Monocular Pose Estimation for VR Motion Capture Applications

Nachiketa Gargi
ngargi@umich.edu

Eugene Kim
keugene@umich.edu

Selinah Liang
selinahl@umich.edu

Grace Ma
gwma@umich.edu

1. Introduction

Virtual reality is a rapidly advancing field. Making virtual reality technology accessible to everyone would be beneficial, as there are many applications. In some sectors, for example, VR can be used to train employees who work in dangerous environments. VR is also useful for entertainment and educational purposes. To create a realistic experience, virtual reality technology requires accurate human pose estimation, of which a main component is localizing the body parts, or ”keypoints”, of the individual user. However, varying angles, inconsistent illumination, occlusion, ambiguity of human poses, and other factors make pose estimation a difficult task.

In this paper, we focus on monocular pose estimation for virtual reality motion capture. Based off of a singular image of a person, we aim to accurately estimate their 2D skeletal pose. With this, we can advance to monocular 3D pose estimation. This would allow 3D pose detection using a single webcam, which can be used for more accessible, commercially-viable virtual reality motion capture. Currently, full-body presence in VR is achieved via expensive trackers which must be worn on the body during usage; while accurate, these trackers are more expensive and complicated to use compared to a webcam. Many researches have worked on solving this problem.

For single-person pose estimation, Su et al. proposed a novel Cascade Feature Aggregation (CFA) method, which cascades hourglass networks for more robust pose estimation. Through their testing on MPII and LIP datasets, this CFA achieves the best performance on state-of-the-art benchmark MPII [1].

The next step after individual pose estimation would be multiple-person pose estimation, which we do not cover. This was considered by Cao et al., who presented an effective method to detect the 2D pose of multiple people in an image using a nonparametric representation [2]. Their work marks the release of OpenPose, which is the first real-time system for multiple-person 2D pose detection [2]. Their paper details a multi-stage CNN, in which they use convolution blocks and concatenate the predictions of each stage.

Our method builds on these. We use convolutional networks to predict the human keypoints for 2D pose estimation, and to predict the z-locations of the joints to create a predicted 3D pose. Our output 3D pose predictions are relative to a root joint, which we chose as the neck; for virtual reality body tracking applications, the exact 3D position of the head is known through other tracking systems on a head-mounted display.

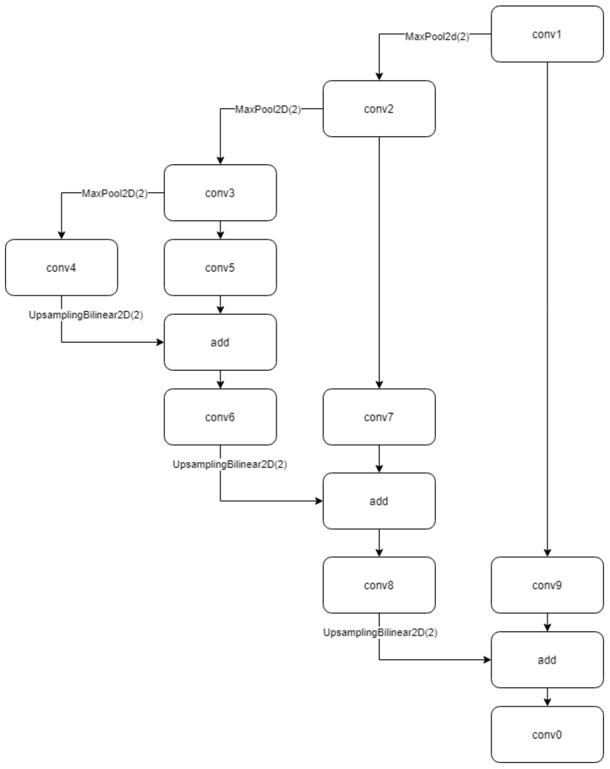

Figure 1. Cascading Convolutional Structure. Upsampled and Downsampled data is added together to preserve features across resolutions.

2. Approach

For 2D pose estimation, we use a convolutional network to create dense predictions for the keypoints that make up a human skeleton. We use the first 10 layers of pre-trained VGG-19 [2] as our encoder network. During training, we freeze gradients for these 10 layers to speed up convergence.

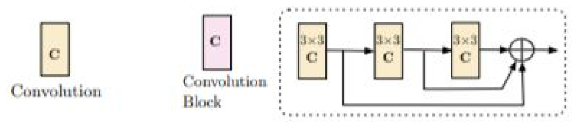

Figure 2. Convolution Block. Three convolutions of kernel size 3 mimics one convolution of kernel size 7.

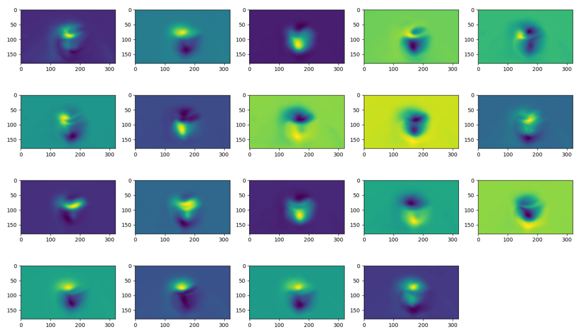

Figure 3. The heatmaps generated during our 2D pose estimation.

Our decoder architecture then uses a Cascade convolutional structure [7] in order to preserve both high resolution and low resolution features, as shown in Figure 1. This is followed by a series of ConvBlocks [2], as shown in Figure 2. We use three layers of convolutions of kernel size 3, each followed by a ReLU. At the end, the three outputs are added together.

Our convolutional network outputs 19 2D heatmaps, one for each keypoint. They represent how likely each joint is at a specific pixel, as shown in Figure 3.

For 3D pose prediction, we use a convolutional network to predict z-locations. We use the 2D heatmaps generated by the 2D predictions, as well as the original image features. We then make a new convolution network consisting of a ConvBlock $\rightarrow$ MaxPool $\rightarrow$ ConvBlock $\rightarrow$ Cascade $\rightarrow$ ConvBlock $\rightarrow$ Linear. This outputs 19 values that represent the z values of all 19 keypoints.

Our loss function consists of two terms: first, the average distance (in pixels) between the predicted 2D points and the ground truth 2D points; second, the average distance (in pixels) between the ground truth z value with the predicted z value for all 19 points. We get the predicted 2D points by taking the spatial soft argmax2D of the heatmap generated [6].

$$l = \sum_{i=0}^{T} \sum_{i=0}^{N} \left(\mathbf{J}_{\text{2D}} - \hat{\mathbf{J}}_{\text{2D}} \right)^2 + \sum_{i=0}^{T} \sum_{i=0}^{N} \left(J_{\text{depth}} - \hat{J}_{\text{depth}} \right)^2$$

The training procedure begins by training on the 2D images, and then moving onto the 3D predictions. We used the AdamW optimizer [9], with a 10^{-5} learning rate and a batch size of 16. Since our dataset is comprised of frames from videos, sequential images are often very similar and can lead to the optimizer getting stuck in a local minimum. Therefore, during training, the images are randomly sampled. In addition, we created a Discord Bot to give live updates on the training progress, loss, and validation images of our model. This allows all group members to monitor training, even though the model was only training on one member's GPU.

3. Experiments

3.1. Data

We originally intended to use the Human3.6M dataset [4], but were not able to obtain access, as the researchers have limited their dataset to established academic research groups only. As an alternative, we used images from the CMU Panoptic Studio dataset by Joo et al. [5], which includes images in which there are 19 3D joint locations (keypoints). The ordering of the keypoints differs from the OpenPose output order, although Joo et al.'s method is based on it [5].

The CMU Panoptic dataset contains videos from multiple angles along with 3D pose information from motion capture systems. Since we do not plan to incorporate time sequences, we treat the videos as multiple series of standalone images. We then resize these images from 1080x1920 to 180x320, as the original image size was far too large to reasonably train on due to GPU memory constraints. When parsing the ground truth joint positions, we normalize the joint positions to the neck, so that we can use the neck as the root joint. Next, we project the 3D joints to 2D using the given camera parameters so we can have 2D ground truth labels. In addition, the dataset included some images in which not all 19 keypoints were present (e.g. a photo in which the subject's legs were not visible). We filtered these images out before training so that they would not skew our results.

For training, we used the `17102_pose1`, `17102_pose2`, `17102_pose3`, and `17102_pose4` datasets using the HD cameras 1-19. For validation, we used the `17102_pose5` dataset using HD camera 1.

3.2. Metrics

Pose estimation is often evaluated using the MPJPE metric, which is the mean per joint position error [3]. Below is the equation, where T is the number of samples and N is the number of joints. Per joint position error is the Euclidean distance between ground truth and prediction for a joint. Mean per joint position error is the mean of per joint position error for all N joints (typically, N = 16, but we use N = 19).

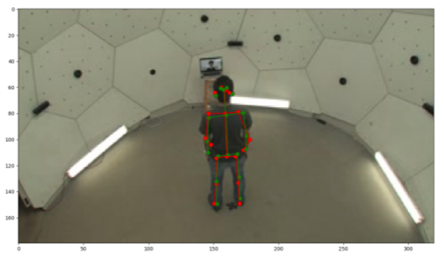

Figure 4. 2D Plot of the predicted keypoints (in red) versus the correct keypoints (in green). The model works well on people in neutral poses, which it had a lot of training images of.

$$\text{MPJPE} = \frac{1}{T}\frac{1}{N}\sum_{t=1}^{T}\sum_{i=1}^{N}||J_i^{(t)} - \hat{J}_i^{(t)}||_2$$

The joints are normalized with respect to the root joint, which is the neck. MPJPE is calculated using the formula above after transforming output points back to the original world coordinate space in order to return to using centimeters as the units.

We were also able to generate 2D and 3D images of our skeletal pose based upon our joint estimations. We layered the ground truth pose and predicted pose on top of the images to see how well it lines up, as shown in Figures 4 and 6. We also made 3D plots of our ground truth pose and predicted pose, as shown in Figures 5 and 7. This allows us to visualize how well our model predicted the depth of the joints.

3.3. Investigation

To make our model, we experimented with many different structures and hyperparameters to try to achieve the best result. We experimented with a variety of encoder networks, including but not limited to ResNet, VGG-19, and VGG-16, and found that VGG-19 provided the most accurate results. Upon training our network with ResNet [6], we found no significant improvements in our error. We also attempted not using an encoder network and instead starting with the raw image. However, this increased training time significantly, and led to greater overfitting due to our limited dataset [4]. To prevent overfitting, we increased the number of training images and randomized their order. We also originally tried to one-hot encode the depth coordinate estimation and use argmax to retrieve the actual coordinate; however, since argmax is non-differentiable, the gradients were not correctly backpropagated through the network. Our final model outputs the z coordinates as just a single float for each joint.

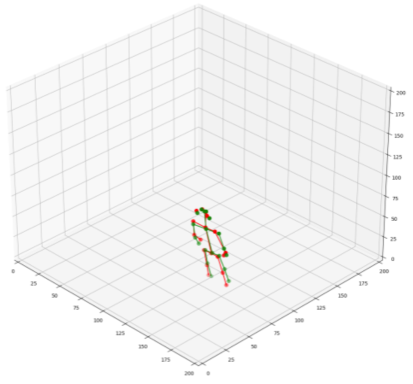

Figure 5. 3D Plot of the predicted keypoints (in red) versus the correct keypoints (in green). The model works well on people in neutral poses, which it had a lot of training images of.

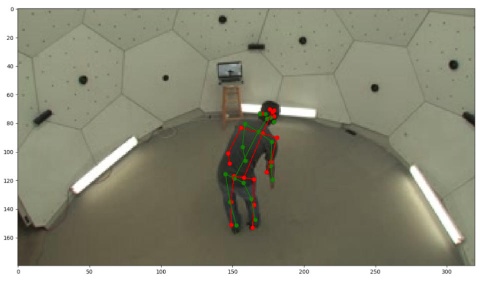

Figure 6. 2D Plot of the predicted keypoints (in red) versus the correct keypoints (in green). The model did a good estimation on all joints, despite the occluded left arm.

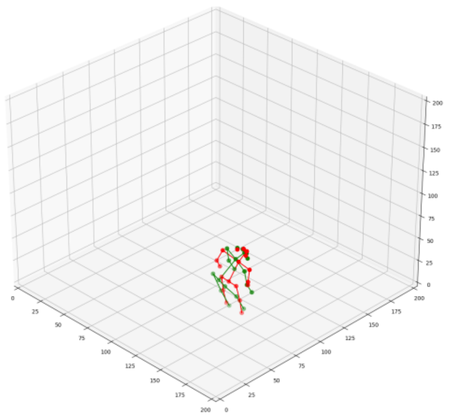

Figure 7. 3D Plot of the predicted keypoints (in red) versus the correct keypoints (in green). The model clearly did an accurate estimation of depth.

3.4. Qualitative Results

We stitched many of these images together to create videos of our pose estimation, which are available at `https://drive.google.com/drive/folders/`

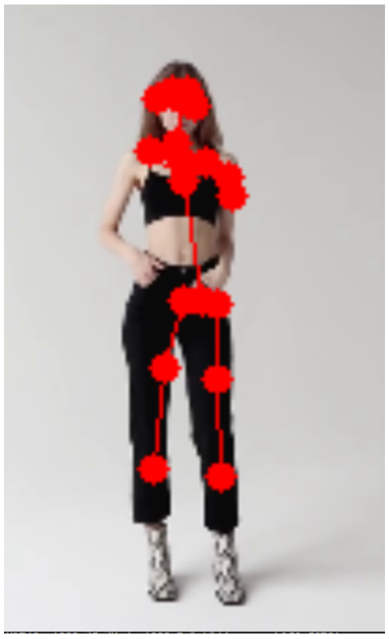

Figure 8. Our model does not work as well on a wild image [11]. While the legs are some what accurate, it does not identify the arms well. We believe that this is due to the fact that most of our training images were of people facing backwards, where the arms are not visible often.

`1XGNP0Skc2gQCGJLupbvfixuDfQmYhgmB?usp=sharing`. We then ran our model in real time against a new video in the wild [11], to see if it could recognize poses outside of the data set. While the video from inside of our dataset gave satisfactory results, the new video was lacking in accuracy, as shown in Figure 8. This is due to the homogeneity of the CMU Panoptic dataset, in which all images had the same background and similar camera angles. As a result, we believe our model may have overfit on the dataset images.

3.5. Quantitative Results

The MPJPE for state-of-the-art monocular 3D pose estimation is 49.6 millimeters [10]. We regard this measurement as the baseline. We ended up with a MPJPE of around 89.71 millimeters, which is around 40 milimeters worse than our baseline. This is still better than our desired MPJPE of 200 millimeters.

Inference of our model was able to run in real-time (60fps) on a GeForce RTX 3080 GPU, which is important for a VR application which requires low latency.

4. Implementation

We implemented our own function for loading the CMU Panoptic Studio dataset into memory. This is where we exclude camera angles where not all 19 keypoints were present. For projecting our 2D points to 3D points using camera intrinsics, extrinsics, and distortion parameters, we use a function provided by the CMU Panoptic Studio [5]. To project these 3D points back to 2D in order to train our network on the data, we use the same projection function that we used to project the 2D points to 3D, and then downsize our output plots so they fit our GPU memory constraints. We implemented a Cascade convolutional structure. We followed the structure in [7], but we only use 3 MaxPool2Ds instead of the 4 MaxPool2Ds used in [7], as our image sizes are smaller. For our ConvBlock, we followed the structure in [2]. However, the structure in [2] concatenates the outputs from each stage for the subsequent stage, while our structure adds the outputs. We use a spatial soft argmax2D to get the predicted 2D points of the heatmaps, as used in [6]. As opposed to standard argmax, the soft argmax function is differentiable which enables us to use it in our loss function and propagate gradients accurately back through the rest of the network.

5. Challenges

In our proposal, we intended to create a differentiable forward kinematics solver for predicting 3D poses. This proved to be more of a challenge than previously anticipated, since we weren't sure whether to have joint locations, rotations, or both as outputs from our network. Additionally, other kinematic models we looked into used the hip as the root of the kinematic tree which minimizes the degrees of freedom and error of limbs. Since we used the neck as our root node due to our VR constraint, there would be a far greater error by the time that the angles propagated all the way from the neck to the feet.

It was difficult to manage a large dataset in memory with only 32 gigabytes on our training machine. We solved this issue by creating a streaming dataloader in PyTorch that loads images on-the-fly and keeps memory usage low while still enabling us to use the full dataset.

Additionally, due to the size of the dataset and our model, tuning hyperparameters was a very tedious process. With more time, we believe it would be possible to find better values of hyperparameters for our model to train on, but since each epoch took over an hour, it was not feasible in this timeframe.

We believe our model has overfitting issues due to the lack of varied angles, lighting conditions, and backgrounds in the training data. We found that our model was unable to differentiate between the left and right side of the body due to joint ambiguity, which was especially apparent in the heatmaps. This could be mitigated by increasing kernel sizes or adding another level of cascading so that the network could detect the overall orientation of the person.

References

[1] Z. Su, M. Ye, G. Zhang, L. Dai, J. Sheng, ”Cascade Feature Aggregation for Human Pose Estimation”, 2019.

[2] Z. Cao, G. Hidalgo, T. Simon, S. Wei, and Y. Sheikh, ”Open-Pose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields” in *IEEE*, 2019.

[3] S. Li, A. Chan, ”3D Human Pose Estimation from Monocular Images with Deep Convolutional Neural Network” in *ACCV*, 2014.

[4] C. Ionescu, D. Papava, V. Olaru, C. Sminchisescu, ”Human3.6M”, 2011.

[5] Joo, Hanbyul and Liu, Hao and Tan, Lei and Gui, Lin and Nabbe, Bart and Matthews, Iain and Kanade, Takeo and Nobuhara, Shohei and Sheikh, Yaser, ”Panoptic Studio: A Massively Multiview System for Social Motion Capture”, at *ICCV*, 2015.

[6] Iskakov, Karim and Burkov, Egor and Lempitsky, Victor and Malkov, Yury, ”Learnable Triangulation of Human Pose”, at *ICCV*, 2019.

[7] A. Bulat, J. Kossaifi, G. Tzimiropoulos, M. Pantic, ”Toward fast and accurate human pose estimation via soft-gated skip connections”, 2020.

[8] S. Souravjha, ”Pose-estimators: Toward fast and accurate human pose estimation via soft-gated skip connections”, https://github.com/shivamsouravjha/Pose-estimators, 2020.

[9] S. Gugger, J. Howard, ”AdamW and Super-convergence is now the fastest way to train neural nets”, 2018.

[10] X. Sun, B. Xiao, F. Wei, S. Liang, Y. Wei, ”Integral human pose regression”, in *ECCV*, 2018.

[11] T. Kurnosova, ”Model posing | Natural simple modeling poses | Fashion model test shoot | How to visual tutorial”, https://www.youtube.com/watch?v=mIuQvMzFWV0, 2020.