HW 02 – REPORT

소속 :

학번 :

이름 :

1. 서론

실습 목표 및 이론적 배경 기술 (1~2페이지)

실습 목표는 파이썬을 이용하여 가우시안 필터 함수를 만들고, 이 함수를 이용하여 hybrid image를 만드는 것이다.

필터링(Filtering)은 원래 픽셀들을 조합하여 새로운 이미지를 만드는 것이다. 필터링을 이용하여 edge 검출이나 스무딩 연산(edge를 뭉개는 효과), 샤프닝 효과(edge를 강조하는 효과)를 구현할 수 있다.

필터링 중에서 Linear filtering은 주변 픽셀에 가중치를 곱한 후 단순히 더하여 해당 픽셀의 값을 계산하는 것이다. 이때 적용하는 가중치를 커널(또는 마스크, 필터)이라고 한다.

이때 이미지에 커널을 적용하는 방법은 2가지가 있는데, cross corelation와 convolution이 있다. 필터의 중앙을 이미지 중 계산하고자 하는 픽셀에 맞추고 서로 대응되는 이미지 화소 – 커널 가중치끼리 곱하여 더하는 것은 같다. 다만 cross corelation은 커널을 그대로 적용하는데 반해서, convolution은 커널을 뒤집는다는 것(좌우, 상하 방향으로)에서 다르다.

커널을 사용하여 계산할 때, 가장자리 처리에 주의하여야한다. 이미지에서 가장자리 값은 중간에 있는 값보다 필터가 덜 적용되고, 최종적으로 처리된 이미지는 원래 이미지보다 크기가 작아진다. 이를 방지하기 위해 padding, 즉 이미지 가장자리에 0 등의 값으로 픽셀을 채운다. 여기서는 fxf 크기의 커널이 있으면 m=(f-1)/2 만큼 가장자리를 확장한다.

Cross-corelation와 convolution을 식으로 표현하면 다음과 같다.

Cross-corleation : $S[f](m,n) = w \otimes f = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m+i, n+j)$ (1.1)

Convolution : $S[f](m,n) = w * f = \sum_{i=-k}^{k} \sum_{j=-k}^{k} w(i,j) f(m-i, n-j)$ (1.2)

그리고 커널의 종류로는 mean filter와 gaussian filter가 있다. Mean filter는 모든 픽셀의 가중치를 동일하게 설정하는 것이다(단, 가중치 합이 1). 그러나 통상적으로 이미지에서 어떤 픽셀은 멀리 있는 픽셀보다 가까운 픽셀과 더 연관이 있으므로, 가까운 픽셀에 더 가중치를 둘 수 있다. 이런 방법 중 하나가 gaussian filter이다.

Gaussian filter에 적용되는 식은 아래와 같다.

$$G_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \quad (1.3)$$

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (1.4)$$

Gaussian 식은 절대값이 0일 때 가장 크고, 절대값이 0에서 멀어질수록 작아진다.

여기서는 1차원 가우시안 필터를 만들 때, sigma 값에 따라 필터의 크기를 달리한다. Sigma에 6을

곱한 값에 다음 홀수를 필터의 크기로 한다. 그 다음 필터의 크기를 n이라고 하면, 필터에 적용되는 x값(식 1.3 참고)은 $a = \lfloor n/2 \rfloor$ 에서 (-a,-a+1,-a+2, … , 0, 1, 2, … , a)이다.

2차원 가우시안 필터는 생성한 1차 가우시안 필터를 외적하여 구할 수 있다(식 1.5, 식 1.6 참조).

식 1.5에서 2차원 가우시안 함수는 x방향 가우시안 함수와 y방향 가우시안 함수의 곱임을 알 수 있다. 외적의 크기가 각 벡터의 크기에 sinθ 를 곱한 것인데, x방향과 y방향 사이의 각도는 90도이므로(sin 90° = 1) 외적을 통하여 $G_\sigma(x,y)$ 를 구할 수 있다.

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{y^2}{2\sigma^2}} = G_\sigma(x)G_\sigma(y) \quad (1.5)$$

$$|\vec{v} \times \vec{w}| = |v||w| \sin\theta \quad (1.6)$$

이미지에 가우시안 필터를 적용하는 것을 스무딩 연산이라고도 한다. 왜냐하면 주변 픽셀을 이용하여 해당 픽셀을 계산하므로 해당 픽셀과 주변 픽셀의 차이가 적어지기 때문이다. 따라서 가우시안 필터를 적용한 이미지는 low-frequency하다라고도 한다. 원래 이미지 픽셀에서 가우시안 필터를 적용한 이미지의 픽셀값을 빼면, 원래 이미지에서 원래 low-frequency였던 부분은 사라지고, high-frequency, 즉 변화가 많은 부분만 남게 된다. 따라서 이를 이용해 high-frequency 이미지를 만들 수 있다(원래 이미지에서 high-frequency 이미지를 더하면 원래 이미지에서 high-frequency 부분이 강조되어 sharpening 효과를 줄 수 있다). 그리고 low-frequency 이미지와 high-frequency 이미지를 합치면, 윤곽은 low-frequency 이미지로, 상세한 부분은 high-frequency 이미지로 보이는 이미지를 만들 수 있다.

2. 본론

실습 내용 및 결과 기술 (2페이지 이상)

Part 1. Gaussian Filtering

1-1. boxfilter

```python
#1-1
def boxfilter(n):
    assert n%2!=0, "Dimension must be odd"
    return np.ones((n,n)) / (n*n)
```

그림 1 boxfilter 구현 코드

그림1은 boxfilter, 즉 필터 내의 픽셀의 가중치가 모두 같은 필터이다. 먼저 필터의 가로(및 세로) 길이가 홀수인지 검사하고, 홀수가 아니라면 에러를 출력한다. 홀수라면, 값이 모두 1인 n*n 크기의 넘파이 배열을 만든 후, 이를 n*n으로 나누어(broadcasting 연산) 각 항목의 값이 1/(n*n)이 되도록 구현하였다. 실행결과는 그림2에 나와있다.

```
Python 3.11.7 | packaged by Anaconda, Inc. | (main, Dec 15 2023, 18:05:47) [MSC
v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

================== RESTART: D:\강의자료\4학년 1학기\컴퓨터비전개론\과제2\hw2.py ====
=============
boxfilter(3)
array([[0.11111111, 0.11111111, 0.11111111],
       [0.11111111, 0.11111111, 0.11111111],
       [0.11111111, 0.11111111, 0.11111111]])
boxfilter(4)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    boxfilter(4)
  File "D:\강의자료\4학년 1학기\컴퓨터비전개론\과제2\hw2.py", line 9, in boxfilter
    assert n%2!=0, "Dimension must be odd"
AssertionError: Dimension must be odd
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]])
```

그림 2 boxfilter 함수 실행 결과. 매개변수가 짝수일 때는 오류를, 홀수 일 때는 각 픽셀의 가중치가 같은 boxfilter를 생성한다.

1-2. gauss1d

```
#1-2
def gauss1d(sigma):
    #Calculate the filter length(rounded up to the next odd integer)
    len = sigma * 6
    len = math.ceil(len)

    if(len%2==0):
        len=len+1 # if len is even, make it odd

    #generated an array of values of distance from center
    x = np.arange(len//2 * -1, len//2 + 1,1)
    #compute gaussian value
    gaussian = np.exp(-1*x*x/(2*sigma*sigma))
    #normalize the filter so that the sum of values is 1
    return gaussian / np.sum(gaussian)
```

그림 3 1차원 가우시안 필터를 생성하는 gauss1d 함수

그림3은 1차원 가우시안 필터를 생성하는 gauss1d 함수이다. 먼저 가우시안 필터의 크기는 입력받은 sigma에 따라 달라진다. 가우시안 필터의 크기는 입력받은 sigma에 1.6을 곱한 후, 그 값의 next odd integer로 한다. 이를 구현하기 위해 sigma*1.6 값을 올림한 후, 그 값이 홀수이면 그대로 쓰고, 짝수이면 1을 더하여 next odd integer를 계산하였다.

그 다음 gaussian 함수에 적용할 x값(식 1.3 참조)을 생성한다. 이 x값은 1차원 배열의 중앙에서 얼마나 멀리 떨어져있나를 판단한다. 이를 구현하기 위해 np.arange함수를 사용하였다. 이 함수는 최소값, 최대값, step size를 입력받아 [최소값, 최소값 + step size, 최소값 + 2*step size, … , 최대값](단, step size에 따라 마지막 값이 최대값 이하일 수 있음)을 생성한다.

생성한 x값 넘파이 배열에 식 1.3을 적용해 1차원 가우시안 필터를 생성한다. 그리고 생성한 가우시안 필터 각 항목의 값의 합이 1이되게 하기 위하여 각 항목을 가우시안 필터의 합으로 나누어 합을 1로 만들어주었다.

Gauss1d의 실행결과는 그림4에 나와있다.

```
Python 3.11.7 | packaged by Anaconda, Inc. | (main, Dec 15 2023, 18:05:47) [MSC
v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: D:\강의자료\4학년 1학기\컴퓨터비전개론\과제2\hw2.py
>>> 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 ])
>>>
```

그림 4 gauss1d 실행결과

1-3. gauss2d

```
def gauss2d(sigma):
    #compute 1-dimension gaussian filter
    gaussian_1d = gauss1d(sigma)
    #cross product of 1-dimension gaussian filter
    gaussian_2d = np.outer(gaussian_1d,gaussian_1d)
    #normailze the filter so that the sum of values is 1
    return gaussian_2d / np.sum(gaussian_2d)
```

그림 5 gauss2d 함수

그림5는 gauss2d 함수, 즉 2차원 가우시안 필터를 생성하는 함수이다. 2차원 가우시안 필터는 1-2

에서 생성한 1차원 가우시안 필터 2개(적용되는 sigma는 같음)를 외적하여 구할 수 있다. 외적연산 후, normalizing 연산을 수행하여 각 항목의 합이 1이되도록 하였다. 그림6은 함수 실행 결과이다.

```
Python 3.11.7 | packaged by Anaconda, Inc. | (main, Dec 15 2023, 18:05:47) [MSC
v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: D:\강의자료\4학년 1학기\컴퓨터비전개론\과제2\hw2.py
>>> 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]])
>>>
```

그림 6 gauss2d 함수 실행 결과

1-4. apply Gaussian filter to image

(a) convolve2d(array, filter)

```python
def convolve2d(array, filter):
    #assume that row size and column size of filter is same

    #make empty image
    result = np.zeros_like(array)
    #get filter size(for calculating padding size)
    filter_row_size = filter.shape[0]
    #calculating padding size
    pad_size = int((filter_row_size-1)/2)
    #applying padding to array(image)
    pad_array = np.pad(array, ((pad_size,pad_size),(pad_size,pad_size)), 'constant', constant_values=0)

    #flip filter based on the axis=0 and axis=1(for convolution)
    flip_filter = np.flip(filter)

    #perform convolution
    for i in range(array.shape[0]):
        for j in range(array.shape[1]):
            result[i,j] = np.sum(pad_array[i:i+filter_row_size,j:j+filter_row_size]*flip_filter)

    return result
```

그림 7 convolve2d 함수

Convolve2d 함수는 이미지에 필터를 convolution 연산을 통해서 적용시키는 함수이다. Convolution 연산을 한 배열을 저장할 배열(result)을 하나 만들고, 각 픽셀에 필터가 동일한 횟수로 적용되게 하기 위하여 padding을 적용한다. Padding 크기는 (필터 열(또는 행)길이 – 1)//2 이다.

Convolution 연산을 위해서는 필터를 뒤집어야 하므로, 필터를 뒤집는다(행과 열 방향으로 모두 뒤집음). 그리고 각 이미지 픽셀에 필터를 적용시켜 convolution 연산을 수행하고 이를 result 배열에 저장한다.

(b) gaussconvolve2d(array, sigma)

```
def gaussconvolve2d(array,sigma):
    #make gaussian filter using sigma
    filter = gauss2d(sigma)
    #perform convolution
    return convolve2d(array,filter)
```

그림 8 gaussconcolve2d(array, sigma) 함수

그림8은 gaussconvolve2d 함수를 구현한 것이다. 이 함수는 이미지에 가우시안 필터를 적용시키는 함수이다. 먼저 입력받은 sigma로 가우시안 필터를 만들고, 이를 convolution 연산을 통해 이미지에 적용시킨다.

(c), (d)

```
#c

im = Image.open('images/3b_tiger.bmp')
im_gray = im.convert('L')
im_array = np.asarray(im_gray)

convolved = gaussconvolve2d(im_array,3)
convolved = np.clip(convolved, 0, 255, out=convolved)
convolved = convolved.astype(np.uint8)
#d
im.show()
convolved_image = Image.fromarray(convolved)
convolved_image.show()
convolved_image.save("part1_result.bmp","bmp")
```

그림 9 이미지를 흑백 모드로 바꾸고, sigma=3의 가우시안 필터를 적용시켜 저장하고 보여주는 코드

여기에서는 호랑이 사진에 가우시안 필터를 적용해보았다. PIL의 image 라이브러리를 사용하여 호랑이 사진을 열고, 이를 흑백으로 전환하고, 넘파이 배열로 바꾸어 convolution 연산을 적용하였

다. 필터가 적용된 배열을 다시 이미지로 바꾸고, 원래 이미지와 필터가 적용된 이미지를 보여주고 필터가 적용된 이미지를 저장하였다.

그림 10 그림9 코드 실행 결과. 호랑이 사진이 흑백으로 바뀌고, 가우시안 필터가 적용되어 흐릿해졌다.

Part2 Hybrid image

```
#apply gauss filter to RGB image(RGB array)
def gaussconvolved2dRGB(array,sigma):
    #numpy array that store the result image array
    low_frequency_result = np.zeros_like(array)
    # filter each of the three color channels (RGB) separately
    for i in range(3):
        low_frequency_result[:,:,i] = gaussconvolve2d(array[:,:,i],sigma)
        low_frequency_result[:,:,i] = np.clip(low_frequency_result[:,:,i], 0, 255)
    return low_frequency_result
```

그림 11 RGB 이미지에 가우시안 필터를 적용하는 함수

그림10은 RGB 이미지에 가우시안 필터를 적용하는 함수이다. 결과를 저장할 배열을 하나 만들고, 원래 이미지 배열을 R, G, B 따로 슬라이싱하여 각각 가우시안 함수를 적용하여 저장하였다. 그리고 0미만, 255 초과 값은 각각 0과 255로 설정하였다.

2-1. Applying gaussian filter to RGB image

```
#2-1
#open image(for making low frequency image)
im2 = Image.open('images/3b_tiger.bmp')

#make array from image
im2_array = np.asarray(im2)

low_frequency_result = gaussconvolved2dRGB(im2_array,4)

# convert result array to image, show and save
low_frequency_img = Image.fromarray(low_frequency_result)
low_frequency_img.show()
low_frequency_img.save("part2-1_result.bmp","bmp")
```

그림 12 Applying gaussian filter to RGB image, show and save image

그림 11은 그림10에서 만든 함수를 이용하여 tiger 이미지에 가우시안 필터를 적용하는 코드이다. 이미지를 열어 배열로 바꾸어주고, sigma=4의 가우시안 필터를 적용시켜 이것을 다시 이미지로 바꾸고, 화면에 띄우고 저장한다.

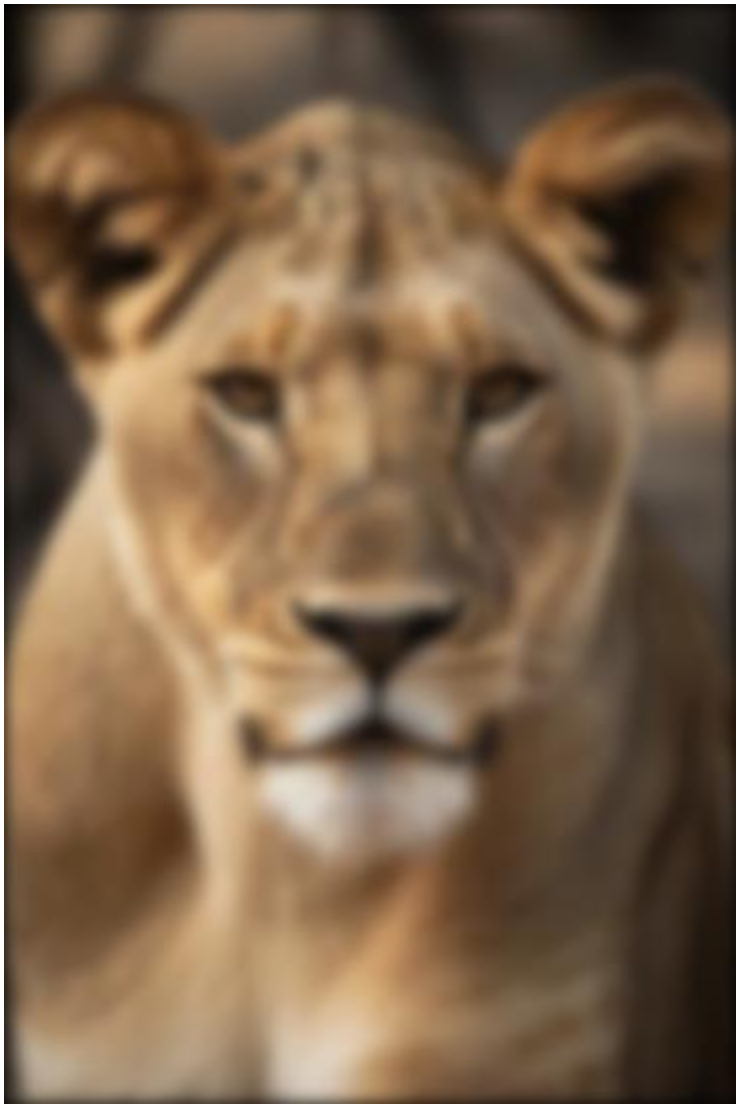

그림 13 그림12 코드 실행 결과. 호랑이 사진에 가우시안 필터가 적용되어 흐릿해졌다.

2-2 making high-frequency image

```
#2-2
#open image(for making high frequency image)
im3 = Image.open('images/3a_lion.bmp')

#make array from image
im3_array = np.asarray(im3)

#for making high frequency image, we should make low frequency image
low_frequency = gaussconvolved2dRGB(im3_array,4)

#make array that store high frequency image
high_frequency_result = np.asarray(im3_array)

#high frequency image is original image - low frequency image
high_frequency_result = high_frequency_result - low_frequency

# convert result array to image, show and store
#values of above high_frequency array are zero-means negative values. so we should add 128 for visualizing
high_frequency_img = Image.fromarray(high_frequency_result + 128)
high_frequency_img.show()
high_frequency_img.save("part2-2_result.bmp","bmp")
```

그림 14 making high-frequency image

그림12는 사자 사진에서 high-frequency 부분만 남기는 코드이다. 먼저 사자 사진을 열어 배열로 변환해준 다음, 그림10의 gaussconvolved2dRGB 함수를 적용하여(sigma=4) low-frequency 사자 사진을 만든다. 그리고 이를 원래 이미지에서 빼서 high-frequency인 부분만 남긴다.

이때 high-frequency인 부분을 보여주기 위해서는 각 픽셀에 128을 더해야 하는데, 음수를 제거하기 위해서이다.

사진을 저장하고, 보여준다.

그림 15 그림14 코드 실행결과. 사자 사진에서 high-frequency 부분만 남았다.

2-3 making hybrid image

```
#2-3
#making hybrid image. hybrid image is low frequency image + high frequency image
hybrid_array = low_frequency_result + high_frequency_result
hybrid_array =  np.clip(hybrid_array, 0, 255)
hybrid_image = Image.fromarray(hybrid_array)
hybrid_image.show()
hybrid_image.save("part2-3_result.bmp","bmp")
```

그림 16 hybrid image를 만들어 보여주고 저장하는 코드

그림16은 hybrid image를 만드는 코드이다. 이 코드는 2-1과 2-2에서 만들었던 low_frequency 이미지와 high_frequency 이미지를 단순히 더하여 만들었다(그리고 값을 0에서 255 사이로 제한한다)

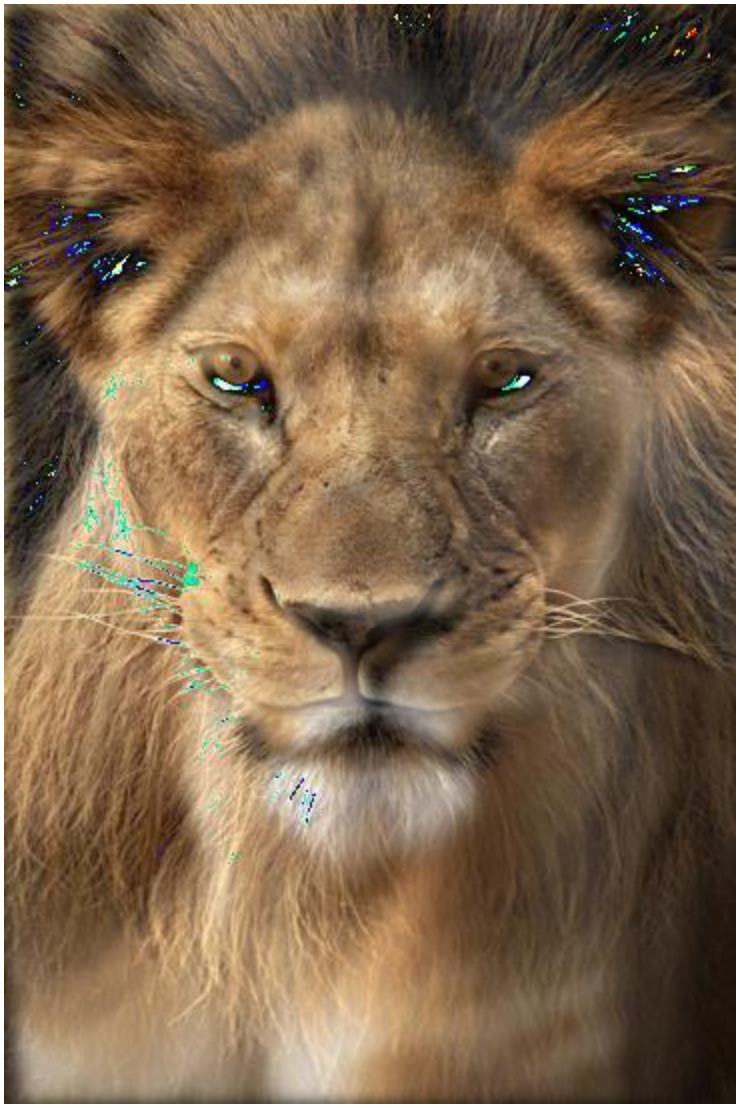

그림 17 그림16 코드 실행 결과. 상세한 부분은 호랑이, 윤곽은 사자인 사진이 만들어졌다.

그림17과 같이 hybrid image가 만들어졌다. 다만 그림17의 사진에 speckle artifact 부분이 있는데, 이를 제거하는 것은 잘 안되었다.

3. 결론

토의 및 결론 (1페이지)

이번 실습을 통해 boxfilter와 gaussian filter를 만드는 방법을 알게 되었고, 해당 필터를 이미지에 convolution하는 방법을 알게 되었다. 또한 gaussian filter를 이용해 이미지를 흐리게 하고, high-frequency 부분만 남기는 방법, 그리고 이를 활용하여 hybrid image를 만드는 방법을 알게 되었다. 다만, hybrid image에서 speckle artifact 부분이 있는데, 이를 완전히 제거하지는 못한 점이 아쉽다. 참고자료는 다음과 같다.

- 오일석 저, '컴퓨터 비전', 한빛아카데미, 2014
- https://thebook.io/006939/0344/
- https://numpy.org/doc/1.26/user/index.html