COMP40600: Multimedia Security

Individual Assignment

Rui Tu (UCD: 13206398)

February 26, 2018

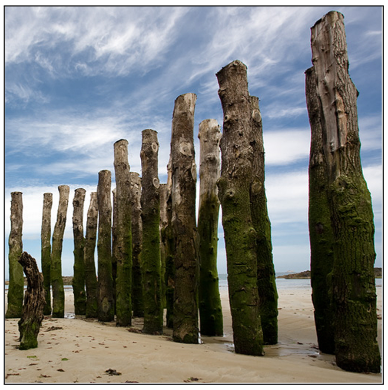
Table of Contents

1 First Assignment: Image Processing and Coding

1.1 Image Filters

In this question, we do transformation with three colour pixel channels (R, G, B) independently. I create a function called 'myfilter'. It takes the colour pixel matrix and the vector of transformation function as the parameters. After normalising the pixel value between $[0 \ldots 1]$, i convert it into a vector. Then use the function polyval to perform polynomial multiplication. Finally reshape it into the original size of the matrix.

```
function F = myfilter(M, v)
%filter function
M_norm = M ./ 255; %normalisation
vec_M = M_norm(:);
tran = polyval(v,vec_M);
F = reshape(tran,640,640);
```


(a)

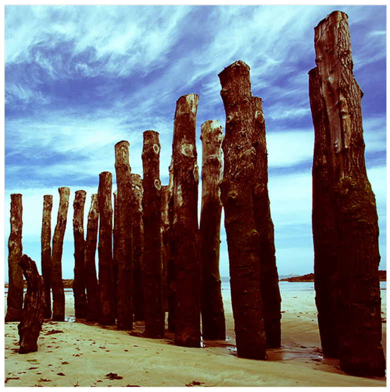

(b)

Figure 1: Original image (a), filtered result (b).

For the vignetting effect, i write a loop to get the distance and the corresponding C value for all the pixels. Then overlay the filtered image with radial gradient image C using the function 'myoverlay'.

```
C_over = zeros(640,640);
        for i = 1:640
            for j = 1:640
                dist = ((i - 320)^2+(j - 320)^2)^.5 / 640;
                if(dist < (1/3))
                    C_over(i,j) = 0.5;
                elseif(dist ≤ (2/3))
                    C_over(i,j) = (1 + cos(3*pi* (dist - 1/3)) ) / 4;
                else C_over(i,j) = 0;
                end
            end
        end
        imshow(C_over);
        pause;
```

Radial Gradient Image

```
function F = myoverlay(C, M)
%overlay function for question 1.1
F = zeros(640,640);
for i = 1:640
    for j = 1:640
        if M(i,j) < 0.5
            F(i,j) = 2 * M(i,j) * C(i,j);
        else
            F(i,j) = 1 - 2 * (1-M(i,j)) * (1-C(i,j));
        end
    end
end
```

myoverlay.m

Parameter C is the radial gradient matrix and the parameter M is the filtered image matrix. This function achieves the blending overlay effect with each colour channel following the given formula.

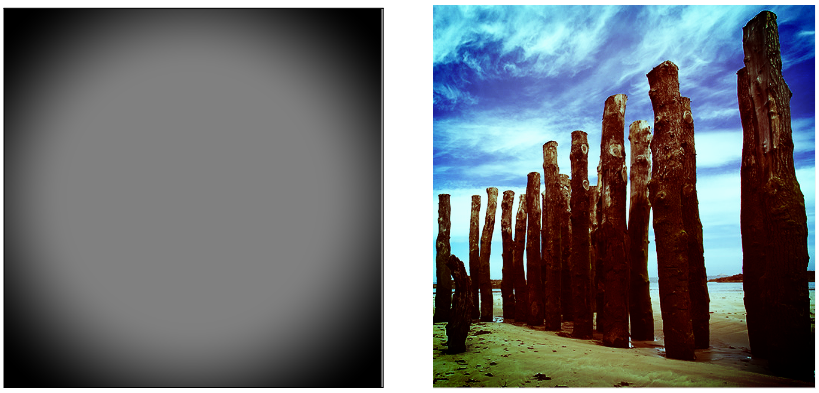

(a) (b)

Figure 2: Overlay image (a), filtered image with blending overlay (b).

1.2 Block Transform

```
Image = double(imread('lena.png'));
        T_DCT = dctmtx(8);
        DCT_tran = @(block) (T_DCT * block.data * T_DCT');
        Image_DCT = blockproc(Image, [8,8], DCT_tran);
        Image_DCT(1:8,1:8)
        min1 = min(Image_DCT);
        min1 = min(min1) %get the range of the first matrix
        max1 = max(Image_DCT);
        max1 = max(max1)
        Image2 = Image - 128;
        Image2_DCT = blockproc(Image2, [8,8], DCT_tran);
        Image2_DCT(1:8,1:8)
        min2 = min(Image2_DCT); %get the range of the second matrix
        min2 = min(min2)
        max2 = max(Image2_DCT);
        max2 = max(max2)
```

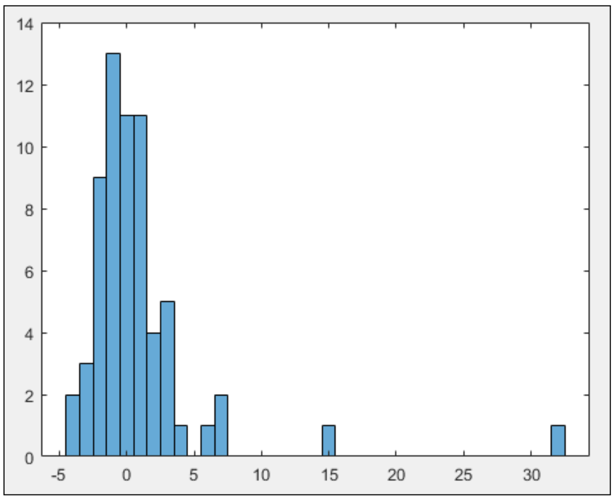
The code above computes the DCT coefficient matrix of 8×8 block of Lena image using block processing. The 'DCT_tran' is the function for DCT matrix transformation. 'Image_DCT' is the DCT coefficient matrix for the original image. 'Image2_DCT' is the DCT coefficient matrix after subtracting 128 from all pixels values. After finding the minimum and maximum value of both matrix, it is observed that the range of DCT values for the original matrix is $[-479 \dots 1664]$ and the range after subtraction is $[-870 \dots 640]$. The reason for this subtraction operation is that the original pixel values level from 0 to 255. It always has a positive range and the midpoint is 128. After the subtraction, the value centred around 0 and the range becomes $[-127 \dots 127]$. Then the range of the output of the DCT coefficient values is $[-1024 \dots 1023]$. Besides, it is equivalent to subtract 1024 after the transformation. But doing subtraction operation on this step is easier to perform.

1.3 Quantization

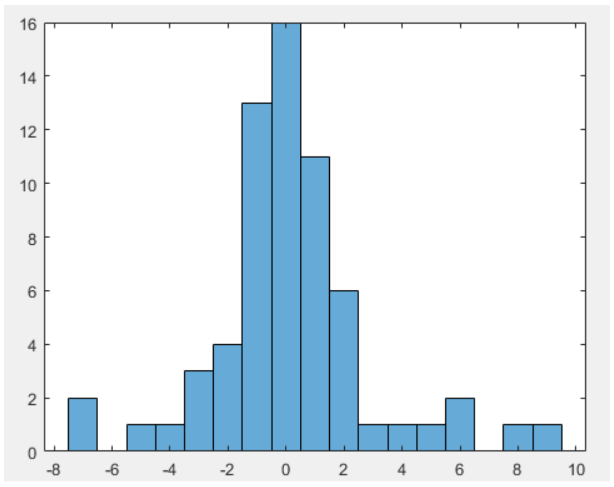
```
function C = quantize_dct(B, gamma)
% the function to perform quantization of 8*8 block
Q = [16 11 10 16 24 40 51 61
            12 12 14 19 26 58 60 55
            14 13 16 24 40 57 69 56
            14 17 22 29 51 87 80 62
            18 22 37 56 68 109 103 77
            24 35 55 64 81 104 113 92
            49 64 78 87 103 121 120 101
            72 92 95 98 112 100 103 99];

if gamma < 50
    alpha = 5000 / gamma;
elseif (gamma ≥ 50) && (gamma ≤ 99)
    alpha = 200 - 2*gamma;
else alpha = 1;
end

Q_qual = zeros(8,8);
for i = 1:8
    for j = 1:8
        Q_qual(i,j) = round((alpha * Q(i,j) + 50) / 100);
    end
end
C = round(B./Q_qual);
```

quantize_dct.m

The 'quantize_dct' function perform the quantization of an 8×8 block. The input parameter B is the block and gamma is the quality factor. The function is called by `FUN = @(block)(quantize_dct(block.dat` where 20 is the quality factor. In this step, we use quantization to store the values using less space wit some loss in accuracy by dividing the DCT matrix with quantization matrix. In the block, the DC data which is the first value of the block is much larger than the other AC data in the block. The DC data is well saved during quantization operation because the values on the top-left corner of the quantization matrix is designed to be smaller than the values from the bottom-right corner. The two screenshots below are the histogram of the DCT coefficients of the first and the second block after quantization.



(a)



(b)

Figure 3: Histograms of the first and second block.

The range of those blocks depends on the quality factor. The larger the quality factor is, the large the range will be. For example, if the gamma value is set to 100. Then the theoratical range of DCT coefficients after quantization is $[-1024 \ldots 1023]$, which is the original range. So it has almost no loss on the quality. The actually range observed in this case is $[-870 \ldots 640]$. If the gamma value is set to 20, then the range shrinks greatly. The actually range observed is $[-21 \ldots 20]$. The smaller quality factor is selected, the more zero values appear in the DCT coefficient matrix, which means more information have lost due to the compression. In this way, we can control the quality of compression.

1.4 Zigzag Reordering

```
function zigzag = zigzag_reorder(Q)
%the function performs zigzag reordering
ZIGZAG = [ 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];
q = reshape(Q, 1, 64);
zigzag = q(ZIGZAG);
```

zigzag_reorder.m

I perform the zigzag reordering by giving a index vector. After reshape the 8×8 block matrix into a vector, we use the ZIGZAG index vector to rearrange the order of the values inside. For the same image, the DC coefficients from adjacent blocks are sometimes correlated or varies slowly where values from non-adjacent block have almost no correlation adn may have a big difference. There are also no correlation for the blocks from different image. This means that we can coding the DC term of a given block as the difference from the previous DC term. A smaller value will be used in this way with fewer bits. The reason for perform zigzag reordering is that the DCT coefficient on the bottom-right corner after quantization is very likely to become zero because of a large value in the quantization matrix. The non-zero value in the matrix consumes more space. After zigzag reordering, the last few values are very likely to be all zero. Consecutive zero values in the vector use less space in storing. So we convert the matrix or the singal into zigzag order.

```
function v_rm = rm_zeros(v)
%the function removing trailing zeros
zero_num = numel(v) - find(v,1,'last');
v_rm = v(1:length(v)-zero_num);
```

rm_zeros.m

Te rm_zeros function takes a vector as the input and find the number of trailing zeros in it. Then it remove the trailing in the vector.

1.5 DPCM on DC component

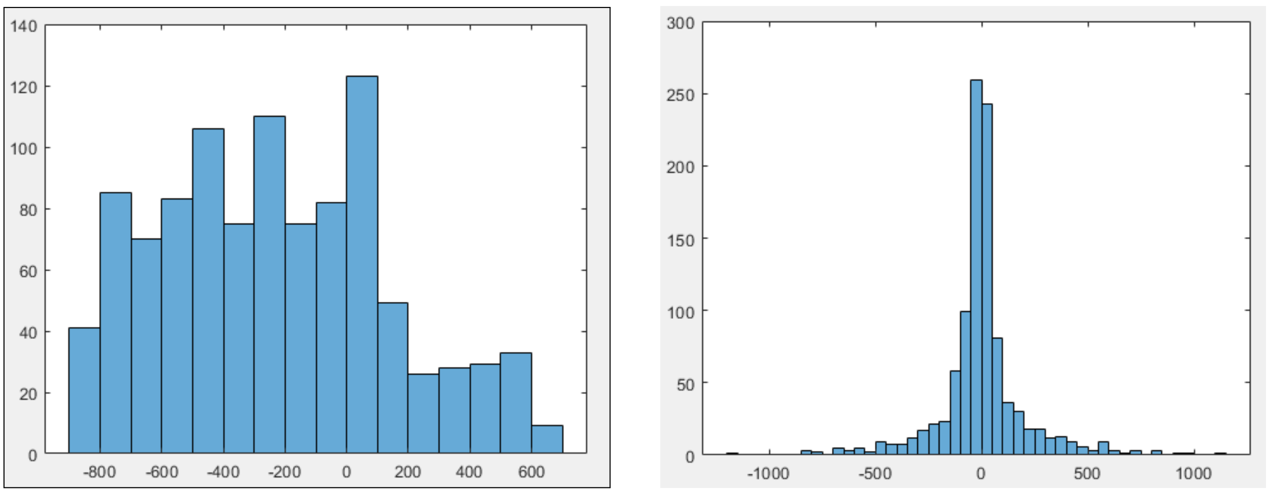


Figure 4: Histogram of DC coefficient (a), Histogram of the difference of DC coefficient (b).

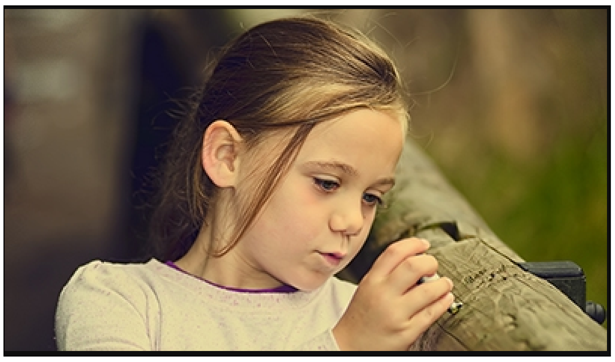
As can be seen from the two screenshots of histogram, the value of DC coefficients varies from 800 to 700 and many of them are in a very large number. The size for storing these numbers is very large. However, in the pic(b) we can find that most of the difference are concentrated around zero. Only very few number of difference value are above one hundred. This means that the average size for storing these amplitude are much smaller than using the DC coefficient directly. From the question above we have already found that there are some correlations of coefficient between adjacent blocks. So coding the DC coefficient of a given block as the difference from the previous DC coefficient typically produces a very small number that can be stored in a fewer number of bits.

1.6 Evaluation (optional)

I haven't try by myself but have find some information on the internet. The sirv.com has results show that:

- ☐ A 92 JPEG quality gives a very high-quality image while gaining a significant reduction on the original file size.
- ☐ A 85 JPEG quality gives a greater file size reduction with almost no loss in quality.
- ☐ A 75 JPEG quality and lower begins to show obvious differences in the image, which can reduce user experience.

And I have also try to vary the quality factor for the demo picture on the website. When the quality factor change to 60, difference can be found comparing to the original picture. When the factor comes to 40, the difference is obvious. When the factor is below 20, the picture becomes very blurred.


(a)

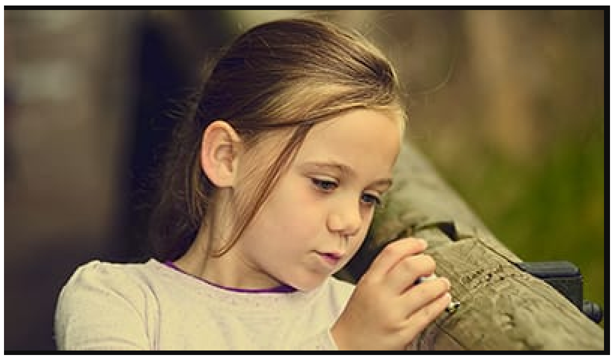
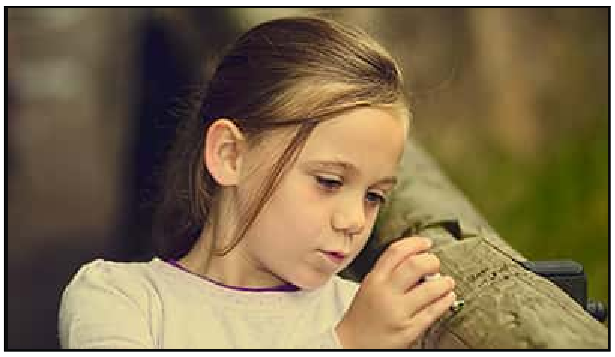

(b)

Figure 5: original picture (a), 80 quality (b).


(a)

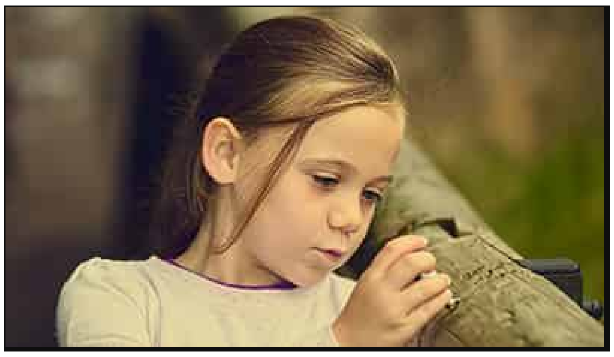
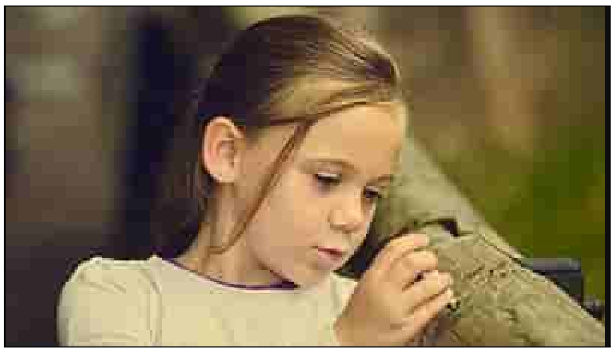

(b)

Figure 6: 60 quality (c), 40 quality (d).


(a)

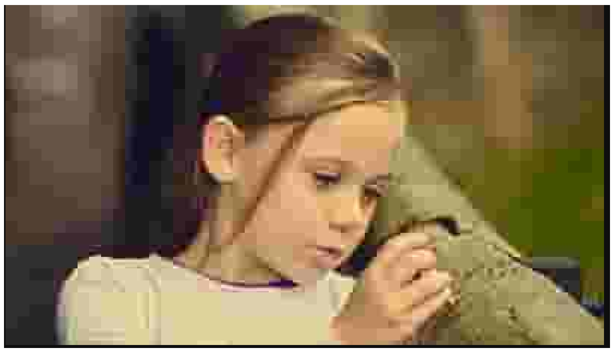

(b)

Figure 7: 20 quality (e), 10 quality (f).