ECE379K: Assignment #1

Due: Feb. 14 2024, 11:59pm TX

Introduction

Answer the following questions and explain solutions. Numbers in parentheses give maximum credit value. You can discuss in small groups, but turn in individual solutions and indicate collaborators. Do not use code from the Internet or high-level functions from other Python libraries (such as image pyramid or edge detection functions), unless specific permission is given.

Turn in assignments by Wednesday, Feb. 14, 11:59 pm TX. Submit to Canvas (1) a .pdf with answers, descriptions, and figures; and (2) a .zip file containing your code. The pdf can be created using latex or converting a word document to pdf or any other method you prefer, as long as it is organized and easy to read.

Info: The overall goal of this homework is to review the basics of filtering and to establish your ability to display and work with images. The time to complete the homework varies widely with expertise and degree of debugging required.

1 Image Pyramids (35%)

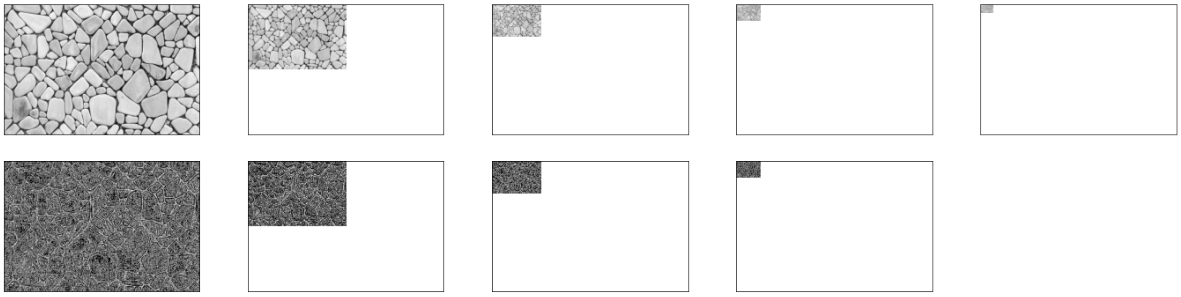

Figure 1: Example of Gaussian and Laplacian pyramid (please use the provided ''`texture.jpeg`'' instead).

Load the provided image ''`texture.jpeg`'' and convert it to grayscale. Write code for a Gaussian and Laplacian pyramid of 5 levels (use `for` loops). At each level, the resolution should be reduced by a factor of 2. Show your pyramids and include the code in your submission.

1. Display a Gaussian and Laplacian pyramid of level 5 (write your code). It should be formatted similarly to Figure 1 above. You may find the included ''`tight_subplot.py`'' helpful for making plots. (15%)
2. Display the FFT amplitudes of your Gaussian/Laplacian pyramids. Appropriate display ranges (''`vmin`'', ''`vmax`'') should be chosen so that frequencies in different pyramid levels are clearly visible. Explain what the Laplacian and Gaussian pyramids are doing in terms of frequency. (20%)

Useful functions include: `cv2.getGaussianKernel, cv2.filter2D, np.fft.fftshift, np.fft.fft2`.

2 Edge Detection (55%)

Figure 2: Problem 2 (from BSDS)

In this problem, we will detect the edges of the provided image "`plane.jpg`".

2.1 Build a simple gradient-based edge detector that includes the following functions (25%)

Complete two functions below, and visualize your detected edges.

```
def gradientMagnitude(im: np.ndarray, sigma: float):
    return mag: np.ndarray, theta: np.ndarray
```

This function should take an RGB image as input, smooth the image with a Gaussian filter (std=sigma), compute the x and y gradient values of the smoothed image, and output image maps of the gradient magnitude and orientation at each pixel. You can compute the gradient magnitude of an RGB image by taking the L2-norm of the R, G, and B gradients. The orientation can be computed from the channel corresponding to the largest gradient magnitude. The overall gradient magnitude is the L2-norm of the x and y gradients. `mag` and `theta` should be the same size as `im`.

```
def edgeGradient(im: np.ndarray):
    return bmap: np.ndarray
```

This function should use `gradientMagnitude` to compute a soft boundary map and then perform non-maxima suppression. You could use the provided `nonmax.py`.

Useful functions include: `cv2.getGaussianKernel`, `cv2.filter2D`, `cv2.spatialGradient`, `np.take_along_axis`.

2.2 Try to improve your results using a set of oriented filters, rather than the simple derivative of Gaussian approach above, including the following functions: (30%)

Complete two functions below, and visualize both filters you used and your detected edges.

```
def orientedFilterMagnitude(im: np.ndarray):
    return mag: np.ndarray, theta: np.ndarray
```

Computes the boundary magnitude and orientation using a set of oriented filters, such as elongated Gaussian derivative filters. Explain your choice of filters. Use at least four orientations. One way to combine filter responses is to compute a boundary score for each filter (simply by filtering with it) and then use the max and argmax over filter responses to compute the magnitude and orientation for each pixel.

```
def edgeOrientedFilters(im: np.ndarray):
    return bmap: np.ndarray
```

Similar to Problem 2.1, this should call `orientedFilterMagnitude`, perform the non-maxima suppression, and output the final soft edge map.

Useful functions include: `np.mgrid, scipy.stats.multivariate_normal`.

Write-up: Include your code with your electronic submission. In your write-up, include:

- Description of any design choices and parameters.

- Visualize the bank of filters used for Problem 2.2 (`pyplot.imshow` may help with visualization).
- Compare your detected edges in Problem 2.1 and Problem 2.2.

2.3 Ideas for improvement (10%)

Describe at least one idea for improving the results. Sketch an algorithm for the proposed idea and explain why it might yield improvement. Improvements could provide, for example, a better boundary pixel score or a better suppression technique. Your idea could come from a paper you read, but cite any sources of ideas.