MMdetection 目标检测框架的使用介绍

一、安装：

1. 从 github 下载源码：https://github.com/open-mmlab/mmdetection
2. 按照官网文件安装完毕后，进行简单测试，测试代码如下：

```
from mmdet.apis import init_detector, inference_detector, show_result
if __name__ == '__main__':
    config_file = 'configs/faster_rcnn_r50_fpn_1x.py'
    checkpoint_file = 'checkpoints/faster_rcnn_r50_fpn_1x_20181010-3d1b3351.pth'
    img_path = 'test.jpg'
    model = init_detector(config_file, checkpoint_file, device='cuda:0')
    result = inference_detector(model, img_path)
    show_result(img_path, result, model.CLASSES)
```

测试注意事项：

1. mmdetection 目前不支持 Windos 系统，我使用 Ubuntu16.04

2.`checkpoint_file` 是我在 mmdetection 目录下创建了一个 checkpoints 文件夹，然后手动下载

（链接：https://pan.baidu.com/s/1jC_9DJWrnwB8tm9LnGAHeQ 提取码：indv ）的权值，也可以自动下载的应该。

二、注册机制介绍：

```
import inspect

import mmcv


class Registry(object):
"""
这里主要实现一个 Registry 类，用来规范注册网络的各个模块，比如 backebone, neck,还有 dataset,pipeline 等等这些模块。
"""
    def __init__(self, name):
        self._name = name
```

```
        self._module_dict = dict()

    def __repr__(self):
        """这个函数主要是给类一个输出，我的理解是就是输出类相关信息。
            用下面的代码自己测试看看。
            #from mmdet.utils import Registry
            #print(Registry('backbone'))
        """
        format_str = self.__class__.__name__ + '(name={}, items={})'.format(
            self._name, list(self._module_dict.keys()))
        return format_str

    @property  #负责修饰一个对象函数,让类生成成员变量对应的 setter 和 getter 函数
    def name(self):
        return self._name

    @property
    def module_dict(self):
        return self._module_dict

    def get(self, key):
        return self._module_dict.get(key, None)

    def _register_module(self, module_class):
        """Register a module.
        Args:
            module (:obj:`nn.Module`): Module to be registered.
        """
        if not inspect.isclass(module_class):
            raise TypeError('module must be a class, but got {}'.format(
                type(module_class)))
        module_name = module_class.__name__
        if module_name in self._module_dict:
            raise KeyError('{} is already registered in {}'.format(
                module_name, self.name))
        self._module_dict[module_name] = module_class

    def register_module(self, cls):
        self._register_module(cls)
        return cls


def build_from_cfg(cfg, registry, default_args=None):
    """
```

这个函数大意就是从 config 文件中 cfg 各个模块使用 registry 进行注册。

```
Build a module from config dict.
Args:
    cfg (dict): Config dict. It should at least contain the key "type".
    registry (:obj:`Registry`): The registry to search the type from.
    default_args (dict, optional): Default initialization arguments.
Returns:
    obj: The constructed object.
"""
assert isinstance(cfg, dict) and 'type' in cfg
assert isinstance(default_args, dict) or default_args is None
args = cfg.copy()
obj_type = args.pop('type')
if mmcv.is_str(obj_type):
    obj_cls = registry.get(obj_type)
    if obj_cls is None:
        raise KeyError('{} is not in the {} registry'.format(
            obj_type, registry.name))
elif inspect.isclass(obj_type):
    obj_cls = obj_type
else:
    raise TypeError('type must be a str or valid type, but got {}'.format(
        type(obj_type)))
if default_args is not None:
    for name, value in default_args.items():
        args.setdefault(name, value)
return obj_cls(**args)
```

三、数据集介绍

mmdetection 现在支持 coco 和 voc 数据集格式，数据集的格式使用官方介绍的

mmdetection

---------- data

-----coco

-----VOCdevkit

主要嵌套关系是

以 mmdetection/mmdet/datasets/cityscapes.py 为例：

```
from .coco import CocoDataset
from .registry import DATASETS
@DATASETS.register_module
class CityscapesDataset(CocoDataset):
    """
    自定义coco格式数据类 <------CocoDataset <----------CustomDataset <----------Dataset(torch.utils.data.Dataset)
    VOCDataset<----------XMLDataset<----------CustomDataset<--------Dataset(同上)

    其他一些函数主要是用来读取分析 coco 的.json 和 voc 的 XML 文件的，具体细节先不谈，先了解整体框架
    """
    CLASSES = ('person', 'rider', 'car', 'truck', 'bus', 'train', 'motorcycle', 'bicycle')
```

四、混合精度计算

1. 关于混合精度计算的官方介绍：https://devblogs.nvidia.com/mixed-precision-training-deep-neural-networks/
2. Mmdetection 中关于混合精度计算的定义：

mmdetection/mmdet/core/fp16/decorators.py

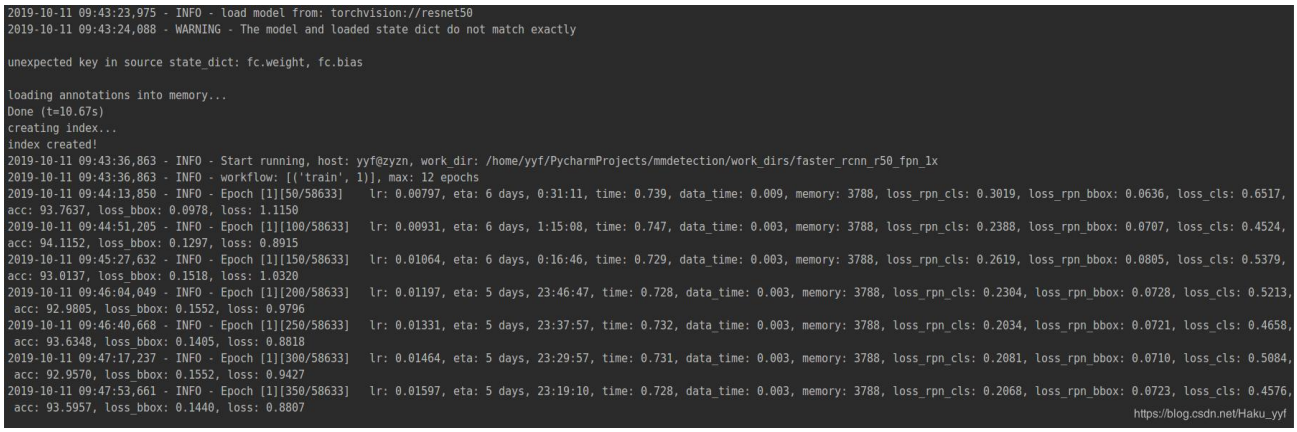
五、argparse 模块的使用

1. Argparse 模块的官方介绍：https://docs.python.org/3/library/argparse.html
2. Mmdetection 中使用 argparse 定义用户与命令行交互参数：

mmdetection/tools/train.py

3. 使用命令行进行模型训练：

在你的环境下终端 mmdetection 目录下：python tools/train.py configs/faster_rcnn_r50_fpn_1x.py --gpus 1



六、添加自定义 backbone：

现在我们开始深入算法模型来学习 ,这个小节我们尝试自定义一个 se_resnet50 来学习自定义 backbone。

在 mmdetection/mmdet/models/backbone 文件下创建一个 senet.py 的 python 文件。

自定义好类后，需要进行注册。

@BACKBONES.register_module

class SENet(nn.Module):

    pass

还有__init__.py 文件里也需要添加相应的模块：

from .hrnet import HRNet

from .resnet import ResNet, make_res_layer

from .resnext import ResNeXt

from .ssd_vgg import SSDVGG

from .senet import SENet

__all__ = ['SENet','ResNet', 'make_res_layer', 'ResNeXt', 'SSDVGG', 'HRNet']

最后在 mmdetection/configs/pascal_voc 目录下创建配置文件

faster_rcnn_ser50_fpn_1x_voc0712.py， 修改相应配置文件。

上面都做好后，还要重新运行 python setup.py develop 重新编译安装。

七、Mmdetection 中 config 文件详解：

在使用 mmdetection 对模型进行调优的过程中总会遇到很多参数的问题，不知道参数在代码中是什么作用，会对训练产生怎样的影响，这里我以 faster_rcnn_r50_fpn_1x.py 为例，简单介绍一下 mmdetection 中的各项参数含义:

```
# model settings
model = dict(
    type='FasterRCNN',                          # model 类型
    pretrained='modelzoo://resnet50',           # 预训练模型：imagenet-resnet50
```

```
    backbone=dict(
        type='ResNet',                          # backbone 类型
        depth=50,                               # 网络层数
        num_stages=4,                           # resnet 的 stage 数量
        out_indices=(0, 1, 2, 3),               # 输出的 stage 的序号
        frozen_stages=1,                        # 冻结的 stage 数量，即该 stage 不更新参数，-1 表示所有的 stage 都更新参数
        style='pytorch'),                       # 网络风格：如果设置 pytorch，则 stride 为 2 的层是 conv3x3 的卷积层；如果设置 caffe，则 stride 为 2 的层是第一个 conv1x1 的卷积层
    neck=dict(
        type='FPN',                             # neck 类型
        in_channels=[256, 512, 1024, 2048],     # 输入的各个 stage 的通道数
        out_channels=256,                       # 输出的特征层的通道数
        num_outs=5),                            # 输出的特征层的数量
    rpn_head=dict(
        type='RPNHead',                         # RPN 网络类型
        in_channels=256,                        # RPN 网络的输入通道数
        feat_channels=256,                      # 特征层的通道数
```

```
            anchor_scales=[8],                  # 生成的 anchor 的
baselen，baselen = sqrt(w*h)，w 和 h 为 anchor 的宽和高
            anchor_ratios=[0.5, 1.0, 2.0],      # anchor 的宽高比
            anchor_strides=[4, 8, 16, 32, 64],  # 在每个特征层上的 anchor
的步长（对应于原图）
            target_means=[.0, .0, .0, .0],      # 均值
            target_stds=[1.0, 1.0, 1.0, 1.0],   # 方差
            use_sigmoid_cls=True),              # 是否使用 sigmoid 来
进行分类，如果 False 则使用 softmax 来分类
        bbox_roi_extractor=dict(
            type='SingleRoIExtractor',                                  #
RoIExtractor 类型
            roi_layer=dict(type='RoIAlign', out_size=7, sample_num=2),  #
ROI 具体参数：ROI 类型为 ROIalign，输出尺寸为 7，sample 数为 2
            out_channels=256,
#  输出通道数
            featmap_strides=[4, 8, 16, 32]),                            #
特征图的步长
        bbox_head=dict(
            type='SharedFCBBoxHead',                    # 全连接层类
型
            num_fcs=2,                                  # 全连接层数量
```

```
            in_channels=256,                                    # 输入通道数
            fc_out_channels=1024,                               # 输出通道数
            roi_feat_size=7,                                    # ROI 特征层尺寸
            num_classes=81,                                     # 分类器的类别
数量+1，+1 是因为多了一个背景的类别
            target_means=[0., 0., 0., 0.],                      # 均值
            target_stds=[0.1, 0.1, 0.2, 0.2],                   # 方差
            reg_class_agnostic=False))                          # 是否采用
class_agnostic 的方式来预测，class_agnostic 表示输出 bbox 时只考虑其是否为前景，后续分类的时候再根据该 bbox 在网络中的类别得分来分类，也就是说一个框可以对应多个类别
# model training and testing settings
train_cfg = dict(
    rpn=dict(
        assigner=dict(
            type='MaxIoUAssigner',                  # RPN 网络的正负样本划分
            pos_iou_thr=0.7,                        # 正样本的 iou 阈值
            neg_iou_thr=0.3,                        # 负样本的 iou 阈值
            min_pos_iou=0.3,                        # 正样本的 iou 最小值。如果 assign 给 ground truth 的 anchors 中最大的 IOU 低于 0.3，则忽略所有的 anchors，否则保留最大 IOU 的 anchor
```

```
            ignore_iof_thr=-1),                    # 忽略 bbox 的阈值，当
ground truth 中包含需要忽略的 bbox 时使用，-1 表示不忽略
        sampler=dict(
            type='RandomSampler',                  # 正负样本提取器类型
            num=256,                               # 需提取的正负样本数
量
            pos_fraction=0.5,                      # 正样本比例
            neg_pos_ub=-1,                         # 最大负样本比例，大于
该比例的负样本忽略，-1 表示不忽略
            add_gt_as_proposals=False),            # 把 ground truth 加入
proposal 作为正样本
        allowed_border=0,                          # 允许在 bbox 周围外扩
一定的像素
        pos_weight=-1,                             # 正样本权重，-1 表示不
改变原始的权重
        smoothl1_beta=1 / 9.0,                     # 平滑 L1 系数
        debug=False),                              # debug 模式
    rcnn=dict(
        assigner=dict(
            type='MaxIoUAssigner',                 # RCNN 网络正负样本划
分
            pos_iou_thr=0.5,                       # 正样本的 iou 阈值
```

```
            neg_iou_thr=0.5,                    # 负样本的 iou 阈值
            min_pos_iou=0.5,                    # 正样本的 iou 最小值。如果 assign 给 ground truth 的 anchors 中最大的 IOU 低于 0.3，则忽略所有的 anchors，否则保留最大 IOU 的 anchor
            ignore_iof_thr=-1),                 # 忽略 bbox 的阈值，当 ground truth 中包含需要忽略的 bbox 时使用，-1 表示不忽略
        sampler=dict(
            type='RandomSampler',               # 正负样本提取器类型
            num=512,                            # 需提取的正负样本数量
            pos_fraction=0.25,                  # 正样本比例
            neg_pos_ub=-1,                      # 最大负样本比例，大于该比例的负样本忽略，-1 表示不忽略
            add_gt_as_proposals=True),          # 把 ground truth 加入 proposal 作为正样本
        pos_weight=-1,                          # 正样本权重，-1 表示不改变原始的权重
        debug=False))                           # debug 模式
test_cfg = dict(
    rpn=dict(                                   # 推断时的 RPN 参数
        nms_across_levels=False,                # 在所有的 fpn 层内做 nms
```

```
            nms_pre=2000,                                   # 在 nms 之前保留的的得分最高的 proposal 数量
            nms_post=2000,                                  # 在 nms 之后保留的的得分最高的 proposal 数量
            max_num=2000,                                   # 在后处理完成之后保留的 proposal 数量
            nms_thr=0.7,                                    # nms 阈值
            min_bbox_size=0),                               # 最小 bbox 尺寸
        rcnn=dict(
            score_thr=0.05,          nms=dict(type='nms',          iou_thr=0.5), max_per_img=100)     # max_per_img 表示最终输出的 det bbox 数量
        # soft-nms is also supported for rcnn testing
        #   e.g.,   nms=dict(type='soft_nms',   iou_thr=0.5,   min_score=0.05) # soft_nms 参数
)
# dataset settings
dataset_type = 'CocoDataset'                                # 数据集类型
data_root = 'data/coco/'                                    # 数据集根目录
img_norm_cfg = dict(
    mean=[123.675,   116.28,   103.53],   std=[58.395,   57.12,   57.375], to_rgb=True)     # 输入图像初始化，减去均值 mean 并处以方差 std，to_rgb 表示将 bgr 转为 rgb
```

```
data = dict(
    imgs_per_gpu=2,                    # 每个 gpu 计算的图像数量
    workers_per_gpu=2,                 # 每个 gpu 分配的线程数
    train=dict(
        type=dataset_type,
# 数据集类型
        ann_file=data_root + 'annotations/instances_train2017.json',
# 数据集 annotation 路径
        img_prefix=data_root + 'train2017/',
# 数据集的图片路径
        img_scale=(1333, 800),
# 输入图像尺寸，最大边 1333，最小边 800
        img_norm_cfg=img_norm_cfg,
# 图像初始化参数
        size_divisor=32,
# 对图像进行 resize 时的最小单位，32 表示所有的图像都会被 resize 成 32 的
倍数
        flip_ratio=0.5,
# 图像的随机左右翻转的概率
        with_mask=False,
# 训练时附带 mask
        with_crowd=True,
```

```
# 训练时附带 difficult 的样本
        with_label=True),
# 训练时附带 label
    val=dict(
        type=dataset_type,
# 同上
        ann_file=data_root + 'annotations/instances_val2017.json',
# 同上
        img_prefix=data_root + 'val2017/',
# 同上
        img_scale=(1333, 800),
# 同上
        img_norm_cfg=img_norm_cfg,
# 同上
        size_divisor=32,
# 同上
        flip_ratio=0,
# 同上
        with_mask=False,
# 同上
        with_crowd=True,
# 同上
```

```
        with_label=True),
# 同上
    test=dict(
        type=dataset_type,
# 同上
        ann_file=data_root + 'annotations/instances_val2017.json',
# 同上
        img_prefix=data_root + 'val2017/',
# 同上
        img_scale=(1333, 800),
# 同上
        img_norm_cfg=img_norm_cfg,
# 同上
        size_divisor=32,
# 同上
        flip_ratio=0,
# 同上
        with_mask=False,
# 同上
        with_label=False,
# 同上
        test_mode=True))
```

```
# 同上
# optimizer
optimizer = dict(type='SGD', lr=0.02, momentum=0.9, weight_decay=0.0001)  # 优化参数，lr 为学习率，momentum 为动量因子，weight_decay 为权重衰减因子
optimizer_config = dict(grad_clip=dict(max_norm=35, norm_type=2))  # 梯度均衡参数
# learning policy
lr_config = dict(
    policy='step',                          # 优化策略
    warmup='linear',                        # 初始的学习率增加的策略，linear 为线性增加
    warmup_iters=500,                       # 在初始的 500 次迭代中学习率逐渐增加
    warmup_ratio=1.0 / 3,                   # 起始的学习率
    step=[8, 11])                           # 在第 8 和 11 个 epoch 时降低学习率
checkpoint_config = dict(interval=1)        # 每 1 个 epoch 存储一次模型
# yapf:disable
log_config = dict(
    interval=50,                            # 每 50 个 batch 输出一次信息
    hooks=[
```

```
        dict(type='TextLoggerHook'),          # 控制台输出信息的风格
        # dict(type='TensorboardLoggerHook')
    ])
# yapf:enable
# runtime settings
total_epochs = 12                                   # 最大 epoch 数
dist_params = dict(backend='nccl')                  # 分布式参数
log_level = 'INFO'                                  # 输出信息的完整度级别
work_dir = './work_dirs/faster_rcnn_r50_fpn_1x' # log 文件和模型文件存储路径
load_from = None                                    # 加载模型的路径，None 表示从预训练模型加载
resume_from = None                                  # 恢复训练模型的路径
workflow = [('train', 1)]                           # 当前工作区名称
```

八、展示 mmdetection 的推理结果：

Mmdetection 只给出了各个模型训练后的 map 计算值，如果你需要将推理结果打印到图片上需要自己扩展，这里给出我的扩展，可供参考：

创建 mmdetection/tools/bbox_show.py 文件：

```
import numpy as np
import mmcv
```

```
import cv2
import json
from pycocotools.coco import COCO
from mmcv import color_val
from mmdet.apis import show_result
from argparse import ArgumentParser
import os

class_names = ['category_1', 'category_2', 'category_3']

def det_bbox_show(args=None):
    """ 这个是用来展示检测结果的 """
    with open(args.json_file, 'r') as f:
        jsondata = json.load(f)
        results = {}
        for result in jsondata:
            result['bbox'][2] = result['bbox'][0] + result['bbox'][2]
            result['bbox'][3] = result['bbox'][1] + result['bbox'][3]
            result['bbox'].append(result['score'])
            if result['image_id'] in results.keys():
                results[result['image_id']][result['category_id']-1]          =
np.concatenate((results[result['image_id']][result['category_id']-1],
```

```
np.array(result['bbox'], dtype=np.float32).reshape(-1,5)))
                    else:
                        results[result['image_id']] = []
                        for i in range(len(class_names)):
                            if result['category_id']-1 == i:

results[result['image_id']].append(np.array(result['bbox'],
dtype=np.float32).reshape(-1,5))
                            else:

results[result['image_id']].append(np.zeros(shape=(0,5),dtype=np.float3
2))
        for k, v in results.items():
            show_result(args.img_prex + '/' + k + '.jpg', v, class_names,
score_thr=0.7, show=False, out_file=args.outfile + '/{}.jpg'.format(k))



def gt_bbox_show(args):
        """这个是用来展示 Ground Truth bbox 的"""
        coco = COCO(args.json_file)
        img2ann = coco.imgToAnns
        imgids = coco.getImgIds()
```

```
    for imgid in imgids:
        filename = args.img_prex + '/{}.jpg'.format(imgid)
        img = mmcv.imread(filename)
        bbox_color = color_val('red')
        text_color = color_val('red')
        anns = img2ann[imgid]
        for ann in anns:
            bbox_int = np.array(ann['bbox']).astype(np.int32)
            left_top = (bbox_int[0], bbox_int[1])
            right_bottom = (bbox_int[0] + bbox_int[2], bbox_int[1] +
bbox_int[3])
            label = ann['category_id'] - 1
            cv2.rectangle(img, left_top, right_bottom, bbox_color,
thickness=1)
            label_text = class_names[label]
            cv2.putText(img, label_text, (bbox_int[0], bbox_int[1] - 2),
cv2.FONT_HERSHEY_COMPLEX, 0.5, text_color)
        mmcv.imwrite(img, args.outfile + '/{}.jpg'.format(imgid))


def main():
    parser = ArgumentParser(description='COCO bbox show Tool')
    parser.add_argument(
```

```
        '--gt-bbox-show',
        action='store_true',
        help='show gt bbox'
    )
    parser.add_argument(
        '--json_file',
        type=str,
        help='json file(results or ann file)'
    )
    parser.add_argument(
        '--img_prex',
        type=str,
        help='img dataset prex'
    )
    parser.add_argument(
        '--outfile',
        type=str,
        help='outfile'
    )
    args = parser.parse_args()
    if not os.path.exists(args.outfile):
        os.makedirs(args.outfile)
```

```
    if args.gt_bbox_show:
        gt_bbox_show(args)
    else:
        det_bbox_show(args)

if __name__ == '__main__':
    import time
    tim = time.time()
    main()
    print("total time = {}".format(time.time() - tim))
```

这段程序前提是图片名称和图片 id 要保持一致，像 coco 这种图片名前面补好多 0 的就不能用了，需要 debug 一下。

参考博客： https://blog.csdn.net/Haku_yyf