Image Processing (CSE281)

Fall 2025/2026

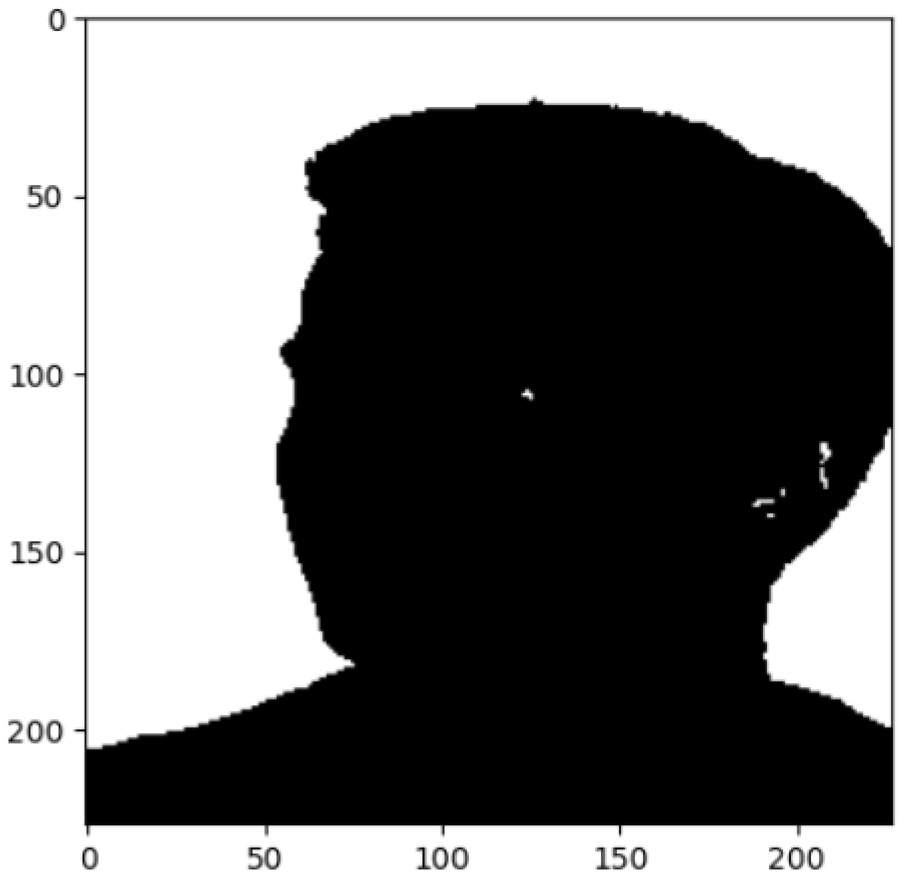
Dr. Essam Abdellatef

Contact Number: 012 8 192 55 90

Email: eabdellatef@Aiu.edu.eg

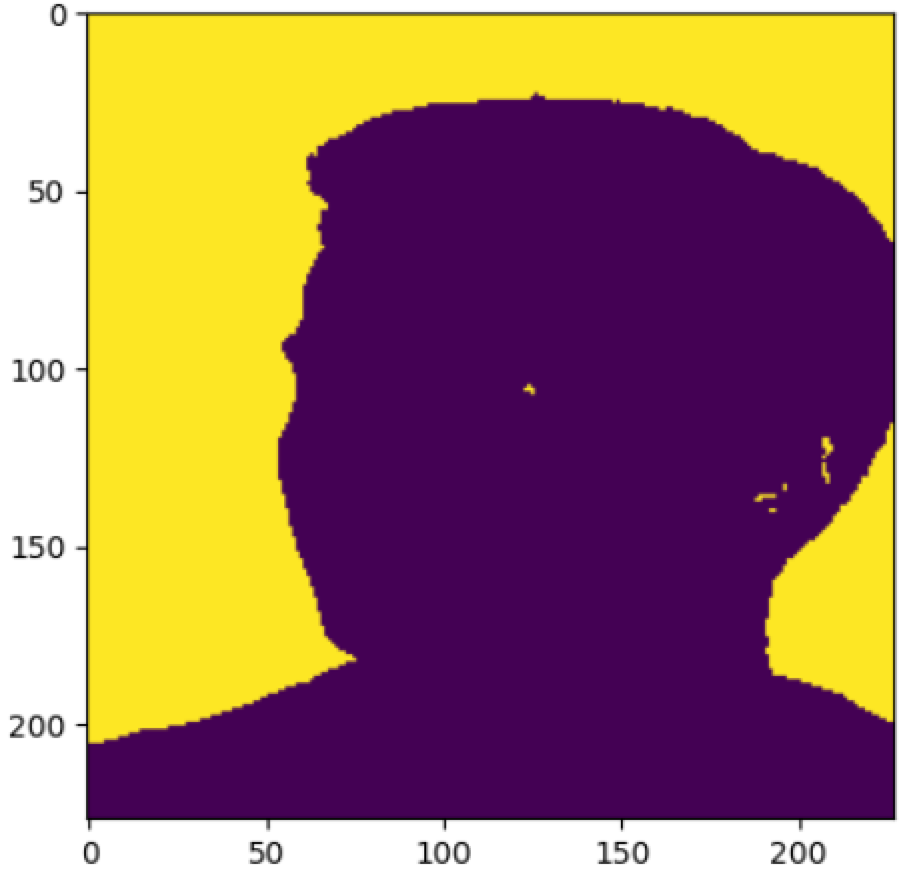
Binary Segmentation

```
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)

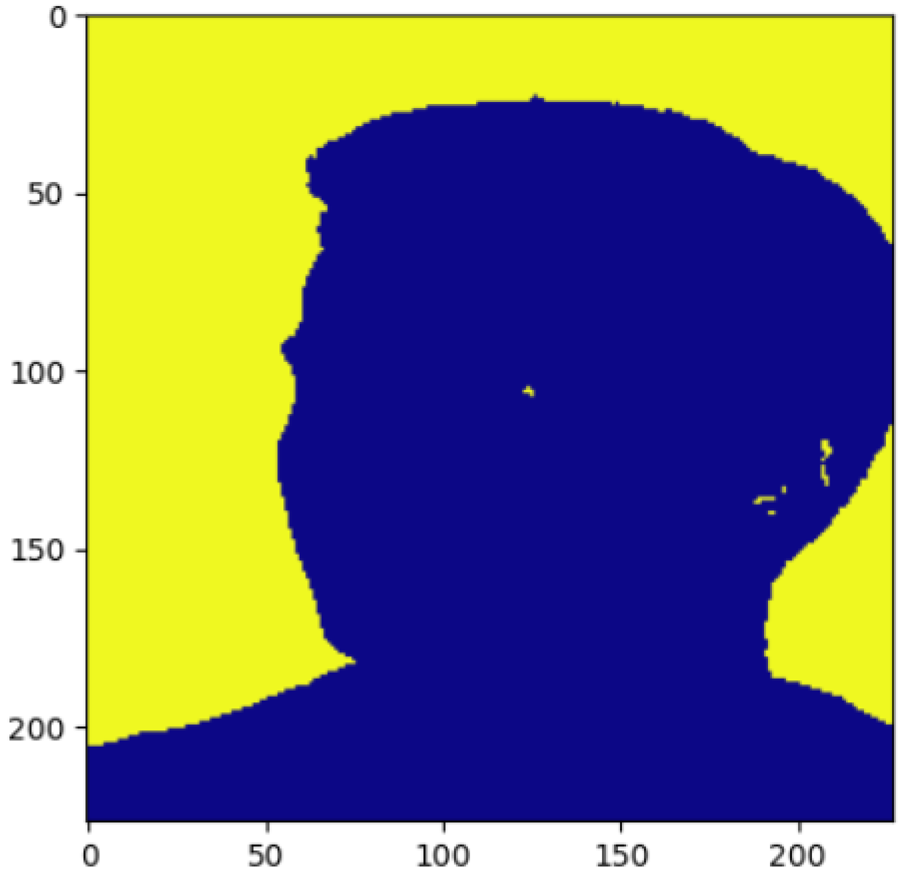
- Uses viridis colormap (default in matplotlib)
- Displays the image in a colorful scale (blue to yellow)
- Good for highlighting patterns and variations
- Can make grayscale images appear in false colors

plt.imshow(image, ‘gray’)

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

Binary Segmentation

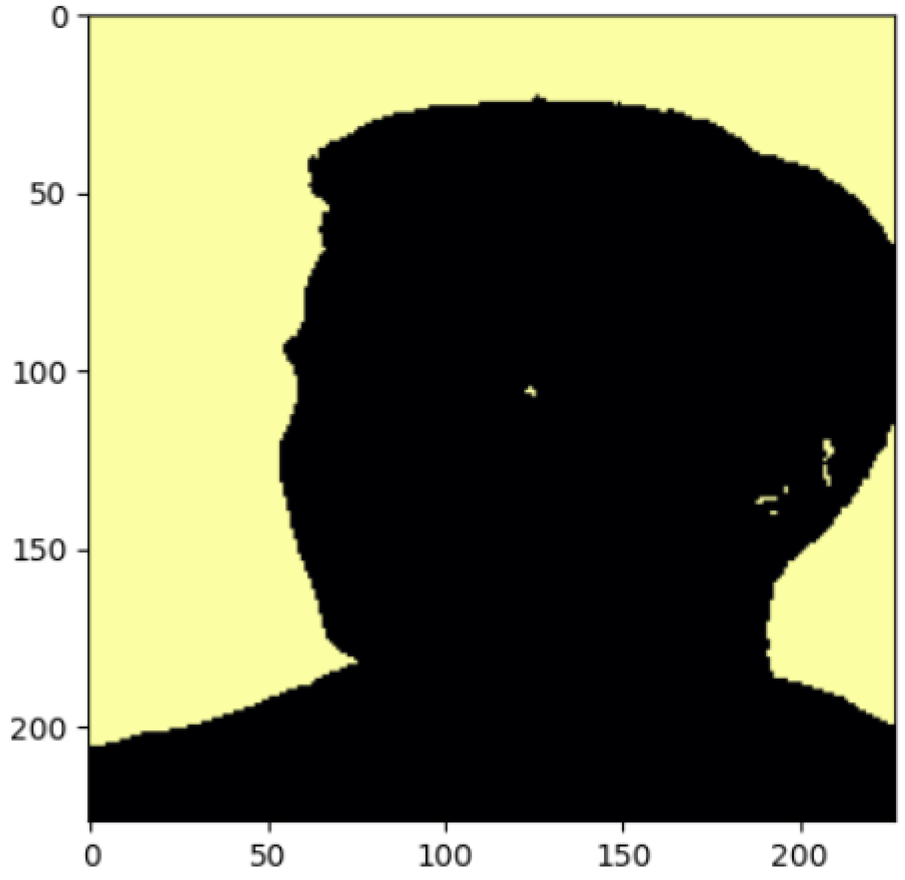
```
from skimage import io
from skimage.color import rgb2gray
from matplotlib import pyplot as plt
image = io.imread("F:\\13.jpg")
img_gray = rgb2gray(image)
BWimage = img_gray.copy()
for i in range(BWimage.shape[0]):
    for j in range(BWimage.shape[1]):
        if BWimage[i][j] >= 0.5:
           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



Purple-red-yellow

Binary Segmentation

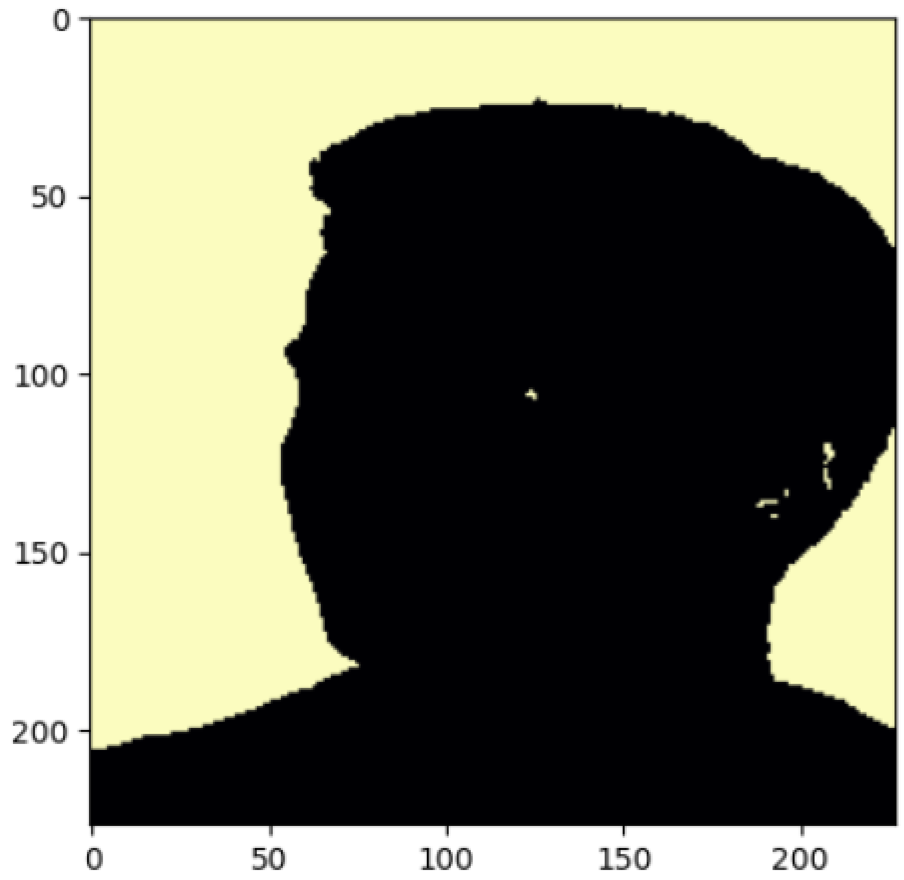
```
from skimage import io
from skimage.color import rgb2gray
from matplotlib import pyplot as plt
image = io.imread("F:\\13.jpg")
img_gray = rgb2gray(image)
BWimage = img_gray.copy()
for i in range(BWimage.shape[0]):
    for j in range(BWimage.shape[1]):
        if BWimage[i][j] >= 0.5:
           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



Black – yellow – red

Binary Segmentation

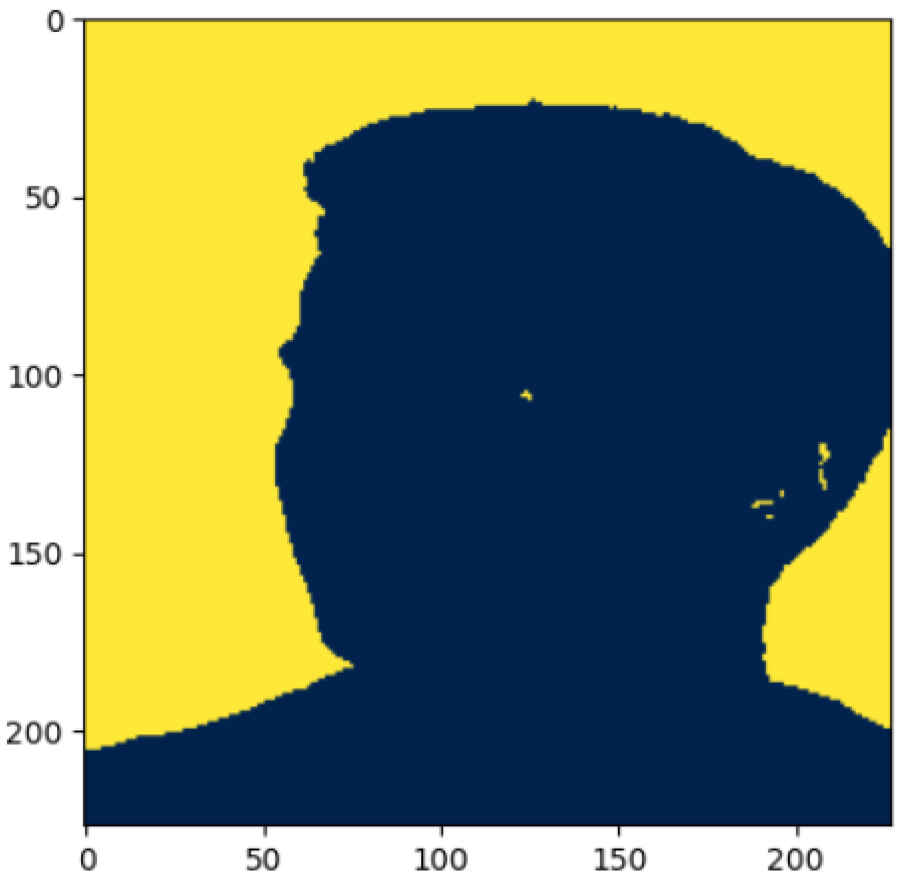
```
from skimage import io
from skimage.color import rgb2gray
from matplotlib import pyplot as plt
image = io.imread("F:\\13.jpg")
img_gray = rgb2gray(image)
BWimage = img_gray.copy()
for i in range(BWimage.shape[0]):
    for j in range(BWimage.shape[1]):
        if BWimage[i][j] >= 0.5:
           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



Black – Purple – pink

Binary Segmentation

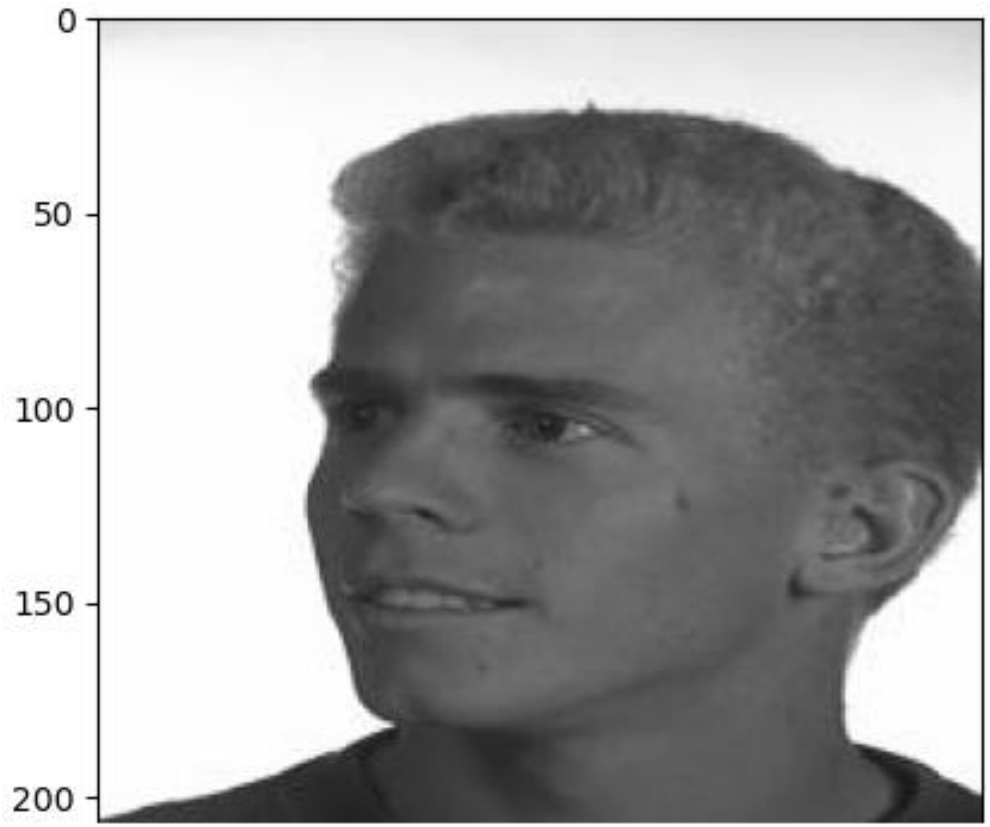
```
from skimage import io
from skimage.color import rgb2gray
from matplotlib import pyplot as plt
image = io.imread("F:\\13.jpg")
img_gray = rgb2gray(image)
BWimage = img_gray.copy()
for i in range(BWimage.shape[0]):
    for j in range(BWimage.shape[1]):
        if BWimage[i][j] >= 0.5:
           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



Blue – Yellow

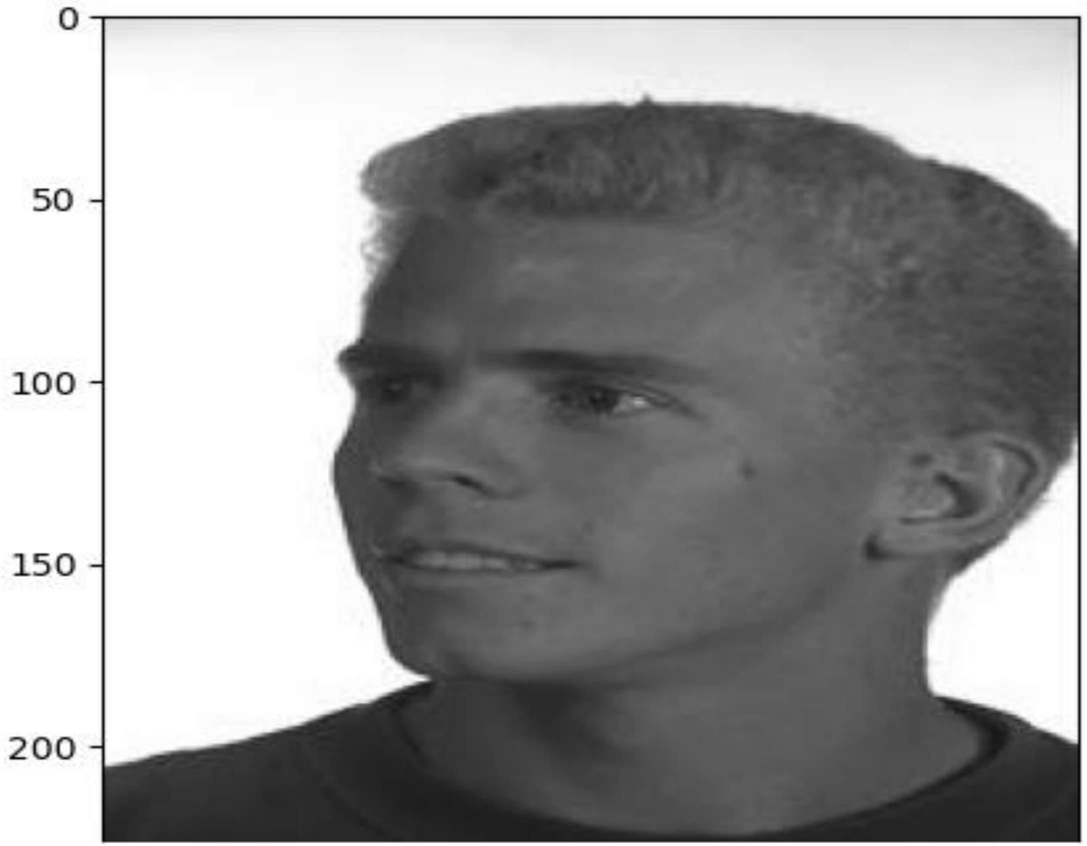
```python
from PIL import Image
from matplotlib import pyplot as plt
im = Image.open("F:\\13.jpg")
plt.figure()
plt.imshow(im, 'gray')
```

<matplotlib.image.AxesImage at 0x28db6c81090>




```python
from PIL import Image
from matplotlib import pyplot as plt
im = Image.open(r"F:\13.jpg")
plt.figure()
plt.imshow(im, cmap = 'cividis')
```

<matplotlib.image.AxesImage at 0x14920b3a490>



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 log(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)

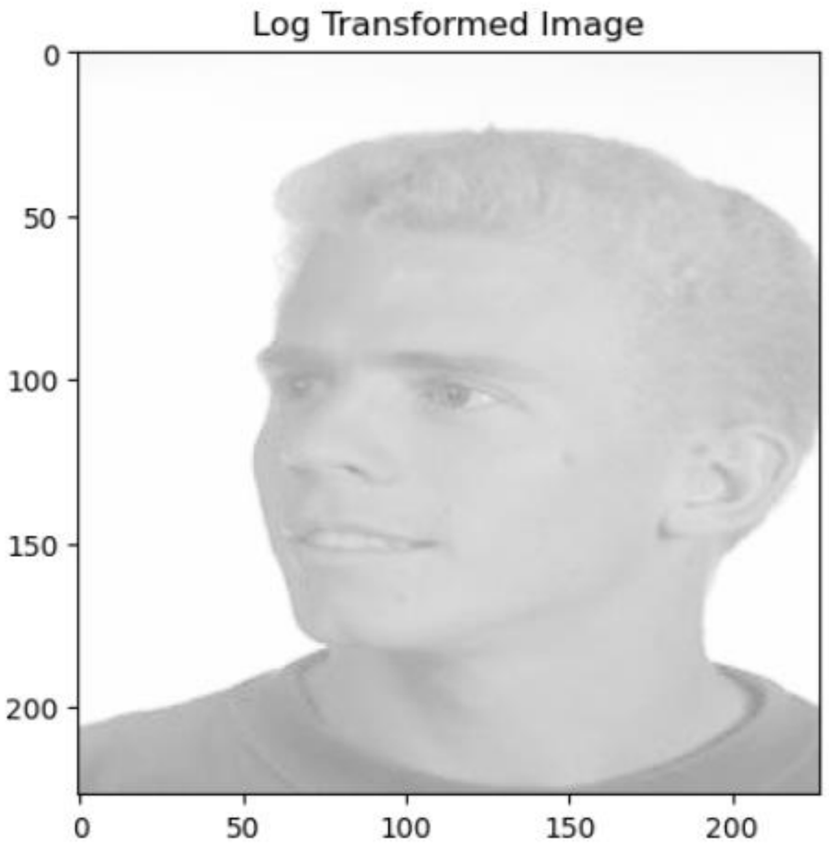
- r = 10 → s = c * ln(11) = 23.9 → s = 10*log(11) / log(e) = 23.9
- r = 20 → s = c * ln(21) = 30.4 → s = 10*log(21) / log(e) = 30.4

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(250) / 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

Logarithmic transformation

image_float = my_image.astype(np.float64)

- Converts image pixel values from integers (0-255) to floating-point numbers
- This prevents issues with logarithmic operations on integers

log_transformed = log_transformed.astype(np.uint8)

- Converts the transformed floating-point values back to 8-bit integers (0-255)

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

```
from skimage import io
from matplotlib import pyplot as plt
import numpy as np
my_image = io.imread("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 in range(len(gamma_values)):
    gamma = gamma_values[i]
    power_law_transformed = c * (image_float ** gamma)
    power_law_transformed = (power_law_transformed * 255).astype(np.uint8)
    plt.subplot(2, 3, i+1)
    plt.imshow(power_law_transformed, 'gray')
    plt.title(f'Gamma = {gamma}')
    plt.axis('off')
plt.tight_layout()
plt.show()
```

plt.axis('off') removes the axes
plt.tight_layout() optimizes the spacing between subplots

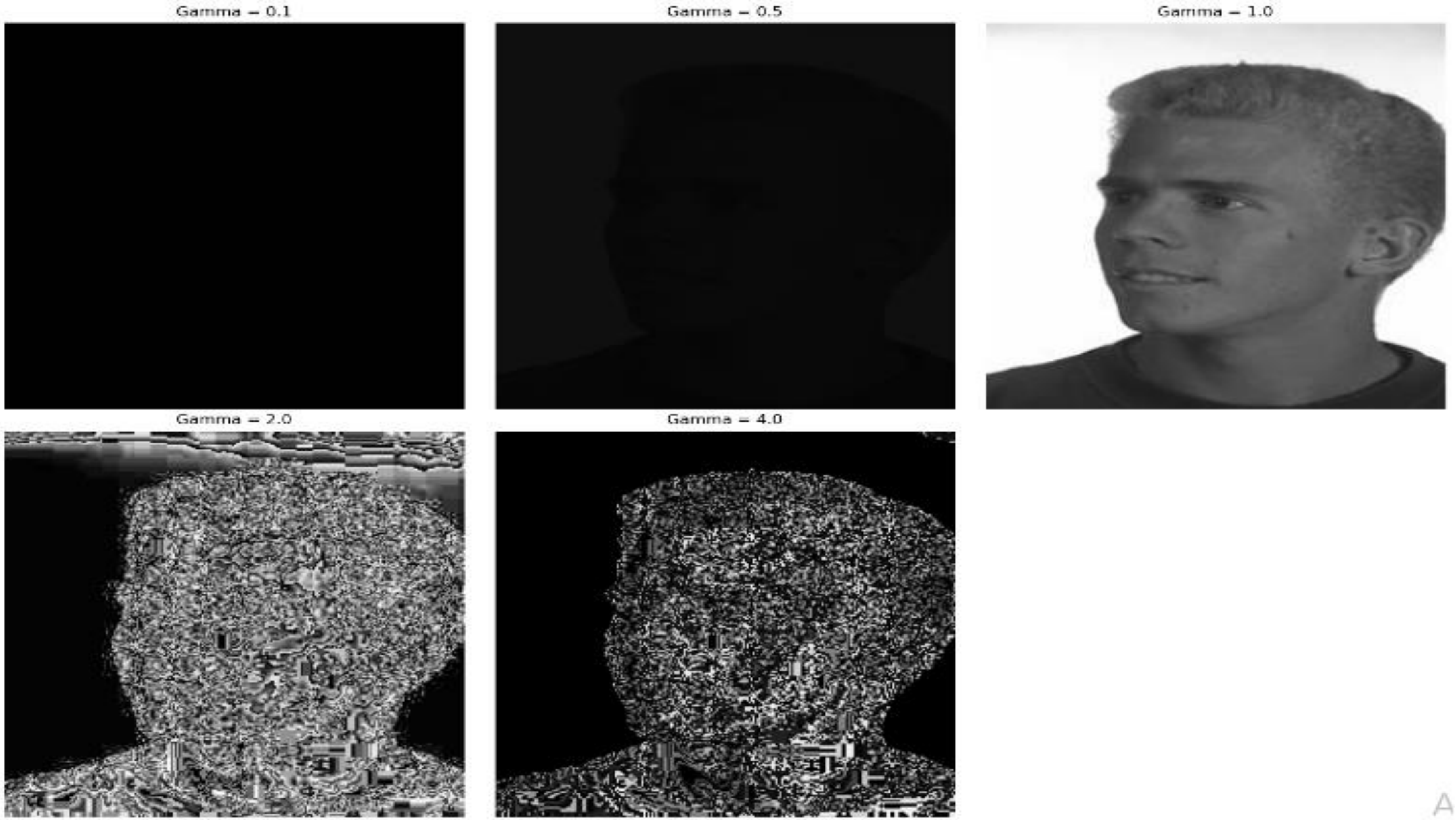
Power-law (gamma) transformation

Power-law (gamma) transformation

Power-law (gamma) transformation

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

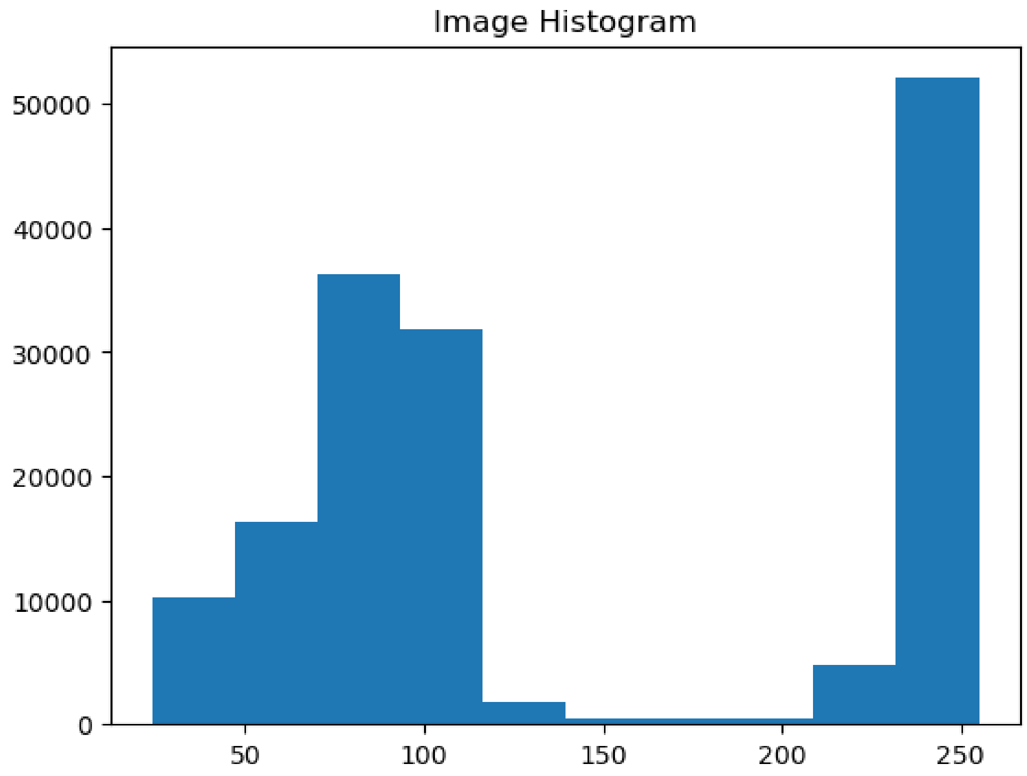
Power-law (gamma) transformation



Histogram Processing

An image histogram is a graphical representation of the tonal distribution in a digital image. It plots the number of pixels for each intensity value.

```
from skimage import io
from matplotlib import pyplot as plt
import numpy as np
my_image = io.imread(r"F:\13.jpg")
plt.figure()
plt.hist(my_image.ravel())
plt.title('Image Histogram')
plt.show()
```



Histogram Processing

my_image.ravel () → converts the image pixels into a simple list of values that plt.hist() can process to count how many pixels have each intensity value.

Histogram Equalization

Histogram equalization is an image enhancement technique that improves contrast by redistributing pixel intensity values so that they cover the entire possible range more evenly.

In a dark or low-contrast image, many pixel values are clustered in a narrow range (***for example, between 50 and 120 out of 255***).

Histogram equalization spreads these pixel values across the full range (0 – 255) → ***enhanced contrast***.

Histogram Equalization

Compute the PDF (Probability Density Function):

$$p_r(r_k) = \frac{n_k}{n}$$

nk: **number of pixels with intensity *rk***

n: **total number of pixels**

Compute the CDF (Cumulative Distribution Function):

$$s_k = T(r_k) = \sum_{j=0}^{k} p_r(r_j)$$

Histogram Equalization

Map intensity values:

$$s'_k = (L - 1) \times s_k$$

This scales the result back to the valid range [0, L−1].
Replace each pixel intensity ***rk*** with ***s'k*** → This yields the equalized image.

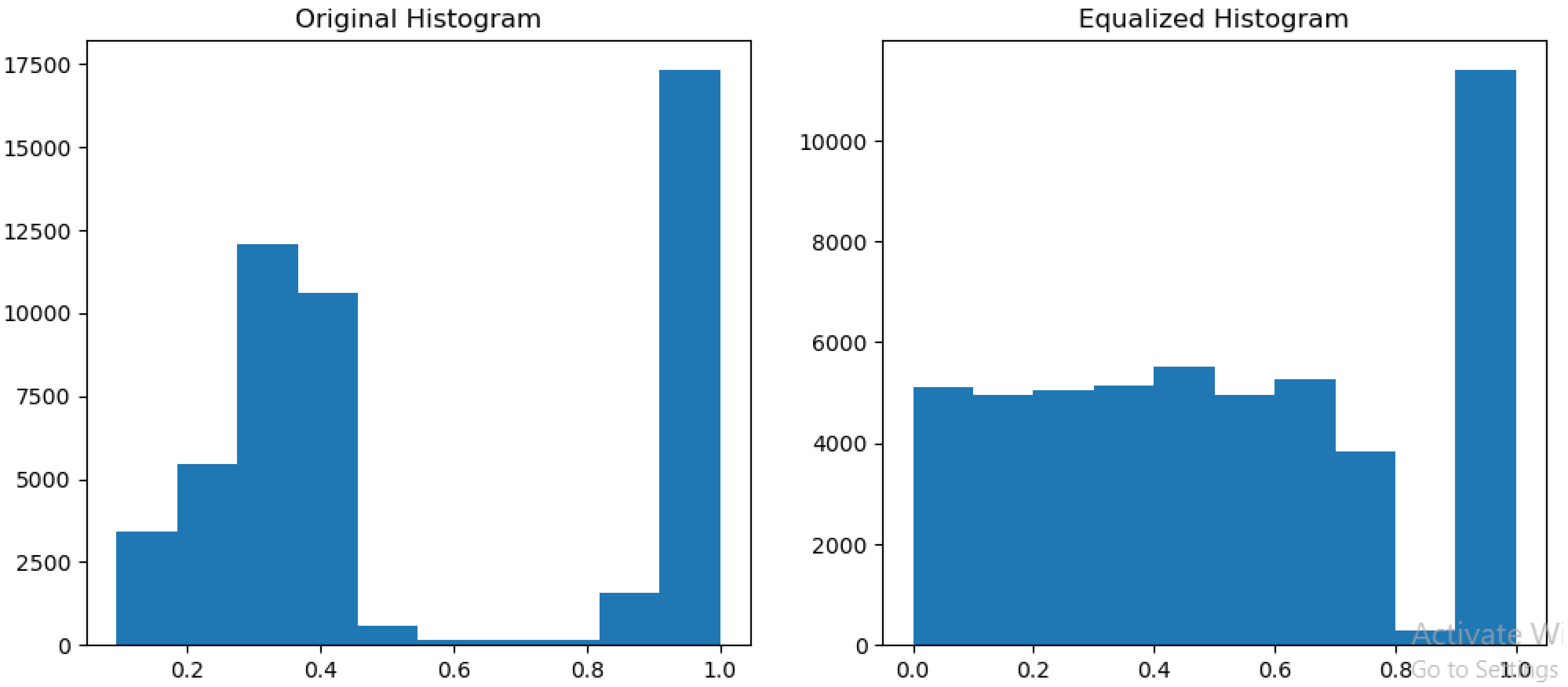
Histogram Equalization

- ✓ Dark regions become more visible.
- ✓ Light regions gain more contrast.
- ✓ The histogram of the equalized image becomes more uniform (spread out across all intensity levels).

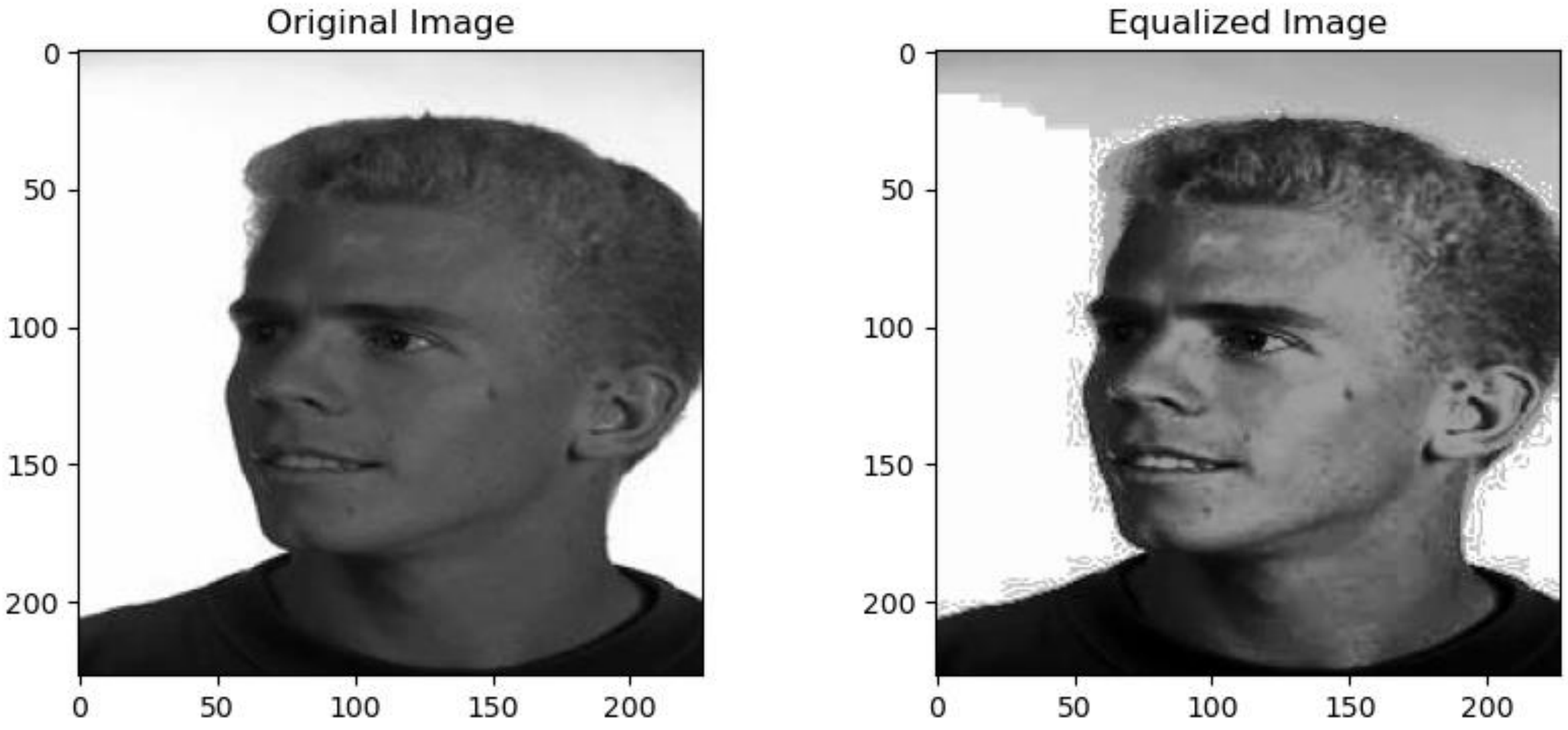
```python
from skimage import io, exposure, color
from matplotlib import pyplot as plt
import numpy as np
my_image = io.imread(r"F:\13.jpg")
gray_image = color.rgb2gray(my_image)
plt.figure(figsize = (12, 5))
plt.subplot(1, 2, 1)
plt.hist(gray_image.ravel())
plt.title('Original Histogram')
equalized_image = exposure.equalize_hist(gray_image)
plt.subplot(1,2,2)
plt.hist(equalized_image.ravel())
plt.title('Equalized Histogram')
plt.show()
plt.figure(figsize = (10, 4))
plt.subplot(1,2,1)
plt.imshow(gray_image, cmap='gray')
plt.title('Original Image')
plt.subplot(1,2,2)
plt.imshow(equalized_image, cmap='gray')
plt.title('Equalized Image')
plt.show()
```

exposure.equalize_hist (gray_image) → is a function that performs histogram equalization.

Histogram Equalization



Histogram Equalization



Example

Apply histogram equalization to the following image

3-bit image: 2^3 = 8 possible intensity levels (0-7)

Image size: 4×8 = 32 pixels

Original intensity values: 0, 1, 2, 3, 4

0	1	1	2	2	3	4	4
0	1	1	2	2	3	4	4
0	1	1	2	2	3	4	4
0	1	1	2	2	3	4	4

Example

Calculate Probability Density Function (PDF)

Intensity (r_k)	Count (n_k)	PDF: $P_r(r_k) = n_k/MN$
0	4	4/32 = 0.125
1	8	8/32 = 0.25
2	8	8/32 = 0.25
3	4	4/32 = 0.125
4	8	8/32 = 0.25

Example

Calculate Cumulative Distribution Function (CDF)

r_k	$P_r(r_k)$	Cumulative Sum	CDF
0	0.125	0.125	0.125
1	0.25	0.125+0.25=0.375	0.375
2	0.25	0.375+0.25=0.625	0.625
3	0.125	0.625+0.125=0.75	0.75
4	0.25	0.75+0.25=1.0	1.0

Example

Apply Histogram Equalization Transform

r_k	CDF	$s_k = 7 \times CDF$	Rounded s_k
0	0.125	0.875	1
1	0.375	2.625	3
2	0.625	4.375	4
3	0.75	5.25	5
4	1.0	7.0	7

Histogram Equalization

Example: Apply the histogram transformation to a 3-bit image ($L = 8$) of size 64×64 pixels ($MN = 4096$).

Intensity	Pixel Count
0	800
1	600
2	500
3	400
4	500
5	600
6	400
7	296

Histogram Equalization

Calculate Probability Density Function (PDF)

r_k	n_k	$P_r(r_k)$
0	800	0.195
1	600	0.146
2	500	0.122
3	400	0.098
4	500	0.122
5	600	0.146
6	400	0.098
7	296	0.072

Histogram Equalization

Calculate Cumulative Distribution Function (CDF)

r_k	$P_r(r_k)$	$CDF(r_k)$
0	0.195	0.195
1	0.146	0.341
2	0.122	0.463
3	0.098	0.561
4	0.122	0.683
5	0.146	0.829
6	0.098	0.927
7	0.072	1.000

Histogram Equalization

Apply the Mapping to Original Image

r_k	$CDF(r_k)$	$s_k = 7\times CDF$	Rounded s_k
0	0.195	1.365	1
1	0.341	2.387	2
2	0.463	3.241	3
3	0.561	3.927	4
4	0.683	4.781	5
5	0.829	5.803	6
6	0.927	6.489	6
7	1.000	7.000	7