Pacific Graphics 2018
H. Fu, A. Ghosh, and J. Kopf
(Guest Editors)

Volume 37 (2018), Number 7

Few-shot Learning of Homogeneous Human Locomotion Styles

I. Mason, S. Starke, H. Zhang, H. Bilen and T. Komura

University of Edinburgh

Abstract

Using neural networks for learning motion controllers from motion capture data is becoming popular due to the natural and smooth motions they can produce, the wide range of movements they can learn and their compactness once they are trained. Despite these advantages, these systems require large amounts of motion capture data for each new character or style of motion to be generated, and systems have to undergo lengthy retraining, and often reengineering, to get acceptable results. This can make the use of these systems impractical for animators and designers and solving this issue is an open and rather unexplored problem in computer graphics. In this paper we propose a transfer learning approach for adapting a learned neural network to characters that move in different styles from those on which the original neural network is trained. Given a pretrained character controller in the form of a Phase-Functioned Neural Network for locomotion, our system can quickly adapt the locomotion to novel styles using only a short motion clip as an example. We introduce a canonical polyadic tensor decomposition to reduce the amount of parameters required for learning from each new style, which both reduces the memory burden at runtime and facilitates learning from smaller quantities of data. We show that our system is suitable for learning stylized motions with few clips of motion data and synthesizing smooth motions in real-time.

CCS Concepts

•***Computing methodologies*** → ***Animation; Neural networks;*** *Motion capture;*

1. Introduction

Training neural networks from motion capture data is attracting researchers and engineers in computer graphics, computer animation and robotics due to their significant merits and proven ability to generate high quality animations from a high level user input such as a keyboard or gamepad. Such smooth and natural motions, that follow user instructions while also adapting to the terrain geometry, could be useful not only for character animation but also for robotics purposes. Additionally, these systems do not require complex data preprocessing such as motion segmentation, alignment and classification, or usage of complex data structures such as KD-trees. Furthermore, given a large amount of motion capture data, a wide range of movements can be learned while maintaining high runtime performance, with some systems being able to produce a wide variation of movements with super-real-time framerates. These advantages make such systems attractive especially for applications in computer games and virtual reality systems.

However, one of the main obstacles in the adoption and use of the systems is the requirement to collect a large amount of motion capture with extensive coverage of the desired motions to be produced. This process is both expensive and time consuming as a large motion capture database must be designed, collected and processed for each new character, new type of motion or new style of motion we wish to create.

In this paper we consider learning new styles of motion from limited data as a few-shot learning task and propose a new approach to reduce the required quantity of data by leveraging already trained models and transferring the learned information to new styles of motion for real-time character controllers. Our strategy is to first train a Phase-Functioned Neural Network (PFNN) using a small number of styles for which we have a large amount of motion capture data. From these few styles we aim to learn a set of style agnostic parameters that capture general locomotion features and a set of style specific parameters that can stylise the output motion. For this purpose, inspired by Rebuffi et al. [RBV17], we introduce an architecture that can learn different styles of character locomotion utilising residual adapters, whose complexity can be adjusted according to the amount of data that is available for a new style. By performing a tensor decomposition to learn phase specific and phase independent parameters we are able to keep the number of style specific parameters small making it easier to learn new styles for which only limited data is available. We additionally use the style agnostic parameters to generalise this limited data to new movement speeds and turning directions. We focus primarily on homogeneous style modelling, that is given a very short clip of stylised walking to generalise this walking animation to new speeds and directions. We discuss some of the challenges in heterogeneous

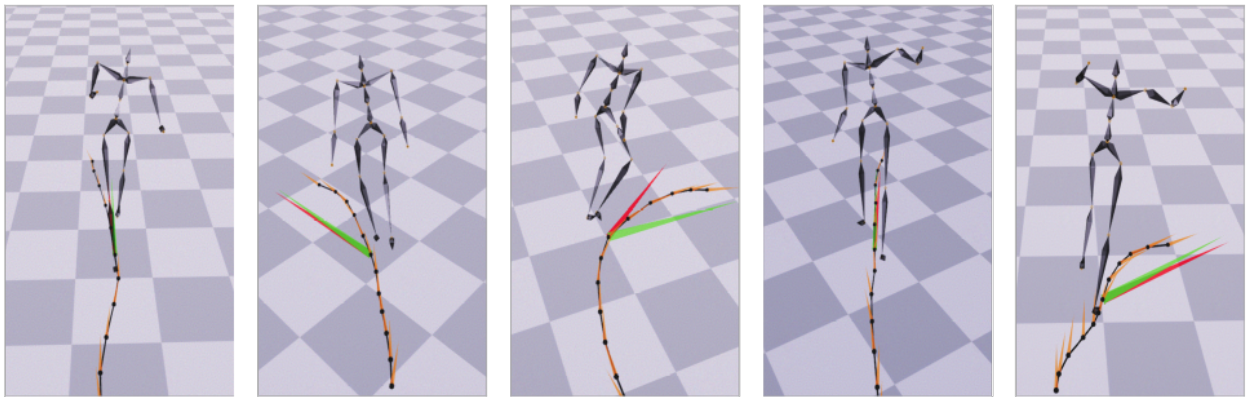

Figure 1: Modelling multiple styles from limited data. From left to right the styles shown are: *joy, on toes crouched, roadrunner, wild arms & zombie*.

modelling, that is transferring the style of the limited data to unseen locomotion types, such as learning to run from a given walking clip.

We show that our system can learn to adapt the neural controller to a wide range of styles from the CMU motion capture database [GS01] using only very short motion clips as training data. We examine different architectures and show that our approach is better than classic transfer learning methods such as fine tuning. We evaluate the system against another neural network based motion stylization approach and discuss the merits and shortcomings of our method. The contributions of this paper can be summarized as follows:

- Provide a method for transfer learning of human locomotion that, after initial training, significantly reduces the quantity of data needed to train generative models in this domain for similar types of locomotion.
- The method is lifelong, meaning new styles can be learned quickly and easily, with only small additional memory constraints due to a tensor decomposition specific to the PFNN.
- A factorisation scheme where the number of parameters for stylisation are adjustable according to the amount of data available.
- Demonstrate the effectiveness of residual adapter domain adaptation in a new domain.

2. Related work

2.1. Modelling human locomotion

Generating human motion is a well studied problem in graphics and vision and many different techniques have been applied. Whilst older methods made use of hand crafted blending of radial basis functions [RCB98] or interpolation using Gaussian Processes [MK05], more recently success has been found using modern data driven methods. One of the first methods to show success was the encoder-recurrent-decoder method of Fragkiadaki et al. [FLFM15] who made use of LSTM cells to successfully predict future motions from given input frames. More recently Li et al. [LZX*18] showed an ability to generate much longer clips of motion by adding in instances of their network's predicted output as well as the ground truth output during training. Bütepage et al. [BBKK17] compare several architectures for human motion prediction, whilst Holden et al. [HSK16] use a convolutional autoencoder to learn a valid manifold of human motion. There have also been successes in using reinforcement learning in physics based systems, [PBYVDP17], [MTS*17].

Most relevant to this work is the Phase-Functioned Neural Network of Holden et al. [HKS17], which reduces blurriness caused by standard recurrent architectures by aligning the phase of a character's locomotion to generate realistic animations in real time from user input.

2.2. Few-shot learning

Few-shot learning is a common problem in machine learning; most modern systems require very large amounts of training data to work as intended, however, in many real world applications such data is not available. Few-shot and one-shot learning tackle the problem of how these systems can be made to learn effectively with very limited data. Much of the work in this area has focussed on images [VBL*16], [BHV*16], [Koc15], but the key idea in most few-shot learning is to learn underlying relevant features for the task at hand through some form of pretraining on a large dataset and to construct different ways of leveraging these features to improve performance when trained on a small amount of data without overfitting. These methods are often very similar to transfer learning methods [Car95] which also rely on learning features from one task and transferring the relevant shared information to improve performance on a similar task, although there is no restriction on available data in this case. There has also been interesting work done on some of the fundamental assumptions used for transfer learning in vision [KSL18], [YCBL14].

2.3. Style transfer and domain adaptation

Style transfer is the task of transferring the style of one piece of data onto the content of another, domain adaptation is concerned with generalising models to work in new domains. In practice however, domain adaptation has often been demonstrated using domains that

consist of items in different styles, e.g. paintings to photos. As the difference between new domains and new styles is often very subtle or non-existent these two areas overlap and are often closely linked.

Early work for style transfer in human motion relied on correspondences between motion clips [HPP05] or made use of physics based optimisations [LHP05]. More recently machine learning has been a popular way to tackle this problem, one of the first works for modelling human motion styles in this area is [TH09] where Taylor et al. used factored conditional restricted Boltzmann machines, conditioned on a style label, to model motion styles. This method however, is a largely theoretical work not designed for graphics applications and lacks the high quality renderings of later works. On the other hand, much of the recent work in style transfer for animation has focussed on transferring the style of one animation clip (e.g. old, angry) onto the content of another (e.g. running, punching, specific directions of locomotion). Xia et al. [XWCH15] use a KNN search of a database of motion to construct a mixture of regression models for transferring style between motion clips and Yumer et al. [YM16] show transfer between heterogeneous actions by optimising in spectral space to match style and content parts of Fourier transforms of reference motions. Holden et al. [HSK16] create stylised motions by optimising the hidden units of a content motion in latent space to match the Gram matrix of a given stylised motion. Later Holden et al. [HHKK17] then increased the speed of this process by learning a feed-forward network for the same task; this method is based on image style transfer from Gatys et al. [GEB15] and Johnson et al. [JAFF16].

For character controllers, such methods are not particularly well suited as they require the entire content of a motion to be defined before being stylised and, additionally, require a database of reference stylised motion clips at run time. Instead, in this work, we view the problem of motion stylisation as a form of domain adaptation where each style is viewed as a separate domain and we attempt to learn how to transfer knowledge about one domain to another. This is much better suited to character controllers as there is no need to define the content ahead of time but instead, when given the same input, we learn to adapt it to different domains (styles) in real time. Our aim is subtly different from the style transfer task in that we do not try to explicitly map a certain style onto certain content, but instead aim to learn parameters that allow us to model style whilst simultaneously generating the animation.

Whilst ideas such as whitening and colouring [LZX*18], and transferring feature summary statistics [HB17] have been applied to style transfer for images, the most popular methods learn mappings between different image domains by making use of some form of cycle consistency loss [ZPIE17] where an image from one domain is translated to another and then back to its original domain so that the final outputted image is close to, or from the same distribution as, the original input. This type of loss has also been adapted by Huang et al. [HLBK18] and Zhu et al. [ZZP*17] to translate one input image to many possible output images in the target domain and Choi et al. [CCK*18] investigate transferring between multiple domains with a single model. Whilst cycle consistency has additionally been used for motion retargeting by Villegas et al. [VYCL18], generally these methods do not easily adapt to time series data. Since the output of a time series model is one or more time steps further on than the input, by the time an input has been mapped into another domain and back to its original domain it is much harder to compare it with the original input. Furthermore using GANs for time series data is still very much an open problem.

Mor et al. [MWPT18] present an autoregressive model for transfer between multiple styles of music. They create an autoencoder model with a shared encoder and separate decoders for each of the styles they work with. Our work is somewhat similar to this but as we are interested in few-shot learning and adding a whole decoder or layer for each style adds a large number of parameters, we do not use an autoencoder model and attempt to model each style with a significantly reduced number of parameters.

In image classification, by adding residual adapters (Section 3.2) to convolutional image classifiers Rebuffi et al. [RBV17] demonstrated domain transfer capabilities between 10 different classification tasks. Rebuffi et al. [RBV17] also showed how, by adding only a small number of new parameters, this method could be used for lifelong learning, that is learning new domains sequentially without decreasing performance on older domains. This method of domain adaptation forms the core of our method for adapting to new motion styles.

3. Background techniques

In this section we give an overview of two key components of our system, first the Phase-Functioned Neural Network (PFNN) from Holden et al. [HKS17] and then residual adapters from Rebuffi et al. [RBV17], [RBV18].

3.1. Phase-Functioned Neural Network

The PFNN [HKS17] is a state of the art architecture for real-time autoregressive modelling of human locomotion animations. It is a simple 3 layer feed forward neural network, where the weights (W_0, W_1, W_2) of the layers are a function of the phase, p (defined in Section 5), of the motion.

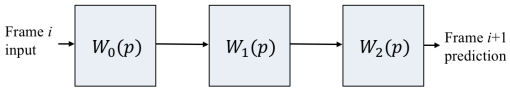

Figure 2: The Phase-Functioned Neural Network, a feedforward network whose weights are a function of the phase of locomotion.

The function of the phase that determines the weights could in theory be any function, but for this work we keep the original function used by Holden et al. [HKS17], that is, a cubic Catmull-Rom Spline. This is defined according to four different settings of the neural network weights, termed control points $\{\underline{\alpha}_1, \underline{\alpha}_2, \underline{\alpha}_3, \underline{\alpha}_4\} = \underline{\beta}$, which are then smoothly blended together dependent on the phase:

$$\begin{aligned}\underline{W}(p;\underline{\beta}) = \underline{\alpha}_{k_1} &+ d(\frac{1}{2}\underline{\alpha}_{k_2} - \frac{1}{2}\underline{\alpha}_{k_0}) \\ &+ d^2(\underline{\alpha}_{k_0} - \frac{5}{2}\underline{\alpha}_{k_1} + 2\underline{\alpha}_{k_2} - \frac{1}{2}\underline{\alpha}_{k_3}) \\ &+ d^3(\frac{3}{2}\underline{\alpha}_{k_1} - \frac{3}{2}\underline{\alpha}_{k_2} + \frac{1}{2}\underline{\alpha}_{k_3} - \frac{1}{2}\underline{\alpha}_{k_0})\end{aligned} \tag{1a}$$

$$d = \frac{4p}{2\pi} \quad (mod\ 1) \tag{1b}$$

$$k_n = \left\lfloor \frac{4p}{2\pi} \right\rfloor + n - 1 \quad (mod\ 4), \tag{1c}$$

where $\underline{W}(p) = \{W_0(p), W_1(p), W_2(p)\}$ is the weights of the neural network for phase p.

3.2. Residual adapters

Originally presented for multi-domain image classification and diagrammed in Fig. 3, a parallel residual adapter [RBV17], [RBV18], which is essentially a skip connection with a weight matrix, is defined as

$$y = f * x + h_{1\times1} * x, \tag{2}$$

where x is input to a convolutional neural network layer, f is the main (arbitrarily sized) filters in this layer, $*$ is the convolution operator and $h_{1\times1}$ is a set of 1×1 filters. Rebuffi et al. [RBV18] learn f to be a set of domain agnostic parameters and $h_{1\times1}$ to be domain specific parameters.

The usage of residual adapters has several beneficial properties: by using a 1×1 filter bank the additional number of parameters required for each new domain is kept small; as f is shared across all domains, new domains can easily be sequentially added to the model by training new residual adapters without decreasing performance on old domains and, the effect of the residual adapter can be easily controlled with well known regularisation techniques as heavy regularisation reduces the adaptation strength.

In this work we adapt this method to the feed forward PFNN architecture and make some necessary changes to improve performance for human locomotion. For example, Rebuffi et al. [RBV18] make use of batch normalisation [IS15] to ensure that the output of the residual adapters are compatible with the original ResNet layers [HZRS16]. However, as the PFNN does not utilise batch normalisation and is relatively shallow, we do not use batch normalisation in our system.

4. System overview

The architecture of our system is shown in Fig. 4. The system is composed of a main Phase-Functioned Neural Network that models the style-independent components of the motions, and a set of residual adapters that model the style-dependent components. The weights of the residual adapters are decomposed into three tensors by a canonical polyadic (CP) decomposition, which reduces the number of parameters to learn for each new style, making few-shot learning easier.

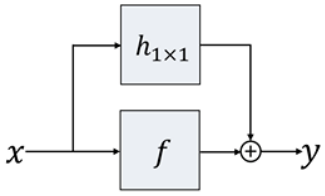

Figure 3: A basic parallel residual adapter. In the convolutional architecture of Rebuffi et al. [RBV18] the output of this module is $y = f * x + h_{1\times1} * x$

For training, we initially train the PFNN and the residual adapters using a rich set of motion capture data containing eight representative styles (Section 5.1). We use short motion clips from the CMU motion capture database for the few-shot scenario (Section 5.2), where we only train the residual adapter weights.

5. Data capture and preprocessing

Here we describe the capture and preprocessing steps for our data. We first describe a larger dataset used for training the main PFNN and associated residual adapters. We then discuss the processing of short motion clips used for few-shot learning of new styles.

5.1. Capturing a large style locomotion dataset

To learn the style-independent parameters of our system, we need to train a PFNN, which requires the capturing of a large amount of motion capture data. To let the network learn common factors of various styles, we capture eight representative styles of locomotion, the same as those used by Xia et al. [XWCH15]: *angry, childlike, depressed, neutral, old, proud, sexy and strutting*. This locomotion is captured by a single actor at 120fps using the Vicon Nexus optical motion capture system. We capture only planar walking and running motions as such locomotions are the most common movements for interactive applications. The data is augmented by mirroring all the captured motions which both doubles the amount of training data and ensures a balance of turning directions. We end up with a large dataset (around 1 hour) of unaligned stylised motions, as there is no mapping between individual clips of different styles the differences between styles must be learned from this unstructured data.

Once captured, the data is preprocessed and converted into the same format as the training data for the original PFNN [HKS17]. First, the data is mapped to the same skeletal structure used in the CMU dataset and by Holden et al. [HKS17]. The joint positions, velocities and rotations are calculated for every frame of the motions, in a local co-ordinate system relative to the root (hips) at the origin. Every frame is then given a phase label which is a number between 0 and 2π which is defined to be 0 when the right foot contacts the ground, π when the left foot contacts the ground, and all other frames found by interpolation. To acquire the phase label, a semi-automatic process is used which first extracts approximate foot contacts based on foot velocities which are then manually cleaned. We refer to one full cycle of the phase label from 0 to 2π

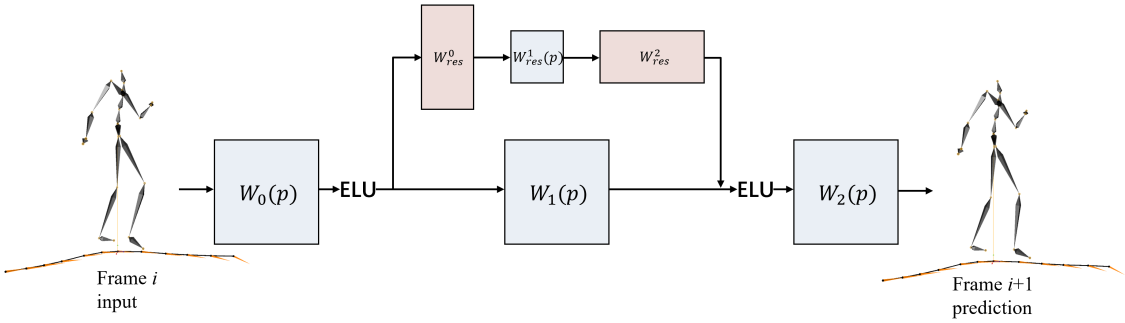

Figure 4: An overview of our proposed system, we add a residual adapter to the PFNN, applying a CP decomposition to the residual adapter where the central matrix is a function of the phase, but the outer matrices are not. The weights that depend on the phase of motion are shown in blue, and those that do not in red. The main weight matrices, $W_0(p), W_1(p), W_2(p)$ are shared between all styles, with separate residual adapter weights learned for each new style. Throughout we use the exponential linear unit for non-linearities [CUH15].

(that is, one right foot contact until the next right contact) as a locomotion cycle. We additionally include the same information for user control as Holden et al. [HKS17]; this takes the form of a trajectory, found by projecting the root onto the ground, centered at the current frame and sampling every ten frames, for 60 frames in the past and 50 frames in the future giving a total of 12 points. At each of these points the position and facing direction of the trajectory is calculated. Finally each locomotion cycle is given a label to represent its style, this is not used as input to the neural network but simply so that we know the style of each input. To ensure a balanced dataset during training, we keep the same number of frames (23104) for each style.

Once preprocessed, the data takes the following form: the input $X_i \in \mathbb{R}^{234}$ consists of: 2×12, x and z (planar) trajectory positions; 2×12, x and z trajectory directions; 3×31, x, y and z joint positions and 3×31, x, y and z joint velocities, for the current frame. The output, $Y_i \in \mathbb{R}^{400}$ consists of: 2×6, x and z current and future trajectory positions, and similarly 2×6, x and z directions; 3×31, x, y and z joint positions; 3×31, x, y and z joint velocities; 3×31, x, y and z joint rotation forward components; 3×31, x, y and z joint rotation upward components, for the next frame; 1 predicted change in phase and 1×3, x and z planar root translations with y root rotation around the vertical axis. At runtime by blending the predicted trajectory with a user inputted trajectory, we are able to control a character using the keyboard.

Before training the neural network the data must be normalised. Importantly we calculate one value for the mean and standard deviation over all the styles, this is done because the mean and standard deviation themselves capture a reasonable degree of information about a style's uniqueness, for example a faster motion will have higher mean joint velocities. As we wish to model motion style with the network's parameters rather than the normalisation values and normalising each style individually would remove this information from the data, we normalise all styles together.

5.2. Processing the style data for few-shot learning

To experiment with few-shot learning we extract 50 styles of motion from the open source CMU motion capture database [GS01]. This database has many styles of locomotion but often very limited data for any one style, for some styles only one locomotion cycle can be extracted from which we attempt to generalise to new turning directions and speeds of locomotion. A full list of styles and the frames available for each is at the end of the paper, Table 2.

This data is processed in the same manner as described above, however, as some styles are asymmetrical, e.g. *left hop*, it does not make sense to mirror these styles because the network would not learn that only the left leg should hop. Furthermore, as we again aim to model style with the parameters of the residual adapters, we normalise the inputs using the mean and standard deviation of the original large mocap dataset. If normalised separately, the summary statistics are specific to the limited speeds and turning directions in the data, meaning generalisation at test time is harder.

6. Motion stylisation network

In this section we describe the architecture we use for modelling motion styles. Namely, we explain how we use the constituent parts described in Section 3 to model different styles and how, in the process, we train this network to learn a set of style agnostic parameters.

First, we aim to create a style independent module to capture common features for synthesising various motion styles. From the large mocap dataset described in Section 5 we learn one set of style agnostic parameters, $\underline{\beta}_{ag}$, and eight sets of style specific parameters, $\{\underline{\beta}^{(s)} | s \in \{1, \ldots 8\}\}$, one for each style in the dataset.

To do this we add a separate residual adapter for each style, s, whose weights are the function of the phase given in Eq. (1a), with control points $\underline{\beta}^{(s)} = \{\underline{\alpha}^{(s)}_1, \underline{\alpha}^{(s)}_2, \underline{\alpha}^{(s)}_3, \underline{\alpha}^{(s)}_4\}$. However, apart from these weights, all the other weights in the neural network are shared

between styles, with control points $\underline{\beta}_{ag} = \{\underline{\alpha}_1, \underline{\alpha}_2, \underline{\alpha}_3, \underline{\alpha}_4\}$. This can be seen in Fig. 5 where the shared, style agnostic, parameters are shown in grey and the style specific parameters in other colours. This process can be seen as analogous to the training of shared encoders with domain specific decoders from Mor et al. [MWPT18].

In order to encourage $\underline{\beta}_{ag}$ to be style invariant and not to tend towards one style over another we balance our large dataset so that it contains exactly the same number of frames for each style of locomotion. We similarly ensure an equal amount of mirrored and non-mirrored motion so that the generated motions do not prefer one turning direction over another.

For one input, $X_i^{(s)}$, with corresponding phase values, $P_i^{(s)}$, for known style, s, we predict the corresponding output, $Y_i^{(s)}$, by minimising:

$$\mathcal{L}(X_i^{(s)}, Y_i^{(s)}, P_i^{(s)}; s) = ||Y_i^{(s)} - \Phi(X_i^{(s)}, P_i^{(s)}; \underline{\beta}^{(s)}, \underline{\beta}_{ag})||_2 + \lambda_{ag}||\underline{\beta}_{ag}||_1 + \lambda_s||\underline{\beta}^{(s)}||_1, \quad (3)$$

where $\underline{\beta}^{(s)}$ is the style specific parameters for style s, learned in the residual adapter, and $\underline{\beta}_{ag}$ is all the remaining network parameters. Φ is the neural network:

$$\Phi(X_i^{(s)}, P_i^{(s)}; \underline{\beta}^{(s)}, \underline{\beta}_{ag}) = W_2 ELU(W_1 ELU(W_0 X_i^{(s)} + b_0) + b_1 + W_{res}^{(s)} ELU(W_0 X_i^{(s)} + b_0) + b_{res}^{(s)}) + b_2 \quad (4)$$

With each of the weight matrices and biases depending on the phase, $P_i^{(s)}$, according to Eq. (1a). For W_0, W_1, W_2 the control points are $\underline{\beta}_{ag} = \{\underline{\alpha}_1, \underline{\alpha}_2, \underline{\alpha}_3, \underline{\alpha}_4\}$ and for W_{res}^s we use $\underline{\beta}^{(s)} = \{\underline{\alpha}_1^{(s)}, \underline{\alpha}_2^{(s)}, \underline{\alpha}_3^{(s)}, \underline{\alpha}_4^{(s)}\}$. We place a small L1 loss on the weights which encourages sparsity and acts as a form of regularization, we set λ_{ag} and all λ_s to 0.01 in Eq. (3). Experiments on varying the regularisation parameters produced minimal qualitative differences.

The data is split into minibatches with each minibatch containing only one style, the minibatches are shuffled to ensure that we cycle through the styles; as we have 8 styles this means every 8^{th} minibatch contains motion data from the same style.

As the dataset is balanced we have the same number of input datapoints for each style, so the overall objective is:

$$L(\mathbf{X}, \mathbf{Y}, \mathbf{P}) = \sum_{i=1}^{N} \sum_{s=1}^{S} \mathcal{L}(X_i^{(s)}, Y_i^{(s)}, P_i^{(s)}; s), \quad (5)$$

where S is the number of styles in the large mocap dataset, in our case $S = 8$, N is the number of input datapoints for each style (equal to the number of frames) and $\mathbf{X}, \mathbf{Y}, \mathbf{P}$ are the full datasets of inputs, outputs and phases respectively (so $\mathbf{X} = \bigcup_{i=1}^{N} \bigcup_{s=1}^{S} X_i^{(s)}$ etc.).

The network is trained for 25 epochs using Adam [KB14] which takes approximately 6 hours on a single NVIDIA GTX 1080 Ti GPU. This early stopping acts as additional regularisation on the network as we find training for large numbers of epochs to produce results that overfit for certain styles.

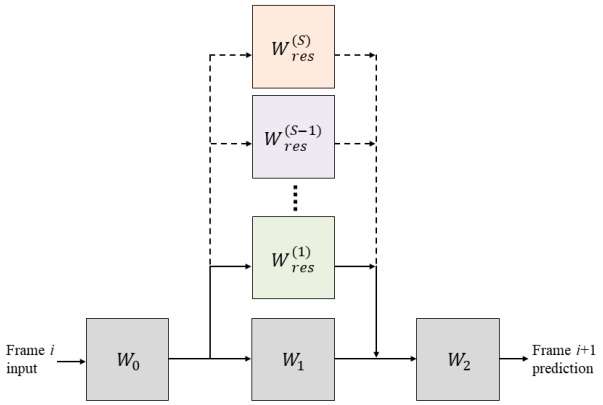

Figure 5: Training the model: shared weights are shown in grey and style specific weights in other colours. We cycle through the S styles evenly so our system does not bias towards any one style.

7. Few-shot learning of new styles

After training with the large database of stylised motions, we attempt to learn new styles with limited data. Here we describe the difficulties with, and our proposed solution for, tackling this.

If the training process successfully separates style agnostic and style specific information, given new data it should be possible to learn new styles by freezing the weights of the main network and learning only a new residual adapter. Given enough data, the style agnostic parameters should learn to model general locomotion features common to all styles, meaning that for a new style we need to only learn how to adapt these features rather than learning the entire network from scratch. This along with the reduced number of tunable parameters means we may be able to learn the new style specific parameters with considerably less data required than if we had also to learn the style agnostic parameters.

The major challenge for such few-shot learning is that the data does not contain an extensive range of motions, frequently a style may only have one very short clip (1-5s) of a walking motion in a straight line, which makes learning to turn and run in a stylised manner difficult. With such a small amount of data it is very easy to overfit and hence only be able to mimic the training data. This is shown in Fig. 6 where the character walks in the *bent forward* style in a straight line but when turning becomes totally unrealistic. For heterogeneous transfer this task becomes even harder as, for example, given an animation of a character walking in a straight line there is no deterministic mapping that tells us how this character should run, and in some cases the style may change drastically.

Another challenge is that adding a new residual adapter, with a full weight matrix for each style, rapidly increases the number of parameters to store. In our case an increase of 44.7% over the original network's parameter count for each new style, see Table 1.

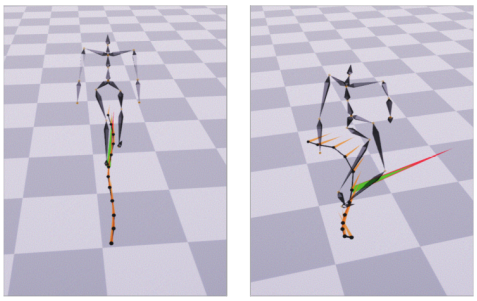

Figure 6: When training with only a small amount of data, the character can match trajectories similar to those seen during in training but cannot generalise to new ones. Here a PFNN is trained on neutral data and then finetuned with the limited bent forwards data.

7.1. CP decomposition

To tackle both of these issues we perform a CP decomposition [KB09] on the residual adapters which reduces the number of parameters being learned for each style, helping prevent overfitting and reducing memory costs. The CP decomposition is a decomposition of a 3D tensor $X \in \mathbb{R}^{I \times J \times K}$ where each matrix slice of the tensor, X_k, can be written as

$$X_k = AD^{(k)}B^T \tag{6}$$

with $A \in \mathbb{R}^{I \times R}$, $B \in \mathbb{R}^{J \times R}$, $D^{(k)} \in \mathbb{R}^{R \times R}$ and $D^{(k)}$ a diagonal matrix for $k = 1, \ldots, K$, some positive integer R. This can be seen as a generalisation of SVD to higher dimensions or as a Tucker decomposition with diagonal core tensor.

In order to apply the CP decomposition, by considering the phase as a third dimension, we can view the weights of a residual adapter as a 3D tensor. In the demo of Holden et al. [HKS17] the phase is discretised at runtime in order to improve the speed of the system. Specifically 50 weight matrices are calculated for phases $[0, 2\pi/50, \ldots, 2\pi)$ and the weight matrices associated with the closest phase value to the current phase of motion are used to predict the next frame. By stacking these matrices we can use the discretised phase as a third dimension; therefore, thinking of the residual adapter as a tensor of size $512 \times 512 \times 50$ we can then decompose each of the 50 slices of this tensor (when sliced over the phase dimension) into 3 matrices of sizes $512 \times n$, $n \times n$ and $n \times 512$ where n is some integer, $n \ll 512$. As in Eq. (6), the rectangular matrices are the same across all slices meaning that the (diagonal) square matrix has weights as a function of the phase but the weights of the two rectangular matrices do not. For the original 8 styles we set $n = 30$ which means adding only 1.5% more parameters per style during training and at run time, when we discretise the phase, storing 50 matrices for each layer, only 0.03%.

The intuition behind performing this decomposition is that a certain amount of style information should be phase independent (such as global changes in speed or posture) whereas other features of a style may be phase dependent. Furthermore, this decomposition method helps solve both of the issues raised in the previous section as it significantly reduces the number of parameters being learned for a new style which helps to prevent overfitting and reduces the required memory. (A cursory experiment applying this decomposition to every layer of the original PFNN on flat ground reduced the parameter size from 136 MB to 1 MB and gave a 3-4X speedup for the forwards pass of the network, albeit with slightly reduced controllability.) As in Zhao et al. [ZHSS17] we perform the decomposition before training, that is we learn the factors A, D and B from Eq. (6) directly during training and then multiply them together. This allows the network to learn the best values for each of the component tensors, reduces the number of parameters needed during training and keeps the process fully differentiable. Finally, note that performing this decomposition changes $\underline{\beta}^{(s)}$ from Section 6, as now only the central matrix depends on control points $\{\underline{\alpha}_1^{(s)}, \underline{\alpha}_2^{(s)}, \underline{\alpha}_3^{(s)}, \underline{\alpha}_4^{(s)}\}$.

7.2. Variable dropout

As discussed in Section 3.2, a major benefit of residual adapters is that we can tune the regularisation to adapt the generalisation strength for each new style, hence we vary the dropout rate used during training the few-shot styles in order to achieve better results. Here the regularisation can be seen as a trade off between the new style specific parameters and the shared style agnostic parameters. The ability of our system to generalise to a new style is determined primarily by two factors, firstly the quantity of data available for the given style and secondly how similar the style is to those in the large dataset which the main network is trained on. Those motions for which we have very little data, or that are very different from motions already seen by the network, require heavier regularisation to avoid artifacts at run time. As similarity between motions is hard to measure quantitavely we run a simple grid search over the dropout rate and select the value which creates the best qualitative results. For any given style it may be possible to improve results with a more extensive search or an intelligent heuristic.

We could also opt for other regularisation methods, such as L1 regularisation or reducing the size of the central tensor in the CP decomposition. We experimented with these forms of regularisation and found them to be roughly equally as effective. We utilise dropout due to its simplicity to understand and implement.

8. Results and comparisons

Here we compare our system to other possible methods of learning to model new styles as well as comparing against the Gram matrix transfer method of Holden et al. [HSK16]. As animation generation does not lend itself to quantitative evaluation, and in general the quantitative evaluation of generated animations or images is still an open problem, the majority of this section is a qualitative discussion of the results best seen in the accompanying video along with some details of training times and memory requirements. For all of our experiments we use the discretised phase method in order to achieve the best possible runtime performance. Some still frames showing the modelling of different styles are shown in Fig. 1.

8.1. Evaluating our model

We first compare our system against other models that we train in order to try and achieve the same goal of learning new styles

Table 1: Table of comparisons showing storage requirements for a new residual adapter, mean training time per style, and the time for a forwards pass for different configurations of the residual adapters. The two values given in the memory requirement column represent only the control points being stored at run time and the phase being discretised for faster performance at run time respectively.

Method	Memory (MB)	Training (s)	Runtime (s)
Full Adapter	4.01 / 50.1	99	0.0013
Diagonal Adapter	0.016 / 0.20	44	0.0010
CP Decomposition	0.126 / 0.131	50	0.0011

from limited data. Trying the most basic methods, namely training the whole PFNN on only the few-shot data or finetuning a PFNN trained on a large amount of neutral motion leads to extremely unrealistic motions for even slight perturbations in the input trajectory due to these methods heavily overfitting - see Fig. 6.

We also experiment with altering the decomposition of the residual adapters, the results of which are shown in Table 1 where we show the memory usage, training time and runtime performance of the different methods. If we use a full matrix of weights for the adapters we can capture style relatively well and, with sufficient regularisation, can often generalise this to unseen directions. However, this method has a tendency to overfit due to the large number of parameters relative to the small amount of data, this can be seen in the accompanying video where the *elated* style is unable to turn left. Furthermore, the storage requirements for using full matrices grows rapidly, albeit linearly, with the number of styles (Table 1, top row). Adapting the work of Rebuffi et al. [RBV17], one way to reduce the number of parameters is to force the residual adapter to be diagonal (Table 1, middle row). However, we found this did not allow enough variance in the model as the system could not learn many styles, instead outputting average motions. Examples of these issues are shown in Fig. 7.

Additionally, we evaluate the run time performance of each of the methods from the previous paragraph, the last column of Table 1 shows the time for a forwards pass of each of the neural networks. As expected the methods with the fewest parameters and hence the fewest number of operations perform fastest. The runtime experiments are run on a single Intel i7-6700HQ CPU and include some overhead from the Unity Editor but do not include the time for rendering the scene.

We note the importance of training on a dataset with multiple locomotion styles, if we add residual adapters to a network trained only on neutral motion, often we cannot reproduce new styles and instead output motions that are quite close to neutral. This is likely because, when only training on neutral data, there is no reason for the first layer of the PFNN to learn to extract good features for style transfer, whereas training on multiple styles encourages this.

To visualise the style agnostic parameters we can generate motion with the weights of the residual adapters set to 0; in some sense this can be seen as the underlying 'average motion'. The motion that is learned is fairly close to a neutral walk and can be seen in the accompanying video. Interestingly when turning quickly this motion has a slight snap in the animation, which is something that we

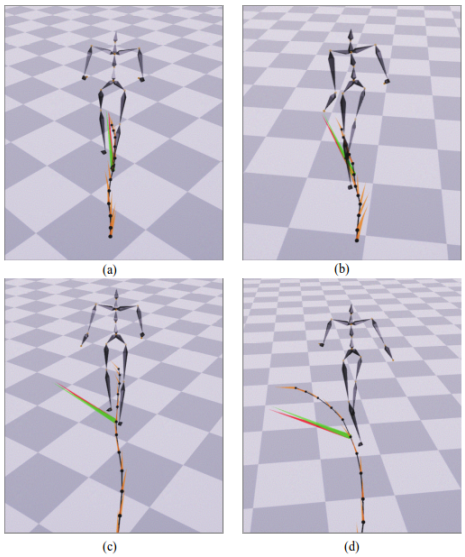

Figure 7: Under and overfitting depending on the number of parameters in the residual adapter. (a) *dinosaur* style, CP decomposed residual adapter, reasonably capturing the style. (b) *dinosaur* style, diagonal residual adapter, the style is captured poorly. (c) *elated* style, full matrix of weights for the residual adapter, the style is captured but the character struggles to turn left. (d) *elated* style, CP decomposed residual adapter, the style is still captured but the character can turn more easily.

then observe with the stylised animations having slightly odd motions for turning quickly. We tried several things to remove this artifact including pretraining on neutral motions and finetuning with the large dataset while regularising the neutral adapter in order to encourage the average motion to remain closer to a neutral motion, but we were unable to remove this oddity. It may be that this is a good underlying motion for our specific style dataset, but it is unlikely to be totally universal, indicating that there may be improvements that can be made by gathering larger quantities of data.

Overall our system is capable of learning to model a new style with only a few seconds of motion capture data and low training time, see Tables 1 and 2. Most of the styles we train on can be generalised to unseen turning motions and small, realistic, changes in walking speeds. However, certain styles cause problems for this architecture. They fall into two main categories: styles which are very different from those in the large training set and styles which are stochastic, aperiodic or have very unique root joint trajectories. The latter of these categories is a limitation of the PFNN, in that it is designed to model regular periodic motion and cannot easily model strange trajectories with lots of stopping and starting like those seen in the martial arts (e.g. *gedanbarai*) styles of the CMU dataset. In the former case we can often learn the styles to some extent but we see the motion is heavily smoothed, for example when trying to learn *left hop*, although we learn something close to a hopping motion the right foot barely lifts off the ground, similarly when learning *march* the knees do not raise as high as they should.

8.2. Comparison with an alternative style transfer method

We now compare our method with the method of Holden et al. [HSK16] which learns a motion manifold using an autoencoder and then optimises in latent space to transfer the style of one clip to the content of another. To compare how well the different methods can model different styles we first select a content clip from our database of neutral motion and stylise it using the method of Holden et al. [HSK16] with one of the clips we use for few-shot learning. We then extract the trajectory of the stylised motion and use this as input to our system at every time step so that both motions follow a similar path.

We show results in the accompanying video, in most cases our method is qualitatively at least as good or better than Holden et al. [HSK16] for creating realistic looking stylisations. However, for certain styles, mostly those in the problem cases mentioned above, our results are not as good. It is worth noting again here that the task we are trying to do is slightly different from the style transfer task; we do not train in order to receive an entire content trajectory at test time but instead to predict the future motion given past inputs and to simultaneously generate and stylise the motion.

9. Limitations

To end, we discuss the current limitations of our system which may provide potential directions for future work. These limitations include the inability to perform heterogeneous transfer, the lack of quantitative evaluation metrics, a loss of animation detail and an inability to handle non-periodic movements.

9.1. Heterogeneous transfer

In this work we have demonstrated homogeneous transfer, that is from a given short walking clip we can learn to generate new walking clips in that style with new trajectories. However, hand crafted solutions such as those of Yumer et al. [YM16] have shown some heterogeneous transfer ability, generating running or punching in a specific style given a clip of walking in that style.

Being able to perform this type of style modelling in systems that can generate animations such as the PFNN is very desirable. We implemented a number of methods to attempt heterogeneous transfer including adding layer normalisation [BKH16] specific to each style, adding walk and run gait labels to the data and attempting to add a simple Fourier based loss function inspired by Yumer et al. [YM16]. However, whilst we were able to learn running for a handful of styles, none of these methods were able to create consistent high quality heterogeneous transfer and, in general, most styles tended to reduce to a neutral motion when running.

9.2. Quantitative evaluation

To the best of our knowledge, no appropriate metric for the quantitative evaluation of generated animations exists since those methods that do exist, such as in Xia et al. [XWCH15], require a ground truth motion, which is unavailable in our case. The lack of a quantitative metric makes taking design choices that subtly alter results very hard, this includes setting regularisation parameters (λ_{ag} and λ_s, Eq. (4)), type of regularisation, or number of training epochs.

The creation of some such metric would therefore be very useful for further evaluation of this work. However, we believe this to be a difficult task partly because, in the absence of ground truth, what makes one animation better than another can often be subjective. In vision, the quantitative evaluation of model generated images is very much an open area of research [Bor18].

9.3. Loss of high frequency details

Many of the animations we generate look realistic and reasonable but when compared with the training data are somewhat smoothed, losing much of the subtlety that is possible with manual animation. From our observations, this is a limitation of many works on neural network generated animation in that high frequency movements are often not perfectly reproduced. In our case this problem is exacerbated by the limited amount of data available; in order to generalise, the model creates a blending between the agnostic motion and the stylised data. Methods such as normalising each style individually allow for more details to be captured but cannot generalise well.

We do however believe that with sufficient resources this system could be engineered further to produce higher quality results. There are many variables that could be altered such as the number of training epochs for each style, the values of the variable dropout rate and alternative ways of normalising the data that may improve the results presented here. We hope this work acts as a proof of concept that can be developed further as required.

9.4. Inability to handle non-periodic motion

Finally, creating systems that can generate non-periodic or stochastic motions from user input is very much an open problem. Motions which have a poorly defined phase or a phase that is very different from standard locomotion, e.g. dancing, or stochastic motions, e.g. random looks behind, are not currently producible with phase based architecture and likely require new ideas and architectures.

10. Conclusion

We have presented a method for learning multiple styles of locomotion, where extensive motion coverage is only required for a small subset of styles from which we can learn to generalise for other styles which may have only limited data available. Additionally, our system requires no database of reference motions at runtime. This method has potential applications in making the PFNN more accessible for creative work where the time, equipment and resources may not be available to capture the large amount of data required to learn the model from scratch and could likely be adapted to model key frame animations.

As we require only small quantities of data, we have evaluated our models using many more styles than most other works in this area. This allows us to perform a thorough examination of our model and pinpoint specific areas for improvement.

Acknowlegements

We thank the NVIDIA Corporation for the generous dontations of the GPUs. This project was partly supported by the EPSRC DTA programme and the Google VR Research Faculty Award.

References

[BBKK17] BÜTEPAGE J., BLACK M., KRAGIC D., KJELLSTRÖM H.: Deep representation learning for human motion prediction and classification. *arXiv preprint arXiv:1702.07486* (2017). 2

[BHV*16] BERTINETTO L., HENRIQUES J. F., VALMADRE J., TORR P., VEDALDI A.: Learning feed-forward one-shot learners. In *Advances in Neural Information Processing Systems* (2016), pp. 523–531. 2

[BKH16] BA J. L., KIROS J. R., HINTON G. E.: Layer normalization. *arXiv preprint arXiv:1607.06450* (2016). 9

[Bor18] BORJI A.: Pros and cons of gan evaluation measures. *arXiv preprint arXiv:1802.03446* (2018). 9

[Car95] CARUANA R.: Learning many related tasks at the same time with backpropagation. In *Advances in neural information processing systems* (1995), pp. 657–664. 2

[CCK*18] CHOI Y., CHOI M., KIM M., HA J.-W., KIM S., CHOO J.: Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2018). 3

[CUH15] CLEVERT D.-A., UNTERTHINER T., HOCHREITER S.: Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289* (2015). 5

[FLFM15] FRAGKIADAKI K., LEVINE S., FELSEN P., MALIK J.: Recurrent network models for human dynamics. In *Proceedings of the IEEE International Conference on Computer Vision* (2015), pp. 4346–4354. 2

[GEB15] GATYS L. A., ECKER A. S., BETHGE M.: A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576* (2015). 3

[GS01] GROSS R., SHI J.: *The CMU Motion of Body (MoBo) Database*. Tech. rep., 2001. 2, 5

[HB17] HUANG X., BELONGIE S.: Arbitrary style transfer in real-time with adaptive instance normalization. In *Computer Vision (ICCV), 2017 IEEE International Conference on* (2017). 3

[HHKK17] HOLDEN D., HABIBIE I., KUSAJIMA I., KOMURA T.: Fast neural style transfer for motion data. *IEEE computer graphics and applications 37*, 4 (2017), 42–49. 3

[HKS17] HOLDEN D., KOMURA T., SAITO J.: Phase-functioned neural networks for character control. *ACM Transactions on Graphics (TOG) 36*, 4 (2017), 42. 2, 3, 4, 5, 7

[HLBK18] HUANG X., LIU M.-Y., BELONGIE S., KAUTZ J.: Multimodal unsupervised image-to-image translation. *arXiv preprint arXiv:1804.04732* (2018). 3

[HPP05] HSU E., PULLI K., POPOVIĆ J.: Style translation for human motion. In *ACM Transactions on Graphics (TOG)* (2005), vol. 24, ACM, pp. 1082–1089. 3

[HSK16] HOLDEN D., SAITO J., KOMURA T.: A deep learning framework for character motion synthesis and editing. *ACM Trans. Graph. 35*, 4 (2016), 138:1–138:11. 2, 3, 7, 9

[HZRS16] HE K., ZHANG X., REN S., SUN J.: Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778. 4

[IS15] IOFFE S., SZEGEDY C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML* (2015). 4

[JAFF16] JOHNSON J., ALAHI A., FEI-FEI L.: Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision* (2016), Springer, pp. 694–711. 3

[KB09] KOLDA T. G., BADER B. W.: Tensor decompositions and applications. *SIAM review 51*, 3 (2009), 455–500. 7

[KB14] KINGMA D. P., BA J.: Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014). 6

[Koc15] KOCH G.: Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop* (2015). 2

[KSL18] KORNBLITH S., SHLENS J., LE Q. V.: Do better imagenet models transfer better? *arXiv preprint arXiv:1805.08974* (2018). 2

[LHP05] LIU C. K., HERTZMANN A., POPOVIĆ Z.: Learning physics-based motion style with nonlinear inverse optimization. In *ACM Transactions on Graphics (TOG)* (2005), vol. 24, ACM, pp. 1071–1081. 3

[LZX*18] LI Z., ZHOU Y., XIAO S., HE C., LI H.: Auto-conditioned lstm network for extended complex human motion synthesis. In *ICLR* (2018). 2, 3

[MK05] MUKAI T., KURIYAMA S.: Geostatistical motion interpolation. In *ACM Transactions on Graphics (TOG)* (2005), vol. 24, ACM, pp. 1062–1070. 2

[MTS*17] MEREL J., TASSA Y., SRINIVASAN S., LEMMON J., WANG Z., WAYNE G., HEESS N.: Learning human behaviors from motion capture by adversarial imitation. *arXiv preprint arXiv:1707.02201* (2017). 2

[MWPT18] MOR N., WOLF L., POLYAK A., TAIGMAN Y.: A universal music translation network. *arXiv preprint arXiv:1805.07848* (2018). 3, 6

[PBYVDP17] PENG X. B., BERSETH G., YIN K., VAN DE PANNE M.: Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG) 36*, 4 (2017), 41. 2

[RBV17] REBUFFI S.-A., BILEN H., VEDALDI A.: Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems* (2017), pp. 506–516. 1, 3, 4, 8

[RBV18] REBUFFI S.-A., BILEN H., VEDALDI A.: Efficient parametrization of multi-domain deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2018). 3, 4

[RCB98] ROSE C., COHEN M. F., BODENHEIMER B.: Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications 18*, 5 (1998), 32–40. 2

[TH09] TAYLOR G. W., HINTON G. E.: Factored conditional restricted boltzmann machines for modeling motion style. In *Proceedings of the 26th annual international conference on machine learning* (2009), ACM, pp. 1025–1032. 3

[VBL*16] VINYALS O., BLUNDELL C., LILLICRAP T., WIERSTRA D., ET AL.: Matching networks for one shot learning. In *Advances in Neural Information Processing Systems* (2016), pp. 3630–3638. 2

[VYCL18] VILLEGAS R., YANG J., CEYLAN D., LEE H.: Neural kinematic networks for unsupervised motion retargetting. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2018). 3

[XWCH15] XIA S., WANG C., CHAI J., HODGINS J.: Realtime style transfer for unlabeled heterogeneous human motion. *ACM Trans. Graph. 34*, 4 (2015), 119:1–119:10. 3, 4, 9

[YCBL14] YOSINSKI J., CLUNE J., BENGIO Y., LIPSON H.: How transferable are features in deep neural networks? In *Advances in neural information processing systems* (2014), pp. 3320–3328. 2

[YM16] YUMER M. E., MITRA N. J.: Spectral style transfer for human motion between independent actions. *ACM Transactions on Graphics (TOG) 35*, 4 (2016), 137. 3, 9

[ZHSS17] ZHAO C., HOSPEDALES T. M., STULP F., SIGAUD O.: Tensor based knowledge transfer across skill categories for robot control. In *International Joint Conference in Artificial Intelligence (IJCAI)* (2017), vol. 10, pp. 1–4. 7

[ZPIE17] ZHU J.-Y., PARK T., ISOLA P., EFROS A. A.: Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on* (2017). 3

[ZZP*17] ZHU J.-Y., ZHANG R., PATHAK D., DARRELL T., EFROS A. A., WANG O., SHECHTMAN E.: Toward multimodal image-to-image translation. In *Advances in Neural Information Processing Systems* (2017), pp. 465–476. 3

Table 2: The number of frames and locomotion cycles for unmirrored data of the styles used for few-shot learning.

Style	Number of Frames	Number of Locomotion Cycles	Symmetric
Balance	1491	12	Yes
Bent Forward	464	6	Yes
Bent Knees	182	2	Yes
Bouncy	304	4	No
Cat	197	2	Yes
Chicken	56	1	Yes
Cool	244	3	Yes
Crossover	230	2	Yes
Crouched	310	4	Yes
Dance	680	9	No
Dinosaur	527	5	Yes
Drag Leg	412	5	No
Dragon	125	1	Yes
Drunk	431	7	Yes
Duck Foot	258	3	Yes
Elated	67	1	Yes
Empi	234	1	Yes
Frankenstein	293	2	Yes
Gangly	493	7	Yes
Gedanbarai	294	2	Yes
Ghost	124	2	Yes
Graceful	323	6	Yes
Heavyset	417	4	Yes
Heiansyodan	95	1	Yes
Hobble	143	2	No
Hurt Leg	455	4	No
Jaunty	78	1	Yes
Joy	138	2	Yes
Lean Right	302	4	No
Left Hop	455	8	No
Legs Apart	186	2	Yes
Mantis	212	5	Yes
March	213	2	Yes
Mawashigeri	342	1	Yes
Monkey	103	2	Yes
Oiduki	228	1	Yes
On Toes Bent Forward	390	5	Yes
On Toes Crouched	591	8	Yes
Painful Left Knee	411	5	No
Penguin	183	6	Yes
Pigeon Toed	249	3	Yes
Prarie Dog	123	3	Yes
Quail	55	2	Yes
Roadrunner	665	17	Yes
Rushed	78	2	Yes
Sneaky	314	2	Yes
Squirrel	129	9	Yes
Stern	958	19	Yes
Stuff	97	2	Yes
Stumble	66	1	Yes
Swing Shoulders	259	3	Yes
Syutouuke	182	1	Yes
Wild Arms	161	2	Yes
Wild Legs	211	2	Yes
Wounded Leg	986	25	No
Yokogeri	135	1	No
Zombie	2241	25	Yes