AIU.
ALAMEIN
INTERNATIONAL UNIVERSITY
جامعة العلمين الدولية
QUALITY ASSURANCE CENTER

Image Processing (CSE281)

Fall 2025/2026

Dr. Essam Abdellatef

Contact Number: 012 8 192 55 90

Email: eabdellatef@Aiu.edu.eg

Quiz (1)

Q1: Given two pixels *P1 (10, 4) and P2 (3, 8),* Compute:

- Euclidean distance
- City Block distance
- Chessboard distance

Q2: Find the convolution between:

$$x(t) = u(t) \text{ and } h(t) = e^{-t}u(t).$$

Binary Segmentation

Binary Segmentation is the process of dividing an image into foreground and background regions by converting it into a binary (black and white) image.

Purpose: To isolate objects of interest from the background, which is useful in object detection and medical imaging.

Principle of Operation: A thresholding method is applied to classify pixels into two categories based on intensity values.

Binary Segmentation

```python
from PIL import Image
import numpy as np
image = Image.open("F:\\13.jpg").convert("L")
img_array = np.array(image)
threshold = 128
binary_array = np.where(img_array > threshold, 255, 0).astype(np.uint8)
binary_image = Image.fromarray(binary_array)
binary_image.show()
```

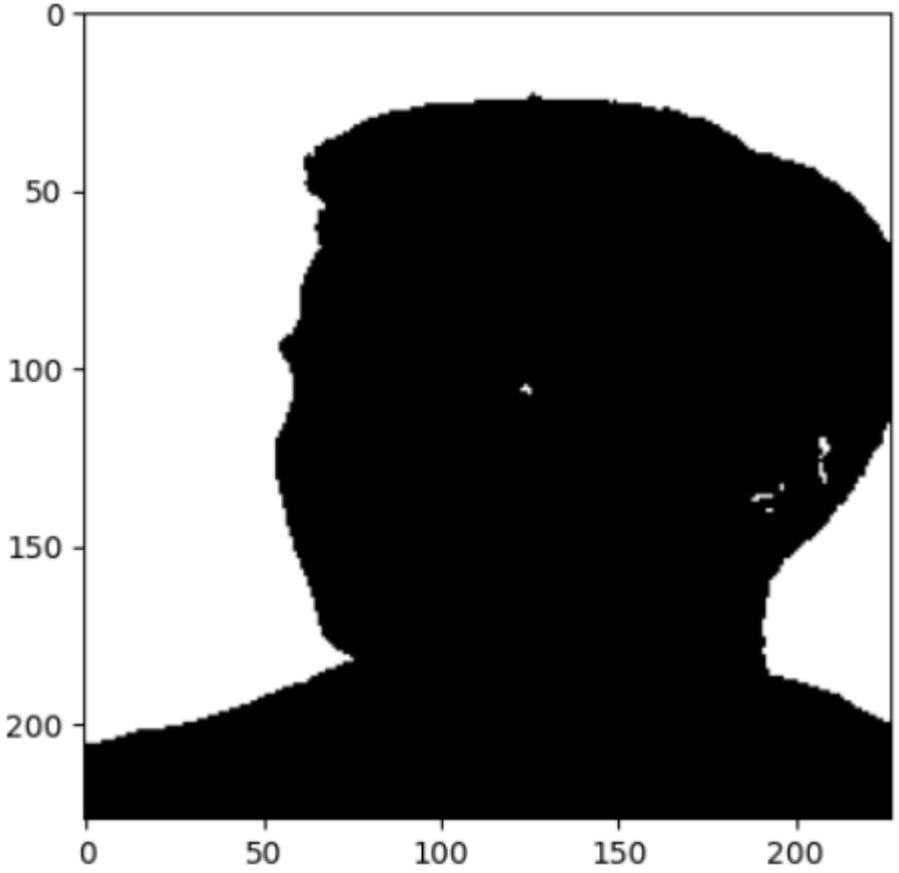
np.array(image) → turns the image into a 2D array of pixel values.

We choose a threshold value (128).
Pixels greater than 128 → set to 255.
Pixels less than or equal to 128 → set to 0.
np.where() → efficiently applies this condition to all pixels.

The binary array is converted back into an image using Image.fromarray().

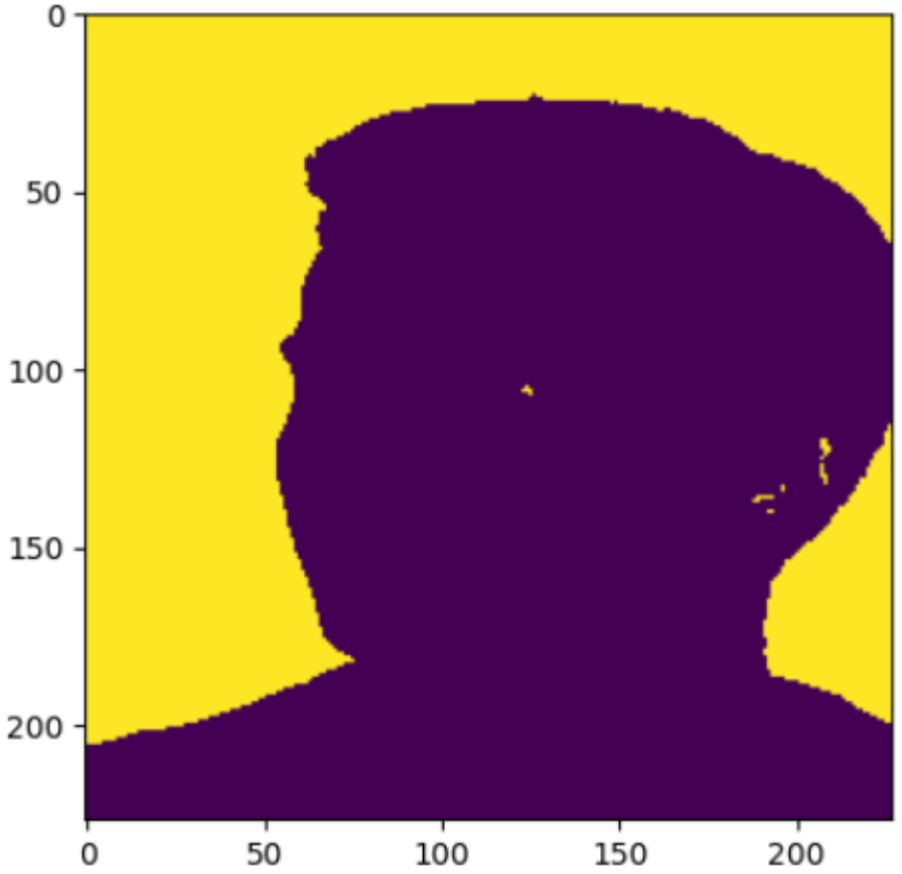
Binary Segmentation

```
from skimage import io
from skimage.color import rgb2gray
from matplotlib import pyplot as plt
image = io.imread("F:\\13.jpg")
img_gray = rgb2gray (image)
BWimage = img_gray.copy()
for i in range(BWimage.shape[0]):
    for j in range(BWimage.shape[1]):
        if (BWimage[i][j] >= 0.5):
            BWimage[i][j] = 1
        else:
            BWimage[i][j] = 0
plt.figure()
plt.imshow(BWimage, 'gray')
```



Binary Segmentation

```
from skimage import io
from skimage.color import rgb2gray
from matplotlib import pyplot as plt
image = io.imread("F:\\13.jpg")
img_gray = rgb2gray(image)
BWimage = img_gray.copy()
for i in range(BWimage.shape[0]):
    for j in range(BWimage.shape[1]):
        if (BWimage[i][j] >= 0.5):
            BWimage[i][j] = 1
        else:
            BWimage[i][j] = 0
plt.figure()
plt.imshow(BWimage)
plt.show()
```



Binary Segmentation

plt.imshow(image)

- Uses viridis colormap (default in matplotlib)
- Displays the image in a colorful scale (blue to yellow)
- Good for highlighting patterns and variations
- Can make grayscale images appear in false colors

plt.imshow(image, 'gray')

- Uses grayscale colormap
- Displays the image in black and white

Binary Segmentation

BWimage.shape[0] → number of rows (height).
BWimage.shape[1] → number of columns (width).
If pixel ≥ 0.5 → set to 1 (white).
Otherwise → set to 0 (black).

.convert("L") → values between 0 – 255
rgb2gray → values between 0 and 1

Image Enhancement

Image enhancement is the process of making images more useful.

The reasons for doing this include:

- Highlighting interesting detail in images.
- Removing noise from images.
- Making images more visually appealing.

Image Enhancement

Most spatial domain enhancement operations can be reduced to the form $g(x, y) = T[f (x, y)]$

- Where $f(x, y)$ *is the input image*, $g(x, y)$ *is* the processed image and T *is some operator.*

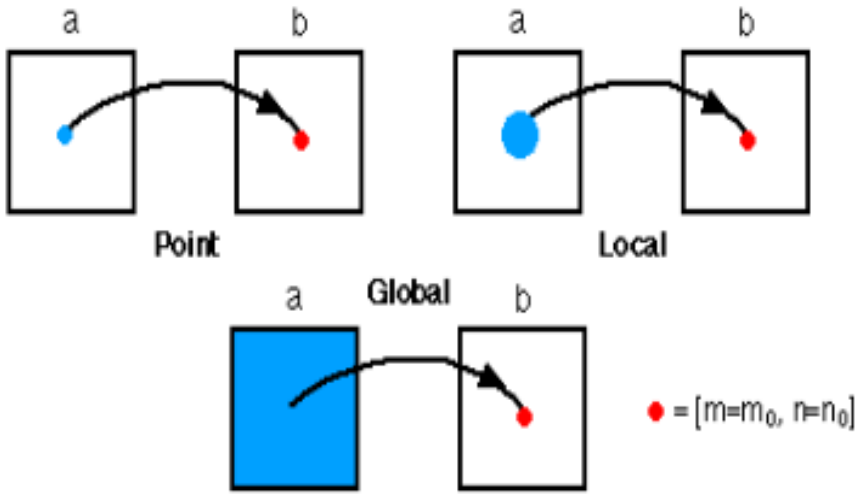
Image Enhancement

Point Operation:

Each pixel in the output image depends only on the corresponding pixel in the input image.

The ***blue dot in image a*** maps directly to the ***red dot in image b.***

There is no influence from neighboring pixels.



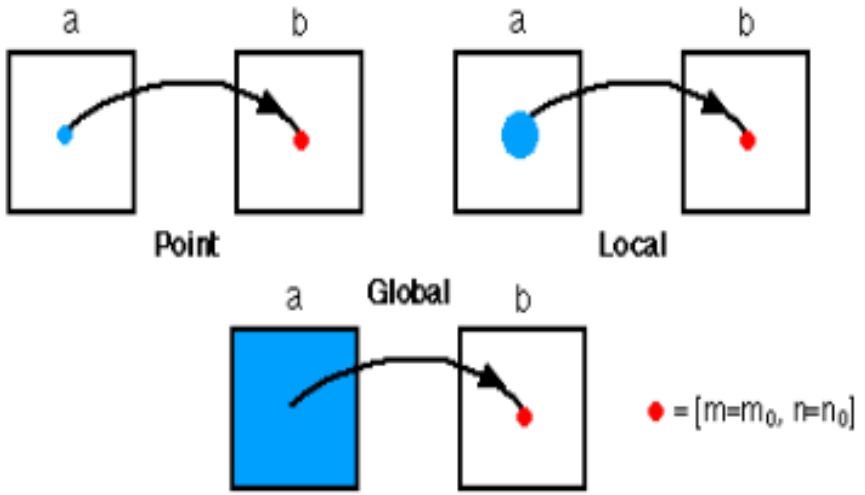
$$b(m,n) = f(a(m,n))$$

Image Enhancement

Local Operation:

Each output pixel depends on the neighborhood (***a small region***) around the corresponding input pixel.

The ***red dot in image b (output)*** is computed from a small area (local region) around the ***blue dot in image a (input)***



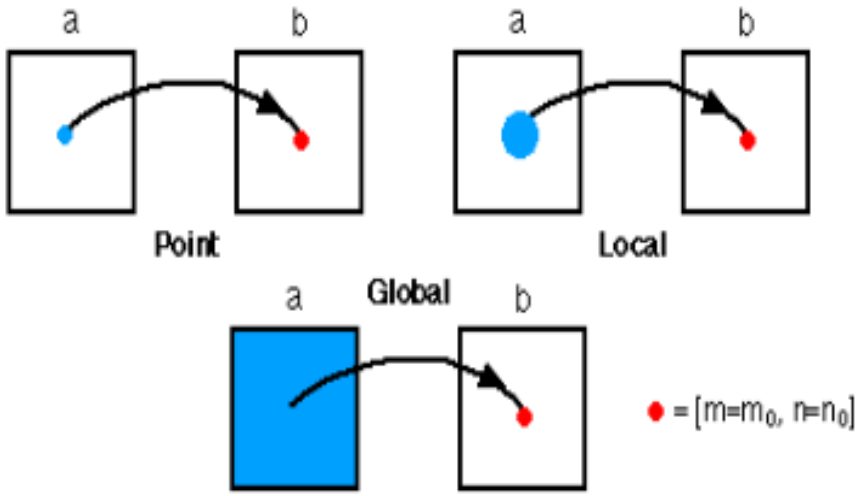
$$b(m,n) = f(a(i,j)), \quad (i,j) \in \text{neighborhood of } (m,n)$$

Image Enhancement

Global Operation:

Each output pixel depends on all pixels in the image.

The ***entire image a (blue region)*** contributes to computing the ***single pixel value at b***.



$$b(m,n) = f(a(i,j)), \quad \forall (i,j)$$

Image Transformation

Most of image processing approaches works directly on spatial domain but in some cases it is better to work on transform domain and applying the inverse transform to return to spatial domain.

A 2-D linear transform can be expressed as:

$$T(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) r(x, y, u, v)$$

Where ***f(x,y)*** is the input image and ***r(x,y,u,v)*** is the forward transformation kernel.

$$r(x, y, u, v) = e^{-j2\pi(ux/M + vy/N)}$$

Image Transformation

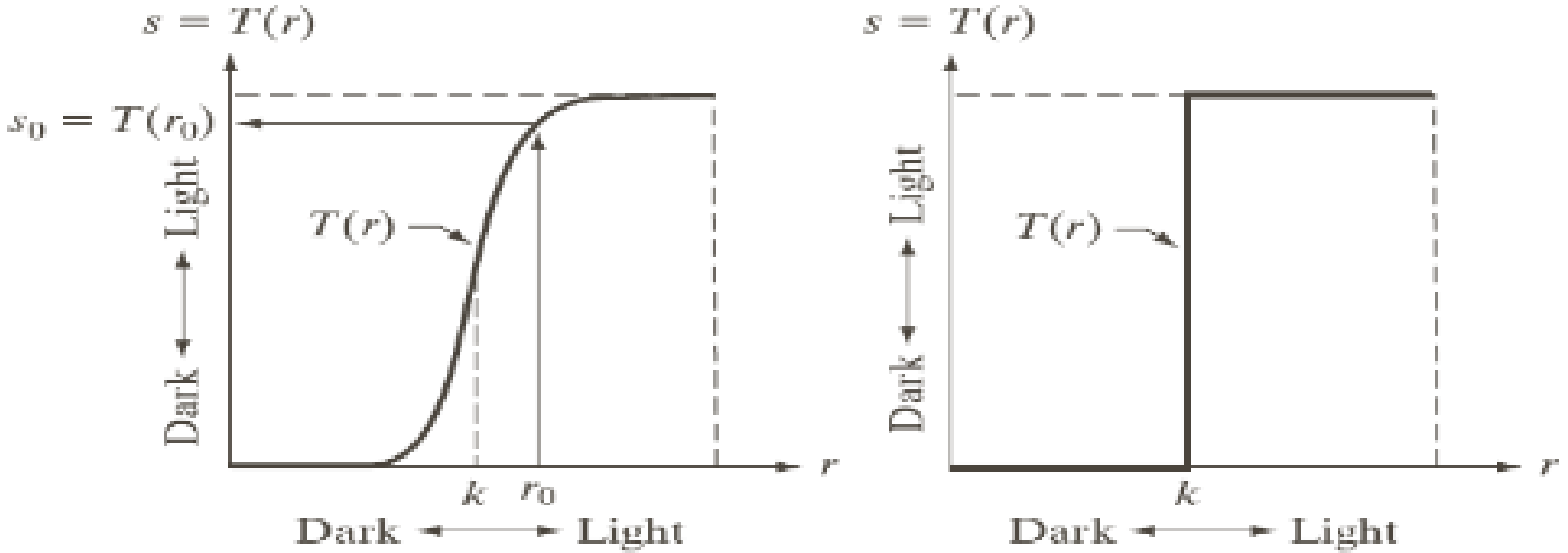
We can recover ***f(x,y)*** using the inverse transform ***T(u,v)***.

$$f(x, y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} T(u, v)s(x, y, u, v)$$

Where ***s(x,y,u,v)*** is called inverse transformation kernel.

$$s(x, y, u, v) = \frac{1}{MN} e^{j2\pi(ux/M + vy/N)}$$

Intensity Transformation

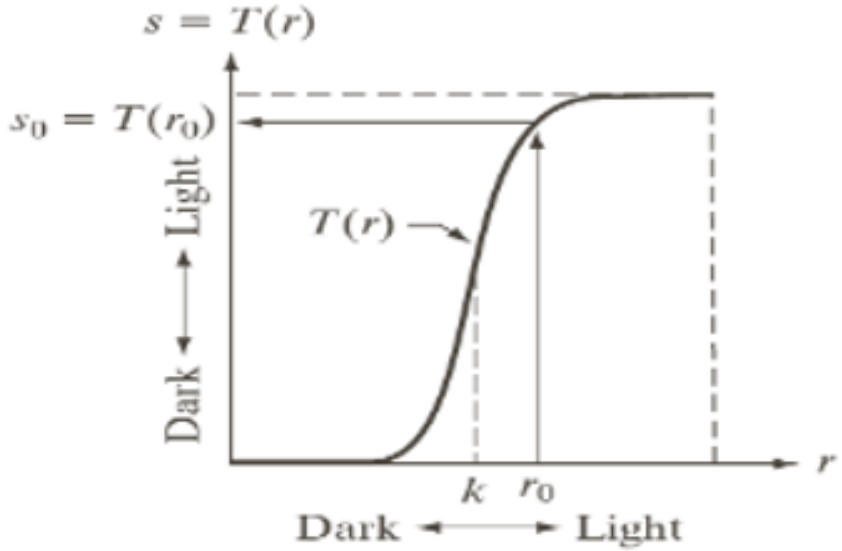


This figure illustrates two intensity transformation functions used in image processing: ***Contrast Stretching*** and ***Thresholding***.

Intensity Transformation

Contrast Stretching

The curve ***s = T(r)*** shows how input gray levels ***r (horizontal axis)*** are mapped to output gray levels ***s (vertical axis).***
r and ***s*** range from dark ***(low values)*** to light ***(high values).***



The inflection point occurs at ***r = k*** (the threshold intensity).

- ***For pixel values smaller than k:*** The transformation compresses those values (they become darker).
- ***For pixel values greater than k:*** The transformation stretches them (they become lighter).

The contrast of the image increases → dark regions become darker, and bright regions become brighter.

Example

Input Image: 8-bit image have range [60-150] After Contrast Stretching: the new range becomes [0-255] using the following equation: $I_{new} = (I - Min)\frac{NewMax - NewMin}{Max - Min} + NewMin$

Where I is the input intensity, I_{new} is the output intensity after contrast stretching, NewMax and NewMin are the new intensity range (0-255), Max and Min are the input intensity range (60-150)

$(100 - 60)\frac{255-0}{150-60} + 0 = 113$, $(125 - 60)\frac{255-0}{150-60} + 0 = 184$

100	120	125
104	110	150
60	130	140

113	170	184
125	142	255
0	198	227

Contrast Stretching

Example:

Apply intensity level slicing to the following image, where:

a. If the intensity in the image is between (***50 → 100***) convert it in the new image into 255 and preserve the background.

b. If the intensity in the image is between (***50 → 100***) convert it in the new image into 255, else convert it to 0.

Contrast Stretching

110	120	130	135
100	94	98	200
30	60	70	30
28	29	25	27

Original Image

110	120	130	135
100	255	255	200
30	255	255	30
28	29	25	27

Sliced Image (a)

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

Sliced Image (b)

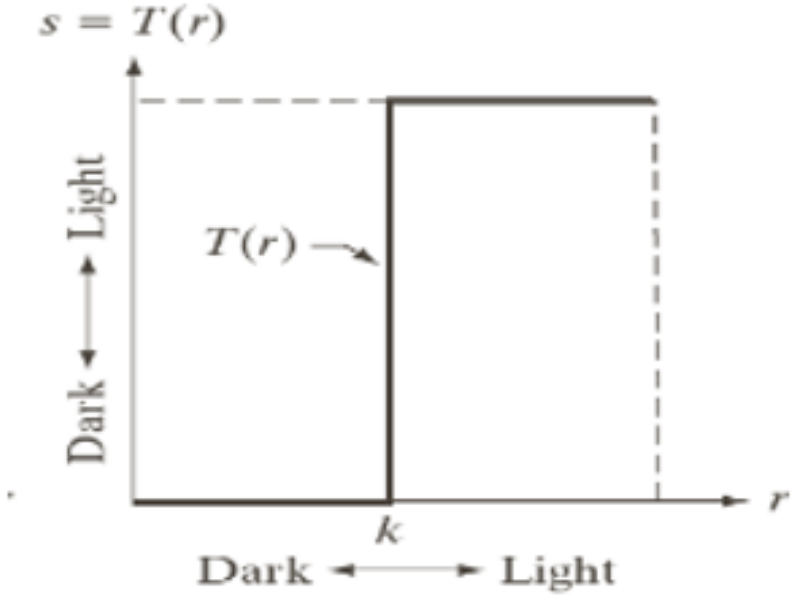
Intensity Transformation

Thresholding

This is a special case of contrast stretching → the curve becomes binary (step function).

For values r < k → s = 0 (black)
For values r ≥ k → s = 1 (white)

Converts a grayscale image into a binary image.
It is used for segmentation, separating objects from background.



Intensity Transformation

There are three basic Types of functions:

- Linear (negative and Identity transformation).
- Logarithmic (Log and invers-log transformation).
- Power-low.

Image Negatives

- The negative of the image with intensity level in the range [0, L-1] is given by the expression: $S = L - 1 - r$
- r → the input pixel intensity.
- s → the output (transformed) pixel intensity.
- L → the maximum possible intensity value (for an 8-bit image, L = 256). So, $S = 255 - r$

This operation reverses (inverts) the intensity levels in the image:

- Dark areas become light.
- Light areas become dark.

Image Negatives

The negative transformation is useful when:

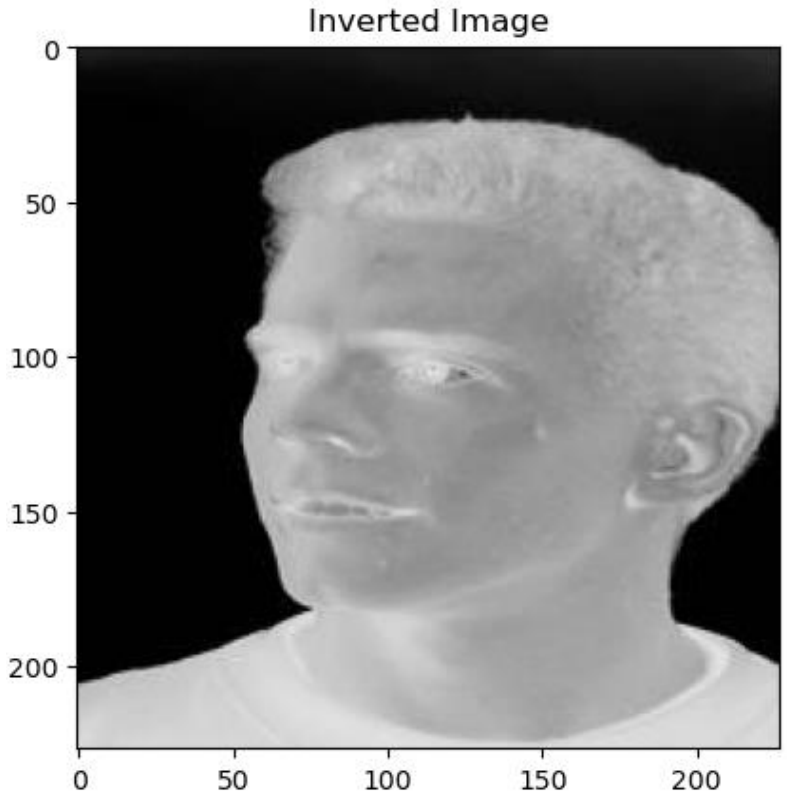
- You need to enhance light or gray details in dark regions of an image.
- It’s easier to visualize details when the dark background is made bright.

Example:

- In X-ray or medical images, the structures of interest (like bones) often appear as light details on a dark background.

Image Negatives

```
from skimage import io
from matplotlib import pyplot as plt
my_image = io.imread("F:\\13.jpg")
plt.figure()
plt.imshow(my_image, 'gray')
plt.title('Original Image')
img_inv = 255 - my_image
plt.figure()
plt.imshow(img_inv, 'gray')
plt.title('Inverted Image')
plt.show()
```



Logarithmic transformation

General form: ***s = c * ln(1 + r)***

- The log transformation maps a narrow range of low input grey level values into a wider range of output values (bright images).
- We add 1 to avoid ln(0) which is undefined
- If r = 0, ln(1+0) = ln(1) = 0
- This ensures all pixel values can be processed
- ***ln*** is just a specific type of ***log***.
- ***ln(x) = log(x) / log(e)***

Logarithmic transformation

Example: Low input values get expanded (let c = 10)

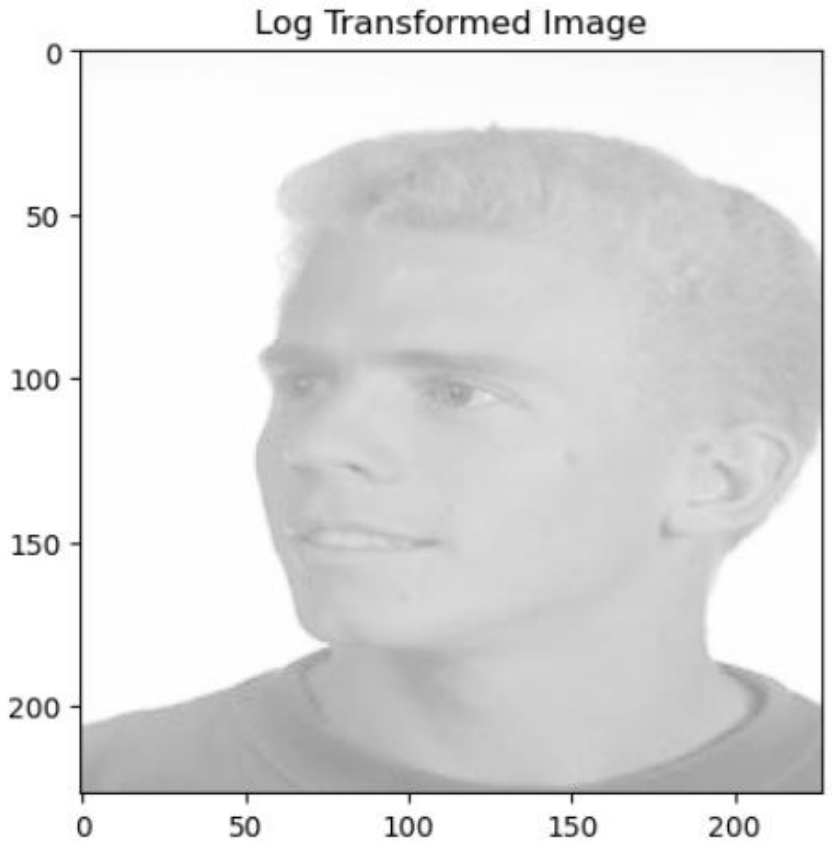
- r = 10 → s = c * ln(11) = 23.9 → s = 10*log(11) / log(e) = 23.9
- r = 20 → s = c * ln(21) = 30.4 → s = 10*log(21) / log(e) = 30.4

Example: High input values get compressed (let c = 10)

- r = 200 → s = c * ln(201) = 53.03 → s = 10*log(201) / log(e) = 53.03
- r = 250 → s = c * ln(251) = 55.25 → s = 10*log(251) / log(e) = 55.25

Logarithmic transformation

```
from skimage import io
from matplotlib import pyplot as plt
import numpy as np
my_image = io.imread(r"F:\13.jpg")
image_float = my_image.astype(np.float64)
c = 255 / np.log(1 + 255)
log_transformed = c * np.log(1 + image_float)
log_transformed = log_transformed.astype(np.uint8)
plt.imshow(log_transformed, 'gray')
plt.title('Log Transformed Image')
```



In Python: np.log(x) or math.log(x) means natural logarithm

Power-law (gamma) transformation

It has the following form: $s = c * r^{\gamma}$

- Power-law curves with fractional values of γ *map a narrow range of* dark input values into a wider range of output values, with the opposite being true for higher values.

Power-law (gamma) transformation

```python
from skimage import io
from matplotlib import pyplot as plt
import numpy as np
my_image = io.imread(r"F:\13.jpg")
image_float = my_image.astype(np.float64) / 255.0
gamma_values = [0.1, 0.5, 1.0, 2.0, 4.0]
c = 1.0
plt.figure(figsize=(15, 10))
for i, gamma in enumerate(gamma_values, 1):
    power_law_transformed = c * (image_float ** gamma)
    power_law_transformed = (power_law_transformed * 255).astype(np.uint8)
    plt.subplot(2, 3, i)
    plt.imshow(power_law_transformed, 'gray')
    plt.title(f'Gamma = {gamma}')
    plt.axis('off')
plt.tight_layout()
plt.show()
```

enumerate (iterable, start)

iterable: any object that can be looped over (list, tuple)

start: the starting value of the counter

Power-law (gamma) transformation

Power-law (gamma) transformation