histolab

Release 0.4.0

histolab

Apr 16, 2022

CONTENTS

The aim of this project is to provide a tool for WSI processing in a reproducible environment to support clinical and scientific research. histolab is designed to handle WSIs, automatically detect the tissue, and retrieve informative tiles, and it can thus be integrated in a deep learning pipeline.

The histo-pathological analysis of tissue sections is the gold standard to assess the presence of many complex diseases, such as tumors, and understand their nature.

In daily practice, pathologists usually perform microscopy examination of tissue slides considering a limited number of regions and the clinical evaulation relies on several factors such as nuclei morphology, cell distribution, and color (staining): this process is time consuming, could lead to information loss, and suffers from inter-observer variability.

The advent of digital pathology is changing the way patholgists work and collaborate, and has opened the way to a new era in computational pathology. In particular, histopathology is expected to be at the center of the AI revolution in medicine [1], prevision supported by the increasing success of deep learning applications to digital pathology.

Whole Slide Images (WSIs), namely the translation of tissue slides from glass to digital format, are a great source of information from both a medical and a computational point of view. WSIs can be coloured with different staining techniques (e.g. H&E or IHC), and are usually very large in size (up to several GB per slide). Because of WSIs typical pyramidal structure, images can be retrieved at different magnification factors, providing a further layer of information beyond color.

However, processing WSIs is far from being trivial. First of all, WSIs can be stored in different proprietary formats, according to the scanner used to digitalize the slides, and a standard protocol is still missing. WSIs can also present artifacts, such as shadows, mold, or annotations (pen marks) that are not useful. Moreover, giving their dimensions, it is not possible to process a WSI all at once, or, for example, to feed a neural network: it is necessary to crop smaller regions of tissues (tiles), which in turns require a tissue detection step.

CHAPTER ONE

INSTALLATION

The histolab package can be installed by using:

```
pip install histolab
```

CHAPTER TWO

PREREQUISITES

Please see installation instructions.

CHAPTER THREE

AUTHORS

- Alessia Marcolini
- Ernesto Arbitrio
- Nicole Bussola

CHAPTER

FOUR

LICENSE

This project is licensed under *Apache License Version 2.0* - see the LICENSE.txt file for details.

CHAPTER FIVE

ACKNOWLEDGEMENTS

- https://github.com/deroneriksson

CHAPTER

SIX

REFERENCES

[1] Colling, Richard, et al. “Artificial intelligence in digital pathology: A roadmap to routine use in clinical practice.” The Journal of pathology 249.2 (2019)

6.1 Overview

The aim of this project is to provide a tool for WSI processing in a reproducible environment to support clinical and scientific research. histolab is designed to handle WSIs, automatically detect the tissue, and retrieve informative tiles, and it can thus be integrated in a deep learning pipeline.

6.1.1 Motivation

The histo-pathological analysis of tissue sections is the gold standard to assess the presence of many complex diseases, such as tumors, and understand their nature.

In daily practice, pathologists usually perform microscopy examination of tissue slides considering a limited number of regions and the clinical evaluation relies on several factors such as nuclei morphology, cell distribution, and color (staining): this process is time consuming, could lead to information loss, and suffers from inter-observer variability.

The advent of digital pathology is changing the way pathologists work and collaborate, and has opened the way to a new era in computational pathology. In particular, histopathology is expected to be at the center of the AI revolution in medicine [1], prevision supported by the increasing success of deep learning applications to digital pathology.

Whole Slide Images (WSIs), namely the translation of tissue slides from glass to digital format, are a great source of information from both a medical and a computational point of view. WSIs can be coloured with different staining techniques (e.g. H&E or IHC), and are usually very large in size (up to several GB per slide). Because of WSIs typical pyramidal structure, images can be retrieved at different magnification factors, providing a further layer of information beyond color.

However, processing WSIs is far from being trivial. First of all, WSIs can be stored in different proprietary formats, according to the scanner used to digitalize the slides, and a standard protocol is still missing. WSIs can also present artifacts, such as shadows, mold, or annotations (pen marks) that are not useful. Moreover, giving their dimensions, it is not possible to process a WSI all at once, or, for example, to feed a neural network: it is necessary to crop smaller regions of tissues (tiles), which in turns require a tissue detection step.

6.1.2 Installation

The histolab package can be installed by using:

```
pip install histolab
```

6.1.3 Prerequisites

Please see installation instructions.

6.1.4 Authors

- Alessia Marcolini
- Ernesto Arbitrio
- Nicole Bussola

6.1.5 License

This project is licensed under *Apache License Version 2.0* - see the LICENSE.txt file for details.

6.1.6 Acknowledgements

- https://github.com/deroneriksson

6.1.7 References

[1] Colling, Richard, et al. "Artificial intelligence in digital pathology: A roadmap to routine use in clinical practice." The Journal of pathology 249.2 (2019)

6.2 Installation

histolab has only one system-wide dependency: `OpenSlide`.

You can download and install it from https://openslide.org/download/ according to your operating system.

Warning: There is a known bug in Pixman versions `0.38.*` that causes `OpenSlide` to produce images with black boxes for large images. See issue https://github.com/openslide/openslide/issues/291 for reference. Install version 0.40 (latest) depending on your operating system.

6.2.1 Install Pixman 0.40 on Ubuntu

Ubuntu 21.04

If you are running histolab on Ubuntu 21.04 you probably already have Pixman 0.40 and you are all set, go have fun

Ubuntu 20.04 LTS

If you are using a conda environment, it is sufficient to run:

```
$ conda install -c anaconda pixman==0.40
```

Otherwise, to force a working version of `libpixman` to be loaded before a bad version, you need to exploit the LD_PRELOAD mechanism on Linux. Make sure you have that file installed first.

```
$ export LD_PRELOAD=/path/of/libpixman-1.so.0.40.0:$LD_PRELOAD
```

If necessary, build `libpixman` from source. It's an easy build since it doesn't have any dependencies:

```
$ wget https://cairographics.org/releases/pixman-0.40.0.tar.gz
$ tar -xvf pixman-0.40.0.tar.gz
$ cd pixman-0.40.0
$ ./configure
$ make
$ sudo make install
```

6.2.2 Install Pixman 0.40 on macOS

If `OpenSlide` is installed via `brew`, pixman 0.40 will be automatically installed ✓

6.2.3 Install Pixman 0.40 on Windows

`OpenSlide` builds are the same for all Windows versions and they include pixman 0.34.

Pixman 0.40 can be retrieved using `pacman` (the package manager of Arch Linux, see https://www.msys2.org/ for more info):

```
$ pacman -S mingw-w64-x86_64-pixman
```

Once pixman 0.40 is installed you have to link the current version of the `dll` to the `OpenSlide` installation. The only thing to do is overwrite `libpixman-1-0.dll` in the `bin` directory of `OpenSlide` with the one installed with pixman 0.40 that should be placed in `/mingw64/bin/libpixman-1-0.dll`.

For example if `OpenSlide` is installed in `C:\` you should replace `C:\OpenSlide\bin\libpixman-1-0.dll` with `/mingw64/bin/libpixman-1-0.dll`.

6.2.4 Verify Correct Pixman installation

Ubuntu

```
$ ldconfig -v | grep libpixman
```

macOS

```
$ brew list --versions pixman
```

Windows (PowerShell)

```
$ (Get-Item "C:-1-0.dll").VersionInfo | format-list
```

6.3 Quick Start

Here we present a step-by-step tutorial on the use of `histolab` to extract a tile dataset from example WSIs. The corresponding Jupyter Notebook is available at https://github.com/histolab/histolab-box: this repository contains a complete `histolab` environment that can be used through Docker on all platforms.

Thus, the user can decide either to use `histolab` through `histolab-box` or installing it in his/her python virtual environment (using conda, pipenv, pyenv, virtualenv, etc...). In the latter case, as the `histolab` package has been published on PyPI, it can be easily installed via the command:

```
$ pip install histolab
```

6.3.1 TCGA data

First things first, let's import some data to work with, for example the prostate tissue slide and the ovarian tissue slide available in the `data` module:

```
from histolab.data import prostate_tissue, ovarian_tissue
```

Note: To use the `data` module, you need to install `pooch`, also available on PyPI (https://pypi.org/project/pooch/). This step is needless if we are using the Vagrant/Docker virtual environment.

The calling to a `data` function will automatically download the WSI from the corresponding repository and save the slide in a cached directory:

```
prostate_svs, prostate_path = prostate_tissue()
ovarian_svs, ovarian_path = ovarian_tissue()
```

Notice that each `data` function outputs the corresponding slide, as an OpenSlide object, and the path where the slide has been saved.

6.3.2 Slide initialization

histolab maps a WSI file into a Slide object. Each usage of a WSI requires a 1-o-1 association with a Slide object contained in the slide module:

```
from histolab.slide import Slide
```

To initialize a Slide it is necessary to specify the WSI path, and the processed_path where the tiles will be saved. In our example, we want the processed_path of each slide to be a subfolder of the current working directory:

```
import os

BASE_PATH = os.getcwd()

PROCESS_PATH_PROSTATE = os.path.join(BASE_PATH, 'prostate', 'processed')
PROCESS_PATH_OVARIAN = os.path.join(BASE_PATH, 'ovarian', 'processed')

prostate_slide = Slide(prostate_path, processed_path=PROCESS_PATH_PROSTATE)
ovarian_slide = Slide(ovarian_path, processed_path=PROCESS_PATH_OVARIAN)
```

Note: If the slides were stored in the same folder, this can be done directly on the whole dataset by using the SlideSet object of the slide module.

With a Slide object we can easily retrieve information about the slide, such as the slide name, the number of available levels, the dimensions at native magnification or at a specified level:

```
print(f"Slide name: {prostate_slide.name}")
print(f"Levels: {prostate_slide.levels}")
print(f"Dimensions at level 0: {prostate_slide.dimensions}")
print(f"Dimensions at level 1: {prostate_slide.level_dimensions(level=1)}")
print(f"Dimensions at level 2: {prostate_slide.level_dimensions(level=2)}")
```

```
Slide name: 6b725022-f1d5-4672-8c6c-de8140345210
Levels: [0, 1, 2]
Dimensions at level 0: (16000, 15316)
Dimensions at level 1: (4000, 3829)
Dimensions at level 2: (2000, 1914)
```

```
print(f"Slide name: {ovarian_slide.name}")
print(f"Levels: {ovarian_slide.levels}")
print(f"Dimensions at level 0: {ovarian_slide.dimensions}")
print(f"Dimensions at level 1: {ovarian_slide.level_dimensions(level=1)}")
print(f"Dimensions at level 2: {ovarian_slide.level_dimensions(level=2)}")
```

```
Slide name: b777ec99-2811-4aa4-9568-13f68e380c86
Levels: [0, 1, 2]
Dimensions at level 0: (30001, 33987)
Dimensions at level 1: (7500, 8496)
Dimensions at level 2: (1875, 2124)
```

Note: If the native magnification, *i.e.*, the magnification factor used to scan the slide, is provided in the slide properties, it is also possible to convert the desired level to its corresponding magnification factor with the `level_magnification_factor` property.

```
print(
    "Native magnification factor:",
    prostate_slide.level_magnification_factor()
)

print(
    "Magnification factor corresponding to level 1:",
    prostate_slide.level_magnification_factor(level=1),
)
```

```
Native magnification factor: 20X
Magnification factor corresponding to level 1: 5.0X
```

Moreover, we can retrieve or show the slide thumbnail in a separate window:

```
prostate_slide.thumbnail
prostate_slide.show()
```

```
ovarian_slide.thumbnail
ovarian_slide.show()
```

6.3.3 Tile extraction

Once that the `Slide` objects are defined, we can proceed to extract the tiles. To speed up the extraction process, `histolab` automatically detects the tissue region with the largest connected area and crops the tiles within this field. The `tiler` module implements different strategies for the tiles extraction and provides an intuitive interface to easily retrieve a tile dataset suitable for our task. In particular, each extraction method is customizable with several common parameters:

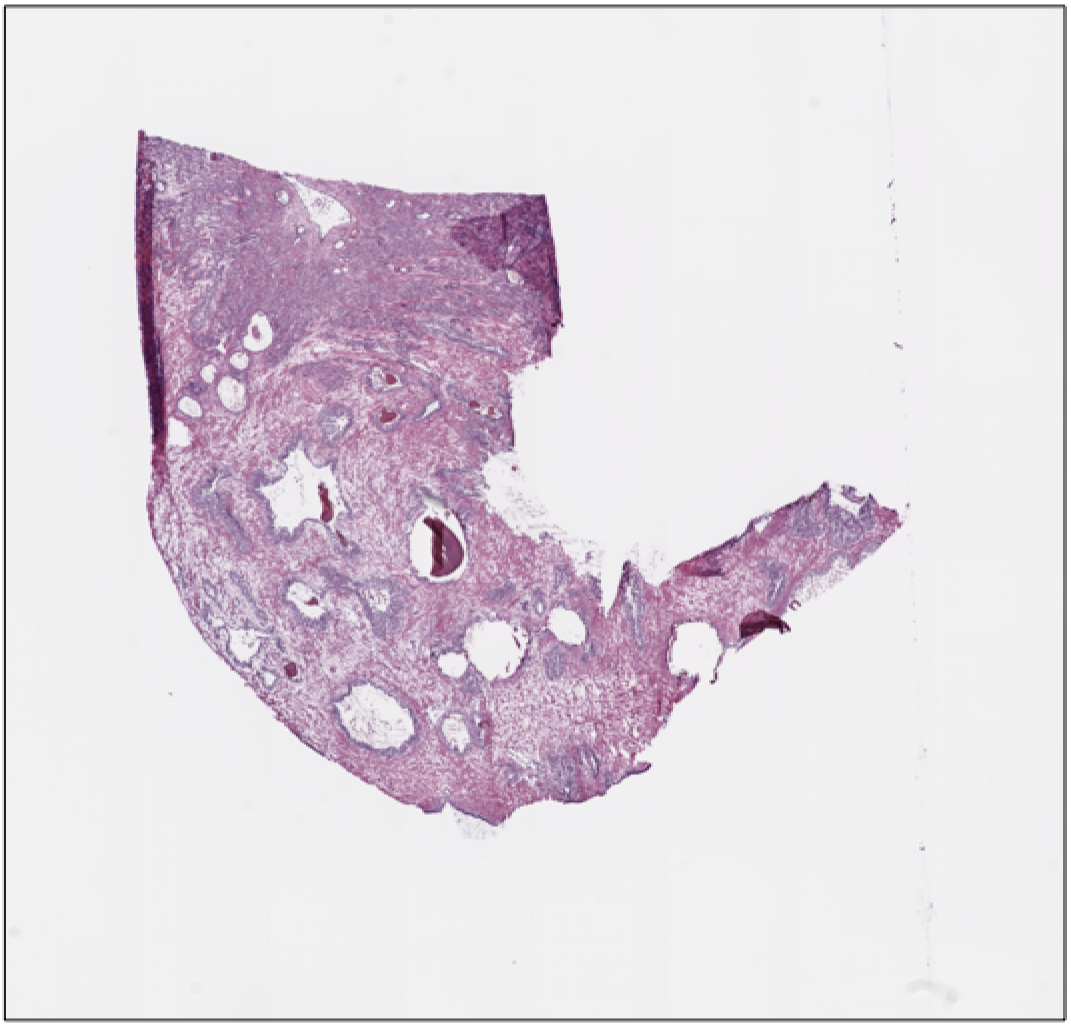
- `tile_size`: the tile size;
- `level`: the extraction level (from 0 to the number of available levels);
- `check_tissue`: if a minimum percentage of tissue is required to save the tiles;
- `` tissue_percent``: number between 0.0 and 100.0 representing the minimum required percentage of tissue over the total area of the image (default is 80.0)
- `prefix`: a prefix to be added at the beginning of the tiles' filename (default is the empty string);
- `suffix`: a suffix to be added to the end of the tiles' filename (default is .png).

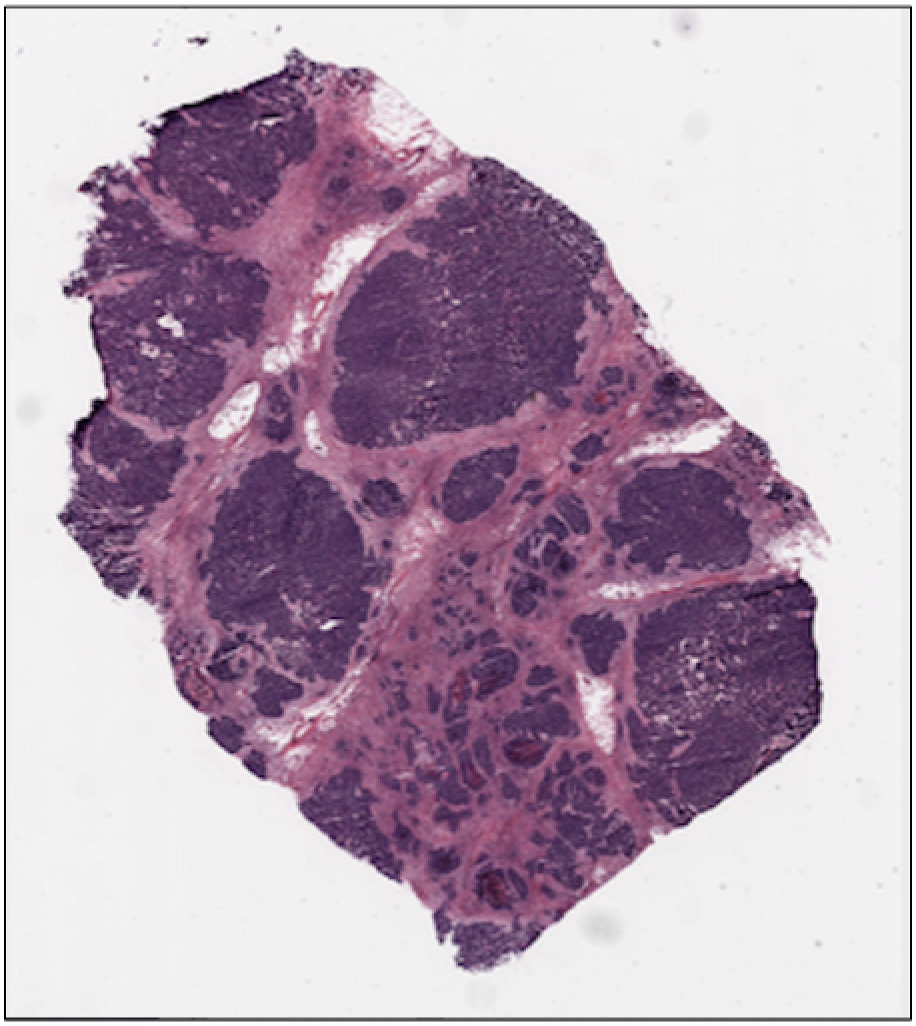
Random Extraction

The simplest approach we may adopt is to randomly crop a fixed number of tiles from our slides; in this case, we need the `RandomTiler` extractor:

```
from histolab.tiler import RandomTiler
```

Let us suppose that we want to randomly extract 30 squared tiles at level 2 of size 128 from our prostate slide, and that we want to save them only if they have at least 80% of tissue inside. We then initialize our `RandomTiler` extractor as follows:





```
random_tiles_extractor = RandomTiler(
    tile_size=(128, 128),
    n_tiles=30,
    level=2,
    seed=42,
    check_tissue=True, # default
    tissue_percent=80.0, # default
    prefix="random/", # save tiles in the "random" subdirectory of slide's processed_path
    suffix=".png" # default
)
```

Notice that we also specify the random seed to ensure the reproducibility of the extraction process.

We may want to check which tiles have been selected by the tiler, before starting the extraction procedure and saving them; the `locate_tiles` method of `RandomTiler` returns a scaled version of the slide with the corresponding tiles outlined. It is also possible to specify the transparency of the background slide, and the color used for the border of the tiles:

```
random_tiles_extractor.locate_tiles(
    slide=prostate_slide,
    scale_factor=24,  # default
    alpha=128,  # default
    outline="red",  # default
)
```

Starting the extraction is then as simple as calling the `extract` method on the extractor, passing the slide as parameter:

```
random_tiles_extractor.extract(prostate_slide)
```

Random tiles extracted from the prostate slide at level 2.

Grid Extraction

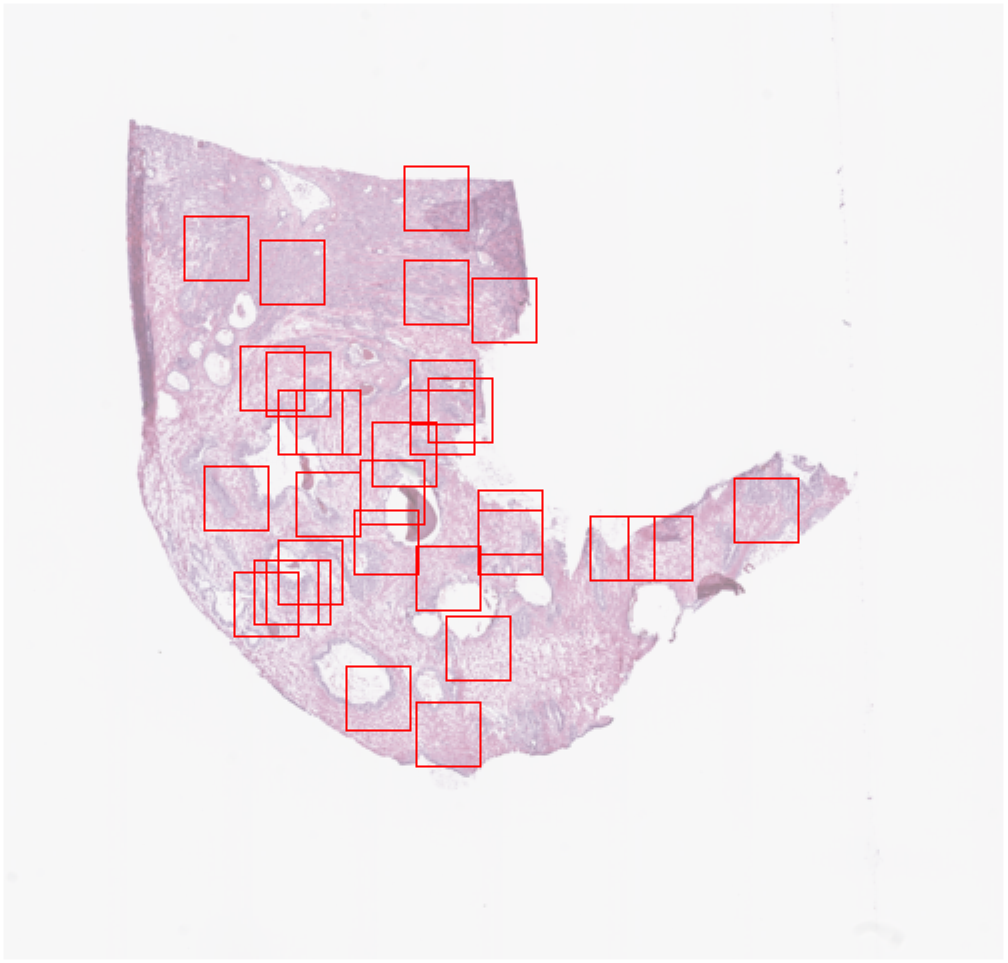
Instead of picking tiles at random, we may want to retrieve all the tiles available. The Grid Tiler extractor crops the tiles following a grid structure on the largest tissue region detected in the WSI:

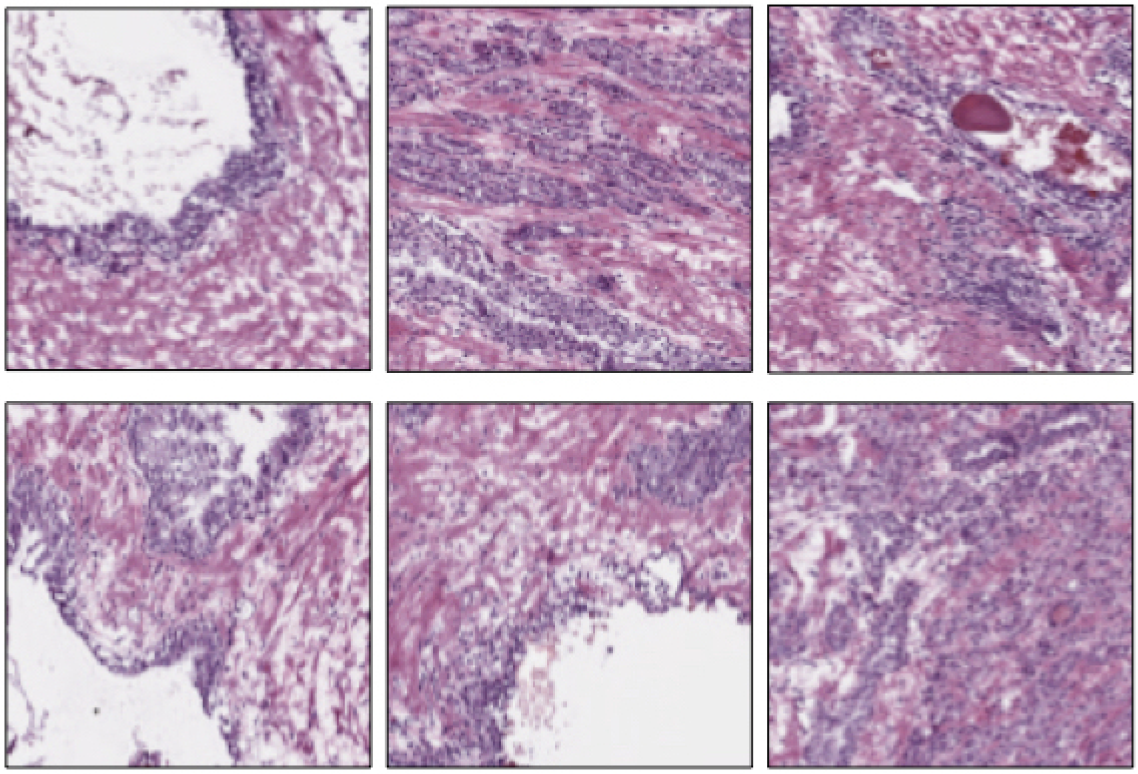
```
from histolab.tiler import GridTiler
```

In our example, we want to extract squared tiles at level 0 of size 512 from our ovarian slide, independently of the amount of tissue detected. By default, tiles will not overlap, namely the parameter defining the number of overlapping pixels between two adjacent tiles, `pixel_overlap`, is set to zero:

```
grid_tiles_extractor = GridTiler(
   tile_size=(512, 512),
   level=0,
   check_tissue=True, # default
   pixel_overlap=0, # default
   prefix="grid/", # save tiles in the "grid" subdirectory of slide's processed_path
   suffix=".png" # default
)
```

Again, we can exploit the `locate_tiles` method to visualize the selected tiles on a scaled version of the slide:





```
grid_tiles_extractor.locate_tiles(
    slide=ovarian_slide,
    scale_factor=64,
    alpha=64,
    outline="#046C4C",
)
```

and the extraction process starts when the extract method is called on our extractor:

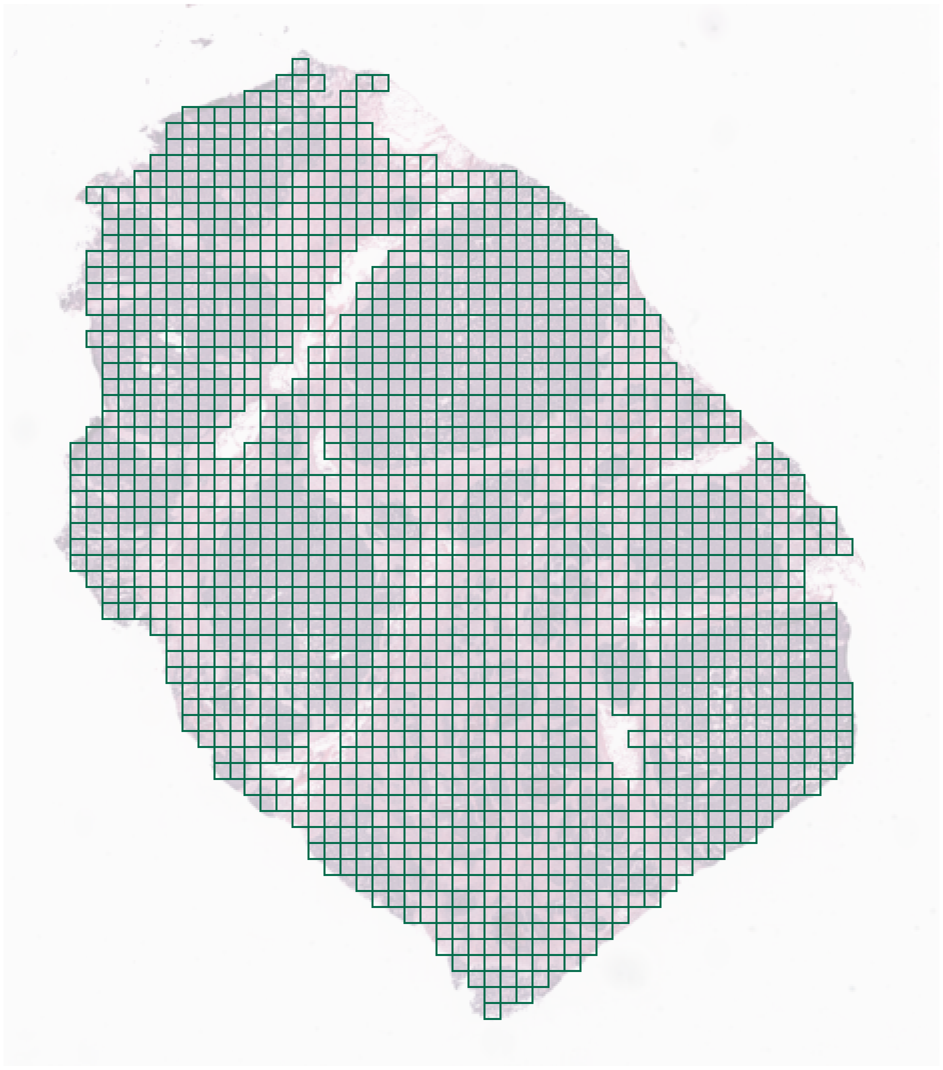
```
grid_tiles_extractor.extract(ovarian_slide)
```

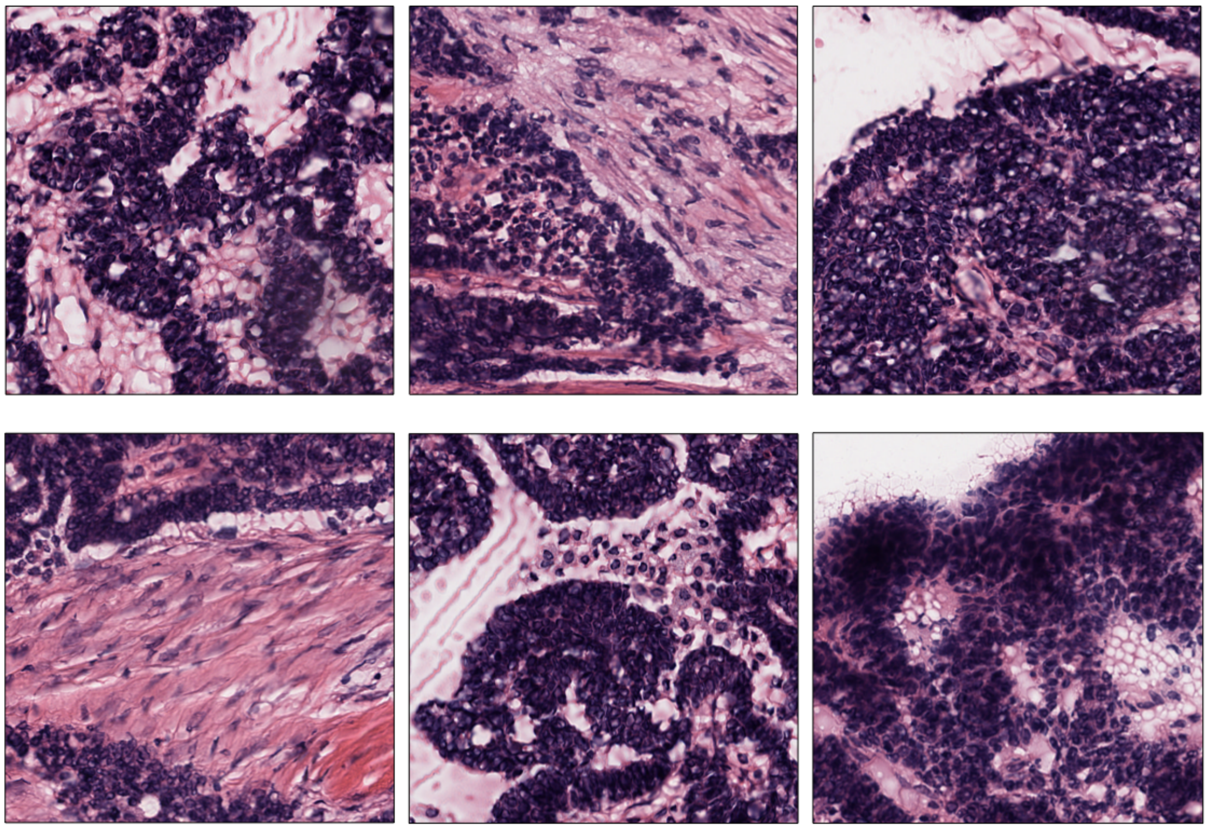
Examples of non-overlapping grid tiles extracted from the ovarian slide at level 0.

Score-based extraction

Depending on the task we will use our tile dataset for, the extracted tiles may not be equally informative. The `ScoreTiler` allows us to save only the "best" tiles, among all the ones extracted with a grid structure, based on a specific scoring function. For example, let us suppose that our goal is the detection of mitotic activity on our ovarian slide. In this case, tiles with a higher presence of nuclei are preferable over tiles with few or no nuclei. We can leverage the `NucleiScorer` function of the `scorer` module to order the extracted tiles based on the proportion of the tissue and of the hematoxylin staining. In particular, the score is computed as $N_t \cdot \tanh(T_t)$, where N_t is the percentage of nuclei and T_t the percentage of tissue in the tile t.

First, we need the extractor and the scorer:





```python
from histolab.tiler import ScoreTiler
from histolab.scorer import NucleiScorer
```

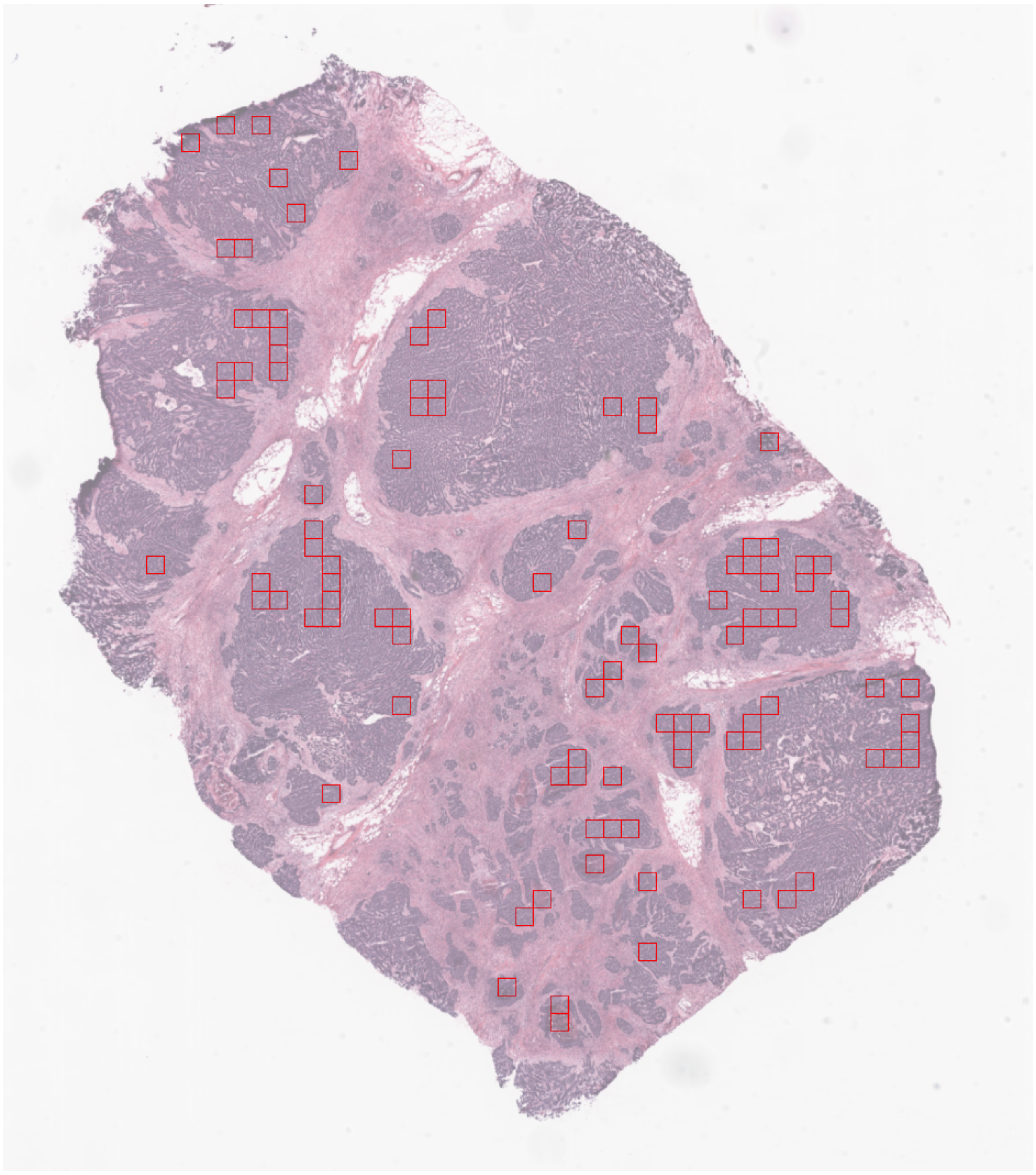
As the `ScoreTiler` extends the `GridTiler` extractor, we also set the `pixel_overlap` as additional parameter. Moreover, we can specify the number of the top tiles we want to save with the `n_tile` parameter:

```python
scored_tiles_extractor = ScoreTiler(
    scorer = NucleiScorer(),
    tile_size=(512, 512),
    n_tiles=100,
    level=0,
    check_tissue=True,
    tissue_percent=80.0,
    pixel_overlap=0, # default
    prefix="scored/", # save tiles in the "scored" subdirectory of slide's processed_path
    suffix=".png" # default
)
```

Notice that also the `ScoreTiler` implements the `locate_tiles` method, which visualizes (on a scaled version of the slide) the first `n_tiles` with the highest scores:

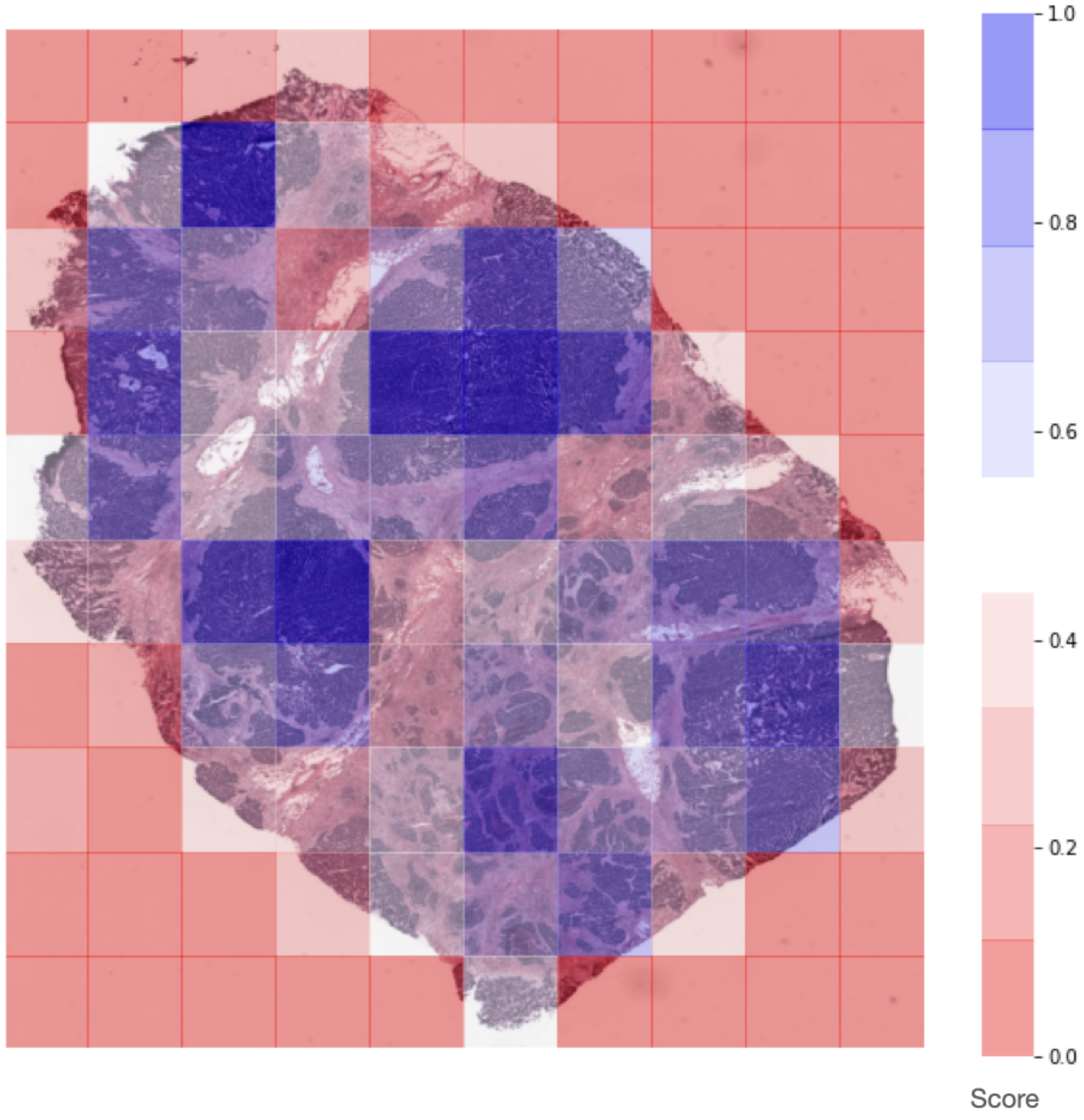
```python
grid_tiles_extractor.locate_tiles(slide=ovarian_slide)
```

Finally, when we extract our cropped images, we can also write a report of the saved tiles and their scores in a CSV file:



```
summary_filename = "summary_ovarian_tiles.csv"
SUMMARY_PATH = os.path.join(ovarian_slide.processed_path, summary_filename)

scored_tiles_extractor.extract(ovarian_slide, report_path=SUMMARY_PATH)
```



Representation of the score assigned to each extracted tile by the NucleiScorer, based on the amount of nuclei detected.

6.4 Define the tissue mask

When extracting the tile dataset from the WSI collection, we may want to consider only a portion of the tissue, rather than the whole slide. For example, a single WSI can include multiple individual slices of tissue, or we may have pathologist annotations with the regions of interest (ROIs) we need to consider.

In this tutorial, we will see how to define different tissue masks in order to refine the tile extraction procedure.

First, we need to load some modules and an example Slide; here we will consider the Kidney WSI available in the data module.

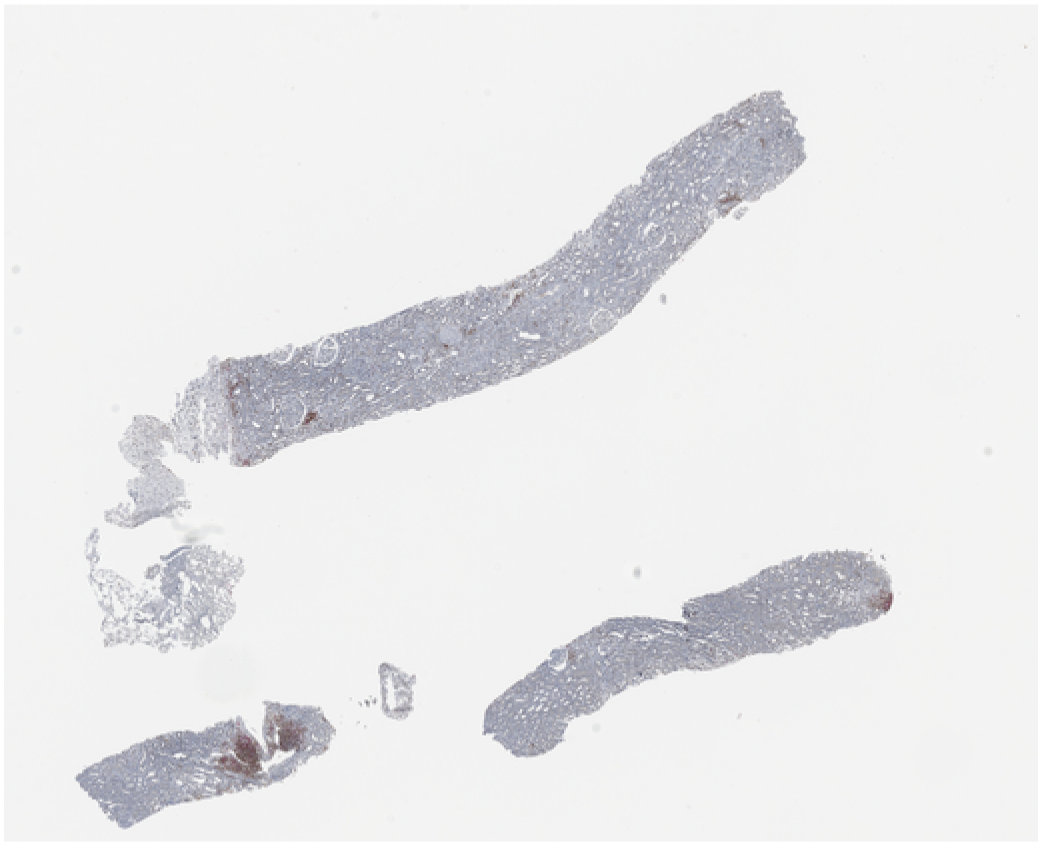
```
from histolab.slide import Slide
from histolab.data import ihc_kidney
import os

BASE_PATH = os.getcwd()

PROCESS_PATH = os.path.join(BASE_PATH, 'kidney', 'processed')

ihc_kidney_svs, ihc_kidney_path = ihc_kidney_svs()
ihc_kidney_slide = Slide(ihc_kidney_path, processed_path=PROCESS_PATH)

ihc_kidney_slide.thumbnail
```



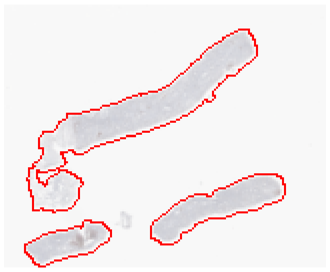
From our Slide object we can now retrieve a binary mask considering specific regions of the tissue. Notice that available masks are defined in the masks module.

As a diagnostic check to visualize the mask, we can call the locate mask method on the Slide, which outlines the boundaries of the selected mask on the slide's thumbnail.

6.4.1 TissueMask

If we want to account for all the tissue detected on the slide, the TissueMask is what we need:

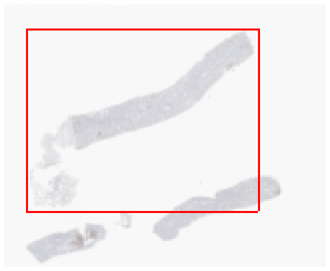
```
from histolab.masks import TissueMask
all_tissue_mask = TissueMask()
ihc_kidney_slide.locate_mask(all_tissue_mask)
```



6.4.2 BiggestTissueBoxMask

The BiggestTissueBoxMask keeps only the largest connected component of the tissue, and returns the bounding box including that region:

```
from histolab.masks import BiggestTissueBoxMask
largest_area_mask = BiggestTissueBoxMask()
ihc_kidney_slide.locate_mask(largest_area_mask)
```



6.4.3 Custom Mask

It is also possible to define a custom binary mask by subclassing the BinaryMask object. For example, we can limit a rectangular region with upper-left coordinates (400, 280) and bottom-right coordinates (300, 320):

```
from histolab.masks import BinaryMask
from histolab.util import rectangle_to_mask
from histolab.types import CP

class MyCustomMask(BinaryMask):
    def _mask(self, slide):
```

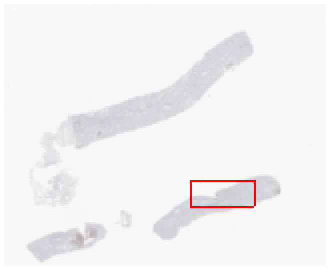
(continues on next page)

(continued from previous page)

```
        thumb = slide.thumbnail
        my_mask = rectangle_to_mask(thumb.size, CP(400, 280, 300, 320))
        return my_mask

custom_mask = MyCustomMask()

ihc_kidney_slide.locate_mask(custom_mask)
```



6.4.4 Tile extraction within the mask

We can finally pass our mask to the extract method of our Tiler object, and visualize the location of the extracted tiles:

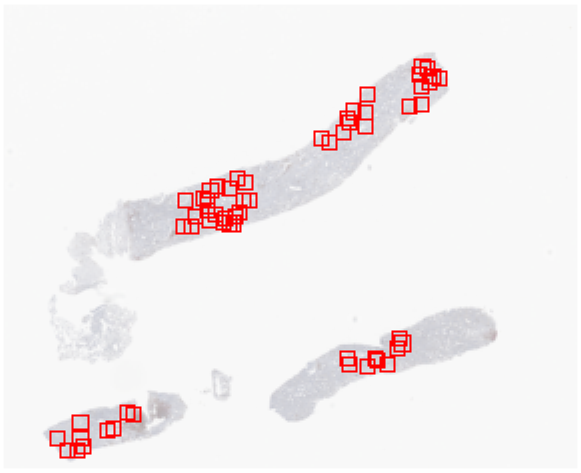
```
from histolab.tiler import RandomTiler

rtiler = RandomTiler(
    tile_size=(128, 128),
    n_tiles=50,
    level=0,
    tissue_percent=90,
    seed=0,
)

rtiler.extract(ihc_kidney_slide, all_tissue_mask)

rtiler.locate_tiles(
    slide=ihc_kidney_slide,
    extraction_mask=all_tissue_mask,
)
```

Note: The BiggestTissueBoxMask is considered as default binary mask.



6.5 Contributing

When contributing to this repository, please first discuss the change you wish to make via issue, email, or any other method with the owners of this repository before making a change.

Please note we have a code of conduct, please follow it in all your interactions with the project.

6.5.1 Contribution guidelines and standards

Before sending your PR for review, make sure your changes are consistent with the guidelines and follow the coding style.

General guidelines and philosophy for contribution

- Include unit tests when you contribute new features, as they help to a) prove that your code works correctly, and b) guard against future breaking changes to lower the maintenance cost.
- Bug fixes also generally require unit tests, because the presence of bugs usually indicates insufficient test coverage.
- Keep API compatibility in mind when you change code in Histolab core.
- Tests coverage cannot decrease from the current %.
- Do not push integration tests without unit tests.

6.5.2 Contribution Workflow

Before working on your next contribution, make sure your local repository is up to date.

1. Set the upstream remote. (You only have to do this once per project, not every time.)

   ```
   $ git remote add upstream git@github.com:histolab/project-repo-name
   ```

2. Switch to the local master branch.

   ```
   $ git checkout master
   ```

3. Pull down the changes from upstream.

   ```
   $ git pull upstream master
   ```

4. Push the changes to your GitHub account.

   ```
   $ git push origin master
   ```

5. Create a new branch if you are starting new work.

   ```
   $ git checkout -b branch-name
   ```

Code contributions, bug fixes, new development, test improvement, all follow a GitHub-centered workflow. To participate in Histolab development, set up a GitHub account. Then:

1. Fork the repo https://github.com/histolab/histolab. Go to the project repo page and use the Fork button. This will create a copy of the repo, under your username. (For more details on how to fork a repository see this guide.)
2. Clone down the forked repo to your local machine.

   ```
   $ git clone git@github.com:your-user-name/project-name.git
   ```

3. Create a new branch to hold your work.

   ```
   $ git checkout -b new-branch-name
   ```

4. Work on your code. Write and run tests.
5. Commit your changes.

   ```
   $ git add .
   ```

   ```
   $ git commit -m "commit message here"
   ```

6. Push your changes to your GitHub repo.

   ```
   $ git push origin branch-name
   ```

7. Open a Pull Request (PR). Go to the original project repo on GitHub. There will be a message about your recently pushed branch, asking if you would like to open a pull request. Follow the prompts, compare across repositories, and submit the PR. For more read here
8. Maintainers and other contributors will review your PR. Please participate in the conversation, and try to make any requested changes. Once the PR is approved, the code will be merged.

Additional git and GitHub resources:

- Git documentation
- Git development workflow
- Resolving merge conflicts

6.5.3 Create your local environment

Before starting contributing to Histolab, test that your local environment is up and running. Here some steps:

- Create a python 3.7, 3.8, 3.9 or 3.10 `virtualenv`
- Activate the env and in the project root run:

  ```
  pip install -e .[testing]
  ```

  ```
  pip install -r requirements-dev.txt
  ```

- Install the pre-commit hooks (Optional, but useful for code style compliance)

  `pre-commit install` <- *to be ran in the project root directory*

- Run the tests

  ```
  pytest tests/
  ```

6.5.4 Code of Conduct

Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting one of the project mantainers/owners. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at http://contributor-covenant.org/version/1/4

6.6 Changelog

6.6.1 v0.4.0

Bug Fix

- Fix *SlideSet* not passing along args to *Slide*. (#351)

New Features

- Add RAG threshold parameters to allow return labels and masking. (#300)
- Support fetching exact MPP resolutions. (#301)
- Allow only keeping a subset of slides for *SlideSet*. (#303)

Documentation

- Fix link to installation instructions. (#350)

6.6.2 v0.3.1

Bug Fix

- Map TCGA old UUID with new UUID to fix *data* module. (#346)

Dependencies

- Support Python 3.10. (#336)
- Support scipy 1.7.2. (#345)
- Upgrade sphinx to 4.2.3. (#349)

Other

- Add test for *np_to_pil* with float values [0,1] array. (#347)

6.6.3 v0.3.0

Bug Fix

- Fix *GridTiler*'s *_are_coordinates_within_extraction_mask* method where tile coordinates are off by 1 or 2 pixels due to conversion of floats to ints. (#308)
- Fix the mismatch between row-column / X-Y coordinates in the RandomTiler (#317)
- Fix return type of RGB to LAB filter. (#323)
- Filter *kmeans_segmentation* is now applied only to RGB images. (#328)
- Conversion from RGB to HED preserves HED color space range (#334)
- Conversion from RGB to HSV preserves HSV color space range (#337)
- Remove HSV and YCBCR references in wrong value range in tests (#343)

New Features

- Add RGB to OD filter. (#290 and #331)
- Add method dispatcher compatible with older Python versions. (#312)
- Add LAB to RGB filter. (#323)
- Finer control of *locate_tiles* (pass tiles to avoid re-extraction and color tiles' border individually). (#304)
- Add *TissueMask* mask for *Tile* with type dispatcher. (#313)
- Add conversion level - magnification factor in *Slide*. (#319)
- Add *CellularityScorer*. (#320)

Maintenance

- Link automatically issues in PR template. (#291)
- Include histolab version in issue template. (#296)
- Add security linter with Bandit in pre commit and CI. (#316)
- Get rid of *src* directory in favor of *histolab* dir within the root. (#324)
- Use Python 3.9 for benchmarks. (#342)

Dependencies

- Support scikit-image 0.18.3. (#196, #200 and #327)
- Support scipy 1.7.1. (#305)
- Upgrade sphinx to 4.2.0 to fix incompatibility with docutils 0.18. (#339)
- Support numpy 1.21.4. (#344)

Documentation

- Fix docs links in *tissue_mask* module. (#321)
- Add note on data module for TCGA example data not available. (#325 and #333)

6.6.4 v0.2.6

Bug Fix

- Fix `polygon_to_mask_array` return mask shape. (#268)
- Fix overlapping extraction grids in `GridTiler`. (#270)

New Features

- Add DAB filter. (#277)
- Allow slide name to contain dot. (#281)

Documentation

- Docs fixes about Slide's processed_path. (#276)
- Add instructions on how to install Pixman 0.40. (#280)

6.6.5 v0.2.5

Bug Fix

- *RandomTiler* coordinates selection within the binary mask. (#256)
- *LocalOtsuThreshold* filter: now it returns correct type (PIL Image). (#258)
- Coordinate definition in the scale coordinates of *RandomTiler* were reversed. (#261)

New Features

- Support and test for IHC-stained slides. (#262)

Documentation

- Extended documentations to include examples, images, and tutorials. Added IHC-stained slides in the data module. (#232)

6.6.6 v0.2.4

Bug Fix

- *RandomTiler* now respects the given tile size (#243)
- Use logger object instead of logging module when logging tiler updates (#237)

New Features

- New *masks* module to create binary masks from slides with different strategies: *BiggestTissueBoxMask* and *TissueMask* (#234)
- Refactor locate_mask to draw mask contours on the slide from an arbitrary BinaryMask object (#248)

Breaking Changes

- Refactor *Slide*: return thumbnail and scaled image instead of saving them (#236)

6.6.7 v0.2.3

New Features

- Allow pathlib.Path as Slide path parameter (#226)
- Tilers *extract* method now has *log_level* param that set the threshold level for the log messages (#229)

6.6.8 v0.2.2

Bug Fix

- Fix of *np_to_pil* in case float input but in a correct range (#199)
- Fix tiles extractor checking if the tile size is larger than the slide size (#202)
- Fix RandomTiler border wackiness extraction (#203)

New Features

- New parameter *tissue_percent* for all the tilers' to be used during the *has_enough_tissue* check (#204)
- Expose wsi properties. The *Slide.properties* returns the whole OpenSlide WSI properties (#209)
- Allow negative indexing for *slide.level* (#210)
- New Filter Protocol available (#213)

Breaking Changes

- Remove pen marks filter (#201)

6.6.9 v0.2.1

Maintenance

- Pin dependencies in requirements.txt to avoid discrepancy with scikit-image v0.18.0

6.6.10 v0.2.0

Bug Fix

- Bug: Fix grid tile coordinates calculation (#186)
- Bug: Fix quickstart tutorial slides' paths (#154 and #165)

New Features

- Add diagnostic method to locate tiles on a slide with every Tiler (#179)
- Add diagnostic method to locate the biggest tissue bounding box on a slide (#188)
- *SlideSet* is iterable and its *slides* property has been dropped (#177)

6.6.11 v0.1.1

New Features

- Add RgbToLab image filter (#147)
- Add Watershed segmentation filter (#153)
- Support Python 3.8 on Linux and macOS (#151)

6.6.12 v0.0.1

Bug Fix

- Fix save path for tiles (#126)
- Fix critical memory issue when extracting biggest tissue box (#128)

New Features

- Add Lambda filter (#124)
- Add ScoreTiler and RandomScorer (#129)
- Add NucleiScorer (#132)
- Add Ovarian Tissue sample in data module (#136)

6.6.13 v0.0.5b

Bug Fix

- Fix issue (#100)
- Fix issue (#108)

New Features

- Grid Tiler (#99)

6.6.14 v0.0.4b

Bug Fix

- Fix kmeans segmentation image filter default parameters
- Fix rag threshold image filter default parameters
- Fix check tissue on *Tile* to discard almost white tiles

6.7 Slide

The `slide` module provides a simple high-level interface to handle a WSI; it contains the Slide class, which wraps functions, methods and properties of a virtual slide in a single object. The Slide class encapsulates `OpenSlide`, and relies on the `openslide-python` library for the low-level operations on digital slides. A WSI is usually stored in pyramidal format, where each level corresponds to a specific magnification factor. Therefore, two relevant properties of a WSI are: (i) its dimensions at native magnification; (ii) the number of levels and the dimensions at a specified level.

Note: `OpenSlide` identifies each magnification level of the WSI with a positive integer number, starting from 0.

A Slide is initialized by providing the path where the WSI is stored and the path where processed images (such as the WSI thumbnail or the extracted tiles) will be saved. Further, the `slide` module implements the SlideSet class, which handles a collection of Slide objects stored in the same directory, possibly filtered by the `valid_extensions` parameter.

The slides_stats property of a SlideSet computes statistics for the WSI collection, namely the number of available slides; the slide with the maximum/minimum width; the slide with the maximum/minimum height; the slide with the maximum/minimum size; the average width/height/size of the slides.

`class Slide`(*path*, *processed_path*, *use_largeimage=False*)

Provide Slide objects and expose property and methods.

Parameters

- **`path`** (*`Union[str, pathlib.Path]`*) – Path where the WSI is saved.
- **`processed_path`** (*`Union[str, pathlib.Path]`*) – Path where the tiles will be saved to.
- **`use_largeimage`** (*`bool, optional`*) – Whether or not to use the *large_image* package for accessing the slide and extracting or calculating various metadata. If this is *False*, *openslide* is used. If it is *True*, *large_image* will try from the various installed tile sources. For example, if you installed it using *large_image[all]*, it will try *openslide* first, then *PIL*, and so on, depending on the slide format and metadata. *large_image* also handles internal logic to enable fetching exact micron-per-pixel resolution tiles by interpolating between the internal levels of the slide. If you don't mind installing an extra dependency, we recommend setting this to True and fetching Tiles at exact resolutions as opposed to levels. Different scanners have different specifications, and the same level may not always encode the same magnification in different scanners and slide formats.

Raises

- **`TypeError`** – If the processed path is not specified.
- **`ModuleNotFoundError`** – when *use_largeimage* is set to True and *large_image* module is not installed.

Return type None

`property base_mpp: float`

Get microns-per-pixel resolution at scan magnification.

Returns Microns-per-pixel resolution at scan (base) magnification.

Return type float

Raises

- **`ValueError`** – If *large_image* cannot detemine the slide magnification.

- `MayNeedLargeImageError` – If *use_largeimage* was set to False when slide was initialized, and we cannot determine the magnification otherwise.

`property dimensions: Tuple[int, int]`
Slide dimensions (w,h) at level 0.

Returns **dimensions** – Slide dimensions (width, height)

Return type Tuple[int, int]

`extract_tile`(*coords*, *tile_size*, *level=None*, *mpp=None*)
Extract a tile of the image at the selected level.

Parameters

- `coords` (*CoordinatePair*) – Coordinates at level 0 from which to extract the tile.
- `tile_size` (*Tuple[int, int]*) – Final size of the extracted tile (x,y). If you choose to specify the *mpp* argument, you may elect to set this as *None* to return the tile as-is from *large_image* without any resizing. This is not recommended, as tile size may be off by a couple of pixels when coordinates are mapped to the exact mpp you request.
- `level` (*int*) – Level from which to extract the tile. If you specify this, and *mpp* is None, *openslide* will be used to fetch tiles from this level from the slide. *openslide* is used for fetching tiles by level, regardless of *self.use_largeimage*.
- `mpp` (*float*) – Micron per pixel resolution. Takes precedence over level. If this is not None, *large_image* will be used to fetch tiles at the exact microns-per-pixel resolution requested.

Returns **tile** – Image containing the selected tile.

Return type *Tile*

`level_dimensions`(*level=0*)
Return the slide dimensions (w,h) at the specified level

Parameters `level` (*int*) – The level which dimensions are requested, default is 0.

Returns **dimensions** – Slide dimensions at the specified level (width, height)

Return type Tuple[int, int]

Raises `LevelError` – If the specified level is not available

`level_magnification_factor`(*level=0*)
Return the magnification factor at the specified level.

Notice that the conversion level-magnification can be computed only if the native magnification is available in the slide metadata.

Parameters `level` (*int*) – The level which magnification factor is requested, default is 0.

Returns **magnification factor** – Magnification factor at speficied level

Return type str

Raises

- `LevelError` – If the specified level is not available.
- `SlidePropertyError` – If the slide's native magnification or the downsample factor for the specified level are not available in the file's metadata.

`property levels: List[int]`
Slide's available levels

Returns The levels available

Return type List[int]

`locate_mask`(*binary_mask*, *scale_factor=32*, *tissue_mask=False*, *alpha=128*, *outline='red'*)

Draw binary mask contours on a rescaled version of the slide

Parameters

- **binary_mask** (`BinaryMask`) – Binary Mask object
- **scale_factor** (*int*) – Scaling factor for the returned image. Default is 32.
- **tissue_mask** (*bool, optional*) – Whether to draw the contours on the binary tissue mask instead of the rescaled version of the slide. Default is False.
- **alpha** (*int*) – The alpha level to be applied to the rescaled slide, default to 128.
- **outline** (*str*) – The outline color for the annotation, default to 'red'.

Returns PIL Image of the rescaled slide with the binary mask contours outlined.

Return type PIL.Image.Image

property name: str

Slide name without extension.

Returns **name**

Return type str

property processed_path: str

Path to store the tiles generated from the slide.

Returns Path to store the tiles generated from the slide

Return type str

property properties: dict

Whole Slide Image properties.

Returns WSI complete properties.

Return type dict

`resampled_array`(*scale_factor=32*)

Return the resampled array from the original slide

Parameters **scale_factor** (*int, optional*) – Image scaling factor. Default is 32.

Returns **resampled_array** – Resampled array

Return type np.ndarray

`scaled_image`(*scale_factor=32*)

Return a scaled image of the slide.

Parameters **scale_factor** (*int, optional*) – Image scaling factor. Default is 32.

Returns A scaled image of the slide.

Return type PIL.Image.Image

`show`()

Display the slide thumbnail.

NOTE: A new window of your OS image viewer will be opened.

Return type None

`property thumbnail: PIL.Image.Image`

Slide thumbnail.

Returns The slide thumbnail.

Return type PIL.Image.Image

`class SlideSet`(*slides_path*, *processed_path*, *valid_extensions*, *keep_slides=None*, *slide_kwargs=None*)

Slideset object. It is considered a collection of Slides.

Parameters

- **slides_path** (*str*) –
- **processed_path** (*str*) –
- **valid_extensions** (*List[str]*) –
- **keep_slides** (*List[str]*) –
- **slide_kwargs** (*dict*) –

Return type None

`scaled_images`(*scale_factor=32*, *n=0*)

Return rescaled images of the slides.

Parameters

- **scale_factor** (*int, optional*) – Image scaling factor. Default is 32.
- **n** (*int, optional*) – First n slides in dataset folder to rescale. Default is 0, meaning that all the slides will be returned.

Returns List of rescaled images of the slides.

Return type List[PIL.Image.Image]

`property slides_stats: dict`

Statistics for the WSI collection, namely the number of available slides; the slide with the maximum/minimum width; the slide with the maximum/minimum height; the slide with the maximum/minimum size; the average width/height/size of the slides.

Returns **basic_stats**

Return type dict of slides stats e.g. min_size, avg_size, etc…

`thumbnails`(*n=0*)

Return slides thumbnails

Parameters **n** (*int, optional*) – First n slides in dataset folder. Default is 0, meaning that the thumbnails of all the slides will be returned.

Returns List of slides thumbnails

Return type List[PIL.Image.Image]

`property total_slides: int`

Number of slides within the slideset.

Returns **n** – Number of slides.

Return type int

6.8 Filters

The filters subpackage implements a pool of functions for image manipulation, including contrast enhancement, color deconvolution, and background removal. Two modalities of filters are defined by their input types: image filters, and morphological filters, which act on binary masks.

Filters in `histolab` are designed to be applied singularly or combined in a chain of transformations. A composition of filters is predefined for tissue segmentation, while custom filter combinations can be used for tissue detection or other tasks.

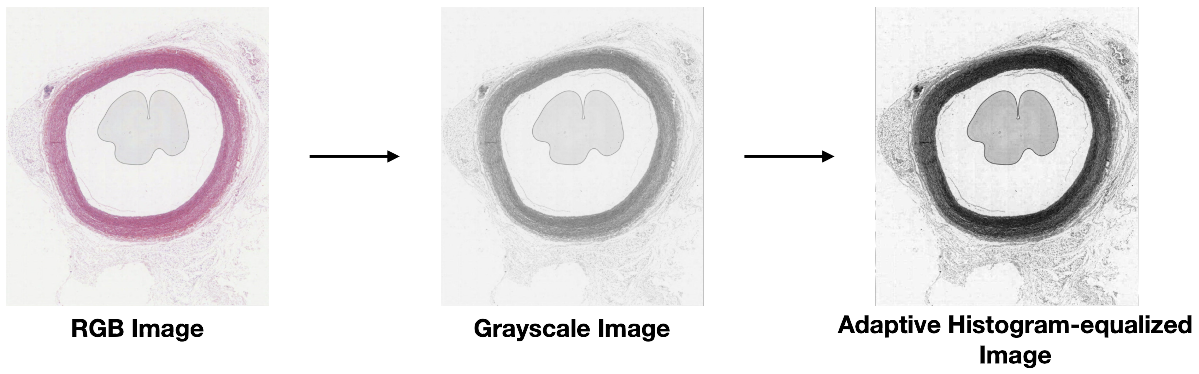
6.8.1 Image Filters

All filters implemented in the image filters submodule take as input a Pillow Image object. Additionally, some of the image filters in histolab leverage functions and utilities by scikit-image. Image filters are divided into sub-categories, depending on their behaviour and output type.

class AdaptiveEqualization(**args*, ***kwds*)

Increase image contrast using adaptive equalization.

Rather than considering the global contrast in the image, the adaptive histogram equalization method applies the histogram equalization to smaller regions, or tiles, of the image; the tiles are then combined together using bilinear interpolation. This local approach is preferred when the image presents significantly darker or lighter regions that may be poorly enhanced by the global histogram equalization transformation.



The Adaptive Equalization filter is based on the scikit-image implementation of the contrast limited adaptive histogram equalization (CLAHE)[1].

Parameters

- **img** (*PIL.Image.Image*) – Input image (gray or RGB)
- **nbins** (*int, optional*) – Number of histogram bins. Default is 256.
- **clip_limit** (*float, optional*) – Clipping limit where higher value increases contrast. Default is 0.01.

Returns Image with contrast enhanced by adaptive equalization.

Return type PIL.Image.Image

[1] S.M. Pizer andet al. "Adaptive histogram equalization and its variations", Comput Vis Graph Image Process 39.3 (1987).

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import AdaptiveEqualization, RgbToGrayscale
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> rgb_to_grayscale = RgbToGrayscale()
>>> adaptive_equalization = AdaptiveEqualization()
>>> image_gray = rgb_to_grayscale(image_rgb)
>>> image_clahe = adaptive_equalization(image_gray)
```

References

class **ApplyMaskImage**(**args*, ***kwds*)

Mask image with the provided binary mask.

Parameters

- **img** (*PIL.Image.Image*) – Input image
- **mask** (*np.ndarray*) – Binary mask

Returns Image with the mask applied

Return type PIL.Image.Image

class **BlueFilter**(**args*, ***kwds*)

Filter out blueish colors in an RGB image.

Create a mask to filter out blueish colors, where the mask is based on a pixel being above a red channel threshold value, above a green channel threshold value, and below a blue channel threshold value.

Parameters

- **img** (*PIl.Image.Image*) – Input RGB image
- **red_thresh** (*int*) – Red channel lower threshold value.
- **green_thresh** (*int*) – Green channel lower threshold value.
- **blue_thresh** (*int*) – Blue channel upper threshold value.

Returns Boolean NumPy array representing the mask.

Return type np.ndarray

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import BlueFilter
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/wsi-blue-pen.png")
>>> blue_filter = BlueFilter(30, 20, 105)
>>> mask_filtered = blue_filter(image_rgb)
```

class **BluePenFilter**(**args*, ***kwds*)

Filter out blue pen marks from a diagnostic slide.

The resulting mask is a composition of green filters with different thresholds for the RGB channels.

Parameters **img** (*PIL.Image.Image*) – Input RGB image

Returns NumPy array representing the mask with the blue pen marks filtered out.

Return type np.ndarray

Example

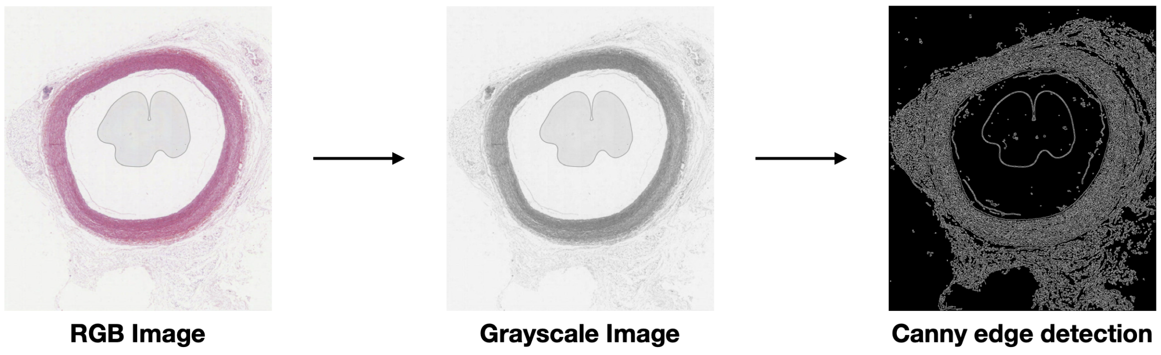
```
>>> from PIL import Image
>>> from histolab.filters.image_filters import BluePenFilter
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/wsi-blue-pen.png")
>>> blue_pen_filter = BluePenFilter()
>>> image_no_blue = blue_pen_filter(image_rgb)
```

`class CannyEdges(*args, **kwds)`

Filter image based on Canny edge algorithm.

The Canny edge detector has been used to generate a version of the image that highlights edges within tissue fragments by detecting changes in pixel intensity[2][3]. The algorithm includes five steps: (i) smoothing the image (i.e. remove the noise); (ii) computing the gradient's magnitude $M_\bigtriangledown$ and direction $\theta_\bigtriangledown$; (iii) keeping the direction $\theta_\bigtriangledown$ with greatest intensity $M_\bigtriangledown$ for each pixel; (iv) thinning the edges by suppressing non-maximal pixels; (v) applying the hysteresis thresholding algorithm for the final edge detection.

Note that input image must be 2D.



Parameters

- **img** (*PIL.Image.Image*) – Input 2-dimensional image
- **sigma** (*float, optional*) – Width (std dev) of Gaussian. Default is 1.0.
- **low_threshold** (*float, optional*) – Low hysteresis threshold value. Default is 0.0.
- **high_threshold** (*float, optional*) – High hysteresis threshold value. Default is 25.0.

Returns Boolean NumPy array representing Canny edge map.

Return type np.ndarray

[2] A Kumar and M Prateek. "Localization of Nuclei in Breast Cancer Using Whole SlideImaging System Supported by Morphological Features and Shape Formulas". CancerManag Res 12 (2020)

[3] M Munoz-Aguirre and et al. "PyHIST: A Histological Image Segmentation Tool". PLOS Comput Biol 16.10 (2020)

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import CannyEdges, RgbToGrayscale
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> rgb_to_grayscale = RgbToGrayscale()
>>> canny_edges_detection = CannyEdges()
>>> image_gray = rgb_to_grayscale(image_rgb)
>>> image_thresholded_array = canny_edges_detection(image_gray)
```

References

`class Compose(*args, **kwds)`
Composes several filters together.

Parameters **filters** (*list of Filters*) – List of filters to compose

`class DABChannel(*args, **kwds)`
Obtain DAB channel from RGB image.

Input image is first converted into HED space and the DAB channel is extracted via color deconvolution.

Parameters **img** (*PIL.Image.Image*) – Input RGB image

Returns RGB image with Eosin staining separated.

Return type PIL.Image.Image

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import DABChannel
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> dab_channel = DABChannel()
>>> image_d = dab_channel(image_rgb)
```

`class EosinChannel(*args, **kwds)`
Obtain Eosin channel from RGB image.

Input image is first converted into HED space and the Eosin channel is extracted via color deconvolution.

Parameters **img** (*PIL.Image.Image*) – Input RGB image

Returns RGB image with Eosin staining separated.

Return type PIL.Image.Image

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import EosinChannel
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> eosin_channel = EosinChannel()
>>> image_e = eosin_channel(image_rgb)
```

class Filter(**args*, ***kwds*)

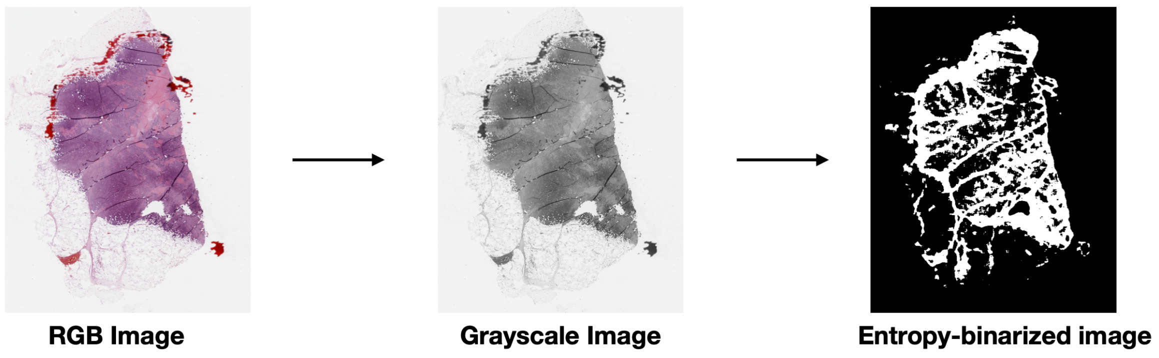
Filter protocol

class FilterEntropy(**args*, ***kwds*)

Filter image based on entropy (complexity).

Entropy measures complexity in an image: the greater the entropy the more heterogeneous structures are found is the image, while slide backgrounds are usually less complex. This method filters out pixels of grayscale images based on the local entropy. In details: (i) the entropy is computed on a neighborhood defined by a squared all-ones matrix of size n (by default n=9); (ii) pixels with entropy greater than a specified threshold t (by default t=5) are replaced with 1, 0 otherwise. This entropy filter can be used to detect highly hematoxylin-stained regions, which represent dense accumulation of nuclei (complex structures).

Note that input must be 2D.



Parameters

- **img** (*PIL.Image.Image*) – input 2-dimensional image
- **neighborhood** (*int, optional*) – Neighborhood size (defines height and width of 2D array of 1's). Default is 9.
- **threshold** (*float, optional*) – Threshold value. Default is 5.0

Returns NumPy boolean array where True represent a measure of complexity.

Return type np.ndarray

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import FilterEntropy, RgbToGrayscale
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> rgb_to_grayscale = RgbToGrayscale()
>>> entropy_filter = FilterEntropy()
>>> image_gray = rgb_to_grayscale(image_rgb)
>>> image_thresholded_array = entropy_filter(image_gray)
```

`class Grays`(**args*, ***kwds*)

Filter out gray pixels in RGB image.

Gray pixels are those pixels where the red, green, and blue channel values are similar, i.e. under a specified tolerance.

Parameters

- **img** (*PIL.Image.Image*) – Input image
- **tolerance** (*int, optional*) – if difference between values is below this threshold, values are considered similar and thus filtered out. Default is 15.

Returns Mask image where the grays values are masked out

Return type PIL.Image.Image

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import Grays
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> grays_filter = Grays(tolerance=5)
>>> filtered_mask = grays_filter(image_rgb)
```

`class GreenChannelFilter`(**args*, ***kwds*)

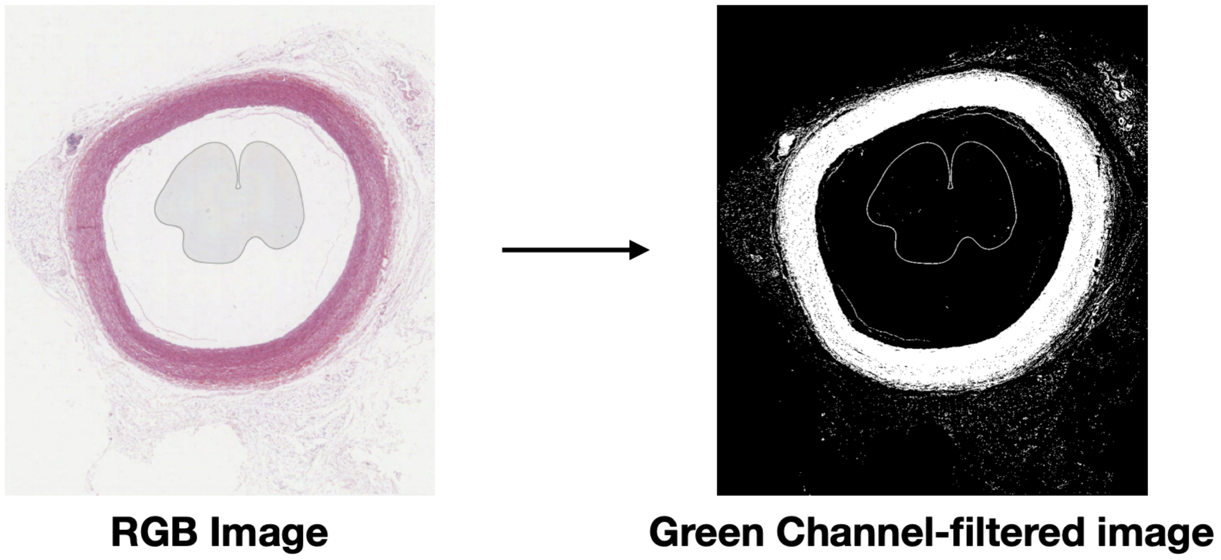
Mask pixels in an RGB image with G-channel greater than a specified threshold.

Create a binary mask where pixels with the green channel value above a specified threshold (by default 200) are set to 0. This filtering method can be used to detect tissue in H&E-stained images, considering that the green dye is poorly used in the tissue-related stains, i.e. eosin (pink) and hematoxylin (purple). To avoid over-masking the image, the overmask_thresh parameter defines the maximum percentage of tissue that can be masked by the green channel filter (by default 90%).

This method alone may be sufficient to segment tissue on H&E-stained images.

Parameters

- **img** (*PIL.Image.Image*) – Input RGB image
- **green_thresh** (*int, optional*) – Green channel threshold value (0 to 255). Default is 200. If value is greater than green_thresh, mask out pixel.
- **avoid_overmask** (*bool, optional*) – If True, avoid masking above the overmask_thresh percentage. Default is True.
- **overmask_thresh** (*float, optional*) – If avoid_overmask is True, avoid masking above this percentage value. Default is 90.



Returns Boolean mask where pixels above a particular green channel threshold have been masked out.

Return type np.ndarray

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import GreenChannelFilter
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> g_channel_filter = GreenChannelFilter(avoid_overmask=True, overmask_thresh=90)
>>> image_thresholded_array = g_channel_filter(image_rgb)
```

class **GreenFilter**(**args*, ***kwds*)

Filter out greenish colors in an RGB image. The mask is based on a pixel being above a red channel threshold value, below a green channel threshold value, and below a blue channel threshold value.

Note that for the green ink, the green and blue channels tend to track together, so for blue channel we use a lower threshold rather than an upper threshold value.

Parameters

- **img** (*PIL.image.Image*) – RGB input image.
- **red_thresh** (*int*) – Red channel upper threshold value.
- **green_thresh** (*int*) – Green channel lower threshold value.
- **blue_thresh** (*int*) – Blue channel lower threshold value.

Returns Boolean NumPy array representing the mask.

Return type np.ndarray

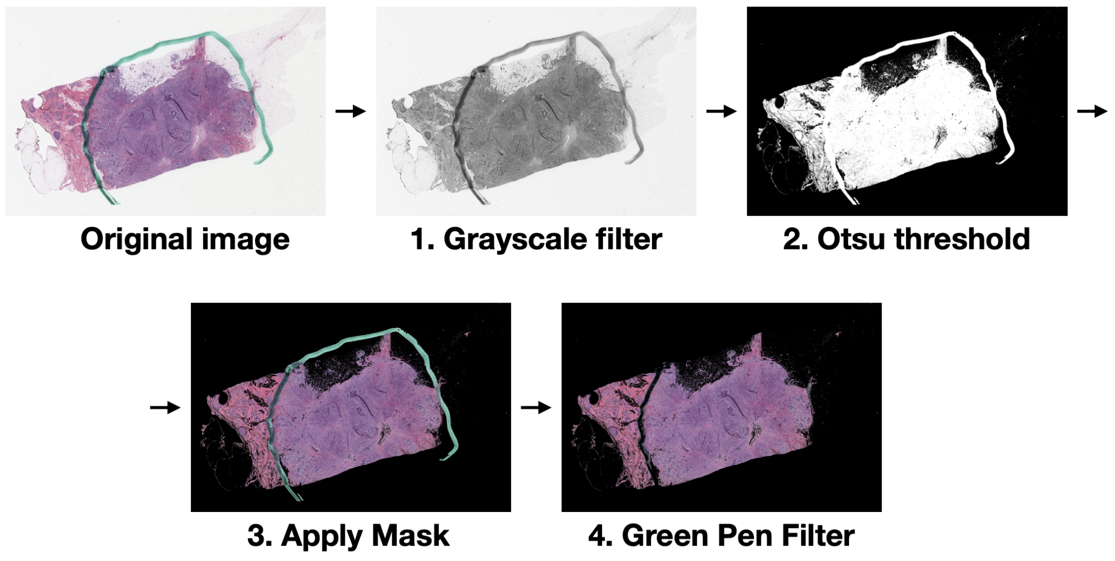
Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import GreenFilter
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-red-pen.png")
>>> green_filter = GreenFilter(230, 10, 105)
>>> mask_filtered = green_filter(image_rgb)
```

`class GreenPenFilter`(**args*, ***kwds*)

Filter out green pen marks from a diagnostic slide.

The resulting mask is a composition of green filters with different thresholds for the RGB channels.



Parameters **img** (*PIL.Image.Image*) – Input RGB image

Returns Image the green pen marks filtered out.

Return type PIL.Image.Image

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import GreenPenFilter
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-green-pen.png")
>>> green_pen_filter = GreenPenFilter()
>>> image_no_green = green_pen_filter(image_rgb)
```

`class HedToRgb`(**args*, ***kwds*)

Convert HED channels to RGB channels.

Parameters **img_arr** (*np.ndarray*) – Array representation of the image in HED color space

Returns Image in RGB space

Return type PIL.Image.Image

Example

```
>>> import numpy as np
>>> from histolab.filters.image_filters import HedToRgb
>>> hed_arr = np.load("tests/fixtures/arrays/diagnostic-slide-thumb-hed.npy")
>>> hed_to_rgb = HedToRgb()
>>> rgb = hed_to_rgb(hed_arr)
```

class **HematoxylinChannel**(**args*, ***kwds*)

Obtain Hematoxylin channel from RGB image.

Input image is first converted into HED space and the hematoxylin channel is extracted via color deconvolution.

Parameters **img** (*PIL.Image.Image*) – Input RGB image

Returns RGB image with Hematoxylin staining separated.

Return type PIL.Image.Image

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import HematoxylinChannel
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> hematoxylin_channel = HematoxylinChannel()
>>> image_h = hematoxylin_channel(image_rgb)
```

class **HistogramEqualization**(**args*, ***kwds*)

Increase image contrast using histogram equalization.

The input image (gray or RGB) is filterd using histogram equalization to increase contrast. In particular, this filter expands the range of intensity values in low contrast images. It first computes the normalized histogram H of an image: H(k) counts pixels with intensity values k, divided by the total number of pixels in the image. Then, it computes the cumulative sum of the histogram values as

$$C[i] = \sum_{k=0}^{i} H[k]$$

for i =0…255. Finally, for each pixel P, the algorithm computes a new value

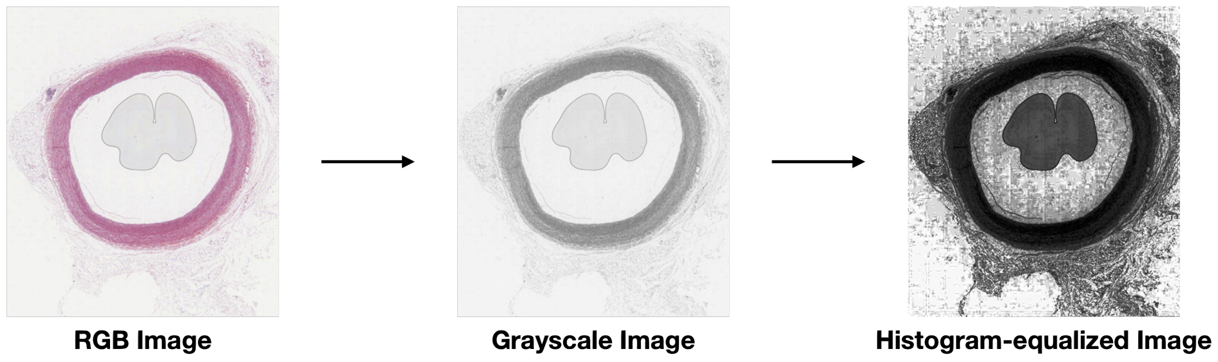
$$p\prime = 255 \cdot C[p].$$

The resulting image will have a uniform intensity distribution. The algorithm described is also called non-adaptive uniform histogram equalization, as it works uniformly on the whole image and the transformation of one pixel is independent from the transformation used on the neighboring pixels[4].

Notice that the histogram equalization method can be used for RGB images by applying the same algorithm on the R, G, and B channels separately[5]; nonetheless, the high correlation of the three channels may distort the image and the color balance can change drastically.

[4] T Strothotte and S Schlechtweg. “Non-photorealistic computer graphics: modeling, rendering, and animation”. Morgan Kaufmann (2002)

[5] Z Rong and et al. “Study of color heritage image enhancement algorithms based on histogram equalization”. Optik 126.24 (2015)



Parameters

- **img** (*PIL.Image.Image*) – Input image.
- **n_bins** (*int. optional*) – Number of histogram bins. Default is 256.

Returns Image with contrast enhanced by histogram equalization.

Return type PIL.Image.Image

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import HistogramEqualization, RgbToGrayscale
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> rgb_to_grayscale = RgbToGrayscale()
>>> histogram_equalization = HistogramEqualization()
>>> image_gray = rgb_to_grayscale(image_rgb)
>>> image_he = histogram_equalization(image_gray)
```

References

class HysteresisThreshold(**args*, ***kwds*)

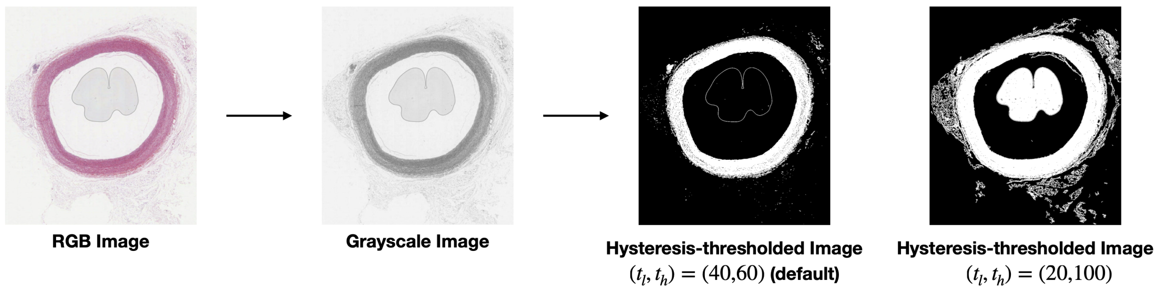
Apply two-level (hysteresis) threshold to an image. The hysteresis thresholding is a two-threshold method used to detect objects on an image, based on the assumption that points connected to an object are most likely objects themselves. In particular, pixels above a specified high threshold t_h are labelled as non-objects, and pixels $o \in [t_l, t_h]$ are defined as weak objects; all the non-objects are removed, while the weak objects are kept only if connected to a strong one. The hysteresis thresholding can be applied to detect edges in an image.

Parameters

- **img** (*PIL.Image.Image*) – Input image
- **low** (*int, optional*) – low threshold. Default is 50.
- **high** (*int, optional*) – high threshold. Default is 100

Returns Image with the hysteresis threshold applied

Return type PIL.Image.Image


RGB Image
Grayscale Image
Hysteresis-thresholded Image
$(t_l, t_h) = (40,60)$ (default)
Hysteresis-thresholded Image
$(t_l, t_h) = (20,100)$

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import HysteresisThreshold, RgbToGrayscale
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> rgb_to_grayscale = RgbToGrayscale()
>>> hyst_threshold = HysteresisThreshold(low=200, high=250)
>>> image_gray = rgb_to_grayscale(image_rgb)
>>> image_thresholded = hyst_threshold(image_gray)
```

class HysteresisThresholdMask(**args*, ***kwds*)

Mask an image using hysteresis threshold

Compute the Hysteresis threshold on the complement of a grayscale image, and return boolean mask based on pixels above this threshold.

Parameters

- **img** (*PIL.Image.Image*) – Input image.
- **low** (*int, optional*) – low threshold. Default is 50.
- **high** (*int, optional*) – high threshold. Default is 100.

Returns Boolean NumPy array where True represents a pixel above hysteresis threshold.

Return type np.ndarray

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import HysteresisThresholdMask,␣
↪RgbToGrayscale
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> rgb_to_grayscale = RgbToGrayscale()
>>> hyst_threshold_mask = HysteresisThresholdMask()
>>> image_gray = rgb_to_grayscale(image_rgb)
>>> image_thresholded_array = hyst_threshold_mask(image_gray)
```

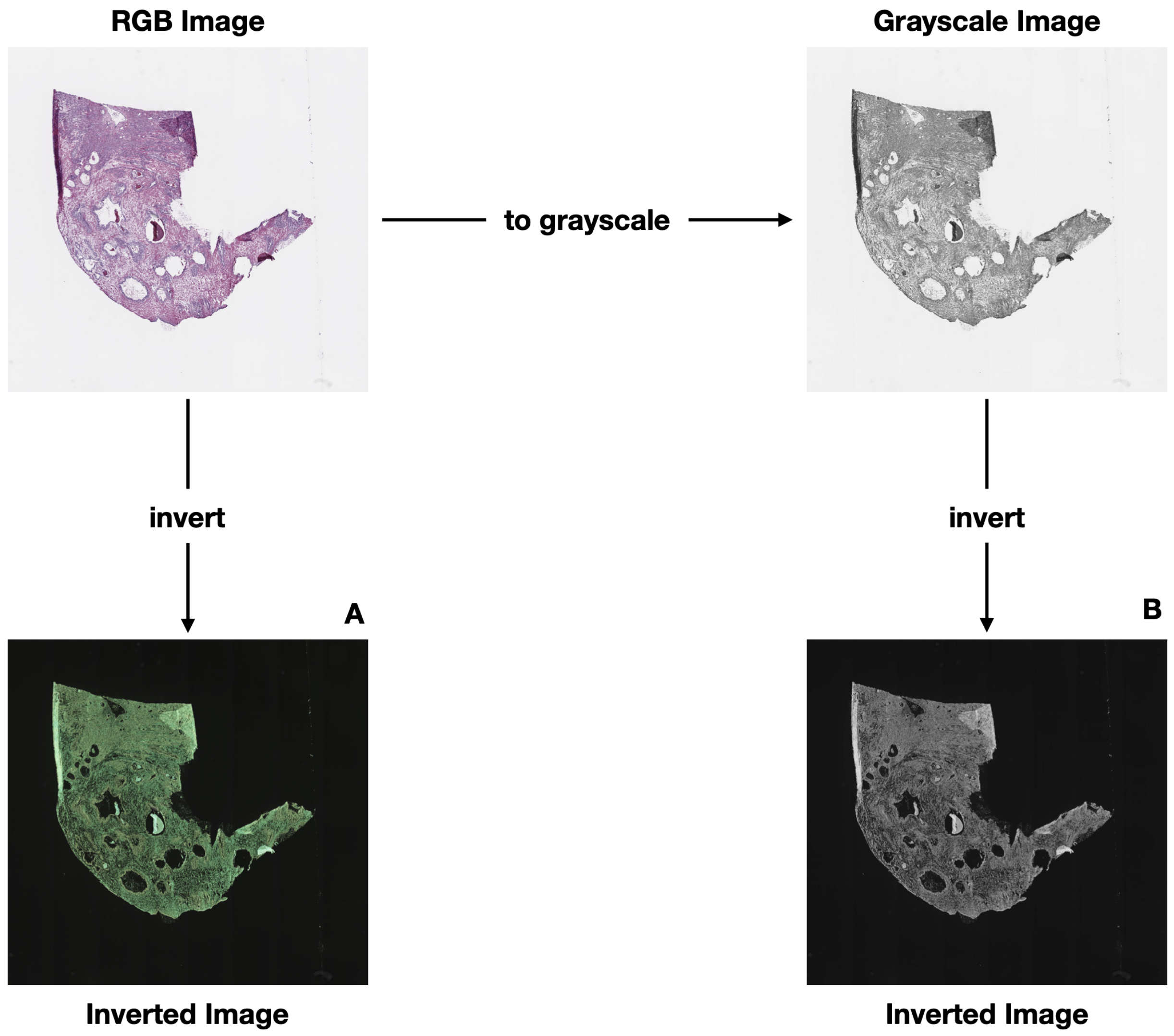
class ImageFilter(**args*, ***kwds*)

Image filter protocol

`class Invert(*args, **kwds)`

Invert an image, i.e. take the complement of the correspondent array.

For binary images, the inversion flips True and False values. For RGB images, each pixel value p is replaced with $\hat{p} - p$ where $\hat{p}$ is the maximum value of pixels of the data type (i.e. 255). Usually, the tissue in a WSI is surrounded by a white light background (values close to 255). Therefore, inverting its values could ease the removal of non-tissue regions (values close or equal to 0).



Parameters `img` (*PIL.Image.Image*) – Input image

Returns Inverted image

Return type PIL.Image.Image

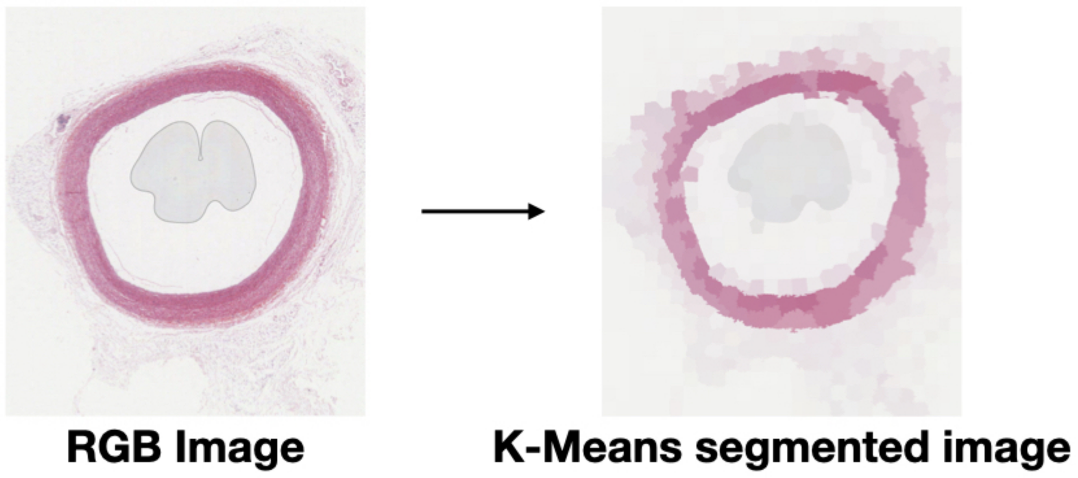
Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import Invert, RgbToGrayscale
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> rgb_to_grayscale = RgbToGrayscale()
>>> invert = Invert()
>>> image_gray = rgb_to_grayscale(image_rgb)
>>> image_inv_rgb = invert(image_rgb)
>>> image_inv_gray = invert(image_gray)
```

`class` **`KmeansSegmentation`**`(`*`*args`*`,` *`**kwds`*`)`

Segment an RGB image with K-means segmentation

By using K-means segmentation (color/space proximity) each segment is colored based on the average color for that segment.



Parameters

- **`img`** (*`PIL.Image.Image`*) – Input image
- **`n_segments`** (*`int, optional`*) – The number of segments. Default is 800.
- **`compactness`** (*`float, optional`*) – Color proximity versus space proximity factor. Default is 10.0.

Returns Image where each segment has been colored based on the average color for that segment.

Return type PIL.Image.Image

Raises **`ValueError`** – If `img` mode is RGBA.

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import KmeansSegmentation
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> kmeans_segmentation = KmeansSegmentation()
>>> kmeans_segmented_image = kmeans_segmentation(image_rgb)
```

`class LabToRgb`(**args*, ***kwds*)

Lab to RGB color space conversion.

Parameters

- **img** (*np.array*) – Input image in Lab space.
- **illuminant** (*{"A", "D50", "D55", "D65", "D75", "E"}, optional*) – The name of the illuminant (the function is NOT case sensitive). Default is "D65".
- **observer** (*{"2", "10"}, optional*) – The aperture angle of the observer. Default is "2".

Returns

- *PIL.Image.Image* – Image in RGB space.
- *Example* – >>> import numpy as np >>> from histolab.filters.image_filters import LabToRgb >>> arr_lab = np.load("tests/fixtures/arrays/diagnostic-slide-thumb-lab.npy") >>> lab_to_rgb = LabToRgb() >>> image_rgb = lab_to_rgb(arr_lab)

`class Lambda`(**args*, ***kwds*)

Apply a user-defined lambda as a filter.

Inspired from: https://pytorch.org/docs/stable/_modules/torchvision/transforms/transforms.html#Lambda

Parameters **lambd** (*callable*) – Lambda/function to be used as a filter.

Returns The image with the function applied.

Return type PIL.Image.Image

`class LocalEqualization`(**args*, ***kwds*)

Filter gray image using local equalization.

Local equalization method uses local histograms based on a disk structuring element.

Parameters

- **img** (*PIL.Image.Image*) – Grayscale input image
- **disk_size** (*int, optional*) – Radius of the disk structuring element used for the local histograms. Default is 50

Returns Grayscale image with contrast enhanced using local equalization.

Return type PIL.Image.Image

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import LocalEqualization
>>> image_rgb = Image.open("tests/fixtures/pil-images-gs/diagnostic-slide-thumb-gs.
↪png")
>>> local_equ = LocalEqualization()
>>> local_equ_image = local_equ(image_rgb)
```

class LocalOtsuThreshold(**args*, ***kwds*)

Mask image based on local Otsu threshold.

Compute Otsu threshold for each pixel and return the image thresholded locally.

Note that the input image must be 2D.

Parameters

- **img** (*PIL.Image.Image*) – Input 2-dimensional image
- **disk_size** (*float, optional*) – Radius of the disk structuring element used to compute the Otsu threshold for each pixel. Default is 3.0

Returns Image thresholded with the Otsu algorithm computed locally

Return type PIL.Image.Image

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import LocalOtsuThreshold, RgbToGrayscale
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> rgb_to_grayscale = RgbToGrayscale()
>>> local_otsu = LocalOtsuThreshold()
>>> image_gray = rgb_to_grayscale(image_rgb)
>>> image_thresholded_locally = local_otsu(image_gray)
```

class OtsuThreshold(**args*, ***kwds*)

Mask image based on pixel above Otsu threshold.

Compute Otsu threshold on image as a NumPy array and return boolean mask based on pixels above this threshold. The Otsu algorithm is a standard method to automatically compute the optimal threshold value to separate image background from the foreground[7]. In this filter, the pixels below the Otsu threshold are considered as foreground.

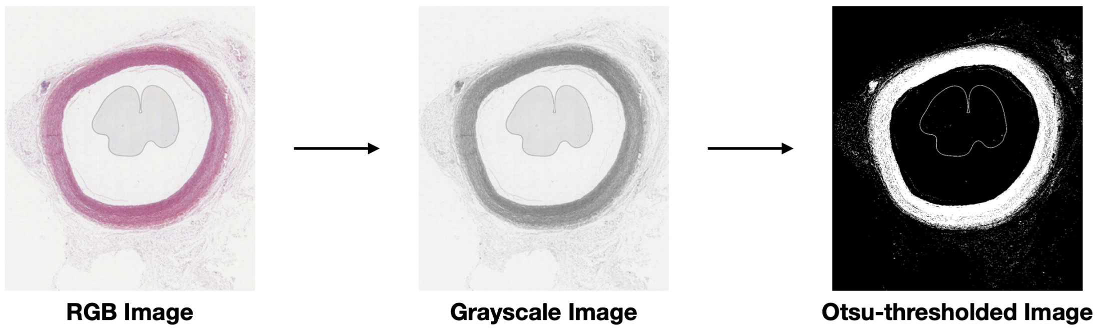
Note that Otsu threshold is expected to work correctly only for grayscale images.

Parameters **img** (*PIL.Image.Image*) – Input image.

Returns Boolean NumPy array where True represents a pixel above Otsu threshold.

Return type np.ndarray

[7] N Otsu. "A threshold selection method from gray-level histograms". IEEE Trans SystMan Cybern Syst 9.1 (1979)



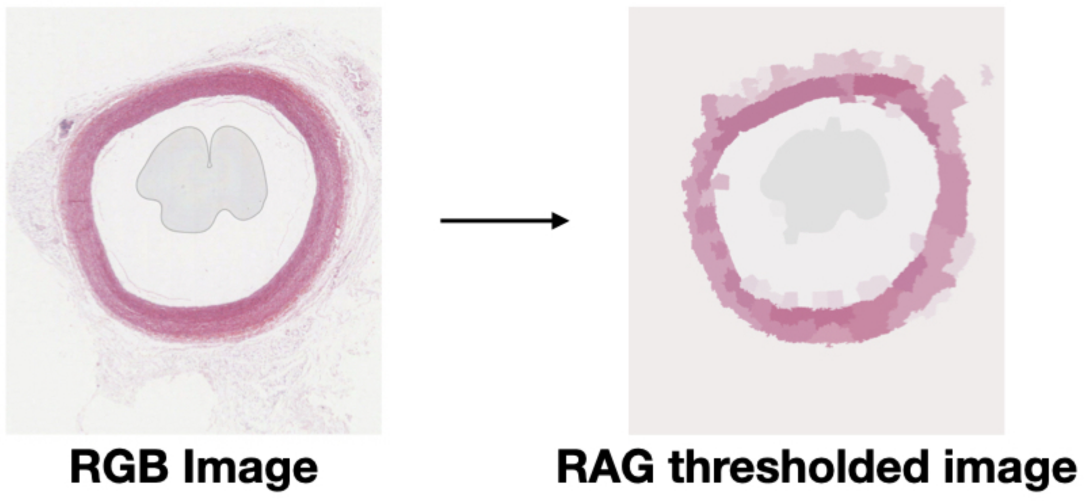
Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import OtsuThreshold, RgbToGrayscale
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> rgb_to_grayscale = RgbToGrayscale()
>>> otsu_threshold = OtsuThreshold()
>>> image_gray = rgb_to_grayscale(image_rgb)
>>> image_thresholded_array = otsu_threshold(image_gray)
```

class RagThreshold(**args*, ***kwds*)

Combine similar K-means segmented regions based on threshold value.

Segment an image with K-means, build region adjacency graph based on the segments, combine similar regions based on threshold value, and then output these resulting region segments.



Parameters

- **img** (*PIL.Image.Image*) – Input image
- **n_segments** (*int, optional*) – The number of segments. Default is 800.
- **compactness** (*float, optional*) – Color proximity versus space proximity factor. Default is 10.0
- **threshold** (*int, optional*) – Threshold value for combining regions. Default is 9.
- **return_labels** (*bool, optional*) – If True, returns a labeled array where the value denotes segment membership. Otherwise, returns a PIL image where each segment is colored by the average color in it. Default is False.

Returns

- PIL.Image.Image, if not `return_labels` – Each segment has been colored based on the average color for that segment (and similar segments have been combined).
- np.ndarray, if `return_labels` – Value denotes segment membership.

Raises

- **ValueError** – If `img` mode is RGBA.
- **Example:** –

```
>>> from PIL import Image
    >>> from histolab.filters.image_filters import RagThreshold
    >>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-
↪lung-rgb.png")
    >>> rag_threshold = RagThreshold()
    >>> rag_thresholded_array = rag_threshold(image_rgb)
```

`class` **`RedFilter`**(**args*, ***kwds*)

Mask reddish colors in an RGB image.

Create a mask to filter out reddish colors, where the mask is based on a pixel being above a red channel threshold value, below a green channel threshold value, and below a blue channel threshold value.

Parameters

- **img** (*PIl.Image.Image*) – Input RGB image
- **red_lower_thresh** (*int*) – Red channel lower threshold value.
- **green_upper_thresh** (*int*) – Green channel upper threshold value.
- **blue_upper_thresh** (*int*) – Blue channel upper threshold value.

Returns Boolean NumPy array representing the mask.

Return type np.ndarray

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import RedFilter
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-red-pen.png")
>>> red_filter = RedFilter(10, 30, 25)
>>> mask_filtered = red_filter(image_rgb)
```

class RedPenFilter(**args*, ***kwds*)

Filter out red pen marks on diagnostic slides.

The resulting mask is a composition of red filters with different thresholds for the RGB channels.

Parameters **img** (*PIL.Image.Image*) – Input RGB image.

Returns Image the green red marks filtered out.

Return type PIL.Image.Image

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import RedPenFilter
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-red-pen.png")
>>> red_pen_filter = RedPenFilter()
>>> image_no_red = red_pen_filter(image_rgb)
```

class RgbToGrayscale(**args*, ***kwds*)

Convert an RGB image to a grayscale image.

Parameters **img** (*PIL.Image.Image*) – Input image

Returns Grayscale image

Return type PIL.Image.Image

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import RgbToGrayscale
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> rgb_to_grayscale = RgbToGrayscale()
>>> image_gray = rgb_to_grayscale(image_rgb)
```

class RgbToHed(**args*, ***kwds*)

Convert RGB channels to HED channels.

image color space (RGB) is converted to Hematoxylin-Eosin-Diaminobenzidine space.

Parameters **img** (*PIL.Image.Image*) – Input image

Returns Array representation of the image in HED space

Return type np.ndarray

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import RgbToHed
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> rgb_to_hed = RgbToHed()
>>> image_hed = rgb_to_hed(image_rgb)
```

`class RgbToHsv`(**args*, ***kwds*)

Convert RGB channels to HSV channels.

image color space (RGB) is converted to Hue - Saturation - Value (HSV) space.

Parameters **`img`** (*`PIL.Image.Image`*) – Input image

Returns Array representation of the image in HSV space

Return type np.ndarray

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import RgbToHsv
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> rgb_to_hsv = RgbToHsv()
>>> image_hsv = rgb_to_hsv(image_rgb)
```

`class RgbToLab`(**args*, ***kwds*)

Convert from the sRGB color space to the CIE Lab colorspace.

sRGB color space reference: IEC 61966-2-1:1999

Parameters

- **`img`** (*`PIL.Image.Image`*) – Input image
- **`illuminant`** (*`{"A", "D50", "D55", "D65", "D75", "E"}, optional`*) – The name of the illuminant (the function is NOT case sensitive).
- **`observer`** (*`{"2", "10"}, optional`*) – The aperture angle of the observer.

Returns Array representation of the image in LAB space

Return type np.ndarray

Raises **`Exception`** – If the `img` mode is not RGB

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import RgbToLab
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> rgb_to_lab = RgbToLab()
>>> image_lab = rgb_to_lab(image_rgb)
```

`class RgbToOd`(**args*, ***kwds*)

Convert from RGB to optical density (OD_RGB) space.

Parameters **img** (*PIL.Image.Image*) – Input image

Returns Array representation of the image in OD space

Return type np.ndarray

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import RgbToOd
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> rgb_to_od = RgbToOd()
>>> image_od = rgb_to_od(image_rgb)
```

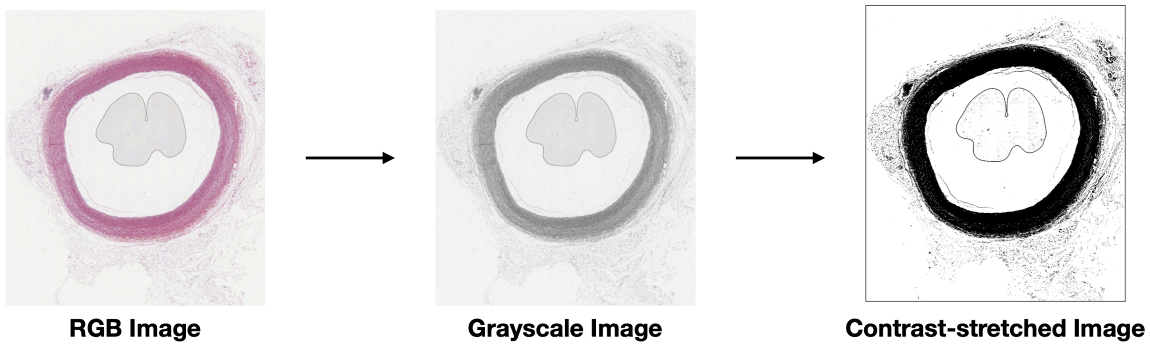
class StretchContrast(**args*, ***kwds*)

Increase image contrast.

A simple way to enhance the contrast in an image is to linearly rescale the intensity values within a desired range $[v_{o,l}, v_{o,h}]$. In particular, if the lowest and highest pixel values of the input image are, respectively, $v_{i,l}$ and $v_{i,h}$, an input pixel p_i is remapped to the output pixel value:

$$p_o = (p_i - v_{i,l})\left(\frac{v_{o,h} - v_{o,l}}{v_{i,h} - v_{i,l}}\right) + v_{o,l}$$

The Stretch Contrast filter stretches the intensity values in an image, with $v_{o,l} = 40$ and $v_{o,l} = 60$ as default values. This filter is useful to highlight details in the input image.



Parameters

- **img** (*PIL.Image.Image*) – Input image
- **low** (*int, optional*) – Range low value (0 to 255). Default is 40.
- **high** (*int, optional*) – Range high value (0 to 255). Default is 60

Returns Image with contrast enhanced.

Return type PIL.Image.Image

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import RgbToGrayscale, StretchContrast
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> rgb_to_grayscale = RgbToGrayscale()
>>> stretch_contrast = StretchContrast()
>>> image_gray = rgb_to_grayscale(image_rgb)
>>> image_stretched = stretch_contrast(image_gray)
```

`class ToPILImage(*args, **kwds)`

Convert a ndarray to a PIL Image, while preserving the value range.

Parameters `np_img` (*np.ndarray*) – The image represented as a NumPy array.

Returns The image represented as PIL Image

Return type PIL.Image.Image

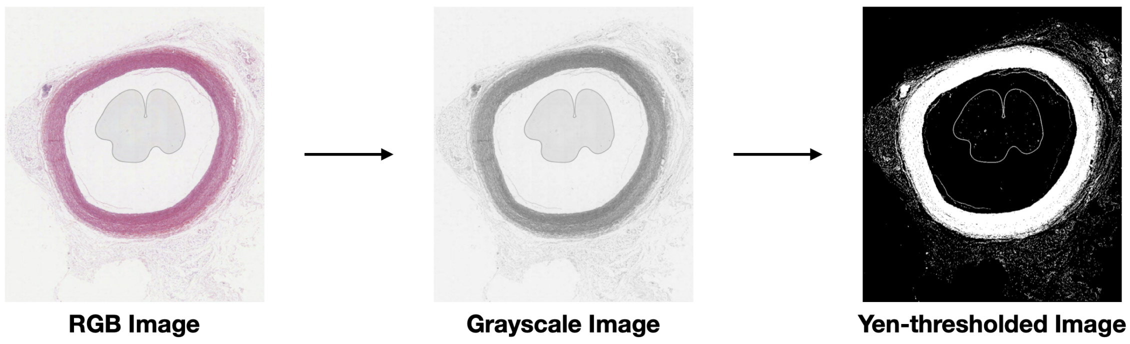
`class YenThreshold(*args, **kwds)`

Mask image based on pixel above Yen threshold.

Compute Yen threshold on image and return boolean mask based on pixels below this threshold. The Yen method[8] is a multi-level image thresholding approach to separate objects from the background. It automatically computes the threshold that maximize the entropic correlation EC for a given gray level s defined as:

$$EC(s) = -\ln\left(G(s) \cdot G'(s)\right) + 2\ln(P(s) \cdot (1 - P(s))$$

where $G(s) = \sum_{i=0}^{s-1} p_i^2$, $G'(s) = \sum_{i=s}^{m-1} p_i^2$, m is the number of gray levels in the image, p_i is the probability of the gray level i and $P(s) = \sum_{i=0}^{s-1} p_i$ is the total probability up to gray level (s-1). In this filter, pixels below the computed threshold are considered as foreground.



Parameters

- `img` (*PIL.Image.Image*) – Input image.

[8] J.C. Yen and et al."A new criterion for automatic multilevel thresholding". IEEE Trans Image Process 4.3 (1995)

- **relate** (*operator, optional*) – Operator to be used to compute the mask from the threshold. Default is operator.lt

Returns Boolean NumPy array where True represents a pixel below Yen's threshold.

Return type np.ndarray

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import RgbToGrayscale, YenThreshold
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> rgb_to_grayscale = RgbToGrayscale()
>>> yen_threshold = YenThreshold()
>>> image_gray = rgb_to_grayscale(image_rgb)
>>> image_thresholded_array = yen_threshold(image_gray)
```

References

6.8.2 Image Filters Functional

adaptive_equalization(*img*, *nbins=256*, *clip_limit=0.01*)

Increase image contrast using adaptive equalization.

Contrast in local region of input image (gray or RGB) is increased using adaptive equalization

Parameters

- **img** (*PIL.Image.Image*) – Input image (gray or RGB)
- **nbins** (*int*) – Number of histogram bins. Default is 256.
- **clip_limit** (*float, optional*) – Clipping limit where higher value increases contrast. Default is 0.01

Returns image with contrast enhanced by adaptive equalization.

Return type PIL.Image.Image

blue_filter(*img*, *red_thresh*, *green_thresh*, *blue_thresh*)

Filter out blueish colors in an RGB image.

Create a mask to filter out blueish colors, where the mask is based on a pixel being above a red channel threshold value, above a green channel threshold value, and below a blue channel threshold value.

Parameters

- **img** (*PIL.Image.Image*) – Input RGB image
- **red_thresh** (*int*) – Red channel lower threshold value.
- **green_thresh** (*int*) – Green channel lower threshold value.
- **blue_thresh** (*int*) – Blue channel upper threshold value.

Returns Boolean NumPy array representing the mask.

Return type np.ndarray

`blue_pen_filter`(*img*)

Filter out blue pen marks from a diagnostic slide.

The resulting mask is a composition of green filters with different thresholds for the RGB channels.

Parameters `img` (*PIL.Image.Image*) – Input RGB image

Returns Input image with the blue pen marks filtered out.

Return type PIL.Image.Image

`canny_edges`(*img*, *sigma=1.0*, *low_threshold=0.0*, *high_threshold=25.0*)

Filter image based on Canny edge algorithm.

Note that input image must be 2D grayscale image

Parameters

- `img` (*PIL.Image.Image*) – Input 2-dimensional image
- `sigma` (*float, optional*) – Width (std dev) of Gaussian. Default is 1.0.
- `low_threshold` (*float, optional*) – Low hysteresis threshold value. Default is 0.0.
- `high_threshold` (*float, optional*) – High hysteresis threshold value. Default is 25.0.

Returns Boolean NumPy array representing Canny edge map.

Return type np.ndarray

`dab_channel`(*img*)

Obtain DAB channel from RGB image.

Input image is first converted into HED space and the DAB channel is extracted via color deconvolution.

Parameters `img` (*PIL.Image.Image*) – Input RGB image

Returns RGB image with DAB staining separated.

Return type PIL.Image.Image

`eosin_channel`(*img*)

Obtain Eosin channel from RGB image.

Input image is first converted into HED space and the Eosin channel is extracted via color deconvolution.

Parameters `img` (*PIL.Image.Image*) – Input RGB image

Returns RGB image with Eosin staining separated.

Return type PIL.Image.Image

`filter_entropy`(*img*, *neighborhood=9*, *threshold=5.0*, *relate=<built-in function gt>*)

Filter image based on entropy (complexity).

The area of the image included in the local neighborhood is defined by a square neighborhood x neighborhood

Note that input must be 2D.

Parameters

- `img` (*PIL.Image.Image*) – input 2-dimensional image
- `neighborhood` (*int, optional*) – Neighborhood size (defines height and width of 2D array of 1's). Default is 9.
- `threshold` (*float, optional*) – Threshold value. Default is 5.0

- **relate** (*callable operator, optional*) – Operator to be used to compute the mask from the threshold. Default is operator.lt

Returns NumPy boolean array where True represent a measure of complexity.

Return type np.ndarray

`grays`(*img*, *tolerance=15*)

Filter out gray pixels in RGB image.

Gray pixels are those pixels where the red, green, and blue channel values are similar, i.e. under a specified tolerance.

Parameters

- **img** (*PIL.Image.Image*) – Input image
- **tolerance** (*int, optional*) – if difference between values is below this threshold, values are considered similar and thus filtered out. Default is 15.

Returns Mask image where the grays values are masked out

Return type PIL.Image.Image

`green_channel_filter`(*img*, *green_thresh=200*, *avoid_overmask=True*, *overmask_thresh=90.0*)

Mask pixels in an RGB image with G-channel greater than a specified threshold.

Create a mask to filter out pixels with a green channel value greater than a particular threshold, since hematoxylin and eosin are purplish and pinkish, which do not have much green to them.

Parameters

- **img** (*PIL.Image.Image*) – Input RGB image
- **green_thresh** (*int, optional*) – Green channel threshold value (0 to 255). Default is 200. If value is greater than green_thresh, mask out pixel.
- **avoid_overmask** (*bool, optional*) – If True, avoid masking above the overmask_thresh percentage. Default is True.
- **overmask_thresh** (*float, optional*) – If avoid_overmask is True, avoid masking above this percentage value. Default is 90.0

Returns Boolean mask where pixels above a particular green channel threshold have been masked out.

Return type np.ndarray

`green_filter`(*img*, *red_thresh*, *green_thresh*, *blue_thresh*)

Filter out greenish colors in an RGB image. The mask is based on a pixel being above a red channel threshold value, below a green channel threshold value, and below a blue channel threshold value.

Note that for the green ink, the green and blue channels tend to track together, so for blue channel we use a lower threshold rather than an upper threshold value.

Parameters

- **img** (*PIL.Image.Image*) – RGB input image.
- **red_thresh** (*int*) – Red channel upper threshold value.
- **green_thresh** (*int*) – Green channel lower threshold value.
- **blue_thresh** (*int*) – Blue channel lower threshold value.

Returns Boolean NumPy array representing the mask.

Return type np.ndarray

`green_pen_filter`(*img*)

Filter out green pen marks from a diagnostic slide.

The resulting mask is a composition of green filters with different thresholds for the RGB channels.

Parameters `img` (*PIL.Image.Image*) – Input RGB image

Returns Input image with the green pen marks filtered out.

Return type PIL.Image.Image

`hed_to_rgb`(*img_arr*)

Convert HED channels to RGB channels.

Parameters `img_arr` (*np.ndarray*) – Array representation of the image in HED color space

Returns Image in RGB space

Return type PIL.Image.Image

`hematoxylin_channel`(*img*)

Obtain Hematoxylin channel from RGB image.

Input image is first converted into HED space and the hematoxylin channel is extracted via color deconvolution.

Parameters `img` (*Image.Image*) – Input RGB image

Returns RGB image with Hematoxylin staining separated.

Return type PIL.Image.Image

`histogram_equalization`(*img*, *nbins=256*)

Increase image contrast using histogram equalization.

The input image (gray or RGB) is filterd using histogram equalization to increase contrast.

Parameters

- `img` (*PIL.Image.Image*) – Input image.
- `nbins` (*int. optional*) – Number of histogram bins. Default is 256.

Returns Image with contrast enhanced by histogram equalization.

Return type PIL.Image.Image

`hysteresis_threshold`(*img*, *low=50*, *high=100*)

Apply two-level (hysteresis) threshold to an image.

Parameters

- `img` (*PIL.Image.Image*) – Input image
- `low` (*int, optional*) – low threshold. Default is 50.
- `high` (*int, optional*) – high threshold. Default is 100.

Returns Image with the hysteresis threshold applied

Return type PIL.Image.Image

`hysteresis_threshold_mask`(*img*, *low=50*, *high=100*)

Mask an image using hysteresis threshold

Compute the Hysteresis threshold on the complement of a grayscale image, and return boolean mask based on pixels above this threshold.

Parameters

- `img` (*PIL.Image.Image*) – Input image.
- `low` (*int, optional*) – low threshold. Default is 50.
- `high` (*int, optional*) – high threshold. Default is 100.

Returns Boolean NumPy array where True represents a pixel above Otsu threshold.

Return type np.ndarray

`invert`(*img*)

Invert an image, i.e. take the complement of the correspondent array.

Parameters `img` (*PIL.Image.Image*) – Input image

Returns Inverted image

Return type PIL.Image.Image

`kmeans_segmentation`(*img*, *n_segments=800*, *compactness=10.0*)

Segment an image with K-means segmentation

By using K-means segmentation (color/space proximity) each segment is colored based on the average color for that segment.

Parameters

- `img` (*PIL.Image.Image*) – Input image
- `n_segments` (*int, optional*) – The number of segments. Default is 800.
- `compactness` (*float, optional*) – Color proximity versus space proximity factor. Default is 10.0.

Returns RGB image where each segment has been colored based on the average color for that segment.

Return type PIL.Image.Image

Raises `ValueError` – If `img` mode is RGBA.

`lab_to_rgb`(*img*, *illuminant='D65'*, *observer='2'*)

Lab to RGB color space conversion.

Parameters

- `img` (*PIL.Image.Image*) – Input image in Lab space.
- `illuminant` (*{"A", "D50", "D55", "D65", "D75", "E"}, optional*) – The name of the illuminant (the function is NOT case sensitive). Default is "D65".
- `observer` (*{"2", "10"}, optional*) – The aperture angle of the observer. Default is "2".

Returns Image in RGB space.

Return type PIL.Image.Image

`local_equalization`(*img*, *disk_size=50*)

Filter gray image using local equalization.

Local equalization method uses local histograms based on a disk structuring element.

Parameters

- `img` (*PIL.Image.Image*) – Input image. Notice that it must be 2D

- **disk_size** (*int, optional*) – Radius of the disk structuring element used for the local histograms. Default is 50.

Returns 2D image with contrast enhanced using local equalization.

Return type PIL.Image.Image

`local_otsu_threshold`(*img*, *disk_size=3.0*)
Mask image based on local Otsu threshold.

Compute local Otsu threshold for each pixel and return boolean mask based on pixels being less than the local Otsu threshold.

Note that the input image must be 2D.

Parameters

- **img** (*PIL.Image.Image*) – Input 2-dimensional image
- **disk_size** (*float, optional*) – Radius of the disk structuring element used to compute the Otsu threshold for each pixel. Default is 3.0.

Returns Resulting image where local Otsu threshold values have been applied to original image.

Return type PIL.Image.Image

`otsu_threshold`(*img*, *relate=<built-in function lt>*)
Mask image based on pixel above Otsu threshold.

Compute Otsu threshold on image and return boolean mask based on pixels above this threshold.

Note that Otsu threshold is expected to work correctly only for grayscale images.

Parameters

- **img** (*PIL.Image.Image*) – Input image.
- **relate** (*operator, optional*) – Operator to be used to compute the mask from the threshold. Default is operator.lt

Returns Boolean NumPy array where True represents a pixel above Otsu threshold.

Return type np.ndarray

`rag_threshold`(*img*, *n_segments=800*, *compactness=10.0*, *threshold=9*, *mask=None*, *return_labels=False*)
Combine similar K-means segmented regions based on threshold value.

Segment an image with K-means, build region adjacency graph based on the segments, combine similar regions based on threshold value, and then output these resulting region segments.

Parameters

- **img** (*PIL.Image.Image*) – Input image
- **n_segments** (*int, optional*) – The number of segments. Default is 800.
- **compactness** (*float, optional*) – Color proximity versus space proximity factor. Default is 10.0.
- **threshold** (*int, optional*) – Threshold value for combining regions. Default is 9.
- **mask** (*np.ndarray, optional*) – If provided, superpixels are computed only where mask is True, and seed points are homogeneously distributed over the mask using a K-means clustering strategy (See skimage). Must be the same size as `img`.

- **return_labels** (*bool, optional*) – If True, returns a labeled array where the value denotes segment membership. Otherwise, returns a PIL image where each segment is colored by the average color in it. Default is False.

Returns

- PIL.Image.Image, if not `return_labels` – Each segment has been colored based on the average color for that segment (and similar segments have been combined).
- np.ndarray, if `return_labels` – Value denotes segment membership.

Raises **ValueError** – If `img` mode is RGBA.

Return type Union[PIL.Image.Image, numpy.ndarray]

red_filter(*img*, *red_thresh*, *green_thresh*, *blue_thresh*)

Mask reddish colors in an RGB image.

Create a mask to filter out reddish colors, where the mask is based on a pixel being above a red channel threshold value, below a green channel threshold value, and below a blue channel threshold value.

Parameters

- **img** (*PIL.Image.Image*) – Input RGB image
- **red_thresh** (*int*) – Red channel lower threshold value.
- **green_thresh** (*int*) – Green channel upper threshold value.
- **blue_thresh** (*int*) – Blue channel upper threshold value.

Returns Boolean NumPy array representing the mask.

Return type np.ndarray

red_pen_filter(*img*)

Filter out red pen marks on diagnostic slides.

The resulting mask is a composition of red filters with different thresholds for the RGB channels.

Parameters **img** (*PIL.Image.Image*) – Input RGB image.

Returns Input image with the pen marks filtered out.

Return type PIL.Image.Image

rgb_to_hed(*img*)

Convert RGB channels to HED channels.

Image color space (RGB) is converted to Hematoxylin-Eosin-Diaminobenzidine space.

Parameters **img** (*PIL.Image.Image*) – Input image

Returns Array representation of the image in HED space

Return type np.ndarray

rgb_to_hsv(*img*)

Convert RGB channels to HSV channels.

Image color space (RGB) is converted to Hue - Saturation - Value (HSV) space.

Parameters **img** (*PIL.Image.Image*) – Input image

Returns Array representation of the image in HSV space

Return type np.ndarray

Raises **Exception** – If the image mode is not RGB

`rgb_to_lab`(*img*, *illuminant='D65'*, *observer='2'*)

Convert from the sRGB color space to the CIE Lab colorspace.

sRGB color space reference: IEC 61966-2-1:1999

Parameters

- `img` (*PIL.Image.Image*) – Input image
- `illuminant` (*{"A", "D50", "D55", "D65", "D75", "E"}, optional*) – The name of the illuminant (the function is NOT case sensitive). Default is "D65".
- `observer` (*{"2", "10"}, optional*) – The aperture angle of the observer. Default is "2".

Returns Array representation of the image in LAB space

Return type np.ndarray

Raises `Exception` – If the image mode is not RGB

`rgb_to_od`(*img*)

Convert from RGB to optical density (OD_RGB) space.

Parameters `img` (*PIL.Image.Image*) – Input image

Returns Array representation of the image in OD space

Return type np.ndarray

`stretch_contrast`(*img*, *low=40*, *high=60*)

Increase image contrast.

Th contrast in image is increased based on intensities in a specified range

Parameters

- `img` (*PIL.Image.Image*) – Input image
- `low` (*int*) – Range low value (0 to 255). Default is 40.
- `high` (*int*) – Range high value (0 to 255). Default is 60.

Returns Image with contrast enhanced.

Return type PIL.Image.Image

`yen_threshold`(*img*, *relate=<built-in function lt>*)

Mask image based on pixel below Yen's threshold.

Compute Yen threshold on image and return boolean mask based on pixels below this threshold.

Parameters

- `img` (*PIL.Image.Image*) – Input image.
- `relate` (*operator, optional*) – Operator to be used to compute the mask from the threshold. Default is operator.lt

Returns Boolean NumPy array where True represents a pixel below Yen's threshold.

Return type np.ndarray

6.8.3 Morphological Filters

class **BinaryClosing**(**args*, ***kwds*)

Close a binary mask.

Closing is a dilation followed by an erosion. Closing can be used to remove small holes.

Parameters

- **np_mask** (*np.ndarray (arbitrary shape, int or bool type)*) – Numpy array of the binary mask
- **disk_size** (*int, optional (default is 3)*) – Radius of the disk structuring element used for closing.
- **iterations** (*int, optional (default is 1)*) – How many times to repeat the closing.

Returns Mask after the closing

Return type np.ndarray

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import RgbToGrayscale, OtsuThreshold
>>> from histolab.filters.morphological_filters import BinaryClosing
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> rgb_to_grayscale = RgbToGrayscale()
>>> otsu_threshold = OtsuThreshold()
>>> binary_closing = BinaryClosing()
>>> image_gray = rgb_to_grayscale(image_rgb)
>>> binary_image = otsu_threshold(image_gray)
>>> image_closed = binary_closing(binary_image)
```

class **BinaryDilation**(**args*, ***kwds*)

Dilate a binary mask.

Parameters

- **np_mask** (*np.ndarray (arbitrary shape, int or bool type)*) – Numpy array of the binary mask
- **disk_size** (*int, optional (default is 5)*) – Radius of the disk structuring element used for dilation.
- **iterations** (*int, optional (default is 1)*) – How many times to repeat the dilation.

Returns Mask after the dilation

Return type np.ndarray

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import RgbToGrayscale, OtsuThreshold
>>> from histolab.filters.morphological_filters import BinaryDilation
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> rgb_to_grayscale = RgbToGrayscale()
>>> otsu_threshold = OtsuThreshold()
>>> binary_dilation = BinaryDilation()
>>> image_gray = rgb_to_grayscale(image_rgb)
>>> binary_image = otsu_threshold(image_gray)
>>> image_dilated = binary_dilation(binary_image)
```

`class BinaryErosion`(**args*, ***kwds*)

Erode a binary mask.

Parameters

- **`np_mask`** (*`np.ndarray (arbitrary shape, int or bool type)`*) – Numpy array of the binary mask
- **`disk_size`** (*`int, optional (default is 5)`*) – Radius of the disk structuring element used for erosion.
- **`iterations`** (*`int, optional (default is 1)`*) – How many times to repeat the erosion.

Returns Mask after the erosion

Return type np.ndarray

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import RgbToGrayscale, OtsuThreshold
>>> from histolab.filters.morphological_filters import BinaryErosion
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> rgb_to_grayscale = RgbToGrayscale()
>>> otsu_threshold = OtsuThreshold()
>>> binary_erosion = BinaryErosion(disk_size=6)
>>> image_gray = rgb_to_grayscale(image_rgb)
>>> binary_image = otsu_threshold(image_gray)
>>> image_eroded = binary_erosion(binary_image)
```

`class BinaryFillHoles`(**args*, ***kwds*)

Fill the holes in binary objects.

Parameters

- **`np_img`** (*`np.ndarray (arbitrary shape, int or bool type)`*) – Numpy array of the binary mask
- **`structure`** (*`np.ndarray, optional`*) – Structuring element used in the computation; The default element yields the intuitive result where all holes in the input have been filled.

Returns Transformation of the initial image input where holes have been filled.

Return type np.ndarray

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import RgbToGrayscale, OtsuThreshold
>>> from histolab.filters.morphological_filters import BinaryFillHoles
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> rgb_to_grayscale = RgbToGrayscale()
>>> otsu_threshold = OtsuThreshold()
>>> binary_fill_holes = BinaryFillHoles()
>>> image_gray = rgb_to_grayscale(image_rgb)
>>> binary_image = otsu_threshold(image_gray)
>>> image_filled_holes = binary_fill_holes(binary_image)
```

`class BinaryOpening`(**args*, ***kwds*)

Open a binary mask.

Opening is an erosion followed by a dilation. Opening can be used to remove small objects.

Parameters

- **np_mask** (*np.ndarray (arbitrary shape, int or bool type)*) – Numpy array of the binary mask
- **disk_size** (*int, optional (default is 3)*) – Radius of the disk structuring element used for opening.
- **iterations** (*int, optional (default is 1)*) – How many times to repeat the opening.

Returns Mask after the opening

Return type np.ndarray

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import RgbToGrayscale, OtsuThreshold
>>> from histolab.filters.morphological_filters import BinaryOpening
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> rgb_to_grayscale = RgbToGrayscale()
>>> otsu_threshold = OtsuThreshold()
>>> binary_opening = BinaryOpening()
>>> image_gray = rgb_to_grayscale(image_rgb)
>>> binary_image = otsu_threshold(image_gray)
>>> image_opened = binary_opening(binary_image)
```

`class MorphologicalFilter`(**args*, ***kwds*)

Morphological filter base class

`class RemoveSmallHoles`(**args*, ***kwds*)

Remove holes smaller than a specified size.

Parameters

- **np_img** (*np.ndarray (arbitrary shape, int or bool type)*) – Input mask
- **area_threshold** (*int, optional (default is 3000)*) – Remove small holes below this size.

Returns Mask with small holes filtered out

Return type np.ndarray

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import RgbToGrayscale, OtsuThreshold
>>> from histolab.filters.morphological_filters import RemoveSmallHoles
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> rgb_to_grayscale = RgbToGrayscale()
>>> otsu_threshold = OtsuThreshold()
>>> remove_small_holes = RemoveSmallHoles()
>>> image_gray = rgb_to_grayscale(image_rgb)
>>> binary_image = otsu_threshold(image_gray)
>>> image_no_small_holes = remove_small_holes(binary_image)
```

`class RemoveSmallObjects(*args, **kwds)`

Remove objects smaller than the specified size.

If avoid_overmask is True, this function can recursively call itself with progressively halved minimum size objects to avoid removing too many objects in the mask.

Parameters

- **np_img** (*np.ndarray (arbitrary shape, int or bool type)*) – Input mask
- **min_size** (*int, optional*) – Minimum size of small object to remove. Default is 3000
- **avoid_overmask** (*bool, optional (default is True)*) – If True, avoid masking above the overmask_thresh percentage.
- **overmask_thresh** (*int, optional (default is 95)*) – If avoid_overmask is True, avoid masking above this threshold percentage value.

Returns Mask with small objects filtered out

Return type np.ndarray

Example

```
>>> from PIL import Image
>>> from histolab.filters.image_filters import RgbToGrayscale, OtsuThreshold
>>> from histolab.filters.morphological_filters import RemoveSmallObjects
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> rgb_to_grayscale = RgbToGrayscale()
>>> otsu_threshold = OtsuThreshold()
>>> remove_small_objects = RemoveSmallObjects()
>>> image_gray = rgb_to_grayscale(image_rgb)
>>> binary_image = otsu_threshold(image_gray)
>>> image_no_small_objects = remove_small_objects(binary_image)
```

`class WatershedSegmentation(*args, **kwds)`

Segment and label an binary mask with Watershed segmentation[1]

The watershed algorithm treats pixels values as a local topography (elevation).

[1] Watershed segmentation. https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_watershed.html

Parameters

- **np_mask** (*np.ndarray*) – Input mask
- **region_shape** (*int, optional*) – The local region within which to search for image peaks is defined as a squared area region_shape x region_shape. Default is 6.

Returns Labelled segmentation mask

Return type np.ndarray

References

Example

```
>>> import numpy as np
>>> from histolab.filters.morphological_filters import WatershedSegmentation
>>> mask = np.array([[0,1],[1,0]]) # or np.load("/path/my_array_mask.npy")
>>> watershed_segmentation = WatershedSegmentation()
>>> mask_segmented = watershed_segmentation(mask)
```

class **WhiteTopHat**(**args*, ***kwds*)

Return white top hat of an image.

The white top hat of an image is defined as the image minus its morphological opening with respect to a structuring element. This operation returns the bright spots of the image that are smaller than the structuring element.

Parameters

- **np_mask** (*np.ndarray (arbitrary shape, int or bool type)*) – Numpy array of the binary mask
- **structure** (*np.ndarray, optional*) – The neighborhood expressed as an array of 1 and 0. If None, use cross-shaped structuring element (connectivity=1).

Example

```
>>> from PIL import Image
>>> import numpy as np
>>> from histolab.filters.image_filters import RgbToGrayscale, OtsuThreshold
>>> from histolab.filters.morphological_filters import WhiteTopHat
>>> image_rgb = Image.open("tests/fixtures/pil-images-rgb/tcga-lung-rgb.png")
>>> rgb_to_grayscale = RgbToGrayscale()
>>> otsu_threshold = OtsuThreshold()
>>> white_that = WhiteTopHat(np.ones((5,5)))
>>> image_gray = rgb_to_grayscale(image_rgb)
>>> binary_image = otsu_threshold(image_gray)
>>> image_out = white_that(binary_image)
```

6.8.4 Morphological Filters Functional

`remove_small_objects`(*np_mask*, *min_size=3000*, *avoid_overmask=True*, *overmask_thresh=95*)

Remove connected components which size is less than min_size.

is True, this function can recursively call itself with progressively to avoid removing too many objects in the mask.

Parameters

- **np_img** (*np.ndarray (arbitrary shape, int or bool type)*) – Input mask
- **min_size** (*int, optional*) – Minimum size of small object to remove. Default is 3000
- **avoid_overmask** (*bool, optional (default is True)*) – If True, avoid masking above the overmask_thresh percentage.
- **overmask_thresh** (*int, optional (default is 95)*) – If avoid_overmask is True, avoid masking above this threshold percentage value.
- **np_mask** (*numpy.ndarray*) –

Returns Mask with small objects filtered out

Return type np.ndarray

`watershed_segmentation`(*np_mask*, *region_shape=6*)

Segment and label an binary mask with Watershed segmentation[1]

The watershed algorithm treats pixels values as a local topography (elevation).

Parameters

- **np_mask** (*np.ndarray*) – Input mask
- **region_shape** (*int, optional*) – The local region within which to search for image peaks is defined as a squared area region_shape x region_shape. Default is 6.

Returns Labelled segmentation mask

Return type np.ndarray

References

6.8.5 Compositions

`class FiltersComposition`(*cls_*)

Provide appropriate filters compositions based on the `cls_` parameter.

Parameters **cls** (*type, {Tile, Slide}*) – The class to get the appropriate filters composition for

[1] Watershed segmentation. https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_watershed.html

Example

```
>>> from histolab.filters.compositions import FiltersComposition
>>> from histolab.slide import Slide
>>> from histolab.tile import Tile
>>> filters_slide = FiltersComposition(Slide).tissue_mask_filters
>>> filters_tile = FiltersComposition(Tile).tissue_mask_filters
```

property tissue_mask_filters: *histolab.filters.image_filters.Compose*

Return filters composition based on the `cls_` parameter.

Returns

If the `cls_` parameter is the class `Slide` the returned filters chain is composed of:

- image_filters.RgbToGrayscale()
- image_filters.OtsuThreshold()
- morphological_filters.BinaryDilation()
- morphological_filters.RemoveSmallHoles()
- morphological_filters.RemoveSmallObjects()

If the `cls_` parameter is the class `Tile` the returned filters chain is composed of:

- image_filters.RgbToGrayscale()
- image_filters.OtsuThreshold()
- morphological_filters.BinaryDilation()
- morphological_filters.BinaryFillHoles(structure=np.ones((5, 5)))

Return type imf.Compose

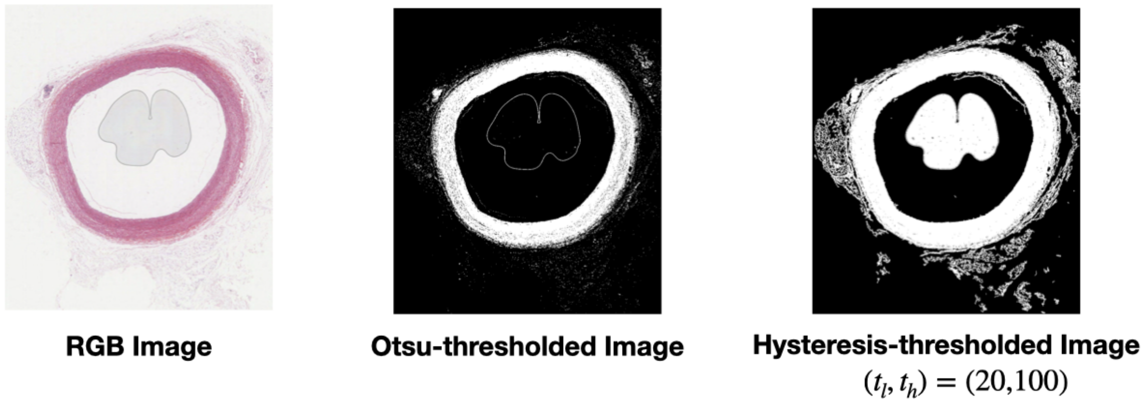
- **Image filters:**
  - **Transforming image color space**: Color images can be represented using alternative color spaces and the most common one is the RGB space, where the image is represented using distinct channels for Red, Green and Blue. RGB images can be converted to grayscale, namely shifting from 3-channel images to single channel images, e.g. for use with thresholding. `histolab` leverages the Pillow's ImageOps module for the conversion to the grayscale color space. Besides RGB and grayscale, several color models are extensively used, such as the HSV space, a cylindrical color model where the three independent channels represent the Hue, Saturation and Value of the color. The HED space[1] has been designed to specifically represent the contribution of Hematoxylin, Eosin and Diaminobenzidine dyes in H&E-stained slides and therefore is widely used in pathology (for example, to analyse microscopic blood images[2]).

  - **Threshold-based filters**: Thresholding is used to compute a binary mask from a grayscale image: the pixels above (or below) a specified threshold become True values, False otherwise. Color images can also be thresholded, using a different cut-off value for each color channel, and then combining the results using a $\wedge$ (logical AND) or a $\vee$ (logical OR) operator. `histolab` implements different

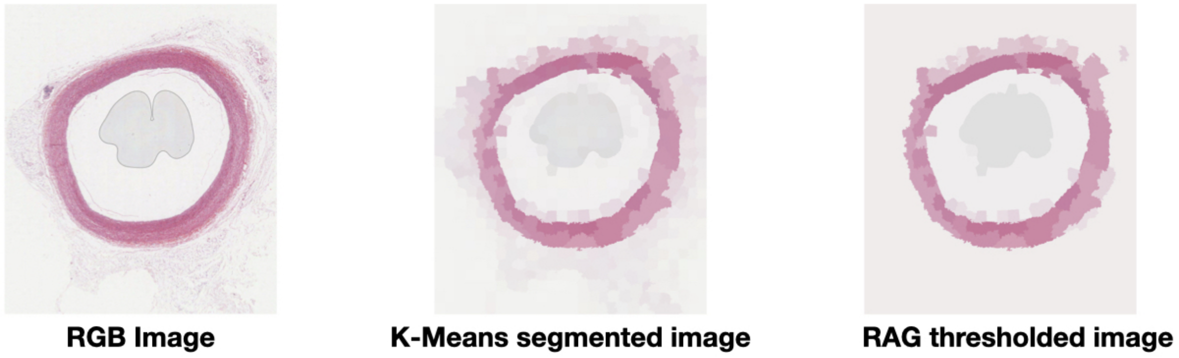
[1] A Ruifrok and et al. "Quantification of histochemical staining by color deconvolution". *Anal Quant Cytol Histol* 23.4 (2001)

[2] K B Suliman and A Krzyzak. "Computerized Counting-Based System for Acute Lymphoblastic Leukemia Detection in Microscopic Blood Images". Artificial Neural Networks and Machine Learning (ICANN 2018). *Springer* (2018)

threshold-based filters, based on popular algorithms of interest for pathology. The threshold-based filters in `histolab` output a binary mask that results from replacing each pixel in the input image above (or below) the computed threshold with 1, 0 otherwise.



- **Color-based segmentation filters**: The *k*-means algorithm is one of the most popular unsupervised Machine Learning algorithms for clustering multidimensional data. The *k*-means approach can be also applied in image segmentation to separate pixel groups in terms of color, e.g. to detect variation in staining[3] or to segment specific structures[4] on histological images. `histolab` implements two color segmentation filters based on the *k*-means algorithm, namely KmeansSegmentation and RagThreshold. The first one segments the image into *n* segments (by default n=800) using *k*-means in the color space, and then colors each segment based on its average color. To overcome the over-segmentation that the *k*-means algorithm may generate, the `RagThreshold` filter allows similarly colored segments to be grouped together: (i) the image is segmented with *k*-means; (ii) the Region Adjacency Graph (RAG) is built based on the segments; (iii) nodes in the graph connected by an edge with weight less than a specific threshold *t* (by default t=9) are combined.

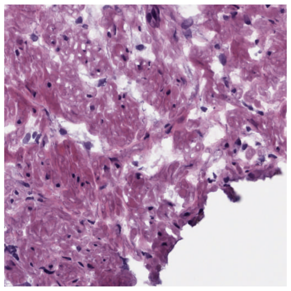


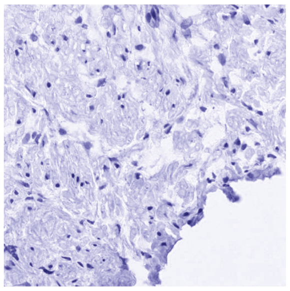
- **Channel extractor filters**: The preparation of histopathological slides is based on processing tissue samples with a sequence of histochemical staining steps. In order to reveal specific structural elements in the tissue, different staining protocols are applied, with a wide range of techniques, colouring

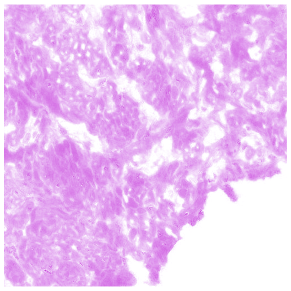
[3] R D Peng. “Reproducible research in computational science”. Science 334.6060 (2011)

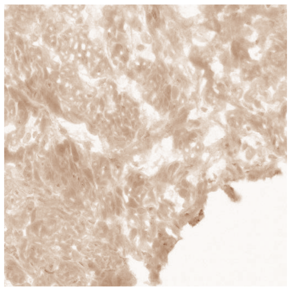
[4] J Sieren andet al. “An automated segmentation approach for highlighting the histologicalcomplexity of human lung cancer”. *Ann Biomed Eng* 38.12 (2010)

reagents (e.g. H&E or specific IHC), and their combinations. Developed for human readers, the protocol modifies the color information to detect, or ignore, specific regions on the WSI according to the task. For example, on H&E-stained images, filtering out pixels with high green channel value remove areas with no presence of nuclei and/or tissue (purple and pink colors, respectively). Also, extracting the hematoxylin channel helps in selecting the regions with cell nuclei, and therefore ease the detection of mitosis. `histolab` provides a set of filters designed to extract a single channel from 3-channels images (e.g. RGB, HSV, HED). In particular, the HematoxylinChannel and the EosinChannel methods extract the hematoxylin and eosin channel, respectively, after converting the image into the appropriate color space (HED), and enhancing the contrast.


Original image


Hematoxylin channel


Eosin channel

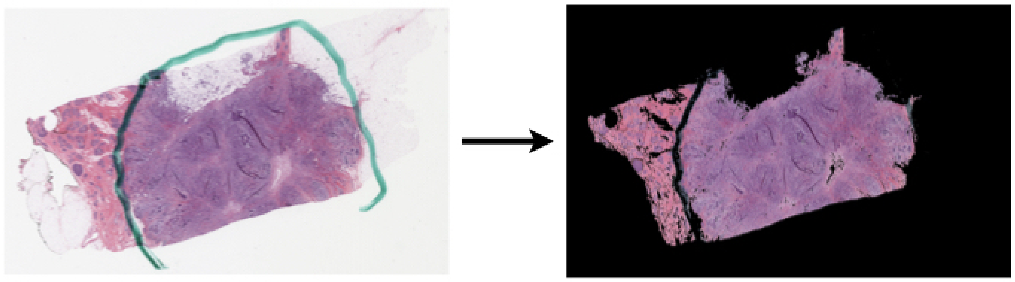

DAB channel

- **Diagnostic annotations filters**: In clinical practice, pathologists often annotate slides with pen marks to simplify the diagnostic process, for example by delineating cancerous areas or by segmenting regions of interest. These manual annotations, while useful for human analysis, are confounds for an automatic pipeline due to the lack in the standardization of annotation procedures (handwritten labels may be subjective and error-prone[5], and because they could alter the feature extraction process. Therefore, it is essential to either eliminate or correct these artifacts[6]. A Deep Learning pipeline has been introduced to erase ink marks from digital slides by Ali et al.[8]. Although the method is efficient on reconstructing regions hidden by the annotations, it requires large (manually annotated) datasets and a relevant computational cost. `histolab` includes methods to clean ink signs in a combination of image filters[7]. In particular, green, red and blue marks are deleted by progressively removing pixels within fixed ranges of intensity. While the green and the blue pen filters are extremely effective on annotated H&E slides, the red pen filter should be used carefully: due to similarity with the eosin staining, it could erode reddish regions, such as aggregation of erythrocytes (blood cells).

Note: `histolab` stores masks as NumPy arrays. The utility class ToPILImage in the image filters module retrieves the Pillow Image from the corresponding array.

- **Morphological filters:**
  - **Image preprocessing**: Morphology is a comprehensive set of image processing operations that transform images by using geometrical structures. Morphological operations act by adjusting image pixels based on the value of the pixels in the neighborhood. The choice of the neighborhood's size and shape

[5] E A Wagar and et al. "Specimen labeling errors: a Q-probes analysis of 147 clinical laboratories". *Arch Pathol Lab Med* 132.10 (2008)
[6] S K Phan and et al. "Biomedical Imaging Informatics for Diagnostic Imaging Marker Selection". *Health Informatics Data Analysis*. Springer (2017)
[8] M Dusenberry and et al. *deep-histopath <https://github.com/CODAIT/deep-histopath>*
[7] S Ali and et al. "Ink removal from histopathology whole slide images by combining classification, detection and image generation models". *IEEE 16th International Symposium on Biomedical Imaging* (ISBI 2019).



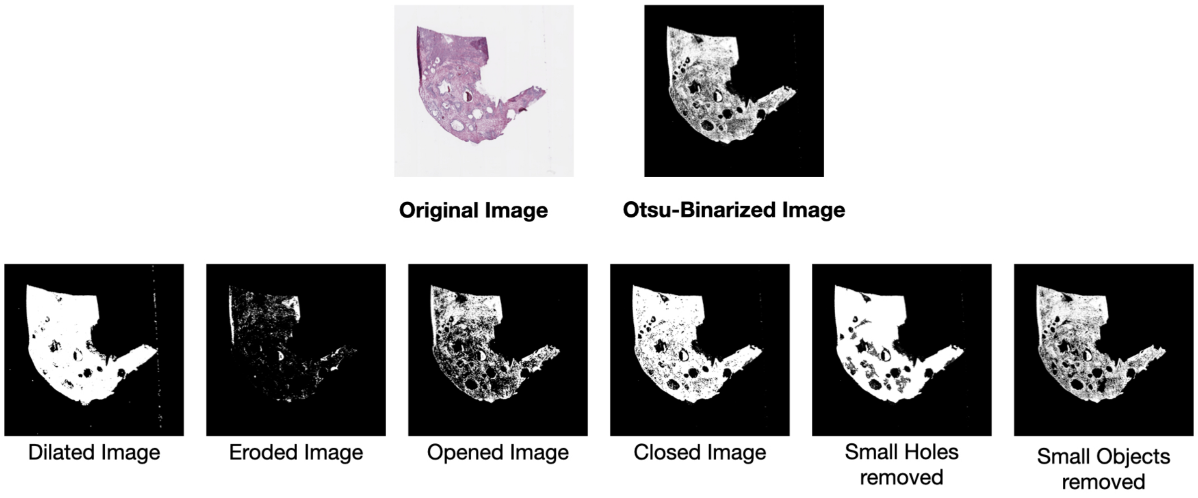
will affect the behaviour of the morphological operation so that it will be sensitive to specific shapes in the input image[9]. The *structuring element* is the component of the morphological operations that defines the considered neighborhood: it is a shape (typically a circle or a square) that determines the area used to process each pixel in the image. Usually, the shape and size of the structuring element are chosen to reflect the geometry of the objects in the image that the structuring element will transform: for example, linear structuring element would be used to detect lines in an image. Morphological operations can be applied to binary masks to shrink or enlarge regions of the image. Classic morphological operations include dilation, erosion, opening and closing; `histolab` implements these operations in the filters submodule morphological_filters. The default structuring element is a disk, with radius 5 for both dilation and erosion, and radius 3 for both opening and closing. However, it is possible to override the default value by passing `disk_size=N` as parameter to the filter constructor. The morphological_filters module implements three additional morphological operations useful for manipulating binary masks: WhiteTopHat, RemoveSmallObjects, and RemoveSmallHoles. The *white top-hat* transformation is defined as the difference between the image and its morphological opening with respect to a structuring element. This operation results in an image including only structures smaller than the structuring element and brighter than their neighborhood, and it is thus used to extract light details on a dark background. The white top-hat filter uses a cross-shaped structuring element with connectivity 1 by default. The RemoveSmallObjects filter removes objects with an area smaller than a specified value while the RemoveSmallHoles filter "fills" holes with an area smaller than the specified threshold. The minimal area value is set to 3000 for both filters.

– **Segmentation**: `histolab` implements the Watershed algorithm, a popular segmentation method for binary masks based on image morphology, useful to separate overlapping objects[10]. This algorithm works by treating the mask as a topographic map, with the value of each pixel representing the elevation, and by flooding basins from user-defined markers until basins attributed to different markers meet on watershed lines. The WatershedSegmentation filter first computes an image that represents for each pixel the Euclidean distance D of that pixel to the closest pixel on the background. Then, the points corresponding to the maxima of the distance D are chosen as markers for the algorithm.

Note: Notice that both the input and the output of morphological filters are binary masks.

[9] A P Vartak and V Mankar. "Morphological image segmentation analysis". *Int J Comput Appl* 6.2 (2013)

[10] L Vincent and P Soille. "Watersheds in digital spaces: an efficient algorithm based on immersion simulations". *IEEE Trans Pattern Anal Mach Intell* 6 (1991)


Original Image
Otsu-Binarized Image
Dilated Image
Eroded Image
Opened Image
Closed Image
Small Holes removed
Small Objects removed

To ease combining filters together, inspired by the `transforms` module of torchvision[11], `histolab` implements the Compose class in the `filters` subpackage. Compose allows `histolab` filters - both image and morphological filters - to be concatenated as a chain of functions, without any intermediate transformation or prior knowledge on their ordering. In order to enable this composition, filters have been designed to follow the same usage pattern, and to input/output either a PIL Image or a NumPy array object. To clarify how the Compose object works, let us consider the sequential application of a set of filters, where the output of a filter is the input of the following one:

```python
from PIL import Image
from histolab.filters.image_filters import (
    ApplyMaskImage,
    OtsuThreshold,
    RgbToGrayscale,
)
from histolab.filters.morphological_filters import BinaryDilation

def not_composed_filters(image_rgb):
    rgb_to_grayscale = RgbToGrayscale()
    otsu_threshold = OtsuThreshold()
    binary_dilation = BinaryDilation()
    apply_mask_image = ApplyMaskImage(
        image_rgb
    )  # apply the resulting mask on the original image
    image_gray = rgb_to_grayscale(image_rgb)
    image_thresholded = otsu_threshold(image_gray)
    image_dilated = binary_dilation(image_thresholded)
    return apply_mask_image(image_dilated)

image_rgb = Image.open("path/to/image.png")
resulting_image = not_composed_filters(image_rgb)
```

Despite being formally correct, the above implementation in neither intuitive or economic (in terms of memory used and lines of code). The use of a Compose object leads to a more compact implementation:

```python
from PIL import Image
from histolab.filters.image_filters import (
    ApplyMaskImage,
    Compose,
    OtsuThreshold,
    RgbToGrayscale,
)
from histolab.filters.morphological_filters import BinaryDilation

def composed_filters(image_rgb):
    filters = Compose(
        [
            RgbToGrayscale(),
            OtsuThreshold(),
            BinaryDilation(),
            ApplyMaskImage(image_rgb),
        ]
    )
```

(continues on next page)

[11] A Paszke and et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". *Advances in Neural Information Processing Systems* 32 (NeurIPS 2019)

(continued from previous page)

```
    return filters(image_rgb)

image_rgb = Image.open("path/to/image.png")
resulting_image = composed_filters(image_rgb)
```

Note: Although `not_composed_filters` and `composed_filters` functions return the same result, the use of Compose avoids storing intermediate results and wasting memory in case of very large input image. For example, the peak RAM usage for an image of size 19,394px x 6,136px (84.9 MB) is ~2600 MB when using `composed_filters` in contrast to the ~3200 MB allocated for the `not_composed_filters` function.

6.8.6 References

6.9 Masks

The classes implemented in the `masks` module define how to retrieve a binary mask of the tissue from a Slide object. This step is necessary during the tiles extraction phase. The `masks` module already implements different approaches to retrieve specific tissue regions within the slide: the TissueMask class segments the whole tissue area in the slide leveraging a sequence of native filters, including conversion to grayscale, Otsu thresholding, binary dilation, small holes and small objects removal; the BiggestTissueBoxMask class applies the same chain of filters as TissueMask but it returns a binary mask corresponding to the bounding box of the largest connected tissue region within the slide. Alternatively, a custom binary mask can be defined with the BinaryMask class.

class BinaryMask

Generic object for binary masks.

This object can be used to create a custom binary mask object.

Example

```
>>> from histolab.slide import Slide
>>> class MyCustomMask(BinaryMask):
...     def _mask(self, slide):
...         my_mask = np.array([0,1])
...         return my_mask
>>> binary_mask = MyCustomMask()
>>> slide = Slide("path/to/slide")
>>> binary_mask(slide)
```

__call__(*slide*)

Call self as a function.

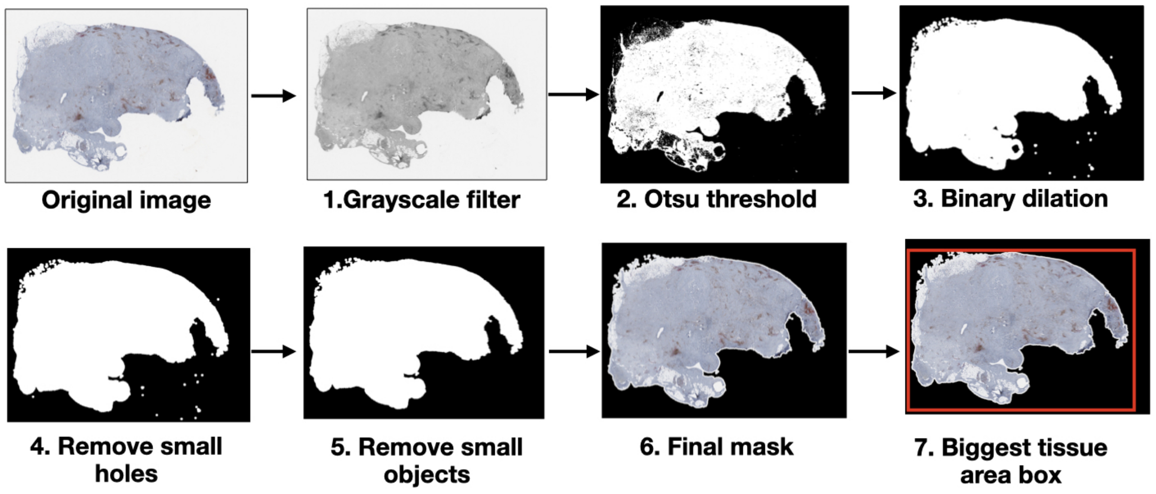
class BiggestTissueBoxMask

Object that represents the box containing the largest contiguous tissue area.

Internally, this class automatically detects the tissue regions via a predefined sequence of filters, and then retain the largest connected component.

__call__(*slide*)

Call self as a function.



class TissueMask

Object that represent the whole tissue area mask.

The tissue within the slide or tile is automatically detected through a predefined chain of filters.

`__call__`(*obj*)

Apply a predefined chain of filters to calculate the tissue area mask.

The applied filters will be different based on the type of `obj`, please see

filters.compositions.FiltersComposition

Parameters **`obj`** (*`Union[Slide, Tile]`*) – `Slide` or `Tile` from which to compute the extraction mask.

Returns Binary mask of the tissue area. The dimensions are those of the thumbnail in case `obj` is a `Slide`, otherwise they are the same as the tile.

Return type np.ndarray

See also:

filters.compositions.FiltersComposition

6.10 Tile

The `tile` module contains the Tile class to manage a rectangular region cropped from a Slide. A Tile object is described by (i) its extraction coordinates at native magnification (corresponding to level 0 in `OpenSlide`), (ii) the level of extraction, (iii) the actual image, stored as a PIL Image. A Tile object will be created internally during the tile extraction process.

class Tile(*image*, *coords*, *level=0*)

Provide Tile object representing a tile generated from a Slide object.

Parameters

- **`image`** (*`PIL.Image.Image`*) – Image describing the tile
- **`coords`** (*`CoordinatePair`*) – Level 0 Coordinates of the Slide from which the tile was extracted

- `level` (*int, optional*) – Level of tile extraction, by default 0

`apply_filters`(*filters*)

Apply a filter or composition of filters on a tile.

Parameters `filters` (*imf.Filter*) – Filter or composition of filters to be applied

Returns Tile with the filters applied

Return type *Tile*

`property coords: histolab.types.CoordinatePair`

Level 0 Coordinates of the Slide from which the tile was extracted

Returns Level 0 Coordinates of the Slide from which the tile was extracted

Return type CoordinatePair

`has_enough_tissue`(*tissue_percent=80.0*, *near_zero_var_threshold=0.1*)

Check if the tile has enough tissue.

This method checks if the proportion of the detected tissue over the total area of the tile is above a specified threshold (by default 80%). Internally, the method quantifies the amount of tissue by applying a chain of filters, including conversion to grayscale, Otsu thresholding, binary dilation and small holes filling.

Parameters

- `tissue_percent` (*float, optional*) – Number between 0.0 and 100.0 representing the minimum required percentage of tissue over the total area of the image, default is 80.0
- `near_zero_var_threshold` (*float, optional*) – Minimum image variance after morphological operations (dilation, fill holes), default is 0.1

Returns **enough_tissue** – Whether the image has enough tissue, i.e. if the proportion of tissue over the total area of the image is more than `tissue_percent` and the image variance after morphological operations is more than `near_zero_var_threshold`.

Return type bool

`property image: PIL.Image.Image`

Image describing the tile.

Returns Image describing the tile.

Return type PIL.Image.Image

`property level: int`

Level of tile extraction.

Returns Level of tile extraction.

Return type int

`save`(*path*)

Save tile at given path.

The format to use is determined from the filename extension (to be compatible to PIL.Image formats). If no extension is provided, the image will be saved in png format.

Parameters `path` (*str or pathlib.Path*) – Path to which the tile is saved.

Return type None

`property tissue_ratio: float`

Ratio of the tissue area over the total area of the tile.

Returns Ratio of the tissue area over the total area of the tile

Return type float

6.11 Tiler

Different logics are implemented for tile extraction in the `tiler` module. The constructor of the three extractors RandomTiler, GridTiler, and ScoreTiler share a similar interface and common parameters that define the extraction design:

1. `tile_size`: the tile size;
2. `level`: the extraction level, from 0 to the number of available levels; negative indexing is also possible, counting backward from the number of available levels to 0 (e.g. `level` =-1 means selecting the last available level);
3. `check_tissue`: True if a minimum percentage of tissue over the total area of the tile is required to save the tiles, False otherwise;
4. `tissue_percent`: number between 0.0 and 100.0 representing the minimum required ratio of tissue over the total area of the image, considered only if `check_tissue` equals to True (default is 80.0);
5. `prefix`: a prefix to be added at the beginning of the tiles' filename (optional, default is the empty string);
6. `suffix`: a suffix to be added to the end of the tiles' filename (optional, default is `.png`).

The general mechanism is to (i) create a tiler object, (ii) define a `Slide` object, used to identify the input image, and (iii) create a mask object to determine the area for tile extraction within the tissue. The extraction process starts when the tiler's `extract()` method is called, with the slide and the mask passed as parameters.

6.11.1 RandomTiler

The RandomTiler extractor allows for the extraction of tiles picked at random within the regions defined by the binary mask object. Since there is no intrinsic upper bound of the number of the tiles that could be extracted (no overlap check is performed), the number of wanted tiles must be specified.

In addition to 1-6, the RandomTiler constructor requires as two additional parameters the number of tiles requested (`n_tiles`), and the random seed (`seed`), to ensure reproducibility between different runs on the same WSI. Note that less than `n_tiles` could be extracted from a slide with not enough tissue pixels and a lot of background, which is checked when the parameter `check_tissue` is set to True. `n_tiles` will be interpreted as the upper bound of the number of tiles requested: it might not be possible to extract `n_tiles` tiles from a slide with a little tissue sample and a lot of background.

The extraction procedure will (i) find the regions to extract tiles from, defined by the binary mask object; (ii) generate `n_tiles` random tiles; (iii) save only the tiles with enough tissue if the attribute `check_tissue` was set to True, save all the generated tiles otherwise.

6.11.2 GridTiler

A second basic approach consists of extracting all the tiles in the areas defined by the binary mask. This strategy is implemented in the GridTiler class. The additional `pixel_overlap` parameter specifies the number of overlapping pixels between two adjacent tiles, i.e. tiles are cropped by using a sliding window with stride s defined as:

$$s = (w - \text{pixel\_overlap}) \cdot (h - \text{pixel\_overlap})$$

where w and h are customizable parameters defining the width and the height of the resulting tiles. Calling the `extract` method on the GridTiler instance will automatically (i) find the regions to extract tiles from, defined by the binary mask object; (ii) generate all the tiles according to the grid structure; (iii) save only the tiles with "enough tissue" if the attribute `check_tissue` was set to True, save all the generated tiles otherwise.

6.11.3 ScoreTiler

Tiles extracted from the same WSI may not be equally informative; for example, if the goal is the detection of mitotic activity on H&E slides, tiles with no nuclei are of little interest. The ScoreTiler extractor ranks the tiles with respect to a scoring function, described in the `scorer` module. In particular, the ScoreTiler class extends the `GridTiler` extractor by sorting the extracted tiles in a decreasing order, based on the computed score. Notably, the ScoreTiler is agnostic to the scoring function adopted, thus a custom function can be implemented provided that it inputs a `Tile` object and outputs a number. The additional parameter `n_tiles` controls the number of highest-ranked tiles to save; if `n_tiles` =0 all the tiles are kept. Similarly to the GridTiler extraction process, calling the `extract` method on the ScoreTiler instance will automatically (i) find the largest tissue area in the WSI; (ii) generate all the tiles according to the grid structure; (iii) retain all the tiles with enough tissue if the attribute `check_tissue` was set to True, all the generated tiles otherwise; (iv) sort the tiles in a decreasing order according to the scoring function defined in the `scorer` parameter; (v) save only the highest-ranked `n_tiles` tiles, if `n_tiles`>0; (vi) write a summary of the saved tiles and their scores in a CSV file, if the `report_path` is specified in the `extract` method. The summary reports for each tile t: (i) the tile filename; (ii) its raw score s_t; (iii) the normalized score, scaled in the interval [0,1], computed as:

$$\hat{s}_t = \frac{s_t - \min_{s\in S}(s)}{\max_{s\in S}(s) - \min_{s\in S}(s)} \ ,$$

where S is the set of the raw scores of all the extracted tiles.

class GridTiler(**args*, ***kwds*)

Extractor of tiles arranged in a grid, at the given level, with the given size.

Parameters

- **tile_size** (*Tuple[int, int]*) – (width, height) of the extracted tiles.
- **level** (*int, optional*) – Level from which extract the tiles. Default is 0. Superceded by mpp if the mpp argument is provided.
- **check_tissue** (*bool, optional*) – Whether to check if the tile has enough tissue to be saved. Default is True.
- **tissue_percent** (*float, optional*) – Number between 0.0 and 100.0 representing the minimum required percentage of tissue over the total area of the image, default is 80.0. This is considered only if `check_tissue` equals to True.
- **pixel_overlap** (*int, optional*) – Number of overlapping pixels (for both height and width) between two adjacent tiles. If negative, two adjacent tiles will be strided by the absolute value of `pixel_overlap`. Default is 0.
- **prefix** (*str, optional*) – Prefix to be added to the tile filename. Default is an empty string.
- **suffix** (*str, optional*) – Suffix to be added to the tile filename. Default is '.png'
- **mpp** (*float, optional*) – Micron per pixel resolution of extracted tiles. Takes precedence over level. Default is None.

extract(*slide*, *extraction_mask=<histolab.masks.BiggestTissueBoxMask object>*, *log_level='INFO'*)

Extract tiles arranged in a grid and save them to disk, following this filename pattern: *{prefix}tile_{tiles_counter}_level{level}_{x_ul_wsi}-{y_ul_wsi}-{x_br_wsi}-{y_br_wsi}{suffix}*

Parameters

- **slide** (`Slide`) – Slide from which to extract the tiles
- **extraction_mask** (`BinaryMask`, *optional*) – BinaryMask object defining how to compute a binary mask from a Slide. Default *BiggestTissueBoxMask*.

- **log_level** (*str, {"DEBUG", "INFO", "WARNING", "ERROR", "CRITICAL"}*) – Threshold level for the log messages. Default "INFO"

Raises

- **TileSizeError** – If the tile size is larger than the slide size
- **LevelError** – If the level is not available for the slide

Return type None

locate_tiles(*slide*, *extraction_mask=<histolab.masks.BiggestTissueBoxMask object>*, *scale_factor=32*, *alpha=128*, *outline='red'*, *linewidth=1*, *tiles=None*)

Draw tile box references on a rescaled version of the slide

Parameters

- **slide** (Slide) – Slide reference where placing the tiles
- **extraction_mask** (BinaryMask*, optional*) – BinaryMask object defining how to compute a binary mask from a Slide. Default *BiggestTissueBoxMask*
- **scale_factor** (*int, optional*) – Scaling factor for the returned image. Default is 32.
- **alpha** (*int, optional*) – The alpha level to be applied to the rescaled slide. Default is 128.
- **outline** (*Union[str, Iterable[str], Iterable[Tuple[int]], optional*) – The outline color for the tile annotations. Default is 'red'. You can provide this as a string compatible with matplotlib, or you can provide a list of the same length as the tiles, where each color is your assigned color for the corresponding individual tile. This list can be a list of matplotlib-style string colors, or a list of tuples of ints in the [0, 255] range, each of length 3, representing the red, green and blue color for each tile. For example, if you have two tiles that you want to be colored yellow, you can pass this argument as any of the following .. - 'yellow' - ['yellow', 'yellow'] - [(255, 255, 0), (255, 255, 0)]
- **linewidth** (*int, optional*) – Thickness of line used to draw tiles. Default is 1.
- **tiles** (*Optional[Iterable[*Tile*]], optional*) – Tiles to visualize. Will be extracted if None. Default is None. You may decide to provide this argument if you do not want the tiles to be re-extracted for visualization if you already have the tiles in hand.

Returns PIL Image of the rescaled slide with the extracted tiles outlined

Return type PIL.Image.Image

property tile_size: Tuple[int, int]

(width, height) of the extracted tiles.

class RandomTiler(**args*, ***kwds*)

Extractor of random tiles from a Slide, at the given level, with the given size.

Parameters

- **tile_size** (*Tuple[int, int]*) – (width, height) of the extracted tiles.
- **n_tiles** (*int*) – Maximum number of tiles to extract.
- **level** (*int, optional*) – Level from which extract the tiles. Default is 0. Superceded by mpp if the mpp argument is provided.
- **seed** (*int, optional*) – Seed for RandomState. Must be convertible to 32 bit unsigned integers. Default is 7.

- `check_tissue` (*bool, optional*) – Whether to check if the tile has enough tissue to be saved. Default is True.
- `tissue_percent` (*float, optional*) – Number between 0.0 and 100.0 representing the minimum required percentage of tissue over the total area of the image, default is 80.0. This is considered only if `check_tissue` equals to True.
- `prefix` (*str, optional*) – Prefix to be added to the tile filename. Default is an empty string.
- `suffix` (*str, optional*) – Suffix to be added to the tile filename. Default is '.png'
- `max_iter` (*int, optional*) – Maximum number of iterations performed when searching for eligible (if `check_tissue=True`) tiles. Must be grater than or equal to `n_tiles`.
- `mpp` (*float, optional*) – Micron per pixel resolution. If provided, takes precedence over level. Default is None.

`extract`(*slide*, *extraction_mask=<histolab.masks.BiggestTissueBoxMask object>*, *log_level='INFO'*)

Extract random tiles and save them to disk, following this filename pattern: *{prefix}tile_{tiles_counter}_level{level}_{x_ul_wsi}-{y_ul_wsi}-{x_br_wsi}-{y_br_wsi}{suffix}*

Parameters

- `slide` (`Slide`) – Slide from which to extract the tiles
- `extraction_mask` (`BinaryMask`, *optional*) – BinaryMask object defining how to compute a binary mask from a Slide. Default *BiggestTissueBoxMask*.
- `log_level` (*str, {"DEBUG", "INFO", "WARNING", "ERROR", "CRITICAL"}*) – Threshold level for the log messages. Default "INFO"

Raises

- `TileSizeError` – If the tile size is larger than the slide size
- `LevelError` – If the level is not available for the slide

Return type None

`locate_tiles`(*slide*, *extraction_mask=<histolab.masks.BiggestTissueBoxMask object>*, *scale_factor=32*, *alpha=128*, *outline='red'*, *linewidth=1*, *tiles=None*)

Draw tile box references on a rescaled version of the slide

Parameters

- `slide` (`Slide`) – Slide reference where placing the tiles
- `extraction_mask` (`BinaryMask`, *optional*) – BinaryMask object defining how to compute a binary mask from a Slide. Default *BiggestTissueBoxMask*
- `scale_factor` (*int, optional*) – Scaling factor for the returned image. Default is 32.
- `alpha` (*int, optional*) – The alpha level to be applied to the rescaled slide. Default is 128.
- `outline` (*Union[str, Iterable[str], Iterable[Tuple[int]]], optional*) – The outline color for the tile annotations. Default is 'red'. You can provide this as a string compatible with matplotlib, or you can provide a list of the same length as the tiles, where each color is your assigned color for the corresponding individual tile. This list can be a list of matplotlib-style string colors, or a list of tuples of ints in the [0, 255] range, each of length 3, representing the red, green and blue color for each tile. For example, if you have two tiles that you want to be colored yellow, you can pass this argument as any of the following .. - 'yellow' - ['yellow', 'yellow'] - [(255, 255, 0), (255, 255, 0)]

- **linewidth** (*int, optional*) – Thickness of line used to draw tiles. Default is 1.
- **tiles** (*Optional[Iterable[Tile]], optional*) – Tiles to visualize. Will be extracted if None. Default is None. You may decide to provide this argument if you do not want the tiles to be re-extracted for visualization if you already have the tiles in hand.

Returns PIL Image of the rescaled slide with the extracted tiles outlined

Return type PIL.Image.Image

`class ScoreTiler`(**args*, ***kwds*)

Extractor of tiles arranged in a grid according to a scoring function.

The extraction procedure is the same as the `GridTiler` extractor, but only the first `n_tiles` tiles with the highest score are saved.

Parameters

- **scorer** (Scorer) – Scoring function used to score the tiles.
- **tile_size** (*Tuple[int, int]*) – (width, height) of the extracted tiles.
- **n_tiles** (*int, optional*) – The number of tiles to be saved. Default is 0, which means that all the tiles will be saved (same exact behaviour of a GridTiler). Cannot be negative.
- **level** (*int, optional*) – Level from which extract the tiles. Default is 0. Superceded by mpp if the mpp argument is provided.
- **check_tissue** (*bool, optional*) – Whether to check if the tile has enough tissue to be saved. Default is True.
- **tissue_percent** (*float, optional*) – Number between 0.0 and 100.0 representing the minimum required percentage of tissue over the total area of the image, default is 80.0. This is considered only if `check_tissue` equals to True.
- **pixel_overlap** (*int, optional*) – Number of overlapping pixels (for both height and width) between two adjacent tiles. If negative, two adjacent tiles will be strided by the absolute value of `pixel_overlap`. Default is 0.
- **prefix** (*str, optional*) – Prefix to be added to the tile filename. Default is an empty string.
- **suffix** (*str, optional*) – Suffix to be added to the tile filename. Default is '.png'
- **mpp** (*float, optional.*) – Micron per pixel resolution. If provided, takes precedence over level. Default is None.

`extract`(*slide*, *extraction_mask=<histolab.masks.BiggestTissueBoxMask object>*, *report_path=None*, *log_level='INFO'*)

Extract grid tiles and save them to disk, according to a scoring function and following this filename pattern: *{prefix}tile_{tiles_counter}_level{level}_{x_ul_wsi}-{y_ul_wsi}-{x_br_wsi}-{y_br_wsi}{suffix}*

Save a CSV report file with the saved tiles and the associated score.

Parameters

- **slide** (Slide) – Slide from which to extract the tiles
- **extraction_mask** (BinaryMask, *optional*) – BinaryMask object defining how to compute a binary mask from a Slide. Default *BiggestTissueBoxMask*.
- **report_path** (*str, optional*) – Path to the CSV report. If None, no report will be saved

- **log_level** (*str, {"DEBUG", "INFO", "WARNING", "ERROR", "CRITICAL"}*) – Threshold level for the log messages. Default "INFO"

Raises

- **TileSizeError** – If the tile size is larger than the slide size
- **LevelError** – If the level is not available for the slide

Return type None

locate_tiles(*slide*, *extraction_mask=<histolab.masks.BiggestTissueBoxMask object>*, *scale_factor=32*, *alpha=128*, *outline='red'*, *linewidth=1*, *tiles=None*)

Draw tile box references on a rescaled version of the slide

Parameters

- **slide** (Slide) – Slide reference where placing the tiles
- **extraction_mask** (BinaryMask, *optional*) – BinaryMask object defining how to compute a binary mask from a Slide. Default *BiggestTissueBoxMask*
- **scale_factor** (*int, optional*) – Scaling factor for the returned image. Default is 32.
- **alpha** (*int, optional*) – The alpha level to be applied to the rescaled slide. Default is 128.
- **outline** (*Union[str, Iterable[str], Iterable[Tuple[int]], optional*) – The outline color for the tile annotations. Default is 'red'. You can provide this as a string compatible with matplotlib, or you can provide a list of the same length as the tiles, where each color is your assigned color for the corresponding individual tile. This list can be a list of matplotlib-style string colors, or a list of tuples of ints in the [0, 255] range, each of length 3, representing the red, green and blue color for each tile. For example, if you have two tiles that you want to be colored yellow, you can pass this argument as any of the following .. - 'yellow' - ['yellow', 'yellow'] - [(255, 255, 0), (255, 255, 0)]
- **linewidth** (*int, optional*) – Thickness of line used to draw tiles. Default is 1.
- **tiles** (*Optional[Iterable[Tile]], optional*) – Tiles to visualize. Will be extracted if None. Default is None. You may decide to provide this argument if you do not want the tiles to be re-extracted for visualization if you already have the tiles in hand.

Returns PIL Image of the rescaled slide with the extracted tiles outlined

Return type PIL.Image.Image

property tile_size: Tuple[int, int]

(width, height) of the extracted tiles.

6.12 Scorer

The goal of the `scorer` module is to provide the grading functions for the ScoreTiler extractor. The `scorer` objects input a Tile object and return their computed score.

class CellularityScorer(**args*, ***kwds*)

Implement a Scorer that estimates the cellularity in an H&E-stained tile.

This class deconvolves the hematoxylin channel and uses the fraction of tile occupied by hematoxylin as the cellularity score.

Notice that this scorer is useful when tiles are extracted at a very low resolution with no artifacts; in this case, using the``NucleiScorer()`` instead would not work well as nuclei are no discernible at low magnification.

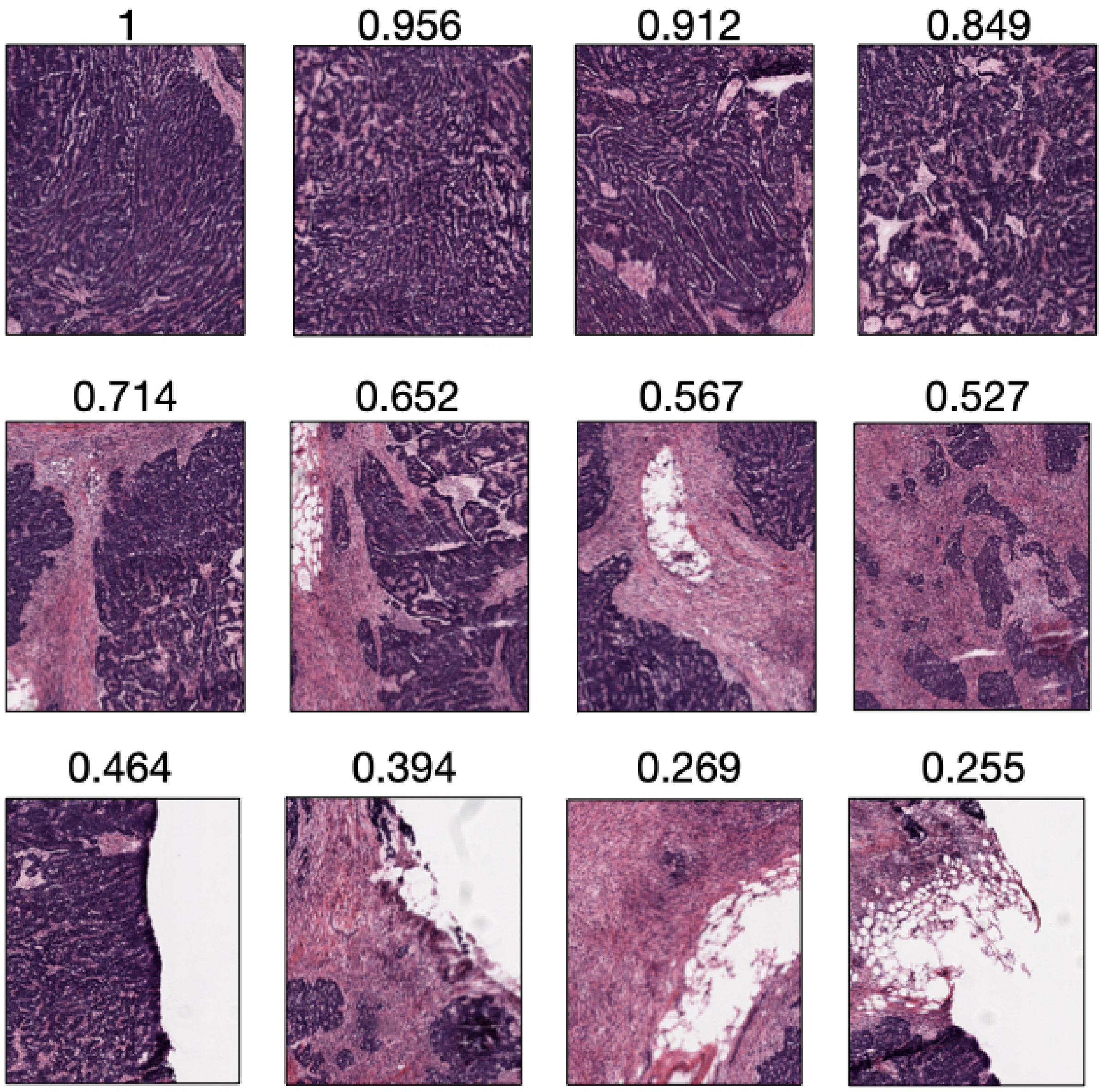


Fig. 1: Example of tiles scored by the NucleiScorer object with their respective score scaled between 0 and 1. Tiles are extracted from the ovarian tissue WSI.

`__call__`(*tile*)

Return the tile cellularity score.

Parameters

- **`tile`** (`Tile`) – The tile to calculate the score from.
- **`consider_tissue`** (*`bool`*) – Whether the cellularity score should be computed by considering the tissue on the tile. Default is True

Returns Cellularity score

Return type float

Parameters **`consider_tissue`** (*`bool, optional`*) – Whether the detected tissue on the tile should be considered to compute the cellularity score. Default is True

Notes

If the tile presents artifacts (e.g., tissue folds, markers), the scorer cannot be fully trusted.

`class` **`NucleiScorer`**(**args*, ***kwds*)

Implement a Scorer that estimates the presence of nuclei in an H&E-stained tile.

This class implements an hybrid algorithm that combines thresholding and morphological operations to segment nuclei on H&E-stained histological images.

The NucleiScorer class defines the score of a given tile t as:

$$s_t = N_t \cdot \tanh(T_t), \, 0 \leq s_t < 1$$

where N_t is the nuclei ratio on t, computed as number of white pixels on the segmented mask over the tile size, and T_t the fraction of tissue in t.

Notice that we introduced the hyperbolic tangent to bound the weight of the tissue ratio over the nuclei ratio.

Notes

If the tile presents artifacts (e.g., tissue folds, markers), the scorer cannot be fully trusted.

`__call__`(*tile*)

Return the nuclei score associated with the tile.

Parameters **`tile`** (`Tile`) – The tile to calculate the score from.

Returns Nuclei score

Return type float

`class` **`RandomScorer`**(**args*, ***kwds*)

Implement a Scorer that returns a random float score between 0 and 1.

`__call__`(*tile*)

Return the random score associated with the tile.

Parameters **`tile`** (`Tile`) – The tile to calculate the score from.

Returns Random score ranging between 0 and 1.

Return type float

class Scorer(**args*, ***kwds*)
General scorer object

abstract __call__(*tile*)
Call self as a function.

Parameters tile (histolab.tile.Tile) –

Return type float

6.13 Data

The `data` module gives access to a set of publicly available WSIs, stained with different techniques (H&E and IHC). In particular, slides in the `data` module are retrieved from the following repositories:

- The Cancer Genome Atlas (TCGA): as detailed in the methods docstring, for each WSI, we access the URL pointing to the corresponding location within the portal, e.g. https://portal.gdc.cancer.gov/files/9c960533-2e58-4e54-97b2-8454dfb4b8c8, to retrieve the WSI;
- OpenSlide, a repository of freely-distributed test slides from different scanner vendors;
- Image Data Resource (IDR): the WSIs are selected from the data collection provided by Schaadt et al.[1] and available at IDR under the accession number *idr0073*.

Note: We use Pooch under the hood, which is an optional requirement for `histolab` and needs to be installed separately with:

```
pip install pooch
```

Table 1: Set of downloadable WSIs.

Tissue	Dimensions (wxh)	Size (MB)	Repository	Staining
Aorta	15374x17497	63.8	OpenSlide	H&E
CMU small sample	2220x2967	1.8	OpenSlide	H&E
Breast	96972x30682	299.1	TCGA-BRCA	H&E
Breast (black pen)	121856x94697	1740.8	TCGA-BRCA	H&E
Breast (green pen)	98874x64427	719.6	TCGA-BRCA	H&E
Breast (red pen)	60928x75840	510.9	TCGA-BRCA	H&E
Breast (IHC)	99606x7121	218.3	IDR	IHC
Heart	32672x47076	289.3	OpenSlide	H&E
Kidney	5179x4192	66.1	IDR	IHC
Ovary	30001x33987	389.1	TCGA-OV	H&E
Prostate	16000x15316	46.1	TCGA-PRAD	H&E

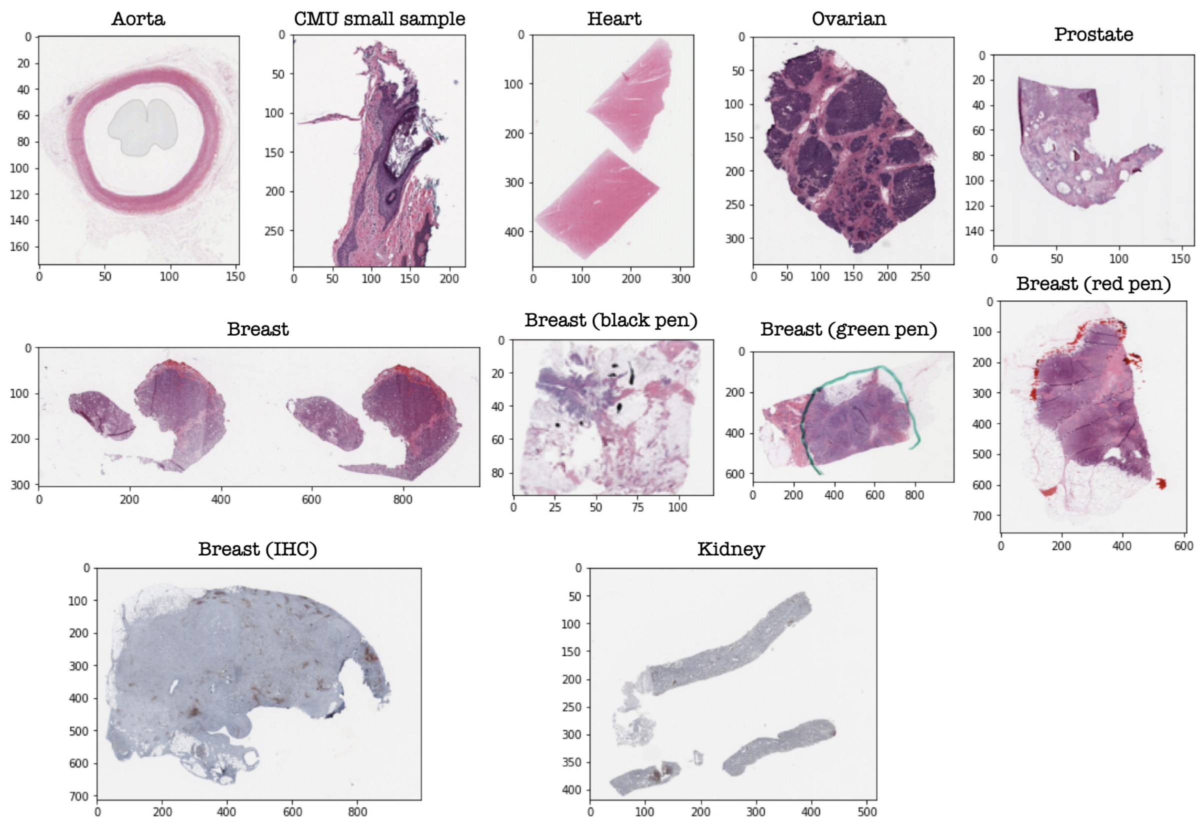
TCGA-BRCA: TCGA Breast Invasive Carcinoma dataset; TCGA-PRAD: TCGA Prostate Adenocarcinoma dataset; TCGA-OV: Ovarian Serous Cystadenocarcinoma dataset.

aorta_tissue() → Tuple[openslide.OpenSlide, str]
Aorta tissue, brightfield, JPEG 2000, YCbCr

This image is available here http://openslide.cs.cmu.edu/download/openslide-testdata/Aperio/

Free to use and distribute, with or without modification

[1] Schaadt NS, Schönmeyer R, Forestier G, et al. "Graph-based description of tertiary lymphoid organs at single-cell level." PLoS Comput Biol. (2020)


Aorta
CMU small sample
Heart
Ovarian
Prostate
Breast
Breast (black pen)
Breast (green pen)
Breast (red pen)
Breast (IHC)
Kidney

Returns

- **aorta_tissue** (*openslide.OpenSlide*) – H&E-stained Whole-Slide-Image of aortic tissue.
- **path** (*str*) – Path where the slide is saved

Return type Tuple[openslide.OpenSlide, str]

`breast_tissue()` → Tuple[openslide.OpenSlide, str]

Breast tissue, TCGA-BRCA dataset.

This image is available here https://portal.gdc.cancer.gov/files/ad9ed74a-2725-49e6-bf7a-ef100e299989 or through the API https://api.gdc.cancer.gov/data/ad9ed74a-2725-49e6-bf7a-ef100e299989

It corresponds to TCGA file *TCGA-A8-A082-01A-01-TS1.3cad4a77-47a6-4658-becf-d8cffa161d3a.svs*

Access: open

Returns

- **breast_tissue** (*openslide.OpenSlide*) – H&E-stained Whole-Slide-Image of breast tissue.
- **path** (*str*) – Path where the slide is saved

Return type Tuple[openslide.OpenSlide, str]

`breast_tissue_diagnostic_black_pen()` → Tuple[openslide.OpenSlide, str]

Breast tissue, TCGA-BRCA dataset. Diagnostic slide with black pen marks.

This image is available here https://portal.gdc.cancer.gov/files/e70c89a5-1c2f-43f8-b6be-589beea55338 or through the API https://api.gdc.cancer.gov/data/e70c89a5-1c2f-43f8-b6be-589beea55338

It corresponds to TCGA file *TCGA-BH-A201-01Z-00-DX1.6D6E3224-50A0-45A2-B231-EEF27CA7EFD2.svs*

Access: open

Returns

- **breast_tissue** (*openslide.OpenSlide*) – H&E-stained Whole-Slide-Image of breast tissue with green black marks.
- **path** (*str*) – Path where the slide is saved

Return type Tuple[openslide.OpenSlide, str]

`breast_tissue_diagnostic_green_pen()` → Tuple[openslide.OpenSlide, str]

Breast tissue, TCGA-BRCA dataset. Diagnostic slide with green pen marks.

This image is available here https://portal.gdc.cancer.gov/files/3845b8bd-cbe0-49cf-a418-a8120f6c23db or through the API https://api.gdc.cancer.gov/data/3845b8bd-cbe0-49cf-a418-a8120f6c23db

It corresponds to TCGA file *TCGA-A1-A0SH-01Z-00-DX1.90E71B08-E1D9-4FC2-85AC-062E56DDF17C.svs*

Access: open

Returns

- **breast_tissue** (*openslide.OpenSlide*) – H&E-stained Whole-Slide-Image of breast tissue with green pen marks.
- **path** (*str*) – Path where the slide is saved

Return type Tuple[openslide.OpenSlide, str]

`breast_tissue_diagnostic_red_pen()` → Tuple[openslide.OpenSlide, str]

Breast tissue, TCGA-BRCA dataset. Diagnostic slide with red pen marks.

This image is available here https://portal.gdc.cancer.gov/files/682e4d74-2200-4f34-9e96-8dee968b1568 or through the API https://api.gdc.cancer.gov/data/682e4d74-2200-4f34-9e96-8dee968b1568

It corresponds to TCGA file *TCGA-E9-A24A-01Z-00-DX1.F0342837-5750-4172-B60D-5F902E2A02FD.svs*

Access: open

Returns

- **breast_tissue** (*openslide.OpenSlide*) – H&E-stained Whole-Slide-Image of breast tissue with red pen marks.
- **path** (*str*) – Path where the slide is saved

Return type Tuple[openslide.OpenSlide, str]

`cmu_small_region`() → Tuple[openslide.OpenSlide, str]

Carnegie Mellon University MRXS sample tissue

This image is available here http://openslide.cs.cmu.edu/download/openslide-testdata/Aperio/

Licensed under a CC0 1.0 Universal (CC0 1.0) Public Domain Dedication.

Returns

- **cmu_mrxs_tissue** (*openslide.OpenSlide*) – H&E-stained Whole-Slide-Image of small tissue region.
- **path** (*str*) – Path where the slide is saved

Return type Tuple[openslide.OpenSlide, str]

`file_hash`(*fname*, *alg='sha256'*)

Calculate the hash of a given file.

Useful for checking if a file has changed or been corrupted.

Parameters

- **`fname`** (`str`) – The name of the file.
- **`alg`** (`str`) – The type of the hashing algorithm

Returns **hash** – The hash of the file.

Return type str

Examples

```
>>> fname = "test-file-for-hash.txt"
>>> with open(fname, "w") as f:
...     __ = f.write("content of the file")
>>> print(file_hash(fname))
0fc74468e6a9a829f103d069aeb2bb4f8646bad58bf146bb0e3379b759ec4a00
>>> import os
>>> os.remove(fname)
```

`heart_tissue`() → Tuple[openslide.OpenSlide, str]

Heart tissue, brightfield, JPEG 2000, YCbCr

This image is available here http://openslide.cs.cmu.edu/download/openslide-testdata/Aperio/

Free to use and distribute, with or without modification

Returns

- **heart_tissue** (*openslide.OpenSlide*) – H&E-stained Whole-Slide-Image of heart tissue.
- **path** (*str*) – Path where the slide is saved

Return type Tuple[openslide.OpenSlide, str]

`ihc_breast()` → Tuple[openslide.OpenSlide, str]

Breast cancer resection, staining CD3 (brown) and CD20 (red).

This image is available here https://idr.openmicroscopy.org/ under accession number idr0073, ID *breast-Cancer12*.

Returns

- **ihc_breast** (*openslide.OpenSlide*) – IHC-stained Whole-Slide-Image of Breast tissue.
- **path** (*str*) – Path where the slide is saved

Return type Tuple[openslide.OpenSlide, str]

`ihc_kidney()` → Tuple[openslide.OpenSlide, str]

Renal allograft, staining CD3 (brown) and CD20 (red).

This image is available here https://idr.openmicroscopy.org/ under accession number idr0073, ID *kidney_46_4*.

Returns

- **ihc_kidney** (*openslide.OpenSlide*) – IHC-stained Whole-Slide-Image of kidney tissue.
- **path** (*str*) – Path where the slide is saved

Return type Tuple[openslide.OpenSlide, str]

`ovarian_tissue()` → Tuple[openslide.OpenSlide, str]

tissue of Ovarian Serous Cystadenocarcinoma, TCGA-OV dataset.

This image is available here https://portal.gdc.cancer.gov/files/e968375e-ef58-4607-b457-e6818b2e8431 or through the API https://api.gdc.cancer.gov/data/e968375e-ef58-4607-b457-e6818b2e8431

It corresponds to TCGA file *CGA-13-1404-01A-01-TS1.cecf7044-1d29-4d14-b137-821f8d48881e.svs*

Access: open

Returns

- **prostate_tissue** (*openslide.OpenSlide*) – H&E-stained Whole-Slide-Image of ovarian tissue.
- **path** (*str*) – Path where the slide is saved

Return type Tuple[openslide.OpenSlide, str]

`prostate_tissue()` → Tuple[openslide.OpenSlide, str]

tissue of Prostate Adenocarcinoma, TCGA-PRAD dataset.

This image is available here https://portal.gdc.cancer.gov/files/5a8ce04a-0178-49e2-904c-30e21fb4e41e or through the API https://api.gdc.cancer.gov/data/5a8ce04a-0178-49e2-904c-30e21fb4e41e

It corresponds to TCGA file *TCGA-CH-5753-01A-01-BS1.4311c533-f9c1-4c6f-8b10-922daa3c2e3e.svs*

Access: open

Returns

- **prostate_tissue** (*openslide.OpenSlide*) – H&E-stained Whole-Slide-Image of prostate tissue.
- **path** (*str*) – Path where the slide is saved

Return type Tuple[openslide.OpenSlide, str]

6.13.1 References

6.14 Util

`apply_mask_image`(*img*, *mask*)

Mask image with the provided binary mask.

Parameters

- **img** (*PIL.Image.Image*) – Input image
- **mask** (*np.ndarray*) – Binary mask

Returns Image with the mask applied

Return type PIL.Image.Image

`lazyproperty`(*f*)

Decorator like @property, but evaluated only on first access.

Like @property, this can only be used to decorate methods having only a *self* parameter, and is accessed like an attribute on an instance, i.e. trailing parentheses are not used. Unlike @property, the decorated method is only evaluated on first access; the resulting value is cached and that same value returned on second and later access without re-evaluation of the method.

Like @property, this class produces a *data descriptor* object, which is stored in the __dict__ of the *class* under the name of the decorated method ('fget' nominally). The cached value is stored in the __dict__ of the *instance* under that same name.

Because it is a data descriptor (as opposed to a *non-data descriptor*), its *__get__()* method is executed on each access of the decorated attribute; the __dict__ item of the same name is "shadowed" by the descriptor.

While this may represent a performance improvement over a property, its greater benefit may be its other characteristics. One common use is to construct collaborator objects, removing that "real work" from the constructor, while still only executing once. It also de-couples client code from any sequencing considerations; if it's accessed from more than one location, it's assured it will be ready whenever needed.

A lazyproperty is read-only. There is no counterpart to the optional "setter" (or deleter) behavior of an @property. This is critically important to maintaining its immutability and idempotence guarantees. Attempting to assign to a lazyproperty raises AttributeError unconditionally. The parameter names in the methods below correspond to this usage example:

```python
class Obj(object):

    @lazyproperty
    def fget(self):
        return 'some result'

obj = Obj()
```

Not suitable for wrapping a function (as opposed to a method) because it is not callable.

Parameters **f** (*Callable[[...], Any]*) –

`method_dispatch`(*func*)

Decorator like @singledispatch to dispatch on the second argument of a method.

It relies on @singledispatch to return a wrapper function that selects which registered function to call based on the type of the second argument.

This is implementation is required in order to be compatible with Python versions older than 3.8. In the future we could use `functools.singledispatchmethod`.

Source: https://stackoverflow.com/a/24602374/7162549

Parameters **func** (*Callable[..., Any]*) – Method to dispatch

Returns Selected method

Return type Callable[…, Any]

np_to_pil(*np_img*)

Convert a NumPy array to a PIL Image.

Parameters **np_img** (*np.ndarray*) – The image represented as a NumPy array.

Returns The image represented as PIL Image

Return type PIL.Image.Image

random_choice_true_mask2d(*binary_mask*)

Return a random pair of indices (column, row) where the `binary_mask` is True.

Parameters **binary_mask** (*np.ndarray*) – Binary array.

Returns Random pair of indices (column, row) where the `binary_mask` is True.

Return type Tuple[int, int]

rectangle_to_mask(*dims*, *vertices*)

Return a binary mask with True inside of rectangle `vertices` and False outside.

The returned mask has shape `dims`.

Parameters

- **dims** (*Tuple[int, int]*) – (rows, columns) of the binary mask
- **vertices** (*CoordinatePair*) – CoordinatePair representing the upper left and bottom right vertices of the rectangle

Returns Binary mask with True inside of the rectangle, False outside.

Return type np.ndarray

region_coordinates(*region*)

Extract bbox coordinates from the region.

Parameters **region** (*Region*) – Region from which to extract the coordinates of the bbox

Returns Coordinates of the bbox

Return type CoordinatePair

regions_from_binary_mask(*binary_mask*)

Calculate regions properties from a binary mask.

Parameters **binary_mask** (*np.ndarray*) – Binary mask from which to extract the regions

Returns Properties for all the regions present in the binary mask

Return type List[Region]

`regions_to_binary_mask`(*regions*, *dims*)

Create a binary mask given a list of `regions`.

For each region `r`, the areas within `r.coords` are filled with True, False outside.

Parameters

- **`regions`** (*`List[Region]`*) – The regions to create the binary mask.
- **`dims`** (*`Tuple[int, int]`*) – Dimensions of the resulting binary mask.

Returns Binary mask from the `regions` coordinates.

Return type np.ndarray

`scale_coordinates`(*reference_coords*, *reference_size*, *target_size*)

Compute the coordinates corresponding to a scaled version of the image.

Parameters

- **`reference_coords`** (*`CoordinatePair`*) – Coordinates referring to the upper left and lower right corners respectively.
- **`reference_size`** (*`tuple of int`*) – Reference (width, height) size to which input coordinates refer to
- **`target_size`** (*`tuple of int`*) – Target (width, height) size of the resulting scaled image

Returns **coords** – Coordinates in the scaled image

Return type CoordinatesPair

`threshold_to_mask`(*img*, *threshold*, *relate*)

Mask image with pixel according to the threshold value.

Parameters

- **`img`** (*`PIL.Image.Image`*) – Input image
- **`threshold`** (*`float`*) – The threshold value to exceed.
- **`relate`** (*`callable operator`*) – Comparison operator between img pixel values and threshold

Returns Boolean NumPy array representing a mask where a pixel has a value True if the corresponding input array pixel exceeds the threshold value. if the corresponding input array pixel exceeds the threshold value.

Return type np.ndarray

6.15 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX