09-Connected-components

January 14, 2020

1 Connected component analysis (CCA)

1.1 What is and object?

[ ]:
```
import skimage.io
import skimage.filters
from matplotlib import pyplot as plt
import skimage.viewer
```

[ ]:
```
image = skimage.io.imread("../data/08-shapes.tif", as_gray=True)
smoothed = skimage.filters.gaussian(image, sigma=2.0)
viewer = skimage.viewer.ImageViewer(smoothed < 0.8)
# so what is an object here?
# How would you describe the pixels that belong to a single object?
# What would the numbers ideally be on such an image?
viewer.show()
```

1.1.1 Pixel Neighborhood

```
0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0
0 1 1 0 0 0 0 0
0 0 0 1 1 1 0 0
0 0 0 1 1 1 1 0
0 0 0 0 0 0 0 0
```

Orthogonal jumps:

- only jumps parallel to one of the axes allowed
- jump may only happen once for each axis

1-jump neigborhood

```
- 1 -
1 x 1
- 1 -
```

-> result from above: ~~~ 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 2 2 2 0 0 0 0 0 0 2 2 2 2 0 0 0 0 0 0 0 0 0 ~~~

2-jump neighborhood

```
2 1 2
1 x 1
2 1 2
```

-> result from above

```
0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0
0 1 1 0 0 0 0 0
0 0 0 1 1 1 0 0
0 0 0 1 1 1 1 0
0 0 0 0 0 0 0 0
```

1.1.2 Exercise:

Consider the following "image":

How many connected objects are there with

1. 1-jump neighborhood

- a. 1
- b. 5
- c. 2

2. 2-jump neighborhood

- a. 2
- b. 3
- c. 5

[ ]:
```
# 1. b - 5
# 2. a - 2
```

1.2 Connected Component Analysis

`skimage.measure.label`

- takes a binary image as an input (`False` is background, `True` foreground)

- CCA produces a new *labeled* image with one integer number for each pixel (where pixels with the same number, belong to the same object)
- pixels neighborhood is specified in means of orthogonal jumps

[ ]:
```
import skimage.measure

image = skimage.io.imread("../data/08-shapes.tif", as_gray=True)
smoothed = skimage.filters.gaussian(image, sigma=2.0)
binary = smoothed < 0.8
skimage.io.imshow(binary)
```

[ ]:
```
# Perform CCA on the mask
labeled_image = skimage.measure.label(binary, connectivity=2)
viewer = skimage.viewer.ImageViewer(labeled_image)
viewer.show()
# -> all black output
```

[ ]:
```
import numpy
print("dtype:", labeled_image.dtype)
print("min:", numpy.min(labeled_image))
print("max:", numpy.max(labeled_image))
```

[ ]:
```
# The `dtype` of `label_image` is `int64`.
# This means that values in this image range from `-2 ** 63` to `2 ** 63 - 1`.
# Those are really big numbers.
# From this available space we only use the range from `0` to `9`.
```

[ ]:
```
import skimage.color
labeled_rgb = skimage.color.label2rgb(labeled_image, bg_label=0)
viewer = skimage.viewer.ImageViewer(labeled_rgb)
viewer.show()
```

[ ]:
```
labeled_rgb = skimage.color.label2rgb(
    labeled_image, image, bg_label=0, alpha=0.6)
viewer = skimage.viewer.ImageViewer(labeled_rgb)
viewer.show()
```

1.2.1 Exercise: How many objects are in that image

Now, it is your turn to practice. Using the original `08-shapes.tif` image, add code to the following incomplete program, to count the number of objects in this image. How many objects do you count manually? How does changing the `sigma` and `threshold` values influence the result?

[ ]:
```
%load ../exercises/09-count-objects.py
```

```
[ ]: import numpy
     max_label = numpy.max(labeled_image)
     print("Found", max_label, "objects.")
     # Lowering the threshold will result in fewer objects.
     # The higher the threshold is set, the more objects are found.
     # More and more background noise gets picked up as objects.> > Lowering the␣
      ↪threshold will result in fewer objects.
     # The higher the threshold is set, the more objects are found.
     # More and more background noise gets picked up as objects.
```

1.3 Morphometrics

```
[ ]: image = skimage.io.imread("../data/08-shapes.tif", as_gray=True)
     smoothed = skimage.filters.gaussian(image, sigma=2.0)
     binary = smoothed < 0.8
     labeled_image = skimage.measure.label(binary)
```

```
[ ]: region_props = skimage.measure.regionprops(labeled_image)
```

```
[ ]: areas = [element["area"] for element in region_props]
     print(areas)
```

1.3.1 Exercise: Plot a histogram of the object area distribution

```
[ ]: %load ../exercises/09-count-objects.py
```

```
[ ]: region_props = skimage.measure.regionprops(labeled_image)
     areas = [element["area"] for element in region_props]
     plt.hist(areas)
```

1.3.2 Size filtering

```
[ ]: for i, area in enumerate(areas):
         if area < 10000:
             labeled_image[labeled_image == i] = 0

     skimage.io.imshow(skimage.color.label2rgb(labeled_image, image, bg_label=0))
```

1.3.3 Exercise: print count of large objects only

[ ]:
```
%load ../exercises/09-count-objects.py
```

[ ]:
```
large_areas = []
for area in areas:
    if area > 10000:
        large_areas.append(area)

print("Found", len(large_areas), "objects.")
```

[ ]:
```
solidities = []
for rp in region_props:
    if rp.area > 10000:
        solidities.append(rp)
plt.hist([x.solidity for x in solidities])
```

[ ]:
```
solidity_image = numpy.zeros_like(labeled_image).astype("float")
for rp in solidities:
    solidity_image[labeled_image == rp.label] = rp.solidity
plt.imshow(solidity_image)
```

Keypoints: - `skimage.measure.label` is used to generate objects. - We use `skimage.measure.regionprops` to measure properties of labelled objects. - Color objects according to feature values.