UNIVERSITY OF COPENHAGEN
FACULTY OF SCIENCE

MSc in Computer Science

Articulated 3D Human Pose Estimation in Climbing

Using RGB Monocular Video

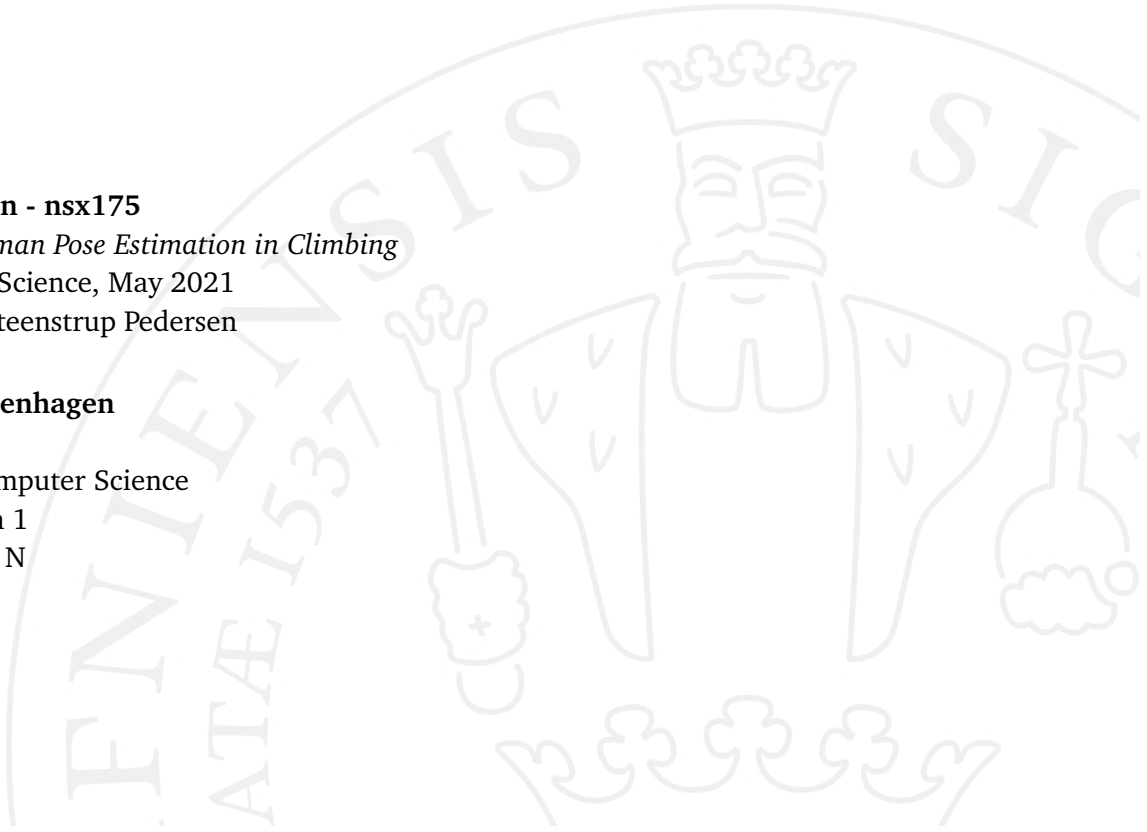
Tormod Mathiesen - nsx175

Supervised by Kim Steenstrup Pedersen

May 2021

Tormod Mathiesen - nsx175
Articulated 3D Human Pose Estimation in Climbing
MSc in Computer Science, May 2021
Supervisor: Kim Steenstrup Pedersen

University of Copenhagen
Faculty of Science
Department of Computer Science
Universitetsparken 1
2100 Copenhagen N



Abstract

The rising popularity of sports climbing has resulted in increased interest in the professional analysis of climbing technique. A 3D model of human pose can be used as a foundation for such in-depth analysis, as well as for comparison between climbers. There are many different computer vision models for creating 3D human models, and many have been created in just the last few years, largely due to the publication of a large dataset mimicking real-life scenarios, called 3D Poses in the Wild. One such model, called MEVA [1], uses a learned latent space of human motion to process predictions from a per-frame model, SPIN [2]. In this work, I train and evaluate versions of SPIN and MEVA on a brand new 2D labeled climbing dataset. The results are promising and show how MEVA is able to learn complex 3D information, even when only 2D annotations are available. The predictions made by the best model, MEVA90-Climb, are both accurate and smooth, giving natural-looking motions that can be used for climbing analysis.

Preface

This is an MSc thesis written as part of the completion of my education in Computer Science at The University of Copenhagen. It is written with a focus on Computer Vision and Machine Learning concepts under the supervision of Kim Steenstrup Pedersen, head of the Image Section at the Department of Computer Science.

Acknowledgements

I would like to thank my supervisor, Kim Steenstrup Pederson, for his moral support and excellent advice. Without which this would not have been possible.

I am also very grateful to Northtech Aps for providing an exciting use-case and motivation for this work, as well as the data collection. Special thanks to Jon Hemmingsen for his interest and driving contribution and Rolf Wiegand Storgaard for his help in planning and facilitation. Also, big thanks to Elias Laghouila, Rasmus Lykke, and Sara Pallesgaard for all the climbing and filming.

I would also like to express my thanks to Dansk Klatreforbund and Rune Windfeld for their aid in data collection and valuable insights from climbing expertise.

Lastly, I would like to thank my wife, Kristina, for her loving support and encouragement throughout writing this thesis.

Contents

Introduction

1

1.1 Motivation

Sport Climbing is an emerging exercise worldwide with increasing popularity among hobbyists as well as professional athletes. This is exemplified by its recent inclusion in large international competitions, like the 2018 Asian Games and the 2018 Summer Youth Olympics, as well as its planned debut in the 2020 Tokyo Olympic Games [3, 4], which are postponed to 2021. With this increase in professional activity comes an increased interest in an in-depth analysis of climbing techniques. This is where pose estimation comes in as a tool to create accurate 3D models that can be used to analyze an athlete's technique and style. Pose estimation tools have previously been used in the context of other sports, but as far as I am aware, not for climbing.

To fill this need, NorthTech ApS has set out to create an analysis tool for the sport, and this project is conducted partly in cooperation with them. Specifically, they would like to create a tool where one can take a regular video of a climber ascending a route and produce a 3D model from which further analysis can be made. The video would be filmed with a standard smartphone camera, requiring no extra equipment to be available for professionals and hobbyists alike. The subsequent analysis could then relate to elements of the climber's technique, including factors like speed and efficiency, as well as visually comparing climbing strategies between climbers. An important part of an analysis of climbing efficiency is the climber's center of gravity in relation to the hands and feet. For example, a mistake regularly made by new climbers is to not use their legs enough, meaning that they do not keep their center of gravity above their feet and rely too much on their arms to carry their weight, leading to faster exhaustion. This project focuses on producing a 3D model that can be the foundation for this kind of analysis rather than the analysis itself. But this application does provide some requirements that inform the selection of a 3D pose estimation model. NorthTech's role in this project has been to provide this use-case and motivation, as well as collect the climbing data detailed in Sec. 2.

1.2 Inherent Difficulties.

Some aspects of this task make it difficult compared to some previous work in the pose estimation field. Firstly, we are attempting to infer 3D information from 2D data, which is inherently difficult. We do not have any depth information, like multiple views or motion sensors, at our disposal. Also, the videos are not filmed by a stationary camera, which means that the camera movements must be taken into account. Sometimes there are also camera shakes that make the subject move around rapidly in the image and can introduce blur. Another issue is that the camera is situated at ground level, and if the climber is ascending a tall route, the subject will move further and further away from the camera. This makes it harder to extract meaningful information from

the images since fewer pixels are used to capture the subject. Additionally, the low viewing angle of the climber ascending a tall route means that features of interest, like the climber's arms and head, can often be occluded by the climber itself. Occlusion is a known problem in the pose estimation field and can be dealt with in two primary ways. One is to use spatial data; based on the shoulder position, one can infer the areas in which the elbow can be without violating the range of motion available to humans. Another is to use temporal data; if occluded for a short time, keypoints will follow a pretty straight path from where they entered occlusion to where they are visible again.

1.3 Related work.

In recent years, work in the 3D pose estimation field has focused on parametric human body models, [5, 6, 7] which can easily be turned into a 3D human mesh that is usable for downstream tasks such as animation. Directly fitting a parametric 3D human body to image input has gained substantial traction over the years, morphing from methods that require silhouette or human input [8, 9, 10], to ones that can directly fit model parameters to 2D joint positions [11], and to ones that can directly estimate shape and pose from images [12, 7, 2, 13, 14]. A problem in the field is the lack of ground truth 3D labels, and therefore many models use a weakly supervised approach to fit the 3D human body to 2D joint positions [12, 14]. Although these single-frame methods achieve impressive results, their predictions tend to be unstable when viewed as a sequence of poses, or motion. Using temporal information is a natural extension to single frame methods. This can be one using an LSTM [15, 16], temporal convolution [17, 18, 19] or fully connected layers [20]. Also , some methods predict 3D joint positions directly from images and use a temporal filter to post-process the motion sequence [21, 22].

1.4 Model selection.

Although there is a shortage of 3D annotated datasets, there are some publicly available ones. One of these is 3D Poses in the Wild (3DPW) [23] which is considered a relatively hard dataset with scenes that are close to real-life scenarios. This dataset is widely used in the field, and most report their performance on the 3DPW test set. Because of this, I decided to use the 3DPW leaderboard [24] to guide my choice of model. When searching for a suitable model, I discarded ones that did not fit the following criteria:

1. It must work without using depth information or multiple views. This is simply because of the limitations in the data available for this project.
2. It somehow makes use of temporal information. Because of the requirements provided by the use case, the 3D model predictions need to be smooth over time.

The best performing models on 3DPW that fulfill these criteria are Pose2Mesh [17], VIBE [25], and MEVA [1]. Pose2Mesh [17] uses a graph-convolutional approach estimating 3D coordinates of human mesh vertices directly from the 2D human pose. VIBE [25] utilizes temporal information by employing a Gated Recurrent Unit (GRU)

to convert the input frames of features into a sequence of temporally correlated features and then uses a discriminator, trained with a Generative Adversarial Network (GAN), to enforce a natural-looking motion. MEVA also uses an initial GRU to encode temporal information but uses compression to a human motion latent space learned by a Variational Autoencoder (VAE) to enforce a natural-looking motion. Out of these, MEVA had the best performance both in terms of accuracy (MPJPE) and smoothness (ACC-ERR). Thus, I chose MEVA as my model, and implementing and evaluating it, has been the primary focus of this project, as well as describing SPIN [2] and SMPL [5], which MEVA relies on.

1.5 Kinds of motion.

In the context of pose estimation, we define motion as a sequence of poses. With this definition, one can start to describe what kind of motions are common in climbing. Most of the time, the climber will be standing in an upright pose, in relation to the ground, with wide legs and hands up in front of them, something like what you see in Fig. 2.1b. There are, however, many cases in which more complicated poses are required, some examples of which are in Fig. 2.2. But irrespective of the specific poses, the motion in climbing can roughly be categorized into two types, namely static and active motion. Static motion is when the climber is in a stable position with hands and feet on grips or features on the wall. In these situations, the climber will make only minor adjustments to their pose, like shifting their weight, changing their grip, or looking around. Active motion is when the climber makes a move, most often from one stable position to another. These moves are often very quick compared to the longer periods of relative stability, and often such moves cause a reconfiguration of every part of the body. In some forms of climbing, like speed climbing, the climber may be almost exclusively in active motion, where they continuously move up the wall, but this is more of a niche, so using these two categories of motion is suitable to describe most climbing scenarios. There will often be a disproportionate amount of static motion, as opposed to active motion. This poses a problem for motion-based pose estimators like MEVA [1], where motions are smoothed by compressing information. If a sequence of frames consists of mostly static motion, and a small part is active motion, this active part may be smoothed out to look more like the static motion. This could cause the model to appear stiff or not using its full range of motion. This could be an area in which pose estimation in climbing differs from pose estimation in general, so whether or not this is the case should be investigated. A variable that affects the magnitude of this smoothing effect is the sequence length, and modifying this is the subject of the experiments described in Sec. 4.4, Sec. 4.5, and Sec. 4.6.

1.6 This Project

Most of my programming work is based on that in the official MEVA Github repository [26], which again has a lot of code from both VIBE [27] and SPIN [28]. I rewrote and adapted the parts necessary to train and evaluate on my data. The exact amount of modification made to the initial codebase is difficult to quantify. Additionally, I wrote

code to pre-process and load the climbing data, train and run the models on DIKU's compute cluster, evaluate and analyze model performance, and more. All my code and videos with rendered model meshes are provided in the attached zip file.

The following chapters describe the collection and annotation of climbing data as well as public datasets used for training Sec. 2, an explanation of the methods and metrics used Sec. 3, conducted experiments and results Sec. 4, a discussion of the results and models' performance Sec. 5, future work Sec. 6, and conclusions Sec. 7.

Datasets

2

2.1 Climbing data

2.1.1 Collection

Before the start of this project, NortTech ApS agreed to collect data for a climbing dataset and, in return, gain access to code and results made during the project. The data consists of videos filmed by NorthTech in two different locations, one outside and one inside. We wanted the videos to be similar to what an average hobby climber would film and therefore chose to use regular smartphone cameras held vertically. The outside location was filmed first, before the project started, and includes 9 videos with an average length of about 4 minutes. The second location was filmed later and included 115 videos, most of which are much shorter, typically 10-20 seconds, although a few are a couple of minutes long. I discarded the ones that were shorter than 90 frames since the pre-trained MEVA model[1] requires sequences of that length. This decreased the number of inside videos to 110.

This gives a climbing dataset of 119 videos, 9 filmed outside and 110 inside. They feature four different climbers, climbing 23 different routes. The inside videos were filmed with two cameras simultaneously, giving two angles on the same scene. However, they are not synchronized and often split the recording of the same scene differently, so they can not be paired one to one. The various combinations of climber, route, resolutions and video lengths are shown in Tab. A.1. All the videos have a frame rate of 30 frames per second, and a total of 149007 frames, which is about 83 minutes of video.

2.1.2 Challenging Features

The outside videos begin with the climber close to or standing on the ground, and he gets further away from the camera until he reaches the top (example in Fig. 2.1). This poses a challenge to pose estimation, as the further away the camera is, the fewer pixels there are to capture the person.

There are some poses in this dataset that, although normal in climbing, are not commonly encountered in other contexts. Some examples of such poses are in Fig. 2.2. These might be especially challenging for models trained without climbing-specific data.

In some sections of the outside videos, there are multiple people in the shot. This poses a problem for the 2D pose predictor used for annotation (more on that in the next section), where it might place a few points on the other subjects, which will make the ensuing bounding box too large. There are, however, only a few short sequences where this is the case, so I have decided to ignore the issue.

(a) Frame number 0 **(b)** Frame number 2000 **(c)** Frame number 5000

Figure 2.1.: Differences in scale. Three frames from the video IMG_2139 illustrating the differences in scale when climbing tall routes.

(a) A move called "Figure Four" from IMG_2306. **(b)** A high step move from IMG_2314. **(c)** A heel hook from from IMG_2315.

Figure 2.2.: Difficult poses. Three frames illustrating poses that are not commonly encountered outside the context of climbing.

2.1.3 Annotation

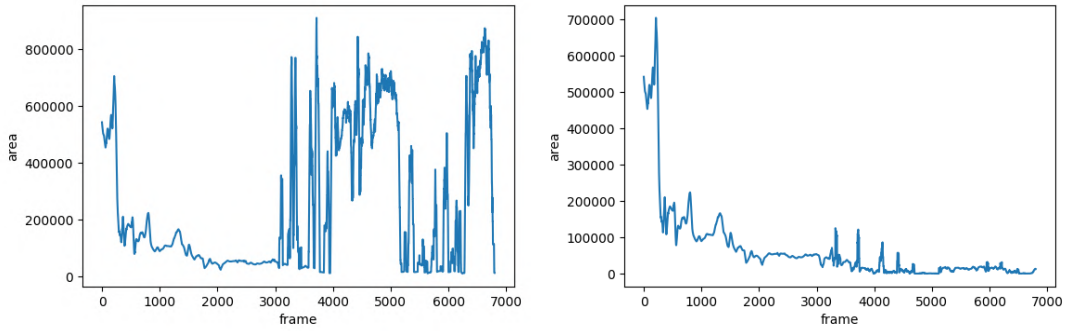
We did not use any motion capture equipment when filming and did not have multiple synchronized views. Therefore, the data can not be annotated with 3D ground truth. The best we can get is 2D annotations of joint locations. However, it is labor-intensive to annotate every frame manually, and time is a limited resource. I quickly realized that I would not be able to annotate all the data by hand, so in addition to hand annotation, I made use of the predictions from a pre-trained 2D pose model to annotate data in two different ways. Firstly, I use the predictions to speed up the manual annotation process, and secondly, I use them as "weak labels" for all the data that I did not get to annotate by hand.

For 2D pose predictions, the DarkPose[29] model is chosen for its good performance in the COCO keypoints dataset[30] and its availability through the MMPose library[31], which I have already used for other parts of the project. It is trained on the COCO keypoints dataset as well as MPII [32], and has state-of-the-art performance as witnessed on the COCO keypoints leaderboard [33].

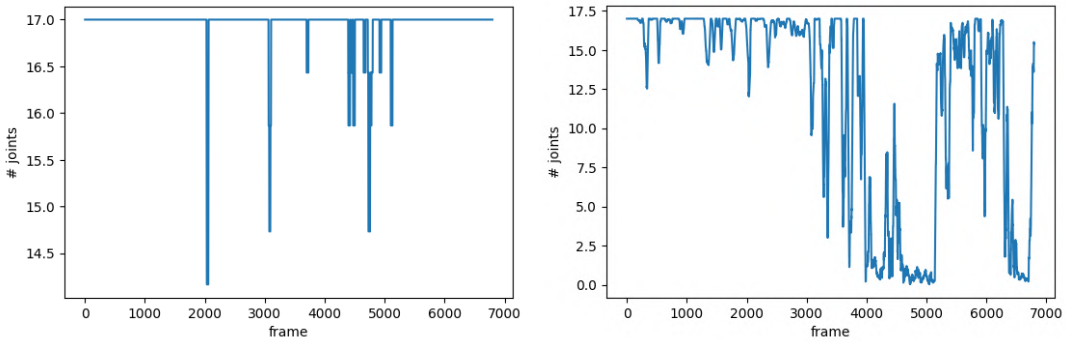
During the training of the MEVA model [1], the labels that come from this model are handled a little differently from hand-annotated ones. Each point has an associated confidence score $c \in [0, 1]$, given by DarkPose. Although imperfect, this score is a good estimate of how likely the point is to be correctly placed. The loss function uses this to weigh the contribution of each point by its confidence. The means that when DarkPose does not give a high confidence for a joint location, it doesn't contribute as much to the training.

Additionally, I use a confidence threshold to discard the worst joint predictions. This is primarily because the annotations are used to make bounding boxes for cropping the frames by finding the minimal square box that, with a margin, encompasses all the points. So if an erroneous point is far away from the subject, the resulting bounding box would be way too big. To find the best value for this threshold, I looked at the bounding box area for each frame and tested to see which threshold values would flatten the area curve, see Fig. 2.3. In the video IMG_2139 we expect the bounding box area to decrease throughout the video, so the larges areas in the latter half of Fig. 2.3a are erroneous. A confidence threshold of 0.2 makes the curve look more like we would expect, but comes at cost of a substantially lowered number of joint for the affected frames, as shown in Fig. 2.3d. The decreased number of joints does decrease the learning value in form affected frames, but since the removed points had low confidence, and probably erroneous, the decrease in learning value is minimal. Subsequently, I decide on using a confidence threshold of 0.2 to avoid the bounding box issue.

For manual annotation, I use an annotation tool called CVAT[34], which provides good quality of life features for annotation and a handy interpolation tool for point location between annotated frames. It will update the interpolated frames whenever you annotate a new frame, making a coarse-to-fine annotation strategy useful for saving time since you do not have to manually annotate the frames where interpolation is adequate. CVAT can be used online for free, but with a data size restriction, so to



(a) Area with a threshold of 0.0 . **(b)** Area with a threshold of 0.2 .



(c) Number of joints with a threshold of 0.0 . **(d)** Number of joints with a threshold of 0.2 .

Figure 2.3.: Threshold analysis. Bounding box area in pixels and the number of kept joints per frame with different joint confidence thresholds on DarkPose [29] predictions for IMG_2139. The plots are smoothed with a rolling average over 30 frames for visibility. In (a), you can clearly see that the enlarged bounding box sizes are a problem as it gets past around 3000 frames. This is remedied by applying the joint confidence threshold in (b). This thresholding comes at the cost of a lowered number of joints, as seen in (c) and (d).

circumvent this, I run it locally as a docker image following the description on their web site.

I want to annotate joints that are relevant for our use case and can be regressed from the SMPL model [5]. SMPL comes with a learned regressor from mesh vertices to a set of 3D points, as well as a regressor to 2D COCO[30] annotations. These two sets of point annotations mostly overlap, but not completely. SMPL has more points (particularly in the torso), and COCO has more points for facial features.

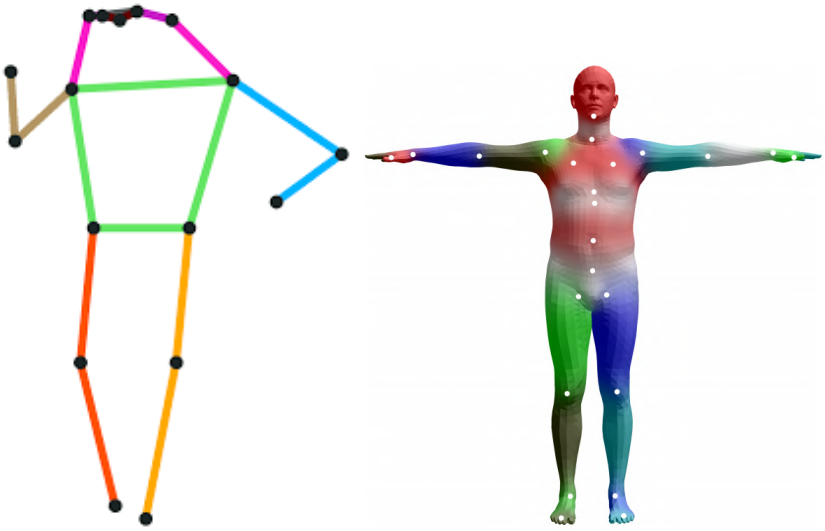
For the manual annotation I chose to use the COCO labels (Fig. 2.4a) with a few modifications for annotating the climbing dataset. I omit two points and add four from the SMPL joint set (Fig. 2.4b), giving a total of 19 points.

- Omitted 2 eye labels. We only care about the head's orientation, which we can infer from the nose and ear points.
- Added 2 hand joints. The joint where the middle finger meets the hand. Gives us the orientation of the hand.
- Added 2 foot joints. The center ball of the foot. Gives us the orientation of the foot.

To annotate as much data as possible in the limited time available, I decided to only annotate every fifth frame and let CVAT's linear interpolation fill in the rest. This does come at the cost of lowered temporal resolution in the ground truth annotation. However, the subsequent annotation data seems to fit quite well most of the time, only veering off a little bit. Only occasionally does the linear interpolation fail by a significant margin. This is when arms or legs move very fast, or the camera shakes. However, these events are rare and only affect the precise location of the joints in a few frames. Even though this will somewhat affect the model training, I think the effects should be negligible compared to the effect of an increased amount of training data.

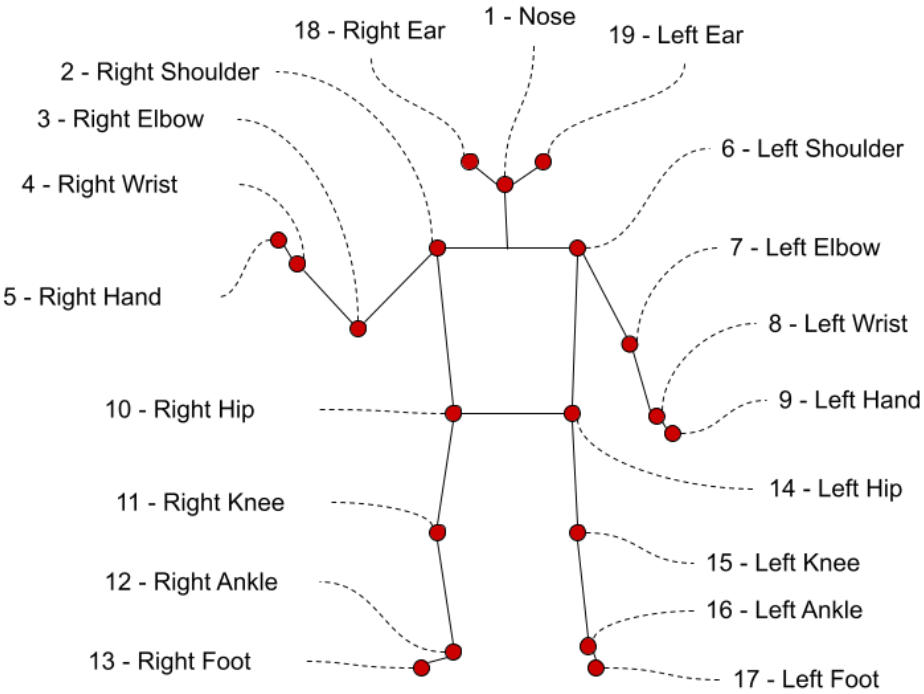
To further increase annotation speed, I use the 2D joint locations from DarkPose at every fitfth frames as a starting point for my annotation. One danger of using predictions from a pre-existing model as ground truth for training another model is that it could learn to reproduce the errors made by the first model. To mitigate this issue, I employ the following strategy:

1. In addition to joint locations, DarkPose gives a per-joint confidence level $c \in [0, 1]$, and I only use the joint location if it has high confidence, $c \geq 0.8$.
2. After importing the prediction into CVAT, I go through every predicted frame and correct inaccurate predictions. This takes some time, but it is much faster than manually annotating these points. It makes many predictions in the earlier parts of the videos, but most of them have high accuracy and do not need correction. Later on, it makes more mistakes, but also fewer predictions due to the confidence threshold. As described above, I only include joint locations for every fifth frame and leaves the rest for linear interpolation. This means that there are fewer predictions to check in this step.



(a) The 17 2D joint locations used in COCO annotation. In my annotation I omit the eye-points.

(b) The 24 3D joint locations in SMPL. In my annotation I use the hand joints from the index finger base as well as the joints on the toe base.



(c) The 19 2D joint locations used to annotate the climbing data.

Figure 2.4.: Annotated joints. Joint annotations in COCO and SMPL, and which of them I use for manually annotating the climbing data.

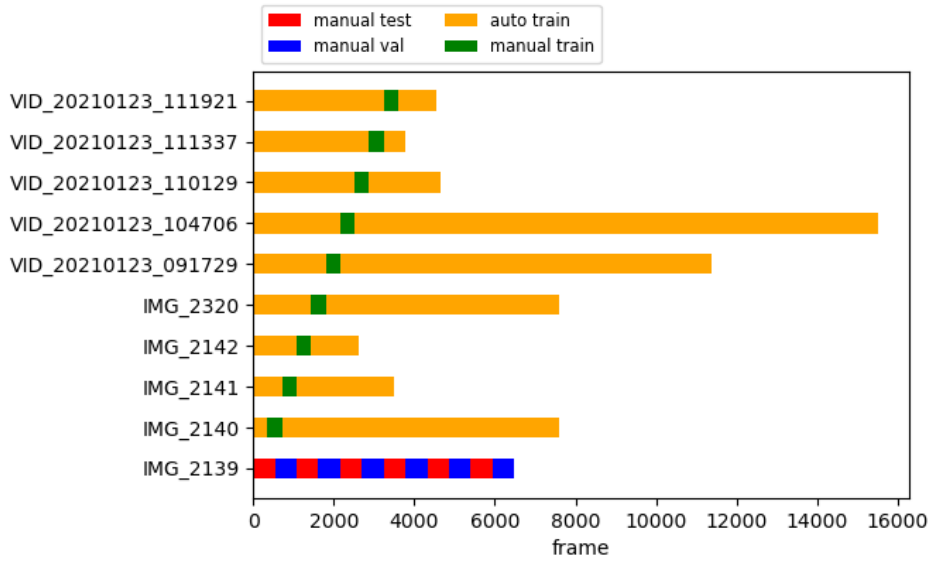


Figure 2.5.: Annotation and data splits. These are the 10 videos that have any manually annotated frames, and they are divided into test, validation, and training splits. Video IMG_2139 is the only one that is fully hand-annotated and is used purely for test and validation. The rest of the videos, not shown in this figure, are used for training and are only annotated automatically.

3. After correcting the predictions from DarkPose, I go through every fifth frame again and correct the joints that were linearly interpolated. The accuracy of these joint locations depends heavily on the frequency of keyframes and therefore takes more time when DarkPose performs poorly.

By the time it came to training the model, I had managed to manually annotate all of IMG_2139 as well as 12-second intervals from 9 other videos, the distribution of which can be seen in Fig. 2.5, giving a total of about 10000 frames, or 7%. The total time spent manually annotating is difficult to estimate, as I did not track the time spent. But I estimate a total time of abut 40 hours.

2.1.4 Test, Validation, and Training Splits

Since the manually annotated data has the most reliable accuracy and IMG_2139 is fully hand-annotated, I decided to use it for testing and validation. This has the advantage that no frames from this video are seen during training, which makes the sets less dependent on each other. Although, the test and validation sets are not as independent since they are drawn from the same video. I want both the test and validation sets to include frames where the climber is far away. To achieve this, I split the video into 12 equal splits of 540 frames and assign the first split to the test set, the second to validation, and so on. This division, as well as which frames are hand-annotated, is shown in Fig. 2.5, and the number of frames in each split is shown in Tab. 2.1.

Split	Videos	Frames	Manually annotated
Test	1	3240	100%
Validation	1	3240	100%
Training	118	142178	2.3%

Table 2.1.: Data split. The test and validation sets include frames from the same video, which is 100% manually annotated. The training set includes all the other videos and is about 44 times as large as the other two, but has very little manual annotation.

2.2 Public datasets

In addition to the dataset collected for this project, a few publicly available datasets are used for training.

Archive of Motion Capture as Surface Shapes (AMASS) [35] is a large and varied database of human motion consisting of 18 different marker-based mocap datasets. AMASS unifies the representation of these different datasets by converting the mocap data into SMPL[5] parameters. The resulting dataset is very rich, having more than 40 hours of motion data, spanning over 300 subjects and more than 11000 motions. There is no image or video data associated with this data, only mesh model motions. This means that it cannot be used to train a model relying on such inputs. But it can be used to learn which motions are possible for humans. AMASS attempts to capture the variety of human motion, including some more uncommon movements that push the body to its extremes, like backbends, splits, and tumbling. Because of this, it is used by MEVA[1] to train the variational autoencoder, which is described further in Sec. 3.4.

3D Poses in the Wild (3DPW) [23] is a 3D annotated video dataset captured outside in natural environments. It features more than 51.000 frames of challenging video sequences with changing and cluttered backgrounds as well as frequent occlusions. 3D annotation is difficult to do by hand and is most often done with motion capture equipment which lends itself to a controlled indoor environment. 3DPW achieves this in-the-wild annotation by equipping the subjects' limbs with IMUs associated with corresponding 2D pose detections and solving a graph-based optimization problem that forces 3D to 2D coherency within a single frame as well as across a range of frames. They prove the efficacy of this method by evaluating it on another dataset with 3D ground truth and synchronized IMUs. The resulting accuracy is high enough for it to serve as a benchmark for image-based 3D pose estimation.

MPI-INF-3DHP [36] is a 3D dataset with ground truth captured with a multi-camera marker-less motion capture system. It features eight different subjects in indoor and outdoor scenes, including everyday motions as well as dynamic sport motions. The training split features 8 different subjects captured from nine different angles, giving a total of roughly 576 minutes of video. The test set features 6 subjects captured from a single angle, with a total of about 14 minutes of video.

InstaVariety [16] is a 2D dataset with pseudo-ground truth annotations. It consists of 28,272 videos scraped off of Instagram by searching 92 movement-related tags, like dancing, golf, and surfing. An off-the-shelf 2D pose predictor produces the annotations,

and the authors show that adding such weakly annotated videos to a model's training monotonically increases its 3D prediction performance. The obvious advantage of automating annotation is the near-endless availability of online video data that becomes usable for training.

PennAction [37] is a 2D dataset with hand-made ground truth annotations. It has 2325 video sequences of 15 different actions. The actions are all sports-related, with examples like bowling, push-ups, and baseball swing. There are 13 annotated joints per frame, as well as a viewing angle associated with each video.

Method

3

3.1 SMPL - A Skinned Multi-Person Linear Model

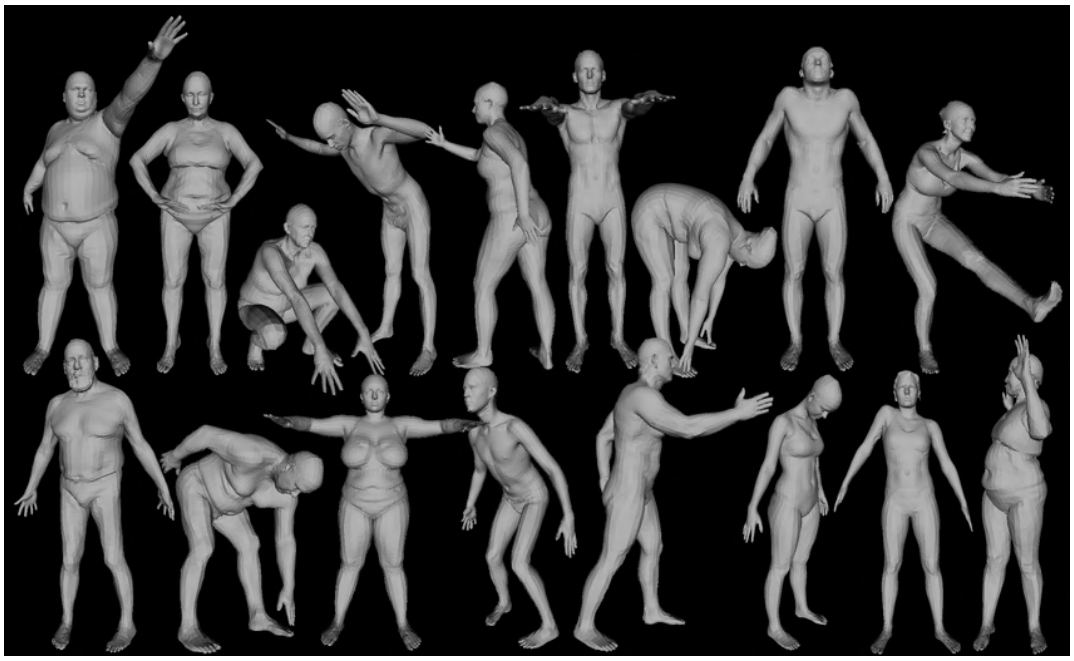
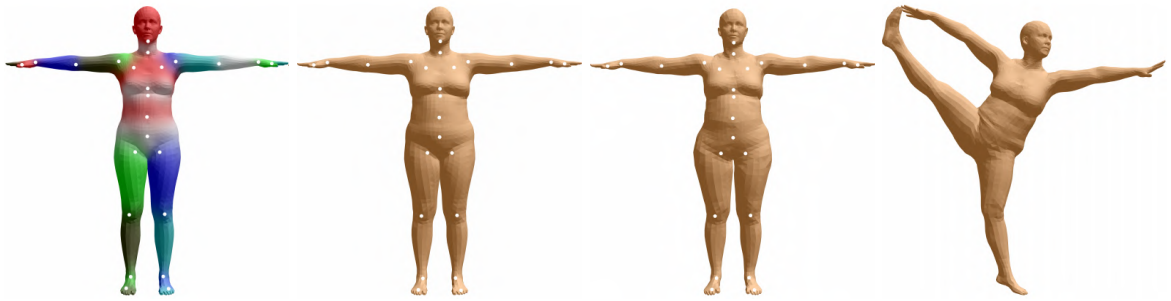


Figure 3.1.: SMPL examples. SMPL sample models showing different shapes and poses. This image is taken from the SMPL paper [5].

SMPL [5] (pronounced "simple") is a model of human body shape and pose. It is a vertex-based 3D mesh model learned from data that can realistically represent a wide range of human body shapes with natural pose-dependent deformations. Traditional methods model how vertices are related to an underlying skeleton structure, but SMPL differs from this in focusing on shape and pose as the two primary parameters. By learning separate functions for how shape and pose affect the vertex mesh and learning joint locations from shape, SMPL achieves higher accuracy than previous models like BlendSCAPE [6]. It uses a mesh with $N = 6890$ vertices and a skeleton tree of $K = 24$ nodes in 3D, where 23 are joints, and one is the root node.

The model is defined by the mean template mesh $\bar{\mathbf{T}} \in \mathbb{R}^{3N}$ in the zero pose $\vec{\theta}^*$, defined as the pose in Fig. 3.2a; a set of blend weights $\mathcal{W} \in \mathbb{R}^{N \times K}$, (Fig. 3.2a); a shape-dependent blend shape function $B_S(\vec{\beta}) : \mathbb{R}^{|\vec{\beta}|} \mapsto \mathbb{R}^{3N}$, (Fig. 3.2b); a function to predict joint locations $J(\vec{\beta})$, (shown as white dots in Fig. 3.2c); and a blend shape function $B_P(\vec{\theta}) : \mathbb{R}^{|\vec{\theta}|} \mapsto \mathbb{R}^{3N}$ accounting for pose-dependant deformations, (Fig. 3.2c). Finally, a standard blend skinning function $W(\cdot)$ (linear or dual-quaternion) is applied to rotate

the vertices around the estimated joint centers with smoothing defined by the blend weights $\mathcal{W}$.



(a) $\bar{\mathbf{T}}$, $\mathcal{W}$ **(b)** $\bar{\mathbf{T}} + B_S(\vec{\beta})$, $J(\vec{\beta})$ **(c)** $T_P(\vec{\beta}, \vec{\theta}) = \bar{\mathbf{T}} + B_S(\vec{\beta}) + B_P(\vec{\theta})$ **(d)** $W(T_P(\vec{\beta}, \vec{\theta}), J(\vec{\beta}), \vec{\theta}, \mathcal{W})$

Figure 3.2.: SMPL model formulation. (a) Template mesh with blend weights indicated by color and joints shown in white. (b) With shape-driven blend shape contribution only. (c) With the addition of pose blend shapes in preparation for the split pose, note the expansion of the hips. (d) Deformed vertices reposed for the split pose. The learned parameters $\{\mathcal{S}, \mathcal{P}, \mathcal{J}\}$ are omitted for brevity. This figure was taken from the SMPL paper [5].

The shape parameter $\vec{\beta} \in \mathbb{R}^D$ is in the shape space computed by running PCA on shape registrations. The PCA space is computed from a large dataset of body scans in the same pose, and choosing the dimensionality D of this space is a design choice that affects the fidelity of shape representations. MEVA[1] and SPIN[2] use an SMPL model with $D = 10$ shape dimensions, limiting the accuracy of the shape representation for the benefit of having fewer parameters to predict. Accurate shape recreation is not a primary goal in pose estimation, so this tradeoff strikes a balance between the advantages of an accurate shape space and model training time.

The pose parameter $\vec{\theta} \in \mathbb{R}^{3K}$ is defined as the set of axis-angle representations $\vec{\omega}_k \in \mathbb{R}^3$ for each joint k with respect to its parent in the skeleton tree. This gives a pose vector $\vec{\theta} = [\vec{\omega}_0^T, ..., \vec{\omega}_K^T]^T$ with $3 \times 24 = 72$ parameters, that is 3 for each of the 23 joints plus 3 for the root orientation. Let $\bar{\omega} = \frac{\vec{\omega}}{||\vec{\omega}||}$ be the rotation axis and $||\vec{\omega}||$ the angle of rotation. Then the axis angle for every joint j is transformed to a rotation matrix using the *Rodrigues formula*:

$$r(\vec{\omega}_j) = \mathcal{I} + \hat{\bar{\omega}}_j \sin(||\vec{\omega}_j||) + \hat{\bar{\omega}}_j^2 (1 - \cos(||\vec{\omega}_j||)) \tag{3.1}$$

where $\mathcal{I}$ is the 3×3 identity matrix and $\hat{\bar{\omega}}$ is the skew symmetric matrix of the 3-vector $\bar{\omega}$:

$$\hat{\bar{\omega}} = \begin{bmatrix} 0 & -\bar{\omega}_3 & \bar{\omega}_2 \\ \bar{\omega}_3 & 0 & -\bar{\omega}_1 \\ -\bar{\omega}_2 & \bar{\omega}_1 & 0 \end{bmatrix} \tag{3.2}$$

The standard linear blend skinning function $W(\bar{\mathbf{T}}, \mathbf{J}, \vec{\theta}, \mathcal{W}) : \mathbb{R}^{3N \times 3K \times |\vec{\theta}| \times |\mathcal{W}|} \mapsto \mathbb{R}^{3N}$ takes vertices in the rest pose $\bar{\mathbf{T}}$, joint locations $\mathbf{J}$, a pose $\vec{\theta}$, blend weights $\mathcal{W}$, and returns the posed vertices. In which, each vertex $\bar{t}_i$ in $\bar{\mathbf{T}}$ is transformed into $\bar{t}'_i$ as

$$\bar{t}'_i = \sum_{k=1}^{K} w_{k,i} G'_k(\vec{\theta}, \mathbf{J}) \bar{t}_i \tag{3.3}$$

$$G'_k(\vec{\theta}, \mathbf{J}) = G_k(\vec{\theta}, \mathbf{J}) G_k(\vec{\theta^*}, \mathbf{J})^{-1} \tag{3.4}$$

$$G_k(\vec{\theta}, \mathbf{J}) = \prod_{j \in A(k)} \left[\begin{array}{c|c} r(\vec{\omega}_j) & \mathbf{j}_j \\ \hline \vec{0} & 1 \end{array} \right] \tag{3.5}$$

where $w_{k,i}$ is an element of the blend weight matrix $\mathcal{W}$, representing how much the rotation of joint k affects the vertex i, $r(\vec{\theta}_j)$ is the local 3×3 rotation matrix corresponding to joint j, $G_k(\vec{\theta, \mathbf{J}})$ is the world transformation of joint k, and $G'_k(\vec{\theta, \mathbf{J}}))$ is the same transformation after removing the transformation due to the rest pose $\vec{\theta^*}$. Each 3-element vector in $\mathbf{J}$ corresponding to a single joint center j is denoted $\mathbf{j}_j$. Finally, $A(k)$ denotes the ordered set of joint ancestors of joint k.

The blend-shape functions B_S and B_P are linear functions defined by the parameters $\mathcal{S}$ and $\mathcal{P}$, respectively. These model how the template mesh $\bar{\mathbf{T}}$ is deformed by shape and pose represented as a vector of concatenated vertex offsets. These are defined as

$$B_S(\vec{\beta}; \mathcal{S}) = \sum_{n=1}^{|\vec{\beta}|} \vec{\beta}_n \mathbf{S}_n \tag{3.6}$$

$$B_P(\vec{\theta}; \mathcal{P}) = \sum_{n=1}^{9K} (R_n(\vec{\theta}) - R_n(\vec{\theta^*})) \mathbf{P}_n \tag{3.7}$$

where $\{\mathbf{S}_n, \mathbf{P}_n\} \in \mathbb{R}^{3N}$ are vertex offsets and elements of $\mathcal{S} \in \mathbb{R}^{3N \times |\vec{\beta}|}$ and $\mathcal{P} \in \mathbb{R}^{3N \times 9K}$. $\mathcal{S}$ and $\mathcal{P}$ are the learned parameters of these functions (nationally marked by being placed to the right of the semicolon). $R : \mathbb{R}^{|\vec{\theta}|} \mapsto \mathbb{R}^{9K}$ is a function mapping a pose vector $\vec{\theta}$ to a vector of concatenated joint relative rotation matrices, $r(\vec{\omega})$, where $R_n(\vec{\theta})$ is the n^{th} element of $R(\vec{\theta})$. Note that the blend shape function B_P is linear in $R^*(\vec{\theta}) = R(\vec{\theta}) - R(\vec{\theta^*})$, where $\vec{\theta^*}$ denotes the rest pose. This differs from previous work and guarantees that the contribution of the pose blend shapes for the rest is zero.

The joint locations are computed by J and depend on a person's shape. This is a linear function

$$J(\vec{\beta}; \bar{\mathbf{T}}, \mathcal{S}, \mathcal{J}) = \mathcal{J}(\bar{\mathbf{T}} + B_S(\vec{\beta}; \mathcal{S})) \tag{3.8}$$

where $\mathcal{J}$ is a learned matrix that transforms mesh vertices into joint locations.

The model. Once learned, the model parameters $\mathbf{\Phi} = \{\bar{\mathbf{T}}, \mathcal{W}, \mathcal{S}, \mathcal{J}, \mathcal{P}\}$ are fixed, and the resulting model $\mathcal{M}(\vec{\beta}, \vec{\theta}; \mathbf{\Phi}) : \mathbb{R}^{|\vec{\beta}| \times |\vec{\theta}|} \mapsto \mathbb{R}^{3N}$ maps shape and pose parameters to vertices (Fig. 3.2d).

$$T_P(\vec{\beta}, \vec{\theta}; \mathbf{\Phi}) = \bar{\mathbf{T}} + B_S(\vec{\beta}; \mathcal{S}) + B_P(\vec{\theta}; \mathcal{P}) \tag{3.9}$$

$$\mathcal{M}(\vec{\beta}, \vec{\theta}; \mathbf{\Phi}) = W\Big(T_P(\vec{\beta}, \vec{\theta}; \bar{\mathbf{T}}, \mathcal{S}, \mathcal{P}), J(\vec{\beta}; \bar{\mathbf{T}}, \mathcal{S}, \mathcal{J}), \vec{\theta}, \mathcal{W}\Big) \tag{3.10}$$

Here $\mathbf{\Phi} = \{\bar{\mathbf{T}}, \mathcal{W}, \mathcal{S}, \mathcal{J}, \mathcal{P}\}$ represents the learned model parameters for the mean template, blend weights, blend shape function parameters for shape, joint location function, and blend shape function parameters for pose. These are trained by minimizing reconstruction error on two datasets, *multi-shape* and *multi-pose*, containing meshes with the same topology as the template mesh that have been aligned to high-resolution 3D scans[38]. Because the model decomposes shape and pose, the parameters are trained separately, first $\{\mathcal{W}, \mathcal{J}, \mathcal{P}\}$ on *multi-pose*, then $\{\bar{\mathbf{T}}, \mathcal{S}\}$ on *multi-shape*, which simplifies optimization.

The creators of SMPL focused on its applicability in the field of 3D rendering by emphasizing its realism and compatibility with standard rendering engines like Maya, Unity, and Blender[5]. But it has also been used for 3D pose estimation to great success [2, 12, 25, 1]. When used in pose estimation, models predict the shape and pose parameters of SMPL rather than joint locations directly. This comes at the cost of an increased number of parameters to predict, but this increase is relatively small when using a shape space limited to 10 dimensions. When used in pose estimation, we need to estimate an additional set of weak perspective camera parameters $\Pi = \{s, t_x, t_y\}$ to be able to re-project joint locations to the camera plane. This gives a total of $10 + 72 + 3 = 85$ parameters, and a simple 3D pose model with the same predicted joint locations would have at least $3 \times 23 = 69$ parameters. However, MEVA uses a pose representation with 6 degrees of freedom per joint instead of 3 for continuity reasons, as proposed by Zhou *et al.*[39], making the total parameter count $10 + 144 + 3 = 157$.

The advantage of using SMPL lies in its decomposition of the spatial relations between joints into shape and pose. Using SMPLs learned shape space greatly restricts the possible configuration of joints. For example, a model where the left forearm is longer than the right simply cannot exist. This is, of course, an obvious example that could have been hard-coded by a programmer, but the advantage of using a shape space learned from data is that it captures correlations between spatial features that are not obvious. SMPL does not, however, restrict the space of possible poses. This means that a model could, for example, have a knee bent the wrong way or a hand located inside the torso. Learning the space of legal poses is left to the pose estimation model, which in the case of MEVA is handled primarily by its variational autoencoder, described in Sec. 3.4.

3.2 SPIN - SMPL oPtimization IN the loop

SPIN [2] is a model that predicts SMPL parameters from a single RGB image. Despite using no temporal information, it achieves a low error on the 3DPW test set [24]

with an MPJPE of 96.9. It achieves this by combining two concepts previously used in separation, namely HMR [12], a single-frame SMPL regressor, and SMPLify [11], an iterative optimization of SMPL parameters for 2D joints. See Fig. 3.3 for an overview of the model structure. A printout of the model's layers and parameters can be found in the appendix section C.

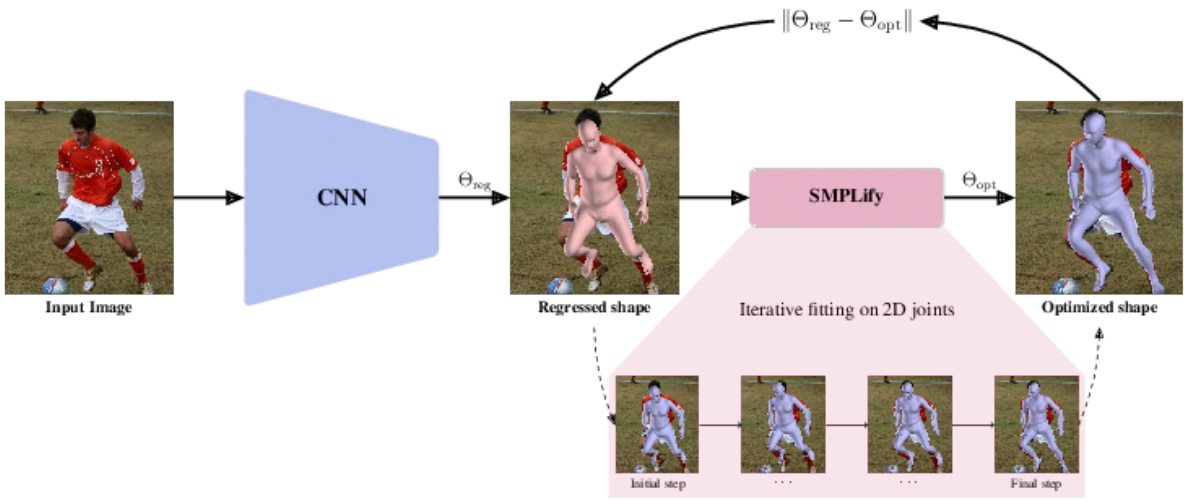


Figure 3.3.: SPIN overview. The convolutional regressor network produces an initial mesh, which SMPLify uses as the starting point for its iterative optimization to fit 2D joints to ground truth. The optimized mesh is then used to supervise the regressor.

Regressor. The regression model is a standard convolution network. Like in HMR [12], it uses the ResNet-50 [40] architecture pre-trained on the ImageNet classification task [41], with average pooling resulting in features $f \in \mathcal{R}^{2048}$. The only difference is that it uses the 6 degrees of freedom representation for 3D rotations proposed by Zhou *et al.* [39], because SPIN's authors observed that this increased convergence during training. With an SMPL mesh M and a pre-trained linear transformation $\mathcal{J}$ for joint locations, we define SMPL's joints as $X = \mathcal{J}M \in \mathbb{R}^{K\times 3}$. A forward pass with an image as input provides the regressed prediction for the SMPL parameters $\Theta_{reg} = \{\vec{\theta}_{reg}, \vec{\beta}_{reg}\}$ and the camera parameters Π_{reg}. This prediction allows us to generate the mesh $M_{reg} = \mathcal{M}(\vec{\theta}_{reg}, \vec{\beta}_{reg})$, as well as the joints $X_{reg} = \mathcal{J}M_{reg}$ and their re-projection $J_{reg} = \Pi_{reg}(X_{reg})$. With this setup, it is common to supervise learning with a re-projection loss on the joints:

$$L_{2D} = ||J_{reg} - J_{gt}||^2 \tag{3.11}$$

where J_{gt} is the ground truth 2D joints. However, SPIN's authors argue that this is a very weak supervisory signal that puts an extra burden on the network, forcing it to search in the parameter space for a valid pose that agrees with the ground truth 2D locations. This leads to the introduction of SMPLify as an alternative model for supervision.

SMPLify. This iterative fitting routine by Bogo *et al.* [11] tries to fit the SMPL model to a set of 2D keypoints using an optimization-based approach. It minimizes an objective function consisting of a re-projection loss, as well as pose and shape priors:

$$L_{opt} = E_J(\vec{\beta}, \vec{\theta}; K, J_{est}) + \lambda_\theta E_\theta(\vec{\theta}) + \lambda_\alpha E_\alpha(\vec{\theta}) + \lambda_\beta E_\beta(\vec{\beta}) \tag{3.12}$$

with intrinsic camera parameters

$$K = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.13}$$

where the focal length is f and the optical center is (c_x, c_y). E_J computes the 2D re-projection of joints J_{reg}, and calculates the re-projection loss L_{2D}. $E_\theta(\vec{\theta})$ is a mixture of a Gaussian pose priors trained with shapes fitted on marker data. $E_\alpha(\vec{\theta})$ is a pose prior penalizing unnatural rotations of the hinge joints in elbows and knees, as these have a fixed axis of rotation and limited range. $E_\beta(\vec{\beta})$ is a quadratic penalty on the shape coefficients. In the first step of the optimization, the model pose and shape are kept frozen so that only camera translation and body orientation are fitted. Then SMPLify attempts to minimize Eq. 3.12 in an iterative fitting procedure. In contrast to the 4-stage fitting procedure in [11], SPIN uses a single-stage procedure with just a small number of iterations because it is typically enough to converge to a good fit when initializing the optimization with regressed parameters. To alleviate problems with erroneous annotations, like under occlusion or with geometric inconsistencies, SPIN combines the annotated ground truth 2D joints with predictions from a 2D pose estimator in J_{est} [42, 43]. The advantage of this is that it can leverage the confidence scores provided by OpenPose to weight error contribution, and this idea is carried into what MEVA does when training on 2D data.

SPIN. Building on the insights from the previous two paradigms, SPIN forms a tight collaboration between them to train a deep regressor for human pose and shape estimation. During a typical training loop, an image is forwarded through the network, providing the regressed parameters Θ_{reg}. These are then used to initialize the optimization routine, which is usually very slow when initialized with the mean pose but is now significantly accelerated by a reasonable initial estimate. Let Θ_{opt} be the SMPL parameters produced by the iterative fitting, explicitly optimized to align with the 2D keypoints. Then we can directly supervise the regressor network on the parameter level with:

$$L_{3D} = ||\Theta_{reg} - \Theta_{opt}||^2 \tag{3.14}$$

In practice, the effect of using this loss is very different from applying the re-projection loss directly. Instead of letting the network search the parameter space to satisfy joint re-projection, we directly supply it with a privileged parametric solution Θ_{opt} which tends to be very close to the actual optimal solution. Fig. 3.4 shows how the optimized fit can differ with a good or a poor initialization.

An important characteristic of SPIN is that it has a built-in element of self-improvement. A good initial regressed estimate Θ_{reg} will lead to a better optimized fit Θ_{opt} which provides better supervision back to the regressor. Additionally, since the optimization routine uses only 2D joints for fitting, it is excellently positioned to utilize all the

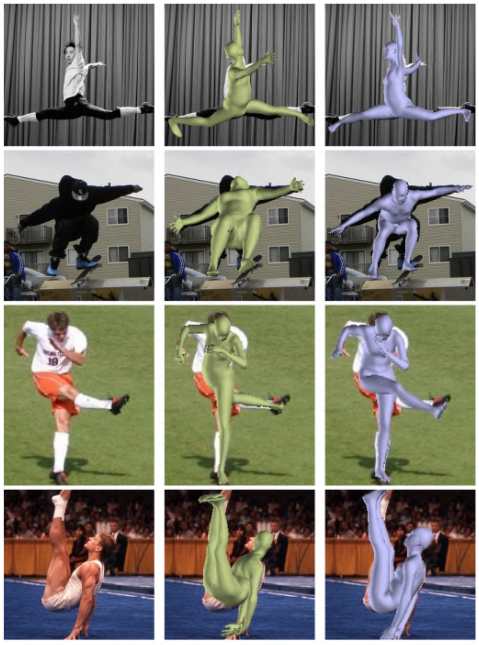


Figure 3.4.: SMPLify fitting. Examples from the beginning of training and at the end of training. Although SMPLify can fail when starting from an inaccurate pose (second column), given a good regressor prediction as initialization, the optimization can converge to an accurate solution (third column). This figure is taken from the SPIN paper [2].

available data where only 2D keypoint annotations are available in a better way than methods using only re-projection loss can. In this project, I do not make any changes to SPIN or re-train in any way, but use the pre-trained model provided in SPIN's official GitHub repository [28], which has a total number of 26 982 749 model parameters.

3.3 MEVA - Motion Estimation with Variational Autoencoding

MEVA [1] is a 3D human motion estimation model, where a motion is defined as a sequence of successive poses. It focuses on achieving both smooth and accurate motion and demonstrates that using temporal information can increase both metrics by outperforming single-frame models like SPIN [2] on the 3DPW test set [24] with an MPJPE of 85.7. It achieves this by splitting the task into three steps. First, it temporally correlates features, then it extracts a smooth motion representation using autoencoder-based compression, and finally, it adds back the fine motion details through a learned residual representation. The compression step captures the coarse overall motion, like general directions of limbs, while the last step reintroduces the fine details necessary for accuracy. Like SPIN, it uses the SMPL [5] representation of 3D human shape and pose, gaining the advantages of its built-in spatial information discussed in Sec. 3.1. MEVA is heavily based on previous work, most prominently VIBE [25], from which it

borrows the overall model architecture shown in Fig. 3.5, and SPIN [2], from which MEVA gets its initial per-frame features.

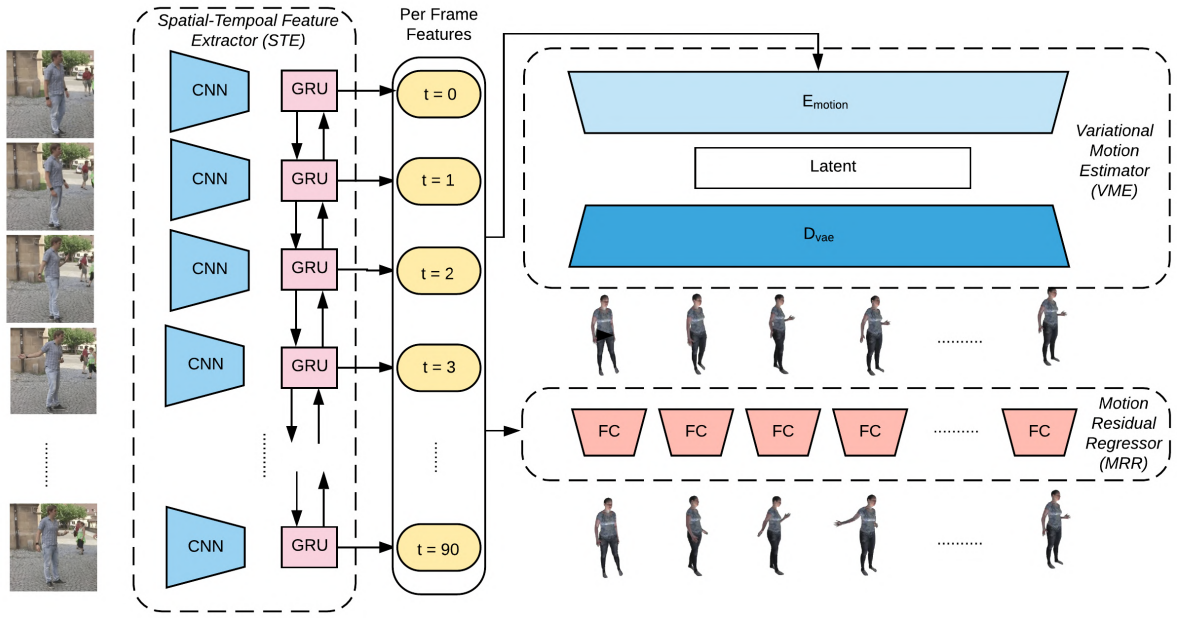


Figure 3.5.: MEVA architecture. The difference from VIBE's architecture is that VIBE uses a GAN module to produce the course motion estimates in place of MEVA's VME module. This figure is taken from the MEVA paper [1].

The MEVA model processes a sequence of frames simultaneously, the length of which is set to $T = 90$ frames in their implementation. The model's architecture is composed of three main parts: the Spatial-Temporal Feature Extractor (STE), the Variational Motion Estimator (VME), and the Motion Residual Regressor (MRR). For reference, a printout of the MEVA model's layers is included in the appendix section B. In total, the are 64 975 949 model parameters, of which 58 737 965 are trainable due to the frozen parameters in D_{vae}.

STE. In the STE module, a CNN model is first applied to each frame transforming the input frames $I_1, I_2, I_3, ..., I_T$ into features $\{f_1, f_2, f_3, ..., f_T\}$. There are many good single-frame 3D pose estimators that use SMPL parameters. Here, MEVA makes use of a model called SPIN[2], described further in Sec. 3.2. Instead of using the SMPL parameters estimated by this model, MEVA takes the features from the last layer of the model, which has 2048 parameters each. These features are then passed through two bi-directional Gated Recurrent Unit (GRU) [44] layers that encode the sequence of features $\{f_1, f_2, f_3, ..., f_T\}$ into temporally correlated features $\{f'_1, f'_2, f'_3, ..., f'_T\}$ of the same size.

VME. The VME module is responsible for extracting the coarse motion in the sequence. It achieves this by encoding the sequence of features into a latent space learned by a variational autoencoder. The autoencoder structure consists of an encoder E_{vae}, decoder D_{vae}, and latent space $z \in \mathbb{R}^{512}$. These are trained separately on the AMASS[35] dataset (more on that in Sec. 3.4), and then D_{vae} is frozen and kept for use in the VME module. Here, a new encoder E_{motion} is used to compress the sequence of features

$f'_1, f'_2, f'_3, ..., f'_T$ into the same latent space z. When training MEVA, D_{vae} is kept frozen, which forces E_{motion} to adapt to the latent space z enforced by D_{vae}. This latent space acts as a prior on human motion and makes the VME smooth the sequence into a motion that is feasible according to the prior.

MRR. Finally, the MRR module consists of two fully connected layers, each with 1024 neurons, taking as input the temporally correlated feature $f'_t \in \mathbb{R}^{2048}$ from the STE and the SMPL parameters $(\beta_t, \theta_t, \Pi_t) \in \mathbb{R}^{157}$ from the VME for each frame t. Using the same optimization technique as in SPIN[2] it iteratively refines the prediction for k iterations. This optimizer is normally initialized by the mean parameters for SMPL, but here it is instead initialized by the coarse motion parameters from the VME and iteratively refined by the residual information in the features. This adds back the fine details of the motion lost during the compression step. Even though this final step is done per frame, the features and SMPL parameters it takes as input are already temporally correlated due to the GRU layers in the STE and VME modules.

Training. Given a trained motion VAE, the MEVA modules STE, VME, and MRR are trained jointly end-to-end. The parts that are not trained are the decoder D_vae and the initial per-frame feature estimator where the SPIN model is used. Using data with various levels of annotation (2D joint positions, 3D joint positions, SMPL parameters), similar to in [16, 25, 2], the network is trained with losses consisting of L_{2D}, L_{3D}, L_{SMPL} as long as respective data is available. Specifically, if jp^{2D} and jp^{3D} denote 2D and 3D joint positions, and a hat $\hat{jp}$ denotes a prediction, then:

$$L_{MEVA} = L_{2D} + L_{3D} + L_{SMPL} \tag{3.15}$$

$$L_{2D} = w_{2D} \sum_{t=1}^{T} ||jp_t^{3D} - \hat{jp_t^{3D}}||_2 \tag{3.16}$$

$$L_{3D} = w_{3D} \sum_{t=1}^{T} ||jp_t^{2D} - \hat{jp_t^{2D}}||_2 \tag{3.17}$$

$$L_{SMPL} = w_\beta ||\beta - \hat{\beta}||_2 + w_\theta \sum_{t=1}^{T} ||\theta_t - \hat{\theta}_t||_2 \tag{3.18}$$

The weights $\{w_{2D}, w_{3D}, w_\beta, w_\theta\}$ decide the level of contribution from each part of the loss. I choose to use the same weights as the ones used to train the pre-trained version of MEVA, which are $w_{2D} = 300, w_{3D} = 300, w_\beta = 0.06, w_\theta = 60$. The optimizer parameters can be seen in Tab. 3.1. Across the experiments in Sec. 4 I use the following procedure when training MEVA:

1. Sequences are drawn randomly from the datasets. If multiple datasets are used, the drawn sequence could be from any of them, although when mixing 2D and 3D datasets, it is ensured that 60% of the sequences per batch are from a 2D dataset. The sampling method ensures that the data is exhausted before it redraws the same sequence so that every sequence has been seen once in a single pass through the data.

2. The sequence windows are generated with an overlap of 75%, or stride of $0.25T$.

Optimizer	ADAM
Learning rate	5e-4
Momentum	0.9

Table 3.1.: Optimizer parameters.

3. The sequences are batched into batches of size 32.
4. The training does not go through all the data per epoch but rather a fixed number of 500 batches per epoch. And the model trains for a total of 500 epochs.
5. During training, whenever the best validation loss so far is encountered, the model parameters are saved.
6. Whenever there has been no improvement to the validation loss for 5 epochs, the learning rate is decreased by one level of magnitude, down to a minimum of 5e-9.

3.4 Variational Autoencoder

A variational autoencoder is a way to learn a latent space that represents the input space using fewer dimensions. An encoder learns to encode samples x into latent codes $z \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. We force the distribution of the encoding to closely match the Gaussian distribution by minimizing the Kullback-Leibler divergence between them. Imposing a Gaussian prior on the latent space ensures that similar data samples will give latent codes that are near each other, as well as allowing more overlap between codes resulting in a smooth latent space. A decoder learns to decode the codes from the same latent space back to the input space, as close as possible to the original input. Formally, following previous work on VAEs [45, 46], the objective is to maximize the evidence lower bound of the log-likelihood

$$L_{VAE} = E_{q_\phi}[\log p_\lambda(x|z)] - \text{KLD}(q_\phi(z|x)||p_\phi(z)) \tag{3.19}$$

where λ and ϕ denote function parametrization.

VAEs are used for different things, like producing new previously unseen data samples that are similar to known data [47], forecasting [48, 46, 49], and denoising [50, 51], but in the context of encoding human motion, we are most interested in the denoising aspect. Here, we train an encoder E_{vae} taking a sequence of T poses as SMPL pose parameters $x = M_T \in \mathbb{R}^{T\times 144}$ and outputs the latent code z. Pose parameters use the 144 dimensional 6 degrees-of-freedom rotation representation[39] discussed in Sec. 3.1. The decoder D_{vae} takes the latent code z and reconstructs the motion $\hat{M}_T$. When a motion is forwarded through the VAE, it is in effect compressed to a lower-dimensional representation that only keeps the information that is most crucial to recreate the motion. But since the latent space cannot perfectly encode the input, some information is lost, resulting in a smoother, more general motion. Both E_{vae} and D_{vae} are implemented with GRUs, with the architecture in Fig. 3.6.

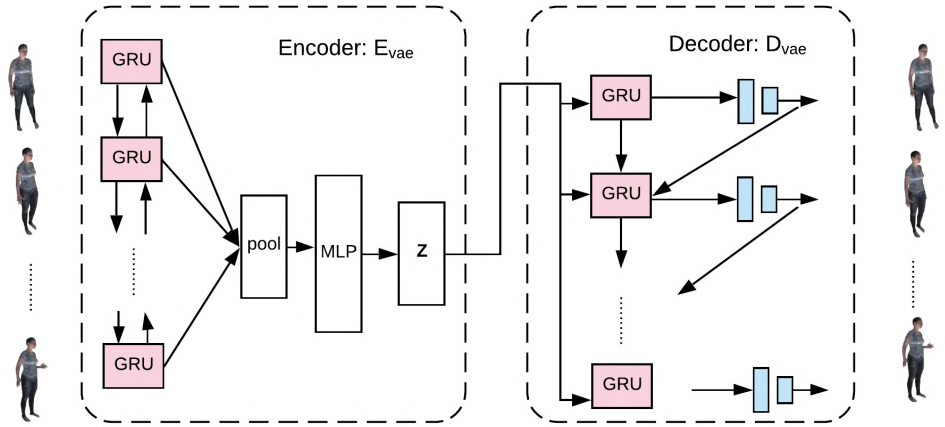


Figure 3.6.: Motion VAE architecture. This figure is taken from the MEVA paper [1].

Training and loss. Before MEVA can be trained, a VAE with the same sequence length T is trained on the large-scale human motion dataset AMASS [35] with over 14.000 motion samples of varying lengths. Given the Gaussian parametrization (μ, σ) of the VAE, the objective function Eq. 3.19 can be written as

$$L_{VAE} = -\frac{1}{S}\sum_{s=1}^{S} ||\hat{M}_T - M_T||^2 + \beta \cdot \frac{1}{S_z}\sum_{j=1}^{S_z}(1 + 2\log\sigma_j - \mu_j^2 - \sigma_j^2) \tag{3.20}$$

where S is the number of samples in the current batch, S_z is the dimension of the latent variable, and β is the weighting parameter. When the VAE is trained to a desirable reconstruction accuracy, the decoder D_{vae} is kept for later use with its parameters frozen.

Generalizeability. An important factor determining the success of the VAE's role in MEVA is its ability to encode new unseen motions, i.e., whether or not it generalizes well. This is crucial when we want to use it in an unknown domain, like climbing. As discussed in Sec. 1, there are many motions in climbing that are not commonly found elsewhere. The VAE training set, AMASS, includes mostly everyday common motions, like walking around and talking, although there are more active motions as well, like dancing and baseball. One part of the dataset, called MPI Limits, is particularly interesting in this context because it includes motions that push movements to their limit, showing exactly how far different joints can bend by doing things like splits and backward bends. This is important for the model to learn to represent the full range of motions that can be expressed by humans. But even with a large and expressive dataset like AMASS, we use data augmentation to expand the motion space even further.

Data Augmentation. To expand the dataset with more diverse but still plausible motions, we employ a few data augmentations. Given a human motion sequence of length T in SMPL parameters $M_T \in \mathbb{R}^{T\times 144}$ with a frame-rate F_{amass}, we do the following:

- **Speeding up and slowing down:** based on F_{amass}, we can uniformly upsample or downsample the frames and produce new sequences that are still plausible and natural human motion.
- **Flipping horizontally:** An action performed by the right arm can be equally validly performed by the left arm. Therefore, we can follow the kinematic tree of the SMPL model and mirror the motion across the left and right to generate a new motion sequence.
- **Random root rotation:** We randomly sample a root rotation from a unit sphere to capture different root orientations for possible human motion. The root orientation is in relation to the ground planes and coordinate systems and can be different for different pose estimators. Sampling random root rotation helps the model cope with different possible coordinate frame choices. Additionally, it will help the model deal with cases where we view the person from different angles, like in the climbing use case where we often view the climber from below.

With this augmentation strategy, we resample motion sequences to three different framerates, 20, 30, and 40 fps, produce a flipped version of all motions, and three different randomly rotated versions. This leads to an augmented dataset that is $3 \times 2 \times 3 = 18$ times larger than the original.

3.5 Metrics

In the field of 3D pose estimation, researchers typically report the performance of their papers in mean per-joint precision error (MPJPE), MPJPE after procrustes alignment (PA-MPJPE), and acceleration error (ACC-ERR). MPJPE describes a model's true accuracy, while PA-MPJPE describes how accurately a model predicts the pose when allowing translation, rotation, and uniform scaling to fit ground truth. ACC-ERR does not relate to accuracy, but rather the smoothness of the predictions over time, by finding the difference between the acceleration in ground truth points compared to predicted points. Normally these metrics are reported with respect to 3D ground truth points, measured in mm and mm/s^2, but since there is only have 2D annotations for my data I will have to do it a bit differently.

Object Keypoint Similarity. Instead of MPJPE and PA-MPJPE we need another metric to capture the accuracy of the model for 2D. In 2D pose estimation, a typical metric is one from the COCO Keypoints challenge [30], called Object Keypoint Similarity (OKS). It is a similarity measure between predicted and ground truth keypoint annotations, and is always in the range between 0 and 1, where 1 is a perfect match. On the COCO Keypoints challenge evaluation page, the measure is defined as

$$OKS = e^{-d_p^2/2s^2k_p^2)} \tag{3.21}$$

for a predicted point p where d_p is the euclidean pixel distance between predicted points and ground truth. d_p is passed through an unnormalized Guassian with standard deviation sk_p, where s is the object scale and k_p is a per-keypoint constant that controls falloff. The constants k_p are tuned so that the OKS is a perceptually meaningful and

easy to interpret similarity measure. This is done by finding the the standard deviation of $\sigma_p^2 = E[d_p^2/s^2]$ for 5000 redundantly annotated samples in in COCO's validation dataset. These values are only defined for the COCO keypoints, and since I use a few more keypoints in my annotation, for hands and feet, I extend it to include these. Values of σ_p are the same for the joints on both sides of the body, and since hand and foot joints are approximately as difficult to annotate as wrist and ankle joints I define $\sigma_{hand} = \sigma_{wrist}$ and $\sigma_{foot} = \sigma_{ankle}$. A perceptually meaningful and interpretable similarity metric is then obtained by setting $k_p = 2\sigma_p$. With this setting of k_p, at one, two, and three standard deviations of d_p/s the keypoint similarity $e^{-d_p^2/2s^2k_p^2)}$ takes on values of $e^{-1/8} = .88$, $e^{-4/8} = .61$ and $e^{-9/8} = .32$. As expected, human annotated keypoints are normally distributed (ignoring occasional outliers). Thus, following the 68–95–99.7 rule [52], setting $k_p = 2\sigma_p$ means that 68%, 95%, and 99.7% of human annotated keypoints should have a keypoint similarity of .88, .61, or .32 or higher, respectively (in practice the percentages are 75%, 95% and 98.7%). Note that this formulation only considers predicted keypoints, so if a point location is not predicted it does not not penalize the score. Also, it does not take occlusion into account, so occluded points count equally towards the score.

Modified acceleration error. To measure the smoothness of predicted 2D points I want to use something like ACC-ERR. Since the regular ACC-ERR, proposed in [16], uses 3D point locations I have to adjust it to the 2D case. We cannot measure the 3D acceleration in mm/s^2, so instead I will use euclidean pixel distance to measure 2D acceleration in d/s^2. And since the scale of the subject varies, like in OKS, this has to be normalized by scale for the ACC-ERR to be comparable between all frames. So for a ground truth point p and it's prediction $\hat{p}$ we have

$$\text{ACC-ERR} = \frac{1}{s} \cdot |\frac{d^2\hat{p}}{dt^2} - \frac{d^2p}{dt^2}| \tag{3.22}$$

where s is the scale of the object scale. We take the absolute difference between accelerations so that negative values do not cancel out the positives when calculating the mean.

For both OKS and ACC-ERR I set the scale s to be the length of one side of the square bounding box used to crop the input images.

Experiments and Results

4

I conduct five different experiments using pre-trained and trained versions of MEVA as well as the pre-trained version of SPIN. The performance of the per-frame SPIN model serves as a benchmark for gauging the value added by MEVA's use of temporal information. I evaluate the performance of each model in terms of the metrics described in Sec. 3.5, with an in-depth metric analysis of the video comprising the test and validation set, video IMG_2139. The metrics for each model, evaluated on the train, test, and validation splits, are in Tab. 4.1. A per-frame comparison of the OKS on the video used for testing and validation is in Fig. 4.1. Additionally, I make a qualitative analysis of each model, looking at how they deal with different aspects of the data by showing rendered example frames. The videos I use to provide these examples are [IMG_2139, IMG_2320, IMG_2304, IMG_2306, IMG_2314, IMG_2315], for which videos with rendered meshes produced by each model are provided in the attached zip file.

	Train Split		Val Split		Test Split	
	OKS	ACC-ERR	OKS	ACC-ERR	OKS	ACC-ERR
SPIN	0.725±0.26	5.7475	**0.912**±0.12	0.0214	**0.929**±0.12	0.1719
MEVA90	0.721±0.27	5.7565	0.839±0.18	0.0072	0.875±0.16	0.0066
MEVA90-Climb	0.726±0.26	**5.6875**	0.910±0.11	**0.0055**	0.922±0.11	**0.0051**
MEVA30	0.721±0.27	5.7874	0.845±0.16	0.0056	0.873±0.15	0.0055
MEVA30-Climb	**0.728**±0.26	5.7524	0.908±0.11	0.0063	0.922±0.10	0.0058

Table 4.1.: Metrics. Mean OKS with standard deviations and mean ACC-ERR for each model on test, validation, and training splits of the climbing dataset. The scores marked in bold font are the best for that metric and split. SPIN gets the best OKS on the validation and test splits by a small margin. But it is, in turn, quite outmatched in terms of ACC-ERR on the same splits. MEVA90-Climb receives the best ACC-ERRs across all splits, but in terms of OKS it performs almost identically to MEVA30-Climb. It seems like the train split is quite a lot harder than the others since it gives the worst scores, despite the models being trained on it.

All the experiments are run on DIKU's compute cluster, using machines with GeForce GTX TITAN X GPUs. Inference runtimes can be seen in Tab. 4.2.

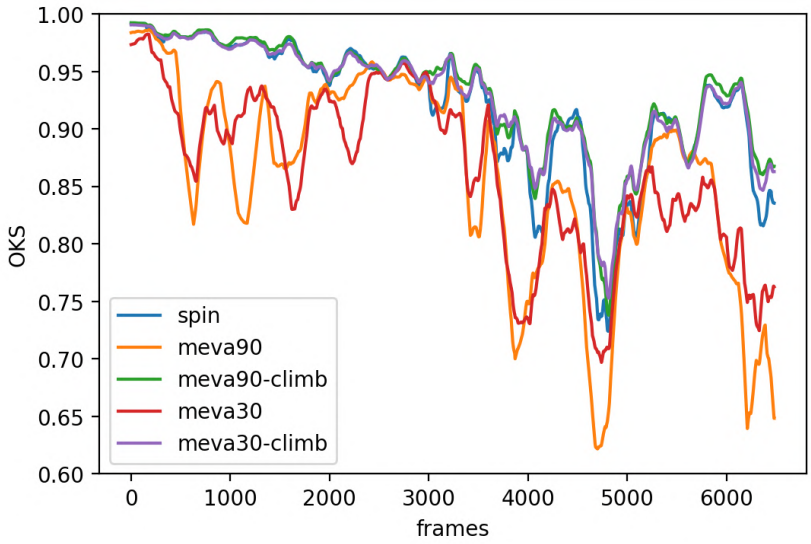


Figure 4.1.: OKS for each model. Mean OKS for every model per frame of video IMG_2139, the video used for testing and validation. The plots are smoothed with a rolling average over 180 frames for visibility. In the later frames, MEVA90-Climb and MEVA30-Climb outperform SPIN on the hardest sequences. The overlapping dips between MEVA90's and MEVA30's curves indicate that there is something about SPIN's predictions in these sequences that the models have not learned to deal with from training on the public datasets.

Model	seconds per frame	fps
SPIN	0.05313	19
MEVA90	0.00039	2589
MEVA30	0.00038	2628

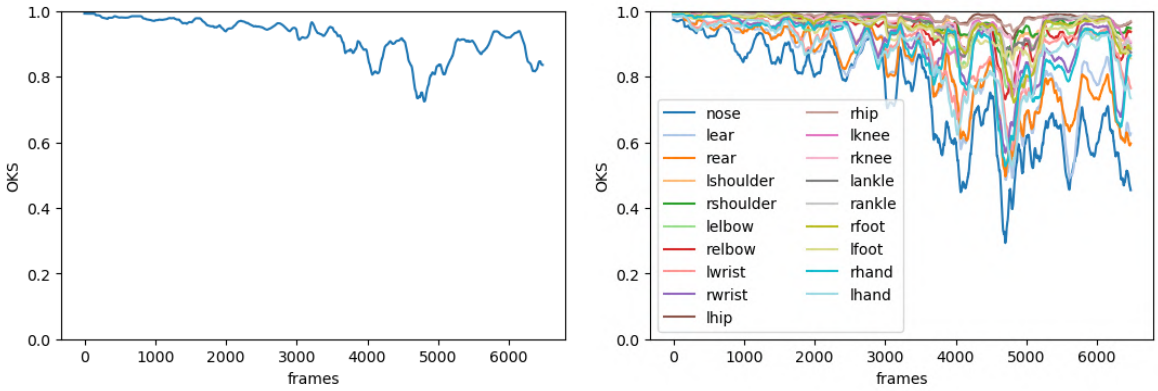
Table 4.2.: Average model inference speeds. Note that MEVA runs on the output from SPIN, so in a real-time scenario, you would have to first run SPIN on at least 90 or 30 frames, depending on MEVA's sequence length, before you could run MEVA. Then, SPIN's runtime would dominate the execution time. I left out the fine-tuned versions MEVA30-Climb and MEVA90-Climb since their lengths are the same as MEVA30 and MEVA90, so the execution time should be roughly equivalent. Even the difference between the runtimes of MEVA30 and MEVA90 is minimal and could easily be due to random changes in the runtime environment. Also, note that SPIN's runtime is heavily influenced by non-optimized IO operations, as described in Sec. 4.1.

4.1 SPIN

In this experiment, I use the pre-trained SPIN[2] model provided in their official GitHub repository [28]. This model has been trained on three 3D datasets: Human3.6M [53], MPI-INF-3DHP [36], and LSP [54], as well as three 2D datasets: LSP-Extended [55], MPII [32], and COCO [30]. I run SPIN on all the frames of my test, validation, and training data. For each frame, the resulting SMPL parameters are stored, as well as the last-layer features. These features are the ones the MEVA model takes as input,

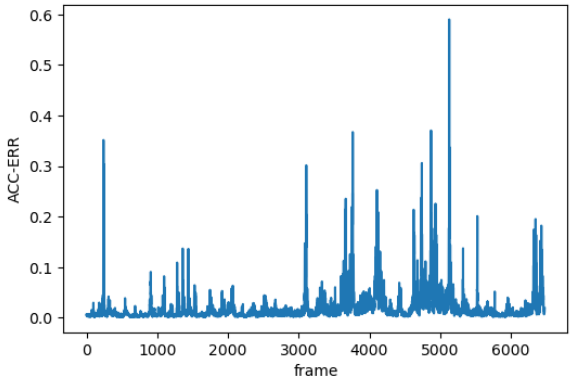
both during training and evaluation. This means that the performance of MEVA is dependant on reasonable performance from SPIN on at least a portion of the frames in a sequence.

SPIN takes normalized images of size 224x224. I crop and resize the frames to the size of the smallest square encompassing the bounding box of the climber, plus a small margin. As mentioned in Sec. 2.1.3, I use the annotated joint locations to produce these bounding boxes for the climbing data. If one were to run SPIN on un-annotated data, one would have to provide bounding boxes in some other way. This can be done using a pre-trained person detector and tracking model, for example, the Tracktor [56] model available in MMTrack [57], which I used while testing in an early stage of this project.



(a) Mean OKS It gets worse as the video progresses, as expected, since the climber gets further away. Although it does not decrease as much as it could have.

(b) OKS per joint The least accurate joints are the ones in the head, like eyes and ears, while the most accurate are the central joints like hips.



(c) ACC-ERR The spikes in ACC-ERR typically correspond with jitter and rotation errors, as shown in Fig. 4.3 and Fig. 4.4.

Figure 4.2.: SPIN metrics. OKS and ACC-ERR for each frame in video IMG_2139, the video used for testing and validation. The OKS plots are smoothed with a rolling average over 180 frames for visibility.

The runtime speed during inference reported in Tab. 4.2 are heavily influenced by non-optimized I/O-operations. During inference, it fetches image frames from a server across the network. This operation was left unoptimized in this project because it was

only performed once, and the resulting predictions and model features were saved for later use by MEVA.

Evaluating SPIN on the video IMG_2139 yields the curves in Fig. 4.2. Since MEVA takes SPIN's features as input, an analysis of what kind of errors SPIN makes can inform the understanding of MEVA's errors. To start, SPIN does, for the most part, give satisfactory predictions. As shown in the metric table Tab. 4.1, it has the highest OKS for the test and validation splits. However, it does have a lot of jitters which gives it a higher ACC-ERR, the significance of which is readily apparent if you look at the provided video renders. It is not as prevalent when looking at static images, but as an example, consider Fig. 4.3. In this example, the model switches up the climber's legs, which results in a completely wrong placement, as well as an inhumanly fast movement from one frame to the next. Another similar failure that introduces jitter is when it rotates the whole model wrong, an example of which is in Fig. 4.4. These particular errors where there are sudden dramatic alterations are examples of what we expect a temporal model like MEVA to be able to handle.

(a) Frame #97 from IMG_2139. **(b)** Frame #98 from IMG_2139.

Figure 4.3.: SPIN jitter example. The first frame fits the climber quite well, with some small positioning errors. In the next frame, the left and right leg are switched. This mistake also makes it rotate the model towards the camera to try and fit the position of the legs, which in turn brings the left arm forwards off of its previously correct placement. This change is over a very small time step, only a single frame, so when rendered in a video at 30 fps, even changes as small as the arm placement here add up to a jarring visual experience.

The third kind of failure is when it fails to interpret the full range of motion in a pose. An example is in Fig. 4.5. This kind of error has less to do with jitter and more to do with a consistent limitation on what kind of poses SPIN is capable of predicting accurately. The same error can be seen in Fig. 4.6c, where the highly placed left leg draws the right leg towards it. Similarly, SPIN also fails when presented with other extreme or uncommon poses, like those in Fig. 4.6).

(a) Frame #239 from IMG_2139. **(b)** Frame #240 from IMG_2139.

Figure 4.4.: SPIN rotation example. Here the model is rotated upside down from one frame to the next. We see that even in the first frame, the models struggle to place the torso and arms correctly. This could be due to the somewhat odd pose where the climber bends forward while keeping the right arm up and back, in addition to the left arm being occluded. These frames are taken from a sequence where the model repeatedly rotates back and forth, which produces a big spike in acceleration error around frame 250, as shown in Fig. 4.2c.

(a) Frame #902 from IMG_2139. **(b)** Frame #911 from IMG_2139.

Figure 4.5.: SPIN tilt example. In the first frame, the pose is estimated quite well, and even the occluded left arm is reasonably placed. A few frames later, the climber lifts his right leg to a high position, but rather than changing the position of just that leg, the whole model tilts backward, trying to fit the right knee. This brings the left leg off of its correct place and also causes corrections in the arms to adjust for the tilt.

(a) Frame #155 from IMG_2306. **(b)** Frame #75 from IMG_2314. **(c)** Frame #80 from IMG_2315.

Figure 4.6.: SPIN difficult poses. These are the same poses depicted in Fig. 2.2. Although SPIN performs quite well on each of them, they all have some problems. In (a), the main problem is that the left leg should be in front of the right arm. In (b), the right hand should be on the same grip as the foot, but it hovers above, and the arm clips through the leg. In (c), the left leg is not fully extended, and the right leg is drawn upwards.

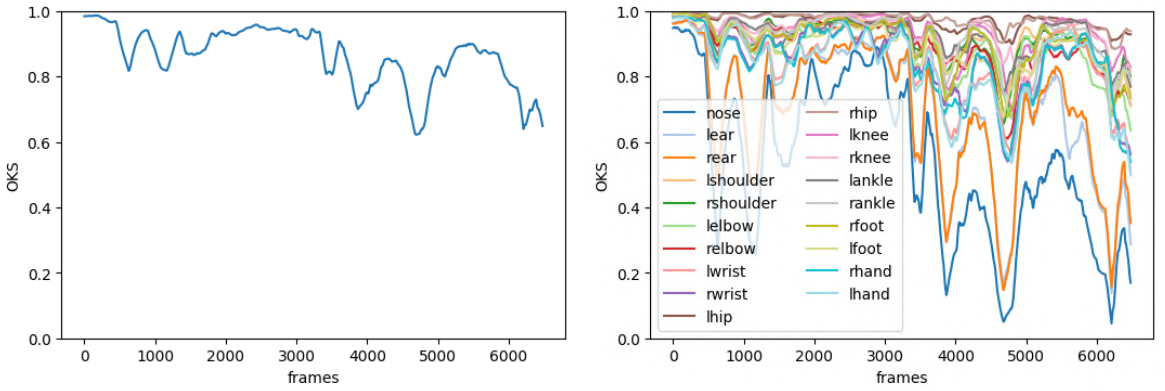
4.2 MEVA90

Here I use the pre-trained MEVA [1] model provided by their official GitHub repository [26], which I will call MEVA90. This model has been trained on three 3D datasets: Human3.6M [53], MPI-INF-3DHP [36], and 3DPW [23], as well as two 2D datasets: InstaVariety [16] and PennAction [37]. Using the features from the pre-trained SPIN model, I run MEVA90 on the test, validation, and training data. This version of the model uses the decoder from the pre-trained VAE model and takes a sequence of length $T = 90$ frames. This model is not trained on the climbing data, but I still evaluate its performance on all three splits of my data so that I can compare it to the other models.

To run this and the other versions of MEVA on the climbing videos, I have to slice the frames up into chunks of T frames which leaves a few frames out at the end of each video, depending on its length. One way to mitigate this would be to introduce some level of overlap and combine the results by averaging. This would have the added advantage of alleviating the problem of error spikes at the window intersections, discussed in the MEVA paper [1]. However, I did not implement that in this project, so every experiment with MEVA was done using windows with no overlap.

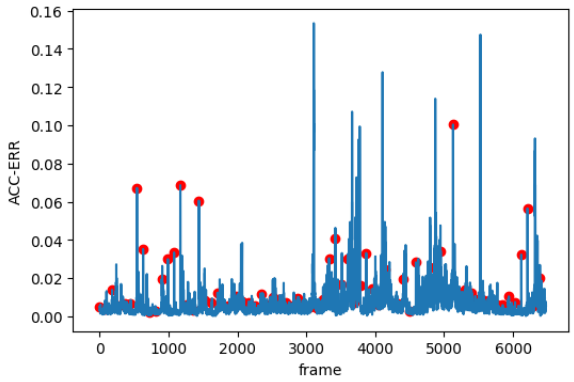
Evaluating MEVA90 on the video IMG_2139 yields the curves in Fig. 4.7, where we see the problem with jitter on the window intersections manifested in Fig. 4.7c. Despite this, the overall result is much smoother than SPIN. The total acceleration is lower in Tab. 4.1, and looking at the rendered videos gives a much smoother viewing experience.

There are, however, still some problems. One issue is that MEVA90 seems to consistently underestimate the scale of the climber, as shown in Fig. 4.8a. This seems to be a problem from MEVA's training, possibly due to some scaling in the pre-processing of the datasets. This issue is, however, known to the creator of the MEVA GitHub site [26]



(a) Mean OKS. Roughly similar to the curve for SPIN in Fig. 4.2a, but it has a few dips or sequences with poor accuracy.

(b) OKS per joint. Also similar to SPIN's curve in Fig. 4.2b, but the difference between head joints and the rest is more extreme.



(c) ACC-ERR The red dots mark where the intersections between the size $T = 90$ windows. We see that many of the spikes in acceleration error are due to these intersections.

Figure 4.7.: MEVA90 metrics. OKS and ACC-ERR for each frame in video IMG_2139, the video used for testing and validation. The OKS plots are smoothed with a rolling average over 180 frames for visibility.

and is mentioned under "Known Issues" on their README. Also, some sequences have radical changes in scale, in some cases making the model vanish entirely and then re-appear. An example frame from such a sequence is in Fig. 4.8b. Also, in combination with a vanishing scale, the model is sometimes flipped upside down, as in **??**.

Another issue is that not all the sequences passed through MEVA90 give reasonable results, some just produce what seems like garbage output. An example is in Fig. 4.9. These garbage sequences, along with the sequences with vanishing scale, explain the dips in OKS seen in Fig. 4.7a. Both phenomena seem to be more common in outdoor videos, like the test-validation video IMG_2139, but they also appear in indoor videos.

(a) Frame #0 from video IMG_2139. The scale is consistently too small.

(b) Frame #455 from video IMG_2320. Some sequences yield more extreme scaling problems. However, the pose still seems reasonable.

(c) Frame #490 from video IMG_2320. Rotation error where the model is flipped upside down.

Figure 4.8.: MEVA90 scaling errors.

(a) Frame #1200 from video IMG_2139.

(b) Frame #6905 from video IMG_2320.

Figure 4.9.: MEVA90 garbage sequences. Example frames from some of the sequences with garbage output.

4.3 MEVA90-Climb

In this experiment, I take the pre-trained MEVA90 and fine-tune it by training on my climbing dataset. I use the same configuration that was used to train MEVA90, i.e., the same values for loss weights, learning rate, number of epochs, etc., only exchanging the training and validation datasets for my own. Subsequently, I call this version of the model MEVA90-Climb. I had to make some adjustments to the training code to deal with my dataset correctly, but I was, for the most part, able to use MEVA's training code directly. The model is trained end-to-end, taking SPIN features as input and keeping the D_{vae} parameters frozen. 2D Keypoint loss while training can be seen in Fig. 4.10. The total time to train was 34 hours and 23 minutes.

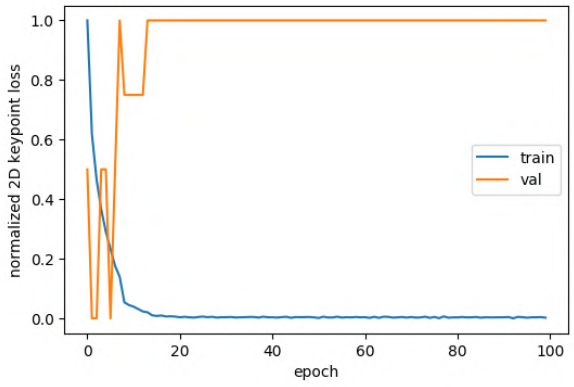
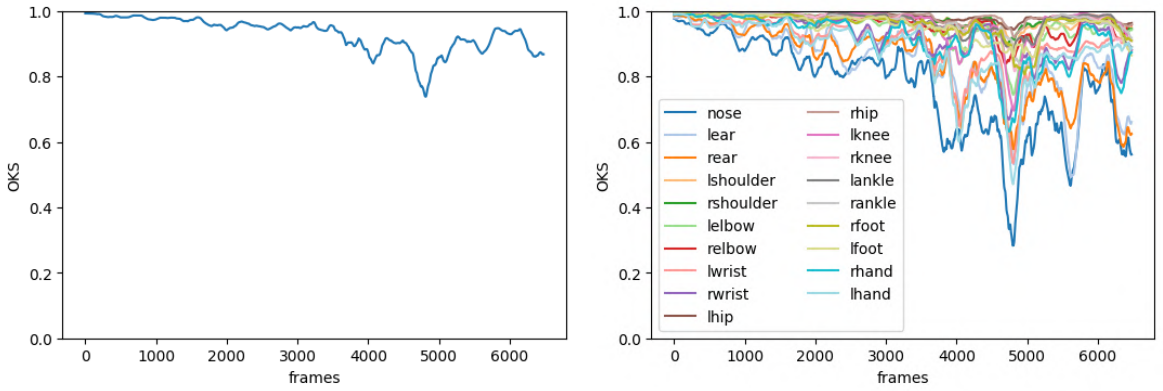


Figure 4.10.: Training MEVA90-Climb. 2D keypoint loss on train and validation set during training, normalized to [0,1] for visibility. The model reaches its lowest validation loss already at the second epoch. The training loss keeps going down until about epoch 20, after which both losses remain stable.

The resulting model gives even smoother and more accurate results than MEVA90, demonstrated by the scores in Tab. 4.1. MEVA90-Climb seems to learn the proper scale of the data and loses both the consistent underscaling and the vanishing scale problem that MEVA90 had. It also no longer has any garbage sequences, and subsequently, none of the extra dips in OKS in Fig. 4.11a. Although the smoothness of this model has benefits, there are also some costs to it. In Fig. 4.12, we see several frames from one of the difficult poses shown before, namely the "heel hook". In it, the model does not match the split pose until the next sequence, when that pose dominates the sequence.

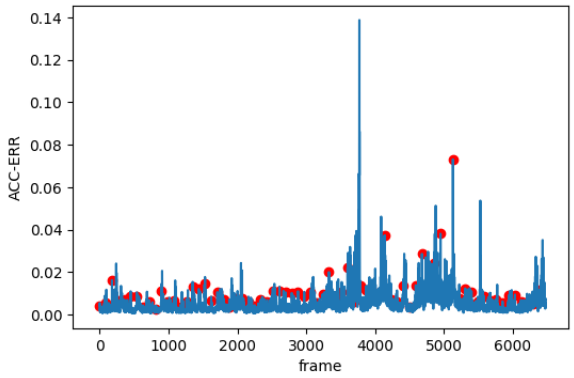
Even though the smoothing could cause some problems, there are also cases where it increases accuracy. In addition to the obvious accuracy advantages of removing outliers, there are some more subtle improvements. In Fig. 4.13b we see the "high step" pose, where previously SPIN yielded a mesh where the right arm clipped through the right leg. MEVA90-Climb manages to correct this by bringing the torso forwards, making the whole pose more accurate in 3D. This kind of improvement does not manifest in an improved OKS since the 2D coordinates were already good, but if we had 3D ground truth, we would see an improvement in depth error.

There is one big spike in acceleration error, visible around frame 3760 in Fig. 4.11c. This is due to a sequence of bad SPIN predictions, which MEVA90-Climb is unable to smooth out, causes the model to rotate when it should remain stable.



(a) Mean OKS. Without the dips in OKS that MEVA90 had, this curve is closer to SPIN's curve.

(b) OKS per joint. There is less spread between head joints and the rest than with MEVA90.



(c) ACC-ERR. There are far fewer spikes in acceleration error than MEVA90 had, including fewer spikes on window intersections (marked by red dots).

Figure 4.11.: MEVA90-Climb metrics. OKS and ACC-ERR for each frame in video IMG_2139, the video used for testing and validation. The OKS plots are smoothed with a rolling average over 180 frames for visibility.

4.4 VAE30

As discussed in Sec. 1, the kinds of motion in climbing may affect the level of smoothing that should be applied. Since climbing typically consists of mostly static motion, interspersed by active motion where the climber moves to a new stable position, it could be that smoothing over shorter sequences would result in higher accuracy. To investigate this, I shorten the model's sequence length from 90 frames down to 30 frames. This requires the model to be completely retrained, including the human motion subspace used to compress motion.

To learn the new human motion subspace, we use a VAE trained on AMASS [35], a large dataset of pose sequences described in standard SMPL parameters. AMASS is a collection of datasets that evolves over time, so depending on exactly which version of AMASS MEVA [1] used for training, mine might have grown to include a few more datasets, possibly a 10% growth. This increase in data size could have

(a) Frame #60 from video IMG_2315.

(b) Frame #80 from video IMG_2315.

(c) Frame #105 from video IMG_2315.

Figure 4.12.: MEVA90-Climb over-smoothing problem. In (a), the legs are together. In (b), the legs are in a split pose, but the model "lags behind". The frames in (a) and (b) are in the same sequence of $T = 90$, where the split pose comes into play late in the sequence. In (c), the model accurately matches the legs in a split pose. The frame in (c) is from a sequence where most of the time is spent in this split pose.

(a) Frame #155 from video IMG_2306. This pose is not more accurate than SPIN's, but the motion is much smoother.

(b) Frame #75 from video IMG_2314. Here the model's right arm is no longer clipping through it's leg. This is because the whole torso is, more correctly, leaning forward. However, the right arm is still placed to high compared to where it should be, on the same grip as the foot.

Figure 4.13.: MEVA90-Climb difficult poses.

lead to a VAE that generalizes better, but I will not get into comparing the two in this report. The MEVA GitHub repository [26] provides code for training the VAE as well as data-augmentation on AMASS. This augmentation consists of changing speeds by downsampling or upsampling, flipping left and right, and random 3D rotation of root orientation. The mean squared error and Kullback-Leibler divergence during training can be seen in Fig. 4.15a and Fig. 4.15b.

(a) Frame #3765 from video IMG_2139. Some accuracy errors, but overall a reasonable estimate.

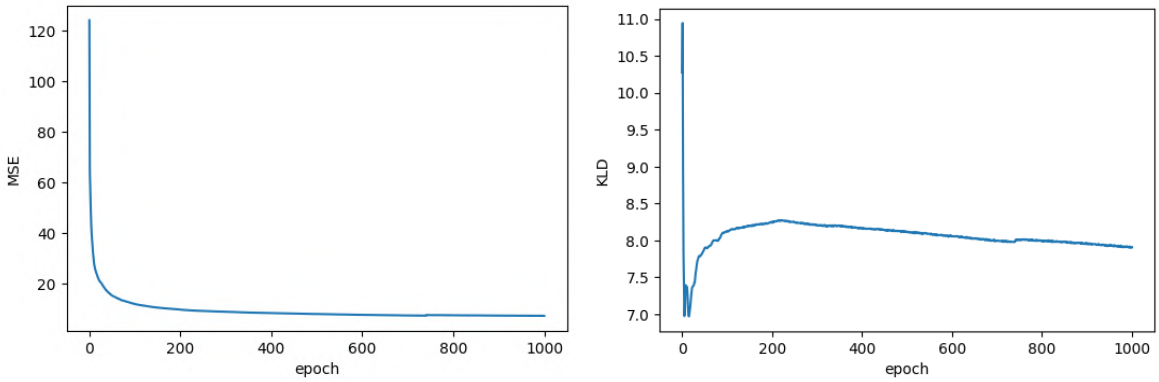
(b) Frame #3770 from video IMG_2139. The whole model is rotated to the left.

(c) Frame #3775 from video IMG_2139. The model is rotated back to how it should be.

Figure 4.14.: MEVA90-Climb under-smoothed rotation. This is the motion sequence explaining the spike in ACC-ERR around from #3770 in Fig. 4.11c. The cause is a series of bad SPIN predictions, which MEVA90-Climb is not able to smooth away.

The model is trained on the augmented data, using the augmentation strategy described in Sec. 3.4 for 1000 epochs. Since the dataset is very large, an epoch is limited to 50,000 randomly sampled sequences. Total training time for VAE30 was 32 hours and 21 minutes.

Since I do not evaluate VAE30 directly on any test data, I cannot know how well it generalizes or how it compares to the VAE used by MEVA90. However, we can get a sense of how well it works by looking at the models MEVA30 and MEVA30-Climb.



(a) VAE30 MSE. The mean square recreation error. It follows a typical asymptotic training curve with diminishing returns per epoch.

(b) VAE30 KLD. Kullback–Leibler divergence.

Figure 4.15.: Losses during training for VAE30. The loss is weighted to prioritize MSE over KLD. After the initial drop in KLD it seems like the model chose to "sacrifice" some KLD in favor of better MSE. But after epoch 200 or so, both metrics decrease steadily.

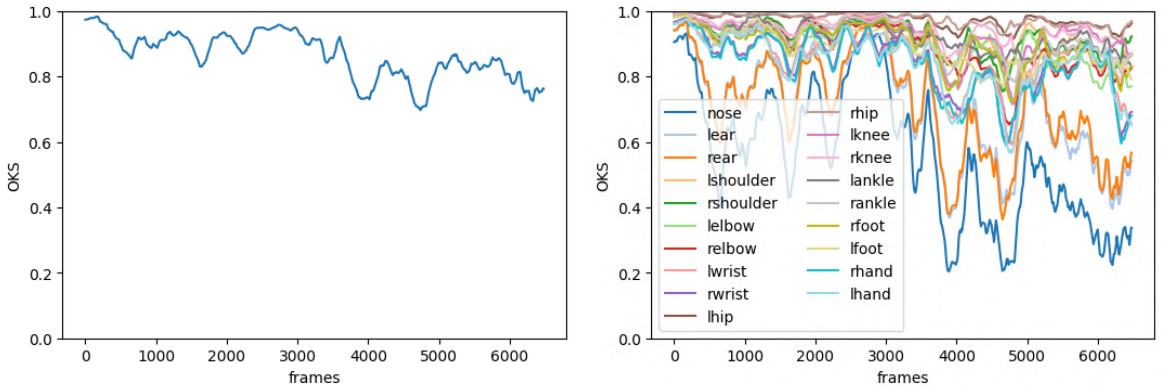
4.5 MEVA30

Model	MPJPE	PA-MPJPE	ACC-ERR
MEVA90	85.7	51.9	11.6
MEVA30	94.2	59.6	13.2
SPIN	96.9	59.2	116.4

Table 4.3.: Performance on 3DPW test set. Note that the scores for MEVA90 differ slightly from those in the paper[1]. This is because of a few small changes implemented since it was published, as described on MEVA's GitHub [26].

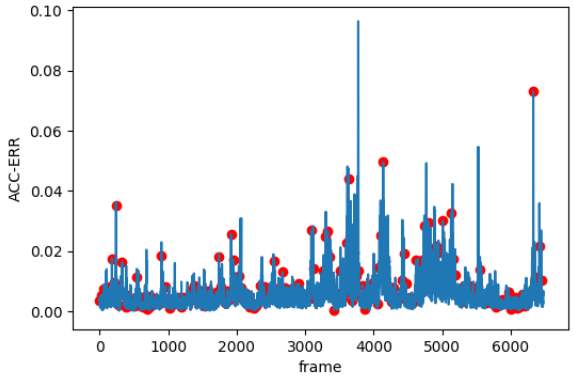
After training VAE30, we take the decoder and use it in MEVA's VME module as described in Sec. 3.3. Then I train MEVA jointly on datasets: MPI-INF-3DHP [36] and 3DPW [23] InstaVariety[16], and PennAction[37], validating on the validation split from MPI-INF-3DHP. This process mirrors what was done with MEVA90 but with a shorter sequence length, and I call this model MEVA30. Note that the list of datasets differs from what MEVA90 was trained on in that Human3.6M is missing. This is because, at the time of writing, the MEVA GitHub repository [26] includes pre-processing code for all these datasets except for Human3.6M, and I did not have time to recreate that for this project. Training and validation losses during training can be seen in Fig. 4.18a. The training time was 23 hours and 34 minutes for this model, although it reached its best validation loss very early, after only 4 epochs.

In addition to the climbing data, I evaluate this model on the test set from 3DPW [23]. The results of this, as well as MEVA90's results as reported by MEVA's authors, are in Tab. 4.3. Here, MEVA30 performs worse on all metrics, although compared to other methods listed on the 3DPW leaderboard [24] MEVA30 would still be competitive. The ACC-ERR, in particular, remains very low, while the MPJPE and PA-MPJPE have dropped to a level comparable to that of SPIN.



(a) Mean OKS. Like with MEVA90, there are some dips in the curve.

(b) OKS per joint. Like with MEVA90, the head joints (eyes and ears) score significantly worse.



(c) ACC-ERR. Like with MEVA90, many of the spikes correspond with window intersections, except now there are more of them since the sequence length is shorter.

Figure 4.16.: MEVA30 metrics. OKS and ACC-ERR for each frame in video IMG_2139, the video used for testing and validation. The OKS plots are smoothed with a rolling average over 180 frames for visibility.

Even though it clearly performs worse on 3DPW, on the climbing data MEVA30 performs quite similarly to MEVA90. In the metrics table Tab. 4.1 we see slight improvements on all scores, except ACC-ERR on the train split, where it is worse than MEVA90. These similarities are mirrored by similar-looking graphs in Fig. 4.16.

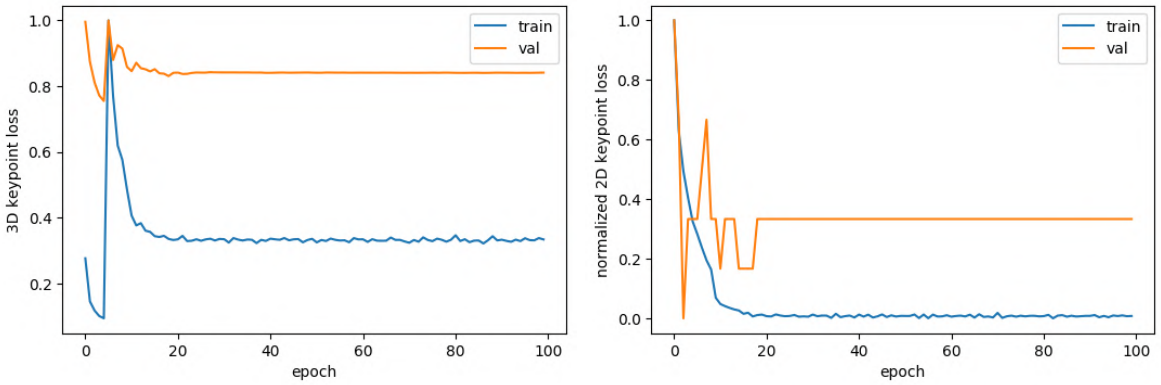
It also has the same scaling problems as we saw in MEVA90. As shown in Fig. 4.17 both the consistent underscaling and the vanishing scale problem appear here. However, there are no garbage sequences like we saw with MEVA90.

(a) Frame #0 from video IMG_2139. The scale is consistently too small.

(b) Frame #445 from video IMG_2320. Some sequences yield more extreme scaling problems. However, the pose still seems reasonable.

(c) Frame #490 from video IMG_2320. In combination with a vanishing scale, the model is sometimes flipped 180 degrees.

Figure 4.17.: MEVA30 scaling errors. This mirrors the same issues as with MEVA90 in Fig. 4.8



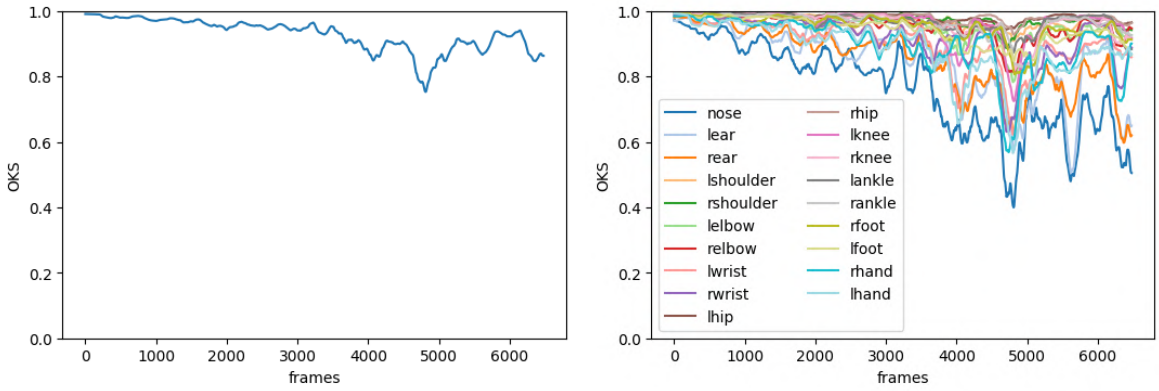
(a) MEVA30. 3D keypoint loss on train and validation set during training, normalized to [0,1] for visibility.

(b) MEVA30-Climb. 2D keypoint loss on train and validation set during training, normalized to [0,1] for visibility.

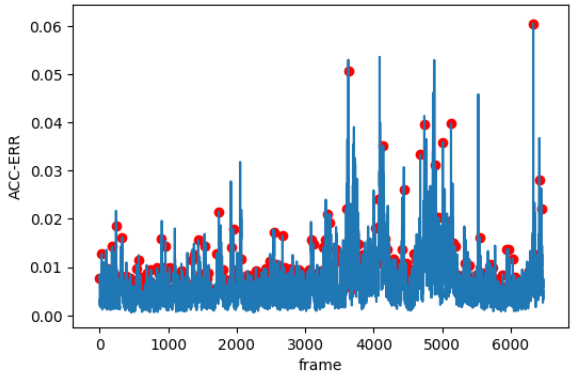
Figure 4.18.: Losses during training for MEVA30 and MEVA30-Climb. Both models converge quite early in the course of the training.

4.6 MEVA30-Climb

Then, like with MEVA90-Climb, I fine-tune MEVA30 by training it on my climbing data, giving the final model MEVA30-Climb. This process mirrors the relationship between MEVA90 and MEVA90-Climb. The training time for MEVA30-Climb was 17 hours and 57 minutes, although like before, it reached its best validation performance very early, after only 3 epochs. This model has the best OKS of all the MEVA models, even outperforming SPIN on the train split. However, in contrast to where MEVA90-Climb got a better ACC-ERR than MEVA90, MEVA30-Climb gets a worse ACC-ERR than MEVA30. It makes sense that MEVA30-Climb's ACC-ERR should be higher than MEVA90-Climb, since the shorter sequence length makes the intersection jitter more frequent, as shown in the acceleration error graph Fig. 4.16c. But this doesn't explain why the ACC-ERR increases from MEVA30 to MEVA30-Climb.



(a) Mean OKS. **(b) OKS per joint.**



(c) ACC-ERR. The curve is flatter than the one for MEVA30, but not as flat as that for MEVA90-Climb.

Figure 4.19.: MEVA30-Climb metrics. OKS and ACC-ERR for each frame in video IMG_2139, the video used for testing and validation. The OKS plots are smoothed with a rolling average over 180 frames for visibility.

(a) Frame #60 from video IMG_2315. **(b)** Frame #80 from video IMG_2315. **(c)** Frame #105 from video IMG_2315.

Figure 4.20.: MEVA30-Climb smoothing problem. The same examples as in Fig. 4.12. In (b), the model has also not yet moved into the split. And when it finally gets there, in (c), the legs are not fully extended, making the final fit worse than with MEVA90-Climb.

When looking at the rendered mesh videos, you can see many of the same improvements from MEVA30 to MEVA30-Climb as from MEVA90 to MEVA90-Climb. It learns to scale correctly, losing both the consistent underscaling and the vanishing scale problem. However, there is noticeably more jitter than in MEVA90-Climb, especially on the window intersections, which are clearly marked by a stutter almost every second. And even though it has the highest OKS scores of the MEVA models, this difference is not really noticeable when looking at the video. On the difficult poses in Fig. 4.21, it does not perform better than MEVA90-Climb. And the smoothing problem example in Fig. 4.20 it actually has lower accuracy.

(a) Frame #155 from video IMG_2306.

(b) Frame #75 from video IMG_2314. Like with MEVA90-Climb the model's right arm is no longer clipping through it's leg and the torso is bent forwards.

Figure 4.21.: MEVA30-Climb difficult poses. No noticeable differences compared to MEVA90-Climb.

(a) Frame #3765 from video IMG_2139.

(b) Frame #3770 from video IMG_2139.

(c) Frame #3775 from video IMG_2139.

Figure 4.22.: MEVA30-Climb rotation example. The same frames as with MEVA90-Climb in Fig. 4.14, but MEVA30-Climb gets them right and manages to correct the erroneous predictions from SPIN.

Discussion

5

5.1 Results

Based on the conducted experiments, it is clear that MEVA provides many benefits from its use of temporal information, discussed further in Sec. 5.3. The 3D meshes produced by MEVA90-Climb and MEVA30-Climb are both accurate and smooth and fulfill the requirements posed by the climbing analysis use case. However, there are still some errors, and the model should be improved further before putting it to use in a real-world scenario. These errors are obvious when looking at the rendered meshes, and it is therefore probably not yet good enough for a detailed comparative analysis between climbers' techniques. But it is probably accurate enough for more rough measures, like a center of gravity analysis, usable for comparing climbing efficiency. Avenues for model improvement are discussed later in this section and in Sec. 6.

We see that the model struggles to understand the depth of some limbs in the more difficult poses, like the "figure four" pose Fig. 4.13. Since the model is only trained with 2D climbing data, this is not surprising. MEVA's high 3D accuracy on 3D datasets like 3DPW indicates that if we trained it on 3D annotated climbing data, it could manage to learn the right 3D representation of difficult climbing poses as well.

The model's errors become especially prevalent when there is a considerable distance to the subject. The errors increase steadily with distance, and after the climber gets past about 10 meters off ground level, the usability of the model is very low. This is expected since it starts to get very hard for even a human standing on ground level at that distance to determine an accurate pose. For reference, a standard tall climbing route is about 15 meters, and a bouldering route is typically 4-5 meters tall.

5.2 Training SPIN.

SPIN's high accuracy on the test and validation sets in terms of OKS is impressive. Especially considering the fact that SPIN has not been trained on climbing data at all, like some of the other models. If we had trained SPIN as well, we would expect to see even better performance. SPIN's predictions are quite noisy, and training it on the climbing data would probably result in less noise, but since it does not use any temporal information, the level of noise would probably still be significant. However, the increase in accuracy from training could be greater than the decrease in noise. Looking at some of SPIN's typical errors, exemplified by Fig. 4.5b and Fig. 4.6c, we see that the model does not seem to use the full range of motion. In Fig. 4.5b, the hip joint of the right leg does not bend enough, leaving the whole right leg too low, and the rest of the model is tilted to compensate. In Fig. 4.6c, both legs are not fully extended, and it seems like the split-like configuration of the hip joints makes the knee joints unable to extend properly. A plausible explanation for this kind of error is that

such extreme poses are not well represented in SPIN's training data, and it, therefore, lacks the capacity to express them. This could be remedied by fine-tuning SPIN on the climbing data, like what I have done with the MEVA models. A fine-tuned SPIN model would, of course, also be beneficial for the MEVA models since they would get more accurate inputs as a starting point.

5.3 Benefits of temporal information

Noise reduction. Even though SPIN's predictions give a high mean accuracy, the noise is substantial, as evident when looking at the rendered mesh videos. If one were to use these predicted meshes for any visualization or most kinds of analysis required for this project's use-case, one would have to use some average filtering first to smooth out the results. Although I have not done any analysis of SPIN's performance with average filtering, such processing generally does not significantly increase accuracy. And even with sophisticated average filtering, it is not certain that we would achieve an acceleration error at the level of MEVA or how much we would "pay" in terms of accuracy to get there. MEVA's use of temporal information consists of a twofold strategy that ensures temporal constancy of motion through GRU's and compliance with a learned space of natural motion. The effect of the first could perhaps be approximated by average filtering, but the latter can not be. In other words, MEVA's learned motion space makes it capable of a sophisticated form of outlier detection, where outliers are defined in terms of temporal consistency as well as in terms of whether the motion is "natural". The "naturalness" of a motion is a quality that is not adequately captured by OKS or ACC-ERR but is noticeable by the human eye.

Accuracy. In addition to reducing acceleration error, there are also advantages to using temporal information for accuracy. Despite the fact that the MEVA models get a slightly worse OKS on the test and validation sets, it does improve the accuracy in some circumstances. As shown in Fig. 4.1, MEVA90-Climb outperforms SPIN on the hardest parts of the video, where the subject is far away. These are typically situations where all the models struggle to get the specifics of the pose right because the climber is far away, and there is considerable occlusion. But there are also simpler scenes where MEVA performs better. In Fig. 4.13b, we see that MEVA90-Climb manages to correctly infer that the model's torso should be bent forward so that the right arm is behind the knee instead of clipping through the leg like SPIN's prediction in Fig. 4.6b. The difference between these two predictions is not noticeable in 2D re-projection error, but we can see in the images that MEVA's prediction better matches the real 3D depth of the pose. This means that there are situations when there is depth information encoded in the temporal sequence that MEVA makes use of. So even though the best MEVA model achieved a worse 2D accuracy than SPIN, we see qualitative examples of MEVA getting a better 3D accuracy. Therefore, we can expect that if we had 3D ground truth annotations, we would see an increase in MEVA's 3D accuracy compared to SPIN's. And perhaps, if trained with 3D labels, we would see MEVA's advantage increase even further, giving results closer to MEVA90's 11.5% improvement compared to SPIN's accuracy for 3DPW-test[2], shown in Tab. 4.3.

5.4 MEVA errors

Over-smoothing. There is an inherent trade-off between smoothness and accuracy in any smoothing process, and it is, therefore, relevant to analyze how MEVA strikes a balance between the two and whether or not it smooths too much or too little. It is hard to find examples of over-smoothing since SPIN's predictions are so noisy and full of jitter. Analyzing the acceleration error reveals that... One example of what appears to be over-smoothing is in Fig. 4.12, where MEVA90-Climb fails to match the split pose until the next sequence when that pose is the dominant one. But further inspection reveals that this is actually the result of bad SPIN predictions, which fail to match the split pose in the first frames. In fact, MEVA90-Climb's prediction significantly improves on SPIN's by correcting which leg is raised, which SPIN initially thought was the left leg (see rendered mesh video IMG_2315 for each model to compare).

Under-smoothing. In the acceleration error graph for MEVA90-Climb Fig. 4.11c, there is a large spike around frame 3775. This is caused by an error in model rotation, as shown in Fig. 4.14. The reason for this is a sequence of bad SPIN predictions that MEVA90-Climb is unable to smooth out. Interestingly, MEVA30-Climb does not replicate the same error. As shown in Fig. 4.22, it manages to correct smooth the erroneous SPIN predictions. This is surprising because the model uses a shorter sequence length, which presumably would make the model more prone to under-smoothing errors. The reason MEVA30-Climb outperforms MEVA90-Climb on this example is unclear. A possible explanation is that the VAEs the models use are slightly different, VAE30 being trained on a slightly larger dataset.

How much does MEVA generalize? The pre-trained MEVA models do not transfer that well into the climbing domain. We see that they both make some errors. One of these, the consistent underscaling, is probably the result of a scaling problem in the training data and is known to the creator of the MEVA GitHub repository [26], so I will ignore this here. However, the other issues, namely the vanishing scale, erroneous rotation, and garbage sequences, are not explainable by the same issue. These errors must be caused by the models not being able to handle the SPIN features from this climbing dataset, i.e., not sufficiently general. This is not surprising since, as noted by the authors of several 3D pose estimation papers [1, 2], there is not enough 3D annotated data available to train models that are sufficiently generalized. We can, in turn, extend this logic to the MEVA models that are trained on the climbing data, MEVA90-Climb and MEVA30-Climb. We observe that they learn to adapt to the climbing data, and both lose the vanishing scale, erroneous rotation, and garbage sequence problems. However, even with these models, we should expect similar issues to re-appear if presented with new data in the climbing domain. Variations like lighting, background, climber appearance, and more, can be sufficient to introduce the necessary changes in input to confuse them, just like we saw with MEVA90 and MEVA30. My testing set is, after all, relatively small and is all from just one climbing video. More data, both for training and testing, is needed to reach a sufficient level of certainty about the models' general performance in the climbing domain.

OKS for head joints. In all the OKS-per-joint plots, we see that the head joints (eyes and ears) score significantly worse than the others. One reason for this is that the OKS

formulation allows for less variance in these points, so the same error in pixel distance will give a larger OKS penalty. Another reason is that, in this dataset, the keypoints in the climber's head are very often occluded. When ascending a tall route, the head is often not visible at all as the climber gets higher up. And even when the climber is close to ground-level, they face the wall most of the time, occluding their eyes. The original OKS includes a term for canceling out the penalty for occluded joints, but as explained in Sec. 3.5, I don't include this when I calculate OKS.

Difficult training set. In the metrics table Tab. 4.1, we see that all the models perform significantly worse on the training set compared to the others. The training set is different from the testing and validation sets in that most of it is annotated with weak labels from a 2D pose model, which means that many frames are mislabeled to some degree. The primary problem with such mislabelling is that there is a risk that the model learns to replicate these errors. This problem is mitigated by the fact that the 2D re-projection error for each joint is weighted by the confidence score the 2D model gives for that prediction. The fact that the models trained on this dataset perform better on the test set than those without it demonstrates that this method of weak supervision provides valuable learning. However, there is a second problem with using weak labels, namely that these predicted points are also used to generate bounding boxes for cropping input images. If one wrongly placed point is far away from the climber, the resulting bounding box will be way too large, even if the confidence score for this point is low. This is the issue discussed in Sec. 2.1.3, where I use a confidence threshold to discard the worst point predictions. However, superficial analysis of the acceleration in ground truth labels reveals that there are many parts of the training data where keypoints have an unnaturally high acceleration. This indicates that the labels are jumping around, possibly making the bounding boxes scale up or down erratically. This would make it much harder for SPIN to produce meaningful predictions and could explain why the metrics for this set are so poor, even for models that are trained on it. If this is the cause of the problem, a threshold on ground truth acceleration could help restrict the number of bad labels. However, further investigation should be made to confirm this theory. Instead of trying to minimize the number of bad labels in the training set, a better solution could be to improve the training set annotations overall. But this requires manually annotating all the training data, which is a sizeable task that should be undertaken with consideration.

The role of the shape parameter. An interesting difference between the models is how the shape of the SMPL [5] mesh changes. With SPIN [2], the mesh takes on a male-looking shape, while with the MEVA models, the mesh looks more female. Especially the models trained on the climbing data seem to have even more exaggerated female features. It seems like the models learn to tweak the shape to account for differences in the way the data are annotated. It could be that when data annotations have different spacing between hip or shoulder joints than what SMPL is trained with, causing the models to account for this difference by altering the shape parameter. If the shape of the subject is known beforehand, this could be given to the model as a set parameter. This would force the model to explain deviations from ground truth 2D keypoints by changing the pose instead, which could give better results and improve model training. SMPL provides learned shape space parameters, a neutral model trained with all the subjects in their dataset, as well as separate shape space parameters for male and

female subjects. The models in this project all use the neutral shape space, but if the subject's gender is known, this could be swapped out for a more customized shape space. Additionally, using alternative methods, like measuring height or the length of some limbs, to determine the subject's shape beforehand could result in a mesh that is more visually accurate.

5.5 Shorter sequence length.

One part of this project's objective was to investigate the effect of changing MEVA's sequence length parameter in the climbing domain. As discussed in the introduction Sec. 1, the hypothesis was that even though the authors of MEVA [1] found that a longer sequence length improved performance on 3DPW [23], this would not necessarily be the case in climbing due to the kind of motion involved in the domain. I investigated this by re-training a VAE for the human motion space and two versions of MEVA with a sequence length of 30 frames. When comparing the results of MEVA30 on 3DPW with that of MEVA90 (Tab. 4.3), we saw that MEVA30 performed worse, as expected. There are, however, some issues with this comparison. One is that MEVA30 and MEVA90 are not trained on exactly the same data. As mentioned in Sec. 4.5, MEVA30 is not trained on Human3.6M [53], which puts MEVA30 at a disadvantage. Another issue is that we do not know whether the difference in performance is due to the parts of MEVA that are trained jointly or due to the human motion subspace that is trained separately by VAE90 and VAE30. Since I don't compare VAE30 and VAE90 directly, we do not know how well the learned latent space is able to represent the input data. An approach to investigate this could be to compare the VAE models' reconstruction error and decoded smoothness on, for example, 3DPW-test.

Since I keep the size of the latent space representation constant while decreasing the sequence length, it could be that VAE30 does not need to compress the motion as much as VAE90, and therefore loses some of its value to MEVA as a motion smoothing mechanism. All of this should be investigated further before one can confirm or deny the hypothesis about shorter sequence lengths in the climbing domain.

When comparing the two best MEVA models, MEVA90-Climb and MEVA30-Climb, we see that the longer sequence length gives more smoothness, as demonstrated by a lower ACC-ERR. But in terms of accuracy, the two get nearly identical OKS measures. From the results in this project, the question of sequence length has not been sufficiently answered. Improving the models, like having more accurate SPIN features as input or adjusting the size of the latent space, could make either one excel more than the other, and further inquiry is necessary.

5.6 Training MEVA.

Some parts of the MEVA models' training on climbing data are sub-optimal. Since we only have 2D annotated data, and train with 2D reprojection loss, the amount of depth information that can be learned is minimal. 2D annotation carries very little depth information, although some can be gleaned from 2D annotations over time which

MEVA does make use of, as exemplified by Fig. 4.13b. Using reprojection loss is also a relatively weak form of supervision since it forces the model to search in a large parameter space to explain the 2D variation. This is discussing in the SPIN paper [2], and is their reason for using the optimization technique SMPLify [11] to first fit an SMPL model to 2D points and then use the resulting model to supervise the rest of the network with stronger 3D and SMPL loss functions. This is a more efficient way of using 2D annotations for training since it limits the search space during training. Even though the MEVA model, inspired by SPIN, does include an iterative optimization as its last step, it still relies solely on reprojection loss. This clever use of 2D annotation is the SPIN paper's greatest innovation, which MEVA could benefit greatly from. Other, more specific details on improving training are discussed in Sec. 6.

5.7 Applying MEVA

When applying MEVA in a real-world scenario, one must really think of it as applying SPIN+MEVA. SPIN is a pure per-frame model operating on images, and MEVA is a temporal model operating on features.

In terms of runtime during inference, SPIN dominates quite clearly, as evident in Tab. 4.2. Although when running SPIN, there were time-consuming I/O operations that were much more optimized for MEVA, so the comparison is not completely fair. When running SPIN, I fetched input images from a server over the network, while when running MEVA, its input features were pre-loaded in memory. With more optimized I/O, one could easily imagine a SPIN fps reaching about 30 on the same hardware, or perhaps even more.

Since MEVA requires a sequence as input, there would be some waiting time involved in a real-time setting. At least T frames would first have to be processed by SPIN before one sequence could be processed by MEVA. Also, if one would like to use overlapping windows to alleviate the jitter on window intersections, there would be an additional waiting time for processing the overlap. But, given a SPIN model with a runtime fps of more than 30. With an overlap of O frames, one could imagine a near-real-time setup, where after an initial wait time of $T + O$ frames could process sequences at approximately real-time speed but with a delay of $\frac{T+O}{30}$ seconds (if the input video has 30 fps).

In some circumstances, it could be that SPIN's runtime is too slow. In that case, a smaller version of SPIN with fewer layers could be trained, and a new MEVA model could be trained on its features. This would involve a decrease in performance, but that might be worth the cost, depending on the circumstance. Also, since 3D pose estimation is an active field of research, it is possible that new per-frame models will keep cropping up in the near future with better performance than SPIN. Then combining that model with MEVA could yield even better results.

Future Work

6

Here, I will detail my thoughts on what future work could be done. Some are low-hanging fruits that could be implemented quickly to improve performance, and others require more effort and further investigation.

6.1 MEVA

Sliding window. As discussed in Sec. 4, all the MEVA models process video sequences without any overlap. This results in varying degrees of jitter at the window intersections. A solution to this problem is to use a sliding window with some overlap of O frames. The setting of O would have to be found experimentally, but setting $O = \frac{1}{3}T$ would probably be sufficient to mitigate most of the jitter issues. With this setting, the sliding window would move with a stride of $T - O = T - \frac{1}{3}T = \frac{2}{3}T$, and every frame would be processed three times, once in the start, once in the center, and once at the end of a sequence. Then, the three predictions for each frame would have to be combined. The simplest way to do this is by flat averaging, but since we know that errors occur at the edges of a sequence, a better approach would be a weighted average with lower weights near the edges, using, for example, a Gaussian.

Training details. There are some smaller issues with MEVA's training that could be improved.

1. We see that all the MEVA models converge very early in the training process, shown in Fig. 4.10 and Fig. 4.18. This is non-ideal since if the best-performing model was found after epoch 4, it could be that an even better model occurred in the middle of epoch 4 or somewhere in epoch 5. In my training configuration, an epoch is set to process 500 batches. Decreasing this number, to say 100, would increase the fidelity of the search for the best model. And even if the best model was found after the same amount of batches, $4 \times 500 = 2000$ batches, giving $2000/100 = 20$ epochs instead of 4, it would still be well within the total training duration of 100 epochs.
2. The consistent under-scaling problem found in found with MEVA90 and MEVA30 is probably due to some scaling error in the pre-processing of one or more of the datasets they are trained on. This is not a very large problem, as the models quickly learn to correct this when fine-tuning on the climbing data. But it does probably contribute to some small errors and should be corrected. As mentioned before, this problem is known to the creator of the MEVA GitHub repository [26], so one could either fix it oneself or wait for a fix from the repository maintainer.
3. Since MEVA30 is trained without Human3.6M [53], it is at a disadvantage compared to MEVA90. This can be remedied by implementing the proper pre-processing to use Human3.6M for training or waiting for this to be released on the MEVA GitHub repository [26].

6.2 Initial single-frame features

Since MEVA takes per-frame SPIN features as input, improving the quality of these features will, in turn, improve MEVA's. An obvious way to improve SPIN's performance on the climbing data would be to fine-tune it as I do with MEVA90-Climb and MEVA30-Climb. This would improve the quality of MEVA's input features, and it could be that this could alleviate some of the problems that MEVA90 and MEVA30 had, like vanishing scale, sudden rotation, and garbage sequences. As the field of 3D pose estimation matures, it is likely that even better per-frame models crop up, and using features from these models as input for MEVA could give even better performance. At the time of writing, a new per-frame model has just jumped to the top of the 3DPW leaderboard [24]. This model is called ROMP [58] and could be investigated for this purpose.

6.3 Data

As discussed in Sec. 5.6, more noise 2D annotations could be filtered out using something like a ground truth acceleration threshold. Alternatively, the training data could be manually annotated to improve annotation quality. This would probably require some sort of crowdsourcing to accomplish, especially if one would want to extend the amount of training data even further. Since MEVA uses many different datasets for training, a common keypoint format is used for annotations. This format does not include all the COCO keypoints my training data is annotated with. This means that we do not get as much training value out of the annotation as we could have. This could be improved by extracting and calculated loss for all the annotated joints, which can be regressed from the SMPL [5] model.

If one were to stick to automatic 2D annotation, a method of extending the amount of training data would be to use videos of the internet, like in InstaVariety [16]. YouTube has thousands of hours of climbing content, with an inordinate amount of variation in setting, lighting, climber appearance, etc. The only limitation here would be in licensed use, which might restrict the applicability of this strategy for private companies.

Of course, 3D annotated climbing data would be even better for training a 3D model. A good motion capture setup for climbing might be difficult to achieve since the climbing wall obstructs any camera views to only half of all possible angles. But it could be possible to achieve using IMU's, like how the authors of 3DPW [23] did when building their dataset.

In this project, I did not perform any data augmentation on the climbing data. This could be useful for extending the amount of data, especially if one also trains SPIN. The hardest parts of the data are typically where the climber is far away from the camera. This scenario might be simulated with data augmentation, like downsampling the input image. There are, however, limitations to how much this would help since part of what makes the long-distance scenario difficult is the viewing angle from below and the subsequent occlusion of the head and arms, which cannot be imitated by data augmentation.

6.4 VAE

Further inquiry should be made into the VAE trained on human motion. In this project, I retrained a VAE with a shorter sequence length, but I did not investigate the effect this change had on the recreation error compared to the original model. Also, I kept the size of the latent space the same, even though I decreased the sequence length, which means that VAE30 has more space to encode motion information than VAE90 had. This means that VAE30 compresses motion information less than VAE90 does. The effect of this and the relation between sequence length and the size of the latent space should be investigated further, as the VAE's smooth motion latent space plays a key role in MEVA.

How well the VAE generalizes to unseen climbing motions should be investigated as well. The VAE models are trained on AMASS [35], which despite being quite large, does not include climbing motions. Extending VAE training with climbing data would ensure that the learned latent space can correctly represent climbing motions. But this would require 3D labeled climbing data, so it could be considered more of a long-term goal.

6.5 Miscellaneous

Known shape. In the use-case of climbing analysis, one could know the shape of the climber beforehand. The SMPL shape parameter could be estimated based on the climber's gender, height, and measurements. Then, the MEVA model could be modified to take a shape parameter as part of its input. This could help the model achieve better accuracy by restricting the number of parameters to fit. Alternatively, the predicted shape could be exchanged for the known shape after inference for visualization purposes.

Route information. Even though we have a 3D model of the climber, we still lack information about the climber's position in relation to the route being climbed. This information is necessary to compare how different climbers ascend a route or compare multiple runs by a single climber. To place the climber in a coordinate system in relation to the route, one could use the positions of the grips. We know that once climbing, the climber will most of the time have hands and feet placed on grips or structural features of the wall. If we knew the position of these grips and features beforehand and used a separate model to detect them in the input image, we could align them with the climber model so that they match up. This would require both creating a 3D model of a route with grips and other necessary features and a model to detect these grips and features while a climber climbs the route. This same information could also be used to improve the 3D model of the climber. Since we know that the climber cannot be soaring in mid-air unless they are jumping or falling, we could use locations of grips to inform the 3D of likely locations for hands and feet. This information could be incorporated into the 3D pose estimation model in several ways, as a per-frame model optimization step, as additional input to a temporal model, or to extend 2D annotation with pseudo-depth labels, to name a few.

Conclusion

7

In this work, I have shown how the successful 3D pose estimators MEVA [1] and SPIN [2] transfer to the domain of sports climbing. I compare the performance of the pre-trained models MEVA90 and SPIN with a fine-tuned model MEVA90-Climb and two models with a shorter sequence length, MEVA30, and MEVA30-Climb. For this, I annotate a brand new dataset with 119 videos of climbers climbing outdoor and indoor routes, on which I train and evaluate the models. The results of this comparison confirm the finding of the MEVA paper [1], that a temporal model, using a learned latent space of human motion to smooth motion sequences, can improve the results of a per-frame estimator.

However, the 2D annotation of the dataset restricts the analysis to 2D metrics, in which all the MEVA models achieve a worse mean accuracy on the test set than SPIN does. Despite this, we see qualitative examples where MEVA models improve the 3D accuracy of predictions, which is not captured by the metrics. This implies that given 3D annotations of the climbing data, we would see MEVA achieve better 3D accuracy as we see for the 3D dataset 3DPW [23]. Although, in terms of smoothness, MEVA firmly out-competes SPIN, giving more natural-looking motions.

The question about the effect of shortening sequence length largely remains unanswered. The models MEVA90-Climb and MEVA30-Climb perform almost identically, which, along with the varying factors in their construction, warrants further investigation into the problem.

As it is, MEVA90-Climb achieves the best combination of accuracy and smoothness suitable for the climbing use-case, with an OKS of 0.922 and ACC-ERR of 0.0051 on my test set. However, it should be improved even further before applying it in real-world scenarios.

Bibliography

8

[1] Zhengyi Luo, S. Alireza Golestaneh, and Kris M. Kitani. „3D Human Motion Estimation via Motion Compression and Refinement“. In: *Proceedings of the Asian Conference on Computer Vision (ACCV)*. Nov. 2020.

[2] Nikos Kolotouros, Georgios Pavlakos, Michael J. Black, and Kostas Daniilidis. „Learning to Reconstruct 3D Human Pose and Shape via Model-Fitting in the Loop“. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2019.

[3] IOC: The International Olympic Committee. *MMTracking: OpenMMLab video perception toolbox and benchmark*. https://olympics.com/ioc/news/ioc-approves-five-new-sports-for-olympic-games-tokyo-2020. Accessed: 2021-05-26.

[4] Wikipedia Contributors. *2020 Summer Olympics*. https://en.wikipedia.org/wiki/2020_Summer_Olympics. Accessed: 2021-05-26.

[5] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. „SMPL: A Skinned Multi-Person Linear Model“. In: *ACM Trans. Graphics (Proc. SIGGRAPH Asia)* 34.6 (Oct. 2015), 248:1–248:16.

[6] David A. Hirshberg, Matthew Loper, Eric Rachlin, and Michael J. Black. „Coregistration: Simultaneous Alignment and Modeling of Articulated 3D Shape“. In: *Computer Vision – ECCV 2012*. Ed. by Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 242–255.

[7] Georgios Pavlakos, Vasileios Choutas, Nima Ghorbani, Timo Bolkart, Ahmed A. A. Osman, Dimitrios Tzionas, and Michael J. Black. „Expressive Body Capture: 3D Hands, Face, and Body From a Single Image“. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[8] Kristen Grauman, Gregory Shakhnarovich, and Trevor Darrell. „Inferring 3D Structure with a Statistical Image-Based Shape Model.“ In: *ICCV*. Vol. 3. 2003, p. 641.

[9] Ankur Agarwal and Bill Triggs. „Recovering 3D human pose from monocular images“. In: *IEEE transactions on pattern analysis and machine intelligence* 28.1 (2005), pp. 44–58.

[10] Peng Guan, Alexander Weiss, Alexandru O Balan, and Michael J Black. „Estimating human shape and pose from a single image“. In: *2009 IEEE 12th International Conference on Computer Vision*. IEEE. 2009, pp. 1381–1388.

[11] Federica Bogo, Angjoo Kanazawa, Christoph Lassner, Peter Gehler, Javier Romero, and Michael J. Black. „Keep It SMPL: Automatic Estimation of 3D Human Pose and Shape from a Single Image“. In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling. Cham: Springer International Publishing, 2016, pp. 561–578.

[12] Angjoo Kanazawa, Michael J. Black, David W. Jacobs, and Jitendra Malik. „End-to-End Recovery of Human Shape and Pose“. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

[13] Mohamed Omran, Christoph Lassner, Gerard Pons-Moll, Peter Gehler, and Bernt Schiele. „Neural body fitting: Unifying deep learning and model based human pose and shape estimation“. In: *2018 international conference on 3D vision (3DV)*. IEEE. 2018, pp. 484–494.

[14] Hsiao-Yu Fish Tung, Hsiao-Wei Tung, Ersin Yumer, and Katerina Fragkiadaki. „Self-supervised learning of motion capture“. In: *arXiv preprint arXiv:1712.01337* (2017).

[15] Mir Rayat Imtiaz Hossain and James J. Little. „Exploiting temporal information for 3D human pose estimation“. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Sept. 2018.

[16] Angjoo Kanazawa, Jason Y. Zhang, Panna Felsen, and Jitendra Malik. „Learning 3D Human Dynamics From Video“. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[17] Hongsuk Choi, Gyeongsik Moon, and Kyoung Mu Lee. „Pose2Mesh: Graph convolutional network for 3D human pose and mesh recovery from a 2D human pose“. In: *European Conference on Computer Vision*. Springer. 2020, pp. 769–787.

[18] Yujun Cai, Liuhao Ge, Jun Liu, Jianfei Cai, Tat-Jen Cham, Junsong Yuan, and Nadia Magnenat Thalmann. „Exploiting Spatial-Temporal Relationships for 3D Pose Estimation via Graph Convolutional Networks“. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2019.

[19] Dario Pavllo, Christoph Feichtenhofer, David Grangier, and Michael Auli. „3d human pose estimation in video with temporal convolutions and semi-supervised training“. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 7753–7762.

[20] Rishabh Dabral, Anurag Mundhada, Uday Kusupati, Safeer Afaque, Abhishek Sharma, and Arjun Jain. „Learning 3d human pose from structure and motion“. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 668–683.

[21] Dushyant Mehta, Srinath Sridhar, Oleksandr Sotnychenko, Helge Rhodin, Mohammad Shafiei, Hans-Peter Seidel, Weipeng Xu, Dan Casas, and Christian Theobalt. „Vnect: Real-time 3d human pose estimation with a single rgb camera“. In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), pp. 1–14.

[22] Dushyant Mehta, Oleksandr Sotnychenko, Franziska Mueller, Weipeng Xu, Mohamed Elgharib, Pascal Fua, Hans-Peter Seidel, Helge Rhodin, Gerard Pons-Moll, and Christian Theobalt. „Xnect: Real-time multi-person 3d human pose estimation with a single rgb camera“. In: *arXiv preprint arXiv:1907.00837* (2019).

[23] Timo von Marcard, Roberto Henschel, Michael Black, Bodo Rosenhahn, and Gerard Pons-Moll. „Recovering Accurate 3D Human Pose in The Wild Using IMUs and a Moving Camera“. In: *European Conference on Computer Vision (ECCV)*. Sept. 2018.

[24] *Leaderboard on the 3DPW test set*. https://paperswithcode.com/sota/3d-human-pose-estimation-on-3dpw. Accessed: 2021-05-20.

[25] Muhammed Kocabas, Nikos Athanasiou, and Michael J. Black. „VIBE: Video Inference for Human Body Pose and Shape Estimation“. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[26] *Official GitHub page for MEVA*. https://github.com/ZhengyiLuo/MEVA. Accessed: 2021-05-16.

[27] *Official GitHub page for VIBE*. https://github.com/mkocabas/VIBE. Accessed: 2021-05-26.

[28] *Official GitHub page for SPIN*. https://github.com/nkolot/SPIN. Accessed: 2021-05-16.

[29] Feng Zhang, Xiatian Zhu, Hanbin Dai, Mao Ye, and Ce Zhu. „Distribution-aware coordinate representation for human pose estimation“. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 7093–7102.

[30] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV].

[31] MMPose Contributors. *OpenMMLab Pose Estimation Toolbox and Benchmark*. https://github.com/open-mmlab/mmpose. 2020.

[32] Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. „2D Human Pose Estimation: New Benchmark and State of the Art Analysis“. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2014.

[33] *Leaderboard on the COCO keypoints task*. https://cocodataset.org/#keypoints-leaderboard. Accessed: 2021-05-04.

[34] *Computer Vision Annotation Tool (CVAT)*. https://github.com/openvinotoolkit/cvat. Accessed: 2021-04-29.

[35] Naureen Mahmood, Nima Ghorbani, Nikolaus F. Troje, Gerard Pons-Moll, and Michael J. Black. „AMASS: Archive of Motion Capture as Surface Shapes“. In: *International Conference on Computer Vision*. Oct. 2019, pp. 5442–5451.

[36] Dushyant Mehta, Helge Rhodin, Dan Casas, Pascal Fua, Oleksandr Sotnychenko, Weipeng Xu, and Christian Theobalt. „Monocular 3D Human Pose Estimation in the Wild Using Improved CNN Supervision“. In: *2017 International Conference on 3D Vision (3DV)*. 2017, pp. 506–516.

[37] Weiyu Zhang, Menglong Zhu, and Konstantinos G. Derpanis. „From Actemes to Action: A Strongly-Supervised Representation for Detailed Action Understanding“. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Dec. 2013.

[38] Federica Bogo, Javier Romero, Matthew Loper, and Michael J. Black. „FAUST: Dataset and Evaluation for 3D Mesh Registration“. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2014.

[39] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. „On the Continuity of Rotation Representations in Neural Networks“. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. „Identity mappings in deep residual networks“. In: *European conference on computer vision*. Springer. 2016, pp. 630–645.

[41] N. Silberman and S. Guadarrama. *TensorFlow-Slim image classification model library*. https://github.com/tensorflow/models/tree/master/research/slim. 2016.

[42] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. „OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields“. In: *IEEE transactions on pattern analysis and machine intelligence* 43.1 (2019), pp. 172–186.

[43] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. „Realtime multi-person 2d pose estimation using part affinity fields“. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7291–7299.

[44] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. „Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation“. In: *CoRR* abs/1406.1078 (2014). arXiv: 1406.1078.

[45] Diederik P Kingma and Max Welling. „Auto-encoding variational bayes“. In: *arXiv preprint arXiv:1312.6114* (2013).

[46] Jacob Walker, Kenneth Marino, Abhinav Gupta, and Martial Hebert. „The pose knows: Video forecasting by generating pose futures“. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 3332–3341.

[47] Zhenyi Wang, Ping Yu, Yang Zhao, Ruiyi Zhang, Yufan Zhou, Junsong Yuan, and Changyou Chen. „Learning diverse stochastic human-action generators by learning smooth latent transitions“. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 07. 2020, pp. 12281–12288.

[48] Ye Yuan and Kris Kitani. „Diverse trajectory forecasting with determinantal point processes“. In: *arXiv preprint arXiv:1907.04967* (2019).

[49] Jacob Walker, Carl Doersch, Abhinav Gupta, and Martial Hebert. „An uncertain future: Forecasting from static images using variational autoencoders“. In: *European Conference on Computer Vision*. Springer. 2016, pp. 835–851.

[50] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. „Extracting and composing robust features with denoising autoencoders“. In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 1096–1103.

[51] Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. „Generalized denoising auto-encoders as generative models“. In: *arXiv preprint arXiv:1305.6663* (2013).

[52] *68-95-99.7 rule*. https://en.wikipedia.org/wiki/68%E2%80%9395%E2%80%9399.7_rule. Accessed: 2021-05-27.

[53] Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. „Human3.6M: Large Scale Datasets and Predictive Methods for 3D Human Sensing in Natural Environments“. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2014).

[54] Sam Johnson and Mark Everingham. „Clustered Pose and Nonlinear Appearance Models for Human Pose Estimation.“ In: *bmvc*. Vol. 2. 4. Citeseer. 2010, p. 5.

[55] Sam Johnson and Mark Everingham. „Learning effective human pose estimation from inaccurate annotation“. In: *CVPR 2011*. 2011, pp. 1465–1472.

[56] Philipp Bergmann, Tim Meinhardt, and Laura Leal-Taixe. „Tracking without bells and whistles“. In: *Proceedings of the IEEE international conference on computer vision*. 2019, pp. 941–951.

[57] MMTracking Contributors. *MMTracking: OpenMMLab video perception toolbox and benchmark*. https://github.com/open-mmlab/mmtracking. 2020.

[58] Yu Sun, Qian Bao, Wu Liu, Yili Fu, and Tao Mei. „CenterHMR: a Bottom-up Single-shot Method for Multi-person 3D Mesh Recovery from a Single Image“. In: *CoRR* abs/2008.12272 (2020). arXiv: 2008.12272.

Video Info

A

Video Name	Resolution	Frames	Climber	Route	Location
IMG_2139	720x1280	6829	A	A	Outside
IMG_2140	720x1280	7609	B	A	Outside
IMG_2141	720x1280	3481	B	B	Outside
IMG_2142	720x1280	2607	A	B	Outside
IMG_2320	1080x1920	7580	C	C	Inside
VID_20210123_091729	2160x3840	11361	D	A	Outside
VID_20210123_104706	2160x3840	15515	B	D	Outside
VID_20210123_110129	1080x1920	4647	D	D	Outside
VID_20210123_111337	2160x3840	3760	B	E	Outside
VID_20210123_111921	2160x3840	4538	D	E	Outside
IMG_2272	1080x1920	427	C	F	Inside
IMG_2273	1080x1920	328	C	F	Inside
IMG_2274	1080x1920	358	C	F	Inside
IMG_2275	1080x1920	324	C	F	Inside
IMG_2276	1080x1920	372	C	F	Inside
IMG_2277	1080x1920	283	C	F	Inside
IMG_2278	1080x1920	331	C	F	Inside
IMG_2279	1080x1920	242	C	F	Inside
IMG_2280	1080x1920	653	C	F	Inside
IMG_2283	1080x1920	1479	C	G	Inside
IMG_2284	1080x1920	1524	C	G	Inside
IMG_2285	1080x1920	1226	C	H	Inside
IMG_2286	1080x1920	1683	C	H	Inside
IMG_2287	1080x1920	277	A	I	Inside
IMG_2288	1080x1920	614	A	I	Inside
IMG_2289	1080x1920	235	A	I	Inside
IMG_2290	1080x1920	431	A	J	Inside
IMG_2292	1080x1920	1509	C	K	Inside
IMG_2293	1080x1920	2072	C	K	Inside
IMG_2294	1080x1920	621	A	L	Inside
IMG_2295	1080x1920	6163	C	M	Inside
IMG_2296	1080x1920	6554	A	N	Inside
IMG_2297	1080x1920	911	C	O	Inside
IMG_2298	1080x1920	772	C	O	Inside
IMG_2299	1080x1920	780	C	P	Inside
IMG_2300	1080x1920	304	C	P	Inside
IMG_2301	1080x1920	957	C	Q	Inside
IMG_2302	1080x1920	630	C	F	Inside
IMG_2303	1080x1920	1013	C	F	Inside
IMG_2304	1080x1920	736	A	K	Inside

IMG_2305	1080x1920	731	A	K	Inside
IMG_2306	1080x1920	655	C	R	Inside
IMG_2307	1080x1920	724	C	R	Inside
IMG_2309	1080x1920	947	C	S	Inside
IMG_2310	1080x1920	674	C	S	Inside
IMG_2311	1080x1920	1668	C	S	Inside
IMG_2312	1080x1920	702	A	T	Inside
IMG_2313	1080x1920	679	A	T	Inside
IMG_2314	1080x1920	1316	C	G	Inside
IMG_2315	1080x1920	261	A	U	Inside
IMG_2317	1080x1920	353	A	K	Inside
IMG_2319	1080x1920	216	A	K	Inside
IMG_2321	1080x1920	4976	A	V	Inside
IMG_2322	1080x1920	230	A	W	Inside
VID_20210414_140919	1080x1920	453	C	F	Inside
VID_20210414_140936	1080x1920	489	C	F	Inside
VID_20210414_140957	1080x1920	391	C	F	Inside
VID_20210414_141014	1080x1920	284	C	F	Inside
VID_20210414_141026	1080x1920	416	C	F	Inside
VID_20210414_141044	1080x1920	259	C	F	Inside
VID_20210414_141055	1080x1920	341	C	F	Inside
VID_20210414_141111	1080x1920	240	C	F	Inside
VID_20210414_141122	1080x1920	292	C	F	Inside
VID_20210414_141135	1080x1920	283	C	F	Inside
VID_20210414_141444	1080x1920	163	C	G	Inside
VID_20210414_141452	1080x1920	392	C	G	Inside
VID_20210414_141507	1080x1920	211	C	G	Inside
VID_20210414_141517	1080x1920	241	C	G	Inside
VID_20210414_141528	1080x1920	302	C	G	Inside
VID_20210414_141541	1080x1920	294	C	G	Inside
VID_20210414_141555	1080x1920	258	C	G	Inside
VID_20210414_141606	1080x1920	332	C	G	Inside
VID_20210414_141620	1080x1920	297	C	G	Inside
VID_20210414_141633	1080x1920	388	C	G	Inside
VID_20210414_142043	1080x1920	304	C	F	Inside
VID_20210414_142055	1080x1920	420	C	F	Inside
VID_20210414_142111	1080x1920	391	C	F	Inside
VID_20210414_142126	1080x1920	343	C	F	Inside
VID_20210414_142141	1080x1920	337	C	F	Inside
VID_20210414_142156	1080x1920	290	C	F	Inside
VID_20210414_142208	1080x1920	441	C	F	Inside
VID_20210414_143311	1080x1920	283	C	K	Inside
VID_20210414_143324	1080x1920	282	C	K	Inside
VID_20210414_143335	1080x1920	288	C	K	Inside
VID_20210414_143347	1080x1920	278	C	K	Inside
VID_20210414_143413	1080x1920	194	C	K	Inside
VID_20210414_143421	1080x1920	195	C	K	Inside

VID_20210414_143432	1080x1920	312	C	K	Inside
VID_20210414_143445	1080x1920	256	C	K	Inside
VID_20210414_143458	1080x1920	228	C	K	Inside
VID_20210414_143510	1080x1920	230	C	K	Inside
VID_20210414_144259	1080x1920	5292	C	M	Inside
VID_20210414_145257	1080x1920	224	C	O	Inside
VID_20210414_145307	1080x1920	185	C	O	Inside
VID_20210414_145333	1080x1920	161	C	O	Inside
VID_20210414_145341	1080x1920	197	C	O	Inside
VID_20210414_145352	1080x1920	145	C	O	Inside
VID_20210414_145509	1080x1920	201	C	P	Inside
VID_20210414_145520	1080x1920	178	C	P	Inside
VID_20210414_145530	1080x1920	184	C	P	Inside
VID_20210414_145542	1080x1920	206	C	P	Inside
VID_20210414_145615	1080x1920	270	C	Q	Inside
VID_20210414_145628	1080x1920	229	C	Q	Inside
VID_20210414_145640	1080x1920	224	C	Q	Inside
VID_20210414_145817	1080x1920	96	C	F	Inside
VID_20210414_145824	1080x1920	113	C	F	Inside
VID_20210414_145829	1080x1920	165	C	F	Inside
VID_20210414_145847	1080x1920	155	C	F	Inside
VID_20210414_145856	1080x1920	191	C	F	Inside
VID_20210414_145905	1080x1920	133	C	F	Inside
VID_20210414_150208	1080x1920	576	C	R	Inside
VID_20210414_150244	1080x1920	595	C	R	Inside
VID_20210414_150417	1080x1920	352	C	S	Inside
VID_20210414_150430	1080x1920	94	C	S	Inside
VID_20210414_150434	1080x1920	1168	C	S	Inside
VID_20210414_150555	1080x1920	1583	C	S	Inside
VID_20210414_151827	1080x1920	1282	C	G	Inside
VID_20210414_152114	1080x1920	101	C	K	Inside
VID_20210414_153226	1080x1920	7441	C	C	Inside

Table A.1.: The combinations of climber, route, and resolution for each video in the dataset. The routes are categorized a bit loosely. In climbing, a route is typically defined as a set of holds (almost always of the same color) with a defined start and finish. Here, videos labelled with the same route does not necessarily have the climber doing the same exact route, but rather climbing in roughly the same section of the wall.

MEVA Model Printout

B

```
MEVA(
  (vae_model): VAErecV2(
    (e_rnn): RNN(
      (rnn_f): GRUCell(144, 256)
      (rnn_b): GRUCell(144, 256)
    )
    (e_mlp): MLP(
      (affine_layers): ModuleList(
        (0): Linear(in_features=512, out_features=1024, bias=True)
        (1): Linear(in_features=1024, out_features=512, bias=True)
      )
    )
    (e_mu): Linear(in_features=512, out_features=512, bias=True)
    (e_logvar): Linear(in_features=512, out_features=512, bias=True)
    (d_rnn): RNN(
      (rnn_f): GRUCell(656, 512)
    )
    (d_mlp): MLP(
      (affine_layers): ModuleList(
        (0): Linear(in_features=512, out_features=1024, bias=True)
        (1): Linear(in_features=1024, out_features=512, bias=True)
      )
    )
    (d_out): Linear(in_features=512, out_features=144, bias=True)
    (init_pose_mlp): MLP(
      (affine_layers): ModuleList(
        (0): Linear(in_features=512, out_features=1024, bias=True)
        (1): Linear(in_features=1024, out_features=512, bias=True)
      )
    )
    (init_pose_out): Linear(in_features=512, out_features=144,
    ↪  bias=True)
  )
  (feat_encoder): TemporalEncoder(
    (gru): GRU(2048, 1024, num_layers=2, bidirectional=True)
    (linear): Linear(in_features=2048, out_features=2048, bias=True)
  )
  (motion_encoder): TemporalEncoder(
    (gru): GRU(2048, 512, num_layers=2, bidirectional=True)
    (linear): Linear(in_features=1024, out_features=512, bias=True)
  )
```

```
  (vae_init_mlp): Sequential(
    (0): Linear(in_features=512, out_features=256, bias=True)
    (1): Tanh()
    (2): Linear(in_features=256, out_features=144, bias=True)
  )
  (regressor): Regressor(
    (fc1): Linear(in_features=2205, out_features=1024, bias=True)
    (drop1): Dropout(p=0.5, inplace=False)
    (fc2): Linear(in_features=1024, out_features=1024, bias=True)
    (drop2): Dropout(p=0.5, inplace=False)
    (decpose): Linear(in_features=1024, out_features=144, bias=True)
    (decshape): Linear(in_features=1024, out_features=10, bias=True)
    (deccam): Linear(in_features=1024, out_features=3, bias=True)
    (smpl): SMPL(
      Gender: NEUTRAL
      Number of joints: 24
      Betas: 10
      (vertex_joint_selector): VertexJointSelector()
    )
  )
  (smpl): SMPL(
    Gender: NEUTRAL
    Number of joints: 24
    Betas: 10
    (vertex_joint_selector): VertexJointSelector()
  )
)
```

SPIN Model Printout

C

```
SPIN(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3,
  ↪  3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
  ↪  track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
  ↪  ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1),
      ↪  bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
      ↪  track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
      ↪  padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
      ↪  track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1),
      ↪  bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
      ↪  track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1),
        ↪  bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
        ↪  track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1),
      ↪  bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
      ↪  track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
      ↪  padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
      ↪  track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1),
      ↪  bias=False)
```

```
    (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (2): Bottleneck(
    (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
    ↪  padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
)
(layer2): Sequential(
  (0): Bottleneck(
    (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2),
    ↪  padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (relu): ReLU(inplace=True)
    (downsample): Sequential(
      (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2),
      ↪  bias=False)
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
      ↪  track_running_stats=True)
    )
  )
  (1): Bottleneck(
    (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
```

```
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
    ↪  padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (2): Bottleneck(
    (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
    ↪  padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (3): Bottleneck(
    (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
    ↪  padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
)
(layer3): Sequential(
  (0): Bottleneck(
    (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
```

```
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2),
    ↪  padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (relu): ReLU(inplace=True)
    (downsample): Sequential(
      (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2),
      ↪  bias=False)
      (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
      ↪  track_running_stats=True)
    )
  )
  (1): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
    ↪  padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (2): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
    ↪  padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (3): Bottleneck(
```

```
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
    ↪  padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (4): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
    ↪  padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (5): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
    ↪  padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
)
(layer4): Sequential(
  (0): Bottleneck(
```

```
    (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2),
    ↪  padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (relu): ReLU(inplace=True)
    (downsample): Sequential(
      (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2),
      ↪  bias=False)
      (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
      ↪  track_running_stats=True)
    )
  )
  (1): Bottleneck(
    (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
    ↪  padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (relu): ReLU(inplace=True)
  )
  (2): Bottleneck(
    (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
    (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
    ↪  padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
    ↪  track_running_stats=True)
    (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1),
    ↪  bias=False)
```

```
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
      ↪  track_running_stats=True)
      (relu): ReLU(inplace=True)
    )
  )
  (avgpool): AvgPool2d(kernel_size=7, stride=1, padding=0)
  (fc1): Linear(in_features=2205, out_features=1024, bias=True)
  (drop1): Dropout(p=0.5, inplace=False)
  (fc2): Linear(in_features=1024, out_features=1024, bias=True)
  (drop2): Dropout(p=0.5, inplace=False)
  (decpose): Linear(in_features=1024, out_features=144, bias=True)
  (decshape): Linear(in_features=1024, out_features=10, bias=True)
  (deccam): Linear(in_features=1024, out_features=3, bias=True)
  (smpl): SMPL(
    Gender: NEUTRAL
    Number of joints: 24
    Betas: 10
    (vertex_joint_selector): VertexJointSelector()
  )
)
```