mmdetection2.6自定义数据集

官方推荐的集中自定义数据集的方式：

- 将自己的数据集组织为标准的数据集格式（常用COCO）
- 将自己的数据集组织为中间格式
- 利用datawraper自定义新的数据集

1. 将数据集组织为coco数据集格式

coco数据集的标注格式：

整体为一个字典形式，主要的键为：`images`，`annotations`，`categorties`

- `images`的值为一个列表，列表的每个元素为如下所示的元素信息
- `annotations`的值为一个列表， 每个元素为如下所示的信息
- `categorties`的值为一个列表，每个元素为如下所示的信息，id从0开始。

```
'images': [
    {
        'file_name': 'COCO_val2014_000000001268.jpg',
        'height': 427,
        'width': 640,
        'id': 1268
    },
    ...
],

'annotations': [
    {
        'segmentation': [[192.81,
            247.09,
            ...
            219.03,
            249.06]],  # if you have mask labels
        'area': 1035.749,
        'iscrowd': 0,
        'image_id': 1268,
        'bbox': [192.81, 224.8, 74.73, 33.43],
        'category_id': 16,
        'id': 42986
    },
    ...
],

'categories': [
    {'id': 0, 'name': 'car'},
 ]
```

最简单的方式，就是将自己的数据集组织为coco数据集的标注 格式，这样训练的过程中仅仅需要在config文件中修改**数据集的路径**与**类别**即可。

2. 将数据集组织为middle格式

mmdetection定义一种比较简单的数据集格式，标注文件的信息是一个字典列表，每个字典对应着一张图片，如下所示：

```
[
    {
        'filename': 'a.jpg',
        'width': 1280,
        'height': 720,
        'ann': {
            'bboxes': <np.ndarray, float32> (n, 4),
            'labels': <np.ndarray, int64> (n, ),
            'bboxes_ignore': <np.ndarray, float32> (k, 4),
            'labels_ignore': <np.ndarray, int64> (k, ) (optional field)
        }
    },
    ...
]
```

转换为上述的格式之后，有两种数据集的使用方式，一种在线的使用方式，一种离线的方式：

- 在线方式

重新写一个继承自`CustomDataset`的类。重写 `load_annotations(self, ann_file)` and `get_ann_info(self, idx)`这两个方法。

- 离线的方法

将数据集转换为标准的COCO或者VOC格式，然后直接使用`CustomDataset`。

3. 简单的自定义数据集的例子

假设我们现有的标注数据的格式为txt文件标注

```
#分别为图片名称，  图片宽高，  bbox的数目，  bbox坐标与类别id
000001.jpg
1280 720
2
10 20 40 60 1
20 40 50 60 2

#
000002.jpg
1280 720
3
50 20 40 60 2
20 40 30 45 2
30 40 50 60 3
```

然后创建一个新的文件`mmdet/datasets/my_dataset.py`来加载数据集。

```
import mmcv
import numpy as np

from .builder import DATASETS
from .custom import CustomDataset


@DATASETS.register_module()
class MyDataset(CustomDataset):

    CLASSES = ('person', 'bicycle', 'car', 'motorcycle')

    def load_annotations(self, ann_file):
        ann_list = mmcv.list_from_file(ann_file)

        data_infos = []
        for i, ann_line in enumerate(ann_list):
            if ann_line != '#':
                continue

            img_shape = ann_list[i + 2].split(' ')
            width = int(img_shape[0])
            height = int(img_shape[1])
            bbox_number = int(ann_list[i + 3])

            anns = ann_line.split(' ')
            bboxes = []
            labels = []
            for anns in ann_list[i + 4:i + 4 + bbox_number]:
                bboxes.append([float(ann) for ann in anns[:4]])
                labels.append(int(anns[4]))

            data_infos.append(
                dict(
                    filename=ann_list[i + 1],
                    width=width,
                    height=height,
                    ann=dict(
                        bboxes=np.array(bboxes).astype(np.float32),
                        labels=np.array(labels).astype(np.int64))
                ))

        return data_infos

    def get_ann_info(self, idx):
        return self.data_infos[idx]['ann']
```

然后在config文件中，使用`MyDataset`。

```
dataset_A_train = dict(
    type='MyDataset',
    ann_file = 'image_list.txt',
    pipeline=train_pipeline
)
```

5. 使用dataset wrappers来自定义数据集

mmdetection支持很多中数据集wrapper来混合数据集或者在训练时修改数据集分布。现在着吃三种数据集包装器（wrapper）：

- `RepeatDataset`：只需重复整个数据集。
- `ClassBalancedDataset`：以类平衡的方式重复数据集。
- `ConcatDataset`：concat数据集。

5.1 RepeatDataset

使用`RepeatDataset`作为数据集包装器来重复数据集。例如重复原始的数据集`Dataset_A`。config文件如下：

```
dataset_A_train = dict(
        type='RepeatDataset',
        times=N,
        dataset=dict(  # This is the original config of Dataset_A
            type='Dataset_A',
            ...
            pipeline=train_pipeline
        )
    )
```

5.2 ClassBalancedDataset

使用`ClassBalancedDataset`作为wrapper来依据类别的频率来重复数据集，重复的数据集需要初始化函数`self.get_cat_ids(idx)`来支持`ClassBalancedDataset`。例如使用过采样率`oversample_thr=1e-3`来重复`Dataset_A`。

```
dataset_A_train = dict(
        type='ClassBalancedDataset',
        oversample_thr=1e-3,
        dataset=dict(  # This is the original config of Dataset_A
            type='Dataset_A',
            ...
            pipeline=train_pipeline
        )
    )
```

5.3 ConcatDataset

有三种方法堆叠数据集

5.3.1 两个数据集是同样的类型

采用如下的方式:

```
dataset_A_train = dict(
    type='Dataset_A',
    ann_file = ['anno_file_1', 'anno_file_2'],
    pipeline=train_pipeline
)
```

这种方式在测试验证过程中，两个数据集会分开进行测试，如果想要整体进行测试，需要`separate_eval=False`

```
dataset_A_train = dict(
    type='Dataset_A',
    ann_file = ['anno_file_1', 'anno_file_2'],
    separate_eval=False,
    pipeline=train_pipeline
)
```

5.3.2 两个数据集不同

```
dataset_A_train = dict()
dataset_B_train = dict()

data = dict(
    imgs_per_gpu=2,
    workers_per_gpu=2,
    train = [
        dataset_A_train,
        dataset_B_train
    ],
    val = dataset_A_val,
    test = dataset_A_test
    )
```

在测试过程中，这种方式支持分离的方式进行测试

5.3.3 明确定义concat的方式

```
dataset_A_val = dict()
dataset_B_val = dict()

data = dict(
    imgs_per_gpu=2,
    workers_per_gpu=2,
    train=dataset_A_train,
    val=dict(
        type='ConcatDataset',
        datasets=[dataset_A_val, dataset_B_val],
        separate_eval=False))
```

使用`separate_eval=False`在测试验证过程中，将所有的数据集当作以整个数据集进行测试。

注意：

1. 该选项`separate_eval=False`假定数据集`self.data_infos`在评估期间使用。因此，COCO数据集不支持此行为，因为COCO数据集不完全依赖于`self.data_infos`评估。因此，不建议结合使用不同类型的数据集并对其进行整体评估。
2. 不支持评估`ClassBalancedDataset`，`RepeatDataset`因此不支持评估这些类型的串联数据集。

更加复杂的方式，重复两个数据集分别N， M次，使用如下的方式：

```
dataset_A_train = dict(
    type='RepeatDataset',
    times=N,
    dataset=dict(
        type='Dataset_A',
        ...
        pipeline=train_pipeline
    )
)
dataset_A_val = dict(
    ...
    pipeline=test_pipeline
)
dataset_A_test = dict(
    ...
    pipeline=test_pipeline
)
dataset_B_train = dict(
    type='RepeatDataset',
    times=M,
    dataset=dict(
        type='Dataset_B',
        ...
        pipeline=train_pipeline
    )
)
data = dict(
    imgs_per_gpu=2,
    workers_per_gpu=2,
    train = [
        dataset_A_train,
        dataset_B_train
    ],
    val = dataset_A_val,
```

```
    test = dataset_A_test
)
```

6. 调整数据集的类别

可以通过调整数据集的类别来训练数据集的子集，例如现存的数据集为20类， 最终可以调整训练所使用的数据集的类别来仅仅训练其中的三类，mmdetection可以自动滤除其他的类别。

```
classes = ('person', 'bicycle', 'car')
data = dict(
    train=dict(classes=classes),
    val=dict(classes=classes),
    test=dict(classes=classes))
```

mmdetection2.0也支持从文件中读取数据集类别的名称，例如txt文件：

```
person
bicycle
car
```

使用如下的方法进行操作：

```
classes = 'path/to/classes.txt'
data = dict(
    train=dict(classes=classes),
    val=dict(classes=classes),
    test=dict(classes=classes))
```

注意：

- 在MMDetection v2.5.0之前，如果设置了类别名称，则数据集将自动过滤出空的GT图像，并且无法通过config禁用它。这是不受欢迎的行为，并且会引起混淆，因为如果未设置类别名称，则数据集仅在`filter_empty_gt=True`和时过滤空的GT图像`test_mode=False`。在MMDetection v2.5.0之后，我们将图像过滤过程与类别修改解耦，即，无论是否设置了类别，数据集都只会在`filter_empty_gt=True`和`test_mode=False`时过滤不含GT的图像。因此，设置类别仅会影响用于训练的类别的注释，并且用户可以决定是否自己过滤不含GT的图像。
- 由于中间格式仅具有框标签且不包含类名称，因此在使用时`CustomDataset`，用户无法通过config过滤出空的GT映像，而只能离线进行。