dask-ndmorph Documentation

Release 0.1.1+0.gb226040.dirty

John Kirkham

Jun 06, 2017

Contents

Contents:

CHAPTER 1

dask-ndmorph

A library for using N-D filters with Dask Arrays

- Free software: BSD 3-Clause
- Documentation: https://dask-ndmorph.readthedocs.io.

Features

- TODO

Credits

This package was created with Cookiecutter and the dask-image/dask-image-cookiecutter project template.

CHAPTER 2

Installation

Stable release

To install dask-ndmorph, run this command in your terminal:

```
$ pip install dask-ndmorph
```

This is the preferred method to install dask-ndmorph, as it will always install the most recent stable release.

If you don’t have pip installed, this Python installation guide can guide you through the process.

From sources

The sources for dask-ndmorph can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/dask-image/dask-ndmorph
```

Or download the tarball:

```
$ curl  -OL https://github.com/dask-image/dask-ndmorph/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use dask-ndmorph in a project:

```python
import dask_ndmorph
```

CHAPTER 4

API

dask_ndmorph package

dask_ndmorph.**binary_closing**(*input*, *structure=None*, *iterations=1*, *origin=0*)

Wrapped copy of "scipy.ndimage.morphology.binary_closing"

Excludes the output parameter as it would not work with Dask arrays.

Original docstring:

Multi-dimensional binary closing with the given structuring element.

The *closing* of an input image by a structuring element is the *erosion* of the *dilation* of the image by the structuring element.

Parameters

- **input** (*array_like*) – Binary array_like to be closed. Non-zero (True) elements form the subset to be closed.
- **structure** (*array_like, optional*) – Structuring element used for the closing. Non-zero elements are considered True. If no structuring element is provided an element is generated with a square connectivity equal to one (i.e., only nearest neighbors are connected to the center, diagonally-connected elements are not considered neighbors).
- **iterations** (*{int, float}, optional*) – The dilation step of the closing, then the erosion step are each repeated *iterations* times (one, by default). If iterations is less than 1, each operations is repeated until the result does not change anymore.
- **origin** (*int or tuple of ints, optional*) – Placement of the filter, by default 0.

Returns **binary_closing** – Closing of the input by the structuring element.

Return type ndarray of bools

See also:

grey_closing(), *binary_opening()*, *binary_dilation()*, *binary_erosion()*, generate_binary_structure()

Notes

Closing [1]_ is a mathematical morphology operation [2]_ that consists in the succession of a dilation and an erosion of the input with the same structuring element. Closing therefore fills holes smaller than the structuring element.

Together with *opening* (*binary_opening*), closing can be used for noise removal.

References

Examples

```
>>> from scipy import ndimage
>>> a = np.zeros((5,5), dtype=int)
>>> a[1:-1, 1:-1] = 1; a[2,2] = 0
>>> a
array([[0, 0, 0, 0, 0],
       [0, 1, 1, 1, 0],
       [0, 1, 0, 1, 0],
       [0, 1, 1, 1, 0],
       [0, 0, 0, 0, 0]])
>>> # Closing removes small holes
>>> ndimage.binary_closing(a).astype(int)
array([[0, 0, 0, 0, 0],
       [0, 1, 1, 1, 0],
       [0, 1, 1, 1, 0],
       [0, 1, 1, 1, 0],
       [0, 0, 0, 0, 0]])
>>> # Closing is the erosion of the dilation of the input
>>> ndimage.binary_dilation(a).astype(int)
array([[0, 1, 1, 1, 0],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [0, 1, 1, 1, 0]])
>>> ndimage.binary_erosion(ndimage.binary_dilation(a)).astype(int)
array([[0, 0, 0, 0, 0],
       [0, 1, 1, 1, 0],
       [0, 1, 1, 1, 0],
       [0, 1, 1, 1, 0],
       [0, 0, 0, 0, 0]])
```

```
>>> a = np.zeros((7,7), dtype=int)
>>> a[1:6, 2:5] = 1; a[1:3,3] = 0
>>> a
array([[0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 1, 0, 0],
       [0, 0, 1, 0, 1, 0, 0],
       [0, 0, 1, 1, 1, 0, 0],
       [0, 0, 1, 1, 1, 0, 0],
       [0, 0, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0]])
```

```
>>> # In addition to removing holes, closing can also
>>> # coarsen boundaries with fine hollows.
>>> ndimage.binary_closing(a).astype(int)
array([[0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 1, 0, 0],
       [0, 0, 1, 1, 1, 0, 0],
       [0, 0, 1, 1, 1, 0, 0],
       [0, 0, 1, 1, 1, 0, 0],
       [0, 0, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0]])
>>> ndimage.binary_closing(a, structure=np.ones((2,2))).astype(int)
array([[0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 1, 1, 0, 0],
       [0, 0, 1, 1, 1, 0, 0],
       [0, 0, 1, 1, 1, 0, 0],
       [0, 0, 1, 1, 1, 0, 0],
       [0, 0, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0]])
```

`dask_ndmorph.`**`binary_dilation`**(*input*, *structure=None*, *iterations=1*, *mask=None*, *border_value=0*, *origin=0*, *brute_force=False*)

Wrapped copy of "scipy.ndimage.morphology.binary_dilation"

Excludes the output parameter as it would not work with Dask arrays.

Original docstring:

Multi-dimensional binary dilation with the given structuring element.

Parameters

- **input** (*array_like*) – Binary array_like to be dilated. Non-zero (True) elements form the subset to be dilated.
- **structure** (*array_like, optional*) – Structuring element used for the dilation. Non-zero elements are considered True. If no structuring element is provided an element is generated with a square connectivity equal to one.
- **iterations** (*{int, float}, optional*) – The dilation is repeated *iterations* times (one, by default). If iterations is less than 1, the dilation is repeated until the result does not change anymore.
- **mask** (*array_like, optional*) – If a mask is given, only those elements with a True value at the corresponding mask element are modified at each iteration.
- **origin** (*int or tuple of ints, optional*) – Placement of the filter, by default 0.
- **border_value** (*int (cast to 0 or 1), optional*) – Value at the border in the output array.

Returns **binary_dilation** – Dilation of the input by the structuring element.

Return type ndarray of bools

See also:

`grey_dilation()`, *`binary_erosion()`*, *`binary_closing()`*, *`binary_opening()`*, `generate_binary_structure()`

Notes

Dilation [1]_ is a mathematical morphology operation [2]_ that uses a structuring element for expanding the shapes in an image. The binary dilation of an image by a structuring element is the locus of the points covered by the structuring element, when its center lies within the non-zero points of the image.

References

Examples

```
>>> from scipy import ndimage
>>> a = np.zeros((5, 5))
>>> a[2, 2] = 1
>>> a
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.]])
>>> ndimage.binary_dilation(a)
array([[False, False, False, False, False],
       [False, False,  True, False, False],
       [False,  True,  True,  True, False],
       [False, False,  True, False, False],
       [False, False, False, False, False]], dtype=bool)
>>> ndimage.binary_dilation(a).astype(a.dtype)
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  1.,  1.,  1.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.]])
>>> # 3x3 structuring element with connectivity 1, used by default
>>> struct1 = ndimage.generate_binary_structure(2, 1)
>>> struct1
array([[False,  True, False],
       [ True,  True,  True],
       [False,  True, False]], dtype=bool)
>>> # 3x3 structuring element with connectivity 2
>>> struct2 = ndimage.generate_binary_structure(2, 2)
>>> struct2
array([[ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True]], dtype=bool)
>>> ndimage.binary_dilation(a, structure=struct1).astype(a.dtype)
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  1.,  1.,  1.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.]])
>>> ndimage.binary_dilation(a, structure=struct2).astype(a.dtype)
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  1.,  1.,  0.],
       [ 0.,  1.,  1.,  1.,  0.],
       [ 0.,  1.,  1.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  0.]])
>>> ndimage.binary_dilation(a, structure=struct1,\
... iterations=2).astype(a.dtype)
array([[ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  1.,  1.,  1.,  0.],
       [ 1.,  1.,  1.,  1.,  1.],
       [ 0.,  1.,  1.,  1.,  0.],
       [ 0.,  0.,  1.,  0.,  0.]])
```

dask_ndmorph.**binary_erosion**(*input*, *structure=None*, *iterations=1*, *mask=None*, *border_value=0*, *origin=0*, *brute_force=False*)

Wrapped copy of "scipy.ndimage.morphology.binary_erosion"

Excludes the output parameter as it would not work with Dask arrays.

Original docstring:

Multi-dimensional binary erosion with a given structuring element.

Binary erosion is a mathematical morphology operation used for image processing.

Parameters

- **input** (*array_like*) – Binary image to be eroded. Non-zero (True) elements form the subset to be eroded.
- **structure** (*array_like, optional*) – Structuring element used for the erosion. Non-zero elements are considered True. If no structuring element is provided, an element is generated with a square connectivity equal to one.
- **iterations** (*{int, float}, optional*) – The erosion is repeated *iterations* times (one, by default). If iterations is less than 1, the erosion is repeated until the result does not change anymore.
- **mask** (*array_like, optional*) – If a mask is given, only those elements with a True value at the corresponding mask element are modified at each iteration.
- **origin** (*int or tuple of ints, optional*) – Placement of the filter, by default 0.
- **border_value** (*int (cast to 0 or 1), optional*) – Value at the border in the output array.

Returns **binary_erosion** – Erosion of the input by the structuring element.

Return type ndarray of bools

See also:

grey_erosion(), *binary_dilation()*, *binary_closing()*, *binary_opening()*, generate_binary_structure()

Notes

Erosion [1]_ is a mathematical morphology operation [2]_ that uses a structuring element for shrinking the shapes in an image. The binary erosion of an image by a structuring element is the locus of the points where a superimposition of the structuring element centered on the point is entirely contained in the set of non-zero elements of the image.

References

Examples

```
>>> from scipy import ndimage
>>> a = np.zeros((7,7), dtype=int)
>>> a[1:6, 2:5] = 1
>>> a
array([[0, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 1, 1, 0, 0],
       [0, 0, 1, 1, 1, 0, 0],
       [0, 0, 1, 1, 1, 0, 0],
       [0, 0, 1, 1, 1, 0, 0],
       [0, 0, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 0]])
>>> ndimage.binary_erosion(a).astype(a.dtype)
array([[0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0]])
>>> #Erosion removes objects smaller than the structure
>>> ndimage.binary_erosion(a, structure=np.ones((5,5))).astype(a.dtype)
array([[0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0]])
```

`dask_ndmorph.`**`binary_opening`**(*input*, *structure=None*, *iterations=1*, *origin=0*)

Wrapped copy of "scipy.ndimage.morphology.binary_opening"

Excludes the output parameter as it would not work with Dask arrays.

Original docstring:

Multi-dimensional binary opening with the given structuring element.

The *opening* of an input image by a structuring element is the *dilation* of the *erosion* of the image by the structuring element.

Parameters

- **input** (*array_like*) – Binary array_like to be opened. Non-zero (True) elements form the subset to be opened.
- **structure** (*array_like, optional*) – Structuring element used for the opening. Non-zero elements are considered True. If no structuring element is provided an element is generated with a square connectivity equal to one (i.e., only nearest neighbors are connected to the center, diagonally-connected elements are not considered neighbors).
- **iterations** (*{int, float}, optional*) – The erosion step of the opening, then the dilation step are each repeated *iterations* times (one, by default). If *iterations* is less than 1, each operation is repeated until the result does not change anymore.
- **origin** (*int or tuple of ints, optional*) – Placement of the filter, by default 0.

Returns **binary_opening** – Opening of the input by the structuring element.

Return type ndarray of bools

See also:

grey_opening(), *binary_closing()*, *binary_erosion()*, *binary_dilation()*, generate_binary_structure()

Notes

Opening [1]_ is a mathematical morphology operation [2]_ that consists in the succession of an erosion and a dilation of the input with the same structuring element. Opening therefore removes objects smaller than the structuring element.

Together with *closing* (*binary_closing*), opening can be used for noise removal.

References

Examples

```
>>> from scipy import ndimage
>>> a = np.zeros((5,5), dtype=int)
>>> a[1:4, 1:4] = 1; a[4, 4] = 1
>>> a
array([[0, 0, 0, 0, 0],
       [0, 1, 1, 1, 0],
       [0, 1, 1, 1, 0],
       [0, 1, 1, 1, 0],
       [0, 0, 0, 0, 1]])
>>> # Opening removes small objects
>>> ndimage.binary_opening(a, structure=np.ones((3,3))).astype(int)
array([[0, 0, 0, 0, 0],
       [0, 1, 1, 1, 0],
       [0, 1, 1, 1, 0],
       [0, 1, 1, 1, 0],
       [0, 0, 0, 0, 0]])
>>> # Opening can also smooth corners
>>> ndimage.binary_opening(a).astype(int)
array([[0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0],
       [0, 1, 1, 1, 0],
       [0, 0, 1, 0, 0],
       [0, 0, 0, 0, 0]])
>>> # Opening is the dilation of the erosion of the input
>>> ndimage.binary_erosion(a).astype(int)
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]])
>>> ndimage.binary_dilation(ndimage.binary_erosion(a)).astype(int)
array([[0, 0, 0, 0, 0],
       [0, 0, 1, 0, 0],
       [0, 1, 1, 1, 0],
       [0, 0, 1, 0, 0],
       [0, 0, 0, 0, 0]])
```

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/dask-image/dask-ndmorph/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

Write Documentation

dask-ndmorph could always use more documentation, whether as part of the official dask-ndmorph docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/dask-image/dask-ndmorph/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *dask-ndmorph* for local development.

1. Fork the *dask-ndmorph* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/dask-ndmorph.git
```

3. Install your local copy into an environment. Assuming you have conda installed, this is how you set up your fork for local development (on Windows drop *source*). Replace *"<some version>"* with the Python version used for testing.:

```
$ conda create -n dask-ndmorphenv python="<some version>"
$ source activate dask-ndmorphenv
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions:

```
$ flake8 dask_ndmorph tests
$ python setup.py test or py.test
```

   To get flake8, just conda install it into your environment.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5, and 3.6. Check https://travis-ci.org/dask-image/dask-ndmorph/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ py.test tests/test_dask_ndmorph.py
```

CHAPTER 6

Credits

Development Lead

- John Kirkham, Howard Hughes Medical Institute <kirkhamj@janelia.hhmi.org>

Contributors

None yet. Why not be the first?

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index