

# 03-skimage-images

January 14, 2020

## 1 Image representation in skimage

### 1.0.1 Reading images

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

```
[ ]: image = skimage.io.imread(fname="../data/chair.tif")
```

### 1.0.2 Displaying images

```
[ ]: import skimage.viewer
```

```
[ ]: # display image  
viewer = skimage.viewer.ImageViewer(image)  
viewer.show()
```

### 1.0.3 Saving images

```
[ ]: # save a new version in .png format  
skimage.io.imsave(fname="chair.png", arr=image)
```

### 1.0.4 Images are represented as NumPy arrays

```
[ ]: import numpy  
print("shape:", image.shape, "dtype:", image.dtype)
```

```
[ ]: print("min", numpy.min(image), "max", numpy.max(image), "mean", numpy.  
      ↳mean(image))
```

### 1.0.5 Exercise: Resize an image you took on your phone

- copy image from phone to the data folder
- load the image
- resize the image using `skimage.transform.resize()`
- save the resized image

```
[ ]: import skimage.transform

# read in image
image = skimage.io.imread(fname="../data/chair.tif")

# resize the image
new_shape = (image.shape[0] // 16, image.shape[1] // 16, image.shape[2])
small = skimage.transform.resize(image=image, output_shape=new_shape)

# write out image
skimage.io.imsave(fname="resized.tif", arr=small)
```

### 1.0.6 Manipulating pixels

#### ignore low intensity pixels

```
[ ]: import skimage.io
import skimage.viewer

# read input image, based on filename parameter
image = skimage.io.imread("../data/roots-bw.tif", as_gray=True)

# display original image
viewer = skimage.viewer.ImageViewer(image)
viewer.show()
```

```
[ ]: # keep only high-intensity pixels
image[image < 128] = 0

# display modified image
viewer = skimage.viewer.ImageViewer(image)
viewer.show()
```

### 1.0.7 Exercise: Keeping only low intensity pixels

- load the `../data/sudoku.png` image
- show the original image
- set pixels with values greater than 200 (white) to gray
- show the modified image

```
[ ]: import skimage.io
import skimage.viewer

# read input image, based on filename parameter
image = skimage.io.imread("../data/sudoku.png", as_gray=True)

# display original image
viewer = skimage.viewer.ImageViewer(image)
viewer.show()

# keep only high-intensity pixels
image[image > 128] = 64

# display modified image
viewer = skimage.viewer.ImageViewer(image)
viewer.show()
```

### 1.0.8 Converting color images to grayscale

```
[ ]: # read input image, based on filename parameter
image = skimage.io.imread("../data/chair.tif")

# display original image
viewer = skimage.viewer.ImageViewer(image)
viewer.show()

# convert to grayscale and display
gray_image = skimage.color.rgb2gray(image)
viewer = skimage.viewer.ImageViewer(gray_image)
viewer.show()
```

```
[ ]: # read input image, based on filename parameter
image = skimage.io.imread("../data/chair.tif", as_gray=True)

# display grayscale image
viewer = skimage.viewer.ImageViewer(image)
viewer.show()
```

### 1.0.9 Access via slicing

```
[ ]: image = skimage.io.imread("../data/roots.tif")

clip = image[1600:1900, 1700:2200, :]
```

```
# display grayscale image
viewer = skimage.viewer.ImageViewer(clip)
viewer.show()
```

```
[ ]: color = image[1800, 1900]
print(color)
image[1600:1900, 1700:2200] = color
viewer = skimage.viewer.ImageViewer(image)
viewer.show()
```

### 1.0.10 Exercise: Practicing with slices

- Load the code in the cell below and modify the program to
  - create,
  - display, and
  - save

a sub-image containing only the plant and its roots. Use ImageJ to determine the bounds of the area you will extract using slicing.

```
[ ]: %load ../exercises/03-RootSlice.py
```

```
[ ]: # WRITE YOUR CODE TO SELECT THE SUBIMAGE NAME clip HERE:
clip = image[:, 1400:2500, :]
```

```
[ ]: # WRITE YOUR CODE TO SAVE clip HERE
skimage.io.imsave("roots-clip.tif", arr=clip)
```

### 1.0.11 Exercise: Metadata

Explore how skimage handles metadata.

- Open the maize-roots.tif image in the data folder in Fiji. Look at the metadata by going to *Image -> Info*
- read the image here and write out a copy to maize-roots-copy.tif, check out the metadata in Fiji again.

What happened to the metadata?

```
[ ]: image = skimage.io.imread("../data/maize-roots.tif")
skimage.io.imsave("maize-roots-copy.tif", arr=image)
```

### 1.0.12 Exercise: Data type

- create a unsigned 16-bit numpy array where all pixels have the value 1 Hint: you can specify the dtype argument to `numpy.ones((num_rows, num_columns), dtype="uint16")`

- set a rectangular region from rows 100:500 and columns 100:700 to the value 200
- set a rectangular region from rows 100:500 and columns 100:700 to the value  $2 \times 14 + 10$
- save this image as a tif file "boring-uint16.tif"
- use the numpy function `.astype` to convert the image to unsigned 8 bit
- skimage has functions to convert images from one datatype to the other. Use `skimage.img_as_ubyte` to convert the image to 8 bit unsigned integers.
- save the converted image as a tif file "boring-uint8.tif"
- save the converted image as a png file "boring-uint8.png"

```
[ ]: import skimage.io
import skimage
import numpy
from matplotlib import pyplot as plt
image = numpy.ones((600, 800), dtype="uint16")
image[100:500, 100:700] = 200
image[300:400, 400:500] = 2 * 14 + 10
line = image[350, :]
plt.plot(line)
```

```
[ ]: skimage.io.imsave("boring-uint16.tif", arr=image)
skimage.io.imsave("boring-uint16-compressed.tif", arr=image, compress=6)
loaded16c = skimage.io.imread("boring-uint16-compressed.tif")
line16c = loaded16c[350, :]
plt.plot(line)
plt.plot(line16c)
```

```
[ ]: image8b = image.astype("uint8")
line8 = image8b[350, :]
plt.plot(line)
plt.plot(line8)
```

```
[ ]: image8s = skimage.img_as_ubyte(image)
line8s = image8s[350, :]
plt.plot(line8s)
```

```
[ ]: skimage.io.imsave("boring-uint8.tif", arr=image8s)
skimage.io.imsave("boring-uint8.png", arr=image8s)
skimage.io.imsave("boring-uint8.jpg", arr=image8s)
```

```
[ ]: image8s_jpg = skimage.io.imread("boring-uint8.jpg")
line8s_jpg = image8s_jpg[350, :]
plt.plot(line8s)
plt.plot(line8s_jpg)
```

### 1.0.13 Exercise: Slicing and the colorimetric challenge

- Consider the image `titration.tif` in the data directory
- We want to sample the color channel values from an image of a solution. To make our graph more reliable, we will want to calculate a mean channel value over several pixels, rather than simply focusing on one pixel from the image.

```
[ ]: image = skimage.io.imread("../data/titration.tif")
      print(image.shape)
      roi = image[200:260, 255:400]
      mean = numpy.mean(roi, axis=(0, 1))
      for i, chan in enumerate(["r", "g", "b"]):
          print("average for", chan, numpy.mean(roi[:, :, i]))
```

```
[ ]:
```

```
[ ]:
```

### 1.0.14 Keypoints

- skimage images are stored as three-dimensional NumPy arrays.
- In skimage images, the red channel is specified first, then the green, then the blue, i.e. RGB.
- Images are read from disk with the `skimage.io.imread()` function.
- We create a window that automatically scales the displayed image with `skimage.viewer.ImageViewer()` and calling `view()` on the viewer object.
- Color images can be transformed to grayscale using `skimage.color.rgb2gray()` or be read as grayscale directly by passing the argument `as_gray=True` to `skimage.io.imread()`.
- We can resize images with the `skimage.transform.resize()` function.
- NumPy array commands, like `img[img < 128] = 0`, can be used to manipulate the pixels of an image.
- Array slicing can be used to extract sub-images or modify areas of images, e.g., `clip = img[60:150, 135:480, :]`.
- Metadata is not retained when images are loaded as skimage images.

```
[ ]: (2 ** 14 + 10) / (2 ** 16 - 1) * (2 ** 8 - 1)
```

```
[ ]:
```