

## **Documentation: trainsegmentation v 0.1.7**

### ***Importing training data***

Training data should be 2d-numpy arrays. Labeled images should be the same size as the corresponding training image and can be imported from binary masks with the filenames from a separate directory using `import_training_data`.

**`import_training_data`**(imgdir,maskdir,ext = '.tif'): imports images and labels from different directories to use for generating feature sets and training data.

#### **Parameters:**

imgdir: *str*

path to the image directory as string

maskdir: *list of str*

list of strings indicating paths containing labeled images with matching filenames to the images in the image directory to ensure corresponding images and labels.

ext: *str*

extension of image filenames. Default is '.tif'

#### **Returns:**

IMG: *list of ndarray*

list containing images (ndarrays) imported from imgdir

LABELS: *list of ndarray*

list containing labeled images corresponding to IMG. Label numbers match the order of paths listed in maskdir (e.g. 1, 2, 3, etc.)

### ***Generating feature sets for training data***

Once training data is imported, feature sets are generated to create the input needed for pixel classification.

**`get_training_data`**(IMG,LABELS,featureselect,loaddatafile = None, savedatafile = None): takes lists of images, corresponding labels and selected features (see Defining Features) to generate feature sets then output training data to use with a classifier..

#### **Parameters:**

IMG: *list of ndarray*

list containing images (ndarrays) imported from imgdir

LABELS: *list of ndarray*

list containing labeled images corresponding to IMG. Label numbers match the order of paths listed in maskdir (e.g. 1, 2, 3, etc.)

featureselect: *list of str*

list of strings indicating features (see Defining Features)

#### **Returns:**

traininglabels: *ndarray*

a flattened (1d) ndarray containing the training labels  
trainingfeatures: *ndarray*  
a flattened (2d) ndarray containing the feature sets, wherein len(axis = 1) is equal to the number of features  
featuresselect: *list of str*  
list of strings indicating features (see Defining Features)  
loaddatafile: *str*  
string with path to a previously generated data file which will be appended to the data generated from the input images and labels  
savedatafile: *str*  
string with path to save data file as a pickle file

## ***Training a classifier***

The training data is then used to train an sklearn classifier.

**train\_classifier**(traininglabels, trainingfeatures, featuresselect, saveclfthtofile = None, clf = None). Trains a classifier using labels and features generated by get\_training\_data.

### **Parameters:**

traininglabels: *ndarray*  
a flattened (1d) ndarray containing the training labels  
trainingfeatures: *ndarray*  
a flattened (2d) ndarray containing the feature sets, wherein len(axis = 1) is equal to the number of features  
featuresselect: *list of str*  
list of strings indicating features (see Defining Features)  
saveclfthtofile: *str*  
string with path to classifier as a pickle file  
clf: *sklearn classifier*  
A sklearn classifier, default (None) is clf =  
sklearn.ensemble.RandomForestClassifier(n\_estimators=50, n\_jobs=-1,max\_depth=10,  
max\_samples=0.05)

### **Returns:**

clf: *sklearn classifier*  
trained sklearn classifier

## ***Applying a classifier***

Once trained, classifiers can be applied to similar images to generate probability or binary masks.

**load\_classifier**(clffile): loads a classifier (pickle file) saved by train\_classifier

### **Parameters:**

clffile: *str*  
string with path to pickle file containing classifier

**Returns:**

clf: *sklearn classifier*  
trained sklearn classifier

**classify\_image**(img,clf,featureselect): generates feature set for an input image and outputs predicted classification.

**Parameters:**

img: *ndarray*  
input image  
clf: *sklearn classifier*  
classifier trained by train\_classifier  
featureselect: *list of str*  
list of strings indicating features (see Defining Features) that must match those used when training the classifier

**Returns:**

result: *ndarray*  
labeled ndarray with shape img.shape

**classify\_image\_probability**(img,clf,featureselect): generates feature set for an input image and outputs probability of classification.

**Parameters:**

img: *ndarray*  
input image  
clf: *sklearn classifier*  
classifier trained by train\_classifier  
featureselect: *list of str*  
list of strings indicating features (see Defining Features) that must match those used when training the classifier

**Returns:**

result: *ndarray*  
labeled ndarray with shape (img.shape[0], img.shape[1], # of labels)

**classify\_image\_label**(img, clf, featureselect, selectlabel = 1): generates feature set for an input image and outputs predicted classification for the indicated label.

**Parameters:**

img: *ndarray*  
input image  
clf: *sklearn classifier*  
classifier trained by train\_classifier  
featureselect: *list of str*

list of strings indicating features (see Defining Features) that must match those used when training the classifier

selectlabel: *int*

integer indicating the label to output classification

**Returns:**

result: *ndarray*

labeled ndarray with shape img.shape

**classify\_image\_label\_probability**(img,clf,featureselect,selectlabel = 1): generates feature set for an input image and outputs probability of classification for the indicated label.

**Parameters:**

img: *ndarray*

input image

clf: *sklearn classifier*

classifier trained by train\_classifier

featureselect: *list of str*

list of strings indicating features (see Defining Features) that must match those used when training the classifier

selectlabel: *int*

integer indicating the label to output classification

**Returns:**

label: *ndarray*

labeled ndarray with shape img.shape

**threshold\_mask**(img, threshmethod = sklearn.filters.threshold\_minimum): generates a mask by thresholding the input image with the given threshold method.

**Parameters:**

img: *ndarray*

input image

threshmethod: *function*

a thresholding function

**Returns:**

mask: *ndarray*

labeled ndarray with shape img.shape; outputs a binary mask 0 or 255.

## Defining Features

The following functions are used to generate image features. The function names are passed into training and classification functions as strings in a list (e.g. ['Neighbors','Mean']).

**Difference\_of\_Gaussians**(img,minSigma = 1,maxSigma = 16)\*\*: performs convolutions with Gaussian kernels with the normal  $n$  variations of  $\sigma$  and subtracts the previous iteration to obtain the feature image, with the first image to be subtracted being the original image.

### Parameters:

img: *ndarray*

input image

minSigma: *int*

minimum value of  $\sigma$

maxSigma: *int*

maximum value of  $\sigma$

### Returns:

meta: *list of str*

Meta data for *features*, wherein strings indicate feature of each ndarray in *features*. e.g. 'Difference\_of\_Gaussian\_1\_0' wherein the integer indicates  $\sigma$  for initial image and  $\sigma$  for the subtracted image respectively (the original image is 0).

features: *list of ndarray*

List containing feature images each of shape(*img*)

**Gaussian\_blur**(img,minSigma = 1,maxSigma = 16)\*\*: performs  $n$  individual convolutions with Gaussian kernels with the normal  $n$  variations of  $\sigma$ . The larger the radius the more blurred the image becomes until the pixels are homogeneous

### Parameters:

img: *ndarray*

input image

minSigma: *int*

minimum value of  $\sigma$

maxSigma: *int*

maximum value of  $\sigma$

### Returns:

meta: *list of str*

Meta data for *features*, wherein strings indicate feature of each ndarray in *features*. e.g. 'Gaussian\_blur\_1' wherein the integer indicates  $\sigma$ .

features: *list of ndarray*

List containing feature images each of shape(*img*)

**Hessian**(img, minSigma = 1, maxSigma = 16)\*\*: runs a Hessian filter (sklearn.filters.hessian). Gaussian blurs with  $\sigma$  varying as usual are performed prior to the filter.

**Parameters:**

img: *ndarray*  
input image  
minSigma: *int*  
minimum value of  $\sigma$   
maxSigma: *int*  
maximum value of  $\sigma$

**Returns:**

meta: *list of str*  
Meta data for *features*, wherein strings indicate feature of each ndarray in *features*. e.g. 'Hessian\_1.0' wherein the integer indicates  $\sigma$ .  
features: *list of ndarray*  
List containing feature images each of shape(*img*)

**Laplace**(img, minsigma = 1, maxsigma = 16): applies skimage.filters.laplace() to each image after a Gaussian blur with  $\sigma$  varying as usual performed prior to the filter.

**Parameters:**

img: *ndarray*  
input image  
minSigma: *int*  
minimum value of  $\sigma$   
maxSigma: *int*  
maximum value of  $\sigma$

**Returns:**

meta: *list of str*  
Meta data for *features*, wherein strings indicate feature of each ndarray in *features*. e.g. 'Median\_1' wherein the integer indicates  $\sigma$ .  
features: *list of ndarray*  
List containing feature images each of shape(*img*)

**Neighbors**(img, minSigma = 1, maxSigma = 16)\*\*: shifts the image in 8 directions by a certain number of pixels,  $\sigma$ . Creates 8n feature images where  $(2^{n-1}) * \text{minSigma} \leq \text{maxSigma}$ .

**Parameters:**

img: *ndarray*  
input image  
minSigma: *int*  
minimum value of  $\sigma$

maxSigma: *int*  
maximum value of  $\sigma$

**Returns:**

meta: *list of str*

Meta data for *features*, wherein strings indicate feature of each ndarray in *features*. e.g. 'Neighbors\_1\_0' wherein the integers indicate  $\sigma$  and direction respectively.

features: *list of ndarray*

List containing feature images each of shape(*img*)

**Maximum**(*img*, minsigma = 1, maxsigma = 16): gets the maximum of each pixel and surrounding pixels distance  $\sigma$  with the normal  $n$  variations of  $\sigma$ .

**Parameters:**

*img*: *ndarray*

input image

minSigma: *int*

minimum value of  $\sigma$

maxSigma: *int*

maximum value of  $\sigma$

**Returns:**

meta: *list of str*

Meta data for *features*, wherein strings indicate feature of each ndarray in *features*. e.g. 'Maximum\_1' wherein the integer indicates  $\sigma$ .

features: *list of ndarray*

List containing feature images each of shape(*img*)

**Mean**(*img*, minsigma = 1, maxsigma = 16): gets the mean of each pixel and surrounding pixels distance  $\sigma$  with the normal  $n$  variations of  $\sigma$ .

**Parameters:**

*img*: *ndarray*

input image

minSigma: *int*

minimum value of  $\sigma$

maxSigma: *int*

maximum value of  $\sigma$

**Returns:**

meta: *list of str*

Meta data for *features*, wherein strings indicate feature of each ndarray in *features*. e.g. 'Mean\_1' wherein the integer indicates  $\sigma$ .

features: *list of ndarray*

List containing feature images each of shape(*img*)

**Median**(img, minsigma = 1, maxsigma = 16): gets the median of each pixel and surrounding pixels distance  $\sigma$  with the normal  $n$  variations of  $\sigma$ .

**Parameters:**

img: *ndarray*  
input image  
minSigma: *int*  
minimum value of  $\sigma$   
maxSigma: *int*  
maximum value of  $\sigma$

**Returns:**

meta: *list of str*  
Meta data for *features*, wherein strings indicate feature of each ndarray in *features*. e.g. 'Median\_1' wherein the integer indicates  $\sigma$ .  
features: *list of ndarray*  
List containing feature images each of shape(*img*)

**Median\_blur**(img, minsigma = 1, maxsigma = 16): applies skimage.filters.median() to each image after a Gaussian blur with  $\sigma$  varying as usual performed prior to the filter.

**Parameters:**

img: *ndarray*  
input image  
minSigma: *int*  
minimum value of  $\sigma$   
maxSigma: *int*  
maximum value of  $\sigma$

**Returns:**

meta: *list of str*  
Meta data for *features*, wherein strings indicate feature of each ndarray in *features*. e.g. 'Median\_1' wherein the integer indicates  $\sigma$ .  
features: *list of ndarray*  
List containing feature images each of shape(*img*)

**Meijering\_filter**(img, minSigma = 1, maxSigma = 16): applies Meijering neuriteness filter on  $n$  images, with Gaussian blurs with  $\sigma$  varying as usual are performed prior to the filter.

**Parameters:**

img: *ndarray*  
input image  
minSigma: *int*  
minimum value of  $\sigma$   
maxSigma: *int*



maximum value of  $\sigma$

**Returns:**

meta: *list of str*

Meta data for *features*, wherein strings indicate feature of each ndarray in *features*. e.g. 'Meijering\_filter\_1' wherein the integer indicates  $\sigma$ .

features: *list of ndarray*

List containing feature images each of shape(*img*)

**Membrane\_projections**(*img*, nAngles = 30, patchSize = 19, membraneSize = 1)\*\*:

enhances membrane-like structures of the image through directional filtering. The initial kernel for this operation is patchsize<sup>2</sup> zero matrix with the membraneSize number of middle column entries set to 1. Multiple kernels are created by rotating the original kernel 180 degrees/nAngles for a number of kernels = nAngles. Each kernel is convolved with the image and then the set of images are Z-projected into a single image via 6 methods:

- sum of the pixels in each image
- mean of the pixels in each image
- standard deviation of the pixels in each image
- median of the pixels in each image
- maximum of the pixels in each image
- minimum of the pixels in each image

Each of the 6 resulting images is a feature. Hence pixels in lines of similarly valued pixels in the image that are different from the average image intensity will stand out in the Z-projections.

**Parameters:**

*img*: *ndarray*

input image

nAngles: *int*

number of kernels

patchSize: *int*

size of matrix (patchSize x patchSize) to convolve image

membraneSize: *int*

number of columns in middle of patch to set to one before rotating and convolving

**Returns:**

meta: *list of str*

Meta data for *features*, wherein strings indicate feature of each ndarray in *features*. e.g. 'Membrane\_projections\_0' wherein the integer indicates the method.

features: *list of ndarray*

List containing feature images each of shape(*img*)

**Minimum**(*img*, minsigma = 1, maxsigma = 16): gets the minimum of each pixel and surrounding pixels distance  $\sigma$  with the normal  $n$  variations of  $\sigma$ .

**Parameters:**

img: *ndarray*  
input image  
minSigma: *int*  
minimum value of  $\sigma$   
maxSigma: *int*  
maximum value of  $\sigma$

**Returns:**

meta: *list of str*  
Meta data for *features*, wherein strings indicate feature of each ndarray in *features*. e.g. 'Minimum\_1' wherein the integer indicates  $\sigma$ .  
features: *list of ndarray*  
List containing feature images each of shape(*img*)

**Sklearn\_basic**(img): runs sklearn.feature.multiscale\_basic\_features(img) to image and returns results.

**Parameters:**

img: *ndarray*  
input image

**Returns:**

meta: *list of str*  
Meta data for *features*. Repeating list of strings 'Sklearn\_basic' with length of features  
features: *list of ndarray*  
List containing feature images each of shape(*img*)

**Sobel\_filter**(img, minSigma = 1, maxSigma = 16)\*\*: calculates an approximation of the gradient of the image intensity at each pixel using ndimage.sobel. Gaussian blurs with  $\sigma$  varying as usual are performed prior to the filter.

**Parameters:**

img: *ndarray*  
input image  
minSigma: *int*  
minimum value of  $\sigma$   
maxSigma: *int*  
maximum value of  $\sigma$

**Returns:**

meta: *list of str*  
Meta data for *features*, wherein strings indicate feature of each ndarray in *features*. e.g. 'Sobel\_filter\_1' wherein the integer indicates  $\sigma$ .  
features: *list of ndarray*

List containing feature images each of shape(*img*)

**Watershed\_distance**(*img*, threshmethod = filters.threshold\_yen): thresholds the image with the indicated threshold method and transforms the image based on distances between objects (see Distances output in example: [https://scikit-image.org/docs/stable/auto\\_examples/segmentation/plot\\_watershed.html](https://scikit-image.org/docs/stable/auto_examples/segmentation/plot_watershed.html)). Default threshmethod is skimage.filters.threshold\_yen. Other threshold methods can be applied from skimage.filters

**Parameters:**

*img*: *ndarray*

input image

*threshmethod*: *function*

select threshold method from skimage.filters. Default is skimage.filters.threshold\_yen.

**Returns:**

*meta*: *list of str*

Meta data for *features*, ['Watershed\_distance']

*features*: *list of ndarray*

List containing feature image of shape(*img*)

**Variance**(*img*, minsigma = 1, maxsigma = 16): gets the variance of each pixel and surrounding pixels distance  $\sigma$  with the normal  $n$  variations of  $\sigma$ .

**Parameters:**

*img*: *ndarray*

input image

*minSigma*: *int*

minimum value of  $\sigma$

*maxSigma*: *int*

maximum value of  $\sigma$

**Returns:**

*meta*: *list of str*

Meta data for *features*, wherein strings indicate feature of each ndarray in *features*. e.g. 'Variance\_1' wherein the integer indicates  $\sigma$ .

*features*: *list of ndarray*

List containing feature images each of shape(*img*)