本地flask部署yolov8 restful api

1.安装yolov8

首先我们配置一个conda虚拟环境

```
conda create -n yolov8 python=3.10
```

然后我们使用Pip在一个环境中安装`ultralytics`包，此环境还需包含PyTorch>=1.8。这也会安装所有必要的依赖项，我们使用国内镜像安装，这样速度会比较快

```
pip install ultralytics -i https://mirrors.aliyun.com/pypi/simple/
```

然后在yolov8官网上下载模型，并将此前训练得到的最好结果(best.pt)放入文件夹

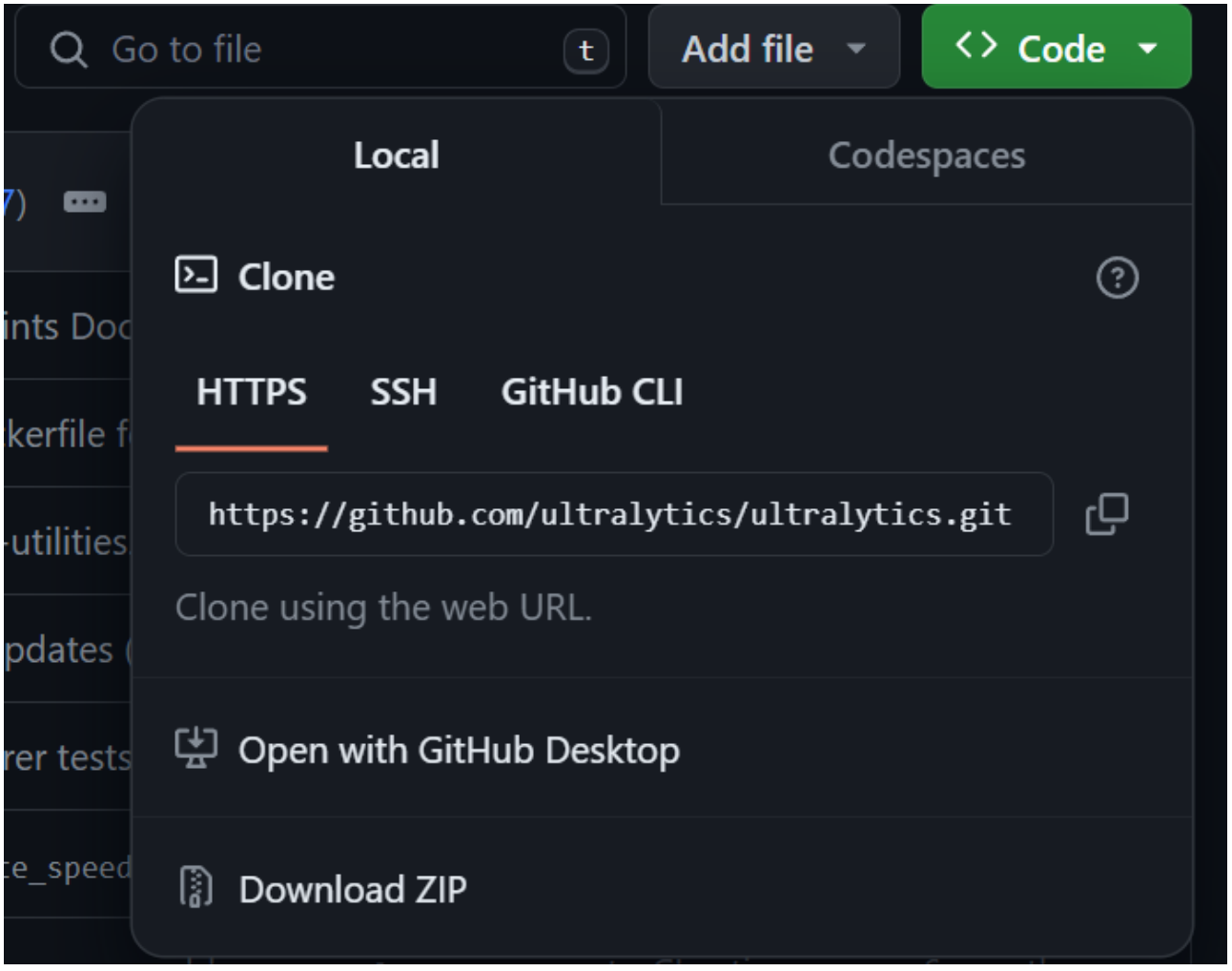

图1 下载yolov8

接下来，为了验证可以预测，我们在网上获取一张垃圾桶照片，对其进行预测，结果如下

```
yolo predict model=best.pt source='a.jpg'
```


```
(yolov8) C:\Users\86158\Desktop\msd\ultralytics-main>yolo predict model=best.pt source='a.jpg'
Ultralytics YOLOv8.1.15 🚀 Python-3.10.13 torch-2.2.0+cpu CPU (AMD Ryzen 7 5800H with Radeon Graphics)
tb1_YOLOv8l summary (fused): 268 layers, 43608921 parameters, 0 gradients, 164.8 GFLOPs

image 1/1 C:\Users\86158\Desktop\msd\ultralytics-main\a.jpg: 448x640 1 garbage, 4 overflows, 398.4ms
Speed: 2.0ms preprocess, 398.4ms inference, 1.5ms postprocess per image at shape (1, 3, 448, 640)
Results saved to runs\detect\predict
💡 Learn more at https://docs.ultralytics.com/modes/predict
```

图2 成功运行yolov8

可以看到，我们的yolov8已经安装成功了，接下来，我们将把yolov8部署成一个restful api，用post对其发送请求后，得到json格式回复。

2.部署restful api

我们编写一个如下的python脚本，放在yolov8的文件夹下

```
import json
from flask_restful import reqparse, Resource, Api
from werkzeug.datastructures import FileStorage
```

```
from flask import Flask
from ultralytics import YOLO
import cv2

model = YOLO("best.pt")


def pred(im):
    results = model.predict(source=im, save=False, save_txt=False)

    cls = results[0].boxes.cls.numpy().tolist()
    conf = results[0].boxes.conf.numpy().tolist()
    xywh = results[0].boxes.xywh.numpy().tolist()

    r = {'r': []}
    names = results[0].names

    for i, j, k in zip(cls, conf, xywh):
        r['r'].append({'cls': names[i], 'conf': float(j), 'xywh':
[float(kk) for kk in k]})

    return r


class UploadImg(Resource):
    def __init__(self):
        self.parser = reqparse.RequestParser()
        self.parser.add_argument('imgFile', required=True,
type=FileStorage, location='files', help="imgFile is wrong.")

    def post(self):
        img_file = self.parser.parse_args().get('imgFile')
        img_file.save(img_file.filename)
        rr = pred(img_file.filename)
        return json.dumps(rr), 201


if __name__ == '__main__':
    app = Flask(__name__)
    api = Api(app)
    api.add_resource(UploadImg, '/uploadimg')
    app.run()
```

这段代码基于 Flask 和 Flask-RESTful，它提供了一个接口来上传图片并使用 YOLO（You Only Look Once）模型进行对象检测。以下是该代码的详细解析：

1. 导入必要的库：
    - `json`：用于处理 JSON 数据。
    - `flask_restful`：用于构建 RESTful API。
    - `reqparse`：用于解析 HTTP 请求。
    - `Resource`：定义一个 RESTful 资源。
    - `Api`：用于添加 RESTful 资源到应用中。
    - `Flask`：用于构建 web 应用。
    - `werkzeug.datastructures`：提供了一些有用的数据结构，如 `FileStorage`。
    - `ultralytics`：包含了 YOLO 模型。
    - `cv2`：OpenCV 库，用于图像处理。
2. 初始化模型：
    - 使用 `ultralytics` 中的 `YOLO` 类加载预训练的模型 `"best.pt"`。
3. 定义预测函数：
    - `pred(im)` 函数接受一个图像路径 `im`，然后使用 YOLO 模型对其进行预测。
    - 从预测结果中提取类别（`cls`）、置信度（`conf`）和边界框坐标（`xywh`）。
    - 将这些信息整理成一个字典 `r` 并返回。
4. 定义 **RESTful** 资源：
    - `UploadImg` 类继承自 `Resource`，表示一个 RESTful 资源。
    - 在 `__init__` 方法中，定义了一个解析器 `self.parser` 用于解析传入的 HTTP 请求。
    - `post` 方法处理 POST 请求。它从请求中解析出图像文件，保存到本地，然后使用 `pred` 函数进行预测，并将预测结果以 JSON 格式返回。
5. 运行应用：
    - 如果这个脚本作为主程序运行，它会创建一个 Flask 应用，添加 `UploadImg` 资源到 `/uploadimg` 路径，并运行应用。

使用说明：

当我们向 `/uploadimg` 路径发送一个 POST 请求，并在请求中包含一个名为 `imgFile` 的文件时，这个应用会保存这个文件，使用 YOLO 模型对其进行对象检测，并将检测结果以 JSON 格式返回。

在这里我们采用`postman`来发送请求，效果如下

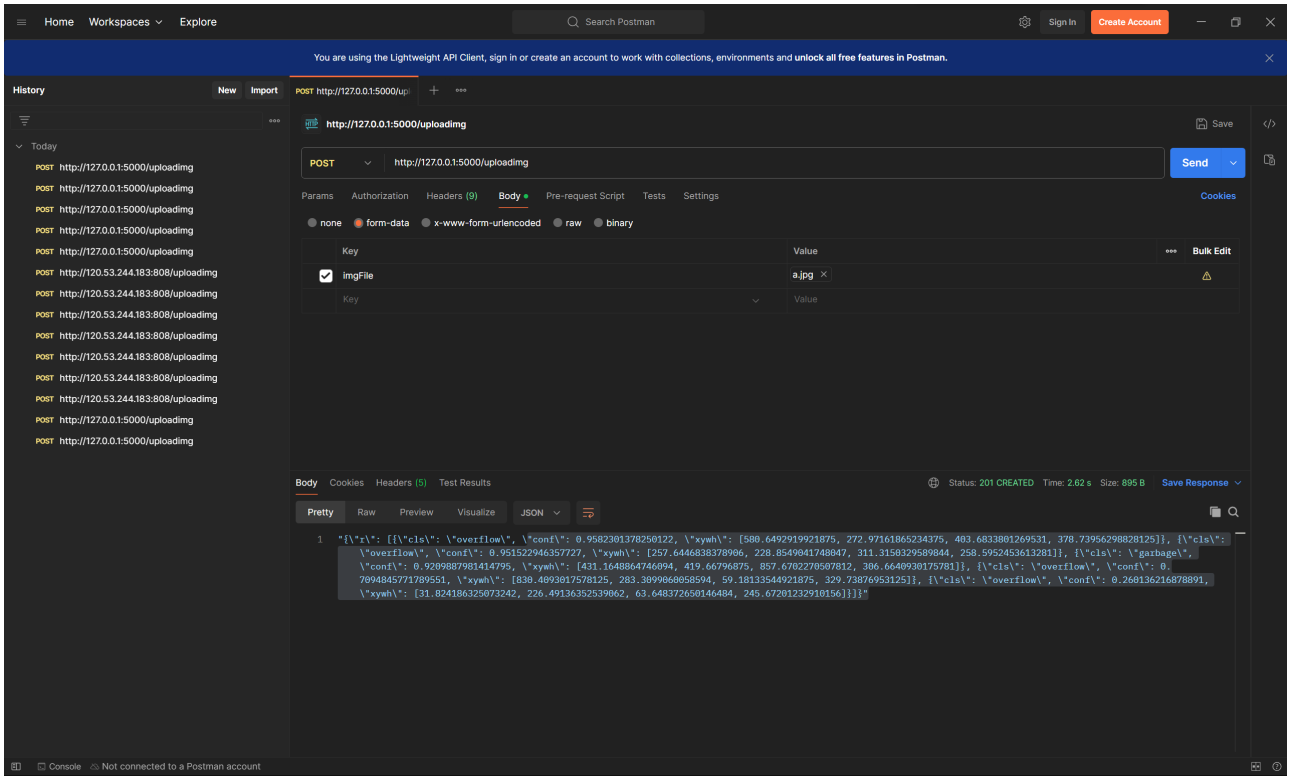

图3 成功发送请求并获得JSON回复

我们可以看到，已经返回了`JSON`消息，说明我们已经在本地成功部署了yolov8 restful api。