AlphaRotate Documentation

Xue Yang

Nov 03, 2021

USER GUIDE

AlphaRotate is an open-source Tensorflow benchmark for performing scalable rotation detection on various datasets, which is maintained by Xue Yang with Shanghai Jiao Tong University supervised by Prof. Junchi Yan.

This repository is developed for the following purposes:

- **Providing modules** for developing rotation detection algorithms to facilitate future research.
- **Providing implementation** of state-of-the-art rotation detection methods.
- **Benchmarking** existing rotation detection algorithms under different dataset & experiment settings, for the purpose of fair comparison.

CHAPTER

ONE

INTRODUCTION TO ROTATION DETECTION

Arbitrary-oriented objects are ubiquitous for detection across visual datasets, such as aerial images, scene text, face and 3D objects, retail scenes, etc. Compared with the large literature on horizontal object detection, research in oriented object detection is relatively in its earlier stage, with many open problems to solve.

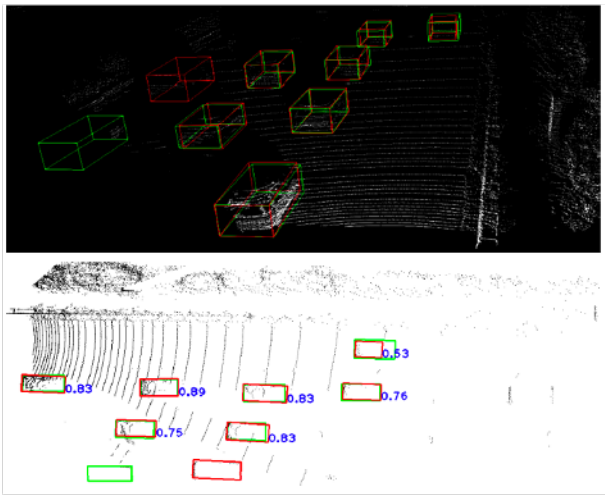
Rotation detection techniques have been applied to the following applications:

- Aerial images

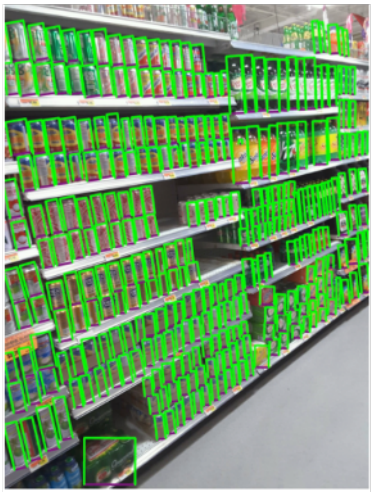
- Scene text

- Face

- 3D object detection



- Retail scenes



- and more…

In this repository, we mainly focus on aerial images due to its challenging.

Readers are referred to the following survey for more technical details about aerial image rotation detection: DOTA-DOAI

CHAPTER

TWO

INSTALLATION

2.1 Docker

We recommend using docker images if docker or other container runtimes e.g. singularity is available on your devices.

We maintain a prebuilt image at dockerhub:

```
yangxue2docker/yx-tf-det:tensorflow1.13.1-cuda10-gpu-py3
```

Note: For 30xx series graphics cards (cuda11), please download image from tensorflow-release-notes according to your development environment, e.g. nvcr.io/nvidia/tensorflow:20.11-tf1-py3

2.2 Manual configuration

This repository is developed and tested with ubuntu 16.04, python 3.5 (anaconda recommend), tensorflow-gpu 1.13, cuda 10.0, opencv-python 4.1.1.26, tqdm 4.54.0, Shapely 1.7.1, tfplot 0.2.0 (optional). If docker is not available, we provide detailed steps to install the requirements by `apt` and `pip`.

Note: For 30xx series graphics cards (cuda11), we recommend this blog to install tf1.xx

CHAPTER

THREE

RUN THE EXPERIMENT

3.1 Download Model

3.1.1 Pretrain weights

Download a pretrain weight you need from the following three options, and then put it to pretrained_weights.

- **MxNet pretrain weights (recommend in this repo, default in NET_NAME): resnet_v1d, resnet_v1b, refer to gluon2TF.**

    1) Baidu Drive (5ht9)

    2) Google Drive

- Tensorflow pretrain weights: resnet50_v1, resnet101_v1, resnet152_v1, efficientnet, mobilenet_v2, darknet53 (Baidu Drive (1jg2), Google Drive).
- **Pytorch pretrain weights, refer to pretrain_zoo.py and Others.**

    1) Baidu Drive (oofm)

    2) Google Drive

3.1.2 Trained weights

Please download trained models by this project, then put them to trained_weights.

3.2 Compile

```
cd $PATH_ROOT/libs/utils/cython_utils
rm *.so
rm *.c
rm *.cpp
python setup.py build_ext --inplace (or make)

cd $PATH_ROOT/libs/utils/
rm *.so
rm *.c
rm *.cpp
python setup.py build_ext --inplace
```

3.3 Train

- **If you want to train your own dataset, please note:**
    1) Select the detector and dataset you want to use, and mark them as #DETECTOR and #DATASET (such as #DETECTOR=retinanet and #DATASET=DOTA)
    2) Modify parameters (such as CLASS_NUM, DATASET_NAME, VERSION, etc.) in $PATH_ROOT/libs/configs/#DATASET/#DETECTOR/cfgs_xxx.py
    3) Copy $PATH_ROOT/libs/configs/#DATASET/#DETECTOR/cfgs_xxx.py to $PATH_ROOT/libs/configs/cfgs.py
    4) Add category information in $PATH_ROOT/libs/label_name_dict/label_dict.py
    5) Add data_name to $PATH_ROOT/dataloader/dataset/read_tfrecord.py
- **Make tfrecord**

If image is very large (such as DOTA dataset), the image needs to be cropped. Take DOTA dataset as a example:

```
cd $PATH_ROOT/dataloader/dataset/DOTA
python data_crop.py
```

If image does not need to be cropped, just convert the annotation file into xml format, refer to example.xml.

```
cd $PATH_ROOT/dataloader/dataset/
python convert_data_to_tfrecord.py --root_dir='/PATH/TO/DOTA/'
                                   --xml_dir='labeltxt'
                                   --image_dir='images'
                                   --save_name='train'
                                   --img_format='.png'
                                   --dataset='DOTA'
```

- **Start training**

```
cd $PATH_ROOT/tools/#DETECTOR
python train.py
```

3.4 Train and Evaluation

- For large-scale image, take DOTA dataset as a example (the output file or visualization is in $PATH_ROOT/tools/#DETECTOR/test_dota/VERSION):

```
cd $PATH_ROOT/tools/#DETECTOR
python test_dota.py --test_dir='/PATH/TO/IMAGES/'
                    --gpus=0,1,2,3,4,5,6,7
                    -ms (multi-scale testing, optional)
                    -s (visualization, optional)
                    -cn (use cpu nms, slightly better <1% than gpu nms but slower,␣
↪optional)
```

or (recommend in this repo, better than multi-scale testing)

```
python test_dota_sota.py --test_dir='/PATH/TO/IMAGES/'
                         --gpus=0,1,2,3,4,5,6,7
                         -s (visualization, optional)
                         -cn (use cpu nms, slightly better <1% than gpu nms but slower,␣
↪optional)
```

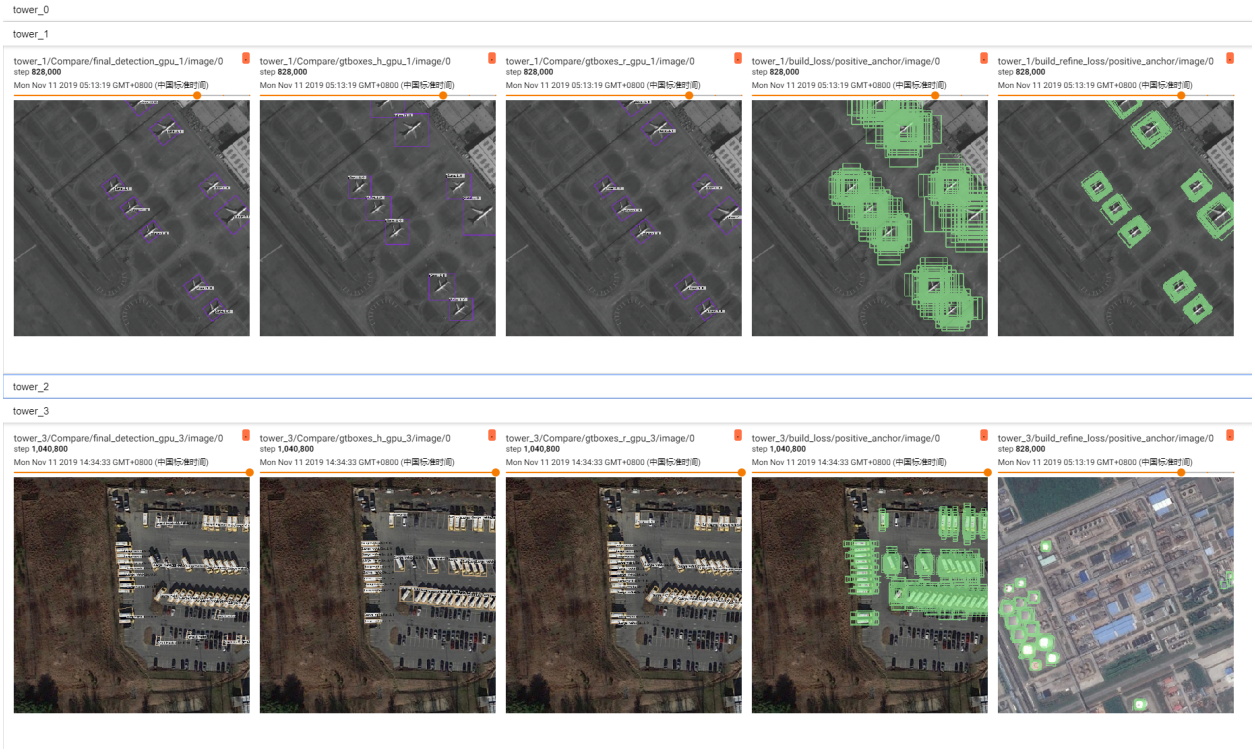
Note: In order to set the breakpoint conveniently, the read and write mode of the file is' a+'. If the model of the same `#VERSION` needs to be tested again, the original test results need to be deleted.

- For small-scale image, take HRSC2016 dataset as a example:

```
cd $PATH_ROOT/tools/#DETECTOR
python test_hrsc2016.py --test_dir='/PATH/TO/IMAGES/'
                        --gpu=0
                        --image_ext='bmp'
                        --test_annotation_path='/PATH/TO/ANNOTATIONS'
                        -s (visualization, optional)
```

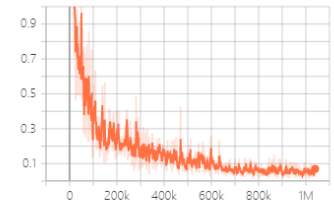
- Tensorboard

```
cd $PATH_ROOT/output/summary
tensorboard --logdir=.
```



cls

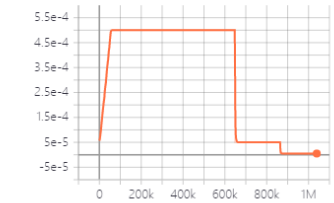
cls_loss
tag: cls/cls_loss



get_batch

lr

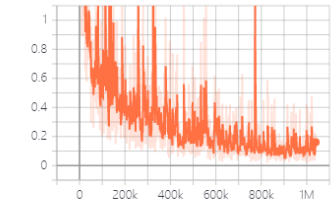
lr



refine

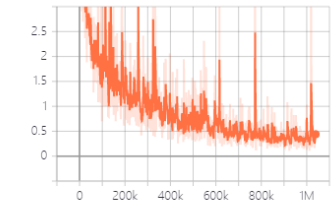
reg

reg_loss
tag: reg/reg_loss



total

total_losses
tag: total/total_losses



CHAPTER

FOUR

MODELS

CHAPTER

FIVE

API REFERENCE

5.1 utils

5.1.1 densely_coded_label

5.1.2 smooth_label

5.1.3 gaussian_metric