CS559 Assignment No.3

Colin Casazza

Due: October 22, 2018

Written Questions

---

(1) Let f(x,y) be an image. Let h(x,y) be the image obtained by applying the a 3 by 3 spatial low pass mask (averaging filter) to f(x,y). Similarly let g(x,y) be the
image obtained by applying a 3 by 3 spatial high pass mask to f(x,y).

```
a)      Prove that g(x,y) = f(x,y) - h(x,y)


Mlp(x,y) = { 1 for r <= r0, 0 for r > r0 } where r = sqrt(
x^2 + y^2)
Mhp(x,y) = { 1 for r > r0, 0 for r <= r0 } where r = sqrt(
x^2 + y^2)


We can see that in these two equations the only difference
 is that the filter is applied at inverse times, over the
same radius, meaning that for any index in the image, the
resulting low mask will be applied everywhere the high pas
s is not applied.


We can also see that the left side of the image shows us s
```

```
omething specific, the difference between the orignal imag
e and the resulting low pass masked image. The result of f
(x,y) - h(x,y) is the inverse of kernel for a low pass fil
ter. We already know that a high pass filter is the invers
e of a low pass filter hence f(x,y) - h(x,y) is a high pas
s filter.
```

```
b) Is the high pass mask separable? What is the implicatio
n of separability on
computations?

Yes, implicitly when the process of applying a mask requir
es n^2 operations per pixel, the process can be reduced to
 applying 2 1D operations.
```

---

(2) Suppose that the image gray level values under a mask are

```
        3   2   1
        7   8   4
        3   6   5
```

(a) Determine the value of the corresponding pixel in the output image for:

(b) In each case comment on the suitability of the filter for reducing Gaussian noise, and provide reasoning for your comments.

```
    - Median
          sorted_mask = sort(mask) => [1,2,3,3,4,5,6,7,8]
          pixel_output = median(sorted_map) => 4
          pixel_output => 4

    - Harmonic mean
         harmonic_mask = (mask) => {
             numerator = mask.length
             denominator = mask.reduce((pixel, total) => (t
otal + 1/pixel))
             return numerator / denominator
         }
         harmonic_mask([1,2,3,3,4,5,6,7,8]) => 2.949668....
         pixel_output = round(harmonic_mask([1,2,3,3,4,5,6,
7,8])) => 3

Both of these masks are subsets of nonlinear filtering met
hods,  which are generally all
viewed as methods for reducing noise, while preserving imp
ortant features like edges.

The case in which you might want to use a median filter wo
uld be when your image has distinct salt and peppered nois
e, as a harmonic mean filter does a worse job at eliminati
ng
these larger patches of corruption.
```

```
However, if there is no salt and peppered noise, a harmoni
c filter will do a better job at reducing gaussian noise a
nd outliar pixels, while preserving important features lik
e
edges.
```

---

(3) Find the output images if Sobel edge operators are applied to the following 8 by 8 input image. Note that you will have three gradient images, one in x-direction, one in y-direction and one gradient magnitude. Ignore the border effects, and produce only 6 by 6 output images.

```
            2   2   2   2   2   2   2   2
            2   2   2   2   2   2   2   7
            2   2   2   2   2   2   7   7
            2   2   2   2   2   7   7   7
            2   2   2   2   7   7   7   7
            2   2   2   7   7   7   7   7
            2   2   7   7   7   7   7   7
            2   7   7   7   7   7   7   7
```

```
def get_gx(image):
    height = len(image)
    width = len(image[0])
    x = 1
```

```
    y = 1
    new = []
    while(y<height-1):
        x = 1
        row = []
        while(x<width-1):
            row.append(
                (-1 * image[y-1][x-1]) +
                (-2 * image[y][x-1]) +
                (-1 * image[y+1][x-1]) +
                (1 * image[y-1][x+1]) +
                (2 * image[y][x+1]) +
                (1 * image[y+1][x+1]))
            x = x + 1
        y = y + 1
        new.append(row)
    return new
```

```
def get_gy(image):
    height = len(image)
    width = len(image[0])

    x = 1
    y = 1

    new = []
    while(y<height-1):
```

```
        x = 1
        row = []
        while(x<width-1):
            row.append((-1 * image[y-1][x-1]) +
                (-2 * image[y-1][x]) +
                (-1 * image[y-1][x+1]) +
                (1 * image[y+1][x-1]) +
                (2 * image[y+1][x]) +
                (1 * image[y+1][x+1]))
            x = x + 1
        y = y + 1
        new.append(row)
    return new
```

```
def combine_gx_gy(gx, gy):
    new = []
    for x in range(len(gx)):
        row = []
        for y in range(len(gx[0])):
            nx = gx[y][x]
            ny = gy[y][x]
            row.append(math.sqrt((nx*nx) + (ny*ny)))
        new.append(row)
    return new
```

```
print(get_gx(image_in))
```

```
    [[0, 0, 0, 0, 5, 15], [0, 0, 0, 5, 15, 15], [0, 0, 5,
15, 15, 5], [0, 5, 15, 15, 5, 0], [5, 15, 15, 5, 0, 0], [1
5, 15, 5, 0, 0, 0]]
```

```
print(get_gy(image_in))
```

```
    [[0, 0, 0, 0, 5, 15], [0, 0, 0, 5, 15, 15], [0, 0, 5,
15, 15, 5], [0, 5, 15, 15, 5, 0], [5, 15, 15, 5, 0, 0], [1
5, 15, 5, 0, 0, 0]]
```

```
print(combine_gx_gy(get_gx(image_in), get_gy(image_in)))
```

```
    [[0.0, 0.0, 0.0, 0.0, 7.0710678118654755, 21.213203435
596427], [0.0, 0.0, 0.0, 7.0710678118654755, 21.2132034355
96427, 21.213203435596427], [0.0, 0.0, 7.0710678118654755,
 21.213203435596427, 21.213203435596427, 7.07106781186547
5], [0.0, 7.0710678118654755, 21.213203435596427, 21.21320
3435596427, 7.0710678118654755, 0.0], [7.0710678118654755,
 21.213203435596427, 21.213203435596427, 7.07106781186547
5, 0.0, 0.0], [21.213203435596427, 21.213203435596427, 7.0
710678118654755, 0.0, 0.0, 0.0]]
```

4.

```
(a) Use the definitions of the derivatives as
    sf/sx = f(x + 1/2,y) - f(x - 1/2, y),
```

```
and similarly for sf/sy to obtain the Laplacian mask.

        0   1   0
        1   -4  1
        0   1   0

Why is Laplacian is rarely used alone?
    - Laplacian filters are very sensitive to noise, so a
gaussian filter is often applied before a laplacian filter
 to reduce the effects of noise on the laplacian filter.

(b) What will be the Laplacian mask if the derivative is d
efined as
sf/sx = f(x+1,y) - f(x -1, y), and similarly for sf/sy ? W
hy is this
definition is not suitable for obtaining the Laplacian?

        0   2   0
        2   -8  2
        0   2   0

This change would further increase the sensitivity of the
laplacian filter. This set of parameters would likely crea
te an output where the laplacian filter marked almost ever
ything as an 'edge', because the chances of encountering a
 greyscale gradient increases as
you increase the distance between the two sample points yo
u take.
```

---

5 . Compute the Fourier transform of the one-dimensional image f(0)=8, f(1)=4, f(2)=2, f(3)=1. Find Fourier spectrum |F(u)|. Comment on your results.

```
fft([8,4,2,1])    =>    15 +  0i,    6 -  3i,    5 +  0i,
   6 +  3i
```

---

6 . Answer the following questions and support your answers with reasoning and analysis

```
(a) Why is it necessary to move the origin of the Fourier
transformed image to the center (i.e. to u = n/2, v = n/2)
.  How is this shifting implemented?

The Fourier transformation of a signal operates symmetrica
lly about it's X and Y axis, so you must align the center
refrence for the fornier transformation with the center of
 your image, so the forier transformation can gather/map d
ata from both sides of the X and Y axis in the image, not
just the positive sides.

(b) Why is bit reversal needed in FFT? Explain.

Bit reversal is important for fft because it helps reduce
the storage complexity for the
```

```
actual runtime of the algorithm. During FFT, the algorithm
 generates intermediate arrays of data, the size of which
can be reduced by taking advantage of in place bit swappin
g instead of linearly mapping array components.

(c) The Fourier spectrum |F(u,v)| of an image f(x,y) is kn
own, but f(x,y) is not known. Can f(x,y) be computed? Expl
ain.

No, FFT is a lossy compression algorithm, because it retur
ns a reduced form of the input provided. The output of FFT
 is the closest possible representation of the input using
 sinusoidal functions.

(d) Prove that the two-dimensional Fourier transform of an
 image f(x,y) can be
  achieved using two one-dimensional transforms.  What is
the significance of this?

By looking at the eqation (5.14) we can see that FFT is co
mposed of two summations, the second of which is a 1-D tra
nsformation on f(x,y) along the vertical axis.  Similarly
the horizontal azis transformation can be composed by the
first summation in FFT.

By computing these dimension separately, we in some places
, compute once and share the data between the computation
for the horizontal axis and vertial axis from a lookup tab
```

```
le which can save storage space and time. The two dimensio
ns can then also be considered concurrent calculations, fu
rther saving time.
```

Programming

```
Comments and descriptions for the programming section can
be found below in the section titled 'Healper Functions'
```

```
# convert image to greyscale and open
convertToGreyScale("./inputs/flowers.jpg", "./inputs/flowers_grey.jpg")
image = Image.open("./inputs/flowers_grey.jpg")
```

```
# get both directions of the edge gradient and combine
get_gx_image(image).save("./outputs/gx_filtered.jpg")
get_gy_image(image).save("./outputs/gy_filtered.jpg")

# combine the gradients
combine_gx_gy_image(get_gx_image(image),get_gy_image(image)).save("./outputs/flower_edges.jpg")
```

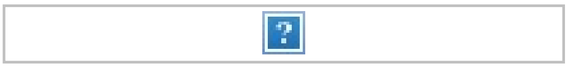
GY FILTERED

GX FILTERED

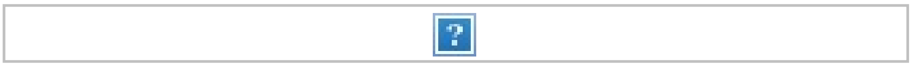
COMBINED

Salt and Pepper Noise

```
image = Image.open("./inputs/flowers.jpg")
salt_pepper_noise(image).save("./inputs/salt_pepper_flower.jpg")
convertToGreyScale("./inputs/salt_pepper_flower.jpg", "./inputs/flowers_salt_pepper_grey.jpg")
```



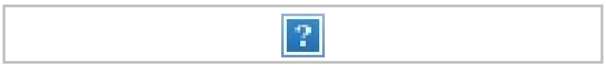
```
salt_pepper_noise = Image.open("./inputs/flowers_salt_pepper_grey.jpg")

# get edges w/o median filter
get_gx_image(salt_pepper_noise).save("./outputs/get_gx_salt_pepper_image(no-median-filter).jpg")
get_gy_image(salt_pepper_noise).save("./outputs/get_gy_salt_pepper_image(no-median-filter).jpg")
combine_gx_gy_image(get_gy_image(salt_pepper_noise),get_gy_image(salt_pepper_noise)).save("./outputs/flower_salt_pepper_edges(no-median-filter).jpg")
```



```
# get edges w/ median filter
salt_pepper_noise = salt_pepper_noise.filter(ImageFilter.MedianFilter(size=3))
salt_pepper_noise.save("./inputs/salt_pepper_flower_filter
```

```
ed.jpg")
```

salt and peppered image after median filter(n=3)



```
get_gx_image(salt_pepper_noise).save("./outputs/get_gx_salt_pepper_image.jpg")
get_gy_image(salt_pepper_noise).save("./outputs/get_gy_salt_pepper_image.jpg")
combine_gx_gy_image(get_gy_image(salt_pepper_noise),get_gy_image(salt_pepper_noise)).save("./outputs/flower_salt_pepper_edges.jpg")
```

Helper Functions

```
import math
from matplotlib import pyplot as plt
import matplotlib.image as mpimg
from PIL import Image, ImageDraw, ImageFilter
import numpy as np
import cv2
import random
import time
import datetime
```

```
# FUNCTION => this Function applies a sobel edge operator to the x domain of an image
# PARAM => image saved in a PIL image format, convert to greyscale before using this function
def get_gx_image(image):
    new_image = image.copy()
    width, height = image.size
    # ignore the first and last pixels of the image (hence x=1 and while(x<height-1)
    x = 1
    y = 1
    new = []
    # for each pixel apply the sobel edge mask
    while(y<height-1):
        x = 1
        row = []
        while(x<width-1):
            # calculate the pixel based on the surrounding pixels
            pixel = ((-1 * image.getpixel( (x-1, y-1) )) +
            (-2 * image.getpixel( (x-1, y) )) +
            (-1 * image.getpixel( (x-1, y+1) )) +
            (1 * image.getpixel( (x+1, y-1) )) +
            (2 * image.getpixel( (x+1, y) )) +
            (1 * image.getpixel( (x+1, y+1) )))
            new_image.putpixel( (x,y) , pixel)
            x = x + 1
```

```
        y = y + 1
    return new_image


# FUNCTION => this Function applies a sobel edge operator to the y domain of an image
# PARAM => image saved in a PIL image format, convert to greyscale before using this function
# operates the same as get_gx_image but applies a mask with different coefficients
def get_gy_image(image):
    new_image = image.copy()
    width, height = image.size
    x = 1
    y = 1
    new = []
    while(y<height-1):
        x = 1
        row = []
        while(x<width-1):
            pixel = ((-1 * image.getpixel( (x-1, y-1) )) +
            (-2 * image.getpixel( (x, y-1) )) +
            (-1 * image.getpixel( (x+1, y-1) )) +
            (1 * image.getpixel( (x-1, y+1) )) +
            (2 * image.getpixel( (x, y+1) )) +
            (1 * image.getpixel( (x+1, y+1) )))
            new_image.putpixel( (x,y) , pixel)
            x = x + 1
        y = y + 1
```

```
    return new_image

# FUNCTION =>  this function takes the two gradients produced by get_gx and get_gy and combines them
# PARAM => both gradients as PIL image objects in greyscale format
def combine_gx_gy_image(image_gx, image_gy):
    new = image_gx
    width, height = image_gx.size

    for y in range(height):
        for x in range(width):
            nx = image_gx.getpixel((x,y))
            ny = image_gx.getpixel((x,y))
            pixel = math.sqrt((nx*nx) + (ny*ny))
            new.putpixel( (x,y) , int(pixel) )

    return new

# takes a an image and for each pixel returns back black 20 percent of the time
def salt_pepper_noise(image):
    new = image.copy()
    width, height = image.size
    for y in range(height):
        for x in range(width):
            rand = random.randint(0,100)
            if rand >= 20:
```

```
                new.putpixel( (x,y) , image.getpixel( (x,y) ) ) )
            if(rand < 20):
                new.putpixel( (x,y) , (0,0,0) )

    return new

# self descriptive...
def convertToGreyScale(in_path, out_path):
    # open file
    img = mpimg.imread(in_path)

    # maps rgb values to rgb to greqscale formula coeificients
    gray = np.dot(img[...,:3], [0.299, 0.587, 0.114])

    # get the greyscale pixel mappings
    plt.imshow(gray, cmap = plt.get_cmap("gray"))

    # convert to true greyscale  (smaller footprint) and save image
    Image.fromarray(gray).convert("L").save(out_path)
```