CPE645:Final Project Report

Realize of Image Enhancement Tools

Shenhao Chen(schen34@stevens.edu)
Yan Wu(ywu39@stevens.edu)
Yunfan Zhu(yzhu20@stevens.edu)

Project Goal

The aim of image enhancement is to improve the interpretability or perception of information in images for human viewers, or to provide `better' input for other automated image processing techniques. The goal of this project will primarily perform four tasks: –increase the contrast between certain intensity values, realize the method of contrast adjustment using the image's histogram, remove small isolated noise and unwanted details and enhance the details and edges of an image

Abstract

Many techniques of contrast enhancement have been listed in literature but most of them manipulate the spatial domain pixel values. Spatial domain techniques include point to point mapping of pixel values between the original and the enhanced images. Enhancement methods for

contrast improvement range from simple contrast stretch techniques to filtering and image transforms. The most common technique of point processing includes arithmetic operations, intensity transformations, and histogram equalization. The common techniques of area processing and filtering includes: smoothing and noise reduction, sharpening and edge enhancement.

Main Process

1. Intensity Transformations(Log transformation)
2. Histogram Modifications
3. Sharpening(Laplacian Operators)
4. Smoothing(Median Filter)

Log Transformation

Log transformation is used to enhance pixel intensities that are otherwise missed due to a wide range of intensity values or lost at the expense of high intensity values. If the intensities in the image range from [0,L-1] then the log transformation at (i, j) is given by $t(i,j) = klog(1 + I(i,j))$ where $k = \frac{L-1}{\log(1+|I_{\max}|)}$ and $I_{\max}$ is maximum magnitude value and I(i, j) is the intensity value of the pixel in the input image at (i, j). If both I(i, j) and are equal to L-1 then t(i, j) = L-1. When I(i, j) = 0, since log(1) = 0 will give t(i, j) = 0. While the end points of the range get mapped to themselves, other input values will be transformed by the above equation. The log can be of any base; however, common log (log base 10) or natural log (log base e) are widely used. The inverse of the above log transformation when the base is e is given by $t^{-1}(x) = e^{\frac{x}{k}} - 1$, which does the opposite of the log transformation.

Similar to the power law transformation with γ < 1 the log transformation also maps a small range of dark or low intensity pixel values in the input image to a wide range of intensities in the output image, while a wide range of bright or high intensity pixel values in the input image get mapped to narrow range of high intensities in the output image. Considering the intensity range is between [0,1].

The python code for log transformation is given below

```
import scipy.misc
import numpy, math
from scipy.misc.pilutil import Image

#opening the image and converting it to grayscale
a = Image.open('intensity transform.jpg').convert('L')
a.save('input.jpg')

# a is converted to an ndarray
b = scipy.misc.fromimage(a)

# b is converted to type float
b1 = b.astype(float)

# maximum value in b1 is determined
b2 = numpy.max(b1)

# performing the log transformation
c = (255.0*numpy.log(1+b1))/numpy.log(1+b2)

# c is converted to type int
c1 = c.astype(int)

# c1 is converted from ndarray to Image
d = scipy.misc.toimage(c1)

#saving d as logtransform_output.png
d.save('output.jpg')
```

Input image

After Log Transformation

Histogram Modifications

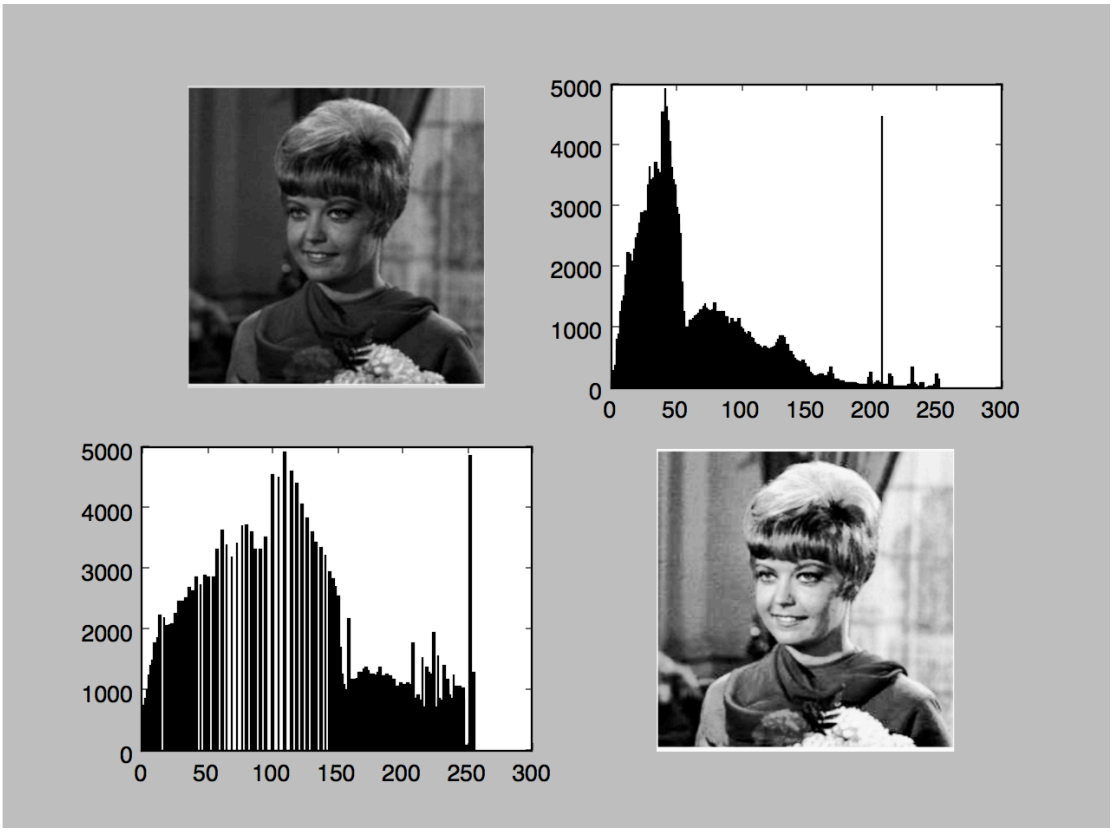
A histogram is a mapping function that gives the relationship between an amplitude value and the number of occurrences of that value in the image. Histogram provides a global description of the appearance of an image, especially the brightness and the contrast. It can be considered as the probability of a certain amplitude value: $p(r_k)=n(r_k)/N$. Where r_k is one of the amplitude values, $n(r_k)$ is the number of occurrences of r_k, and N is the total number of samples (or pixels).

The python code for Histogram Modifications is given below

```
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np


#this function is for histogram equalization
def histeq(image_array,image_bins=256):


    image_array2,bins = np.histogram(image_array.flatten(),image_bins)
    #print image_array2,bins
    # Calculate the cumulative histogram function
    cdf = image_array2.cumsum()

    # The cumulative function was transformed into the interval [ 0,255 ]
    cdf = (255.0/cdf[-1])*cdf

    # Original image matrix using integrated conversion function , interpolation process
    image2_array = np.interp(image_array.flatten(),bins[:-1],cdf)
```

```
    # Returns the image matrix leveled and cumulative function
    return image2_array.reshape(image_array.shape),cdf


#open the image and convert it to grayscale
image = Image.open("image.jpg").convert("L")

#Object into an image matrix
image_array = np.array(image)

#print grayscale image and its histogram
plt.subplot(2,2,1)
plt.imshow(image,cmap=cm.gray)
plt.axis("off")
plt.subplot(2,2,2)
plt.hist(image_array.flatten(),256) #flatten:Matrix can be converted into one-dimensional sequence

a = histeq(image_array)    # histogram equalization
plt.subplot(2,2,3)
plt.hist(a[0].flatten(),256)
plt.subplot(2,2,4)
plt.imshow(Image.fromarray(a[0]),cmap=cm.gray)
plt.axis("off")

plt.show()
```

Input image


5000
4000
3000
2000
1000
0
0 50 100 150 200 250 300
5000
4000
3000
2000
1000
0
0 50 100 150 200 250 300

Result

Laplacian Operators

The first and second order derivates can be used as sharpening operators:

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

$$g(i,j) = f(i+1,j) + f(i-1,j) + f(i,j+1) + f(i,j-1) - 4f(i,j)$$

Then we can get the matrix as follow:

$$H_1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

And we can also get another form:

$$H_2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

If X is the original image: Y = Laplacian(X) + X

Laplacian(·) is a Laplacian operator, that is a High-pass filter: Y is the sharpened image. This expression can be simplified ass the:

Y = Composite_Laplacian(X)

Where the Composite Laplacian integrates the Laplacian operator with the simple summation operator. Then the composite Laplacian Operator is:

$$H_1 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \text{Or} \quad H_2 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

We use H2 as our operator.

The python code for laplacian operator is given below

```
import cv
import cv2

#function for LaplacianSharp
def LFilter(image,array):
    w = image.width
    h = image.height
    size = (w,h)
    iFilter = cv.CreateImage(size,8,1)
    for i in range(h):
        for j in range(w):
            if i in [0,h-1] or j in [0,w-1]:
                iFilter[i,j] = image[i,j]
            else:
                a= [0]*9
                for k in range(3):
                    for l in range(3):
                        a[k*3+l] = image[i-1+k,j-1+l]
                sum = 0
                for m in range(9):
                    sum = sum+array[m]*a[m]
                iFilter[i,j] = int(sum)
    return iFilter



#load image
image = cv.LoadImage('sharp.jpg',0)
cv.ShowImage('Original',image)
#show aftersharping image
H1 = [-1,-1,-1,-1,9,-1,-1,-1,-1]
iH1F = LFilter(image,H1)
cv.ShowImage('AfterSharp',iH1F)
cv.WaitKey(0)
```

Input image

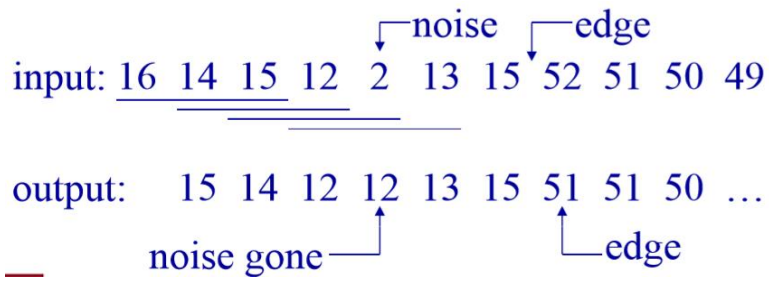
After Laplacian Sharpening

Median Filter

Problem with low-pass filtering: blurring edges and other sharp details.

Median filtering: amplitude value of each pixel is replaced by the median of the amplitude values within the M×M mask. Median filtering is a non-linear operation.

Example: a 1-D median filter with length of 3



Median Filter is fully defined by its window. A 2-D separable median filter can be formed by applying 1-D median filters to the rows and then to the resulting columns. A 2-D non-separable median filter can also be formed by defining a 2-D non-separable window.

The typical window sizes are 3×3, 5×5 etc. The larger the window is, the more sever the smoothing effect will be. Median Filters are suitable for "salt and pepper" noise in gray images and speckle noise in binary images. Median filters are a subset of the general rank order filters.

A rank order filter is defined by its window and the rank of output. Assume a rank order filter has a window with size of N. The M^{th}-rank

filter (M≤N) associated with this window selects the M^{th} smallest value as the output.

e.g. : First-rank filter –minimum value output –min filter, N^{th} rank filter-maximum value output-max filter

The python code for Salt and Pepper is given below

```
import cv2
from numpy import *

def SaltAndPepper(src,percetage):
    NoiseImg=src
    NoiseNum=int(percetage*src.shape[0]*src.shape[1])
    for i in range(NoiseNum):
        randX=random.random_integers(0,src.shape[0]-1)
        randY=random.random_integers(0,src.shape[1]-1)
        if random.random_integers(0,1)==0:
            NoiseImg[randX,randY]=0
        else:
            NoiseImg[randX,randY]=255
    return NoiseImg


img=cv2.imread('original image.jpg',flags=0)

# add SaltAndPepp
# add 10%,30%,50% SaltAndPepper respectively
Pers=[0.1,0.3,0.5]
for i in Pers:
    NoiseImg=SaltAndPepper(img,i)
    fileName='SaltPepper'+str(i)+'.jpg'
    cv2.imwrite(fileName,NoiseImg,[cv2.IMWRITE_JPEG_QUALITY,100])
    cv2.imshow('AfterNoise',NoiseImg)
    cv2.waitKey(0)
```

Original

After Salt and Pepper(10%)

After Salt and Pepper(30%)

After Salt and Pepper(50%)

Then we do the smoothing for these distorted pictures.

The python code for Smoothing is given below

```
import cv

#function for Smoothing
def MedianFilter(image):
    w = image.width
    h = image.height
    size = (w,h)
    iMFilter = cv.CreateImage(size,8,1)
    for i in range(h):
        for j in range(w):
            if i in [0,h-1] or j in [0,w-1]:
                iMFilter[i,j] = image[i,j]
            else:
                a= [0]*9
                for k in range(3):
                    for l in range(3):
                        a[k*3+l] = image[i-1+k,j-1+l]
                a.sort()
                iMFilter[i,j] = a[4]
    return iMFilter

#load image
image_name=raw_input('Please input the image name:')+'.jpg'
image = cv.LoadImage(image_name,0)
cv.ShowImage('Original',image)

#show aftersmoothing image
iMF = MedianFilter(image)
cv.ShowImage('AfterSmoothing',iMF)
cv.WaitKey(0)
```

After Smoothing(input image has 10% salt and pepper noise)

After Smoothing(input image has 30% salt and pepper noise)

After Smoothing(input image has 50% salt and pepper noise)

This median filter has a good effect, but is not so good to restore the picture. If we need to restore it, other technique should be applied.

Summary

Image enhancement techniques deal with accentuation or sharpening of image features, such as contrast, boundaries, edges, etc. Widely used techniques like histogram equalization, intensity transformations, sharpening and smoothing have been used in this project.