# A study on the effectiveness of 3D Human Pose Estimators for human-child interaction detection and classification

Rahul Jajodia

Information and Computing Sciences

Utrecht University

Game and Media Technology

MSc. Thesis

Utrecht, 2021

# Table of Contents


**1. Introduction** **1**
1.1. Motivation ……... ….1
1.2. Research Goal and Research Questions……...……... ….2
1.3. Thesis Outline……..... ….3
**2. Literature Review** **4**
2.1. 2D Pose Estimation…….. ….4
2.1.1. Challenges…….. ….5
2.1.2. Techniques…….. ….5
2.1.3. Network Architectures…… ...11
2.1.4. Deep Learning…… ...13
2.1.5. Deep Learning for 2D Pose Estimation……..... ...17
2.2. 3D Pose Estimation…….. ...21
2.2.1. Challenges…….. ...21
2.2.2. Input Types…… ...22
2.2.3. Deep Learning Pipelines…….. ...23
2.3. Action Recognition and Classification……..... ...29
**3. Methodology** **30**
3.1. Our Plan……..... ...30
3.2. Datasets……..... ...31
3.2.1. Human3.6M……..... ...31
3.2.2. YOUth……..... ...31
3.3. Metrics…… ...33
3.4. Modules…… ...33
3.4.1. Instance Segmentation……...…….. ...33
3.4.2. OpenPose……..... ...35
3.4.3. VideoPose3D…….. ...36
3.4.4. Cross View Fusion…….. ...38
3.4.5. Learnable Triangulation of Human Pose……... ...41
3.4.6. VIBE…….. ...42
3.5. A Comparison of 3D Human Pose Estimation Techniques……..... ...43
3.5.1. Preliminary Analysis……..…….. ...43
3.5.2. Evaluation methodology…….. ...46
3.5.3. Evaluation results……..... ...48
3.5.4. Discussion…….. ...51
3.6. Our Approach……... ...51

3.6.1. Color-Matching module v1……..……... ...52
3.6.2. Improved Color-Matching……..…… ...53
3.6.3. Pose Estimation and Lifting…..……..…… ...53
3.6.4. Action Detection……..……... ...54
**4.1. Conclusions 64**
**4.2. Limitations and possible solutions 64**
**References 66**
**Appendix 72**

# A study on the effectiveness of 3D Human Pose Estimators for human-child interaction detection and classification

## 1.1. Motivation

Understanding communicative behaviour in humans has been a widely researched field. Humans learn and exhibit an abundance of verbal and gestural behaviours over their life course and are quick learners when they encounter unknown ones. Evidence has been found that learning communicative behaviour starts as early as the infant phase [Liszkowksi et al. 2004, Boundy et al. 2019, Cameron-Faulkner et al. 2015]. However, it is a long standing debate whether infants use these gestures intentionally and understand the communication process. Many different interpretations have been proposed for gestural use by infants which makes it hard to ascertain and normalize their intent for using a particular gesture. Now, researchers are more interested in knowing what facilitates this learning and how infants progress from simple gestures to more complex cognitive signals.

It is generally believed that infants develop index finger pointing for communication at 8-12 months age. This gestural behaviour has been linked to later language development by a number of past researchers [Bates et al. 1977; Camaioni 1997, Carpenter et al. 1998, Franco & Butterworth 1996]. In a recent work, [Cameron-Faulkner et al. 2015] argued that early gestures like holdouts and gives that emerge in infants prior to pointing [Bates et al. 1977] also contribute to their communication skills and are used declaratively for shared attention. [Cameron-faulkner et al. 2015] also found strong association of these early gestures with later pointing skills. They speculate three reasonings for the influence of these early gestures during prelinguistic development:

1. It provides infants experience with expressing declaratives before combining with distal references
2. It results in an infant-focussed interaction, thereby facilitating infant development
3. It sets up a framework for social and cultural interactions between infants and their caregivers

Later, [Boundy et al. 2019] confirmed that 10 month olds use holdout gestures declaratively as a means of achieving joint attention with their caregiver. Many such behavioural studies have investigated human-human and human-infant interactions. However, most such experiments involve intensive user studies with infants, thus requiring a lot of time and effort, much more so than doing user studies with only adults. Therefore it is only possible to represent a very small subset of the population in any such study. Due to this constrained nature, similar studies often have conflicting observations [Ingram 1974, Bates et al 1975]. For cases with similar observations, many interpretations exist due to a variety of underlying reasons (cultural, social, developmental, etc). To robustly interpret data and draw conclusions that are representative of the entire class, large-scale data collection is needed but the extensive amount of human labor involved in analyzing interaction videos and extracting all the necessary data by hand makes the process dauntingly slow.

Automatic analysis of the nonverbal behaviour could mitigate some of these limitations. For manual coding, studies require a pre-set annotation scheme to mark important events in the video. This annotation scheme enforces consistency among coders. Traditionally, a primary coder annotates all the videos, while one or more secondary coders annotate only a subset. This subset is then compared with the annotations from the primary coder to measure the coding consistency. Each of these steps, although straightforward, adds work to an already laborious task. Automatic analysis not only curtails this, with well-devised techniques it can also extract a much richer information pool than

manual coding in substantially less time. For example, one can easily adopt interpolation procedures to mark missed detections in a frame using its neighbouring frames at little to no extra cost. Automated analysis, however, is not a trivial task. While some nonverbal behaviours may seem simple to detect, such as hand-waving, capturing subtle instances like a turn of the eye can be challenging even for manual coders.

Very recently, 3D human pose estimation has gained plausibility to capture nonverbal behaviors [Luvizon et al. 2018, Pham et al. 2020]. A 3D pose estimate is a view-independent approximation of the human's orientation and position in real world coordinates. Pose estimates can thus provide information vital to detect a variety of non-verbals. In particular, the automatic estimation of 3D poses can greatly aid the pipeline of collecting and understanding human-human, and specifically, human-infant interaction data. On extracting 3D poses for the humans present in the scene frame-by-frame, we explicitly learn the motion sequences of those humans. These motion sequences can be fed to an action classifier to learn the actions being performed. By doing so individually for both humans we can learn the frequency and distribution of all the actions each human performs in the sequence. This data can then be analyzed to isolate action-action pairs, i.e. interactions, using spatial and temporal cues, enabling researchers to understand interaction patterns in humans while saving a bulk of manual labor. The information can be used to see how interaction schemes change over time in humans, thus providing plenty of insight on how interaction schemes evolve in humans as they age. Learning one's action frequency and distribution can also greatly aid in psychomotor analysis of individuals. Estimating 3D poses from image or video is, however, a challenging feat being actively tackled by the scientific community. The inherent lack of depth information in 2D images and videos makes 3D estimation a complex problem. Humans tend to be either partly self-occluded, or occluded by another object or human. This causes frequent information loss. Having additional views can help mitigate this problem but such a setup is not available in many real-world scenarios. Humans also vary greatly in shapes and sizes. Thus is it challenging to detect every human in a scene. Similarly humans exhibit unique behaviour traits that may not always affirm with the laid out annotation scheme. For example, one may point to an object of interest either steadily or only for a quick second. The former case would be more definitive to detect than the latter one.

With 3D pose estimators getting better constantly, establishing a good baseline for their current-day efficacy on action recognition tasks can help identify problem areas, not only for improving action recognition, but also to improve 3D human pose estimators. 3D pose estimation has seen many years of growth and most of the prominent problems have either been solved or are being actively tackled. However, 3D pose estimation is still far from perfect. Discerning the untended problem areas in current 3D pose estimation is, thus, a necessary step for advancement.

## 1.2. Research Goal and Research Questions:

With all that has been said in the previous section, we aim to answer the following research questions in this thesis:

1. What is the current quality of 3D pose estimation in general?
2. What is the current quality of 3D pose estimation for parent-child interactions in the lab?
3. How is classification accuracy affected by (2)?

## 1.3. Thesis Outline

This thesis is divided into 3 parts. The first part (**Section 2**) discusses all the prerequisite knowledge to get the reader up to speed with the latest techniques used for 2D and 3D human pose estimation, and iterates how the tasks have benefitted from Deep Learning. The second part (**Section 3**) discusses our methodology along with all the existing and novel technologies we use in our work to answer our research goals. Finally, the third and last part (**Section 4.1.**) highlights the main conclusions of our work and provides insight for extending this work in the future (**Section 4.2.**).

# 2. Literature Review

This chapter summarizes the past work done in 2D and 3D pose estimation and how pose estimators have evolved using deep learning. It also describes why deep learning is beneficial and demonstrates how deep learning models vary and how these variations affect their performance.

While some deep learning models exist that directly map from RGB images to 3D keypoints [Li and Chan 2014; Tekin et al. 2016], these models require complex training schemes and are challenging to adapt to a different setting. A better approach is to use 2D representations first and then learn mappings to 3D. This motivated us to arrange the literature review in the following way. We start with 2D pose estimation (**Section 2.1**), where we also delve into deep learning, briefly explaining the process and its components. We then discuss novel frameworks for single-person and multi-person scenarios that showcase the power of different deep learning pipelines for 2D pose estimation. Using this base knowledge, we tackle 3D pose estimation (**Section 2.2**), discussing the task and challenges. 3D pose estimation also uses a variety of input settings. We try to make this distinction clear and discuss how different input settings can affect a framework. We also exemplify different deep learning pipelines, explaining the novelties and limitations of each. Finally we discuss the task of action recognition from pose estimates (**Section 2.3**). This involves feeding the extracted 3D poses to a trained classifier for action classification.

**Figure 1:** 2D Pose estimation in the wild [Cao et al. 2019]

## 2.1. 2D Pose Estimation

2D pose estimation is defined as the ability to assign a posed 2D skeleton that matches the 2D pose of a human as projected in a camera's viewpoint. This is achieved by the 2D localization of human joints, i.e., by extracting *(x, y)* coordinates for each joint in an image. These localizations are used to pose a 2D template skeleton which is then regressed, or fitted, into the extracted coordinates.

Estimating 2D poses of humans from input has been a well researched topic now and significant progress has been seen in the area over the years [Papandreou et al. 2017, Cao et al. 2019]. The main task for in-the-wild 2D pose estimation is to attach a 2D posed skeleton to every person present in a scene, preferably in real-time. To do this, a system must be able to extract points of interest (PoI) for every person in an image. These PoIs are generally called keypoints and are associated with distinct human joints like the left ankle, right shoulder, right elbow, etc. The task is in no way trivial.

### 2.1.1. Challenges


**Figure 2:** Challenges in 2D pose estimation: (a) occlusion, (b) missed detections, (c) false positives, (d) false matching, (e) unseen pose [Cao et al. 2019]

There are a number of challenges to be overcome to robustly estimate 2D poses of humans. Humans tend to be either partly self-occluded, or occluded by another object or human (Figure 2(a)). This makes discerning the occluded keypoint locations dubious, often causing artefacts. Human clothing makes the problem worse by hiding the limbs either partly or completely. Most solutions that handle occlusions and clothing rely on assumptions or prior information which may not always be justified. Out-of-view capture is another common problem for similar reasons, where only a part of the human body is visible. Scaling is again a challenge, when some humans are close to the camera while others are far in the background or, as in our case, where adults and infants are present in the same frame. This is a two-pronged problem; First, to detect said humans at different scales. Second, to regress 2D skeletons to the respective scales. Further, having people in close proximity can make segmentation and keypoint detection a very challenging task, often resulting in missed detections (Figure 2(b)), or false positives (Figure 2(c)), or false matchings. A false matching is when a keypoint belonging to one human gets associated with the skeleton of another human instead (Figure 2(d)). Detecting keypoints is itself a difficult task because humans are articulated, making them capable of composing a very large plethora of poses, only a subset of which is addressed by any available system (Figure 2(e)).

A number of these challenges have been addressed, and are still being actively tackled. Other common challenges pertaining to the input in object segmentation tasks like variations in viewpoint and/or lightning, lens distortions, image resolution and image projection are all inherent to 2D pose estimation as well but have been largely tackled due to extensive research in the area.

### 2.1.2. Techniques

The techniques used to detect people and extract keypoints can be categorized into two components: a region proposal system and feature descriptors. These components are not exclusive of each other, and, more than often, a combination of both a region proposal system and feature descriptor is used.

#### 2.1.2.1. Region-proposals:

Region proposals involve extracting cropped regions of the image that seem relevant to the task, also called regions of interest (RoI). This acts as an attention mechanism for the network, saving the need to process entire images, and instead, focus on only the relevant pixels. These proposals are then fed to a neural network for feature extraction, regression and/or classification. Region proposals, thus, segment an image to capture instances of humans in it. All segmentation techniques (Figure 3) fall

under one of two categories: semantic segmentation [Arbeláez et al 2011, Chen et al. 2015] and instance segmentation [Hariharan et al. 2014, He et al. 2017].

(a) Input Image (b) Semantic Segmentation (c) Instance Segmentation

**Figure 3:** Image segmentation techniques[1]

a. Semantic Segmentation: In this, an image is partitioned into multiple independent parts so as to assign a class label to every pixel in an image. Multiple objects of the same class are treated as a single entity. This limits the ability to fit a 2D skeleton to every human in multi-person settings, making semantic segmentation feasible for only single-person settings.

b. Instance Segmentation: This consists of delineating objects of interest in an image. This takes semantic segmentation to the next step such that every instance of a class is treated as a separate entity. Thus, this is more ideal to create proposals in multi-person settings.

Many systems exist to generate region proposals. Some are obviously better than others. To better understand how these systems can differ and affect task performance, we discuss a few of them below:

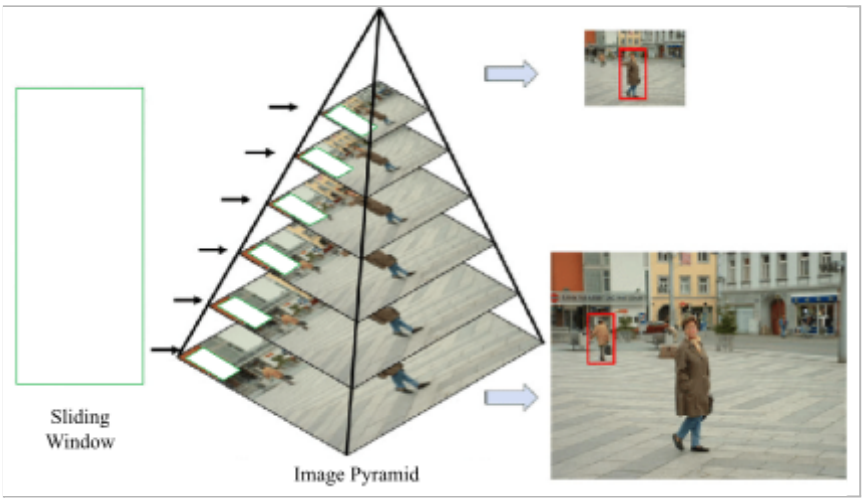



**Figure 4:** Example of a sliding window approach with an image pyramid to create a pyramid of filters for person detection [][2]

- *Sliding Window:* This uses a very simple brute-force algorithm. A fixed size window is iterated to cover every location of an image including overlaps, wherein, at each location features are extracted independently. If the proposed window passes a user-set task-specific

[1] Image retrieved from: https://ccvl.jhu.edu/datasets/
[2] Image retrieved from: [Suleiman & Sze 2015]

threshold, it is accepted and classified, otherwise it is discarded. Note that the fixed window size makes it hard to detect objects of varying sizes. Using a sliding window approach also results in a very high number of overlapping detections which are then post-processed to ideally produce only one detection per object. Non-Maximum Suppression is a common filtering technique for this, which picks the detections with the highest confidence score among detections with high overlap. The final output is an image with detected objects and their bounding boxes. What makes sliding-window effective is its ease-of-use in capturing almost every instance of a class using only brute-force without severe localization errors. However, having to scan every single location results in a lot of wasted computations. This only gets increasingly worse for detecting objects at varying scales. One may use multiple sliding windows with different scales and aspect ratios [Felzenszwalb et al. 2010] or an image pyramid [Dalal et al. 2005] (Figure 4) (**Section 2.1.2.2.**), or even both. In all cases, new feature maps are produced for every window location for every scale independently. This severely increases the already demanding computational cost, making the sliding window approach impractical for real-time performance.

- *Anchor boxes:* An extension of the sliding window approach, called anchor boxes [Ren et. al. 2015] (Figure 5), was later developed to overcome the limitations of using sliding windows. Anchors are a set of boxes with predefined locations and scales relative to the image. This allows binding features of a proposed region to its raw image location at no extra cost. Thus, the novelty of anchor boxes over the sliding window approach is that anchor boxes address scales at no extra computation cost; instead of using multiple feature maps it uses the same feature map to generate multiple proposals at varying scales, acting as a 'pyramid of anchors' as opposed to a 'pyramid of filters' (Figure 4). Further, anchor boxes are guaranteed to be translation invariant; the same proposal is generated for an object that has translated. Recent state-of-the-art object detectors incorporate anchor boxes [He et al. 2017, Redmond & Farhadi 2018] in their framework, demonstrating the effectiveness of the technique.

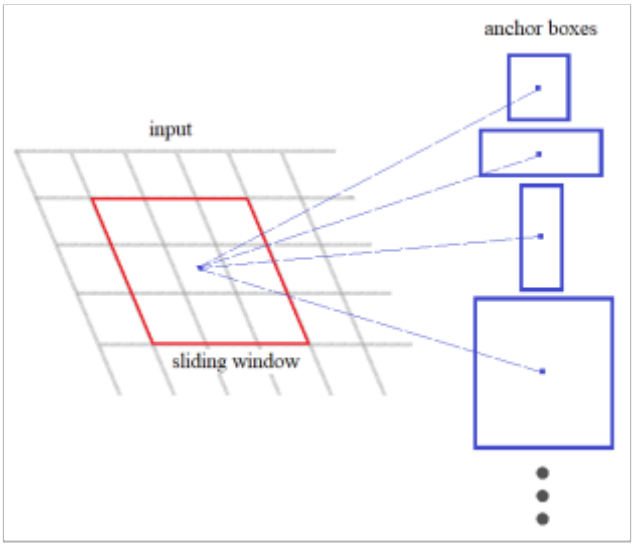



**Figure 5:** Anchor box proposals

Region proposals, thus, allow a network to concentrate on a part of the image. By proposing regions of interest and limiting the range of search for subsequent steps in a network, it also saves a

lot of computational cost. Next, the required information is extracted from the pixels encompassed by each region of interest. A number of ways exist to extract and represent this information; these techniques are collectively termed as feature descriptors.

#### 2.1.2.2. Feature Descriptors:

Feature descriptors represent an image or part of an image by extracting useful information and throwing away extraneous information. What is useful and extraneous is very task-dependent; in our case, we are only interested in extracting humans. Over the years some clever feature descriptor designs have originated that greatly aid the task of pose estimation. Let us discuss some such feature descriptors.

(a) Input Image (b) Histogram of Oriented Gradients

**Figure 6:** An example of HOG[3]

- *Histogram of Oriented Gradients (HOG):* HOG (Figure 6) is a feature descriptor that became very well-known after [Dalal and Triggs 2005] used it in their work to encode the local appearance and shape of people in an image by using local intensity gradients. First, image gradients are computed at the pixel level and the magnitude and direction is recorded per pixel. After this, the image is divided into 8x8 pixel cells. Here, each cell now holds a histogram with the 64 pixel gradient magnitudes and directions cumulatively binned. Next, a 16x16 pixel block (so 2x2 cell) is slid across the entire image to concatenate histograms of 4 cells into a 1D vector which is then normalized. Lastly, the block vectors and concatenated to form the final 1D HOG feature vector. HOG features are invariant to small local deformations and easy to visualize, making it a good tool for person detection and subsequently pose estimation [Danton et al. 2013, Gkioxari et al. 2014]. However, HOG features rely on raw pixel values to compute the image gradients, and thus, are not invariant to lighting. HOG fails when the pixel values of an object-of-interest closely match the pixel values of its surroundings.

[3] Image retrieved from: https://iq.opengenus.org/object-detection-with-histogram-of-oriented-gradients-hog/

(a) PAF for limb connecting right elbow and wrist (b) Encoded 2D vector field

**Figure 7:** An example of Part Affinity Fields [Cao et al. 2019]

- *Part Affinity Fields (PAF):* PAF [Cao et al. 2017] (Figure 7) was developed to capture and preserve both the location and orientation information for each limb in an image using 2D vector fields in a bottom-up fashion. Specifically, for every body part *j1* and *j2*, if a point lies on the limb connecting the two parts, it holds a unit vector pointing from *j1* to *j2* and zero otherwise. Each PAF encodes a distinct limb for all persons present in the scene such that the set of all PAFs encodes every limb for every person in the scene. While [Cao et al. 2017] use PAFs exclusively for human body keypoint annotation, they display its generalizability by performing a vehicle keypoint detection task and achieving a sufficiently high AP. Many researchers have since adopted PAFs to develop pose estimators [Ning et al. 2017, Li 2019], with some even incorporating multi-scale features [Jin et al. 2017, Kreiss et al. 2019] to allow better inferencing for people of smaller scales. In the original work, [Cao et al. 2017] did not use multi-scale features which served as a limiting factor for people of smaller scales; top-down methods (**Section 2.1.3.2.**) at the time like [Papandreou et al. 2017] allowed upscaling patches of detections which aided in detecting people at smaller scales.

**Figure 8:** Different types of heatmaps (left) A heatmap for the left wrist [Wei et al. 2016] (right) a heatmap marking the left shoulder for every person [Cao et al. 2019]

- *Heatmaps:* Heatmaps (Figure 8) are very commonly used for human pose estimation in deep learning pipelines. They are also sometimes called confidence maps or proposal maps. Heatmaps are used to represent the joint locations in an image while preserving location information. Each pixel value is either a 0, denoting the absence of the relevant joint or a positive value to denote the presence such that the peaks in a heatmap coincide with the 2D joint locations in the raw image. Obtaining the peak locations, however, is highly dependent on the heatmap resolution and can cause very large offsets if the heatmap is too small. Using

very large heatmaps is also not practical as it substantially increases the memory cost of the system.

Most encoder-decoder networks (**Section 2.1.5.2.**) use heatmaps, although the exact implementation varies. A single heatmap may be used for multiple keypoints but it is not very practical because it makes occlusions and close-proximity cases very difficult to manage and needs additional post-processing. Heatmaps of much higher resolution could be employed to curtail some of those issues but that greatly increases the memory cost of the system. Instead, researchers usually generate a separate heatmap for each keypoint [Newell et al. 2016, Chen et al. 2017, Wei et al. 2016]. An alternative implementation is seen in [Cao et al. 2019] where heatmaps are used to detect a common keypoint for all human instances in the image. This approach however requires additional post-processing to assign detections to the individual human instances. This post-processing is an NP hard K-dimensional graph matching problem and is typically solved using an integer linear programming solver which is computationally very expensive. In another implementation [Luvizon et al. 2018] use volumetric heatmaps, where heatmaps are extended to 3D, to create a unified 2D and 3D pose estimation technique using camera projection matrices.

Often, feature descriptors are used in combination to mitigate their individual limitations. There are some additional techniques that aid in improving results for pose detection and estimation. These techniques are independent of the task and are employed to improve results in many other domains of image processing.

a. *Image Pyramid:* This is the most common technique used to incorporate multi-scale features. Using multi-scale features increases the scale invariance of a network. The idea is to recursively blur and subsample an image to create smaller variants and then extract features at each scale. This creates a pyramid of feature maps (Figure 4). By tracking at which scale an object is detected, localization is used to segment it in the original-sized image. In most applications, using an image pyramid improves the network. Incorporating multi-scale features helps in improving detection scores for people at varying scales, and keeping track of the scale in which the detection was made allows for more precise skeleton fitting. However, it can add a substantial computational load because every scale is processed independently of each other.
b. *Optical Flow:* Optical flow can only be used when temporal information is available. The idea is to locate every pixel from one frame in the subsequent one. Pixel displacements are denoted using a 2D vector field. Optical flow works on a major assumption that the intensity of a pixel under motion does not change between consecutive frames. This is generally true but causes discontinuities at the boundaries of objects or when camera movement is allowed. There are two forms of optical flow. First is sparse optical flow which only displays vector fields for certain chosen pixels in an image. Second is dense optical flow which extracts the flow vector for every pixel in an image. Consequently, dense optical flow is more accurate but has a much higher computational cost. Optical flow is useful for pose tracking as the movement of humans is naturally encoded between subsequent frames of a video sequence, thus, allowing more accurate localizations. For every frame, adjacent frames can be used to better predict part and joint locations [Zuffi et al. 2013].

These techniques are carefully combined and arranged by researchers to form the pipeline for a network. Depending on the arrangement and techniques used, networks can vary greatly in

performance, both in terms of accuracy and speed. The exact pipelining of a network is what is referred to as that network's architecture.

### 2.1.3. Network architectures

For 2D pose estimation, many architectures exist but a big distinction is observable between single-person 2D pose estimators and multi-person 2D pose estimators.

#### 2.1.3.1. Single Person

Single person estimators know exactly the number of keypoints (except in out-of-view instances) to be regressed and work relatively well. These systems conventionally require pre-processing such that a crop of a single-person is fed to the system. Thus, the position of the person is always known to the system. A robust network would need to account for self-occluded or partial-view cases (including occluded by an object). The most common pipelines directly regress keypoints from the learned features. Another common approach is to generate heatmaps to infer keypoints. The above two types are called regression-based networks. An alternative is to use detection-based networks, which use deep neural networks for body part detections instead of regressing keypoints for joint locations.

#### 2.1.3.2. Multi-Person

Multi-person estimators are more challenging to build because they have no prior information about the number or positions of people. This is typically solved with an initial person detection step. This means that the network requires not only 2D keypoint detection but also robust human segmentation. Such a network needs to account for self-occluded and partial-view scenarios as well as all challenges pertaining to multi-human segmentation (occlusions, scaling, viewpoint, proximity, etc). For solving 2D poses for multiple people, the basic methodology can be categorized as either top-down or bottom-up. Both methods are complementary, each excels where the other fails. Generally, top-down approaches are more accurate than bottom-up approaches for 2D pose estimation but are also far more time-consuming.

*Top-down:*

Top down approaches (Figure 9)take a complex problem and hierarchically analyze it by breaking it down step by step. These approaches have all the required cognitive responses already preprogrammed in its knowledge base. For 2D pose estimation, a simple top-down approach would be to first detect every person in an image and then individually estimate their keypoints using each generated bounding box. This also means that top-down approaches keep getting slower as more and more people appear in the scene, however this scaling is usually linear, and thus computationally affordable. A huge advantage of top-down approaches is the ability to disassemble the task into simpler problems, for example, detecting bounding boxes using an object detector and then running single-person pose estimation on each box [Papandreou et al. 2017, Ning et al. 2019]. This allows the different steps to be fine-tuned independently as knowledge is not shared among steps, making the network more intuitive to train. This approach also allows easier integration of existing technology, for example, using an off-the-shelf object detector for the person detection step.

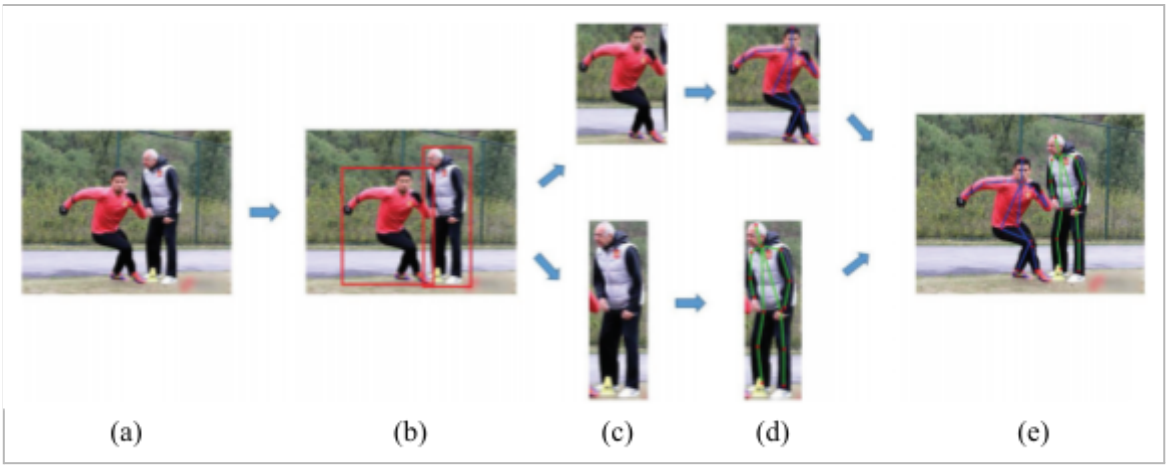



**Figure 9:** Top-down pipeline example [Dang et al. 2019] (a) input image, (b) detect instances (two in this case), (c) crop instances, (d) run single-person pose estimation, (e) combine results


**Figure 10:** Bottom-up pipeline example [Dang et al. 2019] (a) input image, (b) detect all keypoints, (c) associate detected keypoints to form human instances

*Bottom-up:*

Bottom-up approaches (Figure 10) by design consist of simpler components connected to solve a complex problem while building a common knowledge base for all the components. To build this common knowledge base, bottom-up approaches are commonly trained end-to-end. For 2D pose estimation, a simple bottom-up approach would first detect keypoints for all instances in the image and then subsequently assign them to corresponding instances [Cao et al. 2017, He et al. 2017]. This parallelization allows bottom-up approaches to scale generously with the number of people appearing in the scene. The parallelization however limits the ability to allow extra refining of individual instances that may be problematic, such as instances of very small scale. Thus, typically bottom-up approaches offer an excellent trade-off between estimation accuracy and computational cost. A major drawback of bottom-up approaches is that controlling and improving individual components can be very daunting without full knowledge of the system.

*Combined:*

Recently, in an effort to overcome the pros and cons of both architectures, researchers have explored combining both top-down and bottom-up methodologies [Kuo et al. 2011, Newell et al. 2016]. The most famous example is the stacked hourglass detector [Newell et al. 2016] which was developed for 2D pose estimation. Given an RGB image containing a single person, features are progressively pooled down to a very low resolution after which they are progressively upsampled and

combined across multiple resolutions. This gives the network layers and its feature maps an hourglass form. The final outputs are a stack of heatmaps, where each heatmap marks the position of a distinct keypoint, i.e. joint. Normally, the hourglass module is run recursively to allow the network to learn from the heatmaps of the previous iteration, a process generally referred to as multi-stage training. This allows for repeated bottom-up, top-down inferencing across scales. The network thus captures keypoints very robustly across scales without using complex multi-scale mechanisms and achieved state-of-the-art results on the FLIC [Sapp et al. 2013] and MPII Human Pose [Andriluka et al. 2014] benchmark datasets in the past.

A Feature pyramid network (FPN) [Lin et al. 2017] is another very common example that uses both bottom-up and top-down branches for more precise multi-scale inferencing while maintaining a tolerable computational load. Using an initial feature map, the bottom-up branch downsamples it by a factor of 2 at every level while making it semantically richer using convolutional modules. The final, richest feature map is then upsampled iteratively using a top-down branch. Upsampling however can cause serious delocalization issues. To mitigate this, lateral connections are added between maps that match in resolution from the bottom-up branch to the top-down branch. This allows more accurate location inferencing of the features. These connections also act as skip connections, helping with the vanishing gradient problem (**Section 2.1.4.1.**). Each upsampled map is merged with the corresponding map from the bottom-up branch and corrected for aliasing effects to finally output a pyramid of feature maps. This pyramid is then used as input for object prediction, with different resolution maps for different scales. Note that the map at the lowest resolution of the top-down branch requires no merging.

Irrespective of the architecture, deep learning has been found to be the most productive approach to robustly estimate poses. Deep learning scales linearly with the amount of training data, and the availability of large amounts of high-quality data in recent years has greatly supported deep learning frameworks. Thus, deep learning is ubiquitous in pose estimators.

### 2.1.4. Deep Learning

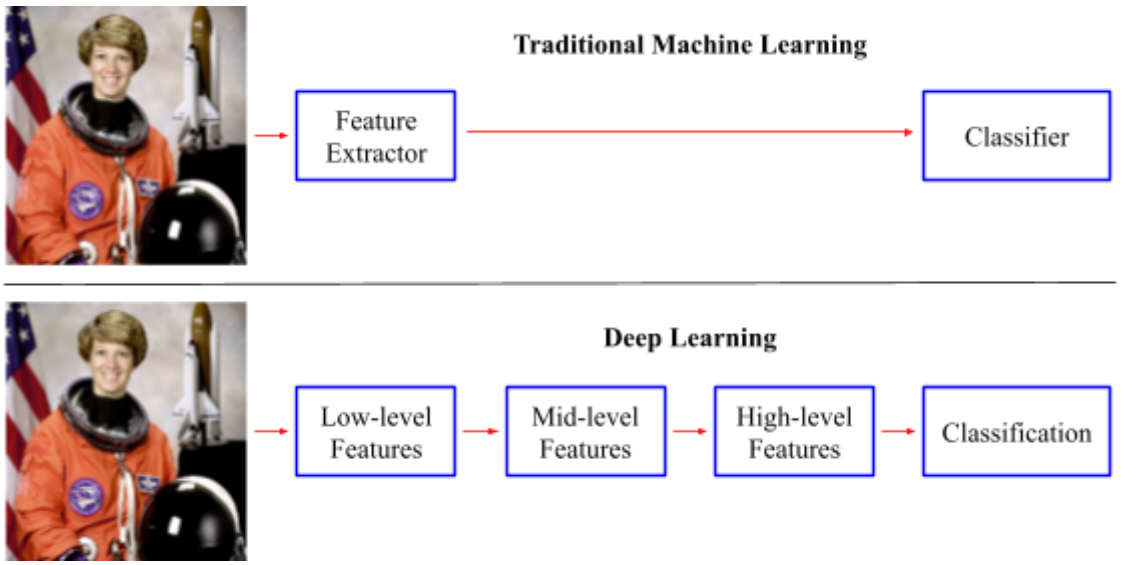



**Figure 11:** Deep Learning uses hierarchical representations

Deep Learning is a machine learning technique used to represent a concept as a nested hierarchy of concepts that are often abstract representations (Figure 11). Deep learning architectures, termed DNN for Deep Neural Networks, use a strata of convolutional layers in series and usually end in one or more fully-connected layers along with some activation function to produce the required outputs (heatmaps, classification scores, bounding boxes, etc). The initial layers learn very simple representations like recognizing edges and curves, called low-level features. These features are then combined by the network in the subsequent layers to learn to recognize more complex shapes like polygons and circles, called mid-level features. Further yet, the last few layers combine these mid-level features to learn to recognize much more complex shapes like body parts or human faces, referred to as high-level features. Usually, features become more and more abstract for human understanding deeper down the network. Finally, output is produced by assembling the high-level features to form task-dependent entities using fully-connected layers.

#### 2.1.4.1. Learning in DNNs

Deep learning is very practical as long as large amounts of good-quality data is available for the task. Traditional machine learning approaches need intervention to identify features relevant to the task. DNNs learn all this information on their own using end-to-end training. End-to-end systems use a single DNN to solve a complex problem rather than using multiple separate systems that specialize in a particular task in the pipeline. This makes them easier to train and use. Given enough variation (rotation, scale, position, lighting, etc) in the training data, DNNs automatically learn to be tolerant to the conditions. Trained DNNs are also very good at filling holes in tasks that have been modelled into the network. This is especially true for partial-view or partially occluded instances. Finally, DNNs are computationally very demanding to train due to the very deep pipelines with millions of learnable weights, and use a slow, iterative learning process. With emerging technology and computers being faster than ever, what seemed like an impossible task before has been made feasible to a good degree.

DNNs [Krizhevsky et al. 2012, Simonyan and Zisserman 2014, He et al. 2015, Szegedy et al. 2015] are conventionally trained over tens or sometimes hundreds of iterations to reach convergence, i.e. to achieve a steady and accurate performance. A user only directly controls the input and output layers in DNNs. All the intermediate layers are termed as ‘hidden’, because their weights are unknown to the user. Instead they are tuned and calibrated automatically using the training data through a process called backpropagation. A user only needs to set the hyperparameters for the network, i.e. network dimensions, learning rate, etc., and then initialize the weights to let learning ensue.

*Backpropagation*:

During training, a loss-function is applied to the network output to penalize discrepancies between the generated output and the expected output. This discrepancy is called the loss term. The loss term influences only the last layer directly. To be able to propagate this loss to all the previous layers to calibrate their weights, backpropagation was introduced. The exact process is mathematically complex, but it essentially computes gradients for every weight in the weight space with respect to the loss function and adjusts it accordingly to decrease the net loss. Note that backpropagation is not a learning method but a gradient computing technique and is usually used in combination with an optimizer. A common problem with gradient based techniques, called the *vanishing gradient problem* [Hochreiter 1998], is that for very deep networks the gradient w.r.t the loss term becomes negligible before reaching the initial layers, making backpropagation ineffective. This problem only gets worse

as network depth increases and was the biggest bottleneck in deciding the depth of a DNN until it was cleverly solved in the residual network (ResNet) [He et al. 2015] architecture using residual connections (more on this under layer design below).

*Loss functions*:

Loss functions calibrate a network's output with the expected output. Loss functions vary from network to network and are dependent on the nature of the task. For keypoint detection tasks, ground truth 2D annotations are most commonly used. These annotations are compared with the network's presented 2D positions and a loss-term is calculated on the discrepancies using a distance measure. This loss term then drives the backpropagation in the network to reduce the net loss value as training progresses. Some common loss functions include the mean square error (MSE), also called L2 loss used by regressors, and the cross entropy loss used by classifiers.

*Learning Rate*:

Learning rate is the factor by which weights are incremented or decremented at every iteration during backpropagation. Most networks start with a larger initial learning rate and then decrease it as learning progresses. A larger learning rate allows a network to approach convergence faster in the initial iterations. Later iterations make model performance more stable by using smaller learning weights. Learning rates are controlled with the help of optimizers.

*Optimizers:*

Optimizers are used to control the learning rates for the weights throughout the network so that the network reaches convergence faster. Optimizers are important because both small and large learning rates can be detrimental during training. Small learning rates can really slow down learning in the initial iterations whereas large learning rates can cause overshooting and prevent a network from ever reaching a local or global minimum. Today, all optimizers in use are based on gradient descent. The most common variants are Adam [Kingma et al. 2014] and RMSProp [Hinton et al. 2012]

*Activation function:*

Activation functions are used to add non-linearity to a network. Without this, the output of a network forms a pure linear combination of the layers, which means that all the existing layers could be collapsed into a single one. This severely limits a network's learning capacity. Many activation functions exist that are commonly used like tahn, sigmoid, softmax, ReLu, etc. It is important to know individual strengths and weaknesses for the activation functions so as to use them in the correct scenarios. For example, softmax assigns a multinomial probability distribution, and thus is used in the output layer for multi-class classification problems to retrieve a probability distribution over predicted output classes.

*Hyperparameters:*

Hyperparameters refer to the number of layers, layer dimensions, layer arrangement, loss function, activation function, optimizer, and any other user-picked parameter in a neural network design. Most changes in a network's architecture can be linked to changes in hyperparameters.

*Layer designs:*

Among choosing the hyperparameters also lies the decision on the different layers to be used in a DNN. Conventionally, all layers used to be convolutional layers, with each followed by pooling,

normalization and an activation unit with the last layer being a fully connected layer to drive the output. Recently some clever layer designs have emerged called blocks or layer modules which parallelly process the same feature maps from the previous layer to produce different activations, which are then concatenated to form the module's output. This can even help in encoding multi-scale features.

For very deep neural networks, the general intuition is that adding layers should not worsen performance. However, naively adding layers has been constantly observed to degrade performance in practice, an issue commonly termed as the degradation problem. This, along with the vanishing gradient problem, made very deep neural networks impractical until residual blocks (Figure 12) were introduced with ResNet [He et al. 2015]. These blocks introduced shortcut/skip/residual connections which allow a layer to feed its output not only to the next layer, but also to layers that are further away. This lets the network decide the exact mapping of the layers, thus tackling the degradation problem. Additionally, the connections allow gradients to flow unimpededly, also tackling the vanishing gradient problem.

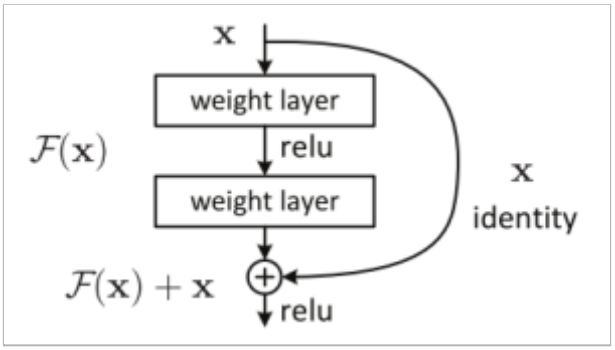



**Figure 12:** A residual block with a shortcut connection [He et al. 2015]

With residual blocks, ResNets were created with many variants (50, 101 and 152 layers deep) and it was shown that residual connections helped the deeper networks outperform their shallower counterparts. ResNet has since been adopted as the backbone in many image analysis tasks, including for 2D pose estimation [Insafutdinov et al. 2016, He et al. 2017, Sun et al. 2017, Xiao et al. 2018, Papandreou et al. 2018, Kocabas et al. 2018, Kreiss et al. 2019]. The most commonly used variant is the ResNet-101, where 101 defines the network depth, which provides a good balance between accuracy and computation load. Many authors also borrow the idea of residual connections in their own network designs [Newell et al. 2016, Nie et al. 2018, Lin et al. 2017] to improve learning. One such extension of the ResNet module is the DenseNet [Huang et al. 2017].

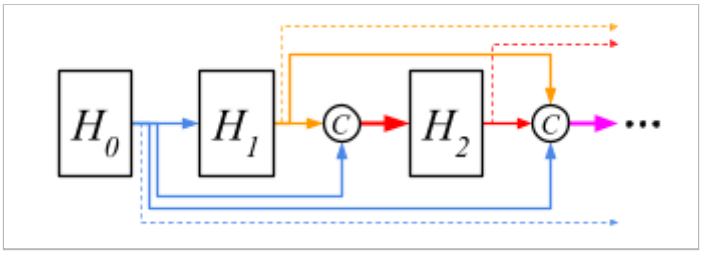



**Fig 13:** A DenseNet module example. $H_i$ represents weight layers and $C$ is the concatenation.

In the traditional ResNet, the output from one module is added element-wise with only its own input before being passed onto the next module. In contrast, a DenseNet layer obtains additional inputs from all preceding layers and concatenates them before processing them to obtain its own feature maps. Because the network uses knowledge from all previous layers, a DenseNet layer with fewer output channels can easily match the accuracy of a ResNet while having a higher computation and memory efficiency (Figure 14). The number of non-linearity layers is increased exponentially and the network is allowed to keep both lower and higher level features. DenseNet modules (Figure 13) are used in the original work for PAFs to leverage a larger receptive field. A receptive field is a measure for the portion of an image that is covered by a feature descriptor. A large receptive field means that a large portion of the image is "looked" at.

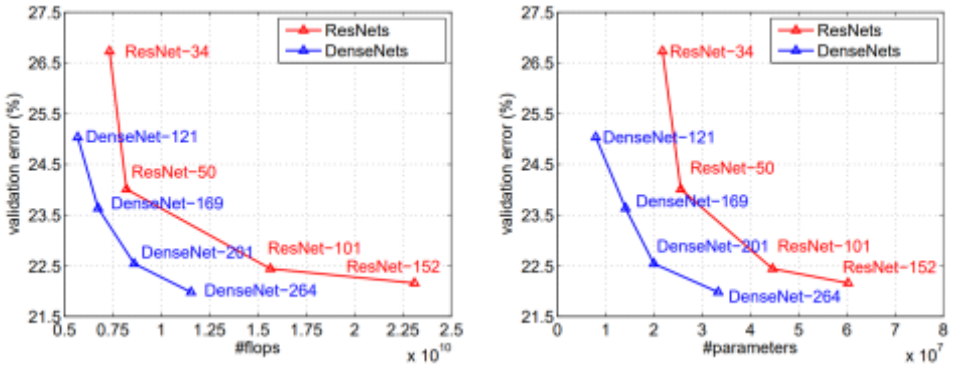



**Fig 14:** A performance comparison of DenseNet-*x* and ResNet-*x* architectures, where *x* is the number of layers in each [Huang et al. 2017].

Another novel layer design that emerged in the recent years are the transposed convolution layers, also sometimes called deconvolutional layers [Zeiler et al. 2010]. Transposed convolutional layers combine convolutional parameters and upsampling into a single layer. Essentially, it is a normal convolutional layer, but with the forwards and backwards pass swapped to allow upsampling of feature maps. Unlike a CPN, an FPN or a stacked hourglass detector, transposed convolution layers do not need extra mechanisms such as skip connections during upsampling for localization.

### 2.1.5. Deep Learning for 2D Pose Estimation

Before deep learning, the simplest pipelines for 2D pose estimation directly regressed keypoints from the extracted features. Such systems were computationally very fast and with little memory requirement but were impractical for commercial or in-the-wild use. This is because direct regression is highly non-linear, and thus has difficulty in learning mappings. Without extensive manual tuning, such systems [Charles et al. 2013] severely suffered in cases with similar foreground and background or in cases with self-occlusions. These systems were also unable to handle cases where multiple humans were present or if the body shapes were unseen. Deep convolution-based neural networks (CNNs), have been a huge advancement for this domain. Instead of relying on extracted features, the network itself extracts features and learns the relationship between them to get the required output. Such networks have been shown to be really robust in object localization without the need for much manual tuning and only rely on large volumes of good-quality data. Very deep networks like ResNet-101 allow for very large receptive fields (>1000 px). This allows for capturing global context without the need of introducing complex hierarchical structures.

Most deep learning CNN models begin with an encoder that progressively shrinks an image using a series of narrowing convolution blocks, or modules, to extract features. The next step is highly dependent on the approach.

#### 2.1.5.1. Regression-based methods:

Regression refers to learning a mapping from image to kinematic body joint coordinates by an end-to-end framework. This can be either done directly, or indirectly with heatmaps. In either case, the deep learning frameworks learn both feature extraction and keypoint regression in the same network.

Direct regression-based single-person 2D pose estimators [Toshev and Szegedy 2014, Pfister et al. 2014] were among the first frameworks to employ deep learning for human pose estimation. These networks require no heatmaps, thus, saving on memory cost. These networks are also easy to apply to 3D scenarios with minor changes. DeepPose [Toshev and Szegedy 2014] is a notable work in this area. DeepPose uses a multi-stage cascaded CNN structure for single-person pose estimation. Cascaded CNNs comprise multiple CNNs usually of similar architecture but different weights such that each CNN is responsible for a different aspect towards the final task. In DeepPose, a DNN called AlexNet [Krizhevsky et al. 2012] starts the cascading by generating an initial pose estimation for the first stage. This initial estimation is used to crop the image around the predicted joint location and each subsequent stage uses a separate DNN regressor to refine this prediction by only working on the cropped area. This allows the network to learn features for finer scales, thus, increasing precision. DeepPose was also the first known top-down approach for multi-person 2D pose estimation; a body detector was used to generate a rough bounding box for every person after which the single-person pose estimator was run on each bounding box to regress keypoints. DeepPose and other regression-based frameworks were found to be very error-prone and unstable because joints were learned independently of each other. In an attempt to exploit joint dependencies, [Sun et al. 2017] used a structure-aware bone-based representation in their work. The system encoded long range interactions between the bones and regressed bones instead of joints, as they are easier to detect. This made the system a lot more stable and helped it achieve impressive results. Note that the bones were defined in the local coordinate system around the root joint as edges connecting corresponding joints, and so, still required direct regression to obtain the root joint coordinates. Direct regression has been largely retired by researchers for 2D pose estimation due to the high nonlinearity acting as a bottleneck in learning mapping. Little to no knowledge about human appearance or structure is used, leading to a very vast search space. This leads to poor generalization in direct regression-based models. Direct regression also does not work in multi-person scenarios as no contextual information is captured around the keypoints, making keypoint association almost impossible. Recently, instead of using raw keypoint coordinates as regression targets, most researchers represent joint locations using heatmaps **(Section 2.1.2.2.**).

#### 2.1.5.2. Detection-based methods:

These approaches learn to predict approximate locations of body parts or joints instead of learning a specific mapping scheme, and then assemble the parts to fit a human body model. Detection targets are represented by small-region representations like heatmaps or image patches. These representations provide dense pixel information, increasing robustness. This approach differs from [Sun et al. 2017] as no direct regression is required for initial estimates. Some examples include single-person 2D pose estimators [Chen and Yuille 2014; Ramakrishna et al. 2014] and multi-person 2D pose estimators like DeepCut [Pishchulin et al. 2016] and DeeperCut [Insafutdinov et al. 2016]. The main difference

between single-person and multi-person 2D pose estimators lies in the post-processing phase; multi-person detectors must assign the detected parts to the individual human instances.

Even for an analogous approach, there are a lot of factors that govern the performance of a DNN. To showcase this, let us discuss DeepCut, a multi-person pose estimator that uses a VGGNet [Simonyan & Zisserman 2015] backbone to detect body parts, and its direct extension DeeperCut.

DeepCut experiments with two different architectures, an adapted Fast R-CNN (AFR-CNN) and a pure VGGnet based framework to train two different multi-person body part detectors. These detections are then used to jointy obtain part clusters and part labels. All part detections within a cluster are assumed to belong to a specific instance. Finally, the detections are fed to an integer linear program solver to assemble it to form coherent 2D skeletons.

- Fast R-CNN [Girshick 2015] (**Section 3.4.1.2**) was adapted for 2D human pose estimation by switching the region proposal system with a deformable part model-based (DPM) part detector. Additionally, each detection box was upscaled by a factor of 4 to capture more context around each part.

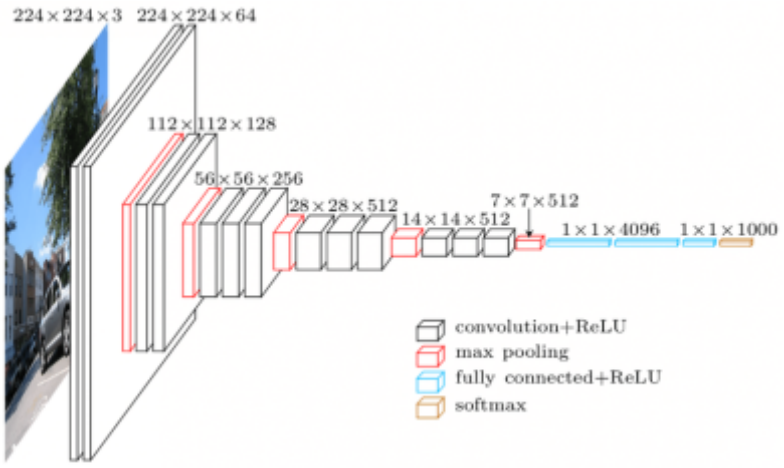



**Figure 15:**VGG-16 architecture[4]

- Dense-CNN on the other hand has a pure CNN structure (Figure 15) that is trained end-to-end. Instead of operating on proposals generated by a DPM, it employs the fully convolutional architecture to compute part probability activation maps. The advantage of this fully convolutional backbone was clear as the base model performance closely matched that of a well-tuned AFR-CNN model with a VGG-16 backbone. This was largely attributed to the fact that AFR-CNN relied on a separate part-proposal generator, leading to suboptimal results. With just the softmax loss replaced for a combination of sigmoid activation and cross entropy loss in the VGG backbone for Dense-CNN, it already outperformed the fully-tuned AFR-CNN. This is because softmax could not assign probabilities above 0.5 to several close-by body parts, thus failing to model close-by parts well. An additional fully connected layer was added in the end to handle bounding box offsets and improve localization accuracy of activation maps which further improved performance. Finally, the network was fine-tuned using more data to achieve state of the art results on the Leeds Sports Poses (LSP) dataset [Johnson & Everingham 2010].

[4] Image retrieved from: https://neurohive.io/en/popular-networks/vgg16/

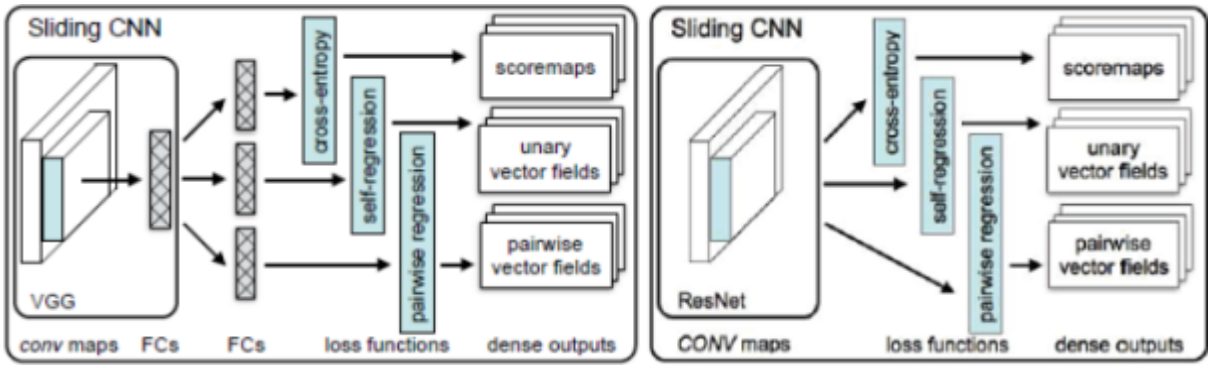



**Figure 16:** DeepCut Dense-CNN architecture[5] (left), DeeperCut architecture[8] (right)

DeeperCut aimed to make DeepCut more robust. This was done by switching the VGG-backbone with the much deeper ResNet (Figure 16). DeeperCut was experimented with Resnet-50, Resnet-101 and Resnet-152. The observations were consistent with [He et al. 2015], where as the network depth grew, the performance improved. DeeperCut used a bigger stride than DeepCut (16px vs 8px) and employed dilated convolutions (upsampling) instead of the hole algorithm [Chen et al. 2015], which led to some information loss in DeeperCut. Even so, DeeperCut easily outperformed DeepCut by using a deeper CNN. Interestingly, it was observed that this simple but much deeper network performed at par with the much more complex multi-scale multi-stage architecture of convolutional pose machines (CPM) [Wei et al. 2016].

From the above examples, it is clear that network performance improves with network depth. Comparing AFR-CNN and Dense-CNN, the architectural changes showcase well how factors like network parameters, network hyperparameters and using more data can affect the performance.

For both regression [Luvizon et al. 2017, Nibali et al. 2018] and detection based [Wei et al. 2016; Cao et al. 2019] frameworks, heatmap representations are much more ubiquitous for two major reasons. First, heatmap supervision allows a network to focus over a larger area for a keypoint and not a specific location, thus, increasing robustness. This is not possible when using raw 2D joint coordinates for supervision. Second, heatmaps provide a visual cue to the image locations that a network focuses on, and thus can greatly aid in understanding and handling failure cases. Today, any well-known 2D pose estimator that uses heatmaps is either a classic encoder-decoder network [Newell et al. 2016; He et al. 2017] (explained below) or a variant of it [Wei et al. 2016; Cao et al. 2019; Papandreou et a. 2018]. Encoder-decoder networks, as the name suggests, consist of an encoder and a decoder. The encoder transforms and maps the input into a fixed-shape hidden representation. This transformation encapsulates the variations in input in a compact representation for the network to operate on. This representation may be a feature map, or a vector, or a tensor, and is used to train higher features in a network. Finally, the decoder translates the fixed shape representations from the final encoder layer back to the intended output format. This encoding-decoding operation removes a network's dependence on a fixed-size input and allows it to learn an abstract mapping between the inputs and expected outputs without human intervention.

[5] Image retrieved from:
https://medium.com/@sh.tsang/review-deepcut-deepercut-multi-person-pose-estimation-human-pose-estimation-da5b469cbbc3

## 2.2. 3D Pose Estimation

3D pose estimation is defined as the ability to produce a 3D skeleton that matches the 3D pose of a human in an image. Analogous to 2D pose estimation, this is achieved by the 3D localization of human joints, i.e., by extracting *(x,y,z)* coordinates for each joint in an image.

Estimating 3D poses of humans from input is still an emerging field of research. The recent years have seen good progress [Mehta et al. 2018, Iskakov et al. 2019, Qiu et al. 2019] but the task of 3D pose estimation is generally considered to be harder than that for 2D pose estimation. In the following sections, we first discuss some additional challenges faced exclusively by 3D pose estimation over 2D pose estimation. We then discuss the different input types used by 3D pose estimators and how they affect the workflow of each. Finally we discuss how deep learning is employed for 3D pose estimation and exemplify the differences in some existing techniques.

### 2.2.1. Challenges

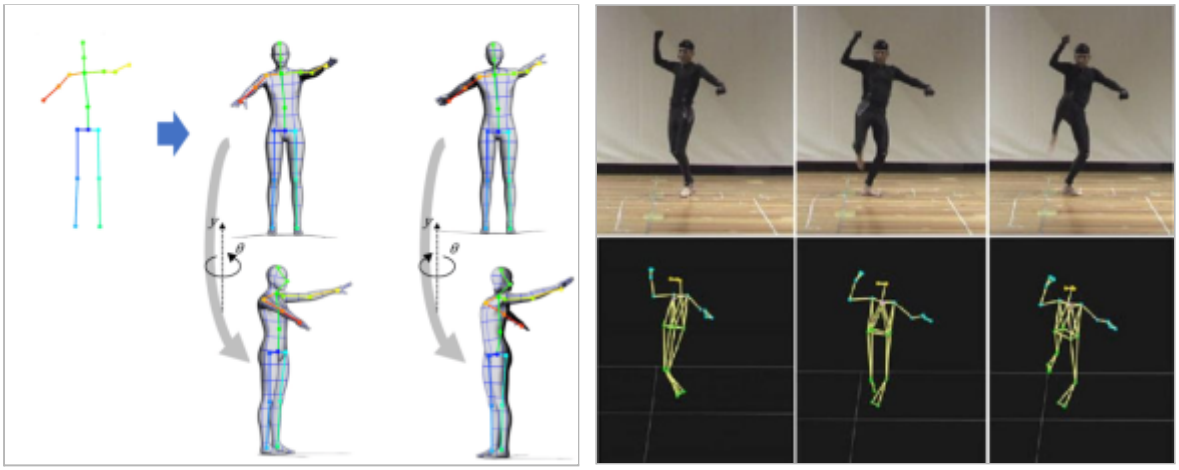


**Figure 17:** (Left) Different 3D poses map to the same 2D keypoints [Kudo et al. 2018] (right) MOCAP used to record ground-truth [Giorgis et al. 2019]

Challenges specific to 2D pose estimation also carry over to 3D pose estimation. In general, 3D pose estimation is considered harder than 2D pose estimation since it needs to predict the depth information of body joints. Depth information is inherently lost in a 2D prediction and thus can be very ambiguous to ascertain from monocular images and/or video. Most implementations learn mappings from 2D keypoints to 3D pose space using a deep learning pipeline. The pose space in 3D is also much larger than that in 2D due to the higher dimensionality. This is reflected by the simple fact that multiple 3D poses map to the same 2D keypoints (Figure 17 (left)). Some researchers however argue that lifting 2D keypoints to 3D poses is relatively straightforward as long as predicted 2D poses are accurate [Martinez et al. 2017, Pavllo et al 2019]. Another challenge is the scarcity of good training data. For 3D pose estimation, ground-truth data is chiefly generated using motion capture (MoCap) systems (Figure 17 (right)), which requires an elaborate setup. This data collection is not possible in-the-wild. 2D settings do not require such setups because it is relatively easy to manually annotate 2D joint keypoints on images and create more training data. Manually annotating 3D keypoints can be a very daunting task due to the lack of depth information, especially in monocular settings. This scarcity in ground-truth data acts as a bottleneck for in-the-wild 3D pose estimation.

Further, in-the-wild videos often encompass camera movements which can make it difficult to exploit camera parameters in certain techniques [Iskakov et al 2019, Qiu et al 2019].

### 2.2.2. Input Types

Different 3D pose estimators use different input formats. The obvious distinction is between an image and a video input. A trivial extension would be to supply a video frame-by-frame to an image-based model. This trivial extension however does not use any information from past or future frames, i.e. it does not use temporal information (**Section 2.2.3.3.**). This leads to inconsistent and jittery results across temporal sequences. Instead, most recent systems are built to exploit temporal information from image sequences to refine the results. Currently, three kinds of video inputs are supported by existing 3D pose estimators:

a. *Monocular vision:* This is the most standard form of input. Most real-world examples are monocular, i.e., have only one viewpoint of the object at all times. This is also why most works on 3D pose estimation use monocular input. Monocular videos struggle the most with occlusions. To tackle this, some researchers have recently developed occlusion-aware networks [Cheng et al. 2019] to train 3D pose estimators that explicitly deal with occlusions [Cheng et al. 2020].
b. *Stereo vision:* This is a more complicated form of input. In conventional stereo vision settings, two cameras are placed very close-by to record an instance. This is not very commonly seen in the real-world but is ubiquitous in controlled settings. The different viewing angles can be used for backprojection and triangulation of keypoints to estimate their 3D positions and draw stronger 3D inferences (Figure 18). Triangulation, however, can be very challenging if the cameras are not static over a temporal sequence. Further, in a stereo vision setup viewpoints only differ slightly, adding no new information on the instance. Another setup for stereo vision input, although much more constrained, is using an RGB-D camera, which comes equipped with a depth sensor to augment the input with depth information. This averts the need for triangulation and makes retrieving 3D poses from given depth information easier as long as 2D keypoints are accurately detected.
c. *Multi-view vision:* Another capturing technique is to use two or more calibrated cameras that are far apart to record an instance, also called a multi-view setup. This setup is frequently used to record MoCap data. This setting differs from conventional stereo vision because here the cameras offer very distinct viewpoints on the instance. This means that each camera may add new information to the system. This information can help model cases where the instance appears to be occluded or self-occluded in one view but not the others.

In general, multi-view vision can be more beneficial for the task of 3D pose estimation. It is beneficial over monocular video as triangulation can be used to infer depth information. Further, the distinct viewpoints add new information to the system unlike conventional stereo setups. If one view is highly-occluded or not ideal (side-views, partial-views, etc), the other views can be used to fill in the gaps. However, these cases are not trivial to solve, and if not tackled well can instead cause degradation of results. This generally happens when cameras contribute equally to the output but one or more cameras have incorrect information [Li. et al 2019]. An alternative is to use one view at all times while using other views for weak supervision [Rhodin et al. 2018]. Some researchers also devise their own attention mechanism to allow uneven camera contribution and filter out views with incorrect detections [Qiu et al. 2019, Iskakov et al. 2019].

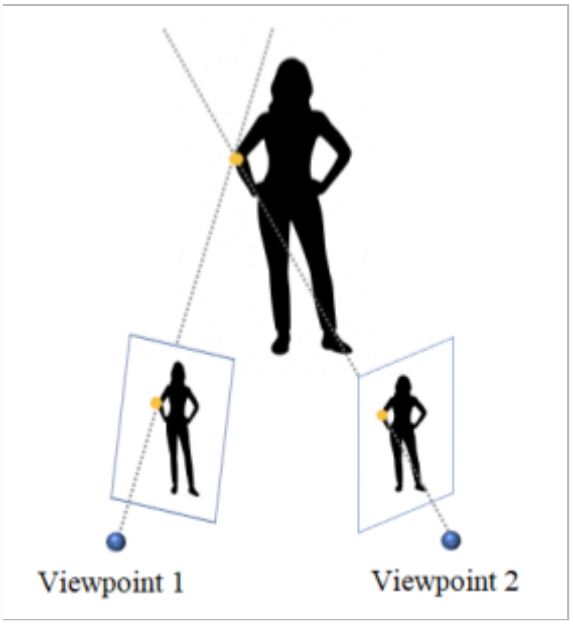



**Figure 18:** Backprojection on two views for triangulation of right elbow joint[6]

### 2.2.3. Deep Learning Pipelines

As with 2D pose estimation, deep learning has been the most widely followed approach for building 3D pose estimators. Depending on the input type, the pipeline can vary greatly in the later stages but the initial stages for almost any 3D pose estimator is based on monocular input. The main distinction in multi-view 3D pose estimators lies in refining the predictions from the monocular input using the additional views available. Additional details are discussed in the next two sections.

#### 2.2.3.1. For Monocular Input

*Lifting:* Keypoint lifting involves producing intermediary 2D pose estimations which are then mapped to 3D coordinate space [Martinez et al. 2017, Zhou et al. 2017, Li and Lee 2019]. Many existing lifting models [Martinez et al. 2017, Pavllo et al. 2019, Li and Lee 2019] use existing state-of-the-art 2D pose estimators [Wei et al 2016, Newell et al. 2016] to obtain offline 2D detections which are then lifted to 3D space. This approach however is susceptible to errors from depth ambiguity, and often requires computationally expensive iterative pose optimizers. For a better understanding on how lifting is employed in different frameworks, let us explore some existing architectures.

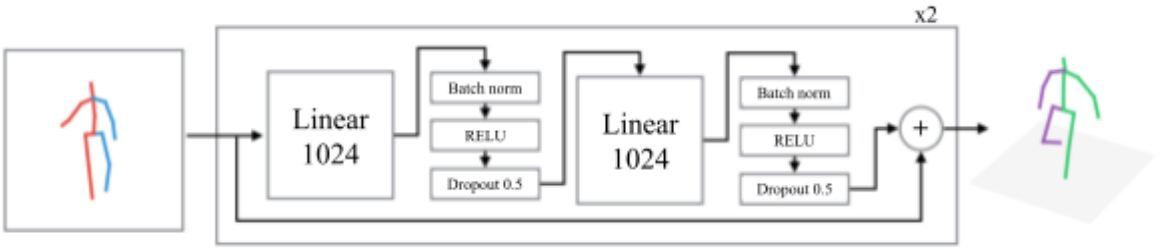



**Figure 19:** Deep feedforward network used in [Martinez et al. 2017].
Input is an array of 2D joint estimates. The entire module is iterated through twice.

[6] Image retrieved from: https://ai.googleblog.com/2019/05/moving-camera-moving-people-deep.html

For lifting coordinates, a baseline approach in the past [Pavlakos et al. 2017a] was to learn direct-mappings between 2D and 3D joint locations. This approach took image pixels as input and learnt direct mapping between intermediary 2D joint heatmaps and 3D joint locations. Direct-mapping however, suffers from an inherent problem [Martinez et al. 2017, Pavllo et al. 2019]: the global orientation and position of the root joint remains ambiguous . Thus, any translation or rotation applied to the input is not reflected in the 3D scene. To overcome this and model global position and orientation, [Martinez et al. 2017] developed an image-based single person 3D pose estimator that takes 2D joint positions as input and outputs 3D joint estimates in camera coordinate space. A deep feedforward network (Figure 19), i.e., a network with residual connections was used to learn mappings from 2D to 3D camera space while the input 2D joint estimates were procured using an off-the-shelf stacked hourglass detector [Newell et al. 2016]. For supervision, ground-truth 3D data from the Human3.6M dataset [Ionescu et al. 2013] was used after transforming them to camera coordinate space using the inverse transform of the camera projection matrices. This system saw a considerable improvement over [Pavlakos et al. 2017a] (71.9mm vs 62.9mm). It was simple, had near real-time performance, and could easily be coupled with a contemporary 2D joint estimator to improve the results.

Later, using [Martinez et al. 2017] as the foundation, VideoPose3D [Pavllo et al. 2019] (**Section 3.4.3.**) was created. This system also used 2D keypoints as input and learned to map 2D joint locations to 3D joint positions in camera coordinate space, but with a slightly different approach. Instead of estimating 3D joints directly in camera space, the authors broke down the problem into distinct parts: 1) learning 3D poses and 2) learning global positions. Two models were jointly trained, i.e., a multi-branch architecture was trained via a single loss term. The first model in this network was a pose model which uses a feedforward network to learn 3D pose estimation from temporal sequences of 2D joint coordinates, which were supplied by an off-the-shelf stacked hourglass detector [Newell et al. 2016]. The second model uses camera intrinsics to build a trajectory model of the input. Both models use the same network architecture but do not share weights. Finally, the lifted 3D pose is transformed according to the trajectory model before being outputted. It is notable that even though VideoPose3D and [Martinez et al. 2017] use very similar strategies for lifting with identical input 2D data, using temporal sequences of 2D poses coupled with the joint training regime improved performance in VideoPose3D by a huge leap (62.9mm vs 46.8mm).

Alternative to mapping, some models approach 3D pose estimation as a matching problem between 2D and 3D representations [Yasin et al. 2016, Chen and Ramanan 2017, Li and Lee 2019]. [Chen and Ramanan 2017] combined 2D pose estimation along with an over-represented library of 3D poses to lift 2D keypoints. First, they projected 3D poses arbitrarily on virtual camera views, thus creating many 2D projections for every 3D pose. This created paired (2D, 3D) data for training. Next, an off-the-shelf stacked hourglass network was used to predict the 2D pose estimation. The closest matching 2D projection from the paired data was then used to learn depths for the predicted 2D keypoints from the associated 3D pose. This work also reaffirmed that state-of-the-art 2D pose estimators had become substantially good at handling detections. Using a divergent approach, [Li and Lee 2019] also presented a keypoint matching model using hourglass modules. In this work, first a pre-trained stacked hourglass network is used to perform 2D pose estimation. Contrary to [Chen and Ramanan 2017], given 2D keypoints, this work generated multiple hypotheses of 3D pose using a mixture density network (MDN)[Bishop 1994]. These hypotheses were then reprojected to 2D and the best matching reprojection was used to learn depths for the predicted 2D keypoints using the associated 3D pose. [Li and Lee 2019] operates in both single-view and multi-view settings for a

single-person and achieved state-of-the-art for both settings on the Human3.6M benchmark dataset. MDNs were used here because instead of outputting a single value, MDNs predict an entire probability distribution for the output, which has been shown to solve occlusion cases [Ye and Kim. 2017].

*Model-free:* Model-free pipelines use no prior information about human behaviour or appearance. Most examples either use direct mapping [Pavlakos et al. 2017a, Sun et al. 2017], indirect mapping [Martinez et al. 2017, Tekin et al. 2017], or treat it as a matching problem [Chen and Ramanan 2017, Yang et al. 2018].

*Model-based:* Model-based pipelines rely on parametric body models or templates for supervision. The network is trained to predict parameters that fit the respective model, in turn tuning the network to produce results consistent with the model. These models can vary, some use known deformable part-based models like SMPL [Bogo et al. 2016, Kanazawa et al. 2018] while some use self-designed kinematic models [Mehta et al. 2017, Zhou et al. 2016]. One such example is VIBE [Kocabas et al. 2019] (Figure 21), a multi-person 3D pose estimator that works on monocular input and uses adversarial training. VIBE requires pre-processing to annotate the 2D keypoints which can be done using existing detectors. Next, a regressor is used to predict parameters for SMPL [Loper et al. 2015], a widely-used deformation model, from 2D keypoint annotations. Adversarially, a discriminator is used to differentiate real from generated parameters by leveraging AMASS [Mahmood et al. 2019], a large-scale dataset with motion capture sequences of humans, thus acting as a weak supervision for the regressor.

Model-based approaches are powerful as they enforce the network to affirm with prior knowledge about human appearance or behaviour, thus, constraining the network to produce valid 3D poses. These networks tend to be slower but are usually more robust for unseen poses than model-free approaches.

*Multi-person:* Multi-person 3D pose estimators require some additional considerations over single-person. For example, many single-person 3D pose estimators learn the pose estimates relative to the root joint. In multi-person scenes, this information is not enough; the relative spatial ordering of different people also needs to be accounted for. Similarly, human-human occlusion is more prevalent in multi-person scenes and lately some research has been employed to make pose estimators more robust to these conditions [Mehta et al. 2018, Véges and Lörincz 2020].

Multi-person pipelines for monocular 3D pose estimation, much like 2D pose estimation (**Section 2.1.3.2.**), can be classified into top-down [Rogez et al. 2017, Zanfir et al. 2018b, Moon et al. 2019, Dabral et al. 2019] or bottom-up [Zanfir et al. 2018a, Mehta et al. 2018, Mehta et al. 2020]. The key difference is that 3D pose estimators infer depth information along with the joint locations while 2D pose estimators only do the latter. Top down approaches for 3D pose estimation first crop every instance of a human out. Then, single-person pipelines are used to localize, classify and regress 3D joint positions before combining detections from all instances. Bottom-up approaches on the other hand detect 3D joint locations across all instances simultaneously before assigning them to each individual instance in the next step. For the assignment step, some frameworks [Mehta et al. 2018] use PAFs (**Section 2.1.2.2.**) while some [Zafnir et al. 2018a] treat it as a binary integer programming problem. Top-down approaches tend to be more accurate than bottom-up approaches but inference is costlier.

#### 2.2.3.2. For Multi-view input

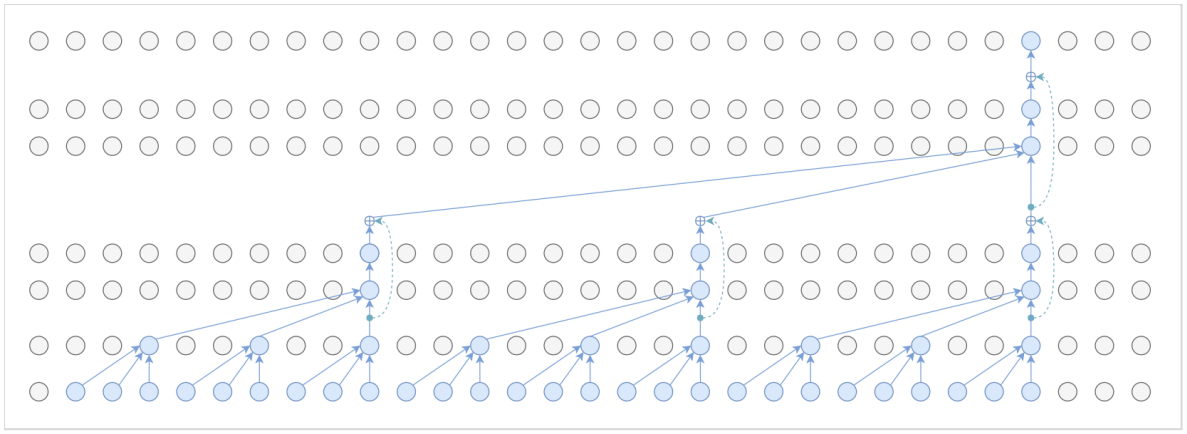
Multi-view pipelines for 3D human pose estimation are relatively new, and thus, borrowed largely from monocular pipelines. The main distinction lies in how information from additional views is combined. These additional views usually benefit the system but are a bit tricky to handle as discussed in **Section 2.2.2.** A few options have been proposed over the years including direct inferencing and weak supervision methods. [Pavlakos et al 2017b] trained a multi-view CNN with multiple monocular CNNs. These CNNs generated 2D joint heatmaps for each joint in each view. Next, the mean of the marginal distribution was calculated for each joint across views and assembled into a 3D pictorial structure model (PSM) [Felzenszwalb & Huttenlocher 2005] to create a 3D pose estimate. This estimate was then further probed with body-structure constraints like limb lengths for refinement before obtaining the final 3D keypoint positions. The network performed really well in two and three camera settings and showed clear improvement as views were increased. However, the network suffered from imprecise localizations due to the low granularity of the used PSM. Increasing the granularity was not a practical solution due to the very high computational complexity of PSMs. This was solved later, when Cross View Fusion [Qiu et al 2019] extended [Pavlakos et al 2017b] by making PSMs more granular without exploding the computational budget. This system uses a novel layer design called fusion layer to combine 2D predictions from multiple views and obtain more accurate 2D pose estimates. Next, a recursive PSM (RPSM) is used to construct the 3D pose estimates, where the recursion subdivides every active grid in the PSM, thus increasing granularity at each step. Cross View Fusion saw considerable improvement over [Pavlakos et al 2017b] (31.17mm vs 56.9mm MPJPE) and achieved state-of-the-art results on the Human3.6M dataset. For more details on Cross View fusion and RPSM please refer to **Section 3.4.4.**

An alternative technique to exploit multi-view inputs is seen in [Iskakov et al 2019]. Here, 2D features are extracted from each view in the form of heatmaps, which is in line with the above multi-view works. [Iskakov et al 2019] diverges as it uses these heatmaps to train a neural network to unproject 2D features into 3D volumes. The model, thus, learnt volumetric triangulations to exploit multi-view settings instead of using geometrical constraints. The effectiveness of this technique is clear when comparing the performance of [Iskakov et al 2019] with [Pavlakos et al 2017b] (20.8mm vs 56.9mm MPJPE). [Iskakov et al 2019] is the current state-of-the-art on the Human3.6M dataset. More details on [Iskakov et al 2019] are presented in **Section 3.4.5.**

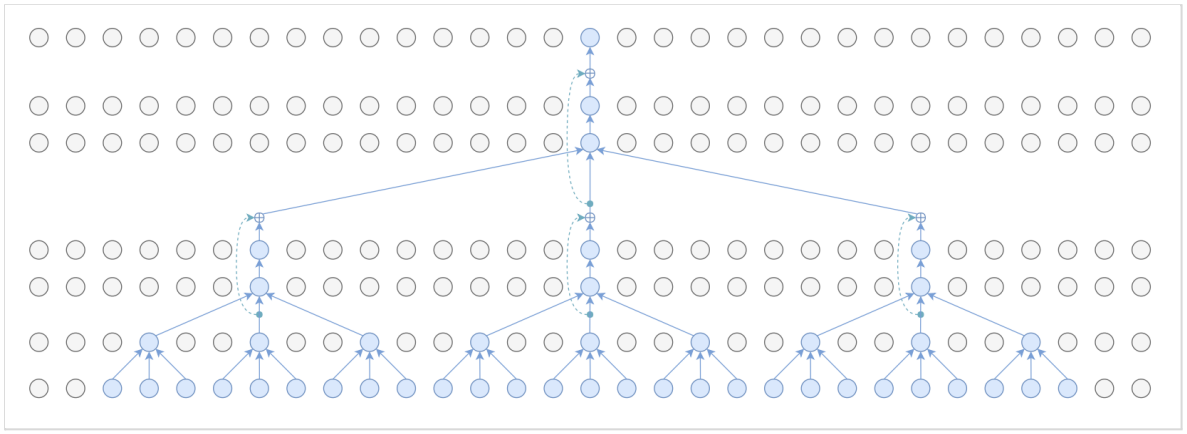
Some supplementary refining techniques also exist that can be applied to both scenarios to get better results. Let us look at these techniques in more detail.

#### 2.2.3.3. Using temporal information

Recently, encoding temporal information has been given more focus for robust 3D poses [Arnab et al. 2019; Pavllo et al. 2019; Kocabas et al. 2019]. Adjacent frames could greatly improve the results for the current frame because human motion tends to be smooth between frames. So if a keypoint is missing in a frame but is present in the past and/or future frames, it can be inferred using its location in those frames. This in turn also reduces jitter and false positives. Temporal information is captured by using temporal convolutional networks (TCN). These are a variation of CNNs that specialize in sequence modelling. A number of architectures exist for TCNs, but all rely on two rules. First, there can be no data leakage from the future into the past. Second, the network must produce an output of the same length as the input.



(a) Causal convolutions



(b) Symmetric convolutions

**Figure 20**: Dilated convolutions [Pavllo et al. 2019].
Each row represents a subsequent layer (bottom-to-top), while
the nodes represent output of each layer at subsequent time-steps (left-to-right).

To satisfy the first rule, dilated convolutions are employed as they have been shown to model long-term dependencies well in a number of other domains such as audio generation [Oord et al. 2016], semantic segmentation [Yu and Koltun 2016] and machine translation [Kalchbrenner et al. 2016]. The dilation factor results in a temporal receptive field which grows exponentially with each layer while parameter count increases only linearly. An example is the dilated causal convolutions [Oord et al. 2016] (Figure 20(a)), which only uses past information. Here, an output for frame $f_x$ is convolved with the output of frames $f_x$, $f_{x-i}$, $f_{x-2i}$, ... in the previous layer with a dilation factor $i$. A contrasting dilated convolution approach is to use symmetric convolutions [Pavllo et al. 2019] (Figure 20(b)), where an output for frame $f_x$ is convolved with the output of frames $f_{x-i}$ and $f_{x+i}$ in the previous layer, where $i$ is the dilation factor. Thus, symmetric convolution uses both past and future information. Traditionally, two dilated convolutional layers are stacked into a residual block. The output of each block is obtained by adding the inputs to the results from the final convolution in the block. To match the dimensions, 1D convolution is applied to the inputs along with padding or slicing, as required, before adding to the results from the final convolution in the block. This allows the TCN to satisfy the second rule.

2.2.3.4. Generating training data

Deep learning has a big dependence on data. So, the more data we have, the better our system will perform. Data augmentation is an obvious choice but adversarial training has also been found to be very helpful in this regard. Adversarial training was first introduced in the generative adversarial networks (GAN) [Goodfellow et al. 2014] (Figure 21) where a transposed convolutional network, or the generator, is made to compete against a convolutional network, or the discriminator, to produce synthetic data that can be passed as real data. Initially, the discriminator is trained on a dataset to learn the true data distribution. After this, the generator is tasked with creating data that is good enough to 'fool' the discriminator into thinking it belongs to the true data distribution. This, in turn, further tunes the discriminator. A GAN can thus help to increase the amount of training data with the aim of improving the classification/detection performance.

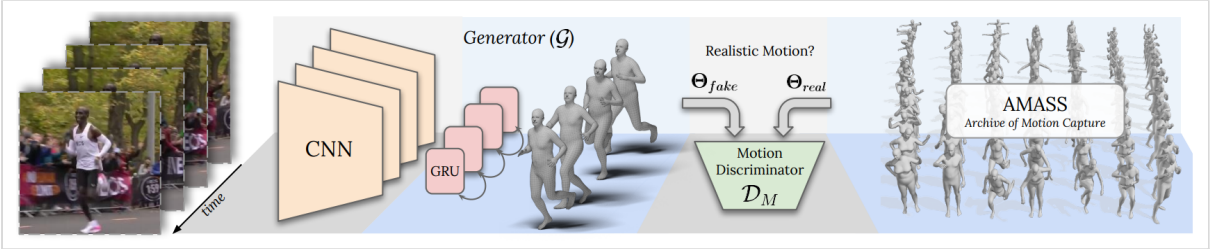



**Figure 21:** GAN used in VIBE [Kocabas et al. 2019].
The generator estimates SMPL parameters. The discriminator differentiates it from AMASS.

The novel idea of two networks adversarially learning from each other has been adopted by a lot of researchers for 3D human pose estimation models [Yang et al. 2018, Kanazawa et al. 2018, Kocabas et al. 2019]. This practice was enabled due to the availability of large, high-quality MoCap datasets in recent years to provide ground-truth data for training the discriminator.

Adversarial generation of data has also been used to produce synthetic datasets [Varol et al. 2017, Ludl et al. 2018]. Obtaining high-quality MoCap data is a very expensive process. The colossal corpus of poses that humans can produce coupled with underlying biological factors such as age, gender and race is thus never fully represented in any available MoCap dataset. Most real-world datasets only focus on basic human-related tasks. Synthetic datasets aim to cover more exotic cases and help extend real-world data in a cheap and flexible fashion. Although synthesizing accurate data is much harder, it does not require an expensive setup or manual labelling. Realistic synthetic data for human pose estimation generated from virtual worlds has also seen success lately [Shafaei and Little 2016, Souza et al. 2017] due to the availability of modern modelling, rendering and simulation softwares.

## 2.3. Action Recognition:

Post 3D pose estimation, the action being performed by a human can be recognized in a number of ways. Earlier, given a temporal sequence of 3D poses for the human, most techniques relied on trained classifiers to identify the class of actions. Traditionally, this is done using the random forest algorithm (RF) [Ho 1995] or a trained Support Vector Machine (SVM) [Cortes and Vapnik 1995] or an artificial neural network (ANN) [Krizhevsky et al. 2012]. Both random forest and SVMs are techniques for machine learning whereas an ANN adopts deep learning. The use-case for each technique is highly dependent on the nature of the task as well the type and volume of data being used.

RF uses an ensemble of decision trees where each tree starts with random samples of the training data. Each tree then splits using a subset of the sampled data and is terminated when it reaches a vote for the class, i.e., the leaf node. The leaf nodes of each tree are then aggregated and the majority-voted class is outputted as the final prediction. RF is non-linear and computationally less expensive than SVM and ANN. RF also allows for easy interpretation of the features at every level and how each variable contributes towards the final output. However, the random nature impairs the user from having much control over the model.

SVM discriminates one class from another using vectors which virtually draw margins of separation, i.e. hyperplanes, between classes. It works well when classes are easily distinguishable or have a high dimensionality. SVM adopts a one-versus-all approach to produce a single binary output, i.e., a separate SVM is required for each class label. As such, SVMs are immune to overfitting and can often outperform ANNs [Sakr et al. 2016] as they are less prone to be stuck in a local minima. This, however, also means that SVM can be really intensive to train when datasets are large or noisy. In the case of human actions, humans can produce a limitless class of actions. Training a separate SVM for each action, therefore, is not at all feasible.

When datasets are large, for both RF and SVM the complexity grows with the number of training samples. ANN models on the other hand are fixed in terms of input nodes, hidden layers and output nodes while model performance only improves as data increases. ANN handles multi-class problems by producing probabilities for each class whereas both RF and SVM traditionally output a single class label. Though training an ANN can be expensive, their run-time computational performance far exceeds an SVM or random forest.

Another way to recognize actions which diverges from all the mentioned methods is by using vector arithmetic. We exemplify this method in detail in **Section 3.6.4.** The method has its merits in that it requires no pre-training and can be easily adapted to recognize any class of actions as long as they can be differentiated mathematically. A simple example is to detect the angle between the knee-hip segment and the hip-shoulder segment to determine if a subject is seated or standing. A seated instance would have the angle closer to 90° while a standing subject would have the angle closer to 180°. This method requires thresholding to differentiate actions, and can thus, often miss very subtle movements.

Some human actions remain dubious to detect no matter the implementation. One such example is when a subject is pointing up vs displaying the number one using their fingers. Both actions look exactly the same but cannot be differentiated without using contextual cues. Contextual cues are hard to gather from video streams.

# 3. Methodology

This chapter summarizes our approach, and motivates our design with the literature that was discussed in the previous section. It highlights the relevant techniques that we used directly for our experiments. The following subsection motivates our work. Next, we go over the datasets (**Section 3.2.**) that we use, followed by the different metrics we will be using for our evaluations (**Section 3.3.**). different pre-existing modules (**Section 3.4.**). Then, we compare the pre-existing modules to benchmark their performances (**Section 3.5.**). Finally, we introduce our work **(Section 3.6.**).

## 3.1. Our Plan

Our first goal is to analyse how 3DHPEs perform in general. We do this in two steps (Figure 22). For the first step, we select two monocular 3DHPE techniques that supposedly work well in the wild. One of the techniques, VideoPose3D (**Section 3.4.3.**) takes temporal sequences of single-person 2D pose estimates as input and lifts them to 3D coordinate space using a CNN. The other technique, VIBE (**Section 3.4.6.**) uses raw image sequences to regress posed 3D human shapes to every instance present in the frames. These human shapes are pre-learned by the network using adversarial training. We perform a qualitative analysis on both these techniques to discern their applicability in real-world scenarios. This is done using a private, non-annotated dataset called YOUth (**Section 3.2.2.**). For the second step, we attempt to explore how monocular and multi-view techniques compete against each other. To this end, we select the model that displays superior applicability from step one, VideoPose3D, and compare it numerically against two state-of-the-art multi-view single-person 3DHPE techniques, CrossView (**Section 3.4.4.**) and LearnTri (**Section 3.4.5.**). Both of the used multi-view techniques are not readily applicable to in-the-wild scenarios due to their heavy dependence on camera parameters. On the other hand, the monocular techniques from step one only require either 2D pose estimates or raw image sequences. Therefore, this step also provides a comparison between techniques that do not require special annotations vs. techniques that do. A well-known, multi-view, single-person dataset called Human3.6M (**Section 3.2.1.**) is used to perform the numerical comparison.

With a good baseline for in-the-wild 3DHPE techniques, we now aim to answer our other two research questions. We again use the YOUth dataset to inspect the current quality of 3DHPE techniques for parent-child interaction videos. Normally, for multi-person cases using a multi-person 3DHPE technique would be the obvious choice. However, to study interaction it is imperative to identify which pose instance belongs to which human instance. One such approach is to use tracking [Bridgeman et al. 2019, Chen et al. 2020]. Such frameworks however require calibrated cameras, which may not always be available in real-world scenarios. An alternative approach is to split the instances into individual streams and use single-person 3DHPE. The pose information from each stream can then be assembled later to study interactions more robustly. We follow this approach and separate instances from the video streams using an off-the-shelf person detector (**Section 3.4.1.**) followed by a self-designed color-matching module (**Sections 3.6.1.-2.**). The person detector isolates the instances in each frame while color-matching identifies the label of the instances across frames and views. With the multi-person streams converted to single-person, we obtain 2D pose annotations using OpenPose (**Section 3.4.2.**) and feed it to VideoPose3D to finally obtain 3D pose estimates. For action detection, we explored a number of ways in **Section 2.3.** Temporal sequences of 3D poses ideally capture enough information of the actions being performed by a subject. We choose rule-based classification as the ideal method for our detection task. This is because rule-based approaches, unlike

machine learning or deep learning techniques, are incapable of inherently handling complex cases, thus providing a higher insight on the quality of the 3D data. This allows us to identify the challenges pertaining to action classification using 3D pose data more accurately, thus, permitting us to answer our third and final research question adeptly. We explore detection of atomic actions for only the infants as the observations provide ample insight on the challenges for the more complex interaction detection task.

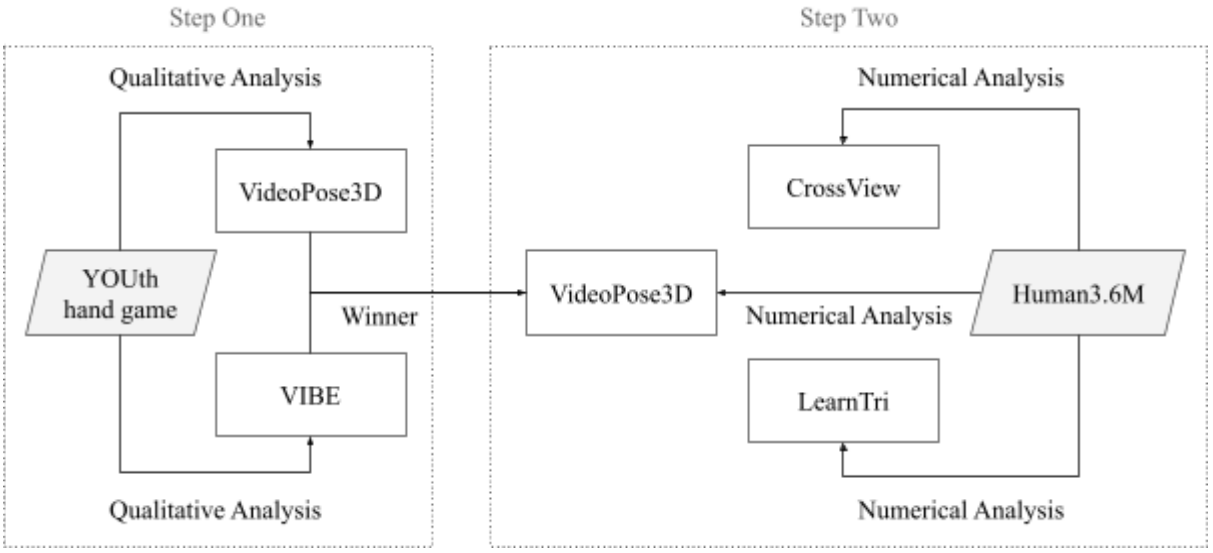



**Figure 22:** Schematic diagram of our evaluation pipeline

## 3.2. Datasets

We use a number of datasets in our research. Each dataset is used for a distinct purpose as explained below.

### 3.2.1. Human3.6M:

Human3.6M is a large-scale dataset containing 3.6 million different 3D articulated poses. These poses are captured by recording the performance of 11 subjects, 5 female and 6 male, from 4 camera view points. Each subject performs 15 kinds of actions which encompass a subset of typical human activities like walking, eating, greeting, with two sessions per action. The recording is done using 10 MoCap cameras and 1 time-of-flight sensor coupled with 4 digital video cameras in a capture space of approximately 4mx3m. The dataset provides synchronized 2D and 3D joint data to abundantly train pose estimation systems. The dataset also has a suggested split for training, validation and testing; S1, S5, S6, S7 and S8 are used for training, S9 and S11 for validation and S2, S3, S4 and S10 are reserved for testing. This allows a satisfactory variability of body shape as well as range of motions. This dataset is used for benchmarking our chosen 3DHPE techniques.

### 3.2.2. YOUth:

YOUth [Verhagen et al. 2017] is a non-annotated large private dataset of infant-caregiver interaction videos. The dataset contains a number of tasks. One such task is the hand game (Figure 23), where an infant and caregiver are seated on the ground in a play area and given a box of toys. This task is divided into two halves. The first half contains about 126 videos of infant-caregiver play sessions where the mean age of the infants is 10 months. The second half contains another 126 videos but with

the same infants 8 months later (mean age 18 months). Each video is about 15-minutes long and is recorded in a multi-view setup with 4 cameras at 25 FPS each (so each frame is 40 milliseconds apart). Frequently, infants tend to veer, placing them out-of-view or too far in the background in one or more views. Due to this, the cameras use panning and zooming frequently in the videos to contain both the infant and caregiver in the field of vision. This dataset is only used for preliminary evaluations (**Section 3.5.1.**) on OpenPose, VideoPose3D and VIBE.

Another task in the YOUth dataset is called Gift-Delay (Figure 24). Gift Delay [Verhagen et al. 2017] is a small set of videos from YOUth developed for a study on preschool children, i.e. two- and three-year-olds. In this task, a gift is concealed in a bag which is placed in front of the child on the table. The child is promised the gift in the bag by the experimenter but is tasked to refrain from touching the bag until the experimenter returns from a short errand. A guardian accompanies the child by sitting behind them so that they are out of the child's line-of-sight. The guardian is given strict instructions to not interact with the child to keep interference to a minimum. The dataset has 66 videos, each about 5 mins long with the task time spanning from 3 to 4 minutes. The recording setup is identical to the hand game task. We filter out sessions where the child is either mostly out-of-view in all cameras or undergoes any parental influence. We also ignore sessions where the child does not understand the task, for example taking the gift out of the bag immediately after the experimenter leaves. We only work with the remaining 35 videos. We annotate these videos following a self-generated coding scheme (**Section 3.6.4.1.**) with the help of BORIS [Friard and Gamba 2016]. These annotations are later used as ground truth to numerically evaluate our action detection task (**Section 3.6.4.2.**). Note that all displayed images from the YOUth dataset are blurred and the faces are omitted to preserve privacy.

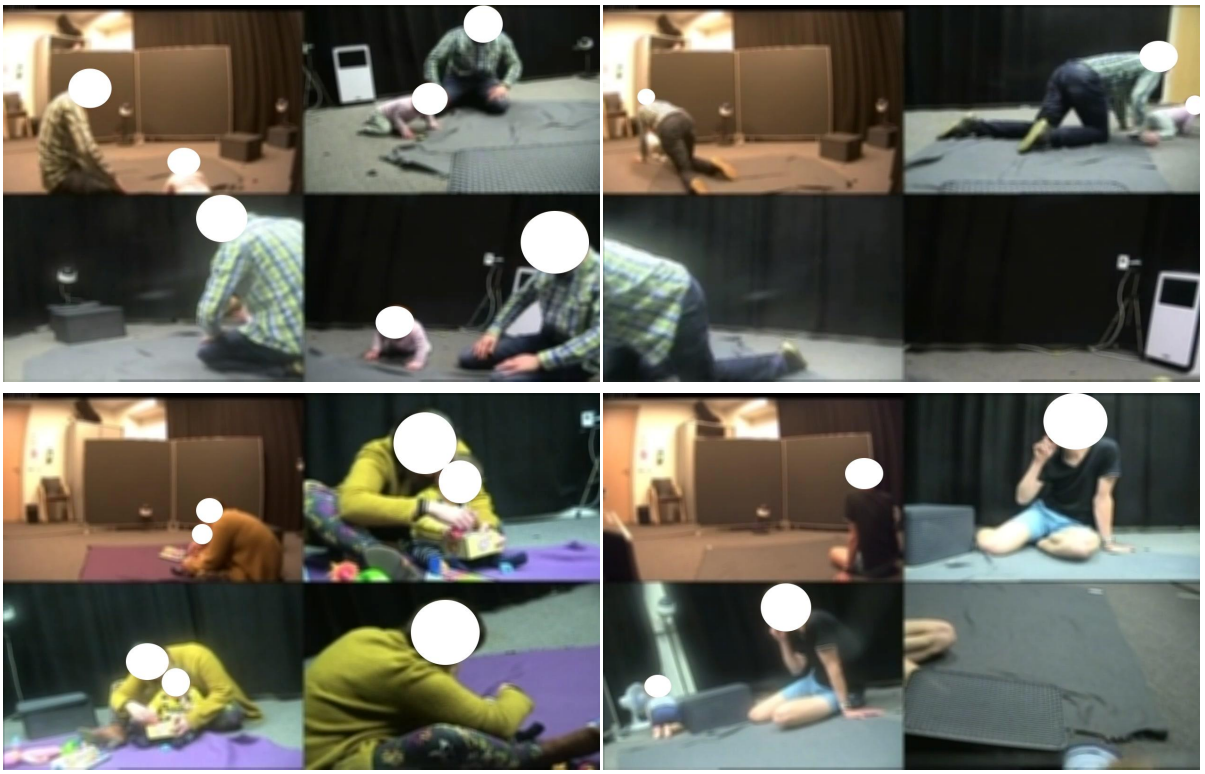


**Figure 23**: YOUth hand game dataset examples (blurred)

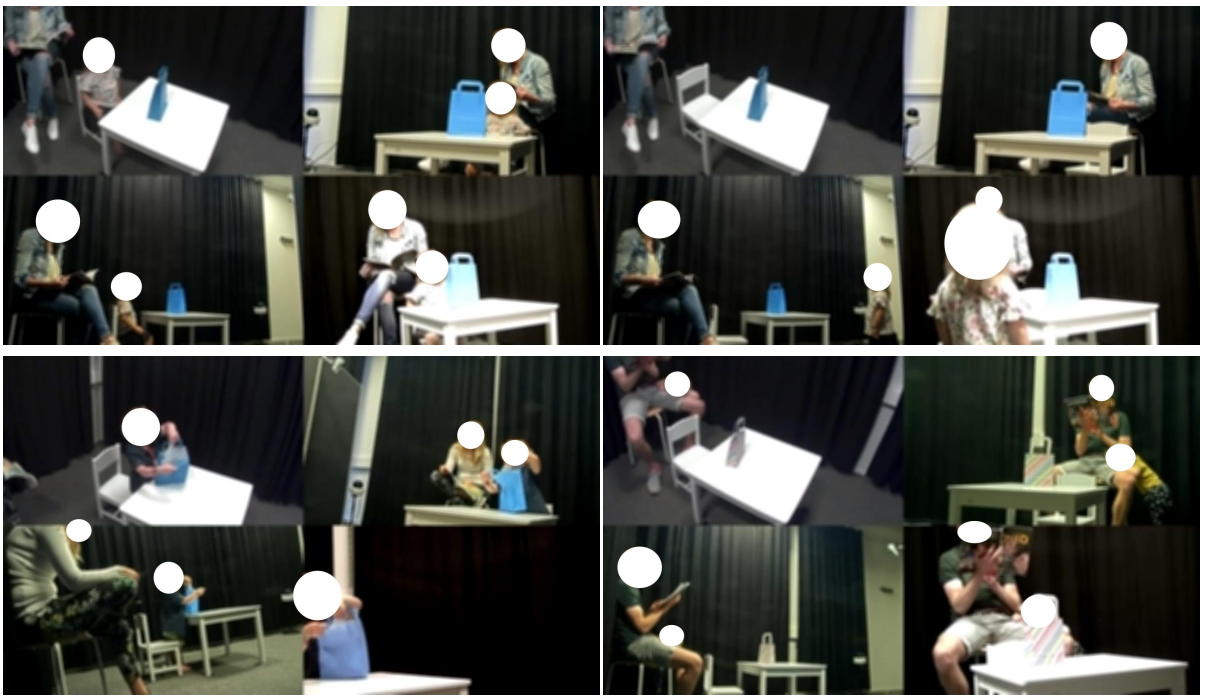


**Figure 24:** Gift-Delay dataset examples (blurred)

## 3.3. Metrics

We use two metrics for all our pose estimation experiments:

- Mean per joint position error (MPJPE): MPJPE [Ionescu et al. 2013] calculates the euclidean distance from estimated 3D or 2D joints to the ground truth. The final value is averaged over the number of joints and reported in millimeters (mm) in 3D settings and in pixels (px) in 2D settings. Lower MPJPE is better.
- Average Precision (AP): Also referred to as Average Precision of Keypoints (APK) [Yang and Ramanan 2013], this measures the tendency of a model's detections to be correct. It is a fraction of correct keypoints in all detected keypoints for each joint. In classification tasks, it is a fraction of correctly classified objects in all objects classified for each label. Mean average precision (mAP) is calculated by averaging the AP over all joints or all class labels.

For the action detection task, we compare the classified actions with the annotations and report the precision and recall.

## 3.4. Modules

A number of off-the-shelf detectors are used in our approach. We will explore each in great detail in this subsection.

### 3.4.1. Instance Segmentation

Mask R-CNN [He et al. 2017] is a state-of-the-art instance segmentation network used for object detection and classification. To fully understand Mask R-CNN's design, a fundamental understanding of its predecessors is very beneficial. The earliest iteration is the R-CNN framework [Girshick et al. 2013] that uses a region proposal system and a CNN structure for object detection and classification.

#### 3.4.1.1. R-CNN

R-CNN was specifically designed to lower inference time for object localization by mitigating the brute-force sliding window approach with selective search [Uijlings et al. 2012]. From an input

image, region proposals are generated using selective search. About 2000 candidate proposals are fed to AlexNet [Krizhevsky et al. 2012] for feature extraction. This AlexNet is pre-trained on image classification using lots of data after which the last layer is removed, giving an output feature vector of size 4096. Extracted features are then fed to class-specific linear SVMs to classify the object. Finally the bounding boxes for the detections are regressed externally and outputted along with the class label. R-CNN solved the issue of object localization but suffered from a number of problems. It required independent training for every part; a fully tuned AlexNet was required to train the SVMs. Additionally, due to the many proposals, inference on R-CNN was very slow as it required a separate pass for each proposal and thus also had a large memory requirement. To solve these problems, Fast R-CNN [Girshick 2015] was developed.

#### 3.4.1.2. Fast R-CNN

This framework was designed specifically to allow end-to-end training and thus combined the CNN, SVMs and Bounding Box regressor from R-CNN into one structure. Instead of extracting features on each proposed region, features are extracted on the entire image using a ConvNet, VGG-16 in the paper. From the output feature map, corresponding parts are cropped using input region proposals, and then warped to fixed map sizes using pooling, referred to as Region of Interest (RoI) pooling layer. This allows the resultant feature vector to have a static size which is fed to fully connected layers for object classification and bounding box regression. Fast R-CNN could be trained 9 times faster than R-CNN and had a marginally higher mAP (66.6 vs 66.0) while maintaining a 213 times shorter inference time. Although Fast R-CNN saw immense improvements in speed over R-CNN, generating input region proposals became the bottleneck for real-time performance. Further, since the used region proposal generator was a static algorithm, no learning happens in that stage. Thus, often bad candidate proposals are generated. To alleviate this bottleneck, Faster R-CNN [Ren et al. 2015] was developed.

#### 3.4.1.3. Faster R-CNN

Faster R-CNN builds over Fast R-CNN by generating region proposals with a novel Region Proposal Network (RPN) instead of a static algorithm. This RPN allows the model to learn region proposals by extending the ConvNet from Fast R-CNN with additional convolutional layers to simultaneously regress region bounds and the objectness score at each location on a regular grid. Sharing layers with the ConvNet allows the RPN to be nearly cost-free. Scaling and aspect ratio of the proposals is addressed by the novel anchor boxes (**Section 2.1.2.1.**). These proposals are then fed to the RoI pooling layer analogous to Fast R-CNN to extract feature vectors for object classification. The resulting Faster R-CNN model saw a 10 times speedup from Fast R-CNN and achieved near real-time performance (~5FPS) while having a consistently higher mAP even though the used number of proposals was ~6.67 times lower than Fast R-CNN (300 vs 2000). Faster R-CNN achieved state-of-the-art, but it required a complicated, alternate training regime, and could only output a regressed bounding box of the detection, i.e., it cannot perform pixel level segmentation. To allow for this, Faster R-CNN was extended to finally yield Mask R-CNN (Figure 25).

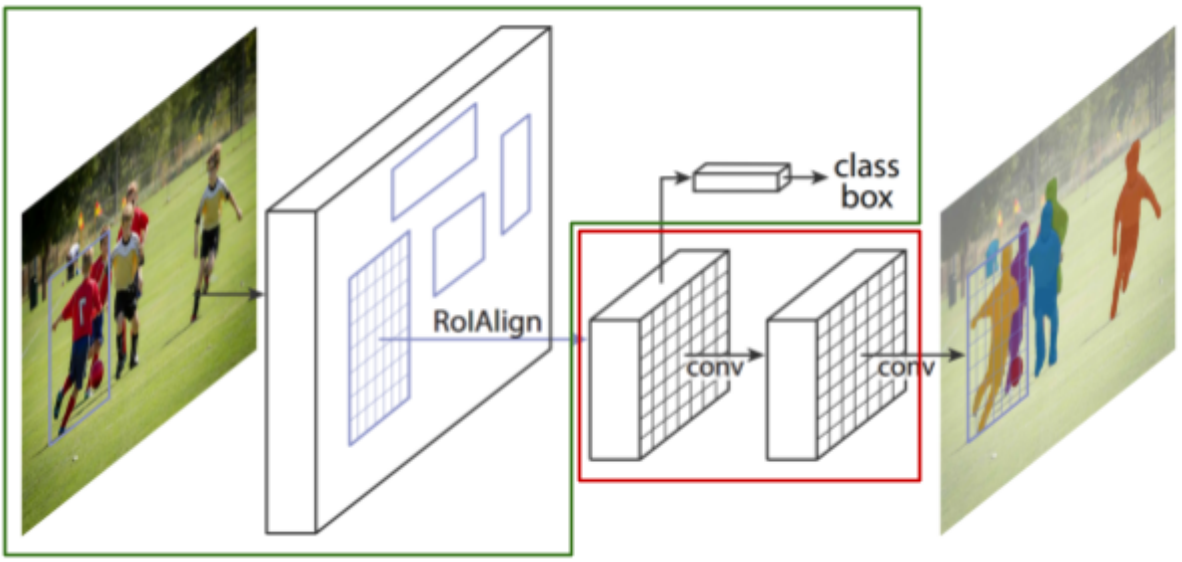



**Figure 25:** Mask R-CNN architecture [He et al. 2017].
□ = Faster R-CNN, □ = Instance segmentation

#### 3.4.1.4. Mask R-CNN

Given an input image, Mask R-CNN operates in two stages. In the first stage, region proposals are generated using a ConvNet and a RPN system, analogous to Faster R-CNN. In the second stage, these proposals are regressed and classified. Parallely (Figure 25), the second stage also generates a pixel-level mask of the detected object shape with a Fully Convolutional Network (FCN) [Long et al. 2014], which extracts finer spatial layouts for each RoI and outputs a binary mask indicating the pixels where the object is present within the bounding box. The used RoIs here are slightly different from its predecessors; instead of pooling pixel values to obtain a fixed map size, bilinear interpolation is used to estimate subpixel values. This change helps in mitigating misalignments and achieving a more accurate pixel-level segmentation. The produced binary masks are combined with the bounding boxes and class labels to generate precise segmentations.

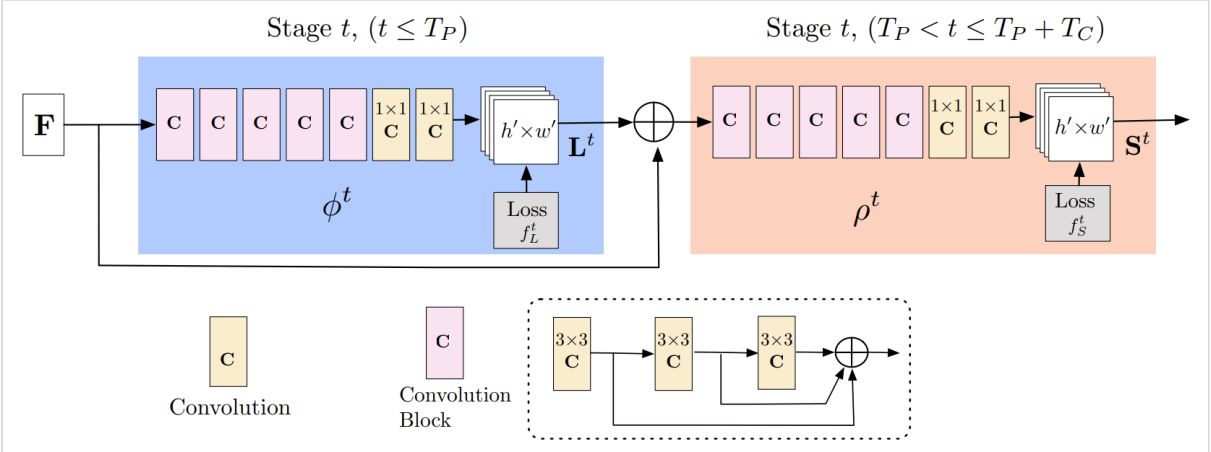
Mask R-CNN achieved state-of-the-art with a ResNet-101 Feature Pyramid Network (FPN) [Lin et al. 2016] backbone for the first stage, outperforming all previous state-of-the-arts. Additionally, it can generate multiple labels for each class unlike its predecessors, i.e., it can perform instance segmentation.

### 3.4.2. OpenPose

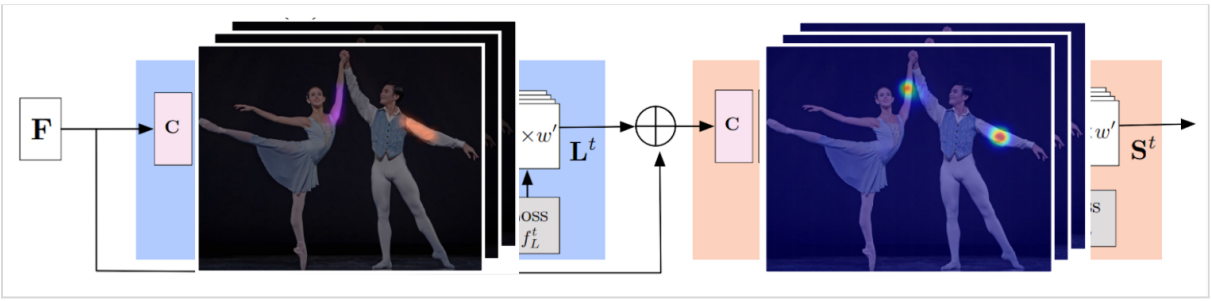
OpenPose [Cao et al. 2019] is a bottom-up model for multi-person 2D pose estimation that jointly detects human body, foot, hand and facial keypoints for a total of 135 keypoints on single images. It is the first system that works well in-the-wild while maintaining near real-time performance.

OpenPose network operates in four steps, and thus leverages multi-stage training (Figure 26(a)). The first step extracts feature maps from an image using the first 10 layers of VGG-19 [Simonyan & Zisserman 2015]. In the second step, the feature maps are used to produce PAFs (**Section 2.1.2.2.**) using DenseNet-like blocks. These PAFs are refined over a set of stages before step three by concatenating the predicted PAFs and the corresponding feature maps at every stage and using it as an input for the next stage. The third step uses the feature maps from step one and the refined PAFs from step two to predict confidence maps i.e., heatmaps (**Section 2.1.2.2.**) with the joint locations, using DenseNet-like modules. Similar to PAFs each confidence map denotes all detections of a particular

joint in the image. Much like step two, the confidence maps are refined over a set of stages before step four by concatenating the predictions with the corresponding input features for each subsequent stage. In the fourth and final step, the generated confidence maps and part affinity fields are processed to obtain the poses for each person in the image. An ideal solution to draw associations would be a K-dimensional graph matching problem. This is, however, known to be NP hard [West et al. 2001] with an exceptionally high run-time cost of few minutes [Insafutdinov et al. 2016] to few hours [Pishchulin et al. 2016] per image. Instead, [Cao et al. 2017] use a modified bipartite matching algorithm which is orders of magnitudes faster as it only operates on local context. This works sufficiently well as PAFs use a large receptive field, thus capturing enough global information to allow the greedy parse, consequently, making the system near real-time.




(a) Multi-stage Architecture




(b) Feature Maps used: PAFs (left) and joint heatmaps (right)

**Figure 26:** OpenPose architecture [Cao et al. 2019]

The bottom-up approach of OpenPose along with the greedy pass allows it to scale well with the number of people present, making OpenPose very efficient in multi-person 2D pose estimation. The technique also generalizes well to real-world cases.

### 3.4.3. VideoPose3D

VideoPose3D [Pavllo et al. 2019] is a simple and efficient approach for 3D human pose estimation that uses dilated temporal convolutions (**Section 2.2.3.3.**) on 2D keypoint trajectories. It also introduces a semi-supervised training method to improve in-the-wild inference. As the theoretical design was discussed earlier (**Section 2.2.3.1.**) we will focus on the inner workings in this section. For the remainder of the text, we will refer to this module as VideoPose.

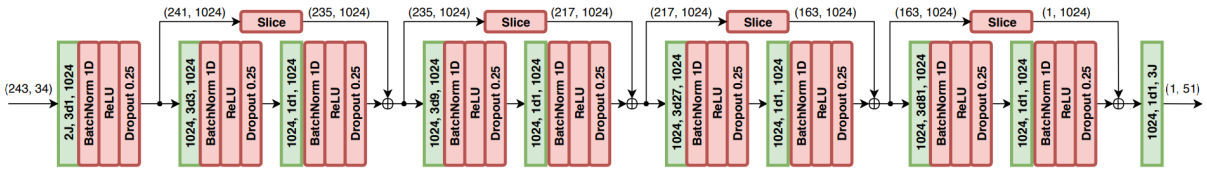



**Figure 27:** VideoPose3D network architecture [Pavllo et al. 2019]

VideoPose uses two identical models that do not share weights. The architecture of the networks (Figure 27) is fully convolutional with residual connections and employs dilated convolutions to model long-term dependencies while maintaining efficiency. The input layer takes $F$ frames of 2D joint data for $J$ joints concatenated together in the form ($J_x$ , $J_y$) to which a temporal convolution is applied with kernel size $W$ and $C$ output channels. $F$ here corresponds to the temporal receptive field size. The resulting feature vector is then dilated over $B$ ResNet-style blocks which are surrounded by skip-connections, with each block performing a 1D convolution and dilation by a factor of $D = W^B$ , i.e., $D$ = [3, 9, 27, 81] for the first, second, third and fourth block respectively if $W$ = 3. This is followed by a convolution with a kernel of size 1. The dilation factor increases the network's receptive field exponentially with every block while the number of parameters increases only linearly. To match the dimensions of the residuals with the subsequent tensors, the residuals are sliced symmetrically from the left and right. The last ResNet block is followed by a final convolution layer with kernel size 1. All convolutional layers except the last are followed by batch normalization, ReLU, and dropout, to increase network non-linearity.

The first network, called the pose model, learns a deterministic mapping between 2D keypoints and 3D keypoints while the second network, called the trajectory model, regresses the global trajectory, i.e., the global position of the root joint and scale of the person in camera space using intrinsics. The pose model is driven by an MPJPE loss while the trajectory model uses a weighted MPJPE (WMPJPE) loss term, where each sample is weighted according to the inverse of the ground-truth depth in camera space. This weight is assigned because regressing precise trajectories of subjects becomes increasingly difficult, the further away they are from the camera.

This supervised setting helped the model achieve state-of-the-art results but to make the model more robust, a mix of labelled and unlabelled data is used in a semi-supervised setting and joint training (Figure 28) is employed. For labelled data, the setting is supervised as described above. For unlabelled data, the predicted 3D pose is first constrained by a bone length loss term to match the mean bone length of the subjects in the unlabelled data to the subjects in the labelled data. Finally, the 3D prediction is coupled with the global position and the scale from the trajectory model and reprojected to 2D. This re-projection is used to compute a 2D MPJPE loss term to drive training.

The VideoPose module lifts keypoints from 2D to 3D and does not rely on any particular 2D annotation technique. It only requires a bounding box of a single person and the corresponding 2D keypoint annotations as input. The drawback is that VideoPose cannot perform multi-person 3D pose estimation. It is also designed for monocular views, and thus, cannot reap any benefits from a multi-view setting

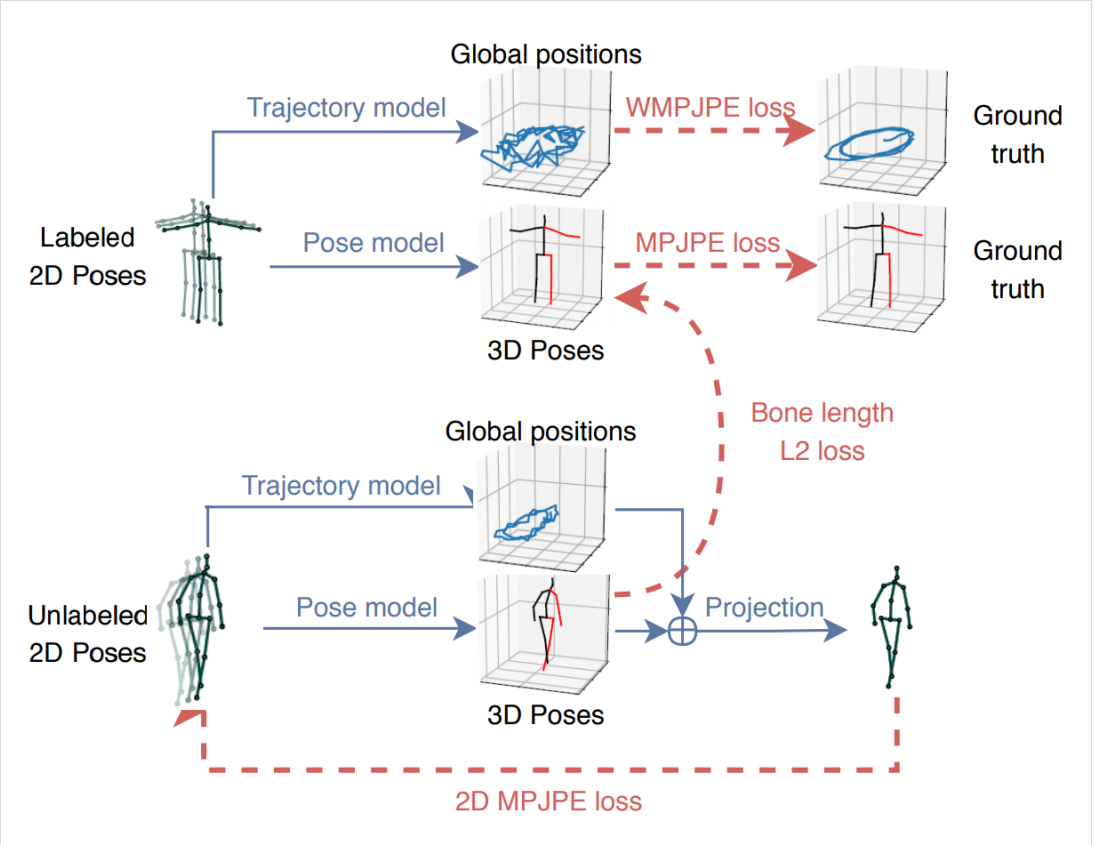



**Figure 28:** VideoPose3D joint training [Pavllo et al. 2019]

### 3.4.4. Cross View Fusion

Unlike VideoPose, cross view fusion [Qiu et al 2019] learns 2D annotations by leveraging data from multiple views before assembling them to 3D poses using a recursive Pictorial Structure Model. For the remainder of the text, we will refer to this module as CrossView. Crossview is the current state-of-the-art on Human3.6M after LearnTri (**Section 3.4.5.**). It consists of two distinct models. The first network, called Cross View Fusion, detects 2D keypoint annotations from RGB images for each camera view and combines them using a novel fusion layer design. For simplicity, we will refer to this network as the fusion model (Figure 29). The second model is a novel recursive pictorial structures model (RPSM) that lifts these 2D annotations to 3D space. Let us look at these models in more detail.

#### 3.4.4.1. Fusion Model

The 2D keypoint detection, akin to SOTA 2D detectors, uses a single CNN structure and is trained in an end-to-end fashion. The CNN structure used is adopted from [Xiao et al 2018]; a ResNet-152 backbone is used with three transposed convolutional layers following the last convolution stage in the ResNet. Each transposed convolutional layer (**Section 2.1.4.1.**) uses 256 filters with kernel size 4 and stride 2, and is followed by batch normalization and ReLU activation. The last layer of the network performs convolutions with kernel size 1 to generate the heatmaps. The resultant network allows an input image size of 320x320 while the resolution of the output heatmap is 80x80. Given two views of an image, each view is separately fed into the CNN to get initial heatmaps. This gives us keypoint data from both views, i.e., two heatmaps per joint. To locate the joints more precisely, it makes sense to combine the two heatmaps in some way. The authors do this with the help of a fusion layer.

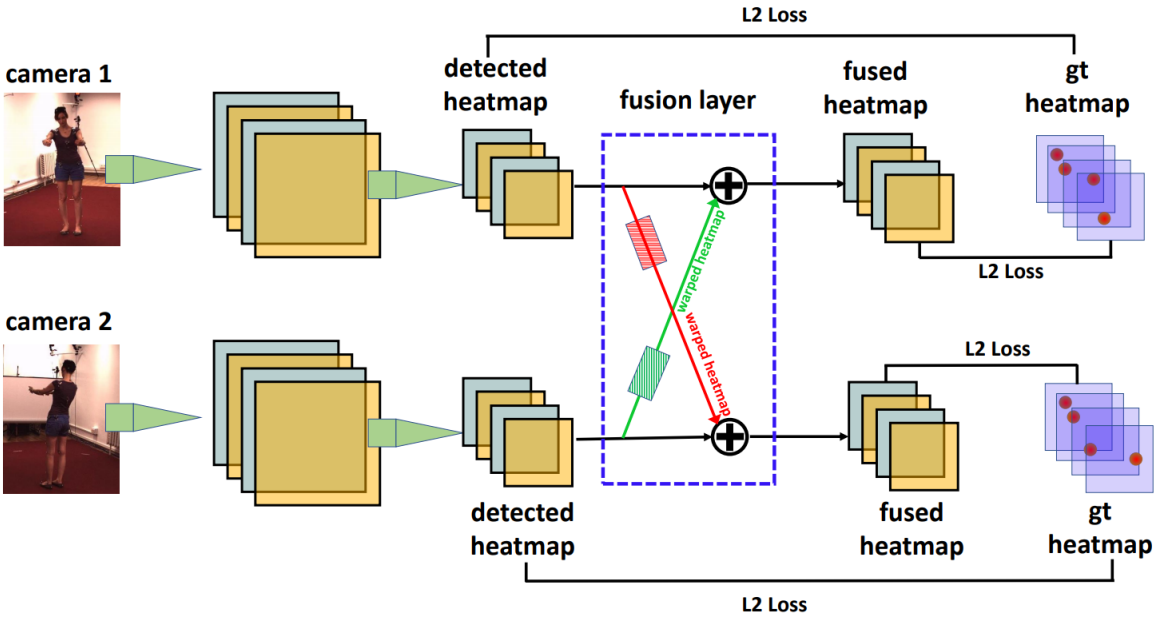



**Figure 29:** Fusion Model used in [Qiu et al. 2019]

*Fusion layer:*

Once initial heatmaps from each view have been obtained, they are passed through a fully connected layer. For a current view *V*, this layer maps each pixel in *V* to all pixels in the other views such that the weights are mostly positive for locations along its epipolar line. Epipolar line is the image in one camera of a ray through an image point in the other camera. This is computed using backprojection (Figure 18) and thus requires the intrinsics for both cameras. Using this layer, warped heatmaps are first generated for each view by weighted addition of the heatmaps from the other views. Warped heatmaps, thus, help in tackling situations where a keypoint is visible correctly in the other views but not in the current view. Finally, the warped heatmaps from other views are fused with the initial heatmap of *V* to obtain the fused heatmap for *V*; each location in *V* is fused with all features on its epipolar line. This fusion generously improves the heatmap quality. Both the initial heatmaps and fused heatmaps are treated as regression targets and compared to ground truth heatmaps to enforce an L2 loss across all views and drive training. A huge drawback of the approach is its direct reliance on camera configurations for epipolar geometry, making the model inept for a different camera configuration. To overcome this, a semi-supervised training approach is used following [Simon et al. 2017]. First, a single view 2D pose estimator is trained on an existing dataset with ground truth annotations. Next, the trained model is fed images from a different unlabelled multi-view setup to capture a set of pseudo labels for poses. These pseudo labels act as annotations for the new data after being filtered using multi-view consistency. The filtered labels are not enforced supervision during training. This allows the fusion model to adapt to any camera configuration without any labelled data.

### 3.4.4.2. RPSM

With the 2D keypoints from the fusion model, the network now requires to lift them to 3D space. This is done with the help of a recursive Pictorial Structure Model (RPSM). PSMs discretize state space to assemble a collection of parts, or joints in this case, in a deformable configuration by lifting 2D coordinates to 3D space while maintaining their spatial relationships. First, the root joint is triangulated to 3D space from one or more views. Next, the 3D bounding volume is discretized by an

*N*x*N*x*N* grid *G* centered around the root joint. These $N^3$ bins are defined by the 3D position of their centers in the world coordinate system and are treated as probable locations for the body joints such that all body joints share the same state space *G*. Increasing *N* increases the granularity of the space and, thus, lowers quantization errors. Finally, pairs of joints are connected with undirected graphs to form the skeletal structure of the deformable model. Optimization occurs according to how well each joint matches its location, called *unary potentials*, as well as how well the spatial relationships are maintained between connected pairs, called *pairwise potentials*. This optimization, however, has a computational complexity of the order $O(N^6)$, making it impractical for very accurate triangulations as it requires large *N* values.

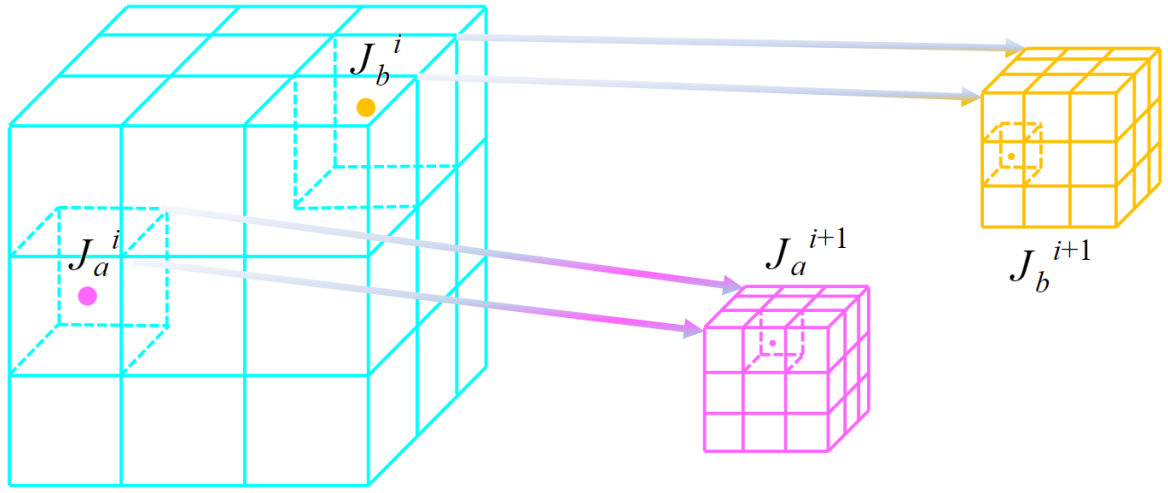



**Figure 30:** RPSM iteration. After the initial subdivision, each bin containing a joint is subdivided further in subsequent stages [Qiu et al. 2019]

In CrossView, the unary potential for each bin in *G* is calculated by projecting its 3D position to all camera views and retrieving the corresponding joint confidences using the multi-view 2D heatmaps from the fusion model. These confidences are then averaged over all views to get the unary potential. Pairwise potentials maintain spatial relationships between joints by enforcing every edge in the PSM to match the mean length of the corresponding edge in the training data. To this end, limb length priors are pre-calculated before training; edge lengths that reasonably match their corresponding priors, i.e. within a threshold $\epsilon$ = 150mm, are assigned a pairwise potential of 1, or 0 otherwise.

In the first stage, the initial PSM is created with *N* = 16, which means the initial PSM has very large quantization errors. To overcome this without adding excessively high computation load, the authors developed a novel recursive design. From the initial PSM, for each subsequent stage, the bins of each joint are subdivided into *N* = 2 grids recursively (Figure 30). This increases granularity at every stage as the edge length of the bins decreases, while allowing every joint to have its own grid space. Further, because bins are subdivided with a small *N* value, the computational overhead for each following stage is minimal.

The resulting RPSM model decreases the error by at least 50% compared to PSM with little increase of inference time. RPSM also performs significantly better than direct triangulation when input 2D poses are inaccurate (94.54mm vs 47.82mm) which shows its efficacy for real-world scenarios. It achieves comparable performance to direct triangulation when input 2D poses are very accurate, but this is not common in practice. A major drawback of the model is its dependence on camera parameters for the triangulation of the root joint and for projecting bin coordinates to the

camera views. This makes it a poor choice for 3DHPE on in-the-wild videos. Additionally, CrossView, like VideoPose, only allows single-person pose estimation.

### 3.4.5. Learnable Triangulation of Human Pose

Similar to CrossView, learnable triangulation of human pose [Iskakov et al. 2019] also leverages data from multi-view settings but unlike CrossView, it does so during the 3D reconstruction step; 2D estimation requires a separate pass for each view. For the remainder of the text, we will refer to this module as LearnTri. LearnTri experiments with two models; one based on algebraic triangulation and the other on volumetric triangulation. LearnTri with volumetric triangulation (Figure 31) is the current multi-view state-of-the-art on Human3.6M.

The 2D estimator for LearnTri is analogous to that for CrossView, with a ResNet-152 backbone followed by three transposed convolutional layers and ending in a convolutional layer. Each transposed convolutional layer uses 256 filters with kernel size 4, stride 2 and padding value of 1, and is followed by batch normalization and ReLU activation. This produces a set of intermediate heatmaps. The network allows an input image size of 384x384 while the resolution of the output heatmap is 96x96. From here, the architecture for the two LearnTri models diverges.

#### 3.4.5.1. Algebraic Triangulation

This model adds a final layer to the 2D estimator to perform convolutions with kernel size 1 and generate interpretable heatmaps, with output channels $J = 17$ where $J$ is the number of joints. This produces a distinct heatmap for every joint in every view, from which the 2D joint positions can be inferred by determining the center of mass of the heatmaps. With the 2D joints positions, linear algebraic triangulation is used to infer their 3D estimates where each view is processed independently. This naive approach, however, suffers if all camera views contribute equally to the triangulation as 2D joints positions cannot be estimated reliably in all views all the time due to occlusion and other factors. To mitigate this problem, a convolutional network is added with two convolutional layers, global average pooling and three fully-connected layers. This acts as an attention mechanism for the network, allowing it to control the contribution of each camera view in the final detection. The entire model is trained end-to-end and achieves considerable improvement (> 50%) over the compared multi-view techniques. For monocular view, the model still performs comparable to other monocular state-of-the-arts. A major drawback of this model is that it requires a separate pass for each camera view. This makes it impossible to filter out cameras with wrong projection matrices. Adding a 3D human pose prior is also not trivial in this case.

#### 3.4.5.2. Volumetric Triangulation

To overcome the limitations of the algebraic model, the authors experiment with a more complex way of triangulation. This is done by un-projecting the feature maps produced by the 2D backbone into 3D volumes. Following the base 2D estimator, a convolutional layer is added with a kernel size 1 and $K = 32$ output channels. The human pelvis is estimated using the above algebraic triangulation model and an $L$x$L$x$L$ grid is created in the global 3D space centered around the pelvis, where $L = 2.5$ m. This grid is discretized by a volumetric cube of dimensions 64x64x64 where each voxel is defined by the global 3D coordinates of its center. These 3D coordinates are then projected to each camera view using their projection matrices, and bilinear sampling is used on the 2D heatmaps from the camera view to fill the cube. This gives a different volumetric map for each camera view, which is then aggregated to remove the model's dependency on the number of camera views for the subsequent

steps. Three different aggregation techniques are explored. The first approach uses a simple raw summation of the voxel data. This adds contribution equally from each camera view. In contrast, the second and third approaches use scores to scale the contribution from each camera view. For the second approach, normalized confidence scores generated by the network are used as the scaling factor during summation. For the third approach, softmax is computed on each voxel to obtain the volumetric coefficient distribution, which is then used as the scaling factor during summation, similar to approach two.

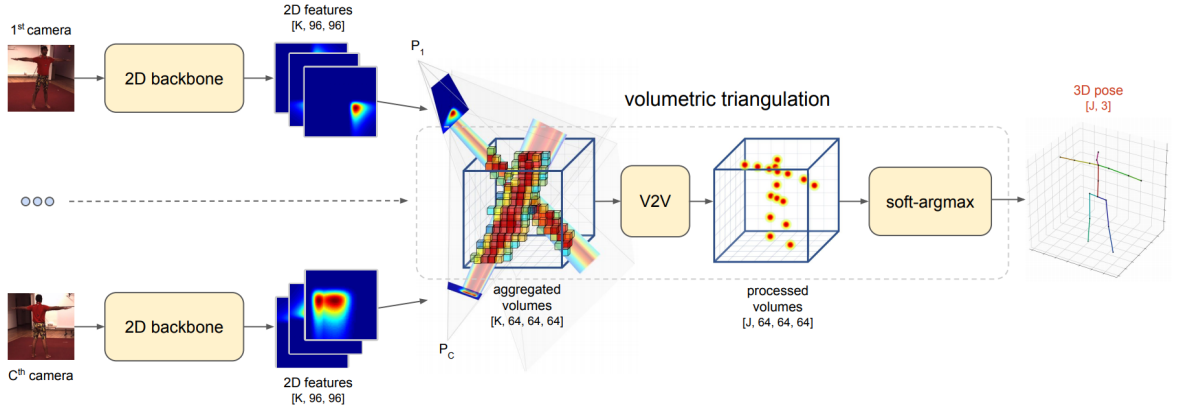



**Figure 31:** LearnTri volumetric triangulation pipeline [Iskakov et al. 2019]

The aggregated maps are then fed to a Voxel-to-Voxel PoseNet [Moon et al. 2018], an encoder-decoder CNN that estimates the per-voxel likelihood for each keypoint. This produces interpretable 3D heatmaps, over which softmax is applied across the spatial axes, and the center of mass for each volumetric map is estimated to finally infer the 3D joint positions.

The entire model can be trained end-to-end and shows significant improvement over the algebraic model (~30%). Using the naive approach for aggregation, i.e. approach one, the model already outperformed their algebraic model by about 20%. Approaches two and three both improve the model performance slightly over the naive approach. They perform very comparable to each other on monocular input. For multi-view input, the softmax approach performs slightly better than approach two.

A major drawback of LearnTri is its dependence on camera parameters for the triangulation of the pelvis joint and for projecting 3D coordinates from the volumetric maps to the camera views. This makes it a poor choice for 3DHPE on in-the-wild videos. Additionally, LeanTri, like VideoPose, only allows single-person pose estimation.

### 3.4.6. VIBE

VIBE [Kocabas et al. 2019] is a monocular multi-person model-based 3D pose estimator that only depends on a sequence of images as input. It uses adversarial training (**Section 2.2.3.4**.) to estimate the 3D body pose and body shape by leveraging the AMASS [Mahmood et al. 2019] corpus for training.

#### 3.4.6.1. The Generator

The goal of the generator is to produce synthetic SMPL parameters that are in line with SMPL parameters for the AMASS corpus. First, a temporal sequence of T = 16 frames is fed to a ResNet-50 network to estimate 2D body pose and body shape parameters. These are then fed to a 2-layer Gated

Recurrent Unit (GRU) [Cho et al. 2014]. GRUs are temporal encoders that use feedforward and feedback connections to process time series data. Both GRU layers use a hidden size of 1024 to produce feature vectors for every frame using the previous frame. Lastly, the body pose and body shape parameters are fed to a pre-trained linear regressor along with the generated feature vectors from the GRU. This regressor uses 2 fully-connected layers with 1024 hidden units and a final layer to generate the SMPL parameters.

#### 3.4.6.2. The Discriminator

The motion discriminator tries to differentiate generated SMPL parameters from the learned true-class distribution, thus, forcing the generator to produce feasible real world poses. It uses an identical setup as the temporal encoder with 2 GRU layers each containing 1024 hidden units. Checking the temporal continuity of the generated parameters in this way helps in discarding multiple inaccurate poses that may be recognized as valid individually. The GRUs produce a latent code at each time step which is then aggregated using a 2-layer feedforward network with 1024 nodes each. This feedforward network acts as the self-attention mechanism. The weights of this network are normalized using softmax to produce a probability distribution and help the discriminator to amplify contributions from the most important frames. Finally, a linear layer predicts the probability of the generated parameters belonging to the learned distribution.

The discriminator is first trained individually using the AMASS corpus to learn the true class distribution. Then, the generator is trained using the pre-trained discriminator to produce feasible parameters. This in turn also fine-tunes the discriminator. VIBE achieved state-of-the-art results on the 3DPW [Marcard et al. 2018] dataset. This dataset contains 60 videos of fully annotated 3D human poses in outdoor environments as opposed to the controlled, indoor environments in Human3.6M.

## 3.5. A Comparison of 3D Human Pose Estimation Techniques

We have four 3DHPE techniques that show promise but have certain dependencies: VideoPose relies on a robust 2D estimator, and LearnTri and Crossview rely on camera projection matrices, while VIBE only requires a video sequence. Camera projection matrices are not always available for in-the-wild videos. Supplying projection matrices is especially hard if camera movement is involved such as zooming, which is often true for real world scenarios. To discount the model from relying on such parameters, we choose VideoPose and VIBE as the more ideal choices for our research. We perform a preliminary analysis (**Section 3.5.1.**) on YOUth to test the efficacy of OpenPose for 2DHPE, and VideoPose and VIBE for 3DHPE for in-the-wild parent-child interaction videos. Next, we get a baseline estimate for VideoPose, which is our technique of choice for 3DHPE after the preliminary evaluation. To this end, we compare it with the current SOTAs on multi-view 3D pose estimation (**Sections 3.5.2.-3.**).

### 3.5.1. Preliminary Analysis

For the preliminary analysis, we use YOUth (**Section 3.2.2.**). All three analyzed techniques, OpenPose, VideoPose and VIBE work on monocular video. Thus, we split the views in YOUth to individual video streams. 2D annotations for VideoPose are generated using the suggested keypoint detector for in-the-wild scenarios, Detectron [Girshick et al. 2018] which uses a Mask R-CNN-like architecture specialized for keypoint detection. This analysis is only performed on the qualitative level.

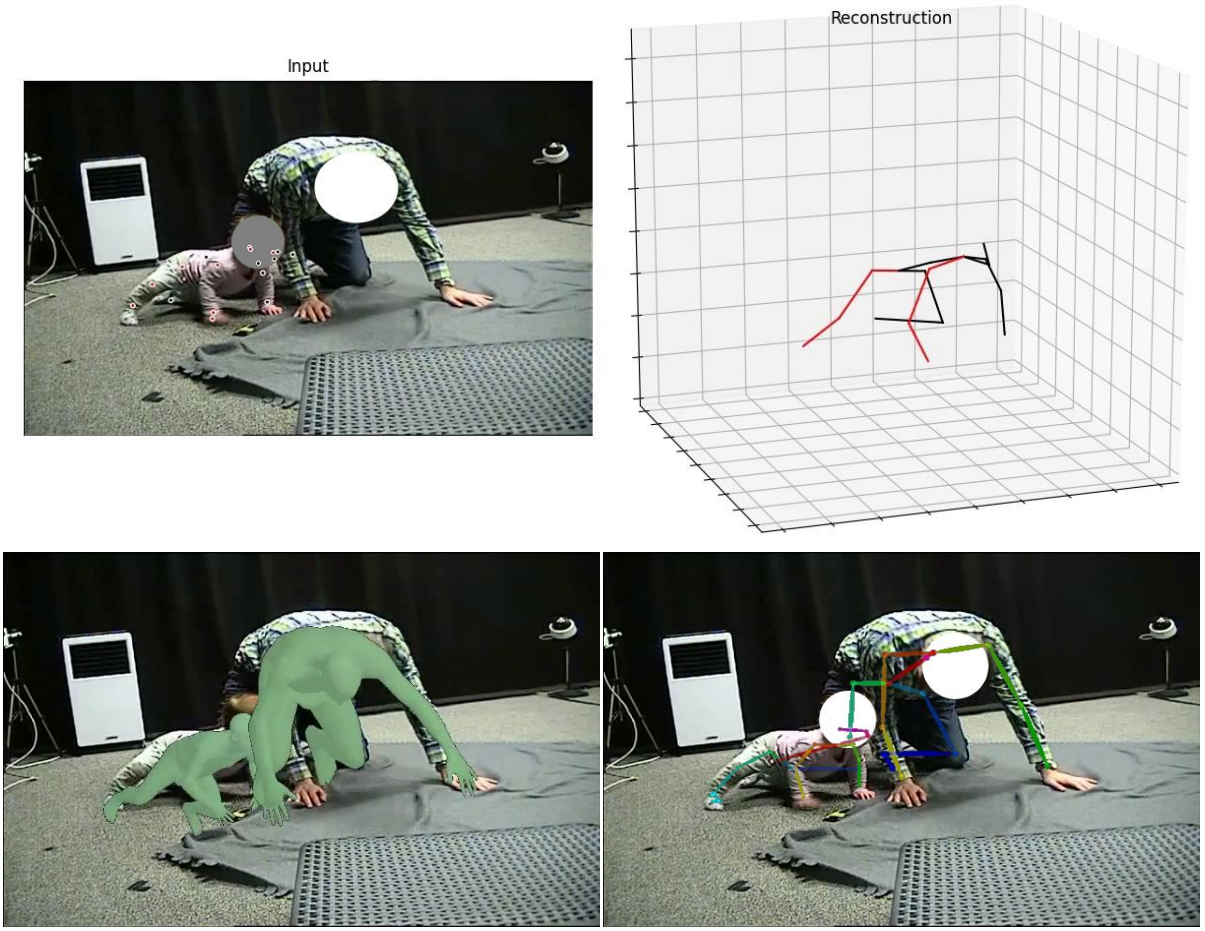
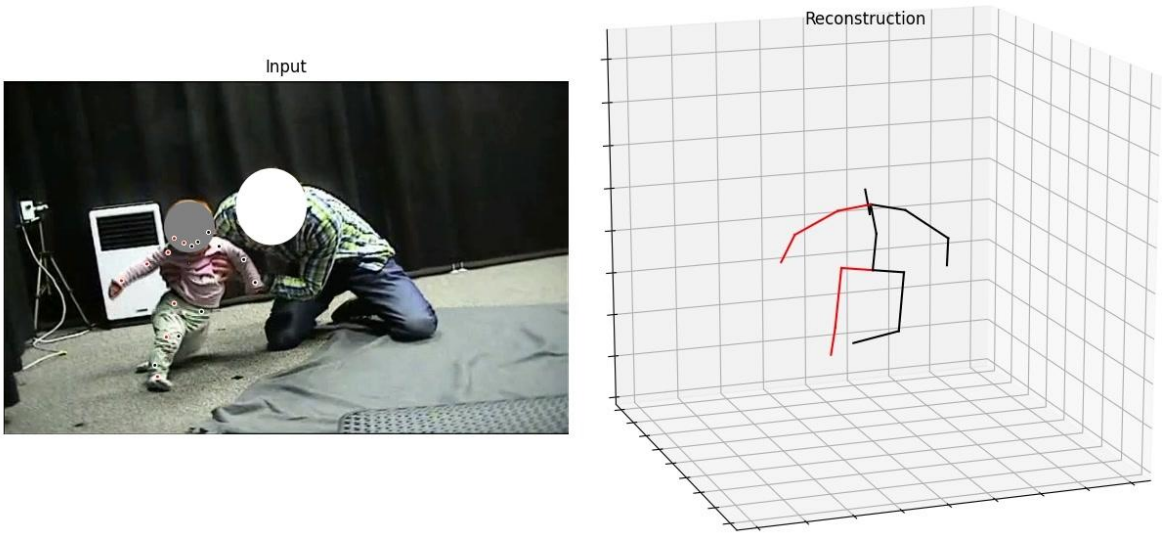



**Figure 32:** Qualitative analysis results. VideoPose (top), VIBE (bottom-left) and OpenPose (bottom-right)



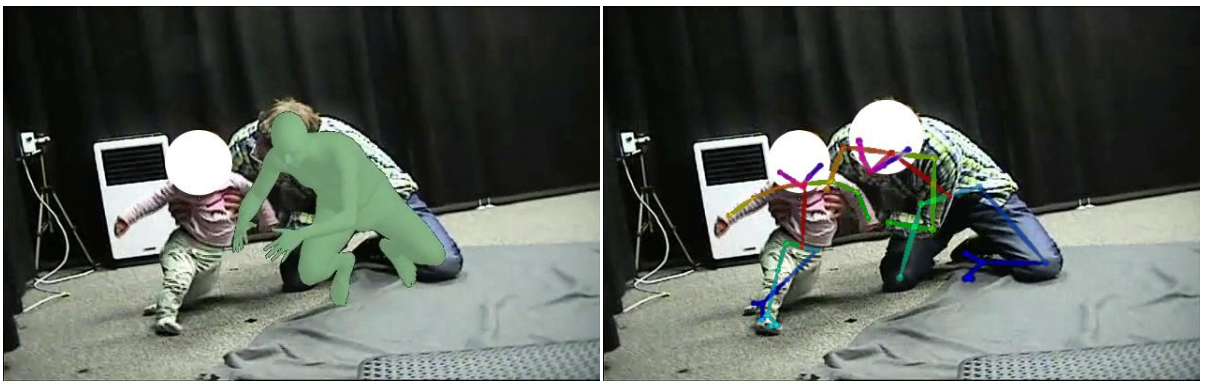


**Figure 33:** Qualitative analysis results. VideoPose (top), VIBE (bottom-left) and OpenPose (bottom-right) Here, VIBE fails to detect multi-person instances.

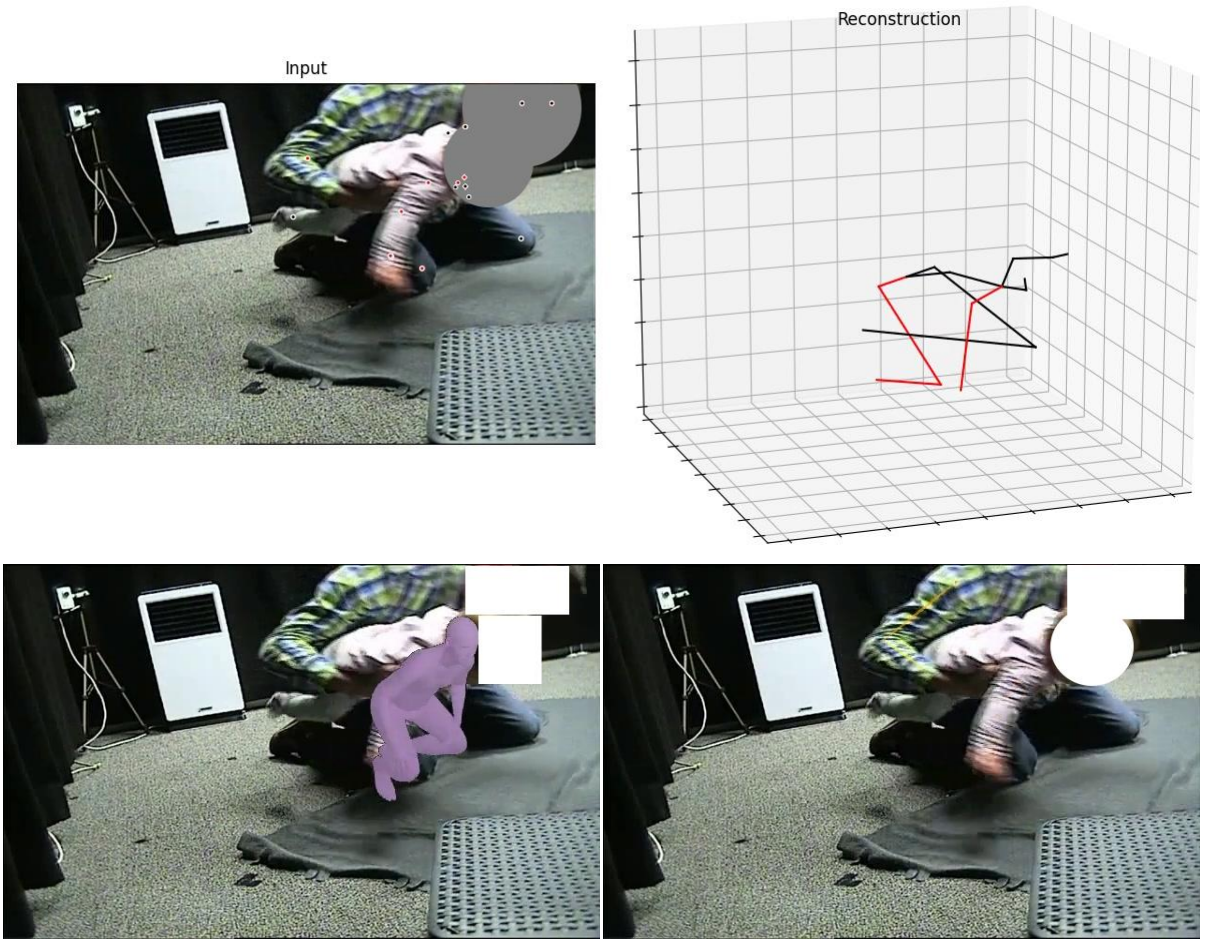



**Figure 34:** Qualitative analysis results: Hard cases. VideoPose (top), VIBE (bottom-left) and OpenPose (bottom-right)

*Discussion:*

In general, OpenPose works reasonably well (Figures 32 and 33) while maintaining near real-time performance (~15FPS on Nvidia RTX2070). Problem cases exist (Figure 34 (bottom-right)) but are limited to small windows of frames and can be largely solved using interpolation. VIBE, in general, is very unstable and jittery. It fails to detect the infant as an instance rather frequently while also failing

to align the estimated 3D shapes with the detected instances satisfactorily. This is especially worse in close-proximity cases due to the used instance detection. This may be attributed to the AMASS corpus used to train VIBE, which does not represent infant-scale humans or actions like crawling/kneeling very well. VideoPose expects a single-person, and performs adequately stable when keypoints belong to the same instance for consecutive frames (Figures 32 and 33). If keypoints jump to-and-fro between instances, such as in cases of close proximity, the temporal convolutions can cause it to "fuse" 3D poses from both instances. As seen in Figure 34 (top), the lower body of the reconstruction represents the parent whereas the upper body resembles the infant more closely. This leads us to believe that providing better instance segmentations could improve VideoPose results. Note that the 3D reconstruction of VideoPose gets destroyed while the camera is moving and takes a few frames to re-stabilize post-camera movement. This artefact is not prevalent in VIBE.

### 3.5.2. Evaluation Methodology

Evaluation is performed on subjects 9 and 11, referred to as S9 and S11, from the Human 3.6M dataset over 15 actions with 2 sub-actions each, for a total of 15x2x2 = 60 videos. We chose S9 and S11 because all three works also use these subjects for evaluation in their papers. The configuration used for each 3DHPE technique is specified below. An exhaustive list of the subactions used by each technique is provided in Table 4. MPJPE is used as the base error metric for all evaluations.

*Data configuration:*

We omit certain sub actions from the Human3.6M dataset for our evaluation (Table 3) to make the results more comparable. We first omit results for subaction 1 of the "Directions" action class for subject 11. This is done because VideoPose does not provide input 2D detections for this subaction. The reason for this is now known. We also omit the results for one subaction from the actions "Greeting", "SittingDown" and "Waiting" for S9 because the authors of both LearnTri and CrossView report that the respective subactions contain erroneous GT data. This anomaly is not observed in VideoPose since VideoPose uses *D3_positions* unlike LearnTri and CrossView which use *D3_positions_mono* and *D3_positions_mono_universal.*

*VideoPose3D configuration:*

We know that VideoPose3D does not depend on a specific 2D detector, i.e., a number of choices exist as long as the input format like joint order is matched. As reported by the authors (Table 1), 2D detections from their fine-tuned Cascaded Pyramid Network (CPN) with bounding boxes provided by their fine-tuned Mask R-CNN achieved the best results after GT 2D data. Therefore, we run VideoPose on the prepared CPN 2D detections for Human3.6M provided in their original git repository[7] for our evaluation. The network hyperparameters are as suggested in the original work (Table 2). As ground truth 3D data, VideoPose uses *D3_positions* which contains the 3D joints positions of the subjects in the world coordinate system. By default, VideoPose outputs poses in camera space. To transform them to global coordinates, we align it with the GT data before calculating the MPJPE in our evaluation, i.e., we use rigid alignment along scale, rotation and translation of the predicted poses with the GT poses as done in the original work.

[7] https://github.com/facebookresearch/VideoPose3D

| 2D Detections | Bounding Boxes | Error (P1) | Error (P2) |
|---|---|---|---|
| CPN | Mask R-CNN | 46.8 mm | 36.5 mm |
| CPN | GT | 47.1 mm | 36.8 mm |
| Mask R-CNN | Mask R-CNN | 51.6 mm | 40.3 mm |
| GT | GT | **37.2 mm** | **27.2 mm** |

**Table 1:** Results on VideoPose for different 2D input sources. P1 reports the MPJPE while P2 reports P-MPJPE, i.e. the MPJPE after aligning the predictions with the GT data. (Taken from [Pavllo et al. 2019])

| | |
|---|---|
| number of Joints, $\boldsymbol{J}$ | 17 |
| Kernel size, $\boldsymbol{W}$ | 3x3 |
| Output channels, $\boldsymbol{C}$ | 1024 |
| number of ResNet blocks, $\boldsymbol{B}$ | 4 |
| Dropout rate, $p$ | 0.25 |
| effective Receptive Field Size, $\boldsymbol{F}$ | 243 |

**Table 2:** VideoPose hyperparameters used for all experiments and results.

*CrossView configuration:*

The base model of CrossView is used which is pre-trained on Human3.6M for the evaluation. No changes are made to the model for our evaluation. As GT data, it uses the *D2_positions* for the 2D detections. For the 3D detections, *D3_Positions_mono*, which records the 3D joint positions in each camera space and the *D3_Positions_mono_universal* which records the 3D joint positions in global space for each camera view.

*LearnTri configuration:*

The volumetric model of LearnTri is used which is pre-trained on Human3.6M for the evaluation. No changes are made to the model. LearnTri uses the same GT data as CrossView; *D2_positions* for the 2D detections and *D3_Positions_mono* + *D3_Positions_mono_universal* for the 3D detections.

### 3.5.3. Evaluation Results

*Quantitative:*

All quantitative evaluation results are reported in Tables 4 and 5.

| Action Class | Code | Sub-action | VideoPose | | CrossView | | LearnTri | |
|---|---|---|---|---|---|---|---|---|
| | | | S9 | S11 | S9 | S11 | S9 | S11 |
| Directions | 02 | 1 | ✔ | ✘ | ✔ | ✘ | ✔ | ✘ |
| | | 2 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| Discussion | 03 | 1 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| | | 2 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| Eating | 04 | 1 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| | | 2 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| Greeting | 05 | 1 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| | | 2 | ✘ | ✔ | ✘ | ✔ | ✘ | ✔ |
| Phoning | 06 | 1 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| | | 2 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| Posing | 07 | 1 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| | | 2 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| Purchases | 08 | 1 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| | | 2 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| Sitting | 09 | 1 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| | | 2 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| SittingDown | 10 | 1 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| | | 2 | ✘ | ✔ | ✘ | ✔ | ✘ | ✔ |
| Smoking | 11 | 1 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| | | 2 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| TakingPhoto | 12 | 1 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| | | 2 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| Waiting | 13 | 1 | ✘ | ✔ | ✘ | ✔ | ✘ | ✔ |
| | | 2 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| Walking | 14 | 1 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| | | 2 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| WalkingDog | 15 | 1 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| | | 2 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| WalkingTogether | 16 | 1 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |
| | | 2 | ✔ | ✔ | ✔ | ✔ | ✔ | ✔ |

**Table 3:** Action-subaction class used to compute MPJPE for each 3DHPE technique

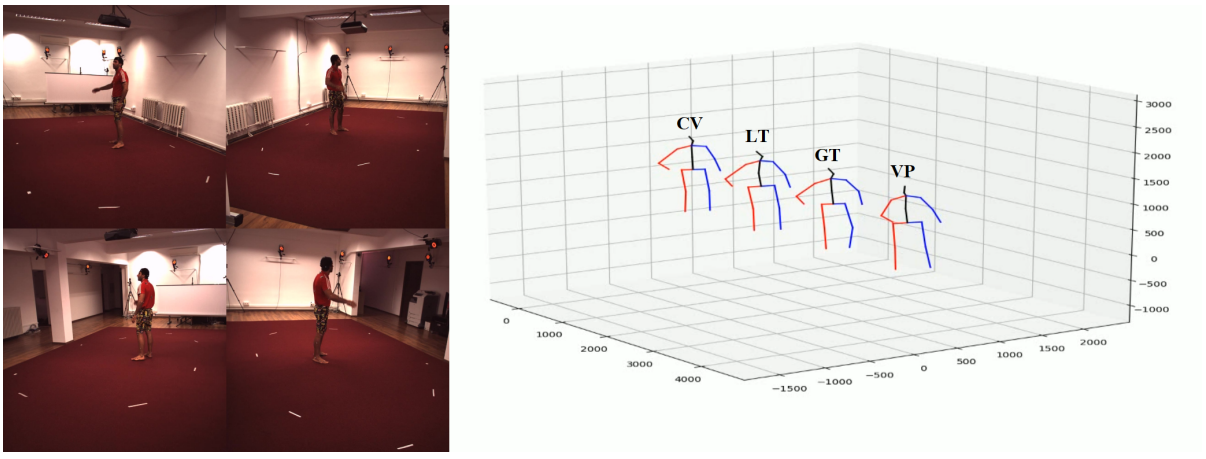
| | Joint code | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | root | rhip | rknee | rank | lhip | lknee | lank | belly | neck | nose | head | lsho | lelb | lwri | rsho | relb | rwri | mean |
| VideoPose | | | | | | | | | | | | | | | | | | |
| S9 | 24.27 | 36.77 | 49.21 | 65.68 | 36.90 | 47.77 | 56.71 | 29.69 | 23.46 | 35.02 | 37.67 | 35.15 | 49.16 | 65.81 | 32.51 | 58.42 | 85.65 | 45.29 |
| S11 | 30.28 | 33.58 | 43.96 | 66.10 | 38.59 | 44.46 | 61.48 | 25.05 | 27.38 | 34.31 | 34.29 | 29.71 | 46.19 | 74.06 | 25.49 | 57.86 | 89.34 | 44.83 |
| mean | 27.28 | 35.17 | 46.59 | 65.89 | 37.75 | 46.11 | 59.09 | 27.37 | 25.42 | 34.66 | 35.98 | 32.43 | 47.67 | 69.94 | 29.00 | 58.14 | 87.49 | 45.06 |
| CrossView | | | | | | | | | | | | | | | | | | |
| S9 | 14.87 | 30.60 | 43.31 | 66.17 | 28.04 | 47.17 | 69.70 | 30.10 | 27.41 | 29.10 | 30.92 | 35.70 | 40.79 | 40.07 | 36.13 | 43.94 | 42.80 | 38.64 |
| S11 | 3.70 | 17.77 | 25.35 | 34.02 | 16.18 | 33.35 | 39.11 | 19.46 | 18.54 | 16.08 | 16.96 | 20.42 | 27.89 | 29.50 | 23.90 | 28.75 | 29.08 | 23.53 |
| mean | **9.29** | **24.19** | 34.33 | 50.10 | **22.11** | 40.26 | 54.41 | 24.78 | 22.98 | 22.59 | 23.94 | 28.06 | 34.34 | 34.78 | 30.02 | 36.34 | 35.94 | 31.09 |
| LearnTri | | | | | | | | | | | | | | | | | | |
| S9 | 23.74 | 37.14 | 30.30 | 46.81 | 30.48 | 35.19 | 53.41 | 32.46 | 24.76 | 22.23 | 17.18 | 33.62 | 26.04 | 25.14 | 35.14 | 31.82 | 26.87 | 31.31 |
| S11 | 14.68 | 21.37 | 16.12 | 17.62 | 17.28 | 18.74 | 19.70 | 14.43 | 12.21 | 13.95 | 5.07 | 16.29 | 11.25 | 14.02 | 14.92 | 12.56 | 14.00 | 14.95 |
| mean | 19.21 | 29.25 | **23.21** | **32.22** | 23.88 | **26.97** | **36.56** | **23.45** | **18.48** | **18.09** | **11.12** | **24.95** | **18.64** | **19.58** | **25.03** | **22.19** | **20.43** | **23.13** |

**Table 4:** MPJPE per joint for our three 3DHPE techniques.
**bold** = best score; underlined = best score per subject; blue = Aggregate mean

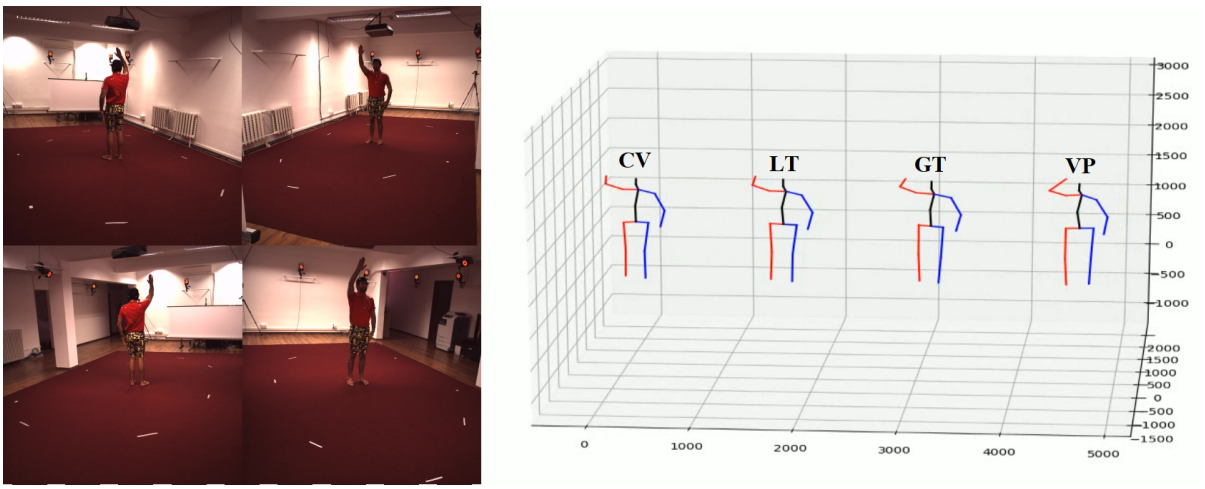
| | Action code (ref. Table 3) | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| VideoPose | | | | | | | | | | | | | | | |
| S9 | 44.81 | 42.65 | 37.83 | 46.45 | 54.11 | 39.34 | 39.57 | 62.08 | 58.12 | 48.17 | 52.48 | 39.44 | 33.00 | 41.79 | 32.65 |
| S11 | 44.41 | 40.10 | 50.40 | 38.64 | 45.88 | 36.13 | 38.07 | 55.40 | 73.27 | 46.82 | 55.78 | 37.92 | 25.41 | 42.93 | 28.87 |
| mean | 44.61 | 41.38 | 44.11 | 42.55 | 49.99 | 37.73 | 38.82 | 58.74 | 65.70 | 47.50 | 54.13 | 38.68 | 29.20 | 42.36 | 30.76 |
| CrossView | | | | | | | | | | | | | | | |
| S9 | 26.38 | 31.98 | 22.92 | 25.95 | 57.45 | 24.63 | 24.46 | 38.59 | 30.91 | 57.72 | 29.06 | 45.34 | 69.79 | 25.64 | 29.53 |
| S11 | 32.18 | 22.40 | 19.13 | 22.31 | 22.38 | 24.41 | 23.71 | 21.49 | 23.40 | 27.63 | 25.36 | 25.58 | 26.75 | 21.47 | 19.15 |
| mean | 29.28 | 27.19 | 21.02 | 24.13 | 39.91 | 24.52 | 24.09 | 30.04 | 27.15 | 42.67 | 27.21 | 35.46 | 48.27 | 23.56 | 24.34 |
| LearnTri | | | | | | | | | | | | | | | |
| S9 | 19.51 | 20.78 | 19.29 | 18.92 | 50.35 | 19.27 | 19.26 | 23.54 | 23.10 | 54.74 | 20.92 | 33.48 | 67.87 | 20.31 | 20.85 |
| S11 | 21.54 | 16.23 | 12.65 | 13.79 | 14.35 | 15.64 | 15.13 | 14.05 | 15.70 | 15.68 | 15.61 | 14.95 | 14.80 | 13.41 | 13.72 |
| mean | **20.52** | **18.50** | **15.97** | **16.35** | **32.35** | **17.46** | **17.19** | **18.80** | **19.40** | **35.21** | **18.27** | **24.22** | **41.33** | **16.86** | **17.29** |

**Table 5:** MPJPE per action for our three 3DHPE techniques.
**bold** = best score; underlined = best score per subject;

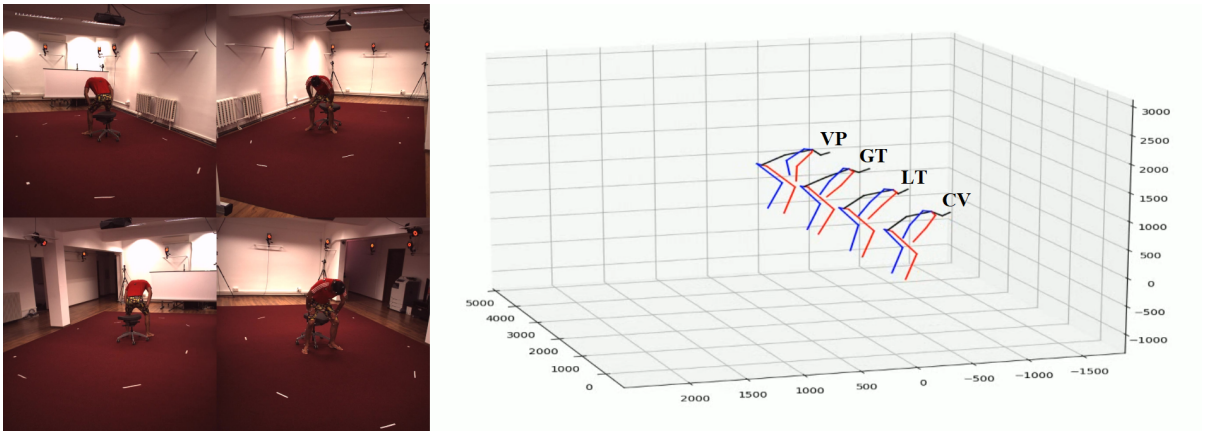
*Qualitative*:



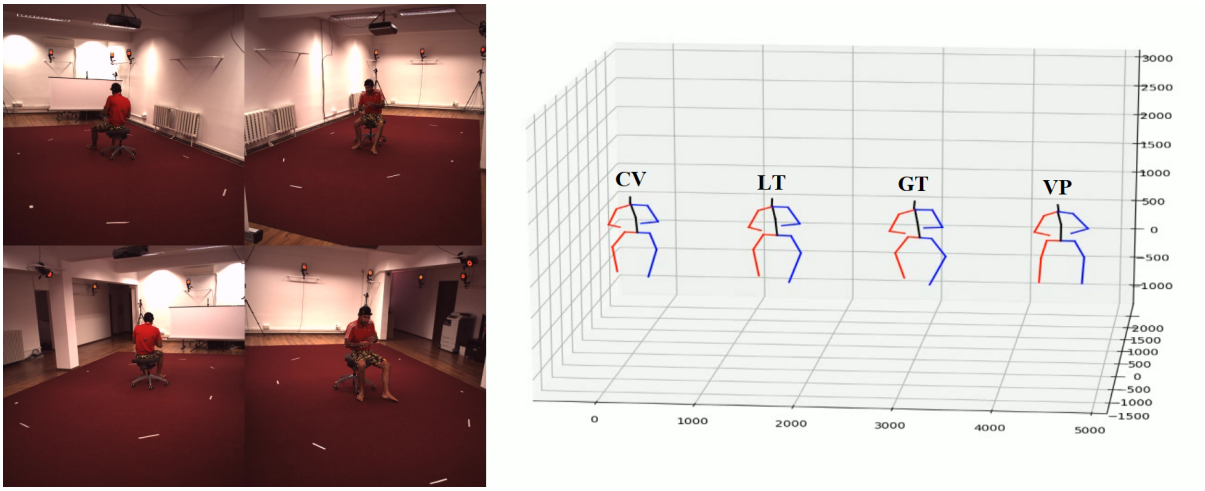

(a)




(b)




(c)




(d)

**Figure 35:** Qualitative results on Human3.6M.
CV = CrossView, LT = LearnTri, GT = Ground Truth, VP = VideoPose
Note that VideoPose only uses the top-left view i.e.view 1

### 3.5.4. Discussion

It is clear from the quantitative and qualitative evaluations that LearnTri and CrossView both outperform VideoPose by modest margins. However, considering VideoPose does not leverage data from other views, i.e. it only uses a single-view, there is much room for improvement. Further, VideoPose does not require camera intrinsics and shows clear improvement with a better 2D detector (Table 1). For these reasons, we choose VideoPose as the ideal model for our research. The evaluations performed so far provide a good estimate on how VideoPose compares numerically to the state-of-the-art multi-view 3D pose estimators.

In general, we observe that using multi-view helps in cases where limbs are not visible in monocular view (Figures 35 (c),(d)). VideoPose only uses the top-left view, and so, often has limbs in the wrong orientation. However, there are a few artefacts prevalent in both multi-view techniques that are less severe in VideoPose. Both Crossview and LearnTri tend to bend the knee joints inwards (Figures 35 (a),(b)) unnaturally. This is less severe when the legs are far apart (Figure 35 (d)), but is very noticeable in other cases. Additionally, the belly joint is frequently bent more than the ground truth for all three techniques, although it is less severe in VideoPose.

## 3.6. Our Approach

VideoPose only supports single person 3DHPE. For our purpose, we want to extend VideoPose to allow multi-person pose estimation. A naive approach would be to extract each instance and feed it to VideoPose frame-by-frame. This however does not allow any benefit to the temporal convolutions in VideoPose, leading to very jittery results. To this end, we develop a new pipeline (Figures 36 and 37). For every frame, we first use Mask R-CNN to extract masks and bounding boxes for every person present in the frame. Mask R-CNN  Next we use a color matching module (explained below) to match instances across frames and split them into separate video streams. Once all frames have been split, OpenPose is run on every stream to extract the 2D poses for that individual human instance. Finally, VideoPose is run using the 2D joint positions and bounding boxes on every stream to extract the 3D joint positions. We use Gift Delay to test our pipeline.

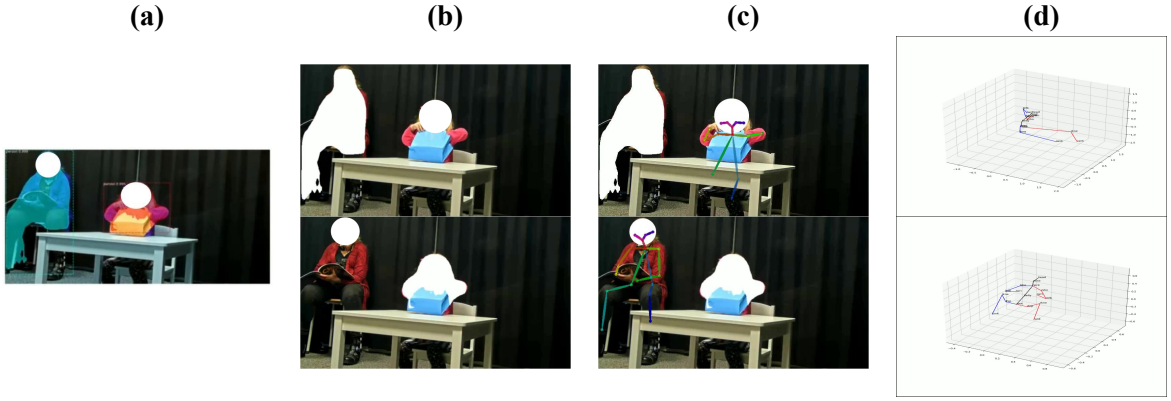



**Figure 36:** Our pipeline. (a) First we perform instance segmentation, (b) next we split the instances, (c) then we extract 2D pose information with OpenPose, and (d) finally we lift the 2D pose to 3D space using VideoPose

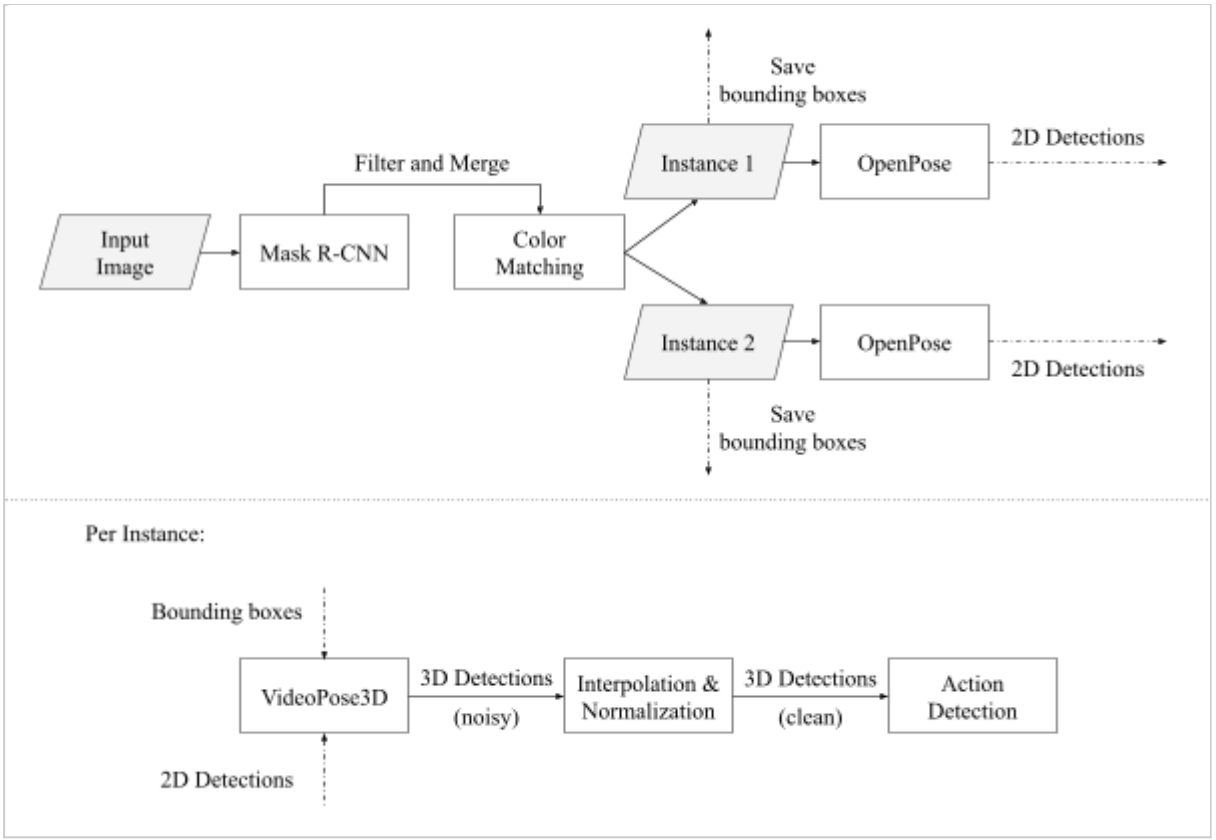



**Figure 37:** Schematic diagram of our pipeline

### 3.6.1. Color-Matching Module v1:

Given an input image with bounding boxes and masks for each person instance, we first normalize the pixel values. This is done to lower the variation in lighting across frames and views. Next we bin RGB pixel values into an 8x8x8 color histogram and normalize the histogram. This is done for all instances in the frame. Naively, we could use the binary mask to bin only the relevant pixels but this is complicated when detections overlap. To overcome this, we only bin pixels that exclusively belong to an instance and do not bin pixels shared by multiple masks. For the first frame, once the color models are constructed, we assign each a unique label and save it separately for the remainder of the video stream across all views. This helps in detecting matching instances in the first frame for subsequent views. For subsequent frames, these color models are used to match instances across frames and views. Given a set of color models from the first frame, $C$, with the corresponding labels $L$ and a new set of histograms for frame two, $H$, we build a dense matrix of dimensions $C$x$H$ with chi-squared distance to find the similarity between every combination of histogram and color model. Then, we assign to each histogram in $H_i$ the corresponding label of $C_i$ with the lowest score, i.e. highest similarity. This can lead to multiple histograms being assigned the same label. To solve this, the histogram with the highest similarity is retained and the next most similar label is picked for the duplicates. This in turn could then make it conflict with another histogram label, which is again solved as above. This operation is not terminated until each histogram is assigned a unique saved label. If the distance score surpasses a threshold, it is considered a new label and the histogram is saved as a new color model.

The final step is to split the instances by whiting out the masks exclusive to other detections to have a single instance per split. These splits are then saved and then later combined into videos.

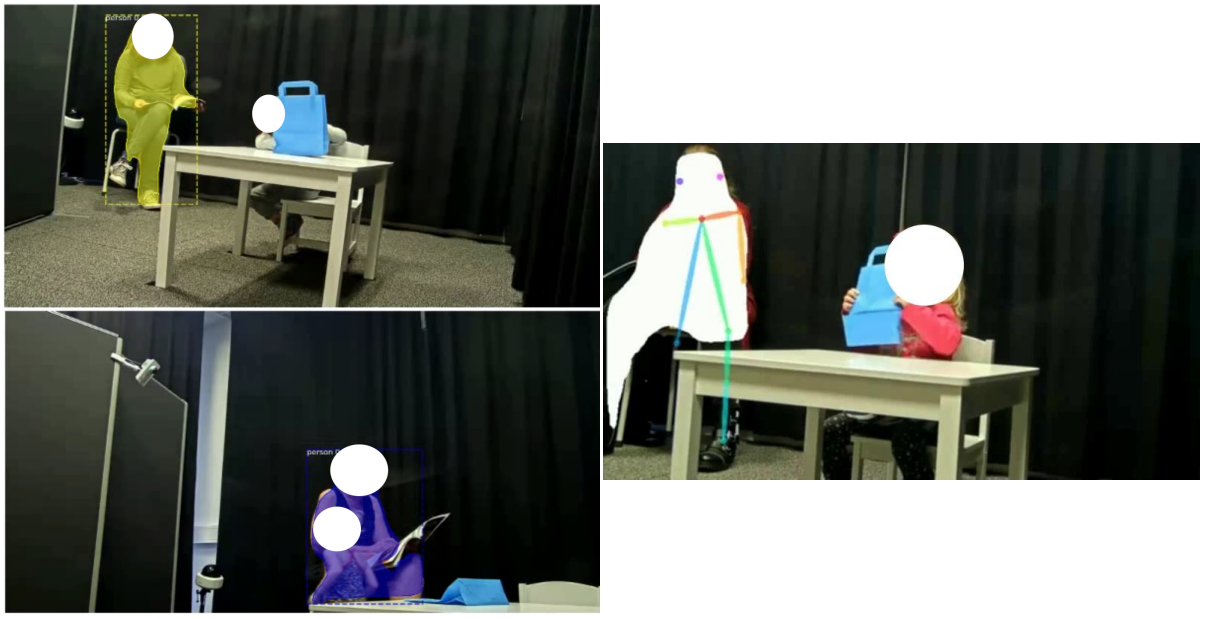


**Figure 38:** Problems in our used modules. (left) Mask R-CNN (right) OpenPose

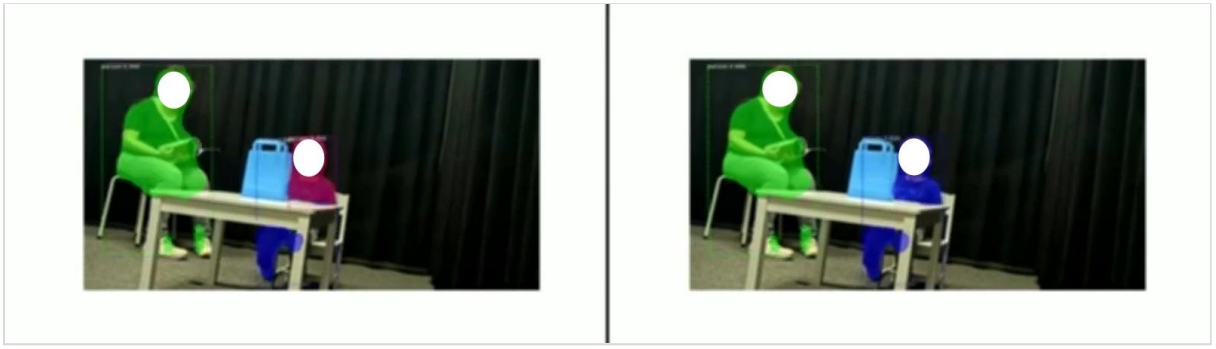
### 3.6.2. Improved Color-Matching:

The initial color matching module does not split instances very well. One of the main bottlenecks is the used Mask R-CNN module. Many instances exist where masks either overlap completely (Figure 39 (a) (left)) or fragmented masks belong to the same instance (Figure 39 (b) (left)). This is more prevalent in occluded views. As a result, multiple instances of the same person are created with distinct labels, eventually causing label-switching. We solve this by post processing the output from Mask R-CNN and merging masks with very high overlap (Figure 39 (right)). Another problem with Mask R-CNN is that it fails to detect occluded instances well (Figure 38 (left-top)), or detect instances as separate in close proximity cases (Figure 38 (left-bottom)). Mask R-CNN also seldom detects false positives in the background. These issues are hard to solve without retraining Mask R-CNN. An additional problem lies in the split step after color-matching. Even though we whiteout the other instances, during 2D pose estimation OpenPose sometimes detected the omitted instance as the primary instance (Figure 38 (right)) because the white shape closely resembled a humanoid. The implications of this artefact get worse while lifting because VideoPose cannot exploit temporal information reliably due to inconsistent 2D pose estimates. To curtail this, we instead only crop out the instance using its bounding box after inflating it to a minimum size and pad it with empty pixels to retain the original image size. This allows for a cleaner segmentation when the instances are not in near-proximity, albeit close-proximity cases are still haunted by this artefact.

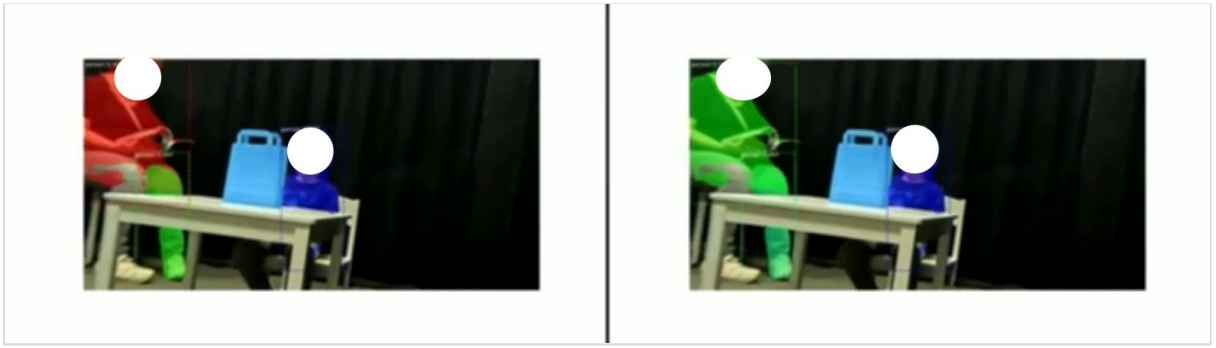
### 3.6.3. Pose Estimation and Lifting:

With the instances separated, we now have a separate video for every subject in every view. Next, we use single-person OpenPose on every stream to obtain the 2D keypoints which are then reordered to match the VideoPose input sequence and saved. The 2D keypoints along with the bounding boxes from Mask R-CNN are finally combined and fed to VideoPose to lift the keypoints to 3D coordinate space. We focus on view-2 for our analyses as it is ideal to analyze the interactions; it captures both the parent and the infant more thoroughly than the other views. Although we do not detect interactions

yet, it is the better choice so as to have a good understanding of the challenges. View-2 however often captures the infant heavily occluded by the gift bag. In such cases, the 2D estimates, and consequently, the 3D estimates suffer from severe noise. Multi-view inferencing can help curtail this to a good extent. However, VideoPose only provides 3D estimates and does not return confidence scores. This makes it hard to converge keypoints from multiple views.



**(a)**



**(b)**

**Figure 39:** Mask RCNN output pre (left) and post (right) mask-merging operation

### 3.6.4. Action Detection:

On extracting the 3D pose estimates, we can tackle the task of action detection. As the infants are the primary subjects in the used dataset, we focus on atomic actions performed by the infants. We approach the detection task using self-generated rules and vector arithmetic. The first step is to denoise the 3D estimates using interpolation and normalization. For this, we adapt [Poppe et al. 2014] for python and use the median operation for interpolation using 30 frames (1.2s) as the threshold. Next, we normalize with respect to position, orientation and limb lengths. This helps in reducing jitter while making estimates across frames more comparable. Once the 3D estimates have been cleaned up, we detect each action per frame using our detection algorithm. Finally, we filter for temporal consistency of the actions to curtail the effects of jitter. We only use the upper body joints for the detections and omit the hip joints, the knee joints, and the ankle joints. This is because the children are seated for a majority of frames and the lower joints are mostly occluded by the table in all views, thus contributing very little to the actions being performed. Further, we omit the belly joint and treat the pelvis-neck connector as a single segment for simplicity.

#### 3.6.4.1. Detection Rules

The actions we are interested in are listed in Table 6. The detection rule for each action type is described below.

| Action Type | Temporal consistency threshold |
|---|---|
| Hand Extension | 10 frames (0.4s) |
| Head Turn | 5 frames (0.2s) |
| Looking Up | 5 frames (0.2s) |
| Looking Down | 5 frames (0.2s) |
| Touching Face | 5 frames (0.2s) |

**Table 6:** Action classes and number of frames used for temporal consistency

*Hand Extension*

We calculate the angle between the shoulder-elbow limb and the elbow-wrist limb using simple vector arithmetic. This is done separately for the left and right arms. We use an angle threshold of 105° and treat the respective arm as extended if the calculated angle exceeds the threshold. Once this is calculated per frame, we filter it such that if the hand is extended for at least 9 out of 10 consecutive frames, the hand extension is considered as a correct detection.

Limitations: Theoretically, a hand extension is rooted around the elbow joint as well as the shoulder joint. Absence of the hip joint makes it very hard to calculate angles between the shoulder-elbow and shoulder-hip segments. For this reason, hand extension is not very robust. With only the angle between the shoulder-elbow and elbow-wrist segments, there exist cases where the arm is actually suspended straight downwards but is detected as an arm extension. An alternative approach would be to use a distance measure between the wrist and the shoulder and consider the hand as extended when this measure closely matches the wrist-elbow-shoulder limb length. However, this would also fail to differentiate between cases where the arm is extended and the arm suspended straight downwards. Moreover, hand extension may occur using only the lower half of the arm, which may not be well detected using the distance measure.

*Touching Face*

We calculate the absolute distance between the wrist joints and the head joint. If this distance matches the distance between the nose and head joints as a loose threshold, the child is considered to be touching the face. This is filtered over 4 out of 5 frames for temporal consistency.

Limitations: A touch may involve resting the entire hand/palm on the face as well as a single finger touching the face. With the absence of hand and additional face keypoints, it is inherently difficult to detect whether a hand is touching the face or not.

*Head Turn*

To detect a head turn, we find the change in angle of the nose joint between consecutive frames. For the head turn, we would like to get this change only along the horizontal axis. To this extent, we use the pelvis-neck vector as the normal for the plane of movement. To account for direction, we use

the nose displacement vectors and calculate the dot products between consecutive frames. If the nose angle differs by at least 3° between frames, the headturn flag is turned on. If the angle change is consistently in the same direction for at least 4 out of 5 frames, the child is considered to be turning their head.

Limitations: To detect the head turns accurately, we need a planar normal to measure change in angle of the nose keypoint only along a horizontal plane. While the pelvis-neck vector is a good substitute in cases where the infant is looking straight, if the head is raised up or down, the substitution often fails. Heavy occlusion also makes the substitution fail frequently as the neck or pelvis joint cannot be accurately estimated. We use a small threshold of 4° to also account for slow head turns. However, very subtle head turns are still missed by our detection algorithm. Using a smaller threshold makes the detection noisy due to jitter in the pose data. In the future when 3D estimates are temporally more consistent, the threshold may be lowered.

*Looking Up/Down*

We calculate the angle between the nose-neck and neck-pelvis segments. We only require the vertical component of the angle. So, we use the left shoulder to right shoulder vector as the normal to the plane of rotation for the angle calculation. If the angle is smaller than or equal to 125°, the infant is considered to be looking down. If the angle exceeds 135°, the infant is considered to be looking up.

Limitations: Much like the head-turn, the planar normal for looking up/down is a good substitution when the infant is looking straight but fails when the head is turned to one side. The gift bag often occludes the shoulder joints, causing inaccurate estimates. An alternative approach is to detect the angle between the head-neck limb and the neck-pelvis limb. However, this is not very different from our current approach and would only use different angle thresholds to differentiate between looking up and looking down.

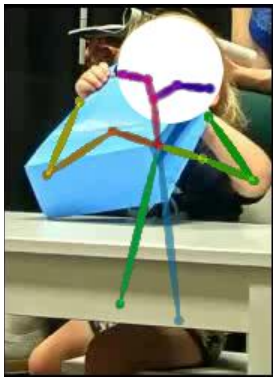
### 3.6.4.2. Quantitative Results

All results are calculated after annotating the videos across all views using BORIS and extracting the annotation labels as ground truth.

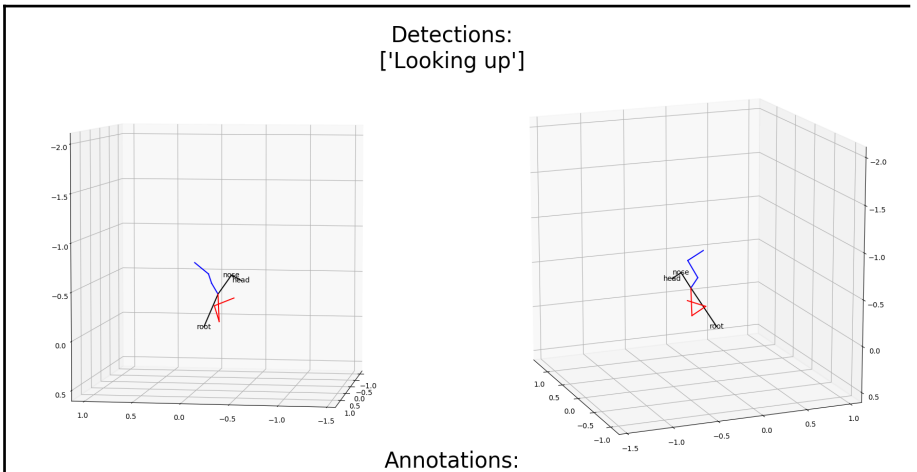
| Action Type | Without temporal consistency | | | With temporal consistency | | |
|---|---|---|---|---|---|---|
| | P | R | A | P | R | A |
| Hand Extension | 17.93% | 74.18% | 39.06% | 18.27% | 73.33% | 39.01% |
| Head Turn | 11.85% | 10.39% | 92.48% | 22.06% | 9.13% | 94.54% |
| Looking Up | 6.48% | 58.89% | 55.71% | 6.33% | 58.83% | 55.14% |
| Looking Down | 33.02% | 41.17% | 54.19% | 32.97% | 42.00% | 53.96% |
| Touching Face | 0.00% | 0.00% | 97.11% | 0.00% | 0.00% | 97.45% |

**Table 7:** Detection results for the action classes on view-2
P = Precision; R = Recall; A = Accuracy

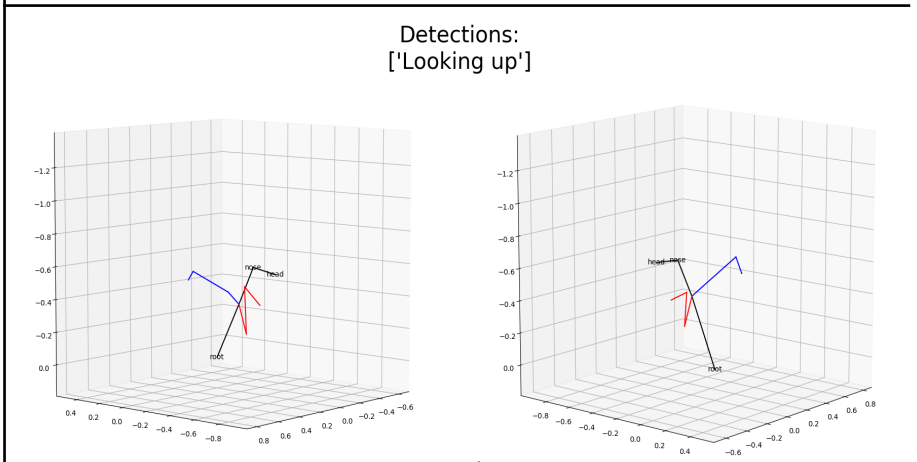
3.6.4.3. Qualitative Results



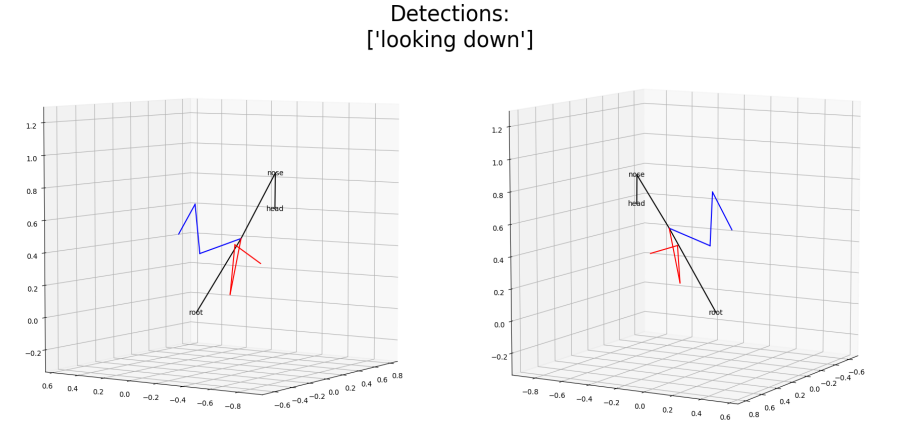
**(a)**




**(b)**

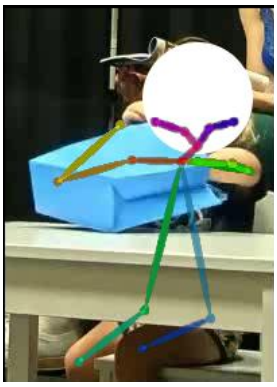



**(c)**

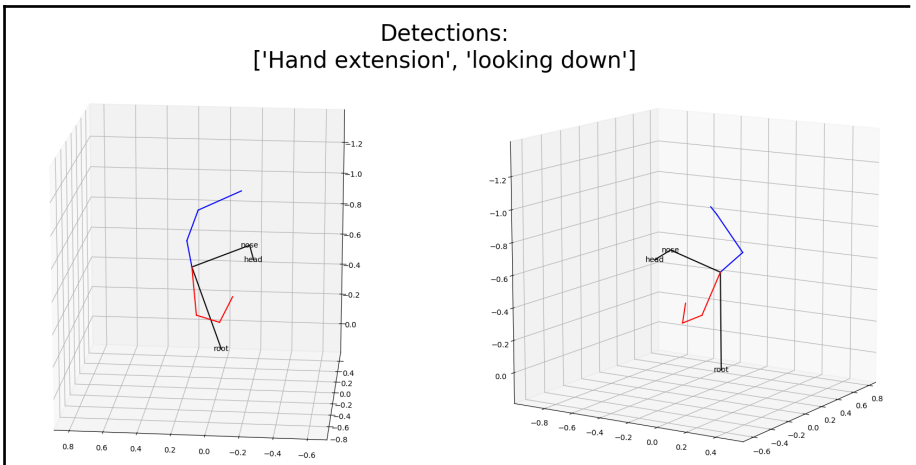



**(d)**

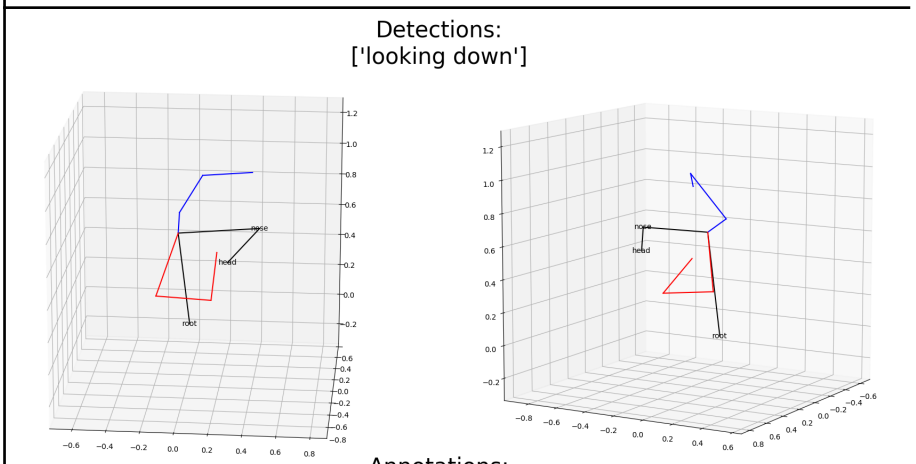
**Figure 40:** Positive effects of data cleanup. (a) Input 2D Pose; (b) Raw 3D estimates; (c) Only interpolation; (d) Both interpolation and normalization w.r.t. position, orientation and limb lengths.



**(a)**

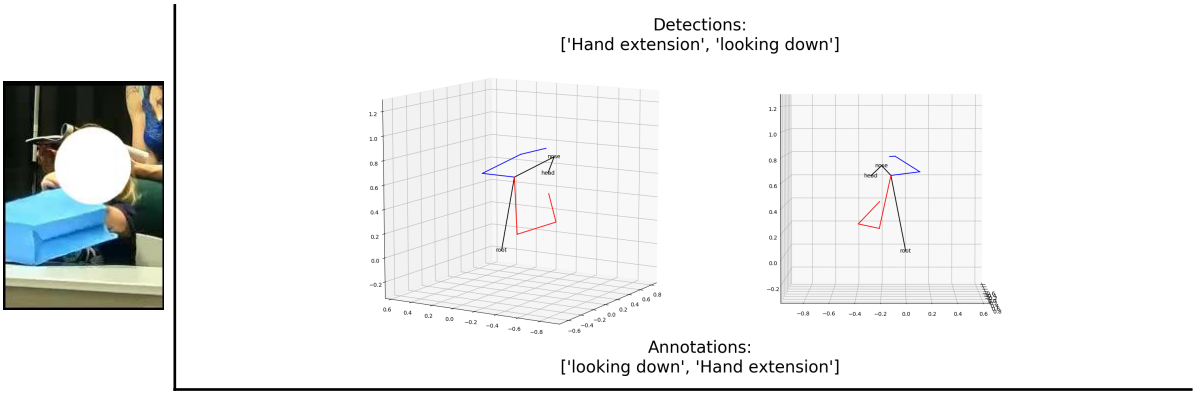



**(b)**

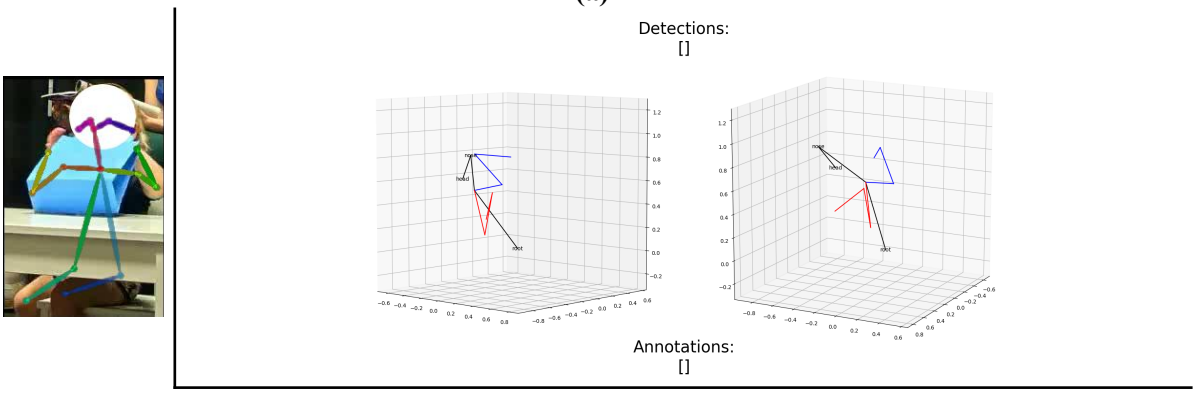



**(c)**

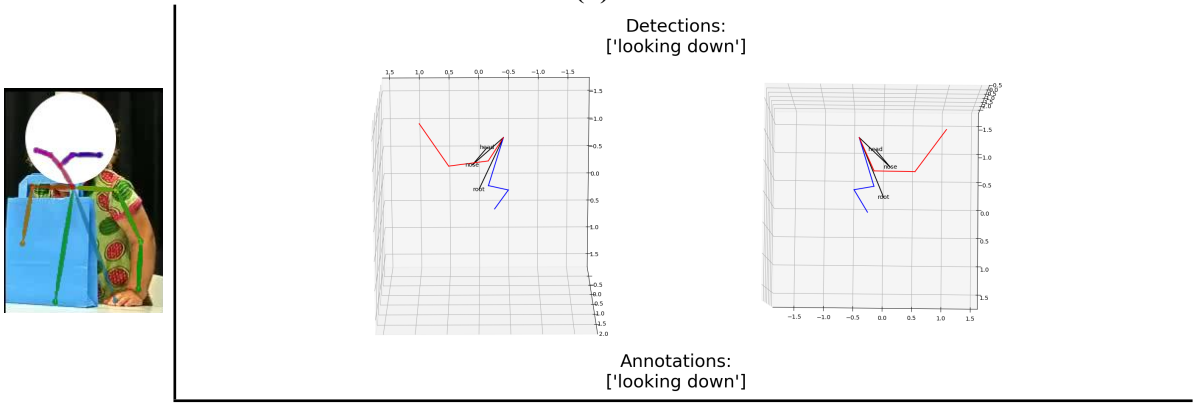
**Figure 41:** Negative effects of data cleanup. (a) Input 2D Pose; (b) Only interpolation; (c) Both interpolation and normalization w.r.t. position, orientation and limb lengths.



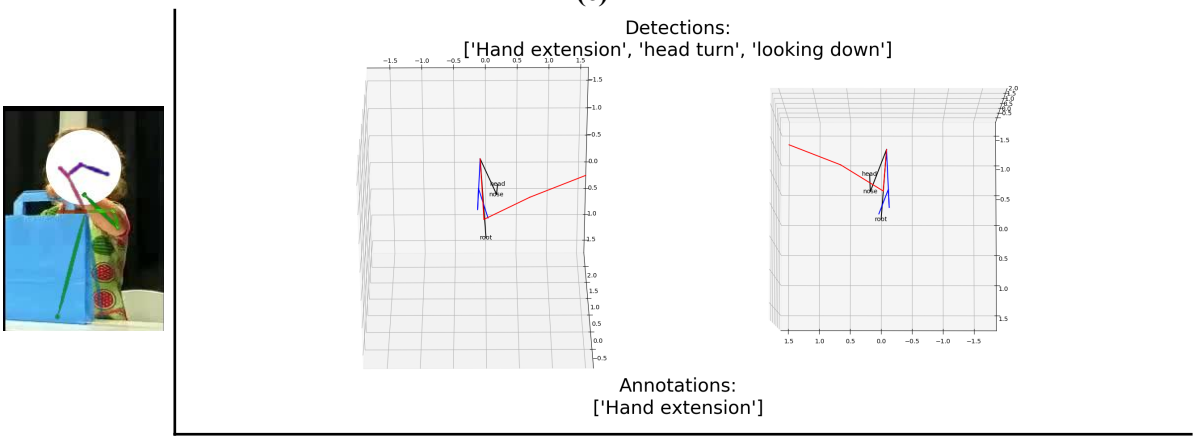

**(a)**




**(b)**




**(c)**




**(d)**

**Figure 42:** Action detection: Positive cases. (left) Input 2D pose; (right) output 3D pose. In these figures, rule-based detection results are mostly consistent with the annotations.

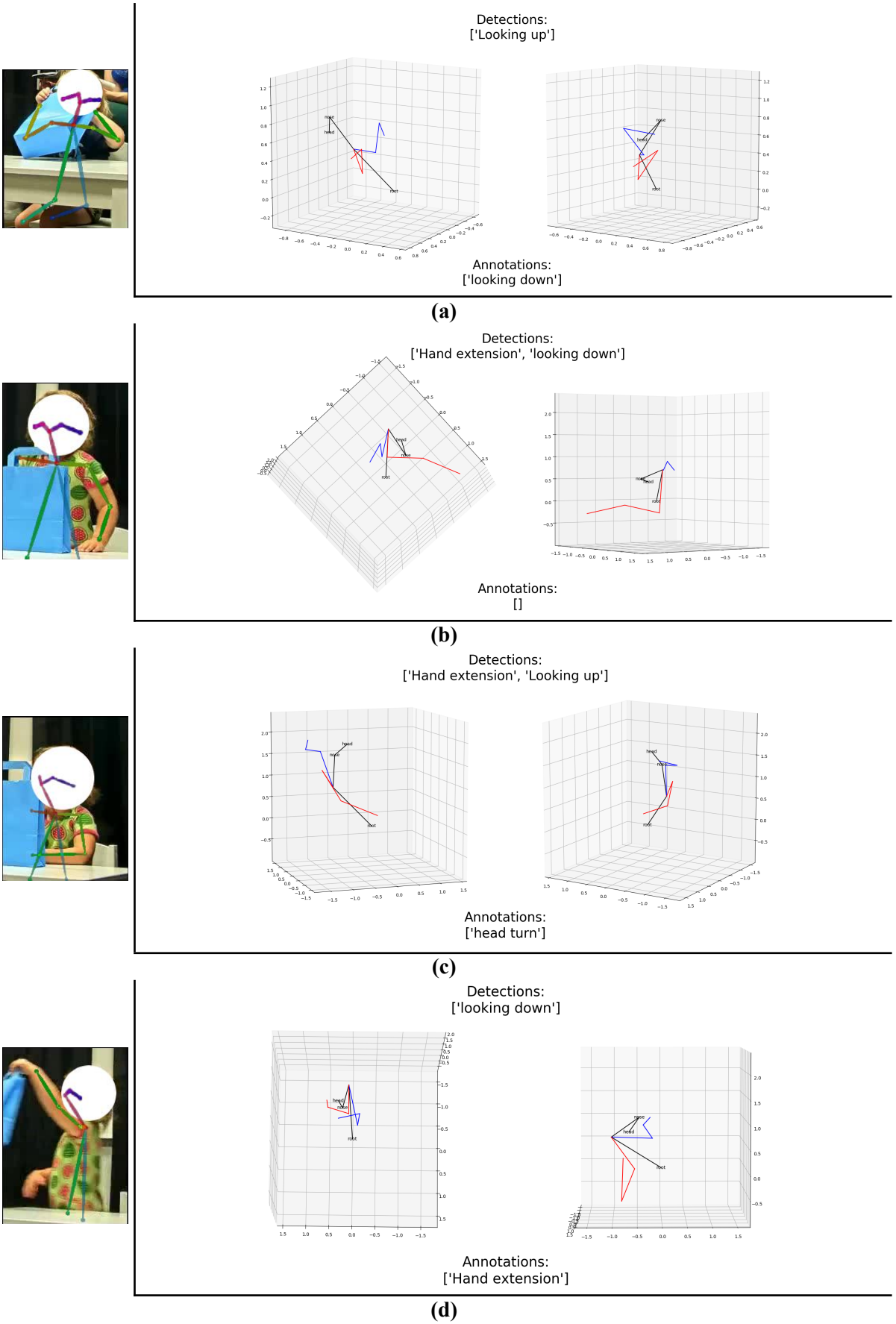



**Figure 43:** Action detection: Negative cases. (left) Input 2D pose; (right) output 3D pose. In these figures, rule-based detection results are inconsistent with the annotations.

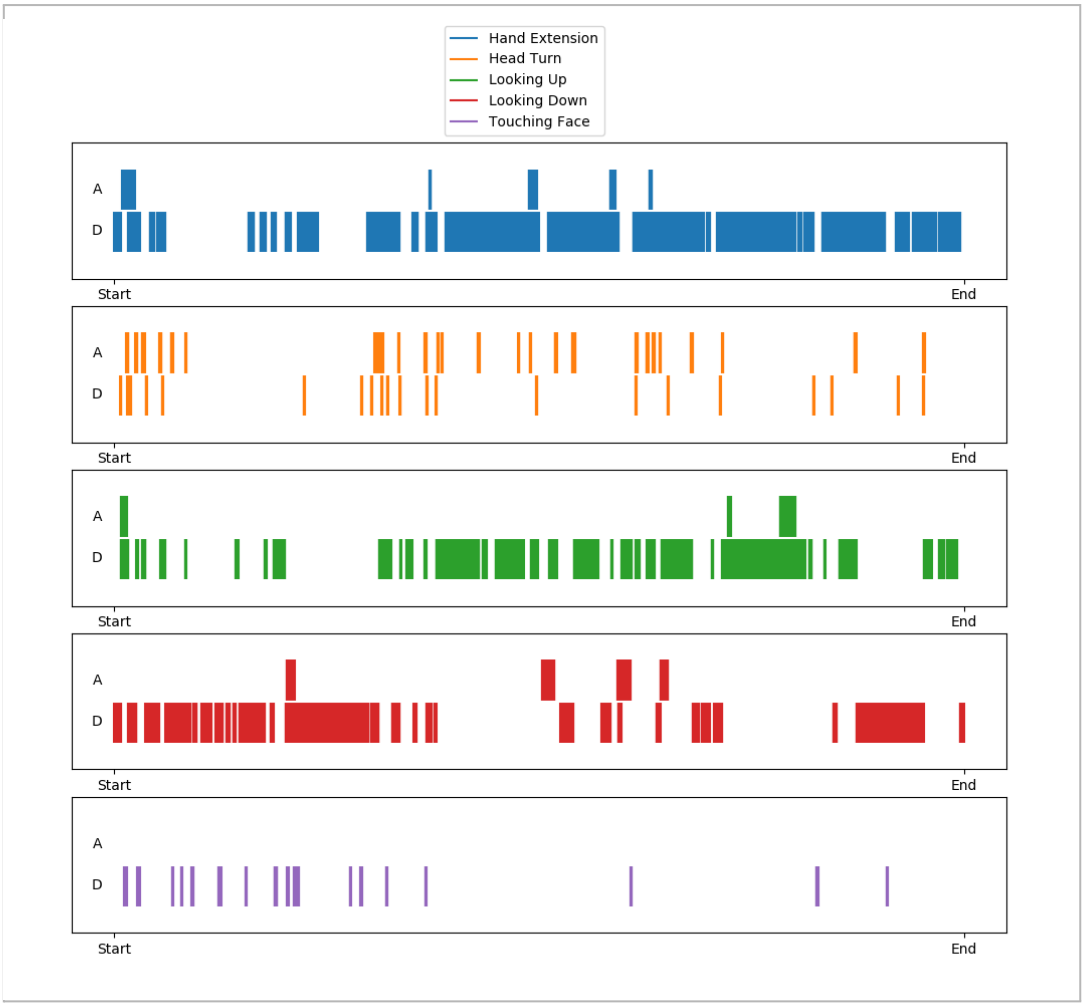



**Figure 44:** Action detection results on session B30686.
A = annotated labels; D = detected labels

### 3.6.4.4. Discussion

Checking for temporal consistency shows very little improvement in the results. In our experiments, the used view has a gift bag placed in front of the child which occludes the infant at least partly in a majority of frames. This makes it hard to obtain accurate 2D and consequently accurate 3D estimates. For detecting atomic actions performed by the infant, view-1 is a more ideal choice. It offers a less occluded view of the infant although it completely cuts off the parent in most sessions. To test if occlusion is the biggest factor for the low detection scores, we test on view-1 on a subset (8 out of 35) of the videos. These results are reported in Table 8. On comparing Tables 7 and 8, we see a significant increase in precision for the *head turn* action class and a significant increase in recall for the *hand extension*, *head turn* and *looking down* action classes. However, accuracy only increases for the *looking down* action class and remains roughly the same for the *head turn* and *touching face* action classes while getting worse for the *hand extension* and *looking up* action classes. The different viewpoint thus only benefits detection of the *head turn* action class. This is due to the fact that view-1 offers a less occluded view of the infant's head, allowing more accurate keypoint detections in that area, unlike view-2 (Figure 43(c)). For the *looking up/down* action classes we initially thought it may be due to the thresholds used to differentiate between the two. However, while experimenting with

view-2, on reducing the angle difference between *looking down* and *looking up*, we noticed a clear increase in recall for both while the precision and accuracy both consistently decreased. Thus, we used the chosen thresholds as a good balance between the three metrics (Tables 11 and 12). The *touching face* action class is inherently difficult to estimate without additional face or hand keypoints as mentioned earlier, leaving little room for improvement. Lastly, the *hand extension* action class is plagued with true negatives and false positives. We think the reason for this is 1) partly due to the used detection algorithm, where a hand dangling straight downwards is also considered as extended (Figure 43(b)), 2) partly due to VideoPose (Figure 43(d)), and 3) partly due to the normalization technique used (Figure 41).

| **Action Type** | **Without temporal consistency** | | | **With temporal consistency** | | |
|---|---|---|---|---|---|---|
| | P | R | A | P | R | A |
| Hand Extension | 17.65% | 86.84% | 33.38% | 17.63% | 88.45% | 32.94% |
| Head Turn | 32.18% | 23.02% | 92.67% | 40.49% | 17.40% | 93.50% |
| Looking Up | 6.94% | 43.02% | 45.05% | 6.85% | 42.92% | 44.39% |
| Looking Down | 32.17% | 49.11% | 65.49% | 32.29% | 50.33% | 65.19% |
| Touching Face | 0.00% | 0.00% | 98.47% | 0.00% | 0.00% | 98.48% |

**Table 8:** Detection results for the action classes on view-1
P = Precision; R = Recall; A = Accuracy

| **Hand Extension threshold** | **Without temporal consistency** | | | **With temporal consistency** | | |
|---|---|---|---|---|---|---|
| | P | R | A | P | R | A |
| > 90° | 17.40% | 80.03% | 33.78% | 17.57% | 80.35% | 33.67% |
| > 95° | 17.58% | 77.40% | 35.50% | 17.62% | 78.02% | 35.42% |
| > 100° | 17.99% | 76.30% | 37.14% | 18.29% | 76.02% | 36.96% |
| **> 105°** | **17.95%** | **74.39%** | **38.73%** | **18.24%** | **73.36%** | **38.71%** |
| > 110° | 18.13% | 72.53% | 40.03% | 18.31% | 72.51% | 39.85% |
| > 115° | 18.16% | 70.95% | 40.83% | 18.13% | 71.03% | 40.47% |
| > 120° | 18.26% | 69.98% | 42.08% | 18.22% | 70.37% | 41.66% |

**Table 9:** Results for different thresholds on hand extension angle. **bold** = chosen

| Head Turn threshold | Without temporal consistency | | | With temporal consistency | | |
|---|---|---|---|---|---|---|
| | P | R | A | P | R | A |
| > 0° | 3.84% | 85.42% | 7.21% | 3.86% | 87.41% | 4.65% |
| ≥ 1° | 7.73% | 25.91% | 79.77% | 8.33% | 22.02% | 83.75% |
| ≥ 1.5° | 8.83% | 18.78% | 86.06% | 11.24% | 15.46% | 89.94% |
| ≥ 2° | 9.78% | 14.96% | 89.31% | 14.43% | 12.24% | 92.06% |
| ≥ 2.5° | 10.96% | 12.63% | 91.19% | 18.41% | 10.70% | 93.32% |
| **≥ 3°** | **12.14%** | **11.06%** | **92.56%** | **22.15%** | **9.31%** | **94.58%** |
| ≥ 3.5° | 12.86% | 9.12% | 93.44% | 21.28% | 6.42% | 94.92% |

**Table 10:** Results for different thresholds on head turn angle. **bold** = chosen

| Looking Up threshold | Without temporal consistency | | | With temporal consistency | | |
|---|---|---|---|---|---|---|
| | P | R | A | P | R | A |
| ≥ 120° | 5.92% | 71.49% | 35.99% | 5.87% | 71.61% | 35.26% |
| ≥ 125° | 6.19% | 69.75% | 40.08% | 6.13% | 69.88% | 39.23% |
| ≥ 130° | 6.80% | 66.37% | 48.31% | 6.75% | 66.95% | 47.35% |
| **≥ 135°** | **6.51%** | **58.92%** | **55.73%** | **6.36%** | **58.85%** | **55.17%** |
| ≥ 140° | 6.92% | 57.36% | 61.37% | 6.85% | 57.64% | 60.85% |
| ≥ 145° | 7.25% | 52.18% | 70.35% | 7.18% | 52.77% | 69.82% |
| ≥ 150° | 8.06% | 46.58% | 76.09% | 7.91% | 46.80% | 75.70% |

**Table 11:** Results for different thresholds on looking up angle. **bold** = chosen

| Looking Down threshold | Without temporal consistency | | | With temporal consistency | | |
|---|---|---|---|---|---|---|
| | P | R | A | P | R | A |
| ≤ 110° | 32.05% | 30.45% | 59.97% | 31.96% | 31.45% | 59.66% |
| ≤ 115° | 32.59% | 31.98% | 59.09% | 32.55% | 33.07% | 58.57% |
| ≤ 120° | 32.82% | 34.29% | 57.56% | 32.60% | 35.30% | 56.97% |
| **≤ 125°** | **33.54%** | **39.15%** | **55.58%** | **33.44%** | **40.77%** | **55.11%** |
| ≤ 130° | 32.07% | 45.53% | 50.32% | 31.75% | 47.10% | 49.67% |
| ≤ 135° | 32.03% | 53.84% | 47.27% | 31.69% | 55.54% | 46.54% |
| ≤ 140° | 33.36% | 61.63% | 46.11% | 33.24% | 63.40% | 45.42% |

**Table 12:** Results for different thresholds on looking down angle. **bold** = chosen

## 4.1. Conclusion

We develop some work to allow automatic analysis of infant-parent interaction videos using 3D pose data without any dependence on extraneous information such as ground truth or camera parameters. Our work begins by ascertaining the efficacy of current 3D pose estimation techniques that do not rely on extraneous information. For this, we perform a preliminary evaluation on two existing 3DHPE techniques, VIBE and VideoPose, on a private, unmarked dataset called YOUth. A qualitative comparison illustrates that VideoPose works better for such in-the-wild videos. Next, we compare VideoPose to two state-of-the-art multi-view 3DHPE techniques called CrossView and LearnTri to obtain a performance evaluation for VideoPose. This comparison uses the Human3.6M dataset that is fully-annotated. VideoPose performs modestly even though LearnTri and CrossView outperform VideoPose due to their ability to draw inferences from multiple views while exploiting camera intrinsics. With a well-informed performance baseline for VideoPose, we finally use it to process parent-child interaction videos from YOUth and obtain 3D pose data.

Our used dataset contains multiple human instances in a multi-view setting with frequent camera panning and zooming. VideoPose however only supports monocular single-person 3DHPE. To allow for this, we run object detection using MaskRCNN on each view and match bounding boxes of instances across frames using a custom-built color-matching module. Matched instances are split into separate video streams to give us our prerequisite single-person views. Next, we use an off-the-shelf 2D pose estimator called OpenPose to provide 2D keypoint annotations for each video stream and use VideoPose to lift these keypoints to 3D coordinate space. The final task is to analyze the pose data for interactions and classify them as such, i.e., action detection.

We focus on detecting atomic actions performed by the child only. For this, we first cleanup the output 3D data using interpolation and normalization w.r.t position, orientation and limb lengths. This improves the stability of the 3D data. Next, we use a rule-based approach to classify the extracted 3D pose data into actions. Our results portray that rule-based approaches are not ideal for action classification if the 3D data is unstable.

## 4.2. Limitations and possible solutions

There is a lot of room for improvement, both for existing modules as well as our work. Starting with the instance segmentation, we observed that Mask R-CNN does not work well for occluded instances. It either produces fragmented masks or fails to detect instances as distinct when they are in close proximity. This is more apparent in parent-infant scenarios. We speculate that this is because Mask R-CNN is not trained to detect instances of varying scales at close-proximity. We also observed frequent false positives in the background during person detection using Mask R-CNN, even when the background is black. Both these problems combined created a lot of false labels for our color-matching module. Merging masks with high overlap helped curtail this to a good extent. However, when instances are occluded from the middle, both sides of the same instance are almost always saved as separate labels. For color-matching the first suggested improvement is to make sure all camera views use the same color palette. In our test case, i.e. Gift-Delay, we can clearly see that the colors for the cameras differ, particularly cameras 1 and 4 differ significantly from cameras 2 and 3. This makes color-matching across views challenging. A good workaround would be to use color correction matrices to match camera colors, but that either requires manually designing the correction matrix or obtaining the different color palettes for each camera to match them accordingly. We would also like to adapt the module to generate extra splits such that a single instance may have multiple labels. In this case, if fragments of the same instance are assigned separate labels, we would still have

at least one split containing all occurrences of a particular label. An alternative would be to use a counting mechanism and merge newer labels with the old one using a distance measure. However, this can cause problems while detecting new instances mid-stream. Additionally we would like to handle ghost instances better where a label that suddenly disappears or leaves the field of vision is still considered as a future candidate label.

For 2DHPE, OpenPose works satisfactorily, but is observed to be inept at handling occlusions. For such scenarios, it would be beneficial to use a 2DHPE technique that can fill holes or use occlusion-aware networks to provide better 2D pose estimates. Another problem with OpenPose is that it frequently jumps instances such that we get 2D pose estimates for the instance that is only partially visible in the camera view rather than the main subject. For this, 2DHPE techniques that use tracking would be a better choice as we can then choose the 2D pose estimates we want to retain. We would also like to explore how top-down 2DHPE techniques compare with OpenPose. The 3DHPE technique we use, VideoPose, only requires bounding boxes and 2D pose estimates as input to lift them to 3D coordinate space and has no dependence on extraneous information such as camera parameters. This makes it very suitable for real-world applications when coupled with a good object detector and 2DHPE model as camera information is not readily available in such scenarios. However, as observed, when keypoints are missing VideoPose becomes unreliable. The temporal nature of the model can make 3D estimates in certain frames diverge greatly from the corresponding 2D estimates due to inaccurate 2D estimates in the neighbouring frames. Adapting VideoPose to use a weighted contribution of 2D estimates from frames such that estimates with higher confidence scores gain more focus could help reduce this problem. VideoPose could also benefit from prior anatomical information to prevent improbable joint orientations. Lastly, we would like to experiment with techniques that are equally free of dependence on extraneous information but also produce confidence scores. This would enable merging information from multiple views, consequently producing more robust 3D pose estimates.

For action detection, we explore the simplest and most easily-deployable technique available, i.e., rule-based classification. However, rules have their fallacies. Classification is highly dependent on the thresholds used to detect action classes and can require extensive fine-tuning to obtain satisfactory results. There is little room for improvement using rule-based classifications as altering the rules can entirely change the action class being detected (looking up vs looking down). Additionally, knowledge of vector arithmetic (and sometimes human anatomy) is required to design rules, especially for complicated actions/interactions. This may not always be an asset available to the researcher. Further. rule-based techniques have no mechanism to learn from their mistakes to produce better outputs in future cases. Considering the challenges we faced to detect simple atomic actions, it would be appropriate to consider detecting and classifying parent-child interactions as an even harder ordeal. Inaccurate 3D pose data coupled with the absence of 3D poses for objects make it hard to gather contextual cues or draw inferences about interactions. Interpolation and normalization are powerful techniques to make 3D pose estimates more robust but, as seen, such techniques are fruitless if the quality of 3D estimates is severely poor. For future work, we would like to experiment with deep learning techniques for action detection and classification.

# References


1. Andriluka, M., Pishchulin, L., Gehler, P., & Schiele, B. (2014). 2d human pose estimation: New benchmark and state of the art analysis. In CVPR, 2014
2. Arbeláez, P., Maire, M., Fowlkes, C., & Malik, J. (2011). Contour Detection and Hierarchical Image Segmentation. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, May 2011, vol. 33, no. 5, pp. 898-916, doi: 10.1109/TPAMI.2010.161.
3. Arnab, A., Doersch, C., Zisserman, A. (2019). Exploiting temporal context for 3d human pose estimation in the wild. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 3395–3404
4. Bates, E., Camaioni, L., & Volterra, V. (1975). The Acquisition of Performatives Prior to Speech. In: Merrill-Palmer Quarterly, vol 21, pp. 205-226.
5. Bishop, C. M. (1994). Mixture density networks. Aston University.
6. Boundy, L. , Cameron‐Faulkner, T. & Theakston, A. (2019). Intention or attention before pointing: Do infants' early holdout gestures reflect evidence of a declarative motive?. In: Infancy, 24, 228-248.
7. Bridgeman, L., Volino, M., Guillemaut, J., & Hilton, A. (2019). Multi-Person 3D Pose Estimation and Tracking in Sports. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 2487-2496.
8. Camaioni, L. (1997). The emergence of intentional communication in ontogeny, phylogeny, and pathology. In: European Psychologist, 2, 216-225.
9. Cameron‐Faulkner, T., Theakston, A., Lieven, E., & Tomasello, M. (2015).. In: Infancy, 20, 576– 586.
10. Cao, Z., Martinez, G. H., Simon, T., Wei, S-E., & Sheikh, Y. (2019). OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, doi: 10.1109/TPAMI.2019.2929257..
11. Cao, Z., Simon, T., Wei, S-E., & Sheikh, Y. (2017). Realtime Multi-person 2D Pose Estimation Using Part Affinity Fields. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, 2017, pp. 1302-1310, doi: 10.1109/CVPR.2017.143..
12. Carpenter, M., Nagell, K., Tomasello, M., Butterworth, G., & Moore, C. (1998). Social cognition, joint attention, and communicative competence from 9 to 15 months of age. In: Monographs of the Society for Research in Child Development, 63, 162– 174.
13. Charles, J., Pfister, T., Everingham, M., & Zisserman, A. (2013). Automatic and Efficient Human Pose Estimation for Sign Language Videos. In: International Journal of Computer Vision. 110. 70-90. 10.1007/s11263-013-0672-6.
14. Chen, C.H., & Ramanan, D. (2017). 3d human pose estimation = 2d pose estimation + matching. In: Proc. IEEE Conference on Computer Vision and Pattern Recognition, pp. 7035–7043.
15. Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2015). Semantic image segmentation with deep convolutional nets and fully connected crfs. In: ICLR, 2015
16. Chen, L., Ai, H., Chen, R., Zhuang, Z., & Liu, S. (2020). Cross-View Tracking for Multi-Human 3D Pose Estimation at Over 100 FPS. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3276-3285.

17. Chen, X., & Yuille, A.L. (2014). Articulated pose estimation by a graphical model with image dependent pairwise relations. In: Advances in neural information processing systems, pp. 1736–1744.
18. Chen, Y., Shen, C.,Wei, X.S., Liu, L., & Yang, J. (2017). Adversarial PoseNet: A Structure-Aware Convolutional Network for Human Pose Estimation. In: ICCV, 2017, pp. 1221-1230. 10.1109/ICCV.2017.137.
19. Cheng, Y., Yang, B., Wang, B., & Tan, R.T. (2020). 3D Human Pose Estimation using Spatio-Temporal Networks with Explicit Occlusion Training
20. Cheng, Y., Yang, B., Wang, B., Yan, W., & Tan, R. T. (2019). Occlusion-Aware Networks for 3D Human Pose Estimation in Video. In: The IEEE International Conference on Computer Vision (ICCV), 2019, pp. 723-732
21. Cho, K., Merrienboer, B.V., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In: ArXiv, abs/1406.1078.
22. Cortes, C., & Vapnik, V. (1995). Support-vector networks. In: Machine Learning, 20, pp. 273-297.
23. D. B. West et al. (2001). Introduction to graph theory, vol 2. Prentice hall Upper Saddle River.
24. Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. In: CVPR, 2005
25. Dang, Q., Yin, J., Wang B., & Zheng, W. (2019) Deep learning based 2D human pose estimation: A survey. In: Tsinghua Science and Technology, vol. 24, no. 6, pp. 663-676, Dec. 2019, doi: 10.26599/TST.2018.9010100.
26. Dantone, M., Gall, J., Leistner, C., & Van Gool, L. (2013). Human Pose Estimation Using Body Parts Dependent Joint Regressors. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Portland, OR, 2013, pp. 3041-3048.
27. Facebookresearch, VideoPose3D (2019). Github repository, https://github.com/facebookresearch/VideoPose3D
28. Felzenszwalb, P. F., Girshick, R. B., McAllester, D., & Ramanan, D. (2010). Object detection with discriminatively trained part-based models. In: IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2010.
29. Felzenszwalb, P., & Huttenlocher, D. (2005). Pictorial Structures for Object Recognition. In: International Journal of Computer Vision, vol 61, pp. 55-79, doi: 10.1023/B:VISI.0000042934.15159.49.
30. Franco, F., & Butterworth, G. (1996). Pointing and social awareness: Declaring and requesting in the second year. In: Journal of Child Language, 23(2), 307–336.
31. Friard, O., & Gamba, M. (2016). BORIS: A free, versatile open-source event-logging software for video/audio coding and live observations. In: Methods in Ecology and Evolution, p. 7, doi: 10.1111/2041-210X.12584.
32. Giorgis, N., Puppo, E., Alborno, P., & Camurri, A. (2019). Evaluating Movement Quality Through Intrapersonal Synchronization. In: IEEE Transactions on Human-Machine Systems. PP. 1-10. 10.1109/THMS.2019.2912498.
33. Girshick, R. (2015). Fast R-CNN. In: IEEE International Conference on Computer Vision (ICCV), Santiago, 2015, pp. 1440-1448, doi: 10.1109/ICCV.2015.169.
34. Girshick, R., Radosavovic, I., Gkioxari, G., Doll, P., & He, K. (2018). Detectron. In: Facebook AI Research

35. Gkioxari, G., Hariharan, B., Girshick, R., & Malik, J. (2014). Using k-poselets for detecting people and localizing their keypoints. In: IEEE Conference on Computer Vision and Pattern Recognition (2014)
36. Gong, W., Zhang, X., González, J., Sobral, A., Bouwmans, T., Tu, C., & Zahzah, E. (2016). Human Pose Estimation from Monocular Images: A Comprehensive Survey. In: Sensors (Basel, Switzerland), 16.
37. Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A.C., & Bengio, Y. (2014). Generative Adversarial Nets. In: NIPS.
38. Guanghan, N., & He, Z. (2017). Dual Path Networks for Multi-Person Human Pose Estimation. In: ArXiv abs/1710.10192 (2017): n. Pag.
39. Hariharan, B., Arbeláez, P., Girshick, R., & Malik J. (2014). Simultaneous detection and segmentation. In: European Conference on Computer Vision, 2014. Springer, pp. 297-312
40. He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask R-CNN. In: IEEE International Conference on Computer Vision (ICCV), Venice, 2017, pp. 2980-2988, doi: 10.1109/ICCV.2017.322.
41. He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. In: arXiv preprint arXiv:1512.03385,2015.
42. Hinton, G., Srivastava, N., & Swersky, K. (2012). Neural networks for machine learning lecture 6a overview of mini-batch gradient descent.
43. Ho, T.K. (1995). Random decision forests. In: Proceedings of 3rd International Conference on Document Analysis and Recognition, Montreal, QC, Canada, 1995, pp. 278-282 vol.1, doi: 10.1109/ICDAR.1995.598994.
44. Hochreiter, S. (1998). The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. In: International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, vol 6, pp. 107-116, doi: 10.1142/S0218488598000094.
45. Huang, G., Liu, Z., Maaten, L. v. d., & Weinberger, K. Q. (2017). Densely Connected Convolutional Networks. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, 2017, pp. 2261-2269, doi: 10.1109/CVPR.2017.243.
46. Ingram, D. (1974). Stages in the development of one-word sentences. In: Stanford Child Language Forum, Stanford, April 1974.
47. Insafutdinov, E., Pishchulin, L, Andres, B., Andriluka, M., & Schiele, B. (2016). Deepercut: A deeper, stronger, and faster multi person pose estimation model. In: ECCV, 2016
48. Ionescu, C., Papava, D., Olaru, V., & Sminchisescu, C. (2013). Human3.6M: Large Scale Datasets and Predictive Methods for 3D Human Sensing in Natural Environments. In: IEEE transactions on pattern analysis and machine intelligence. 36. 10.1109/TPAMI.2013.248.
49. Iskakov, K., Burkov, E., Lempitsky, V.S., & Malkov, Y. (2019). Learnable Triangulation of Human Pose. In: IEEE/CVF International Conference on Computer Vision (ICCV), 7717-7726.
50. Jin, S., Ma, X., Han, Z., Wu, Y., Yang, W., Liu, W., Qian, C., & Ouyang, W. (2017). Towards Multi-Person Pose Tracking : Bottom-up and Top-down Methods.
51. Johnson, S., & M. Everingham, M. (2010). Clustered pose and nonlinear appearance models for human pose estimation. In: BMVC'10
52. Kalchbrenner, N., Espeholt, L., Simonyan, K., Oord, A., Graves, A., & Kavukcuoglu, K. (2016). Neural Machine Translation in Linear Time. In: ArXiv, abs/1610.10099.

53. Kanazawa, A., Black, M.J., Jacobs, D.W., & Malik, J. (2018). End-to-end recovery of human shape and pose. In: Proc. IEEE Conference on Computer Vision and Pattern Recognition, pp. 7122–7131.
54. Kingma, D.P., & Adam, J.L.B. (2014). A method for stochastic optimization. 2014. In: arXiv:1412.6980v9
55. Kocabas, M., Athanasiou, N., & Black, M. (2019). VIBE: Video Inference for Human Body Pose and Shape Estimation.
56. Kocabas, M., Karagoz, S., & Akbas, E. (2018). Multiposenet: Fast multi-person pose estimation using pose residual network. In: Proc. European Conference on Computer Vision. Springer, pp. 437–453.
57. Kreiss, S., Bertoni, L., & Alahi, A. (2019). PifPaf: Composite Fields for Human Pose Estimation. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 11977-11986
58. Krizhevsky, A., Sutskever, I., & Hinton, G. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In: Neural Information Processing Systems, pp. 25, doi: 10.1145/3065386.
59. Kudo, Y., Ogaki, K., Matsui, Y., & Odagiri, Y. (2018). Unsupervised Adversarial Learning of 3D Human Pose from 2D Joint Locations.
60. Kuo, P. & Makris, D. & Nebel, J.-C. (2011). Integration of Bottom-Up/Top-Down Approaches for 2D Pose Estimation Using Probabilistic Gaussian Modelling. In: Computer Vision and Image Understanding. 115. 242-255. 10.1016/j.cviu.2010.09.001
61. Li, C., & Lee, G.H. (2019). Generating multiple hypotheses for 3d human pose estimation with mixture density network. In: Proc. IEEE Conference on Computer Vision and Pattern Recognition, pp. 9887–9895
62. Li, H. (2019). Vector Hourglass Network for Human Pose Estimation based on Deep Learning. In: IEEE 2nd International Conference on Automation, Electronics and Electrical Engineering (AUTEEE), Shenyang, China, 2019, pp. 552-558.
63. Li, J., Wang, C., Liu, W., Qian, C., & Lu, C. (2020). HMOR: Hierarchical Multi-Person Ordinal Relations for Monocular Multi-Person 3D Pose Estimation. In: ECCV, 2020.
64. Li, S., & Chan, A.B. (2014). 3d human pose estimation from monocular images with deep convolutional neural network. In: Proc. Asian Conference on Computer Vision, Springer. pp. 332–347.
65. Li, Z., Heyden, A., & Oskarsson, M. (2019). Parametric Model-Based 3D Human Shape and Pose Estimation from Multiple Views. In: Felsberg M., Forssén PE., Sintorn IM., Unger J. (eds) Image Analysis. SCIA 2019. Lecture Notes in Computer Science, vol 11482. Springer, Cham
66. Lin, T-Y., Dollar, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). Feature Pyramid Networks for Object Detection. In: CVPR, 2017, pp. 936-944, doi: 10.1109/CVPR.2017.106.
67. Liszkowski, U., Albrecht, K., Carpenter, M., & Tomasello, M. (2008). Infants' visual and auditory communication when a partner is or is not visually attending. In: Infant Behavior and Development, 31, 157–167.
68. Loper, M., Mahmood, N., Romero, J., Pons-Moll, G., & Black, M.J. (2015). SMPL: a skinned multi-person linear model. In: ACM Trans. Graph. 34, 6, Article 248 (November 2015), 16 pages. DOI:https://doi.org/10.1145/2816795.2818013

69. Ludl, D., Gulde, T., Thalji, S., &Curio, C. (2018). Using simulation to improve human pose estimation for corner cases. In: 21st International Conference on Intelligent Transportation Systems (ITSC), pp. 3575–3582, Nov 2018.
70. Luvizon, D.C., Picard, D., & Tabia, H., (2018). 2d/3d pose estimation and action recognition using multitask deep learning. In: Proc. IEEE Conference on Computer Vision and Pattern Recognition, pp. 5137–5146
71. Luvizon, D.C., Tabia, H., & Picard, D. (2017). Human pose regression by combining indirect part detection and contextual information. In: arXiv preprint, arXiv:1710.02322 .
72. Mahmood, N., Ghorbani, N., Troje, N.F., Pons-Moll, G., & Black, M.J. (2019). Amass: Archive of motion capture as surface shapes. In: International Confer-ence on Computer Vision, 2019
73. Marcard, T.V., Henschel, R., Black, M.J., Rosenhahn, B., & Pons-Moll, G. (2018). Recovering Accurate 3D Human Pose in the Wild Using IMUs and a Moving Camera. In: ECCV.
74. Martinez, J., Hossain, R., Romero, J., & Little, J.J. (2017). A simple yet effective baseline for 3d human pose estimation. In: Proc. IEEE International Conference on Computer Vision, pp. 2640–2649
75. Mehta, D., Rhodin, H., Casas, D., Fua, P., Sotnychenko, O., Xu, W., & Theobalt, C. (2017). Monocular 3D Human Pose Estimation in the Wild Using Improved CNN Supervision. In: 506-516. 10.1109/3DV.2017.00064.
76. Mehta, D., Sotnychenko, O., Mueller, F., Xu, W., Elgharib, M., Fua, P., Seidel, H., Rhodin, H., Pons-Moll, G., & Theobalt, C. (2020). XNect: Real-time Multi-Person 3D Motion Capture with a Single RGB Camera. In: SIGGRAPH 2020.
77. Mehta, D., Sotnychenko, O., Mueller, F., Xu, W., Sridhar, S., Pons-Moll, G., Theobalt, C. (2018). Single-shot multi-person 3d body pose estimation from monocular rgb input. In: International Conference on 3D Vision, pp. 120-130.
78. Moon, G., Chang, J. Y., & Lee, K. M. (2018). V2v-posenet: Voxel-to-voxel prediction network for accurate 3d hand and human pose estimation from a single depth map. In: CVPR, 2018.
79. Newell, A., Yang, K., & Deng, J. (2016). Stacked Hourglass Networks for Human Pose Estimation. In: Leibe B., Matas J., Sebe N., Welling M. (eds) Computer Vision – ECCV 2016. ECCV 2016. Lecture Notes in Computer Science, vol 9912. Springer, Cham.
80. Nibali, A., He, Z., Morgan, S., & Prendergast, L. (2018). Numerical coordinate regression with convolutional neural networks. In: arXiv preprint, arXiv:1801.07372 .
81. Nie, X., Feng, J., Xing, J., & Yan, S., (2018). Pose partition networks for multi-person pose estimation. In: Proc. European Conference on Computer Vision, pp. 684–699.
82. Ning, G., Liu, P., Fan, X., & Zhang, C. (2019). A Top-down Approach to Articulated Human Pose Estimation and Tracking.
83. Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., & Kavukcuoglu, K. (2016). WaveNet: A Generative Model for Raw Audio. In: ArXiv, abs/1609.03499.
84. Papandreou, G., Zhu, T., Chen, L.-C., Gidaris, S., Tompson, J., & Murphy, K. (2018). Personlab: Person pose estimation and instance segmentation with a bottom up, part-based, geometric embedding model. In: Proc. European Conference on Computer Vision, pp. 269-286.

85. Papandreou, G., Zhu, T., Kanazawa, N., Toshev, A., Tompson, J., Bregler, C., & Murphy, K. (2017). Towards accurate multi-person pose estimation in the wild. In: CVPR, 2017, vol. 3, p. 6.
86. Pavlakos, G., Zhou, X., Derpanis, K., & Daniilidis, K. (2017b). Harvesting Multiple Views for Marker-Less 3D Human Pose Annotations. In: 1253-1262. 10.1109/CVPR.2017.138.
87. Pavlakos, G., Zhou, X., Derpanis, K.G., & Daniilidis, K. (2017a). Coarse-to-fine volumetric prediction for single-image 3d human pose. In: Proc. IEEE Conference on Computer Vision and Pattern Recognition, pp. 1263–1272
88. Pavllo, D., Feichtenhofer, C., Grangier, D., & Auli, M. (2019). 3D Human Pose Estimation in Video With Temporal Convolutions and Semi-Supervised Training. In: 7745-7754. 10.1109/CVPR.2019.00794.
89. Pfister, T., Simonyan, K., Charles, J., & Zisserman, A. (2014). Deep convolutional neural networks for efficient pose estimation in gesture videos. In: Proc. Asian Conference on Computer Vision. Springer, pp. 538–552.
90. Pham, H.H., Salmane, H., Khoudour, L., Crouzil, A., Zegers, P., & Velastin, S. (2020). A Unified Deep Framework for Joint 3D Pose Estimation and Action Recognition from a Single RGB Camera. In: Sensors (Basel, Switzerland), vol 20.
91. Pishchulin, L., Insafutdinov, E., Tang, S., Andres, B., Andriluka, M., Gehler, P., & Schiele, B. (2016). Deepcut: Joint subset partition and labeling for multi person pose estimation. In: CVPR, 2016.
92. Poppe, R. (2007). Vision-based human motion analysis: An overview. In: Comput. Vis. Image Underst., vol 108, pp. 4-18.
93. Poppe, R., Zee, S., Heylen, D., & Taylor, P. (2014). AMAB: Automated measurement and analysis of body motion. In: Behavior Research Methods, 46, pp. 625-633.
94. Qiu, H., Wang, C., Wang, J., Wang, N., & Zeng, W. (2019). Cross View Fusion for 3D Human Pose Estimation. In: ICCV, 2019, pp. 4341-4350, doi: 10.1109/ICCV.2019.00444.
95. Ramakrishna, V., Munoz, D., Hebert, M., Bagnell, J. A., & Sheikh, Y. (2014). Pose machines: Articulated pose estimation via inference machines. In: Proc. European Conference on Computer Vision, Springer. pp. 33–47.
96. Redmon, Joseph & Farhadi, Ali. (2018). YOLOv3: An Incremental Improvement.
97. Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137-1149, 1 June 2017, doi: 10.1109/TPAMI.2016.2577031.
98. Rhodin, H., Spörri, J., Katircioglu, I., Constantin, V., Meyer, F., Müller, E., Salzmann, M., & Fua, P. (2018). Learning Monocular 3D Human Pose Estimation from Multi-view Images
99. Rogez, R., Weinzaepfel, P., & Schmid, C. (2017). Lcr-net: Localization-classification-regression for human pose. In: CVPR, 2017
100. Sakr, G.E., Mokbel, M., Darwich, A., Khneisser, M.N., & Hadi, A.W. (2016). Comparing deep learning and support vector machines for autonomous waste sorting. In: IEEE International Multidisciplinary Conference on Engineering Technology (IMCET), pp. 207-212.
101. Sapp, B., & Taskar, B. (2013). Modec: Multimodal decomposable models for human pose estimation. In: CVPR, 2013.
102. Shafaei, A., Little, J., & Schmidt, M. (2016). Play and Learn: Using Video Games to Train Computer Vision Models. In: ArXiv, abs/1608.01745.

103. Simon, T., Joo, H., Matthews, I., & Sheikh, Y. (2017). Hand keypoint detection in single images using mul-tiview bootstrapping. In: CVPR, pp. 1145–1153, 2017
104. Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. In: arXiv 1409.1556.
105. Souza, C.D., Gaidon, A., Cabon, Y., & Peña, A. (2017). Procedural Generation of Videos to Train Deep Action Recognition Networks. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2594-2604.
106. Sun, X., Shang, J., Liang, S., & Wei, Y. (2017). Compositional human pose regression. In: Proc. IEEE International Conference on Computer Vision, pp. 2602-2611.
107. Szegedy, C., et al. (2015). Going deeper with convolutions. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, 2015, pp. 1-9, doi: 10.1109/CVPR.2015.7298594.
108. Tekin, B., Katircioglu, I., Salzmann, M., Lepetit, V., & Fua, P. (2016). Structured prediction of 3D human pose with deep neural networks. In: BMVC, 2016.
109. Toshev, A., & Szegedy, C. (2014). Deeppose: Human pose estimation via deep neural networks. In: Proc. IEEE Conference on Computer Vision and Pattern Recognition, pp. 1653–1660.
110. Varol, G., Romero, J., Martin, X., Mahmood, N., Black, M.J., Laptev, I., & Schmid, C. (2017). Learning from Synthetic Humans. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4627-4635.
111. Véges, M., & Lörincz, A. (2020). Temporal Smoothing for 3D Human Pose Estimation and Localization for Occluded People. In: ICONIP, 2020.
112. Verhagen, J., Mulder, H., & Leseman, P. (2017). Effects of home language environment on inhibitory control in bilingual three-year-old children. In: Bilingualism: Language and Cognition, 20(1), pp. 114-127. doi:10.1017/S1366728915000590
113. Wei, S., Ramakrishna, V., Kanade, T., & Sheikh, Y. (2016) Convolutional Pose Machines. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 4724-4732.
114. Xiao, B., Wu, H., & Wei, Y. (2018). Simple baselines for human pose estimation and tracking. In: Proc. European Conference on Computer Vision, pp. 466–481
115. Yang, W., Ouyang, W., Wang, X., Ren, J., Li, H., & Wang, X. (2018). 3d human pose estimation in the wild by adversarial learning. In: Proc. IEEE Conference on Computer Vision and Pattern Recognition, pp. 5255–5264.
116. Yang, Y., & Ramanan, D. (2013). Articulated human detection with flexible mixtures of parts. In: PAMI'13
117. Yasin, H., Iqbal, U., Kruger, B., Weber, A., & Gall, J. (2016). A dual source approach for 3d pose estimation from a single image. In: CVPR, 2016.
118. Ye, Q., & Kim, T.-K. (2017). Occlusion-aware hand pose estimation using hierarchical mixture density network. arXiv preprint arXiv:1711.10872, 2017
119. Yu, F., & Koltun, V. (2016). Multi-Scale Context Aggregation by Dilated Convolutions. In: CoRR, abs/1511.07122.
120. Zanfir, A., Marinoiu, E. & Sminchisescu C. (2018a). Monocular 3d pose and shape estimation of multiple people in natural scenes - the importance of multiple scene constraints. In: CVPR, 2018.
121. Zanfir, A., Marinoiu, E., Zanfir, M., Popa, A.-I., & Sminchisescu C. (2018b). Deep network for the integrated 3d sensing of multiple people in natural images. In: NeurIPS. 2018.

122. Zeiler, M. D., Krishnan, D., Taylor, G. W., & Fergus, R. (2010). Deconvolutional networks. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Francisco, CA, 2010, pp. 2528-2535, doi: 10.1109/CVPR.2010.5539957.
123. Zhou, X., Huang, Q., Sun, X., Xue, X., & Wei, Y. (2017). Towards 3d human pose estimation in the wild: a weakly-supervised approach. In: Proc. IEEE International Conference on Computer Vision, pp. 398–407
124. Zhou, X., Sun, X., Zhang, W., Liang, S., & Wei, Y. (2016). Deep kinematic pose regression. In: Proc. European Conference on Computer Vision. Springer, pp. 186–201.
125. Zuffi, S., Romero, J., Schmid, C., & Black, M. (2013). Estimating Human Pose with Flowing Puppets. In: Proceedings of the IEEE International Conference on Computer Vision. 3312-3319. 10.1109/ICCV.2013.411.

# Appendix

### VideoPose/Crossview output joint order:

0: 'root',
1: 'rhip',
2: 'rkne',
3: 'rank',
4: 'lhip',
5: 'lkne',
6: 'lank',
7: 'belly',
8: 'neck',
9: 'nose',
10: 'head',
11: 'lsho',
12: 'lelb',
13: 'lwri',
14: 'rsho',
15: 'relb',
16: 'rwri'

### LearnTri output joint order:

0: 'rank',
1: 'rkne',
2: 'rhip',
3: 'lhip',
4: 'lkne',
5: 'lank',
6: 'root',
7: 'belly',
8: 'neck',
9: 'head',
10: 'rwri',
11: 'relb',
12: 'rsho',
13: 'lsho',
14: 'lelb',
15: 'lwri',
16: 'nose'