Fast Neural Style Transfer for Motion Data

Daniel Holden and Ikhsanul Habibie ■ *University of Edinburgh*

Ikuo Kusajima ■ *University of Tokyo*

Taku Komura ■ *University of Edinburgh*

Automating motion style transfer can help save animators time by allowing them to produce a single set of motions and reuse them for different characters. The proposed fast, efficient technique for performing neural style transfer of human motion data uses a feed-forward neural network trained on a large motion database.

Motion style transfer is a technique for converting an actor's motion into that of a different character, such as one that is old, depressed, happy, or hurt. Automating this process can save animators time because it means they do not need to create different variations of motions for each character. Instead, they can produce a single set of motions, which are then automatically adapted for use with different characters.

Many previous techniques for style transfer have been developed. Most of these methods are data-driven and require a set of corresponding motions both in the neutral and characterized styles. These motions must be temporally aligned so that associated poses can be computed and a regression of some form learned. Constructing such a dataset can be tedious for artists, and even with automatic methods, obtaining a good temporal alignment between motions can be difficult or require significant manual intervention from a technical developer.

Neural style transfer, first introduced by Leon Gatys and his colleagues[1] and later adapted for motion data by Daniel Holden and his colleagues,[2] uses a deep neural network to perform the style transfer task, solving an optimization problem over the neural network hidden units to produce motion in the style of one clip, but with the content of another. This method overcomes a number of the issues with conventional style transfer methods. First, it only requires a single exemplar motion to represent the style rather than a database of corresponding clips. Second, it does not require any kind of alignment between content and style clips, instead calculating the style implicitly by taking an average over all the frames of the motion's Gram matrix. Both of these things are appealing to developers because they reduce the required amount of data preparation.

One of the disadvantages of neural style transfer has conventionally been the speed. Instead of performing a regression, neural style transfer requires solving an optimization problem, which is computationally expensive and may take a long time.

In this article, we present a technique inspired by the work of Justin Johnson and his colleagues[3] that removes this speed limitation while retaining the other advantages of the original style transfer scheme.[1] To do this, we train a fast feed-forward neural network to perform a regression task satisfying the constraints of the original optimization problem as well as new motion-specific constraints. We also replace the pretrained classification network with an autoencoding convolutional network like the one used in the Holden method.[2] This acts as a loss network that has the advantage of being trained unsupervised, and it is additionally used to fix any artifacts or noise that may be present in the produced motion.

 Published by the IEEE Computer Society

State of the Art in Motion Style Transfer

We begin our discussion by first reviewing motion style transfer methods in computer graphics. Then we explore the use of deep-learning methods in data transformation with an emphasis on style transfer. Finally, we discuss previous motion synthesis methods based on deep-learning techniques.

Motion Style Transfer

Motion style transfer is an old problem in computer animation, with the idea being to import a style from one motion clip and apply it to another motion. Motion style transfer is especially useful for applications such as computer games, where the goal is to minimize the amount of motion data in the package.

One approach is to handle the motion in the frequency domain.[4,5] Neutral motions are converted into motions with different emotions by transferring the difference of the Fourier coefficients. Katherine Pullen and Christoph Bregler defined the style of the motion by the motion data's high frequency elements and added them to a novel motion to transfer the style.[6] M. Ersin Yumer and Niloy Mitra slid a window along the motion and applied a fast Fourier transform (FFT) to extract the high frequency data that represents the style.[7]

Another approach is to use dynamic models. Eugene Hsu and his colleagues used a linear-time-invariant model to produce a time-series model where the style is embedded.[8] Jianyuan Min and his colleagues proposed a multilinear analysis approach to synthesize and transfer motion styles between actors.[9] This method can be used to reduce ambiguity because the models can represent the motion data in a low-dimensional space. Another advantage of this method is the omission of the foot-contact definition to prevent the foot-sliding problem as the model is constructed from motion registration. Shihong Xia and his colleagues demonstrated that human motion style transfer can be performed in real time by constructing a local mixture of autoregressive models of styles and content that can be used to stylize a sequence of different motion contents, such as a walking motion immediately followed by jumping.[10] The idea is to calculate the closest possible style from a given current motion using the trained model, which is then used to predict the next frame's motion.

Feed-Forward Transformation and Style Transfer

Researchers have recently applied feed-forward neural networks for data transformation to various image processing tasks. Related works include the use of convolutional neural networks to transform low-resolution images to produce high-resolution images,[11] coloring of grayscale images,[12] and segmentation or semantic scene understanding of an image.[13] Like other neural-network approaches, once the models are trained, they can easily and quickly be used to perform similar transformations on new input data.

Gatys and his colleagues showed that the concept of image transformation using deep neural networks can be used to combine distinct elements of an image by using a method known as style transfer.[1] Their work showed that convolutional neural networks can be used to combine two images by transferring the style of one image to the content of another. The combined image successfully captures the visual style of one image while preserving the content of the other. As Matthew Zeiler and Rob Fergus showed, each convolutional layer in the network captures the shape representation of the objects in the image with increasing levels of detail.[14] This feature of the network can then be used to capture the data's content representation. On the other hand, the style in the image data is represented as the texture characteristics, such as color and local features, generally encoded by the Gram matrix.

Recently, Johnson and his colleagues proposed a deep-neural-network structure with a perceptual loss function that can be used to carry out image transformation tasks such as style transfer and the production of high-quality super-resolution images.[3] They suggested that a loss function based on perceptual difference can be used to improve the performance compared with the common per-pixel mean-squared-error comparison. Their method produces style-transferred images comparable to the original style transfer scheme,[1] but it is faster by three orders of magnitude. We adopt this approach, but use motion-specific constraints and an additional manifold projection step to make it suitable for use with motion data.

Deep Neural Network for Human Motion Analysis

Before their widespread success in various research areas, such as image transformation, the popularity of deep neural networks was based on their ability to achieve state-of-the-art performances in recognition problems, especially for image and speech recognition. These successes can be generalized and used for other problems, including the classification and recognition of human motion data.[15]

The earliest human-motion-synthesis techniques using neural networks included the conditional restricted Boltzmann machine (cRBM)[16]

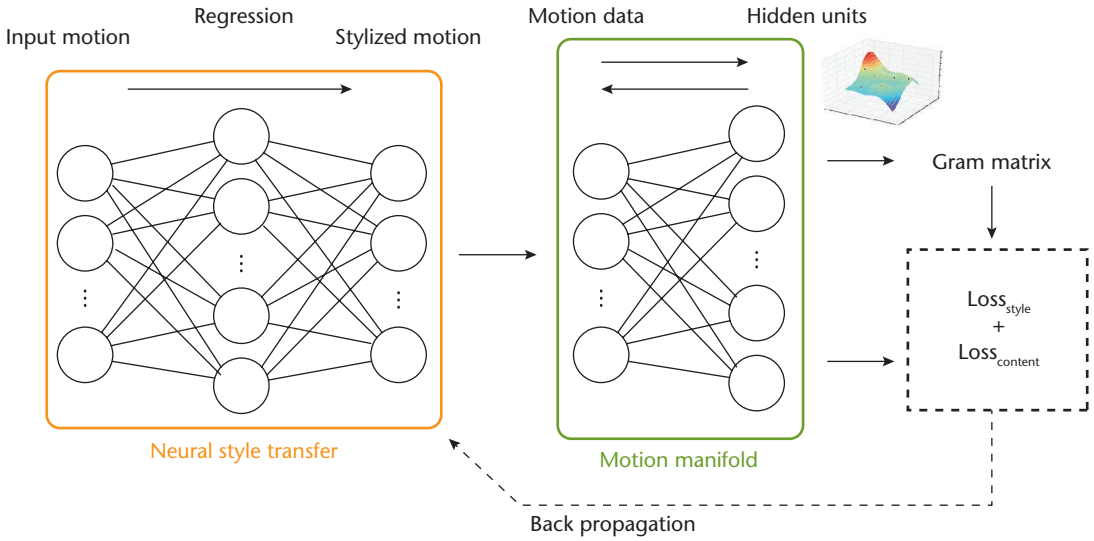

Figure 1. Overview of the two networks used in the proposed system. The transformation network $\mathcal{T}$ on the left (in orange) is a feed-forward convolutional network that performs the style transformation. The loss network $\mathcal{L}$ on the right (in green) is a convolutional autoencoder that represents a manifold over human motion and has two tasks: first help compute the loss between motion content and style and second reproject motion onto the manifold to fix any small artifacts resulting from the style transfer.

and the recurrent temporal restricted Boltzmann machine (RTRBM).[17] Although both approaches successfully produce human motion data, these motions are noisy because of the per-frame sampling and because they often converge to an average pose as a result of mapping ambiguity. Katerina Fragkiadaki and her colleagues proposed a variant of recurrent neural networks to learn and synthesize human motion data known as the encoder-recurrent-decoder network.[18] Work by Holden and his colleagues showed that a convolutional autoencoder can be used to learn a manifold of human motion.[19] This has many purposes in research including reconstructing, cleaning, or denoising motion data. Furthermore, the authors combined this autoencoder with a feed-forward neural network to regress from high-level user inputs to full body motion data.[2] They then provided a framework to edit the generated motions using the motion manifold and by optimizing the motion in the hidden unit space to satisfy constraints such as bone length and foot sliding. This editing process can also be used to convert the motion style using the Gram matrix in a scheme similar to the Gatys approach.[1]

Methodology

As we mentioned earlier, our proposed network architecture is similar to the image transformation technique proposed by Johnson and his colleagues.[3] Figure 1 shows an overview of our system, which consists of two separate neural-network structures: a convolutional autoencoder network that serves as the *loss network* $\mathcal{L}$ and a feed-forward convolutional network to perform motion style transformation, which we refer as the *transformation network* $\mathcal{T}$.

Data Representation

Each pose is represented by the 21 joint positions in 3D Euclidean space local to the character root projected onto the floor plane. Appended to this are three additional variables for turning, forward, and sideways velocities. This gives us 66 degrees of freedom (DOFs) in total per pose.

Although our proposed framework can be used to process motion data of any length, for training purposes motion data is split into overlapping windows of 240 frames at 60 frames per second. Any motions with less than 240 frames are padded with first and last frames.

Loss Network

Unlike Johnson and his colleagues[3] who used a pretrained image classification network, the loss network $\mathcal{L}$ in our system is a convolutional autoencoding network trained to reconstruct the motion data $\mathbf{X}$, which it receives as input. The purpose of using such a network is to learn the latent, manifold structure of the motion database.[19] This lets us partially emulate the classification network used in Johnson's work,[3] but in an unsupervised fashion.

Holden and his colleagues showed that a single convolution and pooling layer was enough to effectively capture motion content style in the latent

variables, and as such, a deeper or more complex network is not required.[2] We therefore follow the Holden construction, which we briefly revise. The loss network is built from a single convolution and pooling layer that uses a 1D convolutional calculated over the temporal dimension. The encoding operation can be written as follows:

$$\mathcal{L}(\mathbf{X}) = \text{ReLU}(\Psi(\mathbf{X} * \mathbf{W}_0 + \mathbf{b}_0)), \quad (1)$$

where the weight matrix $\mathbf{W}_0$ represents a tensor that consists of m hidden units (in our case, m = 256), d features (in our case, d = 66), and a temporal filter width of w_0 (in our case, w_0 = 25). The vector $\mathbf{b}_0$ represents the layer's biases, and the operation Ψ represents a max-pooling operation used to reduce the number of features over the temporal axis by taking the maximum value between two consecutive parameters. We use rectified linear units (ReLU)[20] as an activation function to provide nonlinearity to the network. A filter width of 25 corresponds to approximately half a second of motion after the pooling has taken place, which is a reasonable length of time to represent most movements or movement components. Because our initial input has 66 dimensions, using 256 hidden units allows us to represent motion using a sparse basis, which is more natural than the fully correlated basis that is often found via methods such as principal component analysis (PCA). Similarly, the decoding operation of the convolutional autoencoder is defined as follows:

$$\mathcal{L}^{\dagger}(\mathbf{H}) = (\Psi^{\dagger}(\mathbf{H}) - \mathbf{b}_0) * \tilde{\mathbf{W}}_0, \quad (2)$$

where $\mathbf{H}$ are the hidden units produced by the forward operation, $\Psi^{\dagger}$ is the depooling operation (in our case, average depooling), and $\tilde{\mathbf{W}}_0$ is the deconvolution operation.

To train the convolutional autoencoder, the input motion data $\mathbf{X}$ is encoded into the hidden unit space, and then decoded back to reconstruct $\tilde{\mathbf{X}}$. The loss function

$$\text{Loss}(\mathbf{X}, \theta) = \left\| \mathbf{X} - \mathcal{L}^{\dagger}\left(\mathcal{L}(\mathbf{X})\right) \right\|_2^2 + \alpha \|\theta\|_1 \quad (3)$$

is then back-propagated to optimize the network parameters $\theta = \{\mathbf{W}_0, \mathbf{b}_0\}$ using stochastic gradient descent, where the second term is the L_1 sparsity regularization term.

Training takes approximately 6 hours on a Nvidia GeForce GTX 660 GPU, but because we use the pretrained network from earlier work,[2] for the purposes of this article, training is not required.

Transformation Network

The transformation network $\mathcal{T}$ is a simple three-layer convolutional neural network, where each layer has a construction similar to the loss network's encoding layer without the pooling operation. Each layer contains 128 hidden units, and all but the final output layer use ReLU as a nonlinearity.

Training

The goal of this work is to train the transformation network $\mathcal{T}$ without explicitly providing the desired inputs and outputs. To do this, we use the given loss network to encode various constraints based on the preferred transformed motion.

During training, we define the transformation network's loss function using the following terms.

Content. To ensure the transformation network's output contains the content of the input motion $\mathbf{I}$, we define content loss to be the difference between the hidden unit values of the input motion when passed to the loss network $\mathcal{L}$ and the hidden unit values of the transformed motion $\mathcal{T}(\mathbf{I})$ when passed to the loss network. This comparison can be thought of as encoding the distance along the motion manifold[19] between the transformed and untransformed motion. This is scaled by a user-specified weight c (in our case, 1.0):

$$\text{Loss}_{\text{content}} = c \left\| \mathcal{L}(\mathbf{I}) - \mathcal{L}(\mathcal{T}(\mathbf{I})) \right\|. \quad (4)$$

Style. To ensure the transformation network's output contains the style of the given style clip $\mathbf{S}$, we define the style loss to be the difference between the Gram matrix of the hidden unit values of the transformed motion when passed to the loss network and the Gram matrix of the hidden unit values of the style motion when passed to the loss network. This follows the work of Gatys and his colleagues,[1,21] and comparison ensures the style of the motion produced by the transformation network matches that of the given style clip S. This is scaled by a user-specified weight s (in our case, 0.01):

$$\text{Loss}_{\text{style}} = s \left\| \text{Gram}(\mathcal{L}(\mathbf{S})) - \text{Gram}(\mathcal{L}(\mathcal{T}(\mathbf{I}))) \right\|. \quad (5)$$

The Gram matrix represents the sum over the temporal axis i of the inner product of the hidden unit values and is given by the following:

$$\text{Gram}(\mathbf{H}) = \sum_i \mathbf{H}_i \mathbf{H}_i^T. \quad (6)$$

By pairing stylized clips to their associated content clips during training, we train a transformation

Figure 2. Manifold projection. (a) Images generated by deep neural networks often contain high-frequency noise and other artifacts. (b) Motion generated by our transformation network also sometimes has these problems (see the gray figures), but we fix this using an additional step of projecting back onto the motion manifold previously found using the convolutional autoencoder (see the blue figures).

network that can apply the style in different contexts. For example, given an angry style, all training clips where the content contains running are paired with the "angry run" style, and all content clips containing walking are paired with the "angry walk" style. In this way, although a different network is required for each style, a single network can be trained to work for a variety of content.

Constraints. In addition to the content and style, it is important that the transformed motion $\mathcal{T}(\mathbf{I})$ respect human-motion constraints. This means there should be no foot-sliding artifacts, the bone lengths must not vary, and the trajectory should be similar to the input motion in shape. We therefore calculate additional loss functions relating to human-motion constraints. All these constraints must be scaled with respect to the units of the input data.

To remove foot sliding, when the feet are considered in contact with the ground (a variable previously labeled in the training data), for each foot joint j, we ensure that the sum of the joint's local velocity $\mathbf{v}_j^{\mathcal{T}(\mathbf{I})}$, local translational velocity $\mathbf{v}_r^{\mathcal{T}(\mathbf{I})}$, and the local rotational velocity $\omega^{\mathcal{T}(\mathbf{I})} \times \mathbf{p}_j^{\mathcal{T}(\mathbf{I})}$ negate the velocity of the root of the character relative to the forward direction $\mathbf{r}'$:

$$\text{Loss}_{\text{foot}} = \sum_j \left\| \mathbf{v}_r^{\mathcal{T}(\mathbf{I})} + \omega^{\mathcal{T}(\mathbf{I})} \times \mathbf{p}_j^{\mathcal{T}(\mathbf{I})} + \mathbf{v}_j^{\mathcal{T}(\mathbf{I})} - \mathbf{r}' \right\|^2. \quad (7)$$

For each bone b consisting of joints j_1 and j_2, we calculate the distance between the joint positions in the joint space of the transformed motion given by $\mathbf{p}_{b_{j_1}}^{\mathcal{T}(\mathbf{I})}$ and $\mathbf{p}_{b_{j_2}}^{\mathcal{T}(\mathbf{I})}$ and calculate the mean-squared difference of this with the given bone length l_b:

$$\text{Loss}_{\text{bone}} = \sum_b \left\| \left\| \mathbf{p}_{b_{j_1}}^{\mathcal{T}(\mathbf{I})} - \mathbf{p}_{b_{j_2}}^{\mathcal{T}(\mathbf{I})} \right\| - l_b \right|^2. \quad (8)$$

Given desired trajectory velocities $\mathbf{v}_r'$ and desired turning angle velocities ω', we calculate the mean-squared error of these and the trajectory and turning-angle velocities of the transformed motion given by $\mathbf{v}_r^{\mathcal{T}(\mathbf{I})}$ and $\omega^{\mathcal{T}(\mathbf{I})}$:

$$\text{Loss}_{\text{traj}} = \left\| \omega^{\mathcal{T}(\mathbf{I})} - \omega' \right\| + \left\| \mathbf{v}_r^{\mathcal{T}(\mathbf{I})} - \mathbf{v}_r' \right\|^2. \quad (9)$$

Training. Thus, the final loss function that takes into account the content, style, and motion constraints is finally computed as follows:

$$\text{Loss} = \text{Loss}_{\text{content}} + \text{Loss}_{\text{style}} + \text{Loss}_{\text{foot}} + \text{Loss}_{\text{bone}} + \text{Loss}_{\text{traj}}. \quad (10)$$

Given this loss function, the transformation network $\mathcal{T}$ is trained using stochastic gradient descent with automatic derivative calculation performed using Theano. We use the momentum-based optimization algorithm Adam to improve learning speed.[22] The transformation network is trained for 100 epochs using a locomotion dataset of about 20 minutes of locomotion data. Training for a single style takes around 20 minutes on a Nvidia GeForce GTX 970. Once trained, the transformation network can perform the style transfer task very quickly.

Manifold Projection. Data produced using deep neural networks often contains small amounts of noise or other artifacts (see Figure 2). These small errors may be acceptable for images, but for motion data, they greatly impact the visual quality of the result.

To solve this problem, we repurpose the pretrained convolutional autoencoder used as the loss network and, following earlier work,[19] pass the transformation motion through it to project it back onto the motion manifold, effectively removing any undesirable noise or artifacts.

Experimental Results

We tested our method on 10 different styles and various kinds of locomotion including walking, jogging, and running. In addition to a qualitative evaluation, we compared our results with the most

similar previous work: the Holden optimization-based style-transfer method.[2] For a more detailed look at the results, see the supplementary video at https://youtu.be/AvVQZjygQBQ.

Figure 3 shows a selection of style-transfer results for different styles of locomotion including old man, zombie, injured, and depressed, and Figure 4 shows a comparison of the two techniques. Our method produces results that are in most cases visually similar, but were generated with a much shorter runtime.

Table 1 gives a breakdown of the runtimes for each method. Our method has a vastly superior runtime and better scaling characteristics due to the transformation network's implicit parallelism. All performance characteristics were recorded on a Nvidia GeForce GTX 660 GPU.

Unlike the Holden method,[2] our method requires training a database of motion data similar in content to the kind of motions that will be supplied at runtime. For example, training our method to perform style transfer for locomotion data requires a database of locomotion. For more specific motions, it may be difficult or undesirable to acquire such a database.

Our method also still requires the whole motion to be specified upfront and thus will require further adaptations for use in interactive applications. Additionally, although neural-style transfer requires little manual intervention and no data alignment, compared with previous style-transfer techniques, it can be difficult to control from an artistic standpoint, and often there is no clear way to fix undesirable results.

Experimentally, we found our method produces good results for motions that are not primarily locomotion, such as punching and kicking. Because neural-style transfer is difficult to understand and control, future experimentation is necessary to see how it can most effectively be applied to different kinds of motion.

The manifold-projection operation used to remove artifacts from the stylized motion could introduce foot sliding. In our results, we did not attempt to remove this, showing only the system's raw output. To be useful in production, this must be removed via some fast postprocess such as analytical inverse kinematics (IK).

The framework we present here is specific to style transfer, but it might be useful to learn other transformations of motion data given a single exemplar (in our case, this would be the single style clip). For example, it might be possible to use this framework to transform motions for use on characters of different sizes, weights, or structures.

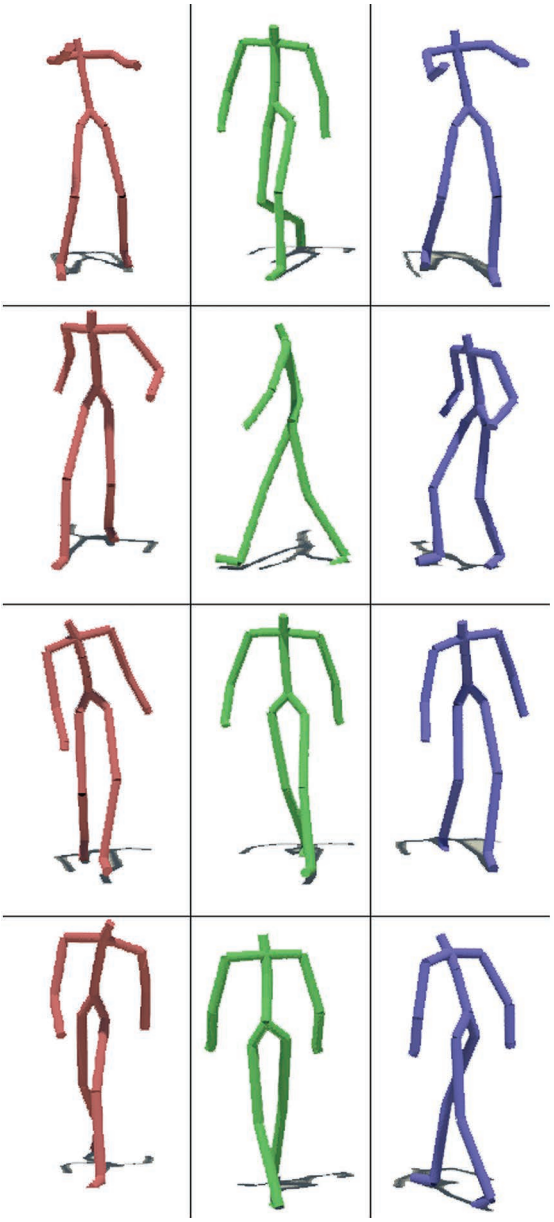

Figure 3. Sample results from the proposed method. The red figures indicate style, green content, and blue transferred. From top to bottom, the figures reflect different styles of locomotion: zombie, old man, injured, and depressed.

References

1. L.A. Gatys, A.S. Ecker, and M. Bethge, "A Neural Algorithm of Artistic Style," arXiv preprint, Cornell Univ. Library Archive, 2015, arXiv:1508.06576.
2. D. Holden, J. Saito, and T. Komura, "A Deep Learning Framework for Character Motion Synthesis and Editing," *ACM Trans. Graphics*, vol. 35, no. 4, 2016, article no. 138.
3. J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution," arXiv preprint, Cornell Univ. Library Archive, 2016, arXiv:1603.08155.
4. M. Unuma, K. Anjyo, and R. Takeuchi, "Fourier

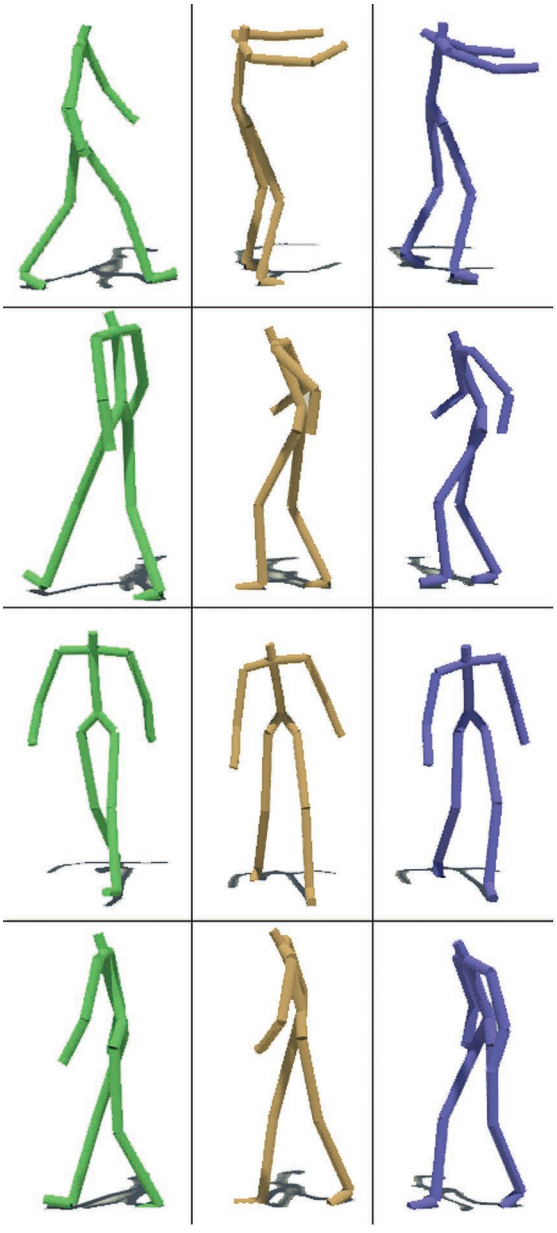

Figure 4. Visual comparison of the proposed method and the Holden method.[2] The green figures indicate content, orange the Holden method, and blue our method. From top to bottom, the figures reflect different styles of locomotion: zombie, old man, injured, and depressed.

Table 1. Performance comparison between our method and the Holden method.[2]

Method	Frames	Runtime (sec)	Frames per second
Our method	240	0.0120	21,818.18
Our method	480	0.0140	33,802.81
Our method	960	0.0170	55,813.95
Holden method	240	15.219	15.76
Holden method	480	20.341	23.59
Holden method	960	33.371	28.76

Principles for Emotion-Based Human Figure Animation," *Proc. 22nd Ann. Conf. Computer Graphics and Interactive Techniques*, 1995, pp. 91–96.

5. A. Bruderlin and L. Williams, "Motion Signal Processing," *Proc. 22nd Ann. Conf. Computer Graphics and Interactive Techniques*, 1995, pp. 97–104.
6. K. Pullen and C. Bregler, "Motion Capture Assisted Animation: Texturing and Synthesis," *ACM Trans. Graphics*, vol. 21, no. 3, 2002, pp. 501–508.
7. M.E. Yumer and N.J. Mitra, "Spectral Style Transfer for Human Motion between Independent Actions," *ACM Trans. Graphics*, vol. 35, no. 4, 2016, article no. 137.
8. E. Hsu, K. Pulli, and J. Popović, "Style Translation for Human Motion," *ACM Trans. Graphics*, vol. 24, no. 3, 2005, pp. 1082–1089.
9. J. Min, H. Liu, and J. Chai, "Synthesis and Editing of Personalized Stylistic Human Motion," *Proc. ACM Siggraph Symp. Interactive 3D Graphics and Games*, 2010, pp. 39–46.
10. S. Xia et al., "Realtime Style Transfer for Unlabeled Heterogeneous Human Motion," *ACM Trans. Graphics*, vol. 34, no. 4, 2015, article no. 119.
11. C. Dong et al., "Learning a Deep Convolutional Network for Image Super-Resolution," *Proc. European Conf. Computer Vision*, 2014, pp. 184–199.
12. Z. Cheng, Q. Yang, and B. Sheng, "Deep Colorization," *Proc. IEEE Int'l Conf. Computer Vision* (ICCV), 2015, pp. 415–423.
13. J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (CVPR), 2015, pp. 3431–3440.
14. M.D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," *Proc. European Conf. Computer Vision*, 2014, pp. 818–833.
15. F.G. Harvey and C. Pal, "Semi-supervised Learning with Encoder-Decoder Recurrent Neural Networks: Experiments with Motion Capture Sequences," arXiv preprint, Cornell Univ. Library Archive, 2015, arXiv:1511.06653.
16. G.W. Taylor and G.E. Hinton, "Factored Conditional Restricted Boltzmann Machines for Modeling Motion Style," *Proc. 26th Ann. Int'l Conf. Machine Learning* (ICML), 2009, pp. 1025–1032.
17. I. Sutskever, G.E. Hinton, and G.W. Taylor, "The Recurrent Temporal Restricted Boltzmann Machine," *Proc. Advances in Neural Information Processing Systems* (NIPS), 2009, pp. 1601–1608.
18. K. Fragkiadaki et al., "Recurrent Network Models for Human Dynamics," *Proc. IEEE Int'l Conf. Computer Vision* (ICCV), 2015, pp. 4346–4354.
19. D. Holden et al., "Learning Motion Manifolds with Convolutional Autoencoders," *SIGGRAPH Asia 2015 Tech. Briefs*, 2015, article no. 18.
20. V. Nair and G.E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," *Proc. 27th Int'l Conf. Machine Learning* (ICML), 2010, pp. 807–814.
21. L. Gatys, A.S. Ecker, and M. Bethge, "Texture Synthesis Using Convolutional Neural Networks,"

Proc. Advances in Neural Information Processing Systems (NIPS), 2015, pp. 262–270.
22. D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," arXiv preprint, Cornell Univ. Library Archive, 2014, arXiv:1412.6980.

Daniel Holden *recently completed his PhD in informatics with the Institute of Perception, Action, and Behavior, School of Informatics at the University of Edinburgh. His research focuses on applying machine learning to character animation in key-framed animation environments and developing deep-learning frameworks for character animation editing and synthesis. Contact him at s0822954@sms.ed.ac.uk.*

Ikhsanul Habibie *is a visiting scholar with the School of Informatics at the University of Edinburgh. His research interests include machine learning, computer vision, biomedical engineering, and deep-learning methods for computer animation application. Habibie has an MS in informatics from the University of Edinburgh. Contact him at abie.ikhsan@gmail.com.*

Ikuo Kusajima *is a PhD student in the Department of Mechano-Informatics, Graduate School of Information Science and Technology, at the University of Tokyo. His research interests include robot intelligence and human motion and language in robotics. Kusajima has an MS in mechano-informatics from the University of Tokyo. Contact him at kusajima@ynl.t.u-tokyo.ac.jp.*

Taku Komura *is a reader (associate professor) with the Institute of Perception, Action and Behavior, School of Informatics, and head of the Computer Animation and Robotics Group at the University of Edinburgh. His research interests include data-driven character animation, physically based character animation, crowd simulation, cloth animation, anatomy-based modeling, and robotics. Komura has a PhD in information science from the University of Tokyo. Contact him at tkomura@ed.ac.uk.*