HW 02 – REPORT

소속 : 정보컴퓨터공학부

학번 : 201925111

이름 : 김건호

1. 서론

2024 컴퓨터비전개론 두 번째 과제에 대한 보고서입니다. Image filtering에 대한 이해를 높이기 위해, box filter(=mean filter)와 Gaussian filter를 직접 구현하고, image convolution을 수행합니다. 이를 통해, image blurring을 이해할 수 있습니다. 두 번째로, gaussian filter를 사용해서 low-frequency image와 high-frequency image를 만들 수 있습니다. 이를 합성하여, low-frequency와 high-frequency의 특징을 둘 다 갖는 새로운 hybrid image를 만들 수 있습니다.

Mean filter란 image의 특정 pixel에서 filter의 size만큼의 값들을 추출해서 이를 평균으로, 새로운 image를 만드는 filter를 말합니다. Mean filter는 특정 이웃수((filter의 size-1)//2, filter의 size는 홀수여야 합니다.)에 기반하여, 모두 동일한 가중치를 두어, 새로운 image를 생성합니다. 하지만 이는 pixel의 특정 이웃의 값이 매우 크다면, 새로운 image가 특정 이웃에 많은 영향을 받습니다. 이러한 단점을 보완하기 위해 거리와 sigma값에 기반하여 만들어진 filter가, gaussian filter입니다. Gaussian filter는 gaussian 분포를 기반으로 구현됩니다. 두 가지 filter를 사용해서 image의 noise를 줄이고, image blurring 및 smoothing 효과를 얻을 수 있습니다.

Filter를 사용해서 image를 보는 방식에는, cross-correlation과 convolution이 존재합니다. Cross-correlation은 image의 위치와 filter의 해당 위치를 대응시켜, 새로운 이미지를 만듭니다.

- Cross correlation

$$S[f] = w \otimes f$$

$$S[f](m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m+i, n+j)$$

Convolution은 180도 회전시킨 filter를 사용해서 cross-corelation합니다. Convolution을 하는 이유는, filter와 유사한 image의 convolution 결과값은 높고, 유사하지 않다면 결과값이 낮기 때문입니다.,

- Convolution

$$S[f] = w * f$$

$$S[f](m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m-i, n-j)$$

Gaussian filter를 사용해서 Image convolution한 결과는 원래 image를 blurring한 결과입니다. 원래 image에서, convolution된 새로운 image를 뺀다면, image의 high-frequency를 추출한 새로운 image를 얻을 수 있습니다. 이는 image를 sharpnig한 결과와 동일합니다. 이 두가지 이를 통해 hybrid image를 생성할 수 있습니다.

Gaussian filter에서 sigma가 커지면, 특정 pixel이 주변의 이웃에 더 많은 영향을 받아지기에, smoothing/blurring 효과가 강해집니다. 이는 sigma가 커진다면, image의 더 낮은 frequency를 얻을 수 있습니다. 이를 토대로, 더 높은 frequency의 image를 생성할 수 있습니다.

2. 본론

Part1-1 :크기가 n인 box filter를 만듭니다. N이 홀수가 아니라면 error message를 출력합니다.

```
def boxfilter(n) :
    if n%2==0 :
        raise AssertionError('Dimension must be odd')
    else :
        box_filter=[ [1/(n**2) for _ in range(n)] for _ in
range(n)]
        return np.array(box_filter)
```

n=3 결과)

```
 [[0.11111111 0.11111111 0.11111111]
  [0.11111111 0.11111111 0.11111111]
  [0.11111111 0.11111111 0.11111111]]
```

n=4 결과)

```
Traceback (most recent call last):
  File "c:\python_workspace\2024_cv\hw2\hw2.py", line 14, in <module>
    boxfilter(n=4)
  File "c:\python_workspace\2024_cv\hw2\hw2.py", line 8, in boxfilter
    raise AssertionError('Dimension must be odd')
AssertionError: Dimension must be odd
```

n=7 결과)

```
[[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]]
```

part1-2 : 1차원 gaussian filter 만들기.

```
def gauss1d(sigma):

    #sigma 에 6 배를 한 후, 반올림합니다.
    round_up_sigma = round(sigma*6)

    #만약 짝수라면 1 을 더해 홀수로 만듭니다.
    #round_up_sigma = round_up_sigma if round_up_sigma%2 else round_up_sigma+1
    if round_up_sigma%2==0 :
        round_up_sigma+=1

    #map + lambda function 과 list comprehension 을 사용해서 1 차원 배열을 만듭니다.
    arr=np.array([i for i in range(-(round_up_sigma//2), 1+ (round_up_sigma//2))])
    array_1d = np.array(list(map(lambda x: 1/math.sqrt(2*math.pi*sigma**2) * math.exp(-(x**2)/(2*sigma**2)),
arr)))

    #합이 1 이 되도록 정규화합니다.
    normalization_array_1d=array_1d/sum(array_1d)
    return normalization_array_1d
```

결과)

```
 sigma=0.3  [0.00383626 0.99232748 0.00383626]
 sigma=0.5  [0.10650698 0.78698604 0.10650698]
 sigma=1  [0.00443305 0.05400558 0.24203623 0.39905028 0.24203623 0.05400558
  0.00443305]
 sigma=2  [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 ]
```

part1-3 : 2차원 gaussian filter 만들기

```
# part1-3 : 2 차원 gaussian filter 만들기
def gauss2d(sigma) :
    # sigma 값을 입력으로, 1 차원 gaussian filter 를 만들고, np.outer 과 np.transpose 를 통해 2 차원 gaussian
filter 를 만듭니다.
    # np.outer 후 결과값의 합은 어차피 1 일 것이므로, 따로 정규화해줄 필요는 없습니다.
    # 왜냐하면, 모든 원소의 합이 1 인 이미 정규화된 1 차원 gaussain filer 에, transpose 후 outer product 한
    # 2 차원 gaussian filer 의 모든 원소의 합은 1 일 것이기 때문입니다.
    array_1d=gauss1d(sigma)
    return np.outer(array_1d, np.transpose(array_1d))
```

결과)

```
 sigma=0.5  [[0.01134374 0.08381951 0.01134374]
  [0.08381951 0.61934703 0.08381951]
  [0.01134374 0.08381951 0.01134374]]
 sigma=1  [[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]]
```

part1-4-(a) : array(=image)와 filter를 사용해서 image convolution 하기

```
def convolve2d(array, filter) :

    #input 변수의 dtype 을 np.float32 로 변경합니다.
    array=array.astype(np.float32)
    filter=filter.astype(np.float32)
    #convoultion_array 를 list comprehension 으로 만듭니다.
    height=len(array)
    width=len(array[0])
    convolution_array = np.array([[0 for _ in range(width)] for _ in range(height)] ,dtype=np.float32)
    # padding size 를 계산 후, array 에 padding 을 추가합니다.
    padding_size=int((len(filter[0])-1)/2)
    padding_array = np.pad(array, ((padding_size, padding_size), (padding_size, padding_size)), 'constant',
constant_values=0)
    # convolution 계산의 편의성을 위해 filter 를 flip 을 합니다.
    filter_size=len(filter)
    flipped_filter=np.flip(filter)
    # pad 가 추가된 2 차원 image 에서 filter_size 만큼 crop 후, flip 된 filter 로 cross-correlation 을 진행합니다.
    # filter 를 flip 후, cross-correlation 한 결과는 convolution 한 결과와 동일합니다.
    for row in range(height) :
        for col in range(width) :
            crop_image=padding_array[row:row+filter_size, col : col+filter_size]
            convolution_array[row][col]=np.sum(crop_image * flipped_filter)

    return convolution_array
```

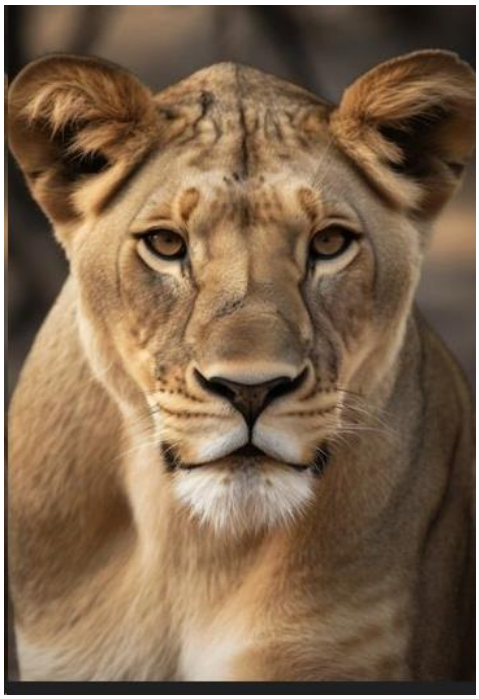
part1-4-(b) : sigma에 해당하는 2d gaussian filter를 만든 후, array(=image)와 convolution

```
def gaussconvolve2d(array,sigma) :
    gaussian_filter=gauss2d(sigma)
    return convolve2d(array, gaussian_filter)
```

part1-4-(c), (d) : sigma=3으로 tiger gaussian convolution하기.

```
image1 = Image.open('./hw2/images/3b_tiger.bmp')
image1.show()
image1=np.asarray(image1.convert('L')) # 흑백으로 전환 후, nparray 로 변환
convolution_tiger=gaussconvolve2d(image1, 3).astype('uint8') # gaussian convolution 후, 정수 값으로 변환
tiger = Image.fromarray(convolution_tiger)
tiger.show()
tiger.save('part1-4-after.png' , 'PNG') #저장
```

결과) 전 / 후



part2-1 : gaussian filter(=low pass filter) 를 사용해서 image blurring하기. image의 high-frequency를 제거하는 효과와 같습니다. (=low-frequency만 남기는 효과와 같습니다.) low-frequency만 남기는 이미지로 mangosteen을 선택했습니다. Sigma 값은 5를 선택했습니다.

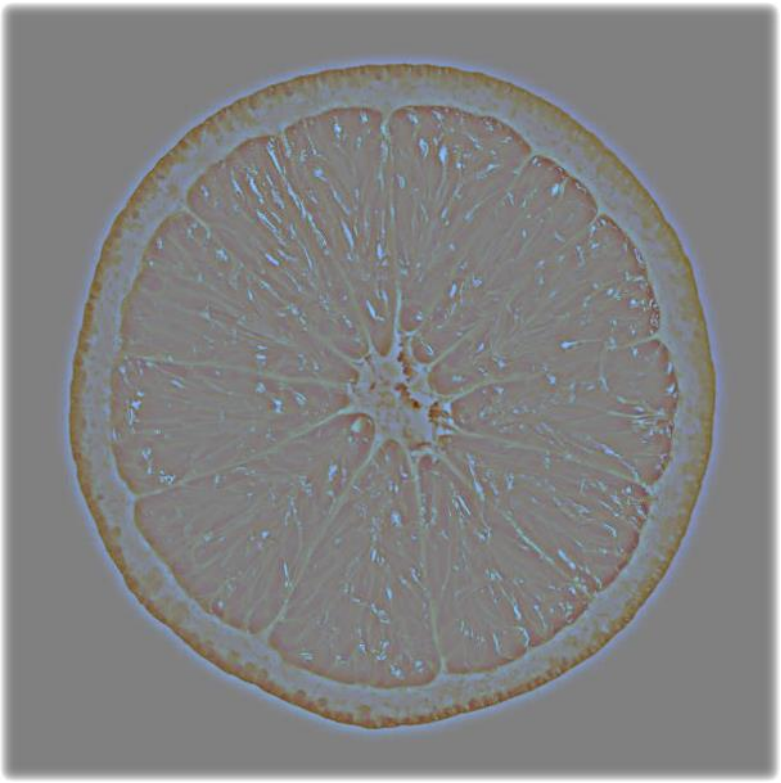
```
image2=Image.open('./hw2/images/2a_mangosteen.bmp')
image2.show()
r,g,b=image2.split() # split method 를 사용해서 간단하게 rgb channel 을 추출할 수 있습니다.
image2_r, image2_g, image2_b = np.asarray(r) , np.asarray(g), np.asarray(b)

# r,g,b channel 을 sigma=2 인 2d-gaussian filter 과 convolution 합니다.
image2_convolved_r=gaussconvolve2d(image2_r, 5)
image2_convolved_g=gaussconvolve2d(image2_g, 5)
image2_convolved_b=gaussconvolve2d(image2_b, 5)

# unit8 type 으로 변환 후, merge method 를 통해, low-frequency image 를 생성합니다.
image2_result_r, image2_result_g, image2_result_b = image2_convolved_r.astype('uint8'),
image2_convolved_g.astype('uint8'), image2_convolved_b.astype('uint8')
new_r, new_g, new_b = Image.fromarray(image2_result_r), Image.fromarray(image2_result_g),
Image.fromarray(image2_result_b)
new_image2 = Image.merge('RGB', (new_r, new_g, new_b))
new_image2.show()
```

결과)

part2-2 : high-frequency image 만들기. 원래 image에서 low-frequency를 뺀 결과는 high-frequency image입니다. high frequency만 남기는 이미지로 orange를 선택했습니다. Numpy의 minimum method를 활용하여 255 초과인 값을 보정했습니다.

```
image3=Image.open('./hw2/images/2b_orange.bmp')
image3.show()
r,g,b=image3.split() # split method 를 사용해서 간단하게 rgb channel 을 추출할 수 있습니다.

# r,g,b channel 을 sigma=2 인 2d-gaussian filter 과 convolution 합니다.
image3_r, image3_g, image3_b = np.asarray(r), np.asarray(g), np.asarray(b)
blur_r = gaussconvolve2d(image3_r, 5)
blur_g = gaussconvolve2d(image3_g, 5)
blur_b = gaussconvolve2d(image3_b, 5)

# 원본 이미지에서 gaussian filter 와 convolution 된 이미지를 빼서, image 의 high-frequency 만 남깁니다.
image3_result_r = image3_r - blur_r
image3_result_g = image3_g - blur_g
image3_result_b = image3_b - blur_b

# 128 을 더해서, 음수 값을 보정합니다.
add_result_r = image3_result_r + 128
add_result_g = image3_result_g + 128
add_result_b = image3_result_b + 128

# 255 초과인 값을 보정합니다.
modified_result_r = np.minimum(add_result_r, 255)
modified_result_g = np.minimum(add_result_g, 255)
modified_result_b = np.minimum(add_result_b, 255)

# unit8 type 으로 변환 후, merge method 를 통해, high-frequency image 를 생성합니다.
modified_result_r, modified_result_g, modified_result_b = modified_result_r.astype('uint8'),
modified_result_g.astype('uint8'), modified_result_b.astype('uint8')
new_r, new_g, new_b = Image.fromarray(modified_result_r), Image.fromarray(modified_result_g),
Image.fromarray(modified_result_b)
new_image2 = Image.merge('RGB', (new_r, new_g, new_b))
new_image2.show()
```

결과)



part2-3 : low-frequency image와 high-frequency image 합성하기. 기존에 구했던, low-frequency image와 high-frequency image를 더한 후, 값을 보정해서 만듭니다.

```
hybrid_r = image2_result_r + image3_result_r
hybrid_g = image2_result_g + image3_result_g
hybrid_b = image2_result_b + image3_result_b

# np.clip 을 이용해서 값을 보정합니다.
hybrid_r_corrected = np.clip(hybrid_r, 0, 255)
hybrid_g_corrected = np.clip(hybrid_g, 0, 255)
hybrid_b_corrected = np.clip(hybrid_b, 0, 255)

# unit8 type 으로 변환 후, merge method 를 통해, hybrid image 를 생성합니다.
hybrid_r_corrected, hybrid_g_corrected, hybrid_b_corrected = hybrid_r_corrected.astype('uint8'),
hybrid_g_corrected.astype('uint8'), hybrid_b_corrected.astype('uint8')
new_r, new_g, new_b = Image.fromarray(hybrid_r_corrected), Image.fromarray(hybrid_g_corrected),
Image.fromarray(hybrid_b_corrected)
new_image3 = Image.merge('RGB', (new_r, new_g, new_b))
new_image3.show()
```

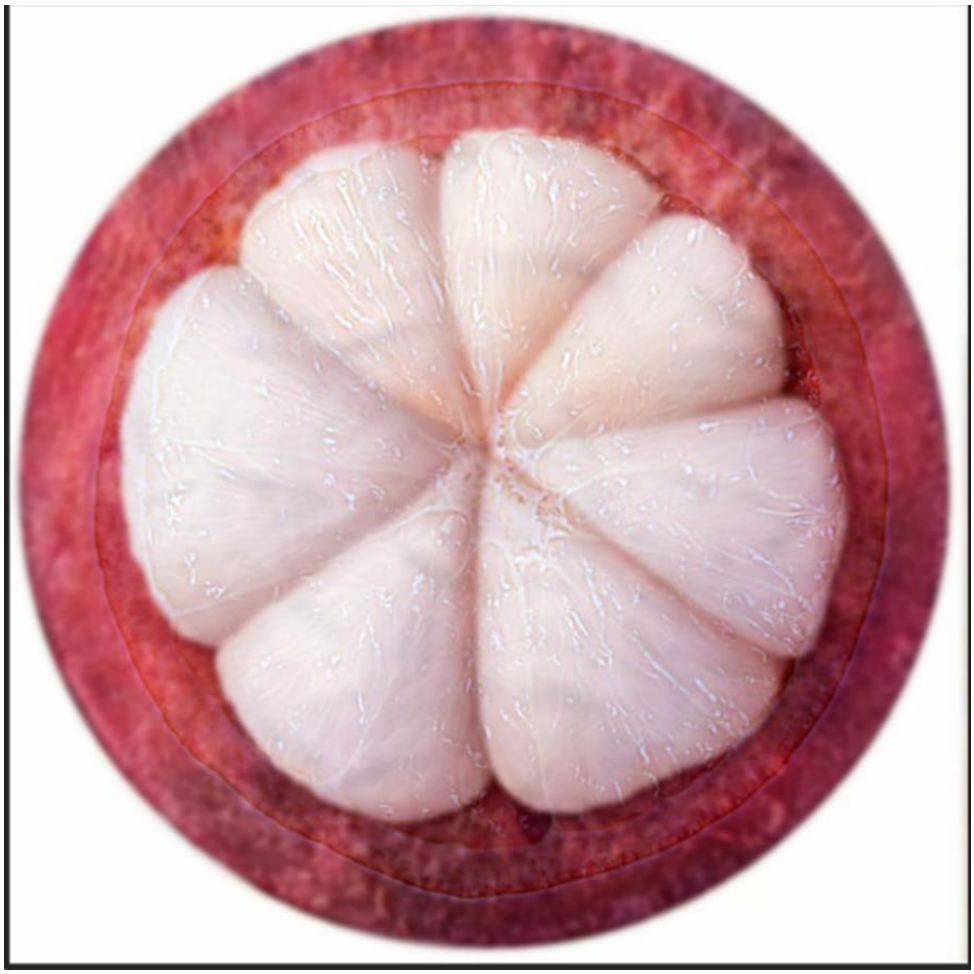
결과)

3. 결론

Numpy library를 통해 Mean filter (= box filter), Gaussian filter를 구현할 수 있었습니다.

PIL library를 통해 불러온 image를 직접 구현한 filter를 사용해서 convolution한 image를 얻을 수 있었습니다. 이를 통해 low-frequency, high-frequency image를 얻고, 두 image를 hybrid한 image를 얻을 수 있었습니다.

Part2에서 sigma값을 5로 설정하여서 얻은 이미지에 비해, sigma값을 2로 설정한다면 이미지의 blurring이 기존보다 덜 해지고, 이를 통해 기존 이미지에 비해, 더 낮은 high-frequency image를 얻을 수 있습니다. 이를 통해 얻은 hybrid image는 기존의 image보다 덜 blurring되고, 덜 sharpning된 결과를 얻을 수 있을 것이라 예상됩니다.



< sigma=2로 만든 hybrid image>

또한, gaussian filter를 사용하는 큰 이유는, image의 linearity를 해치지 않기 때문입니다. Linearity가 보존되기에, 우리는 예상 가능한 결과를 얻을 수 있습니다. 하지만 linearity가 보존되지 않는 filter가 있어야 하는 경우도 존재합니다. 예시로, mean filter는 speckle noise가 존재하는 상황에서는 noise를 잘 줄여주지 못합니다. Non-linear filter는 특정 임계값을 기준으로 흑,백을 구분하는 thresholding filter와, 특정 이웃들의 중앙값을 기준으로 새로운 image를 만드는 median-filter가 존재합니다. 이와 같은 filter를 사용해서 새로운 image를 만들 수 있습니다.