CHEMNITZ UNIVERSITY OF TECHNOLOGY

Superpixel Benchmark - Starting Guide

Peer Neubert, *peer.neubert@etit.tu-chemnitz.de*

Contents

1 Introduction

We provide two benchmarks: one for segmentation quality and another for robustness against affine transformations. This is a short documentation. Feel free to contact the autors in case of questions. Prerequisites to run the benchmark are:

- The benchmark toolbox:
  `http://www.tu-chemnitz.de/etit/proaut/forschung/superpixel.html`
- Matlab with Image Processing Toolbox
- BSDS 500 images:
  `http//www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/BSR/BSR_bsds500.tgz`
- The benchmark is tested only under Linux operating systems

2 How To Use The Image Data

The BSDS 500 images are divided into 200 training images, 100 validation images and 200 test images. These image sets can be accessed via BPF files by setting "mode" to train, val or test (see section 3.1)

train Training data is aimed for training or learning. When additional data is used for training, this should be stated.

val This is the validation data set for parameter adjusting. It is valid to use the training data for parameter adjusting as well.

test For the final benchmark, exactly these 200 test images have to be used. This data must not be used for parameter adjusting. Use this data rarely and with caution.

The BSDS500 dataset contains multiple ground truth segmentations for each image. These are manual labelings of object borders. Since superpixel segmentation is by definition an oversegmentation of the image, those multiple manual segmentations are combined to a single boundary map. This boundary map states for each pixel if this was marked as a boundary in at least one manual segmentation.

3 How To Benchmark Segmentation Quality

3.1 Overview & BPF

There are three steps:

1. Run segmentation algorithm to segment images (*runAlgorithm.m*)
2. Compute the benchmark error metrics (*runBenchmark.m*)
3. Show the results (*evaluateBenchmark.m*)

main_benchmark.m runs all three steps. Configuration of the algorithm settings is done with Benchmark-Parameter-Files (bpf-files) BPFs are simple textfiles containing the configuration of the benchmark and the algorithm. They contain variable name-value pairs and are parsed by the parseBenchmarkParameterFile function from parseBenchmarkParameterFile.m

mode (val or train or test) Use BSDS500 validation, training or test data

BSDS500_root path to the BSDS500 image folder
(e.g. /home/user/bsds500/BSR/BSDS500/data)

nImages number of images (e.g. 200 for test)

algResSavePath path where the results (segmented images and evaluations) are stored. At this path a seperate folder for each parameter set of the algorithm is created.

algorithmCommand This is the tricky part. This is the line of code that is executed by Matlab. There are three predefined variables:

- I is the input image (for BSDS500: 8bit three channel RGB image).
- S is the resulting segmentation image (multi label image, integer, all pixels of the same segment have the same image value, maximum value equals the number of segments).
- *time* is the runtime in seconds.

All other vaiable names are variables, that can be set in BPF file with the parameterset keyword. These variables are replaced with the value of the current parameterset.

parameterset Here, parameters for one setting of the algorithm are given. Write multiple parameterset (each at one line) with different varaible values and all will be evaluated. The parameterSet keyword is followed by name-value pairs of either predefined keywords (e.g. parametersetName) or variables from algorithmCommand. parametersetName is the name of the parameter set in this line and is used to define the name of the subfolder for the storage of the results.

For variables from algorithmCommand: During runAlgorithm-function, the variable values are set via text replacement into the algorithmCommand. Take care to use names that do not appear at other places in algorithmCommand! To not use a parameterset, just comment it with a # or % or //

3.2 Call Hierarchy

Despite there are several files and functions necessary for the benchmark, there is a (hopefully) clear strcuture and flat call hierarchy:

main_benchmark.m
- *runAlgorithm.m*
  - *parseBenchmarkParameterFile.m*
  - *loadBSDS500.m*
- *runBenchmark.m*
  - *parseBenchmarkParameterFile.m*
  - *loadBSDS500.m*
  - *multiLabelImage2boundaryImage.m*
  - *combineMultipleBoundaryImages.m*
  - *compareBoundaryImagesSimple.m*
  - *getUndersegmentationError.m*

evaluateBenchmark.m
- *prepareBoundaryRecallPlot.m*
- *prepareUndersegmentationErrorPlot.m*
- *prepareRuntimePlot.m*
- *getHighContrastColormap.m*
- *evalBPF_plot.m*
  - *parseBenchmarkParameterFile.m*

3.3 Example Run

This example shows application of the toolbox for benchmarking a new algorithm. Make sure to have the BSDS 500 images available. However, you do not need your own segmentation algorithm, we provide the very simple BOX segmentation algorithm in *segmentation_algorithms/segment_box.m*, that divides the image in a regular grid.

```
% add path to your segmentation algorithm
addpath('segmentation_algorithms/');

% create your BPF file, here we use bpf/bpf_BOX.txt
% Very likely you need to adapt the path to the
% BSDS 500 image data.

% segment images with your algorithm
runAlgorithm('bpf/bpf_BOX.txt');

% compute error metrics
runBenchmark('bpf/bpf_BOX.txt');

% visualize the results
evaluateBenchmark('bpf/bpf_BOX.txt');

% visualize the results of several algorithms
% (here two times the same results)
evaluateBenchmark({'bpf/bpf_BOX.txt', 'bpf/bpf_BOX.txt'},
                  'names', {'BOX1', 'BOX2'});
```

4 How To Benchmark Segmentation Robustness

4.1 Overview & APF

Configuration of the affine transformations are implemented via Affine-Parameter-Files (APF). The idea of APF is similar to the one of a BPF, but here, just the affine transformations and the save path extension are stored.

algResSavePathExtension is the folder name extension to the result folder name from the BPF file

affine_trafo is followed by the name of this transformation and the 3x3 matrix in row major order.

Example APF files are given for shift, rotation, shear and scale. New ones at least require adaption of the visualization in evaluateBenchmarkAffine. BPF files for the affine benchmark typically have just one parameter set, since the variables to evauate on are the affine parameters.

4.2 Call Hierarchy

main_runAffine.m
- *runAlgorithmAffine.m*
  - *parseBenchmarkParameterFile.m*
  - *parseAffineParameterFile.m*
  - *loadBSDS500.m*
  - *appplyTransform.m*
- *runBenchmarkAffine.m*
  - *compareBoundaryImagesSimple.m*
  - *loadBSDS500.m*
  - *multiLabelImage2boundaryImage.m*
  - *parseAffineParameterFile.m*
  - *parseBenchmarkParameterFile.m*

evaluateBenchmarkAffine.m
- *parseAffineParameterFile.m*
- *parseBenchmarkParameterFile.m*

4.3 Example Run

```
% add path to your segmentation algorithm
addpath('segmentation_algorithms/');

% create your BPF file, here we use bpf/affine_bpf_BOX_250.txt that
% contains the BOX setting for about 250 segments
% Very likely you need to adapt the path to the BSDS 500 image data.

% main_runAffine.m runs the segmentation algorithm for several APFs.
```

```
main_runAffine('bpf/affine_bpf_BOX_250.txt')

% show results on rotation
evaluateBenchmarkAffine('bpf/affine_bpf_BOX_250.txt',
                        'apf/apf_rotation.txt')
evaluateBenchmarkAffine({'bpf/affine_bpf_BOX_250.txt',
                         'bpf/affine_bpf_BOX_250.txt'},
                        'apf/apf_rotation.txt')
```