Homework 1
M1522.001000 Computer Vision (2020 Fall)
Due: Sep 24 Thursday 11:59PM.

1 Overview

In this assignment, you will implement basic filters and image pyramids using Python. There are also two writing questions at the end.

All the files you need are in `hw1.zip` from the course ETL. Download and unzip the file to create a `hw1` directory. You will do your work here. The directory contains:

- `processs_images.py`, a partial Python program that you need to complete.
- `images`, a directory that contains test images.
- `utils.py`, some utility code for handling images. You don't need to modify this file.

Make sure you have Python 3.6+ installed. You can check your version with command `python3 -V`. To run the program, you will also need Numpy and Pillow packages. To install them with the package manager for Python, run: `pip3 install numpy pillow`.

You can test your program by running `python3 process_images.py`. Without your implementation, the program will just load the test images and exit.

Since there are many students in this course, the grading of your homework will be highly automated with (hopefully) little TA intervention. To avoid the risk of your submission being skipped or losing marks, you should make sure:

- Your program runs without an error and is reasonably fast.
- Your program does not use any third-party image processing library (including `ImageFilter` in Pillow). You can import any Python built-in packages you want.
- The names and signatures of the required functions remain unchanged.
- You follow the submission rule below.

After finishing your assignment, zip your files again and email it to `ta.cv@vision.snu.ac.kr`. Your email subject **must** be in format `[hw1]your-student-number`. For example, `[hw1]2020-12345`.

Finally, note that we will use a code similarity checker to detect plagiarism. You are expected to work on the assignment individually. For your sake, please do not copy and paste other student's code!

2 Image Filters [20 pts]

In this part, you should implement four functions related to filtering an image. From now on, we refer to `pixel_grid` as a three-dimensional Numpy array representing an RGB image. The code for building this grid is already given to you in `utils.py`.

- `get_pixel_at(pixel_grid,i,j)`
  Return RGB values of pixel at row i and column j. If there is no row i and column j, return zeros.
- `get_patch_at(pixel_grid,i,j,size)`
  Return image patch at row i and column j that has height and width of *size*. The output should be a three-dimensional Numpy array with *size* rows and *size* columns. In edge cases where some surrounding pixels do not exist (for example, top-left pixel), the output array should include zeros for non-existing pixels.
- `apply_gaussian_filter(pixel_grid,size)` Apply gaussian filter for every pixel in *pixel_grid* and return the resulting pixel grid. Use *size* as the filter size.
- `apply_median_filter(pixel_grid,size)` Apply median filter for every pixel in *pixel_grid* and return the resulting pixel grid. Use *size* as the filter size.

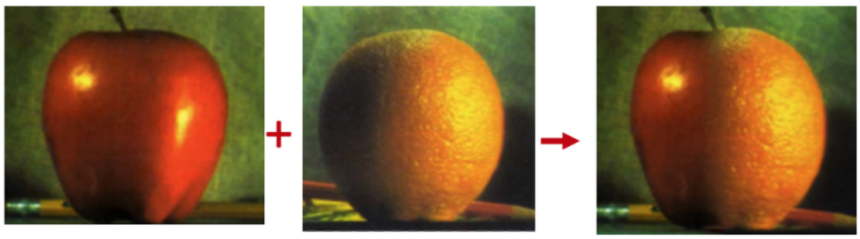
As you learned in class, Guassian filters are useful for removing Gaussian noise from an image. You can test your Gaussian filter function with the test image `images/dog_gaussian_noise.png`.

On the other hand, median filters are useful for more discrete types of noise. Test your median filter function with the test image `images/dog_salt_and_pepper.png`.

3 Image Pyramids [20 pts]

In this section, you will build Gaussian and Laplacian pyramids from images and use them to blend images. To do that, you will need to implement three functions.

- `build_gaussian_pyramid(pixel_grid,size,levels=5)`
  Build and return a gaussian pyramid with given number of levels. You can make use of the downsampling function in `utils.py` here.
- `build_laplacian_pyramid(gaussian_pyramid)`
  Build and return a laplacian pyramid from gaussian pyramid. You can make use of the upsampling function in `utils.py` here.
- `blend_images(left_image,right_image)`
  With `build_gaussian_pyramid` and `build_laplacian_pyramid`, you can implement smooth blending of two images. This function should take left half of *left_image* and right half of *right_image* and return smoothly concatenated version.



The above example shows an example of blending apple and orange images. Test your image blending function with `images/player1.png` and `images/player2.png`. See if the result looks more natural than simple concatenation.

4 Writing questions [10 pts]

- Gaussian filters use distance to the center pixel to determine the weights of the nearby pixels. On the other hand, bilateral filters also consider the difference in pixel intensity. Discuss the pros and cons of bilateral filters compared to Gaussian filters and discuss appropriate use cases. [5pts]
- Prove the shifting property of Fourier transform. [5pts]

Write a document with your solutions (`solution.pdf`) and include it in the zip file for submission.