CS6351 October 13, 2022
Youssef

Homework 3
Due Date: November 3, 2022

a. **Problem 1**: (25 points)

Let $X = [0, 0.52, 0.55, 0.68, 0.91, 0.94, 0.97, 1.03, 1.04, 1.2, 1.3, 1.35, 1.4, 1.47, 1.6, 1.7, 1.85, 1.95, 1.99, 2.2 2.28, 2.45, 2.48, 2.56, 2.63, 2.67, 2.85, 3, 3.39, 3.57, 3.86, 3.99]$.

a. Find the decision levels (d's) and reconstruction values (r's) of the n-level optimal Max-Lloyd quantizer for X. (n-level means the quantizer has n intervals.) Assume that $d_0 = 0$ and $d_n = 4$. Show the values of the d's and r's after each iteration of the Max-Lloyd. Do this part for $n = 5, 6, 8$. For each value of n, call the resulting quantizer ML_n.

b. Quantize X with $ML_n$of part (a), then dequantize it. Show the quantized values as an array, and also the dequantized values $\hat{X}$ as an array. Compute MSE$(X, \hat{X})$. Do so for $n = 5, 6, 8$.

c. Do the same as in (b) except this time the quantizer is an n-level uniform quantizer (UQ_n).

d. Do the same as in (b) except this time the quantizer is an n-level optimal semi-uniform quantizer (SUQ_n). Recall that an optimal semi-uniform quantizer is a quantizer where all the intervals have the same length but the reconstruction values are optimal, i.e., the reconstruction value in each interval is the average of the data points in that interval.

e. Compare the three MSEs you obtained in (b-d). Present that in a table where the column headers are the different values of n ($i.e., 5, 6, 8$), and the row headers are the three different quantizer types (ML_n, UQ_n, SUQ_n).

Problem 2: (25 points)

This problem will require you to download this image from the course Website (River). Call the downloaded image file image.gif.

Remarks: Most images, which are in gif or other standard formats, are represented as an index matrix "*I*" coupled with a colormap (call it "map"). The color map is a 3-column table: every row consists of three components, representing one color. Thus, if *I*(*i*,*j*)=*k*, then the actual pixel value at (*i*,*j*) is the color represented in row *k* of map. This is true even if the image is a grayscale image. The arrangement of the rows (i.e., colors) in the map is random, that is, two very similar colors are not necessarily near each other in the map; in fact, they could be far apart. The implication of this image representation scheme is that none of the image compression processes (e.g., transforms, quantization, etc.) would give the expected results.

Remedy: To resolve this problem, do the following:

- [I,map]=imread('image.gif'); % or imread('image', 'format'); % format can be gif, tiff, etc.
- G=ind2gray(I,map);
- imagesc(I); colormap(map); % for displaying the color image I
- imagesc(G); colormap(gray); % for displaying the grayscale image G

Now the image in G is in the "right" representation. That is, you can run on G all the compression-related algorithms you have studied, and you will get the expected outcome.

a. Compute the entropy of the image.

b. Quantize G into G'_u using an 8-level uniform quantizer, compute the entropy of G'_u, dequantize G'_u into $\hat{G}_u$, display $\hat{G}_u$, and compute $SNR(G, \hat{G}_u)$. Note that you can take d_0=the min of G, and d_8=the max of G.

c. Repeat (b) except this time you should use an 8-level optimal semi-uniform quantizer. Call the dequantized image $\hat{G}_{su}$. (Derive the optimal semi-uniform quantizer before using it)

d. Repeat (b) except this time you should use an 8-level optimal Max-Lloyd quantizer. Call the dequantized image $\hat{G}_{op}$. (Derive the optimal Max-Lloyd quantizer before using it)

e. Compare the SNRs of the three schemes of (b-d).

Problem 3: (25 points)

One can design a clustering-based quantizer CQ_n, specified only by its reconstruction values $r_0, r_1, r_2, \dots, r_{n-1}$for some positive integer n, without needing the decision levels, as explained in this problem.

Quantization: To quantize a value x, find the closest reconstruction value r_k to x, and return the index of that reconstruction value. That is, $CQ(x) = k$ where $|x - r_k| \leq |x - r_j|$ for all $j = 0, 1, \dots, n-1$. In case of ties, select the smallest k among the ties.

Dequantization: If $CQ_n(x) = k$, then the dequantized value is r_k. That is, $DQ_n(k) = r_k$.

The clustering algorithm for finding the reconstruction values $r_0, r_1, r_2, \dots, r_{n-1}$: Suppose the data is an array $X(0: N-1)$, the reconstruction values are computed through the so-called *K-mean clustering algorithm*. Intuitively, the data array X will be divided into n clusters, where the elements within each cluster are closer to each other, whereas the elements of different clusters are farther apart. Then each r_k is the mean (i.e., average) of the k^{th}cluster. Let $C(i)$ denote the cluster of $X(i)$, for all $i = 0,1,2,\dots, N-1$. The K-mean algorithm is this iterative algorithm:

1. Initialization of the $r_0, r_1, r_2, \dots, r_{n-1}$: Let d =floor($\frac{N}{n}$). For k=0 to $n-1$, set $r_k = X[k.d]$; // other initialization methods can be applied, like random selection.
2. While the array $r_0, r_1, r_2, \dots, r_{n-1}$ continues to change do:
   2.1. Cluster assignment: For $i = 0 \; to \; N-1$, set $C(i) = k$ where $|X(i) - r_k| \leq |X(i) - r_j|$ for all $j = 0, 1, \dots, n-1$. That is, we put item $X(i)$ in cluster $C(i)$ as defined.
   2.2. Recomputation of r_k's: For k=0 to $n-1$, set r_k= the mean of all the elements of array X that are in cluster k.

a. For $n = 5, 6, 8$, compute $r_0, r_1, \dots, r_{n-1}$ for the data X of problem 1, then quantize X with this CQ_n, then dequantize it. Show the quantized and the dequantized values as a 4-column table where the 1st column is X, 2nd column is the dequantized data for $n = 5$, the 3rd column is the dequantized data for $n = 6$, and the 4th column is the dequantized data for $n = 8$.

b. Compute the three MSE's corresponding to $n = 5, 6, 8$.

c. Show in a single graph the plot of the MSE's you got here, and another plot of the MSE's you got in Problem 1(b) for ML_n, where the x-axis is $n = 5, 6, 8$.

Problem 4: (25 points)

In this problem you will compress the image G of problem 2 using 8 × 8 block-oriented DCT. First apply dct2 on each of the contiguous 8 × 8 blocks of G. The top leftmost term of each transformed block is called the *DC term*. Quantize all the DC terms of all the blocks as one data set, using a uniform 8-level quantizer. Afterwards, quantize the 14 elements in the 2nd, 3rd, 4th and 5th counter-diagonals of each block with a separate 4-level uniform quantizer, while the remaining 49 elements of each block are zeroed out. The range of each quantizer is from $\lfloor \min(Data) \rfloor$ to $\lceil \max(Data) \rceil$, where $Data$ is the set of data on which the quantizer is to be applied. Note that $\lfloor x \rfloor$ stands for floor(x) and $\lceil x \rceil$ stands for ceil(x), both provided in Matlab.

(See the red remarks at the end of this problem for hints on how to conveniently use block-oriented DCT in MATLAB.)

a. Compute the resulting bitrate and the compression ratio. (Note: you need not carry out any entropy coding beyond the quantization step. Also, assume that the info about the quantizers is implied and not stored.)

b. Reconstruct the image (into $\hat{G}_{15}$) by dequantizing and applying the inverse DCT on each separate block. Display the reconstructed image and compute the SNR.

c. Repeat (a) and (b) except that this time you also drop (zero out) the 5th counter-diagonal of each block. Call the reconstructed image $\hat{G}_{10}$.

d. Repeat (a) and (b) except that this time you also drop (zero out) the 4th counter-diagonal of each block. Call the reconstructed image $\hat{G}_{6}$.

e. Repeat (a) and (b) except that this time only the DC terms are quantized, while the remaining 63 terms of each block are zeroed out. Call the reconstructed image $\hat{G}_{1}$.

f. Compare the bitrates of the four schemes; also compare the SNRs of the four schemes; finally, compare the subjective (visual) quality of the four reconstructed images $\hat{G}_{15}$, $\hat{G}_{10}$, $\hat{G}_{6}$, and $\hat{G}_{1}$.

Remarks (MATLAB hints for applying block-oriented dct2/idct2):

- To apply 8 × 8 block-oriented DCT, make use of MATLAB's blockproc function on the input image, specify the block size, and turn dct2 into a block-structure function as follows:

```
dG = blockproc(G,[8 8],@(blkStruct) dct2(blkStruct.data));
```

- After you have processed dG (e.g., quantized and dequantized), and has become dGhat, you can apply block-oriented inverse DCT as follows:

```
Ghat= blockproc(dGhat,[8 8],@(blkStruct) idct2(blkStruct.data));
```