VL Benchmark Documentation

Release 0.1

Xu Zhang

Mar 03, 2019

INTRODUCTION

CHAPTER ONE

ABOUT THIS PROJECT

A framework for local feature evaluation. Reimplementation of the VLBenchmarks project.

MATLAB implementation: Karel Lenc

Python implementation: Xu Zhang

1.1 Python Interface

1.1.1 Requirement

We recommend to use conda to install all the requirements all at once.

```
conda env create -f ./python/conda/environment.yml
```

1.1.2 Test the code

Test repeatability benchmark

```
>>> python ./python/test/test_rep_bench.py
```

Test matching score benchmark

```
>>> python ./python/test/test_ms_bench.py
```

Test image retrieval benchmark

```
>>> python ./python/test/test_retrieval_bench.py
```

Test wide baseline matching benchmark

```
>>> python ./python/test/test_W1BS_Bench.py
```

Test feature extraction

```
>>> python ./python/test/test_feature_extraction.py
```

Test draw feature

```
>>> python ./python/test/draw_frame.py
```

CHAPTER

TWO

TEST FEATURE AND DESCRIPOR MATCHING BENCHMARK

This is how to run the feature matching (repeatability) and descriptor matching (matching score) benchmark.

Repeatablity Benchmark:

```
# Define retrieval benchmark
rep_bench = bench.repBench.repBench()

# Define feature
vlsift_py = features.cyvlsift_official.cyvlsift_official()

# Define dataset
vggh = dset.vgg_dataset.vggh_Dataset()

# Do the evaluation
rep_result_py = rep_bench.evaluate(
    vggh, vlsift_py, use_cache=False, save_result=True)

# Make the results from different detectors as a list.
# (Only one here, but you can add more)
rep_result = [rep_result_py]

# Show the result
for result_term in rep_result[0]['result_term_list']:
    bench.Utils.print_result(rep_result, result_term)
    bench.Utils.save_result(rep_result, result_term)

#Show result for different sequences
for sequence in vggh.sequence_name_list:
    for result_term in rep_result[0]['result_term_list']:
        bench.Utils.print_sequence_result(rep_result, sequence, result_term)
        bench.Utils.save_sequence_result(rep_result, sequence, result_term)
```

Matching score Benchmark:

```
# Define matching score benchmark
ms_bench = bench.MatchingScoreBench.MatchingScoreBench()

# Define feature 1
vlsift_py = features.cyvlsift_official.cyvlsift_official()

# Define feature 2
vlsift_load_matlab = features.vlsift_load_matlab.vlsift_load_matlab()

# Define dataset
```

(continues on next page)

(continued from previous page)

```python
vggh = dset.vgg_dataset.vggh_Dataset()

# Do the evaluation
ms_result_py = ms_bench.evaluate(
    vggh, vlsift_py, use_cache=True, save_result=True)

ms_result_matlab = ms_bench.evaluate(
    vggh, vlsift_load_matlab, use_cache=True, save_result=True)

# Make the results from different detectors as a list.
ms_result = [ms_result_py, ms_result_matlab]

# Show the result
for result_term in ms_result[0]['result_term_list']:
    bench.Utils.print_result(ms_result, result_term)
    bench.Utils.save_result(ms_result, result_term)

#show result for different sequences
for sequence in vggh.sequence_name_list:
    for result_term in ms_result[0]['result_term_list']:
        bench.Utils.print_sequence_result(ms_result, sequence, result_term)
        bench.Utils.save_sequence_result(ms_result, sequence, result_term)
```

Full code for Repeatablity Benchmark (test/test_retrieval_bench.py):

```python
#!/usr/bin/python
# -*- coding: utf-8 -*-
# ===========================================================
#  File Name: test_rep_bench.py
#  Author: Xu Zhang, Columbia University
#  Creation Date: 01-25-2019
#  Last Modified: Sun Mar  3 18:11:21 2019
#
#  Usage: python test_rep_bench.py
#  Description:test repeatability benchmark
#
#  Copyright (C) 2018 Xu Zhang
#  All rights reserved.
#
#  This file is made available under
#  the terms of the BSD license (see the COPYING file).
# ===========================================================

import os
import sys
cwd = os.getcwd()
sys.path.insert(0, '{}/python/'.format(cwd))

import bench.Utils
import bench.repBench
import features.cyvlsift_official
import dset.vgg_dataset


if __name__ == "__main__":

    # Define repeatability benchmark
```

(continues on next page)

(continued from previous page)

```
    rep_bench = bench.repBench.repBench()

    # Define feature
    vlsift_py = features.cyvlsift_official.cyvlsift_official()

    # Define dataset
    vggh = dset.vgg_dataset.vggh_Dataset()

    # Do the evaluation
    rep_result_py = rep_bench.evaluate(
        vggh, vlsift_py, use_cache=False, save_result=True)

    # Make the results from different detectors as a list.
    # (Only one here, but you can add more)
    rep_result = [rep_result_py]

    # Show the result
    for result_term in rep_result[0]['result_term_list']:
        bench.Utils.print_result(rep_result, result_term)
        bench.Utils.save_result(rep_result, result_term)

    #Show result for different sequences
    for sequence in vggh.sequence_name_list:
        for result_term in rep_result[0]['result_term_list']:
            bench.Utils.print_sequence_result(rep_result, sequence, result_term)
            bench.Utils.save_sequence_result(rep_result, sequence, result_term)
```

Full code for Matching Score Benchmark (test/test_ms_bench.py):

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# ===========================================================
#  File Name: test_ms_bench.py
#  Author: Xu Zhang, Columbia University
#  Creation Date: 01-25-2019
#  Last Modified: Sun Mar  3 18:09:44 2019
#
#  Usage: python test_ms_bench.py
#  Description:test matching score benchmark
#
#  Copyright (C) 2018 Xu Zhang
#  All rights reserved.
#
#  This file is made available under
#  the terms of the BSD license (see the COPYING file).
# ===========================================================

import os
import sys
cwd = os.getcwd()
sys.path.insert(0, '{}/python/'.format(cwd))

import bench.Utils
import bench.MatchingScoreBench
import bench.repBench
import features.cyvlsift_official
import features.vlsift_load_matlab
```

(continues on next page)

(continued from previous page)

```python
import dset.vgg_dataset


if __name__ == "__main__":

    # Define matching score benchmark
    ms_bench = bench.MatchingScoreBench.MatchingScoreBench()

    # Define feature 1
    vlsift_py = features.cyvlsift_official.cyvlsift_official()

    # Define feature 2
    vlsift_load_matlab = features.vlsift_load_matlab.vlsift_load_matlab()

    # Define dataset
    vggh = dset.vgg_dataset.vggh_Dataset()

    # Do the evaluation
    ms_result_py = ms_bench.evaluate(
        vggh, vlsift_py, use_cache=True, save_result=True)

    ms_result_matlab = ms_bench.evaluate(
        vggh, vlsift_load_matlab, use_cache=True, save_result=True)

    # Make the results from different detectors as a list.
    ms_result = [ms_result_py, ms_result_matlab]

    # Show the result
    for result_term in ms_result[0]['result_term_list']:
        bench.Utils.print_result(ms_result, result_term)
        bench.Utils.save_result(ms_result, result_term)

    #show result for different sequences
    for sequence in vggh.sequence_name_list:
        for result_term in ms_result[0]['result_term_list']:
            bench.Utils.print_sequence_result(ms_result, sequence, result_term)
            bench.Utils.save_sequence_result(ms_result, sequence, result_term)
```

CHAPTER

THREE

TEST RETRIEVAL BENCHMARK

This is how to run the retrieval benchmark

```
# Define retrieval benchmark
retrieval_bench = bench.RetrievalBenchmark.RetrievalBenchmark()

# Define feature
vlsift_py = features.cyvlsift_official.cyvlsift_official()

# Define dataset
paris6k = dset.paris6k_dataset.paris6k_Dataset()

# Do the evaluation
map_result_py = retrieval_bench.evaluate(
    paris6k, vlsift_py, use_cache=True, save_result=True)

# Make the results from different detectors as a list.
# (Only one here, but you can add more)
map_result = [map_result_py]

# Show the result
for result_term in map_result[0]['result_term_list']:
    bench.Utils.print_retrieval_result(map_result, 'm' + result_term)
    bench.Utils.save_retrieval_result(map_result, 'm' + result_term)
```

Full code (test/test_retrieval_bench.py):

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# ===========================================================
#  File Name: test_retrieval_bench.py
#  Author: Xu Zhang, Columbia University
#  Creation Date: 01-25-2019
#  Last Modified: Sun Mar  3 17:58:13 2019
#
#  Usage: python test_retrieval_bench.py
#  Description: Test retrieval benchmark
#
#  Copyright (C) 2018 Xu Zhang
#  All rights reserved.
#
#  This file is made available under
#  the terms of the BSD license (see the COPYING file).
# ===========================================================
```

(continues on next page)

(continued from previous page)

```python
import os
import sys

cwd = os.getcwd()
sys.path.insert(0, '{}/python/'.format(cwd))

import bench.RetrievalBenchmark
import features.cyvlsift_official
import dset.oxford5k_dataset
import dset.paris6k_dataset
import bench.Utils

if __name__ == "__main__":

    # Define retrieval benchmark
    retrieval_bench = bench.RetrievalBenchmark.RetrievalBenchmark()

    # Define feature
    vlsift_py = features.cyvlsift_official.cyvlsift_official()

    # Define dataset
    paris6k = dset.paris6k_dataset.paris6k_Dataset()

    # Do the test
    map_result_py = retrieval_bench.evaluate(
        paris6k, vlsift_py, use_cache=True, save_result=True)

    # Make the results from different detectors as a list.
    # (Only one here, but you can add more)
    map_result = [map_result_py]

    # Show the result
    for result_term in map_result[0]['result_term_list']:
        bench.Utils.print_retrieval_result(map_result, 'm' + result_term)
        bench.Utils.save_retrieval_result(map_result, 'm' + result_term)


    # Another dataset
    oxford5k = dset.oxford5k_dataset.oxford5k_Dataset()
    map_result_py = retrieval_bench.evaluate(
        oxford5k, vlsift_py, use_cache=True, save_result=True)
    map_result = [map_result_py]
    for result_term in map_result[0]['result_term_list']:
        Utils.print_retrieval_result(map_result, 'm' + result_term)
        Utils.save_retrieval_result(map_result, 'm' + result_term)
```

CHAPTER

FOUR

TEST BASELINE BENCHMARK (W1BS)

This is how to run the baseline matching benchmark.

```
# Define baseline benchmark
bench = bench.W1BSBench.W1BSBench()

# Define feature
np_sift_py = features.np_sift.np_sift()

# Define dataset
w1bs = dset.W1BS_dataset.W1BS_Dataset()

# Do the evaluation
result_py = bench.evaluate(w1bs, np_sift_py, use_cache=True, save_result=True)

# Make the results from different detectors as a list.
result_list = [result_py]

# Show the result
bench.Utils.print_result(result_list, 'ap')
```

Full code (test/test_W1BS_Bench.py):

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# ===========================================================
#  File Name: test_W1BS_Bench.py
#  Author: Xu Zhang, Columbia University
#  Creation Date: 01-25-2019
#  Last Modified: Sun Mar  3 22:43:21 2019
#
#  Usage: python test_W1BS_Bench.py
#  Description: Test baseline matching benchmark
#
#  Copyright (C) 2018 Xu Zhang
#  All rights reserved.
#
#  This file is made available under
#  the terms of the BSD license (see the COPYING file).
# ===========================================================

import sys
import os

cwd = os.getcwd()
```

(continues on next page)

(continued from previous page)

```
sys.path.insert(0, '{}/python/'.format(cwd))

import bench.Utils
import bench.W1BSBench
import features.np_sift
import dset.W1BS_dataset


if __name__ == "__main__":

    # Define baseline benchmark
    bench = bench.W1BSBench.W1BSBench()

    # Define feature
    np_sift_py = features.np_sift.np_sift()

    # Define dataset
    w1bs = dset.W1BS_dataset.W1BS_Dataset()

    # Do the evaluation
    result_py = bench.evaluate(w1bs, np_sift_py, use_cache=True, save_result=True)

    # Make the results from different detectors as a list.
    result_list = [result_py]

    # Show the result
    bench.Utils.print_result(result_list, 'ap')
```

CHAPTER

FIVE

HOW TO CREATE NEW FEATURE

To test your own detector or descriptor, you need to create your own feature detector or descriptor.

Any detector or descriptor should be a subclass of *features.DetectorDescriptorTemplate.DetectorAndDescriptor*. Create the py file under features/.

To make it work, you should set proper properties and implement corresponding function of feature detection or/and descriptor extraction. Just implement what your module. For example, if the module is a detector only, just implement detect_feature().

Here is an example (features/cyvlsift_official.py):

features.DetectorDescriptorTemplate.DetectorDescriptorBundle helps bundle a feature detecor and a feature extractor. For example, you can bundle the DOG detector with your own descriptor and vice versa.

Full code:

```
"""
This module is a warpper for cyvlsift
"""

import numpy as np
import cv2
import cyvlfeat
import features.feature_utils
from features.DetectorDescriptorTemplate import DetectorAndDescriptor


class cyvlsift_official(DetectorAndDescriptor):
    """A warpper for cyvlsift.

    Attributes
    ----------

    peak_thresh: float
        Peak threshold for feature detector

    """
    def __init__(self, peak_thresh=0.0):
        super(
            cyvlsift_official,
            self).__init__(
            name='cyvlsift_official',
            is_detector=True,
            is_descriptor=True,
```

(continues on next page)

(continued from previous page)

```
            is_both=True)
        self.peak_thresh = peak_thresh

    def detect_feature(self, image):
        """
        Extract feature from image.

        :param image: The image
        :type image: array
        :returns: feature
        :rtype: array(n*d)
        """
        new_image = image.astype(np.float32)
        new_image = new_image/255.0
        new_image = feature_utils.all_to_gray(new_image)
        feature = cyvlfeat.sift.sift(
            new_image, peak_thresh=self.peak_thresh, magnification=5.0)
        return feature

    def extract_descriptor(self, image, feature):
        """
        Extract descriptor from image with feature.

        :param image: The image
        :type image: array
        :param feature: The feature output by detector
        :type feature: array
        :returns: descriptor
        :rtype: array(n*d)
        """
        new_image = image.astype(np.float32)
        new_image = new_image/255.0
        new_image = feature_utils.all_to_gray(new_image)
        feature, descriptor = cyvlfeat.sift.sift(
            new_image, peak_thresh=self.peak_thresh, frames=feature, magnification=5.
↪0, compute_descriptor=True)
        return descriptor

    def extract_all(self, image):
        """
        Extract feature and descriptor from image.

        :param image: The image
        :type image: array
        :returns: feature, descriptor
        :rtype: array(n*d)
        """

        new_image = image.astype(np.float32)
        new_image = new_image/255.0
        new_image = feature_utils.all_to_gray(new_image)
        feature, descriptor_vector = cyvlfeat.sift.sift(
            new_image, peak_thresh=self.peak_thresh, magnification=5.0, compute_
↪descriptor=True)
        return feature, descriptor_vector
```

CHAPTER

SIX

FEATURES MODULE

This module defines basic feature detector and feature descriptor.

This module describe basic detector and descriptor

class `features.DetectorDescriptorTemplate.`**`DetectorAndDescriptor`**(*name*, *is_detector=False*, *is_descriptor=False*, *is_both=True*, *csv_flag=False*, *patch_input=False*)

Bases: `object`

Basic template class for detector and descriptor.

Attributes

name: str Name of the detector

is_detector: boolean, optional Is the module is a detector or not

is_descriptor: boolean, optional Is the module is a descriptor or not

is_both: boolean, optional Is the module is both a detector and a decritpor or not

csv_flag: boolean, optional Can the module load feature from csv file or not

patch_input: boolean, optional Do the module take patch instead of full image as input or not

`detect_feature`(*image*)

Extract feature from image.

Parameters **`image`** (`array`) – The image

Returns feature

Return type array(n*d)

`extract_all`(*image*)

Extract feature and descriptor from image.

Parameters **`image`** (`array`) – The image

Returns feature, descriptor

Return type array(n*d)

`extract_descriptor`(*image*, *feature*)

Extract descriptor from image with feature.

Parameters

- **`image`** (`array`) – The image

- **feature** (*array*) – The feature output by detector

Returns descriptor

Return type array(n*d)

class features.DetectorDescriptorTemplate.**DetectorDescriptorBundle**(*detector*, *descriptor*)

Bases: *features.DetectorDescriptorTemplate.DetectorAndDescriptor*

Combine a detector and a descriptor to make a new detector+descriptor. For paper only focuses on either detector or descriptor.

Attributes

name: str Name of the Bundle

detector: DetectorAndDescriptor The detector to combine

descriptor: DetectorAndDescriptor The descriptor to combine

detect_feature(*image*)

Extract feature from image.

Parameters **image** (*array*) – The image

Returns feature

Return type array(n*d)

extract_all(*image*)

Extract feature and descriptor from image.

Parameters **image** (*array*) – The image

Returns feature, descriptor

Return type array(n*d)

extract_descriptor(*image*, *feature*)

Extract descriptor from image with feature.

Parameters

- **image** (*array*) – The image
- **feature** (*array*) – The feature output by detector

Returns descriptor

Return type array(n*d)

This module is a warpper for cyvlsift

features.feature_utils.**all_to_BGR**(*image*)

Convert image to 3-channel image.

Parameters **image** (*array*) – The image

Returns color_image

Return type array(w*h*3)

features.feature_utils.**all_to_gray**(*image*)

Convert image to gray image (Matlab coeffients).

Parameters **image** (*array*) – The image

Returns gray_image

Return type array(w*h)

`features.feature_utils.`**`all_to_gray_cv`**(*image*)

Convert image to gray image (opencv coeffients).

Parameters **`image`** (*`array`*) – The image

Returns gray_image

Return type array(w*h)

`features.feature_utils.`**`extract_patch`**(*img*, *kp*, *patch_sz=32*, *rectify_flag=False*)

Extract an rectified patch from image with information in the keypoint.

Parameters

- **`img`** (*`array`*) – The image
- **`kp`** (*`array`*) – The key point
- **`patch_sz`** (*`int`*) – patch size
- **`rectify_flag`** (*`boolean`*) – rectified or not

Returns patch

Return type array(w*h)

`features.feature_utils.`**`rectify_patch`**(*img*, *kp*, *patch_sz=32*)

Extract an rectified patch from image with information in the keypoint.

Parameters

- **`img`** (*`array`*) – The image
- **`kp`** (*`array`*) – The key point
- **`patch_sz`** (*`int`*) – patch size

Returns patch

Return type array(w*h)

`features.feature_utils.`**`rgb2gray`**(*img*)

Convert bgr image to gray image (Matlab coeffients).

Parameters **`img`** (*`array`*) – The image

Returns img_gray

Return type array(n*d)

Here is an example of how to make a warpper for cyvlsift

This module is a warpper for cyvlsift

`class` `features.cyvlsift_official.`**`cyvlsift_official`**(*peak_thresh=0.0*)

Bases: *`features.DetectorDescriptorTemplate.DetectorAndDescriptor`*

A warpper for cyvlsift.

Attributes

peak_thresh: float Peak threshold for feature detector

`detect_feature`(*image*)

Extract feature from image.

Parameters **`image`** (*`array`*) – The image

Returns feature

Return type array(n*d)

`extract_all`(*image*)

Extract feature and descriptor from image.

Parameters **`image`** (*`array`*) – The image

Returns feature, descriptor

Return type array(n*d)

`extract_descriptor`(*image*, *feature*)

Extract descriptor from image with feature.

Parameters

- **`image`** (*`array`*) – The image
- **`feature`** (*`array`*) – The feature output by detector

Returns descriptor

Return type array(n*d)

CHAPTER

SEVEN

DSET MODULE

This module defines the structure of different datasets

This module describe dataset template

class dset.dataset.**Image**

Bases: `object`

Image data structure.

Attributes

id: str ID of the image

image_data: array Image data

label: str Description for the label

filename: str Name of the file

filename = ''

idx = ''

image_data = None

label = ''

class dset.dataset.**Link**

Bases: `object`

Link data structure. Describe an image pair, it's useful for matching dataset.

Attributes

source: str ID of the source image

target: str ID of the target image

filename: str filename of the transformation matrix

transform_matrix: array Transform Matrix of the image pair

task: dict Task information

filename = ''

source = ''

target = ''

task = {}

transform_matrix = None

class `dset.dataset.`**`Sequence`**

Bases: `object`

Sequence for a list of images and links.

Attributes

name: str Name of the sequence

description: str Description of the sequence

image_id_list: list List of image id (for keep the order of the images)

image_dict: dict Dict for image data

link_id_list: list List of link id (for keep the order of the links)

link_dict: dict link_dict: Dict for all links in the sequence

`description = ''`

`image_dict = None`

`image_id_list = None`

`images``()`

Return images in the sequence.

Returns images

Return type list

`link_dict = None`

`link_id_list = None`

`links``()`

Return links in the sequence.

Returns links

Return type list

`name = ''`

class `dset.dataset.`**`SequenceDataset`**`(`*name*, *root_dir='./datasets/'*, *download_flag=False*`)`

Bases: `object`

Sequence dataset for image matching test

Attributes

name: str Name of the dataset

root_dir: str Directory for the data

download_flag: boolean

Download data or not. Keep it False, unless you need to update the dataset. Data will automatically download, if there is no data in the root_dir.

`download``()`

Download data

`get_image``(`*sequence_name*, *image_id*`)`

Get a image by sequence name and image ID.

Parameters

- **sequence_name** (*str*) – Name of the sequence
- **image_id** (*str*) – Image ID

Returns image

Return type *Image*

get_link(*sequence_name*, *link_id*)
Get a link by sequence name and link ID.

Parameters

- **sequence_name** (*str*) – Name of the sequence
- **link_id** (*str*) – Link ID

Returns link

Return type *Link*

get_sequence(*sequence_name*)
Get a sequence by name.

Parameters **sequence_name** (*str*) – Name of the sequence

Returns sequence

Return type *Sequence*

get_task(*sequence_name*, *link_id*)
Get a task by sequence name and link ID.

Parameters

- **sequence_name** (*str*) – Name of the sequence
- **link_id** (*str*) – Link ID

Returns task

Return type dict

load_dataset_info()
Load data from hard disk

read_image_data()
Read image data

read_image_data_vggh()
Load image data from vggh like dataset

read_link_data()
Read Link data

read_link_data_vggh()
Load link data from vggh like dataset

set_task()
Deprecated

This module describe dataset template for image retrieval task

class dset.retrieval_dataset.**RetrievalDataset**(*name*, *root_dir='./datasets/'*, *download_flag=False*)
Bases: `object`

Sequence dataset for image retrieval

Attributes

name: str Name of the dataset

root_dir: str Directory for the data

download_flag: boolean

Download data or not. Keep it False, unless you need to update the dataset. Data will automatically download, if there is no data in the root_dir.

`download`()
Download data

`load_dataset_info`()
Load data from hard disk

`read_gallery_list`()
Load gallery image list

`read_query_list`()
Load query image list

`class` `dset.vgg_dataset.`**`vggh_Dataset`**(*root_dir='./datasets/'*, *download_flag=False*)
Bases: *`dset.dataset.SequenceDataset`*

Oxford image matching

`download`()
Download data

`get_image`(*sequence_name*, *image_id*)
Get a image by sequence name and image ID.

Parameters

- **`sequence_name`** (*`str`*) – Name of the sequence
- **`image_id`** (*`str`*) – Image ID

Returns image

Return type *Image*

`get_link`(*sequence_name*, *link_id*)
Get a link by sequence name and link ID.

Parameters

- **`sequence_name`** (*`str`*) – Name of the sequence
- **`link_id`** (*`str`*) – Link ID

Returns link

Return type *Link*

`get_sequence`(*sequence_name*)
Get a sequence by name.

Parameters **`sequence_name`** (*`str`*) – Name of the sequence

Returns sequence

Return type *Sequence*

`get_task`(*sequence_name*, *link_id*)
Get a task by sequence name and link ID.

Parameters

- **`sequence_name`** (*str*) – Name of the sequence
- **`link_id`** (*str*) – Link ID

Returns task

Return type dict

`load_dataset_info`()
Load data from hard disk

`read_image_data`()
Load image data

`read_image_data_vggh`()
Load image data from vggh like dataset

`read_link_data`()
Load link data

`read_link_data_vggh`()
Load link data from vggh like dataset

`set_task`()
Deprecated

class `dset.oxford5k_dataset.`**`oxford5k_Dataset`**(*root_dir='./datasets/'*, *download_flag=False*)
Bases: *`dset.retrieval_dataset.RetrievalDataset`*

Oxford 5K dataset

`download`()
Download data

`load_dataset_info`()
Load data from hard disk

`read_gallery_list`()
Load gallery image list

`read_query_list`()
Load query image list

class `dset.paris6k_dataset.`**`paris6k_Dataset`**(*root_dir='./datasets/'*, *download_flag=False*)
Bases: *`dset.retrieval_dataset.RetrievalDataset`*

Paris 6K dataset

`download`()
Download data

`load_dataset_info`()
Load data from hard disk

`read_gallery_list`()
Load gallery image list

`read_query_list`()
Load query image list

class `dset.W1BS_dataset.`**`W1BS_Dataset`**(*root_dir='./datasets/'*, *download_flag=False*)
Bases: *`dset.dataset.SequenceDataset`*

W1BS dataset for baseline matching

download()
Download data

get_image(*sequence_name*, *image_id*)
Get a image by sequence name and image ID.

Parameters

- **sequence_name** (*str*) – Name of the sequence
- **image_id** (*str*) – Image ID

Returns image

Return type *Image*

get_link(*sequence_name*, *link_id*)
Get a link by sequence name and link ID.

Parameters

- **sequence_name** (*str*) – Name of the sequence
- **link_id** (*str*) – Link ID

Returns link

Return type *Link*

get_sequence(*sequence_name*)
Get a sequence by name.

Parameters **sequence_name** (*str*) – Name of the sequence

Returns sequence

Return type *Sequence*

get_task(*sequence_name*, *link_id*)
Get a task by sequence name and link ID.

Parameters

- **sequence_name** (*str*) – Name of the sequence
- **link_id** (*str*) – Link ID

Returns task

Return type dict

load_dataset_info()
Load data from hard disk

read_image_data()
Load image data

read_image_data_vggh()
Load image data from vggh like dataset

read_link_data()
Load link data

read_link_data_vggh()
Load link data from vggh like dataset

`set_task``()`
    Deprecated

CHAPTER

EIGHT

BENCH MODULE

This module defines the structure of different benchmarks

This module describe benchmark template. A benchmark is given a detector/descriptor and a dataset, the way of performing the evluation.

class `bench.BenchmarkTemplate.`**`Benchmark`**(*name*, *tmp_feature_dir='./data/features/'*, *result_dir='./python_scores/'*)

Bases: `object`

`detect_feature`(*dataset*, *detector*, *use_cache=True*, *save_feature=True*)

Extract feature from image.

Parameters

- **`dataset`** (`SequenceDataset`) – Dataset to extract the feature
- **`detector`** (`DetectorAndDescriptor`) – Detector used to extract the feature
- **`use_cache`** (*`boolean`*) – Load cached feature and result or not
- **`save_feature`** (*`boolean`*) – Save computated feature or not

Returns feature

Return type dict

`detect_feature_custom`(*dataset*, *detector*, *use_cache=False*, *save_feature=True*)

Customized feature extraction method. For special task.

Parameters

- **`dataset`** (`SequenceDataset`) – Dataset to extract the feature
- **`detector`** (`DetectorAndDescriptor`) – Detector used to extract the feature
- **`use_cache`** (*`boolean`*) – Load cached feature and result or not
- **`save_feature`** (*`boolean`*) – Save computated feature or not

Returns feature

Return type dict

See also:

`evaluate_warpper`, *`extract_descriptor_custom`*

`evaluate`(*dataset*, *detector*)

Main function to run the evaluation wrapper. It could be different for different evaluation

Parameters

- **`dataset`** (`SequenceDataset`) – Dataset to extract the feature

- **`detector`** (`DetectorAndDescriptor`) – Detector used to extract the feature

See also:

`evaluate_warpper`

`evaluate_unit` (*feature_1*, *feature_2*, *task*)
Single evaluation unit. Given two features, return the result. Different for different benchmark

Parameters

- **`feature_1`** (*`array`*) – Feature to run. It can be feature or descriptor.
- **`feature_2`** (*`array`*) – Feature to run. It can be feature or descriptor.
- **`task`** (*`dict`*) – What to run

See also:

`evaluate_warpper` How to run the unit.

`dset.dataset.Link` definition of task.

`evaluate_warpper` (*dataset*, *detector*, *result_list*, *extract_descriptor=False*, *use_cache=True*, *save_result=True*, *custom_extraction=False*)
Load descriptor from cached file. If failed, extract descriptor from image.

Structure of the result:

result['dataset_name']: name of the dataset

result['result_term_list']: list of metrics for evaluation

result['task_name']: name of the task

result['detector_name']: name of the dataset

result['sequence_result']: a list for result from each sequence

result['ave_{}']: average value for each metric over all sequences

Structure of the sequence result:

sequence_result['sequence_name']: name of the sequence

sequence_result[result_name]: list of list of metrics over each link

sequence_result['result_label_list']: label of each link in sequence_result (Same order)

sequence_result['result_link_id_list']: ID of each link in sequence_result (Same order)

Parameters

- **`dataset`** (`SequenceDataset`) – Dataset to extract the feature
- **`detector`** (`DetectorAndDescriptor`) – Detector used to extract the feature
- **`result_list`** (*`list`*) – Metric to calculate
- **`extract_descriptor`** (*`boolean`*) – Extract descriptor or not
- **`use_cache`** (*`boolean`*) – Load cached feature and result or not
- **`save_result`** (*`boolean`*) – Save result or not
- **`custom_extraction`** (*`boolean`*) – Use custom extraction method or not. See also and extract_descriptor_custom

Returns result

Return type dict

See also:

detect_feature_custom Extract feature with customized method (special evaluation).

extract_descriptor_custom Extract descriptor with customized (special evaluation).

extract_descriptor(*dataset*, *detector*, *use_cache=False*, *save_feature=True*)
Extract feature from image.

Parameters

- **dataset** (SequenceDataset) – Dataset to extract the descriptor
- **detector** (DetectorAndDescriptor) – Detector used to extract the descriptor
- **use_cache** (*boolean*) – Load cached feature and result or not
- **save_feature** (*boolean*) – Save computated feature or not

Returns feature, descriptor

Return type dict, dict

extract_descriptor_custom(*dataset*, *detector*, *use_cache=False*, *save_feature=True*)
Customized description extraction method. For special task.

Parameters

- **dataset** (SequenceDataset) – Dataset to extract the descriptor
- **detector** (DetectorAndDescriptor) – Detector used to extract the descriptor
- **use_cache** (*boolean*) – Load cached feature and result or not
- **save_feature** (*boolean*) – Save computated feature or not

Returns feature

Return type dict

See also:

evaluate_warpper, extract_feature_custom

load_csv_feature(*csv_feature_file*)
Load feature from csvfile.

Parameters **csv_feature_file** (*str*) – csv file to load feature

Returns feature

Return type array

load_descriptor(*dataset_name*, *sequence_name*, *image*, *detector*)
Load descriptor from cached file. If failed, extract descriptor from image

Parameters

- **dataset_name** (*str*) – Name of the dataset
- **sequence_name** (*str*) – Name of the sequence
- **image** (Image) – Image
- **detector** (DetectorAndDescriptor) – Detector used to extract the descriptor

Returns descriptor

Return type array

load_feature (*dataset_name*, *sequence_name*, *image*, *detector*)
Load feature from cached file. If failed, extract feature from image

Parameters

- **dataset_name** (*str*) – Name of the dataset
- **sequence_name** (*str*) – Name of the sequence
- **image** (Image) – Image
- **detector** (DetectorAndDescriptor) – Detector used to extract the descriptor

Returns feature

Return type array

print_and_save_result (*results*)
Print and save result.

Parameters **results** (*dict*) – Result to show

This module describe benchmark for repeatability.

class bench.repBench.**repBench** (*tmp_feature_dir='./features/'*, *result_dir='./python_scores/'*)
Bases: *bench.BenchmarkTemplate.Benchmark*

Repeatability Template Return repeatability score and number of correspondence

detect_feature (*dataset*, *detector*, *use_cache=True*, *save_feature=True*)
Extract feature from image.

Parameters

- **dataset** (SequenceDataset) – Dataset to extract the feature
- **detector** (DetectorAndDescriptor) – Detector used to extract the feature
- **use_cache** (*boolean*) – Load cached feature and result or not
- **save_feature** (*boolean*) – Save computated feature or not

Returns feature

Return type dict

detect_feature_custom (*dataset*, *detector*, *use_cache=False*, *save_feature=True*)
Customized feature extraction method. For special task.

Parameters

- **dataset** (SequenceDataset) – Dataset to extract the feature
- **detector** (DetectorAndDescriptor) – Detector used to extract the feature
- **use_cache** (*boolean*) – Load cached feature and result or not
- **save_feature** (*boolean*) – Save computated feature or not

Returns feature

Return type dict

evaluate (*dataset*, *detector*, *use_cache=True*, *save_result=True*, *norm_factor='minab'*)
Main function to call the evaluation wrapper. It could be different for different evaluation

Parameters

- **dataset** (`SequenceDataset`) – Dataset to extract the feature
- **detector** (`DetectorAndDescriptor`) – Detector used to extract the feature
- **use_cache** (*boolean*) – Load cached feature and result or not
- **save_result** (*boolean*) – Save result or not
- **norm_factor** (*str*) – How to normalize the repeatability. Option: minab, a, b

Returns result

Return type dict

See also:

`bench.Benchmark`, `bench.Benchmark.evaluate_warpper`

evaluate_unit (*feature_1*, *feature_2*, *task*)

Single evaluation unit. Given two features, return the repeatability.

Parameters

- **feature_1** (*array*) – Feature to run.
- **feature_2** (*array*) – Feature to run.
- **task** (*dict*) – What to run

See also:

evaluate_warpper How to run the unit.

dset.dataset.Link definition of task.

evaluate_warpper (*dataset*, *detector*, *result_list*, *extract_descriptor=False*, *use_cache=True*, *save_result=True*, *custom_extraction=False*)

Load descriptor from cached file. If failed, extract descriptor from image.

Structure of the result:

result['dataset_name']: name of the dataset

result['result_term_list']: list of metrics for evaluation

result['task_name']: name of the task

result['detector_name']: name of the dataset

result['sequence_result']: a list for result from each sequence

result['ave_{}']: average value for each metric over all sequences

Structure of the sequence result:

sequence_result['sequence_name']: name of the sequence

sequence_result[result_name]: list of list of metrics over each link

sequence_result['result_label_list']: label of each link in sequence_result (Same order)

sequence_result['result_link_id_list']: ID of each link in sequence_result (Same order)

Parameters

- **dataset** (`SequenceDataset`) – Dataset to extract the feature
- **detector** (`DetectorAndDescriptor`) – Detector used to extract the feature
- **result_list** (*list*) – Metric to calculate

- **`extract_descriptor`** (*`boolean`*) – Extract descriptor or not
- **`use_cache`** (*`boolean`*) – Load cached feature and result or not
- **`save_result`** (*`boolean`*) – Save result or not
- **`custom_extraction`** (*`boolean`*) – Use custom extraction method or not. See also and extract_descriptor_custom

Returns result

Return type dict

See also:

`detect_feature_custom` Extract feature with customized method (special evaluation).

`extract_descriptor_custom` Extract descriptor with customized (special evaluation).

`extract_descriptor`(*dataset*, *detector*, *use_cache=False*, *save_feature=True*)
Extract feature from image.

Parameters

- **`dataset`** (`SequenceDataset`) – Dataset to extract the descriptor
- **`detector`** (`DetectorAndDescriptor`) – Detector used to extract the descriptor
- **`use_cache`** (*`boolean`*) – Load cached feature and result or not
- **`save_feature`** (*`boolean`*) – Save computated feature or not

Returns feature, descriptor

Return type dict, dict

`extract_descriptor_custom`(*dataset*, *detector*, *use_cache=False*, *save_feature=True*)
Customized description extraction method. For special task.

Parameters

- **`dataset`** (`SequenceDataset`) – Dataset to extract the descriptor
- **`detector`** (`DetectorAndDescriptor`) – Detector used to extract the descriptor
- **`use_cache`** (*`boolean`*) – Load cached feature and result or not
- **`save_feature`** (*`boolean`*) – Save computated feature or not

Returns feature

Return type dict

`load_csv_feature`(*csv_feature_file*)
Load feature from csvfile.

Parameters **`csv_feature_file`** (*`str`*) – csv file to load feature

Returns feature

Return type array

`load_descriptor`(*dataset_name*, *sequence_name*, *image*, *detector*)
Load descriptor from cached file. If failed, extract descriptor from image

Parameters

- **`dataset_name`** (*`str`*) – Name of the dataset

- **sequence_name** (*str*) – Name of the sequence
- **image** (Image) – Image
- **detector** (DetectorAndDescriptor) – Detector used to extract the descriptor

Returns descriptor

Return type array

load_feature (*dataset_name*, *sequence_name*, *image*, *detector*)
Load feature from cached file. If failed, extract feature from image

Parameters

- **dataset_name** (*str*) – Name of the dataset
- **sequence_name** (*str*) – Name of the sequence
- **image** (Image) – Image
- **detector** (DetectorAndDescriptor) – Detector used to extract the descriptor

Returns feature

Return type array

print_and_save_result (*results*)
Print and save result.

Parameters **results** (*dict*) – Result to show

This module describe benchmark for matching score.

class bench.MatchingScoreBench.**MatchingScoreBench** (*tmp_feature_dir='./features/'*, *result_dir='./python_scores/'*, *matchGeometry=True*)

Bases: *bench.BenchmarkTemplate.Benchmark*

Matching score benchmark Return repeatability score, number of correspondence, matching score and number of matches

detect_feature (*dataset*, *detector*, *use_cache=True*, *save_feature=True*)
Extract feature from image.

Parameters

- **dataset** (SequenceDataset) – Dataset to extract the feature
- **detector** (DetectorAndDescriptor) – Detector used to extract the feature
- **use_cache** (*boolean*) – Load cached feature and result or not
- **save_feature** (*boolean*) – Save computated feature or not

Returns feature

Return type dict

detect_feature_custom (*dataset*, *detector*, *use_cache=False*, *save_feature=True*)
Customized feature extraction method. For special task.

Parameters

- **dataset** (SequenceDataset) – Dataset to extract the feature
- **detector** (DetectorAndDescriptor) – Detector used to extract the feature
- **use_cache** (*boolean*) – Load cached feature and result or not

- **save_feature** (*boolean*) – Save computated feature or not

Returns feature

Return type dict

evaluate (*dataset*, *detector*, *use_cache=True*, *save_result=True*, *norm_factor='minab'*)
Main function to call the evaluation wrapper. It could be different for different evaluation

Parameters

- **dataset** (`SequenceDataset`) – Dataset to extract the feature
- **detector** (`DetectorAndDescriptor`) – Detector used to extract the feature
- **use_cache** (*boolean*) – Load cached feature and result or not
- **save_result** (*boolean*) – Save result or not
- **norm_factor** (*str*) – How to normalize the repeatability. Option: minab, a, b

Returns result

Return type dict

See also:

`bench.Benchmark`, `bench.Benchmark.evaluate_warpper`

evaluate_unit (*feature_1*, *feature_2*, *task*)
Single evaluation unit. Given two features, return the repeatability.

Parameters

- **feature_1** (*list of array [feature, descriptor]*) – Feature and descriptor to run.
- **feature_2** (*list of array [feature, descriptor]*) – Feature and descriptor to run.
- **task** (*dict*) – What to run

See also:

evaluate_warpper How to run the unit.

dset.dataset.Link definition of task.

evaluate_warpper (*dataset*, *detector*, *result_list*, *extract_descriptor=False*, *use_cache=True*, *save_result=True*, *custom_extraction=False*)
Load descriptor from cached file. If failed, extract descriptor from image.

Structure of the result:

result['dataset_name']: name of the dataset

result['result_term_list']: list of metrics for evaluation

result['task_name']: name of the task

result['detector_name']: name of the dataset

result['sequence_result']: a list for result from each sequence

result['ave_{}']: average value for each metric over all sequences

Structure of the sequence result:

sequence_result['sequence_name']: name of the sequence

sequence_result[result_name]: list of list of metrics over each link

sequence_result['result_label_list']: label of each link in sequence_result (Same order)

sequence_result['result_link_id_list']: ID of each link in sequence_result (Same order)

Parameters

- **dataset** (SequenceDataset) – Dataset to extract the feature
- **detector** (DetectorAndDescriptor) – Detector used to extract the feature
- **result_list** (*list*) – Metric to calculate
- **extract_descriptor** (*boolean*) – Extract descriptor or not
- **use_cache** (*boolean*) – Load cached feature and result or not
- **save_result** (*boolean*) – Save result or not
- **custom_extraction** (*boolean*) – Use custom extraction method or not. See also and extract_descriptor_custom

Returns result

Return type dict

See also:

detect_feature_custom Extract feature with customized method (special evaluation).

extract_descriptor_custom Extract descriptor with customized (special evaluation).

extract_descriptor (*dataset*, *detector*, *use_cache=False*, *save_feature=True*)
Extract feature from image.

Parameters

- **dataset** (SequenceDataset) – Dataset to extract the descriptor
- **detector** (DetectorAndDescriptor) – Detector used to extract the descriptor
- **use_cache** (*boolean*) – Load cached feature and result or not
- **save_feature** (*boolean*) – Save computated feature or not

Returns feature, descriptor

Return type dict, dict

extract_descriptor_custom (*dataset*, *detector*, *use_cache=False*, *save_feature=True*)
Customized description extraction method. For special task.

Parameters

- **dataset** (SequenceDataset) – Dataset to extract the descriptor
- **detector** (DetectorAndDescriptor) – Detector used to extract the descriptor
- **use_cache** (*boolean*) – Load cached feature and result or not
- **save_feature** (*boolean*) – Save computated feature or not

Returns feature

Return type dict

load_csv_feature (*csv_feature_file*)
Load feature from csvfile.

Parameters **`csv_feature_file`** (*`str`*) – csv file to load feature

Returns feature

Return type array

`load_descriptor`(*dataset_name*, *sequence_name*, *image*, *detector*)
Load descriptor from cached file. If failed, extract descriptor from image

Parameters

- **`dataset_name`** (*`str`*) – Name of the dataset
- **`sequence_name`** (*`str`*) – Name of the sequence
- **`image`** (`Image`) – Image
- **`detector`** (`DetectorAndDescriptor`) – Detector used to extract the descriptor

Returns descriptor

Return type array

`load_feature`(*dataset_name*, *sequence_name*, *image*, *detector*)
Load feature from cached file. If failed, extract feature from image

Parameters

- **`dataset_name`** (*`str`*) – Name of the dataset
- **`sequence_name`** (*`str`*) – Name of the sequence
- **`image`** (`Image`) – Image
- **`detector`** (`DetectorAndDescriptor`) – Detector used to extract the descriptor

Returns feature

Return type array

`print_and_save_result`(*results*)
Print and save result.

Parameters **`results`** (*`dict`*) – Result to show

This module describe benchmark for baseline matching.

`class` `bench.W1BSBench.`**`W1BSBench`**(*tmp_feature_dir='./features/'*, *result_dir='./python_scores/'*)
Bases: *`bench.BenchmarkTemplate.Benchmark`*

Baseline matching benchmark Return ap

`detect_feature`(*dataset*, *detector*, *use_cache=True*, *save_feature=True*)
Extract feature from image.

Parameters

- **`dataset`** (`SequenceDataset`) – Dataset to extract the feature
- **`detector`** (`DetectorAndDescriptor`) – Detector used to extract the feature
- **`use_cache`** (*`boolean`*) – Load cached feature and result or not
- **`save_feature`** (*`boolean`*) – Save computated feature or not

Returns feature

Return type dict

detect_feature_custom(*dataset*, *detector*, *use_cache=False*, *save_feature=True*)

Customized feature extraction method. For special task.

Parameters

- **dataset** (SequenceDataset) – Dataset to extract the feature
- **detector** (DetectorAndDescriptor) – Detector used to extract the feature
- **use_cache** (*boolean*) – Load cached feature and result or not
- **save_feature** (*boolean*) – Save computated feature or not

Returns feature

Return type dict

evaluate(*dataset*, *detector*, *use_cache=True*, *save_result=True*)

Main function to call the evaluation wrapper. It could be different for different evaluation

Parameters

- **dataset** (SequenceDataset) – Dataset to extract the feature
- **detector** (DetectorAndDescriptor) – Detector used to extract the feature
- **use_cache** (*boolean*) – Load cached feature and result or not
- **save_result** (*boolean*) – Save result or not

Returns result

Return type dict

See also:

bench.Benchmark, bench.Benchmark.evaluate_warpper

evaluate_unit(*feature_1*, *feature_2*, *task*)

Single evaluation unit. Given two features, return the repeatability.

Parameters

- **feature_1** (*list of array [feature, descriptor]*) – Feature and descriptor to run.
- **feature_2** (*list of array [feature, descriptor]*) – Feature and descriptor to run.
- **task** (*dict*) – What to run

See also:

evaluate_warpper How to run the unit.

dset.dataset.Link definition of task.

evaluate_warpper(*dataset*, *detector*, *result_list*, *extract_descriptor=False*, *use_cache=True*, *save_result=True*, *custom_extraction=False*)

Load descriptor from cached file. If failed, extract descriptor from image.

Structure of the result:

result['dataset_name']: name of the dataset

result['result_term_list']: list of metrics for evaluation

result['task_name']: name of the task

result['detector_name']: name of the dataset

result['sequence_result']: a list for result from each sequence

result['ave_{}']: average value for each metric over all sequences

Structure of the sequence result:

sequence_result['sequence_name']: name of the sequence

sequence_result[result_name]: list of list of metrics over each link

sequence_result['result_label_list']: label of each link in sequence_result (Same order)

sequence_result['result_link_id_list']: ID of each link in sequence_result (Same order)

Parameters

- **`dataset`** (`SequenceDataset`) – Dataset to extract the feature
- **`detector`** (`DetectorAndDescriptor`) – Detector used to extract the feature
- **`result_list`** (*`list`*) – Metric to calculate
- **`extract_descriptor`** (*`boolean`*) – Extract descriptor or not
- **`use_cache`** (*`boolean`*) – Load cached feature and result or not
- **`save_result`** (*`boolean`*) – Save result or not
- **`custom_extraction`** (*`boolean`*) – Use custom extraction method or not. See also and extract_descriptor_custom

Returns result

Return type dict

See also:

`detect_feature_custom` Extract feature with customized method (special evaluation).

`extract_descriptor_custom` Extract descriptor with customized (special evaluation).

`extract_descriptor`(*dataset*, *detector*, *use_cache=False*, *save_feature=True*)
Extract feature from image.

Parameters

- **`dataset`** (`SequenceDataset`) – Dataset to extract the descriptor
- **`detector`** (`DetectorAndDescriptor`) – Detector used to extract the descriptor
- **`use_cache`** (*`boolean`*) – Load cached feature and result or not
- **`save_feature`** (*`boolean`*) – Save computated feature or not

Returns feature, descriptor

Return type dict, dict

`extract_descriptor_custom`(*dataset*, *detector*, *use_cache=False*, *save_feature=True*)
Customized description extraction method. Get descriptor from images of patches.

Parameters

- **`dataset`** (`SequenceDataset`) – Dataset to extract the descriptor
- **`detector`** (`DetectorAndDescriptor`) – Detector used to extract the descriptor

- **use_cache** (*boolean*) – Load cached feature and result or not
- **save_feature** (*boolean*) – Save computated feature or not

Returns feature

Return type dict

load_csv_feature (*csv_feature_file*)
Load feature from csvfile.

Parameters **csv_feature_file** (*str*) – csv file to load feature

Returns feature

Return type array

load_descriptor (*dataset_name*, *sequence_name*, *image*, *detector*)
Load descriptor from cached file. If failed, extract descriptor from image

Parameters

- **dataset_name** (*str*) – Name of the dataset
- **sequence_name** (*str*) – Name of the sequence
- **image** (Image) – Image
- **detector** (DetectorAndDescriptor) – Detector used to extract the descriptor

Returns descriptor

Return type array

load_feature (*dataset_name*, *sequence_name*, *image*, *detector*)
Load feature from cached file. If failed, extract feature from image

Parameters

- **dataset_name** (*str*) – Name of the dataset
- **sequence_name** (*str*) – Name of the sequence
- **image** (Image) – Image
- **detector** (DetectorAndDescriptor) – Detector used to extract the descriptor

Returns feature

Return type array

print_and_save_result (*results*)
Print and save result.

Parameters **results** (*dict*) – Result to show

This module describe benchmark for image retrieval.

class bench.RetrievalBenchmark.**RetrievalBenchmark** (*tmp_feature_dir='./data/features/'*, *result_dir='./python_scores/'*)
Bases: object

evaluate (*dataset*, *detector*, *use_cache=True*, *save_result=True*)
Main function to run the evaluation wrapper. It could be different for different evaluation

Parameters

- **dataset** (SequenceDataset) – Dataset to extract the feature
- **detector** (DetectorAndDescriptor) – Detector used to extract the feature

- **`use_cache`** (*`boolean`*) – Load cached feature and result or not
- **`save_result`** (*`boolean`*) – Save result or not

See also:

`evaluate_warpper`

`evaluate_warpper`(*dataset*, *detector*, *result_list*, *l2_norm=True*, *use_cache=True*, *save_result=True*, *custom_extraction=False*)

Load descriptor from cached file. If failed, extract descriptor from image.

Structure of the result:

result['dataset_name']: name of the dataset

result['result_term_list']: list of metrics for evaluation

result['task_name']: name of the task

result['detector_name']: name of the dataset

result['ave_{}']: average value for each metric over all sequences

Parameters

- **`dataset`** (`SequenceDataset`) – Dataset to extract the feature
- **`detector`** (`DetectorAndDescriptor`) – Detector used to extract the feature
- **`result_list`** (*`list`*) – Metric to calculate
- **`l2_norm`** (*`boolean`*) – Perform l2 normalization to descriptor or not
- **`use_cache`** (*`boolean`*) – Load cached feature and result or not
- **`save_result`** (*`boolean`*) – Save result or not
- **`custom_extraction`** (*`boolean`*) – Use custom extraction method or not. See also and extract_descriptor_custom

Returns result

Return type dict

See also:

`detect_feature_custom` Extract feature with customized method (special evaluation).

`extract_descriptor_custom` Extract descriptor with customized (special evaluation).

`extract_descriptor`(*dataset*, *detector*, *use_cache=False*, *save_feature=True*)

Extract descriptors from images.

Parameters

- **`dataset`** (`RetrievalDataset`) – Dataset to extract the descriptor
- **`detector`** (`DetectorAndDescriptor`) – Detector used to extract the descriptor
- **`use_cache`** (*`boolean`*) – Load cached feature and result or not
- **`save_feature`** (*`boolean`*) – Save computated feature or not

Returns feature, descriptor

Return type dict, dict

get_sorted_index_and_score(*image_index*, *flat_D*)

Given local feature to image index and distance to each local features, return image score

Parameters

- **dataset** (SequenceDataset) – Dataset to extract the feature
- **detector** (DetectorAndDescriptor) – Detector used to extract the feature
- **use_cache** (*boolean*) – Load cached feature and result or not
- **save_result** (*boolean*) – Save result or not

Returns sorted_index, sorted_score, sorted_count, score_dict, count_dict

Return type array, array, array, dict, dict

return sorted image index based on score, sorted score, sorted number of matched point, dict of image id to score, dict of image id to number of matched points.

load_csv_feature(*csv_feature_file*)

Load feature from csvfile.

Parameters **csv_feature_file** (*str*) – csv file to load feature

Returns feature

Return type array

print_and_save_result()

Print and save result.

Parameters **results** (*dict*) – Result to show

CHAPTER NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX