[학부생 논문]영화 속 등장인물 분석 프로그램 구현

김도희 용자윤 김철연
숙명여자대학교
kdheejb7@naver.com nyongja@naver.com cykim@sm.ac.kr

Character Analysis Program in the movie

Dohee Kim, Jayoon Yong, Chulyeon Kim
Sookmyung Women's University

요 약

현대 사회에는 어디를 가도 카메라가 없는 곳을 찾아볼 수 없다. 매장의 CCTV부터 차량 블랙박스까지 우리는 실생활에서 수많은 카메라를 만나게 된다. 이렇게 많은 카메라가 생성해내는 사진에서 의미 있는 정보의 대표적인 예로 '사람의 얼굴'을 꼽을 수 있다. 본 논문에서는 영화 속 캐릭터 등장에 대한 요약을 제공하는 프로그램을 제안한다. 이 프로그램은 비디오 매체의 대표인 영화에서 사람 얼굴을 인식하여 인물의 등장 시점과 동시 등장 시점을 타임라인으로 제공하여 영화에 대한 요약 정보를 제공한다. 많은 인물이 나오는 대표적인 매체 영화에 대해 등장인물 분석을 제공하기 때문에 이를 활용할 수 있는 다른 분야가 무궁무진하다.

1. 서 론

현대 사회에는 어디를 가도 카메라가 없는 곳을 찾아볼 수 없다. 매장의 CCTV부터 차량 블랙박스까지 우리는 실생활에서 수많은 카메라를 만나게 된다. 이렇게 많은 카메라가 생성해내는 사진에서 의미 있는 정보의 대표적인 예로 '사람의 얼굴'을 꼽을 수 있다. 사람의 얼굴 정보를 이용할 수 있는 분야는 정말 무궁무진하다. 얼굴을 인식하여 사람을 구분함으로서 개인에 대한 데이터를 축적하여 더욱더 의미 있는 정보를 창출할 수도 있다. 예를 들어, 스마트폰에서 Face ID를 이용하여 잠금을 설정/해제할 수 있다.

더 나아가 사진뿐만 아니라 비디오에서도 사람의 얼굴을 인식하고 그 얼굴을 정확히 분류할 수 있다면 상당히 많은 분야에서 응용되어 사용할 수 있는 기술이 될 수 있을 것이라는 판단 하에 비디오에서 사람 얼굴을 분류하는 것에 대한 연구를 진행하기로 하였다. 수많은 비디오 매체 중에서도 많은 인물이 나오는 대표적인 매체가 영화이기 때문에 영화를 선택하였다.

본 논문에서는 영상 매체의 대표인 영화 속에서 얼굴 정보를 기반으로 등장인물을 찾고 분석하는 프로그램을 제안한다. 이 프로그램은 I-Frame에 등장하는 얼굴을 찾아내어 등장 빈도와 등장 시점을 계산하여 주인공인지 엑스트라인지를 판별한 후, 주인공들의 등장 시점과 동시 등장 시점을 타임라인으로 제공한다.

이 프로그램의 목표는 다음과 같다. 영화 속에서 등장인물의 얼굴 정보를 수집하여 동일 인물인지 여부를 판별하고, 등장 횟수를 기반으로 주요인물인지 엑스트라인지를 알아낸다. 그 후, 주요 등장인물의 등장 시점을 타임라인을 이용하여 시각적으로 표현한 후 같이 등장하는 인물 정보까지 제공하여 인물 간 친밀도를 파악하는 것이 프로그램의 목표이다.

2. 시스템 구현

2.1 시스템 요약

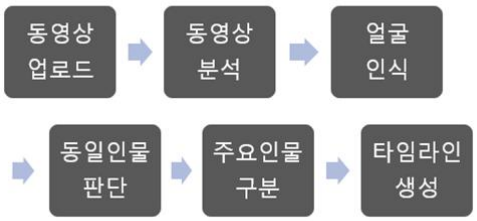

그림 1. 시스템 개요

동작 원리는 다음과 같다. 먼저, 영상의 프레임을

ffprobe를 이용하여 추출한다. 모든 프레임 추출 시 중복되는 데이터의 수가 필요 이상으로 증가하여 추출한 프레임 중 큰 변화가 있는 I-frame만을 keyframe폴더에 저장한다. Facenet 라이브러리를 이용하여 I-Frame 속에서 얼굴을 인식하여 얼굴 부분만 잘라내어 저장한다 [1]. 인식한 얼굴 데이터를 Triplet loss 알고리즘을 이용하여 같은 얼굴을 찾아내는 작업을 한다 [2]. 이 때, 등장한 비율이 평균보다 높은 캐릭터의 경우 영상의 주요인물로 판단하여 등장한 프레임의 번호로 등장시점을 해당 영상의 데이터 파일(.txt)에 저장한다.

영상의 전체 길이에 따라 타임라인의 비율을 계산하고 Tkinter를 이용하여 타임바를 생성한다. 주요 인물의 등장한 프레임과 그 다음 I-Frame까지를 해당 인물의 등장 시간으로 계산하여 주요 인물 별 타임라인을 생성한다. 이때 같은 프레임에서 등장한 주요인물들의 경우 서로 관련도가 높다고 판단하여 동시 등장 타임라인을 함께 생성한다.

2.2. 시스템 기능 구현

2.2.1. I-Frame 및 얼굴 추출

```
for frame_no in i_frames:
    cap.set(cv2.CAP_PROP_POS_FRAMES, frame_no)
    ret, frame = cap.read()
    outname = 'key'+str(frame_no)+'.jpg'
    frameNum.append(frame_no)
    cv2.imwrite("../keyframe/"+outname, frame)
    f.write(str(frame_no)+str(" "))
    img = cv2.imread("../keyframe/"+'key'+str(frame_no)+'.jpg')
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


    img_size = np.asarray(img.shape)[0:2]
    bounding_boxes, _ = align.detect_face.detect_face
                (img, minsize, pnet, rnet, onet, threshold, factor)
```

그림 2. 얼굴을 추출하는 함수 중 일부

비디오 캡처를 이용하여 I-Frame을 추출하여 저장한 후, 각 프레임마다 openCV를 이용하여 얼굴을 찾아내 크롭하여 이를 저장한다 [3].

2.2.2. 얼굴 분류

```
if (len(dist_list) == 0 or min(dist_list) > 0.3):
    os.mkdir(image_path + str(i))
    pil_image = Image.fromarray(input_images[i])
    pil_image.save(image_path+str(i)+"/"+str(video_name[i])+".jpg")
    image_files=[]
    for k in os.listdir(image_path):
        if not k == '.DS_Store':
            for f in os.listdir(os.path.join(image_path, k)):
                ext = os.path.splitext(f)[1]
                if ext == '.jpg' or ext == '.jpeg' or ext == '.png':
                    image_files.append(os.path.join(image_path, k, f))

    images = load_and_align_data(input_images, image_files, image_size, margin, gpu_memory_fraction)

                    # Run forward pass to calculate embeddings
    feed_dict = { images_placeholder: images, phase_train_placeholder:False }
    emb = sess.run(embeddings, feed_dict=feed_dict)

elif (min(dist_list)<=0.3):
    dist_index = dist_list.index(min(dist_list))
    person_path = os.path.abspath(os.path.join(image_files[dist_index], '..'))
    person_name = person_path.split('/')[-1]
    pil_image = Image.fromarray(input_images[i])
    print(person_name)
    pil_image.save(person_name+"/"+str(video_name[i])+".jpg")
else :
    print("패스")
```

그림 3. 얼굴을 분류하는 함수 중 일부

찾은 얼굴을 같은 사람인지 아닌지 분류하는 compare 함수는 triplet loss 기술을 이용하여 구현하였다. triplet loss란, 얼굴 판별에 이용하는 딥러닝 기술로, 이미지 사이의 거리를 기반으로 학습한 후, 같은 얼굴과 다른 얼굴을 알아내는 기술이다. 이 때, threshold에 따라 얼굴 분류의 정확도가 달라지는데 몇 번의 시행착오 후 threshold가 0.3일 때 가장 분류가 잘 됨을 확인하였다.

2.2.3. 정확도 향상

```
med_num = 0
med_min = 1000000
for k in range(len(image_files)):
    dist_list = []
        # print(len(input_images))
            # print(len(emb))
    for j in range(len(image_files), len(emb)):
        dist = spatial.distance.cosine(emb[k,:], emb[j,:])
        dist_list.append(dist)
    sum = 0
    for l in dist_list:
        sum = sum + l

    if (sum <= med_min) :
        med_min = sum
        med_num = k
medoid_img.append(image_files[med_num])  #중앙 이미지 자체 저장
tmp_path.append(image_path[med_num])     #중앙 이미지의 path 저장
tmp_foldpath.append(medoid_path)
```

그림 4. 정확도를 높이는 함수 중 일부

같은 얼굴임에도 폴더가 두 개가 생기게 될 경우, 같은 사람이 두 개의 폴더에 나눠져 들어가는 현상이 발생하는 문제점을 발견하였다. 이를 해결하기 위해 medoid를 이용하여 폴더를 합치는 방법을 선택하였다. 이미지 간의 거리를 기반으로 하여. 각 폴더의 medoid 이미지를 찾은 후, medoid 이미지 끼리 compare를 한 번 더 진행하여 같은 얼굴로 판별될 경우 두 폴더를 합치는 작업을 진행하였다.

2.2.4. 주인공 찾기

```
def find_main_character(image_path):
    character_count = 0
    image_files = []
    count = []
    t = 0
    for i in os.listdir(image_path):
        k = 0
        if not i == '.DS_Store':
            for f in os.listdir(os.path.join(image_path, i)):
                ext = os.path.splitext(f)[1]
                if ext == '.jpg' or ext == '.jpeg' or ext == '.png':
                    image_files.append(os.path.join(image_path, i, f))
                k = k+1
        count.append(k)
        if count[t]!=0 :
            character_count = character_count + 1
        t = t+1

    print("총 얼굴 수 : "+str(len(image_files)))
    character_count = len(image_files)/character_count
    print("주인공 판별 기준 : " + str(character_count) +"이상")
    for i in range(len(count)):
        if(count[i]>=character_count):
            print(str(i)+"번째 주인공 : 폴더 " + str(os.listdir(image_path)[i])
            main_character.append(str(os.listdir(image_path)[i]))
```

그림 5. 주인공 찾는 함수 중 일부

각 폴더의 사진의 수를 총 폴더 수로 나눈 결과를 주인공 판별 기준으로 정했다. 각 폴더에는 해당 인물이 등장한 횟수만큼의 사진이 들어있고, 등장한 인물 별로 폴더가 생기기 때문에 위에서 제시한 기준이 평균 등장 횟수로 정하기에 적합했다. 평균 등장 횟수보다 많이 등장한 (폴더에 평균보다 많은 사진의 수가 들어간) 인물을 주인공으로 선정하였다.

선정된 주인공들에 대한 타임라인을 그리기 위한 데이터인 영상의 길이, 프레임 번호, 주인공으로 판별한 폴더

이름 및 대표 사진과 등장한 프레임 번호를 텍스트 파일로 저장해둔다.

2.2.5. 타임라인 생성

```
lines = f.readlines()
count = 0
im = []
for i in lines:
    character_emerge = i.split()
    if((len(character_emerge)-1)>=(float)(character_count[0])):
        for i in range(0,len(character_emerge)):
            for j in range(0,len(frameNum)-1):
                tmp = character_emerge[i].split('_')
                if(str(frameNum[j])==str(tmp[0])):
                    width__ = width_ - 255
                    check_start = (int)(frameNum[j])
                    check_end = (int)(frameNum[j+1])
                    ratio_start = int(float((width__/(int)(lastFrame))*check_start))
                    ratio_end = int(float((width__/(int)(lastFrame))*check_end))
                    thumbnail = Image.open("C:/project_youngk/data/"+file+"/"+
                                character_emerge[1]).resize((100,100), Image.ANTIALIAS)
                    im.append(ImageTk.PhotoImage(thumbnail))
                    canvas.create_image(50, count*110+80, image=im[len(im)-1], anchor='nw')
                    canvas.create_rectangle(260+ratio_start, count*110+80, 250+ratio_end,
                                100+count*(110)+80, fill = 'red')
    count = count + 1
```

그림 6. 타임라인을 생성하는 함수 중 일부

영상의 러닝타임을 기준으로 타임라인 막대의 개수와 막대의 간격을 계산한다. 영상의 길이에 따라 타임라인의 길이가 달라지는 것을 막기 위해 비율로 계산하였다. 전체 키프레임 데이터와 각 주인공이 등장한 프레임 번호 데이터를 이용하여, 주인공이 등장한 키 프레임부터 다음 키 프레임 까지를 해당 주인공의 등장 시간대로 계산하였다. 각 주인공 별 타임라인을 그린 후, 같은 프레임에 등장한 인물이 있는지 여부를 판단하여 인물 간 친밀도를 예측하는 기반 데이터로 삼는다.

3. 실행 결과

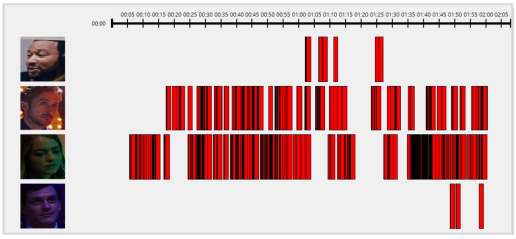

그림 7. 주요인물 타임라인

그림 8. 동시 등장 인물 타임라인

<그림9>와 <그림10>은 예시로 영화 '라라랜드'의 실행 결과이다. <그림 9>는 영화 러닝 타임에 따른 5분 단위의 타임바를 그리고 각 주요 인물들의 등장 시점 타임라인을 생성한 것이다. 포털 사이트에서 제공하는 주요인물 6명 중 4명의 인물을 찾아낸 것을 확인할 수 있다. <그림 10>의 경우 주요인물 중 동시 등장한 인물들에 대한 타임라인이다. 동시 등장한 횟수가 많은 인물들의 경우 서로에 대한 관계가 깊다고 판단하여 따로 타임라인을 제공한다. 실제 남자 주인공과 여자 주인공, 여자 주인공과 서브 남자 주인공을 함께 등장한 인물로 찾아낸 것을 확인할 수 있다.

4. 발전 방향 및 결과

4.1. 발전 방향

이 프로그램은 다양한 분야와의 결합이 기대되는 시스템이다. 먼저, 실시간 인식 기술과 결합된다면 실시간 영상 인식을 통하여 사람의 얼굴을 분류할 수 있게 될 것이고 이는 개인 맞춤형 키오스크 자동 주문, 자동 출결 시스템 등의 서비스로 발전할 수 있다. 또한, 음성 인식 기술과 결합한다면 영상에서 음성을 추출하여 목소리 별 분류를 한 후, 각 장면에 나오는 등장인물과 목소리 조합을 검사하여 어떤 주인공이 어떤 목소리를 가지는지를 알아낼 수 있게 될 것이다. 이는 음성에 해당하는 주인공의 라벨을 추가할 수 있게 되어 시각적 자료가 없더라도 영화의 흐름과 내용을 파악할 수 있다는 이점을 가지게 될 것이다.

4.2. 결론

현대인들은 생활 속에서 자연스레 수많은 카메라에 노출된다. 카메라에 잡히는 사람의 얼굴 데이터를 잘 가공하여 유용한 정보로 만든다면 사람들에게 편리함 및 이점을 가져다주는 여러 방향으로 이용할 수 있을 것이라 생각하였고, 그 초석을 다지기 위한 프로그램을 제안하였다.

이 프로그램은 동영상에서 사람의 얼굴을 분석 및 분류하여 주요 등장인물을 판별해주고 그 등장 시점까지 타임라인이라는 시각정보로 나타내준다. 이 프로그램의 개발은 생활 속 자연스레 얻어지는 무한한 얼굴 데이터를 가공하여 정보로 만들 수 있다는 확신을 줄 수 있다는 데 의의를 가진다. 또한 다른 분야와의 결합 측면에서도 매우 유용한 활용이 기대되는 프로그램이라고 할 수 있다.

참고문헌

[1] 김현곤, 석혜경, 낭종호 (2017). FaceNet의 Fine-Tuning을 통한 드라마 동영상에서의 한국 연예인 얼굴 검출에 대한 실험적 성능분석. 한국정보과학회 학술발표논문집, 826-828

[2] 강봉남, 김대진 (2017). 공동 손실 함수, Triplet, 스코어 레벨 융합 방법에 의한 Deep 얼굴 인식 방법. 한국정보과학회 학술발표논문집, 790-792

[3] 이웅규, 이제민, 정현종, 송인선, 낭종호 (2012). 방송 요약을 위한 중요 프레임 및 비 중요 프레임 검출. 한국정보과학회 학술발표논문집, 39(1A), 331-333