spotlob Documentation

Release 0.9

Fabian Meyer

Jun 17, 2021

Contents

1 When it's helpful **3**
- 1.1 Usage example . . . 3

2 What it's not **5**

3 Installation **7**

4 Further topics **9**
- 4.1 Basic workflow . . . 9
- 4.2 Priciple of operation . . . 10
- 4.3 Extending functionality . . . 13
- 4.4 Structures . . . 13
- 4.5 Default Pipeline . . . 21

5 Indices and tables **23**

Python Module Index **25**

Index **27**

Spotlob is a package to provide a simple, yet flexible and fast workflow to measure properties of features in images for scientific purposes.

It provides implementations for some use cases but can be easily tuned and be extended towards specific applications. Jupyter notebook widgets can be used to quickly find a set of algorithms and parameters that work for a given image and should also work for similar images.

The set of parameters and algorithms are stored as a pipeline which can be restored and distributed and then be applied to a possibly large set of images. This way, standard routines for repetitive and comparable measurements for a defined type of images can be forged into a small and portable file.

CHAPTER 1

When it's helpful

It is meant to be used in a scenario where a detection method has to be applied repetetively onto a large set of similar images, but the exact parameters are not clear. If your set of tasks can be done by a collection of opencv-function calls, that need tweaking and you wish to have a GUI to do that, but without to lose scripting options, spotlob is for you.

If you already have a couple of working python algorithms and want to have a GUI for them to play around, use spotlob.

If you need to evaluate some images and you don't know which of the thousand parameters of an algorithm work best, you might be able to find the right ones faster with spotlob.

1.1 Usage example

```
from spotlob.spim import Spim
from spotlob.defaults import default_pipeline

my_spim = Spim.from_file("image.jpg", cached=True)
my_pipe = default_pipeline()

result_spim = my_pipe.apply_all_steps(my_spim)

print(result_spim.get_data())
```

CHAPTER 2

What it's not

Spotlob is not a complete feature detection library and it does not solve a detection problem, that has not already been solved elsewhere. It is not an alternative to opencv or scikit-image, but rather builds on top of it. At the moment it covers only a tiny fraction of what is possible with these libraries, but it tries to make it easy for the reader to use these (or any other python image processing library) within the spotlob workflow.

Although it might work with machine learning algorithms, it is not tuned towards this usage and it is not designed with this application in mind.

CHAPTER 3

Installation

Install with pip

```
pip install spotlob
```

CHAPTER 4

Further topics

4.1 Basic workflow

Spotlob can be used in various ways: as a library to be integrated in another software, as a framework for development of a image detection task or as a software for semi-automated feature detection for easy up to moderately difficult detection tasks. The following suggested workflow adresses the latter.

4.1.1 1. Tweak parameters using the notebook

Finding a parameter set for an arbitrary feature detection task can be challenging. The approach of spotlob is to use the interactive features of Jupyter to facilitate the tweaking of parameters towards a detection. If you have Jupyter installed, open the *notebooks/simple gui.ipynb* notebook file. Insert the path the image that you wish to analyze:

```
filename = "path/to/your/file.jpg"
```

Then run all the cells up to *show_gui(gui)*. An interactive image preview should appear an you can play around with the sliders. A change of parameter values will be reflected as a preview. Once you press *Evaluate*, the contours are analyzed.

The result of the analysis can be requested as a dataframe with

```
gui.results()
```

4.1.2 2. Store and reload a pipeline

The pipeline holds all the information of the detection - applying the same pipeline on the same image should always yield the same results. A pipeline can be saved to a file, to repeat the same detection later.

To save the parameters and selection of processes, that has been used in the gui, access the pipeline object through the *gui.pipeline* field and then store it:

```
mypipe = gui.pipeline
mypipe.save("my_pipe.pipe")
```

To restore a saved pipeline object

```
from spotlob.pipeline import Pipeline

restored_pipe = Pipeline.from_file("my_pipe.pipe")
```

4.1.3 3. Batch processing

If you have multiple image files, that you wish to evaluate all the same way, you can use the *batch* module to apply a saved pipeline on all of them and get the results collected in a single dataframe.

```
from spotlob.batch import batchprocess

results = batchprocess("my_pipe.pipe",
                       ["file1.jpg",... , "fileN.jpg"],
                       multiprocessing=True)
```

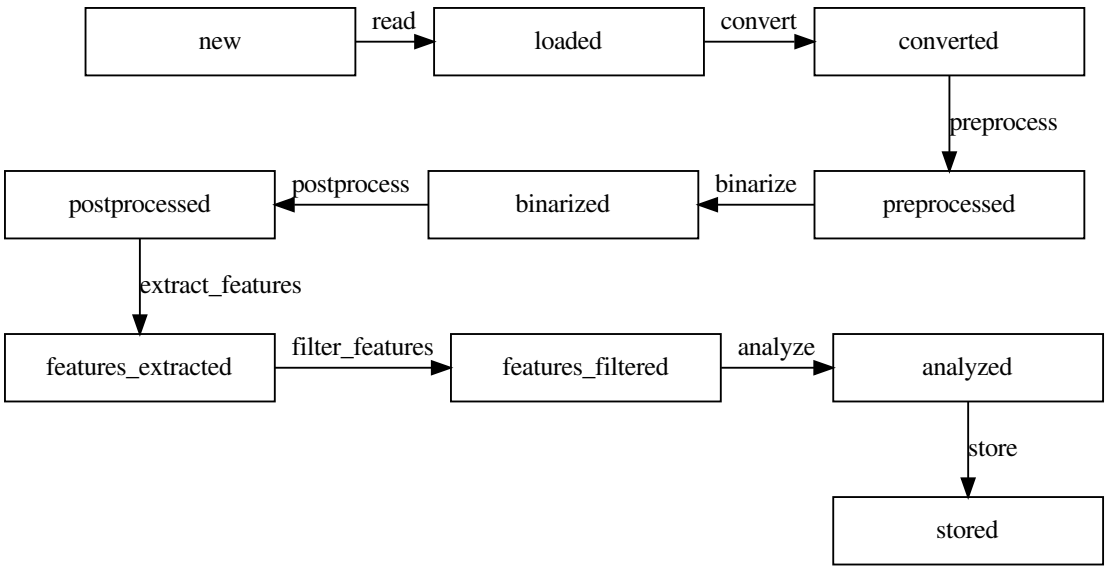
If the *multiprocessing* parameter is set *True*, all available cpu cores are used for parallel execution, giving a significant speedup on multi-core systems.

4.2 Priciple of operation

4.2.1 The Spim and its stages

A Spim is the object holding the images and metadata. It has methods, that return a Spim of the next stage. For example, a blank, empty Spim can be created and is then in the stage *SpimStage.new*. It contains only the information where to find the image file. If *Spim.read(Writer)* is called, a new Spim is returned, which contains the image data and is at stage *SpimStage.loaded*.

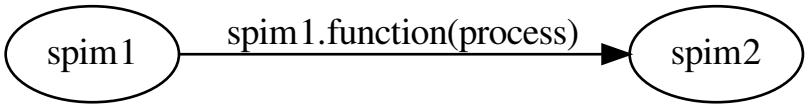
Here is a list of the stages that a Spim can be in and in between, the methods that return a Spim of the next stage.



With every step, information is collected. A spim at a later stage does not duplicate the image data from former stages. However, if this data is still needed, it can contain a reference to its predecessors.

4.2.2 The processes

The detection of features within an image with spotlob is split up into an abstract but fixed sequence of processes. Any of these process steps is applied onto a Spim and returns a new Spim. The new Spim contains the information added by the process step.



For example, a *Spim* at stage *SpimStage.loaded* can be converted, using a concrete subclass of *Converter*, named *process_opencv.GreyscaleConverter* in the following way:

```
from spotlob.defaults import load_image
from spotlob.process_opencv import GreyscaleConverter

loaded_spim = load_image("color_image.jpg")
my_converter = GreyscaleConverter()
converted_spim = loaded_spim.convert(my_converter)
```

Any process step corresponds to one input stage and one method of Spim to use it with

input stage	Spim method	SpotlobProcessStep subclass
new	read	Reader
loaded	convert	Converter
converted	preprocess	Preprocessor
preprocessed	binarize	Binarization
binarized	postprocess	Postprocessor
postprocessed	extract_features	FeatureExtractor
features_extracted	filter_features	FeatureFilter
features_filtered	analyze	Analysis
analyzed	store	Writer
stored		

For every function of *Spim* that returns another *Spim* at a further stage, there is a subclass of *SpotlobProcessStep*, that can be used as super class for an concrete implementation of that step. The return type of a *SpotlobProcessStep.apply* call is different depending on the type of process. The Spim internally passes the modified data to the new Spim created through the process.

class spotlob.process_steps.**Reader**(*function*, *parameters*, *add_to_register=True*)
A reader loads the image data from storage into memory. *apply* returns an image

class spotlob.process_steps.**Converter**(*function*, *parameters*, *add_to_register=True*)
A converter converts a color image to a greyscale image. *apply* returns a greyscale image

class spotlob.process_steps.**Preprocessor**(*function*, *parameters*, *add_to_register=True*)
Preprocessing is applied onto a grey image to prepare for binarization, for example by cleaning the image from unwanted features. *apply* returns a greyscale image

class spotlob.process_steps.**Binarization**(*function*, *parameters*, *add_to_register=True*)
Turns a greyscale image into a black-and-white or binary image. *apply* returns a greyscale image

class spotlob.process_steps.**Postprocessor**(*function*, *parameters*, *add_to_register=True*)
Postprocessing is done on a binary image to facilitate the detection. *apply* returns a greyscale image

class spotlob.process_steps.**FeatureFinder**(*function*, *parameters*, *add_to_register=True*)
The FeatureFinder tries to find contours in a binary image.

apply returns contours

class spotlob.process_steps.**FeatureFilter**(*function*, *parameters*, *add_to_register=True*)
The FeatureFilter can reduce the number of detected features by analyzing them.

apply returns contours

class spotlob.process_steps.**Analysis**(*function*, *parameters*, *add_to_register=True*, *extended_output=True*)
An Analysis class evaluates the metadata (including contours) and yields its results as a dataframe.

apply returns `DataFrame`

4.2.3 The spotlob pipeline

The pipeline structure is used to define a sequence of processes to be applied one after another onto a spim, to automate a detection task consisting of multiple process steps.

class spotlob.pipeline.**Pipeline**(*processes*)
A pipeline is a sequence of processes, that can be applied one after another. The processes are stored in a Dictionary, along with the SpimStage at which they can be applied. The pipeline can be applied completely using the *apply_all_steps* method or partially using the *apply_from_stage_to_stage* method.

4.3 Extending functionality

The register is meant to be used to keep track of available process steps. Using decorators, a function can be internally turned into a `SpotlobProcessStep` subclass, to be used within a `Pipeline`. This way, using only minimal code, new functionality can be added to Spotlob and directly used within its workflow

4.3.1 Example using a decorator

Create a process register

```
from spotlob.register import ProcessRegister
register = ProcessRegister()
```

Create an alternative function, that should replace a single `SpotlobProcessStep`. Here, a binarization function is defined, using upper and lower value boundaries, with *numpy*.

```
import numpy as np

def my_threshold(image, lower_threshold, upper_threshold, invert):
    out = np.logical_and( image > lower_threshold,
                          image < upper_threshold)
    out = out.astype(np.uint8)*255
    if invert:
        out = ~out
    return out
```

To add this function to the register, use the methods of `ProcessRegister` as decorators, giving a list of parameter specifications

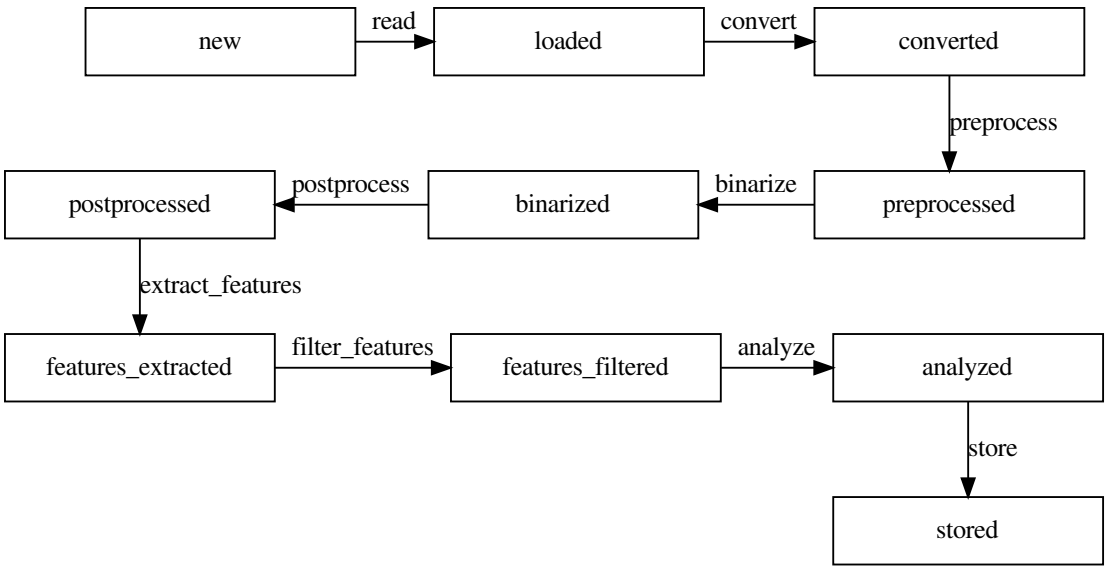
```
@register.binarization_plugin([("lower_threshold",(0,255,100)),
                               ("upper_threshold",(0,255,200)),
                               ("invert", True)])
def my_threshold(image, lower_threshold, upper_threshold, invert):
    out = np.logical_and( image > lower_threshold,
                          image < upper_threshold)
    out = out.astype(np.uint8)*255
    if invert:
        out = ~out
    return out
```

4.4 Structures

4.4.1 Spim

A Spim is the object holding the images and metadata. It has methods, that return a Spim of the next stage. For example, a blank, empty Spim can be created and is then in the stage *SpimStage.new*. It contains only the information where to find the image file. If *Spim.read(Writer)* is called, a new Spim is returned, which contains the image data and is at stage *SpimStage.loaded*.

Here is a list of the stages that a Spim can be in and in between, the methods that return a Spim of the next stage.



With every step, information is collected. A spim at a later stage does not duplicate the image data from former stages. However, if this data is still needed, it can contain a reference to its predecessors.

class spotlob.spim.**Spim**(*image*, *metadata*, *stage*, *cached*, *predecessors*)

Spotlob image item

do_process_at_stage(*process*)

Apply the given process at at this Spim if the process fits this stage or at a predecessor of this Spim that fits the process' input stage

Parameters **process** (SpotlobProcessStep) – Process to apply

Returns The Spim that results from the process being applied. It is in stage *process.input_stage + 1*

Return type *Spim*

classmethod from_file(*image_filepath*, *cached=False*)

Create a Spim object from an image file. The path is stored in the Spim object, but the image is not yet loaded.

Parameters

- **image_filepath** (*str*) – Path to an image file. The image type must be understood by the reader that is given when the *read*-function is called. If an invalid image type is given at this stage, it will not be recognized
- **cached** (*bool, optional*) – If the spim is to be cached, a reference to predecessors will be kept and not be deleted by the garbage collector. This allows to go back to an earlier stage after applying processes, but is more memory consuming. (the default is False)

Returns An empty Spim at SpimStage.new, that does not contain any data except the filepath

Return type *Spim*

func_at_stage(*spimstage*)

The method like *self.read()*, *self.convert()*,... that can be safely called at the given stage

Parameters **spimstage** (*int*) – SpimStage that the requested method corresponds to

Returns the function, that can be applied the given stage

Return type callable

get_at_stage(*spimstage*)
Get the Spim at a given stage. This returns a predecessor if it has been chached

Parameters **spimstage** (*int*) – That the returned Spim should be at

Raises **Exception** – If there is no predecessor at the requested stage, for example if Spim has not been cached

Returns The Spim at the requested Stage

Return type *Spim*

get_data()
get all metadata and results as flat metadata

Returns all metadata including collected results

Return type pandas.Dataframe

image
Gives the image contained in this Spim or in the latest predecessor, that has an image

Raises **Exception** – Exception is raised if no image is present, most likely because it has not been cached

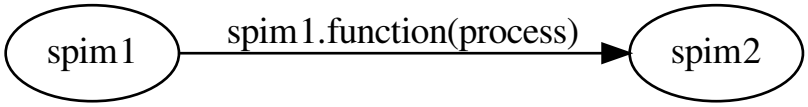
Returns latest image

Return type numpy.array

class spotlob.spim.**SpimStage**
Enumeration of the stages that a Spim can go through

4.4.2 Process

The detection of features within an image with spotlob is split up into an abstract but fixed sequence of processes. Any of these process steps is applied onto a Spim and returns a new Spim. The new Spim contains the information added by the process step.



For example, a *Spim* at stage *SpimStage.loaded* can be converted, using a concrete subclass of *Converter*, named *process_opencv.GreyscaleConverter* in the following way:

```
from spotlob.defaults import load_image
from spotlob.process_opencv import GreyscaleConverter
```

(continues on next page)

(continued from previous page)

```
loaded_spim = load_image("color_image.jpg")
my_converter = GreyscaleConverter()
converted_spim = loaded_spim.convert(my_converter)
```

class spotlob.process.**SpotlobProcessStep**(*function*, *parameters*, *add_to_register=True*)

An abstract super class for process steps. A process step can be applied to bring a spim from one stage to another. It is supposed to save as internal state, if it has already been applied or needs to be applied again, because the parameters have changed. This is stored in the *outdated* parameter

apply(**input_args*)

Apply the process step onto input data (eg. images or contours) and yield the output data (again, could be a modified image or gathered data). The actual return values depend on the concrete implementation of the function. Additional arguments for the function are the parameters of the process as stored in the *parameters* field of each process.

Returns Output of the concrete process implementation

Return type Contours or images

preview(*spim*)

This function takes spim at an undefined stage and draws the effect of the process on top, to provide a preview for the user on how the funciton will work. No storage or sideeffects should take place. In contrast to the apply function it must always return an image

Parameters **spim** (Spim) – Spim to draw the preview on. It must contain an image

For every function of *Spim* that returns another *Spim* at a further stage, there is a subclass of *SpotlobProcessStep*, that can be used as super class for an concrete implementation of that step. The return type of a *SpotlobProcessStep.apply* call is different depending on the type of process. The Spim internally passes the modified data to the new Spim created through the process.

class spotlob.process_steps.**Analysis**(*function*, *parameters*, *add_to_register=True*, *extended_output=True*)

An Analysis class evaluates the metadata (including contours) and yields its results as a dataframe.

apply returns DataFrame

preview(*spim*)

This function takes spim at an undefined stage and draws the effect of the process on top, to provide a preview for the user on how the funciton will work. No storage or sideeffects should take place. In contrast to the apply function it must always return an image

Parameters **spim** (Spim) – Spim to draw the preview on. It must contain an image

class spotlob.process_steps.**Binarization**(*function*, *parameters*, *add_to_register=True*)

Turns a greyscale image into a black-and-white or binary image. *apply* returns a greyscale image

preview(*spim*)

This function takes spim at an undefined stage and draws the effect of the process on top, to provide a preview for the user on how the funciton will work. No storage or sideeffects should take place. In contrast to the apply function it must always return an image

Parameters **spim** (Spim) – Spim to draw the preview on. It must contain an image

class spotlob.process_steps.**Converter**(*function*, *parameters*, *add_to_register=True*)

A converter converts a color image to a greyscale image. *apply* returns a greyscale image

preview(*spim*)

This function takes spim at an undefined stage and draws the effect of the process on top, to provide a preview for the user on how the funciton will work. No storage or sideeffects should take place. In contrast to the apply function it must always return an image

Parameters **spim** (Spim) – Spim to draw the preview on. It must contain an image

class spotlob.process_steps.**FeatureFilter**(*function*, *parameters*, *add_to_register=True*)
The FeatureFilter can reduce the number of detected features by analyzing them.

apply returns contours

preview(*spim*)
This function takes spim at an undefined stage and draws the effect of the process on top, to provide a preview for the user on how the funciton will work. No storage or sideeffects should take place. In contrast to the apply function it must always return an image

Parameters **spim** (Spim) – Spim to draw the preview on. It must contain an image

class spotlob.process_steps.**FeatureFinder**(*function*, *parameters*, *add_to_register=True*)
The FeatureFinder tries to find contours in a binary image.

apply returns contours

preview(*spim*)
This function takes spim at an undefined stage and draws the effect of the process on top, to provide a preview for the user on how the funciton will work. No storage or sideeffects should take place. In contrast to the apply function it must always return an image

Parameters **spim** (Spim) – Spim to draw the preview on. It must contain an image

class spotlob.process_steps.**Postprocessor**(*function*, *parameters*, *add_to_register=True*)
Postprocessing is done on a binary image to facilitate the detection. *apply* returns a greyscale image

preview(*spim*)
This function takes spim at an undefined stage and draws the effect of the process on top, to provide a preview for the user on how the funciton will work. No storage or sideeffects should take place. In contrast to the apply function it must always return an image

Parameters **spim** (Spim) – Spim to draw the preview on. It must contain an image

class spotlob.process_steps.**Preprocessor**(*function*, *parameters*, *add_to_register=True*)
Preprocessing is applied onto a grey image to prepare for binarization, for example by cleaning the image from unwanted features. *apply* returns a greyscale image

preview(*spim*)
This function takes spim at an undefined stage and draws the effect of the process on top, to provide a preview for the user on how the funciton will work. No storage or sideeffects should take place. In contrast to the apply function it must always return an image

Parameters **spim** (Spim) – Spim to draw the preview on. It must contain an image

class spotlob.process_steps.**Reader**(*function*, *parameters*, *add_to_register=True*)
A reader loads the image data from storage into memory. *apply* returns an image

Process step implementations

class spotlob.process_opencv.**BinaryThreshold**(*threshold*)
Converts the image to a binary one, where the parts above the given threshold are set to 255 and the parts below it to 0. The sole parameter is the threshold value. It uses the *cv2.threshold* function.

class spotlob.process_opencv.**ContourFinder**(*mode*)
Finds contours, i.e. lists of points that enclose connected areas of the same value. It is based on the *cv2.findContours* function. It can distinguish between different levels of nested areas

Parameters **mode** (*string*) – Select which kind of blobs should be found and the contour of which should be returned. Select of the following

- all = all contours, both holes and non-holes
- inner = innermost blobs without holes in them
- outer = only outermost blobs
- holes = only holes, that are contained in other blobs
- non-holes = all blobs, that are not holes

class spotlob.process_opencv.**ContourFinderSimple**
Finds contours, i.e. lists of points that enclose connected areas of the same value. It is based on the *cv2.findContours* function

class spotlob.process_opencv.**FeatureFormFilter**(*size*, *solidity*, *remove_on_edge*)
It analyzes the contours and filters them using given criteria:

- the enclosed area must be smaller (i.e. contain fewer pixels) than *minimal_area*
- it solidity, i.e. the ratio of the area of the contour and its convex hull must be below a given value
- if *remove_on_edge* is *True*, contours that touch the border of the image are filtered out

is_off_border(*contour*, *image_shape*)
this function checks if a contour is touching the border of an image shaped like image_shape

class spotlob.process_opencv.**GaussianPreprocess**(*ksize*)
Blur the image with a gaussian blur with kernel size given by *ksize*. It uses the *cv2.filter2D* function

class spotlob.process_opencv.**GreyscaleConverter**
Converts a color image to a greyscale image, by selecting one channel or by converting it to another color space and then selecting one channel.

The supported options are given by the *conversion* parameter, which must be one of the following strings - *Hue*, *Saturation* or *Value* channel - *Red*, *Blue* or *Green* color channel - normal *Greyscale* conversion

It uses the *cv2.cvtColor* function. Additionally the dark an bright parts can be switched using *invert=True*

class spotlob.process_opencv.**OtsuThreshold**
Performs a binarization based on Otsu's algorithm. It uses the *cv2.threshold* function.

class spotlob.process_opencv.**PostprocessNothing**
This process is used as a placeholder for a postprocessing step and does not modify the image at all

class spotlob.process_opencv.**SimpleReader**
Reads an image from a file as an RGB file. Standard image formats, such as *png*, *jpg*, *tif* are supported. It uses *cv2.imread*.

class spotlob.process_opencv.**TifReader**
Reads an image from a file as an RGB file. Only image format *tif* is supported. It uses *tifffile.memmap*.

partial_read(*filepath*, *width_percent*, *height_percent*, *x0_percent*, *y0_percent*)
Returns an array of a part of an .tif image. The arguments must be percentages, even the startingpoint is relative. Starting Point is the top-left corner.

preview(*spim*)
This function takes spim at an undefined stage and draws the effect of the process on top, to provide a preview for the user on how the funciton will work. No storage or sideeffects should take place. In contrast to the apply function it must always return an image

Parameters **spim** (Spim) – Spim to draw the preview on. It must contain an image

class spotlob.analyze_circle.**CircleAnalysis**(*calibration=None*, *extended_output=True*)

class spotlob.analyze_line.**LineAnalysis**(*calibration=None*, *linewidth_percentile=95*, *extended_output=True*)

4.4.3 Pipeline

The pipeline structure is used to define a sequence of processes to be applied one after another onto a spim, to automate a detection task consisting of multiple process steps.

class spotlob.pipeline.**Pipeline**(*processes*)

A pipeline is a sequence of processes, that can be applied one after another. The processes are stored in a Dictionary, along with the SpimStage at which they can be applied. The pipeline can be applied completely using the *apply_all_steps* method or partially using the *apply_from_stage_to_stage* method.

apply_all_steps(*spim*)

Apply the complete pipeline on a given spim

Parameters **spim** (Spim) – the spotlob image item to apply the complete pipeline to

Returns a spotlob image item a the stage after the last process

Return type *Spim*

apply_at_stage(*spim*)

Applies all steps following the stage of the spim

Parameters **spim** (Spim) – the spotlob image item to apply the pipeline to

Returns the processed Spim at stage *to_stage*

Return type *Spim*

apply_from_stage_to_stage(*spim*, *from_stage*, *to_stage*)

Recursively applies the pipeline-processes from a given stage up to another given stage.

Parameters

- **spim** (Spim) – The image item to apply parts of the pipeline to
- **from_stage** (*int*) – SpimStage at which stage the first process should be applied
- **to_stage** (*int*) – SpimStage at which stage the last process should be applied

Returns The processed Spim at stage *to_stage*

Return type *Spim*

Raises **Exception:** – If to_stage is before from_stage

apply_outdated_up_to_stage(*spim*, *up_to_stage*)

Applies all processes since the first outdated one on spim up to a given stage if no process is outdated or if the outdated stage is past up_to_stage, spim is processed up to up_to_stage, or a predecessor is returned at up_to_stage

Parameters

- **spim** (Spim) – Spim to apply the pipeline to
- **up_to_stage** (*int*) – SpimStage up to which the pipeline should be applied

Returns the processed Spim at stage *up_to_stage*

Return type *Spim*

classmethod from_file(*filepath*)

Restore a pipeline from a file

Parameters **filepath** (*str*) – filepath of the pipeline file

Returns the restored pipeline object

Return type Pipepline

replaced_with(*new_process*)
This will give a new pipeline, where one process is replaced with the given one

Parameters **new_process** (SpotlobProcessStep) – the new process to be inserted

Returns the pipeline, that includes *new_process*

Return type *Pipeline*

save(*target_path*)
Store the pipeline including process paramaters for later use.

Parameters **target_path** (*str*) – path of the file to store the pipeline to

Notes

This creates a file that is also suitable for batch process and parallelization.

See also:

Pipeline.from_file() Use *Pipeline.from_file* to restore the pipeline object from storage

4.4.4 Parameters

spotlob.parameters.**parameter_from_spec**(*spec*)
This function will create a SpotlobParameter from a specification

Parameters **spec** (*tuple(str, object)*) – specification for the parameter, must be one of the following options:

float range	**(“parameter_name”, (float_min_value,** float_max_value, float_value))
integer range	**(“parameter_name”, (int_value,** int_min_value, int_max_value))
boolean value	(“parameter name”, boolean)
enumeration	**(“parameter name”, [“option1”,** ”option2”, “option3”])

Returns An instance of a SpotlobParameter subclass: EnumParameter, FloatParameter, ... depending on the type of the spec

Return type SpotlobParameter

4.4.5 Batch processing

spotlob.batch.**batchprocess**(*pipeline_file*, *image_files*, *multiprocessing=False*)
This function applies a pipeline from a file onto a stack of images. The results are collected in one pandas.Dataframe.

Parameters

- **pipeline_file** (*str*) – the filepath of a pickled pipeline
- **image_files** (*list of str*) – paths of the images
- **multiprocessing** (*bool, optional*) – if True, the processing will be done in parallel using multiple cpu cores at once.

Returns Flat Dataframe where one row corresponds to one detected feature

Return type pandas.Dataframe

spotlob.batch.**is_interactive**()
checks wether called in an interactive environment

4.5 Default Pipeline

Spotlob comes with a predefined pipeline, which has some standard routines built-in. It consists of the following process steps

1. *SimpleReader*
2. *GreyscaleConverter*
3. *GaussianPreprocess*
4. *OtsuThreshold* or *BinaryThreshold*
5. *PostprocessNothing*
6. *ContourFinderSimple*
7. *FeatureFormFilter*
8. CircleAnalysis or LineAnalysis

The default pipeline can be configured with parameters of returned by the following function

spotlob.defaults.**default_pipeline**(*mode='circle'*, *thresholding='auto'*)
Gives a pipeline which works for many cases and can be used as a starting point for further tuning or as default for the GUI notebook. By default, features with an area smaller than 500 pixels are ignored.

Parameters

- **mode** (*str, optional*) – this defines the way in which detected features will be evaluated
  - as *line* the linewidth is calculated
  - as *circle* an ellipse is fitted and by default features that touch the edge of the image get ignored (the default is "circle")
- **thresholding** (*str, optional*) –
  - *auto* uses Otsu's thresholding algorithm
  - *simple* uses a fixed threshold value, 100 by default (the default is "auto")

Returns the pipeline including default parameters

Return type *Pipeline*

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

spotlob.analyze_circle, 18
spotlob.analyze_line, 18
spotlob.batch, 21
spotlob.parameters, 20
spotlob.pipeline, 19
spotlob.process, 15
spotlob.process_opencv, 17
spotlob.process_steps, 16
spotlob.register, 13
spotlob.spim, 13

Index

A

Analysis (*class in spotlob.process_steps*), 16
apply() (*spotlob.process.SpotlobProcessStep method*), 16
apply_all_steps() (*spotlob.pipeline.Pipeline method*), 19
apply_at_stage() (*spotlob.pipeline.Pipeline method*), 19
apply_from_stage_to_stage() (*spotlob.pipeline.Pipeline method*), 19
apply_outdated_up_to_stage() (*spotlob.pipeline.Pipeline method*), 19

B

batchprocess() (*in module spotlob.batch*), 21
Binarization (*class in spotlob.process_steps*), 16
BinaryThreshold (*class in spotlob.process_opencv*), 17

C

CircleAnalysis (*class in spotlob.analyze_circle*), 18
ContourFinder (*class in spotlob.process_opencv*), 17
ContourFinderSimple (*class in spotlob.process_opencv*), 18
Converter (*class in spotlob.process_steps*), 16

D

default_pipeline() (*in module spotlob.defaults*), 21
do_process_at_stage() (*spotlob.spim.Spim method*), 14

F

FeatureFilter (*class in spotlob.process_steps*), 17
FeatureFinder (*class in spotlob.process_steps*), 17
FeatureFormFilter (*class in spotlob.process_opencv*), 18
from_file() (*spotlob.pipeline.Pipeline class method*), 19
from_file() (*spotlob.spim.Spim class method*), 14
func_at_stage() (*spotlob.spim.Spim method*), 14

G

GaussianPreprocess (*class in spotlob.process_opencv*), 18
get_at_stage() (*spotlob.spim.Spim method*), 15
get_data() (*spotlob.spim.Spim method*), 15
GreyscaleConverter (*class in spotlob.process_opencv*), 18

I

image (*spotlob.spim.Spim attribute*), 15
is_interactive() (*in module spotlob.batch*), 21
is_off_border() (*spotlob.process_opencv.FeatureFormFilter method*), 18

L

LineAnalysis (*class in spotlob.analyze_line*), 18

O

OtsuThreshold (*class in spotlob.process_opencv*), 18

P

parameter_from_spec() (*in module spotlob.parameters*), 20
partial_read() (*spotlob.process_opencv.TifReader method*), 18
Pipeline (*class in spotlob.pipeline*), 19
PostprocessNothing (*class in spotlob.process_opencv*), 18
Postprocessor (*class in spotlob.process_steps*), 17
Preprocessor (*class in spotlob.process_steps*), 17
preview() (*spotlob.process.SpotlobProcessStep method*), 16
preview() (*spotlob.process_opencv.TifReader method*), 18

preview() (*spotlob.process_steps.Analysis method*), 16
preview() (*spotlob.process_steps.Binarization method*), 16
preview() (*spotlob.process_steps.Converter method*), 16
preview() (*spotlob.process_steps.FeatureFilter method*), 17
preview() (*spotlob.process_steps.FeatureFinder method*), 17
preview() (*spotlob.process_steps.Postprocessor method*), 17
preview() (*spotlob.process_steps.Preprocessor method*), 17

R

Reader (*class in spotlob.process_steps*), 17
replaced_with() (*spotlob.pipeline.Pipeline method*), 20

S

save() (*spotlob.pipeline.Pipeline method*), 20
SimpleReader (*class in spotlob.process_opencv*), 18
Spim (*class in spotlob.spim*), 14
SpimStage (*class in spotlob.spim*), 15
spotlob.analyze_circle (*module*), 18
spotlob.analyze_line (*module*), 18
spotlob.batch (*module*), 21
spotlob.parameters (*module*), 20
spotlob.pipeline (*module*), 19
spotlob.process (*module*), 15
spotlob.process_opencv (*module*), 17
spotlob.process_steps (*module*), 16
spotlob.register (*module*), 13
spotlob.spim (*module*), 13
SpotlobProcessStep (*class in spotlob.process*), 16

T

TifReader (*class in spotlob.process_opencv*), 18