image processing with scikit-image

1. Image Processing
   - about scikit-image
   - detecting edges
   - extracting regions
2. The Region Adjacency Graph
   - image analysis
   - image segmentation
   - the graph of an image
   - mean color segmentation
3. Open Source Computer Vision
   - about OpenCV

MCS 507 Lecture 29
Mathematical, Statistical and Scientific Software
Jan Verschelde, 18 March 2022

image processing with scikit-image

scikit-image

scikit-image is a collection of algorithms for image processing

latest stable release 0.16.2 October 2019

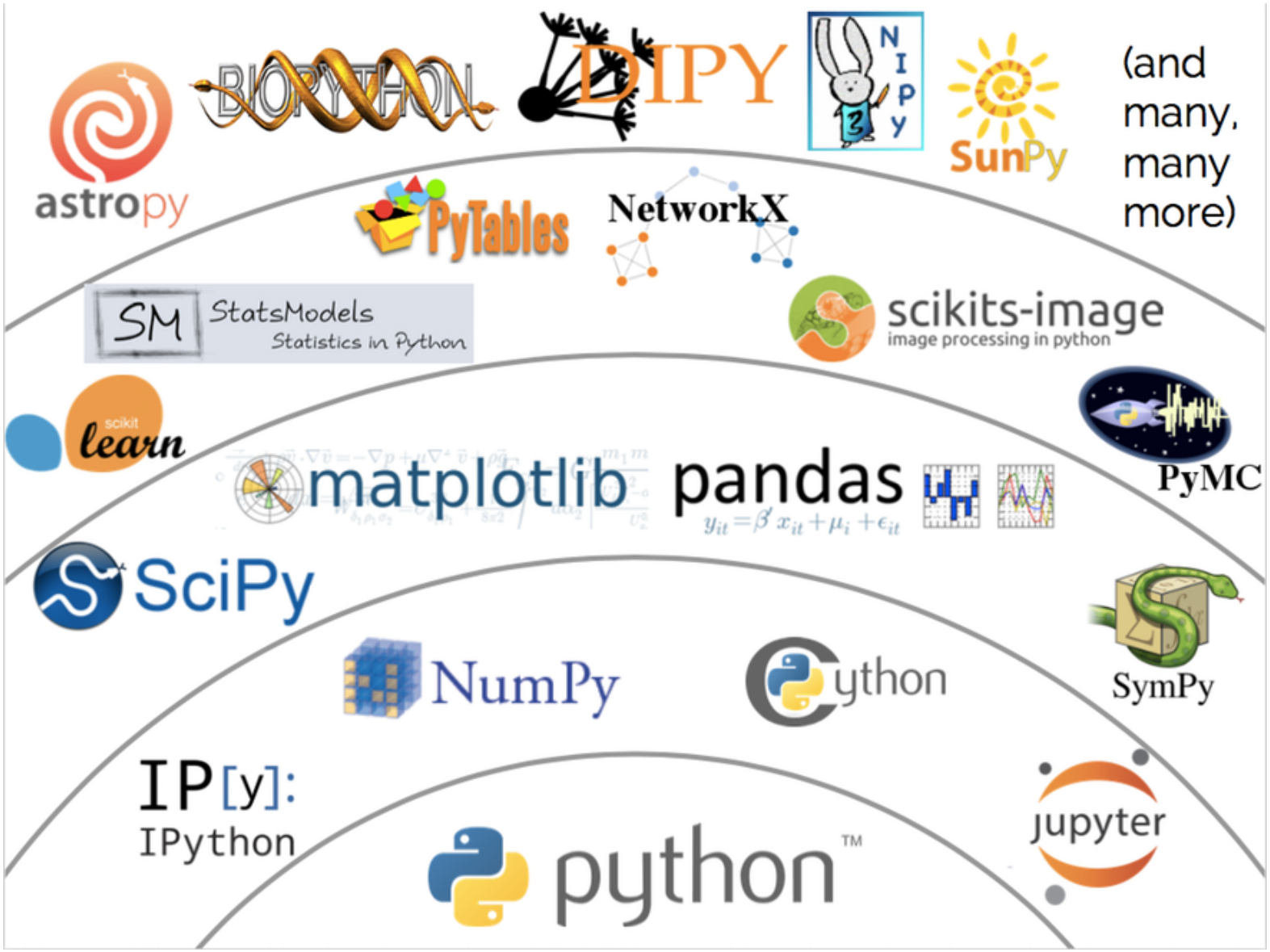
Released under the liberal Modified BSD open source license.

From `scikit-image.org`:
We pride ourselves on high-quality, peer-reviewed code,
written by an active community of volunteers.

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)
https://doi.org/10.7717/peerj.453

scikit-image in the stack

picture from the slides of Jake VanderPlas



getting started

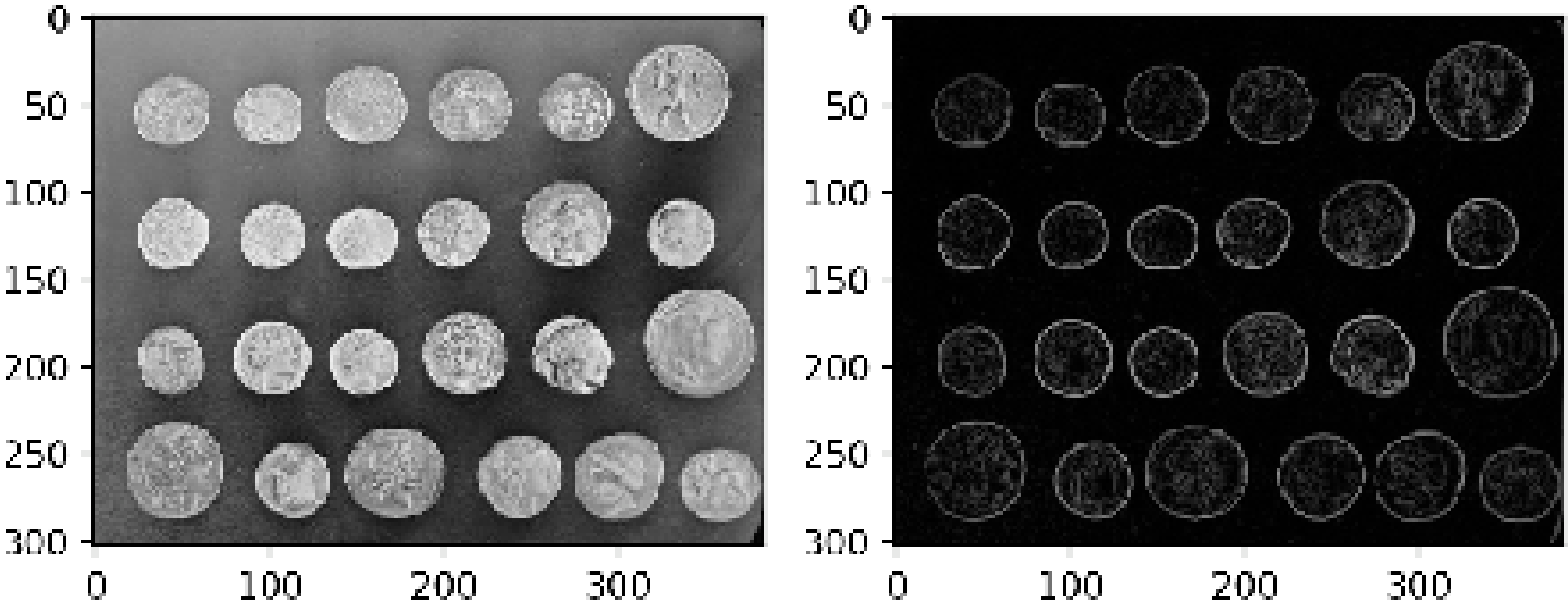
Filtering an image with scikit-image is easy!

```
from skimage import data, filters, io
from matplotlib import pyplot as plt

image = data.coins()
edges = filters.sobel(image)

fig = plt.figure()
ax = fig.add_subplot(121)
io.imshow(image)
ax = fig.add_subplot(122)
io.imshow(edges);
io.show()
```

computing edges of coins



Following the paper on scikit-image, we use the picture with ancient Roman coins from Pompeii, obtained from the Brooklyn Museum.

image processing with scikit-image

1. Image Processing
   - about scikit-image
   - detecting edges
   - extracting regions
2. The Region Adjacency Graph
   - image analysis
   - image segmentation
   - the graph of an image
   - mean color segmentation
3. Open Source Computer Vision
   - about OpenCV

Canny's algorithm

The Canny edge detector has 5 steps:

1. noise reduction with a Gaussian filter,
2. compute the intensity gradients,
3. non-maximum suppression,
4. double threshold,
5. edge tracking by hysteresis.

Many improvements have been made to the original algorithm.
For example, the gradient calculation is commonly done by
a 3-by-3 or 5-by-5 Sobel filter.

John Canny. **A Computational Approach To Edge Detection**,
IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.

the Canny edge detector in scikit-image

A couple of function calls suffice to compute edges.

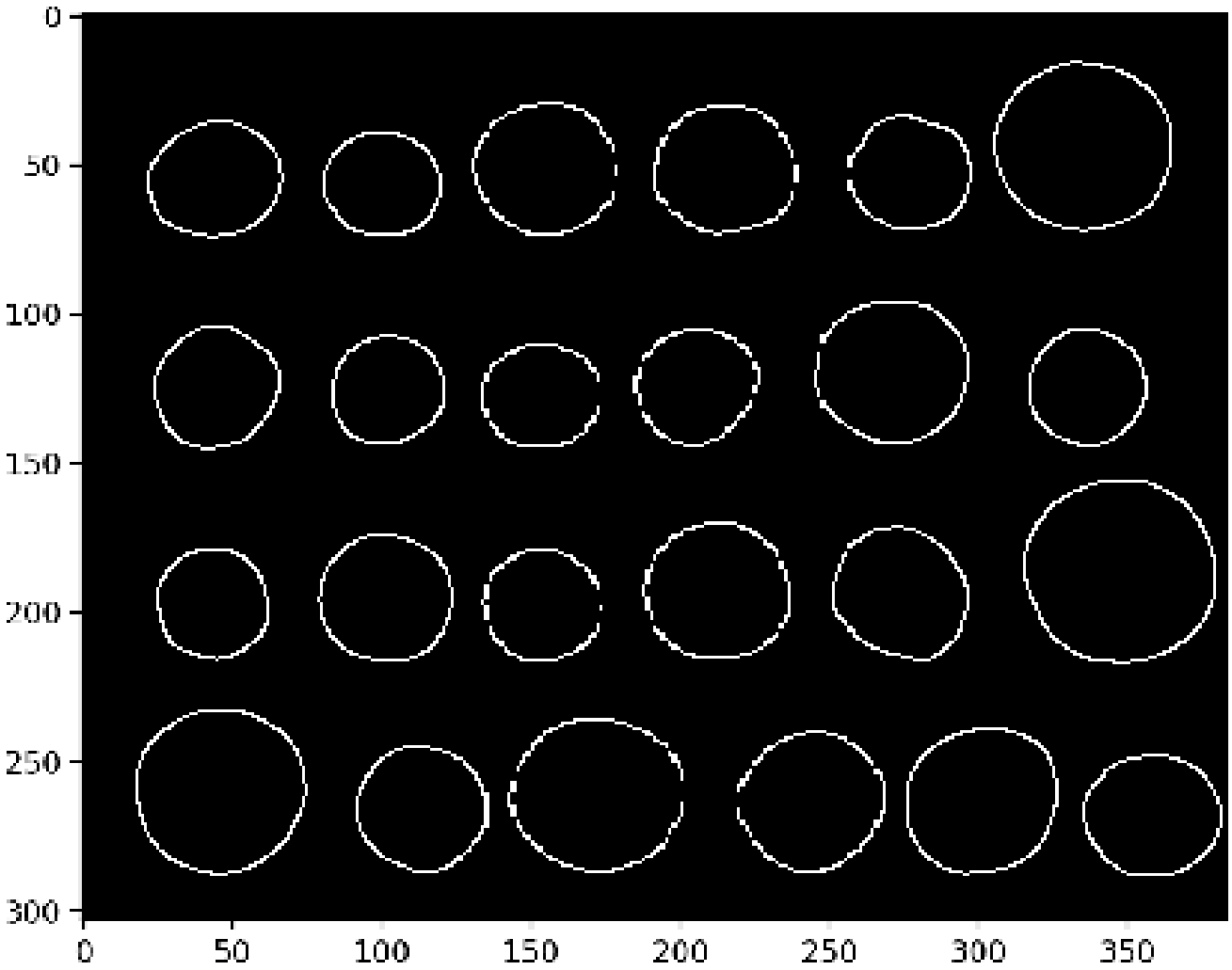
```
from skimage import data, feature, io
from matplotlib import pyplot as plt

image = data.coins()

edges = feature.canny(image, sigma=3,
   low_threshold=10, high_threshold=80)

io.imshow(edges, cmap=plt.cm.gray)
io.show()
```

output of the Canny edge detector



computing a histogram of the intensities

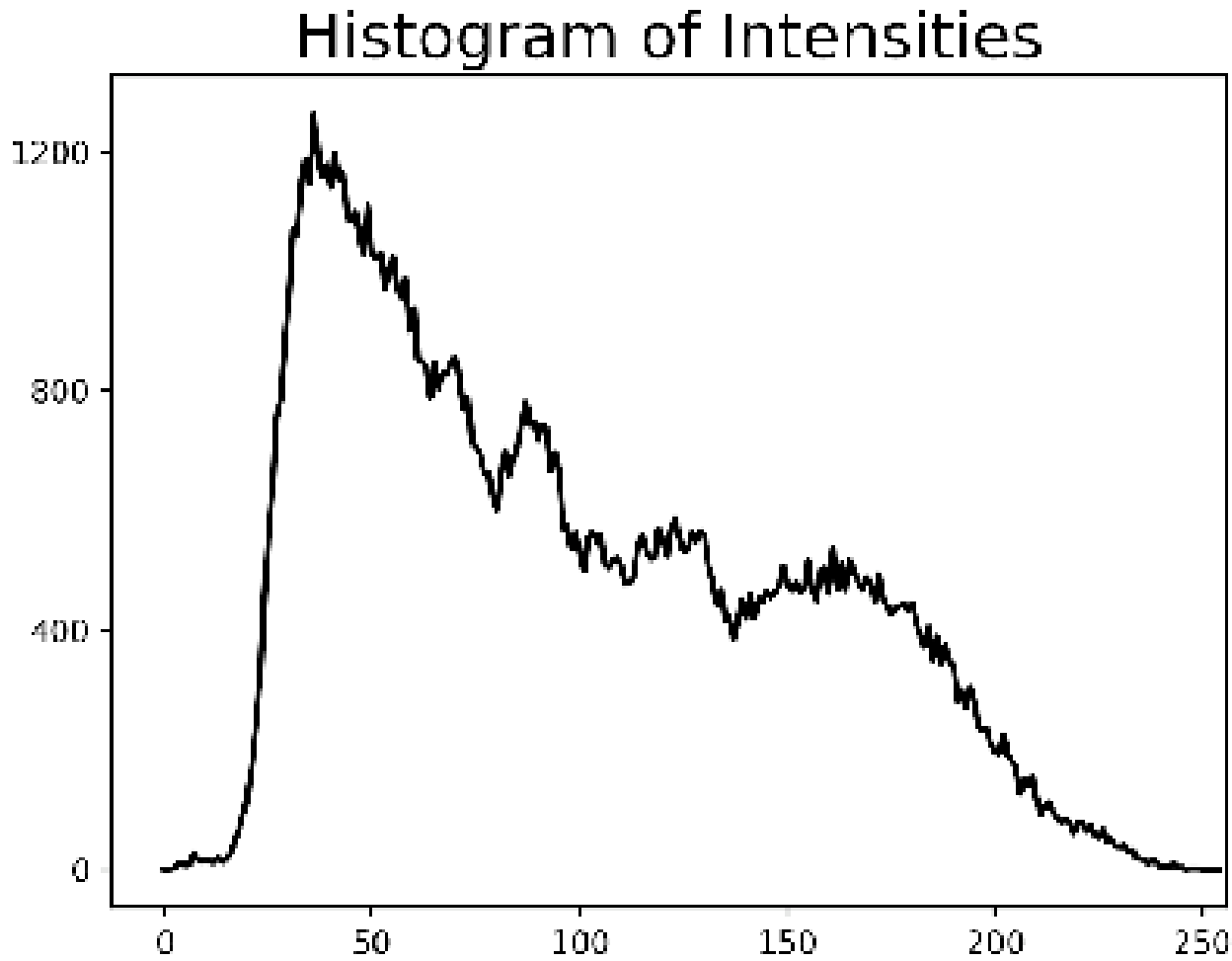
As images are numpy arrays, we make a histogram of the intensities.

```
import numpy as np
from skimage import data
from matplotlib import pyplot as plt

image = data.coins()

values, bins = np.histogram(image,
                            bins=np.arange(256))
fig = plt.figure()
ax = fig.add_subplot()
ax.plot(bins[:-1], values, lw=2, c='k')
ax.set_xlim(xmax=256)
ax.set_yticks([0, 400, 800, 1200])
ax.set_title('Histogram of Intensities', fontsize=20)
plt.show()
```

a histogram of the intensities



local maxima of intensities

Interesting features are local maxima.

```
import numpy as np
from skimage import data
from skimage.feature import peak_local_max
from matplotlib import pyplot as plt

image = data.coins()
coordinates = peak_local_max(image, min_distance=20)

fig = plt.figure()
ax = fig.add_subplot()
ax.imshow(image, cmap=plt.cm.gray)
ax.autoscale(False)
ax.plot(coordinates[:, 1], coordinates[:, 0], 'r.')
ax.set_title('peak local maxima', fontsize=24)
ax.axis('off')
plt.show()
```

red dots mark the local maxima

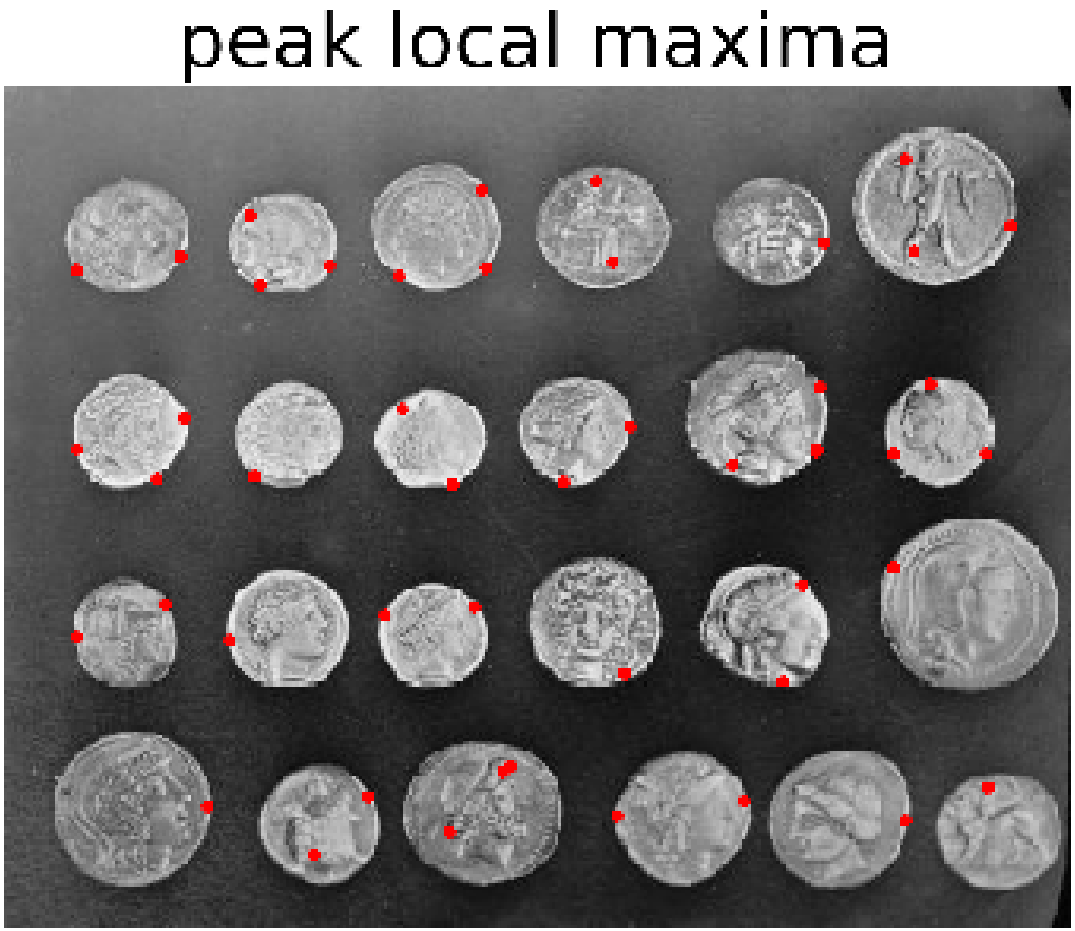


image processing with scikit-image

1. Image Processing
   - about scikit-image
   - detecting edges
   - extracting regions
2. The Region Adjacency Graph
   - image analysis
   - image segmentation
   - the graph of an image
   - mean color segmentation
3. Open Source Computer Vision
   - about OpenCV

image segmentation

To each coin we can attribute a label
and that label can be used to extract a sub picture,
after the application of the Canny edge detector.

```
from skimage import data, feature
from skimage.measure import regionprops
from skimage.morphology import label

image = data.coins()

edges = feature.canny(image, sigma=3,
    low_threshold=10, high_threshold=80)

label_image = label(edges)
```

drawing boxes

```
from matplotlib import pyplot as plt
import matplotlib.patches as mpatches

fig = plt.figure()
ax = fig.add_subplot()
ax.imshow(image, cmap=plt.cm.gray)
ax.set_title('labeled items', fontsize=24)
ax.axis('off')
for region in regionprops(label_image):
    # Draw rectangle around segmented coins.
    minr, minc, maxr, maxc = region.bbox
    rect = mpatches.Rectangle((minc, minr),
        maxc - minc, maxr - minr, fill=False,
        edgecolor='red', linewidth=2)
    ax.add_patch(rect)

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

extracting regions

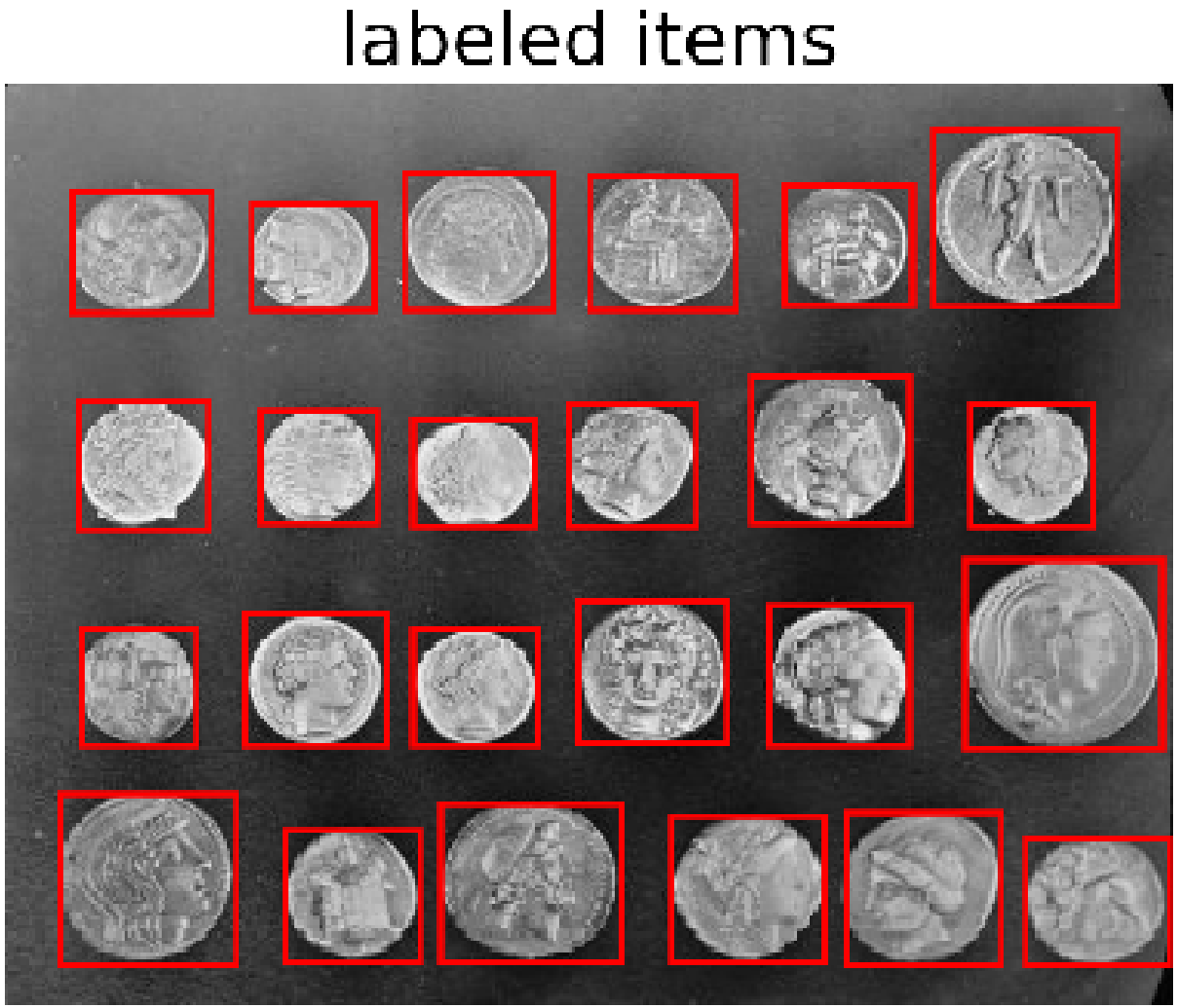


image processing with scikit-image

1. Image Processing
   - about scikit-image
   - detecting edges
   - extracting regions
2. The Region Adjacency Graph
   - image analysis
   - image segmentation
   - the graph of an image
   - mean color segmentation
3. Open Source Computer Vision
   - about OpenCV

the Region Adjacency Graph (RAG)

Although digital images are pixels, pictures consists of larger objects.

A common structure to represent these larger objects is the *Region Adjacency Graph*, or RAG.

- The nodes hold properties of each region in the image, and
- its links hold the spatial relationships between the regions.

Two nodes are linked whenever their corresponding regions touch each other in the input image.

Building a RAG is complicated, but possible

- with a few lines of codes using NetworkX, and
- a filter from the ndimage module of scipy.

segmenting out the tiger from the picture

Following chapter 3 of *Elegant SciPy* by Juan Nunez-Iglesias, Harriet Dashnow, and Stéfan van der Walt, published by O'Reilly Media, 2017; we use a picture from the Berkeley Segmentation Dataset (BSDS).

Our goal is to segment out the tiger of the picture.

an example of a geometric transformation

A geometric transformation changes the location of the pixels.

As an example, consider the `swirl`:

```
from skimage import io
from skimage.transform import swirl
tiger = io.imread('bsdstiger.jpg')
swirled = swirl(tiger, strength=6, radius=512)
io.imshow(swirled)
io.show()
```

a swirled tiger

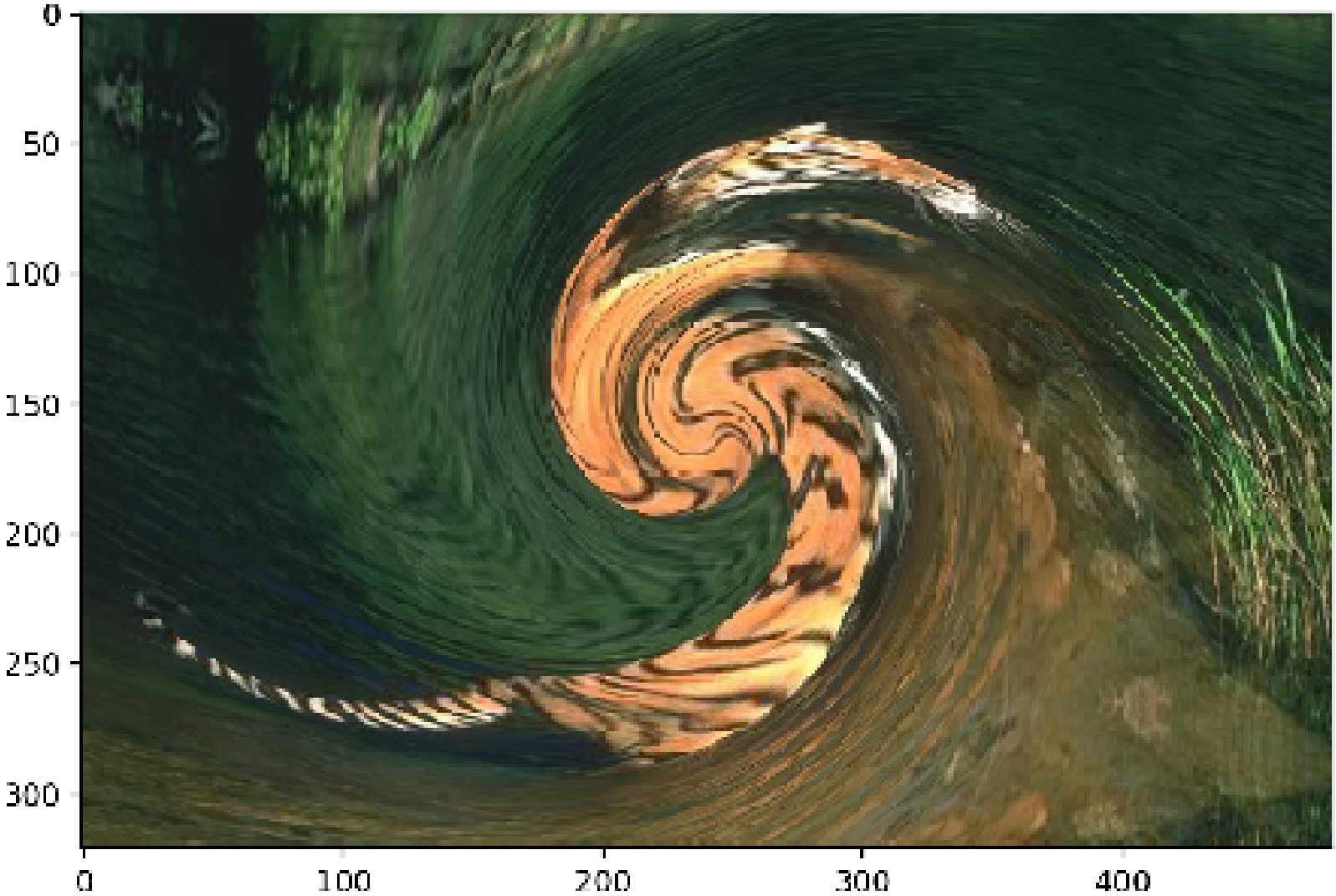


image processing with scikit-image

1. Image Processing
   - about scikit-image
   - detecting edges
   - extracting regions
2. The Region Adjacency Graph
   - image analysis
   - image segmentation
   - the graph of an image
   - mean color segmentation
3. Open Source Computer Vision
   - about OpenCV

simple linear iterative clustering

To segment an image, we apply a clustering algorithm, called Simple Linear Iterative Clustering (SLIC),

```
from skimage import io
from skimage import segmentation
from skimage import color
tiger = io.imread('bsdstiger.jpg')
seg = segmentation.slic(tiger, n_segments=30,
    compactness=40.0, enforce_connectivity=True,
    sigma=3)
io.imshow(color.label2rgb(seg, tiger))
io.show()
```

the output of `segmentation.slic`

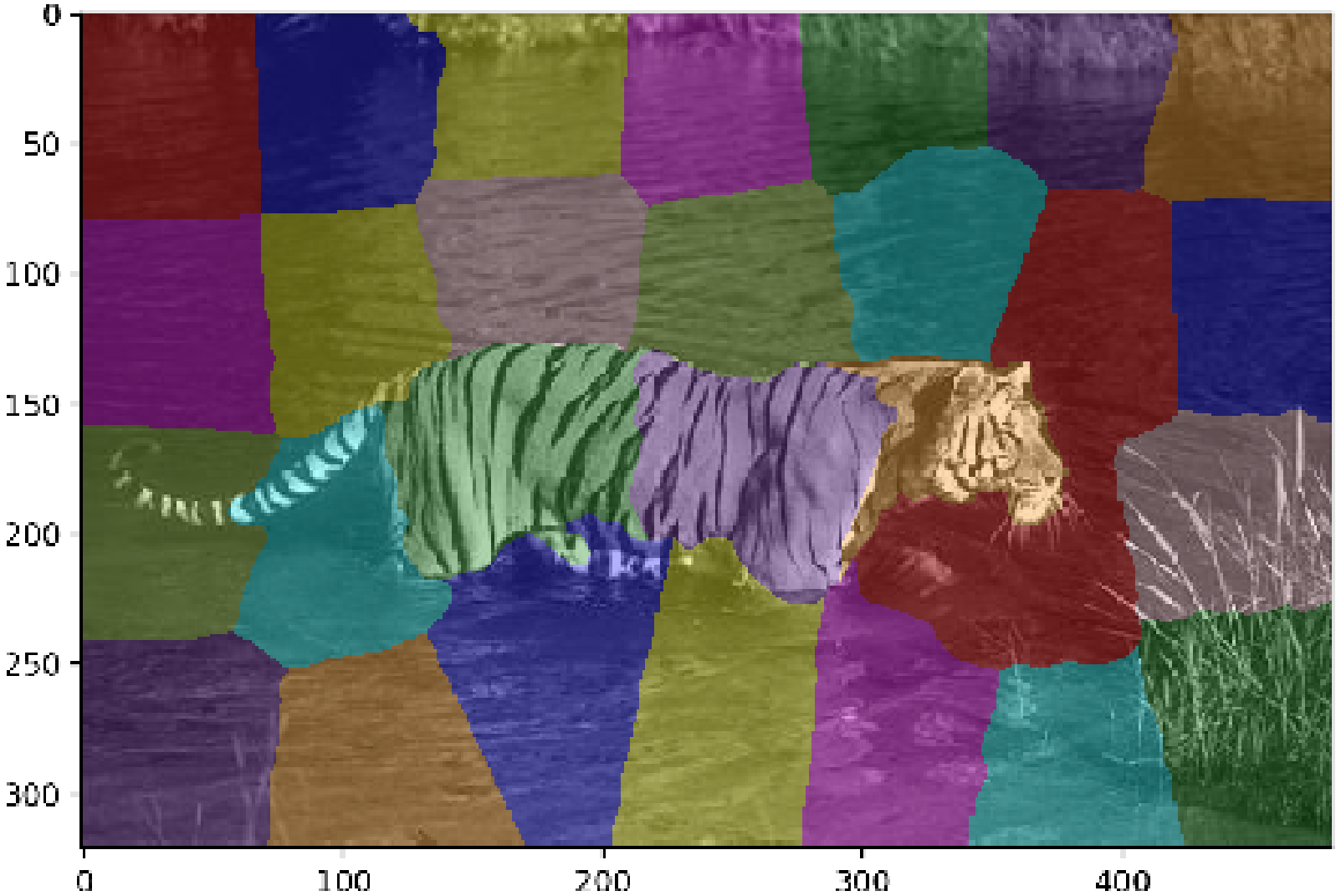


image processing with scikit-image

1. Image Processing
   - about scikit-image
   - detecting edges
   - extracting regions
2. The Region Adjacency Graph
   - image analysis
   - image segmentation
   - the graph of an image
   - mean color segmentation
3. Open Source Computer Vision
   - about OpenCV

constructing a graph

A graph is made after Simple Linear Iterative Clustering.

```
from skimage import io
from skimage import segmentation
from skimage.future import graph

tiger = io.imread('bsdstiger.jpg')
seg = segmentation.slic(tiger, n_segments=30,
    compactness=40.0, enforce_connectivity=True,
    sigma=3)

G = graph.rag_mean_color(tiger, seg)
graph.show_rag(seg, G, tiger)

io.show()
```

the region adjacency graph

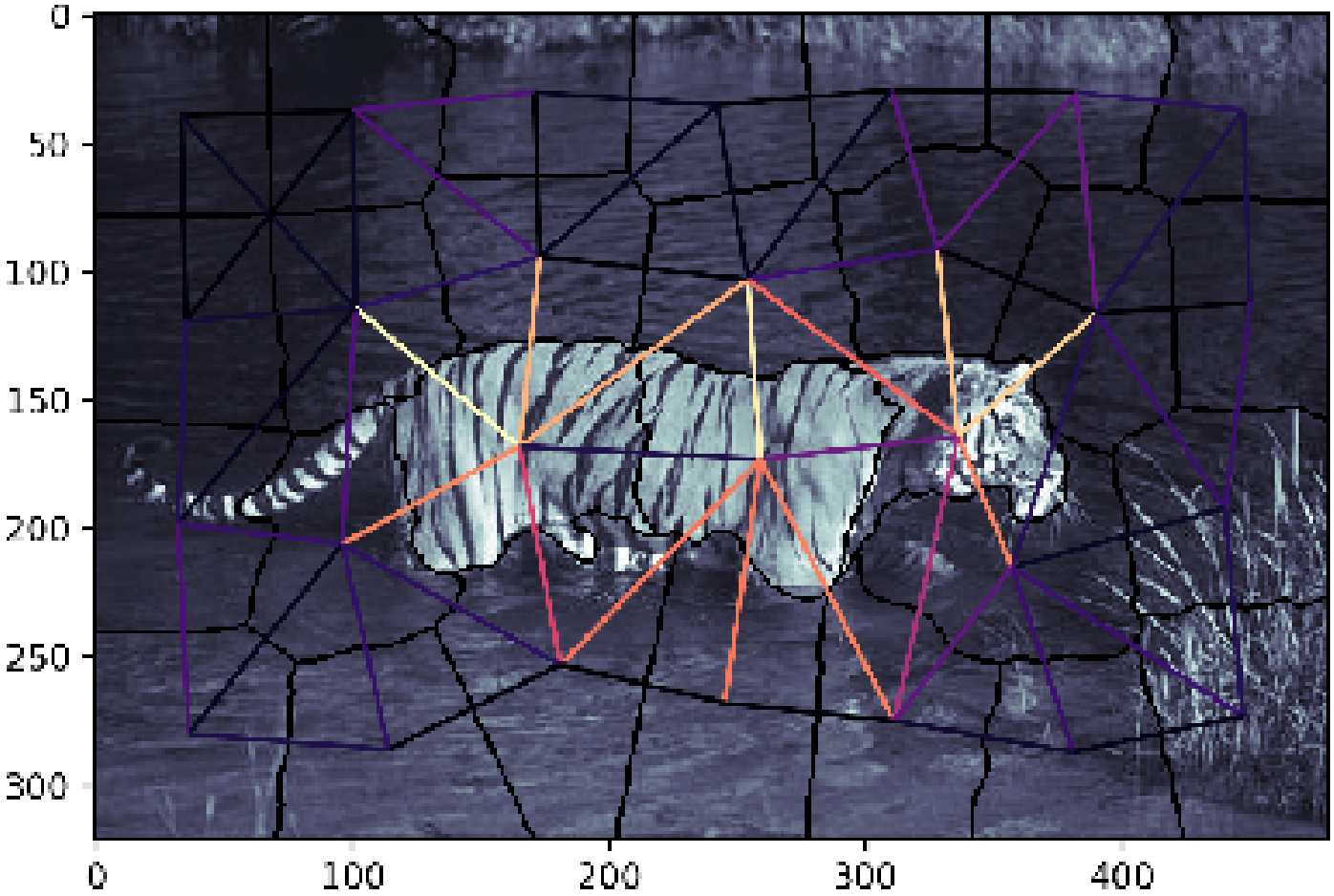


image processing with scikit-image

1. Image Processing
   - about scikit-image
   - detecting edges
   - extracting regions
2. The Region Adjacency Graph
   - image analysis
   - image segmentation
   - the graph of an image
   - mean color segmentation
3. Open Source Computer Vision
   - about OpenCV

building graphs from image regions

Following Chapter 3 of Elegant SciPy, we segment out the tiger.

Applying many tools of scientific Python,
we use numpy, ndimage, matplotlib, networkx, and skimage.

```
from skimage import io
from skimage import segmentation
from skimage import color
import networkx as nx
import numpy as np
from scipy import ndimage as ndi
from matplotlib import pyplot as plt
```

an auxiliary function to filter

```
def add_edge_filter(values, graph):
    """
    Auxiliary function used in the generic_filter.
    """
    center = values[len(values) // 2]
    for neighbor in values:
        if neighbor != center
            and not graph.has_edge(center, neighbor):
             graph.add_edge(center, neighbor)
    # float return value is unused but needed
    # by `generic_filter`
    return 0.0
```

computing the region adjacency graph

```
def build_rag(labels, image):
    """
    On input in labels is the output of segmentation.slic
    applied to the image.
    Returned is the region adjacency graph of the image.
    """
    g = nx.Graph()
    footprint = ndi.generate_binary_structure(labels.ndim,
        connectivity=1)
    _ = ndi.generic_filter(labels, add_edge_filter,
        footprint=footprint, mode='nearest',
        extra_arguments=(g,))
    for n in g:
        g.node[n]['total color'] = np.zeros(3, np.double)
        g.node[n]['pixel count'] = 0
    for index in np.ndindex(labels.shape):
        n = labels[index]
        g.node[n]['total color'] += image[index]
        g.node[n]['pixel count'] += 1
    return g
```

theshold the graph

```
def threshold_graph(g, t):
    """
    Edges in the graph with weight higher than t
    are removed from the graph.
    """
    to_remove = [(u, v) for (u, v, d)
                 in g.edges(data=True)
                 if d['weight'] > t]
    g.remove_edges_from(to_remove)
```

putting it all together

```
def main():
    """
    Does mean color segmentation.
    """
    tiger = io.imread('bsdstiger.jpg')
    seg = segmentation.slic(tiger, n_segments=30,
        compactness=40.0, enforce_connectivity=True,
        sigma=3)
    g = build_rag(seg, tiger)
    for n in g:
        node = g.node[n]
        node['mean']
            = node['total color'] / node['pixel count']
    for u, v in g.edges():
        d = g.node[u]['mean'] - g.node[v]['mean']
        g[u][v]['weight'] = np.linalg.norm(d)
    # every edge holds the difference between
    # the average color in each segment
```

the script continued

```
    threshold_graph(g, 80)
    map_array = np.zeros(np.max(seg) + 1, int)
    for i, segment
       in enumerate(nx.connected_components(g)):
        for initial in segment:
            map_array[int(initial)] = i
    segmented = map_array[seg]
    plt.imshow(color.label2rgb(segmented, tiger));
    plt.show()

main()
```

the tiger taken out of the picture

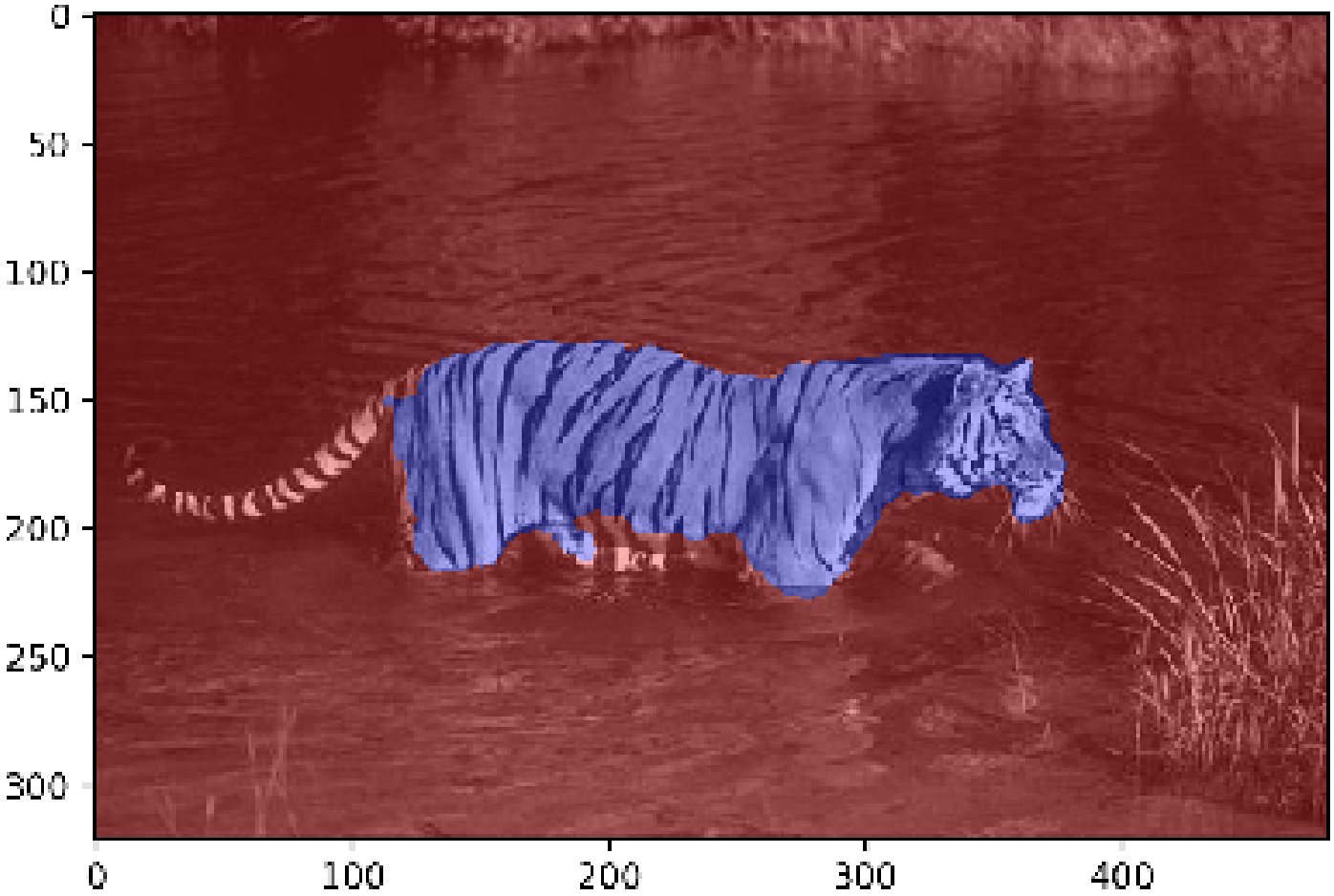


image processing with scikit-image

1. Image Processing
   - about scikit-image
   - detecting edges
   - extracting regions
2. The Region Adjacency Graph
   - image analysis
   - image segmentation
   - the graph of an image
   - mean color segmentation
3. Open Source Computer Vision
   - about OpenCV

about OpenCV

OpenCV (Open Source Computer Vision Library: http://opencv.org) is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms.

OpenCV has a modular structure, which means that the package includes several shared or static libraries. Available modules:

1. core - a compact module defining basic data structures,
2. imgproc - an image processing module
3. video - a video analysis module
4. calib3d - basic multiple-view geometry algorithms
5. features2d - salient feature detectors
6. objdetect - detection of objects
7. highgui - an easy-to-use interface to video capturing
8. gpu - GPU-accelerated algorithms

Exercises

1. Consider the `hubble_deep_field()` of the `data` of `skimage`. This picture displays stars in a Hubble Space Telescope image. Write a script to count automatically the number of stars.
2. Examine the region adjacency graph of the BSDS tiger picture. How many nodes? How many edges?
   What are the degrees of each node in the graph?
   Use `networkx` to draw the graph.
3. The `swirl` transform takes in the strength and radius as parameter. What are the largest values for strength and radius that result in the region adjacency graph of the original and the swirled picture being the same?