Image Processing Homework 2 Report

Andres Ponce, 0616110

2020-11-27

Introduction

Image segmentation involves assigning each pixel (x, y) a location in a region of the image. There are many techniques useful for solving this issue. For example, we could measure the rate of change between two pixels and apply a threshold to determine if there is an edge line that might go through that pixel.

Some important functions that we could use to find different lines and areas in an image are the first derivative, second derivative, and the gradient. As mentioned, all of these techniques measure how much one region changes. We need to apply a mask to all of the pixels in a given area and then take the sum of each pixel multipleied by a certain weight. For example, the mask could be expressed as

$\sum_{d=1}^{D} w_{i,j} m_{i,j}$

for all the pixels d in an area D.

Assignment

For this assignment, we took the first and second derivative of different images to check for their different sections. The different segments of the image will show up clearly marked in a different color. This presented some questions, for example, in the textbook and online resources, most of the examples used a greyascale image, but how would it carry over to an RGB image? We could use a library such as OpenCV to easily convert the image to greyscale, or we could just apply the same mask operation on the three channels. I tried both approaches. In the end, I think that the images were clearly marked into segments.

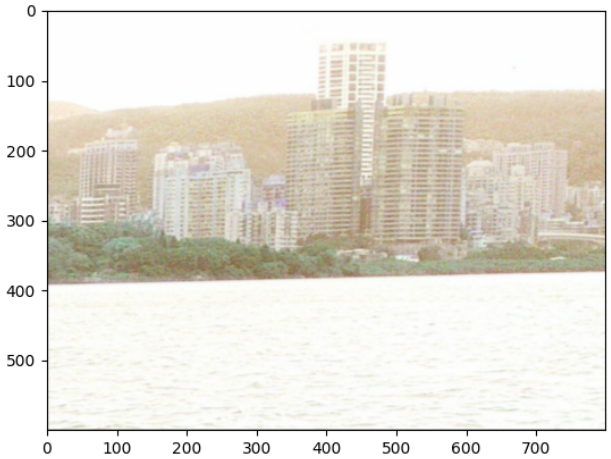

Figure 1: Image 1 after taking first derivative

In the normal image, the sky is a bit of a different color. When we take the first derivative, we can find how the image values change across the image. This is similar to taking the gradient at the image, where for the gradient we use

$f(x, y) = (\partial x, \partial y)$

To find some of the more fine detail in an image, we can use the second derivative. The second derivative of an image will bring out finer details in the image. Following is an example of an image using the second derivative.

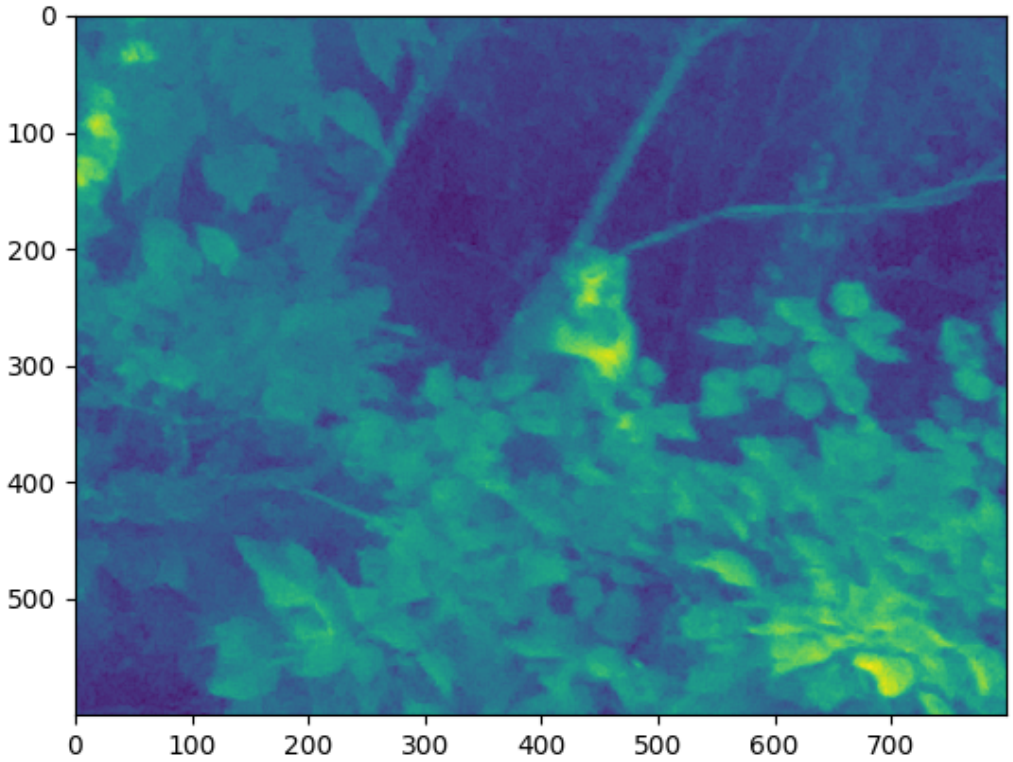

Figure 2: Second derivative of a different image with an averaging filter pre-applied

In this image we can see some of the leaves marked with a different color than the rest of the body. This is our clue that this area, along with the animal's belly, can be separated from the majority of the rest of the image.

Screenshots

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

from edgedetector import EdgeDetector

first_derivative_mask =  [[-1, 0, 1],
                          [-1, 0, 1],
                          [-1, 0, 1]]
second_derivative_mask = [[0, 1, 0],
                          [1, -4, 1],
                          [0, 1, 0]]

def main():
    name  = 'img/p1im3.png'
    image = cv2.imread(name)
    print(type(image))
    detector = EdgeDetector()
    image1 = detector.avg_filter(image)
    image1 = detector.apply_rgb_mask(image, first_derivative_mask)
    #image2 = detector.apply_rgb_mask(image, second_derivative_mask)
    #fig = plt.figure(figsize=(8,8))

    plt.imshow(image1)
    #plt.imshow(image2)
    plt.savefig('img/p1im3_result.png')
    plt.show()
if __name__=='__main__':
    main()
```

Figure 3: Main program

```python
import numpy as np
import math
import cv2
# Quick check if the pixel is OOB
def is_valid(row, col, max_row, max_col):

    if(row < 0 or col < 0 or col >= max_col or row >= max_row):
        return False

    return True


class EdgeDetector:
    def __init__(self):
        pass
    # Check if a pixel location is out of bounds
        # Given a mask, apply that mask on the given pixel as center
    def get_greyscale(self, image):
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        return image

    def apply_mask_3x3(self, image, mask, row, col, factor=1):
        max_row, max_col = image.shape[0], image.shape[1]

        sum_r, sum_g, sum_b = 0, 0, 0

        for row_offset in [-1, 0 ,1]:
            for col_offset in [-1, 0, 1]:
                new_row = row + row_offset
                new_col = col + col_offset

                # Skip if pixel is out of bounds
                if not is_valid(new_row, new_col, max_row, max_col):
                    continue

                sum_r += (image[new_row][new_col][0] * mask[row_offset+1][col_offset+1])
                sum_g += (image[new_row][new_col][1] * mask[row_offset+1][col_offset+1])
                sum_b += (image[new_row][new_col][2] * mask[row_offset+1][col_offset+1])

        return np.array([sum_r/ factor, sum_g / factor, sum_b / factor])

```

Figure 4: Edge Detector class construction with many of our functionality

```
def apply_mask_greyscale(self, image, mask, row, col, factor=1):
    mask_row, mask_col = len(mask[0]), len(mask)
    rows, cols = image.shape[0], image.shape[1]
    sum = 0
    row_offset, col_offset = math.floor(mask_row / 2), math.floor(mask_col / 2)
    currmask_x, currmask_y = 0, 0

    for x in range(row - row_offset, row_offset + 1):
        currmask_x = (currmask_x + 1) % mask_row
        for y in range(col - col_offset, col_offset + 1):
            sum += (image[x][y] * mask[currmask_x][currmask_y])
            currmask_y = (currmask_y +1 ) % mask_col
    return (sum * factor)

# Get the average of the pixel values in the neighborhood
def avg_filter(self, image):
    mask = np.array([[1, 1, 1],
                     [1, 1, 1],
                     [1, 1, 1]])
    #tmp = self.apply_mask(image, mask , 0, 0)

    for row in range(len(image)):
        for col in range(len(image[row])):
           tmp = self.apply_mask_3x3(image, mask, row, col)
           tmp[0] /= 9
           tmp[1] /= 9
           tmp[2] /= 9
           image[row][col] = tmp

    return image

# Call the corresponding filter function
def apply_filter(self, filter_name, image):
    filter_name = filter_name.lower()

    if filter_name == 'avg':
         return self.avg_filter(image)

    print('The filter did not match any. Please try again.')
    return None
```

Figure 5: Functions for applying masks to different kinds of images.

```
#
def apply_rgb_mask(self, image, mask):
    max_rows = image.shape[0]
    max_cols = image.shape[1]

    image2 = image.copy()
    print(image.shape)
    for row in range(image.shape[0]):
        for col in range(image.shape[1]):

            tmp = self.apply_mask_3x3(image, mask, row, col, factor=3)
            image2[row][col] = tmp

    return image
# Make a copy of the image and return the copy. Otherwise the image will just be dark
def apply_greyscale_mask(self, image, mask):
    max_rows = image.shape[0]
    max_cols = image.shape[1]

    image2 = image.copy()
    print(image.shape)
    for row in range(image.shape[0]):
        for col in range(image.shape[1]):

            tmp = self.apply_mask_greyscale(image, mask, row, col, factor=(1/3))
            image2[row][col] = tmp

    return image
```

Figure 6: Final filters applied