COMP 204

Introduction to image analysis with scikit-image (part one)

Mathieu Blanchette, based on slides from Christopher J.F. Cameron and Carlos G. Oliver
Until the end of the semester...

We will learn how to use Python modules that are very useful in life science applications:

▶ Scikit-image: Analysis of images (3 lectures)
▶ BioPython: Bioinformatics applications (2 lectures)
▶ Scikit-learn: Machine learning (2-3 lectures)

Our goal is not to learn everything about those packages (they contain hundreds of functions), but to learn the key ideas about them, and let you more easily use them in the future.
Goal: Process and analyze digital images.

- Very useful for processing microscopy images, medical imaging, etc.
- Closely linked with machine learning for image analysis

**scikit-image module or (skimage)**
- image processing module in Python
- holds a wide library of image processing algorithms: filters, transforms, point detection, etc.
- Documentation (Application Program Interface - API)
  - [http://scikit-image.org/docs/dev/api/api.html](http://scikit-image.org/docs/dev/api/api.html)
Each pixel’s color is represented using 3 integers, each between 0 and 255 (inclusively): \((R, G, B)\), where \(R\) = red intensity, \(G\) = green intensity, \(B\) = blue intensity. All colors can be expressed as RGB:

- black = (0,0,0)
- white = (255,255,255)
- red = (255,0,0)
- purple = (255,0,255)
- dark purple = (120,0,120)
- yellow = (255, 255, 0)
Our mascot for today
Reading an image into memory

Skimage's `io` submodule allows reading images into memory and writing them out to file.

API: http://scikit-image.org/docs/dev/api/skimage.io.html

- `image = io.imread(filename)` reads the image stored in `filename`
- `io.imsave(filename, image)` saves image to `filename`

read_write.py program:

```python
import skimage.io as io
import matplotlib.pyplot as plt

# read image into memory
image = io.imread("monkey.jpg")

# show the image on screen
plt.imshow(image)

# write image to disk
io.imsave("monkey_copy.jpg", image)
```
What’s an image in Python?

An image is stored as a NumPy ndarray (n-dimensional array).

- Ndarrays are easier and more efficient than using 2-dimensional lists as we’ve seen before.

A color image with \( R \) rows and \( C \) columns is

- represented as a 3-dimensional ndarray of dimensions \( R \times C \times 3 \)
- element at position \((i,j)\) of the array corresponds to the pixel’s RGB value at row \( i \) and column \( j \) of the image
- each pixel is represented by 3 numbers, each between 0 and 255: Red, Green, Blue

![Diagram of a 3D array representing an image](image)
NumPy’s ndarray

When loading an image with
image = io.imread("monkey.jpg"), you get an object of type ndarray, which contains the pixel values of the entire image.

Things to know about ndarrays:

▸ Get their dimensions:
    n_row, n_col, n_colours = image.shape

▸ Get a particular element at row \( r \), column \( c \), and depth \( d \):
    value = image[r,c,d]

▸ Get an RGB pixel value at row \( r \) and column \( c \):
    pixel_RGB = image[r,c]

▸ Change the color at row \( r \) and column \( c \):
    image[r,c] = [120,134, 231]
import skimage.io as io
import matplotlib.pyplot as plt

# read image into memory
image = io.imread("monkey.jpg")

n_row, n_col, n_colours = image.shape
print(n_row, n_col, n_colours)  # prints 1362 2048 3

# print pixel value at row 156 and column 293
pixel = image[156, 292]
print(pixel)  # prints [112 158 147]

# change pixel value to red
image[156, 292] = [255, 0, 0]

# Create a purple rectangle between rows 1000-1200 and column 500-900
for i in range(1000, 1200):
    for j in range(500, 900):
        image[i, j] = (255, 0, 255)

plt.imshow(image)
plt.show()
io.imsave("monkey_bar.jpg", image)
Creating the negative of an image

```python
import skimage.io as io
import matplotlib.pyplot as plt

# read image into memory
image = io.imread("monkey.jpg")
n_row, n_col, n_colours = image.shape

# Create the negative of an image
for i in range(n_row):
    for j in range(n_col):
        for c in range(n_colours):
            image[i, j, c] = 255 - image[i, j, c]

# we could just have written:
#image = 255 - image

plt.imshow(image)
io.imsave("monkey_negative.jpg", image)
```
Flipping the image horizontally (incorrect!)

```python
import skimage.io as io
import matplotlib.pyplot as plt

# read image into memory
image = io.imread("monkey.jpg")
n_row, n_col, colours = image.shape

# Flip the image horizontally
for i in range(0, n_row):
    for j in range(0, n_col):
        image[i, j] = image[i, n_col-j-1]

plt.imshow(image)
io.imsave("monkey_flipped_wrong.jpg", image)
```
Problem: For each row $i$, this mirrors the right half of the image into the left half (as it should), but by the time it reaches the right half ($j \leq n_{\text{col}}/2$), the left half of the image has already been changed, so we can no longer recover the original pixel values.
Flipping the image horizontally (correct)

```python
import skimage.io as io
import matplotlib.pyplot as plt

# read image into memory
image = io.imread("monkey.jpg")
n_row, n_col, colours = image.shape

# Flip the image horizontally
for i in range(n_row):
    for j in range(int(n_col/2)):
        colour = image[i, j].copy()
        opposite_colour = image[i, n_col-j-1].copy()
        image[i, j] = opposite_colour
        image[i, n_col-j-1] = colour

plt.imshow(image)
plt.show()
io.imsave("monkey_flipped_right.jpg", image)
```
Combining images

Since images are just arrays of numbers, we can easily combine them.

Example: Create an image that is the average of monkey and tiger.
Combining images

```python
import skimage.io as io
import matplotlib.pyplot as plt
import numpy as np
from skimage.transform import resize

monkey = io.imread("monkey.jpg")
tiger = io.imread("tiger.jpg")

# resize images to 500x1000 pixels
monkey_resized = resize(monkey, (500, 1000))
tiger_resized = resize(tiger, (500, 1000))

combined = np.zeros([500,1000,3])
for i in range(500):
    for j in range(1000):
        for c in range(3):
            combined[i,j,c] = monkey_resized[i,j,c]/2 + tiger_resized[i,j,c]/2

# we could also have replaced lines 13-19 with:
#combined = monkey_resized/2 + tiger_resized/2

plt.imshow(combined)
io.imsave("combined.jpg", combined)
```
Combining images
Color separation

Goal: Produce three images, one for each colors (see next slides)

```python
import skimage.io as io
image = io.imread("monkey.jpg")
n_row, n_col, colours = image.shape

# create three copies of the image
red = image.copy()
green = image.copy()
blue = image.copy()

# set to zero the B and G channels of the red image
# set to zero the R and B channels of the green image
# set to zero the R and G channels of the blue image
for i in range(n_row):
    for j in range(n_col):
        red[i,j,1]=red[i,j,2]=0
        green[i,j,0]=green[i,j,2]=0
        blue[i,j,0]=blue[i,j,1]=0

# We could have replaced the 5 lines above with:
#red[:,:,:,(1,2)] = 0
#green[:,:,:,(0,2)] = 0
#blue[:,:,:,(0,1)] = 0

io.imsave("monkey_red.jpg", red)
io.imsave("monkey_green.jpg", green)
io.imsave("monkey_blue.jpg", blue)
```
red intensity

green intensity

blue intensity
**Shifting colors**

Goal: Produce an image where the three colour channels are shifted (see next slide)

```python
import skimage.io as io
import matplotlib.pyplot as plt
import numpy as np
image = io.imread("monkey.jpg")
n_row, n_col, colors = image.shape

# create a blank image
new_image = np.zeros((n_row, n_col, 3), dtype=np.uint8)

# assemble a new image made of shifted colors
# blue is shifted right by 100 pixels
# green is shifted up by 100 pixels

for i in range(n_row):
    for j in range(n_col):
        new_image[i, j, 0] = image[i, j, 0]  # keep red
        if i >= 100:
            new_image[i, j, 1] = image[i-100, j, 1]  # move green
        if j >= 100:
            new_image[i, j, 2] = image[i, j-100, 2]  # move blue

plt.imshow(new_image)
io.imsave("monkey_shifted.jpg", new_image)
```
Grayscaling

Many image processing algorithms assume a 2D matrix not an image with a third dimension of color

To bring the image into two dimensions
  ▶ we need to summarize the three colors into a single value
  ▶ this process is more commonly know as grayscaling
  ▶ where the resulting image only holds intensities of gray
    ▶ with values between 0 and 1

skimage submodule color has useful functions for this task
  ▶ API
    http://scikit-image.org/docs/dev/api/skimage.color.html
Grayscaling

Goal: Create a grayscale version of a color image (see next slide)

```python
import skimage.io as io
import matplotlib.pyplot as plt
from skimage.color import rgb2gray

# read image into memory
image = io.imread("monkey.jpg")

# convert to grayscale
gray_image = rgb2gray(image)

print(image[0,0])  # prints [255,255,255]
print(gray_image[0,0])  # prints 1.0
plt.imshow(gray_image)
io.imsave("monkey_grayscale.jpg",gray_image)
```
Goal: Produce a black-and-white version of a color image (see next slide).

```python
import skimage.io as io
import matplotlib.pyplot as plt
from skimage.color import rgb2gray
import numpy as np

image = io.imread("monkey.jpg")
n_row, n_col, n_colours = image.shape

gray_image = rgb2gray(image)

black_and_white = gray_image.copy()
for i in range(n_row):
    for j in range(n_col):
        if gray_image[i, j] > 0.5:
            black_and_white[i, j] = 1.0
        else:
            black_and_white[i, j] = 0

# We could have replaced the 7 lines above with:
# black_and_white = np.where(gray_image > 0.5, 1, 0)

plt.imshow(black_and_white)
io.imsave("monkey_bw.jpg", black_and_white)
```