

OpenPose Report

길윤빈 202211259 : 팀장(코드 작성 및 실행 데이터 수집/ 정리)

박세준 202011295 : 팀원(전체 피드백 및 발표 자료 작성/ 발표)

장지은 202211363 : 팀원(코드 분석 및 보고서 작성)

정소현 202211368 : 팀원(결과 분석 및 보고서 작성)

목차

1. OpenPose의 기본 원리

- OpenPose System

- Confidence Map

- Affinity Fields

2. OpenPose 코드 분석(analysis)

- openpose 파일

- 1. body/estimator.py

- 2. body/model.py

- 3. utils.py

- 실행파일 (demo.py 분석)

3. 코드 추가 부분(Visualization)

- 터미널에서 파일명 보이기

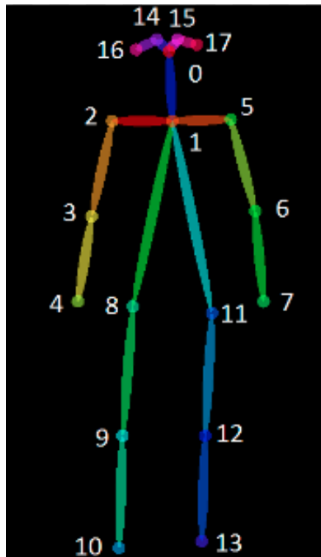
- output 사진 폴더에 저장

4. 실행결과 및 분석

- image.py

- video.py

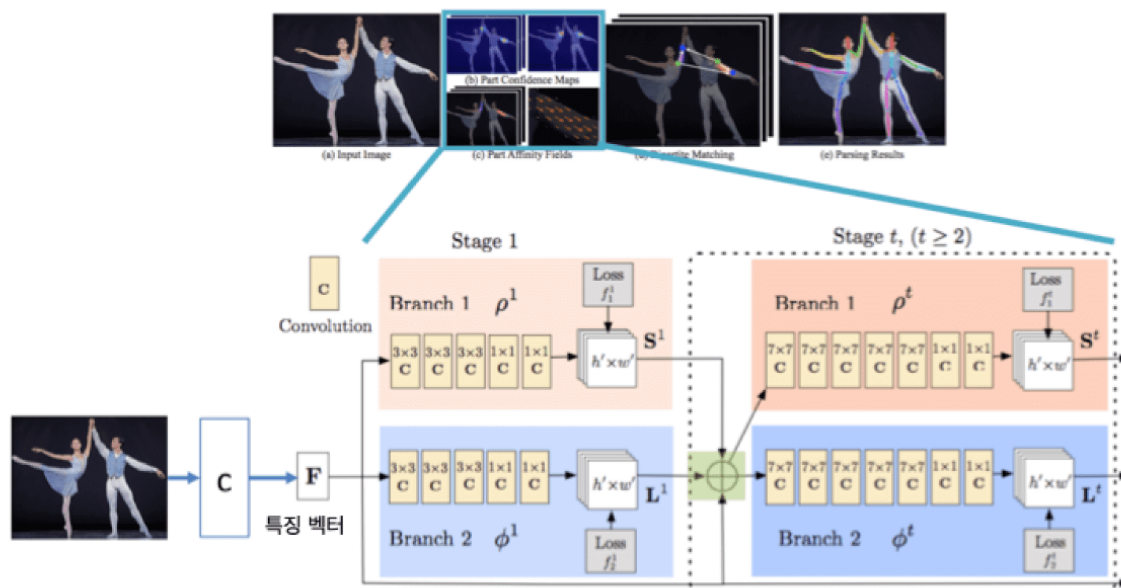
1) openpose의 기본 원리



<OpenPose api wrapper in PyTorch. Whole-body (Head, body, left-top, right-top) 2D Pose Estimation>

OpenPose는 이미지, 동영상에서 몸체, 머리, 왼쪽 상반신, 오른쪽 상반신으로 나누어 각각 **6, 4, 4, 4**개의 키포인트를 감지하여 연결해 다중으로 사람의 움직임을 감지한다.

<OpenPose System>



input으로 image가 들어오면 **VGG-19 network**와 같은 **convolution network**를 통해 **feature map(=F)**를 얻는다. F를 두 **branch**의 input으로 사용하는데, 각 **branch**는 **part confidence map**, **part affinity fields**를 예측한다. 각 **branch**에서 얻은 결과와 **stage1**의 input으로 사용한 **feature map**을 **concat**해서 다음 **stage**의 input으로 사용한다.

-> **confidence map** 추정시, 이전 **stage**의 **affinity field** 정보를 활용해 현재 **stage**의 **confidence map**을 추정한다. **affinity field** 추정시에도 **confidence map** 정보를 활용한다.

-> 이러한 과정으로 얻은 **confidence map**과 **affinity field** 를 통해 **key point**들을 얻고, **key points**끼리의 **connection**을 완성한다.

-Confidence Map



(b) Part Confidence Maps

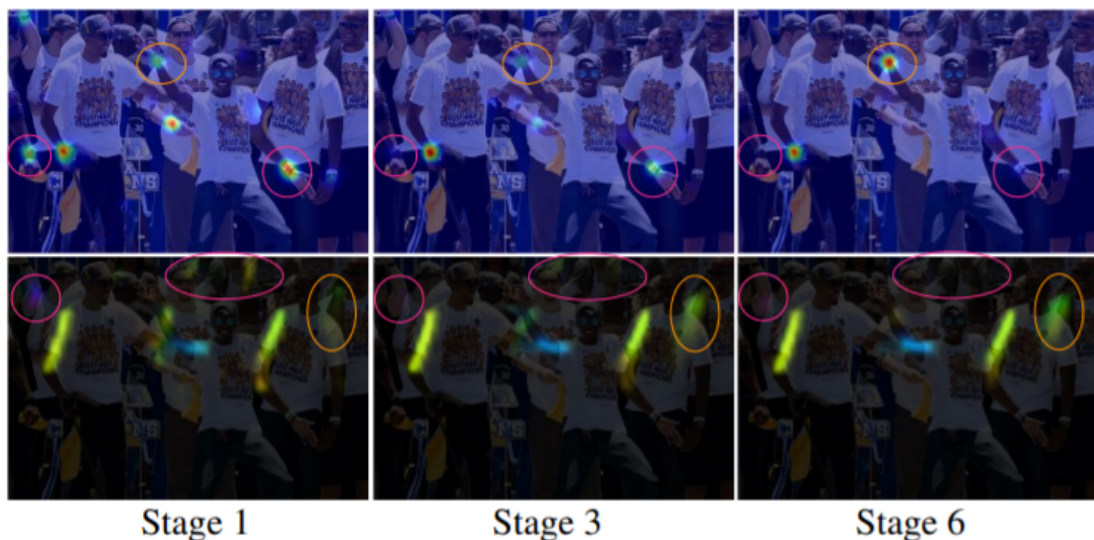
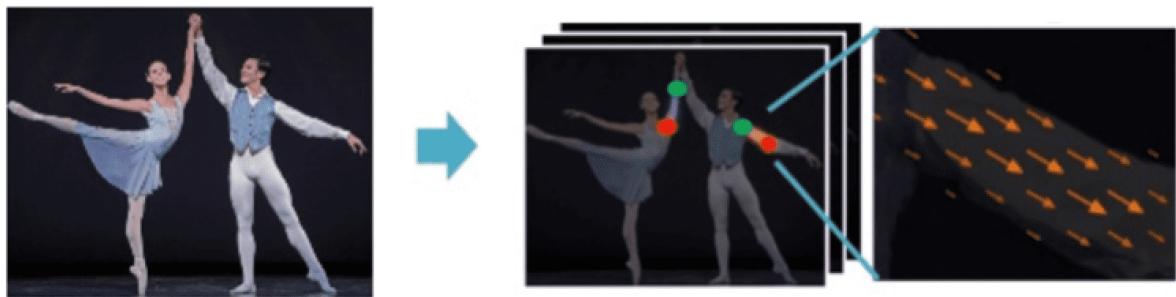


Figure 4. Confidence maps of the right wrist (first row) and PAFs (second row) of right forearm across stages. Although there is confusion between left and right body parts and limbs in early stages, the estimates are increasingly refined through global inference in later stages, as shown in the highlighted areas.

confidence map이란, 이미지를 보고 각 사람의 몸에서 어디에 관절이 있을지를 찾아내는 역할을 하는 것이다. 이 때, 위의 사진에서 **branch1(confidence map)**을 거쳐 나오면 사람의 관절이 있을것이라 예측되는 곳이 **heatmap**으로 표현되어 나오게 된다. 위 사진은 왼쪽에서 오른쪽으로 갈 수록 **stage**가

높아지는데 오른쪽 손목의 관절 위치를 예측한 결과물이다. 그 결과물은 위의 사진처럼 heatmap으로 표현되는데 오른쪽 사진으로 갈 수록 오른손목이라고 예측되는 곳의 heatmap의 선명도가 높아지는 것을 확인 할 수 있다. 이처럼 stage가 진행됨에 따라 confidence map과 part affinity fields를 정확하게 예측한다.

-Affinity Fields



affinity fields는 2차원 벡터공간이다. 이 벡터공간은 (위치, 방향)을 원소로 하고 있어 2차원 벡터공간이 되는데, 각 관절과 관절 사이에 사람 몸 위에서 방향관계를 표시함으로써 현재 관절 다음에 어느 방향에 있는 관절이 해당 사람 몸에 있는 다음 관절이 맞을 지를 예측하는데 도움을 준다.

2) openpose 코드 분석(analysis)

1. openpose 파일: openposw 프로젝트의 주 실행 파일, 사용자 인터페이스와 인자 처리 담당, 사용자가 입력한 설정에 따라 실행

-1. body/estimator.py

-신체 키포인트를 추정하는 핵심 기능.

- **_pad_image:** 입력 이미지 패딩, 비율 유지, 스트라이드, 패드 밸류 고려

- **_get_keypoint:** 후보 키포인트, 서브셋 구성으로 최종 키포인트 좌표 생성

-class BodyPoseEstimator(object):

OpenPose 딥 러닝 아키텍처를 사용하여 이미지에서 인체 자세를 추정,

모델 초기화, 이미지 전처리, 딥러닝 아키텍처에 이미지 전달하는 클래스, 모델 출력은 키포인트 위치와 히트맵, PAFs로 구성

평균화된 히트맵에서 각 키포인트의 피크를 추출, 임계값을 적용한 후 최종 키포인트를 구성 -> PAF를 사용하여 감지된 키포인트 간의 연결 형성 -> 감지된 키포인트 연결을 서브셋 배열로 구성하여 입력 이미지에서 발견된 가능한 인간 자세 나타냄

__init__: 토치 모듈 상속, 사전 학습된 모듈을 사용할 수 있도록 함

estimate: 이미지 전처리, 스케일로 크기를 조정하고 패딩을 적용

_pad_image: 이미지를 전처리하고 패딩을 추가하는 데 사용, 결과 이미지는 모델을 통과할 때 잡음이 줄어든 것으로 인식

body_part_heatmaps, pafs = self(*preprocessed_images): 전처리된 이미지가 모델에 전달된 후 중요한 결과인 신체 부위의 히트맵과 PAF(경로 밀도 필드) 반환

body_part_candidates_list, body_parts_subset_list = self._get_keypoints(heatmaps = body_part_heatmaps, pafs = pafs): 신체부위 히트맵과 PAF를 사용하여 실제 신체부위 키포인트 연결을 산출

return body_parts_subset_list, body_part_candidates_list: 클래스의 출력 값, 좌표로 구성된 키포인트 배열, 이 배열을 이용하여 감지된 인간 자세를 시각화하거나 분석가능

-2. body/model.py

-pose estimation(관절 각 추정)을 위한 모델을 정의.

-바른 구조(coco, mpi, body_25)에 따라 키 포인트의 개수와 연결 관계 정의

-모델에 대한 정보 저장, 텐서를 생성하여 body/estimator.py에 전달

-3. utils.py

-프로젝트 내에서 공통으로 사용되는 배열조작, FPS 계산 등의 기능을 제공

-draw_keypoints, draw_body_connections: 이미지에 키 포인트와 관절 연결을 그림

-draw_keypoints:

원본 이미지를 복사하여 새로운 이미지(overlay)생성

for kp in keypoints: for x,y,v in kp: x,y는 좌표이며 v는 표시 여부, 키포인트를 반복하면서 표시 여부가 True인 곳에 cv2의 circle을 이용하여 원을 그린 후 addWeighted를 사용하여 overlay와 원본이미지를 결합하여 return

-draw_body_connections:

원본 이미지 복사하여 새로운 이미지(overlay)생성.

b_conn: 몸체, h_conn: 머리, l_conn: 왼쪽 상반신, r_conn: 오른쪽 상반신 으로 나누어 연결선 간에 어떤 키포인트 연결해야 하는지 정의

_draw_connection으로 각 정의된 부분 연결

addWeighted로 원본 이미지와 결합 후 반환

-draw_face_connections, draw_hand_connections: 아직 구현 안됐으므로 에러 발생

-_draw_connection 함수 (helper 함수):

x1, y1, v1 = point1 / x2, y2 v2 = point2: 시작점과 끝점의 x, y 좌표 및 표시 여부를 각각 저장

if v1 and v2: 시작점과 끝점의 표시 여부가 모두 True일 때만 선을 그림.

cv2의 line을 이용하여 이미지에 시작점에서 끝점까지 선을 그림

그려진 이미지 return

2. 실행파일 (demo.py 분석)

1. 필요 라이브러리, 모듈 import: cv2로 이미지 처리, estimator.py의 BodyPoseEstimator 클래스를 불러와 신체 키 포인트 추정에 사용, utils.py의 draw_keypoints/draw_body_connections를 불러와 이미지 위에 키포인트와 관절 연결을 그림

2. BodyPoseEstimator 객체 생성: 키포인트 추정에 필요한 BodyPoseEstimator 객체를 생성. 이때 사전 훈련된 가중치를 사용하도록 설정(pretrained=True)

3. 이미지 불러오기: cv2의 imread를 사용하여 입력 이미지를 불러와 image_src에 저장

4. 키포인트 추정: BodyPoseEstimator 객체를 사용하여 이미지에서 키포인트를 찾고, 결과를 keypoints에 저장

5. 관절 연결 그리기: draw_body_connections 함수 사용, 파라미터로 두께 4, 투명도 0.7로 주어 이미지 위에 관절 연결을 그림.

6. 키포인트 그리기: draw_keypoints 함수 사용, 파라미터로 반지름 5, 투명도 0.8로 주어 키포인트를 이미지 위에 그림.

7. 결과 이미지 표시와 저장: cv2의 imshow, waitkey를 통해 생성된 이미지를 화면에 보여줌.

3)코드 추가 부분(visualizations)

(1) 터미널에서 파일명 보이기

수정 부분: requirement.txt 추가, examples에 input, output 파일 추가, 터미널창
-image_filename추가

코드 설명:

```
if __name__ == '__main__':  
    parser = argparse.ArgumentParser()  
    parser.add_argument('--image_filename', required=True, help='File name of video')  
    parser.add_argument('--save_option', required=True, help='Whether to save output as  
JPG(Answer in Y/N)')  
  
    args = parser.parse_args()  
  
    human_pose(args.image_filename, args.save_option)
```

import argparse를 통해 argparse를 불러와 parser의 add_argument를 사용하여
-image_filename, -save_option 설정을 추가한다.

(2) output 사진 폴더에 저장

수정 부분: requirement.txt 추가, examples에 input, output 파일 추가, 터미널창
-save_option 추가

코드 설명:

```
if(save_option == 'Y'):  
  
    cv2.imwrite("./output/" + image_filename.split('.')[0]+"_output.png",image_dst)
```

원래의 repository 코드는 단순히 시각화 된 이미지를 보여주지만 하고 저장하지 않는다.
따라서 시각화 된 이미지가 필요에 따라 저장되도록 하기 위해서 save_option을 추가하고,
해당 옵션이 설정되어 있으면 cv2의 imwrite를 이용하여 output파일 경로에 해당 결과물을
저장하도록 변경하였다.

(3) 이미지 파일 크기 조정

```
MAX_WIDTH = 1000
```

```

MAX_HEIGHT = 1000

image_src = cv2.imread(image_dir)

height, width, _ = image_src.shape

if width > MAX_WIDTH or height > MAX_HEIGHT:

    scale = min(MAX_WIDTH/width, MAX_HEIGHT/height)

    resized_width = int(width*scale)

    resized_height = int(height*scale)

    resized_image = cv2.resize(image_src, (resized_width, resized_height))

else:

    resized_image = image_src

```

원래의 repository 코드는 입력받은 이미지 크기를 조정하지 않고 사용하기때문에 이미지가 크기가 너무 큰 경우 output결과 팝업창에서 사진이 잘려보이는 경우가 발생하였다. 이를 방지하기 위해 cv2의 resize를 이용하여 사진크기에 제한을 두도록 변경하였다.

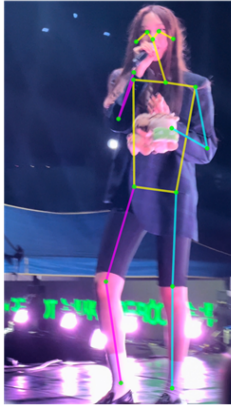


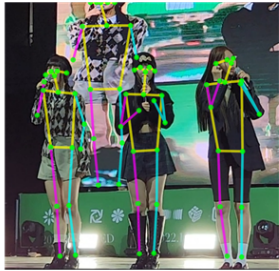
4)실행 결과 및 분석





-image.py





사진 속 인원 수에 따른 프로그램 런타임의 차이를 분석해보기 위해 다음과 같이 시행했다.

- **input**되는 사진들의 크기가 모두 달라,사진의 크기가 런타임에 주는 영향을 줄이기 위해 사진의 크기가 일정 크기를 넘으면 사진을 축소하는 코드를 추가해 실행
- 각 사진 별로 **10**회씩 코드를 실행
- 각 회차 별로 런타임을 기록
- 인원 수에 따른 평균 런타임을 구함
- 사진 속 사람의 수를 **x**축, 평균 런타임을 **y**축으로 하는 점 그래프를 그리고, 보간법을 통해 점들의 경향을 관찰.(여기서는 **4**차 다항식 보간법 사용)
- 결과분석

<실행결과_image>

1명	2명	3명	4명
			

5명	6명	7명	8명
			

9명	10명	11명	12명
			

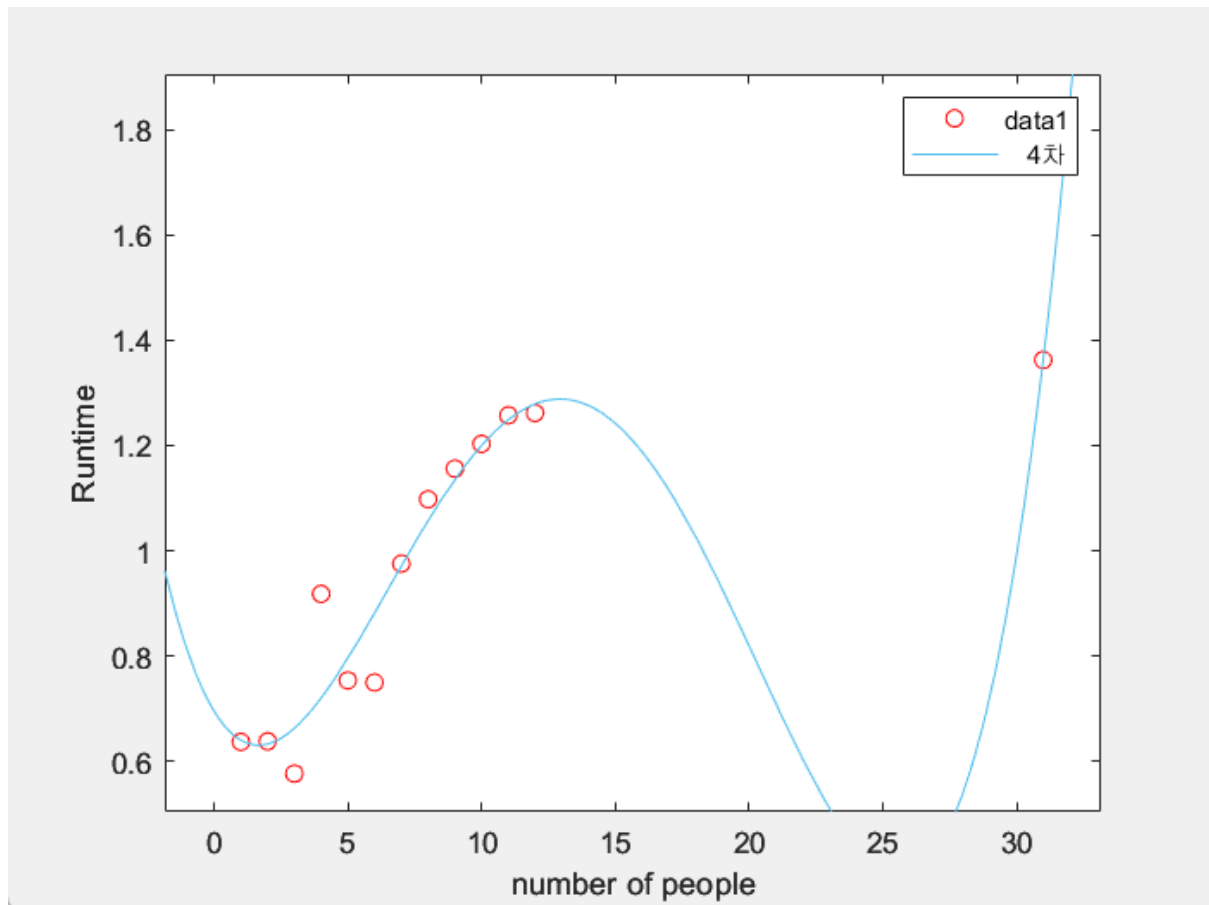


<31명>

<실행결과>

	1	2	3	4	5	6	7	8	9	10	avg
1p	0.731924	0.629802	0.660563	0.649281	0.63538	0.615961	0.582277	0.635907	0.666848	0.56659	0.637453
2p	0.713696	0.62635	0.706271	0.597249	0.606509	0.607126	0.588359	0.64148	0.605354	0.69142	0.638381
3p	0.566041	0.551017	0.571688	0.61673	0.550368	0.565829	0.584046	0.566352	0.631885	0.567224	0.577118
4p	0.865473	0.950094	0.970974	0.846015	0.921371	0.898348	0.939668	0.856227	0.991231	0.944018	0.918342
5p	0.698533	0.776829	0.696253	0.774707	0.740901	0.779801	0.687573	0.788841	0.813562	0.783276	0.754028
6p	0.805309	0.743569	0.756458	0.731766	0.681748	0.692843	0.751124	0.724769	0.772437	0.841989	0.750201
7p	1.015471	1.053477	0.926692	0.965471	1.002537	0.902554	0.994032	0.970961	0.957107	0.969379	0.975768
8p	1.157552	1.059886	1.106317	1.015594	1.079385	1.128143	1.139013	1.068554	1.113256	1.110745	1.097845
9p	1.202587	1.106478	1.332277	1.084012	1.230948	1.156819	1.137406	1.104304	1.156992	1.048948	1.156077
10p	1.28426	1.194611	1.173796	1.180585	1.133211	1.123275	1.284045	1.182108	1.234634	1.239639	1.203016
11p	1.449975	1.30387	1.174254	1.155181	1.231322	1.266159	1.220706	1.31247	1.253551	1.204459	1.257195
12p	1.177192	1.376265	1.294285	1.305545	1.277149	1.157941	1.173533	1.275421	1.300236	1.278272	1.261584
31p	1.432055	1.358507	1.369163	1.422794	1.374142	1.27606	1.315984	1.359236	1.37374	1.340938	1.362262

(노란색: 사진 속 인물들의 수, 선풍색: 시행횟수, 주황색: 평균 프로그램 런타임)



<실행행결과분석>

이 그래프의 **x**축은 사진 속 인물들의 수, **y**축은 **x**값에 따른 프로그램의 런타임을 나타낸다. 파란색 그래프는 점들의 값을 4차 다항식으로 보간한 그래프이다.

그래프를 보면, 사진 속 인원 수가 증가함에 따라 대체적으로 프로그램 런타임이 길어지는 것을 알 수 있다. 특히 1명에서부터 14명까지 인원수가 증가할수록 런타임도 급격하게 증가한다. 그러나 중간에 경향을 따르지 않는 점들도 존재하는데, 이는 사진 속 인물들의 형태가 정확하지 않거나 인원이 많아질수록 다른 인물들과 겹쳐져 있어, 프로그램이 인물들을 잘 인지하지 못해 발생하는 문제로 보인다.

이를 통해 사진 속 인물들의 수가 증가할수록 프로그램의 런타임이 대체적으로 증가한다는 것을 알 수 있다.

-video.py

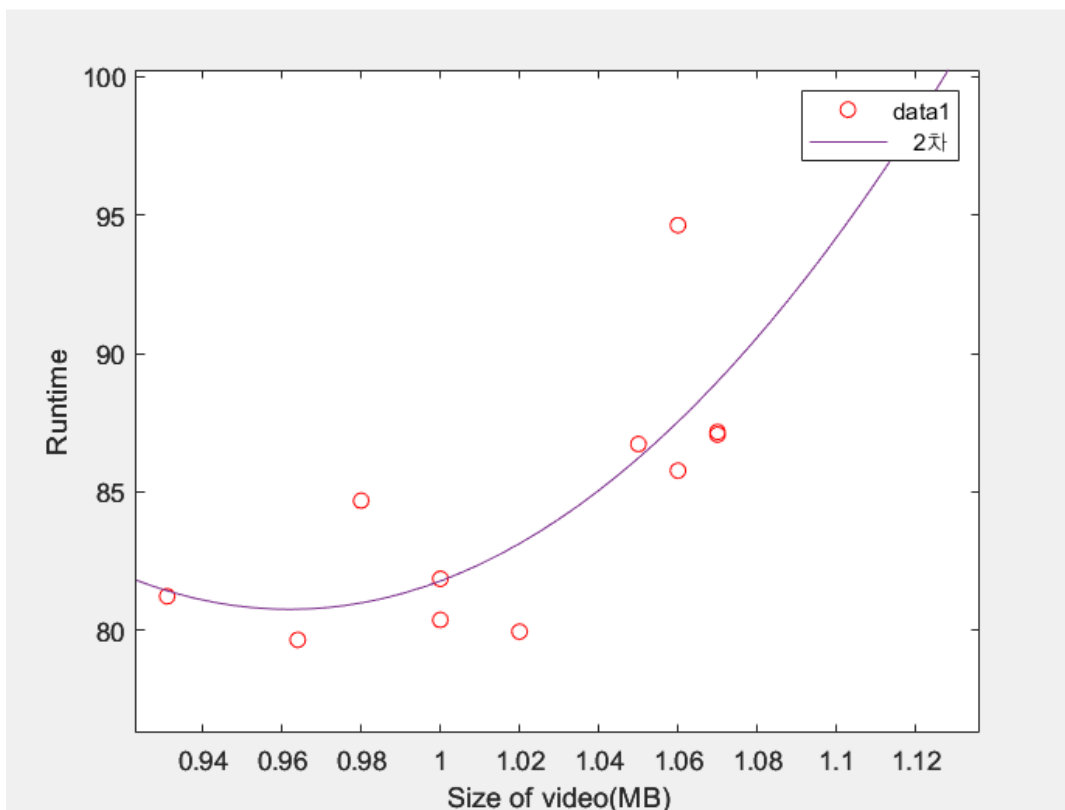
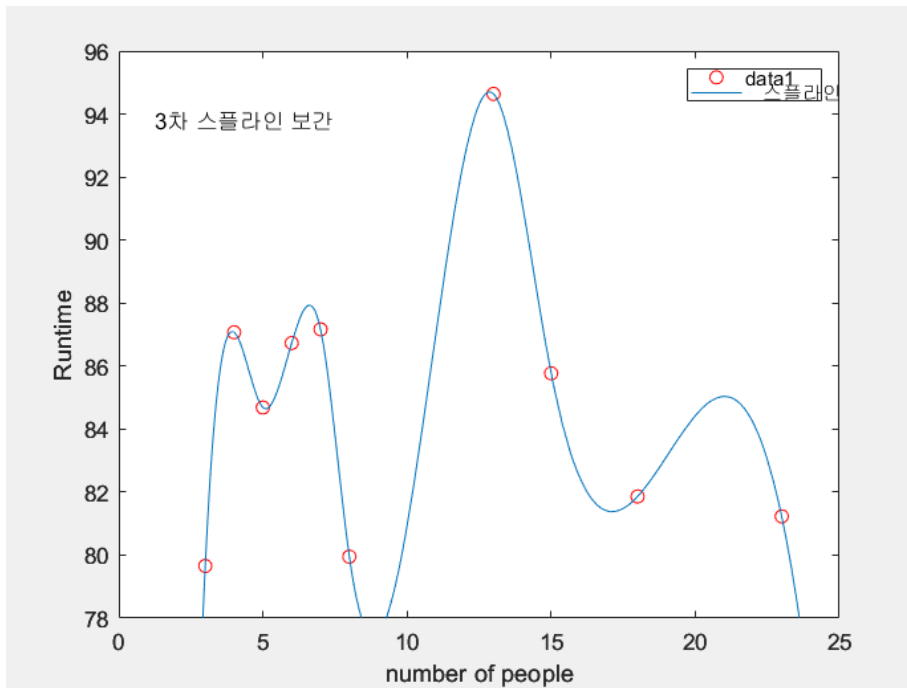
동영상 속의 인물들의 수에 따른 프로그램 런타임의 차이를 알아보기 위해 다음과 같이 시행했다.

- 모든 동영상을 4초로 준비
- 동영상은 사진보다 용량이 크기 때문에 실행횟수를 3번으로 함
- 각 실행 회차 별 프로그램 런타임 기록
- 동영상 속 인물들의 수에 따른 평균 프로그램 런타임 기록
- **x**축을 인물들의 수, **y**축을 평균 프로그램 런타임으로 하는 점 그래프를 그리고, 이를 보간하여 점들의 추이를 분석
- 결과분석

<실행결과>

	1	2	3	avg
3p	90.029731	74.153315	74.784789	79.655945
4p	87.202333	86.688174	87.325421	87.071976
5p	83.692993	80.724797	89.643831	84.687207
6p	88.015694	79.664966	92.51536	86.73200667
7p	80.857947	88.337987	92.303417	87.16645033
8p	77.431354	83.126368	79.295409	79.95104367
13p	88.092207	86.709658	109.1145473	94.6388041
15p	83.763013	86.684659	86.861064	85.76957867
18a	82.17085	82.350153	81.062467	81.86115667
23a	82.261418	82.257946	79.164582	81.227982

(노란색: 동영상 속 인물들의 수, 초록색: 실행횟수, 주황색: 평균 프로그램 런타임)



<실행결과 분석>

첫번째 그래프의 x축은 동영상 속 인물들의 수, y축은 x값에 따른 프로그램의 런타임을 나타낸다. 파란색 그래프는 점들의 값을 스플라인 보간법으로 보간한 그래프이다.

이 그래프를 보면 동영상 속 인물들의 수의 변화와 관계없이 프로그램 런타임이 측정되었다. 이는 동영상의 시간을 통일 할 때 0.n초 차이로 인해 동영상들의 프레임 수와 용량의 크기 차이가 생겨 그런 것으로 보인다. 동영상 속 인물들의 수는 프로그램 런타임의 크기에 큰 영향을 미치지 않는 것으로 보인다.

두번째 그래프의 **x**축은 동영상의 용량(**MB**), **y**축은 **x**값에 따른 프로그램의 런타임을 나타낸다.
보라색 그래프는 점들의 값을 2차 다항식으로 보간한 그래프이다.

이 그래프를 보면 동영상의 용량이 증가하면 어느정도는 프로그램의 런타임이 증가하는 것처럼 보인다.
그러나 뚜렷한 경향이 보이지 않고 용량이 증가해도 프로그램의 런타임이 증가하지 않는 점들도 많기에
용량 자체로는 프로그램의 런타임의 크기에 큰 영향을 미치지 않는 것으로 보인다.

두 그래프를 통해 유의미한 결과를 얻지 못했는데 이는 동영상의 용량을 통일하지 못했기 때문으로
추정한다. 그러나 동영상의 인물들의 수보다는, 용량의 크기가 프로그램의 런타임의 크기에 더 영향을
미치는 것을 알 수 있다.