EEE412-IMAGE PROCESSING-LAB4

Author: Zhipeng Ye

Student ID: 1926908

Faculty: MSc Multimedia Telecommunications

Data: 21th October 2019

Finish the following tasks:

Write a function to calculate the entropy of an image. (**10'**)

Code

```
function entropy_value = my_entropy(varargin)

% check parameters
I = ParseInputs(varargin{:});

% convert 0-1 to 0-255
if ~islogical(I)
    I = im2uint8(I);
end

% calculate histogram counts
pixel_value = imhist(I(:));

% remove zero in p
pixel_value(pixel_value==0) = [];

% normalize p so that sum(p) is one.
p = pixel_value ./ numel(I);

entropy_value = -sum(p.*log2(p));

    % check parameters
    function I = ParseInputs(varargin)

        narginchk(1,1);

        validateattributes(varargin{1},{'uint8','uint16', 'double',
'logical'},...
            {'real', 'nonempty', 'nonsparse'},mfilename, 'I',1);
```

```
        I = varargin{1};

    end
end
```

Analysis

Entropy describe the magnitude of uncertainty. Firstly, I use ParaseInputs() to check whether parameters are valid. Secondly, I use imhist to get histogram of pixels. Then calculate the possibility of each pixels. Finally, I use formula to get value of entropy.

Use the above function in task (1) to calculate the entropy of the following images:

The original image "lenna512.bmp"; (**2'**)

Reduce the "lenna512.bmp" to the half size (both horizontally and vertically) by the down-sampling of using mean value; (**3'**)

Reduce the gray level of "lenna512.bmp" to 16 values by quantization with the base 16. (**3'**)

Compare the above three entropy values, what can you find? And explain your finding. (**8'**)

Code

```
clear
clc

im = imread('lenna512.bmp');
im_entropy = my_entropy(im);

im_half = down_samping(im);

im_half_entropy = my_entropy(im_half);

im_quantization = im./16*16;
im_quantization_entropy = my_entropy(im_quantization);
```

```
% The vision of my matlab is R2017, I can't use the mean function. So I
% implement an algorithm.

function narrow_matrix=down_samping(matrix)
   [m,n]=size(matrix);
   matrix_double=double(matrix);
   narrow_matrix=zeros(m/2,n/2);
   for i=1:2:m
       for j=1:2:n
%            calculate the mean of neighbor
```

```
            mean_value=
(matrix_double(i,j)+matrix_double(i,j+1)+matrix_double(i+1,j)+matrix_double(i
+1,j+1))/4;
            narrow_matrix((i+1)/2,(j+1)/2)=mean_value;
        end
    end
    narrow_matrix = uint8(narrow_matrix);
end
```

Result

```
im_entropy =

    7.3775

im_half_entropy =

    7.4231

im_quantization_entropy =

    3.4846
```

Analysis

As we can compare three entropy, the down-sampling image has a highest entropy, but it doen't change to much comparing with original image. The quantization of image has a lowest entropy, this means it has least information.

1. if we convert image to half size of original image. The entropy wouldn't change too much.
2. But I try some other scale of down-sampling, the entropy will be reduced with the decrease of size of image.
3. On the other hand, when we use quantization, the magnitude of information will be reduced. Because The magnitude of gray value is reduced and uncertainty of whole image come down.

Predictive coding:

Write a function of Raster-scan DPCM (differential pulse code modulation) coding for the input image. Here, the predictive function is the weighted average value of the neighboring pixels (left, left-top, and top): $p(r,c) = (2 * x(r,c-1) + x(r-1,c-1) + 2 * x(r-1,c))/5$, and the difference is presented by $e(r,c) = x(r,c) - p(r,c)$. The padding is suggested to use the same value as its neighbor. (**10'**)

Use the "lenna512.bmp" as input for the function (3.a) and show the output of difference *e*. (**2'**)

Calculate the entropy of the output image of task (3.b). (**2'**)

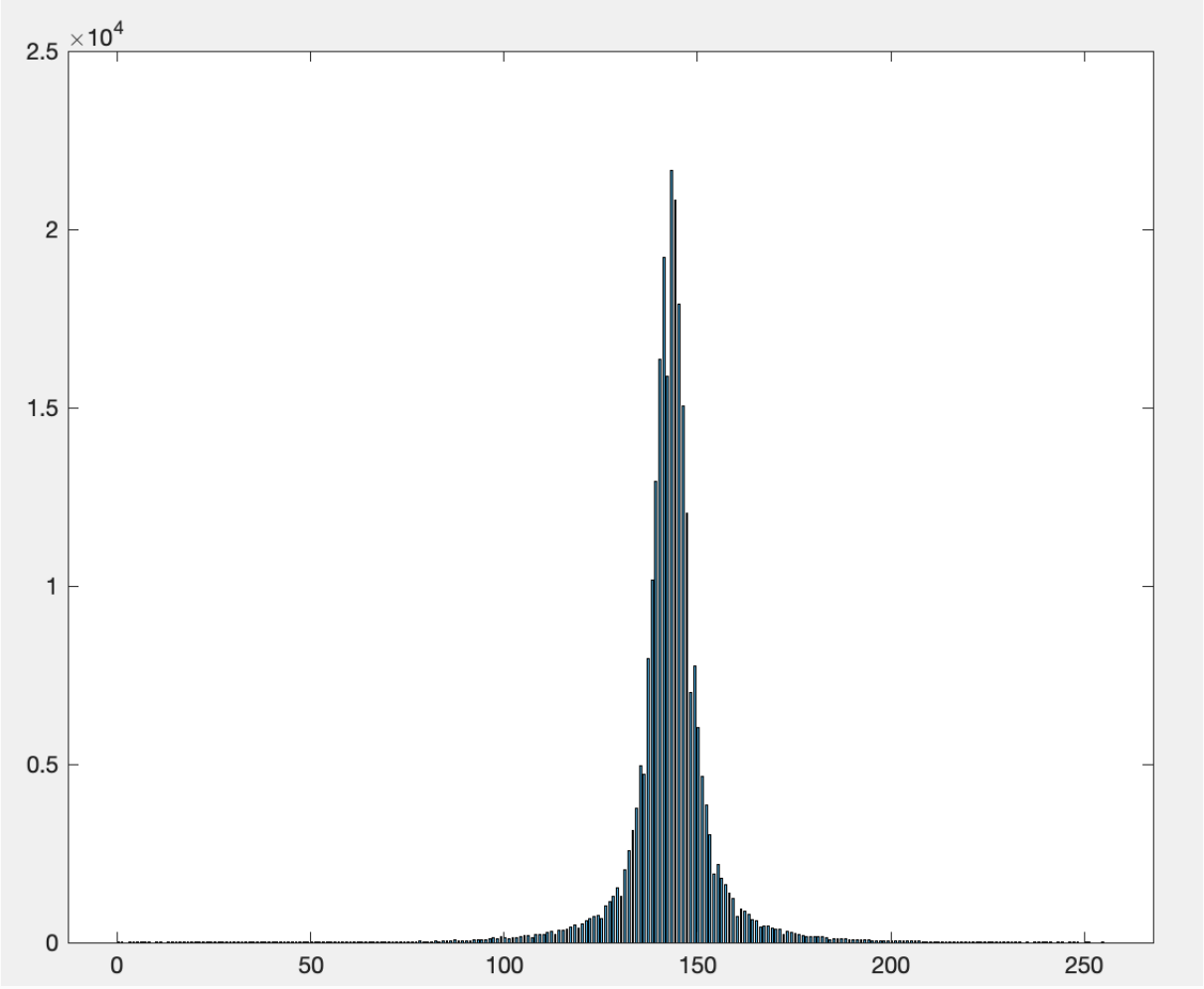
Compare the original image "lenna512.bmp" and the DPCM output image from the view of entropy, which image is easier to be compressed? And Why? (**10**')

Code

```
function im_error = generateDPCM(im)

im = double(im);

% padding
im_padding = padarray(im,[1,1],'replicate');

[m,n] = size(im);

im_decoded = zeros(m,n);

[j,k] = size(im_padding);

for i=2:j-1
    for j = 2:k-1
        im_decoded(i-1,j-1) = (2*im_padding(i,j-1)+ im_padding(i-1,j-1)+...
            2*im_padding(i-1,j))/5;
    end
end

% normalization
im_error = im - im_decoded;

min_value = min(min(im_error));

im_error = im_error + abs(min_value);

max_value = max(max(im_error));

im_error = im_error./max_value*255;

im_error = uint8(im_error);

end
```

```
clear
clc

im = imread('lenna512.bmp');

% generate error image
im_error = generateDPCM(im)

imshow(im_error);

im_error_entropy = my_entropy(im_error)
im_entropy = my_entropy(im)
```

Result




```
im_error_entropy =

    5.0848

im_entropy =

    7.3775
```

Analysis

As we can compare original image "lenna512.bmp" and the DPCM output image from the view of entropy, the entropy of DPCM is lower than that of original image, this means DPCM has litter information than that of original image. So DPCM will be easier to compressed. Because DPCM has litter information to compress and need fewer bytes to compress.

Image DCT and quantization (**50'**)

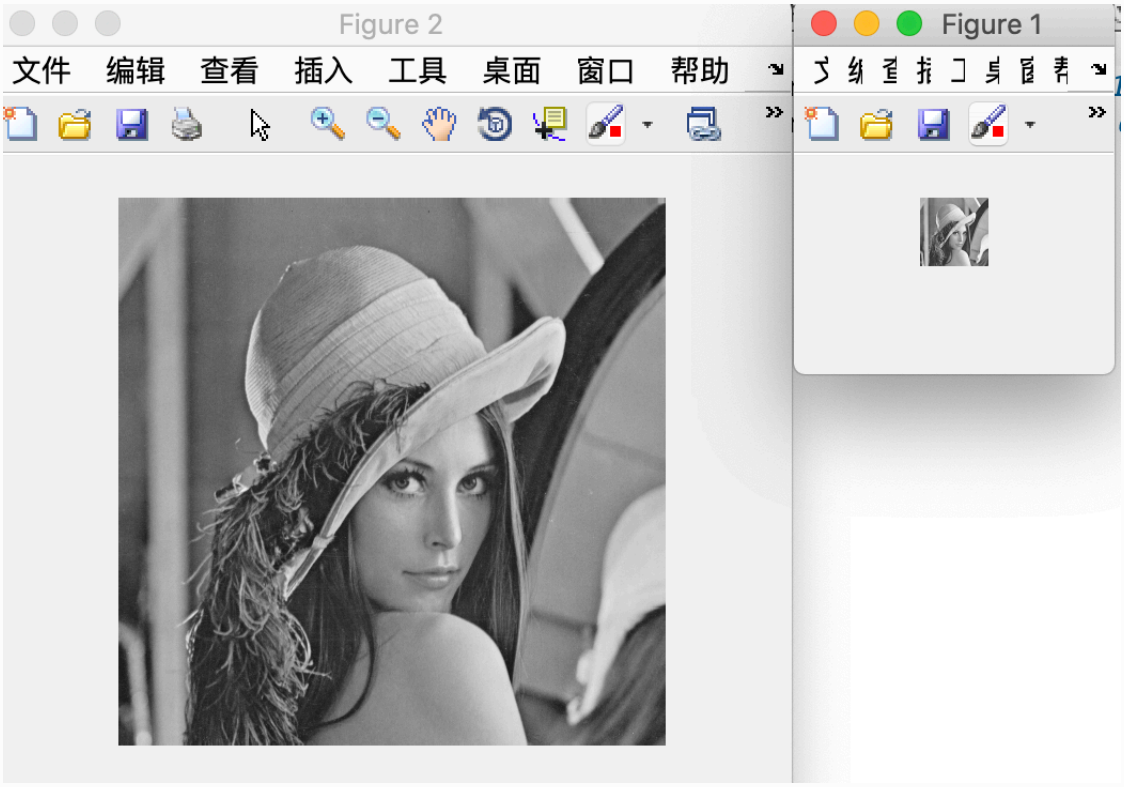
Write a function to do the two-dimensional DCT of all the 8 X 8 non-overlapping blocks of the image **im**, and merge the left-top pixel of all blocks after the DCT transformation to get a smaller image **ims.** Using the image Lenna512.bmp as input image, please show the image **ims** here, and what can you find on comparing it with the original input image and explain your finding (**15**')

Code

```
function ims = generate_ims(im)

im = im2double(im);

my_dct2 = @(block_struct) dct2(block_struct);
im_dct = blkproc(im,[8 8],my_dct2);

% normalization
my_merge = @(block_struct) block_struct(1,1)/max(max(im_dct));
ims = blkproc(im_dct,[8 8],my_merge);

end
```

```
clear
clc

im = imread('lenna512.bmp');

ims = generate_ims(im);
figure(1)
imshow(im2uint8(ims));
figure(2)
imshow(im)
```

Result



Analysis

As we can compare original image and **ims**, ims is smaller than original image and its size is 1/8 as that of original image. The reason for it is that phenomenon is submitted to this formula.

$$F(k) = a(k)\sum_{n=0}^{N-1} x(n)cos[\frac{(2n+1)k\pi}{2N}] \quad k = 0, L, N-1$$

$$a(k) = \begin{cases} \sqrt{\frac{1}{N}}, & k = 0 \\ \sqrt{\frac{2}{N}}, & k \neq 0 \end{cases}$$

When $k = 0$, the formula become this form.

$$F(0) = \sqrt{\frac{1}{N}}\sum_{n=0}^{N-1} x(n)$$

Therefore, it become the summation divided by constant value K . when we can merge left-top pixels, the image will become the small version of original image.

To convert the floating point numbers of the **2D-DCT** into integer numbers, write a function to quantize each 8 X 8 block using the following formula: $round(\frac{b_{ij}}{Sq_{ij}})$, where b_{ij} is the i^{th} row and j^{th} column of the 8 X 8 2D-DCT transformed block, whereas, q_{ij} is the element of the quantization matrix Q_{mat} (equation (1)). And S is a scalar value given by equation (2). (**10**')

Code

```
function quantized_value = my_quantization(im_dct, QP)

Qmat = [16 11 10 16 24 40 51 61;
    12 12 14 19 26 58 60 55;
    14 13 16 24 40 57 69 56;
    14 17 22 29 51 87 80 62;
    18 22 37 56 68 109 103 77;
    24 35 55 64 81 104 113 92;
    49 64 78 87 103 121 120 101;
    72 92 95 98 112 100 103 99];

if QP > 50
    S = (100-QP)/50;
elseif QP <=50
    S = 50/QP;
end


my_qun = @(block_struct) round(block_struct./(S*Qmat));

quantized_value = blkproc(im_dct,[8 8], my_qun);

end
```

Write a function to decompress image **imo**, which should invert the above two steps. (**15**')

Code

```
function decompressed_image = decompress_image(im_dct,QP)

Qmat = [16 11 10 16 24 40 51 61;
    12 12 14 19 26 58 60 55;
    14 13 16 24 40 57 69 56;
    14 17 22 29 51 87 80 62;
    18 22 37 56 68 109 103 77;
    24 35 55 64 81 104 113 92;
    49 64 78 87 103 121 120 101;
    72 92 95 98 112 100 103 99];

if QP > 50
    S = (100-QP)/50;
elseif QP <=50
    S = 50/QP;
end

recover_im_fun = @(block_struct) block_struct.*(S*Qmat);

recover_im = blkproc(im_dct,[8 8], recover_im_fun);

decompresse8 = @(block_struct) idct2(block_struct);
decompressed_image = blkproc(recover_im, [8 8], decompresse8);
end
```

Using the image **Lenna512.bmp** as input image **im** in the above functions, fill the following table with different QP values in the equation (2) for the PSNR between the original input image **im** and the decompressed image **imo**. Please give the comments on this table. (**10**')

Code

```
clear
clc

im = imread('lenna512.bmp');

my_dct2 = @(block_struct) dct2(block_struct);
im_dct = blkproc(im,[8 8],my_dct2);
% count = 1
psnr_list = [];
for i=1:14:99
    im_dct_qun = my_quantization(im_dct, i);

    image = decompress_image(im_dct_qun);
    image = uint8(image);
    %     figure(count);
    %     imshow(image);
    %     title(i);
    %     count = count +1;
    psnr_value = psnr(im,image);
    psnr_list = [psnr_list, psnr_value];
end
```

QP	1	15	29	43	57	71	85	99
PSNR(dB)	17.2018	31.9226	34.1539	35.3327	36.2212	37.3853	39.4034	55.2879

Analysis

As we can see from result, the psnr become larger with the increase of QP. This means the quality of **imo** become better increasingly. This is because magnitude of quantization become larger with the decrease of QP. Therefore, the information of image will lose with the increase of magnitude of quantization.

Please note that you can directly use the Matlab functions of DCT2 and IDCT2 in the above functions.

$$Q_{mat} = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \quad (1)$$

$$\begin{aligned} S &= \frac{100 - QP}{50}, QP > 50 \\ S &= \frac{50}{QP}, QP \leq 50 \end{aligned} \quad (2)$$