dask-ndfilters Documentation

Release 0.1.1+0.geec3d0c.dirty

John Kirkham

May 24, 2017

Contents

1 dask-ndfilters 3

2 Installation 5

3 Usage 7

4 API 9

5 Contributing 21

6 Credits 25

7 Indices and tables 27

Python Module Index 29

Contents:

CHAPTER 1

dask-ndfilters

A library for using N-D filters with Dask Arrays

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

Features

- TODO

Credits

This package was created with Cookiecutter and the nanshe-org/nanshe-cookiecutter project template.

CHAPTER 2

Installation

Stable release

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

```
$ pip install dask-ndfilters
```

This is the preferred method to install dask-ndfilters, 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-ndfilters can be downloaded from the Github repo.

You can either clone the public repository:

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

Or download the tarball:

```
$ curl  -OL https://github.com/dask-image/dask-ndfilters/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-ndfilters in a project:

```
import dask_ndfilters
```

CHAPTER 4

API

dask_ndfilters package

dask_ndfilters.**convolve**(*input*, *weights*, *mode='reflect'*, *cval=0.0*, *origin=0*)

Wrapped copy of "scipy.ndimage.filters.convolve"

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

Original docstring:

Multidimensional convolution.

The array is convolved with the given kernel.

Parameters

- **input** (*array_like*) – Input array to filter.
- **weights** (*array_like*) – Array of weights, same number of dimensions as input
- **mode** (*{'reflect','constant','nearest','mirror', 'wrap'}, optional*) – the *mode* parameter determines how the array borders are handled. For 'constant' mode, values beyond borders are set to be *cval*. Default is 'reflect'.
- **cval** (*scalar, optional*) – Value to fill past edges of input if *mode* is 'constant'. Default is 0.0
- **origin** (*array_like, optional*) – The *origin* parameter controls the placement of the filter, relative to the centre of the current element of the input. Default of 0 is equivalent to `(0,)*input.ndim`.

Returns **result** – The result of convolution of *input* with *weights*.

Return type ndarray

See also:

correlate() Correlate an image with a kernel.

Notes

Each value in result is , where W is the *weights* kernel, j is the n-D spatial index over , I is the *input* and k is the coordinate of the center of W, specified by *origin* in the input parameters.

Examples

Perhaps the simplest case to understand is `mode='constant', cval=0.0`, because in this case borders (i.e. where the *weights* kernel, centered on any one value, extends beyond an edge of *input*.

```
>>> a = np.array([[1, 2, 0, 0],
...               [5, 3, 0, 4],
...               [0, 0, 0, 7],
...               [9, 3, 0, 0]])
>>> k = np.array([[1,1,1],[1,1,0],[1,0,0]])
>>> from scipy import ndimage
>>> ndimage.convolve(a, k, mode='constant', cval=0.0)
array([[11, 10,  7,  4],
       [10,  3, 11, 11],
       [15, 12, 14,  7],
       [12,  3,  7,  0]])
```

Setting `cval=1.0` is equivalent to padding the outer edge of *input* with 1.0's (and then extracting only the original region of the result).

```
>>> ndimage.convolve(a, k, mode='constant', cval=1.0)
array([[13, 11,  8,  7],
       [11,  3, 11, 14],
       [16, 12, 14, 10],
       [15,  6, 10,  5]])
```

With `mode='reflect'` (the default), outer values are reflected at the edge of *input* to fill in missing values.

```
>>> b = np.array([[2, 0, 0],
...               [1, 0, 0],
...               [0, 0, 0]])
>>> k = np.array([[0,1,0], [0,1,0], [0,1,0]])
>>> ndimage.convolve(b, k, mode='reflect')
array([[5, 0, 0],
       [3, 0, 0],
       [1, 0, 0]])
```

This includes diagonally at the corners.

```
>>> k = np.array([[1,0,0],[0,1,0],[0,0,1]])
>>> ndimage.convolve(b, k)
array([[4, 2, 0],
       [3, 2, 0],
       [1, 1, 0]])
```

With `mode='nearest'`, the single nearest value in to an edge in *input* is repeated as many times as needed to match the overlapping *weights*.

```
>>> c = np.array([[2, 0, 1],
...               [1, 0, 0],
...               [0, 0, 0]])
```

```
>>> k = np.array([[0, 1, 0],
...               [0, 1, 0],
...               [0, 1, 0],
...               [0, 1, 0],
...               [0, 1, 0]])
>>> ndimage.convolve(c, k, mode='nearest')
array([[7, 0, 3],
       [5, 0, 2],
       [3, 0, 1]])
```

dask_ndfilters.**correlate**(*input*, *weights*, *mode='reflect'*, *cval=0.0*, *origin=0*)

Wrapped copy of "scipy.ndimage.filters.correlate"

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

Original docstring:

Multi-dimensional correlation.

The array is correlated with the given kernel.

Parameters

- **input** (*array-like*) – input array to filter
- **weights** (*ndarray*) – array of weights, same number of dimensions as input
- **mode** (*{'reflect','constant','nearest','mirror', 'wrap'}, optional*) – The `mode` parameter determines how the array borders are handled, where `cval` is the value when mode is equal to 'constant'. Default is 'reflect'
- **cval** (*scalar, optional*) – Value to fill past edges of input if `mode` is 'constant'. Default is 0.0
- **origin** (*scalar, optional*) – The `origin` parameter controls the placement of the filter. Default 0

See also:

convolve() Convolve an image with a kernel.

dask_ndfilters.**gaussian_filter**(*input*, *sigma*, *order=0*, *mode='reflect'*, *cval=0.0*, *truncate=4.0*)

Wrapped copy of "scipy.ndimage.filters.gaussian_filter"

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

Original docstring:

Multidimensional Gaussian filter.

Parameters

- **input** (*array_like*) – Input array to filter.
- **sigma** (*scalar or sequence of scalars*) – Standard deviation for Gaussian kernel. The standard deviations of the Gaussian filter are given for each axis as a sequence, or as a single number, in which case it is equal for all axes.
- **order** (*{0, 1, 2, 3} or sequence from same set, optional*) – The order of the filter along each axis is given as a sequence of integers, or as a single number. An order of 0 corresponds to convolution with a Gaussian kernel. An order of 1, 2, or 3 corresponds to convolution with the first, second or third derivatives of a Gaussian. Higher order derivatives are not implemented

- **mode** (*{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional*) – The *mode* parameter determines how the array borders are handled, where *cval* is the value when mode is equal to 'constant'. Default is 'reflect'
- **cval** (*scalar, optional*) – Value to fill past edges of input if *mode* is 'constant'. Default is 0.0
- **truncate** (*float*) – Truncate the filter at this many standard deviations. Default is 4.0.

Returns **gaussian_filter** – Returned array of same shape as *input*.

Return type ndarray

Notes

The multidimensional filter is implemented as a sequence of one-dimensional convolution filters. The intermediate arrays are stored in the same data type as the output. Therefore, for output types with a limited precision, the results may be imprecise because intermediate results may be stored with insufficient precision.

Examples

```
>>> from scipy.ndimage import gaussian_filter
>>> a = np.arange(50, step=2).reshape((5,5))
>>> a
array([[ 0,  2,  4,  6,  8],
       [10, 12, 14, 16, 18],
       [20, 22, 24, 26, 28],
       [30, 32, 34, 36, 38],
       [40, 42, 44, 46, 48]])
>>> gaussian_filter(a, sigma=1)
array([[ 4,  6,  8,  9, 11],
       [10, 12, 14, 15, 17],
       [20, 22, 24, 25, 27],
       [29, 31, 33, 34, 36],
       [35, 37, 39, 40, 42]])
```

dask_ndfilters.**gaussian_gradient_magnitude**(*input*, *sigma*, *mode='reflect'*, *cval=0.0*, *truncate=4.0*, ***kwargs*)

Wrapped copy of "scipy.ndimage.filters.gaussian_gradient_magnitude"

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

Original docstring:

Multidimensional gradient magnitude using Gaussian derivatives.

Parameters

- **input** (*array_like*) – Input array to filter.
- **sigma** (*scalar or sequence of scalars*) – The standard deviations of the Gaussian filter are given for each axis as a sequence, or as a single number, in which case it is equal for all axes..
- **mode** (*{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional*) – The *mode* parameter determines how the array borders are handled, where *cval* is the value when mode is equal to 'constant'. Default is 'reflect'

- **cval** (*scalar, optional*) – Value to fill past edges of input if *mode* is 'constant'. Default is 0.0
- **keyword arguments will be passed to gaussian_filter()** (*Extra*) –

dask_ndfilters.**gaussian_laplace**(*input*, *sigma*, *mode='reflect'*, *cval=0.0*, *truncate=4.0*, ***kwargs*)

Wrapped copy of "scipy.ndimage.filters.gaussian_laplace"

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

Original docstring:

Multidimensional Laplace filter using gaussian second derivatives.

Parameters

- **input** (*array_like*) – Input array to filter.
- **sigma** (*scalar or sequence of scalars*) – The standard deviations of the Gaussian filter are given for each axis as a sequence, or as a single number, in which case it is equal for all axes.
- **mode** (*{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional*) – The *mode* parameter determines how the array borders are handled, where *cval* is the value when mode is equal to 'constant'. Default is 'reflect'
- **cval** (*scalar, optional*) – Value to fill past edges of input if *mode* is 'constant'. Default is 0.0
- **keyword arguments will be passed to gaussian_filter()** (*Extra*) –

Examples

```
>>> from scipy import ndimage, misc
>>> import matplotlib.pyplot as plt
>>> ascent = misc.ascent()
```

```
>>> fig = plt.figure()
>>> plt.gray()  # show the filtered result in grayscale
>>> ax1 = fig.add_subplot(121)  # left side
>>> ax2 = fig.add_subplot(122)  # right side
```

```
>>> result = ndimage.gaussian_laplace(ascent, sigma=1)
>>> ax1.imshow(result)
```

```
>>> result = ndimage.gaussian_laplace(ascent, sigma=3)
>>> ax2.imshow(result)
>>> plt.show()
```

dask_ndfilters.**generic_filter**(*input*, *function*, *size=None*, *footprint=None*, *mode='reflect'*, *cval=0.0*, *origin=0*, *extra_arguments=()*, *extra_keywords={}*)

Wrapped copy of "scipy.ndimage.filters.generic_filter"

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

Original docstring:

Calculates a multi-dimensional filter using the given function.

At each element the provided function is called. The input values within the filter footprint at that element are passed to the function as a 1D array of double values.

Parameters

- **input** (*array_like*) – Input array to filter.
- **function** (*callable*) – Function to apply at each element.
- **size** (*scalar or tuple, optional*) – See footprint, below
- **footprint** (*array, optional*) – Either *size* or *footprint* must be defined. *size* gives the shape that is taken from the input array, at every element position, to define the input to the filter function. *footprint* is a boolean array that specifies (implicitly) a shape, but also which of the elements within this shape will get passed to the filter function. Thus `size=(n,m)` is equivalent to `footprint=np.ones((n,m))`. We adjust *size* to the number of dimensions of the input array, so that, if the input array is shape (10,10,10), and *size* is 2, then the actual size used is (2,2,2).
- **mode** (*{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional*) – The *mode* parameter determines how the array borders are handled, where *cval* is the value when mode is equal to 'constant'. Default is 'reflect'
- **cval** (*scalar, optional*) – Value to fill past edges of input if *mode* is 'constant'. Default is 0.0
- **origin** (*scalar, optional*) – The *origin* parameter controls the placement of the filter. Default 0.0.
- **extra_arguments** (*sequence, optional*) – Sequence of extra positional arguments to pass to passed function
- **extra_keywords** (*dict, optional*) – dict of extra keyword arguments to pass to passed function

dask_ndfilters.**laplace**(*input*, *mode='reflect'*, *cval=0.0*)

Wrapped copy of "scipy.ndimage.filters.laplace"

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

Original docstring:

N-dimensional Laplace filter based on approximate second derivatives.

Parameters

- **input** (*array_like*) – Input array to filter.
- **mode** (*{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional*) – The *mode* parameter determines how the array borders are handled, where *cval* is the value when mode is equal to 'constant'. Default is 'reflect'
- **cval** (*scalar, optional*) – Value to fill past edges of input if *mode* is 'constant'. Default is 0.0

Examples

```
>>> from scipy import ndimage, misc
>>> import matplotlib.pyplot as plt
>>> ascent = misc.ascent()
>>> result = ndimage.laplace(ascent)
```

```
>>> plt.gray()  # show the filtered result in grayscale
>>> plt.imshow(result)
```

dask_ndfilters.**maximum_filter**(*input*, *size=None*, *footprint=None*, *mode='reflect'*, *cval=0.0*, *origin=0*)

Wrapped copy of "scipy.ndimage.filters.maximum_filter"

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

Original docstring:

Calculates a multi-dimensional maximum filter.

Parameters

- **input** (*array_like*) – Input array to filter.
- **size** (*scalar or tuple, optional*) – See footprint, below
- **footprint** (*array, optional*) – Either *size* or *footprint* must be defined. *size* gives the shape that is taken from the input array, at every element position, to define the input to the filter function. *footprint* is a boolean array that specifies (implicitly) a shape, but also which of the elements within this shape will get passed to the filter function. Thus `size=(n,m)` is equivalent to `footprint=np.ones((n,m))`. We adjust *size* to the number of dimensions of the input array, so that, if the input array is shape (10,10,10), and *size* is 2, then the actual size used is (2,2,2).
- **mode** (*{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional*) – The *mode* parameter determines how the array borders are handled, where *cval* is the value when mode is equal to 'constant'. Default is 'reflect'
- **cval** (*scalar, optional*) – Value to fill past edges of input if *mode* is 'constant'. Default is 0.0
- **origin** (*scalar, optional*) – The *origin* parameter controls the placement of the filter. Default 0.0.

dask_ndfilters.**median_filter**(*input*, *size=None*, *footprint=None*, *mode='reflect'*, *cval=0.0*, *origin=0*)

Wrapped copy of "scipy.ndimage.filters.median_filter"

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

Original docstring:

Calculates a multidimensional median filter.

Parameters

- **input** (*array_like*) – Input array to filter.
- **size** (*scalar or tuple, optional*) – See footprint, below
- **footprint** (*array, optional*) – Either *size* or *footprint* must be defined. *size* gives the shape that is taken from the input array, at every element position, to define the input to the filter function. *footprint* is a boolean array that specifies (implicitly) a shape, but also which of the elements within this shape will get passed to the filter function. Thus `size=(n,m)` is equivalent to `footprint=np.ones((n,m))`. We adjust *size* to the number of dimensions of the input array, so that, if the input array is shape (10,10,10), and *size* is 2, then the actual size used is (2,2,2).

- **mode** (*{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional*) – The *mode* parameter determines how the array borders are handled, where *cval* is the value when mode is equal to 'constant'. Default is 'reflect'
- **cval** (*scalar, optional*) – Value to fill past edges of input if *mode* is 'constant'. Default is 0.0
- **origin** (*scalar, optional*) – The *origin* parameter controls the placement of the filter. Default 0.0.

Returns **median_filter** – Return of same shape as *input*.

Return type ndarray

dask_ndfilters.**minimum_filter**(*input*, *size=None*, *footprint=None*, *mode='reflect'*, *cval=0.0*, *origin=0*)

Wrapped copy of "scipy.ndimage.filters.minimum_filter"

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

Original docstring:

Calculates a multi-dimensional minimum filter.

Parameters

- **input** (*array_like*) – Input array to filter.
- **size** (*scalar or tuple, optional*) – See footprint, below
- **footprint** (*array, optional*) – Either *size* or *footprint* must be defined. *size* gives the shape that is taken from the input array, at every element position, to define the input to the filter function. *footprint* is a boolean array that specifies (implicitly) a shape, but also which of the elements within this shape will get passed to the filter function. Thus `size=(n,m)` is equivalent to `footprint=np.ones((n,m))`. We adjust *size* to the number of dimensions of the input array, so that, if the input array is shape (10,10,10), and *size* is 2, then the actual size used is (2,2,2).
- **mode** (*{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional*) – The *mode* parameter determines how the array borders are handled, where *cval* is the value when mode is equal to 'constant'. Default is 'reflect'
- **cval** (*scalar, optional*) – Value to fill past edges of input if *mode* is 'constant'. Default is 0.0
- **origin** (*scalar, optional*) – The *origin* parameter controls the placement of the filter. Default 0.0.

dask_ndfilters.**percentile_filter**(*input*, *percentile*, *size=None*, *footprint=None*, *mode='reflect'*, *cval=0.0*, *origin=0*)

Wrapped copy of "scipy.ndimage.filters.percentile_filter"

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

Original docstring:

Calculates a multi-dimensional percentile filter.

Parameters

- **input** (*array_like*) – Input array to filter.
- **percentile** (*scalar*) – The percentile parameter may be less then zero, i.e., percentile = -20 equals percentile = 80
- **size** (*scalar or tuple, optional*) – See footprint, below

- **footprint** (*array, optional*) – Either *size* or *footprint* must be defined. *size* gives the shape that is taken from the input array, at every element position, to define the input to the filter function. *footprint* is a boolean array that specifies (implicitly) a shape, but also which of the elements within this shape will get passed to the filter function. Thus `size=(n,m)` is equivalent to `footprint=np.ones((n,m))`. We adjust *size* to the number of dimensions of the input array, so that, if the input array is shape (10,10,10), and *size* is 2, then the actual size used is (2,2,2).
- **mode** (*{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional*) – The *mode* parameter determines how the array borders are handled, where *cval* is the value when mode is equal to 'constant'. Default is 'reflect'
- **cval** (*scalar, optional*) – Value to fill past edges of input if *mode* is 'constant'. Default is 0.0
- **origin** (*scalar, optional*) – The *origin* parameter controls the placement of the filter. Default 0.0.

dask_ndfilters.**prewitt**(*input*, *axis=-1*, *mode='reflect'*, *cval=0.0*)

Wrapped copy of "scipy.ndimage.filters.prewitt"

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

Original docstring:

Calculate a Prewitt filter.

Parameters

- **input** (*array_like*) – Input array to filter.
- **axis** (*int, optional*) – The axis of *input* along which to calculate. Default is -1.
- **mode** (*{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional*) – The *mode* parameter determines how the array borders are handled, where *cval* is the value when mode is equal to 'constant'. Default is 'reflect'
- **cval** (*scalar, optional*) – Value to fill past edges of input if *mode* is 'constant'. Default is 0.0

Examples

```
>>> from scipy import ndimage, misc
>>> import matplotlib.pyplot as plt
>>> ascent = misc.ascent()
>>> result = ndimage.prewitt(ascent)
>>> plt.gray()  # show the filtered result in grayscale
>>> plt.imshow(result)
```

dask_ndfilters.**rank_filter**(*input*, *rank*, *size=None*, *footprint=None*, *mode='reflect'*, *cval=0.0*, *origin=0*)

Wrapped copy of "scipy.ndimage.filters.rank_filter"

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

Original docstring:

Calculates a multi-dimensional rank filter.

Parameters

- **input** (*array_like*) – Input array to filter.

- **rank** (*int*) – The rank parameter may be less then zero, i.e., rank = -1 indicates the largest element.
- **size** (*scalar or tuple, optional*) – See footprint, below
- **footprint** (*array, optional*) – Either *size* or *footprint* must be defined. *size* gives the shape that is taken from the input array, at every element position, to define the input to the filter function. *footprint* is a boolean array that specifies (implicitly) a shape, but also which of the elements within this shape will get passed to the filter function. Thus `size=(n,m)` is equivalent to `footprint=np.ones((n,m))`. We adjust *size* to the number of dimensions of the input array, so that, if the input array is shape (10,10,10), and *size* is 2, then the actual size used is (2,2,2).
- **mode** (*{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional*) – The *mode* parameter determines how the array borders are handled, where *cval* is the value when mode is equal to 'constant'. Default is 'reflect'
- **cval** (*scalar, optional*) – Value to fill past edges of input if *mode* is 'constant'. Default is 0.0
- **origin** (*scalar, optional*) – The *origin* parameter controls the placement of the filter. Default 0.0.

dask_ndfilters.**sobel**(*input*, *axis=-1*, *mode='reflect'*, *cval=0.0*)

Wrapped copy of "scipy.ndimage.filters.sobel"

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

Original docstring:

Calculate a Sobel filter.

Parameters

- **input** (*array_like*) – Input array to filter.
- **axis** (*int, optional*) – The axis of *input* along which to calculate. Default is -1.
- **mode** (*{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional*) – The *mode* parameter determines how the array borders are handled, where *cval* is the value when mode is equal to 'constant'. Default is 'reflect'
- **cval** (*scalar, optional*) – Value to fill past edges of input if *mode* is 'constant'. Default is 0.0

Examples

```
>>> from scipy import ndimage, misc
>>> import matplotlib.pyplot as plt
>>> ascent = misc.ascent()
>>> result = ndimage.sobel(ascent)
>>> plt.gray()  # show the filtered result in grayscale
>>> plt.imshow(result)
```

dask_ndfilters.**uniform_filter**(*input*, *size=3*, *mode='reflect'*, *cval=0.0*, *origin=0*)

Wrapped copy of "scipy.ndimage.filters.uniform_filter"

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

Original docstring:

Multi-dimensional uniform filter.

Parameters

- **input** (*array_like*) – Input array to filter.
- **size** (*int or sequence of ints, optional*) – The sizes of the uniform filter are given for each axis as a sequence, or as a single number, in which case the size is equal for all axes.
- **mode** (*{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional*) – The *mode* parameter determines how the array borders are handled, where *cval* is the value when mode is equal to 'constant'. Default is 'reflect'
- **cval** (*scalar, optional*) – Value to fill past edges of input if *mode* is 'constant'. Default is 0.0
- **origin** (*scalar, optional*) – The *origin* parameter controls the placement of the filter. Default 0.0.

Notes

The multi-dimensional filter is implemented as a sequence of one-dimensional uniform filters. The intermediate arrays are stored in the same data type as the output. Therefore, for output types with a limited precision, the results may be imprecise because intermediate results may be stored with insufficient precision.

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-ndfilters/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-ndfilters could always use more documentation, whether as part of the official dask-ndfilters 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-ndfilters/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-ndfilters* for local development.

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

```
$ git clone git@github.com:your_name_here/dask-ndfilters.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_ndfiltersenv python="<some version>"
$ source activate dask_ndfiltersenv
$ 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_ndfilters 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-ndfilters/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ python -m unittest tests.test_dask_ndfilters
```

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

d

dask_ndfilters, 9

Index

C

convolve() (in module dask_ndfilters), 9
correlate() (in module dask_ndfilters), 11

D

dask_ndfilters (module), 9

G

gaussian_filter() (in module dask_ndfilters), 11
gaussian_gradient_magnitude() (in module dask_ndfilters), 12
gaussian_laplace() (in module dask_ndfilters), 13
generic_filter() (in module dask_ndfilters), 13

L

laplace() (in module dask_ndfilters), 14

M

maximum_filter() (in module dask_ndfilters), 15
median_filter() (in module dask_ndfilters), 15
minimum_filter() (in module dask_ndfilters), 16

P

percentile_filter() (in module dask_ndfilters), 16
prewitt() (in module dask_ndfilters), 17

R

rank_filter() (in module dask_ndfilters), 17

S

sobel() (in module dask_ndfilters), 18

U

uniform_filter() (in module dask_ndfilters), 18