بسـ الله ــم الرحــمن الراحــيــم

Image Processing

By: Mahmoud Zaky AttaALLAH, B.Sc.
Demo. @ MUFIC FCI

Chapter 3 : Pixels

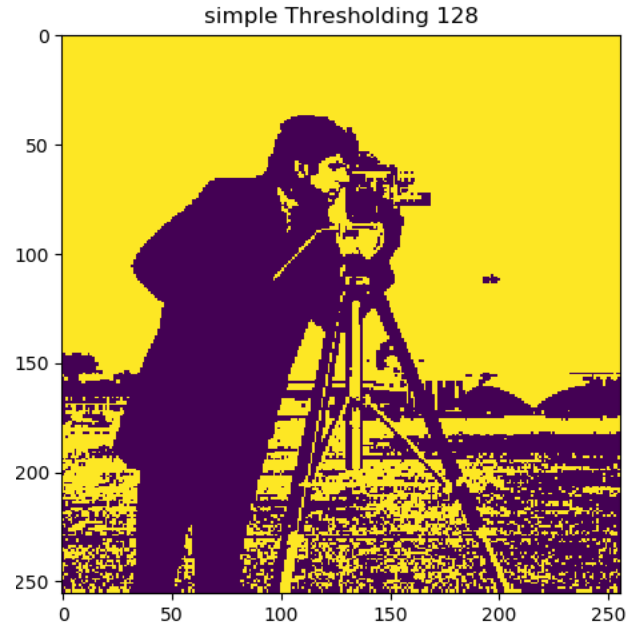
- Image Thresholding (Python Implementation)

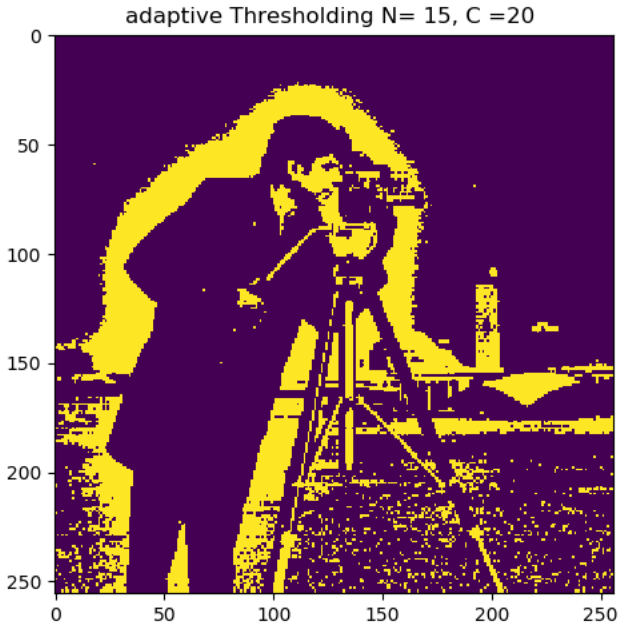
```
import skimage.filters as filter
import matplotlib.pyplot as plt
import scipy.ndimage as ndi
import numpy as np


"""
Mahmoud Zaky AttaALLAH, B.Sc.
matlab Image processing
Chapter3 Tasks

"""


def simple_thresholding( im,T = 128):
    if im.max() <= 1 :
        im = im*255
    return im > T

def simple_adaptive_threshold(im, N= 15, C= 20 ) :
    if im.max() <= 1 :
        im = np.asarray(im*255)
    smoothed = ndi.gaussian_filter(im, N)
    smoothed += C
    return ( im - smoothed ) > 0
```

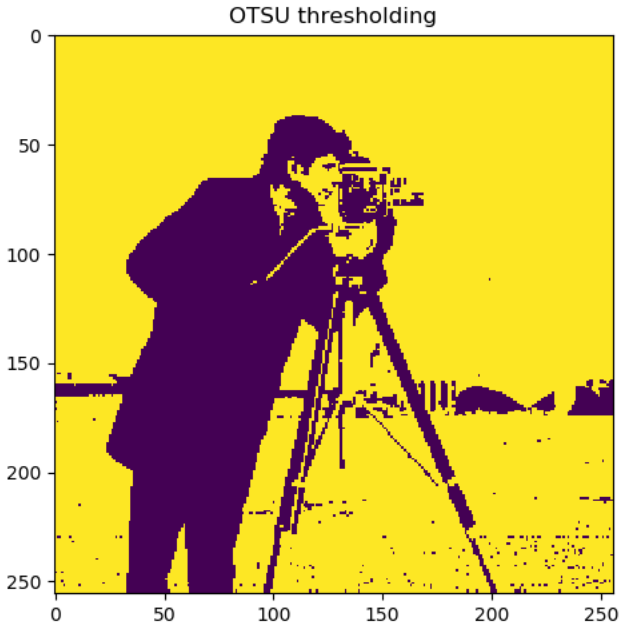
```
def otsu_threshold(im):
    T = filter.threshold_otsu(im)
    return im > T

im = plt.imread('cameraman.png')
plt.subplot(1,4,1)
plt.imshow(im)

for i, func in enumerate([simple_thresholding,
                          simple_adaptive_threshold,
                          otsu_threshold ]):
    plt.subplot(1,4,i+2); plt.imshow(func(im))
```

Chapter 3 : Pixels

- Image Thresholding (Python Implementation)







Chapter 3 : Pixels

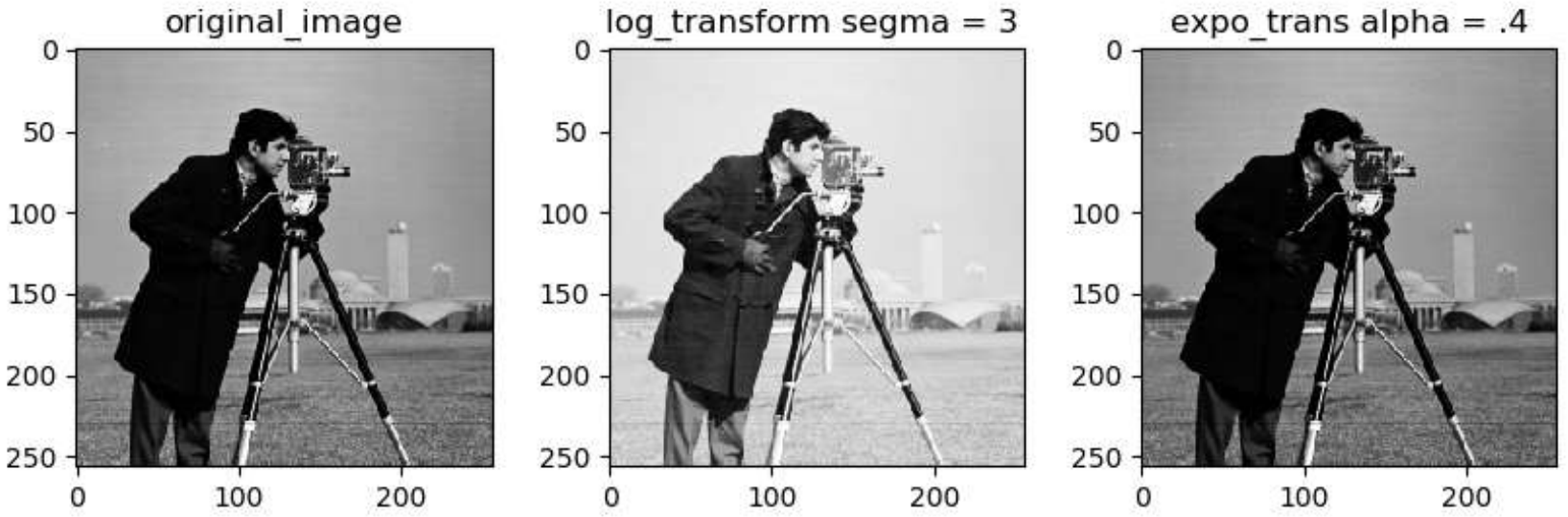
- Image enhancement (Python Implementation)

```
import matplotlib.pyplot as plt
import numpy as np
from skimage import io, exposure



def log_trans(im, sig = 3):
    if im.max() <= 1:
        im = im*255
    C = 255 / np.log10(1+im.max() )
    return C * np.log(1+(np.e ** sig -1 ) * im )

def expo_trans(im, alpha = .4, c = 1) :
    return c * ((1+alpha)** im -1 )

def gamma_trans(im, gamma= 1.5, c= 2):
    return c*(im)**gamma

def contrast_stretching(im, a = 150 ,b= 200):
    return (im - im.min()) *((b-a)/(im.max() - im.min())) +a

def histo_equaliz(im):
    return exposure.equalize_hist(im)
```

```
def get_histo(im) :
    return np.histogram(im, 255)[0]

def imhistplt(im) :
    plt.plot(get_histo(im) )

im = plt.imread('cameraman.png')

plt.subplot(1,7,1)
plt.imshow(im, cmap= 'gray')
for i,func in enumerate([log_trans,expo_trans,
                         gamma_trans,
                         contrast_stretching,
                         histo_equaliz ]):
    out = func(im)
    plt.subplot(1, 7, i+2)
    plt.imshow( out, cmap= 'gray')

plt.subplot(1, 7, 7)
imhistplt(im)
```

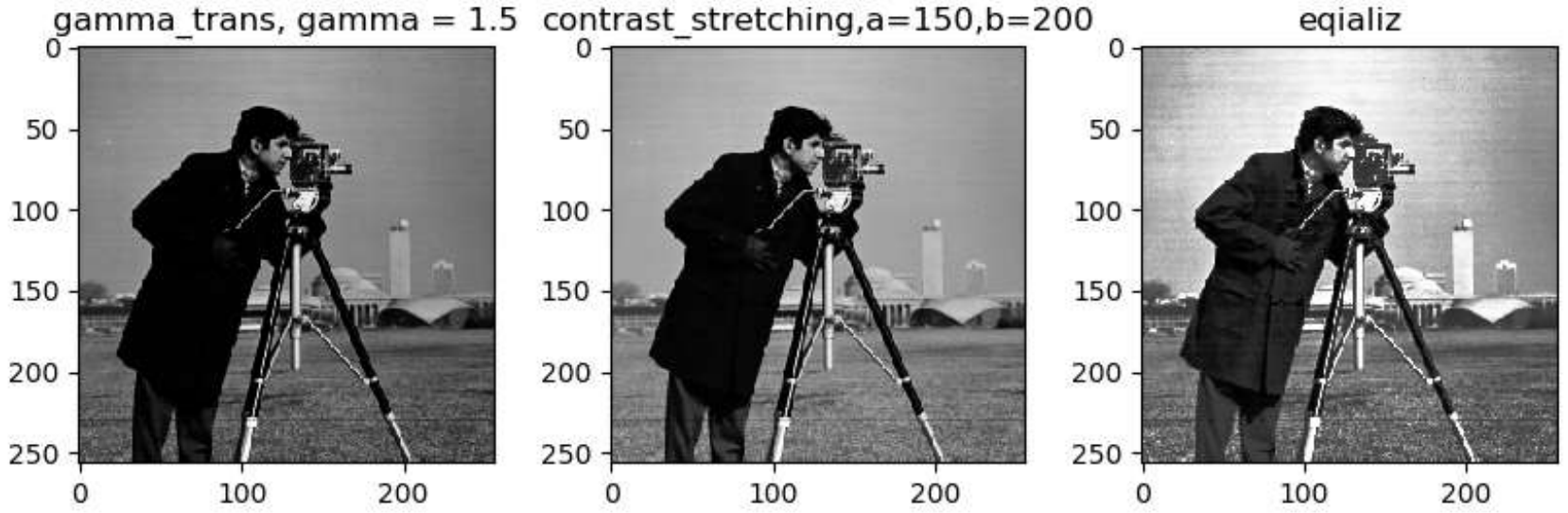
Chapter 3 : Pixels

- Image enhancement (Python Implementation)



Chapter 3 : Pixels

- Image enhancement (Python Implementation)



Chapter 4 : Enhancement

- Filters(Python implementation)

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

def mean_mask(n, m):
    ma = np.zeros((n, m), dtype=np.float)
    ma = ma + 1 / (m * n)
    return ma

def convolv(im, ma=mean_mask(3, 3)):
    ret = np.zeros_like(im,dtype = np.float)
    r, c = im.shape
    mr, mc = [x // 2 for x in ma.shape]

    for i in range(mr, r - mr):
        for j in range(mc, c - mc):
            ret[i, j] = (np.sum(im[i - mr:i + mr + 1
                                   , j - mc: j + mc + 1]*ma))
    return ret

def med(im):
    ma = np.ones((3, 3))
    ret = np.zeros_like(im);
    r, c = im.shape
    mr, mc = [x // 2 for x in ma.shape]

    for i in range(mr, r - mr):
        for j in range(mc, c - mc):
            ret[i, j] = np.median(im[i - mr:i + mr + 1,
                                     j - mc: j + mc + 1]*ma);
    return ret
```

```
def guass(n, m, sig=1):
    def G(n, m):
        return (1 / (2 * np.pi * sig ** 2))
        * np.e ** (-(n ** 2 + m ** 2) / 2 * sig ** 2)

    ma = np.zeros((n, m));
    ns, ms = n // 2, m // 2
    for i in range(n):
        for j in range(m):
            ma[i, j] = G(i - ns, j - ms);
        return ma


def unshapping_filer(im):
    return im + (im - convolv(im, guass(3, 3)))


def smoth(I):
    return convolv(I, guass(5, 5, 2))


def unity(im):
    return im


# mask = np.ones( (7,7) ) * 1/49 ;
mask = guass(3, 3)

im = plt.imread('cameraman.png')
im = np.asarray(im, dtype=np.float) / im.max()
funcs = [unity, convolv, med, smoth, unshapping_filer]
for i, func in enumerate(funcs):
    plt.subplot(1, len(funcs), i + 1)
    plt.imshow(func(im), cmap='gray')
```

Chapter 4 : Enhancement

- Filters(Python implementation)

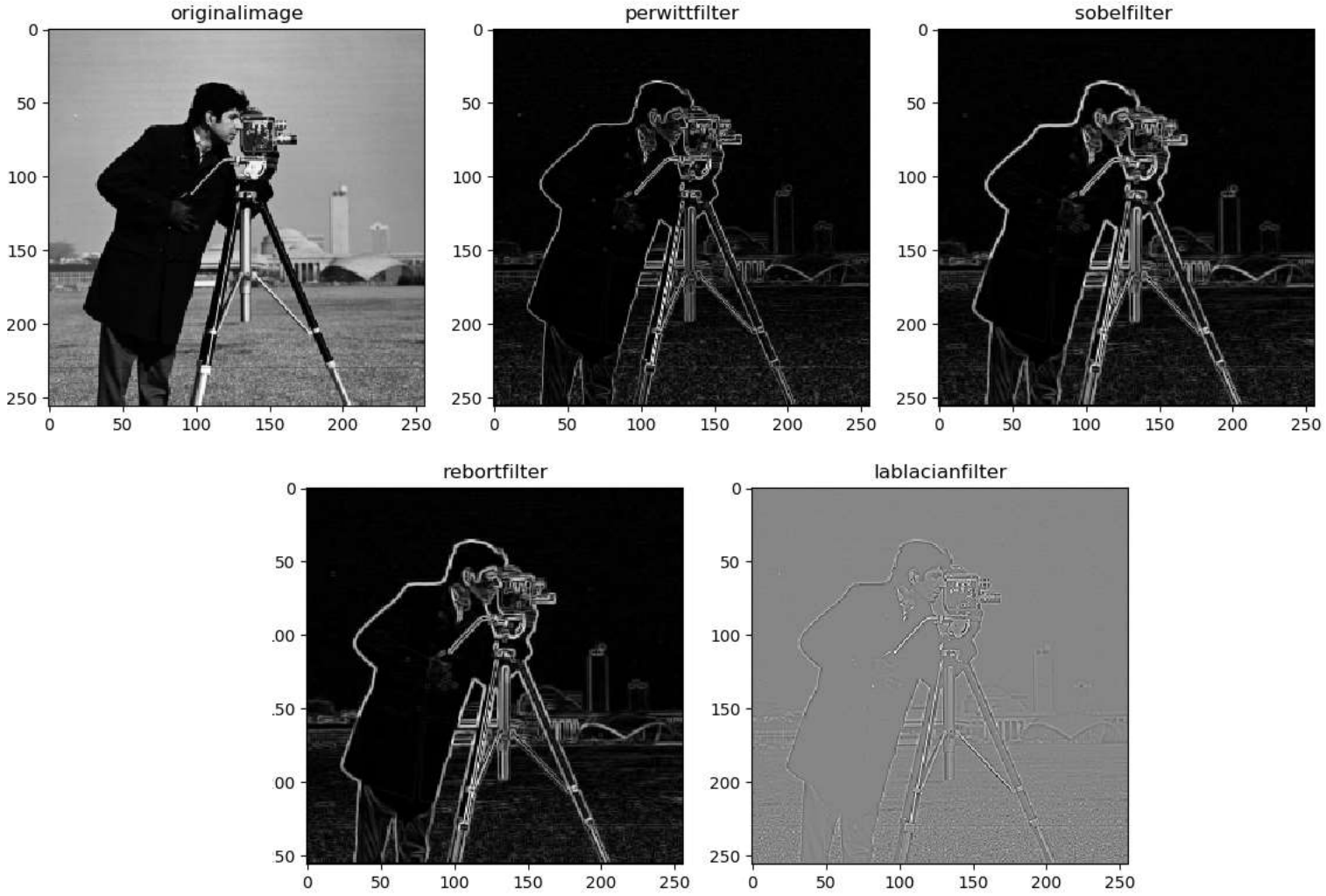
Chapter 4 : Enhancement

- Edge detection(python Implementation)

```
import matplotlib.pyplot as plt
import skimage.filters as filt
import numpy as np



im = plt.imread('cameraman.png')
edge_det = [lambda x: x, filt.roberts,
            filt.sobel, filt.prewitt,
            filt.laplace]
for i, func in enumerate(edge_det):
    plt.subplot(1,len(edge_det), i+1 )
    plt.imshow(func(im), cmap= 'gray')
```

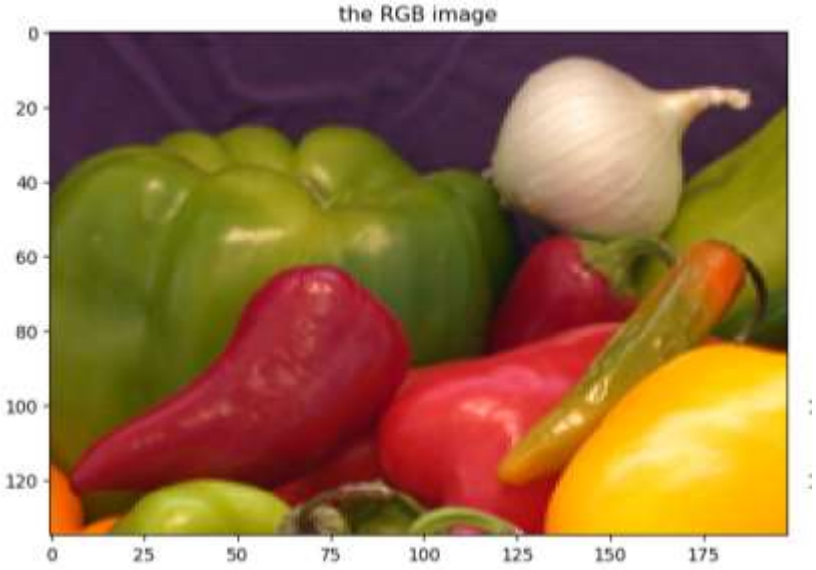
Chapter 4 : Enhancement

- Edge detection(python Implementation)

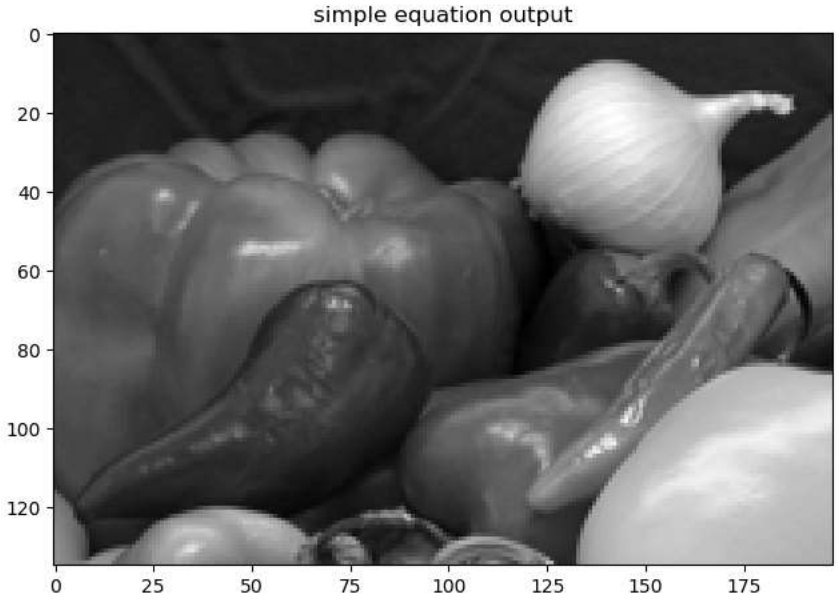


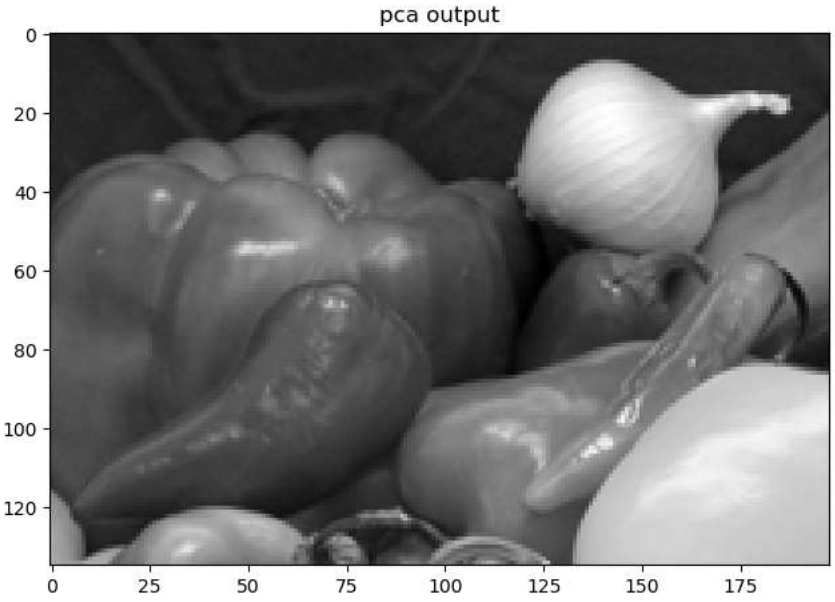
RGB2Gray Task (Python)

```
import numpy as np
import sklearn.decomposition as deco
import matplotlib.pyplot as plt

def simple_rgb2gray(im):
    return .29* im[:,:,0]
        + .58*im[:,:,1] + .11*im[:,:,2]

def pca_gray( im ):
    df = np.zeros((im[:, :, 0].size, 3))

    for i in range(3):
        df[:, i] = im[:, :, i].flatten()

    pca = deco.PCA(1)
    d1 = pca.fit_transform(df)
    return d1.reshape(im[:,:,0].shape)

im  = plt.imread('onion.png')

plt.subplot(1,3,1)
plt.imshow(im)
plt.subplot(1,3,2)
plt.imshow(simple_rgb2gray(im),cmap = 'gray')
plt.subplot(1,3,3)
plt.imshow(pca_gray(im),cmap = 'gray')
```



RGB2Gray Task (Python)





Novel Binarization method

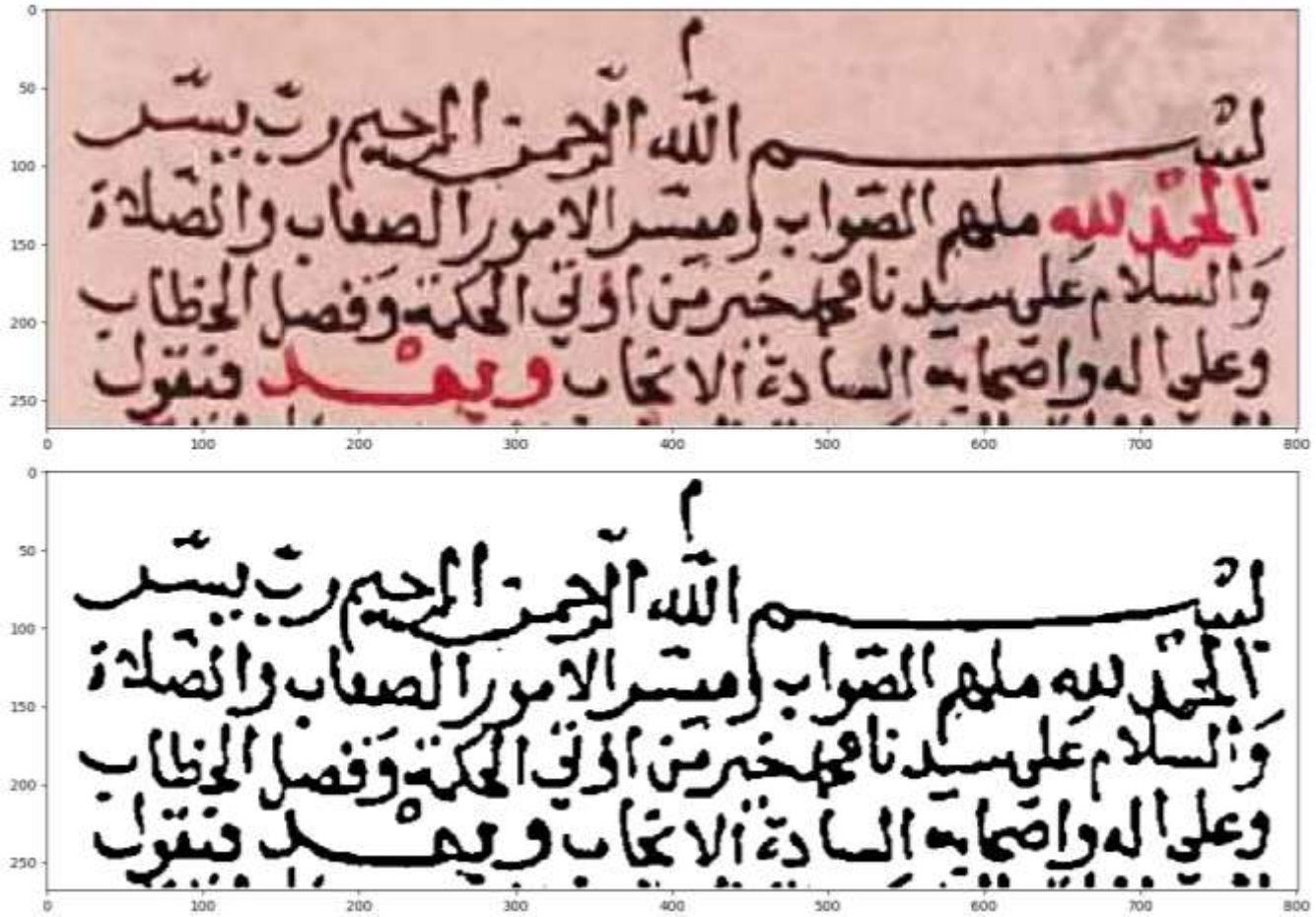
```
import scipy.signal.signaltools as sig
import skimage.filters as filters
import matplotlib.pyplot as plt
import scipy.ndimage as ndim
import numpy as np

Windo = (3,3)

def im2gray(im):
    if len(im.shape) != 3 :
        return im
    return np.dot(im, [0.2989, 0.5870, 0.1140] )

def normalize( im ) :
    min, max = im.max(), im.min()
    return (im - min ) / (max - min)

def get_image_gradiant(im):
    dx = ndim.sobel(im, axis = 0 )
    dy = ndim.sobel(im, axis = 1)
    return np.hypot(dx, dy) # the secondNorme distance

def NS_transform( im ) :
    T = normalize( im )
    F = 1 - T
    I = 1 - normalize( np.abs( get_image_gradiant( im )) )
    return T, I, F
```

Novel Binarization method

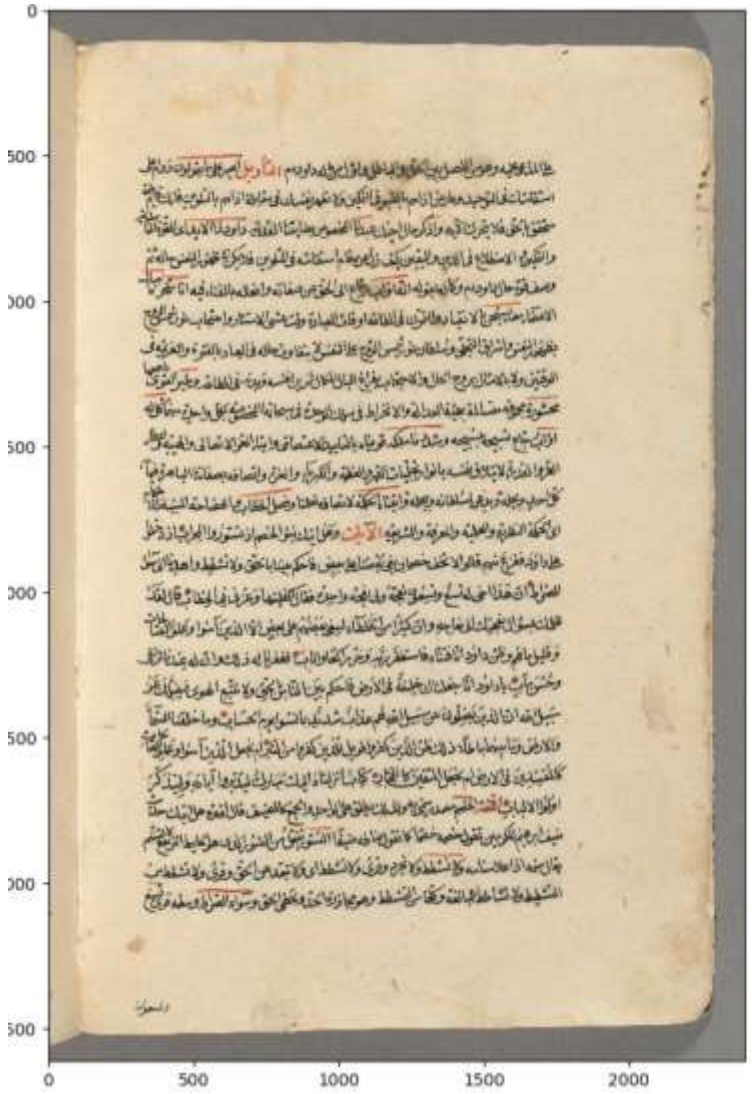
```
def entropy(im):
    flatten = im.flatten()
    value, count = np.unique(flatten,return_counts=True)
    acc = np.sum( count )
    return - np.sum([ x / acc * np.log( x / acc ) for x in count ] )

def mean_filter( im ):
    kernel = np.ones(Windo)/9
    return ndim.convolve(im, kernel )

def NS_alpha_mean(T, I, F, alpha = .8 ):
    T_bar = mean_filter(T)
    mask = I < alpha
    T_bar[mask] = 0
    addt = T.copy() ; addt[~mask] = 0
    T_bar = T_bar + addt
    return NS_transform(T_bar)



def Proposed_method(im):
    gry_im = im2gray(im )/255
    filtered_im = sig.wiener( gry_im, Windo )
    T, I, F = NS_transform(filtered_im)
    alpha = .001 ; pre_entr = entropy( I )
    while True :
        T, I, F = NS_alpha_mean(T, I, F)
        entr = entropy( I )
        print( entr )
        if (entr - pre_entr ) / pre_entr < alpha :
            break
        pre_entr = entr
    return filters.median((T  > filters.threshold_sauvola(T) ))
```

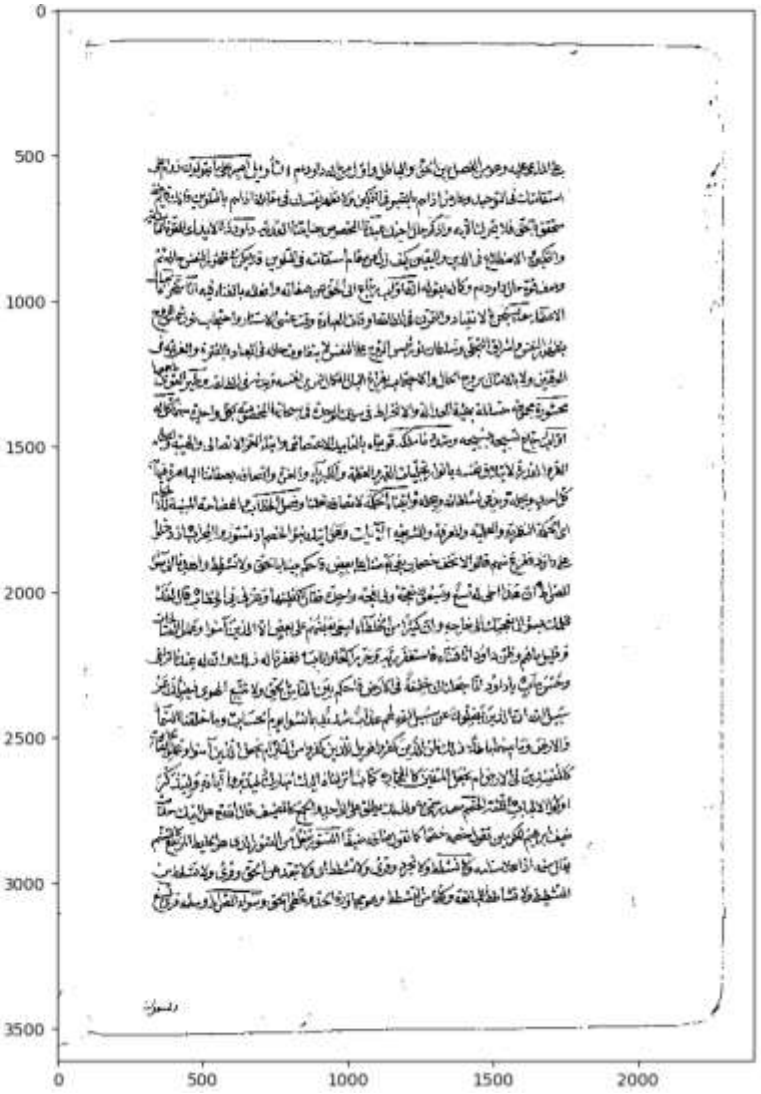
Novel Binarization method

```
im = plt.imread('sample.jpg')
for i, func in enumerate([lambda x:x , Proposed_method]) :
    plt.subplot(2,1,i+1); plt.imshow(func(im) ,cmap='gray')
```

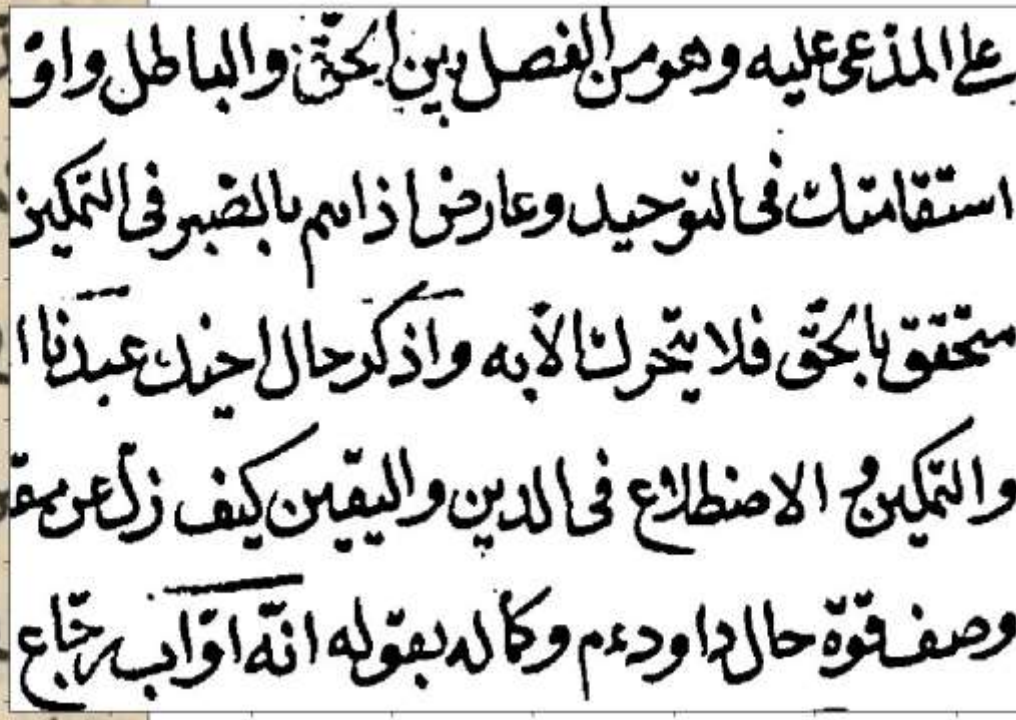


Novel Binarization method





Novel Binarization method



الـــ لله ـحمد
والصلاة والسلام على أشرف المرسلين

Thank You