Dipartimento di Automatica ed Informatica
Politecnico di Torino, Italy
July, 2021

Nonlinear climbing video indexing

Human pose estimation applied to climbing videos

Daniele Pace

Supervised by
Jaakko Lehtinen (Aalto University)
and
Bartolomeo Montrucchio

Final thesis report for the Master of Science in Computer Engineering,
made in collaboration with Aalto University

Politecnico
di Torino

Aalto University

Asa nisi masa
8 1/2

Abstract

Human Pose Estimation is one of the most promising research areas in the field that links together Artificial Intelligence and Computer Vision. Its applications are countless since it mimics one of the principal human capabilities: understanding the physical space that surrounds us, and the interaction of the context with people. Sport is the field that can most benefit from this technology since it involves dynamic movements of individuals. I present in this project the research and the development of a fully-working application that exploits human pose estimation to perform nonlinear queries to retrieve the exact pose and movements that the user is looking for. I applied this framework to climbing, in which the training phase involves mostly figure out how to perform a certain pose by looking at how other climbers have done it.

Contents

Chapter 1

Introduction

Nonlinear climbing video indexing is an ambitious attempt to apply state-of-art algorithms of Human Pose Estimation to sport videos in order to improve the training phase, in particular to climb training.

The hardest part of the climbing workout is to figure out the right sequence of poses to reach the top hold, and most of the time it means waiting for someone more experienced that shows you how to climb the route. Human Pose Estimation perfectly fits with our needs since it makes the machine able to trace the movement of a (or multiple) person(s) and collect the pose frame by frame. The project consists in the development of a web-based application that lets the user select an area over an image of the wall (usually the area between the holds that you cannot manage to figure out) and a body key-point, and retrieves from a database the part of each videos where the climber has the selected body-parts in the selected area. To perform such nonlinear query, the videos in the database are mapped with one of the most advanced model in Human Pose Estimation, OpenPose, that performs an inference of the body key points and limbs and saves in a Json file the position of key points of each person in the video, frame by frame.

After the first chapter, containing the introduction, chapter number two is a broad review of Human Pose Estimation, from the first research in Computer Vision to the most advanced models; I've reconstructed the history of Human Pose Estimation research, trying to define a logical path (that does

not always follow the chronological one) that ends up with the latest algorithms, impressively close to human capabilities. I divided the review into two sections: before and after deep learning, since such technology marked a turning point in HPE performances. Finally, the last section of the chapter is dedicated to practical use and applications of HPE. The second phase (corresponding to the first two sections of the third chapter) includes some preliminary work done before approaching the practical development. After the review, the OpenPose model was selected for our project. It was developed at Carnegie Mellon University and represents a turning point in multi-people pose estimation. Besides the performances (almost unaffected with multiple people or in case of corrupted/noise videos), it was selected due to the open source code, the license compliant with our scopes, and technical details of the algorithm. Before delving into the development phase, following the Agile approach, my supervisor, Jakko Lehtinen, and I started to think about use cases, defining some user stories that have been used as milestones for the development. Based on user stories I've identified the main features of the application: queries to the database based on space location and body key points, scrubber to carefully explore the video, possibility to make comparison between different videos gotten from the same query. Some initial sketches of the user interface have been drawn in order to visualize how features could be implemented. Preliminary research included the selections of architectural patterns and technological framework to be used as well.

Third step is about data wrangling. Data collection consisted in taking videotapes in the gym, hand-cut in order to have one video for each climb from the beginning to the top, and mapping of videos with OpenPose. Output of OpenPose is a collection of Json files, one for each frame. With the mean of a custom script, I clustered them in a single Json file (one Json file for each video), defining a dictionary with the number of frames as key and an array with the coordinates of all 25 points found by OpenPose as value. In case of multiple people in the frame, the values would be an array of arrays. The entire process is described in the second section of the third chapter.

Before the development of the final application, I built a prototype that helped me in testing core functionalities of the application, such as how accurate videos were retrieved with respect to user intention and how to display all retrieved videos, and clarify some features of the application in real-case scenarios. It was built entirely in Python, as a single-page application. The third section of chapter number three describes the development of the prototype and the conclusions that came after it. Thanks to this test I verified that the videos were correctly mapped and retrieved, a selection of more than one key point is preferable, the retrieved videos must also include some frames immediately before the selected area.

Last two sections of chapter three are about the real application. It was designed with a client-server architecture, in order to execute a secure access to the data and move the most expensive computational part to the server (and not on the client device). Data is exchanged through REST Apis, moving almost all the logic parts in the server. The front-end is organized into three different pages: the first one lets the user choose one among all the "views" (i.e. walls) of the gym; in the second page the chosen view is displayed enlarged: here the user can select an area on the image. Moreover, there is a stylized skeleton that allows the user to select one or more key points. When the user has selected an area and at least one key point, he can send the request to the server. The last page gets all the videos retrieved by the server, and shows them in a vertical scroll bar on the right. The user can select one of the videos, and it will be displayed on the left side, with a scrubber that lets the user move back and forth on the video, carefully analyzing the video frame by frame.

Through a REST Api, the server receives a list of key points, the selected view, and the coordinates of the selected area. A function checks, for each video belonging to that specific view (i.e. for each Json file), if at least one climber has the selected key points in the chosen area. Once found, it collects

in an array the starting and the ending frames of the part of the video that the user is looking for. After having checked each json, it sends the array back to the front-end, which is displayed as a thumbnail the first frame of the retrieved part.

Chapter 2

Human Pose Estimation

Human pose estimation is the problem of localization of human joints, i.e. key-points that define the body structure, in images (monocular RGB) and videos.

It is a critical task in Computer Vision, due to its countless applications in real word scenario, that make this technology one of the most appeal for both commercial and research aim; real-cases scenario that benefits from this technology are, among all, systems to detect if a person has fallen down or sick, understanding of full-body sign language, motion capture for augmenting reality, motion tracking for console (without the support of any particular hardware but a simple camera).

In the face of its versatility in use, it has been largely adressed by researchers, but it is considered one of the hardest task in Computer Vision. In particular, the complexity of human body articulations and degrees of freedom of human movement, occlusions, small and barely visible joints, clothing, changes in lighting due to the complexity of environment, require the algorithm to understand the environment, the context that surrounds the person, in order to figure out the right guess among the huge number of possibilities, once discovered the (most) evident joints.

The first part of this chapter is an ambitious attempt to reconstruct the path of human pose estimation from the beginning, highlighting the most significant research and milestone algorithms in the field. For ease of treatment, I introduce some categories with which to organize the whole. Firstly, human

Figure 2.1: Example of human pose estimation algorithms' output.

pose estimation can be split into two main groups: even if both of them look at 2D images, there are models whose aims is to predict the 3D structure of the pose, and others that track the movement (fig. 2.1), without defining the whole 3D structure. I chose to investigate the latter, because of thesis purposes. A second, important split can be made considering the number of people that the algorithm can detect in the same image at a time (without augmenting the computation time). Almost all discussed algorithms follow the single-person approach; multi-people estimation is introduced in the last part of the deep learning section in human pose estimation, along with most recent researches. Inside multi-people pose estimation works two main approaches are identified: top-down, i.e. starting from the human detection and moving down toward single key-points, and bottom-up, where the estimation starts from single key-points moving upward in defining the whole body.

It is worthy to mention the revolution that artificial neural networks have brought to pose estimation when it first appeared in human pose estimation research around 2014, leading to incomparable results in both efficiency and accuracy. Accordingly, all works about human pose estimation can be chronologically and logically divided into before and after deep learning. Section 2.2 is about classical approaches; section 2.3 deals with most recent techniques based on deep learning.

The last part of the chapter is dedicated to the applications of human pose estimation and an overview of its current implementation in commercial/non-commercial systems, focusing on the practical impact that they have had.

2.1 Background

Human Pose Estimation history begins in the '90 when a new application domain of computer vision had been starting to emerge in some specialized research centers. The extensive coverage in the vision literature is apparent from the many special workshops devoted to this topic: the Looking at People workshop in Chambery (1994), the Motion of Non-Rigid and Articulated Objects workshop in Austin (1994), and the two Automatic Face and Gesture Recognition workshops in Zurich (1995) and Killington (1996). This domain (frequently called "looking at people") was a comprehensive approach that included face recognition, hand gesture recognition and whole-body tracking. Applications of such tasks were many and revolutionary: from never-seen-before human-machine interactions and a machine capable of perceiving the environment just by "seeing", to movement tracking for animation or security purposes.

Traditionally, human movement has been the object of study for several disciplines, and human pose estimation at its very beginning drew on these different fields.

In psychology, starting around 70s, human perception of biological motion pattern was studied by Johansson [1], that with his experiment pointed out how humans can instantly recognize biological pose and motion pattern (like walking or dancing) or detect some small deviations from standard pattern. This study analyzed for the very first time how the motion of some points (human body joints) in a rather irregular way can be easily recognized and associated with a well known human motion pattern.

In biomechanics, scientists have investigated how our body functions mechanically. A typical procedure involves obtaining 3-D joint data, performing kinematic analysis, and computing the corresponding forces and torques for a movement of interest [2].

Computer Graphic had already raised the question about spatial interactions between human models [3], and in Computer Vision Martin A. Fischler and Robert A. Elschlager introduced in 1973 the concept of Pictorial Structures: a statistical model for object that can be used to recognize it and its

constituent part in an image [4]. They were trying to address the task that "given some description of a visual object, find that object in an actual photograph". The object to be identified is composed of a number of rigid pieces held together by "springs" (fig. 2.2). The springs joining the rigid pieces serve both to constrain relative movement and to measure the "cost" of the movement by how much they are "stretched". The semantic information, which is application dependent, is embodied in the specific partitioning of the reference into coherent pieces, the placement and cost functions assigned to the springs, and the cost functions associeted with the independent embedding of the coherent pieces. The syntactic information, which is relatively independent of the particular application, defines the class of description which the algorithm can process.

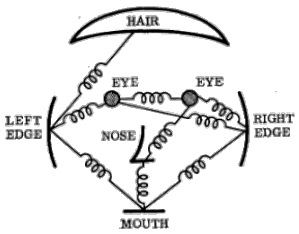


Figure 2.2: Schematic representation of face reference, indicating components and their linkages.

The pictorial framework has become the standard approach for human pose estimation both in classical approaches and in deep learning based ones.

2.2 Classical approaches

Classical approaches follow the Pictorial Structures, even if there have been some experiments without explicitly model the shape of the object to be recognized.

One general approach without shape models has been to bypass a pose recovery step altogether and to describe human movement in terms of simple low-level, 2-D features from the region of interest. An example is a system developed for interpreting the motion in American Sign Language (ASL) [5]; the process involved detection of hand motion, tracking the hand location based on the motion and classification of signs using adaptive clustering of stop position, simple shape of trajectory, matching of the hand shape at the

stop position. Nearest neighbor criteria have been used to estimate posture in reduced-model gesture input system [6], along with general pattern matching. Nelson and Polana investigated in nineties motion recognition through robust motion features [7]. The framework was used to recognized activity based on object motion, using temporal textures of a-priori known activities: motion detectors were used to locate and track moving objects while Fourier image analysis was used to identify objects that moved periodically and matched activity patterns. All the techniques were basically based on the exploitation of a priori knowledge of human motion patterns, therefore very limited for real-cases scenario.

The most popular approach made extensive use of the "Pictorial Structure Framework", whose origins lay on the publication by A. Fischler and Robert A. Elschlager discussed in the previous section [4]. The basic idea of representing an object as a collection of parts arranged in a deformable configuration was further developed in "Pictorial Structures for Object Recognition" paper[8], where the appearance of each part is modeled separately, and the deformable configuration is represented by spring-like connections between pairs of parts. Concerning the 1973 model, the paper's authors addressed some previous limitations such as efficiency, the huge number of parameters and the limited number (one) of best (minimum energy) match. Efficiency was improved thanks to a restriction that made the computation doable in polynomial time while preserving the accuracy: the graph modeling the relationships between parts can be shaped as a tree structure. Even though this assumption is not natural to the original Pictorial framework, had been observed that many classes of objects followed on a tree-shaped model (such as the human skeleton). Moreover, this restriction reduces the number of parameters in the matching algorithms. The limited number of best matching (in their original work, Fischler and Elschlager only considered the problem of finding the best match of a pictorial structure model to an image) was handled by using a statistical framework that provided a natural way of finding several good matches of a model to an image. Besides, the use of a statistical

formulation provides a way of learning pictorial structure models, given a priori knowledge of human body structure, unlike previous works that provided a manual construction of the models (fig. 2.3) .

Figure 2.3: Inputimage, binaryimage, random sample from the posterior distribution of configurations.

The combination of tree-structured model and the use of machine learning to learn part appearances was the dominant approach for subsequent works and algorithms.

Iterative parsing was proposed by Deva Ramanan [9]. A soft estimation of body part positions is done with matching on an edge-based deformable model, then the estimated body part positions are used to build a rough region model for each body part. The algorithm then builds a region-based deformable model for body parts learning some differences (such as the color) with respect to the background. Soft estimates of body position from the new model are then used to build new region models, and the process is repeated. The probabilistic model is a tree-structured conditional random field (CRF) and the basic machinery used for inference is message-passing (the sum-product algorithm) following the tree-structure. Moreover, the authors of the paper built up one of the first (for the time) large datasets for human pose estimation in a collection of 305 images, taking data from limited previous sports datasets and personal pictures.

The iterative image parsing method deeply influenced subsequent works, and we can find traces of their impact in advanced HPE algorithms such as the Hourglass Stacked Network [31].

An enhancement of the pictorial structure was made by Sam Johnson and Mark Everingham [11]: they addressed the problem of articulated 2-D human

pose estimation combining the pictorial structure with strong limb detectors, making an extensive use of gradients and color segmentation. HOG (Histogram of Oriented Gradients, fig. 2.4) descriptor, capturing local appearance such as edge and shading while incorporating a controlled level of invariance to local deformation, is used for capturing limb appearance and, combined with segmentation cues for exploiting coherent properties of the limbs such as uniform color, were proved to reach the state-of-the-art on the "Iterative Image Parsing" (IIP) dataset at the end the last decade (2009).

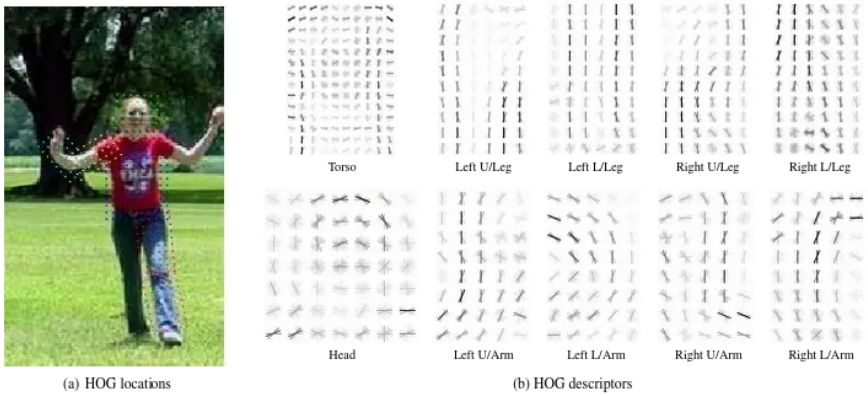


Figure 2.4: HOG descriptors for limb detection.

HOG descriptors were used by Kumal et al. as well [10], proposing an efficient discriminative learning of part-based model, exploiting highly optimized SVM-Light algorithm.

In 2010 a new annotated database of challenging consumer images, an order of magnitude larger than other available, was introduced in order to overcome the lack of training data [13]. Thanks to it, its authors reach the state-of-art in pose estimation using a PSM based model [14]: nonlinear SVM classifier (used with a cascade reduced set machine formulation to reduce the computational expense [12]) for learning the appearance terms among clusters (fig. 2.6) got by partitioning the space of human pose.

It's clear the birectional development that research about Human Pose Estimation embraced: on one side the use of machine learning to automatically learn appearance terms and posible poses and their likelihoods, on the other

side the embedding of a-priori known information to given constraints to the model.

Pictorial Structure Model was significantly augmented by the work of Yang and Deva Ramanan [15], which proposed a new representation of deformable parts models using a mixture of small, non-oriented parts (fig. 2.5), in contrast to warped templates used for modeling articulations. Mixture-of-parts model an exponentially large set of global mixtures, but not all such combinations are equally likely: the model learn notions of local-rigidity and other priors about mixture co-occurences.

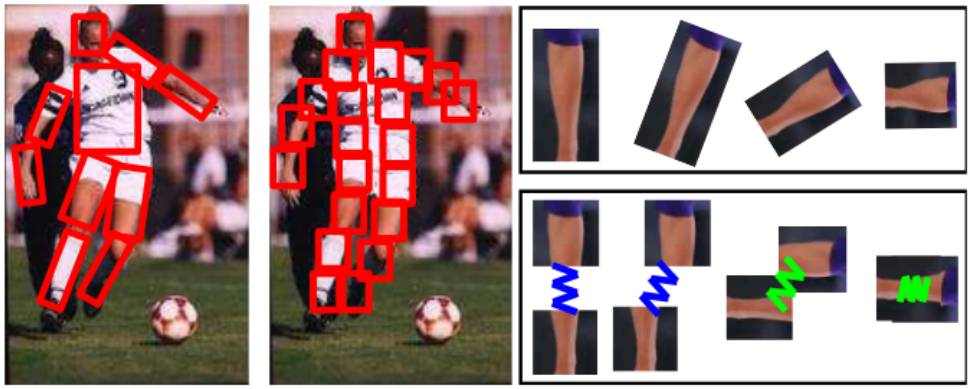


Figure 2.5: Classic approach (left). Mixture-of-parts model (middle). Warping a single template to different orientation and foreshortening states for model articulation (top right). Small warps connected with a spring (bottom right).

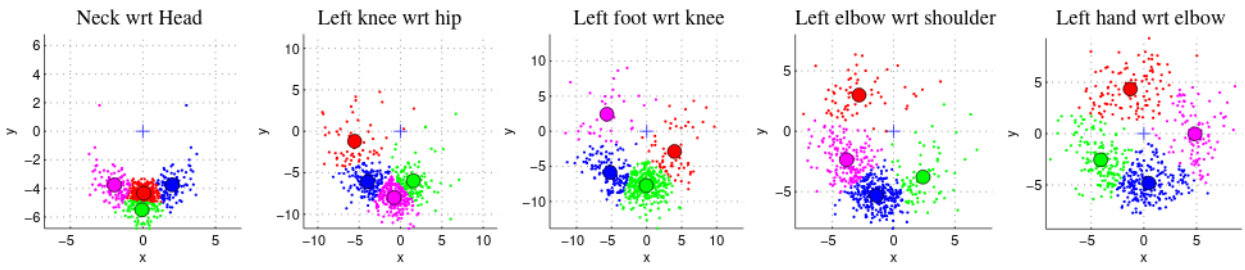


Figure 2.6: "Data-driven" approach to orientation-modeling by clustering the relative locations of parts with respect to their parents. These clusters are used to generate mixture labels for parts during training.

Moreover, in the paper [15], the authors introduced a couple of new evaluation criteria for pose estimation. The classical evaluation criterium was the so-called "PCP", or Probability of Correct Pose: a candidate body part

is labeled as correct if its segment endpoints lie within 50% of the length of the ground-truth annotated endpoints. Issues of this method were its sensitivity to the amount of foreshortening of a limb and the necessity of having a candidate and ground-truth placed in correspondence. Common weak solutions include evaluating the highest-scoring candidate given an image with a single annotated person or a window returned by a person detector. The first proposed evaluation criteria explicitly factors out detection by requiring test images to be annotated with tightly-cropped bounding box for each person, and the candidate keypoints is evaluated as correct if it falls within $\alpha \cdot max(h,w)$ pixels of the ground-truth keypoint, where h and w are the height and the width of the bounding box respectively, and α controls the relative threshold for considering correctness.

The latter was a crucial passage since the possibility of efficiently evaluating an algorithm strongly influeces its potential.

So far every algorithm has leveraged on the assumption of representing the human body as a tree-structured graph, but significant approaches that augmented the tree structure have been proposed. Since tree-structured models allow efficient learning but fail to capture additional dependencies between body parts, other than kinematic constraints between connected parts, a multiple-tree model was prosed by Y. Wang and G. Mori [16]. The hu-

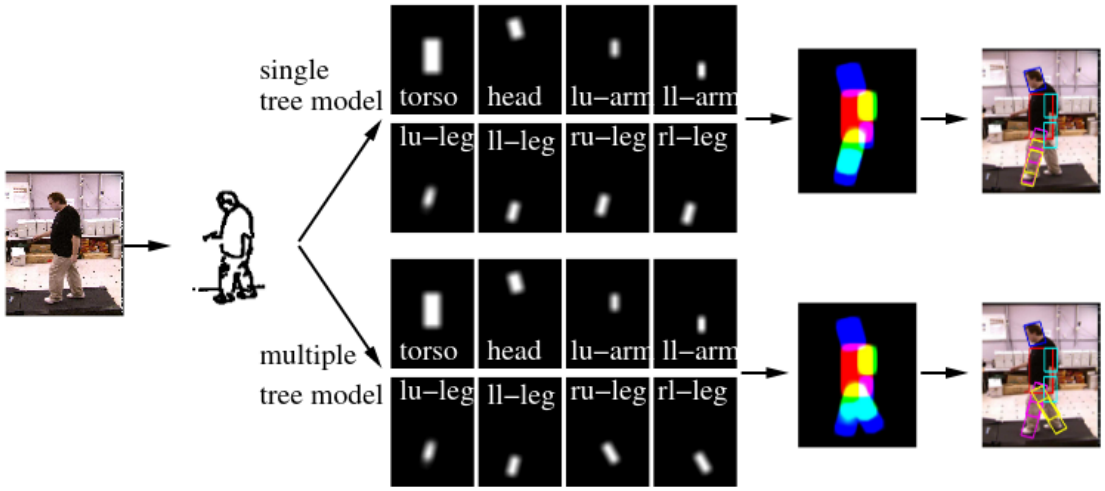


Figure 2.7: Top row shows how the same piece of image patch is used to explain two body parts, the bottom row shows how the multi-tree model occlusion reasoning mechanism can alleviate this problem.

man body is modeled as a weighted combination of several tree-structured deformable models. In the paper the authors proposed a couple of trees: the first one for kinematic constraints and the second one able to capture spatial constraints and to perform occlusion reasoning ("occlusion" to refer to the particular problem of using the same image patch to explain different body parts, fig. 2.7).

Tree-structured was mantained but enforced with a new machine-learning approach by Dantone et al., who proposed, within the pictorial structure framework, a novel joint regressor based on two-layered random forests [17]. The first layer acts as a discriminative, independent body part classifier, that is during both evaluation and training each sampled patch is evaluated without taking its spatially surrounding potentials into account. The second layer takes the estimated class distributions of the first one into account and predicts joint locations by modeling the interdependence and the co-occurrence of the parts. The model was tested on the FashionPose dataset (a collection of pictures with very high variation in cloth and appearance), outperforming the state-of-art models at the time (fig. 2.8). Another approach worth re-

error thres.	0.10	0.10	0.15	0.15
joints	Dantone	Yang et al.	Dantone	Yang et al.
Head	**66.97**	56.16	**78.84**	77.76
L. shoulder	**61.94**	53.21	**73.81**	72.75
R. shoulder	**61.81**	55.39	**74.19**	74.03
Left hip	**57.16**	38.43	**72.90**	58.61
Right hip	**58.58**	34.96	**73.81**	58.09
Left elbow	**41.81**	27.89	**56.00**	46.14
Right elbow	**41.29**	32.51	**58.84**	50.64
Left wrist	**32.26**	24.29	**44.26**	38.17
Right wrist	**29.68**	21.72	**39.48**	33.16
Left knee	**52.13**	39.07	**65.29**	56.94
Right knee	**49.94**	38.43	**62.71**	57.32
Left ankle	**43.87**	32.26	**58.97**	49.61
Right ankle	**41.68**	31.10	**58.58**	48.20

Figure 2.8: Detection accuracy for all joints at error thresholds 0.1 and 0.15.

porting, earlier that two previous methods, considered additional constraints (beyond the kinematic constraints given by the tree structure) such as limb

coordination [18], through the introduction of a small number of latent variables to represent residual correlations between parts that are not captured by a tree model.

2.3 Deep Learning based approaches

Deep Learning is a machine learning technique based on multi-layer artificial neural networks, imitating the working of the human brain in processing data and creating patterns. The main advantage of deep learning with respect to other machine learning techniques is the capability of automatically extracting meaningful features from raw data, then used for the learning task.

A particular type of deep learning model that uses the convolution with a kernel of custom dimension to map the input signal into the output feature maps came out to be very useful for machine learning tasks with images as input. The neural model learns convolutional weights able to extract meaningful features, i.e. simple patterns like straight lines or particular colors in the initial layers gradually more and more complex going deeper into the network, combining with a convolutional approach features from previous layers. This hierarchical path of processing visual information was inspired by the way human (and in general animal) brain succeeds to extract pattern from visual input: in 1959 David Hubel and Torsten Wiesel described how different stages from the retina to striate cortex react differently with the respect to the patterns of light stimuli [21] and, following up on that paper, introduced in 1962 [22] the existence of simple and complex cells, the latter capable of performing "spatial invariance" pattern recognition by summing the output of several simple cells that all prefer the same orientation but different receptive field. The first work on modern convolutional neural networks occurred in the 1990s [23], and demonstrated that a CNN model which aggregates simpler features into progressively more complicated features can be successfully used for handwritten character recognition, but only in 2012 a CNN called AlexNet [24] achieved the state-of-the-art performance labeling pictures in the ImageNet challenge (over 15 million labeled high-resolution images belonging

to roughly 22,000 categories).

Impressive performances achieved by deep learning are only possible with a huge amount of data (which means a lot of meaningful information used to train the model and make it able to generalize) and computational power, characteristics that made it popular just in the last decade.

In 2010 a paper from New York University [19] proposed a nonlinear embedding to handle the problem of visually matching people in similar poses. The model, called Convolutional NCA (Neighbourhood Components Analysis) regression (C-NCAR), consisted of a stack of convolutional layers followed by a single fully-connected layer (fig. 2.9), and trained following the siamese framework [20], used for a pair of examples; the distance between resulting codes drives parameter learning.

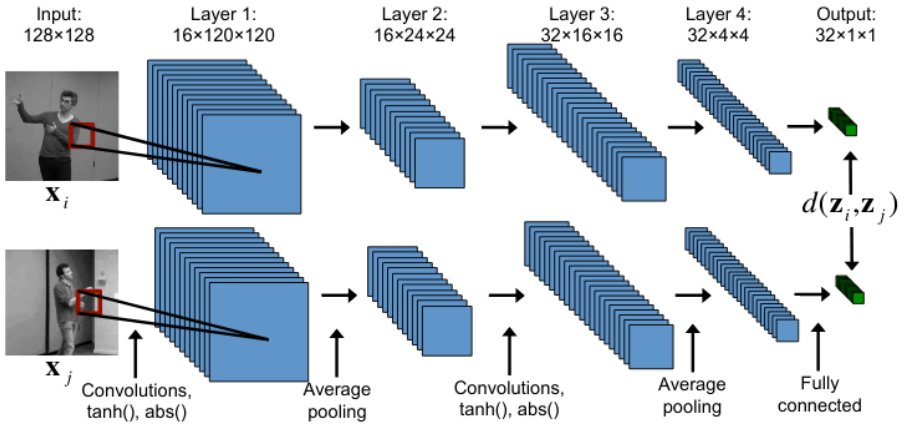


Figure 2.9: Convolutional NCA regression (C-NCAR).

Although it was the first paper that proposed the use of convolutional networks for embedding information regard human pose estimation, only four years later DeepPose [25] proposed a deep learning-based regression of human body joints. The network consists of 5 convolutional layers (followed by a ReLU nonlinearity and a pooling layer) and 2 fully connected layers, that implicitly captured interactions between joints. Since the network-fixed size input (220x220) implies a limited capacity to look at details, in order to achieve better precision, the authors proposed a cascade of pose regressor: each stage is a refinement of the previous one, receiving as input the original

images cropped around the predicted joint location from the previous stage. The same architecture is used at each stage but each one learns different parameters.

The required amount of training data was supplied by a couple of datasets: Frames Labeled in Cinema (FLIC) [26] (which consists of 4000 training images) and Leeds Sports Dataset [27]. The model outperformed all other approaches using PCP (percentage of Correct Parts) and its modified version PDJ (Percentage of Detected Joints), where the joint is considered detected if the distance between the predicted and the true joint is within a certain fraction of the torso diameter.

Hong Kong Chinese University developed a deep learning model for human pose estimation [28] as well, almost simultaneously with DeepPose. They combined in a unique nonlinear deep neural model different kinds of source information: visual appearance score, appearance mixture type and deformation, mixing their high-level representations obtained through some additional hidden layers (hence not mixed in the first layer). Moreover, the task for detecting humans and the task for estimating body locations are jointly learned using a single deep model. They obtained results slightly better than DeepPose.

At this point was clear that Deep Learning would have been the leading approach in Human Pose Estimation. Several researchers then tried to apply some of the most advanced deep learning techniques, developed for other task related to Computer Vision, for the task of "looking at the people".

Critical issues of DeepPose were linked to the direct regression of joint locations, i.e. mapping from RGB input images to a coordinate location. A novel model [29] from New York University solved this problem using a new multi-resolution model (fig. 2.10) that outputs a coarse heat-map for each joint, then used as input for the next model in a cascade-like architecture.
The model was further improved with a custom dropout called "Spatial

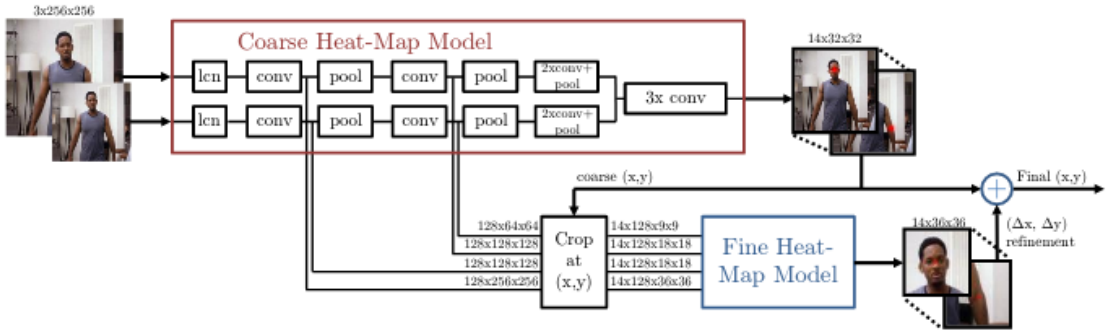


Figure 2.10: Cascade architecture.

Dropout", which maps adjacent pixels in the dropped-out feature map either all 0 (dropped-out) or all active.

Convolutional Pose Machine [30] was a model developed at Carnegie Mellon University, consisting of a sequence of convolutional neural networks that repeatedly produce 2D belief maps for the location of each part. Benefits of such a model were the capability of capturing long-range dependencies between image and multi-part cues and tight integration between learning and inference, thanks to convolutional networks, with a larger receptive field at each stage, that directly operates on intermediate belief maps and learns implicit image-dependent spatial models of the relationships between parts (without using graphical models). The problem of vanishing gradient, due to the huge number of layers, was addressed with intermediate supervision periodically through the network. This method achieved state-of-art but failed with multiple people in close proximity. Once again the iteration of similar/identical models proved to be the secret ingredient to get better results.

Now I'm going to introduce the killer model that settled a new standard for human pose estimation, adding (with respect to the previous model) the analysis of multi-resolution features as a key ingredient. It was called the Stacked Hourglass Network [31]. The network captures and consolidates information across all scales of the image, with a stack of downsampling-upsampling networks (fig. 2.11), placing modules together end-to-end.

Convolutional and max-pooling layers are used to process features down to

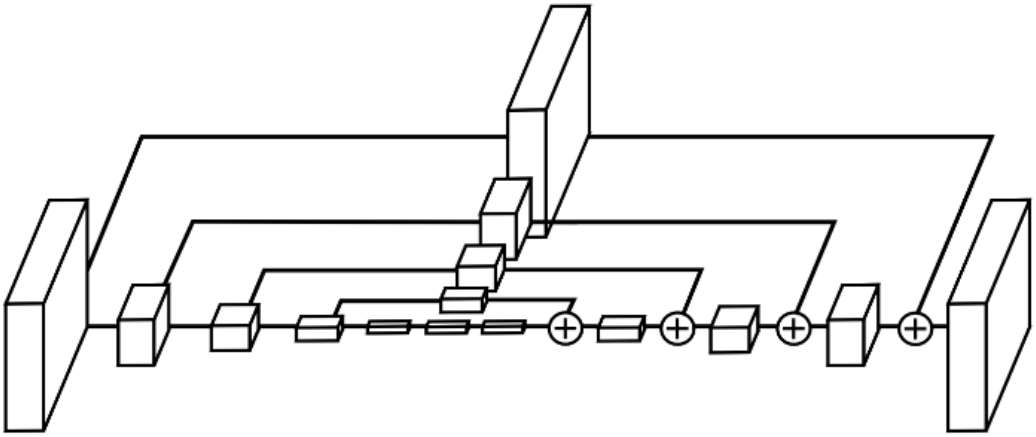


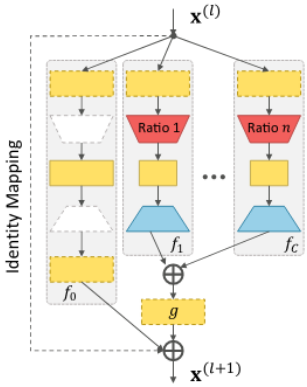
Figure 2.11: Hourglass (single) network.

a very low resolution. At each max pooling step, the network branches off and applies more convolutions at the original pre-pooled resolution. After reaching the lowest resolution, the network begins the top-down sequence of upsampling and combination of features across scales. Residual units are used as skip connections between downsampling and upsampling sides. After reaching the output resolution of the network, two consecutive rounds of 1x1 convolutions are applied to produce the final network predictions. The output of the network is a set of heatmaps where for a given heatmap the network predicts the probability of a joint's presence at each pixel.

Multiple hourglasses are stacked in an end-to-end fashion, feeding the output of one as input into the next; intermediate supervision is used after each module in order to avoid gradient vanishing, using the same ground truth (weights are not shared between different hourglass modules).

The Stacked Hourglass Network pushed forward human pose estimation results for single person pose estimation, but was still less performing with occlusion and multiple people. Some relevant further works tried to improve performances by changing the residual unit.

It was observed that outputs of two residual units summed up caused a doubling of the variance that may lead to issues during optimization. Yank et al. proposed a different residual unit for the Hourglass network called Pyramid Residual Module [32], able to



learn multi-scale feature pyramids: given input features, the module obtains features of different scales via subsampling with different ratios (fig. 2.12).

The model achieved the state-of-art on the MPII dataset [33], totaling peaks of 98.5 score for head recognition, 96.7 for shoulders recognition, 92.5 for elbows.

Similar results were achieved by a cross-study work by The Chinese University of Hong Kong, Tsinghua University, and Johns Hopkins University [34]. Starting from Stacked Hourglass Network, they prosed a novel model with different residual unit and the massive exploitation of attention maps from features at multiple resolutions with various semantics. The baseline was a stack with eight hourglass networks, with intermediate supervision at each stack. Residual Units were replaced with the novel micro hourglass residual (HRUs) units, consisting of three branches: the identity mapping, residual blocks like in ResNet [35] and a third stack with one 2x2 max-pooling, a couple of 3x3 convolutions followed by ReLU nonlinearity, and an upsampling operation. Within each hourglass, the multi-resolution attention maps are generated from features of different scales, mapping features at each level of the upsampling part of the hourglass, modeling spatial correlation with Conditional Random Fields, and then summing them into a single attention map. Different stacks are with different semantics: lower stacks focus on local appearance, while higher stacks encode global representations.

The attention maps help to drive the network to focus on hard negative samples. After several stages of training, the attention maps fire on the human body region, where the true positive samples are highlighted by attention maps. The refined features are used for learning classifiers for the regions with human body, with easy background regions removed at the feature level by the learned attention maps. Consequentially, for part attention maps, the classifiers focus on classifying each body joint based on well-defined human body regions, without considering the background.

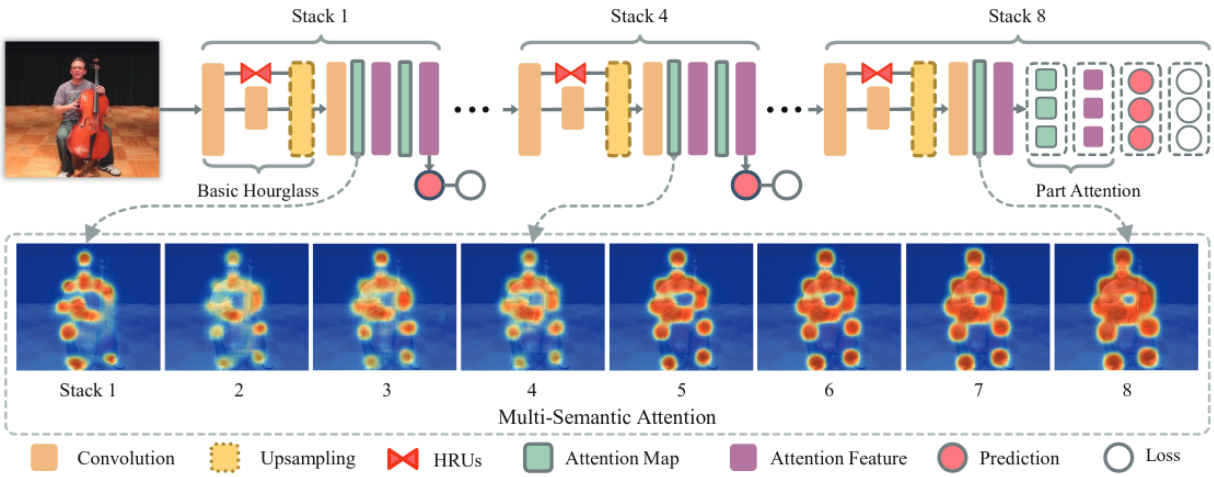


Figure 2.13: Framework of the multi-semantics attention hourglass network.

Hourglass network was used as the base model for the Multi-Scale Structure-Aware Network for human pose estimation [37]. The original model was improved with four key addition: the multi-scale supervision to strengthen contextual feature learning in matching body keypoints by combining feature heatmaps across scales; the multi-scale regression network at the end to globally optimize the structural matching of the multi-scale features; the structure-aware loss used in the intermediate supervision and at the regression to improve the matching of key-points and a keypoint masking training scheme that can handle occlusions. The multi-scale supervision network is designed to learn deep features across multiple scales, while supervision is performed by calculating the residual at each deconv layer using the corresponding down-sampled ground-truth heatmaps in the matching scale (it localizes body keypoints in a way similar to the "attention model"). The multi-scale regression network (fully convolutional) is used after conv-deconv stacks to globally refine the multi-scale keypoint heatmaps: the regression can effectively combine heatmaps across all scales to refine the estimated poses. The structure-aware loss, based on human skeletal graph, is used for intermediate supervision in order to avoid gradient vanishing in deep stack of conv-deconv networks.

Hourglass Networks (and its derived methods) had demonstrated remarkable performance in human pose estimation, but still suffered from several

problems related to the compositionality of visual patterns. None of the methods discussed so far decomposes entities as hierarchies of meaningful and reusable parts or infers across different semantic levels, approach used by Tang, Yu and Wu [36] for their milestone work.

With their model, called DLCM, exploiting deep learning for human pose estimation went further: it attempts to explicitly learn the hierarchical compositionality of visual patterns via deep neural networks: CNNs are used to model Spatially Local Information Summarization (SLIS). As previous deep-learning based model, the first part (bottom-up stage) is used to regress target joints directly from the images; then, score maps of higher-level parts are recursively estimated from those of their children. In the top-down stage, score maps of lower-level parts are recursively refined using their parents' score maps as well as their own score maps estimated in the bottom-up stage.

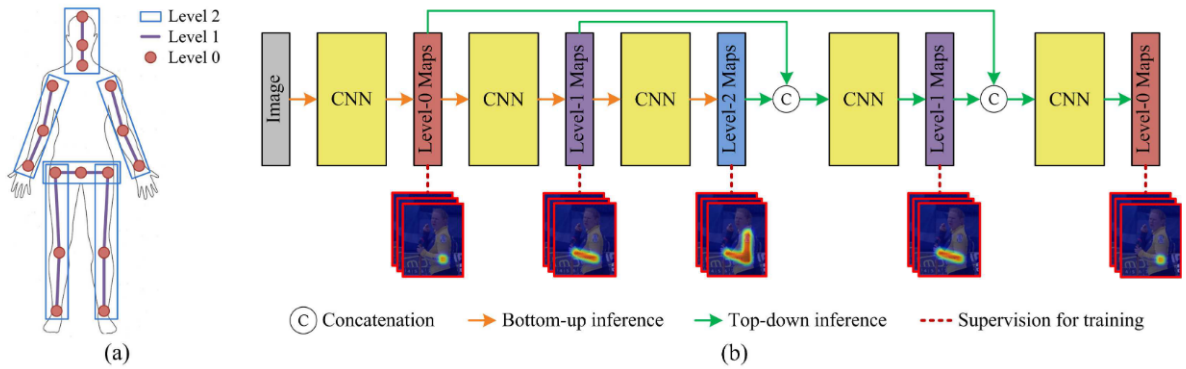


Figure 2.14: (a) Compositional structure, with three semantic level. (b) Network architecture of DLCM.

The problem of efficiently defining priors regard human body was handled in an original way using GANs (Generative Adversarial Network)[38], an architecture composed of two networks: the generator and the discriminator. The first one has the task of generating data from initial random values, picked from a certain distribution; the discriminator, on his side, has to recognize if the input data are synthetically generated (from the generator network) or come from the real dataset. Two different losses guide the learning for each network. The result of this particular architecture is a network capable of generating data that, eventually, are indistinguishable from the real ones, i.e.

lay on the same distribution of the real data.

This model was embedded in the Adversarial Posenet [39] architecture, exploiting the capability of the discriminator of distinguishing the real poses from the fake ones, improved by the generator. The model is composed of three parts: the multi-task pose generator network; a pose discriminator, which have to distinguish the fake poses from the real ones, based on the encoder-decoder architecture in order to use low and high-level information to infer if the predicted poses are biologically plausible; and the confidence discriminator, to discriminate the high-confidence predictions from the low-confidence predictions.

So far the analysis is mostly focused on single-person pose estimation, i.e. techniques that require computing power proportional to the number of people. In real case scenarios, the number of people is unpredictable and it's required a predictable computational cost, therefore the algorithms that had the most relevant commercial use were ones following the multi-people pose estimation approach. Among them, I have found a categorization based on the way the model infers the human body pose: the so-called top-down approach, which regresses the body joints starting from the whole body, and the bottom-up approach, which starts by defining the candidate joints to regress the whole body structure in the final steps. Although this division can be applied to single person pose estimation as well, it's particularly meaningful when analyzing and approaching the multi people estimation, due to the complexity of the regression, and the brilliant approaches that it requires.

Typically, the top-down approach is easier to implement than the bottom-up approach as adding a person detector is much simpler than adding associating/grouping algorithms.

For the top-down approach, the most notable algorithm, different from the ones already discussed, is the Cascaded Pyramid Network for Multi-Person Pose Estimation [40]. It was introduced to deal with the problem of "hard" keypoints: occluded or invisible keypoints in complex backgrounds. The latter

was addressed using a couple of models: GlobalNet, used to extract useful features with a pyramid network, similar to the widely cited Hourglass Network. RefineNet was added to explicitly address the problem of "hard" keypoints, by concatenating all the pyramid features. The multi-person pose estimation problem was addressed with a top-down pipeline. Human detector is first adopted to generate a set of human bounding boxes, followed by the Cascaded Pyramid Network for keypoint localization in each human bounding box.

Even if the joints regression performed well, still remains the problem common to all top-down methods: they are usually dependent on the accuracy of the person detector, as pose estimation is performed on the region where the person is located. Hence, errors in localization and duplicate bounding box predictions can cause the pose extraction algorithm to perform sub-optimally. To overcome this issue, another popular framework was proposed, RMPE (or AlphaPose) [41].

The model exploits the Symmetric Spatial Transformer Network, capable of extracting high-quality single person region from an inaccurate bounding box. The firstly generated proposals are refined by a Parametric Pose NMS, which performs pose non-maximum suppression; moreover, the proposals generator is pose-guided: having the ground truth pose and an object detection bounding box for each person, a large sample of training proposals (with the same distribution as the output of the human detector) is generated. This approach partially solved the problem of human detection accuracy.

Over the commitment to person detection, top-down approaches suffer in handling spatial dependencies across different people. Some approaches started to consider inter-person dependencies, extending the pictorial structures to take a set of interacting people, but still requires some apriori person detector to initialize detection hypotheses.

Starting from top-down approach issues, such as already mentioned the failure of human body detector with close proximity between people, Pis-

chulin et al. proposed the first relevant bottom-up approach[42]: poses of all people is jointly estimated by minimizing a joint objective. Properties of this model are the capacity of dealing with an undefined number of people, reliability given by non-maximum suppression based on the entire sets of candidates (not only on individual part candidates), computational advantage given from the Integer Linear Programming Approach, that, although is a NP-Hard problem, ensures a certified optimality gap. The model was further improved [43] with deeper networks in the joints regression and with a novel image-conditioned pairwise term given by a CNN-based part detectors, that significantly speeded-up the inference.

Figure 2.15: In the upper part there are shown the pairwise terms combined. In the lower part there are the unaries.

The model performed very well with cases with only a subset of body parts of a person visible in the image, as well as with rare body articulations, but still the multi-people factor represented an issue: in some cases the model failed to disambiguate between people in close proximity, or to assign spurious body configurations that can not be assigned to any ground-truth annotation. Nonetheless, the model significantly reduced the run-time and allowed to almost double the pose estimation accuracy in the multi-person case.

The last model I want to present is OpenPose [44], the one I chose for my project. The authors of the model had to face a number of challenges

related to multi-people pose estimation: the unknown number of people at any position and scale, interactions among them (that may lead to contact, occlusions), real-time performances. The chosen approach was bottom-up, avoiding issues related to the dependency from person detectors and exploiting a global inference, taking into account all the key points per time. It was for the first time presented a bottom-up representation of association scores via Part Affinity Fields(PAFs) [45], i.e. a set of 2D vector fields that encode the location and orientation of limbs over the image domain. After several changes to the first experiments with PAFs, in 2019 they came up with Openpose.

The input is a color image, firstly analyzed by a CNN that generates a set of feature maps F that is input to the first stage. The network in the first stage produces a set of PAFs (part affinity fields), and at each subsequent stage, the predictions from the previous stage and the original image features are concatenated and used to produce refined predictions. After a certain number of repetitions, the process is repeated for the confidence map detection. Differently with respect to previous models with PAFs, here the PAFs detection and the confidence maps detection are divided in two different stages, the latter starts from the most updated PAFs detection, rather than repeating the two stages together at each iteration.

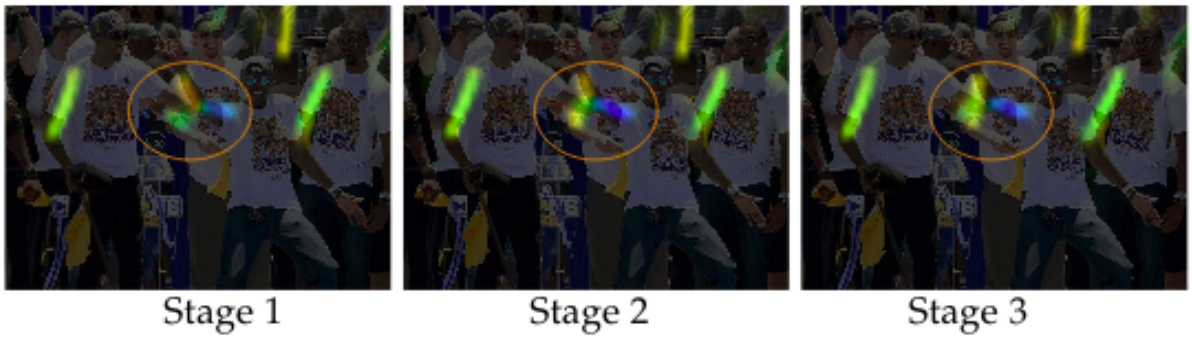


Figure 2.16: PAFs of right forearm across stages.

Part Affinity Fields are very suitable for part association because they preserve both location and orientation information across the region of support of the limb. Each PAF is a 2D vector field for each limb, that encodes the direction that points from one part of the limb to the other.

Multi-person parsing is computed performing non-maximum suppression

on the detection confidence maps to obtain a discrete set of part candidate locations. Each candidate is scored using the line integral computation on the PAF; the problem of finding the optimal parse corresponds to a K-dimensional matching problem that is known to be NP-Hard. In the paper, the authors proposed some relaxations to efficiently compute the inference on the graph: minimal number of edges to obtain a spanning tree skeleton of human pose rather than using the complete graph and decomposition of the matching problem into a set of bipartite matching subproblems. The latter is feasible because, while relationship between adjacent nodes is modeled explicitly by PAFs, the relationship between non-adjacent tree nodes is implicitly modeled by the CNN (trained with a large receptive field).

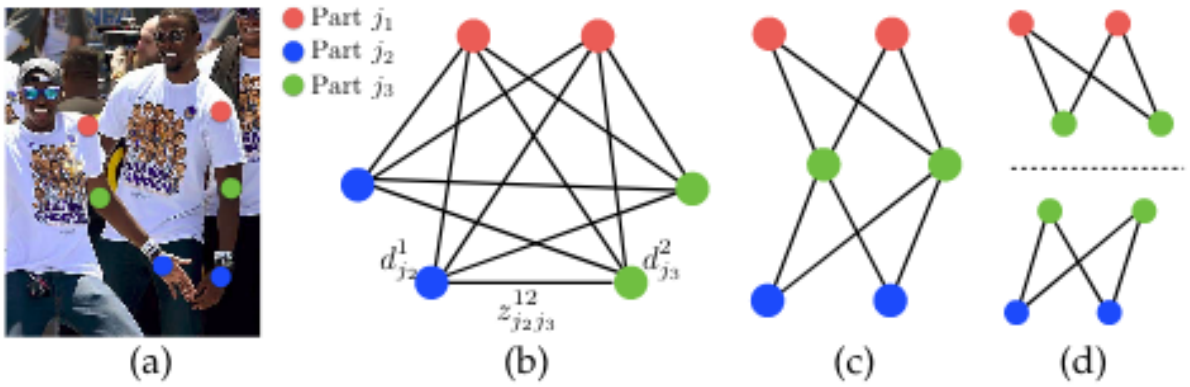


Figure 2.17: Graph matching. (a) Original image with part detec-tions. (b)K-partite graph. (c) Tree structure. (d) A set ofbipartite graphs.

2.4 Why Human Pose Estimation

Human Pose Estimation is a central research topic in artificial intelligence because it investigates one of the central tasks of human life: the ability to perceive complex human movements, and their interactions with the surrounding space. It involves a bunch of tasks, from the vision, to object detection, up to the inference about the relationship between detected pose(s) and the context. Given the central role that this ability plays in everyday life, practical applications are countless. Trying to define an order, I've categorized actual implementations of Human Pose Estimation techniques considering the purposes of the applications:

- Motion tracking for virtual reality, animation, videogames,
- Human-computer interaction,
- Safety,
- Security,
- Sport and training,
- Others.

The field that was (and is) revolutionized the most by human pose estimation is the world of animation, including videogames and augmented and virtual reality. Every expensive and awkward hardware can be replaced by a simple camera (or a couple, for 3D-oriented animations), and the movements are immediately transferred to an alter ego in a fantasy videogame, or a first-person point-of-view hand in a virtual reality context.

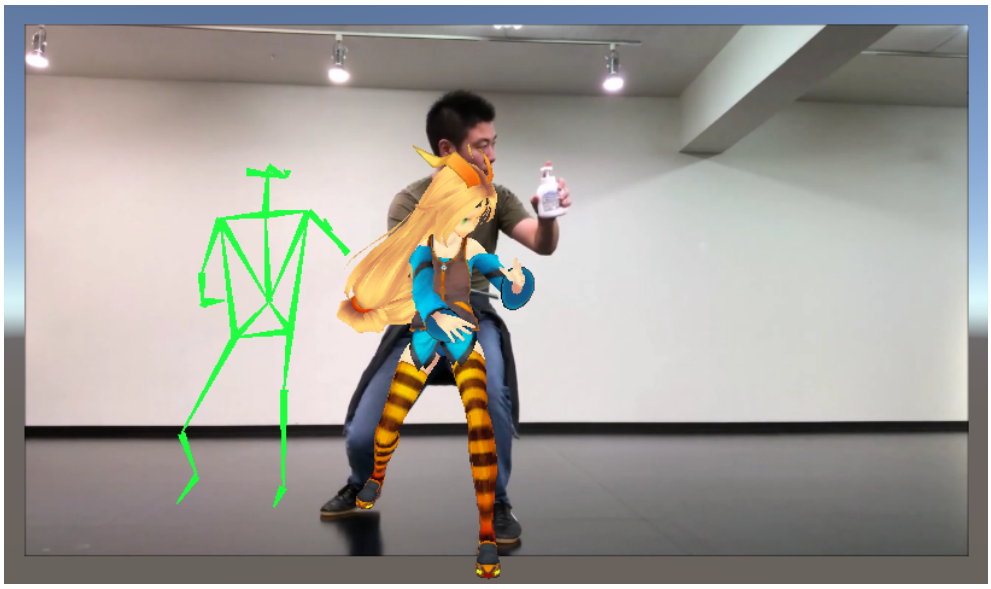


Figure 2.18: Example of motion tracking in augmented reality.

The capability of a machine to perceive the surrounding space and foreseeing human movements makes it able to physically interact with people. Human-computer interaction is the field that studies the interfaces between people and computers, taking into account several factors from behavioral sciences and design to biomechanics and neuroscience in the most advanced

researches. Human pose estimation is a unique tool for enabling new and more human-oriented interaction based on familiar human gestures: it makes possible remote controlling of an application using arms and hands poses or content creation (writing, drawing, etc.) using particular body poses.

Human pose estimation has a wide application in system concerning safety. In autonomous cars, an intelligent system able to map human body position can prevent pedestrians' movements in order to avoid road accidents or simply detect persons going across the pedestrian stripes. Home applications for fall detection can be very useful in case of fainting, falls in the case of older people, or tracking movements of babies.

Even if such applications could be incredibly beneficial, but their practical implementations is limited by the required accuracy, close to 100% that is required since the delicate situations that such systems have to handle.

Security has massively exploited human pose estimation, usually embedded in most modern intelligent surveillance systems. Pose detection can automatically identify dangerous situations like fights or robberies, or have a trace of a suspicious person.

Sport is the field where human pose estimation has had and has the largest commercial application. Almost every sport that includes some physical activity can benefit from pose recognition, for several purposes: training, evaluation, comparison between athletes, simulations of real situations. The most common human pose estimation based applications regard self-training: the model can follow the user movements and discover wrong poses, like a personal trainer. The algorithm can figure out from the relative position of the joints if the user is performing a wrong movement, and instantly it can suggest the custom correction, keeping trace of each movement (data that might be used for medical purposes). Based on our peculiar body and movements the algorithm can also identify which is the training that best fits our natural poses.

Figure 2.19: Example of home training analyzed by a digital personal trainer.

In sports with dynamic schemas, such as football or basket, human pose estimation can be used to analyze the position and trace the movements of each player taking part in the attack or defense pattern, both to figure out the opposing team's tactics and to improve your team's ones.

Sports like dance can benefit from human pose estimation as well, having a powerful tool to analyze specific poses and choreography.

Human pose estimation can be applied to almost every situation in which can be useful having a trace of human pose: some other applications are parsing clothes, i.e. segmentation of different clothes exploiting joints positions, nonlinear video exploration, or nonlinear database queries.

Our application combines the latter two techniques with sports videos, specifically climbing videos.

Chapter 3

Nonlinear climbing video indexing Application

As a climber, the most difficult part of the training session is to figure out which is the right sequence of movements to send the route, and very often the only way is to wait for someone smarter and more experienced to do the route and following his movements.

To face this common issue, my supervisor, Jaakko Lehtinen, and I decided to develop an application where the user can get pieces of different videos where the climber is in the requested pose, or in a certain passage of the route. We applied human pose estimation to climbing videos to create a database that can be accessed from the pose of interest, and a smart interface to get all interesting parts of different videos and to compare, with the usage of a scrubber, how different climbers have done the same passage.

The first section of this chapter is about the work and research done before starting the implementation: human pose estimation algorithms review, current applications, user stories, sketches about user interface and user experience, analysis of the best software architecture to accomplish our goals.

The second section is about the database: after a brief review of OpenPose and how it fits with our goals, I described where I have taken the data and how pose data are stored in the database.

The third section is about the first implementation I did, programming a prototype to test the basic functionalities of the application and how well it performs.

The fourth and fifth sections are about the final application, explaining step by step the frontend, the user interface, and information flow with respect to user actions, and all the components of the server, i.e. where almost all the logic takes place.

3.1 Preliminary work and research

The first step was defining the actual benefits that the application should have brought. I chose the agile framework for development, and the first step that this approach suggest is defining some user stories, in order to identify who, how and in which scenario will benefit from the application. After several discussions with my professor, we have identified three main user stories:

- As a user, I want to get all the videos where a selected body part is in a selected area of the wall I'm going to climb, in order to look at how other climbers have done that passage. (fig. 3.1)
- As a user, I want to scrub across different clips in order to make a comparison between different climbs on the same passage. (fig. 3.3)
- As a user, I want to get all the clips where the climber is in a certain pose by selecting the frame (with the climber in that particular pose), to look at how different climbers had done a certain movement.

Each user story corresponds to a specific feature of the application.

With three sketches, I've defined the structure of the interfaces for the different pages of the application, linking each of them with the user stories (and corresponding features). They showcase the user experience, i.e. how I imagine the information flow and the path that the user has to follow to achieve his scope.

The first page (fig. 3.1) shows the different views of the gym: each image corresponds to a wall or part of a wall. It lets the user select the wall with the route of interest. The second one (fig. 3.2) enlarges the selected view, where the user can select the area of interest (e.g. the area within the holds

of the difficult passage); a stylized skeleton on the left lets the user choose 1 out of the 25 body key points that OpenPose detects.

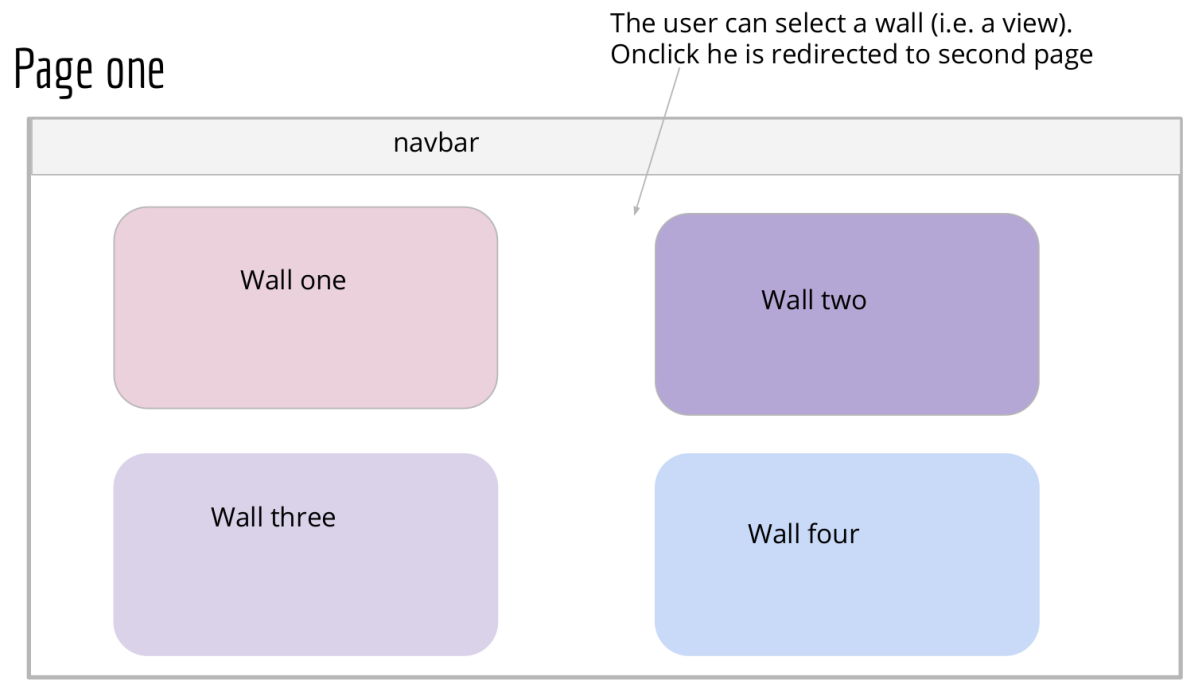


Figure 3.1: First page of the application.

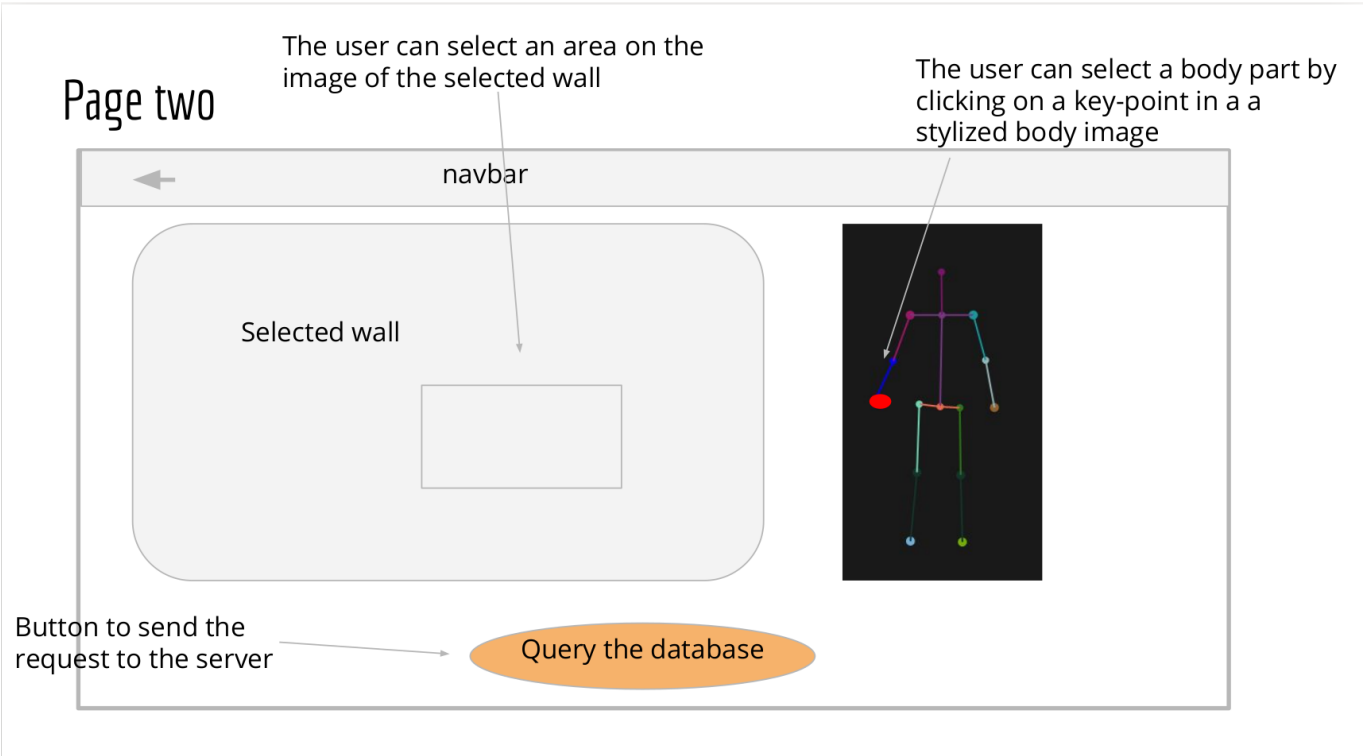


Figure 3.2: Second page of the application.

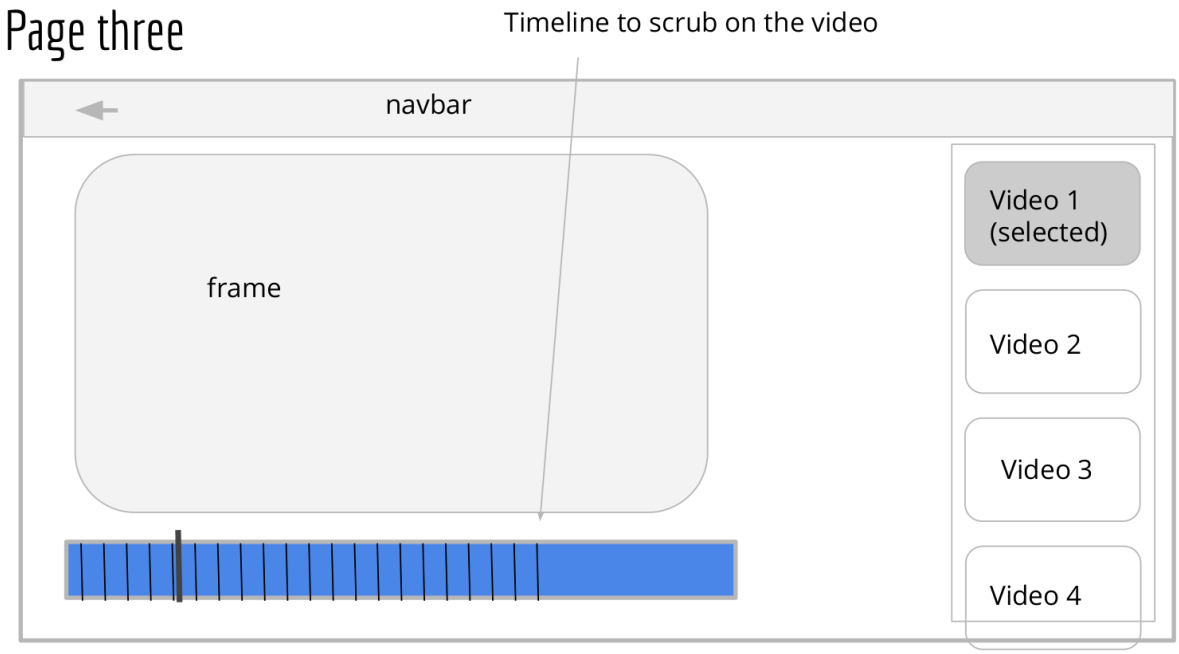


Figure 3.3: Third page of the application.

The third page (fig. 3.3) shows all the videos gotten from the query. The user can select the video that will be displayed, starting from the requested frame, i.e. the first frame where the climber has the selected body part in the selected area. The video can be closely explored with a scrubber that lets the user go back and forth frame by frame.

Once the general shape of the application was defined, the first issue was finding the best algorithm (considering different metrics and features) to map the videos and to get the pose of the climber frame-by-frame.

The first feature was the ability to perform human pose estimation within certain conditions, specifically with video of medium quality, blurry, hard-to-follow, and noisy at times, since the main sources are users' smartphones and gym security cameras (where available).

The second requirement was the open-sourceness, with an inspectable code and a license compliant with our research project.

Regard performances, I did not establish a specific threshold for metrics like accuracy, precision, or recall, but empirically I've checked an overall performance that guaranteed application usability within our conditions. Even

if the detection misses some frames, the model has to guarantee a full detection in at least one frame per second, in order to retrieve the pose that is almost the same as the missing frames. Linked to the latter point, I sought an algorithm that performs detection frame by frame, without considering information about previous frames, i.e. considering each frame as stand-alone.

After a broad review of the most popular open-source frameworks and models for human pose estimation, presented in the previous chapter, I came up with OpenPose [44]. In the section about the database, I demonstrate how it perfectly fits with the application's needs.

Once the main features of the application were defined and the algorithm for pose estimation was selected, I started to think about the architecture of the application itself.

Each climbing gym has to collect its own video set, based on its views and routes. Videos can be manually collected (and added), shot with every kind of camera (from smartphones to professional cameras), or can be automatically taken from fixed cameras in the gym like security cameras or simple webcams. Once the database is set up, the application is intended to be used from two main devices: a personal phone or a (shared) screen in the gym. I choose to model the architecture following the client-server framework, using Rest API to communicate, with the entire logic part located in the server.

An API, or application programming interface, is a set of rules that define how applications or devices can connect to and communicate with each other; it enables an application or service to access a resource within another application or service. The application or service doing the accessing is called the client, and the application or service containing the resource is called the server. A Rest API is an API that

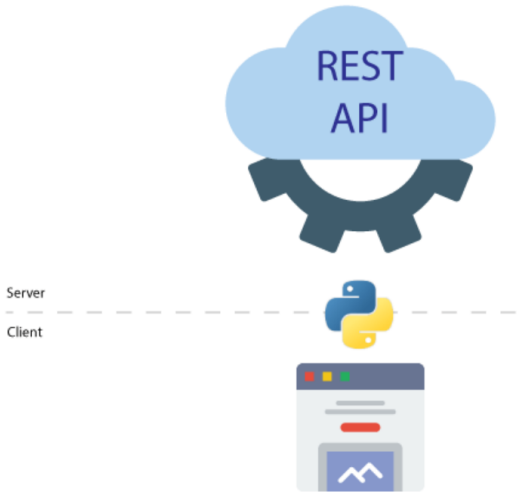


Figure 3.4: Rest API framework.

conforms to the design principles of the
Rest, or representational state transfer architectural style. Principles of this framework are:

- Uniform interface: specification of the API is a contract between the app and the server. The app needs to know that it can hit the same URI to get a particular piece of data, and needs to know that it'll get the data in the format that it expects, and that format will not change.
- Statelessness: The specification of the API should provide all the information. It does not expect nor assume that any state from previous calls has been preserved. The server should have no knowledge of prior requests. The client needs to provide all the information necessary for the server to provide a response.
- Client-Server model: The point of intersection between the server and the app is the database schema. The two most important points of intersection between the client (frontend) and the server (backend) are the data format that client needs and the granularity of the information.
- Caching: Caching is the temporary storage of information outside of the server.
- Layered architecture: The application is organized in layer: each layer knows about the layer next to it, and no more and it is responsible for a specific role.

I choose this pattern in order to reduce the computational demand on the user device (that might vary from a smart tv in the gym to a smartphone), to have a protected and supervised access to the database.

To make the application portable (i.e. easily accessible from different devices) the frontend part was developed as a web-based application, using React framework (discussed in section 3.4). Such a framework is suitable for mapping onto a mobile-based application, for further developments.

Since Python allows most of the scientific computation required by the logic part, the server was built up with Flask Python (discussed in section 3.5), using a React native proxy server in Javascript.

3.2 OpenPose and the database

OpenPose is the selected algorithm for human pose estimation. It was developed at Carnegie Mellon University by Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh [44], in 2018 (the general idea and overall pipeline are still the same as the first model developed around the idea of Part Affinity Field, 2016 [45]). The first requirement was the open-source code, it is entirely on Github and is very well documented. OpenPose is originally written in C++ and Caffe.

The pipeline is structured as follows. An input RGB image is first analyzed by a pre-trained convolutional neural network such as the first 10 layers of VGG-19, to produce a set of feature maps F. Feature maps are fed as input into a "two-branch multi-stage" CNN. Two branch means that the CNN produces two different outputs; multi-stage is intended the stacked architecture of the model (where each network is on top of the other at every stage).

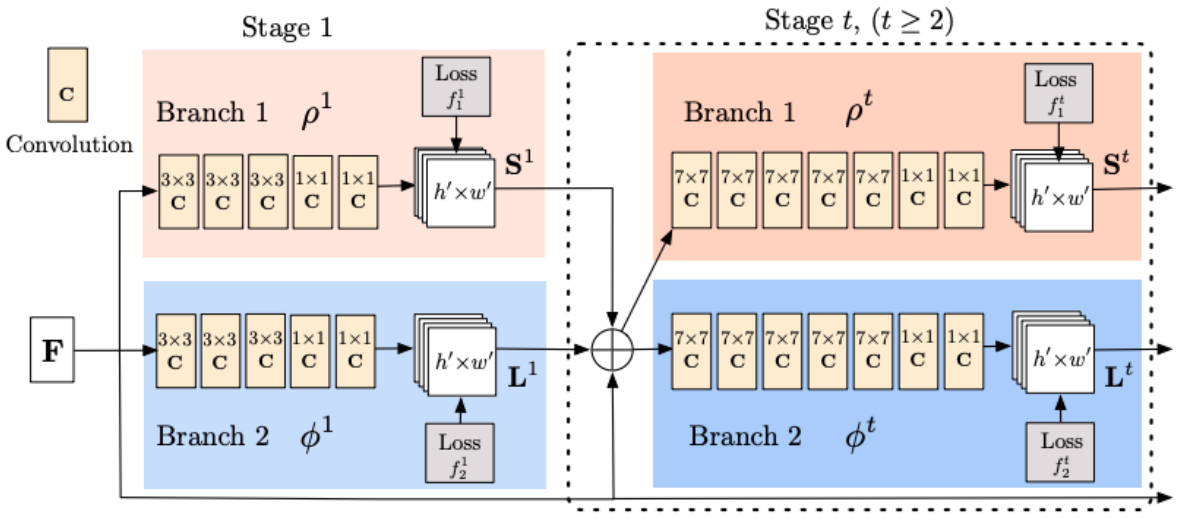


Figure 3.5: Architecture of the two-branch multi-stage CNN.

The top branch, shown in beige, predicts the confidence maps of different body parts location such as the right eye, left eye, right elbow, and others. The bottom branch, shown in blue, predicts the affinity fields, which represent

a degree of association between different body parts. PAF are mathematically defined as

$$L = (L_1, L_2, L_3 ... L_c)$$

$$L_c \in \mathbb{R}^{w \times h \times 2}$$

$$C \in \{1 ... C\}$$

C is the total number of limbs, i.e. part pairs, defined by a couple of connected key points. The latter depends on the dataset that OpenPose is trained with; I choose a model pre-trained on COCO keypoint dataset [47] plus a custom dataset for foot, with a total of 25 key points that define the human body (fig. 3.6).

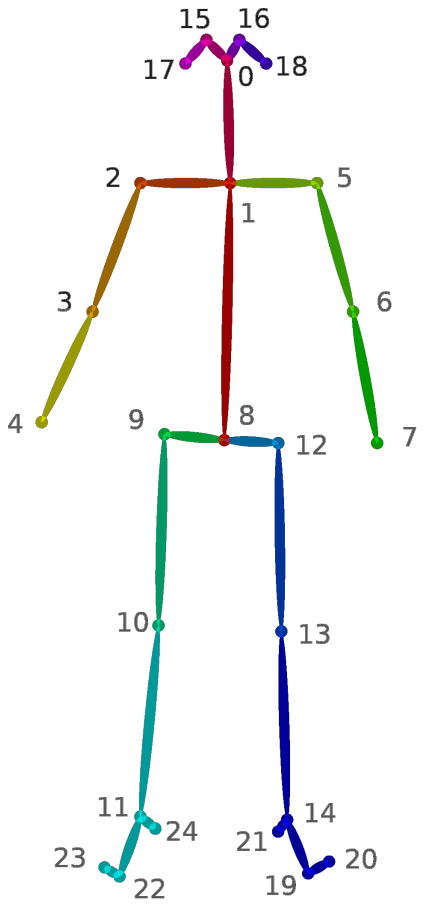


Figure 3.6: Key points detected by OpenPose.

Each element in the set L is a map of size w x h where each cell contains a 2d vector representing the direction of pair elements.

At the first stage, the network produces an initial set of detection confidence maps S and a set of part affinity fields L. Then, in each subsequent stage, the predictions from both branches in the previous stage, along with the original image features F, are concatenated and used to produce more refined predictions. In the OpenPose implementation, the final stage t is chosen to be 6.

Finally, the confidence maps and affinity fields are being processed by greedy inference to output the 2D key points for all people in the image.

In order for the network to learn how to generate the best sets of S and L, the authors apply two loss functions at the end of each stage, one at each

branch respectively. The paper uses a standard L2 loss between the estimated predictions and ground truth maps and fields. The combined loss function are summed up together to come up with the overall objective.

The base pipeline detects the body, including the feet; it can be further extended with face detection, which exploits the facial keypoints output of the core block (ears, eyes, nose, neck), and hand detection, generated starting from hand keypoints. The core block was enough for our purposes; moreover, hands are often hidden by the climber body or the holds, so hard to detect reliably.

OpenPose turned out to be the best choices for our purposes for the following reasons:

- State-of-the-art performance regardless of the number of people, useful with crowded walls. With videos recorded with means of security cameras, or smartphones, videos are automatically sent to the database: the algorithm has to perform well and in a reasonable time regardless of the number of people, potentially many on the view of a camera.
- High performance with medium/low-quality videos. As told before, videos are supposed to be recorded without professional supports: most likely videotape will be taken with smartphones or cameras provided by the gym (security cameras, webcam, etc.). Such an unforeseeable scenario requires a model able to perform well even in presence of noise, blurred scenes, or low-quality videotapes.
- Open-source code. The code has to be inspectable, in order to study the exact behavior and figure out if it fits with our specific purposes.
- Stateless algorithm between frames. Given the unreliable quality of videos, some frames can be difficult to detect. The application requires that the model detects correctly at least one frame per second, meaning 1 over 25. In this way the application guarantees that the pose of missing frames can be approximate to the closest detected frames, without any inconsistency.

```
{
    "version":1.3,
    "people":[{"person_id":[-1],
        "pose_keypoints_2d":[0,0,0,839.338,480.219,0.9287,868.807,483.189,0.844,..],
        "face_keypoints_2d":[],
        "hand_left_keypoints_2d":[],
        "hand_right_keypoints_2d": [],
        "pose_keypoints_3d":[],
        "face_keypoints_3d":[],
        "hand_left_keypoints_3d":[],
        "hand_right_keypoints_3d":[]}]
}
```

Another main advantage of OpenPose is its license: it allows academic and research use.

The code can be cloned and forked from Github [48]. After the building, that required the change of a couple of lines to adopt a new version of C++, the executable takes a video as input and store a JSON file for each frame.

The json schema is represented above. For each person it defines a series of detection: the core is the *pose_keypoints_2d*, that contains 75 values (25x3): each of the detected 25 keypoints (body+feet) is represented with the coordinates of the point (x, y) plus a value k that reports the confidence of the detection, between 0 and 1. The other keys (*face_keypoints_2d*, *hand_left_keypoints_2d*, ...) are void in the basic usage. You can call the executable passing as argument flags for enabling hand (- - *hand*) and face detection (- - *face*).

The first step was mapping the entire set of files to one file per video. We assume each video contains exactly just one climb till the top of a route (even if there are other people in the view); this means that long videos have to be splitted, producing one video for each climb, from the beginning to the top.

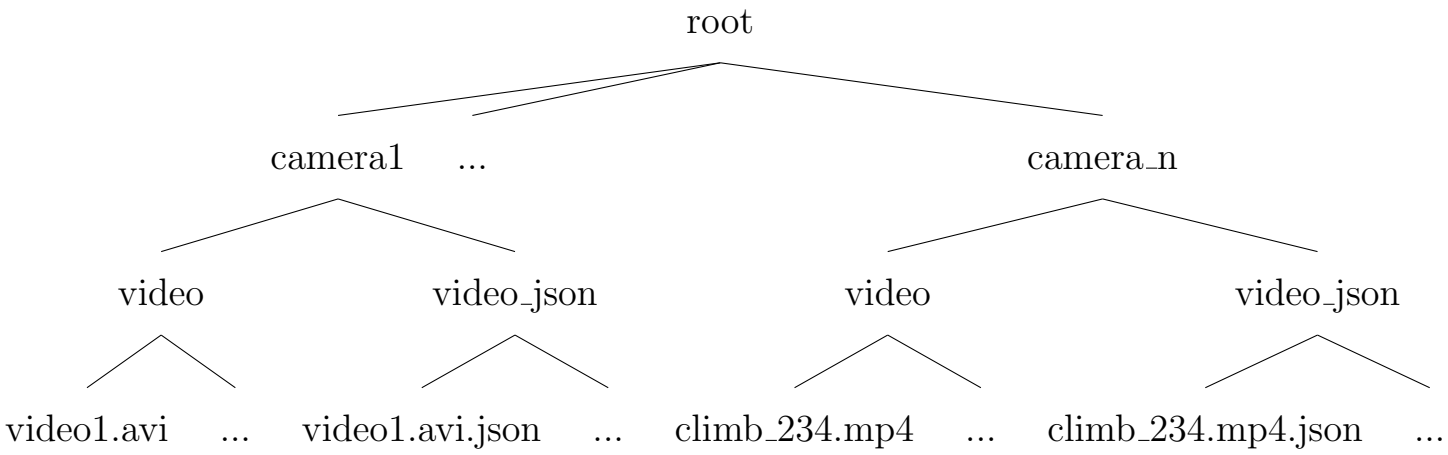
```
for file in files:
        if file.endswith(".json"):
            with open(dir_path+"/"+file) as f:
                data = json.load(f)
                if len(data['people'])>0:
                    n=int(file.split('_')[1])
                    data = data['people'][0]['pose_keypoints_2d']
                    videoFrames[n] = data
```

```
{
    "0": [[83.465, 918.788, 0.0782075, 883.4, 930.514, 0.678635, ...]],
    "1": [[0, 0, 0, 883.465, 927.718, 0.884475, 907.018,930.466, 0.822, ...]],
    "...": [[....]],
    "274": [[974.785, 568.51, 0.061777, 965.892, 598.028, 0.902364, ... ]],
    "...": [[....]],
    "n-frame": [....]
}
```

The code loops for every JSON file of the videos, and collect in an array the values of *pose_keypoints_2d* for each person detected in the frame. The final output for each video is a single JSON as shown above.

The JSON collects a list of pair key-values, with the number of the frame as key and the list of the 75-entries array (one for each person detected) as value. Other entries of the original JSON are ignored.

Files, i.e. videos and JSON, are organized in a file system.



First access to the database is the number of the camera, that corresponds to a view of the gym. Several routes can be on the same view. Each view has two folders: video, which contains all video belonging to that view, each one stored with its original name, and video_json, which contains one JSON file for each video of the view. The name of the latter file is the same as the

corresponding video (with .json extension). In this way, each video can be automatically loaded on the database without any persistency of metadata about videos (like new names, IDS, etc.) and JSON files. Nonrelational databases like MongoDB could be the best solution for storing the data.

Each gym has to collect its own videos, based on its views and routes. They can be recorded from any source, e.g. smartphones, webcams, security cameras, with the only requirements of fixed point of view. Long videos has to be splitted in a series of videos, where each video correspons to a climb from the beginning to the top. OpenPose can help in identifying when the climber begins the climb, and when (and if) he reaches the top. To recognize the climber on the top can be used a query on the mapped video, seeking the frame where the climber has both the hands on the last holds, and follows the path forward till the beggining of the climb. Eventually, once detected path of the climber of that specific video, data regard other people (detected on the same video) can be removed.

Human Pose Estimation can help in cleaning the raw video: data of people too close to the camera (meaning not on the wall) can be detected using a threshold on the distance between every key points. This latter part is outside of the project scopes.

3.3 Prototype

Before implementing the real application, I did some experiments to verify the basic usage and test some behaviors of the application with respect to real scenarios. The prototype was a stand-alone, single window application. It lets select the view, choose the interesting area, choose the key points (just four in this case) and get all the parts of the videos where the climber has the selected body parts in the selected area.

I present the prototype starting from the Graphic User Interface, then talking about features and technical details.

It is entirely built in Python, using *numpy* package for scientific compu-

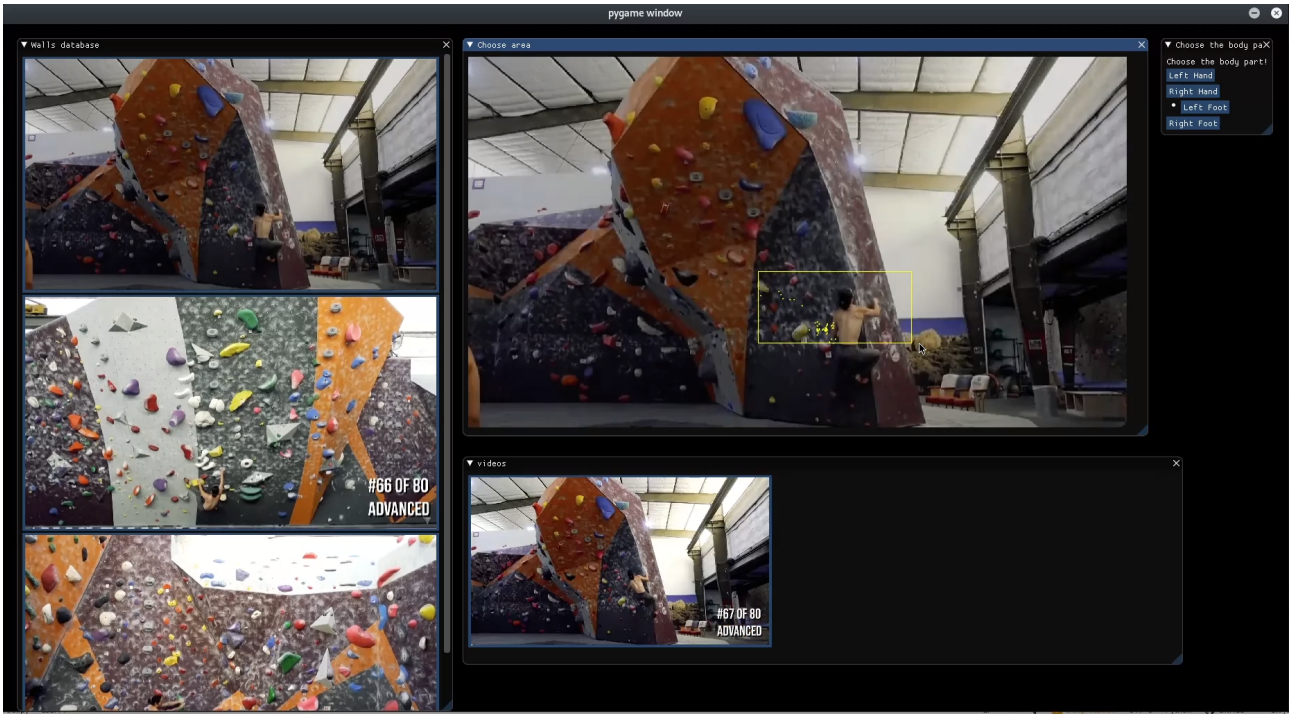


Figure 3.7: GUI of the prototype.

tation (array and matrices operations), *PIL* package for image manipulation, *pygame* and *OpenGL* for building the User Interface. The latter is a set of Python modules designed for writing videogames, but allows to create multimedia programs as well; it can rely on different backends for graphic rendering: I selected the most used one, OpenGL. It requires an initialization with the definition of the size of the interface, the selected backend, the creation of the context, and the render.

```
pygame.init()

size = 1854, 990

pygame.display.set_mode(size, pygame.DOUBLEBUF | pygame.OPENGL)

imgui.create_context()
impl = PygameRenderer()
```

The images (images of the views and thumbnails for retrieved videos) have to be mapped into the TEXTURE variable, required by PyGame for the render of the images.

```
def loadImage(image, widthm):
    textureSurface = pygame.transform.flip(image, False, True)

    ratio = textureSurface.get_width()/widthm

```

```
    width = textureSurface.get_width()/ratio
    height = textureSurface.get_height()/ratio

    textureSurface = pygame.transform.scale(textureSurface, (int(width), int(height)))

    textureData = pygame.image.tostring(textureSurface, "RGBA", 1)

    texture = gl.glGenTextures(1)
    gl.glBindTexture(gl.GL_TEXTURE_2D, texture)
    gl.glTexParameteri(gl.GL_TEXTURE_2D, gl.GL_TEXTURE_MAG_FILTER, gl.GL_LINEAR)
    gl.glTexParameteri(gl.GL_TEXTURE_2D, gl.GL_TEXTURE_MIN_FILTER, gl.GL_LINEAR)
    gl.glTexImage2D(gl.GL_TEXTURE_2D, 0, gl.GL_RGBA, width, height, 0, gl.GL_RGBA,
                    gl.GL_UNSIGNED_BYTE, textureData)

    return texture, width, height, ratio
```

It offers a series of components for the design of the interface: I used boxes, scrollbars, buttons.

At the beginning, the application shows just the vertical box with a scrollbar (fig. 3.7) on the left, containing the set of the available views.

Once the user selects one of them (by clicking on the image itself), the second huge box in the upper-right area (fig. 3.7) is shown. It contains the enlarged selected view. The box on the upper-right corner (fig. 3.7) with selectable body points is displayed as well. Available body points in this first implementation, selectable one per time, are left and right hand, and left and right foot. In the real application, the user can choose among all the 25 body key points detected by OpenPose, more than one per query.

The user can select an area on the enlarged imaged and choose one among the four key points (left hand, right hand, left foot, right foot). Automatically the database is queried in order to get the requested parts of the videos.

Logic part is similar to the final application. The code of the main function of the logic part in the prototype is shown below.

```
for file in files:
        frames = []
        if file.endswith(".json"):
            with open(camera_name+"/videoJson/"+file) as g:
                data = json.load(g)

                for num in data:
                    #print(frames)
                    frame = data[num]
                    #print(num,frame,'\n')
                    #time.sleep(0.5)
```

```
                    if frame[selectedPart]/selected[2]>start[0] and frame[selectedPart]/
    selected[2]<end[0] and frame[selectedPart+1]/selected[2]>start[1] and frame[selectedPart
    +1]/selected[2]<end[1]:
                        frames.append(int(num)) #Collect the frame number where the dected
    part is inside the interesting area
                        date.append((frame[selectedPart]/selected[2],frame[selectedPart+1]/
    selected[2])) #Collect the coordinates for trace later the detected points (Extra
    feature)
if len(frames)>0:
    actual = []
    max = frames[0]
    min = frames[0]
    for g in range(len(frames)):
        if frames[g]>max:
            max = frames[g]
        if frames[g]<min:
            min = frames[g]
```

For each file in the video_json folder (i.e. for each json mapping each video belonging to the view), are selected the frames in which the climber has the selected body part in the selected area; they are collected in an array, storing the number of the frame. If at least one frame is found, the application checks the first and the last frames where the selected body parts were found. The part of the video selected is in between the *min* and *max* frames.

Once videos are found, they are shown in the box on the bottom-left area. There is a thumbnail for each video retrieved, that the user can click to start the video player on new window, displayed using *matpoltlib* package.

Core features tested with the prototype are:

- Query videos in a nonlinear way, exploiting spatial configuration of body parts.
- List all the retrieved (part of) videos.
- View each retrieved video one per time.

An additional feature of the prototype was that it highlights each pixel in which the selected body part was at least once, shown in yellow on the enlarged image (fig. 3.7): the result is a kind of path along the selected area. The feature was removed in the final application, because the selection of multiple key points, introduced in the final application, makes it hard to

highlight each pixel of each keypoint along the path.

Several conclusione comes up after the prototype. First of all, I ascertain that pose and movements of the climber can be efficiently retrieved by selecting the area of interest and the body parts. This lets me go on with the development of the real application.

Testing out the prototype, and using it in a real case scenarios, I realized that the most interesting use is selecting the are between two holds, to find out the right pose between them. Studying one passage per time, it can be repeated for the entire path, defining the entire way to the top.

The prototype lets the user choose just one key point per time, but it can be actually much more interesting query the pose based on multiple key points, e.g. the user may want to get all the videos in which the climber has both the hands over the last hold (meaning that he has reached the top).

The user may want to figure out not just the pose he selected, but the pose right before as well, in order to analyze how to get to that specific pose. To add this features, the retrieved video has to include some frames in which the climber is not yet in the selected area.

Then I've started the development of the final application, starting from the research and consideration done at the beginning and the conclusions that came up with the prototype.

3.4 Frontend

After preliminary research, case studies and prototyping, I started to build the final application. As said before, I choose the server-client architecture: it is a distributed application in which the server component provides functions, services and resources to one or many clients, which initiate requests for such services. Main advantages of client-server model are:

- Centralization: all necessary information are placed in a single location. Access to all the resources is controlled by the server, that redirects the client to the proper data.

- Security: access control can be added to enforce security, and grant access just to authorized users.
- Scalability: whenever the user needs the network can increase the number of resources such as clients and servers.
- Management: since all the files are stored in the central server, it is rather easy to track, find and manage records of required files.

Besides, almost all the computation takes place in the server, without increase the computation demand on the devices (supposedly with far less capacity compared to a central computer).

The communication between two sides, client and server, happens through the means of Rest APIs, whose benefits and features are explained in the section 3.1.

The framework chosen for the development of the frontend is React. It is an open-source JavaScript library for building user interfaces, created and mainteined by Facebook with a large community. Main concern of React is the state management and rendering of that state to the DOM (Document object model, representing the HTML tree-structure of the user interface of web sites); several other libraries are used within React in order to manage routing between pages, style frameworks (e.g. Bootstrap), pre-built components, dynamic page switcher.

React was developed with principles that have become the standard of web programming. First af all, components approach: every element of the dom is an encapsulated component, with its own variables and functions; components can be nested and organized together in order to compose the User Interface. This approach shapes the web page as a components structure, where each one can be easily and dynamically substituted or changed, without compromising the entire page. Such feature lead to the second principle of React: simplicity. It uses a special syntax called JSX, built on top of JavaScript, that makes the rendering logic inherently coupled with other UI logic, such as event handling, state changes, how data is prepared for display. In a way JSX makes able the creation of components that contain both logic and markup

in a single field, making far simpler the management and the control over the entire page behaviour and appearance. Simplicity implies that React is easy to learn, supported from a huge community of developer in the world (React results the second most used language, with a share of 35.9%, preceded only by JQuery). Another important feature of React is the native approach. It can be used to create mobile application (native, as they were build with native languages, meaning without losing any performance), using JavaScript and JSX as programming languages. In this way, it is possible to develope at the same time IOS, Android and web-based native applications. High performance can be guarateed following the several warnings that React includes in case of bad use of memory, components or functions. Moreover, with the deployment the user can select the the production version, that encapsulates and optimizes the code, producing a folder with all the code that can be easily deployed on a web server.

The entire application (frontend side) is configured in the *App.js*, which includes the fixed components (the bootstrap container and the navbar), and redirect the interface to some pages or another based on the current path, using *react-router* package.

```
class App extends React.Component{

  constructor(props){
    super(props)
    this.state = {}
  }

  render(){
    return(
      <>
      <BrowserRouter>
        <Container fluid>
        <Navbars/>
        <Row className="justify-content-center box_views">
          <Switch>
            <Redirect exact from="/" to="views"/>
            <Route path="/views" component={CameraViews}>
            </Route>
            <Route path="/query" component={Query}>
            </Route>
            <Route path="/videos" component={Videos}>
            </Route>
          </Switch>
          </Row>
        </Container>
      </BrowserRouter>
```

```
      </>
    )
  }
}
```

BrowserRoute wrap the entire interface, and enables the dynamic switching between pages, rendering the new components without reloading the entire page.

Each component is defined as a javascript class; they mantain inner variables in a variable called *state*, and, when invoked, can takes properties from parent component through the *props* variable. A special method, called *componentDidMount* is the default method that React uses to launch something as soon as the page is rendered. Every component includes the rendering HTML as *return* argument of the *render* function.

I'm going to present every component of the application, each one corresponsing to a single file.

3.4.1 Navbar

Bootstrap elements are used to model the navbar. It contains only a symbol on the left ,that varies depending on the current page, followed by the name of the project, Nonlinear Climbing Video Indexing (fig 3.8). The boolean value *home*, in the state variable, switch between the "home" button or the "go back" one. The *componentDidMount* function checks if the current path is the homepage, otherwise it changes *home* value to *False*. When shown, the "go back" button calles the *goBack* function, that go back in history queue of the browser.

Figure 3.8: The upper image is the navbar rendered in the home page, the lower is the navbar in the following pages.

The Bootstrap container, the navbar, and a second container defined as a Bootstrap *Row* element are fixed in the page: inner components are removed or added but they remain in the view.

The route component switch between three views:

- */views*: the home page, where different views are shown, i.e. images of the walls in the gym.
- */query*: second page, where the user can select the area in the image, and body key points on a stylized skeleton.
- */videos*: the last page, where there is the list of the thumbnail, one for each retrieved video. It lets the user select one video and scrub frame by frame on it.

When the application is launched, the user is redirect to *views* (fig 3.9). Parent component rendered through the *route* component is the *CameraViews* component.

Figure 3.9: /views page.

3.4.2 CameraViews

Loaded value in the state variable of the component is initialized to *False*. As soon as the component is rendered, the function *componentDidMount* gets all the available images that are exposed by the server, maps them in the right HTML code to be displayed, and changes *loaded* value. It is started a second rendering that add to the page the images. Once the user click on one of the displayed images, the *Link* component (react-router) redirect the user to the new page, */query* (fig,3.10), passing as state value the id of the selected

images.

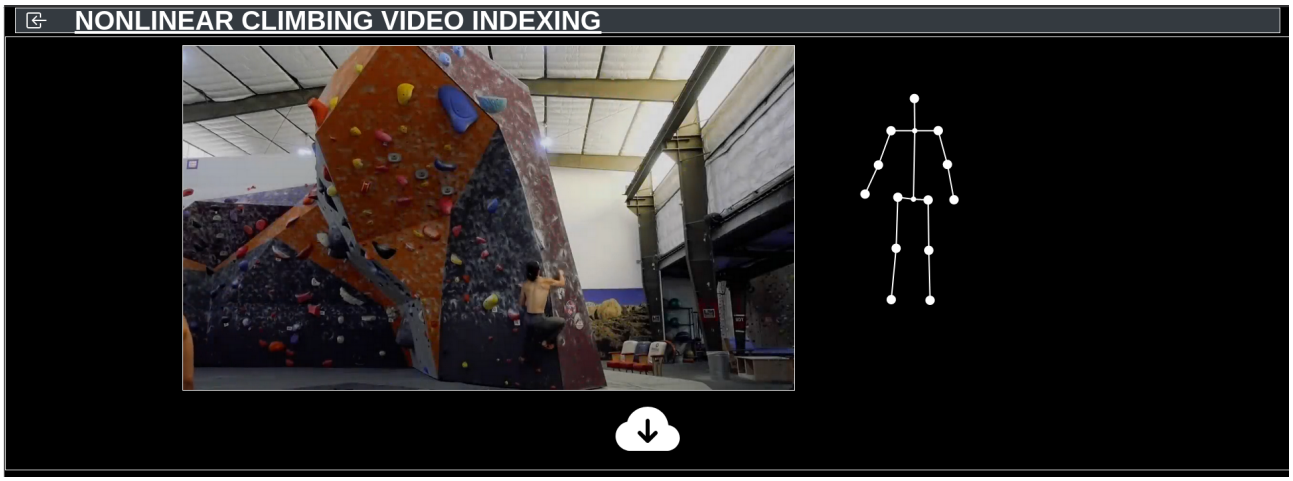


Figure 3.10: /query page.

3.4.3 Query

Query component is rendered when the user is redirect to the query path. It recieves just one variable from the parent components, i.e. the id of the selected image.

The selected image, taken through the id, is shown enlarged (fig. 3.10), and on it is mounted the *RectangleSelection* component. On the left is rendered the *Skeleton component*. A button is used to send the query, with area and selected body parts.

3.4.4 RectangleSelection

```
export default class ReactRectangleSelection extends React.Component {
  constructor(props) {
    super(props);
    this.animationInProgress = null;
    this.state = {
      hold: false,
      selectionBox: false,
      selectionBoxOrigin: [0, 0],
      selectionBoxTarget: [0, 0],
      animation: "",
      found: false
    };
  }
```

Constructor of the component mantain in the *state* variable memory of the following information:

- *hold* : true il the user is currently drawing the box on the image (mouse down on the image).
- *selectionBox* : true if the user has defined a box in the image.
- *selectionBoxOrigin* and *selectionBoxTarget* : contains the coordinates of the box.
- *found* and *animation* : are used to render the box (fig. 3.11).

When the user clicks the mouse on the image, *handleMouseDown(event)* function is called. It resets, if any, the previous box, stores the coordinates of the mouse and trigger the flag *hold* in the state element, in order to save that the user started the selection. *OnMouseMove* property calls another function when the user moves the mouse : if the user is moving the mouse without releasing the click, it simply trigger to True the *selectionBox* value, otherwise, if the mouse down is released, it stores in *selectionBoxTarget* the current position of the mouse. We now have stored the origin and the target point of the selected area. *OnMouseUp* calls a *closeSelectionBox* function, that highlights in red the selected area. The data about the selection are sent back to the parent with a little trick: in the *props* variable, are sent to the child a function that stores in the parent's memory the values of coordinates of the selected box.

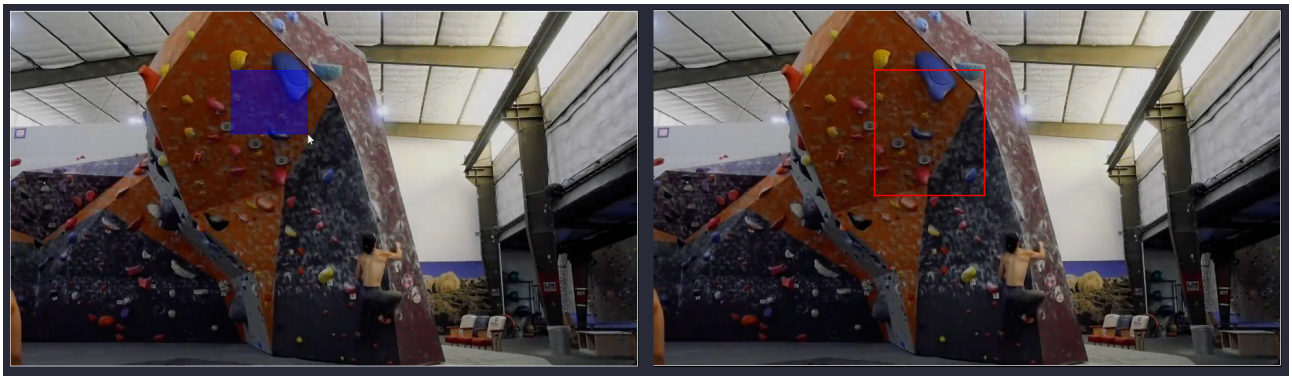


Figure 3.11: On the right, the box shown while the user is still drawing the box. On the left, the box rendered after the selection has ended.

3.4.5 Skeleton

The component initializes each key point in the *state* element:

```
nose: {onClick: (click => this.onclick('nose', this.state.nose)), x: 746/scale, y: 364/scale
    , fill: '#FFFFFF', radius: 7},

left_shoulder: {onClick: (click => this.onclick('left_shoulder', this.state.left_shoulder)),
     x: 540/scale, y: 648/scale, fill: '#FFFFFF', radius: 7},

...
```

Each point contains information about coordinate (parametrized with *scale* variable), color (white if not selected, red if selected), the radius of the circle (7 if not selected, 13 if selected). The *onClick* function changes the color and the dimension of the rendered point, to highlight that the point has been selected (or deselected).

They are linked through limbs, i.e. lines that connect two body key points (fig. 3.12). They are initialized in the *state* element as well.

```
neck_to_rShoulder: { points: [this.state.neck.x, this.state.neck.y, this.state.
    right_shoulder.x, this.state.right_shoulder.y], stroke: this.state.neck.fill },

neck_to_midHip: { points: [this.state.neck.x, this.state.neck.y, this.state.mid_hip.x, this.
    state.mid_hip.y], stroke: this.state.neck.fill },

...
```

The skeleton is drawn with *react-konva* package, which makes available pre-built components that represent geometrical structure. The combination of them can be used to represent more complex figures. Elements used for the skeleton are just circles and lines; every time the user selects a point, the skeleton is rendered again with updated parameters of circle radius and color.

```
<Layer>
 {Object.values(this.state).map(elem => create_point(elem))}
 {Object.values(this.limbs).map(elem => create_limb(elem))}
</Layer>
```

The component *Layer* wrap the components, that will be later rendered in the canvas. Data stored in the *state* variable are mapped with functions that take data of points and limbs (coordinates, color, radius) and maps them in the *Line* and *Point* components.

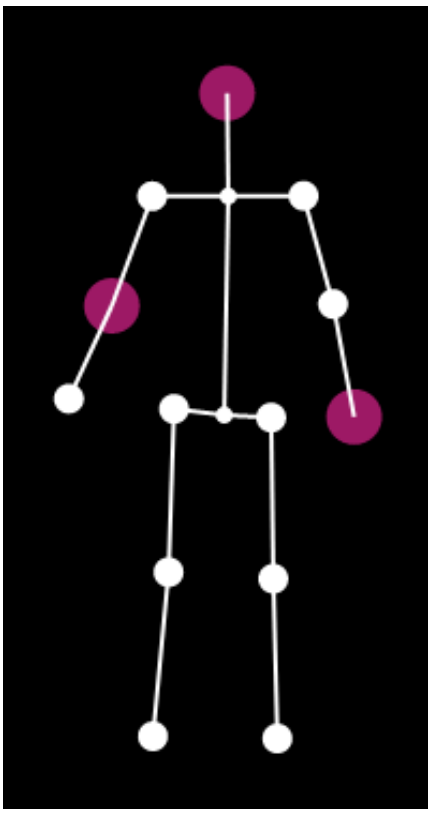


Figure 3.12: Stylized skeleton with key points selected by the user.

Beneath the image and the skeleton, there is the button that sends the request to the server. It redirects the user to the last page, */videos* (fig. 3.13), passing as state the following variables: the id of the selected view, the id of the selected key points, the coordinates of the selected area. Coordinates are normalized taking them with respect to the image, and dividing them by the width and height of the image itself.

Key points and their ids are: nose - 0, neck - 1, right_shoulder - 2, right_elbow - 3, right_hand - 4, left_shoulder - 5, left_elbow - 6, left_hand - 7, mid_hip - 8, right_hip - 9, right_knee - 10, right_ankle - 11, left_hip - 12, left_knee - 13, left_ankle - 14. Other key points are not taken into account since are not useful for climbing pose purposes.

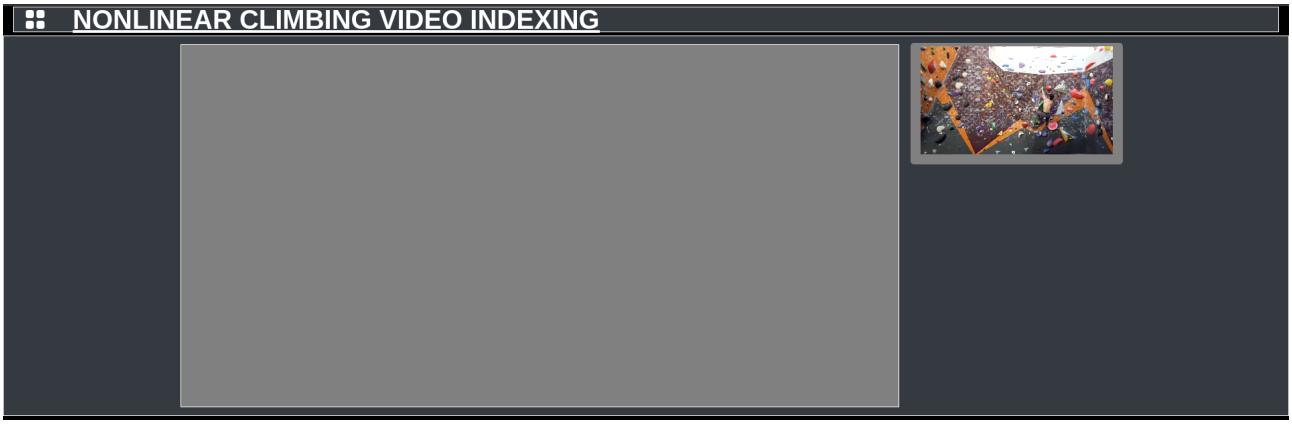


Figure 3.13: /videos page.

3.4.6 Videos

Videos component receives from the previous page coordinates of the selected box and ids of selected key points. if one of the four values defining the box (*x_origin*, *y_origin*, *x_target*, *y_target*) is negative, or the length of the array containing the selected key points is 0, a variable in the state component is toggled to True. While rendering, if any of the variables indicating the error is set to True, an error message is displayed, with a button to go back to the previous page. If there are no errors, at the first render a waiting box is displayed, while the function *componentDidMount* calls the *getVideos* function

from *API* file.

API file contains all the asynchronous functions that communicate with the server, sending the proper HTTP (following the Rest framework). API called by *videos* page at first rendering is *getVideos* function. It takes coordinates of the box, the id of the image, and ids of body key points, and sends the request to *getVideos* endpoint of the server. The request consists of a POST HTTP request that contains in the body all the parameters it has received.

```
async function getVideos(x_origin, y_origin, x_target, y_target, image_name, body_parts){
  console.log(x_origin, y_origin, x_target, y_target, image_name, body_parts)
  let url = '/getvideos'
  return new Promise((resolve, reject) => {
    fetch(url,{
      method: 'POST',
      headers: {
        'Content-Type' : 'application/json',
      },
      body: JSON.stringify({x_origin: x_origin, y_origin: y_origin, x_target: x_target,
    y_target: y_target, image: image_name, joints: body_parts})
    }).then(res => {
      if(res.ok){
        res.json().then(list => resolve(JSON.parse(list)))
      } else {
        res.json().then(obj => reject(obj))
      }
    })
  })
}
```

The response from the server contains a list of maps with the URL of the retrieved video and the frame where the climber enters with the selected key points in the selected area.

Once all the data are fetched and stored in the frontend, the page is rendered again. It contains a wide, void grey box on the left (where later will be displayed the selected video, along with the scrubber), and a vertical scrollbar on the right with all retrieved videos, as images of the first frames of the requested videos (fig. 3.13).

The list is created with a mapping function *create_containers*, that transforms each element of the array returned by the server in a complex HTML component.

```
function create_containers(elem, click){
  return(
        <>
        <div className="container_video" onClick={click} id={elem.url} frame={elem.frame}>
          <div style={{pointerEvents: 'none'}}>
```

```
            <Player src={elem.url} startTime={elem.frame/25} muted={true} playsInline={true}
    preload='auto'>
              <ControlBar disableCompletely={true} className="my-class" />
              <BigPlayButton disabled={true}/>
            </Player>
          </div>
        </div>
        </>
      )
}
```

It takes the element of the array and a function that handles the click event. Most external div manage the information with the parent component (id, frame, onClick function). The *Player* component is imported with the external package *video-react*. I used this package to display the video as an image of the requested frame. The player package takes the URL of the video, which is exposed by the server, the start time, in this case, given by the number of the frame divided by 25 (i.e. the number of frames per second), muted component, set to true. Inside the player component, *ControlBar* component can be used to disable the control bar; the final result is that the video is fixed to the started frame, i.e. the frame in which the climber enters the interesting area with the selected key points.

If the user selected one image, it is called the *onclick* function: it simply changes the state of the component, setting *selected* element to the id of the corresponding video, *frame* element to the frame number. Moreover, it stores in the *state* variable the height, the width, and the duration of the entire video. This information is used later by the scrubber. The page is rendered again, filling the empty, large grey box on the left with the *Scrubber* component (fig. 3.14).

3.4.7 Scrubber

The component receives, under the *props* property, the selected frame, the number of the first frame, height, width, and duration of the video. It stores all the variables in the *state* element, along with a flag variable called *main_extracted*, by default set to False.

After the first render, while it is displayed the loading component, *componentDidMount* function calls the asynchronous function *extractInterest-*

Figure 3.14: Page once the user has selected a video to visualize.

ingFramesFromVideo. Since the scrubber takes an array of images, one for each frame of the video, frames have to be extracted from the video. The *extractInterestingFramesFromVideo* does exactly this: it receives the URL of the video, the requested_frame, the framerate (default set to 25) and gives back an array with all the frames. To do so, the function uses a fake canvas element (that is not attached to the DOM of the page), with the same dimension of the video, to transform frame-by-frame the video into images. I decided, following the conclusions after the prototype to set the central point to the frame where the climber enters into the interesting area, and get 200 frames before that frame, and 200 frames after that frame. This number, even if slows down a little bit the application, since all the extraction is inevitably done on the client-side, lets the user explore the pose starting from some seconds before it.

```
while(currentTime < interval*final) {
        video.currentTime = currentTime;
        await new Promise(r => seekResolve=r);
        context.drawImage(video, 0, 0, w, h);
        let base64ImageData = canvas.toDataURL('image/jpeg', 0.5);
        frames.push(base64ImageData);
        currentTime += interval;
      }
```

Inside our custom scrubber, is included an external *Scrubber* component, from *react-scrubber* package. It takes min and max values, current value, and calls a function *onScrubChange*. When started, the value of the scrubber is at 200, i.e. in the middle, and the first frame is shown in a simple *img* component above the scrubber. When the user changes the value of the scrubber, it is

changed the image shown by the *img* component, rendering from the array the frame at the index corresponding to the new scrubber value (fig. 3.15).

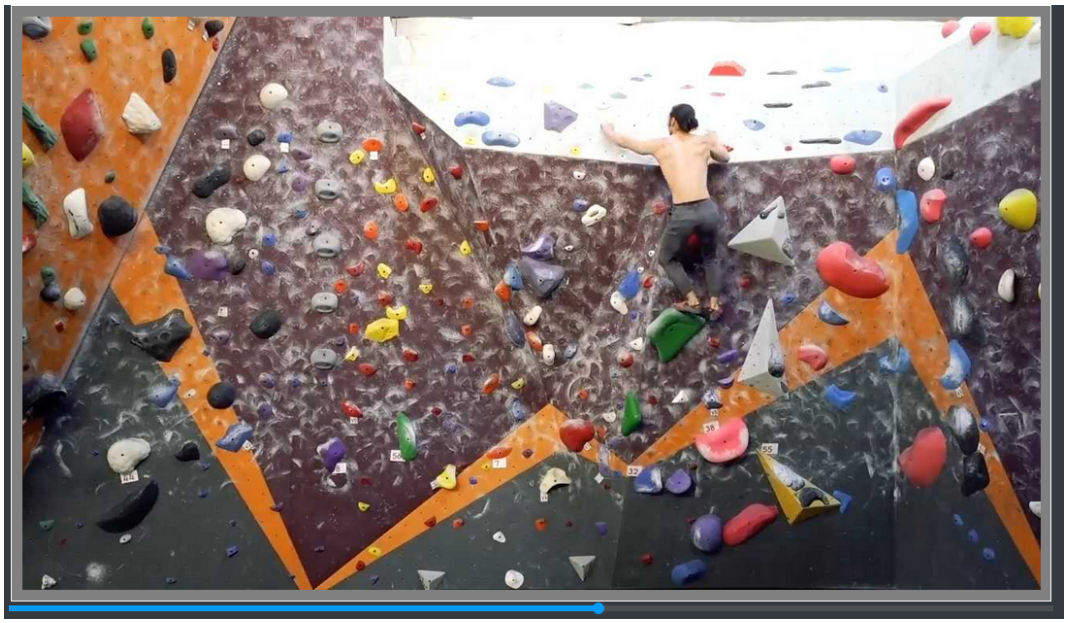


Figure 3.15: The rendered image with the scrubber bar.

The user can select one video per time, explore with the mean of the scrubber the video frame by frame, and figure out how to do a particular pose.

Let's now dive into the backend.

3.4.8 Backend

There are two backends linked together, one the React default server built in Javascript, and a second one built in Python, using the web framework Flask to module the server. The React backend serves and manages the components of the page, by default initialized with the React project. The second one is the server of the application where the logic happens, which exposes the API and media resources.

Flask is a collection of libraries and modules that enable web application developers to write applications. It is very suitable for small projects, that require a robust yet simple server to handle HTTP requests; strengths of Flask are:

- It's easy to set up;
- It's supported by an active community;
- It's well documented;
- It's very simple and minimalistic;
- It's flexible enough that you can add extensions if you need more functionality (not our case).

The *backend* folder contains all the files related to the backend development. It contains the *static* folder, which contains all the media resources that the server exposes to the client: view images and videos. The *venv* folder contains all the resources required by flask, i.e. binaries and libraries.

The *start.sh* script launches the flask environment and the application.

The starting script launches the *api.py* files, that is the file that initializes the server, listens to the requests, and handles them.

The *GetVideos* class contains the API exposed by the server. The *post* function transforms in a JSON object the body of the post request. It takes the id of the view from the json object and builds the name of the folder where are all the resources of that view. For each file in the *videoJson* folder, the function reads the file inside a JSON object, and for each frame of the JSON object checks if all the requested body parts (taken from the body of the request) are in the selected area (taken from the request body as well). The first frame that contains the selected body parts in the selected area is stored in a variable. The variable *url_list* contains all the videos that have at least one interesting frame, storing a couple of pair key-value, containing the URL of the video and the number of the frame.

The variable is eventually encoded as a JSON object and returned as a response to the HTTP call.

```
    HTTP REQUEST

Headers:

    POST /getvideos HTTP/1.1
    Host: localhost:3000
    User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
    Accept: */*
    Accept-Language: en-US,en;q=0.5
    Accept-Encoding: gzip, deflate
    Referer: http://localhost:3000/videos
    Content-Type: application/json
    Origin: http://localhost:3000
    Content-Length: 152
    Connection: keep-alive
    Cookie: _ga=GA1.1.1970342238.1598883523
    Sec-GPC: 1

Request Body:

    {
        image: "camera_2"
        joints: [0: 7, 1: 4]
        x_origin: 0.3423
        x_target: 0.6234
        y_origin: 0.5704
        y_target: 0.7801
    }

```

```
    HTTP RESPONSE

Headers:

    POST /getvideos HTTP/1.1
    Host: localhost:3000
    User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
    Accept: */*
    Accept-Language: en-US,en;q=0.5
    Accept-Encoding: gzip, deflate
    Referer: http://localhost:3000/videos
    Content-Type: application/json
    Origin: http://localhost:3000
    Content-Length: 153
    Connection: keep-alive
    Cookie: _ga=GA1.1.1970342238.1598883523
    Sec-GPC: 1

Response Body:

    {
        "[{\"url\": \"/static/video_1.mp4\", \"frame\": \"211\"}]"
    }

```

Chapter 4

Results and Conclusions

The project started around the idea of combining two common passions of my professor and me: computer science, specifically deep learning, and climbing; Human Pose Estimation was the perfect intersection between both of them. The project was since the beginning the development of a fully-working application, where the user could benefit from the mapping of climbing videos performed by Human Pose Estimation algorithm. With the use of User Stories we defined some common situations that each climber faces during the training phase, and starting from them we defined the main features of the application.

The organization of the database and key points data retrieved by OpenPose was not trivial, since they needed to be stored in a way that facilitated the research (i.e. coordinates of the point included in a certain area). I moved all the JSON files, one per frame, in a unique JSON file, per video. It facilitates the check performed by the Python script called by the API.

The first practical development of the application was the prototype, which demonstrated successfully that climbing videos, mapped with OpenPose, can be queried using spatial coordinates and body parts. It was useful for having a clear picture of the features of the real software as well. Starting from the prototype results, I moved to the final application. since the beginning the idea was about the development of an application with features and architecture that could be used in a real scenario. I structured the application

with a client-server framework, considering the issues that could show up in a practical situation. Flask framework was used to manage the server and Rest API requests. The User Experience was carefully designed: three pages let the user select the proper area on the wall and body points. The retrieved videos are displayed on a scrollbar on the left; once selected, the video is processed to extract the interesting frames. Frames are collected in an array and rendered in the canvas based on the scrubber value. This method makes the scrubber smooth since it does not have to compute the frame on-demand. 400 frames are selected, computed, and stored in the array, using the first frame retrieved by the server as reference point. The scrubber is initialized in the middle, and goes from 0 to 400: each value corresponds to a specific frame.

The result of the project is an application built in React and Python, with the full code open-source. I and my professor did not intend the application to be used for commercial purposes. The main contributions of the project are:

- Amatorial climbing videos can be mapped with OpenPose.
- Client-server architecture, with Rest API communication, can be used to manage the connection between one central database and all the resources.
- Video can be queried using coordinates and body key points.
- The scrubber with pre-loaded frames is an efficient tool for the user to explore the video.

Code is opensource at available on GitHub
(https://github.com/danielekp/nonlinear-climbing-video-indexing).

Regard the application, I did not implemented a way to exploit Human Pose Estimation to cut and select videos to feed the database. Data are supposed to be taken from the gym following the restriction about a fixed point of view. Considering a long videotape, Human Pose Estimation can be used to identify in which part of the video the climber has reached the top, following back

the path in order to identify the started point of the climb; having starting and the top point of the climb the video can be cut and added to the database.

Human Pose Estimation applied to climbing videos can also be used to interactively exploring a long climbing videotape, e.g. competition videos, where a fixed camera records hours of different climbers trying the same routes. The video can be navigated by selecting a particular area and automatically get all the frames where the climber is in the selected in area. It can be used to evaluate different performances and make comparison as well as well.

A final consideration is that all my implementation was based on climbing videos, but Human Pose Estimation offers unlimited benefits to every sports, from training, to analysis and evaluation.

Chapter 5

Bibliography

[1] G. Johansson, Visual perception of biological motion and a model for its analysis, Perception Psychophys.14(2), 1973, 201–211.

[2] T. Calvert and A. Chapman, Analysis and synthesis of human movement, in Handbook of Pattern Recognition and Image Processing: Computer Vision (T. Young, Ed.), pp. 432–474. Academic Press, San Diego, 1994.

[3] N. Badler, C. Phillips and B. Webber, Simulating Humans, 1993.

[4] M. A. Fischler and R. A. Elschlager, The representation and matching of pictorial structures, IEEE Transactions on Computer, 22(1), 1973.

[5] C. Charayaphan and A. E. Marble, Image processing system for interpreting motion in American Sign Language, Journal of Biomedical Engineering, Volume 14, 419-425, 1992.

[6] E. Hunter, J. Schlenzing and R. Jain, Posture estimation in reduced-model gesture input system, 1995.

[7] Randal C. Nelson and Ramprasad Polana, Motion Detection and Recognition Research, https://cs.rochester.edu/u/nelson/research/motion/motion.html, 1990-1995.

[8] Pedro F. Felzenszwalb and Daniel P. Huttenlocher , Pictorial structures for Object Recognition, International Journal of Computer Vision, 2005.

[9] D. Ramanan, Learning to Parse Images of Articulated Bodies, Advances in Neural Information Processing System Conference, 2006.

[10] M. Pawan Kumar, A. Zisserman and P. H. S. Torr, Efficient Discriminative Learning of Part-based Models, 2009

[11] S. Johnson and M. Everingham, Combining Discriminative Appearance and Segmentation Cues for Articulated Human Pose Estimation, 2009.

[12] S. Romdhani, P. Torr, B. Schölkopf and A. Blake. Efficient face detection by a cas-caded reduced support vector expansion, proceedings of the Royal Society, 460(2501), 2004.

[13]

[14] S. Johnson and M. Everingham, Clustered Pose and Nonlinear Appearance Models for Human Pose Estimation, 2010.

[15] Yi Yang and D. Ramanan, Articulated Human Detection with Flwxible Mixtures-of-Parts, 2010.

[16] Yi Yang and G. Mori, Multiple Tree Models for Occlusion and Spatial Constraints in Human Pose Estimation, 2008.

[17] M. Dantone, J. Gali, C. Leistner and L. Van Gool, Human Pose Estimation Using Body Parts Dependent Joint Regressors, 2013.

[18] X. Lan and D.P. Huttenlocher, Beyond Trees: Common Factor Models for 2D Human Pose Recovery, 2010.

[19] G. W. Taylor, R. Fergus, G. Williams, Ian Spiro and C. Bregler, Pose-Sensitive Embeddingby Nonlinear NCA Regression, 2010.

[20] S. Becker and G. Hinton, Self-organizing neural network that discovers surfaces in random-dot stereograms, Nature, 1992.

[21] D. H. Hubel and T. N. Wiesel, Recptive Fields of Single Neurones in the

cat's Striate Cortex, Journal of Physiology, 1959.

[22] D. H. Hubel and T. N. Wiesel, Recptive Fields, Binocular Interaction and functional architecture in the cat's Visual cortex, Journal of Physiology, 1962.

[23] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, Gradient-Based Learning Applied to Document Recognition, 1998.

[24] A. Krizhevsky, I. Sutskever and G, E. Hinto, ImageNet Classification with Deep Convolutional Neural Networks, 2012.

[25] A. Toshev and C. Szegedy, DeepPose: Human Pose Estimation via Deep Neural Networks, IEEE, 2014.

[26] B. Sapp and B. Taskar, Modec: Multimodal decomposable models for human pose estimation, InCVPR, 2013.

[27] S. Johnson and M. Everingham, Clustered pose and nonlinear appearance models for human pose estimation, InBMVC,2010.

[28] W. Ouyang, X. Chu and X. Wang, Multi-source Deep Learning for Human Pose Estimation, 2014.

[29] J. Tompson, R. Goroshin, A. Jain, Y. LeCun and C. Bregler, Efficient Object Localization Using Convolutional Networks, 2015.

[30] Shih-En Wei, Varun Ramakrishna, Takeo Kanade, Yaser Sheikh, Efficient Convolutional Pose Machines, 2016.

[31] Alejandro Newell, Kaiyu Yang, Jia Deng, Stacked Hourglass Networks for Human Pose Estimation, 2016.

[32] Wei Yang, Shuang Li, Wanli Ouyang, Hongsheng Li, Xiaogang Wang, Learning Feature Pyramids for Human Pose Estimation, 2017.

[33] https://pose.mpi-inf.mpg.de/ .

[34] Xiao Chu, Wei Yang, Wanli Ouyang, Cheng Ma, Alan L. Yuille, Xiaogang Wang, Multi-Context Attention for Human Pose Estimation, 2017.

[35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep Residual Learning for Image Recognition, 2015.

[36] Wei Tang, Pei Yu, and Ying Wu,Deeply Learned Compositional Models for Human Pose Estimation, 2018.

[37] Lipeng Ke, Ming-Ching Chang, Honggang Qi, and Siwei Lyu, Multi-Scale Structure-Aware Network forHuman Pose Estimation, 2018.

[38] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, Generative Adversarial Nets, 2014.

[39] Yu Chen, Chunhua Shen, Xiu-Shen Wei, Lingqiao Liu, Jian Yang, Adversarial PoseNet: A Structure-Aware Convolutional Network for Human Pose Estimation, 2017.

[40] Yilun Chen, Zhicheng Wang, Yuxiang Peng, Zhiqiang Zhang, Gang Yu, Jian Sun, Cascaded Pyramid Network for Multi-person Pose Estimation, 2018.

[41] Fang, Hao-Shu and Xie, Shuqin and Tai, Yu-Wing and Lu, Cewu, RMPE: Regional Multi-person Pose Estimation, 2017.

[42] Leonid Pishchulin, Eldar Insafutdinov, Siyu Tang, Bjoern Andres, Mykhaylo Andriluka, Peter Gehler, Bernt Schiele, DeepCut: Joint Subset Partition and Labeling for Multi Person Pose Estimation, 2015.

[43] Eldar Insafutdinov, Leonid Pishchulin, Bjoern Andres, Mykhaylo Andriluka, Bernt Schiele, DeeperCut: A Deeper, Stronger, and Faster Multi-Person Pose Estimation Model, 2016.

[44] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, Yaser Sheikh, Open-

Pose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields, 2018.

[45] Zhe Cao, Tomas Simon, Shih-En Wei, Yaser Sheikh, Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields, 2016.

[47] T.Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan,P. Doll'ar, and C. L. Zitnick, Microsoft COCO: common objects incontext, 2014.

[48] https://github.com/CMU-Perceptual-Computing-Lab/openpose