Canny Edge Detection

Import Library

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

1. Noise reduction

[함수]

```
def gauss1d(sigma):
    # sigma value가 0이 되서는 안된다.(sigma value가 0이면 분모가 0이 된다.)
    assert sigma != 0, "Sigma value sholudn't be zero"

    # sigma를 6으로 곱하고 올림
    length = int(math.ceil(float(sigma) * 6))

    if length % 2 == 0:
        length += 1

    center = length // 2

    # -center ~ center의 1차원 배열 생성
    x = np.arange(-center, center + 1)

    # sum to 1을 위한 처리
    ret = np.exp(-(x**2)/(2*sigma**2))
    ret /= np.sum(ret)

    return ret
```

```
def gauss2d(sigma):
    # np.outer(외적) 연산
    ret = np.outer(gauss1d(sigma), gauss1d(sigma))
    ret /= np.sum(ret)
    return ret
```

```
def convolve2d(img, filter):
    # np.flipud : 위, 아래 방향 뒤집기
    # np.fliplr : 좌, 우 방향 뒤집기
    filter = np.flipud(np.fliplr(filter))

    pad = int(np.sqrt(filter.size)) # 19
    pad2 = pad//2 # 9

    # 리턴값
    output = np.zeros_like(img, dtype = 'float32')

    # Zero padding
    padded = np.zeros((img.size[1] + 2*pad2, img.size[0] + 2*pad2))
    padded[pad2:-pad2, pad2:-pad2] = img

    # 이중 for loop로 zero padding 구현
    for x in range(img.size[0]):
        for y in range(img.size[1]):
            output[y,x] = (filter * padded[y:y+pad, x:x+pad]).sum()

    return output
```

```
def gaussconvolve2d(img, sigma):
    # sigma를 통해 gauss를 통해 convolution 연산을 통해 결과를.
    # convolve2d와 gauss2d를 이용해서 만들면 된다.
    filter = gauss2d(sigma)
    output = convolve2d(img, filter)

    return output
```

[main]

```
original = Image.open('iguana.bmp')
```

```
im = Image.open('iguana.bmp')
im = im.convert('L')
```

```
im = gaussconvolve2d(im, 1.6)
print(im)
```

```
[[ 8.886213 11.487304 12.666539 ... 57.269455 50.504623 37.87184 ]
 [11.804983 15.260447 16.827074 ... 75.86449  66.487335 49.639004]
 [13.42956  17.360685 19.14332  ... 86.52235  75.42679  56.125107]
 ...
 [11.482877 16.171326 19.823963 ... 75.33419  71.90424  57.36441 ]
 [13.394698 19.022598 23.604034 ... 56.2243   56.12303  46.52825 ]
 [12.716841 18.092936 22.507124 ... 36.431942 38.097576 32.806984]]
```

```
im = im.astype('uint8')
im = Image.fromarray(im)
original.show()
im.show()
```

1. 기존의 color 이미지를 grayscale 이미지로 변환

2. gaussian convolution 연산을 통해 blur 처리

[output]

[변경 전] [변경 후]

2. Finding the intensity gradient of the image

[함수]

```
def sobel_filters(img):

    x_filter = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    y_filter = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])

    x_val = convolve2d(img, x_filter)
    y_val = convolve2d(img, y_filter)

    G = np.hypot(x_val, y_val)
    theta = np.arctan2(y_val, x_val)

    # 0-255로 값을 맞춰주기 위해
    if G.max() > 255:
        np.where(G, G, G*255//G.max())


    return (G, theta)
```

[코드]

```
G, theta = sobel_filters(im)
```

```
im2 = G.astype('uint8')
im2 = Image.fromarray(im2)
im2.show()
```

$$G = \sqrt{I_x^2 + I_y^2} \ , \ \theta = \tan^{-1}\frac{I_y}{I_x}$$

1. Sobel filter를 이용하여 gradient와 theta를 찾는다.

2. 찾아낸 gradient를 출력

[output]

3. Non-Maximum Suppression

[함수]

```
def non_max_suppression(G, theta):
    degree = np.rad2deg(theta)

    size = G.shape
    suppressed = np.zeros(size)

    # padding을 제거하기 위해서 0, size[0]은 제외.
    for i in range(1, size[0]-1):
        for j in range(1, size[1]-1):
            # degree : 0 - 수평 방향
            if 0 <= degree[i][j] < 22.5 or 157.5 <= degree[i][j] <= 180:
                temp = max(G[i][j-1], G[i][j+1])
            # degree : 45 - 양의 대각선
            elif 22.5 <= degree[i][j] < 67.5:
                temp = max(G[i-1][j-1], G[i+1][j+1])
            # degree : 90 - 수직 방향
            elif 67.5 <= degree[i][j] < 112.5:
                temp = max(G[i-1][j], G[i+1][j])
            # degree : 135 - 음의 대각선
            else:
                temp = max(G[i-1][j+1], G[i+1][j-1])

            if G[i][j] >= temp:
                suppressed[i][j] = G[i][j]

    suppressed = np.multiply(suppressed, 255 / suppressed.max())
    return suppressed
```

[main]

```
NMS = non_max_suppression(G, theta)
im3 = NMS.astype('uint8')
im3 = Image.fromarray(im3)
im3.show()
```

1. 이전 과정에서 구한 theta를 degree로 변환(numpy.rad2deg 이용)

2. 0, 45, 90, 135의 각도를 이용해서 주변의 pixel과 현재 pixel을 비교하고, 최댓값인 경우 그대로 놔두고 아닐 경우(Non-max) 제거(Suppression)

[output]

4. Double threshold

[함수]

```
def double_thresholding(img):
    img = np.array(img)
    # 비율 설정
    high_ratio = 0.15
    low_ratio = 0.03

    # Threshold high, low 설정
    diff = img.max() - img.min()
    T_high = img.min() + diff*high_ratio
    T_low = img.min() + diff*low_ratio
    print(T_high, T_low)
    '''
    # np.where() 삼중 연산자와 유사한 기능. for loop를 사용하지 않아도 된다.
    # 1. T_high보다 크거나 같으면 255(strong)
    temp1 = np.where(T_high <= img, 255, 0)
    # 2. T_low보다 크거나 같고 T_high 보다 작으면 80(weak)
    temp2 = np.where(T_low <= img, 80, 0)
    # 3. T_low보다 작으면 0(non-relevant)
    # img = np.where(T_low > img, 0, img)
    '''

    h, w = img.shape

    for i in range(1, h-1):
        for j in range(1, w-1):
            if img[i][j] > T_high:
                img[i][j] = 255
            elif img[i][j] > T_low:
                img[i][j] = 80
            else:
                img[i][j] = 0
    return img
```

[main]

```
im4 = double_thresholding(im3)
im4 = im4.astype('uint8')
im4 = Image.fromarray(im4)
im4.show()
```

$$diff = \max(image) - \min(image)$$
$$T_{high} = \min(image) + diff * 0.15$$
$$T_{low} = \min(image) + diff * 0.03$$

1. 주어진 값과 식을 이용해서 threshold를 설정

2. 모든 pixel을 조사하여 T_high를 넘기는 경우 255(strong)로, 255가 되지 않고 남은 것들 중 T_low를 넘기는 경우 80(weak)으로 설정하였고 나머지는 0(non-relevant)으로 처리하였다.

[output]

5. Edge Tracking by hysteresis

[함수]

```
def hysteresis(img):
    img = np.array(img)
    # 비율 설정
    high_ratio = 0.15
    low_ratio = 0.03
    # strong, weak value 변수화
    strong, weak = 255, 80

    # 위에서 사용했던 threshold
    diff = img.max() - img.min()
    T_high = img.min() + diff*high_ratio
    T_low = img.min() + diff*low_ratio

    # BFS
    visited = [[False]*img.shape[1] for _ in range(img.shape[0])]
    r, c = img.shape[0], img.shape[1]

    dx = [-1, 1, 0, 0]
    dy = [0, 0, -1, 1]
    q = deque()
    q.append((1, 1))
    while q:
        x, y = q.popleft()
        # weak주변에 strong이 있는지 최종 확인
        flag = False
        for i in range(4):
            nx = x + dx[i]
            ny = y + dy[i]
            # weak 주변에 strong이 있으면 strong으로
            if img[x][y] == weak and img[nx][ny] == strong:
                img[x][y] = strong
                flag = True
            if 1 <= nx < r and 1 <= ny < c and visited[nx][ny] == False:
                visited[nx][ny] = True
                q.append((nx, ny))
        if flag == False and (img[x][y] == weak or img[x][y] == 0):
            img[x][y] = 0

    return img
```

[main]

```
from collections import deque # 큐를 빠르게 사용하기 위한 라이브러리 import
im5 = hysteresis(im4)
im5 = im5.astype('uint8')
im5 = Image.fromarray(im5)
im5.show()
```

1. 앞에서 주어진 값과 식을 이용해서 threshold를 설정

2. 현재 pixel이 weak인 경우 주변의 pixel을 조사하여 strong인 pixel이 있으면 현재 pixel을 strong으로 변환해주는 작업 수행. (BFS를 이용하였다.)

[output]