COMPUTER VISION PROJECT 1

-Edge Detection-

과목명 : [CSI4116-01] COMPUTER VISION
담당교수 : 연세대학교 컴퓨터공학과 변혜란
학번 : S20131598
이름 : 전상현

REPORT

I. 개발 목표

- Edge Detection은 영상이나 사진에서의 객체를 식별하기 위해 객체의 경계선(Edge)를 찾는 것이다. Edge에는 모양(shape), 방향(direction)을 탐지할 수 있는 등 여러 정보를 가지고 있다. Edge Detection을 구현하는 방법에는 여러 가지 종류가 있는데, 본 프로젝트에서는 Canny Edge Detection을 사용하여 Edge Detection을 수행하도록 한다.

II. 개발 범위 및 내용

가. 개발 범위

1. **Edge Detection** (Canny Edge Detection)

나. 개발 내용

1. **Edge Detection** (Canny Edge Detection)

- Canny Edge Detection을 이용하여 Edge Detection을 수행한다.
- Canny Edge Detection의 상세 내용은 다음과 같다.
    a. Blurring을 통한 Noise suppress (Gaussian Filter 적용)
    b. Gradient magnitude와 direction 계산
    c. Non-Maximum Suppression 수행
    d. Double threshold를 이용하여 edge 구분
    e. Edge 연결

III. 결과

1. 제작 내용

Suppress Noise

```
def edge(img):
    img = color.rgb2gray(img)

    # Suppress noise
    img = filters.gaussian_filter(img, sigma=2)
    height, width = img.shape
    shape = (height, width)
```

- scipy.ndimage에 기본적으로 제공되는 함수를 이용하여 Gaussian filter를 적용함으로서 해당 image의 noise를 제거하여 edge detection의 성능을 높여준다.
- 입력받은 이미지가 rgb 이미지이므로 Gaussian filter를 적용하기 전에 grayscale로 만들어 준다.

Calculate gradient magnitude and direction

```
# Calculate gradient and direction
for y in range(1, height+1):
    for x in range(1, width+1):
        gx[y - 1, x - 1] = (img[y - 1, x + 1] + 2 * img[y, x + 1] + img[y + 1, x + 1]) - ₩
                           (img[y - 1, x - 1] + 2 * img[y, x - 1] + img[y + 1, x - 1])
        gy[y - 1, x - 1] = (img[y + 1, x - 1] + 2 * img[y + 1, x] + img[y + 1, x + 1]) - ₩
                           (img[y - 1, x - 1] + 2 * img[y - 1, x] + img[y - 1, x + 1])

gmag = np.sqrt(gx**2 +gy**2)
theta = np.arctan2(gy, gx)
theta = np.rad2deg(theta) % 180
```

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

그림 1. Sobel Mask

$$\mathbf{G} = \sqrt{\mathbf{G}_x^{\,2} + \mathbf{G}_y^{\,2}}$$

$$\mathbf{\Theta} = \operatorname{atan}\left(\frac{\mathbf{G}_y}{\mathbf{G}_x}\right)$$

그림 2. Magnitude and direction

- x와 y방향의 gradient를 계산하기 위해 [그림 1]과 같은 sobel mask를 적용한다.
- [그림 2]의 계산식을 이용하여 magnitude와 direction을 계산한다. 계산시에는 numpy 모듈에서 제공하는 sqrt와 arctan2 함수를 이용하였다.

Non-maximum suppression

```
# Apply non-maximum suppression
gmax = np.zeros(shape, dtype='float64')
for y in range(height):
    for x in range(width):
        try:
            if 22.5 <= theta[y,x] < 67.5:
                if gmag[y,x] > gmag[y-1,x+1] and gmag[y,x] > gmag[y+1,x-1]:
                    gmax[y,x] = gmag[y,x]
            elif 67.5 <= theta[y,x] < 112.5:
                if gmag[y,x] > gmag[y-1,x] and gmag[y,x] > gmag[y+1,x]:
                    gmax[y,x] = gmag[y,x]
            elif 112.5 <= theta[y,x] < 157.5:
                if gmag[y,x] > gmag[y-1,x-1] and gmag[y,x] > gmag[y+1,x+1]:
                    gmax[y,x] = gmag[y,x]
            else:
                if gmag[y,x] > gmag[y,x-1] and gmag[y,x] > gmag[y,x+1]:
                    gmax[y,x] = gmag[y,x]
        except IndexError as e:
            pass
```

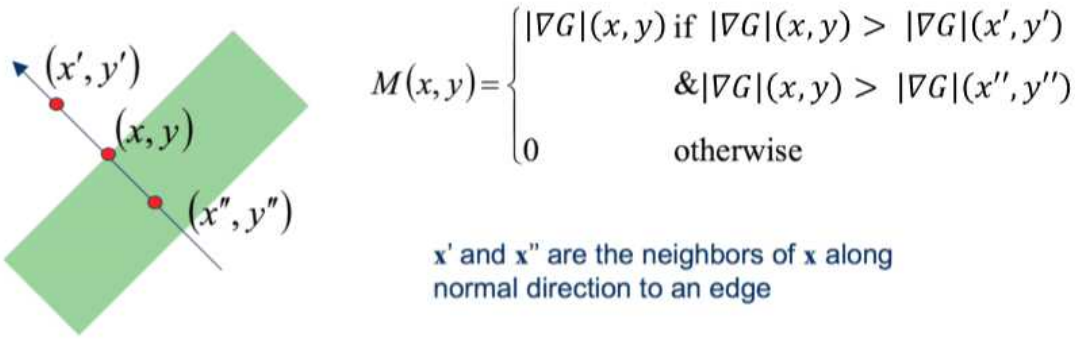

그림 3. Non-maximum suppression method

- [그림 3]의 방법을 통하여 non-maximum suppression을 수행한다.

Hysteresis thresholding

```
# Thresholding
high = 0.130
low = 0.065

line = np.zeros(shape, dtype='uint8')
line[np.where(gmax > high)] = 255
line[np.where((gmax >= low) & (gmax <= high))] = 50
```

- 두 개의 threshold(High, Low)를 정의하여, 각각의 edge를 strong/weak/no로 분류한다.
- threshold의 값은 실험을 통하여 주어진 data set에 대해 가장 높은 성능을 가지는 값을 선택하였다.

Connect edges

```
# Connecting edges
dy = (0, -1, -1, -1, 0, 1, 1, 1)
dx = (1, 1, 0, -1, -1, -1, 0, 1)
for y in range(height):
    for x in range(width):
        if line[y, x] == 50:
            try:
                for i in range(8):
                    if line[y + dy[i], x + dx[i]] == 255:
                        line[y, x] = 255
                        break
                else:
                    line[y, x] = 0
            except IndexError as e:
                pass
return line > 0
```

- strong edge를 연결하고, 해당 strong edge에 연결되 있는 weak edge들을 연결한다.
- 연결한 값을 1로하고, 연결되지 않은 값을 0으로 하는 boolean 형의 matrix를 return 한다.

2. 시험 및 평가 내용

A. Time Comsumption

```
# Debugging
if __name__ == "__main__":
    import os, time

    img = io.imread(os.path.join('images', '001.jpg'))
    t_start = time.time()
    res = edge(img)
    t_elapsed = time.time()-t_start
    GT = io.imread(os.path.join('gt', '001.jpg'), as_grey=True) > 0
    TP = np.bitwise_and(GT, res)
    nTP = TP.sum()
    nReal = GT.sum()
    nResponse = res.sum()
    precision = nTP / nResponse
    recall = nTP / nReal
    fmeasure = 2*precision*recall/(precision+recall)

    print(fmeasure)
    print(t_elapsed)
```

그림 4. 수행 시간 및 정확도 측정 예시 코드

1회	2회	3회	4회	5회	평균(sec)
1.6720	1.8595	1.6095	1.6564	1.6252	1.68452

표 1. 수행 시간 측정 예시 결과표

main 함수에 간단한 디버깅용 프로그램을 작성하여 001.jpg에 대하여 반복적으로 수행시간을 측정하였다. 수행간에 2초 내외의 시간이 소요되는 것을 확인하였으며 이는 프로젝트 수행기준인 60초 이내임을 확인하였다. 또한 grader.py를 수행함으로서 걸리는 시간이 총 498.83(sec)인 것을 확인하였다.

B. Accuracy

001.jpg	002.jpg	003.jpg	004.jpg	005.jpg	평균
0.144838	0.096590	0.217843	0.162271	0.188001	0.161909

표 2. 정확도 측정 예시 결과표

grader.py를 수행함으로서 각각의 이미지의 edge detection의 정확도를 계산하였다. 대체로 11~13% 정도의 정확도를 가장 많이 보였으며 실제 모든 data set을 통하여 계산한 정확도는 13.745% 임을 확인하였다.

Ⅳ. 기타

1. 느낀 점

수업시간에 다루었던 Canny Edge Detection을 이용하여 edge detection을 수행해 보았다. 실제 프로그램 작성시에 scipy와 numpy에 유용한 기능을 제공하는 함수가 많아서 생각보다 작성하기 수월하였다. 다만 정확도를 높이기 위해 직접 high와 low threshold를 바꾸어 가며 여러 번을 테스트하는 어려움이 있었다. 실제 test를 통하여 수행한 결과도 비교적 높지 않은 정확도를 가졌기 때문에 프로젝트 진행 간에 어려움이 많았다. 추후 다른 방법을 통하여 정확도를 더 높일 수 있는지에 대하여 알아보아야겠다는 필요성이 들었다.