Project 1

202011279 김하람

- Used Programming Language : Python3.7
- Used IDE : Pycharm Community Edition 2020.2.3
- Used APIs : Python Libraries [openCV / numpy / math]
- Collaborator : 202011249 고유경
- Submission Date : 2020.10.13.Tue

1. Write a computer program that performs the following.

·◌Code

```
import cv2
import numpy as np
import math


# ImageFile load
imageFile = '../image/taran_grayscale.jpg'
imgMat = cv2.imread(imageFile, 0) / 255
n, n = imgMat.shape


# Denormalized_HaarMatrix return function
def dHaarMatrix(n):
    if n == 1:
        return np.array([1.0])
    else:
        hm = np.kron(dHaarMatrix(int(n / 2)), ([1], [1]))
        hi = np.kron(np.identity(int(n / 2), dtype=int), ([1], [-1]))
        h = np.hstack([hm, hi])
        h = np.array(h, dtype=float)

    return h


# Normalize HaarMatrix function
def normalize(h, n):
    for i in range(0, n):
        temp = 0
        j = 0

        for j in range(0, n):
            if h[j][i] != 0:
                temp += 1

        if temp != 0:
            h[:, i] = 1 / math.sqrt(temp) * h[:, i]

    return h


# Discrete Haar Wavelet Transform return function
def getB(imgMat, HaarMat):
    return np.dot(np.dot(HaarMat.T, imgMat), HaarMat)


# B hat return function
def getBHat(B, s, n):
    k = 2**s

    for i in range(k, n):
        for j in range(0, k):
            B[i, j] = 0

    for i in range(0, n):
        for j in range(k, n):
            B[i, j] = 0

    return B
```

```
# A Hat return function
def getAHat(BHat, HaarMat):
    return np.dot(np.dot(HaarMat, BHat), HaarMat.T)


s = int(input("s:"))
HaarMat = normalize(dHaarMatrix(n), n)
B = getB(imgMat, HaarMat)
BHat = getBHat(B, s, n)
AHat = getAHat(BHat, HaarMat)

cv2.imshow('test', AHat)
cv2.waitKey(0)
```

·◌How to Work

- 사용자에게 s 값을 입력 받아 B hat 을 생성할 때 사용할 k 를 지정해주었고, 이에 따라 이미지를 윈도우에 보여주도록 함.

2. Get any 2 grayscale images of any format from anywhere (Internet, your personal photos, etc.). It is recommended that you get one image with low frequency components, and one image filled with high frequency components. Use your computer program to do the following.

·◌Code

```
import cv2
import numpy as np
import math


# ImageFile load
lowFreqImageFile = '../image/sky_grayscale.jpg'
highFreqImageFile = '../image/taran_grayscale.jpg'
lowFreqImgMat = cv2.imread(lowFreqImageFile, 0) / 255
highFreqImgMat = cv2.imread(highFreqImageFile, 0) / 255
n, n = lowFreqImgMat.shape


# Denormalized_HaarMatrix return function
def dHaarMatrix(n):
    if n == 1:
        return np.array([1.0])
    else:
        hm = np.kron(dHaarMatrix(int(n / 2)), ([1], [1]))
        hi = np.kron(np.identity(int(n / 2), dtype=int), ([1], [-1]))
        h = np.hstack([hm, hi])
        h = np.array(h, dtype=float)

    return h


# HaarMatrix normalize function
def normalize(h, n):
    for i in range(0, n):
        temp = 0
        j = 0

        for j in range(0, n):
            if h[j][i] != 0:
                temp += 1

        if temp != 0:
            h[:, i] = 1 / math.sqrt(temp) * h[:, i]

    return h


# Discrete Haar Wavelet Transform return function
def getB(imgMat, HaarMat):
    return np.dot(np.dot(HaarMat.T, imgMat), HaarMat)


# B hat return function
def getBHat(B, s, n):
    k = 2**s

    for i in range(k, n):
        for j in range(0, k):
```

```
            B[i, j] = 0

    for i in range(0, n):
        for j in range(k, n):
            B[i, j] = 0

    return B


# A Hat return function
def getAHat(BHat, HaarMat):
    return np.dot(np.dot(HaarMat, BHat), HaarMat.T)


HaarMat = normalize(dHaarMatrix(n), n)
# Observe the quality of reconstructed image (low-freq components)
for s in range(int(math.log2(n)/math.log2(2)) + 1):
    B = getB(lowFreqImgMat, HaarMat)
    BHat = getBHat(B, s, n)
    AHat = getAHat(BHat, HaarMat)

    originalImg = AHat * 255
    cv2.imwrite('../image/lowResult/Reconstructed-' + str(2**s) + 'by' + str(2**s)
+ '.jpg', originalImg)


# Observe the quality of reconstructed image (high-freq components)
for s in range(int(math.log2(n)/math.log2(2)) + 1):
    B = getB(highFreqImgMat, HaarMat)
    BHat = getBHat(B, s, n)
    AHat = getAHat(BHat, HaarMat)

    originalImg = AHat * 255
    cv2.imwrite('../image/highResult/Reconstructed-' + str(2**s) + 'by' +
str(2**s) + '.jpg', originalImg)
```

·◌Main Code Description

Line	Description
9, 10	cv2.imread로 불러온 이미지 파일 행렬 각 요소를 255로 나눠줌으로 정규화
14-24	denormalized Haar Matrix를 반환하는 함수로써, 크로네커 곱을 수행하는 np.kron() 연산과 단위행렬을 만드는 np.identity(), 두 행렬을 연결하는 np.hstack(), 행렬 요소를 실수형으로 반환하는 dtype=float을 사용하여 구성되었으며 재귀 함수로 구성
27-40	Denormalized Haar Matrix의 각 column의 길이가 1이 되도록 정규화
43-45	DHWT을 수행하여 B를 반환하는 함수로, np.dot() 연산으로 행렬 곱 수행
48-60	B의 kxk upper left corner를 제외한 나머지 요소를 0으로 바꾸어 B hat을 반환하는 함수
69-86	반복문 for로 2^s가 n이 될 때까지 s값을 0부터 1씩 증가됨에 따라 k값이 증가하고 이에 재구성된 이미지를 cv2.imwrite()로 저장하는 구문. Line9, 10에서 이미지를 255로 나누어 정규화 해주었으므로 여기서 255를 다시 스칼라 곱을 해주어 원본 이미지 행렬을 만들도록 함

·◌(a) As k increases, observe the quality of reconstructed image.

- high-freq. image 의 재구성된 이미지를 ./LAproject/image/highResult 에, low-freq. image 의 재구성된 이미지를 ./LAproject/image/lowResult 에 저장하도록 하였음.

<Image filled with high frequency components>

<Image filled with low frequency components>

·◌(b) Describe any difference between low-freq. image and high-freq. image.

- high-freq. image 와 low-freq. image 를 선정하는 기준은 행렬의 각 요소와 주변 값들과의 차이 즉, 이미지의 색 변화 정도의 차이가 크면 high frequency 요소가 많은 것이고, 차이가 작다면 low frequency 요소가 많은 것으로 간주하여 high-freq. image 는 인물 사진, low-freq. image 는 구름 있는 하늘 사진으로 선정하였다. 두 이미지의 차이는 k 값이 원본 이미지의 픽셀 값에서 낮아질수록 인물과 구름은 비슷한 정도 차이로 화질이 변화하였지만 구름 사진의 하늘 부분은 색 변화가 극심하지 않아 k 값이 16 일 때까지는 형태를 알아볼 수 있었고 전 단계와 차이도 크지 않았다.

·◌(c) Discuss any findings or thoughts.

- (a)에서 k 값을 증가하면서 화질 변화를 확인하였을 때, 인물사진의 원본 픽셀이 512 x 512 였지만 128 x 128 즉 저장공간을 1/16 으로 줄였음에도 불구하고 인물의 이목구비와 옷에 써져 있는 글씨 등은 알아볼 수 있었다. 동시에 사진 압축 기술인 DHWT 와 IDHWT 을 고안해낸 것에서 대단함을 느꼈다.

3. For n-point Haar matrix H, define Hl and Hh as follows. That is, Hl and Hh are the top half and bottom half rows of HT . Note that Hl (or Hh) is a part of basis capturing relatively low (or high) frequency components.

·◌Code

```
import cv2
import numpy as np
import math


# ImageFile load
ImageFile = '../image/taran_grayscale.jpg'
imgMat = cv2.imread(ImageFile, 0) / 255
n, n = imgMat.shape


# Denormalized_HaarMatrix return function
def dHaarMatrix(n):
    if n == 1:
        return np.array([1.0])
    else:
        hm = np.kron(dHaarMatrix(int(n / 2)), ([1], [1]))
        hi = np.kron(np.identity(int(n / 2), dtype=int), ([1], [-1]))
        h = np.hstack([hm, hi])
        h = np.array(h, dtype=float)

    return h


# Normalize HaarMatrix function
def normalize(h, n):
    for i in range(0, n):
        temp = 0
        j = 0

        for j in range(0, n):
            if h[j][i] != 0:
                temp += 1

        if temp != 0:
            h[:, i] = 1 / math.sqrt(temp) * h[:, i]

    return h


# DHWT B return function
def DHWT(imgMat, HaarMat):
    return np.dot(np.dot(HaarMat.T, imgMat), HaarMat)


# IDWHT A return function
def IDHWT(imgMat, HaarMat):
    return np.dot(np.dot(HaarMat, imgMat), HaarMat.T)


# check if two matrices are same function
def checkMat(mat1, mat2, alphabet):
    for i in range(n):
        for j in range(n):
```

```
            if mat1[i, j] != mat2[i, j]:
                print(alphabet + " : Given expression is False!")
                return False

    print(alphabet + " : Given expression is True!")
    return True


# (a)
HaarMat = normalize(dHaarMatrix(n), n)
Hl = HaarMat.T[:int(n / 2), :]
Hh = HaarMat.T[int(n / 2):, :]

matA1 = IDHWT(imgMat, Hl)
matA2 = np.dot(np.dot(Hl, imgMat), Hh.T)
matA3 = np.dot(np.dot(Hh, imgMat), Hl.T)
matA4 = IDHWT(imgMat, Hh)

resultA = np.vstack([np.hstack([matA1, matA2]), np.hstack([matA3, matA4])])

checkMat(DHWT(imgMat, HaarMat), resultA, '(a)')


# (b)
matB1 = DHWT(IDHWT(imgMat, Hl), Hl)
matB2 = np.dot(np.dot(Hl.T, np.dot(np.dot(Hl, imgMat), Hh.T)), Hh)
matB3 = np.dot(np.dot(Hh.T, np.dot(np.dot(Hh, imgMat), Hl.T)), Hl)
matB4 = DHWT(IDHWT(imgMat, Hh), Hh)

resultB = matB1 + matB2 + matB3 + matB4

checkMat(IDHWT(DHWT(imgMat, HaarMat), HaarMat), resultB, '(b)')


# (c)
cv2.imwrite('../image/IDHWT(b)/(H)Reconstructed-term1.jpg', matB1 * 255)
cv2.imwrite('../image/IDHWT(b)/(H)Reconstructed-term2.jpg', matB2 * 255)
cv2.imwrite('../image/IDHWT(b)/(H)Reconstructed-term3.jpg', matB3 * 255)
cv2.imwrite('../image/IDHWT(b)/(H)Reconstructed-term4.jpg', matB4 * 255)


# (d)
Hll = Hl[:int(n / 4), :]
Hlh = Hl[int(n / 4):, :]

matD1 = DHWT(IDHWT(imgMat, Hll), Hll)
matD2 = np.dot(np.dot(Hll.T, np.dot(np.dot(Hll, imgMat), Hlh.T)), Hlh)
matD3 = np.dot(np.dot(Hlh.T, np.dot(np.dot(Hlh, imgMat), Hll.T)), Hll)
matD4 = DHWT(IDHWT(imgMat, Hlh), Hlh)

resultD = matD1 + matD2 + matD3 + matD4

checkMat(matB1, resultD, '(d)')

cv2.imwrite('../image/IDHWT(d)/(H)Reconstructed-term1.jpg', matD1 * 255)
cv2.imwrite('../image/IDHWT(d)/(H)Reconstructed-term2.jpg', matD2 * 255)
cv2.imwrite('../image/IDHWT(d)/(H)Reconstructed-term3.jpg', matD3 * 255)
cv2.imwrite('../image/IDHWT(d)/(H)Reconstructed-term4.jpg', matD4 * 255)
```

·◌Main Code Description

Line	Description
41-43	DHWT 연산을 하여 B를 반환하는 함수
46-48	IDHWT 연산을 하여 A를 반환하는 함수
51-60	(a), (b), (d) 문제에서 좌변의 식과 우변의 식이 같은지를 검사하는 함수로 중첩 반복문으로 행렬의 각 요소를 비교하도록 함
63-75	(a)를 수행하기 위한 구문. Normalized Haar Matrix를 슬라이싱하여 Hl과 Hh를 만들어 각 term을 변수 matA1부터 matA4까지 저장하고 np.hstack()과 np.vstack()연산으로 행렬을 연결하여 checkMat()함수 호출
78-86	(b)를 수행하기 위한 구문. 각 term을 변수 matB1부터 matB4까지 저장하고 반환된 행렬은 numpy라이브러리에서 제공하는 np.array로 반환되기에 행렬의 덧셈은 + 연산자로 덧셈 수행하여 결과 값을 checkMat()함수 인자로 넘겨줌
89-93	(c)를 수행하기 위한 구문으로 (b)에서 연산한 4개의 term을 파일로 저장
96-112	(d)를 수행하기 위한 구문으로 Hl을 슬라이싱하여 Hll과 Hlh를 만들어 각 term을 변수 matD1부터 matD4까지 저장하고 덧셈 연산하여 checkMat() 함수 호출. 각 term을 파일로 저장

·◌(a) Show that the DHWT B = H^TAH is given as

$$H^T AH = \begin{bmatrix} H_l A H_l^T & H_l A H_h^T \\ H_h A H_l^T & H_h A H_h^T \end{bmatrix}$$

- HlAHl^T 는 matA1, HlAHh^T 는 matA2, HhAHl^T 는 matA3, HhAHh^T 는 matA4 변수에 연산 결과를 저장하였고 각 term 을 연결하여 resultA 에 저장하였다. resultA 와 H^TAH 를 checkMat() 함수에 인자로 넘겨줌으로써 두 행렬이 같은지 확인함.

```
(a) : Given expression is True!
```

·◌(b) Show that the IDHWT A = HBH^T is given as

$$HBH^T = H_l^T H_l A H_l^T H_l + H_l^T H_l A H_h^T H_h + H_h^T H_h A H_l^T H_l + H_h^T H_h A H_h^T H_h$$

- (a)와 같은 방식으로 각 term 을 변수에 저장하였고 이를 np.array 타입으로 덧셈 연산을 해주었다. 그러나 두 식이 다르다는 결과가 나왔고 값이 다른 요소를 각 행렬에서 출력해보았는데 다음 이미지와 같이 매우 작은 값이 다름을 알게 되었다.

```
mat1 = IDHWT(DHWT(imgMat, HaarMat), HaarMat)
mat2 = resultB

print(mat1[4, 10])
print(mat2[4, 10])
```

```
(b) : Given expression is False!
0.6627450980392146
0.6627450980392144
```

위와 같은 결과가 나온 이유를 생각해보았고, 원인은 정규화로 인하여 수가 매우 작아졌기 때문이었다. (b)를 증명하기 위해 A 는 이미지 파일을 255 로 나누어 정규화 해준 행렬을, H 는 각 column 의 길이가 1 이 되도록 정규화한 Haar Matrix 를 사용했기에 float 타입으로 표현하다 보니 수가 올림 되었기 때문이라고 생각하였다. 그래서 이미지 정규화와 Haar Matrix 정규화 연산을 제외시키고 다시 비교를 해보니 주어진 식이 참이라는 결과를 얻게 되었다.

```
(b) : Given expression is True!
```

·◌(c) Each term in (b) is an n × n matrix. Show each term in (b) as an image, and discuss what each of matrices contains.

[Image filled with high frequency components]

<term1>

<term2>

<term3>

<term4>

- term1 : 원본 이미지와 비교했을 때 윤곽선이 자연스럽지 않고 픽셀이 두드러지게 보임을 알 수 있다.

<original image>

<term1>

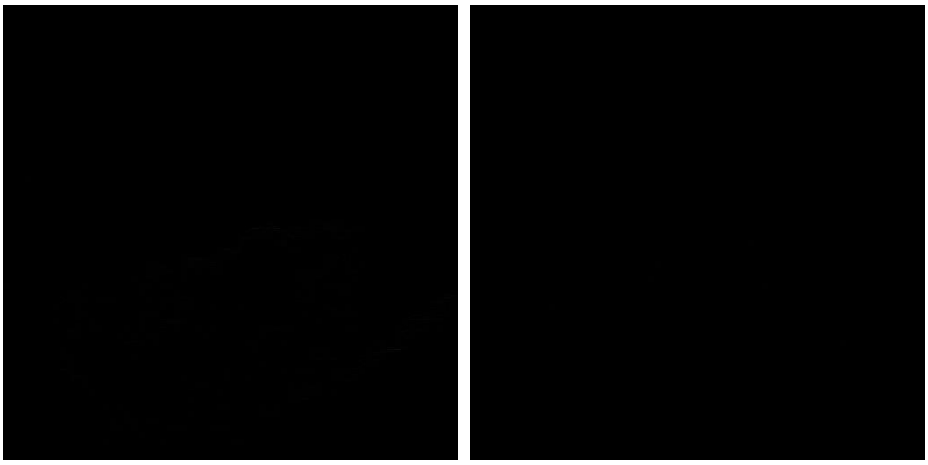
- term2 : 이미지의 왼쪽 옷과 뒤의 벽 사이의 경계선이 출력되지 않았고 그 사이에서 옷의 단추만 출력된 것으로 보아, 인접하는 두 물체의 색 차이가 큰 부분의 경계선만을 나타냄을 알 수 있다. 세로로 된 선 위주로 출력 됨도 알 수 있다.

-term3 : term2 에서는 세로선 위주로 출력 됐다면 term3 에서는 term2 에서 출력되지 않은 인물의 입이 출력됨으로 보아 가로선 위주로 출력 됨을 알 수 있다.

-term4 : term2 와 term3 에서 표현되지 않은 선들을 보강해주는 느낌이었다.

[Image filled with low frequency components]

<term1> <term2>



<term3> <term4>

- term1 : 원본 이미지를 두고도 두 사진을 구분하기 어려울 정도로 비슷했다.

<original image> <term1>

- term2 : 위의 사진 속에서는 잘 안보일지 모르지만 사진을 확대하거나 화면을 어둡게 하면 희미하게 형태가 보인다. term3 와 비교했을 때 확실히 세로선 위주로 출력됨을 알 수 있다.

- term3 : 마찬가지로 희미하게 보인다. term2 와 비교했을 때 확실히 가로선 위주로 출력됨을 알 수 있다. 구름의 가로 길이가 세로 길이보다 길어 더 많은 선을 출력하였다.

- term4 : 거의 보이지 않지만 term2 와 term3 를 보강하는 선들이 보인다.

=> high frequency 가 함유된 이미지보다 low frequency 가 함유된 이미지가 term1 에서는 원본 이미지와 큰 차이를 보이지는 않지만 term2, 3, 4 에서는 인접한 물체의 색 차이가 크지 않아 선이 뚜렷하지 않고 희미하게 보임을 알 수 있었다.

(d) Repeat (a) and (b) for the first term in (b). That is, let $H_l \begin{bmatrix} H_{ll} \\ H_{lh} \end{bmatrix}$, and substitute this into the first term in (b) and show that

$$H_l^T H_l A H_l^T H_l = H_{ll}^T H_{ll} A H_{ll}^T H_{ll} + H_{ll}^T H_{ll} A H_{lh}^T H_{lh} + H_{lh}^T H_{lh} A H_{ll}^T H_{ll} + H_{lh}^T H_{lh} A H_{lh}^T H_{lh}.$$

Show each term as an image on the screen, and interpret them.

- (b)에서와 정규화된 이미지와 Haar Matrix 를 사용했을 때 주어진 식이 거짓임을 보였고 이 역시 매우 작은 값에서 오차가 발생함을 알고 정규화를 제외하여 비교한 결과 주어진 식이 참임을 보였다.

```
(d) : Given expression is False!
0.6647058823529407
0.6647058823529409
```

```
(d) : Given expression is True!
```

[Image filled with high frequency components]

<term1>

<term2>

<term3>

<term4>

- term1 : (b)에서의 term1 과 비교했을 때 윤곽선의 뚜렷함이 줄어들었고 픽셀도 더 잘 보인다.

<term1 in (b)> <term1 in (d)>

- term2, 3, 4 : (b)에서의 term2, 3, 4 보다 더 뚜렷하다.

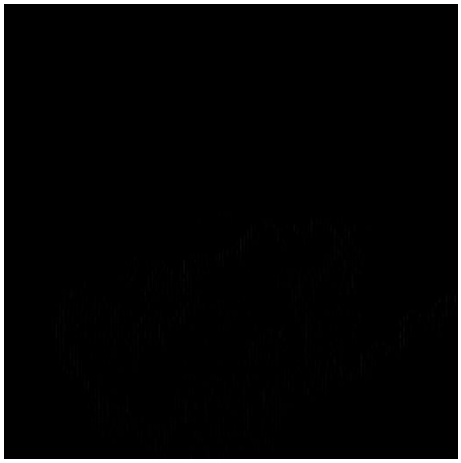

<term2 in (b)> **<term2 in (d)>**


<term3 in (b)> **<term3 in (d)>**

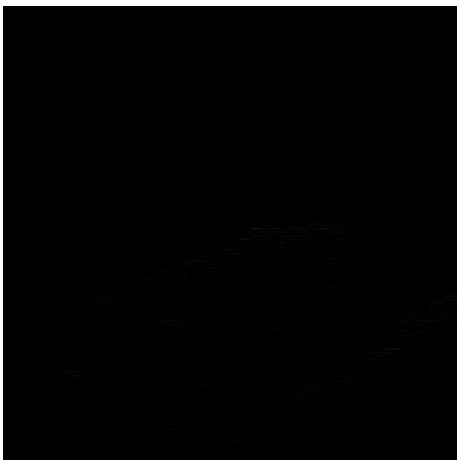

<term4 in (b)> **<term4 in (d)>**

[Image filled with low frequency components]

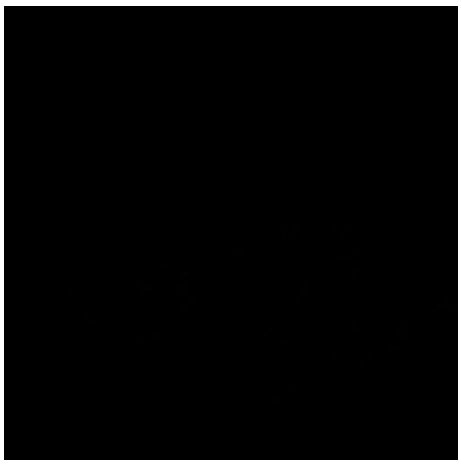
<term1>



<term2>



<term3>



<term4>

·◌Discussion

-(b)와 (d)에서 수행한 내용은 DHWT했던 이미지를 IDHWT로 복원할 때 상대적인 low frequency 성분과 high frequency 성분을 분리한 후 그 중 low frequency 성분에 대해 이미지를 복원하여 출력하여 의미를 해석하는 것이다. 사람이 눈으로 보는 이미지라는 것이 low freq.성분과 high freq.성분으로 나뉘며 이것을 보통 구분할 수 없지만 이번 프로젝트를 통해 이를 구분하여 볼 수 있었던 점이 좋은 경험이 되었다.