基于MMDETECTION框架和VOC数据集训练并测试目标检测模型

金潇睿

项目主页: https://github.com/Anderasderry/2025-CV-Midterm-Project

一、摘要

本实验选择使用 OpenMMLab 的目标检测框架 MMDetection 完成 Mask R-CNN 和 Sparse R-CNN 两种模型在 Pascal VOC 2012 数据集上的训练和测试，并通过 TensorBoard 可视化相关训练进程，比较两种模型的实例分割与目标检测结果。

二、 实验环境

- **操作系统**：Windows 11
- **Python 版本**：3.8
- **PyTorch 版本**：2.0.0
- **CUDA 版本**：12.1
- **GPU**：NVIDIA L20 (48GB)
- **CPU**：20 vCPU Intel(R) Xeon(R) Platinum 8457C
- 需要安装 tensorboard，mmcv，mmengine

三、数据集与预处理

PASCAL VOC（Pattern Analysis, Statistical Modelling and Computational Learning Visual Object Classes）是一个广泛使用的计算机视觉数据集，它为对象识别、对象检测、对象分割、人体布局和行为分类等任务提供了标准化的图像数据集。本实验使用 VOC 2012 子数据集，共有 20 类目标，根据 `ImageSets/Main/` 目录下的 `train.txt` 和 `val.txt` 划分训练集和验证集。

数据集路径结构如下：

```
VOCdevkit/
└── VOC2012/
    ├── Annotations/
    ├── JPEGImages/
    ├── ImageSets/
    └── SegmentationClass/
```

为了让数据集适配 MMDetection 框架下的目标检测和实例分割模型训练，我们利用 `voc2coco.py` 将数据集转换为 COCO 格式。转换过程中，脚本首先从 VOC 的 `ImageSets/Segmentation` 文件夹中读取训练集和验证集的图像列表，然后依次处理每张图像及其对应的标注文件与分割掩码图。对于每张图像，脚本会复制原始图像到指定目录下，并提取图像尺寸等信息，写入 COCO 格式的 JSON 文件中。

在目标标注的处理上，脚本读取每张图像对应的 XML 文件，提取其中属于 VOC 定义的 20 个标准类别的目标。每个目标包含一个边界框（bounding box），用于定位该目标在图像中的位置。与此同时，脚本还会读取 `SegmentationObject` 中的分割掩码图，通过将边界框转换为一个二值掩码，与分割图中的每个实例掩码计算 IoU（交并比），从中选出与边界框重合度最高的实例掩码。若该 IoU 低于设定的阈值（本实验中设定为 0.2），则认为匹配失败，不将该目标纳入最终标注。

一旦成功匹配到实例掩码，脚本使用 `pycocotools` 将其转换为 COCO 所要求的 RLE（游程编码）格式，并将对应的类别、边界框、掩码和目标面积等信息整理成 annotation 项，加入到 COCO 的 JSON 标注结构中。最后，转换完成的数据以标准 COCO 格式保存到输出目录中，包含图像信息和标注信息，结构上与 COCO 官方数据集保持一致。

转化后的数据集路径结构如下：

```
data/
└── coco/
    ├── annotations/
        ├── instances_train2017.json
        ├── instances_val2017.json
    ├── train2017/VOC2012/JPEGImages
    └── val2017/VOC2012/JPEGImages
```

四、训练流程

1. Mask R-CNN

在 `mmdetection/configs/mask_rcnn/` 目录下找到配置文件 `mask-rcnn_r50_fpn_1x_coco.py`，以该文件为基础配置训练参数。在命令行输入如下指令：

```
python tools/train.py configs/mask_rcnn/rcnn_r50_fpn_1x_coco.py
```

在 `work_dirs/` 目录下生成新的配置文件 `mask-rcnn_r50_fpn_1x_coco/mask-rcnn_r50_fpn_1x_coco.py`，进行部分修改，本实验中配置如下：

(1) 模型结构

- **Backbone**：ResNet-50
  - 使用 `torchvision` 提供的预训练权重（`init_cfg`），冻结第一个 stage（`frozen_stages=1`）
  - 输出 4 层特征图（`out_indices=(0,1,2,3)`），用于后续的 FPN.
- **Neck**：FPN (Feature Pyramid Network)
  - 输入通道为 `[256, 512, 1024, 2048]`，输出统一为 256 通道，构建 5 层金字塔特征图。
- **RPN Head**（区域提议网络）：
  - 使用 Anchor Generator 生成 anchors.
  - 分类损失为 `CrossEntropyLoss(use_sigmoid=True)`，回归损失为 `L1Loss`.
  - IoU assigner 阈值：正样本 IoU ≥ 0.7，负样本 IoU < 0.3
- **RoI Head**（两阶段检测与分割）：
  - `Shared2FCBBoxHead` 进行边界框分类与回归，类别数设为 20
  - `FCNMaskHead` 执行分割任务，4 层卷积输出二值掩码，类别数同样为 20
  - RoI 特征提取使用 `RoIAlign`，bbox 尺寸为 7×7，mask 尺寸为 14×14

(2) 数据集与批量设置

- **数据类型**：使用转换为 COCO 格式的 Pascal VOC2012 数据。
- **类别数**：共 20 类，与 VOC 一致。
- **图片尺寸**：统一调整至 `(1333, 800)`。
- **训练集**：`data/coco/annotations/instances_train2017.json`
- **验证集与测试集**：`instances_val2017.json`
- **Batch Size**：训练时取`batch_size=2`，验证与测试时取`batch_size=1`
- **采样器**：
  - 使用带 `AspectRatioBatchSampler` 的 `DefaultSampler

(3) 训练超参数

- **优化器（Optimizer）：SGD**
  - 学习率（`lr`）：0.02（假设总 batch size 为 16，实际每卡为 2，未启用自动缩放
  - 动量（`momentum`）：0.9
  - 权重衰减（`weight_decay`）：0.0001
- **学习率调度器（Param Scheduler）**：
  - `LinearLR`：warm-up 阶段线性增长，持续 500 iter，从 $0.001\times$ base_lr 起。
  - `MultiStepLR`：训练中期后，降低学习率（`gamma=0.1`）。
- **最大训练轮数（Epochs）：30**

(4) 损失函数

- **RPN阶段**：

    - 分类损失：Sigmoid Cross Entropy
    - 回归损失：L1 Loss
- **RCNN阶段**：
    - 分类损失：Softmax Cross Entropy
    - 回归损失：L1 Loss（`target_stds = [0.1, 0.1, 0.2, 0.2]`）

(5) 评价指标

- 使用 `CocoMetric` 计算**目标检测**与**实例分割**效果，输出标准为官方评价指标 mAP.

(6) 其他设置

- 在`train_dataloader`，`val_dataloader`和`test_dataloader`中手动设置了 VOC 数据集的标签。

```
metainfo = dict(
classes=('aeroplane','bicycle','bird','boat','bottle',
        'bus','car','cat','chair','cow',
        'diningtable','dog','horse','motorbike','person',
        'pottedplant','sheep','sofa','train','tvmonitor',))
```

- **日志与可视化**：每 50 iter 记录一次日志，并加入了 TensorBoard 模块进行可视化。
- **保存模型**：
    - 每 10 个 epoch 保存一次模型权重。

2. Sparse R-CNN

在`mmdetection/configs/sparse_rcnn/`目录下找到配置文件`sparse-rcnn_r50_fpn_1x_coco.py`，以该文件为基础配置训练参数。在命令行输入如下指令：

```
python tools/train.py configs/sparse_rcnn/sparse-rcnn_r50_fpn_1x_coco.py
```

在`work_dirs/`目录下生成新的配置文件`sparse-rcnn_r50_fpn_1x_coco/sparse-rcnn_r50_fpn_1x_coco.py`，进行部分修改，本实验中配置如下：

(1) 模型结构

- **Backbone**：ResNet-50
- **Neck**：FPN (Feature Pyramid Network)
- **与 Mask R-CNN 的区别**：

- Sparse R-CNN 不使用传统 Anchor Generator，而是学习一组固定数量的初始 proposals（如 100 个）。
- Proposal 初始化使用 `init_cfg` 设置，后续由 Dynamic Instance Interactive Head（DIIHead）迭代更新。
- 同样使用 `RoIAlign` 提取特征，尺寸为 7×7，但是无 mask.

(2) 数据集与批量设置：与 Mask R-CNN 相同配置

(3) 训练超参数

- **优化器（Optimizer）：AdamW**
  - 学习率（`lr`）：0.0001
  - 权重衰减（`weight_decay`）：0.0001
- **学习率调度器（Param Scheduler）**：与 Mask R-CNN 相同配置
- **最大训练轮数（Epochs）：30**

(4) 损失函数

- **类别预测**：分类损失使用 Focal Loss，用于处理正负样本比例不均。
- **边框回归**： 回归损失使用GIoU Loss，增强位置敏感性。
- **Hungarian Assigner**：基于分类分数 + GIoU + L1 距离进行正负样本匹配

五、 训练结果和可视化分析

运行指令：

```
python tools/train.py work_dirs/rcnn_r50_fpn_1x_coco/mask-
rcnn_r50_fpn_1x_coco.py`
```

将对模型进行 30 轮次训练，每 10 轮保存一次模型权重，最后一次保存的模型权重为 `epoch_30.pth`，运行指令：

```
python tools/test.py $...$/mask-rcnn_r50_fpn_1x_coco.py $...$/epoch_30.pth --
show-dir visual_results
```

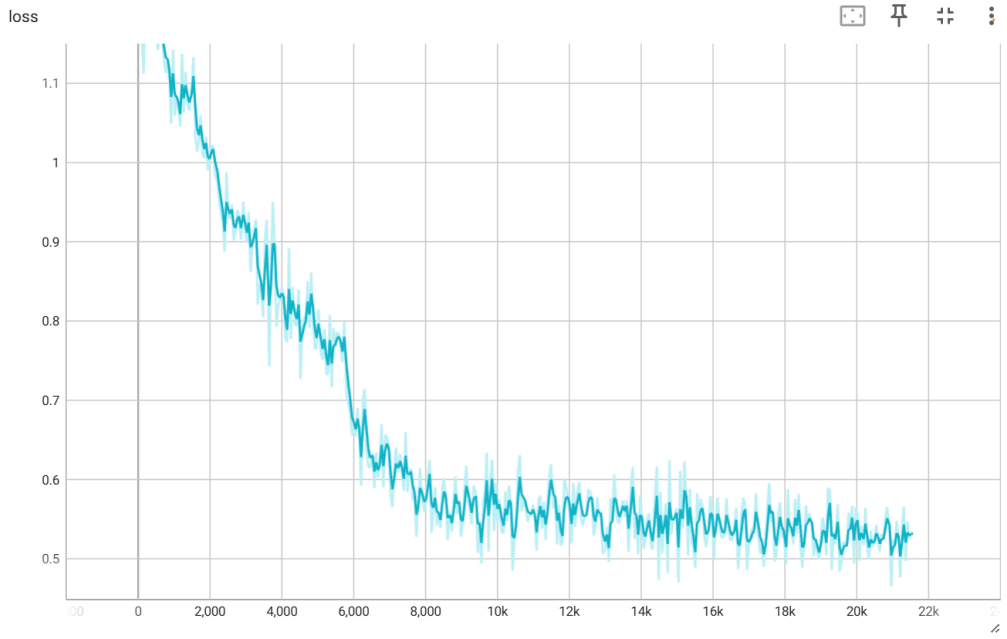
对模型进行测试，训练和测试的记录都保存在`work_dirs/`目录下。

运行指令：

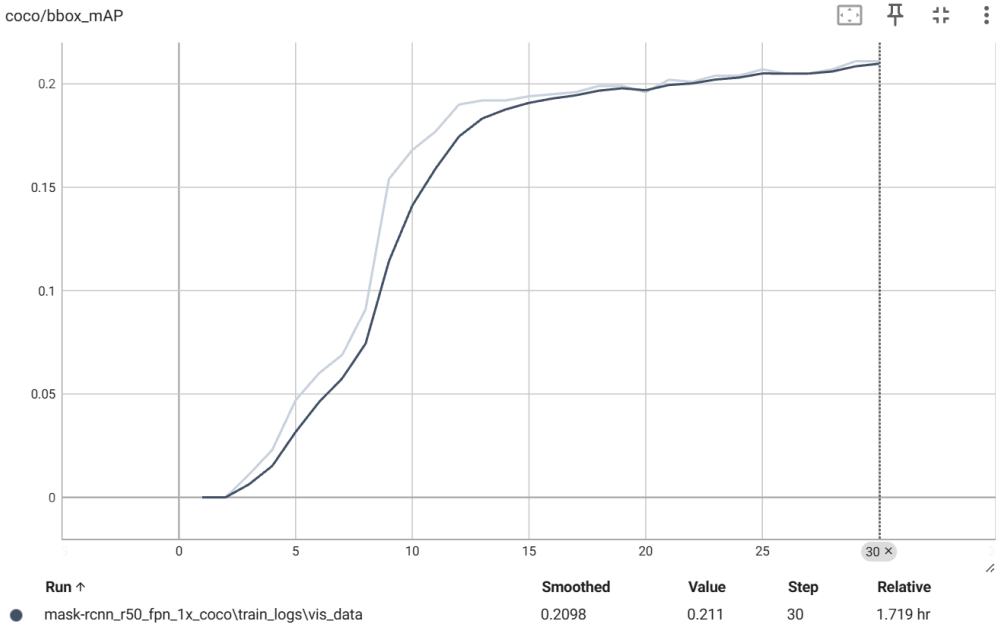
```
tensorboard --logdir work_dirs/
```

观察在训练集上的 loss 曲线和验证集上的mAP曲线。

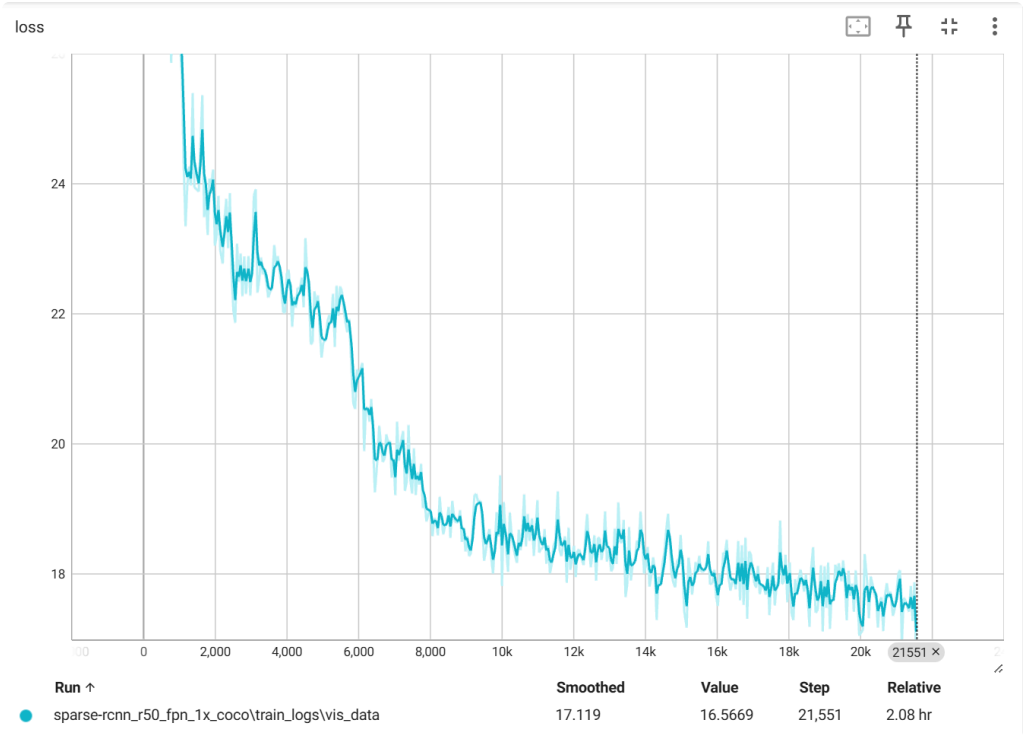
- **Mask R-CNN 在训练集上的 loss 曲线**：下降至收敛，最终接近0.5



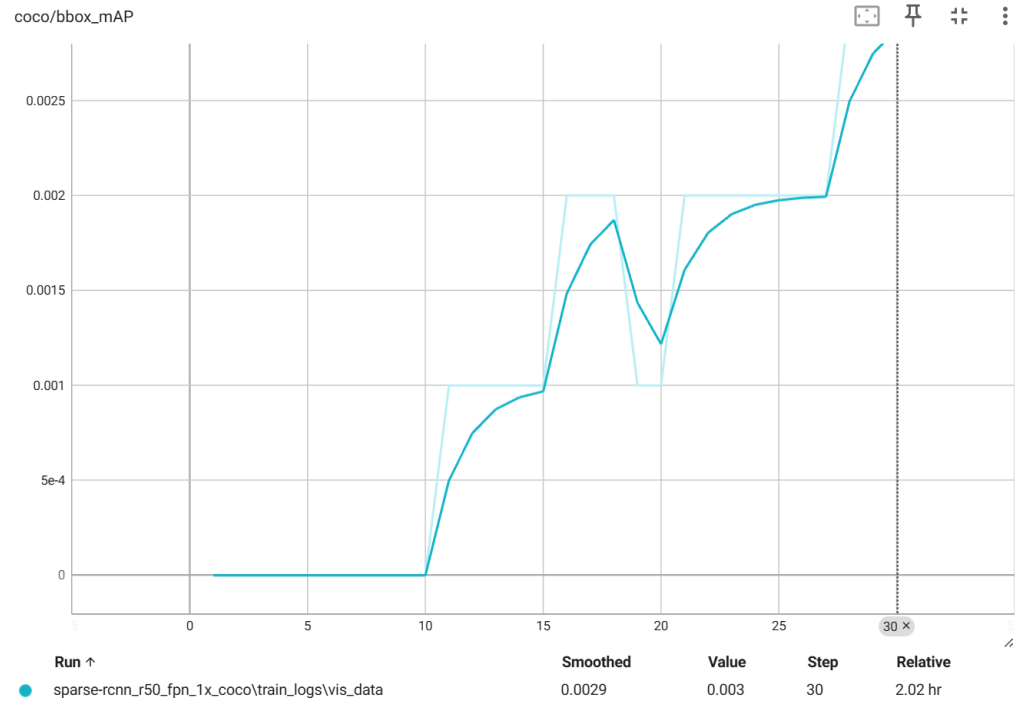
- **Mask R-CNN 在验证集上的 mAP 曲线**：增速渐缓，预计在更多轮次的训练中能够收敛



- **Sparse R-CNN 在验证集上的 loss 曲线**：逐渐下降，但仍维持在较高水平，可能配置有误，导致模型无法达到预期的学习效果



- **Sparse R-CNN 在验证集上的 mAP 曲线**：整体水平较低，训练没有达到预期效果



由于训练好的 Sparse R-CNN 模型未达到预期效果，无法完成目标检测任务，接下来只对 Mask R-CNN 模型进行进一步可视化分析。在测试集中挑选 4 张图像，对比它们在 Mask R-CNN 第一阶段产生的 proposal box 和最终的预测结果。其中利用脚本`visualize.py`生成了第一阶段产生的 proposal box 图像：将输入图像保存在`visualization/in/`目录下，运行`visualize.py`，会把 proposal box 和最终结果分别保存至`visualization/out/`目录下的`proposal/`和`prediction/`，可视化如下：

- **proposal box**

图1（proposal）

图2（proposal）

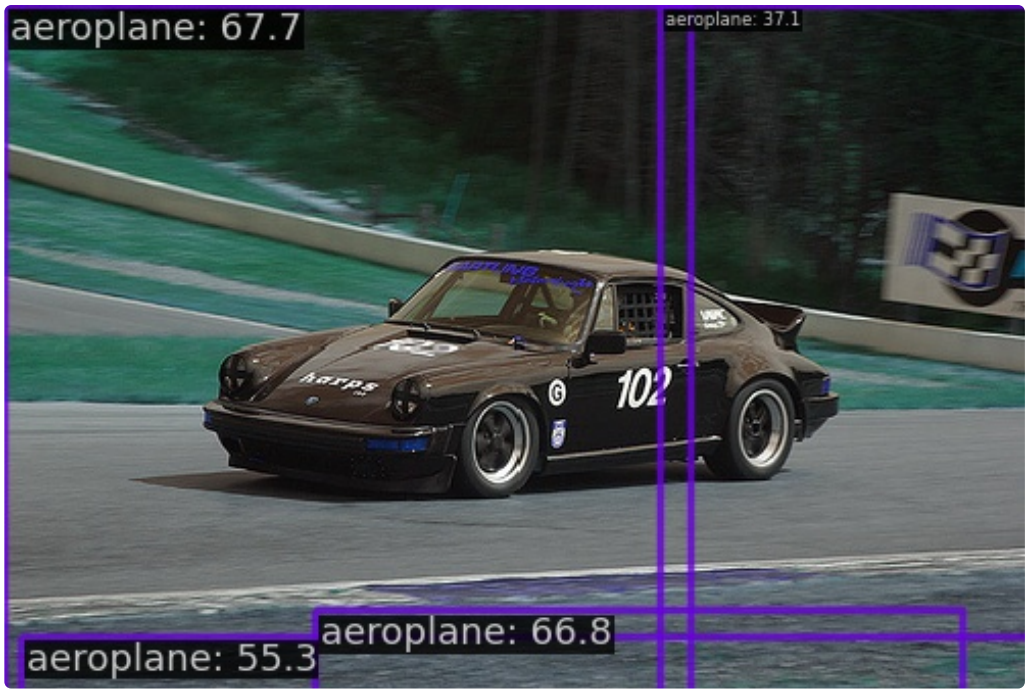


图3（proposal）

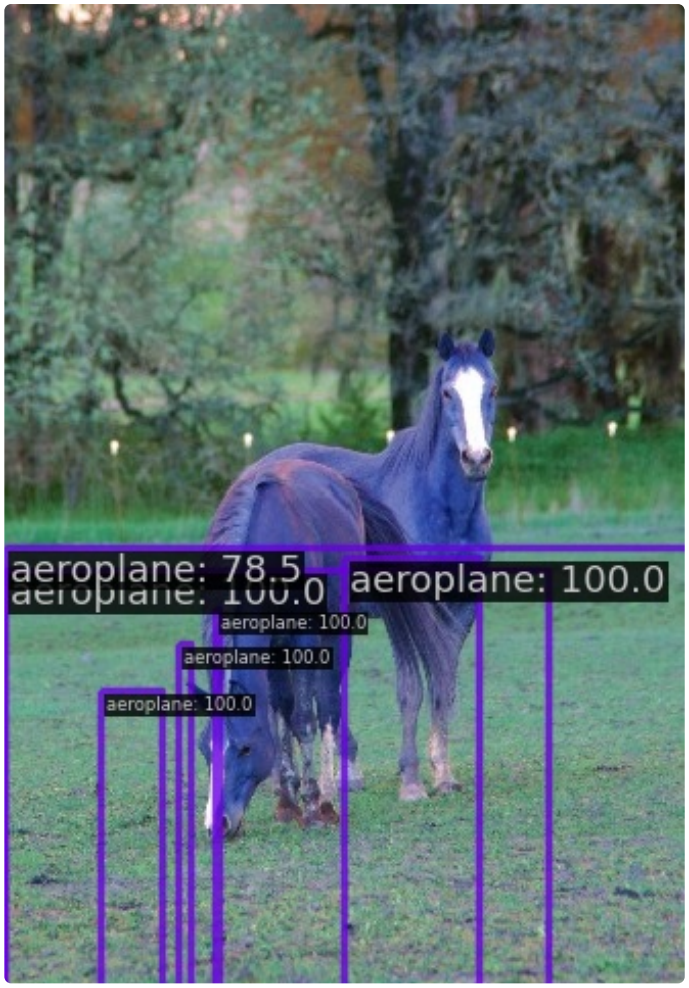


图4（proposal）

- **prediction**

图1（prediction）

图2（prediction）

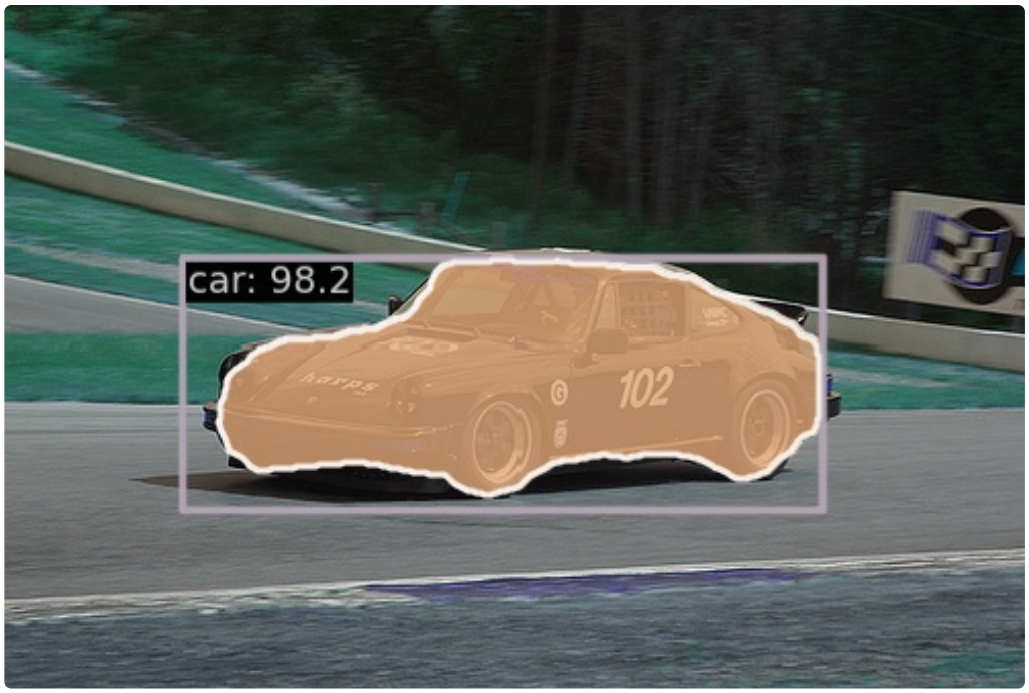


图3（prediction）

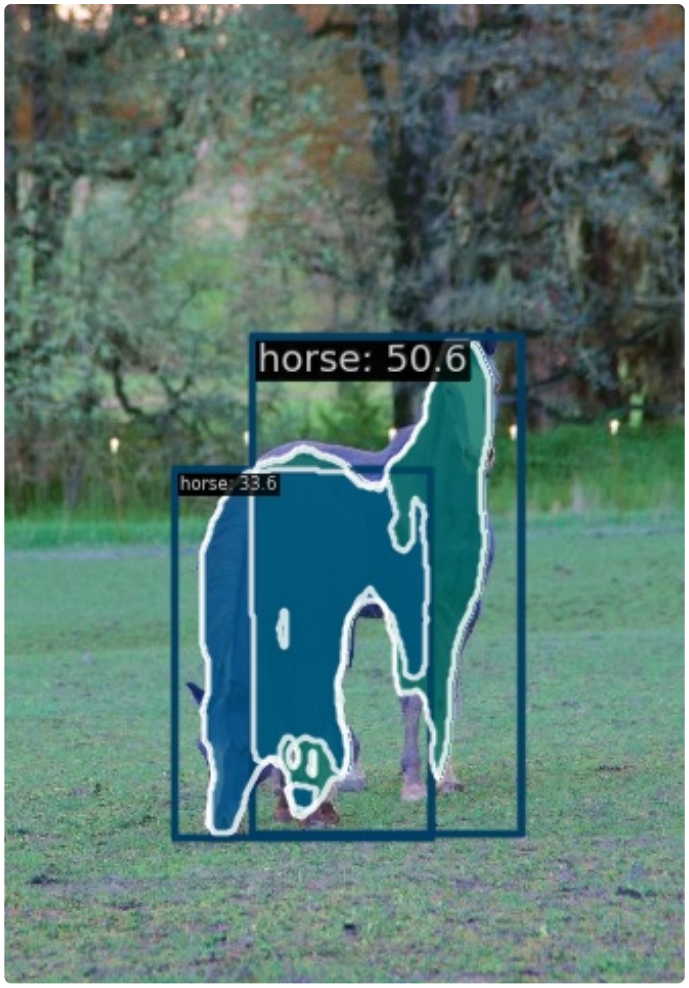


图4（prediction）

可以发现，在最终预测结果中，目标检测结果基本准确，偶尔出现额外识别出图像中并不存在的物体的情况；在实例分割中，对于背景颜色较为简单，物体颜色与背景颜色区别较明显时（如上面例子中的 car 和 horse），模型能较好地识别物体的边界并进行分割，而对其他情况也能大致分割出目标的主要部分（如上面例子中的 person），模型基本达到预期效果。

最后，搜集三张不在 VOC 数据集内且包含有 VOC 中类别物体的图像（jpg格式），利用 MMDetection 提供的 `image_demo.py` 脚本，在训练好的模型上进行测试，在 `mmdetection/` 目录下运行指令：

```
python demo/image_damo.py $图片.jpg$ $配置文件.py$ --weights $模型权重.pth$ --
show
```

可视化结果如下：

- **original images**

图5（original）

图6（original）

图7（original）

- **precdiction**

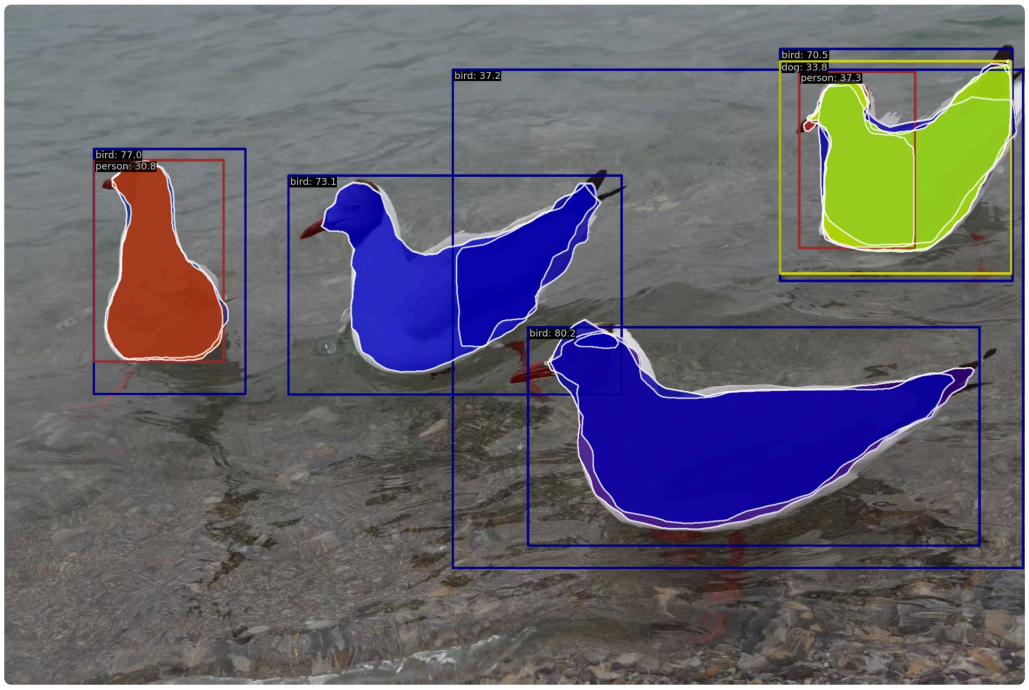


图5（prediction）

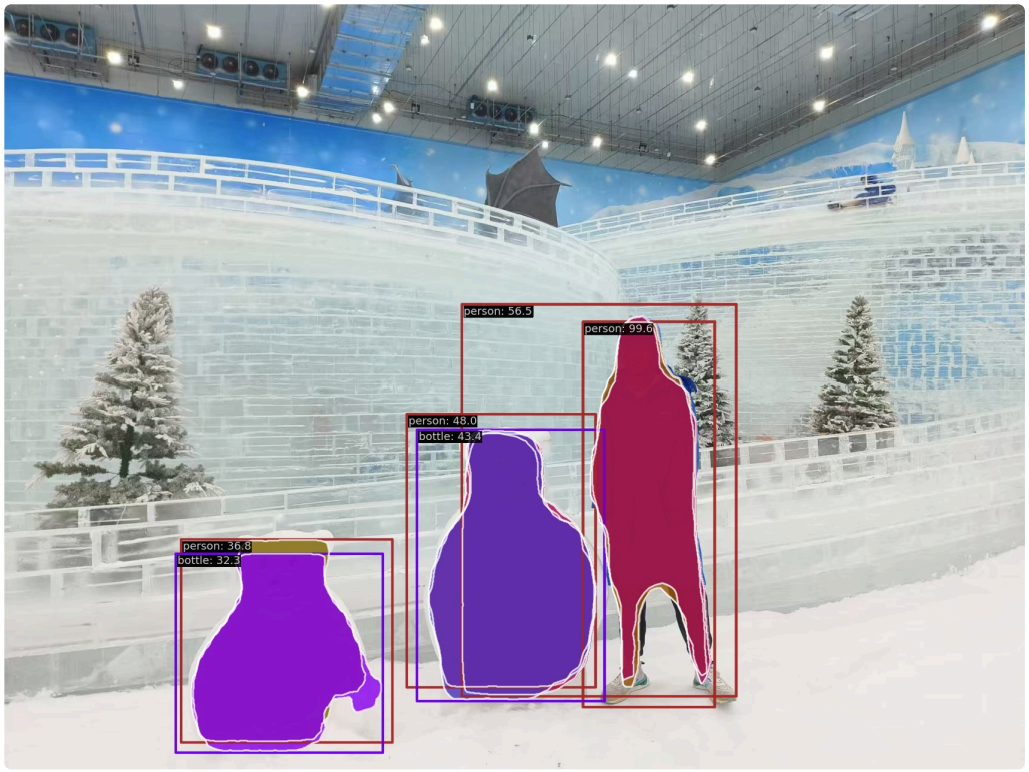


图6（prediction）

图7（prediction）

可以发现模型在 VOC 数据集之外的图像上也有良好的表现，目标检测和实例分割准确率都较高，其中图 2 包含不属于 VOC 数据集的类的物体，因此出现识别错误，但能够准确识别 person 类。综上所述，训练后的 Mask R-CNN 在 VOC 数据集上取得良好表现，而本实验的 Sparse R-CNN 未能达到预期效果，可能需要修改参数配置并增加训练轮次。

六、附录

- GitHub Repo：Anderasderry/2025-CV-Midterm-Project
- 模型权重：Midterm2， 提取码: **n4cm**