Digital Image Processing
Mid-Sem Exam

By

Seelapureddy Venkata Rama Aditya Reddy
IMT2014047
Aditya.seelapureddy@iiitb.org

March 12, 2017

1 Question 1 Answer

a). The Thresholds {T1} for segmenting out regions circle, rectangle, background from Image I1 is as follows

- For circle [148, 152]
- For Rectangle [122, 128]
- For Background not of above two intervals.

b). The Thresholds {T2} for segmenting out regions circle, rectangle, background from Image I2 is as follows

- For circle [144, 154]
- For Rectangle [120, 132]
- For Background not of above two intervals.

The thresholds {T1} and {T2} suggested that I2 is more corrupted to noise than I1 as Thresholding range for segmenting out circle, rectangle is more in I2 when compared to I1. That is, The noise added in Image I2 is more than noise added in Image I1.

2 Quantitative measure on Images to unravel the order between the noises

Assumption is noise added is zero mean Gaussian Noise.
Let original Image be I. Suppose Image I1 is obtained after adding a noise of level n1. Suppose Image I2 is obtained after adding a noise of level n2. We tried to find the order between noise levels n1 and n2 by comparing variances of residual Images obtained from I, I1 and I, I2 respectively.
Another method tried was to compare PSNR (Peak Signal to Noise Ratio) of Image I1 and Image I2.
Another method without reference Image was to estimate the standard deviation using the formulae mentioned below using Noise Estimation Operator derived from Laplacian operator.

$$\sigma_n = \sqrt{\frac{\pi}{2}} \frac{1}{6(W-2)(H-2)} \sum_{imageI} |I(x, y) * N| \tag{1}$$

$I(x, y)$ is the corrupted Image, W, H are width and height of an Image and N is noise estimation operator defined as follow [1]

$$\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

Implemented code is at the end of the Document.

3 Comparing SSIM with UIQI

Let $\mathbf{x} = \{x_i | i = 1, 2, ..., N\}$ and $\mathbf{y} = \{y_i | i = 1, 2, ..., N\}$ be original and test Image signals respectively [2]. The Universal Quality Index is given by

$$Q = \frac{\sigma_{xy}}{\sigma_x \sigma_y} . \frac{2\bar{x}\bar{y}}{\bar{x}^2 + \bar{y}^2} . \frac{2\sigma_x \sigma_y}{\sigma_x^2 + \sigma_y^2} \tag{2}$$

The first component tries to represent correlation between $\mathbf{x}$ and $\mathbf{y}$, second component represents how close mean Illuminance between $\mathbf{x}$ and $\mathbf{y}$, third component represents how much close the contrasts of $\mathbf{x}$ and $\mathbf{y}$ (i.e variance of signals) [2].

The Structural Similarity Index is given by

$$SSIM(\mathbf{x}, \mathbf{y}) = [l(\mathbf{x}, \mathbf{y})]^{\alpha} . [c(\mathbf{x}, \mathbf{y})]^{\beta} . [s(\mathbf{x}, \mathbf{y})]^{\gamma} \tag{3}$$

$\alpha > 0$, $\beta > 0$, $\gamma > 0$ are parameters to adjust the importance given to these three components [3].

where $l(\mathbf{x}, \mathbf{y})$ (luminance comparison function) is given by

$$l(\mathbf{x}, \mathbf{y}) = \frac{2\mu_x \mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{4}$$

Here C_1 avoids instability when $\mu_x^2 + \mu_y^2$ is very close to zero [3].

$c(\mathbf{x}, \mathbf{y})$ (contrast comparision function) is given by

$$c(\mathbf{x}, \mathbf{y}) = \frac{2\sigma_x \sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{5}$$

Here C_2 avoids instability when $\sigma_x^2 + \sigma_y^2$ is very close to zero [3].

$s(\mathbf{x}, \mathbf{y})$ is given by

$$s(\mathbf{x}, \mathbf{y}) = \frac{\sigma_{xy} + C_3}{\sigma_x \sigma_y + C_3} \tag{6}$$

Here C_3 avoids instability when $\sigma_x \sigma_y$ is very close to zero.

Constants $C_n = (K_n L)$, n=1,2,3. L is the Dynamic range of pixel values, $K_n << 1$.

3.1 scenario where SSIM results in the same numbers as UIQI

From the equations (2), (3), (4), (5) when $\alpha = \beta = \gamma = 1$ and $C_1 = C_2 = C_3 = 0$ then SSIM reduces to UIQI [3].

3.2 Scenario where SSIM results in a more reliable index compared to UIQI

At nearly flat regions, the denominator of the contrast comparison formula is close to zero in UIQI, which makes the algorithm unstable. So, SSIM results are more reliable as the mentioned problem donot occur because of presence of constants [3].

4 Detailed note on "Bilateral filtering"

4.1 Introduction

Suppose, If we consider Gaussian Blurring / Gaussian Convolution, we do nothing but weighted averaging. Here, we give more weight to the central pixels and less weights to the neighbors. The farther away the neighbors, the smaller the weight. An Image filtered by gaussian is given by

$$GF[I]_p = \sum_{q \in S} G_\sigma(\parallel \mathbf{p} - \mathbf{q} \parallel) I_q \tag{7}$$

I is the Original Image. $\parallel \mathbf{p} - \mathbf{q} \parallel$ is the Euclidean distance between pixel locations $\mathbf{p}$ and $\mathbf{q}$. S is the set of all possible image locations that we name the spatial domain. $G_\sigma(x)$ is the 2D Gaussian Kernel [4].

$$G_\sigma(x) = \frac{1}{2\pi\sigma^2} e^{\frac{-x^2}{2\sigma^2}} \tag{8}$$

The weight for pixel $\mathbf{q}$ is defined by $G_\sigma(\parallel \mathbf{p} - \mathbf{q} \parallel)$ that is, the weight for pixel at location $\mathbf{q}$ depends on it's euclidean distance from $\mathbf{p}$ not it's Intensity value, σ defines the neighbourhood size. So, there will be blurred edges as pixels accross discontinuities are averaged together [4].

The effect of Gausian Filter is independent of Image Content, that is, the influence of a pixel over other pixel depends only on their euclidean distance but not on their actual Intensity values [4].

4.2 Bilateral filtering

The Bilateral filtering is very similar to Gaussian filtering but the main difference is that bilateral filtering considers difference in Intensities values with neighbours to preserve edges while smoothing. In Bilateral filtering, the main Idea is that for a pixel to influence another pixel it should not only have nearby locations but also similar Intensity values [4]. The Bilateral filter at location $\mathbf{p}$ is given by

$$BF[I]_p = \frac{1}{W_\mathbf{p}} \sum_{q \in S} G_{\sigma_s}(\parallel \mathbf{p} - \mathbf{q} \parallel) G_{\sigma_r}(|I_\mathbf{p} - I_\mathbf{q}|) I_q \tag{9}$$

where $W_{\mathbf{p}}$ is the Normalization factor given by

$$W_{\mathbf{p}} = \sum_{q \in S} G_{\sigma_s}(\| \mathbf{p} - \mathbf{q} \|) G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|) \tag{10}$$

G_{σ_s} is a spatial Gaussian that decreases the influence of pixels that are far, G_{σ_r} is a range Gaussian that decreases the influence of pixels $\mathbf{q}$ when their Intensities differ from $I_{\mathbf{p}}$, $I_{\mathbf{p}}$ is the Intensity value of the pixel at location $\mathbf{p}$ [4].

The Bilateral filter is controlled by parameters σ_s and σ_r. As the σ_r increases the Bilateral filter approximates to Gaussian filter. As σ_s increases it smoothens larger features [4].

4.3 Applications

There are many applications with Bilateral filtering, we will be discussing a few of them.

4.3.1 Flash / No-flash Imaging

The main idea is that illumination slowly varies over the image and is not affected by bilateral filtering where as noise mostly disappears after bilateral filtering. Therefore bilateral filtering both the flash and no-flash images extracts illumination components while the residuals contain part of the image structure. A visually pleasing image is obtained by adding the filtered no-flash image with the flash residual to combine the desired illumination with the high-quality structure [4].

4.3.2 Tone Mapping

Tone mapping tries to compress the intensity values of an high-dynamic range image to low-dynamic range display. Many Important details are lost because of intensity compression. The suggested solution is to isolate the details before compressing the intensity. The solution is to apply the bilateral filter on the log-intensities of the HDR image, scale down uniformly the result, and add back the filter residual, thereby ensuring that the fine details have not been compressed [4].

References

[1] J. Immerkær *Fast Noise Variance Estimation* 1996.

[2] Zhou Wang, Alan C. Bovik *A Universal Image Quality Index* 2002.

[3] Zhou Wang, Alan Conrad Bovik, Hamid Rahim Sheikh *Image Quality Assessment: From Error Visibility to Structural Similarity* 2004.

[4] Sylvain Paris, Pierre Kornprobst, Jack Tumblin and Frédo Durand *Bilateral Filtering: Theory and Applications* 2009.

Code for Sections 1, 2

Listing 1: Code for Section 1

```python
# Create an image of size 100 x 100, of intensity 0. Draw a square of size 40 x 40
# at the bottom of the image, of intensity 125.
# Add a circle of radius 10 units on top of the square, of intensity 150.
# The resulting image is "I".

# http://docs.opencv.org/3.1.0/dc/da5/tutorial_py_drawing_functions.html

import numpy as np
import cv2
import math
from matplotlib import pyplot as plt

img = np.zeros((100,100,1), np.uint8)
cv2.rectangle(img,(35,55),(75,95),125,1)
cv2.circle(img,(55,44),10,150,1)
cv2.imwrite("/home/aditya/diptemp/tkh/imgs/created_image.jpeg",img)

# http://stackoverflow.com/questions/22937589/how-to-add-noise-gaussian-salt-and-pepper-etc-to-image-in-python-with-opencv

# Adding Gaussian noise of variance 0.5 to the entire image.

# Size of Image
(row,column,dim) = img.shape
mean = 0
variance = 0.5
standard_deviation = variance ** (0.5)
Noisy_Image1 = np.random.normal(mean,standard_deviation,(row,column,dim))

plt.hist(Noisy_Image1.ravel(),256,[-256,256]); plt.show()

Noisy_Image1 = img + Noisy_Image1

plt.hist(Noisy_Image1.ravel(),256,[-256,256]); plt.show()
```

```

# Extracting Circle

temp1 = np.copy(Noisy_Image1)
temp2 = np.copy(Noisy_Image1)

for h in range(row):
    for w in range(column):
        if (temp1[h,w,0] < 148 or temp1[h,w,0] > 152): # 148 to 152
            temp1[h][w][0] = 0

cv2.imwrite("/home/aditya/diptemp/tkh/imgs/
    created_image_circle_extract.jpeg",temp1)

# Extracting Rectangle

for h in range(row):
    for w in range(column):
        if (temp2[h,w,0] < 122 or temp2[h,w,0] > 128): # 122 to 128
            temp2[h][w][0] = 0

cv2.imwrite("/home/aditya/diptemp/tkh/imgs/
    created_image_rectangle_extract.jpeg",temp2)

# Adding Gaussian noise of variance 1.5 to the entire image.

# Size of Image
(row,column,dim) = img.shape
mean = 0
variance = 1.5
standard_deviation = variance ** (0.5)
Noisy_Image2 = np.random.normal(mean,standard_deviation,(row,column
    ,dim))

plt.hist(Noisy_Image2.ravel(),256,[-256,256]); plt.show()

Noisy_Image2 = img + Noisy_Image2

plt.hist(Noisy_Image2.ravel(),256,[-256,256]); plt.show()

# Extracting Circle

temp1 = np.copy(Noisy_Image2)
temp2 = np.copy(Noisy_Image2)

for h in range(row):
    for w in range(column):
        if (temp1[h,w,0] < 144 or temp1[h,w,0] > 154): # 146 to
    154, 144 to 154
            temp1[h][w][0] = 0

cv2.imwrite("/home/aditya/diptemp/tkh/imgs/
    created_image_circle_extract_G_Noise_1.5_var.jpeg",temp1)

# Extracting Rectangle

for h in range(row):
```

```
    for w in range(column):
        if (temp2[h,w,0] < 120 or temp2[h,w,0] > 132): # 122 to 130, 120 to 128, 120 to 130, 120 to 132
            temp2[h][w][0] = 0

cv2.imwrite("/home/aditya/diptemp/tkh/imgs/created_image_rectangle_extract_G_Noise_1.5_var.jpeg",temp2)
```

Listing 2: Code for Section 2

```
# Reference Links
# http://dsp.stackexchange.com/questions/11326/difference-between-snr-and-psnr
# http://stackoverflow.com/questions/21117415/finding-the-value-of-the-min-and-max-pixel
# http://stackoverflow.com/questions/2440504/noise-estimation-noise-measurement-in-image

import numpy as np
import cv2
import math
from scipy import signal

img = cv2.imread("/home/aditya/diptemp/tkh/imgs/cameraman.png")
img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

# Adding Noise of power 0.5 = n1
(row,column) = img.shape
mean = 0
variance = 0.5
standard_deviation = variance ** (0.5)

Noisy_Image1 =  np.random.normal(mean,standard_deviation,(row,column))
Noisy_Image1 = img + Noisy_Image1

# Adding Noise of power 1 = n2
(row,column) = img.shape
mean = 0
variance = 1
standard_deviation = variance ** (0.5)


Noisy_Image2 = np.random.normal(mean,standard_deviation,(row,column))

Noisy_Image2 = img + Noisy_Image2

#Distinguishing between between Noise levels using variance

Noise1 = Noisy_Image1 - img
var1 = np.var(Noise1)

Noise2 = Noisy_Image2 - img
var2 = np.var(Noise2)


Difference = var1 - var2
```

```
# print Difference

if (Difference > 0):
    print "n1 > n2"
else:
    if (Difference < 0):
        print "n1 < n2"
    else:
        print "n1 = n2"

#Distinguishing between between Noise levels using PSNR (Peak Signal to Noise Ratio)

MAX_INTENSITY = np.amax(img)

MSE1 = np.sum((img.astype("float") - Noisy_Image1.astype("float")) ** 2)
MSE1 /= float(img.shape[0] * img.shape[1])

MSE2 = np.sum((img.astype("float") - Noisy_Image2.astype("float")) ** 2)
MSE2 /= float(img.shape[0] * img.shape[1])

PSNR1 = ((MAX_INTENSITY)**2)/MSE1
PSNR2 = ((MAX_INTENSITY)**2)/MSE2

if (PSNR1 < PSNR2):
    print "n1 > n2"
else:
    if (PSNR1 > PSNR2):
        print "n1 < n2"
    else:
        print "n1 = n2"

# Distinguishing between between Noise levels with the help of Laplacian operator, Assumption is noise added is zero mean Gaussian Noise

(H, W) = Noisy_Image1.shape
M = [[1, -2, 1],[-2, 4, -2],[1, -2, 1]]
sigma1 = np.sum(np.sum(np.absolute(signal.convolve2d(Noisy_Image1, M))))
sigma1 = sigma1 * math.sqrt(0.5 * math.pi) / (6 * (W-2) * (H-2))

sigma2 = np.sum(np.sum(np.absolute(signal.convolve2d(Noisy_Image2, M))))
sigma2 = sigma2 * math.sqrt(0.5 * math.pi) / (6 * (W-2) * (H-2))

if (sigma1 > sigma2):
    print "n1 > n2"
else:
    if (sigma1 < sigma2):
        print "n1 < n2"
    else:
        print "n1 = n2"
```