Advanced Signal Processing Home Assignment #4: Image Compression

1 Introduction

In this exercise we shall implement a simple JPEG like image compressor. For simplicity we shall use it only on black and white image. The performance of the compressor in terms of rate vs. distortion will be examined and compared to Matlab's standard image compressor.

The the blocks of the encoder (compressor) are the following:

1. Image scaling
2. Blocking
3. DCT (Discrete-Cosine Transform)
4. Uniform quantization
5. DPCM for the DC coefficients
6. Zigzag ordering
7. Finding the last non-zero element in every block (after Zigzag ordering)
8. Entropy coding

At the decoder, the inverse operations should be implemented in reverse order as follows:

1. Entropy decoding
2. Using the output of entropy decoding to fill all non-zero elements. Putting zeros in all the rest
3. Inverse zigzag ordering
4. Inverse uniform quantization
5. Inverse DPCM for DC coefficients
6. IDCT
7. Deblocking
8. Image inverse scaling

The details are given in the following subsections.

1.1 Image files, numeric formats, scaling, blocking

- You will be given bitmap (`.bmp`) files to be used as input. The files contain images of 800×800 pixels, with 24 bits per pixel (i.e. three colors, eight bits per color). The bitmap files should be read using the following Matlab command

  ```
  A =imread(filename);
  ```

  Where the size of `A` is $800 \times 800 \times 3$, namely, one 800×800 matrix per color. Note that the type of the elements is `uint8`. In order to obtain one matrix, per single color, use

  ```
  BWimage = double(rgb2gray(imread(filename)));
  ```

  where `rgb2gray` reduces the image to grayscale, and which will and will change the variable type to `double`.

- In order to display the image use

  ```
  imshow(uint8(BWimage));
  ```

  Note that (unlike most of Matlab commands), the `imshow` command is sensitive to the type of its input. Therefore, the conversion to `uint8` is essential.

- Scale from $[0, 255]$ to $[-0.5, 127/256]$ by subtracting 128 and then dividing by 256.

- *Blocking* is done by dividing the image into blocks of 8×8 ($M = 8$). Namely, the scaled image matrix is of dimension 800×800, and the array of blocks is four dimensional of size $8 \times 8 \times 100 \times 100$. It can be initialized by the command

  ```
  ImBlocks = zeros(8,8,100,100);
  ```

  The content of `ImBlocks(:,:,i,j)` is an 8×8 matrix placed at row `i` and column `j`.

1.2 DCT/IDCT

Use Matlab commands `dct2` and `idct2` on every block. The input of the `dct2` command is the matrix A and its output is the matrix B, both of size $M \times N$. The input to output relation is:

$$B_{pq} = \alpha_p \beta_q \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{mn} \cos \frac{\pi(2m+1)p}{2M} \cos \frac{\pi(2n+1)q}{2N}, \tag{1}$$

where $0 \le p \le M-1$, $0 \le q \le N-1$. The scaling coefficients $\alpha_p \beta_q$ are given by:

$$\alpha_p = \begin{cases} \frac{1}{\sqrt{M}}, & p = 0 \\ \frac{2}{\sqrt{M}}, & 1 \le p \le M-1 \end{cases} \tag{2}$$

and

$$\beta_q = \begin{cases} \frac{1}{\sqrt{N}}, & q = 0 \\ \frac{2}{\sqrt{N}}, & 1 \le q \le N-1 \end{cases} . \tag{3}$$

The `idct2` command performs

$$A_{pq} = \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} \alpha_p \beta_q B_{pq} \cos \frac{\pi(2m+1)p}{2M} \cos \frac{\pi(2n+1)q}{2N}, \tag{4}$$

with the same α_p, β_q as before.

1.3 Quantization

Quantization is done by dividing by a constant δ and then rounding to the nearest integer. The output of the quantization is therefore an integer number. Inverse quantization is done by multiplying the integer by δ.

1.4 DPCM

This operation is elaborated in the lecture notes in Section 5.4. Use prediction within each row separately. No need to implement 2-D prediction here. The output of the DPCM quantizer of the j element at row i is

$$Y_{ij} = Q(X_{i,j} - \hat{X}_{i,j-1}) = Q(X_{i,j}) - Q(X_{i,j-1}) = \hat{X}_{i,j} - \hat{X}_{i,j-1} \tag{5}$$

where $Q(\cdot)$ is the output of a uniform qunatizer with parameter δ.
Note: We only use DPCM for the DC coefficients. Here $X_{i,j}$ refers only to the DC coefficient of the block at row i and column j.

1.5 Zigzag ordering

The zigzag ordering is depicted Figure 1. Note that the rows are numbered top-down (the uppermost row is 1), and the columns are numbered from left to right. The numbers in the figure correspond to the numbering of the elements in Matlab matrices, which goes column-wise. For example, the element in the third row, second column is numbered 11. The blue curve corresponds to the zigzag ordering.

The zigzag ordering (and its inverse) is given in the vector `ZigZagOrd` (and `ZigZagOrdInv`) found in the file `ZigZagOrd.mat`. The first five elements in the vector are $1, 9, 2, 3, 10$ (make sure you understand why they correspond to Figure 1).

1.6 Entropy coding

After the zigzag ordering, a number per every block should be recorded, containing the index of the last non-zero element in it. If the entire block contains only zeros, this number is set to one. This can be implemented using the following code lines, where `current_block` is a vector of 64 elements containing the elements of a 8×8 block **after zigzag ordering**:

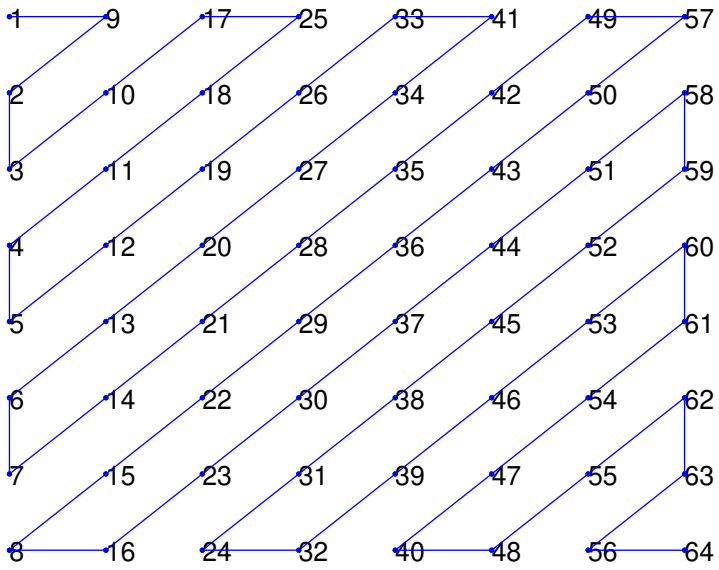

Figure 1: Zigzag ordering

```
current_lastNZ = find(current_block,1,'last');
if isempty(current_lastNZ)
current_lastNZ = 1;
end
```

For each block, `current_lastNZ` can then be turned into a string of bits (6 bits per block) as follows:

```
bitsoflastNZ = dec2bin(current_lastNZ-1,6);
```

A large bit stream containing the last non-zero indices of all blocks one after the other, denoted here by `lastNZ_bit_stream`, should be created. As the size of each `bitsoflastNZ` element is 6, we know prior to the encoding that the size of `lastNZ_bit_stream` is $60,000$. Hence, for each block examined:

```
lastNZ_bit_stream(1+6*(index-1):6*index) = bitsoflastNZ;
```

where `index` corresponds to the index of the currently examined block, with respect to the 100×100 blocks (in order to use a **single** index while sweeping over a 2D array, you may use the MATLAB command `sub2ind`).

The entropy coding is carried out using two functions for the Exponential-Golomb code

- `golomb_enc.m` and `golomb_dec.m`, for the encoder and decoder respectively. These functions are given to you in this assignment.

Encoder: In every block, the values until the last non-zero element (and **including**) should be fed to the entropy encoder (the rest of the zero elements will be decoded as zeros in the decoder). Namely, **for each block**, the function `golomb_enc` should be used only on the indices `1` to `current_lastNZ` of the block, where `current_lastNZ` corresponds to the last non-zero element of the block being currenly examined. Accordingly, for each block, the `golomb_enc` function in MATLAB should be used as follows:

```
current_Gol_bitstream = golomb_enc(current_block(1:current_lastNZ));
```

where `current_block(1:current_lastNZ)` is a vector of integers, and `current_Gol_bitstream` is a bit stream. The bit stream `current_Gol_bitstream` should be saved in a larger bit stream, denoted here by `Gol_bitstream`, that contains all `current_Gol_bitstream` generated for all 100×100 blocks, one after the other. As the size of each `current_Gol_bitstream` depends on the pixels' content and is thus unknown prior to the encoding, the size of `Gol_bitstream` is also unknown prior to the encoding. In order to overcome this issue and create `Gol_bitstream`, we use the function `strings` in MATLAB:

```
Gol_bitstream = strings(1,100x100);
```

Hence, for each block examined:

```
Gol_bitstream(index) = current_Gol_bitstream;
```

where `index` corresponds to the index of the currently examined block with respect to the 100×100 blocks.

Finally, the encoder outputs a total bit stream that is composed of `Gol_bitstream` and `lastNZ_bit_stream`:

```
Encoder_Output_Bit_Stream = [Gol_bitstream lastNZ_bit_stream];
```

This illustrates the image compression process: we compressed a 2D picture into a single bit stream. In the assignment, we'll see that as the quantization parameter δ is larger, the image is compressed more 'aggressively', such that the `Encoder_Output_Bit_Stream` is shorter and the decoded image will appear more distorted.

Note: The `Encoder_Output_Bit_Stream` could also be implemented by saving for each block `bitsoflastNZ` after its corresponding `current_Gol_bitstream`.

Decoder: The corresponding decoder is given in `golomb_dec.m`, and operates as follows:

```
[cursymbol, nextstartindex] = golomb_dec(bitstream,startindex);
```

The decoder outputs a **single** integer at every call, decoded from the bits in `bitstream` that start at the index `startindex`. Its output is the decoded symbol `cursymbol` and the start index of the next symbol, `nextstartindex`. Clearly, using this function to decode all the symbols from a single block's encoded bit stream (namley, `current_Gol_bitstream` defined above) requires a **loop**:

```
for symbol=1:current_lastNZ
    [cursymbol, nextstratindex] = golomb_dec(current_Gol_bitstream, stratindex);
    stratindex = nextstratindex;
end
```

Note that for `Gol_bitstream` and `lastNZ_bit_stream` as defined above, an outer loop over all blocks, sweeping on the `current_lastNZ` of each block, and an inner loop of the form `'for symbol=1:current_lastNZ'` (as in the previous paragraph), should suffice to reconstruct all symbols for all blocks from `Gol_bitstream`, where `stratindex` is initalzied to `1` before the outer loop.

1.7 Performance Measure - PSNR

Assume that the input image matrix is X and the output is $\hat{X}$ with B rows and columns. The mean squared error is naturally defined

$$\mathrm{MSE} = \frac{1}{B^2} \sum_{n=1}^{B} \sum_{m=1}^{B} \left(X_{nm} - \hat{X}_{nm} \right)^2. \tag{6}$$

The performance measure we shall use is the *peak signal-to-noise ratio* (PSNR). The peak is the maximal value (255) and the PSNR is dB scale is defined as:

$$\mathrm{PSNR}_{\mathrm{dB}} = 10 \log_{10} \frac{255}{\sqrt{\mathrm{MSE}}}. \tag{7}$$

Question 1 - JPEG Compressor

Implement the JPEG like image compressor, according to the instructions above. Submit two Matlab functions, one for the encoder and one for the decoder, along with explanations on how they should be used (that is, what are the input variables and output variables of each function). Be sure that your MATLAB submission in this paragraph is correct: Note that in order for your functions to be properly tested after submission, they have to be in separate MATLAB files, which have exactly the title of the function that will be called during the test. Namely, for this paragraph, submit two MATLAB files whose titles are 'imageEncoder' and 'imageDecoder', which contain only the function '[outputBitStream] = imageEncoder(image, delta)' and '[decodedImage] = imageDecoder(inputBitStream, delta)', respectively. Make sure that you follow these steps, as otherwise your function will not be tested!

Notes on implementation

- Instead of writing all the encoder and then all the decoder, it is better to write them incrementally block by block. For example, start with an encoder containing only image scaling, and a decoder inverse scaling and see the image is reconstructed without error. Then add blocking/deblocking etc.
- Make sure that a substantial distortion (other than numeric errors inflicted by DCT/IDCT) happens only after the quantization.
- Make sure that all blocks after the quantization (zigzag, entropy coding etc.) do not add distortion

You are also asked to evaluate the performance of your encoder and decoder on four image files that are attached:

`cat800x800.bmp`
`flower800x800.bmp`
`zakir800x800.bmp`
`shades800x800.bmp`

Answer the following questions for each of the four attached image files.

a. Use the following values of δ: $(0.5, 0.1, 0.01, 0.001)$. For each one of the calculate PSNR, and the size of the description in bits-per-pixel (i.e. count the total number of bits the encoder outputs, and divide by the number of pixels 800×800). Note that before compression, 8 bits-per-pixel were used, corresponding to the integers $0, 1, \ldots, 255$.

b. Attach the image corresponding to the lowest resolution ($\delta = 0.5$). Can you see the distortion?

c. Comparison to JPEG benchmarks. Plot a figure containing three curves for every image file:

   (1) PSNR vs. bits-per-pixel of the Matlab's embedded compression engine (explained in the sequel).

(2) PSNR vs. bits-per-pixel of your Matlab compression.

(3) PSNR vs. bits-per-pixel of your Matlab when the DPCM for the DC coefficients is disabled, i.e., when step 5 in the encoder and decoder is omitted. (In the code you submit, DPCM for the DC coefficients should be enables. Only disable it for generating this curve.)

The performance of the Matlab embedded JPEG compression engine is given in the file `JPEGbenchmark.mat` which contains the following six vectors:

```
catPSNRvecJpeg     flowerPSNRvecJpeg  zakirPSNRvecJpeg   shadesPSNRvecJpeg
catSizevecJpeg     flowerSizevecJpeg  zakirSizevecJpeg   shadesSizevecJpeg
```

These vectors contain the PSNR and the bits-per-pixel corresponding to every image, for various JPEG qualities. These vector where built by activating the Matlab command

```
imwrite(A,outfilename,'jpg','quality',qualityFactor);
```

where `A` is the image matrix with values between 0 to 255 and `qualityFactor` is the quality factor running from 10 to 100.

d. (bonus) Improving the results: Students who manage to improve the results of the basic compressor described in this exercise might get a bonus to their grade on this exercise. In order to qualify the following should be handed in:

(a) The code of the improved encoder/decoder with explanations on how to run it.

(b) Explanation on what improvements you have incorporated with respect to what was described in the exercise. The bonus points will depend on the depth of the proposed improvement.

A List of attached files

Fie Name	Content
`ZigZagOrd.mat`	the zigzag ordering
`golomb_enc.m`	Exponential Golomb encoder
`golomb_dec.m`	Exponential Golomb decoder
`cat800x800.bmp`	a test image file
`flower800x800.bmp`	a test image file
`zakir800x800.bmp`	a test image file
`shades800x800.bmp`	a test image file
`JPEGbenchmark.mat`	the benchmarks of Matlab JPEG engine for all images