Homework Set 1:
Image Filtering and Hybrid Images

Instructions

- This homework contains two parts, i.e., a set of questions (20 pts, in Section 1) and a programming project (80 pts, in Section 2).
- Upload a zipped file named as `id_yourname.zip` via the Beihang SPOC platform, including
  (1) an electronic version (saved as `id_name.pdf`, can be printed or handwritten) of your report, it includes the answers of the questions in Section 1; and describes the algorithm process, shows the results and discussions (if required) in Section 2.
  (2) a folder (named as `id_name`) includes the codes and results. You may use a Jupyter Notebook file (saved as `id_name.ipynb`) instead.
  For more details about python and Jupyter Notebook, as well as some useful packages (such as `numpy` and `matplotlib`), please check the following link.
- The deadline is 23:59 pm, April 10th, 2024.

1 Questions (20 pts)

1.1 (10 pts) **Convolution**, a type of image filtering, is a fundamental image processing tool that you will use repeatedly throughout the course.

(a) *Explicitly describe* the 3 main components of **convolution**, *i.e.* (i) input, (ii) transformation (how it happens), and (iii) output.

(b) How is this operation different from **correlation**? Please check that **covolution** is linear shift-invariant, but **correlation** is not.

(c) Construct a scenario which produces a different output between both operations. Include the kernel you used and your image results.

Please use `scipy.ndimage.convolve` *and* `scipy.ndimage.correlate` *to experiment!*

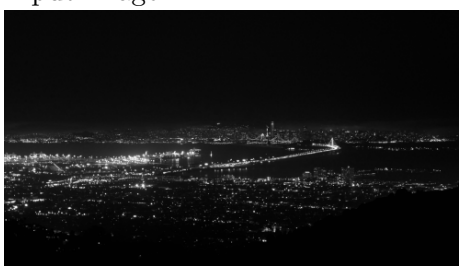
1.2 (10 pts) Answer the following sub-questions. For (a–c), which filter does the kernel represent? For (d–e), which filter produced the output images?

(a) $\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$ (A) High pass (B) Low pass (C) Neither

(b) $\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$ (A) High pass (B) Low pass (C) Neither

(c) $\begin{bmatrix} -\frac{1}{9} & -\frac{1}{9} & -\frac{1}{9} \\ -\frac{1}{9} & \frac{8}{9} & -\frac{1}{9} \\ -\frac{1}{9} & -\frac{1}{9} & -\frac{1}{9} \end{bmatrix}$ (A) High pass (B) Low pass (C) Neither

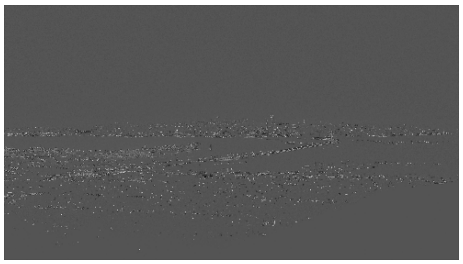
(d) Input image:



Output image 1:

(A) High pass
(B) Low pass

(e) Output image 2:



(A) High pass
(B) Low pass

2 Programming Project (80 pts)

2.1 Template Matching with Cross-correlation (30 pts)

Let's define cross-correlation of an image f with a template g as follows:

$$(g * *f)[m,n] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} g[i,j] \cdot f[m+i, n+j] \tag{1}$$

In the following, we will show how to match a template with an image by cross-correlation, and how to improve it.

2.1.1 Matching with Cross-correlation (10 pts)

Suppose that you are a clerk at a grocery store. One of your responsibilities is to check the shelves periodically and stock them up whenever there are sold-out items. You got tired of this laborious task and decided to build a computer vision system that keeps track of the items on the shelf.

Luckily, you have learned in class that cross-correlation can be used for template matching: a template g is multiplied with regions of a larger image f to measure how similar each region is to the template.

Rubric

(a) The template of a product (`template.jpg`) and the image of shelf (`shelf.jpg`) is provided. We will use cross-correlation to find the product in the shelf.

Figure 1: The shelves at a grocery store

Figure 2: your template

(b) Implement `cross_correlation()` function in `student.py` and run the test function `naive_cross_correlation()` in `proj1.py`.

(c) How does the output of cross-correlation filter look? Explain what problems there might be with using a raw template as a filter.

2.1.2 Matching with Zero-mean cross-correlation (10 pts)

A solution to the aforementioned problem is to subtract the mean value of the template so that it has zero mean.

Rubric

(a) Implement `zero_mean_cross_correlation()` function in `student.py` and run the test function `simple_zero_mean_cross_correlation()` in `proj1.py`.

If your implementation is correct, you should see the blue cross centered over the correct cereal box.

(b) You can also determine whether the product is present with appropriate scaling and thresholding, using the function `check_product_on_shelf()` in `proj1.py`.

2.1.3 Matching with Normalized Cross-correlation (10 pts)

One day the light near the shelf goes out and the product tracker starts to malfunction. The `zero_mean_cross_correlation()` is not robust to change in lighting condition. The function `change_in_lighting_condition()` in `proj1.py` demonstrates this.

A solution is to normalize the pixels of the image and template at every step before comparing them. This is called *normalized cross-correlation*. The mathematical definition for normalized cross-correlation of f and template g is:

$$(g \star f)[m,n] = \sum_{i,j} \frac{g[i,j] - \overline{g}}{\sigma_g} \cdot \frac{f[m+i, n+j] - \overline{f_{m,n}}}{\sigma_{f_{m,n}}} \tag{2}$$

where:

1. $f_{m,n}$ is the patch image at position (m, n)
2. $\overline{f_{m,n}}$ is the mean of the patch image $f_{m,n}$
3. $\sigma_{f_{m,n}}$ is the standard deviation of the patch image $f_{m,n}$
4. $\overline{g}$ is the mean of the template g
5. σ_g is the standard deviation of the template g

Rubric

(a) Implement `normalized_cross_correlation()` function in `student.py` and run the test function `final_normalized_cross_correlation()` in `proj1.py`.

2.2 Hybrid images (50 pts)

We will write an image convolution function and use it to create **hybrid images**! The technique was invented by Oliva, Torralba, and Schyns in 2006, and published in a paper at SIGGRAPH 2006. High frequency image content tends to dominate perception but, at a distance, only low frequency (smooth) content is perceived. By blending high and low frequency content, we can create a hybrid image that is perceived differently at different distances.

2.2.1 Image Filtering

This is a fundamental image processing tool, and is crticial in this project. Common computer vision software packages have efficient functions to perform image filtering, but we will write our own from scratch via *convolution*.

2.2.2 Hybrid Images

A hybrid image is the sum of a low-pass filtered version of a first image and a high-pass filtered version of a second image. We must tune a free parameter for each image pair to controls *how much* high frequency to remove from the first image and how much low frequency to leave in the second image. This is called the "cut-off frequency". The paper suggests to use two cut-off frequencies, one tuned for each image, and you are free to try this too. In the starter code, the cut-off frequency is controlled by changing the standard deviation of the Gausian kernel used to construct a hybrid image.

(a) We provide 5 pairs of aligned images which can be merged reasonably well into hybrid images. The alignment is important because it affects the perceptual grouping (read the paper for details). We encourage you to create additional examples, e.g., change of expression, morph between different objects, change over time, etc.

(b) Hybrid Image Example. Figure 3 shows the original images. The low-pass (blurred) and high-pass versions of these images are shown in Figure 4. The high frequency image is actually zero-mean with negative values so it is visualized by adding 0.5 to each pixel value. In the resulting visualization, bright values are positive and dark values are negative. Adding the high and low frequencies together gives you the image in the left of Figure 5. If you're having trouble seeing the multiple interpretations of the image, a useful way to visualize the effect is by progressively downsampling the hybrid image (shown as the downsampled images in Figure 5). The starter code provides a function in `helpers.py` as `vis_hybrid_image()` to save and display such visualizations.

Figure 3: Original images.

Figure 4: Low-frequency image *versus* high-frequency image.

Figure 5: Progressively downsampling the hybrid image.

2.2.3 Requirements / Rubric

(a) **(0 pts)** Your `my_imfilter()` code must complete within one minute on the cat or dog test images. This means you cannot write around four or five nested for loops. Some loops are necessary, but you **must use some `numpy` operations** to complete the filtering operation.

(b) **(20 pts)** Implement convolution in `student.py` as `my_imfilter()`. Your filtering algorithm should:

(i) **(4 pts)** Pad the input image with zeros.

(ii) **(4 pts)** Support grayscale and color images. Note that grayscale images will be **2D `numpy` arrays**.

(iii) **(4 pts)** Support arbitrary shaped odd-dimension filters (e.g., 7x9 filters but not 4x5 filters).

(iv) **(4 pts)** Raise an Exception with an error message for even filters, as their output is undefined.

(v) **(4 pts)** Return a filtered image which is the same resolution as the input image.

We have provided `proj1_part1.py` to help you debug your image filtering algorithm.

(c) **(15 pts)** Implement hybrid image creation in `student.py` as `gen_hybrid_image()`.

We have provided `proj1_part2.py` to help you generate hybrid images.

(d) **(10 pts)** Extra requirements (max 10 pts total, you may not finish all requirements).

(i) **(4 pts)** : Pad with reflected image content.

(ii) **(4 pts)** : Your own hybrid image included in your report, formed from images not in our code distribution.

(iii) **(10 pts)** : FFT-based convolution. You can use an existing implementation of the Fourier transform for this. Please fill in the function `my_imfilter_fft()` in `student.py` to supplement your existing (required) spatial convolution.

(e) **(5 pts)** Report. Please describe your process and algorithm, show your results, describe any extra requirement, and tell us any other information you feel is relevant.

Forbidden functions Anything that filters or convolves or correlates for you, e.g., `numpy.convolve()`, `scipy.signal.convolve2d()`, `scipy.ndimage.convolve()`, and `scipy.ndimage.correlate()`.

Potentially useful functions Basic numpy operations, e.g., addition `numpy.add()`, element-wise multiplication `numpy.multiply()`, summation `numpy.sum()`, flipping `numpy.flip()`, range clipping `numpy.clip()`, padding `numpy.pad()`, rotating `numpy.rot90()`, etc.

Feedback? (Optional)

Please help us make the course better. If you have any feedback for this assignment, we'd love to hear it!