

# HW 02 – REPORT

소속 : 정보컴퓨터공학부

학번 : 202055536

이름 : 민      예      진

# 1. 서론

## 1) 실습 목표

Convolution과 가우시안 필터링을 직접 구현하고 구현한 함수를 이용하여 하이브리드 이미지를 생성하는 것을 목표로 한다.

## 2) 이론적 배경 기술

### ① Convolution

input 이미지의 픽셀 값과 180도 회전한 kernel를 곱한 후 더한 값을 통해 새로운 픽셀 값을 얻는 연산

### ② Filtering

#### i. Box Filtering(Mean filtering)

특정 픽셀과 주변 픽셀의 산술 평균을 각 픽셀의 값으로 가지는 필터

#### ii. Gaussian Filtering

$$\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

가우시안 분포 함수

가우시안 분포 함수를 근사하여 생성한 필터

### ③ Low / High frequency image

#### i. Low frequency image

원본 이미지와 가우시안 필터를 convolution한 이미지

#### ii. High frequency image

원본 이미지에서 low frequency image를 뺀 이미지

### ④ Hybrid image

서로 다른 두 이미지 중 하나의 이미지에서 low frequency image를 얻고 다른 하나의 이미지에서 high frequency image를 얻은 결과를 합한 이미지. 이미지의 크기가 작을 때는 low frequency image가 두드러지게 보이고 이미지의 크기가 클 때에는 high frequency image가 두드러지게 보인다.

## 2. 본론

## 1) Gaussian Filtering

① boxfilter

행, 열의 길이가 홀수인 boxfilter를 출력한다.

단, 짝수일 경우 assert문의 "Dimension must be odd"를 출력한다.

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

## 실행결과

( 1 ) boxfilter(3)

```
print(boxfilter(3))
```

✓ 0.0s

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

(2) `boxfilter(4)`

```
print(boxfilter(4))
⊗ 0.2s

-----
AssertionError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_1204\1822130950.py in
----> 1 print(boxfilter(4))

~\AppData\Local\Temp\ipykernel_1204\1525015975.py in boxfilter(n)
     2 # 1-1
     3 def boxfilter(n):
----> 4     assert n % 2 != 0, "Dimension must be odd"
     5     return np.ones((n, n)) / (n*n)

AssertionError: Dimension must be odd
```

(3) boxfilter(7)

[illegible]

## ② gauss1d(sigma)

sigma의 6배를 반올림 한 후 홀수로 보정한 수를 length라 할 때, (-length, length)의 값을 가지는 배열을 하나 만든 후, 가우시안 함수를 통해 값을 보정한다. 이때, 배열의 합은 0이 되어야 하므로 배열 값의 합을 나눔으로써 정규화를 해준다.

### 실행결과

```
print(gauss1d(0.3))  
✓ 0.0s  
[0.00383626 0.99232748 0.00383626]  
  
print(gauss1d(0.5))  
✓ 0.0s  
[0.10650698 0.78698604 0.10650698]  
  
print(gauss1d(1))  
✓ 0.0s  
[0.00443305 0.05400558 0.24203623 0.39905028 0.24203623 0.05400558  
0.00443305]  
  
print(gauss1d(2))  
✓ 0.0s  
[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 gauss2d(sigma)

gauss1d의 결과를 2차원으로 바꿔준다. gauss1d 함수의 리턴 값을 arr1d라 할 때, arr1d와 arr1d를 tranpose한 값을 외적함으로써 2차원으로 바꿔준다.

### 실행결과

```
print(gauss2d(0.5))  
✓ 0.0s  
array([[0.01134374, 0.08381951, 0.01134374],  
       [0.08381951, 0.61934703, 0.08381951],  
       [0.01134374, 0.08381951, 0.01134374]])  
  
print(gauss2d(1))  
✓ 0.0s  
[[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 convolution

##### a. convolve2d(array, sigma)

- (1) array와 filter의 자료형을 float32로 바꾼다.
- (2) Image zero padding : convolution 후 결과물이 array보다 작아지는 것을 막기 위해 (filter의 길이-1)/2만큼의 길이로 array의 가장자리에 0을 덧붙인다.
- (3) convolution을 위해 filter를 180도 뒤집는다. filter를 1차원 배열로 바꾼 후 역순으로 뒤집고 다시 2차원배열로 바꾸는 방식으로 구현했다.
- (4) 180도 뒤집은 filter와 array를 cross correlation한다. array를 filter의 크기만큼 자른 후 cross correlation 하는 방식으로 구현했다.

##### b. gaussconvolve2d(array, sigma)

gauss2d 함수의 결과를 convolve2d의 filter로 전달하여 가우시안 필터로 convolution을 진행한다.

##### c. 2b\_dog.bmp에 가우시안 필터 적용

- 1. 2b\_dog.bmp 이미지를 greyscale로 변환한다.
- 2. 변환한 후 gaussconvolve(2b\_dog 이미지, sigma=3)을 통해 가우시안 필터링을 진행한다.

##### d. result



## 2) Hybrid Images

### ① 3b\_tower.bmp blurring

- (1) 이미지를  $r, g, b$  채널로 분리한다.
- (2) 각각의 채널에 `gaussconvolve` 함수를 통해 가우시안 필터링을 적용한다.
- (3) Image 객체로 변환을 위해 `uint8`로 타입 변환 후 이미지를 출력한다.


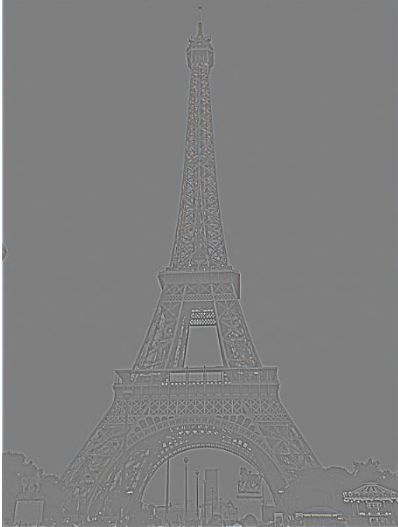
#### 실행결과



### ② 3a\_eiffel.bmp의 high frequency version 추출

- (1) 이미지를  $r, g, b$  채널로 분리한다.
- (2) 각각의 채널에 가우시안 필터링 진행한다.
- (3) 원본이미지 - 가우시안 필터링을 적용한 이미지 후 결과값에 128을 더한다.
- (4) 위의 결과 값 중 255 초과인 값을 255로 보정한다.
- (5) Image 객체로 변환을 위해 `uint8`로 타입 변환 후 각각의  $r, g, b$  채널을 합쳐 이미지를 출력한다.

## 실행결과

	
원본 이미지	High frequency image

### ③ Hybrid 이미지 얻기

- (1) ①의 가우시안 필터링 이미지 + ② high frequency 이미지
- (2) 음수 값과 255를 초과하는 값을 각각 0과 255로 보정한다.
- (3) 원본이미지 - 가우시안 필터링을 적용한 이미지 후 결과값에 128을 더한다.
- (4) 위의 결과 값 중 255 초과인 값을 255로 보정한다.
- (5) Image 객체로 변환을 위해 uint8로 타입 변환 후 각각의 r, g, b 채널을 합쳐 이미지를 출력한다.

## 실행결과



### 3. 결론

Part 1 : Gaussian Filtering에서는 numpy의 여러가지 method를 통해 box filter와 gaussian filter를 직접 구현해 보고 구현한 filter를 바탕으로 convolution 또한 구현해 보았다. Part2 : Hybrid Images에서는 Part 1에서 구현한 함수들을 통해 low frequency image와 high frequency image를 직접 추출하고 서로 다른 이미지의 low frequency image와 high frequency image를 합치면서 hybrid image를 만들어 볼 수 있었다.

위 과정 속에는 모두 convolution 연산이 포함되어 있다. 이 연산의 결과는 filter의 종류에 따라 연산 결과의 이미지의 형태가 달라진다. 위 실습에서는 mean filter와 gaussian filter를 사용해 보았다. 이외의 다른 filter는 어떤 것이 있는지 알아보자.

#### **Median Filter**

커널의 픽셀 값 중 중앙값을 선택한 필터이다. 이 필터를 통해 소금 & 후추(Salt & Pepper) 노이즈를 효과적으로 제거할 수 있다.

#### **Bilateral Filter**

gaussian filter와 edge filter를 합한 필터이다. Mean filter, gaussian filter, Median filter를 활용하여 잡음을 제거할 수 있지만 이는 경계를 흐릿하게 하는 문제가 있다. Bilateral filter를 통해 경계를 살리며 블러링 효과를 낼 수 있다.

이와 같이 이미지 및 영상을 보정하기 위해 다양한 filter를 사용하여 다양한 이미지 및 영상의 결과를 얻을 수 있다.