

---

# **ImSegm Documentation**

***Release 0.1.8***

**Jiri Borovec**

**May 24, 2021**

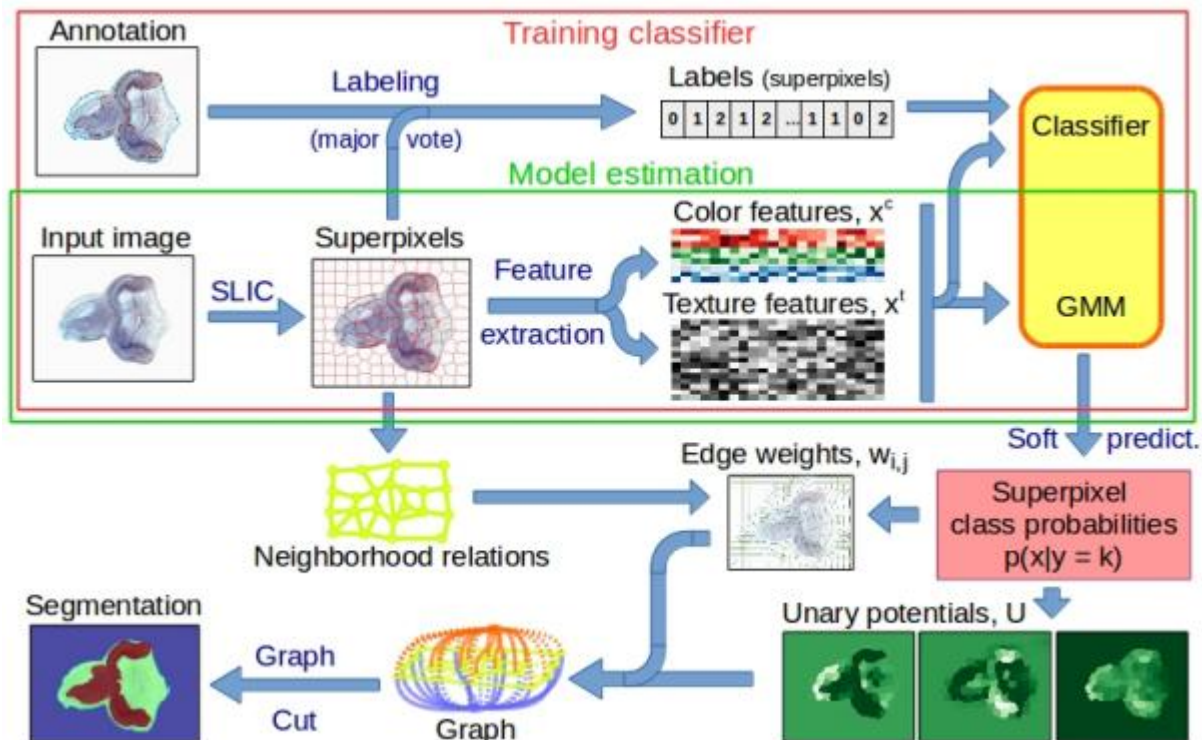


# CONTENTS

<b>1</b>	<b>Contents</b>	<b>1</b>
1.1	Image segmentation toolbox . . . . .	1
1.2	imsegm package . . . . .	10
1.3	Examples . . . . .	132
<b>2</b>	<b>Indices and tables</b>	<b>193</b>
<b>3</b>	<b>General superpixel image segmentation: (un)supervised, center detection, region growing</b>	<b>195</b>
3.1	Superpixel segmentation with GraphCut regularisation . . . . .	195
3.2	Object centre detection and Ellipse approximation . . . . .	196
3.3	Superpixel Region Growing with Shape prior . . . . .	196
3.4	References . . . . .	196
	<b>Python Module Index</b>	<b>197</b>
	<b>Index</b>	<b>199</b>



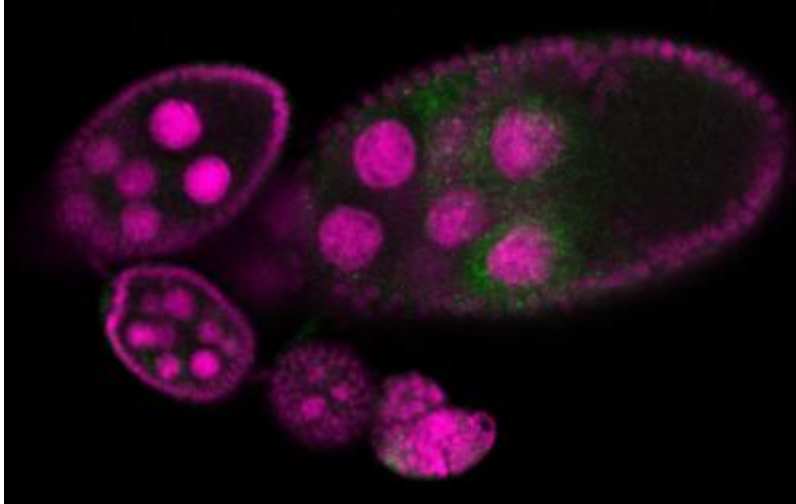
Image segmentation is widely used as an initial phase of many image processing tasks in computer vision and image analysis. Many recent segmentation methods use superpixels because they reduce the size of the segmentation problem by order of magnitude. Also, features on superpixels are much more robust than features on pixels only. We use spatial regularisation on superpixels to make segmented regions more compact. The segmentation pipeline comprises (i) computation of superpixels; (ii) extraction of descriptors such as colour and texture; (iii) soft classification, using a standard classifier for supervised learning, or the Gaussian Mixture Model for unsupervised learning; (iv) final segmentation using Graph Cut. We use this segmentation pipeline on real-world applications in medical imaging (see [sample images](#). We also show that [unsupervised segmentation](#) is sufficient for some situations, and provides similar results to those obtained using [trained segmentation](#).



**Sample ipython notebooks:**

- [Supervised segmentation](#) requires training annotation
- [Unsupervised segmentation](#) just asks for expected number of classes
- **partially annotated images** with missing annotation is marked by a negative number

**Illustration**



Reference: Borovec J., Svihlik J., Kybic J., Habart D. (2017). *Supervised and unsupervised segmentation using superpixels, model estimation, and Graph Cut*. In: *Journal of Electronic Imaging*.

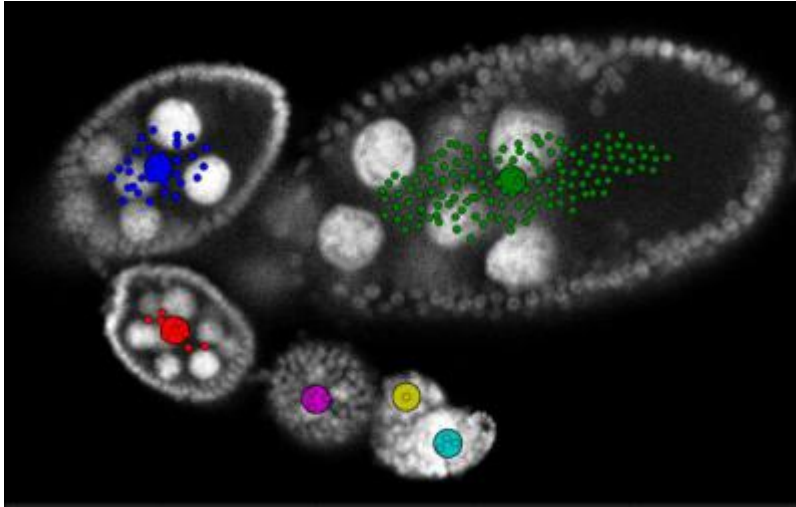
## 1.1.2 Object centre detection and Ellipse approximation

An image processing pipeline to detect and localize Drosophila egg chambers that consists of the following steps: (i) superpixel-based image segmentation into relevant tissue classes (see above); (ii) detection of egg center candidates using label histograms and ray features; (iii) clustering of center candidates and; (iv) area-based maximum likelihood ellipse model fitting. See our [Poster](#) related to this work.

**Sample ipython notebooks:**

- [Center detection](#) consists of center candidate training and prediction, and candidate clustering.
- [Ellipse fitting](#) with given estimated center structure segmentation.

**Illustration**



Reference: Borovec J., Kybic J., Nava R. (2017) *Detection and Localization of Drosophila Egg Chambers in Microscopy Images*. In: *Machine Learning in Medical Imaging*.

### 1.1.3 Superpixel Region Growing with Shape prior

Region growing is a classical image segmentation method based on hierarchical region aggregation using local similarity rules. Our proposed approach differs from standard region growing in three essential aspects. First, it works on the level of superpixels instead of pixels, which leads to a substantial speedup. Second, our method uses learned statistical shape properties which encourage growing leading to plausible shapes. In particular, we use ray features to describe the object boundary. Third, our method can segment multiple objects and ensure that the segmentations do not overlap. The problem is represented as energy minimisation and is solved either greedily, or iteratively using GraphCuts.

**Sample ipython notebooks:**

- [General GraphCut](#) from given centers and initial structure segmentation.
- [Shape modeling](#) estimation from training examples.
- [Region growing](#) from given centers and initial structure segmentation with shape models.

**Illustration**

Reference: Borovec J., Kybic J., Sugimoto, A. (2017). *Region growing using superpixels with learned shape prior*. In: *Journal of Electronic Imaging*.

### 1.1.4 Installation and configuration

#### Configure local environment

Create your own local environment, for more see the [User Guide](#), and install dependencies requirements.txt contains list of packages and can be installed as

```
@duda:~$ cd pyImSegm
@duda:~/pyImSegm$ virtualenv env
@duda:~/pyImSegm$ source env/bin/activate
(env)duda:~/pyImSegm$ pip install -r requirements.txt
(env)duda:~/pyImSegm$ python ...
```

and in the end terminating...

```
(env)@duda:~/pyImSegm$ deactivate
```

### Compilation

We have implemented cython version of some functions, especially computing descriptors, which require to compile them before using them

```
python setup.py build_ext --inplace
```

If loading of compiled descriptors in cython fails, it is automatically swapped to use numpy which gives the same results, but it is significantly slower.

### Installation

The package can be installed via pip

```
pip install git+https://github.com/Borda/pyImSegm.git
```

or using setuptools from a local folder

```
python setup.py install
```

---

## 1.1.5 Experiments

Short description of our three sets of experiments that together compose single image processing pipeline in this order:

1. **Semantic (un/semi)supervised segmentation**
2. **Center detection and ellipse fitting**
3. **Region growing with the learned shape prior**

### Annotation tools

We introduce some useful tools for work with image annotation and segmentation.

- **Quantization:** in case you have some smooth colour labelling in your images you can remove them with following quantisation script.

```
python handling_annotations/run_image_color_quantization.py \  
-imgs "./data-images/drosophila_ovary_slice/segm_rgb/*.png" \  
-m position -thr 0.01 --nb_workers 2
```

- **Paint labels:** concerting image labels into colour space and other way around.

```
python handling_annotations/run_image_convert_label_color.py \  
-imgs "./data-images/drosophila_ovary_slice/segm/*.png" \  
-out ./data-images/drosophila_ovary_slice/segm_rgb
```

- **Visualisation:** having input image and its segmentation we can use simple visualisation which overlap the segmentation over input image.



```
python handling_annotations/run_overlap_images_segms.py \
  -imgs "./data-images/drosophila_ovary_slice/image/*.jpg" \
  -segs ./data-images/drosophila_ovary_slice/segm \
  -out ./results/overlap_ovary_segment
```

- **In-painting** selected labels in segmentation.

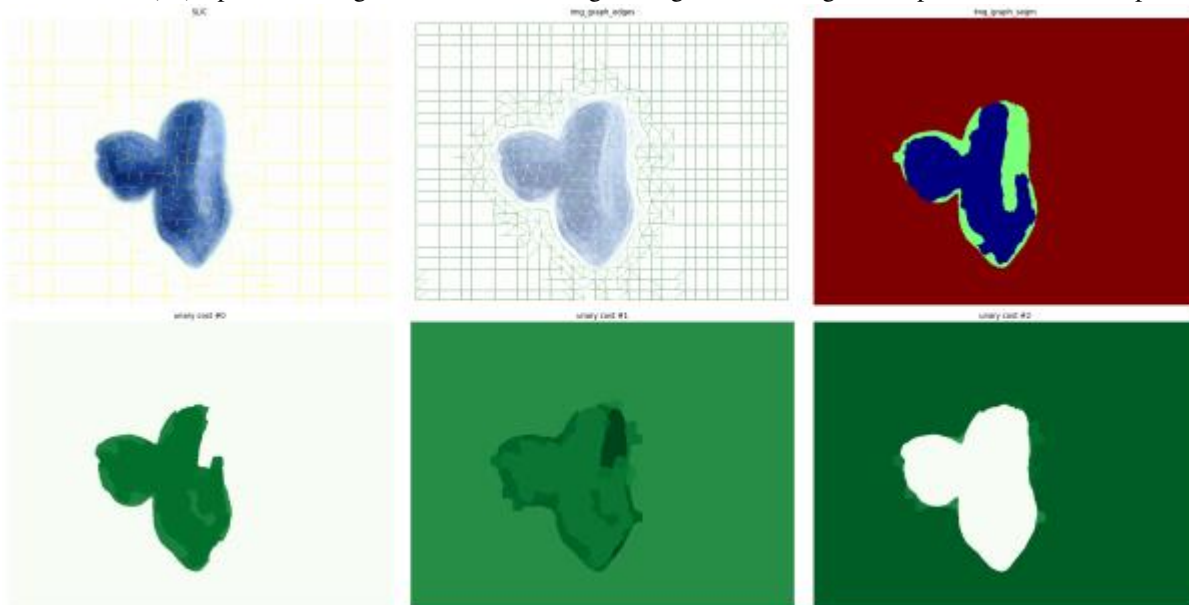
```
python handling_annotations/run_segmn_annot_inpaint.py \
  -imgs "./data-images/drosophila_ovary_slice/segm/*.png" \
  --label 4
```

- **Replace labels:** change labels in input segmentation into another set of labels in 1:1 schema.

```
python handling_annotations/run_segmn_annot_relabel.py \
  -out ./results/relabel_center_levels \
  --label_old 2 3 --label_new 1 1
```

## Semantic (un/semi)supervised segmentation

We utilise (un)supervised segmentation according to given training examples or some expectations.



- Evaluate superpixels (with given SLIC parameters) quality against given segmentation. It helps to find out the best SLIC configuration.

```
python experiments_segmentation/run_eval_superpixels.py \
  -imgs "./data-images/drosophila_ovary_slice/image/*.jpg" \
  -segm "./data-images/drosophila_ovary_slice/annot_eggs/*.png" \
  --img_type 2d_split \
  --slc_size 20 --slc_regul 0.25 --slico
```

- Perform **Un-Supervised** segmentation in images given in CSV

```
python experiments_segmentation/run_segmn_slc_model_graphcut.py \
  -l ./data-images/langerhans_islets/list_lang_isl_imgs-annot.csv -i "" \
```

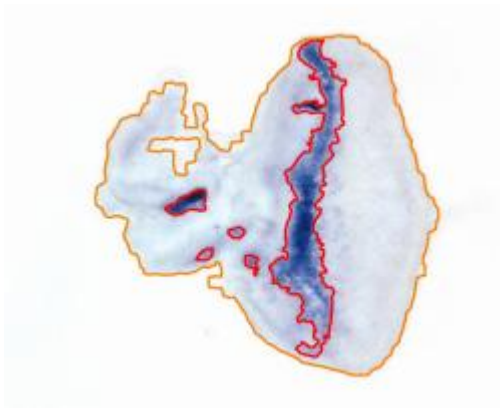
(continues on next page)

(continued from previous page)

```
-cfg experiments_segmentation/sample_config.yml \
-o ./results -n langIsl --nb_classes 3 --visual --nb_workers 2
```

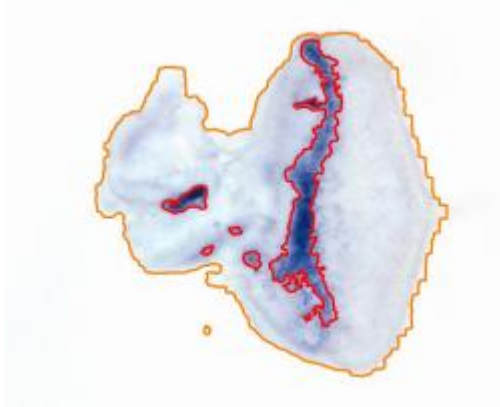
OR specified on particular path:

```
python experiments_segmentation/run_segmslic_model_graphcut.py \
-l "" -i "./data-images/landerhans_islets/image/*.jpg" \
-cfg ./experiments_segmentation/sample_config.yml \
-o ./results -n langIsl --nb_classes 3 --visual --nb_workers 2
```



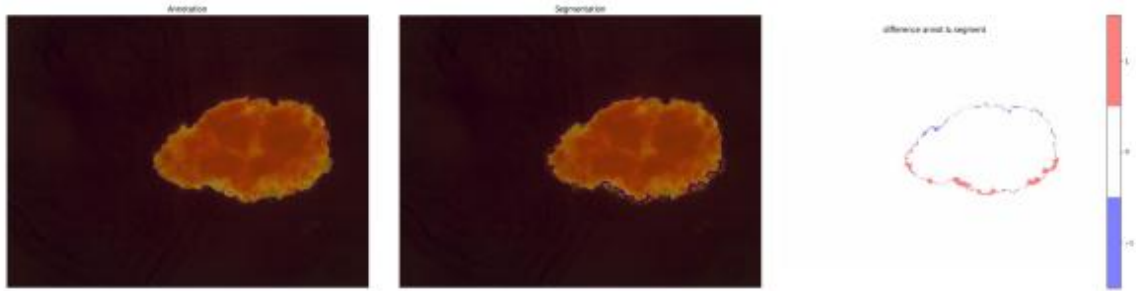
- Perform **Supervised** segmentation with afterwards evaluation.

```
python experiments_segmentation/run_segmslic_classif_graphcut.py \
-l ./data-images/drosophila_ovary_slice/list_imgs-annot-struct.csv \
-i "./data-images/drosophila_ovary_slice/image/*.jpg" \
--path_config ./experiments_segmentation/sample_config.yml \
-o ./results -n Ovary --img_type 2d_split --visual --nb_workers 2
```



- Perform **Semi-Supervised** is using the the supervised pipeline with not fully annotated images.
- For both experiment you can evaluate segmentation results.

```
python experiments_segmentation/run_compute-stat_annot-segm.py \
-a "./data-images/drosophila_ovary_slice/annot_struct/*.png" \
-s "./results/experiment_segmsupervise_ovary/*.png" \
-i "./data-images/drosophila_ovary_slice/image/*.jpg" \
-o ./results/evaluation --visual
```



The previous two (un)segmentation accept [configuration file](#) (YAML) by parameter `-cfg` with some extra parameters which was not passed in arguments, for instance:

```
slic_size: 35
slic_regul: 0.2
features:
  color_hsv: ['mean', 'std', 'eng']
classif: 'SVM'
nb_classif_search: 150
gc_edge_type: 'model'
gc_regul: 3.0
run_LOO: false
run_LPO: true
cross_val: 0.1
```

### Center detection and ellipse fitting

In general, the input is a formatted list (CSV file) of input images and annotations. Another option is set by `-list none` and then the list is paired with given paths to images and annotations.

**Experiment sequence is the following:**

1. We can create the annotation completely manually or use the following script which uses annotation of individual objects and create the zones automatically.

```
python experiments_ovary_centres/run_create_annotation.py
```

2. With zone annotation, we train a classifier for centre candidate prediction. The annotation can be a CSV file with annotated centres as points, and the zone of positive examples is set uniformly as the circular neighbourhood around these points. Another way (preferable) is to use an annotated image with marked zones for positive, negative and neutral examples.

```
python experiments_ovary_centres/run_center_candidate_training.py -list none \
  -segs "./data-images/drosophila_ovary_slice/segm/*.png" \
  -imgs "./data-images/drosophila_ovary_slice/image/*.jpg" \
  -centers "./data-images/drosophila_ovary_slice/center_levels/*.png" \
  -out ./results -n ovary
```

3. Having trained classifier we perform center prediction composed from two steps: i) center candidate clustering and ii) candidate clustering.

```
python experiments_ovary_centres/run_center_prediction.py -list none \
  -segs "./data-images/drosophila_ovary_slice/segm/*.png" \
  -imgs "./data-images/drosophila_ovary_slice/image/*.jpg" \
  -centers ./results/detect-centers-train_ovary/classifier_RandForest.pkl \
  -out ./results -n ovary
```

4. Assuming you have an expert annotation you can compute static such as missed eggs.

```
python experiments_ovary_centres/run_center_evaluation.py
```

5. This is just cut out clustering in case you want to use different parameters.

```
python experiments_ovary_centres/run_center_clustering.py \  
-segs "./data-images/drosophila_ovary_slice/segm/*.png" \  
-imgs "./data-images/drosophila_ovary_slice/image/*.jpg" \  
-centers "./results/detect-centers-train_ovary/candidates/*.csv" \  
-out ./results
```

6. Matching the ellipses to the user annotation.

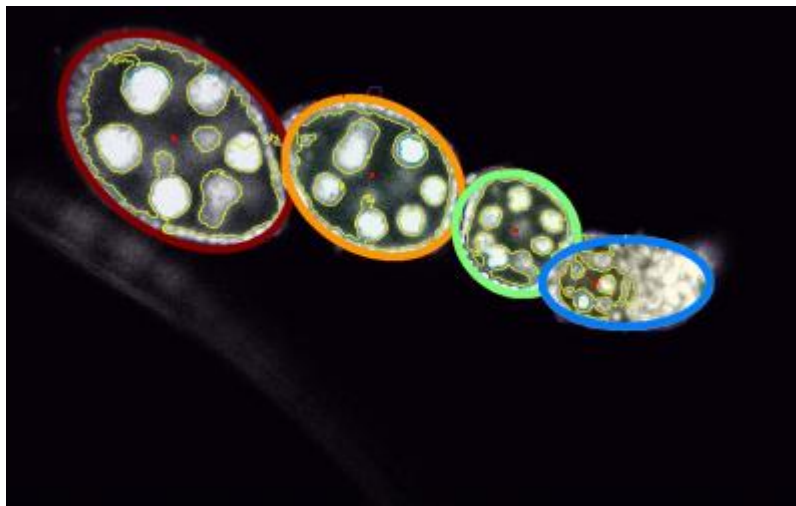
```
python experiments_ovary_detect/run_ellipse_annot_match.py \  
-info "~/Medical-drosophila/all_ovary_image_info_for_prague.txt" \  
-ells "~/Medical-drosophila/RESULTS/3_ellipse_ransac_crit_params/*.csv" \  
-out ~/Medical-drosophila/RESULTS
```

7. Cut eggs by stages and norm to mean size.

```
python experiments_ovary_detect/run_ellipse_cut_scale.py \  
-info ~/Medical-drosophila/RESULTS/info_ovary_images_ellipses.csv \  
-imgs "~/Medical-drosophila/RESULTS/0_input_images_png/*.png" \  
-out ~/Medical-drosophila/RESULTS/images_cut_ellipse_stages
```

8. Rotate (swap) extracted eggs according the larger mount of mass.

```
python experiments_ovary_detect/run_egg_swap_orientation.py \  
-imgs "~/Medical-drosophila/RESULTS/atlas_datasets/ovary_images/stage_3/*.png" \  
↔ " \  
-out ~/Medical-drosophila/RESULTS/atlas_datasets/ovary_images/stage_3
```



## Region growing with a shape prior

In case you do not have estimated object centres, you can use [plugins](#) for landmarks import/export for Fiji.

**Note:** install the multi-snake package which is used in multi-method segmentation experiment.

```
pip install --user git+https://github.com/Borda/morph-snakes.git
```

**Experiment sequence is the following:**

1. Estimating the shape model from set training images containing a single egg annotation.

```
python experiments_ovary_detect/run_RG2Sp_estim_shape-models.py \
    -annot "~/Medical-drosophila/egg_segmentation/mask_2d_slice_complete_ind_egg/
    ↪ *.png" \
    -out ./data-images -nb 15
```

2. Run several segmentation techniques on each image.

```
python experiments_ovary_detect/run_ovary_egg-segmentation.py \
    -list ./data-images/drosophila_ovary_slice/list_imgs-segm-center-points.csv \
    -out ./results -n ovary_image --nb_workers 1 \
    -m ellipse_moments \
        ellipse_ransac_mmt \
        ellipse_ransac_crit \
        GC_pixels-large \
        GC_pixels-shape \
        GC_slic-large \
        GC_slic-shape \
        rg2sp_greedy-mixture \
        rg2sp_GC-mixture \
        watershed_morph
```

3. Evaluate your segmentation ./results to export annotation.

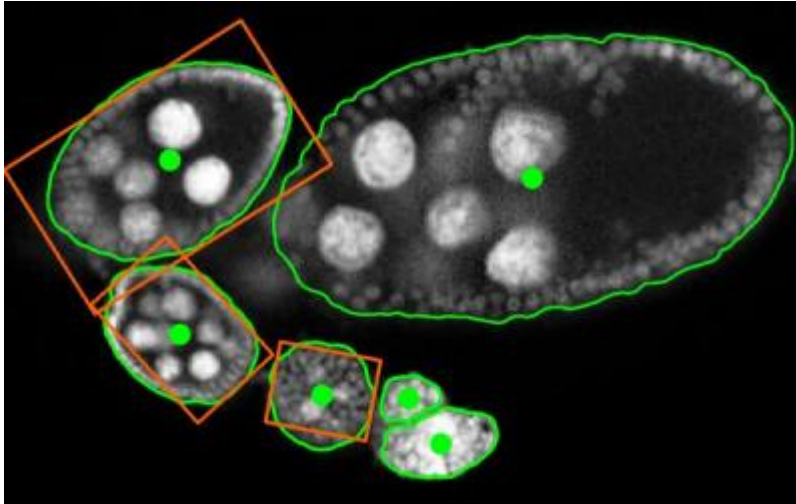
```
python experiments_ovary_detect/run_ovary_segm_evaluation.py --visual
```

4. In the end, cut individual segmented objects comes as minimal bounding box.

```
python experiments_ovary_detect/run_cut_segmented_objects.py \
    -annot "./data-images/drosophila_ovary_slice/annot_eggs/*.png" \
    -img "./data-images/drosophila_ovary_slice/segm/*.png" \
    -out ./results/cut_images --padding 50
```

5. Finally, performing visualisation of segmentation results together with expert annotation.

```
python experiments_ovary_detect/run_export_user-annot-segm.py
```



---

## 1.1.6 References

For complete references see [BibTex](#).

1. Borovec J., Svihlik J., Kybic J., Habart D. (2017). **Supervised and unsupervised segmentation using superpixels, model estimation, and Graph Cut**. SPIE Journal of Electronic Imaging 26(6), 061610. DOI: [10.1117/1.JEI.26.6.061610](#).
2. Borovec J., Kybic J., Nava R. (2017) **Detection and Localization of Drosophila Egg Chambers in Microscopy Images**. In: Wang Q., Shi Y., Suk H.I., Suzuki K. (eds) Machine Learning in Medical Imaging. MLMI 2017. LNCS, vol 10541. Springer, Cham. DOI: [10.1007/978-3-319-67389-9\\_3](#).
3. Borovec J., Kybic J., Sugimoto, A. (2017). **Region growing using superpixels with learned shape prior**. SPIE Journal of Electronic Imaging 26(6), 061611. DOI: [10.1117/1.JEI.26.6.061611](#).

## 1.2 imsegm package

### 1.2.1 Subpackages

**imsegm.utilities package**

**Submodules**

**imsegm.utilities.data\_io module**

Framework for handling input/output

Copyright (C) 2015-2018 Jiri Borovec <[jiri.borovec@fel.cvut.cz](mailto:jiri.borovec@fel.cvut.cz)>

```
imsegm.utilities.data_io.add_padding(img_size, padding, min_row, min_col, max_row,
                                     max_col)
```

add some padding but still be inside image

**Parameters**

- **img\_size** (*tuple(int, int)*) –

- **padding** (*int*) – set padding around segmented object
- **min\_row** (*int*) – setting top left corner of bounding box
- **min\_col** (*int*) – setting top left corner of bounding box
- **max\_row** (*int*) – setting bottom right corner of bounding box
- **max\_col** (*int*) – setting bottom right corner of bounding box

**Return tuple(int,int,int,int)**

```
>>> add_padding((50, 50), 5, 15, 25, 35, 55)
(10, 20, 40, 50)
```

`imsegm.utilities.data_io.convert_img_2_nifti_gray(path_img, path_out)`  
converting standard image to Nifti format

#### Parameters

- **path\_img** (*str*) – path to input image
- **path\_out** (*str*) – path to output directory

**Return str** path to output image

```
>>> np.random.seed(0)
>>> img = np.random.random((150, 125))
>>> p_in = './temp_sample-image.png'
>>> io.imsave(p_in, img)
>>> p_out = convert_img_2_nifti_gray(p_in, '.')
>>> p_out
'temp_sample-image.nii'
>>> os.remove(p_out)
>>> os.remove(p_in)
```

`imsegm.utilities.data_io.convert_img_2_nifti_rgb(path_img, path_out)`  
converting standard image to Nifti format

#### Parameters

- **path\_img** (*str*) – path to input image
- **path\_out** (*str*) – path to output directory

**Return str** path to output image

```
>>> np.random.seed(0)
>>> p_in = './temp_sample-image.png'
>>> io.imsave(p_in, np.random.random((150, 125, 3)))
>>> p_nifty = convert_img_2_nifti_rgb(p_in, '.')
>>> p_nifty
'temp_sample-image.nii'
>>> os.remove(p_nifty)
>>> os.remove(p_in)
```

`imsegm.utilities.data_io.convert_img_color_from_rgb(image, color_space)`  
convert image colour space from RGB to xxx

#### Parameters

- **image** (*ndarray*) – rgb image
- **color\_space** (*str*) –

**Return ndarray** image

```
>>> convert_img_color_from_rgb(np.ones((50, 75, 3)), 'hsv').shape
(50, 75, 3)
```

`imsegm.utilities.data_io.convert_img_color_to_rgb(image, color_space)`  
convert image colour space to RGB to xxx

**Parameters**

- **image** (*ndarray*) – rgb image
- **color\_space** (*str*) –

**Return ndarray** image

```
>>> convert_img_color_to_rgb(np.ones((50, 75, 3)), 'hsv').shape
(50, 75, 3)
```

`imsegm.utilities.data_io.convert_nifti_2_img(path_img_in, path_img_out)`  
given input and output path convert from nifti to image

**Parameters**

- **path\_img\_in** (*str*) – path to input image
- **path\_img\_out** (*str*) – path to output image

**Return str** path to output image

```
>>> np.random.seed(0)
>>> p_in = './temp_sample-image.png'
>>> io.imsave(p_in, np.random.random((150, 125, 3)))
>>> p_nifty = convert_img_2_nifti_rgb(p_in, '.')
>>> p_nifty
'temp_sample-image.nii'
>>> p_img = convert_nifti_2_img(p_nifty, './temp_sample-image.jpg')
>>> p_img
'./temp_sample-image.jpg'
>>> os.remove(p_nifty)
>>> os.remove(p_img)
>>> os.remove(p_in)
```

`imsegm.utilities.data_io.cut_object(img, mask, padding, use_mask=False, bg_color=None, allow_rotate=True)`  
cut an object from image according binary object segmentation

**Parameters**

- **img** (*ndarray*) – inout image
- **mask** (*ndarray*) – segmentation
- **padding** (*int*) – set padding around segmented object
- **use\_mask** (*bool*) – fill BG values also outside the mask
- **bg\_color** – set as default values outside bounding box
- **allow\_rotate** (*bool*) – allowing rotate object to get minimal bbox

**Returns**



```

>>> img = np.ones((10, 20), dtype=int)
>>> img[3:7, 4:16] = 2
>>> mask = np.zeros((10, 20), dtype=int)
>>> mask[4:6, 5:15] = 1
>>> cut_object(img, mask, 2)
array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1],
       [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1],
       [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1],
       [1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])
>>> cut_object(img, mask, 2, use_mask=True, allow_rotate=False)
array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1],
       [1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])

```

`imsegm.utilities.data_io.export_image(path_img, img, stretch_range=True)`  
 export an image with given path and optional pattern for image name

#### Parameters

- **path\_img** (*str*) – path to the output image
- **img** (*ndarray*) – image `np.array<height, width>`
- **stretch\_range** (*bool*) –

**Return str** path to the image

```

>>> # PNG image
>>> np.random.seed(0)
>>> img = np.random.random([5, 10])
>>> path_img = export_image(os.path.join('.', 'testing-image'), img)
>>> path_img
'./testing-image.png'
>>> os.path.exists(path_img)
True
>>> im, name = load_image_2d(path_img)
>>> im.shape
(5, 10)
>>> im
array([[143, 186, 157, 141, 110, 168, 114, 232, 251, 99],
       [206, 137, 148, 241, 18, 22, 5, 216, 202, 226],
       [255, 208, 120, 203, 30, 166, 37, 246, 135, 108],
       [ 68, 201, 118, 148, 4, 160, 159, 160, 245, 177],
       [ 93, 113, 181, 15, 173, 174, 54, 33, 82, 94]], dtype=uint8)
>>> os.remove(path_img)

```

```

>>> # TIFF image
>>> img = np.random.random([5, 20, 25])
>>> path_img = export_image(os.path.join('.', 'testing-image'), img)
>>> path_img
'./testing-image.tiff'
>>> os.path.exists(path_img)
True
>>> im, name = load_image_2d(path_img)

```

(continues on next page)

(continued from previous page)

```
>>> im.shape
(5, 20, 25)
>>> os.remove(path_img)
```

`imsegm.utilities.data_io.find_files_match_names_across_dirs` (*list\_path\_pattern*, *drop\_none=True*)  
walk over dir with images and segmentation and pair those with the same name and if the folder with centers exists also add to each par a center

---

**Note:** returns just paths

---

### Parameters

- **list\_path\_pattern** (*list(str)*) – list of paths with image name patterns
- **drop\_none** (*bool*) – drop if there are some none - missing values in rows

**Returns** DF<path\_1, path\_2, ...>

```
>>> def _mp(d, n):
...     return os.path.join(update_path('data-images'),
...                           'drosophila_ovary_slice', d, n)
>>> df = find_files_match_names_across_dirs([_mp('image', '*.jpg'),
...                                           _mp('segm', '*.png'),
...                                           _mp('center_levels', '*.csv')])
>>> len(df) > 0
True
>>> df.columns.tolist()
['path_1', 'path_2', 'path_3']
>>> df = find_files_match_names_across_dirs([_mp('image', '*.png'),
...                                           _mp('segm', '*.jpg'),
...                                           _mp('center_levels', '*.csv')])
>>> len(df)
0
```

`imsegm.utilities.data_io.get_image2d_boundary_color` (*image*, *size=1*)  
extract background color as median along image boundaries

### Parameters

- **image** (*ndarray*) –
- **size** (*float*) –

### Returns

```
>>> img = np.zeros((5, 15), dtype=int)
>>> img[:4, 3:9] = 1
>>> img
array([[0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
>>> get_image2d_boundary_color(img)
0
```

(continues on next page)

(continued from previous page)

```
>>> get_image2d_boundary_color(np.ones((5, 15, 3), dtype=int), size=2)
array([1, 1, 1])
>>> get_image2d_boundary_color(np.ones((5, 15, 3, 1), dtype=int))
array(0)
```

`imsegm.utilities.data_io.image_open(path_img)`  
just a wrapper to suppress debug messages from the PIL function

**Parameters** `path_img` (*str*) –

**Return Image**

`imsegm.utilities.data_io.io_image_decorate(func)`  
costume decorator to suppress debug messages from the PIL function to suppress PIL debug logging - DEBUG:PIL.PngImagePlugin:STREAM b'IHDR' 16 13

**Parameters** `func` –

**Returns**

`imsegm.utilities.data_io.io_imread(path_img)`  
just a wrapper to suppress debug messages from the PIL function

**Parameters** `path_img` (*str*) –

**Return ndarray**

`imsegm.utilities.data_io.io_imsave(path_img, img)`  
just a wrapper to suppress debug messages from the PIL function

**Parameters**

- `path_img` (*str*) –
- `img` (*ndarray*) – image

`imsegm.utilities.data_io.load_complete_image_folder(path_dir,`  
`img_name_pattern='*.png',`  
`nb_sample=None,`  
`im_range=255, skip=None)`

load complete image folder with specific name pattern

**Parameters**

- `path_dir` (*str*) – loading dictionary
- `img_name_pattern` (*str*) – image name pattern
- `nb_sample` (*int*) – load just some subset of images
- `im_range` – range to scale image values (1. or 255)
- `skip` (*list(str) | None*) – skip some particular images by name

**Returns**

```
>>> p_imgs = os.path.join(update_path('data-images'),
...                        'drosophila_ovary_slice', 'image')
>>> l_imgs, l_names = load_complete_image_folder(p_imgs, '*.jpg')
>>> len(l_imgs)
5
>>> l_names
['insitu4174', 'insitu4358', 'insitu7331', 'insitu7544', 'insitu7545']
```

```
imsegm.utilities.data_io.load_image(path_img, im_range=255)
```

```
imsegm.utilities.data_io.load_image_2d(path_img)
loading any supported image type
```

**Parameters** `path_img` (*str*) – path to the input image

**Return** `str, ndarray` image name, image as matrix

```
>>> # PNG image
>>> img_name = 'testing-image'
>>> img = np.random.randint(0, 255, size=(20, 20, 3))
>>> path_img = export_image(os.path.join('.', img_name), img,
...                         stretch_range=False)
>>> path_img
'./testing-image.png'
>>> os.path.exists(path_img)
True
>>> img_new, _ = load_image_2d(path_img)
>>> np.array_equal(img, img_new)
True
>>> io.imsave(path_img, np.random.random((50, 65, 4)))
>>> img_new, _ = load_image_2d(path_img)
>>> img_new.shape
(50, 65, 3)
>>> Image.fromarray(np.random.randint(0, 2, (65, 50)), mode='1').save(path_img)
>>> img_new, _ = load_image_2d(path_img)
>>> img_new.shape
(65, 50)
>>> os.remove(path_img)
```

```
>>> # TIFF image
>>> img_name = 'testing-image'
>>> img = np.random.randint(0, 255, size=(5, 20, 20))
>>> path_img = export_image(os.path.join('.', img_name), img,
...                         stretch_range=False)
>>> path_img
'./testing-image.tiff'
>>> os.path.exists(path_img)
True
>>> img_new, _ = load_image_2d(path_img)
>>> img_new.shape
(5, 20, 20)
>>> np.array_equal(img, img_new)
True
>>> os.remove(path_img)
```

```
imsegm.utilities.data_io.load_image_tiff_volume(path_img, im_range=None)
loading TIFF image
```

**Parameters**

- `path_img` (*str*) – path to the input image
- `im_range` (*float*) – range to scale image values (1. or 255)

**Return** `ndarray`

```
>>> p_img = os.path.join(update_path('data-images'), 'drosophila_ovary_3D',
...                       'AU10-13_f0011.tif')
```

(continues on next page)

(continued from previous page)

```
>>> img = load_image_tiff_volume(p_img)
>>> img.shape
(30, 323, 512)
>>> p_img = os.path.join(update_path('data-images'),
...                       'drosophila_ovary_slice', 'image', 'insitu7545.tif')
>>> img = load_image_tiff_volume(p_img)
>>> img.shape
(647, 1024, 3)
```

`imsegm.utilities.data_io.load_images_list` (*path\_imgs*, *im\_range*=255)  
load list of images together with image names

**Parameters**

- **path\_imgs** (*list*(*str*)) – paths to input images
- **im\_range** – range to scale image values (1. or 255)

**Return [ndarray], list(str)**

```
>>> np.random.seed(0)
>>> path_in = './temp_sample-image.png'
>>> io.imsave(path_in, np.random.random((150, 125, 3)))
>>> l_imgs, l_names = load_images_list([path_in, './temp_sample.img'])
>>> l_names
['temp_sample-image']
>>> [img.shape for img in l_imgs]
[(150, 125, 3)]
>>> [img.dtype for img in l_imgs]
[dtype('uint8')]
>>> os.remove(path_in)
>>> path_in = './temp_sample-image.tif'
>>> io.imsave(path_in, np.random.random((150, 125, 3)))
>>> l_imgs, l_names = load_images_list([path_in, './temp_sample.img'])
>>> l_names
['temp_sample-image']
>>> os.remove(path_in)
```

`imsegm.utilities.data_io.load_img_double_band_split` (*path\_img*, *im\_range*=1.0, *quantiles*=(2, 98))  
load image and split channels

**Parameters**

- **path\_img** (*str*) – path to the image
- **im\_range** (*float* | *None*) – range to scale image values (1. or 255)
- **quantiles** (*tuple*(*int*, *int*)) – scale image values in certain percentile range

**Returns**

```
>>> p_imgs = os.path.join(update_path('data-images'), 'drosophila_ovary_slice',
↪ 'image')
>>> p_img = os.path.join(p_imgs, 'insitu7545.jpg')
>>> img_b1, img_b2 = load_img_double_band_split(p_img)
>>> img_b1.shape
(647, 1024)
>>> p_img = os.path.join(p_imgs, 'insitu7545.tif')
>>> img_b1, img_b2 = load_img_double_band_split(p_img)
```

(continues on next page)

(continued from previous page)

```
>>> img_b1.shape
(647, 1024)
>>> p_img = os.path.join(update_path('data-images'),
...                       'drosophila_ovary_3D', 'AU10-13_f0011.tif')
>>> img_b1, img_b2 = load_img_double_band_split(p_img)
>>> img_b1.shape
(15, 323, 512)
```

`imsegm.utilities.data_io.load_landmarks_csv(path_file)`

load the landmarks from a given file of TXT type and return array

**Parameters** `path_file` (*str*) – name of the input file(whole path)

**Return ndarray** array of landmarks of size <nbLandmarks> x 2

```
>>> lnds = np.array([[1, 2], [2, 4], [5, 6]])
>>> fp = save_landmarks_csv('./sample_landmarks.test', lnds)
>>> fp
'./sample_landmarks.csv'
>>> lnds_new = load_landmarks_csv(fp)
>>> np.array_equal(lnds, lnds_new)
True
>>> os.remove(fp)
```

`imsegm.utilities.data_io.load_landmarks_txt(path_file)`

load the landmarks from a given file of TXT type and return array

**Parameters** `path_file` (*str*) – name of the input file(whole path)

**Return ndarray** array of landmarks of size <nbLandmarks> x 2

```
>>> lnds = np.array([[1, 2], [2, 4], [5, 6]])
>>> fp = save_landmarks_txt('./sample_landmarks.test', lnds)
>>> fp
'./sample_landmarks.txt'
>>> lnds_new = load_landmarks_txt(fp)
>>> np.array_equal(lnds, lnds_new)
True
>>> os.remove(fp)
```

`imsegm.utilities.data_io.load_params_from_txt(path_file)`

parse the parameter file which was coded by repr function

**Parameters** `path_file` (*str*) – path to file with parameters

**Return dict**

```
>>> p = {'abc': 123}
>>> path_file = './sample_config.txt'
>>> with open(path_file, 'w') as fp:
...     lines = ['"{}" : {}'.format(k, p[k]) for k in p]
...     _ = fp.write(os.linesep.join(lines)) # it may return nb characters
>>> p2 = load_params_from_txt(path_file)
>>> p2
{'abc': '123'}
>>> os.remove(path_file)
```

`imsegm.utilities.data_io.load_tiff_volume_split_double_band(path_img,`  
`im_range=None)`

load TIFF volume assuming that there are two bands in zip style: c1, c2, c1, c2, c1, ... and split each odd index belong to one of two bands

#### Parameters

- **path\_img** (*str*) – path to the input image
- **im\_range** (*float*) – range to scale image values (1. or 255)

#### Return ndarray, ndarray

```
>>> p_img = os.path.join(update_path('data-images'), 'drosophila_ovary_3D',
...                       'AU10-13_f0011.tif')
>>> img_b1, img_b2 = load_tiff_volume_split_double_band(p_img)
>>> img_b1.shape, img_b2.shape
((15, 323, 512), (15, 323, 512))
>>> p_img = os.path.join(update_path('data-images'),
...                       'drosophila_ovary_slice', 'image', 'insitu7545.tif')
>>> img_b1, img_b2 = load_tiff_volume_split_double_band(p_img)
>>> img_b1.shape, img_b2.shape
((1, 647, 1024), (1, 647, 1024))
>>> img = np.random.randint(0, 255, (1, 3, 250, 200))
>>> p_img = './sample-multistack.tif'
>>> io.imsave(p_img, img)
>>> img_b1, img_b2 = load_tiff_volume_split_double_band(p_img)
>>> img_b1.shape, img_b2.shape
((1, 250, 200), (1, 250, 200))
>>> os.remove(p_img)
```

`imsegm.utilities.data_io.load_zvi_volume_double_band_split` (*path\_img*)  
loading zvi image and split by bands

#### Parameters path\_img (*str*) – path to the image

#### Return ndarray, ndarray

```
>>> p_img = os.path.join(update_path('data-images'), 'others', 'sample.zvi')
>>> img_b1, img_b2 = load_zvi_volume_double_band_split(p_img)
>>> img_b1.shape
(2, 488, 648)
```

`imsegm.utilities.data_io.merge_image_channels` (*img\_ch1, img\_ch2, img\_ch3=None*)  
merge 2 or 3 image channels into single image

#### Parameters

- **img\_ch1** (*ndarray*) – image channel
- **img\_ch2** (*ndarray*) – image channel
- **img\_ch3** (*ndarray*) – image channel

#### Return ndarray

```
>>> np.random.seed(0)
>>> merge_image_channels(np.random.random((150, 125)),
...                      np.random.random((150, 125))).shape
(150, 125, 3)
>>> merge_image_channels(np.random.random((150, 125)),
...                      np.random.random((150, 125)),
...                      np.random.random((150, 125))).shape
(150, 125, 3)
```

`imsegm.utilities.data_io.save_landmarks_csv(path_file, landmarks, dtype=<class 'float'>)`  
save the landmarks into a given file of CSV type

**Parameters**

- **path\_file** (*str*) – fName is name of the input file(whole path)
- **int ]]** **landmarks** (*[int,)*) – array of landmarks of size nb\_landmarks x 2
- **dtype** (*type*) – data type

**Return str** path to output file

`imsegm.utilities.data_io.save_landmarks_txt(path_file, landmarks)`  
save the landmarks into a given file of TXT type

**Parameters**

- **path\_file** (*str*) – name of the input file(whole path)
- **landmarks** – array of landmarks of size nb\_landmarks x 2

**Return str** path to output file

`imsegm.utilities.data_io.scale_image_intensity(img, im_range=1.0, quantiles=(2, 98))`  
scale image values with in give quntile range to filter some outlaiers

**Parameters**

- **img** (*ndarray*) – input image
- **im\_range** – range to scale image values (1. or 255)
- **quantiles** (*tuple(int, int)*) – scale image values in certain quantile range

**Return ndarray**

```
>>> np.random.seed(0)
>>> img = np.random.randint(10, 255, (25, 30))
>>> im = scale_image_intensity(img)
>>> im.min()
0.0
>>> im.max()
1.0
```

`imsegm.utilities.data_io.scale_image_size(path_img, size, path_out=None)`  
load image - scale image - export image on the same path

**Parameters**

- **path\_img** (*str*) – path to the image
- **int]** **size** (*[int,)*) – new image size
- **path\_out** (*str*) – path to output image, if none overwrite the input

**Return str** path to output image

```
>>> np.random.seed(0)
>>> path_in = './test_sample_image.png'
>>> io.imsave(path_in, np.random.random((150, 125, 3)))
>>> path_out = scale_image_size(path_in, [75, 50])
>>> Image.open(path_out).size
(75, 50)
>>> os.remove(path_out)
```



`imsegm.utilities.data_io.scale_image_vals_in_range(img, im_range=1.0)`  
 scale image values in given range

#### Parameters

- **img** (*ndarray*) – input image
- **im\_range** – range to scale image values (1. or 255)

#### Return ndarray

```
>>> np.random.seed(0)
>>> img = np.random.randint(10, 255, (25, 30))
>>> im = scale_image_vals_in_range(img)
>>> im.min()
0.0
>>> im.max()
1.0
```

`imsegm.utilities.data_io.swap_coord_x_y(points)`  
 swap X and Y coordinates in vector of positions

**Parameters** `int[] points` (`[int,)` –

**Return** `[[int, int]]`

```
>>> swap_coord_x_y(np.array([[1, 2], [2, 4], [5, 6]]))
[[2, 1], [4, 2], [6, 5]]
```

`imsegm.utilities.data_io.update_path(path_file, lim_depth=5, absolute=True)`  
 bubble in the folder tree up until it found desired file otherwise return original one

#### Parameters

- **path\_file** (*str*) – path to the input file / folder
- **lim\_depth** (*int*) – maximal depth for going up
- **absolute** (*bool*) – format absolute path

**Return** `str` path to output file / folder

```
>>> path = 'sample_file.test'
>>> f = open(path, 'w')
>>> update_path(path, absolute=False)
'sample_file.test'
>>> os.remove(path)
```

`imsegm.utilities.data_io.COLUMNS_COORDS = ['X', 'Y']`  
 position columns

`imsegm.utilities.data_io.DICT_CONVERT_COLOR_FROM_RGB = {'hed': skimage.color.rgb2hed, 'hsv':`  
 conversion function from RGB color space

`imsegm.utilities.data_io.DICT_CONVERT_COLOR_TO_RGB = {'hed': skimage.color.hed2rgb, 'hsv':`  
 conversion function to RGB color space

## imsegm.utilities.data\_samples module

Sandbox with some sample images

Some images are taken from following sources:

- [ANHIR challenge](<https://anhir.grand-challenge.org>)
- [PhD Thesis](<https://www.researchgate.net/publication/323120618>)

Copyright (C) 2015-2018 Jiri Borovec <jiri.borovec@fel.cvut.cz>

`imsegm.utilities.data_samples.get_image_path` (*name\_img*, *path\_base*=*'/home/docs/checkouts/readthedocs.org/user\_uploads/2018/07/lenapng.png'*)

merge default image path and sample image

### Parameters

- **name\_img** (*str*) –
- **path\_base** (*str*) –

### Return str

```
>>> p = get_image_path(IMAGE_LENNA)
>>> os.path.isfile(p)
True
>>> os.path.basename(p)
'lenna.png'
```

`imsegm.utilities.data_samples.load_sample_image` (*name\_img*=*'others/lenna.png'*)

load sample image

### Parameters **name\_img** (*str*) –

### Return ndarray

```
>>> img = load_sample_image(IMAGE_LENNA)
>>> img.shape
(512, 512, 3)
```

`imsegm.utilities.data_samples.sample_color_image_rand_segment` (*im\_size*=(150, 100), *nb\_classes*=3, *rand\_seed*=None)

create samoe image and segmentation

### Parameters

- **im\_size** (*tuple(int, int)*) –
- **nb\_classes** (*int*) –
- **rand\_seed** –

### Returns

```
>>> im, seg = sample_color_image_rand_segment((5, 6), 2, rand_seed=0)
>>> im.shape
(5, 6, 3)
>>> seg
array([[1, 1, 0, 0, 1, 0],
       [0, 1, 1, 0, 1, 0],
```

(continues on next page)

(continued from previous page)

```
[0, 1, 0, 0, 0, 1],
[1, 0, 1, 0, 0, 0],
[0, 0, 1, 0, 1, 0]])
```

```
imsegm.utilities.data_samples.sample_segment_vertical_2d(seg_size=(20, 10),
                                                         nb_labels=3)
```

create sample segmentation with vertical stripes

#### Parameters

- **seg\_size** (*tuple(int, int)*) –
- **nb\_labels** (*int*) –

#### Return ndarray

```
>>> sample_segment_vertical_2d((7, 5), 2)
array([[0, 0, 0, 1, 1, 1],
       [0, 0, 0, 1, 1, 1],
       [0, 0, 0, 1, 1, 1],
       [0, 0, 0, 1, 1, 1],
       [0, 0, 0, 1, 1, 1],
       [0, 0, 0, 1, 1, 1]])
```

```
imsegm.utilities.data_samples.sample_segment_vertical_3d(seg_size=(10, 5, 6),
                                                         nb_labels=3, levels=2)
```

create sample regular 3D segmentation

#### Parameters

- **int, int) seg\_size** (*tuple(int, int) | (int,)*) –
- **nb\_labels** (*int*) –
- **levels** (*int*) –

#### Return ndarray

```
>>> im = sample_segment_vertical_3d((10, 5, 6), 3)
>>> im[:, :, 3]
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [4, 4, 4, 4, 4],
       [4, 4, 4, 4, 4],
       [4, 4, 4, 4, 4]])
```

```
imsegm.utilities.data_samples.ANNOT_DROSOPHILA_DISC = 'drosophila_disc/annot/img_6.png'
three class tissue annotation of microscopy image of Drosophila imaginal disc
```

```
imsegm.utilities.data_samples.ANNOT_DROSOPHILA_OVARY_2D = 'drosophila_ovary_slice/annot_sti
four class tissue annotation of microscopy image of Drosophila ovary
```

```
imsegm.utilities.data_samples.IMAGE_3CLS = 'synthetic/texture_rgb_3cls.jpg'
sample image with three different textures
```

```
imsegm.utilities.data_samples.IMAGE_DROSOPHILA_DISC = 'drosophila_disc/image/img_6.jpg'
sample microscopy image of Drosophila imaginal disc
```

```
imsegm.utilities.data_samples.IMAGE_DROSOPHILA_OVARY_2D = 'drosophila_ovary_slice/image/in
sample microscopy image of Drosophila ovary
```

```
imsegm.utilities.data_samples.IMAGE_DROSOPHILA_OVARY_3D = 'drosophila_ovary_3D/AU10-13_f000.jpg'
    sample 3D microscopy image of Drosophila ovary
imsegm.utilities.data_samples.IMAGE_HISTOL_CIMA = 'histology_CIMA/29-041-Izd2-w35-CD31-3-1a.jpg'
    sample image of histology tissue from CIMA dataset - ANHIR challenge
imsegm.utilities.data_samples.IMAGE_HISTOL_FLAGSHIP = 'histology_Flagship/Case001_Cytokeratin.jpg'
    sample image of histology tissue
imsegm.utilities.data_samples.IMAGE_LANGER_ISLET = 'langerhans_islets/image/gtExoIsl_21.jpg'
    sample microscopy image of Langerhans islets
imsegm.utilities.data_samples.IMAGE_LENNA = 'others/lenna.png'
    sample Lenna image
imsegm.utilities.data_samples.IMAGE_OBJECTS = 'synthetic/reference.jpg'
    sample image with three color objects of different shape
imsegm.utilities.data_samples.IMAGE_STAR = 'others/sea_starfish-2.jpg'
    sample image of sea starfish
imsegm.utilities.data_samples.LIST_ALL_IMAGES = ['others/lenna.png', 'synthetic/texture_rgb.jpg']
    list of all default images
imsegm.utilities.data_samples.PATH_IMAGES = '/home/docs/checkouts/readthedocs.org/user_builds/imsegm/checkouts/0.1.1/data'
    path to the folder with all sample image/data
imsegm.utilities.data_samples.SAMPLE_SEG_NB_CLASSES = 3
    number of classes for segmentation
imsegm.utilities.data_samples.SAMPLE_SEG_SIZE_2D_NORM = (150, 100)
    image size for normal 2D sample image
imsegm.utilities.data_samples.SAMPLE_SEG_SIZE_2D_SMALL = (20, 10)
    image size for small 2D sample image
imsegm.utilities.data_samples.SAMPLE_SEG_SIZE_3D_SMALL = (10, 5, 6)
    image size for small 3D sample image
```

## **imsegm.utilities.drawing module**

Framework for visualisations

Copyright (C) 2016-2018 Jiri Borovec <jiri.borovec@fel.cvut.cz>

```
imsegm.utilities.drawing._ellipse(r, c, r_radius, c_radius, orientation=0.0, shape=None)
    temporary wrapper until release New version scikit-image v0.13
```

### **Parameters**

- **r** (*int*) – center position in rows
- **c** (*int*) – center position in columns
- **r\_radius** (*int*) – ellipse diam in rows
- **c\_radius** (*int*) – ellipse diam in columns
- **orientation** (*float*) – ellipse orientation
- **shape** (*tuple(int, int)*) – size of output mask

**Return** *tuple(list(int),list(int))* indexes of filled positions

```
>>> img = np.zeros((10, 12), dtype=int)
>>> rr, cc = _ellipse(5, 6, 3, 5, orientation=np.deg2rad(30))
>>> img[rr, cc] = 1
>>> img
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0],
       [0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
       [0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
       [0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0],
       [0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

`imsegm.utilities.drawing.closest_point_on_line` (*start, end, point*)  
projection of the point to the line

#### Parameters

- **start** (*list(int)*) – line starting point
- **end** (*list(int)*) – line ending point
- **point** (*list(int)*) – point for estimation

**Return** `list(int)` point on the line

```
>>> closest_point_on_line([0, 0], [1, 2], [0, 2])
array([ 0.8,  1.6])
```

`imsegm.utilities.drawing.create_figure_by_image` (*img\_size, subfig\_size, nb\_subfigs=1, extend=0.0*)  
creating image according background\_image

#### Parameters

- **img\_size** (*tuple(int, int)*) – image size
- **subfig\_size** (*float*) – maximal sub-figure size
- **nb\_subfigs** (*int*) – number of sub-figure
- **extend** (*float*) – extension

**Return** `tuple(Figure, list)`

`imsegm.utilities.drawing.draw_color_labeling` (*segments, lut\_labels*)  
visualise the graph cut results

#### Parameters

- **segments** (*ndarray*) – `np.array<height, width>`
- **lut\_labels** (*list(int)*) – look-up-table

**Return** `ndarray np.array<height, width, 3>`

`imsegm.utilities.drawing.draw_eggs_ellipse` (*mask\_shape, pos\_ant, pos\_lat, pos\_post, threshold\_overlap=0.6*)  
from given 3 point estimate the ellipse

#### Parameters

- **mask\_shape** (*tuple(int, int)*) –

- **pos\_ant** (*tuple(int, int)*) – anterior
- **pos\_lat** (*tuple(int, int)*) – latitude
- **pos\_post** (*tuple(int, int)*) – postlude
- **threshold\_overlap** (*float*) –

**Return ndarray**

```
>>> pos_ant, pos_lat, pos_post = [10, 10], [20, 20], [35, 20]
>>> points = np.array([pos_ant, pos_lat, pos_post])
>>> _ = plt.plot(points[:, 0], points[:, 1], 'og')
>>> mask = draw_eggs_ellipse([30, 50], [pos_ant], [pos_lat], [pos_post])
>>> mask.shape
(30, 50)
>>> _ = plt.imshow(mask, alpha=0.5, interpolation='nearest')
>>> _ = plt.xlim([0, mask.shape[1]]), plt.ylim([0, mask.shape[0]]), plt.grid()
>>> # plt.show()
```

`imsegm.utilities.drawing.draw_eggs_rectangle` (*mask\_shape, pos\_ant, pos\_lat, pos\_post*)  
from given 3 point estimate the ellipse

**Parameters**

- **mask\_shape** (*tuple(int, int)*) – segmentation size
- **pos\_ant** (*tuple(int, int)*) – points
- **pos\_lat** (*tuple(int, int)*) – points
- **pos\_post** (*tuple(int, int)*) – points

**Return [ndarray]**

```
>>> pos_ant, pos_lat, pos_post = [10, 10], [20, 20], [35, 20]
>>> points = np.array([pos_ant, pos_lat, pos_post])
>>> _ = plt.plot(points[:, 0], points[:, 1], 'og')
>>> masks = draw_eggs_rectangle([30, 50], [pos_ant], [pos_lat], [pos_post])
>>> [m.shape for m in masks]
[(30, 50)]
>>> for mask in masks:
...     _ = plt.imshow(mask, alpha=0.5, interpolation='nearest')
>>> _ = plt.xlim([0, mask.shape[1]]), plt.ylim([0, mask.shape[0]]), plt.grid()
>>> # plt.show()
```

`imsegm.utilities.drawing.draw_graphcut_unary_cost_segments` (*segments, unary\_cost*)  
visualise the unary cost for each class

**Parameters**

- **segments** (*ndarray*) – `np.array<height, width>`
- **unary\_cost** (*ndarray*) – `np.array<nb_spx, nb_classes>`

**Return []** `[np.array<height, width, 3>] * nb_cls`

```
>>> seg = np.random.randint(0, 100, (100, 150))
>>> u_cost = np.random.random((100, 3))
>>> imgs = draw_graphcut_unary_cost_segments(seg, u_cost)
>>> len(imgs)
3
```

(continues on next page)

(continued from previous page)

```
>>> [img.shape for img in imgs]
[(100, 150, 3), (100, 150, 3), (100, 150, 3)]
```

`imsegm.utilities.drawing.draw_graphcut_weighted_edges` (*segments*, *centers*, *edges*, *edge\_weights*, *img\_bg=None*, *img\_alpha=0.5*)

visualise the edges on the overlapping a background image

#### Parameters

- **centers** (*tuple(int, int)*) – list of centers
- **segments** (*ndarray*) – np.array<height, width>
- **edges** (*ndarray*) – list of edges of shape <nb\_edges, 2>
- **edge\_weights** (*ndarray*) – weight per edge <nb\_edges, 1>
- **img\_bg** (*ndarray*) – image background
- **img\_alpha** (*float*) – transparency

**Return ndarray** np.array<height, width, 3>

```
>>> slic = np.array([[0] * 3 + [1] * 3 + [2] * 3 + [3] * 3] * 4 +
...                 [[4] * 3 + [5] * 3 + [6] * 3 + [7] * 3] * 4)
>>> centres = [[1, 1], [1, 4], [1, 7], [1, 10],
...            [5, 1], [5, 4], [5, 7], [5, 10]]
>>> edges = [[0, 1], [1, 2], [2, 3], [0, 4], [1, 5],
...          [4, 5], [2, 6], [5, 6], [3, 7], [6, 7]]
>>> img = np.random.randint(0, 256, slic.shape + (3,))
>>> edge_weights = np.ones(len(edges))
>>> edge_weights[0] = 0
>>> img = draw_graphcut_weighted_edges(slic, centres, edges, edge_weights, img_
↪bg=img)
>>> img.shape
(8, 12, 3)
```

`imsegm.utilities.drawing.draw_image_clusters_centers` (*ax*, *img*, *centres*, *points=None*, *labels\_centre=None*, *segm=None*)

draw image as background and clusters centers

#### Parameters

- **ax** – figure axis
- **img** (*ndarray*) – image
- **centres** (*ndarray*) – points
- **points** (*ndarray*) – optional list of all points
- **labels\_centre** (*list(int)*) – optional list of labels for points
- **segm** (*ndarray*) – optional segmentation

```
>>> img = np.random.randint(0, 256, (100, 100, 3))
>>> seg = np.random.randint(0, 3, (100, 100))
>>> centres = np.random.randint(0, 100, (3, 2))
>>> points = np.random.randint(0, 100, (25, 2))
>>> labels = np.random.randint(0, 4, 25)
```

(continues on next page)

(continued from previous page)

```
>>> draw_image_clusters_centers(plt.Figure().gca(), img[:, :, 0], centres,
...                             points, labels, seg)
```

```
imsegm.utilities.drawing.draw_image_segm_points(ax, img, points, labels=None,
                                                slic=None, color_slic='w',
                                                lut_label_marker={-1: ('.',
                                '#7E7E7E'), 0: ('x', '#7E7E7E'), 1: ('.',
                                '#FFB00')}, seg_contour=None)
```

on plane draw background image or segmentation, overlap with SLIC contours, add contour of adative segmentation like annot. for centers plot point with specific property (shape and colour) according label

#### Parameters

- **ax** – figure axis
- **img** (*ndarray*) – image
- **points** (*list(tuple(int, int))*) – collection of points
- **labels** (*list(int)*) – LUT labels for superpixels
- **slic** (*ndarray*) – superpixel segmentation
- **color\_slic** (*str*) – color dor superpixels
- **lut\_label\_marker** (*dict*) – dictionary {int: (str, str)} of label and markers
- **seg\_contour** (*ndarray*) – segmentation contour

```
>>> img = np.random.randint(0, 256, (100, 100))
>>> points = np.random.randint(0, 100, (25, 2))
>>> labels = np.random.randint(0, 5, len(points))
>>> slic = np.random.randint(0, 256, (100, 100))
>>> draw_image_segm_points(plt.Figure().gca(), img, points, labels, slic)
```

imsegm.utilities.drawing.draw\_rg2sp\_results(ax, seg, slic, debug\_rg2sp, iter\_index=-1)  
drawing Region Growing with shape prior

#### Parameters

- **ax** – figure axis
- **seg** (*ndarray*) – segmentation
- **slic** (*ndarray*) – superpixels
- **debug\_rg2sp** (*dict*) – dictionary with debug results
- **iter\_index** (*int*) – iteration index

#### Returns

ax

imsegm.utilities.drawing.ellipse(r, c, r\_radius, c\_radius, orientation=0.0, shape=None)  
temporary wrapper until release New version scikit-image v0.13

---

**Note:** Should be solved in skimage v0.13

---

#### Parameters



- **r** (*int*) – center position in rows
- **c** (*int*) – center position in columns
- **r\_radius** (*int*) – ellipse diam in rows
- **c\_radius** (*int*) – ellipse diam in columns
- **orientation** (*float*) – ellipse orientation
- **shape** (*tuple(int, int)*) – size of output mask

**Return** *tuple(list(int),list(int))* indexes of filled positions

```
>>> img = np.zeros((14, 20), dtype=int)
>>> rr, cc = ellipse(7, 10, 3, 9, np.deg2rad(30), img.shape)
>>> img[rr, cc] = 1
>>> img
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

`imsegm.utilities.drawing.ellipse_perimeter(r, c, r_radius, c_radius, orientation=0.0, shape=None)`

see New version scikit-image v0.14

---

**Note:** Should be solved in skimage v0.14

---

### Parameters

- **r** (*int*) – center position in rows
- **c** (*int*) – center position in columns
- **r\_radius** (*int*) – ellipse diam in rows
- **c\_radius** (*int*) – ellipse diam in columns
- **orientation** (*float*) – ellipse orientation
- **shape** (*tuple(int, int)*) – size of output mask

**Return** *tuple(list(int),list(int))* indexes of filled positions

```
>>> img = np.zeros((14, 20), dtype=int)
>>> rr, cc = ellipse_perimeter(7, 10, 3, 9, np.deg2rad(30), img.shape)
>>> img[rr, cc] = 1
>>> img
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
```

(continues on next page)

(continued from previous page)

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

`imsegm.utilities.drawing.figure_annot_slic_histogram_labels` (*dict\_label\_hist*,  
*slic\_size=-1*,  
*slic\_regul=-1*)

plot ration of labels assigned to each superpixel

#### Parameters

- **dict\_label\_hist** – dictionary of label name and histogram
- **slic\_size** (*int*) – used for figure title
- **slic\_regul** (*float*) – used for figure title

#### Return Figure

```
>>> np.random.seed(0)
>>> dict_label_hist = {'a': np.tile([1, 0, 0, 0, 1], (25, 1)),
...                   'b': np.tile([0, 1, 0, 0, 1], (30, 1))}
>>> fig = figure_annot_slic_histogram_labels(dict_label_hist)
>>> isinstance(fig, matplotlib.figure.Figure)
True
```

`imsegm.utilities.drawing.figure_ellipse_fitting` (*img*, *seg*, *ellipses*, *centers*, *crits*,  
*fig\_size=9*)

show figure with result of the ellipse fitting

#### Parameters

- **img** (*ndarray*) – image
- **seg** (*ndarray*) – segmentation
- **ellipses** (*list(tuple(int, int, int, int, float))*) – collection of ellipse parameters ell. parameters: (x, y, height, width, orientation)
- **centers** (*list(tuple(int, int))*) – points
- **crits** (*list(float)*) –
- **fig\_size** (*float*) – maximal figure size

#### Return Figure

```
>>> img = np.random.random((100, 150, 3))
>>> seg = np.random.randint(0, 2, (100, 150))
>>> ells = np.random.random((3, 5)) * 25
>>> centers = np.random.random((3, 2)) * 25
```

(continues on next page)

(continued from previous page)

```
>>> crits = np.random.random(3)
>>> fig = figure_ellipse_fitting(img[:, :, 0], seg, ells, centers, crits)
>>> isinstance(fig, matplotlib.figure.Figure)
True
```

`imsegm.utilities.drawing.figure_image_adjustment` (*fig, img\_size*)  
adjust figure as nice image without axis

**Parameters**

- **fig** – Figure
- **img\_size** (*tuple(int, int)*) – image size

**Return Figure**

```
>>> fig = figure_image_adjustment(plt.figure(), (150, 200))
>>> isinstance(fig, matplotlib.figure.Figure)
True
```

`imsegm.utilities.drawing.figure_image_segm_centres` (*img, segm, centers=None, cmap\_contour=<matplotlib.colors.LinearSegmentedColor object>*)

visualise the input image and segmentation in common frame

**Parameters**

- **img** (*ndarray*) – image
- **segm** (*ndarray*) – segmentation
- **centers** (*[tuple(int, int)] | ndarray*) – or `np.array`
- **cmap\_contour** (*obj*) –

**Return Figure**

```
>>> img = np.random.random((100, 150, 3))
>>> seg = np.random.randint(0, 2, (100, 150))
>>> centre = [[55, 60]]
>>> fig = figure_image_segm_centres(img, seg, centre)
>>> isinstance(fig, matplotlib.figure.Figure)
True
```

`imsegm.utilities.drawing.figure_image_segm_results` (*img, seg, subfig\_size=9, mid\_labels\_alpha=0.2, mid\_image\_gray=True*)

creating subfigure with original image, overlapped segmentation contours and clean result segmentation... it turns the sequence in vertical / horizontal according major image dim

**Parameters**

- **img** (*ndarray*) – image as background
- **seg** (*ndarray*) – segmentation
- **subfig\_size** (*int*) – max image size
- **mid\_image\_gray** (*bool*) – used color image as bacround in middle
- **mid\_labels\_alpha** (*float*) – alpha for middle segmentation overlap

**Return Figure**

```
>>> img = np.random.random((100, 150, 3))
>>> seg = np.random.randint(0, 2, (100, 150))
>>> fig = figure_image_segm_results(img, seg)
>>> isinstance(fig, matplotlib.figure.Figure)
True
```

`imsegm.utilities.drawing.figure_overlap_annot_segm_image` (*annot*, *segm*, *img=None*,  
*subfig\_size=9*,  
*drop\_labels=None*,  
*segm\_alpha=0.2*)

figure showing overlap annotation - segmentation - image

#### Parameters

- **annot** (*ndarray*) – user annotation
- **segm** (*ndarray*) – segmentation
- **img** (*ndarray*) – original image
- **subfig\_size** (*int*) – maximal sub-figure size
- **segm\_alpha** (*float*) – use transparency
- **drop\_labels** (*list(int)*) – labels to be ignored

#### Return Figure

```
>>> img = np.random.random((100, 150, 3))
>>> seg = np.random.randint(0, 2, (100, 150))
>>> fig = figure_overlap_annot_segm_image(seg, seg, img, drop_labels=[5])
>>> isinstance(fig, matplotlib.figure.Figure)
True
```

`imsegm.utilities.drawing.figure_ray_feature` (*segm*, *points*, *ray\_dist\_raw=None*,  
*ray\_dist=None*, *points\_reconst=None*,  
*title=""*)

visualise the segmentation with specific point and estimated ray dist.

#### Parameters

- **segm** (*ndarray*) – segmentation
- **points** (*list(float)*) – collection of points
- **ray\_dist\_raw** (*list(float)*) –
- **ray\_dist** (*list(float)*) – Ray feature distances
- **points\_reconst** (*ndarray*) – collection of reconstructed points
- **title** (*str*) – figure title

#### Return Figure

---

**Note:** for more examples, see unittests

---

`imsegm.utilities.drawing.figure_rg2sp_debug_complete` (*seg*, *slic*, *debug\_rg2sp*,  
*iter\_index=-1*, *max\_size=5*)

draw figure with all debug (intermediate) segmentation steps

#### Parameters

- **seg** (*ndarray*) – segmentation
- **slic** (*ndarray*) – superpixels
- **debug\_rg2sp** – dictionary with some debug parameters
- **iter\_index** (*int*) – iteration index
- **max\_size** (*int*) – max figure size

#### Return Figure

```
>>> seg = np.random.randint(0, 4, (100, 150))
>>> slic = np.random.randint(0, 80, (100, 150))
>>> dict_debug = {
...     'lut_data_cost': np.random.random((80, 3)),
...     'lut_shape_cost': np.random.random((15, 80, 3)),
...     'labels': np.random.randint(0, 4, (15, 80)),
...     'centres': [np.array([np.random.randint(0, 100, 80),
...                               np.random.randint(0, 150, 80)]).T] * 15,
...     'shifts': np.random.random((15, 3)),
...     'criteria': np.random.random(15),
... }
>>> fig = figure_rg2sp_debug_complete(seg, slic, dict_debug)
>>> isinstance(fig, matplotlib.figure.Figure)
True
```

`imsegm.utilities.drawing.figure_seg_boundary_dist` (*segm\_ref*, *segm*, *subfig\_size*=9)  
visualise the boundary distances between two segmentation

#### Parameters

- **segm\_ref** (*ndarray*) – reference segmentation
- **segm** (*ndarray*) – estimated segmentation
- **subfig\_size** (*int*) – maximal sub-figure size

#### Return Figure

```
>>> seg = np.zeros((100, 100))
>>> seg[35:80, 10:65] = 1
>>> fig = figure_seg_boundary_dist(seg, seg.T)
>>> isinstance(fig, matplotlib.figure.Figure)
True
```

`imsegm.utilities.drawing.figure_seg_graphcut_debug` (*images*, *subfig\_size*=9)  
creating subfigure with slic, graph edges and results in the first row and individual class unary terms in the second row

#### Parameters

- **images** (*dict*) – dictionary composed from name and image array
- **subfig\_size** (*int*) – maximal sub-figure size

#### Return Figure

```
>>> images = {
...     'image': np.random.random((100, 150, 3)),
...     'slic': np.random.randint(0, 2, (100, 150)),
...     'slic_mean': np.random.random((100, 150, 3)),
...     'img_graph_edges': np.random.random((100, 150, 3)),
... }
```

(continues on next page)

(continued from previous page)

```

...     'img_graph_seg': np.random.random((100, 150, 3)),
...     'imgs_unary_cost': [np.random.random((100, 150, 3))],
... }
>>> fig = figure_seg_graphcut_debug(images)
>>> isinstance(fig, matplotlib.figure.Figure)
True

```

`imsegm.utilities.drawing.figure_used_samples` (*img*, *labels*, *slic*, *used\_samples*, *fig\_size=12*)  
 draw used examples (superpixels)

**Parameters**

- **img** (*ndarray*) – input image for background
- **labels** (*list(int)*) – labels associated for superpixels
- **slic** (*ndarray*) – superpixel segmentation
- **used\_samples** (*list(bool)*) – used samples for training
- **fig\_size** (*int*) – figure size

**Return Figure**

```

>>> img = np.random.random((50, 75, 3))
>>> labels = [-1, 0, 2]
>>> used = [1, 0, 0]
>>> seg = np.random.randint(0, 3, img.shape[:2])
>>> fig = figure_used_samples(img, labels, seg, used)
>>> isinstance(fig, matplotlib.figure.Figure)
True

```

`imsegm.utilities.drawing.make_overlap_images_chess` (*images*, *chess\_field=50*)  
 overlap images and show them

**Parameters**

- **images** (*[ndarray]*) – collection of images
- **chess\_field** (*int*) – size of chess field size

**Return ndarray** combined image

```

>>> im1 = np.zeros((5, 10), dtype=int)
>>> im2 = np.ones((5, 10), dtype=int)
>>> make_overlap_images_chess([im1, im2], chess_field=2)
array([[0, 0, 1, 1, 0, 0, 1, 1, 0, 0],
       [0, 0, 1, 1, 0, 0, 1, 1, 0, 0],
       [1, 1, 0, 0, 1, 1, 0, 0, 1, 1],
       [1, 1, 0, 0, 1, 1, 0, 0, 1, 1],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])

```

`imsegm.utilities.drawing.make_overlap_images_optical` (*images*)  
 overlap images and show them

**Parameters** *images* (*[ndarray]*) – collection of images

**Return ndarray** combined image

```
>>> im1 = np.zeros((5, 8), dtype=float)
>>> im2 = np.ones((5, 8), dtype=float)
>>> make_overlap_images_optical([im1, im2])
array([[ 0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5],
       [ 0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5],
       [ 0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5],
       [ 0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5],
       [ 0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5,  0.5]])
```

`imsegm.utilities.drawing.merge_object_masks` (*masks*, *overlap\_thr*=0.7)  
merge several mask into one multi-class segmentation

#### Parameters

- **masks** (*[ndarray]*) – collection of masks
- **overlap\_thr** (*float*) – threshold for overlap

#### Return ndarray

```
>>> m1 = np.zeros((5, 6), dtype=int)
>>> m1[:4, :4] = 1
>>> m2 = np.zeros((5, 6), dtype=int)
>>> m2[2:, 2:] = 1
>>> merge_object_masks([m1, m1])
array([[1, 1, 1, 1, 0, 0],
       [1, 1, 1, 1, 0, 0],
       [1, 1, 1, 1, 0, 0],
       [1, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 0, 0]])
>>> merge_object_masks([m1, m2])
array([[1, 1, 1, 1, 0, 0],
       [1, 1, 1, 1, 0, 0],
       [1, 1, 2, 2, 2, 2],
       [1, 1, 2, 2, 2, 2],
       [0, 0, 2, 2, 2, 2]])
```

`imsegm.utilities.drawing.norm_aplha` (*alpha*)  
normalise alpha in range (0, 1)

#### Parameters **alpha** (*float*) –

#### Return float

```
>>> norm_aplha(0.5)
0.5
>>> norm_aplha(255)
1.0
>>> norm_aplha(-1)
0
```

`imsegm.utilities.drawing.parse_annot_rectangles` (*rows\_slice*)  
parse annotation fromDF to lists

#### Parameters **rows\_slice** – a row from a table

#### Return tuple the three points

```
>>> import pandas as pd
>>> dict_row = dict(ant_x=1, ant_y=2, lat_x=3, lat_y=4, post_x=5, post_y=6)
```

(continues on next page)

(continued from previous page)

```

>>> row = pd.DataFrame([dict_row])
>>> parse_annot_rectangles(row)
[(1, 2)], [(3, 4)], [(5, 6)]]
>>> rows = pd.DataFrame([dict_row, {n: dict_row[n] + 10 for n in dict_row}])
>>> rows
   ant_x  ant_y  lat_x  lat_y  post_x  post_y
0      1     2      3     4        5        6
1     11     12     13     14       15       16
>>> parse_annot_rectangles(rows)
[(1, 2), (11, 12)], [(3, 4), (13, 14)], [(5, 6), (15, 16)]]

```

`imsegm.utilities.drawing.COLUMNS_POSITION_EGG_ANNOT = ('ant_x', 'ant_y', 'post_x', 'post_y')`  
columns from description files which marks the egg annotation by expert

`imsegm.utilities.drawing.DICT_LABEL_MARKER = {-1: ('.', '#7E7E7E'), 0: ('x', '#7E7E7E'), 1: ('x', '#7E7E7E')}`  
define markers for labels of positive (+1) neutral (0) and negative (-1) class

`imsegm.utilities.drawing.SIZE_CHESS_FIELD = 50`  
for blending two images define chess field size in pixels

## imsegm.utilities.experiments module

Framework for general experiments

Copyright (C) 2014-2018 Jiri Borovec <jiri.borovec@fel.cvut.cz>

**class** `imsegm.utilities.experiments.Experiment` (*params*, *time\_stamp=True*)  
Bases: `object`  
Basic experiment class

### Example

```

>>> import shutil
>>> params = {'path_out': './my_experiments', 'name': 'My-Sample'}
>>> expt = Experiment(params)
Traceback (most recent call last):
...
Exception: given folder "./my_experiments" does not exist!
>>> os.mkdir(params['path_out'])
>>> expt = Experiment(params, time_stamp=False)
>>> expt.run()
>>> params = expt.params.copy()
>>> del expt
>>> shutil.rmtree(params['path_out'], ignore_errors=True)

```

constructor

#### Parameters

- **params** (*dict*) – define experiment parameters
- **time\_stamp** (*bool*) – add to experiment unique time stamp

**\_check\_exist\_paths** ()

Check all required paths in parameters whether they exist



**\_create\_folder** (*time\_stamp=True*)

Create the experiment folder and iterate while there is no available

Parameters **time\_stamp** (*bool*) – mark if you want an unique folder per experiment

**\_evaluate** ()

**\_load\_data** (*gt=True*)

loading the experiment data

Parameters **gt** (*bool*) – try to load ground truth

**\_perform** ()

**\_summarise** ()

**run** (*gt=True*)

run the main Experimental body

Parameters **gt** (*bool*) – try to load Ground Truth

```
class imsegm.utilities.experiments.WrapExecuteSequence (wrap_func, iterate_vals,
                                                    nb_workers=2, desc="",
                                                    ordered=False)
```

Bases: `object`

wrapper for execution parallel of single thread as for ...

## Example

```
>>> it = WrapExecuteSequence(lambda x: (x, x ** 2), range(5),
...                          nb_workers=1, ordered=True)
>>> list(it)
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16)]
>>> it = WrapExecuteSequence(sum, [[0, 1]] * 5, nb_workers=2, desc=None)
>>> [o for o in it]
[1, 1, 1, 1, 1]
>>> it = WrapExecuteSequence(min, ([0, 1] for i in range(5)))
>>> [o for o in it]
[0, 0, 0, 0, 0]
```

the init of this wrapper for parallelism

### Parameters

- **wrap\_func** – function which will be executed in the iterations
- **iterate\_vals** (*[]*) – list or iterator which will iterate in iterations
- **nb\_workers** (*int*) – number of jobs running in parallel
- **desc** (*str*) – description for the bar, if it is set None, bar is suppressed
- **ordered** (*bool*) – whether enforce ordering in the parallelism

```
imsegm.utilities.experiments.append_final_stat (out_dir, y_true, y_pred, time_sec,
                                                file_name='resultStat.txt')
```

append (export) statistic to existing default file

### Parameters

- **out\_dir** (*str*) –
- **y\_true** (*[int]*) – annotation

- **y\_pred** (*int*) – predictions
- **time\_sec** (*int*) –
- **file\_name** (*str*) –

**Return str**

```
>>> import numpy as np
>>> np.random.seed(0)
>>> y_true = np.random.randint(0, 2, 25)
>>> y_pred = np.random.randint(0, 2, 25)
>>> f_path = append_final_stat('.', y_true, y_pred, 256)
>>> os.path.exists(f_path)
True
>>> os.remove(f_path)
```

`imsegm.utilities.experiments.create_experiment_folder` (*params*, *dir\_name*,  
*stamp\_unique=True*,  
*skip\_load=True*)

create the experiment folder and iterate while there is no available

**Parameters**

- **params** (*dict*) – configuration
- **dir\_name** (*str*) – folder name
- **stamp\_unique** (*bool*) – use unique timestamp
- **skip\_load** (*bool*) – skip loading folder params

**Return dict**

```
>>> import shutil
>>> import pandas as pd
>>> p = {'path_out': '.'}
>>> p = create_experiment_folder(p, 'my_test', False, skip_load=True)
>>> pd.Series(p).sort_index()
computer      [...]
path_exp      ./my_test_EXAMPLE
path_out      .
dtype: object
>>> p = create_experiment_folder(p, 'my_test', False, skip_load=False)
>>> shutil.rmtree(p['path_exp'], ignore_errors=True)
>>> p = create_experiment_folder(p, 'my_test', stamp_unique=True)
>>> pd.Series(p).sort_index()
computer      [...]
path_exp      ./my_test_EXAMPLE_...-...
path_out      .
dtype: object
>>> shutil.rmtree(p['path_exp'], ignore_errors=True)
```

`imsegm.utilities.experiments.create_subfolders` (*path\_out*, *folders*)  
create subfolders in rood directory

**Parameters**

- **path\_out** (*str*) – root dictionary
- **folders** (*list (str)*) – list of subfolders

**Return int**

```
>>> import shutil
>>> dir_name = 'sample_dir'
>>> create_subfolders('.', [dir_name])
1
>>> os.path.exists(dir_name)
True
>>> shutil.rmtree(dir_name, ignore_errors=True)
```

`imsegm.utilities.experiments.extend_list_params(params, name_param, options)`  
 extend the parameter list by all sub-datasets

#### Parameters

- **params** (`list(dict)`) – list of parameters
- **name\_param** (`str`) – parameter name
- **options** (`[]`) – lost of options

#### Return list(dict)

```
>>> import pandas as pd
>>> params = extend_list_params([{'a': 1}], 'a', [3, 4])
>>> pd.DataFrame(params) [sorted(pd.DataFrame(params))]
  a param_idx
0  3      a-2#1
1  4      a-2#2
>>> params = extend_list_params([{'a': 1}], 'b', 5)
>>> pd.DataFrame(params) [sorted(pd.DataFrame(params))]
  a  b param_idx
0  1  5      b-1#1
```

`imsegm.utilities.experiments.is_iterable(var)`  
 check if the variable is iterable

#### Parameters var –

#### Return bool

```
>>> is_iterable('abc')
False
>>> is_iterable([0])
True
>>> is_iterable((1, ))
True
```

`imsegm.utilities.experiments.load_config_yaml(path_config)`  
 loading the

#### Parameters path\_config(str) –

#### Return dict

```
>>> p_conf = './testing-congif.yaml'
>>> save_config_yaml(p_conf, {'a': 2})
>>> load_config_yaml(p_conf)
{'a': 2}
>>> os.remove(p_conf)
```

`imsegm.utilities.experiments.nb_workers(ratio)`  
 get fraction of of available CPUs

**Parameters** **ratio** (*float*) – range (0, 1)

**Return** **int** number of workers with lower bound 1

```
>>> nb_workers(0)
1
```

`imsegm.utilities.experiments.save_config_yaml (path_config, config)`  
exporting configuration as YAML file

**Parameters**

- **path\_config** (*str*) –
- **config** (*dict*) –

`imsegm.utilities.experiments.set_experiment_logger (path_out, file_name='logging.txt', reset=True)`  
set the logger to file

`imsegm.utilities.experiments.string_dict (d, offset=30, desc='DICTIONARY')`  
transform dictionary to a formatted string

**Parameters**

- **d** (*dict*) –
- **offset** (*int*) – length between name and value
- **desc** (*str*) – dictionary title

**Return** **str**

```
>>> string_dict({'abc': 123})
'DICTIONARY: \n"abc": 123'
```

`imsegm.utilities.experiments.try_decorator (func)`  
costume decorator to wrap function in try/except

**Parameters** **func** –

**Returns**

`imsegm.utilities.experiments.CONFIG_YAML = 'config.yml'`  
default file for loading/exporting experiment configuration

`imsegm.utilities.experiments.CPU_COUNT = 2`  
total number of available CPUs/threads

`imsegm.utilities.experiments.FILE_LOGS = 'logging.txt'`  
default file for streaming experimeny messages

`imsegm.utilities.experiments.FORMAT_DT = '%Y%m%d-%H%M%S'`  
default date-time format

`imsegm.utilities.experiments.RESULTS_TXT = 'resultStat.txt'`  
default name of file for exporting statistics

**imsegm.utilities.read\_zvi module**

<https://searchcode.com/codesearch/view/40141634/>

read ZVI (Zeiss) image file

- incomplete support
- open uncompressed image from multi item image (Count>0)
- require `OleFileIO_PL` - a Python module to read MS OLE2 files <http://www.decalage.info/en/python/olefileio#attachments>

```
>>> import os, sys
>>> sys.path += [os.path.abspath(os.path.join('.', '..'))]
>>> import imsegm.utilities.data_io as tl_io
>>> path_file = os.path.join('data-images', 'others', 'sample.zvi')
>>> path_file = tl_io.update_path(path_file)
>>> n = get_layer_count(path_file)
>>> get_dir(path_file)
[...]
>>> for p in range(n):
...     zvi = zvi_read(path_file, p)
...     arr = zvi.Image.Array
...     arr.shape
(488, 648)
(488, 648)
(488, 648)
(488, 648)
>>> img = load_image(path_file)
>>> img.shape
(4, 488, 648)
```

```
class imsegm.utilities.read_zvi.ImageTuple (Version, Width, Height, Depth, PixelWidth,
                                             PIXEL_FORMAT, ValidBitsPerPixel, Array)

    Bases: tuple

    Create new instance of ImageTuple(Version, Width, Height, Depth, PixelWidth, PIXEL_FORMAT, Valid-
    BitsPerPixel, Array)

    _asdict ()
        Return a new OrderedDict which maps field names to their values.

    classmethod _make (iterable, new=<built-in method __new__ of type object>, len=<built-in func-
                       tion len>)
        Make a new ImageTuple object from a sequence or iterable

    _replace (**kws)
        Return a new ImageTuple object replacing specified fields with new values

    property Array
        Alias for field number 7

    property Depth
        Alias for field number 3

    property Height
        Alias for field number 2
```

```
property PIXEL_FORMAT
    Alias for field number 5

property PixelWidth
    Alias for field number 4

property ValidBitsPerPixel
    Alias for field number 6

property Version
    Alias for field number 0

property Width
    Alias for field number 1

_fields = ('Version', 'Width', 'Height', 'Depth', 'PixelWidth', 'PIXEL_FORMAT', 'ValidBitsPerPixel')
_source = "from builtins import property as _property, tuple as _tuple\nfrom operator import attrgetter"

class imsegm.utilities.read_zvi.ZviImageTuple (Version, FileName, Width, Height, Depth,
                                              PIXEL_FORMAT, Count, ValidBitsPerPixel, m_PluginCLSID, Others, Layers,
                                              Scaling)

    Bases: tuple

    Create new instance of ZviImageTuple(Version, FileName, Width, Height, Depth, PIXEL_FORMAT, Count,
    ValidBitsPerPixel, m_PluginCLSID, Others, Layers, Scaling)

    _asdict ()
        Return a new OrderedDict which maps field names to their values.

    classmethod _make (iterable, new=<built-in method __new__ of type object>, len=<built-in function len>)
        Make a new ZviImageTuple object from a sequence or iterable

    _replace (**kwargs)
        Return a new ZviImageTuple object replacing specified fields with new values

property Count
    Alias for field number 6

property Depth
    Alias for field number 4

property FileName
    Alias for field number 1

property Height
    Alias for field number 3

property Layers
    Alias for field number 10

property Others
    Alias for field number 9

property PIXEL_FORMAT
    Alias for field number 5

property Scaling
    Alias for field number 11

property ValidBitsPerPixel
    Alias for field number 7
```

**property Version**

Alias for field number 0

**property Width**

Alias for field number 2

**\_fields** = ('Version', 'FileName', 'Width', 'Height', 'Depth', 'PIXEL\_FORMAT', 'Count',**\_source** = "from builtins import property as \_property, tuple as \_tuple\nfrom operator**property m\_PluginCLSID**

Alias for field number 8

```
class imsegm.utilities.read_zvi.ZviItemTuple (Version, FileName, Width, Height, Depth,
                                             PIXEL_FORMAT, Count, ValidBitsPer-
                                             Pixel, Others, Layers, Scaling, Image)
```

Bases: `tuple`

Create new instance of ZviItemTuple(Version, FileName, Width, Height, Depth, PIXEL\_FORMAT, Count, ValidBitsPerPixel, Others, Layers, Scaling, Image)

**\_\_asdict** ()

Return a new OrderedDict which maps field names to their values.

**classmethod** **\_\_make** (iterable, new=<built-in method \_\_new\_\_ of type object>, len=<built-in function len>)

Make a new ZviItemTuple object from a sequence or iterable

**\_\_replace** (\*\*kwds)

Return a new ZviItemTuple object replacing specified fields with new values

**property Count**

Alias for field number 6

**property Depth**

Alias for field number 4

**property FileName**

Alias for field number 1

**property Height**

Alias for field number 3

**property Image**

Alias for field number 11

**property Layers**

Alias for field number 9

**property Others**

Alias for field number 8

**property PIXEL\_FORMAT**

Alias for field number 5

**property Scaling**

Alias for field number 10

**property ValidBitsPerPixel**

Alias for field number 7

**property Version**

Alias for field number 0

**property Width**

Alias for field number 2

**\_fields** = ('Version', 'FileName', 'Width', 'Height', 'Depth', 'PIXEL\_FORMAT', 'Count',**\_source** = "from builtins import property as \_property, tuple as \_tuple\nfrom operator`imsegm.utilities.read_zvi.get_dir(file_name, ole=None)`

returns the content structure(streams) of the zvi file + length of each streams

`imsegm.utilities.read_zvi.get_hex(data, n=16)``imsegm.utilities.read_zvi.get_layer_count(file_name, ole=None)`

returns the number of image planes

`imsegm.utilities.read_zvi.i32(data)`

return int32 from len4 string

`imsegm.utilities.read_zvi.load_image(path_img)``imsegm.utilities.read_zvi.parse_image(data)`

returns ImageTuple from raw image data(header+image)

`imsegm.utilities.read_zvi.read_image_container_content(stream)`

returns a ZviImageTuple from a stream

`imsegm.utilities.read_zvi.read_item_storage_content(stream)`

returns ZviItemTuple from the stream

`imsegm.utilities.read_zvi.read_struct(data, t)`

read a t type from data(str)

`imsegm.utilities.read_zvi.zvi_read(fname, plane, ole=None)`

returns ZviItemTuple of the plane from zvi file fname

## Module contents

### 1.2.2 Submodules

#### imsegm.annotation module

Framework for handling annotations

Copyright (C) 2014-2018 Jiri Borovec &lt;jiri.borovec@fel.cvut.cz&gt;

`imsegm.annotation.convert_img_colors_to_labels(img_rgb, lut_label_color)`

take a RGB image and dictionary of labels and apply this dictionary it returns relabels image according given dictionary

**Parameters**

- **img\_rgb** (*ndarray*) – np.array<height, width, 3> input RGB image
- **{int – (int, int, int)}** lut\_label\_color:

**Return ndarray** np.array<height, width> labeling

```
>>> np.random.seed(0)
>>> seg = np.random.randint(0, 2, (5, 7))
>>> img = np.array([(0.2, 0.2, 0.2), (0.9, 0.9, 0.9)]) [seg]
>>> d_lb_clr = {0: (0.2, 0.2, 0.2), 1: (0.9, 0.9, 0.9)}
```

(continues on next page)



(continued from previous page)

```
>>> convert_img_colors_to_labels(img, d_lb_clr)
array([[0, 1, 1, 0, 1, 1, 1],
       [1, 1, 1, 1, 0, 0, 1],
       [0, 0, 0, 0, 0, 1, 0],
       [1, 1, 0, 0, 1, 1, 1],
       [1, 0, 1, 0, 1, 0, 1]])
```

`imsegm.annotation.convert_img_colors_to_labels_reverted`(*img\_rgb*,  
*dict\_color\_label*)

take a RGB image and dictionary of labels and apply this dictionary it returns relabels image according given dictionary

#### Parameters

- **img\_rgb** (*ndarray*) – np.array<height, width, 3> input RGB image
- **int, int** (*{(int,) – int}*) dict\_color\_label:

**Return ndarray** np.array<height, width> labeling

```
>>> np.random.seed(0)
>>> seg = np.random.randint(0, 2, (5, 7))
>>> img = np.array([(0.2, 0.2, 0.2), (0.9, 0.9, 0.9)])[seg]
>>> d_clr_lb = {(0.2, 0.2, 0.2): 0, (0.9, 0.9, 0.9): 1}
>>> convert_img_colors_to_labels_reverted(img, d_clr_lb)
array([[0, 1, 1, 0, 1, 1, 1],
       [1, 1, 1, 1, 0, 0, 1],
       [0, 0, 0, 0, 0, 1, 0],
       [1, 1, 0, 0, 1, 1, 1],
       [1, 0, 1, 0, 1, 0, 1]])
```

`imsegm.annotation.convert_img_labels_to_colors`(*segm*, *lut\_label\_colors*)

convert labeling according given dictionary of colors

#### Parameters

- **segm** (*ndarray*) – np.array<height, width>
- **{int – (int, int, int)}** lut\_label\_colors:

**Return ndarray** np.array<height, width, 3>

```
>>> np.random.seed(0)
>>> seg = np.random.randint(0, 2, (5, 7))
>>> d_lb_clr = {0: (0.2, 0.2, 0.2), 1: (0.9, 0.9, 0.9)}
>>> img = convert_img_labels_to_colors(seg, d_lb_clr)
>>> img[:, :, 0]
array([[ 0.2,  0.9,  0.9,  0.2,  0.9,  0.9,  0.9],
       [ 0.9,  0.9,  0.9,  0.9,  0.2,  0.2,  0.9],
       [ 0.2,  0.2,  0.2,  0.2,  0.2,  0.9,  0.2],
       [ 0.9,  0.9,  0.2,  0.2,  0.9,  0.9,  0.9],
       [ 0.9,  0.2,  0.9,  0.2,  0.9,  0.2,  0.9]])
```

`imsegm.annotation.group_images_frequent_colors`(*paths\_img*, *ratio\_threshold=0.001*)

look all images and estimate most frequent colours

#### Parameters

- **paths\_img** (*list(str)*) – path to images
- **ratio\_threshold** (*float*) – percentage of nb, clr pixels to be assumed as important

**Return list(int)**

```
>>> from skimage import data
>>> from imsegm.utilities.data_io import io_imsave
>>> path_img = './sample-image.png'
>>> io_imsave(path_img, data.astronaut())
>>> d_clrs = group_images_frequent_colors([path_img], ratio_threshold=3e-4)
>>> sorted([d_clrs[c] for c in d_clrs], reverse=True)
[27969, 1345, 1237, 822, 450, 324, 313, 244, 229, 213, 163, 160, 158, 157,
 150, 137, 120, 119, 117, 114, 98, 92, 92, 91, 81]
>>> os.remove(path_img)
```

`imsegm.annotation.image_color_2_labels` (*img*, *colors=None*)  
quantize input image according given list of possible colours

**Parameters**

- **img** (*ndarray*) – `np.array<height, width, 3>`, input image
- **int, int)] colors** (*[int,)*) – list of possible colours

**Return ndarray** `np.array<height, width>`

```
>>> np.random.seed(0)
>>> rand = np.random.randint(0, 2, (5, 7)).astype(np.uint8)
>>> img = np.rollaxis(np.array([rand] * 3), 0, 3)
>>> image_color_2_labels(img)
array([[1, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 1, 1, 0],
       [1, 1, 1, 1, 1, 0, 1],
       [0, 0, 1, 1, 0, 0, 0],
       [0, 1, 0, 1, 0, 1, 0]])
```

`imsegm.annotation.image_frequent_colors` (*img*, *ratio\_threshold=0.001*)  
look all images and estimate most frequent colours

**Parameters**

- **img** (*ndarray*) – `np.array<height, width, 3>`
- **ratio\_threshold** (*float*) – percentage of nb color pixels to be assumed as important

**Return {(int, int, int) int}**

```
>>> np.random.seed(0)
>>> img = np.random.randint(0, 2, (50, 50, 3)).astype(np.uint8)
>>> d = image_frequent_colors(img)
>>> sorted(d.keys())
[(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1),
 (1, 1, 0), (1, 1, 1)]
>>> sorted(d.values())
[271, 289, 295, 317, 318, 330, 335, 345]
```

`imsegm.annotation.image_inpaint_pixels` (*img*, *valid\_mask*)

`imsegm.annotation.load_info_group_by_slices` (*path\_txt*, *stages*, *pos\_columns=('ant\_x', 'ant\_y', 'post\_x', 'post\_y', 'lat\_x', 'lat\_y')*, *dict\_slice\_tol={1: 1, 2: 2, 3: 2, 4: 3, 5: 3, 6: 0}*)

load all info and group position info according name if stack

**Parameters**

- `path_txt (str)` –
- `stages (list (int))` – stages
- `pos_columns (list (str))` –
- `dict_slice_tol (dict (list (int)))` – mapping of int to list

Returns DF

```
>>> from imsegm.utilities.data_io import update_path
>>> path_txt = os.path.join(update_path('data-images'),
...                          'drosophila_ovary_slice', 'info_ovary_images.txt')
>>> load_info_group_by_slices(path_txt, [4])
      ant_x  ant_y  lat_x  lat_y  post_x  post_y
image
insitu7569  [298]  [327]  [673]  [411]  [986]  [155]
```

`imsegm.annotation.quantize_image_nearest_color (img, colors)`  
quantize input image according given list of possible colours

Parameters

- `img (ndarray)` – np.array<height, width, 3>, input image
- `int, int)] colors ([ (int,)` – list of possible colours

Return ndarray np.array<height, width, 3>

```
>>> np.random.seed(0)
>>> img = np.random.randint(0, 2, (5, 7, 3)).astype(np.uint8)
>>> im = quantize_image_nearest_color(img, [(0, 0, 0), (1, 1, 1)])
>>> im[:, :, 0]
array([[1, 1, 1, 1, 0, 0, 0],
       [1, 1, 1, 1, 1, 1, 0],
       [1, 1, 0, 1, 1, 0, 1],
       [0, 0, 1, 0, 1, 0, 1],
       [1, 1, 1, 0, 1, 0, 0]], dtype=uint8)
>>> [np.array_equal(im[:, :, 0], im[:, :, i]) for i in [1, 2]]
[True, True]
```

`imsegm.annotation.quantize_image_nearest_pixel (img, colors)`  
quantize input image according given list of possible colours

Parameters

- `img (ndarray)` – np.array<height, width, 3>, input image
- `int, int)] colors ([ (int,)` – list of possible colours

Return ndarray np.array<height, width, 3>

```
>>> np.random.seed(0)
>>> img = np.random.randint(0, 2, (5, 7, 3)).astype(np.uint8)
>>> im = quantize_image_nearest_pixel(img, [(0, 0, 0), (1, 1, 1)])
>>> im[:, :, 0]
array([[1, 1, 1, 1, 0, 0, 0],
       [1, 1, 1, 1, 0, 0, 0],
       [1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0, 0]])
>>> [np.array_equal(im[:, :, 0], im[:, :, i]) for i in [1, 2]]
[True, True]
```

`imsegm.annotation.unique_image_colors`

find all unique color in image and return its list

**Parameters** `img` (*ndarray*) – np.array<height, width, 3>

**Returns** [(int, int, int)]

```
>>> np.random.seed(0)
>>> img = np.random.randint(0, 2, (50, 50, 3))
>>> unique_image_colors(img)
[(1, 0, 0), (1, 1, 0), (0, 1, 0), (1, 1, 1),
 (0, 1, 1), (0, 0, 1), (1, 0, 1), (0, 0, 0)]
>>> img = np.random.randint(0, 256, (150, 150, 3))
>>> unique_image_colors(img)
[...]
```

`imsegm.annotation.ANNOT_SLICE_DIST_TOL = {1: 1, 2: 2, 3: 2, 4: 3, 5: 3, 6: 0}`  
set distance in Z axis whether near slice may still belong to the same egg

`imsegm.annotation.COLUMNS_POSITION = ('ant_x', 'ant_y', 'post_x', 'post_y', 'lat_x', 'lat_y')`  
names of annotated columns

`imsegm.annotation.DICT_COLOURS = {0: (0, 0, 255), 1: (255, 0, 0), 2: (0, 255, 0), 3: (255, 0, 255)}`  
default colors for particular label

## imsegm.classification module

Supporting file to create and set parameters for scikit-learn classifiers and some preprocessing functions that support classification

Copyright (C) 2014-2018 Jiri Borovec <jiri.borovec@fel.cvut.cz>

**class** `imsegm.classification.CrossValidate` (*nb\_samples, nb\_hold\_out, rand\_seed=None, ignore\_overflow=0.01*)

Bases: `object`

Cross-validator generator. In the hold-out, the data is split only once into a train set and a test set.

### Parameters

- **nb\_samples** (*integer, total number of samples*)–
- **nb\_hold\_out** (*integer, number of samples hold out*)–
- **rand\_seed** (*seed for the random number generator*)–
- **ignore\_overflow** (*float, tolerance while dividing dataset to folds*)–

## Examples

```
>>> # balanced split
>>> cv = CrossValidate(6, 3, rand_seed=False)
>>> cv.indexes
[0, 1, 2, 3, 4, 5]
>>> len(cv)
2
>>> list(cv)
[( [3, 4, 5], [0, 1, 2] ),
```

(continues on next page)

(continued from previous page)

```

([0, 1, 2], [3, 4, 5]])
>>> [(len(tr), len(ts)) for tr, ts in CrossValidate(340, 0.41)]
[(201, 139), (201, 139), (201, 139)]

```

```

>>> # not rounded split
>>> cv = CrossValidate(7, 3, rand_seed=0)
>>> list(cv)
[[[3, 0, 5, 4], [6, 2, 1]],
 [6, 2, 1, 4], [3, 0, 5]],
 [1, 3, 0, 5], [4, 6, 2]]]
>>> len(cv)
3
>>> cv.indexes
[6, 2, 1, 3, 0, 5, 4]

```

```

>>> # larger test then train
>>> cv = CrossValidate(7, 5, rand_seed=0)
>>> list(cv)
[[[6, 2], [1, 3, 0, 5, 4]],
 [1, 3], [6, 2, 0, 5, 4]],
 [0, 5], [6, 2, 1, 3, 4]],
 [4, 6], [2, 1, 3, 0, 5]]]
>>> [(len(tr), len(ts)) for tr, ts in CrossValidate(340, 0.55)]
[(153, 187), (153, 187), (153, 187)]

```

```

>>> # impact of tolerance
>>> len(CrossValidate(340, 0.33, ignore_overflow=0.0))
4
>>> len(CrossValidate(340, 0.33, ignore_overflow=0.05))
3

```

```

>>> [(len(tr), len(ts)) for tr, ts in CrossValidate(4651, 0.25, ignore_overflow=0.
↪)]
[(3488, 1163), (3488, 1163), (3488, 1163), (3488, 1163)]
>>> [(len(tr), len(ts)) for tr, ts in CrossValidate(4651, 0.25, ignore_
↪overflow=1e-2)]
[(3488, 1163), (3488, 1163), (3488, 1163), (3489, 1162)]

```

constructor

#### Parameters

- **nb\_samples** (*int*) – list of sizes
- **nb\_hold\_out** (*int/float*) – how much hold out
- **rand\_seed** (*int/None*) – random seed for shuffling
- **ignore\_overflow** (*float*) – tolerance while dividing dataset to folds

**\_CrossValidate\_\_steps** ()  
adjust this iterator, tol\_balance

**Return list(int)** indexes of steps

```

class imsegm.classification.CrossValidateGroups (set_sizes,          nb_hold_out,
                                                  rand_seed=None,      ig-
                                                  nore_overflow=0.01)

```

Bases: *imsegm.classification.CrossValidate*

Cross-validator generator. In the hold-out, the data is split only once into a train set and a test set.

#### Parameters

- **set\_sizes** (*list of integers, number of samples in each set*)–
- **nb\_hold\_out** (*integer, number of sets hold out*)–
- **rand\_seed** (*seed for the random number generator*)–
- **ignore\_overflow** (*float, tolerance while dividing dataset to folds*)–

#### Examples

```
>>> # balance split
>>> cv = CrossValidateGroups([2, 3, 2, 3], 2, rand_seed=False)
>>> cv.set_indexes
[[0, 1], [2, 3, 4], [5, 6], [7, 8, 9]]
>>> len(cv)
2
>>> list(cv)
[[[5, 6, 7, 8, 9], [0, 1, 2, 3, 4]],
 [[0, 1, 2, 3, 4], [5, 6, 7, 8, 9]]]
>>> [(len(tr), len(ts)) for tr, ts in CrossValidateGroups([7] * 340, 0.41)]
[(1407, 973), (1407, 973), (1407, 973)]
```

```
>>> # unbalanced split
>>> cv = CrossValidateGroups([2, 2, 1, 2, 1], 2, rand_seed=0)
>>> cv.set_indexes
[[0, 1], [2, 3], [4], [5, 6], [7]]
>>> list(cv)
[[[2, 3, 5, 6, 7], [4, 0, 1]],
 [[4, 0, 1, 7], [2, 3, 5, 6]],
 [[0, 1, 2, 3, 5, 6], [7, 4]]]
>>> len(cv)
3
>>> cv.indexes
[2, 0, 1, 3, 4]
```

```
>>> # larger test then train
>>> cv = CrossValidateGroups([2, 2, 1, 2, 1, 1], 4, rand_seed=0)
>>> list(cv)
[[[8, 4], [2, 3, 5, 6, 0, 1, 7]],
 [[2, 3, 5, 6], [8, 4, 0, 1, 7]],
 [[0, 1, 7], [8, 4, 2, 3, 5, 6]]]
>>> [(len(tr), len(ts)) for tr, ts in CrossValidateGroups([7] * 340, 0.55)]
[(1071, 1309), (1071, 1309), (1071, 1309)]
```

construct

#### Parameters

- **set\_sizes** (*list(int)*) – list of sizes
- **nb\_hold\_out** (*int/float*) – how much hold out
- **rand\_seed** (*int/None*) – random seed for shuffling
- **ignore\_overflow** (*float*) – tolerance while dividing dataset to folds

**`_CrossValidateGroups__iter_indexes`** (*sets*)

return enrol indexes from sets

**Parameters** *sets* (*list(int)*) – selection of indexes

**Return** *list(int)*

**class** `imsegm.classification.HoldOut` (*nb\_samples*, *hold\_out*, *rand\_seed=0*)

Bases: `object`

Hold-out cross-validator generator. In the hold-out, the data is split only once into a train set and a test set. Unlike in other cross-validation schemes, the hold-out consists of only one iteration.

**Parameters**

- **`nb_samples`** (*int*, *total number of samples*) –
- **`hold_out`** (*int*, *number where the test starts*) –
- **`rand_seed`** (*seed for the random number generator*) –

### Example

```
>>> ho = HoldOut(10, 7, rand_seed=None)
>>> len(ho)
1
>>> list(ho)
[[0, 1, 2, 3, 4, 5, 6], [7, 8, 9]]
>>> ho = HoldOut(10, 7, rand_seed=0)
>>> list(ho)
[[2, 8, 4, 9, 1, 6, 7], [3, 0, 5]]
```

constructor

**Parameters**

- **`nb_samples`** (*int*) – total number of samples
- **`hold_out`** (*int*) – index where the test starts
- **`rand_seed`** (*obj*) – Seed for the random number generator.

`imsegm.classification.balance_dataset_by` (*features*, *labels*, *balance\_type='random'*,  
*min\_samples=None*)

balance number of training examples per class by several method

**Parameters**

- **`features`** (*ndarray*) – features in dimension *nb\_samples* x *nb\_features*
- **`labels`** (*list(int)*) – annotation for samples
- **`balance_type`** (*str*) – type of balancing dataset
- **`min_samples`** (*int/None*) – if *None* take the smallest class

**Return** *tuple(ndarray, ndarray)*

```
>>> np.random.seed(0)
>>> fts, lbs = balance_dataset_by_(np.random.random((25, 3)),
...                               np.random.randint(0, 2, 25))
>>> fts.shape
(24, 3)
```

(continues on next page)

(continued from previous page)

```
>>> lbs
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

`imsegm.classification.compose_dict_label_features` (*features*, *labels*)  
convert vector of features and related labels to a dictionary of features where key is the labels

**Parameters**

- **features** (*ndarray*) – features in dimension `nb_samples x nb_features`
- **labels** (*list(int)*) – annotation for samples

**Return** {*int ndarray*}: {*int*: `np.array<nb, nb_features>`}

`imsegm.classification.compute_classif_metrics` (*y\_true*, *y\_pred*, *metric\_averages*=('macro', 'weighted'))  
compute standard metrics for multi-class classification

**Parameters**

- **y\_true** (*list(int)*) –
- **y\_pred** (*list(int)*) –
- **metric\_averages** (*str/list(str)*) –

**Return** dict(float)

```
>>> np.random.seed(0)
>>> y_true = np.random.randint(0, 3, 25) * 2
>>> y_pred = np.random.randint(0, 2, 25) * 2
>>> d = compute_classif_metrics(y_true, y_true)
>>> d['accuracy']
1.0
>>> d['confusion']
[[10, 0, 0], [0, 10, 0], [0, 0, 5]]
>>> d = compute_classif_metrics(y_true, y_pred)
>>> d['accuracy']
0.32...
>>> d['confusion']
[[3, 7, 0], [5, 5, 0], [1, 4, 0]]
>>> d = compute_classif_metrics(y_pred, y_pred)
>>> d['accuracy']
1.0
```

`imsegm.classification.compute_classif_stat_segmm_annot` (*annot\_segmm\_name*, *drop\_labels*=None, *relabel*=False)  
compute classification statistic between annotation and segmentation

**Parameters**

- **annot\_segmm\_name** (*tuple(ndarray, ndarray, str)*) –
- **drop\_labels** (*list(int)*) – labels to be ignored
- **relabel** (*bool*) – whether relabel

**Returns**



```

>>> np.random.seed(0)
>>> annot = np.random.randint(0, 2, (5, 10))
>>> segm = np.random.randint(0, 2, (5, 10))
>>> d = compute_classif_stat_seg_annot((annot, annot, 'ttt'), relabel=True,
...                                  drop_labels=[5])
>>> d['(FP+FN)/(TP+FN)']
0.0
>>> d['(TP+FP)/(TP+FN)']
1.0
>>> d = compute_classif_stat_seg_annot((annot, segm, 'ttt'), relabel=True,
...                                  drop_labels=[5])
>>> d['(FP+FN)/(TP+FN)']
0.846...
>>> d['(TP+FP)/(TP+FN)']
1.153...
>>> d = compute_classif_stat_seg_annot((annot, segm + 1, 'ttt'),
...                                  relabel=False, drop_labels=[0])
>>> d['confusion']
[[13, 17], [0, 0]]

```

`imsegm.classification.compute_metric_fpf_n_tpf_n(annot, segm, label_positive=None)`  
compute measure  $(FP + FN) / (TP + FN)$

#### Parameters

- **annot** (*ndarray*) – annotation
- **segm** (*ndarray*) – segmentation
- **label\_positive** (*int*) – indexes of positive labels

#### Return float

```

>>> np.random.seed(0)
>>> annot = np.random.randint(0, 2, (50, 75)) * 3
>>> segm = np.random.randint(0, 2, (50, 75)) * 3
>>> compute_metric_fpf_n_tpf_n(annot, segm)
1.02...
>>> compute_metric_fpf_n_tpf_n(annot, annot)
0.0
>>> compute_metric_fpf_n_tpf_n(annot, np.ones((50, 75)))
nan

```

`imsegm.classification.compute_metric_tpf_n_tpf_n(annot, segm, label_positive=None)`  
compute measure  $(TP + FP) / (TP + FN)$

#### Parameters

- **annot** (*ndarray*) –
- **segm** (*ndarray*) –
- **label\_positive** (*int*) –

#### Return float

```

>>> np.random.seed(0)
>>> annot = np.random.randint(0, 2, (50, 75)) * 3
>>> segm = np.random.randint(0, 2, (50, 75)) * 3
>>> compute_metric_tpf_n_tpf_n(annot, segm)
1.03...

```

(continues on next page)

(continued from previous page)

```
>>> compute_metric_tpf_tpf_n(annot, annot)
1.0
>>> compute_metric_tpf_tpf_n(annot, np.ones((50, 75)))
nan
>>> compute_metric_tpf_tpf_n(annot, np.zeros((50, 75)))
0.0
```

`imsegm.classification.compute_stat_per_image` (*segms*, *annots*, *names=None*,  
*nb\_workers=2*, *drop\_labels=None*, *re-*  
*label=False*)

compute statistic over multiple segmentations with annotation

#### Parameters

- **segms** (*ndarray*) – segmntations
- **annots** (*ndarray*) – annotations
- **names** (*list(str)*) – list of names
- **drop\_labels** (*list(int)*) – labels to be ignored
- **relabel** (*bool*) – whether relabel
- **nb\_workers** (*int*) – running jobs in parallel

#### Return DF

```
>>> np.random.seed(0)
>>> img_true = np.random.randint(0, 3, (50, 100))
>>> img_pred = np.random.randint(0, 2, (50, 100))
>>> df = compute_stat_per_image([img_true], [img_true], nb_workers=2,
↪relabel=True)
>>> from pprint import pprint
>>> pprint(pd.Series(df.iloc[0]).sort_index().to_dict())
{'ARS': 1.0,
 'accuracy': 1.0,
 'confusion': [[1672, 0, 0], [0, 1682, 0], [0, 0, 1646]],
 'f1_macro': 1.0,
 'precision_macro': 1.0,
 'recall_macro': 1.0,
 'support_macro': None}
>>> df = compute_stat_per_image([img_true], [img_pred], drop_labels=[-1])
>>> pd.Series(df.round(4).iloc[0]).sort_index()
ARS                                0.0002
accuracy                          0.3384
confusion      [[836, 826, 770], [836, 856, 876], [0, 0, 0]]
f1_macro                                0.2701
precision_macro                      0.3363
recall_macro                         0.2257
support_macro                        None
Name: 0, dtype: object
```

`imsegm.classification.compute_tp_tn_fp_fn` (*annot*, *segm*, *label\_positive=None*)  
compute measure TruePositive, TrueNegative, FalsePositive, FalseNegative

#### Parameters

- **annot** (*ndarray*) – annotation
- **segm** (*ndarray*) – segmentation

- **label\_positive**(*int*) – indexes of positive labels

**Return tuple(float,float,float,float)**

```
>>> np.random.seed(0)
>>> annot = np.random.randint(0, 2, (5, 7)) * 9
>>> segm = np.random.randint(0, 2, (5, 7)) * 9
>>> annot - segm
array([[ -9,  9,  0, -9,  9,  9,  0],
       [ 9,  0,  0,  0, -9, -9,  9],
       [-9,  0, -9, -9, -9,  0,  0],
       [ 0,  9,  0, -9,  0,  9,  0],
       [ 9, -9,  9,  0,  9,  0,  9]])
>>> compute_tp_tn_fp_fn(annot, annot)
(20, 15, 0, 0)
>>> compute_tp_tn_fp_fn(annot, segm)
(9, 5, 11, 10)
>>> compute_tp_tn_fp_fn(annot, np.ones((5, 7)))
(nan, nan, nan, nan)
>>> compute_tp_tn_fp_fn(np.zeros((5, 7)), np.zeros((5, 7)))
(35, 0, 0, 0)
```

`imsegm.classification.convert_dict_label_features_2_vectors` (*dict\_features*)  
convert dictionary of features where key is the labels to vector of all features and related labels

**Parameters** {*int* – [list(float)]} *dict\_features*: {*int*: [list(float) \* *nb\_features*] \* *nb\_samples*}

**Return tuple(ndarray,list(int))** *np.array*<*nb\_samples*, *nb\_features*>, *list*(*int*)

`imsegm.classification.convert_set_features_labels_2_dataset` (*imgs\_features*,  
*imgs\_labels*,  
*drop\_labels=None*,  
*balance\_type=None*)  
with dictionary for each image we concentrate all features over images and labels into simple form

**Parameters**

- {*str* – *ndarray*} *imgs\_features*: dictionary of name and features
- {*str* – *ndarray*} *imgs\_labels*: dictionary of name and labels
- **drop\_labels** (*list* (*int*)) – labels to be ignored
- **balance\_type** (*bool*) – whether *balance\_type* number of sampler per class

**Return tuple(ndarray,ndarray,ndarray)**

```
>>> np.random.seed(0)
>>> d_fts = {'a': np.random.random((25, 3)),
...         'b': np.random.random((30, 3)), }
>>> d_lbs = {'a': np.random.randint(0, 2, 25),
...         'b': np.random.randint(0, 2, 30)}
>>> fts, lbs, sizes = convert_set_features_labels_2_dataset(d_fts, d_lbs)
>>> fts.shape
(55, 3)
>>> lbs.shape
(55,)
>>> sizes
[25, 30]
```

```
imsegm.classification.create_classif_search(name_clf,      clf_pipeline,      nb_labels,
                                           search_type='random',  cross_val=10,
                                           eval_metric='f1',      nb_iter=250,
                                           nb_workers=5)
```

create sklearn search depending on spec. random or grid

#### Parameters

- **nb\_labels** (*int*) – number of labels
- **search\_type** (*str*) – hyper-params search type
- **eval\_metric** (*str*) – evaluation metric
- **nb\_iter** (*int*) – for random number of tries
- **name\_clf** (*str*) – name of classif.
- **clf\_pipeline** (*obj*) – object
- **cross\_val** (*obj*) – obj specific CV for fix train-test
- **nb\_workers** (*int*) – number jobs running in parallel

#### Returns

```
imsegm.classification.create_classif_search_train_export(clf_name,      features,
                                                         labels,      cross_val=10,
                                                         nb_search_iter=100,
                                                         search_type='random',
                                                         eval_metric='f1',
                                                         nb_workers=1,
                                                         path_out=None,
                                                         params=None,
                                                         pca_coef=0.98,      fea-
                                                         ture_names=None,
                                                         label_names=None)
```

create classifier and train it once or find best parameters. whether tha path out is given export it for later use

#### Parameters

- **clf\_name** (*str*) – name of selected classifier
- **features** (*ndarray*) – features in dimension nb\_samples x nb\_features
- **labels** (*list(int)*) – annotation for samples
- **cross\_val** (*int|obj*) – Cross validation
- **search\_type** (*str*) – search type
- **eval\_metric** (*str*) – evaluation metric
- **params** (*dict*) – extra parameters
- **pca\_coef** (*float*) – sklearn PCA - int/float/None
- **nb\_search\_iter** (*int*) – number of searcher for hyper-parameters
- **path\_out** (*str*) – path to directory for exporting classifier
- **nb\_workers** (*int*) – parallel processes
- **feature\_names** (*list(str)*) – list of extracted features - names
- **label\_names** (*list(str)*) – list of label names

**Returns** (obj, str): classifier, path to the exported classifier

```
>>> np.random.seed(0)
>>> lbs = np.random.randint(0, 3, 150)
>>> fts = np.random.random((150, 5)) + np.tile(lbs, (5, 1)).T
>>> _, _ = create_classif_search_train_export('LogistRegr', fts, lbs, nb_search_
↪iter=0)
>>> clf, p_clf = create_classif_search_train_export('AdaBoost', fts, lbs,
...         nb_search_iter=2, path_out='', search_type='grid')
Fitting ...
>>> clf
Pipeline(...)
>>> clf, p_clf = create_classif_search_train_export('RandForest', fts, lbs,
...         nb_search_iter=2, path_out='.', search_type='random')
Fitting ...
>>> clf
Pipeline(...)
>>> p_clf
'./classifier_RandForest.pkl'
>>> os.remove(p_clf)
>>> import glob
>>> files = glob.glob(os.path.join('.', 'classif_*.txt'))
>>> sorted(files)
['./classif_RandForest_search_params_best.txt',
 './classif_RandForest_search_params_scores.txt']
>>> for p in files: os.remove(p)
```

`imsegm.classification.create_classifiers` (*nb\_workers=-1*)  
create all classifiers with default parameters

**Parameters** `nb_workers` (*int*) – number of parallel if possible

**Return dict** {str: clf}

```
>>> classifs = create_classifiers()
>>> classifs
{...}
>>> sum([isinstance(create_clf_param_search_grid(k), dict)
...      for k in classifs.keys()])
7
>>> sum([isinstance(create_clf_param_search_distrib(k), dict)
...      for k in classifs.keys()])
7
```

`imsegm.classification.create_clf_param_search_distrib` (*name\_classif='RandForest'*)  
create parameter distribution for random search

**Parameters** `name_classif` (*str*) – key name of classifier

**Returns** dict

```
>>> create_clf_param_search_distrib()
{...}
>>> dict_classif = create_classifiers()
>>> all(len(create_clf_param_search_distrib(k)) > 0 for k in dict_classif)
True
>>> create_clf_param_search_distrib('none')
{}
```

`imsegm.classification.create_clf_param_search_grid(name_classif='RandForest')`  
create parameter grid for search

**Parameters** `name_classif` (*str*) – key name of selected classifier

**Returns** dict

```
>>> create_clf_param_search_grid('RandForest')
{'classif__...': ...}
>>> dict_classif = create_classifiers()
>>> all(len(create_clf_param_search_grid(k)) > 0 for k in dict_classif)
True
>>> create_clf_param_search_grid('none')
{}
```

`imsegm.classification.create_clf_pipeline(name_classif='RandForest', pca_coef=0.95)`  
create complete pipeline with all required steps

**Parameters**

- `pca_coef` (*int/float/None*) – sklearn PCA
- `name_classif` (*str*) – key name of classif.

**Returns** object

```
>>> create_clf_pipeline()
Pipeline(...)
```

`imsegm.classification.create_pipeline_neuron_net()`  
create classifier for simple neuronal network

**Returns** clf

```
>>> create_pipeline_neuron_net()
Pipeline(...)
```

`imsegm.classification.down_sample_dict_features_kmean(dict_features, nb_samples)`  
cluser with kmeans the features with nb cluster == given nb\_samples and the retirn features which are closer to each cluster center

**Parameters**

- `dict_features` (*dict*) – {int: [list(float) \* nb\_features] \* nb}
- `nb_samples` (*int*) –

**Return dict** {int: [list(float) \* nb\_features] \* nb\_samples}

```
>>> np.random.seed(0)
>>> d_fts = {'a': np.random.random((100, 3))}
>>> d_fts = down_sample_dict_features_kmean(d_fts, 5)
>>> d_fts['a'].shape
(5, 3)
```

`imsegm.classification.down_sample_dict_features_random(dict_features, nb_samples)`  
browse all label features and take random subset of features to have given nb\_samples per class

**Parameters**

- `dict_features` (*dict*) – {int: [list(float) \* nb\_features] \* nb}
- `nb_samples` (*int*) –

**Return dict** {int: [list(float) \* nb\_features] \* nb\_samples}

```
>>> np.random.seed(0)
>>> d_fts = {'a': np.random.random((100, 3))}
>>> d_fts = down_sample_dict_features_random(d_fts, 5)
>>> d_fts['a'].shape
(5, 3)
```

`imsegm.classification.down_sample_dict_features_unique(dict_features)`  
browse all label features and take unique features

**Parameters** `dict_features` (*dict*) – {int: [list(float) \* nb\_features] \* nb\_samples}

**Return dict** {int: [list(float) \* nb\_features] \* nb}

```
>>> np.random.seed(0)
>>> d_fts = {'a': np.random.random((100, 3))}
>>> d_fts = down_sample_dict_features_unique(d_fts)
>>> d_fts['a'].shape
(100, 3)
```

`imsegm.classification.eval_classif_cross_val_roc(clf_name, classif, features, labels, cross_val, path_out=None, nb_steps=100)`

compute mean ROC curve on cross-validation schema

[http://scikit-learn.org/0.15/auto\\_examples/plot\\_roc\\_crossval.html](http://scikit-learn.org/0.15/auto_examples/plot_roc_crossval.html)

#### Parameters

- **clf\_name** (*str*) – name of selected classifier
- **classif** (*obj*) – sklearn classifier
- **features** (*ndarray*) – features in dimension nb\_samples x nb\_features
- **labels** (*list(int)*) – annotation for samples
- **cross\_val** (*object*) –
- **path\_out** (*str*) – path for exporting statistic
- **nb\_steps** (*int*) – number of thresholds

#### Returns

```
>>> np.random.seed(0)
>>> labels = np.array([0] * 150 + [1] * 100 + [3] * 50)
>>> data = np.tile(labels, (6, 1)).T.astype(float)
>>> data += np.random.random(data.shape)
>>> data.shape
(300, 6)
>>> from sklearn.model_selection import StratifiedKFold
>>> cv = StratifiedKFold(n_splits=5, random_state=0, shuffle=True)
>>> classif = create_classifiers()[DEFAULT_CLASSIF_NAME]
>>> fp_tp, auc = eval_classif_cross_val_roc(DEFAULT_CLASSIF_NAME, classif, data,
→ labels, cv, nb_steps=11)
>>> fp_tp
      FP    TP
0    0.0    0.0
1    0.1    1.0
2    0.2    1.0
```

(continues on next page)

(continued from previous page)

```

3   0.3  1.0
4   0.4  1.0
5   0.5  1.0
6   0.6  1.0
7   0.7  1.0
8   0.8  1.0
9   0.9  1.0
10  1.0  1.0
>>> auc
0.94...
>>> labels[-50:] -= 1
>>> data[-50:, :] -= 1
>>> path_out = 'temp_eval-cv-roc'
>>> os.mkdir(path_out)
>>> fp_tp, auc = eval_classif_cross_val_roc(
...     DEFAULT_CLASSIF_NAME, classif, data, labels, cv, nb_steps=5, path_
    ↪out=path_out)
>>> fp_tp
      FP   TP
0  0.00  0.0
1  0.25  1.0
2  0.50  1.0
3  0.75  1.0
4  1.00  1.0
>>> auc
0.875
>>> import shutil
>>> shutil.rmtree(path_out, ignore_errors=True)

```

`imsegm.classification.eval_classif_cross_val_scores` (*clf\_name*, *classif*, *features*, *labels*, *cross\_val=10*, *path\_out=None*, *scorings=('f1\_macro', 'accuracy', 'precision\_macro', 'recall\_macro')*)

compute statistic on cross-validation schema

[http://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.cross\\_val\\_score.html](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html)

#### Parameters

- **clf\_name** (*str*) – name of selected classifier
- **classif** (*obj*) – sklearn classifier
- **features** (*ndarray*) – features in dimension `nb_samples x nb_features`
- **labels** (*list(int)*) – annotation for samples
- **cross\_val** (*object*) –
- **path\_out** (*str*) – path for exporting statistic
- **scorings** (*list(str)*) – list of used scorings

#### Return DF

```

>>> labels = np.array([0] * 150 + [1] * 100 + [2] * 50)
>>> data = np.tile(labels, (6, 1)).T.astype(float)
>>> data += 0.5 - np.random.random(data.shape)

```

(continues on next page)



(continued from previous page)

```

>>> data.shape
(300, 6)
>>> from sklearn.model_selection import StratifiedKFold
>>> cv = StratifiedKFold(n_splits=5, random_state=0, shuffle=True)
>>> classif = create_classifiers()[DEFAULT_CLASSIF_NAME]
>>> df = eval_classif_cross_val_scores(DEFAULT_CLASSIF_NAME, classif, data,
↳ labels, cv)
>>> df.round(decimals=1)
   fl_macro  accuracy  precision_macro  recall_macro
0         1.0         1.0             1.0           1.0
1         1.0         1.0             1.0           1.0
2         1.0         1.0             1.0           1.0
3         1.0         1.0             1.0           1.0
4         1.0         1.0             1.0           1.0
>>> labels[labels == 1] = 2
>>> cv = StratifiedKFold(n_splits=3, random_state=0, shuffle=True)
>>> df = eval_classif_cross_val_scores(DEFAULT_CLASSIF_NAME, classif, data,
↳ labels, cv, path_out='.')
>>> df.round(decimals=1)
   fl_macro  accuracy  precision_macro  recall_macro
0         1.0         1.0             1.0           1.0
1         1.0         1.0             1.0           1.0
2         1.0         1.0             1.0           1.0
>>> import glob
>>> p_files = glob.glob(NAME_CSV_CLASSIF_CV_SCORES.replace('{}', '*'))
>>> sorted(p_files)
['classif_RandForest_cross-val_scores-all-folds.csv',
 'classif_RandForest_cross-val_scores-statistic.csv']
>>> [os.remove(p) for p in p_files]
[...]
```

`imsegm.classification.export_results_clf_search` (*path\_out*, *clf\_name*, *clf\_search*)

do the final testing and save all results

#### Parameters

- **path\_out** (*str*) – path to directory for exporting classifier
- **clf\_name** (*str*) – name of selected classifier
- **clf\_search** (*object*) –

`imsegm.classification.feature_scoring_selection` (*features*, *labels*, *names=None*, *path\_out=""*)

find the best features and retrain the indexes [http://scikit-learn.org/stable/auto\\_examples/linear\\_model/plot\\_sparse\\_recovery.html](http://scikit-learn.org/stable/auto_examples/linear_model/plot_sparse_recovery.html) [http://scikit-learn.org/stable/auto\\_examples/feature\\_selection/plot\\_feature\\_selection.html](http://scikit-learn.org/stable/auto_examples/feature_selection/plot_feature_selection.html)

#### Parameters

- **features** (*ndarray*) – `np.array<nb_samples, nb_features>`
- **labels** (*ndarray*) – `np.array<nb_samples, 1>`
- **names** (*list(str)*) –
- **path\_out** (*str*) –

**Return tuple(list(int),DF)** indices, Dataframe with scoring

```

>>> from sklearn.datasets import make_classification
>>> features, labels = make_classification(n_samples=250, n_features=5,
...                                     n_informative=3, n_redundant=0,
...                                     n_repeated=0, n_classes=2,
...                                     random_state=0, shuffle=False)
>>> indices, df_scoring = feature_scoring_selection(features, labels)
>>> indices
array([1, 0, 2, 3, 4])
>>> df_scoring
      ExtTree      F-test      k-Best variance
feature
1      0.24...    0.75...    0.75...    2.49...
2      0.33...   58.94...   58.94...    1.85...
3      0.22...    2.24...    2.24...    1.54...
4      0.10...    4.02...    4.02...    0.96...
5      0.09...    0.02...    0.02...    1.01...
>>> features[:, 2] = 1
>>> path_out = 'test_fts-select'
>>> os.mkdir(path_out)
>>> indices, df_scoring = feature_scoring_selection(features.tolist(), labels.
->tolist(), path_out=path_out)
>>> indices
array([1, 0, 3, 4, 2])
>>> import shutil
>>> shutil.rmtree(path_out, ignore_errors=True)

```

`imsegm.classification.load_classifier(path_classif)`

estimate classifier for all data and export it

**Parameters** `path_classif` (*str*) – path to the exported classifier

**Return dict**

```
>>> load_classifier('none.abc')
```

`imsegm.classification.relabel_sequential(labels, uq_labels=None)`

relabel sequential vector staring from 0

**Parameters**

- **labels** (*list(int)*) – all labels
- **uq\_labels** (*list(int)*) – unique labels

**Return []**

```

>>> relabel_sequential([0, 0, 0, 5, 5, 5, 0, 5])
[0, 0, 0, 1, 1, 1, 0, 1]

```

`imsegm.classification.save_classifier(path_out, classif, clf_name, params, feature_names=None, label_names=None)`

estimate classif for all data and export it

**Parameters**

- **path\_out** (*str*) – path for exporting trained classofier
- **classif** – sklearn classif.
- **clf\_name** (*str*) – name of selected classifier
- **feature\_names** (*list(str)*) – list of string names

- **params** (*dict*) – extra parameters
- **label\_names** (*list (str)*) – list of string names of label\_names

**Return str**

```
>>> clf = create_classifiers()['RandForest']
>>> p_clf = save_classifier('.', clf, 'TESTINNG', {})
>>> p_clf
'./classifier_TESTINNG.pkl'
>>> d_clf = load_classifier(p_clf)
>>> sorted(d_clf.keys())
['clf_pipeline', 'features', 'label_names', 'name', 'params']
>>> d_clf['clf_pipeline']
RandomForestClassifier(...)
>>> d_clf['name']
'TESTINNG'
>>> os.remove(p_clf)
```

`imsegm.classification.search_params_cut_down_max_nb_iter` (*clf\_parameters, nb\_iter*)  
create parameters list and count number of possible combination in case they are they are limited

**Parameters**

- **clf\_parameters** (*dict*) – dictionary with parameters
- **nb\_iter** (*int*) – nb of random tries

**Return int**

```
>>> clf_params = create_clf_param_search_grid(DEFAULT_CLASSIF_NAME)
>>> search_params_cut_down_max_nb_iter(clf_params, 100)
100
>>> search_params_cut_down_max_nb_iter(clf_params, 1e6)
1450
```

`imsegm.classification.shuffle_features_labels` (*features, labels*)  
take the set of features and labels and shuffle them together while keeping link between feature and its label

**Parameters**

- **features** (*ndarray*) – features in dimension nb\_samples x nb\_features
- **labels** (*list (int)*) – annotation for samples

**Returns** `np.array<nb_samples, nb_features>`, `np.array<nb_samples>`

```
>>> np.random.seed(0)
>>> fts = np.random.random((5, 2))
>>> lbs = np.random.randint(0, 2, 5)
>>> fts_new, lbs_new = shuffle_features_labels(fts, lbs)
>>> np.array_equal(fts, fts_new)
False
>>> np.array_equal(lbs, lbs_new)
False
```

`imsegm.classification.unique_rows` (*data*)  
with matrix detect unique row and return only them

**Parameters** **data** (*ndarray*) – np.array

**Return ndarray** `np.array`

`imsegm.classification.DEFAULT_CLASSIF_NAME = 'RandForest'`  
default (recommended) classifier for supervised segmentation

`imsegm.classification.DEFAULT_CLUSTERING = 'kMeans'`  
default (recommended) clustering for unsupervised segmentation

`imsegm.classification.DICT_SCORING = {'accuracy': sklearn.metrics.accuracy_score, 'f1': sk`  
mapping of metrics names to used functions

`imsegm.classification.METRIC_AVERAGES = ('macro', 'weighted')`  
default types of computed metrics

`imsegm.classification.METRIC_SCORING = ('f1_macro', 'accuracy', 'precision_macro', 'recall`  
default computed metrics

`imsegm.classification.NAME_CSV_CLASSIF_CV_ROC = 'classif_{}_cross-val_ROC-{}.csv'`  
exporting partial results about trained classifier - Receiver Operating Characteristics

`imsegm.classification.NAME_CSV_CLASSIF_CV_SCORES = 'classif_{}_cross-val_scores-{}.csv'`  
exporting partial results about trained classifier

`imsegm.classification.NAME_CSV_FEATURES_SELECT = 'feature_selection.csv'`  
file name of exported evaluation on feature quality

`imsegm.classification.NAME_TXT_CLASSIF_CV_AUC = 'classif_{}_cross-val_AUC-{}.txt'`  
exporting partial results about trained classifier - Area Under Curve

`imsegm.classification.NB_WORKERS_SERACH = 1`  
default number of workers

`imsegm.classification.ROUND_UNIQUE_FTS_DIGITS = 3`  
rounding unique features, in case to detail precision

`imsegm.classification.TEMPLATE_NAME_CLF = 'classifier_{}.pkl'`  
name template forexporting trained classifier (adding classifier name and version)

## imsegm.descriptors module

### Framework for feature extraction

- color and gray 3D images
- color and texture features
- Ray features
- label histogram

Copyright (C) 2014-2018 Jiri Borovec <jiri.borovec@fel.cvut.cz>

`imsegm.descriptors._check_color_image` (*image*)  
verify proper image

**Parameters** *image* (*ndarray*) –

**Return** *bool*

```
>>> _check_color_image(np.zeros((200, 250, 1)))
Traceback (most recent call last):
...
ValueError: image is not RGB with dims (200, 250, 1)
```

`imsegm.descriptors._check_color_image_seg`*m*(*image*, *segm*)  
verify image - segmentation compatibility

#### Parameters

- **image** (*ndarray*) – image
- **segm** (*ndarray*) – segmentation

#### Return bool

```
>>> _check_color_image_seg(np.zeros((125, 150, 3)), np.zeros((150, 125)))
Traceback (most recent call last):
...
ValueError: ndarrays - image and segmentation do not match (125, 150, 3) vs (150, 125)
```

`imsegm.descriptors._check_gray_image_seg`*m*(*image*, *segm*)  
verify image - segmentation compatibility

#### Parameters

- **image** (*ndarray*) – image
- **segm** (*ndarray*) – segmentation

#### Return bool

```
>>> _check_gray_image_seg(np.zeros((125, 150)), np.zeros((150, 125)))
Traceback (most recent call last):
...
ValueError: ndarrays - image and segmentation do not match (125, 150) vs (150, 125)
```

`imsegm.descriptors._check_unrecognised_feature_group`(*feature\_flags*)  
search for not defined flags

**Parameters** **feature\_flags** (*dict*) – input

**Return** **list(str)** unrecognised

```
>>> _check_unrecognised_feature_group({'color': [], 'texture': []})
['texture']
```

`imsegm.descriptors._check_unrecognised_feature_names`(*feature\_flags*)  
search for not defined flags

**Parameters** **feature\_flags** (*list(str)*) – input

**Return** **list(str)** unrecognised

```
>>> _check_unrecognised_feature_names(['mean', 'average'])
['average']
```

`imsegm.descriptors.adjust_bounding_box_crop`(*image\_size*, *bbox\_size*, *position*)  
adjust the bounding box according image sizes and position

#### Parameters

- **int**] **image\_size** (*tuple(int, int)* | [*int*,]) – image size
- **int**] **bbox\_size** (*tuple(int, int)* | [*int*,]) – size of the bounding box
- **int**] **position** (*tuple(int, int)* | [*int*,]) – position in yhe image

**Return** (0, 0, 0, 0) im\_begin, im\_end, bb\_begin, bb\_end

```
>>> adjust_bounding_box_crop((50, 50), (7, 7), (20, 20))
((17, 17), (24, 24), (0, 0), (7, 7))
>>> adjust_bounding_box_crop((50, 50), (15, 15), (20, 45))
((13, 38), (28, 50), (0, 0), (15, 12))
>>> adjust_bounding_box_crop((50, 50), (15, 15), (5, 5))
((0, 0), (13, 13), (2, 2), (15, 15))
>>> adjust_bounding_box_crop((50, 50), (80, 80), (20, 20))
((0, 0), (50, 50), (20, 20), (70, 70))
```

`imsegm.descriptors.compute_image2d_color_statistic` (*image*, *segm*, *feature\_flags*=('mean', 'std', 'energy', 'median', 'meanGrad'), *color\_name*='color')

compute complete descriptors / statistic on color (2D) images

#### Parameters

- **image** (*ndarray*) –
- **segm** (*ndarray*) – segmentation
- **feature\_flags** (*list(str)*) –
- **color\_name** (*str*) – channel name

**Return** `tuple(ndarray,list(str))` `np.ndarray<nb_samples, nb_features>`

```
>>> image = np.zeros((2, 10, 3))
>>> image[:, 2:6, 0] = 1
>>> image[:, 3:7, 1] = 3
>>> image[:, 4:9, 2] = 2
>>> segm = np.array([[0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
...                 [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]])
>>> features, names = compute_image2d_color_statistic(image, segm)
>>> names
['color-ch1_mean', 'color-ch2_mean', 'color-ch3_mean',
 'color-ch1_std', 'color-ch2_std', 'color-ch3_std',
 'color-ch1_energy', 'color-ch2_energy', 'color-ch3_energy',
 'color-ch1_median', 'color-ch2_median', 'color-ch3_median',
 'color-ch1_meanGrad', 'color-ch2_meanGrad', 'color-ch3_meanGrad']
>>> features.shape
(2, 15)
>>> np.round(features, 1).tolist()
[[0.6, 1.2, 0.4, 0.5, 1.5, 0.8, 0.6, 3.6, 0.8, 1.0, 0.0, 0.0, 0.2, 0.6, 0.4],
 [0.2, 1.2, 1.6, 0.4, 1.5, 0.8, 0.2, 3.6, 3.2, 0.0, 0.0, 2.0, -0.2, -0.6, -0.6]]
```

`imsegm.descriptors.compute_image3d_gray_statistic` (*image*, *segm*, *feature\_flags*=('mean', 'std', 'energy', 'median', 'meanGrad'), *ch\_name*='gray')

compute complete descriptors / statistic on gray (3D) images

#### Parameters

- **image** (*ndarray*) –
- **segm** (*ndarray*) – segmentation
- **feature\_flags** (*list(str)*) –
- **ch\_name** (*str*) – channel name

**Return tuple(ndarray,list(str))** np.ndarray<nb\_samples, nb\_features>

```
>>> image = np.zeros((2, 3, 8))
>>> image[0, :, 2:6] = 1
>>> image[1, :, 3:7] = 3
>>> segm = np.array([[0, 0, 0, 0, 1, 1, 1, 1]] * 3,
...                 [[2, 2, 2, 2, 5, 5, 5, 5]] * 3])
>>> segm.shape
(2, 3, 8)
>>> features, names = compute_image3d_gray_statistic(image, segm)
>>> features.shape
(6, 5)
>>> np.round(features, 3)
array([[ 0.5 ,  0.5 ,  0.5 ,  0.5 ,  0.25 ],
       [ 0.5 ,  0.5 ,  0.5 ,  0.5 , -0.25 ],
       [ 0.75 ,  1.299,  2.25 ,  0. ,  0.75 ],
       [ 0. ,  0. ,  0. ,  0. ,  0. ],
       [ 0. ,  0. ,  0. ,  0. ,  0. ],
       [ 2.25 ,  1.299,  6.75 ,  3. , -1.125]])
>>> names
['gray_mean',
 'gray_std',
 'gray_energy',
 'gray_median',
 'gray_meanGrad']
```

`imsegm.descriptors.compute_img_filter_response2d(img, filter_battery)`  
compute image filter response in 2D

#### Parameters

- **img** ([[float]]) – image
- **filter\_battery** ([[float]]) – filters

#### Return [[float]]

`imsegm.descriptors.compute_img_filter_response3d(img, filter_battery)`  
compute image filter response in 3D

#### Parameters

- **img** (ndarray) –
- **filter\_battery** (ndarray) –

#### Returns

`imsegm.descriptors.compute_label_hist_proba(segm, position, struc_elem)`  
compute histogram of labels for set of centric annulus expecting that each label has own layer

#### Parameters

- **segm** (ndarray) – np.array<height, width>
- **position** (tuple(float, float)) –
- **struc\_elem** (ndarray) – np.array<height, width>

#### Return list(float)

```
>>> seg = np.zeros((50, 50, 2), dtype=float)
>>> seg[15:35, 20:40, 1] = 1
```

(continues on next page)

(continued from previous page)

```
>>> seg[:, :, 0] = 1 - seg[:, :, 1]
>>> compute_label_hist_proba(seg, (15, 20), np.ones((12, 13), dtype=int))
(array([ 114.,   42.]), 156)
```

`imsegm.descriptors.compute_label_hist_seg`(*segm*, *position*, *struc\_elem*, *nb\_labels*)  
compute histogram of labels for set of centric annulus

**Parameters**

- **segm** (*ndarray*) – np.array<height, width>
- **position** (*tuple*(*float*, *float*)) – position in the segmentation
- **struc\_elem** (*ndarray*) – np.array<height, width>
- **nb\_labels** (*int*) – total number of labels in the segmentation

**Return list(float)****See also:**

`imsegm.descriptors.cython_label_hist_seg2d()`

```
>>> segm = np.zeros((10, 10), dtype=int)
>>> segm[1:9, 2:8] = 1
>>> segm[3:7, 4:6] = 2
>>> segm
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 1, 1, 2, 2, 1, 1, 0, 0],
       [0, 0, 1, 1, 2, 2, 1, 1, 0, 0],
       [0, 0, 1, 1, 2, 2, 1, 1, 0, 0],
       [0, 0, 1, 1, 2, 2, 1, 1, 0, 0],
       [0, 0, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
>>> compute_label_hist_seg(segm, [6, 6], np.ones((3, 3)), 3)
(array([ 0.,  7.,  2.]), 9.0)
>>> compute_label_hist_seg(segm, [4, 4], np.ones((5, 5)), 3)
(array([ 0., 17.,  8.]), 25.0)
```

`imsegm.descriptors.compute_label_histograms_positions`(*segm*, *positions*, *diameters*, *nb\_labels*)  
*diameters*=(10, 20, 30, 40, 50),  
*nb\_labels*=None  
compute the histogram features doe consecutive growing diameter of inter circle neighbouring around given points in the segmentation

**Parameters**

- **segm** (*ndarray*) – np.array<height, width>
- **int)] positions** ([*int*,) – list of positions
- **diameters** (*list*(*int*)) – circular diameters
- **nb\_labels** (*int*) –

**Return tuple(ndarray,list(str))** ndarray<nb\_samples, nb\_features>, names



```

>>> segm = np.zeros((10, 10), dtype=int)
>>> segm[1:9, 2:8] = 1
>>> segm[3:7, 4:6] = 2
>>> points = [[3, 3], [4, 4], [2, 7], [6, 6]]
>>> hists, names = compute_label_histograms_positions(segm, points, [1, 2, 4])
>>> names
['hist-d_1-lb_0', 'hist-d_1-lb_1', 'hist-d_1-lb_2',      'hist-d_2-lb_0', 'hist-d_
↪2-lb_1', 'hist-d_2-lb_2',      'hist-d_4-lb_0', 'hist-d_4-lb_1', 'hist-d_4-lb_2
↪']
>>> hists.shape
(4, 9)
>>> np.round(hists, 2)
array([[ 0. ,  0.8 ,  0.2 ,  0.12,  0.62,  0.25,  0.44,  0.41,  0.15],
       [ 0. ,  0.2 ,  0.8 ,  0. ,  0.62,  0.38,  0.22,  0.75,  0.03],
       [ 0.2 ,  0.8 ,  0. ,  0.5 ,  0.5 ,  0. ,  0.46,  0.33,  0.21],
       [ 0. ,  0.8 ,  0.2 ,  0.12,  0.62,  0.25,  0.44,  0.41,  0.15]])
>>> segm = np.zeros((10, 10, 2), dtype=int)
>>> segm[3:7, 4:6, 1] = 1
>>> segm[:, :, 0] = 1 - segm[:, :, 0]
>>> points = [[3, 3], [4, 4], [2, 7], [6, 6]]
>>> hists, names = compute_label_histograms_positions(segm, points, [1, 2, 4])
>>> np.round(hists, 2)
array([[ 1. ,  0.2 ,  1. ,  0.25,  1. ,  0.15],
       [ 1. ,  0.8 ,  1. ,  0.38,  1. ,  0.03],
       [ 1. ,  0. ,  1. ,  0. ,  1. ,  0.21],
       [ 1. ,  0.2 ,  1. ,  0.25,  1. ,  0.15]])

```

`imsegm.descriptors.compute_ray_features_positions` (*segm*, *list\_positions*, *angle\_step*=5.0, *border\_labels*=None, *segm\_open*=None, *smooth\_ray*=None, *shifting*=True, *edge*='up')

compute ray features for multiple points in the segmentation with given boundary labels and step angle

#### Parameters

- **segm** (*ndarray*) – np.array<height, width>
- **int)] list\_positions** (*[(int,)* –
- **angle\_step** (*float*) –
- **border\_labels** (*list(int)*) – all labels to be set as boundaries
- **segm\_open** (*int*) –
- **smooth\_ray** (*float*) –
- **shifting** (*bool*) –
- **edge** (*str*) – type of edge up/down

#### Returns

**Note:** for more examples, see unittests

```

>>> from skimage import draw
>>> np.random.seed(0)
>>> seg = np.zeros((100, 100), dtype=int)

```

(continues on next page)

(continued from previous page)

```

>>> x, y = draw.circle(45, 55, 30, shape=seg.shape)
>>> seg[x, y] = 1
>>> x, y = draw.circle(55, 45, 10, shape=seg.shape)
>>> seg[x, y] = 2
>>> points = [(50, 50), (60, 40), (44, 55)]
>>> ray_dist, shift, _ = compute_ray_features_positions(seg, points, 20)
>>> shift
[314.3..., 314.7..., 90.0...]
>>> ray_dist.astype(int).tolist()
[[37, 37, 35, 32, 30, 27, 25, 24, 23, 23, 24, 25, 26, 30, 31, 33, 35, 38],
 [50, 47, 41, 31, 23, 17, 13, 10, 9, 9, 9, 11, 14, 19, 27, 37, 45, 50],
 [31, 31, 31, 30, 30, 29, 30, 30, 29, 29, 30, 30, 29, 30, 30, 31, 31, 31]]
>>> noise_pos = np.random.randint(10, 80, (2, 300))
>>> seg[noise_pos[0], noise_pos[1]] = 0 # add random noise
>>> ray_dist, shift, names = compute_ray_features_positions(seg, points, 45,
...                                                         segm_open=10)
>>> names
['ray-lb_0-agl_0', 'ray-lb_0-agl_45', 'ray-lb_0-agl_90',
 'ray-lb_0-agl_135', 'ray-lb_0-agl_180', 'ray-lb_0-agl_225',
 'ray-lb_0-agl_270', 'ray-lb_0-agl_315']
>>> shift
[315.0..., 315.0..., 90.0...]
>>> ray_dist.astype(int)
array([[38, 35, 29, 25, 24, 25, 29, 35],
       [52, 41, 21, 11, 9, 11, 21, 41],
       [31, 31, 30, 29, 29, 29, 30, 31]])

```

`imsegm.descriptors.compute_ray_features_segm_2d(seg_binary, position, angle_step=5.0, smooth_coef=0, edge='up')`

compute ray features vector, shift them to be starting from larges and smooth\_coef them by gauss filter (from given point the close distance to boundary)

#### Parameters

- **seg\_binary** (*ndarray*) – np.array<height, width>
- **position** (*tuple(int, int)*) – integer position in the segmentation
- **angle\_step** (*float*) – angular step for ray features
- **edge** (*str*) – pointing to the up of down edge of an boundary
- **smooth\_coef** (*int*) – smoothing the final ray features

**Return list(float)** ray distances

See also:

`imsegm.descriptors.compute_ray_features_segm_2d_vectors()`

**Note:** for more examples, see unittests

```

>>> seg_empty = np.zeros((100, 150), dtype=bool)
>>> compute_ray_features_segm_2d(seg_empty, (50, 75), 90)
array([-1., -1., -1., -1....])
>>> from skimage import draw
>>> seg = np.ones((100, 150), dtype=bool)
>>> x, y = draw.circle(50, 75, 40, shape=seg.shape)

```

(continues on next page)

(continued from previous page)

```

>>> seg[x, y] = False
>>> np.round(compute_ray_features_segm_2d(seg, (50, 75), 45))
array([ 40.,  41.,  40.,  41.,  40.,  41.,  40.,  41.]...)
>>> np.round(compute_ray_features_segm_2d(seg, (60, 40), 30, smooth_coef=1)).
->tolist()
[66.0, 52.0, 32.0, 16.0, 8.0, 5.0, 5.0, 8.0, 16.0, 33.0, 53.0, 67.0]
>>> ray_fts = compute_ray_features_segm_2d(seg, (40, 60), 20)
>>> np.round(ray_fts).tolist()
[54.0, 57.0, 59.0, 55.0, 51.0, 44.0, 38.0, 31.0, 27.0, 24.0, 22.0, 22.0,
 23.0, 26.0, 29.0, 35.0, 42.0, 49.0]

```

`imsegm.descriptors.compute_ray_features_segm_2d_vectors` (*seg\_binary*, *position*, *angle\_step=5.0*, *smooth\_coef=0*, *edge='up'*)

USES WHOLE IMAGE ROTATION SO IT IS VERY SLOW compute ray features vector , shift them to be startig from larges and smooth\_coef them by gauss filter (from fiven point the close distance to boundary)

#### Parameters

- **edge** (*str*) – pointing to the up of down edge o
- **smooth\_coef** (*int*) –
- **seg\_binary** (*ndarray*) – np.array<height, width>
- **position** (*tuple(int, int)*) –
- **angle\_step** (*float*) –

#### Return list(float)

See also:

`imsegm.descriptors.compute_ray_features_segm_2d()`

**Note:** for more examples, see unittests

```

>>> from skimage import draw
>>> seg = np.ones((100, 100), dtype=bool)
>>> x, y = draw.circle(45, 55, 30, shape=seg.shape)
>>> seg[x, y] = False
>>> compute_ray_features_segm_2d_vectors(seg, (50, 50), 45)
array([35, 29, 25, 23, 24, 29, 34, 36])
>>> compute_ray_features_segm_2d_vectors(seg, (60, 40), 30, smooth_coef=1)
array([35, 27, 18, 12, 10, 9, 12, 18, 27, 37, 45, 49])
>>> compute_ray_features_segm_2d_vectors(seg, (40, 60), 20).tolist()
[25, 27, 29, 32, 34, 35, 37, 36, 36, 34, 32, 29, 27, 25, 24, 23, 24, 24]

```

`imsegm.descriptors.compute_selected_features_color2d` (*img*, *segments*, *feature\_flags*={'color': ('mean', 'std', 'energy', 'median', 'mean-Grad'), 'tLM': ('mean', 'std', 'energy', 'median', 'mean-Grad')})

compute selected features color image 2D

#### Parameters

- **img** (*ndarray*) – image
- **segments** (*ndarray*) – segmentation
- **feature\_flags** (*dict(list(str))*) – dictionary of feature flags

Return tuple(*np.ndarray*<nb\_samples, nb\_features>, list(*str*))

```
>>> image = np.zeros((2, 10, 3))
>>> image[:, 2:6, 0] = 1
>>> image[:, 3:7, 1] = 3
>>> image[:, 4:9, 2] = 2
>>> segm = np.array([[0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
...                  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]])
>>> features, names = compute_selected_features_color2d(image, segm,
...                                                       {'color': ('mean', 'std', 'median')})
>>> np.round(features, 3)
array([[ 0.6 ,  1.2 ,  0.4 ,  0.49,  1.47,  0.8 ,  1. ,  0. ,  0. ],
       [ 0.2 ,  1.2 ,  1.6 ,  0.4 ,  1.47,  0.8 ,  0. ,  0. ,  2. ]])
>>> features, names = compute_selected_features_color2d(image, segm,
...                                                       {'color_hsv': ('mean', 'std')})
>>> np.round(features, 3)
array([[ 0.139,  0.533,  1.4 ,  0.176,  0.452,  1.356],
       [ 0.439,  0.733,  2. ,  0.244,  0.389,  1.095]])
>>> _ = compute_selected_features_color2d(image, segm,
...                                         {'tLM': ('mean', 'std', 'energy')})
>>> features, names = compute_selected_features_color2d(image, segm,
...                                                         {'tLM_short': ('mean', 'energy')})
>>> features.shape
(2, 90)
>>> features, names = compute_selected_features_color2d(image, segm)
>>> features.shape
(2, 315)
```

`imsegm.descriptors.compute_selected_features_gray2d` (*img*, *segments*, *features\_flags*={'color': ('mean', 'std', 'energy', 'median', 'mean-Grad'), 'tLM': ('mean', 'std', 'energy', 'median', 'mean-Grad')})

compute selected features for gray image 2D

#### Parameters

- **img** (*ndarray*) – image
- **segments** (*ndarray*) – segmentation
- **feature\_flags** (*dict(list(str))*) – dictionary of feature flags

Return tuple(*np.ndarray*<nb\_samples, nb\_features>, list(*str*))

```
>>> image = np.zeros((2, 10))
>>> image[0, 2:6] = 1
>>> image[1, 3:7] = 3
>>> segm = np.array([[0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
...                  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]])
>>> features, names = compute_selected_features_gray2d(
...     image, segm, {'color': ('mean', 'std', 'median')})
>>> np.round(features, 3)
array([[ 0.9 ,  1.136,  0.5 ],
```

(continues on next page)

(continued from previous page)

```

[ 0.7 , 1.187, 0.  ]])
>>> _ = compute_selected_features_gray2d(
...     image, segm, {'tLM': ('mean', 'std', 'median')})
>>> features, names = compute_selected_features_gray2d(
...     image, segm, {'tLM_short': ('mean', 'std', 'energy')})
>>> features.shape
(2, 45)
>>> features, names = compute_selected_features_gray2d(image, segm)
>>> features.shape
(2, 105)

```

```

imsegm.descriptors.compute_selected_features_gray3d(img, segments, feature_flags={'color': ('mean', 'std', 'energy')})

```

compute selected features on gray 3D image

#### Parameters

- **img** (*ndarray*) – image
- **segments** (*ndarray*) – segmentation
- **feature\_flags** (*dict(list(str))*) – dictionary of feature flags

**Return tuple**(*np.ndarray*<*nb\_samples*, *nb\_features*>, *list(str)*)

```

>>> np.random.seed(0)
>>> img = np.random.random((2, 10, 15))
>>> slic = np.zeros((2, 10, 15), dtype=int)
>>> slic[:, :, :7] += 1
>>> slic[1, :, :] += 2
>>> fts, names = compute_selected_features_gray3d(
...     img, slic, {'color': ('mean', 'std', 'median')})
>>> fts.shape
(4, 3)
>>> names
['gray_mean', 'gray_std', 'gray_median']
>>> _ = compute_selected_features_gray3d(
...     img, slic, {'tLM': ('median', 'std', 'energy')})
>>> fts, names = compute_selected_features_gray3d(
...     img, slic, {'tLM_short': ('mean', 'std', 'energy')})
>>> fts.shape
(4, 45)
>>> names
['tLM_sigma1.4-edge_mean', ..., 'tLM_sigma4.0-GaussLap2_energy']

```

```

imsegm.descriptors.compute_selected_features_img2d(image, segm, features_flags={'color': ('mean', 'std', 'energy')})

```

compute features

#### Parameters

- **img** (*ndarray*) – image
- **segments** (*ndarray*) – segmentation
- **feature\_flags** (*dict(list(str))*) – dictionary of feature flags

#### Returns

```
imsegm.descriptors.compute_texture_desc_lm_img2d_clr(img, seg, feature_flags,  
                                                    bank_type='normal')
```

compute texture descriptors via Lewen-Malik filter response

#### Parameters

- **img** (*ndarray*) – image
- **seg** (*ndarray*) – segmentation
- **feature\_flags** (*list(str)*) –
- **bank\_type** (*str*) – define used LM filter bank ['short', 'normal']

**Return tuple**(*np.ndarray*<nb\_samples, nb\_features>, *list(str)*)

See also:

```
imsegm.descriptors.compute_texture_desc_lm_img3d_val()
```

```
>>> h, w, step = 30, 20, 5
>>> np.random.seed(0)
>>> seg = np.zeros((h, w), dtype=int)
>>> for i in range(int(np.ceil(h / float(step)))):
...     for j in range(int(np.ceil(w / float(step)))):
...         val = i * (w / step) + j
...         i_step, j_step = int(i * step), int(j * step)
...         seg[i_step:int(i_step + step), j_step:int(j_step + step)] = val
>>> img = np.random.random((h, w, 3))
>>> features, names = compute_texture_desc_lm_img2d_clr(img, seg,
...                                                    ['mean', 'std', 'median'], bank_type='short')
>>> features.shape
(24, 135)
>>> names
['tLM_sigma1.4-edge-ch1_mean', ..., 'tLM_sigma1.4-edge-ch3_mean',
'tLM_sigma1.4-edge-ch1_std', ..., 'tLM_sigma1.4-edge-ch3_std',
'tLM_sigma1.4-edge-ch1_median', ..., 'tLM_sigma1.4-edge-ch3_median',
'tLM_sigma1.4-bar-ch1_mean', ..., 'tLM_sigma1.4-bar-ch3_median',
'tLM_sigma1.4-Gauss-ch1_mean', ..., 'tLM_sigma1.4-Gauss-ch3_median',
'tLM_sigma1.4-GaussLap-ch1_mean', ..., 'tLM_sigma1.4-GaussLap-ch3_median',
'tLM_sigma1.4-GaussLap2-ch1_mean', ..., 'tLM_sigma1.4-GaussLap2-ch3_median',
'tLM_sigma2.0-edge-ch1_mean', ..., 'tLM_sigma2.0-GaussLap2-ch3_median',
'tLM_sigma4.0-edge-ch1_mean', ..., 'tLM_sigma4.0-GaussLap2-ch3_median']
```

```
imsegm.descriptors.compute_texture_desc_lm_img3d_val(img, seg, feature_flags,  
                                                    bank_type='normal')
```

compute texture descriptors as mean / std / ... on Lewen-Malik filter bank response

#### Parameters

- **img** (*[[[float]]]*) – image
- **seg** (*[[[int]]]*) – segmentation
- **feature\_flags** (*list(str)*) – list of feature flags
- **bank\_type** (*str*) – define used LM filter bank ['short', 'normal']

**Return tuple**(*ndarray*,*list(str)*) *np.ndarray*<nb\_samples, nb\_features>, names

See also:

```
imsegm.descriptors.compute_texture_desc_lm_img2d_clr()
```

`imsegm.descriptors.create_filter_bank_lm_2d(radius=16, sigmas=(1.4142135623730951, 2, 2.8284271247461903, 4), nb_orient=8)`  
 create filter bank with rotation, Gaussian, Laplace-Gaussian, ...

#### Parameters

- **radius** –
- **sigmas** –
- **nb\_orient** –

**Return** `np.ndarray<nb_samples, nb_features>, list(str)`

```
>>> filters, names = create_filter_bank_lm_2d(6, SHORT_FILTERS_SIGMAS, 2)
>>> [f.shape for f in filters]
[(2, 13, 13), (2, 13, 13), (1, 13, 13), (1, 13, 13), (1, 13, 13),
 (2, 13, 13), (2, 13, 13), (1, 13, 13), (1, 13, 13), (1, 13, 13),
 (2, 13, 13), (2, 13, 13), (1, 13, 13), (1, 13, 13), (1, 13, 13)]
>>> names
['sigma1.4-edge', 'sigma1.4-bar',
 'sigma1.4-Gauss', 'sigma1.4-GaussLap', 'sigma1.4-GaussLap2',
 'sigma2.0-edge', 'sigma2.0-bar',
 'sigma2.0-Gauss', 'sigma2.0-GaussLap', 'sigma2.0-GaussLap2',
 'sigma4.0-edge', 'sigma4.0-bar',
 'sigma4.0-Gauss', 'sigma4.0-GaussLap', 'sigma4.0-GaussLap2']
```

`imsegm.descriptors.cython_img2d_color_energy(img, seg)`  
 wrapper for fast implementation of colour features

#### Parameters

- **img** (`ndarray`) – input RGB image
- **seg** (`ndarray`) – segmentation of the image

**Returns** `np.array<nb_lbs, 3>` matrix features per segment

See also:

`imsegm.descriptors.numpy_img2d_color_energy()`

```
>>> image = np.zeros((2, 10, 3))
>>> image[:, 2:6, 0] = 1
>>> image[:, 3:7, 1] = 3
>>> image[:, 4:9, 2] = 2
>>> segm = np.array([[0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
...                  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]])
>>> cython_img2d_color_energy(image, segm)
array([[ 0.6,  3.6,  0.8],
       [ 0.2,  3.6,  3.2]])
```

`imsegm.descriptors.cython_img2d_color_mean(img, seg)`  
 wrapper for fast implementation of colour features

#### Parameters

- **img** (`ndarray`) – input RGB image
- **seg** (`ndarray`) – segmentation of the image

**Returns** `np.array<nb_lbs, 3>` matrix features per segment

See also:

`imsegm.descriptors.numpy_img2d_color_mean()`

```
>>> image = np.zeros((2, 10, 3))
>>> image[:, 2:6, 0] = 1
>>> image[:, 3:7, 1] = 3
>>> image[:, 4:9, 2] = 2
>>> segm = np.array([[0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
...                  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]])
>>> cython_img2d_color_mean(image, segm)
array([[ 0.6,  1.2,  0.4],
       [ 0.2,  1.2,  1.6]])
```

`imsegm.descriptors.cython_img2d_color_std  
wrapper for fast implementation of colour features`

#### Parameters

- **img** (*ndarray*) – input RGB image
- **seg** (*ndarray*) – segmentation of the image
- **means** (*ndarray*) – precomputed feature means

**Returns** `np.array<nb_lbs, 3>` matrix features per segment

See also:

`imsegm.descriptors.numpy_img2d_color_std()`

```
>>> image = np.zeros((2, 10, 3))
>>> image[:, 2:6, 0] = 1
>>> image[:, 3:7, 1] = 3
>>> image[:, 4:9, 2] = 2
>>> segm = np.array([[0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
...                  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]])
>>> cython_img2d_color_std(image, segm)
array([[ 0.48989794,  1.46969383,  0.80000003],
       [ 0.40000001,  1.46969383,  0.80000001]])
```

`imsegm.descriptors.cython_img3d_gray_energy  
wrapper for fast implementation of colour features`

WARNING: the Z dimension is parallel and without sync, multiple equal labels across Z dim may lead to not mistakes in summing

#### Parameters

- **img** (*ndarray*) – input RGB image
- **seg** (*ndarray*) – segmentation of the image

**Returns** `np.array<nb_lbs, 1>` vector of mean colour per segment

See also:

`imsegm.descriptors.numpy_img3d_gray_energy()`

```
>>> image = np.zeros((2, 3, 8))
>>> image[0, :, 2:6] = 1
>>> image[1, :, 3:7] = 3
>>> segm = np.array([[[0, 0, 0, 0, 1, 1, 1, 1]] * 3,
```

(continues on next page)



(continued from previous page)

```
...          [[2, 2, 2, 2, 3, 3, 3, 3]] * 3])
>>> cython_img3d_gray_energy(image, segm)
array([ 0.5 ,  0.5 ,  2.25,  6.75])
```

`imsegm.descriptors.cython_img3d_gray_mean(img, seg)`

wrapper for fast implementation of colour features

WARNING: the Z dimension is parallel and without sync, multiple equal labels across Z dim may lead to not mistakes in summing

#### Parameters

- **img** (*ndarray*) – input RGB image
- **seg** (*ndarray*) – segmentation of the image

**Returns** `np.array<nb_lbs, 1>` vector of mean colour per segment

See also:

`imsegm.descriptors.numpy_img3d_gray_mean()`

```
>>> image = np.zeros((2, 3, 8))
>>> image[0, :, 2:6] = 1
>>> image[1, :, 3:7] = 3
>>> segm = np.array([[0, 0, 0, 0, 1, 1, 1, 1]] * 3,
...                 [[2, 2, 2, 2, 3, 3, 3, 3]] * 3])
>>> segm.shape
(2, 3, 8)
>>> cython_img3d_gray_mean(image, segm)
array([ 0.5 ,  0.5 ,  0.75,  2.25])
```

`imsegm.descriptors.cython_img3d_gray_std(img, seg, mean=None)`

wrapper for fast implementation of colour features

WARNING: the Z dimension is parallel and without sync, multiple equal labels across Z dim may lead to not mistakes in summing

#### Parameters

- **img** (*ndarray*) – input RGB image
- **seg** (*ndarray*) – segmentation of the image
- **mean** (*ndarray*) – precomputed feature means

**Returns** `np.array<nb_lbs, 1>` vector of mean colour per segment

See also:

`imsegm.descriptors.numpy_img3d_gray_std()`

```
>>> image = np.zeros((2, 3, 8))
>>> image[0, :, 2:6] = 1
>>> image[1, :, 3:7] = 3
>>> segm = np.array([[0, 0, 0, 0, 1, 1, 1, 1]] * 3,
...                 [[2, 2, 2, 2, 3, 3, 3, 3]] * 3])
>>> cython_img3d_gray_std(image, segm)
array([ 0.5 ,  0.5 ,  1.29903811,  1.29903811])
```

`imsegm.descriptors.cython_label_hist_seg2d(seg, select, struc_elem, nb_labels)`

compute histogram of labels for set of centric annulus

**Parameters**

- **segm\_select** (*ndarray*) – np.array<height, width>
- **struc\_elem** (*ndarray*) – np.array<height, width>
- **nb\_labels** (*int*) – total number of labels in the segmentation

**Return list(float)**

See also:

`imsegm.descriptors.compute_label_hist_seg2d()`

---

**Note:** output of this function should be equal to

```
for lb in range(nb_labels):
    hist[lb] = np.sum(np.logical_and(segm_select == lb, struc_elem == 1))
```

```
>>> segm = np.zeros((10, 10), dtype=int)
>>> segm[1:9, 2:8] = 1
>>> segm[3:7, 4:6] = 2
>>> cython_label_hist_seg2d(segm[2:5, 4:7], np.ones((3, 3)), 3)
array([ 0.,  5.,  4.])
>>> cython_label_hist_seg2d(segm[1:6, 3:8], np.ones((5, 5)), 3)
array([ 0., 19.,  6.] )
```

`imsegm.descriptors.cython_ray_features_seg2d`(*seg\_binary*, *position*, *angle\_step=5.0*,  
*edge='up'*)  
computing the Ray features from a segmentation and given position

**Parameters**

- **seg\_binary** (*ndarray*) – np.array<height, width>
- **position** (*tuple(int, int)*) – integer position in the segmentation
- **angle\_step** (*float*) – angular step for ray features
- **edge** (*str*) – pointing to the up of down edge of an boundary

**Return list(float)** ray distances

See also:

`imsegm.descriptors.numpy_ray_features_seg2d()`

```
>>> seg_empty = np.zeros((100, 150), dtype=bool)
>>> cython_ray_features_seg2d(seg_empty, (50, 75), 90)
array([-1., -1., -1., -1.]...)
>>> from skimage import draw
>>> seg = np.ones((100, 150), dtype=bool)
>>> x, y = draw.circle(50, 75, 40, shape=seg.shape)
>>> seg[x, y] = False
>>> cython_ray_features_seg2d(seg, (50, 75), 45).astype(int)
array([40, 41, 40, 41, 40, 41, 40, 41]...)
>>> cython_ray_features_seg2d(seg, (60, 40), 30).astype(int).tolist()
[74, 55, 28, 10, 5, 4, 4, 5, 9, 30, 57, 75]
>>> cython_ray_features_seg2d(seg, (40, 60), 20).astype(int).tolist()
[54, 57, 58, 55, 50, 43, 38, 31, 26, 24, 22, 22, 23, 26, 29, 34, 41, 48]
```

`imsegm.descriptors.image_subtract_gauss_smooth(img, sigma)`  
smoothing by fist dimension assuming the in dim 0. image is independent

#### Parameters

- **img** (*ndarray*) –
- **sigma** –

#### Returns

`imsegm.descriptors.interpolate_ray_dist(ray_dists, order='spline')`  
interpolate ray distances

#### Parameters

- **ray\_dists** (*list(float)*) –
- **order** (*str/int*) – degree of interpolation

#### Return list(float)

```
>>> interpolate_ray_dist([-1] * 5)
array([-1, -1, -1, -1, -1])
>>> vals = np.sin(np.linspace(0, 2 * np.pi, 20)) * 10
>>> np.round(vals).astype(int).tolist()
[0, 3, 6, 8, 10, 10, 9, 7, 5, 2, -2, -5, -7, -9, -10, -10, -8, -6, -3, 0]
>>> vals[3:7] = -1
>>> vals[16:] = -1
>>> vals_interp = interpolate_ray_dist(vals, order=3)
>>> np.round(vals_interp).astype(int).tolist()
[0, 3, 6, 9, 10, 10, 8, 7, 5, 2, -2, -5, -7, -9, -10, -10, -8, -4, 1]
>>> vals_interp = interpolate_ray_dist(vals, order='spline')
>>> np.round(vals_interp).astype(int).tolist()
[0, 3, 6, 8, 9, 10, 9, 7, 5, 2, -2, -5, -7, -9, -10, -10, -9, -7, -5, -3]
>>> vals_interp = interpolate_ray_dist(vals, order='cos')
>>> np.round(vals_interp).astype(int).tolist()
[0, 3, 6, 8, 10, 10, 9, 7, 5, 2, -2, -5, -7, -9, -10, -10, -8, -6, -3, 0]
```

`imsegm.descriptors.make_edge_filter2d(sig, phase, points, sup)`

`imsegm.descriptors.make_gaussian_filter1d(vals, sigma, order=0)`

`imsegm.descriptors.norm_features(features, scaler=None)`  
normalise features to be in range(0;1)

#### Parameters

- **features** (*ndarray*) – vector of features
- **scaler** (*obj*) –

#### Return list(list(float))

`imsegm.descriptors.numpy_img2d_color_energy(img, seg)`  
compute color energy by numpy

#### Parameters

- **img** (*ndarray*) – input RGB image
- **seg** (*ndarray*) – segmentation og the image

**Returns** `np.array<nb_lbs, 3>` matrix features per segment

See also:

`imsegm.descriptors.cython_img2d_color_energy()`

```
>>> image = np.zeros((2, 10, 3))
>>> image[:, 2:6, 0] = 1
>>> image[:, 3:8, 1] = 3
>>> image[:, 4:9, 2] = 2
>>> segm = np.array([[0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
...                  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]])
>>> numpy_img2d_color_energy(image, segm)
array([[ 0.6,   3.6,   0.8],
       [ 0.2,   5.4,   3.2]])
```

`imsegm.descriptors.numpy_img2d_color_mean(img, seg)`  
compute color means by numpy

#### Parameters

- **img** (*ndarray*) – input RGB image
- **seg** (*ndarray*) – segmentation of the image

**Returns** `np.array<nb_lbs, 3>` matrix features per segment

See also:

`imsegm.descriptors.cython_img2d_color_mean()`

```
>>> image = np.zeros((2, 10, 3))
>>> image[:, 2:6, 0] = 1
>>> image[:, 3:8, 1] = 3
>>> image[:, 4:9, 2] = 2
>>> segm = np.array([[0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
...                  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]])
>>> numpy_img2d_color_mean(image, segm)
array([[ 0.6,   1.2,   0.4],
       [ 0.2,   1.8,   1.6]])
```

`imsegm.descriptors.numpy_img2d_color_median(img, seg)`  
compute color median by numpy

#### Parameters

- **img** (*ndarray*) – input RGB image
- **seg** (*ndarray*) – segmentation of the image

**Returns** `np.array<nb_lbs, 3>` matrix features per segment

See also:

`imsegm.descriptors.cython_img2d_color_median()`

```
>>> image = np.zeros((2, 10, 3))
>>> image[:, 2:6, 0] = 1
>>> image[:, 3:8, 1] = 3
>>> image[:, 4:9, 2] = 2
>>> segm = np.array([[0, 0, 0, 0, 1, 1, 1, 1, 1, 1],
...                  [0, 0, 0, 0, 1, 1, 1, 1, 1, 1]])
>>> numpy_img2d_color_median(image, segm)
array([[ 0.5,   0. ,   0. ],
       [ 0. ,   3. ,   2. ]])
```

`imsegm.descriptors.numpy_img2d_color_std(img, seg, means=None)`  
compute color STD by numpy

#### Parameters

- **img** (*ndarray*) – input RGB image
- **seg** (*ndarray*) – segmentation of the image
- **means** (*ndarray*) – precomputed feature means

**Returns** `np.array<nb_lbs, 3>` matrix features per segment

See also:

`imsegm.descriptors.cython_img2d_color_std()`

```
>>> image = np.zeros((2, 10, 3))
>>> image[:, 2:6, 0] = 1
>>> image[:, 3:8, 1] = 3
>>> image[:, 4:9, 2] = 2
>>> segm = np.array([[0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
...                  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]])
>>> numpy_img2d_color_std(image, segm)
array([[ 0.48989795,  1.46969385,  0.8          ],
       [ 0.4          ,  1.46969385,  0.8          ]])
```

`imsegm.descriptors.numpy_img3d_gray_energy(img, seg)`  
compute gray (3D) energy by numpy

#### Parameters

- **img** – input RGB image
- **seg** – segmentation of the image

**Returns** `np.array<nb_lbs, 3>` matrix features per segment

See also:

`imsegm.descriptors.cython_img3d_gray_energy()`

```
>>> image = np.zeros((2, 3, 8))
>>> image[0, :, 2:6] = 1
>>> image[1, :, 3:7] = 3
>>> segm = np.array([[0, 0, 0, 0, 1, 1, 1, 1]] * 3,
...                  [[2, 2, 2, 2, 3, 3, 3, 3]] * 3])
>>> numpy_img3d_gray_energy(image, segm)
array([ 0.5 ,  0.5 ,  2.25,  6.75])
```

`imsegm.descriptors.numpy_img3d_gray_mean(img, seg)`  
compute gray (3D) means by numpy

#### Parameters

- **img** (*ndarray*) – input RGB image
- **seg** (*ndarray*) – segmentation of the image

**Returns** `np.array<nb_lbs, 3>` matrix features per segment

See also:

`imsegm.descriptors.cython_img3d_gray_mean()`

```
>>> image = np.zeros((2, 3, 8))
>>> image[0, :, 2:6] = 1
>>> image[1, :, 3:7] = 3
>>> segm = np.array([[0, 0, 0, 0, 1, 1, 1, 1]] * 3,
...                  [[2, 2, 2, 2, 3, 3, 3, 3]] * 3])
>>> numpy_img3d_gray_mean(image, segm)
array([ 0.5 ,  0.5 ,  0.75,  2.25])
```

```
imsegm.descriptors.numpy_img3d_gray_median(img, seg)
```

compute gray (3D) median by numpy

## Parameters

- **img** (*ndarray*) – input RGB image
- **seg** (*ndarray*) – segmentation of the image

**Returns** np.array<nb\_lbs, 3> matrix features per segment

**See also:**

```
imsegm.descriptors.cython_img3d_gray_median()
```

```
>>> image = np.zeros((2, 3, 8))
>>> image[0, :, 2:6] = 1
>>> image[1, :, 3:7] = 3
>>> segm = np.array([[0, 0, 0, 0, 1, 1, 1, 1]] * 3,
...                  [[2, 2, 2, 2, 3, 3, 3, 3]] * 3])
>>> numpy_img3d_gray_median(image, segm)
array([ 0.5,  0.5,  0. ,  3. ])
```

```
imsegm.descriptors.numpy_img3d_gray_std(img, seg, means=None)
```

compute gray (3D) STD by numpy

## Parameters

- **img** (*ndarray*) – input RGB image
- **seg** (*ndarray*) – segmentation of the image
- **means** (*ndarray*) – precomputed feature means

**Returns** np.array<nb\_lbs, 3> matrix features per segment

**See also:**

```
imsegm.descriptors.cython_img3d_gray_std()
```

```
>>> image = np.zeros((2, 3, 8))
>>> image[0, :, 2:6] = 1
>>> image[1, :, 3:7] = 3
>>> segm = np.array([[ [0, 0, 0, 0, 1, 1, 1, 1]] * 3,
...                  [[2, 2, 2, 2, 3, 3, 3, 3]] * 3])
>>> numpy_img3d_gray_std(image, segm)
array([ 0.5, 0.5, 1.29903811, 1.29903811])
```

```
imsegm.descriptors.numpy_ray_features_seg2d(seg_binary, position, angle_step=5.0,
                                             edge='up')
```

computing the Ray features from a segmentation and given position

## Parameters

- **seg\_binary**(*ndarray*) – np.array<height, width>

- **position** (*tuple(int, int)*) – integer position in the segmentation
- **angle\_step** (*float*) – angular step for ray features
- **edge** (*str*) – pointing to the up of down edge of an boundary

**Return** list(*float*) ray distances

See also:

`imsegm.descriptors.cython_ray_features_seg2d()`

```
>>> seg_empty = np.zeros((100, 150), dtype=bool)
>>> numpy_ray_features_seg2d(seg_empty, (50, 75), 90)
array([-1., -1., -1., -1.]...)
>>> from skimage import draw
>>> seg = np.ones((100, 150), dtype=bool)
>>> x, y = draw.circle(50, 75, 40, shape=seg.shape)
>>> seg[x, y] = False
>>> numpy_ray_features_seg2d(seg, (50, 75), 45).astype(int)
array([40, 41, 40, 41, 40, 41, 40, 41]...)
>>> numpy_ray_features_seg2d(seg, (60, 40), 30).astype(int).tolist()
[74, 55, 28, 10, 5, 4, 4, 5, 9, 30, 57, 75]
>>> numpy_ray_features_seg2d(seg, (40, 60), 20).astype(int).tolist()
[54, 57, 58, 55, 50, 43, 38, 31, 26, 24, 22, 22, 23, 26, 29, 34, 41, 48]
```

`imsegm.descriptors.reconstruct_ray_features_2d` (*position, ray\_features, shift=0*)  
reconstruct ray features for 2D image

#### Parameters

- **position** (*tuple(int, int) | tuple(float, float)*) –
- **ray\_features** (*list(float)*) –
- **shift** (*float*) –

**Return** [[*float, float*]]

---

**Note:** for more examples, see unittests

---

```
>>> reconstruct_ray_features_2d((10., 10), np.array([1] * 4))
array([[ 10.,  11.],
       [ 11.,  10.],
       [ 10.,   9.],
       [  9.,  10.]])
>>> reconstruct_ray_features_2d((10., 10), np.array([-1, 0, 1, np.inf]))
array([[ 10.,  10.],
       [ 10.,   9.]])
```

`imsegm.descriptors.reduce_close_points` (*points, dist\_thr*)  
reduce remove points with smaller internal distance then threshold assumption, the points are in sequence geometrically ordered)

#### Parameters

- **float]] points** (*[[float,)* –
- **dist\_thr** (*float*) – distance threshold

**Return** [[*float, float*]]

```
>>> points = np.array([range(10), range(10)]).T
>>> reduce_close_points(points, 2)
array([[0, 0],
       [2, 2],
       [4, 4],
       [6, 6],
       [8, 8]])
>>> points = np.array([[0, 0], [1, 1], [0, 2]])
>>> reduce_close_points(points, 2)
array([[0, 0],
       [0, 2]])
>>> reduce_close_points(np.ones((10, 2)), 2)
array([[ 1.,  1.]])
```

`imsegm.descriptors.shift_ray_features` (*ray\_dist*, *method*='phase')  
shift Ray features to the global maximum to be rotation invariant

#### Parameters

- **ray\_dist** (*list* (*float*)) – array of features
- **method** (*str*) – use method for estimate shift maxima (phase or max)

#### Return list(float)

```
>>> vec = np.array([43, 46, 44, 39, 28, 18, 12, 10, 9, 12, 22, 28])
>>> ray, shift = shift_ray_features(vec)
>>> shift
41.50...
>>> ray
array([46, 44, 39, 28, 18, 12, 10, 9, 12, 22, 28, 43])
>>> ray2, shift = shift_ray_features(ray)
>>> shift
11.50...
>>> np.array_equal(ray, ray2)
True
>>> ray, shift = shift_ray_features(vec, method='max')
>>> shift
30.0...
```

`imsegm.descriptors.DEFAULT_FILTERS_SIGMAS` = (1.4142135623730951, 2, 2.8284271247461903, 4)  
define sigmas for Lowen-Malik filter bank

`imsegm.descriptors.FEATURES_SET_ALL` = {'color': ('mean', 'std', 'energy', 'median', 'meanG')}  
define the richest version of computed superpixel features

`imsegm.descriptors.FEATURES_SET_COLOR` = {'color': ('mean', 'std', 'energy')}  
define basic color features for superpixels

`imsegm.descriptors.FEATURES_SET_TEXTURE` = {'tLM': ('mean', 'std', 'energy')}  
define basic texture features (complete LM filter bank) for superpixels

`imsegm.descriptors.FEATURES_SET_TEXTURE_SHORT` = {'tLM\_short': ('mean', 'std', 'energy')}  
define basic color features for (small LM filter bank) superpixels

`imsegm.descriptors.HIST_CIRCLE_DIAGONALS` = (10, 20, 30, 40, 50)  
define circular diameters for computing label histogram

`imsegm.descriptors.MAX_SIGNAL_RESPONSE` = 1000000.0  
maximal response is bounded by fix number to prevent overflowing (for LM filter bank)



```
imsegm.descriptors.NAMES_FEATURE_FLAGS = ('mean', 'std', 'energy', 'median', 'meanGrad')
    define all available statistic computed on superpixels
```

```
imsegm.descriptors.SHORT_FILTERS_SIGMAS = (1.4142135623730951, 2, 4)
    define small list/range of sigmas for Lewen-Malik filter bank
```

## imsegm.ellipse\_fitting module

Framework for ellipse fitting

Copyright (C) 2014-2018 Jiri Borovec <jiri.borovec@fel.cvut.cz>

```
class imsegm.ellipse_fitting.EllipseModelSegm(*args: Any, **kwargs: Any)
```

Bases: skimage.measure.fit.EllipseModel

Total least squares estimator for 2D ellipses.

The functional model of the ellipse is:

```
xt = xc + a*cos(theta)*cos(t) - b*sin(theta)*sin(t)
yt = yc + a*sin(theta)*cos(t) + b*cos(theta)*sin(t)
d = sqrt((x - xt)**2 + (y - yt)**2)
```

where  $(x_t, y_t)$  is the closest point on the ellipse to  $(x, y)$ . Thus  $d$  is the shortest distance from the point to the ellipse.

The estimator is based on a least squares minimization. The optimal solution is computed directly, no iterations are required. This leads to a simple, stable and robust fitting method.

The `params` attribute contains the parameters in the following order:

```
xc, yc, a, b, theta
```

## Example

```
>>> from imsegm.utilities.drawing import ellipse_perimeter
>>> params = 20, 30, 12, 16, np.deg2rad(30)
>>> rr, cc = ellipse_perimeter(*params)
>>> xy = np.array([rr, cc]).T
>>> ellipse = EllipseModelSegm()
>>> ellipse.estimate(xy)
True
>>> np.round(ellipse.params, 2)
array([ 19.5,  29.5,  12.45,  16.52,  0.53])
>>> xy = EllipseModelSegm().predict_xy(np.linspace(0, 2 * np.pi, 25), params)
>>> ellipse = EllipseModelSegm()
>>> ellipse.estimate(xy)
True
>>> np.round(ellipse.params, 2)
array([ 20.,  30.,  12.,  16.,  0.52])
>>> np.round(abs(ellipse.residuals(xy)), 5)
array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
         0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]])
>>> ellipse.params[2] += 2
>>> ellipse.params[3] += 2
>>> np.round(abs(ellipse.residuals(xy)))
array([[ 2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.,
         2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.]])
```

**criterion** (*points, weights, labels, table\_prob=(0.1, 0.9)*)

Determine residuals of data to model.

#### Parameters

- **points** – points coordinates
- **weights** – weight for each point represent the region size
- **labels** – vector of labels for each point
- **table\_prob** – is a vector or foreground probabilities for each class and being background is supplement to 1. Another option is define a matrix with number of columns related to number of classes and the first row denote probability being foreground and second being background

#### Returns

#### Example

```
>>> seg = np.zeros((10, 15), dtype=int)
>>> r, c = np.meshgrid(range(seg.shape[1]), range(seg.shape[0]))
>>> el = EllipseModelSegm()
>>> el.params = [4, 7, 3, 6, np.deg2rad(10)]
>>> weights = np.ones(seg.ravel().shape)
>>> seg[4:5, 6:8] = 1
>>> table_prob = [[0.1, 0.9]]
>>> el.criterion(np.array([r.ravel(), c.ravel()]).T, weights, seg.ravel(),
...               table_prob)
87.888...
>>> seg[2:7, 4:11] = 1
>>> el.criterion(np.array([r.ravel(), c.ravel()]).T, weights, seg.ravel(),
...               table_prob)
17.577...
>>> seg[1:9, 1:14] = 1
>>> el.criterion(np.array([r.ravel(), c.ravel()]).T, weights, seg.ravel(),
...               table_prob)
-70.311...
```

`imsegm.ellipse_fitting.add_overlap_ellipse` (*segm, ellipse\_params, label,*  
*thr\_overlap=1.0*)

add to existing image ellipse with specific label if the new ellipse does not ouvelap with already existing object / ellipse

#### Parameters

- **segm** (*ndarray*) – segmentation
- **ellipse\_params** (*tuple*) – parameters
- **label** (*int*) – selected label
- **thr\_overlap** (*float*) – relative overlap with existing objects

#### Return ndarray

```
>>> seg = np.zeros((15, 20), dtype=int)
>>> ell_params = 7, 10, 5, 8, np.deg2rad(30)
>>> ell = add_overlap_ellipse(seg, ell_params, 1)
>>> ell
```

(continues on next page)

(continued from previous page)

```

array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
>>> ell2_params = 4, 5, 2, 3, np.deg2rad(-30)
>>> add_overlap_ellipse(ell, ell2_params, 2)
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 2, 2, 2, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 1, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])

```

`imsegm.ellipse_fitting.filter_boundary_points` (*segm, slic*)

`imsegm.ellipse_fitting.get_slic_points_labels` (*segm, img=None, slic\_size=20, slic\_regul=0.1*)

run SLIC on image or superpixels and return superpixels, their centers and also labels (label from segmentation in position of superpixel centre)

#### Parameters

- **segm** (*ndarray*) – segmentation
- **img** (*ndarray*) – input image
- **slic\_size** (*int*) – superpixel size
- **slic\_regul** (*float*) – regularisation in range (0, 1)

#### Return tuple

`imsegm.ellipse_fitting.prepare_boundary_points_close` (*seg, centers, sp\_size=25, relative\_compact=0.3*)

extract some point around foreground boundaries

#### Parameters

- **seg** (*ndarray*) – input segmentation
- **int)] centers** (*[int,)*) – list of centers

- **sp\_size** (*int*) – superpixel size

**Return [ndarray]**

```
>>> seg = np.zeros((100, 200), dtype=int)
>>> ell_params = 50, 100, 40, 60, np.deg2rad(30)
>>> seg = add_overlap_ellipse(seg, ell_params, 1)
>>> pts = prepare_boundary_points_close(seg, [(40, 90)])
>>> pts
[array([[ 6, 85],
        [ 8, 150],
        ...
        [ 92, 118]])]
```

```
imsegm.ellipse_fitting.prepare_boundary_points_ray_dist(seg, centers,
                                                         close_points=1,
                                                         sel_bg=15, sel_fg=5)
```

extract some point around foreground boundaries

**Parameters**

- **seg** (*ndarray*) – input segmentation
- **int)] centers** (*[(int,)* – list of centers
- **close\_points** (*float*) – remove closest point then a given threshold
- **sel\_bg** (*int*) – smoothing background with morphological operation
- **sel\_fg** (*int*) – smoothing foreground with morphological operation

**Return [ndarray]**

```
>>> seg = np.zeros((10, 20), dtype=int)
>>> ell_params = 5, 10, 4, 6, np.deg2rad(30)
>>> seg = add_overlap_ellipse(seg, ell_params, 1)
>>> pts = prepare_boundary_points_ray_dist(seg, [(4, 9)], 2, sel_bg=0, sel_fg=0)
>>> np.round(pts, 2).tolist()
[[[4.0, 16.0],
  [6.8, 15.0],
  [9.0, 5.5],
  [4.35, 5.0],
  [1.0, 6.9],
  [1.0, 9.26],
  [0.0, 11.31],
  [0.5, 14.0],
  [1.45, 16.0]]]
```

```
imsegm.ellipse_fitting.prepare_boundary_points_ray_edge(seg, centers,
                                                         close_points=5,
                                                         min_diam=25.0,
                                                         sel_bg=15, sel_fg=5)
```

extract some point around foreground boundaries

**Parameters**

- **seg** (*ndarray*) – input segmentation
- **int)] centers** (*[(int,)* – list of centers
- **close\_points** (*float*) – remove closest point then a given threshold
- **min\_diam** (*int*) – minimal size of expected object

- **sel\_bg**(*int*) – smoothing background with morphological operation
- **sel\_fg**(*int*) – smoothing foreground with morphological operation

**Return [ndarray]**

```
>>> seg = np.zeros((10, 20), dtype=int)
>>> ell_params = 5, 10, 4, 6, np.deg2rad(30)
>>> seg = add_overlap_ellipse(seg, ell_params, 1)
>>> pts = prepare_boundary_points_ray_edge(seg, [(4, 9)], 2.5, 3, sel_bg=1, sel_
↪ fg=0)
>>> np.round(pts).tolist()
[[[4.0, 16.0],
  [7.0, 15.0],
  [9.0, 5.0],
  [4.0, 5.0],
  [1.0, 7.0],
  [0.0, 14.0]]]
```

```
imsegm.ellipse_fitting.prepare_boundary_points_ray_join(seg, centers,
close_points=5,
min_diam=25.0,
sel_bg=15, sel_fg=5)
```

extract some point around foreground boundaries

**Parameters**

- **seg**(*ndarray*) – input segmentation
- **int)] centers**(*[(int,)*) – list of centers
- **close\_points**(*float*) – remove closest point then a given threshold
- **min\_diam**(*int*) – minimal size of expected object
- **sel\_bg**(*int*) – smoothing background with morphological operation
- **sel\_fg**(*int*) – smoothing foreground with morphological operation

**Return [ndarray]**

```
>>> seg = np.zeros((10, 20), dtype=int)
>>> ell_params = 5, 10, 4, 6, np.deg2rad(30)
>>> seg = add_overlap_ellipse(seg, ell_params, 1)
>>> pts = prepare_boundary_points_ray_join(seg, [(4, 9)], 5., 3, sel_bg=1, sel_
↪ fg=0)
>>> np.round(pts).tolist()
[[[4.0, 16.0],
  [7.0, 10.0],
  [9.0, 5.0],
  [4.0, 16.0],
  [7.0, 10.0]]]
```

```
imsegm.ellipse_fitting.prepare_boundary_points_ray_mean(seg, centers,
close_points=5,
min_diam=25.0,
sel_bg=15, sel_fg=5)
```

extract some point around foreground boundaries

**Parameters**

- **seg**(*ndarray*) – input segmentation

- **int)] centers** (*[(int,)* – list of centers
- **close\_points** (*float*) – remove closest point then a given threshold
- **min\_diam** (*int*) – minimal size of expected object
- **sel\_bg** (*int*) – smoothing background with morphological operation
- **sel\_fg** (*int*) – smoothing foreground with morphological operation

**Return** [ndarray]

```
>>> seg = np.zeros((10, 20), dtype=int)
>>> ell_params = 5, 10, 4, 6, np.deg2rad(30)
>>> seg = add_overlap_ellipse(seg, ell_params, 1)
>>> pts = prepare_boundary_points_ray_mean(seg, [(4, 9)], 2.5, 3, sel_bg=1, sel_
↪fg=0)
>>> np.round(pts).tolist()
[[[4.0, 16.0],
  [7.0, 15.0],
  [9.0, 5.0],
  [4.0, 5.0],
  [1.0, 7.0],
  [0.0, 14.0]]]
```

`imsegm.ellipse_fitting.ransac_seg`(*points, model\_class, points\_all, weights, labels, table\_prob, min\_samples, residual\_threshold=1, max\_trials=100*)

Fit a model to points with the RANSAC (random sample consensus).

#### Parameters

- **points** (*[list, tuple of] (N, D) array*) – Data set to which the model is fitted, where N is the number of points points and D the dimensionality of the points. If the model class requires multiple input points arrays (e.g. source and destination coordinates of `skimage.transform.AffineTransform`), they can be optionally passed as tuple or list. Note, that in this case the functions `estimate(*points)`, `residuals(*points)`, `is_model_valid(model, *random_data)` and `is_data_valid(*random_data)` must all take each points array as separate arguments.
- **model\_class** (*class*) – Object with the following object methods:
  - `success = estimate(*points)`
  - `residuals(*points)`where *success* indicates whether the model estimation succeeded (*True* or *None* for success, *False* for failure).
- **points\_all** (*list*) –
- **weights** (*list*) –
- **labels** (*list*) –
- **table\_prob** (*list*) –
- **min\_samples** (*int float*) – The minimum number of points points to fit a model to.
- **residual\_threshold** (*float*) – Maximum distance for a points point to be classified as an inlier.

- **max\_trials** (*int*, *optional*) – Maximum number of iterations for random sample selection.

#### Returns

- **model** (*object*) – Best model with largest consensus set.
- **inliers** ((*N*, ) *array*) – Boolean mask of inliers classified as True.

#### Examples

```
>>> seg = np.zeros((120, 150), dtype=int)
>>> ell_params = 60, 75, 40, 65, np.deg2rad(30)
>>> seg = add_overlap_ellipse(seg, ell_params, 1)
>>> slic, points_all, labels = get_slic_points_labels(seg, slic_size=10,
...                                                slic_regul=0.3)
>>> points = prepare_boundary_points_ray_dist(seg, [(40, 90)], 2,
...                                                sel_bg=1, sel_fg=0)[0]
>>> table_prob = [[0.01, 0.75, 0.95, 0.9], [0.99, 0.25, 0.05, 0.1]]
>>> weights = np.bincount(slic.ravel())
>>> ransac_model, _ = ransac_segmm(points, EllipseModelSegm, points_all, weights,
...                                labels, table_prob, 0.6, 3, max_trials=15)
>>> np.round(ransac_model.params[:4]).astype(int)
array([60, 75, 41, 65])
>>> np.round(ransac_model.params[4], 1)
0.5
```

`imsegm.ellipse_fitting.split_segmm_background_foreground(seg, sel_bg=15, sel_fg=5)`  
smoothing segmentation with morphological operation

#### Parameters

- **seg** (*ndarray*) – input segmentation
- **sel\_bg** (*int/float*) – smoothing background with morphological operation
- **sel\_fg** (*int*) – smoothing foreground with morphological operation

#### Return tuple(*ndarray*,*ndarray*)

```
>>> seg = np.zeros((10, 20), dtype=int)
>>> ell_params = 5, 10, 4, 6, np.deg2rad(30)
>>> seg = add_overlap_ellipse(seg, ell_params, 1)
>>> seg_bg, seg_fc = split_segmm_background_foreground(seg, 1.5, 0)
>>> seg_bg.astype(int)
array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1]])
>>> seg_fc.astype(int)
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0]])
```

(continues on next page)

(continued from previous page)

```
[0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]])
```

```
imsegm.ellipse_fitting.MAX_FIGURE_SIZE = 14
```

define maximal Figure size in larger dimension

```
imsegm.ellipse_fitting.MIN_ELLIPSE_DAIM = 25.0
```

define minimal size of estimated ellipse

```
imsegm.ellipse_fitting.STRUC_ELEM_BG = 15
```

smoothing background with morphological operation

```
imsegm.ellipse_fitting.STRUC_ELEM_FG = 5
```

smoothing foreground with morphological operation

## imsegm.features\_cython module

## imsegm.graph\_cuts module

Framework for GraphCut

Copyright (C) 2014-2018 Jiri Borovec <jiri.borovec@fel.cvut.cz>

```
imsegm.graph_cuts.compute_edge_model (edges, proba, metric='l_T')
```

compute the edge weight from the feature space

**small differences are large weights, diff close 0 appears to be 1** setting min weight ~ max difference in proba as weight meaning if two vertexes have same proba to all classes the diff is 0 and weights are 1 on the other hand if there is [0.7, 0.1, 0.2] and [0.2, 0.7, 0.1] gives large diff [0.5, 0.6, 0.1] in 1. and 2. diff and zero in 3 leading to weights [0.5, 0.4, 0.9] and so we take the min values

### Parameters

- **int)] edges** ([*int*,) – edges
- **proba** ([*float*]) – probabilities
- **metric** (*str*) – define metric

### Return list(float)

```
>>> segments = np.array([[0] * 3 + [1] * 5 + [2] * 4,
...                      [4] * 4 + [5] * 5 + [6] * 3])
>>> edges = np.array(get_vertexes_edges(segments)[1], dtype=int)
>>> np.random.seed(0)
>>> img = np.random.random(segments.shape + (3,)) * 255
>>> proba = np.random.random((segments.max() + 1, 2))
>>> weights = compute_edge_model(edges, proba, metric='l1')
>>> np.round(weights, 3).tolist()
[0.002, 0.015, 0.001, 0.002, 0.0, 0.002, 0.015, 0.034, 0.001]
>>> weights = compute_edge_model(edges, proba, metric='l2')
>>> np.round(weights, 3).tolist()
```

(continues on next page)



(continued from previous page)

```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.002, 0.005, 0.0]
>>> weights = compute_edge_model(edges, proba, metric='lT')
>>> np.round(weights, 3).tolist()
[0.0, 0.002, 0.0, 0.005, 0.0, 0.0, 0.101, 0.092, 0.001]
```

`imsegm.graph_cuts.compute_edge_weights`(*segments*, *image=None*, *features=None*,  
*proba=None*, *edge\_type=""*)  
 pp 32, [http://www.coe.utah.edu/~cs7640/readings/graph\\_cuts\\_intro.pdf](http://www.coe.utah.edu/~cs7640/readings/graph_cuts_intro.pdf)  $\exp(-\text{norm value diff}) * (\text{geom dist vertex})^{*-1}$

**Parameters**

- **segments** (*ndarray*) – superpixels
- **image** (*ndarray*) – input image
- **features** (*ndarray*) – features for each segment (superpixel)
- **proba** (*ndarray*) – probability of each superpixel and class
- **edge\_type** (*str*) – contains edge type, if 'model', after '\_' you can specify the metric, eg. 'model\_l2'

**Return [[int, int]], [float]**

```
>>> segments = np.array([[0] * 3 + [1] * 5 + [2] * 4,  
...                      [4] * 4 + [5] * 5 + [6] * 3])
>>> np.random.seed(0)
>>> img = np.random.random(segments.shape + (3,)) * 255
>>> features = np.random.random((segments.max() + 1, 15)) * 10
>>> proba = np.random.random((segments.max() + 1, 2))
>>> edges, weights = compute_edge_weights(segments)
>>> edges.tolist()
[[0, 1], [1, 2], [0, 4], [1, 4], [1, 5], [2, 5], [4, 5], [2, 6], [5, 6]]
>>> np.round(weights, 2).tolist()
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
>>> edges, weights = compute_edge_weights(segments, image=img,  
...                                     edge_type='spatial')
>>> np.round(weights, 3).tolist()
[0.776, 0.69, 2.776, 0.853, 2.194, 0.853, 0.69, 2.776, 0.776]
>>> edges, weights = compute_edge_weights(segments, image=img,  
...                                     edge_type='color')
>>> np.round(weights, 3).tolist()
[0.06, 0.002, 0.001, 0.001, 0.001, 0.009, 0.001, 0.019, 0.044]
>>> edges, weights = compute_edge_weights(segments, features=features,  
...                                     edge_type='features')
>>> np.round(weights, 3).tolist()
[0.031, 0.005, 0.051, 0.032, 0.096, 0.013, 0.018, 0.033, 0.013]
>>> edges, weights = compute_edge_weights(segments, proba=proba,  
...                                     edge_type='model')
>>> np.round(weights, 3).tolist()
[0.001, 0.028, 1.122, 0.038, 0.117, 0.688, 0.487, 1.152, 0.282]
```

`imsegm.graph_cuts.compute_multivarian_otsu`(*features*)  
 compute otsu individually over each sample dimension WARNING: this compute only locally and since it does  
 compare all combinations of orienting the assign for tight cases it may not decide

**Parameters** *features* (*ndarray*) –

**Return** list(bool)

```
>>> np.random.seed(0)
>>> fts = np.row_stack([np.random.random((5, 3)) - 1,
...                     np.random.random((5, 3)) + 1])
>>> fts[:, 1] = - fts[:, 1]
>>> compute_multivarian_otsu(fts).astype(int)
array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1])
```

`imsegm.graph_cuts.compute_pairwise_cost` (*gc\_regul*, *proba\_shape*,  
*max\_pairwise\_cost=100000.0*)  
wrapper for creating GC pairwise cost

#### Parameters

- **gc\_regul** –
- **proba\_shape** (*tuple(int, int)*) –
- **max\_pairwise\_cost** (*float*) –

#### Return ndarray

`imsegm.graph_cuts.compute_pairwise_cost_from_transitions` (*trans*, *min\_prob=1e-09*)  
compute pairwise cost from segments-label transitions

#### Parameters

- **trans** (*ndarray*) –
- **min\_prob** (*float*) – minimal probability

#### Return ndarray

```
>>> trans = np.array([[ 25.,   5.,   0.],
...                  [  5.,  10.,   8.],
...                  [  0.,   8.,  30.]])
>>> np.round(compute_pairwise_cost_from_transitions(trans), 3)
array([[ 0.182,  1.526, 20.723],
       [ 1.526,  0.833,  1.056],
       [20.723,  1.056,  0.236]])
>>> np.round(compute_pairwise_cost_from_transitions(np.ones(3)), 2)
array([[ 1.1,   1.1,   1.1],
       [ 1.1,   1.1,   1.1],
       [ 1.1,   1.1,   1.1]])
>>> np.round(compute_pairwise_cost_from_transitions(np.eye(3)), 2)
array([[ 0. , 20.72, 20.72],
       [20.72,  0. , 20.72],
       [20.72, 20.72,  0. ]])
```

`imsegm.graph_cuts.compute_spatial_dist` (*centres*, *edges*, *relative=False*)  
compute spatial distance between all neighbouring segments

#### Parameters

- **int]] centres** (*[[int,)* – superpixel centres
- **int]] edges** (*[[int,)* –
- **relative** (*bool*) – normalise the distances to mean distance

#### Returns

```
>>> from imsegm.superpixels import superpixel_centers
>>> segments = np.array([[0] * 3 + [1] * 2 + [2] * 5,
...                       [4] * 4 + [5] * 2 + [6] * 4])
>>> centres = superpixel_centers(segments)
>>> edges = [[0, 1], [1, 2], [4, 5], [5, 6], [0, 4], [1, 5], [2, 6]]
>>> np.round(compute_spatial_dist(centres, edges), 2)
array([ 2.5 ,  3.5 ,  3.  ,  3.  ,  1.12,  1.41,  1.12])
>>> np.round(compute_spatial_dist(centres, edges, relative=True), 2)
array([ 1.12,  1.57,  1.34,  1.34,  0.5 ,  0.63,  0.5 ])
```

`imsegm.graph_cuts.compute_unary_cost` (*proba*, *min\_prob=0.01*)  
compute the GC unary cost with some threshold on minimal values

#### Parameters

- **proba** (*ndarray*) –
- **min\_prob** (*float*) –

#### Return ndarray

```
>>> compute_unary_cost(np.random.random((50, 2))).shape
(50, 2)
```

`imsegm.graph_cuts.count_label_transitions_connected_segments` (*dict\_slices*,  
*dict\_labels*,  
*nb\_labels=None*)  
count transitions among labeled segment in between connected segments

#### Parameters

- **dict\_slices** (*dict(list(list(int)))*) – image name: ndarray
- **dict\_labels** (*dict(list(int))*) – image name: ndarray
- **nb\_labels** (*int*) –

#### Return ndarray

matrix of shape *nb\_labels* x *nb\_labels*

```
>>> dict_slices = {'a':
...               np.array([[0] * 3 + [1] * 3 + [2] * 3 + [3] * 3 + [4] * 3,
...                           [5] * 3 + [6] * 3 + [7] * 3 + [8] * 3 + [9] * 3])}
>>> dict_labels = {'a': np.array([0, 0, 1, 1, 2, 0, 1, 1, 0, 2])}
>>> dict_slices['a']
array([[0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4],
       [5, 5, 5, 6, 6, 6, 7, 7, 7, 8, 8, 8, 9, 9, 9]])
>>> dict_labels['a'][dict_slices['a']]
array([[0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 2],
       [0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 2, 2, 2]])
>>> count_label_transitions_connected_segments(dict_slices, dict_labels)
array([[ 2.,  5.,  1.],
       [ 5.,  3.,  1.],
       [ 1.,  1.,  1.]])
```

`imsegm.graph_cuts.create_pairwise_matrix` (*gc\_regul*, *nb\_classes*)  
wrapper for create pairwise matrix - uniform or specific

#### Parameters

- **gc\_regul** –
- **nb\_classes** (*int*) –

**Returns** np.array<nb\_classes, nb\_classes>

```
>>> create_pairwise_matrix(0.6, 3)
array([[ 0. ,  0.6,  0.6],
       [ 0.6,  0. ,  0.6],
       [ 0.6,  0.6,  0. ]])
>>> create_pairwise_matrix([((1, 2), 0.5), ((0, 2), 0.7)], 3)
array([[ 0. ,  1. ,  0.7],
       [ 1. ,  0. ,  0.5],
       [ 0.7,  0.5,  0. ]])
>>> trans = np.array([[ 341.,  31.,  22.],
...                  [ 31.,  12.,  21.],
...                  [ 22.,  21.,  44.]])
>>> gc_regul = compute_pairwise_cost_from_transitions(trans)
>>> np.round(create_pairwise_matrix(gc_regul, len(gc_regul)), 2)
array([[ 0. ,  0.58,  1.23],
       [ 0.58,  1.53,  0.97],
       [ 1.23,  0.97,  0.54]])
```

`imsegm.graph_cuts.create_pairwise_matrix_specif` (*pos\_weights*, *nb\_classes=None*)  
create GC pairwise matrix with specific values on particular positions

**Parameters**

- **int), float)] pos\_weights** ([ (*int*), – pair of coord in matrix and values
- **nb\_classes** (*int*) – initialise as empty matrix

**Returns** np.array<nb\_classes, nb\_classes>

```
>>> create_pairwise_matrix_specif([((1, 2), 0.5), ((1, 0), 0.7)], 4)
array([[ 0. ,  0.7,  1. ,  1. ],
       [ 0.7,  0. ,  0.5,  1. ],
       [ 1. ,  0.5,  0. ,  1. ],
       [ 1. ,  1. ,  1. ,  0. ]])
>>> create_pairwise_matrix_specif([((1, 2), 0.5), ((0, 2), 0.7)])
array([[ 0. ,  1. ,  0.7],
       [ 1. ,  0. ,  0.5],
       [ 0.7,  0.5,  0. ]])
```

`imsegm.graph_cuts.create_pairwise_matrix_uniform` (*gc\_reg*, *nb\_classes*)  
create GC pairwise matrix - uniform with zeros on diagonal

**Parameters**

- **gc\_reg** (*float*) –
- **nb\_classes** (*int*) –

**Return ndarray**

```
>>> create_pairwise_matrix_uniform(0.2, 3)
array([[ 0. ,  0.2,  0.2],
       [ 0.2,  0. ,  0.2],
       [ 0.2,  0.2,  0. ]])
```

`imsegm.graph_cuts.estim_class_model` (*features*, *nb\_classes*, *estim\_model='GMM'*,  
*pca\_coef=None*, *use\_scaler=True*, *max\_iter=99*)  
create pipeline (scaler, PCA, model) over several options how to cluster samples and fit it on data

**Parameters**

- **features** (*ndarray*) –
- **nb\_classes** (*int*) – number of expected classes
- **pca\_coef** (*float*) – range (0, 1) or None
- **use\_scaler** (*bool*) – whether use a scaler
- **estim\_model** (*str*) – used model
- **max\_iter** (*int*) –

#### Returns

```
>>> np.random.seed(0)
>>> fts = np.row_stack([np.random.random((50, 3)) - 1,
...                     np.random.random((50, 3)) + 1])
>>> mm = estim_class_model(fts, 2)
>>> mm.predict_proba(fts).shape
(100, 2)
>>> mm = estim_class_model(fts, 2, estim_model='GMM_kmeans',
...                         pca_coef=0.95, max_iter=3)
>>> mm.predict_proba(fts).shape
(100, 2)
>>> mm = estim_class_model(fts, 2, estim_model='GMM_Otsu', max_iter=3)
>>> mm.predict_proba(fts).shape
(100, 2)
>>> mm = estim_class_model(fts, 2, estim_model='kmeans_quantiles',
...                         use_scaler=False, max_iter=3)
>>> mm.predict_proba(fts).shape
(100, 2)
>>> mm = estim_class_model(fts, 2, estim_model='BGM', max_iter=3)
>>> mm.predict_proba(fts).shape
(100, 2)
>>> mm = estim_class_model(fts, 2, estim_model='Otsu', max_iter=3)
>>> mm.predict_proba(fts).shape
(100, 2)
```

`imsegm.graph_cuts.estim_class_model_gmm(features, nb_classes, init='kmeans')`  
 from all features estimate Gaussian Mixture Model and assuming each cluster is a single class compute probability that each feature belongs to each class

#### Parameters

- **features** (*[[float]]*) – list of features per segment
- **nb\_classes** (*int*) – number of classes
- **init** (*int*) – initialisation

**Return** *[[float]]* probabilities that each feature belongs to each class

```
>>> np.random.seed(0)
>>> fts = np.row_stack([np.random.random((50, 3)) - 1,
...                     np.random.random((50, 3)) + 1])
>>> mm = estim_class_model_gmm(fts, 2)
>>> mm.predict_proba(fts).shape
(100, 2)
```

`imsegm.graph_cuts.estim_class_model_kmeans(features, nb_classes, init_type='k-means++', max_iter=99)`  
 from all features estimate Gaussian from k-means clustering

**Parameters**

- **features** (*[[float]]*) – list of features per segment
- **nb\_classes** (*int*) – number of classes
- **init\_type** (*str*) – initialization
- **max\_iter** (*int*) – maximal number of iterations

**Return** *[[float]]* probabilities that each feature belongs to each class

```
>>> np.random.seed(0)
>>> fts = np.row_stack([np.random.random((50, 3)) - 1,
...                    np.random.random((50, 3)) + 1])
>>> mm, y = estim_class_model_kmeans(fts, 2, max_iter=9)
>>> y.shape
(100,)
>>> mm.predict_proba(fts).shape
(100, 2)
```

`imsegm.graph_cuts.estim_gmm_params` (*features, prob*)  
with given soft labeling (take the maxim) get the GMM parameters

**Parameters**

- **features** (*ndarray*) –
- **prob** (*ndarray*) –

**Returns**

```
>>> np.random.seed(0)
>>> prob = np.array([[1, 0]] * 30 + [[0, 1]] * 40)
>>> fts = prob + np.random.random(prob.shape)
>>> mm = estim_gmm_params(fts, prob)
>>> mm['weights']
[0.42857142857142855, 0.5714285714285714]
>>> mm['means']
array([[ 1.49537196,  0.53745455],
       [ 0.54199936,  1.42606497]])
```

`imsegm.graph_cuts.get_vertexes_edges` (*segments*)  
wrapper - get list of vertexes edges for 2D / 3D images

**Parameters** *segments* (*ndarray*) –

**Returns**

`imsegm.graph_cuts.insert_gc_debug_images` (*debug\_visual, segments, graph\_labels, unary\_cost, edges, edge\_weights*)

wrapper for placing intermediate variable to a dictionary

`imsegm.graph_cuts.segment_graph_cut_general` (*segments, proba, image=None, features=None, gc\_regul=1.0, edge\_type='model', edge\_cost=1.0, debug\_visual=None*)

segment the image segmented via superpixels and estimated features

**Parameters**

- **features** (*ndarray*) – features sor each instance
- **segments** (*ndarray*) – segmentation mapping each pixel into a class

- **image** (*ndarray*) – image
- **proba** (*ndarray*) – probabilities that each feature belongs to each class
- **edge\_type** (*str*) –
- **edge\_cost** (*str*) –
- **gc\_regul** – regularisation for GrphCut
- **debug\_visual** (*dict*) –

**Return** *list(int)* labelling by resulting classes

```
>>> np.random.seed(0)
>>> segments = np.array([[0] * 3 + [2] * 3 + [4] * 3 + [6] * 3 + [8] * 3,
...                       [1] * 3 + [3] * 3 + [5] * 3 + [7] * 3 + [9] * 3])
>>> proba = np.array([[0.1] * 6 + [0.9] * 4,
...                  [0.9] * 6 + [0.1] * 4], dtype=float).T
>>> proba += (0.5 - np.random.random(proba.shape)) * 0.2
>>> compute_unary_cost(proba)
array([[ 2.40531242,  0.15436155],
       [ 2.53266106,  0.11538463],
       [ 2.1604864 ,  0.13831863],
       [ 2.18495711,  0.19644636],
       [ 4.60517019,  0.0797884 ],
       [ 3.17833405,  0.11180231],
       [ 0.12059702,  4.20769207],
       [ 0.0143091 ,  1.70059894],
       [ 0.01005034,  3.39692559],
       [ 0.16916609,  3.64975219]])
>>> segment_graph_cut_general(segments, proba, gc_regul=0., edge_type='')
array([1, 1, 1, 1, 1, 1, 0, 0, 0, 0], dtype=int32)
>>> labels = segment_graph_cut_general(segments, proba, gc_regul=1.,
...                                  edge_type='spatial')
>>> labels[segments]
array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
       [1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0]], dtype=int32)
>>> slic = np.array([[0] * 4 + [1] * 6 + [2] * 4,
...                  [3] * 5 + [4] * 4 + [5] * 5])
>>> proba = np.array([[1] * 3 + [0] * 3, [0] * 3 + [1] * 3], dtype=float).T
>>> proba += np.random.random(proba.shape) / 2.
>>> np.argmax(proba, axis=1)
array([0, 0, 0, 1, 1, 1])
>>> debug_visual = dict()
>>> segment_graph_cut_general(slic, proba, gc_regul=0., edge_type='',
...                          debug_visual=debug_visual)
array([0, 0, 0, 1, 1, 1], dtype=int32)
>>> sorted(debug_visual.keys())
['edge_weights', 'edges', 'img_graph_edges', 'img_graph_seg',
 'imgs_unary_cost', 'segments']
```

`imsegm.graph_cuts.DEFAULT_GC_ITERATIONS = 25`

define number of iteration in Grap-Cut optimization

`imsegm.graph_cuts.MAX_PAIRWISE_COST = 100000.0`

define maximal value of pairwise (smoothness) term in Graph-Cut

`imsegm.graph_cuts.MIN_MAX_EDGE_WEIGHT = 1000.0`

max is this value and min is inverse (1 / val)

`imsegm.graph_cuts.MIN_UNARY_PROB = 0.01`  
define minimal value of unary (being a class) term in Graph-Cut

## imsegm.labeling module

Framework for labeling

Copyright (C) 2014-2018 Jiri Borovec <jiri.borovec@fel.cvut.cz>

`imsegm.labeling.assign_label_by_max(label_hist)`  
assign label according maximal label count in particular region

**Parameters** `label_hist` (`dict(list(int))`) – mapping of label to histogram

**Return** `list(int)` resulting LookUpTable

```
>>> slic = np.array([[0] * 4 + [1] * 3 + [2] * 3 + [3] * 3] * 4 +
...                 [[4] * 3 + [5] * 3 + [6] * 3 + [7] * 4] * 4)
>>> segm = np.zeros(slic.shape, dtype=int)
>>> segm[4:, 6:] = 1
>>> lb_hist = segm_labels_assignment(slic, segm)
>>> assign_label_by_max(lb_hist)
array([0, 0, 0, 0, 0, 0, 1, 1])
```

`imsegm.labeling.assign_label_by_threshold(dict_label_hist, thresh=0.75)`  
assign label if the purity reach certain threshold

**Parameters**

- `dict_label_hist` (`dict(list(int))`) – mapping of label to histogram
- `thresh` (`float`) – threshold for region purity

**Return** `list(int)` resulting LookUpTable

```
>>> slic = np.array([[0] * 4 + [1] * 3 + [2] * 3 + [3] * 3] * 4 +
...                 [[4] * 3 + [5] * 3 + [6] * 3 + [7] * 4] * 4)
>>> segm = np.zeros(slic.shape, dtype=int)
>>> segm[4:, 6:] = 1
>>> lb_hist = segm_labels_assignment(slic, segm)
>>> assign_label_by_threshold(lb_hist)
array([0, 0, 0, 0, 0, 0, 1, 1])
```

`imsegm.labeling.assume_bg_on_boundary(segm, bg_label=0, boundary_size=1)`  
swap labels such that the background label will be mostly on image boundary

**Parameters**

- `segm` (`ndarray`) – segmentation
- `bg_label` (`int`) – background label
- `boundary_size` (`float`) –

**Returns**

```
>>> segm = np.zeros((6, 12), dtype=int)
>>> segm[1:4, 4:] = 2
>>> assume_bg_on_boundary(segm, boundary_size=1)
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2],
```

(continues on next page)



(continued from previous page)

```

    [0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2],
    [0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
>>> segm[segm == 0] = 1
>>> assume_bg_on_boundary(segm, boundary_size=1)
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2],
       [0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2],
       [0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])

```

`imsegm.labeling.binary_image_from_coords` (*coords*, *size*)  
create binary image just from point contours

#### Parameters

- **coords** (*ndarray*) –
- **size** (*tuple(int, int)*) –

#### Return ndarray

```

>>> img = np.zeros((6, 6), dtype=int)
>>> img[1:5, 2:] = 1
>>> coords = contour_coords(img)
>>> binary_image_from_coords(coords, img.shape)
array([[0, 0, 0, 0, 0, 0],
       [0, 0, 1, 1, 1, 0],
       [0, 0, 1, 0, 0, 0],
       [0, 0, 1, 0, 0, 0],
       [0, 0, 1, 1, 1, 0],
       [0, 0, 0, 0, 0, 0]])

```

`imsegm.labeling.compute_boundary_distances` (*segm\_ref*, *segm*)  
compute distances among boundaries of two segmentation

#### Parameters

- **segm\_ref** (*ndarray*) – reference segmentation
- **segm** (*ndarray*) – input segmentation

#### Return ndarray

```

>>> segm_ref = np.zeros((6, 10), dtype=int)
>>> segm_ref[3:4, 4:5] = 1
>>> segm = np.zeros((6, 10), dtype=int)
>>> segm[:, 2:9] = 1
>>> pts, dist = compute_boundary_distances(segm_ref, segm)
>>> pts
array([[2, 4],
       [3, 3],
       [3, 4],
       [3, 5],
       [4, 4]])
>>> dist.tolist()
[2.0, 1.0, 2.0, 3.0, 2.0]

```

`imsegm.labeling.compute_distance_map(seg, label=1)`  
compute distance from label boundaries

#### Parameters

- **seg** (*ndarray*) – integer images, typically a segmentation
- **label** (*int*) – selected single label in segmentation

#### Return ndarray

```
>>> img = np.zeros((6, 6), dtype=int)
>>> img[1:5, 2:] = 1
>>> dist = compute_distance_map(img)
>>> np.round(dist, 2)
array([[ 2.24,  1.41,  1.   ,  1.   ,  1.   ,  1.41],
       [ 2.   ,  1.   ,  0.   ,  0.   ,  0.   ,  1.   ],
       [ 2.   ,  1.   ,  0.   ,  1.   ,  1.   ,  1.41],
       [ 2.   ,  1.   ,  0.   ,  1.   ,  1.   ,  1.41],
       [ 2.   ,  1.   ,  0.   ,  0.   ,  0.   ,  1.   ],
       [ 2.24,  1.41,  1.   ,  1.   ,  1.   ,  1.41]])
```

`imsegm.labeling.compute_labels_overlap_matrix(seg1, seg2)`  
compute overlap between the segmentation atlases with same sizes

#### Parameters

- **seg1** (*ndarray*) – np.array<height, width>
- **seg2** (*ndarray*) – np.array<height, width>

#### Return ndarray np.array<height, width>

```
>>> seg1 = np.zeros((7, 15), dtype=int)
>>> seg1[1:4, 5:10] = 3
>>> seg1[5:7, 6:13] = 2
>>> seg2 = np.zeros((7, 15), dtype=int)
>>> seg2[2:5, 7:12] = 1
>>> seg2[4:7, 7:14] = 3
>>> compute_labels_overlap_matrix(seg1, seg1)
array([[76,  0,  0,  0],
       [ 0,  0,  0,  0],
       [ 0,  0, 14,  0],
       [ 0,  0,  0, 15]])
>>> compute_labels_overlap_matrix(seg1, seg2)
array([[63,  4,  0,  9],
       [ 0,  0,  0,  0],
       [ 2,  0,  0, 12],
       [ 9,  6,  0,  0]])
```

`imsegm.labeling.contour_binary_map(seg, label=1, include_boundary=False)`  
get object boundaries

#### Parameters

- **seg** (*ndarray*) – integer images, typically a segmentation
- **label** (*int*) – selected single label in segmentation
- **include\_boundary** (*bool*) – assume that the object ends with image boundary

#### Return ndarray

```
>>> img = np.zeros((6, 6), dtype=int)
>>> img[1:5, 2:] = 1
>>> contour_binary_map(img)
array([[0, 0, 0, 0, 0, 0],
       [0, 0, 1, 1, 1, 0],
       [0, 0, 1, 0, 0, 0],
       [0, 0, 1, 0, 0, 0],
       [0, 0, 1, 1, 1, 0],
       [0, 0, 0, 0, 0, 0]])
>>> contour_binary_map(img, include_boundary=True)
array([[0, 0, 0, 0, 0, 0],
       [0, 0, 1, 1, 1, 1],
       [0, 0, 1, 0, 0, 1],
       [0, 0, 1, 0, 0, 1],
       [0, 0, 1, 1, 1, 1],
       [0, 0, 0, 0, 0, 0]])
```

`imsegm.labeling.contour_coords` (*seg*, *label=1*, *include\_boundary=False*)  
get object boundaries

#### Parameters

- **seg** (*ndarray*) – integer images, typically a segmentation
- **label** (*int*) – selected single label in segmentation
- **include\_boundary** (*bool*) – assume that the object end with image boundary

**Return** `[[int, int]]`

```
>>> img = np.zeros((6, 6), dtype=int)
>>> img[1:5, 2:] = 1
>>> contour_coords(img)
[[1, 2], [1, 3], [1, 4], [2, 2], [3, 2], [4, 2], [4, 3], [4, 4]]
>>> contour_coords(img, include_boundary=True)
[[1, 2], [1, 3], [1, 4], [2, 2], [3, 2], [4, 2], [4, 3], [4, 4],
 [1, 5], [2, 5], [3, 5], [4, 5]]
```

`imsegm.labeling.convert_segms_2_list` (*segms*)  
convert segmentation to a list that can be simply user for standard evaluation (classification or clustering metrics)

**Parameters** **segms** (*[ndarray]*) – list of segmentation

**Return** `list(int)`

```
>>> seg_pipe = np.ones((2, 3), dtype=int)
>>> convert_segms_2_list([seg_pipe, seg_pipe * 0, seg_pipe * 2])
[1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2]
```

`imsegm.labeling.histogram_regions_labels_counts` (*slic*, *segm*)  
histogram or overlapping region between two segmentations, the typical usage is label superpixel from annotation

#### Parameters

- **slic** (*ndarray*) – input superpixel segmentation
- **segm** (*ndarray*) – reference segmentation

**Return** `ndarray`

```
>>> slic = np.array([[0] * 3 + [1] * 3 + [2] * 3] * 4 +
...                 [[4] * 3 + [5] * 3 + [6] * 3] * 4)
>>> segm = np.zeros(slic.shape, dtype=int)
>>> segm[4:, 5:] = 2
>>> histogram_regions_labels_counts(slic, segm)
array([[ 12.,   0.,   0.],
       [ 12.,   0.,   0.],
       [ 12.,   0.,   0.],
       [  0.,   0.,   0.],
       [ 12.,   0.,   0.],
       [  8.,   0.,   4.],
       [  0.,   0.,  12.]])
```

`imsegm.labeling.histogram_regions_labels_norm(slic, segm)`

normalised histogram or overlapping region between two segmentation, the typical usage is label superpixel from annotation - relative overlap

#### Parameters

- **slic** (*ndarray*) – input superpixel segmentation
- **segm** (*ndarray*) – reference segmentation

#### Return ndarray

```
>>> slic = np.array([[0] * 3 + [1] * 3 + [2] * 3] * 4 +
...                 [[4] * 3 + [5] * 3 + [6] * 3] * 4)
>>> segm = np.zeros(slic.shape, dtype=int)
>>> segm[4:, 5:] = 2
>>> histogram_regions_labels_norm(slic, segm)
array([[ 1.,   0.,   0.],
       [ 1.,   0.,   0.],
       [ 1.,   0.,   0.],
       [ 0.,   0.,   0.],
       [ 1.,   0.,   0.],
       [ 0.66666667, 0., 0.33333333],
       [ 0.,   0.,   1.]])
```

`imsegm.labeling.mask_seg_labels(img_labeling, labels, mask_init=None)`

with given labels image and list of desired labels it create mask finding all labels in the list (perform logical or on image with a list of labels)

#### Parameters

- **im\_labeling** (*ndarray*) – np.array<height, width> input labeling
- **labels** (*list(int)*) – list of wanted labels to be detected in image
- **mask\_init** (*ndarray*) – np.array<height, width> initial bool mask on the beginning

#### Return ndarray np.array<height, width> bool mask

```
>>> img = np.zeros((4, 6))
>>> img[:-1, 1:] = 1
>>> img[1:2, 2:4] = 2
>>> mask_seg_labels(img, [1])
array([[False,  True,  True,  True,  True,  True],
       [False,  True, False, False,  True,  True],
       [False,  True,  True,  True,  True,  True],
       [False, False, False, False, False, False]], dtype=bool)
```

(continues on next page)

(continued from previous page)

```
>>> mask_segmlabels(img, [2], np.full(img.shape, True, dtype=bool))
array([[ True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True],
       [ True,  True,  True,  True,  True,  True]], dtype=bool)
```

`imsegm.labeling.merge_probab_labeling_2d(proba, dict_labels)`  
merging probability labeling

**Parameters**

- **proba** (*ndarray*) – probabilities
- **dict\_labels** (*dict(list(int))*) – mapping of label to histogram

**Return ndarray**

```
>>> p = np.ones((5, 5))
>>> proba = np.array([p * 0.3, p * 0.4, p * 0.2])
>>> proba = np.rollaxis(proba, 0, 3)
>>> proba.shape
(5, 5, 3)
>>> proba_new = merge_probab_labeling_2d(proba, {0: [1, 2], 1: [0]})
>>> proba_new.shape
(5, 5, 2)
>>> proba_new[0, 0]
array([ 0.6,  0.3])
```

`imsegm.labeling.neighbour_connect4(seg, label, pos)`  
check incoherent part of the segmentation

**Parameters**

- **seg** (*ndarray*) – segmentation
- **label** (*int*) – selected label
- **pos** (*tuple(int, int)*) – position

**Returns**

```
>>> neighbour_connect4(np.eye(5), 1, (2, 2))
True
>>> neighbour_connect4(np.ones((5, 5)), 1, (3, 3))
False
```

`imsegm.labeling.relabel_by_dict(labels, dict_labels)`  
relabel according given dictionary of new - old labels

**Parameters**

- **labels** (*ndarray*) –
- **dict\_labels** (*dict(list(int))*) – mapping of label to histogram

**Return ndarray**

```
>>> labels = np.array([2, 1, 0, 3, 3, 0, 2, 3, 0, 0])
>>> relabel_by_dict(labels, {0: [1, 2], 1: [0, 3]}).tolist()
[0, 0, 1, 1, 1, 1, 0, 1, 1, 1]
```

`imsegm.labeling.relabel_max_overlap_merge(seg_ref, seg_relabel, keep_bg=False)`

relabel the second segmentation cu that maximise relative overlap for each pattern (object), if one pattern in reference atlas is likely composed from multiple patterns in relabel atlas, it merge them

---

**Note:** it skips background class - 0

---

#### Parameters

- **seg\_ref** (*ndarray*) – reference segmentation
- **seg\_relabel** (*ndarray*) – segmentation for relabeling
- **keep\_bg** (*bool*) – the label 0 holds

**Return ndarray** resulting segentation

```
>>> atlas1 = np.zeros((7, 15), dtype=int)
>>> atlas1[1:4, 5:10] = 1
>>> atlas1[5:7, 3:13] = 2
>>> atlas2 = np.zeros((7, 15), dtype=int)
>>> atlas2[0:3, 7:12] = 1
>>> atlas2[3:7, 1:7] = 2
>>> atlas2[4:7, 7:14] = 3
>>> atlas2[:,2, :3] = 5
>>> relabel_max_overlap_merge(atlas1, atlas2, keep_bg=True)
array([[1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0],
       [1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0],
       [0, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0],
       [0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0],
       [0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0]])
>>> relabel_max_overlap_merge(atlas2, atlas1, keep_bg=True)
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0],
       [0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0]])
>>> relabel_max_overlap_merge(atlas1, atlas2, keep_bg=False)
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 0],
       [0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 0],
       [0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 0]])
```

`imsegm.labeling.relabel_max_overlap_unique(seg_ref, seg_relabel, keep_bg=False)`

relabel the second segmentation cu that maximise relative overlap for each pattern (object), the relation among patterns is 1-1

---

**Note:** it skips background class - 0

---

**Parameters**

- **seg\_ref** (*ndarray*) – reference segmentation
- **seg\_relabel** (*ndarray*) – segmentation for relabeling
- **keep\_bg** (*bool*) – keep the background

**Return ndarray** resulting segentation

```
>>> atlas1 = np.zeros((7, 15), dtype=int)
>>> atlas1[1:4, 5:10] = 1
>>> atlas1[5:7, 3:13] = 2
>>> atlas2 = np.zeros((7, 15), dtype=int)
>>> atlas2[0:3, 7:12] = 1
>>> atlas2[3:7, 1:7] = 2
>>> atlas2[4:7, 7:14] = 3
>>> atlas2[:,2, :3] = 5
>>> relabel_max_overlap_unique(atlas1, atlas2, keep_bg=True)
array([[5, 5, 5, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0],
       [5, 5, 5, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0],
       [0, 3, 3, 3, 3, 3, 3, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 0],
       [0, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 0],
       [0, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 0]])
>>> relabel_max_overlap_unique(atlas2, atlas1, keep_bg=True)
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 0],
       [0, 0, 0, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, 0]])
>>> relabel_max_overlap_unique(atlas1, atlas2, keep_bg=False)
array([[5, 5, 5, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0],
       [5, 5, 5, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0],
       [0, 3, 3, 3, 3, 3, 3, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 0],
       [0, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 0],
       [0, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 0]])
>>> atlas2[0, 0] = -1
>>> relabel_max_overlap_unique(atlas1, atlas2, keep_bg=True)
array([[ -1,  5,  5,  0,  0,  0,  0,  1,  1,  1,  1,  1,  0,  0,  0],
       [  5,  5,  5,  0,  0,  0,  0,  1,  1,  1,  1,  1,  0,  0,  0],
       [  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  0,  0,  0],
       [  0,  3,  3,  3,  3,  3,  3,  0,  0,  0,  0,  0,  0,  0,  0],
       [  0,  3,  3,  3,  3,  3,  3,  2,  2,  2,  2,  2,  2,  2,  0],
       [  0,  3,  3,  3,  3,  3,  3,  2,  2,  2,  2,  2,  2,  2,  0],
       [  0,  3,  3,  3,  3,  3,  3,  2,  2,  2,  2,  2,  2,  2,  0]])
```

`imsegm.labeling.segm_labels_assignment` (*segm*, *segm\_gt*)  
create labels assign to the particular regions

**Parameters**

- **segm** (*ndarray*) – input segmentation
- **segm\_gt** (*ndarray*) – true segmentation

**Returns**

```
>>> slic = np.array([[0] * 3 + [1] * 3 + [2] * 3 + [3] * 3] * 4 +
...                 [[4] * 3 + [5] * 3 + [6] * 3 + [7] * 3] * 4)
>>> segm = np.zeros(slic.shape, dtype=int)
>>> segm[4:, 6:] = 1
>>> segm_labels_assignment(slic, segm)
{0: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 1: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 2: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 3: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 4: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 5: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 6: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
 7: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}
```

`imsegm.labeling.sequence_labels_merge` (*labels\_stack*, *dict\_colors*, *labels\_free*, *change\_label=-1*)

the input is time series of labeled images and output idx labeled image with labels that was constant for all the time the special case is using free labels which can be assumed as any labeled

Example if labels series, {0, 1, 2} and 0 is free label: - 11111111 -> 1 - 11211211 -> CHANGE\_LABEL - 10111100 -> 1 - 00000000 -> CHANGE\_LABEL

**Parameters**

- **labels\_stack** (*ndarray*) – np.array<date, height, width> input stack of labeled images
- **{int – (int, int, int)}** *dict\_colors*: dictionary of labels-colors
- **labels\_free** (*list(int)*) – list of free labels
- **change\_label** (*int*) – label that is set for non constant time series

**Return ndarray** np.array<height, width>

```
>>> dict_colors = {0: [], 1: [], 2: []}
>>> sequence_labels_merge(np.zeros((8, 1, 1)), dict_colors, [0])
array([-1])
>>> sequence_labels_merge(np.ones((8, 1, 1)), dict_colors, [0])
array([1])
>>> sequence_labels_merge(np.array([[1], [1], [2], [1], [1], [1], [2], [1]]), dict_colors, [0])
array([-1])
>>> sequence_labels_merge(np.array([[1], [0], [1], [1], [1], [1], [0], [0]]), dict_colors, [0])
array([1])
```

**imsegm.pipelines module**

Pipelines for supervised and unsupervised segmentation

Copyright (C) 2014-2018 Jiri Borovec <jiri.borovec@fel.cvut.cz>

`imsegm.pipelines.compute_color2d_superpixels_features` (*image*, *dict\_features*, *sp\_size=30*, *sp\_regul=0.2*)

segment image into superpixels and estimate features per superpixel

**Parameters**



- **image** (*ndarray*) – input RGB image
- **dict\_features** (*dict(list(str))*) – list of features to be extracted
- **sp\_size** (*int*) – initial size of a superpixel(meaning edge length)
- **sp\_regul** (*float*) – regularisation in range(0;1) where “0” gives elastic and “1” nearly square segments

**Return** `list(list(int)), [[floats]]` superpixels and related of features

```
imsegm.pipelines.estim_model_classes_group(list_images, nb_classes, dict_features,
                                           sp_size=30, sp_regul=0.2, use_scaler=True,
                                           pca_coef=None, model_type='GMM',
                                           nb_workers=1)
```

estimate a model from sequence of input images and return it as result

#### Parameters

- **list\_images** (*[ndarray]*) –
- **nb\_classes** (*int*) – number of classes
- **sp\_size** (*int*) – initial size of a superpixel(meaning edge length)
- **sp\_regul** (*float*) – regularisation in range(0;1) where “0” gives elastic and “1” nearly square slic
- **dict\_features** (*dict(list(str))*) – list of features to be extracted
- **pca\_coef** (*float*) – range (0, 1) or None
- **use\_scaler** (*bool*) – whether use a scaler
- **model\_type** (*str*) – model type
- **nb\_workers** (*int*) – number of jobs running in parallel

#### Returns

```
imsegm.pipelines.pipe_color2d_slic_features_model_graphcut(image, nb_classes,
                                                            dict_features,
                                                            sp_size=30,
                                                            sp_regul=0.2,
                                                            pca_coef=None,
                                                            use_scaler=True,
                                                            estim_model='GMM',
                                                            gc_regul=1.0,
                                                            gc_edge_type='model',
                                                            debug_visual=None)
```

complete pipe-line for segmentation using superpixels, extracting features and graphCut segmentation

#### Parameters

- **image** (*ndarray*) – input RGB image
- **nb\_classes** (*int*) – number of classes to be segmented(indexing from 0)
- **sp\_size** (*int*) – initial size of a superpixel(meaning edge length)
- **sp\_regul** (*float*) – regularisation in range(0,1) where 0 gives elastic and 1 nearly square slic
- **dict\_features** (*dict*) – {clr: list(str)}
- **pca\_coef** (*float*) – range (0, 1) or None

- **estim\_model** (*str*) – estimating model
- **gc\_regul** (*float*) – GC regularisation
- **gc\_edge\_type** (*str*) – graphCut edge type
- **use\_scaler** (*bool*) – using scaler block in pipeline
- **debug\_visual** – dict

**Return** `list(list(int))` segmentation matrix mapping each pixel into a class

```
>>> np.random.seed(0)
>>> image = np.random.random((125, 150, 3)) / 2.
>>> image[:, :75] += 0.5
>>> segm, seg_soft = pipe_color2d_slic_features_model_graphcut(image, 2, {'color':
↪ ['mean']})
>>> segm.shape
(125, 150)
>>> seg_soft.shape
(125, 150, 2)
```

```
imsegm.pipelines.pipe_gray3d_slic_features_model_graphcut(image, nb_classes,
dict_features,
spacing=(12, 1, 1), sp_size=15,
sp_regul=0.2,
gc_regul=0.1)
```

complete pipe-line for segmentation using superpixels, extracting features and graphCut segmentation

#### Parameters

- **image** (*ndarray*) – input RGB image
- **sp\_size** (*int*) – initial size of a superpixel(meaning edge lenght)
- **sp\_regul** (*float*) – regularisation in range(0;1) where “0” gives elastic and “1” nearly square segments
- **nb\_classes** (*int*) – number of classes to be segmented(indexing from 0)
- **spacing** (*tuple(int, int, int)*) –
- **gc\_regul** (*float*) – regularisation for GC

**Return** `list(list(int))` segmentation matrix mapping each pixel into a class

```
>>> np.random.seed(0)
>>> image = np.random.random((5, 125, 150)) / 2.
>>> image[:, :, :75] += 0.5
>>> segm = pipe_gray3d_slic_features_model_graphcut(image, 2, {'color': ['mean']})
>>> segm.shape
(5, 125, 150)
```

```
imsegm.pipelines.segment_color2d_slic_features_model_graphcut(image,
model_pipeline,
dict_features,
sp_size=30,
sp_regul=0.2,
gc_regul=1.0,
gc_edge_type='model',
debug_visual=None)
```

complete pipe-line for segmentation using superpixels, extracting features and graphCut segmentation

### Parameters

- **image** (*ndarray*) – input RGB image
- **model\_pipeline** (*obj*) –
- **sp\_size** (*int*) – initial size of a superpixel(meaning edge lenght)
- **sp\_regul** (*float*) – regularisation in range(0;1) where “0” gives elastic and “1” nearly square slic
- **dict\_features** (*dict(list(str))*) – list of features to be extracted
- **gc\_regul** (*float*) – GC regularisation
- **gc\_edge\_type** (*str*) – select the GC edge type
- **debug\_visual** – dict

**Return** `list(list(int))` segmentation matrix mapping each pixel into a class

### Examples

```
>>> # UnSupervised:
>>> import imsegm.descriptors as seg_fts
>>> np.random.seed(0)
>>> seg_fts.USE_CYTHON = False
>>> image = np.random.random((125, 150, 3)) / 2.
>>> image[:, :75] += 0.5
>>> model, _ = estim_model_classes_group([image], 2, {'color': ['mean']})
>>> segm, seg_soft = segment_color2d_slic_features_model_graphcut(image, model, {
↪ 'color': ['mean']})
>>> segm.shape
(125, 150)
>>> seg_soft.shape
(125, 150, 2)
```

```
>>> # Supervised:
>>> import imsegm.descriptors as seg_fts
>>> np.random.seed(0)
>>> seg_fts.USE_CYTHON = False
>>> image = np.random.random((125, 150, 3)) / 2.
>>> image[:, 75:] += 0.5
>>> annot = np.zeros(image.shape[:2], dtype=int)
>>> annot[:, 75:] = 1
>>> clf, _, _ = train_classif_color2d_slic_features([image], [annot], {'color':
↪ ['mean']})
>>> segm, seg_soft = segment_color2d_slic_features_model_graphcut(image, clf, {
↪ 'color': ['mean']})
>>> segm.shape
(125, 150)
>>> seg_soft.shape
(125, 150, 2)
```

```
imsegm.pipelines.train_classif_color2d_slic_features(list_images,      list_annots,
                                                    dict_features,      sp_size=30,
                                                    sp_regul=0.2,
                                                    clf_name='RandForest',
                                                    label_purity=0.9,      fea-
                                                    feature_balance='unique',
                                                    pca_coef=None,
                                                    nb_classif_search=1,
                                                    nb_hold_out=2,
                                                    nb_workers=1)
```

train classifier on list of annotated images

#### Parameters

- **list\_images** (*[ndarray]*) –
- **list\_annots** (*[ndarray]*) –
- **sp\_size** (*int*) – initial size of a superpixel(meaning edge lenght)
- **sp\_regul** (*float*) – regularisation in range(0;1) where “0” gives elastic and “1” nearly square segments
- **dict\_features** (*dict(list(str))*) – list of features to be extracted
- **clf\_name** (*str*) – selet udsed classifier
- **label\_purity** (*float*) – set the sample-labels purity for training
- **feature\_balance** (*str*) – set how to balance datasets
- **pca\_coef** (*float*) – select PCA coef or None
- **nb\_classif\_search** (*int*) – number of tries for hyper-parameters seach
- **nb\_hold\_out** (*int*) – cross-val leave out
- **nb\_workers** (*int*) – parallelism

#### Returns

```
imsegm.pipelines.wrapper_compute_color2d_slic_features_labels(img_annot,
                                                             sp_size, sp_regul,
                                                             dict_features,
                                                             label_purity)
```

```
imsegm.pipelines.CLASSIF_NAME = 'RandForest'
    select default Classifier for supervised segmentation
```

```
imsegm.pipelines.CLUSTER_METHOD = 'kMeans'
    select default Modeling/clustering for unsupervised segmentation
```

```
imsegm.pipelines.CROSS_VAL_LEAVE_OUT = 2
    define how many images will be left out during cross-validation training
```

```
imsegm.pipelines.FTS_SET_SIMPLE = {'color': ('mean', 'std', 'energy')}
    select basic features extracted from superpixels
```

```
imsegm.pipelines.NB_WORKERS = 1
    default number of workers
```

## imsegm.region\_growing module

### Framework for region growing

- general GraphCut segmentation with and without shape model
- region growing with shape prior - greedy & GraphCut

Copyright (C) 2016-2018 Jiri Borovec <jiri.borovec@fel.cvut.cz>

`imsegm.region_growing.compute_centre_moment_points` (*points*)  
compute centre and moment from set of points

**Parameters** `float[]` *points* (`[float,)` –

**Returns**

```
>>> points = list(zip([0] * 10, np.arange(10))) + [(0, 0)] * 5
>>> compute_centre_moment_points(points)
(array([ 0.,  3.]), 0.0)
>>> points = list(zip(np.arange(10), [0] * 10)) + [(10, 0)]
>>> compute_centre_moment_points(points)
(array([ 5.,  0.]), 90.0)
>>> points = list(zip(-np.arange(10), -np.arange(10))) + [(0, 0)] * 5
>>> compute_centre_moment_points(points)
(array([-3., -3.]), 45.0)
>>> points = list(zip(-np.arange(10), np.arange(10))) + [(-10, 10)]
>>> compute_centre_moment_points(points)
(array([-5.,  5.]), 135.0)
```

`imsegm.region_growing.compute_cumulative_distrib` (*means, stds, weights, max\_dist*)  
compute invers cumulative distribution based given means, covariance and weights for each segment

**Parameters**

- **means** (`[float]`) – mean values for each model and ray direction
- **stds** (`[float]`) – STD for each model and ray direction
- **weights** (`[float]`) – model wights
- **max\_dist** (`float`) – maxim distance for shape model

**Return** `[float]`

```
>>> cdist = compute_cumulative_distrib(np.array([[1, 2]]),
...                                   np.array([[1.5, 0.5], [0.5, 1]]), np.array([0.5]), 6)
>>> np.round(cdist, 2)
array([[ 1. ,  0.67,  0.34,  0.12,  0.03,  0. ,  0. ],
       [ 1. ,  0.98,  0.5 ,  0.02,  0. ,  0. ,  0. ]])
```

`imsegm.region_growing.compute_data_costs_points` (*slic, slic\_prob\_fg, centres, labels*)  
compute Look up Table ro date term costs

**Parameters**

- **slic** (*nadarray*) – superpixel segmentation
- **slic\_prob\_fg** (*list float*) – weight for particular pixel belongs to FG
- **int[]** *centres* (`[int,)` – actual centre position
- **labels** (*list int*) – labels for points to be assigned to an object

**Returns**

```
imsegm.region_growing.compute_object_shapes(list_img_objects, ray_step=5, in-  
                                           interp_order=3, smooth_coef=0,  
                                           shift_method='phase')
```

for all object in all images compute gravity center and Ray beatures (if object are not split already by different label is made here)

#### Parameters

- **list\_img\_objects** (*[narray]*) – list of binary segmentation
- **ray\_step** (*int*) – select the angular step for Ray features
- **interp\_order** (*int*) – if None, no interpolation is performed
- **smooth\_coef** (*float*) – smoothing the ray features
- **shift\_method** (*str*) – use method for estimate shift maxima (phase or max)

**Return tuple(list(list(int)),list(int))**

```
>>> img1 = np.zeros((100, 100))  
>>> img1[20:50, 30:60] = 1  
>>> img1[40:80, 50:90] = 2  
>>> img2 = np.zeros((100, 100))  
>>> img2[10:40, 20:50] = 1  
>>> img2[50:80, 20:50] = 1  
>>> img2[50:80, 60:90] = 1  
>>> list_imgs = [img1, img2]  
>>> list_rays, list_shifts = compute_object_shapes(list_imgs, ray_step=45)  
>>> np.array(list_rays).astype(int)  
array([[19, 17,  9, 17, 19, 14, 19, 14],  
       [29, 21, 28, 20, 28, 20, 28, 21],  
       [22, 16, 21, 15, 21, 15, 21, 16],  
       [22, 16, 21, 15, 21, 15, 21, 16],  
       [22, 16, 21, 15, 21, 15, 21, 16]])  
>>> np.array(list_shifts) % 180  
array([ 135.,  45.,  45.,  45.,  45.] )
```

```
imsegm.region_growing.compute_pairwise_penalty(edges, labels, prob_bg_fg=0.05,  
                                                prob_fg1_fg2=0.01)
```

compute cost of neighboring labels pions

#### Parameters

- **int)] edges** (*[int,)*) – graph edges, connectivity
- **labels** (*[int]*) – labels for vertexes
- **prob\_bg\_fg** (*float*) – penalty between background and foreground
- **prob\_fg1\_fg2** (*float*) – penalty between two different foreground classes

#### Returns

```
>>> edges = np.array([[0, 1], [1, 2], [0, 3], [2, 3], [2, 4]])  
>>> labels = np.array([0, 0, 1, 2, 1])  
>>> compute_pairwise_penalty(edges, labels, 0.05, 0.01)  
array([ 0.          ,  2.99573227,  2.99573227,  4.60517019,  0.          ])
```

```
imsegm.region_growing.compute_rg_crit(labels, lut_data_cost, lut_shape_cost, slic_weights,  
                                       edges, coef_data, coef_shape, coef_pairwise,  
                                       prob_label_trans)
```

`imsegm.region_growing.compute_segmn_object_shape` (*img\_object*, *ray\_step=5*, *interp\_order=3*, *smooth\_coef=0*, *shift\_method='phase'*)

assuming single object in image and compute gravity centre and for this point compute Ray features and optionally: - interpolate missing values - smooth the Ray features

#### Parameters

- **img\_object** (*ndarray*) – binary segmentation of single object
- **ray\_step** (*int*) – select the angular step for Ray features
- **interp\_order** (*int*) – if None, no interpolation is performed
- **smooth\_coef** (*float*) – smoothing the ray features
- **shift\_method** (*str*) – use method for estimate shift maxima (phase or max)

**Return** tuple(list(int), int)

```
>>> img = np.zeros((100, 100))
>>> img[20:70, 30:80] = 1
>>> rays, shift = compute_segmn_object_shape(img, ray_step=45)
>>> rays
[36.7..., 26.0..., 35.3..., 25.0..., 35.3..., 25.0..., 35.3..., 26.0...]
```

`imsegm.region_growing.compute_segmn_prob_fg` (*slic*, *segm*, *labels\_prob*)

compute probability being foreground from input segmentation

#### Parameters

- **slic** (*ndarray*) –
- **segm** (*ndarray*) –
- **labels\_prob** (*list(float)*) –

#### Returns

```
>>> slic = np.array([[0, 0, 0, 0, 1, 1, 1, 1], [2, 2, 2, 2, 3, 3, 3, 3]])
>>> segm = np.array([0, 1, 1, 0])[slic]
>>> compute_segmn_prob_fg(slic, segm, [0.3, 0.8])
array([ 0.3, 0.8, 0.8, 0.3])
```

`imsegm.region_growing.compute_shape_prior_table_cdf` (*point*, *cum\_distribution*, *centre*, *angle\_shift=0*)

compute shape prior for a point based on centre, rotation shift and cumulative histogram

#### Parameters

- **point** (*tuple(int, int)*) – single points
- **centre** (*tuple(int, int)*) – center of model
- **cum\_distribution** (*[[float]]*) – cumulative histogram
- **angle\_shift** (*float*) –

**Return** float

```
>>> chist = [[1.0, 1.0, 0.8, 0.7, 0.6, 0.5, 0.3, 0.0, 0.0],
...          [1.0, 1.0, 0.9, 0.8, 0.7, 0.3, 0.2, 0.2, 0.0],
...          [1.0, 1.0, 1.0, 0.7, 0.6, 0.5, 0.3, 0.1, 0.1],
...          [1.0, 1.0, 0.6, 0.5, 0.4, 0.3, 0.2, 0.0, 0.0]]
```

(continues on next page)

(continued from previous page)

```

>>> centre = (1, 1)
>>> compute_cdf = compute_shape_prior_table_cdf
>>> compute_cdf([1, 1], chist, centre)
1.0
>>> compute_cdf([10, 10], chist, centre)
0.0
>>> compute_cdf([10, -10], chist, centre)
0.100...
>>> compute_cdf([2, 3], chist, centre)
0.805...
>>> compute_cdf([-3, -2], chist, centre)
0.381...
>>> compute_cdf([3, -2], chist, centre)
0.676...
>>> compute_cdf([2, 3], chist, centre, angle_shift=270)
0.891...

```

`imsegm.region_growing.compute_update_shape_costs_points_close_mean_cdf` (*lut\_shape\_cost*, *slic*, *points*, *la-*  
*bels*, *init\_centres*, *cen-*  
*tres*, *shifts*, *vol-*  
*umes*, *shape\_model\_cdfs*, *se-*  
*lected\_idx=None*, *swap\_shift=False*, *dict\_thresholds=None*)

update the shape prior for given segmentation (new centre is computed), set of points and cumulative histogram representing the shape model

#### Parameters

- **lut\_shape\_cost** – look-up-table for shape cost for GC
- **slic** (*ndarray*) – superpixel segmentation
- **int]] points** (*[[int,)* – subsample space, points = superpixel centres
- **labels** (*list(int)*) – labels for points to be assigned to an object
- **int]] init\_centres** (*[[int,)* – initial centre position for compute center shift during the iterations
- **int]] centres** (*[[int,)* – actual centre position
- **shifts** (*list(int)*) – orientation for each region / object
- **volumes** (*list(int)*) – size / volume for each region
- **shape\_model\_cdfs** – represent the shape prior and histograms
- **selected\_idx** (*list(int)*) – selected object for update



- **swap\_shift** (*bool*) – allow swapping orientation by 90 degree, try to get out from local optimal
- **dict\_thresholds** (*dict / None*) – configuration with thresholds
- **dict\_thresholds** – set some threshold updating shape prior

**Return tuple(list(float),list(int))**

```
>>> np.random.seed(0)
>>> h, w, step = 8, 8, 2
>>> slic = np.array([[ 0,  0,  1,  1,  2,  2,  3,  3],
...                  [ 0,  0,  1,  1,  2,  2,  3,  3],
...                  [ 4,  4,  5,  5,  6,  6,  7,  7],
...                  [ 4,  4,  5,  5,  6,  6,  7,  7],
...                  [ 8,  8,  9,  9, 10, 10, 11, 11],
...                  [ 8,  8,  9,  9, 10, 10, 11, 11],
...                  [12, 12, 13, 13, 14, 14, 15, 15],
...                  [12, 12, 13, 13, 14, 14, 15, 15]])
>>> points = np.array([(0, 0), (0, 2), (0, 4), (0, 6), (2, 0), (2, 2),
...                   (2, 4), (2, 6), (4, 0), (4, 2), (4, 4), (4, 6),
...                   (6, 0), (6, 2), (6, 4), (6, 6)])
>>> labels = np.array([0] * 4 + [0, 1, 1, 0, 0, 1, 1, 0] + [0] * 4)
>>> cdf1, cdf2 = np.zeros((8, 10)), np.zeros((8, 7))
>>> cdf1[:,7] = 0.5
>>> cdf1[:,4] = 1.0
>>> cdf2[:,6] = 1.0
>>> set_m_cdf = [(4] * 8, cdf1), ([5] * 8, cdf2)]
>>> s_costs = np.zeros((len(points), 2))
>>> mm = mixture.GaussianMixture(2).fit(np.random.random((100, 8)))
>>> s_costs, centres, shifts, _ = compute_update_shape_costs_points_close_mean_
    ↪cdf(
...             s_costs, slic, points, labels, [(0, 0)],
...             [(np.Inf, np.Inf)], [0], [0], (mm, set_m_cdf))
>>> centres
array([[3, 3]])
>>> shifts
array([ 90.])
>>> np.round(s_costs, 3)
array([[ 0.    , -0.01 ],
       [ 0.    , -0.01 ],
       [ 0.    , -0.01 ],
       [ 0.    , -0.01 ],
       [ 0.    , -0.01 ],
       [ 0.    , -0.01 ],
       [ 0.    , -0.01 ],
       [ 0.    ,  0.868],
       [ 0.    , -0.01 ],
       ...
       [ 0.    ,  4.605]])
```

```
imsegm.region_growing.compute_update_shape_costs_points_table_cdf(lut_shape_cost,
                                                                    points,
                                                                    labels,
                                                                    init_centres,
                                                                    centres,
                                                                    shifts,
                                                                    volumes,
                                                                    shape_chist,
                                                                    selected_idx=None,
                                                                    swap_shift=False,
                                                                    dict_thresholds=None)
```

update the shape prior for given segmentation (new centre is computed), set of points and cumulative histogram representing the shape model

#### Parameters

- **lut\_shape\_cost** – look-up-table for shape cost for GC
- **int[] points** (*[[int,)* – subsample space, points = superpixel centres
- **labels** (*list(int)*) – labels for points to be assigned to an object
- **int[] init\_centres** (*[[int,)* – initial centre position for compute center shift during the iterations
- **int[] centres** (*[[int,)* – actual centre position
- **shifts** (*list(int)*) – orientation for each region / object
- **volumes** (*list(int)*) – size / volume for each region
- **shape\_chist** – represent the shape prior and histograms
- **selected\_idx** (*list(int)*) – selected object for update
- **swap\_shift** (*bool*) – allow swapping orientation by 90 degree, try to get out from local optimal
- **dict\_thresholds** (*dict|None*) – configuration with thresholds
- **dict\_thresholds** – set some threshold updating shape prior

Return tuple(list(float),list(int))

```
>>> cdf = np.zeros((8, 20))
>>> cdf[:10] = 0.5
>>> cdf[:4] = 1.0
>>> points = np.array([[13, 16], [1, 5], [10, 15], [15, 25], [10, 5]])
>>> labels = np.ones(len(points))
>>> s_costs = np.zeros((len(points), 2))
>>> s_costs, centres, shifts, _ = compute_update_shape_costs_points_table_cdf(
...     s_costs, points, labels, [(0, 0)], [(np.Inf, np.Inf)], [0], [0], (None,
↪cdf))
>>> centres
array([[10, 13]])
>>> shifts
array([ 209.])
>>> np.round(s_costs, 3)
array([[ 0.   ,  0.673],
       [ 0.   , -0.01 ],
       [ 0.   ,  0.184],
```

(continues on next page)

(continued from previous page)

```

    [ 0.    ,  0.543],
    [ 0.    ,  0.374]])
>>> dict_thrs = RG2SP_THRESHOLDS
>>> dict_thrs['centre_init'] = 1
>>> _, centres, _, _ = compute_update_shape_costs_points_table_cdf(
...     s_costs, points, labels, [(7, 18)], [(np.Inf, np.Inf)], [0], [0], (None, _
↪cdf),
...     dict_thresholds=dict_thrs)
>>> np.round(centres, 1)
array([[ 7.5, 17.1]])

```

`imsegm.region_growing.enforce_center_labels` (*slic, labels, centres*)  
force the labels to hold label of the center, prevention of desepearing labels of any center in list

#### Parameters

- **slic** –
- **labels** –
- **centres** –

#### Returns

`imsegm.region_growing.get_neighboring_candidates` (*slic\_neighbours, labels, object\_idx, use\_other\_obj=True*)  
get neighboring candidates from background and optionally also from foreground if it is allowed

#### Parameters

- **slic\_neighbours** (*[[int]]*) – list of neighboring superpixel for each one
- **labels** (*[[int]]*) – labels for each superpixel
- **object\_idx** (*int*) –
- **use\_other\_obj** (*bool*) – allowing use another foreground object

#### Return [int]

```

>>> neighbours = [[1], [0, 2, 3], [1, 3], [1, 2]]
>>> labels = np.array([0, 0, 1, 1])
>>> get_neighboring_candidates(neighbours, labels, 1)
[1]

```

`imsegm.region_growing.object_segmentation_graphcut_pixels` (*segm, centres, labels\_fg\_prob=(0.1, 0.9), gc\_regul=1, seed\_size=0, coef\_shape=0.0, shape\_mean\_std=(50.0, 10.0), debug\_visual=None*)

object segmentation using Graph Cut directly on pixel level

#### Parameters

- **centres** (*ndarray*) –
- **segm** (*ndarray*) – input structure segmentation
- **int)] centres** (*[[int,)* – superpixel centres

- **labels\_fg\_prob** (*list(float)*) – set how much particular label belongs to foreground
- **gc\_regul** (*float*) – regularisation for GC
- **seed\_size** (*int*) – create circular neighborhood around initial centre
- **coef\_shape** (*float*) – set the weight of shape prior
- **shape\_mean\_std** – mean and STD for shape prior
- **debug\_visual** (*dict*) – dictionary with some intermediate results

Return `list(list(int))`

```
>>> segm = np.array([[0] * 10,
...                  [1] * 5 + [0] * 5, [1] * 4 + [0] * 6,
...                  [0] * 6 + [1] * 4, [0] * 5 + [1] * 5,
...                  [0] * 10])
>>> centres = [(1, 2), (4, 8)]
>>> object_segmentation_graphcut_pixels(segm, centres, gc_regul=0., coef_shape=0.
↪5)
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [2, 2, 1, 2, 2, 0, 0, 0, 0, 0],
       [2, 2, 2, 2, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 2, 2, 2, 2],
       [0, 0, 0, 0, 0, 2, 2, 2, 2, 2],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]], dtype=int32)
>>> object_segmentation_graphcut_pixels(segm, centres, gc_regul=.5, seed_size=1)
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 2, 2, 2, 2],
       [0, 0, 0, 0, 0, 2, 2, 2, 2, 2],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]], dtype=int32)
```

```
imsegm.region_growing.object_segmentation_graphcut_slic(slic, segm, centres,
                                                         labels_fg_prob=(0.1,
                                                         0.9), gc_regul=1,
                                                         edge_coef=0.5,
                                                         edge_type='model',
                                                         coef_shape=0.0,
                                                         shape_mean_std=(50.0,
                                                         10.0),
                                                         add_neighbours=False,
                                                         debug_visual=None)
```

object segmentation using Graph Cut directly on super-pixel level

#### Parameters

- **slic** (*ndarray*) – superpixel pre-segmentation
- **segm** (*ndarray*) – input structure segmentation
- **int)] centres** (*[(int,)]*) – superpixel centres
- **labels\_fg\_prob** (*list(float)*) – weight for particular label belongs to FG
- **gc\_regul** (*float*) – regularisation for GC
- **edge\_coef** (*float*) – weight of edges on GC
- **edge\_type** (*str*) – select the edge weights on graph

- **coef\_shape** (*float*) – set the weight of shape prior
- **shape\_mean\_std** – mean and STD for shape prior
- **add\_neighbours** (*bool*) – add also neighboring supepixels to the center
- **debug\_visual** (*dict*) – dictionary with some intermediate results

**Return** list(list(int))

```
>>> slic = np.array([[0] * 3 + [1] * 3 + [2] * 3 + [3] * 3 + [4] * 3,
...                  [5] * 3 + [6] * 3 + [7] * 3 + [8] * 3 + [9] * 3])
>>> segm = np.array([[0] * 15, [1] * 12 + [0] * 3])
>>> object_segmentation_graphcut_slic(slic, segm, [(1, 7)],
...                                   gc_regul=0., edge_coef=1., coef_shape=1.)
array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0], dtype=int32)
>>> object_segmentation_graphcut_slic(slic, segm, [(1, 7)],
...                                   gc_regul=1., edge_coef=1., debug_visual={})
array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0], dtype=int32)
```

`imsegm.region_growing.prepare_graphcut_variables` (*candidates*, *slic\_points*,  
*slic\_neighbours*, *slic\_weights*, *la-*  
*bels*, *nb\_centres*, *lut\_data\_cost*,  
*lut\_shape\_cost*, *coef\_data*,  
*coef\_shape*, *coef\_pairwise*,  
*prob\_label\_trans*)

for boundary get connected points in BG and FG construct graph and set potentials and hard connect BG and FG in unary

#### Parameters

- **candidates** (*[int]*) – list of candidates, neighbours of actual objects
- **int)] slic\_points** (*[(int,)* –
- **slic\_neighbours** (*[[int]]*) – list of neighboring superpixel for each one
- **slic\_weights** (*list(float)*) – weight for each superpixel
- **labels** (*[int]*) – labels for each superpixel
- **nb\_centres** (*int*) – number of centres - classes
- **lut\_data\_cost** (*ndarray*) – look-up-table for data cost for each object (class) with superpixel as first index
- **lut\_shape\_cost** (*ndarray*) – look-up-table for shape cost for each object (class) with superpixel as first index
- **coef\_data** (*float*) – weight for data priors
- **coef\_shape** (*float*) – weight for shape priors
- **coef\_pairwise** (*float*) – CG pairwise coefficient
- **prob\_label\_trans** – probability transition between background (first) and objects and among objects (second)

#### Returns

## Parameters

- **slic** (*ndarray*) – superpixel segmentation
- **slic\_prob\_fg** (*list (float)*) – weight for particular superpixel belongs to FG
- **int)] centres** (*[(int,)* – list of initial centres
- **shape\_model** – represent the shape prior and histograms
- **shape\_type** (*str*) – identification of used shape model
- **coef\_data** (*float*) – weight for data prior
- **coef\_shape** (*float*) – weight for shape prior
- **coef\_pairwise** (*float*) – setting for pairwise cost
- **prob\_label\_trans** – probability transition between background (first) and objects and among objects (second)
- **optim\_global** (*bool*) – optimise the GC as global or per object
- **allow\_obj\_swap** (*bool*) – allow swapping foreground object labels
- **dict\_thresholds** (*dict / None*) – configuration with thresholds
- **nb\_iter** (*int*) – maximal number of iterations
- **dict\_thresholds** – set some threshold updating shape prior

[illegible]

## Chapter 1. Contents

```
[0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]

>>> slic = np.zeros((h, w), dtype=int)
>>> for i in range(int(np.ceil(h / float(step)))):
...     for j in range(int(np.ceil(w / float(step)))):
...         val = i * (w / step) + j
...         i_step, j_step = int(i * step), int(j * step)
...         slic[i_step:int(i_step + step), j_step:int(j_step + step)] = val
>>> centres = [(7.5, 10)]
>>> chist = [[1.] * 3 + [0.8, 0.7, 0.6, 0.5, 0.3, 0.1, 0.0],
...          [1.] * 3 + [0.9, 0.8, 0.7, 0.3, 0.2, 0.2, 0.1],
...          [1.] * 3 + [1.0, 0.7, 0.6, 0.5, 0.3, 0.1, 0.1],
...          [1.] * 3 + [0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.0]]
>>> dict_debug = {}
>>> slic_prob_fg = compute_segm_prob_fg(slic, segm, [0.1, 0.9])
>>> labels = region_growing_shape_slic_graphcut(slic, slic_prob_fg, centres,
...                                              (None, chist), coef_pairwise=0,
...                                              debug_history=dict_debug)
>>> np.round(dict_debug['criteria']).astype(int)
array([397, 325, 206, 111, 81, 81])
>>> labels[slic]
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
>>> labels = region_growing_shape_slic_graphcut(slic, slic_prob_fg, centres,
...                                              (None, chist), coef_pairwise=2,
...                                              debug_history=dict_debug)
>>> np.round(dict_debug['criteria']).astype(int)
array([415, 380, 289, 193, 164, 164])
>>> labels[slic]
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
```

## 1.2. imsegm package

(continued from previous page)

```

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
>>> segm = np.ones((h, w), dtype=int)
>>> chist = np.zeros((16, 9))
>>> chist[:, :5] = 1.
>>> dict_debug = {}
>>> slic_prob_fg = compute_segm_prob_fg(slic, segm, [0.1, 0.9])
>>> labels = region_growing_shape_slic_graphcut(slic, slic_prob_fg, [(6.5, 9)],
...                                             (None, chist), coef_shape=10.,
...                                             coef_pairwise=1,
...                                             debug_history=dict_debug)
>>> np.round(dict_debug['criteria']).astype(int)
array([7506, 7120, 6328, 5719, 5719])
>>> labels[slic]
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])

```

```

imsegm.region_growing.region_growing_shape_slic_greedy(slic, slic_prob_fg, centres,
                                                         shape_model,
                                                         shape_type='cdf',
                                                         coef_data=1.0,
                                                         coef_shape=1,
                                                         coef_pairwise=1,
                                                         prob_label_trans=(0.1,
                                                         0.01),
                                                         al-
                                                         low_obj_swap=True,
                                                         greedy_tol=0.001,
                                                         dict_thresholds=None,
                                                         nb_iter=999,
                                                         de-
                                                         bug_history=None)

```

Region growing method with given shape prior on pre-segmented images it uses the Greedy strategy and set some stopping criterion

#### Parameters

- **slic** (*ndarray*) – superpixel segmentation
- **slic\_prob\_fg** (*list(float)*) – weight for particular superpixel belongs to FG
- **int)] centres** (*[(int,)* – list of initial centres
- **shape\_model** – represent the shape prior and histograms
- **shape\_type** (*str*) – identification of used shape model
- **coef\_data** (*float*) – weight for data prior



- **coef\_shape** (*float*) – weight for shape prior
- **coef\_pairwise** (*float*) – setting for pairwise cost
- **prob\_label\_trans** – probability transition between background (first) and objects and among objects (second)
- **allow\_obj\_swap** (*bool*) – allow swapping foreground object labels
- **greedy\_tol** (*float*) – stopping criterion - energy change between inters
- **dict\_thresholds** (*dict* / *None*) – configuration with thresholds
- **nb\_iter** (*int*) – maximal number of iterations
- **dict\_thresholds** – set some threshold updating shape prior

### Returns

```
>>> np.random.seed(0)
>>> h, w, step = 15, 20, 2
>>> segm = np.zeros((h, w), dtype=int)
>>> segm[3:12, 5:17] = 1
>>> segm
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
>>> slic = np.zeros((h, w), dtype=int)
>>> for i in range(int(np.ceil(h / float(step)))):
...     for j in range(int(np.ceil(w / float(step)))):
...         val = i * (w / step) + j
...         i_step, j_step = int(i * step), int(j * step)
...         slic[i_step:int(i_step + step), j_step:int(j_step + step)] = val
>>> centres = [(7.5, 10)]
>>> chist = [[1.] * 3 + [0.8, 0.7, 0.6, 0.5, 0.3, 0.1, 0.0],
...          [1.] * 3 + [0.9, 0.8, 0.7, 0.3, 0.2, 0.2, 0.1],
...          [1.] * 3 + [1.0, 0.7, 0.6, 0.5, 0.3, 0.1, 0.1],
...          [1.] * 3 + [0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0.0]]
>>> dict_debug = {}
>>> slic_prob_fg = compute_segm_prob_fg(slic, segm, [0.1, 0.9])
>>> labels = region_growing_shape_slic_greedy(slic, slic_prob_fg, centres,
...                                           (None, chist), coef_pairwise=0,
...                                           debug_history=dict_debug)
>>> np.round(dict_debug['criteria']).astype(int)
array([397, 325, 307, 289, 272, 238, 204, 188, 173, ..., 81, 81])
>>> labels[slic]
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
```

(continues on next page)

(continues on next page)

(continued from previous page)

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

`imsegm.region_growing.transform_rays_model_cdf_histograms(list_rays, nb_bins=10)`  
 from list of all measured rays create cumulative histogram for each ray

**Parameters**

- **list\_rays** (`list(list(int))`) – list ray features (distances)
- **nb\_bins** (`int`) – binarise histogram

**Returns**

```
>>> list_rays = [[9, 4, 9], [4, 9, 7], [9, 7, 11], [10, 8, 10],
...             [9, 11, 8], [4, 8, 5], [8, 10, 6], [9, 7, 11]]
>>> chist = transform_rays_model_cdf_histograms(list_rays, nb_bins=5)
>>> chist
[[1.0, 1.0, 1.0, 1.0, 0.75, 0.75, 0.75, 0.625, 0.625, 0.0, 0.0, 0.0],
 [1.0, 1.0, 1.0, 1.0, 0.875, 0.875, 0.875, 0.375, 0.25, 0.25, 0.0, 0.0],
 [1.0, 1.0, 1.0, 1.0, 1.0, 0.75, 0.625, 0.5, 0.375, 0.375, 0.0, 0.0]]
```

`imsegm.region_growing.transform_rays_model_cdf_kmeans(list_rays,`  
`nb_components=None)`  
 compute the mixture model and transform it into cumulative distribution

**Parameters**

- **list\_rays** (`list(list(int))`) – list ray features (distances)
- **nb\_components** (`int`) – number components in mixture model

**Return any, list(list(int))** mixture model, list of stat/param of models

```
>>> np.random.seed(0)
>>> list_rays = [[9, 4, 9], [4, 9, 7], [9, 7, 11], [10, 8, 10],
...             [9, 11, 8], [4, 8, 5], [8, 10, 6], [9, 7, 11]]
>>> mm, cdist = transform_rays_model_cdf_kmeans(list_rays)
>>> np.round(cdist, 1).tolist()
[[1.0, 1.0, 1.0, 1.0, 0.9, 0.8, 0.7, 0.7, 0.6, 0.4, 0.2, 0.0, 0.0],
 [1.0, 1.0, 1.0, 1.0, 0.9, 0.9, 0.8, 0.7, 0.5, 0.3, 0.2, 0.1, 0.0],
 [1.0, 1.0, 1.0, 1.0, 1.0, 0.9, 0.8, 0.7, 0.5, 0.4, 0.2, 0.1, 0.0]]
>>> mm, cdist = transform_rays_model_cdf_kmeans(list_rays, nb_components=2)
```

`imsegm.region_growing.transform_rays_model_cdf_mixture(list_rays,`  
`coef_components=1)`  
 compute the mixture model and transform it into cumulative distribution

**Parameters**

- **list\_rays** (`list(list(int))`) – list ray features (distances)
- **coef\_components** (`int`) – multiplication for number of components

**Return any, list(list(int))** mixture model, cumulative distribution

```
>>> np.random.seed(0)
>>> list_rays = [[9, 4, 9], [4, 9, 7], [9, 7, 11], [10, 8, 10],
...             [9, 11, 8], [4, 8, 5], [8, 10, 6], [9, 7, 11]]
>>> mm, cdist = transform_rays_model_cdf_mixture(list_rays)
>>> # the rounding variate a bit according GMM estimated model
```

(continues on next page)

(continued from previous page)

```
>>> np.round(np.array(cdist) * 4) / 4.
array([[ 1. ,  1. ,  1. ,  1. ,  1. ,  1. ,  0.75,  0.75,  0.5 ,  0.25,  0. ],
       [ 1. ,  1. ,  1. ,  1. ,  1. ,  1. ,  1. ,  0.75,  0.5 ,  0.25,  0. ],
       [ 1. ,  1. ,  1. ,  1. ,  1. ,  1. ,  ...,  0.75,  0.5 ,  0.25,  0. ]])
```

`imsegm.region_growing.transform_rays_model_cdf_spectral(list_rays,`  
`nb_components=5)`  
 compute the mixture model and transform it into cumulative distribution

**Parameters**

- **list\_rays** (`list(list(int))`) – list ray features (distances)
- **nb\_components** (`int`) – number components in mixture model

**Return tuple(any,list(list(int)))** mixture model, list of stat/param of models

```
>>> np.random.seed(0)
>>> list_rays = [[9, 4, 9], [4, 9, 7], [9, 7, 11], [10, 8, 10],
...             [9, 11, 8], [4, 8, 5], [8, 10, 6], [9, 7, 11]]
>>> mm, cdist = transform_rays_model_cdf_spectral(list_rays)
>>> np.round(cdist, 1).tolist()
[[1.0, 1.0, 1.0, 1.0, 1.0, 0.9, 0.8, 0.6, 0.5, 0.2, 0.0],
 [1.0, 1.0, 1.0, 1.0, 1.0, 0.9, 0.9, 0.7, 0.5, 0.2, 0.0],
 [1.0, 1.0, 1.0, 1.0, 1.0, 0.9, 0.8, 0.7, 0.5, 0.3, 0.0]]
```

`imsegm.region_growing.transform_rays_model_sets_mean_cdf_kmeans(list_rays,`  
`nb_components=5)`  
 compute the mixture model and transform it into cumulative distribution

**Parameters**

- **list\_rays** (`list(list(int))`) – list ray features (distances)
- **nb\_components** (`int`) – number components in mixture model

**Return tuple(any,list(list(int)))** mixture model, list of stat/param of models

```
>>> np.random.seed(0)
>>> list_rays = [[9, 4, 9], [4, 9, 7], [9, 7, 11], [10, 8, 10],
...             [9, 11, 8], [4, 8, 5], [8, 10, 6], [9, 7, 11]]
>>> mm, mean_cdf = transform_rays_model_sets_mean_cdf_kmeans(list_rays, 2)
>>> len(mean_cdf)
2
```

`imsegm.region_growing.transform_rays_model_sets_mean_cdf_mixture(list_rays,`  
`nb_components=5,`  
`slic_size=15)`  
 compute the mixture model and transform it into cumulative distribution

**Parameters**

- **list\_rays** (`list(list(int))`) – list ray features (distances)
- **nb\_components** (`int`) – number components in mixture model
- **slic\_size** (`int`) – superpixel size

**Return tuple(any,list(list(int)))** mixture model, list of stat/param of models

```

>>> np.random.seed(0)
>>> list_rays = [[9, 4, 9], [4, 9, 7], [9, 7, 11], [10, 8, 10],
...             [9, 11, 8], [4, 8, 5], [8, 10, 6], [9, 7, 11]]
>>> mm, mean_cdf = transform_rays_model_sets_mean_cdf_mixture(list_rays, 2)
>>> len(mean_cdf)
2

```

`imsegm.region_growing.update_shape_costs_points` (*lut\_shape\_cost*, *slic*, *points*, *labels*, *init\_centres*, *centres*, *shifts*, *volumes*, *shape\_model*, *shape\_type*, *selected\_idx=None*, *swap\_shift=False*, *dict\_thresholds=None*)

update the shape prior for given segmentation (new centre is computed), set of points and shape model

#### Parameters

- **lut\_shape\_cost** – look-up-table for shape cost for GC
- **slic** (*nadarray*) – superpixel segmentation
- **int[] points** (*[int,)* – subsample space, points = superpixel centres
- **labels** (*list(int)*) – labels for points to be assigned to an object
- **int[] init\_centres** (*[int,)* – initial centre position for compute center shift during the iterations
- **int[] centres** (*[int,)* – actual centre position
- **shifts** (*list(int)*) – orientation for each region / object
- **volumes** (*[int]*) – size / volume for each region
- **shape\_model** – represent the shape prior and histograms
- **shape\_type** (*str*) – type or shape model
- **selected\_idx** (*[int]*) – selected object for update
- **swap\_shift** (*bool*) – allow swapping orientation by 90 degree, try to get out from local optima
- **dict\_thresholds** (*dict|None*) – configuration with thresholds
- **dict\_thresholds** – set some threshold updating shape prior

**Return tuple(list(float),list(int))**

`imsegm.region_growing.GC_REPLACE_INF = 100000.0`  
all infinity values in Grah-Cut terms replace by this value

`imsegm.region_growing.MAX_UNARY_PROB = 0.99`  
define maximal value of unary (being a class) term in Graph-Cut

`imsegm.region_growing.MIN_SHAPE_PROB = 0.01`  
define minimal value for any model of shape prior term

`imsegm.region_growing.RG2SP_THRESHOLDS = {'centre': 30, 'centre_init': 50, 'shift': 15, 'v'`  
define thresholds parameters for iterative Region Growing

## imsegm.superpixels module

### Framework for superpixels

- wrapper over skimage.SLIC
- other related functions

SEE: \* [http://scikit-image.org/docs/dev/auto\\_examples/plot\\_segmentations.html](http://scikit-image.org/docs/dev/auto_examples/plot_segmentations.html)

Copyright (C) 2014-2018 Jiri Borovec <jiri.borovec@fel.cvut.cz>

`imsegm.superpixels.get_neighboring_segments` (*edges*)

get the indexes of neighboring superpixels for each superpixel the input is list edges of all neighboring segments

**Parameters** `int[] edges` (`[[int,)` –

**Return** `[[int]]`

```
>>> get_neighboring_segments([[0, 1], [1, 2], [1, 3], [2, 3]])
[[1], [0, 2, 3], [1, 3], [1, 2]]
```

`imsegm.superpixels.get_segment_diffs_2d_conn4` (*grid*)

wrapper for getting 4-connected in 2D image plane

**Parameters** `grid` (*ndarray*) – segmentation

**Return** `((int, int))`

`imsegm.superpixels.get_segment_diffs_3d_conn6` (*grid*)

wrapper for getting 6-connected in 3D image plane

**Parameters** `grid` (*ndarray*) – segmentation

**Return** `((int, int, int))`

`imsegm.superpixels.make_graph_seg_connect_grid2d_conn4` (*grid*)

construct graph of connected components

**Parameters** `grid` (*ndarray*) – segmentation

**Return** `[int], [(int, int)]`

```
>>> grid = np.array([[0] * 5 + [1] * 5, [2] * 5 + [3] * 5])
>>> v, edges = make_graph_seg_connect_grid2d_conn4(grid)
>>> v
array([0, 1, 2, 3])
>>> edges
[[0, 1], [0, 2], [1, 3], [2, 3]]
```

`imsegm.superpixels.make_graph_seg_connect_grid3d_conn6` (*grid*)

construct graph of connected components

**Parameters** `grid` (*ndarray*) – segmentation

**Return** `[int], [(int, int)]`

```
>>> grid_2d = np.array([[0] * 5 + [1] * 5, [2] * 5 + [3] * 5])
>>> grid = np.array([grid_2d, grid_2d + 4])
>>> v, edges = make_graph_seg_connect_grid3d_conn6(grid)
>>> v
array([0, 1, 2, 3, 4, 5, 6, 7])
>>> edges
```

(continues on next page)

(continued from previous page)

```
[[0, 1], [0, 2], [1, 3], [2, 3], [0, 4], [1, 5], [4, 5], [2, 6], [4, 6],
 [3, 7], [5, 7], [6, 7]]
```

`imsegm.superpixels.make_graph_segment_connect_edges(vertices, all_edges)`

make graph of connencted components SEE: <http://peekaboo-vision.blogspot.cz/2011/08/region-connectivity-graphs-in-python.html>

#### Parameters

- **vertices** (*ndarray*) –
- **all\_edges** (*ndarray*) –

**Return tuple(ndarray, ndarray)**

`imsegm.superpixels.segment_slic_img2d(img, sp_size=50, relative_compact=0.1, slico=False)`

segmentation by SLIC superpixels using original SLIC implementation

#### Parameters

- **img** (*ndarray*) – input color image
- **sp\_size** (*int*) – superpixel initial size
- **relative\_compact** (*float*) – relative regularisation in range (0, 1) where 0 is for free form and 1 for nearly rectangular superpixels
- **slic** (*bool*) – whether use parameter free version ASLIC/SLICO

**Return ndarray** segmentation

```
>>> np.random.seed(0)
>>> img = np.random.random((100, 150, 3))
>>> slic = segment_slic_img2d(img, 20, 0.2)
>>> slic.shape
(100, 150)
>>> img = np.random.random((150, 100))
>>> slic = segment_slic_img2d(img, 20, 0.2)
>>> slic.shape
(150, 100)
```

`imsegm.superpixels.segment_slic_img3d_gray(im, sp_size=50, relative_compact=0.1, space=(1, 1, 1))`

segmentation by SLIC superpixels using originla SLIC implementation

#### Parameters

- **im** (*ndarray*) – input 3D grayscale image
- **sp\_size** (*int*) – superpixel initial size
- **relative\_compact** (*float*) – relative regularisation in range (0, 1) where 0 is for free form and 1 for nearly rectangular superpixels
- **space** (*tuple(int, int, int)*) – spacing in 3d image may not be equal

**Return ndarray**

```
>>> np.random.seed(0)
>>> img = np.random.random((100, 100, 10))
>>> slic = segment_slic_img3d_gray(img, 20, 0.2, (1, 1, 5))
>>> slic.shape
(100, 100, 10)
```

`imsegm.superpixels.superpixel_centers` (*segments*)  
estimate centers of each superpixel

**Parameters** `segments` (*ndarray*) – segmentation `np.array<height, width>`

**Return** [(float, float)]

```
>>> segm = np.array([[0] * 6 + [1] * 5, [0] * 6 + [2] * 5])
>>> superpixel_centers(segm)
[(0.5, 2.5), (0.0, 8.0), (1.0, 8.0)]
>>> superpixel_centers(np.array([segm, segm, segm]))
[[1.0, 0.5, 2.5], [1.0, 0.0, 8.0], [1.0, 1.0, 8.0]]
```

`imsegm.superpixels.IMAGE_SPACING` = (1, 1, 1)  
spacing among neighboring pixels in axes X, Y, Z

## 1.2.3 Module contents

General superpixel image segmentation: (un)supervised, center detection, region growing

## 1.3 Examples

### 1.3.1 Sample unsupervised segmentation on Color images

Image segmentation is widely used as an initial phase of many image processing tasks in computer vision and image analysis. Many recent segmentation methods use superpixels, because they reduce the size of the segmentation problem by an order of magnitude. In addition, features on superpixels are much more robust than features on pixels only. We use spatial regularization on superpixels to make segmented regions more compact. The segmentation pipeline comprises: (i) computation of superpixels; (ii) extraction of descriptors such as color and texture; (iii) soft classification, using the Gaussian Mixture Model for unsupervised learning; (iv) final segmentation using Graph Cut. We use this segmentation pipeline on four real-world applications in medical imaging. We also show that unsupervised segmentation is sufficient for some situations, and provides similar results to those obtained using trained segmentation.

Borovec, J., Svihlik, J., Kybic, J., & Habart, D. (2017). **Supervised and unsupervised segmentation using superpixels, model estimation, and Graph Cut.** Journal of Electronic Imaging.

```
[1]: %matplotlib inline
import os, sys, glob, time
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from skimage.segmentation import mark_boundaries
sys.path += [os.path.abspath('.'), os.path.abspath '..')] # Add path to root
import imsegm.utilities.data_io as tl_data
import imsegm.pipelines as segm_pipe
```



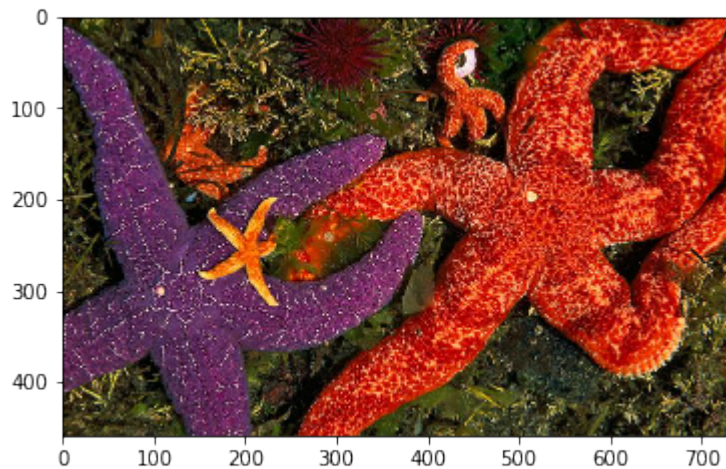
## Load image

```
[8]: path_dir = os.path.join(tl_data.update_path('data-images'), 'others')
print ([os.path.basename(p) for p in glob.glob(os.path.join(path_dir, '*.jpg'))])
path_img = os.path.join(path_dir, 'sea_starfish-2.jpg')

img = np.array(Image.open(path_img))

FIG_SIZE = (8. * np.array(img.shape[:2]) / np.max(img.shape))[:, -1]
_ = plt.imshow(img)

['stars_nb2.jpg', 'star_nb1.jpg']
```



## Segment Image

Set segmentation parameters:

```
[4]: nb_classes = 3
sp_size = 25
sp_regul = 0.2
dict_features = {'color': ['mean', 'std', 'median']}
```

Estimate the model without any annotation

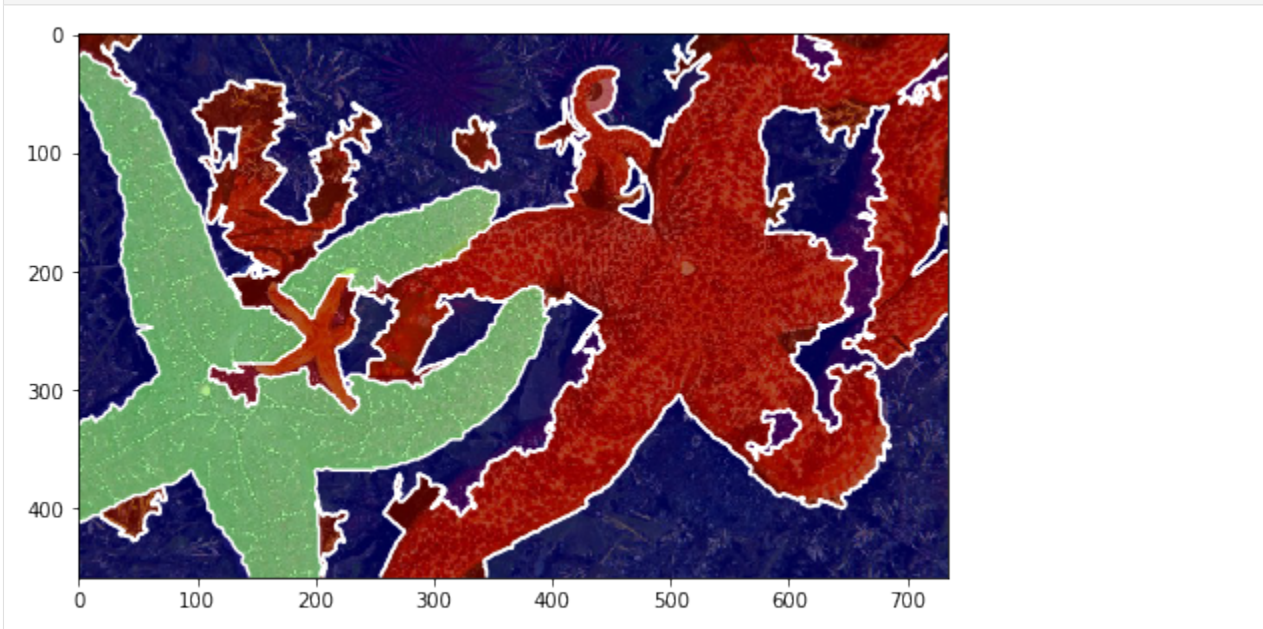
```
[5]: model, _ = segm_pipe.estim_model_classes_group([img], nb_classes, sp_size=sp_size, sp_
↪ regul=sp_regul,
dict_features=dict_features, pca_
↪ coef=None, model_type='GMM')
```

Perform segmentation with estimated model

```
[6]: dict_debug = {}
seg, _ = segm_pipe.segment_color2d_slic_features_model_graphcut(img, model, sp_
↪ size=sp_size, sp_regul=sp_regul,
dict_features=dict_features, gc_regul=5., gc_edge_type='color',
↪ debug_visual=dict_debug)
```

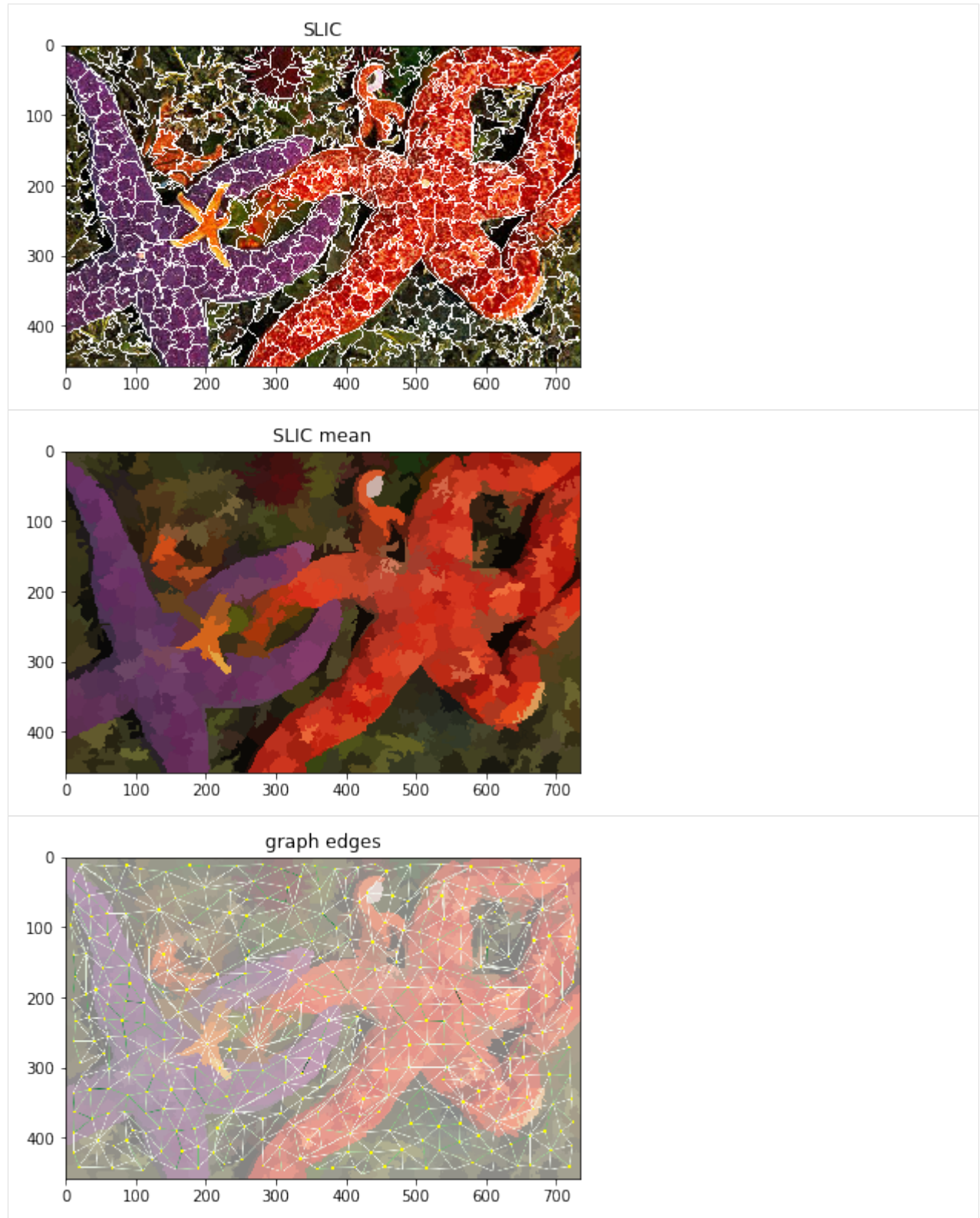
```
/usr/local/lib/python3.5/dist-packages/sklearn/mixture/base.py:237:
↳ConvergenceWarning: Initialization 1 did not converge. Try different init_
↳parameters, or increase max_iter, tol or check for degenerate data.
% (init + 1), ConvergenceWarning)
```

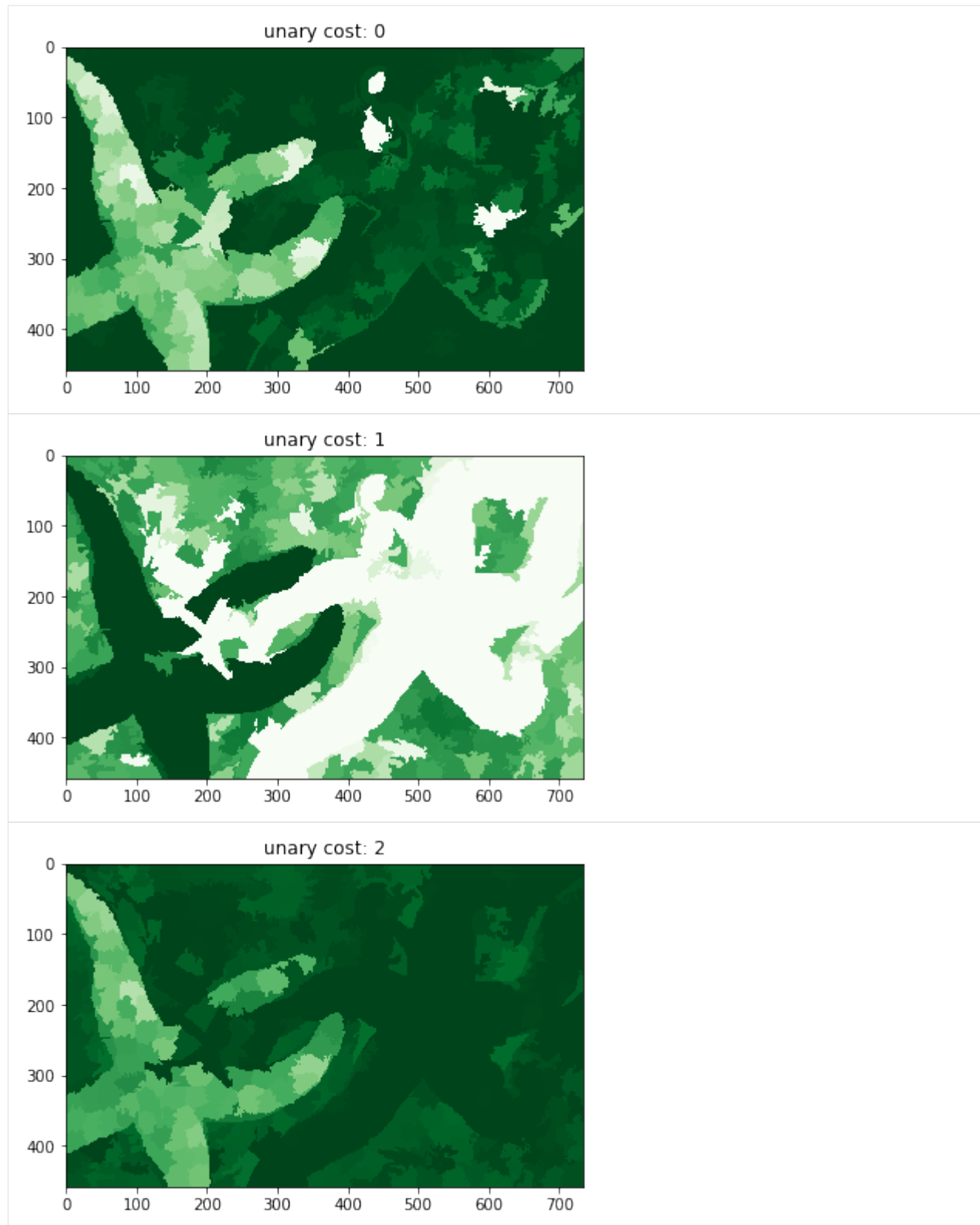
```
[9]: fig = plt.figure(figsize=FIG_SIZE)
plt.imshow(img)
plt.imshow(seg, alpha=0.6, cmap=plt.cm.jet)
_ = plt.contour(seg, levels=np.unique(seg), colors='w')
```



### Visualise intermediate steps

```
[12]: plt.figure(), plt.imshow(mark_boundaries(img, dict_debug['slic'], color=(1, 1, 1))),
↳plt.title('SLIC')
plt.figure(), plt.imshow(dict_debug['slic_mean']), plt.title('SLIC mean')
plt.figure(), plt.imshow(dict_debug['img_graph_edges']), plt.title('graph edges')
for i, im_u in enumerate(dict_debug['imgs_unary_cost']):
    plt.figure(), plt.title('unary cost: %i' % i), plt.imshow(im_u)
# plt.figure(), plt.imshow(dict_debug['img_graph_seg'])
```





[ ]:

### 1.3.2 Sample supervised segmentation on Gray images

Image segmentation is widely used as an initial phase of many image processing tasks in computer vision and image analysis. Many recent segmentation methods use superpixels, because they reduce the size of the segmentation problem by an order of magnitude. In addition, features on superpixels are much more robust than features on pixels only. We use spatial regularization on superpixels to make segmented regions more compact. The segmentation pipeline comprises: (i) computation of superpixels; (ii) extraction of descriptors such as color and texture; (iii) soft classification, using a standard classifier for supervised learning; (iv) final segmentation using Graph Cut. We use this segmentation pipeline on four real-world applications in medical imaging. We also show that unsupervised segmentation is sufficient for some situations, and provides similar results to those obtained using trained segmentation.

Borovec, J., Svihlik, J., Kybic, J., & Habart, D. (2017). **Supervised and unsupervised segmentation using superpixels, model estimation, and Graph Cut.** Journal of Electronic Imaging.

```
[1]: %matplotlib inline
import os, sys, glob, time
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from skimage.segmentation import mark_boundaries
sys.path += [os.path.abspath('.'), os.path.abspath('../')] # Add path to root
import imsegm.utilities.data_io as tl_data
import imsegm.pipelines as segm_pipe
```

#### Load image

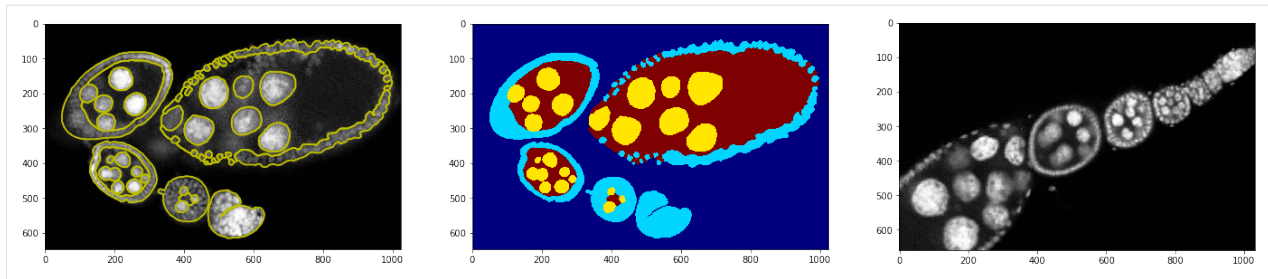
```
[2]: path_dir = os.path.join(tl_data.update_path('data-images'), 'drosophila_ovary_slice')
path_images = os.path.join(path_dir, 'image')
print ([os.path.basename(p) for p in glob.glob(os.path.join(path_images, '*.jpg'))])
# loading images
path_img = os.path.join(path_images, 'insitu7545.jpg')
img = np.array(Image.open(path_img))[:, :, 0]
path_img = os.path.join(path_images, 'insitu4174.jpg')
img2 = np.array(Image.open(path_img))[:, :, 0]
# loading annotations
path_annot = os.path.join(path_dir, 'annot_struct')
path_annot = os.path.join(path_annot, 'insitu7545.png')
annot = np.array(Image.open(path_annot))

['insitu7331.jpg', 'insitu4174.jpg', 'insitu4358.jpg', 'insitu7545.jpg', 'insitu7544.
↪jpg']
```

#### Show that training example with annotation and testing image

```
[14]: FIG_SIZE = (8. * np.array(img.shape[:2]) / np.max(img.shape))[:-1]
fig = plt.figure(figsize=FIG_SIZE * 3)
_ = plt.subplot(1,3,1), plt.imshow(img, cmap=plt.cm.Greys_r), plt.contour(annot, _
↪colors='y')
_ = plt.subplot(1,3,2), plt.imshow(annot, cmap=plt.cm.jet)
_ = plt.subplot(1,3,3), plt.imshow(img2, cmap=plt.cm.Greys_r)
```





## Segment Image

Set segmentation parameters:

```
[4]: sp_size = 25
     sp_regul = 0.2
     dict_features = {'color': ['mean', 'std', 'median'], 'tLM': ['mean']}
```

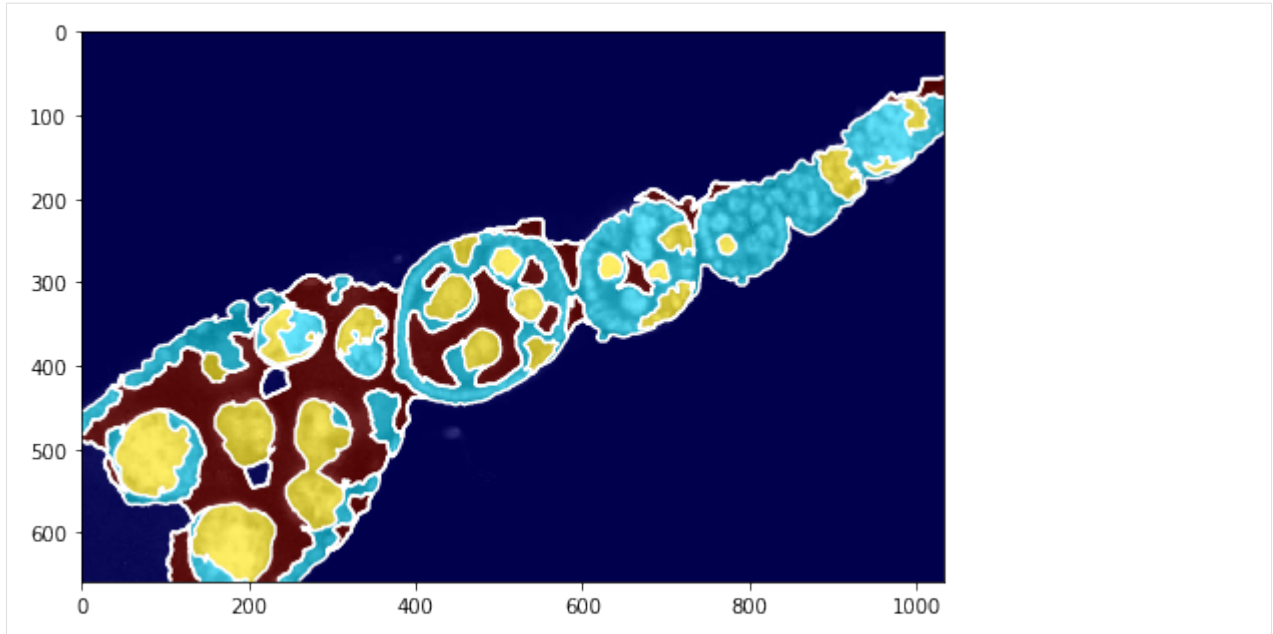
Train the classifier

```
[5]: classif, list_slic, list_features, list_labels = segm_pipe.train_classif_color2d_slic_
     ↪ features([img], [annot],
     ↪          sp_size=sp_size, sp_regul=sp_regul, dict_features=dict_features, pca_
     ↪ coef=None)
```

Perform the segmentation with trained classifier

```
[6]: dict_debug = {}
     seg, _ = segm_pipe.segment_color2d_slic_features_model_graphcut(img2, classif, sp_
     ↪ size=sp_size, sp_regul=sp_regul,
     ↪          gc_regul=1., dict_features=dict_features, gc_edge_type='model', _
     ↪ debug_visual=dict_debug)
```

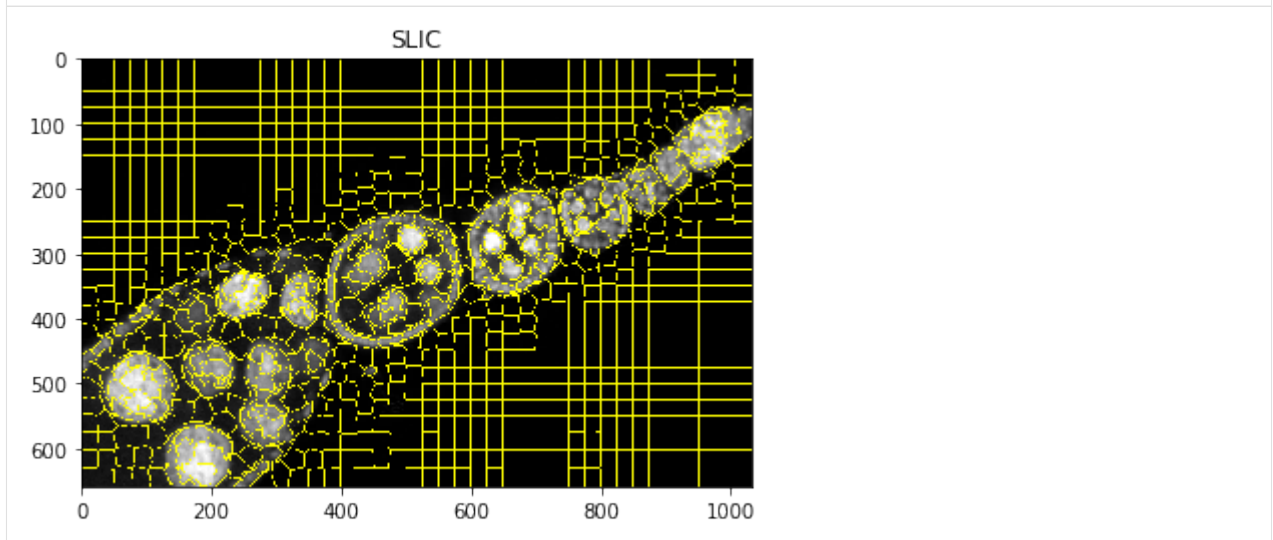
```
[15]: fig = plt.figure(figsize=FIG_SIZE)
     plt.imshow(img2, cmap=plt.cm.Greys_r)
     plt.imshow(seg, alpha=0.6, cmap=plt.cm.jet)
     _ = plt.contour(seg, levels=np.unique(seg), colors='w')
```

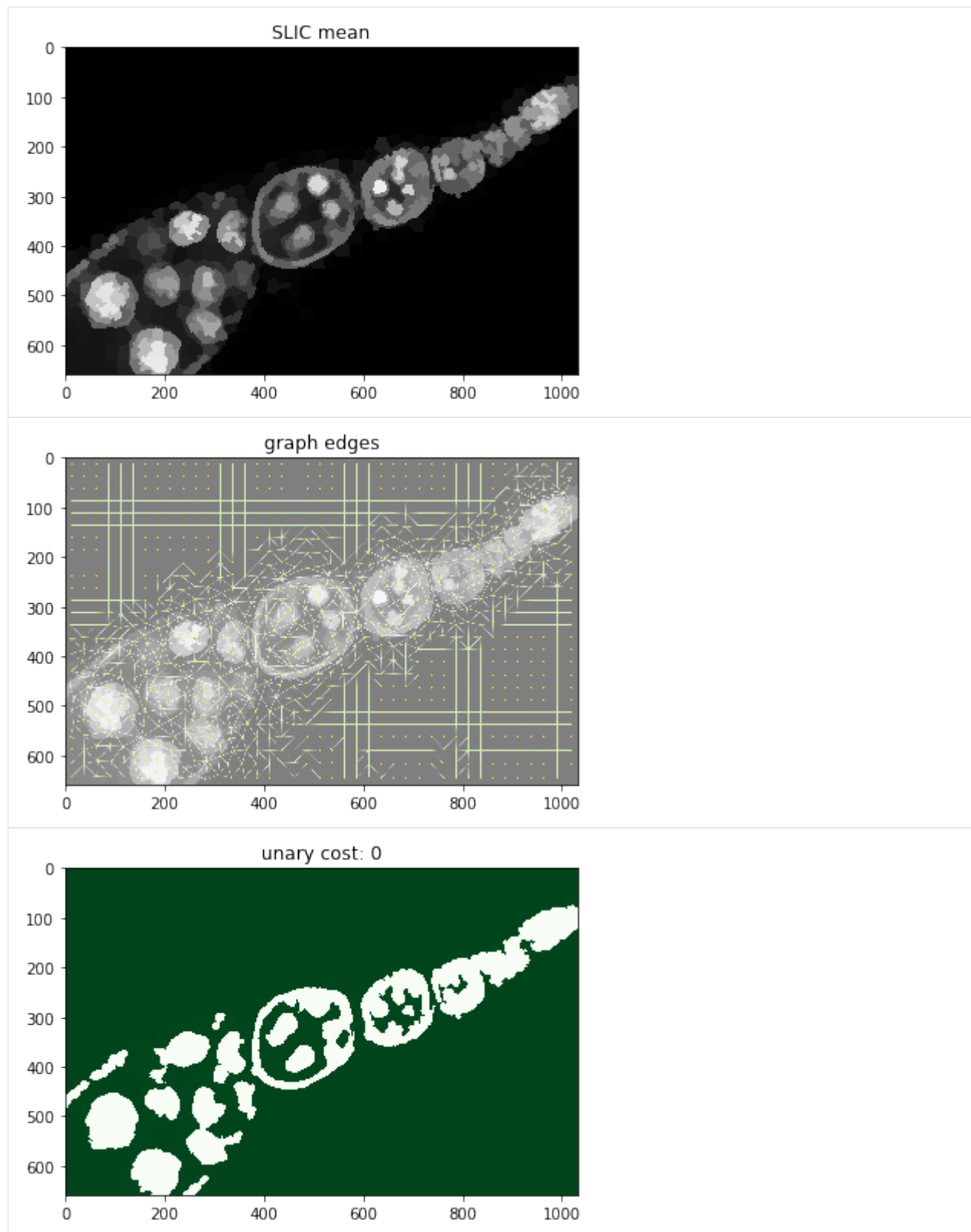


### Visualise intermediate steps

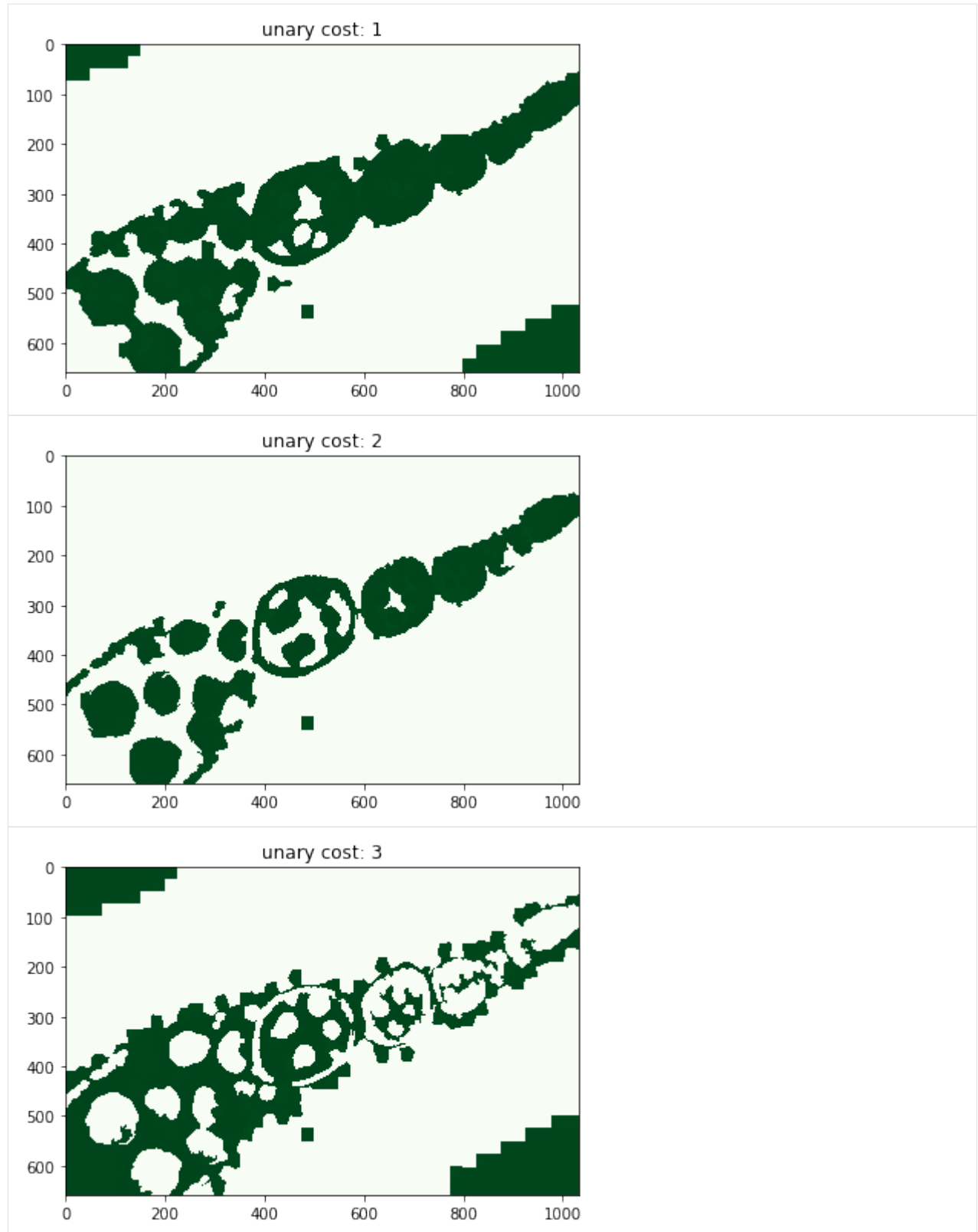
```
[13]: print ('debug fields: %s' % repr(dict_debug.keys()))
plt.figure(), plt.imshow(mark_boundaries(img2, dict_debug['slic']), plt.title('SLIC'))
plt.figure(), plt.imshow(dict_debug['slic_mean'], plt.title('SLIC mean'))
plt.figure(), plt.imshow(dict_debug['img_graph_edges'], plt.title('graph edges'))
for i, im_u in enumerate(dict_debug['imgs_unary_cost']):
    plt.figure(), plt.title('unary cost: %i' % i), plt.imshow(im_u)
# plt.figure(), plt.imshow(dict_debug['img_graph_seg'])
```

debug fields: dict\_keys(['slic', 'img\_graph\_seg', 'slic\_mean', 'segments', 'edge\_weights', 'imgs\_unary\_cost', 'img\_graph\_edges', 'image', 'edges'])









```
[ ]:
```

### 1.3.3 Object shape model - estimation

Measure ray features and estimate the model over whole dataset. We play a bit with different mixture strategies such as GMM, Mean shift, Kmeans, etc.

```
[1]: %matplotlib inline
import os, sys, glob
import numpy as np
import pandas as pd
from PIL import Image
import matplotlib.pyplot as plt
from scipy import ndimage

[2]: sys.path += [os.path.abspath('.'), os.path.abspath('../')] # Add path to root
import imsegm.utilities.data_io as tl_io
import imsegm.region_growing as tl_rg
import imsegm.descriptors as tl_fts
```

#### Loading ovary

```
[4]: COLORS = 'bgrmyck'
PATH_IMAGES = tl_io.update_path(os.path.join('data-images', 'drosophila_ovary_slice'))
PATH_DATA = tl_io.update_path('data-images', absolute=True)
PATH_OUT = tl_io.update_path('output', absolute=True)
PATH_MEASURED_RAYS = os.path.join(PATH_IMAGES, 'eggs_ray-shapes.csv')
print ([os.path.basename(p) for p in glob.glob(os.path.join(PATH_IMAGES, '*')) if os.
↳ path.isdir(p)])
dir_annot = os.path.join(PATH_IMAGES, 'annot_eggs')
# dir_annot = os.path.expanduser('/home/jirka/Dropbox/temp/mask_2d_slice_complete_ind_
↳ egg')

['center_levels', 'image', 'annot_struct', 'ellipse_fitting', 'annot_eggs', 'segm_rgb
↳ ', 'segm', 'image_cut-stage-2']
```

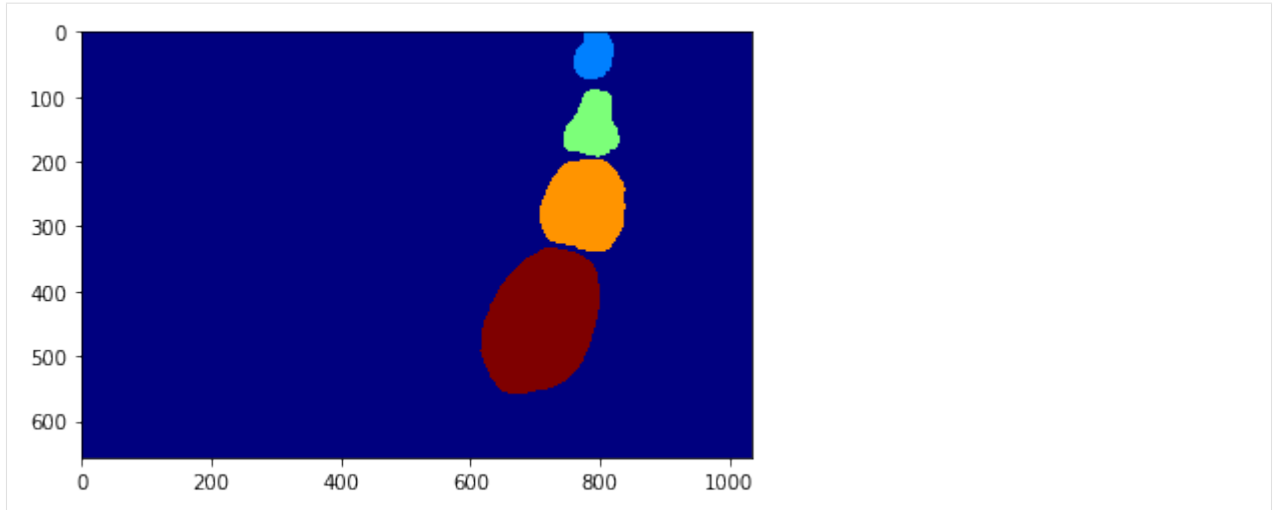
#### Loading...

```
[5]: list_paths = sorted(glob.glob(os.path.join(dir_annot, '*.png')))
print ('nb images: %i SAMPLES: \n %s' % (len(list_paths), repr([os.path.basename(p)
↳ for p in list_paths[:5]])))
list_segms = []
for path_seg in list_paths:
    seg = np.array(Image.open(path_seg))
    list_segms.append(seg)

nb images: 67 SAMPLES:
['insitu14807.png', 'insitu14808.png', 'insitu14809.png', 'insitu14810.png',
↳ 'insitu14811.png']
```

Randomly selected sample image from fiven dataset.

```
[6]: seg = list_segms[np.random.randint(0, len(list_segms))]
_ = plt.imshow(seg, cmap=plt.cm.jet)
```



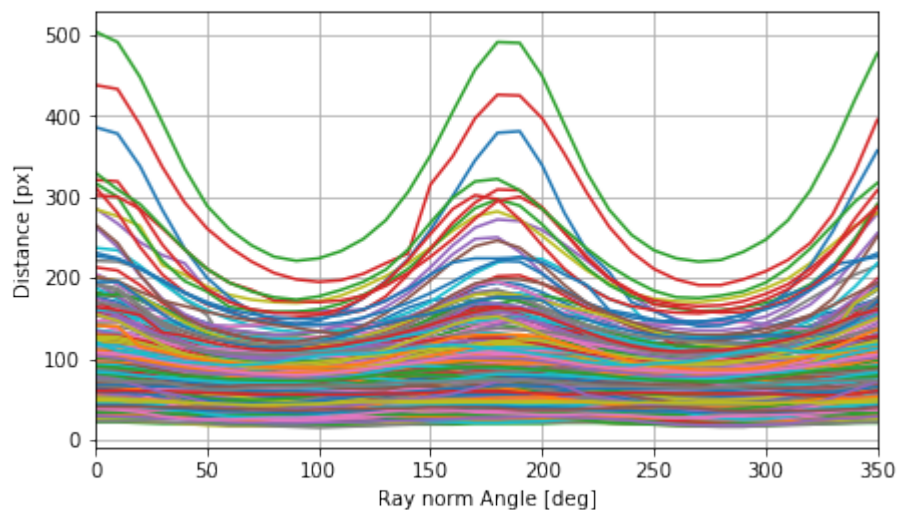
### Compute Ray features

Simple statistic over measured normalised Ray features over whole dataset.

```
[7]: list_rays, list_shifts = tl_rg.compute_object_shapes(list_segms, ray_step=10, interp_
      ↪order='spline', smooth_coef=1, shift_method='max')
print ('nb eggs: %i ; nb rays: %i' % (len(list_rays), len(list_rays[0])))
```

```
nb eggs: 241 ; nb rays: 36
```

```
[9]: fig = plt.figure(figsize=(7, 4))
x_axis = np.linspace(0, 360, len(list_rays[0]), endpoint=False)
plt.plot(x_axis, np.array(list_rays).T, '-')
plt.grid(), plt.xlim([0, 350])
_ = plt.xlabel('Ray norm Angle [deg]', plt.ylabel('Distance [px]')
#fig.savefig(os.path.join(PATH_OUT, 'shape-rays_all.pdf'), bbox_inches='tight')
```

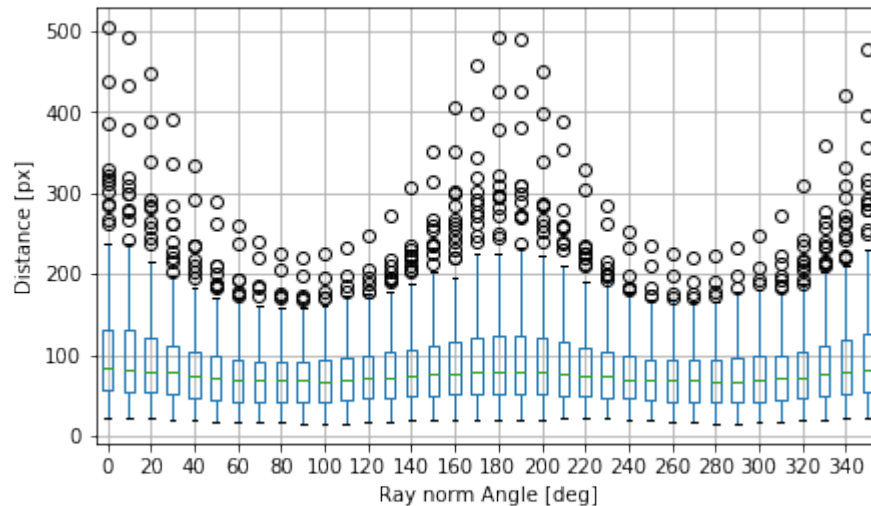


```
[10]: df = pd.DataFrame(np.array(list_rays), columns=x_axis.astype(int))
df.to_csv(PATH_MEASURED_RAYS)
```

(continues on next page)

(continued from previous page)

```
df.plot.box(figsize=(7, 4), grid=True)
plt.xticks(range(1, 37, 2), [str(i * 10) for i in range(0, 36, 2)])
_ = plt.xlabel('Ray norm Angle [deg]', plt.ylabel('Distance [px]')
# plt.savefig(os.path.join(PATH_OUT, 'shape-rays_statistic.pdf'), bbox_inches='tight')
```



## Clustering

You can compute three Ray features from segmentation or load precomputed vectors in CSV file.

```
[11]: df = pd.read_csv(PATH_MEASURED_RAYS, index_col=0)
list_rays = df.values
x_axis = np.linspace(0, 360, list_rays.shape[1], endpoint=False)
```

## Spectral Clustering

```
[12]: from sklearn import cluster

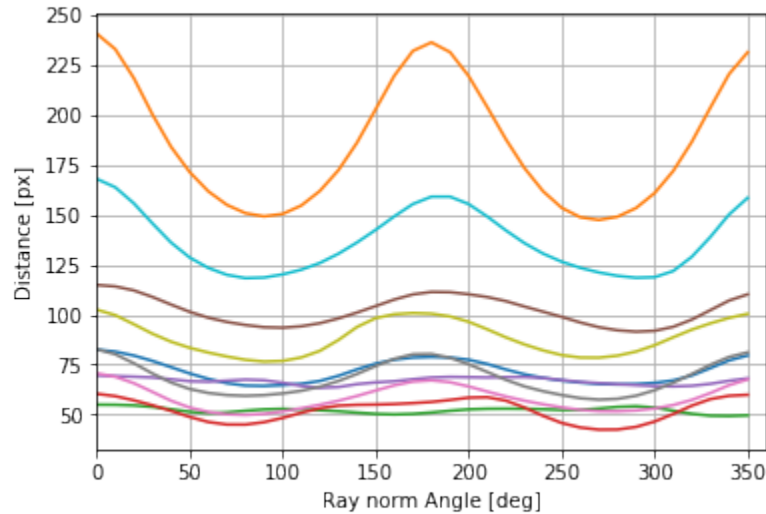
model = cluster.SpectralClustering(10)
model.fit(np.array(list_rays))
print ('label histogram: %s' % repr(np.bincount(model.labels_)))

list_ray_core = []
for lb in np.unique(model.labels_):
    mean_rays = np.mean(np.asarray(list_rays)[model.labels_ == lb], axis=0)
    mean_rays = ndimage.filters.gaussian_filter1d(mean_rays, 1)
    list_ray_core.append(mean_rays)
```

```
label histogram: array([ 6,  2,  2,  2,  2,  2,  2,  2,  2, 219])
```

```
/usr/local/lib/python2.7/dist-packages/sklearn/manifold/spectral_embedding_.py:234:
↳ UserWarning: Graph is not fully connected, spectral embedding may not work as
↳ expected.
warnings.warn("Graph is not fully connected, spectral embedding")
```

```
[18]: #fig = plt.figure(figsize=(10, 5))
plt.plot(x_axis, np.array(list_ray_core).T, '-')
plt.grid(), plt.xlim([0, 360])
_ = plt.xlabel('Ray norm Angle [deg]'), plt.ylabel('Distance [px]')
```



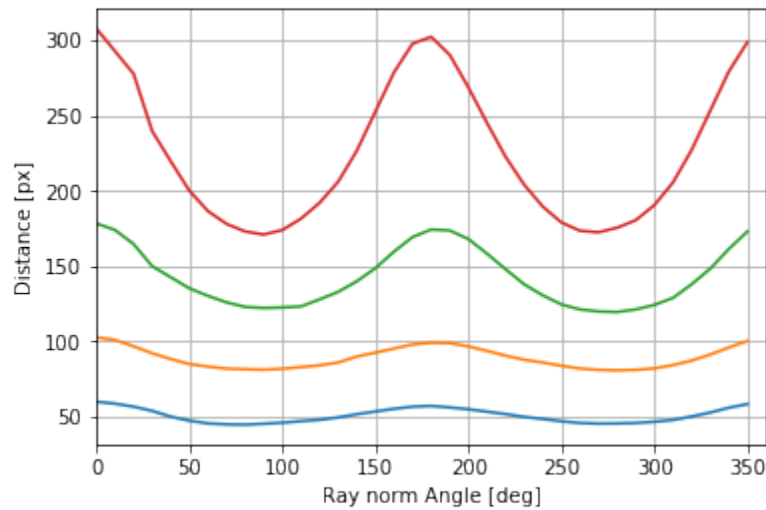
## Mean Shift

```
[19]: from sklearn import cluster

mean_shift = cluster.MeanShift()
mean_shift.fit(np.array(list_rays))
print ('label histogram: %s' % repr(np.bincount(mean_shift.labels_)))

[8 6 3 2]
```

```
[20]: #fig = plt.figure(figsize=(10, 5))
plt.plot(x_axis, mean_shift.cluster_centers_.T, '-')
plt.grid(), plt.xlim([0, 360])
_ = plt.xlabel('Ray norm Angle [deg]'), plt.ylabel('Distance [px]')
```



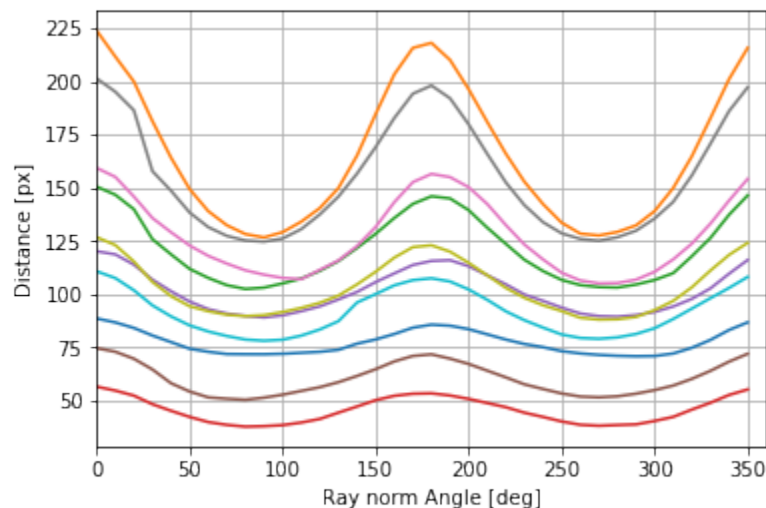
## Gaussian Mixture Model

```
[21]: from sklearn import mixture
```

```
gmm = mixture.BayesianGaussianMixture(n_components=10)
gmm.fit(np.array(list_rays))
# gmm.fit(np.array(list_rays), mean_shift_labels)
print ('weights:', gmm.weights_, 'means:', gmm.means_.shape, 'covariances:', gmm.
      ↪ covariances_.shape)

weights: [0.19914697 0.09375047 0.1318912  0.16338295 0.11149782 0.16540886
 0.0531903  0.04021706 0.02724381 0.01427057] means: (10, 36) covariances: (10, 36, ↪
↪ 36)
```

```
[22]: #fig = plt.figure(figsize=(10, 5))
plt.plot(x_axis, gmm.means_.T, '-', label=' ')
plt.grid(), plt.xlim([0, 360])
_ = plt.xlabel('Ray norm Angle [deg]', plt.ylabel('Distance [px]')
```



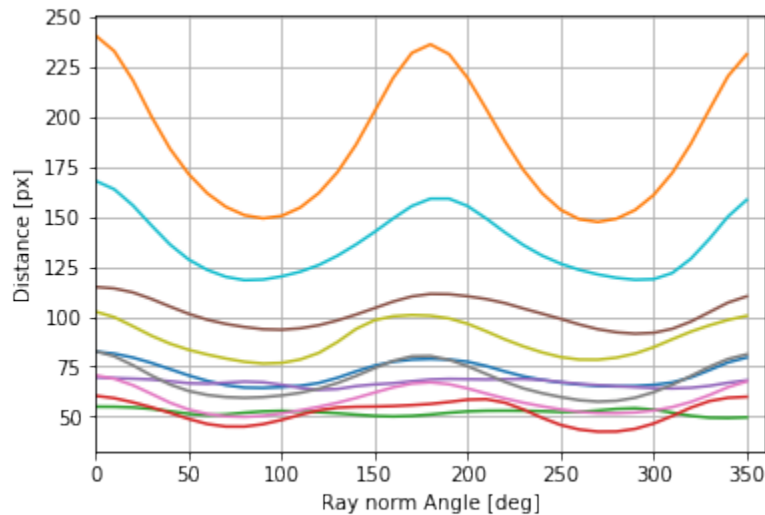
## Agglomerative Clustering

```
[23]: agg = cluster.AgglomerativeClustering(7)
agg.fit(np.array(list_rays))
# gmm.fit(np.array(list_rays), mean_shift_labels)
print ('label histogram: %s' % repr(np.bincount(agg.labels_)))

list_ray_core = []
for lb in np.unique(model.labels_):
    mean_rays = np.mean(np.asarray(list_rays)[model.labels_ == lb], axis=0)
    mean_rays = ndimage.filters.gaussian_filter1d(mean_rays, 1)
    list_ray_core.append(mean_rays)

[2 4 4 2 3 3 1]
```

```
[24]: #plt.figure(figsize=(10, 5))
plt.plot(x_axis, np.array(list_ray_core).T, '-')
plt.grid(), plt.xlim([0, 360])
_ = plt.xlabel('Ray norm Angle [deg]'), plt.ylabel('Distance [px]')
```



## Cumulativ Priors

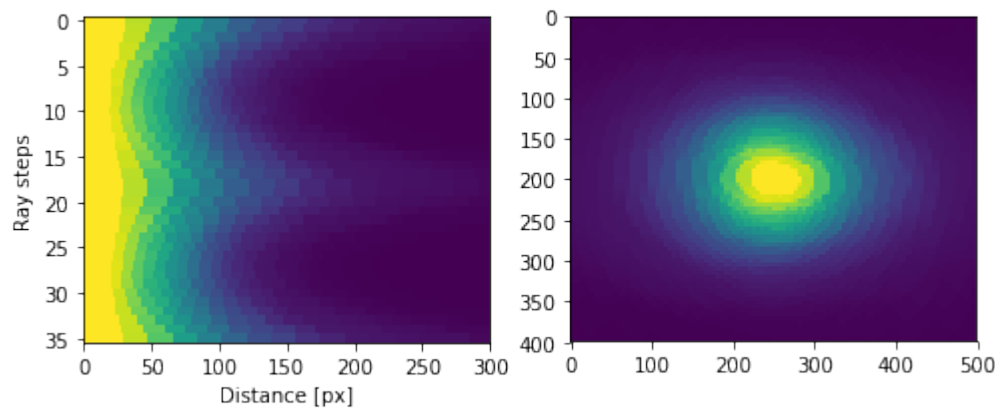
```
[13]: df = pd.read_csv(PATH_MEASURED_RAYS, index_col=0)
list_rays = df.values

[14]: def compute_prior_map(cdist, size=(500, 800), step=5):
    prior_map = np.zeros(size)
    centre = np.array(size) / 2
    for i in np.arange(prior_map.shape[0], step=step):
        for j in np.arange(prior_map.shape[1], step=step):
            prior_map[i:i+step, j:j+step] = \
                tl_rg.compute_shape_prior_table_cdf([i, j], cdist, centre, angle_
↪ shift=0)
    return prior_map
```

## Histogram

```
[15]: list_cdf = tl_rg.transform_rays_model_cdf_histograms(list_rays, nb_bins=25)
      cdist = np.array(list_cdf)

      fig = plt.figure(figsize=(8, 3))
      _ = plt.subplot(1, 2, 1), plt.imshow(cdist[:, :300], aspect='auto'), plt.ylabel('Ray ↵
      ↵ steps'), plt.xlabel('Distance [px]')
      _ = plt.subplot(1, 2, 2), plt.imshow(compute_prior_map(cdist, size=(400, 500), step=5))
```

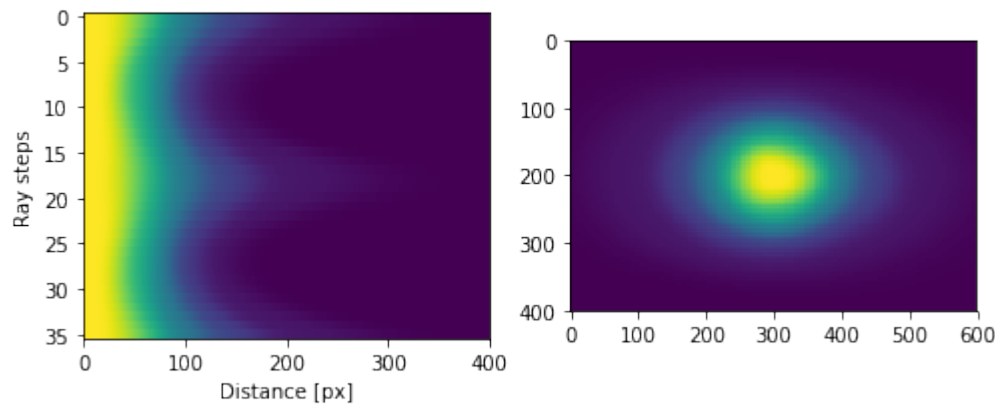


## Mixture model

```
[16]: mm, list_cdf = tl_rg.transform_rays_model_cdf_mixture(list_rays, coef_components=1)
      cdist = np.array(list_cdf)
      print (mm.weights_)

      fig = plt.figure(figsize=(8, 3))
      _ = plt.subplot(1, 2, 1), plt.imshow(cdist[:, :], aspect='auto'), plt.ylabel('Ray ↵
      ↵ steps'), plt.xlabel('Distance [px]')
      _ = plt.subplot(1, 2, 2), plt.imshow(compute_prior_map(cdist, size=(400, 600), step=5))
      # plt.savefig('shape-rays_gmm-cdf-proj.pdf')

[ 0.21486968  0.05343626  0.17167657  0.13797019  0.01205806  0.1950365
  0.21495274]
```



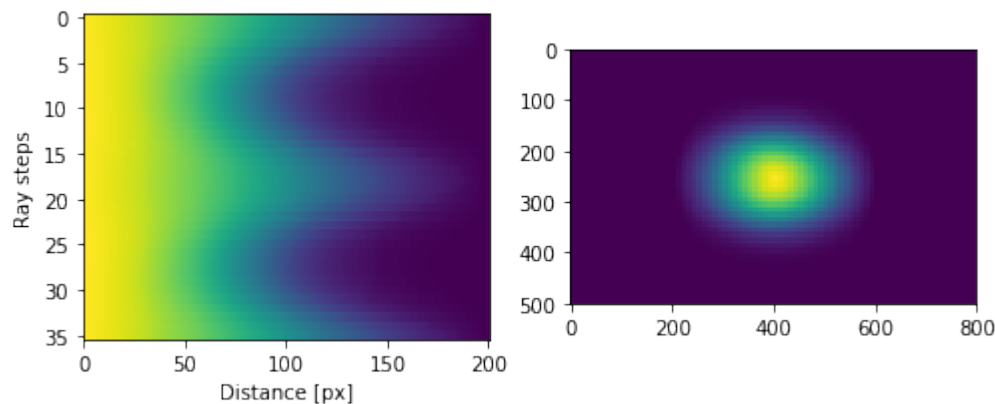


## Spectral Clustering

```
[17]: sc, list_cdf = tl_rg.transform_rays_model_cdf_spectral(list_rays)
      cdist = np.array(list_cdf)
      print ('label histogram: %s' % repr(np.bincount(sc.labels_)))

      fig = plt.figure(figsize=(8, 3))
      _ = plt.subplot(1, 2, 1), plt.imshow(cdist, aspect='auto'), plt.ylabel('Ray steps'),
      plt.xlabel('Distance [px]')
      _ = plt.subplot(1, 2, 2), plt.imshow(compute_prior_map(cdist, step=10))

      label histogram: array([233,  2,  2,  2,  2])
```

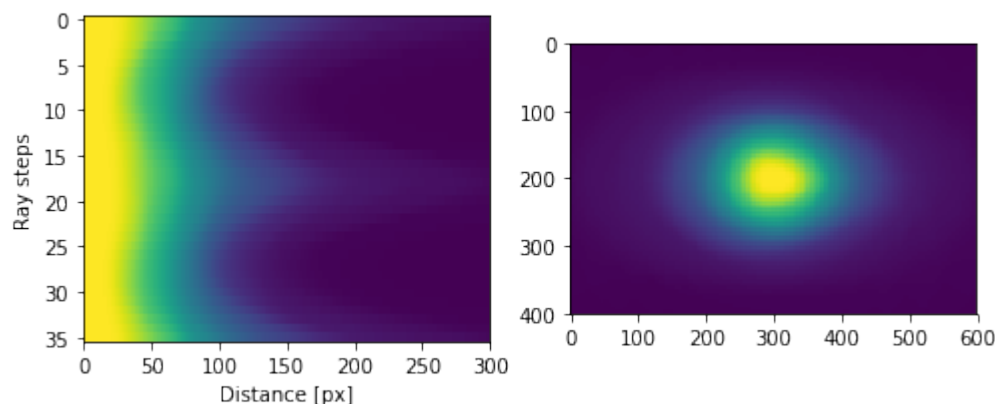


## K-means

```
[18]: km, list_cdf = tl_rg.transform_rays_model_cdf_kmeans(list_rays, 15)
      cdist = np.array(list_cdf)
      print ('label histogram: %s' % repr(np.bincount(km.labels_)))

      fig = plt.figure(figsize=(8, 3))
      _ = plt.subplot(1, 2, 1), plt.imshow(cdist[:, :300], aspect='auto'), plt.ylabel('Ray_
      steps'), plt.xlabel('Distance [px]')
      _ = plt.subplot(1, 2, 2), plt.imshow(compute_prior_map(cdist, size=(400, 600), step=5))

      label histogram: array([28,  5, 30, 12,  1, 32, 33,  7, 34,  7,  1, 19,  1,  1, 30])
```



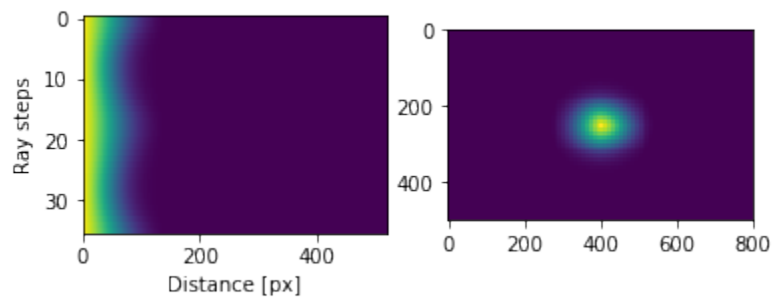
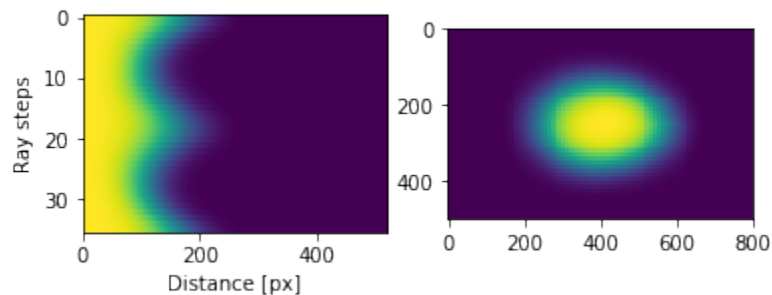
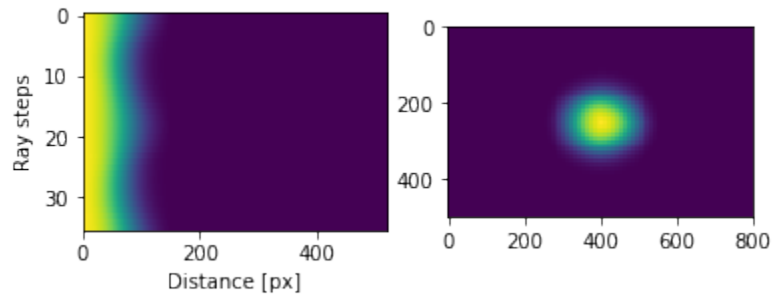
## Mixture of Cumulativ Models

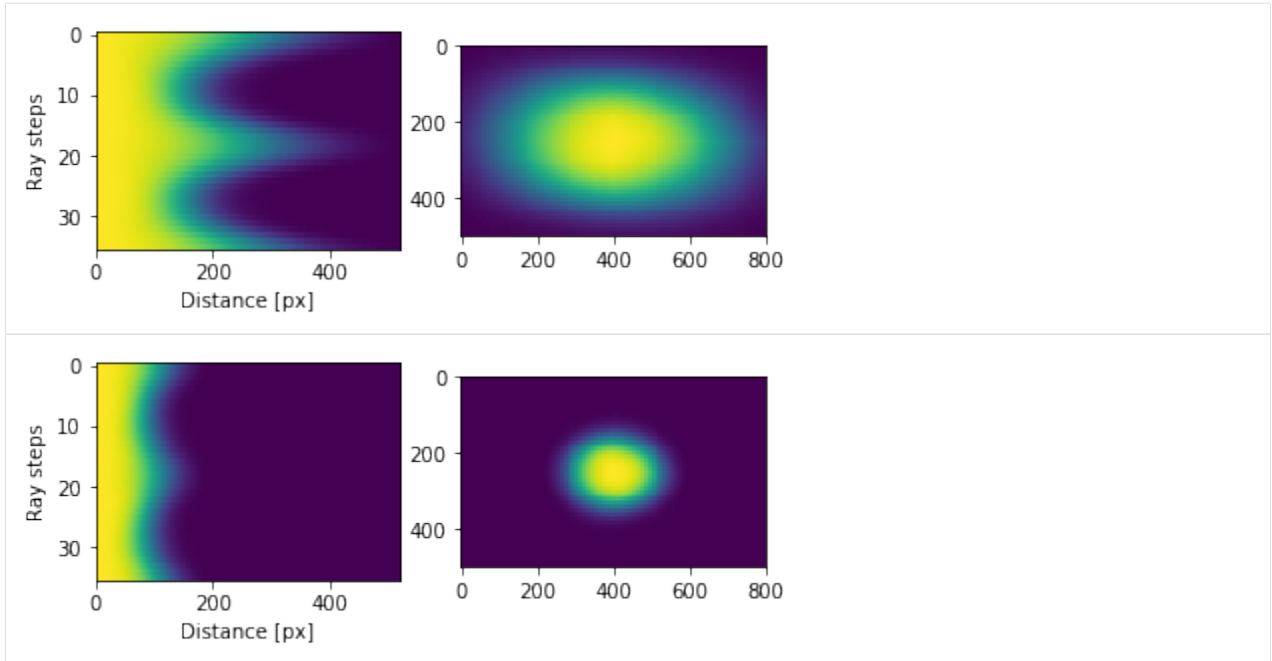
```
[19]: df = pd.read_csv(PATH_MEASURED_RAYS, index_col=0)
list_rays = df.values
```

## Gaussian mixture

```
[20]: model, list_mean_cdf = tl_rg.transform_rays_model_sets_mean_cdf_mixture(list_rays, 5)
max_len = max([np.asarray(l_cdf).shape[1] for _, l_cdf in list_mean_cdf])

for i, (mean, list_cdf) in enumerate(list_mean_cdf):
    cdist = np.zeros((len(list_cdf), max_len))
    cdist[:, :len(list_cdf[0])] = np.array(list_cdf)
    plt.figure(figsize=(6, 2))
    plt.subplot(1, 2, 1), plt.imshow(cdist, aspect='auto'), plt.xlim([0, max_len]),
    plt.ylabel('Ray steps'), plt.xlabel('Distance [px]')
    plt.subplot(1, 2, 2), plt.imshow(compute_prior_map(cdist, step=10))
    # plt.savefig('shape-rays_gmm-cdf-proj-%i.pdf' % (i + 1))
```

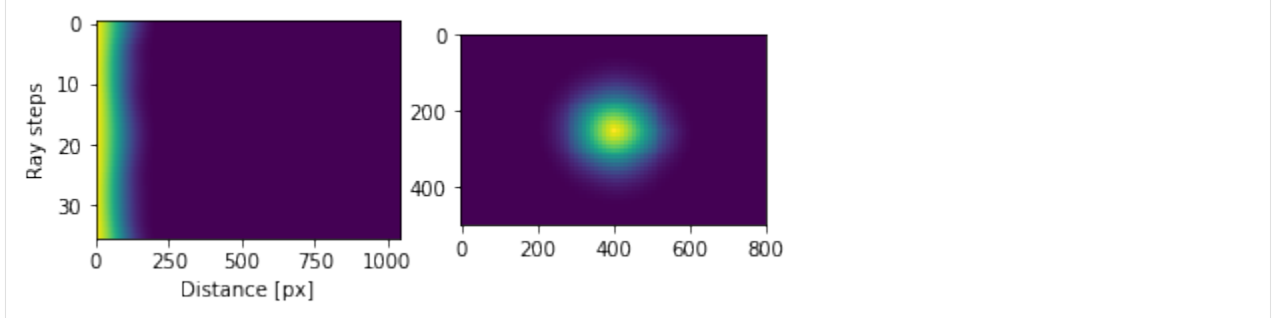


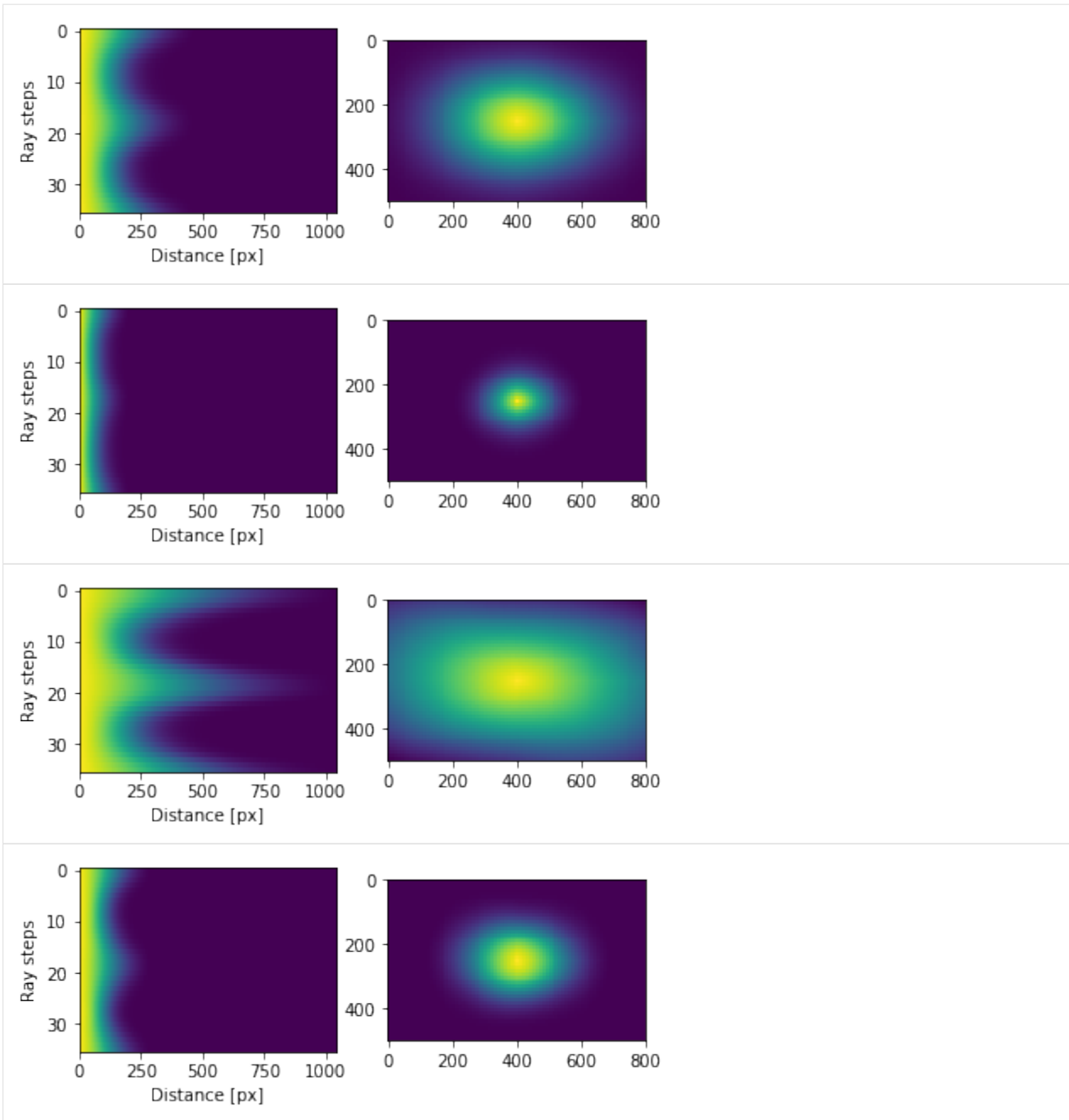


## K-Means

```
[21]: model, list_mean_cdf = tl_rg.transform_rays_model_sets_mean_cdf_kmeans(list_rays, 5)
max_len = max([np.asarray(l_cdf).shape[1] for _, l_cdf in list_mean_cdf])

for mean, list_cdf in list_mean_cdf:
    cdist = np.zeros((len(list_cdf), max_len))
    cdist[:, :len(list_cdf[0])] = np.array(list_cdf)
    plt.figure(figsize=(6, 2))
    plt.subplot(1, 2, 1), plt.imshow(cdist, aspect='auto'), plt.xlim([0, max_len]),
    plt.ylabel('Ray steps'), plt.xlabel('Distance [px]')
    plt.subplot(1, 2, 2), plt.imshow(compute_prior_map(cdist, step=10))
```





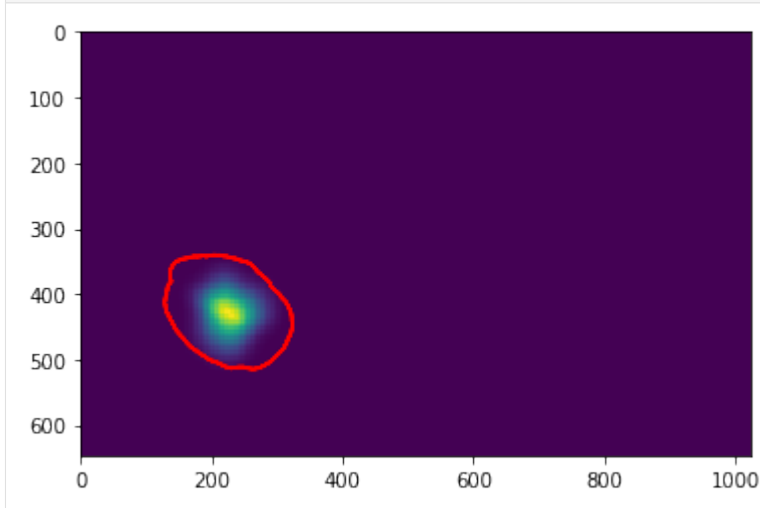
### Show Shape prior with sample egg

```
[43]: seg_object = (seg == 3)
centre = ndimage.measurements.center_of_mass(seg_object)
ray = tl_fts.compute_ray_features_seg2d(seg_object, centre, edge='down')
_, shift = tl_fts.shift_ray_features(ray)
print ('centre: %s' % repr(centre))
print ('angle shift: %f' % shift)
```

```
centre: (426.3299148683429, 224.6435953276579)
angle shift: 220.0
```

```
[44]: prior_map = np.zeros(seg_object.shape)
      error_pos = []
      for i in np.arange(prior_map.shape[0], step=5):
          for j in np.arange(prior_map.shape[1], step=5):
              prior_map[i:i+5, j:j+5] = tl_rg.compute_shape_prior_table_cdf([i, j], cdist,
              ↪centre, angle_shift=shift)

      _ = plt.imshow(prior_map), plt.contour(seg_object, colors='r')
```



```
[ ]:
```

### 1.3.4 Object segmentation with Region Growing with Shape Prior

Region growing is a classical image segmentation method based on hierarchical region aggregation using local similarity rules. Our proposed method differs from classical region growing in three important aspects. First, it works on the level of superpixels instead of pixels, which leads to a substantial speedup. Second, our method uses learned statistical shape properties which encourage growing leading to plausible shapes. In particular, we use ray features to describe the object boundary. Third, our method can segment multiple objects and ensure that the segmentations do not overlap. The problem is represented as an energy minimization and is solved either greedily, or iteratively using GraphCuts.

Borovec, J., Kybic, J., & Sugimoto, A. (2017). **Region growing using superpixels with learned shape prior**. Journal of Electronic Imaging. Dvorak, J. et al. (2018). *Volume estimation from single images: an application to pancreatic islets*. Image Analysis & Stereology.

```
[1]: %matplotlib inline
      import os, sys, glob
      import pickle
      import numpy as np
      import pandas as pd
      from PIL import Image
      from scipy import ndimage
      from skimage import segmentation as sk_seg
      from skimage import io, measure
      import matplotlib.pyplot as plt
```

```
[2]: sys.path += [os.path.abspath('.'), os.path.abspath('../')] # Add path to root
import imsegm.utilities.data_io as tl_io
import imsegm.utilities.drawing as tl_visu
import imsegm.superpixels as seg_spx
import imsegm.region_growing as seg_rg
import imsegm.graph_cuts as seg_gc
import imsegm.pipelines as seg_pipe
```

## Loading data

```
[3]: COLORS = 'bgrmyck'
PATH_IMAGES = os.path.join(tl_io.update_path('data-images'), 'langerhans_islets')
PATH_DATA = tl_io.update_path('data-images', absolute=True)
PATH_OUT = tl_io.update_path('output', absolute=True)
PATH_MEASURED_RAYS = os.path.join(PATH_IMAGES, 'islets_ray-shapes_1400x1050.csv')
print ([os.path.basename(p) for p in glob.glob(os.path.join(PATH_IMAGES, '*')) if os.
↳ path.isdir(p)])
# dir_annot = os.path.expanduser('~/.Dropbox/Workspace/segment_Medical/Ovary-eggs/mask_
↳ 2d_slice_complete_ind_egg')

['image_red', 'image', 'binary-masks_1400x', 'binary-masks', 'annot']
```

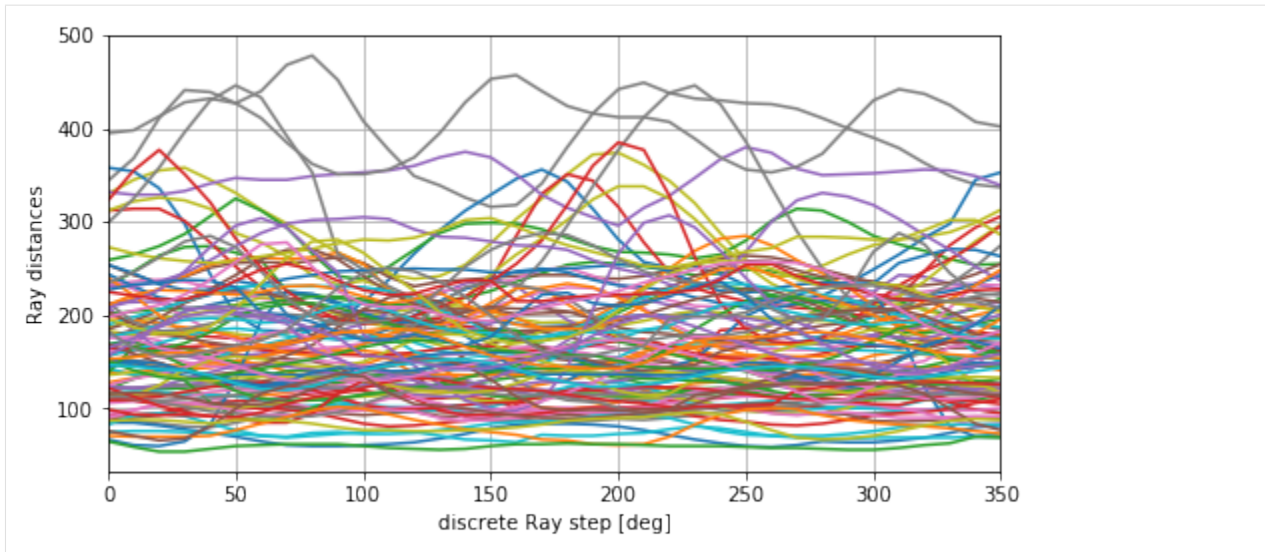
```
[4]: DIR_IMAGES = os.path.join(PATH_IMAGES, 'image')
DIR_SEGMS = os.path.join(PATH_IMAGES, 'segm')
DIR_ANNOTS = os.path.join(PATH_IMAGES, 'annot')
```

## Estimate shape models

```
[5]: def compute_prior_map(cdist, size=(500, 800), step=5):
    prior_map = np.zeros(size)
    centre = np.array(size) / 2
    for i in np.arange(prior_map.shape[0], step=step):
        for j in np.arange(prior_map.shape[1], step=step):
            prior_map[i:i+step, j:j+step] = seg_rg.compute_shape_prior_table_cdf([i,
↳ j], cdist, centre, angle_shift=0)
    return prior_map
```

Visualisation of all normalised Ray features with given angular step.

```
[7]: df = pd.read_csv(PATH_MEASURED_RAYS, index_col=0)
list_rays = df.values.tolist()
fig = plt.figure(figsize=(8, 4))
plt.plot(np.linspace(0, 360, len(list_rays[0]) + 1)[: -1], np.array(list_rays).T, '-')
_ = plt.xlim([0, 350]), plt.grid(), plt.xlabel('discrete Ray step [deg]'), plt.ylabel(
↳ 'Ray distances')
```



## Computing shape prior model

Estimate and exporting model...

```
[8]: model_rg, list_mean_cdf = seg_rg.transform_rays_model_sets_mean_cdf_mixture(list_rays,
    ↪ 5)
    # model, list_mean_cdf = tl_rg.transform_rays_model_sets_mean_cdf_kmeans(list_rays,
    ↪ 25)
    PATH_MODEL_MIXTURE = os.path.join(PATH_DATA, 'RG2SP_islets_single-model.pkl')

    with open(PATH_MODEL_MIXTURE, 'w') as fp:
        pickle.dump({'name': 'set_cdfs', 'cdfs': list_mean_cdf, 'mix_model': model_rg},
    ↪ fp)
```

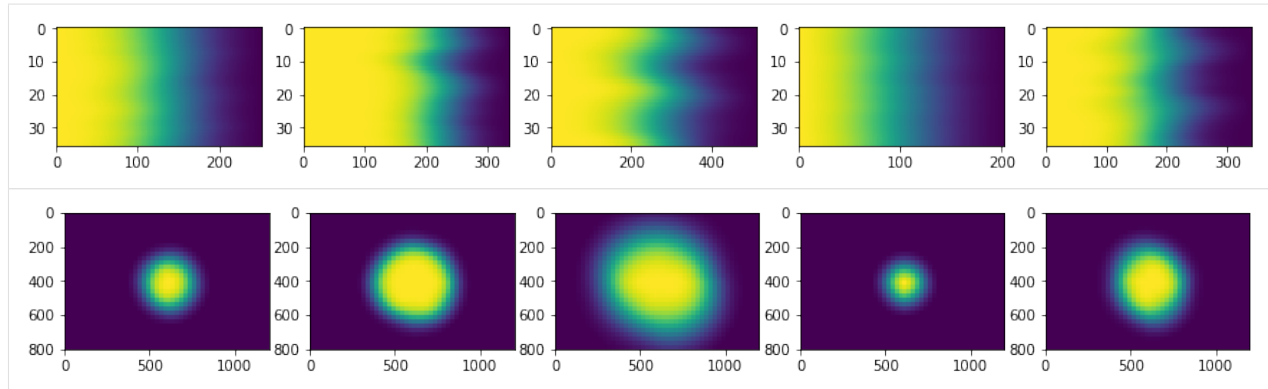
Try to load just exported model to verify that it is readable.

```
[9]: file_model = pickle.load(open(PATH_MODEL_MIXTURE, 'r'))
    print (file_model.keys())
    list_mean_cdf = file_model['cdfs']
    model_rg = file_model['mix_model']

    ['mix_model', 'cdfs', 'name']
```

Visualise the set of components from Mixture model and their back reconstructions

```
[12]: subfig_per_row = 8
    subfig_per_col = int(np.ceil(len(list_mean_cdf) / float(subfig_per_row)))
    fig = plt.figure(figsize=(3 * subfig_per_row, 1.5 * subfig_per_col))
    for i in range(len(list_mean_cdf)):
        plt.subplot(subfig_per_col, subfig_per_row, i+1), plt.imshow(list_mean_cdf[i][1],
    ↪ aspect='auto')
    fig = plt.figure(figsize=(3 * subfig_per_row, 2 * subfig_per_col))
    for i in range(len(list_mean_cdf)):
        plt.subplot(subfig_per_col, subfig_per_row, i+1)
        plt.imshow(compute_prior_map(list_mean_cdf[i][1], size=(800, 1200), step=25))
```

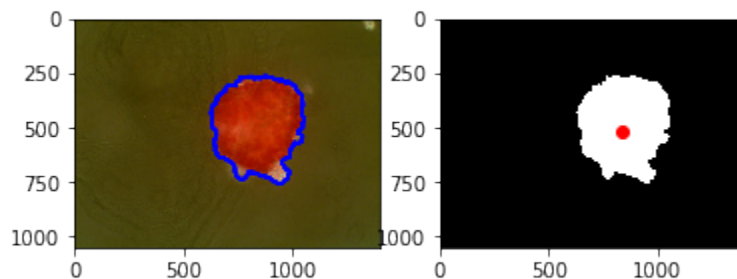


## Loading Image

```
[13]: name = 'Lh05-09'
img = io.imread(os.path.join(DIR_IMAGES, name + '.jpg'))
annot = measure.label(io.imread(os.path.join(DIR_ANNOTS, name + '.png')))
centers = np.array([ndimage.measurements.center_of_mass(annot == l) for l in range(1,
↳ np.max(annot) + 1)])
centers += (np.random.random(centers.shape) - 0.5) * 100
print('centres: %s' % repr(centers))
FIG_SIZE = (8. * np.array(img.shape[:2]) / np.max(img.shape))[:2]

centres: array([[ 513.78095009,  837.2978496 ]])
```

```
[15]: plt.subplot(1, 2, 1)
plt.imshow(img), plt.contour(annot, colors='b')
plt.subplot(1, 2, 2)
plt.imshow(annot, cmap=plt.cm.Greys_r)
plt.plot(centers[:, 1], centers[:, 0], 'or')
_= plt.xlim([0, img.shape[1]]), plt.ylim([img.shape[0], 0])
```





## Compute superpixels and probabilities

```
[16]: img_tf = img.copy()
img_tf[:, :, 2] = img_tf[:, :, 0]
SLIC_SIZE = 25
SLIC_REGUL = 0.2
DICT_FEATURES = {'color': ['mean', 'median']}
slic, features = seg_pipe.compute_color2d_superpixels_features(img_tf, dict_
↳ features=DICT_FEATURES, sp_size=SLIC_SIZE, sp_regul=SLIC_REGUL)
features = np.nan_to_num(features)
print ('feature space: %s' % repr(features.shape))

model_seg = seg_gc.estim_class_model(features, 2, pca_coef=None, estim_model='GMM',
↳ use_scaler=True)
slic_preds = model_seg.predict_proba(features)
print ('prediction shape: %s' % repr(slic_preds.shape))

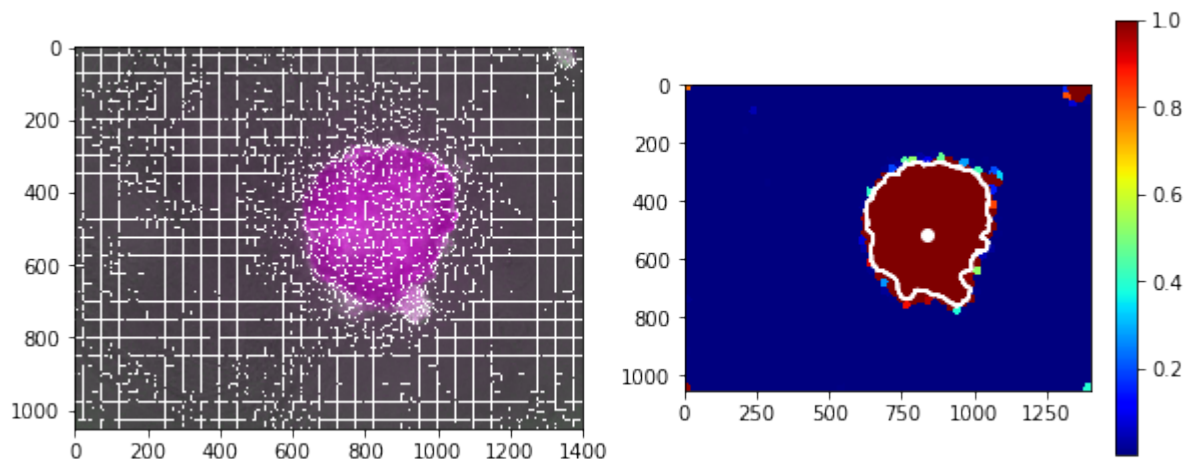
feature space: (2329, 6)
prediction shape: (2329, 2)
```

```
[17]: grid_x, grid_y = np.mgrid[0:img.shape[0], 0:img.shape[1]]
grid_x -= int(img.shape[0] / 2.)
grid_y -= int(img.shape[1] / 2.)
dist_center = np.sqrt(grid_x ** 2 + grid_y ** 2)
img_bg_score = [np.mean(dist_center * slic_preds[:, i][slic]) for i in range(slic_
↳ preds.shape[1])]
slic_prob_fg = slic_preds[:, np.argmin(img_bg_score)]
print ('image BG scores: %s' % repr(img_bg_score))

plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1), plt.imshow(sk_segmark_boundaries(img_tf, slic, color=(1, 1,
↳ 1)))

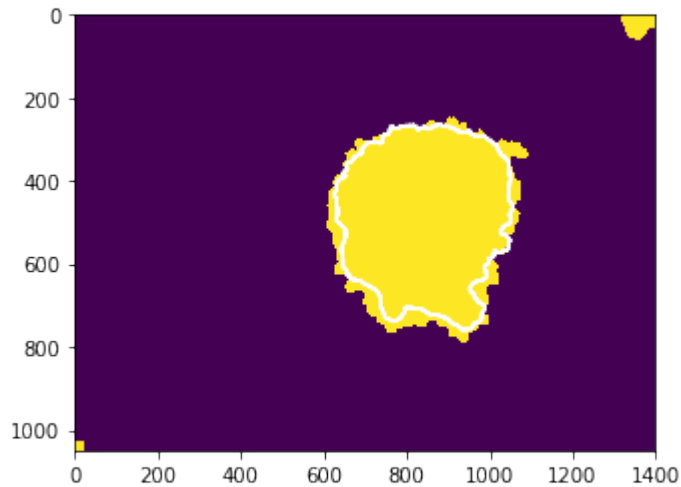
plt.subplot(1, 2, 2)
plt.imshow(slic_prob_fg[slic], cmap=plt.cm.jet), plt.colorbar()
plt.contour(annot, colors='w')
plt.plot(centers[:, 1], centers[:, 0], 'ow')
_ = plt.xlim([0, img.shape[1]]), plt.ylim([img.shape[0], 0])

image BG scores: [441.35775946597442, 30.228474053176999]
```



```
[29]: # model_seg, _, _, _ = seg_pipe.train_classif_color2d_slic_features([img_red],
↳[annot], DICT_FEATURES, sp_size=SLIC_SIZE, sp_regul=SLIC_REGUL)
segm_gc, _ = seg_pipe.segment_color2d_slic_features_model_graphcut(img_tf, model_seg,
↳DICT_FEATURES, sp_size=SLIC_SIZE, sp_regul=SLIC_REGUL, gc_edge_type='color', gc_
↳regul=20.)

_= plt.imshow(segm_gc), plt.contour(annot, colors='w')
# plt.subplot(1, 2, 2), plt.imshow(seg_soft[:, :, 1])
```



```
[24]: from IPython.html import widgets
from IPython.display import display

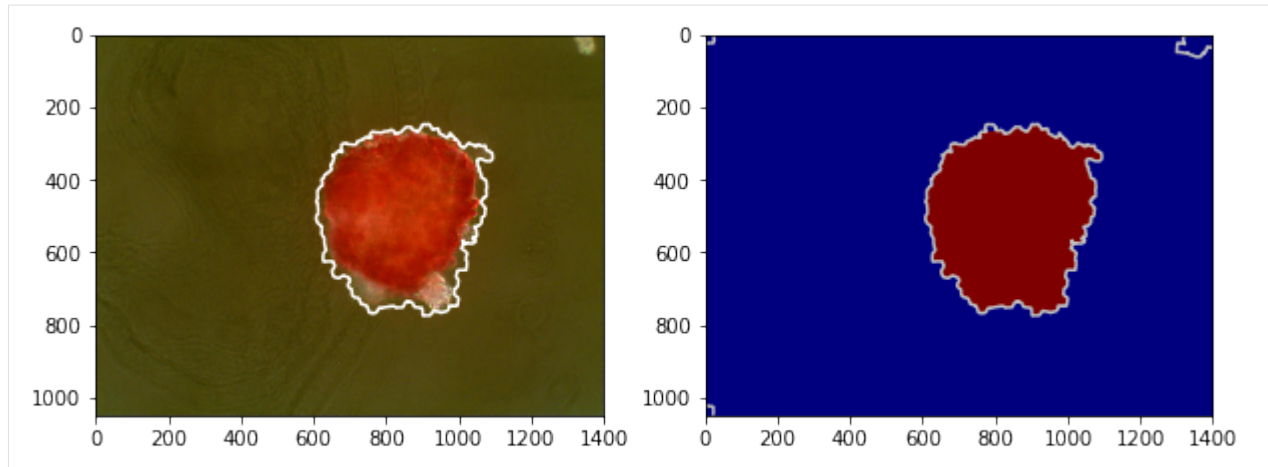
:0: FutureWarning: IPython widgets are experimental and may change in the future.
```

## Region growing - GraphCut

```
[26]: debug_gc_mm = {}
labels_gc = seg_rg.region_growing_shape_slic_graphcut(slic, slic_prob_fg, centers,
↳(model_rg, list_mean_cdf), 'set_cdfs',
coef_shape=2., coef_pairwise=5., prob_label_trans=[0.1, 0.03], optim_
↳global=False,
allow_obj_swap=False, dict_thresholds=seg_rg.RG2SP_THRESHOLDS, nb_
↳iter=100, debug_history=debug_gc_mm)
segm_obj = labels_gc[slic]
print (debug_gc_mm.keys())

fig = plt.figure(figsize=(10, 4))
_= plt.subplot(1, 2, 1), plt.imshow(img), plt.contour(segm_obj, levels=np.unique(segm_
↳obj), colors='w')
_= plt.subplot(1, 2, 2), plt.imshow(segm_obj, cmap=plt.cm.jet), plt.contour(segm_gc,
↳levels=np.unique(segm_gc), colors='#bfbfbf')

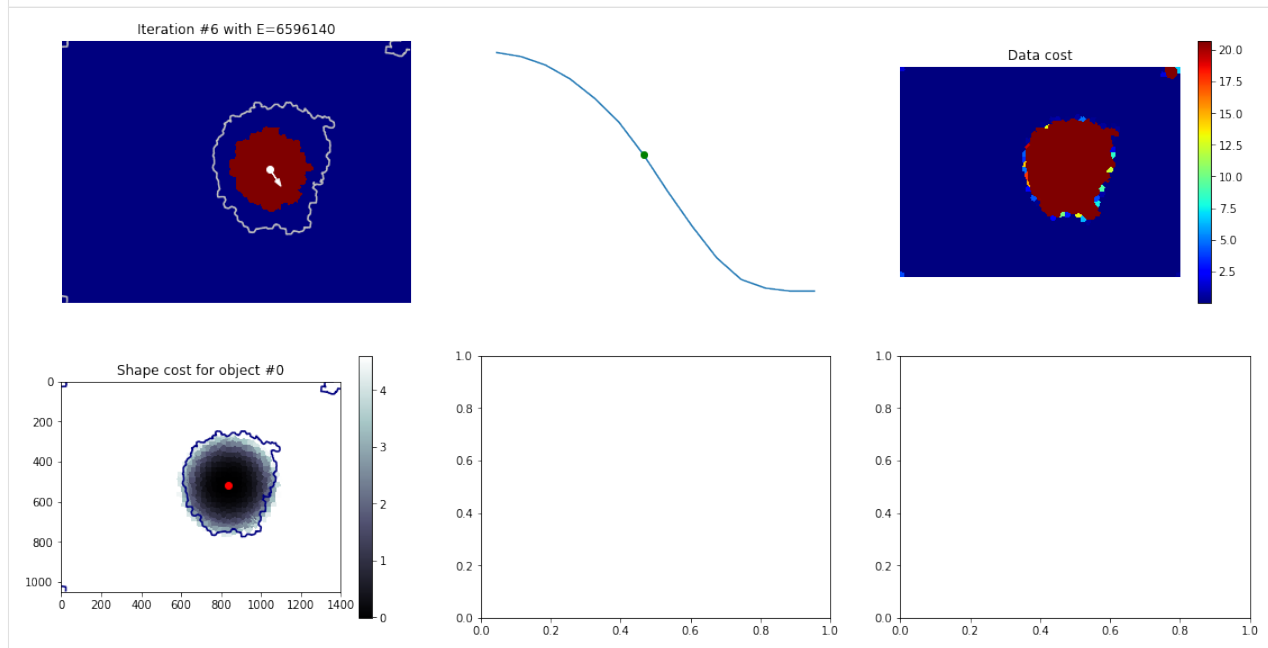
['shifts', 'lut_data_cost', 'lut_shape_cost', 'criteria', 'labels', 'centres']
```



Interactive visualisation - over iterations

```
[27]: def show_partial(i):
      _ = plt.close(), tl_visu.figure_rg2sp_debug_complete(segm_gc, slic, debug_gc_mm, i,
      ↪ max_size=5)
      # show the interact
      widgets.interact(show_partial, i=(0, len(debug_gc_mm['criteria']) -1))
```

```
[27]: <function __main__.show_partial>
```



```
[ ]:
```

### 1.3.5 Object segmentation with Region Growing with Shape Prior

Region growing is a classical image segmentation method based on hierarchical region aggregation using local similarity rules. Our proposed method differs from classical region growing in three important aspects. First, it works on the level of superpixels instead of pixels, which leads to a substantial speedup. Second, our method uses learned statistical shape properties which encourage growing leading to plausible shapes. In particular, we use ray features to describe the object boundary. Third, our method can segment multiple objects and ensure that the segmentations do not overlap. The problem is represented as an energy minimization and is solved either greedily, or iteratively using GraphCuts.

Borovec, J., Kybic, J., & Sugimoto, A. (2017). **Region growing using superpixels with learned shape prior**. Journal of Electronic Imaging.

```
[1]: %matplotlib inline
import os, sys, glob
import pickle
import numpy as np
import pandas as pd
from PIL import Image
# from scipy import spatial, ndimage
from skimage import segmentation as sk_seg
import matplotlib.pyplot as plt
```

```
[2]: sys.path += [os.path.abspath('.'), os.path.abspath('../')] # Add path to root
import imsegm.utilities.data_io as tl_io
import imsegm.utilities.drawing as tl_visu
import imsegm.superpixels as seg_spx
import imsegm.region_growing as seg_rg
```

#### Loading data

```
[3]: COLORS = 'bgrmyck'
RG2SP_THRESHOLDS = seg_rg.RG2SP_THRESHOLDS
PATH_IMAGES = os.path.join(tl_io.update_path('data-images'), 'drosophila_ovary_slice')
PATH_DATA = tl_io.update_path('data-images', absolute=True)
PATH_OUT = tl_io.update_path('output', absolute=True)
PATH_MEASURED_RAYS = os.path.join(PATH_IMAGES, 'eggs_ray-shapes.csv')
print ([os.path.basename(p) for p in glob.glob(os.path.join(PATH_IMAGES, '*')) if os.
↳ path.isdir(p)])

['center_levels', 'image', 'annot_struct', 'ellipse_fitting', 'annot_eggs', 'segm_rgb
↳ ', 'segm', 'image_cut-stage-2']
```

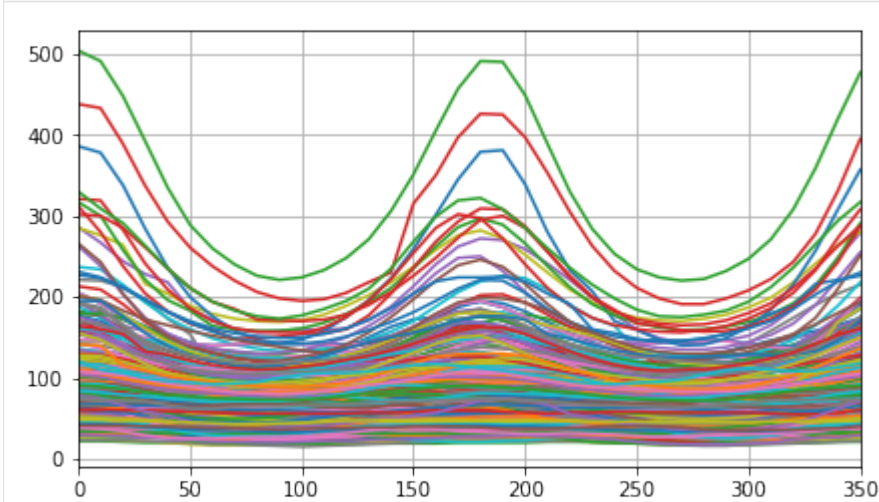
```
[4]: DIR_IMAGES = os.path.join(PATH_IMAGES, 'image')
DIR_SEGMS = os.path.join(PATH_IMAGES, 'segm')
DIR_ANNOTS = os.path.join(PATH_IMAGES, 'annot_eggs')
DIR_CENTERS = os.path.join(PATH_IMAGES, 'center_levels')
```

## Estimate shape models

```
[5]: def compute_prior_map(cdist, size=(500, 800), step=5):
    prior_map = np.zeros(size)
    centre = np.array(size) / 2
    for i in np.arange(prior_map.shape[0], step=step):
        for j in np.arange(prior_map.shape[1], step=step):
            prior_map[i:i+step, j:j+step] = seg_rg.compute_shape_prior_table_cdf([i,
            ↪ j], cdist, centre, angle_shift=0)
    return prior_map
```

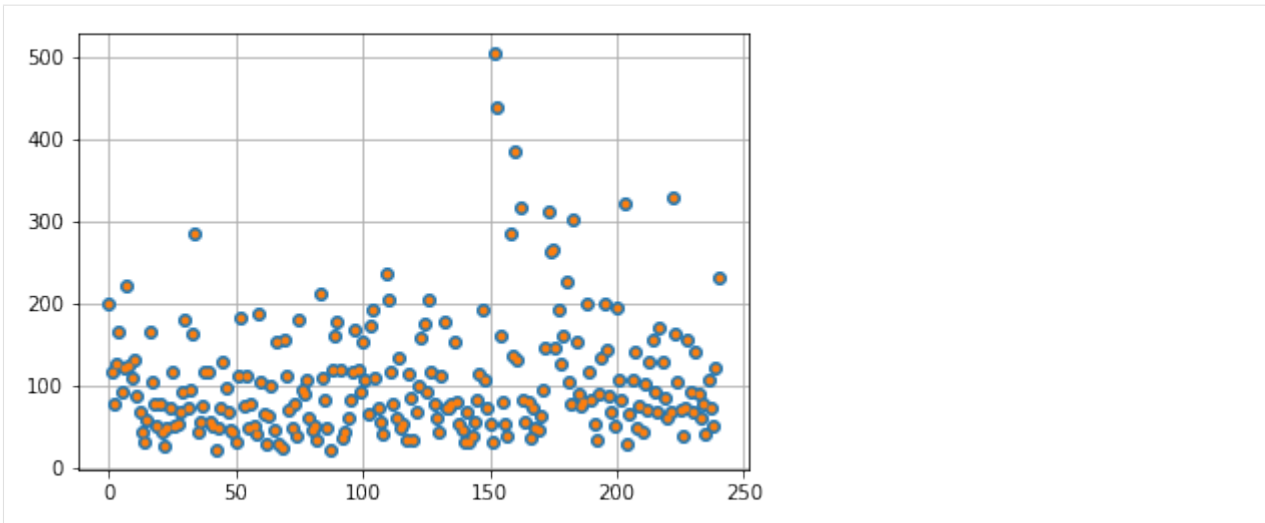
Visualisation of all normalised Ray features with given angular step.

```
[7]: df = pd.read_csv(PATH_MEASURED_RAYS, index_col=0)
list_rays = df.values.tolist()
fig = plt.figure(figsize=(7, 4))
plt.plot(np.linspace(0, 360, len(list_rays[0]) + 1)[: -1], np.array(list_rays).T, '-')
_ = plt.xlim([0, 350]), plt.grid()
```



Visualise the maxima for each egg of measured Ray features.

```
[8]: maxima = np.max(list_rays, axis=1)
plt.plot(maxima, 'o'), plt.grid()
# list_rays += [ray * (1 + np.random.random() / 10.) for ray in np.array(list_
    ↪ rays)[maxima > 250]]
# list_rays += [ray * (1 + np.random.random() / 10.) for ray in np.array(list_
    ↪ rays)[maxima > 250]]
_ = plt.plot(np.max(list_rays, axis=1), '.')
```

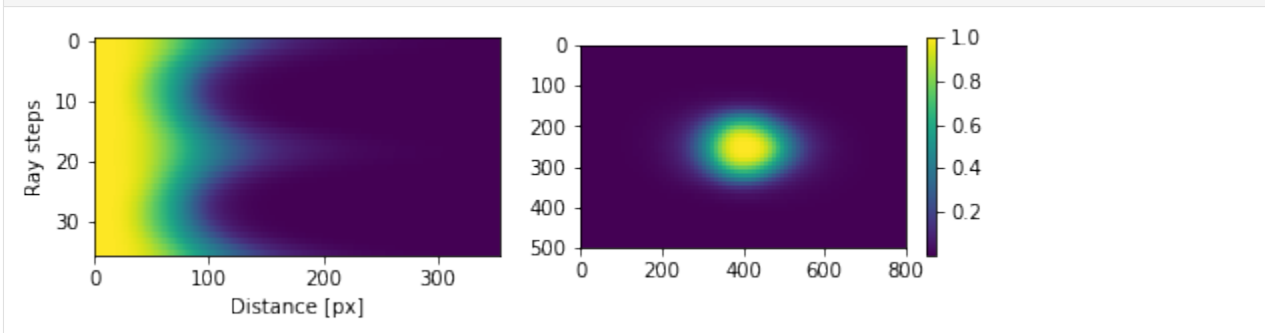


Estimate and exporting model...

```
[9]: # list_cdf = tl_rg.transform_rays_model_cdf_histograms(list_rays, nb_bins=15)
mm, list_cdf = seg_rg.transform_rays_model_cdf_mixture(list_rays, 15)
cdf = np.array(np.array(list_cdf))
PATH_MODEL_SINGLE = os.path.join(PATH_DATA, 'RG2SP_eggs_single-model.pkl')

with open(PATH_MODEL_SINGLE, 'w') as fp:
    pickle.dump({'name': 'cdf', 'cdfs': cdf, 'mix_model': mm}, fp)
```

```
[11]: fig = plt.figure(figsize=(8, 2))
plt.subplot(1, 2, 1), plt.imshow(cdf, aspect='auto'), plt.ylabel('Ray steps'), plt.
    xlabel('Distance [px]')
_ = plt.subplot(1, 2, 2), plt.imshow(compute_prior_map(cdf, step=10)), plt.colorbar()
# fig.savefig(os.path.join(PATH_OUT, 'shape-prior.pdf'), bbox_inches='tight')
```



## Computing shape prior model

Estimate and exporting model...

```
[16]: model, list_mean_cdf = seg_rg.transform_rays_model_sets_mean_cdf_mixture(list_rays,
    15)
# model, list_mean_cdf = tl_rg.transform_rays_model_sets_mean_cdf_kmeans(list_rays,
    25)
PATH_MODEL_MIXTURE = os.path.join(PATH_DATA, 'RG2SP_eggs_mixture-model.pkl')
```

(continues on next page)

(continued from previous page)

```
with open(PATH_MODEL_MIXTURE, 'w') as fp:
    pickle.dump({'name': 'set_cdfs', 'cdfs': list_mean_cdf, 'mix_model': model}, fp)
```

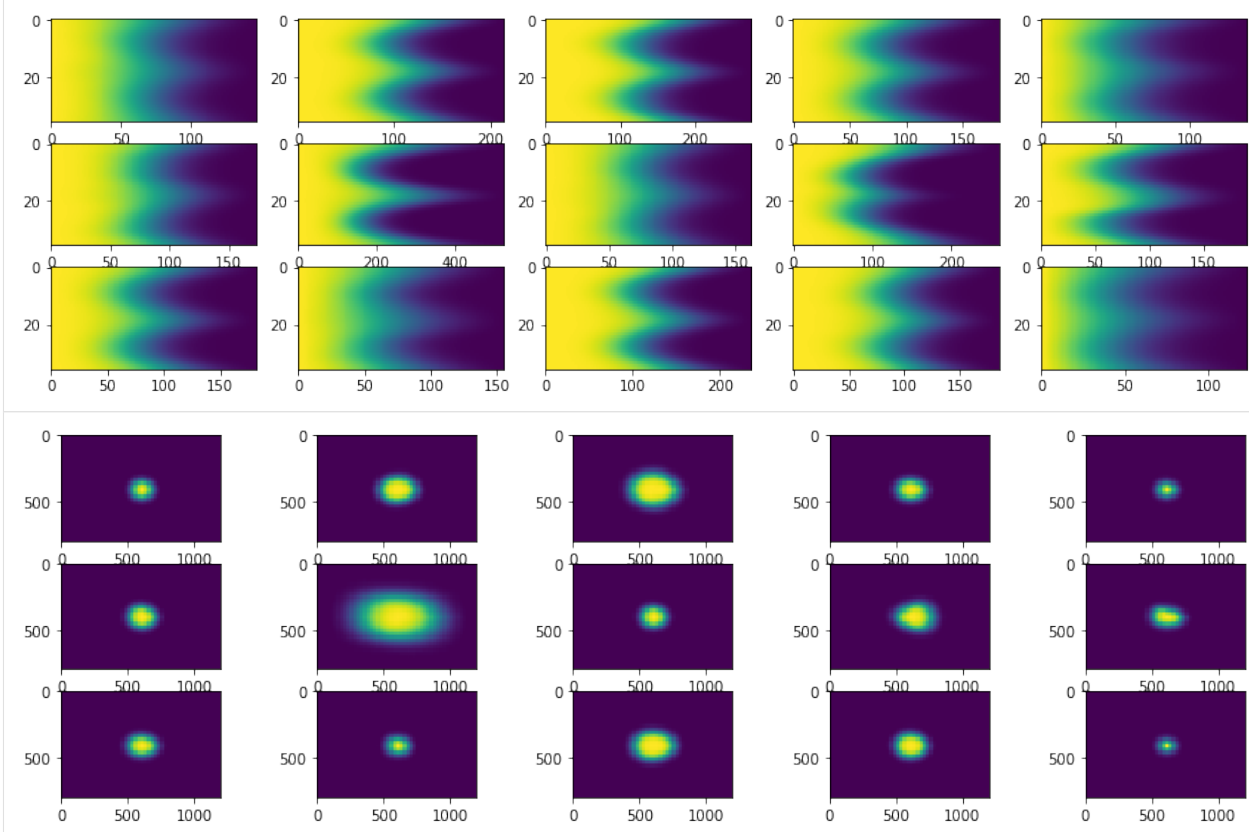
Try to load just exported model to verify that it is readable.

```
[17]: file_model = pickle.load(open(PATH_MODEL_MIXTURE, 'r'))
print (file_model.keys())
list_mean_cdf = file_model['cdfs']
model = file_model['mix_model']

['mix_model', 'cdfs', 'name']
```

Visualise the set of components from Mixture model and their back reconstructions

```
[19]: subfig_per_row = 5
subfig_per_col = int(np.ceil(len(list_mean_cdf) / float(subfig_per_row)))
fig = plt.figure(figsize=(3 * subfig_per_row, 1.5 * subfig_per_col))
for i in range(len(list_mean_cdf)):
    plt.subplot(subfig_per_col, subfig_per_row, i+1), plt.imshow(list_mean_cdf[i][1],
↪ aspect='auto')
fig = plt.figure(figsize=(3 * subfig_per_row, 1.5 * subfig_per_col))
for i in range(len(list_mean_cdf)):
    plt.subplot(subfig_per_col, subfig_per_row, i+1)
    plt.imshow(compute_prior_map(list_mean_cdf[i][1], size=(800, 1200), step=25))
```



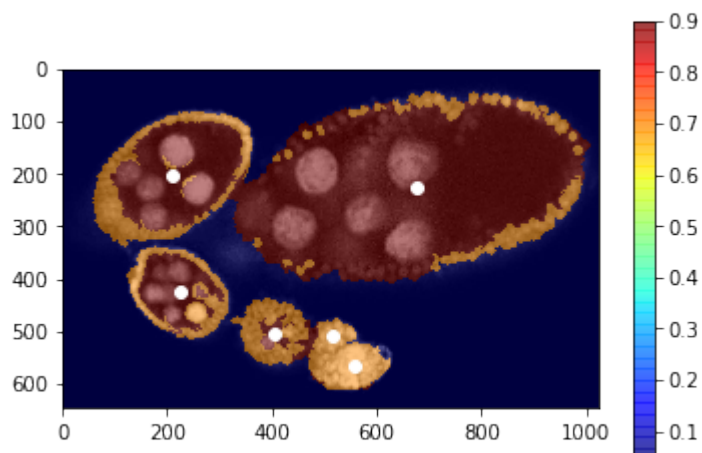
## Loading Image

```
[20]: name = 'insitu7545'
      # name = 'insitu4200'
      img = np.array(Image.open(os.path.join(DIR_IMAGES, name + '.jpg')))
      seg = np.array(Image.open(os.path.join(DIR_SEGMS, name + '.png')))
      # seg = np.array(Image.open(os.path.join(DIR_SEGMS, name + '_gc.png')))
      centers = pd.read_csv(os.path.join(DIR_CENTERS, name + '.csv'), index_col=0).values
      centers[:, [0, 1]] = centers[:, [1, 0]]
      FIG_SIZE = (6. * np.array(img.shape[:2]) / np.max(img.shape))[:-1]
      # print centers
```

Project the structure segmentation with probabbilities for each class

```
[21]: labels_fg_prob = [0.05, 0.7, 0.9, 0.9]
      # labels_fg_prob = [0.05, 0.8, 0.95, 0.95]

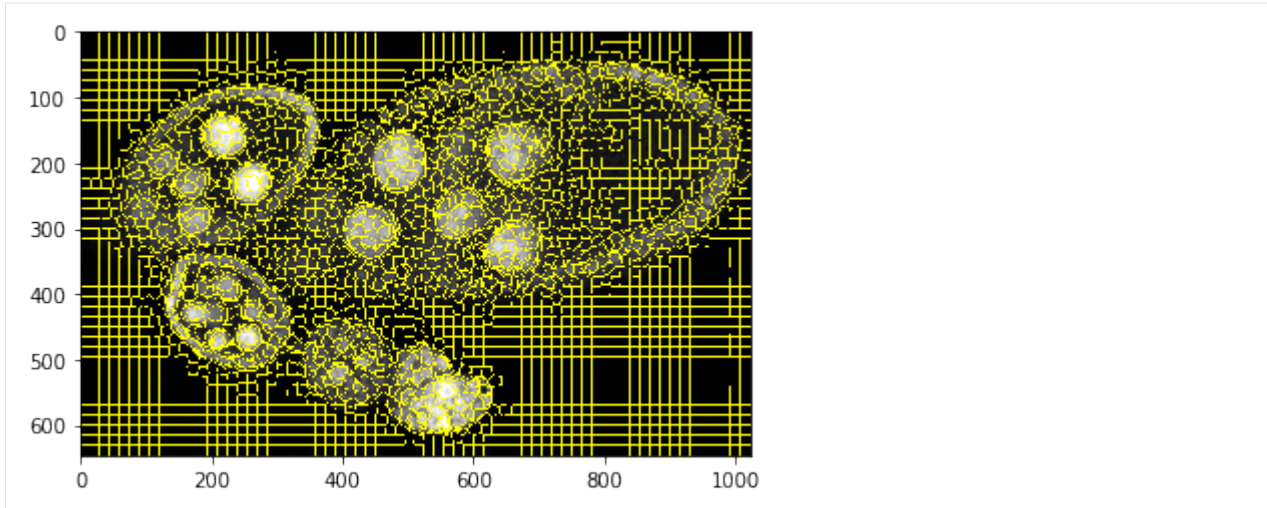
      # plt.figure(figsize=FIG_SIZE)
      plt.imshow(img[:, :, 0], cmap=plt.cm.Greys_r)
      plt.imshow(np.array(labels_fg_prob)[seg], alpha=0.5, cmap=plt.cm.jet), plt.colorbar()
      plt.plot(centers[:, 1], centers[:, 0], 'ow')
      _ = plt.xlim([0, img.shape[1]]), plt.ylim([img.shape[0], 0])
```



## Compute superpixels

```
[22]: slic = seg_spx.segment_slic_img2d(img, sp_size=15, relative_compact=0.35)
      # plt.figure(figsize=FIG_SIZE[:-1])
      _ = plt.imshow(sk_segmark_boundaries(img[:, :, 0], slic))
```





Loading ipython (interactive) visualisation functions

```
[23]: from IPython.html import widgets
      from IPython.display import display

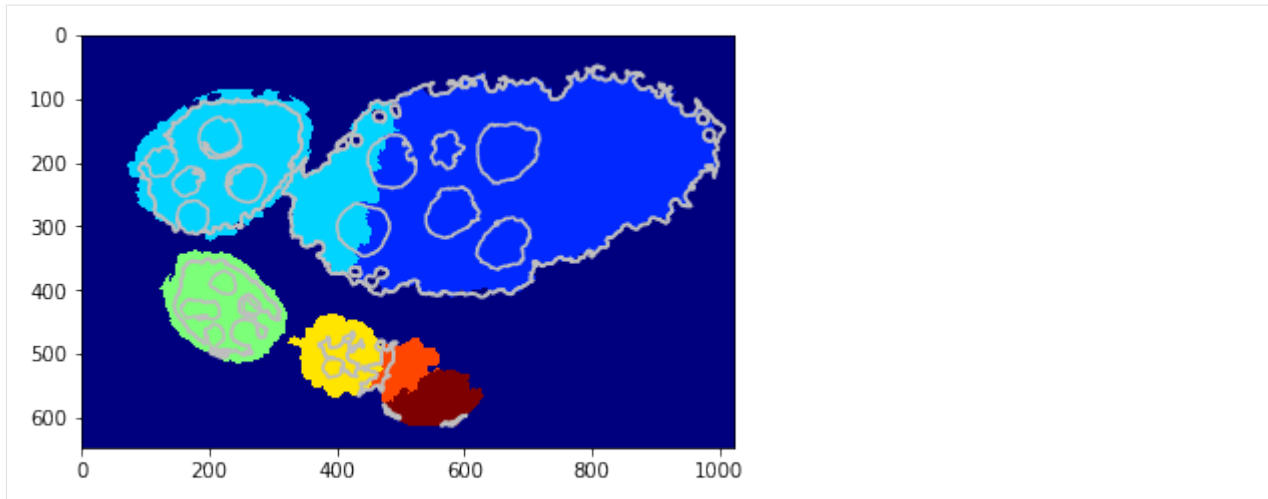
      :0: FutureWarning: IPython widgets are experimental and may change in the future.
```

## Region growing - Greedy

### Single model

```
[24]: debug_gd_lm = {}
      slic_prob_fg = seg_rg.compute_segm_prob_fg(slic, seg, labels_fg_prob)
      labels_greedy = seg_rg.region_growing_shape_slic_greedy(slic, slic_prob_fg, centers,
      ↪ (None, cdf), 'cdf',
      ↪ coef_shape=1., coef_pairwise=5., prob_label_trans=[0.1, 0.01],
      ↪ greedy_tol=2e-1,
      ↪ allow_obj_swap=True, dict_thresholds=RG2SP_THRESHOLDS, nb_
      ↪ iter=250, debug_history=debug_gd_lm)
      segm_obj = labels_greedy[slic]
      print (debug_gd_lm.keys())
      fig = plt.figure(figsize=FIG_SIZE)
      _ = plt.imshow(segm_obj, cmap=plt.cm.jet), plt.contour(seg, levels=np.unique(seg),
      ↪ colors='#bfbfbf')

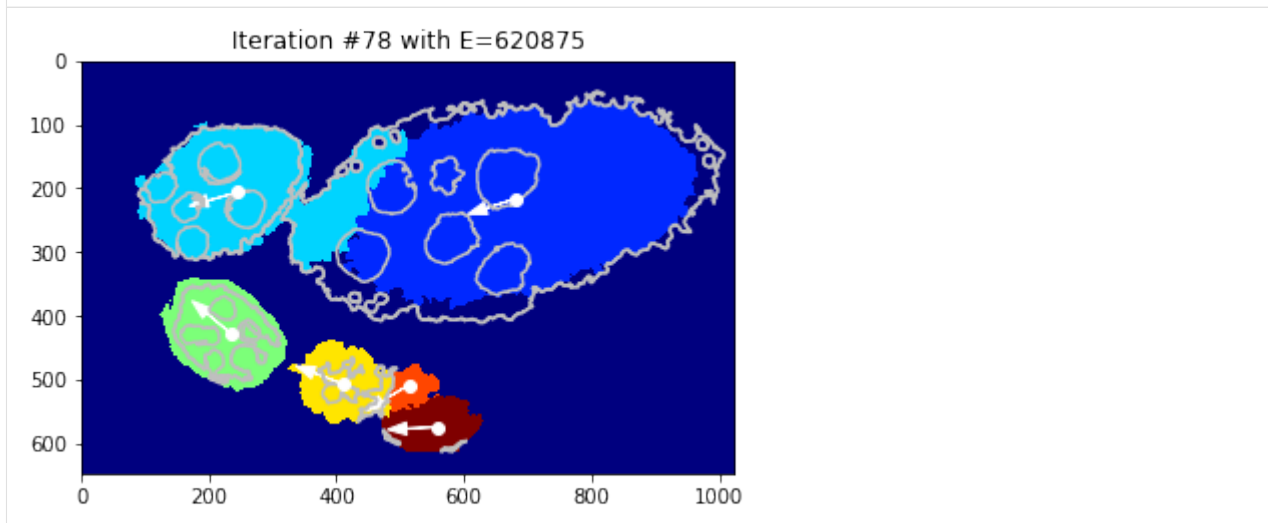
      ['shifts', 'lut_data_cost', 'lut_shape_cost', 'criteria', 'labels', 'centres']
```



Interactive visualisation - over iterations

```
[25]: def show_partial(i):
    _ = plt.close(), tl_visu.draw_rg2sp_results(plt.figure(figsize=FIG_SIZE).gca(),
    ↪ seg, slic, debug_gd_lm, i)
    # show the interact
    widgets.interact(show_partial, i=(0, len(debug_gd_lm['criteria']) -1))
```

```
[25]: <function __main__.show_partial>
```



## Mixture model

```
[26]: debug_gd_mm = {}
slic_prob_fg = seg_rg.compute_segmm_prob_fg(slic, seg, labels_fg_prob)
labels_greedy = seg_rg.region_growing_shape_slic_greedy(slic, slic_prob_fg, centers,
    ↪ (model, list_mean_cdf), 'set_cdfs',
    coef_shape=5., coef_pairwise=15., prob_label_trans=[0.1, 0.03],
    ↪ greedy_tol=3e-1,
    allow_obj_swap=True, dict_thresholds=RG2SP_THRESHOLDS, nb_
    ↪ iter=250, debug_history=debug_gd_mm)
```

(continues on next page)

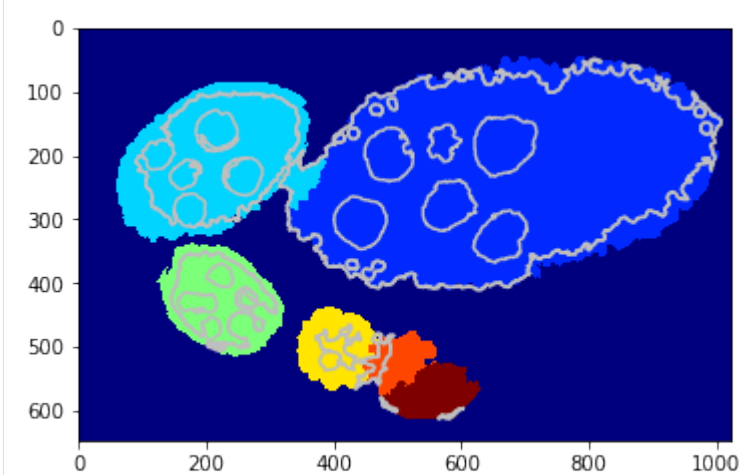
(continued from previous page)

```

segm_obj = labels_greedy[slic]
print (debug_gd_mm.keys())
fig = plt.figure(figsize=FIG_SIZE)
_= plt.imshow(segm_obj, cmap=plt.cm.jet), plt.contour(seg, levels=np.unique(seg),
↪ colors='#bfbfbf')

```

```
['shifts', 'lut_data_cost', 'lut_shape_cost', 'criteria', 'labels', 'centres']
```



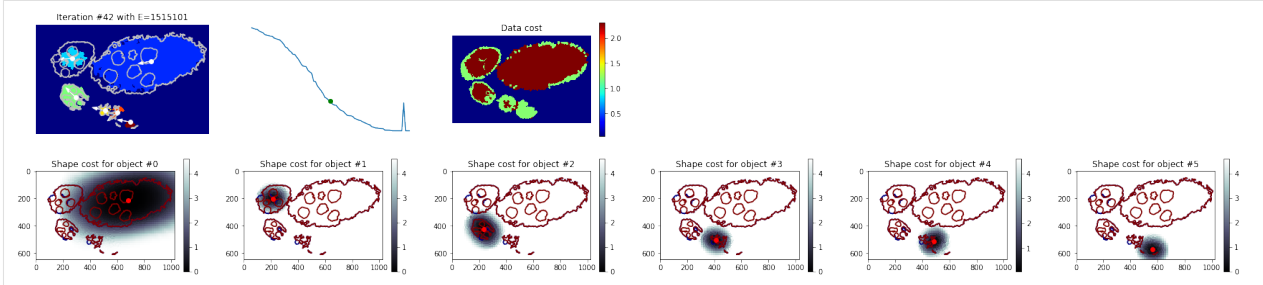
### Interactive visualisation - over iterations

```

[27]: def show_partial(i):
        _= plt.close(), tl_visu.figure_rg2sp_debug_complete(seg, slic, debug_gd_mm, i,
↪ max_size=4)
        # show the interact
        widgets.interact(show_partial, i=(0, len(debug_gd_mm['criteria']) - 1))

```

```
[27]: <function __main__.show_partial>
```

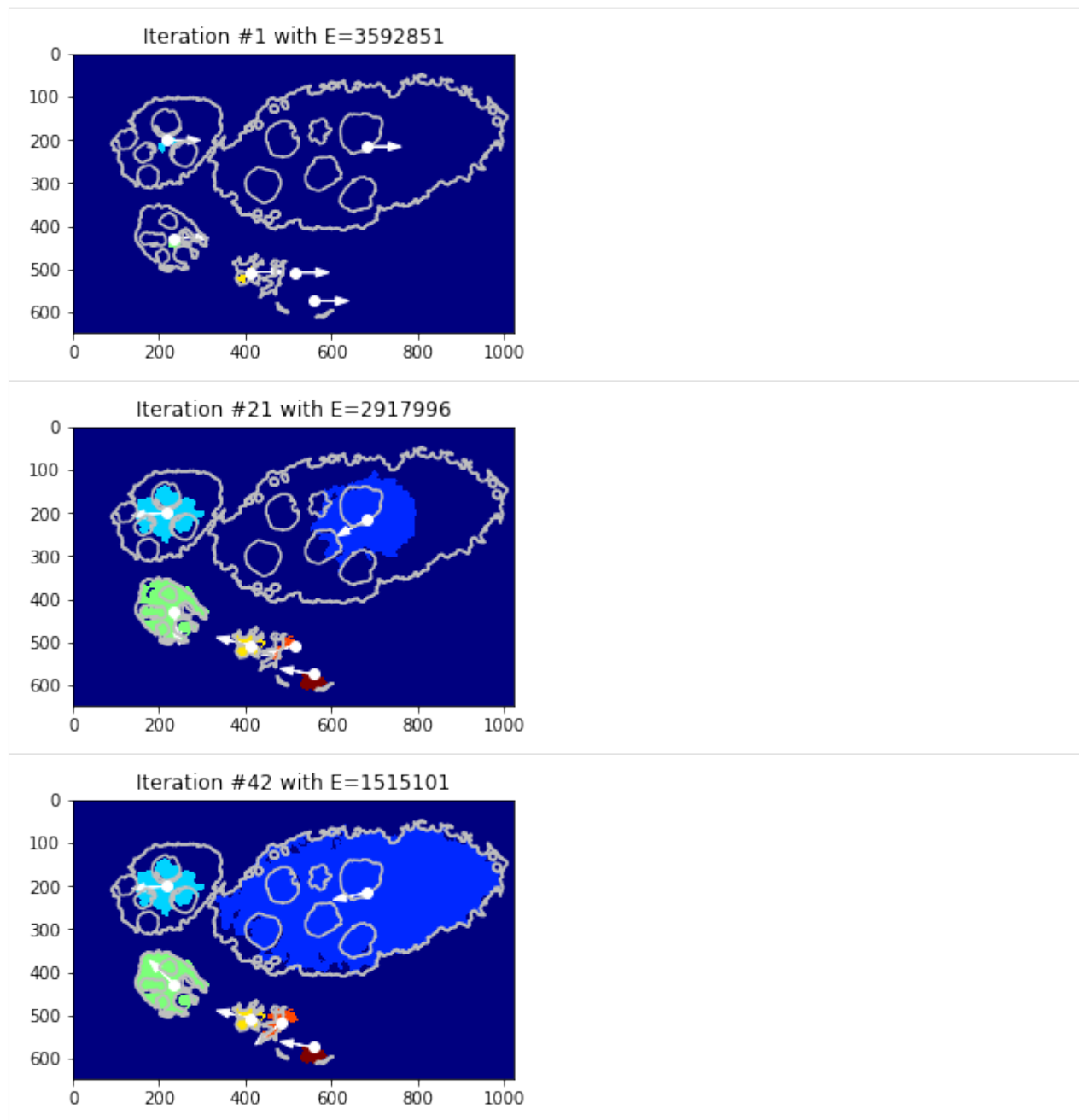


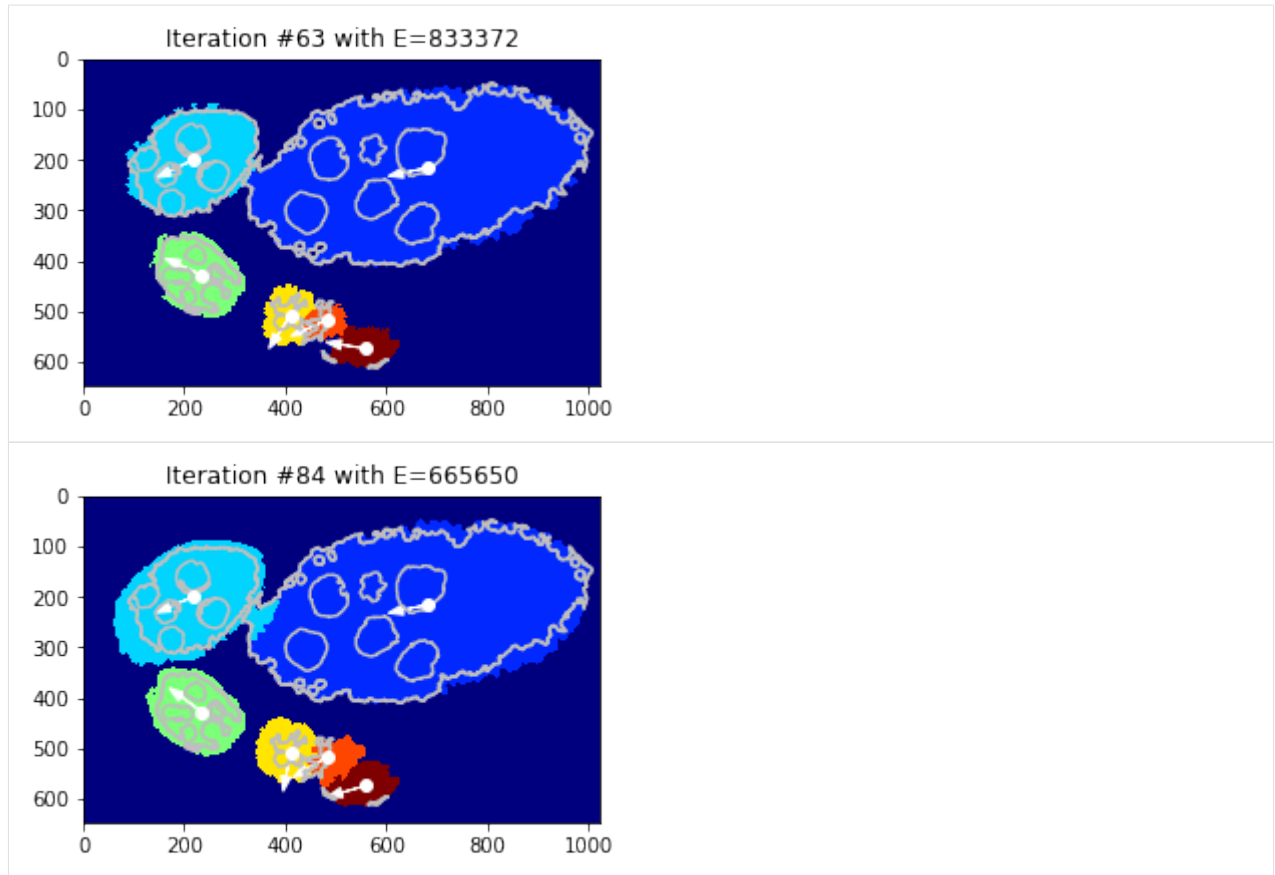
### Visualise some iterations

```

[28]: nb_iter = len(debug_gd_mm['criteria'])
        for i in np.linspace(1, nb_iter - 1, 5):
            _= tl_visu.draw_rg2sp_results(plt.figure(figsize=(6, 3)).gca(), seg, slic, debug_
↪ gd_mm, int(i))

```





## Exporting iterations

```
[35]: nb_iter = len(debug_gd_mm['criteria'])
fig_size = np.array(FIG_SIZE) * np.array([debug_gd_mm['lut_data_cost'].shape[1] - 1,
↪ 2]) / 2.
for i in range(nb_iter):
    fig = plt.figure(figsize=fig_size)
    tl_visu.draw_rg2sp_results(fig.gca(), seg, slic, debug_gd_mm, int(i))
    plt.savefig(os.path.join(PATH_OUT, 'debug-gd-mm_iter-%03d' % i))
    plt.close(fig)
```

## Region growing - GraphCut

### Single model

```
[29]: debug_gc_lm = {}
slic_prob_fg = seg_rg.compute_seg_prob_fg(slic, seg, labels_fg_prob)
labels_gc = seg_rg.region_growing_shape_slic_graphcut(slic, slic_prob_fg, centers,
↪ (None, cdf), 'cdf',
↪ coef_shape=5., coef_pairwise=25., prob_label_trans=[0.1, 0.03], optim_
↪ global=True,
↪ allow_obj_swap=True, dict_thresholds=RG2SP_THRESHOLDS, nb_iter=65,
↪ debug_history=debug_gc_lm)
```

(continues on next page)

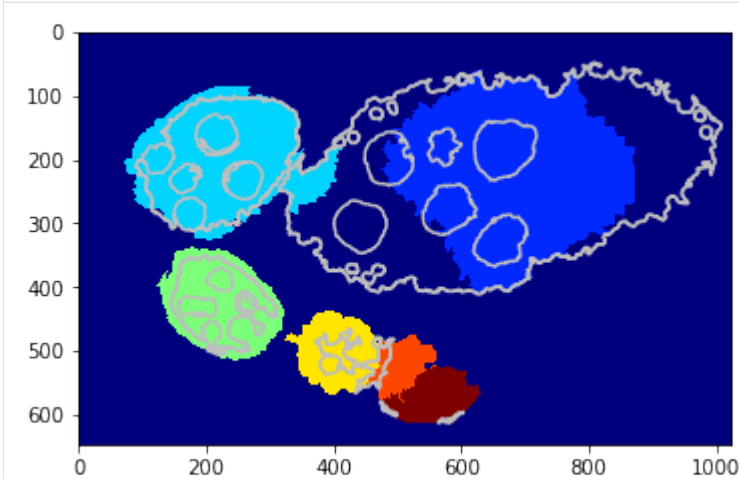
(continued from previous page)

```

segm_obj = labels_gc[slic]
print (debug_gc_lm.keys())
fig = plt.figure(figsize=FIG_SIZE)
_= plt.imshow(segm_obj, cmap=plt.cm.jet), plt.contour(seg, levels=np.unique(seg),
↪ colors='#bfbfbf')
# print debug_gc_lm.keys(), debug_gc_lm['centres']

['shifts', 'lut_data_cost', 'lut_shape_cost', 'criteria', 'labels', 'centres']

```



### Interactive visualisation - over iterations

```

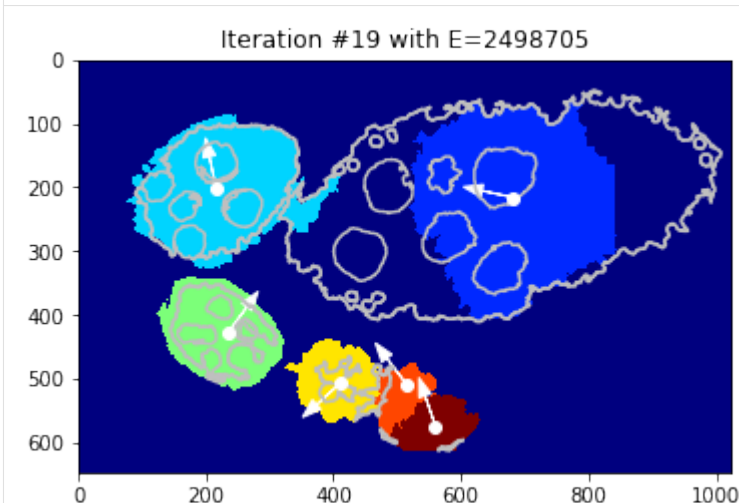
[30]: def show_partial(i):
      _= plt.close(), tl_visu.draw_rg2sp_results(plt.figure(figsize=FIG_SIZE).gca(),
↪ seg, slic, debug_gc_lm, i)
      # show the interact
      widgets.interact(show_partial, i=(0, len(debug_gc_lm['criteria']) -1))

```

```

[30]: <function __main__.show_partial>

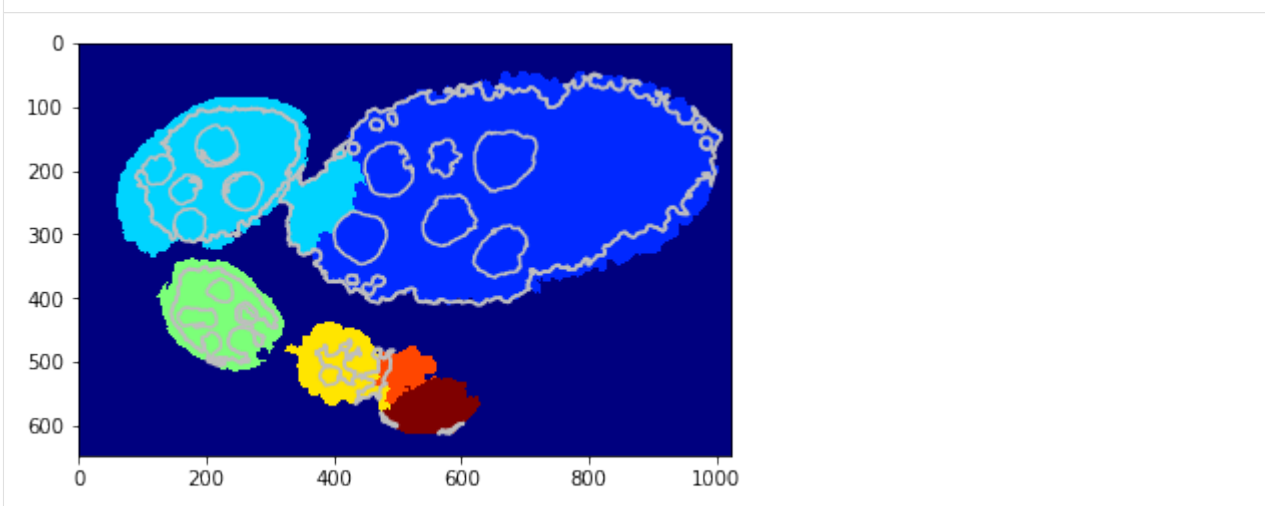
```



## Mixture model

```
[31]: debug_gc_mm = {}
      slic_prob_fg = seg_rg.compute_seg_prob_fg(slic, seg, labels_fg_prob)
      labels_gc = seg_rg.region_growing_shape_slic_graphcut(slic, slic_prob_fg, centers,
      ↪(model, list_mean_cdf), 'set_cdfs',
      ↪coef_shape=5., coef_pairwise=15., prob_label_trans=[0.1, 0.03], optim_
      ↪global=False,
      ↪allow_obj_swap=False, dict_thresholds=RG2SP_THRESHOLDS, nb_iter=65,
      ↪debug_history=debug_gc_mm)
      segm_obj = labels_gc[slic]
      print (debug_gc_mm.keys())
      fig = plt.figure(figsize=FIG_SIZE)
      _ = plt.imshow(segm_obj, cmap=plt.cm.jet), plt.contour(seg, levels=np.unique(seg),
      ↪colors='#bfbfbf')

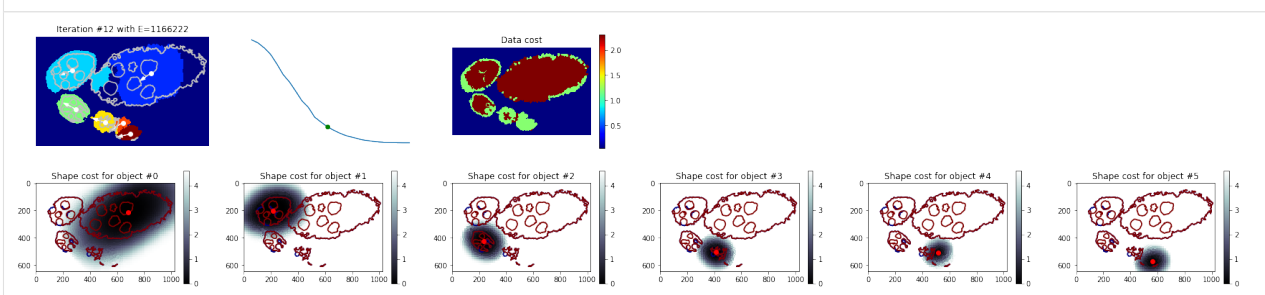
      ['shifts', 'lut_data_cost', 'lut_shape_cost', 'criteria', 'labels', 'centres']
```



## Interactive visualisation - over iterations

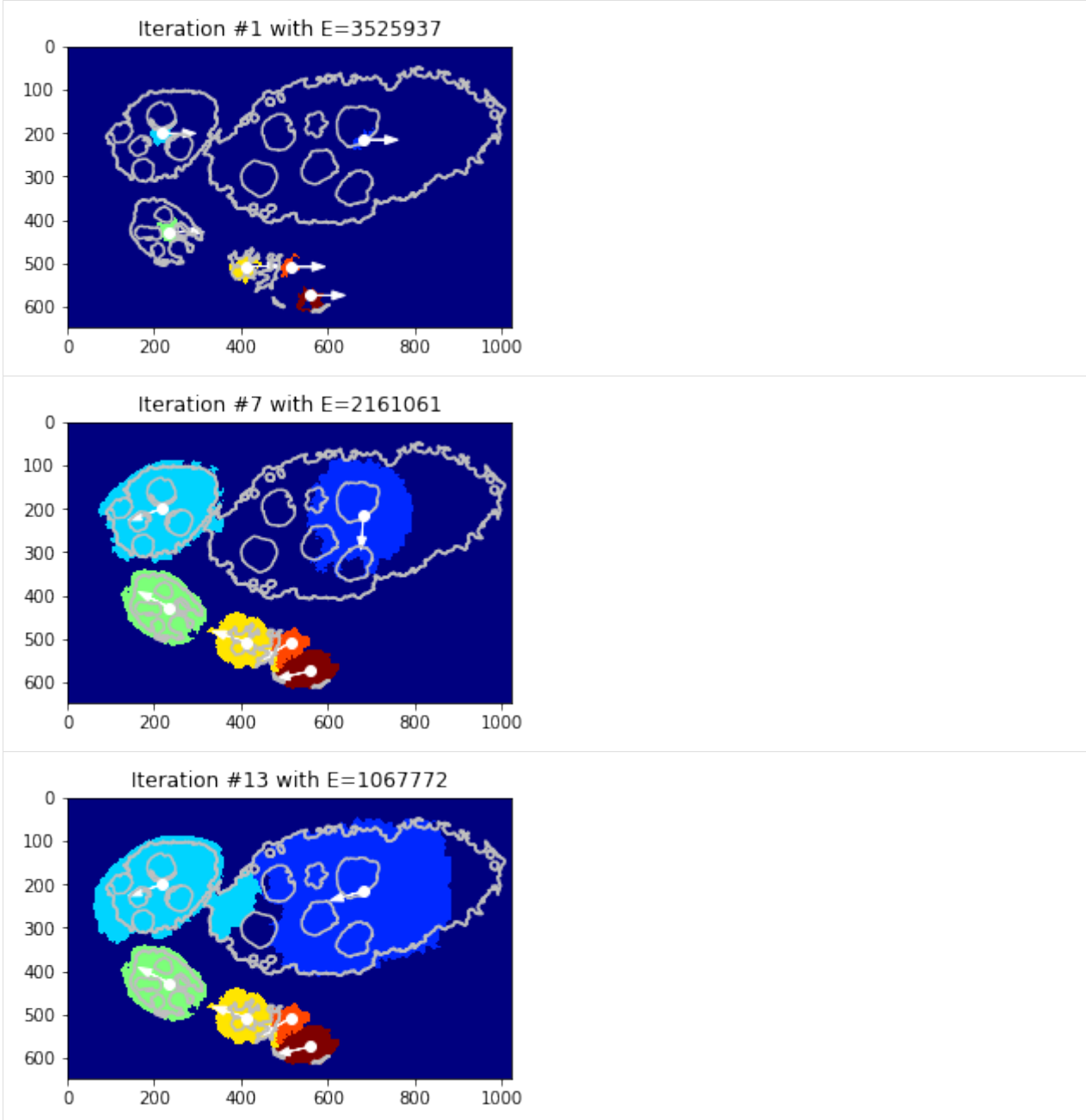
```
[32]: def show_partial(i):
      _ = plt.close(), tl_visu.figure_rg2sp_debug_complete(seg, slic, debug_gc_mm, i,
      ↪max_size=4)
      # show the interact
      widgets.interact(show_partial, i=(0, len(debug_gc_mm['criteria']) -1))
```

```
[32]: <function __main__.show_partial>
```

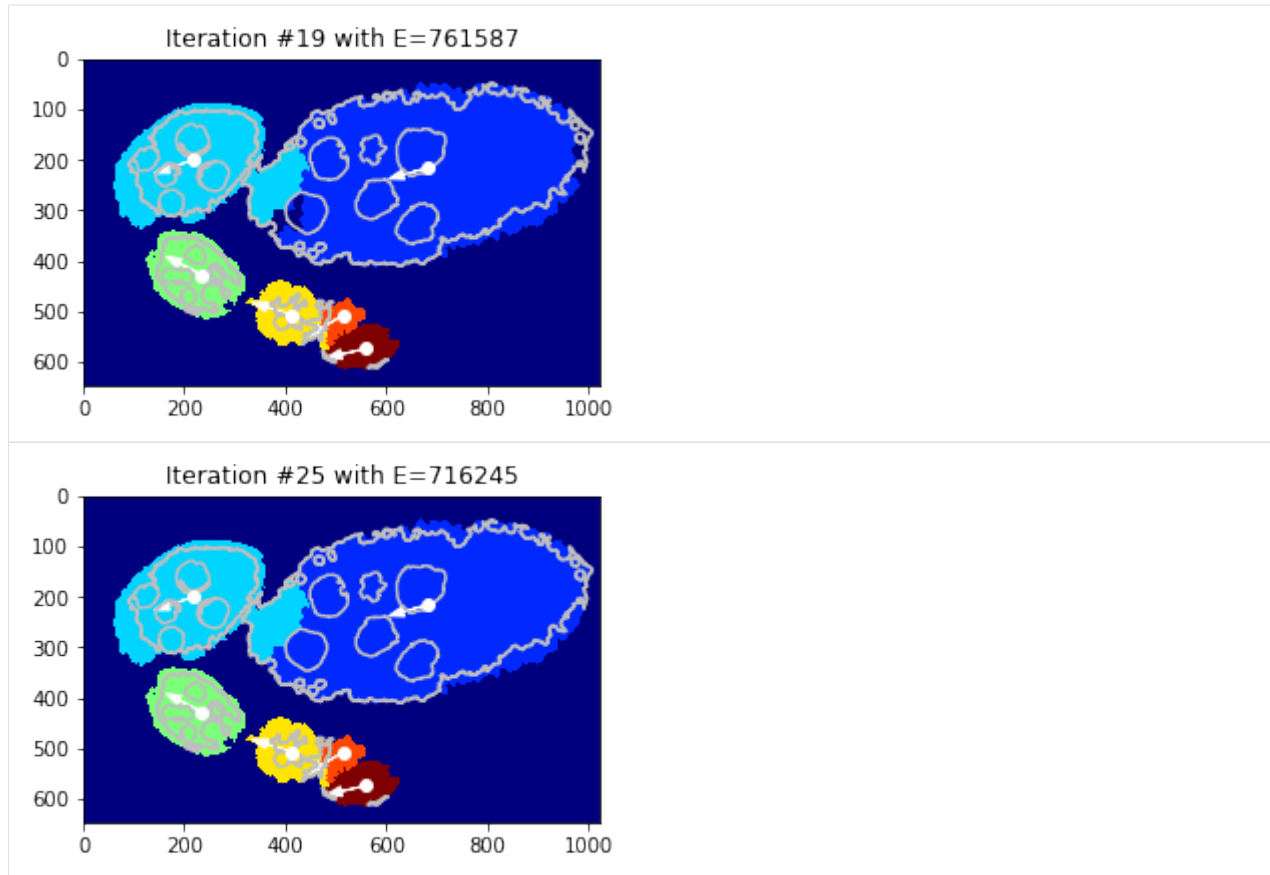


## Visualise some iterations

```
[34]: nb_iter = len(debug_gc_mm['criteria'])
      for i in np.linspace(1, nb_iter - 1, 5):
          _ = tl_visu.draw_rg2sp_results(plt.figure(figsize=(6, 3)).gca(), seg, slic, debug_
          ↪gc_mm, int(i))
```







## Exporting iterations

```
[44]: nb_iter = len(debug_gc_mm['criteria'])
path_out = tl_io.find_path_bubble_up('output')
fig_size = np.array(FIG_SIZE) * np.array([debug_gc_mm['lut_data_cost'].shape[1] - 1,
→ 2]) / 2.
for i in range(nb_iter):
    fig = plt.figure(figsize=fig_size)
    tl_visu.draw_rg2sp_results(fig.gca(), seg, slic, debug_gc_mm, int(i))
    plt.savefig(os.path.join(path_out, 'debug-gc-mm_iter-%03d' % i))
    plt.close(fig)
```

## Compare GC and Greedy

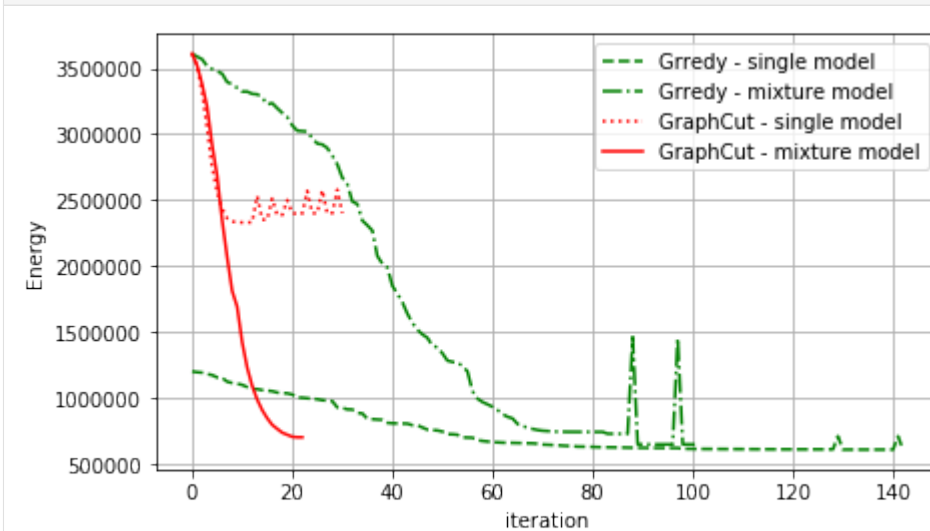
Comparing the energy evolving for all strategies

```
[31]: fig = plt.figure(figsize=(7, 4))
plt.plot(debug_gd_lm['criteria'], 'g--', label='Grredy - single model')
plt.plot(debug_gd_mm['criteria'], 'g-.', label='Grredy - mixture model')
plt.plot(debug_gc_lm['criteria'], 'r:', label='GraphCut - single model')
plt.plot(debug_gc_mm['criteria'], 'r-', label='GraphCut - mixture model')
_ = plt.ylabel('Energy'), plt.xlabel('iteration'), plt.grid(), plt.legend(loc='upper_
→ right')
```

(continues on next page)

(continued from previous page)

```
# fig.subplots_adjust(left=0.15, bottom=0.12, right=0.99, top=0.9)
# fig.savefig(os.path.join(PATH_OUT, 'figure_RG-energy-iter.pdf'))
```



```
[ ]:
```

### 1.3.6 Egg segm. from centre with GraphCut

A simple object segmentation method using Graph Cut over whole image initialised from a few initial seeds.

```
[1]: %matplotlib inline
import os, sys, glob
import numpy as np
import pandas as pd
from PIL import Image
# from scipy import spatial, ndimage
from skimage import segmentation as sk_seg
import matplotlib.pyplot as plt
```

```
[3]: sys.path += [os.path.abspath('.'), os.path.abspath('../')] # Add path to root
import imsegm.utilities.data_io as tl_io
import imsegm.superpixels as tl_spx
import imsegm.region_growing as tl_rg
```

#### Loading data

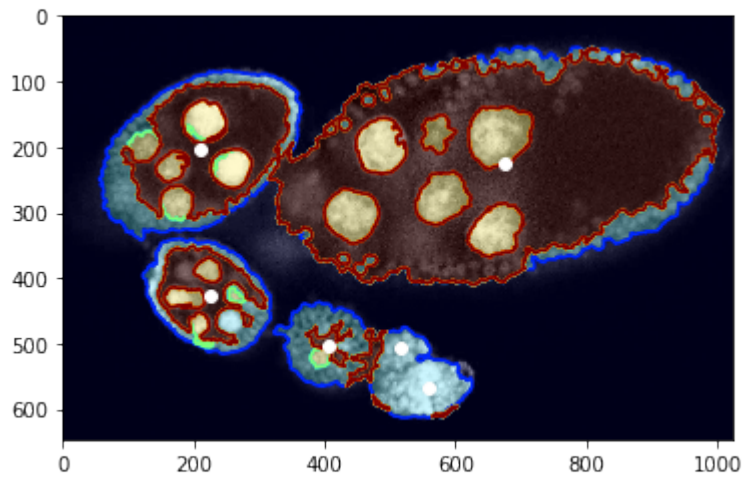
```
[4]: COLORS = 'bgrmyck'
PATH_IMAGES = tl_io.update_path(os.path.join('data-images', 'drosophila_ovary_slice'))
print ([os.path.basename(p) for p in glob.glob(os.path.join(PATH_IMAGES, '*')) if os.
↳ path.isdir(p)])
dir_img = os.path.join(PATH_IMAGES, 'image')
dir_seg = os.path.join(PATH_IMAGES, 'segm')
dir_annot = os.path.join(PATH_IMAGES, 'annot_eggs')
dir_center = os.path.join(PATH_IMAGES, 'center_levels')
```

```
['center_levels', 'image', 'annot_struct', 'annot_eggs', 'segm_rgb', 'segm']
```

```
[5]: name = 'insitu7545'
img = np.array(Image.open(os.path.join(dir_img, name + '.jpg')))
seg = np.array(Image.open(os.path.join(dir_seg, name + '.png')))
centers = pd.read_csv(os.path.join(dir_center, name + '.csv'), index_col=0).values
centers[:, [0, 1]] = centers[:, [1, 0]]
FIG_SIZE = (8. * np.array(img.shape[:2]) / np.max(img.shape))[:-1]

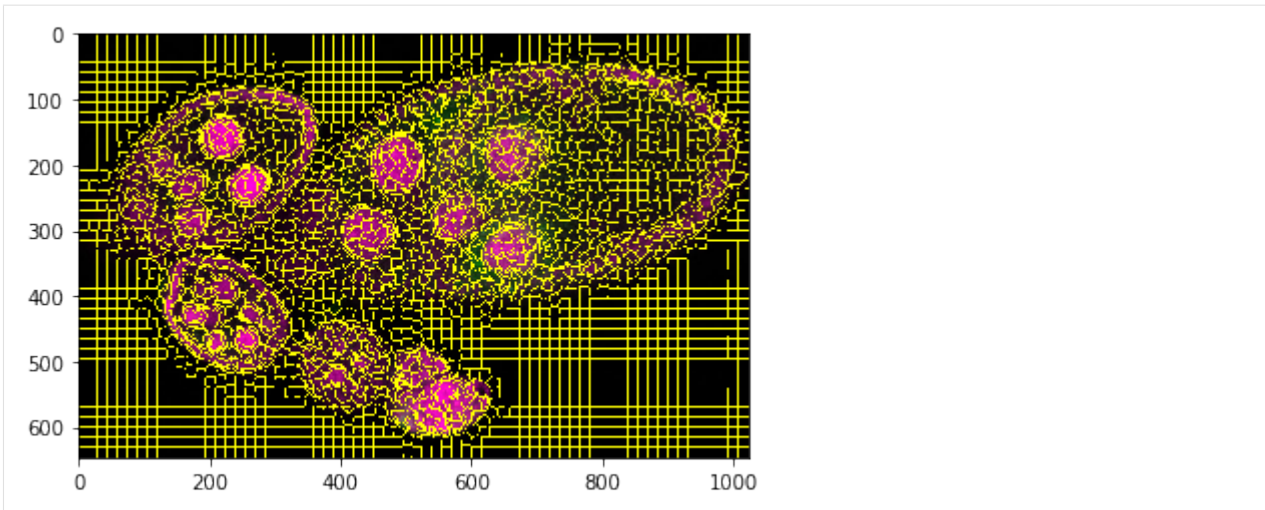
/usr/local/lib/python3.5/dist-packages/IPython/kernel/__main__.py:4: FutureWarning:
↳from_csv is deprecated. Please use read_csv(...) instead. Note that some of the
↳default arguments are different, so please refer to the documentation for from_csv
↳when changing your function calls
```

```
[6]: # plt.figure(figsize=FIG_SIZE)
plt.imshow(img[:, :, 0], cmap=plt.cm.Greys_r)
plt.imshow(seg, alpha=0.2, cmap=plt.cm.jet), plt.contour(seg, cmap=plt.cm.jet)
_= plt.plot(centers[:, 1], centers[:, 0], 'ow')
```



## Superpixels

```
[7]: slic = tl_spx.segment_slic_img2d(img, sp_size=15, relative_compact=0.3)
_= plt.imshow(sk_segmark_boundaries(img, slic))
```

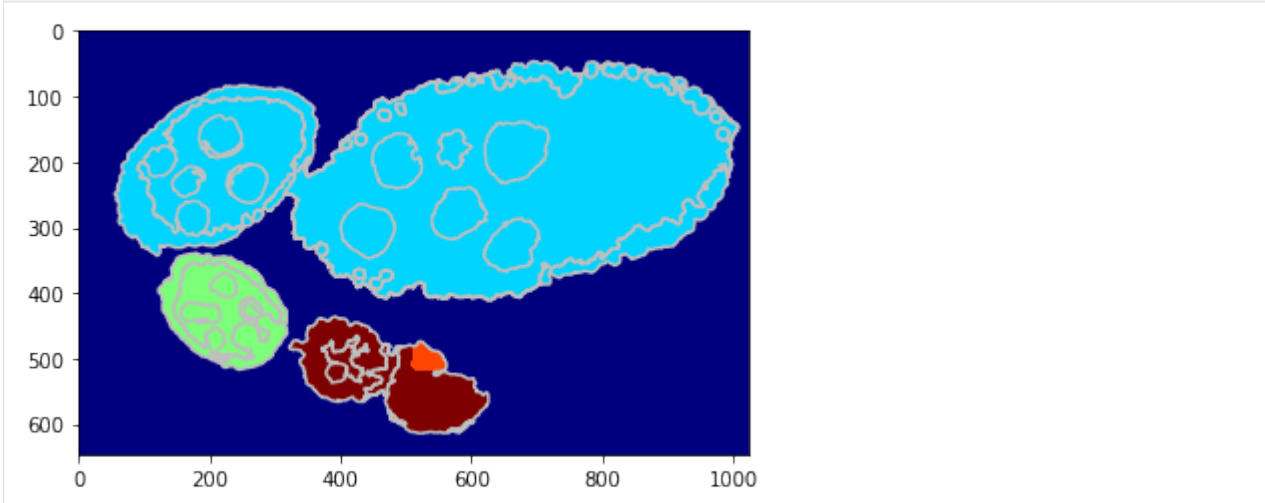


### Object segmentation on pixel level

Estimating eggs with just single global GraphCut on probability according structure segmentation and annotated centre points.

```
[8]: labels_fg_prob = [0.05, 0.7, 0.9, 0.9]
```

```
[10]: debug_visual = dict()
      segm_obj = tl_rg.object_segmentation_graphcut_pixels(seg, centers, labels_fg_prob, gc_
      ↪ regul=1.,
      seed_size=10, debug_visual=debug_
      ↪ visual)
      #fig = plt.figure(figsize=FIG_SIZE)
      plt.imshow(segm_obj, cmap=plt.cm.jet)
      _ = plt.contour(seg, levels=np.unique(seg), colors='#bfbfbf')
```



Visualise the unary potential as maxima over join matrix U.

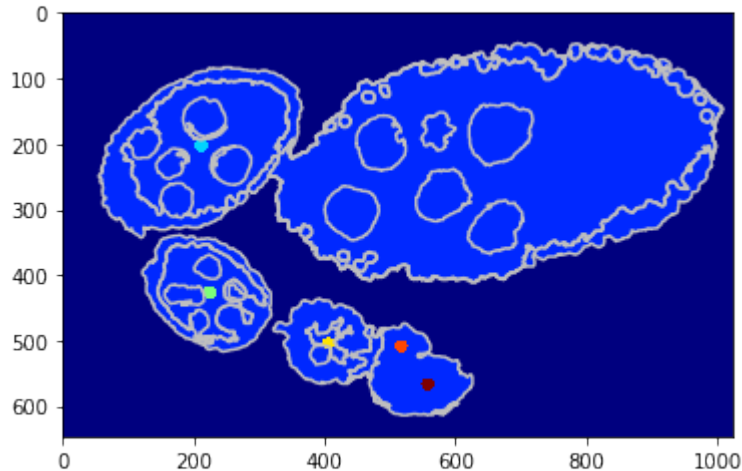
```
[11]: unary = np.array([u.tolist() for u in debug_visual['unary_imgs']])
      print ('shape: %s and unique labels %s' % (repr(unary.shape), repr(np.unique(np.
      ↪ argmin(unary, axis=0)))))
```

(continues on next page)

(continued from previous page)

```
#fig = plt.figure(figsize=FIG_SIZE)
plt.imshow(np.argmax(unary, axis=0), cmap=plt.cm.jet)
_ = plt.contour(seg, levels=np.unique(seg), colors='#bfbfbf')
```

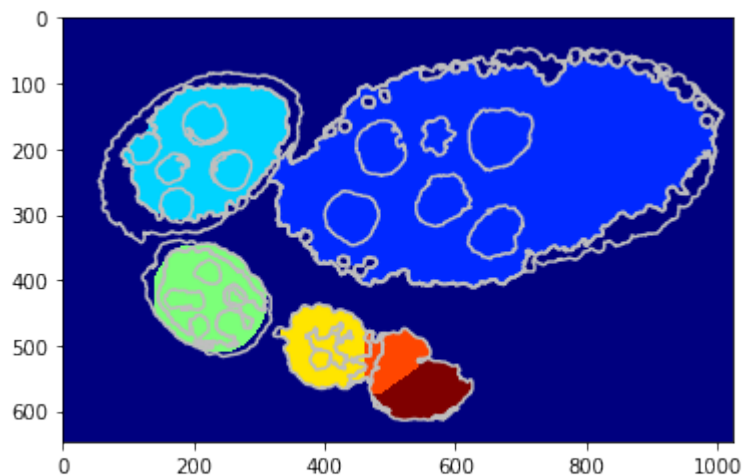
```
(7, 647, 1024) [0 1 2 3 4 5 6]
```



### Using a shape prior

Similar scenario as before but in this case we draw a circular shape prior around each center which helps better egg identification.

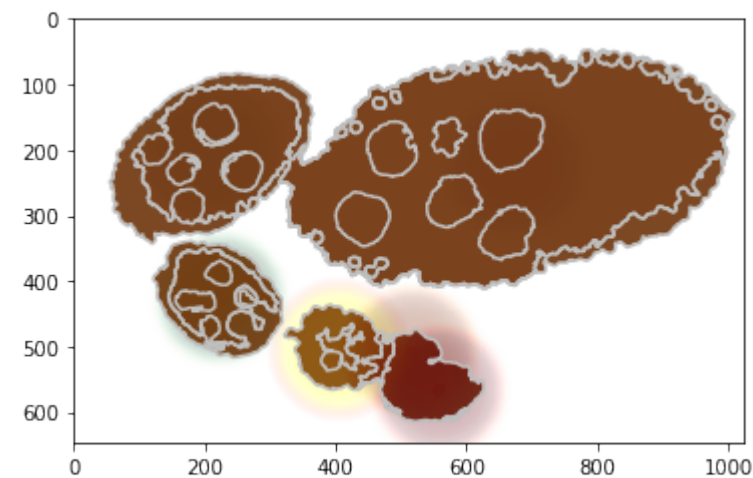
```
[12]: debug_visual = dict()
segm_obj = tl_rg.object_segmentation_graphcut_pixels(seg, centers, labels_fg_prob, gc_
    regul=2., seed_size=10,
    coef_shape=0.1, shape_mean_
    std=(50., 10.), debug_visual=debug_visual)
#fig = plt.figure(figsize=FIG_SIZE)
plt.imshow(segm_obj, cmap=plt.cm.jet)
_ = plt.contour(seg, levels=np.unique(seg), colors='#bfbfbf')
```



Visualise the unary potential as maxima over join matrix U.

```
[13]: unary = np.array([u.tolist() for u in debug_visual['unary_imgs']])
print ('shape: %s and unique labels %s' % (repr(unary.shape), repr(np.unique(np.
    ↪argmin(unary, axis=0)))))
#fig = plt.figure(figsize=FIG_SIZE)
plt.contour(seg, levels=np.unique(seg), colors='#bfbfbf')
CMAPS = [plt.cm.Greys, plt.cm.Purples, plt.cm.Blues, plt.cm.Greens, plt.cm.autumn,
    ↪plt.cm.Oranges, plt.cm.Red]
for i in range(1, len(unary)):
    im = 1 - (unary[i, :, :] / np.max(unary[i, :, :]))
    im = (im * 1000).astype(int)
    lut = CMAPS[i](range(1001))
    lut[:, 3] = np.linspace(0, 0.7, 1001)
    plt.imshow(lut[im], alpha=1)
```

```
(7, 647, 1024) [0 1 2 3 4 5 6]
```

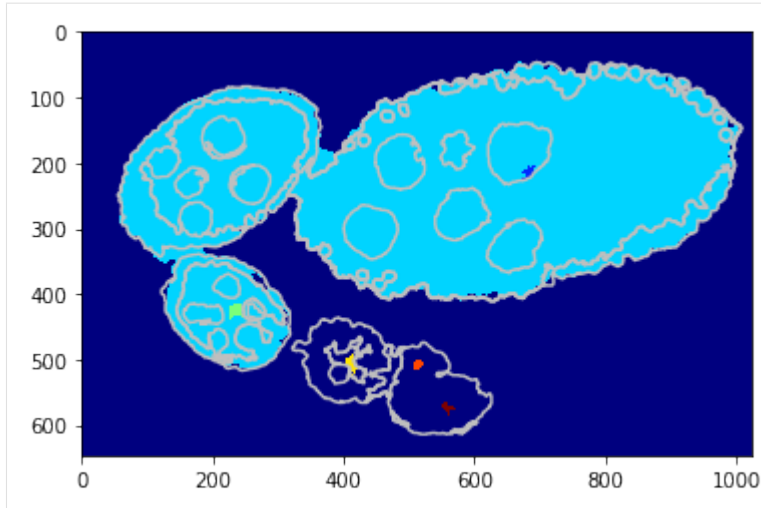


### Object segmentation on SLIC level

We replicate the previous methods but apply on superpixels - flat unary potential.

```
[15]: debug_visual = dict()
gc_labels = tl_rg.object_segmentation_graphcut_slic(slic, seg, centers, labels_fg_
    ↪prob, gc_regul=2., edge_coef=1.,
    edge_type='ones', add_
    ↪neighbours=True, debug_visual=debug_visual)
segm_obj = np.array(gc_labels)[slic]

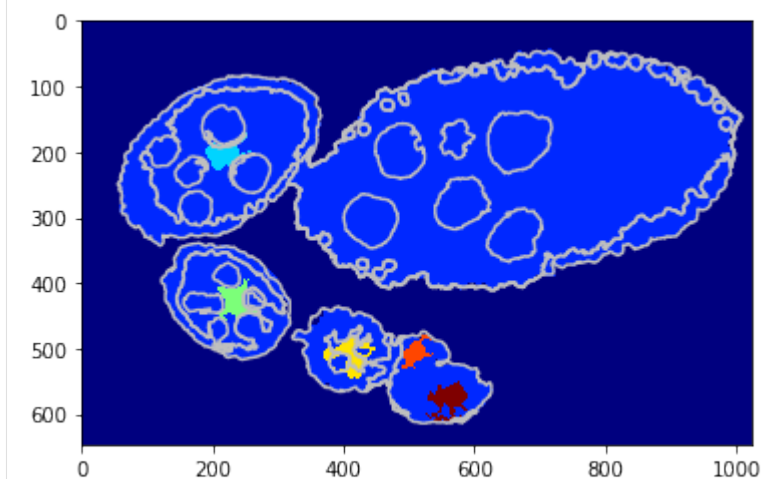
#fig = plt.figure(figsize=FIG_SIZE)
plt.imshow(segm_obj, cmap=plt.cm.jet)
_ = plt.contour(seg, levels=np.unique(seg), colors='#bfbfbf')
```



Visualise the unary potential as maxima over join matrix U.

```
[16]: unary = np.array([u.tolist() for u in debug_visual['unary_imgs']])
print ('shape: %s and unique labels %s' % (repr(unary.shape), repr(np.unique(np.
    ↪argmin(unary, axis=0))))))
#fig = plt.figure(figsize=FIG_SIZE)
plt.imshow(np.argmax(unary, axis=0), cmap=plt.cm.jet)
_ = plt.contour(seg, levels=np.unique(seg), colors='#bfbfbf')
```

```
(7, 647, 1024) [0 1 2 3 4 5 6]
```



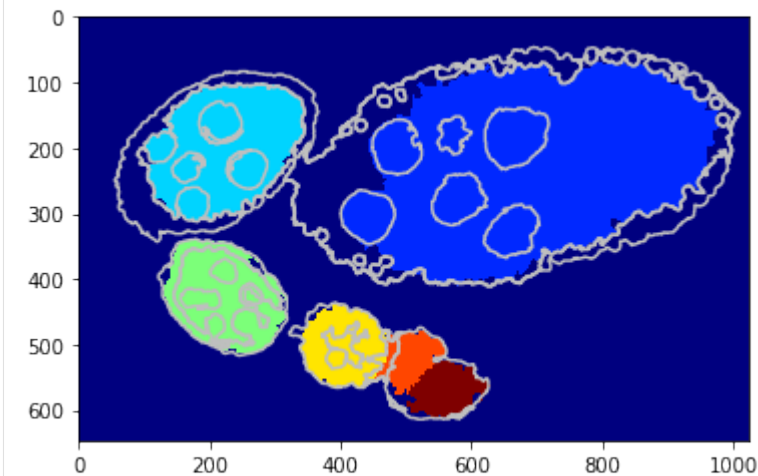
### Using a shape prior

We replicate the previous methods but apply on superpixels - unary potential with circular shape prior.

```
[17]: debug_visual = dict()
gc_labels = tl_rg.object_segmentation_graphcut_slic(slic, seg, centers, labels_fg_
    ↪prob, gc_regul=1., edge_coef=1.,
    edge_type='ones', coef_shape=0.1,
    ↪shape_mean_std=(50., 10.),
    add_neighbours=False, debug_
    ↪visual=debug_visual)
(continues on next page)
```

(continued from previous page)

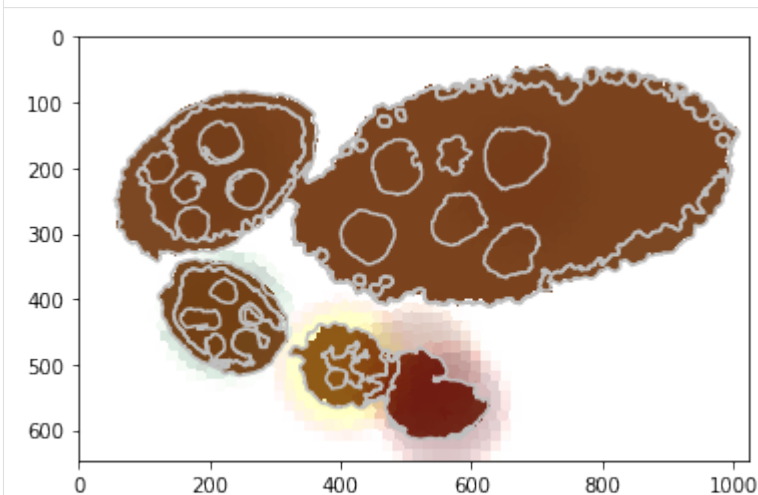
```
segm_obj = np.array(gc_labels)[slic]
#fig = plt.figure(figsize=FIG_SIZE)
plt.imshow(segm_obj, cmap=plt.cm.jet)
_ = plt.contour(seg, levels=np.unique(seg), colors='#bfbfbf')
```



Visualise the unary potential as maxima over join matrix U.

```
[18]: unary = np.array([u.tolist() for u in debug_visual['unary_imgs']])
print ('shape: %s and unique labels %s' % (repr(unary.shape), repr(np.unique(np.
    ↪argmin(unary, axis=0)))))
#fig = plt.figure(figsize=FIG_SIZE)
plt.contour(seg, levels=np.unique(seg), colors='#bfbfbf')
CMAPS = [plt.cm.Greys, plt.cm.Purples, plt.cm.Blues, plt.cm.Greens, plt.cm.autumn,
    ↪plt.cm.Oranges, plt.cm.Red]
for i in range(1, len(unary)):
    im = 1 - (unary[i, :, :] / np.max(unary[i, :, :]))
    im = (im * 1000).astype(int)
    lut = CMAPS[i](range(1001))
    lut[:, 3] = np.linspace(0, 0.7, 1001)
    plt.imshow(lut[im], alpha=1)
```

```
(7, 647, 1024) [0 1 2 3 4 5 6]
```





[ ]:

### 1.3.7 Centre candidates and clustering

An image processing pipeline to detect and localize *Drosophila* egg chambers that consists of the following steps: (i) superpixel-based image imsegm into relevant tissue classes (see above); (ii) detection of egg center candidates using label histograms and ray features; (iii) clustering of center candidates. Prepare zones for training center candidates and perform desity clustering.

Borovec, J., Kybic, J., & Nava, R. (2017). **Detection and Localization of *Drosophila* Egg Chambers in Microscopy Images**. In Q. Wang, Y. Shi, H.-I. Suk, & K. Suzuki (Eds.), *Machine Learning in Medical Imaging*, (pp. 19–26).

```
[1]: %matplotlib inline
import os, sys
import numpy as np
from PIL import Image
from sklearn import cluster
import matplotlib.pyplot as plt
```

```
[2]: sys.path += [os.path.abspath('.'), os.path.abspath('../')] # Add path to root
import imsegm.utilities.data_io as tl_io
import imsegm.ellipse_fitting as seg_fit
import imsegm.descriptors as seg_fts
import imsegm.classification as seg_clf
```

```
WARNING:root:descriptors: using pure python libraries
/usr/local/lib/python3.5/dist-packages/sklearn/cross_validation.py:41:
↳ DeprecationWarning: This module was deprecated in version 0.18 in favor of the
↳ model_selection module into which all the refactored classes and functions are
↳ moved. Also note that the interface of the new CV iterators are different from that
↳ of this module. This module will be removed in 0.20.
    "This module will be removed in 0.20.", DeprecationWarning)
/usr/local/lib/python3.5/dist-packages/sklearn/grid_search.py:42: DeprecationWarning:
↳ This module was deprecated in version 0.18 in favor of the model_selection module
↳ into which all the refactored classes and functions are moved. This module will be
↳ removed in 0.20.
    DeprecationWarning)
```

#### Load images

```
[3]: name = 'insitu7545'
PATH_BASE = tl_io.update_path(os.path.join('data-images', 'drosophila_ovary_slice'))
PATH_IMAGES = os.path.join(PATH_BASE, 'image')
PATH_SEGM = os.path.join(PATH_BASE, 'segm')
PATH_ANNOT = os.path.join(PATH_BASE, 'annot_eggs')
PATH_CENTRE = os.path.join(PATH_BASE, 'center_levels')
COLORS = 'bgrcymk'
```

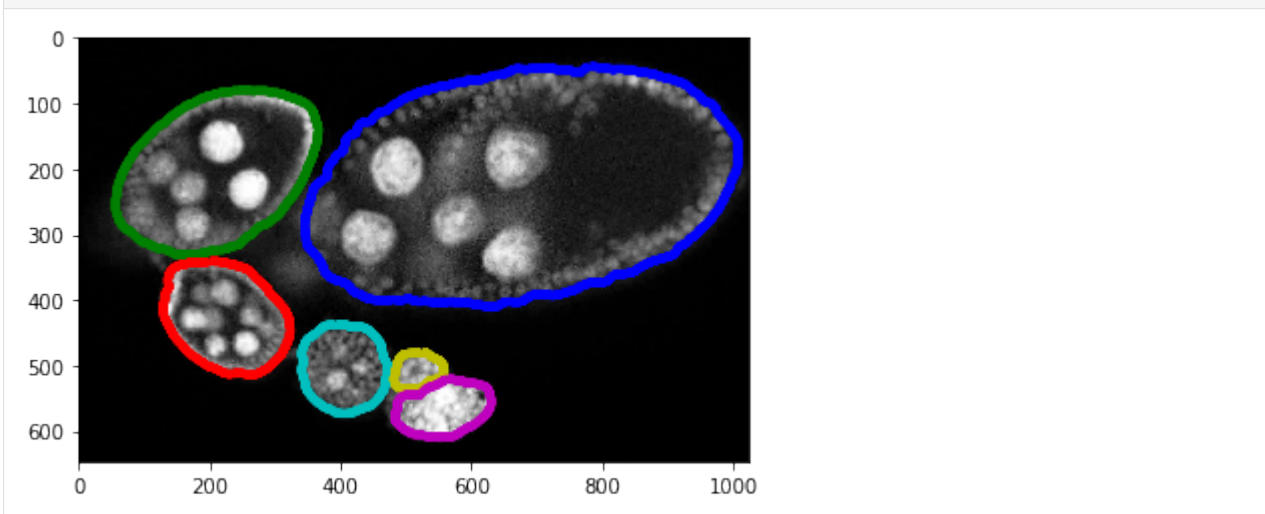
```
[4]: img = np.array(Image.open(os.path.join(PATH_IMAGES, name + '.jpg')))
segm = np.array(Image.open(os.path.join(PATH_SEGM, name + '.png')))
annot = np.array(Image.open(os.path.join(PATH_ANNOT, name + '.png')))
levels = np.array(Image.open(os.path.join(PATH_CENTRE, name + '.png')))
FIG_SIZE = (10. * np.array(img.shape[:2]) / np.max(img.shape))[:-1]
```

## Show training for center detection

In this part we discuss the procedure of preparing training data...

### Individual egg annotation

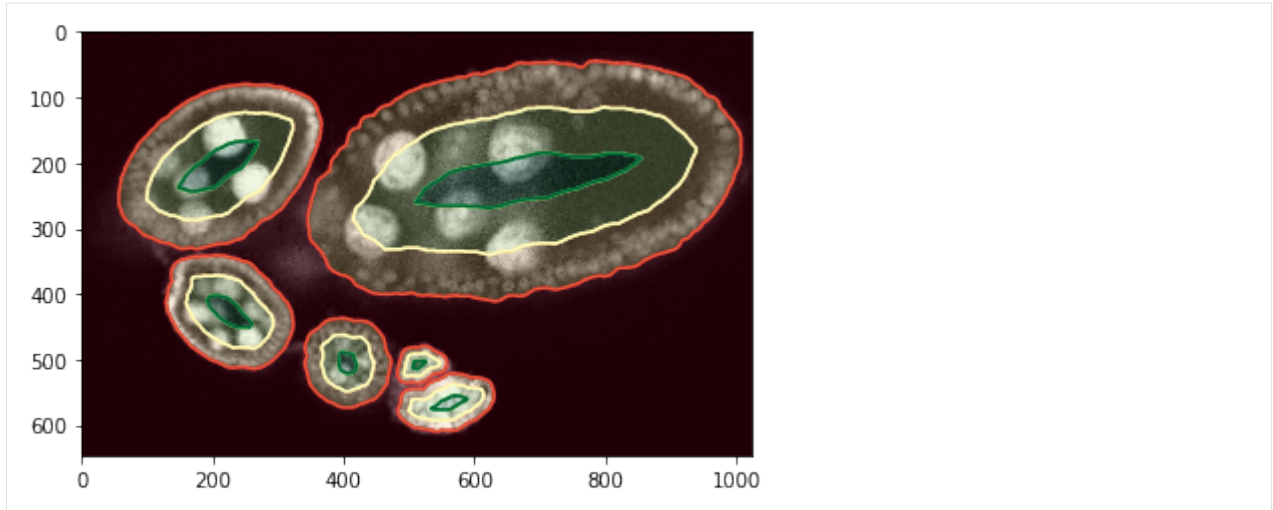
```
[5]: #plt.figure(figsize=FIG_SIZE)
plt.imshow(img[:, :, 0], cmap=plt.cm.Greys_r)
# _= plt.contour(annot, levels=np.unique(annot), linewidths=(4,))
for i in range(1, np.max(annot) + 1):
    _= plt.contour(annot == i, colors=COLORS[(i-1) % len(COLORS)], linewidths=(4,))
```



### Zones inside eggs

We have generated zones inside each object/egg such as we divide into 3 centric shapes. In fact this annotation can be automatically created by our script and then manually adjusted.

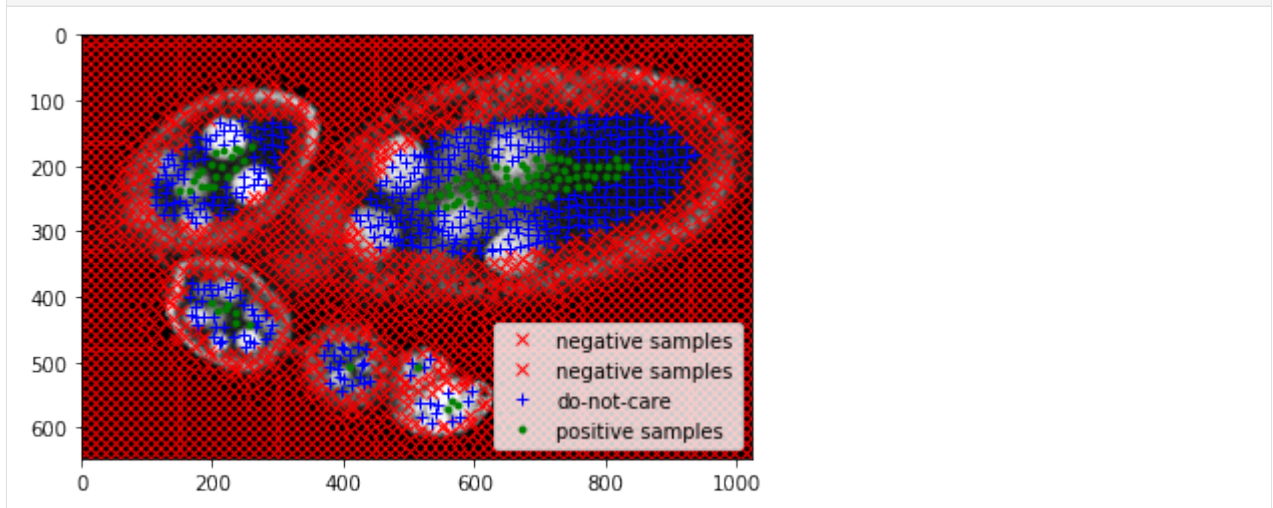
```
[6]: #plt.figure(figsize=FIG_SIZE)
plt.imshow(img[:, :, 0], cmap=plt.cm.Greys_r)
_= plt.imshow(levels, alpha=0.2, cmap=plt.cm.RdYlGn), plt.contour(levels, cmap=plt.cm.
↪RdYlGn)
```



### Training examples, points

We assume that the center zone is positive (green), background and near boundary is negative (red) and the zone in between we ignore (blue).

```
[8]: slic, points, labels = seg_fit.get_slic_points_labels(levels, img, slic_size=15, slic_
    ↪ regul=0.3)
    #plt.figure(figsize=FIG_SIZE)
    plt.imshow(img[:, :, 0], cmap=plt.cm.Greys_r)
    plt.plot(points[labels == 0, 1], points[labels == 0, 0], 'xr', label='negative samples
    ↪')
    plt.plot(points[labels == 1, 1], points[labels == 1, 0], 'xr', label='negative samples
    ↪')
    plt.plot(points[labels == 2, 1], points[labels == 2, 0], '+b', label='do-not-care')
    plt.plot(points[labels == 3, 1], points[labels == 3, 0], '.g', label='positive samples
    ↪')
    _ = plt.xlim([0, img.shape[1]]), plt.ylim([img.shape[0], 0]), plt.legend(loc='lower_
    ↪right')
```



Feature extraction for each sample point.

```
[9]: features_hist, names_hist = seg_fts.compute_label_histograms_positions(seg, points,
↳ diameters=[25, 50, 100, 150, 200, 300])
features_ray, _, names_ray = seg_fts.compute_ray_features_positions(seg, points,
↳ angle_step=15, edge='up', border_labels=[0], smooth_ray=0)
features = np.hstack((features_hist, features_ray))
```

Adjustment the annotation as we decribed before and train a classifier...

```
[10]: clf_pipeline = seg_clf.create_clf_pipeline()
labels[labels == 1] = 0
labels[labels == 3] = 2
clf_pipeline.fit(features, labels)

[10]: Pipeline(memory=None,
      steps=[('scaler', StandardScaler(copy=True, with_mean=True, with_std=True)), (
↳ 'reduce_dim', PCA(copy=True, iterated_power='auto', n_components=0.95, random_
↳ state=None,
      svd_solver='auto', tol=0.0, whiten=False)), ('classif',
↳ RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini'..._jobs=-
↳ 1,
      oob_score=False, random_state=None, verbose=0,
      warm_start=False))])
```

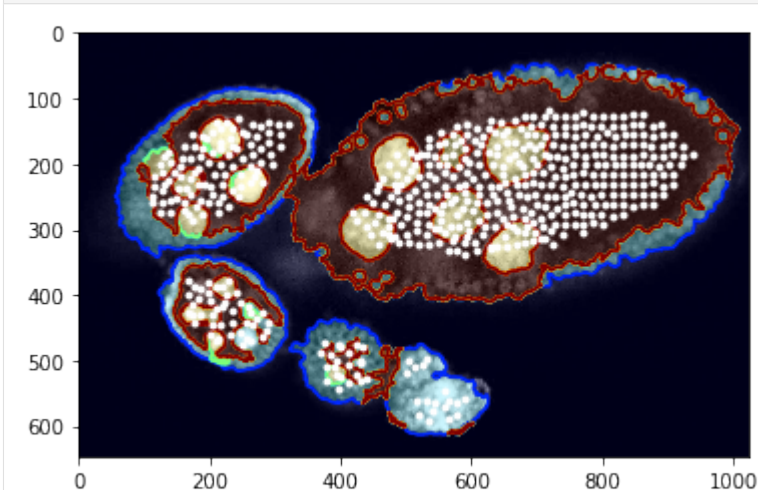
Predict lables for these sample points, note that overfitting may appeare.

```
[11]: labels = clf_pipeline.predict(features)
candidates = points[labels == 2, :]
```

## Center clustering

Visualisation of predicted positive sample points...

```
[12]: # candidates = points[labels == 3, :]
plt.figure(figsize=FIG_SIZE)
plt.imshow(img[:, :, 0], cmap=plt.cm.Greys_r)
plt.imshow(seg, alpha=0.2, cmap=plt.cm.jet), plt.contour(seg, cmap=plt.cm.jet)
_ = plt.plot(candidates[:, 1], candidates[:, 0], '.w')
```



As you can see that potentila points inside egg are more less grouped together. We apply decity clustering which alloes us to identify individual eggs.

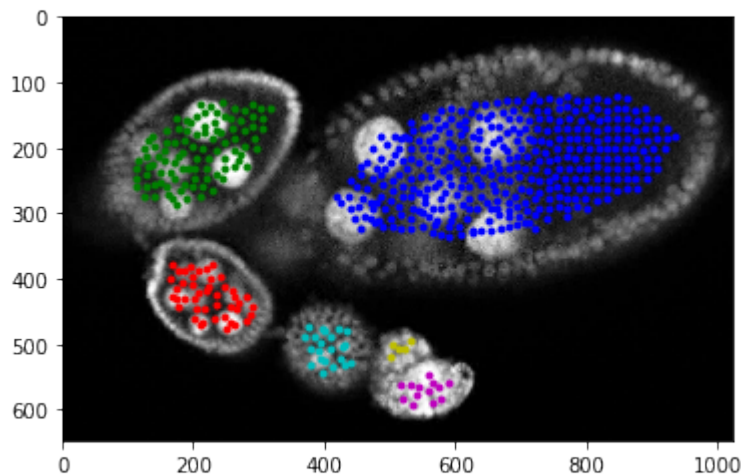
```
[17]: dbscan = cluster.DBSCAN(eps=40, min_samples=1)
      dbscan.fit(candidates)

[17]: DBSCAN(algorithm='auto', eps=40, leaf_size=30, metric='euclidean',
          metric_params=None, min_samples=1, n_jobs=1, p=None)

[18]: #fig = plt.figure(figsize=FIG_SIZE)
      plt.imshow(img[:, :, 0], cmap=plt.cm.Greys_r)
      # plt.imshow(seg, alpha=0.2), plt.contour(seg)
      centers = []

      for i in range(max(dbscan.labels_) + 1):
          select = candidates[dbscan.labels_ == i]
          centers.append(np.mean(select, axis=0))
          plt.plot(select[:, 1], select[:, 0], '.', color=COLORS[i % len(COLORS)])
      centers = np.array(centers)

      for i in range(len(centers)):
          fig.gca().scatter(centers[i, 1], centers[i, 0], s=300, c=COLORS[i % len(COLORS)])
          # plt.plot(centers[i, 1], centers[i, 0], 'o', color=COLORS[i % len(COLORS)])
      _, _ = plt.xlim([0, img.shape[1]]), plt.ylim([img.shape[0], 0])
```



```
[ ]:
```

### 1.3.8 Ellipse fitting from center

Having a center per egg and structural segmentation we want to approximate the egg by an ellipse such way that it maximises the expectation being single egg. Some ellipse fitting references: [\\* Fitting an Ellipse to a Set of Data Points](#) [\\* Numerically Stable Direct Least Squares Fitting Of Ellipses](#) [\\* Non-linear fitting to an ellipse](#)

Borovec, J., Kybic, J., & Nava, R. (2017). **Detection and Localization of Drosophila Egg Chambers in Microscopy Images**. In Q. Wang, Y. Shi, H.-I. Suk, & K. Suzuki (Eds.), *Machine Learning in Medical Imaging*, (pp. 19–26).

```
[1]: %matplotlib inline
      import os, sys
      import numpy as np
      import pandas as pd
```

(continues on next page)

(continued from previous page)

```
from PIL import Image
import matplotlib.pyplot as plt
```

```
[2]: sys.path += [os.path.abspath('.'), os.path.abspath('../')] # Add path to root
import imsegm.utilities.data_io as tl_io
import imsegm.utilities.drawing as tl_visu
import imsegm.ellipse_fitting as tl_fit
```

```
WARNING:root:descriptors: using pure python libraries
```

## Loading data

```
[3]: PATH_BASE = tl_io.update_path(os.path.join('data-images', 'drosophila_ovary_slice'))
PATH_IMAGES = os.path.join(PATH_BASE, 'image')
PATH_SEGM = os.path.join(PATH_BASE, 'segm')
PATH_ANNOT = os.path.join(PATH_BASE, 'annot_eggs')
PATH_CENTRE = os.path.join(PATH_BASE, 'center_levels')
COLORS = 'bgrmyck'
```

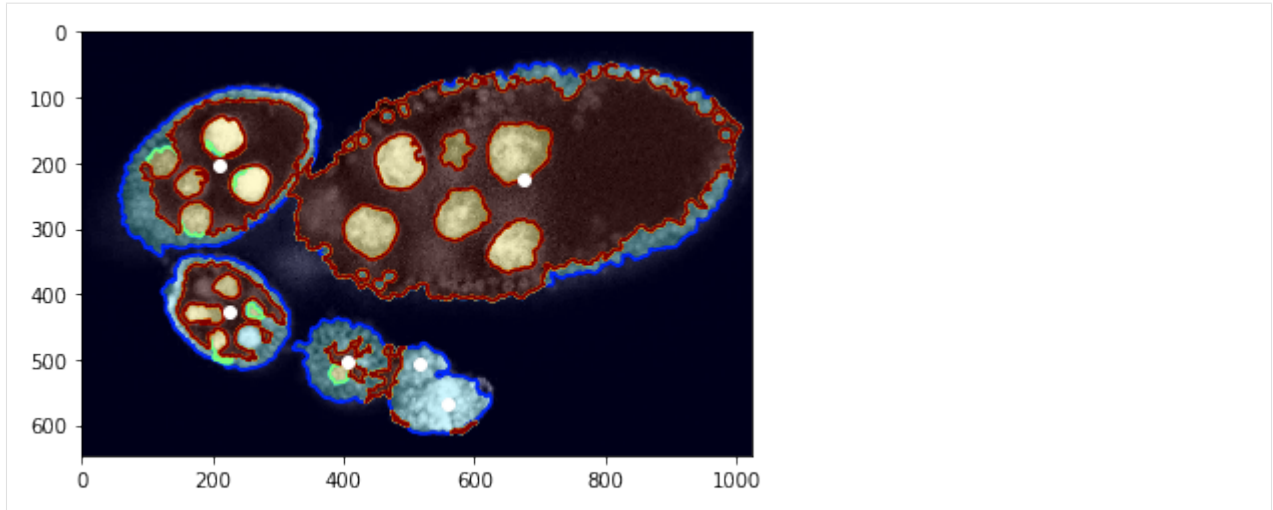
## Loading images...

```
[4]: name = 'insitu7545'
# name = 'insitu11151'
img = np.array(Image.open(os.path.join(PATH_IMAGES, name + '.jpg')))
seg = np.array(Image.open(os.path.join(PATH_SEGM, name + '.png')))
centers = pd.read_csv(os.path.join(PATH_CENTRE, name + '.csv'), index_col=0).values
centers = centers[:, [1, 0]]
FIG_SIZE = (8. * np.array(img.shape[:2]) / np.max(img.shape))[:, -1]

/usr/local/lib/python3.5/dist-packages/IPython/kernel/__main__.py:5: FutureWarning:
↳ from_csv is deprecated. Please use read_csv(...) instead. Note that some of the
↳ default arguments are different, so please refer to the documentation for from_csv
↳ when changing your function calls
```

Visualisation of structure segmentation overlped over input image and marked center points.

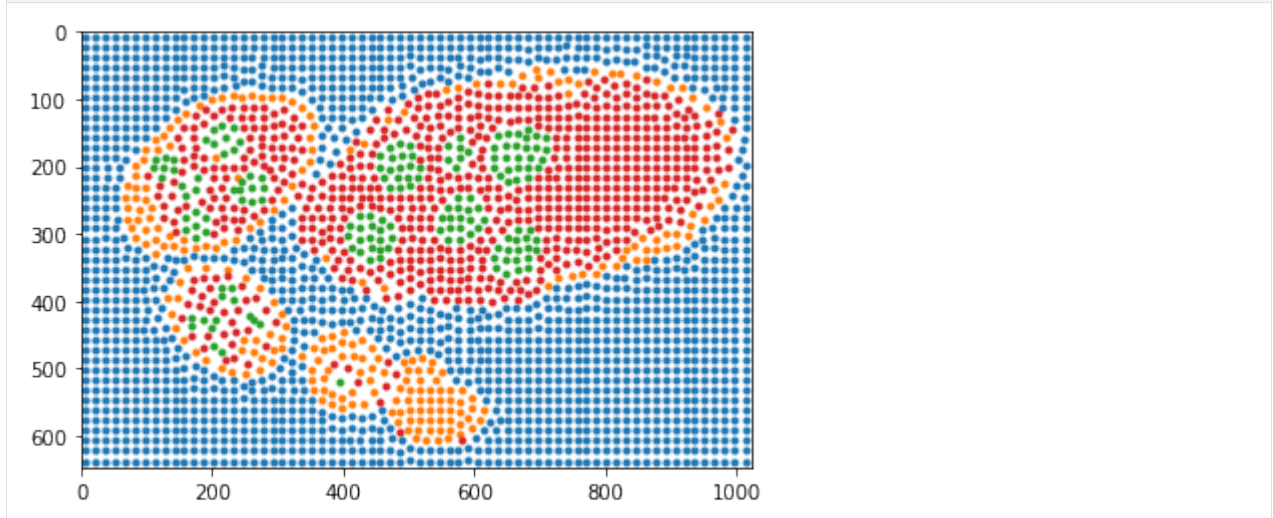
```
[5]: #fig = plt.figure(figsize=FIG_SIZE)
plt.imshow(img[:, :, 0], cmap=plt.cm.Greys_r)
plt.imshow(seg, alpha=0.2, cmap=plt.cm.jet), plt.contour(seg, cmap=plt.cm.jet)
_ = plt.plot(centers[:, 1], centers[:, 0], 'ow')
```



### Preprocess - estimate boundary points

Visualisation of labeled sample points according structure segmentation / annotations.

```
[6]: slic, points_all, labels = tl_fit.get_slic_points_labels(seg, slic_size=15, slic_
    ↪ regul=0.3)
# points_all, labels = egg_seg.get_slic_points_labels(seg, img, size=15, regul=0.25)
for lb in np.unique(labels):
    plt.plot(points_all[labels == lb, 1], points_all[labels == lb, 0], '.')
_ = plt.xlim([0, seg.shape[1]]), plt.ylim([seg.shape[0], 0])
weights = np.bincount(slic.ravel())
```



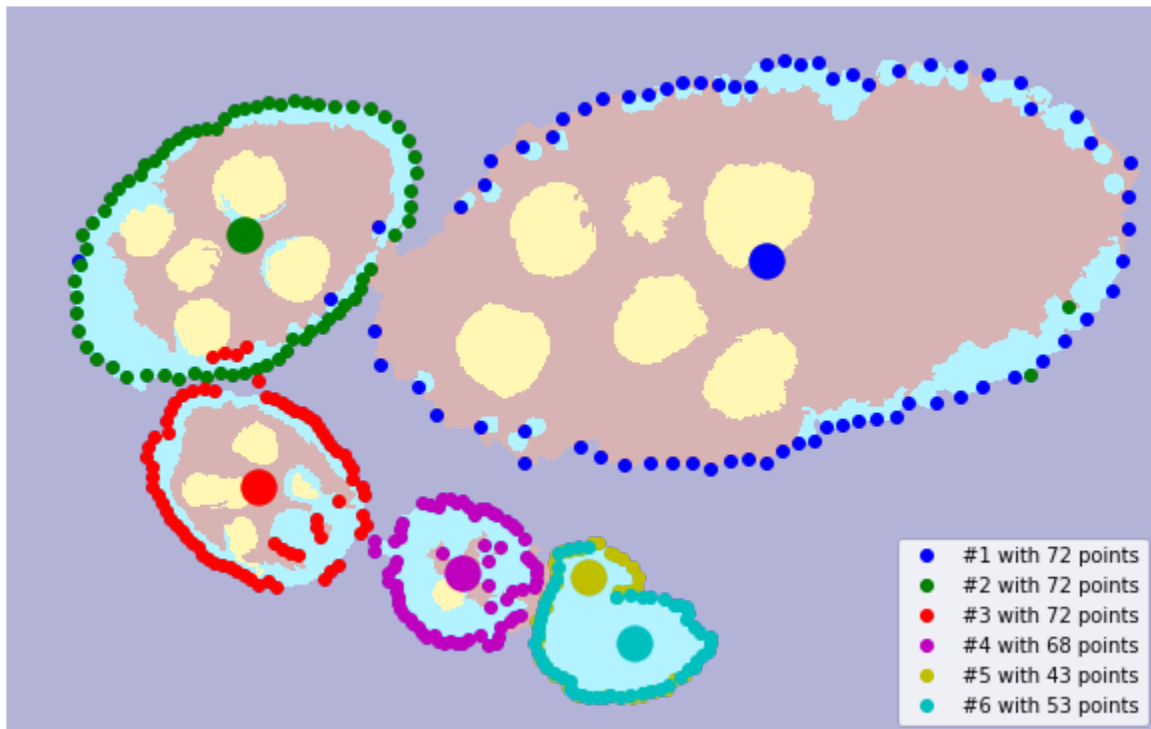
Reconstruction of boundary points using ray features from centers.

```
[7]: points_centers = tl_fit.prepare_boundary_points_ray_edge(seg, centers, close_points=5)
# points_centers = tl_fit.prepare_boundary_points_ray_mean(seg, centers, close_
    ↪ points=5)
# points_centers = tl_fit.prepare_boundary_points_dist(seg, centers)
```



```
[8]: fig = plt.figure(figsize=FIG_SIZE)
plt.imshow(seg, alpha=0.3, cmap=plt.cm.jet)
for i, points in enumerate(points_centers):
    plt.plot(points[:, 1], points[:, 0], 'o', color=COLORS[i % len(COLORS)], label='#
    ↪ %i with %i points' % ((i + 1), len(points)))
for i in range(len(centers)):
    # plt.plot(centers[i, 1], centers[i, 0], 'o', color=COLORS[i % len(COLORS)])
    plt.scatter(centers[i, 1], centers[i, 0], s=300, c=COLORS[i % len(COLORS)])
_ = plt.legend(loc='lower right', plt.axes().set_aspect('equal')
_ = plt.xlim([0, seg.shape[1]]), plt.ylim([seg.shape[0], 0])
plt.axis('off'), fig.subplots_adjust(left=0, right=1, top=1, bottom=0)
# fig.savefig('fig1.pdf')
```

```
[8]: ((0.0, 1024.0, 647.0, 0.0), None)
```



## Fit ellipse by RANSAC and hypothesis

Set the probability of being egg for each class from initial segmentation.

```
[9]: TABLE_FB_PROBA = [[0.01, 0.7, 0.95, 0.8],
                        [0.99, 0.3, 0.05, 0.2]]
print ('points: %i weights: %i labels: %i' % (len(labels), len(points_all),
    ↪ len(weights)))
```

```
points: 2652 weights: 2652 labels: 2652
```

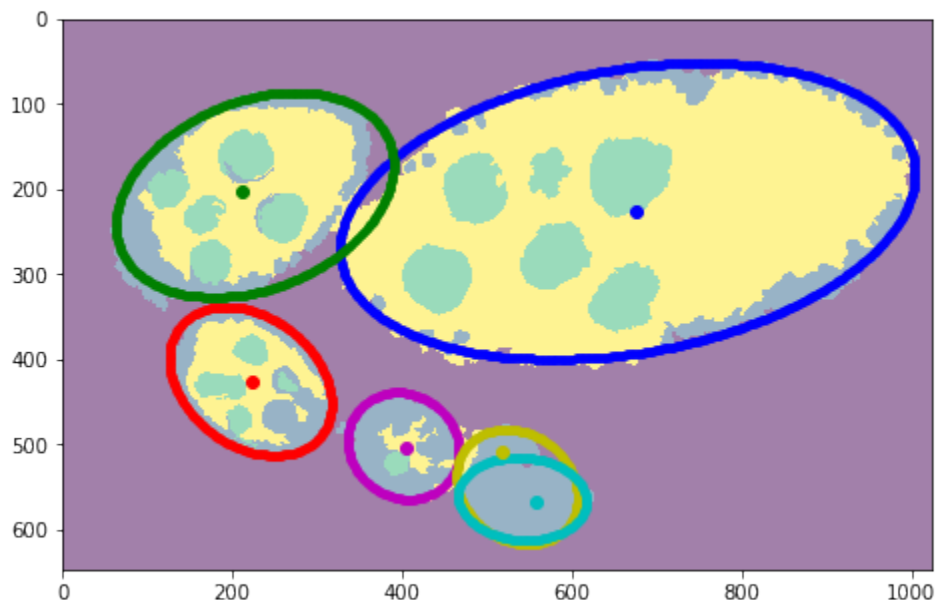
Fit the ellipse to maximise the hypotheses having single egg inside.



```
[10]: plt.figure(figsize=FIG_SIZE)
plt.imshow(seg, alpha=0.5)
ellipses, crits = [], []
for i, points in enumerate(points_centers):
    model, _ = tl_fit.ransac_seg(points, tl_fit.EllipseModelSegm, points_all,
    weights, labels,
    TABLE_FB_PROBA, min_samples=0.4, residual_
    threshold=10, max_trials=150)
    if model is None: continue
    c1, c2, h, w, phi = model.params
    ellipses.append(model.params)
    crit = model.criterion(points_all, weights, labels, TABLE_FB_PROBA)
    crits.append(np.round(crit))
    print('model params: %s' % repr(int(c1), int(c2), int(h), int(w), phi))
    print('-> crit: %f' % model.criterion(points_all, weights, labels, TABLE_FB_
    PROBA))
    rr, cc = tl_visu.ellipse_perimeter(int(c1), int(c2), int(h), int(w), phi)
    plt.plot(cc, rr, '.', color=COLORS[i % len(COLORS)])

# plt.plot(centers[:, 1], centers[:, 0], 'o')
for i in range(len(centers)):
    plt.plot(centers[i, 1], centers[i, 0], 'o', color=COLORS[i % len(COLORS)])
_ = plt.xlim([0, seg.shape[1]]), plt.ylim([seg.shape[0], 0])
```

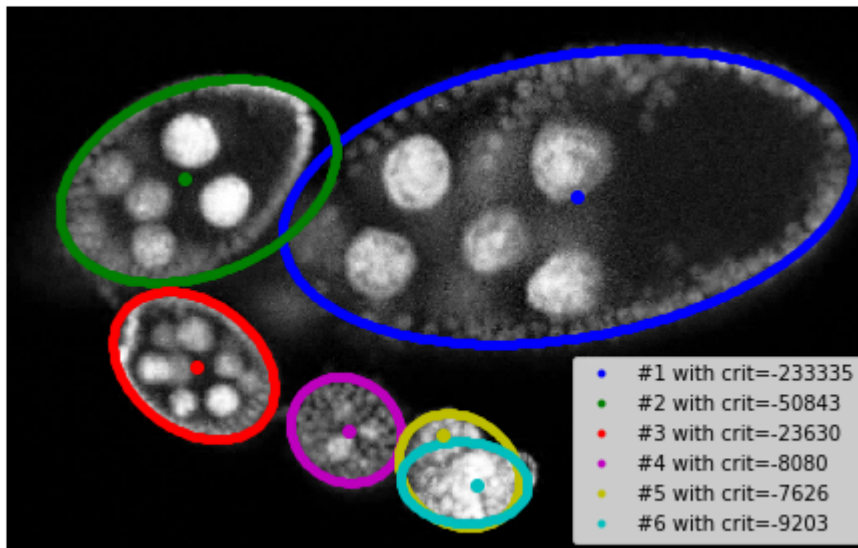
```
model params: (226, 666, 166, 342, 0.17135703308989025)
-> crit: -233334.76990445034
model params: (207, 227, 170, 110, 1.9451957515390759)
-> crit: -50842.71570683058
model params: (426, 222, 75, 104, -0.6423584612776315)
-> crit: -23629.744544518864
model params: (502, 402, 67, 60, 1.0308971690398319)
-> crit: -8080.076096030485
model params: (550, 536, 62, 75, -0.6448045520560932)
-> crit: -7626.153548780507
model params: (564, 542, 48, 75, -0.06859638756844778)
-> crit: -9202.723298924388
```



## Visualizations

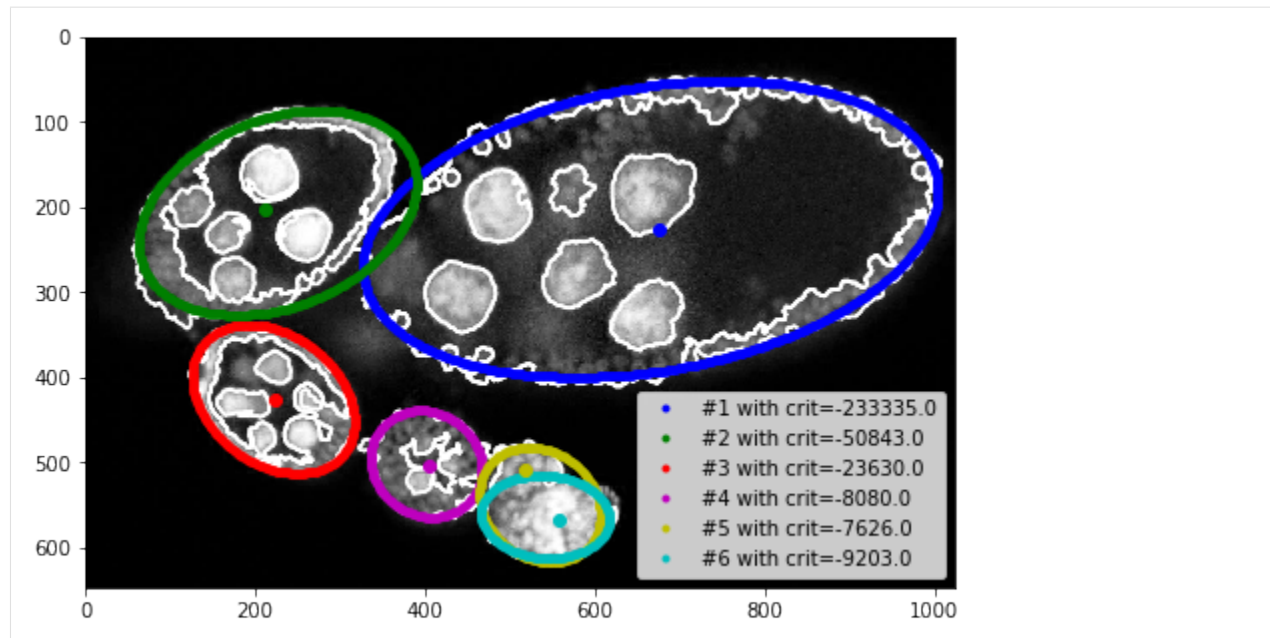
```
[12]: #fig = plt.figure(figsize=FIG_SIZE)
# plt.imshow(img)
plt.imshow(img[:, :, 0], cmap=plt.cm.Greys_r)
for i, params in enumerate(ellipses):
    c1, c2, h, w, phi = params
    rr, cc = tl_visu.ellipse_perimeter(int(c1), int(c2), int(h), int(w), phi)
    plt.plot(cc, rr, '.', color=COLORS[i % len(COLORS)], label='#%i with crit=%d' % (
        i + 1), int(crits[i]))
plt.legend(loc='lower right')

# plt.plot(centers[:, 1], centers[:, 0], 'ow')
for i in range(len(centers)):
    plt.plot(centers[i, 1], centers[i, 0], 'o', color=COLORS[i % len(COLORS)])
plt.xlim([0, seg.shape[1]], plt.ylim([seg.shape[0], 0])
_ = plt.axis('off'), fig.subplots_adjust(left=0, right=1, top=1, bottom=0)
# fig.savefig('fig2.pdf')
```



```
[14]: plt.figure(figsize=FIG_SIZE)
plt.imshow(img[:, :, 0], cmap=plt.cm.Greys_r)
plt.contour(seg, colors='w') #, plt.imshow(seg, alpha=0.2)
for i, params in enumerate(ellipses):
    c1, c2, h, w, phi = params
    rr, cc = tl_visu.ellipse_perimeter(int(c1), int(c2), int(h), int(w), phi)
    plt.plot(cc, rr, '.', color=COLORS[i % len(COLORS)], label='#%i with crit=%.1f' % (
        i + 1), crits[i])
plt.legend(loc='lower right')

# plt.plot(centers[:, 1], centers[:, 0], 'ow')
for i in range(len(centers)):
    plt.plot(centers[i, 1], centers[i, 0], 'o', color=COLORS[i % len(COLORS)])
_ = plt.xlim([0, seg.shape[1]], plt.ylim([seg.shape[0], 0])
```



```
[ ]:
```



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## GENERAL SUPERPIXEL IMAGE SEGMENTATION: (UN)SUPERVISED, CENTER DETECTION, REGION GROWING

This package is aiming at (un/semi)supervised segmentation on superpixels with computing some basic colour and texture features. This general segmentation can be followed by an object centre detection and proximate ellipse fitting to expected object boundaries. Last included method is region growing with learned shape prior also running on superpixel grid. The package contains several low-level Cython implementation to speed up some feature extraction methods. Overall the project/repository contains example codes with visualisation in ipython notebooks and experiments required for replicating all published results.

### 3.1 Superpixel segmentation with GraphCut regularisation

Image segmentation is widely used as an initial phase of many image processing tasks in computer vision and image analysis. Many recent segmentation methods use superpixels because they reduce the size of the segmentation problem by order of magnitude. Also, features on superpixels are much more robust than features on pixels only. We use spatial regularisation on superpixels to make segmented regions more compact. The segmentation pipeline comprises (i) computation of superpixels; (ii) extraction of descriptors such as colour and texture; (iii) soft classification, using a standard classifier for supervised learning, or the Gaussian Mixture Model for unsupervised learning; (iv) final segmentation using Graph Cut. We use this segmentation pipeline on real-world applications in medical imaging. We also show that unsupervised segmentation is sufficient for some situations, and provides similar results to those obtained using trained segmentation.

## 3.2 Object centre detection and Ellipse approximation

An image processing pipeline to detect and localize *Drosophila* egg chambers that consists of the following steps: (i) superpixel-based image segmentation into relevant tissue classes (see above); (ii) detection of egg center candidate using label histograms and ray features; (iii) clustering of center candidates and; (iv) area-based maximum likelihood ellipse model fitting.

## 3.3 Superpixel Region Growing with Shape prior

Region growing is a classical image segmentation method based on hierarchical region aggregation using local similarity rules. Our proposed approach differs from standard region growing in three essential aspects. First, it works on the level of superpixels instead of pixels, which leads to a substantial speedup. Second, our method uses learned statistical shape properties which encourage growing leading to plausible shapes. In particular, we use ray features to describe the object boundary. Third, our method can segment multiple objects and ensure that the segmentations do not overlap. The problem is represented as energy minimisation and is solved either greedily, or iteratively using GraphCuts.

## 3.4 References

- Borovec J., Svihlik J., Kybic J., Habart D. (2017). Supervised and unsupervised segmentation using superpixels, model estimation, and Graph Cut. *SPIE Journal of Electronic Imaging* 26(6), 061610. DOI: 10.1117/1.JEI.26.6.061610.
- Borovec J., Kybic J., Nava R. (2017) Detection and Localization of *Drosophila* Egg Chambers in Microscopy Images. In: Wang Q., Shi Y., Suk H.I., Suzuki K. (eds) *Machine Learning in Medical Imaging. MLMI 2017. LNCS*, vol 10541. Springer, Cham. DOI: 10.1007/978-3-319-67389-9\_3.
- Borovec J., Kybic J., Sugimoto, A. (2017). Region growing using superpixels with learned shape prior. *SPIE Journal of Electronic Imaging* 26(6), 061611. DOI: 10.1117/1.JEI.26.6.061611.



## PYTHON MODULE INDEX

### i

- `imsegm`, [132](#)
- `imsegm.annotation`, [44](#)
- `imsegm.classification`, [48](#)
- `imsegm.descriptors`, [64](#)
- `imsegm.ellipse_fitting`, [85](#)
- `imsegm.graph_cuts`, [92](#)
- `imsegm.labeling`, [100](#)
- `imsegm.pipelines`, [108](#)
- `imsegm.region_growing`, [113](#)
- `imsegm.superpixels`, [130](#)
- `imsegm.utilities`, [44](#)
- `imsegm.utilities.data_io`, [10](#)
- `imsegm.utilities.data_samples`, [22](#)
- `imsegm.utilities.drawing`, [24](#)
- `imsegm.utilities.experiments`, [36](#)
- `imsegm.utilities.read_zvi`, [41](#)



## Symbols

- `_CrossValidateGroups__iter_indexes()` (*imsegm.classification.CrossValidateGroups method*), 50
  - `_CrossValidate__steps()` (*imsegm.classification.CrossValidate method*), 49
  - `_asdict()` (*imsegm.utilities.read\_zvi.ImageTuple method*), 41
  - `_asdict()` (*imsegm.utilities.read\_zvi.ZviImageTuple method*), 42
  - `_asdict()` (*imsegm.utilities.read\_zvi.ZviItemTuple method*), 43
  - `_check_color_image()` (*in module imsegm.descriptors*), 64
  - `_check_color_image_seg()` (*in module imsegm.descriptors*), 64
  - `_check_exist_paths()` (*imsegm.utilities.experiments.Experiment method*), 36
  - `_check_gray_image_seg()` (*in module imsegm.descriptors*), 65
  - `_check_unrecognised_feature_group()` (*in module imsegm.descriptors*), 65
  - `_check_unrecognised_feature_names()` (*in module imsegm.descriptors*), 65
  - `_create_folder()` (*imsegm.utilities.experiments.Experiment method*), 36
  - `_ellipse()` (*in module imsegm.utilities.drawing*), 24
  - `_evaluate()` (*imsegm.utilities.experiments.Experiment method*), 37
  - `_fields` (*imsegm.utilities.read\_zvi.ImageTuple attribute*), 42
  - `_fields` (*imsegm.utilities.read\_zvi.ZviImageTuple attribute*), 43
  - `_fields` (*imsegm.utilities.read\_zvi.ZviItemTuple attribute*), 44
  - `_load_data()` (*imsegm.utilities.experiments.Experiment method*), 37
  - `_make()` (*imsegm.utilities.read\_zvi.ImageTuple class method*), 41
  - `_make()` (*imsegm.utilities.read\_zvi.ZviImageTuple class method*), 42
  - `_make()` (*imsegm.utilities.read\_zvi.ZviItemTuple class method*), 43
  - `_perform()` (*imsegm.utilities.experiments.Experiment method*), 37
  - `_replace()` (*imsegm.utilities.read\_zvi.ImageTuple method*), 41
  - `_replace()` (*imsegm.utilities.read\_zvi.ZviImageTuple method*), 42
  - `_replace()` (*imsegm.utilities.read\_zvi.ZviItemTuple method*), 43
  - `_source` (*imsegm.utilities.read\_zvi.ImageTuple attribute*), 42
  - `_source` (*imsegm.utilities.read\_zvi.ZviImageTuple attribute*), 43
  - `_source` (*imsegm.utilities.read\_zvi.ZviItemTuple attribute*), 44
  - `_summarise()` (*imsegm.utilities.experiments.Experiment method*), 37
- ## A
- `add_overlap_ellipse()` (*in module imsegm.ellipse\_fitting*), 86
  - `add_padding()` (*in module imsegm.utilities.data\_io*), 10
  - `adjust_bounding_box_crop()` (*in module imsegm.descriptors*), 65
  - `ANNOT_DROSOPHILA_DISC` (*in module imsegm.utilities.data\_samples*), 23
  - `ANNOT_DROSOPHILA_OVARY_2D` (*in module imsegm.utilities.data\_samples*), 23
  - `ANNOT_SLICE_DIST_TOL` (*in module imsegm.annotation*), 48
  - `append_final_stat()` (*in module imsegm.utilities.experiments*), 37
  - `Array()` (*imsegm.utilities.read\_zvi.ImageTuple property*), 41
  - `assign_label_by_max()` (*in module imsegm.labeling*), 100
  - `assign_label_by_threshold()` (*in module imsegm.labeling*), 100

`assume_bg_on_boundary()` (in module `im-segm.labeling`), 100

## B

`balance_dataset_by_()` (in module `im-segm.classification`), 51

`binary_image_from_coords()` (in module `im-segm.labeling`), 101

## C

`CLASSIF_NAME` (in module `imsegm.pipelines`), 112

`closest_point_on_line()` (in module `im-segm.utilities.drawing`), 25

`CLUSTER_METHOD` (in module `imsegm.pipelines`), 112

`COLUMNS_COORDS` (in module `im-segm.utilities.data_io`), 21

`COLUMNS_POSITION` (in module `imsegm.annotation`), 48

`COLUMNS_POSITION_EGG_ANNOT` (in module `im-segm.utilities.drawing`), 36

`compose_dict_label_features()` (in module `imsegm.classification`), 52

`compute_boundary_distances()` (in module `im-segm.labeling`), 101

`compute_centre_moment_points()` (in module `imsegm.region_growing`), 113

`compute_classif_metrics()` (in module `im-segm.classification`), 52

`compute_classif_stat_seg_annot()` (in module `imsegm.classification`), 52

`compute_color2d_superpixels_features()` (in module `imsegm.pipelines`), 108

`compute_cumulative_distrib()` (in module `im-segm.region_growing`), 113

`compute_data_costs_points()` (in module `im-segm.region_growing`), 113

`compute_distance_map()` (in module `im-segm.labeling`), 101

`compute_edge_model()` (in module `im-segm.graph_cuts`), 92

`compute_edge_weights()` (in module `im-segm.graph_cuts`), 93

`compute_image2d_color_statistic()` (in module `imsegm.descriptors`), 66

`compute_image3d_gray_statistic()` (in module `imsegm.descriptors`), 66

`compute_img_filter_response2d()` (in module `imsegm.descriptors`), 67

`compute_img_filter_response3d()` (in module `imsegm.descriptors`), 67

`compute_label_hist_proba()` (in module `im-segm.descriptors`), 67

`compute_label_hist_seg()` (in module `im-segm.descriptors`), 68

`compute_label_histograms_positions()` (in module `imsegm.descriptors`), 68

`compute_labels_overlap_matrix()` (in module `imsegm.labeling`), 102

`compute_metric_fpfm_tpfm()` (in module `im-segm.classification`), 53

`compute_metric_tpfm_tpfm()` (in module `im-segm.classification`), 53

`compute_multivarian_otsu()` (in module `im-segm.graph_cuts`), 93

`compute_object_shapes()` (in module `im-segm.region_growing`), 113

`compute_pairwise_cost()` (in module `im-segm.graph_cuts`), 94

`compute_pairwise_cost_from_transitions()` (in module `imsegm.graph_cuts`), 94

`compute_pairwise_penalty()` (in module `im-segm.region_growing`), 114

`compute_ray_features_positions()` (in module `imsegm.descriptors`), 69

`compute_ray_features_seg_2d()` (in module `imsegm.descriptors`), 70

`compute_ray_features_seg_2d_vectors()` (in module `imsegm.descriptors`), 71

`compute_rg_crit()` (in module `im-segm.region_growing`), 114

`compute_seg_object_shape()` (in module `im-segm.region_growing`), 114

`compute_seg_prob_fg()` (in module `im-segm.region_growing`), 115

`compute_selected_features_color2d()` (in module `imsegm.descriptors`), 71

`compute_selected_features_gray2d()` (in module `imsegm.descriptors`), 72

`compute_selected_features_gray3d()` (in module `imsegm.descriptors`), 73

`compute_selected_features_img2d()` (in module `imsegm.descriptors`), 73

`compute_shape_prior_table_cdf()` (in module `imsegm.region_growing`), 115

`compute_spatial_dist()` (in module `im-segm.graph_cuts`), 94

`compute_stat_per_image()` (in module `im-segm.classification`), 54

`compute_texture_desc_lm_img2d_clr()` (in module `imsegm.descriptors`), 73

`compute_texture_desc_lm_img3d_val()` (in module `imsegm.descriptors`), 74

`compute_tp_tn_fp_fn()` (in module `im-segm.classification`), 54

`compute_unary_cost()` (in module `im-segm.graph_cuts`), 95

`compute_update_shape_costs_points_close_mean_cdf()` (in module `imsegm.region_growing`), 116

compute\_update\_shape\_costs\_points\_table\_create\_filter\_bank\_lm\_2d() (in module *im-segm.region\_growing*), 117

CONFIG\_YAML (in module *im-segm.utilities.experiments*), 40

contour\_binary\_map() (in module *im-segm.labeling*), 102

contour\_coords() (in module *im-segm.labeling*), 103

convert\_dict\_label\_features\_2\_vectors() (in module *im-segm.classification*), 55

convert\_img\_2\_nifti\_gray() (in module *im-segm.utilities.data\_io*), 11

convert\_img\_2\_nifti\_rgb() (in module *im-segm.utilities.data\_io*), 11

convert\_img\_color\_from\_rgb() (in module *im-segm.utilities.data\_io*), 11

convert\_img\_color\_to\_rgb() (in module *im-segm.utilities.data\_io*), 12

convert\_img\_colors\_to\_labels() (in module *im-segm.annotation*), 44

convert\_img\_colors\_to\_labels\_reverted() (in module *im-segm.annotation*), 45

convert\_img\_labels\_to\_colors() (in module *im-segm.annotation*), 45

convert\_nifti\_2\_img() (in module *im-segm.utilities.data\_io*), 12

convert\_segms\_2\_list() (in module *im-segm.labeling*), 103

convert\_set\_features\_labels\_2\_dataset() (in module *im-segm.classification*), 55

Count() (*im-segm.utilities.read\_zvi.ZviImageTuple* property), 42

Count() (*im-segm.utilities.read\_zvi.ZviItemTuple* property), 43

count\_label\_transitions\_connected\_segments() (in module *im-segm.graph\_cuts*), 95

CPU\_COUNT (in module *im-segm.utilities.experiments*), 40

create\_classif\_search() (in module *im-segm.classification*), 55

create\_classif\_search\_train\_export() (in module *im-segm.classification*), 56

create\_classifiers() (in module *im-segm.classification*), 57

create\_clf\_param\_search\_distrib() (in module *im-segm.classification*), 57

create\_clf\_param\_search\_grid() (in module *im-segm.classification*), 57

create\_clf\_pipeline() (in module *im-segm.classification*), 58

create\_experiment\_folder() (in module *im-segm.utilities.experiments*), 38

create\_figure\_by\_image() (in module *im-segm.utilities.drawing*), 25

create\_filter\_bank\_lm\_2d() (in module *im-segm.descriptors*), 74

create\_pairwise\_matrix() (in module *im-segm.graph\_cuts*), 95

create\_pairwise\_matrix\_specif() (in module *im-segm.graph\_cuts*), 96

create\_pairwise\_matrix\_uniform() (in module *im-segm.graph\_cuts*), 96

create\_pipeline\_neuron\_net() (in module *im-segm.classification*), 58

create\_subfolders() (in module *im-segm.utilities.experiments*), 38

criterion() (*im-segm.ellipse\_fitting.EllipseModelSegm* method), 85

CROSS\_VAL\_LEAVE\_OUT (in module *im-segm.pipelines*), 112

CrossValidate (class in *im-segm.classification*), 48

CrossValidateGroups (class in *im-segm.classification*), 49

cut\_object() (in module *im-segm.utilities.data\_io*), 12

cython\_img2d\_color\_energy() (in module *im-segm.descriptors*), 75

cython\_img2d\_color\_mean() (in module *im-segm.descriptors*), 75

cython\_img2d\_color\_std() (in module *im-segm.descriptors*), 76

cython\_img3d\_gray\_energy() (in module *im-segm.descriptors*), 76

cython\_img3d\_gray\_mean() (in module *im-segm.descriptors*), 77

cython\_img3d\_gray\_std() (in module *im-segm.descriptors*), 77

cython\_label\_hist\_seg2d() (in module *im-segm.descriptors*), 77

cython\_ray\_features\_seg2d() (in module *im-segm.descriptors*), 78

## D

DEFAULT\_CLASSIF\_NAME (in module *im-segm.classification*), 63

DEFAULT\_CLUSTERING (in module *im-segm.classification*), 64

DEFAULT\_FILTERS\_SIGMAS (in module *im-segm.descriptors*), 84

DEFAULT\_GC\_ITERATIONS (in module *im-segm.graph\_cuts*), 99

Depth() (*im-segm.utilities.read\_zvi.ImageTuple* property), 41

Depth() (*im-segm.utilities.read\_zvi.ZviImageTuple* property), 42

Depth() (*im-segm.utilities.read\_zvi.ZviItemTuple* property), 43

DICT\_COLOURS (in module *im-segm.annotation*), 48

DICT\_CONVERT\_COLOR\_FROM\_RGB (in module *im-segm.utilities.data\_io*), 21  
DICT\_CONVERT\_COLOR\_TO\_RGB (in module *im-segm.utilities.data\_io*), 21  
DICT\_LABEL\_MARKER (in module *im-segm.utilities.drawing*), 36  
DICT\_SCORING (in module *imsegm.classification*), 64  
down\_sample\_dict\_features\_kmean() (in module *imsegm.classification*), 58  
down\_sample\_dict\_features\_random() (in module *imsegm.classification*), 58  
down\_sample\_dict\_features\_unique() (in module *imsegm.classification*), 59  
draw\_color\_labeling() (in module *im-segm.utilities.drawing*), 25  
draw\_eggs\_ellipse() (in module *im-segm.utilities.drawing*), 25  
draw\_eggs\_rectangle() (in module *im-segm.utilities.drawing*), 26  
draw\_graphcut\_unary\_cost\_segments() (in module *imsegm.utilities.drawing*), 26  
draw\_graphcut\_weighted\_edges() (in module *imsegm.utilities.drawing*), 27  
draw\_image\_clusters\_centers() (in module *imsegm.utilities.drawing*), 27  
draw\_image\_seg points() (in module *im-segm.utilities.drawing*), 28  
draw\_rg2sp\_results() (in module *im-segm.utilities.drawing*), 28

## E

ellipse() (in module *imsegm.utilities.drawing*), 28  
ellipse\_perimeter() (in module *im-segm.utilities.drawing*), 29  
EllipseModelSegm (class in *imsegm.ellipse\_fitting*), 85  
enforce\_center\_labels() (in module *im-segm.region\_growing*), 119  
estim\_class\_model() (in module *im-segm.graph\_cuts*), 96  
estim\_class\_model\_gmm() (in module *im-segm.graph\_cuts*), 97  
estim\_class\_model\_kmeans() (in module *im-segm.graph\_cuts*), 97  
estim\_gmm\_params() (in module *im-segm.graph\_cuts*), 98  
estim\_model\_classes\_group() (in module *im-segm.pipelines*), 109  
eval\_classif\_cross\_val\_roc() (in module *im-segm.classification*), 59  
eval\_classif\_cross\_val\_scores() (in module *imsegm.classification*), 60  
Experiment (class in *imsegm.utilities.experiments*), 36

export\_image() (in module *im-segm.utilities.data\_io*), 13  
export\_results\_clf\_search() (in module *im-segm.classification*), 61  
extend\_list\_params() (in module *im-segm.utilities.experiments*), 39

## F

feature\_scoring\_selection() (in module *im-segm.classification*), 61  
FEATURES\_SET\_ALL (in module *imsegm.descriptors*), 84  
FEATURES\_SET\_COLOR (in module *im-segm.descriptors*), 84  
FEATURES\_SET\_TEXTURE (in module *im-segm.descriptors*), 84  
FEATURES\_SET\_TEXTURE\_SHORT (in module *im-segm.descriptors*), 84  
figure\_annot\_slic\_histogram\_labels() (in module *imsegm.utilities.drawing*), 30  
figure\_ellipse\_fitting() (in module *im-segm.utilities.drawing*), 30  
figure\_image\_adjustment() (in module *im-segm.utilities.drawing*), 31  
figure\_image\_seg centres() (in module *im-segm.utilities.drawing*), 31  
figure\_image\_seg\_results() (in module *im-segm.utilities.drawing*), 31  
figure\_overlap\_annot\_seg\_image() (in module *imsegm.utilities.drawing*), 32  
figure\_ray\_feature() (in module *im-segm.utilities.drawing*), 32  
figure\_rg2sp\_debug\_complete() (in module *imsegm.utilities.drawing*), 32  
figure\_seg\_boundary\_dist() (in module *im-segm.utilities.drawing*), 33  
figure\_seg\_graphcut\_debug() (in module *im-segm.utilities.drawing*), 33  
figure\_used\_samples() (in module *im-segm.utilities.drawing*), 34  
FILE\_LOGS (in module *imsegm.utilities.experiments*), 40  
FileName() (*imsegm.utilities.read\_zvi.ZviImageTuple* property), 42  
FileName() (*imsegm.utilities.read\_zvi.ZviItemTuple* property), 43  
filter\_boundary\_points() (in module *im-segm.ellipse\_fitting*), 87  
find\_files\_match\_names\_across\_dirs() (in module *imsegm.utilities.data\_io*), 14  
FORMAT\_DT (in module *imsegm.utilities.experiments*), 40  
FTS\_SET\_SIMPLE (in module *imsegm.pipelines*), 112



## G

GC\_REPLACE\_INF (in module *im-segm.region\_growing*), 129

get\_dir() (in module *imsegm.utilities.read\_zvi*), 44

get\_hex() (in module *imsegm.utilities.read\_zvi*), 44

get\_image2d\_boundary\_color() (in module *im-segm.utilities.data\_io*), 14

get\_image\_path() (in module *im-segm.utilities.data\_samples*), 22

get\_layer\_count() (in module *im-segm.utilities.read\_zvi*), 44

get\_neighboring\_candidates() (in module *im-segm.region\_growing*), 119

get\_neighboring\_segments() (in module *im-segm.superpixels*), 130

get\_segment\_diffs\_2d\_conn4() (in module *im-segm.superpixels*), 130

get\_segment\_diffs\_3d\_conn6() (in module *im-segm.superpixels*), 130

get\_slic\_points\_labels() (in module *im-segm.ellipse\_fitting*), 87

get\_vertexes\_edges() (in module *im-segm.graph\_cuts*), 98

group\_images\_frequent\_colors() (in module *imsegm.annotation*), 45

## H

Height() (*imsegm.utilities.read\_zvi.ImageTuple* property), 41

Height() (*imsegm.utilities.read\_zvi.ZviImageTuple* property), 42

Height() (*imsegm.utilities.read\_zvi.ZviItemTuple* property), 43

HIST\_CIRCLE\_DIAGONALS (in module *im-segm.descriptors*), 84

histogram\_regions\_labels\_counts() (in module *imsegm.labeling*), 103

histogram\_regions\_labels\_norm() (in module *imsegm.labeling*), 104

HoldOut (class in *imsegm.classification*), 51

## I

i32() (in module *imsegm.utilities.read\_zvi*), 44

Image() (*imsegm.utilities.read\_zvi.ZviItemTuple* property), 43

IMAGE\_3CLS (in module *im-segm.utilities.data\_samples*), 23

image\_color\_2\_labels() (in module *im-segm.annotation*), 46

IMAGE\_DROSOPHILA\_DISC (in module *im-segm.utilities.data\_samples*), 23

IMAGE\_DROSOPHILA\_OVARY\_2D (in module *im-segm.utilities.data\_samples*), 23

IMAGE\_DROSOPHILA\_OVARY\_3D (in module *im-segm.utilities.data\_samples*), 23

image\_frequent\_colors() (in module *im-segm.annotation*), 46

IMAGE\_HISTOL\_CIMA (in module *im-segm.utilities.data\_samples*), 24

IMAGE\_HISTOL\_FLAGSHIP (in module *im-segm.utilities.data\_samples*), 24

image\_inpaint\_pixels() (in module *im-segm.annotation*), 46

IMAGE\_LANGER\_ISLET (in module *im-segm.utilities.data\_samples*), 24

IMAGE\_LENNA (in module *im-segm.utilities.data\_samples*), 24

IMAGE\_OBJECTS (in module *im-segm.utilities.data\_samples*), 24

image\_open() (in module *imsegm.utilities.data\_io*), 15

IMAGE\_SPACING (in module *imsegm.superpixels*), 132

IMAGE\_STAR (in module *im-segm.utilities.data\_samples*), 24

image\_subtract\_gauss\_smooth() (in module *imsegm.descriptors*), 78

ImageTuple (class in *imsegm.utilities.read\_zvi*), 41

imsegm (module), 132

imsegm.annotation (module), 44

imsegm.classification (module), 48

imsegm.descriptors (module), 64

imsegm.ellipse\_fitting (module), 85

imsegm.graph\_cuts (module), 92

imsegm.labeling (module), 100

imsegm.pipelines (module), 108

imsegm.region\_growing (module), 113

imsegm.superpixels (module), 130

imsegm.utilities (module), 44

imsegm.utilities.data\_io (module), 10

imsegm.utilities.data\_samples (module), 22

imsegm.utilities.drawing (module), 24

imsegm.utilities.experiments (module), 36

imsegm.utilities.read\_zvi (module), 41

insert\_gc\_debug\_images() (in module *im-segm.graph\_cuts*), 98

interpolate\_ray\_dist() (in module *im-segm.descriptors*), 79

io\_image\_decorate() (in module *im-segm.utilities.data\_io*), 15

io\_imread() (in module *imsegm.utilities.data\_io*), 15

io\_imsave() (in module *imsegm.utilities.data\_io*), 15

is\_iterable() (in module *im-segm.utilities.experiments*), 39

## L

Layers() (*imsegm.utilities.read\_zvi.ZviImageTuple* property), 42

Layers() (*imsegm.utilities.read\_zvi.ZviItemTuple property*), 43

LIST\_ALL\_IMAGES (*in module imsegm.utilities.data\_samples*), 24

load\_classifier() (*in module imsegm.classification*), 62

load\_complete\_image\_folder() (*in module imsegm.utilities.data\_io*), 15

load\_config\_yaml() (*in module imsegm.utilities.experiments*), 39

load\_image() (*in module imsegm.utilities.data\_io*), 15

load\_image() (*in module imsegm.utilities.read\_zvi*), 44

load\_image\_2d() (*in module imsegm.utilities.data\_io*), 16

load\_image\_tiff\_volume() (*in module imsegm.utilities.data\_io*), 16

load\_images\_list() (*in module imsegm.utilities.data\_io*), 17

load\_img\_double\_band\_split() (*in module imsegm.utilities.data\_io*), 17

load\_info\_group\_by\_slices() (*in module imsegm.annotation*), 46

load\_landmarks\_csv() (*in module imsegm.utilities.data\_io*), 18

load\_landmarks\_txt() (*in module imsegm.utilities.data\_io*), 18

load\_params\_from\_txt() (*in module imsegm.utilities.data\_io*), 18

load\_sample\_image() (*in module imsegm.utilities.data\_samples*), 22

load\_tiff\_volume\_split\_double\_band() (*in module imsegm.utilities.data\_io*), 18

load\_zvi\_volume\_double\_band\_split() (*in module imsegm.utilities.data\_io*), 19

make\_overlap\_images\_optical() (*in module imsegm.utilities.drawing*), 34

mask\_seg\_labels() (*in module imsegm.labeling*), 104

MAX\_FIGURE\_SIZE (*in module imsegm.ellipse\_fitting*), 92

MAX\_PAIRWISE\_COST (*in module imsegm.graph\_cuts*), 99

MAX\_SIGNAL\_RESPONSE (*in module imsegm.descriptors*), 84

MAX\_UNARY\_PROB (*in module imsegm.region\_growing*), 129

merge\_image\_channels() (*in module imsegm.utilities.data\_io*), 19

merge\_object\_masks() (*in module imsegm.utilities.drawing*), 35

merge\_probab\_labeling\_2d() (*in module imsegm.labeling*), 105

METRIC\_AVERAGES (*in module imsegm.classification*), 64

METRIC\_SCORING (*in module imsegm.classification*), 64

MIN\_ELLIPSE\_DAIM (*in module imsegm.ellipse\_fitting*), 92

MIN\_MAX\_EDGE\_WEIGHT (*in module imsegm.graph\_cuts*), 99

MIN\_SHAPE\_PROB (*in module imsegm.region\_growing*), 129

MIN\_UNARY\_PROB (*in module imsegm.graph\_cuts*), 99

## M

m\_PluginCLSID() (*imsegm.utilities.read\_zvi.ZviImageTuple property*), 43

make\_edge\_filter2d() (*in module imsegm.descriptors*), 79

make\_gaussian\_filter1d() (*in module imsegm.descriptors*), 79

make\_graph\_seg\_connect\_grid2d\_conn4() (*in module imsegm.superpixels*), 130

make\_graph\_seg\_connect\_grid3d\_conn6() (*in module imsegm.superpixels*), 130

make\_graph\_segment\_connect\_edges() (*in module imsegm.superpixels*), 131

make\_overlap\_images\_chess() (*in module imsegm.utilities.drawing*), 34

NAME\_CSV\_CLASSIF\_CV\_ROC (*in module imsegm.classification*), 64

NAME\_CSV\_CLASSIF\_CV\_SCORES (*in module imsegm.classification*), 64

NAME\_CSV\_FEATURES\_SELECT (*in module imsegm.classification*), 64

NAME\_TXT\_CLASSIF\_CV\_AUC (*in module imsegm.classification*), 64

NAMES\_FEATURE\_FLAGS (*in module imsegm.descriptors*), 84

NB\_WORKERS (*in module imsegm.pipelines*), 112

nb\_workers() (*in module imsegm.utilities.experiments*), 39

NB\_WORKERS\_SERACH (*in module imsegm.classification*), 64

neighbour\_connect4() (*in module imsegm.labeling*), 105

norm\_alpha() (*in module imsegm.utilities.drawing*), 35

norm\_features() (*in module imsegm.descriptors*), 79

numpy\_img2d\_color\_energy() (*in module imsegm.descriptors*), 79



`numpy_img2d_color_mean()` (in module `im-segm.descriptors`), 80  
`numpy_img2d_color_median()` (in module `im-segm.descriptors`), 80  
`numpy_img2d_color_std()` (in module `im-segm.descriptors`), 80  
`numpy_img3d_gray_energy()` (in module `im-segm.descriptors`), 81  
`numpy_img3d_gray_mean()` (in module `im-segm.descriptors`), 81  
`numpy_img3d_gray_median()` (in module `im-segm.descriptors`), 82  
`numpy_img3d_gray_std()` (in module `im-segm.descriptors`), 82  
`numpy_ray_features_seg2d()` (in module `im-segm.descriptors`), 82

## O

`object_segmentation_graphcut_pixels()` (in module `imsegm.region_growing`), 119  
`object_segmentation_graphcut_slic()` (in module `imsegm.region_growing`), 120  
`Others()` (`imsegm.utilities.read_zvi.ZviImageTuple` property), 42  
`Others()` (`imsegm.utilities.read_zvi.ZviItemTuple` property), 43

## P

`parse_annot_rectangles()` (in module `im-segm.utilities.drawing`), 35  
`parse_image()` (in module `imsegm.utilities.read_zvi`), 44  
`PATH_IMAGES` (in module `im-segm.utilities.data_samples`), 24  
`pipe_color2d_slic_features_model_graphcut()` (in module `imsegm.pipelines`), 109  
`pipe_gray3d_slic_features_model_graphcut()` (in module `imsegm.pipelines`), 110  
`PIXEL_FORMAT()` (`im-segm.utilities.read_zvi.ImageTuple` property), 41  
`PIXEL_FORMAT()` (`im-segm.utilities.read_zvi.ZviImageTuple` property), 42  
`PIXEL_FORMAT()` (`im-segm.utilities.read_zvi.ZviItemTuple` property), 43  
`PixelWidth()` (`imsegm.utilities.read_zvi.ImageTuple` property), 42  
`prepare_boundary_points_close()` (in module `imsegm.ellipse_fitting`), 87  
`prepare_boundary_points_ray_dist()` (in module `imsegm.ellipse_fitting`), 88

`prepare_boundary_points_ray_edge()` (in module `imsegm.ellipse_fitting`), 88  
`prepare_boundary_points_ray_join()` (in module `imsegm.ellipse_fitting`), 89  
`prepare_boundary_points_ray_mean()` (in module `imsegm.ellipse_fitting`), 89  
`prepare_graphcut_variables()` (in module `im-segm.region_growing`), 121

## Q

`quantize_image_nearest_color()` (in module `imsegm.annotation`), 47  
`quantize_image_nearest_pixel()` (in module `imsegm.annotation`), 47

## R

`ransac_seg()` (in module `imsegm.ellipse_fitting`), 90  
`read_image_container_content()` (in module `imsegm.utilities.read_zvi`), 44  
`read_item_storage_content()` (in module `im-segm.utilities.read_zvi`), 44  
`read_struct()` (in module `imsegm.utilities.read_zvi`), 44  
`reconstruct_ray_features_2d()` (in module `imsegm.descriptors`), 83  
`reduce_close_points()` (in module `im-segm.descriptors`), 83  
`region_growing_shape_slic_graphcut()` (in module `imsegm.region_growing`), 121  
`region_growing_shape_slic_greedy()` (in module `imsegm.region_growing`), 124  
`relabel_by_dict()` (in module `imsegm.labeling`), 105  
`relabel_max_overlap_merge()` (in module `im-segm.labeling`), 105  
`relabel_max_overlap_unique()` (in module `im-segm.labeling`), 106  
`relabel_sequential()` (in module `im-segm.classification`), 62  
`RESULTS_TXT` (in module `im-segm.utilities.experiments`), 40  
`RG2SP_THRESHOLDS` (in module `im-segm.region_growing`), 129  
`ROUND_UNIQUE_FTS_DIGITS` (in module `im-segm.classification`), 64  
`run()` (`imsegm.utilities.experiments.Experiment` method), 37

## S

`sample_color_image_rand_segment()` (in module `imsegm.utilities.data_samples`), 22  
`SAMPLE_SEG_NB_CLASSES` (in module `im-segm.utilities.data_samples`), 24

SAMPLE\_SEG\_SIZE\_2D\_NORM (in module *im-segm.utilities.data\_samples*), 24  
 SAMPLE\_SEG\_SIZE\_2D\_SMALL (in module *im-segm.utilities.data\_samples*), 24  
 SAMPLE\_SEG\_SIZE\_3D\_SMALL (in module *im-segm.utilities.data\_samples*), 24  
 sample\_segment\_vertical\_2d() (in module *im-segm.utilities.data\_samples*), 23  
 sample\_segment\_vertical\_3d() (in module *im-segm.utilities.data\_samples*), 23  
 save\_classifier() (in module *im-segm.classification*), 62  
 save\_config\_yaml() (in module *im-segm.utilities.experiments*), 40  
 save\_landmarks\_csv() (in module *im-segm.utilities.data\_io*), 19  
 save\_landmarks\_txt() (in module *im-segm.utilities.data\_io*), 20  
 scale\_image\_intensity() (in module *im-segm.utilities.data\_io*), 20  
 scale\_image\_size() (in module *im-segm.utilities.data\_io*), 20  
 scale\_image\_vals\_in\_range() (in module *im-segm.utilities.data\_io*), 20  
 Scaling() (*imsegm.utilities.read\_zvi.ZviImageTuple* property), 42  
 Scaling() (*imsegm.utilities.read\_zvi.ZviItemTuple* property), 43  
 search\_params\_cut\_down\_max\_nb\_iter() (in module *imsegm.classification*), 63  
 segm\_labels\_assignment() (in module *im-segm.labeling*), 107  
 segment\_color2d\_slic\_features\_model\_graphcut() (in module *imsegm.pipelines*), 110  
 segment\_graph\_cut\_general() (in module *im-segm.graph\_cuts*), 98  
 segment\_slic\_img2d() (in module *im-segm.superpixels*), 131  
 segment\_slic\_img3d\_gray() (in module *im-segm.superpixels*), 131  
 sequence\_labels\_merge() (in module *im-segm.labeling*), 108  
 set\_experiment\_logger() (in module *im-segm.utilities.experiments*), 40  
 shift\_ray\_features() (in module *im-segm.descriptors*), 84  
 SHORT\_FILTERS\_SIGMAS (in module *im-segm.descriptors*), 85  
 shuffle\_features\_labels() (in module *im-segm.classification*), 63  
 SIZE\_CHESS\_FIELD (in module *im-segm.utilities.drawing*), 36  
 split\_segm\_background\_foreground() (in module *imsegm.ellipse\_fitting*), 91  
 string\_dict() (in module *im-segm.utilities.experiments*), 40  
 STRUC\_ELEM\_BG (in module *imsegm.ellipse\_fitting*), 92  
 STRUC\_ELEM\_FG (in module *imsegm.ellipse\_fitting*), 92  
 superpixel\_centers() (in module *im-segm.superpixels*), 131  
 swap\_coord\_x\_y() (in module *im-segm.utilities.data\_io*), 21

## T

TEMPLATE\_NAME\_CLF (in module *im-segm.classification*), 64  
 train\_classif\_color2d\_slic\_features() (in module *imsegm.pipelines*), 111  
 transform\_rays\_model\_cdf\_histograms() (in module *imsegm.region\_growing*), 127  
 transform\_rays\_model\_cdf\_kmeans() (in module *imsegm.region\_growing*), 127  
 transform\_rays\_model\_cdf\_mixture() (in module *imsegm.region\_growing*), 127  
 transform\_rays\_model\_cdf\_spectral() (in module *imsegm.region\_growing*), 128  
 transform\_rays\_model\_sets\_mean\_cdf\_kmeans() (in module *imsegm.region\_growing*), 128  
 transform\_rays\_model\_sets\_mean\_cdf\_mixture() (in module *imsegm.region\_growing*), 128  
 try\_decorator() (in module *im-segm.utilities.experiments*), 40

## U

unique\_image\_colors() (in module *im-segm.annotation*), 47  
 unique\_rows() (in module *imsegm.classification*), 63  
 update\_path() (in module *imsegm.utilities.data\_io*), 21  
 update\_shape\_costs\_points() (in module *im-segm.region\_growing*), 129

## V

ValidBitsPerPixel() (*im-segm.utilities.read\_zvi.ImageTuple* property), 42  
 ValidBitsPerPixel() (*im-segm.utilities.read\_zvi.ZviImageTuple* property), 42  
 ValidBitsPerPixel() (*im-segm.utilities.read\_zvi.ZviItemTuple* property), 43  
 Version() (*imsegm.utilities.read\_zvi.ImageTuple* property), 42  
 Version() (*imsegm.utilities.read\_zvi.ZviImageTuple* property), 42  
 Version() (*imsegm.utilities.read\_zvi.ZviItemTuple* property), 43

## W

`Width()` (*imsegm.utilities.read\_zvi.ImageTuple* property), [42](#)  
`Width()` (*imsegm.utilities.read\_zvi.ZviImageTuple* property), [43](#)  
`Width()` (*imsegm.utilities.read\_zvi.ZviItemTuple* property), [43](#)  
`WrapExecuteSequence` (class in *imsegm.utilities.experiments*), [37](#)  
`wrapper_compute_color2d_slic_features_labels()` (in module *imsegm.pipelines*), [112](#)

## Z

`zvi_read()` (in module *imsegm.utilities.read\_zvi*), [44](#)  
`ZviImageTuple` (class in *imsegm.utilities.read\_zvi*), [42](#)  
`ZviItemTuple` (class in *imsegm.utilities.read\_zvi*), [43](#)