

1. Construct the Huffman code of the source whose symbol probabilities are defined below. [3 points]

r_k	$p_r(r_k)$
$r_0 = 0$	0.11
$r_1 = 1/7$	0.01
$r_2 = 2/7$	0.09
$r_3 = 3/7$	0.17
$r_4 = 4/7$	0.23
$r_5 = 5/7$	0.07
$r_6 = 6/7$	0.17
$r_7 = 1$	0.15

Huffman Coding

range all probabilities in ascending order

r_k	r_1	r_5	r_2	r_0	r_7	r_3	r_6	r_4
$p_r(r_k)$	0.01	0.07	0.09	0.11	0.15	0.17	0.17	0.23

insert first two elements which are having smaller probabilities.

		0.08	
--	--	------	--

r_k	r_1	r_2	r_0	r_7	r_3	r_6	r_4
$p_r(r_k)$	0.08	0.09	0.11	0.15	0.17	0.17	0.23

insert next smaller numbers & insert in correct place in ascending order.

0.17	0.165582
------	----------

0.165582	0.08
----------	------

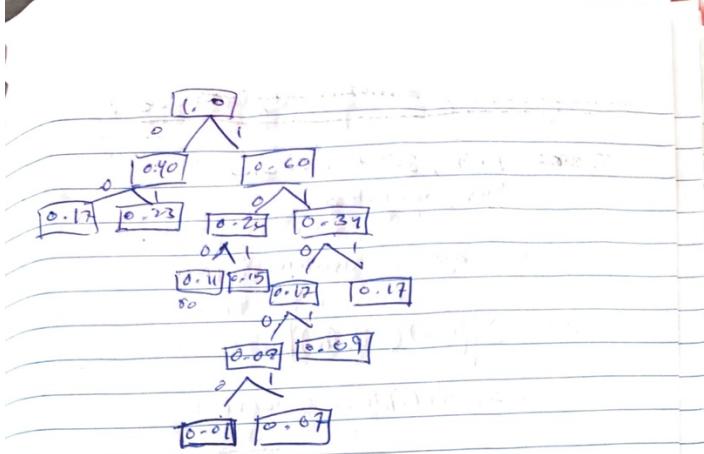
0.26	0.057	0.11	0.15	0.17	0.17	0.17	0.23
$p_r(r_k)$							

insert first two elements

0.26	0.057	0.11	0.15	0.17	0.17	0.17	0.23
$p_r(r_k)$							

0.54

0.54	0.17	0.17	0.17	0.17	0.17	0.17	0.23
$p_r(r_k)$							



Tree	$P_F(S_k)$	code
S_0	0.11	100
S_1	0.01	11000
S_2	0.09	1001
S_3	0.12	111
S_4	0.23	01
S_5	0.07	11001
S_6	0.12	00
S_7	0.15	101

2. Consider the simple 7×7 , 8-bit image: [6 points]

77	122	111	0	0	255	255
0	111	111	77	111	77	77
111	111	0	111	111	0	255
122	0	111	111	122	255	0
111	111	77	111	111	255	255
49	49	3	49	49	3	3
3	122	111	200	122	111	111

a. Compress the image using Huffman coding

b) Average number of bits required to represent each pixel in the Huffman encoded image is:

$$\text{Average} = 9(7/49) + 6(4/49) + 7(4/49) + \\ 5(5/49) + 2(17/49) + 5(5/49) + \\ 7(1/49) + 2(6/49) \\ = \frac{183}{49} = 3.73 \text{ bits/symbols}$$

Image data before compression:

Entropy of image:

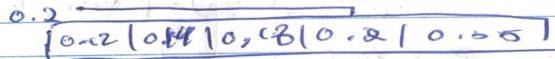
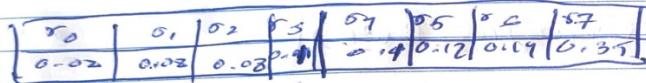
$$H = - \sum_{k=0}^{K-1} [p_k(\text{symbol}) \log_2 p_k(\text{symbol})]$$

$$= - [7/49 \log_2 (7/49) + 4/49 \log_2 (4/49) + \\ 4/49 \log_2 (4/49) + 5/49 \log_2 (5/49) + \\ 17/49 \log_2 (17/49) + 5/49 \log_2 (5/49) + \\ 1/49 \log_2 (1/49) + 6/49 \log_2 (6/49)] \\ = 8 \text{ bits/symbols}$$

b. Compute the compression achieved and the effectiveness of Huffman coding.

@ composition using Hadamard coding techniques.

Number	Probability	Code (Huffman)
06	0	0000
02	3	000001
01	49	0000000
04	77	000000
07	111	00
05	102	00001
08	200	0000001
05	255	01



Original image data was : 8-bits/symbols

After compression using Huffman coding technique,
reduced the original image to

$$\boxed{\text{Coverage} = 3.73 \text{ bits/symbols}}$$

Reducing the image by $\boxed{4.27 \text{ bits/symbols}}$.

3. Write programs to perform the following transformations on the grayscale images (you may pick any image). Display the output images along with the respective input images. [11 points]

$$g(x, y) = \begin{cases} 0 & \text{if } f(x, y) < t \\ L-1 & \text{if } f(x, y) \geq t \end{cases} \quad \text{where } t = 70 \text{ and } 170$$

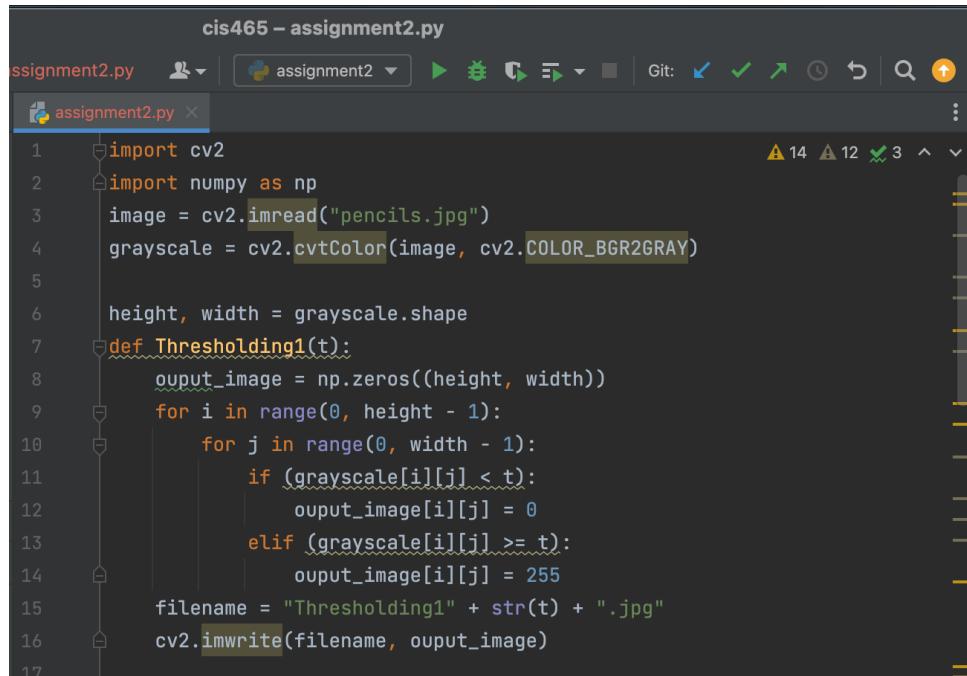
$$g(x, y) = \begin{cases} 0 & \text{if } f(x, y) < t1 \\ f(x, y) & \text{if } t1 \leq f(x, y) \leq t2 \\ 0 & \text{if } f(x, y) > t2 \end{cases} \quad \text{where } t1 = 70 \text{ and } t2 = 170$$

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

$$s = c r^\eta$$

Output for:

$$g(x, y) = \begin{cases} 0 & \text{if } f(x, y) < t \\ L-1 & \text{if } f(x, y) \geq t \end{cases} \quad \text{where } t = 70 \text{ and } 170$$



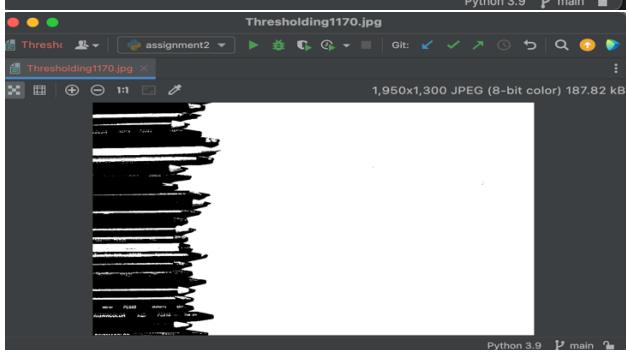
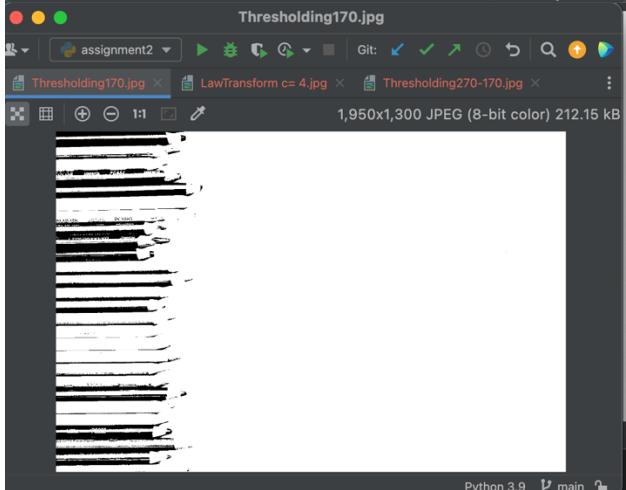
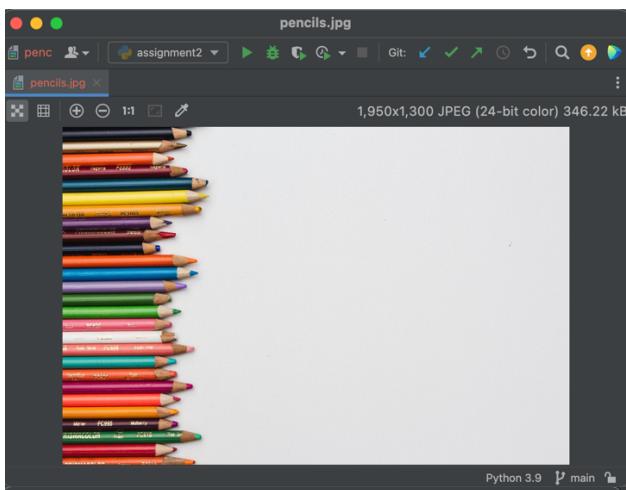
The screenshot shows a code editor window titled "cis465 – assignment2.py". The file "assignment2.py" is open, containing the following Python code:

```

1  import cv2
2  import numpy as np
3  image = cv2.imread("pencils.jpg")
4  grayscale = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
5
6  height, width = grayscale.shape
7  def Thresholding1(t):
8      ouput_image = np.zeros((height, width))
9      for i in range(0, height - 1):
10         for j in range(0, width - 1):
11             if (grayscale[i][j] < t):
12                 ouput_image[i][j] = 0
13             elif (grayscale[i][j] >= t):
14                 ouput_image[i][j] = 255
15
16     filename = "Thresholding1" + str(t) + ".jpg"
17     cv2.imwrite(filename, ouput_image)

```

The code imports cv2 and numpy, reads a grayscale image named "pencils.jpg", converts it to grayscale, and defines a function "Thresholding1" that takes a threshold value "t". The function creates a zeroed output image of the same dimensions as the input. It then iterates through each pixel, setting it to 0 if its grayscale value is less than "t" and to 255 if it is greater than or equal to "t". Finally, it saves the resulting binary image with a filename like "Thresholding170.jpg".



Output for:

$$g(x, y) = \begin{cases} 0 & \text{if } f(x, y) < t1 \\ f(x, y) & \text{if } t1 \leq f(x, y) \leq t2 \\ 0 & \text{if } f(x, y) > t2 \end{cases} \quad \text{where } t1 = 70 \text{ and } t2 = 170$$

The screenshot shows a Jupyter Notebook interface with three code cells and their corresponding outputs.

Code Cell 1:

```
# assignment2.py
# import numpy as np
# import math
# image = cv2.imread("pencils.jpg")
grayscale = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
height, width = grayscale.shape

def PowerLawTransform(c, gama):
    ouput_image = np.zeros((height, width))
    for i in range(0, height - 1):
        for j in range(0, width - 1):
            ouput_image[i][j] = c * pow(grayscale[i][j], gama)

filename = "LawTransform" + " c= " + str(c) + ".jpg"
cv2.imwrite(filename, ouput_image)

# for C=4 and gama=1
PowerLawTransform(4, .7)
```

Code Cell 2:

pencils.jpg

1,950x1,300 JPEG (24-bit color) 346.22 kB

Code Cell 3:

Thresholding270-170.jpg

1,950x1,300 JPEG (8-bit color) 287.04 kB

Output for:

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

$$S = C \cdot r^\eta$$

The screenshot shows a PyCharm IDE interface with the following details:

- Title Bar:** cis465 – assignment2.py
- Toolbars:** Standard PyCharm toolbars for file operations, navigation, and Git integration.
- Status Bar:** Shows code coverage metrics: 14, 12, 3, and 1%.
- Code Editor:** Displays Python code for thresholding an image. The code uses OpenCV's cv2 module to convert an image to grayscale and then applies a thresholding operation based on two thresholds, t1 and t2. It outputs the result as a binary image where pixels are either black or white.
- Bottom Bar:** Shows imports at the bottom of the script.

```
grayscale = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
height, width = grayscale.shape

def Thresholding2(t1, t2):
    output_image = np.zeros((height, width))
    for i in range(0, height - 1):
        for j in range(0, width - 1):
            if (grayscale[i][j] < t1):
                output_image[i][j] = 0
            elif (t1 <= grayscale[i][j] <= t2):
                output_image[i][j] = grayscale[i][j]
            elif (grayscale[i][j] > t2):
                output_image[i][j] = 0

filename = "Thresholding2" + str(t1) + "-" + str(t2) + ".jpg"
cv2.imwrite(filename, output_image)

Thresholding2(70, 170)
# import cv2
```

