COMP.SGN.210 Signal Compression Project

Deadlines: Early: Monday 29.04.2024, 18:00. Latest:Wednesday 15.05.2024, 18:00

Upload the PDF report and the Matlab code to Moodle Topic 11
Questions can be asked preferably during the exercise sessions Thursdays 14:15-16:00, but also during the other interactive sessions, on Mondays and Thursday. All questions received by e-mail will be answered only during the interactive sessions.

March 14, 2024

1 Main project, max 6 points

The input files used to test the lossless coding methods in this project work have names of the form **Image_X.png**, where X is a number from 0 to 4. If your student number is ID, compute X as $X = \text{ID} \bmod 5$ and use the input gray image **Image_X.png** for your project work and report.The files can be found in the archive saved in moodle at the project item.

Project Tasks:

(1) (0.5 points) Read the gray image using the **imread** MATLAB function. Save the gray values that represent the image in a matrix A. Convert matrix A from the *uint8* representation to a *double* representation.

Report: 1.1. The plot of the histogram for the image

(2) (0.5 points) Select from matrix A the gray values between the rows 51 and 306, and between the columns 51 and 306, and save them in a matrix denoted B, having $n_r = 256$ rows and $n_c = 256$ columns. Write the new image in a file **MyImage_X.png** using the **imwrite** MATLAB function and find the size of the file MyImage_X.png on the disk (in matlab use: sd = dir('MyImage_X.png') function and check the sd.byte field). Draw the histogram of all the sample values found in the image (use the **histc**(B(:),0:255) function). Use the collected statistics to compute the empirical entropy corresponding to the probability distribution obtained from histogram.

Report: 2.1. The size of the file **MyImage_X.png**.2.2. The value of the entropy corresponding to the empirical probability.

(3) (1 point) For decorrelating the samples, the Martucci predictor, also known as Median Adaptive Predictor (MAP), is used to estimate each element of the B matrix, based on the data available when scanning the image row-by-row. For example, for the element $B(i,j)$ found at the current pixel position (i,j) in matrix B, we compute an estimate using the values found in the northern position, $(i-1,j)$, western position, $(i,j-1)$, and northwestern position, $(i-1,j-1)$, in the causal neighborhood of the current pixel. If we denote $y_N = B(i-1,j)$, $y_W = B(i,j-1)$, $y_{NW} = B(i-1,j-1)$, then MAP estimates $B(i,j)$ as

$$\hat{y}_{MAP}(i,j) = median\{y_N, y_W, y_N + y_W - y_{NW}\}. \tag{1}$$

which selects the median value of three possible predictors: pixel above, y_N; pixel on the left, y_W; and the planar prediction $y_N + y_W - y_{NW}$. When some of these three predictors cannot be computed, use inside the median function only the predictors that can be computed. (Note that for some pixel positions (i, j) the causal neighbourhood has only two, or one, or none neighbour pixels). If the argument of the median in (1) is empty, take $\hat{y}_{MAP}(i, j) = 128$. Apply rounding if the result of median operation is a fractional number.

Define the prediction error matrix E, (also called prediction residual matrix) having as elements:

$$E(i, j) = B(i, j) - \hat{y}_{MAP}(i, j), \; i = 1, 2, \ldots, nr; \; j = 1, 2, \ldots, nc. \tag{2}$$

where i is the row index, j is the column index, and the size of B is $nr \times nc$. Note that the possible values of $E(i, j)$ are all integers belonging to the set $\{-255, \ldots, 255\}$.

Concatenate all the columns $E_j = E(:, j), \; j = 1, 2, \ldots, nc$ of the prediction error matrix E into a prediction error vector e_{vec} of length $nr \cdot nc$ as

$$e_{vec} = \; [E_1^T \quad E_2^T \quad \cdots \quad E_{nc}^T]^T. \tag{3}$$

Report: 3.1. Display the prediction error matrix E using the **imshow** or **imagesc** MATLAB function and save it to your report (you might need some shifting/scaling to show the possible negative elements of E as well).

(4) (1 point) The prediction error integer samples $e(1), \ldots, e(n)$, in the prediction error vector e_{vec}, must be losslessly encoded. The prediction residual values are assumed to be uncorrelated (as a result of the previous prediction step) and therefore each value may be encoded independently.

We will use for compression the class of codes known as Golomb-Rice codes [1], which encode an integer $i \geq 0$, first by using the unary code for the quotient $\lfloor \frac{i}{2^p} \rfloor$ and then using the fixed number of p bits to represent the reminder (i mod 2^p). These class of codes are parametrized by an integer parameter $p \geq 0$. Since we need to encode the residuals $E(i, j)$, which can be both negative and positive, the signed integer value $E(i, j)$ is transmitted as follows: $abs(E(i, j))$ is first encoded using GR code, and then, if $abs(E(i, j)) \neq 0$, the sign bit is encoded: ($b = 1$; if $E(i, j) < 0$; and $b = 0$ if $E(i, j) > 0$).

For example, if we found the optimal GR parameter $p = 3$, and we want to encode $x = 36 = 100100$, we transmit 36 as follows: the $p = 3$ least significant bits (i.e. 100 are appended to the bitstream (since 36 mod $2^3 = 4$); then the quotient ($q = \lfloor 36/2^3 \rfloor = 4$) is encoded in unary, as the sequence 00001, and is appended to the bitstream (we need to encode in unary any value of $q \in \{0, 1, 2, \ldots\}$, so we list q bits of zero, followed by a 1). Finally since the number x is non-zero, we transmit its sign bit (in this case we append a 0 to the bitstream, signalling that x is positive). In total we appended 100000010 to the bitstream for transmitting $x = 36$.

The codeword length for coding an integer x is hence $p + (\lfloor |x|/2^p \rfloor + 1) + delta(x)$, where $delta(x) = (x \neq 0)$ (in matlab notation, since transmitting the sign is necessary only if x==0). Given a block of values, $x_1, \ldots, x_n$ we compute the codelength estimation for the block as the sum of codeword length of each integer x_i.

Write a MATLAB function **GR_estimation.m** which takes as input arguments: a) a linear array of signed integer values (e.g. the vector $X = [x_1, \ldots, x_n]$), and b) the value p of the GR parameter and generates at the output the total codelength, computed as $CL(X, p) = \sum_{i=1}^{n} (p + (\lfloor |x_i|/2^p \rfloor + 1) + delta(x_i))$. You should add in the end a 4 to

$CL(X,p)$ to account for the need to transmit to the decoder the value of p that it needs to use.

Find the optimum parameter p^* by running the **GR_estimation** function with different p values, and in the end picking that p which gives the minimum codelength $CL(X,p^*)$ for the block.

Write a MATLAB function **GR_encode.m** which takes as input $X = [x_1, \dots, x_n]$ and p (here $p = p^*$, obtained by calling several times **GR_estimation.m**) and then it encodes X by generating a variable BITSTREAM. The BITSTREAM is initialized with the bits for the parameter p (If we know p is smaller than 15, we use bitget(x, 4:-1:1) for getting its bits, and then we store them in the vector BITSTREAM). Then we start appending to BITSTREAM the bits necessary for encoding each x_i: its GR code of parameter p and in the case of $x_i \neq 0$, also appends its sign $b(x_i)$. We append these codes one by one for all $x_1, \dots, x_n$. After creating the BITSTREAM variable, write it in a file to the disk, using the hint below.

(Hint) To generate the BITSTREAM use the following constructs: To generate a vector of k zero bits followed by a one bit we can use the MATLAB construct **[zeros(1, k) 1]**. For any number x the corresponding binary vector with the bits of x on positions from k to 1, can be obtained with the MATLAB construct **bitget(x, k:-1:1)**.

For compression, the bits should be written into a vector of bits BITSTREAM and finally saved to the binary compressed file named 'BITSTREAM_FILE.bin'. The predefined function **fwrite** should be used with the 'ubit1' precision specifier, like **fwrite(fid, BITSTREAM, 'ubit1')** to write the bit vector **BITSTREAM**.

(5) (1 point) Write a decoder which reads the BITSTREAM from the file 'BITSTREAM_FILE.bin' created at (4), and based on its syntax it reconstructs the sequence of prediction errors e_{vec}.

(6) (1 point) Segment the prediction error vector e_{vec} into blocks of length L. Write a function **Segment_L** having as inputs L and e_{vec}, and which estimates the codelength for encoding the vector e_{vec} as the sum of all block codelengths. For each block, the codelength is obtained by running the Golomb-Rice algorithm with various GR parameters, $p \in \{0, 1, \dots, 8\}$ (use here **GR_estimation.m**) and pick for each segment the best parameter p^*. (Each p^* value can be transmitted "raw", i.e. using the number of bits needed to encode the maximum value p^*, say 4 bits.) Compute the estimated codelength for encoding the vector e_{vec} as the sum of all block codelengths and return that value **CL = Segment_L(L,e_vec)**

Call the function **CLi = Segment_L(L,e_vec)** for 40 block-length values, $L \in \{50, 100, 150, \dots, 2000\}$. Find the block size that produces the smallest codelength and use it to finally encode the vector e_{vec}.

Compute *bpp_CL* using CLi by dividing each value in CLi with the number of pixels existing in the matrix B. Hence, *bpp_CL* contains the number of bits per pixel needed to compress the matrix B.

Report: 6. Include in the report the plot of *bpp_CLi* versus L and comment on the best L for encoding the prediction errors e_{vec} .

(7) (1 point) Write a top level MATLAB function, named **image_compress**, which takes as input arguments the name of the input file (initial image) and the name of the compressed image, and produces the compressed file according to the procedure discussed at Steps 1-5.

Include also a function that takes the vector of prediction errors e_{vec} reconstructed by the function at Step 5, and then it reconstructs the initial image B, by running the same prediction process as the encoder. At each pixel (i, j) we reconstructed alread a part of B_{rec} and we can construct the same prediction $\hat{y}_{MAP}(i, j)$ as the encoder using B_{rec}; then we add the decoded $E(i, j)$ and so we obtain $B_{rec}(i, j)$. We continue this process of reconstructing on the fly $B_{rec}(i, j + 1)$ and continue in the normal scanning order, of "row-by-row".

You must load to Moodle, at project's item, a ZIP archive including the report in .pdf format and the commented MATLAB code used for solving the project (including at least one .m file called **image_compress.m** which may be called independently as described in Step 6) no later than the date indicated on the web page. The report should contain at least the following information:

- A figure with the histogram obtained at point (1) and the computed empirical entropy.
- A figure with the prediction error matrix E. What image features are emphasized in this image?
- Describe the way you predicted the first column and the first row in matrix B.
- A plot of vector bpp_CL, when the x axis is the block size L.
- Comment the results. How would you modify the algorithm to improve the results?

2 Bonus task, additional 7 points

(1) (1 point) Prediction error coding using Huffman dictionary.

See lecture notes chapter 2 for the Huffman code and algorithm [1].

Replace the Golomb-Rice coding in tasks (4) and (5) with Huffman coding. Use Matlab's **huffmandict()** to design the prefix code for the symbols in the prediction error vector e_{vec}. Use Matlab's **huffmanenco()** to encode the prediction error vector. Verify that the encoding of the prediction error is lossless using **huffmandeco()**.

Compute the code length of the Huffman encoded prediction error e_{vec}, and account also for the required code length of the designed Huffman dictionary.

Compare the code length of the Huffman encoded prediction error against the code length achieved using Golomb-Rice coding of the prediction error in tasks (4) and (5).

Compare the code length of the Huffman encoded prediction error against the theoretical code length achieved by using entropy as the average code length per symbol.

(2) (1 point) Use the matlab imwrite to encode the block of the image B, using the lossless formats provided as options in imwrite (at least .png, .jp2, .jpeg), Be sure that you call the function imwrite with the paramter specifying lossless compression. Compare the results.

(3) (2 points) Instead of encoding the matrix E by the vectorized form e_{vec}, organize the segmentation in blocks of $k \times k$ pixels (and then vectorize each such square block). Find the optimal size of k when the image is split uniformly into such $k \times k$ pixel blocks of the same size. Study each possible choice, $k = 8, 16, 32, 64, 128, 256$ and find the optimum.

(4) (3 points) Then consider the possible *non-uniform split* into blocks of $k \times k$ with $k = 8, 16, 32, 64, 128, 256$, by using a hierarchical split: Initially we consider the image itself, a 256×256 block. Then split the image into 4 blocks of 128×128 pixels, and compare the codelength of the 256×256 block against codelength of 4 blocks of 128×128 pixels and decide to keep the split or not. If the split has smaller codelength, keep the split into blocks. Then take each block 128×128 and compare against its split into 4 blocks 64×64. Stop the split tests at the level 8×8 blocks and collect all split decisions into a "quadtree", which needs to be transmitted at the decoder (similar to the sending of a binary tree discussed in Lecture 1).

References

[1] I. Tabus. COMP.SGN.210 Signal Compression, course slides available in Moodle.