GPI, LPI를 이용하여 손에 눈을 합성한 이미지 생성

2021204045 이성민

본 리포트는 사전에 촬영한 손 사진과 눈 사진을 이용하여 손 사진에 눈을 합성하는 과정을 기록한 것이다. 중간 과정 및 중간 사진을 통해 손에 눈을 합성하는 절차를 상세히 설명하고, 더 자연스러운 합성을 위해 사진을 2차 가공하여 시도한 내용을 추가적으로 소개한다.

1. 이미지 촬영

휴대폰으로 촬영한 얼굴 사진(**face.jpg**)과 손 사진(**hand.jpg**)이 있다. 두 사진은 동일한 시간대와 장소, 그리고 환경에서 촬영되었지만, 얼굴 사진은 휴대폰의 앞 카메라로, 손 사진은 뒤 카메라로 찍혔기 때문에 서로 크기가 다르다. 그래서 두 사진 모두 동일 크기로 맞출 뿐 아니라, 목표 크기인 640 x 640 픽셀로 조정할 필요가 있다.

2. 크기 맞추기 (실제로 제출할 .py 파일에는 절대 경로 대신 상대 경로로 지정되어 있다.)

```python
hand_img =cv2.imread('C:/Visual_Studio_Code/Visual_Computing/hand.jpg')
# 이미지의 원본 크기
hand_height, hand_width =hand_img.shape[:2]

# 가로 세로 비율 계산
hand_aspect_ratio =hand_width /hand_height

# 이미지 크기 조정 (짧은 쪽을 640으로 맞춤)
if hand_aspect_ratio >1:
    # 가로가 더 긴 경우, 세로를 640으로 맞추고 가로를 비율에 맞춰 조정
    new_hand_height =640
    new_hand_width =int(640 *hand_aspect_ratio)
else:
    # 세로가 더 긴 경우, 가로를 640으로 맞추고 세로를 비율에 맞춰 조정
    new_hand_width =640
    new_hand_height =int(640 /hand_aspect_ratio)


# 이미지 크기 조정
hand_resized_image =cv2.resize(hand_img, (new_hand_width, new_hand_height))
```

```
# 중심을 기준으로 640 x 640 크기로 잘라내기
hand_x_center =new_hand_width //2
hand_y_center =new_hand_height //2

hand_x_start =hand_x_center -320
hand_y_start =hand_y_center -320

hand640X640_img =hand_resized_image[hand_y_start:hand_y_start +640, hand_x_start:hand_x_start +640]

# 결과 저장
cv2.imwrite('C:/Visual_Studio_Code/Visual_Computing/hand640X640.jpg', hand640X640_img)
```

사진 크기 640 x 640

기존의 head.jpg를 가로 세로 비율을 유지하면서 640 x 640 크기로 축소한다. 이 과정에서 가로 세로 중 더 짧은 쪽을 640 픽셀로 설정하여 긴 쪽을 잘라냄으로써 화면 비가 맞지 않아 생기는 검정색 칸이 보이지 않게 처리한다. 이후 중심을 기준으로 640 x 640 크기로 잘라내 최종적으로 **hand640X640.jpg**를 만든다.

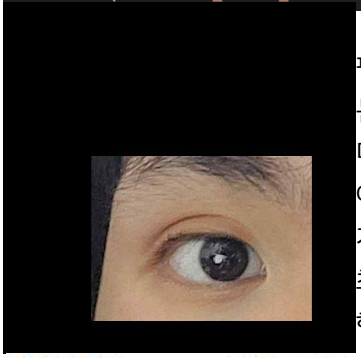
```
# 눈의 좌표(640 X 640 기준)
eye_x_center, eye_y_center =500, 1500

# 크기 자르기 (좌상단 좌표, 우하단 좌표 계산)
width, height =400, 300 # 대충 큰 크기
half_width =width //2
half_height =height //2

top_left_x =eye_x_center -half_width
top_left_y =eye_y_center -half_height
bottom_right_x =eye_x_center +half_width
bottom_right_y =eye_y_center +half_height

# 이미지 자르기
eye_img =face_img[top_left_y:bottom_right_y, top_left_x:bottom_right_x]

# 결과 저장
cv2.imwrite('C:/Visual_Studio_Code/Visual_Computing/eye.jpg', eye_img)
```

사진 크기 400 x 300

face.jpg의 크기를 맞추는 과정은 2단계로 나뉜다. 첫 번째 단계는 얼굴에서 눈만 떼어내는 과정이다. 최종 크기인 640 x 640보다 작으면서 적당히 큰 크기로 눈 부위만 떼어내어 **eye.jpg**로 저장한다.

```
# 손 이미지의 크기와 맞춤. 합성에 필요 없는 부분이니 검정색으로 처리
non_masked_eye_img =np.zeros((640, 640, 3), dtype=np.uint8)

# 눈 이미지를 삽입할 좌표
insert_x =160
insert_y =280

# 눈 이미지의 크기
eye_height, eye_width =eye_img.shape[:2]

# 손 이미지에 삽입할 공간의 끝 좌표
```

```
end_x =min(insert_x +eye_width, non_masked_eye_img.shape[1])
end_y =min(insert_y +eye_height, non_masked_eye_img.shape[0])

# 잘라낼 부분 계산
cropped_eye_image =eye_img[0:(end_y -insert_y), 0:(end_x -insert_x)]

# 검은 이미지에 눈 이미지를 삽입
non_masked_eye_img[insert_y:end_y, insert_x:end_x] =cropped_eye_image

# 결과 이미지 저장
cv2.imwrite('C:/Visual_Studio_Code/Visual_Computing/non masked eye.jpg', non_masked_eye_img)
```



eye.jpg가 완성되면, 이제 손의 중앙에 눈을 올려두는 작업이 필요하다. 먼저 640 x 640 크기의 검은 이미지를 만든다(이 크기는 **hand640X640.jpg**와 같아야 GPI, LPI를 통한 합성이 가능하다). 그 검은 이미지에 만들어둔 eye.jpg를 넣고, 손의 중앙에 눈이 위치하도록 eye.jpg의 위치를 조정한다. 이 과정에서 eye.jpg가 검은 이미지를 넘어설 경우, 이를 잘라낸다.

최종적으로 640 x 640 크기이면서 손의 중앙 위치에 눈이 위치한 합성에 최적화된 **non_masked_eye.jpg**가 완성된다. (파일 이름이 non_masked인 이유는 2차 가공 단계에서 설명할 예정이다.)

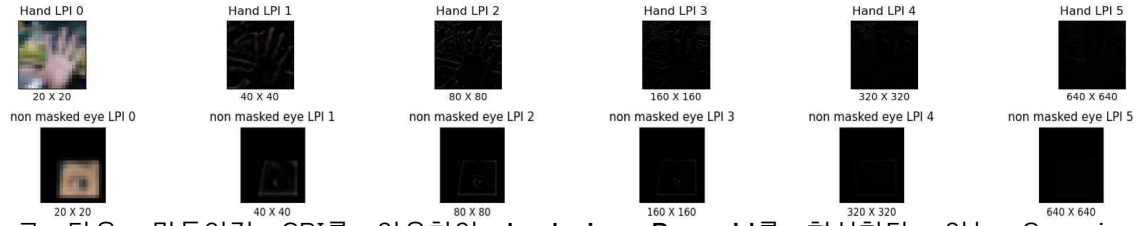
3. GPI, LPI를 이용한 합성

```
# hand640X640_img에 대한 Gaussian pyramid 생성
GA =hand640X640_img.copy()
gpA = [GA]
for i in range(6):
    GA =cv2.pyrDown(GA)
    gpA.append(GA)

# non_masked_eye_img 대한 Gaussian pyramid 생성
GB =non_masked_eye_img.copy()
gpB = [GB]
for i in range(6):
    GB =cv2.pyrDown(GB)
    gpB.append(GB)
```

먼저 **Gaussian Pyramid**를 이용하여 이미지를 여러 해상도로 반복적으로 필터링하고 축소하여 계층 구조를 형성한다. 위의 표는 6단계의 Gaussian Pyramid를 이미지의 크기와 함께 나타낸 것이다. 여기서 알 수 있듯이, 한 단계를 거칠 때마다 이미지의 가로와 세로 크기가 1/2로 줄어든다. Gaussian Pyramid는 이미지의 저해상도 버전을 생성함으로써 곧 실행될 이미지 합성에 활용할 수 있다.

```
# hand640X640_img에 대한 Gaussian Pyramid로 생성된 결과로부터 Laplacian Pyramids를 생성
lpA = [gpA[5]]
for i in range(5, 0, -1):
    GEA =cv2.pyrUp(gpA[i])
    if GEA.shape !=gpA[i-1].shape:
        GEA =cv2.resize(GEA, (gpA[i-1].shape[1], gpA[i-1].shape[0]))
    LA =cv2.subtract(gpA[i-1], GEA)
    lpA.append(LA)
```

```python
# non_masked_eye_img 대한 Gaussian Pyramid로 생성된 결과로부터 Laplacian Pyramids를 생성
lpB = [gpB[5]]
for i in range(5, 0, -1):
    GEB =cv2.pyrUp(gpB[i])
    if GEB.shape !=gpB[i-1].shape:
        GEB =cv2.resize(GEB, (gpB[i-1].shape[1], gpB[i-1].shape[0]))
    LB =cv2.subtract(gpB[i-1], GEB)
    lpB.append(LB)
```



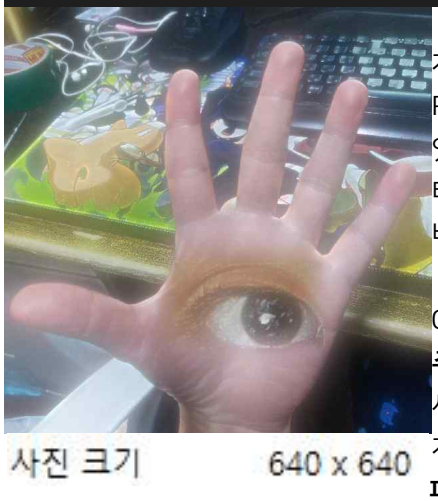
그 다음, 만들어진 GPI를 이용하여 **Laplacian Pyramid**를 형성한다. 이는 Gaussian Pyramid의 각 레벨의 차이를 강조하여 생성된다. 아래 표는 5단계의 Laplacian Pyramid를 이미지의 크기와 함께 나타낸 것이다. 가장 눈에 띄는 점은 Laplacian Pyramid의 **첫 번째 단계가 컬러로 출력**된다는 것이다. 이를 이해하기 위해 먼저 Laplacian Pyramid의 이미지가 흑백으로 보이는 이유를 설명하겠다.

Laplacian Pyramid는 이미지의 디테일과 경계를 나타내기 위해 높은 레벨의 Gaussian Pyramid 이미지를 낮은 레벨로 확장(**cv2.pyrUp**)한 후, 두 이미지의 차이(**cv2.subtract**)를 계산하여 만드는 피라미드이다. 이 과정은 색상보다는 이미지의 경계, 즉 강한 변화가 있는 부분에서 주로 나타난다. 즉, 이미지의 고주파 성분인 세부 경계와 텍스처만을 담고 있다는 의미다. 이로 인해 색상 정보는 거의 사라지고 경계 부분만 남아 흑백처럼 보인다. 여기서 Laplacian Pyramid의 첫 번째 단계는 Gaussian Pyramid에서 가장 작은 해상도를 가진 이미지다. 이 레벨은 이미지의 전체적인 구조를 가지고 있어 상대적으로 색상을 더 많이 유지할 수 있기 때문에 컬러로 나타난다.

또한 표에서 볼 수 있듯이, Gaussian Pyramid와는 반대로 마지막 단계에서 첫 번째 단계까지 역주행한다. 이로 인해 한 단계를 거칠 때마다 이미지의 가로와 세로 크기가 2배로 커지는 것을 확인할 수 있다. Laplacian Pyramid는 이미지의 다양한 세부 정보와 경계를 효과적으로 캡처하여 이미지 복원, 압축, 특징 추출에 사용할 수 있다. 물론 곧 실행될 이미지 합성에 또한 사용할 수 있다.

```python
AB = []

# Laplacian 피라미드의 각 레벨에 이미지의 위쪽과 아래쪽 절반을 추가
for la, lb in zip(lpA, lpB):
    rows, cols, dpt =la.shape

    # 위쪽은 손 사진에서, 아래쪽은 눈 사진에서 가져옴
    lsab =np.vstack((la[0:int(rows *0.6), :], lb[int(rows *0.6):, :]))

    # 남은 눈 사진의 왼쪽 부분을 손 사진으로 채움
    lsab[int(rows *0.6):, 0:int(cols *0.4)] =la[int(rows *0.6):, 0:int(cols *0.4)]

    # 남은 눈 사진의 오른쪽 25% 영역을 손 사진으로 덮음 (눈 사진의 전체 영역 중 오른쪽 25%)
    lsab[:, int(cols *0.75):] =la[:, int(cols *0.75):]

    # 남은 눈 사진의 아래쪽 20% 영역을 손 사진으로 덮음 (눈 사진의 전체 영역 중 아래쪽 20%)
    lsab[int(rows *0.8):, :] =la[int(rows *0.8):, :]
```

```
    AB.append(lsab)

hand_with_non_masked_eye =AB[0]
for i in range(1, 6):
    hand_with_non_masked_eye =cv2.pyrUp(hand_with_non_masked_eye)
    if hand_with_non_masked_eye.shape !=AB[i].shape:
        hand_with_non_masked_eye =cv2.resize(hand_with_non_masked_eye, (AB[i].shape[1], AB[i].shape[0]))
    hand_with_non_masked_eye =cv2.add(hand_with_non_masked_eye, AB[i])

cv2.imwrite('C:/Visual_Studio_Code/Visual_Computing/hand with non masked eye.jpg', hand_with_non_masked_eye)
```



사전에 눈의 위치를 손의 중앙에 맞췄기 때문에, 두 이미지의 Gaussian Pyramid와 이를 통해 만들어진 Laplacian Pyramid를 이용하여 위치 조정 없이 손과 눈을 합성할 수 있다. 눈을 가리지 않는 범위 내에서 손 사진을 사각형 형태로 눈을 감싸 합성하였고, 눈은 손의 중앙에 자연스럽게 배치되었다.

사진이 원본보다 뿌옇게 보이는 이유는 각 레벨에서 Gaussian 필터를 적용하면서 **고주파 성분이 제거되고 저주파 성분이 강조**되기 때문이다. 고주파 성분은 이미지의 세부 정보와 경계선에 해당하므로, 이들이 제거되면 이미지가 흐릿해질 수 있다. 하지만 더 큰 문제는 **눈 주변의 피부색과 손의 피부색이 달라서 합성이 부자연스럽게 보인다는 점이다.**

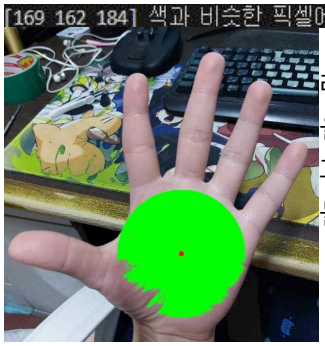
4. 2차 가공

우리는 지금까지 손 사진에 눈 사진을 합성하는 과정을 살펴보았다. 그러나 눈 주위의 피부색과 손의 피부색이 달라 합성 사진이 부자연스러운 문제가 발생하였다. 이를 해결하기 위해 2차 가공을 통해 손 중앙의 평균 색상을 눈 주위의 피부색에 적용하는 방법을 사용할 수 있다.

```
# hand640X640_img을 hand640X640_debug로 복사
hand640X640_debug =hand640X640_img.copy()

# 기준이 되는 손의 색을 추출할 좌표
hand_center_coords = (360, 460)

# 좌표에 빨간 점 찍기 (디버깅용)
cv2.circle(hand640X640_debug, hand_center_coords, radius=5, color=(0, 0, 255), thickness=-1)

# 색상 추출
hand_center_color =hand640X640_debug[hand_center_coords]

# 색상 범위 설정 (색상이 비슷한 픽셀 찾기)
lower_bound_hand_color =np.maximum(hand_center_color -40, 0)
upper_bound_hand_color =np.minimum(hand_center_color +30, 255)

# 마스크 생성
hand_mask =cv2.inRange(hand640X640_debug, lower_bound_hand_color, upper_bound_hand_color)

# 거리 기반 조건 추가: 마스크에서 손 중심과의 거리 계산
height, width, _=hand640X640_debug.shape
for y in range(height):
    for x in range(width):
        if hand_mask[y, x] !=0:  # 색상이 비슷한 픽셀인 경우
```

```
            distance =np.sqrt((x -hand_center_coords[0]) **2 + (y -hand_center_coords[1]) **2)
            if distance >130:  # 거리가 130보다 크면 마스크에서 제외
                hand_mask[y, x] =0

# 마스크에 해당하는 부분의 색상 평균 계산
hand_color =cv2.mean(hand640X640_debug, mask=hand_mask)

# 색상의 평균 출력
print(f"{hand_center_color} 색과 비슷한 픽셀에 대한 평균 색상: {hand_color[:3]}")  # B, G, R 순서로 출력

# 변환할 부분 설정
hand640X640_debug[hand_mask !=0] = (0, 255, 0)  # 초록색

# 결과 출력
cv2.imshow('Image with Red Dot', hand640X640_debug)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

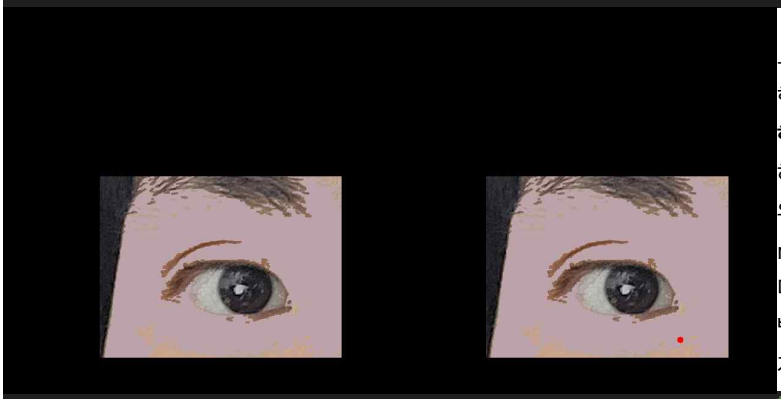
```
[169 162 184] 색과 비슷한 픽셀에 대한 평균 색상: (166.00614355231144, 159.23852392538524, 184.45879967558798)
```



먼저 손 중앙의 좌표를 구한 후, 해당 좌표에서 색이 비슷하고 거리가 130 이하인 픽셀들의 평균 색상을 계산한다(이는 손의 색상만을 구하기 위함이다). 옆 사진에서 초록색 마스크로 덮인 부분이 이 과정에서 참조된 모든 픽셀들을 나타낸다. 거리가 130 이하인 픽셀들만 참조했기 때문에 원형에 가까운 형상을 볼 수 있다.

```
# non_masked_eye_img을 masked_eye_img로 복사
masked_eye_img =non_masked_eye_img.copy()

# 기준이 되는 얼굴의 색을 추출할 좌표
face_center_coords = (480, 550)

# 색상 추출
face_color =masked_eye_img[face_center_coords]

print(f"{face_center_coords}의 색: {face_color}")

# 색상 범위 설정 (색상이 비슷한 픽셀 찾기)
lower_bound_face_color =np.maximum(face_color -40, 0)
upper_bound_face_color =np.minimum(face_color +10, 255)

# 마스크 생성
face_mask =cv2.inRange(masked_eye_img, lower_bound_face_color, upper_bound_face_color)

# 마스크를 한층 깔끔하게 만들기 위한 마스크 확장(침식 & 팽창)
kernel =np.ones((3, 5), np.uint8)  # 1x3 커널 생성(눈이 양 옆으로 길기 때문에 커널 또한 양 옆으로 길게 설정)
face_mask =cv2.dilate(face_mask, kernel, iterations=1)  # 팽창을 통해 작은 영역을 연결
face_mask =cv2.erode(face_mask, kernel, iterations=1)   # 침식을 통해 원래의 크기로 복원


# 변환할 부분 설정
masked_eye_img[face_mask !=0] =hand_color[:3]  # mean_color의 첫 세 요소 사용


# masked_eye_imgg을 masked_eye_with_red_dot_img로 복사
masked_eye_with_red_dot_img =masked_eye_img.copy()

# 좌표에 빨간 점 찍기(디버깅용)
cv2.circle(masked_eye_with_red_dot_img, face_center_coords, radius=5, color=(0, 0, 255), thickness=-1)
```

```
# 결과 출력
cv2.imshow('Masked Eye with Red Dot Image', masked_eye_with_red_dot_img)
cv2.waitKey(0)
cv2.destroyAllWindows()

# 결과 저장
cv2.imwrite('C:/Visual_Studio_Code/Visual_Computing/masked eye.jpg', masked_eye_img)
```



다음으로, 눈 사진에서 특정 피부의 좌표를 구한 후, 그와 비슷한 색을 가진 픽셀들에 **'앞서 구한 손 중앙의 평균 색상'**을 적용하는 방식으로 마스크를 씌운다. 왼쪽 이미지는 이렇게 만들어진 masked eye.jpg이고, 오른쪽 이미지는 특정 피부 좌표를 표시한 빨간 점이 포함된 디버깅용 이미지다.

```
# masked_eye_img 대한 Gaussian pyramid 생성
GC =masked_eye_img.copy()
gpC = [GC]
for i in range(6):
    GC =cv2.pyrDown(GC)
    gpC.append(GC)
# masked_eye_img 대한 Gaussian Pyramid로 생성된 결과로부터 Laplacian Pyramids를 생성
lpC = [gpC[5]]  # 리스트로 초기화
for i in range(5, 0, -1):
    GEC =cv2.pyrUp(gpC[i])  # 여기서 pyrUp의 결과를 lpC가 아닌 GEC에 저장
    if GEC.shape !=gpC[i-1].shape:
        GEC =cv2.resize(GEC, (gpC[i-1].shape[1], gpC[i-1].shape[0]))
    LC =cv2.subtract(gpC[i-1], GEC)
    lpC.append(LC)

AC = []

# Laplacian 피라미드의 각 레벨에 이미지의 위쪽과 아래쪽 절반을 추가
for la, lc in zip(lpA, lpC):
    rows, cols, dpt =la.shape

    # 위쪽은 손 사진에서, 아래쪽은 눈 사진에서 가져옴
    lsac =np.vstack((la[0:int(rows *0.6), :], lc[int(rows *0.6):, :]))

    # 남은 눈 사진의 왼쪽 부분을 손 사진으로 채움
    lsac[int(rows *0.6):, 0:int(cols *0.4)] =la[int(rows *0.6):, 0:int(cols *0.4)]

    # 남은 눈 사진의 오른쪽 25% 영역을 손 사진으로 덮음 (눈 사진의 전체 영역 중 오른쪽 25%)
    lsac[:, int(cols *0.75):] =la[:, int(cols *0.75):]

    # 남은 눈 사진의 아래쪽 20% 영역을 손 사진으로 덮음 (눈 사진의 전체 영역 중 아래쪽 20%)
    lsac[int(rows *0.8):, :] =la[int(rows *0.8):, :]

    AC.append(lsac)

hand_with_masked_eye =AC[0]
for i in range(1, 6):
    hand_with_masked_eye =cv2.pyrUp(hand_with_masked_eye)
    if hand_with_masked_eye.shape !=AC[i].shape:
        hand_with_masked_eye =cv2.resize(hand_with_non_masked_eye, (AC[i].shape[1], AC[i].shape[0]))
```

```
    hand_with_masked_eye =cv2.add(hand_with_masked_eye, AC[i])

cv2.imwrite('C:/Visual_Studio_Code/Visual_Computing/hand with masked eye.jpg', hand_with_masked_eye)
```

이후, 이전에 했던 합성 과정을 다시 거치면 한층 더 자연스러운 합성 이미지인 **hand with masked eye.jpg**를 얻을 수 있다.

5. 결론

본 리포트에서는 손 사진과 눈 사진을 합성하는 과정을 **GPI(Gaussian Pyramid)**를 이용한 이미지 축소와 **LPI(Laplacian Pyramid)**를 활용한 이미지 차이 계산을 통해 구현하였다. 첫 번째 합성 결과에서는 눈 주위의 피부색과 손의 피부색이 어울리지 않는 문제가 발생했으나, 이를 해결하기 위해 손 중앙의 평균 색상을 계산하여 눈 주위에 적용하는 2차 가공을 진행하였다.

결과적으로, 이러한 과정은 합성을 더욱 자연스럽게 만들어주었고, 색상의 부조화를 손 중앙의 평균 색상을 통해 보정하는 방법을 제시하였다. 이처럼 이미지 합성 작업에서는 색상, 경계선, 질감 등의 요소가 중요한 역할을 하며, 이를 효과적으로 처리하기 위해 다양한 이미지 처리 기법을 적절히 활용하는 것이 필요하다는 점을 확인할 수 있었다.