

Computer Vision

THỊ GIÁC MÁY TÍNH

ThS. Huỳnh Minh Vũ

Khoa Kỹ thuật cơ khí

Trường Đại học Kỹ thuật – Công nghệ Cần Thơ

Email: hmvu@ctuet.edu.vn



Chương 3: Xử lý nâng cao chất lượng ảnh

3.1 Giới thiệu

3.2 Biến đổi mức sáng hình ảnh

3.3 Biểu đồ Histogram

3.4 Ảnh nhị phân và nhị phân hóa với ngưỡng động

3.5 Lọc ảnh

3.1 Giới thiệu

Mục tiêu:

- Tăng cường ảnh,
- Tăng cường độ cảm thụ ảnh,
- Nâng cao chất lượng ảnh.

Phương pháp:

- Xử lý điểm ảnh,
- Biến đổi độ tương phản,
- Xử lý biểu đồ mức xám (Histogram),
- Lọc không gian (lọc nhiễu, lọc tăng cường độ nét...),
- Lọc tần số.

3.1 Giới thiệu

Ví dụ:



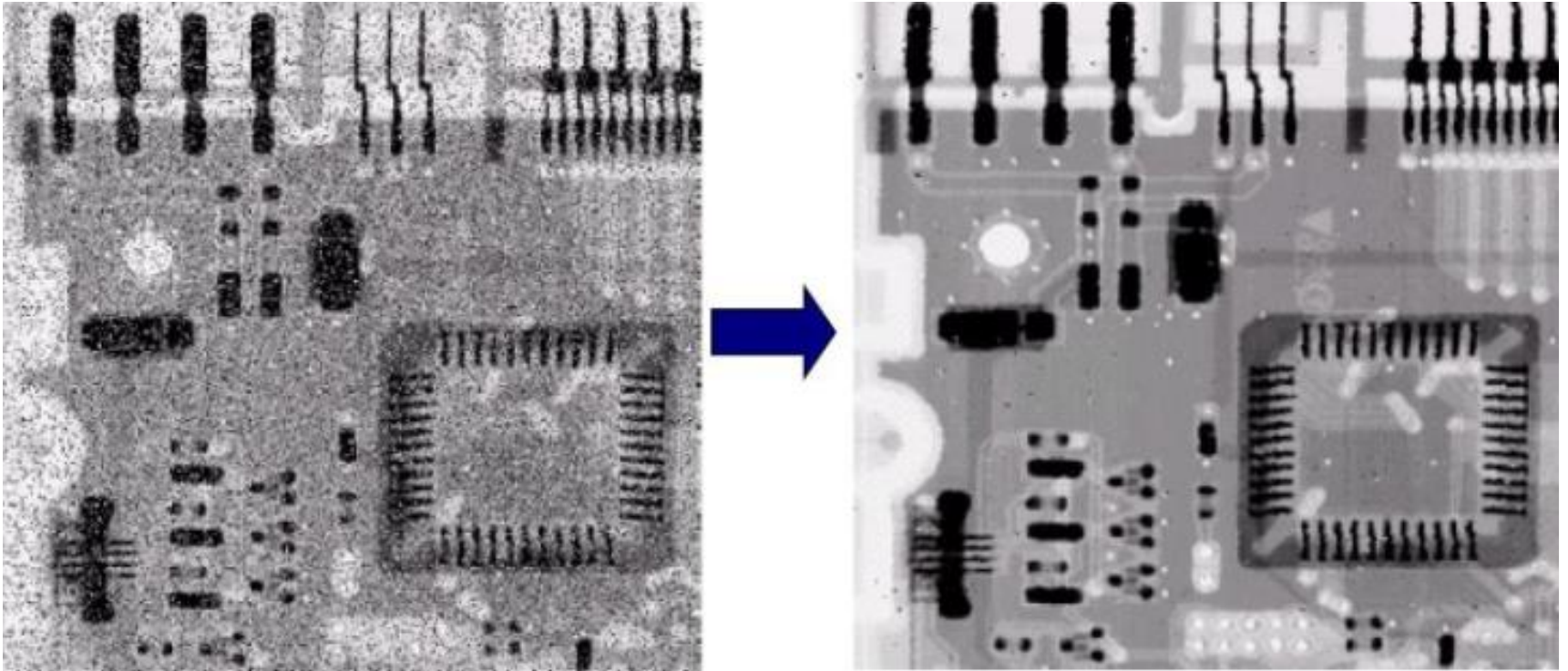
3.1 Giới thiệu

Ví dụ:



3.1 Giới thiệu

Ví dụ:



3.1 Giới thiệu

Ví dụ:



3.2 Biến đổi mức sáng hình ảnh

Điều chỉnh mức sáng các pixel trong ảnh theo công thức:

$$g(x, y) = a.f(x, y) + b$$

Trong đó:

- **a, b:** hằng số.
 - **a:** là giá trị đặc trưng cho độ tương phản.
 - **b:** là lượng độ sáng thay đổi
- **f(x,y):** mức sáng của pixel trong ảnh tại điểm có tọa độ (x,y).
- **g(x,y):** mức sáng của pixel trong ảnh tại điểm có tọa độ (x,y) sau khi biến đổi.

3.2 Biến đổi mức sáng hình ảnh



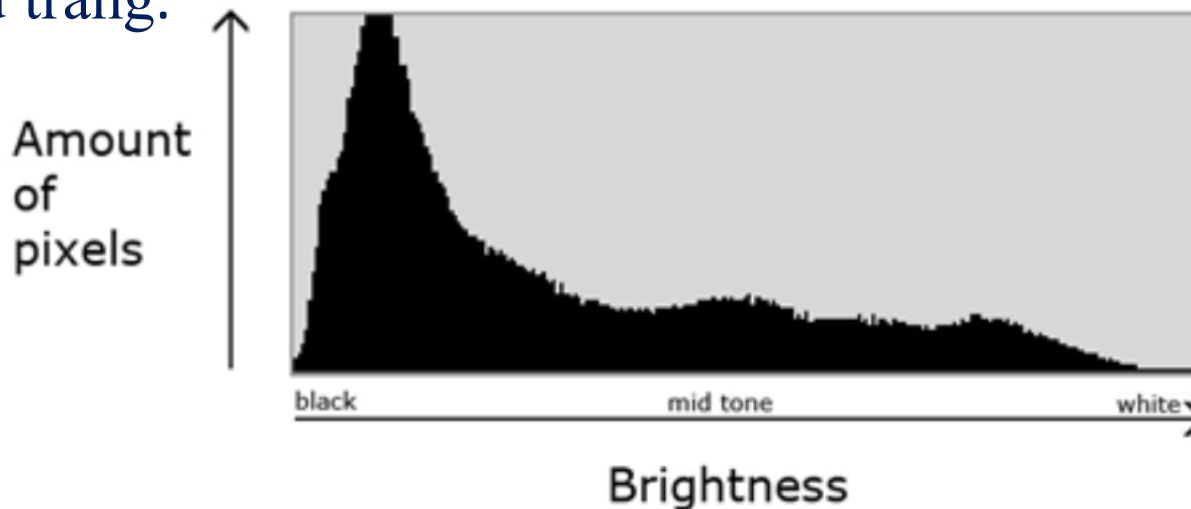
```
import cv2
import numpy as np
img = cv2.imread('GauTruc.png')
alpha = 1
beta = 40

img_new = cv2.addWeighted(img, alpha, np.zeros(img.shape, img.dtype), 0, beta)

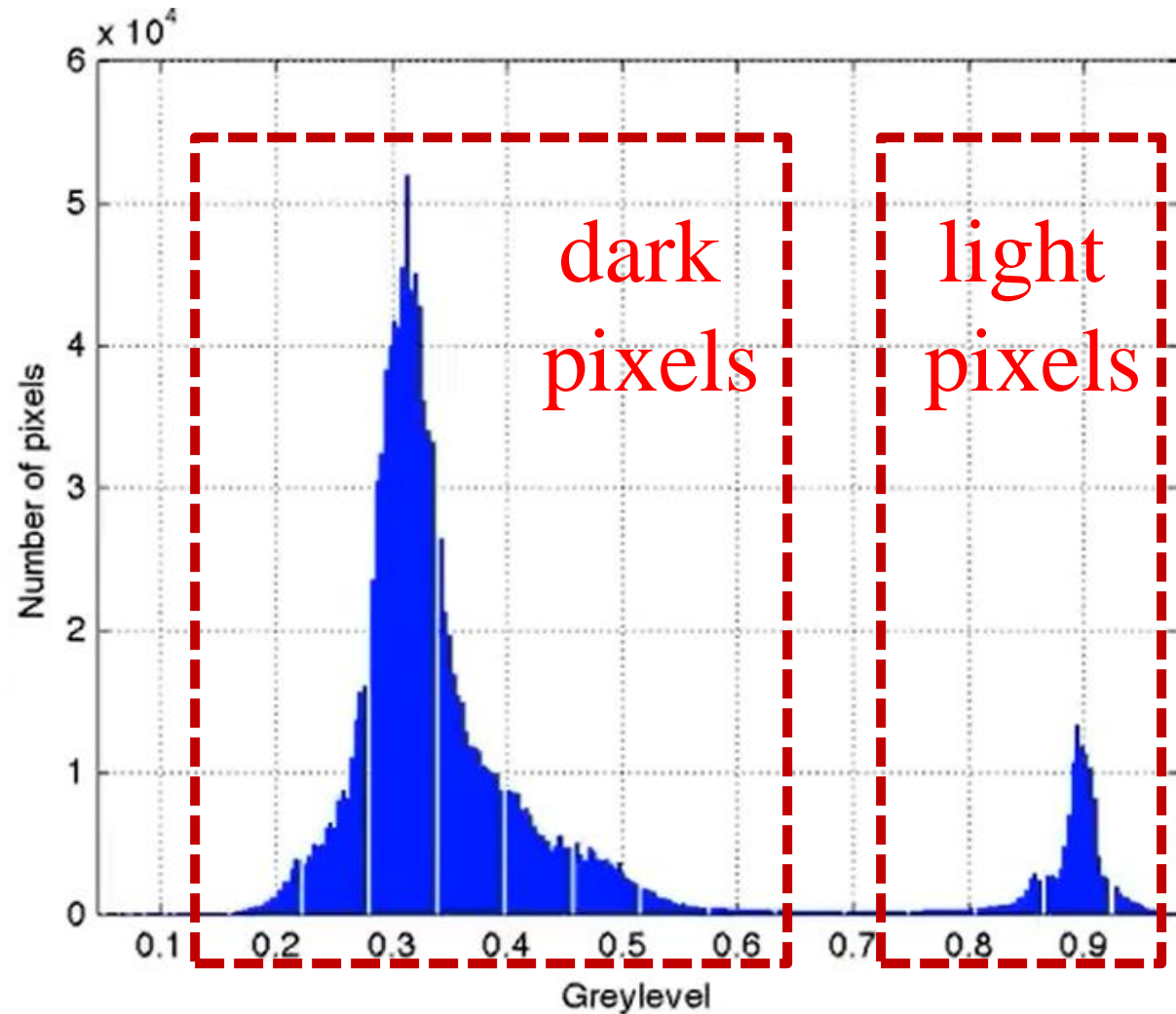
cv2.imwrite('img_new.jpg', img_new)
```

3.3 Biểu đồ Histogram

- Biểu đồ histogram của một ảnh là một biểu đồ nói lên mối quan hệ giữa các giá trị của pixel ảnh (điểm ảnh) và tần suất xuất hiện của chúng.
- Biểu đồ có 2 chiều với trục X là các giá trị màu của ảnh, trục Y là tổng số pixel. Với ảnh có 8 bit màu mỗi kênh thì trục X có 256 cột giá trị từ 0 – 255. Từ trái qua phải là bit có cường độ sáng thấp nhất là màu đen đến số lượng pixel có cường độ sáng sáng nhất là màu trắng.



3.3 Biểu đồ Histogram



3.3 Biểu đồ Histogram

- Histogram của một ảnh số với mức xám trong khoảng $[0, L-1]$ được mô tả bởi công thức:

$$H(r_k) = n_k$$

Trong đó: r_k là mức xám với k trong khoảng $[0, L-1]$ và n_k là số điểm ảnh tương ứng có cùng mức xám r_k . Ví dụ: $r_0 = 0, r_1 = 1, \dots$ và $L = [0, 8]$ (ảnh 3 bit), $L = [0, 255]$ (ảnh 8 bit).

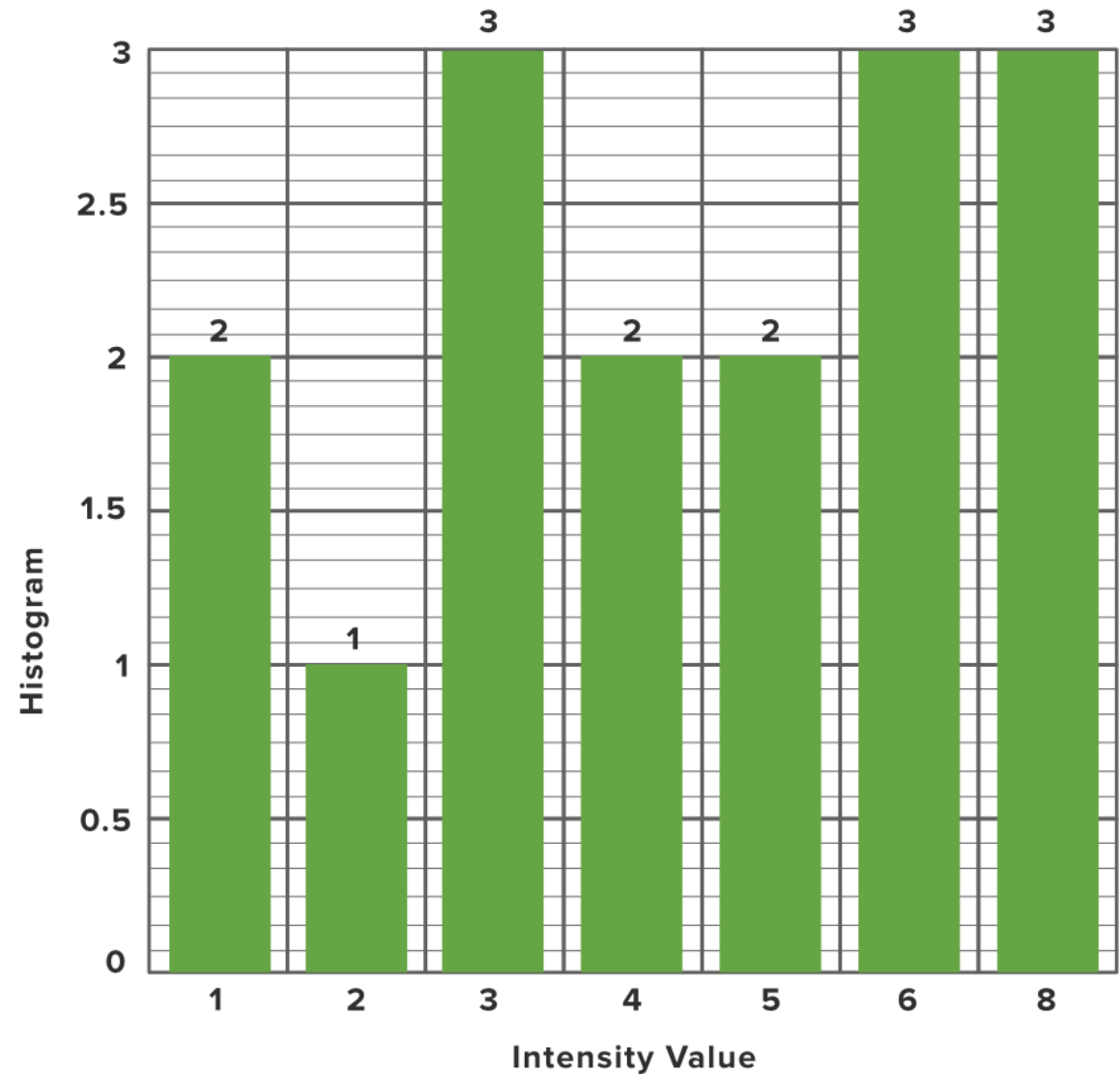
- Hàm mật độ xuất hiện (xác suất xuất hiện) của giá trị mức xám r_k được xác định theo công thức:

$$P_r(r_k) = \frac{n_k}{n} = \frac{H(r_k)}{n}$$

Trong đó: n là tổng số pixels của ảnh.

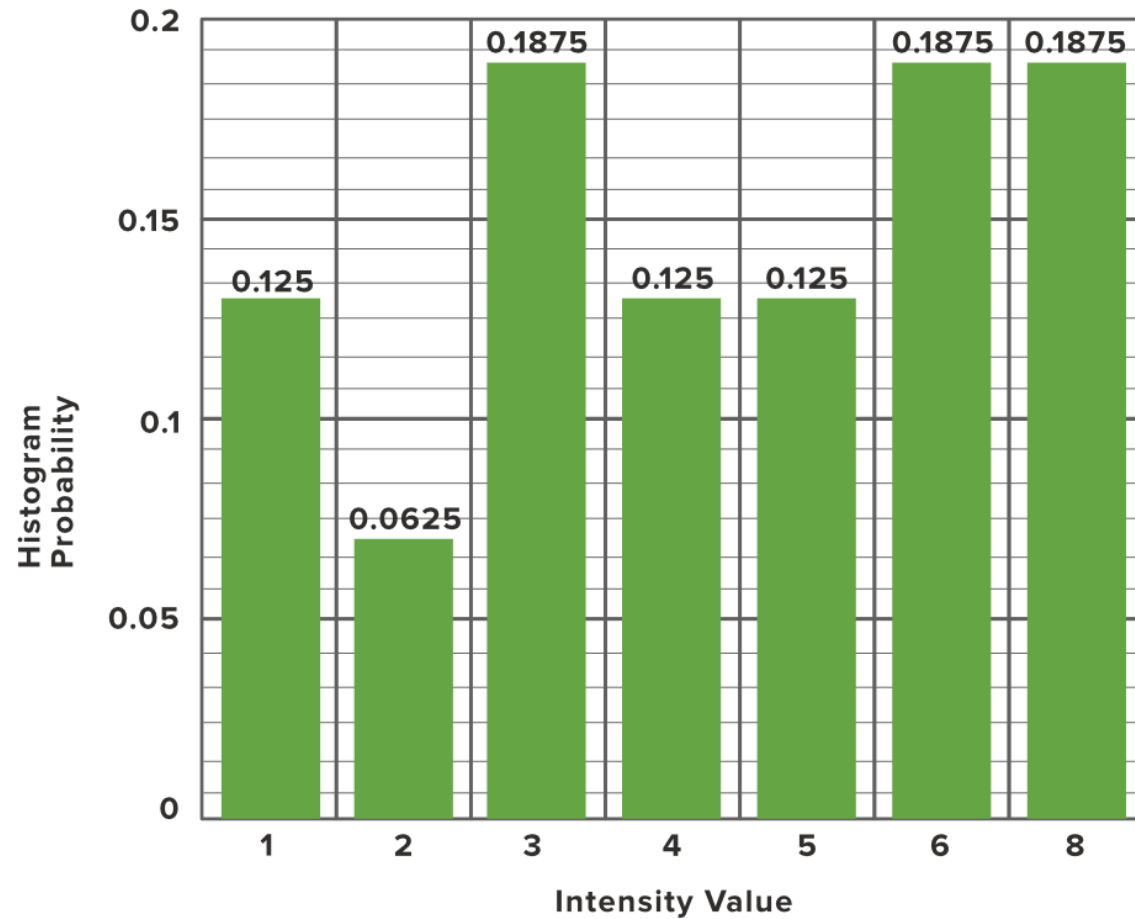
3.3 Biểu đồ Histogram

3	6	6	8
5	3	1	4
8	6	5	1
4	8	2	3



3.3 Biểu đồ Histogram

Gray Level	number of pixels	Probability
1	2	0.125
2	1	0.0625
3	3	0.1875
4	2	0.125
5	2	0.125
6	3	0.1875
8	3	0.1875



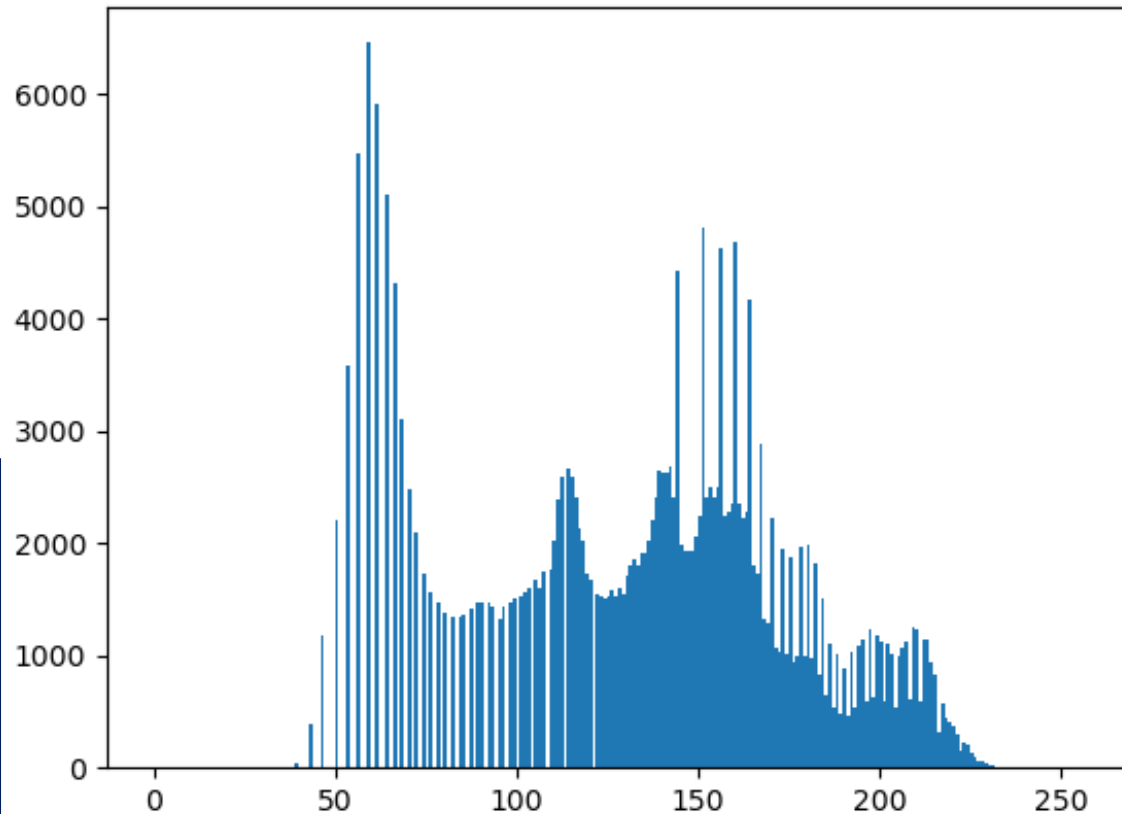
3.3 Biểu đồ Histogram

Biểu đồ Histogram của ảnh xám



```
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('Lena.png',0)
cv2.imshow('AnhGoc',img)
plt.hist(img.ravel(),256,[0,255])
plt.show()
cv2.waitKey()
cv2.destroyAllWindows()
```



3.3 Biểu đồ Histogram

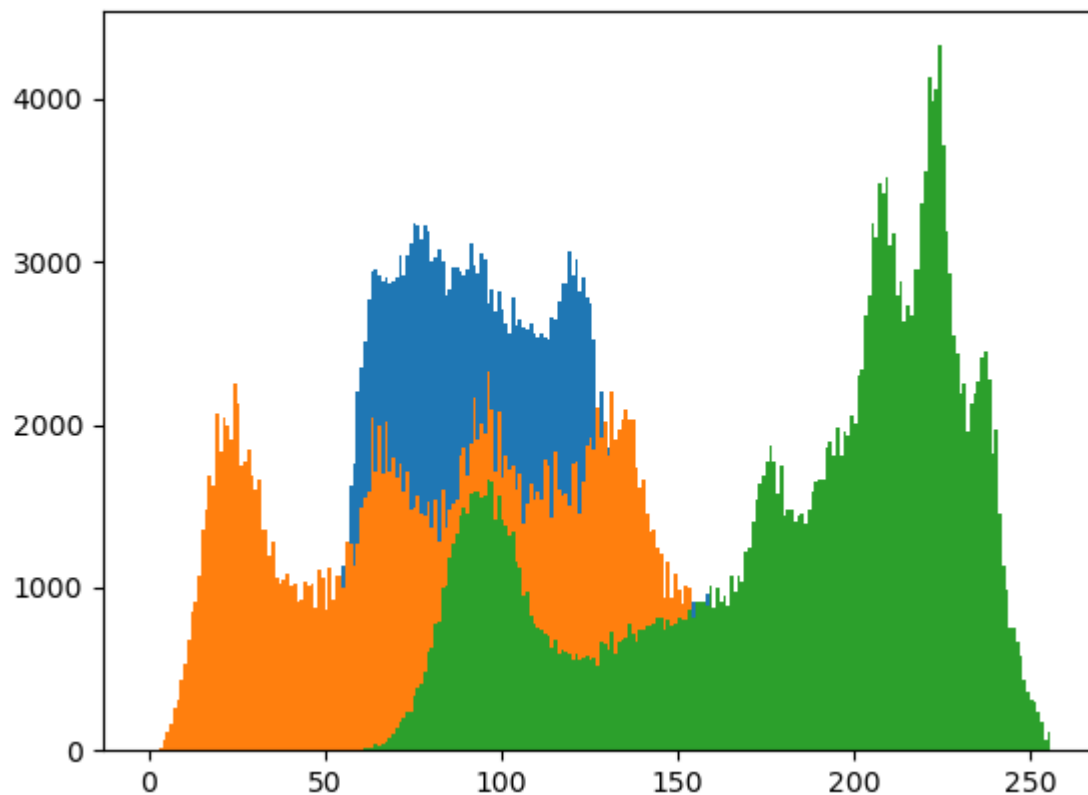
Biểu đồ Histogram của ảnh ảnh màu BGR

```
import cv2 as cv
from matplotlib import pyplot as plt

img = cv.imread("Lena.png")

b, g, r = cv.split(img)
cv.imshow("img", img)
cv.imshow("b", b)
cv.imshow("g", g)
cv.imshow("r", r)

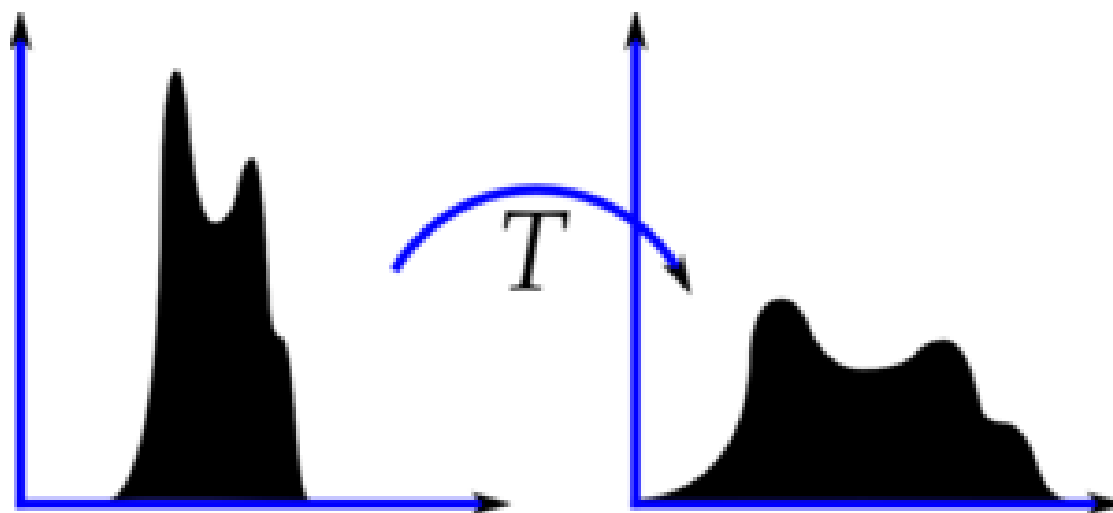
plt.hist(b.ravel(), 256, [0, 255])
plt.hist(g.ravel(), 256, [0, 255])
plt.hist(r.ravel(), 256, [0, 255])
plt.show()
cv.waitKey(0)
cv.destroyAllWindows()
```



3.3 Biểu đồ Histogram

Cân bằng histogram (Histogram Equalization)

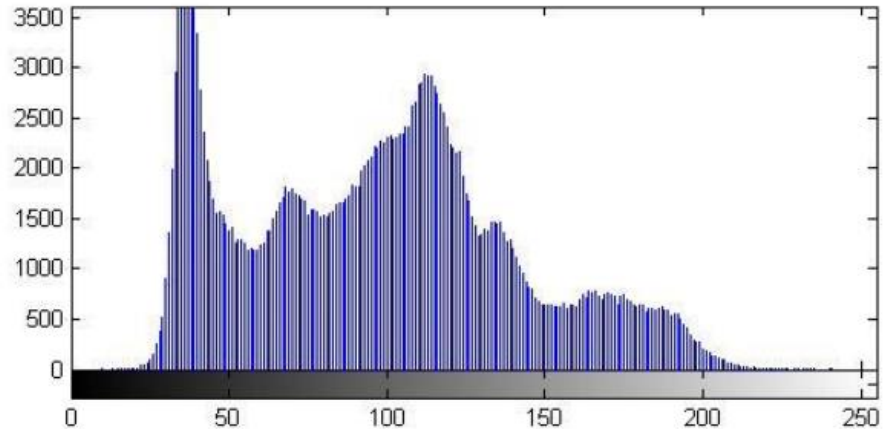
- Cân bằng histogram (histogram equalization) là phương pháp làm cho biểu đồ histogram của ảnh được phân bố một cách đồng đều.
- Đây là một biến đổi khá quan trọng giúp nâng cao chất lượng ảnh, thông thường đây là bước tiền xử lý của một ảnh đầu vào cho các bước tiếp theo.



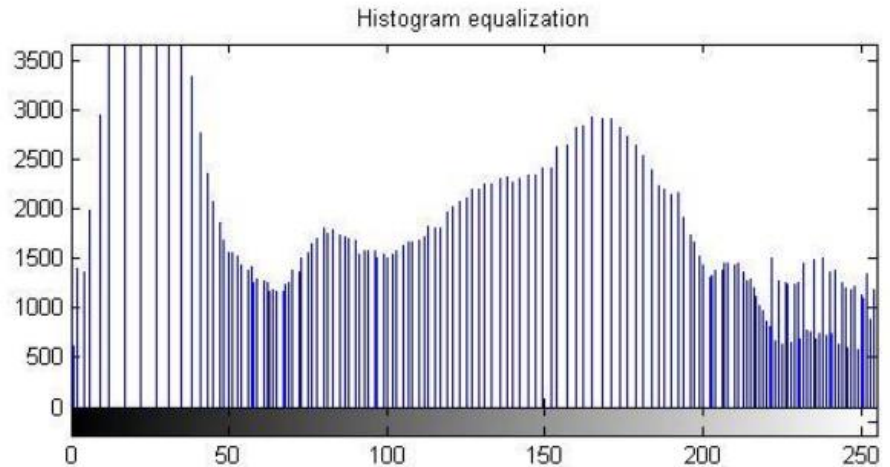
3.3 Biểu đồ Histogram

Cân bằng histogram (Histogram Equalization)

Original Image



Enhanced Image



3.3 Biểu đồ Histogram

Cân bằng histogram (Histogram Equalization)

- Để cân bằng mức xám cho ảnh số ta thực hiện cân bằng trên từng điểm ảnh.
- Công thức cân bằng mức xám cho từng điểm ảnh:

$$s_k = T(r_k) = (L-1) \sum_{j=0}^k P_r(r_j) = \frac{(L-1)}{M \cdot N} \sum_{j=0}^k n_j$$

Trong đó: L là số mức xám của ảnh,
M là số pixels của hàng,
N là số pixels của cột,
 n_j là số pixels của ảnh có mức xám là j,
 $k = 0, 1, 2, \dots, (L-1)$.

3.3 Biểu đồ Histogram

Cân bằng histogram (Histogram Equalization)

$$L = 8, M = N = 64, M.N = 4096$$

$$s_k = T(r_k) = (L-1) \sum_{j=0}^k P_r(r_j)$$

$$s_0 = T(r_0) = 7 \sum_{j=0}^0 p_r(r_j) = 7p_r(r_0) = 1.33$$

$$s_1 = T(r_1) = 7 \sum_{j=0}^1 p_r(r_j) = 7p_r(r_0) + 7p_r(r_1) = 3.08$$

$$s_2 = 4.55, s_3 = 5.67, s_4 = 6.23, s_5 = 6.65, s_6 = 6.86, s_7 = 7.00.'$$

r_k	n_k	$p_r(r_k) = n_k/MN$
$r_0 = 0$	790	0.19
$r_1 = 1$	1023	0.25
$r_2 = 2$	850	0.21
$r_3 = 3$	656	0.16
$r_4 = 4$	329	0.08
$r_5 = 5$	245	0.06
$r_6 = 6$	122	0.03
$r_7 = 7$	81	0.02

3.3 Biểu đồ Histogram

Cân bằng histogram (Histogram Equalization)



3.3 Biểu đồ Histogram

Cân bằng histogram (Histogram Equalization)

- Ví dụ 3.1: Cho ảnh xám có ma trận giá trị điểm ảnh như hình bên dưới. Hãy vẽ biểu đồ Histogram và cân bằng histogram.

20	30	40	10	20
10	80	90	100	60
50	150	160	170	30
30	220	230	240	50
40	60	30	40	20

3.3 Biểu đồ Histogram

Cân bằng histogram (Histogram Equalization)

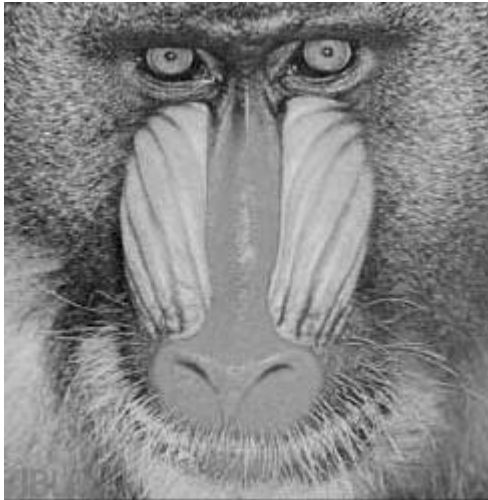
Cường độ	Số lượng	Xác suất	Tỉ lệ cộng dồn	Pixel tính toán 256 bin	Pixel tương ứng
10	2	0.080	0.080	20.480	20
20	3	0.120	0.200	51.200	51
30	4	0.160	0.360	92.160	92
40	3	0.120	0.480	122.880	122
50	2	0.080	0.560	143.360	143
60	2	0.080	0.640	163.840	163
80	1	0.040	0.680	174.080	174
90	1	0.040	0.720	184.320	184
100	1	0.040	0.760	194.560	194
150	1	0.040	0.800	204.800	204
160	1	0.040	0.840	215.040	215
170	1	0.040	0.880	225.280	225
220	1	0.040	0.920	235.520	235
230	1	0.040	0.960	245.760	245
240	1	0.040	1.000	256.000	256

20	30	40	10	20
10	80	90	100	60
50	150	160	170	30
30	220	230	240	50
40	60	30	40	20

51	92	122	20	51
20	174	184	194	163
143	204	215	225	92
92	235	245	256	143
122	163	92	122	51

3.3 Biểu đồ Histogram

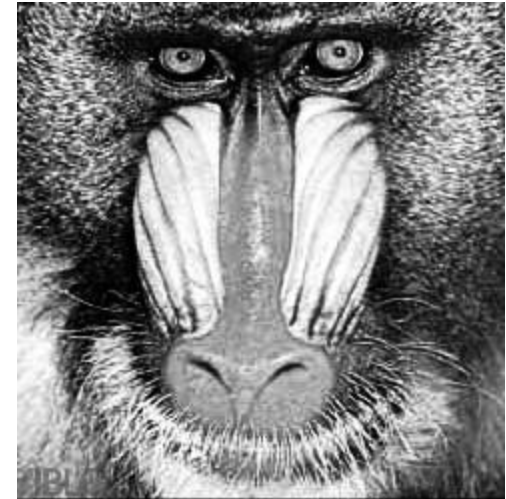
Cân bằng histogram trong OpenCV



Histogram Equalization



`cv2.equalizeHist()`



```
import cv2
import numpy as np

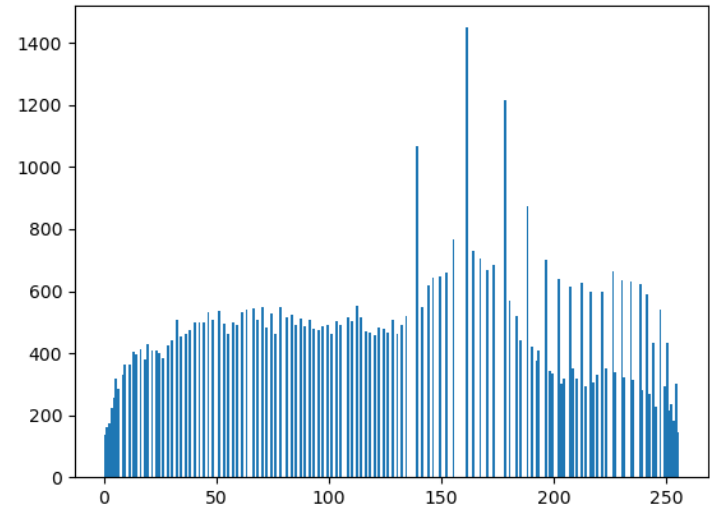
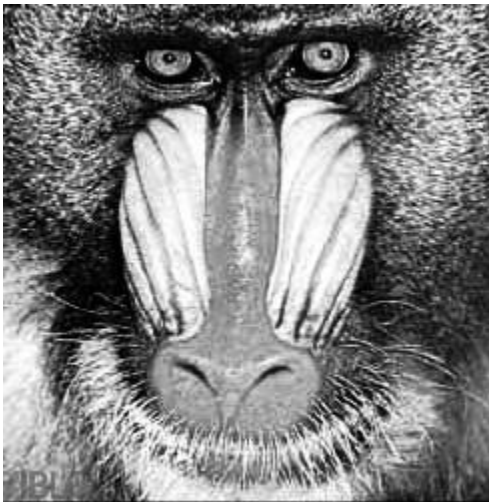
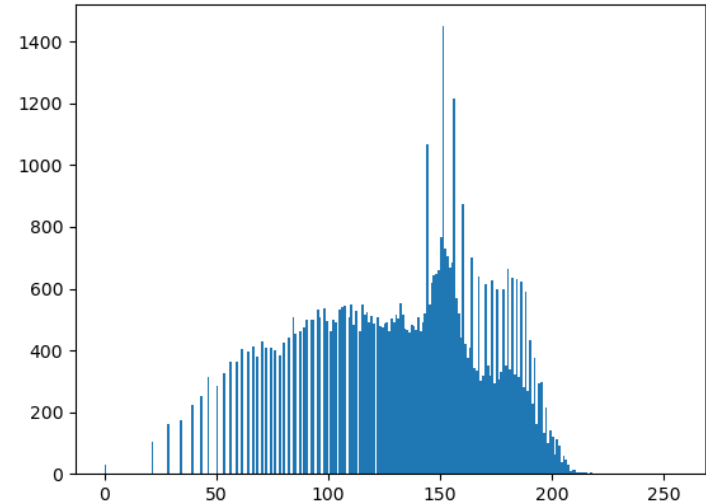
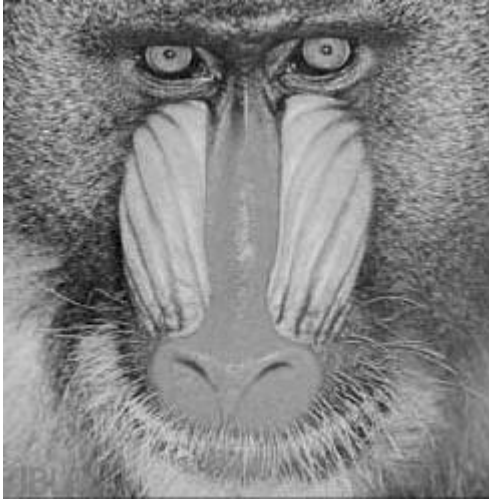
img = cv2.imread('Monkey.png',0)

equ = cv2.equalizeHist(img)

cv2.imwrite('MonkeyOut.png', equ)
```


3.3 Biểu đồ Histogram

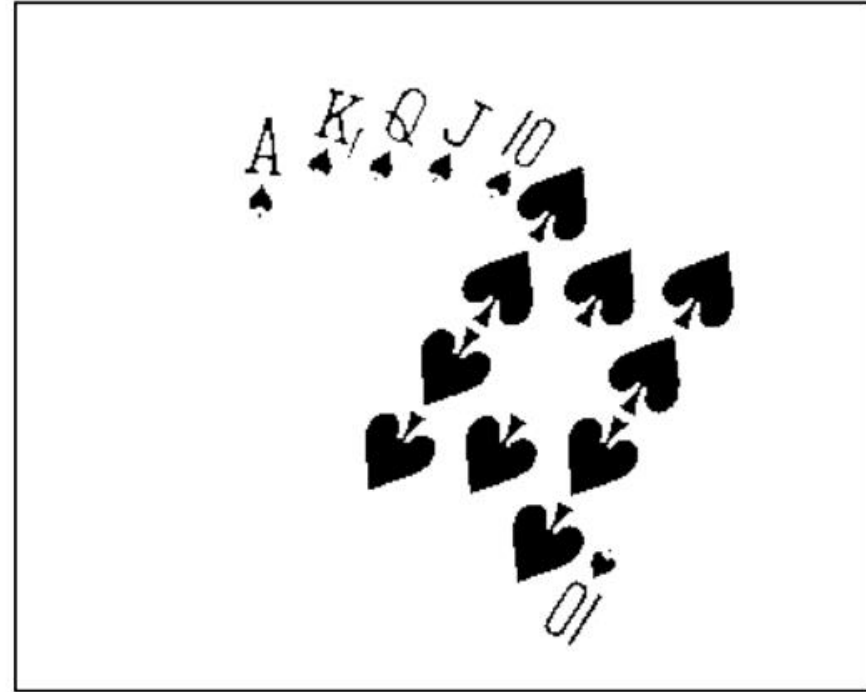
Kiểm tra kết quả cân bằng histogram



3.4 Ảnh nhị phân và nhị phân hóa với ngưỡng động



Original Image



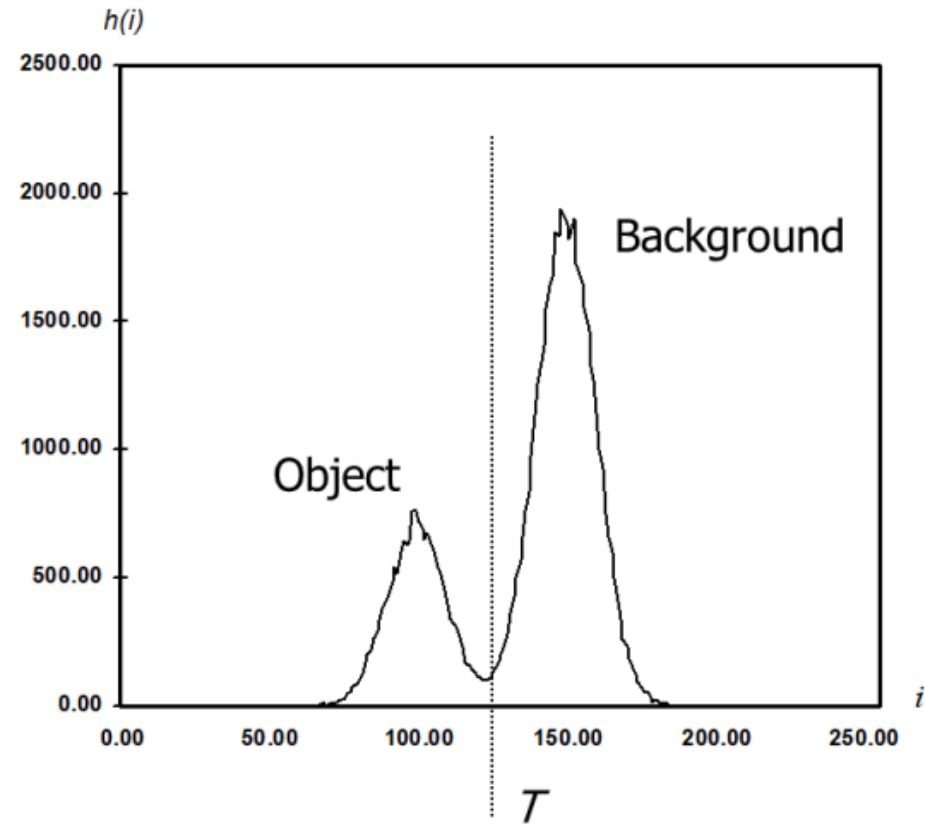
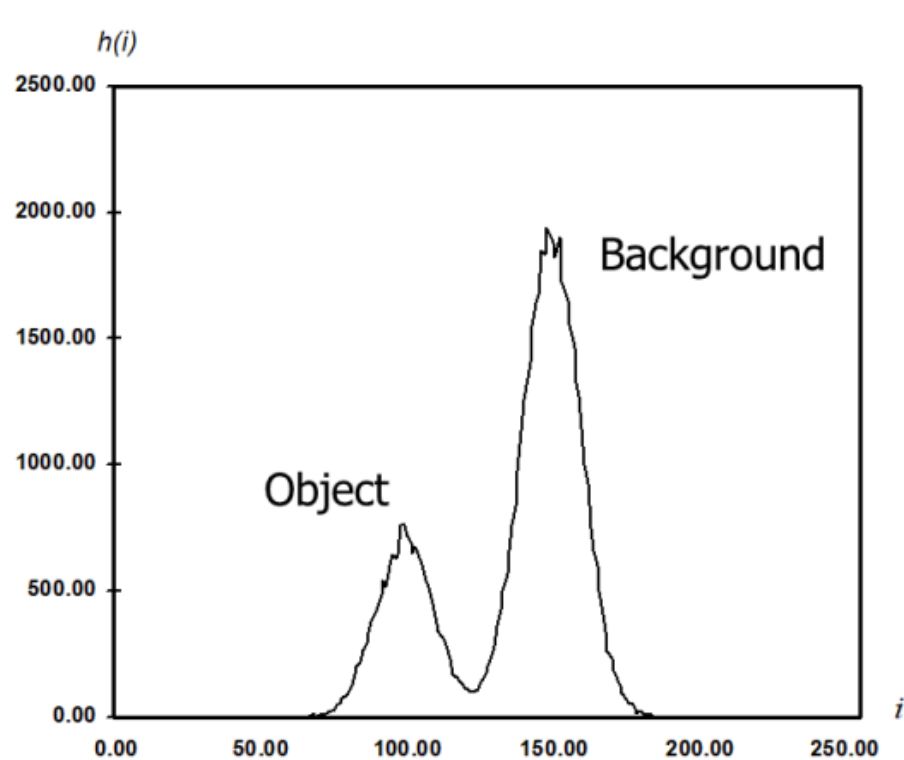
Thresholded Image

3.4 Ảnh nhị phân và nhị phân hóa với ngưỡng động

- Ảnh nhị phân là ảnh mà các giá trị điểm ảnh được biểu diễn bằng hai giá trị là 0 hoặc 255 (hoặc 1), tương ứng với hai màu đen hoặc trắng.
- Nhị phân hóa một ảnh là quá trình biến đổi một ảnh (nên là ảnh xám) thành ảnh nhị phân.
- Nếu gọi $f(x,y)$ là giá trị cường độ sáng của một điểm ảnh ở vị trí (x,y) , T là ngưỡng nhị phân. Khi đó, ảnh xám f sẽ được chuyển thành ảnh nhị phân dựa vào công thức:

$$f(x,y) = \begin{cases} 0, & f(x,y) \leq T \\ 255, & f(x,y) > T \end{cases}$$

3.4 Ảnh nhị phân và nhị phân hóa với ngưỡng động



3.4 Ảnh nhị phân và nhị phân hóa với ngưỡng động

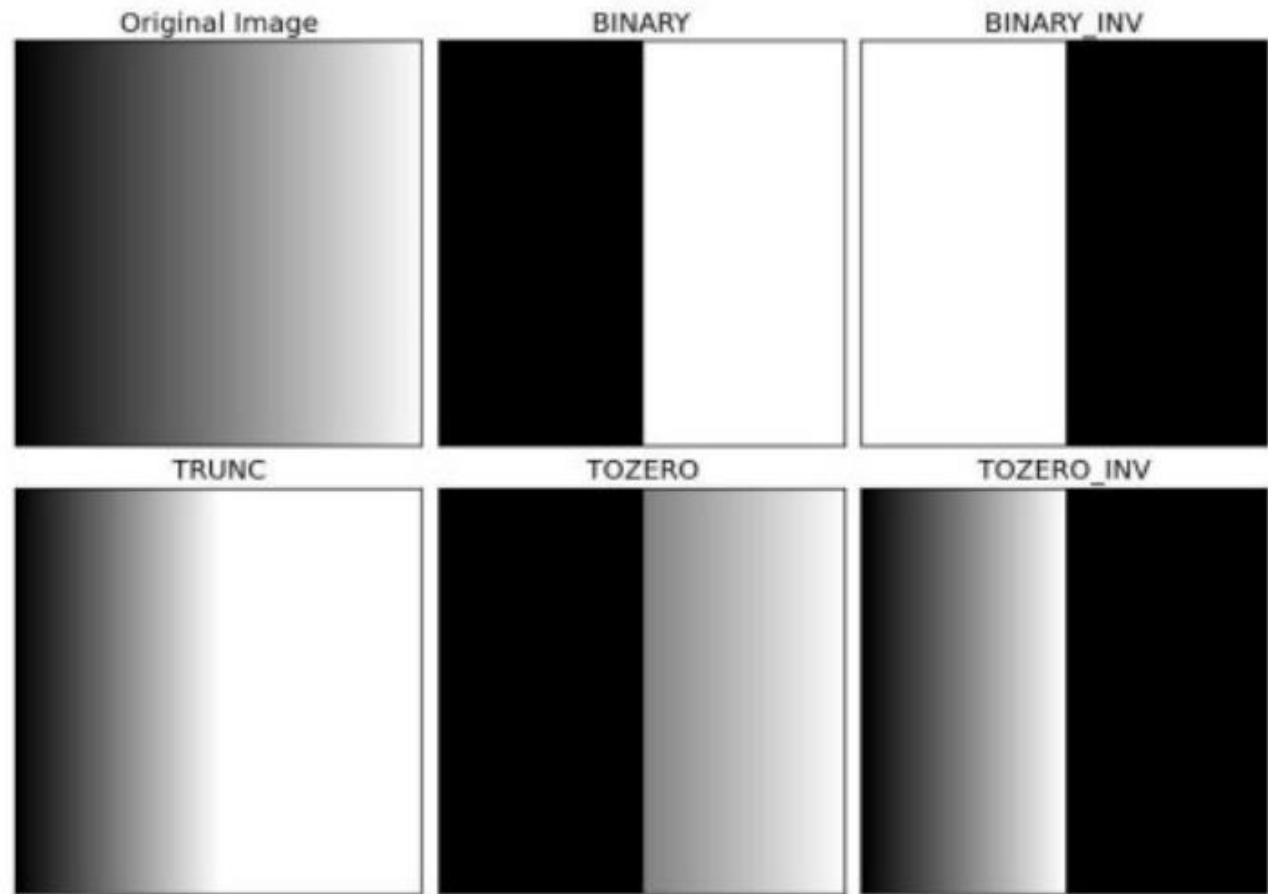
Các bước nhị phân hóa ảnh:

- Biến đổi ảnh màu (color) sang ảnh xám (gray) hoặc đọc ảnh lên bằng cờ ảnh xám,
- Thiết lập ngưỡng (threshold) để nhị phân hóa ảnh,
- Áp dụng ngưỡng vào ảnh xám để tạo ảnh nhị phân hoặc áp dụng giải thuật nhị phân hóa ảnh. Các pixel có giá trị lớn hơn ngưỡng ta thiết lập bằng 255 (hoặc 1), nhỏ hơn ngưỡng ta thiết lập bằng 0.

3.4 Ảnh nhị phân và nhị phân hóa với ngưỡng động

Các kiểu phân ngưỡng (threshold) trong OpenCV:

`cv2.THRESH_BINARY`
`cv2.THRESH_BINARY_INV`
`cv2.THRESH_TRUNC`
`cv2.THRESH_TOZERO`
`cv2.THRESH_TOZERO_INV`



3.4 Ảnh nhị phân và nhị phân hóa với ngưỡng động

Nhị phân hóa với ngưỡng T



```
import cv2
img = cv2.imread('Lena.png',0) # tham so la 0 de chuyen sang anh xam
# Ngưỡng T = 120
ret,th1 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY)

cv2.imshow('Image', img)
cv2.imshow('BasicThreshold', th1)
cv2.waitKey()
cv2.destroyAllWindows()
```

3.4 Ảnh nhị phân và nhị phân hóa với ngưỡng động

Nhị phân hóa hóa với ngưỡng T



$T = 100$



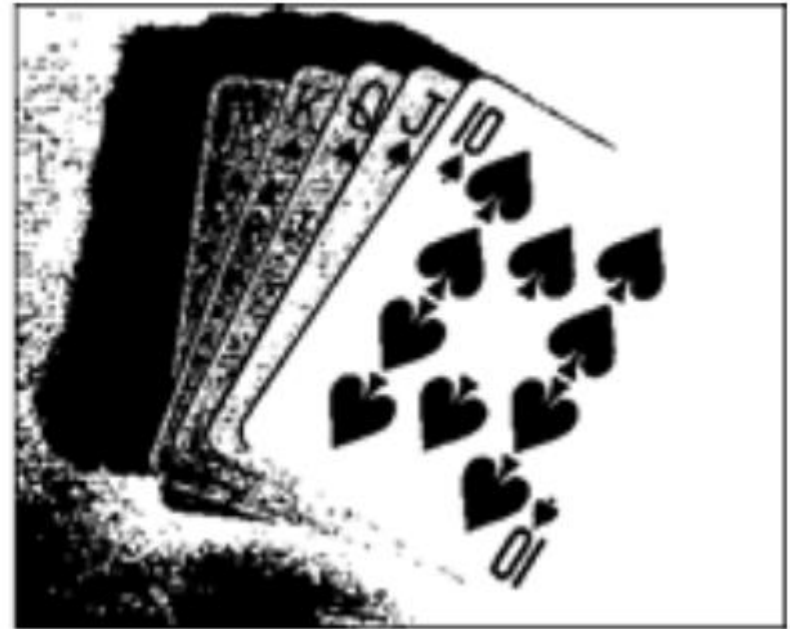
$T = 120$



$T = 140$

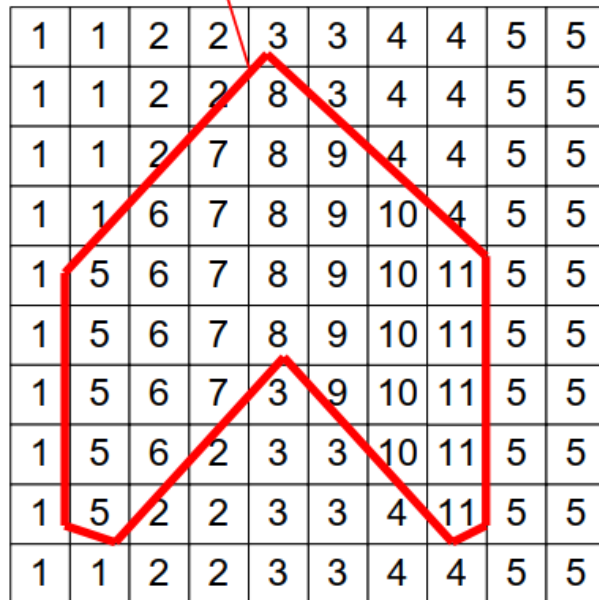
3.4 Ảnh nhị phân và nhị phân hóa với ngưỡng động

Nhị phân hóa với ngưỡng động



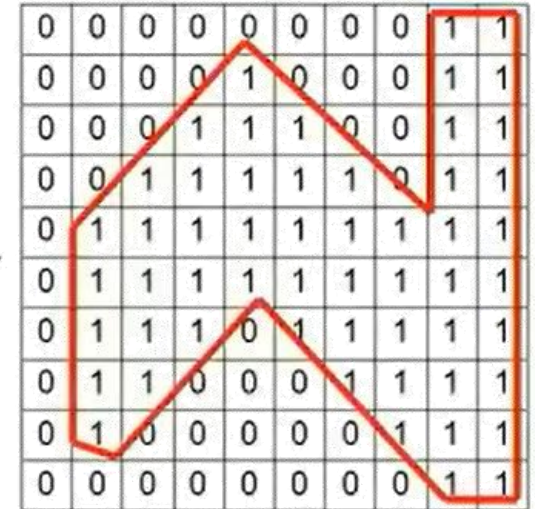
3.4 Ảnh nhị phân và nhị phân hóa với ngưỡng động

Nhị phân hóa với ngưỡng động true object boundary



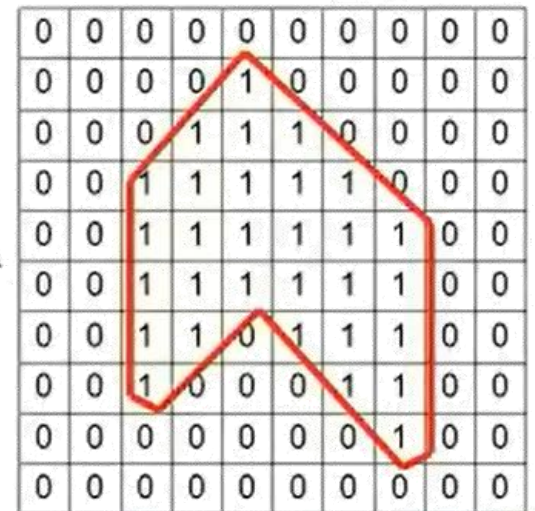
1	1	2	2	3	3	4	4	5	5
1	1	2	2	8	3	4	4	5	5
1	1	2	7	8	9	4	4	5	5
1	1	6	7	8	9	10	4	5	5
1	5	6	7	8	9	10	11	5	5
1	5	6	7	8	9	10	11	5	5
1	5	6	7	3	9	10	11	5	5
1	5	6	2	3	3	10	11	5	5
1	5	2	2	3	3	4	11	5	5
1	1	2	2	3	3	4	4	5	5

Thresholding
 $T = 4.5$



0	0	0	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	1
0	0	0	1	1	1	1	0	1	1
0	0	1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1
0	1	1	1	0	1	1	1	1	1
0	1	1	0	0	0	1	1	1	1
0	1	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	1	1

Thresholding
 $T = 5.5$



0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0
0	0	1	1	1	1	1	0	0	0
0	0	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	0	0
0	0	1	1	0	1	1	1	0	0
0	0	1	0	0	0	1	1	0	0
0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0

3.4 Ảnh nhị phân và nhị phân hóa với ngưỡng động

Nhị phân hóa với ngưỡng động

- Có nhiều phương pháp khác nhau để thực hiện việc này, tuy nhiên chúng đều dựa trên ý tưởng chính là chia ảnh ra thành những vùng nhỏ, với mỗi vùng áp dụng việc nhị phân cho vùng đó với những ngưỡng nhị phân khác nhau.
- Các ngưỡng nhị phân ở các vùng được tính toán dựa trên độ lớn mức xám của chính các pixel trên vùng đó.
- Giả sử ta tính toán ngưỡng cho một vùng nào đó dựa trên độ trung bình của các pixel trong vùng đó (ta có thể xem một vùng là một cửa sổ).

3.4 Ảnh nhị phân và nhị phân hóa với ngưỡng động

Nhị phân hóa với ngưỡng động

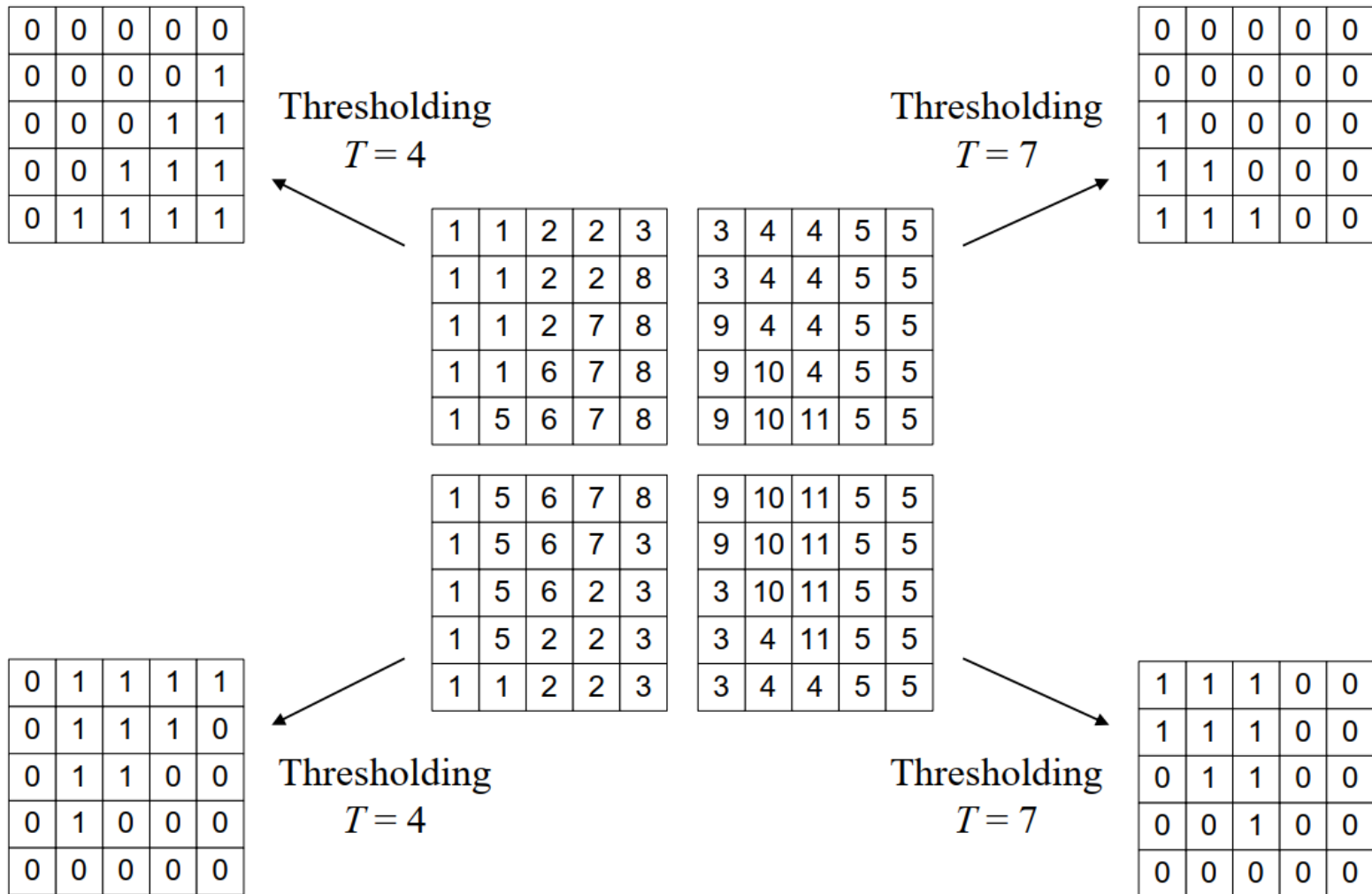
1	1	2	2	3	3	4	4	5	5
1	1	2	2	8	3	4	4	5	5
1	1	2	7	8	9	4	4	5	5
1	1	6	7	8	9	10	4	5	5
1	5	6	7	8	9	10	11	5	5
1	5	6	7	8	9	10	11	5	5
1	5	6	7	3	9	10	11	5	5
1	5	6	2	3	3	10	11	5	5
1	5	2	2	3	3	4	11	5	5
1	1	2	2	3	3	4	4	5	5

Split

1	1	2	2	3	3	4	4	5	5
1	1	2	2	8	3	4	4	5	5
1	1	2	7	8	9	4	4	5	5
1	1	6	7	8	9	10	4	5	5
1	5	6	7	8	9	10	11	5	5
1	5	6	7	8	9	10	11	5	5
1	5	6	7	3	9	10	11	5	5
1	5	6	2	3	3	10	11	5	5
1	5	2	2	3	3	4	11	5	5
1	1	2	2	3	3	4	4	5	5

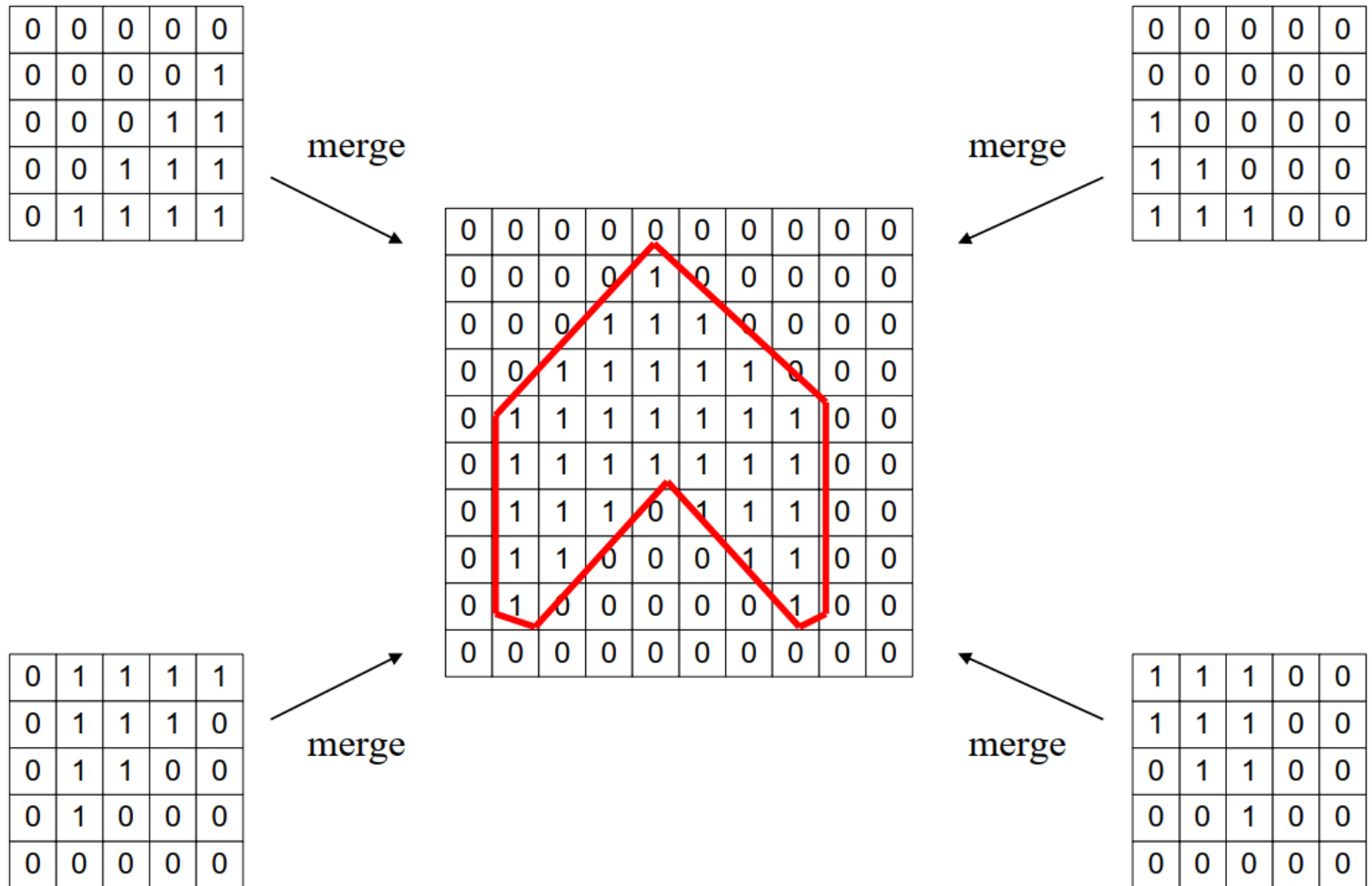
3.4 Ảnh nhị phân và nhị phân hóa với ngưỡng động

Nhị phân hóa với ngưỡng động



3.4 Ảnh nhị phân và nhị phân hóa với ngưỡng động

Nhị phân hóa với ngưỡng động



3.4 Ảnh nhị phân và nhị phân hóa với ngưỡng động

Có 2 phương pháp nhị phân hóa với ngưỡng động được sử dụng trong OpenCV:

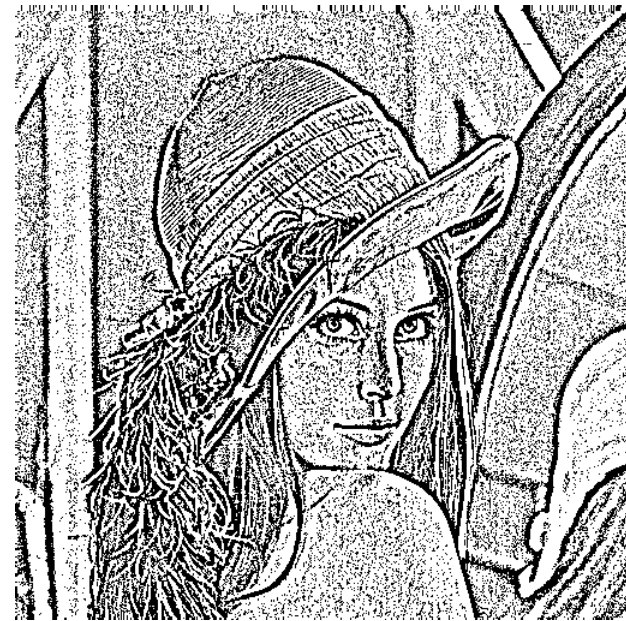
- Adaptive Mean Thresholding
- Adaptive Gaussian Thresholding



Global Thresholding



Mean Thresholding



Gaussian Thresholding

3.4 Ảnh nhị phân và nhị phân hóa với ngưỡng động

Nhị phân hóa với ngưỡng động trong OpenCV

```
import cv2

img = cv2.imread('Lena.png',0) # tham so la 0 de chuyen sang anh xam

ret,th1 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY)
th2 = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
\cv2.THRESH_BINARY, 11, 2)
th3 = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
\cv2.THRESH_BINARY, 11, 2)

cv2.imshow('Image', img)
cv2.imshow('BasicThreshold', th1)
cv2.imshow('MeanThreshold', th2)
cv2.imshow('GausstianThreshold', th3)

cv2.waitKey()
cv2.destroyAllWindows()
```


3.4 Ảnh nhị phân và nhị phân hóa với ngưỡng động

Nhị phân hóa bằng thuật toán Otsu

Otsu là tên một nhà nghiên cứu người Nhật đã nghĩ ra ý tưởng cho việc tính ngưỡng T một cách tự động (adaptive) dựa vào giá trị điểm ảnh của ảnh đầu vào nhằm thay thế cho việc sử dụng ngưỡng cố định. Các bước xử lý như sau:

- Chọn một giá trị khởi tạo cho T (nên chọn giá trị mang tính công thức, ví dụ $T = (\min + \max) / 2$, $T =$ giá trị trung bình,...)
- Phân hoạch ảnh sử dụng T . kết quả của bước này sẽ tạo ra 2 nhóm điểm ảnh: G_1 chứa tất cả các điểm ảnh với giá trị $> T$ và G_2 chứa các điểm ảnh với giá trị $\leq T$.
- Tính trung bình m_1 và m_2 của các điểm ảnh thuộc G_1 và G_2 .
- Tính lại T dựa vào m_1 và m_2 : $T = (m_1 + m_2) / 2$
- Lặp lại bước 2 đến 4 cho tới khi nào giá trị chênh lệch giữa T cũ và T mới là không đáng kể.

3.4 Ảnh nhị phân và nhị phân hóa với ngưỡng động

Nhị phân hóa bằng thuật toán Otsu

- Ví dụ: Cho ảnh xám có ma trận giá trị điểm ảnh như hình bên dưới. Tính bằng tay threshold bằng phương pháp Otsu với T ban đầu là 40.

51	52	53	54
55	56	57	58
59	60	61	91
92	93	94	95
96	97	98	99
100	101	102	103

3.4 Ảnh nhị phân và nhị phân hóa với ngưỡng động

Nhị phân hóa bằng thuật toán Otsu

$T_0 = 40$:

51	52	53	54
55	56	57	58
59	60	61	91
92	93	94	95
96	97	98	99
100	101	102	103

$$m_1 = 0/0 ; m_2 = 1877/24 = 78$$

$T_1 = 78 ; \Delta T = 78$:

$T_1 = 78 ; \Delta T = 78$:

51	52	53	54
55	56	57	58
59	60	61	91
92	93	94	95
96	97	98	99
100	101	102	103

$$m_1 = 616/11 = 56 ; m_2 = 1261/13 = 97$$

$T_2 = (56 + 97)/2 = 76 ; \Delta T = 2$:

3.4 Ảnh nhị phân và nhị phân hóa với ngưỡng động

Nhị phân hóa bằng thuật toán Otsu

$$T_2 = (56 + 97)/2 = 76 ; \Delta T = 2:$$

51	52	53	54
55	56	57	58
59	60	61	91
92	93	94	95
96	97	98	99
100	101	102	103

$$m_1 = 616/11 = 56 ; m_2 = 1261/13 = 97$$

$$T_2 = (56 + 97)/2 = 76 ; \Delta T = 0$$

Vậy ngưỡng $T = 76$

0	0	0	0
0	0	0	0
0	0	0	255
255	255	255	255
255	255	255	255
255	255	255	255

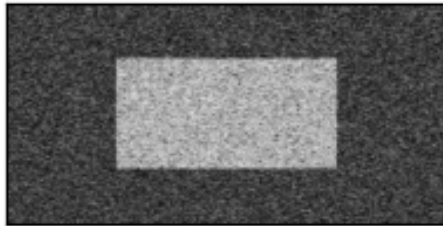
3.4 Ảnh nhị phân và nhị phân hóa với ngưỡng động

```
img = cv2.imread('OtsuH.png', 0)
ret1, th1 = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
ret2, th2 = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
blur = cv2.GaussianBlur(img, (5, 5), 0)
ret3, th3 = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
images = [img, 0, th1,
          img, 0, th2,
          blur, 0, th3]
titles = ['Original Noisy Image', 'Histogram', 'Global Thresholding (T=127)',
          'Original Noisy Image', 'Histogram', "Otsu's Thresholding",
          'Gaussian filtered Image', 'Histogram', "Otsu's Thresholding"]
for i in range(3):
    plt.subplot(3, 3, i * 3 + 1), plt.imshow(images[i * 3], 'gray'),
    plt.title(titles[i * 3]), plt.xticks([]), plt.yticks([])
    plt.subplot(3, 3, i * 3 + 2), plt.hist(images[i * 3].ravel(), 256)
    plt.title(titles[i * 3 + 1]), plt.xticks([]), plt.yticks([])
    plt.subplot(3, 3, i * 3 + 3), plt.imshow(images[i * 3 + 2], 'gray')
    plt.title(titles[i * 3 + 2]), plt.xticks([]), plt.yticks([])
plt.show()
```

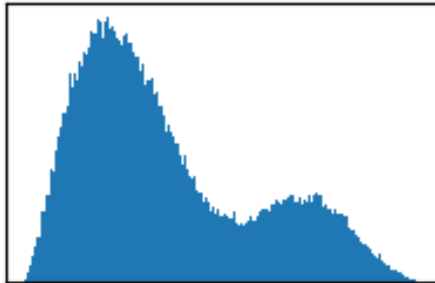
3.4 Ảnh nhị phân và nhị phân hóa với ngưỡng động

Nhị phân hóa bằng thuật toán Otsu

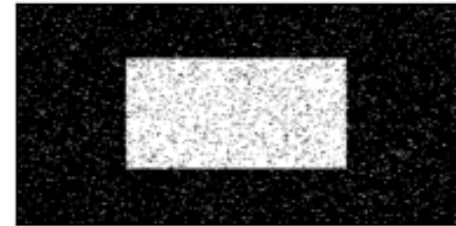
Original Noisy Image



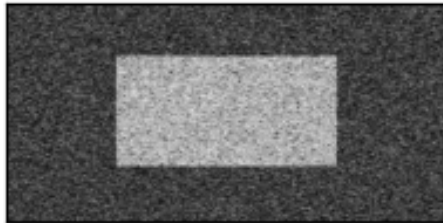
Histogram



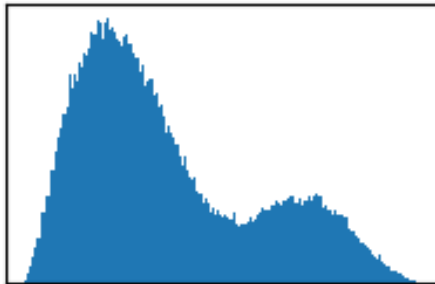
Global Thresholding ($T=127$)



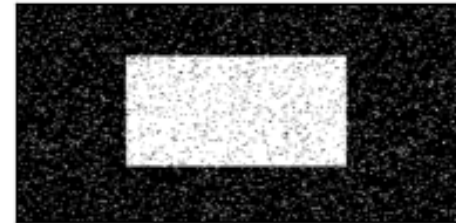
Original Noisy Image



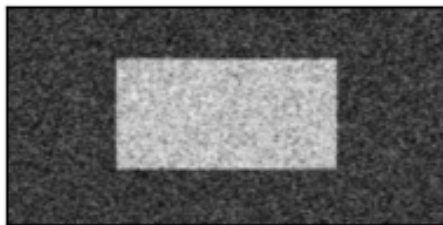
Histogram



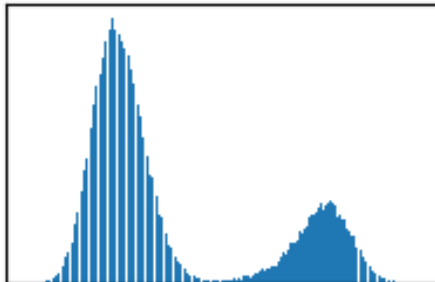
Otsu's Thresholding



Gaussian filtered Image



Histogram



Otsu's Thresholding



3.5 Lọc ảnh

- Lọc ảnh (làm mịn ảnh, làm mượt ảnh) là một bước rất quan trọng trong xử lý ảnh. Lọc ảnh thực tế có rất nhiều tác dụng như loại bỏ nhiễu, tìm biên đối tượng.
- Nguyên tắc chung của các phương pháp lọc là cho ma trận ảnh nhân với một ma trận lọc (Kernel). **Ma trận lọc lọc (Kernel)** còn có thể được gọi là cửa sổ chập (trong phép nhân chập), cửa sổ lọc, mặt nạ,...

- Công thức:
$$I_{dst} = M * I_{src}$$

Trong đó:

- I_{src} là ảnh gốc
- I_{dst} là ảnh sau khi thực hiện phép lọc ảnh
- M là ma trận lọc (Kernel)

3.5 Lọc ảnh

Đặc điểm ma trận lọc (Kernel)

- Với mỗi phép lọc ta có những ma trận lọc M khác nhau, không có quy định cụ thể nào cho việc xác định M .
- Kích thước của ma trận thường là một số lẻ chẳng hạn 3×3 , 5×5 ... Khi đó, tâm của ma trận sẽ nằm ở giao của hai đường chéo và là điểm áp đặt lên ảnh mà ta cần tính nhân chập.
- Tổng các phần tử trong ma trận thông thường bằng 1. Nếu tổng này lớn hơn 1, ảnh qua phép lọc sẽ có độ sáng lớn hơn ảnh ban đầu và ngược lại.

3.5 Lọc ảnh

Một số bộ lọc làm mịn ảnh:

- Lọc trung bình (Average Filter)
- Lọc Gauss (Gaussian Filter)
- Lọc trung vị (Median Filter)
- Lọc hai chiều (Bilateral Filter)
- ...

3.5 Lọc ảnh

Lọc trung bình (Average Filter)

- Lọc trung bình được xây dựng dựa trên ý tưởng tính giá trị một điểm ảnh bằng trung bình cộng các điểm ảnh xung quanh nó.
- Đây là bộ lọc đơn giản nhất.
- Cách lọc này thường được áp dụng cho làm trơn ảnh vẫn muốn giữ lại biên không bị mờ.
- Ma trận lọc của lọc trung bình có dạng:

$$K = \frac{1}{K_width \cdot K_height} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \\ \cdot & \cdot & \cdot & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}$$

- Ví dụ ma trận lọc trung bình: $K = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ $K = \frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$

3.5 Lọc ảnh

Lọc trung bình (Average Filter)

```
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('Lena_Noise.jpg')
average = cv2.blur(img, (5,5))

img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
average_rgb = cv2.cvtColor(average, cv2.COLOR_BGR2RGB)

plt.subplot(121),plt.imshow(img_rgb),plt.title('Lena_Noise')
plt.xticks([], plt.yticks([]))
plt.subplot(122),plt.imshow(average_rgb),\
plt.title('Lena_Noise-Blurred')
plt.xticks([], plt.yticks([]))
plt.show()
```

Lena_Noise



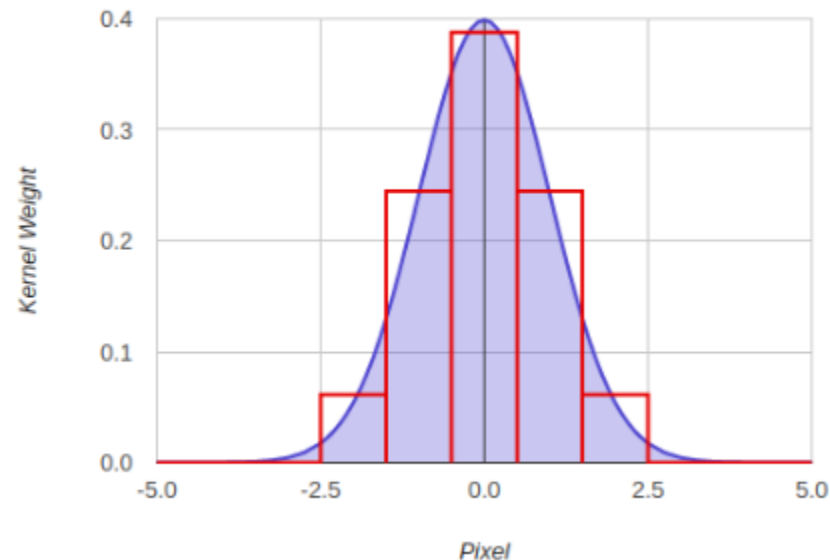
Lena_Noise-Average



3.5 Lọc ảnh

Lọc Gauss (Gaussian Filter)

- Bộ lọc Gauss được cho là bộ lọc hữu ích nhất, được thực hiện bằng cách nhân chụp ảnh đầu vào với một ma trận lọc Gauss sau đó cộng chúng lại để tạo thành ảnh đầu ra.
- Ý tưởng chung là giá trị mỗi điểm ảnh sẽ phụ thuộc nhiều vào các điểm ảnh ở gần hơn là các điểm ảnh ở xa. Trọng số của sự phụ thuộc được lấy theo hàm Gauss.



3.5 Lọc ảnh

Lọc Gauss (Gaussian Filter)

```
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('Lena_Noise.jpg')
gauss = cv2.GaussianBlur(img, (5, 5), 0)

img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
gauss_rgb = cv2.cvtColor(gauss, cv2.COLOR_BGR2RGB)

plt.subplot(211), plt.imshow(img_rgb), \
plt.title('Lena_Noise')
plt.xticks([], plt.yticks([]))
plt.subplot(212), plt.imshow(gauss_rgb), \
plt.title('Lena_Noise-Gauss')
plt.xticks([], plt.yticks([]))
plt.show()
```

Lena_Noise



Lena_Noise-Gauss



3.5 Lọc ảnh

Lọc trung vị (Median Filter)

- Phép lọc trung vị cũng được thực hiện với các ma trận lọc.
- Tuy nhiên, nó tính trung vị tất cả các giá trị điểm ảnh trong vùng ma trận lọc và sử dụng trung vị này cho giá trị điểm trung tâm.
- Với các bộ lọc ở trên, giá trị điểm trung tâm được tính mới (có thể bằng hoặc khác với giá trị một điểm trong vùng ma trận lọc), còn với phép lọc trung vị, giá trị điểm trung tâm luôn được thay bằng một giá trị điểm ảnh trong bức ảnh đầu vào.
- Phương pháp lọc này có khả năng loại bỏ nhiễu muối tiêu (salt-and-pepper noise) khá tốt.
- Phép lọc trung bình và lọc Gauss là phép lọc tuyến tính, nhưng phép lọc trung vị là một phép lọc phi tuyến.

3.5 Lọc ảnh

Lọc trung vị (Median Filter)

```
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('Rose_SaltAndPepper.jpg')
median = cv2.medianBlur(img, 5)

img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
median_rgb = cv2.cvtColor(median,
\cv2.COLOR_BGR2RGB)

plt.subplot(211),plt.imshow(img_rgb),\
plt.title('Rose_SaltAndPepper')
plt.xticks([], plt.yticks([]))
plt.subplot(212),plt.imshow(median_rgb),\
plt.title('Rose_SaltAndPepper-Blurred')
plt.xticks([], plt.yticks([]))
plt.show()
```

Rose_SaltAndPepper



Rose_SaltAndPepper-Median



3.5 Lọc ảnh

Lọc hai chiều (Bilateral Filter)

- Bộ lọc Bilateral cũng sử dụng một bộ lọc Gauss với khoảng cách đến điểm trung tâm, đảm bảo chỉ có các điểm ở gần tham gia vào giá trị của điểm ảnh trung tâm.
- Tuy nhiên, nó sử dụng thêm một hàm Gauss cho mức xám, đảm bảo chỉ các điểm ảnh có mức xám tương đồng với điểm ảnh trung tâm tham gia vào quá trình làm mịn. Vì thế bộ lọc Bilateral bảo toàn được các đường biên trong ảnh bởi vì điểm ảnh ở biên có sự thay đổi về mức xám rất rõ ràng.
- Tuy vậy, bộ lọc hai chiều cũng có nhược điểm là chậm hơn các bộ lọc khác.

3.5 Lọc ảnh

Lọc hai chiều (Bilateral Filter)

```
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('Lena_Noise.jpg')
bilateral = cv2.bilateralFilter(img, 9, 75, 75)

img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
bilateral_rgb = cv2.cvtColor(bilateral, cv2.COLOR_BGR2RGB)

plt.subplot(211), plt.imshow(img_rgb), \
plt.title('Lena_Noise')
plt.xticks([], plt.yticks([]))
plt.subplot(212), plt.imshow(bilateral_rgb), \
plt.title('Lena_Noise-Bilateral')
plt.xticks([], plt.yticks([]))
plt.show()
```

Lena_Noise



Lena_Noise-Bilateral



TỔNG HỢP MỘT SỐ HÀM QUAN TRỌNG

`cv.medianBlur(src, ksize)`

Parameters

src input 1-, 3-, or 4-channel image .

ksize aperture linear size; it must be odd and greater than 1, for example: 3, 5, 7 ...

`cv.GaussianBlur(src, ksize, sigmaX, sigmaY)`

Parameters

src input image; the image can have any number of channels .

ksize Gaussian kernel size. `ksize.width` and `ksize.height` can differ but they both must be positive and odd.

sigmaX Gaussian kernel standard deviation in X direction.

sigmaY Gaussian kernel standard deviation in Y direction.

TỔNG HỢP MỘT SỐ HÀM QUAN TRỌNG

plt.hist(img.ravel(), histSize , ranges)

Parameters

- img** Source arrays.
- histSize** Array of histogram sizes in each dimension.
- ranges** Array of the dims arrays of the histogram bin boundaries in each dimension.

cv.threshold(src, thresh, maxval, type[dst]) => **retval**, **dst**

Parameters

- src** input array (multiple-channel, 8-bit or 32-bit floating point).
- dst** output array of the same size and type and the same number of channels as src.
- thresh** threshold value.
- maxval** maximum value to use with the **THRESH_BINARY** and **THRESH_BINARY_INV**
- type** thresholding type (see **ThresholdTypes**).

TỔNG HỢP MỘT SỐ HÀM QUAN TRỌNG

cv.adaptiveThreshold(src, maxValue, adaptiveMethod, thresholdType, blockSize, C) => dst

Parameters

src	Source 8-bit single-channel image.
dst	Destination image of the same size and the same type as src.
maxValue	Non-zero value assigned to the pixels for which the condition is satisfied
adaptiveMethod	Adaptive thresholding algorithm to use, see AdaptiveThresholdTypes .
thresholdType	Thresholding type that must be either THRESH_BINARY or THRESH_BINARY_INV .
blockSize	Size of a pixel neighborhood that is used to calculate a threshold value for the pixel: 3, 5, 7, and so on.
C	Constant subtracted from the mean or weighted mean (see the details below). Normally, it is positive but may be zero or negative as well.

ADAPTIVE_THRESH_MEAN_C

Python: cv.ADAPTIVE_THRESH_MEAN_C

ADAPTIVE_THRESH_GAUSSIAN_C

Python: cv.ADAPTIVE_THRESH_GAUSSIAN_C

CASE STUDY 3: NHỊ PHÂN HÓA ẢNH

Mô tả: Phân ngưỡng (threshold) ảnh với nhiều phương pháp.

Yêu cầu:

- Phân ngưỡng với T được chọn bằng thử-sai;
- Phân ngưỡng với T được chọn dựa vào biểu đồ Histogram;
- Phân ngưỡng với T được chọn dựa vào thuật toán Ostu;
- Phân ngưỡng với T được chọn dựa vào ROI;
- Phân ngưỡng với T được chọn dựa vào Trackbar window.

Thank you !!!