# Bài 1: Viết hàm đọc vào một ảnh, cắt ảnh theo một kích thước và gốc tọa độ đưa vào theo tham số.

from PIL import Image

import matplotlib.pyplot as plt

import numpy as np

name = "Image/Fig0309(a)(washed\_out\_aerial\_image).tif"

img1 = Image.open(name)

old = np.asarray(img1)

plt.imshow(old, cmap="gray")

def slice\_image(image, x\_coor, y\_coor, x\_size, y\_size):

    return image[y\_coor:y\_coor+y\_size, x\_coor:x\_coor+x\_size]

y\_coor = 500

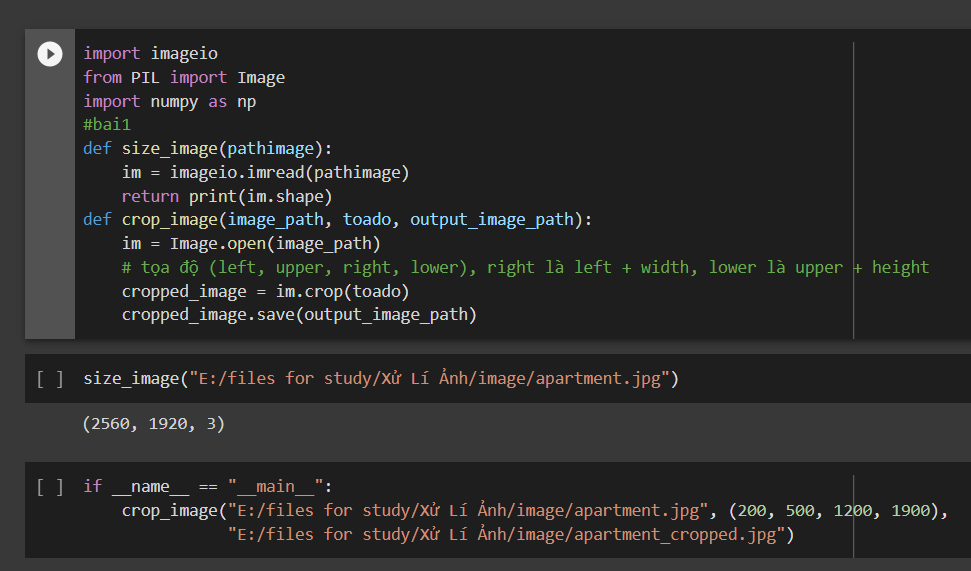
x\_coor = 400

y\_size = 150

x\_size = 130

cropped= slice\_image(old, x\_coor, y\_coor, x\_size, y\_size)

plt.imshow(cropped, cmap="gray")



Bài 2: Viết hàm xoay ảnh một góc alpha, sử dụng nội suy nearest neighbour  
Nearest neighbour: những pixal mới sẽ được gán giá trị của pixel cũ chứa điểm tương ứng của nó trong ảnh ban đầu.

def nn(img, angle):

    radians = float(angle\*(math.pi/180))

    width,height = img.shape

    a=int(width/2)

    b=int(height/2)

    rotate\_img = np.zeros((width,height),dtype="uint8")

    for i in range(0,width):

        for j in range(0,height):

            x=(i-a)\*math.cos(radians)+(j-b)\*math.sin(radians)+a

            y=-(i-a)\*math.sin(radians)+(j-b)\*math.cos(radians)+b

            # print(cx, yx)

            if x<width and y<height and x>0 and y>0:

                rotate\_img[i, j] = img[int(x),int(y)]

    return rotate\_img

from matplotlib import image as img

import numpy as np

import matplotlib.pyplot as plt

import math

m = img.imread("./Image/page.png")

w,h = m.shape[:2]

x1 = w\*math.cos(math.pi/6) - h\*math.sin(math.pi/6)

y1= w\*math.sin(math.pi/6) + h\*math.cos(math.pi/6)

x2 = -h\*math.sin(math.pi/6)

y2 = h\*math.cos(math.pi/6)

x3= w\*math.cos(math.pi/6)

y3= w\*math.sin(math.pi/6)

x\_new = int(math.fabs(x3)+ math.fabs(x2))+1

y\_new = int(math.fabs(y1))

newImage = np.zeros((x\_new,y\_new,3),dtype = "float32")

for k in range(3):

    for i in range(x\_new):

        for j in range(y\_new):

            i\_new = i + x2

            x = i\_new\*math.cos(math.pi/6) + j\*math.sin(math.pi/6)

            y = -i\_new\*math.sin(math.pi/6) + j\*math.cos(math.pi/6)

            x\_int = int(x)

            y\_int= int(y)

            if(x<0) or (y<0) or (y>=h) or (x>=w):

                intensity =0.0

            else:

                x\_int = min(x\_int, w-2)

                y\_int = min(y\_int, h-2)

                x\_diff = x - x\_int

                y\_diff = y - y\_int

                a = m[x\_int, y\_int,k]

                b = m[x\_int + 1, y\_int,k]

                c = m[x\_int + 1, y\_int+1,k]

                d = m[x\_int,y\_int+1,k]

                value1 = a\*(1-y\_diff) + d\*y\_diff

                value2 = b\*(1-y\_diff) + c\*y\_diff

                intensity = value2\*x\_diff + value1\*(1-x\_diff)

            newImage[i,j,k] = intensity

# img.imsave("bug\_rotate.png",newImage)

plt.imshow(newImage)

Bài 3: Viết hàm phóng to ảnh dùng nội suy bilinear

Bilinear Interpolation sẽ lấp đầy những khoảng trống khi phóng ảnh dựa vào những pixel xung quanh.

"""

 use bilinear interpolation to resize an image

"""

from PIL import Image

import cv2

import matplotlib.pyplot as plt

import numpy as np

# def resizeImage(name, scale):

#     img1 = Image.open(name)

#     old = np.asarray(img1)  # convert to Numpy array

#     rows, cols, layers = old.shape

#     plt.imshow(old)

#     new = np.zeros((rows\*scale-1, cols\*scale-1, layers))

#     for layer in range(4):

#         new[:, :, layer] = resizeLayer(old[:, :, layer], scale)

#     # convert the values to unsigned, 8-bit integers

#     new = new.astype(np.uint8)

#     img2 = Image.fromarray(new) # convert back to Image

#     newName = 'resize.png'

#     img2.save(newName)

def resizeImage(name, scale):

    img1 = cv2.imread(name)

    old = np.asarray(img1)  # convert to Numpy array

    enlargedShape = list(map(int, [old.shape[0]\*scale, old.shape[1]\*scale, old.shape[2]]))

    enlargedImg = np.empty(enlargedShape, dtype=np.uint8)

    rowScale = float(old.shape[0]) / float(enlargedImg.shape[0])

    colScale = float(old.shape[1]) / float(enlargedImg.shape[1])

    for r in range(enlargedImg.shape[0]):

        for c in range(enlargedImg.shape[1]):

            orir = r \* rowScale #Find position in original image

            oric = c \* colScale

            enlargedImg[r, c] = GetBilinearPixel(old, oric, orir)

    # convert the values to unsigned, 8-bit integers

    # new = new.astype(np.uint8)

    img2 = Image.fromarray(enlargedImg) # convert back to Image

    newName = 'resize.png'

    img2.save(newName)

    img2.show()

def GetBilinearPixel(imArr, posX, posY):

    out = []

    #Get integer and fractional parts of numbers

    modXi = int(posX)

    modYi = int(posY)

    modXf = posX - modXi

    modYf = posY - modYi

    modXiPlusOneLim = min(modXi+1,imArr.shape[1]-1)

    modYiPlusOneLim = min(modYi+1,imArr.shape[0]-1)

    #Get pixels in four corners

    for chan in range(imArr.shape[2]):

        bl = imArr[modYi, modXi, chan]

        br = imArr[modYi, modXiPlusOneLim, chan]

        tl = imArr[modYiPlusOneLim, modXi, chan]

        tr = imArr[modYiPlusOneLim, modXiPlusOneLim, chan]

        #Calculate interpolation

        b = modXf \* br + (1. - modXf) \* bl

        t = modXf \* tr + (1. - modXf) \* tl

        pxf = modYf \* t + (1. - modYf) \* b

        out.append(int(pxf+0.5))

    return out

# def resizeLayer(old, scale):

#     rows, cols = old.shape

#     # move old points

#     rNew = scale\*rows - 1

#     cNew = scale\*cols - 1

#     new = np.zeros((rNew, cNew))

#     new[0:rNew:scale, 0:cNew:scale] = old[0:rows, 0:cols]

#     # produce vertical values

#     new[1: rNew:scale, :] = (new[0:rNew-1:2, :] + new[2:rNew:2, :])/2

#     # produce horizontal values

#     new[:, 1:cNew:scale] = (new[:, 0:cNew-1:2] + new[:, 2:cNew:2])/2

#     # produce center values

#     new[1:rNew:scale, 1:cNew:scale] = ( new[0:rNew-2:2, 0:cNew-2:2] +

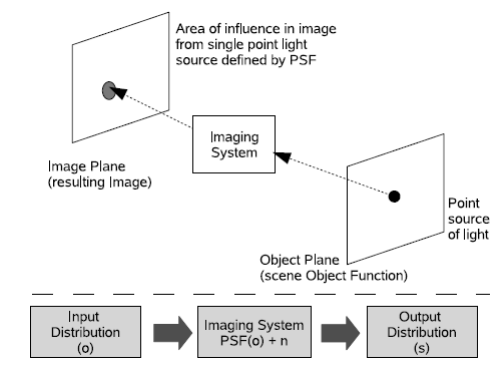
#                                         new[0:rNew-2:2, 2:cNew:2] +

#                                         new[2:rNew:2, 0:cNew-2:2] +

#                                         new[2:rNew:2, 2:cNew:2]) / 4

#     return new

Bài 4: : Trình bày mô hình tạo ảnh.



Từ ảnh rời rạc đầu vào, đi qua hàm chuyển điểm giúp miêu tả thông tin từ áng sáng chiếu tới độ thu nhận ảnh được ghi lại ở đầu ra. Ví dụ ánh sáng mặt trời chiếu trong phòng là 1000 lm/m^2 (i) và giá trị điển hình của r=0.01 đối với vải nhu thì l = 1000\*0.01 = 100. Ảnh đầu ra được xác định là: PSF(o) + nhiễu.

Quy trình tạo ảnh bao gồm 2 bước:

* Lấy mẫu (sampling): Thay vì đo từ mảng 2 chiều A[x][y] với

0 ≤x, y≤∞ ta s*ẽ* *đ*o trong m*ả*ng A[0,  M][0,  N] l*à* c*á*c kho*ả*ng kh*ô*ng gian nh*ỏ*

* Lượng tử hóa (quantization): Ánh xạ cường độ đo được trong các mảng trên thành một số hữu hạn các mức rời rạc

Mô hình toán học là:

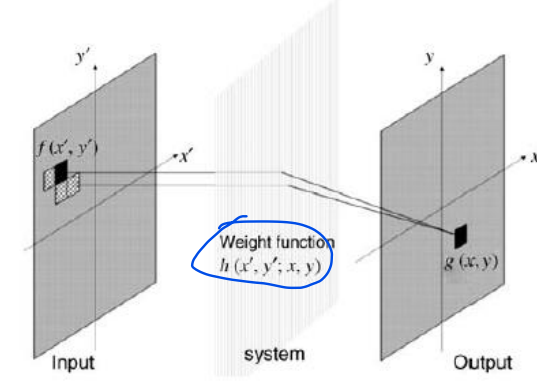
F(x’, y’) là đầu vào tại vị trí (x’, y’)

G(x, y) là đầu ra của hệ tạo ảnh.

Với



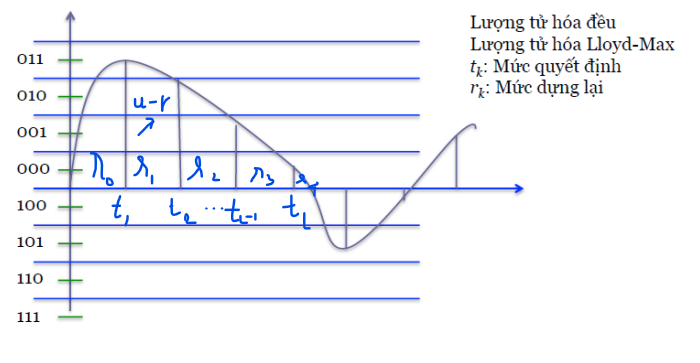
Trong đó h(x, y; x’, y’) là weight function ánh xạ từ f(x’, y’) tới g(x, y).



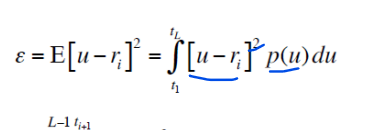
Bài 5 Trình bày phương pháp lượng tử hóa Lloyd Max. Phương pháp này dùng để làm gì?

Phương pháp lượng tử hóa Lloyd-max:

Với u là một biến thực ngẫu nhiên ( 1 điểm ảnh trong khoảng t\_k -> t\_(k+1) ) mình sẽ gán cho nó một giá trị r\_k.

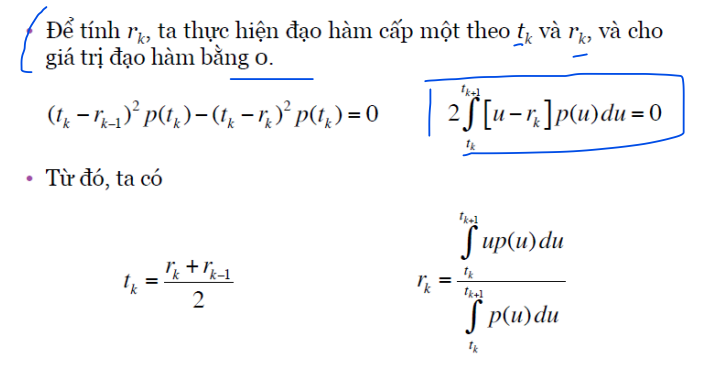


Tiến hành tính tổng bình phương lỗi



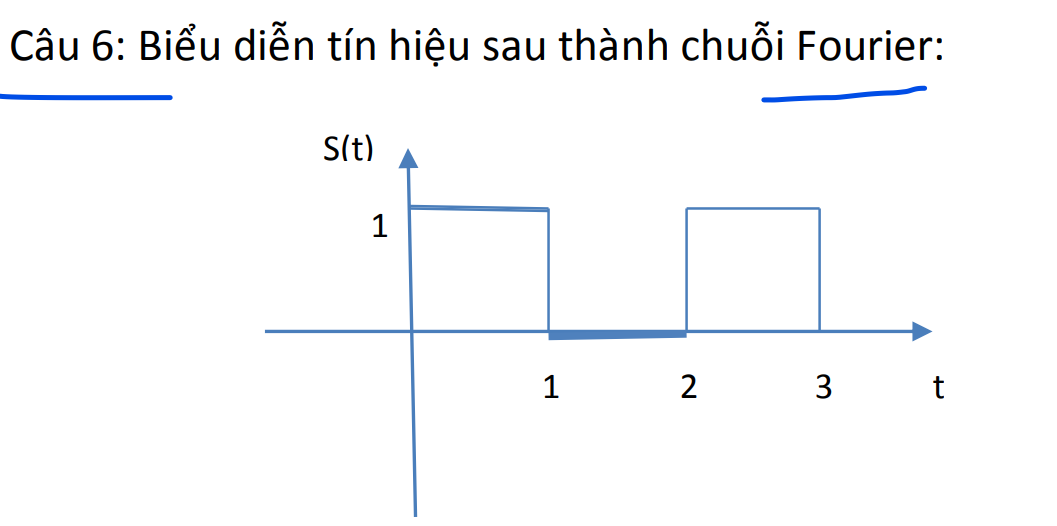
Với [ u – r\_i ] là độ lỗi, lấy bình phương tránh trường hợp âm.

P(u): mật độ ( số điểm ảnh )



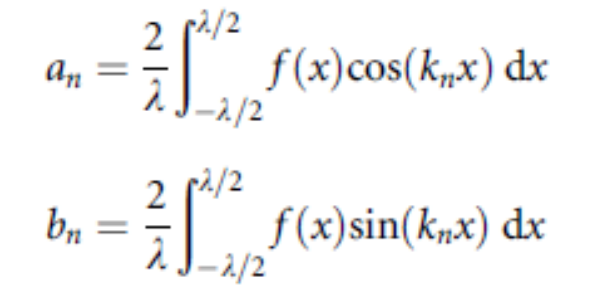
* Phương pháp Lloyd-max: dùng để chia ảnh ra thành các mức sáng. Như vậy việc lấy mẫu càng nhỏ giúp ảnh càng nét.

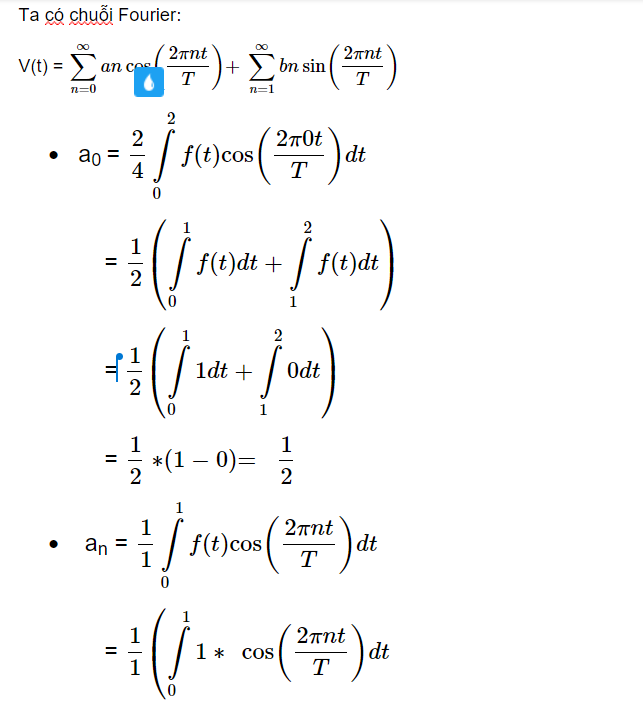
(Xem thêm trong vở ghi ).

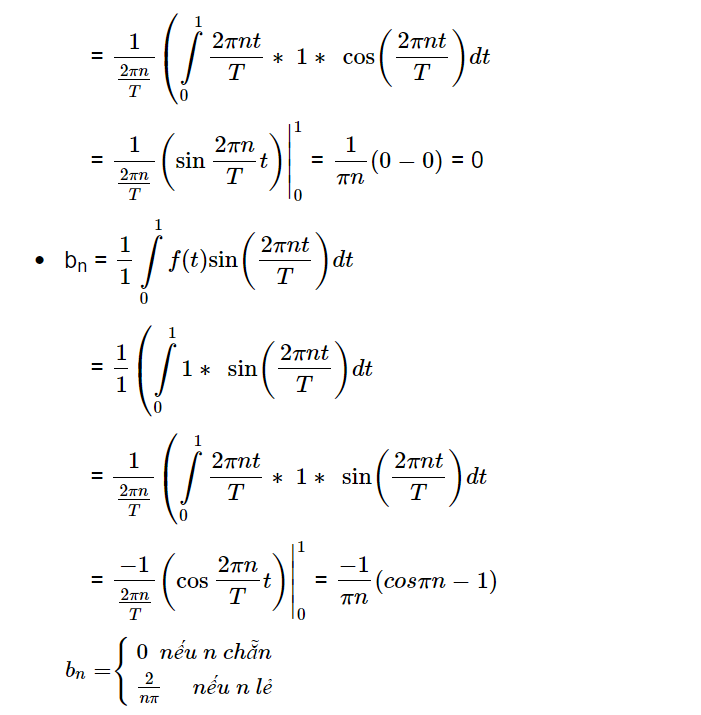


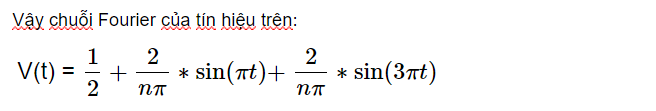
Với T = 2 ?

( chép chuỗi fourier)









Câu 8: Đọc vào một ảnh và hiển thị ảnh đó dưới dạng phổ fourier, dùng lệnh Python

from PIL import Image

import numpy as np

import matplotlib.pyplot as plt

image = Image.open("bugs.png")

#chuyen doi tuong image sang dang numpy array

im = np.array(image)

dark\_image\_grey = image.convert('L')

plt.figure(num=None, figsize=(8, 6), dpi=80)

plt.subplot(121)

plt.imshow(dark\_image\_grey, cmap='gray');

ft = np.fft.ifftshift(dark\_image\_grey)

ft = np.fft.fft2(ft)

dark\_image\_grey\_fourier = np.fft.fftshift(ft)

plt.subplot(122)

plt.imshow(np.log(abs(dark\_image\_grey\_fourier)), cmap='gray');

Câu 9: Dùng phương pháp phân vùng bằng ngưỡng (thresholding) theo thuật toán OTSU để tách chữ trong ảnh page.png và sau đó chuyển sang một ảnh khác.

Bài làm:

Cách hoạt động của Otsu: Tập trung vào việc khai thác các tính toán từ thông tin Histogram của bức ảnh. Bằng việc tính toán trên tất cả các mức Threshold ta có thể chọn mức thỏa mãn việc phân chia giữa Foreground và Background tốt nhất.

http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html

import numpy as np

import matplotlib.pyplot as plt

import skimage.io

import skimage.color

import skimage.filters

%matplotlib widget

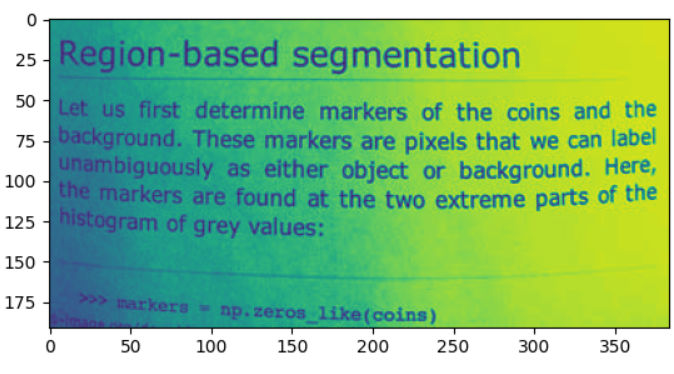
# load the image

image = skimage.io.imread("Image/page.png", as\_gray=True)

fig, ax = plt.subplots()

plt.imshow(image)

plt.show()



histogram, bin\_edges = np.histogram(image, bins=256, range=(0.0, 1.0))

fig, ax = plt.subplots()

plt.plot(bin\_edges[0:-1], histogram)

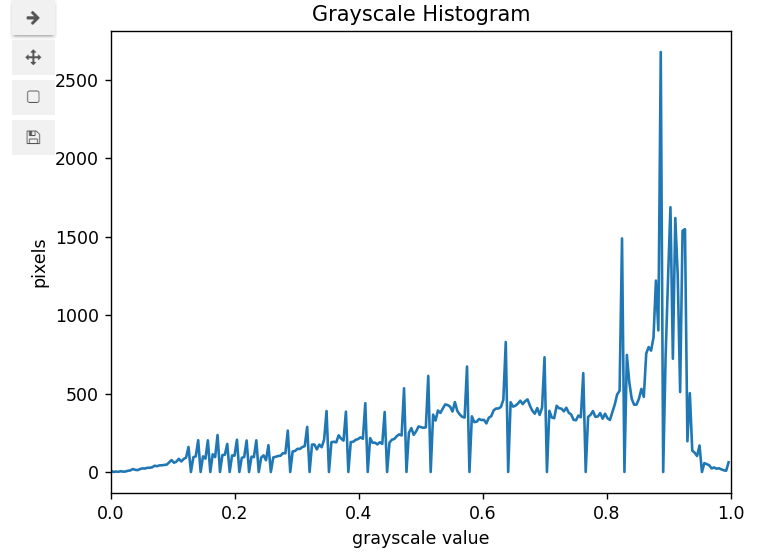
plt.title("Grayscale Histogram")

plt.xlabel("grayscale value")

plt.ylabel("pixels")

plt.xlim(0, 1.0)

plt.show()



# perform automatic thresholding

t = skimage.filters.threshold\_otsu(image)

print("Found automatic threshold t = {}.".format(t))

Found automatic threshold t = 0.615234375.

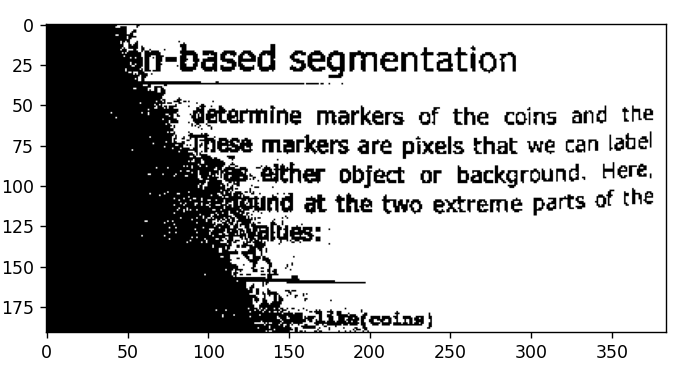
# create a binary mask with the threshold found by Otsu's method

binary\_mask = image > t

fig, ax = plt.subplots()

plt.imshow(binary\_mask, cmap='gray')

plt.show()



# apply the binary mask to select the foreground

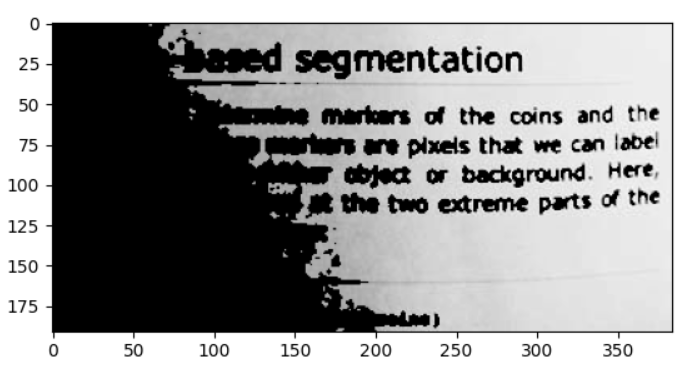
selection = image.copy()

selection[~binary\_mask] = 0

fig, ax = plt.subplots()

plt.imshow(selection, cmap='gray')

plt.show()



from matplotlib import image as img

import numpy as np

import matplotlib.pyplot as plt

import skimage.io

image = skimage.io.imread("Image/page.png", as\_gray=True)

# Set total number of bins in the histogram

bins\_num = 256

# Get the image histogram

hist, bin\_edges = np.histogram(image, bins=bins\_num)

# Get normalized histogram if it is required

# Calculate centers of bins

bin\_mids = (bin\_edges[:-1] + bin\_edges[1:]) / 2.

# Iterate over all thresholds (indices) and get the probabilities w1(t), w2(t)

weight1 = np.cumsum(hist)

weight2 = np.cumsum(hist[::-1])[::-1]

# Get the class means mu0(t)

mean1 = np.cumsum(hist \* bin\_mids) / weight1

# Get the class means mu1(t)

mean2 = (np.cumsum((hist \* bin\_mids)[::-1]) / weight2[::-1])[::-1]

inter\_class\_variance = weight1[:-1] \* weight2[1:] \* (mean1[:-1] - mean2[1:]) \*\* 2

# Maximize the inter\_class\_variance function val

index\_of\_max\_val = np.argmax(inter\_class\_variance)

threshold = bin\_mids[:-1][index\_of\_max\_val]

image[image<threshold] = 0

image[image>threshold] = 1

img.imsave("otsu\_thresholding.png",image)

Bài 10: Dùng phương pháp phân vùng bằng phương pháp ngưỡng thích ứng (tự lập trình, không dùng lệnh có sẵn trong thư viện) để tách chữ trong ảnh page.png và sau đó chuyển sang một ảnh khác.

Bài làm:

Adaptive Thresholding lựa chọn ngưỡng động trong vùng lân cận. Ngưỡng t tại thời điểm xét có thể dựa trên thống kê.

Ngưỡng t có thể là: t = mean + c, t = median + c hoặc t = floor(max – min)/2) + C của vùng lân cận quanh điểm ảnh.

import numpy as np

import matplotlib.pyplot as plt

import skimage

def threshold\_local(image, block\_size=3, method='gaussian', C=0):

    block\_size = (block\_size,) \* image.ndim

    block\_size = tuple(block\_size)

    image = image.astype('float64', copy=False)

    thresh\_image = np.zeros(image.shape, dtype='float64')

    if method == 'mean':

        # calculate the mean in each block

        for i in range(image.shape[0]):

            for j in range(image.shape[1]):

                block = image[i:i+block\_size[0], j:j+block\_size[1]]

                thresh\_image[i,j] = block.mean()

    elif method == 'median':

        # calculate the median in each block

        for i in range(image.shape[0]):

            for j in range(image.shape[1]):

                block = image[i:i+block\_size[0], j:j+block\_size[1]]

                thresh\_image[i,j] = np.median(block)

    elif method == 'min\_max':

        # calculate the min and max in each block

        for i in range(image.shape[0]):

            for j in range(image.shape[1]):

                block = image[i:i+block\_size[0], j:j+block\_size[1]]

                thresh\_image[i,j] = (block.max() - block.min()) / 2

    return thresh\_image + C

%matplotlib widget

# load the image

image = skimage.io.imread("../Image/page.png", as\_gray=True)

binary\_image1 = image > threshold\_local(image, 15, 'min\_max')

plt.imshow(binary\_image1, cmap='gray')

Bài 11: Ý nghĩa của phép biến đổi âm bản. Thực hiện trên ảnh page.png



<https://vtct.wordpress.com/2013/05/14/tao-anh-am-ban-negative-image/>

Ý nghĩa: Cách xử lí này nhằm tăng cường các chi tiết trắng hoặc xám nằm trong các vùng tối của ảnh, nhất là khi các vùng đen chiếm diện tích nhiều hơn.

Trong đó L là số mức xám hoặc số mức cường độ. Phương pháp này dùng để tăng cường các chi tiết trắng hoặc xám nằm trong vùng tối, đặc biệt là ở các bức ảnh mà vùng tối chiếm nhiều hơn.

import numpy as np

import matplotlib.pyplot as plt

import skimage

#negative transformation function

def neg\_trans(img):

  #get width and height of image

  img = np.array(img)

  width,height, layer = img.shape

  # get number of brightness levels of image

  L = 256

  #traverse through pixels

  print(layer)

  for x in range(width):

    for y in range(height):

      #if image is RGB, subtract individual RGB values

      if layer is not None:

        I=img[x,y]

        #s=(L-1)-r

        red\_pixel= (L - 1) -I[0]

        green\_pixel= (L - 1) -I[1]

        blue\_pixel= (L - 1) -I[2]

        #replace the pixel of the image

        img[x,y] = (red\_pixel,green\_pixel,blue\_pixel,I[3])

        # img.putpixel((x,y),(red\_pixel,green\_pixel,blue\_pixel))

      else:

        I=img[x,y]

        #s=(L-1)-r

        S = (L - 1) -I

        #replace the pixel

        img[x,y]  = S

  return img

from PIL import Image

image = Image.open("../Image/page.png")

# image = skimage.io.imread("../Image/page.png", as\_gray=True)

plt.imshow(image, cmap='gray')

Bài 12: Khi nào cần dùng phép biến đổi log trên ảnh. Áp dụng đối với ảnh Fig0323(a)(mars\_moon\_phobos).tif và cho nhận xét

Khi ta muốn mở rộng phạm vi giá trị của các điểm tối và thu hẹp vùng giá trị của các điểm sáng.

+ Trong trường hợp ảnh có pixel thấp ta sử dụng biến đổi log ảnh

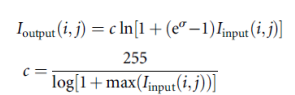
khiến ảnh được sử lý rõ nét hơn

+ Chúng ta áp dụng phương pháp này trong một hình ảnh có giá trị pixel

cao hơn thì nó sẽ nâng cao hình ảnh hơn và thông tin thực tế của hình

ảnh sẽ bị mất

+ Nói chung chuyển đổi log hoạt động tốt trong những hình ảnh tối



import numpy as np

from PIL import Image

import math

import matplotlib.pyplot as plt

#im = img.imread("Pictures/coins.png")

im = Image.open("../Image/Fig0323(a)(mars\_moon\_phobos).tif")

im\_arr = np.array(im)

c= 255/math.log(1+np.max(im\_arr))

d= 5000

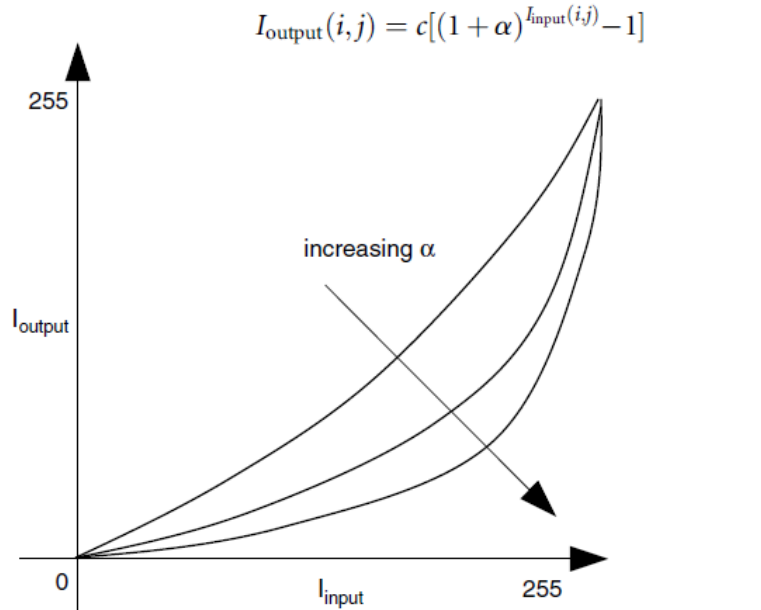
im\_out = (c\*np.log(1+ (np.exp(d)-1)\*im\_arr))

im\_transformed = Image.fromarray(im\_out)

# im\_transformed.save("./cameraman\_log\_transform\_.tif")

plt.imshow(im\_transformed, cmap='gray')

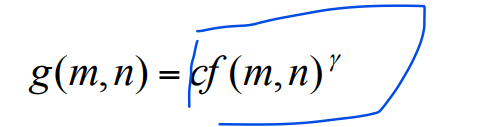
Bài 13: : Khi nào cần dùng phép biến đổi mũ trên ảnh. Áp dụng đối với ảnh Fig0309(a)(washed\_out\_aerial\_image).tif và cho nhận xét



Phạm vi của ánh sáng thấp sẽ được thu hép. Ánh sáng cao được mở rộng

Khi muốn các giá trị điểm ảnh đầu vào sẽ tăng/ giảm theo một mức lũy thừa cố định.

Công thức:



g: ảnh đầu ra

f: ảnh đầu vào

y:l*ũ*y th*ừ*a gammay:lũy thừa gamma

c: hằng số

import numpy as np

import matplotlib.image as img\_pyp

import math

import matplotlib.pyplot as plt

#im = img.imread("Pictures/coins.png")

# im = Image.open()

def gamma\_transform(c, f, gamma):

    """

        c: constant

        r: input image

    """

    return c \* np.array(f \*\* gamma, dtype = 'uint8')

# Đọc ảnh gốc

img = img\_pyp.imread("../Image/Fig0323(a)(mars\_moon\_phobos).tif")

# làm tròn ảnh gốc float -> int để visualize

img = np.array(img, dtype = np.uint8)

# show ảnh gốc

plt.imshow(img)

plt.title("Anh goc")

plt.show()

# Thử 3 giá trị gamma

# for gamma in [0.3, 3, 5]:

# ảnh được biến đổi với hàm mũ

gamma = 5

gamma\_corrected = gamma\_transform(1, img, gamma)

    # visualize ảnh

title = "Anh co gamma=" + str(gamma)

plt.imshow(gamma\_corrected)

plt.title(title)

plt.show()

Bài 14: Vì sao cần kéo giãn vùng giá trị ảnh. Hiển thị histogram Fig0354(a)(einstein\_orig).tif và thực hiện kéo dãn vùng giá trị ảnh. Cho nhận xét.

Giúp cải thiện độ tương phải của ảnh bằng cách kéo phạm vi giá trị cường độ mà nó chưa một phạm vi giá trị mong muốn.

Gọi a và b là lower và upper linits

C, d là giá trị pixel thấp nhất và cao nhất của ảnh

<https://homepages.inf.ed.ac.uk/rbf/HIPR2/stretch.htm>

giãn ảnh :

Sự giãn nở hình thái làm cho các vật thể dễ nhìn thấy hơn và lấp đầy các lỗ nhỏ trên vật thể. Các đường kẻ có vẻ dày hơn và các hình được tô có vẻ lớn hơn.

def contract\_stretching(image):

    a = 0

    b = 255

    c = np.max(image)

    d = np.min(image)

    # im\_out = np.zeros(image.shape)

    im\_out = (image - c) \* ((a - b) / (c - d)) + a

    return im\_out

# create image ouput with contrast stretching

# image\_out = skimage.exposure.rescale\_intensity(image, out\_range=(0, 1))

image\_out = contract\_stretching(image)

fig, ax = plt.subplots()

plt.imshow(image\_out, cmap='gray')

plt.show()

# create the histogram

histogram\_1, bin\_edges\_1 = np.histogram(image\_out, bins=256, range=(0, 255))

# configure and draw the histogram figure

plt.figure()

plt.title("Grayscale Histogram")

plt.xlabel("grayscale value")

plt.ylabel("pixel count")

plt.plot(bin\_edges\_1[0:-1], histogram\_1)  # <- or here

plt.show()

Câu 15: Thực hiện cân bằng histogram và histogram đặc trưng đối với ảnh Fig0320(2)(2nd\_from\_top).tif

Histogram chỉ định trong phương pháp histogram đặc trưng có thể lấy từ ảnh Fig0316(3)(third\_from\_top).tif

* Y1

import numpy as np

import matplotlib.pyplot as plt

from PIL import Image

from skimage import io

import matplotlib.image as im

img = im.imread('../Image/Fig0320(2)(2nd\_from\_top).tif')

# pr so diem anh

pr, bins = np.histogram(img, bins = 256, range=(0,256))

# mat do

cr = pr / (img.shape[0] \* img.shape[1])

# bins[:-1] bo khoang cuoi

plt.plot(bins[:-1], pr)

# TInh phan phoi tich luy

sum = 0

for i in range(cr.shape[0]):

    sum += cr[i]

    cr[i] = sum

plt.plot(bins[:-1], cr)

s = np.empty\_like(256)

s = (256 - 1) \* cr

s = s.astype('uint8')

equalized\_img = np.empty\_like(img)

for i in range(img.shape[0]):

    for j in range(img.shape[1]):

        equalized\_img[i][j] = s[img[i][j]]

plt.imshow(equalized\_img, cmap='gray')

# new histogram

pr\_new, bins\_new = np.histogram(equalized\_img, bins = 256, range=(0,256))

plt.plot(bins\_new[:-1], pr\_new)

* Ý 2

import numpy as np

import matplotlib.pyplot as plt

from PIL import Image

from skimage import io

import matplotlib.image as im

img = im.imread('../Image/Fig0316(3)(third\_from\_top).tif')

# pz so diem anh

pz, bins = np.histogram(img, bins = 256, range=(0,256))

plt.plot(bins[:-1], pz)

pz\_specified = pz/(img.shape[0] \* img.shape[1])

# nr so diem anh

nr, bis = np.histogram(img, bins = 256, range=(0,256))

# mat do

pr = nr / (img.shape[0] \* img.shape[1])

pr = pr.astype('uint8')

# bins[:-1] bo khoang cuoi

plt.plot(bis[:-1], nr)

# TInh phan phoi tich luy

# s dc xac dinh trong phan can bang anh

sum = 0

arr\_nr = np.zeros\_like(nr)

for i in range(nr.shape[0]):

    sum += nr[i]

    arr\_nr[i] = sum

s = np.empty\_like(256)

s = (256 - 1) \* arr\_nr

s = s.astype('uint8')

gz = np.zeros\_like(pz\_specified)

sum\_z = 0

# xac xuat tich luy

for i in range(pz\_specified.shape[0]):

    sum\_z += pz\_specified[i]

    gz[i] = sum\_z

gz = (256 - 1) \* gz

gz = gz.astype('uint8') # Lam tron

zq = []

for i in range(s.shape[0]):

    # find the corresponding value of z\_q. Gz is closet to s

    zq.append(np.where(gz==s[i])[0])

pz\_actual = np.zeros\_like(pz\_specified)

for i in range(s.shape[0]):

    if len(zq[i]) == 0:

        pz\_actual[i] = 0

    else:

        sum = 0

        for j in zq[i]:

            # print(nr[j])

            # print(j)

            sum += nr[j]

        # print(i, sum)

        pz\_actual[i] = sum / (img.shape[0] \* img.shape[1])

Câu 16: Thực hiện histogram cục bộ đối với ảnh Fig0326(a)(embedded\_square\_noisy\_512).tif

import matplotlib.pyplot as plt

from skimage import exposure

import skimage.morphology as morp

from skimage.filters import rank

import matplotlib.image as img

# Original image

img = img.imread("../Image/Fig0326(a)(embedded\_square\_noisy\_512).tif")

# Global equalize

img\_global = exposure.equalize\_hist(img)

# Local Equalization, disk shape kernel

# Better contrast with disk kernel but could be different

kernel = morp.disk(30)

img\_local = rank.equalize(img, selem=kernel)

fig, (ax\_img, ax\_global, ax\_local) = plt.subplots(1, 3)

ax\_img.imshow(img, cmap=plt.cm.gray)

ax\_img.set\_title('Low contrast image')

ax\_img.set\_axis\_off()

ax\_global.imshow(img\_global, cmap=plt.cm.gray)

ax\_global.set\_title('Global equalization')

ax\_global.set\_axis\_off()

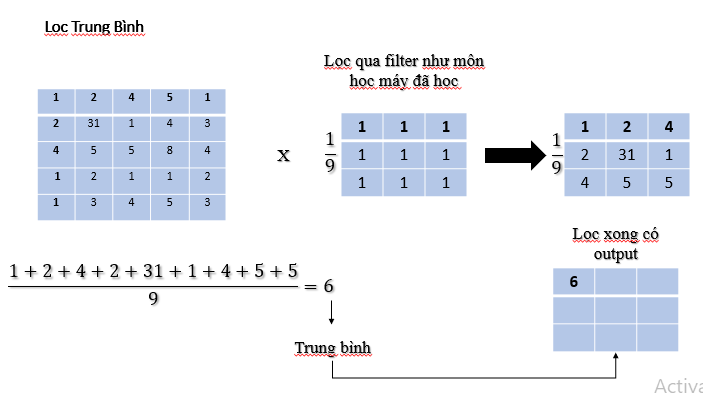
ax\_local.imshow(img\_local, cmap=plt.cm.gray)

ax\_local.set\_title('Local equalization')

ax\_local.set\_axis\_off()

plt.show()

Câu 17: Thực hiện lọc trung bình trên ảnh Fig0441(a)(characters\_test\_pattern).tif . Vì sao bộ lọc trung bình giảm được nhiễu, đặc biệt là nhiễu Gausian C



Khi đó: ảnh nhiễu gauss sẽ bị nhiễu phủ dạng sương do hàm mật độ phân phối gauss mà lọc trung bình sẽ đi làm mờ các hàm mật độ gauss đó bằng cách tính trung bình điểm ảnh.

Code:

from skimage import io

from scipy.ndimage.filters import convolve

# from scipy.ndimage import correlate

# from scipy.ndimage.filters import median\_filter

import numpy as np

im  =io.imread("../Image/Fig0441(a)(characters\_test\_pattern).tif")

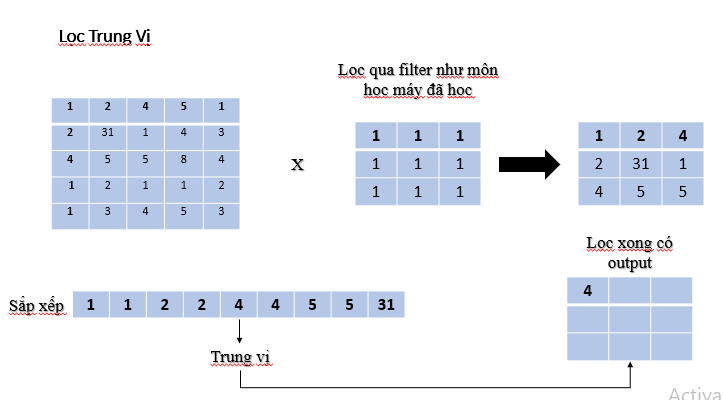
k = (1/9)\*np.ones((3,3),dtype = 'float32')

# mean filter of im

im\_mean = convolve(im,k)

io.imshow(im\_mean)

âu 18: Thực hiện lọc trung vị trên ảnh Fig0441(a)(characters\_test\_pattern).tif. Vì sao bộ lọc trung vị giảm được nhiễu muối tiêu



Tại sao lọc trung vị hiệu quả với nhiễu salt&pepper?

Vì Phép lọc trung vị sẽ lọc loại bỏ các giá trị cực trị khỏi ảnh input. Phép lọc trung vị giữ lại các giá trị thật của ảnh input.

Phép lọc trung vị chậm hơn so với lọc trung bình vì cần sắp xếp để tìm giá trị trung vị.

Bộ lọc trung vị xử lý nhiễu muối tiêu rất hiệu quả vì:

Nguyên tắc của bộ lọc trung vị là thay thế mức xám của mỗi

pixel bằng trung vị của các mức xám trong vùng lân cận của pixel,

thay vì sử dụng phép toán trung bình.

from skimage import io

# from scipy.ndimage.filters import convolve

# from scipy.ndimage import correlate

from scipy.ndimage.filters import median\_filter

import numpy as np

# im  =io.imread("../Image/Fig0441(a)(characters\_test\_pattern).tif")

im = Image.open('../Image/Fig0441(a)(characters\_test\_pattern).tif').convert('L')

# k = (1/9)\*np.ones((3,3),dtype = 'float32')

# mean filter of im

im\_mean = median\_filter(im,size=5, footprint=None,output=None,mode='reflect', cval=0.0,origin=0)

io.imshow(im\_mean)