CS-111 Programing Assignment 2: Image Pyramids

Submission instructions:

Please **submit your code, output images and a PDF file (containing the output images) in a single zip file** to Canvas. You must also **submit the same PDF file** to Gradescope.
BOTH submissions are required for full points.
Your work is due by **11:59 p.m. on Wednesday, the 24th of April**.

Introduction:

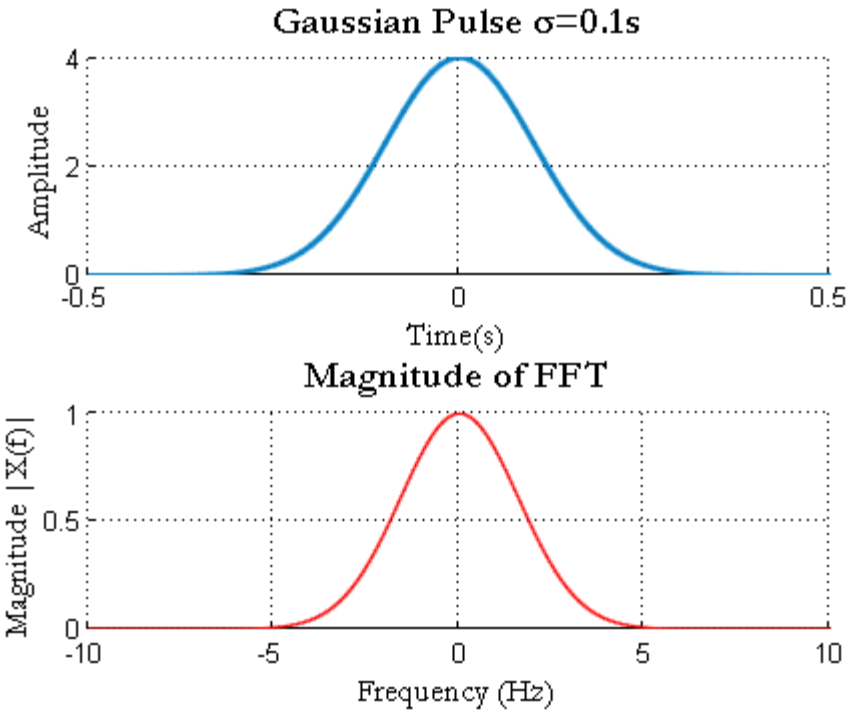
This programming assignment is focused on filtering in the spatial domain. You will write some functions to create a filter kernel, and apply the filter on input images. Then, by computing the gaussian and laplacian pyramids, you will visualize the low and high frequency components of an image.

The template of all the needed functions is provided in a **.cpp** file along with the input images. In this assignment, you should work with gray images only. Use CV_LOAD_IMAGE_GRAYSCALE when reading your images. Save all of your images in **PNG format**.

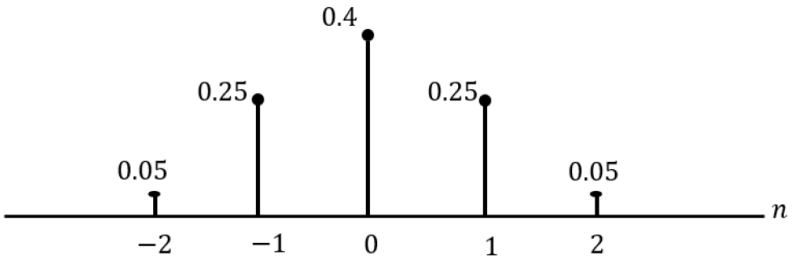
Gaussian Filter:

I. Gaussian Kernel

a. Gaussian filters are widely used in image processing due to their frequency and spatial characteristics. A gaussian filter's frequency response is also gaussian. They are better than box filters since they cut the higher frequencies and do not have frequency leakage. In the figure below, the gaussian filter in both time **(blue)** and frequency **(red)** domain is shown. Both the filter kernel and its frequency response are zero beyond certain values.



b. One good discrete approximation for gaussian filter in 1D is as follows. The center of the filter is in the middle with value 0.4. Note that the sum of the weights is equal to 1.



c. Now, we want to create a 5x5 gaussian kernel out of the above 1D kernel for image processing purposes. In order to do that, assume this 1D kernel as a Nx1 column vector and multiply it by its transpose (a 1xN row vector). You should do this and create the 5x5 gaussian kernel in the function CreateGaussianFilter().

$$2D\ kernel = [0.05\ 0.25\ 0.4\ 0.25\ 0.05\ ]' * [0.05\ 0.25\ 0.4\ 0.25\ 0.05\ ]$$

Complete the body of the function *CreateGaussianFilter()* that returns the 5x5 gaussian kernel. Submit the completed code for this function.

II. Filtering

a. In the previous programming assignment, you wrote a filtering function that applies a 3x3 box filter on an input image. In this assignment, you should write a more general filtering function. The function *ApplyFilter* takes as input an image and a filter kernel as OpenCV's *Mat* and returns the filtered image. Here are the assumptions for this function:
   1. The input image is a single channel (gray) image (CV_8UC1)
   2. The kernel is a single channel *Mat* with float values (CV_32FC1)
   3. The kernel is always square with odd width (3, 5, 7, etc)
   4. The center of the filter (coordinate (0,0)) is at the center of the kernel.

b. You should go through each pixel of the output image and perform a 2D convolution to find the value for that output image pixel. Note that the filter values are floating points and input and output image values are unsigned char. You should be careful about choosing the appropriate data types and type conversions when performing arithmetic operations.

c. In the previous programming assignment, you implemented two edge-handling techniques: zero-padding and mirroring. While zero-padding is mathematically correct, it does not yield good results. Use the mirroring method in your filter function.

Complete the body of the function *ApplyFilter(Mat input, Mat filter)***that takes as input a gray image and a filter kernel and returns the filtered image. Submit the completed code.**

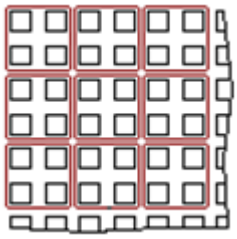
Gaussian and Laplacian Pyramid

III. Gaussian Pyramid

a. As you learned in lectures, the image (as a 2D signal) can be converted to frequency domain using Fourier Transform. The frequency space will show us the information about the content of the image. The high frequency contents are sharp edges and the lower frequency contents are wider objects in the image. In this part of assignment you will create a gaussian pyramid of the input image.

b. Using the gaussian filter and filtering function you created in the previous sections, filter the images in the input image gallery. Then, halve the size of the filtered image using *Reduce* operation. This is a subsampling operation done in the *Reduce*

function. This function takes the input of size WxW and returns the same image, but of size (W/2)x(W/2), by averaging four pixels and assigning it to the corresponding output pixel. As illustrated in the figure below, you should take the average of pixels in a red box and assign it to corresponding pixel in the output image.



Complete the body of the function *Reduce(Mat input)* **that takes as input an image and returns the output image with half the size in both dimensions.**

c. For every step after you filtered the image, you should reduce the size to half. The original images are 512x512, which is the level 0 of the pyramid. Now, after filtering and reducing, the level 1 image will be 128x128. You should continue up to level 8 which is a single pixel. Then, resize each of these images to 512x512 using the OpenCV *resize* function. Here is how to use this function:

```
// I is the image needs to be resized to 512x512
// J is the result of resizing which is 512x512
resize(I, J, Size(512,512));
```

You should create all the levels of the gaussian pyramid and save the resized (512x512) images of each level by following the naming convention *<img>_g<level_num>.png*. For example, if the original name is *face.png*, you will save your images as *face_g0.png*, *face_g1.png* and so on. Submit all of these images and include them in your PDF file.

IV. Laplacian Pyramid

a. Each level of gaussian filtering and subsampling (*reduce*) removes some of high-frequency components of the original

image. In every step, we halve the cut-off frequency and retain less information of the image. So, if you subtract two levels of the gaussian pyramids images, you will get the bandpass components between those cut-off frequencies. Once you visualize these band-pass levels, you can see the different levels of details of the image in each step.

b. Recall that gray images are in the range of [0 to 255] and the data type to store them is unsigned char. However, subtracting two such values can result in a range of [-255 to 255]. Such a range of values cannot be stored in a byte. In this section, you should complete the function *Deduct* that takes two images of type unsigned char (CV_8UC1) as input, calculates their difference and normalizes the result between 0 to 255 in order to return it to a gray image of the same type as the input. For this purpose, you should go through the pixels of input images, calculate their difference and keep them in a intermediate *Mat* of type int. As you calculate the intermediate *Mat* values, you should keep track of the maximum and minimum values of the difference. You will use these max and min values for the normalization step. In the normalization step, each pixel of the output image should be calculated as:

$$output\ pixel = 255\ \frac{difference - minVal}{maxVal - minVal}$$

Complete the *Deduct(Mat I, Mat J)* **function that takes the input images I and J and returns the normalized image of their difference I – J. You should submit your completed code for this function.**

c. Once you have the *Deduct* function, you can calculate the difference between consecutive levels of your gaussian pyramid and construct your laplacian pyramid. You should do this by deducting the first level image (e.g. *face_g1.png*) from the original image (*face_g0.png*) and name it (e.g. *face_L1.png*). Continue doing this for all the levels, start from level 0 – level 1 and end with level 8 – level 9. There will be total of eight images.

You should create all the levels of laplacian pyramid and save them by following the naming convention *<img>_L<level_num>.png*. For example, if the original name is *face.png*, you will save your images as

***face_L1.png*, *face_L2.png* and so on. Submit all of these images and include them in your PDF file.**