Image Processing - Programming Assignment #1

310553056 陳威齊

Introduction / Objectives

Using techniques of contrast adjustment, noise reduction, color correction to enhance images.

Methods

- Program Language：Python3.8
- Image Processing：

  Intensity Transform：

  1. Linear
  2. Log
  3. Gamma correction
  4. Piecewise-Linear
  5. Histogram equalization

  Noise Filter：

  1. Mean Filter
  2. Median Filter
  3. Alpha-Trimmed-Mean Filter
  4. Bilateral Filter

Result & Discussions

	
Origin	Gamma = 1/10
Low exposure, low contrast	over-exposure
	
Gamma = 1/0.5	Gamma = 1/2.2
under-exposure	The outline of the animal's back becomes clearer

	
Origin	Histogram Equalization
Low exposure	The house on the right and the street sign on the left are clearly visible
	
Gamma = 1/0.7	Gamma = 1/0.5
Compared with Histogram Equalization, the sky is smoother	Compared with Origin, photo tends to be red

	
Origin	Log
Low exposure, low contrast	Compared with Origin, the houses on the left and the roads on the right are much more obvious
	
Median Filter	Log + Median Filter
Compared with Log, the color of the houses on the left is lighter	Compared with only Median Filter, the sky on the left is darker

	
Origin	Piecewise-Linear
	The color of the street sign becomes lighter, and the rear street sign becomes blurred
	
Alpha-Trimmed-Mean	Piecewise-Linear + Alpha-Trimmed-Mean
Looks similar to Piecewise-Linear	

	
Origin	Linear
Low exposure, low contrast	The fence is clearly visible, and the leaves on the upper two sides are more distinct
	
Bilateral Filter	Linear + Bilateral Filter
Compared to Linear, the sky is brighter	

Discussions（Code）

- All filters are implemented with a 3x3 window. If it rises to 5x5, the overall efficiency is much worse.
- In the implementation, I found that the efficiency of the filter was very poor. Without considering python itself was slow, the execution efficiency of Bilateral was the worst. After searching on the internet, I found that someone used the integral image to speed up Bilateral.
- Instead of creating the function of commanding and running the specified image processing, all the necessary functions are added to the original code one by one.

hw01.py

```
import numpy as np
import sys
import cv2

from intensity_transform import *
from noise_reduction import *
from bilateral_filter import *

path = sys.argv[1]
img = cv2.imread(path)
row, col, channel = img.shape

dst = img.copy()

#############################################################
intensity_transform
dst = ConstrastAdjust(dst, 0.5, 50)
#dst = GammaCorrection(dst, 2.2)
#dst = LogTransform(dst)
#pixelVal_vec = np.vectorize(pixelVal)
#dst = pixelVal_vec(dst, 70, 30, 180, 230)
#dst = HistogramEqual(dst, row, col)

dst_extend = np.zeros((row + 2, col + 2, channel))

for i in range(1, row + 1):
   for j in range(1, col + 1):
       dst_extend[i][j] = dst[i-1][j-1]

#############################################################
noise_reduction
#mean_filter = GetMeanFilter()
#dst = Convolution(dst, row, col, dst_extend, mean_filter)
#dst = GetMedianFilter(dst, row, col, dst_extend)
#dst = GetAlphaTrimmedMeanFilter(dst, row, col, dst_extend, 6)

#############################################################
bilateral_filter
dst = BilateralFilter(dst, row, col, dst_extend, 1, 5)



dst = np.round(dst)
```

```
dst = dst.astype(np.uint8)

cv2.imshow('img', img)
cv2.imshow('dst', dst)

cv2.waitKey(0)

cv2.imwrite("images/test.png", dst)
cv2.destroyAllWindows()
```

intensity_transform.py

```
import numpy as np

def ConstrastAdjust(dst, contrast, brightness):
    a = contrast
    b = brightness

    dst = dst * a + b
    dst[dst > 255] = 255
    dst[dst < 0] = 0

    return dst

def GammaCorrection(dst, gamma):
    dst = (dst / 255) ** (1 / gamma)
    dst = dst * 255
    dst[dst > 255] = 255
    dst[dst < 0] = 0

    return dst

def LogTransform(dst):
    c = 255/(np.log(1 + np.max(dst)))
    dst = c * np.log(1 + dst)

    return dst

def pixelVal(dst, r1, s1, r2, s2):
    if (0 <= dst and dst <= r1):
        return (s1 / r1)*dst
    elif (r1 < dst and dst <= r2):
        return ((s2 - s1)/(r2 - r1)) * (dst - r1) + s1
    else:
        return ((255 - s2)/(255 - r2)) * (dst - r2) + s2

def HistogramEqual(dst, row, col):
    R = dst[:,:,0]
    G = dst[:,:,1]
    B = dst[:,:,2]

    R_cdf = np.zeros((256,))
    R_sum = 0
    G_cdf = np.zeros((256,))
```

```
    G_sum = 0
    B_cdf = np.zeros((256,))
    B_sum = 0
    for i in range(256):
        R_count = np.count_nonzero(R == i)
        if R_count != 0:
            R_sum = R_sum + R_count
            R_cdf[i] = R_sum

        G_count = np.count_nonzero(G == i)
        if G_count != 0:
            G_sum = G_sum + G_count
            G_cdf[i] = G_sum

        B_count = np.count_nonzero(B == i)
        if B_count != 0:
            B_sum = B_sum + B_count
            B_cdf[i] = B_sum

    R_eq = R_cdf / 480000 * 255
    G_eq = G_cdf / 480000 * 255
    B_eq = B_cdf / 480000 * 255

    for i in range(row):
        for j in range(col):
            R_color, G_color, B_color = dst[i][j]
            dst[i][j] = [R_eq[R_color], G_eq[G_color], B_eq[B_color]]

    return dst
```

noise_reduction.py

```
import numpy as np

def GetMeanFilter():
    filter = np.zeros((3, 3))
    filter = filter + 1
    filter = filter / filter.sum()

    return filter

def GetGaussianFilter(sigma):
    x, y = np.mgrid[-1:2, -1:2]
    gaussion_filter = np.exp(-(x**2 + y**2) / 2 * pow(sigma, 2))
    gaussion_filter = gaussion_filter / gaussion_filter.sum() # normalize

    return gaussion_filter

def Convolution(dst, row, col, img_extend, filter):
    filter_row, filter_col = filter.shape

    RGB = np.array([0, 0, 0])
    for m in range(row):
        for n in range(col):
            for i in range(filter_row):
                for j in range(filter_col):
                    RGB = RGB + filter[i][j] * img_extend[i + m][j + n]
            dst[m][n] = RGB
            RGB = np.array([0, 0, 0])

    return dst

def GetMedianFilter(dst, row, col, img_extend):
    for i in range(row):
        for j in range(col):
            tmp = img_extend[i:i+3, j:j+3]
            tmp = tmp.reshape(9, 3)
            median = np.median(tmp, axis=0)
            dst[i][j] = median

    return dst

def GetAlphaTrimmedMeanFilter(dst, row, col, img_extend, d):
    d2 = int(d/2)
```

```
    for i in range(row):
        for j in range(col):
            tmp = img_extend[i:i+3, j:j+3]
            tmp = tmp.reshape(9, 3)
            tmp = tmp[d2:9-d2, 0:3]
            mean = np.mean(tmp, axis=0)
            dst[i][j] = mean

    return dst
```

bilateral_filter.py

```
import numpy as np

def distance(x, y, i, j):
   return np.sqrt(((x-i)**2 + (y-j)**2))

def gaussian(x, sigma):
   return np.exp(- (x ** 2) / (2 * sigma ** 2))

def BilateralFilter(dst, row, col, img_extend, sigma_d=10, sigma_r=10):
   for x in range(row):
       for y in range(col):
           pixel_filter = np.zeros((3,))
           w_sum = np.zeros((3,))
           for i in range(x, x+3):
               for j in range(y, y+3):
                   d = distance(i, j, x+1, y+1)
                   Id = img_extend[i][j] - img_extend[x+1][y+1]
                   space_w = gaussian(d, sigma_d)
                   color_w = gaussian(Id, sigma_r)
                   w = space_w * color_w
                   pixel_filter += img_extend[i][j] * w
                   w_sum += w
           pixel_filter = pixel_filter / w_sum
           dst[x][y] = pixel_filter

   return dst
```