

电子科技大学

实验报告

学生姓名：谢卿云 学号：2022010910017 指导教师：沈复民

一、实验项目名称：

Scene Recognition with Bag of Words

二、实验原理：

（一）目标检测概述

目标检测是计算机视觉领域的一个重要分支，旨在识别图像或视频中特定目标的位置和类别。它结合了图像分类和目标定位两项任务，是许多高级视觉应用（如自动驾驶、智能监控、机器人视觉等）的基础。

（二）目标检测算法分类

目标检测算法大致可以分为两类：

- **两阶段检测器 (Two-stage Detectors):** 这类算法首先生成候选区域（Region Proposals），然后对这些候选区域进行分类和边界框回归。代表算法有 R-CNN 系列（R-CNN, Fast R-CNN, Faster R-CNN）。
- **单阶段检测器 (One-stage Detectors):** 这类算法直接在图像上预测目标的类别和边界框，无需生成候选区域。代表算法有 YOLO 系列、SSD、RetinaNet、FCOS 等。

（三）MMDetection 工具箱

MMDetection 是一个由 OpenMMLab 开发的开源目标检测工具箱，它基于 PyTorch 实现。MMDetection 提供了丰富多样的 SOTA (State-of-The-Art) 目标检测算法实现，包括本文实验中使用的 YOLO 系列、Faster R-CNN 和 FCOS 等。它采用了模块化的设计，方便用户进行自定义模型的构建和实验。MMDetection 工具箱为目标检测的研究和应用提供了高效灵活的平台，极大地促进了算法的复现和性能评估。

(四) YOLO 系列算法

YOLO (You Only Look Once) 系列算法是单阶段目标检测器的典型代表，以其速度快、实时性好而闻名。YOLOv3 引入了多尺度预测、Darknet-53 骨干网络和 FPN (Feature Pyramid Network) 结构，显著提升了检测精度。YOLOv5 在 YOLOv3 的基础上进行了多项优化，包括 Mosaic 数据增强、自适应锚框计算、Focus 结构等，进一步提升了性能和易用性。YOLOv8 是 Ultralytics 公司推出的最新版本，在模型结构、损失函数和训练策略上进行了改进，提供了更快的速度和更高的精度。图 1 描述了 YOLOv8 的网络架构。

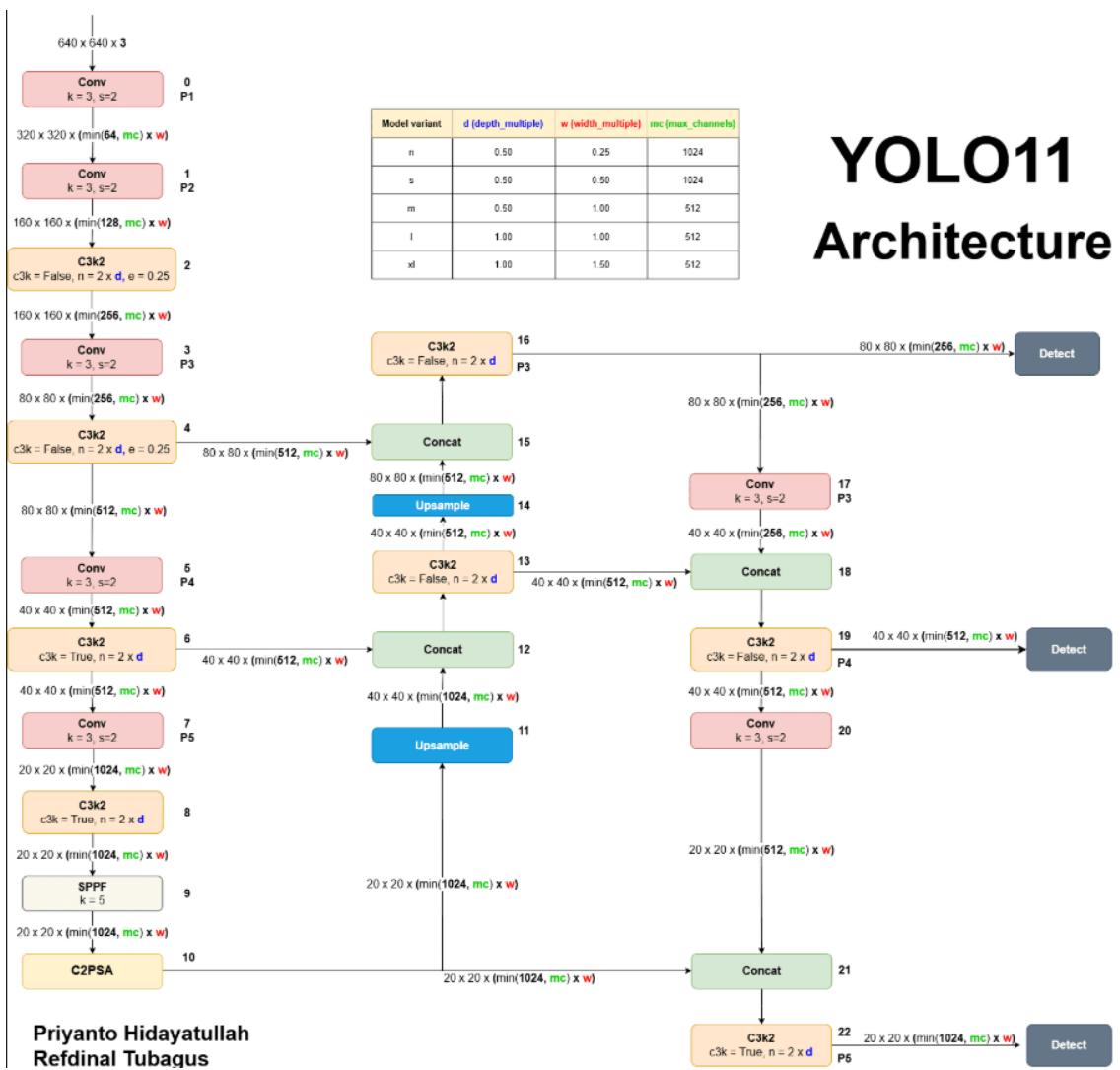


图 1: YOLOv8 网络结构示意图

(五) Faster R-CNN

Faster R-CNN 是两阶段检测器的代表，由 Ren 等人于 2015 年提出。它通过引入区域候选网络（Region Proposal Network, RPN）实现了端到端的训练，极大地提升了检测速度。RPN 是一个全卷积网络，用于预测目标边界框和目标得分，从而生成高质量的候选区域。RoI Pooling 将不同大小的候选区域映射到固定大小的特征图上，以便后续的全连接层进行分类和回归。图 2 描述了 Faster R-CNN 的网络架构图。

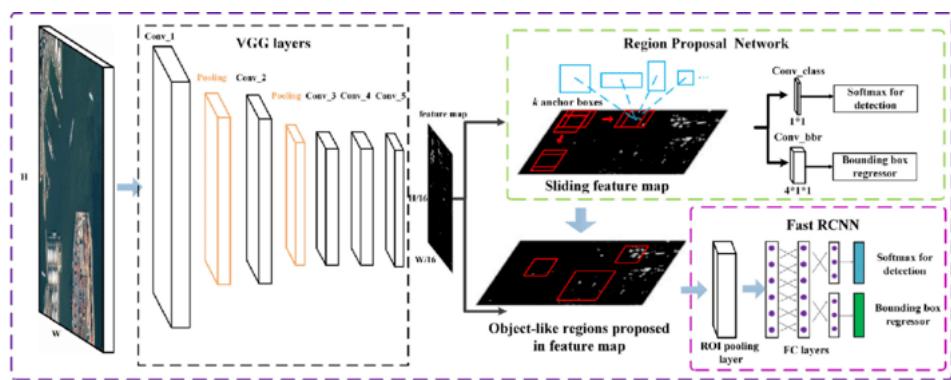


图 2: Faster R-CNN 网络结构示意图

三、实验目的:

- 熟悉并掌握至少 3-5 种主流目标检测算法（如 YOLO 系列、Faster R-CNN、FCOS 等）的基本原理和应用。
- 学习如何在 Python 环境中部署和调用开源目标检测项目的 API。
- 实现对校园内自行拍摄图像的目标检测，并对检测结果进行验证和分析。

四、实验内容:

调用开源的目标检测算法（YOLO, Faster RCNN, FCOS 等）的 API，实现目标检测可自行选择自己感兴趣的 3-5 个目标检测算法，部署其开源项目，使用校园内自行拍摄的多个图像进行目标检测结果验证和分析。

五、实验步骤：

（一）开源项目部署与模型下载

从 GitHub 克隆所选算法的官方开源项目到本地。进入每个项目的根目录，根据其 ‘requirements.txt‘ 或 ‘setup.py‘ 安装特有的依赖库。从官方提供的模型库下载对应算法的预训练权重文件。例如，YOLOv8 的 ‘yolov8n.pt‘，MMDetection 中 Faster R-CNN 或 FCOS 的 ‘.pth‘ 文件。

（二）图像数据准备

在不同校园内不同地点、不同光照条件下拍摄多张图像，地点包括办公室，食堂，教室等，时间主要包括上午和晚上的情况以模拟不同情况下的光照条件。将所有拍摄的图像统一存放在一个新建的文件夹中。

（三）目标检测 API 调用与结果获取

针对每种选定的算法，查阅模型的官方文档，编写独立的 Python 模块，用于加载模型、读取图像、执行推理并获取检测结果。将每张图像的检测结果直接在图像上绘制的方式保存。

（四）结果可视化与分析

在 detectionAPIs.ipynb 笔记本上开辟独立的单元格，执行上一步写好的 API 调用函数，分别查看不同算法在相同图像上的检测效果，定性观察其在目标识别准确性、定位精度、小目标检测、密集目标处理、抗遮挡能力等方面的表现。

六、实验数据及结果分析：

我们选取的校园场景图片如图 1 所示。

6.0.1. MMDetection 检测结果

我们利用 MMDetection 的 API 工具，检测到的结果如图 2 所示。

6.0.2. YOLOv8 检测结果

我们利用 MMDetection 的 API 工具，检测到的结果如图 3 所示。

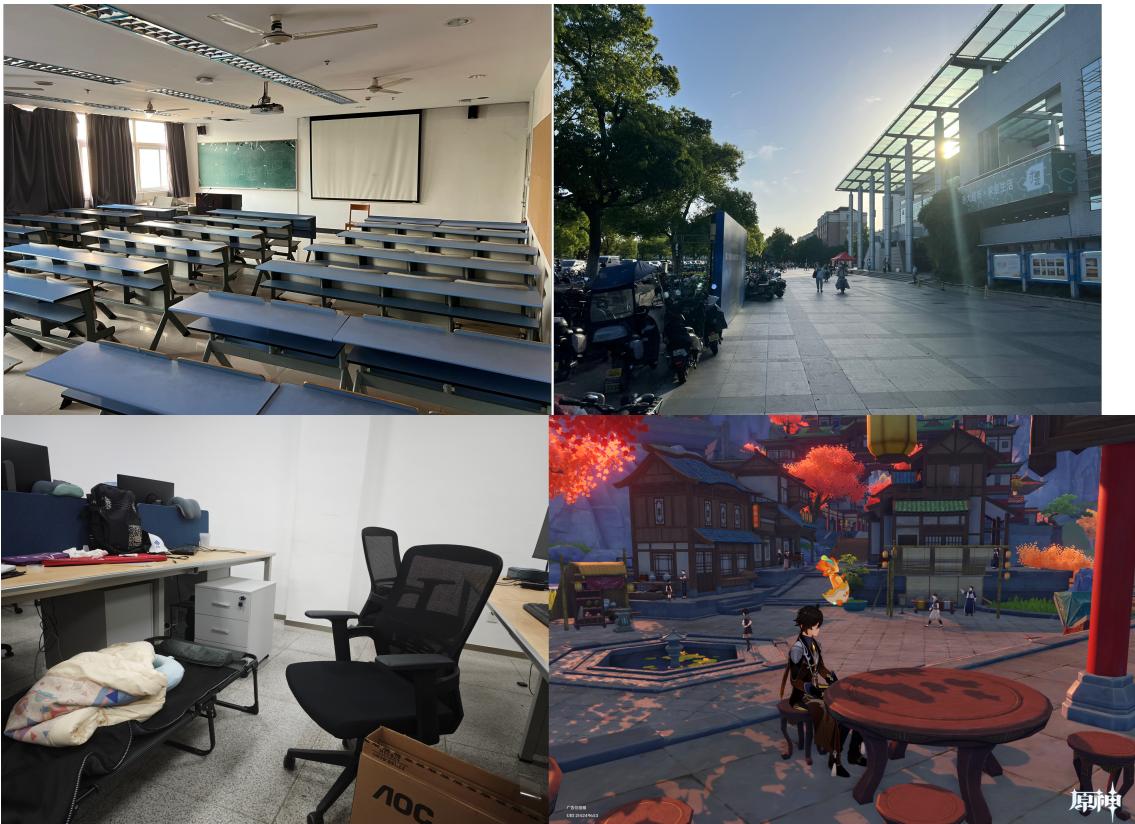


图 3: 原始校园场景图片

6.0.3. Faster R-CNN 检测结果

由于此 API 没有可视化较好的工具，因此选择一张样例图片，这是上述场景图片的第四张。输出检测出的 bbox 坐标，结果如图 4 所示。

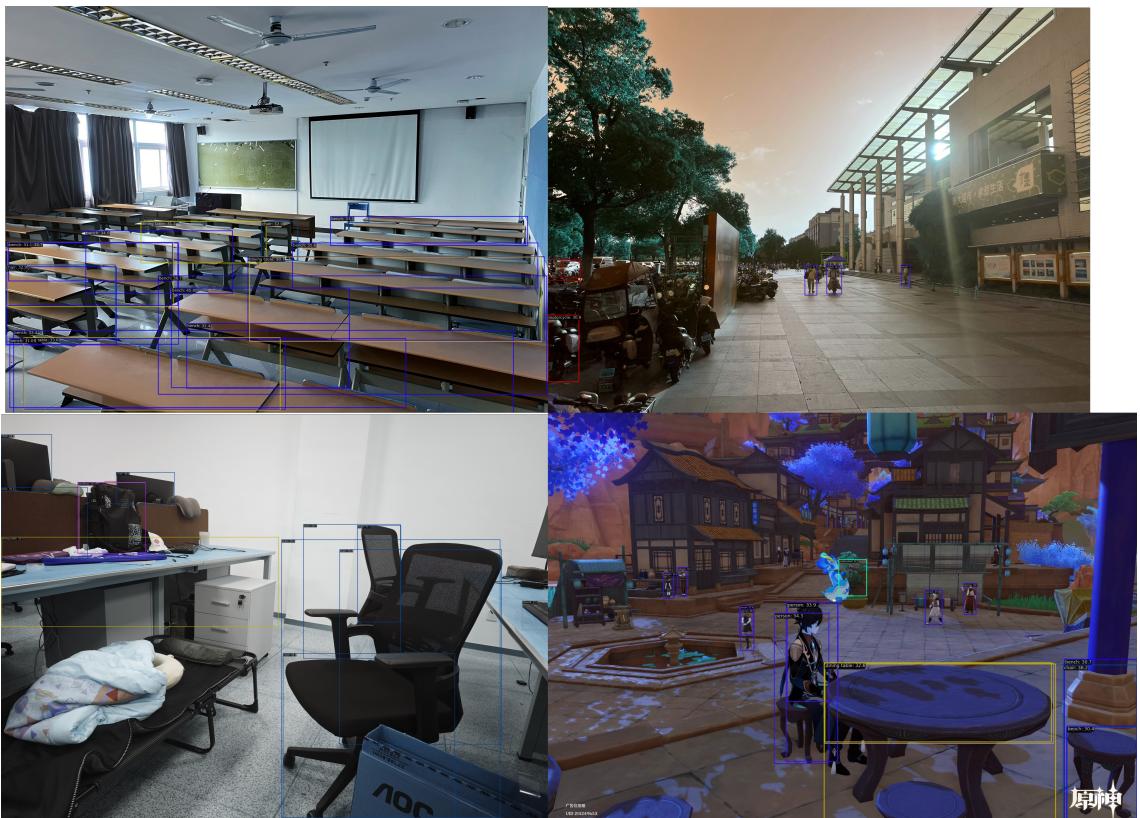


图 4: FCOS 检测结果示意图

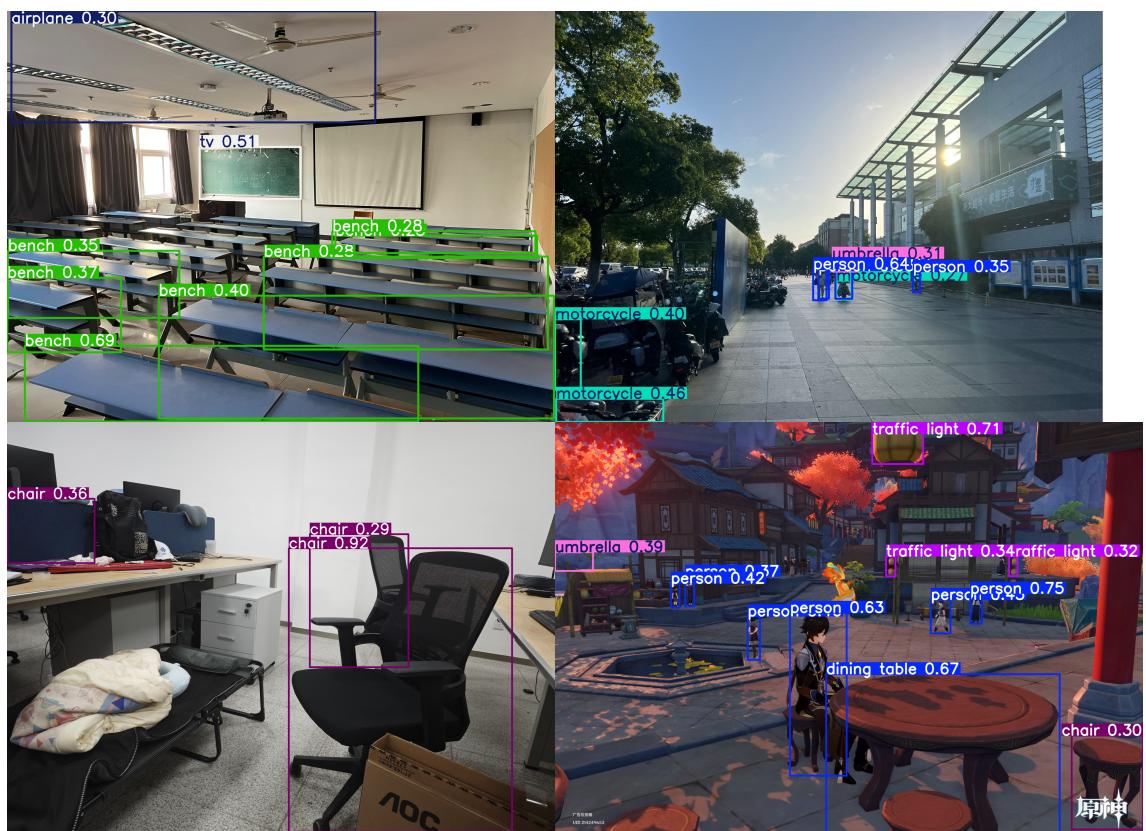


图 5: YOLOv8 检测结果示意图

```

{'boxes': tensor([[1.3278e+03, 1.4951e+03, 1.6784e+03, 1.6596e+03],
[6.6823e+02, 7.0854e+02, 8.9868e+02, 7.9751e+02],
[2.3777e+01, 1.2602e+03, 5.4252e+02, 1.4784e+03],
[2.5740e+02, 7.9877e+02, 3.4375e+02, 8.8838e+02],
[1.6363e+03, 7.4854e+02, 1.6630e+03, 8.1343e+02],
[8.9966e+02, 8.9614e+02, 9.7760e+02, 1.0058e+03],
[1.3277e+03, 8.8486e+02, 1.9712e+03, 1.6680e+03],
[1.7997e+03, 7.2787e+02, 2.0464e+03, 8.2396e+02],
[0.0000e+00, 9.1061e+02, 9.6811e+02, 1.6680e+03],
[1.9098e+02, 1.5709e+03, 5.3074e+02, 1.6645e+03],
[1.7703e+03, 6.9046e+02, 2.0895e+03, 8.4730e+02],
[1.1131e+03, 1.5817e+03, 1.3007e+03, 1.6671e+03],
[1.6759e+03, 6.9241e+02, 1.7498e+03, 8.0722e+02],
[6.1619e+02, 6.7785e+02, 1.0083e+03, 8.0257e+02],
[2.1501e+03, 1.2197e+03, 2.3735e+03, 1.6431e+03],
[8.5105e+02, 1.5256e+03, 1.3394e+03, 1.6680e+03],
[1.0981e+03, 5.5331e+02, 1.2884e+03, 8.0066e+02],
[2.0833e+03, 1.0495e+03, 2.3572e+03, 1.3213e+03],
[1.1904e+03, 7.2571e+02, 1.3011e+03, 8.1028e+02],
[1.1616e+03, 8.2754e+02, 2.3451e+03, 1.4898e+03],
[4.1376e+00, 1.3868e+03, 6.3677e+01, 1.6295e+03],
[1.2939e+01, 5.7116e+02, 1.8381e+02, 8.8987e+02],
[1.1641e+03, 4.7957e+02, 1.2853e+03, 5.9153e+02],
[4.6490e+01, 8.6850e+02, 9.9934e+02, 1.1028e+03],
[1.1660e+03, 5.5476e+02, 1.3597e+03, 8.0135e+02],
...
0.6537, 0.6524, 0.6521, 0.6508, 0.6504, 0.6488, 0.6486, 0.6485, 0.6462,
0.6453, 0.6436, 0.6429, 0.6392, 0.6390, 0.6389, 0.6362, 0.6361, 0.6360,
0.6357, 0.6340, 0.6336, 0.6318, 0.6294, 0.6284, 0.6282, 0.6281, 0.6270,
0.6264])}

```

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#)...

图 6: Faster R-CNN 检测结果示意图

七、实验结论：

1. 检测准确性：可以看到 YOLO 模型效果比 MMDetection 效果更好，能检测的物体位置更准确。
2. 定位精度：可以看到 YOLO 在观察小目标、密集目标和遮挡情况下的定位更好。
3. 误检与漏检：MMDetection 漏检情况较多，但两个模型基本都没有错检的情况。

八、总结及心得体会：

本次目标检测实验让我对计算机视觉领域有了更深入的理解，特别是对主流目标检测算法的原理、部署和应用有了实践经验。

通过亲手部署和调用 YOLO 系列、Faster R-CNN、FCOS 等算法的 API，我不仅巩固了这些算法的理论知识，更直观地理解了它们在实际图像处理中的工作方式。从论文中的抽象概念到代码中的具体实现，这种结合极大地加深了我的学习效果。

使用自行拍摄的校园图像进行实验，让我认识到实际场景的复杂性。与标准数据集（如 COCO）相比，校园图像可能存在光照不均、目标遮挡、背景杂乱、小目标密集等问题，这些都对算法的鲁棒性提出了挑战。

在环境配置、依赖安装和 API 调用过程中，我遇到了各种问题，例如 CUDA 版本不匹配、库冲突、模型加载失败等。通过查阅文档、搜索解决方案和调试代码，我的问题解决能力得到了显著提升。

九、对本实验过程及方法的改进建议：

总的来说，本实验的可改进之处集中在以下几个部分：

1. 本实验缺少对不同算法的性能的定量对比，比如记录并对比不同算法的推理速度（FPS）和检测精度，限于实验设备，该步骤没有完成；
2. 如果条件允许，可以对自行拍摄的校园图像进行人工标注，构建一个小型定制数据集，并在此数据集上对模型进行微调（Fine-tuning），以提升在特定校园场景下的检测性能。
3. 将目标检测技术应用于校园监控视频流，实现实时目标跟踪、行为分析等功能。

报告评分：
指导教师签字：

附录一 代码示例

如代码 1, 2, 3 所示, 分为三个部分 mmdet.py, yolo.py, fastrcnn.py

代码 1: mmdet.py

```
1 from mmdet.registry import VISUALIZERS
2 import mmcv
3 from mmdet.apis import init_detector, inference_detector
4 import cv2
5
6 # init the visualizer(execute this block only once)
7 config_file = '../data/mmdet/rtmde_tiny_8xb32-300e_coco.py'
8 checkpoint_file = '../data/mmdet/rtmde_tiny_8xb32-300e_coco_20220902_112414-78e30dcc.pth'
9 model = init_detector(config_file, checkpoint_file, device='cpu') # or
10 device='cuda:0'
11 visualizer = VISUALIZERS.build(model.cfg.visualizer)
12 # the dataset_meta is loaded from the checkpoint and
13 # then pass to the model in init_detector
14 visualizer.dataset_meta = model.dataset_meta
15
16 # show the results
17 for i in range(4):
18     img = mmcv.imread(f'../data/{i}.jpg', channel_order='rgb')
19     result = inference_detector(model, img)
20     visualizer.add_datasample(
21         'result',
22         img,
23         data_sample=result,
24         draw_gt=False,
25         wait_time=0,
26     )
27     visualizer.show()
28     visualized_img = visualizer.get_image()
29     cv2.imwrite(f'../data/mmdet/{i}_result.jpg', visualized_img)
```

代码 2: yolo.py

```
1 from ultralytics import YOLO
2
3 # Create a new YOLO model from scratch
4 model = YOLO("yolo11.yaml")
5
6 # Load a pretrained YOLO model (recommended for training)
7 model = YOLO("yolo11n.pt")
8
9 # Train the model using the 'coco8.yaml' dataset for 3 epochs
10 results = model.train(data="coco8.yaml", epochs=3)
11
12 # Evaluate the model's performance on the validation set
13 results = model.val()
14
15 # Perform object detection on an image using the model
16 for i in range(4):
17     results = model(f'../data/{i}.jpg')
18     results[0].save(f'../data/yolo/{i}_result.jpg')
19
20 # Export the model to ONNX format
21 success = model.export(format="onnx")
```

代码 3: fastrcnn.py

```
1 import torch
2 import torchvision
```

```
3   from torchvision.models.detection import FasterRCNN_ResNet50_FPN_Weights
4   from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
5   from torchvision.transforms import functional as F
6   from PIL import Image
7
8   weights = FasterRCNN_ResNet50_FPN_Weights.DEFAULT
9   model = torchvision.models.detection.fasterrcnn_resnet50_fpn(weights=
10    weights)
11  print(model)
12  num_classes = 2
13  in_features = model.roi_heads.box_predictor.cls_score.in_features
14  model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes
15   )
16  model.eval()
17  dummy_image = torch.rand(1, 3, 800, 800)
18
19  img_path = "../data/3.jpg"
20  img = Image.open(img_path).convert("RGB")
21  preprocess = weights.transforms()
22  image_tensor = preprocess(img).unsqueeze(0) # Add batch dimension
23
24  with torch.no_grad():
25    predictions = model(image_tensor)
    print(predictions[0])
```