Table of Contents

MMSP2 - Lab 5 1
1) Load the image 'lena512color.tiff' and extract the luminance component 1
2) Consider the first 8x8 pixels block and compute its 8x8 DCT coefficients. 1
3) JPEG baseline coding - estimate the PSNR and display the reconstructed image 3
4a) Reconstruct the image using only the DC component of the DCT - estimate the PSNR and display the reconstructed image 4
4b) Reconstruct the image using only one AC component of the DCT 6
4a) consider block of size 8x8 6
4b) compute the DCT 6
4c) keep only coeff (k1, k2) 6
4d) reconstruct the image from quantized coefficients 6
5) Consider blocks of dimension 8x8 and estimate the correlation matrix 7
6) Perform KLT coding - estimate the PRNR and display the reconstructed image 7

MMSP2 - Lab 5

```
Exercise 1 - DCT/KLT comparison
```

```
clear
close all
clc
```

1) Load the image 'lena512color.tiff' and extract the luminance component

```
im = imread('lena512color.tiff');
R = double(im(:,:,1));
G = double(im(:,:,2));
B = double(im(:,:,3));

Y = 0.299*R + 0.587*G + 0.114*B;

[Nr,Nc] = size(Y);
```

2) Consider the first 8x8 pixels block and compute its 8x8 DCT coefficients.

Use different methods and compare them.

```
N = 8; % dimension of the image block
K = 8; % dimension of the projection space

block = Y(1:N,1:N);
block_dct = zeros(K,K,4);

% Method 1: use the separability property of DCT
% Define transform matrix using the given equation
```

```
% Apply transform matrix by rows and by columns

a = sqrt(2/N);

Tdct1 = zeros(K,K);
for k = 1:K
    if k == 1
        a = sqrt(1/N);
    end

    for l = 1:N
        Tdct1(k,l) = a*cos((2*l-1)*pi*(k-1)/(2*N));
    end

end

block_dct(:,:,1) = Tdct1 * block * Tdct1';

% Method 2: as method 1 but using the function dctmtx() to build the
 transform matrix
Tdct2 = dctmtx(K);
block_dct(:,:,2) = Tdct2 * block * Tdct2';

% Method 3: use function dct2()
block_dct(:,:,3) = dct2(block);

% Method 4: define the transformation g(n1,n2,k1,k2) and apply it
g = zeros(N,N,K,K);
for n1 = 1:N
    for n2 = 1:N
        for k1=1:K
            for k2=1:K
                a_k1 = sqrt(2/N);
                if k1 == 1
                    a_k1 = sqrt(1/N);
                end
                a_k2 = sqrt(2/N);
                if k2 == 1
                    a_k2 = sqrt(1/N);
                end
                g(n1,n2,k1,k2) = a_k1*cos((2*(n1-1)+1)*pi*(k1-1)/
(2*N))*a_k2*cos((2*(n2-1)+1)*pi*(k2-1)/(2*N));
                block_dct(k1,k2,4) = block_dct(k1,k2,4) +
 g(n1,n2,k1,k2)*block(n1,n2);
            end
        end
    end
end


% Compute the MSE between coefficients in the different cases
block_mse = zeros(4,4);
for idx1 = 1:4
    for idx2 = 1:4
```

```
        block_mse(idx1,idx2) = (sum(sum((block_dct(:,:,idx1)-
block_dct(:,:,idx2)).^2)))/(N.^2);
    end
end

disp('MSE');
disp(block_mse);

% Are really all these methods equal?
% Yes, the small difference is only due to numerical limitations
```

```
MSE
         0    0.8852    0.8852    0.8852
    0.8852         0    0.0000    0.0000
    0.8852    0.0000         0    0.0000
    0.8852    0.0000    0.0000         0
```

3) JPEG baseline coding - estimate the PSNR and display the reconstructed image

```
load('Qjpeg.mat'); % quantization matrix
Q = 1; % scaling factor (fine --> coarse)
Qmatrix = QJPEG * Q;

% store the quantized symbols to compute entropy later on
[h,w] = size(Y);
num_block = h/N*w/N;
jpeg_symbols = zeros(K,K,num_block);
block_idx = 1;

Y_jpeg = zeros(size(Y));
% loop over each 8x8 pixels block (no overlap)
for r = 1:size(Y,1)/8
    for c = 1:size(Y,2)/8
        block_r_idxs = (r-1)*N+1:r*N;
        block_c_idxs = (c-1)*N+1:c*N;

        % 3a) consider block of size 8x8
        block = Y(block_r_idxs,block_c_idxs);
        % 3b) compute the DCT (function dct2())
        block_dct = dct2(block-127);
        % 3c) threshold quantization
        block_dct_coeff = round(block_dct./Qmatrix);
        block_dc_q = block_dct_coeff.*Qmatrix;

        jpeg_symbols(:,:,block_idx) = block_dct_coeff;
        block_idx = block_idx + 1;

        % 3d) reconstruct the image from quantized coefficients
 (function idct2())
        block_idct = idct2(block_dc_q);
```

```
        Y_jpeg(block_r_idxs,block_c_idxs) = block_idct + 127;
    end
end

% display image and compute PSNR and entropy

% figure()
% subplot(1,2,1);
% imshow(uint8(Y));
% title('Original');
%
% subplot(1,2,2);
% imshow(uint8(Y_jpeg));
% title('JPEG');

mse_jpeg = mean((Y(:)-Y_jpeg(:)).^2);
PSNR_jpeg = pow2db(255.^2/mse_jpeg);

disp(['PSNR JPEG: ' num2str(PSNR_jpeg)])

jpeg_symbols = jpeg_symbols(:);
jpeg_symbols_values = unique(jpeg_symbols);
jpeg_symbols_count = hist(jpeg_symbols, jpeg_symbols_values);
jpeg_symbols_prob = jpeg_symbols_count./sum(jpeg_symbols_count);
jpeg_entropy = -sum(jpeg_symbols_prob.*log2(jpeg_symbols_prob));

fprintf('Entropy JPEG: %.4f bit/symbol\n',jpeg_entropy);
```

PSNR JPEG: 35.8455
Entropy JPEG: 0.9295 bit/symbol

4a) Reconstruct the image using only the DC component of the DCT - estimate the PSNR and display the reconstructed image

```
% store the quantized symbols to compute entropy later on
dc_symbols = zeros(K,K,num_block);
block_idx = 1;

dc_only_mask = false(K,K);
dc_only_mask(1,1) = true;

Y_dc = zeros(size(Y));
for r = 1:size(Y,1)/8
    for c = 1:size(Y,2)/8

        block_r_idxs = (r-1)*N+1:r*N;
        block_c_idxs = (c-1)*N+1:c*N;

        % 4a) consider block of size 8x8
```

```
            block = Y(block_r_idxs,block_c_idxs);

            % 4b) compute the DCT
            block_dct = dct2(block-127);

            % 4c) keep only DC
            block_dc_coeff = round(block_dct./Qmatrix);
            block_dc_coeff(~dc_only_mask) = 0;
            block_dc_q = block_dc_coeff .* Qmatrix;

            dc_symbols(:,:,block_idx) = block_dc_coeff;
            block_idx = block_idx + 1;

            % 4d) reconstruct the image from quantized coefficients

            block_idct = idct2(block_dc_q);
            Y_dc(block_r_idxs,block_c_idxs) = block_idct + 127;
        end
    end

    % display image and compute PSNR

    % figure()
    % subplot(1,2,1);
    % imshow(uint8(Y));
    % title('Original');
    %
    % subplot(1,2,2);
    % imshow(uint8(Y_dc));
    % title('DC');

    mse_dc = mean((Y(:)-Y_dc(:)).^2);
    PSNR_dc = pow2db(255.^2/mse_dc);

    disp(['PSNR DC: ' num2str(PSNR_dc)])

    dc_symbols = dc_symbols(:);
    dc_symbols_values = unique(dc_symbols);
    dc_symbols_count = hist(dc_symbols, dc_symbols_values);
    dc_symbols_prob = dc_symbols_count./sum(dc_symbols_count);
    dc_entropy = -sum(dc_symbols_prob.*log2(dc_symbols_prob));

    fprintf('Entropy DC: %.4f bit/symbol\n',dc_entropy);

    % any comment?
```

```
PSNR DC: 23.6617
Entropy DC: 0.2102 bit/symbol
```

4b) Reconstruct the image using only one AC component of the DCT

Do not quantize, just for the sake of reconstruction

```
Y_ac = zeros(size(Y));

% fix one component
k1 = 2;
k2 = 3;

for r = 1:size(Y,1)/N
    for c = 1:size(Y,2)/N
```

4a) consider block of size 8x8

```
        block = Y((r-1)*N+1 : r*N, (c-1)*N+1 : c*N);
```

4b) compute the DCT

```
        block_dct = dct2(block);
```

4c) keep only coeff (k1, k2)

```
        coeff = block_dct(k1,k2);
        block_dct = zeros(size(block_dct));
        block_dct(k1,k2) = coeff;
```

4d) reconstruct the image from quantized coefficients

```
        Y_ac((r-1)*N+1 : r*N, (c-1)*N+1 : c*N) = idct2(block_dct);

    end
end

% figure()
% subplot(1,2,1);
% imshow(uint8(Y));
% title('Original');
%
% subplot(1,2,2);
% imshow(Y_ac, []);
% title('AC (2,3)');
```

5) Consider blocks of dimension 8x8 and estimate the correlation matrix

```
% compute image blocks
imblocks = im2col(Y,[N,N],'distinct');

% compute and remove the mean block
block_mean = mean(imblocks,1);
imblocks_zm = imblocks - block_mean;

r_blocks = zeros(N^2,N^2,num_block);
for block_idx = 1:num_block
    block = imblocks_zm(:,block_idx);
    r_blocks(:,:,block_idx) = block * block';
end

R = mean(r_blocks,3);
```

6) Perform KLT coding - estimate the PRNR and display the reconstructed image

```
% Compute transform matrix from correlation
[V,D] = eig(R);
T_klt = V; %just for convenience

Q = 23;
Qvector = Q*ones(N*N,1);

klt_symbols = zeros(K^2,num_block);
block_idx = 1;

y_klt = zeros(size(Y));
% For each block
for r = 1:size(Y,1)/8
    for c = 1:size(Y,2)/8
        % 6a) consider block of size 8x8
        block_r_idxs = (r-1)*N+1:r*N;
        block_c_idxs = (c-1)*N+1:c*N;

        block = Y(block_r_idxs,block_c_idxs);

        % 6b) compute the KLT
        this_block_mean = mean(block(:));
        block_klt = T_klt'*(block(:)- this_block_mean);

        % 6c) threshold quantization
        block_klt_coeff = round(block_klt./Qvector);
        block_klt_q = block_klt_coeff.*Qvector;

        klt_symbols(:,block_idx) = block_klt_coeff;
```

```
        block_idx = block_idx + 1;

        % 6d) reconstruct the image from quantized coefficients (use
 reshape() if needed)
        block_iklt = T_klt * block_klt_q + this_block_mean;
        y_klt(block_r_idxs,block_c_idxs) = reshape(block_iklt,8,8);

    end
end

% display image and compute PSNR  and entropy
%
% figure()
% subplot(1,2,1);
% imshow(uint8(Y));
% title('Original');
%
% subplot(1,2,2);
% imshow(uint8(y_klt));
% title('KLT');

mse_klt = mean((Y(:)-y_klt(:)).^2);
PSNR_klt = pow2db(255.^2/mse_klt);

disp(['PSNR KLT: ' num2str(PSNR_klt)])

klt_symbols = klt_symbols(:);
klt_symbols_values = unique(klt_symbols);
klt_symbols_count = hist(klt_symbols, klt_symbols_values);
klt_symbols_prob = klt_symbols_count./sum(klt_symbols_count);
klt_entropy = -sum(klt_symbols_prob.*log2(klt_symbols_prob));

fprintf('Entropy KLT: %.4f bit/symbol\n',klt_entropy);
```

```
PSNR KLT: 36.0751
Entropy KLT: 0.6559 bit/symbol
```

Published with MATLAB® R2019b