[RT704] Advanced Medical Image Processing

< Assignment 4 >

(2021-12-17)

Robotics Engineering

202123008 Jinmin Kim

Phone: 010-6266-6099

Mail: rlawlsals@dgist.ac.kr

Problem #1 ~ #4

- 파일 => “pro1234.py” 참조
- 소스 이미지(.png) 파일들은 코드 파일(.py)과 동일한 폴더에 위치됨.

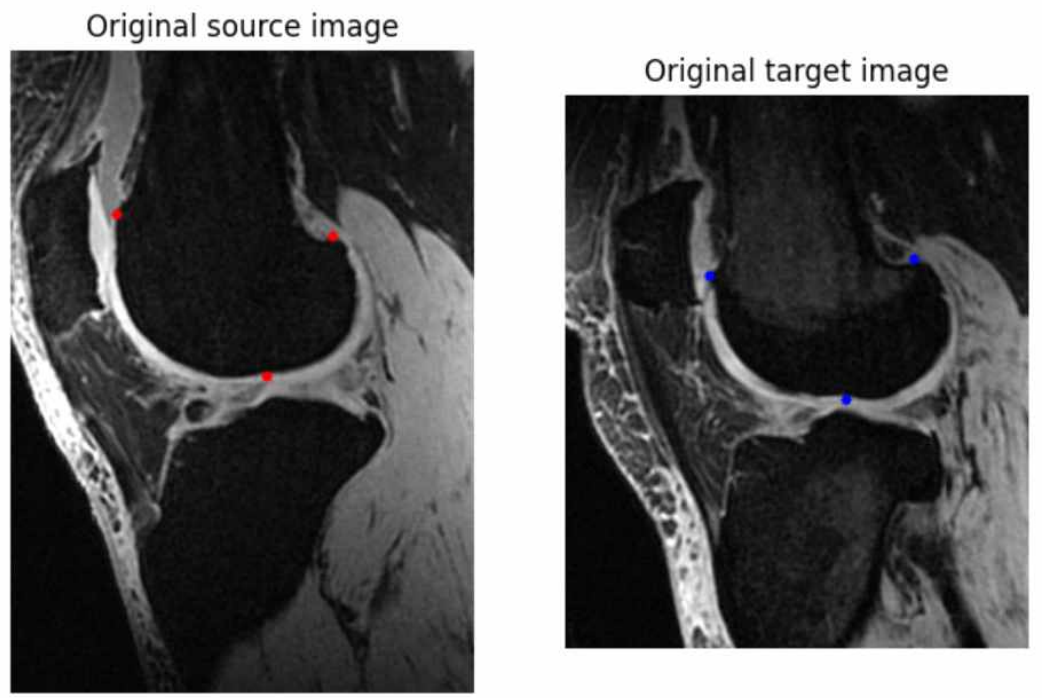
1) 코드 설명

```
1   import numpy as np
2   import cv2
3   from matplotlib import pyplot as plt
4
5   # ##############################################################################################
6   # 데이터 준비
7   source_image = cv2.imread("source.png")
8   source_label = cv2.imread("source_label.png")
9   target_image = cv2.imread("target.png")
10  target_label = cv2.imread("target_label.png")
11
12
```

1~12 : 필요 라이브러리를 불러오고 Opencv를 이용하여 이미지를 로드한다.

```
13  # ##############################################################################################
14  # Q1. Annotate more than 3 corresponding points from the source and target images manually.
15
16  # 포인트들 정의
17  pts1 = np.float32([[64, 99], [155, 197], [195, 112]])
18  pts2 = np.float32([[87, 109], [170, 184], [211, 99]])
19
20  # Source 이미지에서 선택된 포인트들 빨간색으로 표시
21  cv2.circle(source_image, (64, 99), 3, (255,0,0), -1)
22  cv2.circle(source_image, (155, 197), 3, (255,0,0), -1)
23  cv2.circle(source_image, (195, 112), 3, (255,0,0), -1)
24
25  # Target 이미지에서 선택된 포인트들 파란색으로 표시
26  cv2.circle(target_image, (87, 109), 3, (0,0,255), -1)
27  cv2.circle(target_image, (170, 184), 3, (0,0,255), -1)
28  cv2.circle(target_image, (211, 99), 3, (0,0,255), -1)
29
30  # Original Source label, Target label 디스플레이
31  plt.figure(figsize=(16, 8))
32
33  plt.subplot(141)
34  plt.title('Original source image')
35  plt.imshow(source_image, 'gray')
36  plt.axis("off")
37
38  plt.subplot(142)
39  plt.title('Original target image')
40  plt.imshow(target_image, 'gray')
41  plt.axis("off")
42
```

13~42 : <u>(문제 1)</u> 가상의 포인트들 3개를 정의한다. (manually)



위 그림은 정의한 세 점을 표시한 것인데, 이미지를 디스플레이 했을 때, 특징이 될만한 포인트들의 위치로 정의하였다.

```
43  # ##########################################################################
44  # Q2. Find an affine transformation matrix.
45
46  # Affine 변환행렬 출력
47  mat = cv2.getAffineTransform(pts1, pts2)
48  print(mat)
49  # Affine 역변환행렬 출력
50  inv_mat = cv2.getAffineTransform(pts2, pts1)
51  print(inv_mat)
52
```

43~52 : **(문제 2)** Opencv의 getAffineTransform함수를 이용하여 소스 이미지의 점(빨간색)에서 타겟 이미지의 점(파란색)으로 변환하는 Affine 변환행렬을 출력한다. 그리고 반대로 변환하는 행렬도 출력하였다.

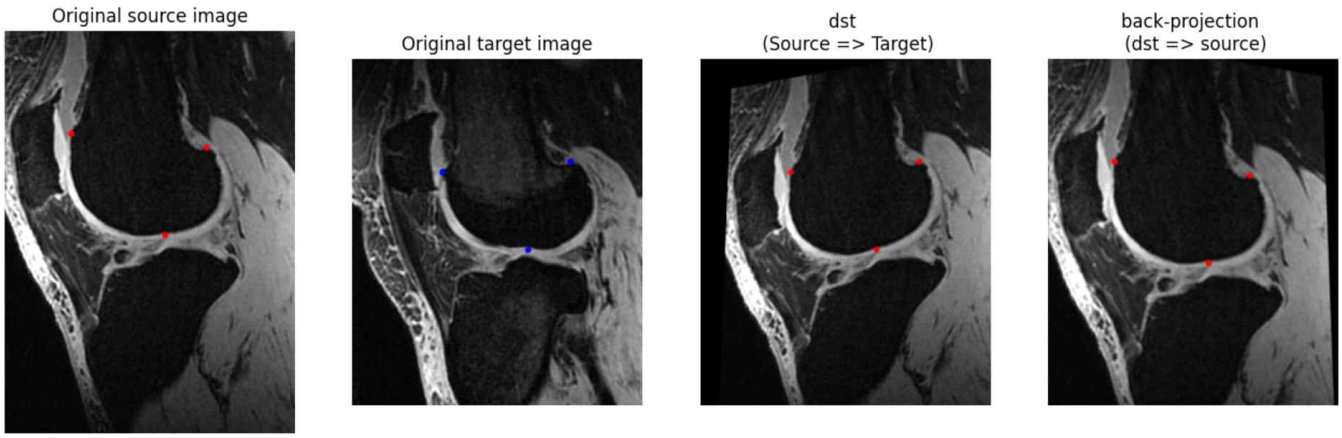
```
[[ 0.95006435 -0.03526384 29.68700129]
 [-0.16773917  0.92106392 28.54997855]]
[[  1.05972359   0.04057256 -32.6183613 ]
 [  0.19299112   1.09308983 -36.93701876]]
```

위 2x3 행렬 : (소스 => 타겟) Affine 변환행렬

아래 2x3 행렬 : (타겟 => 소스) Affine 변환행렬

```
53  # ##############################################################################
54  # 3. Transform the source image to the target image. Use the back-projection with bilinear interpolation.
55
56  # dst (source => target) 정의
57  h,w,_ = target_image.shape
58  dst = cv2.warpAffine(source_image, mat, (w, h))
59
60  # dst 디스플레이
61  plt.subplot(143)
62  plt.title('dst \n (Source => Target)')
63  plt.imshow(dst, 'gray')
64  plt.axis("off")
65
66  # back-projection (dst => source) 정의. bilinear interpolation 옵션을 넣음
67  h1,w1,_ = source_image.shape
68  inv_dst = cv2.warpAffine(dst, inv_mat,  (w, h), flags=cv2.INTER_LINEAR)
69
70  plt.subplot(144)
71  plt.title('back-projection \n (dst => source)')
72  plt.imshow(inv_dst, 'gray')
73  plt.axis("off")
74
75  plt.show()
76
```

53~76 : <u>(문제 3)</u> 소스 이미지 => 타겟 이미지로 변환한다.
dst는 원본 소스 이미지에서 타겟 이미지로 변환된 이미지이고,
inv_dst는 backward-warping을 통해 원본 소스 이미지로 다시 변환된 이미지이다.
Opencv의 warpAffine함수를 이용하여 변환을 진행하였고, Interpolation을 Bilinear로 설정해주었다.



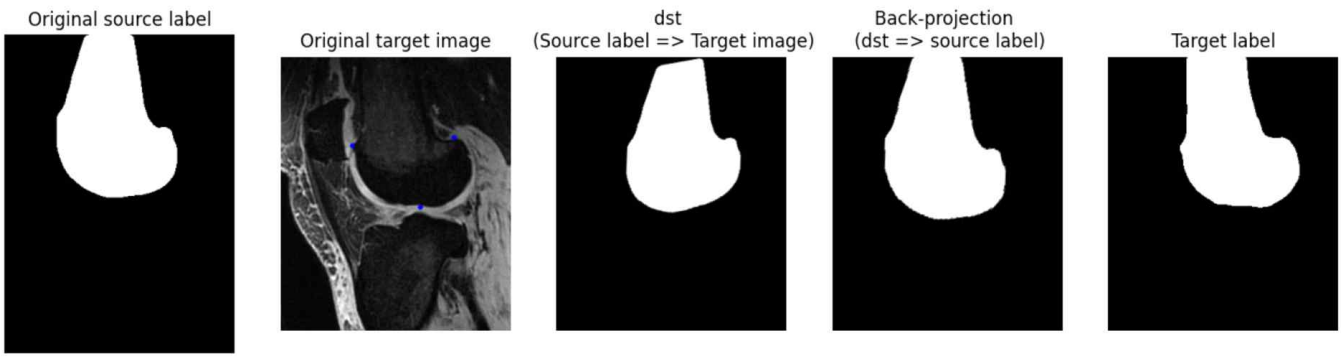
결과 : 좌측부터 원본 소스 이미지, 원본 타겟 이미지, 변환된 소스 이미지, 다시 변환된 소스 이미지이다.

dst를 보면 원본 소스 이미지가 Affine 변환 행렬에 의해 변형되며 타겟 이미지의 크기에 맞춰서 변환되었음을 알 수 있다. 이것을 다시 backward-warping을 통해 복원하면 타겟 이미지와 이미지 크기가 같아졌음을 알 수 있다.

하지만, 변환을 거치는 과정에서 빈 좌표 공간에 의한 이미지 손실이 일어나게 되는데, Bilinear interpolation을 통해 빈 공간을 보간해주었다.

```
77  # ##############################################################################
78  # 4. Transform the source label to the target image. Use the back-projection with nearest neighbor interpolation.
79
80  # Original Source label, Target image 디스플레이
81  plt.figure(figsize=(16, 8))
82
83  plt.subplot(151)
84  plt.title('Original source label')
85  plt.imshow(source_label, 'gray')
86  plt.axis("off")
87
88  plt.subplot(152)
89  plt.title('Original target image')
90  plt.imshow(target_image, 'gray')
91  plt.axis("off")
92
93  # dst2 (source label => target image) 정의
94  dst2 = cv2.warpAffine(source_label, mat, (w, h), )
95
96  # dst2 디스플레이
97  plt.subplot(153)
98  plt.title('dst \n (Source label => Target image)')
99  plt.imshow(dst2, 'gray')
100 plt.axis("off")
101
102 # back-projection 2 (dst2 => source label) 정의. nearest neighbor interpolation 옵션을 넣음.
103 inv_dst2 = cv2.warpAffine(dst2, inv_mat, (w, h), flags=cv2.INTER_NEAREST)
104
105 plt.subplot(154)
106 plt.title('Back-projection \n (dst => source label)')
107 plt.imshow(inv_dst2, 'gray')
108 plt.axis("off")
109
110 # Target label 디스플레이
111 plt.subplot(155)
112 plt.title('Target label')
113 plt.imshow(target_label, 'gray')
114 plt.axis("off")
115
116 plt.show()
117
```

77~117 : <u>(문제 4)</u> 소스 레이블을 타겟 이미지로 변환시킨다.
위에서 정의한 행렬을 그대로 사용하였고, 소스 이미지에서 소스 레이블로 변환시킬 대상만 바꾸었다.



결과 : 좌측부터 원본 소스 레이블, 원본 타겟 이미지, 변환된 소스 레이블, 다시 변환된 소스 레이블, 원본 타겟 레이블이다.

세 번째와 네 번째 그림을 보면, (문제 3)과 마찬가지로 변환이 진행되었음을 알 수 있고, 원본 타겟 레이블과 이미지 크기가 같아졌음을 알 수 있다. 변환을 거치는 과정에서 Nearest Neighborhood interpolation 옵션을 지정해주었다.

```
118  # Compute the DSC score between the transformed label and the target label.
119  def dice_coef2(y_true, y_pred):
120      y_true_f = y_true.flatten()
121      y_pred_f = y_pred.flatten()
122      union = np.sum(y_true_f) + np.sum(y_pred_f)
123      if union==0: return 1
124      intersection = np.sum(y_true_f * y_pred_f)
125      return 2. * intersection / union
126
127  true = np.array(target_label[:,:,0])
128  pred = np.array(inv_dst2[:,:,0])
129
130  print('Dice Similarity Score : ', dice_coef2(true, pred))
131
```

118~131 : DSC score를 비교한다.
DSC score 함수를 만들어서 변환된 레이블과 타겟 레이블의 Dice score를 비교한다.

```
Dice Similarity Score :  0.01039725895560683
```

DSC = 0.0103972... 의 결과가 나왔다.

DSC를 넣을 때 인풋 이미지가 제대로 설정되지 않은 것 같은데, 해결하지는 못하였다.

Problem #5 ~ #8

- 파일 => “pro5678.py” 참조
- 소스 이미지(.png) 파일들은 코드 파일(.py)과 동일한 폴더에 위치됨.

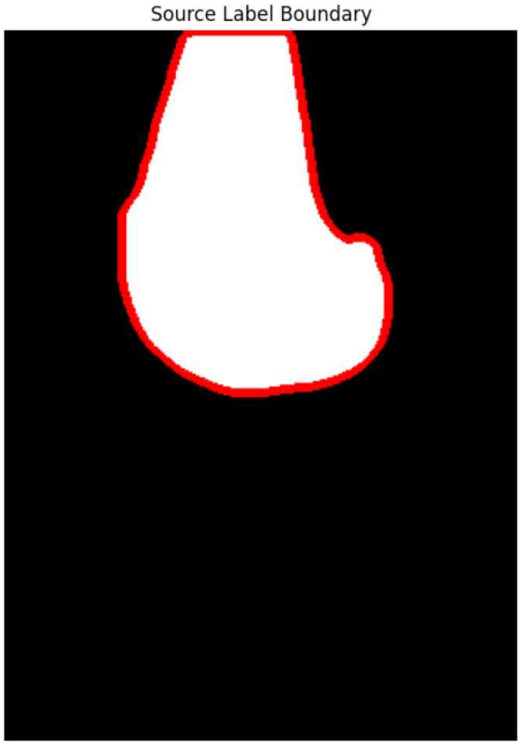
1) 코드 설명

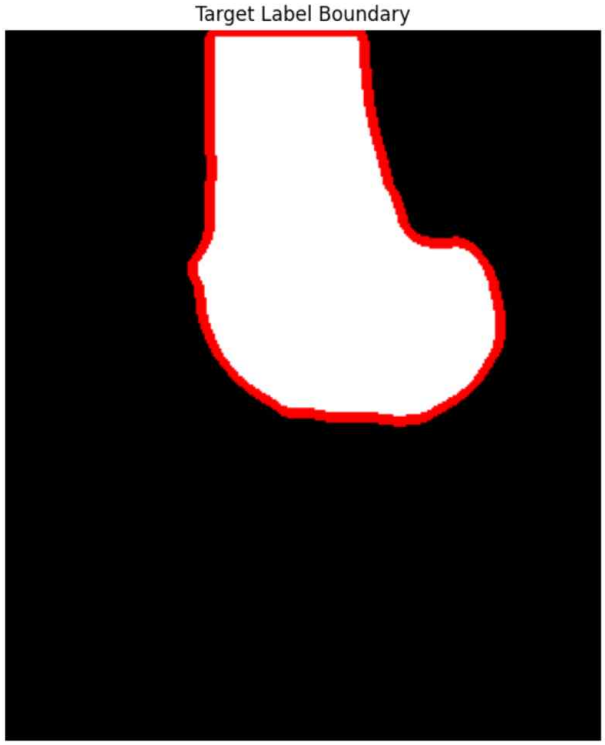
```
1   import numpy as np
2   import cv2
3   from matplotlib import pyplot as plt
4   from sklearn.neighbors import NearestNeighbors
5
6   # ##############################################################################################
7   # 데이터 준비
8   source_image = cv2.imread("source.png")
9   source_label = cv2.imread("source_label.png")
10  target_image = cv2.imread("target.png")
11  target_label = cv2.imread("target_label.png")
12
```

1~12 : 필요 라이브러리를 불러오고 Opencv를 이용하여 이미지를 로드한다.

```
13  # ##############################################################################################
14  # Q5. Extract object boundary points from the source and target labels.
15  source_label_gray = cv2.cvtColor(source_label, cv2.COLOR_RGB2GRAY)
16  target_label_gray = cv2.cvtColor(target_label, cv2.COLOR_RGB2GRAY)
17  contours1, hierarchy1 = cv2.findContours(source_label_gray, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
18  contours2, hierarchy2 = cv2.findContours(target_label_gray, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
19  boundary1 = cv2.drawContours(source_label, contours1, -1, (255,0,0), 3)
20  boundary2 = cv2.drawContours(target_label, contours2, -1, (255,0,0), 3)
21
22  plt.figure(figsize=(16, 8))
23  plt.subplot(121)
24  plt.title('Source Label Boundary')
25  plt.imshow(source_label, 'gray')
26  plt.axis("off")
27
28  plt.subplot(122)
29  plt.title('Target Label Boundary')
30  plt.imshow(target_label, 'gray')
31  plt.axis("off")
32
33  plt.show()
34
35  bd1 = np.array(contours1[0])
36  bd2 = np.array(contours2[0])
37
38
39
```

13~39 : **(문제 5)** 소스, 타겟 레이블의 바운더리를 추출한다.
Opencv의 cvtColor함수를 이용하여 3채널(RGB)이었던 이미지를 1채널로 복사한다.
-> Opencv의 findContours함수를 이용하여 바운더리를 추출한다. 바운더리는 빨간색, 두께는 3으로 설정하였다.





소스, 타겟 레이블의 바운더리를 추출함.
bd1 : 소스 레이블의 바운더리
bd2 : 타겟 레이블의 바운더리

```
40  # ##############################################################################################
41  # Q6. Implement Iterative Closest Point (ICP) method and find an affine transformation matrix.
42  indexes = [539, 538, 537, 536, 535, 534, 533]
43  bd1_modified = np.delete(bd1, indexes, 0)
44  bd1_modified = np.reshape(bd1_modified, (533,2))
45  bd2 = np.reshape(bd2, (533,2))
46
```

(문제 6) ICP를 구현하고 Affine 변환행렬을 찾아냄.
40~46 : 우선 bd1과 bd2 포인트들의 개수를 맞춰 줌. bd1의 포인트가 540개, bd2의 포인트가 533개로 추출되어서, bd1의 포인트 7개를 삭제해서 사용함.

47~163 : ICP 알고리즘 구현 함수
[출처 : https://daddynkidsmakers.blogspot.com/2021/09/icpiterative-closest-point.html]

ICP 알고리즘을 이용하여 변환행렬을 찾아냄.
함수의 인풋, 아웃풋에 해당하는 시스템 설명은 다음과 같음.

'''
주어짐 점군 A, B에 대해 정합 행렬을 계산해 리턴함.
Input:
    A: numpy 형태 Nxm 행렬. 소스(Src) mD points
    B: numpy 형태 Nxm 행렬. 대상(Dst) mD points
    init_pose: (m+1)x(m+1) 동차좌표계(homogeneous) 변환행렬
    max_iterations: 알고리즘 계산 중지 탈출 횟수
    tolerance: 수렴 허용치 기준
Output:
    T: 최종 동차좌표계 변환 행렬. maps A on to B
    distances: 가장 가까운 이웃점 간 유클리드 오차 거리
    i: 수렴 반복 횟수
'''

```
164    # ICP 실행
165    T, distances, iterations = icp(bd2, bd1_modified, tolerance=0.000001)
166
167    # Affine 변환 행렬, 반복 회수 출력
168    print('ICP algorithm affine transformation Matrix : \n', T)
169    print('iterations : ', iterations)
170
171
```

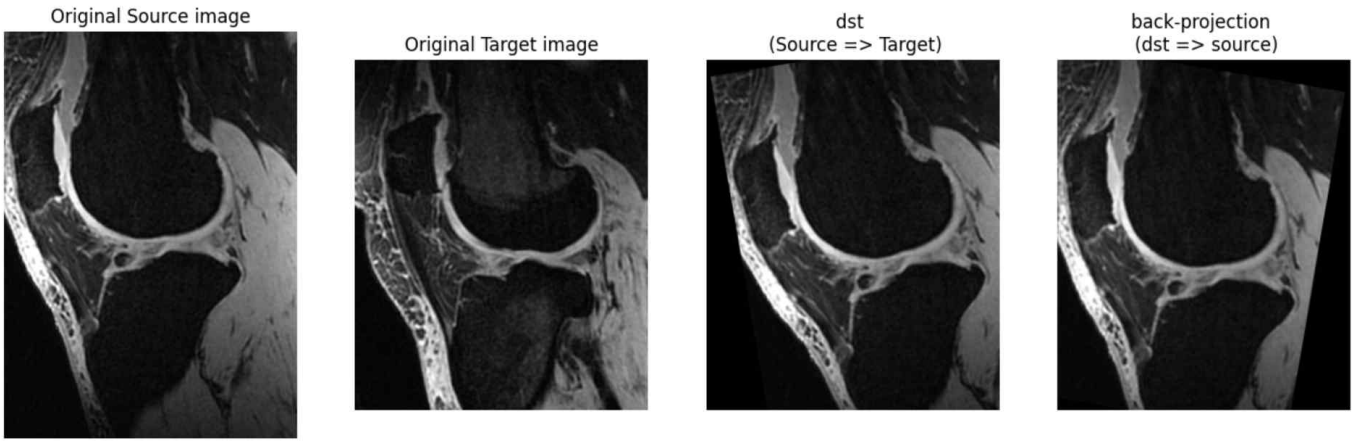
164~171 : ICP 알고리즘 함수를 실행하여 Affine 변환행렬, 반복 회수를 출력함.

```
ICP algorithm affine transformation Matrix :
 [[  0.98576631  -0.16812135  -1.07556548]
 [  0.16812135   0.98576631 -16.51537946]
 [  0.           0.           1.        ]]
iterations :  32
```

3x3의 Affine 변환행렬을 얻을 수 있고, 반복 회수는 32임을 알 수 있음.

```
172  # ##############################################################################
173  # Q7. Transform the source image to the target image. Use the back-projection with bilinear interpolation.
174
175  # Affine 역변환행렬 출력
176  mat = T.copy()
177  inv_mat = np.linalg.inv(mat)
178  indexes = [2]
179  inv_mat = np.delete(inv_mat, indexes, 0)
180  mat = np.delete(mat, indexes, 0)
181
182  # 디스플레이
183  plt.figure(figsize=(16, 8))
184
185  # Original source image & target image 디스플레이
186  plt.subplot(141)
187  plt.title('dst \n (Source => Target)')
188  plt.imshow(source_image, 'gray')
189  plt.axis("off")
190
191  plt.subplot(142)
192  plt.title('dst \n (Source => Target)')
193  plt.imshow(target_image, 'gray')
194  plt.axis("off")
195
196  # dst (source => target) 디스플레이
197  h,w,_ = target_image.shape
198  dst = cv2.warpAffine(source_image, inv_mat, (w, h))
199
200  plt.subplot(143)
201  plt.title('dst \n (Source => Target)')
202  plt.imshow(dst, 'gray')
203  plt.axis("off")
204
205  # back-projection (dst => source) 디스플레이, bilinear interpolation 옵션을 넣음
206  h1,w1,_ = source_image.shape
207  inv_dst = cv2.warpAffine(dst, mat,  (w, h), flags=cv2.INTER_LINEAR)
208
209  plt.subplot(144)
210  plt.title('back-projection \n (dst => source)')
211  plt.imshow(inv_dst, 'gray')
212  plt.axis("off")
213
214  plt.show()
215
216
```

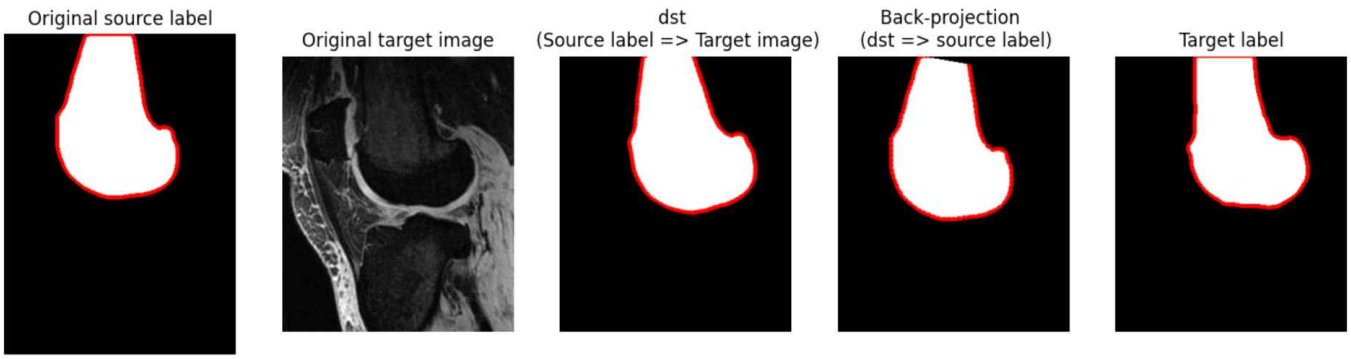
172~216 : **(문제 7)** (문제 3)에서와 동일한 조건, 다른 Affine 변환행렬(from ICP 알고리즘)을 이용하여 이미지 변환을 진행함.



결과 : 원본 소스 이미지에서 타겟 이미지로 변환하고, 다시 원본 이미지로 변환함.

```
217  # ##############################################################################
218  # Q8. Transform the source label to the target image. Use the back-projection with nearest neighbor interpolation.
219
220
221  # Original Source label, Target image 디스플레이
222  plt.figure(figsize=(16, 8))
223
224  plt.subplot(151)
225  plt.title('Original source label')
226  plt.imshow(source_label, 'gray')
227  plt.axis("off")
228
229  plt.subplot(152)
230  plt.title('Original target image')
231  plt.imshow(target_image, 'gray')
232  plt.axis("off")
233
234  # dst2 (source label => target image) 정의
235  dst2 = cv2.warpAffine(source_label, inv_mat, (w, h) )
236
237  # dst2 디스플레이
238  plt.subplot(153)
239  plt.title('dst \n (Source label => Target image)')
240  plt.imshow(dst2, 'gray')
241  plt.axis("off")
242
243  # back-projection 2 (dst2 => source label) 정의. nearest neighbor interpolation 옵션을 넣음.
244  inv_dst2 = cv2.warpAffine(dst2, mat, (w, h), flags=cv2.INTER_NEAREST)
245
246  plt.subplot(154)
247  plt.title('Back-projection \n (dst => source label)')
248  plt.imshow(inv_dst2, 'gray')
249  plt.axis("off")
250
251  # Target label 디스플레이
252  plt.subplot(155)
253  plt.title('Target label')
254  plt.imshow(target_label, 'gray')
255  plt.axis("off")
256
257  plt.show()
258
```

217~258 : **(문제 8)** (문제 4)에서와 동일한 조건에서 레이블 변환을 진행함.



결과 : (문제 4)와 유사한 방식으로 변환이 진행되었음을 알 수 있다.

```
259  # Compute the DSC score between the transformed label and the target label.
260  def dice_coef2(y_true, y_pred):
261      y_true_f = y_true.flatten()
262      y_pred_f = y_pred.flatten()
263      union = np.sum(y_true_f) + np.sum(y_pred_f)
264      if union==0: return 1
265      intersection = np.sum(y_true_f * y_pred_f)
266      return 2. * intersection / union
267
268  true = np.array(target_label[:,:,1])
269  pred = np.array(inv_dst2[:,:,1])
270
271  print('Dice Similarity Score : ', dice_coef2(true, pred))
272
```

259~272 : DSC score를 비교한다.
(문제 4)에서 사용한 DSC score 함수를 다시 만들어서 변환된 레이블과 타겟 레이블의 Dice score를 비교한다.

```
Dice Similarity Score :  0.0096406399924425O4
```

DSC = 0.0964063... 의 결과가 나왔다.

(문제 4)에서와 마찬가지로 DSC를 넣을 때 인풋 이미지가 제대로 설정되지 않은 것 같은데, 해결하지 못하였다.