Image Filtering 실습

Import Library

```
from PIL import Image
import numpy as np
import math
```

Part 1 Gaussian Filtering

 # 1.

Filter는 2k+1(k = 0, …)의 값이 되어야 하므로 assert를 이용해서 조건이 True이면 통과, False이면 assertion Error가 발생하도록 하였다.

boxfilter 함수 구현.

```
def boxfilter(n):
    # n % 2 == 1이면 통과, 아니면 Error 문구 출력하고 에러 처리
    assert n % 2 == 1, "Dimension must be odd"
    # numpy 이용해서 행렬 형태 만들어준다.
    ret = (1/(n**2))*np.ones([n, n])
    return ret
```

간단한 예시

1) 인자로 3 입력

```
boxfilter(3)
```

```
array([[0.11111111, 0.11111111, 0.11111111],
       [0.11111111, 0.11111111, 0.11111111],
       [0.11111111, 0.11111111, 0.11111111]])
```

2) 인자로 4 입력

```
boxfilter(4)
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-4-5870f78beb34> in <module>
----> 1 boxfilter(4)

<ipython-input-2-57c29ebf796a> in boxfilter(n)
      3 def boxfilter(n):
      4     # n % 2 == 1이면 통과, 아니면 Error 문구 출력하고 에러 처리
----> 5     assert n % 2 == 1, "Dimension must be odd"
      6     # numpy 이용해서 행렬 형태 만들어준다.
      7     ret = (1/(n**2))*np.ones([n, n])

AssertionError: Dimension must be odd
```

3) 인자로 7 입력

```
boxfilter(7)
```

```
array([[0.02040816, 0.02040816, 0.02040816, 0.02040816, 0.02040816,
        0.02040816, 0.02040816],
       [0.02040816, 0.02040816, 0.02040816, 0.02040816, 0.02040816,
        0.02040816, 0.02040816],
       [0.02040816, 0.02040816, 0.02040816, 0.02040816, 0.02040816,
        0.02040816, 0.02040816],
       [0.02040816, 0.02040816, 0.02040816, 0.02040816, 0.02040816,
        0.02040816, 0.02040816],
       [0.02040816, 0.02040816, 0.02040816, 0.02040816, 0.02040816,
        0.02040816, 0.02040816],
       [0.02040816, 0.02040816, 0.02040816, 0.02040816, 0.02040816,
        0.02040816, 0.02040816],
       [0.02040816, 0.02040816, 0.02040816, 0.02040816, 0.02040816,
        0.02040816, 0.02040816]])
```

2.

분모가 0이 되므로 sigma value가 0이 되지 않도록 설정하였다.

np.arange를 이용해서 문제의 [-5, -4, -3, …, 4, 5] 과 같은 방식을 구현하였다.

gauss1d(sigma) 함수 구현

```
def gauss1d(sigma):
    # sigma value가 0이 되서는 안된다.(sigma value가 0이면 분모가 0이 된다.)
    assert sigma != 0, "Sigma value sholudn't be zero"

    # sigma를 6으로 곱하고 올림
    length = int(math.ceil(float(sigma) * 6))

    # 짝수라면 홀수로 바꾸기 위해서 +1
    if length % 2 == 0:
        length += 1

    center = length // 2

    # -center ~ center의 1차원 배열 생성
    x = np.arange(-center, center + 1)

    # sum to 1을 위한 처리
    ret = np.exp(-(x**2)/(2*(sigma**2)))
    ret /= np.sum(ret)

    return ret
```

<u>간단한 예시</u>

- 각각 0.3, 0.5, 1, 2를 gauss1d 함수에 인자로 전달.

```
gauss1d(0.3)
```

```
array([0.00383626, 0.99232748, 0.00383626])
```

```
gauss1d(0.5)
```

```
array([0.10650698, 0.78698604, 0.10650698])
```

```
gauss1d(1)
```

```
array([0.00443305, 0.05400558, 0.24203623, 0.39905028, 0.24203623,
       0.05400558, 0.00443305])
```

```
gauss1d(2)
```

```
array([0.0022182 , 0.00877313, 0.02702316, 0.06482519, 0.12110939,
       0.17621312, 0.19967563, 0.17621312, 0.12110939, 0.06482519,
       0.02702316, 0.00877313, 0.0022182 ])
```

3.

앞서 만들었던 gauss1d 함수를 이용하여 외적 연산(np.outer())을 수행해 gauss2d를 구현하였다.

gauss2d 구현

```
def gauss2d(sigma):
    # np.outer(외적) 연산
    ret = np.outer(gauss1d(sigma), gauss1d(sigma))
    ret /= np.sum(ret)
    return ret
```

간단한 예시

- 각각 0.5, 1을 gauss2d 함수에 인자로 전달

```
gauss2d(0.5)
```

```
array([[0.01134374, 0.08381951, 0.01134374],
       [0.08381951, 0.61934703, 0.08381951],
       [0.01134374, 0.08381951, 0.01134374]])
```

```
gauss2d(1)
```

```
array([[1.96519161e-05, 2.39409349e-04, 1.07295826e-03, 1.76900911e-03,
        1.07295826e-03, 2.39409349e-04, 1.96519161e-05],
       [2.39409349e-04, 2.91660295e-03, 1.30713076e-02, 2.15509428e-02,
        1.30713076e-02, 2.91660295e-03, 2.39409349e-04],
       [1.07295826e-03, 1.30713076e-02, 5.85815363e-02, 9.65846250e-02,
        5.85815363e-02, 1.30713076e-02, 1.07295826e-03],
       [1.76900911e-03, 2.15509428e-02, 9.65846250e-02, 1.59241126e-01,
        9.65846250e-02, 2.15509428e-02, 1.76900911e-03],
       [1.07295826e-03, 1.30713076e-02, 5.85815363e-02, 9.65846250e-02,
        5.85815363e-02, 1.30713076e-02, 1.07295826e-03],
       [2.39409349e-04, 2.91660295e-03, 1.30713076e-02, 2.15509428e-02,
        1.30713076e-02, 2.91660295e-03, 2.39409349e-04],
       [1.96519161e-05, 2.39409349e-04, 1.07295826e-03, 1.76900911e-03,
        1.07295826e-03, 2.39409349e-04, 1.96519161e-05]])
```

4.

convolve 연산을 수행하기 위해 numpy를 이용하여 좌우, 상하 반전을 통해 뒤집어주었으며, filter의 크기에 맞는 same padding을 구현하기 위해서 zero padding을 해주었다. convolution 연산 후 위치에 맞는 곳에 합계를 넣어주기 위해서 2중 for loop를 이용하였다.

```
def convolve2d(array, filter):
    # np.flipud : 위, 아래 방향 뒤집기
    # np.fliplr : 좌, 우 방향 뒤집기

    filter = np.flipud(np.fliplr(filter))

    pad = int(np.sqrt(filter.size)) # 19
    pad2 = pad//2 # 9

    # 리턴값
    output = np.zeros_like(array, dtype = 'float32')

    # Zero padding
    padded = np.zeros((array.size[1] + 2*pad2, array.size[0] + 2*pad2))
    padded[pad2:-pad2, pad2:-pad2] = array

    # 이중 for loop로 zero padding 구현
    for x in range(array.size[0]):
        for y in range(array.size[1]):
            output[y,x] = (filter * padded[y:y+pad, x:x+pad]).sum()

    return output
```

```python
from scipy import signal # 테스트 용도
def gaussconvolve2d(array, sigma):
    # sigma를 통해 gauss를 통해 convolution 연산을 통해 결과를.
    # convolve2d와 gauss2d를 이용해서 만들면 된다.
    filter = gauss2d(sigma)
    # output = signal.convolve2d(array, filter, 'same') # 테스트 용도
    output = convolve2d(array, filter)

    return output
```

앞서 구현한 gauss2d를 이용하기 위해서 grayscale로 변환해준 후에 gaussconvolve2d(array, sigma)함수를 적용하였으며, 배열을 float32로 변환해준 후 다시 정수형으로 바꿔서 이미지 파일로 변환하여 출력하였다.

테스트 용도로 scipy의 signal을 사용하였는데, 동일한 결과값이 나와서 제대로 구현한 것을 확인할 수 있었다.

Original Image

```python
# dog사진을 gaussconvolve2d를 통해 나온 결과값을 출력하면 된다.
im = Image.open('Image/hw2_image/2b_dog.bmp')
print(type(im))
# color to grayscale
im = im.convert('L')
im.show()
```

Filtered Image

```python
output = gaussconvolve2d(im, 3)
print(output)
# PIL로 출력하기 위해서 정수형 'uint8'로 변경
print(type(output))
output = output.astype('uint8')
output2 = Image.fromarray(output)
output2.show()
```

[변경 전] **[변경 후]**

Part 2 Hybrid Images

low-pass filter : sigma가 높으면 blur가 많이 되고, sigma가 낮으면 blur가 적게 된다. 적절한 값을 조정해서 blur된 이미지가 나오도록 구현하였다. RGB Channel 각각으로 분리한 다음 gaussian filter를 적용해서 convolve 연산해준 후에 RGB Channel을 합쳤다. 이 때, 각 RGB Channel의 sigma는 동일하게 구현하였다.

Bicycle과 Motorcycle을 Pair로 적용하였다.

 # 1.

low-pass filter

split()을 이용해 각 채널을 나눠준 후에, 각각의 channel(RGB)마다 gaussian filter를 이용해서 low-pass filter로 만들어주었으며, filter 적용이 모두 완료된 후에 merge()를 이용하여 합쳐주었다.

```
im1a = Image.open('Image/hw2_image/1a_bicycle.bmp').convert('RGB')
```

```
# split()을 이용해 각 채널을 나누었다.
red, green, blue = im1a.split()
```

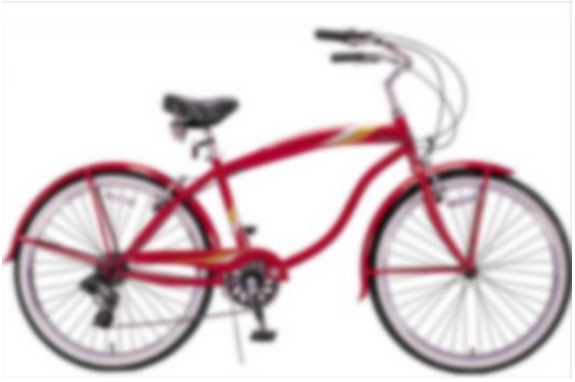
```
red = gaussconvolve2d(red, 2)
print(red)
# PIL로 출력하기 위해서 정수형 'uint8'로 변경
red = red.astype('uint8')
red = Image.fromarray(red)
red.show()
```

```
green = gaussconvolve2d(green, 2)
print(red)
# PIL로 출력하기 위해서 정수형 'uint8'로 변경
green = green.astype('uint8')
green = Image.fromarray(green)
green.show()
```

```
blue = gaussconvolve2d(blue, 2)
print(red)
# PIL로 출력하기 위해서 정수형 'uint8'로 변경
blue = blue.astype('uint8')
blue = Image.fromarray(blue)
blue.show()
```

```
# merge()를 이용해 각 채널을 합쳤다.
im1a_lowfreq = Image.merge("RGB", (red, green, blue))
im1a_lowfreq.show()
```

[Output]



2.

high-pass filter

앞선 1번의 방법과 같이 low-pass filter를 만들었으며, 그 후 원본 이미지를 활용하여 high-pass filter를 만들어주었다.

```
im1b = Image.open('Image/hw2_image/1b_motorcycle.bmp').convert('RGB')
```

```
red_1b, green_1b, blue_1b = im1b.split()
```

```
red_1b_lowfreq = gaussconvolve2d(red_1b, 10)
red_1b_lowfreq = red_1b_lowfreq.astype('uint8')
red_1b_lowfreq = Image.fromarray(red_1b_lowfreq)
# red_1b_lowfreq.show()

green_1b_lowfreq = gaussconvolve2d(green_1b, 10)
green_1b_lowfreq = green_1b_lowfreq.astype('uint8')
green_1b_lowfreq = Image.fromarray(green_1b_lowfreq)

blue_1b_lowfreq = gaussconvolve2d(blue_1b, 10)
blue_1b_lowfreq = blue_1b_lowfreq.astype('uint8')
blue_1b_lowfreq = Image.fromarray(blue_1b_lowfreq)
```

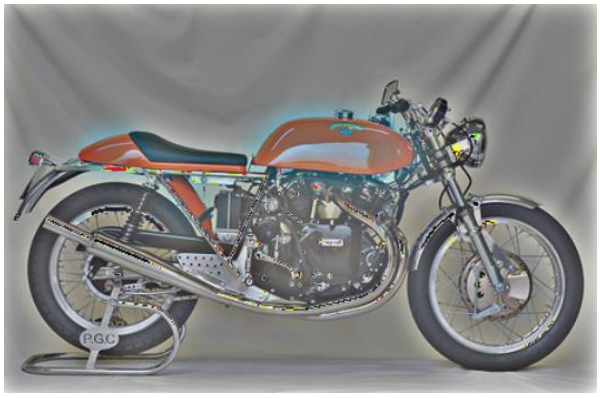
```
# R, G, B 각각을 다시 합친다.
im1b_lowfreq = Image.merge("RGB", (red_1b_lowfreq, green_1b_lowfreq, blue_1b_
im1b_lowfreq.show()
```

```
im1b = np.array(im1b)
im1b_lowfreq = np.array(im1b_lowfreq)
```

```
# 원본 이미지와 lowfreq filter를 이용하여 highfreq filter를 만들어주었다.
im1b_highfreq = im1b - im1b_lowfreq + 128
```

```
im1b_highfreq = Image.fromarray(im1b_highfreq)
im1b_highfreq.show()
```

[Output]



3.

high-pass filter(Motorcycle) + low-pass filter(Bicycle)

[output] sigma 값 - low-pass filter sigma = 0.5,

- high-pass filter sigma = 2

np.array로 변환하여 더해준 뒤에 출력 가능하도록 변환하였다.

```
im1a_lowfreq.show()
```

```
im1b_highfreq.show()
```

```
output = np.array(im1a_lowfreq) + np.array(im1b_highfreq)
output = Image.fromarray(output)
output.show()
```

[Output]