EE 596 – LABORATORY 01
HUFFMAN CODING

AMARASINGHE E.G.C.L.

E/18/023

SEMESTER 7

01/12/2023

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING,

FACULTY OF ENGINEERING,

UNIVERSITY OF PERADENIYA

EE 596: IMAGE AND VIDEO CODING

LABORATORY 01-HUFFMAN CODING

INTRODUCTION:

In this lab we focus on compression of digital images. Compression is a general concept that applies to all types of data, not just images. Image compression schemes aim to decrease the number of bits used to represent the image. Using fewer bits allow the image to take up less storage space on a computer, and it can also be transmitted faster over a network. In this lab, we apply Huffman codes to achieve lossless compression in a given image and compare the output quality of the input image and reconstructed image.

Standard Software: MATLAB

Note: You may use C++ or Python. You are not allowed to use any libraries unless otherwise stated by the instructor.

LABORATORY ACTIVITY:

Step 1: Download the images from the webpage (Instructor will provide the URL at the lab).

Step 2: Read the original image into a Matrix.

Step 3: Select 16×16 cropped sub-image from your input at step2. Note that the starting point of the cropping window will depend on your Registration number. (Instructor will provide these details at the lab.)

Step 4: Quantize the output at Step 3 into 8 levels (level 0-7) using uniform quantization.

Step 5: Find the probability of each symbol distribution of the output at Step 4.

Step 6: Construct the Huffman coding algorithm for cropped image at Step 4.(Do not use inbuilt algorithms.)

Step 7: Compress both cropped and original images using the algorithm and the codebook generated at step 6. You may round any intensity values outside the codebook, to the nearest intensity value in the codebook, where necessary.

Step 8: Save the compressed image into a text file.

Step 9: Compress the original image using Huffman encoding function in the Matlab tool box and save it into another text file.

Step 10: Decompress the outputs at Step 8 and 9, by reading in the text files.

Step 11: Calculate the entropy of the Source

Step 12: Evaluate the PSNR of

i. The original images

ii. The decompressed images

DISCUSSION:

1. Calculate the entropy of,
    i. The original image
    ii. The cropped image
    iii. The decompressed images
2. Calculate the average length of the cropped image.
3.Compare the performance of your algorithm and inbuilt algorithm of Matlab by comparing the compression ratios, for cropped and original images.
3. Discuses about Entropy of the input image, the compression ratio achieved, and the output quality of the decompressed image.
4. How can you improve the compression ratio of the given image? Discuss.

LAB REPORT:

1. Submit MATLAB code and the cropped image (Make sure that each figure is properly labeled and also include your registration number in the title).
2. The report must be a pdf file with the figures and MATLAB codes for each step in the laboratory activity. Include extractions from the m file, and command prompt, figures. Also, please include the probability distribution of the cropped image and the Huffman code developed
3. The answers to discussion question should also be provided on the same pdf file.
4. Adhere to a proper report writing format throughout the lab report.

LABORATORY ACTIVITY

Step 1: Download the images from the webpage (Instructor will provide the URL at the lab).

Step 2: Read the original image into a Matrix.

```
original_img = cv2.imread("pattern.jpg")
original_img_array = np.array(original_img)
cv2.imshow('origional IMAGE (E/18/023)',original_img_array)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

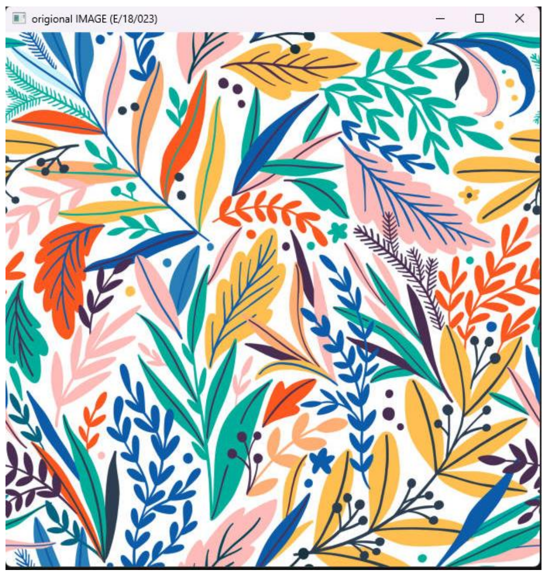


Figure 01 : Original image

Step 3: Select 16×16 cropped sub-image from your input at step2

```
x = 0*60
y = 23*4

cropped_img = original_img[y:y+16, x:x+16]

cv2.imshow('cropped image (E/18/023)',cropped_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

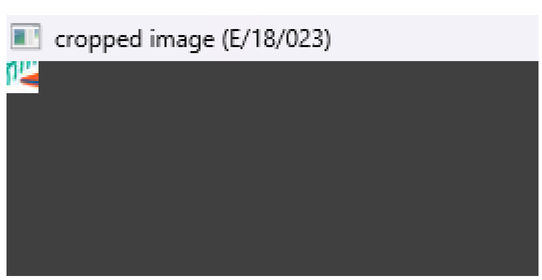


Figure 02 : Cropped image

Step 4: Quantize the output at Step 3 into 8 levels (level 0-7) using uniform quantization.

```
red_img = original_img[:,:,2]

cv2.imshow('origional red image (E/18/023)',red_img)
cv2.waitKey(0)
cv2.destroyAllWindows()

red_cropped_img = red_img[y:y+16, x:x+16]

cv2.imshow('cropped red image (E/18/023)',red_cropped_img)
cv2.waitKey(0)
cv2.destroyAllWindows()

quantised_img =quantiser(red_cropped_img)

cv2.imshow('quantized red image (E/18/023)',quantised_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
# Quantizer -------------------------------------------------------
def quantiser(array):

    # max=np.max(red_array)
    # min=np.min(red_array)
    max=255
    min=0

    q = (max-min)/7

    for i in range(array.shape[0]):
        for j in range(array.shape[1]):
            if array[i][j]>=min and array[i][j]<min+q/2 :
                    array[i][j]= min
            elif array[i][j]>=min+q/2 and array[i][j]<min+3*q/2 :
                    array[i][j]= min + q
            elif array[i][j]>=min+3*q/2 and array[i][j]<min+5*q/2 :
                    array[i][j]= min + 2*q
            elif array[i][j]>=min+5*q/2 and array[i][j]<min+7*q/2 :
                    array[i][j]= min + 3*q
            elif array[i][j]>=min+7*q/2 and array[i][j]<min+9*q/2 :
                    array[i][j]= min + 4*q
            elif array[i][j]>=min+9*q/2 and array[i][j]<min+11*q/2 :
                    array[i][j]= min + 5*q
            elif array[i][j]>=min+11*q/2 and array[i][j]<min+13*q/2 :
                    array[i][j]= min + 6*q
            elif array[i][j]>=min+13*q/2 and array[i][j]<min+15*q/2 :
                    array[i][j]= min + 7*q        You, yesterday • till huff

    return np.array(array)

# ----------------------------------------------------------------
```

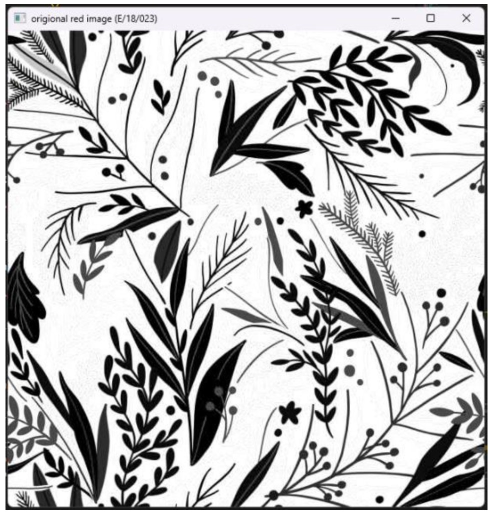


Figure 03 : Red-filtered image

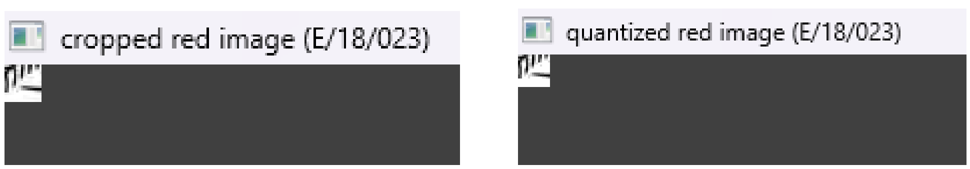


Figure 04 : Cropped and quantized image

Step 5: Find the probability of each symbol distribution of the output at Step 4.

```
img_probability = probability(quantised_img)
print('image probabilities :')
print(img_probability)
```

```
# Probability ---------------------------------------------------

def probability(q_vals):

    unique_values, counts = np.unique(q_vals, return_counts=True)

    value_counts = dict(zip(unique_values, counts))

    for key in value_counts:
        value_counts[key]=value_counts[key]/(16*16)

    return value_counts

# -------------------------------------------------------------
```

```
image probabilities :
{72: 0.02734375, 109: 0.1171875, 145: 0.14453125, 182: 0.1015625, 218: 0.109375, 255: 0.5}
```

Figure 05 : probability distribution

Step 6: Construct the Huffman coding algorithm for cropped image at Step 4.

```python
codeBook = img_probability.copy()

sorted_img_prob = sort(img_probability)

huf_img = huffman(sorted_img_prob,codeBook)

print('code book :')
print(codeBook)
```

```python
# Huffman ---------------------------------------------------------

def huffman(sorted_dict,input_dict):
    for key in input_dict:
        input_dict[key] = ""

    while (len(sorted_dict)>1):
        new_value = list(sorted_dict.values())[0] + list(sorted_dict.values())[1]
        key = str(list(sorted_dict.keys())[0])+"_"+str(list(sorted_dict.keys())[1])

        sorted_dict[key] = new_value

        result = str(list(sorted_dict.keys())[0]).split('_')

        for i in result:
            input_dict[int(i)] = "0" + input_dict[int(i)]

        sorted_dict.pop(list(sorted_dict.keys())[0])

        result = str(list(sorted_dict.keys())[0]).split('_')

        for i in result:
            input_dict[int(i)] = "1"+input_dict[int(i)]

        sorted_dict.pop(list(sorted_dict.keys())[0])

        # Step 4: Sort the dictionary again using bubblesort
        sorted_dict = sort(sorted_dict)

    return input_dict

# ------------------------------------------------------------------
```

```python
# Sort data ------------------------------------------------------

def sort(input_dict):
    items = list(input_dict.items())
    n = len(items)

    for i in range(n - 1):
        for j in range(0, n - i - 1):
            if items[j][1] > items[j + 1][1]:
                items[j], items[j + 1] = items[j + 1], items[j]

    sorted_img = dict(items)

    return sorted_img

# ---------------------------------------------------------------
```

```
code book :
{72: '1100', 109: '101', 145: '111', 182: '1101', 218: '100', 255: '0'}
```

Figure 06 : Code book

Step 7: Compress both cropped and original images using the algorithm and the codebook generated at step 6.

Step 8: Save the compressed image into a text file.

```
compress(red_cropped_img,codeBook,"cropped")
compress(red_img,codeBook,"original")
```

```
# compress -------------------------------------------------------

def compress(arr,arr2,name):
    arr1=np.array(arr[:,:])

    q=36.4285714

    hight=arr1.shape[0]
    width=arr1.shape[1]

    file_path = str(name)+".txt"
    with open(file_path, "w") as file:
        count=0

        for i in range(hight):
            for j in range(width):
                for l in list(arr2.keys()):
                    if arr1[i][j]>=(l-q/2) and arr1[i][j]<(l+q/2):

                        # Write data to the file
                        file.write(arr2[l])
                        count+=1
                        break

#------------------------------------------------------------------
```

Step 9: Compress the original image using Huffman encoding function in the Matlab tool box and save it into another text file.

Step 10: Decompress the outputs at Step 8 and 9, by reading in the text files.

```
file = open("cropped.txt","r")
decodedImg = decode(file,codeBook,16,16,'cropped')

file = open("original.txt","r",)
decodedImg = decode(file,codeBook,red_img.shape[0],red_img.shape[1],'original')
```

```
# Decoding -----------------------------------------------------------------

def decode(c_img,cBook,hight,width,name):

    # decompresed the rgb img

    code=list(cBook.values())
    key=list(cBook.keys())


    f1=c_img.readline()

    arr19=np.zeros((hight,width, 3),dtype=np.uint8)

    count =0

    for i in range(hight):
            for j in range(width):
                count += 1
                print(count)
                s=""
                for x in f1:
                    s+=x

                    if s in code:
                        index_of_element = code.index(s)
                        arr19[i][j]=key[index_of_element]

                        if len(f1)>len(s):
                            f1=f1[len(s):]

                        s=""
                        break

    display = str(name)+" decoded image (E/18/023)"
    cv2.imshow(display, np.array(arr19))
    cv2.waitKey(0)      You, 1 second ago • Uncommitted changes
    cv2.destroyAllWindows()
    print(arr19)
    return

# --------------------------------------------------------------------------
```

Decompressed images for part 8 :

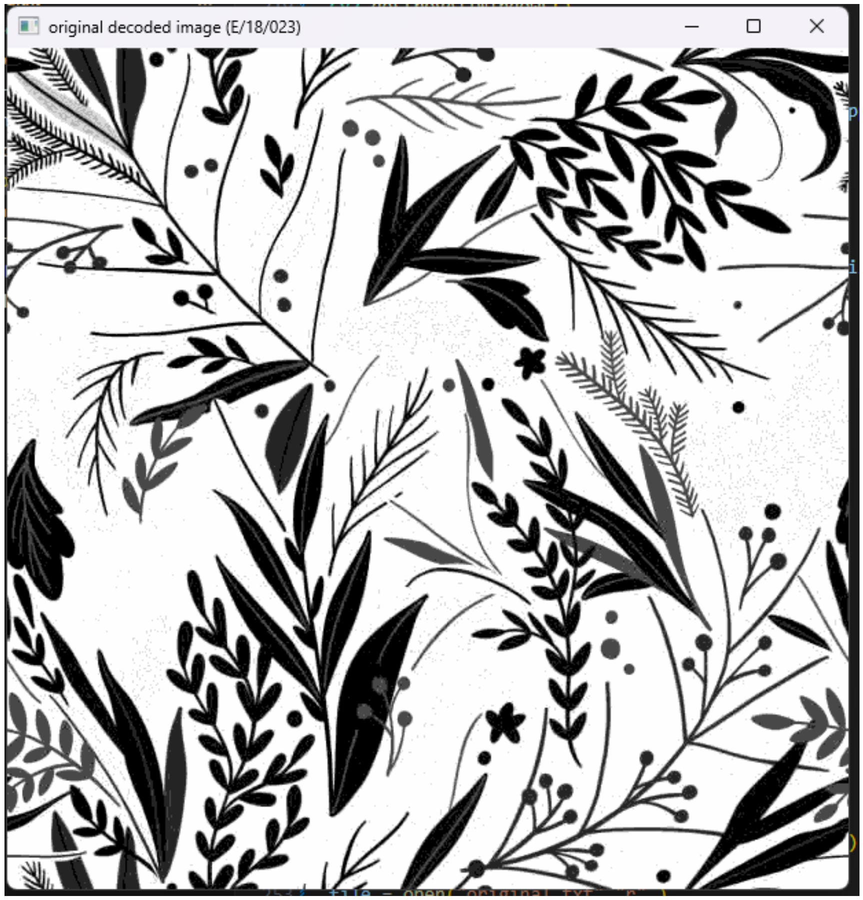


Figure 07 : Decompressed red-filtered image

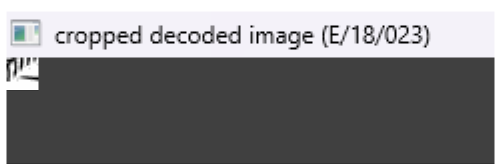
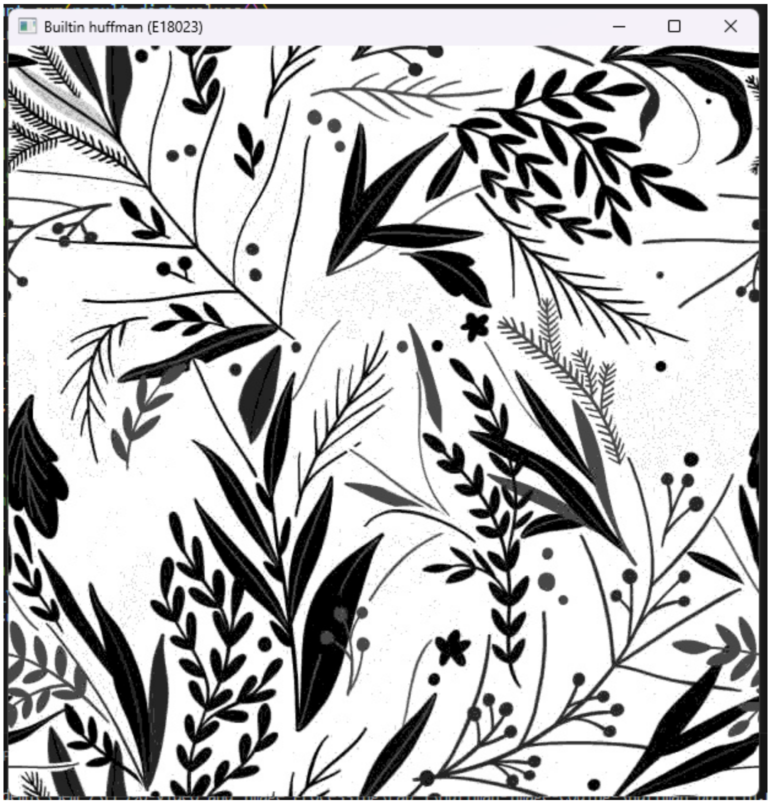


Figure 08 : Decompressed cropped image

Decompressed images for part 9 :



Step 11: Calculate the entropy of the Source

```
import numpy as np
import cv2
from scipy.stats import entropy


x_pos=0
y_pos=23*4

img2 = cv2.imread("Pattern.jpg")
red_img = img2[:,:,2]
cropped_image = red_img[y_pos:y_pos+16, x_pos:x_pos+16]

def find_entropy(image,name):


    # gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    _bins = 128

    hist, _ = np.histogram(image.ravel(), bins=_bins, range=(0, _bins))

    prob_dist = hist / hist.sum()
    image_entropy = entropy(prob_dist, base=2)
    print(f"{name} Entropy {image_entropy}")

find_entropy(img2,"original image")

find_entropy(cropped_image,"cropped image")

img1 = cv2.imread("original_decoded.jpg")

find_entropy(img1,"decompressed img")
```

```
original image Entropy 6.6988692982364695
cropped image Entropy 5.361373769664156
decompressed img Entropy 4.101455375140723
```

Figure 08 : Entropy

Step 12: Evaluate the PSNR of

```
from math import log10, sqrt
import cv2
import numpy as np

def PSNR(original, compressed):
    mse = np.mean((original - compressed) ** 2)
    if(mse == 0):
        return 100
    max_pixel = 255.0
    psnr = 20 * log10(max_pixel / sqrt(mse))
    return psnr

def main():
    original = cv2.imread("pattern.jpg")
    compressed = cv2.imread("original_decoded.jpg")
    value = PSNR(original, compressed)
    print(f"PSNR value is {value} dB")

if __name__ == "__main__":
    main()
```

i. The original images

ii. The decompressed images

```
PSNR value is 30.123443574851052 dB
```

DISCUSSION

1. Calculate the entropy of,
   i. The original image = 6.6988692982364695
   ii. The cropped image = 5.361373769664156
   iii. The decompressed images = 4.101455375140723

2. Calculate the average length of the cropped image.

   $\sum_{k=0}^{L-1} l_k(r_k)P_k(r_k)$ = (0.02734375 x 4) + (0.1171875 x 3) + (0.14453125 x 3) + (0.1015625 x 4) + (0.109375 x 3) + (0.5 x 1)
   = 2.12890625

3. Compare the performance of your algorithm and inbuilt algorithm of Matlab by comparing the compression ratios, for cropped and original images.

   $$\text{Compression Ratio} = \frac{\text{Total bits before compression}}{\text{Total bits after compression}}$$

   = 8/ 2.12890625
   = 3.7577981651376146788990825688073

4. Discuses about Entropy of the input image, the compression ratio achieved, and the output quality of the decompressed image

   The provided values offer insights into the performance of Huffman coding on the input image. The entropy value of 6.6988692982364695 indicates a notable degree of unpredictability in the pixel values, suggesting a diverse range of colors or intensities in the image. However, higher entropy may result in longer Huffman codes, potentially affecting the compression efficiency.

   The compression ratio of 3.7 implies that the compressed data is 3.7 times smaller than the original, showcasing a reasonable level of data reduction. Nonetheless, the PSNR value of 3.1 is concerning, indicating a significant loss in image quality during the compression-decompression process. This low PSNR suggests that the achieved compression ratio might have been attained at the expense of sacrificing important image details.

   The discrepancy between the high entropy and relatively low PSNR prompts a critical examination of the compression process. It raises questions about whether the compression algorithm adequately preserved the essential features of the image or if aggressive quantization or lossy compression techniques were employed. Striking a balance between compression efficiency and maintaining acceptable image quality is crucial for the overall effectiveness of the compression algorithm.

In conclusion, while Huffman coding effectively reduces redundancy in the input image, the observed low PSNR underscores the importance of optimizing the compression process. Exploring alternative compression algorithms or adjusting parameters may help strike a better balance between achieving a favorable compression ratio and preserving the quality of the decompressed image.

5. How can you improve the compression ratio of the given image? Discuss

To enhance the compression ratio of the given image, various strategies can be explored. Firstly, considering advanced compression algorithms like LZW or BWT, which may offer improved compression ratios compared to Huffman coding. Additionally, adopting transform coding techniques such as DCT or DWT, commonly used in standards like JPEG, can enhance compression efficiency. Fine-tuning Huffman coding parameters, including symbol representation and coding strategy, is another avenue for optimization. Lossy compression techniques, such as quantization, may be applied if acceptable for the application, intentionally trading some image quality for higher compression ratios. Entropy coding with variable-length codes and predictive coding techniques can also be considered to exploit redundancies and improve compression efficiency. Exploring hybrid compression schemes and evaluating compression parameters systematically are essential steps toward achieving an optimal balance between compression ratio and image quality, ensuring the effectiveness of the chosen compression strategy for the specific image characteristics.

REFERENCE

1. "Canonical Huffman Codes" by David A. Huffman, IEEE Transactions on Information Theory, vol. 25, no. 6, pp. 658-663, 1979.
   https://ieeexplore.ieee.org/document/8293507

2. “Huffman Coding" by Khalid Sayood, in Introduction to Data Compression, Morgan Kaufmann, pp. 131-158, 2005.
   https://www.sciencedirect.com/book/9780126208627/introduction-to-data-compression

3. "Huffman Coding" by Wikipedia, en.wikipedia.org/wiki/Huffman_coding.
   https://en.wikipedia.org/wiki/Huffman_coding