1. MPI_INF_3DHP on VideoPose3D

I want to show that VideoPose3D can be generalized to MPI_INF_3DHP dataset. MPI_INF_3DHP [5] is a more challenging dataset than Human3.6M. It uses 8 subjects performing 8 actions using 14 synchronized cameras. In addition, some of the test dataset is in an outdoor environment.
In order to train MPI_INF_3DHP on VideoPose3D, I need to process the dataset to the same format as the Human3.6M dataset. And I also need to create a MPI_INF_3DHP dataset class.

Prepare dataset

I created a new file prepare_data_mpi_inf_3dhp.py. It is based on prepare_data_h36m.py. I basically structured the file into the same format as described above with one important difference. For Human3.6M, the ground truth 3D joints are recorded with World coordinate. Thus, during the data processing step. The 3D joint coordinates need to be converted into Camera Coordinates. However, Mpi_inf_3dhp is recorded in Camera coordinates already, so I do not need to convert them. Thus, I made some changes to the training script as shown in Figure 1 to reflect this.

```
print('Preparing data...')
if args.dataset.startswith('mpi'):
        for subject in dataset.subjects():
            for action in dataset[subject].keys():
                anim = dataset[subject][action]

                if 'positions' in anim:
                    anim['positions_3d'] = anim['positions']
else:
        for subject in dataset.subjects():
            for action in dataset[subject].keys():
                anim = dataset[subject][action]

                if 'positions' in anim:
                    positions_3d = []
                    for cam in anim['cameras']:
                        pos_3d = world_to_camera(anim['positions'], R=cam['orientation'], t=cam['translation'])
                        pos_3d[:, 1:] -= pos_3d[:, :1] # Remove global offset, but keep trajectory in first position
                        positions_3d.append(pos_3d)
                    anim['positions_3d'] = positions_3d

```

Figure 1. Code snippet of preparing data in the main code

Dataset class

Next, I mimic the structure of h36m_dataset.py to make mpi_inf_3dhp_dataset.py. Three important changes I need to make are fps, skeleton, and camera. For fps, the paper [6] documents the fps for MPI_INF_3DHP as 25fps.
For skeleton, MPI_INF_3DHP provides a script that can convert the index to Human3.6M style index. Then, I am able to reuse the human3.6M skeleton model since it has the matching joint index. I also made a drawing as shown in Figure 2 to show the index.

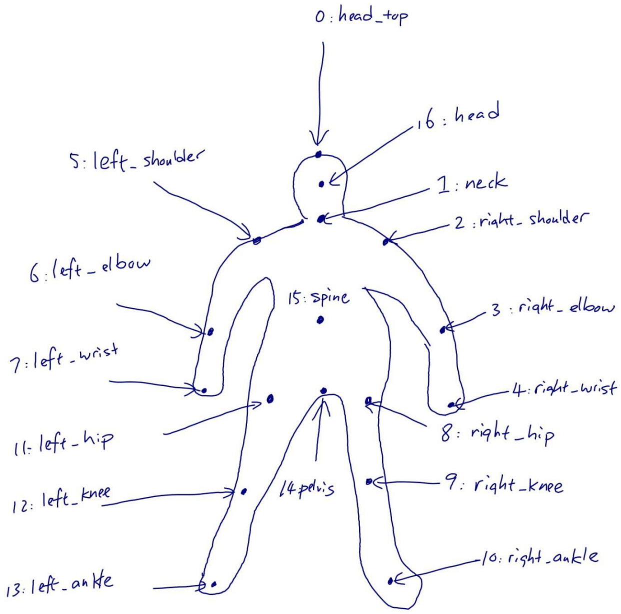

Figure 2. The hand drawn index mapping for the human pose

Lastly, I need to handle the camera parameters.
To project 3D joints to 2D. It requires camera intrinsic parameters. For visualization purposes, it also needs extrinsic parameters. As shown in Figure 3, the following intrinsic parameters are needed.

```
h36m_cameras_intrinsic_params = [
    {
        'id': '54138969',
        'center': [512.54150390625, 515.4514770507812],
        'focal_length': [1145.0494384765625, 1143.7811279296875],
        'radial_distortion': [-0.20709891617298126, 0.24777518212795258, -0.0030751503072679043],
        'tangential_distortion': [-0.0009756988729350269, -0.00142447161488235],
        'res_w': 1000,
        'res_h': 1002,
        'azimuth': 70, # Only used for visualization
    },
```

Figure 3. Intrinsic parameters for Human3.6M dataset
I need a center, focal length, radial and tangential distortion, width and height.
As shown in Figure 4, for extrinsic parameters, I need to specify the orientation and the translation.

```
h36m_cameras_extrinsic_params = {
    'S1': [
        {
            'orientation': [0.1407056450843811, -0.1500701755285263, -0.755240797996521, 0.6223280429840088],
            'translation': [1841.1070556640625, 4955.28466796875, 1563.4454345703125],
        },
        {
            'orientation': [0.6157187819480896, -0.764836311340332, -0.14833825826644897, 0.11794740706682205],
            'translation': [1761.278564453125, -5078.0068359375, 1606.2650146484375],
        },
        {
            'orientation': [0.14651472866535187, -0.14647851884365082, 0.7653023600578308, -0.6094175577163696],
            'translation': [-1846.7777099609375, 5215.04638671875, 1491.972412109375],
        },
        {
            'orientation': [0.5834008455276489, -0.7853162288665771, 0.14548823237419128, -0.14749594032764435],
            'translation': [-1794.7896728515625, -3722.698974609375, 1574.8927001953125],
        },
    ],
```

Figure 4. Extrinsic parameters for Human3.6M dataset

Within the MPI_INF_3DHP, I’ve been given the following parameters shown in figure 5.

```
name         0
  sensor     10 10
  size       2048 2048
  animated   0
  intrinsic  1497.693 0 1024.704 0 0 1497.103 1051.394 0 0 0 1 0 0 0 0 1
  extrinsic  0.9650164 0.00488022 0.262144 -562.8666 -0.004488356 -0.9993728
  radial     0
```

Figure 5. Camera parameters for MPI_INF_3DHP dataset

We have size, intrinsic matrix and extrinsic matrix. The size specifies the width and height. First of all, let's take a look at the extrinsic parameters. I am putting the line into matrix form as shown in Table1.

0.9650164	0.00488022	0.262144	-562.8666
-0.004488356	-0.9993728	0.0351275	1398.138
0.262151	-0.03507521	-0.9643893	3852.623
0	0	0	1

Table 1. Extrinsic parameters in matrix form.

Figure 6 shows how camera extrinsic parameters are mapped:

$$[R\,|\,\boldsymbol{t}] = \left[\begin{array}{ccc|c} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \end{array}\right]$$

It's common to see a version of this matrix with extra row of (0,0,0,1) added to the bottom.

Figure 6. How extrinsic parameters are mapped [2].

Thus, we can see the upper 3*3 is the rotation matrix, while the three variables at right are the translational matrix. However, this is not the format I wanted. Rather than a 3x3 matrix, I need a Quaternion. Thanks to this website [3], I am able to make the conversion.

This handles the extrinsic parameters. As for the intrinsic parameters, I will first put the line into matrix form as shown in Table 2.

1497.693	0	1024.704	0
0	1497.103	1051.394	0
0	0	1	0
0	0	0	1

Table 2. Intrinsic parameters in matrix form.

Figure 7 shows how intrinsic parameters are mapped:

$$K = \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}$$

Figure 7. How intrinsic parameters are mapped [4].

f_x and f_y are the focal length, x_0 and y_0 are the center. The last parameter I need are the radial and tangential distortion. Radial distortion is specified as 0 in the calibration file. However, it did not specify the tangential distortion, I will assume it is 0. The resulting code is available on my github page: https://github.com/zemingxie/MPI_INF_3DHP-on-VideoPose3D.git

Result

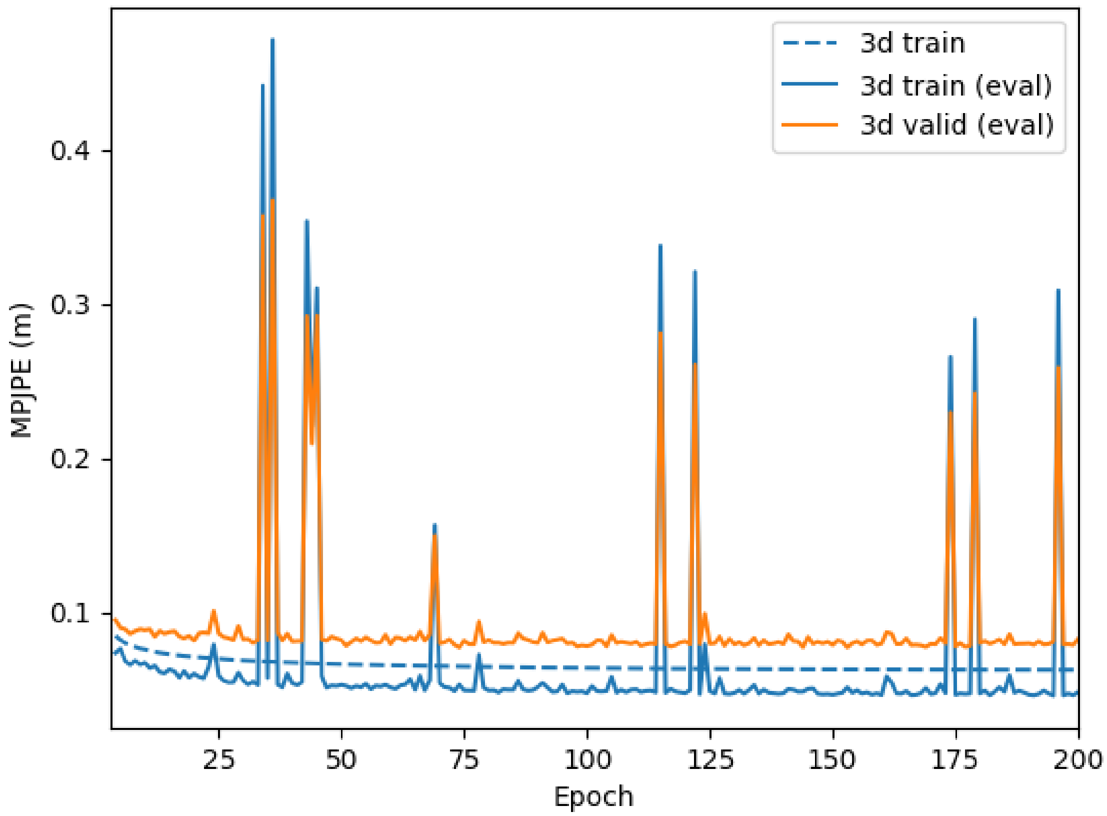

Figure 8. The training curve for MPI_INF_3DHP on VideoPose3D.
The resulting MPJPE for MPI_INF_3DHP is at around 80mm. Figure 8 shows the training curve. It does have some spikes which is due to the improper momentum value set for the batch normalization layer. MPI_INF_3DHP is smaller than HumanPose3.6M, thus it needs a higher momentum value which also needs a smaller architecture. The training parameter I set for the above graph is default 3,3,3 arc which is 27 frames. Within the run code, I modified the momentum to be around 0.8 to 0.6. With more parameter tuning, we expect to further improve the result. Thus, my work shows that VideoPose3D can also be generalized on MPI_INF_3DHP dataset.

References

1. Pavllo, D., Feichtenhofer, C., Grangier, D., & Auli, M. (2019). 3D human pose estimation in video with temporal convolutions and semi-supervised training. In Conference on Computer Vision and Pattern Recognition (CVPR).
2. Simek, K. (2012, August 22). *Dissecting the camera matrix, part 2: The extrinsic matrix*. Retrieved April 29, 2022, from https://ksimek.github.io/2012/08/22/extrinsic/
3. Gaschler, A. (n.d.). 3D Rotation Converter. Retrieved April 29, 2022, from https://www.andre-gaschler.com/rotationconverter/
4. Simek, K. (2013, August 13). *Dissecting the camera matrix, part 3: The Intrinsic matrix*. Retrieved April 29, 2022, from https://ksimek.github.io/2013/08/13/intrinsic/
5. Mehta, D., Rhodin, H., Casas, D., Fua, P., Sotnychenko, O., Xu, W., & Theobalt, C. (2017). Monocular 3D Human Pose Estimation In The Wild Using Improved CNN Supervision. In *3D Vision (3DV), 2017 Fifth International Conference on*.
6. Jiahao Lin and Gim Hee Lee. Trajectory space factorization for deep video-based 3d human pose estimation. In BMVC, 2019.