XỬ LÝ ẢNH

(Image Processing)

BÀI 3: CẢI THIỆN ẢNH

CÁC PHÉP TOÁN TRÊN ĐIỂM ẢNH

(Point Processing)

Nội dung

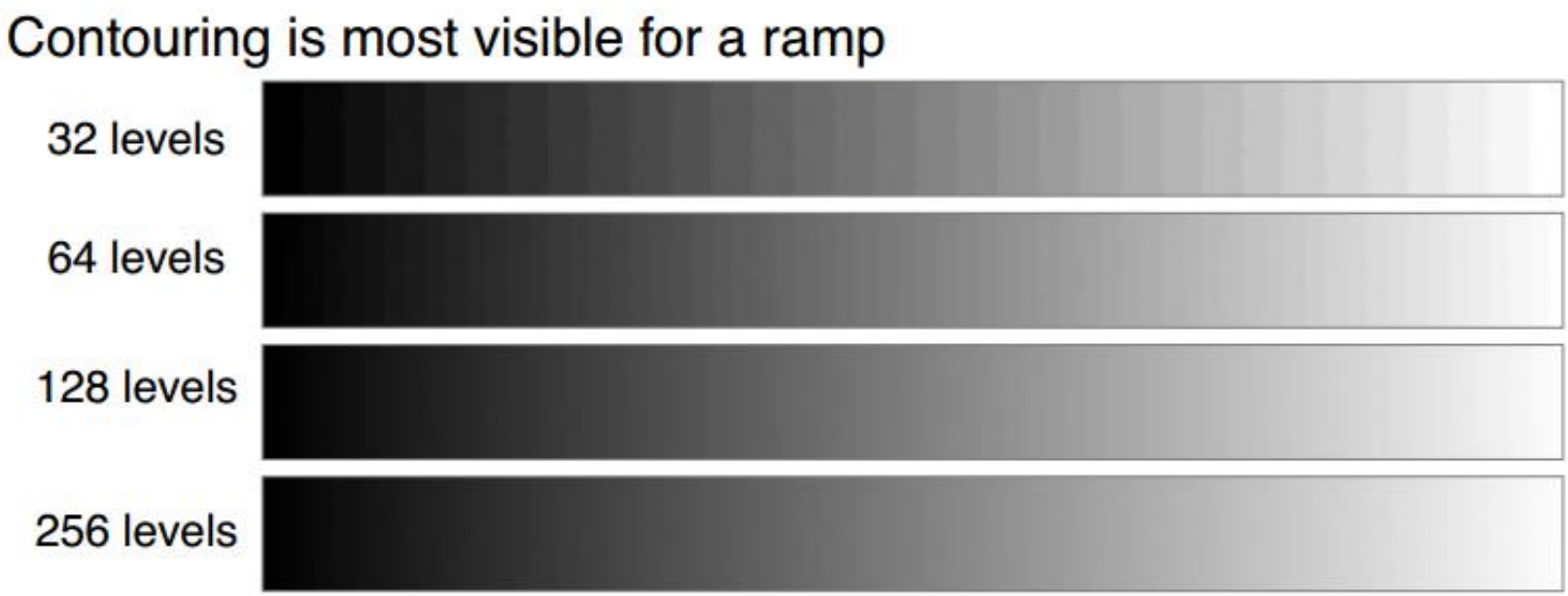
- Cải thiện ảnh là gì?
- Các phương thức biến đổi ảnh
- Các kỹ thuật cải thiện ảnh
- **Các phép toán trên điểm ảnh** ***(Point operations)***
- **Lát cắt mặt phẳng bit** ***(Bit-plane Slicing)***
- **Make borders for images** ***(padding)***

Nhắc lại: Ảnh số

Therefore, the digital image can be simply coded in a matrix form as:

$$\begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & \dots & f(1,N-1) \\ \vdots & \vdots & \ddots & \vdots \\ f(M-1,0) & f(M-1,1) & \dots & f(M-1,N-1) \end{bmatrix}$$

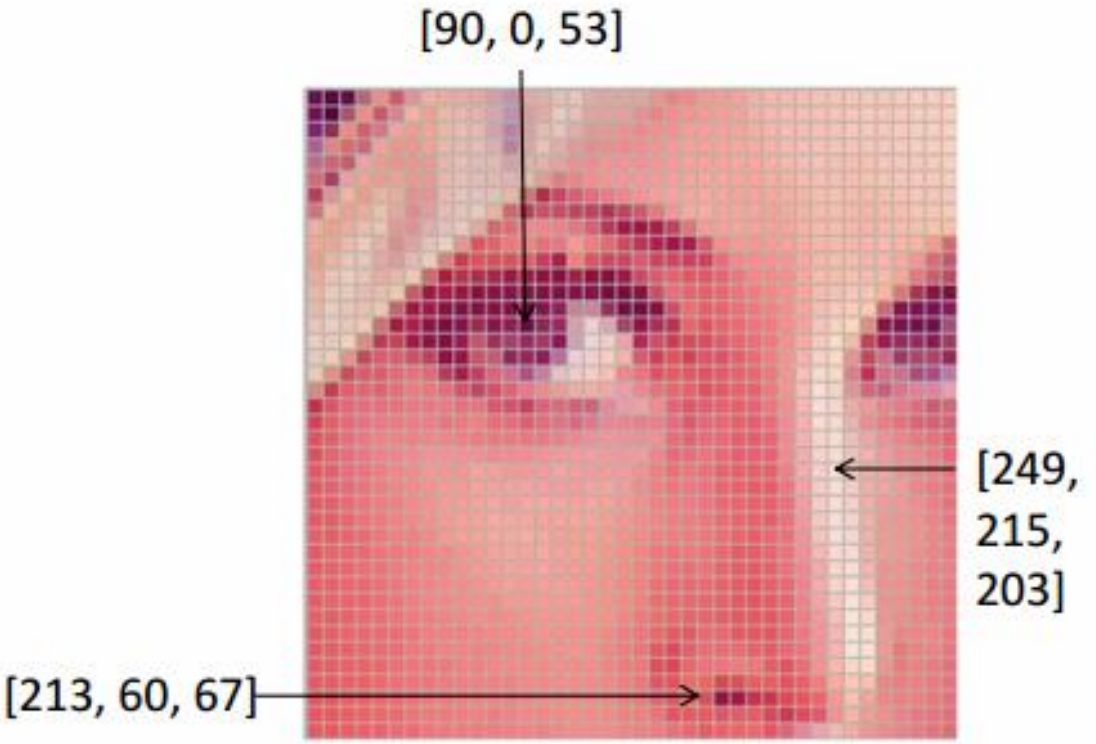
Nhắc lại: Mức độ lượng tử hoá



Ảnh số thường được lượng tử hoá thành 256 mức xám

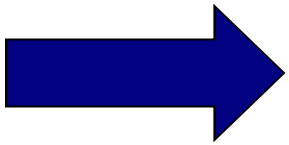
Nhắc lại: Biểu diễn ảnh số

- Ảnh số chứa các điểm ảnh rời rạc
- Giá trị điểm ảnh:
  - "grayscale" (or "intensity"): [0,255]
  - "color"
    - RGB: [R, G, B]
    - Lab: [L, a, b]
    - HSV: [H, S, V]



Cải thiện ảnh là gì?

- Ví dụ:

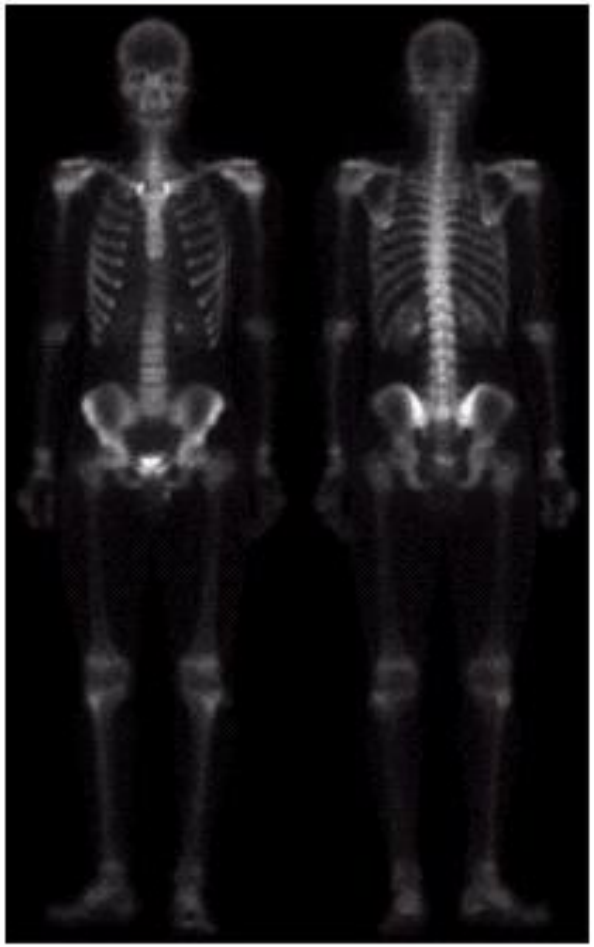


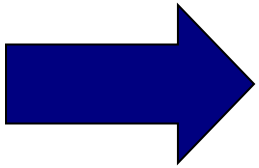
Mức xám không cân bằng

Mức xám cân bằng

Cải thiện ảnh là gì?

- Ví dụ:



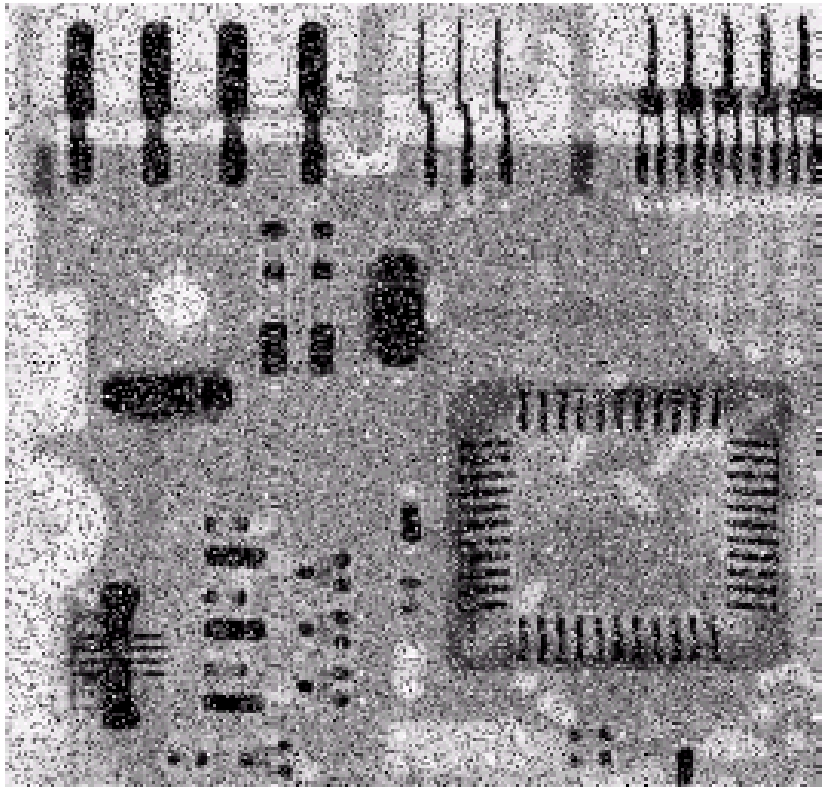


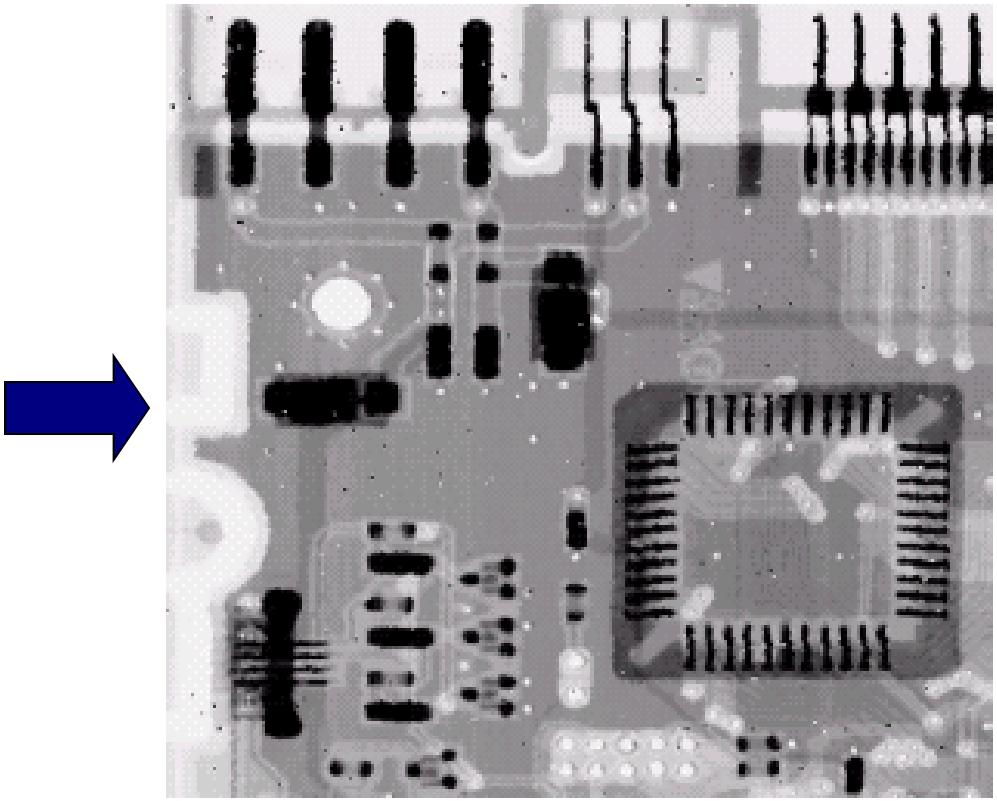
Độ tương phản kém

Độ tương phản tốt hơn

Cải thiện ảnh là gì?

- Ví dụ:





Ảnh nhiễu

Độ mịn/mượt tốt hơn

Cải thiện ảnh là gì?

Ảnh rõ hơn

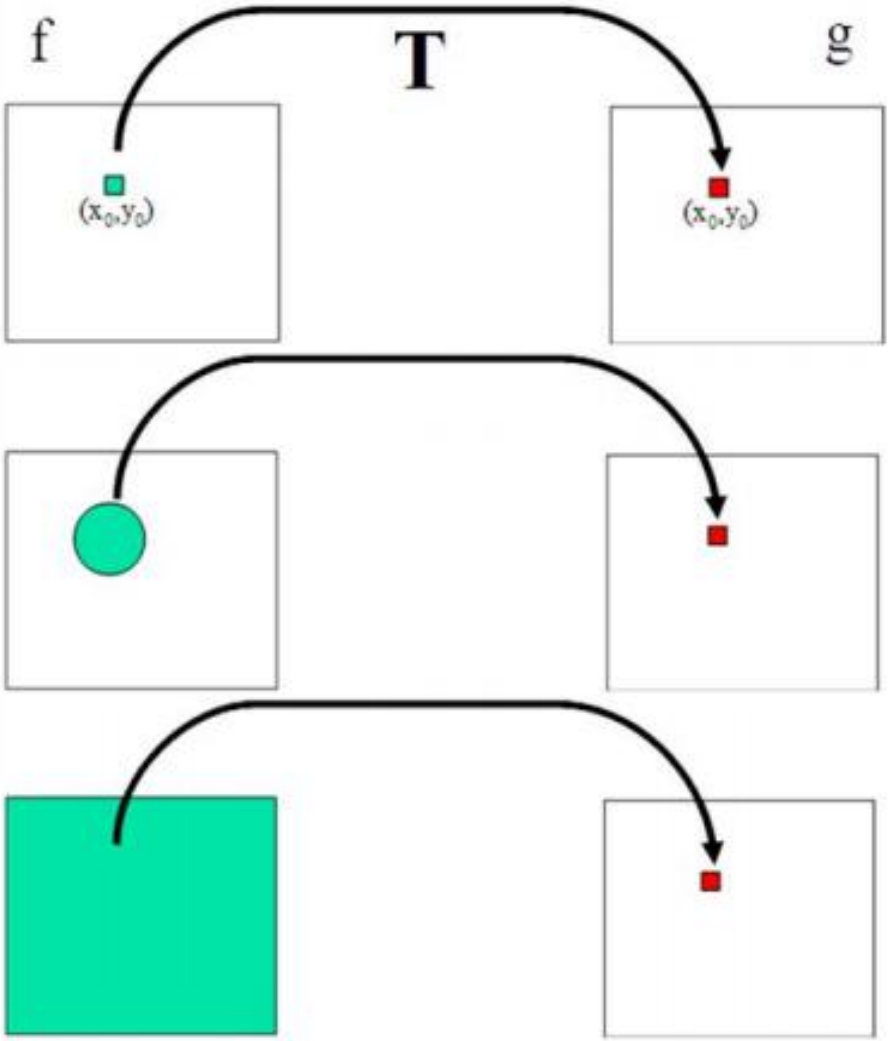
Ảnh giảm độ nhoè

Ảnh nhoè

Cải thiện ảnh là gì?

- Cải thiện ảnh là quá trình làm cho hình ảnh hữu ích hơn
- Là quá quá trình xử lý một hình ảnh để cho kết quả **phù hợp hơn** so với ảnh ban đầu cho một **ứng dụng cụ thể**
- *Lý do:*
  - Làm nổi bật các chi tiết **cần quan tâm** trong ảnh.
  - Loại bỏ nhiễu khỏi hình ảnh.
  - Làm cho hình ảnh trực quan hấp dẫn hơn.

Các phương thức biến đổi ảnh



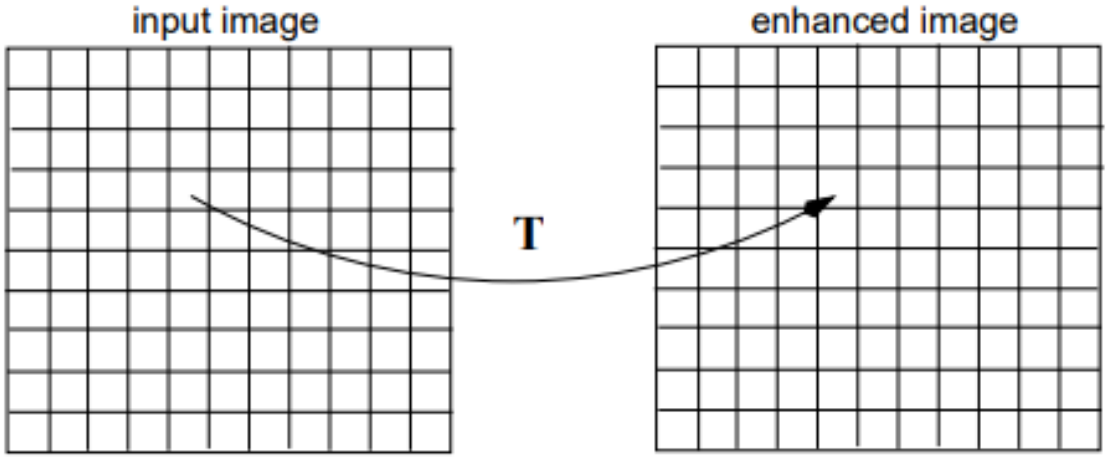
Isolated: $g(x_0,y_0)=T[f(x_0,y_0)]$

Local: $g(x_0,y_0)=T[f(V)]$
V:neighbors of (x_0,y_0)

Global: $g(x_0,y_0)=T[f(x,y)]$
example: FFT

Các phương thức biến đổi ảnh

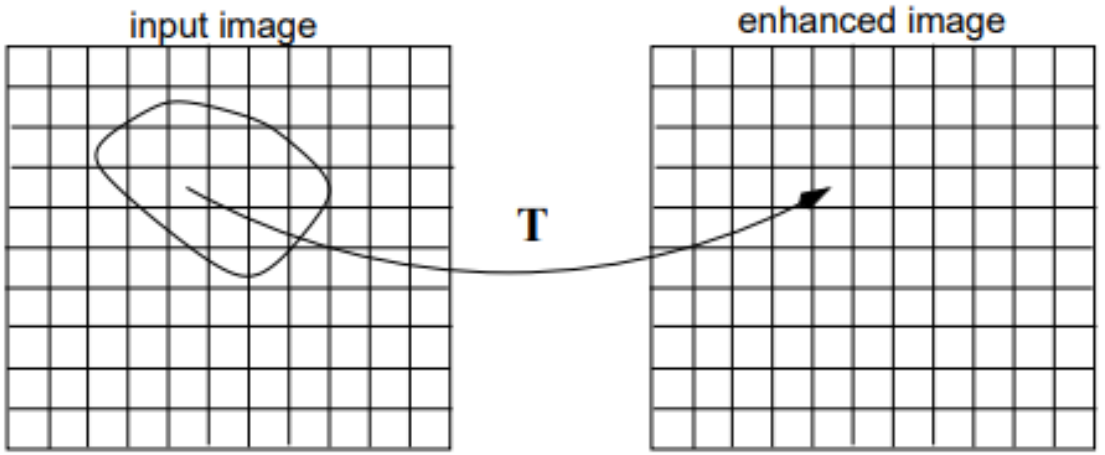
Point Processing Methods



$$g(x,y) = T[f(x,y)]$$

T operates on 1 pixel

Area or Mask Processing Methods



$$g(x,y) = T[f(x,y)]$$

T operates on a neighborhood of pixels

Các kỹ thuật cải thiện ảnh

- Các kỹ thuật cải thiện ảnh chia thành 2 nhóm:
  - Các kỹ thuật theo miền không gian
    - Các phép toán trên điểm ảnh
    - Các bộ lọc
    - Kỹ thuật Histogram
  - Các kỹ thuật theo miền tần số
    - Ảnh có thể xem như tín hiệu 2 chiều
    - Có thể tác động lên tần số để cải thiện chất lượng ảnh
    - Biến đổi **Fourrier**

Point operations

- Biểu diễn thao tác xử lý điểm ảnh:

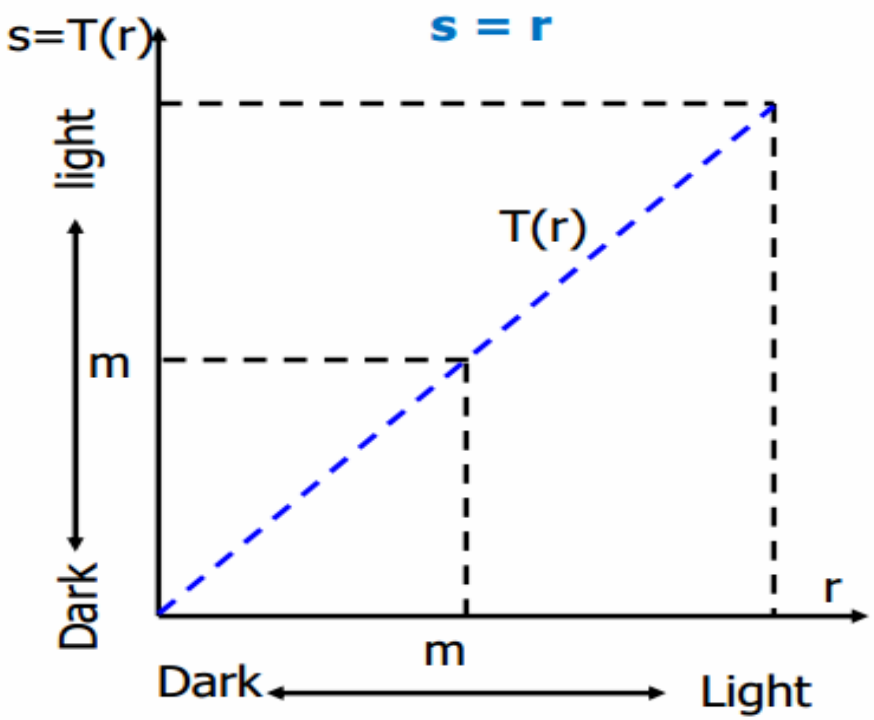
$$s = T(r)$$

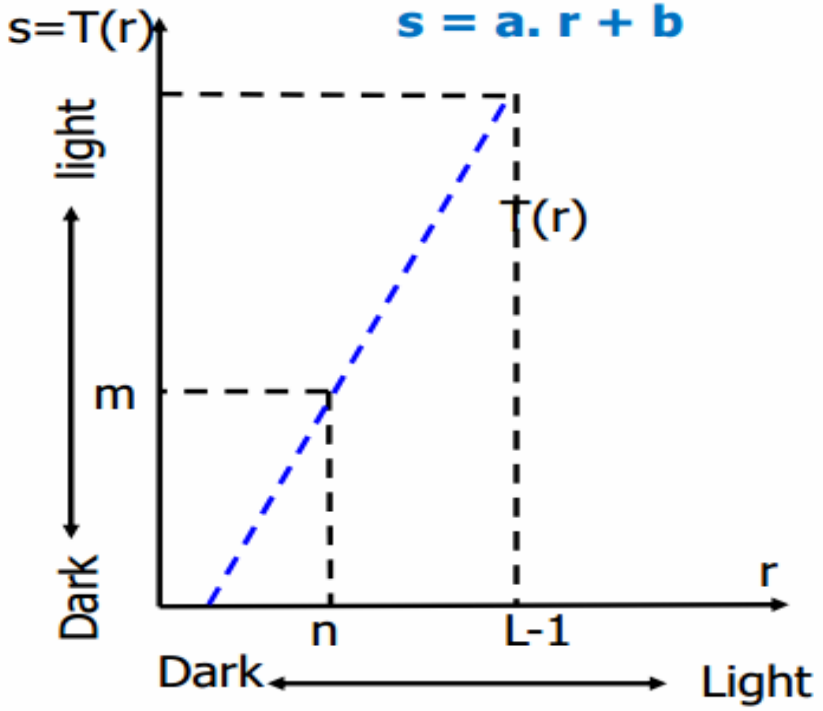
  - Với s là giá trị của điểm ảnh **output**
  - r là giá trị của điểm ảnh **input**
  - T là toán xử lý điểm ảnh
    - Phép đảo ảnh
    - Phép biến đổi Logarit
    - Phép biến đổi Gamma
    - Cắt ngưỡng: chuyển đổi sang ảnh nhị phân

Biến đổi tuyến tính

Addition: $f(u) = \begin{cases} u + c & , \text{if } u + c \leqslant 255 \\ 255 & , \text{else} \end{cases}$

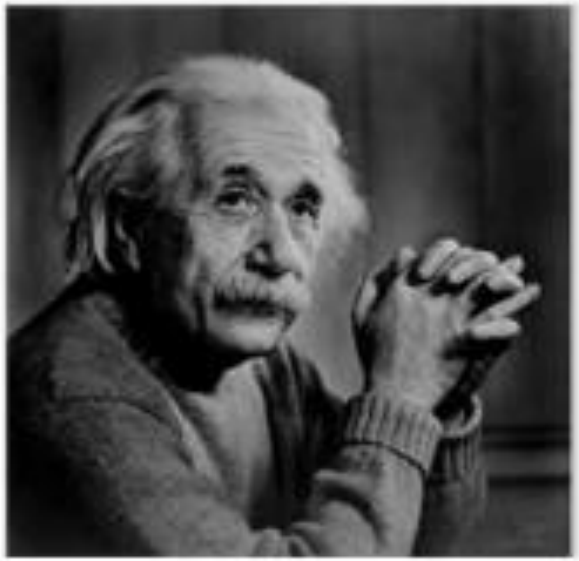
Subtraction: $f(u) = \begin{cases} u - c & , \text{if } u - c \geqslant 0 \\ 0 & , \text{else} \end{cases}$





Example: Tăng/giảm sáng ảnh

$$G(x,y) = F(x,y) + c$$



Ảnh gốc → Ảnh sau biến đổi

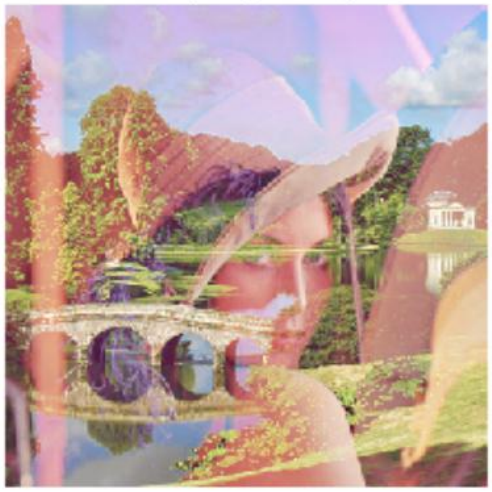
- Với c > 0: ảnh sáng hơn; c < 0: ảnh tối đi

Lena[80%]+Garden[20%] (Blending Image)

$f(img_x, img_y) = (img_x * 0.8) + (img_y * 0.2)$

min(Lena, Garden)

max(Lena, Garden)

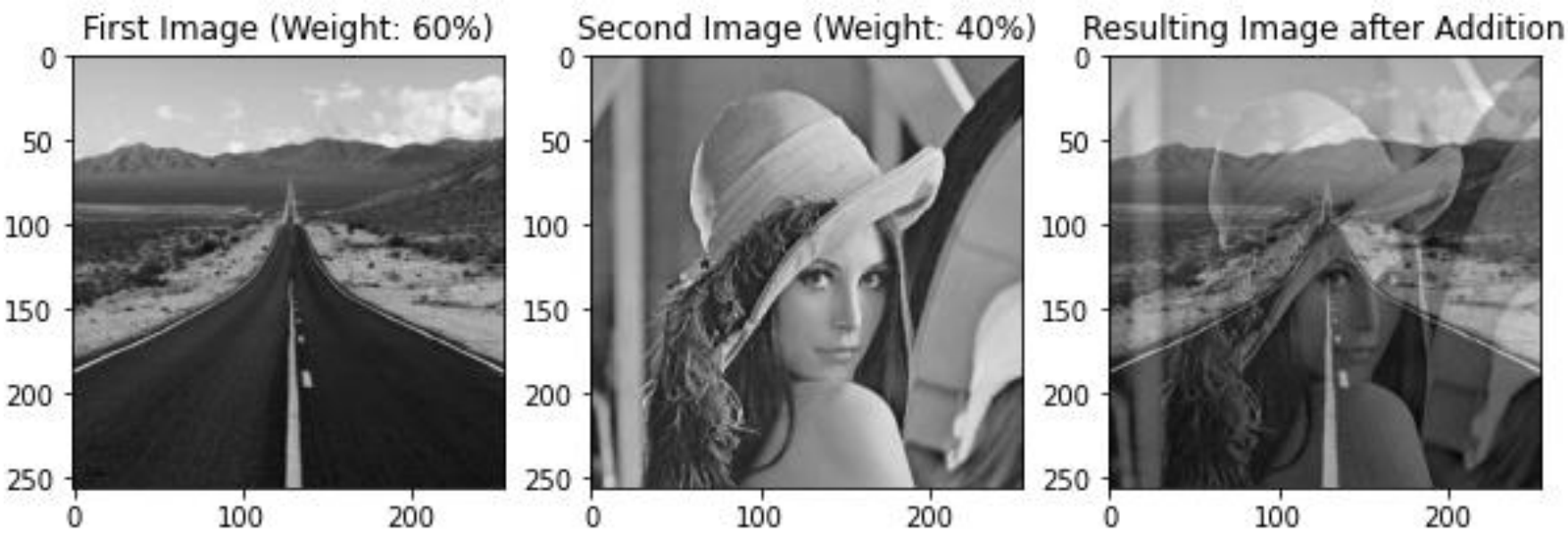


Maximum: $f(u) = max(u_1, u_2)$

Minimum: $f(u) = min(u_1, u_2)$

Example: Blending Image

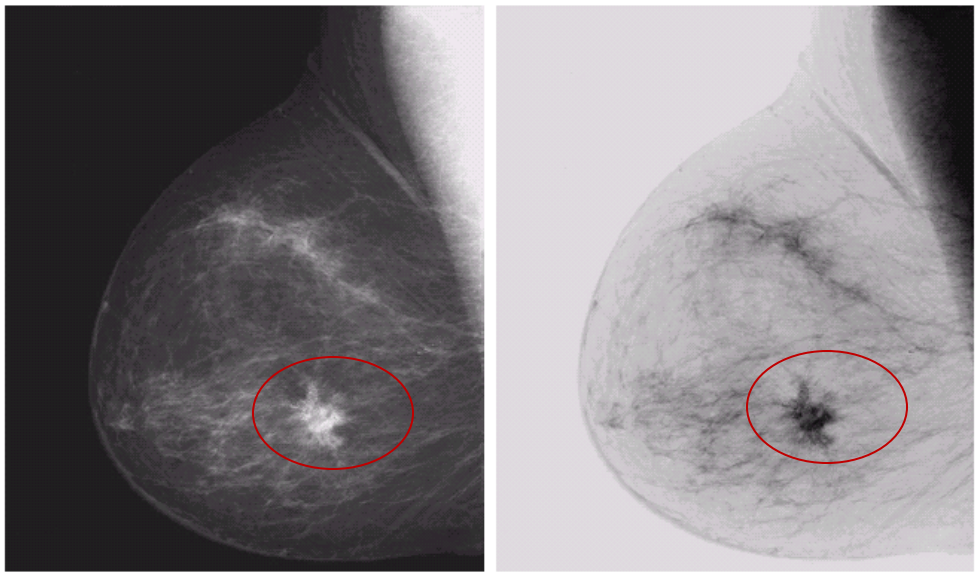
```
res = cv2.addWeighted(image1, 0.6, image2, 0.4, 0)
```



Ảnh âm bản – Negative/Inverse Images

- Là thao tác xử lý trên điểm ảnh có dạng: s = (L-1) - r
- Với s là điểm ảnh output, r là điểm ảnh input, L là mức xám lớn nhất.

  Ví dụ: $f(u) = 255 - u$

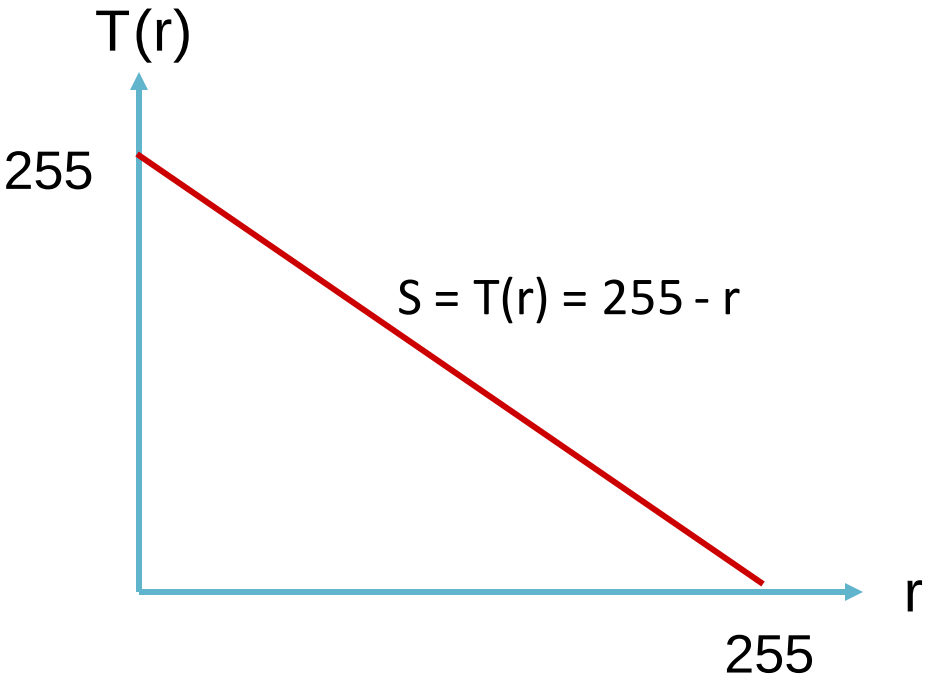


a b

FIGURE 3.4 (a) Original digital mammogram. (b) Negative image obtained using the negative transformation in Eq. (3.2-1). (Courtesy of G.E. Medical Systems.)

- Đảo ảnh hữu ích cho việc cải thiện các chi tiết màu trắng hay màu xám nằm trong vùng tối của ảnh

Negative/ Inverse Images



Code:

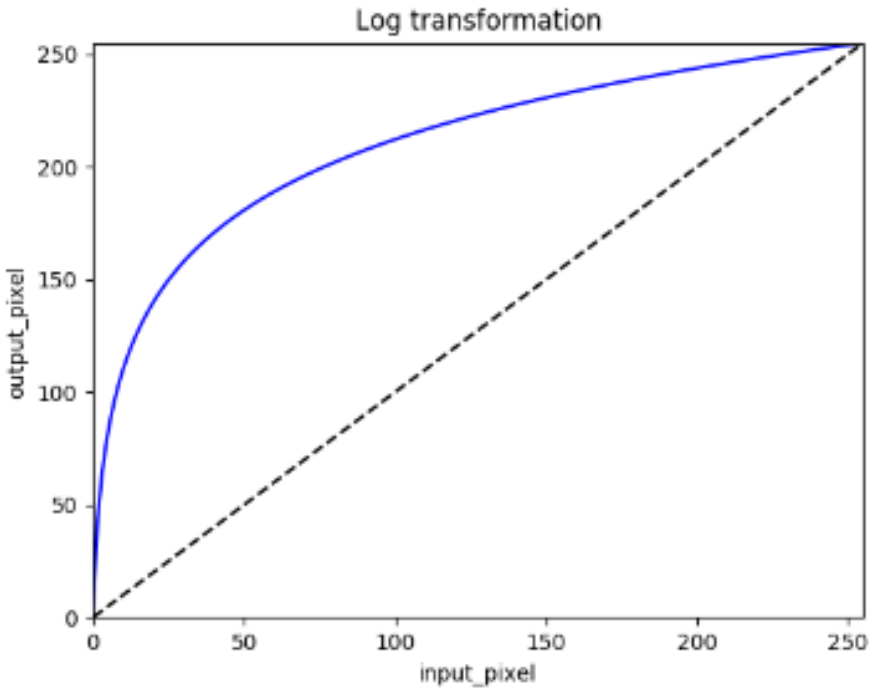
```
def dao_anh(img):
    return 255-img
```

Biến đổi ảnh bằng hàm Logarit

- *Xử lý trên điểm ảnh có dạng:*

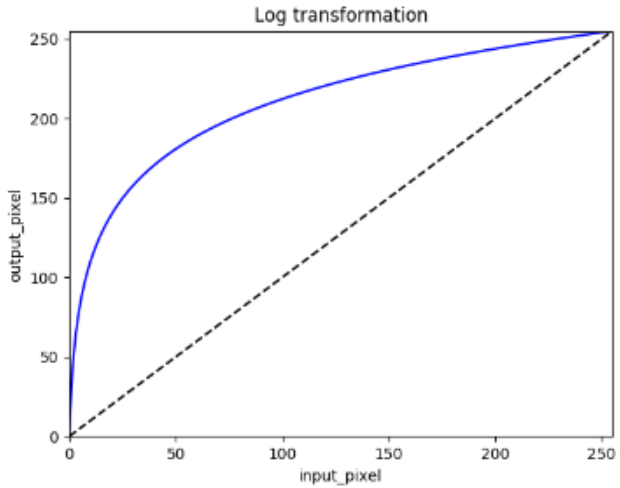
$$s = c \times log(1+r)$$

- Với c là hằng số. r bị giới hạn trong khoảng $[0, 255]$, ta có thể chọn c để giới hạn output s.
- Với $c = \frac{255}{log(256)}$, $s \in [0, 255]$.



Biến đổi ảnh bằng hàm Logarit

- Nhận xét:
  - S > r với mọi r. **Log transformation** dùng làm sáng các ảnh có độ sáng thấp.
  - Với thấp, đồ thị dốc và thoải hơn khi lớn. **Log transformation** ánh xạ vùng *pixel* nhỏ (tối), hẹp thành vùng rộng hơn, lớn (sáng) hơn và ngược lại, vùng *pixel* lớn được nén lại.



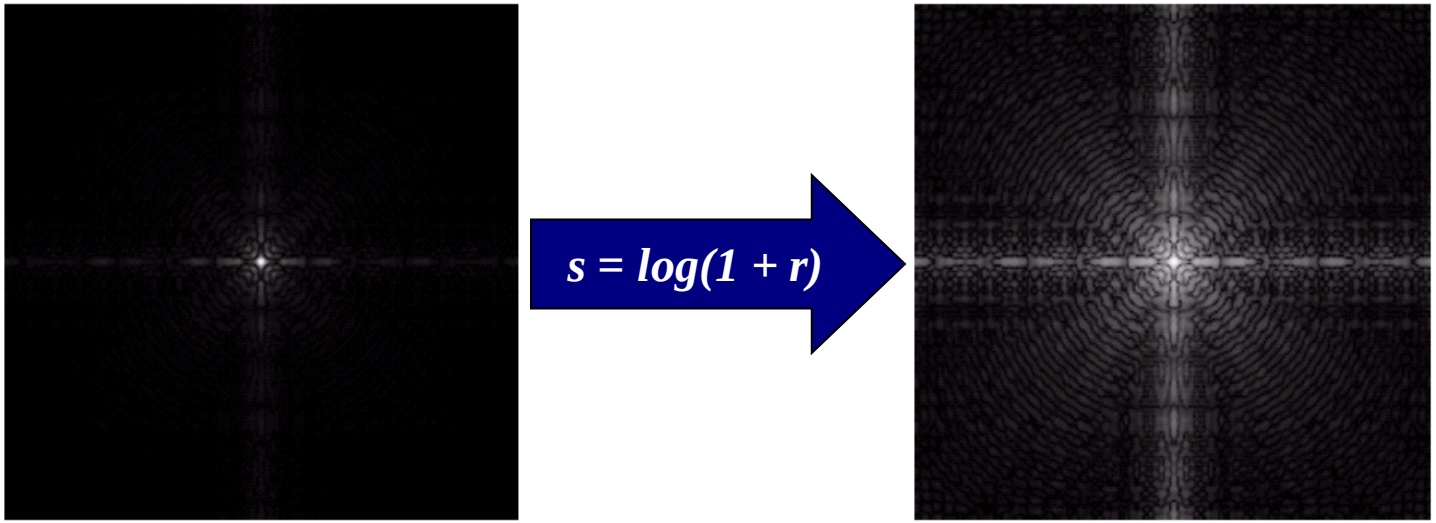

```
def Chuyen_doi_logarit(img, c):
    img = img.astype(float)
    return float(c) * cv.log(1 + img)
```

Biến đổi ảnh bằng hàm Logarit

$c=100$

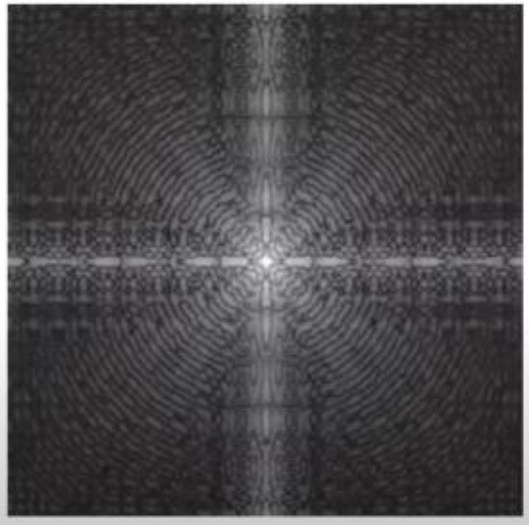
Ví dụ

- Làm sáng hơn các điểm có mức xám thấp trong khi nén các giá trị có mức xám cao



Ví dụ

$$s = 2 \times \log(1 + r)$$

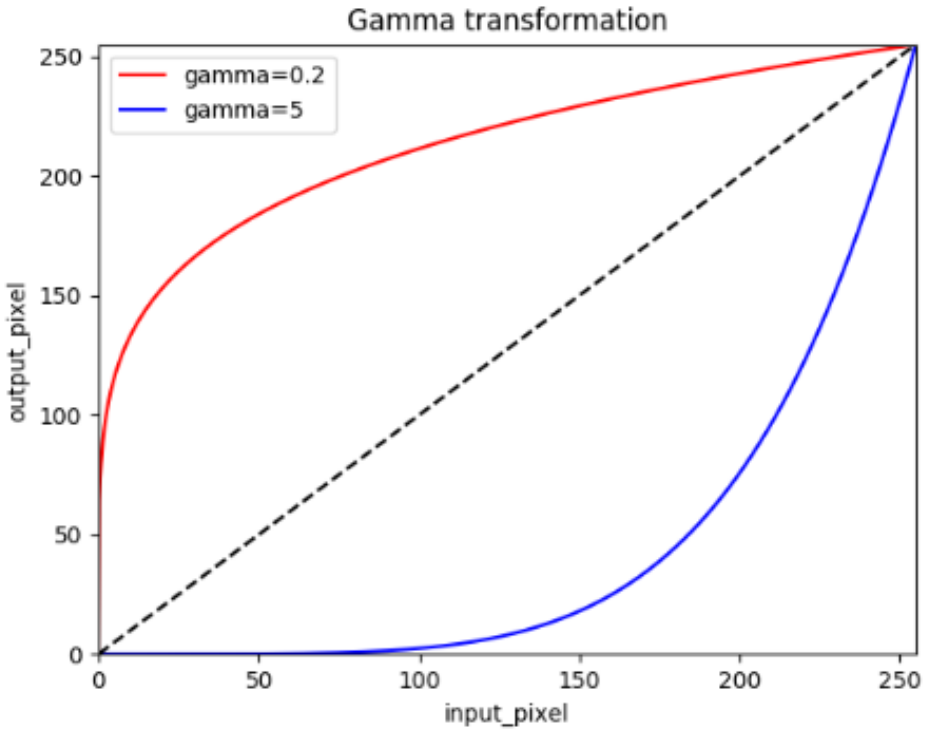


Biến đổi ảnh bằng hàm mũ (Power – law)/ Gamma

- *Là thao tác xử lý trên điểm ảnh có dạng:*

$$s = c \times r^{\gamma}$$

- c là hằng số để giới hạn output s
- Với $c = \frac{1}{255^{\gamma}}$, $s \in [0, 255]$.



Biến đổi ảnh bằng hàm mũ (Power – law)/ Gamma

- Nhận xét:
  - Ta dùng $\gamma > 1$ sẽ thu được ảnh *tối hơn* ảnh gốc và ngược lại với $\gamma < 1$.

Code:

```
def gamma_trans(img,c,gamma):
    return c*(img**gamma)
```

Biến đổi ảnh bằng hàm mũ (Power Law)/ Gamma

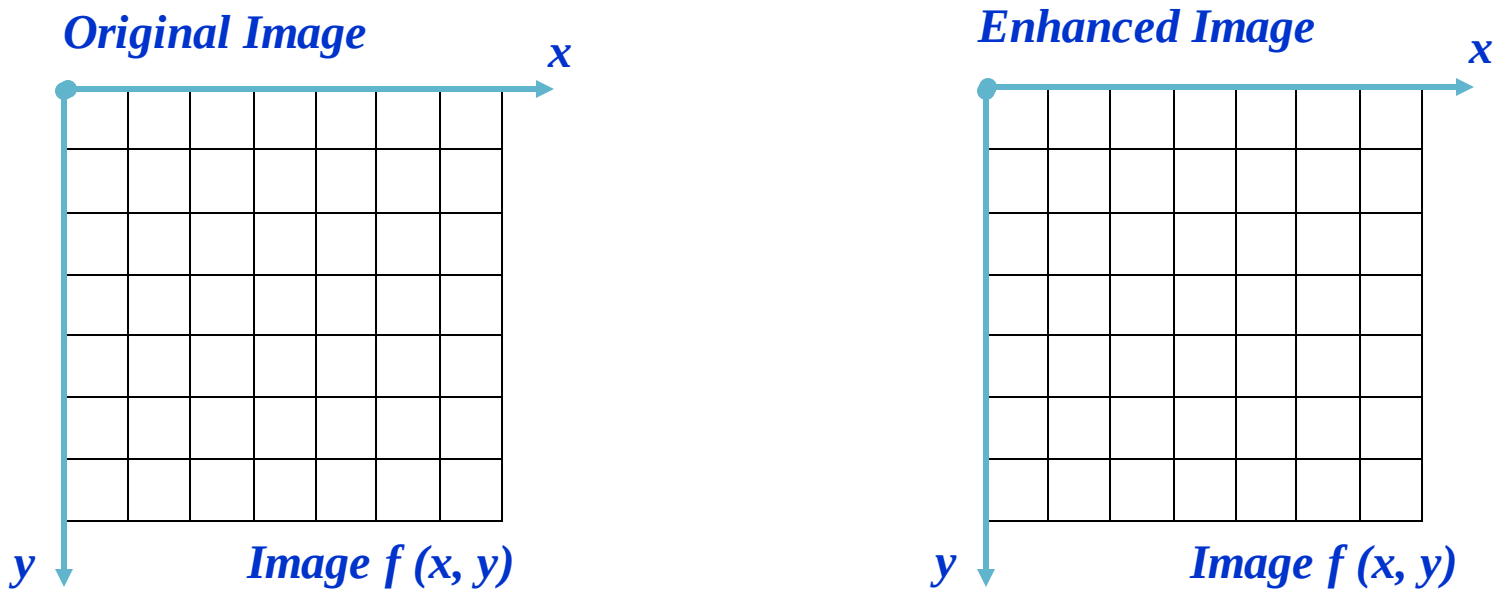
Hoặc: với mọi c > o

```
def Chuyen_Doi_Gamma(img, gamma, c):
    img = img.astype(float)
    return float(c) * pow(img, float(gamma))
```

Original

Gamma transformation(gamma=0.4)

Power Law Transformations



$$s = r^{\gamma}$$

- We usually set C to 1
- Grey levels must be in the range [0.0, 1.0]

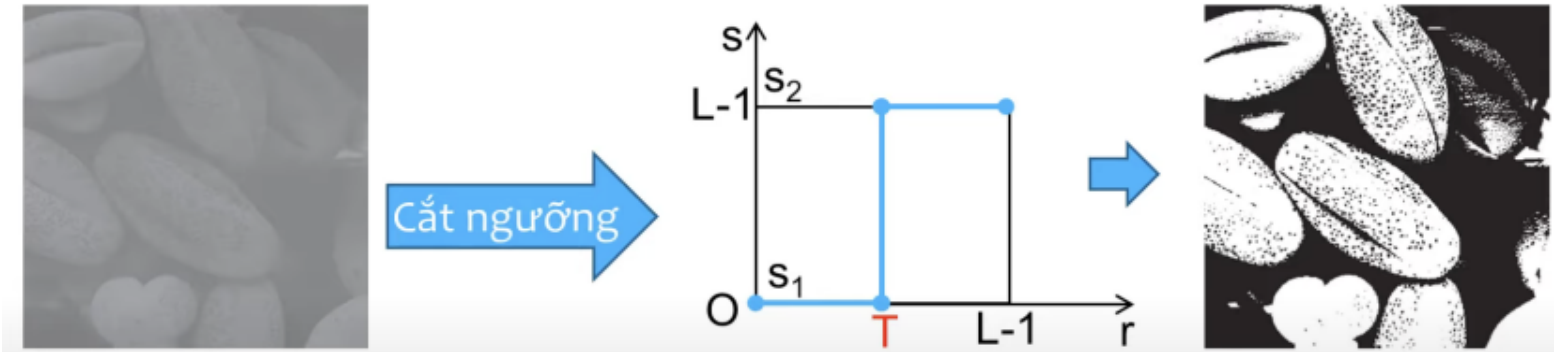
Power Law Example

Power Law Example

Ảnh cộng hưởng từ (MRI) của cột sống người bị gãy.

Cắt ngưỡng

- Chuyển ảnh đa mức xám về 2 mức giá trị xám



Code:

```
def cat_nguong(img, t):
    return img > t
```

Lát cắt mặt phẳng bit

❖ ***Nhắc lại:***

- Chuyển số hệ thập phân sang nhị phân? Ví dụ: $(80)_{10} = (1010000)_2$

  -> biểu diễn 8 bit là: 01010000

- Chuyển số nhị phân sang thập phân:

  $b_7b_6b_5b_4b_3b_2b_1b_0 = b_7x2^7 + b_6x2^6 + \ldots + b_0x2^0$

  Example: 01010000

  $= 0x2^7 + 1x2^6 + 0x2^5 + 1x2^4 + 0x2^3 + 0x2^2 + 0x2^1 + 0x2^0$

  $= 80$

Lát cắt mặt phẳng bit

- Lát cắt mặt phẳng bit (bit plane sciling): Là kỹ thuật chia ảnh đa cấp xám thành nhiều ảnh nhị phân.
  - Bước 1: Chuyển ảnh xám sang ma trận mà mỗi phần tử chứa chuỗi bit.

7	3	6
4	6	3
2	7	2

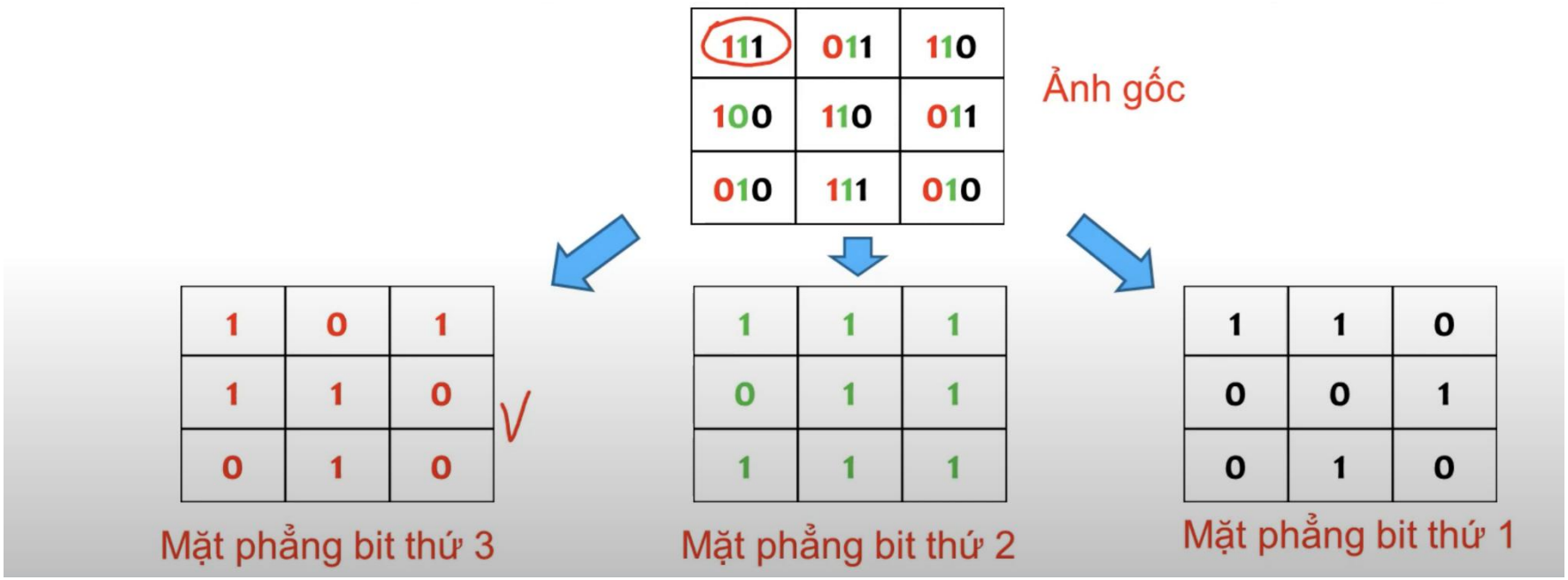
Ảnh đa cấp xám

111	011	110
100	110	011
010	111	010

Ảnh chuỗi các bit (biểu diễn 3 bit)

Lát cắt mặt phẳng bit

- Bước 2: Tách thành các ma trận (mặt phẳng bit), mỗi phần tử chứa 1 bit.



Lát cắt mặt phẳng bit

- Bước 3: Tái tạo ảnh: chuyển ảnh nhị phân thành ảnh đa mức xám.

Mặt phẳng bit thứ 3

1	0	1
1	1	0
0	1	0

x $2^{(3-1)}$

Mặt phẳng bit thứ 2

1	1	1
0	1	1
1	1	1

x $2^{(2-1)}$

Mặt phẳng bit thứ 1

1	1	0
0	0	1
0	1	0

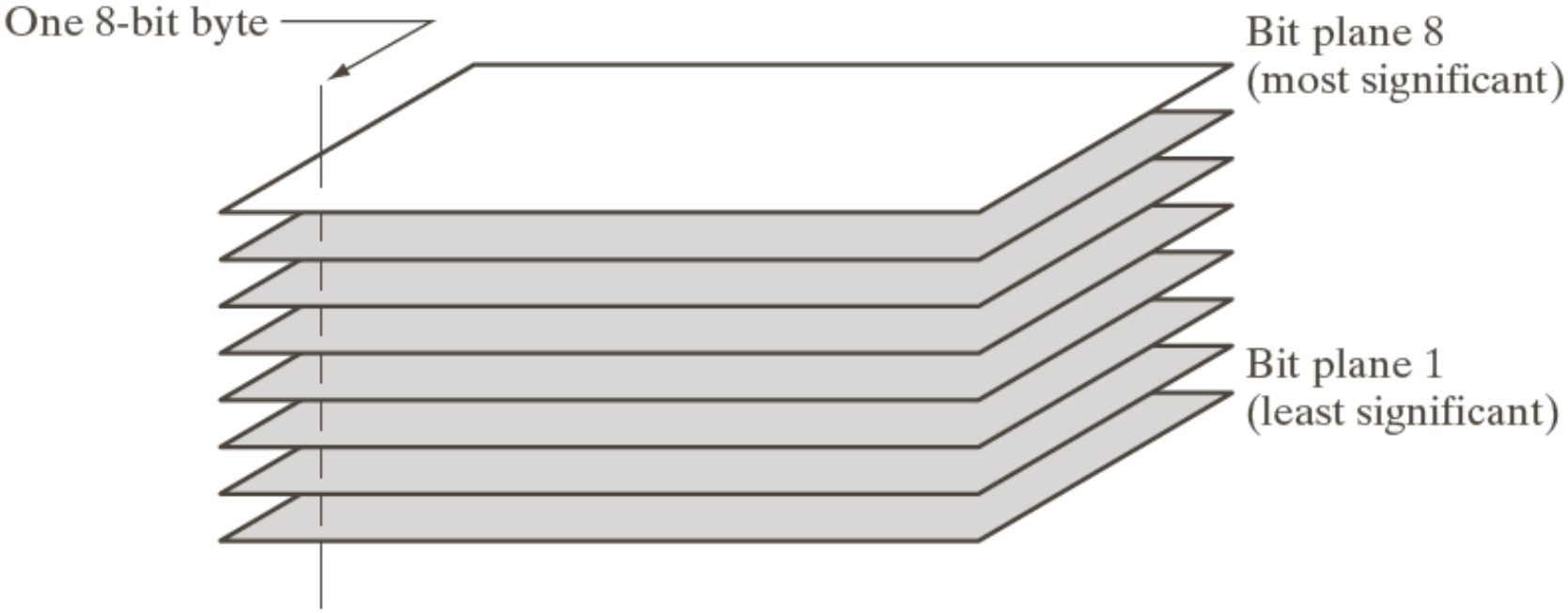
x $2^{(1-1)}$

4	0	4
4	4	0
0	4	0

2	2	2
0	2	1
2	2	2

1	1	0
0	0	1
0	1	0

Lát cắt mặt phẳng bit



8 mặt phẳng bit của ảnh 8-bit

Lát cắt mặt phẳng bit

- Ví dụ lát cắt mặt phẳng bít của ảnh 8 bit có kích thước 550 × 1192 pixel

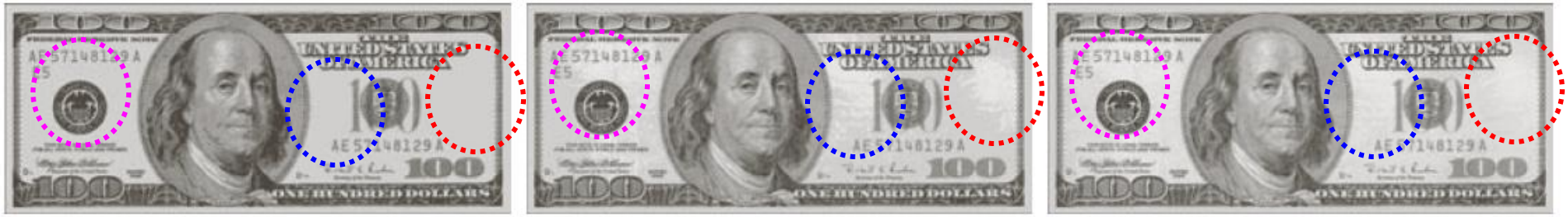
- Cắt thành 8 mặt phẳng bít, mỗi mặt phẳng bit là ảnh nhị phân
- Mặt phẳng bit có thứ tự cao nhất có giá trị 1 1 0 0 0 0 1 0

Lát cắt mặt phẳng bit

a b c
d e f
g h i

FIGURE 3.14 (a) An 8-bit gray-scale image of size 500 × 1192 pixels. (b) through (i) Bit planes 1 through 8, with bit plane 1 corresponding to the least significant bit. Each bit plane is a binary image.

Lát cắt mặt phẳng bit



a b c

FIGURE 3.15 Images reconstructed using (a) bit planes 8 and 7; (b) bit planes 8, 7, and 6; and (c) bit planes 8, 7, 6, and 5. Compare (c) with Fig. 3.14(a).

Lát cắt mặt phẳng bit

Kết quả tái tạo ảnh từ các mặt phẳng bit:

Hình ảnh được tái tạo chỉ sử dụng mặt phẳng bit 8 và 7

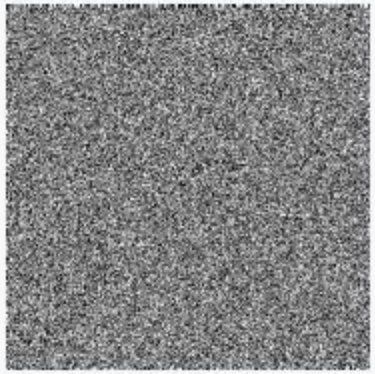
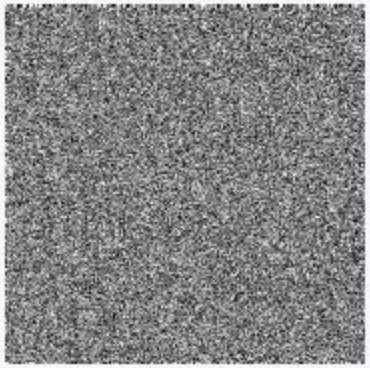
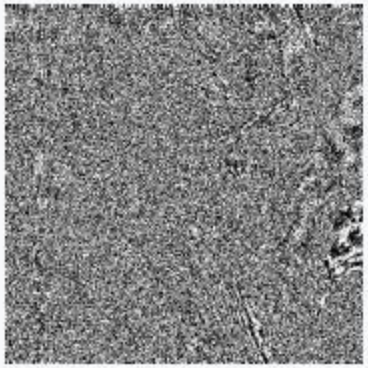
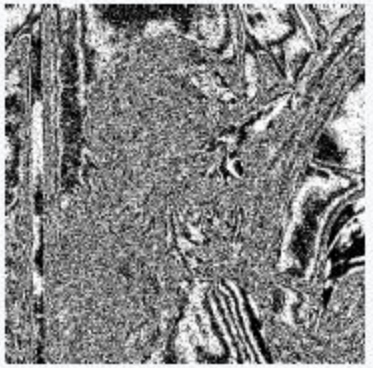
Hình ảnh được tái tạo chỉ sử dụng mặt phẳng bit 8,7 và 6

Hình ảnh được tái tạo chỉ sử dụng mặt phẳng bit 7,6 và 5

Ảnh tái tạo vẫn giữa các chi tiết như ảnh gốc
➔ Ứng dụng nén ảnh

Lát cắt mặt phẳng bit

- **Các bước thực hiện kỹ thuật Lát cắt mặt phẳng bit:**
  - Bước 1: Chuyển mức xám của mỗi pixel sang chuỗi bít nhị phân
  - Bước 2: Tách ảnh tương ứng với vị trí bit trong chuỗi bit nhị phân
  - Bước 3: Tái tạo ảnh từ các mặt phẳng bit

input image			
			
Bit plane 0	Bit plane 1	Bit plane 2	Bit plane 2
			
Bit plane 4	Bit plane 5	Bit plane 6	Bit plane 7
			

```
def int2bitarray(img):
    arr = []
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            arr.append(np.binary_repr(img[i][j], width=8))
    return arr
# read image convert to bit stream
img = cv2.imread('lena.jpg',0)
arr = np.array(int2bitarray(img))
arr = arr.reshape(img.shape)

plane = np.zeros((img.shape))
for k in range(0,8):
    for i in range(arr.shape[0]):
        for j in range(arr.shape[1]):
            plane[i,j]=int(arr[i,j][k])
    plt.imshow(plane*255, cmap='gray')
    plt.show()
```

Make borders for images *(padding)*

- To add borders around the image, similar to a photo frame, the *cv2.copyMakeBorder()* function is used.
- Adding borders has its applications for convolution operations for zero padding, symmetric padding, circular padding etc.
- This function takes the following arguments:
  - **src** - input image
  - **top, bottom, left, right** - Border width in number of pixels in corresponding directions
  - **borderType** - Flag defining what kind of border to be added. It can be following types:
    - cv2.BORDER_REPLICATE - Last element is replicated throughout.
    - cv2.BORDER_REFLECT - Border will be mirror reflection of the border elements.
    - cv2.BORDER_REFLECT_101
    - cv2.BORDER_WRAP
    - cv2.BORDER_CONSTANT - Adds a constant colored border. The value should be given as next argument.
  - **value** - Color of border if border type is cv2.BORDER_CONSTANT

```
# cv2.copyMakeBorder(src, top, bottom, left, right, borderType, value)
replicated = cv2.copyMakeBorder(src, 20, 20, 20, 20, cv2.BORDER_REPLICATE)
reflected = cv2.copyMakeBorder(src, 20, 20, 20, 20, cv2.BORDER_REFLECT)
reflected101 = cv2.copyMakeBorder(src, 20, 20, 20, 20, cv2.BORDER_REFLECT_101)
wrapped = cv2.copyMakeBorder(src, 20, 20, 20, 20, cv2.BORDER_WRAP)
constant= cv2.copyMakeBorder(src, 20, 20, 20, 20, cv2.BORDER_CONSTANT, value=[255, 255, 255])
```

Make borders for images *(padding)*

```
def task3(square=500,padding=10):
    img = np.ones((padding+square+padding,padding+square+padding));

    print np.shape(img);

    #top x rows where x = padding
    img[:padding , :] = 0;

    #bottom x rows where x = padding
    img[-padding: , :] = 0;

    #left x cols where x = padding
    img[: , :padding] = 0;

    #right x cols where x = padding
    img[: , -padding:] = 0;
     cv2.imshow('a',img);
    cv2.waitKey(0);
```

Exercise

Thực hiện các biến đổi sau. Nhận xét các kết quả.

a) *Arithmetic operations on images:*
   - *Addition, subtraction, blending image*

b) *Negative Images*

c) *Log Transformations*

d) *Power-Law Transformations (gamma)*

e) *Bit-plane Slicing*

f) *Make borders for images (padding)*