Image_Processing

Display the images

```
### **Load an image:** .bmp, .jpg or jpeg, tif or tiff
from PIL import Image # import library
import urllib.request, skimage.io
urllib.request.urlretrieve('htttps://...') # save image from the url
img_lenna = Image.open('lenna.png) # load the image using PIL.Image
display(img_lenna)
img2 = skimage.io.imread(fname='x.png') # load imge using [skimage.io]
(http://skimage.io/) - 坐标
skimage.io.imshow(img2)
### **Format**
img.format         ⇒ PNG
“%s”%(img.size,)   ⇒ (512, 512)
img.mode           ⇒ RGB
import numpy as np
im = np.array(img)
str(type(im))      ⇒ array type: <class 'numpy.ndarray'>
str(im.dtype)      ⇒ datatype: uint8
"%s"%(im.shape,)   ⇒ (512, 512, 3)
```

Play with the images (colors, sizes, rotations)

```
**Check out Channels**
Image.fromarray(im[:,:,0]) // R
Image.fromarray(im[:,:,1]) // G
Image.fromarray(im[:,:,2]) // B
**Warm color → cool color**
cool_img=Image.fromarray(255-im)  // inverse pix val in range[0,255].
img_lenna.convert("L")# Gray scale
**Resize an image(smaller)**
from numpy.core.fromnumeric import shape
cool_array = 255-im
h_new, w_new = int(im.shape[0]/2), int(im.shape[1]/2)
cool_array_small = np.ndarray((height_new, width_new,3), dtype=np.uint8)
for i in range(h_new):
    for j in range(w_new): cool_array_small[i,j,:] = cool_array[i*2,j*2,:] #
array: [H,W,C]
cool_img_small = Image.fromarray(cool_array_small) # array to img
cool_array_small.dtype    ⇒ uint8
type(cool_array_small)    ⇒ <class 'numpy.ndarray'>
cool_img_small.size       ⇒ (256, 256)

**(larger)**
cool_array_large[i,j,:]=cool_array[i//2,j//2,:]  ⇒ img_size: (1024, 1024)
```

Problem for enlarge: Image Interpolations

```
**Rotation by index**
height_cw90,width_cw90=cool_array.shape[1],cool_array.shape[0]        ⇒ change height&width
cool_array_cw90[i,j,:] = cool_array[width_cw90-1-j, i, :]             ⇒ clockwise 90
Image.fromarray(cool_array_cw90[:, ::-1])                             ⇒ flip horizontally
cool_array_new[i,j,:]cool_img_small[height_new-1-i,width_new-1-j,:]   ⇒ flip vertically

**Rotation in OpenCV**
img_rotate_90_clockwise = cv2.rotate(cool_array, cv2.ROTATE_90_CLOCKWISE)
img_flip_ud = cv2.flip(cool_array, 0)

**Color in OpenCV**
im_gray = cv2.cvtColor(cool_array, cv2.COLOR_BGR2GRAY) # convert img to gray
th, im_gray_th_otsu = cv2.threshold(im_gray, 128, 192, cv2.THRESH_OTSU)
cool_img_new = Image.fromarray(im_gray_th_otsu) # binarize the image using a threshold

**Resize in OpenCV**
cool_img1= cool_img.resize((800,300),Image.BICUBIC)

**Example:**
top_left_img = cv2.resize(cool_array, (256,256), interpolation = cv2.INTER_AREA)
top_right_img = cv2.flip(top_left_img,1)           bottom_left_img = cv2.flip(top_left_img,0)
bottom_right_img = cv2.flip(top_right_img,0)
cool_img_final[:256,:256,:]=top_left_img
cool_img_final[:256,256:,:]=top_right_img
cool_img_final[256:,:256,:]=bottom_left_img
cool_img_final[256:,256:,:]=bottom_right_img
```

Play with the videos (files, webcam)

Illumination Normalization, Sensor Gap and Color Normalization

```
**Load video stream(webcam)**
import cv2
vid = cv2.Videocapture(0)   # declare a video capture object
while True:
    ret, frame_raw = vid.read() # capture frame
    frame = cv2.flip(frame_raw, 1) # optional
    key=cv2.waitKey(1)& 0xFF        if key == ord('q'):  break  # quit when 'q' pressed
vid.release()  # release cap object cv2.destroyAllwindows()    # destroy all
```

```
windows
**Invert the images (frames)**
inversion_on=False
cv2.imshow('COMP 4423', 255-frame if inversion on else frame) # result frame
if key == ord('i'):  inversion_on = not inversion_on # no inv when '1' pressed
**Put the hair on**
im_hair = cv2.imread('redhair.png', CV2.IMREAD UNCHANGED)
hh,wh,ch = im_hair.shape # load the redhair image
im_hair = cv2.resize(im_hair,(int(wh*0.7), int(hh*0.7)), interpolation =
CV2.INTER_AREA)
non_trans=np.argwhere(im_readhair[:,:,3]>0) # find index of non-transparent pix
h2,w2,c2=im_readhair.shape         alpha = im_readhair[:,:,3] ; hair_on=False
h, w, c= frame.shape #get dimensions of the frame print(h,w,c,'-',h2,w2,c2)
if hair on:     top,left=50,int((w-w2)/2)
    #frame[top:top+h2,left:left+w2,:]im readhair[:,:,0:3]
    frame[top+non _trans[:,0],left+non trans[:,1],:]=im readhair[non
trans[:,0],non trans[:,1],0:3] # put the hair on
if key==ord('h'): hart_on=not hair_on # no hair when '1' pressed
```

Filters and Convolutions

Edge Detection: edges in image brightness may result from the change of depth/surface orientation/material/illumination

(66 *image) x (*33 filter) = 4* image output after edge detection box * matrix then move

Edge Filters

Prewitt filter ⇒ edge detector for horizontal & vertical edges $Gx=[1,0,-1][1,0,-1][1,0,-1]$ ⇒ vertical edge detection, $Gy=[1,1,1][0,0,0][-1,-1,-1]$ ⇒ horizontal edge

Sobel filter ⇒ ****reduces noise, differentiates regions, & responses to edges. $Gx=[1,0,-1][2,0,-2][1,0,-1]$⇒ vertical edge detection, $Gy=[1,2,1][0,0,0][-1,-2,-1]$⇒ horizontal edge Magnitude: $|G| = √(Gx^2 + Gy^2)$ Approximation: $|G| = |Gx|+|Gy|$ Orientation: $θ = arctan(Gx/Gy)$

Laplacian Filter ⇒ 2nd-order derivative → noise-sensitive, do Gaussian smoothing to avoid noise $Gx=[-1,-1,-1][-1,8,-1][-1,-1,-1]$⇒ vertical edge detection, $Gy=[0,-1,0][-1,4,-1][0,-1,0]$⇒ horizontal edge $∇*f_x*(*x*,*y*)=f(x,y)-f(x+1,y)$ $∇*f*^2(*x*, *y*)=∇f*x*^2(*x*, *y*)+∇*fy*^2(*x*,y)$ $∇*f*^2_x(*x*, *y*)≈-f(x-1,y)+2f(x,y)-f(x+1,y)$ $∇*f*^2_y(*x*, *y*)≈-f(x,y-1)+2f(x,y)-f(x,y+1)$

```
**Detect Edge by code**
img_gray.filter(ImageFilter.FIND_EDGES) # argumented filter
img_gray.filter(ImageFilter.Kernel((3, 3), (-1, -1, -1, -1, 8, -1, -1, -1, -1), 1,
0)) # detect edges - Laplacian filter
img_gray.filter(ImageFilter.Kernel((3, 3), (matrix), 1, 0))  # detect edges -
Sobel filters
```

Noise Reduction

```
# Add salt-and-pepper noise
noise_array = random_noise(img_lenna, mode='s&p',amount=0.05)
# convert array into uint8 before composing the image
noise_array = np.array(255*noise_array, dtype = 'uint8')
img_denoise=noise_img.filter(ImageFilter.Kernel((3, 3), (1/9, 1/9, 1/9, 1/9, 1/9,
1/9, 1/9, 1/9, 1/9), 1, 0))
# **Median Filter:** *reduce noise by replacing central pix by median of pix in
neighborhood.*
image_denoise = noise_img.filter(ImageFilter.MedianFilter)
```

Noise → Recognized Filter | salt & pepper → Median | Poisson→ Mean | Gaussian → Gaussian | Speckle → Weiner

Morphological Operations

Binary images ← structuring element (probe image with a small shape)

Erosion → $f \ominus s$, fits the input image f, i.e. $g(x,y)=1$if $s$fits $f$and 0 otherwise

1. remove all small scale details from a binary img
2. reduces the size of regions of interest

subtraction erode img → $b=f-(f \ominus s)$ region boundaries

Example

```
# covert to gray, get binary image, fromarray(255-img_th)
SE = np.ones((5,5),np.uint8)
img_erosion = cv2.erode(255-img_th,SE,iterations = 1)
bw_img=cv2.erode(255-img_th,SE,iterations = 1) 5*5 => 12*12
# merge
****np_merge=cv2.merge((img_erosion,img_erosion,img_erosion))*0.4+np_virus*0.6
Image.fromarray(cv2.cvtColor(np_merge.astype(np.uint8), cv2.COLOR_BGR2RGB))
# **count num**
from skimage.measure import label,regionprops
np_labeled=label(img_erosion)
io.imshow(np_labeled) # from skimage import io
len(regionprops(np_labeled)) # count num

# Build Mask
np_regions=[]
np_masks=[]
merged_mask=np.zeros(np_labeled.shape)
for index in range(1, np_labeled.max()+1):
    np_regions.append((((np_labeled==index)+0)*255).astype(np.uint8))
    np_dilation = cv2.dilate(np_regions[-1],SE,iterations = 6)
    np_masks.append(np_dilation)
    merged_mask=merged_mask+np_dilation # display(Image.fromarray(np_dilation))
np_merge=cv2.merge((merged_mask,merged_mask,merged_mask))*0.4+np_virus*0.6
Image.fromarray(cv2.cvtColor(np_merge.astype(np.uint8), cv2.COLOR_BGR2RGB))
```

```
# Seperate
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(60, 10))
binary_np=255-img_th
idx=1
plts=[]
for mask in np_masks:
    binary_virus=((mask==255)*(binary_np==255))
    area=np.sum(binary_virus)
    #display(Image.fromarray(cv2.cvtColor((binary_virus*255).astype(np.uint8), cv2.COLOR_BGR2RGB)))
    plts.append(fig.add_subplot(3, 12, idx))
    plts[-1].set_title('mask'+str(idx))
    plt.imshow(mask)
    plts.append(fig.add_subplot(3, 12, idx+12))
    plts[-1].set_title('binary')
    plt.imshow(binary_np)
    plts.append(fig.add_subplot(3, 12, idx+24))
    plts[-1].set_title('virus'+str(idx)+'(area='+str(area)+')')
    plt.imshow(binary_virus)
    print('Area of virus '+str(idx)+':\t'+str(area))
    idx=idx+1
```