ECE 637 Lab 9 - JPEG encoding

Yifan Fei

Electrical Engineering
Purdue University

Instructor: Prof. Charles A. Bouman

Spring 2018

Section 2: DCT Block Transforms and Quantization

2.1: Exercise

2.1.1: Matlab script for block transforming, quantizing, and storing the file img03y.dq

```
clear all;
clc;
run('Qtables.m');

img = imread('img03y.tif');
% gamma = 0.25;
gamma = 1;
% gamma = 4;
img = double(img) - 128;

%% Block transforming, quantizing and storing
dct_blk = blockproc(img, [8 8],@(x)round(dct2(x.data, [8 8])./(Quant*gamma)));
f = fopen('img03y.dq', 'w');
% f = fopen('img03y_3.dq', 'w');
fwrite(f, size(dct_blk, 1), 'integer*2');
fwrite(f, size(dct_blk, 2), 'integer*2');
fwrite(f, dct_blk', 'integer*2');
```

2.1.2: Matlab script for restoring the image from the file img03y.dq.

```
f2 = fopen('img03y.dq', 'r');
% f2 = fopen('img03y_3.dq', 'r');
data = fread(f2, 'integer*2');
img_restoring = reshape(data(3:end), [data(2) data(1)])';
img_restoring = blockproc(img_restoring, [8 8],@(x) round(idct2(x.data.*Quant*gamma, [8 8]
img_restoring = img_restoring + 128;
img_restoring = uint8(img_restoring);
```

2.1.3:the original, restored, and difference images for γ = 0.25, 1, and 4.

```
%% difference images
img = imread('img03y.tif');
img_diff = 10 * (img - img_restoring) + 128;

figure(1);
image(img);
truesize;
colormap(gray(256));

figure(2);
image(img_restoring);
truesize;
colormap(gray(256));
imwrite(uint8(img_restoring),'img03y_2.tif');

figure(3);
image(img_diff);
```

```
truesize;
colormap(gray(256));
imwrite(uint8(img_diff),'img03y_diff_2.tif');
```

Figure 1: the original image vs the restored image vs the difference image for $\gamma = 0.25$

Figure 2: the original image vs the restored image vs the difference image for $\gamma = 1$

Comments: The distortion increases with the increasing of γ by observing the difference image. When γ=4, the image has the biggest distortion since the difference image is the most noisy out of three gamma values.

Figure 3: the original image vs the restored image vs the difference image for $\gamma = 4$

2.3: Differential Encoding and the Zig-Zag Scan Pattern

2.3.1:the image formed by the DC coefficients

The image formed by the DC coefficients is similar to the original image, though the resolution is lower, making it blurring.

Figure 4: the image formed by the DC coefficients

2.3.2: why the DC coefficients of adjacent blocks are correlated

The DC coefficients of adjacent blocks are correlated because the average gray level of adjacent image blocks is likely to be similar, and the average intensity of neighboring pixels is usually close in terms of values.

2.3.3:mean value of the magnitude of the AC coefficients for $\gamma = 1$

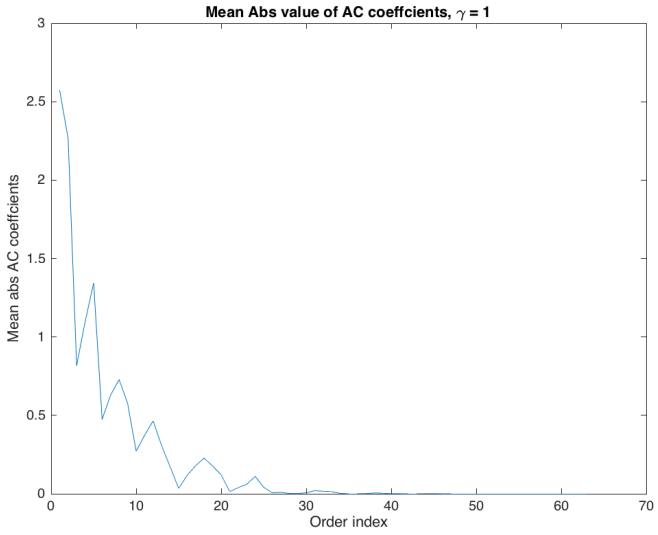

Figure 5: mean value of the magnitude of the AC coefficients for $\gamma = 1$

Matlab Code:

```
%% Section 2.3

[m,n] = size(img);
DC = zeros(m/8,n/8);

for i = 1:m/8
    for j = 1:n/8
        DC(i,j) = dct_blk((i-1)*8+1,(j-1)*8+1)+128;
    end
end
figure(4);
image(DC);
truesize;
colormap(gray(256));
imwrite(uint8(DC),'img03y_DC.tif');

N = m*n/64;

for p = 1:8
    for q = 1:8
        AC(p,q) = 0;
        for i = 1:m/8
            for j = 1:n/8
```

```
                AC(p,q) = AC(p,q)+abs(dct_blk((i-1)*8+p,(j-1)*8+q));
            end
        end
        AC(p,q) = AC(p,q)/ N;
    end
end

AC_coeff = AC(Zig);
AC_coeff = AC_coeff(2:end)
figure(5)
% index = 1:64
index = 1:63
plot(index,AC_coeff);
title('Mean Abs value of AC coeffcients, \gamma = 1')
xlabel('Order index')
ylabel('Mean abs AC coeffcients')
```

3: Entropy Encoding of Coefficients

3.1:C code for the subroutines

```c
#include "JPEGdefs.h"
#include "Htables.h"

int BitSize(int value) {
  int bitsize = 0;

  if (value < 0) {
    value *= -1;
  }
  while (value > 0) {
    bitsize++;
    value >>= 1;
  }

  return bitsize;
}

void VLI_encode(int bitsize, int value, char *block_code) {
  // VLC + VLI
  char VLI[13] = {0};

  if (value < 0) {
    value--; // 8-bit 2's complement
  }

  for (int i = bitsize - 1; i >= 0; i--) {
    if (value & 1){
        VLI[i] = '1';
      }
    else  {
        VLI[i] = '0';
    }
```

```
    // printf("%d\n",value);
    // printf("%s\n", VLI);
    value >>= 1;
  }
  strcat(block_code, VLI);
}

void ZigZag(int ** img, int y, int x, int *zigline) {
  for (int i = 0; i < 8; i++) {
    for (int j = 0; j < 8; j++) {
      zigline[Zig[i][j]] = img[y+i][x+j];
    }
  }
}

void DC_encode(int dc_value, int prev_value, char *block_code) {
  int diff_value = dc_value - prev_value;
  // printf("diff_value = %d\n", diff_value);
  int size = BitSize(diff_value);
  strcat(block_code, dcHuffman.code[size]);
  VLI_encode(size, diff_value, block_code);
}

void AC_encode(int *zigzag, char *block_code) {
  int idx = 1;
  int zerocnt = 0;
  int bitsize;

  while (idx < 64) {
    if (zigzag[idx] == 0) {
      zerocnt++;
    }
    else {
      for (; zerocnt > 15; zerocnt -= 16){
        strcat(block_code, acHuffman.code[15][0]);
      }

      bitsize = BitSize(zigzag[idx]);
      strcat(block_code, acHuffman.code[zerocnt][bitsize]);
      VLI_encode(bitsize, zigzag[idx], block_code);
      zerocnt = 0;
      // fprintf(stdout, "debug in AC_encode()\n");
    }
    idx++;
  }
  if (zerocnt) {
    strcat(block_code, acHuffman.code[0][0]);
  }
}

void Block_encode(int prev_value, int *zigzag, char *block_code) {

  DC_encode(zigzag[0], prev_value, block_code);
  AC_encode(zigzag, block_code);
```

```
}

int Convert_encode(char *block_code, unsigned char *byte_code) {
  int len = strlen(block_code);
  int bytes = len / 8;
  int idx = 0;

  for (int i = 0; i < bytes; i++) {
    for (int j = 0; j < 8; j++) {
      byte_code[idx] <<= 1;

      if (block_code[i*8 + j] == '1') {
        byte_code[idx]++;
      }
    }

    if (byte_code[idx] == 0xff) {
      byte_code[++idx] = 0x00;
      bytes++;
    }
    idx++;
  }
  strcpy(block_code, block_code + len / 8 * 8);

  return bytes;
}

unsigned char Zero_pad(char *block_code) {
  unsigned char byte_code;
  int len = strlen(block_code);

  for (int i = 0; i < len; i++) {
    byte_code <<= 1;

    if (block_code[i] == '1') {
      byte_code++;
    }
  }
  byte_code <<= (8 - len);

  return byte_code;
}
```

3.2:main program JPEG encode

```
/*********************************************************/
/* JPEG_encoder    By Jinwha Yang and Charles Bouman   */
/* Apr. 2000.      Built for EE637 Lab.                */
/* All right reserved for Prof. Bouman                 */
/*********************************************************/

#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>

#include "Htables.h"
#include "JPEGdefs.h"
#include "allocate.h"



int main(int argc, char* argv[])
{
  int  **input_img;  /* Input set of DCT coefficients read from matlab file */
  FILE  *outfp;      /* File pointer to output JPEG image */
  int    row;        /* height of image */
  int    column;     /* width of image */
  double gamma;      /* scaling factor for quantizer */

  /* Use command line arguements to read matlab file, and return */
  /* values of height, width, quantizer scaling and file pointer */
  /* to output JPEG file.                                        */
  input_img = get_arguments(argc,argv,&row,&column,&gamma,&outfp) ;

  /* scale global variable for quantization matrix */
  if( gamma > 0 )
    change_qtable(gamma) ;
  else {
    fprintf(stderr, "\nQuantizer scaling must be > 0.\n") ;
    exit(-1) ;
  }

  /* Encode quantized DCT coefficients into JPEG image */
  jpeg_encode(input_img,row,column,outfp) ;

  return 1 ;
}


void change_qtable(double scale)
{
  int     i,j ;
  double  val ;

  for(i=0;i<8;i++){
    for(j=0;j<8;j++){
      val = Quant[i][j]*scale ;
      /* w.r.t spec, Quant entry can be bigger than 16 bit */
      Quant[i][j] = (val>65535) ? 65535 : (int)(val+0.5) ;
    }
  }
}


int **get_arguments(int argc,
       char *argv[],
       int *row,
```

```
        int *col,
        double *gamma,
        FILE **fp )
{
  FILE *   inp ;
  short**  img ;
  int  **  in_img ;
  short    tmp ;
  int      i,j ;


  /* needs at least 2 argument */
  switch(argc){
  case 0:
  case 1:
  case 2:
  case 3: usage(); exit(-1) ; break ;
  default:

    /* read Quant scale */
    sscanf(argv[1],"%lf",gamma) ;

    /* prepare output file */
    *fp = fopen(argv[3],"wb") ;
    if(*fp==NULL) {
      fprintf(stderr,
      "\n%s file error\n",argv[3]) ;
      exit(-1) ;
    }

    /* read input file */
    inp = fopen(argv[2],"rb") ;
    if( inp == NULL ) {
      fprintf(stderr,
        "\n%s open error\n",argv[2]) ;
      exit(-1) ;
    }
    /* input file has 2 16 bit(short) row, column info */
    /* valid 2-D array follows */
    fread(&tmp,sizeof(short),1,inp) ;
    *row = (int) tmp ;
    fread(&tmp,sizeof(short),1,inp) ;
    *col = (int) tmp ;

    img = (short **)get_img(*col,*row,sizeof(short)) ;
    fread(img[0],sizeof(short),*col**row,inp) ;
    fclose(inp) ;

    break ;
  }

  in_img = (int **)get_img(*col,*row,sizeof(int)) ;
  for( i=0 ; i<*row; i++ ){
    for( j=0 ; j<*col; j++ ){
```

```
      in_img[i][j] = (int) img[i][j] ;
    }
  }
  free_img((void**)img) ;
  return( in_img ) ;
}



void jpeg_encode(int **img, int h, int w, FILE *jpgp)
{
  int    x, y, length ;
  int    prev_dc = 0 ;
  unsigned char val ;
  static int    zigline[64] ;
  static char   block_code[8192] = {0} ;
  static unsigned char byte_code[1024] ;

  printf("\n JPEG encode starts...") ;
  /* JPEG header writes */
  put_header(w,h,Quant,jpgp) ;

  printf("\n Header written...\n Image size %d row  %d column\n",h,w) ;
  /* Normal block processing */
  for( y = 0 ; y < h ; y += 8) {
    for( x = 0 ; x < w ; x += 8 ){
      /* read up 8x8 block */
      ZigZag(img,y,x,zigline) ;
      Block_encode(prev_dc,zigline,block_code) ;
      prev_dc = zigline[0] ;
      length = Convert_encode(block_code,byte_code) ;
      fwrite(byte_code,sizeof(char),length,jpgp) ;
    }
    printf("\r (%d)th row processing   ",y) ;
  }
  printf("\nEncode done.\n") ;
  /* Zero padding */
  if( strlen(block_code) ){
    val = Zero_pad(block_code) ;
    fwrite(&val,sizeof(char),1,jpgp) ;
  }

  /* EOI */
  put_tail(jpgp) ;
  fclose(jpgp) ;
  free_img((void **)img) ;
}



void usage(void)
{
  fprintf(stderr,"\nJPEG_encode <Quant scale> <in_file> <out_file>");
  fprintf(stderr,"\n<Quant scale> - gamma value in eq (1)");
```

```
  fprintf(stderr,"\n<in_file> - output file using section 2.1");
  fprintf(stderr,"\n<out_file> - JPEG output file");
}
```

3.3: The encoded image using $\gamma = 1$

Figure 6: The encoded image using $\gamma = 1$

3.4: three printouts, the encoded image using $\gamma = 0.25$ or 4

Figure 7: The encoded image using $\gamma = 0.25$

Figure 8: The encoded image using $\gamma = 4$