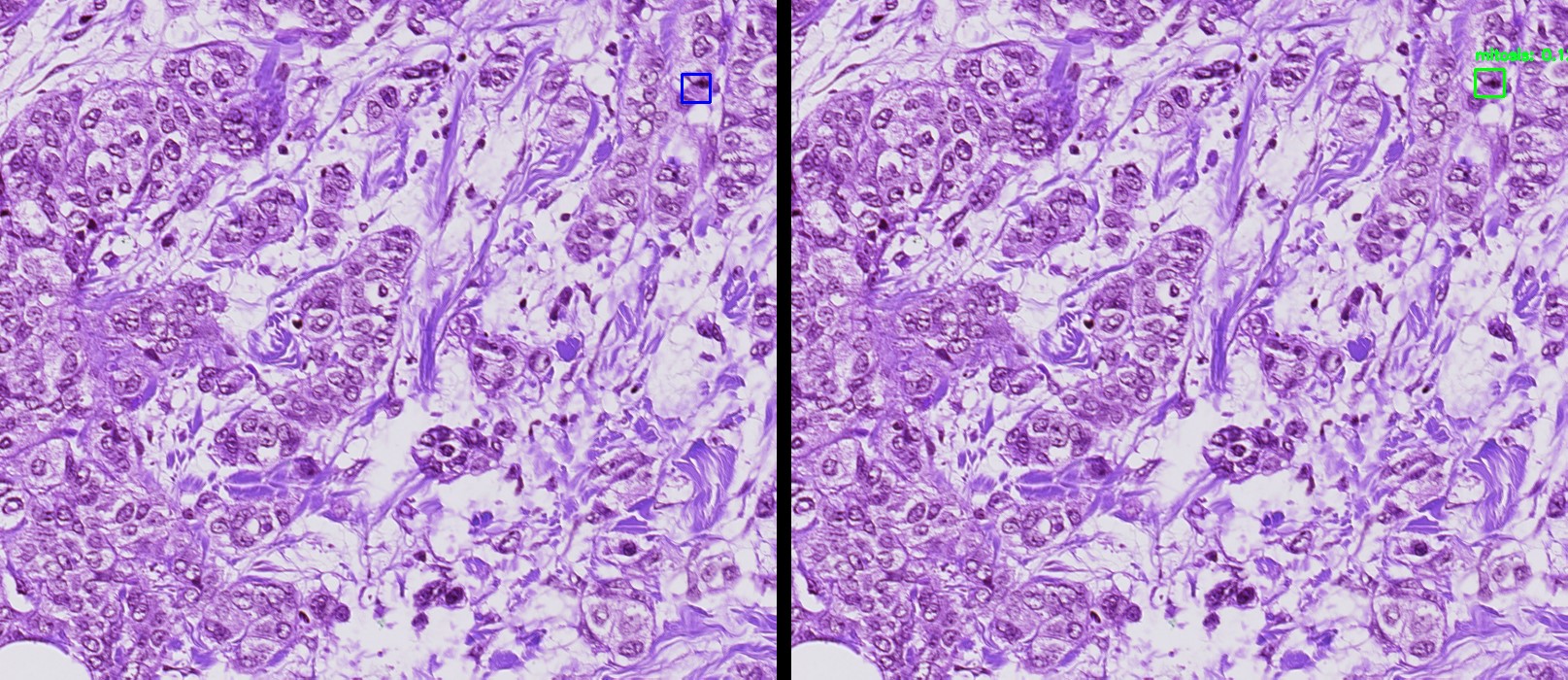
我绘制了最佳模型的PR曲线，我的计算方法是根据VOC2010中的给出方法[2]。先按可信度排序之后绘出图像，在计算AP的时候按照从右向左的方式，对Precision不断的取大，计算两个相邻点横坐标之间的面积，通过将面积加起来之后得到单个种类的AP，再将每个种类的AP求平均得到mAP，虽然这个数据集只有一个类型，但我写代码的时候还是适配了多分类问题。

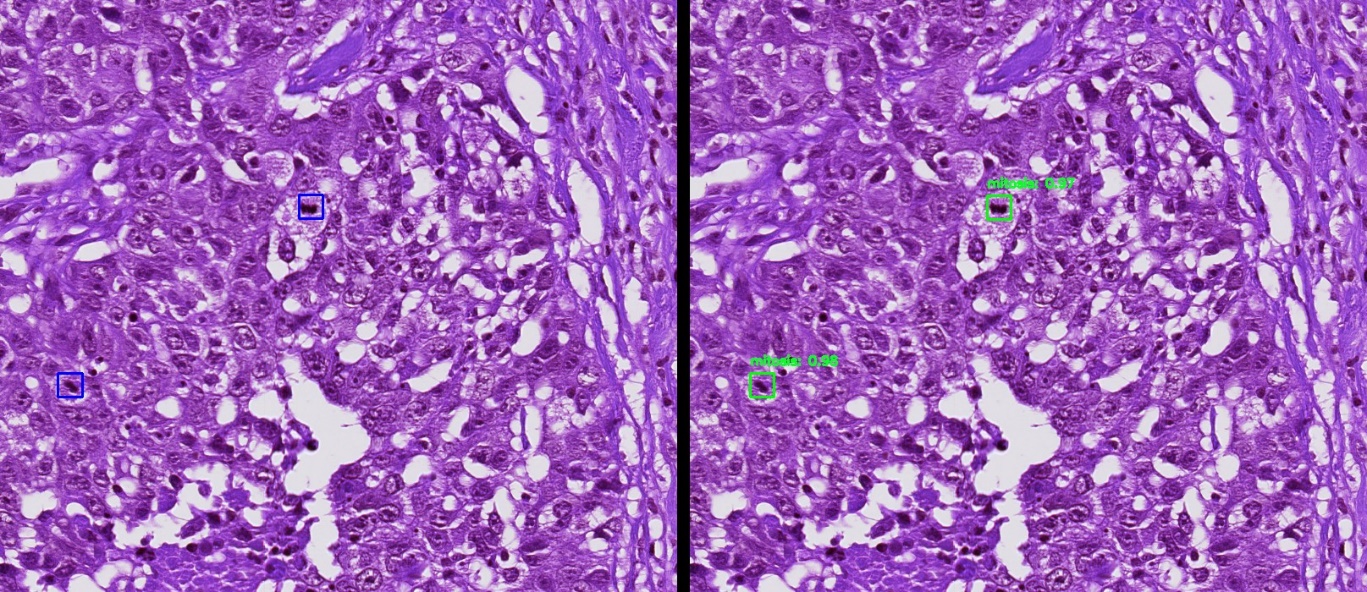
### 在测试集上的一些结果

在训练好模型之后，我使用cv2的库将标记的bbox绘制在图像上，代码比较长就不贴了，完整代码在附件有。左边的蓝色框是原图的标记，右边的绿色框是训练模型的标记。

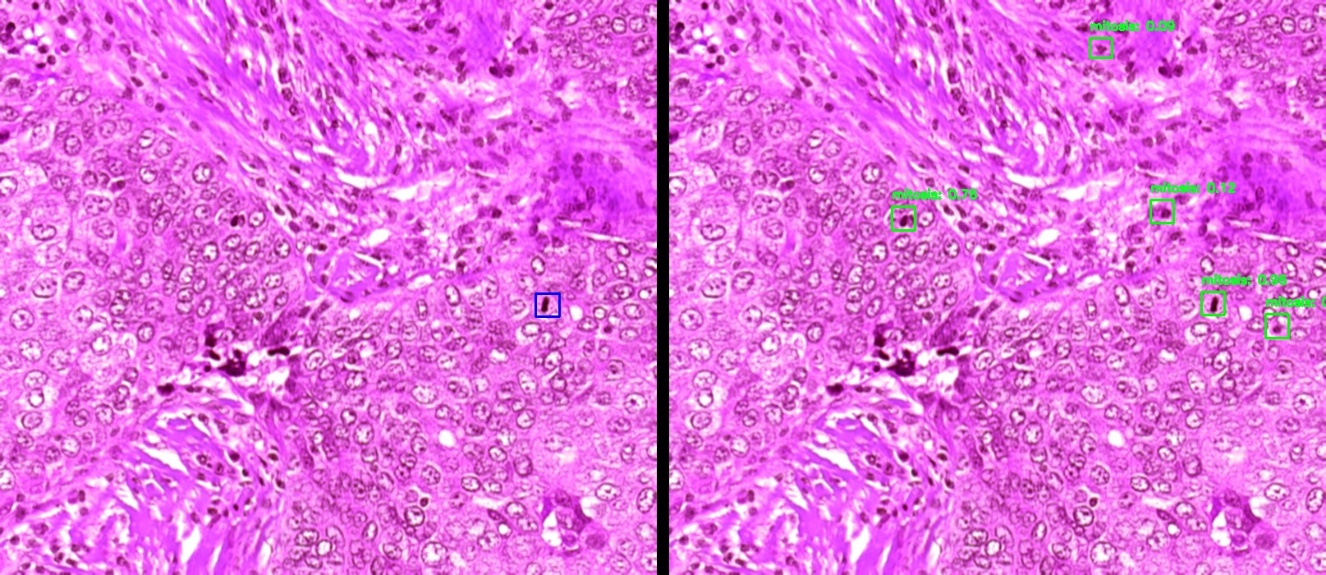
这些是效果比较好的：

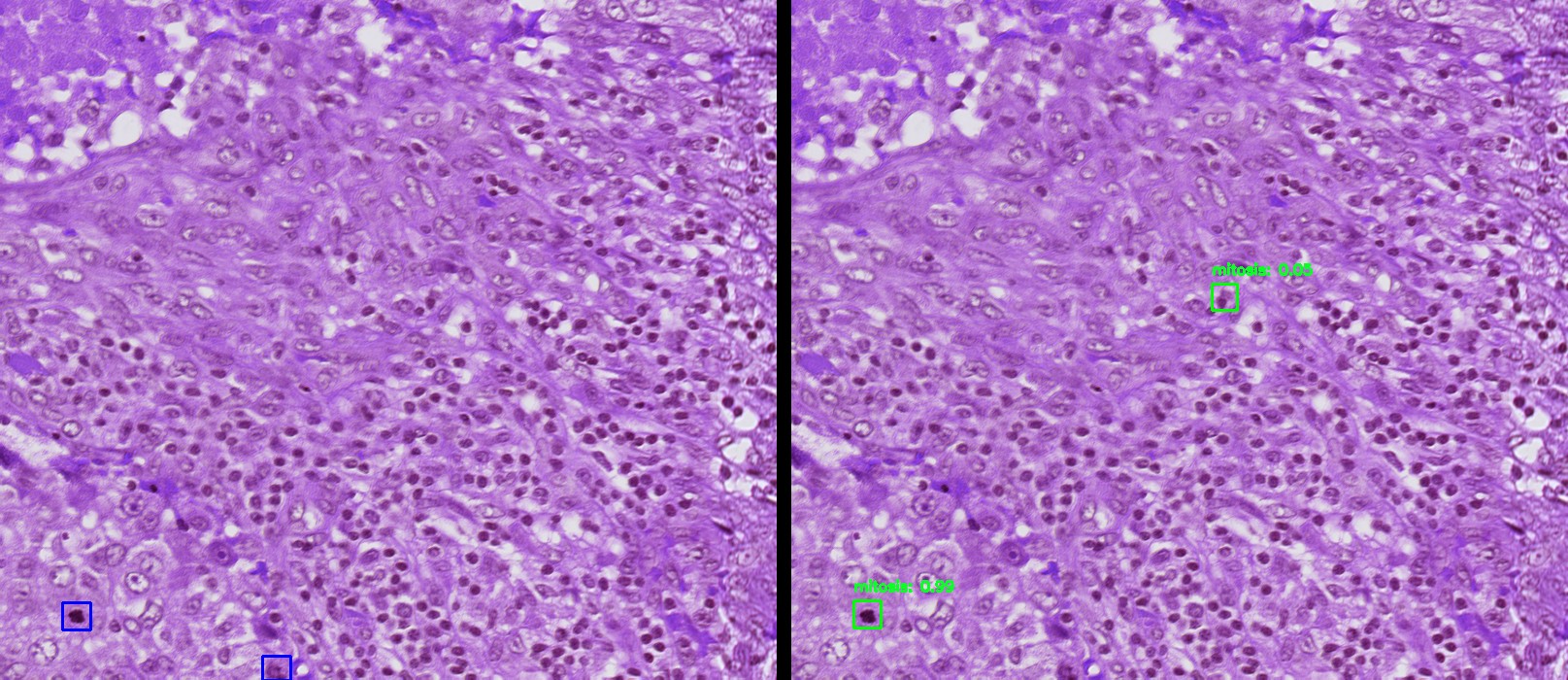






当然也有效果不好的：





# 总结

这次的实验从结果上来说还算是差强人意的，不过我也发现这个数据集并不是特别好，我感觉有些图片让我用人眼看也看不出来。而且Faster RCNN由于使用了得到候选区域再识别物体的策略，导致训练效率非常低，我有同学使用了现在非常火的yolo v9，训练时间是我的1/100，最后的效果还比我好很多，由此可见模型很重要。

这次实验花费了我非常多的时间和精力，主要是之前没有学过这方面的知识，只是上个学期学了一下CNN，不过这次使用的Faster RCNN也大量的使用了CNN，所以学起来也不算特别难受，除此之外我也都遇到了一些困难：

这次我花费最多时间解决的问题就是解决老师提供的代码的问题，这个代码相当的古老。使用的是CUDA8和python2.7，而且还要在Linux系统下运行。然后当我花了很多时间将环境终于搭好了之后，在运行代码的过程中还出现了显存不足的问题。

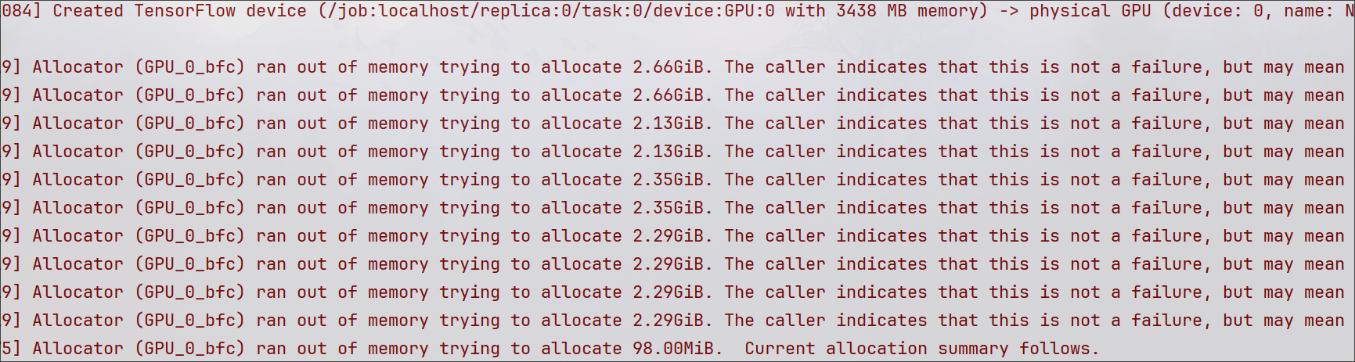


Figure 8 遇到显存不足的问题

在训练的一开始马上就吃完了我的4GB显存（batch\_size已经调整为1），于是训练就没法继续下去了。当然也有可能是因为我的显卡是RTX3050Ti，新显卡和CUDA8兼容不是很好，我后面也尝试了使用更新版本的TensorFlow，但是那需要更新版本的python。这个库的代码只能在python2.7运行良好，在更高的python版本也会出现问题。我最后也没能使用给出的代码成功训练，在询问老师之后我选择了torch框架使用了torchvision提供的预训练模型，完成了上面的实验。

这次的实验让我受益匪浅，使我对torch，torchvision和cv2库的理解更进一步的同时也锻炼了我解决问题的能力。上面提到的两个库都是在机器学习领域非常重要的工具，只有用好他们才能更好的完成实验，并理解这些模型的底层原理。我将继续保持学习，多学，多写，争取有朝一日能成为机器学习大师。

# 附件

## 完整代码

### 导入训练数据并记录每一个epoch的信息：train.py

import torch

import torch.nn as nn

import matplotlib.pyplot as plt

import data\_utils

import engine

import config

import test

def save\_loss\_plot(

    OUT\_DIR,

    train\_loss\_list,

    x\_label="iterations",

    y\_label="train loss",

    save\_name="train\_loss\_iter",

):

    figure\_1 = plt.figure(figsize=(10, 7), num=1, clear=True)

    train\_ax = figure\_1.add\_subplot()

    train\_ax.plot(train\_loss\_list, color="tab:blue")

    train\_ax.set\_xlabel(x\_label)

    train\_ax.set\_ylabel(y\_label)

    figure\_1.savefig(f"{OUT\_DIR}/{save\_name}.png")

    print("SAVING PLOTS COMPLETE...")

    # plt.close('all')

if \_\_name\_\_ == "\_\_main\_\_":

    torch.multiprocessing.freeze\_support()

    device = config.device

    torch.backends.cudnn.enabled = False

    torch.backends.cudnn.benchmark = False

    # 加载训练数据集

    train\_dataset = data\_utils.create\_train\_dataset(

        config.train\_img\_dir,

        config.train\_xml\_dir,

        config.img\_format,

        config.img\_width,

        config.img\_height,

        config.CLASSES,

    )

    train\_dataloader = data\_utils.create\_train\_loader(train\_dataset, 1)

    # 训练模型

    model = config.get\_model()

    model.to(device)

    optimizer = torch.optim.SGD(

        model.parameters(), lr=config.lr, momentum=0.9, weight\_decay=0.0005

    )

    criterion = nn.CrossEntropyLoss()

    train\_loss\_hist = engine.Averager()

    best\_mAP = 0

    # Train and validation loss lists to store loss values of all

    # iterations till ena and plot graphs for all iterations.

    train\_loss\_list = []

    loss\_cls\_list = []

    loss\_box\_reg\_list = []

    loss\_objectness\_list = []

    loss\_rpn\_list = []

    train\_loss\_list\_epoch = []

    mAP\_list = []

    for epoch in range(config.num\_epoch):

        train\_loss\_hist.reset()

        (

            batch\_loss\_list,

            batch\_loss\_cls\_list,

            batch\_loss\_box\_reg\_list,

            batch\_loss\_objectness\_list,

            batch\_loss\_rpn\_list,

        ) = engine.train\_one\_epoch(

            model,

            optimizer,

            train\_dataloader,

            device,

            epoch,

            config.num\_epoch,

            train\_loss\_hist,

        )

        mAP = 0

        for class\_idx in range(1, config.num\_classes):

            prec, rec, ap = test.eval\_on\_test(model, class\_idx)

            mAP += ap

        mAP /= config.num\_classes - 1

        mAP\_list.append(mAP)

        # Append the current epoch's batch-wise losses to the `train\_loss\_list`.

        train\_loss\_list.extend(batch\_loss\_list)

        loss\_cls\_list.extend(batch\_loss\_cls\_list)

        loss\_box\_reg\_list.extend(batch\_loss\_box\_reg\_list)

        loss\_objectness\_list.extend(batch\_loss\_objectness\_list)

        loss\_rpn\_list.extend(batch\_loss\_rpn\_list)

        # Append curent epoch's average loss to `train\_loss\_list\_epoch`.

        train\_loss\_list\_epoch.append(train\_loss\_hist.value)

        # Save loss plot for batch-wise list.

        save\_loss\_plot(config.OUT\_DIR, train\_loss\_list)

        # Save loss plot for epoch-wise list.

        save\_loss\_plot(

            config.OUT\_DIR,

            train\_loss\_list\_epoch,

            "epochs",

            "train loss",

            save\_name="train\_loss\_epoch",

        )

        save\_loss\_plot(

            config.OUT\_DIR,

            loss\_cls\_list,

            "iterations",

            "loss cls",

            save\_name="loss\_cls",

        )

        save\_loss\_plot(

            config.OUT\_DIR,

            loss\_box\_reg\_list,

            "iterations",

            "loss bbox reg",

            save\_name="loss\_bbox\_reg",

        )

        save\_loss\_plot(

            config.OUT\_DIR,

            loss\_objectness\_list,

            "iterations",

            "loss obj",

            save\_name="loss\_obj",

        )

        save\_loss\_plot(

            config.OUT\_DIR,

            loss\_rpn\_list,

            "iterations",

            "loss rpn bbox",

            save\_name="loss\_rpn\_bbox",

        )

        save\_loss\_plot(

            config.OUT\_DIR,

            mAP\_list,

            "epochs",

            "mAP",

            save\_name="mAP\_on\_eval",

        )

        # 保存最新模型和最佳模型

        torch.save(model.state\_dict(), config.get\_lastest\_model\_path())

        if mAP > best\_mAP:

            best\_mAP = mAP

            torch.save(model.state\_dict(), config.get\_best\_model\_path())

### 定义每个epoch具体如何训练：engine.py

import math

import sys

from tqdm import tqdm

import torch

# import torchvision.models.detection.mask\_rcnn

import torch.distributed as dist

from torchvision.models.detection.faster\_rcnn import FasterRCNN

def train\_one\_epoch(

    model: FasterRCNN,

    optimizer,

    data\_loader,

    device,

    epoch,

    num\_epoch,

    train\_loss\_hist,

    scaler=None,

    scheduler=None,

):

    model.train()

    # print(in\_blue(f"epoch: {epoch}"))

    # List to store batch losses.

    batch\_loss\_list = []

    batch\_loss\_cls\_list = []

    batch\_loss\_box\_reg\_list = []

    batch\_loss\_objectness\_list = []

    batch\_loss\_rpn\_list = []

    lr\_scheduler = None

    if epoch == 0:

        warmup\_factor = 1.0 / 1000

        warmup\_iters = min(1000, len(data\_loader) - 1)

        lr\_scheduler = torch.optim.lr\_scheduler.LinearLR(

            optimizer, start\_factor=warmup\_factor, total\_iters=warmup\_iters

        )

    step\_counter = 0

    bar = tqdm(data\_loader, desc=f"Training: epoch[{epoch+1}/{num\_epoch}]")

    for images, targets in bar:

        step\_counter += 1

        images = list(image.to(device) for image in images)

        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]

        with torch.cuda.amp.autocast(enabled=scaler is not None):

            loss\_dict = model(images, targets)

            losses = sum(loss for loss in loss\_dict.values())

        # reduce losses over all GPUs for logging purposes

        loss\_dict\_reduced = reduce\_dict(loss\_dict)

        losses\_reduced = sum(loss for loss in loss\_dict\_reduced.values())

        loss\_value = losses\_reduced.item()

        # tqdm.tqdm.write(f"Current value: {i}")

        bar.set\_postfix\_str(f"loss value: {loss\_value:.5}")

        if not math.isfinite(loss\_value):

            print(f"Loss is {loss\_value}, stopping training")

            print(loss\_dict\_reduced)

            sys.exit(1)

        optimizer.zero\_grad()

        if scaler is not None:

            scaler.scale(losses).backward()

            scaler.step(optimizer)

            scaler.update()

        else:

            losses.backward()

            optimizer.step()

        if lr\_scheduler is not None:

            lr\_scheduler.step()

        batch\_loss\_list.append(loss\_value)

        batch\_loss\_cls\_list.append(loss\_dict\_reduced["loss\_classifier"].detach().cpu())

        batch\_loss\_box\_reg\_list.append(loss\_dict\_reduced["loss\_box\_reg"].detach().cpu())

        batch\_loss\_objectness\_list.append(

            loss\_dict\_reduced["loss\_objectness"].detach().cpu()

        )

        batch\_loss\_rpn\_list.append(loss\_dict\_reduced["loss\_rpn\_box\_reg"].detach().cpu())

        train\_loss\_hist.send(loss\_value)

        if scheduler is not None:

            scheduler.step(epoch + (step\_counter / len(data\_loader)))

    return (

        batch\_loss\_list,

        batch\_loss\_cls\_list,

        batch\_loss\_box\_reg\_list,

        batch\_loss\_objectness\_list,

        batch\_loss\_rpn\_list,

    )

class Averager:

    def \_\_init\_\_(self):

        self.current\_total = 0.0

        self.iterations = 0.0

    def send(self, value):

        self.current\_total += value

        self.iterations += 1

    @property

    def value(self):

        if self.iterations == 0:

            return 0

        else:

            return 1.0 \* self.current\_total / self.iterations

    def reset(self):

        self.current\_total = 0.0

        self.iterations = 0.0

def reduce\_dict(input\_dict, average=True):

    world\_size = get\_world\_size()

    if world\_size < 2:

        return input\_dict

    with torch.inference\_mode():

        names = []

        values = []

        # sort the keys so that they are consistent across processes

        for k in sorted(input\_dict.keys()):

            names.append(k)

            values.append(input\_dict[k])

        values = torch.stack(values, dim=0)

        dist.all\_reduce(values)

        if average:

            values /= world\_size

        reduced\_dict = {k: v for k, v in zip(names, values)}

    return reduced\_dict

def is\_dist\_avail\_and\_initialized():

    if not dist.is\_available():

        return False

    if not dist.is\_initialized():

        return False

    return True

def get\_world\_size():

    if not is\_dist\_avail\_and\_initialized():

        return 1

    return dist.get\_world\_size()

### 继承torch的dataloader类以导入数据：data\_utils.py

import torch

import cv2

import numpy as np

import os

import glob as glob

import albumentations as A

from albumentations.pytorch import ToTensorV2

from xml.etree import ElementTree as et

from torch.utils.data import Dataset, DataLoader

def in\_blue(s):

    return "\033[94m" + str(s) + "\033[0m"

# the dataset class

class CustomDataset(Dataset):

    def \_\_init\_\_(

        self,

        images\_path,

        labels\_path,

        width,

        height,

        classes,

        image\_format,

        transforms=None,

        train=False,

    ):

        self.transforms = transforms

        # self.use\_train\_aug = use\_train\_aug

        self.images\_path = images\_path

        self.labels\_path = labels\_path

        self.height = height

        self.width = width

        self.classes = classes

        self.train = train

        # self.image\_file\_types = ["\*.jpg", "\*.jpeg", "\*.png", "\*.ppm"]

        self.image\_format = image\_format

        self.all\_image\_paths = []

        # get all the image paths in sorted order

        self.all\_image\_paths.extend(

            glob.glob(os.path.join(self.images\_path, "\*" + image\_format))

        )

        # print(in\_blue(self.all\_image\_paths))

        self.all\_annot\_paths = glob.glob(os.path.join(self.labels\_path, "\*.xml"))

        self.all\_images = [

            image\_path.split(os.path.sep)[-1] for image\_path in self.all\_image\_paths

        ]

        self.all\_images = sorted(self.all\_images)

        # Remove all annotations and images when no object is present.

        self.read\_and\_clean()

    def read\_and\_clean(self):

        # Discard any images and labels when the XML

        # file does not contain any object.

        for annot\_path in self.all\_annot\_paths:

            tree = et.parse(annot\_path)

            root = tree.getroot()

            object\_present = False

            for member in root.findall("object"):

                if member.find("bndbox"):

                    object\_present = True

            if object\_present is False:

                image\_name = annot\_path.split(os.path.sep)[-1].split(".xml")[0]

                image\_root = self.all\_image\_paths[0].split(os.path.sep)[:-1]

                # remove\_image = f"{'/'.join(image\_root)}/{image\_name}.jpg"

                remove\_image = os.path.join(

                    # os.sep.join(image\_root), image\_name + ".jpg"

                    os.sep.join(image\_root),

                    image\_name + self.image\_format,

                )

                print(f"Removing {annot\_path} and corresponding {remove\_image}")

                self.all\_annot\_paths.remove(annot\_path)

                self.all\_image\_paths.remove(remove\_image)

        # Discard any image file when no annotation file

        # is not found for the image.

        for image\_name in self.all\_images:

            possible\_xml\_name = os.path.join(

                self.labels\_path, image\_name.split(self.image\_format)[0] + ".xml"

            )

            if possible\_xml\_name not in self.all\_annot\_paths:

                print(f"{possible\_xml\_name} not found...")

                print(f"Removing {image\_name} image")

                # items = [item for item in items if item != element]

                self.all\_images = [

                    image\_instance

                    for image\_instance in self.all\_images

                    if image\_instance != image\_name

                ]

                # self.all\_images.remove(image\_name)

    def load\_image\_and\_labels(self, index):

        image\_name = self.all\_images[index]

        image\_path = os.path.join(self.images\_path, image\_name)

        # Read the image.

        image = cv2.imread(image\_path)

        # Convert BGR to RGB color format.

        image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB).astype(np.float32)

        image\_resized = cv2.resize(image, (self.width, self.height))

        image\_resized /= 255.0

        # Capture the corresponding XML file for getting the annotations.

        annot\_filename = image\_name[:-4] + ".xml"

        annot\_file\_path = os.path.join(self.labels\_path, annot\_filename)

        boxes = []

        orig\_boxes = []

        labels = []

        tree = et.parse(annot\_file\_path)

        root = tree.getroot()

        # Get the height and width of the image.

        image\_width = image.shape[1]

        image\_height = image.shape[0]

        # Box coordinates for xml files are extracted and corrected for image size given.

        for member in root.findall("object"):

            # Map the current object name to `classes` list to get

            # the label index and append to `labels` list.

            labels.append(self.classes.index(member.find("name").text))

            # xmin = left corner x-coordinates

            xmin = int(member.find("bndbox").find("xmin").text)

            # xmax = right corner x-coordinates

            xmax = int(member.find("bndbox").find("xmax").text)

            # ymin = left corner y-coordinates

            ymin = int(member.find("bndbox").find("ymin").text)

            # ymax = right corner y-coordinates

            ymax = int(member.find("bndbox").find("ymax").text)

            ymax, xmax = self.check\_image\_and\_annotation(

                xmax, ymax, image\_width, image\_height

            )

            orig\_boxes.append([xmin, ymin, xmax, ymax])

            # Resize the bounding boxes according to the

            # desired `width`, `height`.

            xmin\_final = (xmin / image\_width) \* self.width

            xmax\_final = (xmax / image\_width) \* self.width

            ymin\_final = (ymin / image\_height) \* self.height

            ymax\_final = (ymax / image\_height) \* self.height

            boxes.append([xmin\_final, ymin\_final, xmax\_final, ymax\_final])

        # Bounding box to tensor.

        boxes = torch.as\_tensor(boxes, dtype=torch.float32)

        # Area of the bounding boxes.

        area = (boxes[:, 3] - boxes[:, 1]) \* (boxes[:, 2] - boxes[:, 0])

        # No crowd instances.

        iscrowd = torch.zeros((boxes.shape[0],), dtype=torch.int64)

        # Labels to tensor.

        labels = torch.as\_tensor(labels, dtype=torch.int64)

        return (

            image,

            image\_resized,

            orig\_boxes,

            boxes,

            labels,

            area,

            iscrowd,

            (image\_width, image\_height),

        )

    def check\_image\_and\_annotation(self, xmax, ymax, width, height):

        """

        Check that all x\_max and y\_max are not more than the image

        width or height.

        """

        if ymax > height:

            ymax = height

        if xmax > width:

            xmax = width

        return ymax, xmax

    def \_\_getitem\_\_(self, idx):

        # Capture the image name and the full image path.

        # if not self.mosaic:

        image, image\_resized, orig\_boxes, boxes, labels, area, iscrowd, dims = (

            self.load\_image\_and\_labels(index=idx)

        )

        # visualize\_mosaic\_images(boxes, labels, image\_resized, self.classes)

        # Prepare the final `target` dictionary.

        target = {}

        target["boxes"] = boxes

        target["labels"] = labels

        target["area"] = area

        target["iscrowd"] = iscrowd

        image\_id = torch.tensor([idx])

        target["image\_id"] = image\_id

        train\_aug = get\_train\_aug()

        sample = train\_aug(image=image\_resized, bboxes=target["boxes"], labels=labels)

        image\_resized = sample["image"]

        target["boxes"] = torch.Tensor(sample["bboxes"])

        return image\_resized, target

    def \_\_len\_\_(self):

        return len(self.all\_images)

def collate\_fn(batch):

    """

    To handle the data loading as different images may have different number

    of objects and to handle varying size tensors as well.

    """

    return tuple(zip(\*batch))

# Prepare the final datasets and data loaders.

def create\_train\_dataset(

    train\_dir\_images,

    train\_dir\_labels,

    image\_format,

    resize\_width,

    resize\_height,

    classes,

):

    train\_dataset = CustomDataset(

        train\_dir\_images,

        train\_dir\_labels,

        resize\_width,

        resize\_height,

        classes,

        image\_format,

        train=True,

    )

    return train\_dataset

def create\_valid\_dataset(

    valid\_dir\_images,

    valid\_dir\_labels,

    image\_format,

    resize\_width,

    resize\_height,

    classes,

):

    valid\_dataset = CustomDataset(

        valid\_dir\_images,

        valid\_dir\_labels,

        resize\_width,

        resize\_height,

        classes,

        image\_format,

        train=False,

    )

    return valid\_dataset

def create\_train\_loader(train\_dataset, batch\_size, num\_workers=0):

    train\_loader = DataLoader(

        train\_dataset,

        batch\_size=batch\_size,

        shuffle=True,

        num\_workers=num\_workers,

        collate\_fn=collate\_fn,

    )

    return train\_loader

def create\_valid\_loader(valid\_dataset, batch\_size, num\_workers=0):

    valid\_loader = DataLoader(

        valid\_dataset,

        batch\_size=batch\_size,

        shuffle=False,

        num\_workers=num\_workers,

        collate\_fn=collate\_fn,

    )

    return valid\_loader

def get\_train\_aug():

    return A.Compose(

        [

            A.MotionBlur(blur\_limit=3, p=0.5),

            A.Blur(blur\_limit=3, p=0.5),

            A.RandomBrightnessContrast(brightness\_limit=0.2, p=0.5),

            A.ColorJitter(p=0.5),

            # A.Rotate(limit=10, p=0.2),

            A.RandomGamma(p=0.2),

            A.RandomFog(p=0.2),

            # A.RandomSunFlare(p=0.1),

            # `RandomScale` for multi-res training,

            # `scale\_factor` should not be too high, else may result in

            # negative convolutional dimensions.

            # A.RandomScale(scale\_limit=0.15, p=0.1),

            # A.Normalize(

            #     (0.485, 0.456, 0.406),

            #     (0.229, 0.224, 0.225)

            # ),

            ToTensorV2(p=1.0),

        ],

        bbox\_params={"format": "pascal\_voc", "label\_fields": ["labels"]},

    )

### 计算模型Precision，Recall和AP的评估工具：test.py

import os

import torch

import data\_utils

import config

import cv2

import torchvision

import matplotlib.pyplot as plt

import numpy as np

from tqdm import tqdm

device = config.device

# 加载测试数据集

test\_dataset = data\_utils.create\_valid\_dataset(

    config.test\_img\_dir,

    config.test\_xml\_dir,

    config.img\_format,

    config.img\_width,

    config.img\_height,

    config.CLASSES,

)

test\_dataloader = data\_utils.create\_valid\_loader(test\_dataset, 1)

def plot\_RP(prec, rec, name):

    plt.figure()

    plt.plot(rec, prec)

    plt.xlabel("Recall")

    plt.ylabel("Precision")

    plt.tight\_layout()

    plt.savefig(os.path.join(config.OUT\_DIR, name))

# 在测试集上验证

def eval\_on\_test(

    model, class\_idx, iou\_threshold=0.5, show\_img=False, mark\_img=False, plot\_rp=False

):

    model.eval()

    if not show\_img and mark\_img:

        output\_path = os.path.join(config.OUT\_DIR, "test\_out")

        if not os.path.exists(output\_path):

            os.mkdir(output\_path)

    # 真正类，假正类，置信度

    list\_tp, list\_fp, list\_conf = [], [], []

    # GT个数

    num\_gt = 0

    for i, (images, targets) in enumerate(tqdm(test\_dataloader, desc="Evaluating: ")):

        images = list(image.to(device) for image in images)

        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]

        with torch.no\_grad():

            outputs = model(images)

        for j, (image, target) in enumerate(zip(images, targets)):

            # img = image.permute(1, 2, 0).byte().cpu().numpy()

            img = (image.permute(1, 2, 0) \* 255).byte().cpu().numpy()

            if len(target["boxes"]) == 0:

                continue

            cur\_class\_ind = target["labels"] == class\_idx

            anno\_boxes = target["boxes"][cur\_class\_ind]

            num\_gt += len(anno\_boxes)

            pred\_boxes, confs, pred\_labels = (

                outputs[j]["boxes"],

                outputs[j]["scores"],

                outputs[j]["labels"],

            )

            cur\_class\_ind = pred\_labels == class\_idx

            pred\_boxes = pred\_boxes[cur\_class\_ind]

            confs = confs[cur\_class\_ind]

            confs, indices = torch.sort(confs, descending=True)

            pred\_boxes = pred\_boxes[indices]

            iou = torchvision.ops.box\_iou(pred\_boxes, anno\_boxes)

            iou = iou.cpu().numpy()

            # 是否第一次标记GT

            gt\_vis = [False] \* len(anno\_boxes)

            confs = confs.cpu()

            for areas, conf in zip(iou, confs):

                # 当前box的GT下标

                gt\_idx = np.argmax(areas)

                if areas[gt\_idx] >= iou\_threshold and gt\_vis[gt\_idx] is False:

                    # TP=1, FP=0

                    gt\_vis[gt\_idx] = True

                    list\_tp.append(1)

                    list\_fp.append(0)

                else:

                    # TP=0, FP=1

                    list\_tp.append(0)

                    list\_fp.append(1)

                list\_conf.append(conf)

            # 在图片上标记

            if mark\_img:

                # 标注框和预测框图片

                img\_with\_anno = img.copy()

                img\_with\_pred = img.copy()

                for box in target["boxes"]:

                    x1, y1, x2, y2 = [int(coord.item()) for coord in box]

                    cv2.rectangle(img\_with\_anno, (x1, y1), (x2, y2), (255, 0, 0), 2)

                for box, conf, label in zip(pred\_boxes, confs, pred\_labels):

                    x1, y1, x2, y2 = [int(coord.item()) for coord in box]

                    cv2.rectangle(img\_with\_pred, (x1, y1), (x2, y2), (0, 255, 0), 2)

                    cv2.putText(

                        img\_with\_pred,

                        f"{config.CLASSES[label.item()]}: {conf.item():.2f}",

                        # f"{conf.item():.2f}",

                        (x1, y1 - 10),

                        cv2.FONT\_HERSHEY\_SIMPLEX,

                        0.5,

                        (36, 255, 12),

                        2,

                    )

                # 将两个图像水平并排显示

                img\_height, img\_width, \_ = img.shape

                combined\_width = img\_width \* 2 + 15  # 增加图像间隔

                img = np.zeros((img\_height, combined\_width, 3), dtype=np.uint8)

                img[:, :img\_width] = img\_with\_anno

                img[:, img\_width + 15 :] = img\_with\_pred

                if show\_img:

                    # 展示图像

                    cv2.imshow(f"Prediction on class {config.CLASSES[class\_idx]}", img)

                    cv2.waitKey(0)

                    cv2.destroyAllWindows()

                else:

                    # 保存图像

                    output\_path = os.path.join(

                        config.OUT\_DIR, "test\_out", f"image\_{i}\_{j}.png"

                    )

                    cv2.imwrite(output\_path, img)

    list\_conf = np.array(list\_conf)

    ind = list\_conf.argsort()

    ind = ind[::-1]

    # print(list\_conf[ind])

    # list\_conf = list\_conf[ind]

    list\_tp = np.array(list\_tp)[ind]

    list\_fp = np.array(list\_fp)[ind]

    # 查准率，查全率

    prec, rec = [], []

    tp, fp = 0, 0

    for cur\_tp, cur\_fp in zip(list\_tp, list\_fp):

        tp += cur\_tp

        fp += cur\_fp

        prec.append(tp / (tp + fp))

        rec.append(tp / num\_gt)

    if plot\_rp:

        plot\_RP(prec, rec, "AP")

    # correct AP calculation

    # first append sentinel values at the end

    mrec = np.concatenate(([0.0], rec, [1.0]))

    mpre = np.concatenate(([0.0], prec, [0.0]))

    # compute the precision envelope

    for i in range(mpre.size - 1, 0, -1):

        mpre[i - 1] = np.maximum(mpre[i - 1], mpre[i])

    # to calculate area under PR curve, look for points

    # where X axis (recall) changes value

    i = np.where(mrec[1:] != mrec[:-1])[0]

    # and sum (\Delta recall) \* prec

    ap = np.sum((mrec[i + 1] - mrec[i]) \* mpre[i + 1])

    return prec, rec, ap

if \_\_name\_\_ == "\_\_main\_\_":

    model = config.get\_model()

    model.load\_state\_dict(torch.load(config.get\_best\_model\_path(), map\_location=device))

    model.to(device)

    mAP = 0

    for class\_idx in range(1, config.num\_classes):

        prec, rec, ap = eval\_on\_test(model, class\_idx, plot\_rp=True, mark\_img=True)

        mAP += ap

    mAP /= config.num\_classes - 1

    print(f"mAP = {mAP}")

### 配置文件（记录类名，学习率，数据集位置）：config.py

import os

from torchvision.models.detection import (

    fasterrcnn\_resnet50\_fpn\_v2,

    FasterRCNN\_ResNet50\_FPN\_V2\_Weights,

)

import torch

from torchvision.models.detection.faster\_rcnn import FastRCNNPredictor, FasterRCNN

def in\_blue(s):

    return "\033[94m" + str(s) + "\033[0m"

def in\_red(s):

    return "\033[91m" + str(s) + "\033[0m"

# 参数

num\_epoch = 150

lr = 0.001

# 定义类别

CLASSES = ["\_\_background\_\_", "mitosis"]

num\_classes = len(CLASSES)  # 背景类+类别数

# 图片缩放

img\_width = 800  # 1663

img\_height = 700  # 1485

# 图片格式

img\_format = ".png"

# 数据存放位置

data\_path = "C:\\Users\\suxto\\Downloads\\data"

# 输出文件保存位置

OUT\_DIR = "C:\\Users\\suxto\\Downloads\\RCNN\_Out"

# 训练数据集

train\_img\_dir = os.path.join(data\_path, "train", "img")

train\_xml\_dir = os.path.join(data\_path, "train", "xml")

# 测试数据集

test\_img\_dir = os.path.join(data\_path, "test", "img")

test\_xml\_dir = os.path.join(data\_path, "test", "xml")

device = torch.device("cuda") if torch.cuda.is\_available() else torch.device("cpu")

print(in\_blue(f"using {device}"))

if not os.path.exists(data\_path):

    print(in\_red(f"Data path not exist!({data\_path})"))

    exit(1)

if not os.path.exists(OUT\_DIR):

    os.mkdir(OUT\_DIR)

# 保存模型名称

latest\_modle\_name = "faster\_rcnn\_model\_latest"

best\_modle\_name = "faster\_rcnn\_model\_best"

modle\_suffix = ".pth"

# 创建 Fast RCNN 模型

def get\_model():

    weights = FasterRCNN\_ResNet50\_FPN\_V2\_Weights.DEFAULT

    model: FasterRCNN = fasterrcnn\_resnet50\_fpn\_v2(weights=weights, pretrained=True)

    in\_features = model.roi\_heads.box\_predictor.cls\_score.in\_features

    model.roi\_heads.box\_predictor = FastRCNNPredictor(in\_features, num\_classes)

    return model

def get\_lastest\_model\_path(num=None):

    if num:

        return os.path.join(OUT\_DIR, latest\_modle\_name + num + modle\_suffix)

    else:

        return os.path.join(OUT\_DIR, latest\_modle\_name + modle\_suffix)

def get\_best\_model\_path(num=None):

    if num:

        return os.path.join(OUT\_DIR, best\_modle\_name + num + modle\_suffix)

    else:

        return os.path.join(OUT\_DIR, best\_modle\_name + modle\_suffix)

## 参考资料

[1] REN S, HE K, GIRSHICK R, 等. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks[M/OL]. arXiv, 2016[2024-04-17]. http://arxiv.org/abs/1506.01497.

[2] EVERINGHAM M, ESLAMI S M A, VAN GOOL L, 等. The Pascal Visual Object Classes Challenge: A Retrospective[J/OL]. International Journal of Computer Vision, 2015, 111(1): 98-136. DOI:10.1007/s11263-014-0733-5.