ECE 637 Lab 9 Report
JPEG Image Coding

Xihui Wang

Section2 – DCT Block Transforms and Quantization

2.1.1 Hard copy of Matlab script for block transforming, quantizing, and storing the file img03y.dq

```
% write
function []=write(img, gamma)

img = double(img)-128;
run('Qtables.m');

fn = @(x) round(dct2(x.data,[8,8])./(Quant*gamma));
dct_blk = blockproc(img,[8,8],fn);

[m,n] = size(dct_blk);
id = fopen('img03y.dq','w');
fwrite(id, m, 'integer*2');
fwrite(id, n, 'integer*2');
fwrite(id, dct_blk', 'integer*2');
fclose(id);
```

2.1.2 Hard copy of Matlab script for restoring the image from the file img03y.dq

```
% read
function [imgg]=read(gamma)

run('Qtables.m');
id = fopen('img03y.dq', 'r');
dct_blkk = fread(id, 'integer*2');
fclose(id);
imgg = reshape(dct_blkk(3:end), [dct_blkk(2) dct_blkk(1)])';

fn =  @(x) round(idct2(x.data.*Quant*gamma, [8 8]));
imgg = blockproc(imgg, [8 8],fn);
imgg = imgg + 128;
```

2.1.3 Hard copy of the original, restored, and difference images for $\gamma = 0.25,\ 1,\ and\ 4$

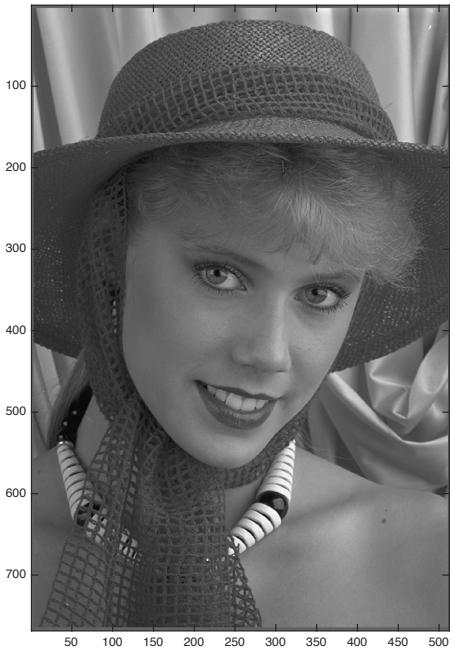

Fig 2-1-3-1-1 The original image $\gamma = 0.25$

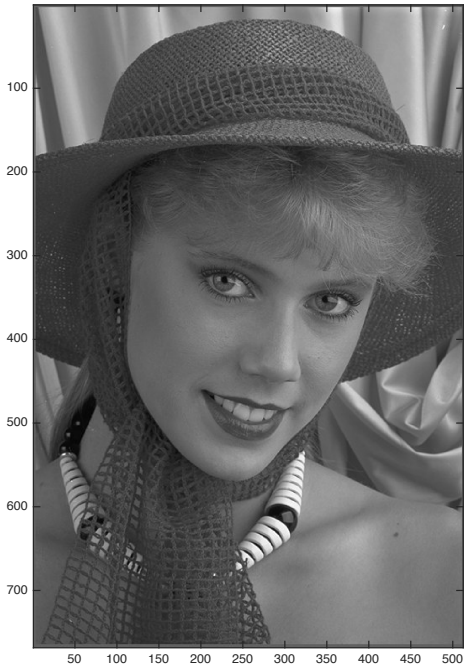

Fig 2-1-3-1-2 The restored image $\gamma = 0.25$

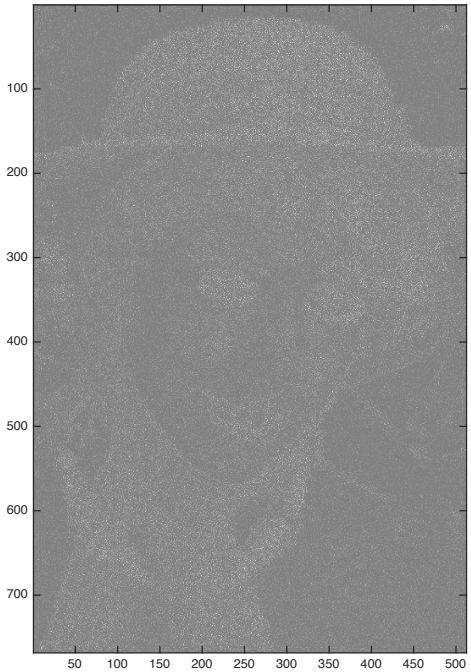

Fig 2-1-3-1-3 The difference image $\gamma = 0.25$

Fig 2-1-3-2-1 The original image $\gamma = 1$

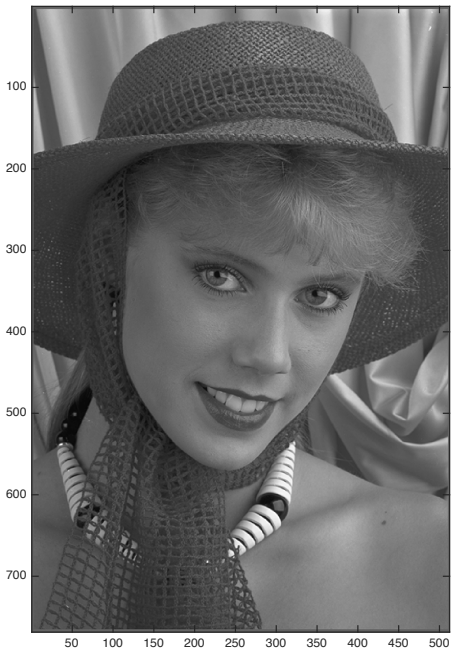

Fig 2-1-3-2-2 The restored image $\gamma = 1$

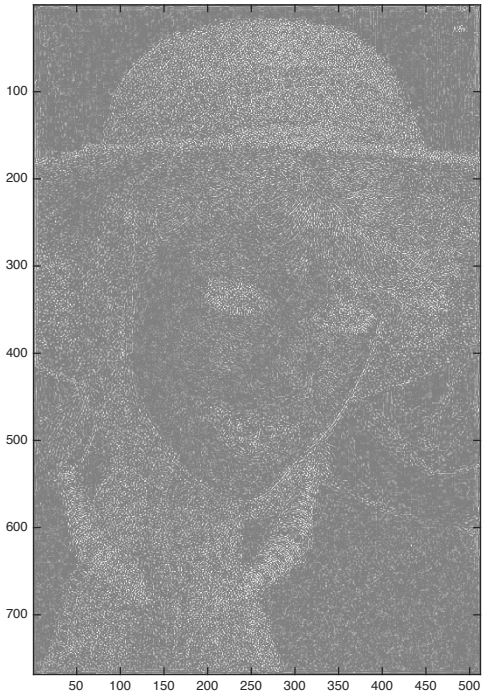

Fig 2-1-3-2-3 The difference image $\gamma = 1$

Fig 2-1-3-3-1 The original image $\gamma = 4$

Fig 2-1-3-3-2 The restored image $\gamma = 4$

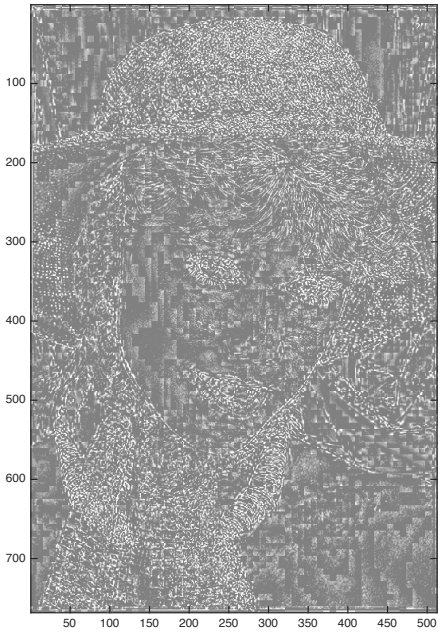

Fig 2-1-3-3-3 The difference image $\gamma = 4$

2.1.4 Comment on the effect of γ on my results

Based on the restored image results and difference image results for different γ. I noticed that the larger the γ value, the worse the image quality.

2.2.1 Hard copy of the image formed by the DC coefficients. What does it look like?

The copy of the image formed by the DC coefficients is looked like the blurry original image, and much smaller than the original image.

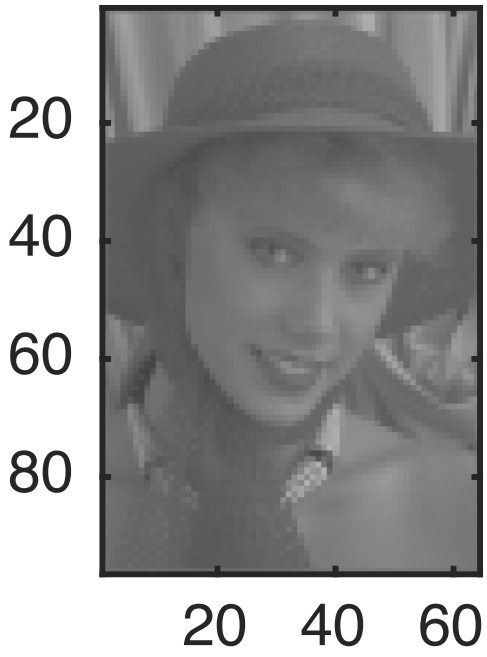

Fig 2-2-1 The image formed by the DC coefficients

2.2.2 Explain why the DC coefficients of adjacent blocks are correlated

Because the DC coefficients are corresponding to the average gray level of each 8 x 8 block, and the average gray level of adjacent image blocks is likely to be similar. For normal pictures, the value of a pixel is similar to its neighborhood pixels.

2.2.3 Plot of the mean value of the magnitude of the AC coefficients for $\gamma = 1.0$. Explain the form of this plot

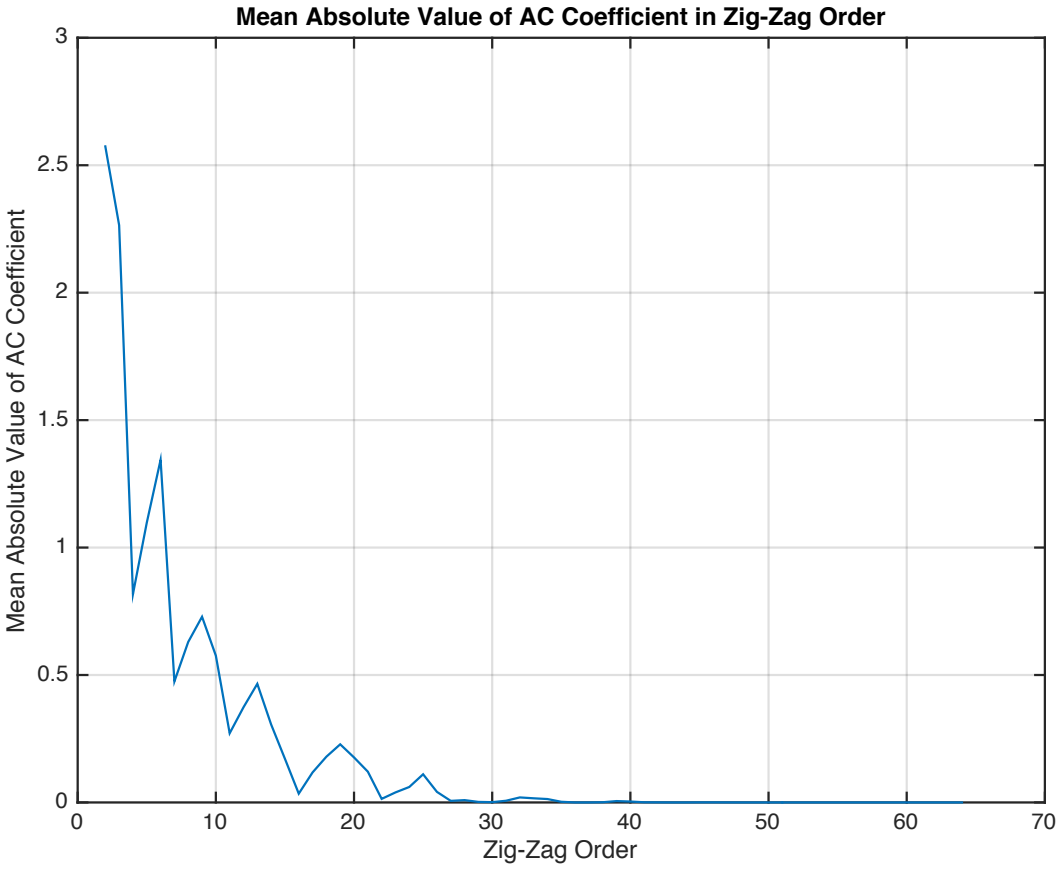

Fig 2-2-3 Mean Absolute Value of AC Coefficient in Zig-Zag Order

The DC coefficient of each block has the highest energy, and the distance between each AC coefficient and DC coefficient determines the energy of that AC coefficient. AC coefficients that close to the DC coefficient has higher energy than AC coefficients that far from the DC coefficient.

Section3 – Entropy Encoding of Coefficients

3.1 Hand in C code for the subroutines BitSize, VLI encode, ZigZag, DC encode, AC encode, Block encode, Convert encode, and Zero pad.

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

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

        while (value>0)
        {
                bitsize++;
                value = value/2;
        }

        return bitsize;
}

void VLI_encode(int bitsize, int value, char *block_code)
{
        int i;
        char VLI[20] = "";
        bitsize = BitSize(value);
        value = (value < 0)? (value-1) : value;

        for (i = bitsize - 1; i >= 0; i--)
        {
                VLI[i] = (value & 1)? '1':'0';
                value >>= 1;
        }

        strcat(block_code, VLI);
```

```
        return;
}

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

void DC_encode(int dc_value, int prev_value, char *block_code)
{
        int diff = dc_value - prev_value;
        int bitsize = BitSize(diff);

        strcat(block_code, dcHuffman.code[bitsize]);
        VLI_encode(bitsize, diff, block_code);
        return;
}

void AC_encode(int *zigzag, char *block_code)
{
        /* Init variables */
        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;
			}
			idx++;
		}

		/* EOB coding */
		if(zerocnt)	strcat(block_code, acHuffman.code[0][0]);
		return;
}

void Block_encode(int prev_dc, int *zigzag, char *block_code)
{
		DC_encode(zigzag[0], prev_dc, block_code);
		AC_encode(zigzag, block_code);
		return;
}

unsigned char bin2dec(char *bin)
{
		int dec = 0;
		for(int i=0; i<8; i++)
		{
			dec = dec + ((int)bin[i]-48)*pow(2, 7-i);
		}
		return (unsigned char) dec;
```

```
}

int Convert_encode(char *block_code, unsigned char *byte_code)
{
		int length = strlen(block_code) / 8;
		int len = length;
		int i,tmp,k;
		char temp[9]="";

		k = 0;
		for (i = 0; i < len; i++)
		{
			tmp = strxfrm (temp, block_code, 9);
			memmove(block_code, block_code+8, strlen(block_code));
			byte_code[k] = bin2dec(temp);

			if (byte_code[k] == 0xff)
			{
				k++;
				byte_code[k] = 0x00;
				length++;
			}
			k++;
		}
		return length;
}

unsigned char Zero_pad(char *block_code)
{
		int byte_value;
		int length = strlen(block_code);

		for(int i=0; i<length; i++)
		{
```

```
                byte_value = byte_value + ((int)block_code[i]-48)*pow(2, 7-
            i);
            }
            return (unsigned char) byte_value;
}
```

3.2 Hand in C code for the 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 Hand in the three printouts from xv.

Fig 3-3-1 The printout from xv with $\gamma = 0.25$

Fig 3-3-2 The printout from xv with $\gamma = 1$

Fig 3-3-3 The printout from xv with $\gamma = 4$