DEIM使用教程

0、拉取项目

```
# https://github.com/ShihuaHuang95/DEIM/tree/main
git clone https://github.com/ShihuaHuang95/DEIM.git
```

1、环境安装

```
conda create -n deim python=3.9

conda activate deim

pip install torch==2.0.1 torchvision==0.15.2 torchaudio==2.0.2 -i
https://mirrors.huaweicloud.com/repository/pypi/simple  # 华为源下载贼快

pip install -r requirements.txt  # requirements.txt见下面
```

requirements.txt

```
faster-coco-eval==1.6.5
PyYAML==6.0.2
tensorboard==2.14.0
scipy==1.10.1
calflops==0.3.2
transformers==4.46.3
```

2、数据集准备

以YOLO格式的数据集为例：

```
dataset/
├── train/
|   ├── 1.jpg
|   ├── 1.txt
|   ├── ...
├── val/
|   ├── 1.jpg
|   ├── 1.txt
|   ├── ...
└──
```

将其转换为CoCo数据集格式，转换代码如下，更改最后dataset路径

```
import os
import json
from PIL import Image
```

```
from tqdm import tqdm


def yolo_to_coco(yolo_annotations_dir, image_dir, output_json_path):
    coco_format = {
        "images": [],
        "annotations": [],
        "categories": []
    }

    category_list = []
    annotation_id = 1
    image_id = 1

    for image_name in tqdm(os.listdir(image_dir)):
        if image_name.endswith(".jpg") or image_name.endswith(".png"):
            image_path = os.path.join(image_dir, image_name)
            img = Image.open(image_path)
            width, height = img.size

            # 创建image对象
            coco_format["images"].append({
                "id": image_id,
                "file_name": image_name,
                "width": width,
                "height": height
            })

            # 找到对应的YOLO标注文件
            txt_file = os.path.join(yolo_annotations_dir, f"
{os.path.splitext(image_name)[0]}.txt")
            if os.path.exists(txt_file):
                with open(txt_file, 'r') as f:
                    lines = f.readlines()
                    for line in lines:


                        parts = line.strip().split()
                        class_id = int(parts[0])
                        center_x = float(parts[1])
                        center_y = float(parts[2])
                        width_rel = float(parts[3])
                        height_rel = float(parts[4])
                        if class_id not in list(range(9)):
                            pass

                        # 计算绝对坐标并确保不超出图像边界
                        x_min = max(0, (center_x - width_rel / 2) * width)
                        y_min = max(0, (center_y - height_rel / 2) * height)
                        x_max = min(width, (center_x + width_rel / 2) * width)
                        y_max = min(height, (center_y + height_rel / 2) *
height)

                        bbox = [x_min, y_min, x_max-x_min, y_max-y_min]

                        if (class_id + 1) not in category_list:
                            coco_format["categories"].append({
                                    "id": class_id + 1,  # 直接使用class_id作为id
```

```
                                    "name": f"category_{class_id + 1}",
                                    "supercategory": "none"
                                })
                            category_list.append(class_id + 1)


                        coco_format["annotations"].append({
                            "id": annotation_id,
                            "image_id": image_id,
                            "category_id": class_id + 1,  # 直接使用class_id作为
category_id
                            "bbox": bbox,
                            "area": (bbox[2] - bbox[0]) * (bbox[3] - bbox[1]),
                            "iscrowd": 0
                        })
                        annotation_id += 1
            image_id += 1

    # 保存为COCO格式的JSON文件
    with open(output_json_path, 'w') as json_file:
        json.dump(coco_format, json_file, indent=4)

dataset_path = ""

task = "train"
yolo_to_coco(yolo_annotations_dir=rf"{dataset_path}/{task}",
             image_dir=rf"{dataset_path}/{task}",
             output_json_path=rf"{dataset_path}/{task}.json")

task = "val"
yolo_to_coco(yolo_annotations_dir=rf"{dataset_path}/{task}",
             image_dir=rf"{dataset_path}/{task}",
             output_json_path=rf"{dataset_path}/{task}.json")
```

得到数据集结构

```
dataset/
├── train/
├── val/
├── train.json
└── val.json
```

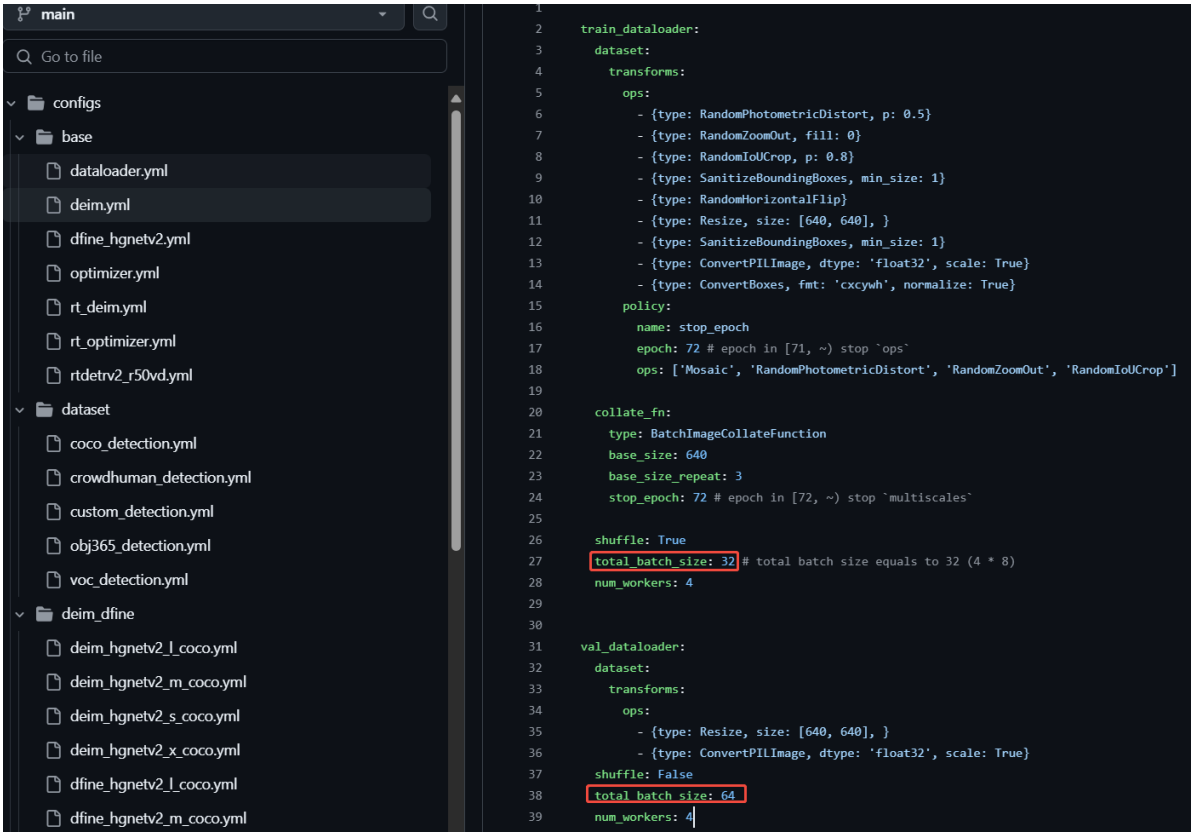
3、训练前配置文件

3.1 数据加载器

修改项目文件夹下 configs/base/dataloader.yml文件

这里主要修改有以下几点:

1、训练以及验证时的epoch，以适配所使用的显卡的总显存



3.2 数据加载器

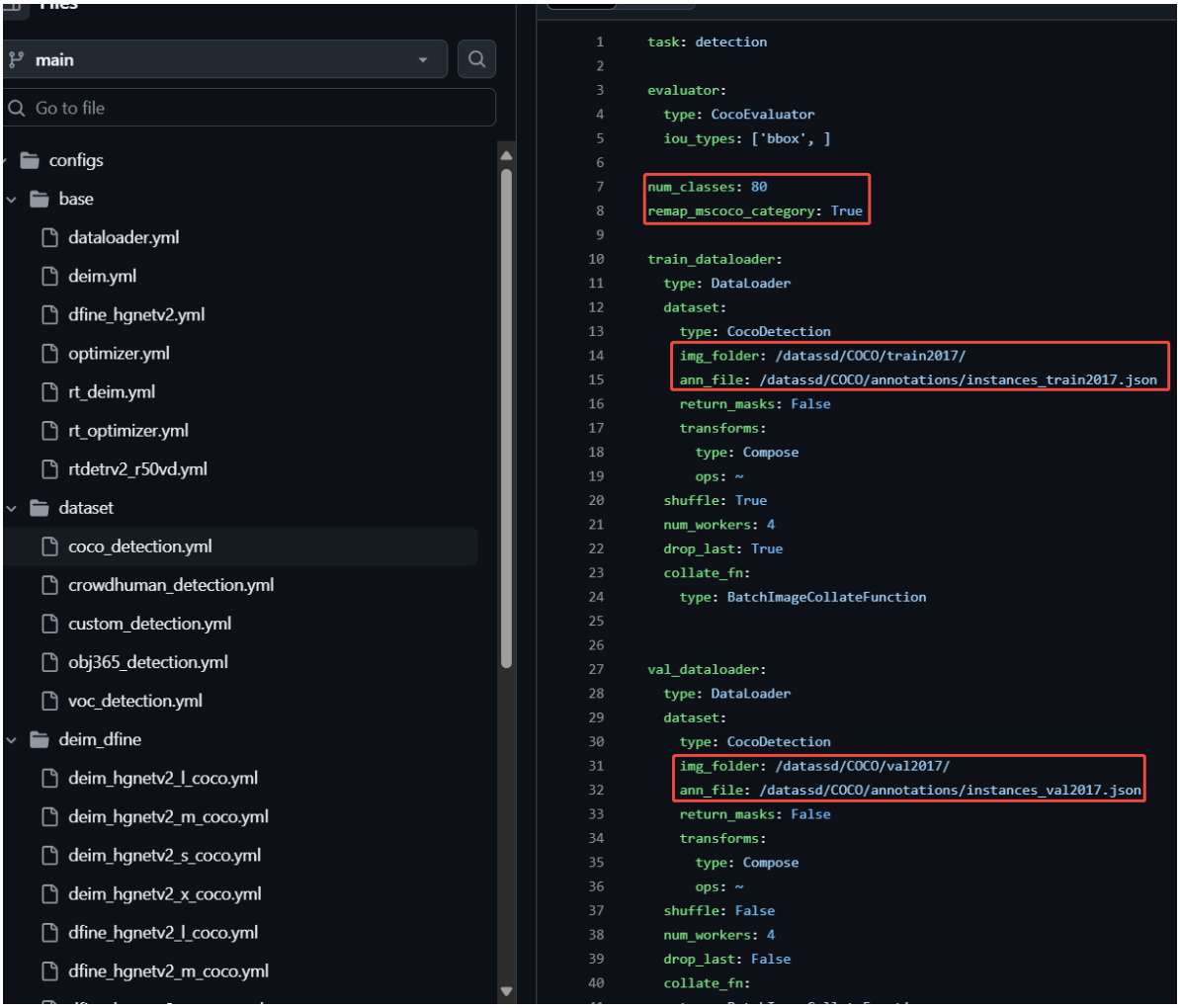
修改项目文件夹下 configs/dataset/coco_detection.yaml文件

这里主要修改有以下几点:

1、num_classes修改为训练样本类别数

2、remap_mscoco_category: True

3、训练集和验证集的图片路径 img_folder以及相应训练集和验证集生成的coco标签



3.3 训练参数

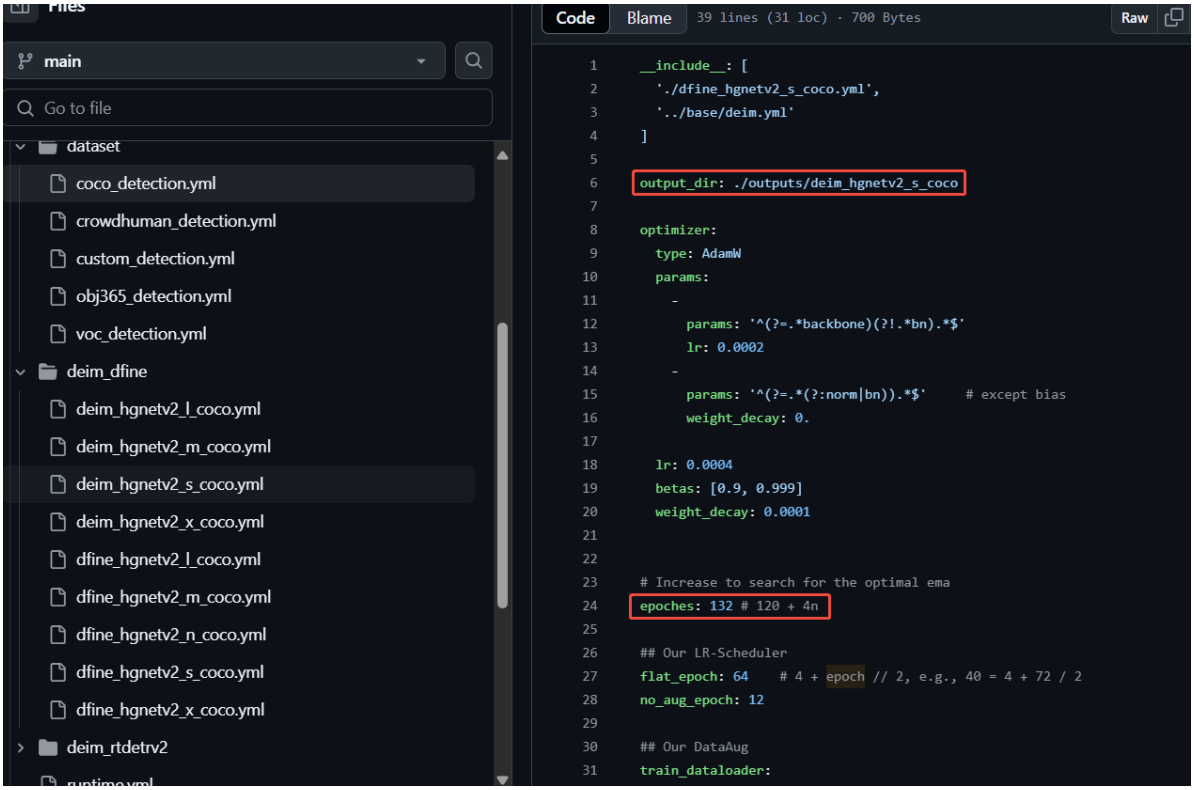
> 注意： 训练时，3.3和3.4的文件，最好不同的检测任务重新建立相应位置的新文件，开头对应的导入也进行相应修改

修改项目文件夹下 configs/deim_dfine/deim_hgnetv2_s_coco.yml文件

这里主要修改有以下几点:

1、模型的训练轮次epoches

2、output_dir，也可以在训练命令行里面改



3.4 模型各模块学习率

项目文件夹下 configs/deim_dfine/dfine_hgnetv2_s_coco.yml

可修改模型相应各模块学习率

4、模型训练

```
# 指定deim_hgnetv2_s_coco.yml文件以及，设置混合精度循，设置输出文件夹
python train.py -c configs/deim_dfine/deim_hgnetv2_s_coco.yml --use-amp --seed=0
--output-dir xxxx

# 多GPU训练
CUDA_VISIBLE_DEVICES=0,1 torchrun --master_port=7777 --nproc_per_node=2 train.py
-c configs/deim_dfine/deim_hgnetv2_s_coco.yml --use-amp --seed=0
```

5、模型onnx导出

```
python tools/deployment/export_onnx.py --check -c
configs/deim_dfine/deim_hgnetv2_s_coco.yml -r xxxx/xxx.pth
```