Unsupervised Co-part Segmentation through Assembly (*Supplementary Material*)

Qingzhe Gao, Bin Wang, Libin Liu, Baoquan Chen

1. Outline

This document contains implementation details of our method. The content is organized as follows.

- **Section 2:** Network structure.
- **Section 3:** Implementation details.

2. Network Structure

As stated in the main paper, our segmentation network consists of two major modules: image encoder $\mathcal{E}$ and segment decoder $\mathcal{D}$. Their structures are shown in Figure 1. In this document, we assume the shape of a tensor is in the format of $C \times H \times W$, where C is the number of channels, H and W are the height and the width of the tensor, respectively.

2.1. Image Encoder

The image encoder $\mathcal{E}$ includes three parts: feature extractor, feature map estimator, and transformation estimator.

Feature Extractor. The feature extractor takes an image I of size $(3 \times 128 \times 128)$ as input and computes a feature image f of size $((128 \times (K+1)) \times 32 \times 32)$, where $(K+1)$ corresponds to K foreground parts and one background part.

The network is constructed based on the standard U-Net architecture (Ronneberger et al., 2015), where three cascaded downsampling blocks and the corresponding upsampling blocks are employed. The initial feature count is 512 and the max feature count is set to 1024 in our implementation. Before being input into the U-Net module, an input image will first pass through three convolutional layers, each followed by a ELU layer and an instance normalization layer, and two max-pooling layers. The output of the U-Net will pass through an additional convolutional layer to compute the final feature image. The parameters of these layers can be found in Table 1.

Feature Map Estimator. Based on the output feature image f of the feature extractor, the feature map estimator computes $(K+1)$ feature maps $\{V_k\}, k \in \{0, \dots, K\}$, each of size $10 \times 32 \times 32$. We employ two convolutional layers in this module, where a leaky ReLU with 0.2 negative slope is used as the activation function of the first convolutional layer. The parameters of these layers are shown in Table 2.

Transformation Estimator. Similar to the feature map estimator, the transformation estimator also takes the feature image f as input and computes the transformations of every part. The network first convert f into a one-dimensional feature vector using three convolutional layers, then three separate MLPs are employed to estimate the scaling (s_x, s_y), rotation (s_θ, c_θ), and translation (t_x, t_y) of the transformation. Note that we only estimate transformations of the K foreground parts. The transformations of the background part is assumed to be identity. Table 3 reports the parameters of these layers.

2.2. Segment Decoder

Segment Decoder. The segment decoder $\mathcal{D}$ takes a feature map V_k $(10 \times 32 \times 32)$ as input, feeds it through two convolutional layers, one residual block (He et al., 2016), and two upsampling blocks (Johnson et al., 2016), then finally outputs a tensor with size $(4 \times 128 \times 128)$. The first three channels of this tensor are considered as the part image P_k $(3 \times 128 \times 128)$, while the fourth channel is used as the depth map D_k $(1 \times 128 \times 128)$. The details are shown in Table 4.

3. Implementation Details

Our system is implemented in the PyTorch framework. We train our networks using Adam (Kingma & Ba, 2015), and the learning rate is set to 0.00005. We use batch size 6 for $K = 10$ and 4 for $K = 15$. The training is performed on one Titan X GPU with 12GB memory. We terminate the training when the learning progress stalls or exceeds $500,000$ iterations.

In experiment, we set K same to previous work to compare. In practice, the K is not sensitive to the result when K is big enough. The model can automatically select which channels to segment and set redundant channels to be empty. For hand, quadruped, and robot arm, we set the K is 16, it works well.

We empirically choose weights to balance the magnitude of each loss term in a preliminary training, except for the

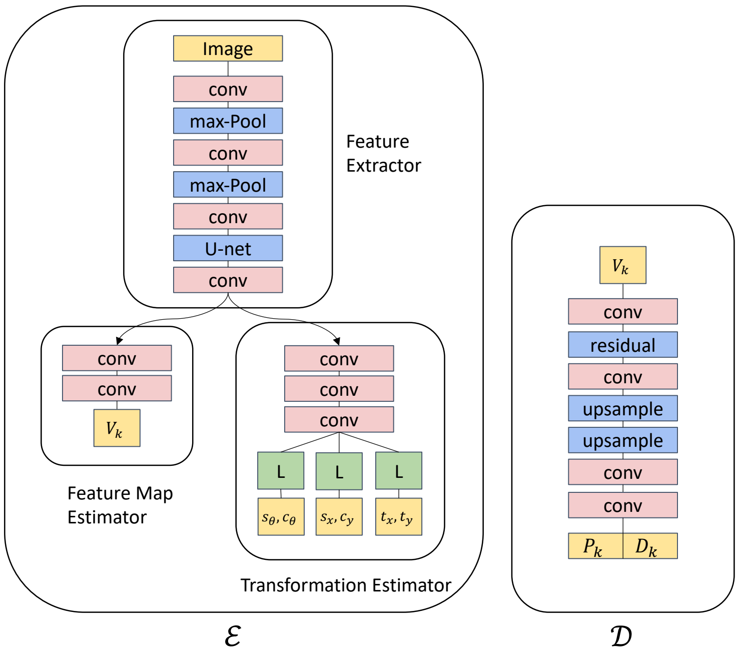

Figure 1. Two major modules of our network: image encoder $\mathcal{E}$ and segment decoder $\mathcal{D}$. The image encoder $\mathcal{E}$ is composes of three parts: feature extractor, feature map estimator and transformation estimator.

Table 1. Structure of Feature Extractor. Specifically, *conv*, *inst*, *elu* represent the convolution layer, instance normalization layer,and ELU activation respectively.

Parameter	Kernel	Stride	Channel in	Channel out	InpRes	OutRes
conv1+elu+inst	3×3	1	3	64	128×128	128×128
maxpool	2×2	1	64	64	128×128	64×64
conv2+elu+inst	3×3	1	64	256	64×64	64×64
maxpool	2×2	1	256	256	64×64	32×32
conv3+elu+inst	3×3	1	256	512	32×32	32×32
U-Net	3×3	1	512	1024	32×32	32×32
conv4	3×3	1	1024	$128 \times (K+1)$	32×32	32×32

image reconstruction loss, which is an order of magnitude larger than the other regularization terms due to its critical role in the training.The performance of our model is not sensitive to the specific choice of these loss weights, and similar video categories can share the loss weights.

Table 5 reports the weights of the training losses as described in Section 3.2 of the main paper. Specifically, the vgg loss in the image reconstruction loss is computed in five different resolutions: 128×128, 64×64, 32×32, 16×16, and 8×8. The weights of the corresponding loss terms are set to $[1.0/32, 1.0/16, 1.0/8, 1.0/4, 1.0]$ as suggested in (Wang et al., 2018). We use a smaller weight $\lambda_{con}/10$ for the concentration loss computed on canonical parts M_k^*. The random transformations used to compute the equivariance loss are generated uniformly in the scaling range $[0.8, 1.05]$, the rotation range $[-180°, 180°]$, and the translation range $[-0.2, 0.2]$.

We train separate networks for each dataset used in this work. The two images of an image pair are both randomly selected from the same video clip. We augment the training data using random rotations (in the range of $[-15°, 15°]$) and color jittering to increase the robustness of the network, where the same augmentation is applied to both the images of an image pair.

We have included some of our results and comparisons to the state-of-the-art approaches in the main paper. For additional results and comparison, please refer to Figure 2, Figure 3, Figure 4, Figure 5, Figure 6, and Figure 7 in this document, as well as the the supplementary video.

Table 2. Structure of Feature Map Estimator. *lrelu* represents leaky ReLU activation.

Name	Kernel	Stride	Channel in	Channel out	InpRes	OutRes
conv5+lrelu	7×7	1	$128 \times (K+1)$	$32 \times (K+1)$	32×32	32×32
conv6	7×7	1	$32 \times (K+1)$	$10 \times (K+1)$	32×32	32×32

Table 3. Structure of Transformation Estimator. *linear* represents linear layer.

Name	Kernel	Stride	Channel in	Channel out	InpRes	OutRes
conv7+lrelu	4×4	2	$128 \times (K+1)$	$32 \times (K+1)$	32×32	16×16
conv8+lrelu	4×4	2	$32 \times (K+1)$	$16 \times (K+1)$	16×16	8×8
conv9+lrelu	4×4	2	$16 \times (K+1)$	$4 \times (K+1)$	8×8	4×4
resize	-	-	$4 \times (K+1)$	$64 \times (K+1)$	4×4	1
linear1	-	-	$64 \times (K+1)$	$2 \times K$	1	1
linear2	-	-	$64 \times (K+1)$	$2 \times K$	1	1
linear3	-	-	$64 \times (K+1)$	$2 \times K$	1	1

Table 4. Structure of Decoder. *residual* and *upsample* represent residual block (He et al., 2016) and upsampling block (Johnson et al., 2016).

Name	Kernel	Stride	Channel in	Channel out	InpRes	OutRes
conv10+lrelu	7×7	1	10	64	32×32	32×32
residual	3×3	1	64	64	32×32	32×32
conv11+lrelu	3×3	1	64	128	32×32	32×32
upsample1	3×3	1	128	64	32×32	64×64
upsample2	3×3	1	64	32	64×64	128×128
conv12+lrelu	3×3	1	32	16	128×128	128×128
conv13	3×3	1	16	4	128×128	128×128

Table 5. Parameters setting.

Parameter	λ_1	λ_2	λ_3	λ_s	λ_t	λ_{bg}	λ_{tran}	λ_{rots}	λ_{eq}	λ_{con}
Tai-Chi-HD	10	5	1	0.1	1	0.5	1	0.1	0.02	0.35
VoxCeleb	10	5	1	0.1	1	0.05	1	0.05	0.02	0.35
Exercise	10	5	1	0.1	1	0.5	1	0.1	0.02	0.35

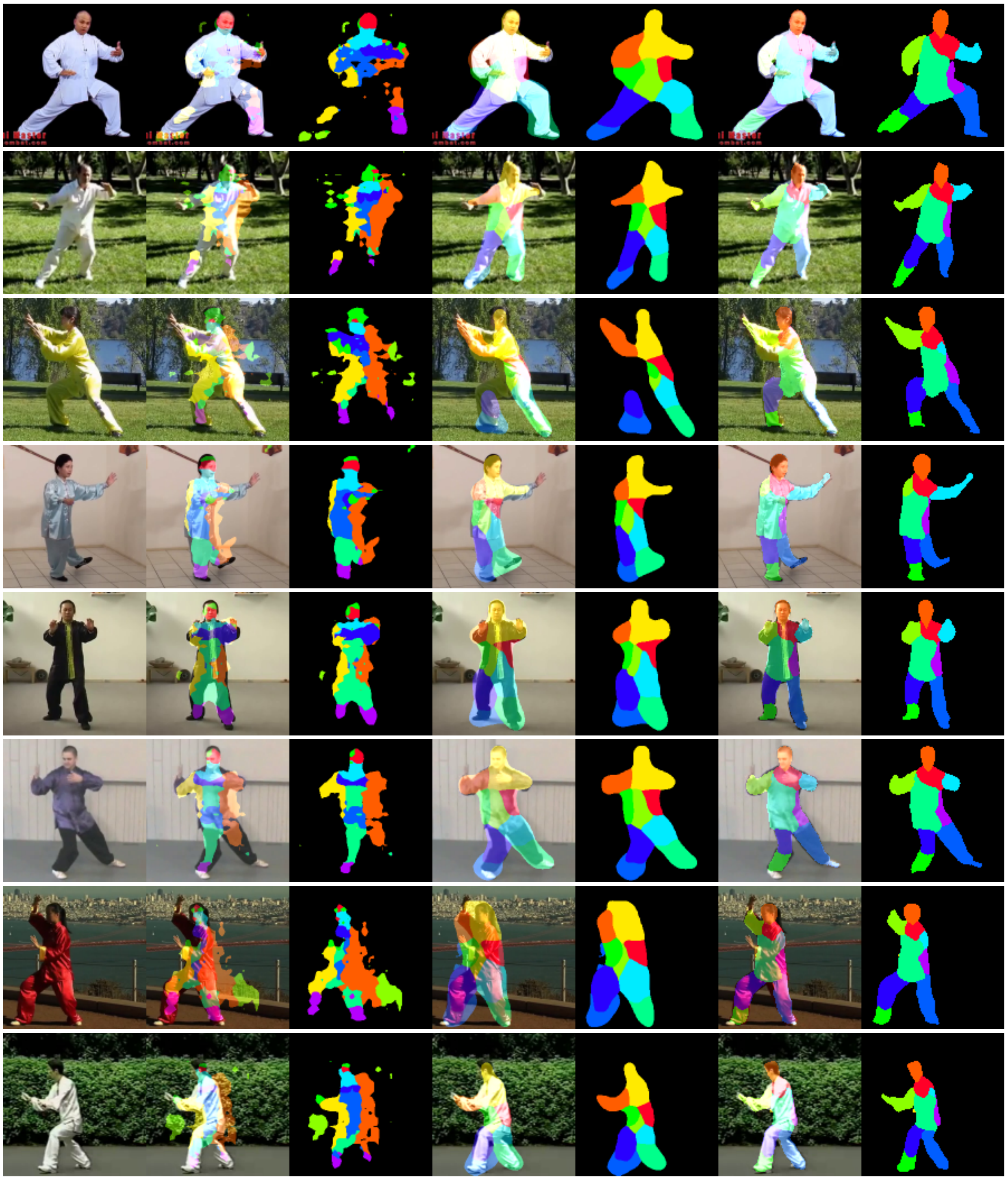

Figure 2. Additional results and comparisons on Tai-Chi-HD

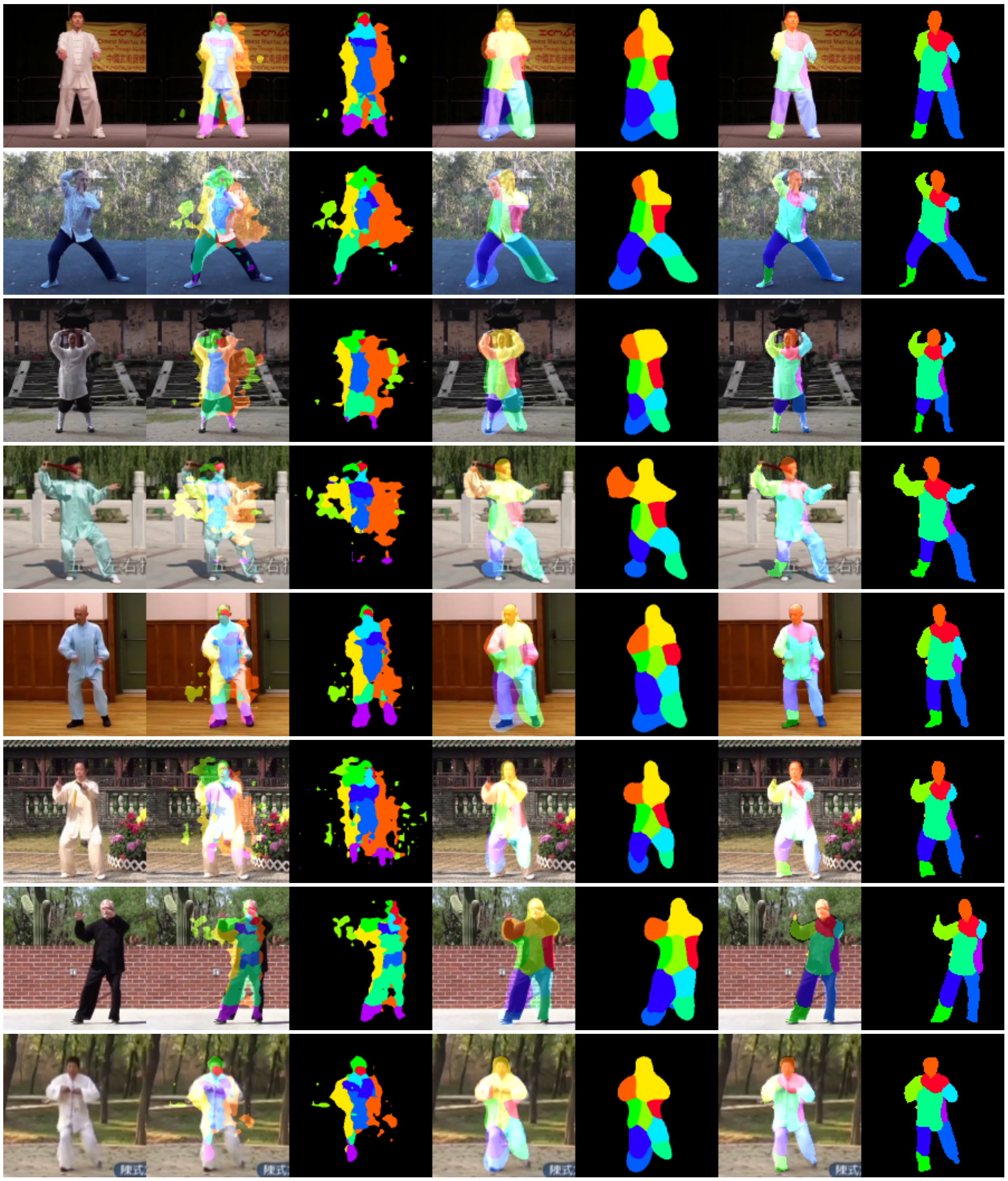

Figure 3. Additional results and comparisons on Tai-Chi-HD (cont.)

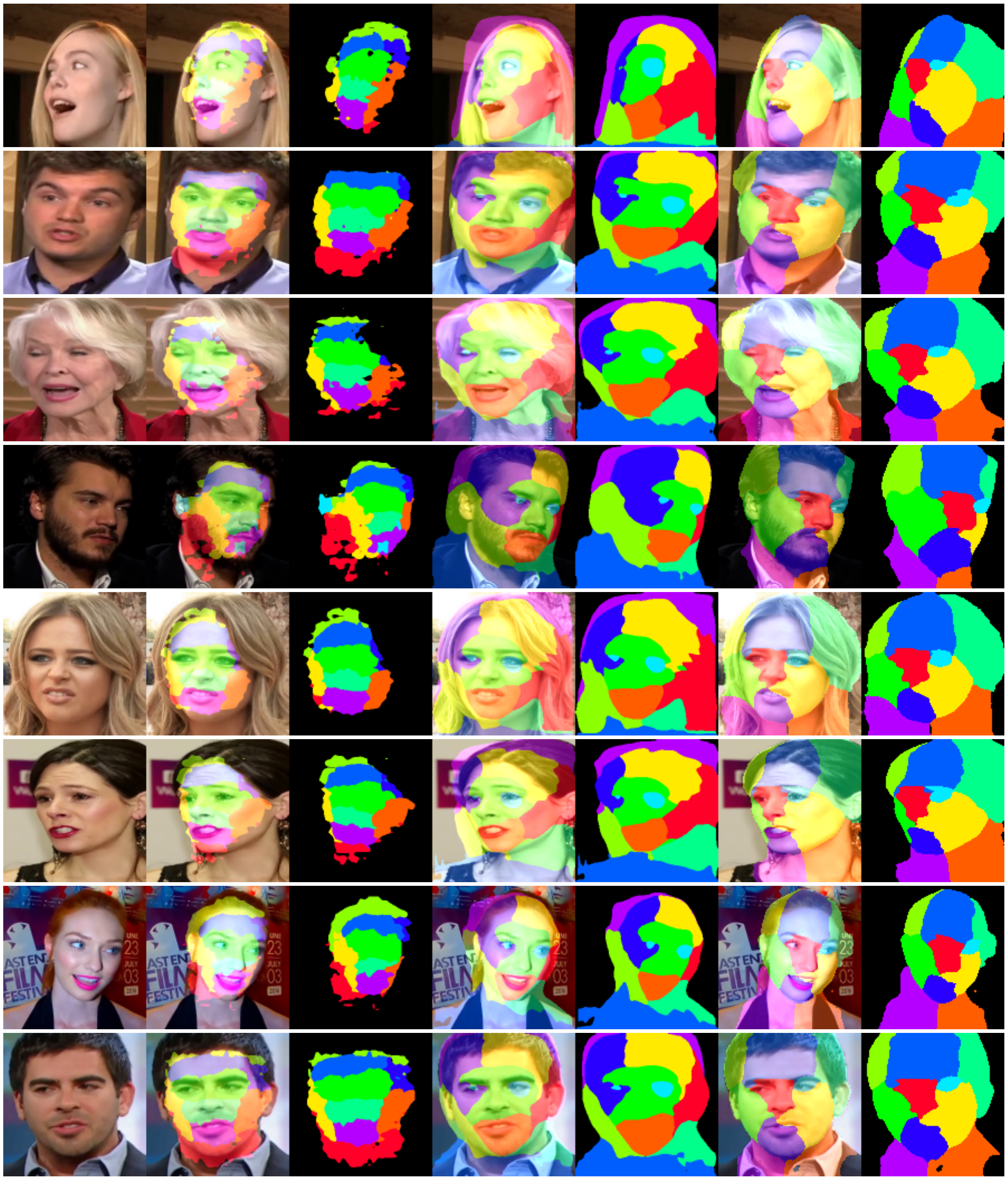

Figure 4. Additional results and comparisons on VoxCeleb

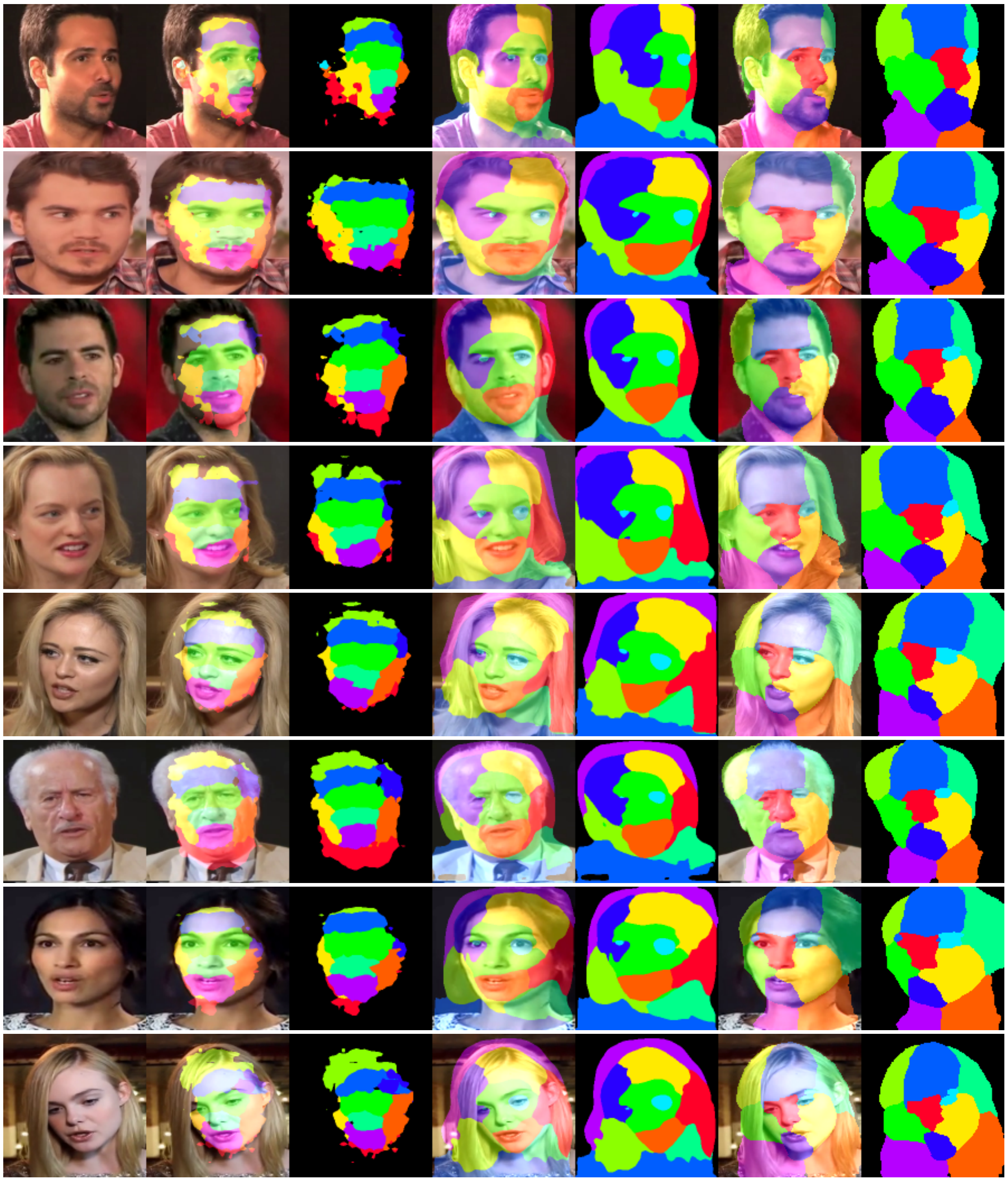

Figure 5. Additional results and comparisons on VoxCeleb (cont.)

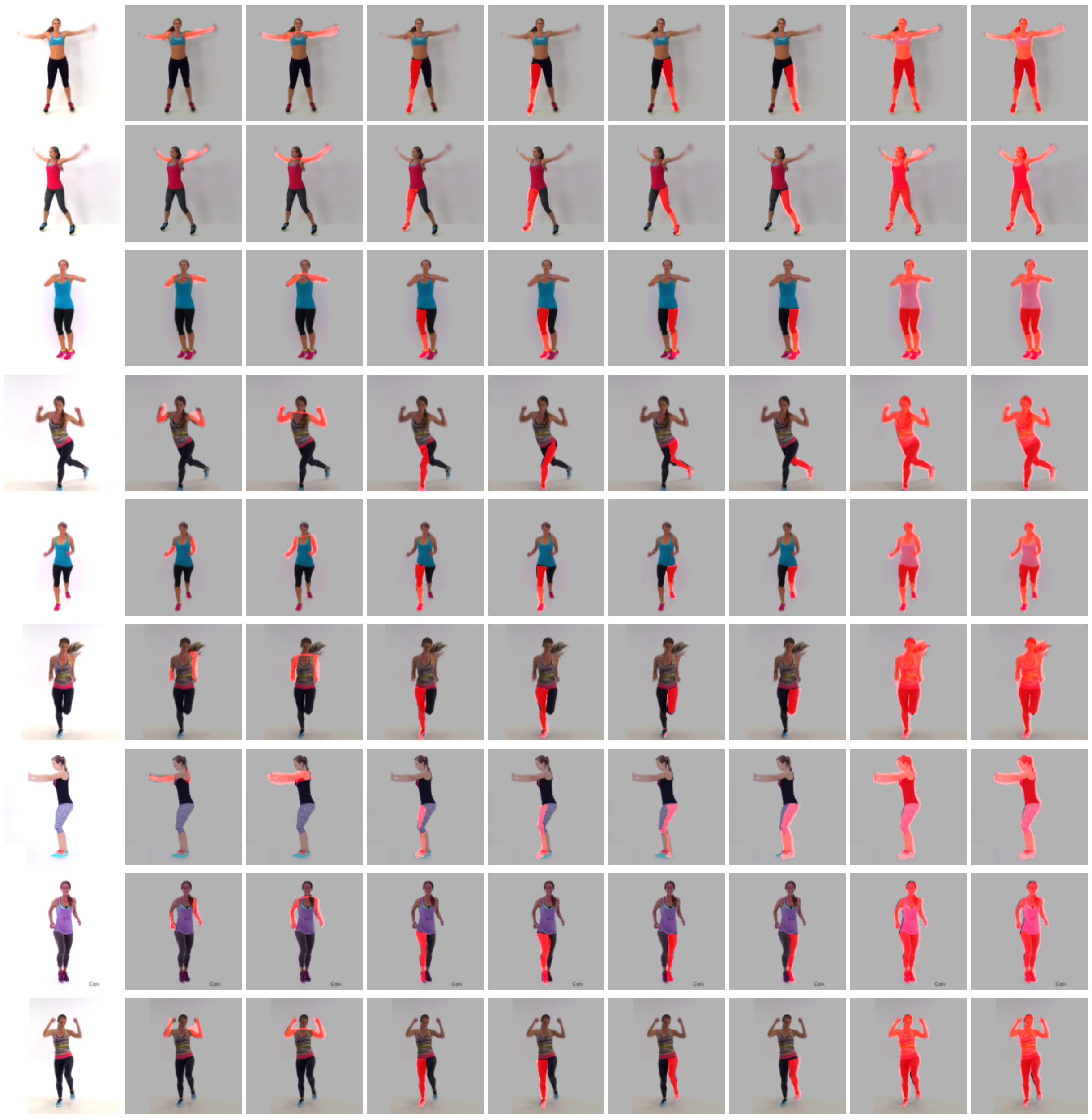

Figure 6. Segmentation results on Exercise dataset. We show both our results and the ground-truth (GT) segmentation provided by the dataset. The part masks are superimposed on the input images.

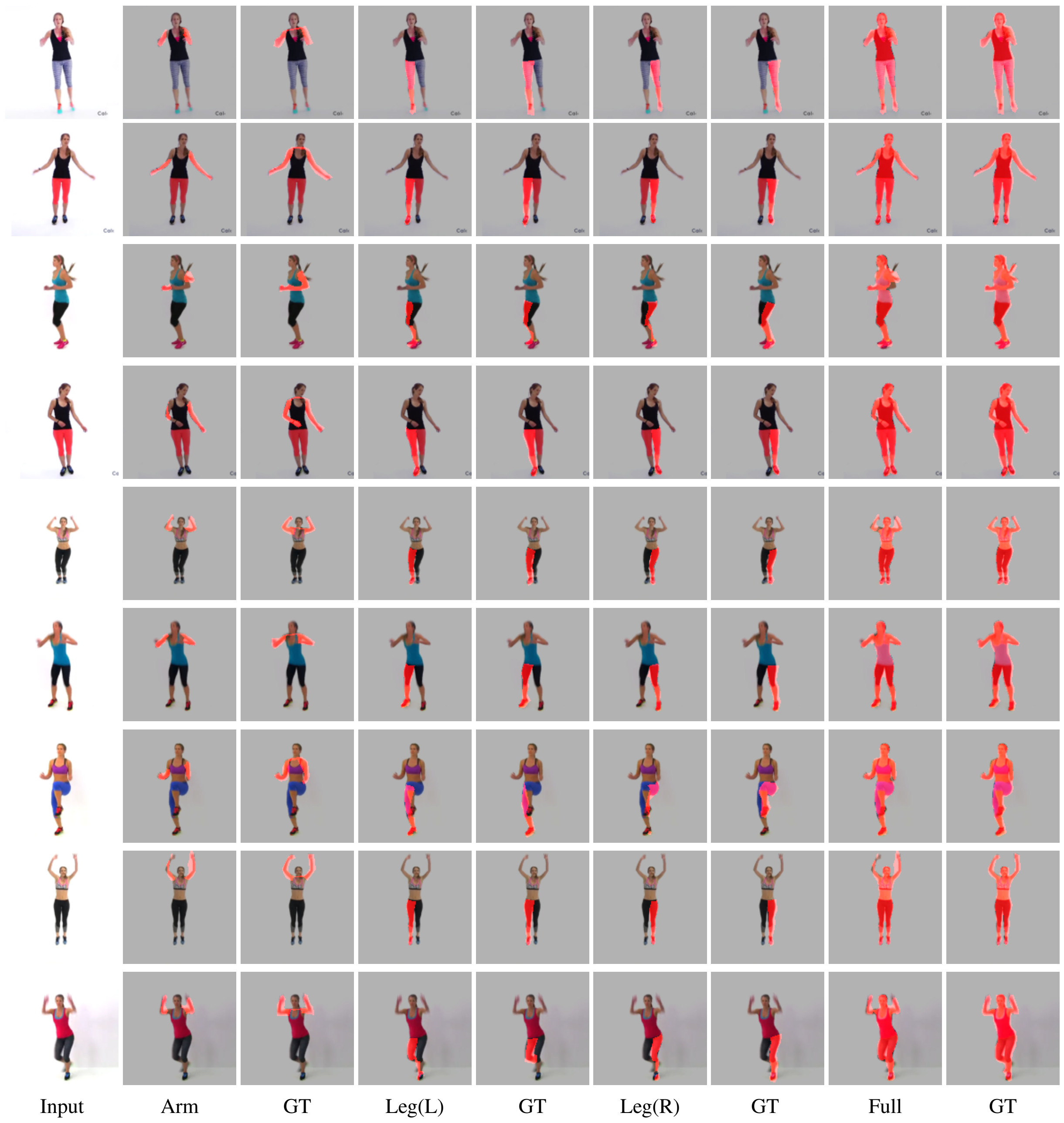

Figure 7. Segmentation results on Exercise dataset (cont.). We show both our results and the ground-truth (GT) segmentation provided by the dataset. The part masks are superimposed on the input images.

References

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Johnson, J., Alahi, A., and Fei-Fei, L. Perceptual losses for real-time style transfer and super-resolution. In *Proceedings of the European Conference on Computer Vision*, pp. 694–711, 2016.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Wang, T.-C., Liu, M.-Y., Zhu, J.-Y., Tao, A., Kautz, J., and Catanzaro, B. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.