

Image Generation, Processing & Understanding with Python

Yongduek Seo

2019-09-18

Contents

1	Primer	5
2	Preparation & Documents	7
3	Pixelwise Operations	9
3.1	Blending Two Images	9
3.2	Negative Film Effect	14
3.3	Histogram of RGB numpy image	16
4	Float32 Pixel Representation	19
5	Geometric Transformations	23
6	Histogram Equalization	25
7	Gamma Correction	31
8	HSV Color Space	37
9	Video File Manipulation	45
9.1	Video File Read/Write	45
9.2	Video File Set Frame Position	46
10	Convolution (Space Filter)	49
10.1	Smoothing or Blurring	49
11	Morphological Filters	59

12 Gradient & Edge Operation	61
12.1 Sobel-X	61
12.2 Canny Edge Detector	65
13 MoviePy to manipulate movie files	69
13.1 Install youtube-dl	69
13.2 Text overlay on a video clip	69
13.3 Extract audio (MP3) from a movie file	70
13.4 Movie Concatanation	71
13.5 Apply Filter Frame by Frame	71
13.6 MoviePy Open & Get A Frame	72
13.7 Errors	73
14 k-means clustering	75
15 EOF	83
16 SuperPixel Segmentation	85
17 Non-Photorealistic Rendering	87
17.1 OpenCV non-photorealistic rendering	87
17.2 Hand-made cartoon-like filtering	92
18 Drawing Text in Image with a Font (OTF) file	97
19 Photo Mosaic	99
19.1 General Procedure	99
19.2 References	99

Chapter 1

Primer

Chapter 2

Preparation & Documents

1. python 3.x
2. pip install opencv-python numpy matplotlib
3. OpenCV-Python Tutorial
4. docs.opencv.org
5. Python Image Library: Pillow
6. scikit-image.org
7. github.com/opencv

Chapter 3

Pixelwise Operations

3.1 Blending Two Images

$$I_B = \alpha I_1 + (1 - \alpha) I_2, \quad \alpha \in [0, 1]$$

Procedure:

1. prepare two image files
2. read the two files
3. set α to a value, e.g. 0.5
4. compute the weighted sum of the two images
5. display the result

Notice:

- `np.uint8` is the type of a pixel value for display.
- BGR in OpenCV!

Try:

- make a video showing a progressive change from one image to another by increasing α from 0 to 1 smoothly.

```
# filename blending.py

import sys
import numpy as np
import cv2
```

```

import matplotlib
matplotlib.use ('TkAgg')
import matplotlib.pyplot as plt

size = (256, 300)
# read two images

file1 = 'data/dooly.jpeg'
i1 = cv2.imread (file1)
if i1 is None:
    print ('image file read error: ', file1)
    sys.exit()

i1 = cv2.resize(i1, size)

file2 = 'data/pororo.jpeg'
i2 = cv2.imread (file2)
if i2 is None:
    print ('image file read error: ', file2)
    sys.exit()

i2 = cv2.resize (i2, size)

results = []
alphas = np.linspace(0, 1, 6) # allocate alpha values
print ('alpha: ', alphas)

```

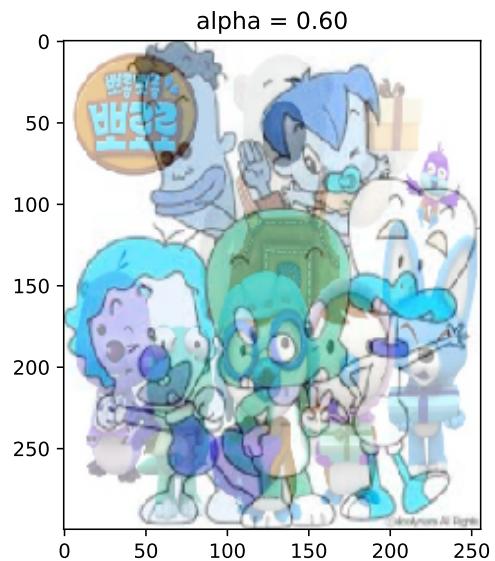
```
## alpha: [0. 0.2 0.4 0.6 0.8 1. ]
```

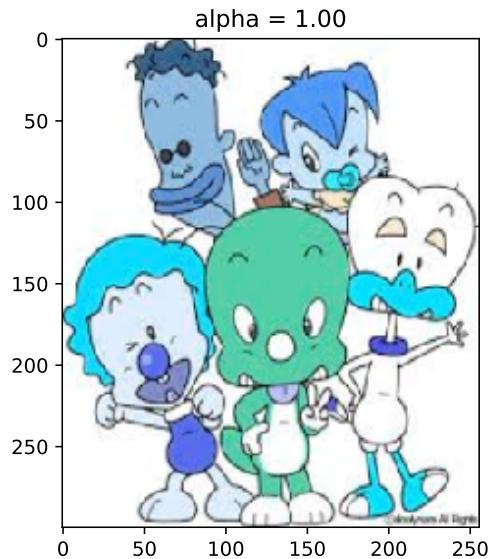
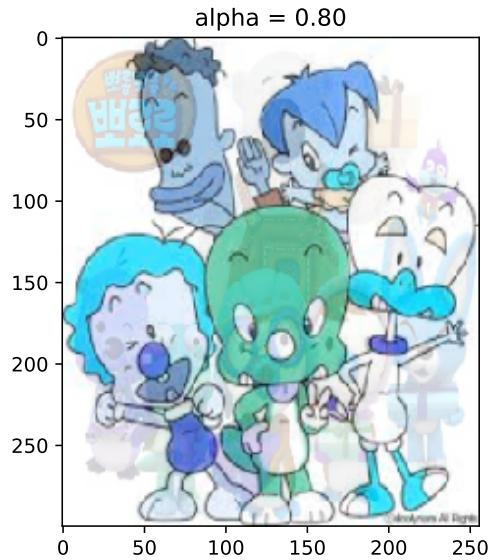
```

for a in alphas:
    J = a * i1 + (1. - a) * i2
    J = np.clip(J, 0, 255).astype(np.uint8)
    results.append (J)
    print (a)
    plt.imshow (J)
    plt.title ('alpha = %.2f' % a)
    plt.pause (1)
#

```

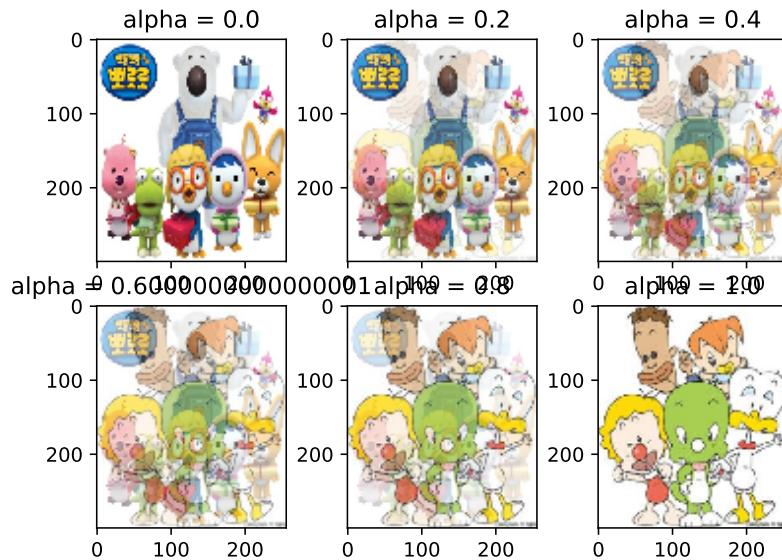






```
fig, axes = plt.subplots (2, 3)
for i, ax in enumerate(axes.ravel()):
    ax.imshow (results[i][:,:,:-1]) # BGR -> RGB
    ax.set_title ('alpha = {}'.format(alphas[i]))
```

```
#  
plt.pause(2)
```



```
plt.close()  
  
# EOF
```

3.2 Negative Film Effect

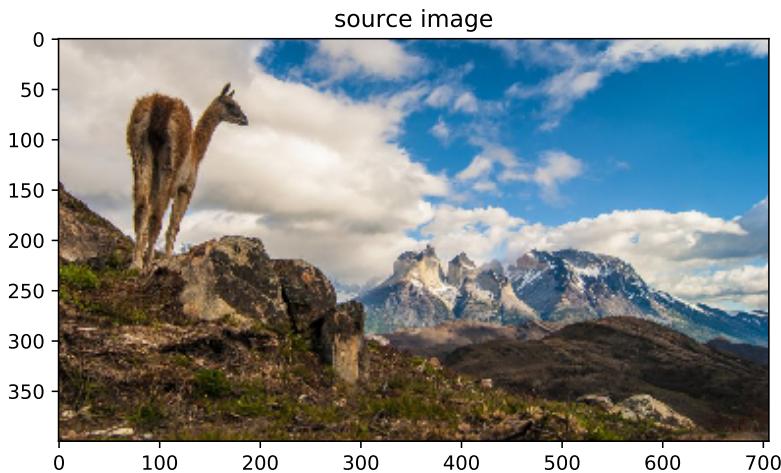
The image looks like a negative film in old days.

```
# filename negative_film.py  
  
import sys  
import numpy as np  
import cv2  
import matplotlib  
matplotlib.use('TkAgg')  
import matplotlib.pyplot as plt  
import imageio # this will be used to load an image file  
  
# show the image
```

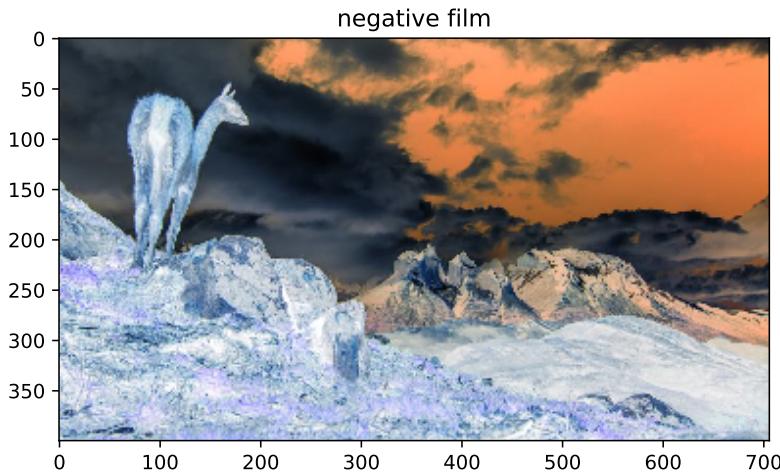
```
def imshow (img, title=None):
    plt.imshow (img)
    if title is None: title = 'imshow'
    plt.title (title)
    plt.pause (1)
    plt.close ()

#
def negative_film (img):
#    return 255 - img # simple way using numpy.
    neg = np.zeros_like (img)
    for r in range (img.shape[0]):
        for c in range (img.shape[1]):
            for d in range (img.shape[2]):
                neg[r,c,d] = 255 - img[r,c,d]
    return neg
#
img = imageio.imread ('data/torres-del-paine.jpg')

imshow (img, 'source image')
```



```
neg = negative_film(img)
imshow(neg, 'negative film')
# EOF
```



3.3 Histogram of RGB numpy image

```
# filename: numpy-hist.py

import sys
import matplotlib.pyplot as plt
import numpy as np
import cv2

imagefile = 'data/torres-del-paine.jpg'

frame = cv2.imread(imagefile)

bluehist = np.zeros((256), dtype=np.float)
redhist = np.zeros((256), dtype=np.float)
```

```
greenhist = np.zeros((256), dtype=np.float)

# make histograms, one for each color
for r in range(frame.shape[0]):
    for c in range(frame.shape[1]):
        blue_intensity = frame[r,c][0]
        bluehist[blue_intensity] += 1
        redhist[frame[r,c,2]] += 1
        greenhist[frame[r,c][1]] += 1
    #

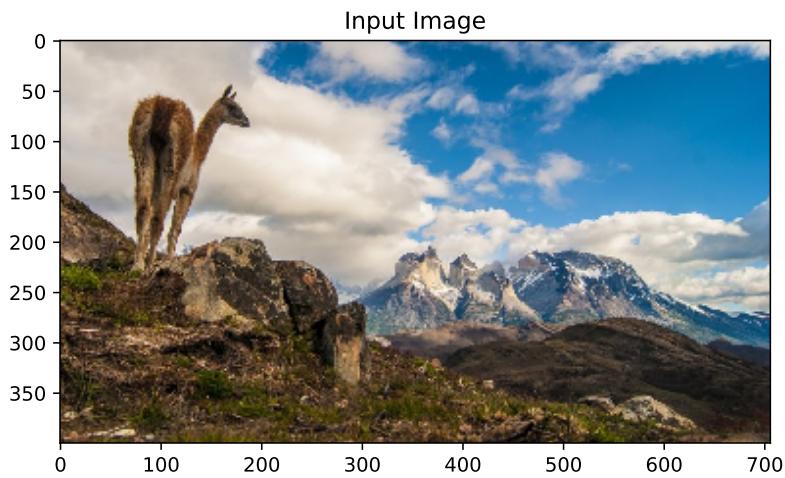
# convert to ratio = count / num_pixels
num_pixels = frame.shape[0] * frame.shape[1]
bluehist /= num_pixels
greenhist /= num_pixels
redhist /= num_pixels

plt.imshow (frame[:, :, ::-1]) ## cv2's BGR -> RGB
plt.title ('Input Image')
plt.pause (1)
plt.close()

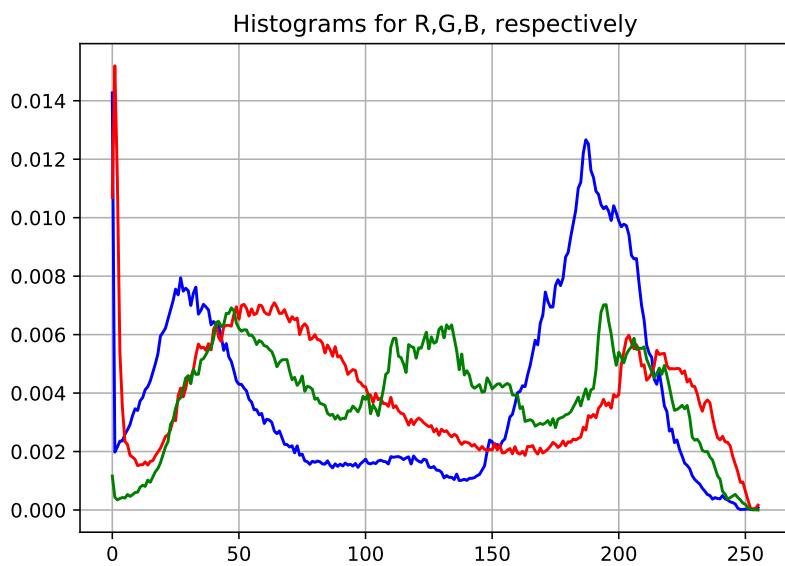
x = range(0,256,1)
plt.plot (x, bluehist, 'b', x, redhist, 'r', x, greenhist, 'g')
plt.grid(True)
plt.title ('Histograms for R,G,B, respectively')
plt.pause (1)
plt.close ()

#EOF
```

A result of the program is shown below.



```
## [
```



Chapter 4

Float32 Pixel Representation

- `uint8` is a popular choice. 24bit representation.
- `float32` for $[0, 1]$ pixel values is another choice.
 - One RGB pixel occupies $3 \times 32\text{bits}$ or 12byte representation.
- TIF image file format can be used to save images of `float32` pixel data.

```
# filename: float-image-type.py

# image pixel data is represented normally by uint8
# but float32 is also a popular representation.
# Use TIF format to save a float32 RGB image
# Image Viewer for TIF float32 is ImageMagick/display

import imageio
import matplotlib.pyplot as plt
import numpy as np

im = imageio.imread ('data/imgKorea012.png')
plt.imshow (im)
plt.title('Original Image')
plt.pause(1); plt.close()

# convert to float type image in the range [0, 1.]
```



```
im = np.float32(im / 255.)  
print ('float32 image: ', im.dtype, im[100,100])
```

```
## float32 image:  float32 [0.95686275 0.9764706  0.9764706 ]
```

```
plt.imshow (im)  
plt.title ('float32 type image can also be displayed by plt.')  
plt.pause(1); plt.close()  
  
# automatic conversion to uint8 is provided.
```

float32 type image can also be displayed by plt.



```
png_filename = 'data/imgKorea012-float32.png'
imageio.imwrite(png_filename, im)
```

```
## Lossy conversion from float32 to uint8. Range [0, 1]. Convert image to uint8 prior to saving t
```

```
im2 = imageio.imread(png_filename)
print ('png file reloaded: ', im2.dtype, im2[100,100])
#
```

```
## png file reloaded:  uint8 [244 249 249]
```

```
tif_filename = 'data/imgKorea012-float32.tif'
imageio.imwrite (tif_filename, im)
imtif = imageio.imread(tif_filename)
print ('tif file reloaded: ', imtif.dtype, imtif[100,100])
```

```
## tif file reloaded:  float32 [0.95686275 0.9764706  0.9764706 ]
```

```
plt.imshow (imtif)
plt.title ('TIF image format can save float32 type image.')
plt.pause(1); plt.close()
```

```
#
```

TIF image format can save float32 type image.



```
import cv2
imcv = cv2.imread (tif_filename)
if imcv is None:
    print ('tif float32 cannot be loaded by cv2.imread()')
else:
    print ('tif loaded by cv2: {}'.format(imcv.shape))
#
# EOF

## tif float32 cannot be loaded by cv2.imread()
```

Chapter 5

Geometric Transformations

1. Translation
2. Scaling (magnifying x2)
3. interpolation
4. Rotation
5. Matrix Notation & Homogeneous Coordinate Representation
6. Similarity Transformation
7. Affine Transformation
8. Projective (Perspective) Transformation

Chapter 6

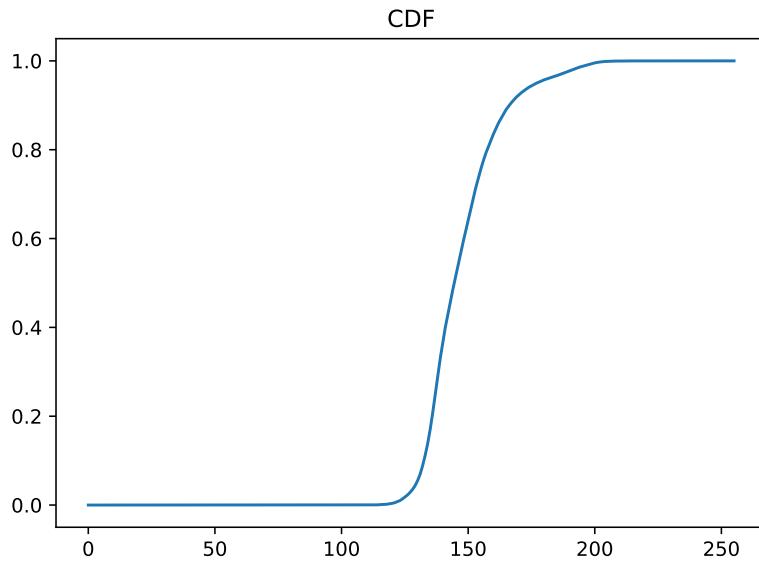
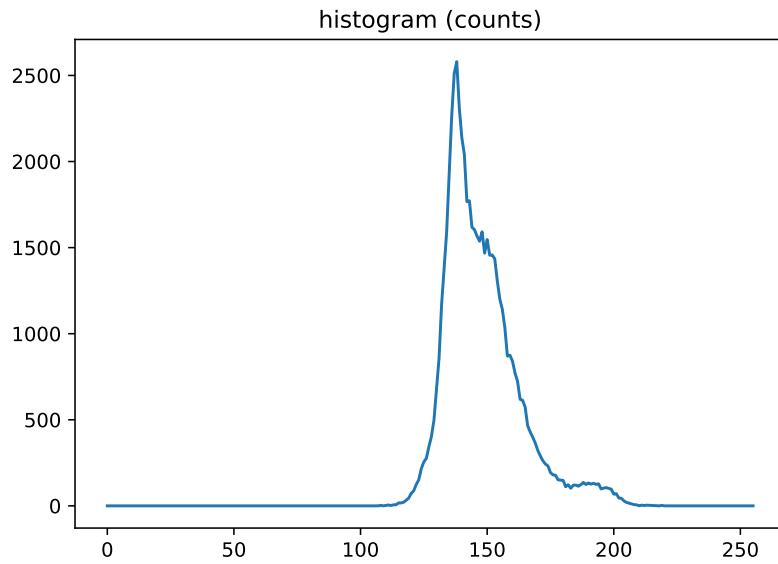
Histogram Equalization

- Check Wikipedia Histogram Equalization

```
## image size (gray scale): (200, 300)
```

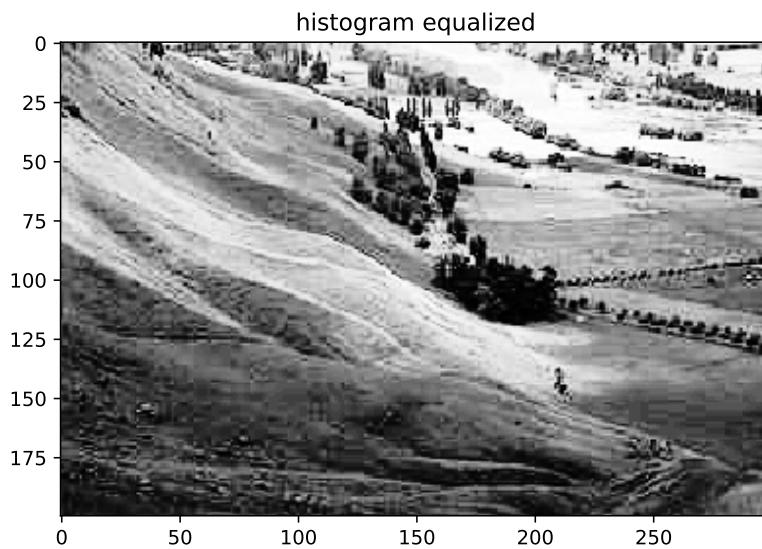
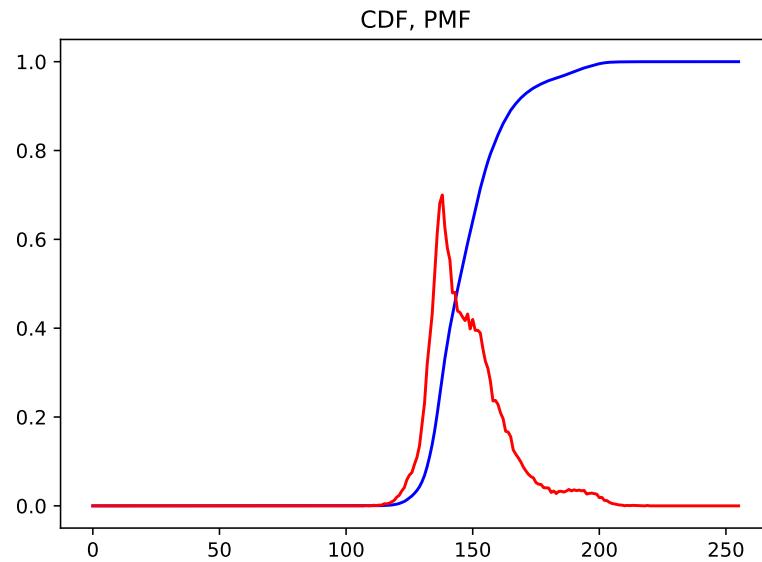


```
## cdf: (256,)
```

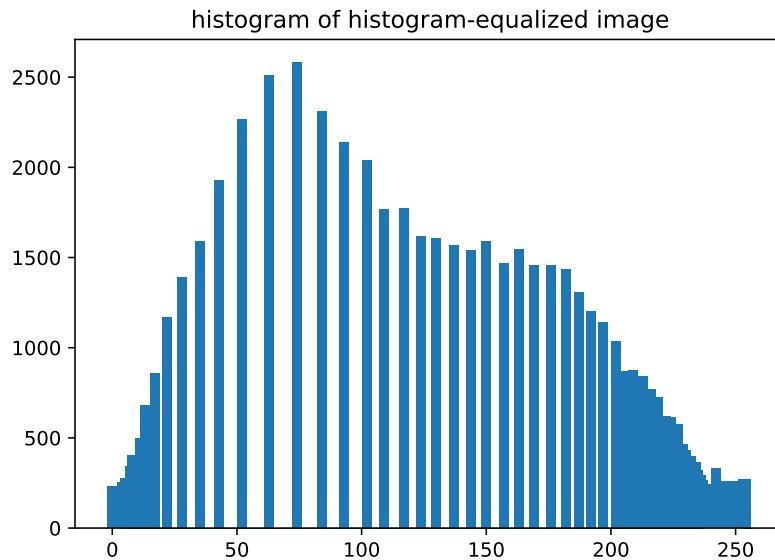


```
## max of pmf:  0.043
```

```
## [, <matplotlib.lines.Line2D object
```



```
## <BarContainer object of 256 artists>
```



```
# filename: histogram_equalization.py
# https://en.wikipedia.org/wiki/Histogram_equalization

import cv2
import numpy as np
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt

imagefilename = 'data/300px-Unequalized_Hawkes_Bay_NZ.jpg'
img = cv2.imread(imagefilename, cv2.IMREAD_GRAYSCALE)
print ('image size (gray scale): ', img.shape)

plt.imshow (img, cmap='gray')
plt.pause (1)
plt.close()

def histogram (ii):
    h = np.zeros(256)
    for val in ii.flatten():
        h[val] += 1
    return h
#

def make_cdf (pmf):
```

```

cdf = np.zeros (pmf.shape)
print ('cdf: ', cdf.shape)
cdf[0] = pmf[0]
for x in range(1, cdf.shape[0]):
    cdf[x] = cdf[x-1] + pmf[x]
return cdf
# 

# compute histogram
h = histogram (img)

# normalize to get PMF, Probability Mass Function
pmf = h / float(img.shape[0]*img.shape[1])

# Accumulate to get CDF, Cumulative Distribution Function
cdf = make_cdf (pmf)

plt.plot (h); plt.title ('histogram (counts)') # show the histogram
plt.pause (1) # seconds
plt.close()

plt.plot (cdf); plt.title('CDF') # check the CDF
plt.pause (1)
plt.close()

#
pmf_max = pmf.max()
print ('max of pmf: ', pmf_max)
plt.plot (range(cdf.shape[0]), cdf, 'b-', range(cdf.shape[0]), pmf*0.7/pmf_max, 'r-');
plt.title ('CDF, PMF')
plt.pause(1)
plt.close()

def histogram_equalization (img, cdf):
    ieq = np.zeros_like (img)
    for r in range(img.shape[0]):
        for c in range(img.shape[1]):
            pixelvalue = img[r,c]
            ieq[r,c] = np.clip(255. * cdf[pixelvalue], 0, 255).astype (np.uint8)
    #
    return ieq
#

# do it now
img_eq = histogram_equalization (img, cdf)

```

```
plt.imshow (img_eq, cmap='gray'); plt.title('histogram equalized')
plt.pause(1)
plt.close()

# check the histogram of the equalized image.
# verify what you did.

heq = histogram (img_eq)

plt.bar (range(0,256), heq, width=4)
plt.title('histogram of histogram-equalized image')
plt.pause (1)
plt.close()

# Now, make & plot the CDF of heq

# EOF
```

Q. For an RGB image, convert it to HSV format, apply hisotgram equalization to V channel, convert back to RGB. Show an example.

Chapter 7

Gamma Correction

- Read Wikipedia Gamma Correction Page

```
# filename gamma_correction.py
# https://en.wikipedia.org/wiki/Gamma_correction
# The image used from wikipedia seems to be from https://www.art.com/gallery/id--c23951/black-and-white-art/black-and-white-art-1000x1000px

import sys
import numpy as np
import cv2
import matplotlib
matplotlib.use ('TkAgg')
import matplotlib.pyplot as plt
import imageio # this will be used to load an image file
import skimage # rgb <-> hsv conversion in [0,1] pixel scale

# show the image
def imshow (img, title=None):
    if img.ndim == 3:
        plt.imshow (img)
    else:
        plt.imshow (img, cmap='gray')

    if title is None: title = 'imshow'
    plt.title (title)
    plt.pause (1)
    plt.close ()

#
img = imageio.imread ('data/art.com.jpg') # it is an RGB format even though ...
```

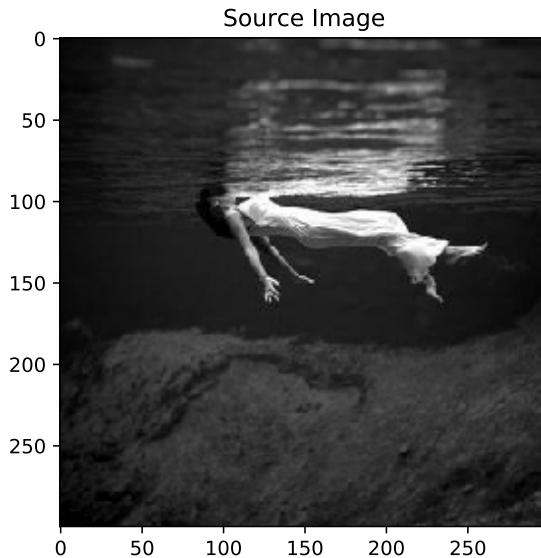
```

if img is None:
    print ('image file open error')
    sys.exit ()
#
print (img.shape)

## (300, 300, 3)

imshow (img, 'Source Image')

```

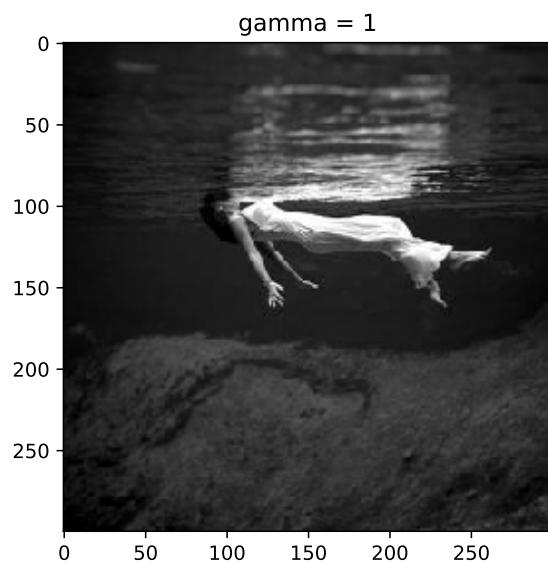
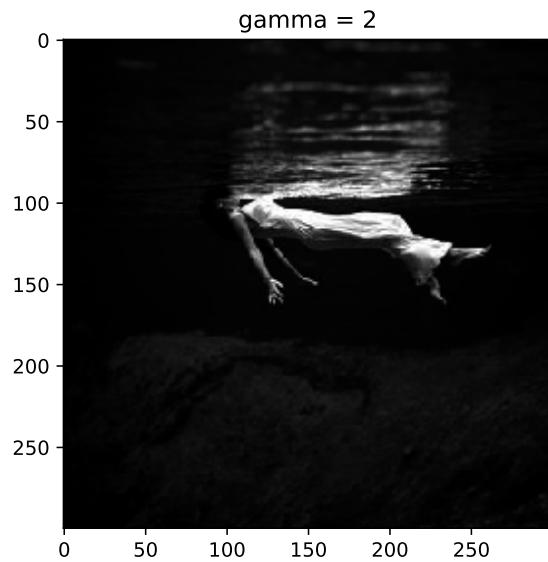


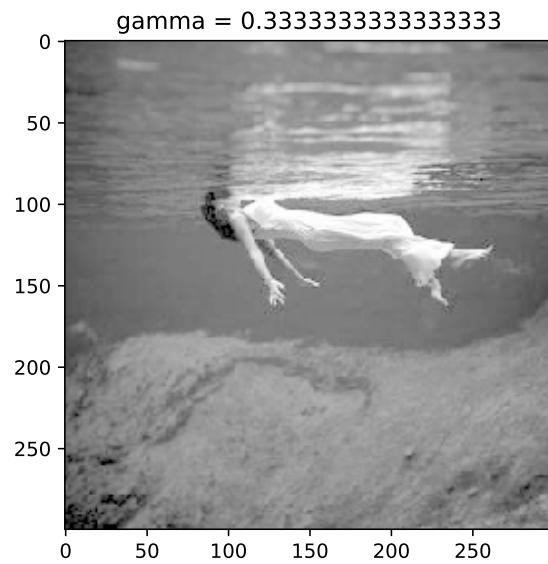
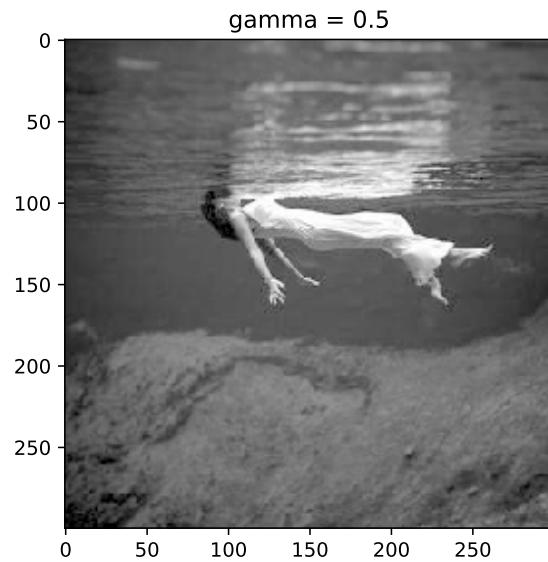
```

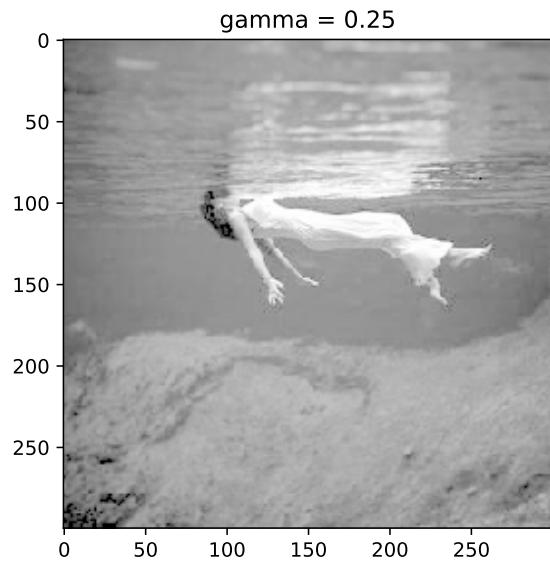
gammas = [2, 1, 1./2, 1./3, 1./4]

for gamma in gammas:
    ii = np.power(img/255., gamma)
    imshow (ii, 'gamma = {}'.format(gamma))
#
# Q. Plot histograms
# EOF

```







Chapter 8

HSV Color Space

HSV color representation is another popular way.

```
# filename: rgb_hsv.py
# REF: http://scikit-image.org/docs/dev/auto_examples/color_exposure/plot_rgb_to_hsv.html

# RGB -> HSV is done with skimage.color.rgb2hsv()
#     * RGB, HSV data are assumed to be in the range [0,1]

import sys
import numpy as np
import cv2
import matplotlib
matplotlib.use ('TkAgg')
import matplotlib.pyplot as plt
import imageio # this will be used to load an image file
import skimage # rgb <-> hsv conversion in [0,1] pixel scale

# show the image
def imshow (img, title=None):
    if img.ndim == 3:
        plt.imshow (img)
    else:
        plt.imshow (img, cmap='gray')

    if title is None: title = 'imshow'
    plt.title (title)
    plt.pause (1)
    plt.close ()

#
```

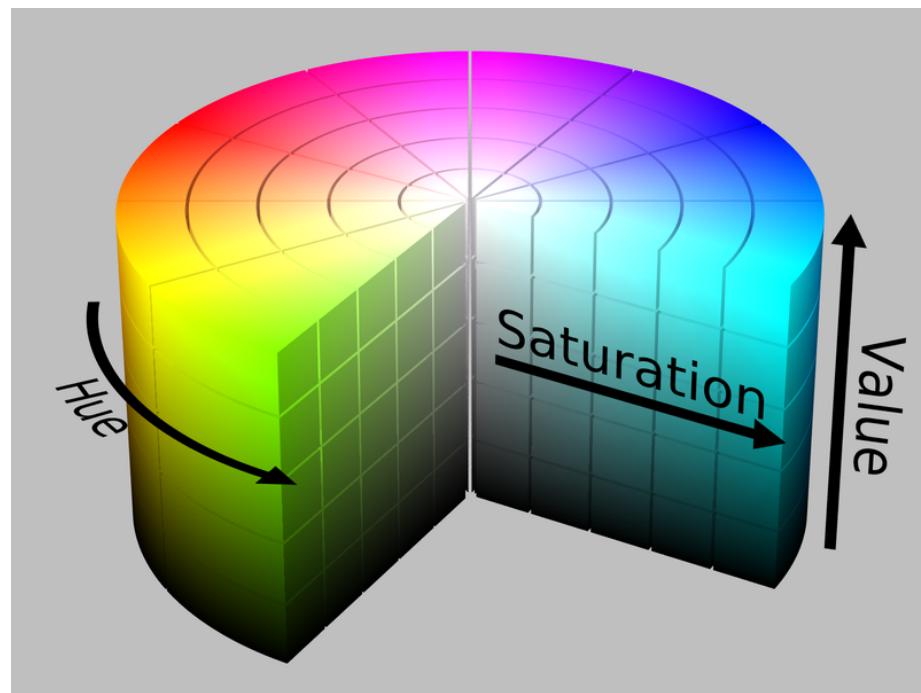


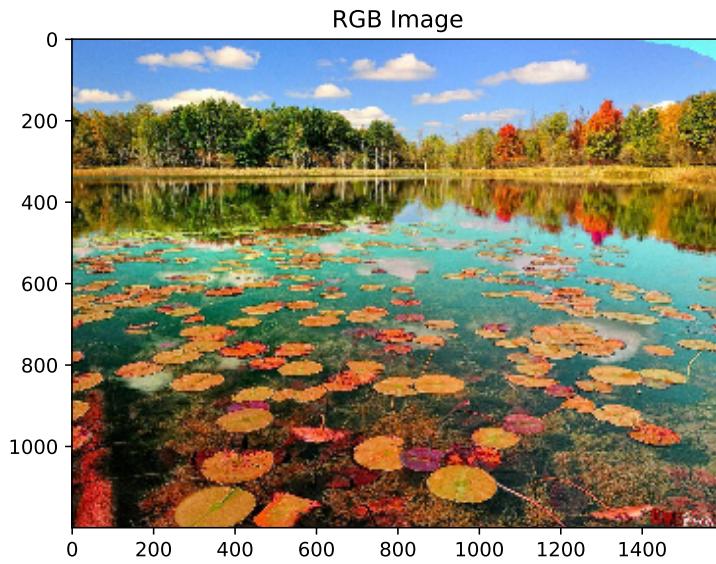
Figure 8.1: HSV Cylinder

```

rgb = imageio.imread ('data/nature.jpg') #https://wallpapercave.com/pretty-nature-wallpapers
if rgb is None:
    print ('image file open error')
    sys.exit ()
#
imshow (rgb, 'RGB Image')

# The conversion assumes an input data range of [0, 1] for all color components.

```

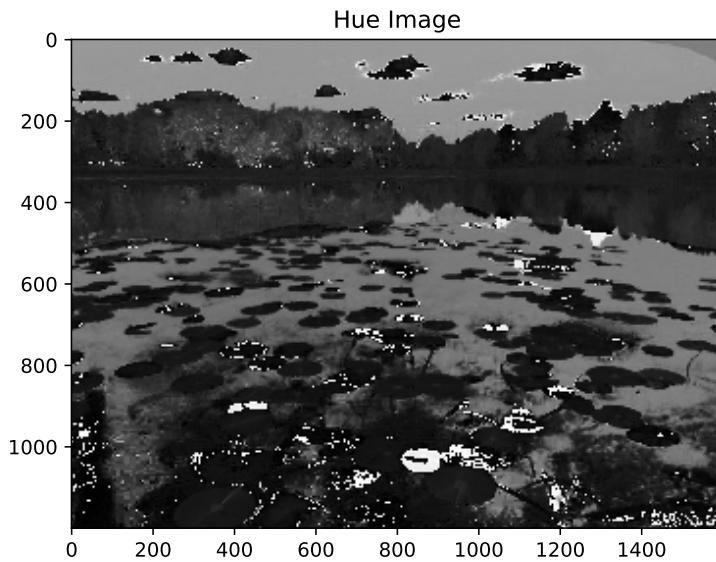


```

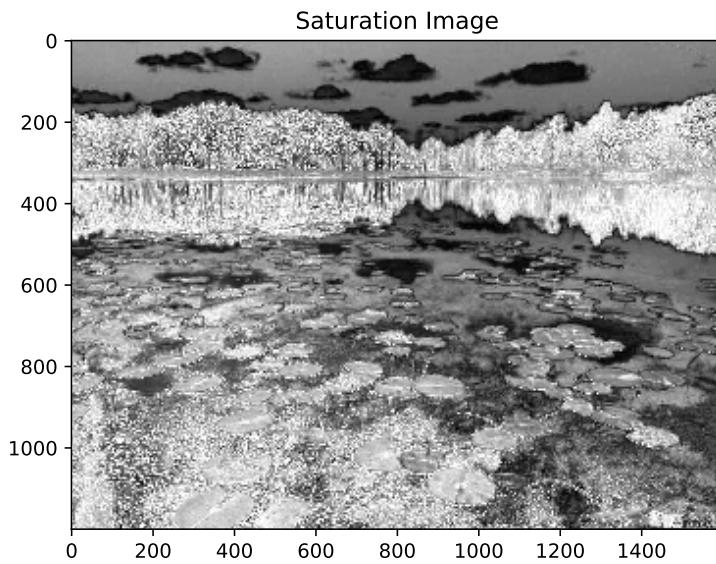
rgb01 = rgb/255.
hsv = skimage.color.rgb2hsv (rgb01)
hue = hsv[:, :, 0]
saturation = hsv[:, :, 1]
value = hsv[:, :, 2]

imshow (hue, 'Hue Image')

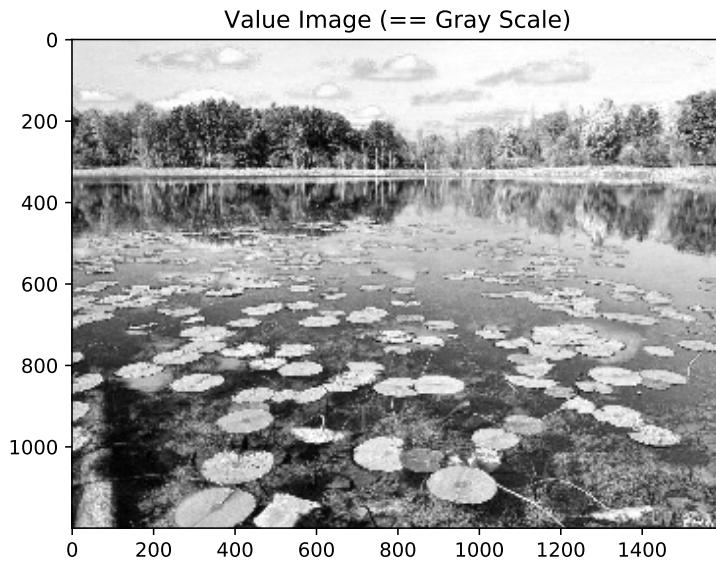
```



```
imshow (saturation, 'Saturation Image')
```



```
imshow (value, 'Value Image (== Gray Scale)')
```



```
print ('HSV(Red)    = {}'.format(skimage.color.rgb2hsv([[1.,0,0]])))

## HSV(Red)    = [[[0. 1. 1.]]]

print ('HSV(Yellow)= {}'.format(skimage.color.rgb2hsv([[1.,1.,0]])))

## HSV(Yellow)= [[[0.16666667 1.           1.           ]]]]

print ('HSV(Green) = {}'.format(skimage.color.rgb2hsv([[0,1.,0]])))

## HSV(Green) = [[[0.33333333 1.           1.           ]]]]

print ('HSV(Blue)   = {}'.format(skimage.color.rgb2hsv([[0,0,1.]])))

## HSV(Blue)   = [[[0.66666667 1.           1.           ]]]]

print ('HSV(White)  = {}'.format(skimage.color.rgb2hsv([[1.,1.,1.]])))

## HSV(White)  = [[[0. 0. 1.]]]
```

```

hsvpixel = [[[0.999, 1., 1.]]]
print ('RGB({}) = '.format(hsvpixel), skimage.color.hsv2rgb (np.array(hsvpixel)))

## RGB([[[0.999, 1.0, 1.0]]]) =  [[[1.      0.      0.006]]]

hsvpixel = [[[0., 1., 1.]]]
print ('RGB({}) = '.format(hsvpixel), skimage.color.hsv2rgb (np.array(hsvpixel)))

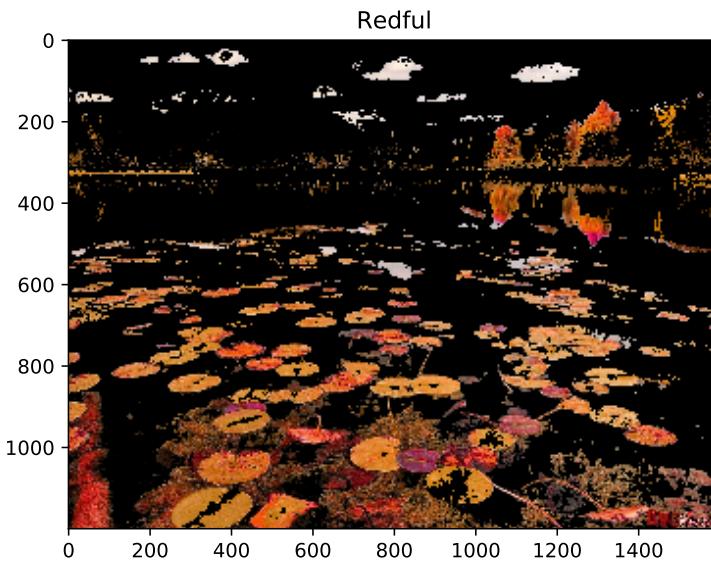
## RGB([[[0.0, 1.0, 1.0]]]) =  [[[1. 0. 0.]]]

print ('Now Extract Redful pixels only.')

## Now Extract Redful pixels only.

for r in range(hsv.shape[0]):
    for c in range (hsv.shape[1]):
        if (hsv[r,c,0] > 0.9 or hsv[r,c,0] < 0.1):
            pass # this is a red
        else:
            hsv[r,c,:] = [0,0,0] # black
#
rgb_redful = skimage.color.hsv2rgb (hsv)
imshow (rgb_redful, 'Redful')

```



```
imageio.imwrite ('data/redful.png', (rgb_redful*255).astype(np.uint8))

# Q. Can you remove white pixels in the clouds?
# Hint: Examine the HSV Cylinder. ( hsv[r,c,1] > 0.5 )

# Q. Rotate the color along the HUE axis, make a video of the chage.
# Step = 0, 0.01, 1

# EOF
```


Chapter 9

Video File Manipulation

9.1 Video File Read/Write

```
# filename: video-open.py
# https://opencv-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_gui/py_drawing_functions.py

import numpy as np
import cv2

video_file = 'data/avideo.mov'
cap = cv2.VideoCapture(video_file)

if cap.isOpened() is False:
    print ('video file open error: ', video_file)
#
width = cap.get (cv2.CAP_PROP_FRAME_WIDTH)
height = cap.get (cv2.CAP_PROP_FRAME_HEIGHT)
nframes = cap.get (cv2.CAP_PROP_FRAME_COUNT)
fps = cap.get (cv2.CAP_PROP_FPS)
print (height, width, nframes, fps)

outvideofile = 'data/outvideo.mov'
out_wh = (640, 480)
outVideo = cv2.VideoWriter (outvideofile, cv2.VideoWriter_fourcc(*'XVID'), 30.0, out_wh)

while(cap.isOpened()):
    ret, frame = cap.read()
    if ret == False:
        break
```

```

frame = cv2.resize(frame, dsize=out_wh)

font = cv2.FONT_HERSHEY_SIMPLEX
text_xy = (100, 200)
frame = cv2.putText (frame, 'OpenCV', text_xy, font, 4, (0,255,255), 2, cv2.LINE_AA)

cv2.imshow('frame', frame)

outVideo.write (frame)

if cv2.waitKey(30) == 27:
    break
#
cap.release()
outVideo.release()

cv2.destroyAllWindows()

# EOF

```

Notes:

1. The color image in opencv is BGR order, not RGB order.
2. `cap.get()` returns `float` numbers, not integer numbers.
3. The video `frame` obtained from `cap.read()` is a `numpy` array, in BGR order. If you want to know its color values at `(x,y)`, then try `frame[y,x]` and you will get the BGR at the location.
4. The fourcc `cv2.VideoWriter_fourcc()` is always confusing. Please search for a concrete explanation on it.
5. There is no way to deal with sound with OpenCV. Try another python module such as `moviepy`, see MoviePy for its documentation. Below is an example:

Try:

- Negative Film Effect Operation for a duration of the video
- Gray Scale
- Reversed-mode play

9.2 Video File Set Frame Position

This is a way of getting a video frame at a specific frame number.

- `videoCaputre().set(cv2.CAP_PROP_POS_FRAMES, nth_frame)`
- `videoCaputre().set(cv2.CAP_PROP_POS_AVI_RATIO, relative_position_0_to_1)`

```
# filename: video-lastframe.py

import sys
import numpy as np
import cv2

video_file = 'data/avideo.mov'
cap = cv2.VideoCapture(video_file)

if cap.isOpened() is False:
    print ('video file open error: ', video_file)
    sys.exit()

#
width = cap.get (cv2.CAP_PROP_FRAME_WIDTH)
height = cap.get (cv2.CAP_PROP_FRAME_HEIGHT)
nframes = cap.get (cv2.CAP_PROP_FRAME_COUNT)
fps = cap.get (cv2.CAP_PROP_FPS)
print (height, width, nframes, fps)

for i in range (int(nframes)):
    fcount = nframes - 1 - i
    cap.set (cv2.CAP_PROP_POS_FRAMES, int(nframes-1-i))
    ret, frame = cap.read()
    if frame is None:
        print ('frame None', i)
        continue
    if ret == False:
        print ('ret False', i)
        continue
    print ('frame read: ', i)

    frame = cv2.resize(frame, dsize=None, fx=.25, fy=.25)

    cv2.imshow('frame', frame)
    if cv2.waitKey(3) == 27:
        break

#
cap.release()
cv2.destroyAllWindows()
# EOF
```

- See: OpenCV VideoCapture Document for `cv2.CV_PROP_POS_FRAMES`

which sets ‘0-based index of the frame to be decoded/captured next.’

Chapter 10

Convolution (Space Filter)

- OpenCV filtering
- Convolution operation: linear filter
- Fast computation should be implemented in C/C++, or use `numpy` with care after its manual.
-

10.1 Smoothing or Blurring

```
# filename: filter-box-blur.py

import time
import numpy as np
import imageio
import skimage
import cv2 # opencv
import matplotlib.pyplot as plt

def imshow (im, title=None, ptime=1):
    if im.ndim == 2:
        plt.imshow (im, cmap='gray')
    elif im.ndim == 3:
        plt.imshow (im)
    else:
        print ('strange image.')
    return
```

```

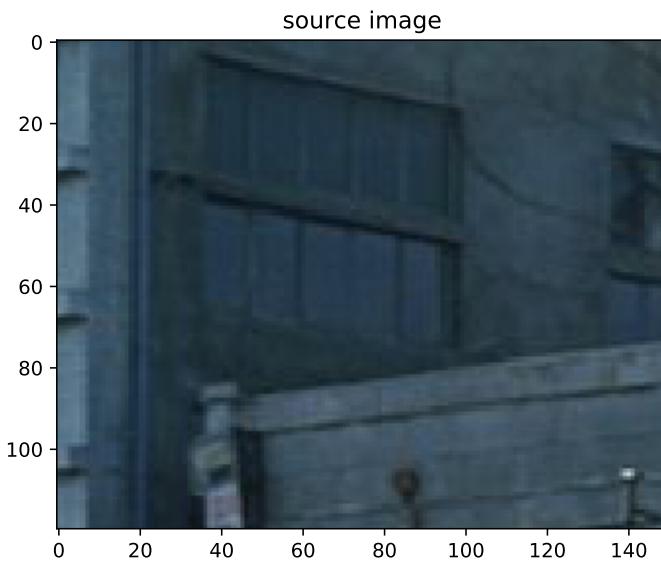
#
if title==None: title='image'
plt.title (title)
plt.pause(ptime)
plt.close()

#
def convolution (im, k):
    dst = np.zeros_like (im)
    for d in range (im.shape[2]):
        for r in range (im.shape[0] - k.shape[0] + 1):
            for c in range (im.shape[1] - k.shape[1] + 1):
                # convolution = inner product at (r,c,d), pivot is at left-top
                inner = 0.
                for ii in range (k.shape[0]):
                    for jj in range (k.shape[1]):
                        inner += im[r+ii, c+jj, d] * k[ii,jj]
                dst[r,c,d] = np.clip(inner+0.5, 0, 255)           # why complicated ?
#
    return dst

#
im = imageio.imread ('data/img536.jpg')      # read an image file
im = im[:120, 0:150, :]

imshow (im, title='source image')

```



```

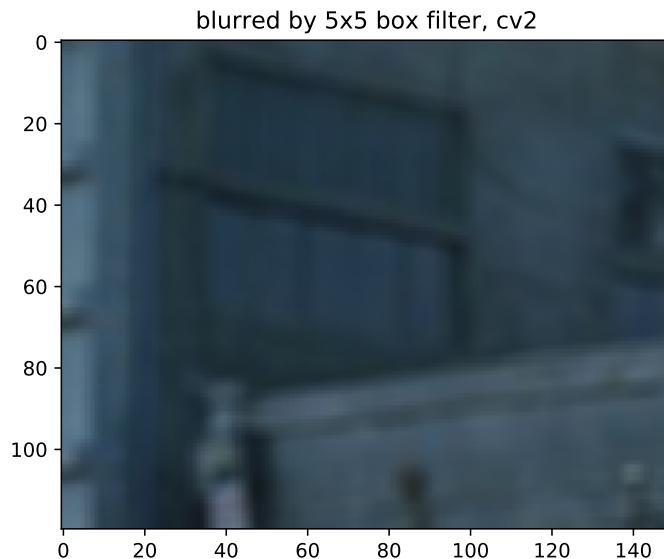
kernel = np.ones((5,5), np.float32) / 25. # filter kernel

t0 = time.time()
blurred = cv2.filter2D (im, -1, kernel)      # filtering
print ('blurred: shape={}, pixel[0,0]={}'.format(blurred.shape, blurred[0,0]), ' %.2f seconds' %

## blurred: shape=(120, 150, 3), pixel[0,0]=[58 80 94] 0.01 seconds

imshow (blurred, 'blurred by 5x5 box filter, cv2')

```



```
print ('Q. How can you produce a motion blurred image?')
```

```

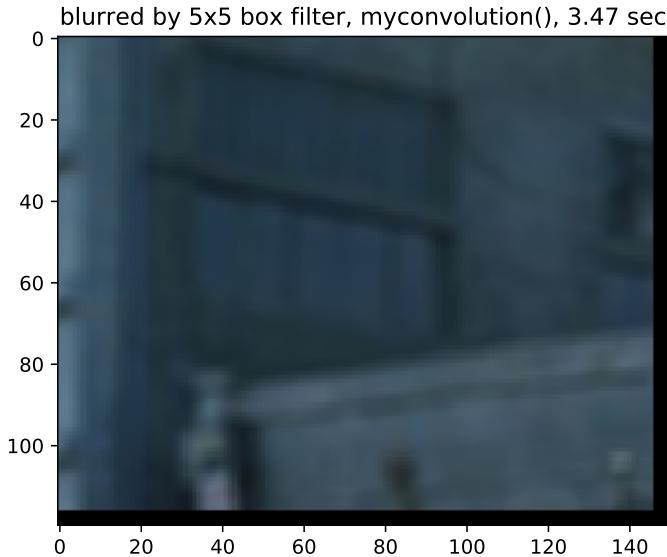
## Q. How can you produce a motion blurred image?

t0 = time.time()
myblurred = convolution (im, kernel)          # filtering of my own
print ('myblurred: shape={}, pixel[0,0]={}'.format(myblurred.shape, myblurred[0,0]), ' %.2f seconds'

## myblurred: shape=(120, 150, 3), pixel[0,0]=[ 62  86 100]  3.47 seconds.

```

```
imshow (myblurred, 'blurred by 5x5 box filter, myconvolution(), %.2f sec' % (time.time() - start_time))
```



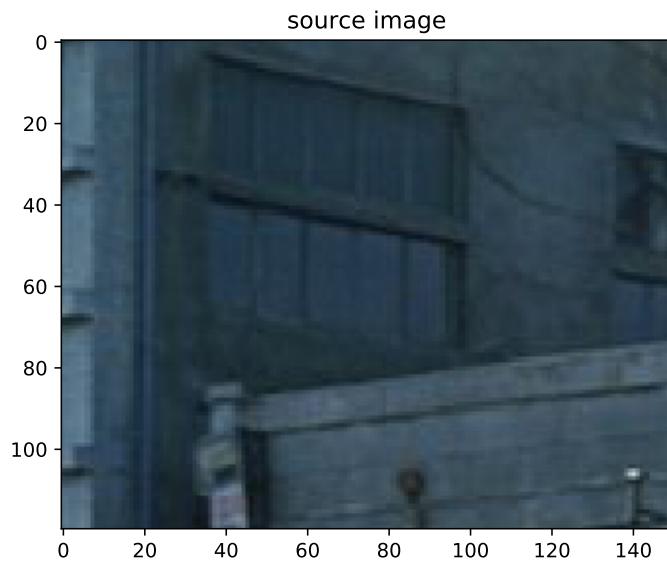
```
print ('Q. Why are the pixel colors different?')  
# Gaussian Blurring
```

```
## Q. Why are the pixel colors different?
```

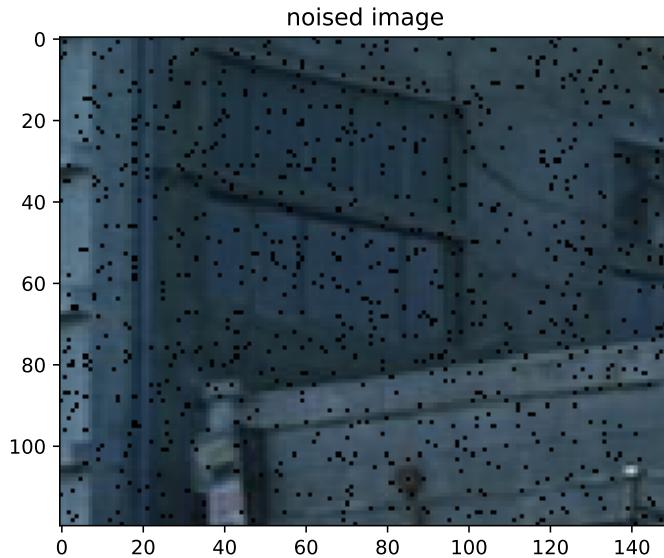
```
blurr_g = cv2.GaussianBlur (im, (5,5), 0) # 0 means automatic 5x5 Gaussian kernel  
imshow (blurr_g, 'Gaussian blur with 5x5')  
  
# Median blur
```



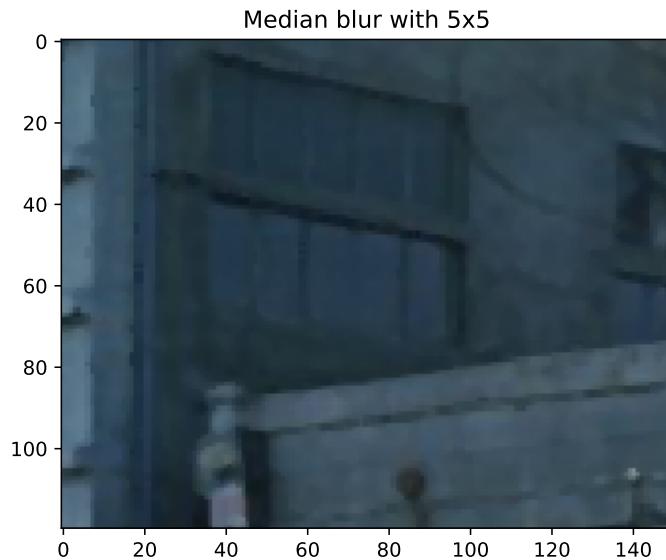
```
im2 = im.copy()
imshow (im2, 'source image')
## speckle noise (black or white)
```



```
for i in range(im.shape[0]*im.shape[1]//10): # 10 % noise
    r = np.random.randint(0, im.shape[0])
    c = np.random.randint(0, im.shape[1])
    if np.random.rand() < 0.5:
        im2[r,c] = 0 # black
    else:
        im[r,c] = 255 # white
#
imshow (im2, 'noised image', ptime=3)
##
```



```
median = cv2.medianBlur (im2, 3)
imshow (median, 'Median blur with 5x5')
```

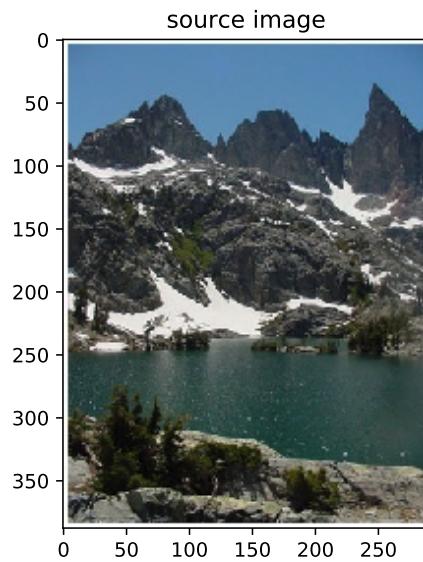


```
print ('Q. Try various sizes and observe differences.')
```

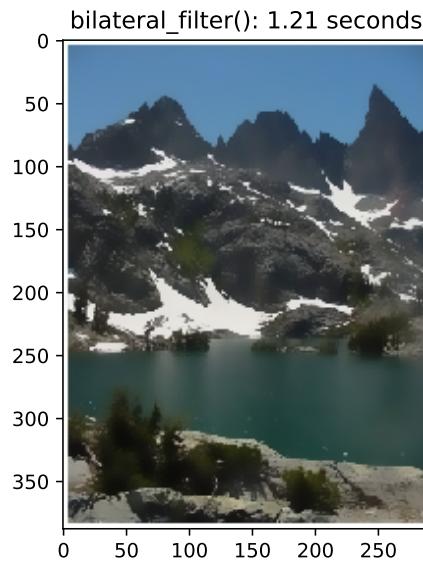
Bilateral Filtering

```
## Q. Try various sizes and observe differences.
```

```
im = imageio.imread ('data/filter_input.jpg')
t0 = time.time()
bf_out = cv2.bilateralFilter (im, 15, 80, 80)
imshow (im, 'source image')
```



```
imshow (bf_out, 'bilateral_filter(): {:.2f} seconds'.format(time.time()-t0))
```



```
print ('Q. Try other parameters for bilateral Filter.')
```

```
# EOF
```

```
## Q. Try other parameters for bilateral Filter.
```


Chapter 11

Morphological Filters

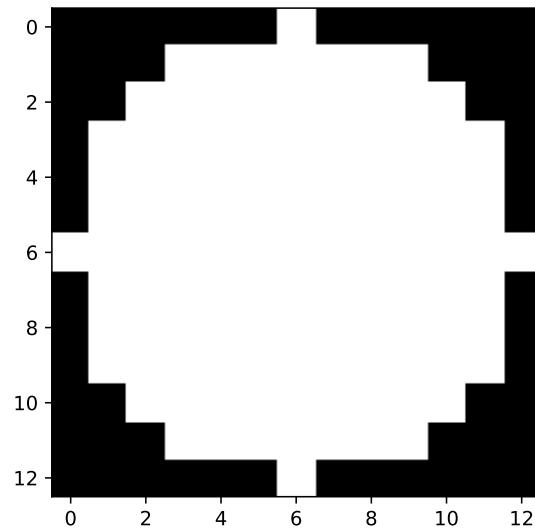
- Dilate = choose max
- Erode = choose min
- see skimage examples

```
import os
import imageio
import matplotlib.pyplot as plt
from skimage.morphology import erosion, dilation, opening, closing, white_tophat
from skimage.morphology import black_tophat, skeletonize, convex_hull_image
from skimage.morphology import disk

selem = disk(6)
print (type(selem), selem.dtype, selem.shape, '\n', selem)

## <class 'numpy.ndarray'> uint8 (13, 13)
## [[0 0 0 0 0 0 1 0 0 0 0 0 0]
## [0 0 0 1 1 1 1 1 1 1 0 0 0]
## [0 0 1 1 1 1 1 1 1 1 1 0 0]
## [0 1 1 1 1 1 1 1 1 1 1 1 0]
## [0 1 1 1 1 1 1 1 1 1 1 1 0]
## [0 1 1 1 1 1 1 1 1 1 1 1 0]
## [1 1 1 1 1 1 1 1 1 1 1 1 1]
## [0 1 1 1 1 1 1 1 1 1 1 1 0]
## [0 1 1 1 1 1 1 1 1 1 1 1 0]
## [0 1 1 1 1 1 1 1 1 1 1 1 0]
## [0 0 1 1 1 1 1 1 1 1 1 1 0]
## [0 0 0 1 1 1 1 1 1 1 1 0 0]
## [0 0 0 0 0 0 0 1 0 0 0 0 0]]
```

```
plt.imshow (selem, cmap='gray')
plt.pause(2)
```



Chapter 12

Gradient & Edge Operation

- Check Tutorial on Sobel Derivatives
 - Linear operators can be exchanged:
 - The gradient of $G(\sigma)$ is approximated in discrete space: Sobel, Prewit, etc.
 - Edge magnitude:
 - Norm (mathematics)
 - L0 Norm, L1 Norm, L-Infinity Norm
1. Sobel X, Y
 2. Sobel Magnitude & Direction
 3. Canny Edge Detector

12.1 Sobel-X

```
# filename: sobel-x.py
# ref: https://docs.opencv.org/3.4.3/d2/d2c/tutorial_sobel_derivatives.html

import cv2
import matplotlib
matplotlib.use ('TkAgg') # my linux machine requires this, but your may not!
import matplotlib.pyplot as plt
import imageio
import skimage
import numpy as np
```

```

def imshow (gray, title=None):
    plt.imshow (gray, cmap='gray')
    if title == None: title='imshow'
    plt.title(title)
    plt.pause(5); plt.close()
#

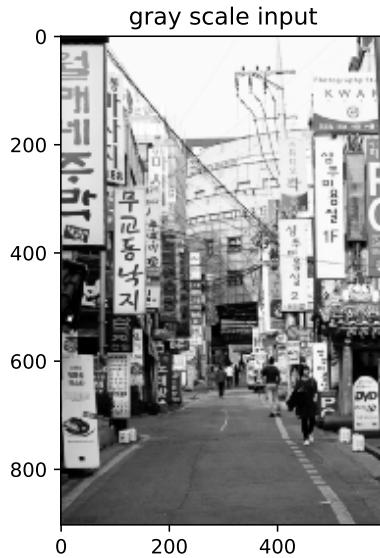
src = imageio.imread ('data/imgKorea012.png')
gray = skimage.color.rgb2gray(src)
gray = skimage.filters.gaussian (gray/255., sigma=1)

grad_x = skimage.filters.sobel_v(gray) # vertical edge comes from x-gradient
grad_y = skimage.filters.sobel_h(gray) # horizontal edge from y-grad
sobel = skimage.filters.sobel(gray) # sobel-magnitude

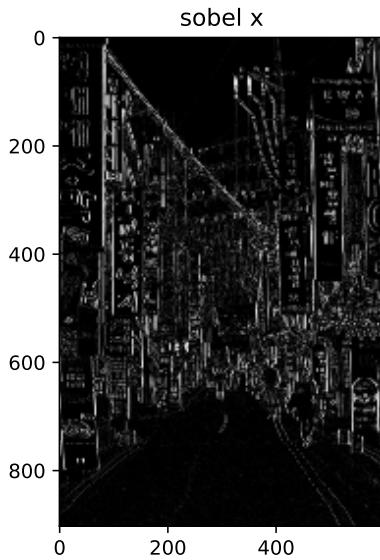
mag = np.sqrt( grad_x**2 + grad_y**2 ) / np.sqrt(2.)

imshow (gray, 'gray scale input')

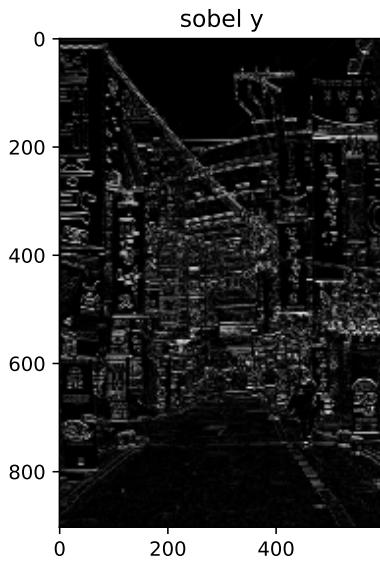
```



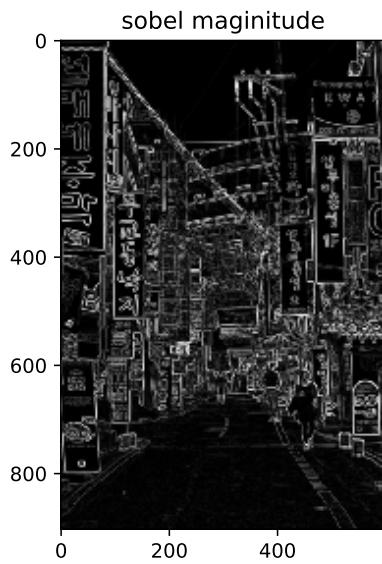
```
imshow (np.abs(grad_x), 'sobel x')
```



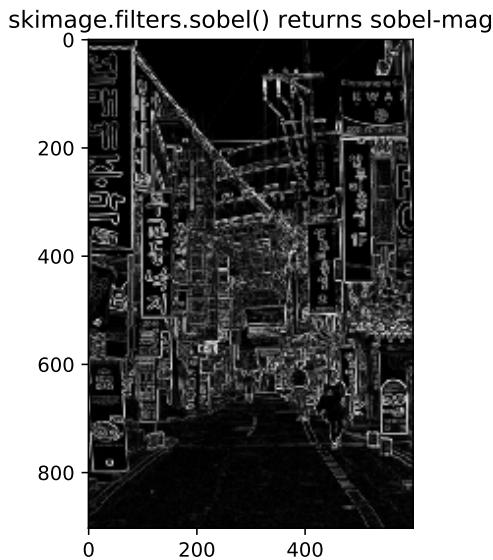
```
imshow (np.abs(grad_y), 'sobel y')
```



```
imshow (mag, 'sobel maginitude')
```



```
imshow (sobel, 'skimage.filters.sobel() returns sobel-mag')  
# EOF
```



Q. Produce edge-enhanced image using a combination of binarization and sobel edge map.

12.2 Canny Edge Detector

- Edge map is extracted. That is, the output contains 0/1 values only (1 for edge pixels).
- The result depends on the value of `sigma` (σ) that controls the width of Gaussian operator for edge detection.

```
# canny-edge.py
# ref: http://scikit-image.org/docs/dev/auto_examples/edges/plot_canny.html#sphx-glr-auto-example
```

```
import numpy as np
import matplotlib
matplotlib.use ('TkAgg') # my linux machine requires this, but your may not!
import matplotlib.pyplot as plt
from scipy import ndimage as ndi
import imageio
import skimage
from skimage import feature
```

```

# Generate noisy image of a square
im = np.zeros((128, 128)) # black image
im[32:-32, 32:-32] = 1 # white square

im = ndi.rotate(im, 15, mode='constant') # rotate
im = ndi.gaussian_filter(im, 4)           # smoothing/blurring
im += 0.2 * np.random(im.shape)          # noise

# Compute the Canny filter for two values of sigma
# output: binary edge map (1: edge pixel, 0: non-edge pixel)
edges1 = feature.canny(im)              # sigma = 1, default
edges2 = feature.canny(im, sigma=3)

# display results
fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3,
                                    figsize=(8, 3),
                                    sharex=True, sharey=True)

ax1.imshow(im, cmap=plt.cm.gray)
ax1.axis('off'); ax1.set_title('noisy image', fontsize=20)

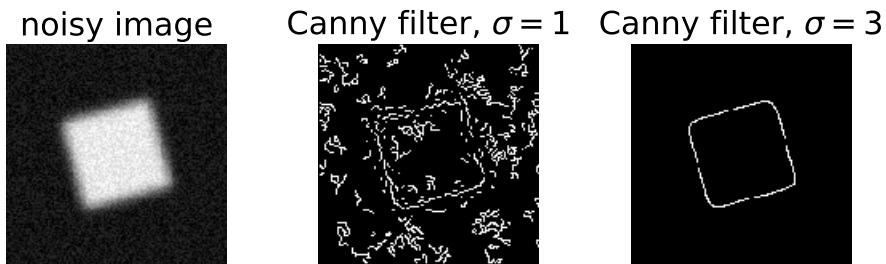
ax2.imshow(edges1, cmap=plt.cm.gray)
ax2.axis('off'); ax2.set_title('Canny filter, $\sigma=1$', fontsize=20)

ax3.imshow(edges2, cmap=plt.cm.gray)
ax3.axis('off'); ax3.set_title('Canny filter, $\sigma=3$', fontsize=20)

fig.tight_layout()
plt.pause(1); plt.close()

# Apply Canny to a natural Image

```



```

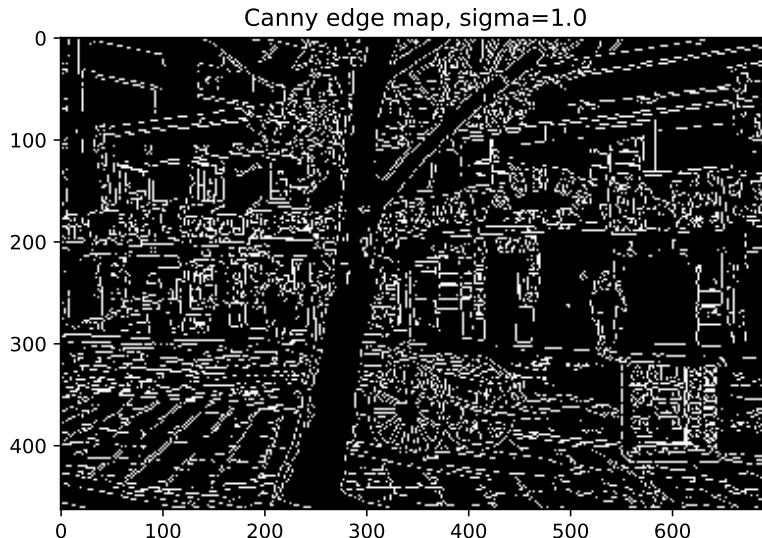
im = imageio.imread('data/img536.jpg')
gray = skimage.color.rgb2gray (im)
sigma = 1.
edge = skimage.feature.canny(gray, sigma=sigma)
print ('edge: ', edge.shape)

## edge: (463, 697)

plt.imshow (edge, cmap='gray')
plt.title ('Canny edge map, sigma={:.1f}'.format(sigma)); plt.pause(1); plt.close()

# Get edge color, just for displaying

```



```

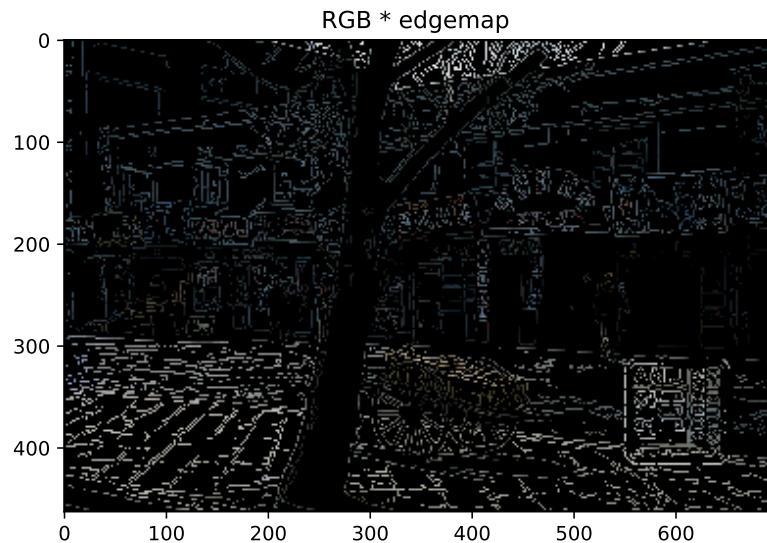
imedge = im.copy()
for i in range(3):
    imedge[:, :, i] = im[:, :, i] * edge
print (imedge.shape)

## (463, 697, 3)

plt.imshow (imedge)
plt.title ('RGB * edgemap'); plt.pause(1); plt.close()

# EOF

```



1. Q. Try various values of `sigma`.
2. Q. Apply morphological operators to the edge map.
3. Q. Try to extract edge's connect data with `cv.findContours()`.

Chapter 13

MoviePy to manipulate movie files

If you need to manipulate audio & video together, `moviepy` is an option.

- Check the site.

```
$ pip3 install moviepy  
$ sudo apt-get install imagemagick
```

13.1 Install youtube-dl

```
$ pip install youtube-dl  
$ youtube-dl <youtube.com url>
```

13.2 Text overlay on a video clip

```
# filename: moviepy_textoverlay.py  
  
# ref: https://zulko.github.io/moviepy/getting_started/quick_presentation.html  
  
# Import everything needed to edit video clips  
from moviepy.editor import *
```

```

# Load a movie file
clip = VideoFileClip("./data/avideo.mov")

# Reduce the audio volume (volume x 0.8)
clip = clip.volumex(0.8)

# Generate a text clip. You can customize the font, color, etc.
txt_clip = TextClip("My Holidays 2013", fontsize=70, color='white')

# Say that you want it to appear at the center of the screen
# The same duration of the video
txt_clip = txt_clip.set_pos('center').set_duration(clip.duration)

# Overlay the text clip on the first video clip
video = CompositeVideoClip([clip, txt_clip])

# Write the result to a file (many options available !)
# 'mp4' was OK.
# Other extensions caused system error:
# Could not load source '</usr/local/lib/python3.6/site-packages/decorator.py:decorator'

video.write_videofile("./data/avideo_overlay_text.mp4")

# EOF

```

- Use ‘mp4’ extension to save to a file (in Linux). No experience with Windows 10 / Mac OS X.

13.3 Extract audio (MP3) from a movie file

```

# filename: moviepy_get_audio.py

from moviepy.editor import *

moviefile = 'data/dooly.mp4'
video = VideoFileClip(moviefile) # open a movie file

audio = video.audio # get the audio part

audiofile = 'data/dooly_audio_extracted.mp3'
audio.write_audiofile(audiofile) # file save

```

```
import os
os.listdir ('data/*.mp3')

# EOF
```

13.4 Movie Concatanation

```
# filename: moviepy_concat.py

from moviepy.editor import VideoFileClip, concatenate_videoclips

clip1 = VideoFileClip("myvideo.mp4")
clip2 = VideoFileClip("myvideo2.mp4").subclip(50,60)
clip3 = VideoFileClip("myvideo3.mp4")

# The result is a concatatanation of the three movie clips.
final_clip = concatenate_videoclips([clip1,clip2,clip3])

final_clip.write_videofile("my_concatenation.mp4")

# EOF
```

13.5 Apply Filter Frame by Frame

```
# filename: moviepy_filter.py

# https://zulko.github.io/moviepy/getting_started/effects.html

# Import everything needed to edit video clips
from moviepy.editor import *
import cv2

# Load a movie file
clip = VideoFileClip("./data/avideo.mov")

# filter
def myFilter (frame):
    # print (frame.flags) # frame is read-only
    # frame: numpy array (rows, cols, 3) of np.uint8 type
    # print (type(frame), frame.dtype, frame.shape)
```

```

# frame[10:20, 20:30,0] = 0 # frame is read-only
edited = frame.copy()
h,w,c = edited.shape
# apply my own filter, remove red & green color, leave blue
edited[h//2-100:h//2+100, w//2-50:w//2+50,0:2] = 0

return edited # return the filtered image
#
# apply filter to every frame
video = clip.fl_image (myFilter)

# save to file
video.write_videofile("./data/avideo_myfilter.mp4")

# EOF

```

13.6 MoviePy Open & Get A Frame

```

#
from moviepy.editor import *
import numpy as np

videofile = 'data/avideo.mov'
movie = VideoFileClip(videofile)
audio = movie.audio
duration = movie.duration # == audio.duration, presented in seconds, float
#note video.fps != audio.fps
print ('video.duration: {} seconds with fps= {}'.format(movie.duration, movie.fps))
print ('audio.duration: ', audio.duration, audio.fps)

step = 0.1
for t in range(int(duration / step)): # runs through audio/video frames obtaining them
    t = t * step
    if t > audio.duration or t > movie.duration: break
    audio_frame = audio.get_frame(t) #numpy array representing mono/stereo values
    video_frame = movie.get_frame(t) #numpy array representing RGB/gray frame

    movie.show(t)
#
# EOF

```

13.7 Errors

- Use ‘mp4’ to save a movie clip into a file! Error:

```
Could not load source '</usr/local/local/lib/python3.6/site-packages/decorator.py:decorator-gen-4'
```

- Edit /etc/ImageMagick-6/policy.xml : comment out (or remove the line that reads) <policy domain="path" rights="none" pattern="@*" />

```
<!-- <policy domain="path" rights="none" pattern="@*" /> -->
```

Error:

```
ImageMagick not detected by moviepy while
```


Chapter 14

k-means clustering

```
# filename: kmeans-colors.py
# yndk@sogang.ac.kr
# ref: https://scikit-learn.org/stable/auto_examples/cluster/plot_color_quantization.html

import collections
import numpy as np
import matplotlib.pyplot as plt
import skimage
from sklearn.cluster import KMeans
from time import time

# Load the Summer Palace photo
china = skimage.io.imread('data/nature-500x375.jpg')
china = china.astype(np.float32) / 255

# Load Image and transform to a 2D numpy array.
assert china.shape[2] == 3 # Color image only
image_array = china.reshape(-1, 3)
print ('image_array_shape = ', image_array.shape)

# Count the number of distinct colors

## image_array_shape = (187500, 3)

colorlist = []
for c in image_array:
    colorlist.append ( (c[0], c[1], c[2]) )
pixelCounter = collections.Counter (colorlist)
```

```

n_src_colors = len(pixelCounter)
print ('The number of distinct colors in src image: ', n_src_colors)

## The number of distinct colors in src image: 139038

mostCommon10 = pixelCounter.most_common(10)
print ('The most frequent color is {}'.format(mostCommon10[0][0]))

## The most frequent color is (0.53333336, 0.74509805, 0.9882353)

plt.imshow(china)
plt.axis('off'); plt.title('Original image ({} colors'.format(n_src_colors))
plt.pause(1); plt.close()

# Apply kmeans clustering

```

Original image (139038) colors



```

print("Perform k-means clustering on {} pixels ... ".format(china.shape[0]*china.shape[1]))
## Perform k-means clustering on 187500 pixels ...

```

```
t0 = time()
n_colors = 10 # the num. of clusters
kmeans = KMeans(n_clusters=n_colors, random_state=0).fit(image_array)
print ('done in %.3f seconds.' % (time() - t0))

# labels is the array of index labels to cluster center
```

done in 4.856 seconds.

```
labels = kmeans.predict(image_array)
print ('labels ({} {}) = {}'.format(type(labels), labels))
```

labels (<class 'numpy.ndarray'>) = [2 2 2 ... 1 5 5]

```
labelsCounter = collections.Counter (labels)

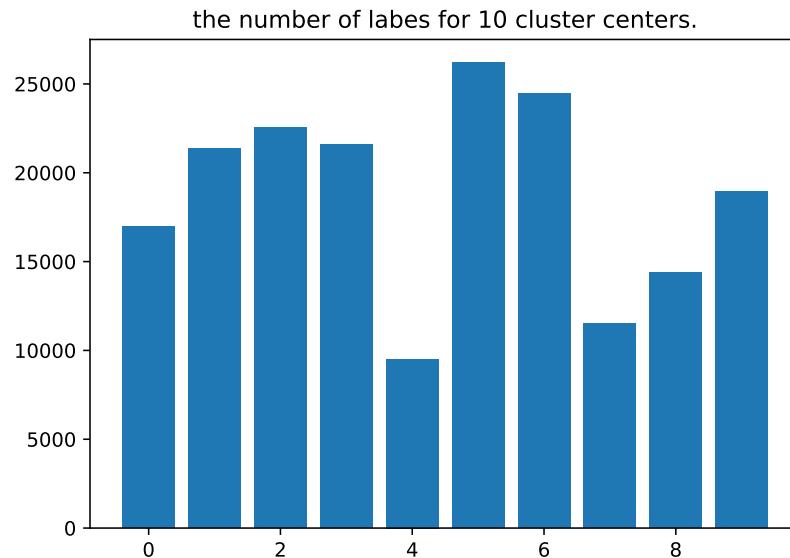
maxLabel = labelsCounter.most_common()
print ('The label of maximum population: ', maxLabel)
```

The label of maximum population: [(5, 26194), (6, 24471), (2, 22540), (3, 21613), (1, 21359),

```
plt.bar (labelsCounter.keys(), labelsCounter.values())
```

<BarContainer object of 10 artists>

```
plt.title ('the number of labes for {} cluster centers.'.format(n_colors))
plt.pause(1); plt.close()
```



```

print ('cluster_centers: \n', kmeans.cluster_centers_)

## cluster_centers:
## [[0.24919401 0.3565876 0.27463475]
## [0.14168221 0.14650837 0.08169955]
## [0.47987813 0.69187105 0.9496904 ]
## [0.80619204 0.4875915 0.19484363]
## [0.8101161 0.8147739 0.8102013 ]
## [0.54372305 0.4375924 0.22541834]
## [0.39751473 0.2721694 0.12270811]
## [0.49875775 0.7651623 0.7272209 ]
## [0.36237583 0.5783887 0.50040054]
## [0.78249586 0.6431192 0.41146764]]

print ('The most common color is ', kmeans.cluster_centers_[maxLabel[0][0]])

## The most common color is [0.54372305 0.4375924 0.22541834]

def recreate_image(codebook, labels, ishape):
    """Recreate the (compressed) image from the code book & labels"""
    image = np.zeros(ishape)
    label_idx = 0
    for i in range(ishape[0]):

```

```

        for j in range(ishape[1]):
            image[i][j] = codebook[labels[label_idx]]
            label_idx += 1
        return image

plt.imshow(recreate_image(kmeans.cluster_centers_, labels, china.shape))
plt.title('Quantized image {} colors by K-Means'.format(n_colors)); plt.axis('off')

## Text(0.5, 1.0, 'Quantized image (10) colors by K-Means')
## (-0.5, 499.5, 374.5, -0.5)

plt.pause(1); plt.close()

# Apply kmeans clustering for k=2

```

Quantized image (10) colors by K-Means



```

print("Perform k-means clustering on {} pixels ... ".format(china.shape[0]*china.shape[1]), end="")

## Perform k-means clustering on 187500 pixels ...

t0 = time()
n_colors = 2 # the num. of clusters

```

```
kmeans = KMeans(n_clusters=n_colors, random_state=0).fit(image_array)
print ('done in {:.3f} seconds.' % (time() - t0))
```

```
# labels is the array of index labels to cluster center
```

```
## done in 0.563 seconds.
```

```
labels = kmeans.predict(image_array)
print ('labels ({}) = {}'.format(type(labels), labels))
```

```
## labels (<class 'numpy.ndarray'>) = [1 1 1 ... 0 0 0]
```

```
labelsCounter = collections.Counter (labels)

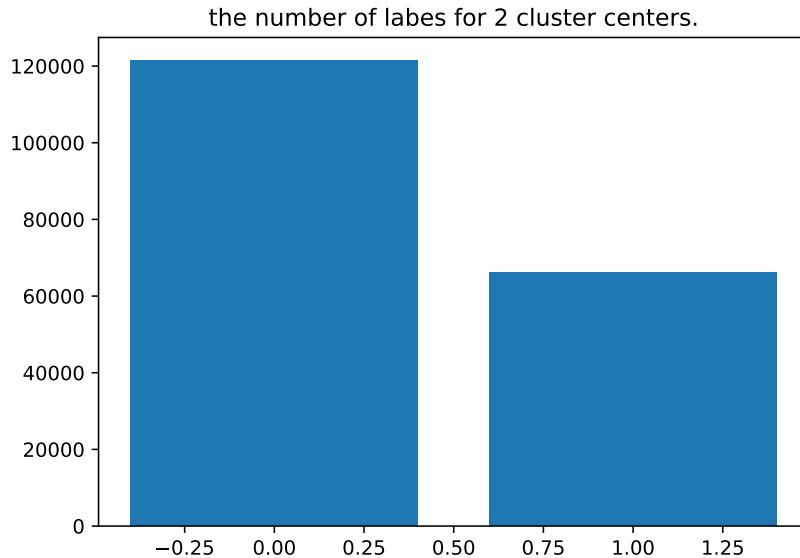
maxLabel = labelsCounter.most_common()
print ('The label of maximum population: ', maxLabel)
```

```
## The label of maximum population: [(0, 121393), (1, 66107)]
```

```
plt.bar (labelsCounter.keys(), labelsCounter.values())
```

```
## <BarContainer object of 2 artists>
```

```
plt.title ('the number of labes for {} cluster centers.'.format(n_colors))
plt.pause(1); plt.close()
```



```

print ('cluster_centers: \n', kmeans.cluster_centers_)

## cluster_centers:
##  [[0.44632792 0.36047846 0.19587587]
##  [0.58648443 0.70717937 0.72987974]]

print ('The most common color is ', kmeans.cluster_centers_[maxLabel[0][0]])

## The most common color is  [0.44632792 0.36047846 0.19587587]

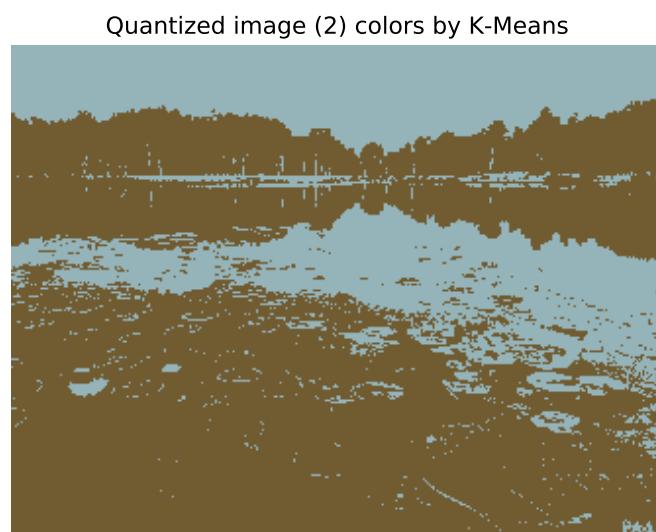
plt.imshow(recreate_image(kmeans.cluster_centers_, labels, china.shape))
plt.title('Quantized image ({}) colors by K-Means'.format(n_colors)); plt.axis('off')

## Text(0.5, 1.0, 'Quantized image (2) colors by K-Means')
## (-0.5, 499.5, 374.5, -0.5)

plt.pause(1); plt.close()

# remove or change `random_state=0` & try the program again
# Its purpose is to get an identical result from KMeans procedure.
# You should remove it in practice.

```



Chapter 15

EOF

Chapter 16

SuperPixel Segmentation

- Check Labeling superpixel colorfulness

Chapter 17

Non-Photorealistic Rendering

17.1 OpenCV non-photorealistic rendering

- Domain Transform for Edge-Aware Image & Video Processing

```
# filename: non-photorealistic.py

# https://www.learnopencv.com/non-photorealistic-rendering-using-opencv-python-c/
# https://docs.opencv.org/trunk/d9/dac/group__photo__render.html

import sys
import numpy as np
import cv2
import matplotlib
matplotlib.use ('TkAgg')
import matplotlib.pyplot as plt
import imageio # this will be used to load an image file
import skimage # rgb <-> hsv conversion in [0,1] pixel scale

# show the image
def imshow (img, title=None):
    if img.ndim == 3:
        plt.imshow (img)
    else:
        plt.imshow (img, cmap='gray')

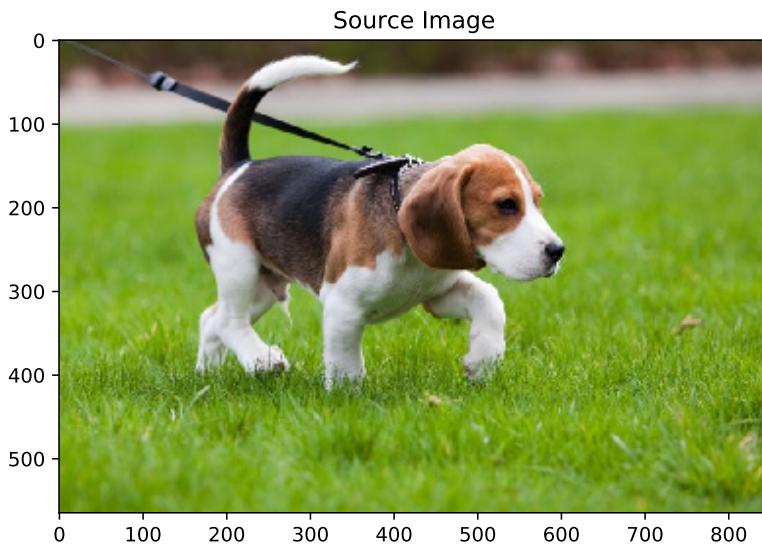
    if title is None: title = 'imshow'
```

```
plt.title (title)
plt.pause (1)
plt.close ()

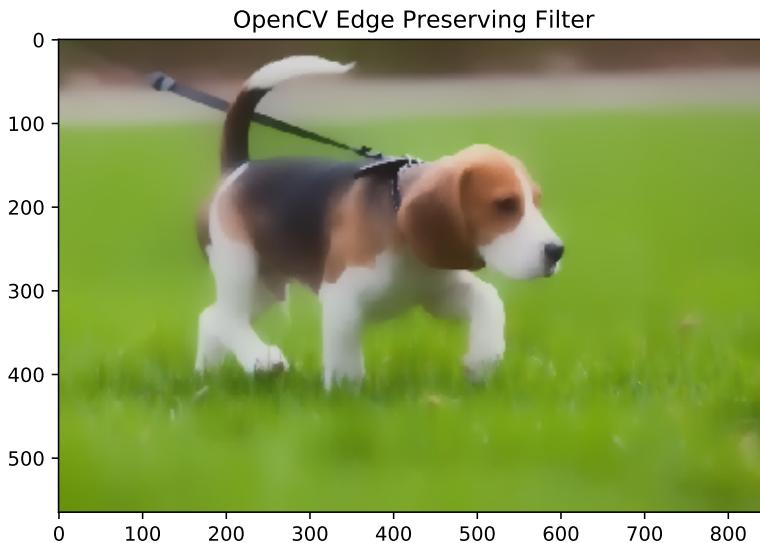
#
src = imageio.imread ('data/petinsider.com.jpg') # it is an RGB format even though ...
if src is None:
    print ('image file open error')
    sys.exit ()
#
print (src.shape)

## (565, 849, 3)

imshow (src, 'Source Image')
```



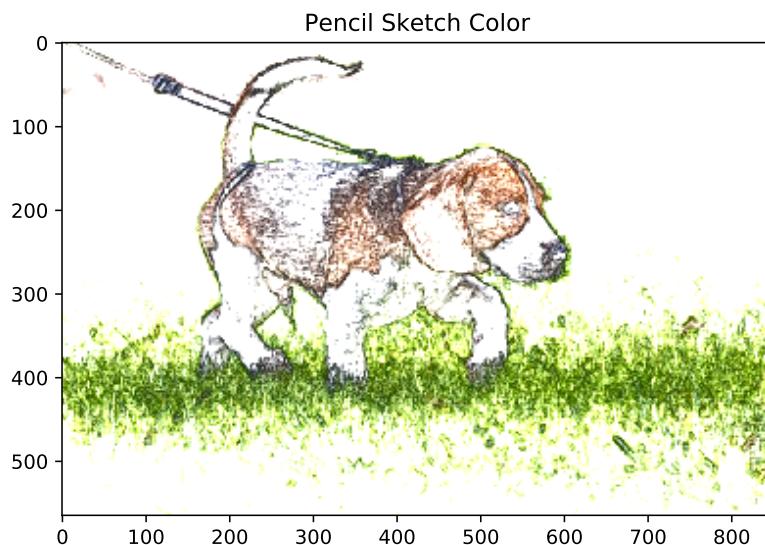
```
epf = cv2.edgePreservingFilter(src, flags=1, sigma_s=60, sigma_r=0.8)
imshow (epf, 'OpenCV Edge Preserving Filter')
```



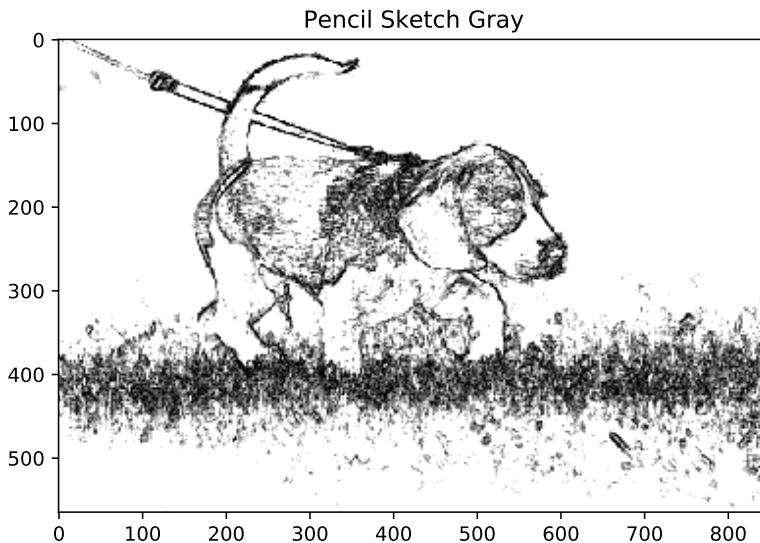
```
detf = cv2.detailEnhance (src, sigma_s = 10, sigma_r=0.2)
imshow (detf, 'Detail Enhancement Filter')
```



```
pencil_g, pencil_c = cv2.pencilSketch (src, sigma_s=60, sigma_r=0.07, shade_factor=0.1)
imshow (pencil_c, 'Pencil Sketch Color')
```

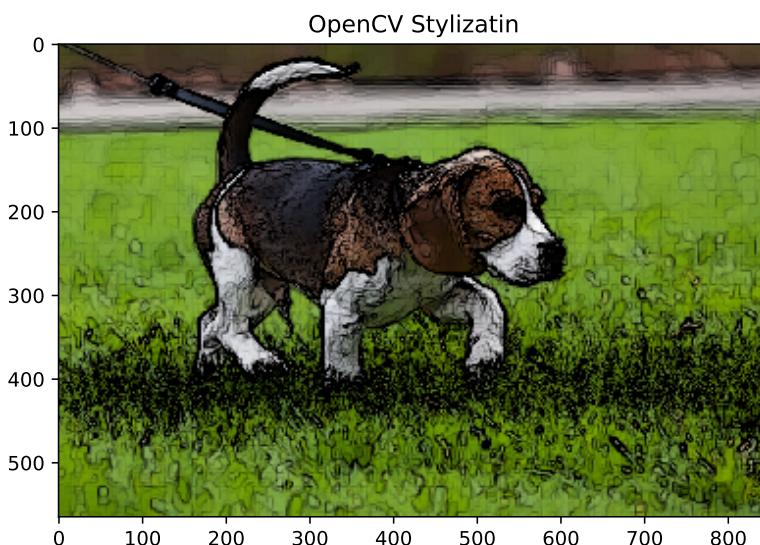


```
imshow (pencil_g, 'Pencil Sketch Gray')
```



```
styl = cv2.stylization (src, sigma_r=0.05, sigma_s=50)
imshow (styl, 'OpenCV Stylizatin')

# EOF
```



17.2 Hand-made cartoon-like filtering

```

import cv2
import numpy as np
import imageio
import matplotlib.pyplot as plt

# show the image
def imshow (img, title=None):
    if img.ndim == 3:
        plt.imshow (img)
    else:
        plt.imshow (img, cmap='gray')

    if title is None: title = 'imshow'
    plt.title (title)
    plt.pause (1)
    plt.close ()

#

img = imageio.imread('data/karakoram-imgur.com.jpg')
if img is None:
    print ('image file open error')
    sys.exit ()
#
print (img.shape)

```

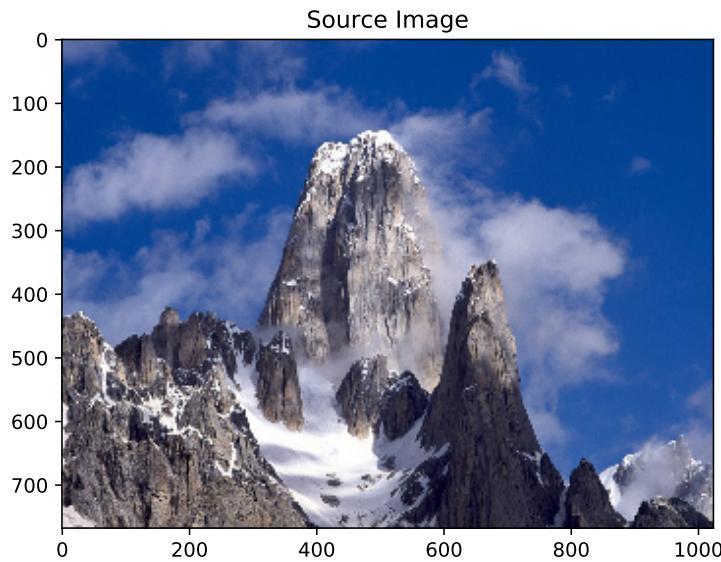
```
## (768, 1024, 3)
```

```

imshow (img, 'Source Image')

# 1) Edges

```

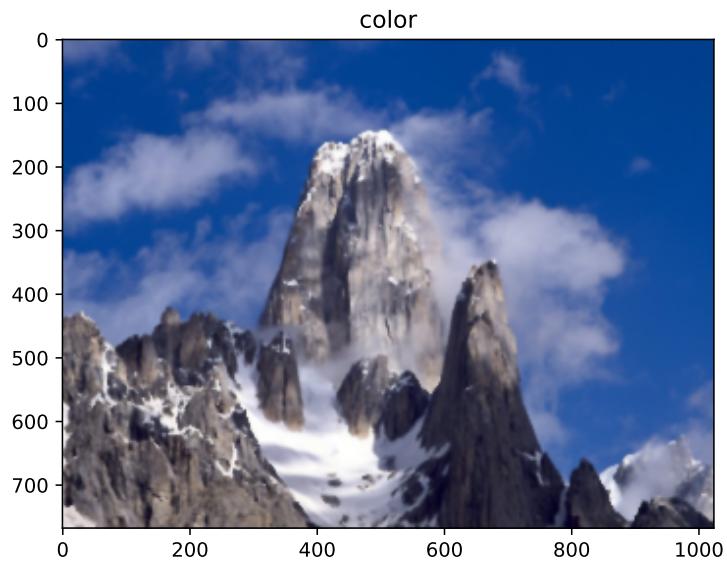


```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
gray = cv2.medianBlur(gray, 5)
edges = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 9, 9)

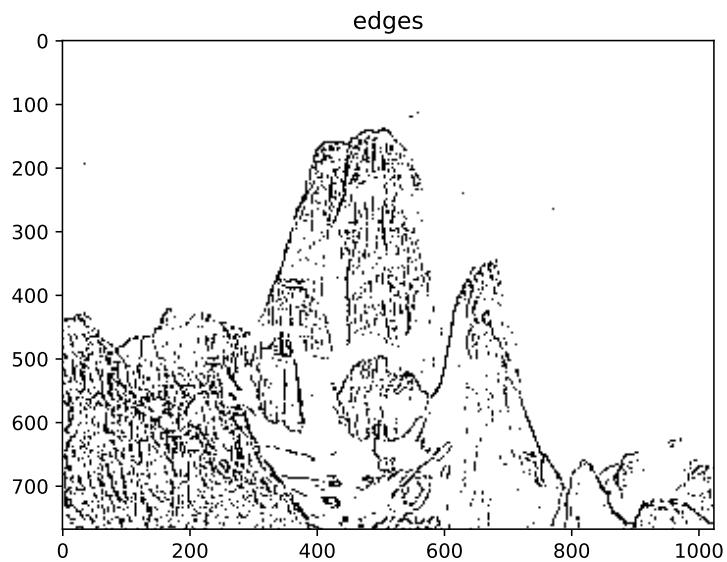
# 2) Color
color = cv2.bilateralFilter(img, 9, 300, 300)

# 3) Cartoon
cartoon = cv2.bitwise_and(color, color, mask=edges)

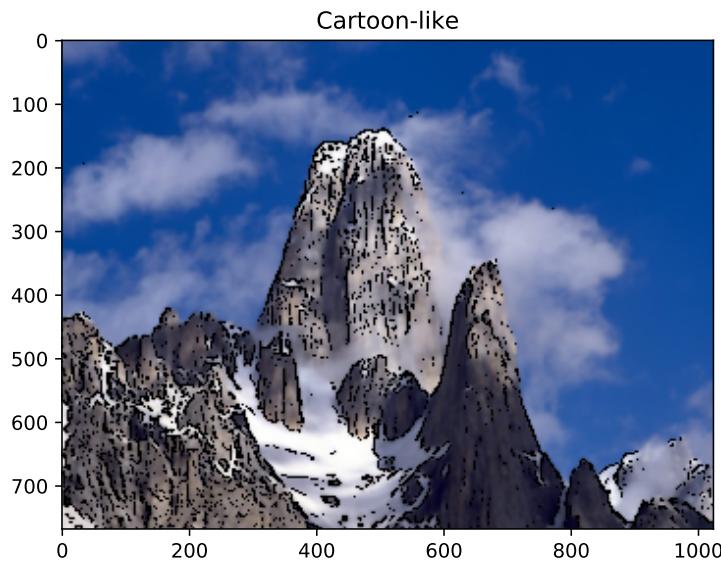
# display
imshow(color, "color")
```



```
imshow(edges, "edges")
```



```
imshow(cartoon, "Cartoon-like")
# EOF
```



Chapter 18

Drawing Text in Image with a Font (OTF) file

- Install Google Noto CJK Fonts
- Use PIL with python
 - `PIL.ImageFont`
 - Drawing anti-aliased unicode text with python
- Computer_font
 - Outline font:
 - * Outline fonts or vector fonts are collections of vector images, consisting of lines and curves defining the boundary of glyphs.
 - * The primary advantage of outline fonts is that, unlike bitmap fonts, they are a set of lines and curves instead of pixels; they can be scaled without causing pixellation.
- OpenType Font
 - ensuring that all of the required glyphs for a document are contained in one cross-platform font file throughout the workflow. Adobe open-type
- GNU Charmap
- OpenGL Text Rendering
- FreeType Library

- FreeType is a freely available software library to render fonts. -It is written in C, designed to be small, efficient, highly customizable, and portable while capable of producing high-quality output (glyph images) of most vector and bitmap font formats.

•

Chapter 19

Photo Mosaic

- python Photomosaic package by Daniel Ballan

19.1 General Procedure

1. Prepare lots of image data. One image can also be used by applying proper image processing techniques such as brightness control, color control, or saturation control.
 2. Define grid size on the source image
 3. For each grid, generate proper thumb nail image.
 4. Save the output.
- What kind of image processing or treatment can be thought of to make an interesting photomosaic?

19.2 References