AlphaRotate Documentation

Xue Yang

Jan 22, 2023

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.

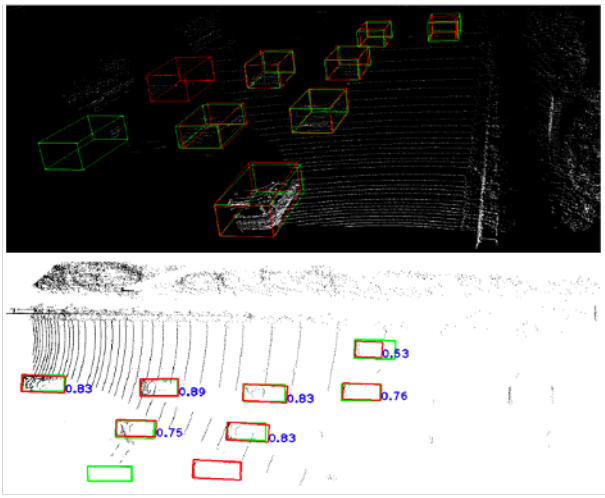
Rotation detection techniques have been applied to the following applications:

- Aerial images

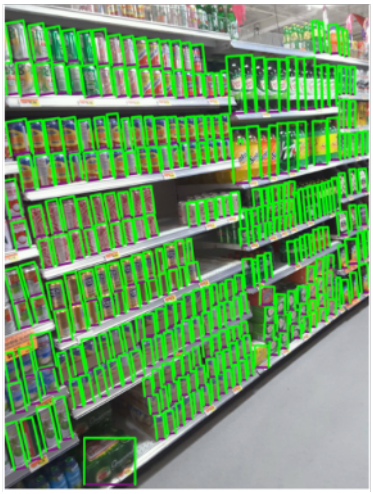
- Scene text

- 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 (cuda version < 11):

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

Note: For 30xx series graphics cards (cuda version >= 11), 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

If docker is not available, we provide detailed steps to install the requirements by `pip` (cuda version < 11):

```
pip install -r requirements.txt
pip install -v -e .  # or "python setup.py develop"
```

Or, you can simply install AlphaRotate with the following commands (cuda version < 11):

```
pip install alpharotate  # Not suitable for dev.
```

Note: For 30xx series graphics cards (cuda version >= 11), we recommend this blog to install tf1.xx

For 30xx series graphics cards (cuda version >= 11):

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

cd alpharotate/libs/utils/
rm *.so
rm *.c
```

(continues on next page)

(continued from previous page)

```
rm *.cpp
python setup.py build_ext --inplace
```

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)
```

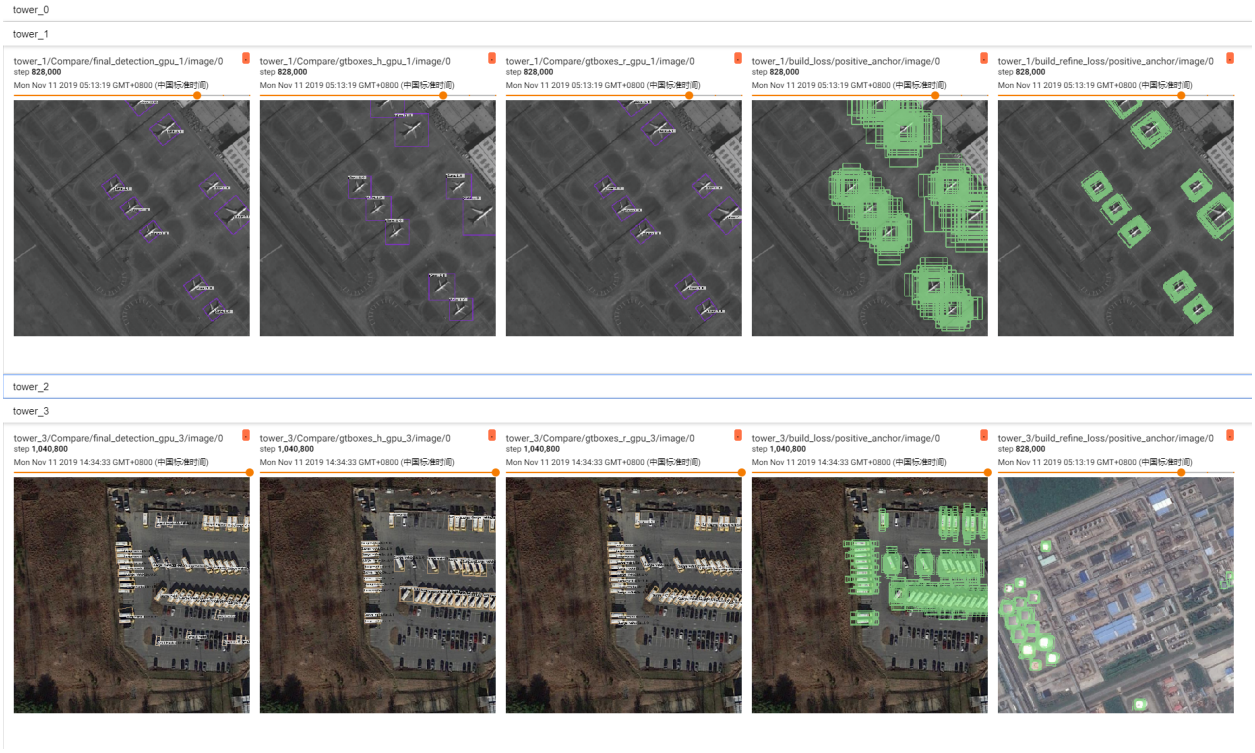
Note: In order to set the breakpoint conveniently, the read and write mode of the file is' a+'. If the model of the same `#VERSION` needs to be tested again, the original test results need to be deleted.

- 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)
```

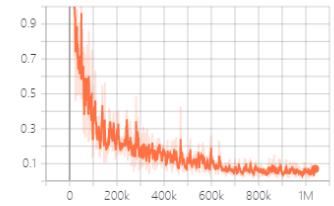
- Tensorboard

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



cls

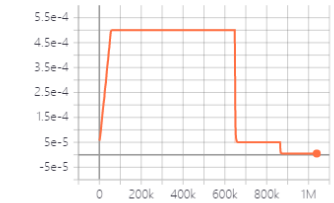
cls_loss
tag: cls/cls_loss



get_batch

lr

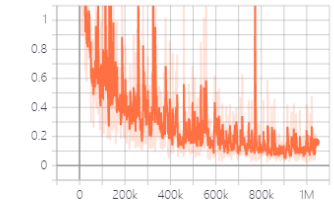
lr



refine

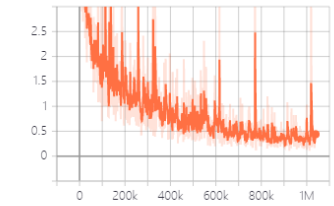
reg

reg_loss
tag: reg/reg_loss



total

total_losses
tag: total/total_losses



CHAPTER

FOUR

API REFERENCE

4.1 alpharotate.utils

4.1.1 densely_coded_label

alpharotate.utils.densely_coded_label.**angle_label_decode**(*angle_encode_label*, *angle_range*, *omega=1.0*, *mode=0*)

Decode binary/gray label back to angle label

Parameters

- **angle_encode_label** – binary/gray label
- **angle_label** – angle label, range in [-90,0) or [-180, 0)
- **angle_range** – 90 or 180
- **mode** – 0: binary label, 1: gray label

Returns angle label

alpharotate.utils.densely_coded_label.**angle_label_encode**(*angle_label*, *angle_range*, *omega=1.0*, *mode=0*)

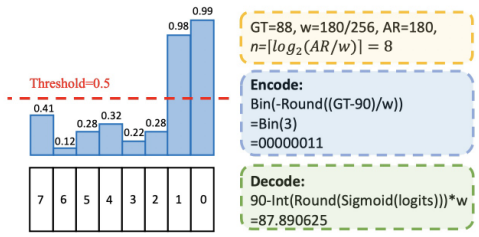
Encode angle label as binary/gray label

Parameters

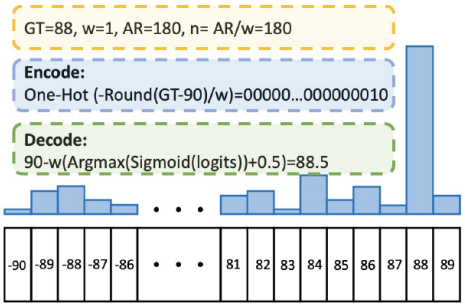
- **angle_label** – angle label, range in [-90,0) or [-180, 0)
- **angle_range** – 90 or 180
- **omega** – angle discretization granularity
- **mode** – 0: binary label, 1: gray label

Returns binary/gray label

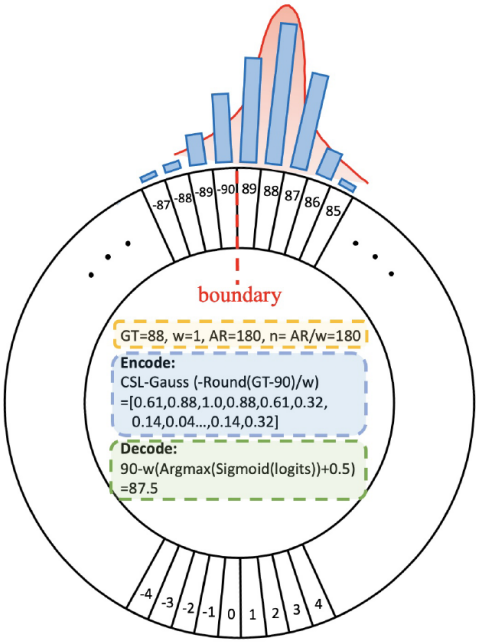
Dense Coded Label: Proposed by "Xue Yang et al. Dense Label Encoding for Boundary Discontinuity Free Rotation Detection. CVPR 2021."



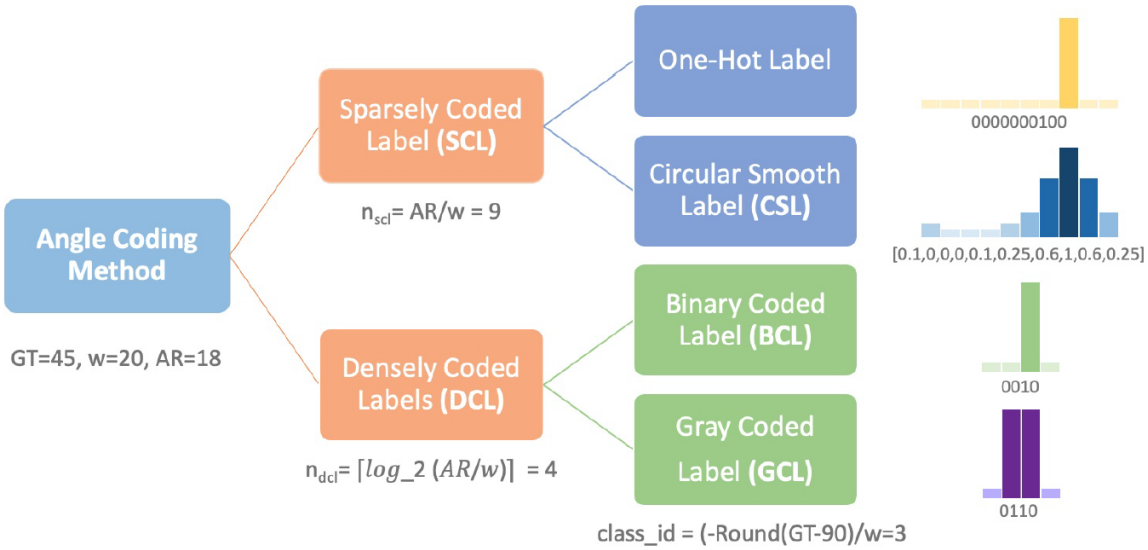
(a) DCL: Binary Coded Label



(b) SCL: One-Hot Label



(c) SCL: Circular Smooth Label



alpharotate.utils.densely_coded_label.**binary_label_decode**(*binary_label*, *angle_range*, *omega=1.0*)

Decode binary label back to angle label

Parameters

- **binary_label** – binary label
- **angle_range** – 90 or 180
- **omega** – angle discretization granularity

Returns angle label

alpharotate.utils.densely_coded_label.**binary_label_encode**(*angle_label*, *angle_range*, *omega=1.0*)
Encode angle label as binary label

Parameters

- **angle_label** – angle label, range in [-90,0) or [-180, 0)
- **angle_range** – 90 or 180
- **omega** – angle discretization granularity

Returns binary label

alpharotate.utils.densely_coded_label.**get_all_binary_label**(*num_label*, *class_range*)
Get all binary label according to num_label

Parameters

- **num_label** – angle_range/omega, 90/omega or 180/omega
- **class_range** – angle_range/omega, 90/omega or 180/omega

Returns all binary label

alpharotate.utils.densely_coded_label.**get_all_gray_label**(*angle_range*)
Get all gray label

Parameters **angle_range** – 90/omega or 180/omega

Returns all gray label

alpharotate.utils.densely_coded_label.**get_code_len**(*class_range*, *mode=0*)
Get encode length

Parameters

- **class_range** – angle_range/omega
- **mode** – 0: binary label, 1: gray label

Returns encode length

alpharotate.utils.densely_coded_label.**gray_label_decode**(*gray_label*, *angle_range*, *omega=1.0*)
Decode gray label back to angle label

Parameters

- **gray_label** – gray label
- **angle_range** – 90 or 180
- **omega** – angle discretization granularity

Returns angle label

alpharotate.utils.densely_coded_label.**gray_label_encode**(*angle_label*, *angle_range*, *omega=1.0*)
Encode angle label as gray label

Parameters

- **angle_label** – angle label, range in [-90,0) or [-180, 0)
- **angle_range** – 90 or 180
- **omega** – angle discretization granularity

Returns gray label

4.1.2 smooth_label

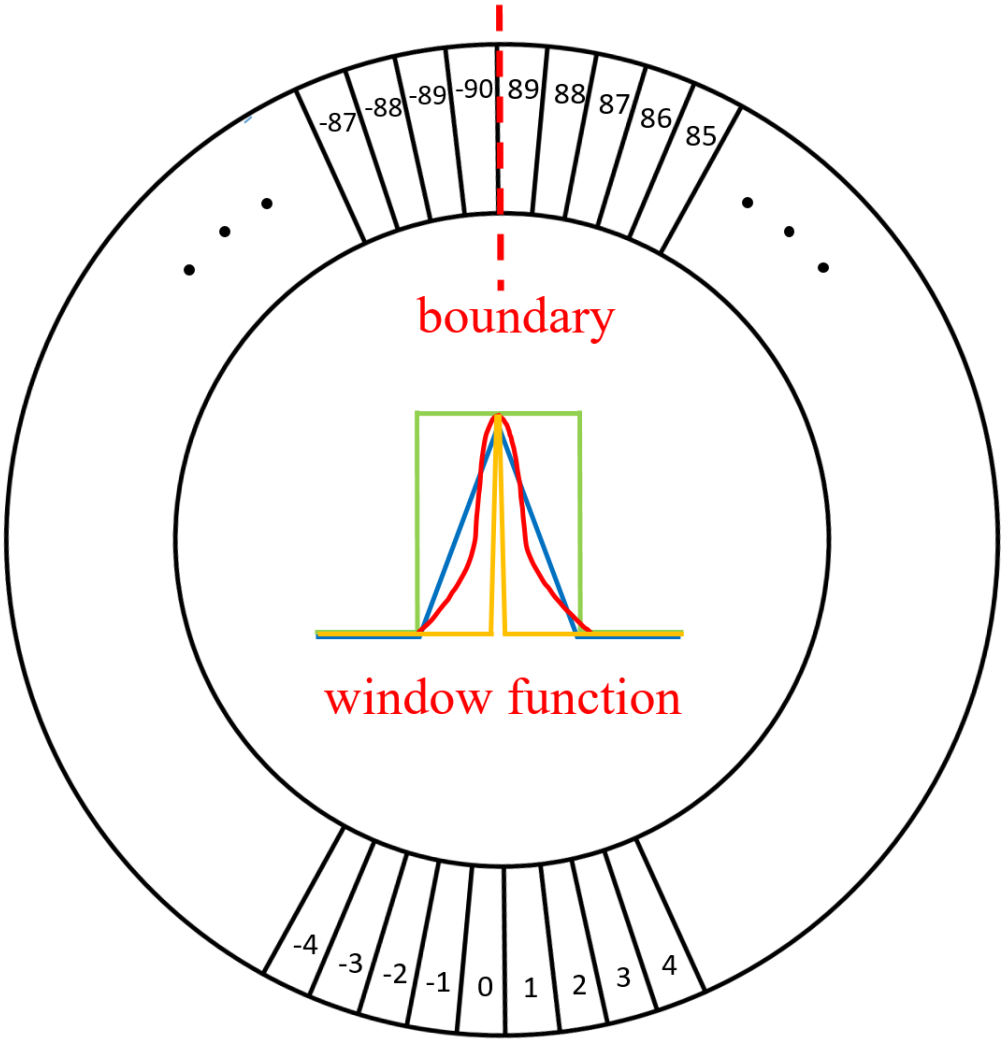
alpharotate.utils.smooth_label.**angle_smooth_label**(*angle_label*, *angle_range=90*, *label_type=0*, *radius=4*, *omega=1*)

Parameters

- **angle_label** – angle label, range in [-90,0) or [-180, 0)
- **angle_range** – 90 or 180
- **label_type** – 0: gaussian label, 1: rectangular label, 2: pulse label, 3: triangle label
- **radius** – window radius
- **omega** – angle discretization granularity

Returns

Circular Smooth Label: Proposed by "Xue Yang et al. Arbitrary-Oriented Object Detection with Circular Smooth Label. ECCV 2020."



alpharotate.utils.smooth_label.**gaussian_label**(*label*, *num_class*, *u=0*, *sig=4.0*)
Get gaussian label

Parameters

- **label** – angle_label/omega
- **num_class** – angle_range/omega
- **u** – mean
- **sig** – window radius

Returns gaussian label

alpharotate.utils.smooth_label.**pulse_label**(*label*, *num_class*)
Get pulse label

Parameters

- **label** – angle_label/omega
- **num_class** – angle_range/omega

Returns pulse label

alpharotate.utils.smooth_label.**rectangular_label**(*label*, *num_class*, *radius=4*)
Get rectangular label

Parameters

- **label** – angle_label/omega
- **num_class** – angle_range/omega
- **radius** – window radius

Returns rectangular label

alpharotate.utils.smooth_label.**triangle_label**(*label*, *num_class*, *radius=4*)
Get triangle label

Parameters

- **label** – angle_label/omega
- **num_class** – angle_range/omega
- **radius** – window radius

Returns triangle label

4.1.3 gaussian_metric

alpharotate.utils.gaussian_metric.**box2gaussian**(*boxes1*, *boxes2*)
Convert box $(x, y, w, h, heta)$ to Gaussian distribution $(\mu, \mathbf{\Sigma})$

Parameters

- **boxes1** – $(x_1, y_1, w_1, h_1, heta_1)$, [-1, 5]
- **boxes2** – $(x_2, y_2, w_2, h_2, heta_2)$, [-1, 5]

Returns $(\mu, \mathbf{\Sigma})$

alpharotate.utils.gaussian_metric.**gaussian_kullback_leibler_divergence**(*boxes1*, *boxes2*)
Calculate the kullback-leibler divergence between boxes1 and boxes2

Parameters

- **boxes1** – $(x_1, y_1, w_1, h_1, heta_1)$, shape: [-1, 5]
- **boxes2** – $(x_2, y_2, w_2, h_2, heta_2)$, shape: [-1, 5]

Returns kullback-leibler divergence, $\mathbf{D}_{kl}$

alpharotate.utils.gaussian_metric.**gaussian_wasserstein_distance**(*boxes1*, *boxes2*)

Calculate the wasserstein distance between boxes1 and boxes2: $\mathbf{D}_w = ||\mu_1 - \mu_2||_2^2 + \mathbf{T}r(\mathbf{\Sigma}_1 + \mathbf{\Sigma}_2 - 2(\mathbf{\Sigma}_1^{1/2}\mathbf{\Sigma}_2\mathbf{\Sigma}_1^{1/2})^{1/2})$

Parameters

- **boxes1** – $(x_1, y_1, w_1, h_1, heta_1)$, shape: [-1, 5]
- **boxes2** – $(x_2, y_2, w_2, h_2, heta_2)$, shape: [-1, 5]

Returns wasserstein distance, $\mathbf{D}_w$

alpharotate.utils.gaussian_metric.**kullback_leibler_divergence**(*mu1*, *mu2*, *mu1_T*, *mu2_T*, *sigma1*, *sigma2*)

Calculate the kullback-leibler divergence between two Gaussian distributions : $\mathbf{D}_{kl} = 0.5 * ((\mu_1 - \mu_2)^T\mathbf{\Sigma}_2^{1/2}(\mu_1 - \mu_2) + 0.5 * \mathbf{T}r(\mathbf{\Sigma}_2^{-1}\mathbf{\Sigma}_1) + 0.5 * \ln|\mathbf{\Sigma}_2|/|\mathbf{\Sigma}_1| - 1$

Parameters

- **mu1** – mean (μ_1) of the Gaussian distribution, shape: [-1, 1, 2]
- **mu2** – mean (μ_2) of the Gaussian distribution, shape: [-1, 1, 2]
- **mu1_T** – transposition of (μ_1), shape: [-1, 2, 1]
- **mu2_T** – transposition of (μ_2), shape: [-1, 2, 1]
- **sigma1** – covariance (Σ_1) of the Gaussian distribution, shape: [-1, 2, 2]
- **sigma2** – covariance (Σ_1) of the Gaussian distribution, shape: [-1, 2, 2]

Returns kullback-leibler divergence, $\mathbf{D}_{kl}$

alpharotate.utils.gaussian_metric.**wasserstein_distance_item2**(*sigma1*, *sigma2*)

Calculate the second term of wasserstein distance: $\mathbf{T}r(\mathbf{\Sigma}_1 + \mathbf{\Sigma}_2 - 2(\mathbf{\Sigma}_1^{1/2}\mathbf{\Sigma}_2\mathbf{\Sigma}_1^{1/2})^{1/2})$

Parameters

- **sigma1** – covariance (Σ_1) of the Gaussian distribution, shape: [-1, 2, 2]
- **sigma2** – covariance (Σ_1) of the Gaussian distribution, shape: [-1, 2, 2]

Returns the second term of wasserstein distance

PYTHON MODULE INDEX

a

INDEX