컴퓨터비전
실습 1

Longbin Jin

2020.10

Artificial Inteligence
& Computer Vision
Laboratory

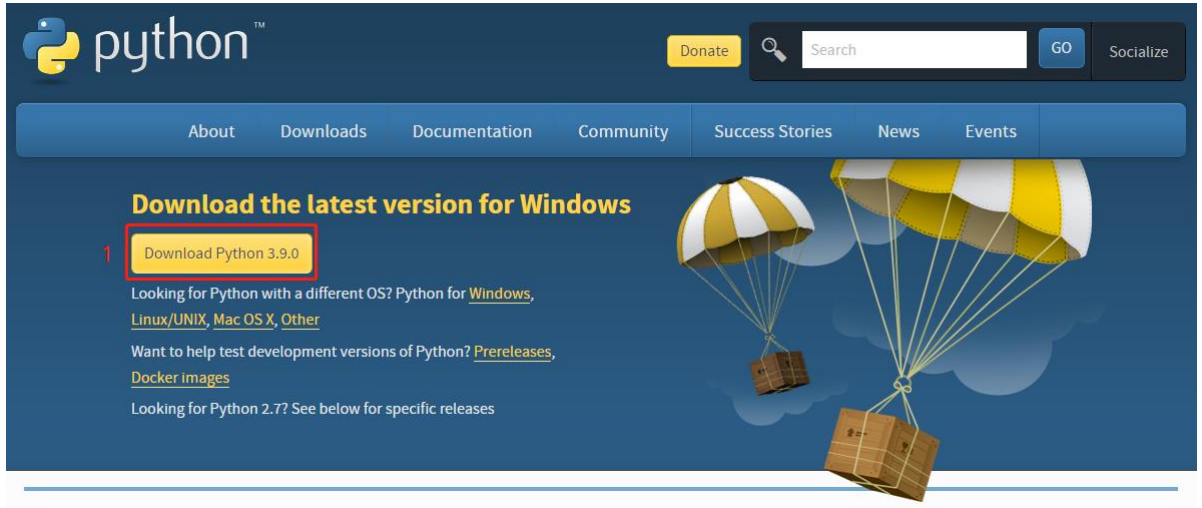
Index

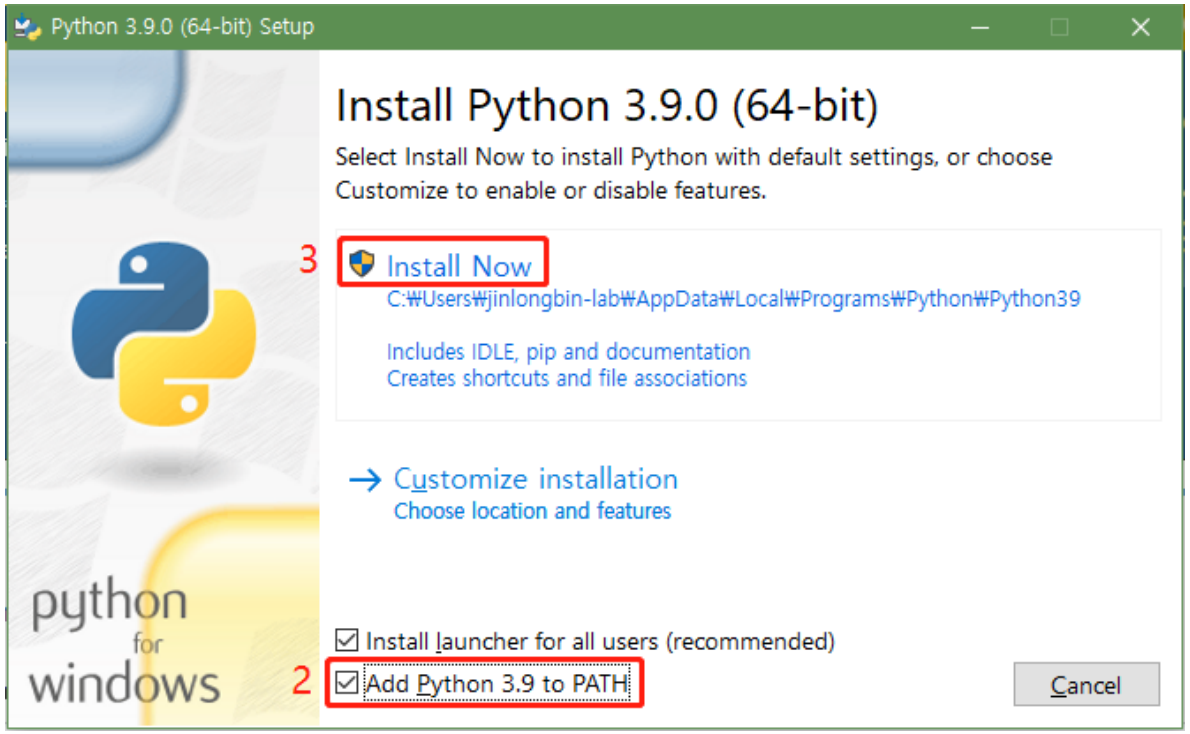
- Python / 모듈 설치

- GLCM
- Law’s Energy
- Harris corner
- SIFT
- Kmeans

Python 설치

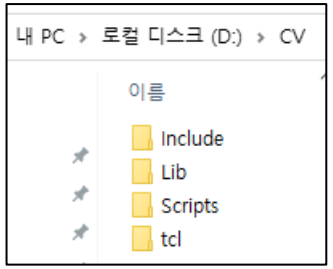
- https://www.python.org/downloads/
- Python 3.x 버전 모두 가능 (예전에 설치했으면 새로 설치 안 해도 됨)





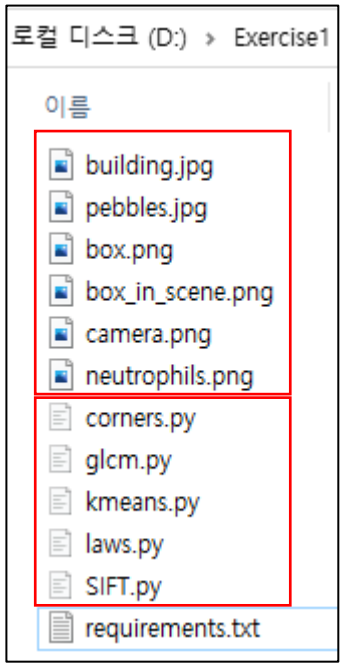
가상환경 만들기

- 명령 프롬프트(window) 혹은 Terminal (maxos, linux) 실행
- 가상환경 모듈 설치
  - `C:₩Users₩jinlongbin-lab>pip install virtualenv`
- D드라이브에 CV 라는 이름을 가진 가상환경 만들기
  - ```
    C:₩Users₩jinlongbin-lab>d:

    D:₩>virtualenv CV
    ```
  - 

    CV 폴더가 생성된다
  - `D:₩>D:₩CV₩Scripts₩activate.bat` 가상환경 활성화
  - `(CV) D:₩>` 활성화 성공
    ```

가상환경내 모듈 설치

- `(CV) D:₩>pip install -r D:₩Exercise1₩requirements.txt`
- 

실습에서 사용할 이미지

실습을 통해 만들 python 파일

numpy
matplotlib
scikit-image
scipy
opencv-python==3.4.2.16
opencv-contrib-python==3.4.2.16

GLCM (glcm.py)

```python
import matplotlib.pyplot as plt
import cv2
from skimage.feature import greycomatrix, greycoprops
import numpy as np

# === 이미지 읽기 ===
image = cv2.imread('camera.png', cv2.IMREAD_GRAYSCALE)                # 이미지를 흑백으로 불러오기

# === 이미지에서 풀과 하늘 영역 짤라내기 ===
PATCH_SIZE = 21                                                       # 이미지에서 짤라낼 영역 너비와 길이 (너비 = 길이 = PATCH_SIZE)

grass_locations = [(370, 454), (372, 22), (444, 244), (455, 455)]     # 이미지에서 풀 부분 위치 선정, 이미지 좌측 상단이 원점 기준 (y축, x축)
grass_patches = list()                                                # 각 위치에서 짤라낸 부분들을 grass_patches list에 저장
for loc in grass_locations:
    grass_patches.append(image[loc[0]:loc[0] + PATCH_SIZE,
                               loc[1]:loc[1] + PATCH_SIZE])

sky_locations = [(38, 34), (139, 28), (37, 437), (145, 379)]          # 이미지에서 하늘 부분 위치 선정, 이미지 좌측 상단이 원점 기준 (y축, x축)
sky_patches = list()                                                  # 각 위치에서 짤라낸 부분들을 sky_patches list에 저장
for loc in sky_locations:
    sky_patches.append(image[loc[0]:loc[0] + PATCH_SIZE,
                             loc[1]:loc[1] + PATCH_SIZE])

# === 짤라낸 풀과 하늘 영역에서 GLCM dissimilarity와 correlation을 계산하기 ===
xs = list()                                                           # dissimilarity(entropy)값 저장할 list
ys = list()                                                           # correlation값 저장할 list
for patch in (grass_patches + sky_patches):
    glcm = greycomatrix(patch, distances=[1], angles=[0], levels=256,  # GLCM co-occurrence 계산
                        symmetric=False, normed=True)
    xs.append(greycoprops(glcm, 'dissimilarity')[0, 0])               # GLCM dissimilarity(entropy)값 계산하여 xs에 저장
    ys.append(greycoprops(glcm, 'correlation')[0, 0])                 # GLCM correlation 계산하여 ys에 저장
                                                                      # 그외에도 다른 feature 계산 가능
                                                                      # {'contrast', 'dissimilarity', 'homogeneity',
                                                                      # 'energy', 'correlation', 'ASM'}

```

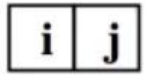
GLCM

1	1	0	0
1	1	0	0
0	0	2	2
0	0	2	2

Image I

$C_{(0,1)}$

i \ j	0	1	2
0	4	0	2
1	2	2	0
2	0	0	2




```
glcm = greycomatrix(patch, distances=[1], angles=[0], levels=3,
                    symmetric=False, normed=False)
```

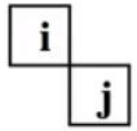
$C_{(1,0)}$

i \ j	0	1	2
0	4	0	2
1	2	2	0
2	0	0	2


```
glcm = greycomatrix(patch, distances=[1], angles=[np.pi/2], levels=3,
                    symmetric=False, normed=False)
```

$C_{(1,1)}$

i \ j	0	1	2
0	2	0	2
1	2	1	1
2	0	0	1

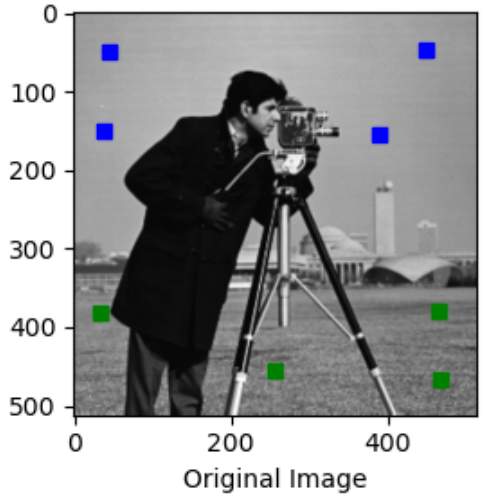


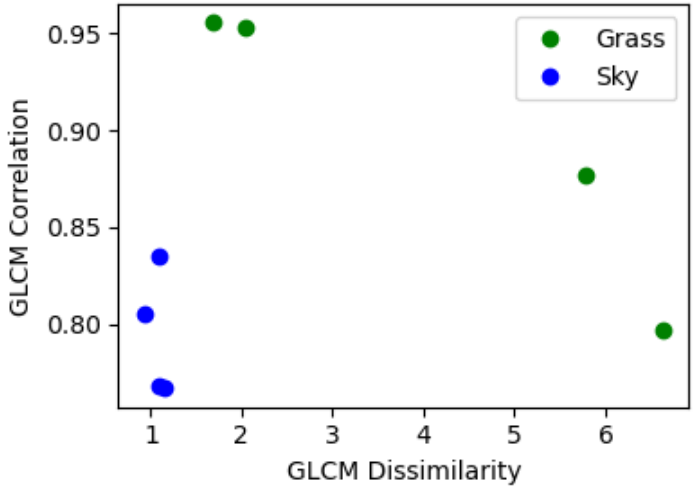

```
glcm = greycomatrix(patch, distances=[1], angles=[np.pi/4], levels=3,
                    symmetric=False, normed=False)
```

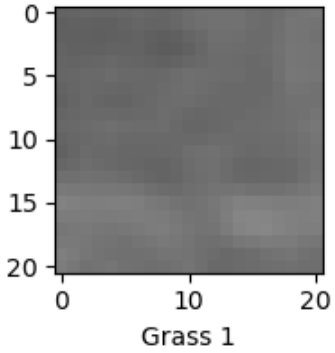
GLCM (glcm.py)

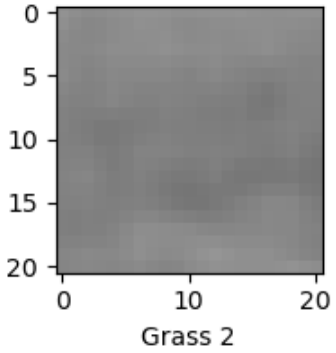
```
36  # === 결과 시각화 ===
37  fig = plt.figure(figsize=(8, 8))                    # 그림판(백지) 만들기
38
39  ax = fig.add_subplot(3, 2, 1)                       # 그림판을 3행2열로 나누고 1번 영역에 그림그리기
40  ax.imshow(image, cmap=plt.cm.gray, vmin=0, vmax=255)  # 흑백이미지 추가
41  for (y, x) in grass_locations:                      # 풀에 해당하는 위치
42      ax.plot(x + PATCH_SIZE / 2, y + PATCH_SIZE / 2, 'gs')  # 중심에 초록색 네모(green square) 추가
43  for (y, x) in sky_locations:                        # 하늘에 해당하는 위치
44      ax.plot(x + PATCH_SIZE / 2, y + PATCH_SIZE / 2, 'bs')  # 중심에 하늘색 네모(blue square) 추가
45  ax.set_xlabel('Original Image')                     # x축 이름을 Original Image으로 지정
46
47  ax = fig.add_subplot(3, 2, 2)                       # 그림판을 3행2열로 나누고 2번 영역에 그림그리기
48  ax.plot(xs[:len(grass_patches)], ys[:len(grass_patches)], 'go',  # dissimilarity값을 x축, correlation값을 y축인 지점에 초록색 점추가
49          label='Grass')
50  ax.plot(xs[len(sky_patches):], ys[len(sky_patches):], 'bo',      # dissimilarity값을 x축, correlation값을 y축인 지점에 하늘색 점추가
51          label='Sky')
52  ax.set_xlabel('GLCM Dissimilarity')                 # x축 이름을 GLCM Dissimilarity으로 지정
53  ax.set_ylabel('GLCM Correlation')                   # y축 이름을 GLCM Correlation으로 지정
54  ax.legend()                                         # 라벨 이름 추가
55
56
57  for i, patch in enumerate(grass_patches):           # 각 풀 영역마다
58      ax = fig.add_subplot(3, len(grass_patches),     # 그림판을 3행4열로 나누고
59                           len(grass_patches)*1 + i + 1)  # 5,6,7,8번 영역에 그림그리기
60      ax.imshow(patch, cmap=plt.cm.gray,              # 풀 영역 추가하기
61                vmin=0, vmax=255)
62      ax.set_xlabel('Grass %d' % (i + 1))             # 각 풀 영역 이름 지정
63
64  for i, patch in enumerate(sky_patches):             # 각 하늘 영역마다
65      ax = fig.add_subplot(3, len(sky_patches),       # 그림판을 3행4열로 나누고
66                           len(sky_patches)*2 + i + 1)  # 9,10,11,12번 영역에 그림그리기
67      ax.imshow(patch, cmap=plt.cm.gray,              # 하늘 영역 추가하기
68                vmin=0, vmax=255)
69      ax.set_xlabel('Sky %d' % (i + 1))               # 각 하늘 영역 이름 지정
70
71  plt.tight_layout()                                  # 여백공간 설정
72  plt.show()                                          # 그림 시각화
```

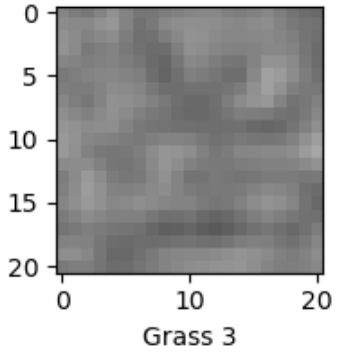
GLCM

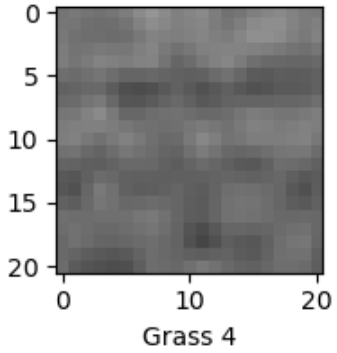

0
100
200
300
400
500
0
200
400
Original Image

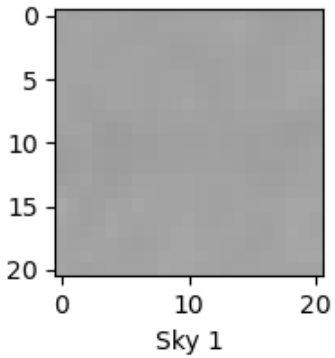

Grass
Sky
GLCM Correlation
0.95
0.90
0.85
0.80
1
2
3
4
5
6
GLCM Dissimilarity

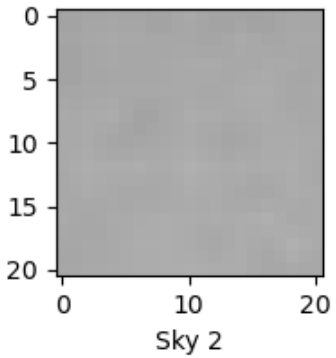

0
5
10
15
20
0
10
20
Grass 1

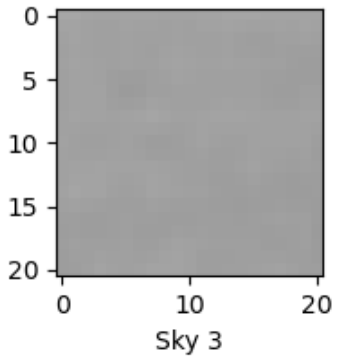

0
5
10
15
20
0
10
20
Grass 2

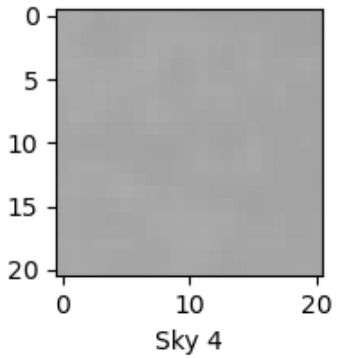

0
5
10
15
20
0
10
20
Grass 3


0
5
10
15
20
0
10
20
Grass 4


0
5
10
15
20
0
10
20
Sky 1


0
5
10
15
20
0
10
20
Sky 2

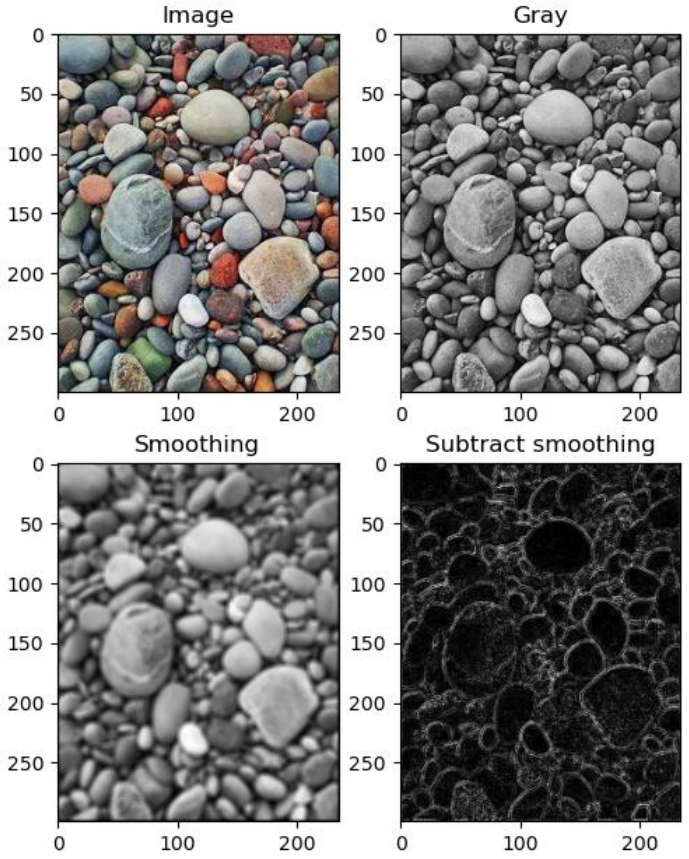

0
5
10
15
20
0
10
20
Sky 3


0
5
10
15
20
0
10
20
Sky 4

Law’s Energy (laws.py)

```python
import numpy as np
import cv2
from matplotlib import pyplot as plt
from scipy import signal as sg

# === 이미지 읽기 ===
image = cv2.imread('pebbles.jpg')                    # 이미지를 컬러(BGR)로 불러오기
image2 = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)      # BGR이미지를 RGB이미지로 변환
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)       # BGR이미지를 흑백이미지로 변환
(rows, cols) = gray.shape[:2]                        # 이미지의 길이(rows)와 너비(cols)

# === 이미지 전처리 ===
smooth_kernel = (1/25)*np.ones((5,5))                # smoothing filter 만들기
gray_smooth = sg.convolve(gray ,smooth_kernel,"same")# 흑백이미지 smoothing하기
                                                     # 출력이미지 사이즈 = 입력이미지 사이즈 (same)
gray_processed = np.abs(gray - gray_smooth)          # 원본이미지에서 smoothing된 이미지 빼기

# === 전처리된 이미지 시각화 ===
plt.figure('Image pre-processing')                   # 그림판(백지) 만들기
plt.subplot(221)                                     # 그림판을 2행2열로 나누고 1번 영역에 그림그리기
plt.title('Image')                                   # 이름 지정
plt.imshow(image2)                                   # RGB이미지 추가하기
plt.subplot(222)                                     # 그림판을 2행2열로 나누고 2번 영역에 그림그리기
plt.title('Gray')                                    # 이름 지정
plt.imshow(gray,'gray')                              # 흑백이미지 추가하기
plt.subplot(223)                                     # 그림판을 2행2열로 나누고 3번 영역에 그림그리기
plt.title('Smoothing')                               # 이름 지정
plt.imshow(gray_smooth,'gray')                       # smoothing된 이미지 추가
plt.subplot(224)                                     # 그림판을 2행2열로 나누고 4번 영역에 그림그리기
plt.title('Subtract smoothing')                      # 이름 지정
plt.imshow(gray_processed,'gray')                    # smoothing이미지를 뺀 이미지 추가
```

Law’s Energy (laws.py)



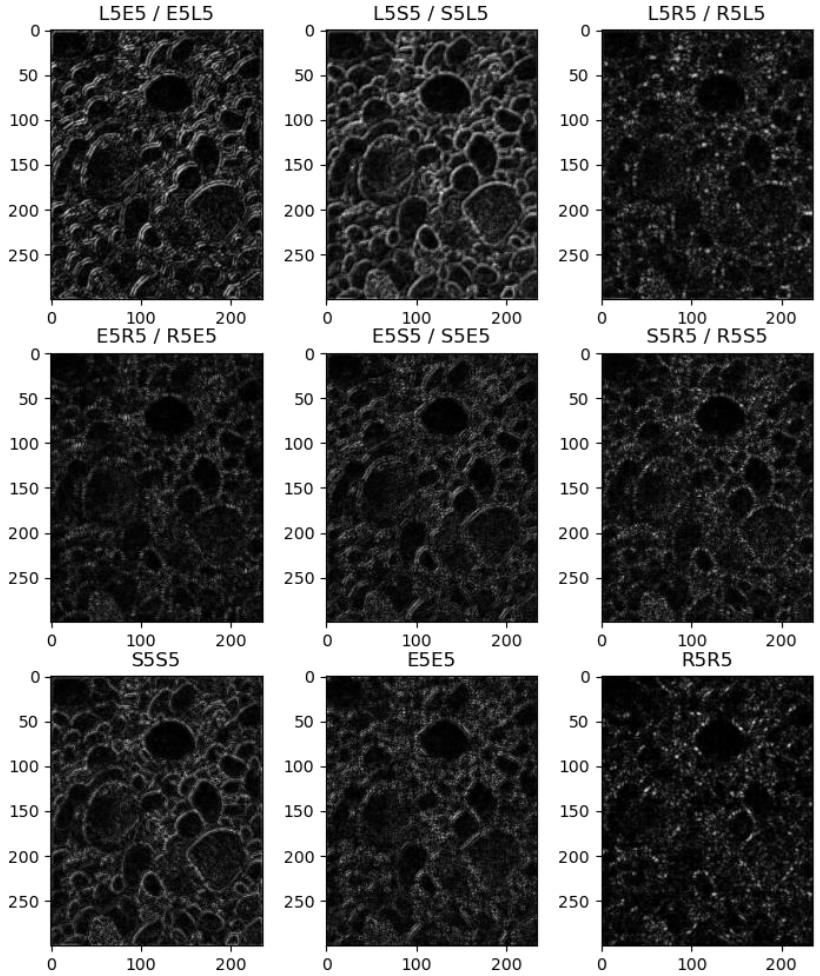
Law’s Energy (laws.py)

```
33  # === Law's Texture filter ===
34  filter_vectors = np.array([[ 1,  4,  6,  4, 1],                    # L5
35                             [-1, -2,  0,  2, 1],                    # E5
36                             [-1,  0,  2,  0, 1],                    # S5
37                             [ 1, -4,  6, -4, 1]])                   # R5
38
39                                                                     # 0:L5L5, 1:L5E5, 2:L5S5, 3:L5R5,
40                                                                     # 4:E5L5, 5:E5E5, 6:E5S5, 7:E5R5,
41                                                                     # 8:S5L5, 9:S5E5, 10:S5S5, 11:S5R5,
42                                                                     # 12:R5L5, 13:R5E5, 14:R5S5, 15:R5R5
43  filters = list()                                                   # 16(4x4)개 filter를 저장할 filters
44  for i in range(4):
45      for j in range(4):
46          filters.append(np.matmul(filter_vectors[i][:].reshape(5,1), # 매트릭스 곱하기 연산을 통해 filter값 계산
47                                   filter_vectors[j][:].reshape(1,5)))
48
49  # === Convolution연산 및 convmap조합 ===
50  conv_maps = np.zeros((rows, cols,16))                              # 계산된 convolution 결과를 저장할 conv_maps
51  for i in range(len(filters)):
52      conv_maps[:, :, i] = sg.convolve(gray_processed,               # 전처리된 이미지에 16개 필터 적용
53                                       filters[i],'same')
54
55  # === 9+1개 중요한 texture map 계산 ===
56  texture_maps = list()
57  texture_maps.append((conv_maps[:, :, 1]+conv_maps[:, :, 4])//2)    # L5E5 / E5L5
58  texture_maps.append((conv_maps[:, :, 2]+conv_maps[:, :, 8])//2)    # L5S5 / S5L5
59  texture_maps.append((conv_maps[:, :, 3]+conv_maps[:, :, 12])//2)   # L5R5 / R5L5
60  texture_maps.append((conv_maps[:, :, 7]+conv_maps[:, :, 13])//2)   # E5R5 / R5E5
61  texture_maps.append((conv_maps[:, :, 6]+conv_maps[:, :, 9])//2)    # E5S5 / S5E5
62  texture_maps.append((conv_maps[:, :, 11]+conv_maps[:, :, 14])//2)  # S5R5 / R5S5
63  texture_maps.append(conv_maps[:, :, 10])                           # S5S5
64  texture_maps.append(conv_maps[:, :, 5])                            # E5E5
65  texture_maps.append(conv_maps[:, :, 15])                           # R5R5
66  texture_maps.append(conv_maps[:, :, 0])                            # L5L5 (use to norm TEM)
67
68  # === Law's texture energy 계산 ===
69  TEM = list()
70  for i in range(9):
71      TEM.append(np.abs(texture_maps[i]).sum() /                     # TEM계산 및 L5L5 값으로 정규화
72                        np.abs(texture_maps[9]).sum())
73  print(TEM)                                                         # 9차원의 TEM feature 추출
```

Law’s Energy (laws.py)

```
75  # === 결과 시각화 ===
76  def norm(ar):
77      """ Convolution된 이미지를 0~255로 정규화 """
78      return 255.*np.absolute(ar)/np.max(ar)
79
80  plt.figure('Texture Maps')                    # 그림판(백지) 만들기
81  plt.subplot(331)                              # 그림판을 3행3열로 나누고 1번 영역에 그림그리기
82  plt.title('L5E5 / E5L5')                      # 이름 지정
83  plt.imshow(norm(texture_maps[0]),'gray')      # L5E5 / E5L5 filter로 처리한 이미지를 흑백으로 추가하기
84  plt.subplot(332)
85  plt.title('L5S5 / S5L5')
86  plt.imshow(norm(texture_maps[1]),'gray')
87  plt.subplot(333)
88  plt.title('L5R5 / R5L5')
89  plt.imshow(norm(texture_maps[2]),'gray')
90  plt.subplot(334)
91  plt.title('E5R5 / R5E5')
92  plt.imshow(norm(texture_maps[3]),'gray')
93  plt.subplot(335)
94  plt.title('E5S5 / S5E5')
95  plt.imshow(norm(texture_maps[4]),'gray')
96  plt.subplot(336)
97  plt.title('S5R5 / R5S5')
98  plt.imshow(norm(texture_maps[5]),'gray')
99  plt.subplot(337)
100 plt.title('S5S5')
101 plt.imshow(norm(texture_maps[6]),'gray')
102 plt.subplot(338)
103 plt.title('E5E5')
104 plt.imshow(norm(texture_maps[7]),'gray')
105 plt.subplot(339)
106 plt.title('R5R5')
107 plt.imshow(norm(texture_maps[8]),'gray')
108 plt.show()                                    # 그림 시각화
```

Law’s Energy (laws.py)



TEM:
[0.0701863469163514 4,
0.126647174237767 45,
0.106912667397733 38,
0.037309314591146 225,
0.015616353222989 132,
0.031591729380653 535,
0.019701832161771 134,
0.025009817044985 554,
0.109398946816118 6]

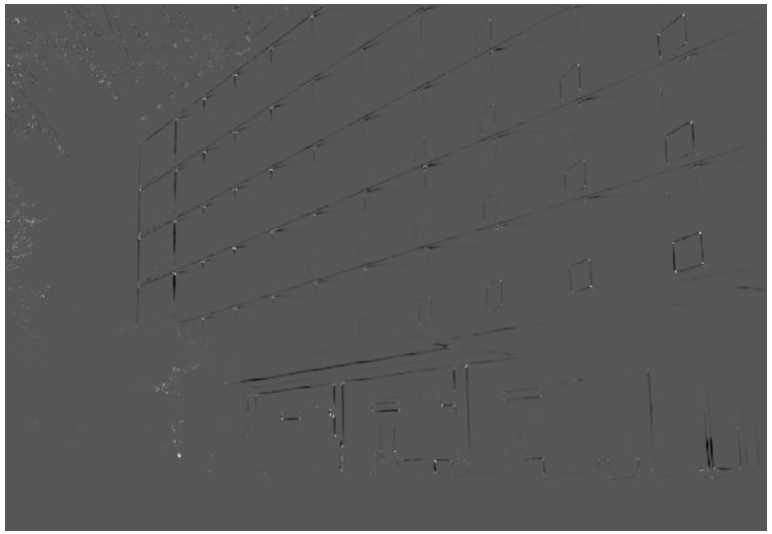
Harris corner (corner.py)

```python
import numpy as np
import cv2

image = cv2.imread('building.jpg', cv2.IMREAD_GRAYSCALE)                    # 이미지를 흑백으로 불러오기

harris = cv2.cornerHarris(image, blockSize=3, ksize=3, k=0.04)              # harris corner R값 계산
                                                                            # blockSize: size of neighbourhood considered for corner detection
                                                                            # ksize: Aperture parameter of Sobel derivative used
                                                                            # k: Harris detector free parameter in the equation.
harris_norm = cv2.normalize(harris, None, 0, 255, cv2.NORM_MINMAX, cv2.CV_8U)   # 0~255 사이로 정규화

image2 = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)                            # 흑백이미지를 BGR이미지로 변환

for y in range(1, harris_norm.shape[0]-1):
    for x in range(1, harris_norm.shape[1]-1):
        if harris_norm[y, x] > 120:                                         # R값이 120보다 크면
            if (harris[y, x] > harris[y-1, x] and                           # 해당 픽셀 R값이 주위 4개 픽셀 R값보다 크면
                    harris[y, x] > harris[y+1, x] and
                    harris[y, x] > harris[y, x-1] and
                    harris[y, x] > harris[y, x+1]):
                cv2.circle(image2, (x, y), radius=5, color=(0, 0, 255),     # 빨간색 동그라미로 코너 표시
                           thickness=2)

cv2.imshow('image', image)                                                  # 흑백이미지 출력
cv2.imshow('harris_norm', harris_norm)                                      # harris계산값 출력
cv2.imshow('output', image2)                                                # 코너 검출 값 출력
cv2.waitKey()
cv2.destroyAllWindows()
```

$$R = det(M) - k(trace(M))^2$$

Harris corner (corner.py)

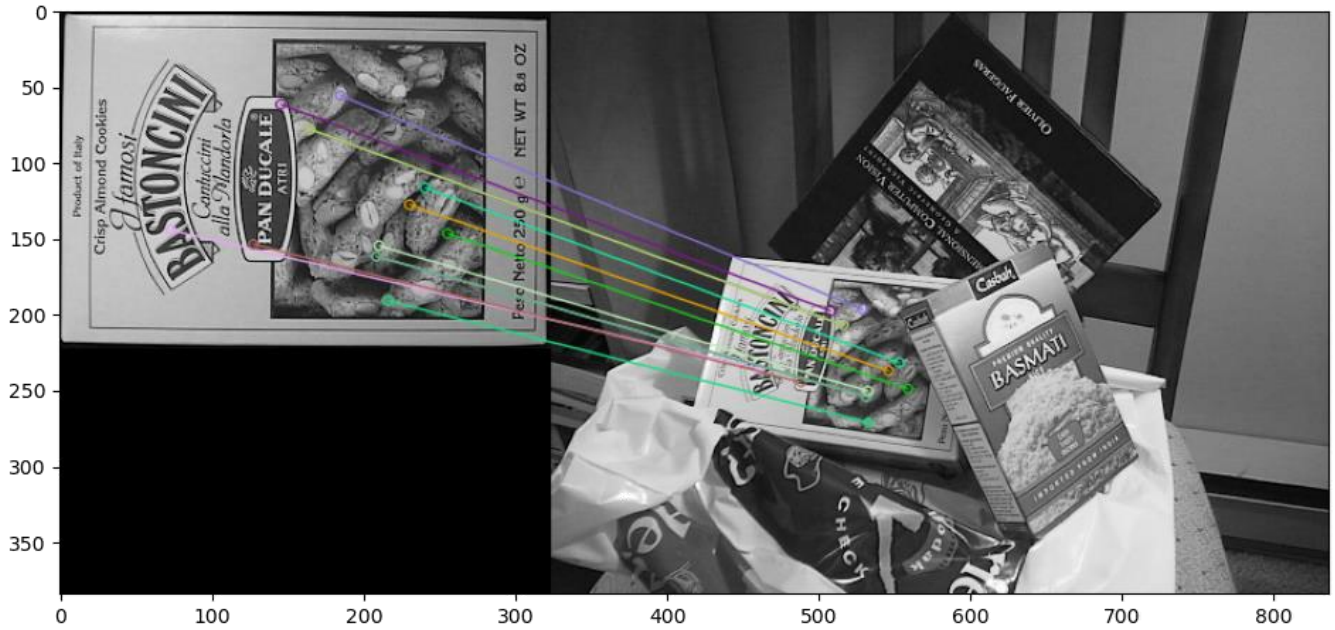
Original image



Harris norm

Corner image

SIFT (sift.py)




```python
import numpy as np
import cv2

img1 = cv2.imread('box.png')                                    # 첫번째 이미지 불러오기
img2 = cv2.imread('box_in_scene.png')                           # 두번째 이미지 불러오기

sift = cv2.xfeatures2d.SIFT_create()                            # sift객체 만들기
kp1, des1 = sift.detectAndCompute(img1,None)                    # 첫번째 이미지에서 keypoint와 특징 추출
kp2, des2 = sift.detectAndCompute(img2,None)                    # 두번째 이미지에서 keypoint와 특징 추출

bf = cv2.BFMatcher()                                            # 점을 둘둘씩 비교하는 객체
matches = bf.knnMatch(des1,des2, k=2)                           # knn 방식으로 비슷한 점 match

good = []                                                       # 가장 비슷한 점 골라내기
for m,n in matches:
    if m.distance < 0.3*n.distance:                             # 다른점들보다 많이 비슷하면 (비슷하다=distance가 작다)
        good.append([m])

img3 = cv2.drawMatchesKnn(img1,kp1,img2,kp2,good,None,flags=2)  # 두개의 특징 점을 서로 연결
cv2.imshow('image', img3)                                       # 흑백이미지 출력
cv2.waitKey()
cv2.destroyAllWindows()
```

Kmeans

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

image = cv2.imread("neutrophils.png")
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# reshape the image to a 2D array of pixels and 3 color values (RGB)
pixel_values = image.reshape((-1, 3))
pixel_values = np.float32(pixel_values)
print(pixel_values.shape)

criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.2)
# number of clusters (K)
k = 3
_, labels, (centers) = cv2.kmeans(pixel_values, k, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)
centers = np.uint8(centers)
labels = labels.flatten()

# convert all pixels to the color of the centroids
segmented_image = centers[labels.flatten()]

# reshape back to the original image dimension
segmented_image = segmented_image.reshape(image.shape)
# show the image
plt.figure('Kmeans Image')
plt.subplot(211)
plt.title('original image')
plt.imshow(image)
plt.subplot(212)
plt.title('segmented_image')
plt.imshow(segmented_image)
plt.show()
```