Laplacian Pyramid for Image Encoding: Part 1

In this project you will use the Laplacian pyramid algorithm for image encoding/compression.

The motivation behind this application, lies in the fact that the Laplacian pyramid stores image differences at high resolutions which are quite predictable, in the sense that the variety of the values is pretty small, with most pixels being concentrated around zero. To capitalize on that, it was proposed by Burt and Adelson to encode these values in a more compact representation, by quantizing the values, requiring essentially less bits of information to represent each one of those. This is relevant to the Image Approximation of the second lab, where we were able to compress the image by cutting the high frequencies of the image and storing a low frequency representation. Here the motivation is the same, since we again attempt to compress the image, however, the approach is different, where we do not necessarily cut down the high frequencies, so details of the image can still be visible. Take a look at the result below, where we attempt to compress the image with the two techniques, for approximately the same compression ratio.

Original image

Laplacian Encoding

SNR = 28.5122

Fourier Approximation

SNR = 23.5873

For the first part of this project, you will quantize the values of the images (the differences and the low resolution blurred version) stored in the Laplacian pyramid using uniform quantization. While for most images, we assume that the pixel values lie in the interval [0, 255], here we will assume only a small discrete set of these values are available, e.g. {0, 63, 127, 191, 255}, and each pixel will be assigned a value from this discrete set only. Since we limit ourselves in this small set, we lose some accuracy in our values, but we need much less space to represent the values of each pixel. In the above example, each pixel can take only one of the 5 values, instead of the 256 values that are originally available.

In the next script, follow the instruction to complete the function **encoding**, that takes as input a Laplacian pyramid and encodes it by uniformly quantizing the values of the pixels for the different images. You can find the all the relevant material for this lab by downloading the zip file Project1.zip (http://courses.edx.org/asset-v1:PennX+ROBO2x+2T2017+type@asset+block@Project1.zip).

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);

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};
```

```
            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

Previous Assessment: All Tests Passed