

北 京 林 业 大 学

2023 学年— 2024 学年第 一 学期 图像处理与深度学习 ——基于 PyTorch 实现 Faster R-CNN Two-stage 目标检测

编 号： 10

专 业： 数字媒体技术 班 级： 数媒 211

姓 名： 高月涵 学 号： 211002129

任课教师： 聂笑盈

设计题目：基于 PyTorch 实现 Faster R-CNN Two-stage 目标检测

严禁剽窃、抄袭等作弊行为！

作品主要内容：

作品最终可以实现一个 Two-stage 目标检测过程，一是查找包含对象的边界框，使得每个边界框仅包含一个对象。二是对每个边界框内的图像进行分类并为其分配标签，其中包括了对每个候选区域进行特征提取、目标分类和边界框回归。

作品主要用到了 Pycharm 进行调试和测试，在 Anaconda 环境中运行，并应用了 Pytorch 这一功能强大且灵活的深度学习框架来辅助目标检测任务的实现。本次大作业中使用的深度学习架构 Faster R-CNN 能够准确地定位图像中的目标物体，提供较为精确的边界框信息。同时支持多类别的目标检测，使得模型在处理复杂场景中的各种目标时具有很好的通用性，对于不同尺寸、形状和姿态的目标物体都相对具有较好的鲁棒性。这使得它适用于不同场景和数据集，具有较好的泛化能力。Faster R-CNN 在一定程度上能够处理目标物体的遮挡，使得模型对于复杂场景下的目标检测更为鲁棒。

详细实现步骤：

准备：环境配置

1. Anaconda：选择使用 Anaconda 以便于环境管理，也可以同时在一个电脑上安装多种环境，不同环境放置不同框架：pytorch、tensorflow、keras 可以在不同的环境下安装，在本次大作业中使用了命令行 `conda create - pytorch` 创建一个名为 pytorch 的新环境，接下来继续完成环境配置。

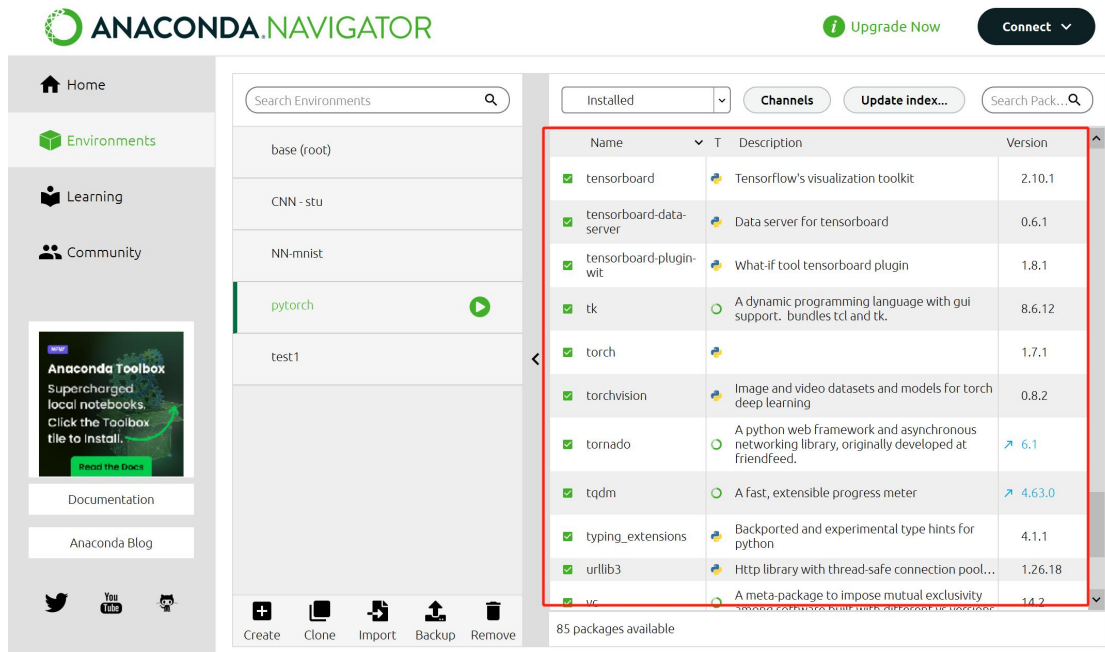


图 1 使用 anaconda 管理环境

2. CUDA: 由于要在 GPU 上进行深度学习的模型的训练，且我的电脑装有 NVIDIA 显卡，说明电脑 GPU 支持 CUDA。在使用 PyTorch 进行 GPU 加速训练时，需要确保安装的 PyTorch 版本与 CUDA 版本兼容，于是按照官方文档对应查找安装的版本为 CUDA 10.2

	torch-1.7.1-cp36-cp36m-win_amd64.whl	2023/12/20 15:30
	cuda_10.2.89_441.22_win10.exe	2023/12/27 14:54
	torchvision-0.8.2-cp36-cp36m-win_amd64....	2023/12/27 15:55

图 2 CUDA 版本为 10.2

安装完成后在 C 盘找到目录

C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.2 将 cuDNN 压缩包解压后的内容复制到目录下。

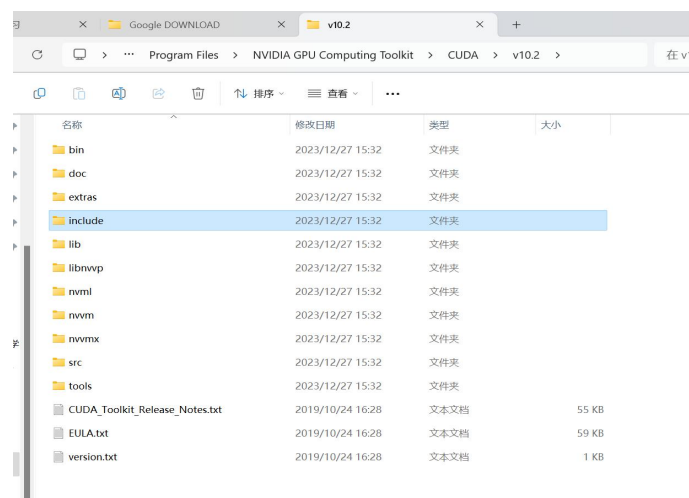


图 3 配置 CUDA 根目录

3. torch 和 torchvision: 整个 Faster R-CNN 网络训练模型都是基于 Pytorch 深度学习框架实现的, 安装版本在 torch1.2 以上的基本都可以支持跑完整个网络, 由于要用到 fp16 以使用 .amp 功能, 需要满足 1.7 及以上的版本, 故选择了 torch 1.7.1。根据 PyTorch 和 torchvision 的兼容性矩阵选择了 torchvision 0.8.2。


 torch-1.7.1-cp36-cp36m-win_amd64.whl	2023/12/26 15:58	WHL 文件	1,252,321 KB
 cuda_10.2.89_441.22_win10.exe	2023/12/27 14:54	应用程序	2,556,900 KB
 torchvision-0.8.2-cp36-cp36m-win_amd64....	2023/12/27 15:55	WHL 文件	1,463 KB

图 4 torch1.7.1 对应 torchvision 版本 0.8.2

4. 其他依赖库: scipy、numpy、matplotlib、opencv_python、torch、torchvision 等依赖库的组合提供了一个完整的深度学习工具链, 从数据处理、模型构建到可视化进度监控都有所涉及。具体用到的依赖库和安装版本如下:

```
scipy==1.2.1
numpy==1.17.0
matplotlib==3.1.2
opencv_python==4.1.2.30
torch==1.2.0
torchvision==0.4.0
tqdm==4.60.0
Pillow==8.2.0
h5py==2.10.0
```

图 5 需要安装的 python 依赖库和版本号

5. Pycharm 调试运行: 由于代码是用 python 语言编写, 故选择了用于 Python 开发的强大 IDE, 同时 PyCharm 拥有丰富的插件生态系统, 可以通过安装插件来扩展其功能, 支持用于深度学习开发的扩展插件, 提供了与流行深度学习框架的集成和支持。

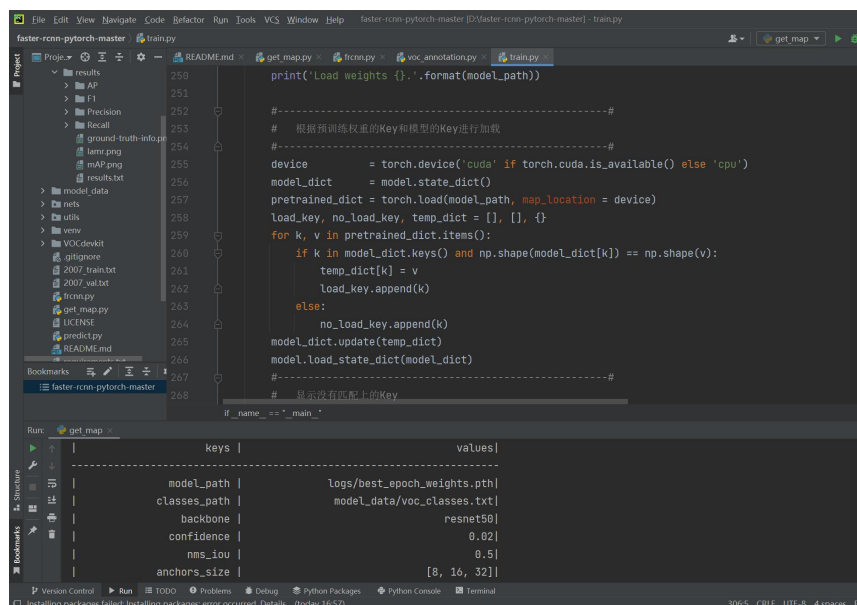


图 6 调试软件 Pycharm

实现步骤：

总流程图：

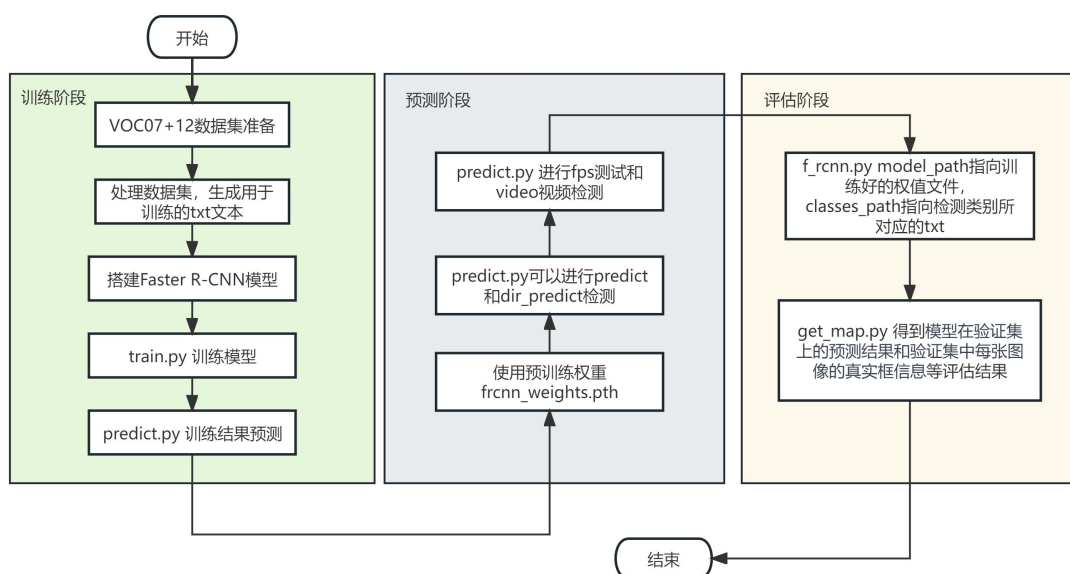
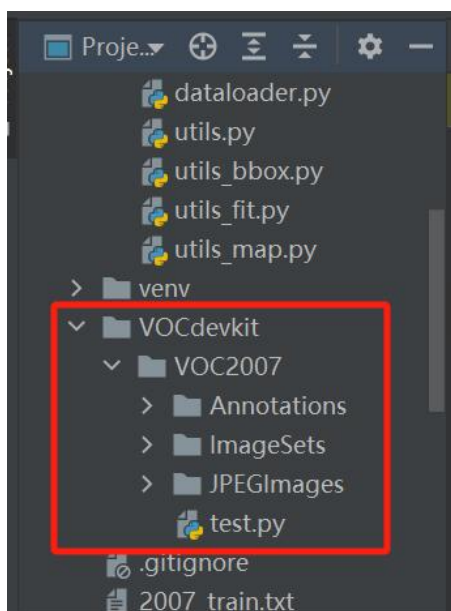


图 7 总流程图

一、训练 VOC07+12 数据集

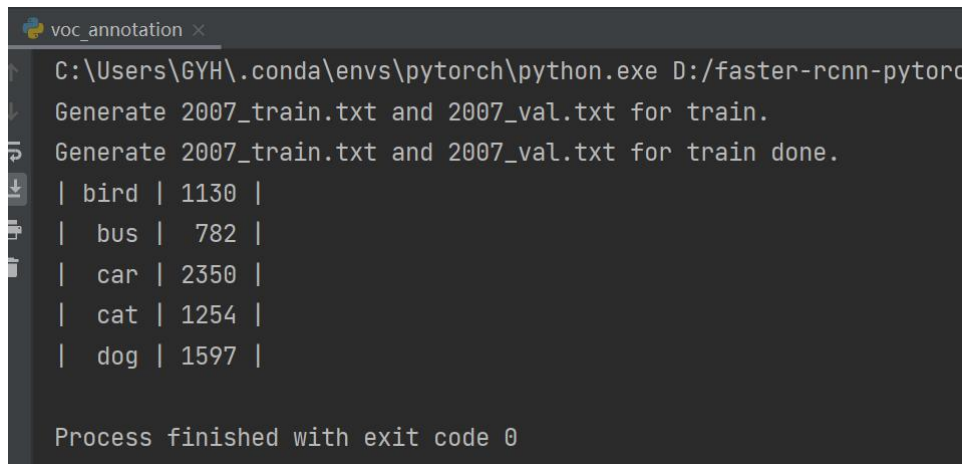
1. 数据集准备： 实验中选择使用 VOC 格式进行训练，在训练开始前需要下载好 VOC2017 数据集，解压后放在根目录中。



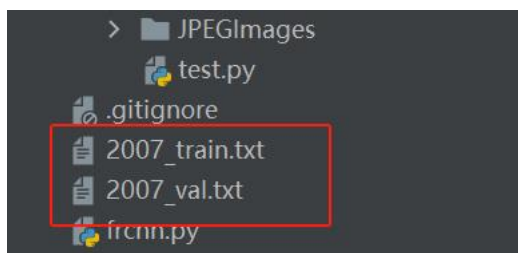
2. 处理 VOC 数据集： 运行代码文件 `voc_annotation.py`，处理后生成用于训练的文本文件。代码中通过 `annotation_mode` 指定不同的运行模式。0 表示整个标签处理过程，1 表示生成 `VOCdevkit/VOC2007/ImageSets` 目录下的 txt 文件，2 表示生成训练用的 `2007_train.txt` 和 `2007_val.txt`。生成 `ImageSets` 目录下的 txt 文件，主要包括生成训练、验证和测试的图像文件列表。`trainval.txt` 包含了所有

的训练和验证集图片的文件名，**test.txt** 包含了所有测试集图片的文件名，**train.txt** 包含了用于训练的图片文件名，**val.txt** 包含了用于验证的图片文件名。这些文件列表用于后续的训练、验证和测试集划分。

生成训练用的 **2007_train.txt** 和 **2007_val.txt** 文件通过遍历 VOC2007 和集合，生成包含图像路径和目标信息的训练用 **txt** 文件，最后在训练 **Faster R-CNN** 模型时使用。且在生成训练用的 **txt** 文件时，统计了每个类别的目标数量，并在最后输出了一个表格，展示每个类别的目标数量。下图为运行 **voc_annotation.py** 的结果和最终生成的 **txt** 文件。



```
voc_annotation x
C:\Users\GYH\.conda\envs\pytorch\python.exe D:/faster-rcnn-pytorch
Generate 2007_train.txt and 2007_val.txt for train.
Generate 2007_train.txt and 2007_val.txt for train done.
| bird | 1130 |
| bus | 782 |
| car | 2350 |
| cat | 1254 |
| dog | 1597 |
Process finished with exit code 0
```



3. 定义 **Faster R-CNN** 模型：定义模型的功能在 **nets/frcnn.py** 中实现，可选择 **vgg** 或 **resnet50** 作为主干网络，**Region Proposal Network** 负责生成候选框，在 **rpn.py** 中实现，是 **Faster R-CNN** 模型的一个关键组件，用于生成候选目标区域，从而在后续的目标检测任务中减少需要处理的区域数量。**head**：分类头部，负责对候选框进行分类和回归。**forward** 方法实现模型的前向传播。**freeze_bn** 方法用于冻结 **Batch Normalization** 层，将其设为评估模式，主要用于模型推理阶段。

```

class FasterRCNN(nn.Module):
    def __init__(self, num_classes,
                  mode = "training",
                  feat_stride = 16,
                  anchor_scales = [8, 16, 32],
                  ratios = [0.5, 1, 2],
                  backbone = 'vgg',
                  pretrained = False):
        super(FasterRCNN, self).__init__()
        self.feat_stride = feat_stride
        #-----#
        # 一共存在两个主干
        #  vgg和resnet50

```

4. 训练 Faster R-CNN 模型：首先冻结模型的一部分参数，通常是骨干特征提取网络（通常是预训练的卷积层）。这有助于防止这些参数在初始训练阶段被破坏。根据 `batch_size` 动态调整学习率的初始值 `Init_lr_fit` 和最小值 `Min_lr_fit`。

```

#-----#
# 判断当前batch_size, 自适应调整学习率
#-----#
nbs = 16
lr_limit_max = 1e-4 if optimizer_type == 'adam' else 5e-2
lr_limit_min = 1e-4 if optimizer_type == 'adam' else 5e-4
Init_lr_fit = min(max(batch_size / nbs * Init_lr, lr_limit_min), lr_limit_max)
Min_lr_fit = min(max(batch_size / nbs * Min_lr, lr_limit_min * 1e-2), lr_limit_max * 1e-2)

```

接下来根据 `optimizer_type` 的值选择使用 Adam 还是 SGD 优化器。

```

#-----#
# 根据optimizer_type选择优化器
#-----#
optimizer = {
    'adam' : optim.Adam(model.parameters(), Init_lr_fit, betas = (momentum, 0.999), weight_decay = v
    'sgd' : optim.SGD(model.parameters(), Init_lr_fit, momentum = momentum, nesterov=True, weight_d
}[optimizer_type]

```

获取学习率下降的公式，`get_lr_scheduler` 函数根据不同的策略返回学习率下降函数。

```

#-----#
# 获得学习率下降的公式
#-----#
lr_scheduler_func = get_lr_scheduler(lr_decay_type, Init_lr_fit, Min_lr_fit, UnFreeze_Epoch)

```

模型训练循环，调用 `fit_one_epoch` 函数执行模型的一个训练周期。


```

#-----#
# 开始模型训练
#-----#
for epoch in range(Init_Epoch, UnFreeze_Epoch):
    #-----#
    # 如果模型有冻结学习部分
    # 则解冻，并设置参数
    #-----#
    if epoch >= Freeze_Epoch and not UnFreeze_flag and Freeze_Train:
        batch_size = Unfreeze_batch_size

    #-----#
    # 判断当前batch_size，自适应调整学习率
    #-----#

```

当达到指定的 Freeze_Epoch 时，解冻模型的一部分参数，并重新设置学习率，根据学习率下降函数动态调整当前学习率。

```

# 如果模型有冻结学习部分
# 则解冻，并设置参数
#-----#
if epoch >= Freeze_Epoch and not UnFreeze_flag and Freeze_Train:
    batch_size = Unfreeze_batch_size

#-----#
# 判断当前batch_size，自适应调整学习率
#-----#
nbs = 16
lr_limit_max = 1e-4 if optimizer_type == 'adam' else 5e-2
lr_limit_min = 1e-4 if optimizer_type == 'adam' else 5e-4
Init_lr_fit = min(max(batch_size / nbs * Init_lr, lr_limit_min), lr_limit_max)
Min_lr_fit = min(max(batch_size / nbs * Min_lr, lr_limit_min * 1e-2), lr_limit_max * 1e-2)

```

训练结束后关闭用于记录 loss 的 TensorBoard writer。

```

set_optimizer_lr(optimizer, lr_scheduler_func, epoch)

fit_one_epoch(model, train_util, loss_history, eval_callback, optimizer)

loss_history.writer.close()

```

train.py 这段代码实现了 Faster R-CNN 模型的训练过程，其中包含动态调整学习率、冻结部分网络、保存模型的功能。

二、预测

以 mode 区分为 4 种预测功能：

Mode= 'predict' —— 单张图片预测：

用户输入待预测的图片路径，通过 frCNN.detect_image 函数对单张图片进行目标检测。用户可以选择是否对目标进行截取（crop）和计数（count），并可以根据

根据需要自行修改保存图片等操作。预测结果通过弹出窗口显示，并保存为 predict_image.jpg。

```
if mode == "predict":
    while True:
        img = input('Input image filename:')
        try:
            image = Image.open(img)
        except:
            print('Open Error! Try again!')
            continue
        else:
            r_image = frcnn.detect_image(image, crop = crop, count = count)
            r_image.show()
            r_image.save('predict_image.jpg')
            # print('Output image saved as output_image.jpg')
```

Mode='video' —— 视频检测:

用户输入指定 video_path，可以选择是否保存检测结果的 video_save_path 以及保存视频的帧率。通过 cv2.VideoCapture 读取视频帧，将每帧转换为 Image 对象，并通过 frcnn.detect_image 进行目标检测。可以实时显示检测结果，同时可以选择保存检测结果的视频。

```
elif mode == "video":
    capture=cv2.VideoCapture(video_path)
    if video_save_path!="":
        fourcc = cv2.VideoWriter_fourcc(*'XVID')
        size = (int(capture.get(cv2.CAP_PROP_FRAME_WIDTH)), int(capture.get(cv2.CAP_PROP_FRAME_HEIGHT)))
        out = cv2.VideoWriter(video_save_path, fourcc, video_fps, size)

    fps = 0.0
    while(True):
        t1 = time.time()
        # 读取某一帧
        ref,frame=capture.read()
        # 格式转变, BGRtoRGB
        frame = cv2.cvtColor(frame,cv2.COLOR_BGR2RGB)
```

Mode='fps'——FPS 测试:

用户指定用于测试的图像路径 fps_image_path 以及测试的间隔次数 test_interval。通过 frcnn.get_FPS 函数测试模型在指定图片上的处理速度，输出处理时间和 FPS。

```
elif mode == "fps":
    img = Image.open(fps_image_path)
    tact_time = frcnn.get_FPS(img, test_interval)
    print(str(tact_time) + ' seconds, ' + str(1/tact_time) + 'FPS, @batch_size 1')
```

Mode='dir_predict' —— 目录预测:

用户指定待检测图片所在文件夹 `img` 和保存检测结果的文件夹 `img_out`。遍历指定文件夹中的图片，利用 `frcnn.detect_image` 函数进行目标检测，并保存检测结果。

```
elif mode == "dir_predict":
    import os
    from tqdm import tqdm

    img_names = os.listdir(dir_origin_path)
    for img_name in tqdm(img_names):
        if img_name.lower().endswith(('.bmp', '.dib', '.png', '.jpg', '.jpeg', '.pbm', '.pgm', '.ppm', '.tif', '.tiff')):
            image_path = os.path.join(dir_origin_path, img_name)
            image = Image.open(image_path)
            r_image = frcnn.detect_image(image)
            if not os.path.exists(dir_save_path):
                os.makedirs(dir_save_path)
            r_image.save(os.path.join(dir_save_path, img_name.replace(".jpg", ".png")), quality=95, subsample=2)
    else:
        raise AssertionError("Please specify the correct mode: 'predict', 'video', 'fps' or 'dir_predict'.")
```

三、评估 VOC07+12 的测试集

1. 设置评估参数：初始时设置一些计算过程中所需的参数，比如评估模式 `map_mode`、类别文件路径 `classes_path`、重叠阈值 `MINOVERLAP`、置信度阈值 `confidence`、非极大值抑制的 `IoU` 阈值 `nms_iou` 等。

```
# 受到mAP计算原理的限制，网络在计算mAP时需要获得近乎所有的预测框，这样才可以计算mAP
# 因此，confidence的值应当设置的尽量小进而获得全部可能的预测框。
#
# 该值一般不调整。因为计算mAP需要获得近乎所有的预测框，此处的confidence不能随便更改。
# 想要获得不同门限值下的Recall和Precision值，请修改下方的score_threshold。
#-----#
confidence      = 0.02
#-----#
# 预测时使用到的非极大抑制值的大小，越大表示非极大抑制越不严格。
#
# 该值一般不调整。
#-----#
nms_iou         = 0.5
#-----#
# Recall和Precision不像AP是一个面积的概念，因此在门限值不同时，网络的Recall和Precision值是不同的。
```

2. 通过指定 `map_mode` 的值，选择计算的内容。默认设置 `map_mode` 为 0 时代表整个 mAP 计算流程，包括获取预测结果、获取真实框、计算 VOC mAP。根据指定的 `classes_path` 加载目标类别信息。

3. 从验证集的文本文件中读取图像 ID，通过 `get_classes` 函数加载类别信息。

```

image_ids = open(os.path.join(VOCdevkit_path, "VOC2007/ImageSets/Main/val.txt")).read

if not os.path.exists(map_out_path):
    os.makedirs(map_out_path)
if not os.path.exists(os.path.join(map_out_path, 'ground-truth')):
    os.makedirs(os.path.join(map_out_path, 'ground-truth'))
if not os.path.exists(os.path.join(map_out_path, 'detection-results')):
    os.makedirs(os.path.join(map_out_path, 'detection-results'))
if not os.path.exists(os.path.join(map_out_path, 'images-optional')):
    os.makedirs(os.path.join(map_out_path, 'images-optional'))

class_names, _ = get_classes(classes_path)

```

4. 如果 map_mode 为 0 或 1 时，获取预测结果，加载 Faster R-CNN 模型，对验证集进行预测，将预测结果保存为特定格式的文件。

```

if map_mode == 0 or map_mode == 1:
    print("Load model.")
    frcnn = FRCNN(confidence = confidence, nms_iou = nms_iou)
    print("Load model done.")

    print("Get predict result.")
    for image_id in tqdm(image_ids):
        image_path = os.path.join(VOCdevkit_path, "VOC2007/JPEGImages/"+image_id+".jpg")
        image = Image.open(image_path)
        if map_vis:
            image.save(os.path.join(map_out_path, "images-optional/" + image_id + ".jpg"))
        frcnn.get_map_txt(image_id, image, class_names, map_out_path)
    print("Get predict result done.")

```

5. 如果 map_mode 为 0 或 2，遍历验证集图像，解析相应的 XML 文件，将真实框信息保存为特定格式的文件。

```

if map_mode == 0 or map_mode == 2:
    print("Get ground truth result.")
    for image_id in tqdm(image_ids):
        with open(os.path.join(map_out_path, "ground-truth/"+image_id+".txt"), "w") as new_f:
            root = ET.parse(os.path.join(VOCdevkit_path, "VOC2007/Annotations/"+image_id+".xml")).getroot()
            for obj in root.findall('object'):
                difficult_flag = False
                if obj.find('difficult')!=None:
                    difficult = obj.find('difficult').text
                    if int(difficult)==1:
                        difficult_flag = True
                obj_name = obj.find('name').text
                if obj_name not in class_names:

```

6. 如果 map_mode 为 0 或 3 时时计算 mAP: 调用 get_map 函数计算平均精度 mAP，并保存评估结果。

```

if map_mode == 0 or map_mode == 3:
    print("Get map.")
    get_map(MINOVERLAP, True, score_threshold = score_threshold, path = map_out_path)
    print("Get map done.")

```

实现结果:

一、 处理数据集（voc_annotation.py）运行结果:

1. 生成了训练需要使用的 2007_train.txt 和 2007_val.txt

```
voc_annotation x
C:\Users\GYH\.conda\envs\pytorch\python.exe D:/faster-rcnn-pytorch-master/voc_annotation.py
Generate 2007_train.txt and 2007_val.txt for train.
Generate 2007_train.txt and 2007_val.txt for train done.
```

2. 根据目标种类统计数量：输出了每个类别的目标数量。表格中的每一行代表一个类别，第一列是类别名字，第二列是该类别在数据集中出现的目标数量。

```
voc_annotation x
C:\Users\GYH\.conda\envs\pytorch\python.exe D:/faster-rcnn-pytorch-master/voc_annotation.py
Generate 2007_train.txt and 2007_val.txt for train.
Generate 2007_train.txt and 2007_val.txt for train done.
| bird | 1130 |
| bus | 782 |
| car | 2350 |
| cat | 1254 |
| dog | 1597 |

Process finished with exit code 0
```

二、 预测阶段（Predict.py）运行结果:

1. Mode = 'predict': （设置了 crop=true & count = true ）

```
predict x
Configurations.
-----
| keys | values |
-----
| model_path | logs/best_epoch_weights.pth |
| classes_path | model_data/voc_classes.txt |
| backbone | resnet50 |
| confidence | 0.5 |
| nms_iou | 0.3 |
| anchors_size | [8, 16, 32] |
| cuda | True |
-----
Input image filename: ./VOCdevkit/VOC2007/JPEGImages/2008_000134_bird.jpg
top_label: [0 0]
bird : 2 对目标的计数结果
classes_nums: [2. 0. 0. 0. 0.]
save crop_0.png to img_crop
save crop_1.png to img_crop
```

```
Input image filename:VOCdevkit/VOC2007/JPEGImages/2009_004224_car.jpg
top_label: [1 2 2 2 2]
bus : 1
car : 4
classes_nums: [0. 1. 4. 0. 0.]
save crop_0.png to img_crop
save crop_1.png to img_crop
save crop_2.png to img_crop
save crop_3.png to img_crop
save crop_4.png to img_crop
```

对目标的计数结果

图 9 Mode = 'predict'时 predict.py 运行结果

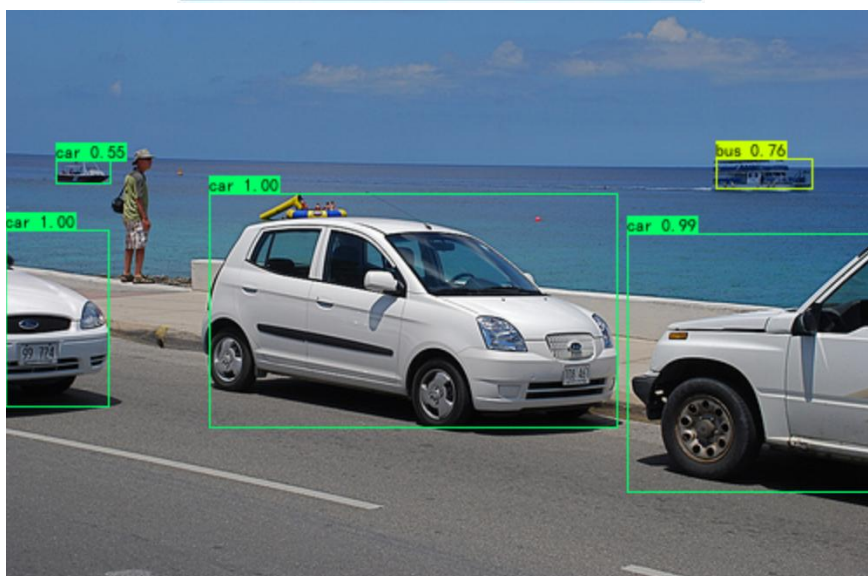
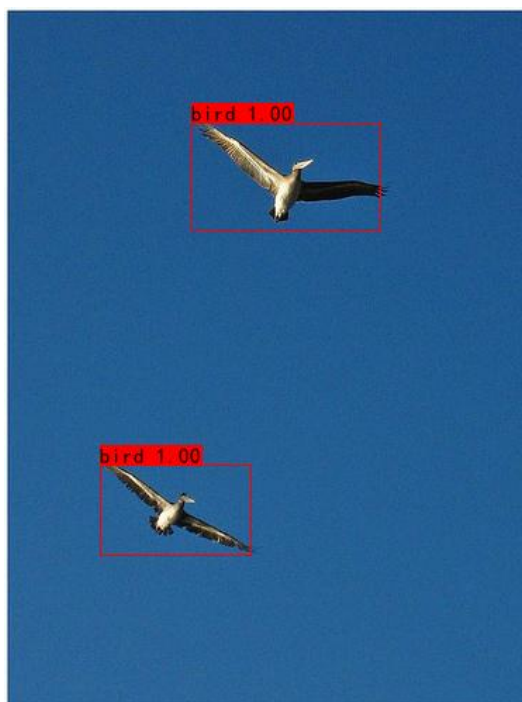


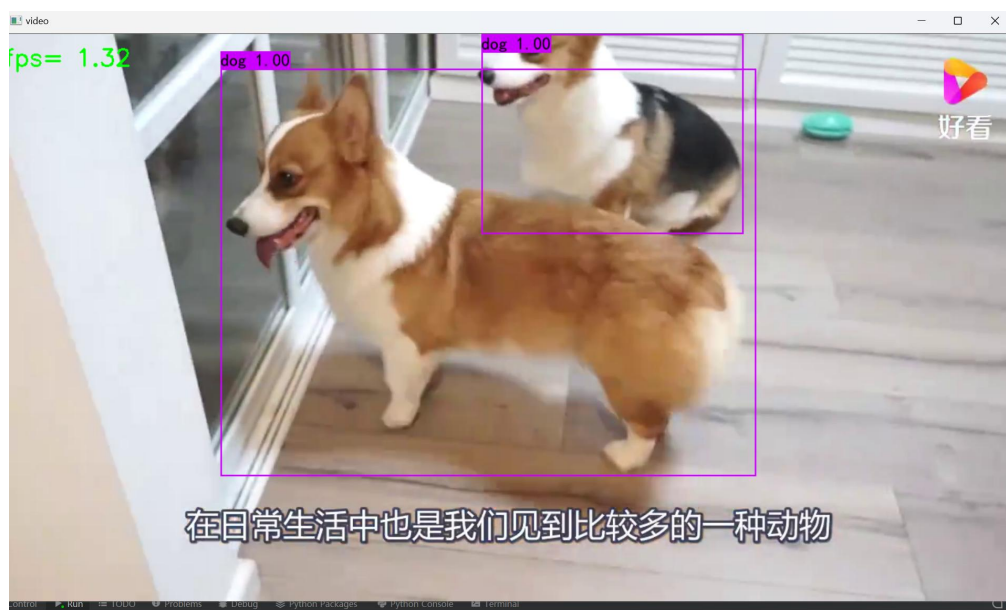
图 10 图像预测结果



图 11 单张图片预测后对目标进行截取

2. Mode = 'video':

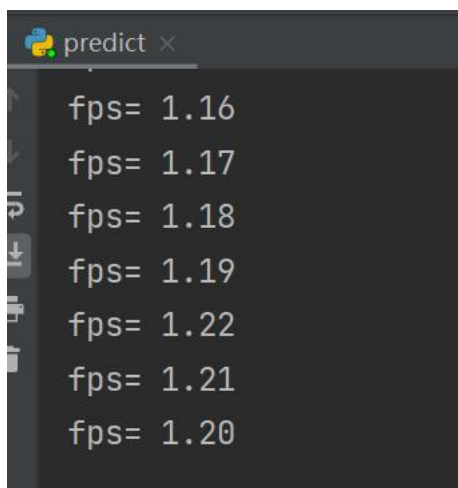
输入视频命名为 pets.mp4，输出视频命名为 out_pets.mp4



输入视频命名为 carvideo.mp4，输出视频命名为 out_carvideo.mp4



图 12 输出视频的实时识别



可以看出搭建的 Faster R-CNN 模型基本可以正确检测、定位出视频中的目标，并且对不同种类的目标都具有较好的识别功能；模型在视频中识别的准确率较高，但是会出现在视频的某一帧出现错误的检测框。从输出的 out_carvideo.mp4 中可以看出 Faster R-CNN 模型对目标的跟踪稳定性较强，目标在视频帧之间的

检测框连续性很高，在目标有运动或遮挡的情况下依旧可以做到检测框的稳定跟踪。且可以看出，两个视频中均存在多个目标，对于 Faster R-CNN 模型来说其具有同时处理多个目标，在背景相对稳定的情况下，可以在目标场景下同时正确处理多个目标。

但可以看到视频左上角显示的 fps 约为 1.3 左右，由于电脑硬件的限制，不能达到实时目标识别要求，系统的实时性能就相对较低，无法在实际应用中正常运行，有待利用高性能 GPU 加快模型推理速度和适当调整模型结构以减小模型的计算复杂度。

3. Mode = 'fps': 测试 Faster R-CNN 模型的帧率 (Frames Per Second, FPS)

```
logs/best_epoch_weights.pth model, anchors, and classes loaded.
Configurations:
-----
|               keys |               values|
-----
|   model_path   | logs/best_epoch_weights.pth|
| classes_path   | model_data/voc_classes.txt|
|   backbone     |         resnet50|
| confidence     |             0.5|
| nms_iou        |             0.3|
| anchors_size   |          [8, 16, 32]|
| cuda           |             True|
-----
0.6307302331924438 seconds, 1.5854638756390282FPS, @batch_size 1
```

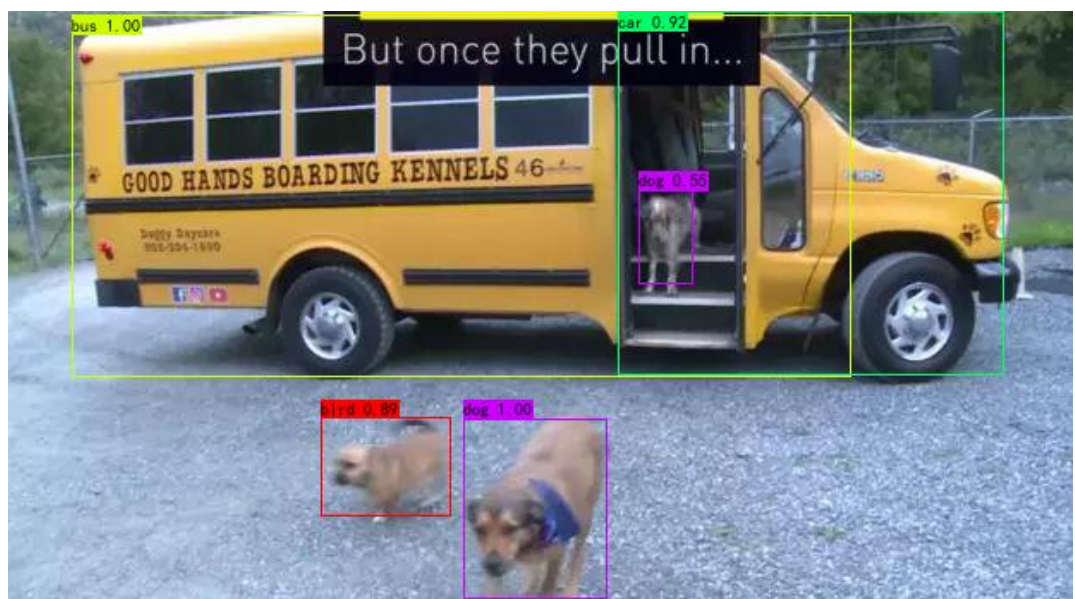
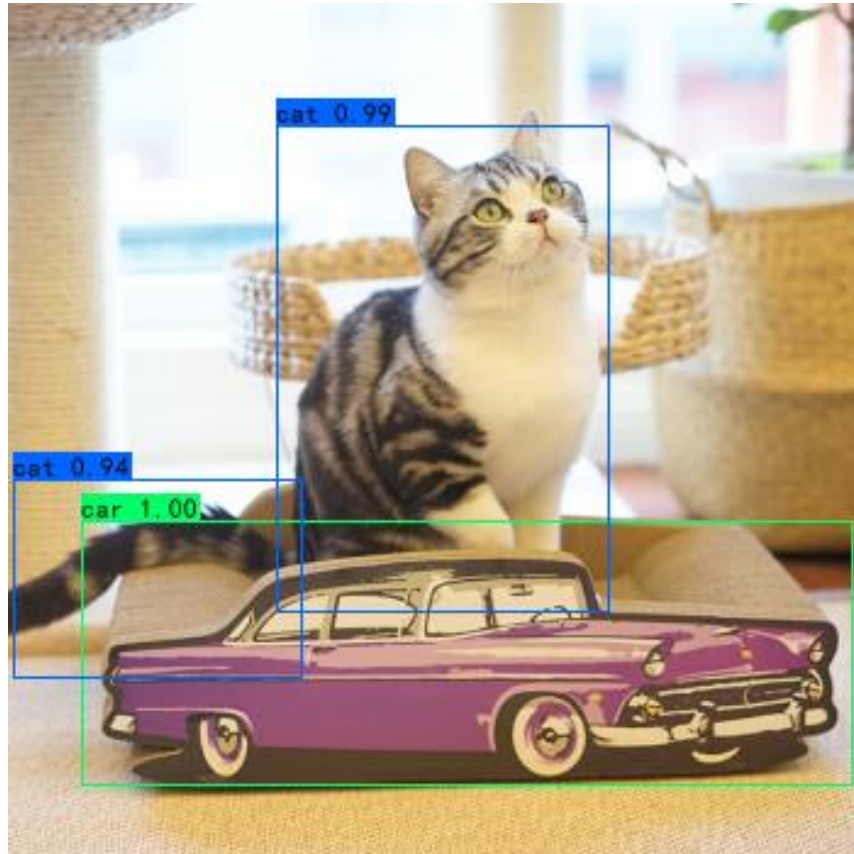
图 13 Mode='fps'时运行输出

从 tact_time 参数可以看出模型预测一张图片所花费的平均时间为约 0.66 秒。从 FPS 输出数值可以看出模型的平均帧率为约 1.52 帧每秒。以及批量大小 batch size 为 1。

模型的推理速度相对较慢，每秒处理约 1.52 帧图片。可能受到模型复杂性、硬件设备等因素的影响。单张图片推理时间为约 0.66 秒，对于实时性不是很高的场景，这个推理速度是比较快的，可以应用于一般的实用场景中。

4. Mode = 'dir_predict':

遍历文件夹进行检测并保存。默认遍历 img 文件夹，保存 img_out 文件夹
e.g.



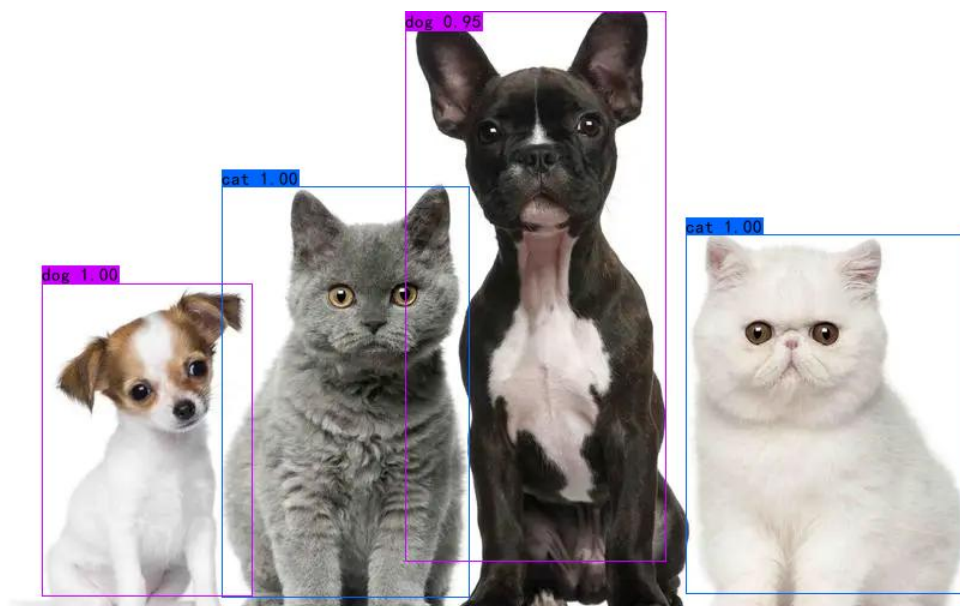
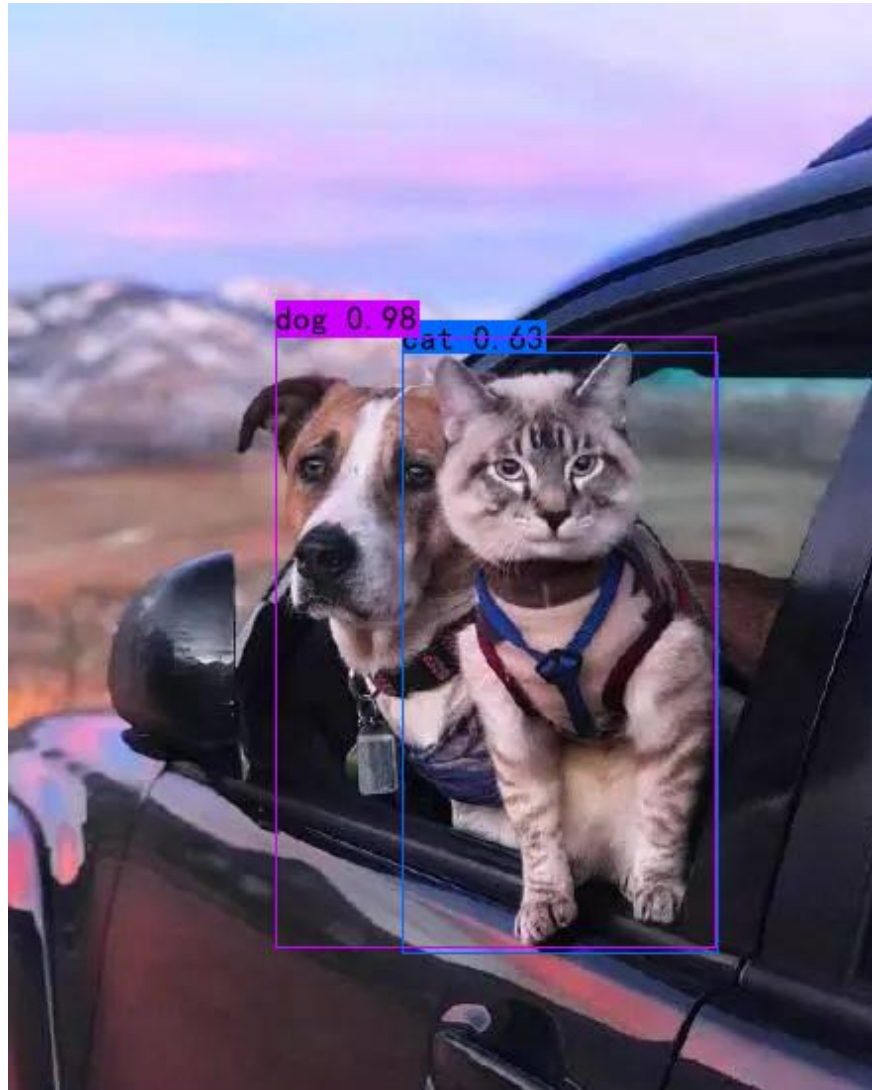




图 14 （组） 遍历文件夹输出的部分预测结果

输出图像用不同颜色的边界框框出了目标物体，并标注了类别信息和置信率，可视化效果较好，十分直观有利于结果分析。

经过人眼识别和分析，输出图像中真实目标标签和对应的边界框基本正确，显示置信率与实际验证情况相符，基本没有出现高置信率但是检测失误的情况。以及 Faster R-CNN 模型实现的目标检测准确度 Precision、mAP 值较高，而且对画面中的目标识别的比较全面，多目标识别的效果很好，几乎画面中真正的目标都可以被精准检测到。但由于文件夹中的图片一共只有 15 张，检测存在个别偏差，具体分析在后续评估步骤中分析。

通过分析，发现会有边界框范围不正确，少覆盖了一部分目标物体的情况，例如图 14 组的第 2 张图像中巴士车的框选范围缺失了车头部分，导致将车头单独标记为了一类目标物体 car。也发现存在将物体的一部分识别成另一个物体的情况。例如将狗的一部分识别成猫。其中 cat 和 dog 类、car 和 bus 类容易出现混淆。

三、评估阶段（get_map.py）运行结果：

运行 get_map.py: 大概需要 30min 左右完成运行

```
Run: get_map x
C:\Users\GYH\.conda\envs\pytorch\python.exe D:/faster-rcnn-pytorch-master/get_map.py
Load model.
logs/best_epoch_weights.pth model, anchors, and classes loaded.
Configurations:
-----
|               keys |               values|
-----
|      model_path | logs/best_epoch_weights.pth|
|   classes_path | model_data/voc_classes.txt|
|      backbone | resnet50|
|   confidence | 0.02|
|      nms_iou | 0.5|
| anchors_size | [8, 16, 32]|
|       cuda | True|
-----
Load model done.
Get predict result.
43%|██████████| 1017/2344 [13:42<18:04, 1.22it/s]
```

运行结果：

```
get_map x
|               keys |               values|
-----
|      model_path | logs/best_epoch_weights.pth|
|   classes_path | model_data/voc_classes.txt|
|      backbone | resnet50|
|   confidence | 0.02|
|      nms_iou | 0.5|
| anchors_size | [8, 16, 32]|
|       cuda | True|
-----
Load model done.
Get predict result.
100%|██████████| 2344/2344 [31:32<00:00, 1.24it/s]
0%|          | 0/2344 [00:00<?, ?it/s]Get predict result done.
Get ground truth result.
100%|██████████| 2344/2344 [00:18<00:00, 125.89it/s]
Get ground truth result done.
Get map.
90.82% = bird AP    || score_threshold=0.5 : F1=0.71 ; Recall=92.18% ; Precision=57.73%
90.12% = bus AP    || score_threshold=0.5 : F1=0.69 ; Recall=90.70% ; Precision=56.16%
83.47% = car AP    || score_threshold=0.5 : F1=0.65 ; Recall=86.83% ; Precision=52.42%
95.19% = cat AP    || score_threshold=0.5 : F1=0.73 ; Recall=98.23% ; Precision=58.45%
96.20% = dog AP    || score_threshold=0.5 : F1=0.75 ; Recall=97.60% ; Precision=61.35%
mAP = 91.16%
Get map done.
```

图 15 get_map.py 运行结果

根据输出结果分析：

1. 鸟类别：AP：90.82%

召回率：92.18%

F1（在 score_threshold=0.5 时）：0.71

精确率：57.73%

分析：鸟类别的 AP 很高，表明模型对鸟类别的检测效果较好。F1 分数为 0.71，显示在召回率和精确率之间取得了相对平衡的性能。

2.公交车类别：AP：90.12% F1（在 score_threshold=0.5 时）：0.69

召回率：90.70% 精确率：56.16%

分析：公交车类别的 AP 高，模型对公交车的检测效果较好。F1 分数为 0.69，同样显示了在召回率和精确率之间的平衡。

3.汽车类别：AP：83.47% F1（在 score_threshold=0.5 时）：0.65

召回率：86.83% 精确率：52.42%

分析：汽车类别的 AP 较高，但相对于其他类别稍低。说明识别汽车类别的平均精度低于其他类别。F1 分数为 0.65，召回率为 86.83%，精确率为 52.42%。其中召回率也相对较低，说明对于汽车类别来说误检率相对较高。

4.猫类别：AP：95.19% F1（在 score_threshold=0.5 时）：0.73

召回率：98.23% 精确率：58.45%

分析：猫类别的 AP 非常高，表明模型对猫类别的检测效果非常好。高 F1 分数和高召回率也强调了性能的优越性。

5. 狗类别：AP：96.20% F1（在 score_threshold=0.5 时）：0.75

召回率：97.60% 精确率：61.35%

分析：狗类别的 AP 非常高，显示模型在狗类别的检测方面表现出色。表明模型在图像中可以找到大部分正例，而且漏检率很低。

mAP 为 91.16%。

总体分析：模型在多个类别上表现良好，整体 mAP 值为 91.16%，显示了较高的目标检测性能。在每个类别中，F1 分数都相对平衡，召回率普遍较高，这是一个积极的迹象。对于一些类别，例如检测猫和狗的模型性能尤为显著，具有较高的 AP 和 F1 分数。

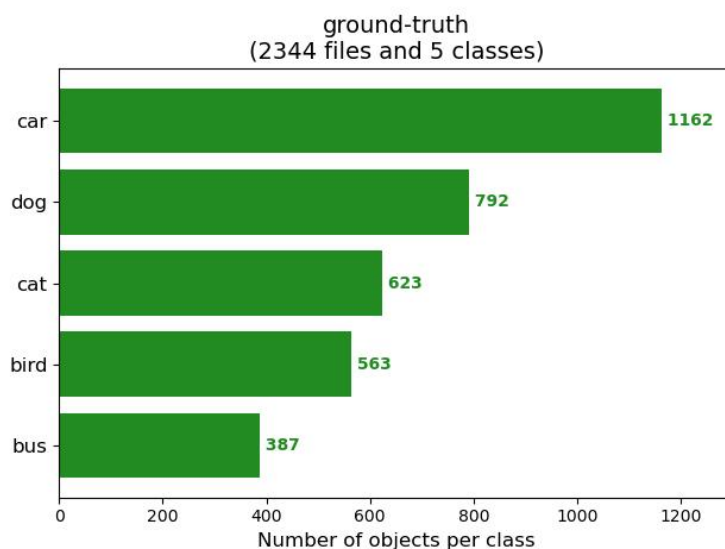


图 16 真实标签值

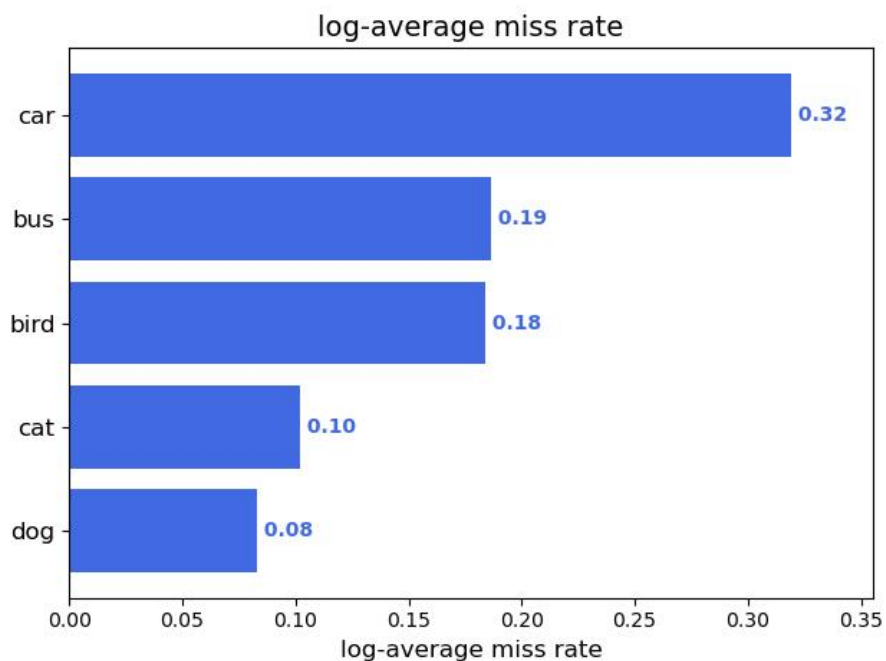


图 17 对数平均漏检率

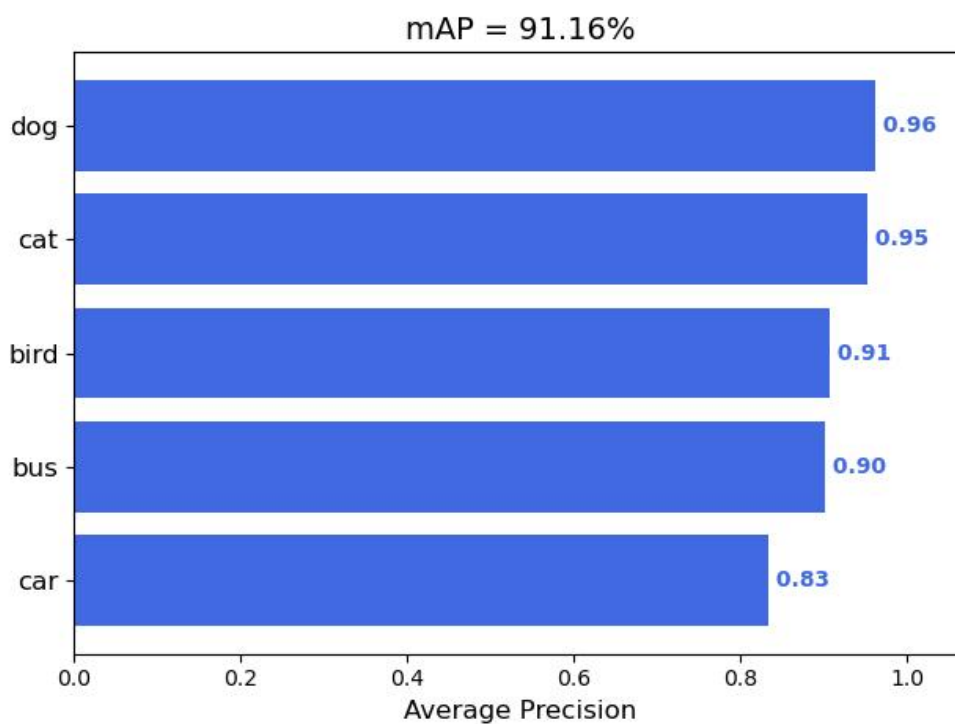


图 18 Faster R-CNN 模型检测平均精度

其中的对数平均漏检率 (Log-average missing rate) 是目标检测任务中用于评估漏检率的指标。漏检率是指模型未能检测到的目标的比例。从数据来看，dog 类别的对数漏检率最低，为 0.08，说明模型在检测狗的任务中表现最好。而 car 类别的对数漏检率最高，为 0.32，表示汽车类别是模型性能相对较差的类别，说明对汽车类别的检测失误概率更高，例如在上述例子中也可以看出有把海上的船

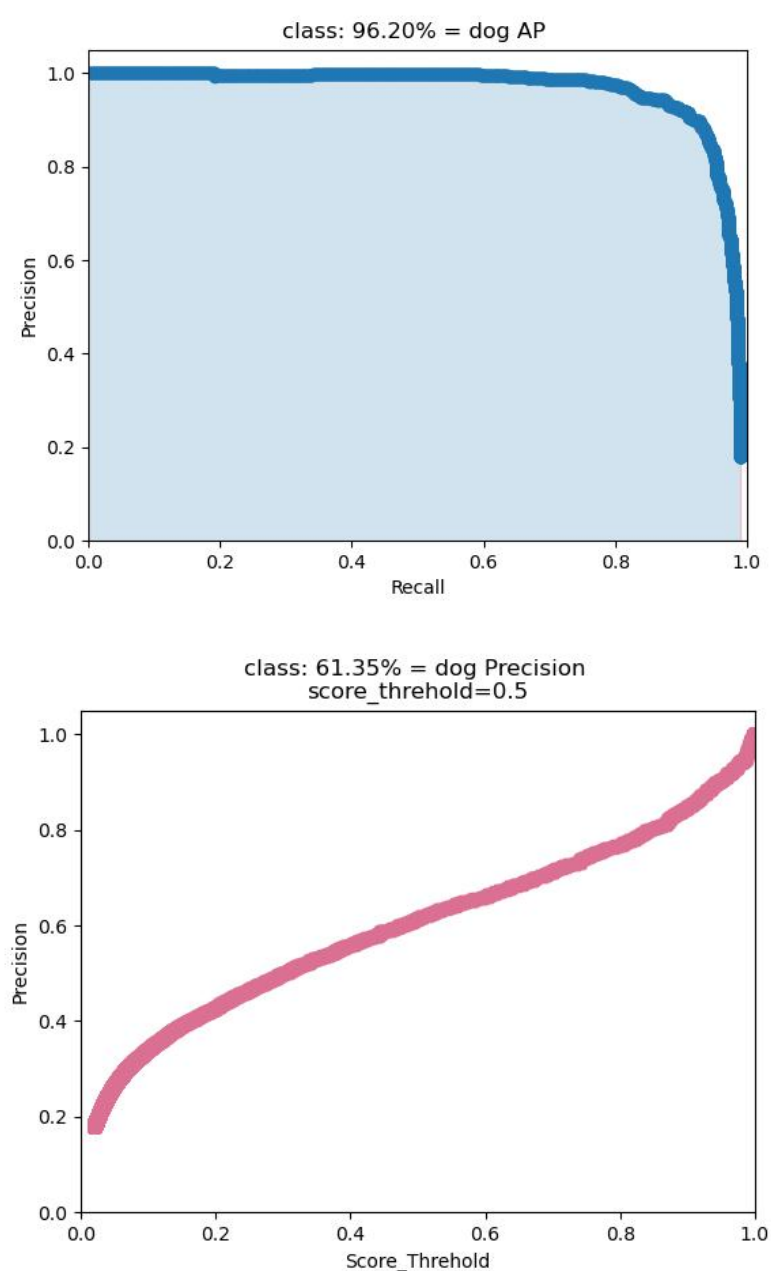
错误识别为汽车的案例。

AP 是一种用于度量模型在不同类别上性能的指标，通过计算 Precision-Recall 曲线下的面积来得到。对单独类别的 AP 进行计算，并对所有类别的 AP 取平均得到 mAP，可以看出猫和狗类别的检测精度较高，AP 最低的是车类别，精度为 0.83，但整体 mAP 为 91.16 是相当可观的数据，证明 Faster R-CNN 模型的性能良好，对目标类别的整体检测效果好。

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

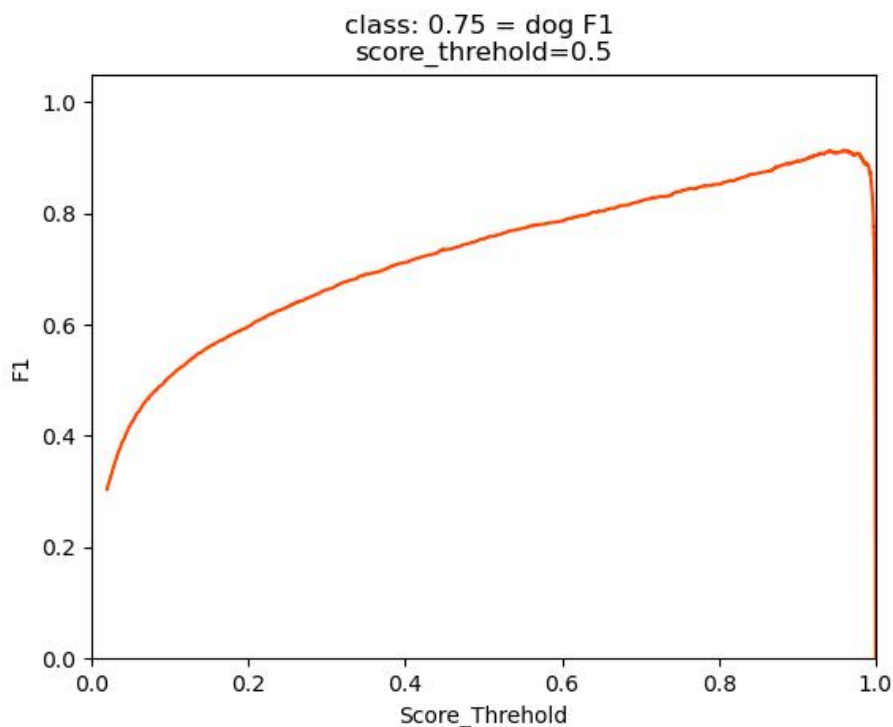
图 19 精度 Precision 的计算公式 TP 代表真正例 FP 代表假正例

附：dog 类别的评估数据（所有类别数据均放在\map_out\results 文件夹中）

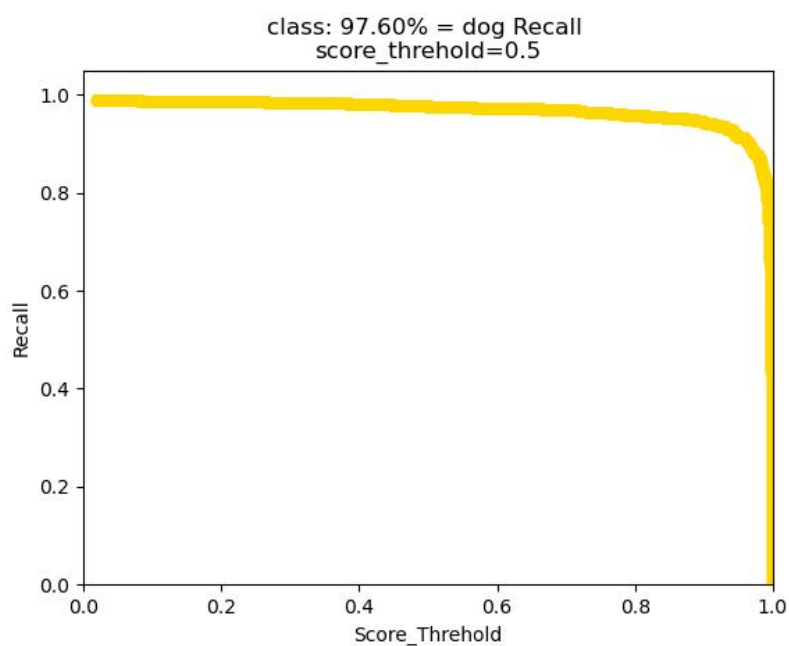


Precision 表示模型正确预测为目标区域占所有预测为目标区域的

比例。由图可知，当 `score_threshold` 较高时，模型更加谨慎，只对置信度较高的预测进行标记，因此 `Precision` 通常会增加。但是，这也导致了较低的 `Recall`。



F1 值是 `Precision` 和 `Recall` 的调和平均数，它在 `Precision` 和 `Recall` 之间取得平衡。上图结合 `Recall` 图像中可以看出，当 `score_threshold` 较低时，`Recall` 增加，但 `Precision` 通常下降，因为更多的预测可能是错误的。因此，F1 值在不同的 `score_threshold` 下呈现出不同的趋势。



Recall 表示被模型正确识别的目标数量占总实际目标数量的比例。从图中可以看出，当 `score_threshold` 较低时，模型会更容易预测出更多的目标，因此 Recall 趋近于 1。但是不能将 `score_threshold` 设置的过低，可能导致更多的误检，即将一些不是目标的区域错误地标记为目标。

结论分析:

问题:

1. 环境配置时，由于用到的 `.amp` 函数需要 `torch1.7` 以上才能正常使用 `fp16`，于是升级了初始下载的 `torch1.2`，之后遇到了下载的 CUDA 版本与 `torch` 版本不匹配的问题，以及 `torch` 版本与 `torchvision` 版本不匹配的问题，以及 `python` 版本过旧导致的不兼容问题。

解决方案：升级为 `torch1.7` 之后，卸载原始 `CUDA10.0` 版本，先重新安装了 `CUDA10.2`，在命令行中检查无误。继续按照 `PyTorch` 和 `torchvision` 的兼容性矩阵重新安装匹配的 `torch` 和 `torchvision`，最后升级 `python` 版本至 3.6 以上，再使用命令行更新所有导入的包以匹配新的 `python` 版本。

2. 目标检测时，`Faster R-CNN` 模型有时会把目标的其中一部分识别成另一个类别。例如，狗的下半身识别成猫，巴士的车前部识别成 `car`。

解决方案：在 `frcnn.py` 中调整 `nms_iou` 非极大值抑制参数，以减少重叠较大的框，确保每个目标只有一个检测结果。以及调整置信度阈值 `confidence`，使模型更加谨慎地预测目标。通过适当增加置信度阈值，可以降低误检率。最后通过多次测试，找到了相对适合大多数场景的最佳阈值。

```
#-----#
# 只有得分大于置信度的预测框会被保留下来
#-----#
"confidence"      : 0.6,
#-----#
# 非极大抑制所用到的nms_iou大小
#-----#
"nms_iou"         : 0.2,
```

图 20 Confidence 和 nms_iou 的调整

3. 在预测阶段，处理视频文件时，可以看到输出视频的左上角显示的 `fps` 约为 1.3 左右，由于电脑硬件的限制，不能达到实时目标识别要求，系统的实时性能就相对较低，无法在实际应用中正常运行。

解决方案：可以利用高性能 `GPU` 加快模型推理速度，或者使用轻量级模型、剪枝技术或量化模型，以减小模型的计算复杂度。以及在可能的情况下，利用并行计算的能力提高处理效率，尽可能在实际情况下采取适当的性能优化策略。

4. 在评估阶段，通过数据分析可以看出 `car` 类别的对数平均漏检率为 0.32，在所有类别中最高，说明图像中的目标漏检率很高。

解决方案：在后续的测试运行中增加置信度阈值尽可能减少假阳性，从而降低漏检率。选择适当提高置信度阈值以确保只有置信度较高的预测被接受。从参

数调度完成调整之外,也可以考虑调整 Faster R-CNN 模型的超参数,如 anchor 大小、网络层数、学习率等。并提高训练时的 Epoch 数,更多的训练次数有助于模型更好地收敛,并提高对目标的检测能力。

实验收获:

在学习图像处理与深度学习这门课的最后,我选择了搭建 Faster R-CNN 模型实现目标检测来作为我的大作业作品,让我进一步深入学习领会到了深度学习搭建神经网络的知识,加深了对深度学习这一热点方向的认识,也在实操当中提高了 python 的代码调适能力,对于输出评估数据的分析能力也相应地有所提高。

在配置本次大作业环境时,我用到了 PyTorch 这一灵活且强大的深度学习框架,使得实现和调试目标检测模型变得相对简单。做到在 Faster R-CNN 中,两阶段的设计使得模型具备更强的准确性和鲁棒性。

在大作业实操过程中,我深入学习了 Faster R-CNN 模型的核心原理,包括区域建议网络(RPN)、ROI 池化层等组成部分。通过构建和训练模型,我更好地理解目标检测中的 anchor-based 方法,并学到如何在训练过程中优化损失函数以提高模型性能。在实验中,调整超参数如置信度阈值、非极大值抑制(NMS)的 IoU 阈值等,我通过调整了解它们对模型性能的影响,体会到这些参数对目标检测识别中 precision、AP 等评估指标的影响。

最后,通过评估模型在验证集上的性能,我了解了如何使用 Recall, mAP 等指标来客观地评价目标检测模型的质量。这种实验经验为我深入理解目标检测算法奠定了基础,也为今后进一步学习和研究图像处理与深度学习方向的项目提供了坚实的基础。

参考文献:

Blog: https://blog.csdn.net/weixin_44791964/article/details/106037141
<https://zhuanlan.zhihu.com/p/611298538>
<https://wenku.csdn.net/answer/638aee6ce78a11edbc5fa163eeb3507>
<https://zhuanlan.zhihu.com/p/660531880>
https://blog.csdn.net/weixin_44791964/article/details/104702142

论文: Faster R-CNN Towards Real-Time Object Detection with Region Proposal Networks

资源网站:

<https://www.anaconda.com/distribution/>
<https://mirrors.tuna.tsinghua.edu.cn/> (清华开源镜像网站)
https://download.pytorch.org/whl/torch_stable.html
<https://github.com/chenyuntc/simple-faster-rcnn-pytorch>
<https://github.com/eriklindernoren/PyTorch-YOLOv3>
https://github.com/BobLiu20/YOLOv3_PyTorch

视频讲解：

<https://www.bilibili.com/video/BV1hK411p7Ki/>

https://www.bilibili.com/video/BV1af4y1m7iL/?spm_id_from=333.337.search-card.all.click

https://www.bilibili.com/video/BV1BK41157Vs/?spm_id_from=333.337.search-card.all.click

致谢：

感谢授课老师聂笑盈老师在课堂上的悉心教导与栽培，感谢钱奕同学的交流指导，感谢网络资源的便利，让我在网络上获得了许多学习的资源与共同交流的伙伴。此外，感谢 CSDN 用户 Bubbliiiiing 提供的 VOC 数据集，为大作业的完成提供了重要的基础。最后，万分感谢上述提到的帮助与指导，以顺利完成此次大作业任务。