Bonus Lab – Human Pose Estimation

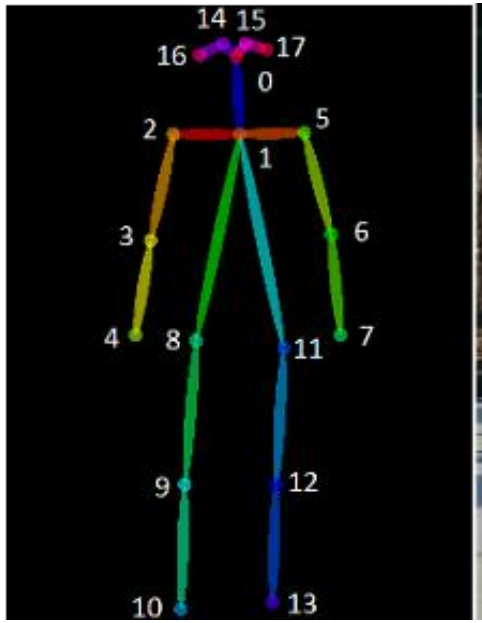
Advisor：Tsai, Chia-Chi

TA：林芷萱

Contact: course.aislab@gmail.com

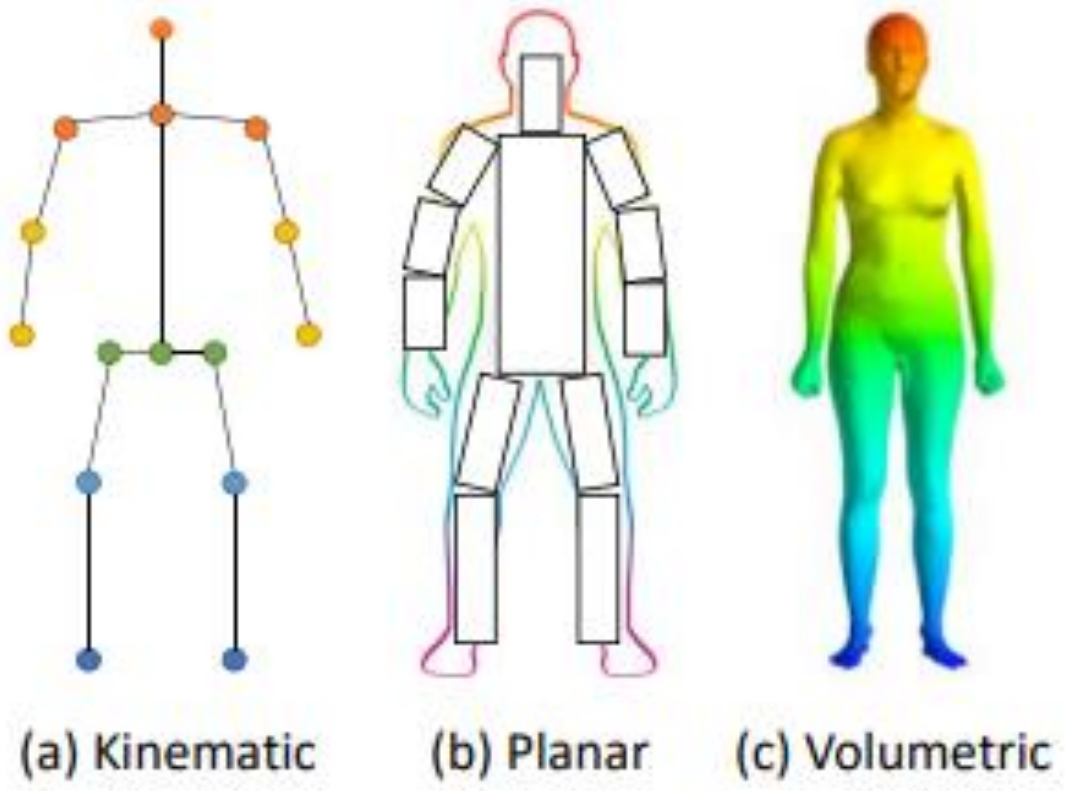
What is Human Pose Estimation?

- Human Pose Estimation (HPE) is a way to capture a set of coordinates for each joint (arm, head, torso, etc.,) which is known as a **keypoint** that can describe a pose of a person. The connection between these points is known as a **pair**.



Human body modeling

- There are three types of approaches to model the human body:



Deep Learning-based approaches to 2D HPE

- Deep learning-based approaches are well defined by their ability to generalize any function.

- When it comes to computer vision tasks, deep convolutional neural networks (CNN) surpass all other algorithms, and this is true in HPE as well.

Deep Learning-based approaches to 2D HPE

- DeepPose: Human Pose Estimation via Deep Neural Networks

Toshev, A., & Szegedy, C. (2014). Deeppose: Human pose estimation via deep neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1653-1660).

- In the paper that they had released, they defined the whole problem as a CNN-based regression problem towards body joints.

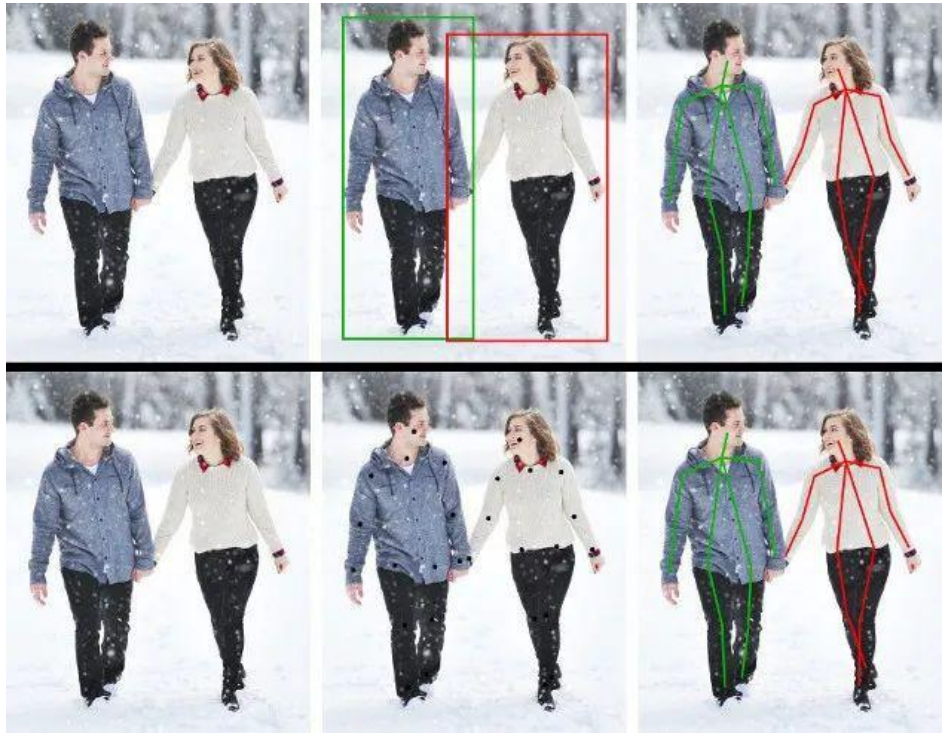
- With strong and promising results shown by DeepPose, the HPE research naturally gravitated towards the deep learning-based approaches.

HPE using Deep Neural Networks

- DNNs are very proficient in estimating single human pose but when it comes to estimating multi-human they struggle because:
  1. An image can contain multiple numbers of people in different positions.
  2. As the number of people increases, the interaction between increases leads to computational complexities.
  3. An increase in computational complexities often leads to an increase in inference time in real-time.

HPE using Deep Neural Networks

- In order to tackle these problems, we introduced two approaches:
  1. Top-down: Localize the humans in the image or video and then estimate the parts followed by calculating the pose.
  2. Bottom-up: Estimate the human body parts in the image followed by calculating the pose.



OpenPose

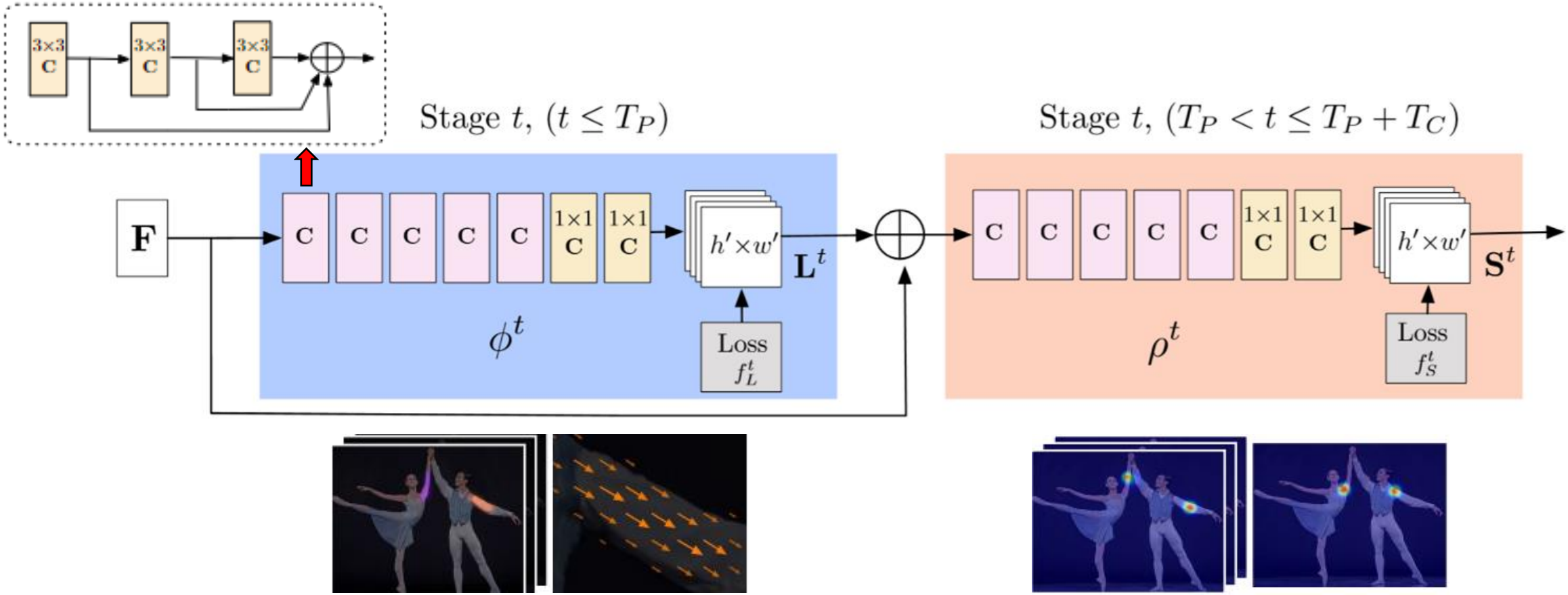
Cao, Z., Simon, T., Wei, S. E., & Sheikh, Y. (2017). Realtime multi-person 2d pose estimation using part affinity fields. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7291-7299).

- An efficient method for multi-person pose estimation.
- Bottom-up approach.
- Part Affinity Fields (PAFs), a set of 2D vector fields that encode the location and orientation of limbs over the image domain.

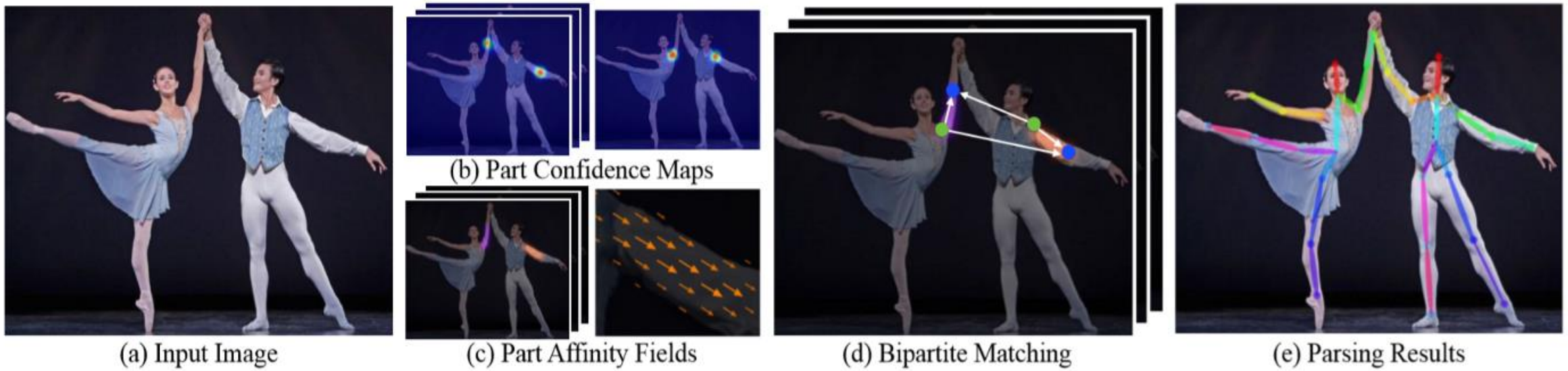
OpenPose

Network architecture

- Iteratively predicts **PAFs** that encode part-to-part association, shown in blue, and detection **Confidence Maps**, shown in beige.



Method



- 2D confidence maps S of body part locations
  - S = (S1,S2,…,SJ) has J confidence maps, one per part.

- 2D vector fields L of part affinity fields (PAFs)
  - L = (L1,L2,…,LC) has C vector fields, one per limb.

Simultaneous detection and association

- First stage input: feature maps F
  - Initialized by the first 10 layers of VGG-19 and fine-tuned.
  - PAFs: $\mathbf{L}^1 = \phi^1(\mathbf{F})$

- Subsequent stage:

$$\mathbf{L}^t = \phi^t(\mathbf{F}, \mathbf{L}^{t-1}), \forall 2 \leq t \leq T_P, \quad (1)$$

- Confidence maps stage:

$$\mathbf{S}^{T_P} = \rho^t(\mathbf{F}, \mathbf{L}^{T_P}), \forall t = T_P, \quad (2)$$

$$\mathbf{S}^t = \rho^t(\mathbf{F}, \mathbf{L}^{T_P}, \mathbf{S}^{t-1}), \forall T_P < t \leq T_P + T_C, \quad (3)$$

Simultaneous detection and association

- Loss function(L2 loss):

$$f_{\mathbf{L}}^{t_i} = \sum_{c=1}^{C} \sum_{\mathbf{p}} \mathbf{W}(\mathbf{p}) \cdot \|\mathbf{L}_c^{t_i}(\mathbf{p}) - \mathbf{L}_c^*(\mathbf{p})\|_2^2, \quad (4)$$

$$f_{\mathbf{S}}^{t_k} = \sum_{j=1}^{J} \sum_{\mathbf{p}} \mathbf{W}(\mathbf{p}) \cdot \|\mathbf{S}_j^{t_k}(\mathbf{p}) - \mathbf{S}_j^*(\mathbf{p})\|_2^2, \quad (5)$$

- The overall objective:

$$f = \sum_{t=1}^{T_P} f_{\mathbf{L}}^t + \sum_{t=T_P+1}^{T_P+T_C} f_{\mathbf{S}}^t. \quad (6)$$

Confidence maps for part detection

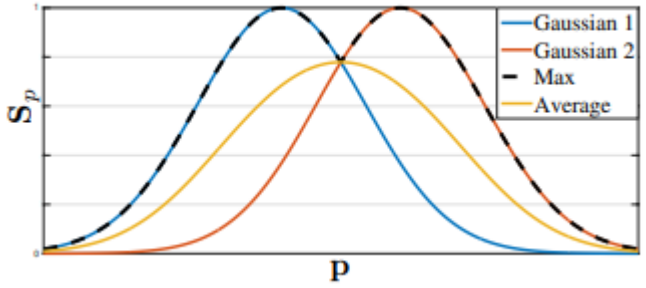
- The value at location p in $\mathbf{S}^*_{j,k}$:

$$\mathbf{S}^*_{j,k}(\mathbf{p}) = \exp\left(-\frac{||\mathbf{p} - \mathbf{x}_{j,k}||_2^2}{\sigma^2}\right), \qquad (7)$$

  - σ: controls the spread of the peak

- The groundtruth confidence map:

$$\mathbf{S}^*_j(\mathbf{p}) = \max_k \mathbf{S}^*_{j,k}(\mathbf{p}). \qquad (8)$$

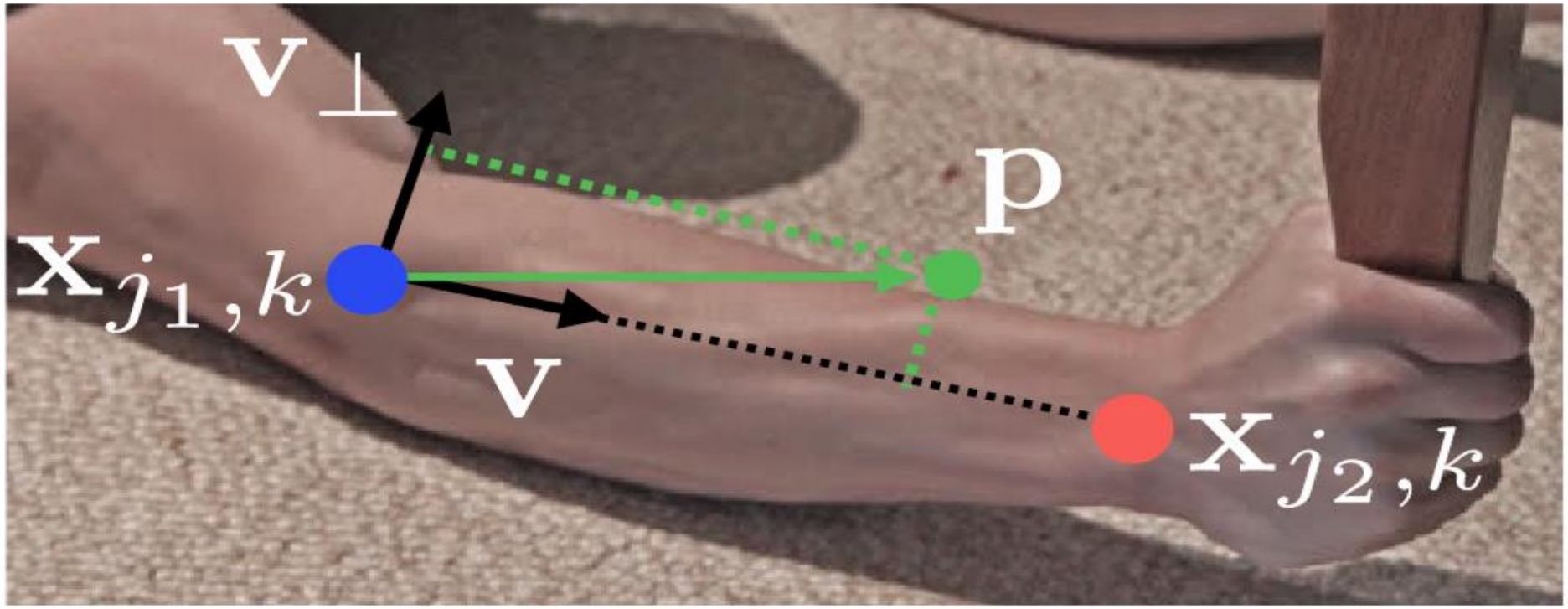


Part affinity fields for part association

(a) (b) (c)

- Two limitations
  - It encodes only the position, and not the orientation, of each limb.
  - It reduces the region of support of a limb to a single point.

Part affinity fields for part association



- Groundtruth PAF at an image point p:

$$\mathbf{L}^*_{c,k}(\mathbf{p}) = \begin{cases} \mathbf{v} & \text{if } \mathbf{p} \text{ on limb } c,k \\ \mathbf{0} & \text{otherwise.} \end{cases} \qquad (9)$$

Part affinity fields for part association

- Groundtruth PAF at an image point p:

$$\mathbf{L}^*_{c,k}(\mathbf{p}) = \begin{cases} \mathbf{v} & \text{if } \mathbf{p} \text{ on limb } c,k \\ \mathbf{0} & \text{otherwise.} \end{cases} \qquad (9)$$

- Points p:

$$0 \leq \mathbf{v} \cdot (\mathbf{p} - \mathbf{x}_{j_1,k}) \leq l_{c,k} \text{ and } |\mathbf{v}_\perp \cdot (\mathbf{p} - \mathbf{x}_{j_1,k})| \leq \sigma_l,$$

- Groundtruth part affinity field averages the affinity fields:

$$\mathbf{L}^*_c(\mathbf{p}) = \frac{1}{n_c(\mathbf{p})} \sum_k \mathbf{L}^*_{c,k}(\mathbf{p}), \qquad (10)$$

Part affinity fields for part association

- Score each candidate limb:

$$E = \int_{u=0}^{u=1} \mathbf{L}_c\left(\mathbf{p}(u)\right) \cdot \frac{\mathbf{d}_{j_2} - \mathbf{d}_{j_1}}{||\mathbf{d}_{j_2} - \mathbf{d}_{j_1}||_2} du, \qquad (11)$$

where **p**(u) interpolates the position of the two body parts:

$$\mathbf{p}(u) = (1-u)\mathbf{d}_{j_1} + u\mathbf{d}_{j_2}. \qquad (12)$$

Multi-person parsing using PAFs

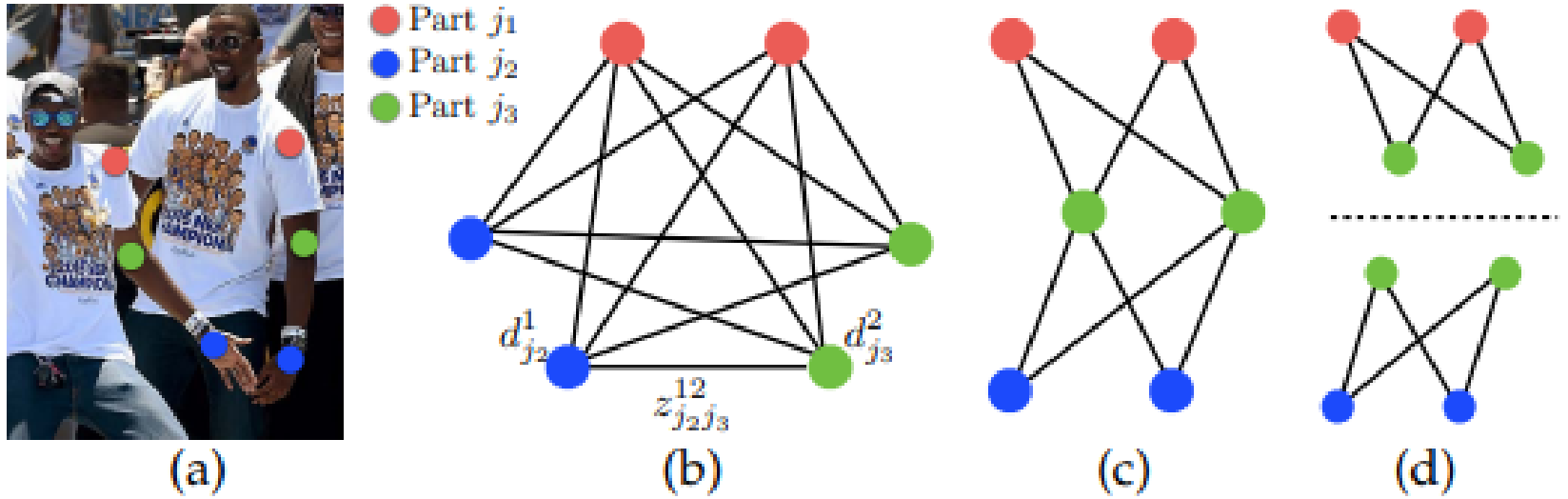


Fig. 6: Graph matching. (a) Original image with part detections. (b) K-partite graph. (c) Tree structure. (d) A set of bipartite graphs.

- These part candidates define a large set of possible limbs.

Multi-person parsing using PAFs

- Determine whether the two keypoints are connected:

$$z_{j_1 j_2}^{mn} \in \{0, 1\}$$

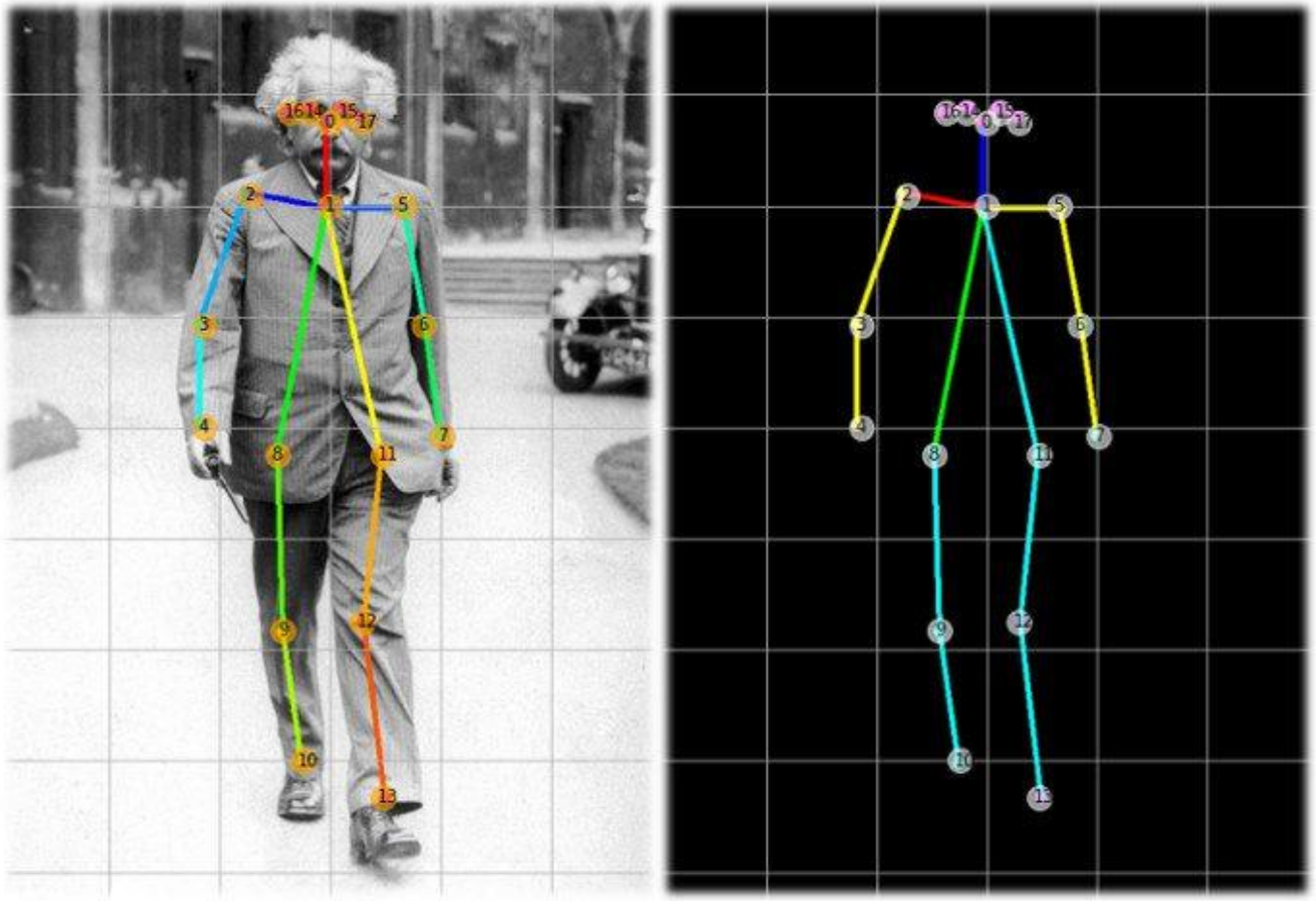
- Final problem we need to solve:

$$\max_{\mathcal{Z}_c} E_c = \max_{\mathcal{Z}_c} \sum_{m \in \mathcal{D}_{j_1}} \sum_{n \in \mathcal{D}_{j_2}} E_{mn} \cdot z_{j_1 j_2}^{mn}, \tag{13}$$

$$\text{s.t.} \quad \forall m \in \mathcal{D}_{j_1}, \sum_{n \in \mathcal{D}_{j_2}} z_{j_1 j_2}^{mn} \leq 1, \tag{14}$$

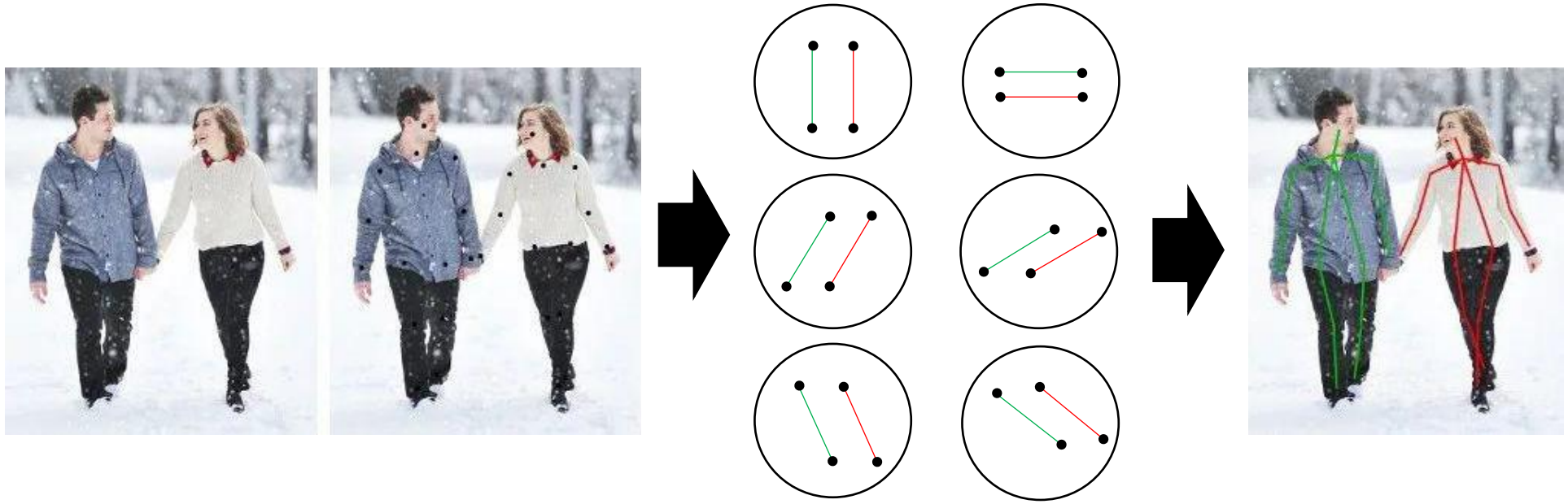
$$\forall n \in \mathcal{D}_{j_2}, \sum_{m \in \mathcal{D}_{j_1}} z_{j_1 j_2}^{mn} \leq 1, \tag{15}$$

Human Pose Estimation



EAI Bonus Lab introduction

- Design your own grouping algorithm.



Environment setting

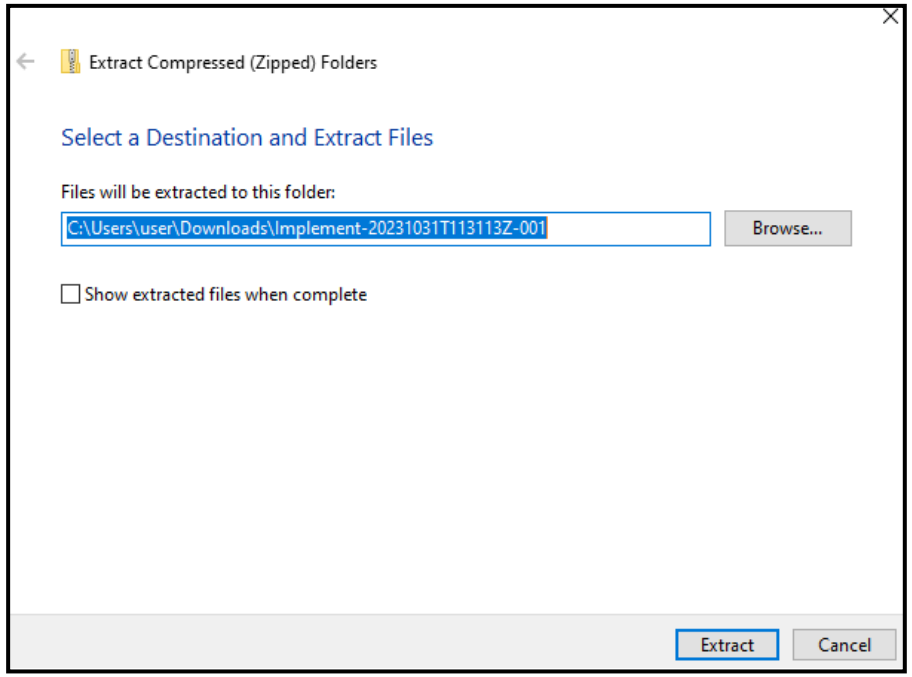
- Anaconda

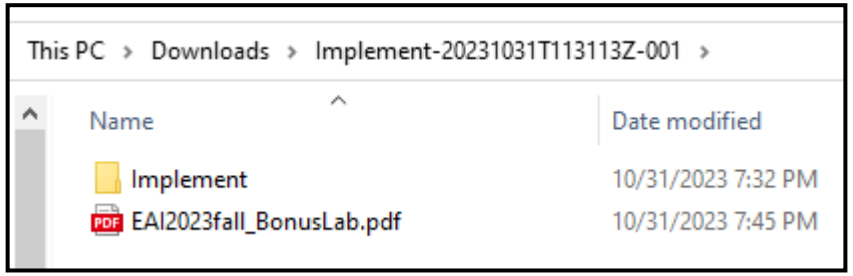
Anaconda is a distribution of the Python programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. (download link)

Environment setting

- Download and extract file

  https://drive.google.com/drive/folders/1JLmCjdcOLL5ComxvRWUQHwVK8U8Ujqu8?usp=drive_link

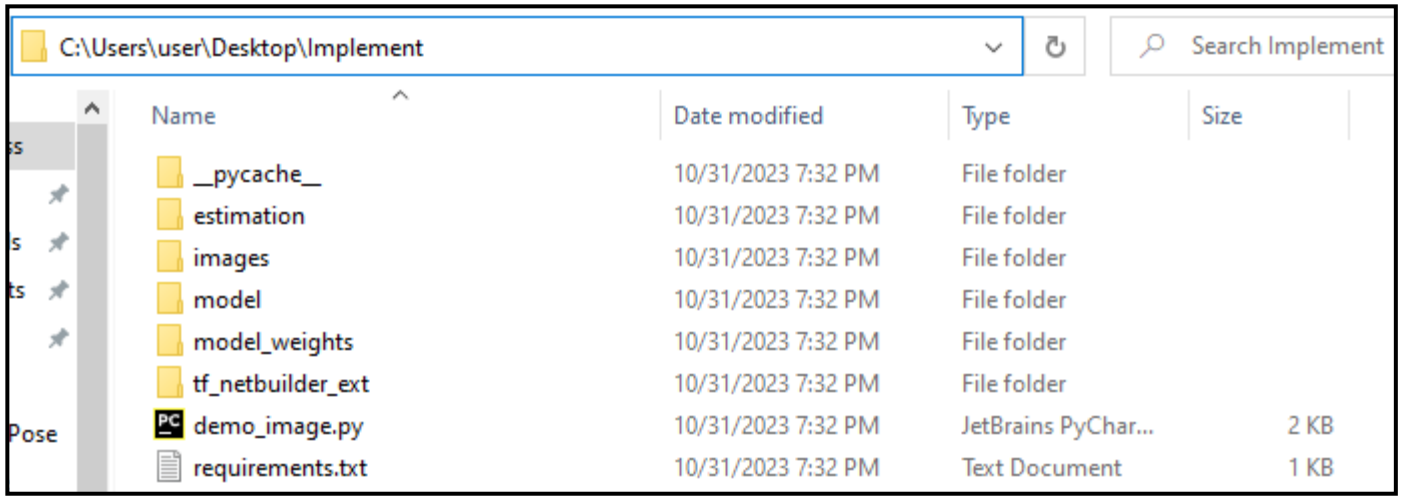




Environment setting

Open Anaconda Prompt and type the following command line below:

- conda create --name poseLab python=3.9
- conda activate poseLab
- conda install git
- cd C:\Users\user\Desktop\Implement (You need to use your own path.)
- pip install -r requirements.txt

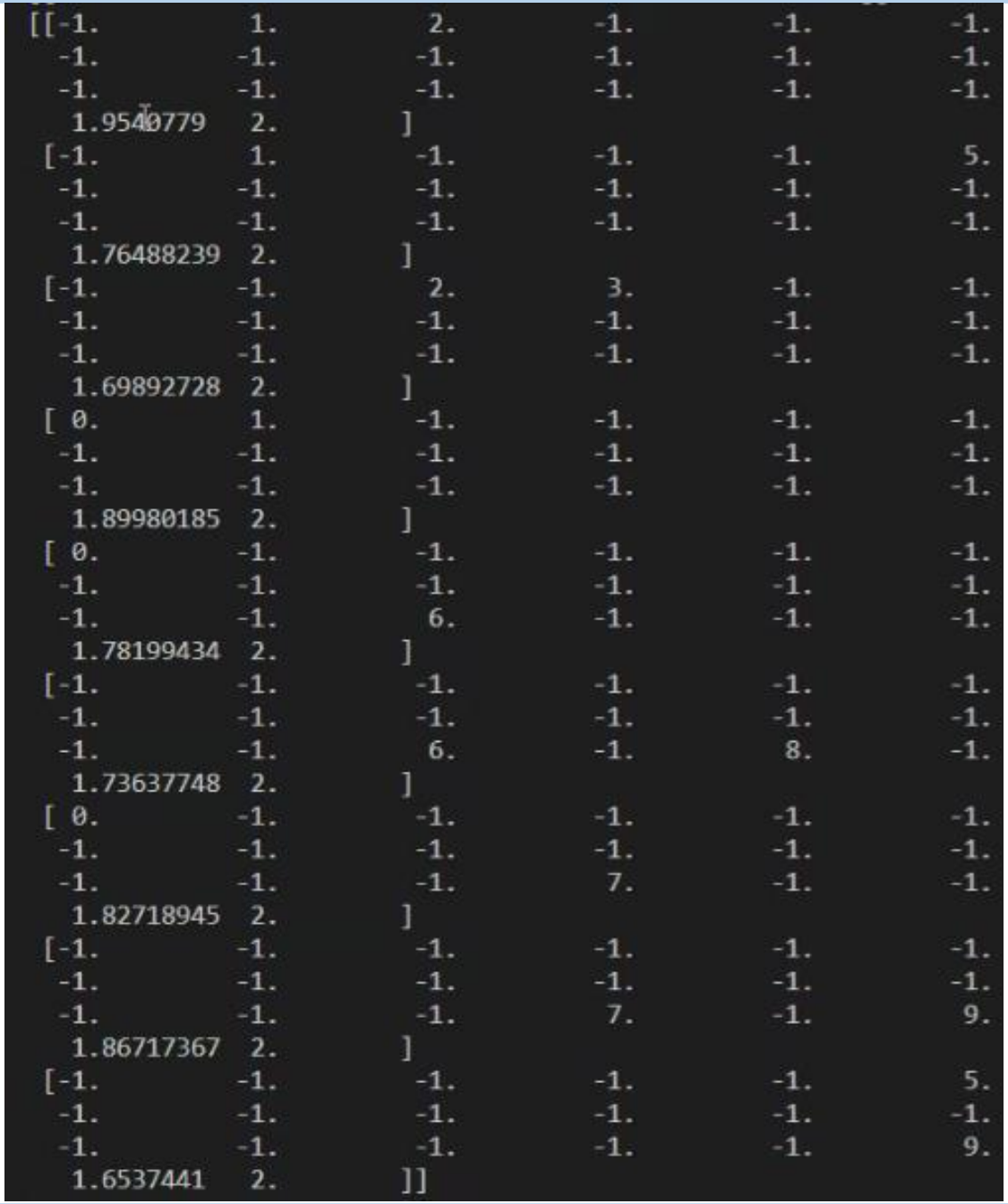


Lab_Task

```
position meaning:
    [   [nose       , neck          , right_shoulder , right_elbow      , right_wrist  , left_shoulder
         left_elbow , left_wrist    , right_hip      , right_knee       , right_ankle  , left_hip
         left_knee  , left_ankle    , right_eye      , left_eye         , right_ear    , left_ear
         score, parts_num],
    ]
```

- Complete Implement/estimation/merge.py
- Input: subset = [[-1. 1. 2. -1. -1. -1.
  -1. -1. -1. -1. -1. -1.
  -1. -1. -1. -1. -1. -1.
  1.9540779 2.]
  [-1. 1. -1. -1. -1. 5.
  -1. -1. -1. -1. -1. -1.
  -1. -1. -1. -1. -1. -1.
  1.76488239 2.]]

Lab_Task



```
[[-1.         1.         2.        -1.        -1.        -1.
  -1.        -1.        -1.        -1.        -1.        -1.
  -1.        -1.        -1.        -1.        -1.        -1.
   1.9540779  2.       ]
 [-1.         1.        -1.        -1.        -1.         5.
  -1.        -1.        -1.        -1.        -1.        -1.
  -1.        -1.        -1.        -1.        -1.        -1.
   1.76488239 2.       ]
 [-1.        -1.         2.         3.        -1.        -1.
  -1.        -1.        -1.        -1.        -1.        -1.
  -1.        -1.        -1.        -1.        -1.        -1.
   1.69892728 2.       ]
 [ 0.         1.        -1.        -1.        -1.        -1.
  -1.        -1.        -1.        -1.        -1.        -1.
  -1.        -1.        -1.        -1.        -1.        -1.
   1.89980185 2.       ]
 [ 0.        -1.        -1.        -1.        -1.        -1.
  -1.        -1.        -1.        -1.        -1.        -1.
  -1.        -1.         6.        -1.        -1.        -1.
   1.78199434 2.       ]
 [-1.        -1.        -1.        -1.        -1.        -1.
  -1.        -1.        -1.        -1.        -1.        -1.
  -1.        -1.         6.        -1.         8.        -1.
   1.73637748 2.       ]
 [ 0.        -1.        -1.        -1.        -1.        -1.
  -1.        -1.        -1.        -1.        -1.        -1.
  -1.        -1.        -1.         7.        -1.        -1.
   1.82718945 2.       ]
 [-1.        -1.        -1.        -1.        -1.        -1.
  -1.        -1.        -1.        -1.        -1.        -1.
  -1.        -1.        -1.         7.        -1.         9.
   1.86717367 2.       ]
 [-1.        -1.        -1.        -1.        -1.         5.
  -1.        -1.        -1.        -1.        -1.        -1.
  -1.        -1.        -1.        -1.        -1.         9.
   1.6537441  2.       ]]
```

Lab_Task

- Output subset = [[0. 1. 2. 3. -1. 5.
  -1. -1. -1. -1. -1. -1.
  -1. -1. 6. 7. 8. 9.
  16.18416846 9.]]

```
[[ 0.          1.          2.          3.         -1.          5.
  -1.         -1.         -1.         -1.         -1.         -1.
  -1.         -1.          6.          7.          8.          9.
  16.18416846  9.        ]
```

Some problems you may encounter

- The connections may have misjudgments and may encounter some conflicts.

- Remove some conflicting connections when encountering conflicts

Demo

1. Revise the path in demo_image.py
   - input_image = "./images/1.jpg"
2. Open Anaconda Prompt
3. Type the following command line below:
   - conda activate poseLab(your own env name)
   - C:\Users\user\Desktop\Implement (You need to use your own path.)
   - python demo_image.py

Trace Code

- demo_image.py
  1. Main()

demo_image.py

C: > Users > user > Desktop > EAI_Lab4 > demo_image.py > ...

```
14  from tf_netbuilder_ext.extensions import register_tf_netbuilder_extensions
15
16  # ↓ Fill your own ↓ #
17  model_weights_path = "./model_weights/openpose"
18  input_image = "./images/ski_368.jpg"
19  output_image = "output.png"
20  # ↑ Fill your own ↑ #
21
22  register_tf_netbuilder_extensions()
23
24  module = importlib.import_module('model')
25  create_model = getattr(module, "create_openpose")
26  model = create_model()
27  model.load_weights(model_weights_path)
28
29  img = cv2.imread(input_image)  # B,G,R order
30  input_img = img[np.newaxis, :, :, [2, 1, 0]]
31  inputs = tf.convert_to_tensor(input_img)
32
33  outputs = model.predict(inputs)
34  pafs = outputs[10][0, ...]
35  heatmaps = outputs[11][0, ...]
36  cfg = get_default_configuration()
37  coordinates = get_coordinates(cfg, heatmaps)
38  connections = get_connections(cfg, coordinates, pafs)
39  # ↓ You can analsis the time of your algo. ↓ #
40  x = time.time()
41  skeletons = estimate(cfg, connections)
42  y = time.time()
43  print(y-x) # Inference time
44  # ↑ You can analsis the time of your algo. ↑ #
45  output = draw(cfg, img, coordinates, skeletons, resize_fac=8)
46  cv2.imwrite(output_image, output)
```

Trace Code

- coordinates.py
  1. Finds the gaussian peaks coordinates(x,y) in the heatmaps.
  2. Assign the Gaussian peaks Identity number.

```
:param config: pose estimation configuration
:param heatmaps: heatmaps
:param threshold: threshold for the intensity value
                  in the heatmap at the position of a peak
:return: dictionary:
    { body_part_name:
        [(x ,y, score, id),
         (x ,y, score, id),
         ...
        ],
      ...
    }
"""
all_peaks = dict()
peak_counter = 0

# Find gaussian peak
for part_meta in config.body_parts.values():
    hmap = heatmaps[:, :, part_meta.heatmap_idx]

    hmap_right = np.zeros(hmap.shape)
    hmap_right[:, 1:] = hmap[:, :-1]
    hmap_left = np.zeros(hmap.shape)
    hmap_left[:, :-1] = hmap[:, 1:]
    hmap_down = np.zeros(hmap.shape)
    hmap_down[1:, :] = hmap[:-1, :]
    hmap_up = np.zeros(hmap.shape)
    hmap_up[:-1, :] = hmap[1:, :]

    peaks_binary = np.logical_and.reduce(
        (hmap >= hmap_right,
         hmap >= hmap_left,
         hmap >= hmap_down,
         hmap >= hmap_up,
         hmap > threshold))
    peaks = list(zip(np.nonzero(peaks_binary)[1],
                     np.nonzero(peaks_binary)[0]))

    # pack and give id
    sequence_numbers = range(peak_counter, peak_counter + len(peaks))
    peaks_with_score_and_id = [peak + (
        hmap[peak[1], peak[0]],
        seq_num) for peak, seq_num in zip(peaks, sequence_numbers)]

    all_peaks[part_meta.body_part.name] = peaks_with_score_and_id
    peak_counter += len(peaks)

return all_peaks
```

Trace Code

- connections.py
  1. Finds the connection candidates and returns only valid connections.

$$\mathbf{L}^*_{c,k}(\mathbf{p}) = \begin{cases} \mathbf{v} & \text{if } \mathbf{p} \text{ on limb } c,k \\ \mathbf{0} & \text{otherwise.} \end{cases} \quad (9)$$

Here, $\mathbf{v} = (\mathbf{x}_{j_2,k} - \mathbf{x}_{j_1,k})/\|\mathbf{x}_{j_2,k} - \mathbf{x}_{j_1,k}\|_2$ is the unit vector in the direction of the limb.

```
def get_connections(config, coords, paf, threshold=0.05, mid_num=10, minimum_mid_num=8):
    """
    Finds the connection candidates and returns only valid connections.
    :param config: pose estimation configuration.
    :param coords: dictionary with coordinates of all body parts.
    :param paf: paf maps.
    :param threshold: threshold for the intensity value in paf for a given mid point. If value at a mid point
    is below the threshold the mid point is not taken into account.
    :param mid_num: number of mid point for sampling
    :param minimum_mid_num: minimum number of valid mid points for the connection candidate
    :return: list of arrays containing identified connections of a given type :
        [
            array(
              [id1, id2, score1, score2, total_score]
              [id1, id2, score1, score2, total_score]
              ...
            ),
            array(
            ...
            )
        ]
    """
    all_cand_connections = []
    for conn in config.connection_types:
        # select dx and dy PAFs for this connection type
        paf_dx = paf[:, :, conn.paf_dx_idx]
        paf_dy = paf[:, :, conn.paf_dy_idx]
        # get coordinates lists for 2 body part types which belong to the current connection type
        cand_a = coords[conn.from_body_part.name]
        cand_b = coords[conn.to_body_part.name]
        n_a = len(cand_a)
        n_b = len(cand_b)
        max_connections = min(n_a, n_b)
        # lets check each combination of detected 2 body parts - candidate connections
        if n_a != 0 and n_b != 0:
            # here we will store the connection candidates 5 columns:
            # [ body part id1, body part id2, body part score1, body part score2, total score of connection ]
            connection_candidates = np.zeros((0, 5))
            for i in range(n_a):
                for j in range(n_b):
                    # find the distance between the 2 body parts. The expression cand_b[j][:2]
                    # returns an 2 element array with coordinates x,y
                    vec = np.subtract(cand_b[j][:2], cand_a[i][:2])
                    norm = math.sqrt(vec[0] * vec[0] + vec[1] * vec[1])
                    # skip the connection if 2 body parts overlaps
                    if norm == 0:
                        continue
                    # normalize the vector
                    vec = np.divide(vec, norm)
```

Trace Code

- connections.py
  1. Finds the connection candidates and returns only valid connections.

$$E = \int_{u=0}^{u=1} \mathbf{L}_c\left(\mathbf{p}(u)\right) \cdot \frac{\mathbf{d}_{j_2} - \mathbf{d}_{j_1}}{\left\|\mathbf{d}_{j_2} - \mathbf{d}_{j_1}\right\|_2} du, \qquad (11)$$

```
            # get the set midpoints between 2 body parts (their coordinates x,y)
            start_end = list(zip(np.linspace(cand_a[i][0], cand_b[j][0], num=mid_num),
                                 np.linspace(cand_a[i][1], cand_b[j][1], num=mid_num)))
            # having the coord of midpoint we can read the intensity value in paf map at the midpoint
            # for dx component
            vec_x = np.array(
                [paf_dx[int(round(start_end[i][1])), int(
                    round(start_end[i][0]))] for i in range(mid_num)]
            )
            # for dy component
            vec_y = np.array(
                [paf_dy[int(round(start_end[i][1])), int(
                    round(start_end[i][0]))] for i in range(mid_num)]
            )
            # calculate the score for the connection weighted by the distance between body parts
            score_midpts = np.multiply(vec_x, vec[0]) + np.multiply(vec_y, vec[1])
            # get the total score
            total_score = sum(score_midpts) / len(score_midpts)
            # number of midpoints with intensity above the threshold shouldn't be less than 80% of all midpoints
            criterion1 = len(np.nonzero(score_midpts > threshold)[0]) > minimum_mid_num
            criterion2 = total_score > 0
            if criterion1 and criterion2:
                # add this connection to the list  [id1, id2, score1, score2, total score]
                connection_candidates = np.vstack(
                    [connection_candidates,
                     [cand_a[i][3],
                      cand_b[j][3],
                      cand_a[i][2],
                      cand_b[j][2],
                      total_score]])
        # sort the array by the total score - descending. (the sorted array is reversed by the expression [::-1])
        sorted_connections = connection_candidates[connection_candidates[:, 4].argsort()][::-1]
        # make sure we get no more than max_connections
        all_cand_connections.append(sorted_connections[:max_connections, :])
    else:
        # not found any body parts but we still need to add empty list to preserve the correct indexing in the
        # output array
        all_cand_connections.append([])
return all_cand_connections
```

Trace Code

- estimators.py
  1. Convert connections to human subsets

```
    :param connections: valid connections
    :param min_num_body_parts: minimum number of body parts for a skeleton
    :param min_score: minimum score value for the skeleton
    :return: list of skeletons. Each skeleton has a list of identifiers of body parts:
        [
            [id1, id2,...,idN, score, parts_num],
            [id1, id2,...,idN, score, parts_num]
            ...
        ]
    """

    # 2 extra slots for number of valid parts and overall score
    number_of_slots = config.body_parts_size() + 2

    # the connections are solely used to group body parts into separate skeletons. As a result we will
    # get an array where each row represents a skeleton, each column contains an identifier of
    # specific body part belonging to a skeleton (plus 2 extra columns for: num of parts and skeleton total score)
    # we will be adding the skeletons to this array:
    subset = np.empty((0, number_of_slots))

    for k, conn in enumerate(config.connection_types):
        if len(connections[k]) > 0:

            # retrieve id and score of all body parts pairs for the current connection type
            part_a = connections[k][:, [0, 2]]  # idA, scoreA
            part_b = connections[k][:, [1, 3]]  # idB, scoreB

            # determine the slot number for 2 body parts types
            slot_idx_a = config.body_parts[conn.from_body_part].slot_idx
            slot_idx_b = config.body_parts[conn.to_body_part].slot_idx

            for i in range(len(connections[k])):
                found = 0
                slot_idx = [-1, -1]
                for j in range(len(subset)):
                    if subset[j][slot_idx_a] == part_a[i, 0] and \
                            subset[j][slot_idx_b] == part_b[i, 0]:
                        slot_idx[found] = j
                        found += 1

                # if find no partA in the subset, create a new subset
                if not found:
                    row = -1 * np.ones(number_of_slots)
                    row[slot_idx_a] = part_a[i, 0]
                    row[slot_idx_b] = part_b[i, 0]
                    row[-1] = 2
                    row[-2] = part_a[i, 1] + part_b[i, 1] + connections[k][i][2]
                    subset = np.vstack([subset, row])

    subset = merge(subset)
    return subset
```

Trace Code

- merge.py
  1. Grouping

```python
def merge(subset, min_num_body_parts=4, min_score=0.4):
    """
    Estimates the skeletons.
    :param connections: valid connections
    :param min_num_body_parts: minimum number of body parts for a skeleton
    :param min_score: minimum score value for the skeleton
    :return: list of skeletons. Each skeleton has a list of identifiers of body parts:
        [
            [id1, id2,...,idN, score, parts_num],
            [id1, id2,...,idN, score, parts_num]
            ...
        ]

    position meaning:
        [   [nose       , neck           , right_shoulder , right_elbow      , right_wrist  , left_shoulder
             left_elbow , left_wrist     , right_hip      , right_knee       , right_ankle  , left_hip
             left_knee  , left_ankle     , right_eye      , left_eye         , right_ear    , left_ear
             score, parts_num],
        ]
    """

    # 2 step :
    #---merge----
    # Merge the limbs in the subset
    # score : score
    # parts_num : How many limbs are in the subset
    ##############################


    # after merge
    #---delete---
    # Delete the non-compliant subset
    # 1. parts_num < 4
    # 2. Average score(score / parts_num) < 0.4
    ##########################################
    delete_idx = []
    for i in range(len(subset)):
        if ???????????????????  `  ????????????????:  # revise here
            delete_idx.append(i)
    subset = np.delete(subset, delete_idx, axis=0)
    return subset
```

Report

1. Complete description of your algorithm.

2. The results of test pictures.

Score assignment

1. Code(60%)

2. Report(40%)

File submission

- Upload platform: NCKU moodle

- Upload compressed .zip file, named StudentID_BonusLab.zip, including:
  - merge.py
  - StudentID_BonusLab_report.pdf

END

Advisor：Tsai, Chia-Chi