Image Processing - 67829

Exercise 5: Discrete Wavelet Transform

Due date: 4/2/2015

1 Overview & Background

The purpose of this exercise is to get familiar with the *2-dimensional Discrete Wavelet Transform (DWT)* by performing the transform and its inverse. Moreover, you'll apply simple manipulations over the transformed coefficients. You may find it helpful to review the slides of Tirgul 10 in which wavelet transform was discussed.

2 Discrete Wavelet Transform - 60 points

In this section you should implement the decomposition into wavelet representation and reconstructing the image.

2.1 Image Decomposition - 30 points

Write a function that converts a 2-dimensional discrete signal (image) to its wavelet transform coefficients. The function should have the following interface:

```
waveletDecomp = DWT(image,lowFilt,highFilt,levels)
```

where:

`image` is a input image.

`lowFilt` is a row vector representing the scaling (or approximation) function which acts as a low pass filter.

`highFilt` is a row vector representing the wavelet function which acts as a high pass filter.

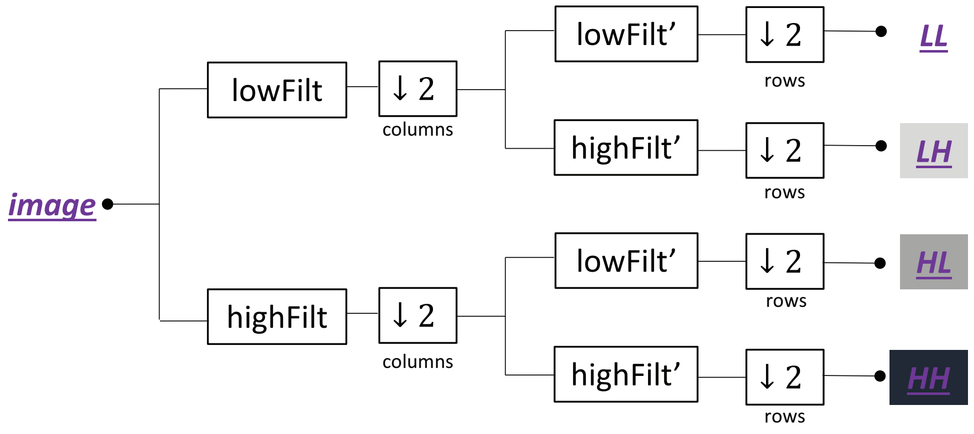


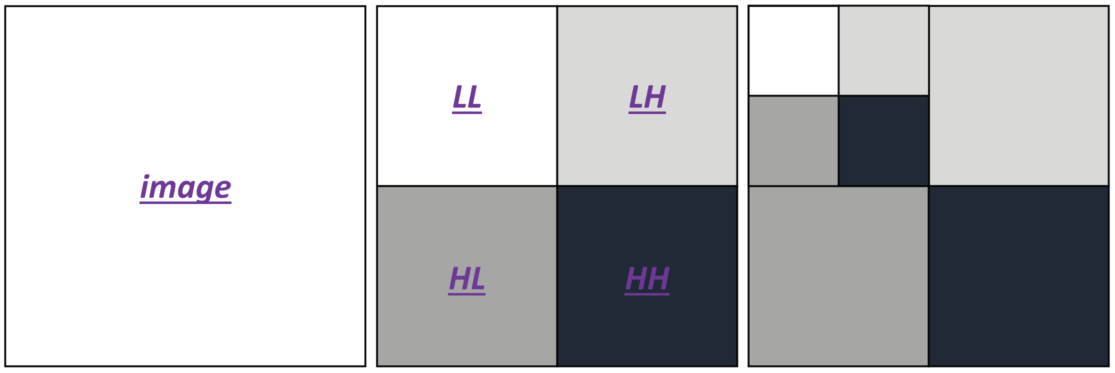
Figure 1: Wavelet transform. The figure depicts the entire process for decomposition. We start with the input image and apply a convolution with low- and high-pass row filters. Each of the images is downsampled by columns. The two images are convolved again with low- and high-pass column filters producing 4 subimages. Theses images are then downsampled by rows. ↓ denotes the down-sample procedure.

`levels` is the number of decomposition levels.
`waveletDecomp` is the wavelet transform coefficients.
The function should not display the output `waveletDecomp` image.

At each level of the wavelet transform we decompose the approximation coefficients in level j into 4 sets of coefficients to be the $j+1$ level. For the first level, the approximation coefficients are the `image` intensities. You should convolve the rows of the approximation coefficients with `lowFilt` and `highFilt` to obtain two images. Each of them should be downsampled by columns to reduce their horizontal resolutions by a factor of 2. Both subimages are then filtered column-wise and downsampled to yield four quarter-size output subimages - LL, LH, HL, HH. L stands for 'low' filter and H for 'high' filter. LL is the approximation image since it contains the low frequencies of the `image`. The remaining 3 subimages contain the information of the horizontal, vertical and diagonal details and denoted as LH, HL and HH respectively. A summary of this procedure is depicted in Figure 1. In the next level, the approximation image LL is decomposed into 4 subimages, this should be done iteratively depending on the number of `levels` needed.

Comments:

- In Figure 2b, you can see how the 4 subimages are arranged inside `waveletDecomp` image. Figure 2c



(a) input image (b) 1 level decomposition (c) 2 level decomposition

Figure 2: The decomposition procedure for 1 and 2 levels. LL image is being decomposed with the same process as shown in Figure 1.

depicts 2 levels decomposition.

- In order to properly **display** `waveletDecomp`, which contains negative values, you should use a linear transformation for the values to be in the range $[0, 1]$.
- you may assume that for each level of decomposition the spatial dimensions of the approximation coefficients (LL subimage) can be divided by 2 in each direction with no remainder.

2.2 Image Reconstruction - 30 points

Write a function that converts the wavelet coefficients into the original image. The function should have the following interface:

```
image = IDWT(waveletDecomp,lowFilt,highFilt,levels)
```

where:

`waveletDecomp` is the output image from Section 2.1.

`lowFilt` is the the scaling (or approximation) function.

`highFilt` is the wavelet function.

`levels` is the number of the decompositions levels.

`image` is the output image after reconstruction.

The function should not display the output `image`.

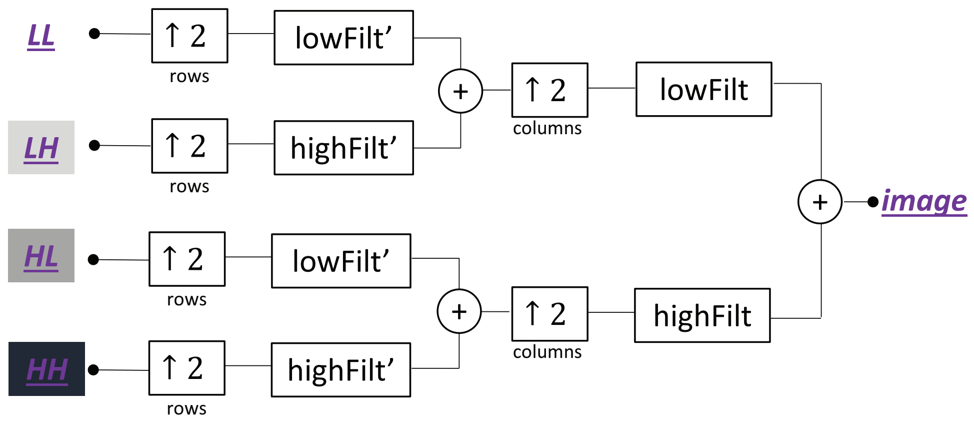


Figure 3: Image reconstruction from wavelet coefficients. The figure depicts the entire process for reconstruction. ↑ denotes the up-sample procedure.

At the beginning of this function you should multiply the `lowFilt` and `highFilt` by 2. Moreover, `highFilt` should be mirrored (see slide 25 in Tirgul 10). The reconstruction scheme is the reverse process and can be easily understood from Figure 3.

Comments:

- `lowFilt`, `highFilt` and `levels` are exactly the same as in Section 2.1.
- To avoid pixels misalignment please downsample by taking every odd pixel and upsample by inserting zeros in odd locations.
- Pay attention that in case of executing `IDWT(DWT(image,lowFilt,highFilt,levels),lowFilt,highFilt,levels)` you should obtain the original image up to a negligible floating error.

General comments:

- In this exercise we would like you to use Haar wavelet. `lowFilt` $= \frac{1}{2}[1,1]$ and `highFilt` $= \frac{1}{2}[1,-1]$.
- `image` will be a graysacle image in doubles representation.

3 Simple Applications for Wavelet Transform - 40 points

3.1 Compression - 20 points

In this section we use the wavelet representation from Section (2.1) in order to compress an image. All the values in `waveletDecomp` image (except for the LL image) can be compressed easily because the information is statistically concentrated in a narrow range of values. We call these values the details coefficients. In this section you will implement the compression procedure using **optimal quantization** function from Ex-1. You should implement the following interface:

```
[compressedImage,waveletDecompCompressed] = waveletQuantization(image,lowFilt,highFilt,levels,nQuant)
```

where:
`image`, `lowFilt`, `highFilt`, and `levels` have the same meaning as in Section 2.1.
`nQuant` is the number of values for the quantization process.
`compressedImage` is the output image after compression.
`waveletDecompCompressed` is the wavelet coefficients after applying the quantization procedure over the details coefficients (as described below).
The function should also: (a) display the output `compressedImage`. (b) save the wavelet coefficients `waveletDecomp` before the quantization procedure (c) save the wavelet coefficients `waveletDecompCompressed`.

This function should first apply the DWT over the `image` to get the transform image `waveletDecomp`. Then, the function takes the details coefficients and send it to `quantizeImage` function that you've written in Ex-1. In this step, image LL remains the same and images LH,HL and HH of all levels should have `nQuant` intensities. Lastly, reconstruct the image using IDWT.

You should save the wavelet transform image `waveletDecomp` before and after the quantization procedure. You should use the command 'save' in matlab with the flag '-v6' which forces Matlab to save without compression. Please save the two `waveletDecomp` as: 'beforeCompress.mat' and 'afterCompress.mat'. Afterwards, you should compress them using zip (which uses the lempel-ziv algorithm). Pay attention that you should save the two types of `waveletDecomp` images in uint8 representation (and not as doubles) or else the size of those images will be bigger than the size of the input one which makes the procedure irrelevant.

- As noted earlier, the details coefficients are LH, HL and HH images from all the levels. i.e., in Figure 2b these coefficients are $\frac{3}{4}$ of the entire wavelet coefficients and in Figure 2c these coefficients are $\frac{15}{16}$ of the pixels.
- You may change `quantizeImage` function from Ex-1 in order to suite your needs for this exercise. For example: it can return only the look-up-table. Please do not show the error graph. The focus of this section is on the compression procedure and **not** on the `quantizeImage` function and its interface.

- The details coefficients are in the range $[-\frac{1}{2}, \frac{1}{2}]$. The `quantizeImage` function receives values in the range of $[0, 1]$, hence you should add the minimum coefficient value to get the values in the desired range. After applying the quantization - you should subtract the value that you have added in order for the values to return to their original range $[-\frac{1}{2}, \frac{1}{2}]$.
- You may set the number of optimal quantization iterations to 8.
- There is a difficulty to quantize the details coefficients into many values since the intensities are concentrated in a very narrow range of values. The problem is in the initial division guess where each segment should contains approximately the same number of pixels. Therefore, we won't ask you to quantize to more values than the initial guess lets.
- The quantized intensities are global for all the details coefficients.
- Pay attention that when saving, you should transform the wavelet coefficients to be in the range of $[0, 255]$ before casting them to uint8.
- For compressing, use Matlab's 'gzip' command. We would like to find both '*.mat' and '*.gz' files in your running directory
- Explain in your README:
  1. why do the 'beforeCompress.mat' and 'afterCompress.mat' have the same file size?
  2. Why do we get different file sizes when compressing the results with zip, although they are the same resolution?
  3. Save and compress the input image (in the same manner as mentioned above). Why do we get a different file size from the zip file containing 'afterCompress.mat'? This question is not part of the function, hence, do not save and compress the input image inside the `waveletQuantization`.

3.2 Deleting horizonal lines - 10 points

In this section we use the wavelet representation from Section (2.1) in order to eliminate horizontal lines in an image. You are supplied with an image called 'page.png' which have horizontal and vertical lines. **Try to find a way to delete the horizontal lines**. You should implement the following interface:

```
outImage = deleteHorizontal(image,lowFilt,highFilt,levels)
```

where:

`image`, `lowFilt`, `highFilt`, and `levels` have the same meaning as in Section 2.1.

`outImage` is the output image without the horizontal lines.

The function should also display the output image.

The process should be done by manipulating the values of the wavelet coefficients.

3.3 Denoising - 10 points

In this section we use the wavelet representation from Section (2.1) in order to improve noisy images. You should implement the following interface:

```
denoisImage = denoising(image,lowFilt,highFilt,levels)
```

where:
`image`, `lowFilt`, `highFilt`, and `levels` have the same meaning as in Section 2.1.
`denoisImage` is the output image, with noise reduction.
The function should also display the input noisy image next to the de-noised output image (use 'subplot' function).

When analyzing the details coefficients (LH, HL and HH images of all levels), noise is characterize by values lower than edges. Hence, reducing noise can be done by thresholding the details coefficients. That is, select a threshold and set to zero elements of whose absolute values are lower than the threshold. This procedure is not perfect, and cannot remove the noise completely. However, modifying the wavelet coefficients can improve the image appearance. You should find the threshold using "trial and error".

You should create a noisy image by taking an image and adding a Gaussian noise with standard deviation of 0.04 (you can use matlab's 'random' function). Then, apply the DWT and threshold the details coefficients. Lastly, apply the IDWT in order to reconstruct the image. Write a script called 'filterNoisyImage.m' which runs the `denoising` function over your example image. In the submission file please add the noisy image you created as 'myNoisyImage.png'.

4 Submission

Submit the file `ex5.zip` should contain:

1. your `README` file, which should include answers to the 3 questions in Section 3.1.
2. 5 functions from Sections 2 and 3.
3. the noisy image you've created in Section 3.3 ('myNoisyImage.m').
4. the script of reducing the noise from Section 3.3 ('filterNoisyImage.png').

Good luck and enjoy!