

EE 669 Multimedia Data Compression Homework #2 Report

Problem 1: QM Coder

Part A: Encode and Decode Using the QM coder

(1) Implementation of QM coder:

- a) Use StartQM() and reset() function to do the initialization, it will set all the variables to their correct values for the QM encoder.
- b) Read input file by characters until end of file and then loop through all 8 bits of each character. Each time call the function encode(symbol, context) to encode each symbol.
- c) Use the function Flush() to write to the output file and then use the function Counting() to calculate the size of the compressed file.
- d) The mapping scheme I used for QM coding is that read in input file character by character, then deal with each bit at a time. Doing it his way, we are always dealing with “0” or “1”. This is how I map multiple symbols into the 2 symbols “0” and “1”.

For detail information, please refer to the source code.

(2) Results for the four test media files:

	binary.dat	audio.dat	image.dat	text.dat
Original filesize	65536	65536	65536	8358
Compressed Filesize	6793	66711	67415	8609
Compressed Filesize	89.63%	-1.77%	-2.82%	-3.00%

(3) Discussion:

As we can see from the above table of the sizes of the compressed files, only the binary file achieves in compression. The other three files increase in size. This is reasonable for the QM compression since it works best in the cases when there are long runs of MPS. The result that audio.dat, image.dat and text.dat did not benefit from the QM Compression tells us that these three types of data have random symbol sets which make the MPS kept changing.

We can incorporate some preprocessing scheme for the QM compression. For example instead of dealing a bit at a time, we could combine several bits together, and map it to some other symbols instead of “0” and “1”. Because the reason why we don’t get good compression result for some of the file is that symbol “0” and “1” are too random, the MPS keeps changing. By combining some input symbols

together and map them into another symbol set instead of “0” and “1”, we could have better compression results if the new symbol set is not that random.

PartB: Written Questions

I used the program written for Part A to output the values of A, C and Qe for this part.

- (1) MPS = 1, Current State = 0, A = 0x10000, C = 0x0.

Input bit stream: 0111111111

Result:

Input Symbol	Current MPS; A; C; Qe; State
0	MPS = 1 A = 65536 C = 0 Qe = 23069 Current State = 0
1	MPS = 1 A = 46138 C = 84934 Qe = 9606 Current State = 1
1	MPS = 1 A = 36532 C = 84934 Qe = 9606 Current State = 1
1	MPS = 1 A = 53852 C = 169868 Qe = 4372 Current State = 2
1	MPS = 1 A = 49480 C = 169868 Qe = 4372 Current State = 2
1	MPS = 1 A = 45108 C = 169868 Qe = 4372 Current State = 2
1	MPS = 1 A = 40736 C = 169868

	Qe = 4372 Current State = 2
1	MPS = 1 A = 36364 C = 169868 Qe = 4372 Current State = 2
1	MPS = 1 A = 63984 C = 339736 Qe = 2059 Current State = 3
1	MPS = 1 A = 61925 C = 339736 Qe = 2059 Current State = 3

The output stream:

- (2) MPS = 0, Current State = 0, A = 0x10000, C = 0x0.
Input bit stream: 1111010000

Result:

Input Symbol	Current MPS; A; C; Qe; State
1	MPS = 0 A = 65536 C = 0 Qe = 23069 Current State = 0
1	MPS = 0 A = 46138 C = 84934 Qe = 9606 Current State = 1
1	MPS = 0 A = 38424 C = 485864 Qe = 23167 Current State = 14
1	MPS = 0 A = 61028 C = 1943456 Qe = 16165 Current State = 15

0	MPS = 0 A = 64660 C = 7953276 Qe = 23265 Current State = 36
1	MPS = 0 A = 41395 C = 7953276 Qe = 23265 Current State = 36
0	MPS = 0 A = 36260 C = 15906552 Qe = 18508 Current State = 37
0	MPS = 0 A = 37016 C = 31848608 Qe = 14861 Current State = 38
0	MPS = 0 A = 44310 C = 63697216 Qe = 12017 Current State = 39
0	MPS = 0 A = 64586 C = 516736 Qe = 9759 Current State = 40

Problem 2: Scalar and Vector Quantization

Part A: Written Questions

(1) Short Questions:

- a) Quantization is a lossy compression process. It first groups input data into subset of small groups according to the codebook, which is a quantization table, and then uses its codeword to represent each symbol in the sub-group. By doing this, multiple different symbols in the original dataset will be replaced by the same codeword. Therefore, the compressed data will be different from the original data. This will be the major difference between the symbol sets before and after quantization. Since the codeword uses fewer bits than the original data, the dataset will be compressed. But you will lose information from the original dataset.

- b) Since quantization is a lossy compression process, the source data cannot be perfectly reconstructed at the decoder side if quantization is involved. The error is introduced when you map multiple symbols in the source data to a single codeword, and use that codeword to represent the sub-group of symbols.
- c) The way to reduce the errors in (b) is to divide the source data into smaller sub-groups, which increases the resolution. It means we have to have a larger codebook with more codewords. The trade-off of having larger codebook is increasing the bits for the codewords, which means the quantized data will be larger.
- (2) Quantize a gray-level image:

a) Original gray-level image is

4	7	1	7
5	2	3	5
1	2	7	4
7	0	6	5

The codebook is $\{(2,6),(6,2),(3,4)\}$

b) Quantization Scheme:

Take two adjacent pixels in a row at a time from the original image as a block, calculate its distance from each codeword, by using equation $D^2 = (S1-C1)^2 + (S2-C2)^2$. See which distance is the shortest. Replace the two pixel values with the codeword which has the shortest distance.

c) Quantized image:

2	6	2	6
6	2	3	4
3	4	6	2
6	2	6	2

d) Mean-Squared Error (MSE):

$$\begin{aligned} \text{MSE} &= 1/M \sum (y_i - x_i)^2 \\ &= 1/16 * (2^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 2^2 + 2^2 + 1^2 + 2^2 + 1^2 + 2^2 + 3^2) \\ &= 2.25 \end{aligned}$$

e) Peak Signal Noise Ratio (PSNR):

$$\begin{aligned} \text{PSNR} &= 10\log_{10}((2^n - 1)^2 / \text{MSE}) \\ &= 10\log_{10}(7^2 / 2.25) \\ &= 13.38 \end{aligned}$$

Part B: Lloyd-Max Scalar Quantizer Design

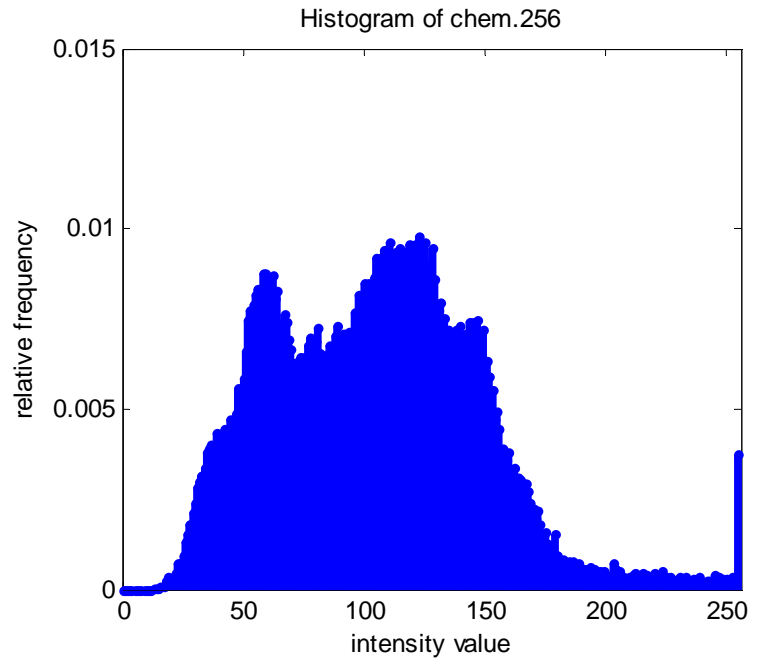
(1) Plot the histogram of each image file:

I used MATLAB to read in the images and to plot their histogram. The m code is provided in my .zip file called plot_histogram.m.

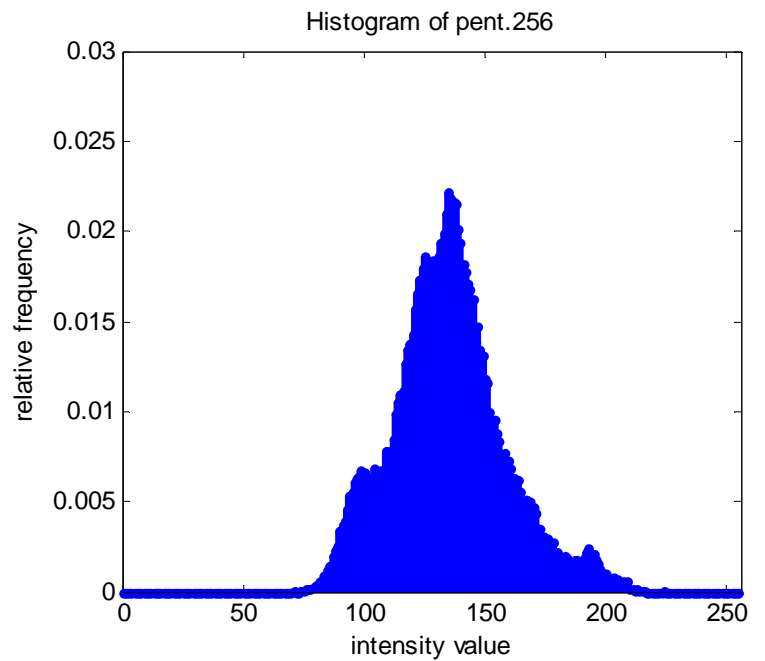
The plots:



chem.256

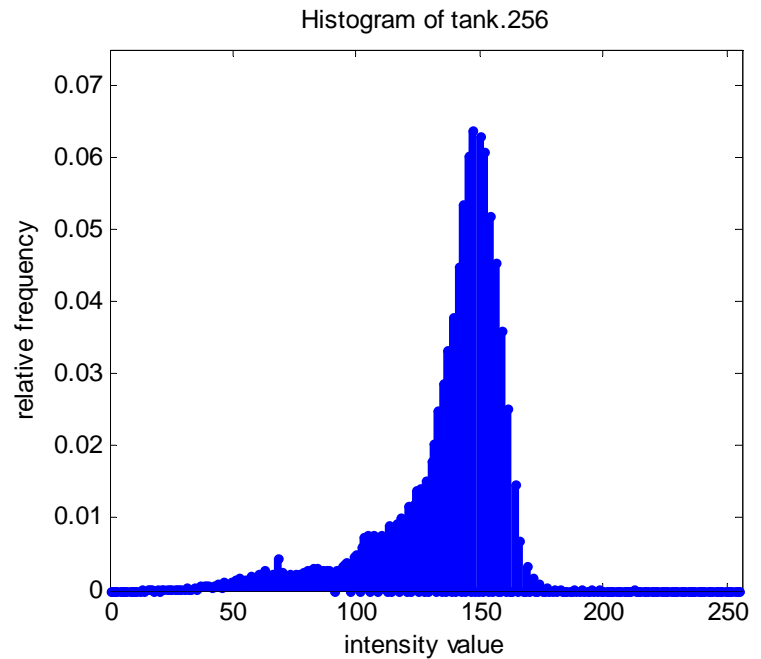


pent.256





tank.256



(2) Implementation of Scalar Quantizer using C++

- a) Firstly, use the function training() to combine the three images chem..256, pent.256 and tank.256 into a training set. The way I combine the image set is take one pixel value from every 3 pixels in a image, and combine the 3 values from each image together. For example, take the pixel value at position $3*i$ from chem.256; take the pixel value at position $3*i+1$ from pent.256; take the pixel value at position $3*i+2$ from tank.256.
- b) Secondly, use the function initialize_codebook() to initialize the codebook according to the bits it required, either 3-bit or 5-bit. The way I choose the initial codebook is by uniform distribution. For 3-bit case, divide the 256 values into 8 intervals uniformly, take the middle point of each interval as the codeword R_i ; For 5-bit case, divide the 256 values into 32 intervals uniformly, also take the middle point of each interval as the codeword R_i .
- c) Thirdly, use the function update_codebook() to get the final codebook. It uses a loop to do the iteration, each time it computes the new R_i first using the Lloyd-Max equation #1, then uses the new R_i to compute the new T_i using the Lloyd-Max equation #2. The iteration will stop when the MSE satisfies $\epsilon = 0.001$. In order to waste no codeword for the fixed-length code, I try to replace the codeword in an empty cell with a new code-word, generated by splitting the most populated cell. This should be the best scheme to deal with the issue.

- d) Finally, use the `encode()` function to quantize the input image. It uses the `updated_codebook` to quantize each pixel of the image. And write the quantized value into a file.

The codes I have for this part still have some problems, therefore I cannot get the results of the quantized image. I think one of the issues is that I didn't set the file stream to the beginning each time I read in the training set. Therefore, when I read the training set for the second time to update the codebook, it gives an EOF symbol. Due to time issue, I couldn't finish the code for this part.

Part C: The Lloyd-Max Vector Quantizer Design

- (1) Implementation of Lloyd-Max Vector Quantizer using C:
- a) "block.c" file is used for setting blocks from the training set. It takes a raw image inputfile and creates outputfile which is a list of vectors where each vector is a block from inputfile. The inputfile has dimensions rows and columns, and the block has dimensions blockheight and blockwidth.
- b) "lloyd0.c", "lloyd1.c" and "lloyd2.c" is used for performing the generalized lloyd algorithm for the given codebook with size codewords. lloyd exits with a codebook if the percent change in distortion from one pass to the next is less than threshold. If there are any empty cells, lloyd will try to split the most populous cells. lloyd will attempt to split cells up to size times unless the distortion is zero. If the distortion is zero, then those codewords are returned and the global variable codebooksize is modified since large codebooks can no longer be made. This ensures that the zero distortion codebook is returned to the user, but allows that the program stdvq terminate normally. If the `write_all_codebooks` option is not selected that the program will terminate.
- c) "stdvq.c" file is used for running the GLA for codebooks of size 2^n $n = 0, 1, 2, \dots$ until the final size is reached. The final size can be any integer value. Each increase in the size of the codebook is done by splitting codewords from the next smallest codebook, (perturbed versions of the old codewords). The GLA continues to run until the change in distortion is less than threshold. The GLA will abort if there are cells which cannot be filled. If there are empty cells, the lloyd iteration tries to split the most populous cells only, (individual cell distortion is not considered). There is one flag (-W). If the flag is not specified then only the final codebook is written. If the flag is specified, then all intermediary codebooks are written as well. Each codebook has the following format:
- | TYPE | SIZE | DESCRIPTION |
|---------|----------------|------------------------------|
| long | 1 | number of codewords (size) |
| integer | 1 | vector dimension (dimension) |
| double | size*dimension | codewords |

- d) I set the “training.bat” to 2x2 blocks for read in, name the blocked file training_data.TS. Then use “stdvq” to get the codebook. For detail information about how to run the “.o” files for codebook, please refer to my readme file.

For detail information, please refer to the source code.

*Note: I used the codes written by Jill Goldschneider from Stanford. Modified some parts to fit the homework requirements. Compiled and ran the codes to get a codebook of size 8 for the “training.bat” provided.

- (2) Here is the result for the cookbook:

OPTIONS	DESCRIPTIONS	SETTINGS
-t	training sequence	training_data.TS
-c	codebook name	codebook
-d	vector dimension	4
-f	codebook size	8
-h	convergence threshold	0.01
-a	codeword split additive offset	0.01
-m	codeword split multiplicative offset	0.01
-s	constrained search choice	1
-W	write intermediate codebooks	

```
Codebook distortion of size 1 : 20699.000488
Codebook distortion of size 2 : 15807.557617
Codebook distortion of size 4 : 12041.945696
Codebook distortion of size 8 : 6100.876551
```

As we can see from the result above, by doing iteration, we get a codebook of size 8. The distortion is decreasing when we split the codeword and increase the codebook size.

The codebook of size 8 is stored in the same folder with all the codes, called “codebook.8”

Part D: Tree Structured Vector Quantization

(1) Implementation of TSVQ:

- a) The main objective is to understand the supplied TSVQ code and implement it on the test image with the various codebooks. The codebooks of size 64, 128, 256 were obtained from the eight images which were blocked into three vectors of size 4, 16, 64. I used the source code provided for most parts.
- b) Combine the 8 supplied images, chem.256, couple.256, elaine.256, fl6.256, house.256, moon.256, pent.256, and tank.256 into a single training set by using the ‘cat’ command at the unix prompt, a training set will then be created.

- c) Use the “block.c” file provided by Jill Goldschneider to set the blocks for the training set file. It will break the training data into 3 different vector dimensions of 4, 16, 64 or 2x2, 4x4, and 8x8 blocks, respectively.
- d) Call the tsvq program 9 times to create 9 sets of codebooks with different dimension requirements and bit-rate requirements.

For detail information, please refer to the source code.

(2) Results:

9 sets of different codebooks are showed below. Codebook MSE (distortion) and entropy results are provided as well as encoding performance results.

Dimension:	Codebook #1	Codebook #2	Codebook #3
(2x2 Block)	N = 64	N = 128	N = 256

Codebook #1:

Codebook Statistics		Performance Results	
RATE	DISTORTION	Codebook file:	codebook1
0.000000	13174.839146	Vector dimension:	4
1.000000	4937.973083	Number of Nodes:	127
1.300049	3863.834415	Image to encode:	test.256
2.000000	1522.077163	Encoded file:	test1.256
2.131439	1414.172659	Number of pixels encoded:	65536
2.300049	1278.961841	Average rate:	6.000000
2.790932	939.910107	Empirical entropy:	4.828254
3.000000	782.214867	Average distortion:	936.495240
3.075165	768.778528	Maximum codeword length:	6
3.131439	756.812599		
The number of nodes is 127			
The empirical entropy is 5.063443			

Codebook #2:

Codebook Statistics		Performance Results	
RATE	DISTORTION	Codebook file:	codebook2
0.000000	13174.839146	Vector dimension:	4
1.000000	4937.973083	Number of Nodes:	255
1.300049	3863.834415	Image to encode:	test.256
2.000000	1522.077163	Encoded file:	test2.256
2.131439	1414.172659	Number of pixels encoded:	65536
2.300049	1278.961841	Average rate:	7.000000
2.790932	939.910107	Empirical entropy:	5.556185
3.000000	782.214867	Average distortion:	739.139638

3.075165 768.778528 3.131439 756.812599 The number of nodes is 255 The empirical entropy is 5.754440	Maximum codeword length: 7
---	----------------------------

Codebook #3:

Codebook Statistics	Performance Results
RATE DISTORTION 0.000000 13174.839146 1.000000 4937.973083 1.300049 3863.834415 2.000000 1522.077163 2.131439 1414.172659 2.300049 1278.961841 2.790932 939.910107 3.000000 782.214867 3.075165 768.778528 3.131439 756.812599 The number of nodes is 511 The empirical entropy is 6.499334	Codebook file: codebook3 Vector dimension: 4 Number of Nodes: 511 Image to encode: test.256 Encoded file: test3.256 Number of pixels encoded: 65536 Average rate: 8.000000 Empirical entropy: 6.363653 Average distortion: 513.800683 Maximum codeword length: 8

Dimension: Codebook #4 Codebook #5 Codebook #6
 (4x4 Block) N = 64 N = 128 N = 256

Codebook #4:

Codebook Statistics	Performance Results
RATE DISTORTION 0.000000 52698.202046 1.000000 21458.050947 1.275330 17746.066016 2.000000 8440.446070 2.123260 8084.135429 2.275330 7620.089046 2.795502 6168.350291 3.000000 5529.983788 3.071350 5482.804600 3.123260 5441.488511 The number of nodes is 127 The empirical entropy is 5.005739	Codebook file: codebook4 Vector dimension: 16 Number of Nodes: 127 Image to encode: test.256 Encoded file: test4.256 Number of pixels encoded: 65536 Average rate: 6.000000 Empirical entropy: 4.849769 Average distortion: 10983.032114 Maximum codeword length: 6

Codebook #5:

Codebook Statistics		Performance Results	
RATE	DISTORTION	Codebook file:	codebook5
0.000000	52698.202046	Vector dimension:	16
1.000000	21458.050947	Number of Nodes:	255
1.275330	17746.066016	Image to encode:	test.256
2.000000	8440.446070	Encoded file:	test5.256
2.123260	8084.135429	Number of pixels encoded:	65536
2.275330	7620.089046	Average rate:	7.000000
2.795502	6168.350291	Empirical entropy:	5.617530
3.000000	5529.983788	Average distortion:	9412.849119
3.071350	5482.804600	Maximum codeword length:	7
3.123260	5441.488511		
The number of nodes is 255			
The empirical entropy is 5.694968			

Codebook #6:

Codebook Statistics		Performance Results	
RATE	DISTORTION	Codebook file:	codebook6
0.000000	52698.202046	Vector dimension:	16
1.000000	21458.050947	Number of Nodes:	511
1.275330	17746.066016	Image to encode:	test.256
2.000000	8440.446070	Encoded file:	test6.256
2.123260	8084.135429	Number of pixels encoded:	65536
2.275330	7620.089046	Average rate:	8.000000
2.795502	6168.350291	Empirical entropy:	6.367953
3.000000	5529.983788	Average distortion:	8501.320199
3.071350	5482.804600	Maximum codeword length:	8
3.123260	5441.488511		
The number of nodes is 511			
The empirical entropy is 6.478179			

Dimension: Codebook #7 Codebook #8 Codebook #9
(8x8 Block) N = 64 N = 128 N = 256

Codebook #7:

Codebook Statistics		Performance Results	
RATE	DISTORTION	Codebook file:	codebook7
0.000000	210779.996195	Vector dimension:	64
1.000000	93728.299325	Number of Nodes:	127
1.256104	80274.442430	Image to encode:	test.256

2.000000	44488.575677	Encoded file:	test7.256
2.113525	43425.382679	Number of pixels encoded:	65536
2.256104	41669.618909	Average rate:	6.000000
2.799927	35907.141271	Empirical entropy:	3.987154
3.000000	33223.534707	Average distortion:	110981.087023
3.070312	33041.413166	Maximum codeword length:	6
3.113525	32927.765510		
The number of nodes is 127			
The empirical entropy is 5.056808			

Codebook #8:

Codebook Statistics		Performance Results	
RATE	DISTORTION	Codebook file:	codebook8
0.000000	210779.996195	Vector dimension:	64
1.000000	93728.299325	Number of Nodes:	255
1.256104	80274.442430	Image to encode:	test.256
2.000000	44488.575677	Encoded file:	test8.256
2.113525	43425.382679	Number of pixels encoded:	65536
2.256104	41669.618909	Average rate:	7.000000
2.799927	35907.141271	Empirical entropy:	4.666980
3.000000	33223.534707	Average distortion:	106275.601288
3.070312	33041.413166	Maximum codeword length:	7
3.113525	32927.765510		
The number of nodes is 255			
The empirical entropy is 5.893883			

Codebook #9:

Codebook Statistics		Performance Results	
RATE	DISTORTION	Codebook file:	codebook9
0.000000	210779.996195	Vector dimension:	64
1.000000	93728.299325	Number of Nodes:	471
1.256104	80274.442430	Image to encode:	test.256
2.000000	44488.575677	Encoded file:	test9.256
2.113525	43425.382679	Number of pixels encoded:	65536
2.256104	41669.618909	Average rate:	8.000000
2.799927	35907.141271	Empirical entropy:	5.254683
3.000000	33223.534707	Average distortion:	101746.320433
3.070312	33041.413166	Maximum codeword length:	8
3.113525	32927.765510		
The number of nodes is 471			
The empirical entropy is 6.637334			

(3) Discussion:

The distortion or MSE of the various vector sets increased when the vector dimensions gets bigger. This is because when the block is larger, the resolution is poorer. Therefore, MSE is larger.

Here the rate is the average number of bits per vector and the distortion is the average mean squared error per vector. The distortions decreased when doing the iterations and building the codebook. This is because during each iteration, the codeword of the codebook will split using a binary trees structure. By doing this, the codebook size is increasing, and there will be more codewords for mapping.

The convergence rate of the generalized Lloyd iteration happened most quickly in the case of the 2x2 block or dimension vector 1 training set. This is expected as there is better resolution during the Lloyd iteration to provide a better split of the cell and hence a quicker convergence. Notice that the distortion from the first to second iteration is reduced much quicker during the 2x2 block than the 8x8 block.

For more information about the source codes and how to compile and run the code, please refer to *Readme.txt* file in *EE669_HW_2_5044493461_Zhang.zip* file

Readme.txt

EE669 Homework #2

Submission Date: February 23, 2007

Name: Xiao ZHANG

USC ID: 5044-4934-61

Email: zhangxia@usc.edu

Platform: Unix(Aludra)

Editor: Emacs

Compiler: g++

QM folder contains all the files for QM Compression:

1. put all the 4 data sets in the same folder
2. use compile.bat file to compile all the codes. After that, there will be a qm executable file
3. use qm to compress all 4 data. don't have to pass any argument, just call qm. It will show each compression information

*note: I used the same code structure to calculate the written problem. Within the code, there are some parts that I comment

out, they are used for written problem.

SQ folder contains the files I wrote for Scalar Quantization:

1. the read_raw_image.m file is used for plotting the histogram in MATLAB

2. the .cpp files are for the implementation of SQ Quantizer. However, there is still some problem with my code, I can't compile it.

VQ folder contains all the files for Vector Quantization:

1. put training.dat file in the folder
2. first use compile.bat file to compile all the codes. There might be some warnings, but it will create the right output file.
3. then use run.bat file to run the executable program.

TSVQ folder contains all the files for Tree-Structured Vector Quantization:

1. put all the required data files in the folder
2. use the compile.bat file to compile all the codes.
3. after compiling, there will be several .o files.
 - Use "block" to set the the dimension of different blocks.
 - Use "tsvq" to generate codebooks with different bit-rate.
 - Use "tsvqe" to encode input file.
 - Use "unblock" to reset the block structures.

Since we have to implement this executable file to different dimension and bit-rate requirement. I didn't make any run file.

User have to pass in correct specifications for dimensions and bit-rates. Extra README file included for this part.

For detail information, please refer to the source code and the Report