UNIVERSITY OF CALIFORNIA, SANTA BARBARA
Department of Electrical and Computer Engineering

ECE 178 **Image Processing** Fall 2020

Homework Assignment #8

(Due before Thursday 12/10/2020, 11:59 pm)

Problem # 1. Uniform quantizers:

a) A signal takes values in the range $[-1.5, 1.5]$. The range was divided into 4 quantization intervals for uniform quantization. Quantize the following sequence:

$$1.4, -1.1, 0.2, -0.8, 0.3, 1.2 \ldots$$

i.e., what is the sequence of quantized values?

b) Next, more generally, consider a signal taking value in the range $[-A, A]$. A uniform quantizer is designed by dividing the range into N quantization intervals. Using the high resolution approximation, find an expression for the mean square quantization error.

c) A misinformed engineer designed a uniform quantizer for the range $[-2A, 2A]$ instead. What will be the mean squared quantization error when this quantizer is applied to the signal of (b)?

Problem # 2. Entropy and Huffman code: A random variable takes value in a 5-letter alphabet $\mathcal{A} = \{a_0, a_1, a_2, a_3, a_4\}$, with probabilities: $p(a_0) = 0.1$, $p(a_1) = 0.07$, $p(a_2) = 0.71$, $p(a_3) = 0.09$, and $p(a_4) = 0.03$.

a) Calculate the entropy. (Here you need a calculator).

b) Design a Huffman code for this signal.

c) Find the code's average length $\bar{\ell}$, and compare to the entropy. The difference is the remaining redundancy after applying this code (would be 0 if we achieve entropy).

c) Decode the following bit string: 10100100111110110 . . .

Problem # 3. A variation on a familiar theme: Here we do some "manual" 2D-DCT work in the very simple setting of 2×2 images. Consider the images

$$f_1[m,n] = \begin{bmatrix} \boxed{2} & 2 \\ 2 & 2 \end{bmatrix} \qquad f_2[m,n] = \begin{bmatrix} \boxed{1} & 0 \\ 0 & 1 \end{bmatrix} \qquad \text{with the convention:}$$

$$\begin{tikzcd} {} \arrow[r] \arrow[d] & n \\ m & \end{tikzcd}$$

.

Of course, your favorite programming language may use a different convention but we are doing things manually here. The origin, $[0,0]$ entry, is "boxed".

a) Manually calculate each one of the basis functions, as a $2{\times}2$ matrix, (see lecture slides):

$$g_{k,\ell}[m,n] = \frac{2}{N}\beta[k]\beta[\ell] \cos\frac{(2m+1)k\pi}{2N} \cos\frac{(2n+1)\ell\pi}{2n}$$

It is OK for your result to include terms such as $\frac{\sqrt{2}}{2}$, no need to use a calculator.

b) Find the 2D-DCT of images f_1, f_2, i.e., find each transform coefficient (without using a computer program), and write out the matrices $F_1[k,\ell]$, $F_2[k,\ell]$ in the corresponding convention, with k the vertical coordinate and ℓ horizontal, and where $F[0,0]$ is at the top left corner. You may do so by straightforward derivation, or you may "cut through the red tape" and use insights from the interpretation of DCT as orthogonal transform.

c) Now write out the inverse 2D-DCT as a weighted sum of your basis functions, and verify that the weighted sum yields the original images f_1, f_2.

Problem # 4. *A hands-on problem:* In this problem you will compare the energy compaction achieved by the Discrete Fourier Transform (DFT) with that of the Discrete Cosine Transform (DCT), and hence their relative usefulness for image compression.
Given the image **ECE178_HW8.jpg**, sequentially extract non-overlapping 8×8 blocks (patches), and transform each block by DFT and then DCT. (Feel free to use built-in functions like fft2, dct2, np.fft, scipy.fftpack.dct; you will also find attached some "helper code" in Matlab and python). In essence, we perform the transform on the image on a per-block basis (which is common practice in image compression to save in transform complexity and to adapt to local statistics).

a) After performing the transform on a block, sort the coefficients in decreasing order of magnitude and retain only as many of the top (highest magnitude) coefficients such that they capture $K\%$ of the energy in that block (energy is the sum of coefficients' squared magnitude). Note that the number of coefficients needed will vary from block to block. Next, count the total number of coefficients retained in the entire image by DFT and by DCT, respectively, as measure of their energy compaction (fewer coefficients $\Longrightarrow$ more compaction). What is the ratio of number of DFT coefficients retained to number of DCT coefficients retained? Next reconstruct the image using the retained coefficients (inverse transform each block and then patch the blocks together to obtain the image) and comment if you notice any differences between the DFT-based and DCT-based reconstructions. Perform the above for 3 levels of retained energy: $K = 85, 90, 95\%$.

b) This part is similar to (a), except "in reverse". You will retain a specified **fixed** number (M) of top magnitude coefficients in each block, and consider how much energy was captured. Specifically, sum up the energy of the M retained coefficients over all blocks and divide by the total energy (sum of squares of all coefficients in all blocks). This is the portion of the image energy captured by "M out of 64" coefficients per block. What is the ratio of the energy portion retained by DFT to the portion retained by DCT? Recall that, here, both transforms retain the same number of coefficients. Next, reconstruct the image using the retained coefficients and comment if you notice any differences between the reconstruction using DFT and the one using DCT. Perform the above for $M = 8, 16, 32$. Based on your results, what conclusions can you draw about the energy compaction of these transforms?

c) (Bonus points) For each DFT coefficient we retain, we need to store two numbers (the real and imaginary part), so it seems that, for the same number of coefficients retained, DFT has double the storage footprint compared to DCT, whose coefficients are real. This suggests that our comparison above is unfair. Is that really true?

   Hint: The input image is always real valued, which may have some implications on its DFT domain representation.