Image Processing Assignment 1

Introduction

- Goal:
  - Enhancing the supplied images using techniques of contrast adjustment, noise reduction, color correction and so on.
  - You CAN NOT use toolbox/library functions for:
    - Image resizing.
    - Intensity transformations.
    - Histogram computation.
    - Spatial filtering.
- Environment: Python 3.8.8
- Code: https://github.com/maikurufeza/NCTU-Image-Processing-2021/blob/main/Assignment/Assignment1.ipynb

Implemented method

I. Contrast adjustment
   A. Power law transform
   B. Histogram equalization
II. Noise reduction
   A. Averaging Filters
   B. Midpoint filter
   C. Adaptive Local Noise Reduction Filter
   D. Adaptive Median Filter
III. Color correction
IV. Spatial Filtering
   A. Sobel Filter (Top, Bottom, left, right)
   B. Laplacian Filter
   C. Sharpen Filter
   D. Gaussian Smoothing Filter
V. Other Order Statistics Filters

Experiments

I. Contrast adjustment

A. Power law transform

Method: $s = cr^{\gamma}$, both s and r scaled to between 0 and 1.

Experiment:

$c = 1, \gamma = 0.7$

Original image

Implemented image

$c = 1, \gamma = 2$

Original image

Implemented image

Result: If $\gamma < 1$, the output will be darker. If $\gamma > 1$, output will be brighter.

B. Histogram equalization

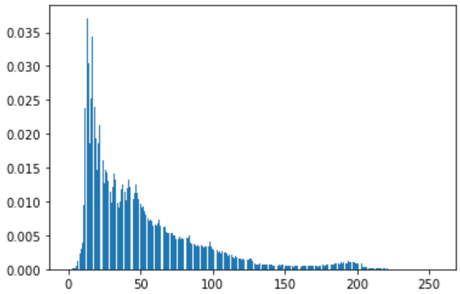
Method: $p_r(r_k) = \frac{n_k}{n}$

$$s_k = T(r_k) = (L-1)\sum_{j=1}^{k} p_r(r_j)$$

Experiment:

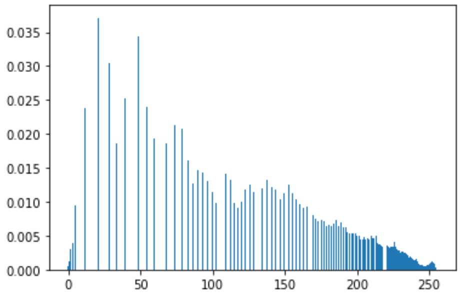
Original image

Original image histogram



Implemented image

Implemented image histogram



Result: We can see that a more uniform histogram generally corresponds to better overall contrast of the image.

II. Noise reduction

Orignal image

Gaussian noise

Impulse noise

A. Averaging Filters

Method: Convolution by the kernel $\begin{bmatrix} 0.1111 & 0.1111 & 0.1111 \\ 0.1111 & 0.1111 & 0.1111 \\ 0.1111 & 0.1111 & 0.1111 \end{bmatrix}$

Experiment:

Average Filter of Gaussian noise

Average Filter of Impulse noise

B. Adaptive Local Noise Reduction Filter

Method: $\hat{f}(x,y) = g(x,y) - \frac{\sigma_\eta^2}{\sigma_L^2}[g(x,y) - m_L]$, where

σ_η^2: estimated noise variance, σ_L^2: local variance, m_L: local mean

Experiment:

Adaptive Filter of Gaussian noise

Adaptive Filter of Impulse noise

C. Medium filter

Method: Order Statistics Filters of Medium filter

Experiment:

Medium filter of Gaussian noise

Medium filter of Impulse noise

D. Adaptive Median Filter

Method:

Level A: If $z_{\text{min}} < z_{\text{med}} < z_{\text{max}}$, go to Level B

Else, increase the size of S_{xy}

If $S_{xy} \leq S_{\text{max}}$, repeat level A

Else, output z_{med}.

Level B: If $z_{\text{min}} < z_{xy} < z_{\text{max}}$, output z_{xy}

Else output z_{med}.

Experiment:

Adaptive Median of Gaussian noise

Adaptive Median of Impulse noise

Result:

We can observe that:

1. Averaging Filters and Adaptive Local Noise Reduction Filter do well on Gaussian noise images. Moreover, Adaptive Local Noise Reduction Filter is better than Averaging Filter.
2. Medium Filter and Adaptive Median Filter do well on Impulse noise images. Moreover, Adaptive Median Filter is better than Medium Filter.

III. Color correction

Method: Power law transformation applied to individual channels can be used to apply color correction to images.

Experiment:

Orignal image

$\gamma = 0.5$ applied to Red

$\gamma = 0.5$ applied to Green

$\gamma = 0.5$ applied to Blue

Result: The image is redder (resp. blue, green) when we applid the transform to red (resp. blue, green). We do color correction to individual channels.

IV. Spatial Filtering

Method: Convolution by the given kernel.

←Original image

A. Sobel Filter (Top, Bottom, left, right)

Top Sobel Filter

```
[ 1  2  1]
[ 0  0  0]
[-1 -2 -1]
```

Bottom Sobel Filter

```
[-1 -2 -1]
[ 0  0  0]
[ 1  2  1]
```

Right Sobel Filter

```
[-1  0  1]
[-2  0  2]
[-1  0  1]
```

Left Sobel Filter

```
[ 1  0 -1]
[ 2  0 -2]
[ 1  0 -1]
```

Result: Top and Bottom Sobel Filter get more horizontal edges. Right and Left Sobel Filter get vertical edges. We can find that Sobel Filter is ‘directional’.

B. Laplacian Filter

```
[ 0 -1  0]
[-1  4 -1]
[ 0 -1  0]
```

Result: We get edges. We can find that Laplacian Filter is ‘non-directional’.

C. Sharpen Filter

Sharpen Filter:
```
[ 0 -1  0]
[-1  5 -1]
[ 0 -1  0]
```

Original image

Implemented image

Result: Edges are more clear.

D. Gaussian Smoothing Filter

Method: The filter is generated by

$$h(s,t) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{s^2+t^2}{2\sigma^2}\right)$$

Experiment:

Size:(7,7) $\sigma = 1$

```
[0.0000 0.0002 0.0011 0.0018 0.0011 0.0002 0.0000]
[0.0002 0.0029 0.0131 0.0215 0.0131 0.0029 0.0002]
[0.0011 0.0131 0.0585 0.0965 0.0585 0.0131 0.0011]
[0.0018 0.0215 0.0965 0.1592 0.0965 0.0215 0.0018]
[0.0011 0.0131 0.0585 0.0965 0.0585 0.0131 0.0011]
[0.0002 0.0029 0.0131 0.0215 0.0131 0.0029 0.0002]
[0.0000 0.0002 0.0011 0.0018 0.0011 0.0002 0.0000]
```

Size:(7,7) $\sigma = 2$

```
[0.0042 0.0078 0.0114 0.0129 0.0114 0.0078 0.0042]
[0.0078 0.0146 0.0213 0.0241 0.0213 0.0146 0.0078]
[0.0114 0.0213 0.0310 0.0351 0.0310 0.0213 0.0114]
[0.0129 0.0241 0.0351 0.0398 0.0351 0.0241 0.0129]
[0.0114 0.0213 0.0310 0.0351 0.0310 0.0213 0.0114]
[0.0078 0.0146 0.0213 0.0241 0.0213 0.0146 0.0078]
[0.0042 0.0078 0.0114 0.0129 0.0114 0.0078 0.0042]
```

Result: The bigger σ is, the more blurred image is. Contrast to Averaging Filter, Gaussian Smoothing Filter is smoother.

VI. Other Order Statistics Filters

Original image

Min Filter

Max Filter

Midpoint Filter

Result:

Min Filter: Black edge is more clear

Max Filter: Black edge is less clear

Observation and Discussion

1. What is the parameter c in Power law transform doing?

   We can fix γ and change c to see what c is doing.

Original image

Original image

$c = 1, \gamma = 2$

$c = 1, \gamma = 0.7$

$c = 2, \gamma = 2$

$c = 1, \gamma = 0.7$

We can observe that, no matter what γ is, c is to adjust brightness.

2. Is a larger Averaging filter size results in more blur?

Original image

Filter size:(3,3)

Filter size:(5,5) | Filter size:(7,7)

Yes,the larger Averaging filter size is, it results in more blur.

3. Is a larger Gaussian Smoothing filter size results in more blur?
We can fix σ and change filter size to see how blurred it is.

$\sigma = 1$, filter size = (7,7)

$\sigma = 1$, filter size = (9,9)

$\sigma = 1$, filter size = (15,15)

$\sigma = 1$, filter size = (21,21)

It seems that the bigger filter size is, the smoother image is. But the blur don't increase when the filter size increase.

Code Analysis

Power_law_transform

```
def power_law_transform(image, gamma = 1, c = 1, image_is_array = False):
    image_array = np.array(image) if not image_is_array else image
    power_law = lambda p: 255*(c*(p/255)**gamma)
    output_array = power_law(image_array)
    output_array = np.clip(output_array, 0, 255)
    output = Image.fromarray(output_array.astype('uint8')) if not image_is_array else output_array
    return output
```

$s = cr^{\gamma}$

where r is normalize to [0,1]

Histogram_equalization

```
def plotBar(x,y,title=None):
    plt.bar(x,y)
    plt.title(title)
    plt.show()

def histogram_equalization(image, plot_bar = False):
    if image.mode != 'L':
        print('Input is not gray-scale image')
        return
    else:
        image_array = np.array(image)
        height, width =image_array.shape

        counts = np.zeros(256)
        for i in range(height):
            for j in range(width):
                counts[image_array[i,j]]+=1

        pdf = counts/image_array.size
        cdf = np.cumsum(pdf)
        image_array = 255*cdf[image_array] # (L-1)*cdf

        output = Image.fromarray(image_array.astype('uint8'))

        if plot_bar:
            plotBar(range(256),pdf,title = 'origin image histograms pdf')
            plotBar(range(256),cdf,title = 'origin image histograms cdf')
            unique, counts = np.unique(image_array, return_counts=True)
            plotBar(unique,counts/image_array.size, title = 'output image histograms pdf')

    return output
```

Be used to plot image histogram

Calculate the probability density function(pdf) of the image. i.e $p_r(r_k) = \frac{n_k}{n}$

$\sum_{j=1}^{k} p_r(r_j)$

$s_k = T(r_k) = (L-1)\sum_{j=1}^{k} p_r(r_j)$

Color correction

```
def color_correction_3D(image, gamma = [1,1,1], c = [1,1,1]):
    image_array = np.array(image)
    output_array = np.zeros_like(image_array)
    height, width, channel = image_array.shape
    for i in range(channel):
        output_array[:,:,i] = power_law_transform(image_array[:,:,i], gamma[i], c[i], image_is_array = True)
    output = Image.fromarray(output_array)
    return output
```

Power law transformation applied to individual channels

Class Filter: used to store each filter's parameters and information

```
class Filter:
    def __init__(self, array = None, is_statistics = False, statistics = '', size = None):
        self.array = array                                          # kernal array
        self.is_statistics = is_statistics                          # is the kernal a statistics filter
        self.statistics = statistics                                # statistics filter name
        self.shape = self.array.shape if size == None else size     # kernal size

    def __str__(self):
        if not self.is_statistics: print(self.array)
        return str(self.shape) + ( ' Spatial Filter' if not self.is_statistics else 'Statistic Filter' )
```

Define all kind of filter

```
Identity_Filter = Filter(np.array([[0, 0, 0], [0, 1, 0], [0, 0, 0]]))
Top_Sobel_Filter = Filter(np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]]))
Left_Sobel_Filter = Filter(np.array([[1, 0, -1], [2, 0, -2], [1, 0, -1]]))
Bottom_Sobel_Filter = Filter(np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]]))
Right_Sobel_Filter = Filter(np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]))
Laplacian_Filter = Filter(np.array([[0, -1, 0], [-1, 4, -1], [0, -1, 0]]))
Sharpen_Filter = Filter(np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]]))
Outline_Filter = Filter(np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]]))

def Averaging_Filter(size = (3,3)):
    return Filter(np.full(size,1)/(size[0]*size[1]))

def Gaussian_Smooting_Filter(size = (3,3), sigma = 1):
    output = np.zeros(size)
    h = (size[0]-1)/2
    w = (size[1]-1)/2
    for i in range(size[0]):
        for j in range(size[1]):
            output[i,j] = (1/(2*math.pi*(sigma**2)))*math.exp(-((i-h)**2+(j-w)**2)/(2*sigma**2))
    return Filter(output)

def Max_Filter(size = (3,3)):
    return Filter(is_statistics = True, statistics = 'Max', size = size)

def Min_Filter(size = (3,3)):
    return Filter(is_statistics = True, statistics = 'Min', size = size)

def Mean_Filter(size = (3,3)):
    return Filter(is_statistics = True, statistics = 'Mean', size = size)

def Medium_Filter(size = (3,3)):
    return Filter(is_statistics = True, statistics = 'Medium', size = size)

def Midpoint_Filter(size = (3,3)):
    return Filter(is_statistics = True, statistics = 'Midpoint', size = size)

def Adaptive_Filter(size = (3,3)):
    return Filter(is_statistics = True, statistics = 'Adaptive', size = size)

def Adaptive_Medium_Filter(size = (3,3)):
    return Filter(is_statistics = True, statistics = 'Adaptive Medium', size = size)
```

$$h(s,t) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{s^2+t^2}{2\sigma^2}\right)$$

Add noise on the image, there are two kinds of noise: Gaussian noise, Salt and Pepper (impulse) noise

```
def add_noise(image, strength, mode):
    image_array = np.array(image)/255
    if mode == 'Gaussian':
        noise = np.random.normal(0, strength, image_array.shape)
    elif mode == 'Salt and Pepper':
        noise = np.random.choice([-2,0,2],image_array.shape[0:2], p = [strength/2, 1-strength, strength/2])
        if len(image_array.shape) == 3:
            noise = np.repeat(noise[:,:,np.newaxis], 3, axis=2)
    output_array = image_array + noise
    output_array = np.clip(output_array, 0, 1)*255
    output = Image.fromarray(output_array.astype('uint8'))
    return output
```

Convolution2D: Be used to implement all kind of filters in gray-scale images

```python
def convolution2D(image, kernal, zero_padding = True, image_is_array = False):
    if(not image_is_array and len(np.array(image).shape)!=2):
        print('this is for 2d image, your input image is ' + str(len(np.array(image).shape)) + 'd')
        return

    image_array = np.array(image) if not image_is_array else image
    if zero_padding:
        h = (kernal.shape[0]-1)//2
        w = (kernal.shape[1]-1)//2
        image_array = np.pad(image_array, ((h,h),(w,w)), 'constant')

    image_height, image_width = image_array.shape
    kernal_height, kernal_width = kernal.shape
    output_array = np.zeros((image_height-kernal_height+1, image_width-kernal_width+1)).astype(int)
    noice_var = np.var(image_array)
    for x in range(image_height-kernal_height+1):
        for y in range(image_width-kernal_width+1):
            if not kernal.is_statistics:
                output_array[x,y] = np.sum(image_array[x:x+kernal_height,y:y+kernal_width]*kernal.array)
            else:
                if kernal.statistics == 'Max':
                    output_array[x,y] = np.max(image_array[x:x+kernal_height,y:y+kernal_width])
                elif kernal.statistics == 'Min':
                    output_array[x,y] = np.min(image_array[x:x+kernal_height,y:y+kernal_width])
                elif kernal.statistics == 'Medium':
                    output_array[x,y] = np.median(image_array[x:x+kernal_height,y:y+kernal_width])
                elif kernal.statistics == 'Mean':
                    output_array[x,y] = np.mean(image_array[x:x+kernal_height,y:y+kernal_width])
                elif kernal.statistics == 'Midpoint':
                    output_array[x,y] = (np.max(image_array[x:x+kernal_height,y:y+kernal_width]) +
                                         np.min(image_array[x:x+kernal_height,y:y+kernal_width])) / 2
                elif kernal.statistics == 'Adaptive':
                    g_xy = image_array[x+(kernal_height-1)//2,y+(kernal_width-1)//2]
                    local_var = np.var(image_array[x:x+kernal_height,y:y+kernal_width])
                    local_var = max(noice_var, local_var)
                    output_array[x,y] = g_xy-\
                    (noice_var/local_var)*(g_xy-np.mean(image_array[x:x+kernal_height,y:y+kernal_width]))
                elif kernal.statistics == 'Adaptive Medium':
                    kernal_height, kernal_width = kernal.shape
                    Z_xy = image_array[x+(kernal_height-1)//2,y+(kernal_width-1)//2]
                    i = 0 #increase
                    while(x-i>=0 and x+kernal_height+i<image_height \
                          and y-i>=0 and y+kernal_width+i<image_width):
                        z_min = np.min(image_array[x-i:x+kernal_height+i,y-i:y+kernal_width+i])
                        z_max = np.max(image_array[x-i:x+kernal_height+i,y-i:y+kernal_width+i])
                        z_med = np.median(image_array[x-i:x+kernal_height+i,y-i:y+kernal_width+i])

                        if(not(z_min<z_med and z_med<z_max)):
                            i += 1
                        else: #Level b
                            if(z_min<Z_xy and Z_xy<z_max):
                                output_array[x,y] = Z_xy
                            else:
                                output_array[x,y] = z_med
                            break;
                    else:
                        output_array[x,y] = z_med
    output_array = np.clip(output_array, 0, 255)
    output = Image.fromarray(output_array.astype('uint8')) if not image_is_array else output_array
    return output
```

σ_η^2 for adaptive filter

Special filter

Max filter

Min filter

Medium filter

Midpoint filter

σ_L^2

$$g(x,y) - \frac{\sigma_\eta^2}{\sigma_L^2}[g(x,y) - m_L]$$

If $S_{xy} \leq S_{\max}$, repeat level A

If $z_{\min} < z_{\text{med}} < z_{\max}$, go to Level B
Else, increase the size of S_{xy}

Level B: If $z_{\min} < z_{xy} < z_{\max}$, output z_{xy}
Else output z_{med}.

Else, output z_{med}.