Skeleton-Aware Networks for Deep Motion Retargeting

KFIR ABERMAN*
PEIZHUO LI*
DANI LISCHINSKI
OLGA SORKINE-HORNUNG
DANIEL COHEN-OR
BAOQUAN CHEN

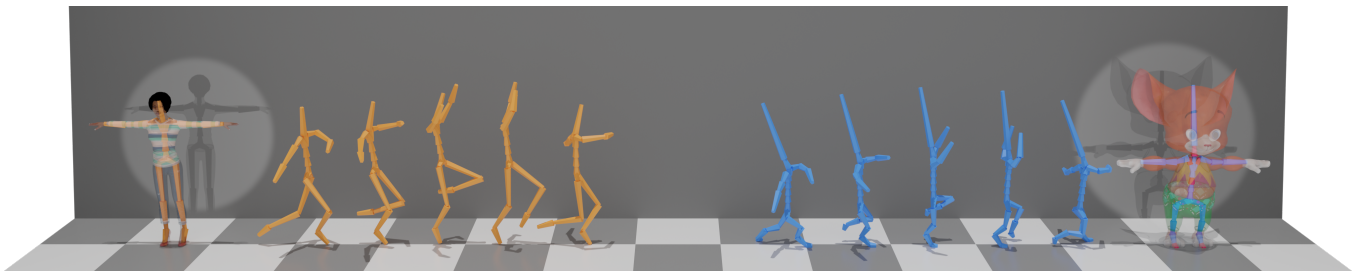

Fig. 1. Unpaired, cross-structural, motion retargeting. An input motion sequence (orange skeletons) is retargeted to a target skeleton (rightmost, blue), which has different body proportions, as well as a different number of bones (marked in red).

We introduce a novel deep learning framework for data-driven motion retargeting between skeletons, which may have different structure, yet corresponding to homeomorphic graphs. Importantly, our approach learns how to retarget without requiring any explicit pairing between the motions in the training set.

We leverage the fact that different homeomorphic skeletons may be reduced to a common *primal skeleton* by a sequence of edge merging operations, which we refer to as *skeletal pooling*. Thus, our main technical contribution is the introduction of novel differentiable convolution, pooling, and unpooling operators. These operators are *skeleton-aware*, meaning that they explicitly account for the skeleton's hierarchical structure and joint adjacency, and together they serve to transform the original motion into a collection of deep temporal features associated with the joints of the primal skeleton. In other words, our operators form the building blocks of a new deep motion processing framework that embeds the motion into a common latent space, shared by a collection of homeomorphic skeletons. Thus, retargeting can be achieved simply by encoding to, and decoding from this latent space.

Our experiments show the effectiveness of our framework for motion retargeting, as well as motion processing in general, compared to existing approaches. Our approach is also quantitatively evaluated on a synthetic dataset that contains pairs of motions applied to different skeletons. To the best of our knowledge, our method is the first to perform retargeting between skeletons with differently sampled kinematic chains, without any paired examples.

CCS Concepts: • **Computing methodologies** → **Motion processing**; **Neural networks**.

Additional Key Words and Phrases: Neural motion processing, motion retargeting

ACM Reference Format:
Kfir Aberman, Peizhuo Li, Dani Lischinski, Olga Sorkine-Hornung, Daniel Cohen-Or, and Baoquan Chen. 2020. Skeleton-Aware Networks for Deep Motion Retargeting. *ACM Trans. Graph.* 39, 4, Article 62 (July 2020), 14 pages. https://doi.org/10.1145/3386569.3392462

1 INTRODUCTION

Capturing the motion of humans is a fundamental task in motion analysis, computer animation, and human-computer interaction. Motion capture (MoCap) systems typically require the performer to wear a set of markers, whose positions are sampled by magnetic or optical sensors, yielding a temporal sequence of 3D skeleton poses. Since different MoCap setups involve different marker configurations and make use of different software, the captured skeletons may differ in their structure and number of joints, in addition to differences in bone lengths and proportions, corresponding to different captured individuals. Thus, *motion retargeting* is necessary, not only for transferring captured motion from one articulated character to another, within the same MoCap setup, but also across different setups. The latter is also essential for using data from multiple different motion datasets in order to train *universal*, setup-agnostic, data-driven models, for various motion processing tasks.

*equal contribution

Authors' addresses: Kfir Aberman, kfiraberman@gmail.com; Peizhuo Li, peizhuo@pku.edu.cn; Dani Lischinski, danix3d@gmail.com; Olga Sorkine-Hornung, sorkine@inf.ethz.ch; Daniel Cohen-Or, cohenor@gmail.com; Baoquan Chen, baoquan@pku.edu.cn.

0730-0301/2020/7-ART62 $15.00
https://doi.org/10.1145/3386569.3392462

Deep neural networks, which revolutionized the state-of-the-art for many computer vision tasks, leverage the regular grid representation of images and video, which is well suited for convolution and pooling operations. Unlike images, skeletons of different characters exhibit irregular connectivity. Furthermore, the structure of a skeleton is typically hierarchical. These differences suggest that the existing operators commonly used in CNNs might not be the best choice for analysis and synthesis of articulated motion.

In this paper, we introduce a new motion processing framework consisting of a representation for motion of articulated skeletons, designed for deep learning, and several differentiable operators, including convolution, pooling and unpooling, that operate on this representation. The operators are *skeleton-aware*, which means that they explicitly account for the skeleton structure (hierarchy and joint adjacency). These operators constitute the building blocks of a new deep framework, where the shallower layers learn local, low-level, correlations between joint rotations, and the deeper layers learn higher-level correlation between body parts.

The proposed motion processing framework can be useful for various motion analysis and synthesis learning based tasks. In this work, we focus on the task of motion retargeting between skeletons that have the same set of end-effectors, but might differ in the number of joints along the kinematic chains from the root to these end effectors. Such skeletons may be represented by homeomorphic (topologically equivalent) graphs.

Although motion retargeting is a long-standing problem, current approaches cannot automatically perform retargeting between skeletons that differ in their structure or the number of joints [Villegas et al. 2018]. In this scenario, correspondences between the different skeletons should be manually specified, often resulting in unavoidable retargeting errors. Animators must then manually correct such errors by manipulating key frames, which is a highly tedious process.

We treat the retargeting problem as a multimodal translation between unpaired domains, where each domain contains a collection of motions performed by skeletons with different proportions and bone lengths that share a specific skeletal structure (same set of kinematic chains). Thus, all of the motions in a given domain may be represented using the same graph. Different domains contain skeletons with different structure, i.e., represented by different graphs; however, the graphs are assumed to be homeomorphic.

Previous work has demonstrated that multimodal unpaired image translation tasks may be carried out effectively using a shared latent space [Gonzalez-Garcia et al. 2018; Huang et al. 2018]. In these works, same-sized images from different domains are embedded in a shared space that represents, for example, the content of the image, disentangled from its style. On images, such an embedding is straightforward to carry out using standard convolution and pooling operators; however, this is not the case for skeletons with different structures. In this work, we utilize our skeleton-aware motion processing framework, in particular *skeletal pooling*, to embed motions performed by different skeletons into a shared latent domain space.

Our key idea exploits the fact that different, yet homeomorphic, skeletons may be reduced to a common *primal skeleton*, which may be viewed as the common ancestor of all the different skeletons in the training data, by merging pairs of adjacent edges/armatures. Through interleaving of skeletal convolution and pooling layers, the shared latent space consists of a collection of deep temporal features, associated with the joints of the primal skeleton. The latent space is jointly learned by an encoder-decoder pair for each skeletal structure (domain).

In addition, we exploit our deep motion representation to disentangle the motion properties from the shape properties of the skeleton, which allows us to perform motion retargeting using a simple algorithm in our deep feature space. Similarly to the raw, low-level, representation of motion, which consists of static (joint offsets) and dynamic (joint rotations) components, our deep motion features are also split into static and dynamic parts. However, in the raw input motion, the two components are strongly coupled: a specific sequence of joint rotations is bound to a specific bone length and skeleton structure. In contrast, our encoders learn to decouple them: the dynamic part of the latent code becomes skeleton-agnostic, while the static one corresponds to the common primal skeleton. This property of the latent space makes it possible to retarget motion from skeleton A to skeleton B simply by feeding the latent code produced by the encoder of A into the decoder of B.

In summary, our two main contributions in this work are:

(1) A new motion processing framework that consists of a deep motion representation and differentiable skeleton-aware convolution, pooling, and unpooling operators.
(2) A new architecture for unpaired motion retargeting between topologically equivalent skeletons that may have different number of joints.

In addition to presenting the first automatic method for retargeting between skeletons with different structure, without any paired examples, we also demonstrate the effectiveness of our new deep motion processing framework for motion denoising (Section 4.4). We evaluate our motion retargeting approach and compare it to existing approaches in Section 6. A synthetic dataset, which contains pairs of motions applied to different skeletons, is used to perform a quantitative evaluation of our method.

2 RELATED WORK

2.1 Motion Retargeting

In their pioneering work, Gleicher et al. [1998] tackled character motion retargeting by formulating a spacetime optimization problem with kinematic constraints, and solving it for the entire motion sequence. Lee and Shin [1999] explored a different approach, which first applies inverse kinematics (IK) at each frame to satisfy constraints, and then smooths the resulting motion by fitting multilevel B-spline curves. The online retargeting method of Choi and Ko [2000] performs IK at each frame and computes the change in joint angles corresponding to the change in end-effector positions, while ensuring that the high-frequency details of the original motion are well preserved. In their physically-based motion retargeting filter, Tak and Ko [2005] make use of dynamics constraints to achieve physically plausible motions. Feng et al. [2012] proposed heuristics that can map arbitrary joints to canonical ones, and describe an algorithm that enables to instill a set of behaviors onto an arbitrary humanoid skeleton.

Classical motion retargeting approaches, such as the one mentioned above, rely on optimization with hand-crafted kinematic constraints for particular motions, and involve simplifying assumptions. Increased availability of captured motion data has made data-driven approaches more attractive. Delhaisse et al. [2017] describe a method for transferring the learnt latent representation of motions from one robot to another. Jang et al. [2018] train a deep autoencoder, whose latent space is optimized to create the desired change in bone lengths. These methods require paired training data.

Inspired by methods for unpaired image-to-image translation [Zhu et al. 2017], Villegas et al. [2018] propose a recurrent neural network architecture with a Forward Kinematics layer and cycle consistency based adversarial training objective for motion retargeting using unpaired training data. Lim et al. [2019] report better results for the unpaired setting by learning to disentangle pose and movement. Aberman et al. [2019] disentangle 2D human motion extracted from video into character-agnostic motion, view angle and skeleton, enabling retargeting of 2D motion while bypassing 3D reconstruction. All of these data-driven methods assume that the source and target articulated structures are the same.

Several works have explored retargeting human motion data to non-humanoid characters [Seol et al. 2013; Yamane et al. 2010]. In this scenario, the source and target skeletons may differ greatly from each other, however, the above approaches require captured motions of a human subject acting in the style of the target character. It is also necessary to select a few key poses from the captured motion sequence and match them to corresponding character poses [Yamane et al. 2010], or pair together corresponding motions [Seol et al. 2013], in order to learn the mapping. Celikcan et al. [2015] retarget human motion to arbitrary mesh models, rather than to skeletons. Similarly to the aforementioned approaches, learning such a mapping requires a number of pose-to-pose correspondences.

Abdul-Massih et al. [2017] argue that the motion style of a character may be represented by the motions of groups of body parts (GBPs). Thus, motion style retargeting may be done across skeleton structures by establishing a correspondence between GBPs, followed by constrained optimization to preserve the original motion. This requires defining the GBPs and the correspondence between them for each pair of characters.

Another loosely related problem is that of mesh deformation transfer. Earlier works, e.g., [Baran et al. 2009; Sumner and Popović 2004] require multiple correspondences between the meshes. The recent method of Gao et al. [2018] uses unpaired datasets to train a VAE-CycleGAN that learns a mapping between shape spaces. We also leverage adversarial training to avoid the need for paired data or correspondences, but the setting of our work is somewhat different, since we deal with temporal sequences of hierarchical articulated structures (the skeletons), rather than meshes.

2.2 Neural Motion Processing

Holden et al. [2016; 2015] were among the first to apply CNNs to 3D character animation. Motion is represented as a temporal sequence of 3D joint positions, and convolution kernels are local along the temporal dimension, but global in the joints dimension (the support includes all of the skeleton's joints). Thus, joint connectivity and the hierarchical structure of the skeleton are ignored.

Furthermore, representing motion using 3D joint positions doesn't fully describe motion, and requires IK to extract an animation. Pavllo et al. [2019] proposed QuaterNet, which processes joint rotations (quaternions), but performs forward kinematics on a skeleton to penalize joint positions, rather then angle errors. However, convolutional features extracted from joint rotations alone cannot fully capture 3D motion, since the same set of joint quaternions results in different poses when applied to different skeletons.

Since the skeleton of an articulated character may be represented as a graph, one might consider using graph convolutional networks (GCNs) to process motion data. In such networks, convolution filters are applied directly on graph nodes and their neighbors (e.g., [Bruna et al. 2013; Niepert et al. 2016]). Yan et al. [2018] propose a spatial-temporal GCN (ST-GCN) for skeleton-based action recognition, where spatial convolution filters are applied to the 1-neighbors of each node in the skeleton, and temporal convolution is applied on successive positions of each joint in time. In this approach, no pooling takes place in the spatial dimension, but only in the temporal one. Furthermore, the set of 1-neighbors of each node is split into several sets, according to a hand-crafted strategy, in order to assign each set with a learnable convolution weight. Ci et al. [2019] apply a Locally Connected Network (LCN), which generalizes both fully connected networks and GCNs, to tackle 3D pose estimation. This work does not address processing of 3D motion.

Another option, which is not convolution-based, is to perform learning on spatio-temporal graphs using deep RNNs [Jain et al. 2015]. Wang et al. [2019] propose a spatio-temporal RNN, where the skeletons are encoded into a latent space using a hierarchical neural network on the skeleton graph structure. The hierarchy is hand-crafted, while the weights of the fully-connected layers that merge different nodes together are learned. Thus, neither skeleton convolution, nor skeleton pooling takes place. The goals of this approach are also different from ours (prediction of motion, rather than its retargeting).

3 OVERVIEW

Our goal is to cope with the task of motion retargeting between skeletons that may have different structure, but are topologically equivalent. The key idea is to exploit the fact that topologically equivalent skeletons may be represented by homeomorphic graphs. A pair of such graphs may be reduced into a common minimal graph by eliminating nodes of degree 2 along linear branches, as illustrated in Figure 2. We refer to the skeleton represented by the reduced common graph as the *primal skeleton*.

This observation suggests encoding motions performed by different homeomorphic skeletons into a deep representation that is independent of the original skeletal structure or bone proportions. The resulting latent space is thus common to motions performed by skeletons with different structure, and we use it to learn data-driven motion retargeting, without requiring any paired training data. The retargeting process, depicted in Figure 3, uses an encoder E_A trained on a domain of motions, performed by skeletons with the same structure, to encode the source motion into the common latent space.

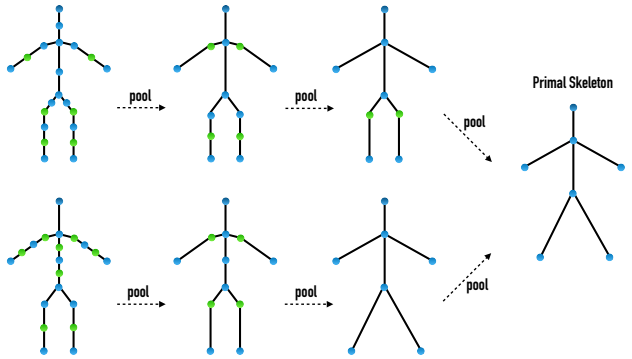

Fig. 2. Pooling to the primal skeleton. Our pooling operator removes nodes of degree two (green points) and merges their adjacent edges. After a few pooling steps the skeletal structures of different, topologically equivalent, skeletons are reduced into a common primal skeleton.

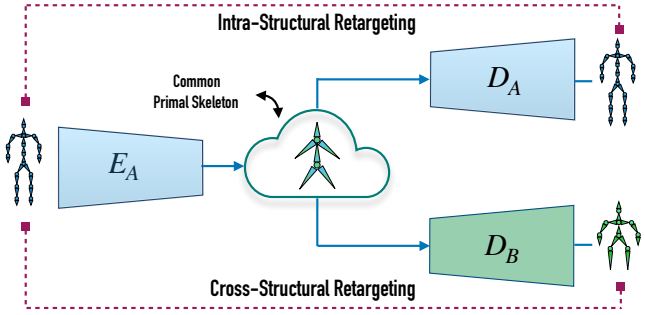

Fig. 3. Unpaired cross-structural motion retargeting. Our architecture encodes motions of different homeomorphic skeletons into a shared deep latent space, corresponding to a common primal skeleton. This representation can then be decoded into motions performed by skeletons within the same domain (intra-structural retargeting) or from another homeomorphic domain (cross-structural retargeting).

From this space, the latent representation may be decoded into a motion performed by a target skeleton. The target skeleton might have the same structure, but different bone lengths, in which case the decoding is done by a decoder D_A, trained on the same domain. However, using a decoder D_B trained on a different domain, the motion may also be retargeted across different skeletal structures.

In order to implement the approach outlined above, we introduce a new deep motion processing framework, consisting of two novel components:

(1) **Deep Motion Representation.** We represent a motion sequence as a temporal set of *armatures* that constitute a graph, where each armature is represented by a dynamic, time-dependent, feature vector (usually referred to as joint rotation) as well as a static, time-independent one (usually referred to as offset), as depicted in Figure 4. The static-dynamic structure is a common low-level representation in character animation, which our framework preserves along the processing chain. Specifically, we use two branches (static and dynamic) that convert the low-level information into a deep, static-dynamic representation of motion features.

(2) **Deep Skeletal Operators.** We define new differential operators that can be applied to animated skeletons. The operators are *skeleton-aware*, namely, the skeletal structure (hierarchy and joint adjacency) is considered by the operators. Concatenating these operators into an optimizable neural network enables the learning of deep temporal features that represent low-level, local joint correlations in shallow layers and high-level, global body part correlations in deeper layers.

Our motion processing framework, including the new representation and the skeleton-aware operators, is described in Section 4, while the architecture and loss functions that enable data-driven cross-structural motion retargeting are described in Section 5.

4 SKELETON-AWARE DEEP MOTION PROCESSING

Below we describe our motion processing framework, which consists of our motion representation, as well as our new skeletal convolution, pooling, and unpooling operators.

4.1 Motion Representation

Our motion representation for articulated characters is illustrated in Figure 4. A motion sequence of length T is described by a static component $\mathbf{S} \in \mathbb{R}^{J \times S}$ and a dynamic one $\mathbf{Q} \in \mathbb{R}^{T \times J \times Q}$, where J is the number of armatures, and S and Q are the dimensions of the static and the dynamic features, respectively (typically, $S = 3$ and $Q = 4$).

The static component $\mathbf{S}$ consists of a set of offsets (3D vectors), which describe the skeleton in some arbitrary initial pose, while the dynamic component $\mathbf{Q}$ specifies the temporal sequences of rotations of each joint (relative to its parent's coordinate frame in the kinematic chain), represented by unit quaternions. The root joint $\mathbf{R} \in \mathbb{R}^{T \times (S+Q)}$ is represented separately from the J armatures (its children), as a sequence of global translations and rotations (orientations).

The skeleton structure is represented by a tree graph whose nodes correspond to joints and end-effectors, while the edges correspond to armatures, as illustrated in Figure 4. Thus, for a skeleton with J armatures, the graph has $J+1$ nodes. Connectivity is determined by the kinematic chains (the paths from the root joint to the end-effectors) and expressed by adjacency lists $\mathcal{N}^d = \{\mathcal{N}_1^d, \mathcal{N}_2^d, \ldots, \mathcal{N}_J^d\}$, where $\mathcal{N}_i^d$ denotes the edges whose distance in the tree is equal or less than d from the i-th edge (see Figure 4).

4.2 Skeletal Convolution

We process motion in two parallel branches: a dynamic branch, which yields time-dependent deep features, and a static branch, which produces static deep features. Both branches share the property that their convolution kernels consider the skeleton structure to compute local features across armatures, as illustrated in Figure 5.

The dynamic branch performs *skeleto-temporal* convolutions, using kernels with local support both along the armature and the temporal axes, as illustrated in Figure 6. Note that while kernels across the time axis are temporally-invariant (namely, the kernel weights are shared along time), they are not shared across different armatures. This is due to the fact that different body parts move in

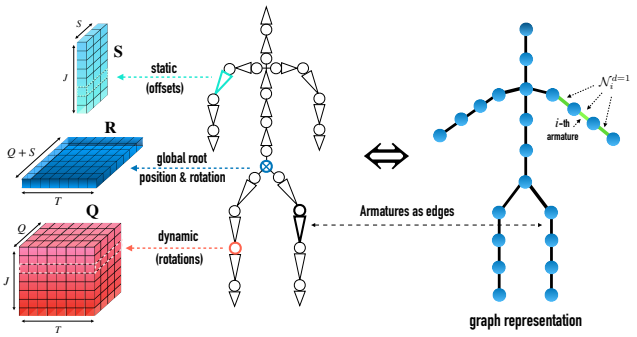

Fig. 4. We represent a motion clip (with temporal length T) of an articulated character by a set of J armatures (middle), each described by a static $\mathbf{S}$ (cyan) and a dynamic $\mathbf{Q}$ (red) feature vector. Additionally, we store a sequence of global root positions and orientations $\mathbf{R}$ (blue). The skeleton is represented as a graph (right), where each edge i corresponds to an armature with an adjacency list $\mathcal{N}_i^d$, defined based on the kinematic chain.

different patterns; thus, the extracted features in each part might be unique. For instance, it is plausible to expect that the features extracted by kernels that are centered at the spine joint might be different than those centered at the knee.

Since motion is fully described by a combination of static and dynamic features, it is important for the convolution kernels to consider both components during computation. Thus, we tile the static representation $\mathbf{S}$ along the temporal axis and concatenate the result to $\mathbf{Q}$ along the channels axis, to yield $\mathbf{M} \in \mathbb{R}^{T \times J \times (Q+S)}$.

In practice, a skeleto-temporal convolution in the dynamic branch with local support d is applied at every armature via

$$\hat{\mathbf{Q}}_i = \frac{1}{|\mathcal{N}_i^d|} \sum_{j \in \mathcal{N}_i^d} \mathbf{M}_j * \mathbf{W}_j^i + \mathbf{b}_j^i, \tag{1}$$

where $\mathbf{M}_j \in \mathbb{R}^{T \times (Q+S)}$ represents the features of the j-th armature, $\mathbf{W}_j^i \in \mathbb{R}^{k \times (Q+S) \times K}$ and $\mathbf{b}_j^i \in \mathbb{R}^K$ are K learned filters with temporal support k, and $*$ denotes a temporal, one dimensional convolution. Note that the number of armatures before and after convolution is preserved. Figure 6 illustrates two skeletal convolution kernels (red and blue), where each is applied to a different group of neighbor armatures.

In contrast to the dynamic branch, the static branch takes only the static feature matrix $\mathbf{S}$ as input, while ignoring the dynamic part. This is a choice that we make to ensure that the static components of the resulting deep feature spaces depend only on the structure of the skeleton and not on any particular motion. Thus, the static convolution operator may be viewed as a special case of the dynamic one in (1), with $Q = 0$ (i.e., S input channels), $T = 1$ and $k = 1$. In practice, this is a matrix multiplication operator.

Both the static and the dynamic branches share the connectivity map $\mathcal{N}^d$, which enables us to recursively apply skeletal convolution to motion sequences while maintaining dimensional and semantic consistency between the branches.

Note that the root, which is treated as a special armature (with two dynamic parts: global positions and global rotations, is convolved

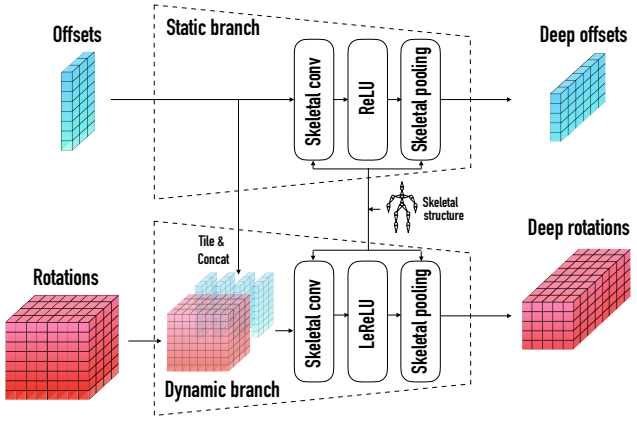

Fig. 5. Our framework performs skeletal convolution, activation, and skeletal pooling using blocks consisting of two parallel branches, dynamic and static. The dynamic branch takes tiled and concatenated static features as part of the input to its skeleto-temporal convolution layer. The static branch operates only on the static features.

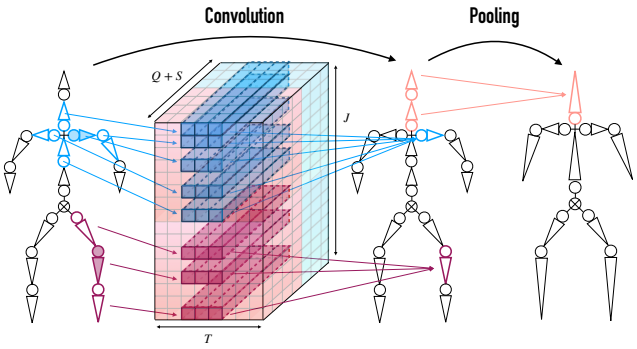

Fig. 6. Skeletal Convolution and Pooling. The skeleto-temporal convolution kernels (e.g., blue and purple) have local support. Support is contiguous along the time axis; for armatures, the support of each kernel is dictated by the connectivity map and the distance (d=1) to the armature at which the kernel is "centered" (shown filled in the left skeleton). Weights along the time axis are temporally-invariant, but they are not shared across different armatures. The right skeleton shows the result of topology-preserving skeletal pooling that merges features of pairs of consecutive armatures into single ones.

by a kernel whose support contains the closest armatures (up to distance d), as well as the end effectors. The support is chosen in this manner due to the existing low-level correlation between the global root motion to the local motion of end effectors, as can be clearly observed during running and walking, for example. This connection enables the global information to be injected into the dynamic features of deeper layers.

4.3 Topology Preserving Skeletal Pooling

In order to enable our skeleton-aware network to learn higher level, deep skeleto-temporal features, we next define pooling over armatures, which is inspired by the MeshCNN framework [Hanocka et al. 2019], which merges mesh edges, while pooling their deep features.

In general, pooling encourages the network to learn an efficient basis (kernels) that enables it to extract features of lower dimension, which are optimized to satisfy a specified loss function. For pooling on regular data such as temporal signals or images, adjacency is inherently implied by the signal structure, and the kernel size determines the pooling region. In that case, the features in the corresponding region are merged (usually by averaging or taking the maximum) into a smaller uniform grid, where adjacency is also well-defined.

There are various ways to define pooling on armatures. Our pooling is *topology-preserving*, meaning that the pooled skeleton (graph), which contains fewer armatures (edges), is homeomorphic, to the input one. Specifically, our pooling operator removes nodes of degree 2, by merging the features of their adjacent edges. This definition is motivated in the next section.

Pooling is applied to skeletal branches with a consecutive sequence of edges that connect nodes of degree 2, where the pooling regions are disjoint sets of edges $\{P_1, P_2, \ldots, P_{\tilde{J}}\}$, whose size is not larger than p. A sequence of N edges will be split into $\lfloor \frac{N}{p} \rfloor$ sets of p edges and another set of size $N - p\lfloor \frac{N}{p} \rfloor$, in the case that N is not divisible by p. We select the remainder set to be the one that is closest to the root. In practice, since sequential branches in human skeletons are short, we consistently use $p = 2$.

The skeletal pooling is applied to both the static and dynamic feature activations, and formally given by

$$\hat{\mathbf{S}}_i = \text{pool}\{\mathbf{S}_j \mid j \in P_i\} \quad \text{and} \quad \hat{\mathbf{Q}}_i = \text{pool}\{\mathbf{Q}_j \mid j \in P_i\}, \qquad (2)$$

where pool can be either max or average. The skeletal pooling operator over the armature axis is illustrated in Figure 6. It can be seen, for example, that the sequential branch from the neck to the head (marked in red), which contains two armatures is pooled into a single armature in the resulting skeleton. Our pooling can be intuitively interpreted as an operation that enables the network to learn a deep skeleton, with fewer armatures, which approximates the motion of the original skeleton. Note that in the dynamic branch a standard downsampling is also additionally applied to the temporal axis.

Unpooling. The unpooling operator is the counterpart of pooling, increasing the resolution of the feature activations without increasing the information. In our case, unpooling is performed based on the recorded structure of the prior pooled skeleton. We expand the number of edges (armatures) by copying the feature activations of each edge that is originated by a merging of two edges in the corresponding pooling step. Since unpooling does not have learnable parameters, it is usually combined with convolutions to recover the original resolution lost in the pooling operation. Note that in the dynamic branch standard upsampling is additionally applied to the temporal axis.

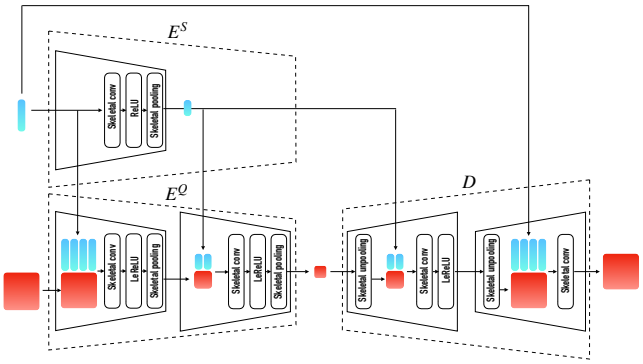

Fig. 7. An autoencoder with two skeletal blocks in the encoder and in the decoder. The static features are encoded and concatenated into both downsampling and upsampling blocks.

4.4 Evaluation

Our framework can be useful for various learning-based motion processing tasks. We next evaluate the building blocks of our framework against those proposed by Holden et al. [2016], who introduced a deep learning framework for motion editing. Their building blocks consist of standard 1D temporal convolutions, with a full support over the channel (joint) axis and pooling operators which are performed only on the temporal axis.

In order to evaluate the effectiveness of the two frameworks, we implemented two autoencoders, which share the same number of components and type of layers. In the first, we used the standard convolution and pooling proposed Holden et al. [2016], while in the second we used our skeleton-aware operators. Figure 7 depicts a diagram of our autoencoder, that contains a set of static and dynamic encoders (E^Q and E^C, respectively), and a decoder D. Details about the number of input/output channels in each layer are given in the Appendix A.

Both autoencoders are trained with a single reconstruction loss (ℓ_2 norm), using our dataset described in Section 6.

For a fair comparison, both autoencoders were trained on joint rotations, represented by unit quaternions (although in the original paper, Holden et al. [2016] use joint positions to represent motion). However, in order to avoid error accumulation along the kinematic chains, the reconstruction loss is applied to the corresponding joint positions, obtained from the rotations by forward kinematics (FK).

During training, each of the autoencoders learns a latent space that represents a motion manifold: a continuous space of natural motions; thus, motion denoising can be performed by simply projecting a noisy motion onto the manifold, using the trained encoder, and then decoding it back to space-time.

We evaluate the performance of the autoencoders by measuring the reconstruction (ℓ_2 loss) on a test set of unseen motions, where the inputs are injected with two types of noise: (i) White Gaussian noise ($\mu = 0$, $\sigma = 0.01$) (2) random zeros: we randomly select pairs of joints and frames and overwrite the existing values with zeros (simulating MoCap glitches).

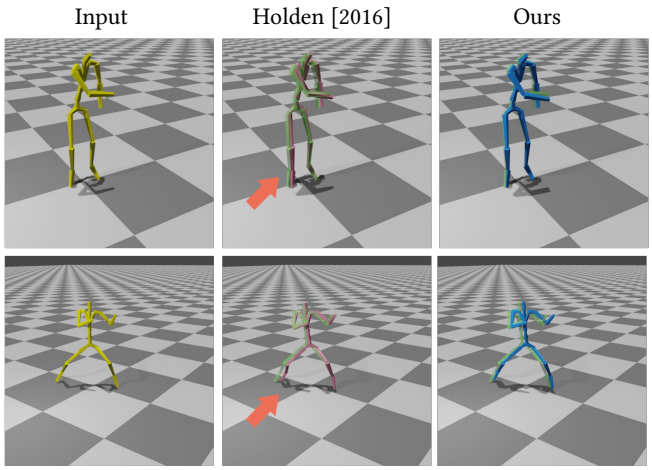

Fig. 8. Motion denoising performance comparison. We compare our skeleton-aware operators to the operators used by Holden et al. [2016]. Input (left), Holden et al. [2016] (middle), ours (right). Ground truth is overlaid (green skeleton).

Table 1. Quantitative comparison between our method and that of Holden et al. [2016] on a denoising task, where two different types of noise are applied to the inputs. We report the average error over all joints and all motions, normalized by the skeleton's height (multiplied by 10^3, for clarity).

	Holden et al. [2016]	Ours
White noise	1.08	0.74
Random zeros	1.08	0.81

Figure 8 shows frames extracted from the video sequences that can be found in the supplementary material, and Table 1 reports a quantitative comparison between the methods. It can be seen that our skeleton-aware operators achieve better performance for both noise types. As can be observed in the video, our results demonstrate smaller local errors in joint positions and also better global positions, as well as stability. The conventional operators ignore the skeleton structure, while ours pool and compress the information in a more structured manner.

5 CROSS-STRUCTURAL MOTION RETARGETING

Motion retargeting is not a precisely defined task. When a source motion is manually retargeted by animators to a target character, they usually attempt to achieve two main goals: first, the resulting motion should appear natural and visually plausible for the target character, while closely resembling the original source motion. Second, that joint positions satisfy perceptually sensitive constraints, such as foot and hand contact, typically achieved by applying IK optimization. Below, we explain how our framework enables unsupervised retargeting that follows the aforementioned rules.

5.1 Problem Setting

We formulate motion retargeting as an unpaired cross-domain translation task. Specifically, let $\mathcal{M}_A$ and $\mathcal{M}_B$ denote two motion domains, where the motions in each domain are performed by skeletons with the same skeletal structure ($\mathcal{S}_A$ and $\mathcal{S}_B$, respectively), but may have different bone lengths and proportions. This formulation fits existing public MoCap datasets, where each dataset contains different characters that share the skeletal structure and performing various motions. It is further assumed that a homeomorphism exists between the skeletal structures of $\mathcal{S}_A$ and $\mathcal{S}_B$. Note that the domains are unpaired, which means that there are no explicit pairs of motions (performed by different skeletons) across the two domains.

Let each motion $i \in \mathcal{M}_A$ be represented by the pair $(\mathbf{S}_A, \mathbf{Q}_A^i)$, where $\mathbf{S}_A \in \mathcal{S}_A$ is the set of skeleton offsets and $\mathbf{Q}_A^i$ are the joint rotations, as described in Section 4.1. Given the offsets of a target skeleton $\mathbf{S}_B \in \mathcal{S}_B$, our goal is to map $(\mathbf{S}_A, \mathbf{Q}_A^i)$ into a retargeted set of rotations $\tilde{\mathbf{Q}}_B^i$ that describe the motion as it should be performed by $\mathbf{S}_B$. Formally, we seek a data-driven mapping $G^{A\to B}$

$$G^{A\to B}\left((\mathbf{S}_A, \mathbf{Q}_A^i) \in \mathcal{M}_A,\ \mathbf{S}_B \in \mathcal{S}_B\right) \ \to\ (\mathbf{S}_B, \tilde{\mathbf{Q}}_B^i) \tag{3}$$

In our settings, the translation mapping $G^{A\to B}$ is learned concurrently with the mapping $G^{B\to A}$.

5.2 Network Architecture

Our architecture consists of encoders, decoders and discriminators, with an individual combination of an encoder $E_m = [E_m^Q, E_m^S]$, a decoder D_m, and a discriminator C_m is trained for each domain $\mathcal{M}_m,\ m \in \{A, B\}$. Here, E_m^Q is the dynamic encoder and E_m^S is the static one, as shown in Figure 7. Also, see Figure 9(a), which shows a higher level view of the information flow in our network.

Having trained the aforementioned components for each motion domain, the desired mapping $G^{A\to B}$ is obtained, at test time, by using the decoder D_B to combine the dynamic motion representation produced by E_A^Q with the static representation produced by E_S^B, as depicted in Figure 9(b). This is possible, since our encoders produce a deep encoding of motion, which is independent of the original skeletal properties, and the shared latent space is associated with a common primal skeletal structure. In other words, the encoders disentangle the low-level, correlated dynamic and static parameters, and the retargeting can be simply performed using their deep, disentangled representation.

Below, we describe the different losses used to train our network, also depicted in Figure 9(a). For simplicity, we denote the encoded dynamic features by $\bar{\mathbf{Q}}_A^i = E_A^Q(\mathbf{Q}_A^i, \mathbf{S}_A)$, $\bar{\mathbf{Q}}_B^i = E_B^Q(\mathbf{Q}_B^i, \mathbf{S}_B)$, and denote the set of static deep features, coupled with the input skeleton, by $\bar{\mathbf{S}}_A = \{E_A^S(\mathbf{S}_A), \mathbf{S}_A\}$, $\bar{\mathbf{S}}_B = \{E_B^S(\mathbf{S}_B), \mathbf{S}_B\}$. The latter notation simplifies the equations since the dynamic encoders and decoders receive the input skeleton as well as a set of deep skeletal features (which are concatenated along their deep layers), as can be seen in Figure 7. Note that our loss terms are described only for one direction ($A \to B$). The symmetric term is obtained by swapping the roles of A and B.

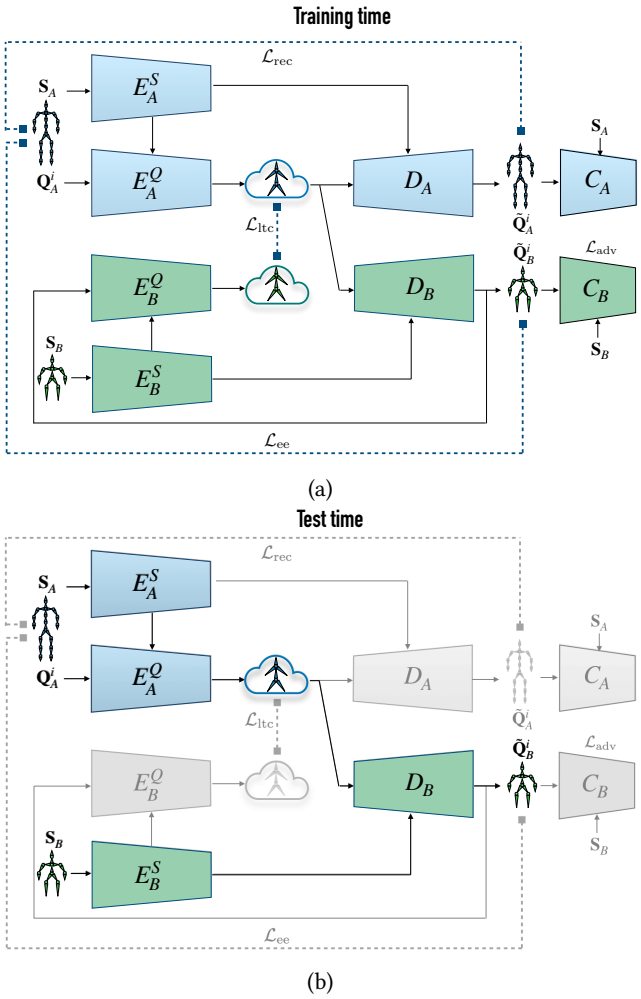

Fig. 9. Our cross-structural retargeting architecture. (a) Training time: Dynamic and static features from domain A, $(\mathbf{S}_A, \mathbf{Q}_A^i)$, are encoded by E_A^Q and E_A^S, respectively. Reconstruction by decoder D_A is enforced via $\mathcal{L}_{\text{rec}}$. Cross-structural motion translation is achieved by feeding the output of E_A^Q and E_B^S into D_B and applying the end effectors loss, $\mathcal{L}_{\text{ee}}$, along with latent consistency loss $\mathcal{L}_{\text{lts}}$ between the original latent representation produced by E_A^Q and that of the translated motion by E_B^Q. (b) Test time: Retargeting is performed by using D_B to combine motion encoded by E_A^Q with skeletal features encoded by E_B^S. Note that the diagrams describe only $A \rightarrow B$ retargeting, while in practice the training is symmetric.

Reconstruction Loss. To train an auto encoder $\left([E_A^Q, E_A^S], D_A\right)$ for motions in the same domain, we employ a standard reconstruction loss over the joint rotations and joint positions

$$\mathcal{L}_{\text{rec}} = \mathbb{E}_{(\mathbf{S}_A, \mathbf{Q}_A^i)\sim\mathcal{M}_A}\left[\left\|(D_A(\bar{\mathbf{Q}}_A^i, \bar{\mathbf{S}}_A), \mathbf{S}_A) - \mathbf{Q}_A^i\right\|^2\right] \quad (4)$$

$$+ \mathbb{E}_{(\mathbf{S}_A, \mathbf{Q}_A^i)\sim\mathcal{M}_A}\left[\left\|\text{FK}(D_A(\bar{\mathbf{Q}}_A^i, \bar{\mathbf{S}}_A), \mathbf{S}_A) - \mathbf{P}_A^i\right\|^2\right], \quad (5)$$

where FK is a forward kinematic operator that returns the joint positions (given rotations and a skeleton) and $\mathbf{P}_A^i = \text{FK}(\mathbf{Q}_A^i, \mathbf{S}_A)$ are the joint positions of the input character. Note that the positions are normalized by the height (feet to head) of the character. The presence of the loss over the joint positions prevents accumulaiton of error along the kinematic chain [Pavllo et al. 2019].

Latent Consistency Loss. As mentioned earlier, our skeletal pooling operator enables embedding motions of homeomorphic skeletons into a common deep primal skeleton latent space, by pooling the features of consecutive armatures, as illustrated in Figure 2.

Embedding samples from different domains in a shared latent space has proved to be efficient for multimodal image translation tasks [Gonzalez-Garcia et al. 2018; Huang et al. 2018]. Constraints may be applied directly on this intermediate representation, facilitating disentanglement. Inspired by this, we apply a latent consistency loss to the shared representation to ensure that the retargeted motion $\tilde{\mathbf{Q}}_B^i$ retains the same dynamic features as the original clip:

$$\mathcal{L}_{\text{ltc}} = \mathbb{E}_{(\mathbf{S}_A, \mathbf{Q}_A^i)\sim\mathcal{M}_A}\left[\left\|E_B^Q(\tilde{\mathbf{Q}}_B^i, \bar{\mathbf{S}}_B) - E_A^Q(\mathbf{Q}_A^i, \bar{\mathbf{S}}_A)\right\|_1\right], \quad (6)$$

where $\|\cdot\|_1$ is the L_1 norm.

Adversarial Loss. Since our data is unpaired, the retargeted motion has no ground truth to be compared with. Thus, we use an adversarial loss, where a discriminator C_B assesses whether or not the decoded temporal set of rotations $\tilde{\mathbf{Q}}_B^i$ appears to be a plausible motion for skeleton $\mathbf{S}_B$:

$$\mathcal{L}_{\text{adv}} = \mathbb{E}_{i\sim\mathcal{M}_A}\left[\|C_B(\tilde{\mathbf{Q}}_B^i, \bar{\mathbf{S}}_B)\|^2\right] + \mathbb{E}_{j\sim\mathcal{M}_B}\left[\|1 - C_B(\mathbf{Q}_B^j, \bar{\mathbf{S}}_B)\|^2\right]. \quad (7)$$

As in other generative adversarial networks, the discriminator C_B is trained using the motions in $\mathcal{M}_B$ as the real examples, and the output of $G^{A\rightarrow B}$ as the fake ones.

End-Effectors Loss. While homeomorphic skeletons may differ in the number of joints, they share the same set of end-effectors. We exploit this property to require that the end-effectors of the original and the retargeted skeleton have the same normalized velocity. The normalization is required since velocities may be at different scales for different characters. This requirement is particularly helpful to avoid common retargeting artifacts, such as foot sliding; frames with zero foot velocity in the input motion should result in zero foot velocity in the retargeted motion. This is formulated by

$$\mathcal{L}_{\text{ee}} = \mathbb{E}_{i\sim\mathcal{M}_A}\sum_{e\in\mathcal{E}}\left\|\frac{V_{A_e}^i}{h_{A_e}} - \frac{V_{B_e}^i}{h_{B_e}}\right\|^2, \quad (8)$$

where $V_{A_e}^i$ and $V_{B_e}^i$ are the magnitudes of the velocity of the e-th end-effector of skeletons $\mathbf{S}_A$ and $\mathbf{S}_B$, respectively, while performing motion i, $\mathcal{E}$ is the set of end-effectors, and h_{A_e}, h_{B_e} are the lengths of the kinematic chains from the root to the end-effector e, in each of the skeletons $\mathbf{S}_A$ and $\mathbf{S}_B$.

Although our end-effectors loss significantly mitigates foot sliding artifacts, we further clean foot contact using standard Inverse Kinematics (IK) optimization. The cleanup is fully automatic: we extract binary foot contact labels from the motion input sequence, and apply IK to enforce foot contact by fixing the position of the foot to the average position along contact time slots. The effects of

our end-effectors loss and the IK-based cleanup, are demonstrated in supplementary video.

The full loss used for training combines the above loss terms:

$$\mathcal{L} = \mathcal{L}_{\text{rec}} + \lambda_{\text{ltc}}\mathcal{L}_{\text{ltc}} + \lambda_{\text{adv}}\mathcal{L}_{\text{adv}} + \lambda_{\text{ee}}\mathcal{L}_{\text{ee}} \tag{9}$$

where $\lambda_{\text{ltc}} = 1$, $\lambda_{\text{adv}} = 0.25$, $\lambda_{\text{ee}} = 2$.

6 EXPERIMENTS AND EVALUATIONS

In this section we evaluate our results, compare them to other retargeting methods, and demonstrate the efficiency of various components in our framework. In order to qualitatively evaluate our results, please refer to the supplementary video.

6.1 Implementation Details

Our motion processing framework is implemented in PyTorch, and the experiments are performed on a PC equipped by an NVIDIA GeForce GTX Titan Xp GPU (12 GB) and Intel Core i7-695X/3.0GHz CPU (16 GB RAM). We optimize the parameters of our network, with the loss term in (9), using the Adam optimizer [Kingma and Ba 2014]. Training our skeleton-aware network takes about 22 hours (around 5000 epochs), however, the performance can be improved with a parallel implementation of our operators.

In order to evaluate our method, we construct a dataset with 2400 motion sequences, performed by 29 distinct characters, from the Mixamo 3D characters collection [Adobe Systems Inc. 2018]. For each motion, we randomly choose a single character to perform it, to ensure that our dataset contains no motion pairs. Furthermore, to enable training the network in batches, motions are trimmed into fixed temporal windows with $T = 64$ frames each. However, note that our network is fully-convolutional, so there is no limitation on the length of the temporal sequence at test time.

The characters can be divided into two groups, A and B, each containing skeletons with similar structure but different body proportions. This configuration fits our problem setting and also enables us to quantitatively evaluate our method against a ground truth. Note that due to imbalance in the Mixamo character collection, group A contains 24 characters, while group B contains only 5 (the characters in group B are the only characters with this skeletal structure in the original dataset). Our framework assumes that all of the characters contain 5 main limbs (2 hands, 2 feet, and head), thus, we clip the fingers of the Mixamo characters.

6.2 Intra-Structural Retargeting

Our framework can be used also for retargeting of skeletons with the same structure, but different proportions, where a single translator (encoder-decoder) is used to perform the task. In this experiment we compare our method to other methods that perform intra-structural retargeting. We use group A to train the network on 20 characters and test it on 4 (unseen) ones.

The first method we compare to is Neural Kinematic Networks (NKN) of Villegas et al. [2018], a recurrent neural network architecture with a Forward Kinematics layer to cope with intra-structural motion retargeting. Another state-of-the-art method we compare to is PMnet, by Lim et al. [2019], which uses a two-branch CNN to perform unsupervised motion retargeting for skeletons with similar structures. In addition, since retargeting in our settings corresponds to an unpaired translation, we also compare to a naive adaptation of CycleGAN [Zhu et al. 2017] to the motion domain. In this implementation, each of the domains contains a set of motions that are performed by a distinct character. Two generators and two discriminators (each consisting of a set of 1D temporal convolutions) are trained to translate motions between two domains. An adversarial loss and a cycle consistency loss are used to perform the training. We train such a system for each pair of characters (domains). Each motion is represented as a temporal set of joint rotations (static information is not considered, since there is a single skeleton in each domain). Note that (by definition) the characters are seen, thus, no unseen evaluation is performed on this network. The full implementation details are described in the Appendix A.

Figure 10 shows a few frames from the comparison, which is included in the supplementary video. It can be seen that our results (right column) are more stable than those of the other methods and aligned better with the ground truth (a green skeleton, overlaid on top of each output), in terms of global positioning and local errors of joints.

Another naive baseline for motion retargeting is obtained by simply using the unmodified joint rotations of the source motion with the target skeleton. In practice, in IK based intra-structure retargeting, the source rotations are used to initialize the optimized rotation values, which are then tuned based on contact constraints that depends on the content of the motion. In the cross-structural setting, where there is no one-to-one joint correspondence, the mapping is defined manually, which causes unavoidable retargeting errors (due to missing corresponding joints), that should be manually corrected.

Since in both cases, manual intervention is required (specifying motion-dependent constraints and/or joint correspondence) this baseline is not scalable. Thus, we use a naive copy of rotations as a baseline instead, only for the intra-structural retargeting scenario.

Table 2 presents a quantitative evaluation of the different methods described above. Errors are measured by performing retargeting on each pair in the test set (6 pairs totally) over all the $N = 106$ test motions and comparing to the ground truth, available from the original Mixamo dataset. Since (in the cross-structural case) we may deal with a different number of joints (per domain) we calculate the retargeting error of every target skeleton k, to all the others in the test set C, as the average distance between joint positions, which is given by

$$E^k = \frac{1}{NJ(|C|-1)h_k} \sum_{c \in C, c \neq k} \sum_{i=1}^{N} \sum_{j=1}^{J} \|\tilde{\mathbf{P}}^j_{ik,c} - \mathbf{P}^j_{ik}\|, \tag{10}$$

where $\tilde{\mathbf{P}}^j_{ik,c}$ denotes the j-th joint position of the retargeted motion sequence i that was originally performed by skeleton c and transferred to skeleton k, h_k is the height of skeleton k and $\mathbf{P}^j_{ik}$ is the ground truth. The final error is calculated by averaging the errors E_k for all the skeletons $k \in C$. The results reported in Table 2 demonstrate that our method outperforms the other approaches in intra-structural retargeting.

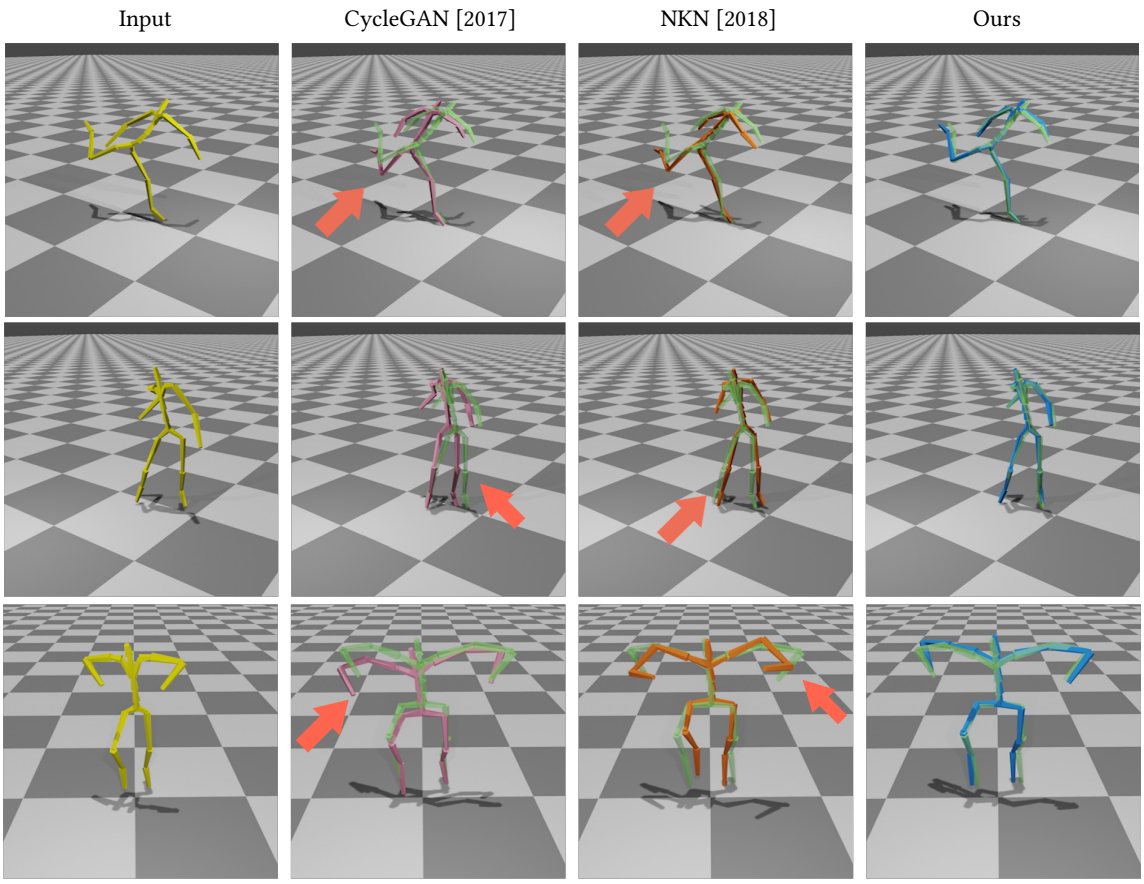

Fig. 10. Intra-structure motion retargeting. Our method is compared to a naive adaptation of CycleGAN [Zhu et al. 2017] to the motion domain and to NKN of Villegas et al. [2018]. The outputs are overlaid with the ground truth (green skeleton). It can be seen that our results are more stable and better aligned with the ground truth. The full comparison can be seen in the supplementary video.

Table 2. Quantitative comparison between our method to the method of Villegas et al. [2018], a naive adaptation of CycleGAN [2017] to unpaired motion translation, and an ablation study to evaluate the various components and loss terms in our framework. We report the average error over all joints and all motions, normalized by the skeleton's height (multiplied by 10^3, for clarity).

	Intra-Structural	Cross-Structural
Villegas et al. [2018] (NKN)	6.24	243
Lim et al. [2019] (PMnet)	5.72	N/A
CycleGAN [2017] adaptation	7.66	8.97
Copy rotations	8.86	N/A
Ours - conventional operators	3.95	3.56
Ours - no shared latent space	3.01	3.06
Ours - no $\mathcal{L}_{adv}$	**0.47**	3.81
Ours - full approach	2.76	**2.25**

6.3 Cross-Structural Retargeting

We next compare our method in a cross-structural setting, between the two groups A and B. In practice, group B originally contains an extra joint in each leg, and another one in the neck (compared to A), and we further increase the existing difference by adding more joints (one in the spine, and one in each arm), by splitting each of the affected offsets into two equal parts. Since group B contains only 5 characters we use 4 of them to train the model and only 1 to test, while in group A, we keep the same setup.

Since there are no existing methods that can cope with the task of cross-strtucral retargeting in a fully automatic manner, we naively adapt previous approaches to support a cross-structural setup. Here, we use again the CycleGAN [Zhu et al. 2017] motion translator, but modify the number of input and output channels in each translator to match the number of joints in each domain, as well as the number of input channels of the discriminators accordingly.

In the case of NKN [Villegas et al. 2018], which uses same-structure skeletons (single domain), we modified the original implementation, to support two different domains. However, since the number of joints is different between the domains, NKN's reconstruction loss

had to be removed. In addition, since the domains have different dimensions, the two networks ($A \rightarrow B$, $A \rightarrow B$) cannot share weights, so they had to be trained separately.

Both the qualitative results in the supplementary video (see extracted frames in Figure 11), and the quantitative comparison in Table 2, show that our method outperforms these two alternatives. It can be seen that our cross-structural error is, in fact, lower than the intra-structural one, which might come as a surprise; this is due to the fact that the set in group B is much smaller, with smaller differences in their body proportions, making the single test character closer to the ones seen during training.

As our comparison shows, the reconstruction loss in NKN [Villegas et al. 2018] plays a key role in the good performance that the original system achieves for intra-structural retargeting, and without this loss term, the quality of their results is significantly degraded.

Special Characters. In the supplementary video we demonstrate a few examples of characters with special properties, like asymmetric limbs (missing bone in the leg or arm, see Figure 12), and a character with an extra bone in each arm (see Figure 13). Since each of them has a unique skeletal structure, no other skeletons belong to their domain. Thus, in order to perform motion retargeting to other skeletons, we train a special model for each of the examples which translates skeletons from group A to the specific skeleton (and vice versa). It can be seen that our model can learn to retarget skeletons with various structures, as long as they are homeomorphic, including non-trivial motions like clapping hands (see Figure 13), which is a very challenging motion to retarget, especially when the lengths of the arms of the source and target skeletons are different. In general, in order to retarget such a motion, the user must manually specify the contact time-positions, and to solve an IK optimization with rotations that are initialized to the rotations of the source character. In the example that appears in the supplementary video, it may be seen that the initial solution achieved by copying the rotations is not satisfactory (due to a missing joint correspondence).

6.4 Ablation Study

In this part we evaluate the impact on performance of our operators, shared latent space, and the various loss terms. The results are reported in Table 2 and demonstrated some parts in the supplementary video.

Skeleton-Aware Operators. To evaluate the effectiveness of our skeleton-aware operators within the retargeting framework, we perform a comparison where all of them are replaced by conventional ones. The skeletal-convolution is replaced by a standard 1D convolution with a full support over the channels (joints) axis, and the skeletal pooling and unpooling are discarded and upsampling and downsampling is performed only on the temporal axis. In order to still support the structure of a shared latent space, we simply modify the number of output and input channels in each encoder and decoder, respectively, such that both autoencoders share the same latent space size, which equals to the one in the original system (see Appendix A for more details).

As reported in Table 2, although the conventional operators are inferior to the skeleton-aware ones, they still outperform the baselines. We attribute this to the fact that in our approach the latent space is structured. Thanks to the local armature support of our skeletal convolutions, different kinematic chains are encoded into distinct parts in the latent representation, from which they can be decoded into the target chains by different decoders. For example, an arm with 2 bones and an arm with 3-bones, will be encoded into the same set of channels in the shared representation, and decoded by the relevant encoder.

Shared Latent Space. In this experiment we retrain our framework without a shared latent space. This is done by replacing the latent consistency loss $\mathcal{L}_{\text{ltc}}$ with a full cycle consistency loss on the input. The results in Table 2 show that the performance of this variant is worse. Without a shared latent space the encoder-decoder can be interpreted as a general translator that translate motion from one domain to another. While this might work well enough for unimodal tasks (where each source point can be mapped into a single target point), multimodal tasks stand more to gain from having a shared latent domain, as already demonstrated in multimodal image translation tasks, e.g., [Huang et al. 2018].

Adversarial Loss. In this experiment we discard the adversarial loss $\mathcal{L}_{\text{adv}}$ and retrain our network. It can be seen that in the intra-structural setting omitting the adversarial loss actually improves performance, while in the cross-structural retargeting this is not the case. The main reason, in our opinion, is that intra-structural retargeting is a simpler task, where reasonable results may be achieved by copying the rotations and constraining end-effectors. Similarly, the network can easily learn a simple mapping that copies the input rotation values and tunes them using the end-effectors loss, and achieve good results in this manner.

In contrast, in a cross-structural setting, when the joint correspondence between the two domains is not well defined, $\mathcal{L}_{\text{adv}}$ is necessary to assist the network in learning how to transfer motions between the two domains.

End-Effectors Loss. In this experiment we show the effectiveness of our end-effectors loss, by discarding $\mathcal{L}_{\text{ee}}$ and retraining our network. It is well known that deep networks for character animation, which are not physically-based may output motions with severe foot-sliding artifacts [Holden et al. 2016], where the main cause is the natural smoothness of the temporal filters and rotation errors that are accumulated along the kinematic chains.

Without $\mathcal{L}_{\text{ee}}$, when no special attention is given to end-effectors, artifacts can be clearly observed in the outputs (see the supplementary video). The foot positions at contact times are characterized by large errors, which appear like slides. However, when the model is trained with our end-effectors loss, the large errors are mitigated and converted to subtle motions with high-frequency that are concentrated at a fixed point during contact time. These errors, which are less perceptually noticeable, can be easily fixed using a simple, fully-automated, IK based foot-sliding cleanup, as described in Section 5 and demonstrated in the supplementary video. In addition, Figure 14 shows the magnitude of the velocity of the right foot as a function of time in the motion clip used for the ablation study of

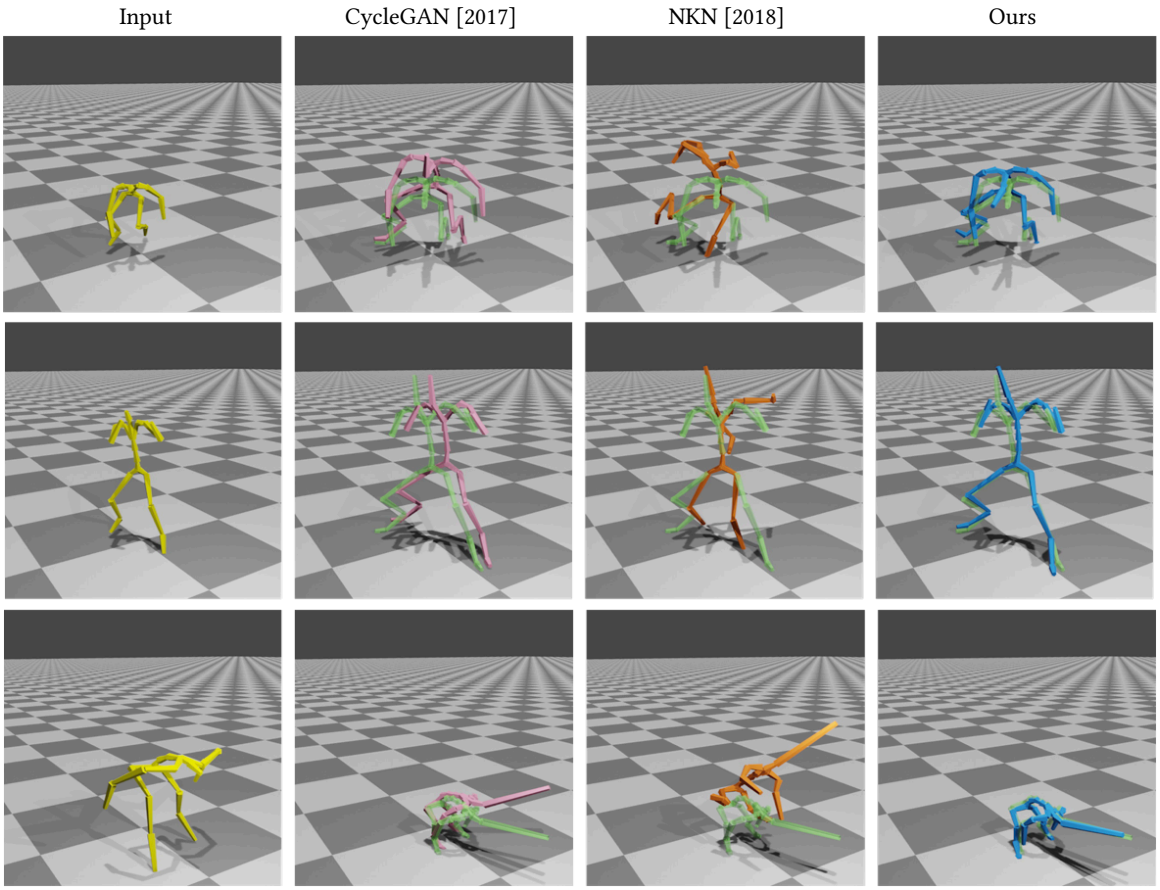

Fig. 11. Cross-structure motion retargeting. Our method is compared to a naive adaptation of CycleGAN [Zhu et al. 2017] to the motion domain and to a cross-structural version of NKN of Villegas et al. [2018]. The outputs are overlaid with the ground truth (green skeleton). The full comparison can be seen in the supplementary video.

$\mathcal{L}_{\text{ee}}$ (can be found in the supplementary video 05:18-05:22). It may be seen that without IK (green), the magnitude of the velocity is small but still noisy, characterized by high-frequencies. However, after performing IK (orange), the output is more correlated with the ground truth (blue) and cleaner zero velocity slots can be observed during contact periods.

7 DISCUSSION AND FUTURE WORK

We have presented a framework where networks are trained to encode sequences of animations of homeomorphic skeletons to a common latent space, and to decode them back, effectively allowing transferring motions between the different skeletons. The success of defining a common encoding space for different skeletons is attributed to three reasons: (i) the primal skeleton is topologically close to the original source skeletons, (ii) it carries enough deep geometric features, which is weakly decoupled from the source skeleton and (iii) we use spatially-variant kernels for convolving the appropriate joint neighborhoods.

It is tempting to believe that the primal skeleton could have been reduced down to a single point, and hence potentially allowing to transfer motion among skeletons that are not necessarily homeomorphic. However, our preliminary experiments in that direction suggest that it is too hard to encapsulate such a large amount of information. Nevertheless, extending the scope of compatible skeletons is an interesting topic for future work.

A legitimate question is commonly asked: "does this really require a deep network solution?", or "what does the network learn here"? The answer is that, yes, the deep features that are encoded on the primal skeleton are learned. They are learned by training numerous sequences. The features required to encode an animation decoupled from its joints are overly complex to be modeled by design. Moreover, the deep features of the primal skeleton encode also the bones lengths. This compensates proportion difference between the source and the target skeletons. In other words, motion transfer applied by the network translates both the topology and geometry jointly. Professionals can map motions between pairs of skeletons, however, not only that it is a tedious task, but it often introduces errors that need to manually amended. On the other hand, training networks once, facilitates the transfer between pair of arbitrary sequences from the trained domain.

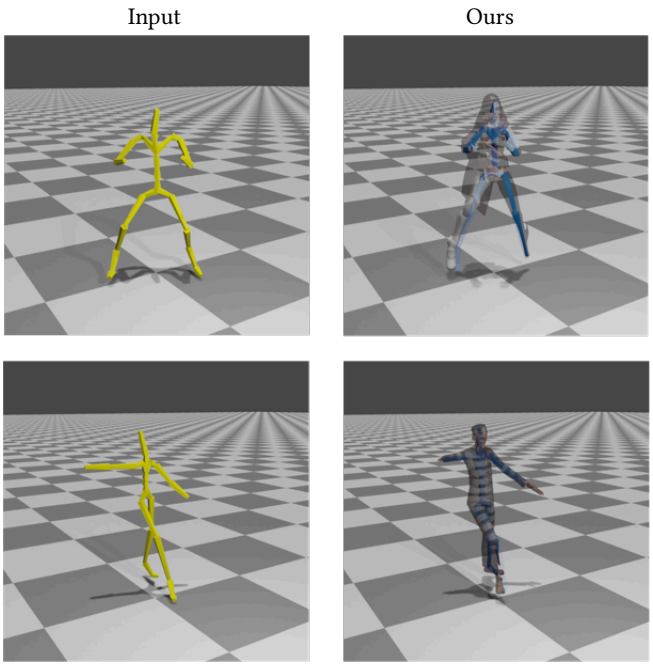

Fig. 12. Retargeting to asymmetric characters. This result shows that our method can deal with asymmetric characters and generates natural results. The full comparison can be seen in the supplementary video.

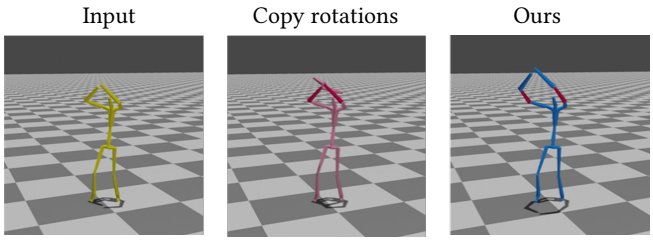

Fig. 13. Retargeting to a character with three bones per arm. Our method is compared to a naive nearest neighbor rotation copying. It can be seen that copying rotations causes implausible results, especially for fine motions like clapping. The full comparison can be seen in the supplementary video.

In the future, we would like to develop means that allows transfer between skeletons that are not homeomorphic. This possibly would require to define a generalized primal skeleton where the topology is disentangled and encoded separately.

A notable limitation is that we cannot retarget motion well between homeomorphic skeletons, if they move very differently from each other and have different T-poses. One such example, attempting to retarget the motion from a dog to a gorilla is included in the supplementary video. Because the joint rotations describing the motion are specified with respect to the T-pose, having very different T-poses, as shown in Figure 15, implies a completely different interpretation of the joint rotations, making the translation task much more challenging. Moreover, despite our use of the end-effectors loss, complex tasks that involve interactions with objects

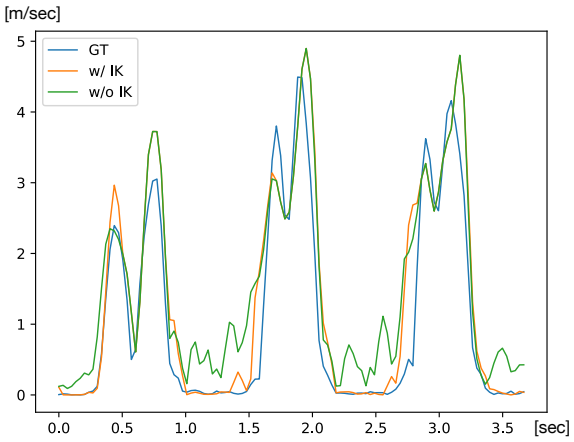

Fig. 14. Foot contact cleanup using IK. The magnitude of the foot velocity in the raw output of the network (green) and the post processed motion signal (orange) are compared to the one in the GT (blue).

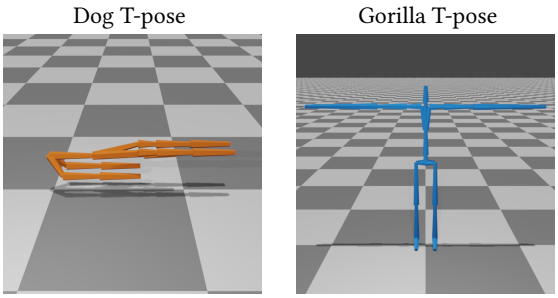

Fig. 15. Drastically different T-poses may hinder retargeting. See the resulting failure example in the supplementary video.

(like picking up a cup) will not be retargeted properly, if they were not seen in the dataset.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive comments. This work was supported in part by National Key R&D Program of China (2018YFB1403900, 2019YFF0302902), and by the Israel Science Foundation (grant no. 2366/16).

REFERENCES

Michel Abdul-Massih, Innfarn Yoo, and Bedrich Benes. 2017. Motion Style Retargeting to Characters With Different Morphologies. *Comp. Graph. Forum* 36 (2017), 86–99.

Kfir Aberman, Rundi Wu, Dani Lischinski, Baoquan Chen, and Daniel Cohen-Or. 2019. Learning Character-Agnostic Motion for Motion Retargeting in 2D. *ACM Trans. Graph.* 38, 4 (2019), 75.

Adobe Systems Inc. 2018. Mixamo. https://www.mixamo.com. https://www.mixamo.com Accessed: 2018-12-27.

Ilya Baran, Daniel Vlasic, Eitan Grinspun, and Jovan Popović. 2009. Semantic Deformation Transfer. *ACM Trans. Graph.* 28, 3, Article 36 (July 2009), 6 pages.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral Networks and Locally Connected Networks on Graphs. arXiv:cs.LG/1312.6203

Ufuk Celikcan, Ilker O. Yaz, and Tolga K. Çapin. 2015. Example-Based Retargeting of Human Motion to Arbitrary Mesh Models. *Comp. Graph. Forum* 34 (2015), 216–227.

Kwang-Jin Choi and Hyeong-Seok Ko. 2000. Online motion retargetting. *The Journal of Visualization and Computer Animation* 11, 5 (2000), 223–235.

Hai Ci, Chunyu Wang, Xiaoxuan Ma, and Yizhou Wang. 2019. Optimizing Network Structure for 3D Human Pose Estimation. In *Proc. ICCV 2019*.
Brian Delhaisse, Domingo Esteban, Leonel Dario Rozo, and Darwin G. Caldwell. 2017. Transfer learning of shared latent spaces between robots with similar kinematic structure. In *Proc. IJCNN 2017*. 4142–4149.
Andrew Feng, Yazhou Huang, Yuyu Xu, and Ari Shapiro. 2012. Automating the transfer of a generic set of behaviors onto a virtual character. In *International Conference on Motion in Games*. Springer, 134–145.
Lin Gao, Jie Yang, Yi-Ling Qiao, Yu-Kun Lai, Paul L. Rosin, Weiwei Xu, and Shihong Xia. 2018. Automatic unpaired shape deformation transfer. *ACM Trans. Graph.* 37 (2018), 237:1–237:15.
Michael Gleicher. 1998. Retargetting motion to new characters. In *Proc. 25th annual conference on computer graphics and interactive techniques*. ACM, 33–42.
Abel Gonzalez-Garcia, Joost van de Weijer, and Yoshua Bengio. 2018. Image-to-image translation for cross-domain disentanglement. In *Advances in Neural Information Processing Systems*. 1287–1298.
Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. 2019. MeshCNN: a network with an edge. *ACM Trans. Graph.* 38, 4 (2019), 1–12.
Daniel Holden, Jun Saito, and Taku Komura. 2016. A deep learning framework for character motion synthesis and editing. *ACM Trans. Graph.* 35, 4 (July 2016), 138:1–11.
Daniel Holden, Jun Saito, Taku Komura, and Thomas Joyce. 2015. Learning motion manifolds with convolutional autoencoders. In *SIGGRAPH Asia 2015 Technical Briefs*. ACM, 18:1–18:4.
Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. 2018. Multimodal unsupervised image-to-image translation. In *Proc. ECCV 2018*. 172–189.
Ashesh Jain, Amir Roshan Zamir, Silvio Savarese, and Ashutosh Saxena. 2015. Structural-RNN: Deep Learning on Spatio-Temporal Graphs. In *Proc. IEEE CVPR 2015*. 5308–5317.
Hanyoung Jang, Byungjun Kwon, Moonwon Yu, Seong Uk Kim, and Jongmin Kim. 2018. A variational U-Net for motion retargeting. In *SIGGRAPH Asia 2018 Posters*.
Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
Jehee Lee and Sung Yong Shin. 1999. A hierarchical approach to interactive motion editing for human-like figures. In *Proc. 26th annual conference on computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 39–48.
Jongin Lim, Hyung Jin Chang, and Jin Young Choi. 2019. PMnet: learning of disentangled pose and movement for unsupervised motion retargeting. In *Proc. BMVC*.
Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning Convolutional Neural Networks for Graphs. In *Proc. ICML 2016*, Vol. 48. 2014–2023.
Dario Pavllo, Christoph Feichtenhofer, Michael Auli, and David Grangier. 2019. Modeling Human Motion with Quaternion-Based Neural Networks. *International Journal of Computer Vision* (October 2019), 1–18.
Yeongho Seol, Carol O'Sullivan, and Jehee Lee. 2013. Creature features: online motion puppetry for non-human characters. In *Proc. SCA '13*.
Robert W. Sumner and Jovan Popović. 2004. Deformation Transfer for Triangle Meshes. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 399–405.
Seyoon Tak and Hyeong-Seok Ko. 2005. A physically-based motion retargeting filter. *ACM Trans. Graph.* 24, 1 (2005), 98–117.
Ruben Villegas, Jimei Yang, Duygu Ceylan, and Honglak Lee. 2018. Neural Kinematic Networks for Unsupervised Motion Retargetting. In *Proc. IEEE CVPR*. 8639–8648.
He Wang, Edmond S. L. Ho, Hubert P. H. Shum, and Zhanxing Zhu. 2019. Spatio-temporal Manifold Learning for Human Motions via Long-horizon Modeling. arXiv:cs.GR/1908.07214
Katsu Yamane, Yuka Ariki, and Jessica K. Hodgins. 2010. Animating non-humanoid characters with human motion data. In *Proc. SCA '10*.
Sijie Yan, Yuanjun Xiong, and Dahua Lin. 2018. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Thirty-Second AAAI Conference on Artificial Intelligence*.
Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proc. ICCV 2017*. 2223–2232.

A NETWORK ARCHITECTURES

In this section we describe the details for the network architectures.

Table 3 describes the architecture for our skeleton-aware network, where `Conv`, `LReLU`, `AP`, `UP` and `UpS` denote convolution, leaky ReLU, average skeletal pooling, skeletal unpooling and temporal linear upsampling layers, respectively. All of the convolution layers use reflected padding. k is the kernel width, s is the stride, and the number of input and output channels per joint, and number of input and output joint number is reported in the rightmost column.

Name	Layers	k	s	i/o_c	i/o_j
E_A^Q	Conv + LReLU + AP	15	2	4/8	28/18
	Conv + LReLU + AP	15	2	8/16	18/7
D_A^Q	UP + UpS + Conv + LReLU	15	1	16/8	7/18
	UP + UpS + Conv	15	1	8/4	18/28
E_A^S	Conv + LReLU + AP	1	1	3/8	28/18
E_B^Q	Conv + LReLU + AP	15	2	4/8	22/12
	Conv + LReLU + AP	15	2	8/16	12/7
D_B^Q	UP + UpS + Conv + LReLU	15	1	16/8	7/12
	UP + UpS + Conv	15	1	8/4	12/22
E_B^S	Conv + LReLU + AP	1	1	3/8	22/12

Table 3. Architecture for skeleton-aware network

Name	Layers	k	s	i/o_c
E_A^Q	Conv + LReLU	15	2	112/144
	Conv + LReLU	15	2	144/112
D_A^Q	UpS + Conv + LReLU	15	1	112/144
	UpS + Conv	15	1	144/112
E_A^S	Conv + LReLU	1	1	84/144
E_B^Q	Conv + LReLU	15	2	88/96
	Conv + LReLU	15	2	96/122
D_B^Q	UpS + Conv + LReLU	15	1	122/96
	UpS + Conv	15	1	96/88
E_B^S	Conv + LReLU	1	1	66/96

Table 4. Architecture for regular skeleton-unaware network

Table 4 describes the architecture of the regular skeleton-unaware network, where `Conv`, `LReLU`, and `UpS` denote 1-D temporal convolution, LeakyReLU, and temporal linear upsampling layers, respectively. It is implemented by achieving the same total number of channels (i.e., number of joints × channels per joint) as in the skeleton-aware network.

Discriminators C_A, C_B are implemented as patch-GAN discriminators. They share the same architecture as of E_A, E_B, except using sigmoid as the last activation.