Laplacian Pyramid for Image Encoding: Part 2

For the second part, you will implement the function **pyramident**, that will compute the entropy for an encoded Laplacian pyramid we provide. Without going into too many details, the entropy of an image is a statistical measure that quantifies the least number of bits required to represent each pixel in the image. If an image has low entropy, then we need less bits to represent it (as in the example of the previous step). Please follow the instructions in the script below to implement the function **pyramident**.

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);

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});
    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 L = laplacianpyr(I,depth)

    % Add your code from the previous step
    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)

    % Add your code from the previous step
    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 = reduce(I)

    % Add your code from the previous step
    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 g = expand(I)

    % Add your code from the previous step
    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
```

▶ Run Script