dask-image Documentation

Release 0.6.0+19.g8e784e0.dirty

John Kirkham

Dec 17, 2021

CONTENTS

CHAPTER

ONE

FEATURES

- Support focuses on Dask Arrays.
- Provides support for loading image files.
- Implements commonly used N-D filters.
- Includes a few N-D Fourier filters.
- Provides some functions for working with N-D label images.
- Supports a few N-D morphological operators.

CHAPTER

TWO

CONTENTS

2.1 Installation

2.1.1 Stable release

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

```
$ conda install -c conda-forge dask-image
```

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

If you don't have conda installed, you can download and install it with the Anaconda distribution here.

Alternatively, you can install dask-image with pip:

```
$ pip install dask-image
```

If you don't have pip installed, this *Python installation guide* can guide you through the process. http://docs.python-guide.org/en/latest/starting/installation/

2.1.2 From sources

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

You can either clone the public repository:

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

Or download the tarball:

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

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

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

2.2 Quickstart

2.2.1 Importing dask-image

Import dask image is with an underscore, like this example:

```
import dask_image.imread
import dask_image.ndfilters
```

2.2.2 Dask Examples

We highly recommend checking out the dask-image-quickstart.ipynb notebook (and any other dask-image example notebooks) at the dask-examples repository. You can find the dask-image quickstart notebook in the `applications` folder of this repository:

https://github.com/dask/dask-examples

The direct link to the notebook file is here:

https://github.com/dask/dask-examples/blob/main/applications/image-processing.ipynb

All the example notebooks are available to launch with mybinder and test out interactively.

2.2.3 An Even Quicker Start

You can read files stored on disk into a dask array by passing the filename, or regex matching multiple filenames into `imread()`.

```
filename_pattern = 'path/to/image-*.png'
images = dask_image.imread.imread(filename_pattern)
```

If your images are parts of a much larger image, dask can stack, concatenate or block chunks together: http://docs.dask.org/en/latest/array-stack.html

Calling dask-image functions is also easy.

```
import dask_image.ndfilters
blurred_image = dask_image.ndfilters.gaussian_filter(images, sigma=10)
```

Many other functions can be applied to dask arrays. See the dask_array_documentation for more detail on general array operations.

```
result = function_name(images)
```

2.2.4 Further Reading

Good places to start include:

- The dask-image API documentation: http://image.dask.org/en/latest/api.html
- The documentation on working with dask arrays: http://docs.dask.org/en/latest/array.html

2.2.5 Talks and Slides

Here are some talks and slides that you can watch to learn dask-image:

- 2020, Genevieve Buckley's talk at PyConAU and SciPy Japan
  - Watch the talk in PyConAU
  - Scipy Japan(:, :) Watch the talk at SciPy Japan (presentation in English, captions in Japanese)
  - See the slides
- 2019, John Kirkham's SciPy talk
  - Watch the talk
  - See the slides

2.3 Function Coverage

2.3.1 Coverage of dask-image vs scipy ndimage functions

This table shows which SciPy ndimage functions are supported by dask-image.

Function name	SciPy ndimage	dask-image
`affine_transform`	✓	✓
`binary_closing`	✓	✓
`binary_dilation`	✓	✓
`binary_erosion`	✓	✓
`binary_fill_holes`	✓	
`binary_hit_or_miss`	✓	
`binary_opening`	✓	✓
`binary_propagation`	✓	
`black_tophat`	✓	
`center_of_mass`	✓	✓
`convolve`	✓	✓
`convolve1d`	✓	
`correlate`	✓	✓
`correlate1d`	✓	
`distance_transform_bf`	✓	
`distance_transform_cdt`	✓	
`distance_transform_edt`	✓	
`extrema`	✓	✓
`find_objects`	✓	✓
`fourier_ellipsoid`	✓	
`fourier_gaussian`	✓	✓
`fourier_shift`	✓	✓
`fourier_uniform`	✓	✓
`gaussian_filter`	✓	✓
`gaussian_filter1d`	✓	
`gaussian_gradient_magnitude`	✓	✓
`gaussian_laplace`	✓	✓
`generate_binary_structure`	✓	

continues on next page

Table 1 – continued from previous page

generic_filter	✓	
generic_filter1d	✓	✓
generic_gradient_magnitude	✓	
generic_laplace	✓	
geometric_transform	✓	
grey_closing	✓	
grey_dilation	✓	
grey_erosion	✓	
grey_opening	✓	
histogram	✓	✓
imread	✓	✓
iterate_structure	✓	
label	✓	✓
labeled_comprehension	✓	✓
laplace	✓	✓
map_coordinates	✓	
maximum	✓	✓
maximum_filter	✓	✓
maximum_filter1d	✓	
maximum_position	✓	✓
mean	✓	✓
median	✓	✓
median_filter	✓	✓
minimum	✓	✓
minimum_filter	✓	✓
minimum_filter1d	✓	
minimum_position	✓	✓
morphological_gradient	✓	
morphological_laplace	✓	
percentile_filter	✓	✓
prewitt	✓	✓
rank_filter	✓	✓
rotate	✓	
shift	✓	
sobel	✓	✓
spline_filter	✓	✓
spline_filter1d	✓	✓
standard_deviation	✓	✓
sum_labels	✓	✓
uniform_filter	✓	✓
uniform_filter1d	✓	
variance	✓	✓
watershed_ift	✓	
white_tophat	✓	
zoom	✓	

2.4 API

2.4.1 dask_image package

Subpackages

dask_image.dispatch package

dask_image.imread package

`dask_image.imread.`**`imread`**(*fname*, *nframes=1*, ***, *arraytype='numpy'*)

Read image data into a Dask Array.

Provides a simple, fast mechanism to ingest image data into a Dask Array.

Parameters

- **`fname`** (*`str or pathlib.Path`*) – A glob like string that may match one or multiple filenames.
- **`nframes`** (*`int, optional`*) – Number of the frames to include in each chunk (default: 1).
- **`arraytype`** (*`str, optional`*) – Array type for dask chunks. Available options: "numpy", "cupy".

Returns **array** – A Dask Array representing the contents of all image files.

Return type dask.array.Array

dask_image.ndfilters package

`dask_image.ndfilters.`**`convolve`**(*image*, *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

- **`image`** (*`array_like`*) – The image array.
- **`weights`** (*`array_like`*) – Array of weights, same number of dimensions as image
- **`mode`** (*`{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional`*) – The *mode* parameter determines how the image array is extended beyond its boundaries. Default is 'reflect'. Behavior for each valid value is as follows:

  'reflect' (*d c b a | a b c d | d c b a*) The image is extended by reflecting about the edge of the last pixel.

  'constant' (*k k k k | a b c d | k k k k*) The image is extended by filling all values beyond the edge with the same constant value, defined by the *cval* parameter.

  'nearest' (*a a a a | a b c d | d d d d*) The image is extended by replicating the last pixel.

'mirror' (*d c b | a b c d | c b a*) The image is extended by reflecting about the center of the last pixel.

'wrap' (*a b c d | a b c d | a b c d*) The image is extended by wrapping around to the opposite edge.

- **cval** (*scalar, optional*) – Value to fill past edges of image if *mode* is 'constant'. Default is 0.0
- **origin** (*int or sequence, optional*) – Controls the placement of the filter on the image array's pixels. A value of 0 (the default) centers the filter over the pixel, with positive values shifting the filter to the left, and negative ones to the right. By passing a sequence of origins with length equal to the number of dimensions of the image array, different shifts can be specified along each axis.

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

Return type ndarray

See also:

`correlate()` Correlate an image with a kernel.

Notes

Each value in result is $C_i = \sum_j I_{i+k-j} W_j$, where W is the *weights* kernel, j is the N-D spatial index over W, I is the *image* and k is the coordinate of the center of W, specified by *origin* in the image parameters.

`dask_image.ndfilters.`**`correlate`**(*image*, *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:

Multidimensional correlation.

The array is correlated with the given kernel.

Parameters

- **image** (*array_like*) – The image array.
- **weights** (*ndarray*) – array of weights, same number of dimensions as image
- **mode** (*{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional*) – The *mode* parameter determines how the image array is extended beyond its boundaries. Default is 'reflect'. Behavior for each valid value is as follows:

'reflect' (*d c b a | a b c d | d c b a*) The image is extended by reflecting about the edge of the last pixel.

'constant' (*k k k k | a b c d | k k k k*) The image is extended by filling all values beyond the edge with the same constant value, defined by the *cval* parameter.

'nearest' (*a a a a | a b c d | d d d d*) The image is extended by replicating the last pixel.

'mirror' (*d c b | a b c d | c b a*) The image is extended by reflecting about the center of the last pixel.

'wrap' (*a b c d | a b c d | a b c d*) The image is extended by wrapping around to the opposite edge.

- **cval** (*scalar, optional*) – Value to fill past edges of image if *mode* is 'constant'. Default is 0.0.
- **origin** (*int or sequence, optional*) – Controls the placement of the filter on the image array's pixels. A value of 0 (the default) centers the filter over the pixel, with positive values shifting the filter to the left, and negative ones to the right. By passing a sequence of origins with length equal to the number of dimensions of the image array, different shifts can be specified along each axis.

Returns result – The result of correlation of *image* with *weights*.

Return type ndarray

See also:

`convolve()` Convolve an image with a kernel.

`dask_image.ndfilters.gaussian_filter`(*image*, *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

- **image** (*array_like*) – The image array.
- **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** (*int or sequence of ints, 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. A positive order corresponds to convolution with that derivative of a Gaussian.
- **mode** (*str or sequence, optional*) – The *mode* parameter determines how the image array is extended when the filter overlaps a border. By passing a sequence of modes with length equal to the number of dimensions of the image array, different modes can be specified along each axis. Default value is 'reflect'. The valid values and their behavior is as follows:

  'reflect' (*d c b a | a b c d | d c b a*) The image is extended by reflecting about the edge of the last pixel.

  'constant' (*k k k k | a b c d | k k k k*) The image is extended by filling all values beyond the edge with the same constant value, defined by the *cval* parameter.

  'nearest' (*a a a a | a b c d | d d d d*) The image is extended by replicating the last pixel.

  'mirror' (*d c b | a b c d | c b a*) The image is extended by reflecting about the center of the last pixel.

  'wrap' (*a b c d | a b c d | a b c d*) The image is extended by wrapping around to the opposite edge.
- **cval** (*scalar, optional*) – Value to fill past edges of image 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 *image*.

Return type ndarray

Notes

The multidimensional filter is implemented as a sequence of 1-D 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.

`dask_image.ndfilters.`**`gaussian_gradient_magnitude`**(*image*, *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

- **`image`** (*`array_like`*) – The image array.
- **`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`** (*`str or sequence, optional`*) – The *mode* parameter determines how the image array is extended when the filter overlaps a border. By passing a sequence of modes with length equal to the number of dimensions of the image array, different modes can be specified along each axis. Default value is 'reflect'. The valid values and their behavior is as follows:

  'reflect' (*d c b a | a b c d | d c b a*) The image is extended by reflecting about the edge of the last pixel.

  'constant' (*k k k k | a b c d | k k k k*) The image is extended by filling all values beyond the edge with the same constant value, defined by the *cval* parameter.

  'nearest' (*a a a a | a b c d | d d d d*) The image is extended by replicating the last pixel.

  'mirror' (*d c b | a b c d | c b a*) The image is extended by reflecting about the center of the last pixel.

  'wrap' (*a b c d | a b c d | a b c d*) The image is extended by wrapping around to the opposite edge.
- **`cval`** (*`scalar, optional`*) – Value to fill past edges of image if *mode* is 'constant'. Default is 0.0.
- **`keyword arguments will be passed to gaussian_filter()`** (*`Extra`*) –

Returns **gaussian_gradient_magnitude** – Filtered array. Has the same shape as *image*.

Return type ndarray

`dask_image.ndfilters.`**`gaussian_laplace`**(*image*, *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

- **`image`** (*`array_like`*) – The image array.
- **`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`** (*`str or sequence, optional`*) – The *mode* parameter determines how the image array is extended when the filter overlaps a border. By passing a sequence of modes with length equal to the number of dimensions of the image array, different modes can be specified along each axis. Default value is 'reflect'. The valid values and their behavior is as follows:

  'reflect' (*d c b a | a b c d | d c b a*) The image is extended by reflecting about the edge of the last pixel.

  'constant' (*k k k k | a b c d | k k k k*) The image is extended by filling all values beyond the edge with the same constant value, defined by the *cval* parameter.

  'nearest' (*a a a a | a b c d | d d d d*) The image is extended by replicating the last pixel.

  'mirror' (*d c b | a b c d | c b a*) The image is extended by reflecting about the center of the last pixel.

  'wrap' (*a b c d | a b c d | a b c d*) The image is extended by wrapping around to the opposite edge.
- **`cval`** (*`scalar, optional`*) – Value to fill past edges of image if *mode* is 'constant'. Default is 0.0.
- **`keyword arguments will be passed to gaussian_filter()`** (*`Extra`*) –

`dask_image.ndfilters.`**`generic_filter`**(*image*, *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:

Calculate a multidimensional filter using the given function.

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

Parameters

- **`image`** (*`array_like`*) – The image array.
- **`function`** (*`{callable, scipy.LowLevelCallable}`*) – Function to apply at each element.
- **`size`** (*`scalar or tuple, optional`*) – See footprint, below. Ignored if footprint is given.
- **`footprint`** (*`array, optional`*) – Either *size* or *footprint* must be defined. *size* gives the shape that is taken from the image array, at every element position, to define the image 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 image array, so that, if the image array is shape (10,10,10), and *size* is 2, then the actual size used is (2,2,2). When *footprint* is given, *size* is ignored.

- **mode** (*{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional*) – The *mode* parameter determines how the image array is extended beyond its boundaries. Default is 'reflect'. Behavior for each valid value is as follows:

  'reflect' (*d c b a | a b c d | d c b a*) The image is extended by reflecting about the edge of the last pixel.

  'constant' (*k k k k | a b c d | k k k k*) The image is extended by filling all values beyond the edge with the same constant value, defined by the *cval* parameter.

  'nearest' (*a a a a | a b c d | d d d d*) The image is extended by replicating the last pixel.

  'mirror' (*d c b | a b c d | c b a*) The image is extended by reflecting about the center of the last pixel.

  'wrap' (*a b c d | a b c d | a b c d*) The image is extended by wrapping around to the opposite edge.

- **cval** (*scalar, optional*) – Value to fill past edges of image if *mode* is 'constant'. Default is 0.0.
- **origin** (*int or sequence, optional*) – Controls the placement of the filter on the image array's pixels. A value of 0 (the default) centers the filter over the pixel, with positive values shifting the filter to the left, and negative ones to the right. By passing a sequence of origins with length equal to the number of dimensions of the image array, different shifts can be specified along each axis.
- **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.

Notes

This function also accepts low-level callback functions with one of the following signatures and wrapped in *scipy.LowLevelCallable*:

```
int callback(double *buffer, npy_intp filter_size,
             double *return_value, void *user_data)
int callback(double *buffer, intptr_t filter_size,
             double *return_value, void *user_data)
```

The calling function iterates over the elements of the image and output arrays, calling the callback function at each element. The elements within the footprint of the filter at the current element are passed through the `buffer` parameter, and the number of elements within the footprint through `filter_size`. The calculated value is returned in `return_value`. `user_data` is the data pointer provided to *scipy.LowLevelCallable* as-is.

The callback function must return an integer error status that is zero if something went wrong and one otherwise. If an error occurs, you should normally set the python error status with an informative message before returning, otherwise a default error message is set by the calling function.

In addition, some other low-level function pointer specifications are accepted, but these are for backward compatibility only and should not be used in new code.

`dask_image.ndfilters.`**`laplace`**(*image*, *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-D Laplace filter based on approximate second derivatives.

Parameters

- **`image`** (*`array_like`*) – The image array.
- **`mode`** (*`str or sequence, optional`*) – The *mode* parameter determines how the image array is extended when the filter overlaps a border. By passing a sequence of modes with length equal to the number of dimensions of the image array, different modes can be specified along each axis. Default value is ‘reflect’. The valid values and their behavior is as follows:

  ’reflect’ (*d c b a | a b c d | d c b a*) The image is extended by reflecting about the edge of the last pixel.

  ’constant’ (*k k k k | a b c d | k k k k*) The image is extended by filling all values beyond the edge with the same constant value, defined by the *cval* parameter.

  ’nearest’ (*a a a a | a b c d | d d d d*) The image is extended by replicating the last pixel.

  ’mirror’ (*d c b | a b c d | c b a*) The image is extended by reflecting about the center of the last pixel.

  ’wrap’ (*a b c d | a b c d | a b c d*) The image is extended by wrapping around to the opposite edge.
- **`cval`** (*`scalar, optional`*) – Value to fill past edges of image if *mode* is ‘constant’. Default is 0.0.

`dask_image.ndfilters.`**`maximum_filter`**(*image*, *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:

Calculate a multidimensional maximum filter.

Parameters

- **`image`** (*`array_like`*) – The image array.
- **`size`** (*`scalar or tuple, optional`*) – See footprint, below. Ignored if footprint is given.
- **`footprint`** (*`array, optional`*) – Either *size* or *footprint* must be defined. *size* gives the shape that is taken from the image array, at every element position, to define the image 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 image array, so that, if the image array is shape (10,10,10), and *size* is 2, then the actual size used is (2,2,2). When *footprint* is given, *size* is ignored.
- **`mode`** (*`str or sequence, optional`*) – The *mode* parameter determines how the image array is extended when the filter overlaps a border. By passing a sequence of modes with length equal to the number of dimensions of the image array, different modes can be

specified along each axis. Default value is 'reflect'. The valid values and their behavior is as follows:

'reflect' (*d c b a | a b c d | d c b a*) The image is extended by reflecting about the edge of the last pixel.

'constant' (*k k k k | a b c d | k k k k*) The image is extended by filling all values beyond the edge with the same constant value, defined by the *cval* parameter.

'nearest' (*a a a a | a b c d | d d d d*) The image is extended by replicating the last pixel.

'mirror' (*d c b | a b c d | c b a*) The image is extended by reflecting about the center of the last pixel.

'wrap' (*a b c d | a b c d | a b c d*) The image is extended by wrapping around to the opposite edge.

- **`cval`** (*`scalar, optional`*) – Value to fill past edges of image if *mode* is 'constant'. Default is 0.0.
- **`origin`** (*`int or sequence, optional`*) – Controls the placement of the filter on the image array's pixels. A value of 0 (the default) centers the filter over the pixel, with positive values shifting the filter to the left, and negative ones to the right. By passing a sequence of origins with length equal to the number of dimensions of the image array, different shifts can be specified along each axis.

Returns **maximum_filter** – Filtered array. Has the same shape as *image*.

Return type ndarray

Notes

A sequence of modes (one per axis) is only supported when the footprint is separable. Otherwise, a single mode string must be provided.

`dask_image.ndfilters.`**`median_filter`**(*image, 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:

Calculate a multidimensional median filter.

Parameters

- **`image`** (*`array_like`*) – The image array.
- **`size`** (*`scalar or tuple, optional`*) – See footprint, below. Ignored if footprint is given.
- **`footprint`** (*`array, optional`*) – Either *size* or *footprint* must be defined. *size* gives the shape that is taken from the image array, at every element position, to define the image 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 image array, so that, if the image array is shape (10,10,10), and *size* is 2, then the actual size used is (2,2,2). When *footprint* is given, *size* is ignored.

- **mode** (*{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional*) – The *mode* parameter determines how the image array is extended beyond its boundaries. Default is 'reflect'. Behavior for each valid value is as follows:

  'reflect' (*d c b a | a b c d | d c b a*) The image is extended by reflecting about the edge of the last pixel.

  'constant' (*k k k k | a b c d | k k k k*) The image is extended by filling all values beyond the edge with the same constant value, defined by the *cval* parameter.

  'nearest' (*a a a a | a b c d | d d d d*) The image is extended by replicating the last pixel.

  'mirror' (*d c b | a b c d | c b a*) The image is extended by reflecting about the center of the last pixel.

  'wrap' (*a b c d | a b c d | a b c d*) The image is extended by wrapping around to the opposite edge.

- **cval** (*scalar, optional*) – Value to fill past edges of image if *mode* is 'constant'. Default is 0.0.
- **origin** (*int or sequence, optional*) – Controls the placement of the filter on the image array's pixels. A value of 0 (the default) centers the filter over the pixel, with positive values shifting the filter to the left, and negative ones to the right. By passing a sequence of origins with length equal to the number of dimensions of the image array, different shifts can be specified along each axis.

Returns **median_filter** – Filtered array. Has the same shape as *image*.

Return type ndarray

`dask_image.ndfilters.`**`minimum_filter`**(*image*, *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:

Calculate a multidimensional minimum filter.

Parameters

- **image** (*array_like*) – The image array.
- **size** (*scalar or tuple, optional*) – See footprint, below. Ignored if footprint is given.
- **footprint** (*array, optional*) – Either *size* or *footprint* must be defined. *size* gives the shape that is taken from the image array, at every element position, to define the image 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 image array, so that, if the image array is shape (10,10,10), and *size* is 2, then the actual size used is (2,2,2). When *footprint* is given, *size* is ignored.
- **mode** (*str or sequence, optional*) – The *mode* parameter determines how the image array is extended when the filter overlaps a border. By passing a sequence of modes with length equal to the number of dimensions of the image array, different modes can be specified along each axis. Default value is 'reflect'. The valid values and their behavior is as follows:

  'reflect' (*d c b a | a b c d | d c b a*) The image is extended by reflecting about the edge of the last pixel.

'constant' (*k k k k | a b c d | k k k k*) The image is extended by filling all values beyond the edge with the same constant value, defined by the *cval* parameter.

'nearest' (*a a a a | a b c d | d d d d*) The image is extended by replicating the last pixel.

'mirror' (*d c b | a b c d | c b a*) The image is extended by reflecting about the center of the last pixel.

'wrap' (*a b c d | a b c d | a b c d*) The image is extended by wrapping around to the opposite edge.

- **`cval`** (*`scalar, optional`*) – Value to fill past edges of image if *mode* is 'constant'. Default is 0.0.
- **`origin`** (*`int or sequence, optional`*) – Controls the placement of the filter on the image array's pixels. A value of 0 (the default) centers the filter over the pixel, with positive values shifting the filter to the left, and negative ones to the right. By passing a sequence of origins with length equal to the number of dimensions of the image array, different shifts can be specified along each axis.

Returns **minimum_filter** – Filtered array. Has the same shape as *image*.

Return type ndarray

Notes

A sequence of modes (one per axis) is only supported when the footprint is separable. Otherwise, a single mode string must be provided.

`dask_image.ndfilters.`**`percentile_filter`**(*image*, *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:

Calculate a multidimensional percentile filter.

Parameters

- **`image`** (*`array_like`*) – The image array.
- **`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. Ignored if footprint is given.
- **`footprint`** (*`array, optional`*) – Either *size* or *footprint* must be defined. *size* gives the shape that is taken from the image array, at every element position, to define the image 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 image array, so that, if the image array is shape (10,10,10), and *size* is 2, then the actual size used is (2,2,2). When *footprint* is given, *size* is ignored.
- **`mode`** (*`{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional`*) – The *mode* parameter determines how the image array is extended beyond its boundaries. Default is 'reflect'. Behavior for each valid value is as follows:

'reflect' (*d c b a | a b c d | d c b a*) The image is extended by reflecting about the edge of the last pixel.

'constant' (*k k k k | a b c d | k k k k*) The image is extended by filling all values beyond the edge with the same constant value, defined by the *cval* parameter.

'nearest' (*a a a a | a b c d | d d d d*) The image is extended by replicating the last pixel.

'mirror' (*d c b | a b c d | c b a*) The image is extended by reflecting about the center of the last pixel.

'wrap' (*a b c d | a b c d | a b c d*) The image is extended by wrapping around to the opposite edge.

- **cval** (*scalar, optional*) – Value to fill past edges of image if *mode* is 'constant'. Default is 0.0.
- **origin** (*int or sequence, optional*) – Controls the placement of the filter on the image array's pixels. A value of 0 (the default) centers the filter over the pixel, with positive values shifting the filter to the left, and negative ones to the right. By passing a sequence of origins with length equal to the number of dimensions of the image array, different shifts can be specified along each axis.

Returns **percentile_filter** – Filtered array. Has the same shape as *image*.

Return type ndarray

`dask_image.ndfilters.`**`prewitt`**(*image*, *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

- **image** (*array_like*) – The image array.
- **axis** (*int, optional*) – The axis of *image* along which to calculate. Default is -1.
- **mode** (*str or sequence, optional*) – The *mode* parameter determines how the image array is extended when the filter overlaps a border. By passing a sequence of modes with length equal to the number of dimensions of the image array, different modes can be specified along each axis. Default value is 'reflect'. The valid values and their behavior is as follows:

'reflect' (*d c b a | a b c d | d c b a*) The image is extended by reflecting about the edge of the last pixel.

'constant' (*k k k k | a b c d | k k k k*) The image is extended by filling all values beyond the edge with the same constant value, defined by the *cval* parameter.

'nearest' (*a a a a | a b c d | d d d d*) The image is extended by replicating the last pixel.

'mirror' (*d c b | a b c d | c b a*) The image is extended by reflecting about the center of the last pixel.

'wrap' (*a b c d | a b c d | a b c d*) The image is extended by wrapping around to the opposite edge.

- **cval** (*scalar, optional*) – Value to fill past edges of image if *mode* is 'constant'. Default is 0.0.

`dask_image.ndfilters.`**`rank_filter`**(*image*, *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:

Calculate a multidimensional rank filter.

Parameters

- **`image`** (*`array_like`*) – The image array.
- **`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. Ignored if footprint is given.
- **`footprint`** (*`array, optional`*) – Either *size* or *footprint* must be defined. *size* gives the shape that is taken from the image array, at every element position, to define the image 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 image array, so that, if the image array is shape (10,10,10), and *size* is 2, then the actual size used is (2,2,2). When *footprint* is given, *size* is ignored.
- **`mode`** (*`{'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional`*) – The *mode* parameter determines how the image array is extended beyond its boundaries. Default is 'reflect'. Behavior for each valid value is as follows:

  'reflect' (*d c b a | a b c d | d c b a*) The image is extended by reflecting about the edge of the last pixel.

  'constant' (*k k k k | a b c d | k k k k*) The image is extended by filling all values beyond the edge with the same constant value, defined by the *cval* parameter.

  'nearest' (*a a a a | a b c d | d d d d*) The image is extended by replicating the last pixel.

  'mirror' (*d c b | a b c d | c b a*) The image is extended by reflecting about the center of the last pixel.

  'wrap' (*a b c d | a b c d | a b c d*) The image is extended by wrapping around to the opposite edge.
- **`cval`** (*`scalar, optional`*) – Value to fill past edges of image if *mode* is 'constant'. Default is 0.0.
- **`origin`** (*`int or sequence, optional`*) – Controls the placement of the filter on the image array's pixels. A value of 0 (the default) centers the filter over the pixel, with positive values shifting the filter to the left, and negative ones to the right. By passing a sequence of origins with length equal to the number of dimensions of the image array, different shifts can be specified along each axis.

Returns **rank_filter** – Filtered array. Has the same shape as *image*.

Return type ndarray

`dask_image.ndfilters.`**`sobel`**(*image*, *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

- **`image`** (*`array_like`*) – The image array.
- **`axis`** (*`int, optional`*) – The axis of *image* along which to calculate. Default is -1.
- **`mode`** (*`str or sequence, optional`*) – The *mode* parameter determines how the image array is extended when the filter overlaps a border. By passing a sequence of modes with length equal to the number of dimensions of the image array, different modes can be specified along each axis. Default value is 'reflect'. The valid values and their behavior is as follows:

  'reflect' (*d c b a | a b c d | d c b a*) The image is extended by reflecting about the edge of the last pixel.

  'constant' (*k k k k | a b c d | k k k k*) The image is extended by filling all values beyond the edge with the same constant value, defined by the *cval* parameter.

  'nearest' (*a a a a | a b c d | d d d d*) The image is extended by replicating the last pixel.

  'mirror' (*d c b | a b c d | c b a*) The image is extended by reflecting about the center of the last pixel.

  'wrap' (*a b c d | a b c d | a b c d*) The image is extended by wrapping around to the opposite edge.
- **`cval`** (*`scalar, optional`*) – Value to fill past edges of image if *mode* is 'constant'. Default is 0.0.

`dask_image.ndfilters.`**`threshold_local`**(*image*, *block_size*, *method='gaussian'*, *offset=0*, *mode='reflect'*, *param=None*, *cval=0*)

Compute a threshold mask image based on local pixel neighborhood.

Also known as adaptive or dynamic thresholding[1]_. The threshold value is the weighted mean for the local neighborhood of a pixel subtracted by a constant. Alternatively the threshold can be determined dynamically by a given function, using the 'generic' method.

Parameters

- **`image`** (*`(N, M) dask ndarray`*) – Input image.
- **`block_size`** (*`int or list/tuple/array`*) – Size of pixel neighborhood which is used to calculate the threshold value. (1) A single value for use in all dimensions or (2) A tuple, list, or array with length equal to image.ndim
- **`method`** (*`{'generic', 'gaussian', 'mean', 'median'}, optional`*) – Method used to determine adaptive threshold for local neighbourhood in weighted mean image.
  - 'generic': use custom function (see *param* parameter)
  - 'gaussian': apply gaussian filter (see *param* parameter for custom sigma value)
  - 'mean': apply arithmetic mean filter
  - 'median': apply median rank filter

  By default the 'gaussian' method is used.
- **`offset`** (*`float, optional`*) – Constant subtracted from weighted mean of neighborhood to calculate the local threshold value. Default offset is 0.

- **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'.
- **param** (*{int, function}, optional*) – Either specify sigma for 'gaussian' method or function object for 'generic' method. This functions takes the flat array of local neighbourhood as a single argument and returns the calculated threshold for the centre pixel.
- **cval** (*float, optional*) – Value to fill past edges of input if mode is 'constant'.

Returns **threshold** – Threshold image. All pixels in the input image higher than the corresponding pixel in the threshold image are considered foreground.

Return type (N, M) dask ndarray

References

Examples

```
>>> import dask.array as da
>>> image = da.random.random((1000, 1000), chunks=(100, 100))
>>> result = threshold_local(image, 15, 'gaussian')
```

dask_image.ndfilters.**uniform_filter**(*image*, *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:

Multidimensional uniform filter.

Parameters

- **image** (*array_like*) – The image array.
- **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** (*str or sequence, optional*) – The *mode* parameter determines how the image array is extended when the filter overlaps a border. By passing a sequence of modes with length equal to the number of dimensions of the image array, different modes can be specified along each axis. Default value is 'reflect'. The valid values and their behavior is as follows:

  'reflect' (*d c b a | a b c d | d c b a*) The image is extended by reflecting about the edge of the last pixel.

  'constant' (*k k k k | a b c d | k k k k*) The image is extended by filling all values beyond the edge with the same constant value, defined by the *cval* parameter.

  'nearest' (*a a a a | a b c d | d d d d*) The image is extended by replicating the last pixel.

  'mirror' (*d c b | a b c d | c b a*) The image is extended by reflecting about the center of the last pixel.

  'wrap' (*a b c d | a b c d | a b c d*) The image is extended by wrapping around to the opposite edge.

- **cval** (*scalar, optional*) – Value to fill past edges of image if *mode* is 'constant'. Default is 0.0.
- **origin** (*int or sequence, optional*) – Controls the placement of the filter on the image array's pixels. A value of 0 (the default) centers the filter over the pixel, with positive values shifting the filter to the left, and negative ones to the right. By passing a sequence of origins with length equal to the number of dimensions of the image array, different shifts can be specified along each axis.

Returns **uniform_filter** – Filtered array. Has the same shape as *image*.

Return type ndarray

Notes

The multidimensional filter is implemented as a sequence of 1-D 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.

dask_image.ndfourier package

dask_image.ndfourier.**fourier_gaussian**(*image*, *sigma*, *n=- 1*, *axis=- 1*)

Multi-dimensional Gaussian fourier filter.

The array is multiplied with the fourier transform of a Gaussian kernel.

Parameters

- **image** (*array_like*) – The input image.
- **sigma** (*float or sequence*) – The sigma of the Gaussian kernel. If a float, *sigma* is the same for all axes. If a sequence, *sigma* has to contain one value for each axis.
- **n** (*int, optional*) – If *n* is negative (default), then the image is assumed to be the result of a complex fft. If *n* is larger than or equal to zero, the image is assumed to be the result of a real fft, and *n* gives the length of the array before transformation along the real transform direction.
- **axis** (*int, optional*) – The axis of the real transform.

Returns **fourier_gaussian**

Return type Dask Array

Examples

```
>>> from scipy import ndimage, misc
>>> import numpy.fft
>>> import matplotlib.pyplot as plt
>>> fig, (ax1, ax2) = plt.subplots(1, 2)
>>> plt.gray()  # show the filtered result in grayscale
>>> ascent = misc.ascent()
>>> image = numpy.fft.fft2(ascent)
>>> result = ndimage.fourier_gaussian(image, sigma=4)
>>> result = numpy.fft.ifft2(result)
>>> ax1.imshow(ascent)
```

`dask_image.ndfourier.`**`fourier_shift`**(*image*, *shift*, *n=- 1*, *axis=- 1*)

Multi-dimensional fourier shift filter.

The array is multiplied with the fourier transform of a shift operation.

Parameters

- **`image`** (*`array_like`*) – The input image.
- **`shift`** (*`float or sequence`*) – The size of the box used for filtering. If a float, *shift* is the same for all axes. If a sequence, *shift* has to contain one value for each axis.
- **`n`** (*`int, optional`*) – If *n* is negative (default), then the image is assumed to be the result of a complex fft. If *n* is larger than or equal to zero, the image is assumed to be the result of a real fft, and *n* gives the length of the array before transformation along the real transform direction.
- **`axis`** (*`int, optional`*) – The axis of the real transform.

Returns **fourier_shift**

Return type Dask Array

Examples

```
>>> from scipy import ndimage, misc
>>> import matplotlib.pyplot as plt
>>> import numpy.fft
>>> fig, (ax1, ax2) = plt.subplots(1, 2)
>>> plt.gray()  # show the filtered result in grayscale
>>> ascent = misc.ascent()
>>> image = numpy.fft.fft2(ascent)
>>> result = ndimage.fourier_shift(image, shift=200)
>>> result = numpy.fft.ifft2(result)
>>> ax1.imshow(ascent)
>>> ax2.imshow(result.real)  # the imaginary part is an artifact
>>> plt.show()
```

`dask_image.ndfourier.`**`fourier_uniform`**(*image*, *size*, *n=- 1*, *axis=- 1*)

Multi-dimensional uniform fourier filter.

The array is multiplied with the fourier transform of a box of given size.

Parameters

- **`image`** (*`array_like`*) – The input image.
- **`size`** (*`float or sequence`*) – The size of the box used for filtering. If a float, *size* is the same for all axes. If a sequence, *size* has to contain one value for each axis.
- **`n`** (*`int, optional`*) – If *n* is negative (default), then the image is assumed to be the result of a complex fft. If *n* is larger than or equal to zero, the image is assumed to be the result of a real fft, and *n* gives the length of the array before transformation along the real transform direction.
- **`axis`** (*`int, optional`*) – The axis of the real transform.

Returns **fourier_uniform** – The filtered image. If *output* is given as a parameter, None is returned.

Return type Dask Array

Examples

```
>>> from scipy import ndimage, misc
>>> import numpy.fft
>>> import matplotlib.pyplot as plt
>>> fig, (ax1, ax2) = plt.subplots(1, 2)
>>> plt.gray()  # show the filtered result in grayscale
>>> ascent = misc.ascent()
>>> image = numpy.fft.fft2(ascent)
>>> result = ndimage.fourier_uniform(image, size=20)
>>> result = numpy.fft.ifft2(result)
>>> ax1.imshow(ascent)
>>> ax2.imshow(result.real)  # the imaginary part is an artifact
>>> plt.show()
```

dask_image.ndinterp package

`dask_image.ndinterp.`**`affine_transform`**(*image*, *matrix*, *offset=0.0*, *output_shape=None*, *order=1*, *output_chunks=None*, ***kwargs*)

Apply an affine transform using Dask. For every output chunk, only the slice containing the relevant part of the image is processed. Chunkwise processing is performed either using *ndimage.affine_transform* or *cupyx.scipy.ndimage.affine_transform*, depending on the input type.

Notes

Differences to *ndimage.affine_transformation*: - currently, prefiltering is not supported

(affecting the output in case of interpolation *order > 1*)

- default order is 1
- modes 'reflect', 'mirror' and 'wrap' are not supported

Arguments equal to *ndimage.affine_transformation*, except for *output_chunks*.

Parameters

- **image** (*array_like (Numpy Array, Cupy Array, Dask Array...)*) – The image array.
- **matrix** (*array (ndim,), (ndim, ndim), (ndim, ndim+1) or (ndim+1, ndim+1)*) – Transformation matrix.
- **offset** (*float or sequence, optional*) – The offset into the array where the transform is applied. If a float, *offset* is the same for each axis. If a sequence, *offset* should contain one value for each axis.
- **output_shape** (*tuple of ints, optional*) – The shape of the array to be returned.
- **order** (*int, optional*) – The order of the spline interpolation. Note that for order>1 scipy's affine_transform applies prefiltering, which is not yet supported and skipped in this implementation.
- **output_chunks** (*tuple of ints, optional*) – The shape of the chunks of the output Dask Array.

Returns **affine_transform** – A dask array representing the transformed output

Return type Dask Array

dask_image.ndmeasure package

`dask_image.ndmeasure.`**`area`**(*image*, *label_image=None*, *index=None*)

Find the area of specified subregions in an image.

Parameters

- **`image`** (*`ndarray`*) – N-D image data
- **`label_image`** (*`ndarray, optional`*) – Image features noted by integers. If None (default), returns area of total image dimensions.
- **`index`** (*`int or sequence of ints, optional`*) – Labels to include in output. If None (default), all values where non-zero `label_image` are used. The `index` argument only works when `label_image` is specified.

Returns **area** – Area of `index` selected regions from `label_image`.

Return type ndarray

Example

```
>>> import dask.array as da
>>> image = da.random.random((3, 3))
>>> label_image = da.from_array(
    [[1, 1, 0],
     [1, 0, 3],
     [0, 7, 0]], chunks=(1, 3))
```

```
>>> # No labels given, returns area of total image dimensions
>>> area(image)
9
```

```
>>> # Combined area of all non-zero labels
>>> area(image, label_image).compute()
5
```

```
>>> # Areas of selected labels selected with the ``index`` keyword argument
>>> area(image, label_image, index=[0, 1, 2, 3]).compute()
array([4, 3, 0, 1], dtype=int64)
```

`dask_image.ndmeasure.`**`center_of_mass`**(*image*, *label_image=None*, *index=None*)

Find the center of mass over an image at specified subregions.

Parameters

- **`image`** (*`ndarray`*) – N-D image data
- **`label_image`** (*`ndarray, optional`*) – Image features noted by integers. If None (default), all values.
- **`index`** (*`int or sequence of ints, optional`*) – Labels to include in output. If None (default), all values where non-zero `label_image` are used.

  The `index` argument only works when `label_image` is specified.

Returns **center_of_mass** – Coordinates of centers-of-mass of `image` over the `index` selected regions from `label_image`.

Return type ndarray

`dask_image.ndmeasure.`**`extrema`**(*image*, *label_image=None*, *index=None*)

Find the min and max with positions over an image at specified subregions.

Parameters

- **`image`** (*`ndarray`*) – N-D image data
- **`label_image`** (*`ndarray, optional`*) – Image features noted by integers. If None (default), all values.
- **`index`** (*`int or sequence of ints, optional`*) – Labels to include in output. If None (default), all values where non-zero `label_image` are used.

  The `index` argument only works when `label_image` is specified.

Returns **minimums, maximums, min_positions, max_positions** – Values and coordinates of minimums and maximums in each feature.

Return type tuple of ndarrays

`dask_image.ndmeasure.`**`histogram`**(*image*, *min*, *max*, *bins*, *label_image=None*, *index=None*)

Find the histogram over an image at specified subregions.

Histogram calculates the frequency of values in an array within bins determined by `min`, `max`, and `bins`. The `label_image` and `index` keywords can limit the scope of the histogram to specified sub-regions within the array.

Parameters

- **`image`** (*`ndarray`*) – N-D image data
- **`min`** (*`int`*) – Minimum value of range of histogram bins.
- **`max`** (*`int`*) – Maximum value of range of histogram bins.
- **`bins`** (*`int`*) – Number of bins.
- **`label_image`** (*`ndarray, optional`*) – Image features noted by integers. If None (default), all values.
- **`index`** (*`int or sequence of ints, optional`*) – Labels to include in output. If None (default), all values where non-zero `label_image` are used.

  The `index` argument only works when `label_image` is specified.

Returns **histogram** – Histogram of `image` over the `index` selected regions from `label_image`.

Return type ndarray

`dask_image.ndmeasure.`**`label`**(*image*, *structure=None*)

Label features in an array.

Parameters

- **`image`** (*`ndarray`*) – An array-like object to be labeled. Any non-zero values in `image` are counted as features and zero values are considered the background.
- **`structure`** (*`ndarray, optional`*) – A structuring element that defines feature connections. `structure` must be symmetric. If no structuring element is provided, one is automatically generated with a squared connectivity equal to one. That is, for a 2-D `image` array, the default structuring element is:

```
[[0,1,0],
 [1,1,1],
 [0,1,0]]
```

Returns

- **label** (*ndarray or int*) – An integer ndarray where each unique feature in `image` has a unique label in the returned array.
- **num_features** (*int*) – How many objects were found.

`dask_image.ndmeasure.`**`labeled_comprehension`**(*image*, *label_image*, *index*, *func*, *out_dtype*, *default*, *pass_positions=False*)

Compute a function over an image at specified subregions.

Roughly equivalent to [func(image[labels == i]) for i in index].

Sequentially applies an arbitrary function (that works on array_like image) to subsets of an n-D image array specified by `label_image` and `index`. The option exists to provide the function with positional parameters as the second argument.

Parameters

- **`image`** (*`ndarray`*) – N-D image data
- **`label_image`** (*`ndarray, optional`*) – Image features noted by integers. If None (default), all values.
- **`index`** (*`int or sequence of ints, optional`*) – Labels to include in output. If None (default), all values where non-zero `label_image` are used.

  The `index` argument only works when `label_image` is specified.
- **`func`** (*`callable`*) – Python function to apply to `label_image` from `image`.
- **`out_dtype`** (*`dtype`*) – Dtype to use for `result`.
- **`default`** (*`int, float or None`*) – Default return value when a element of `index` does not exist in `label_image`.
- **`pass_positions`** (*`bool, optional`*) – If True, pass linear indices to `func` as a second argument. Default is False.

Returns **result** – Result of applying `func` on `image` over the `index` selected regions from `label_image`.

Return type ndarray

`dask_image.ndmeasure.`**`maximum`**(*image*, *label_image=None*, *index=None*)

Find the maxima over an image at specified subregions.

Parameters

- **`image`** (*`ndarray`*) – N-D image data
- **`label_image`** (*`ndarray, optional`*) – Image features noted by integers. If None (default), all values.
- **`index`** (*`int or sequence of ints, optional`*) – Labels to include in output. If None (default), all values where non-zero `label_image` are used.

  The `index` argument only works when `label_image` is specified.

Returns **maxima** – Maxima of `image` over the `index` selected regions from `label_image`.

Return type ndarray

`dask_image.ndmeasure.`**`maximum_position`**(*image*, *label_image=None*, *index=None*)

Find the positions of maxima over an image at specified subregions.

For each region specified by `label_image`, the position of the maximum value of `image` within the region is returned.

Parameters

- **`image`** (*`ndarray`*) – N-D image data
- **`label_image`** (*`ndarray, optional`*) – Image features noted by integers. If None (default), all values.
- **`index`** (*`int or sequence of ints, optional`*) – Labels to include in output. If None (default), all values where non-zero `label_image` are used.

  The `index` argument only works when `label_image` is specified.

Returns **maxima_positions** – Maxima positions of `image` over the `index` selected regions from `label_image`.

Return type ndarray

`dask_image.ndmeasure.`**`mean`**(*image*, *label_image=None*, *index=None*)

Find the mean over an image at specified subregions.

Parameters

- **`image`** (*`ndarray`*) – N-D image data
- **`label_image`** (*`ndarray, optional`*) – Image features noted by integers. If None (default), all values.
- **`index`** (*`int or sequence of ints, optional`*) – Labels to include in output. If None (default), all values where non-zero `label_image` are used.

  The `index` argument only works when `label_image` is specified.

Returns **means** – Mean of `image` over the `index` selected regions from `label_image`.

Return type ndarray

`dask_image.ndmeasure.`**`median`**(*image*, *label_image=None*, *index=None*)

Find the median over an image at specified subregions.

Parameters

- **`image`** (*`ndarray`*) – N-D image data
- **`label_image`** (*`ndarray, optional`*) – Image features noted by integers. If None (default), all values.
- **`index`** (*`int or sequence of ints, optional`*) – Labels to include in output. If None (default), all values where non-zero `label_image` are used.

  The `index` argument only works when `label_image` is specified.

Returns **medians** – Median of `image` over the `index` selected regions from `label_image`.

Return type ndarray

`dask_image.ndmeasure.`**`minimum`**(*image*, *label_image=None*, *index=None*)

Find the minima over an image at specified subregions.

Parameters

- **`image`** (*`ndarray`*) – N-D image data

- **label_image** (*ndarray, optional*) – Image features noted by integers. If None (default), all values.
- **index** (*int or sequence of ints, optional*) – Labels to include in output. If None (default), all values where non-zero `label_image` are used.

  The `index` argument only works when `label_image` is specified.

Returns **minima** – Minima of `image` over the `index` selected regions from `label_image`.

Return type ndarray

`dask_image.ndmeasure.`**`minimum_position`**(*image*, *label_image=None*, *index=None*)

Find the positions of minima over an image at specified subregions.

Parameters

- **image** (*ndarray*) – N-D image data
- **label_image** (*ndarray, optional*) – Image features noted by integers. If None (default), all values.
- **index** (*int or sequence of ints, optional*) – Labels to include in output. If None (default), all values where non-zero `label_image` are used.

  The `index` argument only works when `label_image` is specified.

Returns **minima_positions** – Maxima positions of `image` over the `index` selected regions from `label_image`.

Return type ndarray

`dask_image.ndmeasure.`**`standard_deviation`**(*image*, *label_image=None*, *index=None*)

Find the standard deviation over an image at specified subregions.

Parameters

- **image** (*ndarray*) – N-D image data
- **label_image** (*ndarray, optional*) – Image features noted by integers. If None (default), all values.
- **index** (*int or sequence of ints, optional*) – Labels to include in output. If None (default), all values where non-zero `label_image` are used.

  The `index` argument only works when `label_image` is specified.

Returns **standard_deviation** – Standard deviation of `image` over the `index` selected regions from `label_image`.

Return type ndarray

`dask_image.ndmeasure.`**`sum`**(*image*, *label_image=None*, *index=None*)

DEPRECATED FUNCTION. Use *sum_labels* instead.

`dask_image.ndmeasure.`**`sum_labels`**(*image*, *label_image=None*, *index=None*)

Find the sum of all pixels over specified subregions of an image.

Parameters

- **image** (*ndarray*) – N-D image data
- **label_image** (*ndarray, optional*) – Image features noted by integers. If None (default), all values.

- **`index`** (*`int or sequence of ints, optional`*) – Labels to include in output. If None (default), all values where non-zero `label_image` are used.

  The `index` argument only works when `label_image` is specified.

Returns **sum_lbl** – Sum of `image` over the `index` selected regions from `label_image`.

Return type ndarray

`dask_image.ndmeasure.`**`variance`**(*image*, *label_image=None*, *index=None*)

Find the variance over an image at specified subregions.

Parameters

- **`image`** (*`ndarray`*) – N-D image data
- **`label_image`** (*`ndarray, optional`*) – Image features noted by integers. If None (default), all values.
- **`index`** (*`int or sequence of ints, optional`*) – Labels to include in output. If None (default), all values where non-zero `label_image` are used.

  The `index` argument only works when `label_image` is specified.

Returns **variance** – Variance of `image` over the `index` selected regions from `label_image`.

Return type ndarray

dask_image.ndmorph package

`dask_image.ndmorph.`**`binary_closing`**(*image*, *structure=None*, *iterations=1*, *origin=0*, *mask=None*, *border_value=0*, *brute_force=False*)

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

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

Original docstring:

Multidimensional binary closing with the given structuring element.

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

Parameters

- **`image`** (*`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, 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. Only an integer of iterations is accepted.
- **`origin`** (*`int or tuple of ints, optional`*) – Placement of the filter, by default 0.
- **`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.

New in version 1.1.0.

- **border_value** (*int (cast to 0 or 1), optional*) – Value at the border in the output array.

  New in version 1.1.0.

- **brute_force** (*boolean, optional*) – Memory condition: if False, only the pixels whose value was changed in the last iteration are tracked as candidates to be updated in the current iteration; if true al pixels are considered as candidates for update, regardless of what happened in the previous iteration. False by default.

  New in version 1.1.0.

Returns **binary_closing** – Closing of the image 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 image 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

`dask_image.ndmorph.`**`binary_dilation`**(*image*, *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:

Multidimensional binary dilation with the given structuring element.

Parameters

- **image** (*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, 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. Only an integer of iterations is accepted.
- **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.

- **brute_force** (*boolean, optional*) – Memory condition: if False, only the pixels whose value was changed in the last iteration are tracked as candidates to be updated (dilated) in the current iteration; if True all pixels are considered as candidates for dilation, regardless of what happened in the previous iteration. False by default.

Returns **binary_dilation** – Dilation of the image 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

`dask_image.ndmorph.`**`binary_erosion`**(*image, 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:

Multidimensional binary erosion with a given structuring element.

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

Parameters

- **image** (*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, 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.
- **brute_force** (*boolean, optional*) – Memory condition: if False, only the pixels whose value was changed in the last iteration are tracked as candidates to be updated (eroded) in the current iteration; if True all pixels are considered as candidates for erosion, regardless of what happened in the previous iteration. False by default.

Returns **binary_erosion** – Erosion of the image 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

`dask_image.ndmorph.`**`binary_opening`**`(`*image*, *structure=None*, *iterations=1*, *origin=0*, *mask=None*, *border_value=0*, *brute_force=False*`)`

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

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

Original docstring:

Multidimensional binary opening with the given structuring element.

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

Parameters

- **`image`** (*`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, 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. Only an integer of iterations is accepted.
- **`origin`** (*`int or tuple of ints, optional`*) – Placement of the filter, by default 0.
- **`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.

  New in version 1.1.0.
- **`border_value`** (*`int (cast to 0 or 1), optional`*) – Value at the border in the output array.

  New in version 1.1.0.
- **`brute_force`** (*`boolean, optional`*) – Memory condition: if False, only the pixels whose value was changed in the last iteration are tracked as candidates to be updated in the current iteration; if true all pixels are considered as candidates for update, regardless of what happened in the previous iteration. False by default.

  New in version 1.1.0.

Returns **binary_opening** – Opening of the image 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 image 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

2.5 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:

2.5.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/dask/dask-image/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-image could always use more documentation, whether as part of the official dask-image docs, in docstrings, or even on the web in blog posts, articles, and such.

To build the documentation locally and preview your changes, first set up the conda environment for building the dask-image documentation:

```
$ conda env create -f environment_doc.yml
$ conda activate dask_image_doc_env
```

This conda environment contains dask-image and its dependencies, sphinx, and the dask-sphinx-theme.

Next, build the documentation with sphinx:

```
$ cd dask-image/docs
$ make html
```

Now you can preview the html documentation in your browser by opening the file: dask-image/docs/_build/html/index.html

Submit Feedback

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

2.5.2 Get Started!

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

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

   ```
   $ git clone git@github.com:your_name_here/dask-image.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-image-env python="<some version>"
   $ source activate dask-image-env
   $ 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_image 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.

2.5.3 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 all supported Python versions. Check CIs and make sure that the tests pass for all supported Python versions and platforms.

2.5.4 Running tests locally

To setup a local testing environment that matches the test environments we use for our continuous integration services, you can use the `.yml` conda environment files included in the `continuous_integration` folder in the dask-image repository.

There is a separate environment file for each supported Python version.

We will use conda to create an environment from a file (`conda env create -f name-of-environment-file.yml`).

Note: If you do not have Anaconda/miniconda installed, please follow these instructions.

```
$ conda env create -f continuous_integration/environment-latest.yml
```

This command will create a new conda test environment called `dask-image-testenv` with all required dependencies.

Now you can activate your new testing environment with:

```
.. code-block:: console
```

$ conda activate dask-image-testenv

Finally, install the development version of dask-image:

```
.. code-block:: console
```

$ pip install -e .

For local testing, please run `pytest` in the test environment:

```
.. code-block:: console
```

$ pytest

To run a subset of tests, for example all the tests for ndfourier:

```
$ pytest tests/test_dask_image/test_ndfourier
```

2.6 Credits

2.6.1 Development Lead

- John Kirkham @jakirkham

2.6.2 Contributors

See the full list of contributors here

2.7 History

2.7.1 2021.12.0

We're pleased to announce the release of dask-image 2021.12.0!

Highlights

The major highlights of this release include the introduction of new featurees for `find_objects` and spline filters. We have also moved to using CalVer (calendar version numbers) to match the main Dask project.

New Features

- Find objects bounding boxes (#240)
- Add spline_filter and spline_filter1d (#215)

Improvements

- ENH: add remaining kwargs to binary_closing and binary_opening (#221)
- ndfourier: support n > 0 (for rfft) and improve performance (#222)
- affine_transform: increased shape of required input array slices (#216)

Bug Fixes

- BUG: add missing import of warnings in dask_image.ndmeasure (#224)
- Fix wrap bug in ndfilters convolve and correlate (#243)

- Upgrade for compatibility with latest dask release (#241)

Test infrastructure

- GitHub actions testing (#188)
- Set up gpuCI testing on PRs (#248)
- Remove *RAPIDS_VER* axis, bump *CUDA_VER* in gpuCI matrix (#249)

Documentation updates

- Code style cleanup (#227)
- Remove out of date email address, strip __author__ & __email__ (#225)
- Update release guide, Dask CalVer uses YYYY.MM.DD (#236)
- Update min python version in setup.py (#250)
- Use new Dask docs theme (#245)
- Docs: Add *find_objects* to the coverage table (#254)

Other Pull Requests

- Switch to CalVer (calendar versioning) (#233)

6 authors added to this release (alphabetical)

- anlavandier - @anlavandier
- Charles Blackmon-Luca - @charlesbluca
- Genevieve Buckley - @GenevieveBuckley
- Gregory R. Lee - @grlee77
- Jacob Tomlinson - @jacobtomlinson
- Marvin Albert - @m-albert

6 reviewers added to this release (alphabetical)

- anlavandier - @anlavandier
- Genevieve Buckley - @GenevieveBuckley
- Gregory R. Lee - @grlee77
- Jacob Tomlinson - @jacobtomlinson
- jakirkham - @jakirkham
- Marvin Albert - @m-albert

2.7.2 0.6.0 (2021-05-06)

We're pleased to announce the release of dask-image 0.6.0!

Highlights

The highlights of this release include GPU support for binary morphological functions, and improvements to the performance of `imread`.

Cupy version 9.0.0 or higher is required for GPU support of the `ndmorph` subpackage. Cupy version 7.7.0 or higher is required for GPU support of the `ndfilters` and `imread` subpackages.

New Features

- GPU support for ndmorph subpackage: binary morphological functions (#157)

Improvements

- Improve imread performance: reduced overhead of pim.open calls when reading from image sequence (#182)

Bug Fixes

- dask-image imread v0.5.0 not working with dask distributed Client & napari (#194)
- Not able to map actual image name with dask_image.imread (#200, fixed by #182)
- affine_transform: Remove inconsistencies with ndimage implementation #205

API Changes

- Add alias `gaussian` pointing to `gaussian_filter` (#193)

Other Pull Requests

- Change default branch from master to main (#185)
- Fix rst formatting in release_guide.rst (#186)

4 authors added to this release (alphabetical)

- Genevieve Buckley - @GenevieveBuckley
- Julia Signell - @jsignell
- KM Goh - @K-Monty
- Marvin Albert - @m-albert

2 reviewers added to this release (alphabetical)

- Genevieve Buckley - @GenevieveBuckley
- KM Goh - @K-Monty

2.7.3 0.5.0 (2021-02-01)

We're pleased to announce the release of dask-image 0.5.0!

Highlights

The biggest highlight of this release is our new affine transformation feature, contributed by Marvin Albert. The SciPy Japan sprint in November 2020 led to many improvements, and I'd like to recognise the hard work by Tetsuo Koyama and Kuya Takami. Special thanks go to everyone who joined us at the conference!

New Features

- Affine transformation feature added: from dask_image.ndinterp import affine_transform (#159)
- GPU support added for local_threshold with method='mean' (#158)
- Pathlib input now accepted for imread functions (#174)

Improvements

- Performance improvement for 'imread', we now use *da.map_blocks* instead of *da.concatenate* (#165)

Bug Fixes

- Fixed imread tests (add *contiguous=True* when saving test data with tifffile) (#164)
- FIXed scipy LooseVersion for sum_labels check (#176)

API Changes

- ‘sum’ is renamed to ‘sum_labels’ and a add deprecation warning added (#172)

Documentation improvements

- Add section Talks and Slides #163 (#169)
- Add link to SciPy Japan 2020 talk (#171)
- Add development guide to setup environment and run tests (#170)
- Update information in AUTHORS.rst (#167)

Maintenance

- Update dependencies in Read The Docs environment (#168)

6 authors added to this release (alphabetical)

- Fabian Chong - @feiming
- Genevieve Buckley - @GenevieveBuckley
- jakirkham - @jakirkham
- Kuya Takami - @ku-ya
- Marvin Albert - @m-albert
- Tetsuo Koyama - @tkoyama010

7 reviewers added to this release (alphabetical)

- Fabian Chong - @feiming
- Genevieve Buckley - @GenevieveBuckley
- Gregory R. Lee - @grlee77
- jakirkham - @jakirkham
- Juan Nunez-Iglesias - @jni
- Marvin Albert - @m-albert
- Tetsuo Koyama - @tkoyama010

2.7.4 0.4.0 (2020-09-02)

We’re pleased to announce the release of dask-image 0.4.0!

Highlights

The major highlight of this release is support for cupy GPU arrays for dask-image subpackages imread and ndfilters. Cupy version 7.7.0 or higher is required to use this functionality. GPU support for the remaining dask-image subpackages (ndmorph, ndfourier, and ndmeasure) will be rolled out at a later date, beginning with ndmorph.

We also have a new function, threshold_local, similar to the scikit-image local threshold function.

Lastly, we’ve made more improvements to the user documentation, which includes work by new contributor @abhisht51.

New Features

- GPU support for ndfilters & imread modules (#151)
- threshold_local function for dask-image ndfilters (#112)

Improvements

- Add function coverage table to the dask-image docs (#155)
- Developer documentation: release guide (#142)
- Use tifffile for testing instead of scikit-image (#145)

3 authors added to this release (alphabetical)

- Abhisht Singh - @abhisht51
- Genevieve Buckley - @GenevieveBuckley
- jakirkham - @jakirkham

2 reviewers added to this release (alphabetical)

- Genevieve Buckley - @GenevieveBuckley
- Juan Nunez-Iglesias - @jni

2.7.5 0.3.0 (2020-06-06)

We're pleased to announce the release of dask-image 0.3.0!

Highlights

- Python 3.8 is now supported (#131)
- Support for Python 2.7 and 3.5 has been dropped (#119) (#131)
- We have a dask-image quickstart guide (#108), available from the dask examples page: https://examples.dask.org/applications/image-processing.html

New Features

- Distributed labeling has been implemented (#94)
- Area measurement function added to dask_image.ndmeasure (#115)

Improvements

- Optimize out first *where* in *label* (#102)

Bug Fixes

- Bugfix in *center_of_mass* to correctly handle integer input arrays (#122)
- Test float cast in *_norm_args* (#105)
- Handle Dask's renaming of *atop* to *blockwise* (#98)

API Changes

- Rename the input argument to image in the ndimage functions (#117)
- Rename labels in ndmeasure function arguments (#126)

Support

- Update installation instructions so conda is the preferred method (#88)
- Add Python 3.7 to Travis CI (#89)
- Add instructions for building docs with sphinx to CONTRIBUTING.rst (#90)
- Sort Python 3.7 requirements (#91)
- Use double equals for exact package versions (#92)

- Use flake8 (#93)
- Note Python 3.7 support (#95)
- Fix the Travis MacOS builds (update XCode to version 9.4 and use matplotlib 'Agg' backend) (#113)

7 authors added to this release (alphabetical)

- Amir Khalighi - @akhalighi
- Elliana May - @Mause
- Genevieve Buckley - @GenevieveBuckley
- jakirkham - @jakirkham
- Jaromir Latal - @jermenkoo
- Juan Nunez-Iglesias - @jni
- timbo8 - @timbo8

2 reviewers added to this release (alphabetical)

- Genevieve Buckley - @GenevieveBuckley
- jakirkham - @jakirkham

2.7.6 0.2.0 (2018-10-10)

- Construct separate label masks in *labeled_comprehension* (#82)
- Use *full* to construct 1-D NumPy array (#83)
- Use NumPy's *ndindex* in *labeled_comprehension* (#81)
- Cleanup *test_labeled_comprehension_struct* (#80)
- Use 1-D structured array fields for position-based kernels in *ndmeasure* (#79)
- Rewrite *center_of_mass* using *labeled_comprehension* (#78)
- Adjust *extrema*'s internal structured type handling (#77)
- Test labeled_comprehension with object type (#76)
- Rewrite *histogram* to use *labeled_comprehension* (#75)
- Use labeled_comprehension directly in more function in ndmeasure (#74)
- Update mean's variables to match other functions (#73)
- Consolidate summation in *_ravel_shape_indices* (#72)
- Update HISTORY for 0.1.2 release (#71)
- Bump dask-sphinx-theme to 1.1.0 (#70)

2.7.7 0.1.2 (2018-09-17)

- Ensure *labeled_comprehension*'s *default* is 1D. (#69)
- Bump dask-sphinx-theme to 1.0.5. (#68)
- Use nout=2 in ndmeasure's label. (#67)
- Use custom kernel for extrema. (#61)
- Handle structured dtype in labeled_comprehension. (#66)
- Fixes for *_unravel_index*. (#65)
- Bump dask-sphinx-theme to 1.0.4. (#64)
- Unwrap some lines. (#63)
- Use dask-sphinx-theme. (#62)
- Refactor out *_unravel_index* function. (#60)
- Divide *sigma* by *-2*. (#59)
- Use Python 3's definition of division in Python 2. (#58)
- Force dtype of *prod* in *_ravel_shape_indices*. (#57)
- Drop vendored compatibility code. (#54)
- Drop vendored copy of indices and uses thereof. (#56)
- Drop duplicate utility tests from *ndmorph*. (#55)
- Refactor utility module for imread. (#53)
- Reuse *ndfilter* utility function in *ndmorph*. (#52)
- Cleanup freq_grid_i construction in _get_freq_grid. (#51)
- Use shared Python 2/3 compatibility module. (#50)
- Consolidate Python 2/3 compatibility code. (#49)
- Refactor Python 2/3 compatibility from imread. (#48)
- Perform *2 * pi* first in *_get_ang_freq_grid*. (#47)
- Ensure *J* is negated first in *fourier_shift*. (#46)
- Breakout common changes in fourier_gaussian. (#45)
- Use conda-forge badge. (#44)

2.7.8 0.1.1 (2018-08-31)

- Fix a bug in an ndmeasure test of an internal function.

2.7.9 0.1.0 (2018-08-31)

- First release on PyPI.
- Pulls in content from dask-image org.
- Supports reading of image files into Dask.
- Provides basic N-D filters with options to extend.
- Provides a few N-D Fourier filters.
- Provides a few N-D morphological filters.
- Provides a few N-D measurement functions for label images.
- Has 100% line coverage in test suite.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX