REPORT

“ 디지털영상처리 3차 과제 ”

과 목 명	디지털영상처리
담당교수	김진헌 교수님
학 과	컴퓨터 공학과
학 번	2016305078
이 름	최영환
제 출 일	2021.12.22

목 차

< 과제 소개 및 설명 >

1. 기본 사항 – matplotlib 출력 화면에서 해야 할 일

(1) 런닝맨 멤버가 포함된 사진에서 멤버를 찾아 사진에서 검출된 인물의 얼굴에 사각형으로 표시한다.

(2) 사각형 박스의 좌측 상, 하단에 0번부터 얼굴 번호를 적어넣는다.

(3) 멤버라고 판단한 인물 박스 상단에는 영문 이니셜을 적어 넣는다.

(4) 멤버가 아니면 unknown으로 명기한다.

(5) 종합적인 정보를 아래와 같이 plt.title()에 적는다.(fontsize=20 추천)

[파일의 이름] 검출한 얼굴의 수, 멤버라고 판단한 인물의 수

사례: [File: sample.png] 4 faces found including 2 members

2. 추가 가점 사항 – 추가 정보 출력

(1) 수행창에서 다음의 정보를 print()하여 제공

얼굴 검출 번호 별 판단한 멤버의 이니셜(혹은 이름)과 그때의 유클리디언 거리

(2) matplplot 창에서 그림으로 보이기

검출된 인물 별로 그 사람이라고 판단하게 된 얼굴을 3순위까지 선택하여 유클리디언 거리 순으로 표시, 이때 타이틀에는 그때의 유클리디언 거리를 함께 보인다.

< 프로그램 요약 >

※ 본 과제를 구현한 프로그램의 흐름을 간단히 요약하면 아래와 같다.

1. 테스트 할 이미지 내에서 검출된 모든 얼굴들의 위치와 디스크립터를 획득한다.

2. 검출된 얼굴들에 대한 판별을 진행하고, 결과를 토대로 과제의 요구 사항을 충족하기 위한 처리를 진행한다.

3. 얼굴 부분의 박스와 해당 얼굴의 정보(텍스트)를 이미지에 추가한다.

4. 멤버로 판별된 얼굴 부분 이미지와, 가장 가깝다고 판단한 후보 3개를 유클리디언 거리와 함께 출력한다.

※ 본 페이지부터는 프로그램에 관한 설명입니다.

< 프로그램 구현 >

※ dlib model, 테스트, DB 등의 파일 로드 및 변수 선언은 대부분 생략하였습니다.

1. 테스트 할 이미지 내에서 검출된 모든 얼굴들의 위치와 디스크립터를 획득한다.

```
# 입력된 사진에 대한 인코딩 진행
face_locations, face_dscrptr_lst = face_encodings(rgb,
                                                  number_of_times_to_upsample=0)
```

테스트 할 이미지를 face_encodings 메소드에 매개변수로 전달하면서 호출하고, 출력값으로 얼굴의 위치 정보와 디스크립터 리스트를 받는다. 이때, number_of_times_to_upsample 를 0으로 전달하여 여러 명의 얼굴을 검출하도록 한다.

```
# face_encodings은 128차원의 ndarray 데이터들을 사람 얼굴에 따라 리스트 자료형과 얼굴 위치좌표를 반환
def face_encodings(face_image, number_of_times_to_upsample=1, num_jitters=2):
    """Returns the 128D descriptor for each face in the image"""
    # Detect faces:
    # hog 기반의 dlib face detector. gray 변환된 영상을 사용.
    gray = cv.cvtColor(face_image, cv.COLOR_RGB2GRAY)
    face_locations = detector(gray, number_of_times_to_upsample)
    # Detected landmarks:
    raw_landmarks = [pose_predictor_5_point(face_image, face_location)
                     for face_location in face_locations]
    # Calculate the face encoding for every detected face using the detected landmarks for each one:
    return face_locations, \
           [np.array(face_encoder.compute_face_descriptor(face_image, raw_landmark_set, num_jitters))
                           for raw_landmark_set in raw_landmarks]
```

hog 기반의 face detector 가 사용되었으며, 이때, detector에는 grayscale로 변환된 영상을 전달해준다. 최종적으로 정보를 가지고 있으며, 얼굴의 위치정보를 가지는 face_locations는 얼굴 위치 표현과 출력에 face_encoding 메소드는 주석에 언급된 바와 같이 128차원의 ndarray 데이터들을 반환한다. 이 데이터들은 검출된 얼굴에 대한 사용되며, face_descrptr_lst는 이후 진행될 본 프로그램의 주요 루틴인 얼굴분류에 사용될 것이다.

※ face_encodings 메소드는 배포된 자료를 참조하였음

< 프로그램 구현 >

2. 검출된 얼굴들에 대한 판별을 진행하고, 결과를 토대로 과제의 요구 사항을 충족하기 위한 처리를 진행한다.

```
# 검출된 모든 얼굴에 대한 처리 진행
for i, (unknown_encoding, loc) in enumerate(zip(face_dscrptr_lst, face_locations)):
    # 검출된 얼굴을 나중에 보여줄 용도로 임시로 저장한다.
    face_cut = img[loc.top():loc.bottom(), loc.left():loc.right()].copy()
    face_cut = face_cut[..., ::-1]  # RGB 형식으로 변환

    # 검출된 얼굴의 인코딩과 미리 준비해 놓은 인코딩을 비교하여 누군인지 판별한다.
    computed_distances_ordered, ordered_faces, ordered_names = compare_faces_ordered \
        (all_member_face, all_member_encodings1d_lst, all_member1d_label_lst, unknown_encoding)

    # 판별결과를 통해 검출된 얼굴에 대한 처리 진행
    if computed_distances_ordered[0] > threshold:
        id = 'Unknown'
    else:
        id = ordered_names[0]              # id : 일치하는 멤버 이름
```

앞서 얻은 얼굴의 위치 정보를 통하여 검출된 얼굴부분의 영상을 잘라내어 이후 출력을 위해 따로 저장하고, 검출된 얼굴의 디스크립터와 기존에 저장된 런닝맨 멤버들의 디스크립터를 compare_faces_ordered 메소드에 매개변수로 전달하고, 동시에 기존에 저장된 멤버들의 얼굴 사진과 라벨 값을 같이 넘겨주고, 출력 값을 computed_distances_ordered, ordered_faces, ordered_names 변수에 저장한다. 각각 정렬된 유클리디언 거리 리스트와 정렬된 얼굴 사진, 라벨 값을 표현한다. 유클리디언 거리의 첫번째 값이 threshold 의 값보다 크다면 멤버가 아닌 것으로 분류하고 id 값을 Unknown으로 두고, 이하인 경우에는 일치하는 것으로 판단하고, 가장 닮았다고 판단된 멤버의 이름을 id 값으로 저장한다.

```
def compare_faces_ordered(faces, encodings, face_names, encoding_to_check):
    """Returns the ordered distances and names
    when comparing a list of face encodings against a candidate to check"""
    # 매칭값 순으로 나열하여 반환한다. ... 작은 값부터
    # 매칭값에 따라 face_names 순서도 바꾸어 반환한다.

    distances = list(np.linalg.norm(encodings - encoding_to_check, axis=1))
    return zip(*sorted(zip(distances, faces, face_names)))
```

compare_faces_ordered 메소드의 루틴은 단순하다. 비교할 인코딩 값들의 유클리디언 거리를 오름차순으로 정렬함과 동시에 다른 값들도 이에 맞춰 정렬한다.

※ face_encodings 메소드는 배포된 자료를 참조하였음

< 프로그램 구현 >

3. 얼굴 부분의 박스와 해당 얼굴의 정보(텍스트)를 이미지에 추가한다.

```
# 검출된 얼굴을 박스로 표시한다.
font = cv.FONT_HERSHEY_DUPLEX
cv.rectangle(img2, (loc.left() + 6, loc.top() + 6), (loc.right(), loc.bottom()), (255, 0, 0), 4)
# 박스 위에 검출된 얼굴의 번호를 출력한다.
cv.putText(img2, str(i), (loc.left() - 30, loc.top() + 30), font, 1.5, (0, 0, 255), 3)
# 박스 위에 검출된 얼굴의 이름을 출력한다.
cv.putText(img2, id, (loc.left(), loc.top() - 5), font, 1.0, (0, 0, 255), 2)
```

간단한 루틴이다. 검출된 얼굴부분을 박스로 표시하고, 그 얼굴이 검출된 얼굴들 중에서 몇 번 얼굴인지와, 그 얼굴의 id 값을 출력용 이미지에 추가한다.

4. 멤버로 판별된 얼굴 부분 이미지와, 가장 가깝다고 판단한 후보 3개를 유클리디언 거리와 함께 출력한다.

```
    # 결과 출력 부분
    print(f"face {i}: ", end='')
    # 런닝맨 멤버 중 한명인 경우
    if id != 'Unknown':
        matching_count += 1
        print(f"{ordered_names[0]}={computed_distances_ordered[0]:#.3f}")

        # 분석화면(win_anlys)으로 넘어가 원본 영상에서 검출된 얼굴을 출력한다.
        plt.figure(num=win_anlys)
        plt.subplot(4, 8, matching_count)
        plt.axis('off')
        plt.imshow(face_cut)
```

런닝맨 멤버들 중 한명이라도 판단이 되었을 경우의 출력 부분이다. 해당 얼굴에 대한 정보(멤버 중 누구로 판별되었는지와 유클리디언 거리)를 출력하고, 검출된 얼굴 이미지를 분석화면에 추가한다.

< 프로그램 구현 >

```
        # 원본과 가장 가깝다고 판단한 3개의 후보 사진을 아래에 유클리디어 거리와 함께 열로 나열한다.
        for th in range(3):
            face = ordered_faces[th]
            plt.subplot(4, 8, (8 * (th + 1)) + matching_count)
            plt.title(f"{computed_distances_ordered[th]:#.3f}")
            plt.axis('off')
            plt.imshow(ordered_faces[th])
    # 런닝맨 멤버가 아닌 경우
    else:
        print("Unknown")

# 다시 화면을 바꾸어 검출된 얼굴의 신원과 검색번호와 함께 얼굴을 사각형으로 표시한 화면을 출력한다.
plt.figure(num=im)
plt.imshow(img2[..., ::-1])
plt.title(f"[File: {im}] {len(face_locations)} faces found including {matching_count} members", fontsize=20)
plt.axis('off')

print(f'[File: {im}] {len(face_locations)} faces found including {matching_count} members')
```

이후, 유클리디언 거리가 가장 가까웠던 3개의 후보 사진을 검출된 얼굴의 아래에 유클리디언 거리와 함께 나열한다. 런닝맨 멤버가 아닌 경우 Unknown 을 출력하고, 화면을 테스트 이미지에 대한 출력 화면으로 변경하고, 박스와 텍스트가 추가된 영상을 출력한다. 이후의 출력문들은 전부 과제의 요구사항들이다.

최종적으로 show 메소드를 통해 모든 화면이 출력되면서 본 프로그램은 종료된다.
이로써 프로그램 구현 부분에 대한 설명을 마치고 결과 관측 및 결과 분석 부분으로 넘어가도록 한다.

※ 본 과제의 프로그램은 직접 메소드나 루틴 구현이 아니므로, 결과에 대한 관측과 분석에 중점을 두었습니다.

※ 또한, 이후 결과 관측 부분에서 print 문에 대한 결과도 본 보고서에 추가할 경우 양이 너무 많다고 판단되어, 시연영상에서만 해당 결과를 보였습니다.

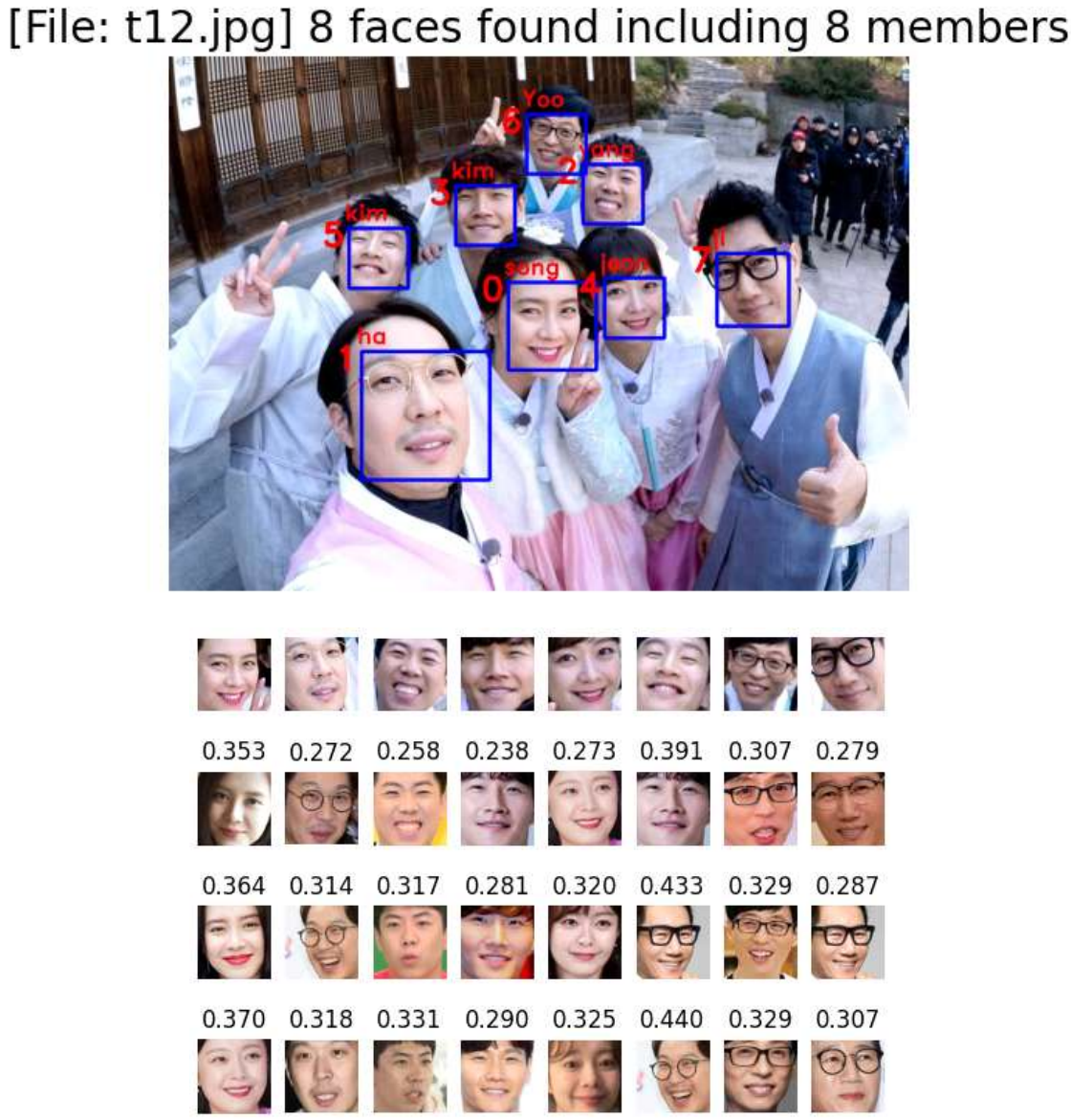
※ 본 페이지부터는 결과 관측 및 분석입니다.

< 결과 화면 >

※ threshold 의 값이 0.6인 경우만 보였습니다.

사례1. 목표 : 모든 얼굴 검출, 멤버 7명, 비 멤버 1명 판별

결과 : 모든 얼굴 검출, 멤버 8명 판별

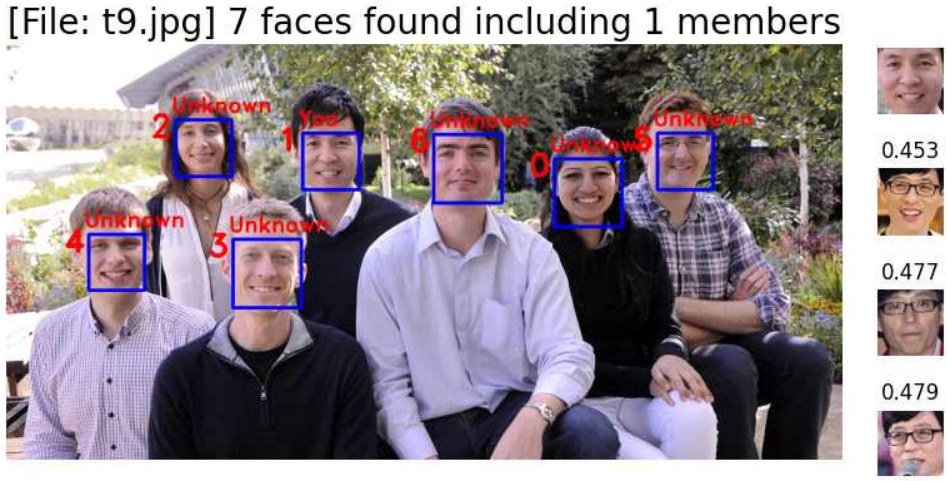


모든 얼굴 검출에는 성공하였으나, 이광수가 김종국으로 판별이 되어있다. 1위는 김종국으로 판별 되었으며, 이후 2, 3위의 데이터를 보면, 각각 지석진과 하하로 판별된 것을 확인 할 수 있었다. 다른 사진들은 대부분 0.2 ~ 0.3 사이인 반면, 이들은 거리가 0.4에 근접하거나, 큰 값으로 계산되었다. 'Unknown' 으로 분류 되었어야 하는데, 서로 다른 인물 셋 중 한명으로 판별하고 있으며, 이는 "해당 사진이 완벽한 정면이 아니기 때문이기도 하고, 실제로 비슷해 보이긴 한다." 라고 판단하였다.

< 결과 화면 >

사례2. 목표 : 모든 얼굴 검출, 멤버 0명 판별

결과 : 모든 얼굴 검출, 멤버 1명 판별



모든 얼굴 검출에는 성공하였으나, 엉뚱하게 유재석이 존재한다고 판별되었다.
유클리디언 거리의 값을 확인해보면 상위 3개의 값은 0.45를 넘는 값이 출력되었다.
전원이 정면인 사진이고, 화질도 좋으며, 사진의 크기도 적당하였는데, 유재석이 검출되었다.

< 결과 화면 >

사례3. 목표 : 모든 얼굴 검출, 멤버 전원 판별

결과 : 모든 얼굴 검출 실패, 멤버 4명 판별

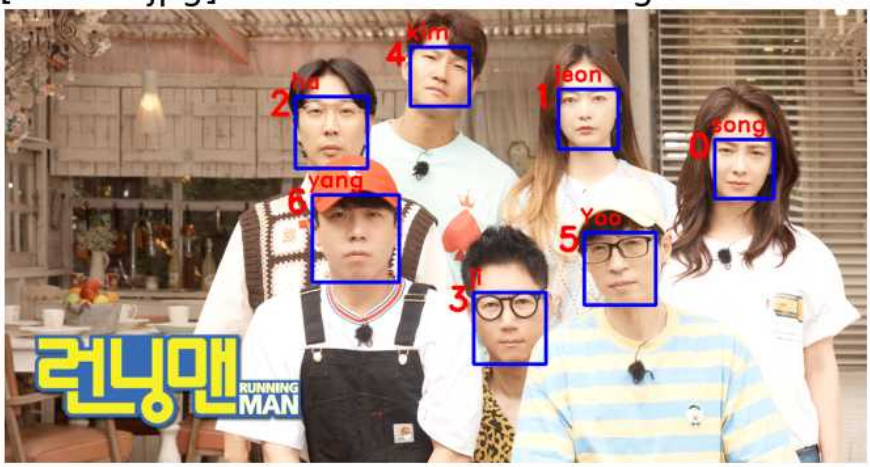
모든 얼굴 검출에 실패하였으나, 검출된 얼굴 모두를 멤버로 판별하는 것은 성공하였다.
양세찬의 얼굴이 검출되지 않은 이유는 완벽한 정면이 아니고, 얼굴의 크기가 별로 크지 않으며, 빛을 못 받아 어둡고, 화질이 좋지 못한 점이 이유가 될 수 있다.
그러나 검출된 얼굴에 한해서는 완벽하게 판별해내었다.

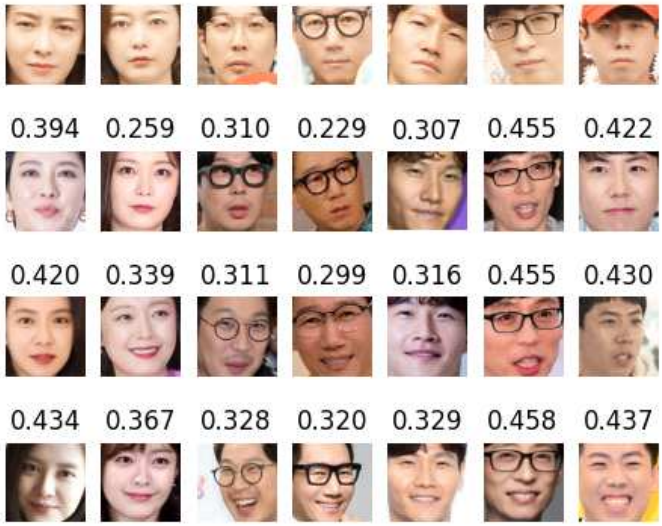
< 결과 화면 >

사례4. 목표 : 모든 얼굴 검출, 멤버 전원 판별

결과 : 모든 얼굴 검출, 멤버 전원 판별

[File: t2.jpg] 7 faces found including 7 members





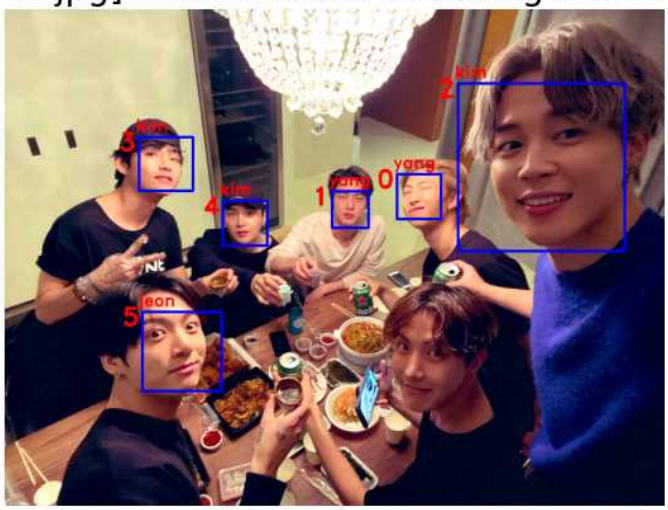
처음으로 완벽하게 성공한 사례이다. 모든 얼굴을 검출 하는 것이 성공하였으며, 멤버들을 판별하는 것도 성공하였다.

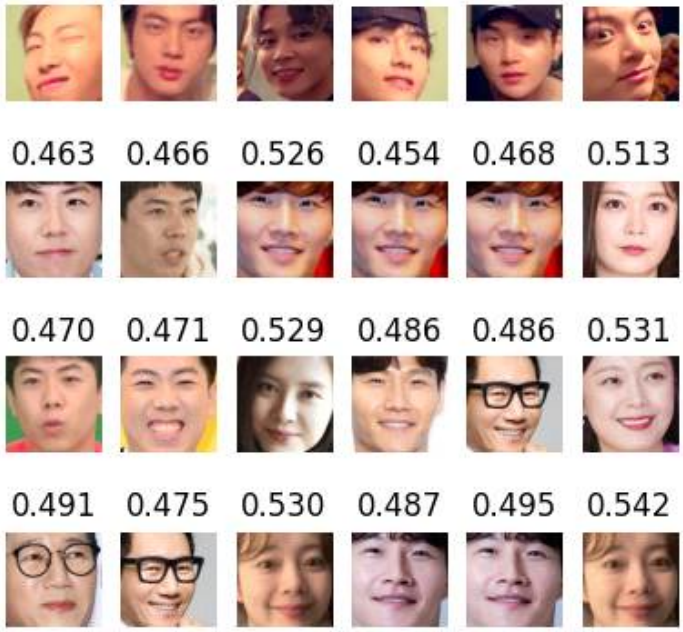
< 결과 화면 >

사례5. 목표 : 모든 얼굴 검출, 멤버 0명 판별

결과 : 모든 얼굴 검출 실패, 멤버 6명 판별

[File: t8.jpg] 6 faces found including 6 members





어느 하나 얻지 못한 실험 결과이다. 총 7개의 얼굴 중 하나를 검출해내지 못하였으며, 심지어 모두를 런닝맨의 멤버라고 인식하고 있다. 앞서 사례2의 경우에서도 동양인을 런닝맨의 멤버로판별 하였으나, 서양인은 아닌 것으로 판별하였고, 이번에도 동양인이었으므로, 여기서 동양인들끼리는 많이 차이가 나야 0.5 정도가 나는 것으로 예상할 수 있다.

종합하자면, 얼굴 검출을 못하는 것은 개선할 방법이 없는 상태이므로, 수용해야하지만,
판별에 대한 것은 threshold 의 값 조정을 통해 개선의 여지가 보이므로, 맞게 분류해낸 대부분의 상위 3개의 유클리디언 거리가 0.4 이하였던 것을 토대로 추가적인 테스트를 진행하였다.

< 결론 및 후기 >

1. 결론

(1) threshold 값에 대한 결론

앞서 언급하였듯이 판별이 된 경우 상위 3개에 대한 유클리디언 거리는 대부분 0.2 ~ 0.4의 값을 가진다. 이를 통해 동일 인물로 판단할 경우 0.2 ~ 0.4 사이의 유클리디언 거리 값을 가진다고 예상할 수 있었으며, 이후 0.4와 0.45로 설정하여 테스트를 진행하였다.

이후 얻어낸 결론은 0.4로 설정하는 경우가 더 정확하게 얼굴을 판별해낸다는 것이었고, 0.45는 닮은 사람도 판별해낸다는 단점이 있었다. 또한, 0.4보다 작은 값의 경우, 완벽한 정면을 보고 있지 않을 경우 동일 인물이더라도 동일 인물이 아닌 것으로 판별하는 경우도 존재하였다.
따라서 최종적인 threshold 의 값은 0.4로 설정하였다.

(2) 유클리디언 거리 사용 방법의 한계

유클리디언 거리를 사용하는 방법에 대한 자체적인 한계를 발견할 수 있었는데, threshold 값을 0.6 으로 설정 했을 때, 사례5의 BTS 멤버들이나 사례2의 동양인을 멤버로 판별한 것이 대표적인 예이지만, 이 예시는 threshold 값을 0.4로 설정할 경우 해결되는 문제였다.
가장 큰 문제는 사례 1과 사례 4에서 확인이 가능했다.

사례 1의 경우 이광수를 김종국, 지석진, 하하 순으로 판별한 것인데, 지석진과 하하의 경우는 0.4를 초과하기 때문에 충분히 틀린 값으로 저장이 가능하지만, 김종국의 경우 0.4 이하의 값으로 계산이 되었다. 물론 이 값보다 작은 값을 threshold 로 설정하면 해결이 가능하지만, 그럴 경우 동일인물을 동일인물로 판별하지 못하는, 이 판별 프로그램의 성능이 하락하는 역효과가 발생한다. 따라서 이 경우는 유클리디언 거리 사용 방법의 한계 중 하나의 사례로 볼 수 있었다.

사례4의 경우, 유재석에 대한 상위 3개의 유클리디언 거리 값이 모두 0.4를 넘는 큰 값으로, 해당 사례에서의 유재석은 threshold 값이 0.456 이상의 값이어야만 유재석으로 판별이 된다.
그럼 여기서 간단히 threshold 값을 0.456 으로 설정한다면 사례2와 사례5에서 엉뚱한 사람을 멤버로 판별하는, 앞서 언급된 프로그램 성능 하락의 역효과가 발생한다.
따라서 이 경우 또한 유클리디언 거리 사용 방법의 한계 중 하나의 사례라고 할 수 있다.

최종 결론으로, 유클리디언 거리 사용 방법은 제약 조건이 많은 얼굴 분류 방법이라고 생각한다.

2. 후기

공모전 출품작이 dlib를 사용하여 얼굴과 눈동자 검출을 기반으로 하는 작품이었기 때문에 해당 프로그램 구현과 라이브러리 사용 이해에는 큰 어려움이 없었고, 반가운 느낌도 존재했다. 그러나 기말고사 기간 등이 겹쳐 추가적인 테스트와 고찰을 할 시간이 부족해서 상당히 아쉬운 과제였다. 위와 같은 제약 조건이 없었다면 데이터를 더 많이 수집하여 프로그램의 성능이 더 좋았을 것이며, 더 많은 테스트와 고찰의 기회가 있었을 것이라고 생각이 든다.