[RT704] Advanced Medical Image Processing

< Assignment 1 >

(2021-10-30)

Robotics Engineering

202123008 Jinmin Kim

Phone: 010-6266-6099

Mail: rlawlsals@dgist.ac.kr

Problem #1

- 파일 => “pro1.py” 참조

1. 함수 정의

```
# ##############################################################################
# Function Definition
def Box(img, size):
    kernel = np.ones((size,size), np.float32) / (size*size)
    dst = cv2.filter2D(img, -1, kernel)
    return dst

def Gauss(img, size, sigma):
    kernel1d = cv2.getGaussianKernel(size, sigma)
    kernel2d = np.outer(kernel1d, kernel1d.transpose())
    low_im_array = cv2.filter2D(img, -1, kernel2d)
    low_array = Image.fromarray(low_im_array)
    return low_array

def Sharp(img, Sharpen_value):
    kernel = np.array([[-1,-1,-1],[-1,Sharpen_value,-1],[-1,-1,-1]])
    Sharpen = cv2.filter2D(img, -1, kernel)
    return Sharpen

def Median(img, filter_size, stride):
    img_shape = np.shape(img)
    result_shape = tuple(np.int64(
        (np.array(img_shape)-np.array(filter_size))/stride+1
    ))
    result = np.zeros(result_shape)
    for h in range(0, result_shape[0], stride):
        for w in range(0, result_shape[1], stride):
            tmp = img[h:h+filter_size[0],w:w+filter_size[1]]
            tmp = np.sort(tmp.ravel())
            result[h,w] = tmp[int(filter_size[0]*filter_size[1]/2)]
    return result

def psnr(ori_img, con_img):
    max_pixel = np.max(con_img)
    mse = np.mean((ori_img - con_img)**2)
    if mse ==0:
        return 100
    psnr = 20* math.log10(max_pixel / math.sqrt(mse))
    return psnr
```

> 필터 함수 (a) box, (b) Gaussian, (c) sharpening, (d) median filtering 와 PSNR scoring 함수

2. 결과

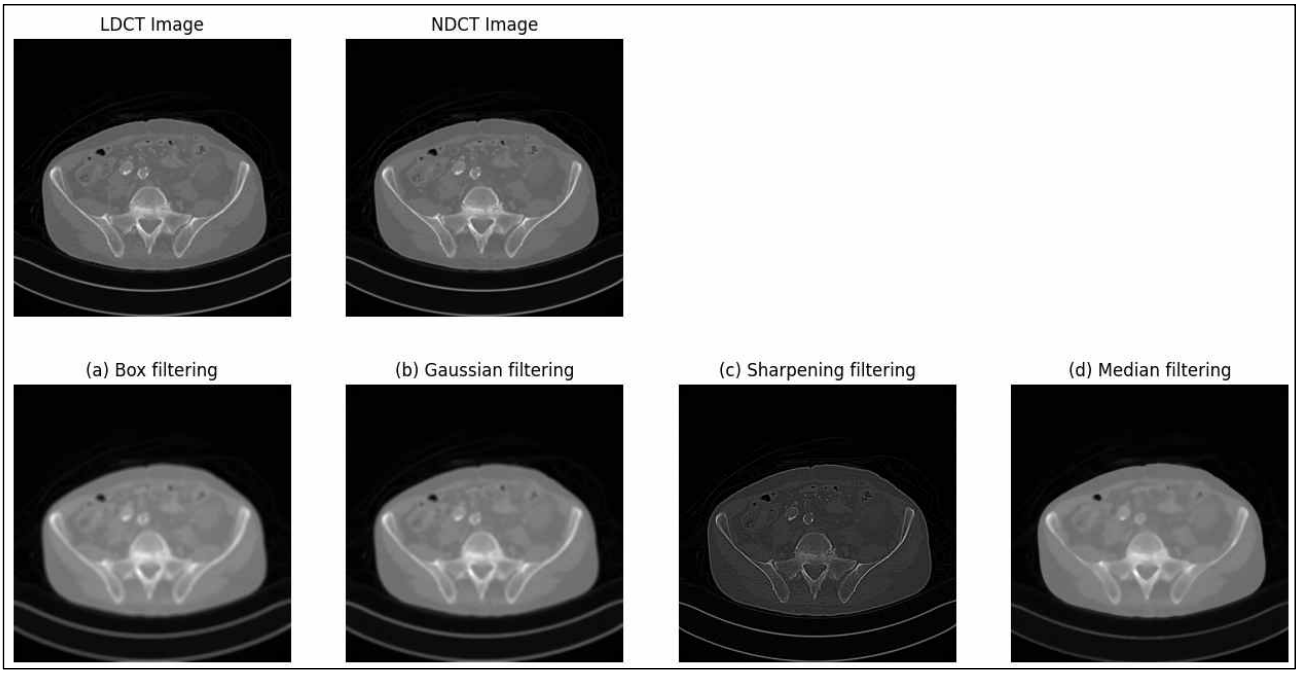

그림 1.1. 1행: 원본 LDCT, NDCT 이미지, 2행: 필터링된 LDCT 이미지

(a) Box filter : Blurred 효과를 얻을 수 있다.
(b) Gaussian filter : Blurred 효과를 얻을 수 있다.
(c) Sharpening filter : Edge 강조 효과를 얻을 수 있다.
(d) Median filter : Salt&Pepper 노이즈 제거 효과를 얻을 수 있다.

3. PSNR 수치 비교

```
# ###########################################################################
# Perform denoising of "img_LDCT" via (a) box, (b) Gaussian, (c) sharpening, and (d) median filtering
FilterSize = 9
sigma = 3
Sharpen_value = 10
stride = 1

Box_img = Box(img_LDCT,FilterSize)
Gauss_img = Gauss(img_LDCT,FilterSize,sigma)
Sharp_img = Sharp(img_LDCT,Sharpen_value)
Median_img = Median(img_LDCT, (FilterSize, FilterSize), stride)

psnr_l = psnr(img_NDCT,img_LDCT)
psnr_a = psnr(img_NDCT,Box_img)
psnr_b = psnr(img_NDCT,Gauss_img)
psnr_c = psnr(img_NDCT,Sharp_img)
psnr_d = psnr(np.resize(img_NDCT, (512-FilterSize+stride,512-FilterSize+stride)),Median_img)

print(psnr_l, psnr_a, psnr_b, psnr_c, psnr_d)
```

Result

```
43.36402009895004 30.421049603895693 31.27313012086025 34.78288565810073 12.86798230125008
```

> PSNR 수치가 높을수록 대상 이미지와의 유사도가 높다.

FilterSize = 9, sigma = 3, Sharpen_value = 10, stride = 1에서,
psnr_l : 43.3640
psnr_a : 30.4210
psnr_b : 31.2731
psnr_c : 34.7828
psnr_d : 12.8679 이다.

아래 조건들을 바꾸어준다면 psnr값은 변동될 수 있다. 대상 이미지와의 유사성이 낮아지면 psnr값도 낮아짐을 추측할 수 있다.

Ex) FilterSize = 15, sigma = 5, Sharpen_value = 20, stride = 1일 때,

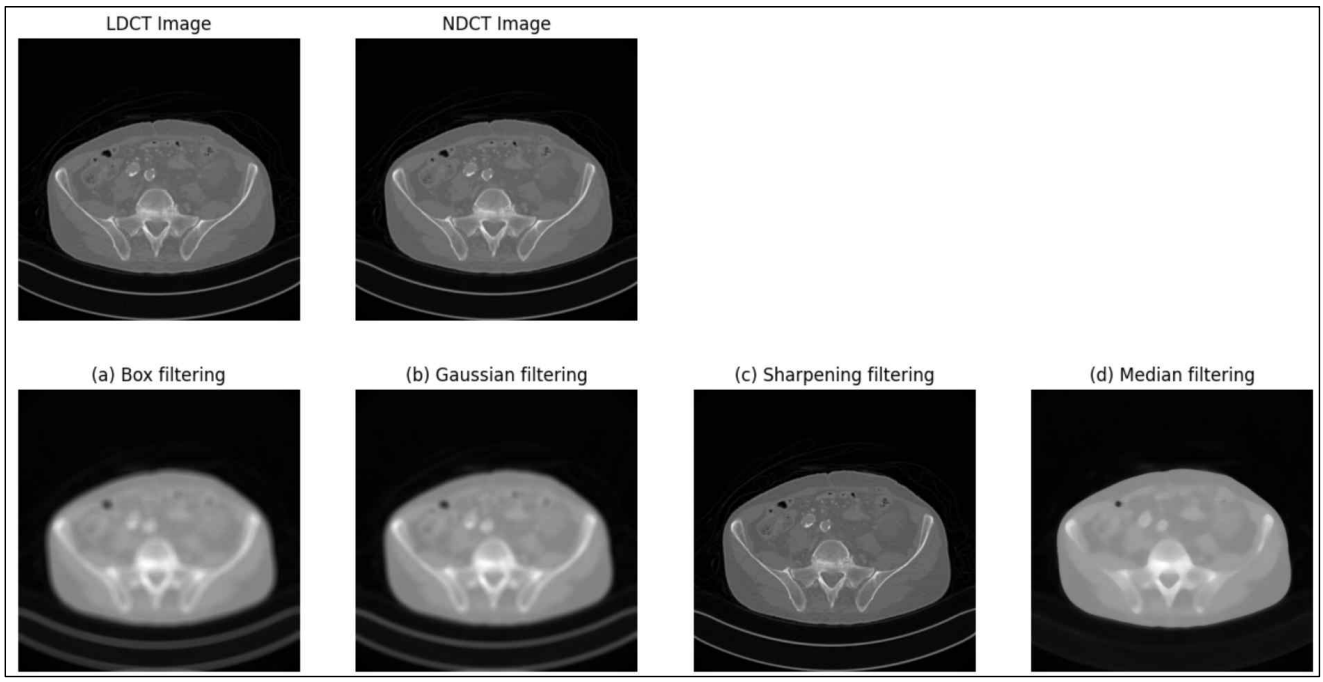

그림 1.2. 다른 파라미터로 변경한 결과

```
43.36402009895004 27.692506072656528 28.385887278751404 45.23541542895889 11.065451506942743
```

psnr_a : 30.4210 > 27.6925
psnr_b : 31.2731 > 28.3858
psnr_c : 34.7828 < 45.2354
psnr_d : 12.8679 > 11.0654
> (a),(b),(d)의 경우 Filtersize가 커짐에 따라 Blurring이 심해져 psnr 수치가 감소하였다. 그러나 (c)의 경우 Sharpen mask의 중앙값을 올려줌으로서 노이즈가 제거되어 psnr값이 소폭 증가했음을 알 수 있다.

Problem #2

- 파일 => “pro2.py” 참조
- 구현된 코드 분석

1) 라이브러리 임포트

```
1   print(__doc__)
2
3   from time import time
4
5   import matplotlib.pyplot as plt
6   import numpy as np
7   import scipy as sp
8
9   from sklearn.decomposition import MiniBatchDictionaryLearning
10  from sklearn.feature_extraction.image import extract_patches_2d
11  from sklearn.feature_extraction.image import reconstruct_from_patches_2d
12
13
```

1~13 : 필요 라이브러리를 불러온다.

2) 학습 데이터와 테스트 데이터 준비

```
14  try:  # SciPy >= 0.16 have face in misc
15      from scipy.misc import face
16      face = face(gray=True)
17  except ImportError:
18      face = sp.face(gray=True)
19
20  # Convert from uint8 representation with values between 0 and 255 to
21  # a floating point representation with values between 0 and 1.
22  face = face / 255
23
24  # downsample for higher speed
25  face = face[::4, ::4] + face[1::4, ::4] + face[::4, 1::4] + face[1::4, 1::4]
26  face /= 4.0
27  height, width = face.shape
28
29  # Distort the right half of the image
30  print('Distorting image...')
31  distorted = face.copy()
32  distorted[:, width // 2:] += 0.075 * np.random.randn(height, width // 2)
33
```

14~33 : 이미지 데이터를 scipy.misc로부터 가져온다. 가져온 이미지 데이터에 중앙 세로선을 기준으로 좌측 절반은 원본 이미지, 우측 절반은 랜덤하게 왜곡된 이미지로 만든다.

3) 패치 추출 및 딕셔너리 제작

```
34  # Extract all reference patches from the left half of the image
35  print('Extracting reference patches...')
36  t0 = time()
37  patch_size = (7, 7)
38  data = extract_patches_2d(distorted[:, :width // 2], patch_size)
39  data = data.reshape(data.shape[0], -1)
40  data -= np.mean(data, axis=0)
41  data /= np.std(data, axis=0)
42  print('done in %.2fs.' % (time() - t0))
43
44  # ##############################################################################
45  # Learn the dictionary from reference patches
46
47  print('Learning the dictionary...')
48  t0 = time()
49  dico = MiniBatchDictionaryLearning(n_components=100, alpha=1, n_iter=500)
50  V = dico.fit(data).components_
51  dt = time() - t0
52  print('done in %.2fs.' % dt)
53
54  plt.figure(figsize=(4.2, 4))
55  for i, comp in enumerate(V[:100]):
56      plt.subplot(10, 10, i + 1)
57      plt.imshow(comp.reshape(patch_size), cmap=plt.cm.gray_r,
58                 interpolation='nearest')
59      plt.xticks(())
60      plt.yticks(())
61  plt.suptitle('Dictionary learned from face patches\n' +
62               'Train time %.1fs on %d patches' % (dt, len(data)),
63               fontsize=16)
64  plt.subplots_adjust(0.08, 0.02, 0.92, 0.85, 0.08, 0.23)
65
66
```

34~66 : 좌측 이미지(원본)로부터 7*7 크기의 기준 패치를 추출한다. 그리고 그 기준 패치로부터 딕셔너리를 만든다. 그림 2.1의 각각의 패치는 하나의 atom으로서 딕셔너리의 열벡터로 들어간다.

그림 2.1. Dictionary learned from face patches

4) 학습 전 이미지 디스플레이

```
67  # ##############################################################################
68  # Display the distorted image
69
70  def show_with_diff(image, reference, title):
71      """Helper function to display denoising"""
72      plt.figure(figsize=(5, 3.3))
73      plt.subplot(1, 2, 1)
74      plt.title('Image')
75      plt.imshow(image, vmin=0, vmax=1, cmap=plt.cm.gray,
76                 interpolation='nearest')
77      plt.xticks(())
78      plt.yticks(())
79      plt.subplot(1, 2, 2)
80      difference = image - reference
81
82      plt.title('Difference (norm: %.2f)' % np.sqrt(np.sum(difference ** 2)))
83      plt.imshow(difference, vmin=-0.5, vmax=0.5, cmap=plt.cm.PuOr,
84                 interpolation='nearest')
85      plt.xticks(())
86      plt.yticks(())
87      plt.suptitle(title, size=16)
88      plt.subplots_adjust(0.02, 0.02, 0.98, 0.79, 0.02, 0.2)
89
90
91  show_with_diff(distorted, face, 'Distorted image')
92
```

67~92 : 학습하기 전 이미지를 디스플레이 해본다. show_with_diff 함수는 기준 이미지와 테스트 이미지를 받아서 그림 2.2와 같은 이미지를 디스플레이 해준다. 그림 2.2의 좌측 이미지는 원본과 왜곡된 이미지를 각각 보여주고 있다. 우측 이미지는 원본 이미지에서 좌측 이미지를 제거해서 구한 것으로, 왜곡의 정도를 시각화하여 보여준다.

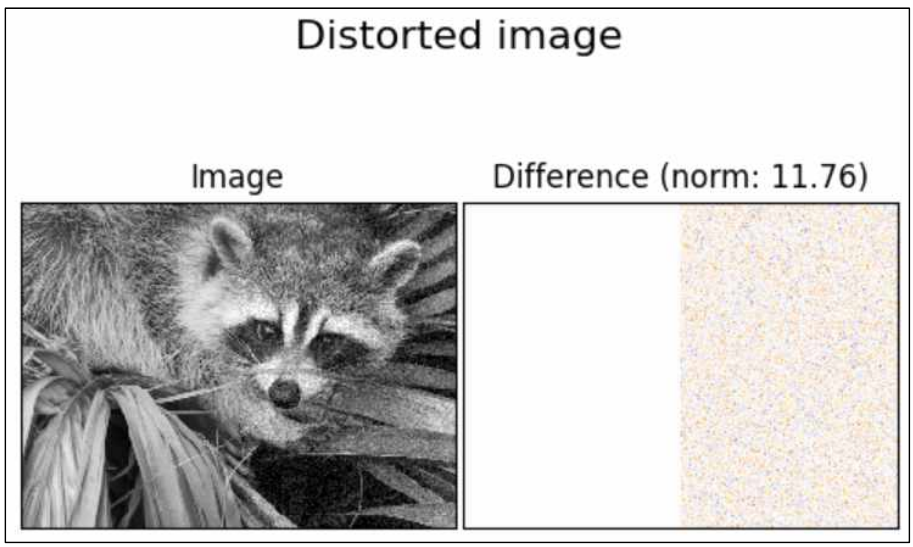

그림 2.2. Distorted image

5) 딕셔너리 학습 및 이미지 재구성 결과

```
93  # ############################################################################
94  # Extract noisy patches and reconstruct them using the dictionary
95
96  print('Extracting noisy patches... ')
97  t0 = time()
98  data = extract_patches_2d(distorted[:, width // 2:], patch_size)
99  data = data.reshape(data.shape[0], -1)
100 intercept = np.mean(data, axis=0)
101 data -= intercept
102 print('done in %.2fs.' % (time() - t0))
103
104 transform_algorithms = [
105     ('Orthogonal Matching Pursuit\n1 atom', 'omp',
106      {'transform_n_nonzero_coefs': 1}),
107     ('Orthogonal Matching Pursuit\n2 atoms', 'omp',
108      {'transform_n_nonzero_coefs': 2}),
109     ('Least-angle regression\n5 atoms', 'lars',
110      {'transform_n_nonzero_coefs': 5}),
111     ('Thresholding\n alpha=0.1', 'threshold', {'transform_alpha': .1})]
112
113 reconstructions = {}
114 for title, transform_algorithm, kwargs in transform_algorithms:
115     print(title + '...')
116     reconstructions[title] = face.copy()
117     t0 = time()
118     dico.set_params(transform_algorithm=transform_algorithm, **kwargs)
119     code = dico.transform(data)
120     patches = np.dot(code, V)
121
122     patches += intercept
123     patches = patches.reshape(len(data), *patch_size)
124     if transform_algorithm == 'threshold':
125         patches -= patches.min()
126         patches /= patches.max()
127     reconstructions[title][:, width // 2:] = reconstruct_from_patches_2d(
128         patches, (height, width // 2))
129     dt = time() - t0
130     print('done in %.2fs.' % dt)
131     show_with_diff(reconstructions[title], face,
132                    title + ' (time: %.1fs)' % dt)
133
134 plt.show()
```

93~134 : 그림 2.3~2.6은 딕셔너리 학습 및 이미지 재구성 결과를 보여준다. 그림 2.3은 OMP에서 0이 아닌 atom을 1개만 사용한 방법이고, 그림 2.4는 atom을 2개 사용한 방법이다. 그림 2.5는 최소각도회귀(LARS) 방법에서 0이 아닌 atom을 5개 사용한 방법이다. 마지막으로 그림 2.6은 Thresholding 방법에서 알파 값을 0.1로 설정한 방법이다.

그림 2.3과 그림 2.4를 보면, 두 결과 모두 그림 2.2에서의 결과보다 더 원본 이미지와의 차이가 적어졌다. 그리고 그림 2.4처럼 OMP에서 atom 개수를 더 많이 사용하였을 때, 원본 이미지와의 차이가 더 적어짐을 알 수 있다.

하지만 그림 2.5와 그림 2.6을 보면 그림 2.2에서의 결과보다 더 원본 이미지와의 차이가 커졌다. 이는 컴퓨터 비전의 저주파 필터와 비슷한 역할을 한다고 볼 수 있다.

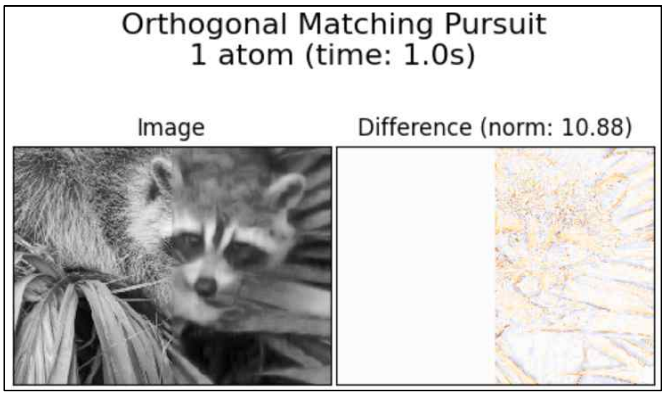

그림 2.3. OMP (1 atom)

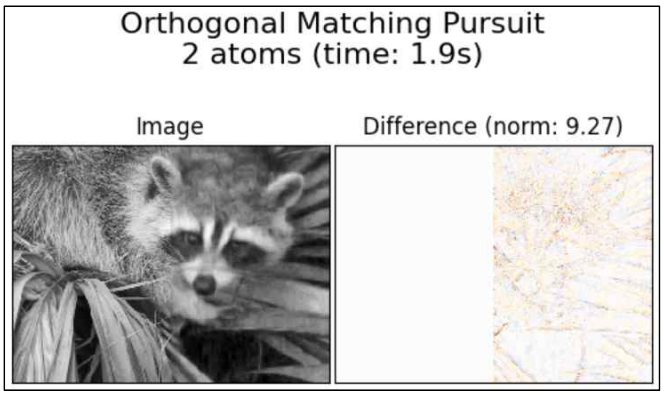

그림 2.4. OMP (2 atom)

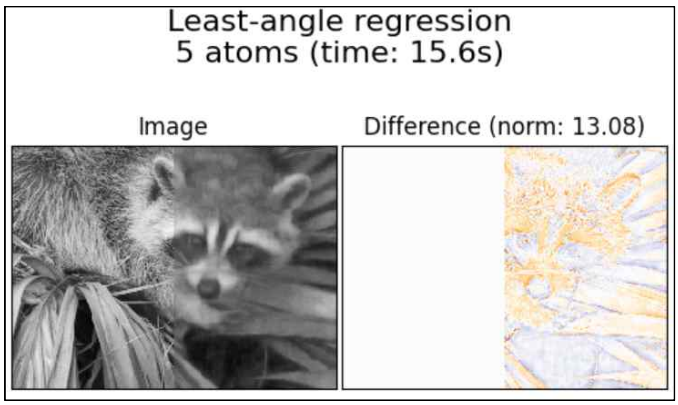

그림 2.5. Least-angle regression (5 atom)

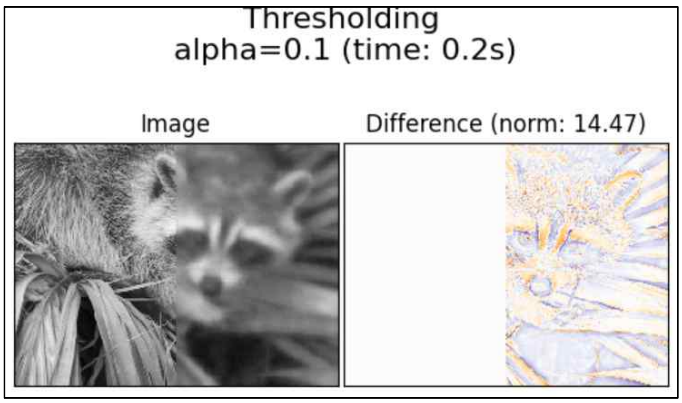

그림 2.6. Thresholding (alpha=0.1)

Problem #3

- 파일 => “pro3.py” 참조

전반적인 코드 구성은 Problem #2의 구성을 따랐다. 위 문제에서는 한 이미지의 너비를 절반씩 나누어 학습하였지만 나는 트레이닝 이미지와 테스트 이미지를 따로 사용하였다.

1) 라이브러리 임포트

=> “위 문제와 동일”

2) 학습 데이터와 테스트 데이터 준비

```
15  # ##############################################################################
16  # 이미지 로드
17  path_LDCT105 = "L506_QD_3_1.CT.0003.0105.2015.12.22.20.45.42.541197.358793241.IMA"
18  path_NDCT105 = "L506_FD_3_1.CT.0001.0105.2015.12.22.20.19.39.34094.358586575.IMA"
19
20
21  # 테스트와 트레이닝할 이미지 정의
22  print('Upload images...')
23  test_img = pydicom.dcmread(path_LDCT105).pixel_array
24  train_img = pydicom.dcmread(path_NDCT105).pixel_array
25
26
27  # 0과 1사이의 값으로 표현하기 위한 변환
28  train = train_img / np.max(train_img)
29  test = test_img / np.max(test_img)
30  height, width = train.shape
31
32
```

15~32 : NDCT105 이미지를 트레이닝 이미지로, LDCT105이미지를 테스트 이미지로 사용하였다. 그리고 동일한 방식으로 변수를 정의하였는데, 다운 샘플링은 하지 않았다. 시간이 오래 걸리더라도 더 원본에 가까운 결과를 얻어서 Problem #1과 PSNR 값을 비교해보기 위해서이다.

3) 패치 추출 및 딕셔너리 제작

```
33  # 트레이닝 이미지로부터 모든 기준 패치들을 추출
34  print('Extracting reference patches...')
35  t0 = time()
36  patch_size = (7, 7)
37  data = extract_patches_2d(train[:, :], patch_size)
38  data = data.reshape(data.shape[0], -1)
39  data -= np.mean(data, axis=0)
40  data /= np.std(data, axis=0)
41  print('done in %.2fs.' % (time() - t0))
42
43  # ############################################################################
44  # 기준 패치로부터 딕셔너리 학습
45  print('Learning the dictionary...')
46  t0 = time()
47  dico = MiniBatchDictionaryLearning(n_components=100, alpha=1, n_iter=500)
48  V = dico.fit(data).components_
49  dt = time() - t0
50  print('done in %.2fs.' % dt)
51
52  plt.figure(figsize=(4.2, 4))
53  for i, comp in enumerate(V[:100]):
54      plt.subplot(10, 10, i + 1)
55      plt.imshow(comp.reshape(patch_size), cmap=plt.cm.gray_r,
56                 interpolation='nearest')
57      plt.xticks(())
58      plt.yticks(())
59  plt.suptitle('Dictionary learned from NDCT patches\n' +
60               'Train time %.1fs on %d patches' % (dt, len(data)),
61               fontsize=16)
62  plt.subplots_adjust(0.08, 0.02, 0.92, 0.85, 0.08, 0.23)
63
64
```

43~64 : 트레이닝 이미지(NDCT)로부터 7*7 크기의 패치들을 추출하였다. 그리고 그 기준 패치들로부터 딕셔너리를 만들었다.

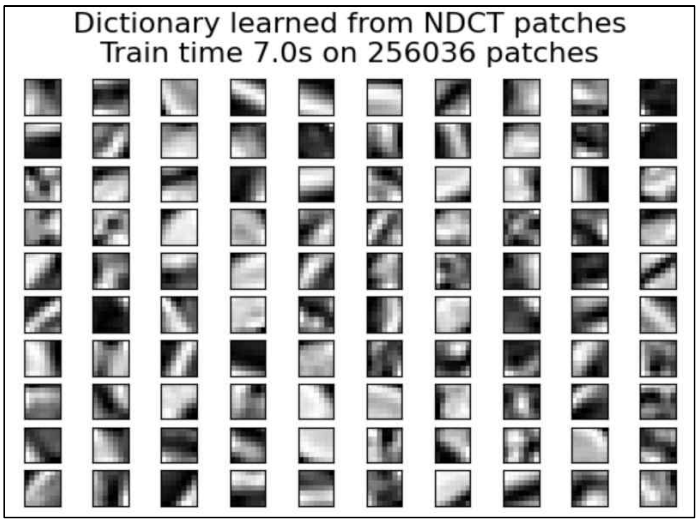

그림 3.1. Dictionary learned from NDCT patches

4) 학습 전 이미지 디스플레이

```
65  # ########################################################################
66  # 학습 전 이미지 디스플레이
67  def show_with_diff(image, reference, title):
68      """Helper function to display denoising"""
69      plt.figure(figsize=(5, 3.3))
70
71      plt.subplot(1, 3, 1)
72      plt.title('Training Image')
73      plt.imshow(reference, vmin=0, vmax=1, cmap=plt.cm.gray,
74                 interpolation='nearest')
75      plt.xticks(())
76      plt.yticks(())
77
78      plt.subplot(1, 3, 2)
79      plt.title('Test Image')
80      plt.imshow(image, vmin=0, vmax=1, cmap=plt.cm.gray,
81                 interpolation='nearest')
82      plt.xticks(())
83      plt.yticks(())
84
85      plt.subplot(1, 3, 3)
86      difference = image - reference
87      plt.title('Difference (norm: %.2f)' % np.sqrt(np.sum(difference ** 2)))
88      plt.imshow(difference, vmin=-0.5, vmax=0.5, cmap=plt.cm.PuOr,
89                 interpolation='nearest')
90      plt.xticks(())
91      plt.yticks(())
92
93      plt.suptitle(title, size=16)
94      plt.subplots_adjust(0.02, 0.02, 0.98, 0.79, 0.02, 0.2)
95
96
97  show_with_diff(test, train, 'Before Learning')
98
```

65~98 : 학습하기 전 이미지를 디스플레이 해본다. 그림 3.2의 좌측 이미지는 원본 이미지(NDCT)를 보여주고, 중앙 이미지는 왜곡된 이미지(LDCT)를 각각 보여주고 있다. 우측 이미지는 원본 이미지에서 왜곡된 이미지를 제거해서 구한 것으로, 왜곡의 정도를 시각화하여 보여준다.

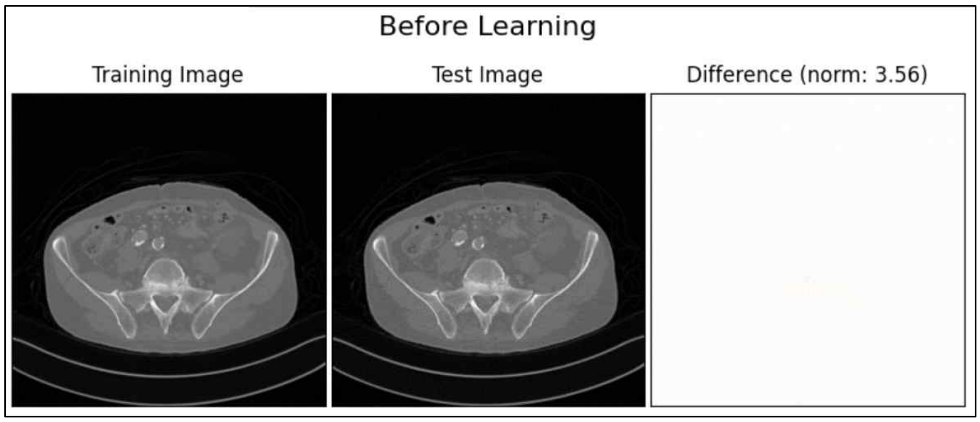

그림 3.2. 학습 전 이미지들과 그 차이

5) 딕셔너리 학습 및 이미지 재구성 결과

```
99  # ############################################################################
100 # 노이즈 패치들을 추출하고 딕셔너리를 사용하여 이미지 재구성
101 print('Extracting noisy patches... ')
102 t0 = time()
103 data = extract_patches_2d(train[:, :], patch_size)
104 data = data.reshape(data.shape[0], -1)
105 intercept = np.mean(data, axis=0)
106 data -= intercept
107 print('done in %.2fs.' % (time() - t0))
108
109 transform_algorithms = [
110     ('Orthogonal Matching Pursuit\n1 atom', 'omp',
111      {'transform_n_nonzero_coefs': 1}),
112     ('Orthogonal Matching Pursuit\n2 atoms', 'omp',
113      {'transform_n_nonzero_coefs': 2}),
114     ('Least-angle regression\n5 atoms', 'lars',
115      {'transform_n_nonzero_coefs': 5}),
116     ('Thresholding\n alpha=0.1', 'threshold', {'transform_alpha': .1})]
117
118 reconstructions = {}
119 for title, transform_algorithm, kwargs in transform_algorithms:
120     print(title + '...')
121     reconstructions[title] = train.copy()
122     t0 = time()
123     dico.set_params(transform_algorithm=transform_algorithm, **kwargs)
124     code = dico.transform(data)
125     patches = np.dot(code, V)
126
127     patches += intercept
128     patches = patches.reshape(len(data), *patch_size)
129     if transform_algorithm == 'threshold':
130         patches -= patches.min()
131         patches /= patches.max()
132     reconstructions[title][:, :] = reconstruct_from_patches_2d(
133         patches, (height, width))
134     dt = time() - t0
135     print('done in %.2fs.' % dt)
136     show_with_diff(reconstructions[title], train,
137                    title + ' (time: %.1fs)' % dt)
138
139 plt.show()
```

99~139 : 그림 3.3~3.6은 Problem #2와 동일한 알고리즘을 사용하여 재구성한 이미지를 보여준다. 각각의 그림의 좌측 이미지는 트레이닝 이미지(NDCT)이고 중앙 이미지는 테스트 이미지(LDCT)이다. 그리고 우측의 이미지는 트레이닝 이미지에서 테스트 이미지를 뺀 이미지이다.

결과는 Problem #2와 유사한 경향을 보임을 알 수 있다.

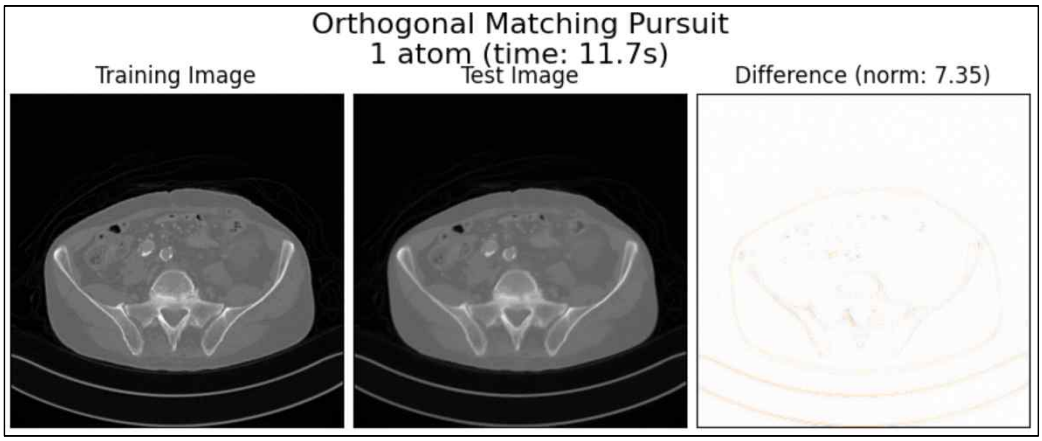

그림 3.3. OMP (1 atom)

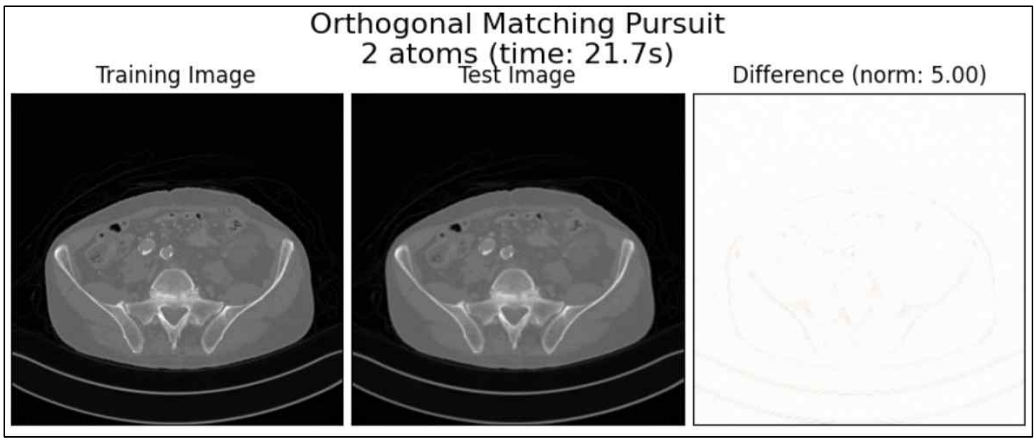

그림 3.4. OMP (2 atom)

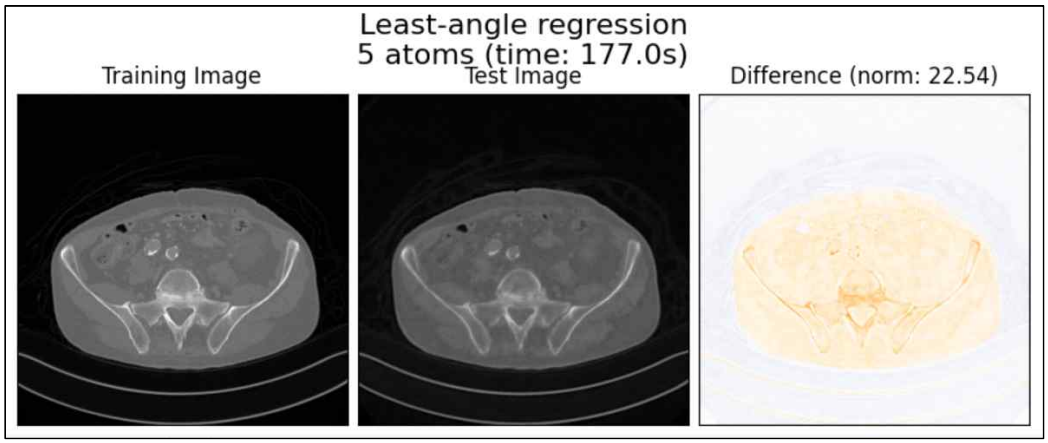

그림 3.5. Least-angle regression (5 atom)

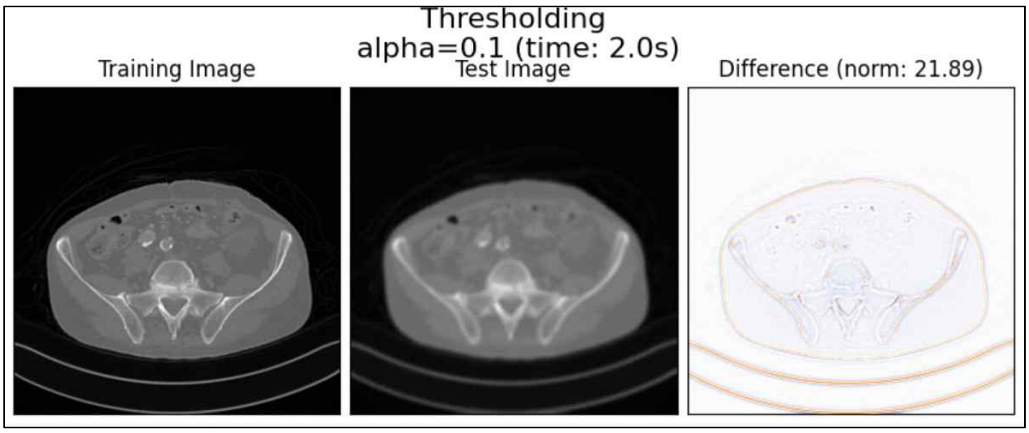

그림 3.6. Thresholding (alpha=0.1)

<Discussion>

1. PSNR 값 비교

```
43.15977921963342 35.809926547933316 39.926802089680116 24.509196662182774 26.474315201348887
```

< 딕셔너리 학습(그림 3.2~3.6)의 PSNR 값 >

```
43.36402009895004 30.421049603895693 31.27313012086025 34.78288565810073 12.86798230125008
```

< Problem #1의 PSNR 값 >

딕셔너리 학습과 전통 필터링을 사용했을 때의 결과가 비슷하다고 볼 수도 있다. 하지만 아래 사진과 같이 파라미터를 수정하여 더 향상된 결과를 얻을 수 있다.

```
43.15977921963342 36.1700363767065 45.18211749109781 24.486906796579454 26.445640319363374
```

< 딕셔너리 학습(그림 3.2~3.6)의 PSNR 값 **(파라미터 수정)** >

```
43.36402009895004 27.692506072656528 28.385887278751404 45.23541542895889 11.065451506942743
```

< Problem #1의 PSNR 값 **(파라미터 수정)** >

딕셔너리 학습 3번째 데이터는 OMP의 atom값을 2에서 5로 수정한 것이고, **Problem #1의 4번째 데이터**는 Sharpening 필터의 필터 크기를 9에서 15로 수정한 결과이다.

전통 필터링의 경우 정확도 향상에 한계가 있지만, 딕셔너리 학습의 경우는 여러 데이터를 학습하거나 참조할 atom 값을 더 많이 사용하면서 정확도를 향상시킬 수 있을 것이다.