CS 3430: Scientific Computing
Assignment 05
Edges as Derivatives of Luminosity

Vladimir Kulyukin
Department of Computer Science
Utah State University

February 12, 2022

Learning Objectives

1. Edges as Derivatives of Luminosity
2. Edge Detection
3. PIL: Python Image Library

Introduction

A fundamental theme of this assignment is the derivative, a fundamental tool of scientific computing (and, of course, Calculus). Derivatives surface where one wouldn't expect to see them. Edge detection is one of the domains where derivatives become handy, because detecting edges can be thought of as taking derivatives of luminosity.

Problem 1 (3 points)

In this problem, we'll implement the simple edge detection algorithm we learned in lectures 09. You'll need to install the Python Image Library (PIL) for this assignment. Go to https://python-pillow.org/ and click on the installation instructions link to install PIL on your computer.

You'll save your solutions in `cs3430_s22_hw05.py` and submit it in Canvas. I've included my unit tests `cs3430_s22_hw05_uts.py` for this problem. You'll quickly notice that these unit tests don't contain assertions. Edge detection is as much of an art as a science. A useful edge or line in one domain (e.g., road lane detection) is useless in a different one (e.g., plant classification).

Once you install PIL on your computer, it's very easy to load the image. If the `inpath` is the string with an image path to the image, then all you need to do is this.

```
input_image  = Image.open(inpath)
```

Here's what you need to do if you want to save the image.

```
output_image.save(outpath)
```

Once you're done with the image, you may want to delete it.

```
del input_image
```

Here's how we can load an image, process it with the function `depil()` (read below), saving the output and then deleting both images.

```
input_image  = Image.open(inpath)
output_image = depil(input_image, default_delta=default_delta,
                     magn_thresh=magn_thresh)
output_image.save(outpath)
del input_image
del output_image
```

Implement the function `depil(pil_img, default_delta=1.0, magn_thresh=20)` in `cs3430_s22_hw05.py`. The function name abbreviates the phrase "detect edges with PIL." This function takes a PIL image `pil_img` and applies the edge detection algorithm from Lecture 09. It returns a new one channel PIL image where the edge pixels are labeled as white (i.e., their pixel value is 255) and non-edge pixels are labeled as black (i.e., their pixel value is 0). The keyword parameter `default_delta` is used in computing the vertical and horizontal luminosity changes at pixels. The `magn_thresh` specifies the value of the gradient's magnitude at a pixel for the pixel to qualify as an edge pixel. When you work on this function, remember that in PIL pixels are referenced in a column-first manner. In other words, a pixel at (x, y) is at column x and row y.

Let's implement this function step by step. Start by implementing the function `pil_pix_dxdy(pil_img, cr, default_delta)` in `cs3430_s22_hw0.py` that returns a 2-tuple (dx, dy) of the horizontal and vertical changes in luminosity at a pixel in a PIL image `pil_img` whose position is specified by the 2-tuple `cr` where `cr[0]` is the pixel's column and `cr[1]` is the pixel's row. The luminosity values are computed with the relative luminosity formula implemented in the `lumin` function in `cs3430_s22_hw05.py`.

Recall from Lecture 09 that the vertical change (i.e., dy) is computed as the difference between the luminosities of the pixel's upper and lower neighbors (i.e., the two pixels at positions `(cr[0],cr[1]-1)` and `cr[0],cr[1]+1)` and the horizontal change (i.e., dx) is computed as the difference between the luminosities of the pixel's right and left neighbors (i.e., the two pixels at positions `(cr[0]+1,cr[1])` and `(cr[0]-1,cr[1])`.

Implement `grd_magn(dx, dy)` that takes the dx and dy values computed by `pil_pix_dxdy()` and computes the gradient's magnitude from them. Proceed with the implementation of the function `grd_deg_theta(dx, dy)` that computes the theta (i.e., the orientation of the gradient) from dx and dy. This function should return the degree value. Why degrees? What's wrong with radians? Absolutely nothing! However, in my opinion, it's much easier to debug with degrees than radians.

Everything is in place now for us to implement `depil()`. As you iterate through the image's pixel and compute magnitude and orientation of the gradient at each pixel, you may want to covert the magnitude and orientation values to integers. This is, of course, lossy, but most functions in image processing are. Also, throw in an assertion `assert abs(th) <= 180` to make sure that your orientations make sense. The function `depil()` thresholds only on magnitudes but, when you're debugging, orientations are a great help. You can also add orientation thresholding to your implementation in the future.

The file `cs3430_s22_hw05_uts.py` has a few unit tests for Problem 01. Each test corresponds to a separate image. All the test images (JPG and PNG) are in the directory `imgs`. The unit tests, when you run them, save your output images in the directory `out_imgs`. Your output images should look similar to mine. When we run the unit tests, we won't be comparing your output images to mine pixel by pixel for exact matches. But, your images should look sufficiently similar, especially the edge pixels detected in simple images such as `EdgeImage_01.jpg`.

Unit Test Outputs

In this section, I give you the output of the unit tests in `cs3430_s22_hw05_uts.py` when I tested them with my implementation. The directory `out_imgs` contains the images that my code generates for each unit test. The name of every image with detected edges has ends with `_ed`.

What To Submit

Submit your code in `cs3430_s22_hw05.py`. It'll be easiest for us to grade your code if you place all the files, zip it into `hw05.zip`, and upload your zip in Canvas.

Happy Hacking!

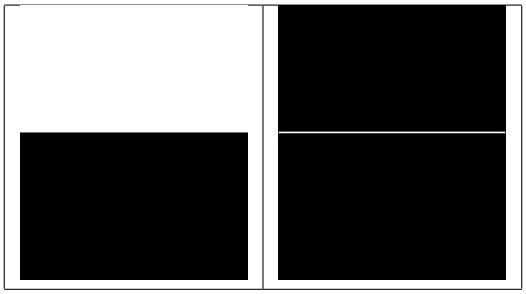

Figure 1: Unit Test 01: input image (left); output image (right).

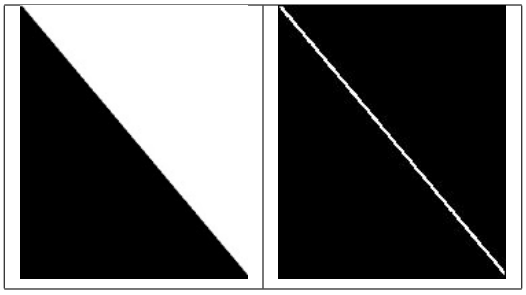

Figure 2: Unit Test 01: input image (left); output image (right).

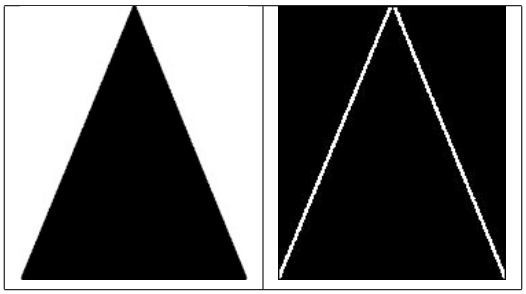

Figure 3: Unit Test 03: input image (left); output image (right).

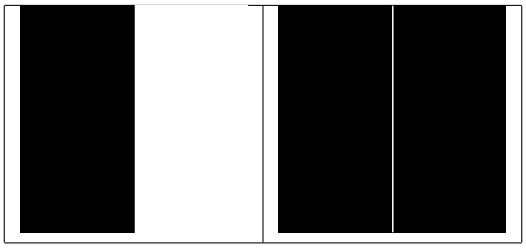

Figure 4: Unit Test 04: input image (left); output image (right).

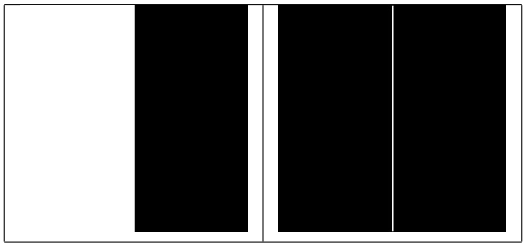

Figure 5: Unit Test 05: input image (left); output image (right).

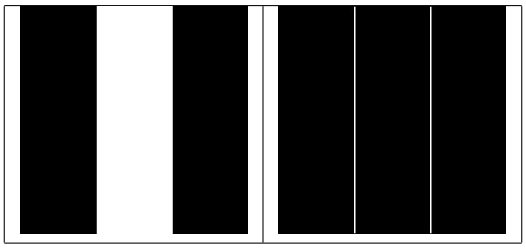

Figure 6: Unit Test 06: input image (left); output image (right).

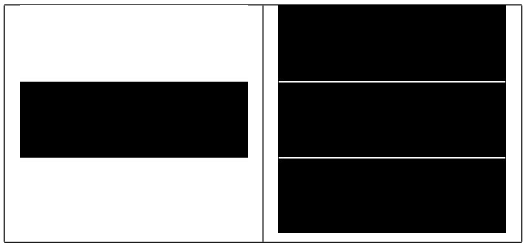

Figure 7: Unit Test 07: input image (left); output image (right).

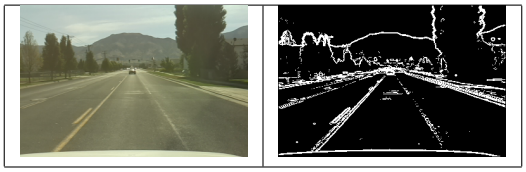

Figure 8: Unit Test 08: input image (left); output image (right).

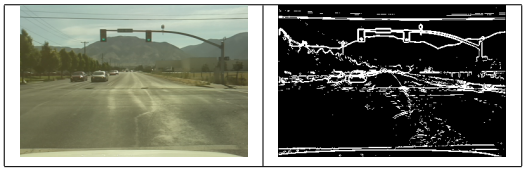

Figure 9: Unit Test 09: input image (left); output image (right).

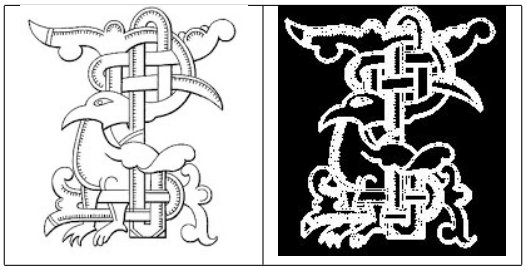

Figure 10: Unit Test 10: input image (left); output image (right).

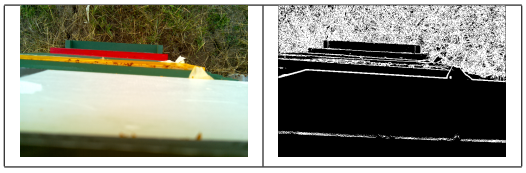

Figure 11: Unit Test 11: input image (left); output image (right).

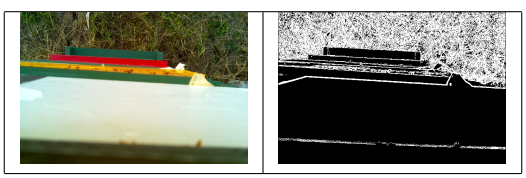

Figure 12: Unit Test 12: input image (left); output image (right).

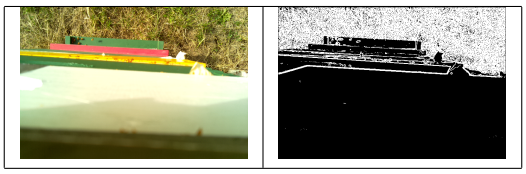

Figure 13: Unit Test 13: input image (left); output image (right).

Figure 14: Unit Test 14: input image (left); output image (right).

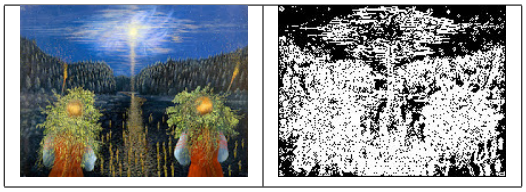

Figure 15: Unit Test 15: input image (left); output image (right).

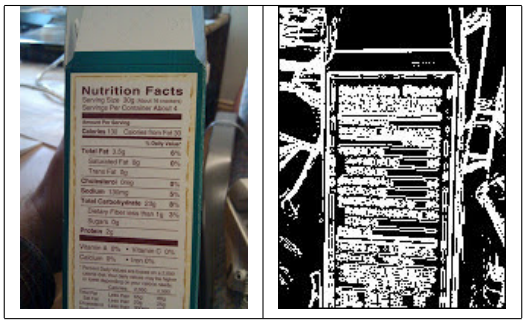

Figure 16: Unit Test 16: input image (left); output image (right).

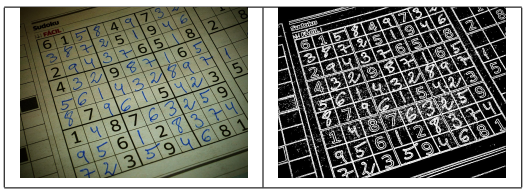

Figure 17: Unit Test 17: input image (left); output image (right).

Figure 18: Unit Test 18: input image (left); output image (right).

Figure 19: Unit Test 19: input image (left); output image (right).

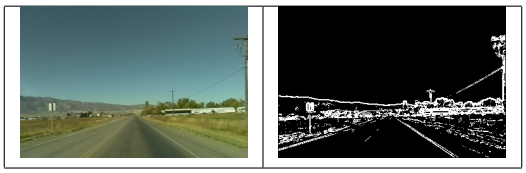

Figure 20: Unit Test 20: input image (left); output image (right).