Draft

Title:

Yoga Mirror: a yoga teaching system for real-time human motion comparison using 3-dimension weighted dynamic time warping

Highlights:

* **Webcam & Keep user data local:** A new application of PoseNet, created by tensorflow.js Google, generated human pose heatmaps and the outputs of position only using simple webcam(both of browsers and mobile cameras) , which means the user’s data could be processed in the local computer.
* **Similarity score using weighted DTW**: To compare the motional similarity of the output data in teaching video and the real-time data transferred by camera, this paper used dynamic time warping algorithm and weighted it by confidence score.
* Personal suggestions for users based on the real-time performance of similarity score.

Index

Abstract

1. Introduction
2. Related Work
   1. The applications in yoga teaching area
   2. The development of **human pose estimation**
   3. The comparison of similarity using **dynamic time warping**
3. The Architecture Of The System
   1. The interaction between teaching videos and real-time feedback
   2. The skeleton representation of human pose and the the output data
   3. The similarity score using multi-dimension weighted dynamic time warping
4. Experiments And Results
   1. The continuity of real-time data via webcam
   2. The accuracy of similarity score using multi-dimension weighted dynamic time warping
      1. 2-dimension matrix of position x and position y
      2. 3-dimension weighted matrix of positions and confidence score
5. Evaluation
6. Conclusion
7. References

Related Work

<https://getpocket.com/a/read/1972778115>

**The Architecture Of The System**

**Human Pose Estimation Overview**

**The interaction between teaching videos and real-time feedback**

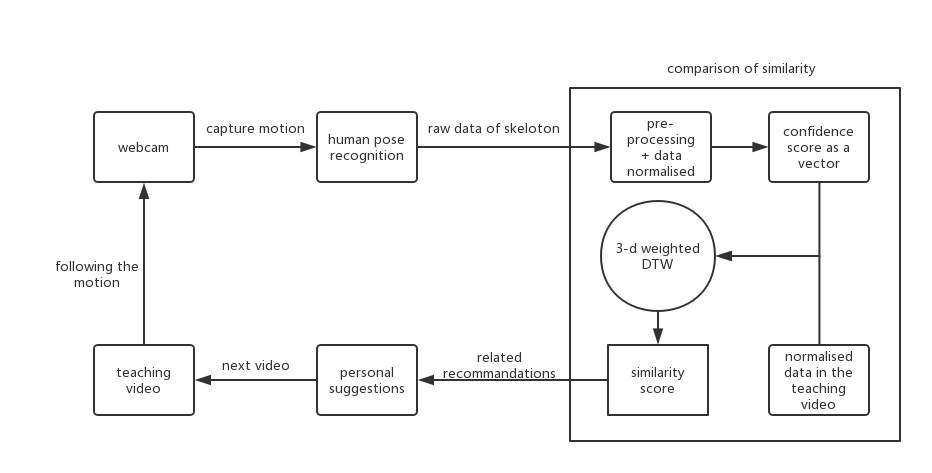


Fig.1 Graphical representation of the interaction between the system and users

The skeleton representation of human pose and the the output data

Openpose vs Posenet

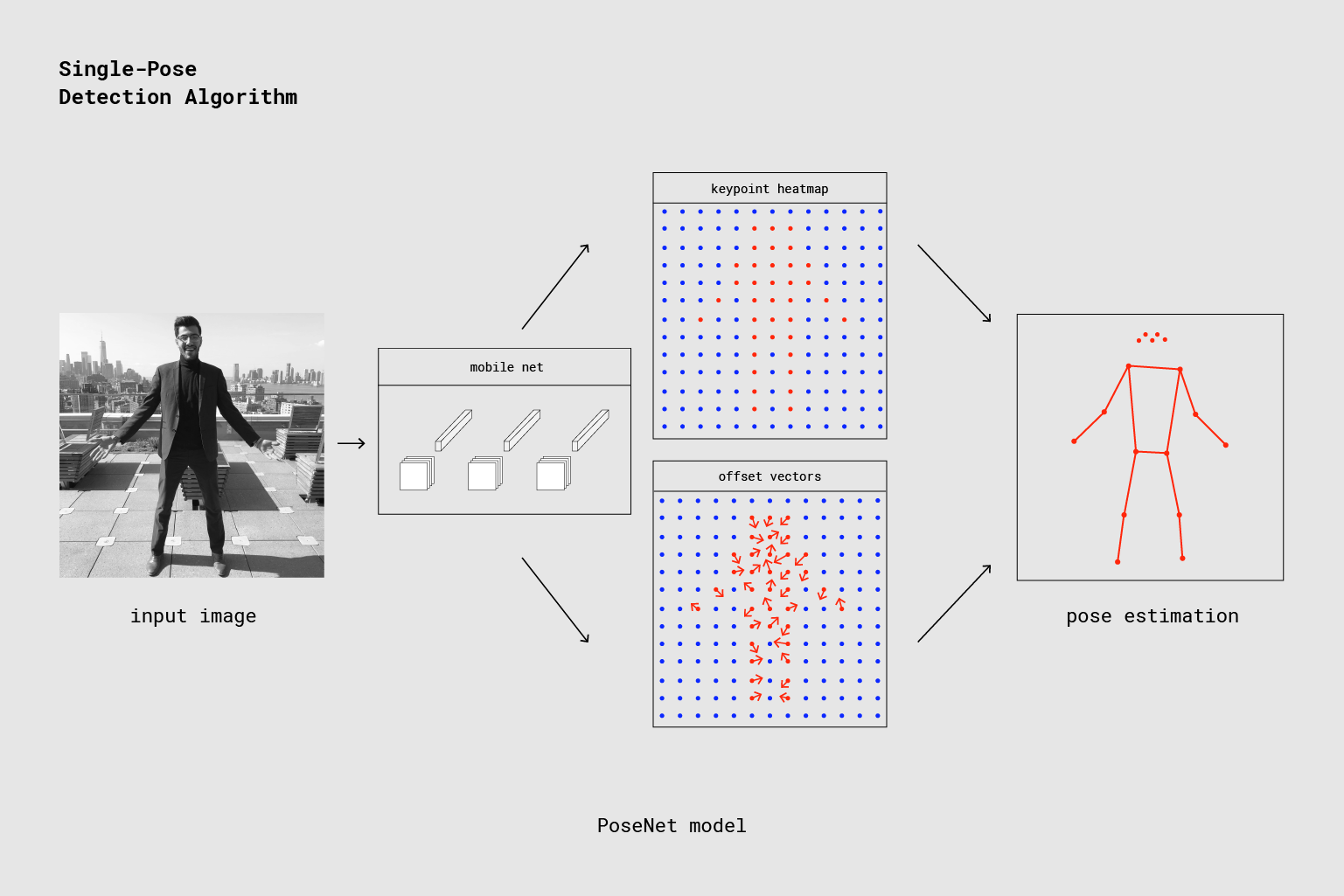
Openpose

*OpenPose represents the first* ***real-time system*** *to jointly detect human body, hand and facial keypoints (in total 130 keypoints) on single images. In addition, the system computational performance on body keypoint estimation is* ***invariant to the number of detected people*** *in the image*

Posenet

<https://medium.com/tensorflow/move-mirror-an-ai-experiment-with-pose-estimation-in-the-browser-using-tensorflow-js-2f7b769f9b23> Move Mirror

<https://medium.com/tensorflow/real-time-human-pose-estimation-in-the-browser-with-tensorflow-js-7dd0bc881cd5>



Single person pose detector pipeline using PoseNet

Output

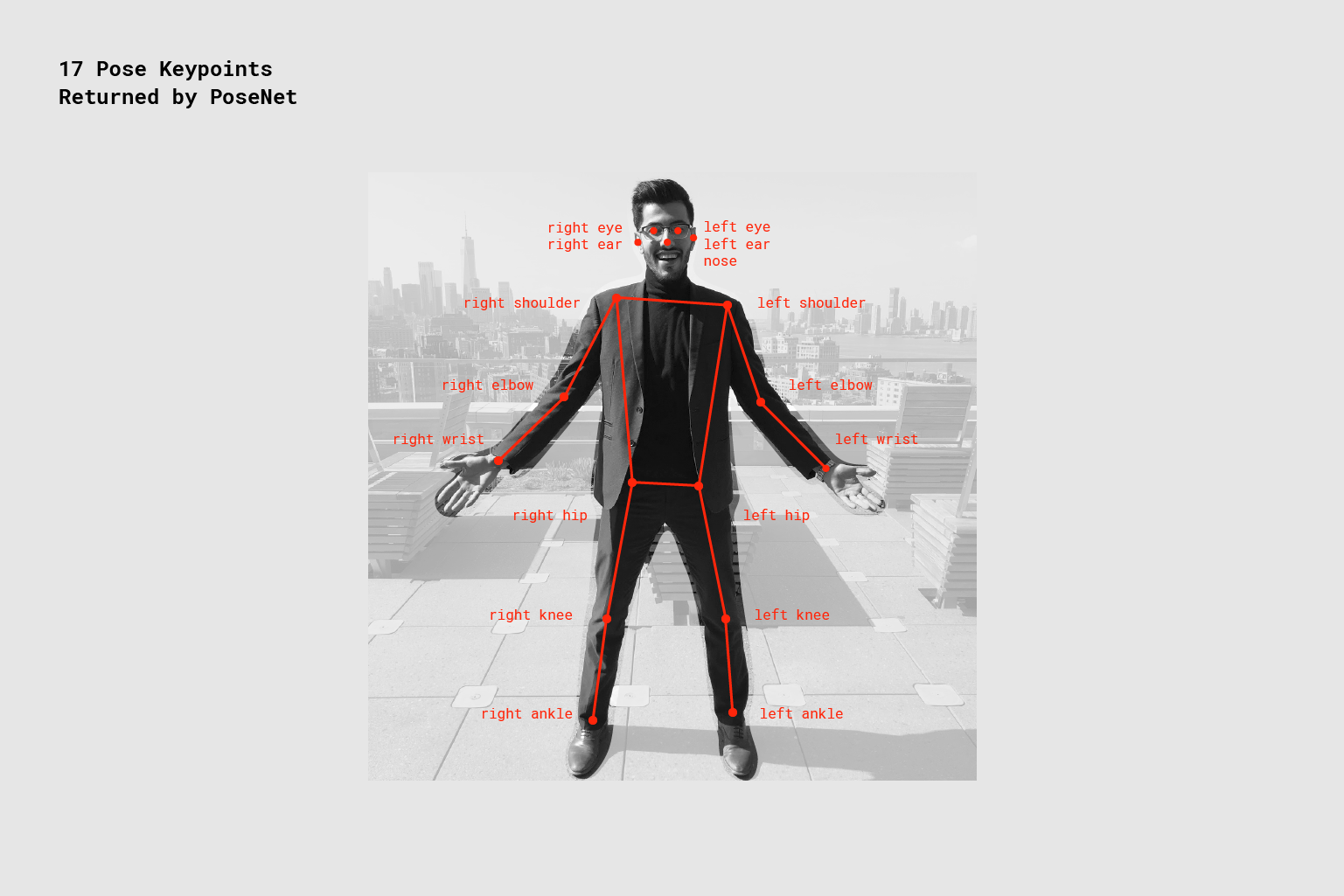


Fig PoseNet detects 17 pose keypoints on the face and body. Each keypoint has three important pieces of data: an (x,y) position (representing the pixel location in the input image where PoseNet found that keypoint) and a confidence score (how confident PoseNet is that it got that guess right).

|  |  |
| --- | --- |
| **Id** | **Part** |
| 0 | nose |
| 1 | leftEye |
| 2 | rightEye |
| 3 | leftEar |
| 4 | rightEar |
| 5 | leftShoulder |
| 6 | rightShoulder |
| 7 | leftElbow |
| 8 | rightElbow |
| 9 | leftWrist |
| 10 | rightWrist |
| 11 | leftHip |
| 12 | rightHip |
| 13 | leftKnee |
| 14 | rightKnee |
| 15 | leftAnkle |
| 16 | rightAnkle |

* **Pose confidence score** — this determines the overall confidence in the estimation of a pose. It ranges between 0.0 and 1.0. It can be used to hide poses that are not deemed strong enough.
* **Keypoint** — a part of a person’s pose that is estimated, such as the nose, right ear, left knee, right foot, etc.It contains both a position and a keypoint confidence score. PoseNet currently detects 17 keypoints illustrated in the following diagram:

|  |
| --- |
| const net = await PoseNet.load(multiplier);  var imageElement = document.getElementById('human');  var imgScaleFacor = 0.5;  var outputStride = 16;  var flipHorizontal = true;  PoseNet.load().then(function(net){  return net.estimateSinglePose(imageElement, imgScaleFacor, flipHorizontal, outputStride)  }).then(function(pose){  console.log(pose);  }) |

**Step 1 Pre-processing data**

So that was our first step: cleaning data from an object to an array.



A snippet of the JSON coming from PoseNet, and a snippet of the flattened array of X and Y positions. (You’ll notice this array doesn’t take confidence into account — we’ll get back to that in a bit!)

**Step 2 Resize, scale and L2 normalize**

because all the images in our dataset can be of different widths/heights, and because each person can appear within a different subset of the image (top left, bottom right, center, etc.), we performed two additional steps to be able to compare the data consistently:

1. Resize and scale: We used each person’s bounding box coordinates to crop and scale each image (and corresponding keypoint coordinates) to a consistent size.
2. Normalize: We further normalized the resulting keypoints coordinates by treating them as an L2 normalized vector array.

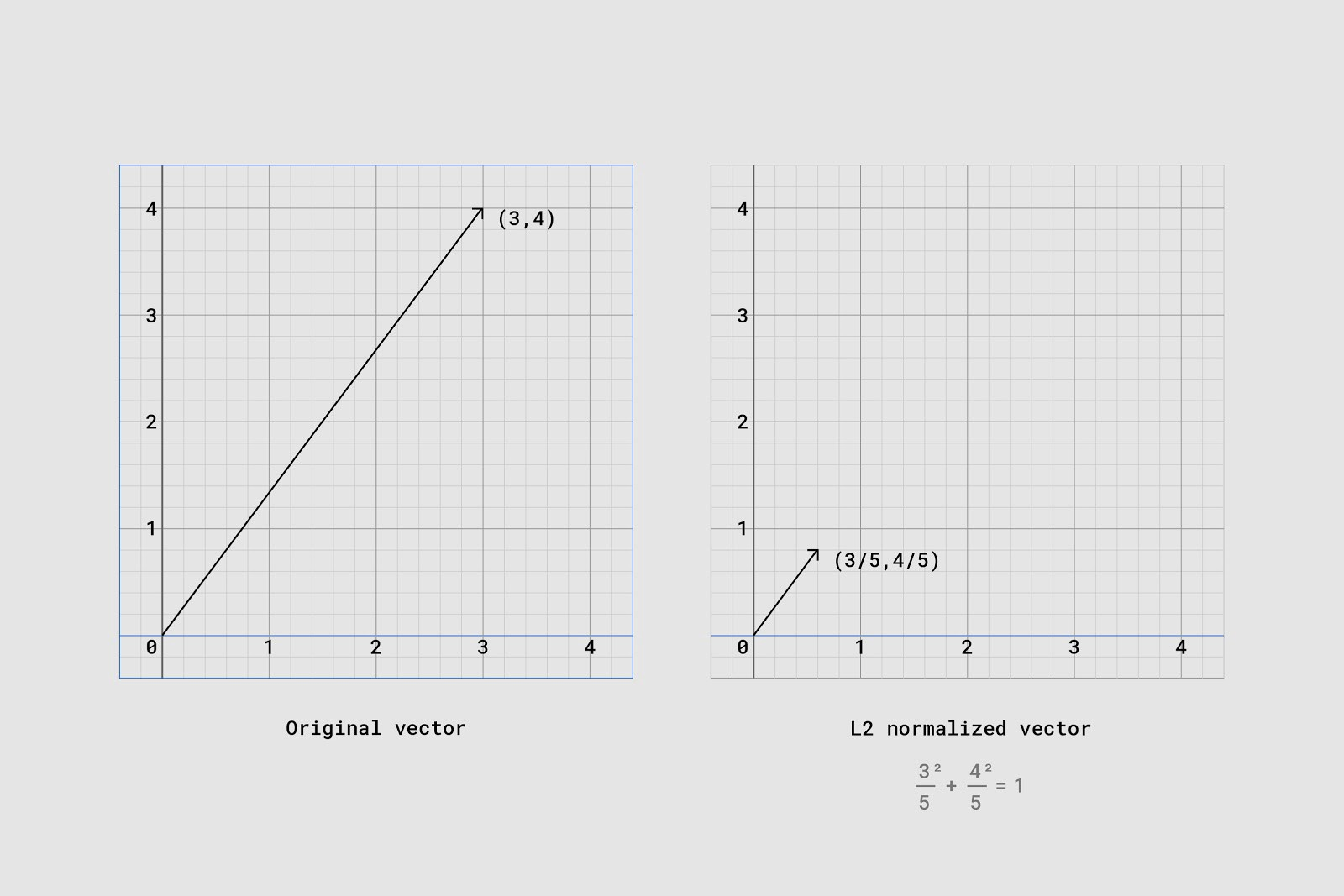


Fig A vector scaled with L2 normalization

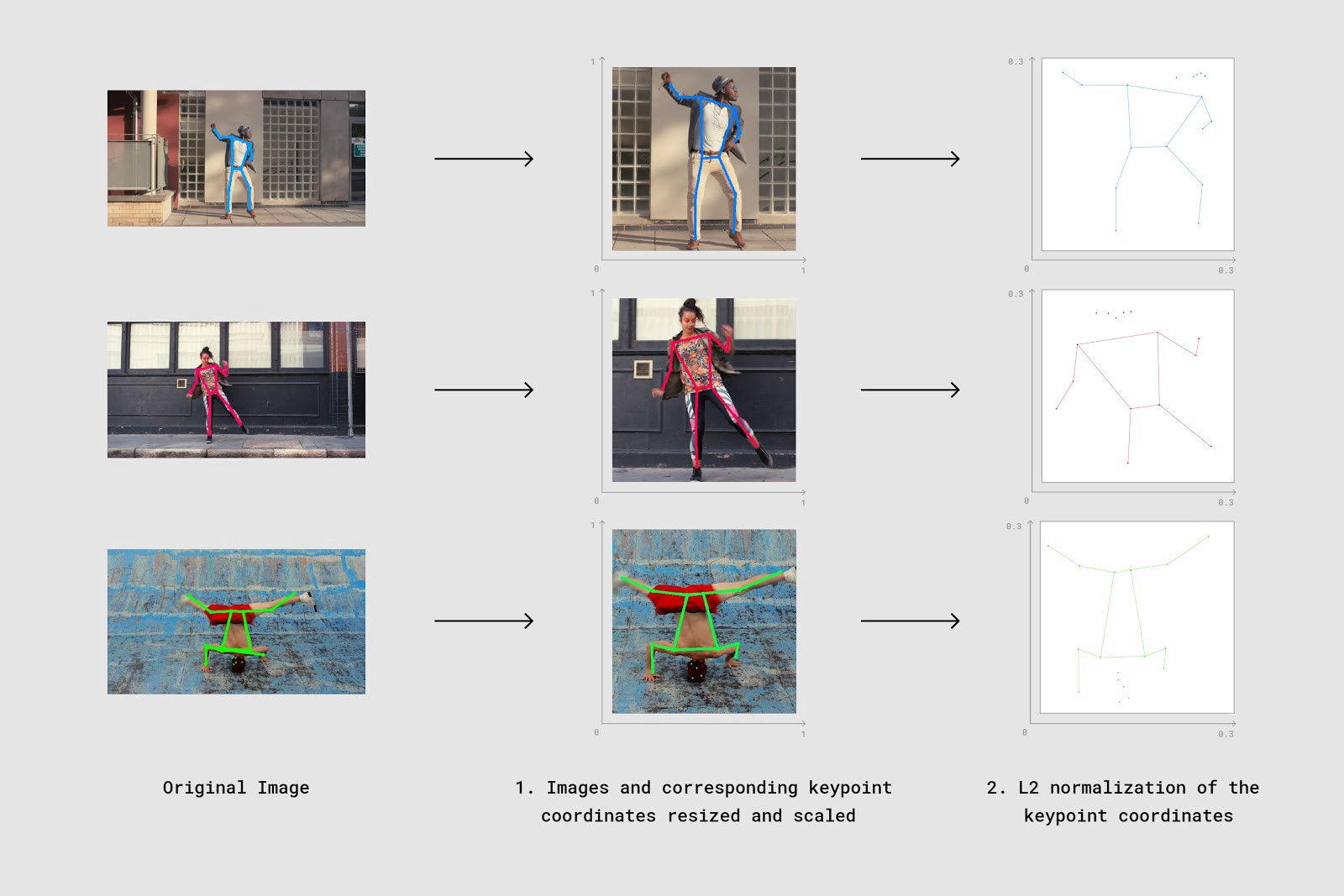


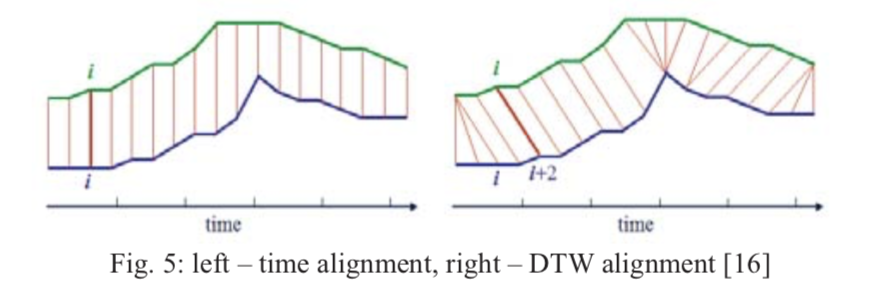
Fig Steps taken to normalized data

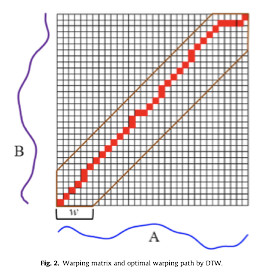
The similarity score using 3-dimension weighted dynamic time warping

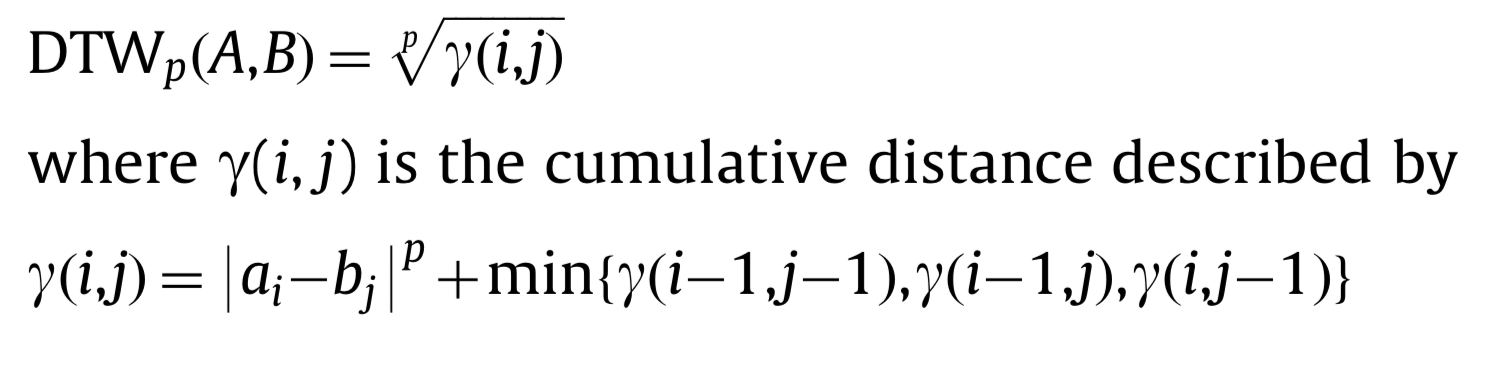
Deciding what ‘similarity’ meant became our first hurdle. How should we decide how similar a set of 17 keypoints in time series from a user is to a set of 17 keypoints from another video? We tried a few different measures for similarity and settled on two that seemed to work well: dynamic time warping and a weighted match taking into account keypoint confidence scores.

#### **How to determine similarity between 2 sets of matching 2D points**

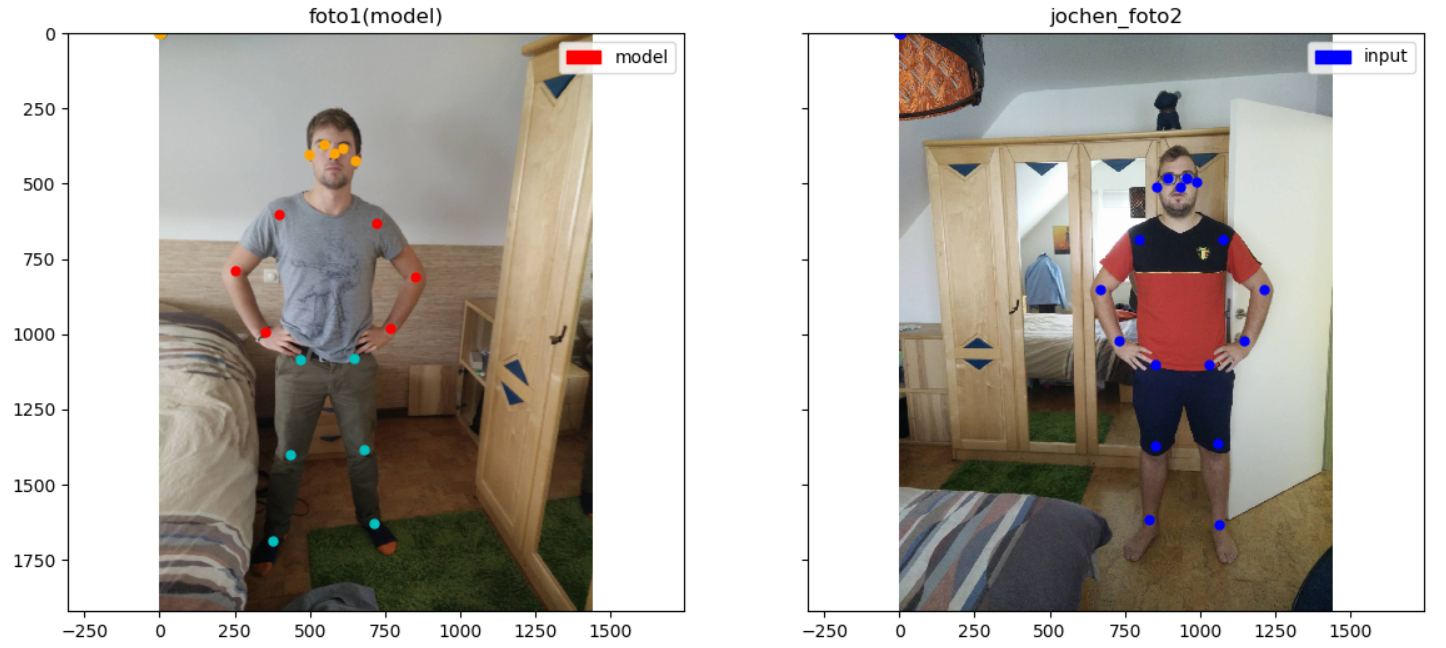
DTW (without confidence score)







|  |
| --- |
| var dtw = new DynamicTimeWarping(video1,video2,function(a,b){  var xDiff = a[0] - b[0];  var yDiff = a[1] - a[1];  return diff = Math.sqrt(xDiff \* xDiff + yDiff \* yDiff);  });  var distance = dtw.getDistance();  var path = dtw.getPath();  var distance = distance/path.length; |

left: model || right: input pose

<https://becominghuman.ai/human-pose-matching-on-mobile-a-fun-application-using-human-pose-estimation-part-1-intro-93c5cbe3a096>

So we have 2 **sets of matching points** (each consisting of 18 2D coordinates);

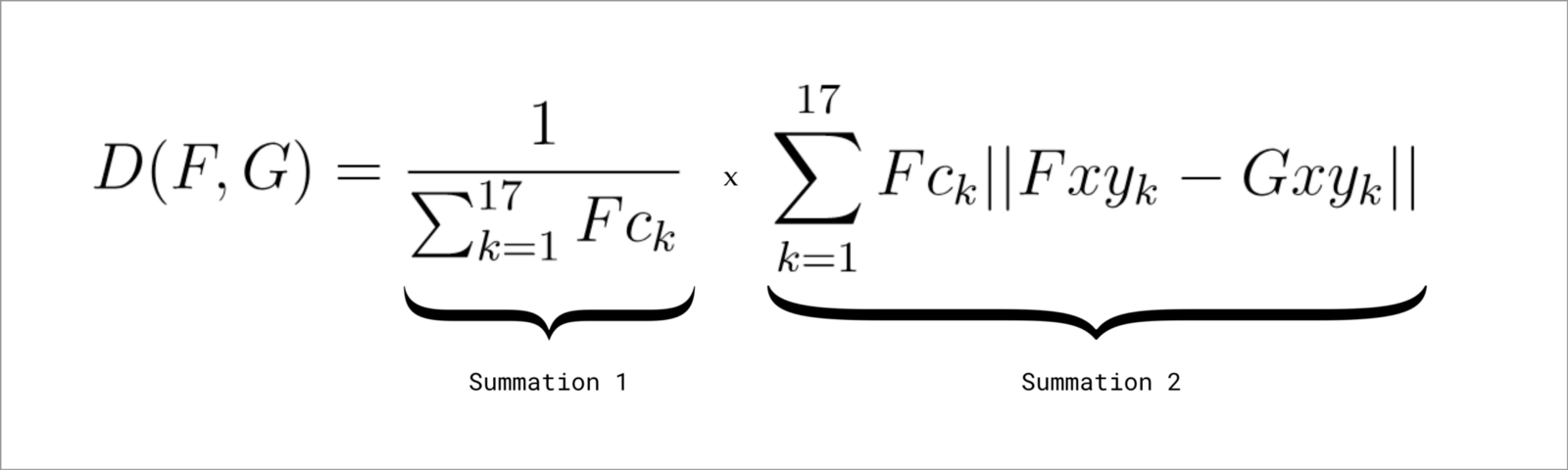
1. The model X (which needs to be mimicked) -left hand side

2. The input Y (needs to be checked on matching grade) -right hand side

Essentially this comes down to checking if the two poses (described by their 18 2D coordinates) are **similar.**

Weighted matching

Posenet returns a confidence score for each keypoint. The model predicts keypoints with a higher confidence score to be more accurate.Posenet returns a confidence score for each keypoint. The model predicts keypoints with a higher confidence score to be more accurate.



(In the formula above, F and G are two pose vectors to be compared after L2 normalization (explained in the previous section). Fck is the confidence score of the kth keypoint of F. Fxy and Gxy represent the x and y positions of the kth keypoint for each vector. )

|  |
| --- |
| // poseVector1 and poseVector2 are 52-float vectors composed of:  // Values 0-33: are x,y coordinates for 17 body parts in alphabetical order  // Values 34-51: are confidence values for each of the 17 body parts in alphabetical order  // Value 51: A sum of all the confidence values  // Again the lower the number, the closer the distance  function weightedDistanceMatching(poseVector1, poseVector2) {  let vector1PoseXY = poseVector1.slice(0, 34);  let vector1Confidences = poseVector1.slice(34, 51);  let vector1ConfidenceSum = poseVector1.slice(51, 52);  let vector2PoseXY = poseVector2.slice(0, 34);  // First summation  let summation1 = 1 / vector1ConfidenceSum;  // Second summation  let summation2 = 0;  for (let i = 0; i < vector1PoseXY.length; i++) {  let tempConf = Math.floor(i / 2);  let tempSum = vector1Confidences[tempConf] \* Math.abs(vector1PoseXY[i] - vector2PoseXY[i]);  summation2 = summation2 + tempSum;  }  return summation1 \* summation2;  } |