컴퓨터 비전 과제 1, 2

소프트웨어학과 20186889 권용한

1. 과제 1

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
#load
img_file='lena.jpg'
img_color=cv.imread(img_file)
#gray_scaling
b,g,r=cv.split(img_color.astype(np.uint16))
img_gray=((b+g+r)/3).astype(np.uint8)
#histogram before Equalization
hist_before=np.zeros(256)
#histogram after Equalization
hist_after=np.zeros(256)
h,w=img_gray.shape
bit_n=0
img_gray_res=img_gray.copy()
#LUT
LUT=np.zeros(256)

for i in range(h):
    for j in range(w):
        bit_n=img_gray[i][j]
        hist_before[bit_n]=hist_before[bit_n]+1

MN=h*w

sum_hist=0
#Calculation LUT
for i in range(256):
    sum_hist=sum_hist+hist_before[i]
    LUT[i]=255*sum_hist/MN

for i in range(h):
    for j in range(w):
        img_gray_res[i][j]=LUT[img_gray[i][j]]
        hist_after[img_gray_res[i][j]]=hist_after[img_gray_res[i][j]]+1

pdf_orig=hist_before/np.sum(hist_before)
cdf_orig=np.cumsum(pdf_orig)
pdf_Eq=hist_after/np.sum(hist_after)
cdf_Eq=np.cumsum(pdf_Eq)
```

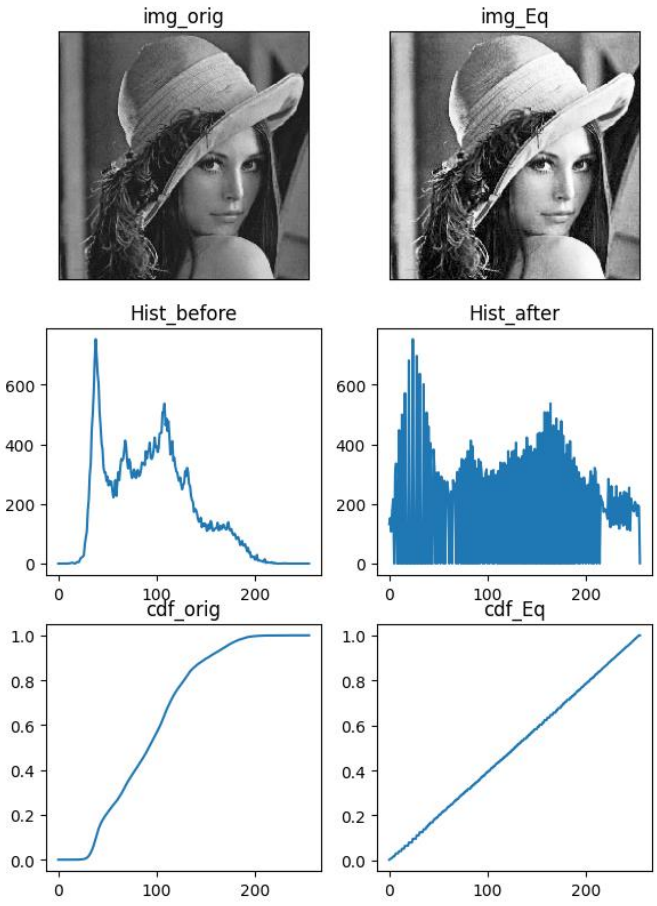
input으로 사용한 lena.png 의 이미지입니다.

Color 이미지를 사용했기 때문에, 과제에서 의도한 Histogram equalization을 수행하기 위해선 gray scale로 변경시켜 주어야 했습니다.

opencv 에서 imread 결과 uint8 형으로 r,g,b 데이터를 읽기 때문에, 해당 값의 평균치로 gray scaling 해주었으며, overflow 가 일어나는 것을 방지하기 위해 uint16 형으로 변환 후 계산한 후 uint8 형으로 변경을 진행하였습니다.

이후 LUT 를 만들었으며, img_gray 의 pixel 값들을 hist_before 로 미리 저장해 둡니다.

저장해 둔 값을 바탕으로 LUT 를 제작하고, 해당 식은 255*(hist_before(0~i 까지의 합)/전체 픽셀 수 로 구할 수 있습니다.



저장된 LUT 와 img_gray 를 바탕으로 새로운 img_gray_res 를 새롭게 만들었으며, LUT 에 맞게 기존 저장된 값을 변경시켜주었습니다.

이후 hist_after 에 새롭게 제작된 img_gray_res 의 pixel 값을 저장하고, hist 값을 통해 기존 이미지의 cdf 와 Equalization 이후의 cdf 를 계산한 후, 결과값들을 matplotlib.pyplot 모듈을 통해 display 합니다.

해당 이미지는 mlt.show 한 결과입니다.

출력 결과 histogram 이 더 고르게 출력된 것을 확인 가능하며, cdf 또한 기존에 치우친 모양에서 y=1/256x 에 근사한 결과를 확인할 수 있습니다.

2. 과제 2

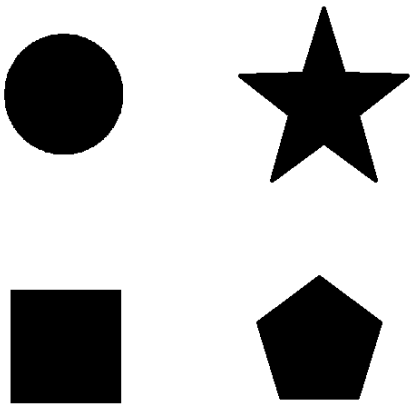
```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
import math

#AL function
def AL(arr,dist_func):
    for i in range(1,h):
        for j in range(1,w):
            arr[i][j]=min(arr[i][j],arr[i-1][j]+dist_func(-1,0))
            arr[i][j]=min(arr[i][j],arr[i-1][j-1]+dist_func(-1,-1))
            arr[i][j]=min(arr[i][j],arr[i][j-1]+dist_func(0,-1))
            if i!=h-1:
                arr[i][j]=min(arr[i][j],arr[i+1][j-1]+dist_func(1,-1))
#BR function
def BR(arr,dist_func):
    for i in range(h-1,-1,-1):
        for j in range(w-1,-1,-1):
            if i!=h-1:
                arr[i][j]=min(arr[i][j],arr[i+1][j]+dist_func(1,0))
            if j!=w-1:
                arr[i][j]=min(arr[i][j],arr[i][j+1]+dist_func(0,1))
            if i!=h-1 and j!=w-1:
                arr[i][j]=min(arr[i][j],arr[i+1][j+1]+dist_func(1,1))
            if i!=0 and j!=w-1:
                arr[i][j]=min(arr[i][j],arr[i-1][j+1]+dist_func(-1,1))
#Calculation for euclidean
def euclidean(a,b):
    return math.sqrt(math.pow(a,2)+math.pow(b,2))
#Calculation for city block
def city(a,b):
    return abs(a)+abs(b)
#Calculation for chess board
def chess(a,b):
    return max(abs(a),abs(b))
#load
img_file='CV_assign2.png'

img=cv.imread(img_file,cv.IMREAD_GRAYSCALE)
h,w=img.shape
print(h,w)
img_dist_orig=np.zeros((h,w))
#initialization
for i in range(h):
    for j in range(w):
```

```
        if img[i][j] == 0:
            img_dist_orig[i][j]=0
        elif img[i][j]==255:
            img_dist_orig[i][j]=h*w

img_dist_eucl=img_dist_orig.copy()
img_dist_city=img_dist_orig.copy()
img_dist_chess=img_dist_orig.copy()
#max length for scaling
max_eucl=0
max_city=0
max_chess=0
AL(img_dist_eucl,euclidean)
BR(img_dist_eucl,euclidean)
AL(img_dist_city,city)
BR(img_dist_city,city)
AL(img_dist_chess,chess)
BR(img_dist_chess,chess)
# Calculation max length
for i in range(h):
    for j in range(w):
        if max_eucl < img_dist_eucl[i][j]:
            max_eucl=img_dist_eucl[i][j]
        if max_city < img_dist_city[i][j]:
            max_city=img_dist_city[i][j]
        if max_chess < img_dist_chess[i][j]:
            max_chess=img_dist_chess[i][j]
#Scaling(max length to 255)
for i in range(h):
    for j in range(w):
        img_dist_eucl[i][j]=255*img_dist_eucl[i][j]/max_eucl
        img_dist_city[i][j]=255*img_dist_city[i][j]/max_city
        img_dist_chess[i][j]=255*img_dist_chess[i][j]/max_chess
```



해당 이미지는 input으로 사용한 CV_assign2.png 입니다. 해당 이미지를 읽은 후, img_dist_orig array를 만들어 검은색(object, img==0)이면 0, 흰색(배경, img==255)이라면 h*w값을 저장시켰습니다. (h*w는 충분히 큰 수라 생각해 사용했습니다.)

이후 해당 array를 img_dist_eucl 와 img_dist_city, img_dist_chess에 copy한 후, 각각의 max변수를 선언하며 거리 측정 및 Scaling을 위해 필요한

준비를 합니다.

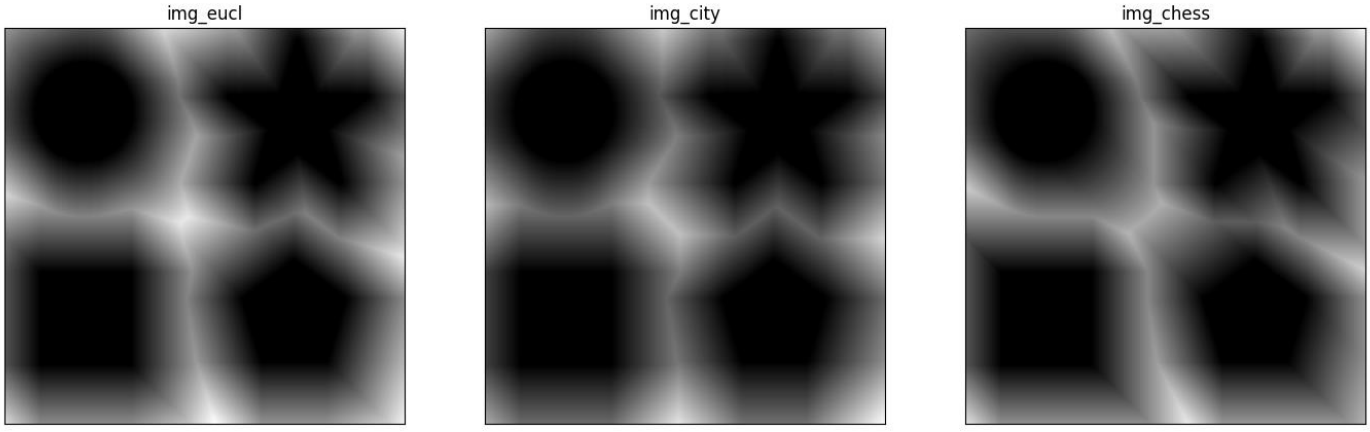
AL과 BR함수를 각각 선언하며, 해당 함수는 거리 측정 방법마다 필요로 하는 array와 function(거리측정 함수)를 매개변수로 받도록 하였습니다.

AL의 경우 (i,j)를 기준으로 (i-1,j-1), (i-1,j), (i,j-1), (i+1,j-1)에서 각 블록과의 거리만큼 추가한 후, 거리가 가장 짧은 위치를 선택하였으며, 구현의 편의상 (1,1)에서 (h-1,w-1)까지 반복하며, 이때 overflow를 방지하기 위해 i가 h-1일 경우엔 예외처리를 하였습니다.

BR도 마찬가지로 i가 0인 경우 예외처리를 필요로 하며, 또한 AL에서 제대로 측정하지 못하는 위치를 파악하기 위해 추가적인 예외처리를 하여 모든 pixel에 유의미한 값이 저장되도록 하였습니다.

확인결과 obj가 등장하기 전까지의 열들의 경우 유의미한 값이 입력되지 않고, 이를 고려하지 않고 편의를 위해 (h-2,w-2)부터 측정을 시작한다면 각 열에서 마지막 행, 또는 각 열에서 마지막 행에서 dummy data가 유의미한 data로 전환되지 않음을 확인하였으며, 때문에 (h-1,w-1)부터 탐색을 진행하며 이때 발생 가는한 overflow를 일으키는 범위에 예외처리를 진행해 빠지는 부분 없이 탐색 및 측정을 진행하였습니다.

각 측정방식별로 AL과 BR을 진행한 이후 거리의 최대값을 255로 정규화하는 과정을 거쳤으며, 이후 해당 내용을 출력하였습니다.



해당 이미지는 정규화 이후 결과 이미지로, 유사해 보이지만 측정방법에 따라 결과가 차이가 나타남을 확인할 수 있습니다.