

半监督目标检测

半监督目标检测同时利用标签数据和无标签数据进行训练，一方面可以减少模型对检测框数量的依赖，另一方面也可以利用大量的未标记数据进一步提高模型。

按照以下流程进行半监督目标检测：

- [半监督目标检测](#)
 - [准备和拆分数据集](#)
 - [配置多分支数据流程](#)
 - [配置半监督数据加载](#)
 - [配置半监督模型](#)
 - [配置MeanTeacherHook](#)
 - [配置TeacherStudentValLoop](#)

准备和拆分数据集

我们提供了数据集下载脚本，默认下载 coco2017 数据集，并且自动解压。

```
python tools/misc/download_dataset.py
```

解压后的数据集目录如下：

```
mmdetection
├── data
│   └── coco
│       ├── annotations
│       │   ├── image_info_unlabeled2017.json
│       │   ├── instances_train2017.json
│       │   └── instances_val2017.json
│       ├── test2017
│       ├── train2017
│       ├── unlabeled2017
│       └── val2017
```

半监督目标检测在 coco 数据集上有两种比较通用的实验设置：

(1) 将 **train2017** 按照固定百分比（1%，2%，5% 和 10%）划分出一部分数据作为标签数据集，剩余的训练集数据作为无标签数据集，同时考虑划分不同的训练集数据作为标签数据集对半监督训练的结果影响较大，所以采用五折交叉验证来评估算法性能。我们提供了数据集划分脚本：

```
python tools/misc/split_coco.py
```

该脚本默认会按照 1%，2%，5% 和 10% 的标签数据占比划分 `train2017`，每一种划分会随机重复 5 次，用于交叉验证。生成的半监督标注文件名称格式如下：

- 标签数据集标注名称格式：`instances_train2017.{fold}@{percent}.json`
- 无标签数据集名称标注：`instances_train2017.{fold}@{percent}-unlabeled.json`

其中，`fold` 用于交叉验证，`percent` 表示标签数据的占比。划分后的数据集目录结构如下：

```
mmdetection
├── data
│   ├── coco
│   │   ├── annotations
│   │   │   ├── image_info_unlabeled2017.json
│   │   │   ├── instances_train2017.json
│   │   │   └── instances_val2017.json
│   │   └── semi_anns
│   │       ├── instances_train2017.1@1.json
│   │       ├── instances_train2017.1@1-unlabeled.json
│   │       ├── instances_train2017.1@2.json
│   │       ├── instances_train2017.1@2-unlabeled.json
│   │       ├── instances_train2017.1@5.json
│   │       ├── instances_train2017.1@5-unlabeled.json
│   │       ├── instances_train2017.1@10.json
│   │       ├── instances_train2017.1@10-unlabeled.json
│   │       ├── instances_train2017.2@1.json
│   │       └── instances_train2017.2@1-unlabeled.json
│   ├── test2017
│   ├── train2017
│   ├── unlabeled2017
│   └── val2017
```

(2) 将 `train2017` 作为标签数据集，`unlabeled2017` 作为无标签数据集。由于 `image_info_unlabeled2017.json` 没有 `categories` 信息，无法初始化 `CocoDataset`，所以需要将 `instances_train2017.json` 的 `categories` 写入 `image_info_unlabeled2017.json`，另存为 `instances_unlabeled2017.json`，相关脚本如下：

```
from mmengine.fileio import load, dump

anns_train = load('instances_train2017.json')
anns_unlabeled = load('image_info_unlabeled2017.json')
anns_unlabeled['categories'] = anns_train['categories']
dump(anns_unlabeled, 'instances_unlabeled2017.json')
```

处理后的数据集目录如下：

```
mmdetection
├── data
```

```
├── coco
│   ├── annotations
│   │   ├── image_info_unlabeled2017.json
│   │   ├── instances_train2017.json
│   │   ├── instances_unlabeled2017.json
│   │   └── instances_val2017.json
│   ├── test2017
│   ├── train2017
│   ├── unlabeled2017
│   └── val2017
```

配置多分支数据流程

半监督学习有两个主要的方法，分别是[一致性正则化](#)和[伪标签](#)。一致性正则化往往需要一些精心的设计，而伪标签的形式比较简单，更容易拓展到下游任务。我们主要采用了基于伪标签的教师学生联合训练的半监督目标检测框架，对于标签数据和无标签数据需要配置不同的数据流程：（1）标签数据的数据流程：

```
# pipeline used to augment labeled data,
# which will be sent to student model for supervised training.
sup_pipeline = [
    dict(type='LoadImageFromFile', backend_args = backend_args),
    dict(type='LoadAnnotations', with_bbox=True),
    dict(type='RandomResize', scale=scale, keep_ratio=True),
    dict(type='RandomFlip', prob=0.5),
    dict(type='RandAugment', aug_space=color_space, aug_num=1),
    dict(type='FilterAnnotations', min_gt_bbox_wh=(1e-2, 1e-2)),
    dict(type='MultiBranch', sup=dict(type='PackDetInputs'))
]
```

(2) 无标签的数据流程：

```
# pipeline used to augment unlabeled data weakly,
# which will be sent to teacher model for predicting pseudo instances.
weak_pipeline = [
    dict(type='RandomResize', scale=scale, keep_ratio=True),
    dict(type='RandomFlip', prob=0.5),
    dict(
        type='PackDetInputs',
        meta_keys=('img_id', 'img_path', 'ori_shape', 'img_shape',
                    'scale_factor', 'flip', 'flip_direction',
                    'homography_matrix')),
]

# pipeline used to augment unlabeled data strongly,
# which will be sent to student model for unsupervised training.
strong_pipeline = [
    dict(type='RandomResize', scale=scale, keep_ratio=True),
    dict(type='RandomFlip', prob=0.5),
    dict(
```

```

        type='RandomOrder',
        transforms=[
            dict(type='RandAugment', aug_space=color_space, aug_num=1),
            dict(type='RandAugment', aug_space=geometric, aug_num=1),
        ]),
    dict(type='RandomErasing', n_patches=(1, 5), ratio=(0, 0.2)),
    dict(type='FilterAnnotations', min_gt_bbox_wh=(1e-2, 1e-2)),
    dict(
        type='PackDetInputs',
        meta_keys=('img_id', 'img_path', 'ori_shape', 'img_shape',
                   'scale_factor', 'flip', 'flip_direction',
                   'homography_matrix')),
]

# pipeline used to augment unlabeled data into different views
unsup_pipeline = [
    dict(type='LoadImageFromFile', backend_args = backend_args),
    dict(type='LoadEmptyAnnotations'),
    dict(
        type='MultiBranch',
        unsup_teacher=weak_pipeline,
        unsup_student=strong_pipeline,
    )
]

```

配置半监督数据加载

(1) 构建半监督数据集。使用 `ConcatDataset` 拼接标签数据集和无标签数据集。

```

labeled_dataset = dict(
    type=dataset_type,
    data_root=data_root,
    ann_file='annotations/instances_train2017.json',
    data_prefix=dict(img='train2017/'),
    filter_cfg=dict(filter_empty_gt=True, min_size=32),
    pipeline=sup_pipeline)

unlabeled_dataset = dict(
    type=dataset_type,
    data_root=data_root,
    ann_file='annotations/instances_unlabeled2017.json',
    data_prefix=dict(img='unlabeled2017/'),
    filter_cfg=dict(filter_empty_gt=False),
    pipeline=unsup_pipeline)

train_dataloader = dict(
    batch_size=batch_size,
    num_workers=num_workers,
    persistent_workers=True,
    sampler=dict(
        type='GroupMultiSourceSampler',

```

```

        batch_size=batch_size,
        source_ratio=[1, 4]),
    dataset=dict(
        type='ConcatDataset', datasets=[labeled_dataset,
        unlabeled_dataset]))

```

(2) 使用多源数据集采样器。使用 `GroupMultiSourceSampler` 从 `labeled_dataset` 和 `labeled_dataset` 采样数据组成 `batch` , `source_ratio` 控制 `batch` 中标签数据和无标签数据的占比。`GroupMultiSourceSampler` 还保证了同一个 `batch` 中的图片具有相近的长宽比例, 如果不需要保证 `batch` 内图片的长宽比例, 可以使用 `MultiSourceSampler`。`GroupMultiSourceSampler` 采样示意图如下:



`sup=1000` 表示标签数据集的规模为 1000 , `sup_h=200` 表示标签数据集中长宽比大于等于1的图片规模为 200, `sup_w=800` 表示标签数据集中长宽比小于1的图片规模为 800 , `unsup=9000` 表示无标签数据集的规模为 9000 , `unsup_h=1800` 表示无标签数据集中长宽比大于等于1的图片规模为 1800, `unsup_w=7200` 表示标签数据集中长宽比小于1的图片规模为 7200 , `GroupMultiSourceSampler` 每次按照标签数据集和无标签数据集的图片的总体长宽比分布随机选择一组, 然后按照 `source_ratio` 从两个数据集中采样组成 `batch` , 因此标签数据集和无标签数据集重复采样次数不同。

配置半监督模型

我们选择 `Faster R-CNN` 作为 `detector` 进行半监督训练, 以半监督目标检测算法 `SoftTeacher` 为例, 模型的配置可以继承 `_base_/models/faster-rcnn_r50_fpn.py`, 将检测器的骨干网络替换成 `caffe` 风格。注意, 与监督训练的配置文件不同的是, `Faster R-CNN` 作为 `detector`, 是作为 `model` 的一个属性, 而不是 `model`。此外, 还需要将 `data_preprocessor` 设置为 `MultiBranchDataPreprocessor`, 用于处理不同数据流程图片的填充和归一化。最后, 可以通过 `semi_train_cfg` 和 `semi_test_cfg` 配置半监督训练和测试需要的参数。

```

_base_ = [
    '../_base_/models/faster-rcnn_r50_fpn.py',
    '../_base_/default_runtime.py',
    '../_base_/datasets/semi_coco_detection.py'
]

detector = _base_.model
detector.data_preprocessor = dict(
    type='DetDataPreprocessor',
    mean=[103.530, 116.280, 123.675],
    std=[1.0, 1.0, 1.0],
    bgr_to_rgb=False,
    pad_size_divisor=32)
detector.backbone = dict(
    type='ResNet',
    depth=50,
    num_stages=4,
    out_indices=(0, 1, 2, 3),
    frozen_stages=1,
    norm_cfg=dict(type='BN', requires_grad=False),
    norm_eval=True,

```

```
style='caffe',
init_cfg=dict(
    type='Pretrained',
    checkpoint='open-mmlab://detectron2/resnet50_caffe'))

model = dict(
    _delete_=True,
    type='SoftTeacher',
    detector=detector,
    data_preprocessor=dict(
        type='MultiBranchDataPreprocessor',
        data_preprocessor=detector.data_preprocessor),
    semi_train_cfg=dict(
        freeze_teacher=True,
        sup_weight=1.0,
        unsup_weight=4.0,
        pseudo_label_initial_score_thr=0.5,
        rpn_pseudo_thr=0.9,
        cls_pseudo_thr=0.9,
        reg_pseudo_thr=0.02,
        jitter_times=10,
        jitter_scale=0.06,
        min_pseudo_bbox_wh=(1e-2, 1e-2)),
    semi_test_cfg=dict(predict_on='teacher'))
```

此外，我们也支持其他检测模型进行半监督训练，比如，`RetinaNet` 和 `Cascade R-CNN`。由于 `SoftTeacher` 仅支持 `Faster R-CNN`，所以需要将其替换为 `SemiBaseDetector`，示例如下：

```
_base_ = [
    '../_base_/models/retinanet_r50_fpn.py',
    '../_base_/default_runtime.py',
    '../_base_/datasets/semi_coco_detection.py'
]

detector = _base_.model

model = dict(
    _delete_=True,
    type='SemiBaseDetector',
    detector=detector,
    data_preprocessor=dict(
        type='MultiBranchDataPreprocessor',
        data_preprocessor=detector.data_preprocessor),
    semi_train_cfg=dict(
        freeze_teacher=True,
        sup_weight=1.0,
        unsup_weight=1.0,
        cls_pseudo_thr=0.9,
        min_pseudo_bbox_wh=(1e-2, 1e-2)),
    semi_test_cfg=dict(predict_on='teacher'))
```

沿用 **SoftTeacher** 的半监督训练配置，将 **batch_size** 改为 2，**source_ratio** 改为 [1, 1]，**RetinaNet**，**Faster R-CNN**，**Cascade R-CNN** 以及 **SoftTeacher** 在 10% coco 训练集上的监督训练和半监督训练的实验结果如下：

Model	Detector	BackBone	Style	sup-0.1-coco mAP	semi-0.1-coco mAP
SemiBaseDetector	RetinaNet	R-50-FPN	caffe	23.5	27.7
SemiBaseDetector	Faster R-CNN	R-50-FPN	caffe	26.7	28.4
SemiBaseDetector	Cascade R-CNN	R-50-FPN	caffe	28.0	29.7
SoftTeacher	Faster R-CNN	R-50-FPN	caffe	26.7	31.1

配置MeanTeacherHook

通常，教师模型采用对学生模型指数滑动平均（EMA）的方式进行更新，进而教师模型随着学生模型的优化而优化，可以通过配置 **custom_hooks** 实现：

```
custom_hooks = [dict(type='MeanTeacherHook')]
```

配置TeacherStudentValLoop

由于教师学生联合训练框架存在两个模型，我们可以用 **TeacherStudentValLoop** 替换 **ValLoop**，在训练的过程中同时检验两个模型的精度。

```
val_cfg = dict(type='TeacherStudentValLoop')
```