Image Processing – HW01

309553048_多工所_陳柏丞

- Introduction：
  Using techniques of contrast adjustment, noise reduction, color correction to enhance 6 images. As a result, you should write your processing techniques as modules so that you can try different combinations or parameters for different images.

- Methods：
  - Program Language：Python3.8
  - Image Processing：
    - Intensity Transform：
      1. Linear
      2. Log
      3. Gamma
      4. Piecewise-Linear
      5. Histogram equalization
    - Noise Filter：
      1. Mean Filter
      2. Median Filter
      3. Alpha-Trimmed-Mean Filter
      4. Bilateral Filter

- Results：

Image 1：

Origin

Gamma = 1/2.2

Gamma + Alpha-trimmed-Mean

Gamma + Bilateral

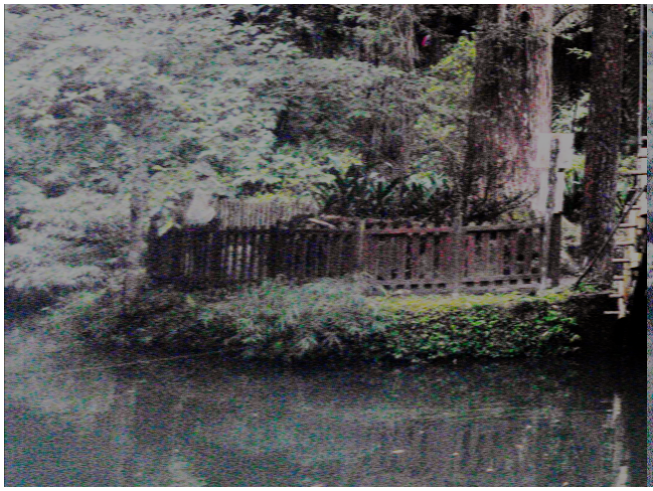
Image 2：

Origin

Log

Histogram Equalization

Histogram + Mean



Histogram + Alpha-Trimmed-Mean

Histogram + Bilateral

Image 3：

Origin

Piecewise-Mean

Histogram Equalization

Histogram + Bilateral

Image 4：

Origin

Gamma = 2.2

Log

Gamma + Bilateral

Image 5：

Origin

Gamma = 1/2.2

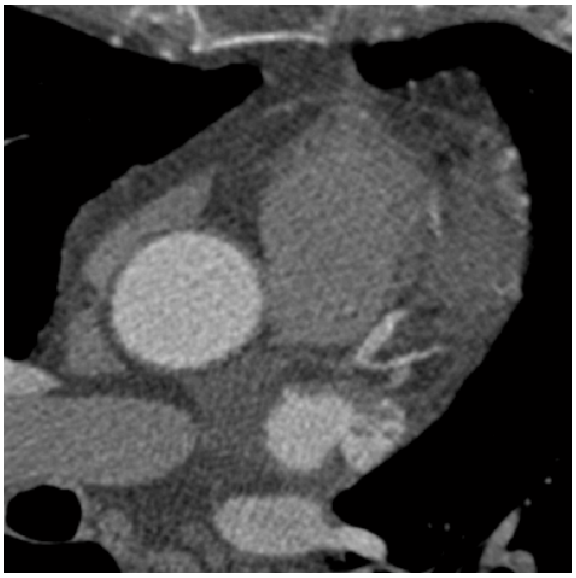
Piecewise-Linear

Histogram Equalization

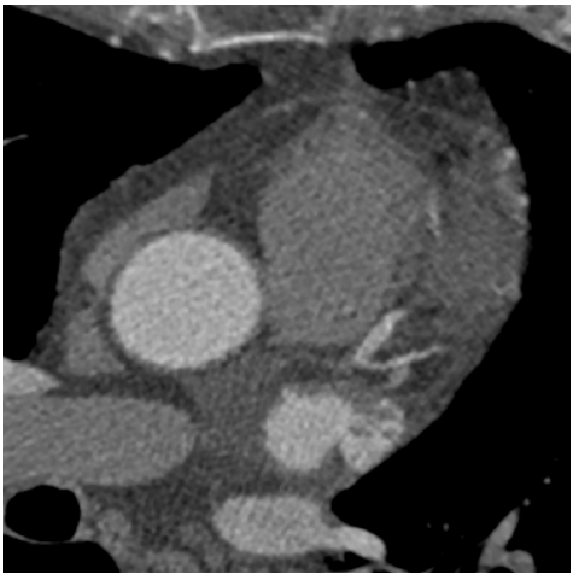
Piecewise + Alpha-Trimmed-Mean.

Piecewise + Bilateral

Image 6：



Origin



Bilateral

- Discussions：

1. 各張影像處理：

1-1.

圖一高曝光，對比低，Log & Histogram 會整個變超白，用 Gamma 剛剛好，而 mean & median 圖片會糊糊的，用 Bilateral 比較不會糊。

1-2.

圖二低曝光，對比低，Gamma 會變很黑，Log 可以變亮但對比低，Histogram 則剛好可以高對比，而 Mean 系列都會糊糊的，因此還是選擇 Bilateral。

1-3.

圖三有點像圖二的情況，因此選擇 Histogram + Bilateral。

1-4.

圖四希望可以將文字突出且背景不要那麼暗，因此選擇 Gamma + Bilateral，其他的 Filter 文字都會有點糊糊的。

1-5.

圖五清晰的照片但亮度偏暗，經過嘗試發現 Histogram 表現不錯但看起來不是那麼舒服，因此選擇 Piecewise，由於 Mean 系列都會讓樓頂的 A+糊掉，因此選擇 Bilateral

1-6.

圖六題目限定 Bilateral

2. 程式：

2-1.

在所有的 Filter 都是用 3x3 的 window 去實作，如果上升到 5x5 整個效率差很多。

2-2.

在實作方面發現 Filter 的效率都很差，去除 python 本身比較慢，Bilateral 的執行效率最差，網路查詢後發現有人利用積分影像來加速 Bilateral。

2-3.

沒有花時間去做指令化跑指定的影像處理的 Function，因此都是在原程式碼一個一個加入需要的 Function

- Code：

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
```

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()

# 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]

# 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)
# 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()
```