Laplacian Pyramid for Image Encoding: Part 3

Finally, you are going to connect what you have implemented in the previous two labs with the code of this project. We will use the example image in the first step of the project and we will use the Laplacian encoding to compress it. We have set a predefined number of bins for each level of the pyramid, but you can select different values and see the effect in the image after the encoding.

For a comparison, we will also leverage code from the second lab, and approximate the image using Fourier transform. We will attempt to keep approximately the same compression rate for this technique, such that we need the same number of bits for each image (notice though that in both cases there are still options to optimize for the compression, although this is beyond the scope of this project).

Proceed with filling the last details of the code and visualizing the results using the script **Laplacian_encoding.m**. You can see the effect that different options have on the SNR of the image, as well as on quantitative aspects of the output (smoothing, high frequency details, noise, etc).

Your Script

Save Reset MATLAB Documentation (https://www.mathworks.com/help/)

```
% load the image we will experiment with
I = imresize(double(rgb2gray(imread('lena.png'))),[256 256]);

% build the Laplacian pyramid of this image with 6 levels
depth = 6;
L = laplacianpyr(I,depth);

% compute the quantization of the Laplacian pyramid
bins = [16,32,64,128,128,256]; % number of bins for each pyramid level
LC = encoding(L,bins);

% compute the entropy for the given quantization of the pyramid
ent = pyramident(LC);

% Use the collapse command of the Lab 3 to recover the image
Ic = collapse(LC);

% compute the snr for the recovered image
snr_c = compute_snr(I,Ic);

% use the code from Lab 2 to compute an approximation image with
% the same level of compression approximately
[rows,cols] = size(I);
n_0 = rows*cols;
M = n_0/8;
Id = decompress(compress(I,sqrt(M)));
snr_d = compute_snr(I,Id);

% plot the resulting images
subplot(1,3,1);
imshow(I,[]); title('Original image');
subplot(1,3,2); imshow(Ic,[]);
title('Laplacian Encoding'); xlabel(['SNR = ' num2str(snr_c)]);
subplot(1,3,3); imshow(Id,[]);
title('Fourier Approximation'); xlabel(['SNR = ' num2str(snr_d)]);

function ent = pyramident(LC)

    % Input:
    % LC: the quantized version of the images stored in the Laplacian pyramid
    % Output:
    % br: the bitrate for the image given the quantization

    % Please follow the instructions to fill in the missing commands.

    ent = 0;                % initialization of entropy
    [r, c] = size(LC{1});
```

```matlab
    pixI = r*c;             % number of pixels in the original image

    for i = 1:numel(LC)

        % 1) Compute the number of pixels at this level of the pyramid
        [r, c] = size(LC{i});
        pix_i = r*c;

        % 2) Compute the entropy at this level of the pyramid
        % (MATLAB command: entropy)
        ent_i = entropy(LC{i});

        % 3) Each level contributes to the entropy of the pyramid by a
        % factor that is equal to the sample density at this level, times
        % the entropy at this level. The sample density is computed as
        % (number of pixels at this level)/(number of pixels of original image).
        % Add this to the current sum of the entropy 'ent'
        ent = ent + (pix_i/pixI)*ent_i;

    end

end

function LC = encoding(L, bins)

    % Input:
    % L: the Laplacian pyramid of the input image
    % bins: [an array of ints, position i representing the]
    % number of bins used for discretization of each pyramid level
    % Output:
    % LC: the quantized version of the image stored in the Laplacian pyramid

    % Please follow the instructions to fill in the missing commands.

    depth = numel(bins);
    LC = cell(1,depth);

    for i = 1:depth

        % 1) Compute the edges of the bins we will use for discretization
        % (MATLAB command: linspace)
        % For level i, the linspace command will give you a row vector
        % with bins(i) linearly spaced points between [X1,X2].
        % Remember that the range [X1,X2] depends on the level of the
        % pyramid. The difference images (levels 1 to depth-1) are in
        % the range of [-128,128], while the blurred image is in the
        % range of [0,256]
        if i == depth % blurred image in range [0, 256]
            edges = linspace(0,256,bins(i));
        else % difference image in range [-128,128]
            edges = linspace(-128,128,bins(i));
        end

        % 2) Compute the centers that correspond to the above edges
        % The 1st center -> (1st edge + 2nd edge)/2
        % The 2nd center -> (2nd edge + 3rd edge)/2 and so on
        half_lng_int = (edges(2)-edges(1))/2;
        centers = edges + half_lng_int .* ones(1,bins(i));
        centers(end) = [];

        % 3) Discretize the values of the image at this level of the
        % pyramid according to edges (MATLAB command: discretize)
        % Hint: use 'centers' as the third argument of the discretize
        % command to get the value of each pixel instead of the bin index.

        LC{i} = discretize(L{i}, edges, centers);


```

```
    end

end

function I = collapse(L)

    % Input:
    % L: the Laplacian pyramid of an image
    % Output:
    % I: Recovered image from the Laplacian pyramid

    % Please follow the instructions to fill in the missing commands.

    depth = numel(L);

    % 1) Recover the image that is encoded in the Laplacian pyramid
    for i = depth:-1:1
        if i == depth
            % Initialization of I with the smallest scale of the pyramid
            I = L{i};
        else
            % The updated image I is the sum of the current level of the
            % pyramid, plus the expanded version of the current image I.
            I = L{i} + expand(I);
        end
    end

end

function L = laplacianpyr(I,depth)

    % Input:
    % I: the input image
    % depth: number of levels of the Laplacian pyramid
    % Output:
    % L: a cell containing all the levels of the Laplacian pyramid

    % Please follow the instructions to fill in the missing commands.

    L = cell(1,depth);

    % 1) Create a Gaussian pyramid
    % Use the function you already created.
    G = gausspyr(I,depth);

    % 2) Create a pyramid, where each level is the corresponding level of
    % the Gaussian pyramid minus the expanded version of the next level of
    % the Gaussian pyramid.
    % Remember that the last level of the Laplacian pyramid is the same as
    % the last level of the Gaussian pyramid.
    for i = 1:depth
        if i < depth
            % same level of Gaussian pyramid minus the expanded version of next level
            L{i} = G{i} - expand(G{i+1});
        else
            % same level of Gaussian pyramid
            L{i} = G{i};
        end
    end

end

function G = gausspyr(I,depth)

    % Input:
    % I: the input image
    % depth: number of levels of the Gaussian pyramid
    % Output:
```

```
    % G: a cell containing all the levels of the Gaussian pyramid

    % Please follow the instructions to fill in the missing commands.

    G = cell(1,depth);

    % 1) Create a pyramid, where the first level is the original image
    % and every subsequent level is the reduced version of the previous level
    for i = 1:depth
        if i == 1
            G{i} = I; % original image
        else
            G{i} = reduce(G{i-1}); % reduced version of the previous level
        end
    end

end

function g = expand(I)

    % Input:
    % I: the input image
    % Output:
    % g: the image after the expand operation

    % Please follow the instructions to fill in the missing commands.

    % 1) Create the expanded image.
    % The new image should be twice the size of the original image.
    % So, for an n x n image you will create an empty 2n x 2n image
    % Fill every second row and column with the rows and columns of the original image
    % i.e., 1st row of I -> 1st row of expanded image
    %       2nd row of I -> 3rd row of expanded image
    %       3rd row of I -> 5th row of expanded image, and so on
    I = im2double(I);
    [m,n,clr] = size(I);
    I_exp = zeros(2*m, 2*n, clr);
    % note: 1:2 gives odd indices
    I_exp(1:2:2*m, 1:2:2*n,:) = I(1:m, 1:n,:);

    % 2) Create a Gaussian kernel of size 5x5 and
    % standard deviation equal to 1 (MATLAB command fspecial)
    Gauss = fspecial('gaussian',5,1);

    % 3) Convolve the input image with the filter kernel (MATLAB command imfilter)
    % Tip: Use the default settings of imfilter
    % Remember to multiply the output of the filtering with a factor of 4
    g = 4*imfilter(I_exp,Gauss);

end


function g = reduce(I)

    % Input:
    % I: the input image
    % Output:
    % g: the image after Gaussian blurring and subsampling

    % Please follow the instructions to fill in the missing commands.

    % 1) Create a Gaussian kernel of size 5x5 and
    % standard deviation equal to 1 (MATLAB command fspecial)
    Gauss = fspecial('gaussian',5,1);

    % 2) Convolve the input image with the filter kernel (MATLAB command imfilter)
    % Tip: Use the default settings of imfilter
    I = im2double(I);
```

```
    im_filtered = imfilter(I,Gauss);

    % 3) Subsample the image by a factor of 2
    % i.e., keep only 1st, 3rd, 5th, .. rows and columns
    g = im_filtered(1:2:end, 1:2:end,:);

end

function [Fcomp] = compress(I,M_root)

    % Input:
    % I: the input image
    % M_root: square root of the number of coefficients we will keep
    % Output:
    % Fcomp: the compressed version of the image

    % Please follow the instructions in the comments to fill in the missing commands.

    % 1) Perform the FFT transform on the image (MATLAB command fft2).
    Fcomp = fft2(I);

    % 2) Shift zero-frequency component to center of spectrum (MATLAB command fftshift).
    Fcomp = fftshift(Fcomp);

    % We create a mask that is the same size as the image. The mask is zero everywhere,
    % except for a square with sides of length M_root centered at the center of the image.
    [rows,cols] = size(I);
    idx_rows = abs((1:rows) - ceil(rows/2)) < M_root/2 ;
    idx_cols = abs((1:cols)- ceil(cols/2)) < M_root/2 ;
    M = (double(idx_rows')) * (double(idx_cols));

    % 3) Multiply in a pointwise manner the image with the mask.
    Fcomp = Fcomp .* M;

end

function [Id] = decompress(Fcomp)

    % Input:
    % F: the compressed version of the image
    % Output:
    % Id: the approximated image

    % Please follow the instructions in the comments to fill in the missing commands.

    % 1) Apply the inverse FFT shift (MATLAB command ifftshift)
    Id = ifftshift(Fcomp);

    % 2) Compute the inverse FFT (MATLAB command ifft2)
    Id = ifft2(Id);

    % 3) Keep the real part of the previous output
    Id = real(Id);

end

function snr = compute_snr(I, Id)

    % Input:
    % I: the original image
    % Id: the approximated (noisy) image
    % Output:
    % snr: signal-to-noise ratio

    % Please follow the instructions in the comments to fill in the missing commands.

    % 1) Compute the noise image (original image minus the approximation)
    noise_image = I - Id;
```

```

    % 2) Compute the Frobenius norm of the noise image
    noise_fnorm = norm(noise_image, 'fro');

    % 3) Compute the Frobenius norm of the original image
    orig_fnorm = norm(I,'fro');

    % 4) Compute SNR
    snr = -20*log10(noise_fnorm/orig_fnorm);

end

```

▶ Run Script

Previous Assessment: All Tests Passed

Submit

Is the estimated output for the image encoded with the Laplacian pyramid correct?

Is the estimated output for the SNR of the Laplacian encoding correct?

Is the estimated output for the image approximated with the 2D Fourier transform correct?

Is the estimated output for the SNR of the Fourier approximation correct?

Output