디지털영상처리 프로젝트 (결과보고서)

프로젝트명	국문	어두운 영상의 개선 및 경계 추출 분석		
	영문	Analysis of improvement and boundary extraction methods for dark images		
프로젝트 수행팀	팀명	5인 이상 집합 금지		
	학번		성명	E-mail
	20193107		김성안	pcplove852@naver.com
	20193146		정현수	328ch@naver.com
	20193148		황진주	jinjoo021@naver.com
	20193166		남유정	yujeong0528@naver.com
	20193176		조은희	dmsgml4579@naver.com

< 1. 아이디어 발상 >

✓ **목표**

어두운 영상을 다양한 기법 투입으로 영상 밝기 개선 및 영상의 엣지 검출 후 최적안 모색

✓ **기술**

1. 밝기 및 시인성 개선

- 콘트라스트 조절
- 히스토그램 평활화 적용
- 전체 값 증가

2. 엣지 검출

- high pass filter : 고속 푸리에 변환 후 원점 부근의 저주파 대역을 삭제함으로 엣지 검출
- 그레디언트(gradient) 영상화 : 급등한 변화를 알 수 있는 미분과 이미지의 유사부분을 통합하는 스무딩을 기반으로 하여 나온 결과값을 통해 엣지 검출

✓ **입력 영상**

다양한 밝기의 어두운 그레이 스케일 영상

✓ **예상 결과**

- 어두운 영상 밝기 개선
- 밝기가 개선된 영상에 대한 엣지 검출

✓ **기대 효과**

지금까지 학습한 영상 처리 기법 중 다양한 방법의 엣지 검출 알고리즘을 동일한 영상에 적용해봄으로써 도출되는 각 방법의 장단점 등을 학습할 수 있고, 각기 다른 특징의 영상에 대해 최적화된 엣지 검출 알고리즘을 확인할 수 있다.

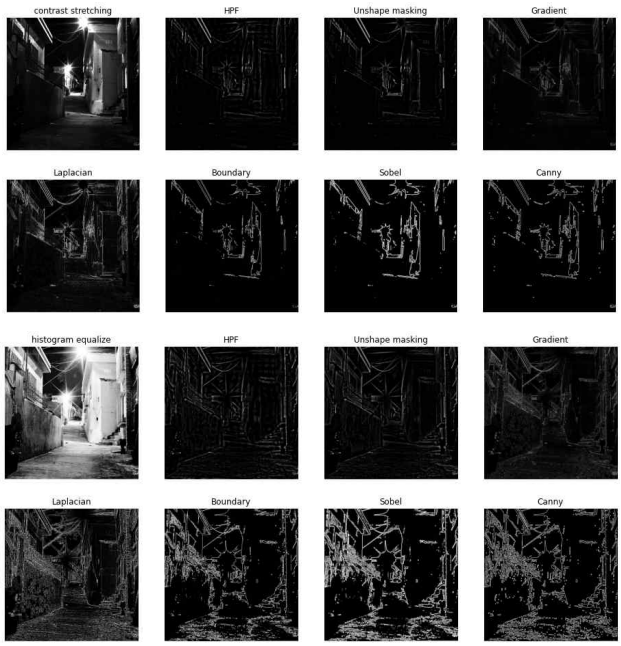
< 2. 프로젝트 수행 내용 >

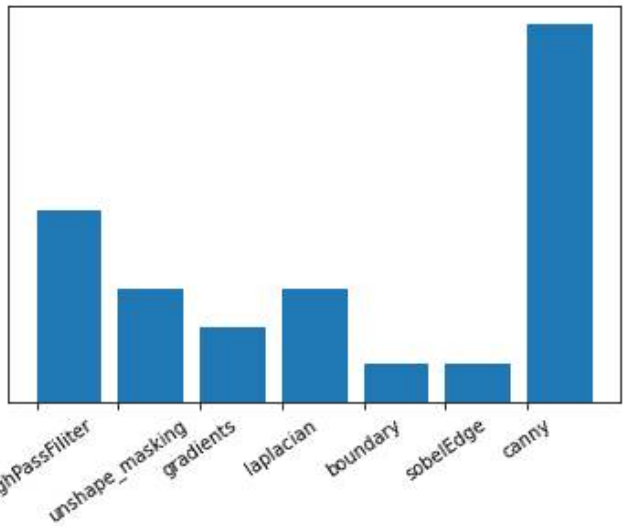
(조별 수행 시 개인별 수행 내용 작성, 개인 수행 시 자기 수행 내용 작성)

성명 / 학번		수행 내용
학번	성명	▪ Unsharp Masking 기법 함수 작성 ▪ Gradient 기법 함수 작성
20193107	김성안	
학번	성명	▪ main 함수 작성 ▪ Laplacian 기법 함수 작성
20193146	정현수	
학번	성명	▪ High Pass Filter 기법 함수 작성 ▪ Sobel 기법 함수 작성
20193148	황진주	
학번	성명	▪ Equalized 기법 함수 작성 ▪ Contrast 기법 함수 작성
20193166	남유정	
학번	성명	▪ Canny 기법 함수 작성 ▪ Boundary 기법 함수 작성
20193176	조은희	

< 3. 수행 목표 한 문장으로 >

어두운 영상의 밝기를 개선 후 엣지를 검출할 때, 가장 효율적인 방안 탐색





< 4. 구현 과정에 적용한 영상 처리 알고리즘 >

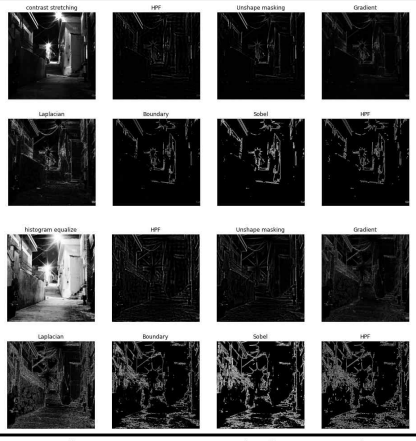
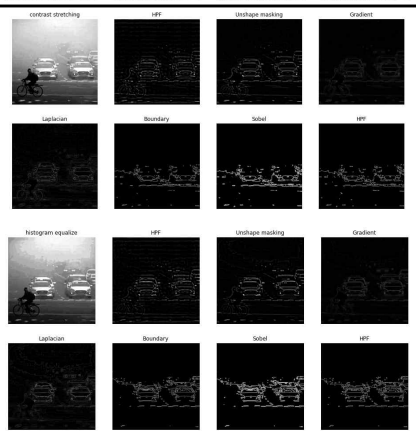
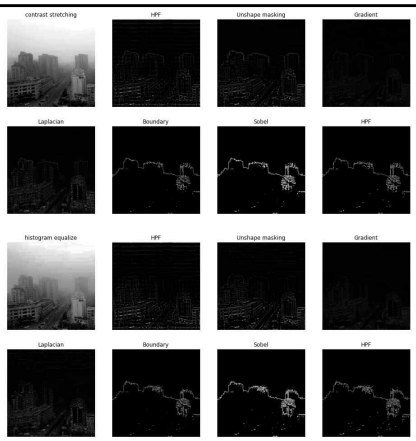
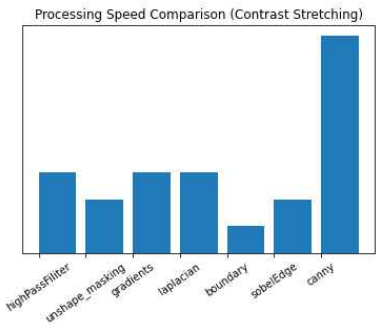
<table>
<tr><th>알고리즘 명</th><th>적용 방법 및 이유</th></tr>
<tr><td>Contrast</td><td>최적의 비교 환경 구성을 위해 정의된 함수 사용
콘트라스트 스트레칭에 의한 영상 개선 후 결과 비교를 위해 적용</td></tr>
<tr><td>Equalized</td><td>최적의 비교 환경 구성을 위해 정의된 함수 사용
히스토그램 평활화에 의한 영상 개선 후 결과 비교를 위해 적용</td></tr>
<tr><td>High Pass Filter</td><td rowspan="7">응용을 통해 직접 구현하여 적용하되,
구현이 다소 복잡하거나 직접 구현방식으로 인해 불필요한 연산속도의 증가가 우려되는 알고리즘에 한해 정의된 함수 이용

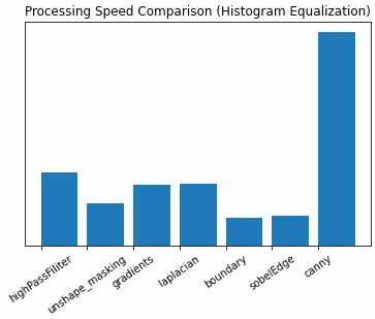
다양한 영상 에지 검출을 통한 개선도 side-by-side 방식으로 비교하기 위함

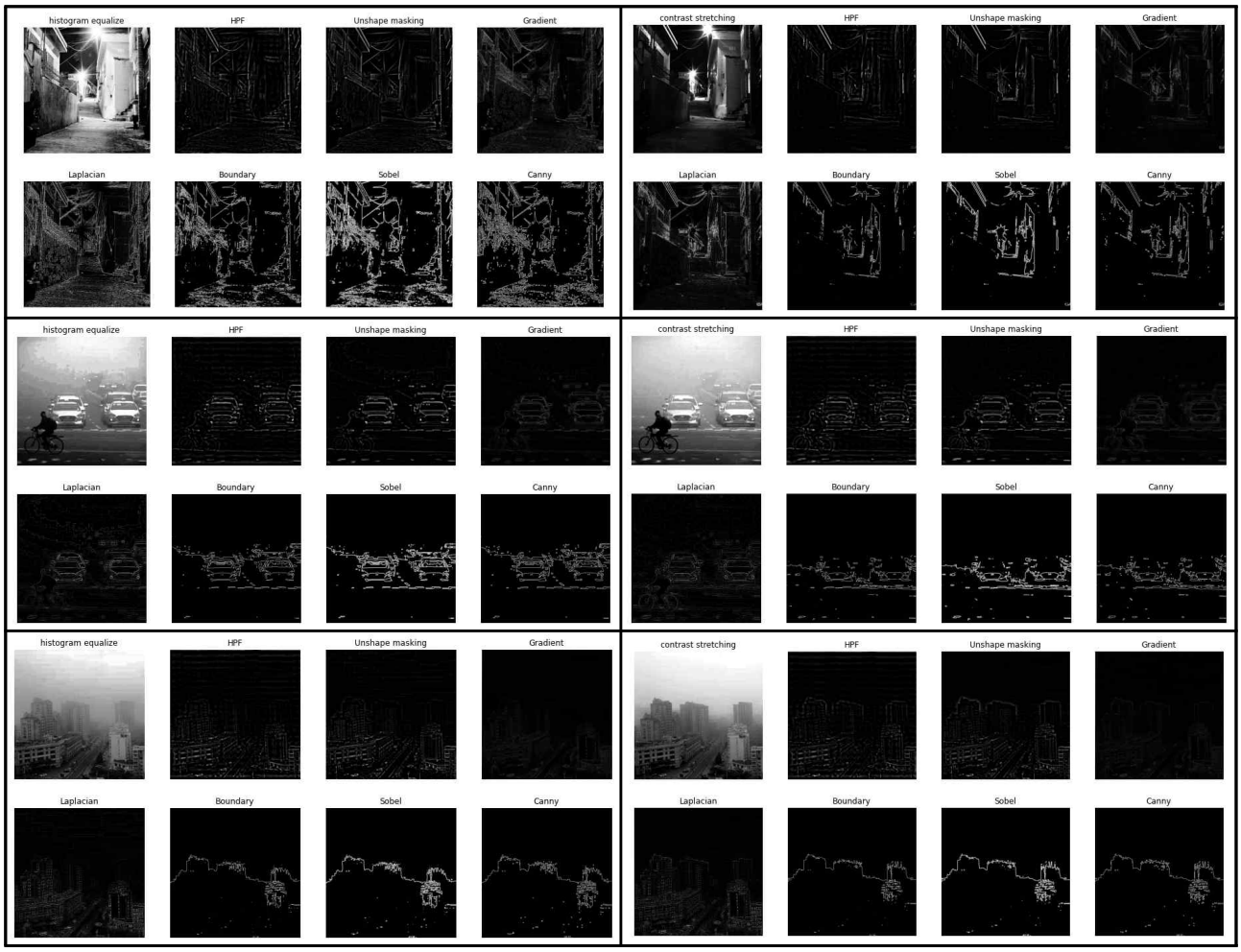
에지 검출까지의 연산 시간 및 결과물 분석으로 종합적인 효율성 비교 용도
개선도와 효율성을 기반으로 최선의 알고리즘 도출</td></tr>
<tr><td>Unsharp Masking</td></tr>
<tr><td>Boundary</td></tr>
<tr><td>Gradient</td></tr>
<tr><td>Laplacian</td></tr>
<tr><td>Sobel</td></tr>
<tr><td>Canny</td></tr>
</table>

< 5. 구현 결과 정리하기 >

출력영상의 확대 모습은 아래에 크게 삽입하였습니다.

입력영상	출력영상	결과분석
		평활화와 대비조정에 따른 결과 차이가 가장 극명하게 나타난 예로, 대비 조정 영상에 대해 라플라시안, 경계 변환, 캐니를 적용하면 뛰어난 엣지 검출 결과가 도출되었으나, 평활화 영상에 대해 이를 적용하면 잡음이 많이 발생하였음.
		평활화 영상의 엣지 검출 알고리즘 중 잡음이 적고 명확한 이미지를 도출해낸 것은 라플라시안과 소벨이었음. 대비조정 영상에 대해서는 라플라시안 방법이 가장 잡음이 적고 명확한 결과를 도출해냄.
		그래디언트 방식은 양쪽 모든 변환에서 오히려 영상식별이 어려워졌음을 확인할 수 있었음. HPF, 언샤프닝, 그래디언트, 라플라시안은 건물에 대한 엣지가 생성되었고, 경계추출, 소벨, 캐니는 하늘과 건물사이의 경계가 형성되었음.
위 3가지 영상의 결과도출 시간		대비 조정 영상에 대해 각 엣지 검출 동작 시간을 계산하여 시각화한 결과, 캐니 알고리즘이 훌륭한 결과물을 도출함과 동시에 처리 속도가 가장 길다는 점을 확인할 수 있었음.

입력영상	출력영상	결과분석
위 3가지 영상의 결과도출 시간		평활화 영상에 대해 각 엣지 검출 동작 시간을 계산하여 시각화한 결과, 대비 조정 영상에 대한 엣지 검출 동작 속도에는 유의미한 차이가 없음을 확인할 수 있었음.



< 6. 구현 코드 첨부하기 >

```
import numpy as np
from scipy import signal, ndimage
from skimage import filters, feature, img_as_float, exposure
from skimage.io import imread
from skimage.color import rgb2gray
import matplotlib.pylab as plt
import datetime
from skimage.morphology import binary_erosion

# sobelEdge 사용 시 필요
import scipy.fftpack as fp

# 히스토그램 평활화
def equalize_func(im):
    start_time = datetime.datetime.now()
    hist = exposure.equalize_hist(im)
    end_time = datetime.datetime.now()
    elapsed_time = end_time - start_time
    return hist, elapsed_time.microseconds / 1000

# 콘트라스트 스트레칭
def contrast_str_func(im):
    start_time = datetime.datetime.now()
    from_, to_ = np.percentile(im, (2, 98))
    cont_ = exposure.rescale_intensity(im, in_range=(from_, to_))
    end_time = datetime.datetime.now()
    elapsed_time = end_time - start_time
    return cont_, elapsed_time.microseconds / 1000

# 1. HPF
def highPassFiliter(im, t):
  start_time = datetime.datetime.now()
  freq = fp.fft2(im) # 고속 fft변환
  (w, h) = freq.shape # 너비값 저장
  half_w, half_h = int(w/2), int(h/2) # 중앙값 계산
  freq1 = np.copy(freq) # fft 변환값 저장
  freq2 = fp.fftshift(freq1) # fft 변환값 쉬프팅
  # 하이패스 필터 적용
  freq2[half_w-20:half_w+21,half_h-20:half_h+21] = 0 # 저주파(스펙트럼 중앙 부분)대역 삭제
  im1 = np.clip(fp.ifft2(fp.ifftshift(freq2)).real,0,255) # FFT 후 허수값을 제외한 0에서 255값의 변환이미지 값 생성
  end_time = datetime.datetime.now()
  elapsed_time = end_time - start_time
  return im1, elapsed_time.microseconds / 1000

# 2. unshape masking
def unshape_masking(im, t):
    start_time = datetime.datetime.now()
    im = rgb2gray(img_as_float(im))  # skimage라이브러리의 함수 rgb2gray()를 통한 이진화
    im_blurred = ndimage.gaussian_filter(im, 5)     # 가우시안필터를 이용하여 블러링처리
    im_detail = np.clip(im - im_blurred, 0, 1)      # 원본이미지에서 블러링한 이미지를 빼서 마스크 생성
    #im_sharp = np.clip(im + 5*im_detail, 0, 1)    # 마스크를 영상에 더한다(가중치 =5, 가장 원본과 차이가 큼)
    end_time = datetime.datetime.now()
    elapsed_time = end_time - start_time
    return im_detail, elapsed_time.microseconds / 1000
```

```
# 3. 그레디언트
def gradients_func(im, t):
    start_time = datetime.datetime.now()
    ker_x = [[-1, 1]]                    #컨볼루션을 위한 커널 설정
    ker_y = [[-1], [1]]
    g_im = rgb2gray(im)          #skimage라이브러리의 함수 rgb2gray()를 통한 이진화
    im_x = signal.convolve2d(g_im, ker_x, mode='same')  #scipy 패키지 제공 convolve2d ()는 다음의 파라미터로 컨벌루션을 달성하기 위해 기능
    im_y = signal.convolve2d(g_im, ker_y, mode='same')  # 커널을 이용한 컨볼루션을 통해 경계선을 탐지
    im_mag = np.sqrt(im_x**2 + im_y**2)                    #그래디언트의 크기 계산
    end_time = datetime.datetime.now()
    elapsed_time = end_time - start_time
    return im_mag, elapsed_time.microseconds / 1000

# 4. 라플라시안
def laplacian_func(img, t):
    start_time = datetime.datetime.now()
    laplac_kern = [[1,1,1],[1,-8,1],[1,1,1]] # 라플라시안 필터 생성
    result = np.clip(signal.convolve2d(img, laplac_kern, mode='same'), 0, 1) #라플라시안 필터로 컨볼루션 적용
    end_time = datetime.datetime.now()
    elapsed_time = end_time - start_time
    return result, elapsed_time.microseconds / 1000

# 5. 경계 추출
def boundary_func(im, t):   #경계 추출
    start_time = datetime.datetime.now()
    threshold = 0.5 # 임계값 설정
    im[im<threshold], im[im >= threshold] = 0, 1 # 임계값보다 작은 값은 0, 크거나 같은 값은 1
    boundary = im - binary_erosion(im) # 경계를 추출하기 위해 이진 영상에서 침식된 영상을 뺀다
    result = np.array(boundary)
    end_time = datetime.datetime.now()
    elapsed_time = end_time - start_time
    return result, elapsed_time.microseconds / 1000

# 6. Sobel 엣지
def sobelEdge_func(im, t):
    start_time = datetime.datetime.now()
    edges = filters.sobel(im) # 소벨에지 적용
    end_time = datetime.datetime.now()
    elapsed_time = end_time - start_time
    return edges, elapsed_time.microseconds / 1000

# 7. 캐니
def canny_func(im, t):
    start_time = datetime.datetime.now()
    canny = feature.canny(im, 1) # 캐니 적용
    result = np.array(canny)
    end_time = datetime.datetime.now()
    elapsed_time = end_time - start_time
    return result, elapsed_time.microseconds / 1000
```

```
im = []
for i in range(3):
    path_ = './test/' + str(i+1) +'.jpg'
    im.append(rgb2gray(imread(path_)))



hist_equ = []
cont_img = []
t_tmp1 = [0 for i in range(3)]
t_tmp2 = [0 for i in range(3)]
for i in range(3):
    img_t, t_tmp1[i] = equalize_func(im[i])
    hist_equ.append(img_t)
    img_t, t_tmp2[i] = contrast_str_func(im[i])
    cont_img.append(img_t)
histo_speed_mean = np.mean(t_tmp1)
cont_speed_mean = np.mean(t_tmp2)

result_img = []
tmp = 0

time_ = [];
time_temp = 0;
for j in range(2):
    if j == 0:
        img_temp = hist_equ
        title = 'histogram equalize'
    if j == 1:
        img_temp = cont_img
        title = 'contrast stretching'
    for k in range(3):
        fig, axes=plt.subplots(2, 4, figsize=(16,8))
        axes[0][0].imshow(img_temp[k], cmap=plt.cm.gray)
        tmp, time_temp = highPassFiliter(img_temp[k], time_temp)
        axes[0][1].imshow(tmp, cmap=plt.cm.gray)
        time_.append(round(time_temp*1000, 2))
        tmp, time_temp = unshape_masking(img_temp[k], time_temp)
        axes[0][2].imshow(tmp, cmap=plt.cm.gray)
        time_.append(round(time_temp*1000, 2))
        tmp, time_temp = gradients_func(img_temp[k], time_temp)
        axes[0][3].imshow(tmp, cmap=plt.cm.gray)
        time_.append(round(time_temp*1000, 2))
        tmp, time_temp = laplacian_func(img_temp[k], time_temp)
        axes[1][0].imshow(tmp, cmap=plt.cm.gray)
        time_.append(round(time_temp*1000, 2))
        tmp, time_temp = boundary_func(img_temp[k], time_temp)
        axes[1][1].imshow(tmp, cmap=plt.cm.gray)
        time_.append(round(time_temp*1000, 2))
        tmp, time_temp = sobelEdge_func(img_temp[k], time_temp)
        axes[1][2].imshow(tmp, cmap=plt.cm.gray)
        time_.append(round(time_temp*1000, 2))
        tmp, time_temp = canny_func(img_temp[k], time_temp)
        axes[1][3].imshow(tmp, cmap=plt.cm.gray)
        time_.append(round(time_temp*1000, 2))
```

```
            axes[0][0].axis('off')
            axes[0][1].axis('off')
            axes[0][2].axis('off')
            axes[0][3].axis('off')
            axes[1][0].axis('off')
            axes[1][1].axis('off')
            axes[1][2].axis('off')
            axes[1][3].axis('off')

            axes[0][0].set_title(title)
            axes[0][1].set_title('HPF')
            axes[0][2].set_title('Unshape masking')
            axes[0][3].set_title('Gradient')
            axes[1][0].set_title('Laplacian')
            axes[1][1].set_title('Boundary')
            axes[1][2].set_title('Sobel')
            axes[1][3].set_title('HPF')
            plt.show()

sum_temp = [[0 for col in range(7)] for row in range(2)]
for i in range(2):
  for j in range(7):
    sum_temp[i][j] = float(0.0)
for i in range(42):
    if i < 21:
        sum_temp[0][i%7] += time_[i]
    else:
        sum_temp[1][i%7] += time_[i]

for j in range(2):
    for k in range(7):
      sum_temp[j][k] = sum_temp[j][k] / 3.0
      if j == 0: sum_temp += t_tmp1
      if j == 1: sum_temp += t_tmp2

text_ = []
text_.append("highPassFiliter")
text_.append("unshape_masking")
text_.append("gradients")
text_.append("laplacian")
text_.append("boundary")
text_.append("sobelEdge")
text_.append("canny")

x = []
x = np.arange(7)
plt.bar(x, sum_temp[0], align='edge')
plt.xticks(x, text_)
plt.xticks(rotation=35)
plt.show()

x = np.arange(7)
plt.bar(x, sum_temp[1], align='edge')
plt.xticks(x, text_)
plt.xticks(rotation=35)
plt.show()
```