Homework 5 – Wavelet

Arya Araban

Abstract

In this Article we will be exploring the frequency domain yet again, this time with a new subject known as Wavelet Transformations.

After the main points are explained, we will try to create what's known as a Laplacian and Wavelet pyramid.

Report Info

Date: 1399/10/21

Keywords:
Pyramid Representation
Frequency domain
Laplacian Pyramid
Image Denoising
Haar filters

When considering wavelets, we consider that an image has different information stored on the different resolution levels. By doing some certain operations and the downsampling, we reach a new, lower level. When we put these levels next to each other, we'll get what's known as a multi-resolution pyramid.

A Guassian Pyramid is constructed by repeatedly calculating a weighted average of the neighboring pixels(the Guassian filter) of a source image and scaling the image down. It can be visualized by stacking progressively smaller versions of the image on top of one another. This process creates a pyramid shape with the base as the original image and the tip a single pixel representing the average value of the entire image.

A Laplacian Pyramid is rather similar to how a Guassian Pyramid is constructed, however, after applying the Guassian filter and downsample the image, we upsample it and find the difference between the

1- Introduction

Wavelet transformations are one of the important subjects in mathematics which has a lot of uses across different study fields. The main idea of using Wavelet transformations is to overcome the weaknesses that the standard Fourier Transform shows. A fourier transform will tell us what frequencies are present in the signal. A wavelet transform will tell us what frequencies are present and where (or at what scale) they are located.

2- Technical Description

Problem 6.1.1:

What's the maximum number we're able to to have in an approximation pyramid representation? How does this number compare to the original pixel number? What are the benefits of using this method?

Solution:

In order to answer this question, we'll give a brief introduction to Wavelet in general, which we'll use what we gain from this introduction for the other problems as well.

Problem 6.1.3:

compute the wavelet transform (with 3-level) using the Haar analysis filters.

Solution:

In this section we look at Pyramid decomposition using the Haar wavelet filter.

Each time we apply a filter, we get four different sections for an image.

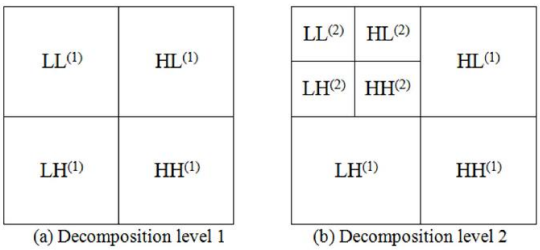
The LL section (LL stands for low-horizontal & low-vertical frequency section) which is the approximation section we use.

The HL section which gives the horizontal edges.

The LH section which gives the vertical edges.

The HH section which gives the diagonal edges.

We can apply the filter to the LL again, to get the next level for the approximation:



(a) Decomposition level 1 (b) Decomposition level 2

If we compare this to the Laplacian Pyramid that we found in the previous section, We can see that this transformation holds a lower space complexity compared to the Laplacian Pyramid (due to using less pixels), however the Time complexity may be higher due that applying the haar filter is computationally more expensive than applying a box filter.

original image and the result (that's being downsampled and upsampled) we continue to do this for different levels until the Laplacian Pyramid is formed.

*If we consider our Image to be of size N*N And N = 2^J, then the maximum number of levels we can have will be equal to $\log_2(N)$ = J +1 due to Shannon's law.*

The sum of all the pixels in the different levels will be as follows:

Total Pixels = 1×1+2×2+4×4+8×8+⋯

⇨ *Total Pixels = (4^(J+1)-1)/4-1*

*Keeping in mind that the number of pixels in the original image equals 2^J*2^J = 2^(J+1) then we can notice that the total number of pixels is* ***more*** *than what we have in the original image.*

The reason we have use for these pyramids is that even though we are using more pixels than the original image, most of the matrices information are sparse, and they take up less space. So in the end we'll end up with an image that is compressed, and takes up less space.

Problem 6.1.2:

manually compute a 3-level approximation pyramid and corresponding prediction residual pyramid.

Solution:

In the previous section we've seen how we can create a Laplacian Pyramid. If we wanted to reconstruct the original image given the output of the final level, then we upsample this output by applying pixel replication and multiplying the image size times 2, in the end we finally sum the obtained image with the Laplacian we have found. We keep doing these steps until all levels have been iterated and a close approximation to the original image is found.

N and the modified detail coefficients of levels from 1 to N.

To explain what soft thresholding is we first need to understand Hard thresholding:
Hard thresholding can be described as the usual process of setting to zero the elements whose absolute values are lower than the threshold. The hard threshold signal is x if x > thr, and is 0 if

Soft thresholding is an extension of hard thresholding, first setting to zero the elements whose absolute values are lower than the threshold, and then shrinking the nonzero coefficients towards 0. The soft threshold signal is sign(x)(x-thr) if x > thr and is 0 if x ≤ thr.

We will be applying both soft and hard thresholding for up to three levels for the second step to see which one of the methods performs better.

3- Describing the Results

Problem 6.1.1 & 6.1.2:

Since both of these problems follow the same procedure, we'll bring them both in one section.

The original Grayscale Lena Image is as follows:

Problem 6.1.4:

Quantize all the wavelet coefficients (whole sub-bands) by a step size of $\gamma = 2$. Then reconstruct the image from the quantized wavelet coefficients.

Solution:

The exact same steps of the previous question are taken, except that after finding each of the coefficients, we apply the function given in the question and replace it with the original coefficient. Note that in order to reconstruct the original image, we need to store the values original of HL,LH,HH (after applying the function) for all levels somewhere, so we can make use of these to apply the single-level inverse wavelet transform using these values, alongside the image(LL part) that we already have a number of L times (where L is the total number of levels we have)

Problem 6.2.1:

Implement two of the various image denoising techniques presented that is based on wavelet and compare them on both noised image sequences and real-world sequences.

Solution:

The main procedure to denoise an image can be broken down into three steps:

Step 1 -> Decompose:

Choose a wavelet, choose a level N. Compute the wavelet decomposition of the signal at level N.

Step 2 -> Threshold detail coefficients:

For each level from 1 to N, select a threshold and apply soft(or hard) thresholding to the detail coefficients.

Step 3 -> Reconstruct:

Compute wavelet reconstruction using the original approximation coefficients of level

Now lets consider that given this pyramid with 9 levels, we want to reconstruct to the original image. By doing the reconstruction procedure explained previously, This is the result that is obtained:

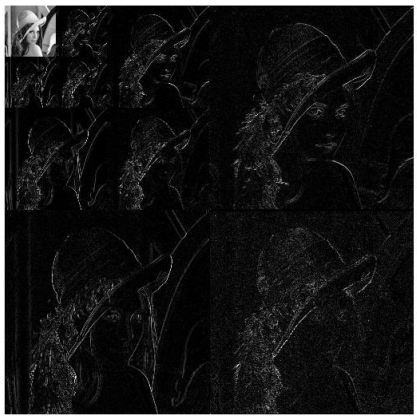
If we compare this image to the original image, the results are as follows:

```
MSE=0 --- PSNR=Inf
```

Which shows that both the reconstructed image, and the original image are exactly identical!

Problem 6.1.3:

If we follow the procedure to find a wavelet transformation, considering 3 levels, the result of the pyramid will be as follows:



We can clearly see that each of the three non-approximation regions give the different types of edges.

From the previous section we noted that the maximum number of levels we can Have is J where J= $log_2(N)$. *in the Lena image N = 512, so we will have that J = 9.*

We'll look at the case where we consider we want to have a Laplacian pyramid (which we find with the explained procedure) which has 3 levels:

*Note that on the bottom side of each image, is the laplacian of the image, which is rather hard to see.

If we consider 9 levels we get the following results:

And the image obtained from reconstructing this image is:

And if we compare this image to the original image, the PSNR and MSE values will be as follows :

```
MSE=1.3768 --- PSNR=46.742
```

Which shows that by applying quantization with the step size of 2 for each of the four regions during the pyramid construction, we get a slight change in the resulting image.

Problem 6.2.1:

In order to solve this problem, we construct the wavelet pyramid how we normally would, with the difference being that for each of the regions, excluding the LL region, we apply either Soft Thresholding or Hard Thresholding after finding their values. Finally we reconstruct with the new coefficient values,and the result will be an image which is denoised

Note that the value which we take as the threshold value depends on the region. We find the median value of each region and divide it by a value which we pick.
Picking the value of 0.6745 seems to give good results.

If we reconstruct the original image by performing the procedure mentioned in the technical details of 6.1.4 (by applying the idwt2 method of Matlab a number of L times) the resulted image will be as follows:

And if we compare the reconstruced image to the original image, yet again..:

```
MSE=0 --- PSNR=Inf
```

Which shows that the result, like before, is the exact same as the original image.

Problem 6.1.4:

The way we solve this problem is similar to the previous problem, but with a slight difference that for each level, after we apply dwt2 and getting the four regions, we apply the quantization method for each of the given regions.
The resulting pyramid will be as follows:

~~~~ Hard Thresholding ~~~~

By considering the Number of decompositions as N=4, and applying hard thresholding during the construction, the reconstructed image will be as follows:

By comparing this image and the original image:

```
MSE=1655.9449 --- PSNR=15.9403
```

Which shows that a slight denoising has happened, but it is so small that can't easily be noticed.

The results we obtained show just how much more effective soft thresholding is for denoising compared to hard thresholding.

We give the Lena image which we have applied guassian noise to (which has the mean of 0 and variance of 0.035)

The comparisons of this image and the original image is as follows:

```
MSE=2064.6466 --- PSNR=14.9823
```

Now we'll try the two different types of thresholding..:

~~~~ Soft Thresholding ~~~~

By considering the Number of decompositions as N=4, and applying soft thresholding during the construction, the reconstructed image will be as follows:

And by comparing this image and the original image:

```
MSE=429.2997 --- PSNR=21.8032
```

Which shows a drastic improvement over our noisy image. The obtained image is now much closer to the original image than it previously was!

```
function output = pixel_replication(img)
[X,Y] = size(img);
%make sure output is 2* the size of the input
output = zeros(2*X,2*Y);
for x = 1:X
    for y = 1:Y
        %first we find the top-left, and then find others in respect to it
        j = 2*(x-1) + 1;
        i = 2*(y-1) + 1;
        output(j,i) = img(x,y); %// Top-left
        output(j+1,i) = img(x,y); %// Bottom-left
        output(j,i+1) = img(x,y); %// Top-right
        output(j+1,i+1) = img(x,y); %// Bottom-right
    end
end
end

function output=guassian_filter(image,filter)
    [R,C] = size(image);
    output = zeros(R/2,C/2,'double');
    for i=1:2:R
        for j=1:2:C
            part = double(image(i:i+1,j:j+1));
            mult = part.*filter;
            out = sum(mult,'all');
            output(ceil(i/2),ceil(j/2)) = out;
        end
    end
end

function output = normalize(img)
    output = mat2gray(img);
    Max = max(max(output));
    Min = min(min(output));
    output = (255/(Max-Min))*output;
end
```

And finally, the main code is as follows:

```
img = imread('C:\Users\Arya\Desktop\computer vision\HW\Images\6\Lena.bmp');
img = rgb2gray(img);
L = 3;
%CONSTRUCT PYRAMID
[pyramid,laplacians] = laplacian_pyramid(img,L);

%RECONSTRUCTION
[n,~] = size(laplacians);
last_img = laplacians{n};
for i=1:L
    n = n-1;
    last_img = pixel_replication(last_img);
    last_img = last_img + laplacians{n};
end
last_img = uint8(last_img);

disp("MSE=" + immse(img,last_img) + " --- " + "PSNR=" + psnr(img,last_img));
imshow(last_img);


figure();imshow(last_img);

imwrite(last_img, "1_reconstructed.jpg");
```

Problem 6.1.3:

The code used to for finding the wavelet pyramid is as follows:

```
img = rgb2gray(imread('C:\Users\Arya\Desktop\computer vision\HW\Images\6\Lena.bmp'));

nLevel = 3;  % Number of decompositions
map = gray; %grayscale colormap
nColors = size(map, 1);  % Number of colors in colormap
cA = cell(1, nLevel);    % Approximation coefficients
cH = cell(1, nLevel);    % Horizontal detail coefficients
cV = cell(1, nLevel);    % Vertical detail coefficients
cD = cell(1, nLevel);    % Diagonal detail coefficients

startImage = img;
%CONSTRUCT
for iLevel = 1:nLevel,
  [cA{iLevel}, cH{iLevel}, cV{iLevel}, cD{iLevel}] = dwt2(startImage, 'haar');
  startImage = cA{iLevel};
end
%CREATE TILED IMAGE
tiledImage = wcodemat(cA{nLevel}, nColors);
for iLevel = nLevel:-1:1,
  tiledImage = [tiledImage                    wcodemat(cH{iLevel}, nColors); ...
                wcodemat(cV{iLevel}, nColors) wcodemat(cD{iLevel}, nColors)];
end

figure;imshow(tiledImage, map);
%RECONSTRUCT ORIGINAL
fullRecon = cA{nLevel};
for iLevel = nLevel:-1:1,
  fullRecon = idwt2(fullRecon, cH{iLevel}, cV{iLevel}, cD{iLevel}, 'haar');
end
fullRecon = uint8(fullRecon);

figure;imshow(fullRecon);
disp("MSE=" + immse(img,fullRecon) + " --- " + "PSNR=" + psnr(img,fullRecon));
```

4- Appendix(code)

Problem 6.1.1 & 6.1.2:

The code for these two problems are very similar. So I've brought them in the same section.

The main code used is constructed by several functions:

The function used to construct the Laplacian Pyramid is as follows:

```
function [final,laplacians_last_img] = laplacian_pyramid(img,levels)

    avg_filter = [1 1;1 1]/4;
    [R,C]  = size(img);
    final = zeros(2*R,2*C,'uint8');
    old_img = double(img);

    N = levels;
    laplacians_last_img = cell(N,1);
    for i=1:levels

        r1 = R-(0.5)^(i-1)*R+1;
        r2 = R;

        c1 = ((2^(i-1)-1)/(2^(i-2)))*C+1;
        c2 = (2^(i)-1)/(2^(i-1))*C;

        final(r1:r2,c1:c2) = old_img;

        new_img = guassian_filter(old_img,avg_filter);


        laplace = old_img - pixel_replication(new_img);

        laplacians_last_img{i} = laplace;

        laplace = normalize(laplace);
        laplace = uint8(laplace);

        r1 =R+1;
        r2 = R+(0.5)^(i-1)*R;

        final(r1:r2,c1:c2)=laplace ;
        old_img = new_img;
    end

     i = levels+1;
     r1 = R-(0.5)^(i-1)*R+1;
     r2 = R;
     c1 = ((2^(i-1)-1)/(2^(i-2)))*C+1;
     c2 = (2^(i)-1)/(2^(i-1))*C;
     final(r1:r2,c1:c2) = old_img;
     r1 =R+1;
     r2 = R+(0.5)^(i-1)*R;
     final(r1:r2,c1:c2) = old_img;

     laplacians_last_img{levels+1}=old_img;
end
```

The three other functions which are used for applying pixel replication, applying the Guassian filter, and normalizing the image are as follows:

Problem 6.2.1:

The code used to apply denoising to an image based on a given threshold:

```
img = rgb2gray(imread('C:\Users\Arya\Desktop\computer vision\HW\Images\6\Lena.bmp'))

nLevel = 3;  % Number of decompositions
th = 's'; %th = 'h'; %HARD OR SOFT THRESHOLD
thresh_val = 0.6745;

map = gray; %grayscale colormap
nColors = size(map, 1);  % Number of colors in colormap
cA = cell(1, nLevel);    % Approximation coefficients
cH = cell(1, nLevel);    % Horizontal detail coefficients
cV = cell(1, nLevel);    % Vertical detail coefficients
cD = cell(1, nLevel);    % Diagonal detail coefficients

noisy_image = imnoise(img,'gaussian',0,0.035);
startImage = noisy_image;
figure;imshow(noisy_image); imwrite(noisy_image, "5_noisyLena.jpg");
%CONSTRUCT
for iLevel = 1:nLevel,
  [cA{iLevel}, cH{iLevel}, cV{iLevel}, cD{iLevel}] = dwt2(startImage, 'haar');

  %APPLYING THRESHOLD TO ALL,EXCEPT THE LL
  cH{iLevel} = wthresh(cH{iLevel},th,(median(abs(cH{iLevel}(:)))/thresh_val));
  cV{iLevel} = wthresh(cV{iLevel},th,(median(abs(cV{iLevel}(:)))/thresh_val));
  cD{iLevel} = wthresh(cD{iLevel},th,(median(abs(cD{iLevel}(:)))/thresh_val));

  startImage = cA{iLevel};
end
% RECONSTRUCT
fullRecon = cA{nLevel};
for iLevel = nLevel:-1:1,
  fullRecon = idwt2(fullRecon, cH{iLevel}, cV{iLevel}, cD{iLevel}, 'haar');
end
fullRecon = uint8(fullRecon);

figure;imshow(fullRecon); imwrite(fullRecon, "5_Lena_HT_Reconstucted.jpg");

disp("MSE=" + immse(img,fullRecon) + " --- " + "PSNR=" + psnr(img,fullRecon));
```

Problem 6.1.4:

In order to quantize an image with the step size of 2, the following function has been used:

```
function out = quantFunc (img) %quantize an image with by the step size(gamma) of two
    [X,Y] = size(img);
    out = double(zeros(X,Y));
    for i=1:X
        for j=1:Y
            out(i,j) = 2* sign(img(i,j))*floor( abs(img(i,j))/2);
        end
    end
end
```

And the main code that is used to apply this quantization to each of the four regions during the construction phase of a wavelet pyramid:

```
img = rgb2gray(imread('C:\Users\Arya\Desktop\computer vision\HW\Images\6\Lena.bmp'));
nLevel = 3;  % Number of decompositions
map = gray; %grayscale colormap
nColors = size(map, 1);  % Number of colors in colormap
cA = cell(1, nLevel);    % Approximation coefficients
cH = cell(1, nLevel);    % Horizontal detail coefficients
cV = cell(1, nLevel);    % Vertical detail coefficients
cD = cell(1, nLevel);    % Diagonal detail coefficients

startImage = img;
%Pyramid Construction
for iLevel = 1:nLevel,
  [cA{iLevel}, cH{iLevel}, cV{iLevel}, cD{iLevel}] = dwt2(startImage, 'haar');
  %  applying the quantFunc for each of the regions
  cA{iLevel} = quantFunc(cA{iLevel});
  cH{iLevel} = quantFunc(cH{iLevel});
  cV{iLevel} = quantFunc(cV{iLevel});
  cD{iLevel} = quantFunc(cD{iLevel});
  startImage = cA{iLevel};
end

tiledImage = wcodemat(cA{nLevel}, nColors);
for iLevel = nLevel:-1:1,
  tiledImage = [tiledImage                    wcodemat(cH{iLevel}, nColors); ...
                wcodemat(cV{iLevel}, nColors) wcodemat(cD{iLevel}, nColors)];
end

figure;imshow(tiledImage, map);

%Original Image Reconstruction
fullRecon = cA{nLevel};
for iLevel = nLevel:-1:1,
  fullRecon = idwt2(fullRecon, cH{iLevel}, cV{iLevel}, cD{iLevel}, 'haar');
end
fullRecon = uint8(fullRecon);

imwrite(fullRecon, "4_reconstructed.jpg");
figure;imshow(fullRecon);
disp("MSE=" + immse(img,fullRecon) + " --- " + "PSNR=" + psnr(img,fullRecon));
```