Supplementary Material: Lightweight Multi-View 3D Pose Estimation through Camera-Disentangled Representation

Edoardo Remelli[1] Shangchen Han[2] Sina Honari[1] Pascal Fua[1] Robert Wang[2]
[1]CVLab, EPFL, Lausanne, Switzerland
[2]Facebook Reality Labs, Redmond, USA

1. Architectures

In Figure 1, we depict the different architectures (*baseline*, *fusion*, *canonical fusion*) compared in the main article. Recall that our encoder consists of a ResNet152 [4] backbone pre-trained on ImageNet [2] for all three architectures, taking in 256×256 image crops as input and producing $2048 \times 18 \times 18$ features maps. Similarly, all methods share the same convolutional decoder, consisting of

- ConvTranspose2D(2048, 256) + BatchNorm + ReLU
- ConvTranspose2D(256, 256) + BatchNorm + ReLU
- ConvTranspose2D(256, 256) + BatchNorm + ReLU
- Conv2D(256, K).

This produces $K \times 64 \times 64$ output heatmaps, where K is the number of joints. The only difference between the networks is in the feature fusion module, respectively defined as follows:

- *baseline*: no feature fusion.
- *fusion*: a 1×1 convolution is first applied to map features from 2048 channels to 300. Then, the feature maps from different views are concatenated to make a feature map of size $n \times 300$, where n indicates the number of views. This feature map is then processed jointly by two 1×1 convolutional layers, finally producing a feature map with $n \times 300$ channels, which is later split into view-specific feature maps with 300 channels in each view. Each view-specific feature map is then lifted back to 2048 channels.
- *canonical fusion*: a 1×1 convolution is first applied to map features from 2048 channels to 300. The feature maps from different views are then transformed to a shared canonical representation (world coordinate system) by feature transform layers. Once they live in the same coordinate system, they are concatenated into a $n \times 300$ feature map and processed jointly by two 1×1 convolutional layers, producing a *unified* feature map with 300 channels that is disentangled from the camera view-point. This feature map, denoted as $p_{3\mathrm{D}}$ in the main article, is then projected back to each view-point by using feature transform layers and the corresponding camera transform matrix. Finally each view-specific feature map is mapped back to 2048 channels. Note that in contrast to *fusion* that learns separate latent representations for different views, in *canonical fusion* all views are reconstructed from the same latent representation, effectively forcing the model to learn a unified representation across all views.

2. Efficient Direct Linear Transformation

In this section we prove Theorem 1 from the main article, and then illustrate how in practice we use it to design an efficient algorithm for Direct Linear Transformation by using Shifted Inverse Iterations method [9]. Finally, we provide some insight on why SVD is not efficient on GPUs (see Figure 3d in the main article).

Theorem 1. *Let A be the DLT matrix associated with the non-perturbed case, i.e. $\sigma_{min}(A) = 0$. Let us assume i.i.d Gaussian noise $\varepsilon = (\varepsilon_u, \varepsilon_v) \sim \mathcal{N}(0, s^2 I)$ in our 2d observations, i.e. $(u^*, v^*) = (u + \varepsilon_u, v + \varepsilon_v)$, and let us denote A^* the DLT matrix associated with the perturbed system. Then, it follows that:*

$$0 \leq \mathbb{E}[\sigma_{min}(A^*)] \leq Cs, \text{ where } C = C(\{u_i, P_i\}_{i=1}^N) \quad (1)$$

Proof. Let us recall the structure of matrix $A \in \mathbb{R}^{2n \times 4}$, which is the DLT matrix for non-noisy 2D observations:

$$A = \begin{bmatrix} \vdots \\ u_i p_i^{3T} - p_i^{1T} \\ v_i p_i^{3T} - p_i^{2T} \\ \vdots \end{bmatrix}. \quad (2)$$

Now considering noisy observations $(u_i^*, v_i^*) = (u_i + \varepsilon_{2i}, v_i + \varepsilon_{2i+1})$, where we drop the subscripts u, v from ε (as noise is i.i.d.), the DLT matrix can be written as

$$A^* = \begin{bmatrix} \vdots \\ (u_i + \varepsilon_{2i})\, p_i^{3T} - p_i^{1T} \\ (v_i + \varepsilon_{2i+1})\, p_i^{3T} - p_i^{2T} \\ \vdots \end{bmatrix}, \tag{3}$$

which is equivalent to

$$A^* = A + \begin{bmatrix} \vdots \\ \varepsilon_{2i}\, p_i^{3T} \\ \varepsilon_{2i+1}\, p_i^{3T} \\ \vdots \end{bmatrix} \tag{4}$$

$$= A + \begin{bmatrix} \ddots & & & \\ & \varepsilon_{2i} & & \\ & & \varepsilon_{2i+1} & \\ & & & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ p_i^{3T} \\ p_i^{3T} \\ \vdots \end{bmatrix} \tag{5}$$

$$= A + \Sigma P, \tag{6}$$

where $\Sigma \in \mathbb{R}^{2n \times 2n}$ and $P \in \mathbb{R}^{2n \times 4}$.

Using the classical perturbation theory (see Stewart *et al.* [10] for an overview), we can write

$$|\sigma_{\min}(A^*) - \sigma_{\min}(A)| \leq \|A^* - A\|_2. \tag{7}$$

By exploiting $\sigma_{\min}(A) = 0$, Equation 6, and the fact that singular values are always positive we can infer

$$\sigma_{\min}(A^*) \leq \|\Sigma P\|_2. \tag{8}$$

Then by leveraging Cauchy-Schwartz inequality [1] and recalling that the norm 2 of a diagonal matrix is bounded by the absolute value of the biggest element in the diagonal we get

$$\sigma_{\min}(A^*) \leq \|\Sigma\|_2 \|P\|_2 \leq \|P\|_2 \max_i |\varepsilon_i|. \tag{9}$$

Recall that that the max of $2n$ i.i.d. variables is smaller than their sum, so we can write

$$\sigma_{\min}(A^*) \leq \|P\|_2 \sum_{i=0}^{2n-1} |\varepsilon_i|. \tag{10}$$

We can then simply take the expected value on both sides of Equation (10) and obtain

$$\mathbb{E}\Big[\sigma_{\min}(A^*)\Big] \leq \mathbb{E}\Big[\|P\|_2 \sum_{i=0}^{2n-1} |\varepsilon_i|\Big] \tag{11}$$

$$\leq \|P\|_2 \sum_{i=0}^{2n-1} \mathbb{E}[|\varepsilon_i|] \tag{12}$$

$$\leq \|P\|_2\, 2n\, \mathbb{E}[|\varepsilon_0|]. \tag{13}$$

Knowing that the expected value of the *half-normal* distribution is $E[|\varepsilon_i|] = s\sqrt{2/\pi}$ we finally obtain

$$\mathbb{E}[\sigma_{\min}(A^*)] \leq 2n\sqrt{2/\pi}\|P\|_2\, s = Cs. \tag{14}$$

The other side of inequality (1) trivially follows from the fact that singular values are always positive. □

In the main article, we proposed (in Algorithm 1) to find the singular vector of A^* associated with $\sigma_{\min}(A^*)$ by means of Shifted Inverse Iterations (SII) [9] applied to matrix $A^{*T}A^*$. This iterative algorithm (which takes as input a singular value estimate μ) has the following properties:

1. The iterations will converge to the eigenvector that is closest to the provided estimate.

2. The rate of convergence of the algorithm is geometric, with ratio $\dfrac{\sigma_4(A^*) + \mu}{\sigma_3(A^*) + \mu}$, where $\sigma_3 \geq \sigma_4 = \sigma_{\min}$.

Combining property 1 with the result of Theorem 1 ascertains that Algorithm 1 will converge to the desired singular vector if we provide it with a small value for μ. Although in theory we could set $\mu = 0$, in practice we choose $\mu = 0.001$ to avoid numerical instabilities when matrix $A^{*T}A^*$ is close to being singular.

Note also that property 2 is confirmed by what we see in Figure 3b in the main article, where the number of iterations needed by the algorithm to reach convergence increases with more Gaussian noise in the 2D observation. In practice, we have found two iterations to be sufficient in our experiments.

SVD parallelization on GPU. In our experiments, carried in `PyTorch v1.3` on a Pascal TITAN X GPU, we found DLT implementations based on Singular Value Decomposition (SVD) to be inefficient on GPU (see Figure 3d in the main paper). Below we provide an insight on why this is the case.

SVD numerical implementations [3] involve two steps:

1. Two orthogonal matrices Q and P are applied to the left and right of matrix A, respectively, to reduce it to a bidiagonal form, $B = Q^T AP$.

2. Divide and conquer or QR iteration is then used to find both singular values and left-right singular vectors of B yielding $B = \bar{U}^T \Sigma \bar{V}$. Then, singular vectors of B are back-transformed to singular vectors of A by $U = Q\bar{U}$ and $V = \bar{V}P$.

There are many ways to formulate these problems mathematically and solve them numerically, but in all cases, designing an efficient computation is challenging because of the nature of the reduction algorithm. In particular, the

orthogonal transformations applied to the matrix are two-sided, i.e., transformations are applied on both the left and the right side of the matrix. This creates data dependencies and prevents the use of standard techniques to increase the computational efficiency of the operation, for example blocking and look-ahead, which are used extensively in the one-sided algorithms (such as in LU, QR, and Cholesky factorizations [3]). A recent work [12] has looked into ways to increase stability of SVD while reducing its computational time. Similarly, we also found SVD factorization to be slow, which motivated us to design a more efficient solution involving only GPU-friendly operations (see Algorithm 1 in the main article).

3. Feature Transform Layer

Below we first review feature transform layers (FTLs), introduced in [13] as an effective way to learn interpretable embeddings. Then we explain how FTLs are used in our approach.

Let us consider a representation learning task, where images $\mathbf{X}$ and $\mathbf{Y}$ are related by a known transform T and the latent vector $\mathbf{x}$ is obtained from $\mathbf{X}$ via an encoder network. The feature transform layer performs a linear transformation on $\mathbf{x}$ via transformation matrix F_T such that the output of the layer is defined as

$$\mathbf{y} = F_T[\mathbf{x}] = F_T\mathbf{x}, \tag{15}$$

where $\mathbf{y}$ is the transformed representation. Finally $\mathbf{y}$ is decoded to reconstruct the target sample $\mathbf{Y}$. This operation forces the neural network to learn a mapping from image-space to feature-space while preserving the intrinsic structure of the transformation.

In practice, the transforming matrix F_T should be chosen such that it is invertible and norm preserving. To this end [13] proposes to use rotations since they are simple and respect these properties. Periodical transformations can trivially be converted to rotations. Although less intuitive, arbitrary transformation defined on an interval can also be thought of as rotations by mapping them onto circles in feature space. Figure 2 illustrates in detail how to compute this mapping.

Note that if $\mathbf{X}$ and $\mathbf{Y}$ differ by more than one factor of variation, disentanglement can be achieved by transforming features as follows:

$$\mathbf{y} = F_{T_1,\ldots,T_n}[\mathbf{x}] = \begin{bmatrix} F_{T_1} & & \\ & \ddots & \\ & & F_{T_n} \end{bmatrix} \mathbf{x}. \tag{16}$$

In [13] FTLs are presented as a way to learn representations from data that are 1) interpretable, 2) disentangled, and 3) better suited for down-stream tasks, such as classification.

In our work, we use FTLs to feed camera transformations explicitly into the network in order to design an architecture that can reason both efficiently and effectively about epipolar geometry in the latent space. As a consequence, the model learns a camera-disentangled representation of 3D pose, that recovers 2D joint locations from multi-view input imagery. This shows that FTLs can be used to learn disentangled latent representations also in supervised learning tasks.

4. Additional results

In Figures 3 and 4 we provide additional visualizations, respectively for TotalCapture [11] (using both seen and unseen cameras) and Human3.6M [5, 6] datasets. These uncurated figures illustrate the quality of our predictions. We encourage the reader to look at our supplementary videos for further qualitative results.

References

[1] Augustin-Louis Cauchy. Sur les formules qui resultent de l'emploie du signe et sur¿ ou¡, et sur les moyennes entre plusieurs quantites. *Cours d'Analyse, 1er Partie: Analyse algebrique*, pages 373–377, 1821. 2

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. 1

[3] Jack Dongarra, Mark Gates, Azzam Haidar, Jakub Kurzak, Piotr Luszczek, Stanimire Tomov, and Ichitaro Yamazaki. Accelerating numerical dense linear algebra calculations with gpus. In *Numerical computations with GPUs*, pages 3–28. Springer, 2014. 2, 3

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1

[5] Catalin Ionescu, Fuxin Li, and Cristian Sminchisescu. Latent structured models for human pose estimation. In *2011 International Conference on Computer Vision*, pages 2220–2227. IEEE, 2011. 3

[6] Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1325–1339, jul 2014. 3

[7] Abdolrahim Kadkhodamohammadi and Nicolas Padoy. A generalizable approach for multi-view 3d human pose regression. *ArXiv*, abs/1804.10462, 2018. 5

[8] Haibo Qiu, Chunyu Wang, Jingdong Wang, Naiyan Wang, and Wenjun Zeng. Cross view fusion for 3d human pose estimation. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019. 5

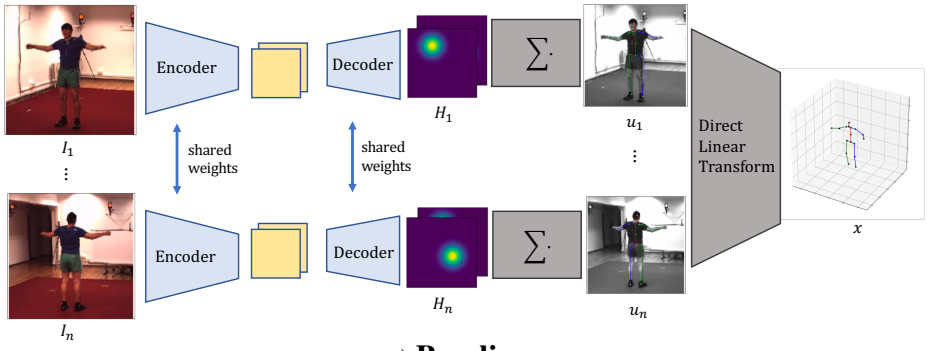
[9] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical mathematics*, volume 37. Springer Science & Business Media, 2010. 1, 2

[10] Gilbert W Stewart. Perturbation theory for the singular value decomposition. Technical report, 1998. 2

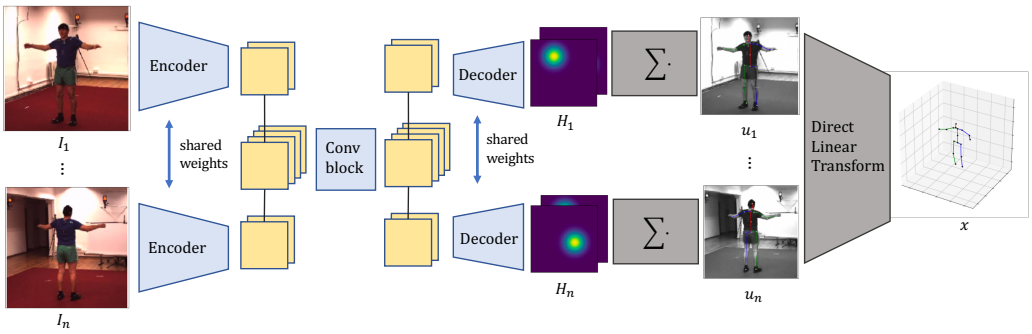
[11] Matthew Trumble, Andrew Gilbert, Charles Malleson, Adrian Hilton, and John Collomosse. Total capture: 3d human pose estimation fusing video and inertial sensors. In *BMVC*, volume 2, page 3, 2017. 3

[12] Wei Wang, Zheng Dang, Yinlin Hu, Pascal Fua, and Mathieu Salzmann. Backpropagation-friendly eigendecomposition. *arXiv preprint arXiv:1906.09023*, 2019. 3

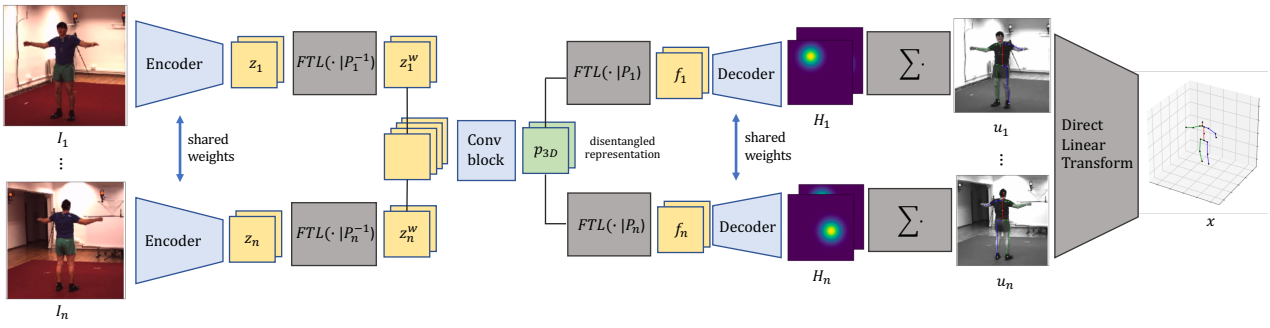
[13] Daniel E Worrall, Stephan J Garbin, Daniyar Turmukhambetov, and Gabriel J Brostow. Interpretable transformations with encoder-decoder networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5726–5735, 2017. 3



a) **Baseline**



b) **Fusion**



c) **Canonical Fusion**

Figure 1. Overview of different multi-view architectures: a) *baseline*, which detects 2D locations of joints for each view separately and then lifts detections to 3D via DLT triangulation. b) the multi-view feature fusion technique (*fusion*) that performs joint reasoning in the latent space, similar in spirit to the methods of [7, 8]. This approach does not exploit epipolar geometry and hence overfits to the camera setting. c) our novel fusion method (*canonical fusion*), exploiting camera transform layers to fuse views flexibly into a unified pose representation that is disentangled from camera view-points and thus can generalize to novel views.

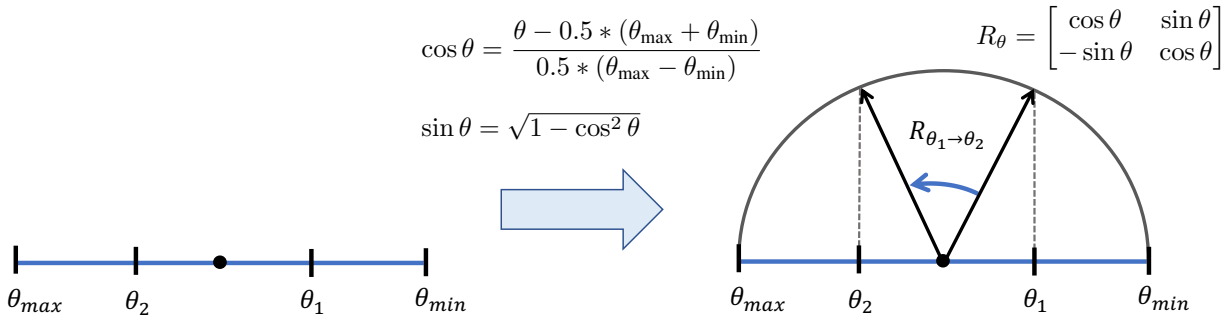


Figure 2. FTL encodes transformations by mapping them onto circles in the feature space. Consider the setting in which a factor of variation θ (e.g. x-component of camera position in world coordinates), defined in the interval $\theta \in [\theta_{\min}, \theta_{\max}]$, changes from $\theta = \theta_1$ to $\theta = \theta_2$. Exploiting trigonometry, we can map this transformation onto a circle, as depicted on the right-hand side of the figure, where the transformation is defined as a rotation.

Figure 3. Randomly picked samples from the test set of Human3.6M. Numbers denote cameras. To stress that the pose representation learned by our network is effectively *disentangled* from the camera view-point, we intentionally show predictions *before* triangulating them, rather than re-projecting triangulated keypoints to the image space.

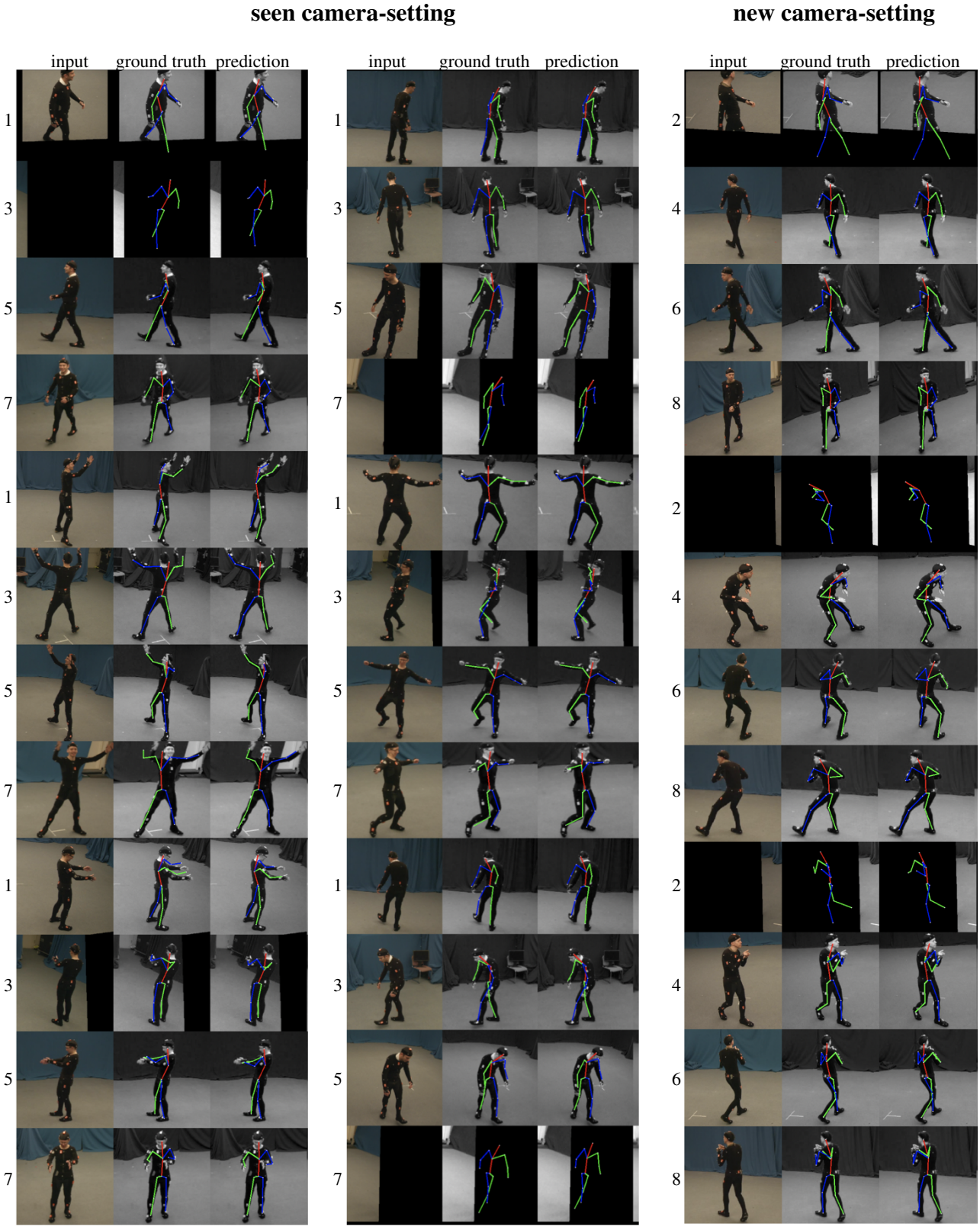


Figure 4. Randomly picked samples from the test set of TotalCapture. Numbers denote cameras. In the two left columns we test our model on unseen images captured from seen camera view-points. In the right column, instead, we use images captured from unseen camera view-points. To stress that the pose representation learned by our network is effectively *disentangled* from the camera view-point, we intentionally show predictions *before* triangulating them, rather than re-projecting triangulated keypoints to the image space.