NUS-ISS
Real Time Audio-Visual Sensing and Sense Making

Pose Estimation

Dr. Tan Jen Hong
Lecturer & Consultant
Institute of System Science
National University of Singapore
issjht@nus.edu.sg

The challenge

Deeper understanding

Source: https://www.needpix.com/photo/1766772/car-vehicle-concept-auto-automobile-automotive-transportation-drive-speed

- Deep learning can now detect multiple objects in a scene
- However many times, there is a need not just to identify human or objects in a scene, but to understand the behaviour of human or the state of objects in a scene
- Solution: Estimate the pose of human or objects in a scene
- OpenPose: the first open-source real-time system for multi-person 2D pose detection

Source: https://www.needpix.com/photo/1766772/car-vehicle-concept-https://www.wallpaperflare.com/woman-and-man-dancing-under-light-dance-dancers-people-performance-wallpaper-engxr

The challenge

Deeper understanding

- We are going to used a model trained on COCO dataset
- The model tries to identfiy 18 key anatomical points on a person
- Model file can be obtained from here: https://github.com/CMU-Perceptual-Computing-Lab/openpose/tree/master/models/pose/coco
- Model weight: http://posefs1.perception.cs.cmu.edu/OpenPose/models/pose/coco/pose_iter_440000.caffemodel

OpenPose

The image we are going to use as example

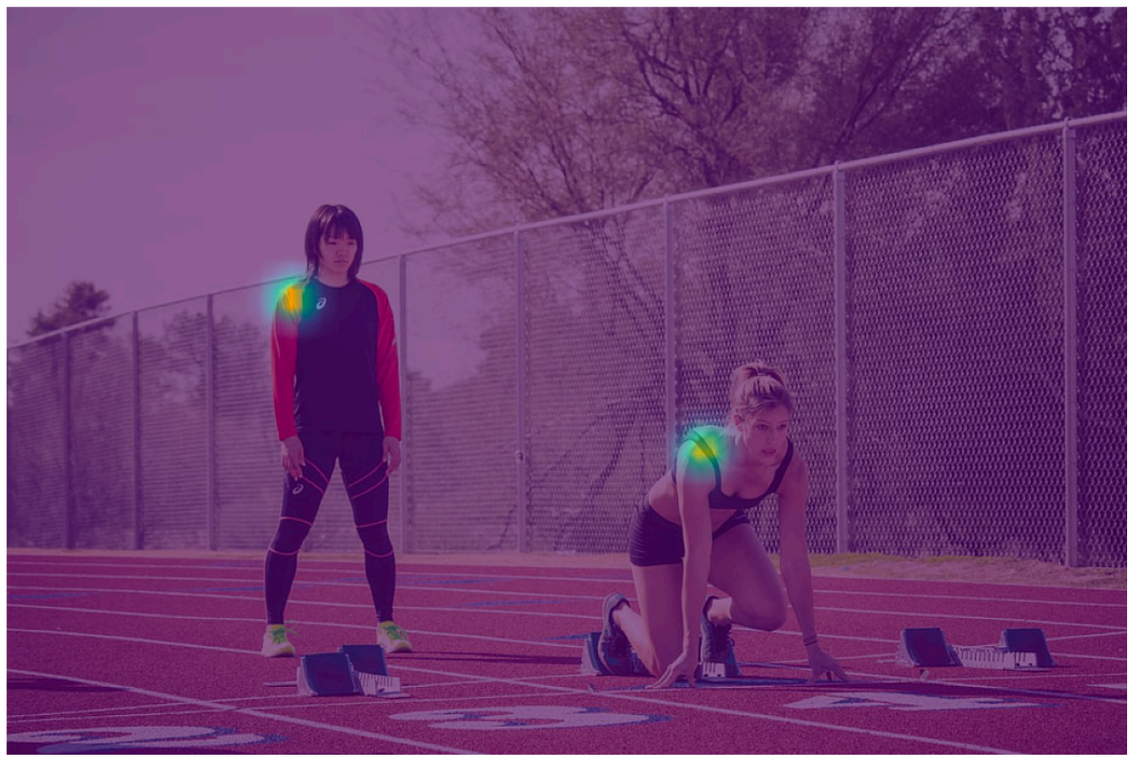
Source: https://www.wallpaperflare.com/two-women-near-gray-fence-at-daytime-person-track-sprinter-wallpaper-zpkxn

OpenPose

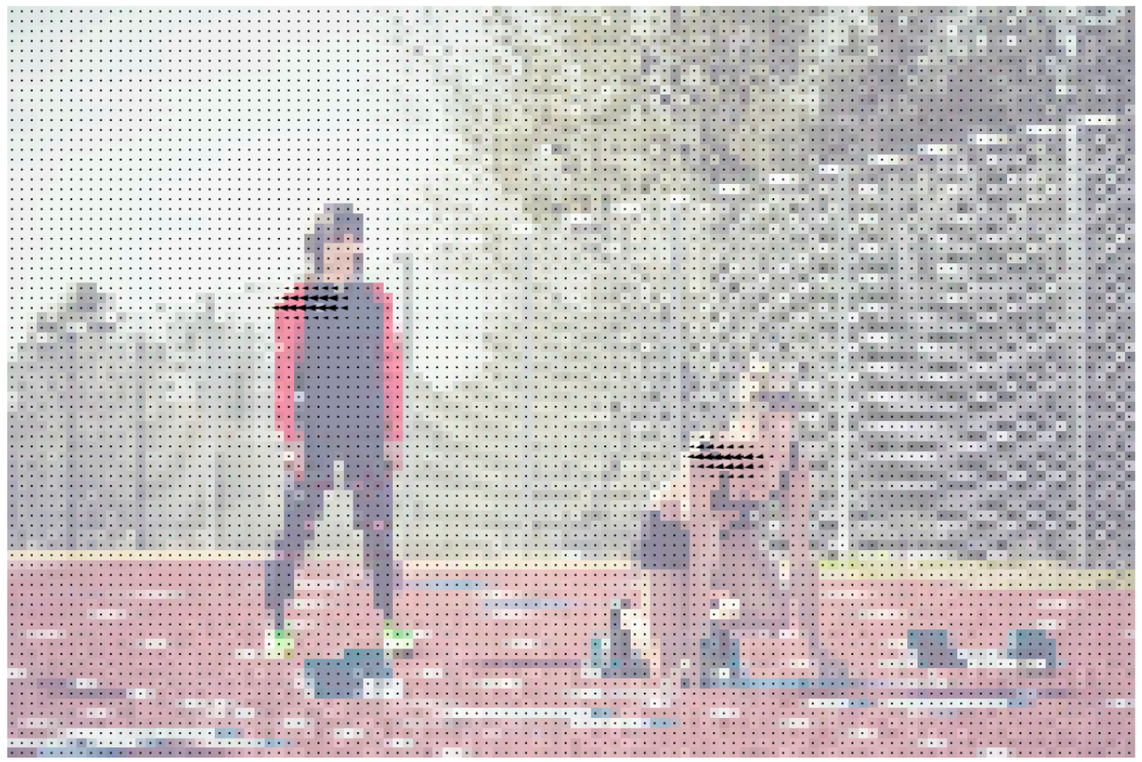
Step 1

- There are three main steps in pose estimation
- Step 1, generate Confidence Maps and Part Affinity Fields from deep learning net

Confidence map for right shoulder



Part affinity field for neck to right shoulder



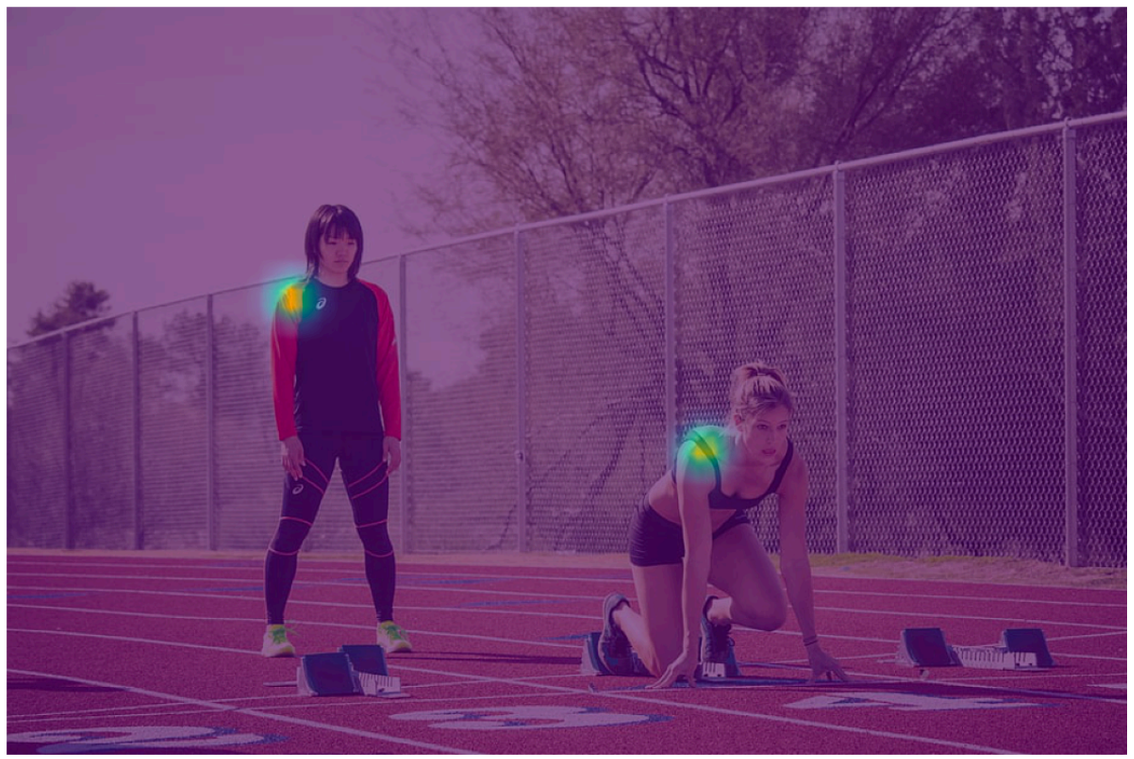
OpenPose

Step 1

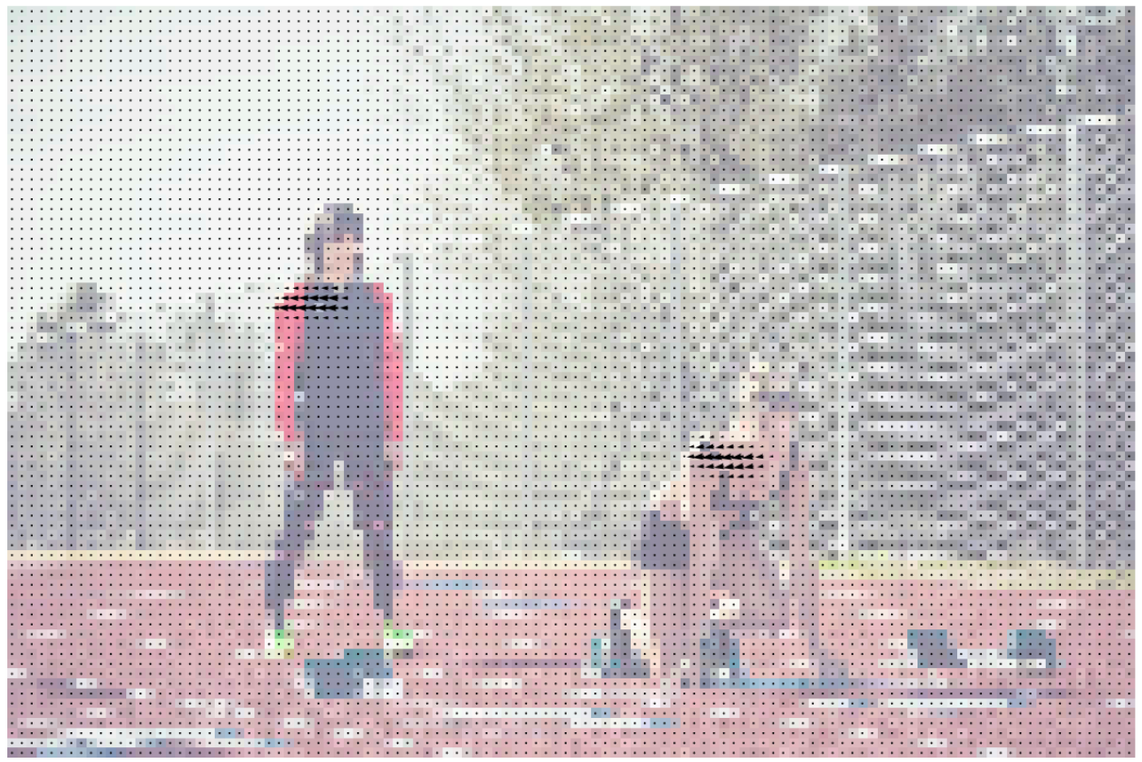
- Confidence maps tell us where the anatomical points are likely to be

- Part affinity fields tell us how anatomical points should connect with each other points

Confidence map for right shoulder



Part affinity field for neck to right shoulder



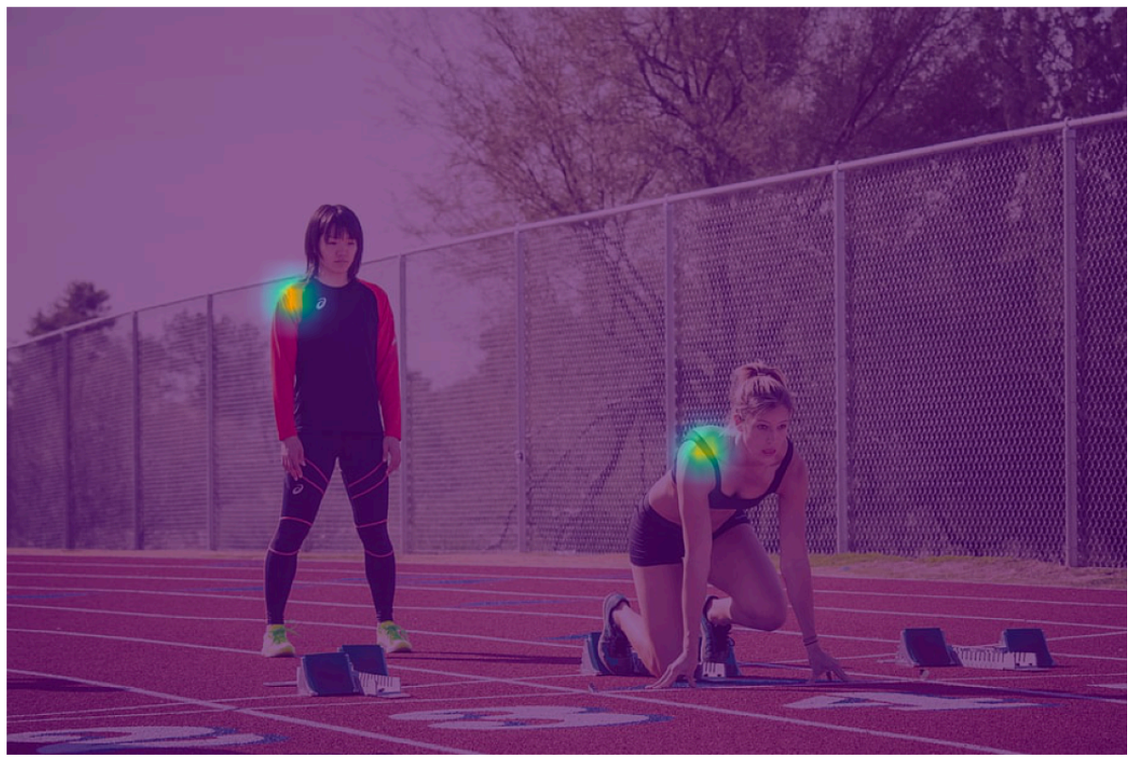
OpenPose

Step 2

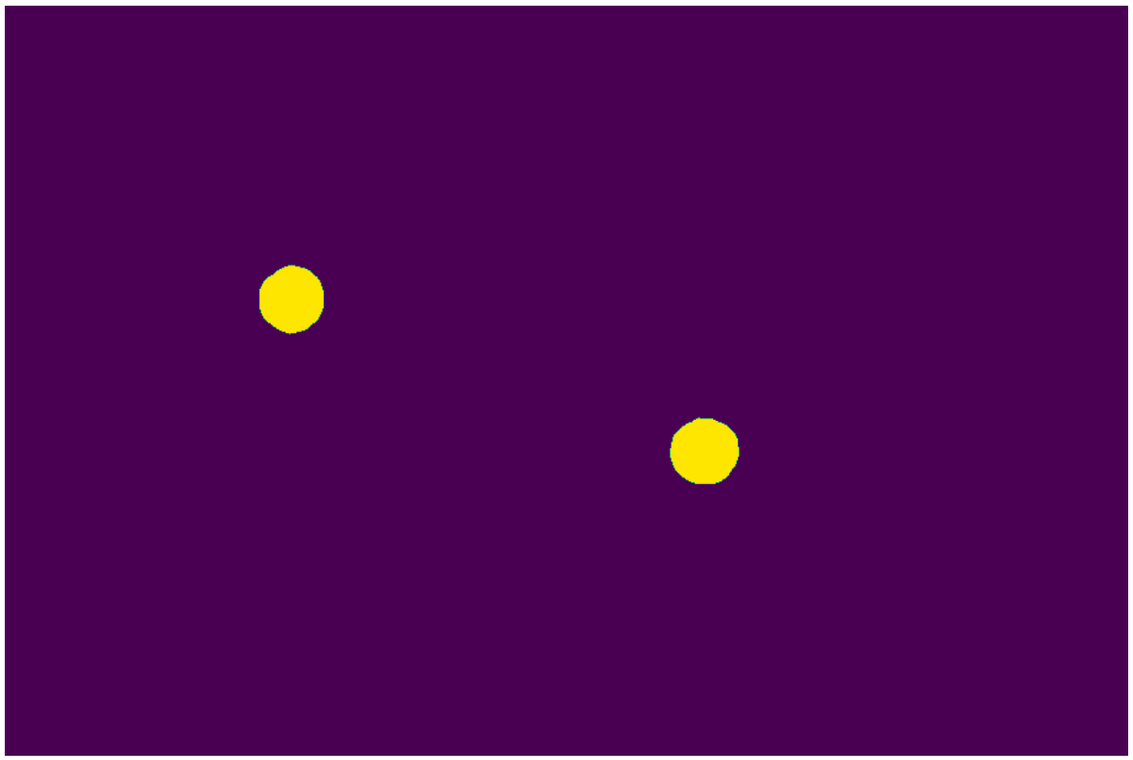
- Step 2, identify the anatomical points from the Confidence Maps

- Confidence map gives a region of possible locations for each anatomical point, there is a need to reduce these regions into specific points

Confidence map for right shoulder



The located regions for right shoulder



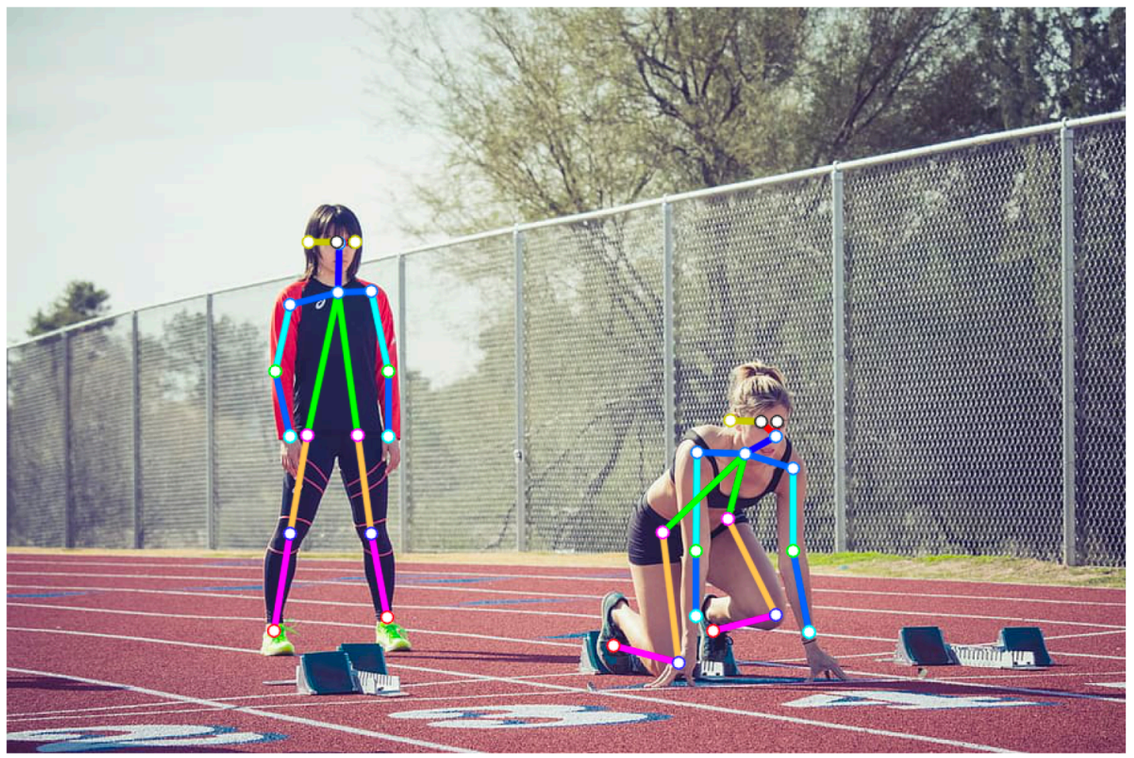
OpenPose

Step 3

- Step 3, pair the anatomical points and form human subjects

All the identified anatomical points

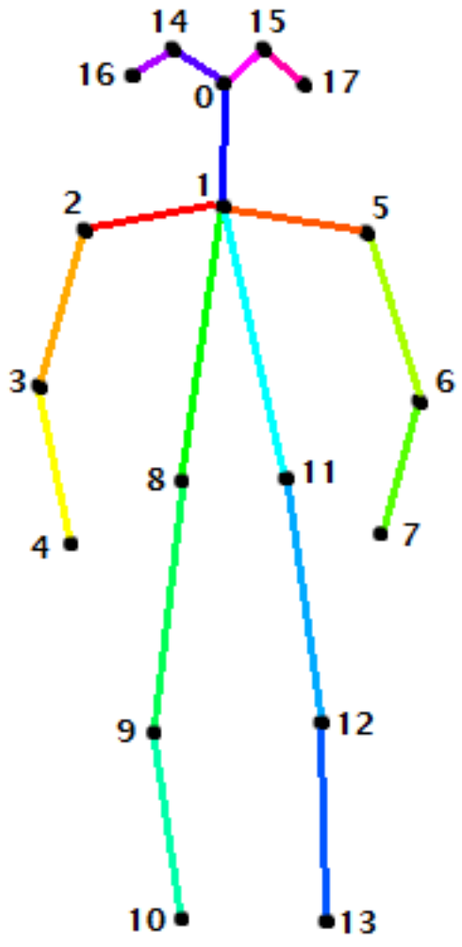
The final output



To start

setup a few things

```
> import cv2
> import time
> import numpy as np
> import matplotlib.pyplot as plt

> img          = cv2.imread("sprinter.jpg")
> prototxt     = "pose/coco/pose_deploy_linevec.prototxt"
> caffemodel   = "pose/coco/pose_iter_440000.caffemodel"
> numOfPts     = 18
> ptsLbl       = ['Nose',           # 0
                  'Neck',           # 1
                  'R:Shoulder',     # 2
                  'R:Elbow',        # 3
                  'R:Wrist',        # 4
                  'L:Shoulder',     # 5
                  'L:Elbow',        # 6
                  'L:Wrist',        # 7
                  'R:Hip',          # 8
                  'R:Knee',         # 9
                  'R:Ankle',        # 10
                  'L:Hip',          # 11
                  'L:Knee',         # 12
                  'L:Ankle',        # 13
                  'R:Eye',          # 14
                  'L:Eye',          # 15
                  'R:Ear',          # 16
                  'L:Ear']          # 17
```



The 18 anatomical points

Source: https://arvrjourney.com/human-pose-estimation-using-openpose-with-tensorflow-part-2-e78ab9104fc8

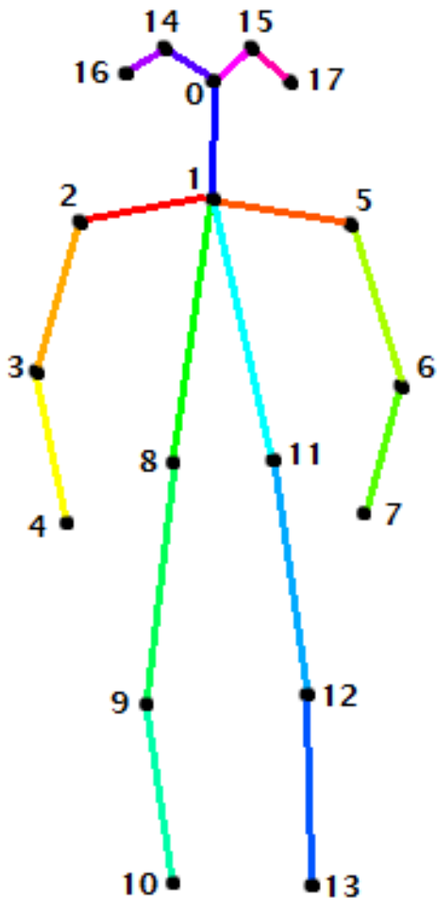
To start

setup a few things

- pafCh is the pairs of channel (from the net output) that corresponds to the pairs in ‘links’

- For example, the PAFs for link [1,2] are located at channel 31, 32 of the net output

```
> links       = [[1,2],
                 [1,5],
                 [2,3],
                 [3,4],
                 [5,6],
                 [6,7],
                 [1,8],
                 [8,9],
                 [9,10],
                 [1,11],
                 [11,12],
                 [12,13],
                 [1,0],
                 [0,14],
                 [14,16],
                 [0,15],
                 [15,17],
                 [2,17],
                 [5,16]]
```




```
> pafCh       = [[31,32],
                 [39,40],
                 [33,34],
                 [35,36],
                 [41,42],
                 [43,44],
                 [19,20],
                 [21,22],
                 [23,24],
                 [25,26],
                 [27,28],
                 [29,30],
                 [47,48],
                 [49,50],
                 [53,54],
                 [51,52],
                 [55,56],
                 [37,38],
                 [45,46]]
```

```
# In total there are 19 pairs in 'links'
```

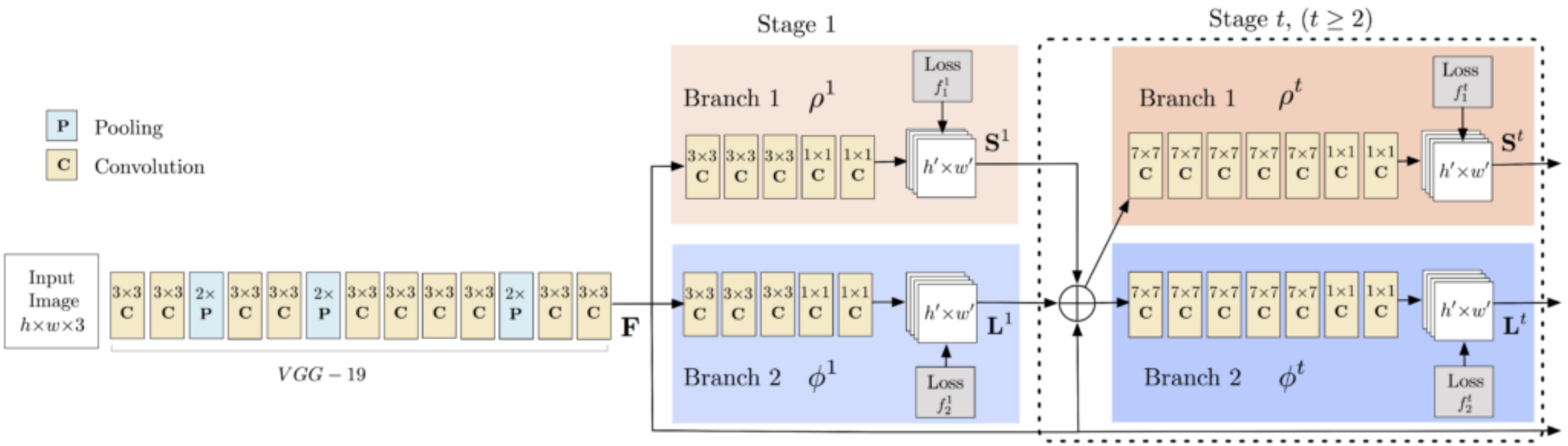
Step 1

Generate Confidence Maps and Part Affinity Fields

OpenPose

Net structure drawn by the original authors ….

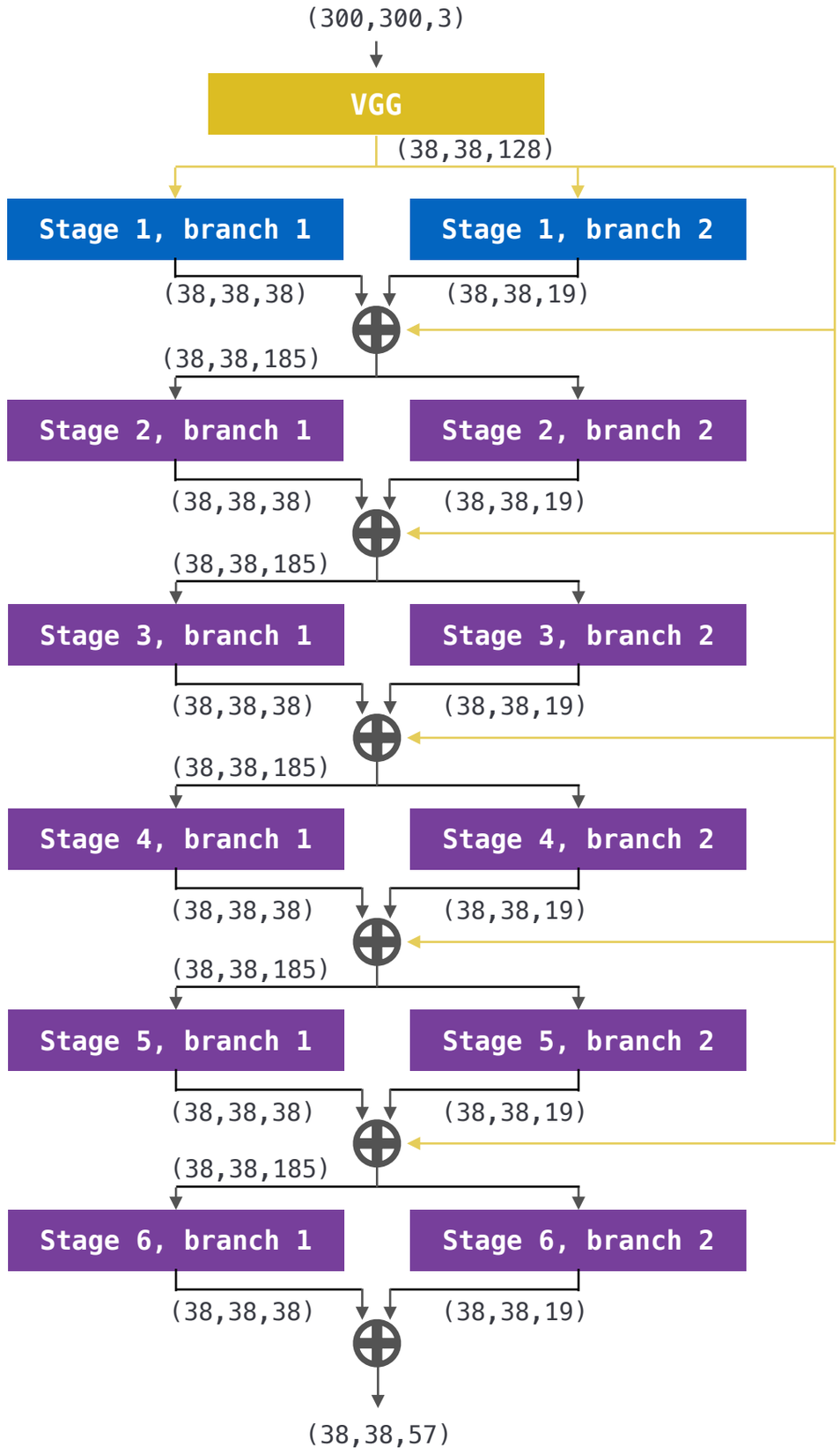
- The figure is hard to understand, see mine in next few slides



Source: https://www.learnopencv.com/multi-person-pose-estimation-in-opencv-using-openpose/

OpenPose

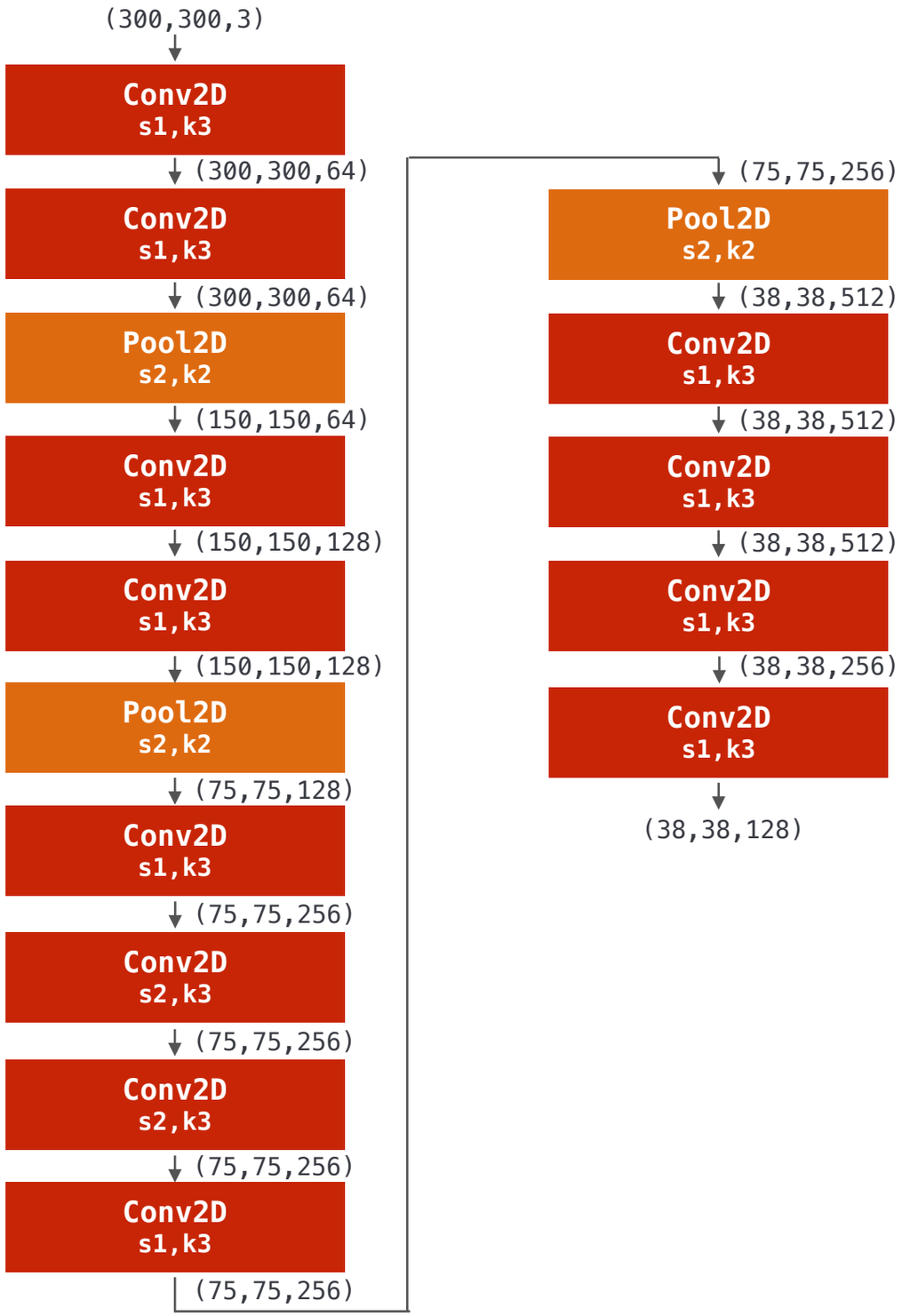
Net structure, overview



Note: The input to the net can be of any size, not just restricted to (300,300,3)

OpenPose

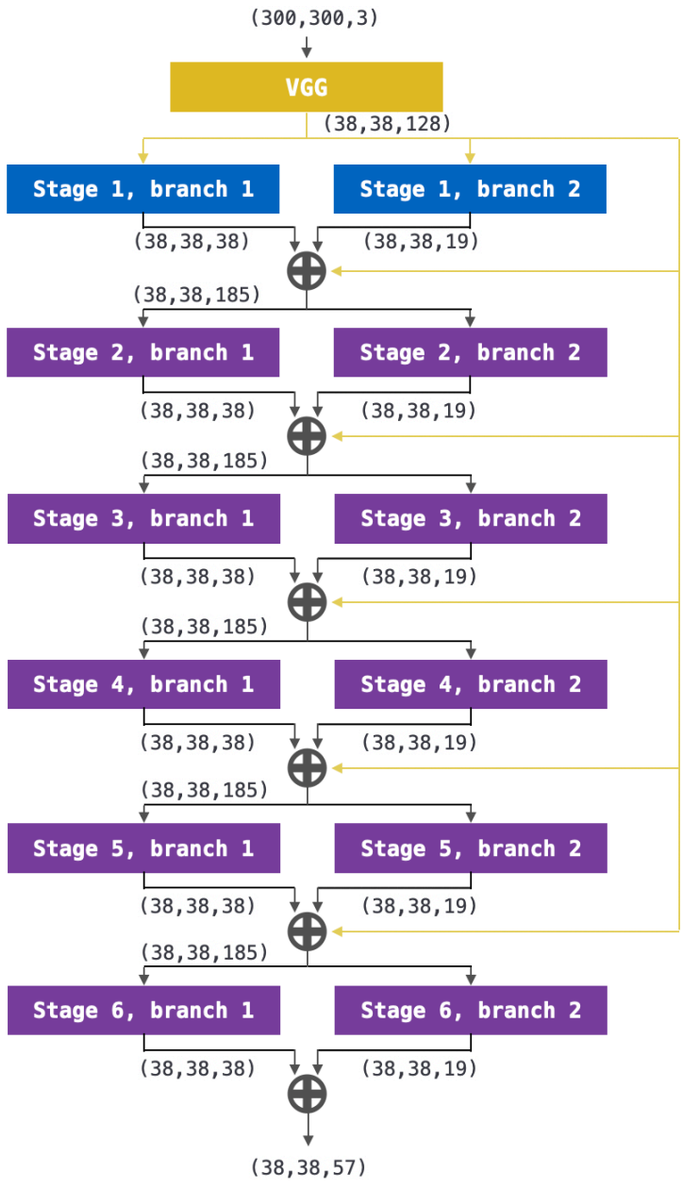
Net structure, VGG



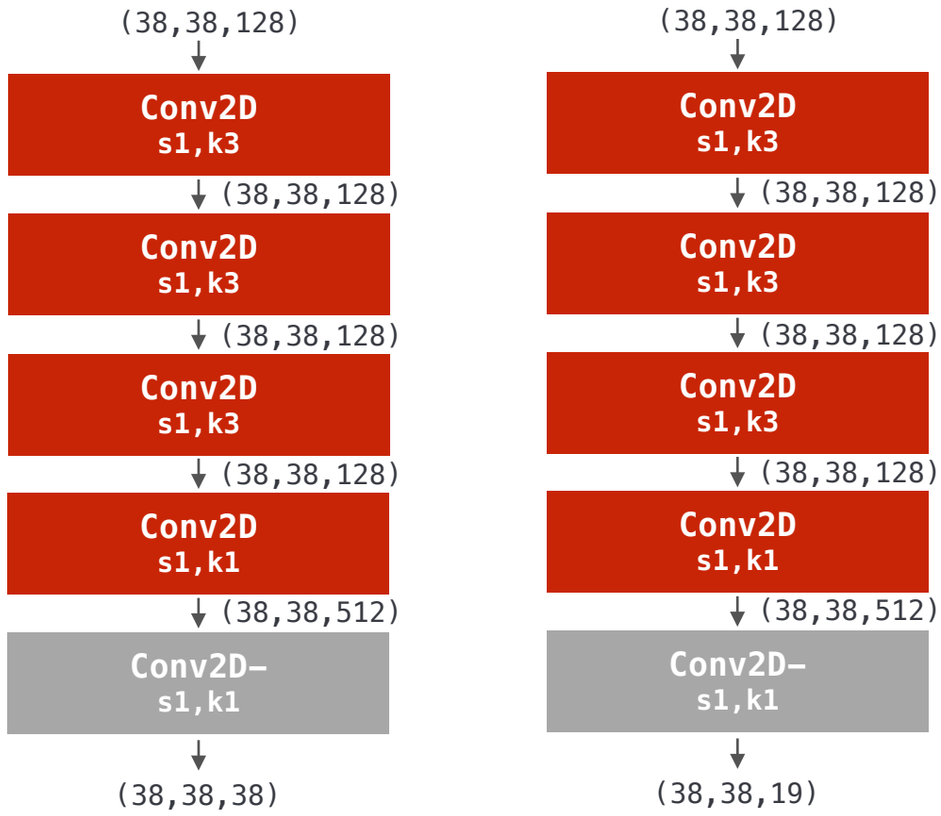
Conv2D	2D convolution with padding and ReLU
Pool2D	2D pooling
s1	stride (1, 1)
s2	stride (2, 2)
k1	kernel of size (1, 1)
k3	kernel of size (3, 3)

OpenPose

Net structure, stage 1

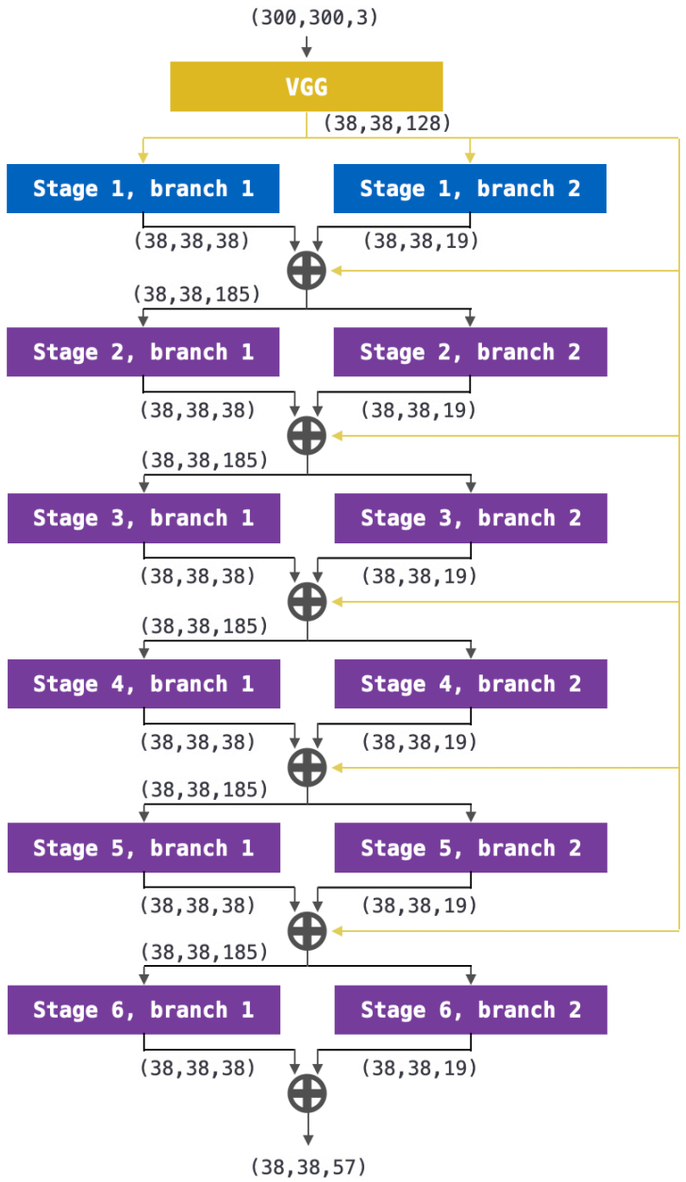


Conv2D	2D convolution with padding and ReLU
Conv2D–	2D convolution with padding but without ReLU
s1	stride (1, 1)
s2	stride (2, 2)
k1	kernel of size (1, 1)
k3	kernel of size (3, 3)

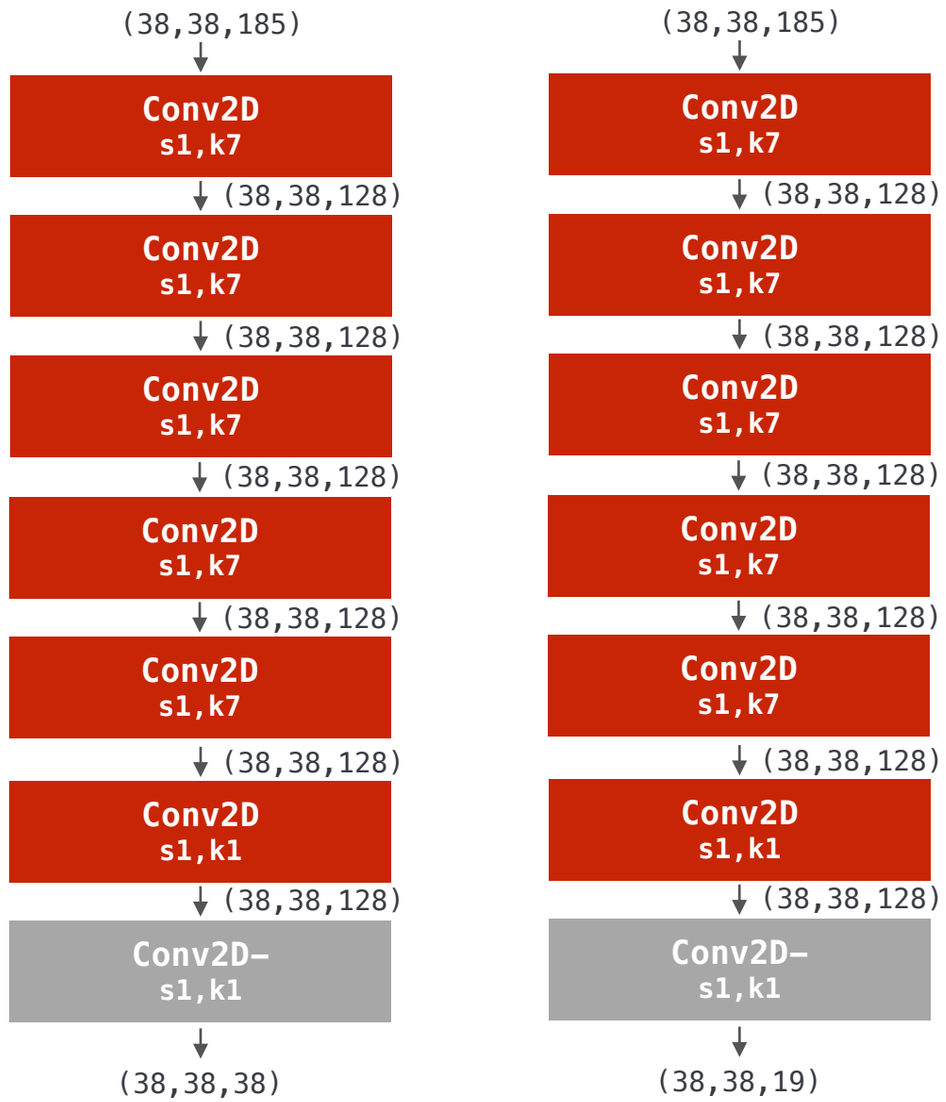


OpenPose

Net structure, stage 2 to stage 6



Conv2D	2D convolution with padding and ReLU
Conv2D–	2D convolution with padding but without ReLU
s1	stride (1, 1)
s2	stride (2, 2)
k1	kernel of size (1, 1)
k3	kernel of size (3, 3)



branch 1

branch 2

Get the net working

Forward inference

- Load the model and perform the inference; the image size is 607 x 910 (H X W)

- The shape of the `output` is `(1,57,46,69)`

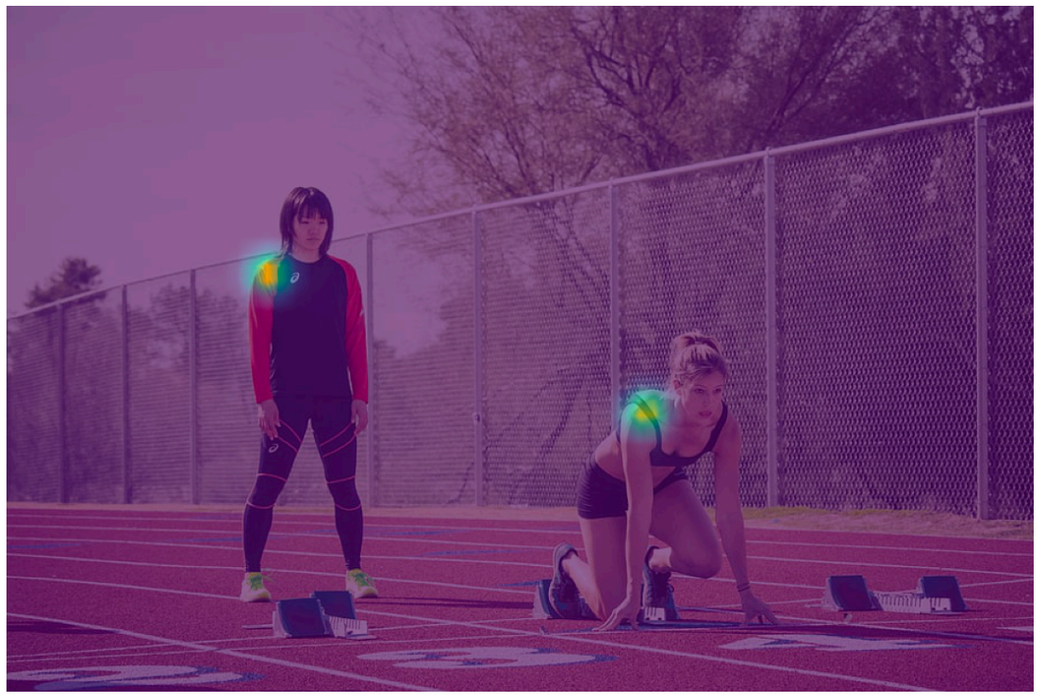
```
> timeBegin   = time.time()

> (H,W,_)     = img.shape          Going to be used for the purpose of resizing
> net         = cv2.dnn.readNetFromCaffe(prototxt,
                                         caffemodel)   Load the deep learning model
> iptH        = 368                Fixed the height of the input to net
> iptW        = int((iptH/H)*W)    Get the corresponding width
> blob        = cv2.dnn.blobFromImage(image=img,
                                      scalefactor=1.0/255,   Rescale the image value to 0~1
                                      size=(iptW,iptH),      The input size to the net
                                      mean=(0,0,0),          No substraction on value is required
                                      swapRB=False,          No swapping of between R and B channel
                                      crop=False)            No cropping
> net.setInput(blob)
                                                             Perform the inference
> output      = net.forward()
> print("Forward duration: %.2fs" % (time.time()-timeBegin))   Report the time needed for the inference
```

Step 2

Identify the anatomical points from Confidence Maps

The next challenge

Get the points



- A need to reduce each (almost) circular region to a single point
- Confidence maps gives us a region of possible anatomical locations. But confidence maps can be noisy, it may have some scattered values across the map.
- To remove those scattered values, image smoothing is required and probability values below a certain threshold to be discarded
- Loop through each region to find the point that has the maximum value
- The point with the maximum value will be taken as an anatomical point

The next challenge

Get all the points

- `getAllPoints` is designed to get all the anatomical points for a single image
- The function returns two outputs.
- One is a list consists of 18 items, each item (which is also a list, consists of points) corresponds to a confidence map.

- The second output is a numpy array that consists of all the points

Get all the points

The function

```
> def getAllPoints(cfMaps,
                   imgWidth,
                   imgHeight,
                   cfThres=0.1,
                   numOfKeyPts=18):
    …

> (ptGrp,ptList)  = getAllPoints(cfMaps=output,
                                 imgWidth=W,
                                 imgHeight=H)
```

to hold the output from the net

width of original image

height of original image

threshold value for searchPts function

the number of confidence maps to look into

The next challenge

Let's first take a look at the outputs

point index point index

Index	Type	Size	Value
0	list	2	[(625, 349, 0.86890876, 0), (270, 192, 0.8351924, 1)]
1	list	2	[(600, 363, 0.8517196, 2), (269, 232, 0.7101424, 3)]
2	list	2	[(561, 362, 0.72589225, 4), (230, 242, 0.7667445, 5)]
3	list	2	[(560, 442, 0.82478887, 6), (218, 296, 0.76174384, 7)]
4	list	2	[(560, 495, 0.5105155, 8), (230, 349, 0.8749416, 9)]
5	list	2	[(639, 375, 0.7806273, 10), (296, 231, 0.83989304, 11)]
6	list	2	[(639, 442, 0.8433347, 12), (310, 296, 0.80592227, 13)]
7	list	3	[(573, 494, 0.11944631, 14), (652, 508, 0.78840846, 15), (310, 349, 0. ...
8	list	2	[(533, 427, 0.61125463, 17), (244, 348, 0.7065, 18)]
9	list	3	[(546, 533, 0.65774935, 19), (626, 494, 0.1291445, 20), (230, 428, 0.8 ...
10	list	3	[(493, 519, 0.3776133, 22), (560, 519, 0.1282323, 23), (217, 507, 0.86 ...
11	list	2	[(586, 416, 0.63797826, 25), (285, 348, 0.6445745, 26)]
12	list	2	[(625, 494, 0.69098675, 27), (296, 428, 0.84796077, 28)]
13	list	2	[(574, 507, 0.2159471, 29), (309, 496, 0.77929324, 30)]
14	list	2	[(613, 337, 0.8635507, 31), (268, 191, 0.8052894, 32)]
15	list	2	[(626, 337, 0.9047314, 33), (282, 191, 0.8517861, 34)]
16	list	2	[(588, 336, 0.75467736, 35), (245, 191, 0.7495347, 36)]
17	list	1	[(283, 191, 0.3748921, 37)]

x y probability

- The first output is `ptGrp`, and it looks like the below
- The point index corresponds to the row number in `ptList`

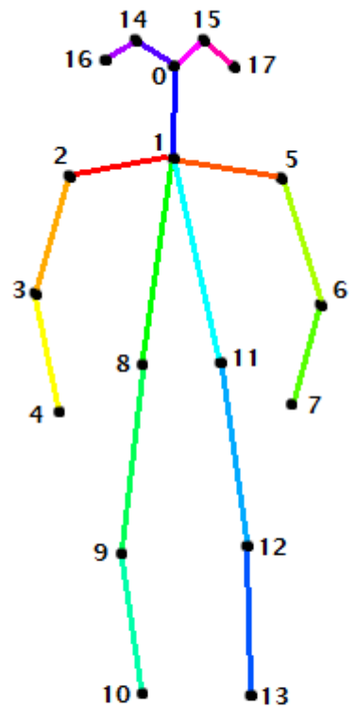
2 points identified for nose

2 points identified for neck

2 points identified for right shoulder

. . . . . .

1 point identified for left ear



The next challenge

Get all the points

- The second output is `ptList`, a numpy array (float64)

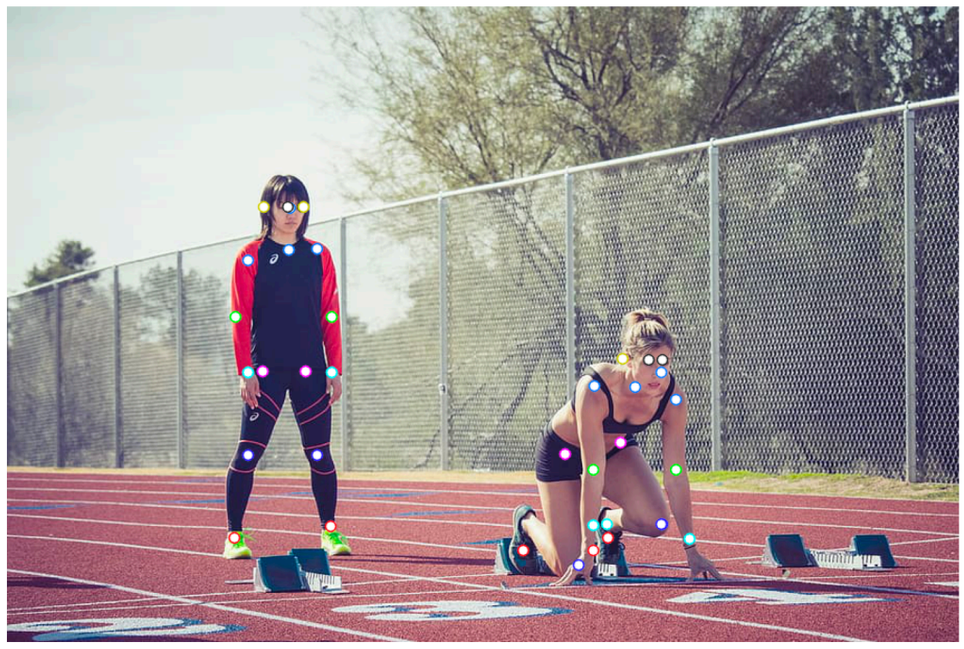
	x	y	probability
0	625	349	0.868909
1	270	192	0.835192
2	600	363	0.85172
3	269	232	0.710142
4	561	362	0.725892
5	230	242	0.766744
6	560	442	0.824789
7	218	296	0.761744
8	560	495	0.510516
9	230	349	0.874942
10	639	375	0.780627
11	296	231	0.839893
12	639	442	0.843335
13	310	296	0.805922
14	573	494	0.119446
15	652	508	0.788408
16	310	349	0.89019
17	533	427	0.611255
18	244	348	0.7065
19	546	533	0.657749
20	626	494	0.129145
21	230	428	0.835902
22	493	519	0.377613
23	560	519	0.128232
24	217	507	0.864127
25	586	416	0.637978
26	285	348	0.644575
27	625	494	0.690987
28	296	428	0.847961
29	574	507	0.215947
30	309	496	0.779293
31	613	337	0.863551
32	268	191	0.805289
33	626	337	0.904731
34	282	191	0.851786
35	588	336	0.754677
36	245	191	0.749535
37	283	191	0.374892

Step 3, part 1

Pair the anatomical points

The next challenge

Get the links

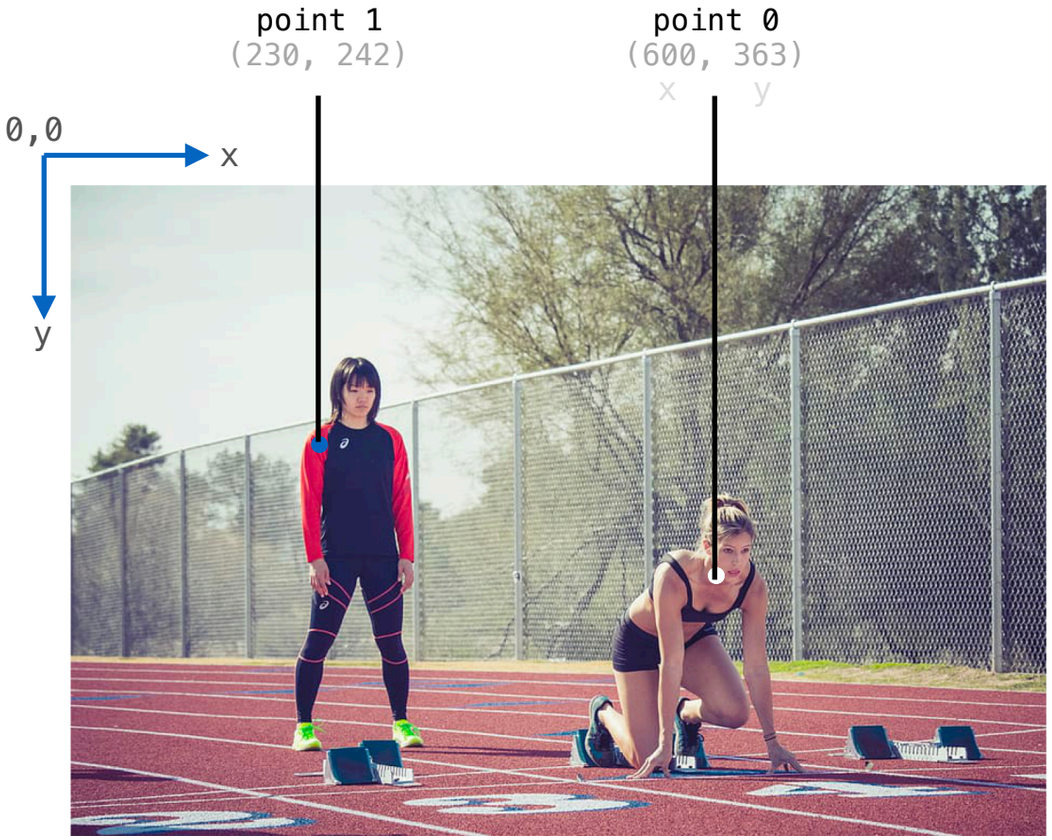


- A need to establish how points are linked to each other
- Solution: For each anatomical link (say, neck to right shoulder), for each identified neck point, loop through every neck-right shoulder links, and evaluate each link against Part Affinity Fields
- Note: each part affinity field is made up by two 2D matrix, extracted from the net output according to the channels specified by `pafCh`

The process
to evaluate a link

- This example tries to evaluate a link from neck (point 0) to right shoulder (point 1)

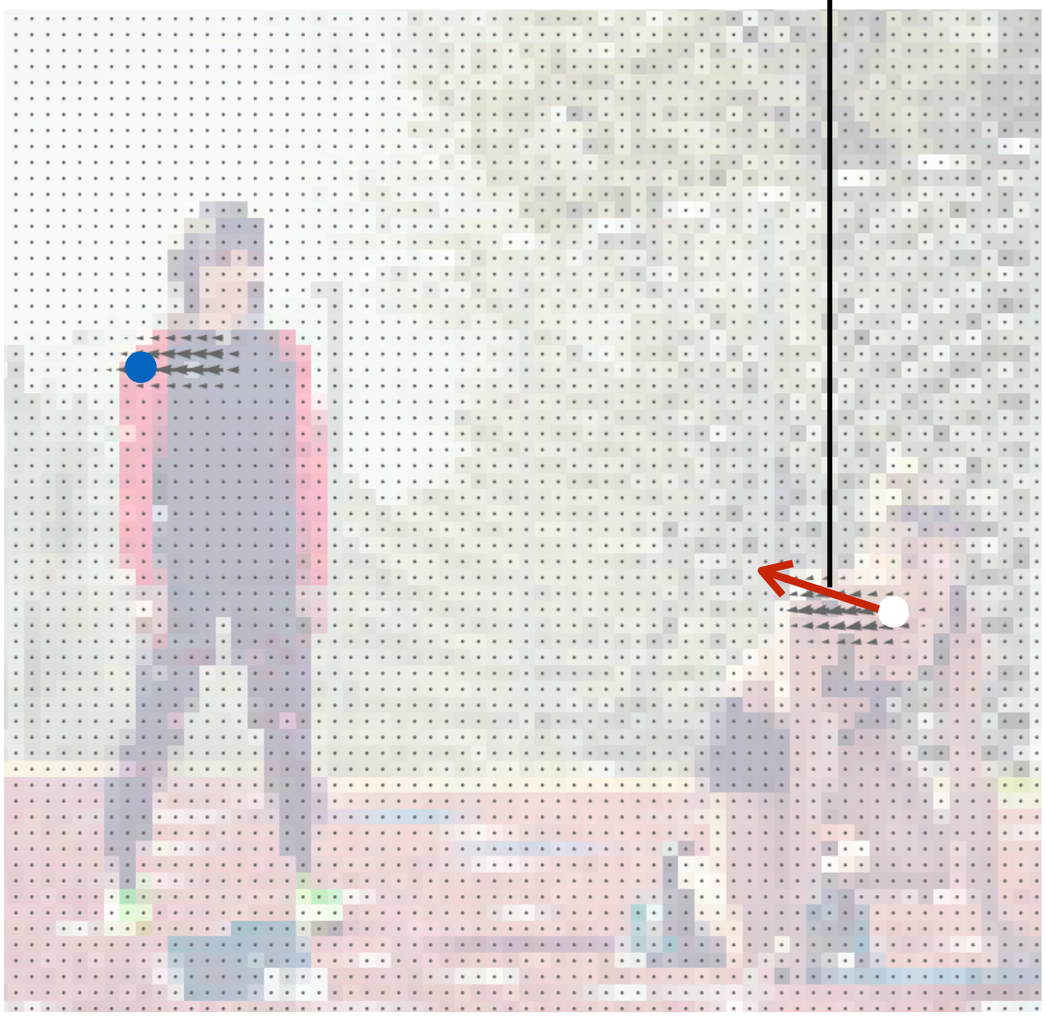
- The corresponding channels to form a part affinity field is 31 and 32 from the net output



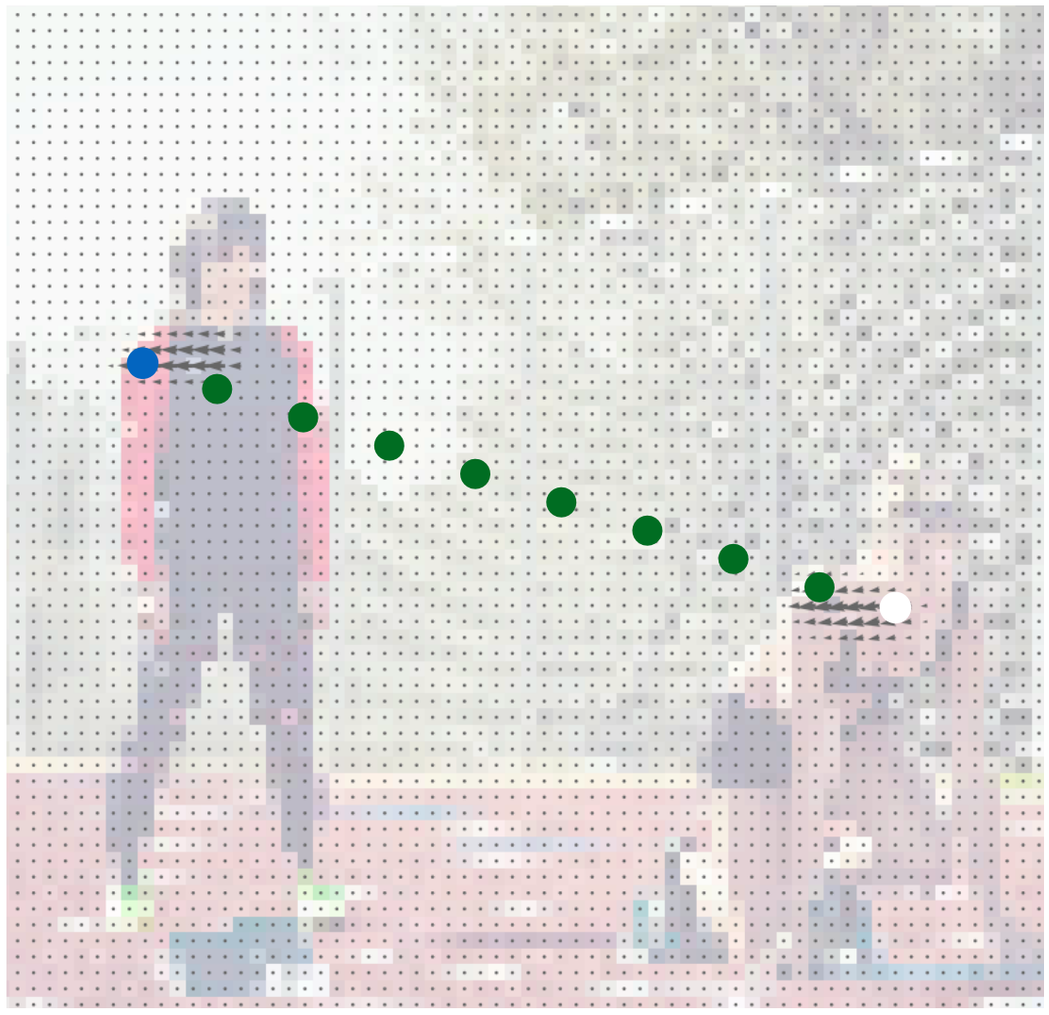
The process
to evaluate a link

unit vector
(*made large for illustration purpose*)

1. Calculate the unit vector for the link (from point 0 to point 1)

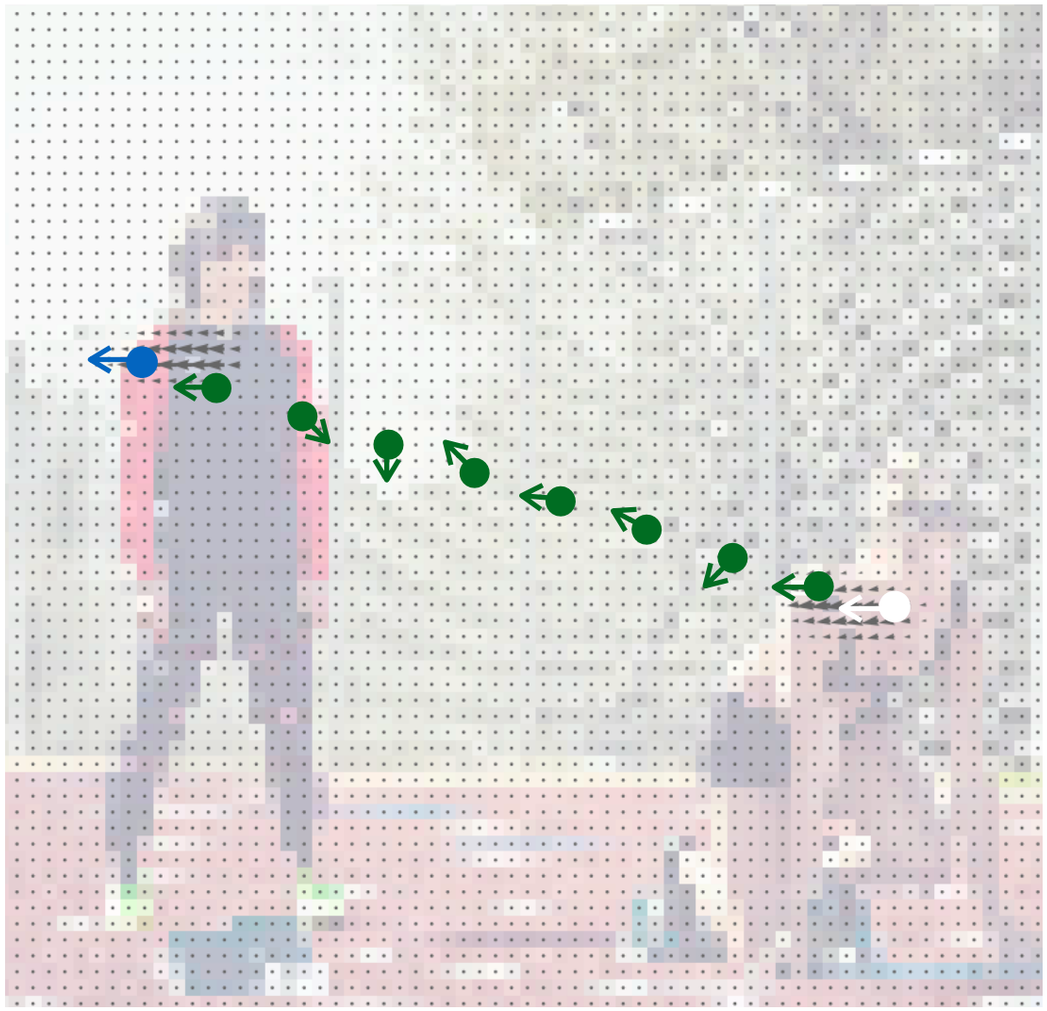


2. Generate the 10 points between the two points



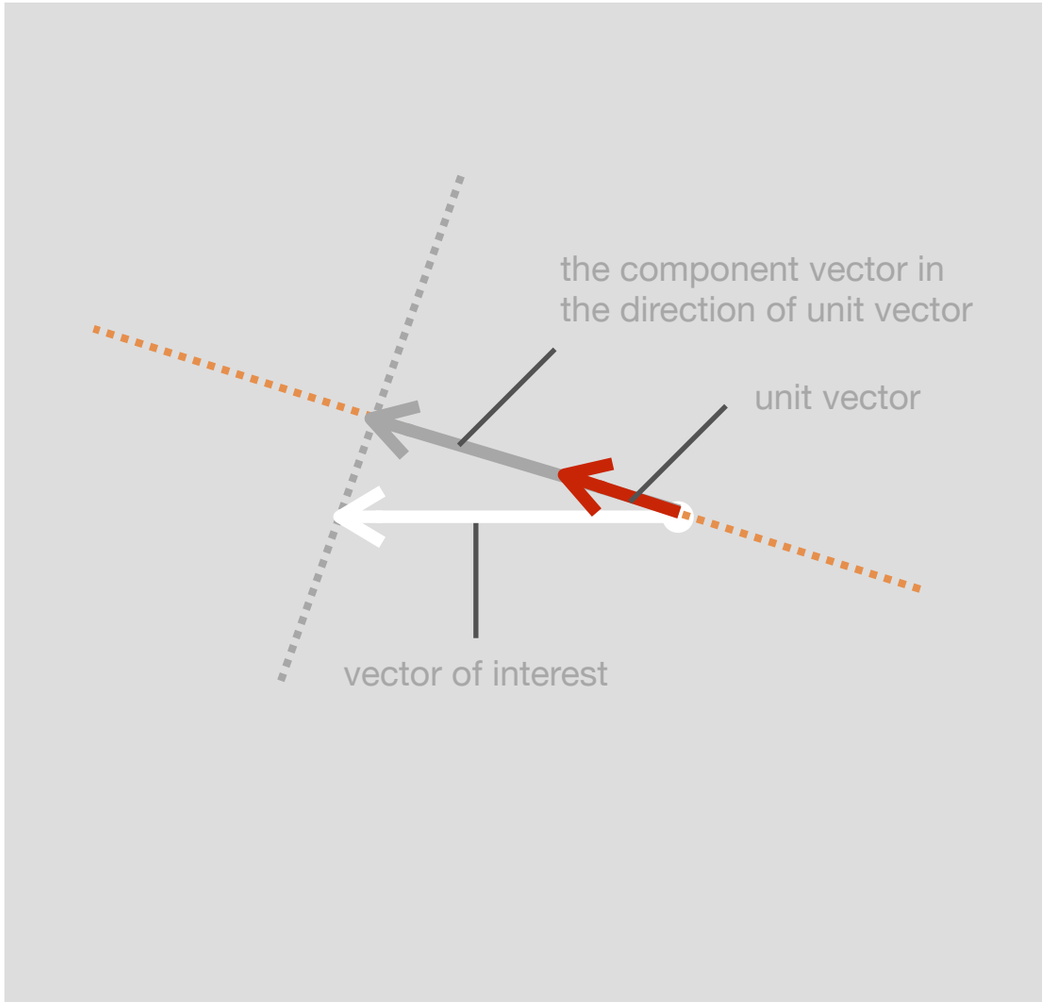
The process
to evaluate a link

3. For each point, get the vector from part affinity field



4. For each vector, perform dot product with the unit vector to determine the strength of the component in the direction of unit vector

The strength of the component is called PAF score



The process
to evaluate a link

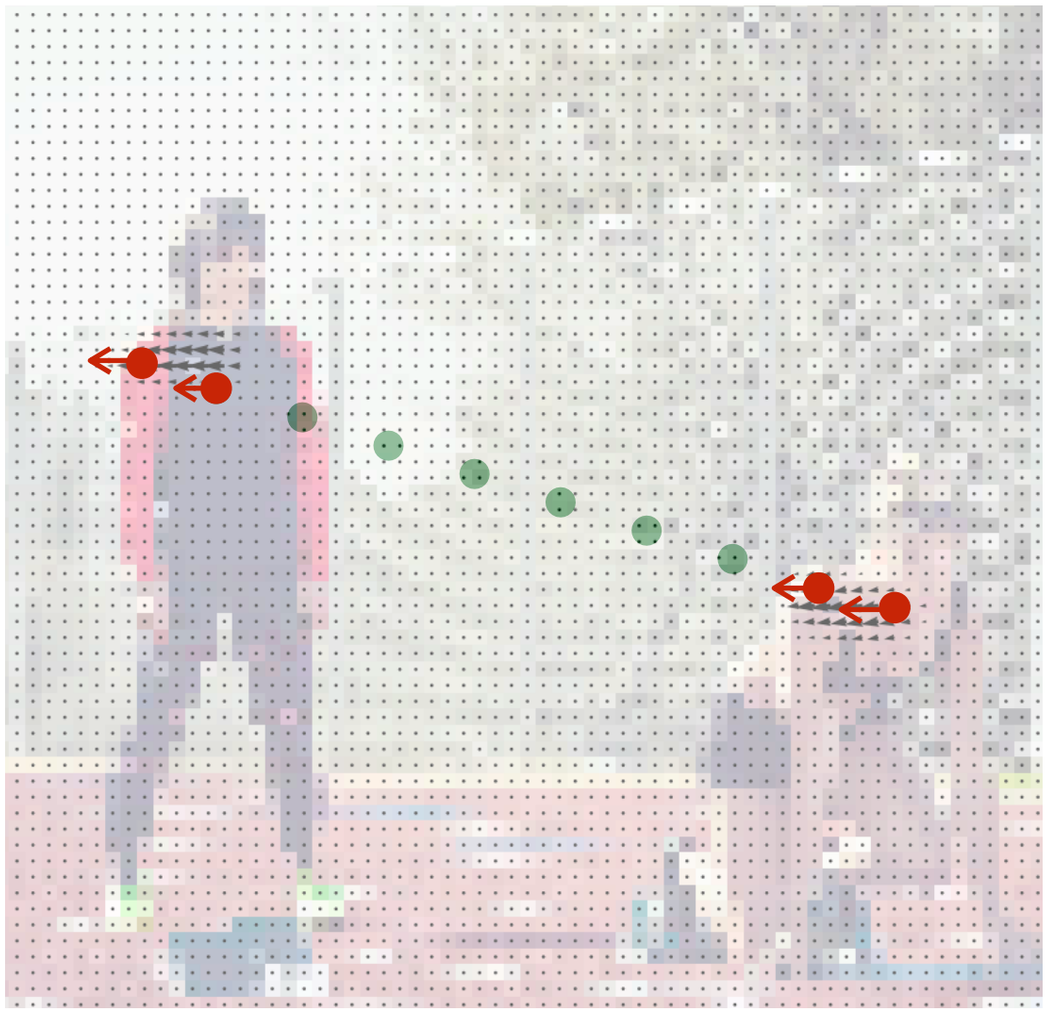
5. Calculate the average PAF score (from the 10 vectors)

6. Determine which vector has a PAF score higher than pafThres (a threshold)

7. If there are more than 7 vectors, each with a PAF score higher than pafThres, this is *likely* a true link (a candidate)

8. If the average PAF score is the highest among the candidates, this is a true link

In this link, there are only 4 vectors that has a PAF score higher than pafThres



getAllLinks

A function to evaluate all the possible links

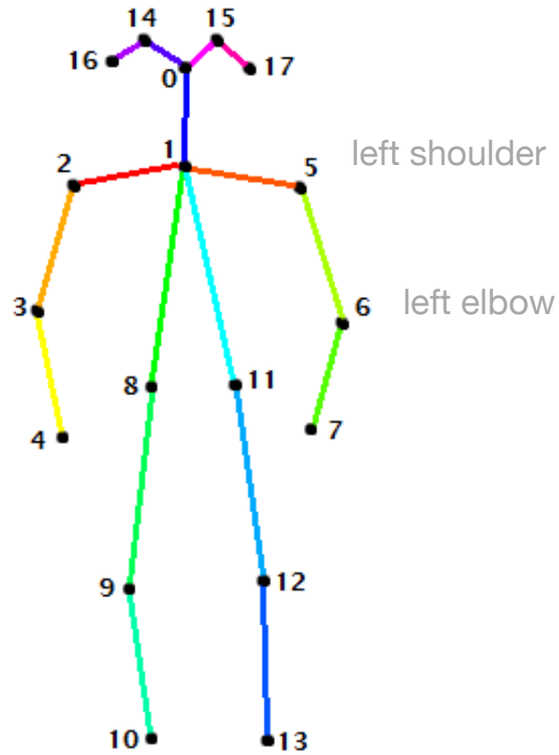
- In the script provided, the function is `getAllLinks`, the below shows the use of the function:

```
> (linksWithPairs,
   linksNoPairs)      = getAllLinks(output,
                                    links,
                                    pafCh,
                                    ptGrp,
                                    (H,W))
```

linksNoPairs

its purpose

```
links       = [[1,2],
               [1,5],
               [2,3],
               [3,4],
               [5,6],
               [6,7],
               [1,8],
               [8,9],
               [9,10],
               [1,11],
               [11,12],
               [12,13],
               [1,0],
               [0,14],
               [14,16],
               [0,15],
               [15,17],
               [2,17],
               [5,16]]
```



- `linksNoPairs` is a list of integer
- It stores integers that corresponding to links not located in an image
- For example, take the link from left shoulder to left elbow.
- If an image has no link between any left shoulder and left elbow, then an integer of 4 (0 stands for the first link) will be appended to `linksNoPairs`

```
> (linksWithPairs,
   linksNoPairs)   = getAllLinks(output,
                                 links,
                                 pafCh,
                                 ptGrp,
                                 (H,W))
```

linksWithPairs

its purpose

- `linksWithPairs` is a list of 19 items, each item is either a numpy array or empty list

- The numpy has a shape of n x 3, where n is the number of pairs found for the type of link

point index point index

`linksWithPairs`

Index	Type	Size	Value
0	float64	(2, 3)	[[2. 4. 0.9145169] [3. 5. 0.79380524 ...
1	float64	(2, 3)	[[2. 10. 0.86870136] [3. 11. 0.78 ...
2	float64	(2, 3)	[[4. 6. 0.85824512] [5. 7. 0.72862764 ...
3	float64	(2, 3)	[[6. 8. 0.89267865] [7. 9. 0.88040276 ...
4	float64	(2, 3)	[[10. 12. 0.948304] [11. 13. 0.92 ...
5	float64	(2, 3)	[[12. 15. 0.8486575] [13. 16. 0.98 ...
6	float64	(2, 3)	[[2. 17. 0.81237098] [3. 18. 0.86 ...
7	float64	(2, 3)	[[17. 19. 0.8756363] [18. 21. 0.86 ...
8	float64	(2, 3)	[[19. 22. 0.68773632] [21. 24. 0.93 ...
9	float64	(2, 3)	[[2. 25. 0.79615942] [3. 26. 0.87 ...
10	float64	(2, 3)	[[25. 27. 0.85914514] [26. 28. 0.74 ...
11	float64	(2, 3)	[[27. 29. 0.64831891] [28. 30. 0.95 ...
12	float64	(2, 3)	[[2. 0. 0.75919725] [3. 1. 0.94778754 ...
13	float64	(2, 3)	[[0. 31. 0.9403746] [1. 32. 0.91 ...
14	float64	(2, 3)	[[31. 35. 0.85877966] [32. 36. 0.79 ...
15	float64	(2, 3)	[[0. 33. 0.6527288] [1. 34. 0.60 ...
16	float64	(1, 3)	[[34. 37. 0.30559434]]
17	list	0	[]
18	list	0	[]

2 pairs located for neck to right shoulder

2 pairs located for neck to left shoulder

1 pair located for left eye to left ear

No pair located for right shoulder to left ear

No pair located for left shoulder to right ear

Step 3, part 2

Form human subjects

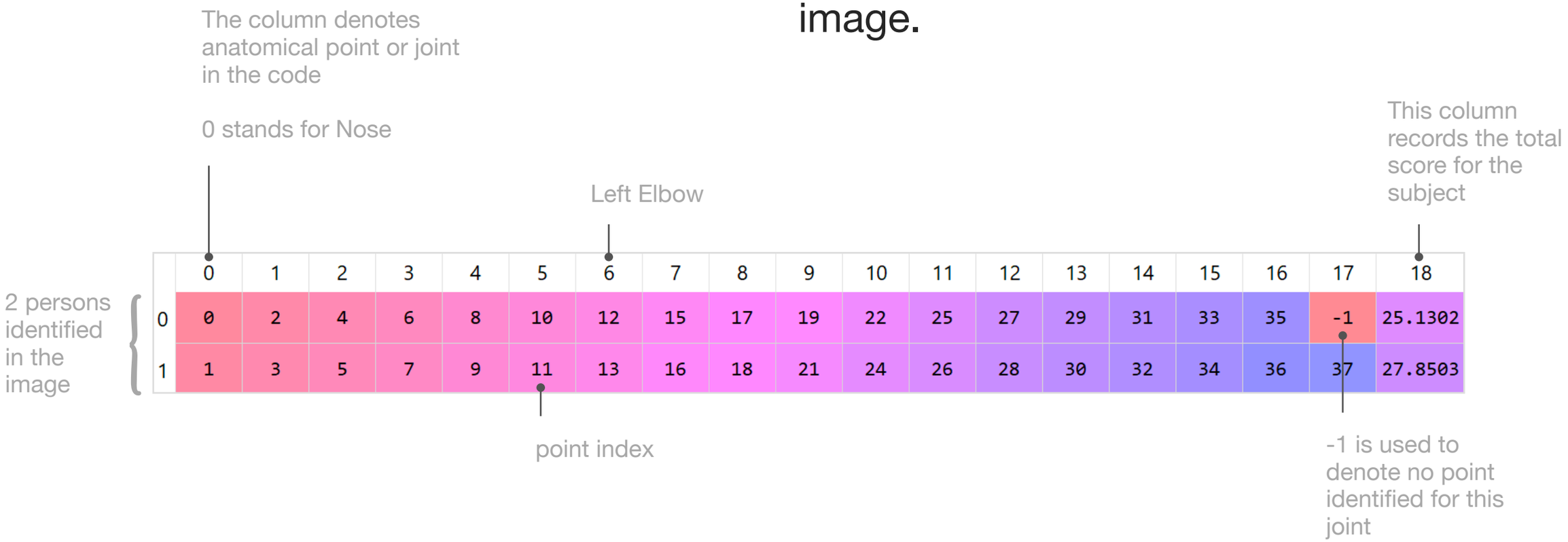
The last part

Get the human subjects

- With all links identified, it is then possible to group links to form human subjects

- The function to perform the task is `getPersons`

- The output `persons` is a n x 19 numpy array. n is the number of human subjects identified in the image.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	0	2	4	6	8	10	12	15	17	19	22	25	27	29	31	33	35	-1	25.1302
1	1	3	5	7	9	11	13	16	18	21	24	26	28	30	32	34	36	37	27.8503

Finale

A function to evaluate all the possible links

- The `persons` can be obtained from this

```
> persons     = getPersons(linksWithPairs,
                           linksNoPairs,
                           links[0:17],
                           ptList)
```

I used only the first 17 types of link to determine person

- In the script provided, the function `drawSkeleton` is created to draw the points and the links:

```
> sklImg    = cv2.cvtColor(img.copy(),
                           cv2.COLOR_BGR2RGB)
> skel      = drawSkeleton(sklImg,
                           links[0:17],
                           persons,
                           ptList,
                 The colour set for links  colours,
                 The colour set for points ptColours)
```

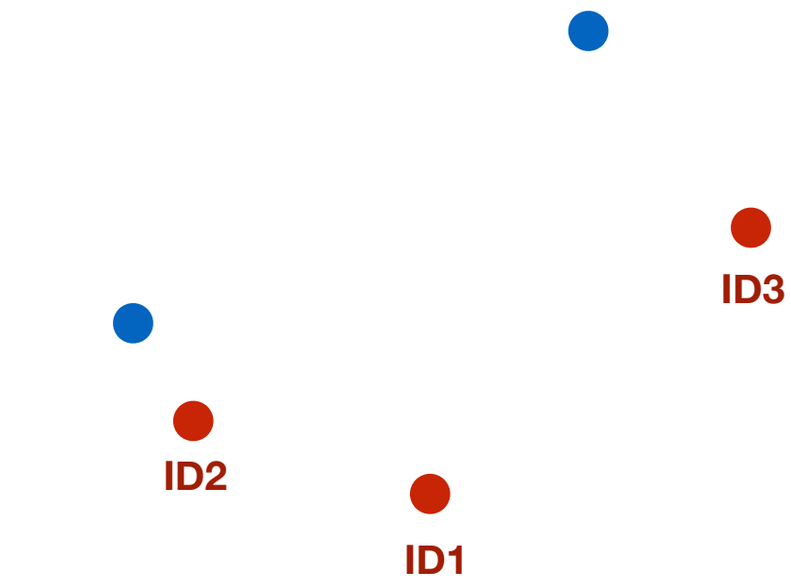
What next?

Person tracking

with OpenPose

Source: https://towardsdatascience.com/fall-detection-using-pose-estimation-a8f7fd77081d

Person tracking
with OpenPose



Red dots are the centroids in frame n
Blue dots are the centroids in frame n+1

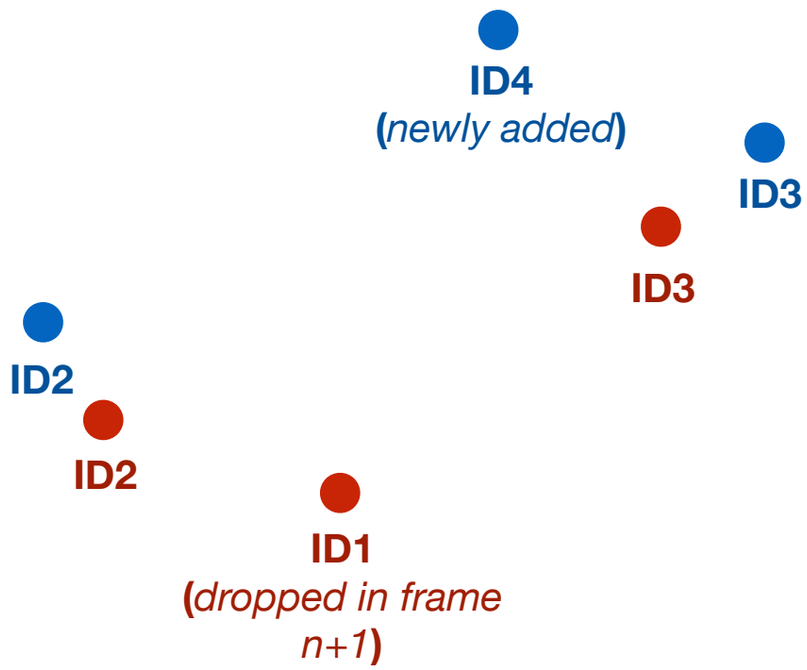
- With OpenPose we can identify every person in a scene / image / frame
- It is possible to perform simple tracking on each person in a video stream
- Assume we are at frame n.
- Step 1: For each identified person, calculate its centroids (or take the neck point as centroid)
- Step 2: For each identified person at frame $n+1$, calculate its centroid (or take the neck point as centroid)
- Step 3: For each centroid in frame $n+1$, calculate its euclidean distance with EVERY centroid in frame n

Person tracking
with OpenPose

- Step 4: Pair the centroids between frame $n+1$ and n that has the minimum distance.

- Step 5: If a centroid in frame n has no match in frame $n+1$, drops the point in frame $n+1$

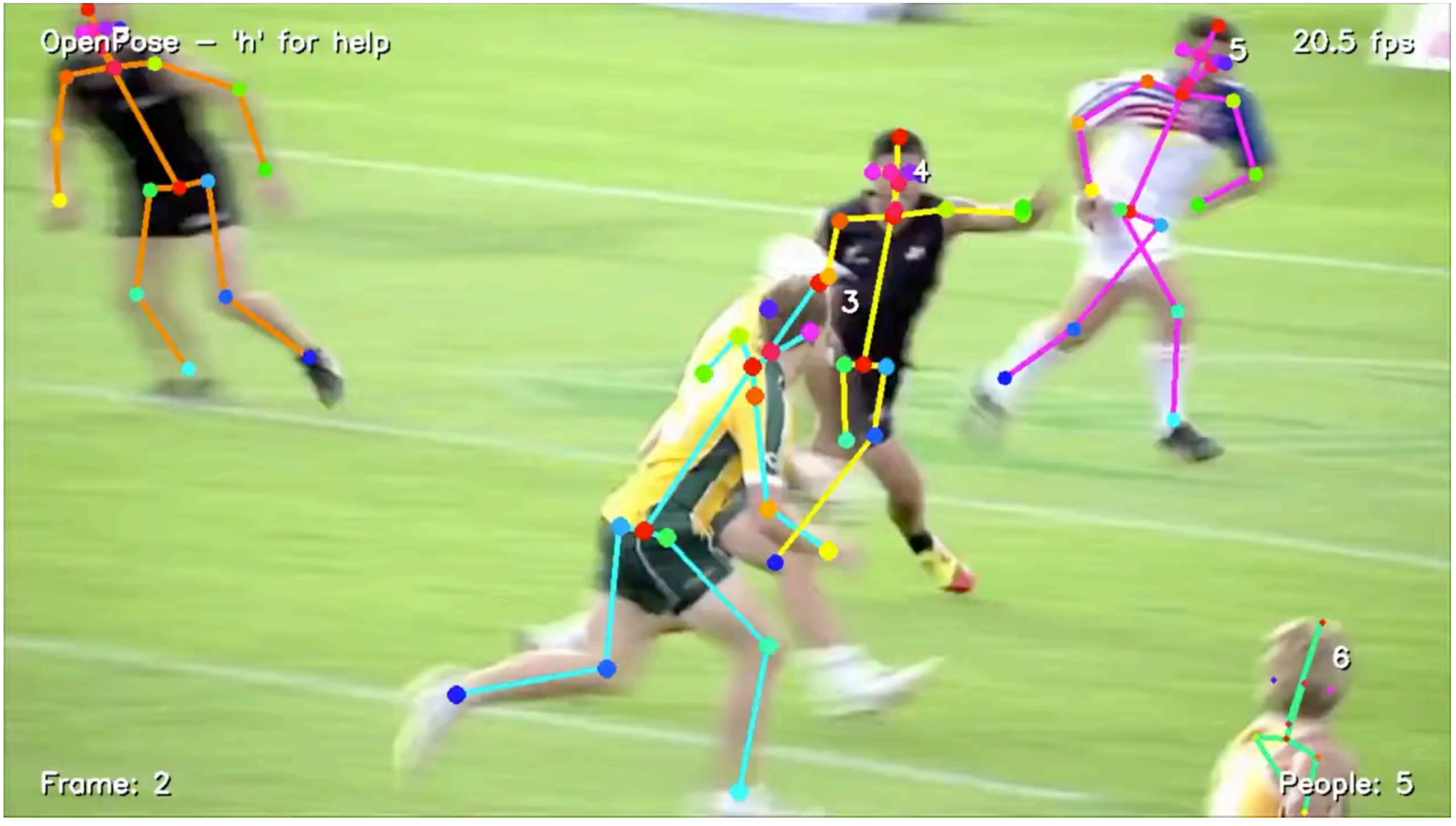
- Step 6: If a centroid in frame $n+1$ has no match in frame n, create a new ID for the centroid.



Red dots are the centroids in frame n
Blue dots are the centroids in frame $n+1$

Temporal affinity field

extended version of part affinity field



Source: https://www.youtube.com/watch?v=1Hg39MVNKBw

Multi-person pose tracking

not just pose estimation, but also tracking

- OpenPose by default does not have the capability for tracking on video. It also can be slow at times on video

- Raaj et al. proposed temporal affinity field to encode connections ***between keypoints across frames***, and because of this setup, the method can do both pose estimation and tracking of key points across frames

- Their body model has 21 key anatomical points, and the input frame size is 368 x368

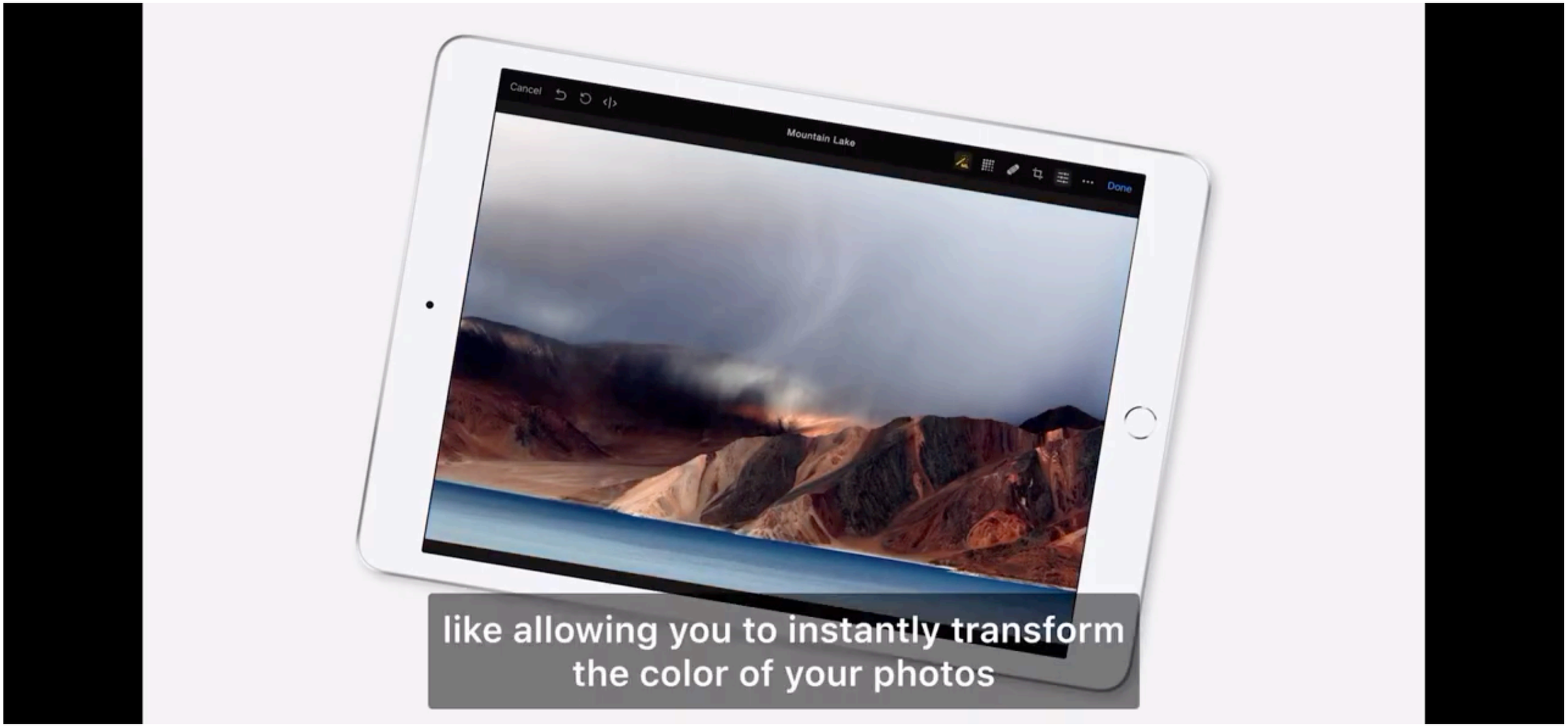
- They have achieved almost 30 fps, but do take note, they wrote their code in C++ and CUDA, tested their code on 1080 Ti, and i7 7800X

Source: https://arxiv.org/pdf/1811.11975.pdf

Pose estimation

In actual use

- Tennis swing analysis by SwingVision using pose estimation

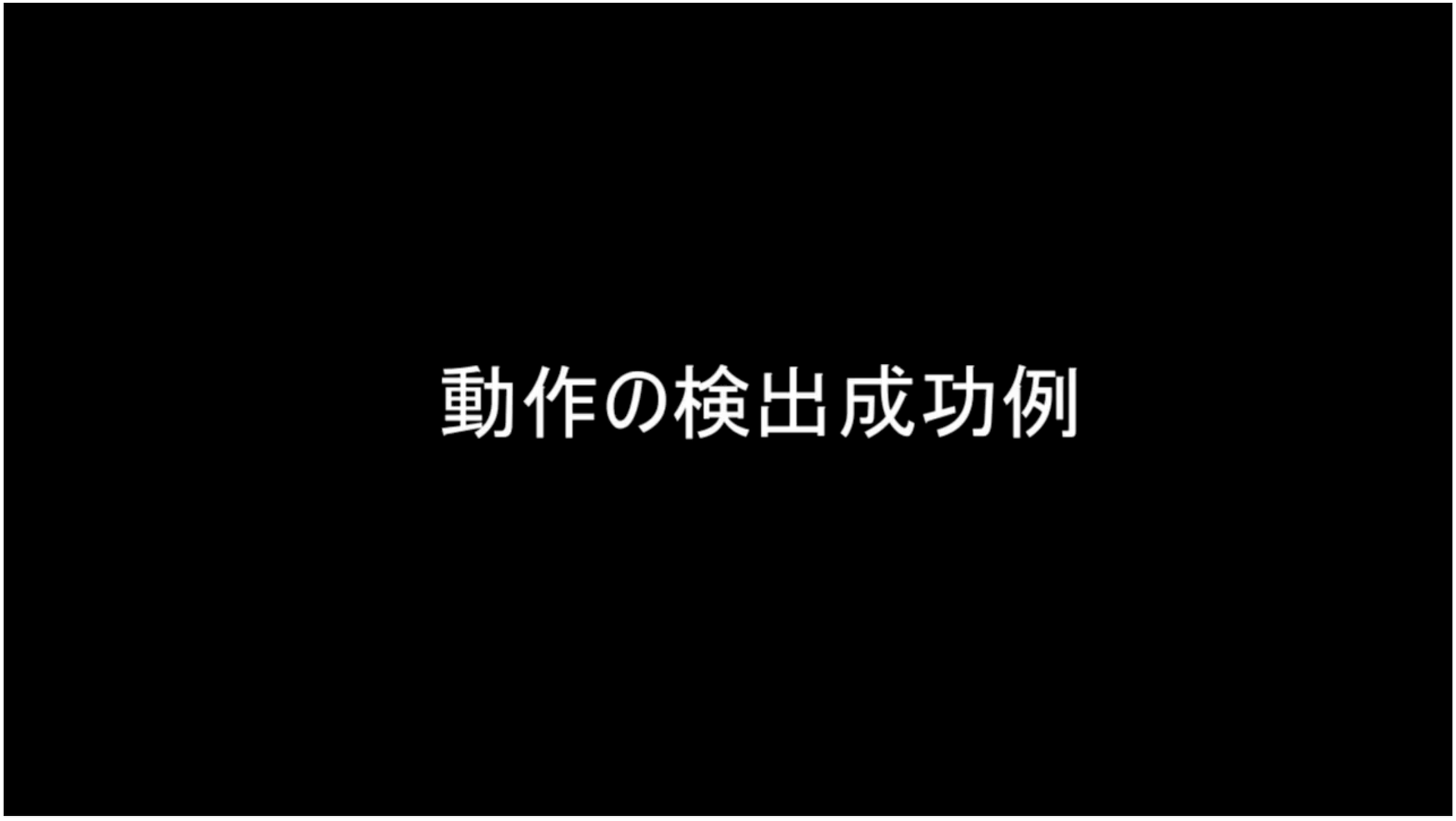


Source: https://www.youtube.com/watch?v=hQkcaoIFZ9U

Pose estimation

In actual use

- Research version of tennis playing identification

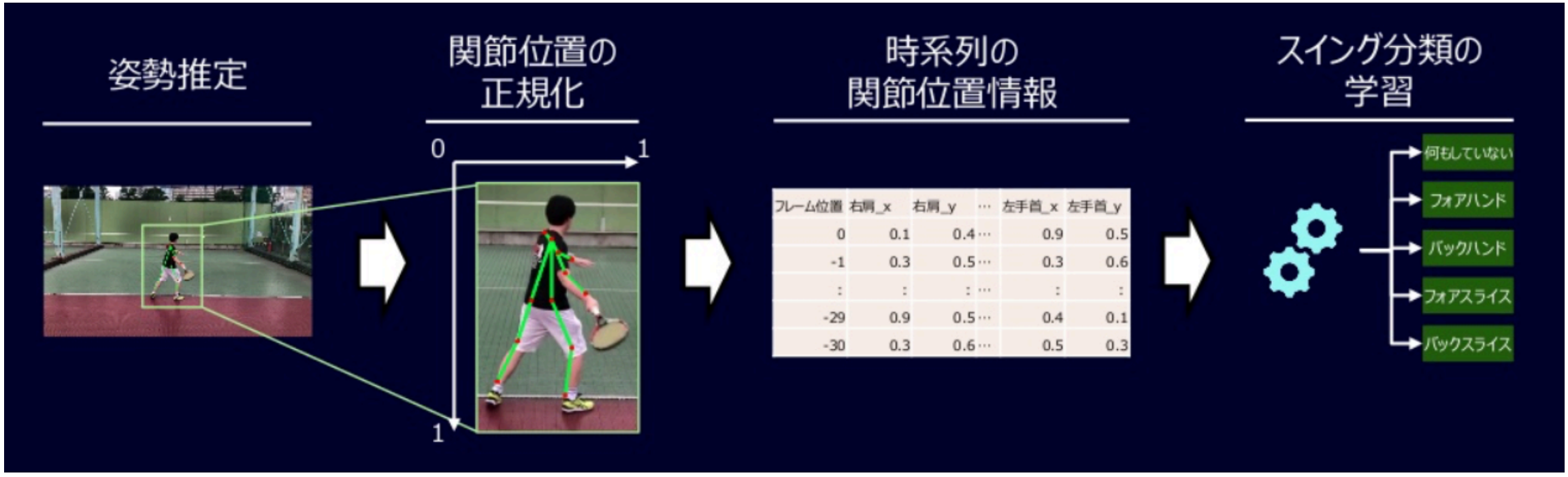


Source: https://www.youtube.com/watch?v=hQkcaoIFZ9U

Pose estimation

In actual use

- Step 1: Perform pose estimation in each frame
- Step 2: Normalize the positions of key anatomical points with respect to the player in each frame
- Step 3: Send the normalized positions from **15 frames** to lightGBM model to classify 5 classes: idle, backhand, forehand, fore slice, back slice



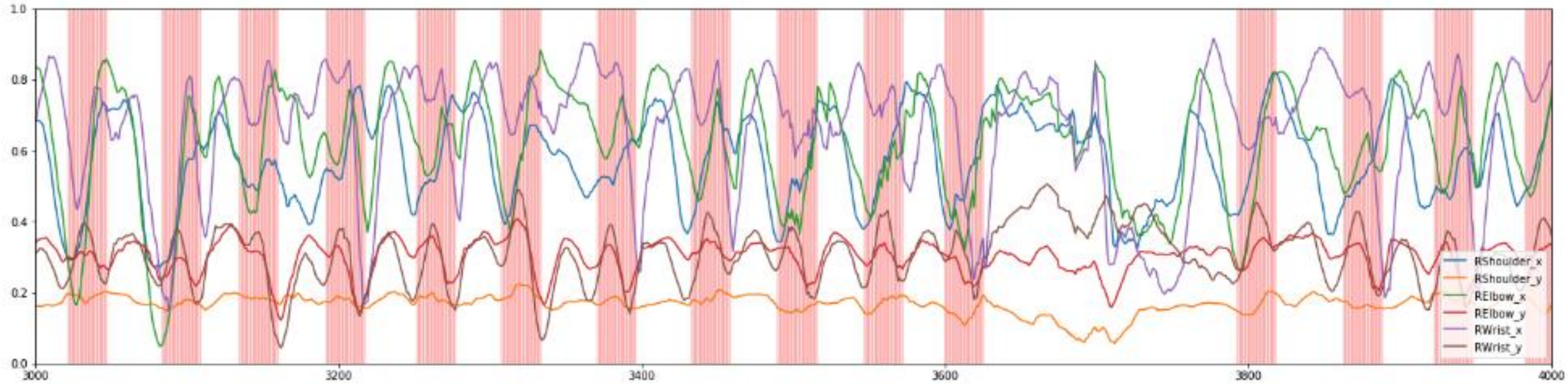
Source: https://www.slideshare.net/YuyaMochimaru/tennis-swing-recognition-based-on-pose-estimation-and-lightgbm

Pose estimation

Visualization of data

- Red shaded regions are the times when forehand playing was performed

- The legend:
  - Right shoulder x-axis
  - Right shoulder y axis
  - Right elbow x-axis
  - Right elbow y-axis
  - Right wrist x-axis
  - Right wrist y-axis



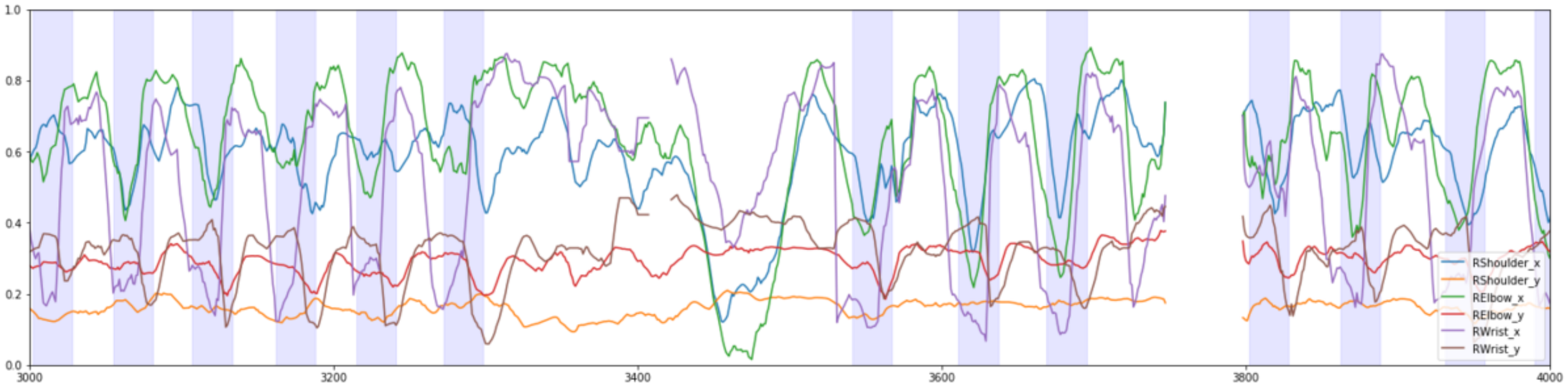
Source: https://hampen2929.hatenablog.com/entry/2019/09/21/202026

Pose estimation

Visualization of data

- Purple shaded regions are the times when back hand playing was performed

- The legend:
  - Right shoulder x-axis
  - Right shoulder y axis
  - Right elbow x-axis
  - Right elbow y-axis
  - Right wrist x-axis
  - Right wrist y-axis



Source: https://hampen2929.hatenablog.com/entry/2019/09/21/202026