Report4

Cheng Xinhao

Dalian University of Technology

2019/4/12

1.The contrast enhancement in DCT

Concept: DFT is a discrete Fourier transform for discrete signals and spectra. DFT is a change in DTFT, in fact, it is to change the continuous time t to nT. The reason is that computer works in a digital environment. It is impossible to see or process continuous signals in reality, only to perform discrete calculations, and to approximate the continuous signal as much as possible in terms of authenticity. So DFT is created for us to analyze signals with tools. Usually we have very few opportunities to use DTFT directly.

DCT is a form of DFT. The so-called "cosine transformation" is in the DTFT Fourier series expansion, if the function to be expanded is a real function, then the Fourier series only contains the cosine term, and then discretizes it (DFT) to derive the cosine transform. It is therefore called the discrete cosine transform (DCT). In fact, DCT is a subset of DFT. DCT is used for voice and image processing.

Formula:

One-dimensional transformation:

$$C(u) = \alpha(u)\sum_{x=0}^{N-1} f(x)cos[\frac{\pi(2x+1)u}{2N}]$$

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} & u = 0 \\ \sqrt{\frac{2}{N}} & u \neq 0 \end{cases}$$

Inverse transformation:

$$f(x) = \sum_{u=0}^{N-1} \alpha(u)C(u)cos[\frac{\pi(2x+1)u}{2N}]$$

Two-dimensional transformation:

$$C(u,v) = \alpha(u)\alpha(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)cos[\frac{\pi(2x+1)u}{2N}]cos[\frac{\pi(2y+x)v}{2N}]$$

Inverse transformation:

$$f(x,y) = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} \alpha(u)\alpha(v)cos[\frac{\pi(2x+1)u}{2N}]cos[\frac{\pi(2y+1)v}{2N}]$$

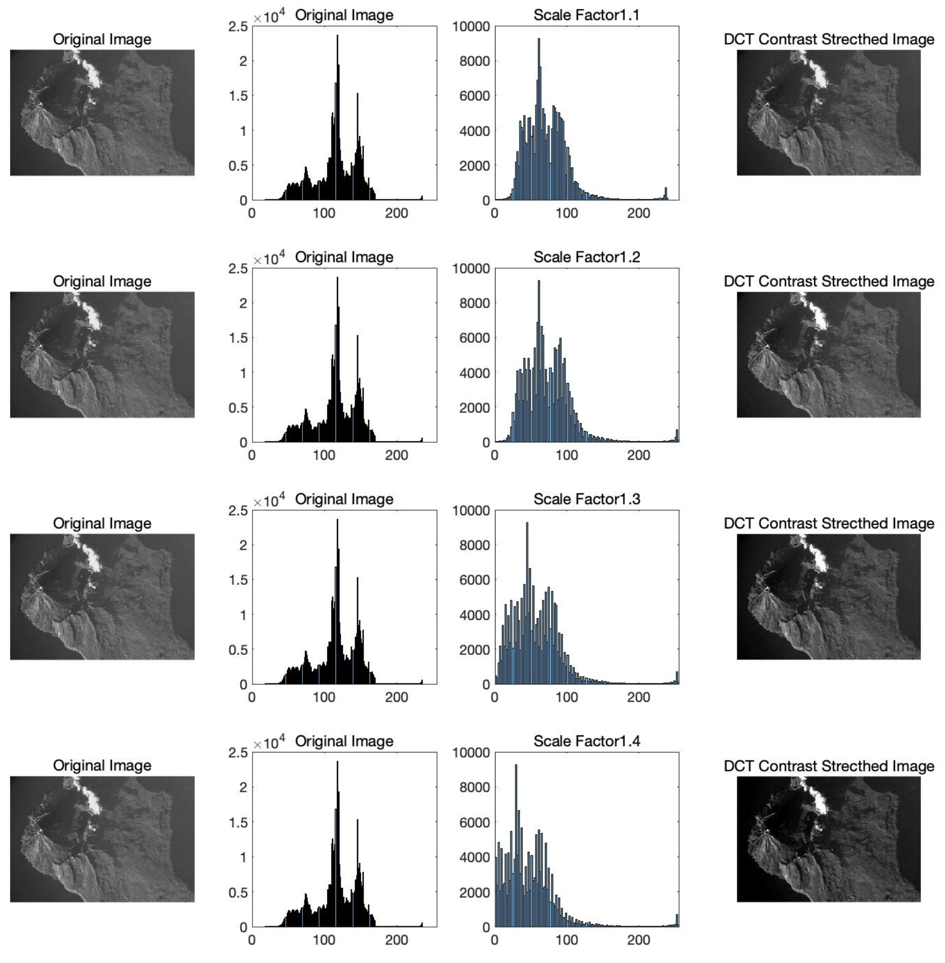
Use DCT to do contrast enhancement:

the procedure:

1.first, pick the Y channel of the image

2.take the top&left pixel as the dc
3.take all the pixels like the buf
4.use the dc and buf to calculate **VarDC** & **VarAC**
5.choose a coefficient called ScaleAc
6.use the ScaleAc to multiply the image and change with idct2()
7.choose the smaller one from $b = min(min(ims))$ and
$ts = max(max(ims)) - 255$ as s
8.use the image to minus the s
9.perform the result

Output the result



2.The Huffman coding

The Huffman coding algorithm is based on a binary tree to construct a coding compression structure, which is a classic algorithm in data compression. The algorithm re-encodes the characters based on the frequency of occurrence of the text characters.

The major process(with codes)

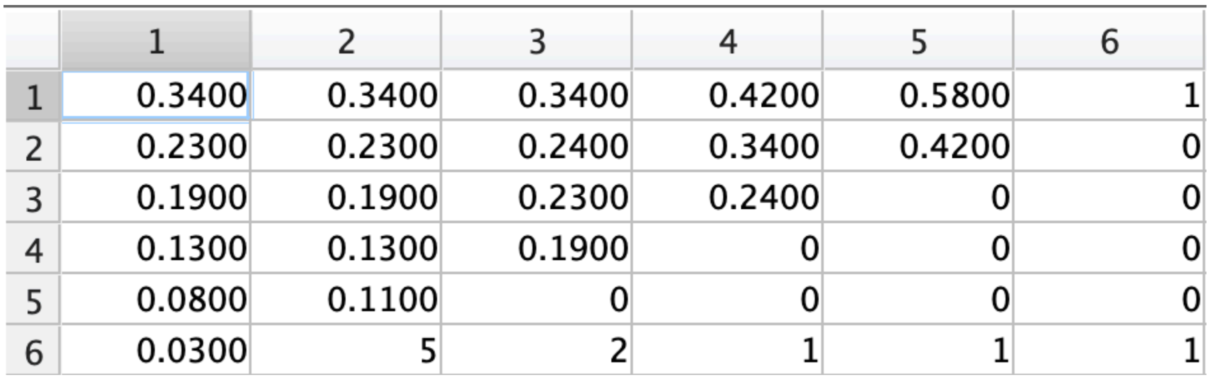
```
N=6;%N data
SiganlSourcewithoutNormilization = [3,8,13,19,23,34];
sum=0;
for n=1:N
    sum=sum+SignalSourcewithoutNormilization(n);
end
SignalSource=SignalSourcewithoutNormilization./sum;
%Normalization to (0,1)
SignalSource = sort(SignalSource,'descend');
%sort
```

input the data, then normalize to (0,1). and sort the data

```
SortingMatrix=zeros(N,N);
for i=1:N
    P=sort(SignalSource,'descend');%descending sort
    SortingMatrix(i,1)=P(i);
end
rear=SortingMatrix(i,1)+SortingMatrix(i-1,1);
%Minimum probability of combining
P(N-1)=rear;P(N)=0;
P=sort(P,'descend');
t=N-1;

for j=2:n%Generate additional columns of the code table
        for i=1:t
            SortingMatrix(i,j)=P(i);
        end
        if t>1
            K=find(P==rear);
            SortingMatrix(N,j)=K(end);
%The last element in each column is used to record the location where
 the merge occurred.
            rear=(SortingMatrix(t-1,j)+SortingMatrix(t,j));
            P(t-1)=rear;
            P(t)=0;
            P=sort(P,'descend');
            t=t-1;
```

```matlab
        else
            SortingMatrix(n,j)=1;
        end
    end
```

make a N*N matrix to code the huffman tree. add the smallest two data in the matrix. and remember the place of new element.do this process until all the data have been coded



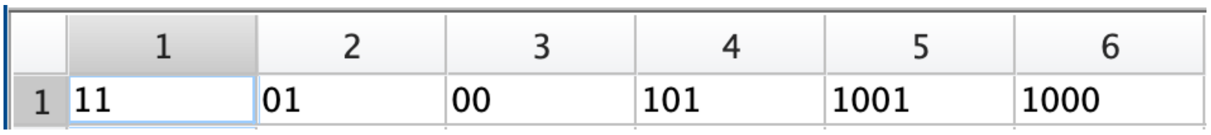
	1	2	3	4	5	6
1	0.3400	0.3400	0.3400	0.4200	0.5800	1
2	0.2300	0.2300	0.2400	0.3400	0.4200	0
3	0.1900	0.1900	0.2300	0.2400	0	0
4	0.1300	0.1300	0.1900	0	0	0
5	0.0800	0.1100	0	0	0	0
6	0.0300	5	2	1	1	1

the matrix of huffman tree

```matlab
%%
    %Code table of sorted elements
    m=3;
    s1=str2sym('[2,1]');
    s2=s1;
    %Code table, with 1 for 0 and 2 for 1, because 0 will not be recorded because it is omitted.
    %Starting from the third last column
    for i=N-2:-1:1
        p=SortingMatrix(N,i+1);
        if p==1
            s2(1:m-2)=s1(2:m-1);
        else if p==m
            s2(1:m-2)=s1(1:m-2);
        else if p==2
            s2(1)=s1(1);
            s2(2:m-2)=s1(3:m-1);
        else
            s2(1:p-1)=s1(1:p-1);
            s2(p+1:m-2)=s1(p+1:m-2);
        end
        s2(m-1)=[char(s1(p)),'2'];
        s2(m)=[char(s1(p)),'1'];
        m=m+1;
        s1=s2;
    end
    L=zeros(1,N);
    for i=1:N
        [~,rear]=size(char(s2(i)));
        L(i)=rear;
    end

    %Convert the code table to 0 and 1 output
    array=zeros(1,N);
    array(1:n)=s2(1:N);
    String=[];
    for i=1:n
        s=num2str(array(i));
        s=strrep(s,'1','0');
        s=strrep(s,'2','1');
        if i==1
            String=s;
        else
            String=[String,' ',s];
        end
    end

  String = regexp(String,' ','split');%output the code table
```

code the table of huffman tree

the table of huffman code



	1	2	3	4	5	6
1	11	01	00	101	1001	1000

3.JPEG compression

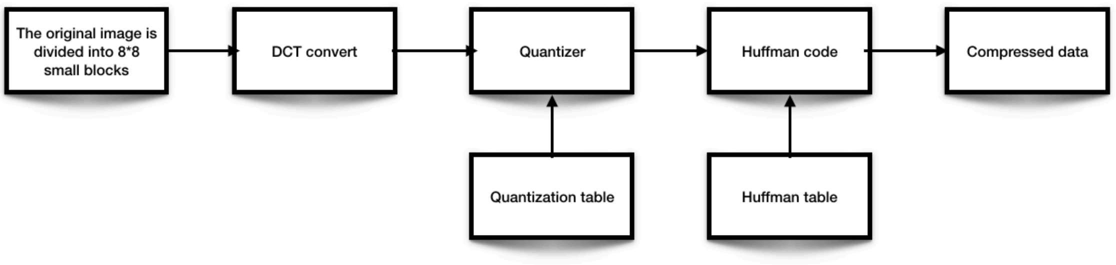
Procedure:

1.Divide the original image into 8 * 8 small blocks with 64 pixels in each block.

2.Perform a DCT transform on each 8 * 8 block in the image. After the image of 8 * 8 is transformed by DCT, the low frequency components are concentrated in the upper left corner, and the high frequency components are distributed in the lower right corner.

3. Use a quantization table to suppress high frequency variables. The quantization operation is to divide a value by the corresponding value in the quantization table. Since the value in the upper left corner of the quantization table is small, the value in the upper right corner is large, which serves to maintain the low frequency component and suppress the high frequency component. Converts the color space of a color image from RGB to YUV when compressing. The Y component represents the luminance information, and the UV component represents the color difference information. (In contrast, the Y component is more important. We can use fine-grained Y and coarse-grained UV to further increase the compression ratio. So there are usually two quantization tables, one for the standard luminance quantization table for Y and one for the standard color quantization table for UV.)

4.Use Huffman coding to rearrange the data to produce an array of integers.



The major process(with codes)

```
im1 = imread('dog.jpg');
figure;
subplot(1,2,1);
imshow(im1);
title('original picture');
im1 = rgb2ycbcr(im1);
im11 = im1(:,:,1);%Y channel
im12 = im1(:,:,2);
```

```
im13 = im1(:,:,3);
im2y = im2double(im11);
im2u = im2double(im12);
im2v = im2double(im13);
```

first, input the picture and convert to YCBCR color, pick each channel.

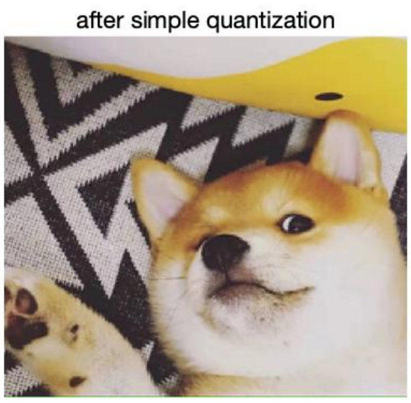
```
%compression of  Y channel
T = dctmtx(8);
Imdcty = blkproc(im2y,[8 8],'P1*x*P2', T , T');
mask = [1 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0];

im3y = blkproc(Imdcty,[8 8] ,'P1.*x',mask);% dct convert
Imidcty = blkproc(im3y , [8 8] , 'P1*x*P2' , T' , T);
%quantization of the 8*8 matrix
Imidcty = im2uint8(Imidcty);%inverse dct
im1(:,:,1) = Imidcty;
```

then, do the dct convert with 8*8 matrix, and use the ‘**mask**’ matrix to keep the top-left information(always valuable information)

***The Cb and Cr channel is similar(but take half sample)**

```
im1 = ycbcr2rgb(im1);
subplot(1,2,2);
imshow(im1);title('after simple compression');
```

Finally we put the three channel together and convert to RGB color, perform the result.(Did not lose too much information)

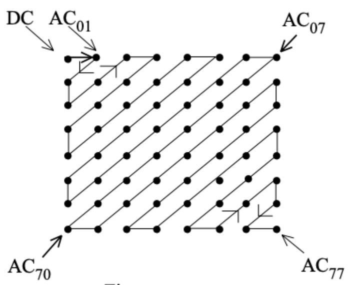


Then we can use the z sequence to compress the data of image

```
function [y] = Zig(x,flag)
[xm,xn] = size(x);
%do the Zig-Zig sequence(image to jpeg)
order = [1 9  2  3  10 17 25 18 11 4  5  12 19 26 33  ...
    41 34 27 20 13 6  7  14 21 28 35 42 49 57 50  ...
    43 36 29 22 15 8  16 23 30 37 44 51 58 59 52  ...
    45 38 31 24 32 39 46 53 60 61 54 47 40 48 55  ...
    62 63 56 64];% the sequence
y = im2col(x,[8 8] , 'distinct');
```

make the order

of ‘Z’ in the 8*8 matrix



Zig–zag sequence

```
xb = size(y,2);
y = y(order , :);
eob = max(y(:)) + 1;                  % set a flag of matrix
r = zeros(numel(y) + size(y, 2), 1);
count = 0;
for j = 1:xb
    i = max(find(y(:, j)));            % find the last non-zero ele
    if isempty(i)
        i = 0;
    end
    p = count + 1;
    q = p + i;
    r(p:q) = [y(1:i, j); eob];
    count = count + i + 1;             % counter
end

r((count + 1):end) = [];              % delete the ‘0’
[r1,r2]=size(r);
```

take the ‘1’ and delete the ‘0’ and set a struct of 8*8 matrix. keep the struct for decompression

Then use the order and mask to do the de-zig and inverse of the image.

```matlab
%%
rev = order;                                      % Computational reverse order
for k = 1:length(order)
    rev(k) = find(order == k);
end

xb = double(y.numblocks);                         % Number of blocks
sz = double(y.size);
xn = sz(2);                                       % col
xm = sz(1);                                       % row
x = y.r;
eob = max(x(:));

z = zeros(64, xb);   k = 1;
for j = 1:xb
    for i = 1:64
        if x(k) == eob
            k = k + 1;
            break;
        else
            z(i, j) = x(k);
            k = k + 1;
        end
    end
end
```

use the revere order to recover the 8*8 matrix after quantization

```matlab
%%
T=dctmtx(8);                                                %8*8 dct matrix
z = z(rev, :);                                              % recover the sequence

x = col2im(z, [8 8], [xm xn], 'distinct');                  % matrix
if y.flag==1
    x = blkproc(x, [8 8], 'x .* P1', mask);                    % quantization inv

else
    x = blkproc(x, [8 8], 'x .* P1', b);
end
x = blkproc(x, [8 8], 'P1 * x * P2', T', T);               % DCT inverse
```

do the inverse process we have done in the quantization

Finally we perform the origin image with the image after “compression” and “decompression”

4.DCT Wiener filter

Wiener filter

1. First, use dct2() to do DCT convert to the image

2. take the $log10()$ of the absolute value of the image

3.the **Valid value** is always in the low frequency, the **Noise** is always in the high frequency. So take the high corner as a sample of the noise.

4. calculate the NoiseVariance with the noise sample sigma($NoiseVariance = mean(mean(sigma))$)

5.choose a $beta$ to multiply the NoiseVariance coefficient.

6.$SignalVariance = imd.*imd + 0.001$; (the image data)

7.$WienerFilter = 1 + (NoiseVariance./SignalVariance)$; (the WienerFIiter)

8.use the WienerFilter to multiply the image and perform the result

The major process(with codes)

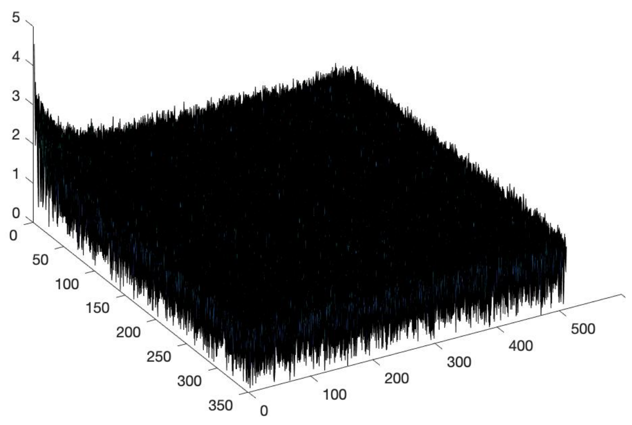
```
im1 = imread('showimage.jpeg');
im2 = imread('showimage1.jpeg');


im1_1 = rgb2ycbcr(im1);
im2_1 = rgb2ycbcr(im2);
im1_2 = im1_1(:,:,1);
```

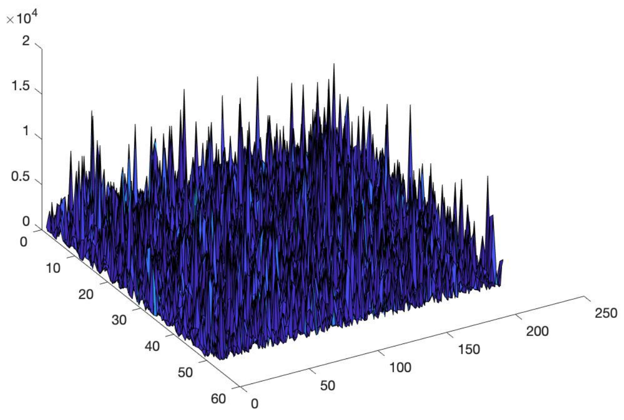
```
im2_2 = im2_1(:,:,1);


im1_3 = imnoise(im1_2,'gaussian',0,0.02);
imshowpair(im1_2,im1_3,"montage");
```

make the gaussian noise

```
%perform 2D DCT image
im1_4 = dct2(im1_3);
ima = log10(abs(im1_4)+1);
s1 = size(im1_4);
x = [1:s1(1)];
y = [1:s1(2)];
[x,y] = meshgrid(x,y);
figure;
surface(x,y,ima');
view(60,45);

%pick the high frequency corner as sample
sigma = im1_4(300:end,300:end).*im1_4(300:end,300:end);
s3 = size(sigma)

xs3 = [1:s3(1)];
ys3 = [1:s3(2)];

[xs3,ys3] = meshgrid(xs3,ys3);

figure
surface(xs3,ys3,sigma')
view(60,45)
```

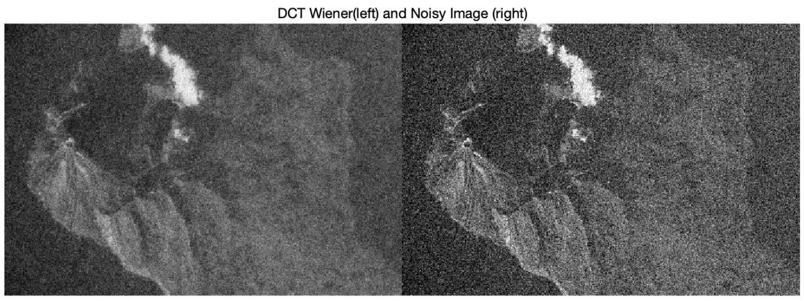
perform the 2d DCT and choose the high frequency corner as the noise sample





```
NoiseVariance = mean(mean(sigma));


beta = 2.0
NoiseVariance = beta*NoiseVariance;

% now use this value ns  for wiener filter

SignalVariance = im1_4.*im1_4 + 0.001;
WienerFilter = 1 + (NoiseVariance./SignalVariance);
WienerFilter = 1./WienerFilter;

% Now apply the Wiener Filter to the signal DCT coefficients

FilteredImageDCT = im1_4.*WienerFilter;

% Do the inverse DCT transform to go back to Gray Value

FilteredImage = idct2(FilteredImageDCT);

% Get rid of all values below 0 and highr than 255

imo = uint8(FilteredImage);

% Display filtered Image



figure;

imshowpair(imo,im1_3,'montage');
title('DCT Wiener(left) and Noisy Image (right)')
```

calculate the noise coefficient and determine the wiener filter. perform the result



5.Comparison in Wiener filter and Nonlocal means

Wiener filter is an optimal estimator for stationary processes based on minimum mean square error criteria. The mean square error between the output of this filter and the desired output is minimal, so it is an optimal filtering system. It can be used to extract signals that are contaminated by stationary noise.

$$G(f) = \frac{1}{H(f)}[\frac{|H(f)^2|}{|H(f)^2| + 1/SNR(f)}]$$

$$\frac{d(f)}{dG(f)} = G^*(f)N(f) - H(f)[1 - G(f)H(f)]^*S(f) = 0$$

Nonlocal means is a Non-local averaging algorithm,It is a method of smoothing the mean value around a target pixel. Non-local mean filtering means that it uses all the pixels in the image, and these pixels are weighted averaged according to some similarity. Filtered image with high definition and no loss of detail

In the same image, the regions with the same properties are classified and weighted to obtain the denoised image, and the noise reduction effect should be better. The algorithm uses redundant information that is ubiquitous in natural images to denoise. It takes advantage of the entire image for denoising. That is, the similar regions are searched for in the image block unit, and then the regions are averaged to better filter out the **Gaussian noise** in the image.

$$NL[v](i) = \sum_{j \epsilon I} w(i,j)v(j) \text{ (w is the weight of each pixel)}$$

the different result of the same original image

Wiener(left) and Nonlocal means (right)