edxia Documentation

Release 0.0.1

Fabien Georget

May 27, 2021

Contents:

Warning: The documentation is under construction !

Tip: A step-by-step tutorial is available in a separate documentation: edxia through the Glueviz interface. This is the recommended document for non-technical users !

CHAPTER 1

About

1.1 edxia

edxia is an image analysis framework for EDS/BSE hyperspectral maps. It is available as an API to develop your own tool adapte to your problems or as a plugin to Glueviz to be available to any researcher.

The code is developed by Fabien Georget in the Laboratory of Construction Materials, at EPFL, Lausanne, Switzerland. This work is a collaboration with William Wilson (EPFL, University of Sherbrooke) and Karen Scrivener (EPFL).

1.2 Quickstart

A step-by-step tutorial is available in a separate documentation: edxia through the Glueviz interface.

1.3 Citing

If you use this work, please cite the following:

- Fabien Georget, William Wilson and Karen Scrivener, edxia. Zenodo. http://doi.org/10.5281/zenodo.3246902
- Fabien Georget, William Wilson and Karen Scrivener, Comprehensive microstructure characterization from quantified SEM-EDS maps in cementitious materials (in preparation)

CHAPTER 2

Install

2.1 Requirements

edxia requires

- Python 3 (tested with python versions 3.6 and 3.7).
- scipy/numpy/matplotlib
- pandas
- Glue (version >0.15)
- py_expression_eval..

Note: The Anaconda distribution might be used to install these dependances.

2.2 Installation

2.2.1 Installation with conda

With a valid anaconda distribution installation (anaconda or miniconda), the *edxia* package can be installed with the requirements using the conda command

```
conda install -c specmicp edxia
```

To update a previous installation, the following command can be executed:

```
conda update -c specmicp edxia
```

2.2.2 Installation with pip

The *edxia* package is uploaded to Pypi, the Python Package Index. Therefore it is possible to install the latest version with its requirements very easily in any valid python installation:

```
pip install edxia
```

By default, this command attempt a system-wide installation. The package can be installed for the user only:

```
pip install edxia --user
```

To update a previous installation, the following command can be executed:

```
pip install edxia --upgrade
```

2.2.3 Installation from sources

The package can be installed directly from the sources. The latest version can be dowloaded directly from the git repository:

```
git clone https://bitbucket.org/specmicp/edxia.git
```

It can then be installed with the following commands

```
cd edxia
python setup.py install
```

CHAPTER 3

Input files

edxia requires raw maps in text format from the microscope/EDS software. The text format is required to easily accept data from various acquisition software.

The text format must be one of the various csv-type formats. The exact delimiter, and normalization to apply are described by *edxia.io.raw_io.TextMapFormat*. Several predefined format are available in this module (*edxia.io.raw_io*).

All maps from a a set must be in the same folder. Using the default classes, the user only select the BSE map, and the program find the other maps available based on the filenames. For example, the following is a valid set of names for the maps:

- cement/durability/slag_28d_BSE.txt
- cement/durability/slag_28d_Ca.txt
- cement/durability/slag_28d_Al.txt
- cement/durability/slag_28d_Si.txt

The corresponding pattern would be

- cement/durability/slag_28d_component}.txt

The choice of this pattern was made to avoid selecting every single patterns. However, it means that the name of the files need to be consistent so they can be recognized.

This is the default in :mod:edxia: but the user can defines it's own loading functions by specializing *edxia.io.loader.base_loader.AbstractLoader*.

CHAPTER 4

Main concepts

The main concept between edxia is the extraction of representative points from the map. The representative points are extracted following these steps:

1. Loading the individual maps
2. Creation of a composite map
3. Segmentation of the composite maps, and extraction of representative points
4. Analysis of the representative points

4.1 Individual maps

The individual maps are loaded from the text-formatted quantified maps as provided by the microscope software. The details are described in: *Input files*. The maps are then filtered according the parameters provided by the user.

4.2 The composite map

The individual maps are assigned a color as shown in the figure of *BSE/EDS components*. These maps are then assigned to the channels (Red, Blue and Green) of a RGB image (see for example the Scikit-Image documentation). Since only 3 color channels are available, 3 main components must be chosen. The following choice is quite common:

- **Red**: Si
- **Green**: Al
- **Blue**: Ca

An additional component (such as the BSE) can be used as an offset on all channels, or as a transparent overlay. A *composite image* is obtained, the colors represent the mixture of phases.

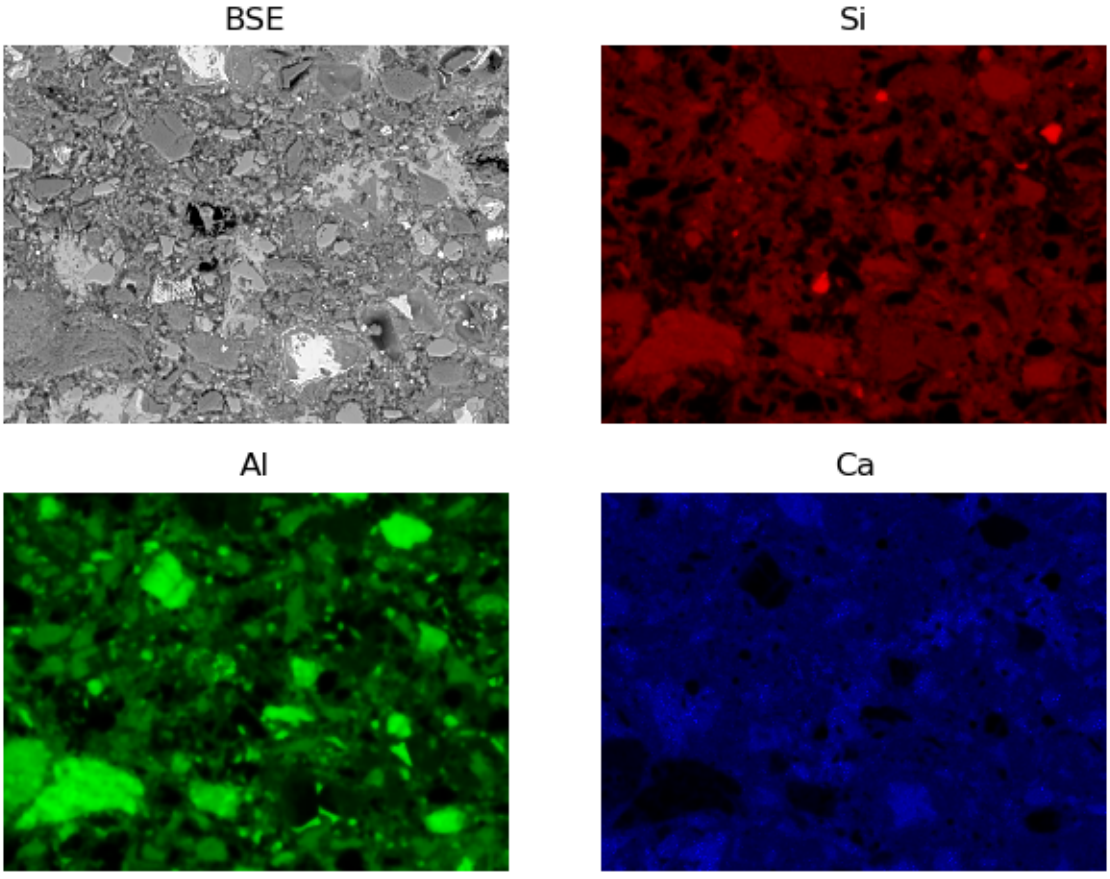


Fig. 1: Individual component BSE/EDS maps

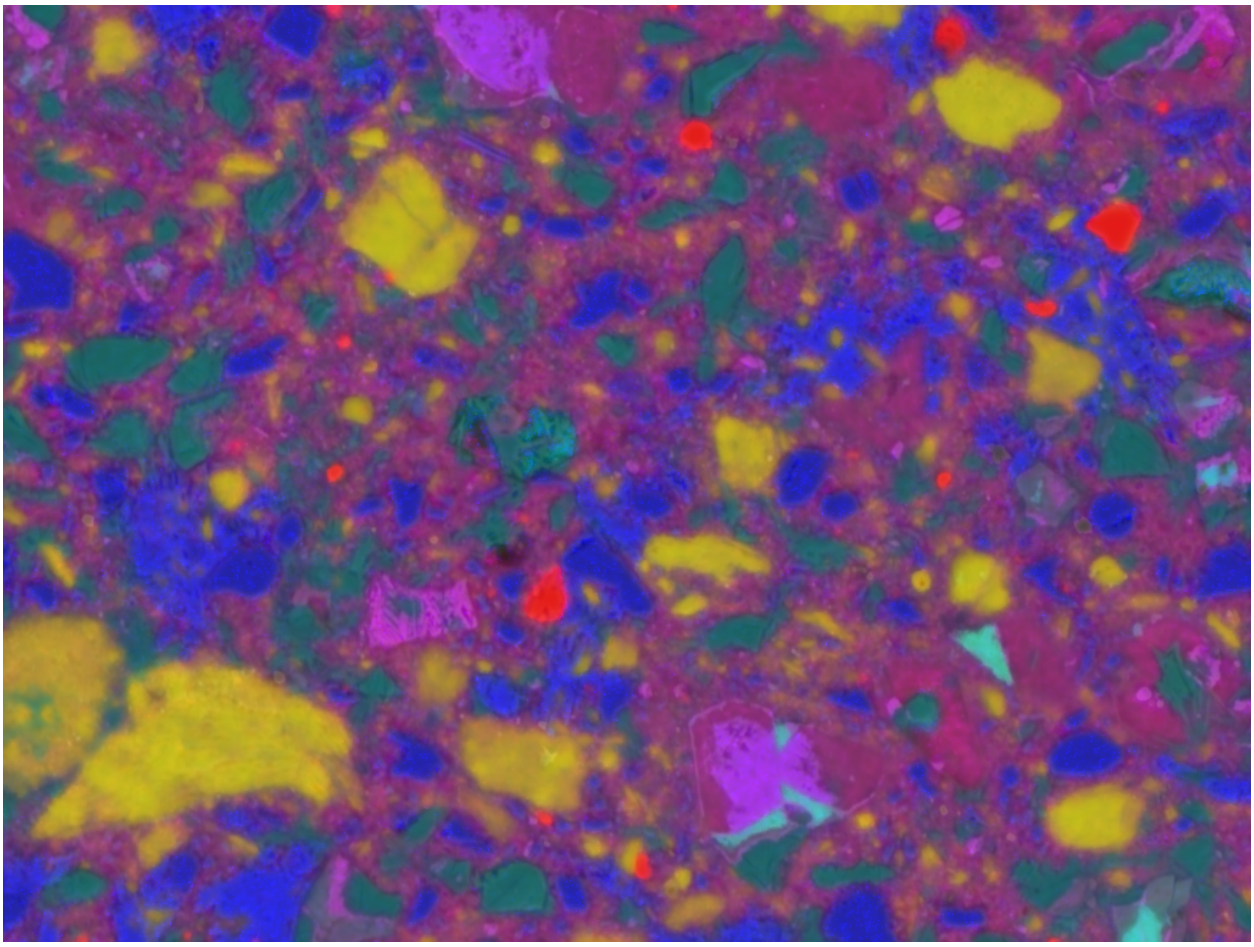


Fig. 2: The composite image corresponding to the individual *BSE/EDS components*.

The colors corresponds to the following phases:

- Blue: Portlandite/Calcite
- Purple CSH/C2S/C3S
- Green: AFm/C3A
- Red: Quartz
- Cyan: C4AF
- Yellow: Metakaolin

The *composite image* can be used to quickly identify the phases present in the maps. Additional thresholding on the hue can provides a first estimation of the volume fraction of these phases. However, this method is limited because only 3 or 4 independant components can be analysed.

4.3 Segmentation

The *composite image* is interesting because it allows to quickly define the region of similar composition in the map. The pixels in the region of similar composition can be grouped to define "particles" or agglomeration of particles. The gathering of pixels into larger region is the purpose of a segmentation algorithm. Many segmentation algorithms exist as seen on this segmentation example from the Scikit-image documentation. By default in `edxia`, the [SLIC] algorithm is used.

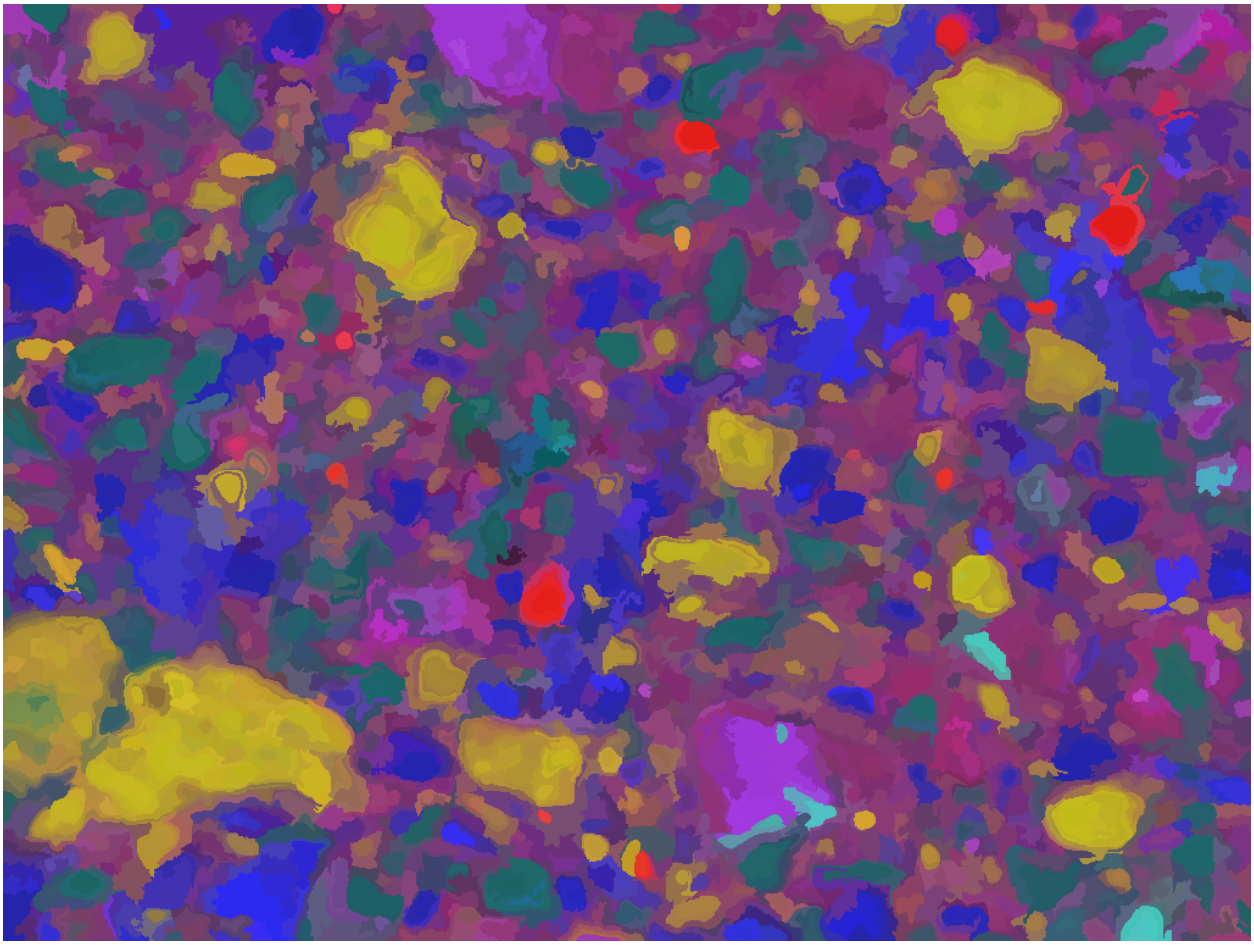
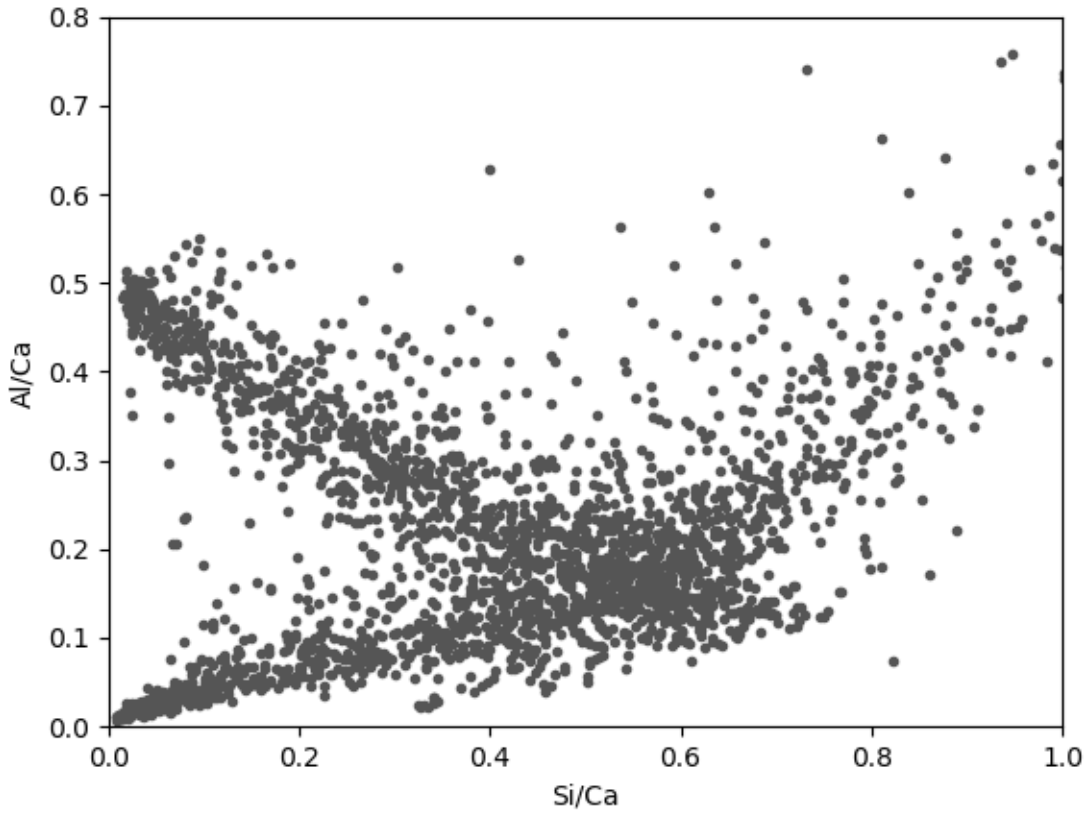


Fig. 3: The segmented *composite image* using the [SLIC] algorithm.

Each region defined by the segmentation algorithm is assumed to be relatively uniform. The most uniform point in the region is the point at the center, where the mixing effects are the lowest. Therefore, the geometrical center of each center is used as a representative point of the maps. The composition of these points are assembled into a dataset which can be further analysed.

4.4 Ratio plots

A simple analysis tools for EDS composition are ratio plots. These ratio plots are scatter plots where the axis are ratios of elemental composition (e.g. Si/Ca, Al/Ca, S/Ca, ...). The position of the pure phases can be easily identify on this plots. The points around these points acan be identified as being the respective phase. Mixture lines can be defined between the position of two phases. Points along these lines corresponds to a mix of these two phases. From this analysis, a researcher can detect the existence of a phase, or find the average composition of a variable phase (e.g C-S-H).



The main drawbacks of these ratio plots is that they do not display spatial information. As such, an additional step is required to understand the distribution of phases on the maps. This method was implemented as a plugin to the Glueviz software to answer this problem. This is described in the next section of the documentation: *edxia/Glue*.

CHAPTER 5

edxia/Glue

5.1 Glue

Glue is a "Python library to explore relationships within and between related datasets". The Glue documentation is a good place to start to learn about this software.

5.2 edxia plugin

In the Glue interface, `edxia` is available as a plugin. The goal of the plugin is to load the dataset and provide some tools to analyse the composition and the distributions of the phases. The separation of the phases, and the clustering of points is done by the user with the selection tools offered by Glue.

This process is the recommended way for new user to use `edxia`. A step-by-step process is provided in the corresponding documentation ..

CHAPTER 6

API

This sections contains the documentation of the API

6.1 Core module

The core modules contains the core classes used to manipulate Maps.

6.1.1 Experiment

This module contains the utilities to describe an experiment. The main class is *edxia.core.experiment.MappingExperiment*.

class edxia.core.experiment.**MappingExperiment**(*pattern*, *label=None*, *description=None*, *map_format=None*, *bse_format=None*, *map_scale_factor=1*, *components=None*)

This class contains the main information describing an experiment, such as the filepaths, the map formats, the list of components... This class is shared between many other classes.

Example:

exp = MappingExperiment("mymap_{components}.csv", label="OPC 28d")

Parameters

- **pattern** (*str*) – the pattern of the file paths
- **label** (*str or None*) – short, unique label for this set of map
- **description** (*str or None*) – optional, longer description
- **map_format** (edxia.io.raw_io.TextMapFormat) – the file format of the raw EDS maps.

- **bse_format** (edxia.io.raw_io.TextMapFormat) – the file format of the raw EDS maps.
- **components** (*list[str]*) – list of available components for this experiment

bse_format

Returns the format of the raw BSE map

Return type *edxia.io.raw_io.TextMapFormat*

get_path_map(*component*)

Returns the filepath to the raw map of 'component'

Return type str

Raises **RuntimeError** – if the component is not valid

is_valid()

Check if the experiment describes a valid set of maps.

Returns True if the experiment is valid, and the exception otherwise

Return type tuple(bool, Exception or None)

label

Returns an identifying label for this experiment

Return type str

list_components

Returns a list of available components

Return type list[str]

load_raw_map(*component*)

Load the map for a component.

This is a low-level function, a loader (*edxia.io.loader*) should be used in normal operations

Returns a map

Return type a numpy array

map_format

Returns the format of the EDS maps

Return type *edxia.io.raw_io.TextMapFormat*

pattern

Returns the pattern of filepaths to the maps

Return type str

class edxia.core.experiment.**PartOfExperiment**(*parent=None*)

Base class for a subset of an experiment.

Classes that should have access to the information provided by edxia.core.experiment.Experiment should inherit from this class.

parent

Returns the experiment from which this effect was created

Return type edxia.core.experiment.Experiment

class edxia.core.experiment.**PointsExperiment**(*components*, *label=None*, *description=None*)

An experiment describing a EDS point experiment

is_valid()

Check if the experiment describes a valid set of maps.

Returns True if the experiment is valid, and the exception otherwise

Return type tuple(bool, Exception or None)

label

Returns an identifying label for this experiment

Return type str

list_components

Returns a list of available components

Return type list[str]

6.1.2 Map

class edxia.core.map.**Map**(*component*, *eds_map*, *parent*)

A 2D EDS Map

component

The map component

flat_map

Return the map as a 1D array.

map

The map

nb_cols

The number of columns

nb_rows

The number of rows

rc(*r*, *c*)

Return the value of the map in the rc coordinates system.

shape

The shape of the map

xy(*x*, *y*)

Return the value of the map in the xy coordinates system.

class edxia.core.map.**MapsStack**(*components*, *shape*, *parent*)

A stack of maps

components

Returns the list of components.

composition(*r*, *c*)

Return the composition at a point

index(*component*)
Returns the index of the component.

map(*component*)
Return the map for a component.

maps
Returns the numpy array containing the maps

normalize(*copy=True*)
Normalize the EDS maps, so their sum at each point is 1.

shape
Return the shape of the maps array.

sum_of_oxides_from_mass()
Compute the sum of oxides from a mass map.

to_atomic(*copy=True*)
Transform the maps to atomic

6.2 Composite module

6.2.1 Channels

Channels information for a composite map.

class edxia.composite.channels.**CompositeChannels**(*components*, *factors*)
Channels information for a composite map.

blue
The component of the blue channel.

blue_factor
The scaling factor for the blue channel.

gray
The component of the gray channel.

gray_factor
The scaling factor for the gray channel.

green
The component of the green channel.

green_factor
The scaling factor for the green channel.

nb_components
The number of components.

red
The component of the red channel.

red_factor
The scaling factor for the red channel.

6.2.2 Composite

class edxia.composite.composite.**CompositeMap**(*composite_img*, *channels*, *parent*)
A composite map

channels
Return the channels use to build this composite image.

6.2.3 Segmentation

class edxia.composite.segmentation.**FullSlicSegmenter**(*compactness*, *nb_segments*)
Slic segmenter not restricted to RGB space

class edxia.composite.segmentation.**SegmentedLabels**(*labels*, *composite_parent*)
The labels issued from segmentation.

composite
Return the composite image

get_center_labels()
Return the centroids of each segmented part.

get_color_labels(*sampler_points=None*, *color_img=None*)
Return the color for the labels.

nb_labels
Return the number of labels.

class edxia.composite.segmentation.**Segmenter**
Abstract base class for a segmenter

apply_impl()
Return the raw labels

class edxia.composite.segmentation.**SlicSegmenter**(*compactness*, *nb_segments*, ***kwargs*)
The slic segmenter

apply_impl(*composite*)
Return the raw labels

class edxia.composite.segmentation.**WatershedSegmenter**(*markers*, ***kwargs*)
The watershed segmenter

apply_impl(*composite*)
Return the raw labels

6.3 IO module

This module provides the input/output tools to read/save maps and datasets.

It is made of 3 main modules:

- *edxia.io.loader*: Load set of maps
- *edxia.io.raw_io*: Load and save individual maps
- *edxia.io.hdf5*: Load and save HDF5 datasets

6.3.1 Loaders

The *edxia.io.loader* module offers loader classes to be able to easily load component maps, composite or map stacks.

Three main loaders are defined:

- *edxia.io.loader.DefaultLoader*: A simple loader, applies the filters every time
- *edxia.io.loader.PickleLoader*: A loader with a cache, apply the filters only once
- *edxia.io.loader.StackLoader*: A loader not working from the filesystem, but using a *edxia.core.map.MapsStack* as an input data.

The main functions are:

- *edxia.io.loader.base_loader.AbstractLoader.load_edsmap()*: load a BSE/EDS map, *edxia.core.map.Map*.
- *edxia.io.loader.base_loader.AbstractLoader.load_stack()*: load a set of maps, *edxia.core.map.MapsStack*
- *edxia.io.loader.base_loader.AbstractLoader.load_composite()*: load a composite map, `edxia.composite.CompositeMap`

class edxia.io.loader.**DefaultLoader**(*exp_manager*, *filters=None*, *no_BSE_filter=True*)
Bases: *edxia.io.loader.base_loader.AbstractLoader*

Default loader without loading.

load_edsmap_single(*component*)
Load the map of a 'component'.

class edxia.io.loader.**PickleLoader**(*exp_manager*, *filters=None*, *no_BSE_filter=True*, *remove_previous=True*)
Bases: *edxia.io.loader.base_loader.AbstractLoader*

This loader cache filtered map so long filtering can be done only once.

denoise_and_renormalise()
Denoise and renormalize all maps.

This can be used to correctly normalize the maps after filtering. Might be useful, or not, depending on the quality of the map.

get_path_picklemap(*component*)
Return the path to the pickle map of 'component'.

load_edsmap_single(*component*)
Load a map.

Search the cache first if such a map exist.

remove_npy_files()
Remove existing pickle files.

reset_filters(*filters*)
Reset the filters

class edxia.io.loader.**StackLoader**(*stack*)
Bases: *edxia.io.loader.base_loader.AbstractLoader*

Loader using a set of maps already read (and optionally filtered) using a different loader.

load_edsmap_single(*component*)
Load the map of a 'component'.

load_stack(*components=None*, *factors=None*)
Copy and return the stack of maps of the loader.

class edxia.io.loader.base_loader.**AbstractLoader**(*exp_manager*, *filters=None*)
Abstract base class for a map loader. This class is not intended for use, instead the user should use one of the class in *edxia.io.loader*.

add_filter(*afilter*)
Add a filter to the list of filters to apply at loading.

exp_manager
Return the experience manager

filters
Return the list of filters to apply at loading.

get_path_map(*component*)
Return the path to a component map.

load_composite(*channels*, *is_rgb=True*)
Load a composite map

Parameters

- **channels** – the different channels to consider
- **is_rgb** – if true, the map is reduced to an RGB image

load_composite_rgb(*channels*)
Load a composite map.

load_edsmap(*component*)
Load the map of 'component'. It can be either a simple component (e.g. 'BSE', 'Ca'), or a combination of components (e.g. "Si+Al/Ca").

load_edsmap_complex(*expr*)
Load an expresion map.

load_edsmap_single(*component*)
Load the map of a 'component'.

load_stack(*components=None*, *factors=None*)
Load a stack of maps

Parameters

- **components** – the list of components, if not provided it is the entire list of components available
- **factors** – the list of factors, default to 1

reset_filters(*filters*)
Reset the list of filters.

6.3.2 Raw IO

This module provides raw loading and saving helper functions. It defines how the maps are read before any transformation from the algorithm.

Example: Read txt format from bruker Esprit software:

```
bse_map_asarray = load_txt_map("map234_BSE.txt", esprit_ascii_map_format)
```

class edxia.io.raw_io.**TextMapFormat**(*delimiter*, *min_value*, *max_value*, *saveformat='%.4f'*)
Bases: object

A struct-like class to contain formatting information about raw EDS maps.

copy()
Copy the format

Returns a copy of the text map format

delimiter
Return the column delimiter

escaped_delimiter
Return an escaped version of the delimiter

max_value
Return the maximum value allowed in the map

min_value
Return the minimum value allowed in the map

saveformat
Set the save format

edxia.io.raw_io.**esprit_ascii_map_format = TextMapFormat(';',0,100)**
Default EDS map format for the Bruker Esprit software

edxia.io.raw_io.**esprit_ascii_bse_format = TextMapFormat(';',0,-1)**
Default BSE map format for the Bruker Esprit software

edxia.io.raw_io.**aztec_ascii_map_format = TextMapFormat(',',0,100)**
Default EDS map format for the Oxford Aztec software

edxia.io.raw_io.**aztec_ascii_bse_format = TextMapFormat(',',0,-1)**
Default BSE map format for the Oxford Aztec software

edxia.io.raw_io.**imagej_ascii_bse_format = TextMapFormat('\t',0,-1)**
Format for a text image produced by ImageJ

edxia.io.raw_io.**load_txt_map**(*filename*, *txtformat*, *resize=1*)
Load a 2D map

Parameters

- **filename** – filepath to the text map
- **txtformat** – the format of the text map
- **resize** – Stretch the map in each axis)

Returns a 2D numpy array normalized between [0,1]

Raises whatever numpy.loadtxt can raise

edxia.io.raw_io.**save_txt_map**(*filename*, *amap*, *txtformat*)
Save a 2D map

Perform a copy first to normalize the data according the format.

Parameters

- **filename** – filepath to the file
- **amap** – a 2D map

Format txtformat the format of the ascii saved map

edxia.io.raw_io.**load_pickle_map**(*filepath*)
Load a map that was previously saved as a pickle dump

edxia.io.raw_io.**save_pickle_map**(*filepath*, *data*)
Dump a map in a binary format

6.3.3 HDF5

edxia.io.hdf5.**read_composite**(*hf5file*, *exp*)
Read a composite object from an open hdf5 file

edxia.io.hdf5.**read_dataset**(*filepath*)
Read a set of edxia data from a hdf5 file

edxia.io.hdf5.**read_experiment**(*h5file*)
Read an experiment from an open hdf5 file

edxia.io.hdf5.**read_points**(*hf5file*, *exp*)
Read a points object from an open hdf5 file

edxia.io.hdf5.**read_stack**(*hf5file*, *exp*)
Read a stack object from an open hdf5 file

edxia.io.hdf5.**save_dataset**(*filepath*, *exp*, *stack=None*, *composite=None*, *points=None*, *extras=None*, *override=True*)
This function save a set of treated data into a hdf5 file.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Bibliography

[SLIC] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Suesstrunk, SLIC Superpixels Compared to State-of-the-art Superpixel Methods, TPAMI, May 2012.

Python Module Index

e

Index

S

T

W

X