yolo使用说明

- 模型的所有代码为pytorch版本，且在矩池云的GPU服务器上训练测试，所以代码中路径的配置均以矩池云中的路径配置为标准（/mnt），这里的文件夹中包含用官方COCO数据集训练完的完整的网络数据。
- yolo文件夹格式如下：

```
---- yolo
    ---- result_imgs
    ---- test_imgs
    ---- coco.names
    ---- yolo.py
    ---- yolov3.cfg
    ---- yolov3.weight
```

- result_imgs : 储存识别后的图片
- test_imgs : 储存要识别的图片
- coco.names : 储存官方COCO数据集中的类别名称，这里包含80种类别，这里适用的有以下八类：person , bicycle , car , motorbike , traffic light , fire hydrant , stop sigh , bench。
- yolo.py : 物体识别的代码，可以通过修改下面八个参数来调整输入和输出。

```
# pathIn：原始图片的路径
# pathOut：结果图片的路径
# label_path：类别标签文件的路径
# config_path：模型配置文件的路径
# weights_path：模型权重文件的路径
# confidence_thre：0-1，置信度（概率/打分）阈值，即保留概率大于这个值的边界框，默认为0.5
# nms_thre：非极大值抑制的阈值，默认为0.3
# jpg_quality：设定输出图片的质量，范围为0到100，默认为80，越大质量越好
```

- yolov3.cfg : 使用官方COCO数据集训练好的模型网络参数
- yolov3.weights : 使用官方COCO数据集训练好的权重参数。

- 因为还有一些物体需要识别，例如空调外机、井盖、捕虫器、路障、垃圾桶等等，所以需要自己拍摄图片进行标注和训练，考虑到SSD和yolov3对于官方权威标准的数据集训练效果大致相同，再加上SSD需要的xml文件可以直接由LabelImg软件标注生成，所以后面尝试用SSD进行自己数据集的训练。

SSD使用说明

- 模型的所有代码为pytorch版本，且在矩池云的GPU服务器上训练测试，所以代码中路径的配置均以矩池云中的路径配置为标准（/mnt）。

1. 配置环境

1. 下载适合的torch版本

```
pip3 install https://download.pytorch.org/whl/cu100/torch-1.0.0-cp36-cp36m-linux_x86_64.whl
```

2. 配置其他环境

```
cd /mnt/ssd
pip install -r requirements.txt  --user
```

2. 数据集的准备

- 这里我们使用的数据集为我们组拍摄的两千张照片，并使用LabelImg进行标注生成对应的xml文件，储存在data文件夹中，data文件夹的结构如下：

```
└── data
    └── maskornot
        ├── Annotations
        ├── ImageSets
        │   └── Main
        │       ├── test.txt
        │       ├── train.txt
        │       ├── trainval.txt
        │       └── val.txt
        ├── JPEGImages
        ├── create_txt.py
        └── jieya.py
    ├── _init_.py
    ├── config.py
    └── mask.py
```

- Annotations : 储存每张图片的标注信息，xml格式文件
- JPEGImages： 储存每张图片
- ImageSets ：用于分类训练集和测试集的文本文件
  - train.txt是各个分类的训练图片
  - val.txt是各个分类的验证图片
  - trainval.txt 是各个分类的训练和验证图片
  - test.txt是各个分类的测试图片
- create_txt.py： 用于随机生成训练集和测试集txt文件的代码，这里可以控制调整参数改变训练集和测试集的大小，这里默认设置70%的trainval和30%的test，其中trainval中85%的train和15%的val。

3. 训练过程(train)

- 通过修改下面的代码可以实现自己的数据集的SSD模型训练（其他运行代码修改完毕，这里不进行赘述）

1. 修改config.py

```
cd /mnt/ssd/data
# 可以修改参数num_classes,max_iter
mask = {
    'num_classes': 11, # 种类数 + 1
    'lr_steps': (80000, 100000, 120000),
    'max_iter': 10000, # 训练迭代次数
```

```
    'feature_maps': [38, 19, 10, 5, 3, 1],
    'min_dim': 300,
    'steps': [8, 16, 32, 64, 100, 300],
    'min_sizes': [30, 60, 111, 162, 213, 264],
    'max_sizes': [60, 111, 162, 213, 264, 315],
    'aspect_ratios': [[2], [2, 3], [2, 3], [2, 3], [2], [2]],
    'variance': [0.1, 0.2],
    'clip': True,
    'name': 'MASK',
}
```

2. 修改mask.py

```
cd /mnt/ssd/data
# 可以修改参数MASK_CLASSES,MASK_ROOT
# 类别名称
MASK_CLASSES = ('ac','car','rb','ev','bike','mc','fh','mt','tree','tc')
# 空调外机，车，路障，电动车，自行车，井盖，消防栓，捕蝇器，树，垃圾桶
# 数据集目录地址
MASK_ROOT = osp.join(HOME, "data/maskornot/")
```

3. 修改ssd.py

```
cd /mnt/ssd
# 可以修改num_classes
self.cfg = (coco, voc, mask)[num_classes == 11] # 等于种类数 + 1
```

4. 训练train.py

```
cd /mnt/ssd
# dataset 数据集代号 dataset_root 脚本代号 learning-rate 学习率
python train.py --dataset MASK --dataset_root MASK_ROOT --learning-rate 1e-5
```

5. 部分训练的效果图

```
Loading base network...
Initializing weights...
Loading the dataset...
Training SSD on: MASK
Using the specified args:
Namespace(basenet='vgg16_reducedfc.pth', batch_size=32, cuda=True,
dataset='MASK', dataset_root='MASK_ROOT', gamma=0.1, lr=1e-05, momentum=0.9,
num_workers=4, resume=None, save_folder='weights/', start_iter=0, visdom=False,
weight_decay=0.0005)
/usr/local/lib/python3.6/dist-packages/torch/nn/_reduction.py:43: UserWarning:
size_average and reduce args will be deprecated, please use reduction='sum'
instead.
  warnings.warn(warning.format(ret))
timer: 10.9482 sec.
iter 0 || Loss: 19.6539 || timer: 0.4676 sec.
iter 10 || Loss: 18.4427 || timer: 0.2489 sec.
iter 20 || Loss: 16.3053 || timer: 0.4680 sec.
iter 30 || Loss: 14.5833 || timer: 0.4679 sec.
iter 40 || Loss: 12.6759 || timer: 0.2453 sec.
iter 50 || Loss: 11.4251 || timer: 0.4731 sec.
```

```
iter 60 || Loss: 9.5738 || timer: 0.4748 sec.
......
```

4. 验证过程(eval)

- eval.py具体详情见代码
- 验证eval.py

```
cd ./mnt/ssd
# trained_model 训练好的网络名称
python eval.py --trained_model ./weights/ssd300_AC_5000.pth --top_k 1
```

- 部分运行结果

```
Finished loading model!
/pytorch/torch/csrc/autograd/python_function.cpp:622: UserWarning: Legacy
autograd function with non-static forward method is deprecated and will be
removed in 1.3. Please use new-style autograd function with static forward
method. (Example:
https://pytorch.org/docs/stable/autograd.html#torch.autograd.Function)
im_detect: 1/557 2.360s
。。。。。。
/pytorch/torch/csrc/autograd/python_function.cpp:622: UserWarning: Legacy
autograd function with non-static forward method is deprecated and will be
removed in 1.3. Please use new-style autograd function with static forward
method. (Example:
https://pytorch.org/docs/stable/autograd.html#torch.autograd.Function)
im_detect: 557/557 0.152s
Evaluating detections
Writing mask MASK results file
Writing nomask MASK results file
VOC07 metric? Yes
AP for ac = 0.3672
AP for car = 0.1625
AP for rb = 0.2644
AP for ev = 0.0025
AP for bike = 0.0082
AP for mc = 0.1629
AP for fh = 0.2982
AP for mt = 0.2629
AP for tree = 0.0002
AP for tc = 0.1629
Mean AP = 0.1692
~~~~~~~~
Results:
0.367
0.163
0.265
0.003
0.008
0.163
0.298
0.263
0.000
0.163
0.169
```

```
~~~~~~~~

----------------------------------------------------------------
Results computed with the **unofficial** Python eval code.
Results should be very close to the official MATLAB eval code.
----------------------------------------------------------------
```

```
# 空调外机，车，路障，电动车，自行车，井盖，消防栓，捕蝇器，树，垃圾桶
# 'ac','car','rb','ev','bike','mc','fh','mt','tree','tc'
```

分析结果发现：

1. 相较于COCO数据集，因为拍摄的图片中首先每类物体样式少，其次像素较专业的摄像机针对性不强，每张图片中的噪点较多，对于官方的SSD算法训练难度大，最后又因为标注标准不如COCO数据集专业，导致模型在ac，fh等物体识别中效果较差。同时，物体拍摄不清晰或像素较小时，识别效果较差。
2. 因为学校中的“树”差距较大，缺乏像COCO数据集那样大而全面的数据图片集，导致训练过程中SSD算法未能找到优良的特征；且大多时候树连成一片，对标注工作造成了很大的困扰，极大地受到了噪音的影响。从而影响了模型训练，所以对tree的识别正确率低。
3. 在学校拍摄自行车和电动车时因为两者现实中差距较小，外加上种类繁多，且多扎堆放置，甚至部分车增置了防风罩，导致ev和bike的识别正确率低。相较于上节的yolo模型，识别效果相比差距大。

针对上述问题总结如下，训练的网络因下列几种原因导致效果差：

1. 物体本身种类较多，且受拍摄数量影响，无法根据有限的图片拾取有用的特征信息。
2. 受拍摄环境、拍摄设备等因素影响，所建立的数据集图片质量较COCO数据集差距较大。
3. 所拍图片中背景中噪音太大，因为标注时是用矩形框选图片，往往会导致很多背景被网络认为是物体的一部分，从而导致训练偏差增大。这也是数据集建立中最大的问题。

改进方向：

下一步可以在权威的数据集网站中查找数据集，而不是用自己所拍摄的图片，过程中要尽量寻找多而优的照片，尽量寻找背景噪音小的图片进行标注，尽可能减少模型训练中不必要的误差，尽量最终数据集效果与COCO数据集类似，使模型最后训练效果更好。

5.测试过程(test)

- test.py具体详情见代码
- 测试test.py

```
cd ./mnt/ssd
# trained_model 训练好的网络名称
python test.py --trained_model ./weights/ssd300_AC_5000.pth
```

- 部分运行结果

```
Finished loading model!
Testing image 1/557....
/pytorch/torch/csrc/autograd/python_function.cpp:622: UserWarning: Legacy
autograd function with non-static forward method is deprecated and will be
removed in 1.3. Please use new-style autograd function with static forward
method. (Example:
https://pytorch.org/docs/stable/autograd.html#torch.autograd.Function)
Testing image 2/557....
。。。。。。
/pytorch/torch/csrc/autograd/python_function.cpp:648: UserWarning: Legacy
autograd function object was called twice.  You will probably get incorrect
gradients from this computation, as the saved tensors from the second invocation
will clobber the saved tensors from the first invocation.  Please consider
rewriting your autograd function in the modern style; for information on the new
format, please see:
https://pytorch.org/docs/stable/notes/extending.html#extending-torch-autograd
Testing image 557/557....
```

- 生成文档

```
cd /mnt/ssd/eval/test1.txt
GROUND TRUTH FOR: test_000001500
label: 216.0 || 155.0 || 986.0 || 759.0 || 0
PREDICTIONS:
1 label: ac score: tensor(0.9097) 215.426735 || 150.9471 || 960.25903 ||
740.36295
。。。。。。
```

6. 可视化

- 因为在云服务器上例如plt等函数无法使用，所以这里通过编写代码直接生成图片保存在云盘上。
- 代码详情见/mnt/ssd/demo/keshi.py