Lab Assignment #4

Due Monday, November 13 by 11:59pm

(upload as a single PDF to CatCourses)

Title: Linear Spatial Filtering and Edge Detection

In this lab, you will implement linear spatial filtering. Once you have gotten your linear spatial filtering to work for image smoothing by averaging, you will use it to perform simple edge detection. In particular, you will compute the gradient magnitude of an image by filtering with Sobel filters. You will then apply a threshold to the gradient magnitude to create a binary edge image.

Part 1: Linear Spatial Filtering

Familiarize yourself with the Python script `test_SpatialFiltering.py`. This script:

- Creates a simple 100x100 pixel image in which the left half is black (0) and the right half is white (255).
- Creates a 3x3 "impulse filter" which has a 1 in the center and 0 elsewhere. Performing linear spatial filtering with an impulse filter does not change the image.
- Calls the function `spatial_filter` (which you must write) to apply the impulse filter to the simple image. This should not change the image.
- Creates a 3x3 averaging filter with has the value 1/9 everywhere.
- Calls the function `spatial_filter` (which again you must write) to apply the averaging filter to the simple image. This should result in a slightly smoothed version of the simple image.
- Repeats the above impulse and average filtering for the image `watertower.tif`.

Your tasks:

a) Create the script `Lab04_functions.py` and write the function `spatial_filter` that takes two inputs: a grayscale image (a numpy array of pixel values) and a filter (also a numpy array of filter values). This function applies the filter to the image as described in section 3.4.1 of the 3rd edition of the text or section 3.4 of the 4th edition. Specifically, it should implement the equation at the bottom of page 145 of the 3rd edition or equation 3.4 of the 4th edition. Your function should output the

filtered image (as a numpy array of pixel values). This filtered image should have the same dimensions as the input image. You can assume that the filter is odd sized. Use zero padding to extend the image when the filter goes beyond the boundaries of the image.

Use the provided script `test_SpatialFiltering.py` to develop and test your `spatial_filter` function. Specifically, filtering an image with an impulse function should not change the image. Filtering an image with an averaging filter should smooth the image. Here is what I got when applying the 3x3 averaging filter to the simple and `watertower.tif` images:

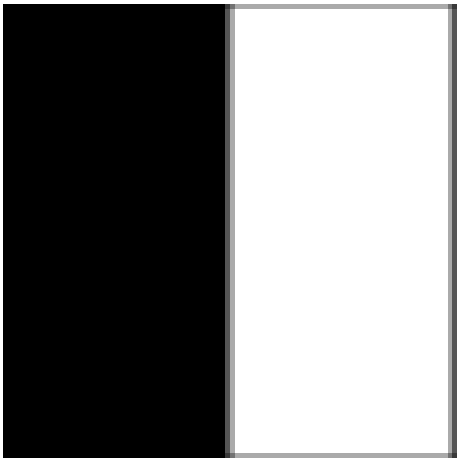

Figure 1: Result of applying the 3x3 averaging filter to the simple image. The dark regions along the edges of the image are due to zero-padding when applying the filter.

Figure 2: Result of applying the 3x3 averaging filter to the water tower image.

Part 2: Edge Detection

Once you have gotten your function `spatial_filter` to work in Part 1, you can move on to using it to perform edge detection.

Familiarize yourself with the Python script `test_EdgeDetection.py`. This script:

- Creates a simple 100x100 pixel image in which the left half is black (0) and the right half is white (255).
- Calls the function `find_edges` (which you must write) to perform edge detection in the simple image. It does by computing the gradient magnitude and applying a threshold to it. The gradient magnitude is computed by spatial filtering with Sobel filters.
- Calls the function `find_edges` (which again you must write) to perform edge detection in the `watertower.tif` image.

Your tasks:

a) In your file `Lab04_functions.py` write the function `gradient_magnitude` that takes one input: a grayscale image (a numpy array of pixel values). This function should return an image (a numpy array of pixel values) with the same dimensions as

the input. This output contains the magnitude of the gradient. You should compute the gradient using the Sobel masks shown in figure 10.14 (both editions) using your `spatial_filter` function. (Remember, the two components of the gradient vector can be calculated with the two Sobel masks. To compute the magnitude of this vector, you simply compute the length of the gradient vector.)

b) Write the function `find_edges` which takes two inputs: a grayscale image (a numpy array of pixel values) and a threshold value (a scalar). This function should return an "edge" image **(**a numpy array of pixel values) with the same dimensions as the input. The edge image should have a value of 255 where edges are detected and 0 elsewhere. Your `find_edges` function should detect edges by applying a threshold to the magnitude of the gradient as computed using your `gradient_magnitude` function. Pixels whose gradient magnitude exceeds the threshold should have a value of 255 in the output image and 0 where the gradient magnitude is less than or equal to the threshold elsewhere. In summary, `find_edges` calls `gradient_magnitude` which makes calls to `spatial_filtering`.

Use the provided script `test_EdgeDetection.py` to develop and test your `find_edges` and `gradient_magnitude` functions. When applying `find_edges` function to the `watertower.tif` image, experiment with different threshold values to find a value which gives a reasonable edge image. Here is what I got when applying my `find_edges` function to the simple and `watertower.tif` images:

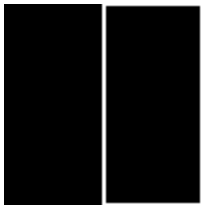

Figure 3 Result of applying my `find_edges` function to the simple image with a threshold value of 200. Note that in addition to the edge down the middle of the image, edges were detected on the other boundaries of the white part of the image. This is due to zero-padding when applying the Sobel filters.

Figure 4: Result of applying my `find_edges` function to the water tower image with a threshold value of 200.

Questions:

1. The effects of different threshold values when applying your `find_edges` function to the water tower image.
2. Figure 5 below shows the result of applying the Canny edge detector to the water tower image.

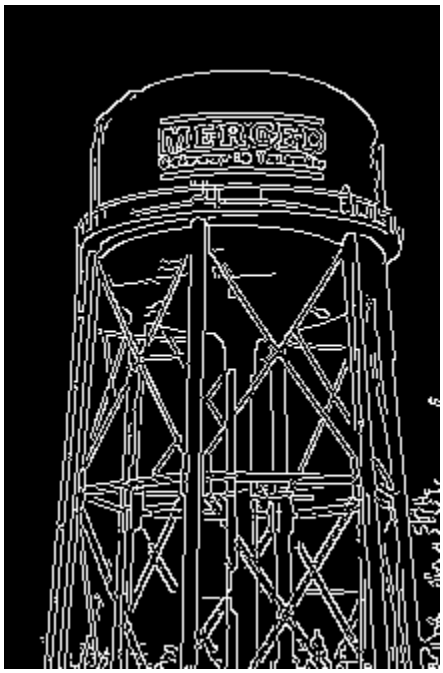

Figure 5: Result of performing Canny edge detection on the water tower image.

(You can get this result by running the provided Python script `Canny_edge_detection.py`. This script uses the OpenCV Python package which you might need to install.)

Discuss how the edge image created using your `find_edges` function is different from the one in Figure 5 created using the Canny edge detector. What aspects of the Canny edge detector explain these differences.

3. What was the most difficult part of this assignment?

What to submit:

A lab report as a single PDF containing:

- Lab number and title, your name, your lab section, your TA's name, and the date.
- Abstract: In a few sentences, describe the purpose of this lab.
- Figures containing:
    - Your results of applying the 3x3 averaging filter to the water tower image.
    - The results of applying your `find_edges` function to the water tower image.

- Questions: Your answers to the three questions above.
- The scripts `test_SpatialFiltering.py` and `test_EdgeDetection.py`. (These could be exactly the same as the ones provided or with some modifications.)
- Your Python file `Lab04_functions.py` which contains your three functions: `spatial_filter`, `find_edges`, and `gradient_magnitude`.

What your report will be graded on

- Whether you included an abstract.
- Whether your report includes the figures you were asked to include and they have appropriate captions.
- Whether you answered the assignment questions.
- Whether your report includes the code you were asked to include.
- Whether your code is correct.
- Whether your code is commented and your functions have headers.