OpenPose 이해

SangGyu Nam
halite@gmail.com
2018-07-31

목차

일러두기

OpenPose를 공부하고 그 내용을 정리한 문서로 다음 사이트를 참조 했고 그림과 코드를 가져다 분석해 사용했다.

1. Human pose estimation using OpenPose with TensorFlow (Part 2)
2. keras-openpose-reproduce

1. OpenPose

OpenPose 사이트는 https://github.com/CMU-Perceptual-Computing-Lab/openpose 이다. OpenPose는 real-time으로 인체를 감지해 얼굴, 팔, 다리 등의 주요 관절이 어디인지 찾아주고 그 관절의 연결 정보를 제공해 준다.

그림 1. OpenPose 예

그림 1은 신체 이미지를 OpenPose 모델을 사용해 prediction한 결과에서 관절과 연결 선 정보를 얻어 표시한 것이다. OpenPose는 한 이미지에 여러 사람이 있는 경우에도 각 사람에 대한 정보를 모두 추출해 내준다. 다만 이 경우는 각 사람에 속하는 관절과 연결 정보를 따로 구분하는 작업이 필요하다.

1.1. OpenPose 기초

OpenPose의 Neural Network prediction 결과물은 Heatmap과 PAF이다.

1.1.1. Keras Model

필자가 Keras를 선호해서 OpenPose를 Keras로 구현한 것을 찾아 사용했다 (https://github.com/kevinlin311tw/keras-openpose-reproduce).
이 사이트에서 training 된 model과 weight를 받을 수 있다. OpenPose 사이트에서도 언급하듯이 training이 좋은 GPU를 써서도 며칠 씩 걸리기 때문에 필자는 그냥 받아 사용했다(사실 OpenPose 속 내부를 공부하려는 게 목표가 아니기 때문에 결과 물에 대해서만 공부를 했다).

model 디렉토리의 get_keras_model.sh를 사용하면 최신 모델을 다운로드 할 수 있다. 혹은 직접 Dropbox(https://www.dropbox.com/s/76b3r8rj82wicik/weights.0100.h5?dl=0)에서 받을 수도 있다.

다운로드 받은 model은 keras.model의 load_model()을 사용하면 바로 사용할 수 있다.

1.1.2. OpenPose Model Output

Keras OpenPose Model은 코드 1과 같이 구성된다.

```
model = Model(inputs=[img_input], outputs=[stageT_branch1_out, stageT_branch2_out])
```

코드 1. Keras OpenPose Model

outputs는 2개가 지정되었고 첫번째는 PAF를 의미하고 두번째는 Heatmap을 의미한다.
그림 1과 같은 이미지는 320x320 크기이고 이 이미지에 대한 출력은 PAF가 (1, 40, 40, 38)의 shape을 갖고 Heatmap이 (1, 40, 40, 19)의 shape을 갖는다.
OpenPose는 입력 이미지를 8x8 크기의 이미지로 잘라 그 영역에 대한 PAF와 heatmap 결과를 출력해주도록 되어 있어 입력 이미지가 320x320 이므로 8로 나눈 40x40 영역에 대한 결과물을 만들어낸다.
그리고 40x40 영역에 대해 각각 PAF는 38개, Heatmap은 19개의 결과를 만들어 낸다. PAF와 Heatmap의 결과 개수는 OpenPose에서 정의한 것이다. 이렇게 해서 각각 (1, 40, 40, 38), (1, 40, 40, 19)의 출력을 만들어 내는데 dimension의 처음 1은 필요 없는 것이다.

1.1.3. Part, Pair

OpenPose를 이해하기 위해서는 OpenPose에서 이미 정의해 놓은 Part, Pair에 대해 이해할 필요가 있다.

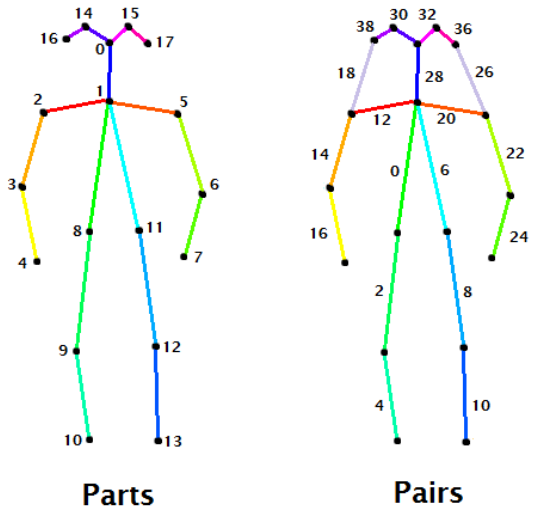

그림 2. OpenPose Part, Pair

Part는 Heatmap에서 얻어지는 정보로 관절 혹은 그림 2의 Parts에 표시된 것과 같은 위치를 의미한다. 0은 얼굴의 중심을 의미하고 14, 15는 눈 16, 17은 귀. 2, 5는 어깨 3, 6은 팔꿈치 4, 7은 손목을 의미한다. 이런 식으로 미리 정의된 정보가 Parts로 0~18까지 총 19개가 존재한다. 19개는 1.1.2에서 언급한 것과 같이 Heatmap에 해당하는 출력 shape의 19를 의미한다.

좀더 이해를 쉽게 하기 위해 Heatmap에 해당하는 출력 shape을 (1, 40, 40, 19)를 (19, 40, 40)으로 변경하면 좋다. (19, 40, 40)의 의미는 Part 0에 해당하는 정보가 40x40개 만큼 있다는 소리인데, 8x8 크기로 나눈 이미지에서 Part 0에 해당하는 정보를 추출해 40x40 크기로 만들었단 소리다.

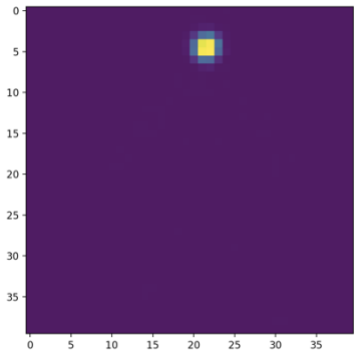

그림 3. 1명 PAF

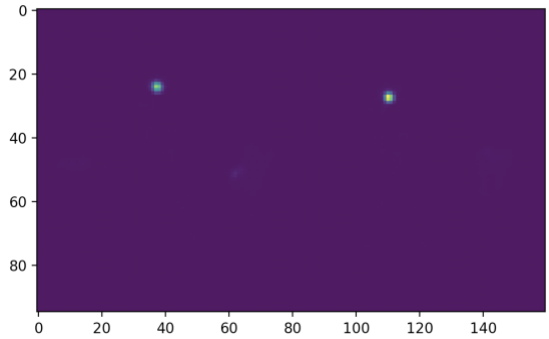

그림 4. 2명 PAF

그림 3은 Heatmap[0]를 그림으로 표현한 것이다. 40x40 크기의 데이터 들이 들어 있는데 노랗게 표현된 곳이 Parts 중의 0번(그림 2의 Parts 중 얼굴의 중심을 나타내는 0번)에 해당하는 위치를 의미한다. 다시 말하면 Heatmap[0]에서 max 값의 위치를 찾으면 (노란 영역의 중심) 그 곳이 Part 0의 위치가 되는 것이다.

이렇게 19개 Heatmap에 대해 max 값을 찾으면 Parts의 0~18까지의 위치를 알 수 있게 된다. 이미 정의된 대로 각 pair를 연결해주면 그림 2의 Parts와 같은 그림이 만들어지는 것이다. 즉 Heatmap[0]의 max와 Heatmap[1]의 max를 연결하면 얼굴 중심점에서 양쪽 어깨 중간 점이 연결되는 것이다.

문제는 한 이미지에 여러 사람이 존재하는 경우다. 2명이 있는 이미지라면 Heatmap[0]의 40x40 데이터에는 각각의 얼굴 중심에 해당하는 값이 2개가 존재하게 된다. 이 때는 단순히 max 값을 찾으면 한 명의 데이터는 사라지게 되므로 max 값을 사용하면 안되고 NMS(Non-Maximum Suppression)이라는 방법을 사용해 peak 들을 모두 남긴다. NMS는 peak에 해당하는 것을 모두 남기는 기능이라 생각하면 되고 이미지에 여러 명의 사람이 있는 경우 Heatmap에 Peak 여러 개 나오므로 이 모두를 남기면서 각 Peak의 가장 높은 곳 1점만을 남겨주는 기능이다.

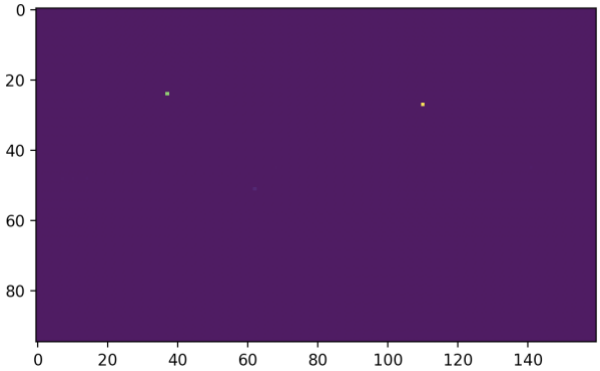

그림 5. 두 명 Heatmap의 NMS 결과

그림 4가 2명의 사람이 있는 이미지의 Heatmap[0]를 그린 것이다. 각각의 얼굴 중심에 해당하는 Peak가 보인다. 그림 5는 그림 4의 NMS를 거친 후 결과로 Peak 주변으로 퍼져있는 데이터는 사라지고 Peak의 max만 남기고 Peak는 모두 살린 결과를 볼 수 있다.

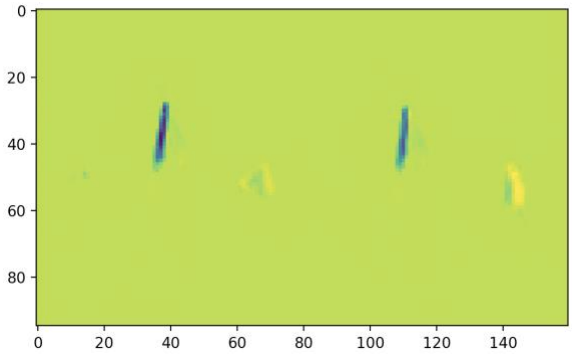

그림 6. PAF[0]

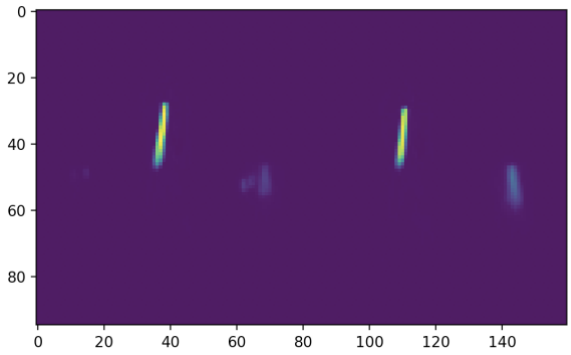

그림 7. PAF[1]

Heatmap으로부터 여러 사람의 Part 정보를 추출해 냈다고 하더라도 추출해 낸 Part 들이 여러 개이므로 어떤 Part와 어떤 Part를 연결해야 한 사람에 해당하는 Part 끼리 연결한 것일까? 예를 들어 그림 4에는 2명의 Heatmap[0] 정보가 있는데 편의상 좌우에 한 명씩 있다고 가정하자. 다음 Heatmap[1]에서 데이터를 얻었는데 우리는 좌우에 한 명씩 사람이 있다는 것을 알고 있으므로 Heatmap[0]의 좌측 Part와 Heatmap[1]의 좌측 Part를 연결하면 된다는 것을 알 수 있다. 그러나 code에서는 이런 것을 어떻게 판단할 수 있을까? 이미지 내에 사람이 많은 경우거나 이미지 상에서 사람이 겹쳐져 있는 경우라면 어려움이 꽃핀다.

이런 경우를 위해 PAF 정보가 OpenPose 모델로부터 prediction 후에 제공 되는데 PAF는 Part Affinity Field로 Pair의 위치와 방향에 대한 정보를 제공해 준다. 2개의 Part Pair에 대해 어떻게 연결되고 어디쯤 위치하는지에 대한 정보를 제공한다고 보면 된다. PAF는 x 방향, y 방향에 대해 정보를 제공하고 Pair(0, 1)에 대해 PAF[0]가 x 방향, PAF[1]이 y 방향에 대한 정보를 제공한다.

그림 6에서 확인 할 수 있듯이 진하게 표시된 긴축 2개가 왼쪽 오른쪽 각각의 사람에 해당하고

그 양 끝이 Part를 의미하는 것으로 목 아래에서 오른쪽 고관절을 잇는 (0, 1) Pair를 의미한다. 그림 7은 같은 정보에 대한 y 방향 PAF를 의미한다. 이 두 정보를 Heatmap과 조합해 어느 Part가 어느 Part와 연결되는 것인지를 알 수 있고 관련 있는 연결끼리 모아 놓으면 한 사람에 해당하는 Part, Pair 정보를 얻을 수 있다. 최종적으로 각 사람에 대한 정보를 바탕으로 뼈대를 그려주면 사람이 여러 명이든 겹쳐 있든 상관 없이 뼈대를 그릴 수 있게 되는 것이다.

PAF를 사용해 어느 Part끼리 연결할 것인지를 정리하는 것은 Line Integral 이라는 기술을 사용한다. Youtube의 알기 쉬운 예제는 https://www.youtube.com/watch?v=uXjQ8yc9Pdg을 보면 되고 간략하게 말하면 두 점 사이의 주어진 line을 따라 integration을 한다는 것인데 예를 들어 여기에서는 Heatmap[0]의 NMS 결과에 존재하는 Peak 들과 Heatmap[1]의 NMS 결과에 존재하는 Peak 들과의 조합(각각 2개씩 Peak가 존재 한다면 총 4가지의 조합)에 대해 두 Peak를 직선으로 연결했다고 가정하고 그 직선을 따라 PAF를 사용해 값을 계산한다.
그림 7에서 알 수 있듯이 왼쪽 편의 두 Peak 조합에 대해 계산된 Line Integral 값은 크지만 상관없어 보이는 왼쪽과 오른쪽 Peak Pair에 대해서는 왼쪽과 오른쪽으로 이어지는 PAF 값이 없으므로 Line Integral의 결과가 작거나 0이 된다.
이런 식으로 모든 조합에 대해 Line Integral을 구하고 일정 기준 이상을 만족하는 Pair만을 남기면 실제 이어져야 하는 Pair를 얻게 된다.

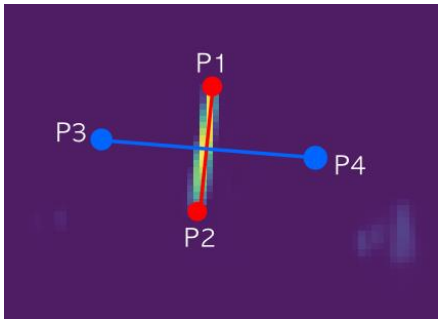

그림 8. PAF Line Integral 예

그림 8은 그림 7의 왼쪽 부분을 잘라낸 것으로 연두색으로 표시된 부분이 PAF의 값이 큰 것을 의미하고 바탕의 보라색은 값이 0이거나 아주 작은 것을 의미한다.
Line Integral은 간략하게 설명하면 그림 8에서와 같이 4개의 Heatmap Peak가 검출되었을 경우 4개의 점에 대해 각각 연결을 한다고 생각하고 그 Line에 대해 PAF 값을 누적해 그 값을 비교하는 것이다. 즉, P1-P2, P1-P3, P1-P4, P2-P3, P2-P4, P3-P4의 6가지 조합이 존재하고 각각의 조합 Peak를 연결하는 Line에 대해 PAF 값을 누적 계산하는 것이다.
그림 8에서는 두 가지 경우만 표시 했는데 P1-P2 Line에 대해 PAF 값을 더하면 Line을 따라 PAF 값이 계속 존재하기 때문에 누적 결과는 꽤 큰 값이 될 것이다. 반면에 P3-P4 Line에 대해서는 대부분 0이다가 중간의 P1-P2와 겹치는 부분만 PAF 값이 있으므로 결과는 꽤 작은 값이 될 것이다. 이렇게 모든 조합에 대해 Line Integral을 수행해 보면 결국 P1-P2가 가장 크고 나머지는 작으므로 일정 기준 이상을 만족하는 P1-P2만을 남기고 나머지 쌍에 대해서는 무시한다.
다시 말하면 PAF를 사용해 Line Integral을 거치고 나면 실제로 연결되어야 하는 Heatmap Peak가

정리 되는 것이다. 하나의 이미지에 사람이 아무리 많다고 해도 PAF 정보를 사용하면 연결 되어야 하는 Peak 들을 정리할 수 있게 된다.

이렇게 얻어진 Pair는 아직은 사람 별로 묶이진 않았기 때문에 연결점 정보를 사용해 사람 별로 묶어 주면 최종적으로 이미지에서 사람 별로 뼈대를 그릴 수 있는 정보를 얻게 된다.

지금까지의 절차를 도식화 하면 그림 9과 같다.

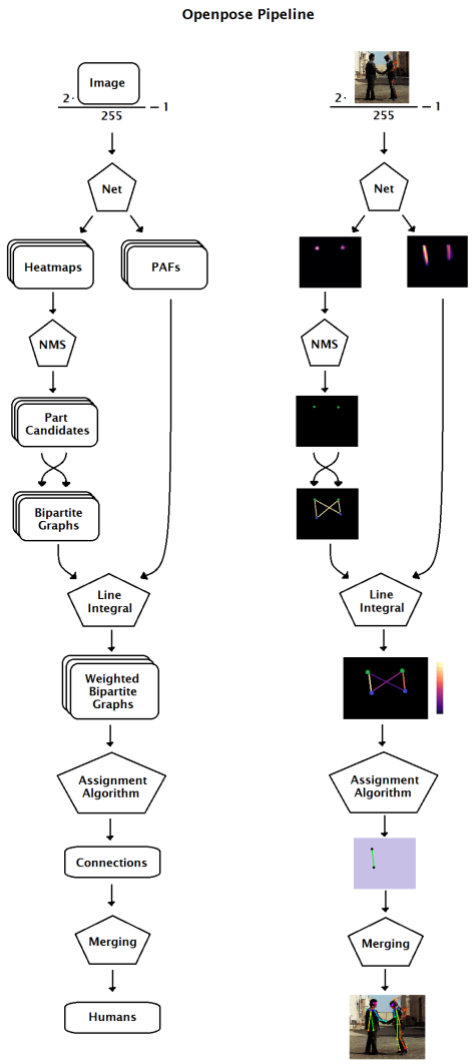

그림 9. OpenPose 절차

2. OpenPose 테스트

2.1. OpenPose 테스트 #1

우선 가장 기초적인 동작만 이해하는 수준에서의 테스트 코드를 작성했다.

코드 2는 이미지를 읽어 OpenPose Model로부터 prediction을 실행해 Heatmap, PAF를 얻지만 Heatmap 만을 사용해 뼈대를 그리는 일을 한다.

```
import numpy as np
import cv2
from PIL import Image
from keras.models import load_model

def makePairs(hm):
    pairs = np.zeros((17, 2, 2), dtype=np.int)

    pairs[0] = np.array([hm[0], hm[1]])

    pairs[1] = np.array([hm[1], hm[2]])
    pairs[2] = np.array([hm[2], hm[3]])
    pairs[3] = np.array([hm[3], hm[4]])

    pairs[4] = np.array([hm[1], hm[5]])
    pairs[5] = np.array([hm[5], hm[6]])
    pairs[6] = np.array([hm[6], hm[7]])

    pairs[7] = np.array([hm[1], hm[8]])
    pairs[8] = np.array([hm[8], hm[9]])
    pairs[9] = np.array([hm[9], hm[10]])

    pairs[10] = np.array([hm[1], hm[11]])
    pairs[11] = np.array([hm[11], hm[12]])
    pairs[12] = np.array([hm[12], hm[13]])

    pairs[13] = np.array([hm[0], hm[14]])
    pairs[14] = np.array([hm[14], hm[16]])

    pairs[15] = np.array([hm[0], hm[15]])
    pairs[16] = np.array([hm[15], hm[17]])

    return pairs

def nsm(hm):
    # non-maximum suppression으로 peak만 남긴다. 사람이 여럿인 경우 peak가 여러 개가 된다.
    hm = hm * (hm == maximum_filter(hm, footprint=np.ones((5, 5))))
    return hm

def test_keras_openpose():
    # load input image
    # W=width, H=height
    sourceMat = cv2.imread('test5.jpg')

    # load model
```

```
    model = load_model('keras_openpose_trained_model.hd5')

    # get paf, heatmap
    # pred[0]=paf, pred[1]=heatmap
    pred = model.predict(np.array([sourceMat]))

    #~첫 dimension을 제거
    # heatmap.shape=(1, int(H/8), int(W/8), 19)
    heatmap = np.squeeze(pred[1])

    # display heatmap
    grayMat = cv2.cvtColor(sourceMat, cv2.COLOR_BGR2GRAY)
    grayMat = cv2.cvtColor(grayMat, cv2.COLOR_GRAY2BGR)
    maxHeatmap = np.zeros((heatmap.shape[2] - 1, 2))

    # heatmap에 점을 그려준다. 18번째까지만 그린다.
    for i in range(heatmap.shape[2] - 1):
        hm = heatmap[:, :, i]

        # heatmap은 입력 image를 w,h 각각 1/8한 크기이다
        hm = cv2.resize(hm, (grayMat.shape[1], grayMat.shape[0]), interpolation=cv2.INTER_CUBIC)
        # find maximum position
        y, x = np.unravel_index(np.argmax(hm), hm.shape)
        maxHeatmap[i] = np.array([y, x])

        if i == 0:
            c = (0, 0, 255)
        elif i < (heatmap.shape[2] - 1):
            c = (0, 255, 0)
        else:
            c = (255, 0, 0)
        grayMat = cv2.circle(grayMat, (x, y), 2, c, thickness=2)

    # pair 간에 선을 이어준다
    pairs = makePairs(maxHeatmap)
    for i in range(pairs.shape[0]):
        p1 = (pairs[i, 0, 1], pairs[i, 0, 0])
        p2 = (pairs[i, 1, 1], pairs[i, 1, 0])
        grayMat = cv2.line(grayMat, p1, p2, (0, 255, 0), thickness=1)

    cv2.imshow('heatmap', grayMat)
    cv2.waitKey()

if __name__ == '__main__':
    test_keras_openpose()
```

코드 2. OpenPose Test #1

load_model()로 읽어지는 모델은 사전에 다운로드 받아 두었던 OpenPose의 Keras version으로

training이 되어 있는 것이다.
이미지를 읽어 prediction()을 하면 출력되는 결과물은 pred[0]가 PAF, pref[1]이 Heatmap을 의미한다. 우선 PAF는 제외하고 Heatmap만 사용해 뼈대를 그려 보았다.

19개의 Heatmap에서 실제 사용되는 것은 18개이므로 18번 for loop를 돌면서 각 Heatmap에서 max 값을 가진 위치를 찾아 그 위치에 동그라미를 그려준다. 2D array에서 max 값의 row, column index를 찾는 것은 68번째 줄의 unravel_index() 함수이다. Heatmap은 입력 이미지를 8x8 크기의 영역으로 나눈 것이므로 width, height가 모두 1/8된 크기의 Array로 표현된다. 예를 들어 max 값이 [1, 1]에 있다면 실제로는 row, column 모두 8~15 사이에 있다는 소리가 된다. 이를 쉽게 하기 위해 Heatmap을 원본 이미지 크기로 resize하고 그 중에서 max 값을 찾으면 바로 그 위치가 실제 이미지에서의 위치가 되므로 편하게 하기 위해 OpenCV의 resize 함수를 사용했다.

구해진 Part 좌표에 동그라미를 그린 후 각 Part의 미리 정해진 연결에 대해 선을 그려 뼈대를 만들어준다. (그림 1)

코드 2가 가능한 이유는 이미지 내에 사람이 1명만 존재하기 때문이고 여러 명이 존재하는 경우 이 코드를 사용하면 제대로 된 결과를 얻지 못한다.

2.2. OpenPose 테트스 #2

여러 사람이 존재하는 이미지에 대해서도 처리를 해보자. 코드는 test_keras_openpose_2.py의 estimate_pose() 함수를 보면 된다.

```
def estimate_pose(heatMat, pafMat):
    # paf.shape=(38, int(H/8), int(W/8))
    # heatmap.shape=(19, int(H/8), int(W/8))
    paf = np.rollaxis(np.squeeze(pafMat), 2, 0)
    heatmap = np.rollaxis(np.squeeze(heatMat), 2, 0)

    # # reliability issue.
    # heatmap = heatmap - heatmap.min(axis=1).min(axis=1).reshape(19, 1, 1)
    # heatmap = heatmap - heatMat.min(axis=2).reshape(19, heatmap.shape[1], 1)

    # get peak points from heatmap by NSM
    coords = []
    for hm in heatmap[:-1]:
        # non-maximum suppression
        hm = nsm(hm)
        # nsm 결과에서 찾은 peak 들의 row, col 좌표를 얻는다
        # [[x1, x2, ...], [y1, y2, ...]]과 같은 결과가 append됨.
        # coords는 (18, 2, ?)의 shape을 갖는다.
        # coords[:, :, 0]는 row 값으로 y를 의미하고, coords[:, :, 1]는 col 값으로 x를 의미함
        coords.append(np.where(hm > 0))
```

```
# 정의해 놓은 connection pair 만큼 반복하면서 모든 connection을 구해 connection_all에 추가 한다.
connection_all = []
for (idx1, idx2), (paf_x_idx, paf_y_idx) in zip(CocoPairs, CocoPairsNetwork):
    # 현재 pair에 해당하는 모든 connection을 찾는다.
    connection = estimate_pose_pair(coords, idx1, idx2, paf[paf_x_idx], paf[paf_y_idx])
    # connection_all에 추가 (append가 아님)
    connection_all.extend(connection)

# 한 사람, 한 사람에 속하는 connection을 모아 준다.
conns_by_human = dict()
for idx, c in enumerate(connection_all):
    conns_by_human['human_%d' % idx] = [c] # 처음에 모든 connection은 다른 사람으로 취급됨

# defaultdict()는 키값에 기본값을 정의하고 키값이 없어도 에러를 출력하지 않고 기본 값을 출력한다.
no_merge_cache = defaultdict(list)
empty_set = set()
while True:
    is_merged = False
    # .combinations()는 주어진 iterable에서 주어진 개수 만큼의 combination을 만들어 준다.
    # ('ABCD', 2)는 AB AC AD BC BD CD를 만들어 낸다.
    # .keys()는 dictionary에서 key 값들만 얻어 낸다. 여기서는 'human_0'와 같은 값을 만들어 준다.
    for h1, h2 in itertools.combinations(conns_by_human.keys(), 2):
        # h1과 h2가 같으면 무시
        if h1 == h2:
            continue
        # h2가 no_merge_cache에 있으면 무시
        if h2 in no_merge_cache[h1]:
            continue
        # c1, c2는 estimate_pose_pair에서 추가된 dict로
        # uPartIdx가 connection pair 정보를 담고 있는데
        # 예를 들어 왼쪽 어깨 연결인 (1, 2)는 오른쪽 어깨 연결인 (1, 5)와 1이 서로 같다.
        # set() & set()에서 이 연결 pair중 같은 값이 있는지를 검사해
        # 같은 값이 있으면 연결되는 것으로 보고 연결점 밑으로 옮겨 주는 것이다.
        for c1, c2 in itertools.product(conns_by_human[h1], conns_by_human[h2]):
            if set(c1['uPartIdx']) & set(c2['uPartIdx']) != empty_set:
                # 연결 점이 있다
                is_merged = True
                # h1 밑으로 h2를 넣어 준다
                conns_by_human[h1].extend(conns_by_human[h2])
                # h2는 원래 것에서 제거
                conns_by_human.pop(h2)
                break
        if is_merged:
            # 연결된 것이 있으면 연결되지 않은 것을 나타내는 no_merge_cache에서 제거
            no_merge_cache.pop(h1, None)
            break
        else:
            # 연결된 것이 없으면 no_merge_cache에 추가
```

```
                    no_merge_cache[h1].append(h2)

            if not is_merged:
                break

    # 연결점이 4개 이상(적어도 2개의 연결이 있음을 의미), score가 0.8 이상이 포함된 것에 대해서만 남긴다
    conns_by_human = {h: conns for (h, conns) in conns_by_human.items() if len(conns) >= Min_Subset_Cnt}
    conns_by_human = {h: conns for (h, conns) in conns_by_human.items() if max([conn['score'] for conn in conns]) >= Min_Subset_Score}

    humans = [human_conns_to_human_parts(human_conns, heatmap) for human_conns in conns_by_human.values()]
    return humans
```

코드 3. estimate_pose() 함수

2.2.1. NMS

여러 사람이 존재하는 이미지를 처리하기 위해서는 Heatmap에서 Peak 들만 남겨야 하는데 NMS가 그 역할을 해준다. NMS 알고리즘은 다음과 같이 동작한다.

- Heatmap의 첫 번째 pixel로부터 시작
- 5x5와 같은 window로 pixel을 둘러 싸고 이 window 내에서 max 값을 찾는다
- 중심 pixel의 값을 해당 max 값으로 바꿔준다
- window를 한 pixel 움직여 반복해 전체 Heatmap에 대해 진행 한다
- 원래 Heatmap과 비교해 같은 값을 가진 pixel이 원하는 peak이므로 나머지 pixel을 0으로 만든다

Python에서는 코드 4과 같은 코드로 NMS를 구현한다.

```
def nsm(hm):
    hm = hm * (hm == maximum_filter(hm, footprint=np.ones((5, 5))))
    return hm
```

코드 4. NMS 함수

5x5 window로 maximum₩_filter를 거친 후 원래 array와 비교하면 원래 값과 같은 위치는 1(True), 나머지는 0(False)가 되므로 원래 array와 곱하면 Peak만 남고 나머지는 0이 된다(이 코드는 Python이 아닌 다른 환경에서는 알맞게 구현해줘야 하는데 고민이 좀 필요해 보인다).

NMS의 결과는 np.where(hm > 0)이라는 코드를 사용해 Heatmap의 값이 0이 아닌 값의 row(y 좌표), column(x 좌표) 값을 얻어 배열에 따로 저장한다.

2.2.2. Pair 구하기

estimate_pose_pair() 함수를 사용해 미리 정의 되어 있는 Pair에 대한 연결을 찾는다. estimate_pose_pair() 함수는 코드 5과 같다.

```
def estimate_pose_pair(coords, partIdx1, partIdx2, pafX, pafY):
    connection_temp = []
    # heatmap에서 pair인 두 index에 해당하는 nms 결과 peak 들을 추출
    peak_coord1, peak_coord2 = coords[partIdx1], coords[partIdx2]

    # heatmap index가 2개 전달되는데 각 heatmap의
    # heatmap pair 중 첫번째 것에 대해 peak 개수 만큼 for loop를 돈다
    for idx1, (y1, x1) in enumerate(zip(peak_coord1[0], peak_coord1[1])):
        # heatmap pair 중 두번째 것에 대해 peak 개수 만큼 for loop를 돈다
        for idx2, (y2, x2) in enumerate(zip(peak_coord2[0], peak_coord2[1])):
            # 두 점 사이의 line integral 값을 얻는다.
            # 두 점을 직선으로 연결하고 그 사이의 paf 값을 더한다. (여기에서는 10개로 나누어 계산)
            # 연결된 점이라면 score나 count의 값이 크게 된다.
            score, count = get_score(x1, y1, x2, y2, pafX, pafY)

            if (partIdx1, partIdx2) in [(2, 3), (3, 4), (5, 6), (6, 7)]: # arms
                # 팔의 경우 count가 3개 이하거나 score가 0이하이면 무시
                if count < InterMinAbove_Threshold // 2 or score <= 0.0:
                    continue
            elif count < InterMinAbove_Threshold or score <= 0.0:
                # 팔 이외의 경우는 count가 6이하거나 score가 0이하면 무시
                continue

            # 두 점 사이의 연결이 인정되는 것들은 추가
            connection_temp.append({
                'score': score,
                'coord_p1': (x1, y1),
                'coord_p2': (x2, y2),
                'idx': (idx1, idx2),
                'partIdx': (partIdx1, partIdx2),
                'uPartIdx': ('{}-{}-{}'.format(x1, y1, partIdx1), '{}-{}-{}'.format(x2, y2, partIdx2))
            })

    # 구해진 connection 들 중에서 실제 connection 만을 남긴다
    # 남기는 기준은 score가 높은 것부터 남기고 낮은 것은 없앤다.
    connection = []
    used_idx1, used_idx2 = [], []
    # 위에서 구한 connection을 score가 높은 것부터 sorting해 실제 connection을 구한다.
    for conn_candidate in sorted(connection_temp, key=lambda x: x['score'], reverse=True):
        # 이미 사용된 connection이면 무시
        if conn_candidate['idx'][0] in used_idx1 or conn_candidate['idx'][1] in used_idx2:
            continue
        # 사용되지 않은 connection은 등록
        connection.append(conn_candidate)
        used_idx1.append(conn_candidate['idx'][0])
```

```
            used_idx2.append(conn_candidate['idx'][1])

    return connection
```

코드 5. estimate_pose_pair() 함수

estimate_pose_pair() 함수를 호출하는 부분을 먼저 이해 해야하는데, Heatmap에서 찾은 Peak의 x, y 좌표 값인 coords와 미리 정의되어 있는 Heatmap 내의 연결 pair, PAF의 연결 pair index를 넘겨준다. CocoPairs의 첫 값은 (1, 2)이고 CocoPairsNetwork의 첫 값은 (12, 13)이다. 이 값 들은 그림 2에서 Parts, Pairs의 오른쪽 어깨 연결을 의미하는 것이다.

```
# 정의해 놓은 connection pair 만큼 반복하면서 모든 connection 을 구해 connection₩_all 에 추가 한다.
connection_all = []
for (idx1, idx2), (paf_x_idx, paf_y_idx) in zip(CocoPairs, CocoPairsNetwork):
    # 현재 pair에 해당하는 모든 connection을 찾는다.
    connection = estimate_pose_pair(coords, idx1, idx2, paf[paf_x_idx], paf[paf_y_idx])
    # connection₩_all에 추가 (append가 아님)
    connection_all.extend(connection)
```

estimate_pose_pair() 함수로는 Heatmap에서 찾은 Peak 들의 x, y 좌표 값 들과 찾으려고 하는 connection에 대한 index 정보 그리고 paf 값이 전달 되는 것이다.
peak_coord1 = coords[partIdx1]인데 coords가 (18, 2, ?)의 shape이므로 peak_coord1은 (2, ?)의 shape을 갖는다. 이 것은 Heatmap에서 NSM으로 찾은 Peak들의 x 좌표를 의미한다. estimate_pose_pair() 함수는 Part pair가 (1, 2)라면 Part 1의 Heatmap에서 찾은 Peak 들 모두와 Part 2의 Heatmap에서 찾은 모든 Peak 들의 조합에 대해 서로 간의 연결을 가정해 그 연결의 점수와 연결 개수를 구한다. 이렇게 구해진 점수나 연결 개수에 따라 어느 정도 이상인 경우만 남긴다. 서로 관련이 없는 Peak 들 사이의 연결은 PAF 값이 0이거나 아주 작은 값이므로 최종 점수나 연결 개수가 작게 되어 무시하게 된다.

각 Peak 간의 점수나 연결 개수를 구하는 것은 코드~₩ref{code:get_score_function}과 같고 PAF를 사용한 line integral을 사용하는데 두 점 사이를 10등분하고 각각의 line integral 값을 더한 것이 점수가 되고 10등분 중에 점수가 0.1 이상인 것의 개수를 만들어낸다.

$$\int_{y_1}^{y_2}\int_{x_1}^{x_2}\begin{bmatrix}PAF_x(x,y)\\PAF_y(x,y)\end{bmatrix}\cdot\begin{bmatrix}v_x\\v_y\end{bmatrix}dxdy \qquad (1)$$

수식(1)에서와 같이 Line Integral은 두 점 사이의 벡터 v_x, v_y에 PAF_x, PAF_y를 곱한 값을 더한 것이다. 수식에서는 연속이지만 그렇게 계산할 수는 없으므로 실제 코드에서는 두 점을 연결한 직선을 10등분해서 각각에 대해 계산해 더하는 방식을 취했다.

```
# score 는 line integral 값으로 얻는데, 두 점 사이의 paf 와 vector 를 곱해 얻는다.
# paf가 두 점 사이의 흐름?의 정도를 나타내는데
# 이 흐름의 정도가 크다는 소리는 연결되어 있을 가능성이 높다는 소리다.
# https://cdn-images-1.medium.com/max/1600/1*Ce7iDD5ac7i26YXqoLQPyw.png 의 수식을 참조할 것
def get_score(x1, y1, x2, y2, pafMaxX, pafMaxY):
    num_inter = 10
    dx, dy = x2 - x1, y2 - y1
    normVec = math.sqrt(dx ** 2 + dy ** 2) # 두점 사이의 vector

    if normVec < 1e-4: # 같은 점이거나 거리가 가까우면 score는 0이다
        return 0.0, 0

    vx, vy = dx / normVec, dy / normVec

    # x1과 x2가 다르면 x1에서 <x2까지 dx/num_inter 간격으로 num_inter만큼의 숫자를 만든다.
    # 같으면 x1으로 num_inter만큼 채운다.
    xs = np.arange(x1, x2, dx / num_inter) if x1 != x2 else np.full((num_inter, ), x1)
    ys = np.arange(y1, y2, dy / num_inter) if y1 != y2 else np.full((num_inter, ), y1)
    # xs의 값 들을 반올림 하고 np.int8 타입으로 변환한다.
    # x1, x2 사이의 간격이 pair의 index 차이이고 heatmap은 19까지,
    # paf는 38까지 이므로 int8 타입으로 다 커버 되므로 int8 타입으로 만든 것이다.
    xs = (xs + 0.5).astype(np.int8)
    ys = (ys + 0.5).astype(np.int8)

    # without vectorization
    pafXs = np.zeros(num_inter)
    pafYs = np.zeros(num_inter)
    for idx, (mx, my) in   enumerate(zip(xs, ys)):
        # pafMaxX, pafMaxY는 전체 paf의 shape인 38xInt(H/8)xInt(W/8) 중에
        # 첫 index 38의 값에 대해 xindx, yidx를 적용해
        # Int(H/8)xInt(W/8) 크기의 paf 값이 들어있는 array가 넘어 온다.
        pafXs[idx] = pafMaxX[my][mx]
        pafYs[idx] = pafMaxY[my][mx]

    # 수식에 따라 pafX, pafY와 vx, vy를 곱한 값을 구한다.
    local_scores = pafXs * vx + pafYs * vy
    # 곱한 값이 0.1이상인 경우만 찾고 그 index의 값은 0(False)또는 1(True)가 된다.
    thidxs = local_scores > Inter_Threashold

    # return의 첫번째 값은 local_score가 0.1이상인 값 들만 모두 합한 값이고
    # 두번째 값은 local_score가 0.1이상인 개수를 나타낸다.
    return sum(local_scores * thidxs), sum(thidxs)
```

코드 6. get_score() 함수

즉 두 점이 연결되어야 하는 것이라면 PAF 값이 클 것이기 때문에 최종 점수가 높고 10등분한 것에 대해 상당수가 PAF 값을 가지고 있을 것이다. 만약 다른 연결이 지금 계산 중이 연결과 겹쳐져 있다면 어느 정도 PAF 값이 있을 것이고 10등분에 대해서도 PAF 값을 가진 것이 있겠지만 그 점수가 진짜 연결의 값보다는 작고 개수도 적을 것이다. 이런 점을 사용해 실제 연결인지 아

닌지를 찾아내는 것이다.

2.2.3. 사람에 따라 분류

연결되어야 하는 Pair가 다 정리 되면 이제 그 Pair 들을 각각의 사람에 따라 분류를 하는 작업이 필요하다 (개인적인 생각으로는 화면에 그리기만 하는 것이라면 구지 구분할 필요는 없어 보인다).

연결을 분류하는 방법은 2.2.2에서 Pair를 구할 때 마지막으로 Pair의 정보를 저장할 때 uPartIdx라는 정보를 저장하는데 (x1, y1, partIdx1)과 (x2, y2, partIdx2)를 저장한다. 즉, 두 점의 위치와 Part 번호를 저장한다. AB라는 두 점 사이의 직선과 BC라는 두 점 사이의 직선은 B라는 점에서 이어지고 두 직선이 사이에 연결되는 점이 존재한다면 이 두 직선은 이어져 있다는 얘기이므로 이 정보를 사용해 (x, y)가 연결되는 Pair 들을 모아 놓으면 그게 한 사람에 해당하는 연결이 되는 것이다.

3. 동영상 테스트

OpenPose의 개인적인 최종 목표는 동영상에 real-time으로 동작시키는 것이다. 그래서 우선 동영상에서 얼마나 빨리 도는지 확인하고 Pose를 얼마나 잘 찾아내는지 확인을 했다.

```
from optparse import OptionParser
import os
import cv2
from keras.models import load_model
from test_keras_openpose_2 import estimate_pose, draw_humans
import numpy as np
import time
import sys

# 키보드 입력 기다리는 함수. q가 눌리면 중단한다.
def wk(waitTime=1, char='q'):
    c = cv2.waitKey(waitTime) & 0xFF
    if c == ord(char):
        return True
    else:
        return False

def test_keras_openpose_3(inputFileName, outputFileName):
    # load model
    model = load_model('keras_openpose_trained_model.hd5')

    # 동영상 파일 열기
    videoCapture = cv2.VideoCapture(os.path.abspath(os.path.expanduser(inputFileName)))
    if not videoCapture.isOpened():
        print('Input file not opened.')
```

```
        return

    frameCount = 0

    # 동영상에서 정보를 뽑아낸다
    width = int(videoCapture.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(videoCapture.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = int(videoCapture.get(cv2.CAP_PROP_FPS) + 0.5)
    total_frames = int(videoCapture.get(cv2.CAP_PROP_FRAME_COUNT))
    print('Input : {0}x{1}, {2} fps, {3} frames'.format(width, height, fps, total_frames))

    # 저장 codec은 mp4
    fourcc = cv2.VideoWriter_fourcc(*'MP4V')
    videoWriter = cv2.VideoWriter(os.path.abspath(os.path.expanduser(outputFileName)), fourcc, fps, (width, height))
    if not videoWriter.isOpened():
        print('Output file not opened.')
        videoCapture.release()
        return

    while True:
        success, image = videoCapture.read()

        # 모든 frame을 다 읽을 때까지 반복
        if not success:
            break

        print('frame={}/{}'.format(frameCount, total_frames), end='')
        # 다음 frame 읽을 때까지 쉬지 않고 처리를 하므로 화면 갱신하는 시간이 따로 없어 강제로 갱신한다
        sys.stdout.flush()

        # 이미지 크기는 656x368로 줄여 입력한다
        img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (656, 368))

        # prediction
        s = time.perf_counter()
        pred = model.predict(np.array([img]))
        e = time.perf_counter()
        print('_{0:.4f}'.format(e - s), end='')
        sys.stdout.flush()

        # pose 찾기
        s = time.perf_counter()
        humans = estimate_pose(pred[1], pred[0])
        e = time.perf_counter()
        print('_{0:.4f}'.format(e - s), end='')
        sys.stdout.flush()

        # pose 그리기
```

```
        s = time.perf_counter()
        drawMat = draw_humans(image, humans)
        e = time.perf_counter()
        print('_{0:.4f}'.format(e - s), end='')
        sys.stdout.flush()

        # 동영상으로 저장
        s = time.perf_counter()
        videoWriter.write(drawMat)
        e = time.perf_counter()
        print('_{0:.4f}'.format(e - s), end='')
        sys.stdout.flush()

        frameCount += 1

        print('')
        sys.stdout.flush()

        # 키 입력을 확인
        cv2.imshow('resutl', drawMat)
        if wk() == True:
            break

videoCapture.release()
videoWriter.release()


if __name__ == '__main__':
        parser = OptionParser()
        parser.add_option("--input", dest="input", default=None, help="Input movie file name")
        parser.add_option("--output", dest="output", default=None, help="Output file name")
        (options, args) = parser.parse_args()

        test_keras_openpose_3(options.input, options.output)
```

코드 7. 동영상 테스트 코드

OpenCV를 사용해 동영상에서 frame을 하나씩 꺼내 OpenPose를 돌리고 그 결과를 다시 동영상으로 저장하는 것이다. OpenPose는 (656, 368) 크기의 이미지에 최적화 된 것으로 보여진다. 그래서 (1280x720) 크기의 이미지를 줄여 OpenPose Model에서 prediction 하도록 했다. OpenPose 결과는 (1280x720) 크기의 원본 이미지에 그리고 그것을 동영상으로 저장했다.

i7, P4000 Quadro, RAM 24GB 시스템에서 prediction은 약 0.3초 정도 걸리나 estimate_pose() 함수가 약 5~6초 정도 걸려 대부분의 시간을 소비하는 것으로 나타났다.

그림 10. 동영상 테스트

용량 때문에 이 문서에 넣지는 못했지만 동영상의 중간에 보면 Pose를 제대로 못 찾는 경우가 발생한다. 그림 11와 같은 경우인데 왼쪽 사람의 경우 오른쪽 다리를 못 찾고 오른쪽 사람의 경우 오른쪽 무릎에서 발목 까지를 제대로 못 찾는다. 팔은 골프 동작 때문에 겹쳐서 하나만 찾은 것이니 에러는 아니다. 이런 경우에 대해서는 우선 Pose를 찾을 때의 threshold 값을 조절해 가면서 실험을 해봐야 할 것 같다.

그리고 아직은 training된 Model의 성능이 굉장히 좋은 것은 아니란 것을 알 수 있다. 아마도 필자가 사용한 Keras Version의 training 성능이 딸리는 것이고 공식 OpenPose 사이트의 Model은 이보다 나은 성능을 가졌을 것이라 추측하고 있다. (이 버전의 사용은 나중에 시도해보기로...)

그림 11. 잘 못 찾은 경우

그리고 동영상에서 모든 경우에 대해 모자를 쓰고 있어 얼굴을 제대로 인식하지 못하기 때문에 얼굴에 대한 인식은 힘든 상황이다.