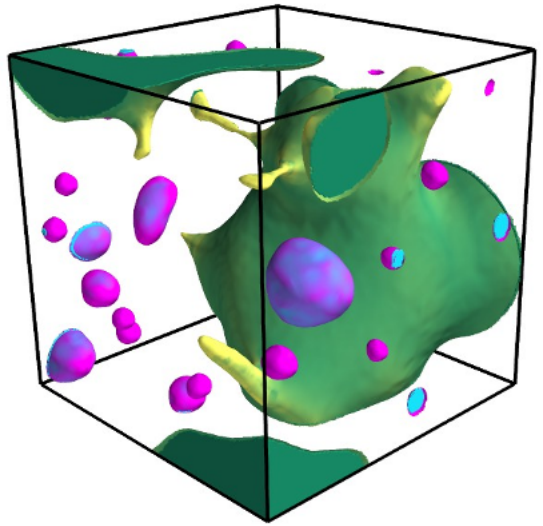
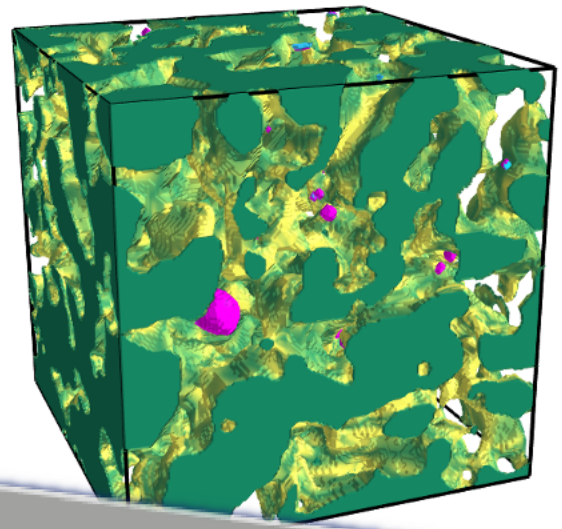
Image processing with `scikit-image`

Emmanuelle Gouillart
joint Unit CNRS/Saint-Gobain SVI

@EGouillart

`scikit-image` team



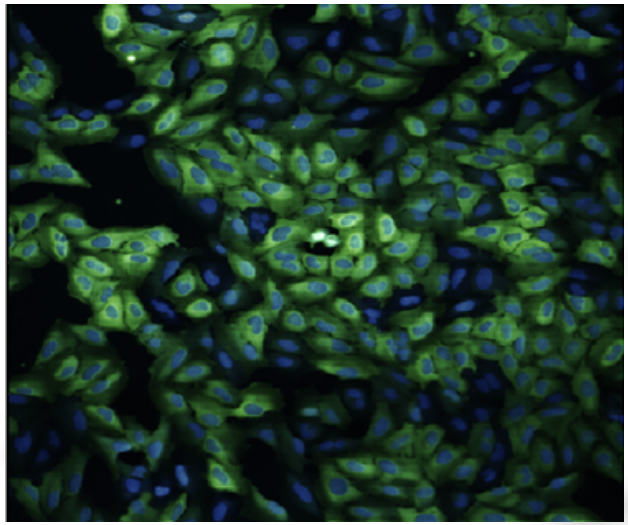


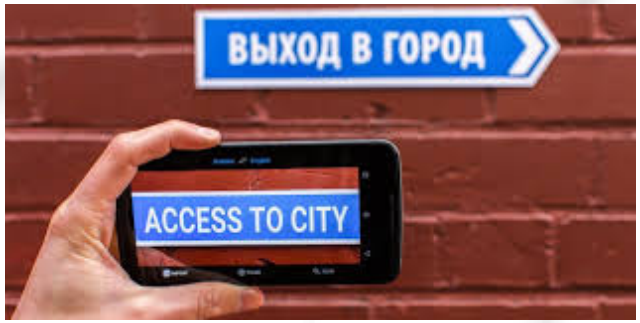
scikit-image
image processing in python

cnrs
dépasser les frontières

SAINT-GOBAIN

Image processing applications





What is scikit-image?

An **open-source** (BSD)

generic **image processing library**

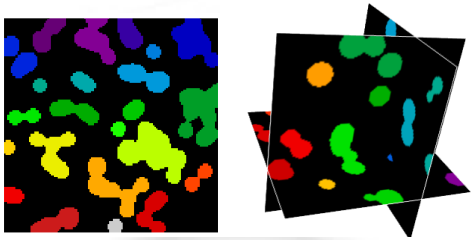
for the **Python** language
(and **NumPy** data arrays)

What is `scikit-image`?

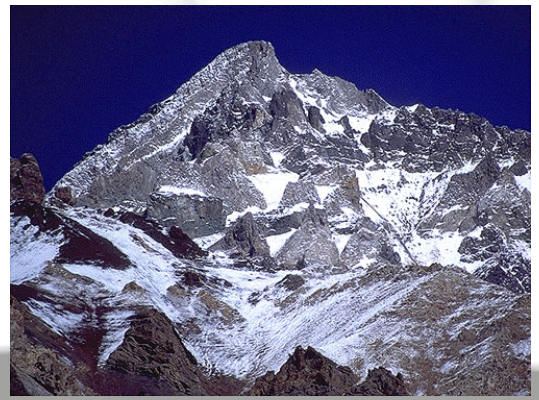
An **open-source** (BSD)

generic **image processing library**

for the **Python** language
(and **NumPy** data arrays)



for 2D & **3D** images



simple API & **gentle learning** curve

NumpPy: Python objects for numerical arrays

Multi-dimensional numerical data container (based on compiled code)

\+ utility functions to create/manipulate them

```
>>> a = np.random.random_integers(0, 1, (2, 2, 2))
>>> a
array([[[0, 1],
        [1, 0]],

       [[0, 0],
        [0, 1]]])
>>> a.shape, a.dtype
((2, 2, 2), dtype('int64'))
```

NumpPy: Python objects for numerical arrays

Multi-dimensional numerical data container (based on compiled code)
+ utility functions to create/manipulate them

```
>>> a = np.random.random_integers(0, 1, (2, 2, 2))
>>> a
array([[[0, 1],
        [1, 0]],

       [[0, 0],
        [0, 1]]])
>>> a.shape, a.dtype
((2, 2, 2), dtype('int64'))
```

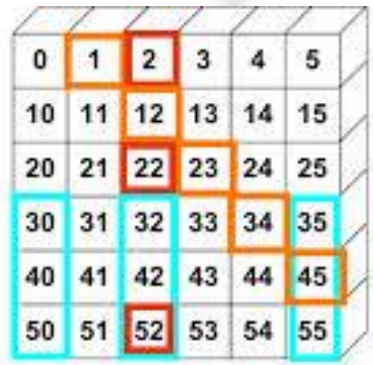
Efficient and versatile data access

indexing and slicing

```
>>> a[0,3:5]
array([3,4])

>>>-a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,22,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```

fancy indexing

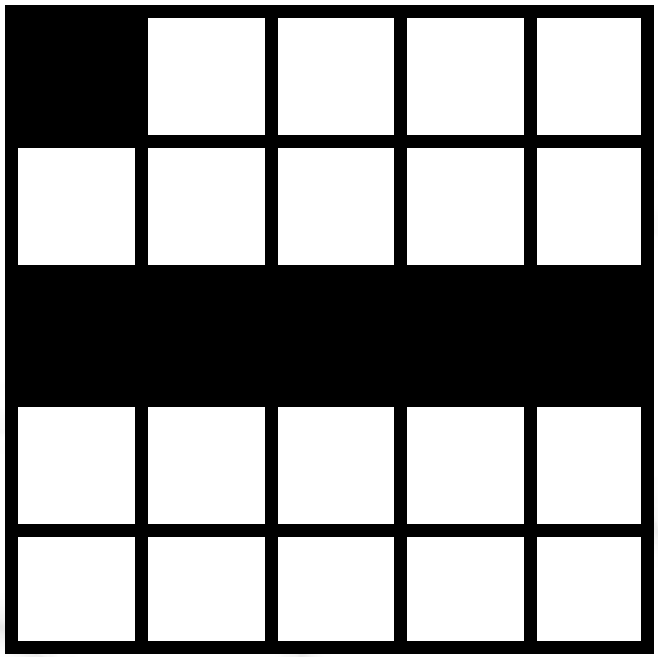
```
>>> a[(0,1,2,3,4),(1,2,3,4,5)]
array([ 1, 12, 23, 34, 45])

>>> a[3:,[0, 2, 5]]
array([[30, 32, 35]
       [40, 42, 45]])
       [50, 52, 55]])

>>> mask = array([1,0,1,0,0,1],
                 dtype=bool)
>>> a[mask,2]
array([2,22,52])
```



Manipulating images as numerical (numpy) arrays

- Pixels are arrays elements

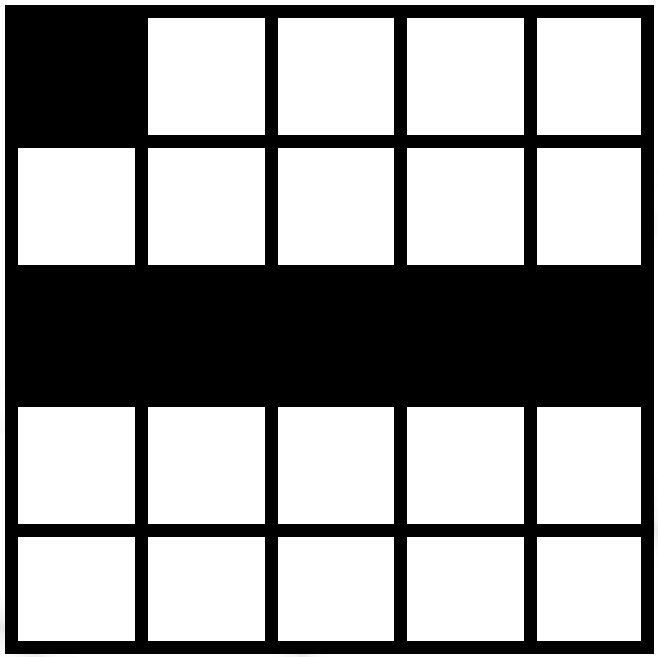
```
import numpy as np
image = np.ones((5, 5))
image[0, 0] = 0
image[2, :] = 0
```



Manipulating images as numerical (numpy) arrays

- Pixels are arrays elements

```
import numpy as np
image = np.ones((5, 5))
image[0, 0] = 0
image[2, :] = 0
```



```
>>> coffee.shape
(400, 600, 3)
>>> red_channel =
coffee[..., 0]
>>> image_3d =
np.ones((100, 100, 100))
```

NumPy-native: images as NumPy arrays

- NumPy arrays as arguments and outputs

```
>>> from skimage import io, filters
>>> camera_array = io.imread('camera_image.png')
>>> type(camera_array)
<type 'numpy.ndarray'>
>>> camera_array.dtype
dtype('uint8')
>>> filtered_array = filters.gaussian_filter(
    camera_array, sigma=5)
>>> type(filtered_array)
<type 'numpy.ndarray'>
>>> import matplotlib.pyplot as plt
>>> plt.imshow(filtered_array, cmap='gray')
```

The Scientific Python ecosystem

image file(s)

skimage.io

PyTables

Development environment

IP[y] spyder jupyter

NumPy array

1 75 ... 32
⋮
10 3 ... 2

scikit-image
image processing in python

skimage.filters
skimage.restoration
skimage.segmentation
⋮

machine learning

scikit learn

8 23 ... 36
⋮
1 13 ... 0

12 43 ... 6
⋮
66 ... 97 1

1 41 ... 98
⋮
38 75 ... 6

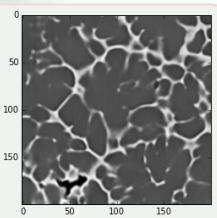
8 5 ... 73
⋮
78 98 ... 9

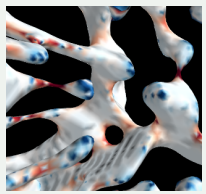
4 10 ... 55
⋮
10 3 ... 9

skimage.measure
skimage.feature

$[39 \ldots 5]$

matplotlib

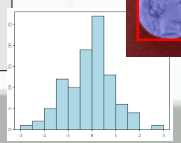


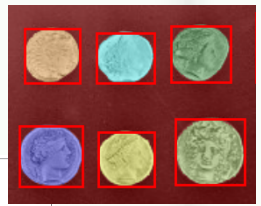


Mayavi

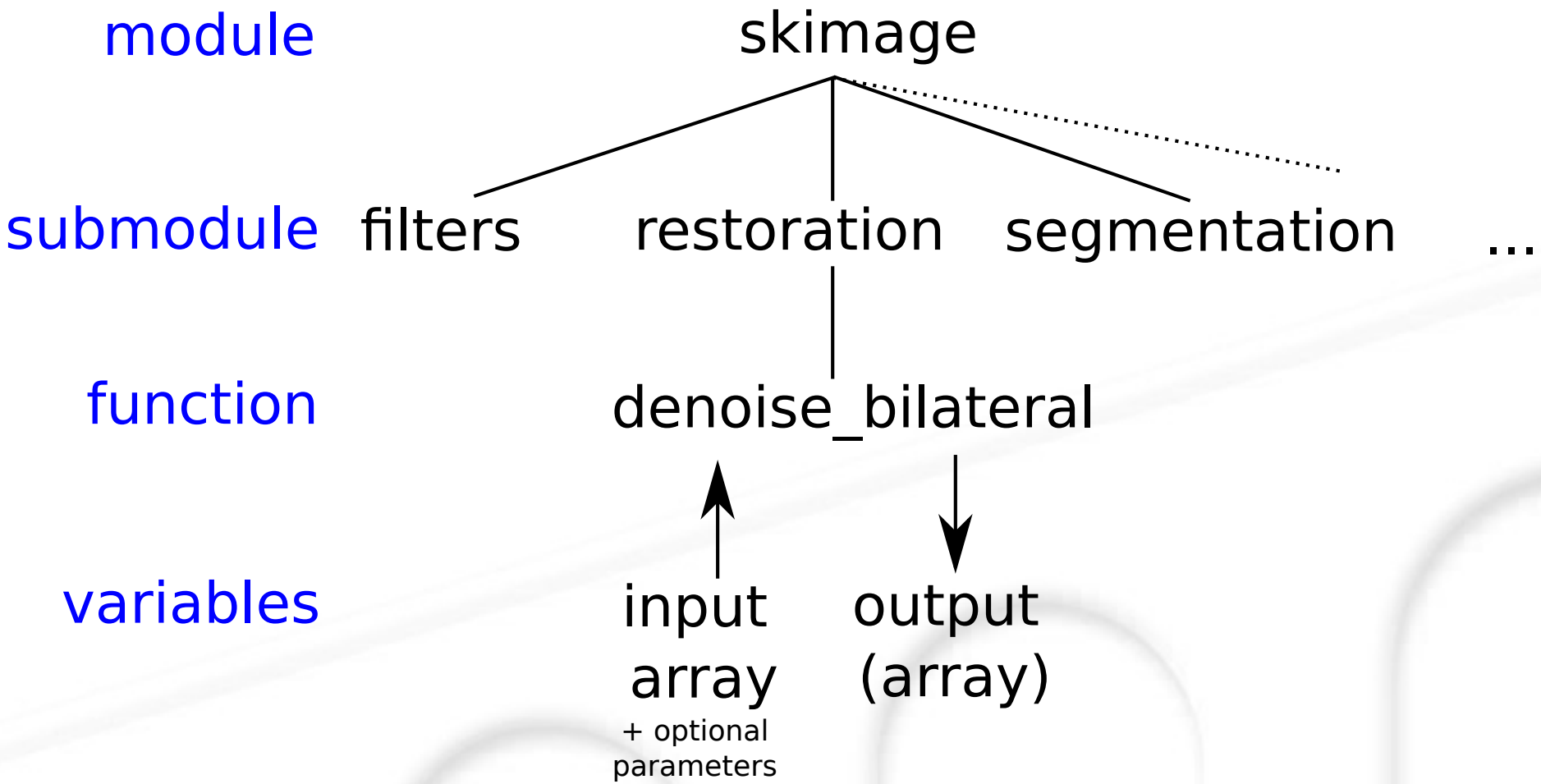
visualization

image properties
object properties

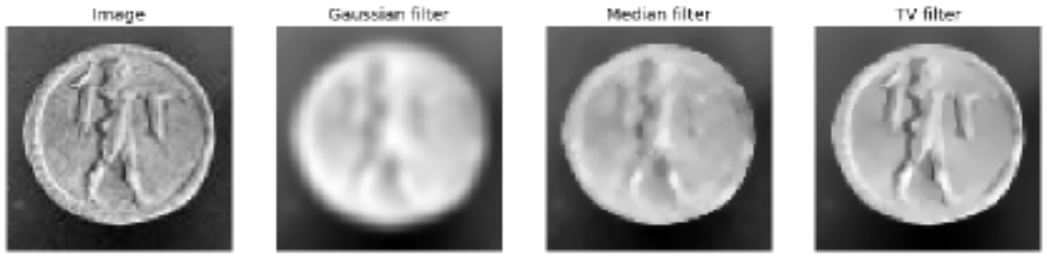


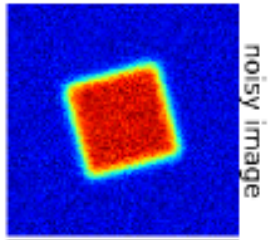


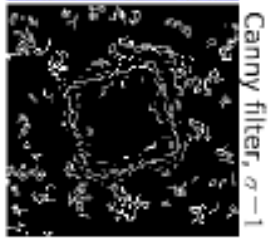
API of scikit-image

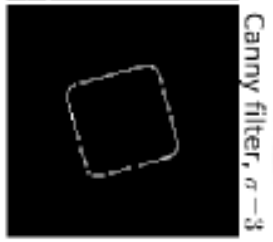


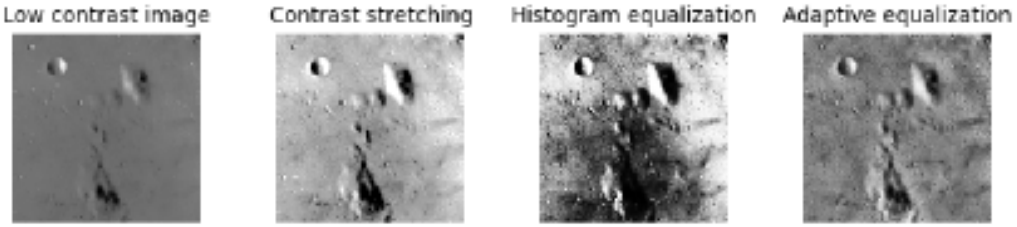
Filtering: transforming image data

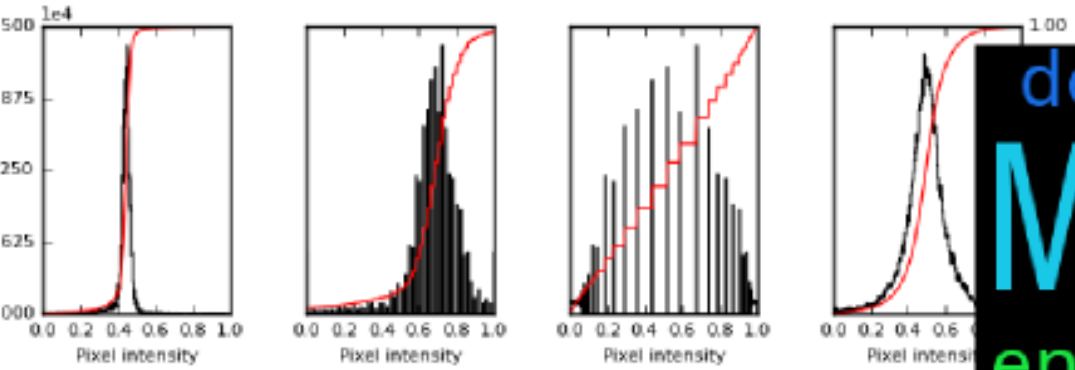


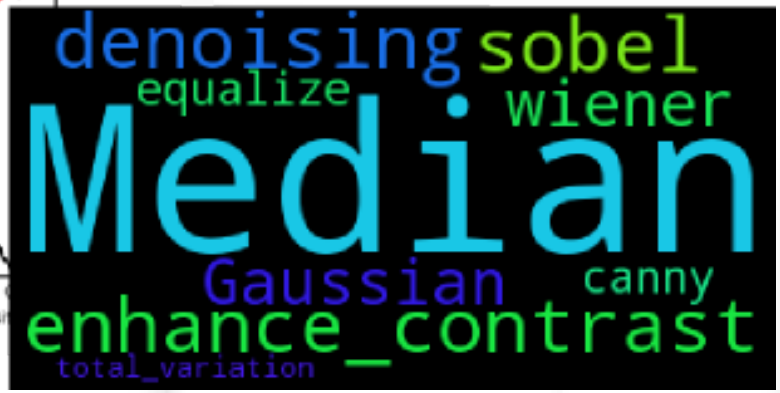






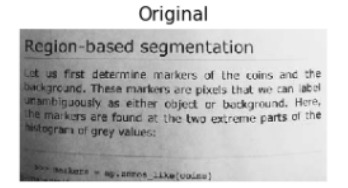


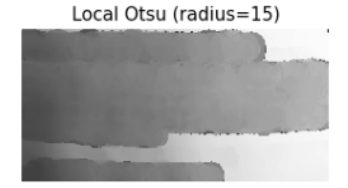


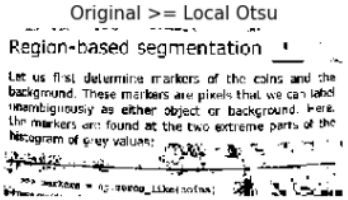


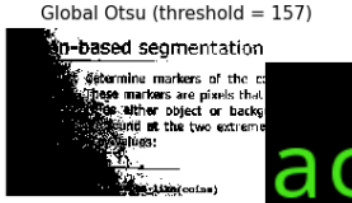
`skimage.filters`, `skimage.exposure`, `skimage.restoration`

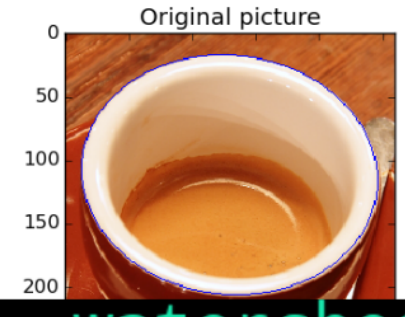
Segmentation: labelling regions

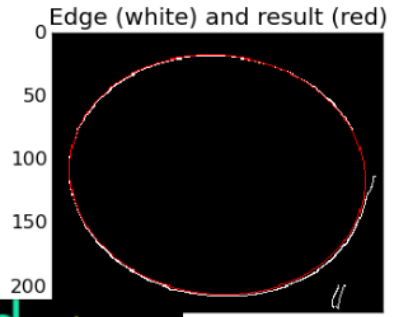




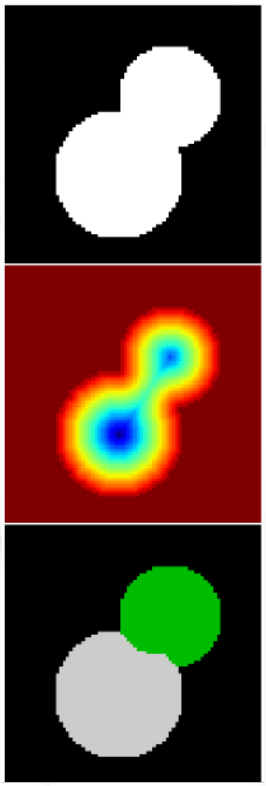








watershed otsu
activecontour
randomwalker
thresholding
superpixel

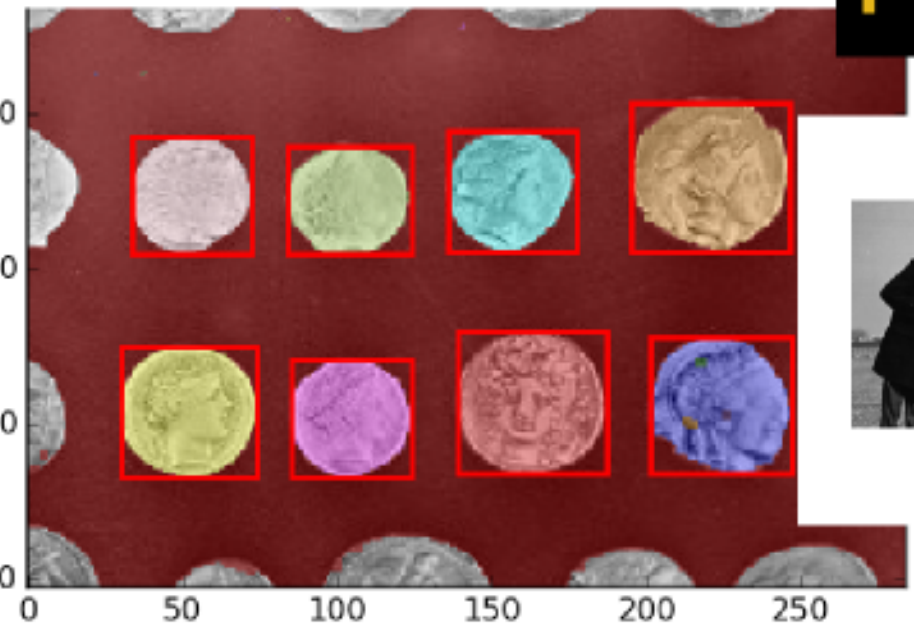


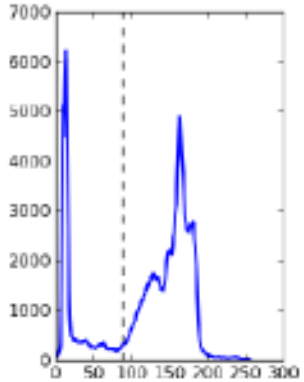
Felzenszwalbs's method

skimage.segmentation

Measures on images

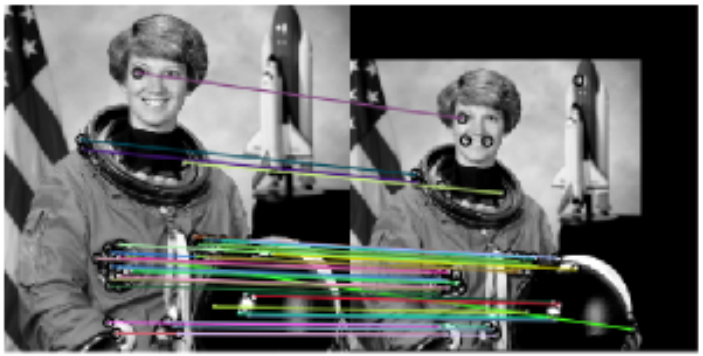
size label
measure
histogram
regionprops

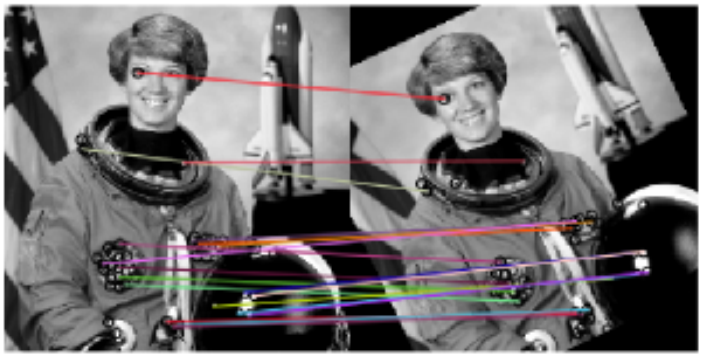


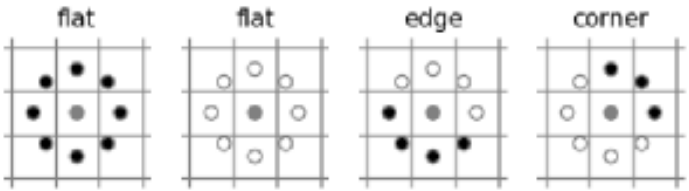


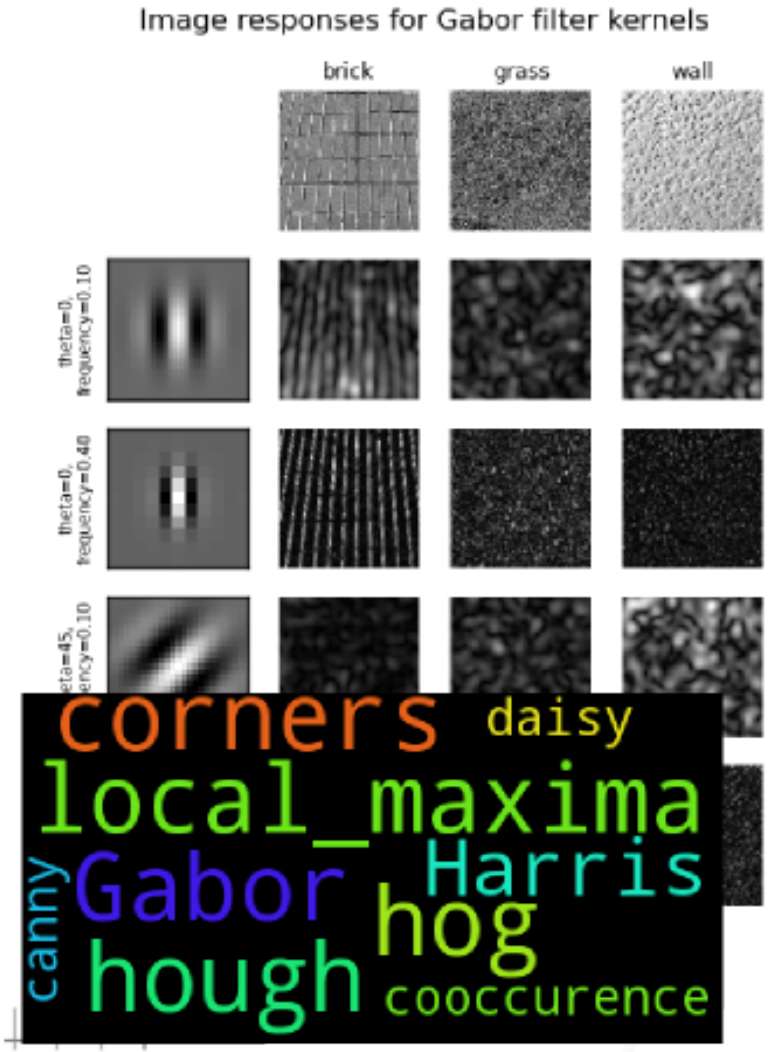
skimage.measure

Extracting features









`skimage.feature`, `skimage.filters`

Getting started: finding documentation

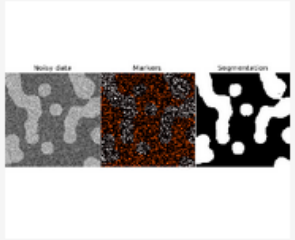
scikit-image
image processing in python

Home Download Gallery Documentation Source Search documentation ...

Image processing in Python

scikit-image is a collection of algorithms for image processing. It is available free of charge and free of restriction. We pride ourselves on high-quality, peer-reviewed code, written by an active community of volunteers.

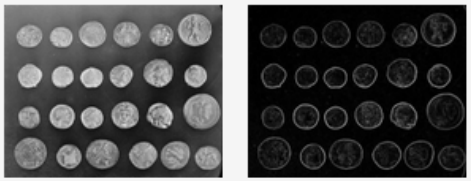
Download



Getting Started

Filtering an image with `scikit-image` is easy! For more examples, please visit our gallery.

```
from skimage import data, io, filter

image = data.coins() # or any NumPy arr
ay!
edges = filter.sobel(image)
io.imshow(edges)
io.show()
```



If you find this project useful, please cite: [BiBTeX]

Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu and the scikit-image contributors. **scikit-image: Image processing in Python**. PeerJ 2:e453 (2014) http://dx.doi.org/10.7717/peerj.453

Stable
0.10.1 - June 2014
Download

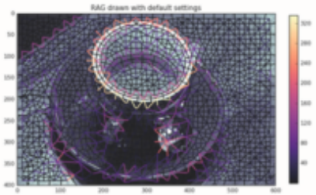
Development
pre-0.11
Download

+1 307 Star 522

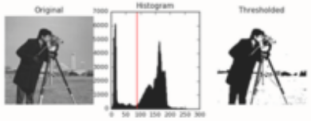
Links
- Issue tracker
- Mailing list
- Test results

Related Projects
- OpenCV
- Scikit-learn
- Mahotas
- SimpleCV
- Ilastik

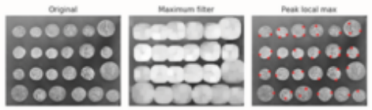
Gallery of examples



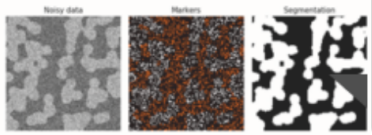
Drawing Region Adjacency Graphs (RAGs)



Thresholding

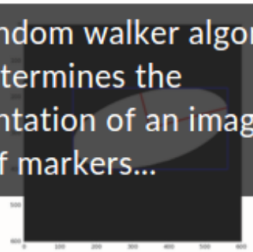


Finding local maxima

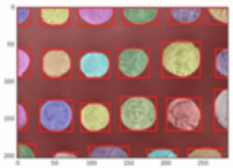


Random walker segmentation

The random walker algorithm [1]_ determines the segmentation of an image from a set of markers...

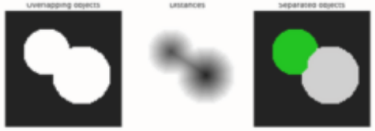


Measure region properties



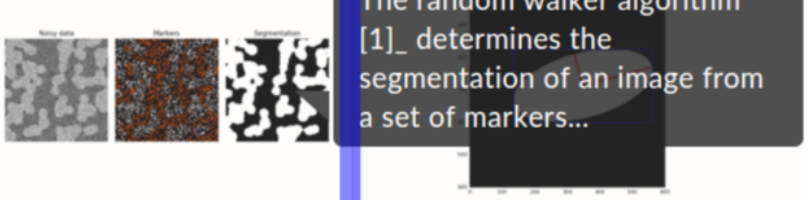
Label image regions

Comparison of segmentation and superpixel algorithms



Watershed segmentation

Markers for watershed transform



Random walker segmentation

Measure region properties

Label image regions

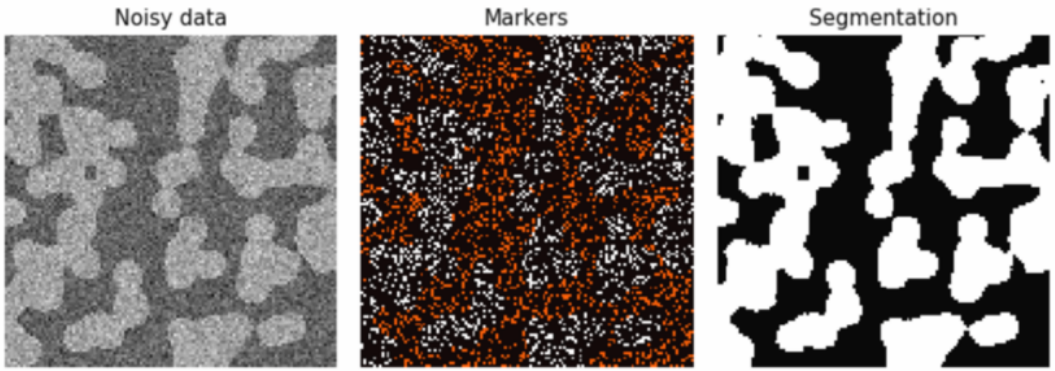
Code, figure and mini-tutorial

Random walker segmentation

The random walker algorithm [1] determines the segmentation of an image from a set of markers labeling several phases (2 or more). An anisotropic diffusion equation is solved with tracers initiated at the markers' position. The local diffusivity coefficient is greater if neighboring pixels have similar values, so that diffusion is difficult across high gradients. The label of each unknown pixel is attributed to the label of the known marker that has the highest probability to be reached first during this diffusion process.

In this example, two phases are clearly visible, but the data are too noisy to perform the segmentation from the histogram only. We determine markers of the two phases from the extreme tails of the histogram of gray values, and use the random walker for the segmentation.

[1] *Random walks for image segmentation*, Leo Grady, IEEE Trans. Pattern Anal. Mach. Intell. 2006 Nov; 28(11):1768-83




```
import numpy as np
import matplotlib.pyplot as plt

from skimage.segmentation import random_walker
from skimage.data import binary_blobs
import skimage

# Generate noisy synthetic data
data = skimage.img_as_float(binary_blobs(length=128, seed=1))
data += 0.35 * np.random.randn(*data.shape)
markers = np.zeros(data.shape, dtype=np.uint)
markers[data < -0.3] = 1
markers[data > 1.3] = 2

# Run random walker algorithm
labels = random_walker(data, markers, beta=10, mode='bf')

# Plot results
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(8, 3.2),
                                    sharex=True, sharey=True)
ax1.imshow(data, cmap='gray', interpolation='nearest')
ax1.axis('off')
ax1.set_adjustable('box-forced')
ax1.set_title('Noisy data')
ax2.imshow(markers, cmap='hot', interpolation='nearest')
ax2.axis('off')
ax2.set_adjustable('box-forced')
ax2.set_title('Markers')
ax3.imshow(labels, cmap='gray', interpolation='nearest')
ax3.axis('off')
ax3.set_adjustable('box-forced')
ax3.set_title('Segmentation')

fig.tight_layout()
plt.show()
```