COMP.SGN.210 Signal Compression Project

Deadline: Monday 10.05.2021, 18:00

Upload the PDF report and the Matlab code to Moodle
Questions can be asked during the exercise sessions or by email to emre.kaya@tuni.fi

March 18, 2021

1 Main project, max 6 points

The input files used to test the lossless coding methods in this project work have a filename **Image_X.png**, where X is a number from 0 to 4. If your student number is ID, compute X as $X = \text{ID mod } 5$ 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. Draw the histogram of all the sample values found in the image (you can use the **histogram** function). Use the collected statistics to compute the empirical entropy corresponding to the probability distribution obtained from histogram.

(2) (0.5 points) Select from matrix A the gray values between the rows 50 and 249, and between the columns 50 and 249, and save them in a matrix denoted B, having $n_r = 200$ rows and $n_c = 200$ columns. Write the new image in a file **MyImage_X.png** using the **imwrite** MATLAB function.

(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

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

Note that for some pixel positions the causal neighborhood has only one neighbor pixel and one need to use the median from only the available elements of the set $\{y_N, y_W, y_N + y_W - y_{NW}\}$. Apply rounding if the result of median 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 $nr \times nc$. For the first column, $j = 1$, set $\hat{y}_{MAP}(i,1) = 0, \forall i$. Notice that the values of $E(i,j)$ are integer, and they belong to the set $\{-255,\ldots,255\}$.

Display the prediction error matrix E using the **imshow** or **imagesc** MATLAB function and save it to your report (check that you show the negative elements of E as well).

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 of length $nr \cdot nc$ as

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

(4) (1 point) The prediction error integer samples $e(n)$, in the prediction error vector e, 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 coded 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 ($i \bmod 2^p$). These class of codes are parametrized by an integer parameter $p \geq 0$. For a signed residual $E(i,j)$, each signed integer value is split into the sign, the p least significant bits, and the remaining most significant bits. The sign bit and the p least significant bits are coded raw, and the value represented by the most significant bits is coded in unary.

For example, if we have $p = 3$ and we want to code $x = 35$, we split 35 into the sign bit (0 means positive and 1 negative), the 3 least significant bits ($35 \bmod 2^3 = 3$) and the most significant bits ($\lfloor 35/2^3 \rfloor = 4$). The codeword associated to the value will therefore be 0 (sign), 011 (least significant bits), and 00001 (4 in unary).

The codeword length for coding an integer x is hence $1 + p + (\lfloor |x|/2^p \rfloor + 1)$. Given a block of values, compute the codelength estimation of the block as the sum of codeword length of each integer x.

Write a MATLAB function **GR_estimation.m** which takes as input arguments: a) a linear array of signed integer values (e.g. the vector e), and b) the value of the p parameter. Find the optimum parameter p^* by running the **GR_estimation** function with different p values.

Create a file (or more) to store the elements generated by the GR code (sign, the p least significant bits, and the most significant bit). For example, one way is to encode the signs of the errors, $e(n)$, to one file, and use a different file to store the bitstream produced by the GR code for the absolute values of the prediction errors.

(5) (1 point) Write a decoder which reads from the files created at (4) the signs of prediction error, $e(n)$, and the GR bitstream, and based on these reconstructs the sequence of prediction errors.

(6) (1 point) Segment the prediction error vector e into blocks of length L. Estimate for each block the codelength obtained using the Golomb-Rice algorithm. Compute a codelength estimation for transmitting to the decoder the optimal parameters p^* obtained for each block. (Each p^* value can be transmitted "raw", i.e. using the number of bits needed

to encode the maximum value p^*.) Compute the codelength estimation for encoding the vector e as the sum of all block codelengths plus the codelength for encoding the optimal parameters.

Obtain a vector CL of 40 codelength estimations, each computed using a block of length $L \in \{50, 100, 150, \ldots, 2000\}$. Find the block size that produces the smallest codelength and use it to finally encode the vector e.

Compute a vector *bpp_CL* using the CL vector by dividing each value in CL 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.

(Hint) 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 saved to a binary compressed file. The bits can be written to a binary compressed file and read from it with the built-in functions **fwrite()** and **fread()** using the ubit1 precision specifier. Say BS is a double or uint array with size (1 x number of bits), following commands work in most computers:

ENCODER:

```
fileID = fopen('binary.bin', 'w');
fwrite(fileID, BS, 'ubit1');
fclose(fileID);
```

DECODER:

```
fileID = fopen('binary.bin', 'r');
BS = fread(fileID, inf, 'ubit1');
fclose(fileID);
```

You can compare the size of the file 'binary.bin' with the length of your bitstream BS to check if it writes correctly. MATLAB allocates complete bytes for the filesize so even if your bitstream has a length of 57 bits, the filesize will be 8 bytes.

(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.

You must send by email 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 can you see in this image?
- Describe the way you predicted the first column and the first row in matrix B.
- Describe the way you encoded the first element in vector e.
- A plot of vector *bpp_CL*, when the x axis is the block size.

- The compressed size of the image **MyImage_X.png**, obtained using the segmentation of the vector e into blocks of optimal size, and the compressed size obtained by applying the Golomb-Rice algorithm directly to the whole vector e.
- Comment the results. How would you modify the algorithm to improve the results?

2 Bonus task, max 2 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. 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, 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 parameter specifying lossless compression. You can check the size of a file using dir command in MATLAB.

References

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