Assignment A4: Compression Techniques

CS 4640
Spring 2018

Assigned: 20 February 2018

Due: 8 March 2018

For this problem, handin Matlab .m files for the functions described by the headers below. Note that one of these is a driver which creates inputs for each function and runs the function on those inputs to obtain the output.

You are also to handin a pdf file (called A4.pdf) reporting on the analysis of the image *lennag* with:

1. A plot of the RMS error for lennag.
2. A plot of the components percentage. Discuss the tradeoff in loss of perfect recovery of the image and the compression ratio selected.
3. An imshow (with something useful visible!) of the decoded images at indexes: [1,2,3,4,8,16,32,64,86,100].
4. In looking at the difference image of the original image and the decoded version 16, can you suggest a further use of this method as an image processing technique?

Some notes:

- Indent headers correctly (5 spaces indented lines)
- Do not exceed 72 characters per source line
- CS4640_A4_driver: should show that each function works

None of the functions should write to the interpreter, draw, etc.

```
function [H_tree,codes] = CS4640_Huffman_encode(info)
% CS4640_Huffman_encode - produce Huffman code tree
% On input:
%     info (nx2 array): col 1 gives symbols indexes (i.e., [1:n])
%                       col 2 gives frequencies
% On output:
%     H_tree (tree struct):
%        .root (int): index of root of tree
%        .nodes (node vector struct)
%           .parent (int): index of parent
%           .left (int): index of left child
%           .right (int): index of right child
%           .children (1xk vector): indexes of left and right children
%           .symbol (int): col 1 of info for this symbol
%           .state (int): node's own index in tree
%           .frequency (double): occurrence count (absolute or
%           relative)
%           .val (nil): unused
%           .level (int): node's level in tree (i.e., root is 0)
%     codes (nx1 struct vector): codes for the n symbols
%         (q).code (1xn_q binary vector): binary code for symbol q
% Call:
%     info = [1,5; 2,9; 3,12; 4,13; 5,16; 6,45];
%     [H,codes] = CS4640_Huffman_encode(info);
%     CS4640_show_tree(H.nodes,11)
%      [11]
%       [6]
%       [10]
%        [8]
%         [3]
%         [4]
%        [9]
%         [7]
%          [1]
%          [2]
%         [5]
```

```
% Author:
%     <Your name>
%     UU
%     Spring 2018
%


function coded_im = CS4640_im2Hcode(im,codes,tab)
% CS4640_im2Hcode - produce Huffman coded image
% On input:
%     im (MxN gray level image): input image
%     codes (nx1 struct vector): codes for n symbols
%     tab (nx1 vector): gray level to index table
% On output:
%     coded_im (1xq binary vector): Huffman coded image
% Call:
%     H_coded = CS4640_im2Hcode(imH,H_codes,indexes);
% Author:
%     <Your name>
%     UU
%     Spring 2018
%


function list = CS4640_Huffman_decode(s,H,tab)
% CS4640_Huffman_decode - decode string using Huffman tree
% On input:
%     s (1xk vector): binary string (Huffman compressed code)
%     H (tree struct): see Huffman encode
%     tab (nx1 vector): gray level to index table
% On output:
%     list (1xn vector): decoded symbols separated by 0's
% Call:
%     sd = CS4640_Huffman_decode([1,1,0,1,1,1,0,0],H,indexes);
% Author:
%     <Your name>
%     UU
%     Spring 2018
%


function [low,high] = CS4640_arith_encode(info,seq)
```

```
% CS4640_arith_encode - produce arithmentic encoding of sequence
% On input:
%     info (nx2 array): col 1 has symbol indexes (i.e., [1:n])
%                       col 2 has symbol freuqencies (absolute or
%                       relative
%     seq (1xk vector): input sequence of symbol indexes
% On output:
%     low (double): low value of interval
%     high (double): high value of interval
% Call:
%     [low,high] = CS4640_arith_encode(info,[2,2,1,4]);
% Author:
%     <Your name>
%     UU
%     Spring 2018
%


function ime = CS4640_RLE_encode(im)
% CS4640_RLE_encode - produce run-length encoding of gray level image
% On input:
%     im (MxN array): gray level image
% On output:
%     ime (1x2k vector): k run length encodings; odd-numbered give run
%        length, while even numbered give gray level
% Call:
%     ime = CS4640_RLE_encode(im);
% Author:
%     T. Henderson
%     UU
%     Spring 2018
%


function im = CS4640_RLE_decode(ime,M,N)
% CS4640_RLE_decode - produce gray level image from run-length
% encoding
% On input:
%     ime (1x2k vector): image encoding:
%        odd elements: run length
%        even elements: gray level
```

```
%     M (int): number of rows in image
%     N (int): number of cols in image
% On output:
%     im (1x2k vector): image gray levels as linear vector
% Call:
%     im = CS4640_RLE_decode(ime,M,N);
% Author:
%     <Your name>
%     UU
%     Spring 2018
%


function [RMS,components,resim] = CS4640_DCT_analysis(im)
% CS4640_DCT_analysis - compute RMS error for DCT image encoding
% On input:
%     im (MxN array): gray level image
% On output:
%     RMS (101x1 vector): RMS error between original and encoded image
%     components (101x1 vector): percentage of non-zero components
%                                in coded image
%     resim (101x1 struct vector): has decoded images
%       (k).im (PxP array): inverse image of DCT coded image
%        where k = 1:101, and for k_th image, all components with
%        less than (k-1)*0.01*max_component_value are set to 0
% Call:
%     [res,resim] = CS4640_DCT_analysis(lennag);
% Author:
%     <Your name>
%     UU
%     Spring 2018
%


function CS4640_A4_driver
% CS4640_A4_driver - driver for A4 functions
% On input:
%     N/A
% On output:
%     N/A
% Call:
```

```
%     CS4640_A4_driver
% Author:
%     <Your name>
%     UU
%     Spring 2018
%
```