

**TRƯỜNG ĐẠI HỌC XÂY DỰNG HÀ NỘI**

Môn học: Xử lý ảnh

Mã môn học: 608809

Bài tập tổng hợp

|  |  |
| --- | --- |
| Sinh viên:  **Dương Công Sơn**  **Mã lớp: 64CS3**  **Mã sinh viên: 167464** | Giảng viên:  **Thái Thị Nguyệt** |

Bộ môn Khoa học máy tính

**Abstract**

Mục lục

[1 Đọc và cắt ảnh 4](#_Toc101526814)

[2 Xoay ảnh sử dụng nội suy nearest neighbour 5](#_Toc101526815)

[3 Phóng to ảnh sử dụng nội suy bilinear 6](#_Toc101526816)

[4 Mô hình tạo ảnh 8](#_Toc101526817)

[5 Lượng tử hóa Lloyd Max 9](#_Toc101526818)

[6 Biến đổi Fourier 10](#_Toc101526819)

[7 Biến đổi Fourier 12](#_Toc101526820)

[8 Hiển thị ảnh dạng phổ Fourier 13](#_Toc101526821)

[9 Phân vùng ảnh thuật toán OTSU 14](#_Toc101526822)

[10 Phân vùng ảnh phương pháp ngưỡng thích ứng 16](#_Toc101526823)

[11 Biến đổi âm bản 18](#_Toc101526824)

[12 Biến đổi log 19](#_Toc101526825)

[13 Biến đổi mũ 21](#_Toc101526826)

[14 Kéo giãn vùng giá trị ảnh 22](#_Toc101526827)

[15 Cân bằng histogram và histogram đặc trưng. 24](#_Toc101526828)

[16 Histogram cục bộ. 28](#_Toc101526829)

[17 Lọc trung bình. 29](#_Toc101526830)

[18 Lọc trung vị. 31](#_Toc101526831)

# Đọc và cắt ảnh

**Yêu cầu:** Viết hàm đọc vào một ảnh, cắt ảnh theo một kích thước và gốc tọa độ đưa vào theo tham số.

**Bài làm**

|  |  |
| --- | --- |
| Hàm đọc vào một ảnh   1. def read\_image(filename): 2. img = Image.open(filename) 3. img = img.convert('L') 4. img = np.array(img) 5. return img 6. file\_name = "Image/Fig0309(a)(washed\_out\_aerial\_image).tif" 7. old = read\_image(file\_name) 8. plt.imshow(old, cmap="gray") |  |

Hàm cắt ảnh theo một kích thước

|  |  |
| --- | --- |
| 1. def crop\_size\_image(image, width\_new, height\_new): 2. return image[:height\_new, :width\_new] 3. cropped= crop\_size\_image(old, 100, 200) 4. plt.imshow(cropped, cmap="gray") |  |

Hàm cắt ảnh theo gốc tọa độ

|  |  |
| --- | --- |
| 1. def slice\_image(image, x\_coor, y\_coor, x\_size, y\_size): 2. return image[y\_coor:y\_coor+y\_size,   x\_coor:x\_coor+x\_size]   1. y\_coor = 500 2. x\_coor = 400 3. y\_size = 150 4. x\_size = 130 5. cropped= slice\_image(old, x\_coor, y\_coor, x\_size, y\_size) 6. plt.imshow(cropped, cmap="gray") |  |

# Xoay ảnh sử dụng nội suy nearest neighbour

**Yêu cầu:** Viết hàm xoay ảnh một góc alpha, sử dụng nội suy nearest neighbour.

**Bài làm:**

Nội suy láng giềng gần nhất ( Nearest neighbour ) :Điểm ảnh mới sẽ lấy giá trị của điểm ảnh gốc gần nó nhất và không xem xét các giá trị khác ở tất cả các điểm lân cận.

|  |  |  |
| --- | --- | --- |
|  |  |  |

*Affine transformations*

|  |  |
| --- | --- |
| Hàm xoay ảnh góc alpha   1. def nn(img, angle): 2. radians = float(angle\*(math.pi/180)) 3. w,h = img.shape[:2] 4. x1= w\*math.cos(radians) 5. x2 = -h\*math.sin(radians) 6. y1= w\*math.sin(radians) + h\*math.cos(radians) 7. x\_new = int(math.fabs(x1)+ math.fabs(x2))+1 8. y\_new = int(math.fabs(y1)) 9. rotate\_img = np.zeros((x\_new,y\_new,3),dtype = "float32") 10. # ảnh RGB nên có 3 chiều 11. for k in range(3): 12. for i in range(x\_new): 13. for j in range(y\_new): 14. i\_new = i + x2 15. x = i\_new\*math.cos(radians) + j\*math.sin(radians) 16. y = -i\_new\*math.sin(radians) + j\*math.cos(radians) 17. x\_int = int(x) 18. y\_int= int(y) 19. if(x<0) or (y<0) or (y>=h) or (x>=w): 20. intensity =0.0 21. else: 22. x\_int = min(x\_int, w-2) 23. y\_int = min(y\_int, h-2) 24. x\_diff = x - x\_int 25. y\_diff = y - y\_int 26. # giá trị mới bằng những giá trị cũ tạo ra. 27. a = img[x\_int, y\_int,k] 28. b = img[x\_int + 1, y\_int,k] 29. c = img[x\_int + 1, y\_int+1,k] 30. d = img[x\_int,y\_int+1,k] 31. value1 = a\*(1-y\_diff) + d\*y\_diff 32. value2 = b\*(1-y\_diff) + c\*y\_diff 33. intensity = value2\*x\_diff + value1\*(1-x\_diff) 34. rotate\_img[i,j,k] = intensity 35. return rotate\_img | Ảnh gốc    Ảnh xoay    Nhận xét:  Phương pháp này có thời gian xử lý nhanh. Phù hợp với bài toán xoay ảnh vì không phải tạo ra nhiều điểm ảnh mới mà chỉ dịch chuyển vị trí của các điểm ảnh. Nhưng với bài toán phóng to ảnh, phương pháp này thường làm cho ảnh có hiệu ứng răng cưa. |

# Phóng to ảnh sử dụng nội suy bilinear

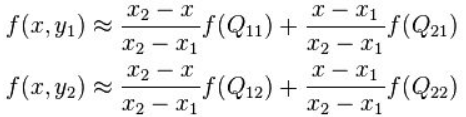
**Yêu cầu:** Viết hàm phóng to ảnh sử dụng nội suy bilinear

**Bài làm:**

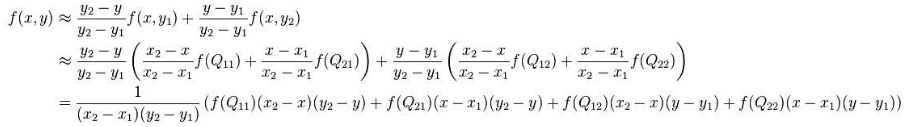
Nội suy song tuyến (Bilinear Interpolation) sẽ lấp đầy những khoảng trống khi phóng ảnh dựa vào những pixel xung quanh. Bằng việc tính trung bình của 4 điểm ảnh gần nhất để tính giá trị của điểm ảnh mới.

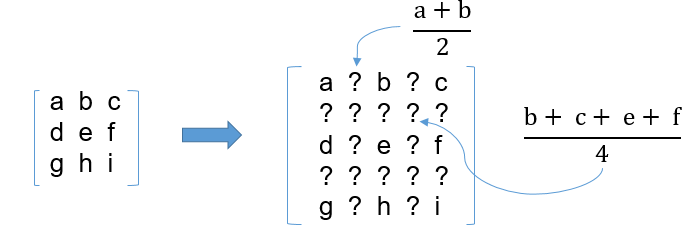
Ví dụ: Gọi f(x, y) là pixel chưa biết khi phóng to ảnh, xung quanh nó là 4 pixels

Công thức sử dụng nội suy Bilinear cho chiều x là:



Công thức sử dụng nội suy Bilinear cho chiều y là:



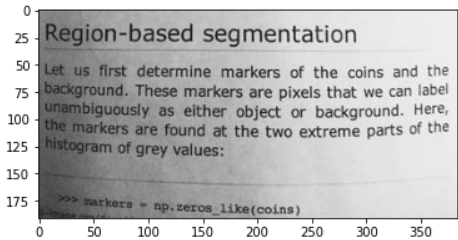


Mô tả quy trình phóng to ảnh.

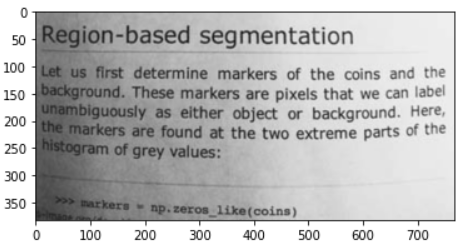
Phóng ảnh gấp đôi

1. def resizeLayer(old, scale=2):
2. rows, cols = old.shape
3. # move old points
4. rNew = 2\*rows - 1
5. cNew = 2\*cols - 1
6. new = np.zeros((rNew, cNew))
7. new[0:rNew:scale, 0:cNew:scale] = old[0:rows, 0:cols]
8. # produce vertical values
9. new[1: rNew:scale, :] = (new[0:rNew-1:2, :] + new[2:rNew:2, :])/2
10. # produce horizontal values
11. new[:, 1:cNew:scale] = (new[:, 0:cNew-1:2] + new[:, 2:cNew:2])/2
12. # produce center values
13. new[1:rNew:scale, 1:cNew:scale] = ( new[0:rNew-2:2, 0:cNew-2:2] +
14. new[0:rNew-2:2, 2:cNew:2] +
15. new[2:rNew:2, 0:cNew-2:2] +
16. new[2:rNew:2, 2:cNew:2]) / 4
17. return new

Ảnh gốc:



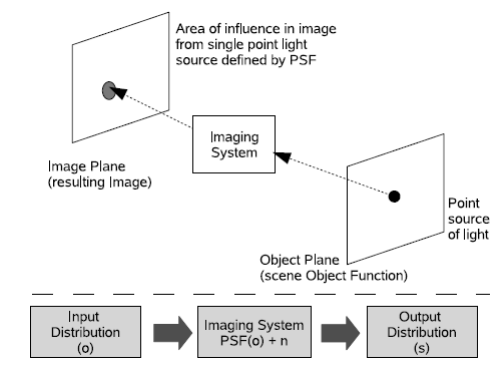
Ảnh sau khi phóng to



**Nhận xét:** Nội suy song tuyến tính có thời gian thực hiện và độ phức tạp cao hơn so với nội suy láng giềng gần nhất. Phương pháp nội suy này làm giảm sự biến dạng của ảnh gốc khi phóng to, làm mờ đường viền hình ảnh, đem lại hiệu quả thị giác tốt hơn.

# Mô hình tạo ảnh

**Yêu cầu:** Trình bày mô hình tạo ảnh



Từ ảnh rời rạc đầu vào, đi qua hàm truyền 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) tốc độ lấy mẫu tỉ lệ với việc chia nhỏ đồ thị mẫu và Lượng tử hóa (quantization): Ánh xạ cường độ đo được trong các mảng trên thành một số hữu hạn các mức rời rạc ( việc lấy mẫu càng nhỏ => ảnh càng nét).

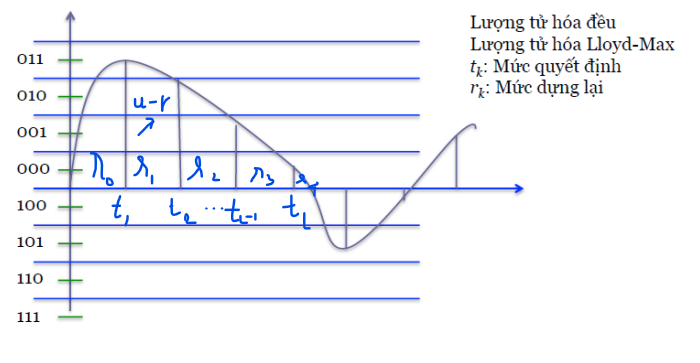
# Lượng tử hóa Lloyd Max

**Yêu cầu:** Trình bày phương pháp lượng tử hóa Lloyd Max. Phương pháp này dùng để làm gì?

**Bài làm:**

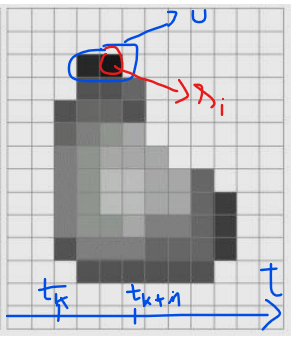
**Phương pháp lượng tử hóa Lloy-max**

**Bước 1:** 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.

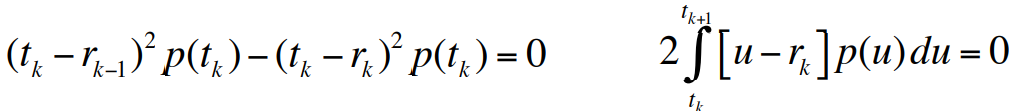


**Bước 2:** Tìm giá trị của r\_i theo công thức với cận của r\_i là [t\_k, t\_k+1)

Giá trị tk và rk của bộ lượng tử hóa L mức được xác định sao cho sai số trung bình bình phương là nhỏ nhất.



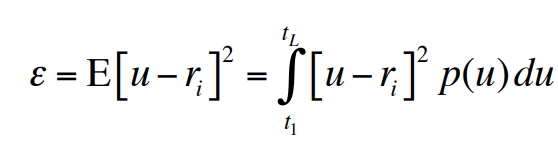
Để tính rk, ta thực hiện đạo hàm cấp một theo tk và rk, và cho giá trị đạo hàm bằng 0.



Từ đó, ta có:



**Bước 3:** Tiến hành tính tổng bình phương lỗi, lặp lại các bước trên trên để lỗi giảm.

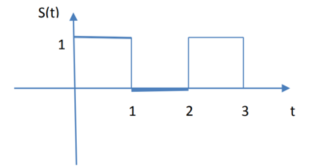


Với [u - Phương pháp Lloyd Max với mục tiêu để tối ưu sai số dựa vào tổng lỗi.

# Biến đổi Fourier

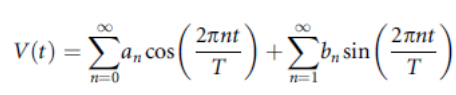
**Khái niệm:** Biến đổi Fourier là một loại biến đổi tích phân cho phép chúng ta xem xét hình ảnh và xử lý hình ảnh từ một góc nhìn khác thay thế bằng cách chuyển đổi bài toán sang một không gian khác. Đưa từ tín hiệu liên tục từ miền thời gian sang miền tần số.

**Yêu cầu:** Biểu diễn tín hiệu sau thành chuỗi Fourier.

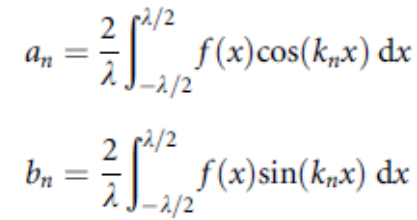


**Bài làm:**

Ta có chuỗi fourier:



Và các hệ số của phổ fourier:



Trong đó =

Với

=

=

=

= | =

= (

= () = 0

= (

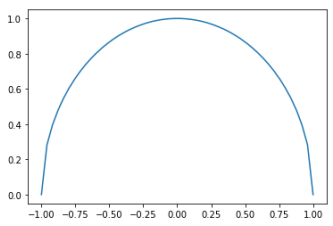
= ()

Với =

Vậy chuỗi Fourier của tín hiệu trên là:

# Biến đổi Fourier

**Yêu cầu:** Biểu diễn tín hiệu sau thành biến đổi Fourier cho tín hiệu y = f(x) với + =1, với y >=0 ( tức nửa đường tròn phía trên trục ox)

****

Ta có mà y = f(x)

Do vậy f(x) = ( |x| 1 )

Biến đổi Fourier của f(x) là:

Diagram

Description automatically generated

Với n = 1 và z =

Đặt x = sin(u)

Chuỗi fourier của đồ thị hàm số trên là:

# Hiển thị ảnh dạng phổ Fourier

**Yêu cầu:** Đọc vào một ảnh và hiển thị ảnh đó dưới dạng phổ fourier, dùng lệnh Python.

|  |  |
| --- | --- |
| Hàm đọc vào một ảnh   1. def read\_image(filename): 2. img = Image.open(filename) 3. img = img.convert('L') 4. img = np.array(img) 5. return img 6. file\_name = "Image/Fig0309(a)(washed\_out\_aerial\_image).tif" 7. old = read\_image(file\_name) 8. plt.imshow(old, cmap="gray") |  |

Hàm chuyển ảnh dưới dạng phổ fourier

|  |  |
| --- | --- |
| 1. def fourier\_transfer(image): 2. ft = np.fft.ifftshift(image) 3. ft = np.fft.fft2(ft) 4. fourier\_image = np.fft.fftshift(ft) 5. return fourier\_image |  |

# Phân vùng ảnh thuật toán OTSU

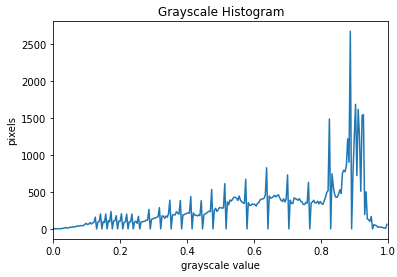
**Yêu cầu:** Dùng phương pháp phân vùng bằng ngưỡng (thresholding) theo thuật toán OTSU để tách chữ trong ảnh page.png và sau đó chuyển sang một ảnh khác.

**Khái niệm:** Thuật toán OTSU là một phương pháp tính ngưỡng tự động dựa vào giá trị điểm ảnh của ảnh đầu vào.

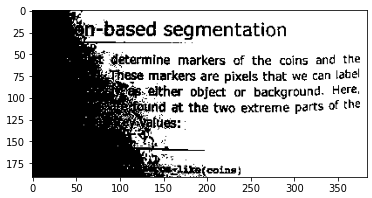
**Sử dụng thư viện skimage**

1. import matplotlib.pyplot as plt
2. import skimage.io
3. # load the image
4. image = skimage.io.imread("Image/page.png", as\_gray=True)
5. # perform automatic thresholding
6. t = skimage.filters.threshold\_otsu(image)
7. print(‘t = ‘ , t)
8. # create a binary mask with the threshold found by Otsu's method
9. binary\_mask = image > t
10. # apply the binary mask to select the foreground
11. selection = image.copy()
12. selection[~binary\_mask] = 0
13. fig, ax = plt.subplots()
14. plt.imshow(selection, cmap='gray')
15. plt.show()

Kết quả:



T = 0.615234327



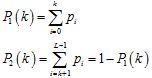
**Cài đặt lại thuật toán OTSU và lưu thành ảnh mới**

**Bước 1:** Sử dụng histogram biểu diễn tần suất xuất hiện:



**Bước 2:** Chọn ngưỡng T = k ( k là điểm chính giữa của histogram ). Gọi C1 là tập điểm ảnh nhỏ hơn k, C2 tập điểm ảnh lớn hơn k.

* P1 tỉ lệ C1 với số lượng điểm ảnh k với tổng số điểm ảnh.
* P2 tỉ lệ C2 với số lượng điểm ảnh k với tổng số điểm ảnh.



**Bước 3:** Tính trung bình mean1 của lớp c1 và mean2 của lớp c2





**Bước 4:** Tính ngưỡng k\* sao cho giá trị tại đó sự chênh lệch giữa hai đoạn là cực đại.

****

****

**Code cài đặt**

1. from matplotlib import image as img
2. import numpy as np
3. import matplotlib.pyplot as plt
4. import skimage.io
5. image = skimage.io.imread("Image/page.png", as\_gray=True)
6. # Set total number of bins in the histogram
7. bins\_num = 256
8. # Get the image histogram
9. hist, bin\_edges = np.histogram(image, bins=bins\_num)
10. # Calculate centers of bins
11. k\_mids = (bin\_edges[:-1] + bin\_edges[1:]) / 2.
12. # Tỉ lệ lớp c1 với số lượng điểm ảnh k với số lượng điểm ảnh p1\_k tương tự p2\_k
13. p1\_k = np.cumsum(hist)
14. p2\_k = np.cumsum(hist[::-1])[::-1]
15. # Get the class means C1(t)
16. mean1 = np.cumsum(hist \* k\_mids) / p1\_k
17. # Get the class means C2(t)
18. mean2 = (np.cumsum((hist \* bin\_mids)[::-1]) / p2\_k[::-1])[::-1]
19. variance\_b = p1\_k[:-1] \* p2\_k[1:] \* (mean1[:-1] - mean2[1:]) \*\* 2
20. # Độ chênh lệch cực đại
21. max\_val\_b = np.argmax(variance\_b)
22. threshold = k\_mids[:-1][max\_val\_b]
23. image[image<threshold] = 0
24. image[image>threshold] = 1
25. img.imsave("otsu\_thresholding.png",image)

Kết quả: t = 0.615234327

# Phân vùng ảnh phương pháp ngưỡng thích ứng

**Yêu cầu:** 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.

**Khái niệm:** Lựa chọn ngưỡng tự động trong vùng lân cận. Ngưỡng t tại thời điểm xét sẽ được dựa trên thống kê của vùng lân cận quanh điểm ảnh. Giống như cách hoạt động của *Convolutions.*

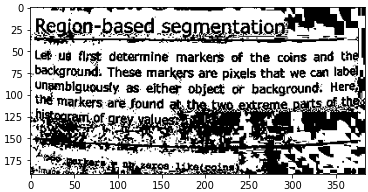
Ngưỡng t có thể là:

T =

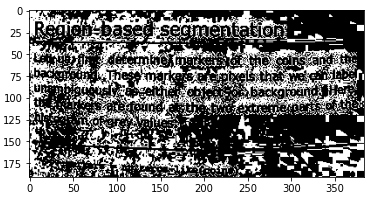
**Bài làm**

1. import numpy as np
2. import matplotlib.pyplot as plt
3. import skimage
4. def threshold\_adaptive(image, block\_size=3, method='mean', C=0):
5. block\_size = (block\_size,) \* image.ndim
6. block\_size = tuple(block\_size)
7. image = image.astype('float64', copy=False)
8. thresh\_image = np.zeros(image.shape, dtype='float64')
9. if method == 'mean':
10. # calculate the mean in each block
11. for i in range(image.shape[0]):
12. for j in range(image.shape[1]):
13. block = image[i:i+block\_size[0], j:j+block\_size[1]]
14. thresh\_image[i,j] = block.mean()
15. elif method == 'median':
16. # calculate the median in each block
17. for i in range(image.shape[0]):
18. for j in range(image.shape[1]):
19. block = image[i:i+block\_size[0], j:j+block\_size[1]]
20. thresh\_image[i,j] = np.median(block)
21. elif method == 'min\_max':
22. # calculate the min and max in each block
23. for i in range(image.shape[0]):
24. for j in range(image.shape[1]):
25. block = image[i:i+block\_size[0], j:j+block\_size[1]]
26. thresh\_image[i,j] = (block.max() - block.min()) / 2
27. return thresh\_image + C
28. # load the image
29. image = skimage.io.imread("../Image/page.png", as\_gray=True)
30. binary\_image1 = image > threshold\_adaptive(image, 15, 'min\_max')
31. img.imsave("otsu\_thresholding.png", binary\_image1)

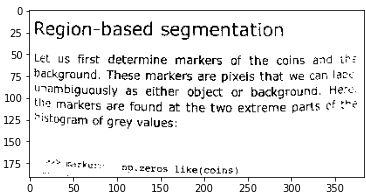
Kết quả phương pháp t = mean + c



Kết quả phương pháp t = median + c



Kết quả phương pháp t =



# Biến đổi âm bản

**Yêu cầu:** Ý nghĩa của phép biến đổi âm bản. Thực hiện trên ảnh page.png.

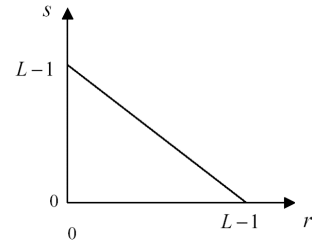
**Bài làm:**

**Ý nghĩa của biến đổi âm bản:**

Biến đổi âm bản giúp tăng cường các chi tiết trắng hoặc xám nằm trong các vùng tối của ảnh. Đặc biệt là các vùng đen sẽ chiến diện tích nhiều hơn.

Để tạo ảnh âm bản, sử dụng hàm đảo ngược độ sáng của ảnh gốc.





Trong đó: Các mức độ sáng của ảnh nằm trong đoạn [0, L-1]

L = 256 là số lượng mức xám, r là mức xám đầu vòa, s là mức sáng đầu ra.

|  |  |
| --- | --- |
| **Triển khai trên ảnh page.png**   1. #negative transformation function 2. def neg\_trans(img): 3. #get width and height of image 4. img = np.array(img) 5. width,height = img.shape 6. # get number of brightness levels of image 7. L = 256 8. for x in range(width): 9. for y in range(height): 10. I=img[x,y] 11. #s=(L-1)-r 12. S = (L - 1) - I 13. #replace the pixel 14. img[x,y]  = S 15. return img | **Ảnh gốc:**    **Ảnh biến đổi âm bản** |

**Nhận xét:** Ta thấy mức xám đầu ra “ngược” với đầu vào, tức là thuật toán này biến ảnh đen trở thành ảnh trắng và trắng thành đen.

# Biến đổi log

**Yêu cầu:** 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.

**Bài làm:**

**Khi nào cần dùng biến đổi log:**

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 sử dụng biến đổi log kiến ảnh rõ nét hơn.

+ Phù hợp với những ảnh tối.

+ 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.

**Phương pháp biến đổi**

|  |  |
| --- | --- |
| Quan sát đồ thị thấy đường log làm cho ảnh trở lên sáng hơn. |  |

**Triển khai biến đổi log**

1. import numpy as np
2. from PIL import Image
3. import math
4. import matplotlib.pyplot as plt
5. im = Image.open("../Image/Fig0323(a)(mars\_moon\_phobos).tif")
6. im\_arr = np.array(im)
7. c= 255/math.log(1+np.max(im\_arr))
8. d= 5000
9. im\_out = (c\*np.log(1+ (np.exp(d)-1)\*im\_arr))
10. im\_transformed = Image.fromarray(im\_out)
11. plt.imshow(im\_transformed, cmap='gray')

|  |  |
| --- | --- |
| **Ảnh gốc** | **Ảnh biến đổi log** |

**Nhận xét:** Quan sát ảnh đầu ra, ta thấy ảnh sau khi được biến đổi sáng hơn so với ảnh gốc.

# Biến đổi mũ

**Yêu cầu:** 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.

**Bài làm**

**Khi nào cần dùng phép biến đổi mũ trên ảnh:**

Phương pháp này giúp tăng hoặc giảm giá trị điểm ảnh đầu vào theo một mức lũy thừa cố định.

Khi ta xây dựng ứng dụng có thể xử dụng phương pháp này để điều chỉnh độ sáng và độ tối của ảnh thông qua việc điều chỉnh hằng số mũ.

**Phương pháp biến đổi**

|  |  |
| --- | --- |
| Biến đổi lũy thừa được cho bằng phương trình: s=c  Với r là mức xám đầu vào, s là mức xám đầu ra, c, γ là các hằng số dương.  • γ < 1: làm cho ảnh sáng hơn.  • γ > 1: làm cho ảnh tối hơn.  c: được tính như sau:  Ta thấy trên đồ thị:  • r=0 thì s=0  • r=L-1 thì s= L-1 |  |

1. import numpy as np
2. import matplotlib.image as img\_pyp
3. import math
4. import matplotlib.pyplot as plt
5. def exponential\_transform(c, r, gamma):
6. return c \* np.array(r \*\* gamma, dtype = 'uint8')
7. # Đọc ảnh gốc
8. img = img\_pyp.imread("../Image/Fig0309(a)(washed\_out\_aerial\_image).tif")
9. img = np.array(img, dtype = np.uint8)
10. # ảnh được biến đổi với hàm mũ
11. gamma= 3
12. L = 256
13. c = (L-1) / (L-1)\*\* gamma
14. exp\_image = exponential\_transform(c, img, gamma)
15. plt.imshow(exp\_image)
16. plt.show()

|  |  |
| --- | --- |
| C:\Users\Admin\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\C95589FB.tmp  C:\Users\Admin\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\A327177.tmp | C:\Users\Admin\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\630FB0C1.tmp  **C:\Users\Admin\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\E1CE72B3.tmp** |

**Nhận xét:** Ta thấy rằng khi gamma > 1 thì ảnh sẽ tối hơn so với gamma < 1.

# Kéo giãn vùng giá trị ảnh

**Yêu cầu:** 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.

**Bài làm:**

**Vì sao cần kéo giãn vùng giá trị ảnh:**

Kéo giãn vùng ảnh giúp cải thiện độ tương phản của hình ảnh giúp ta có thể thấy rõ những chi tiết mà mắt thường khó nhìn được.

**Cách hoạt động:**

Phương pháp này áp dụng trực tiếp trên hình ảnh bằng cách sửa đổi từng pixel có trong hình ảnh.

Sử dụng công thức chuẩn hóa để tính giá trị cường độ của pixel được sửa đổi.

Trong đó:

* Giá trị pixel được sửa
* Giá trị pixel ảnh đầu vào
* Min\_i Giá trị pixel nhỏ nhất của ảnh đầu vào
* max\_i Giá trị pixel lớn nhất của ảnh đầu vào
* min\_0 Giá trị pixel nhỏ nhất của ảnh đầu ra
* max\_0 Giá trị pixel lớn nhất của ảnh đầu ra

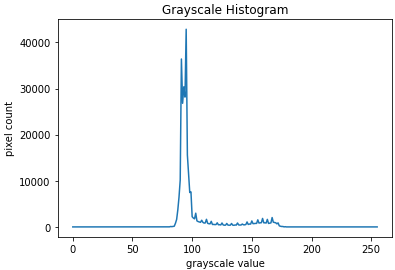
**Code cài đặt**

1. import numpy as np
2. import matplotlib.pyplot as plt
3. from PIL import Image
4. image = Image.open("../Image/Fig0354(a)(einstein\_orig).tif")
5. def contract\_stretching(image):
6. a = 0
7. b = 255
8. c = np.max(image)
9. d = np.min(image)
10. im\_out = (image - c) \* ((a - b) / (c - d)) + a
11. return im\_out
12. image\_out = contract\_stretching(image)
13. fig, ax = plt.subplots()
14. plt.imshow(image\_out, cmap='gray')
15. plt.show()

|  |  |
| --- | --- |
|  |  |

**Hiển thị histogram**

1. histogram, bin\_edges = np.histogram(image, bins=256, range=(0, 255))
2. plt.figure()
3. plt.title("Grayscale Histogram")
4. plt.xlabel("grayscale value")
5. plt.ylabel("pixel count")
6. plt.plot(histogram)
7. plt.show()

****

# Cân bằng histogram và histogram đặc trưng.

**Yêu cầu:** 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

**Bài làm:**

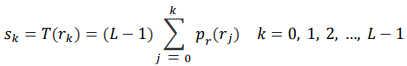
**Mục đích:** Cân bằng histogram là làm cho histogram đồng đều. Khi đó ta sẽ làm tăng độ tương phản của ảnh.

Phương trình cân bằng histogram:



Trong đó : Xác suất xảy ra mức xám w

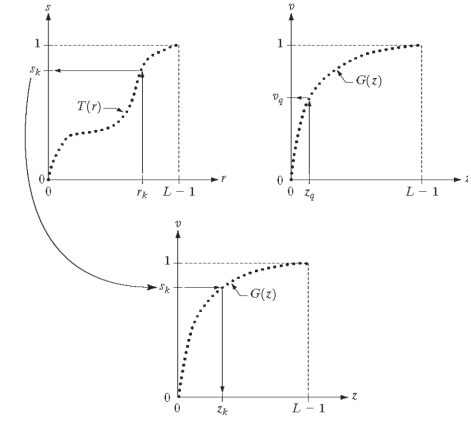
Trong xác suất, tích phân của hàm mật độ là phân phối. Công thức trên có w là biến liên tục, ta không thể lập trình nó. Ta phải dùng công thức rời rạc:



**Thực hiện cân bằng ảnh**

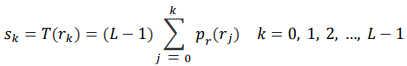
|  |  |
| --- | --- |
| 1. import numpy as np 2. import matplotlib.pyplot as plt 3. import matplotlib.image as im 4. img = im.imread('../Image/Fig0320(2)(2nd\_from\_top).tif') 5. # pr so diem anh 6. pr, bins = np.histogram(img, bins = 256, range=(0,256)) 7. # mat do 8. cr = pr / (img.shape[0] \* img.shape[1]) 9. # bins[:-1] bo khoang cuoi 10. plt.plot(bins[:-1], pr) 11. # Tinh phan phoi tich luy 12. sum = 0 13. for i in range(cr.shape[0]): 14. sum += cr[i] 15. cr[i] = sum 16. s = np.empty\_like(256) 17. s = (256 - 1) \* cr 18. s = s.astype('uint8') 19. equalized\_img = np.empty\_like(img) 20. for i in range(img.shape[0]): 21. for j in range(img.shape[1]): 22. equalized\_img[i][j] = s[img[i][j]] 23. plt.imshow(equalized\_img, cmap='gray') |  |

Không phải cân bằng histogram lúc nào cũng tốt, nên đôi khi ta phải đưa ảnh về histogram như ta mong muốn. Chẳng bạn việc tăng cường độ quá nhanh làm ảnh bị vỡ.



Tính histogram của ảnh rồi suy ra hàm biến đổi histogram tự động





Cho histogram mong muốn pz(zi) càng gần pr càng tốt.

Biến đổi histogram mong muốn.



Cho rk, ta suy ra zk như sau:



Mục tiêu: có mức xám đầu vào rk, ta suy ra mức xám đầu ra zk.

**Code triển khai**

1. import numpy as np
2. import matplotlib.pyplot as plt
3. from PIL import Image
4. from skimage import io
5. import matplotlib.image as im
6. img = im.imread('../Image/Fig0316(3)(third\_from\_top).tif')
7. # pz so diem anh
8. pz, bins = np.histogram(img, bins = 256, range=(0,256))
9. plt.plot(bins[:-1], pz)
10. pz\_specified = pz/(img.shape[0] \* img.shape[1])
11. # nr so diem anh
12. nr, bis = np.histogram(img, bins = 256, range=(0,256))
13. # mat do
14. pr = nr / (img.shape[0] \* img.shape[1])
15. pr = pr.astype('uint8')
16. # bins[:-1] bo khoang cuoi
17. plt.plot(bis[:-1], nr)
18. # Tinh phan phoi tich luy
19. # s dc xac dinh trong phan can bang anh
20. sum = 0
21. arr\_nr = np.zeros\_like(nr)
22. for i in range(nr.shape[0]):
23. sum += nr[i]
24. arr\_nr[i] = sum
25. s = np.empty\_like(256)
26. s = (256 - 1) \* arr\_nr
27. s = s.astype('uint8')
28. gz = np.zeros\_like(pz\_specified)
29. sum\_z = 0
30. # xac xuat tich luy
31. for i in range(pz\_specified.shape[0]):
32. sum\_z += pz\_specified[i]
33. gz[i] = sum\_z
34. gz = (256 - 1) \* gz
35. gz = gz.astype('uint8') # Lam tron
36. zq = []
37. for i in range(s.shape[0]):
38. # find the corresponding value of z\_q. Gz is closet to s
39. zq.append(np.where(gz==s[i])[0])
40. pz\_actual = np.zeros\_like(pz\_specified)
41. for i in range(s.shape[0]):
42. if len(zq[i]) == 0:
43. pz\_actual[i] = 0
44. else:
45. sum = 0
46. for j in zq[i]:
47. sum += nr[j]
48. pz\_actual[i] = sum / (img.shape[0] \* img.shape[1])

# Histogram cục bộ.

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

**Ý tưởng:** Trong trường hợp 2 mức xám kề nhau chêch lệch quá nhỏ, nếu ta cân bằng histogram tự động sẽ không có kết quả. Trong trường hợp đó ta phải dùng histogram cục bộ.

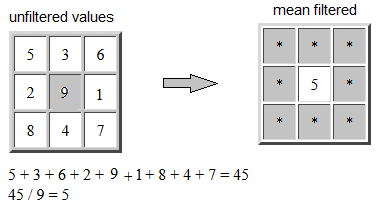
Chọn cửa sổ có kích thước lẻ để có phần tử trung tâm, ví dụ: m=n=3, cân bằng histogram của dữ liệu lấy từ cửa sổ và thay phần tử trung tâm bằng phần tử mới.

**Bài làm**

1. Def Localhistogram(w, m, n, p):
2. For k in range(L):
3. P[k] = 0
4. For x in range(m):
5. For y in range(n):
6. R = w[x][y]
7. P[r] ++
8. For k in range(L):
9. P[k] = p[k] / (m\*n)
10. Return
11. Def LocalHistogramEqualizetion(p, T):
12. For k in range(L):
13. T[k] = 0
14. For j in range(k):
15. T[k] = T[k] + p[j]
16. Return
17. Def LocalEnhencement(f, g):
18. M = 3, n=3
19. A , b= m/2, n/2
20. W = np.array(m, n)
21. For x in range(M-a):
22. For y in range(N-b):
23. For s in range(-a, a):
24. for t in range(-b, b):
25. W[s+a][t+b] = f[x+s][y+t]
26. LocalHistogram(w, m, n, p)
27. LocalHistogramEqualization(p, T)
28. R = f[x][y]
29. G[x][y] = int((L-1) \* T[r])

# Lọc trung bình.

**Yêu cầu:** 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.



**Khái niệm:** Nhiễu Gauss: Là mô hình xấp xỉ nhiễu phổ biến hiện nay. Hàm mật độ theo phân phối Gauss với giá trị trung bình 𝑧 và độ lệch chuẩn σ

**Triển khai code**

**Dùng thư viện skimage**

1. from skimage import io
2. from scipy.ndimage.filters import convolve
3. import numpy as np
4. im  =io.imread("../Image/Fig0441(a)(characters\_test\_pattern).tif")
5. k = (1/9)\*np.ones((3,3),dtype = 'float32')
6. # mean filter of im
7. im\_mean = convolve(im,k)
8. io.imshow(im\_mean)

**Code thủ công bằng python.**

* 1. import matplotlib.image as img
  2. import numpy as np
  3. from PIL import Image
  4. import matplotlib.pyplot as plt
  5. image = img.imread("../Image/Fig0441(a)(characters\_test\_pattern).tif")
  6. w,h = image.shape[:2]
  7. width = image.shape[1]
  8. height = image.shape[0]
  9. result = np.zeros((image.shape[0], image.shape[1]), int)
  10. def meanFilter(image):
  11. for row in range(height):
  12. for col in range(width):
  13. currentElement=0; left=0; right=0; top=0; bottom=0; topLeft=0;
  14. topRight=0; bottomLeft=0; bottomRight=0;
  15. counter = 1
  16. currentElement = image[row][col]
  17. if not col-1 < 0:
  18. left = image[row][col-1]
  19. counter +=1
  20. if not col+1 > width-1:
  21. right = image[row][col+1]
  22. counter +=1
  23. if not row-1 < 0:
  24. top = image[row-1][col]
  25. counter +=1
  26. if not row+1 > height-1:
  27. bottom = image[row+1][col]
  28. counter +=1
  29. if not row-1 < 0 and not col-1 < 0:
  30. topLeft = image[row-1][col-1]
  31. counter +=1
  32. if not row-1 < 0 and not col+1 > width-1:
  33. topRight = image[row-1][col+1]
  34. counter +=1
  35. if not row+1 > height-1 and not col-1 < 0:
  36. bottomLeft = image[row+1][col-1]
  37. counter +=1
  38. if not row+1 > height-1 and not col+1 > width-1:
  39. bottomRight = image[row+1][col+1]
  40. counter +=1
  41. total = int(currentElement)+int(left)+int(right)+int(top)+int(bottom)

+int(topLeft)+int(topRight)+int(bottomLeft)+int(bottomRight)

* 1. avg = total/counter
  2. result[row][col] = avg
  3. return result
  4. result = meanFilter(image)
  5. im = Image.fromarray(result)
  6. plt.imshow(im)

|  |  |
| --- | --- |
|  |  |

**Vì sao lọc trung bình hiệu quả với nhiễu Gauss?**

Phương pháp lọc trung bình sẽ giúp làm mờ đi các nhiễu gauss bằng cách tính trung bình trên điểm ảnh

# Lọc trung vị.

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

**Khái niệm:** Lọc median ( lọc trung vị ) lấy vị trí chính giữa, hoạt động trên miền không gian của hình ảnh. Là bộ lọc giúp làm mịn và loại bỏ nhiễu mà vẫn đảm bảo các cạnh.

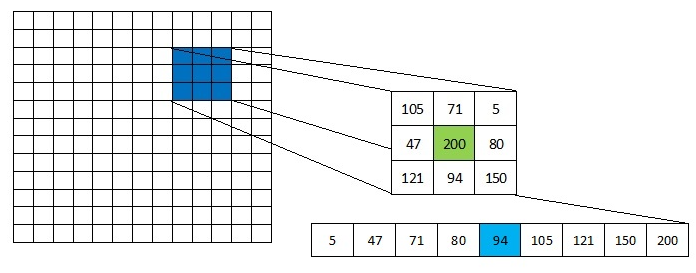
**Nhiễu xung muối tiêu:** Nhiễu xung (Nhiễu muối tiêu-salt&pepper) đặc trưng bởi một điểm ảnh có giá trị mức xám khác biệt lớn so với những điểm lân cận. Xung của nhiễu có thể âm hoặc dương. Xung nhiễu âm đại diện cho điểm ảnh đen (pepper), trong khi xung nhiễu dương đại diện cho điểm ảnh trắng (salt). Nhiễu muốn tiêu xuất hiệu khi ảnh bị bão hòa bởi nhiễu xung.

**Vì sao bộ lọc trung vị giảm được nhiễu muối tiêu?**

Lọc median đặc biệt hiệu quả khi trong ảnh có nhiễu xung, nhiễu xung còn được gọi là nhiễu muối tiêu. Nhiễu muối tiêu là những đốm đen và đốm trắng trong ảnh.

* Lọc trung vị loại bỏ các giá trị cực trị khỏi ảnh, giữ lại giá trị thật của ảnh đầu vào
* 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.

Thực hiện lọc median như sau: Lấy số liệu vào từng cửa sổ (3x3). Tạo mảng một chiều, sắp tăng dần và phần tử f(x,y) được thay bằng phần tử chính giữa.



**Bài làm:**

1. import numpy
2. from PIL import Image
3. import matplotlib.pyplot as plt
4. def median\_filter(data, filter\_size):
5. temp = []
6. indexer = filter\_size // 2
7. data\_final = []
8. data\_final = numpy.zeros((len(data),len(data[0])))
9. for i in range(len(data)):
10. for j in range(len(data[0])):
11. for z in range(filter\_size):
12. if i + z - indexer < 0 or i + z - indexer > len(data) - 1:
13. for c in range(filter\_size):
14. temp.append(0)
15. else:
16. if j + z - indexer < 0 or j + indexer > len(data[0]) - 1:
17. temp.append(0)
18. else:
19. for k in range(filter\_size):
20. temp.append(data[i + z - indexer][j + k - indexer])
21. temp.sort()
22. data\_final[i][j] = temp[len(temp) // 2]
23. temp = []
24. return data\_final
25. img = Image.open("../Image/Fig0441(a)(characters\_test\_pattern).tif").convert("L")
26. arr = numpy.array(img)
27. removed\_noise = median\_filter(arr, 3)
28. img = Image.fromarray(removed\_noise)
29. plt.imshow(img)

|  |  |
| --- | --- |
|  |  |