satproc Documentation

Release unknown

Dymaxion Labs

Apr 11, 2023

CONTENTS

CHAPTER

ONE

CONTENTS

1.1 satproc

Python library and CLI tools for processing geospatial imagery for ML

*This digital tool is part of the catalog of tools of the **Inter-American Development Bank**. You can learn more about the IDB initiative at code.iadb.org*

1.1.1 Documentation

- Stable
- Latest

1.1.2 Bugs / Questions

- Report bugs/feature requests
- Ask questions in our chat room

1.1.3 Contributing

Bug reports and pull requests are welcome on GitHub at the issues page. This project is intended to be a safe, welcoming space for collaboration, and contributors are expected to adhere to the Contributor Covenant code of conduct.

Made with contrib.rocks.

The current roadmap is available at GitHub at the projects page.

1.1.4 License

This project is licensed under Apache 2.0. Refer to LICENSE.txt.

1.2 Installation

1.2.1 Stable release

Use pip to install from PyPI:

Install from pip:

```
pip install pysatproc
```

1.2.2 From source

The source for satproc can be installed from the GitHub repo.

```
python -m pip install git+https://github.com/dymaxionlabs/satproc.git
```

To install for local development, you can clone the repository:

```
git clone https://github.com/dymaxionlabs/satproc.git
```

If you don't have Poetry installed, follow these instructions first.

Then, install all dependencies. Poetry will automatically create a virtual environment for you:

```
cd satproc
poetry install
```

Whenever you want to bring the latest changes, just run `git pull` from the cloned repository.

1.3 Getting started

satproc helps you work with large amount of geospatial raster images (satellite, drone, etc.) and process them for training machine learning, for object detection or semantic segmentation problems.

You can either use it from the command line or from Python as a library.

1.3.1 Usage

Command Line (CLI)

When installed, satproc makes available a series of command-line scripts to process files without resorting to writing a Python script.

- `satproc_extract_chips`: Extract chips from raster images, optionally creating masks for each chip using a labels vector file.
- `satproc_make_masks`: Create masks from raster images and a labels vector file.
- `satproc_polygonize`: Polygonizes chip images into a single polygon vector file.
- `satproc_generalize`: Generalizes vector files by simplyfing and smoothing polygon boundary lines.
- `satproc_smooth_stitch`: Smoothes overlapping probability result chips.

- `satproc_spatial_filter`: Applies a spatial filter (median, gaussian) to an image
- `satproc_scale`: Rescales values from raster images
- `satproc_match_histograms`: Matches histograms of raster images from a reference image.

Run any command with the `-h`/`--help` flag to see the available options and information on how to use them.

1.4 Tutorials

The following tutorials showcase how to use the command line tools provided by **satproc** for different tasks.

1.4.1 Extract chips from images

To do…

[1]:
```
!satproc_extract_chips -h
```

```
usage: satproc_extract_chips [-h] [--size SIZE] [--step-size STEP_SIZE]
                             [--sliding-windows-mode {exact,whole,whole_overlap}]
                             [--labels LABELS]
                             [--label-property LABEL_PROPERTY]
                             [--classes CLASSES [CLASSES ...]]
                             [--masks {extent,boundary,distance} [{extent,boundary,
↪distance} ...]]
                             [--mask-type {single,class,instance}] [--aoi AOI]
                             [--within] [--no-within] [-o OUTPUT_DIR]
                             [--rescale] [--no-rescale]
                             [--rescale-mode {percentiles,values,s2_rgb_extra}]
                             [--lower-cut LOWER_CUT] [--upper-cut UPPER_CUT]
                             [--min MIN] [--max MAX] [-b BANDS [BANDS ...]]
                             [-t {jpg,tif}] [--write-footprints]
                             [--no-write-footprints] [--crs CRS]
                             [--skip-existing] [--no-skip-existing]
                             [--skip-low-contrast] [--no-skip-low-contrast]
                             [--extent-no-border] [--version] [-v] [-vv]
                             raster [raster ...]

Extract chips from a raster file, and optionally generate mask chips

positional arguments:
  raster                input raster file(s)

optional arguments:
  -h, --help            show this help message and exit
  --size SIZE           size of image tiles, in pixels (default: 256)
  --step-size STEP_SIZE
                        step size (i.e. stride), in pixels (default: 128)
  --sliding-windows-mode {exact,whole,whole_overlap}
                        mode of sliding windows (default: whole_overlap)
  --labels LABELS       inpul label shapefile (default: None)
  --label-property LABEL_PROPERTY
                        label property to separate in classes (default: class)
```

(continues on next page)

(continued from previous page)

```
  --classes CLASSES [CLASSES ...]
                        specify classes order in result mask. (default: None)
  --masks {extent,boundary,distance} [{extent,boundary,distance} ...], -m {extent,
↪boundary,distance} [{extent,boundary,distance} ...]
  --mask-type {single,class,instance}
  --aoi AOI             Filter by AOI vector file (default: None)
  --within              only create chip is it is within AOI (if provided)
                        (default: False)
  --no-within           create chip if it intersects with AOI (if provided)
                        (default: False)
  -o OUTPUT_DIR, --output-dir OUTPUT_DIR
                        output dir (default: .)
  --rescale             rescale intensity using percentiles (lower/upper cuts)
                        (default: False)
  --no-rescale          do not rescale intensity (default: True)
  --rescale-mode {percentiles,values,s2_rgb_extra}
                        choose mode of intensity rescaling (default:
                        percentiles)
  --lower-cut LOWER_CUT
                        (for 'percentiles' mode) lower cut of percentiles for
                        cumulative count in intensity rescaling (default: 2)
  --upper-cut UPPER_CUT
                        (for 'percentiles' mode) upper cut of percentiles for
                        cumulative count in intensity rescaling (default: 98)
  --min MIN             (for 'values' mode) minimum value in intensity
                        rescaling (default: None)
  --max MAX             (for 'values' mode) maximum value in intensity
                        rescaling (default: None)
  -b BANDS [BANDS ...], --bands BANDS [BANDS ...]
                        specify band indexes. If type is 'jpg', defaults to
                        (1, 2, 3). If type is 'tif', defaults to the total
                        band count. (default: None)
  -t {jpg,tif}, --type {jpg,tif}
                        output chip format (default: tif)
  --write-footprints    write a GeoJSON file of chip footprints (default:
                        False)
  --no-write-footprints
                        do not write a GeoJSON file of chip footprints
                        (default: True)
  --crs CRS             force CRS of input files (default: None)
  --skip-existing       skip already existing chips (and masks) (default:
                        True)
  --no-skip-existing    do not skip already existing chips (and masks)
                        (default: True)
  --skip-low-contrast   skip image chips with low contrast (default: False)
  --no-skip-low-contrast
                        do not skip image chips with low contrast (default:
                        True)
  --extent-no-border    do not include border in extent mask (default: False)
  --version             show program's version number and exit
  -v, --verbose         set loglevel to INFO (default: None)
  -vv, --very-verbose   set loglevel to DEBUG (default: None)
```

[ ]:

1.4.2 Generating masks

[ ]:

1.4.3 Post-processing segmentation results

[ ]:

1.5 satproc

1.5.1 satproc package

Subpackages

satproc.console namespace

Submodules

satproc.console.extract_chips module

This is a skeleton file that can serve as a starting point for a Python console script. To run this script uncomment the following lines in the [options.entry_points] section in setup.cfg:

console_scripts = fibonacci = satproc.skeleton:run

Then run *python setup.py install* which will install the command *fibonacci* inside your current environment. Besides console scripts, the header (i.e. until _logger...) of this file can also be used as template for Python modules.

Note: This skeleton file can be safely removed if not needed!

`satproc.console.extract_chips.`**`main`**`(`*args*`)`
Main entry point allowing external calls

Parameters **`args`** (*[str]*) – command line parameter list

`satproc.console.extract_chips.`**`parse_args`**`(`*args*`)`
Parse command line parameters

Parameters **`args`** (*[str]*) – command line parameters as list of strings

Returns command line parameters namespace

Return type `argparse.Namespace`

`satproc.console.extract_chips.`**`run`**`()`
Entry point for console_scripts

`satproc.console.extract_chips.`**`setup_logging`**`(`*loglevel*`)`
Setup basic logging

Parameters **loglevel** (*int*) – minimum loglevel for emitting messages

satproc.console.filter module

This is a skeleton file that can serve as a starting point for a Python console script. To run this script uncomment the following lines in the [options.entry_points] section in setup.cfg:

console_scripts = fibonacci = satproc.skeleton:run

Then run *python setup.py install* which will install the command *fibonacci* inside your current environment. Besides console scripts, the header (i.e. until _logger...) of this file can also be used as template for Python modules.

Note: This skeleton file can be safely removed if not needed!

satproc.console.filter.**main**(*args*)
Main entry point allowing external calls

Parameters **args** (*[str]*) – command line parameter list

satproc.console.filter.**parse_args**(*args*)
Parse command line parameters

Parameters **args** (*[str]*) – command line parameters as list of strings

Returns command line parameters namespace

Return type argparse.Namespace

satproc.console.filter.**run**()
Entry point for console_scripts

satproc.console.filter.**setup_logging**(*loglevel*)
Setup basic logging

Parameters **loglevel** (*int*) – minimum loglevel for emitting messages

satproc.console.generalize module

satproc.console.generalize.**main**(*args*)
Main entry point allowing external calls

Parameters **args** (*[str]*) – command line parameter list

satproc.console.generalize.**parse_args**(*args*)
Parse command line parameters

Parameters **args** (*[str]*) – command line parameters as list of strings

Returns command line parameters namespace

Return type argparse.Namespace

satproc.console.generalize.**run**()
Entry point for console_scripts

satproc.console.generalize.**setup_logging**(*loglevel*)
Setup basic logging

Parameters **loglevel** (*int*) – minimum loglevel for emitting messages

satproc.console.make_masks module

This is a skeleton file that can serve as a starting point for a Python console script. To run this script uncomment the following lines in the [options.entry_points] section in setup.cfg:

console_scripts = fibonacci = satproc.skeleton:run

Then run *python setup.py install* which will install the command *fibonacci* inside your current environment. Besides console scripts, the header (i.e. until _logger...) of this file can also be used as template for Python modules.

Note: This skeleton file can be safely removed if not needed!

satproc.console.make_masks.**main**(*args*)
Main entry point allowing external calls

Parameters **args** ([*str*]) – command line parameter list

satproc.console.make_masks.**parse_args**(*args*)
Parse command line parameters

Parameters **args** ([*str*]) – command line parameters as list of strings

Returns command line parameters namespace

Return type argparse.Namespace

satproc.console.make_masks.**run**()
Entry point for console_scripts

satproc.console.make_masks.**setup_logging**(*loglevel*)
Setup basic logging

Parameters **loglevel** (*int*) – minimum loglevel for emitting messages

satproc.console.match_histograms module

Apply histogram matching between two rasters.

It manipulates the pixels of an input image so that its histogram matches the histogram of the reference image. The matching is done independently for each band, as long as the number of bands is equal in the input image and the reference.

satproc.console.match_histograms.**main**(*args*)
Main entry point allowing external calls

Parameters **args** ([*str*]) – command line parameter list

satproc.console.match_histograms.**parse_args**(*args*)
Parse command line parameters

Parameters **args** ([*str*]) – command line parameters as list of strings

Returns command line parameters namespace

Return type argparse.Namespace

satproc.console.match_histograms.**run**()
Entry point for console_scripts

satproc.console.match_histograms.**setup_logging**(*loglevel*)
Setup basic logging

Parameters **loglevel** (*int*) – minimum loglevel for emitting messages

satproc.console.polygonize module

satproc.console.polygonize.**main**(*args*)
Main entry point allowing external calls

Parameters **args** ([*str*]) – command line parameter list

satproc.console.polygonize.**parse_args**(*args*)
Parse command line parameters

Parameters **args** ([*str*]) – command line parameters as list of strings

Returns command line parameters namespace

Return type argparse.Namespace

satproc.console.polygonize.**run**()
Entry point for console_scripts

satproc.console.polygonize.**setup_logging**(*loglevel*)
Setup basic logging

Parameters **loglevel** (*int*) – minimum loglevel for emitting messages

satproc.console.scale module

This is a skeleton file that can serve as a starting point for a Python console script. To run this script uncomment the following lines in the [options.entry_points] section in setup.cfg:

console_scripts = fibonacci = satproc.skeleton:run

Then run *python setup.py install* which will install the command *fibonacci* inside your current environment. Besides console scripts, the header (i.e. until _logger...) of this file can also be used as template for Python modules.

Note: This skeleton file can be safely removed if not needed!

satproc.console.scale.**main**(*args*)
Main entry point allowing external calls

Parameters **args** ([*str*]) – command line parameter list

satproc.console.scale.**parse_args**(*args*)
Parse command line parameters

Parameters **args** ([*str*]) – command line parameters as list of strings

Returns command line parameters namespace

Return type argparse.Namespace

satproc.console.scale.**run**()
Entry point for console_scripts

satproc.console.scale.**setup_logging**(*loglevel*)
Setup basic logging

Parameters **loglevel** (*int*) – minimum loglevel for emitting messages

satproc.console.smooth_stitch module

satproc.console.smooth_stitch.**main**(*args*)
Main entry point allowing external calls

Parameters **args** ([*str*]) – command line parameter list

satproc.console.smooth_stitch.**parse_args**(*args*)
Parse command line parameters

Parameters **args** ([*str*]) – command line parameters as list of strings

Returns command line parameters namespace

Return type argparse.Namespace

satproc.console.smooth_stitch.**run**()
Entry point for console_scripts

satproc.console.smooth_stitch.**setup_logging**(*loglevel*)
Setup basic logging

Parameters **loglevel** (*int*) – minimum loglevel for emitting messages

satproc.console.spatial_filter module

satproc.console.spatial_filter.**check_kernel_size**(*value*)

satproc.console.spatial_filter.**main**(*args*)
Main entry point allowing external calls

Parameters **args** ([*str*]) – command line parameter list

satproc.console.spatial_filter.**parse_args**(*args*)
Parse command line parameters

Parameters **args** ([*str*]) – command line parameters as list of strings

Returns command line parameters namespace

Return type argparse.Namespace

satproc.console.spatial_filter.**run**()
Entry point for console_scripts

satproc.console.spatial_filter.**setup_logging**(*loglevel*)
Setup basic logging

Parameters **loglevel** (*int*) – minimum loglevel for emitting messages

satproc.postprocess namespace

Submodules

satproc.postprocess.generalize module

satproc.postprocess.generalize.**generalize**(**, input_files, output_dir, target_crs=None, simplify='douglas', douglas_tolerance=0.1, smooth=None, chaikins_refinements=5*)

satproc.postprocess.generalize.**smooth_chaikin**(*shp*, *refinements=5*)

satproc.postprocess.polygonize module

satproc.postprocess.polygonize.**apply_threshold**(*src*, *dst*, *value=None*, ***, *threshold*)
Output source values (probabilities) instead of simply a binary mask

Make sure nodata=0, so that gdal_polygonize step ignores pixels under threshold.

Parameters

- **src** (*str*) – path to input raster
- **dst** (*str*) – path to output raster
- **threshold** (*int*) – threshold value
- **value** (*int*) – value to use on raster output when values are over threshold if None, use the original value from src

satproc.postprocess.polygonize.**gdal_polygonize**(*src*, *dst*)

satproc.postprocess.polygonize.**merge_vector_files**(***, *input_dir*, *output*, *tmpdir*)

satproc.postprocess.polygonize.**polygonize**(*threshold=None*, *value=None*, *temp_dir=None*, *input_files=[]*, *input_dir=None*, *tile_size=None*, ***, *output*)

satproc.postprocess.polygonize.**process_image**(*image*, *value=None*, ***, *tmpdir*, *threshold*)

satproc.postprocess.polygonize.**retile**(*raster*, *output_dir*, *tile_size*)

satproc.postprocess.polygonize.**retile_all**(*input_files*, *tile_size*, *temp_dir*)

satproc.postprocess.smooth module

satproc.postprocess.smooth.**build_bounds_index**(*image_files*)
Returns bounds of merged images and builds an R-Tree index

satproc.postprocess.smooth.**generate_spline_window_chips**(***, *image_paths*, *output_dir*, *power=2*)
Interpolates all images using a squared spline window

satproc.postprocess.smooth.**merge_chips**(*images_files*, ***, *win_bounds*)
Merge by taking mean between overlapping images

satproc.postprocess.smooth.**sliding_windows**(*size*, *whole=False*, *step_size=None*, ***, *width*, *height*)
Slide a window of +size+ by moving it +step_size+ pixels

satproc.postprocess.smooth.**smooth_stitch**(***, *input_dir*, *output_dir*, *power=1.5*, *temp_dir=None*)
Takes input directory of overlapping chips, and generates a new directory of non-overlapping chips with smooth edges.

satproc.postprocess.smooth.**spline_window**(*window_size*, *power=2*)
Squared spline window function: https://www.wolframalpha.com/input/?i=y%3Dx**2,+y%3D-(x-2)**2+%2B2,+y%3D(x-4)**2,+from+y+%3D+0+to+2

satproc.postprocess.smooth.**window_2D**(*size*, *power=2*, *n_channels=1*)

satproc.postprocess.spatial_filter module

satproc.postprocess.spatial_filter.**spatial_filter**(*, *input_paths*, *output_path*, *mode='gaussian'*, *size=5*, *merge=True*)

Submodules

satproc.chips module

satproc.chips.**extract_chips**(*rasters*, *aoi=None*, *labels=None*, *label_property='class'*, *masks={'extent'}*, *mask_type='class'*, *extent_no_border=False*, *rescale_mode=None*, *rescale_range=None*, *bands=None*, *chip_type='tif'*, *write_footprints=False*, *classes=None*, *crs=None*, *skip_existing=True*, *within=False*, *windows_mode='whole_overlap'*, *skip_low_contrast=False*, *skip_with_empty_mask=True*, ***, *size*, *step_size*, *output_dir*)

satproc.chips.**extract_chips_from_raster**(*raster*, *rescale_mode=None*, *rescale_range=None*, *bands=None*, *chip_type='tif'*, *write_footprints=False*, *labels=None*, *label_property='class'*, *mask_type='class'*, *masks={'extent'}*, *classes=None*, *crs=None*, *skip_existing=True*, *within=False*, *aoi_poly=None*, *polys_dict=None*, *windows_mode='whole_overlap'*, *skip_low_contrast=False*, *skip_with_empty_mask=True*, *extent_no_border=False*, ***, *size*, *step_size*, *output_dir*)

satproc.chips.**get_shape**(*feature*)

Get shape geometry from feature

Parameters **feature** (*dict*) – Feature as read from Fiona

Return type shapely.geometry.BaseGeometry

satproc.chips.**prepare_aoi_shape**(*aoi*)

satproc.chips.**write_image**(*img*, *path*, ***, *percentiles=None*, *skip_low_contrast=False*)

satproc.chips.**write_tif**(*img*, *path*, ***, *skip_low_contrast=False*, *window*, *meta*, *transform*, *bands*)

satproc.filter module

satproc.filter.**filter_by_max_prob**(*input_dir*, *output_dir*, *threshold*)

satproc.filter.**filter_chip**(*src*, ***, *threshold*, *output_dir*)

satproc.filter.**get_max_prob**(*p*)

satproc.histogram module

satproc.histogram.**match_histograms**(*src_path*, *dst_path*, *size=128*, *step_size=128*, ***, *reference_path*)
Match histograms of an image using another one as reference

Parameters

- **src_path** (*str*) – path to input raster
- **dst_path** (*str*) – path to output raster
- **size** (*int*) – size of windows
- **step_size** (*int*) – step size, when sliding windows
- **reference_path** (*str*) – path to the reference raster

Return type None

satproc.histogram.**read_window**(*ds*, *window*)
Read from a rasterio dataset using a window

NaNs are coerced to zero.

Parameters

- **ds** (*rasterio.Dataset*) – input dataset
- **window** (*rasterio.windows.Window*) – window to read from

Returns image data on window

Return type numpy.ndarray

satproc.histogram.**write_window**(*img*, *ds*, *window*)
Write array to raster on window

Parameters

- **img** (*numpy.ndarray*) – image array
- **ds** (*rasterio.Dataset*) – dataset to write to (must be opened with write access)
- **window** (*rasterio.windows.Window*) – window to write to

Return type None

satproc.masks module

satproc.masks.**all_masks_empty**(*masks*)
Check if all masks are empty

satproc.masks.**classify_polygons**(*labels*, *label_property*, *classes*)

satproc.masks.**make_masks**(*rasters*, ***, *output_dir*, *labels*, *label_property='class'*, *classes=None*, *mask_type='class'*, *masks={'extent'}*, *extent_no_border=False*)

satproc.masks.**mask_from_polygons**(*polygons*, ***, *win*, *t*, *extent_no_border=True*, *boundary_mask=None*, *distance_mask=None*)
Generate a binary mask array from a set of polygon

It can also generate a distance transform mask

Parameters

- **polygons** (*List[Union[Polygon, MultiPolygon]]*) – list of polygon or multipolygon geometries
- **win** (*rasterio.windows.Window*) – window
- **t** (*rasterio.transform.Affine*) – affine transform
- **extent_no_border** (*bool*) – if True, the extent mask will not include the border of the polygon
- **boundary_mask** (*bool*) – whether to generate boundary (edges) mask
- **distance_mask** (*bool*) – whether to generate a distance mask

Return type numpy.ndarray

satproc.masks.**multiband_chip_mask_by_classes**(*classes*, *transform*, *window*, *label_property*, *polys_dict=None*, *label_path=None*, *extent_mask_path=None*, *boundary_mask_path=None*, *distance_mask_path=None*, *extent_no_border=False*)

satproc.masks.**prepare_label_shapes**(*labels*, *mask_type='class'*, *label_property='class'*, *classes=None*)

satproc.masks.**write_window_masks**(*masks*, *, *window*, *metadata*, *transform*)
Write window masks to files

satproc.scale module

satproc.scale.**get_min_max**(*img*, *window_size=512*)
Return minimum and maximum values on array, in blocks

Parameters

- **img** (*numpy.ndarray*) – image array
- **window_size** (*int*) – size of window (default: 512)

Returns minimum and maximum values

Return type Tuple[float, float]

satproc.scale.**minmax_scale**(*img*, *, *min_values*, *max_values*)
Scale bands of image separately, to range 0..1

Parameters

- **img** (*numpy.ndarray*) – image array
- **min_values** (*List[float]*) – minimum values for each band
- **max_values** (*List[float]*) – maximum values for each band

Returns rescaled image

Return type numpy.ndarray

satproc.scale.**scale**(*input_img*, *output_img*, *window_size=512*)
Read a raster, rescale each band with min-max values, and save as another raster

Parameters

- **input_img** (*str*) – path to input image
- **output_img** (*str*) – path to output image
- **window_size** (*int*) – size of window

Return type None

satproc.utils module

satproc.utils.**build_virtual_raster**(*image_paths*, *output_path*, *separate=None*, *band=None*)

satproc.utils.**fiona_crs_from_proj_crs**(*proj_crs*)

satproc.utils.**get_raster_band_count**(*path*)
Get raster band count

Parameters **path** (*str*) – path of the raster image

Returns band count

Return type int

satproc.utils.**grouper**(*iterable*, *n*, *fillvalue=None*)
Collect data into fixed-length chunks or blocks

satproc.utils.**map_with_threads**(*items*, *worker*, *num_jobs=None*, *total=None*, *desc=None*)
Map a worker function to an iterable of items, using a thread pool

Parameters

- **items** (*iterable*) – items to map
- **worker** (*Function*) – worker function to apply to each item
- **num_jobs** (*int*) – number of threads to use
- **total** (*int (optional)*) – total number of items (for the progress bar)
- **desc** (*str (optional)*) – description of the task (for the progress bar)

Return type None

satproc.utils.**proj_crs_from_fiona_dataset**(*fio_ds*)

satproc.utils.**reproject_shape**(*shp*, *from_crs*, *to_crs*, *project=None*)
Reproject a shape from *from_crs* to *to_crs*

Parameters

- **shp** (*Shape*) – shape to reproject
- **from_crs** (*str*) – CRS epsg code of shape geometry
- **to_crs** (*str*) – CRS epsg code of reprojected shape geometry
- **project** (*Optional[str]*) – a Transformer instance to use for reprojection

Returns reprojected shape

Return type Shape

satproc.utils.**rescale_intensity**(*image*, *rescale_mode*, *rescale_range*)
Calculate percentiles from a range cut and rescale intensity of image to byte range

Parameters

- **image** (*numpy.ndarray*) – image array
- **rescale_mode** (*str*) – rescaling mode, either 'percentiles' or 'values'
- **rescale_range** (*Tuple[number, number]*) – input range for rescaling

Returns rescaled image

Return type numpy.ndarray

satproc.utils.**run_command**(*cmd*, *quiet=True*, ***, *cwd=None*)
Run a shell command

Parameters

- **cmd** (*str*) – command to run
- **quiet** (*bool (default: True)*) – silent output (stdout and sterr)

Return type None

satproc.utils.**sliding_windows**(*size*, *step_size*, *width*, *height*, *mode='exact'*)
Slide a window of +size+ by moving it +step_size+ pixels

Parameters

- **size** (*int*) – window size, in pixels
- **step_size** (*int*) – step or *stride* size when sliding window, in pixels
- **width** (*int*) – image width
- **height** (*int*) – image height
- **mode** (*str (default: 'exact')*) – either one of 'exact', 'whole', 'whole_overlap'. - 'exact': clip windows at borders, if needed - 'whole': only whole windows - 'whole_overlap': only wohle windows, allow overlapping windows at borders.

Yields *Tuple[Window, Tuple[int, int]]* – a pair of Window and a pair of position (i, j)

satproc.utils.**write_chips_geojson**(*output_path*, *chip_pairs*, ***, *chip_type*, *crs*, *basename*)
Write a GeoJSON containing chips polygons as features

Parameters

- **output_path** (*str*) – GeoJSON output path
- **chip_pairs** (*Tuple[Shape, Tuple[int, int, int]]*) – a pair with the chip polygon geometry, and a tuple of (feature id, x, y)
- **chip_type** (*str*) – chip file type extension (e.g. tif, jpg)
- **crs** (*str*) – CRS epsg code of chip polygon geometry
- **basename** (*str*) – basename of chip files

Return type None

Module contents

1.6 Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

1.6.1 [Unreleased]

Added

- `extract_chips`, `make_masks`: Add `--skip-with-empty-mask` and `--no-skip-with-empty-mask` options to skip chips with empty masks.
- `smooth_stitch`: Add `--power/-p` option to specify spline exponent
- `smooth_stitch`: Add `--temp-dir` to specify temporary directory
- New `spatial_filter` script to apply image filters (median filter and gaussian blur) (#43)
- `extract_chips`, `make_masks`: Add `--mask-type single` option (#44)

Changed

- Do not write image and masks if all masks are empty by default (#29)
- Try to improve smooth stitching process by doing a max-based merge and some other fixes (#42)

Fixed

- If `--step-size` is not specified, it should use the same `--size` (no overlapping) (#45)
- Update `rasterio` to 1.3b1 to fix issue with Python 3.10 (#47)

1.6.2 [0.1.9] - 2022-01-12

Changed

- Lowered Python requirement to 3.7+

1.6.3 [0.1.8] - 2022-12-28

Added

- Define `satproc_make_masks` console script for only generating masks of images (#15)
- Define `satproc_generalize` console script for simplifying and smoothing polygons (#22)
- Add new sliding window modes: `exact`, `whole`, and `whole_overlap`
- Add `--skip-low-contrast` to skip low contrast images on `extract_chips` (#10)
- Add `--masks` option to generate `extent`, `boundary` and `distance` masks on `extract_chips`
- Add `--extent-no-border` option on `make_masks` and `extract_chips` for removing polygon boundary from extent mask. (#21)

Changed

- Use `whole_overlap` when extracting chips by default
- Dissolve adjacent polygons from windows after merging groups (#19)
- Rename `--write-geojson` option to `--write-footprints`
- Do not skip low contrast images by default (#10)
- Bugfix when using `--rescale` with values mode
- Use tqdm in ASCII mode to be friendlier on log files
- Pin pyproj>3 version

Removed

- `build_dataset` module and CLI script

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

P

R

S

W