HW 02 – REPORT

소속 : 전기컴퓨터공학부

학번 : 201824469

이름 : 박동진

1. 서론

이미지의 각 픽셀에 커널과의 컨볼루션 연산을 통해서 이미지를 필터링하여 사용자의 의도에 맞게 새로운 이미지를 만들어 낼 수 있다.

이번 과제에서는 파이썬을 이용하여 다양한 커널을 생성하고 이미지에 적용하여 새로운 이미지를 얻고 생성된 이미지를 다양하게 사용하는 실습을 진행한다.

간단한 박스필터와 가우시안 필터를 만드는 함수를 작성하고, 해당 필터를 여러 흑백에서부터 컬러까지 다양한 이미지에 적용해볼 것이다. 최종적으로는 필터가 적용된 이미지를 이용해서 하이브리드 이미지를 만들어 볼 것이다.

과제의 진행을 위해 HW1에서 사용한 라이브러리를 import한다.

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

2.본론

기본적인 커널인 박스필터를 만들어 보는 것으로 시작한다. 일반적으로 커널은 타겟 픽셀을 가운데에 두고 적용되기 때문에 가로와 세로의 크기가 같고 홀수 * 홀수의 행렬의 형태를 띈다. 다양한 박스필터를 만들어 보겠다.

2-1. Gaussian Filtering

2-1-1 `boxfilter(n)`

n * n의 박스필터를 만드는 함수는 아래와 같다. n * n 행렬의 각 엔트리에 1 / n ** 2 한 값을 넣는다.

`np.full((n, n),value)` 함수는 모든 엔트리에 value값이 입력된 n*n배열을 반환하는 함수이다.

```
def boxfilter(n):
    if n <= 0 or n % 2 == 0:  # n이 조건에 맞지 않을 경우 Assertion Erorr를 발생시킵니다.
        raise AssertionError('Dimension must be odd')

    # 각 픽셀에 들어갈 값을 계산한다.
    # 모두 합해서 1이어야 하고, 모든 픽셀의 가중치가 같아야 한다.
    box_entry = 1.0 / n ** 2
    filter = np.full((n, n), box_entry)
    return filter
```

```
print(boxfilter(3))
#print(boxfilter(4))
print(boxfilter(7))
```

문제에서 요구한 각 n에 따른 함수의 결과를 출력한다.

```
(ComputerVision) minmunui@bagdongjin-ui-MacBookAir ComputerVisionPrinciples % python HW2/201824469_ParkDongJin_code.py
[[0.11111111 0.11111111 0.11111111]
 [0.11111111 0.11111111 0.11111111]
 [0.11111111 0.11111111 0.11111111]]
(ComputerVision) minmunui@bagdongjin-ui-MacBookAir ComputerVisionPrinciples %
```

n = 3일 때

```
(ComputerVision) minmunui@bagdongjin-ui-MacBookAir ComputerVisionPrinciples % python HW2/201824469_ParkDongJin_code.py
Traceback (most recent call last):
  File "/Users/minmunui/Desktop/ComputerVisionPrinciples/HW2/201824469_ParkDongJin_code.py", line 14, in <module>
    print(boxfilter(4))
  File "/Users/minmunui/Desktop/ComputerVisionPrinciples/HW2/201824469_ParkDongJin_code.py", line 7, in boxfilter
    raise AssertionError('Dimension must be odd')
AssertionError: Dimension must be odd
(ComputerVision) minmunui@bagdongjin-ui-MacBookAir ComputerVisionPrinciples %
```

n = 4 일 때, 4는 짝수이기 때문에 에러를 발생시킨다.

```
(ComputerVision) minmunui@bagdongjin-ui-MacBookAir ComputerVisionPrinciples % python HW2/201824469_ParkDongJin_code.py
[[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]]
(ComputerVision) minmunui@bagdongjin-ui-MacBookAir ComputerVisionPrinciples %
```

n = 7일 때

2-1-2 `gauss1d(sigma)`

1차원 가우시안 분포를 만든다. 문제에서 주어진 방법을 이용할 것이다.

배열의 크기를 sigma * 6 다음에 오는 가장 작은 홀수로 설정하고, 배열의 가운데를 기준으로 값을 설정한다.

`math.exp(-gaussian[i]**2/(2*sigma**2))` 함수를 적용하여 각 인덱스에 맞는 가우시안 분포를 삽입한다.

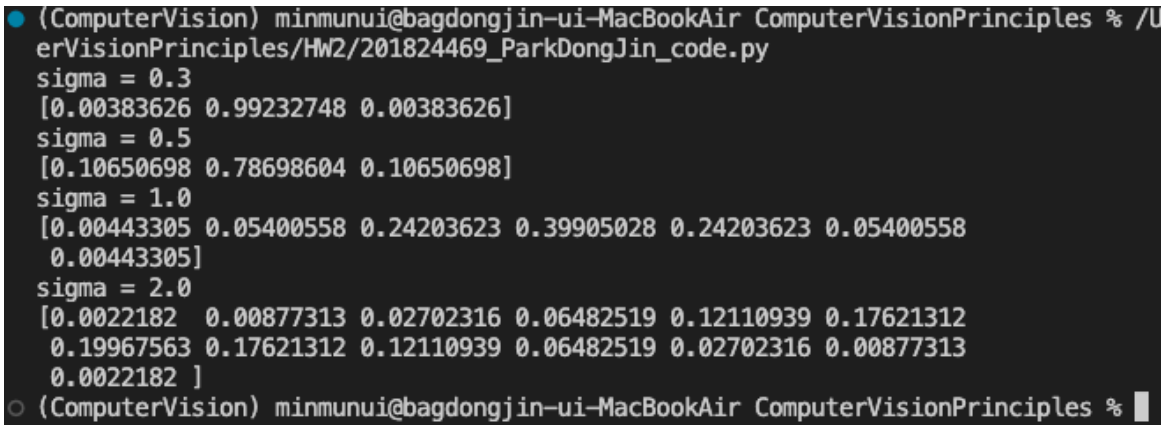
모든 과정이 끝나면 배열 모든 값의 합이 1이 되도록 일반화(normalize)한다.

```
def gauss1d(sigma):
    width: int = int(math.ceil(6 * sigma))  # 6*sigma의 다음 정수로 width를 설정
    if width % 2 == 0:  # 다음 정수가 짝수라면 +1 하여 홀수로 변경
        width += 1

    center = math.floor(width / 2)  # 가운데 index를 획득
    gaussian = np.zeros((1, width)).flatten()  # 반환할 1D Numpy Array 생성
    for i in range(width):
        gaussian[i] = i - center  # Hint:를 토대로 x값을 입력
    for i in range(width):
        gaussian[i] = math.exp(-gaussian[i] ** 2 / (2 * sigma ** 2))  # x값에 따라 gaussian값 획득 및 입력

    # 아래는 normalize, 전체 합으로 각 엔트리를 나눈다.
    kernal_sum = np.sum(gaussian)
    for i in range(width):
        gaussian[i] /= kernal_sum
    return gaussian
```

```
# 결과 출력
print("sigma = 0.3\n{}".format(gauss1d(0.3)))
print("sigma = 0.5\n{}".format(gauss1d(0.5)))
print("sigma = 1.0\n{}".format(gauss1d(1.0)))
print("sigma = 2.0\n{}".format(gauss1d(2.0)))
```



위 코드의 출력, sigma가 작을 수록 뾰족한 분포를 띈다.

2-1-3 `gauss2d(sigma)`

`gauss2d(sigma)` 함수를 만든다. 두 개의 gauss1d 배열을 outer product하면 된다.

```
def gauss2d(sigma):
    return np.outer(gauss1d(sigma), gauss1d(sigma))
# outer product로 2d 가우시안 필터를 구한다.

print("gauss2d(0.5) = \n{}".format(gauss2d(0.5)))
print("gauss2d(1.0) = \n{}".format(gauss2d(1.0)))
```

출력값은 아래와 같다.

```
gauss2d(0.5) =
[[0.01134374 0.08381951 0.01134374]
 [0.08381951 0.61934703 0.08381951]
 [0.01134374 0.08381951 0.01134374]]
gauss2d(1.0) =
[[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]]

Process finished with exit code 0
```

sigma의 값이 클 수록 펑퍼짐한 모양이 된다. 각 entry 값의 합은 1이 된다.

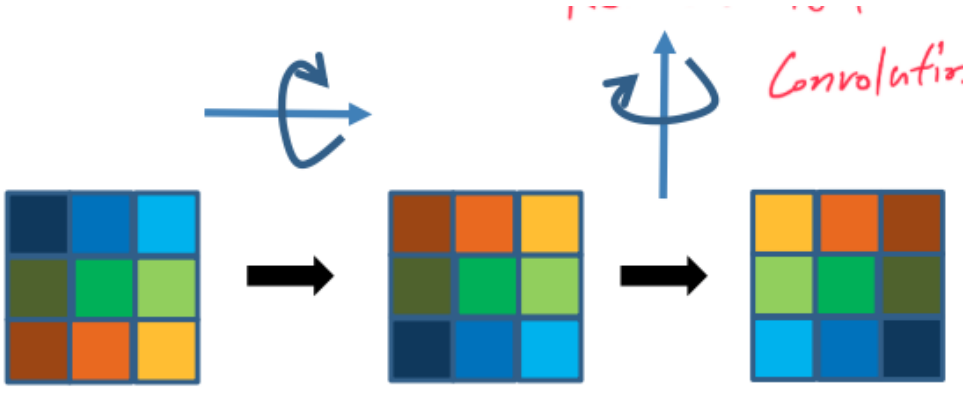
2-1-4

(a) `convolve2d(array, filter)`

이미지를 뜻하는 array의 모든 픽셀에 filter를 적용하는 함수를 만든다.

필터를 적용하기 위해서는 필터의 크기에 따라 이미지 테두리에 zero-padding을 더해 주어야 한다.

패딩이 아닌 부분에는 array의 값을 넣어준다.



convolution 연산을 위해 filter를 각 축에 맞게 뒤집는다.

뒤집어진 flipped_filter 를 array의 각 픽셀에 * 연산한다. numpy.array에 내장된 * 연산을 이용한다.

모든 준비가 완료되면 각 픽셀을 순회하며 컨볼루션 연산을 진행한다. 필터링이 작동하는 것이다.

작업이 끝나면 padding을 덜어낸 부분반 리턴한다.

```
def convolove2d(array, filter):
    padding_width = int(len(filter) / 2)  # filter에 따른 padding 크기
    zero_padding = np.zeros((np.shape(array)[0] + padding_width * 2,
                             np.shape(array)[1] + padding_width * 2))
    # padding한 크기에 맞게 새로운 np.array를 생성

    # convolution연산을 위해서 filter를 축의 방향에 대해서 뒤집는다.
    flipped_filter = np.flip(filter, axis=0)
    flipped_filter = np.flip(flipped_filter, axis=1)
    # 모든 픽셀에 대해서 paading을 제외한 부분에 array를 집어넣어 준다.
    for i in range(padding_width, np.shape(zero_padding)[0] - padding_width):
        for j in range(padding_width, np.shape(zero_padding)[1] - padding_width):
            zero_padding[i][j] = array[i - padding_width][j - padding_width]

    # padding된 이미지의 각 픽셀을 순회하면서 각 픽셀에 컨벌루젼한 값을 입력한다.
    for i in range(padding_width, np.shape(zero_padding)[0] - padding_width):
        for j in range(padding_width, np.shape(zero_padding)[1] - padding_width):
            subpart = zero_padding[i - padding_width:i + padding_width + 1,
                      j - padding_width:j + padding_width + 1]
            zero_padding[i][j] = np.sum(subpart * flipped_filter)
    # 컨벌루젼한 값을 리턴
    return zero_padding[padding_width:-padding_width, padding_width:-padding_width]
```

(b) `gaussconvolve2d(array,sigma)`

위에서 작성한 두 개의 함수를 이용해 이미지에 가우시안 필터를 적용하는 함수를 작성한다.

```
def gaussconvolve2d(array, sigma):
    return convolove2d(array, gauss2d(sigma))
```

(c) 이미지 필터링

작성한 코드를 이용해 greyscale 및 sigma = 3.0의 가우시안 효과 적용

```
# 파일 불러오기
dog_img = Image.open('hw2_image/2b_dog.bmp')
dog_img_grey = dog_img.convert('L')
dog_img_array = np.asarray(dog_img_grey)

#적용할 가우시안 효과의 시그마는 3.0
blurred_dog_array = np.asarray(gaussconvolve2d(dog_img_array, 3.0))
```

(d) 결과 확인

```
# 두 개의 원본과 하나의 결과물 이미지 확인
dog_img.show()
dog_img_grey.show()
Image.fromarray(blurred_dog_array).show()
```



왼 쪽에서부터 원본, 흑백, 가우시안 효과 적용

2-2. Hybrid Images

2-2-1

이미지를 로드하여 배열로 변환한다.

```
# 사용할 이미지를 불러오기
dog_img_img = Image.open('hw2_image/2b_dog.bmp')
cat_img = Image.open('hw2_image/2a_cat.bmp')

# 이미지를 배열로 변환
dog_img_img_array = np.asarray(dog_img_img)
cat_img_array = np.asarray(cat_img)

# 변환된 배열의 shape를 가져온다.
dog_img_height, dog_img_width, rgb = np.shape(dog_img_img_array)
```

각 RGB에 대해서 가우시안 효과를 적용하고 병합한다.

```
# 각 픽셀의 RGB값을 가져온다.
dog_img_img_array_r = dog_img_img_array[:, :, 0].reshape(dog_img_height, dog_img_width)
dog_img_img_array_g = dog_img_img_array[:, :, 1].reshape(dog_img_height, dog_img_width)
dog_img_img_array_b = dog_img_img_array[:, :, 2].reshape(dog_img_height, dog_img_width)

# 각 RGB배열에 가우시안 컨벌루젼을 진행한다.
blur_sigma = 5.0
dog_img_img_array_r_blurred = gaussconvolve2d(dog_img_img_array_r, blur_sigma)
dog_img_img_array_g_blurred = gaussconvolve2d(dog_img_img_array_g, blur_sigma)
```

```
dog_img_img_array_b_blurred = gaussconvolve2d(dog_img_img_array_b, blur_sigma)

# 결과를 저장할 배열을 만든다.
dog_img_img_blurred = np.zeros((dog_img_height, dog_img_width, rgb))

# 각 픽셀의 RGB값을 병합한다.
for row in range(dog_img_height):
    for pixel in range(dog_img_width):
        dog_img_img_blurred[row][pixel] = np.array([dog_img_img_array_r_blurred[row][pixel],
                                                    dog_img_img_array_g_blurred[row][pixel],
                                                    dog_img_img_array_b_blurred[row][pixel]])

# 병합된 이미지를 보여준다.
Image.fromarray(dog_img_img_blurred.astype(np.uint8)).show()
```

위와 같은 결과가 나온다. 개의 저주파이미지이다.

2-2-2

각 RGB배열에 가우시안 효과를 적용한 값을 원본에서 빼서 고주파이미지를 얻어낸다.

```
# 변환된 배열의 shape를 가져온다.
cat_height, cat_width, rgb = np.shape(cat_img_array)

# 각 픽셀의 RGB값을 가져온다.
cat_img_array_r = cat_img_array[:, :, 0].reshape(cat_height, cat_width)
cat_img_array_g = cat_img_array[:, :, 1].reshape(cat_height, cat_width)
cat_img_array_b = cat_img_array[:, :, 2].reshape(cat_height, cat_width)

# 각 RGB에 대해서 가우시안 효과 적용
sharp_sigma = 5.0
cat_img_array_r_blurred = gaussconvolve2d(cat_img_array_r, sharp_sigma)
cat_img_array_g_blurred = gaussconvolve2d(cat_img_array_g, sharp_sigma)
cat_img_array_b_blurred = gaussconvolve2d(cat_img_array_b, sharp_sigma)

# 결과물을 저장할 배열을 만든다.
cat_img_sharpened = np.zeros((dog_img_height, dog_img_width, rgb))
```

```
# 각 픽셀에 원본에서 가우시안효과를 뺀 값을 저장한다.
for row in range(cat_height):
    for pixel in range(cat_width):
        cat_img_sharpened[row][pixel] = np.array([cat_img_array_r[row][pixel]
                                                  - cat_img_array_r_blurred[row][pixel],
                                                  cat_img_array_g[row][pixel]
                                                  - cat_img_array_g_blurred[row][pixel],
                                                  cat_img_array_b[row][pixel]
                                                  - cat_img_array_b_blurred[row][pixel]])
```

모든 픽셀에 128을 더해서 결과물을 보여준다. 음숫값을 처리하기 위함이다.

```
# 고주파 이미지를 보여주기 위한 보정값 128을 각 픽셀에 더한 후 이미지 보여주기
cat_img_show = cat_img_sharpened + 128
Image.fromarray(cat_img_show.astype(np.uint8)).show()
```

위와 같은 결과가 나온다. 고양이의 고주파이미지이다.

2-2-3

두 개의 이미지를 합친다. 오버플로를 막기 위해 합을 2로 나눈다.

```
# 두 개의 이미지를 더한다. 픽셀 값의 오버플로를 막기 위해 2로 나눈다.
hybrid = (cat_img_sharpened + dog_img_img_blurred)/2
Image.fromarray(hybrid.astype(np.uint8)).show()
```

위와 같은 결과가 나온다.

3. 결론

가우시안필터를 생성하여 컨벌루전 연산을 통해 간단한 이미지 필터링을 해 보았다. 가우시안 분포를 이용한 가우시안 효과는 주변 픽셀의 값을 참조하여 전체적인 이미지를 흐릿하게 만든다. 편차를 의미하는 시그마의 값이 클 수록 이러한 흐릿함은 더 강해진다. 이렇게 디테일이 사라지고 흐릿한 이미지를 저주파 이미지라고 한다. 반대로 디테일한 부분은 고주파이미지라고 한다.

사람이 이미지를 크게 보면 자세한 것에 집중하게 되어 고주파의 이미지를 인식하게 된다. 반대로 이미지를 작게 보면 전체적인 형태에 집중하게 되어 저주파의 이미지를 인식하게 된다. 서로 다른 이미지의 고주파와 저주파부분을 함쳐 생성한 이미지를 하이브리드이미지라고 한다. 하이브리드이미지를 가까이서 보면 고주파이미지의 사물이 보이고 멀리서 보면 저주파이미지의 사물로 보인다. 사람이 이미지를 인식하는 경향을 이용한 것이다.

이번 과제를 통해 가까이서 보면 오토바이가 보이고 멀리서 보면 자전거가 보이는 하이브리드이미지를 제작했다.