ELEN 4810 Homework 4

Due **Tuesday, December 2**. Please submit your solutions via Courseworks.

Analytical Questions

Please complete problems 10.32, 3.37, 3.40, 3.48 in Oppenheim and Schafer (3rd Edition). Justify your answers!

Computational Questions

1 Lossy DCT Compression Encoding in JPEG

Many popular frequency transforms, such as the length-N DFT

$$X[k] = \sum_{n=0}^{N-1} x[n] \exp\left(-j\frac{2\pi kn}{N}\right) \tag{1}$$

are linear in the input $x[n]$, and are designed to operate on sequences of finite length. When working with applications involving real signals – such as images or audio – it is often preferred to use a frequency transform that stays within $\mathbb{R}^N$. A popular choice is the length-N *Discrete Cosine Transform* (DCT):

$$\mathrm{DCT}_N\{x\}[k] = \sum_{n=0}^{N-1} x[n] \cdot \underbrace{w_k \cos\left(\tfrac{\pi}{N}(n+\tfrac{1}{2})k\right)}_{k\text{-th vector } \mathbf{d}_k}, \tag{2}$$

$$w_k = \begin{cases} \sqrt{\frac{1}{N}}, & \text{if } k = 0, \\ \sqrt{\frac{2}{N}}, & \text{otherwise.} \end{cases}, \tag{3}$$

for $k = 0, \ldots, N-1$. This is again linear with respect to $x[n]$.

For images and other 2D signals, 2D equivalents of the DFT and DCT are needed, and frequencies have 2D interpretations based on the basis functions of the transform. That is to say, the 2D-DCT is a function of *two* frequency variables, which we will denote by k and l below. The size-(N, N) 2D-DCT is given by the formula

$$2\mathrm{DCT}_N\{X\}[k,l] = (w_k w_l) \cdot \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} X[m,n] \ \cos\left(\tfrac{\pi}{N}(n+\tfrac{1}{2})k\right) \ \cos\left(\tfrac{\pi}{N}(m+\tfrac{1}{2})l\right). \tag{4}$$

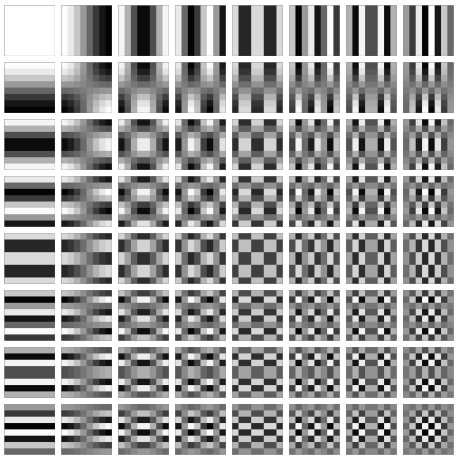

Figure 1. Basis images for the size-$(8, 8)$ 2D-DCT. As k, l increases, oscillations increase along their respective directions.

In (4), we can move the terms around and observe that

$$\begin{aligned} \text{2DCT}_N\{X\}[k,l] &= \text{DCT}_N^{(n)}\{\ \text{DCT}_N^{(m)}\{X\}\ \}[k,l] \\ &= \text{DCT}_N^{(m)}\{\ \text{DCT}_N^{(n)}\{X\}\ \}[k,l], \end{aligned}$$

That is, we can take the 2D-DCT by taking the DCT twice, once along the columns followed by once along the rows, or vice versa. Furthermore the order this is done doesn't matter. This is called the *seperability* property of the transform.

Motivation. For this problem we are going to explore the role of DCT in JPEG encoding. JPEG encoding is an extremely popular image compression technique due to its relative simplicity and large compression ratios: a 10 to 50 fold loss in image size can be achieved whilst retaining acceptable image quality. The JPEG pipeline is described in Figure 2.

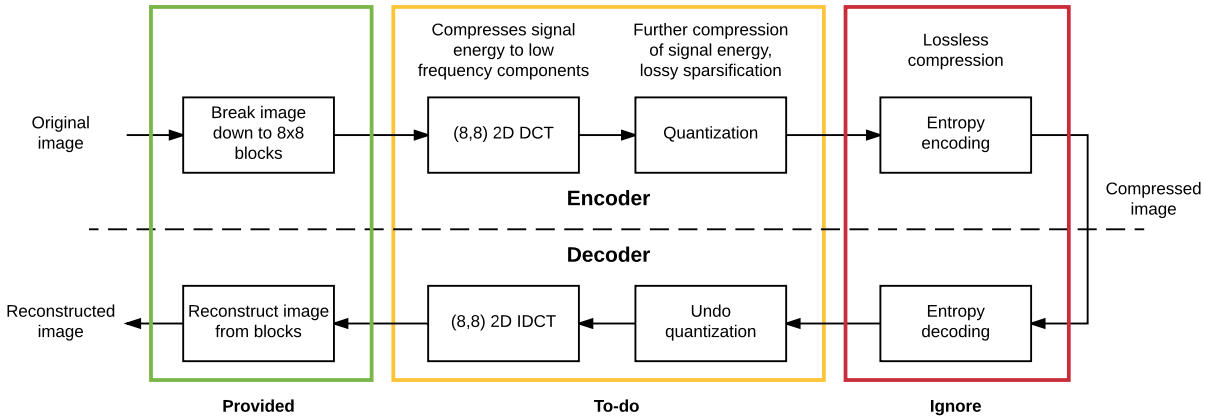

Figure 2. The JPEG pipeline. We will ignore entropy coding for this problem.

The image is first split up into patches of size 8×8, thus standardizing the application of the DCT to the $(8, 8)$ 2D-DCT for all incoming images. The DCT fulfills two functions. Firstly, the *energy* of natural images – or the squared magnitudes of its entries – mostly lie in low-frequency components, so DCT patches tend to aggregate signal energy to these entries.

Furthermore, he human eye, on average, also tends to be less sensitive to the differences in the high-frequency content in an image. This is conveniently exploited in JPEG by aggressively quantizing high-frequency entries in each DCT patch. Quantization is achieved dividing by each DCT patch entriwise by a *quantization matrix* $\mathbf{Q}$, rounding afterwards so the result can be stored digitally. In other words, for an 8×8 image patch $\mathbf{X}$

$$B_{ij} = \text{round}\big(G_{ij}/Q_{ij}\big), \tag{5}$$

where $\mathbf{G} = \texttt{dct2(X)}$ and $\mathbf{B}$ are 8×8 patches after DCT on the image patch $\mathbf{X}$ (after normalization, see To-do) and quantization of $\mathbf{G}$, respectively.

A common choice for the quantization matrix is

$$\mathbf{Q} = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 60 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 69 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 80 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 103 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 113 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 120 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 103 & 103 & 99 \end{bmatrix}, \tag{6}$$

the entries were determined subjectively based on human feedback. As a result the high-frequency entries being aggressively quantized, $\mathbf{B}$ tends to be *sparse*, i.e. most of its entries are zero.

Sparsity is a very attractive quality for signal compression. On an 8×8 patch with s nonzero entries. The contents of an image patch can in principle be stored using at most $2s$ rather than the full 64 values (since the location also needs to be stored).

For this problem, we will explore the DCT and quantization process by ignoring any further lossless compression techniques and other extraneous factors, and define the compression ratio achieved as

$$\text{compression ratio} = \frac{\text{total pixels in image}}{2 \cdot \text{total number of nonzeros}}. \tag{7}$$

To-do. Please work with the skeleton file `lossy.m` and make sure you have the file for the function `image2patches()` and `DCTQ.mat`, all of which are provided to you. The function `image2patches()` reads the specified image file, converts it to greyscale, then breaks it up into 8×8 patches. This process is taken care of for you in `lossy.m`, although you may try using different images.

The patches returned by `image2patches()` are stored in a cell array (you may need to look up how to use these); each cell contains an array representing a patch. Process each patch $\mathbf{X}^{(i)}$ by doing the following:

1. Apply the $(8, 8)$ 2D DCT to $(\mathbf{X}^{(i)} - 128)$ to get $\mathbf{G}^{(i)}$. You may use either matrix multiplications or `dct2()` to achieve this. The subtraction shifts the dynamic range from $[0, 255]$ to $[-128, 127]$, decreasing the magnitude of DCT values, and preventing the negative portion of the DCT from being wasted.

2. We will look at the average signal energy of each DCT coefficient over patches. Maintain an 8×8 array `avgenergy` by incrementing its entries so we end up with the following

$$\texttt{avgenergy(j,k)} = \frac{1}{\#\text{ of patches}} \sum_{i=1}^{\#\text{ of patches}} \left(G_{jk}^{(i)}\right)^2. \tag{8}$$

3. Quantize $\mathbf{G}^{(i)}$ by applying (5) to get $\mathbf{B}^{(i)}$. The quantization matrix should already be loaded for you from `DCTQ.mat` in the skeleton file.

4. We will keep track of the frequency that the entries $B_{jk}^{(i)}$ are nonzero after quantization. Maintain the 8×8 array `nzeros` so we end up with the following

$$\texttt{nzeros(j,k)} = \sum_{i=1}^{\#\text{ of patches}} \mathbf{1}\{B_{jk}^{(i)} \neq 0\}. \tag{9}$$

   Here $\mathbf{1}\{B_{jk}^{(i)} \neq 0\}$ just returns 1 if the entry is nonzero, and zero otherwise.

5. $\mathbf{B}^{(i)}$ now represents the encoded patch! The "size" of the resulting patch would be

$$2 \cdot \sum_{j,k} \mathbf{1}\{B_{jk}^{(i)} \neq 0\}.$$

   Decode $\mathbf{B}^{(i)}$ by multiplying it entrywise with $\mathbf{Q}$, then applying the inverse 2D DCT and adding 128. Again you may use matrix multiplication or `idct2()`.

   Put the reconstructed patch in the corresponding element in a cell array `pats_rec`.

The rest of the skeleton code will reconstruct the image from the patches, and display a set of statistics:

- A subplot display of the original and reconstructed images.
- The original image size and nonzero entries after quantization,
- The compression ratio defined in (7),
- The root mean squared error (RMSE) between the original and reconstructed images,
- The mean absolute deviation, or the average absolute difference between the pixels in the original and reconstructed images,
- The log average energy of the entries of $\mathbf{G}^{(i)}$, and
- The log frequency that the entries of $\mathbf{B}^{(i)}$ are nonzero.

See if you can make any observations from the statistics displayed (no need to submit observations).

Observations & discussion.

- (Optional) So far lossy compression and sparsity has only been achieved through the quantizer, which rounds small DCT values to zero. Modify your code so that it retains at most s nonzero entries for each DCT patch by keeping the entries with largest magnitude. This lower-bounds compression ratio:

$$\text{compression ratio} \geq \frac{\text{total number of pixels}}{2s}.$$

  Vary your choice of s, how does the quality of the recovered image and the compression statistics change?

- (Optional) Another way to modify the compression ratio is to change the quantization matrix: try scaling $\mathbf{Q}$ with a factor $\alpha > 0$ and observe any changes.

- In practice, the locations of the nonero entries are not stored directly. Rather entropy encoding techniques provide more efficient ways to encode the information obtained from $\mathbf{B}$, and can significantly increase the compression ratio.