image analysis

- we've already dealt a bit with images using numpy and matplotlib
- 3 main packages specifically for image manipulation and analysis in python:

    i. scikit image (skimage)
    ii. scipy.ndimage
    iii. opencv - more advanced algorithms (machine learning), multiple languages and versions, tricky to install, multiple versions

- recall how we can display numpy arrays as images:

```
import numpy as np
import matplotlib.pyplot  as plt
from skimage import io # functions for image file input/output
faceg = io.imread( 'face_gray.png ') # 2D array, dtype is uint8
faceg.shape # 782 x 782, fairly big
f, ax = plt.subplots()
im = ax.imshow(faceg)  # this uses default colormap 'viridis'
```

  - why is matplotlib displaying colour, when we know the image is grayscale?
  - 2D arrays represent only "single channel" images, each pixel can only represent one thing: luminance. By default, MPL puts single channel images through a color map, default is 'viridis', but we can choose 'gray' instead:

```
f.colorbar(im)  # add color bar to figure
plt.colormaps()  # list all available colormaps
im.set_cmap( 'gray') # use grayscale instead
```

  - 3 general types of colour maps:
    - sequential - data going ranging from low to high value,
    - diverging - data that deviates around a center value like 0
    - categorical - discrete colours for particular ranges of data
  - see colormap plots and explanations at: http://matplotlib.org/users/colormaps.html
  - can also directly specify desired colormap in imshow():

```
f, ax = plt.subplots()
im = ax.imshow(faceg,  cmap='gray')
f.colorbar(im)
```

  - how to turn off the axes x and y ticks? set them to empty lists:
    - `ax.set_xticks([]), ax.set_yticks([])`
  - how to rotate arrays, and therefore images?

```
f, ax = plt.subplots()
face90 = np.rot90(faceg)  # 90 deg counterclockwise
```

```
ax.imshow(face90, cmap='gray')
io.imsave('face90.png', face90) # save it back to a new file
```

  - for specific (non 90 deg) angles of rotation:

```
from scipy import ndimage
face45 = ndimage.rotate(faceg, 45) # 45 deg counterclockwise
ax.imshow(face45, cmap='gray')
```

  - can use `np.flipud` and `np.fliplr` to flip arrays vertically or horizontally
  - subsample an image using array slicing:

```
f, ax = plt.subplots()
lowres = faceg[::10, ::10] # every 10th pixel in both dimensions
lowres.shape # 79x79, 1/10 the size in both dimensions
ax.imshow(lowres, cmap='gray')
```

  - `imshow` can *display* images using different kinds of interpolation, default is no interpolation

```
f, ax = plt.subplots()
ax.imshow(lowres, cmap='gray', interpolation='gaussian')
```

  - 'bilinear', 'bicubic', 'gaussian', 'spline16', are common, see list of all interpolation methods
  - this is just for display, doesn't modify the array. Let's create a new array by interpolating the existing one, with more control over the smoothing:

```
from skimage import filters
lowresgauss = filters.gaussian(lowres, sigma=2) # sigma in number of pixels
lowresgauss.shape # it's still a small 79x79 array...
f, ax = plt.subplots()
ax.imshow(lowresgauss, cmap='gray') # ... but its pixels have been smoothed
```

- resizing an image, either up or down:

```
biglowresgauss = ndimage.zoom(lowresgauss, 10) # resize up to original size
biglowresgauss.shape # 790 x 790
f, ax = plt.subplots()
ax.imshow(biglowresgauss, cmap='gray') # ... but its pixels have been smoothed
```

- colour
  - if you have image data in a 2D array, the values for each pixel can only represent luminance, there's no colour information
  - can map the luminance to colour using a colour map, but you can't independently represent luminance and colour with a single value per pixel
  - how can colour be represented in an array? as red, green and blue channels (RGB)

- you could use 3 separate 2D arrays to represent RGB for an image, but normally these are combined into a single 3D array: `nrows x ncols x 3`

```
cface = io.imread('face.png')
cface.shape # notice it's a 3D array, last dimension represents colour
f, ax = plt.subplots()
ax.imshow(cface)  # no colormap used when image already has color info
```

- scikit-image has several demo images built in:

```
from skimage import data
immun = data.immunohistochemistry()
immun.shape # notice it's a 3D array, last dimension represents colour
f, ax = plt.subplots()
ax.imshow(immun)
```

- we can modify the R, G and B channels separately. Let's remove all the red from the image. How can we do this by manipulating the array data?
  - set the red channel to 0, and then the green channel:

    ```
    temp = immun.copy()
    temp[:, :, 0] = 0 # no more red
    ax.imshow(temp)  # teal remains
    temp[:, :, 1] = 0 # no more green
    ax.imshow(temp)  # only blue remains
    ```

- color conversion functions in `skimage.color`
  - convert RGB image to grayscale:

    ```
    from skimage import color
    immung = color.rgb2gray(immun)
    immung.shape # now it's only 2D
    ax.imshow(immun, cmap='gray')
    ```

- images can also have a 4th channel: alpha, aka transparency. The alpha channel is a transparency mask for the image. 0 is fully transparent, 255 is fully opaque. So pixels you don't want painted are set to 0:

```
facea = io.imread('face_alpha.png')
facea.shape # gives (782, 782, 4), i.e. RGBA
alpha = facea[:, :, 3] # prints last (4th) hypercolumn
f, ax = plt.subplots()
ax.imshow(alpha, cmap='gray')
bg = np.zeros_like(facea)  # init a background
bg[:, :] = 255, 0, 0, 255 # set to red, and fully opaque
```

- now display background, and then image with transparency to see the effect:

```
f, ax = plt.subplots()
ax.imshow(bg)  # first show background
ax.imshow(facea)  # see red where foreground image alpha is low
```

- can also blend array with a red background with `color.rgba2rgb(facea, [1, 0, 0])`

- contrast
    - plot histogram of pixel values, have to ravel (flatten) to 1D array first!
    - `ax.hist(immung.ravel(), bins=np.arange(256))`
    - change contrast by scaling pixel values within limits of 0 to 255
- edge detection:

```
ohki = io.imread('ohki2005.png')
f, ax = plt.subplots()
ax.imshow(ohki, cmap='gray')
f, ax = plt.subplots()
ax.hist(ohki.ravel(), bins=np.arange(256))
edges = filters.sobel(ohki)
f, ax = plt.subplots()
ax.imshow(edges, cmap='gray')
```

- edge-based segmentation:

```
from skimage.feature import canny
edges = canny(ohki, sigma=1.5)
from scipy import ndimage
mask = ndimage.binary_fill_holes(edges)
f, ax = plt.subplots()
ax.imshow(mask, cmap='gray')
```

- watershed-based segmentation:
    - see exercises
- loading movies using pyav:
    - to install pyav in anacoda: `conda install -c soft-matter pyav` at the command line

```
import av
v = av.open('movie.avi')
mv = []
for frame in v.decode(video=0): # get the first video stream in file
    mv.append(np.asarray(frame.to_image()))
mv = np.asarray(mv)
mv.shape # 4D array: nframes x nrows x ncols x 3 colour channels
f, ax = plt.subplots()
ax.imshow(mv[0]) # plot the first frame
```

- useful resources:
- skimage image gallery:
    - http://scikit-image.org/docs/dev/auto_examples/
- recent local "SuperPython" talk by Joe Donovan for lots of examples:
    - https://github.com/superpythontalks/image_analysis/blob/master/image%20processing.ipynb