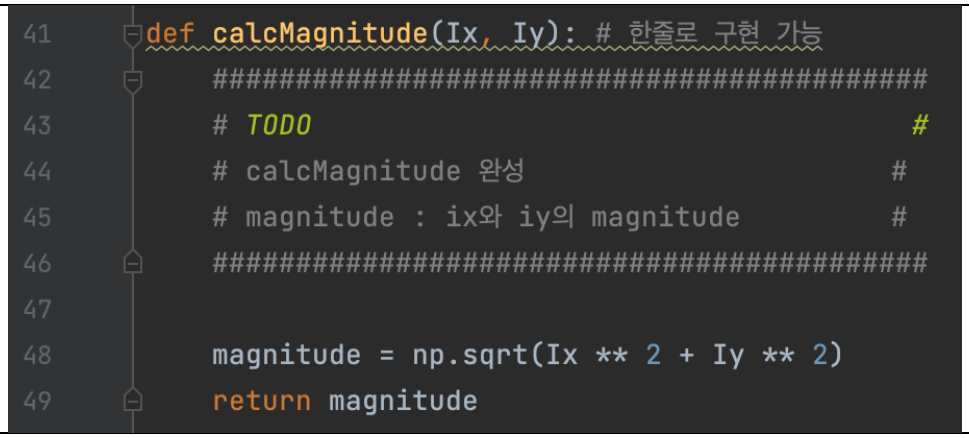
영상처리

6 주차 보고서

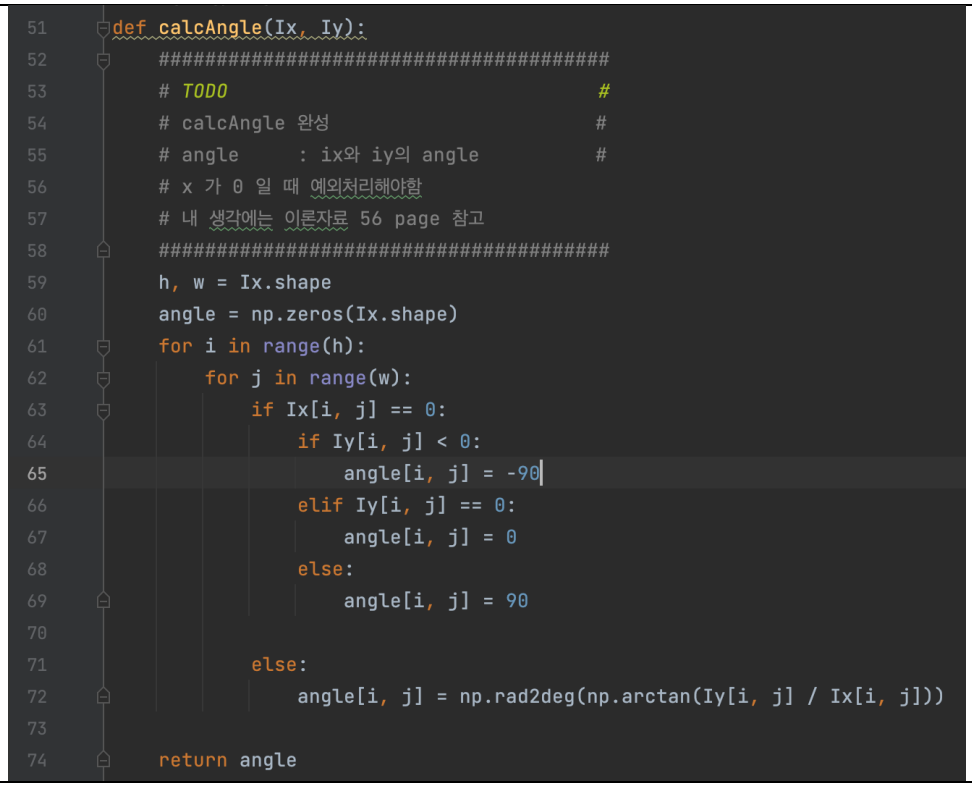
컴퓨터융합학부

202102699 정민경

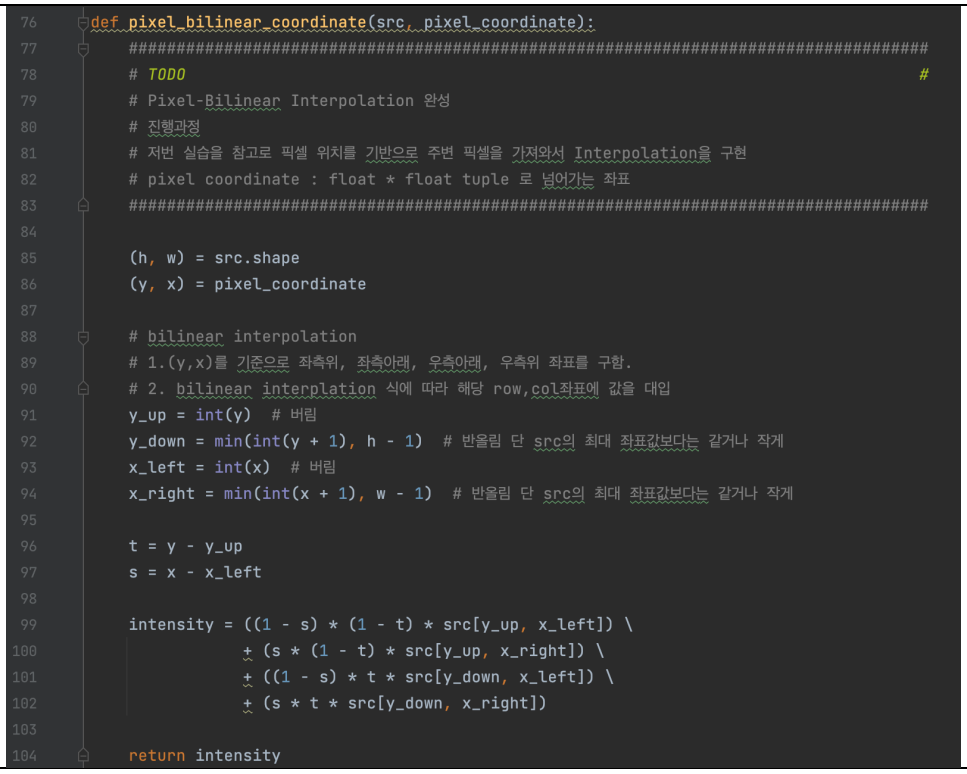
1. 과제 설명

함수이름	calcMagnitude()
코드설명	이론자료에 나와있는 수식대로 ix 와 iy 를 인자로 넘겨받아 각각의 제곱의 합에 루트를 씌워 magnitude 를 구해주었다.
이미지	

```python
def calcMagnitude(Ix, Iy): # 한줄로 구현 가능
    ##########################################
    # TODO                                   #
    # calcMagnitude 완성                     #
    # magnitude : ix와 iy의 magnitude        #
    ##########################################

    magnitude = np.sqrt(Ix ** 2 + Iy ** 2)
    return magnitude
```

함수이름	calcAngle()
코드설명	사이 각을 구할 때 사용하는 함수로 이론자로의 수식대로 iy / ix를 아크탄젠트에 넣어서 값을 계산해주었다. 이 때 나누는 값인 ix 가 0일 때 iy 의 값에 따라 angle 을 따로 구해줘야한다. Ix 가 0일 때 나올 수 있는 경우는 아래와 같다. 1) iy == 0 일 때 -> 둘다 0이므로 angle 도 0이다. 2) Iy > 0 일 때 -> y방향의 변화값이 양수이므로 angle 은 + 90도이다. 3) Iy < 0 일 때 -> y방향의 변화값이 양수이므로 angle 은 - 90도이다.
이미지	

```python
def calcAngle(Ix, Iy):
    ######################################
    # TODO                               #
    # calcAngle 완성                     #
    # angle     : ix와 iy의 angle        #
    # x 가 0 일 때 예외처리해야함
    # 내 생각에는 이론자료 56 page 참고
    ######################################
    h, w = Ix.shape
    angle = np.zeros(Ix.shape)
    for i in range(h):
        for j in range(w):
            if Ix[i, j] == 0:
                if Iy[i, j] < 0:
                    angle[i, j] = -90
                elif Iy[i, j] == 0:
                    angle[i, j] = 0
                else:
                    angle[i, j] = 90

            else:
                angle[i, j] = np.rad2deg(np.arctan(Iy[i, j] / Ix[i, j]))

    return angle
```

함수이름	pixel_bilinear_coordinate()
코드설명	Bilinear interpolation 방법으로 intensity 를 구하는 함수이다. 이전 과제와 동일하게 구현하면 되는데, 다른점은 이번 과제는 인자로 src 와 내가 채워넣을 pixel 의 좌표를 넘겨받는다. 그러면 현재 위치는 알기 때문에 pixel_coordinate 값을 (y, x) 로 받아 이전 과제와 동일하게 수행하도록 구현했다.
이미지	

```
def pixel_bilinear_coordinate(src, pixel_coordinate):
    ###############################################################################
    # TODO                                                                        #
    # Pixel-Bilinear Interpolation 완성
    # 진행과정
    # 저번 실습을 참고로 픽셀 위치를 기반으로 주변 픽셀을 가져와서 Interpolation을 구현
    # pixel coordinate : float * float tuple 로 넘어가는 좌표
    ###############################################################################

    (h, w) = src.shape
    (y, x) = pixel_coordinate

    # bilinear interpolation
    # 1.(y,x)를 기준으로 좌측위, 좌측아래, 우측아래, 우측위 좌표를 구함.
    # 2. bilinear interplation 식에 따라 해당 row,col좌표에 값을 대입
    y_up = int(y)  # 버림
    y_down = min(int(y + 1), h - 1)  # 반올림 단 src의 최대 좌표값보다는 같거나 작게
    x_left = int(x)  # 버림
    x_right = min(int(x + 1), w - 1)  # 반올림 단 src의 최대 좌표값보다는 같거나 작게

    t = y - y_up
    s = x - x_left

    intensity = ((1 - s) * (1 - t) * src[y_up, x_left]) \
                + (s * (1 - t) * src[y_up, x_right]) \
                + ((1 - s) * t * src[y_down, x_left]) \
                + (s * t * src[y_down, x_right])

    return intensity
```

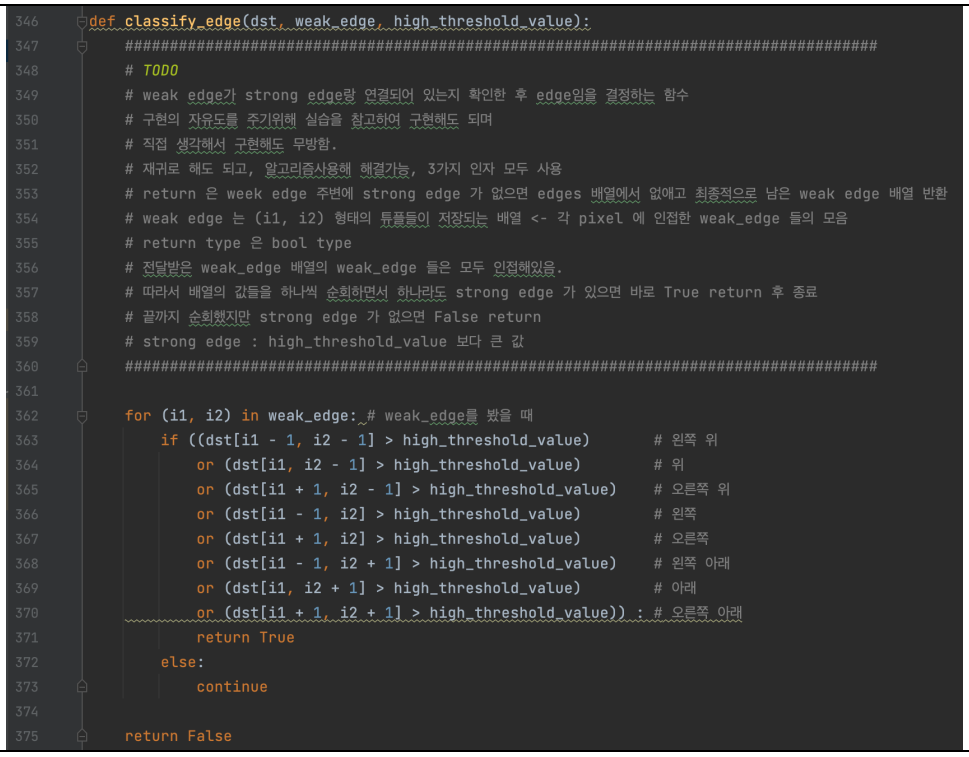
함수이름	non_maximum_supression_five_size()
코드설명	현재 보고있는 좌표가 edge 좌표가 맞는지 확인 후 주변 8개 + 본인까지 총 9개의 좌표 중 본인이 가장 큰 값이면 edge로 판단 후 largest_magnitude 에 본인의 magnitude 를 저장하고, 가장 큰 값이 아니라면 edge 좌표가 아니므로 현재 위치의 largest_magnitude 에 0을 저장한다. 우리는 step 을 0.5로 총 9개의 좌표를 비교한다. 범위는 -2 ~ 2 까지이므로 반복문의 범위를 -2에서부터 2.5까지 0.5만큼씩 더하면서 본인 제외 총 8개의 magnitude 를 구해주었다. 나는 이 함수를 구할 때 for문 안에서 magnitude 를 하나 구한 후 현재 보고있는 magnitude (코드상에서는 pixel_magnitude 와 magnitude) 를 비교해 현재 보고있는 magnitude 가 크다면 largest_magnitude 에 현재 magnitude 값을 저장하고, 만약 한번이라도 pixel_magnitude 가 크다면 현재 보고있는 magnitude 가 edge 가 아니란 소리이기 때문에 largest_magnitude 에 0을 저장하고 반복문을 빠져나가도록 구현했다. 이 방법을 for문에서 하나씩 비교해서 총 8번을 도는 것이 아닌 주변 8개 + 본인 magnitude 를 하나의 배열에 모두 저장해 그 중 max 값이 현재 보고있는 magnitude 와 같다면 largest_magnitude 의 값을 magnitude 로 설정하는 방법으로 구현할 수도 있다. 이렇게 하게되면 for문마다 매번 반복을 하지 않아도 되고, 코드에 겹치는 부분을 많이 제거할 수 있어 좋은 방법인 것 같다.

이미지

```python
def non_maximum_supression_five_size(magnitude, angle, step = 0.5):
    ###############################################################################
    # TODO
    # non_maximum_supression 완성
    # largest_magnitude     : non_maximum_supression 결과(가장 강한 edge만 남김)
    ###############################################################################
    (h, w) = magnitude.shape  # h = 512, w = 512
    # angle의 범위 : -90 ~ 90
    largest_magnitude = np.zeros((h, w))

    for row in range(2, h - 2):
        for col in range(2, w - 2):
            degree = angle[row, col]

            # gradient의 degree는 edge와 수직방향이다.
            if 0 <= degree and degree < 45:
                for i in np.arange(-2, 2.5, step):
                    if (i != 0) :
                        rate = np.tan(np.deg2rad(degree)) * i
                        pixel_coordinate = ((row + rate), (col + i))
                        pixel_magnitude = pixel_bilinear_coordinate(magnitude, pixel_coordinate)
                        # 현재 보고있는 좌표의 magnitude 가 bilinear interpolation 으로 구한 모든 magnitude 보다 크다면
                        # largest_magnitude 에 현재 보고있는 좌표의 magnitude 를 저장
                        if magnitude[row, col] == max(pixel_magnitude, magnitude[row, col]):
                            largest_magnitude[row, col] = magnitude[row, col]
                        else: # 그렇지 않은 경우가 하나라도 있으면 edge 가 아니므로 0으로 값 변경 후 break
                            largest_magnitude[row, col] = 0
                            break
                        # 위의 if-else 문을 반복문안에서 계속 수행하면 효율이 좋지 않음. -> 어떠한 배열에 모든 값 저장 후 한번에 비교해도 무방.
            elif 45 <= degree and degree <= 90:
                for i in np.arange(-2, 2.5, step):
                    if (i != 0):
                        rate = np.tan(np.deg2rad(90 - degree)) * i
                        pixel_coordinate = ((row + i), (col + rate))
                        pixel_magnitude = pixel_bilinear_coordinate(magnitude, pixel_coordinate)
                        if magnitude[row, col] == max(pixel_magnitude, magnitude[row, col]):
                            largest_magnitude[row, col] = magnitude[row, col]
                        else:
                            largest_magnitude[row, col] = 0
                            break


            elif -45 <= degree and degree < 0:
                for i in np.arange(-2, 2.5, step):
                    if (i != 0):
                        rate = -np.tan(np.deg2rad(degree)) * i
                        pixel_coordinate = ((row + rate), (col + i))
                        pixel_magnitude = pixel_bilinear_coordinate(magnitude, pixel_coordinate)
                        if magnitude[row, col] == max(pixel_magnitude, magnitude[row, col]):
                            largest_magnitude[row, col] = magnitude[row, col]
                        else:
                            largest_magnitude[row, col] = 0
                            break

            elif -90 <= degree and degree < -45:
                for i in np.arange(-2, 2.5, step):
                    if (i != 0):
                        rate = -np.tan(np.deg2rad(90 - degree)) * i
                        pixel_coordinate = ((row + i), (col + rate))
                        pixel_magnitude = pixel_bilinear_coordinate(magnitude, pixel_coordinate)
                        if magnitude[row, col] == max(pixel_magnitude, magnitude[row, col]):
                            largest_magnitude[row, col] = magnitude[row, col]
                        else:
                            largest_magnitude[row, col] = 0
                            break

            else:
                print(row, col, 'error!  degree :', degree)

    return largest_magnitude
```

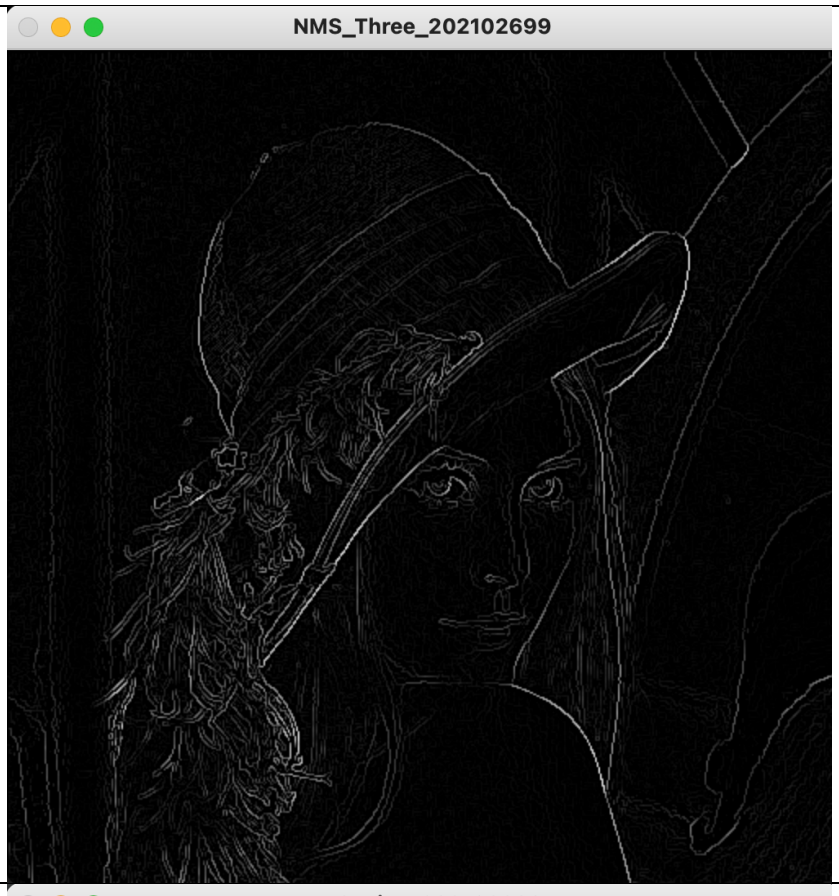
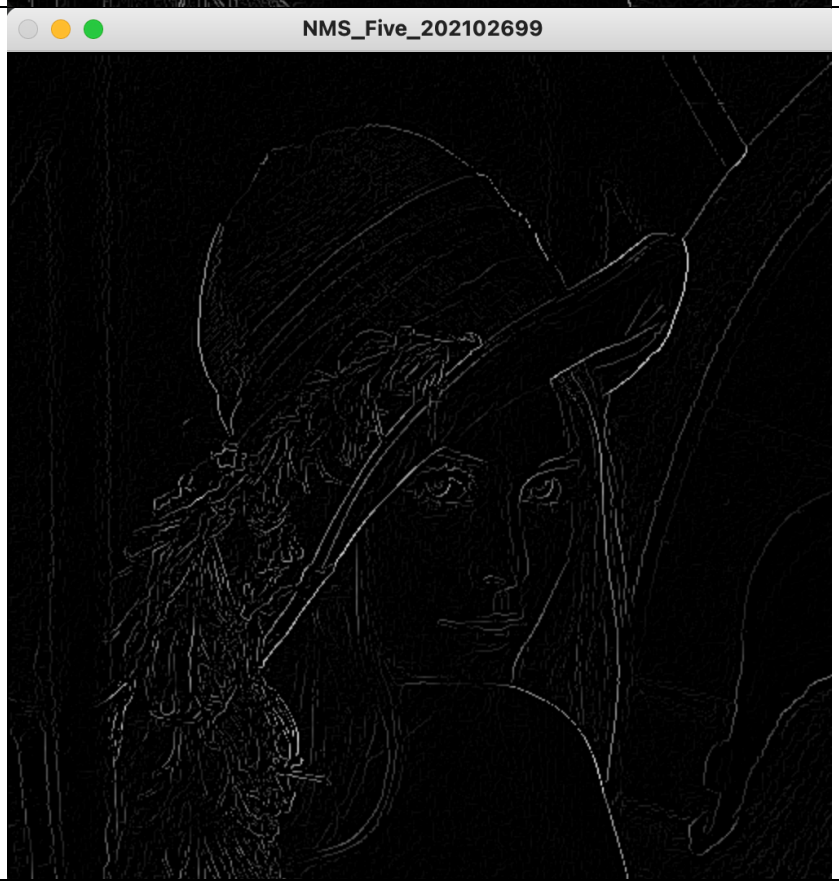
함수이름	search_weak_edge()
코드설명	이 함수는 현재 좌표에서로부터 주변 8개의 위치에 인접해있는 weak_edge 들이 저장되어있는 배열을 반환하는 함수이다. 실습코드의 find_connected_dots 코드를 참고해서 구현하였다. 현재 받은 좌표가 이미 확인한 index이거나, weak_edge 가 아닐때 즉, 현재 좌표의 dst 의 값이 low_threshold 보다 작거나 high_threshold 보다 큰 경우에는 주변 8개의 값들을 확인하지 않고 바로 return 하고, 그렇지 않을 경우 weak_edge 가 저장되는 배열에 추가해준 후, 그 값 주변 8개의 값을 재귀로 탐색한다. 8개의 좌표에 대해 재귀로 탐색한 후 update 된 중복 제거 후 중복되어있지않은 weak_edge 만 저장되어있는 list 를 반환한다.
이미지	

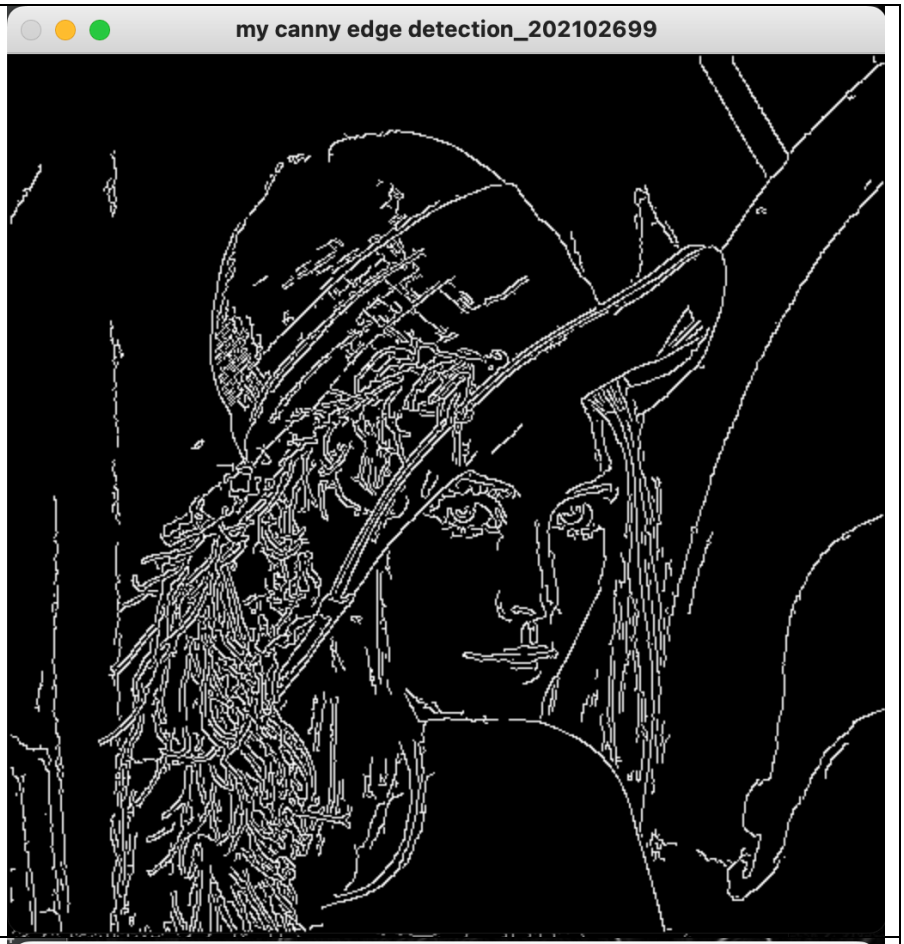
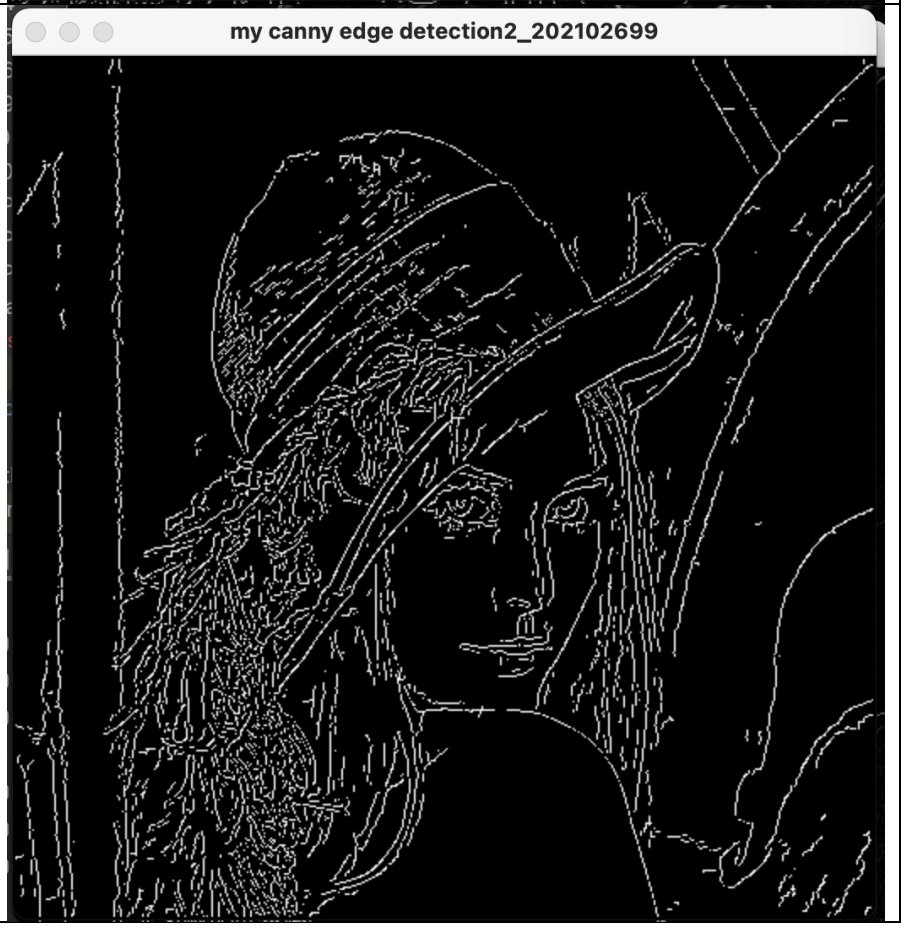
```python
def search_weak_edge(dst, edges, high_threshold_value, low_threshold_value, coordinate):
    ###############################################################################
    # TODO
    # search_weak_edge 함수
    # Goal : 연결된 Weak Edge를 찾아서 저장하는 함수
    # 구현의 자유도를 주기위해 실습을 참고하여 구현해도 되며
    # 직접 생각해서 구현해도 무방함. -> 재귀로 해결하기 위해 좌표값인 coordinate 인자 추가
    # dst : image [0 ~ 255]
    # edges : weak_edge 가 저장되는 배열 ( Tl <= dst <= Th 을 만족하는 dst 저장 )
    # high_threshold_value : Th
    # low_threshold_value : Tl
    # coordinate : 인접한 8개의 pixel 을 확인하기 위한 위치를 넘겨주는 float * float tuple 좌표
    # return 은 week edge 가 저장되어 있는 배열
    # weak edge 는 (i1, i2) 형태의 튜플들이 저장되는 배열
    # edges 로 전달받은 인덱스 (1개) 에 인접해있는 weak_edge 들만 저장되어있음.
    ###############################################################################

    h, w = dst.shape
    row, col = coordinate

    # 이미 확인한 index 이거나, weak_edge 가 아닐때는 현재의 edge 를 return 한다.
    if ((row, col) in edges) or (dst[row, col] < low_threshold_value) or (dst[row, col] >= high_threshold_value):
        return edges

    # 이번에 처음 확인한 index 이고, weak_edge 이면 edge 배열에 추가하고 다시 주변 8개의 pixel 에 대해 탐색한다.
    edges.append((row, col))
    # 1번 좌표
    if row > 0 and col > 0:
        edges = search_weak_edge(dst, edges, high_threshold_value, low_threshold_value, (row-1, col -1))
    # 2번 좌표
    if row > 0:
        edges = search_weak_edge(dst, edges, high_threshold_value, low_threshold_value, (row-1, col))
    # 3번 좌표
    if row > 0 and col < w - 1:
        edges = search_weak_edge(dst, edges, high_threshold_value, low_threshold_value, (row-1, col + 1))
    # 4번 좌표
    if col > 0:
        edges = search_weak_edge(dst, edges, high_threshold_value, low_threshold_value, (row, col - 1))
    # 6번 좌표
    if col < w - 1:
        edges = search_weak_edge(dst, edges, high_threshold_value, low_threshold_value, (row, col + 1))
    # 7번 좌표
    if row < h - 1 and col > 0:
        edges = search_weak_edge(dst, edges, high_threshold_value, low_threshold_value, (row + 1, col - 1))
    # 8번 좌표
    if row < h - 1:
        edges = search_weak_edge(dst, edges, high_threshold_value, low_threshold_value, (row + 1, col))
    # 9번 좌표
    if row < h - 1 and col < w - 1:
        edges = search_weak_edge(dst, edges, high_threshold_value, low_threshold_value, (row + 1, col + 1))

    # 중복 제거
    return list(set(edges))
```

함수이름	classify_edge()
코드설명	인접한 Weak_edge 들이 저장되어있는 배열을 보고 weak_edge 주변 8개에 strong edge 가 없으면 False 를 return 하고, 주변에 strong edge 가 하나라도 있으면 True 를 return 하도록 구현했다. 전달받은 weak_edge 배열의 weak_edge 들은 모두 인접해있다. 그렇기 때문에 전달받은 weak_edge 배열의 값들을 하나씩 순회하면서 하나라도 주변에 strong edge 가 있으면 true 를 반환하면 되는 것이다.
이미지	

```python
def classify_edge(dst, weak_edge, high_threshold_value):
    ###############################################################################
    # TODO
    # weak edge가 strong edge랑 연결되어 있는지 확인한 후 edge임을 결정하는 함수
    # 구현의 자유도를 주기위해 실습을 참고하여 구현해도 되며
    # 직접 생각해서 구현해도 무방함.
    # 재귀로 해도 되고, 알고리즘사용해 해결가능, 3가지 인자 모두 사용
    # return 은 week edge 주변에 strong edge 가 없으면 edges 배열에서 없애고 최종적으로 남은 weak edge 배열 반환
    # weak edge 는 (i1, i2) 형태의 튜플들이 저장되는 배열 <- 각 pixel 에 인접한 weak_edge 들의 모음
    # return type 은 bool type
    # 전달받은 weak_edge 배열의 weak_edge 들은 모두 인접해있음.
    # 따라서 배열의 값들을 하나씩 순회하면서 하나라도 strong edge 가 있으면 바로 True return 후 종료
    # 끝까지 순회했지만 strong edge 가 없으면 False return
    # strong edge : high_threshold_value 보다 큰 값
    ###############################################################################

    for (i1, i2) in weak_edge: # weak_edge를 봤을 때
        if ((dst[i1 - 1, i2 - 1] > high_threshold_value)       # 왼쪽 위
            or (dst[i1, i2 - 1] > high_threshold_value)        # 위
            or (dst[i1 + 1, i2 - 1] > high_threshold_value)    # 오른쪽 위
            or (dst[i1 - 1, i2] > high_threshold_value)        # 왼쪽
            or (dst[i1 + 1, i2] > high_threshold_value)        # 오른쪽
            or (dst[i1 - 1, i2 + 1] > high_threshold_value)    # 왼쪽 아래
            or (dst[i1, i2 + 1] > high_threshold_value)        # 아래
            or (dst[i1 + 1, i2 + 1] > high_threshold_value)) : # 오른쪽 아래
            return True
        else:
            continue

    return False
```

2. 결과 이미지

Non maximum 3x3	
Non maximum 5x5	

Canny Edge 3x3	
Canny Edge 5x5	

3. 결과 분석

- 3x3 Canny Edge Detection 수행 시 Threshold 값이랑 5x5 Canny Edge Detection 수행 시 Threshold 값이랑 비교 시 왜 3x3 > 5x5 가 나오는 지 :
-

  ■ 현재 좌표가 edge 인지 판별할 때 3x3 은 본인 포함 3개의 좌표 중 max 값이면 edge 이고, 5x5 은 본인 포함 9개의 좌표 중 max 값이어야 edge 이다. 확률적으로 3 개 중 max 일 확률이 9개 중 max 일 확률보다 크다. 그렇다는 것은 3x3 이 현재 좌표를 edge로 더 자주 판별한다는 것이고, 5x5 보다 오른쪽으로 조금 더 치우쳐져있음을 알 수 있다.

  그렇기 때문에 오른쪽으로 좀 더 치우쳐져있는 3x3 이 5x5 보다 threshold 가 더 큰 것이다.