VDSM: Unsupervised Video Disentanglement with State-Space Modeling and Deep Mixtures of Experts (Supplementary Material)

Matthew J. Vowels
m.j.vowels@surrey.ac.uk

Necati Cihan Camgoz
n.camgoz@surrey.ac.uk

Richard Bowden
r.bowden@surrey.ac.uk

Centre for Vision, Speech and Signal Processing
University of Surrey
Guildford, UK

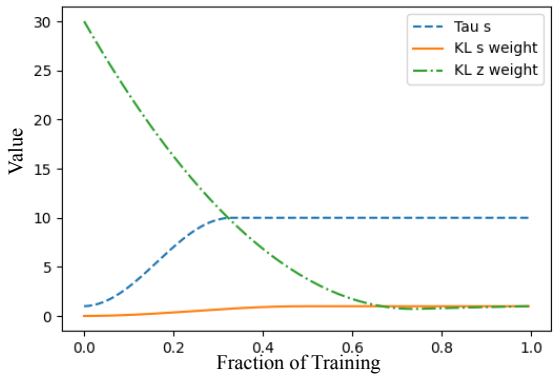

Figure 1. Annealing schedules and temperature schedules for VDSM during pretraining. Shows the weight λ_z on the KL divergence for the time varying pose factor $\mathbf{z}_t^n$, the weight λ_s on the KL divergence for the identity factor $\mathbf{s}^n$, and the temperature τ_s for the identity factor.

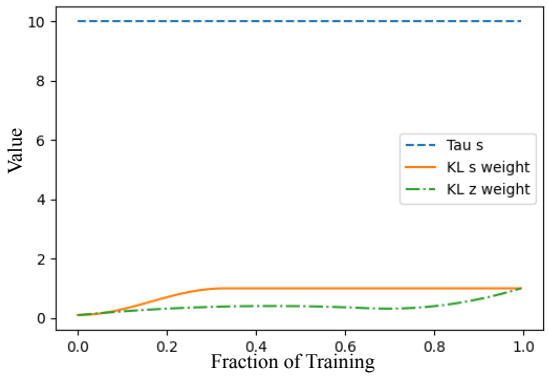

Figure 2. Annealing schedules and temperature schedules for VDSM during training of the sequential components. Shows the weight λ_z on the KL divergence for the time varying pose factor $\mathbf{z}_t^n$, the weight λ_s on the KL divergence for the identity factor $\mathbf{s}^n$ (constant during this phase of training), and the temperature τ_s for the identity factor.

1. Code and Qualitative Video Samples

Code and example video can be found via the following URL: **https://github.com/matthewvowels1/DisentanglingSequences** as well as in the 'samples' folder in the supplementary material.[1]

2. Overview of Supplementary Material

This supplementary material provides additional information and results for the work titled 'VDSM: Unsupervised Video Disentanglement with State-Space Modeling and Deep Mixtures of Experts'. We first provide details about the network architecture, training details, run-time estimates, and briefly discuss the results of a simple ablation experiment. Qualitative results for the synthetic pendulum dataset are presented in Figure 3, and this is the figure referenced in Section 4.2 of the main paper. We then provide a derivation for the ELBO presented in Equation 4 in the main text, and finally present a range of qualitative results for the MUG [1], Sprites [5], moving MNIST [8] and synthetic pendulum dataset.

3. Network Architecture

The network was implementation using a combination of Pytorch [7] and Pyro [2], and the code has been included as part of the supplementary material. Various relevant hyperparameters and dimensionalities are shown in Table 1 and 2.

Encoder and Static Factors: The encoder comprises the following blocks:

[1]These are .gif files which may need to be viewed in (e.g.) an internet browser for animation.

Dataset	lr-Pre	lr-Seq	Epochs-Pre	Epochs-Seq	BS-Pre	BS-seq	Seq Len	BPE-Pre	BPE-Seq
MUG	1e-3	1e-3	250	200	20	20	20	50	50
Sprites	8e-3	1e-3	300	200	20	20	8	50	50
MMNIST	1e-3	1e-3	300	200	20	30	16	50	50
Pendula	5e-4	1e-3	200	100	50	20	16	50	50

Table 1. VDSM hyperparameter settings for each dataset. 'Pre' and 'Seq' refer to the pre-training and sequence training respectively. 'BS' is batch size, 'lr' is learning rate, and 'BPE' is the number of batches per epoch.

[Conv2D(32,4,1), LeakyReLU, BlurPool],
[Conv2D(32,4,2), LeakyReLU, BlurPool],
[Conv2D(32,4,2), LeakyReLU, BlurPool],
[Conv2D(64,4,2), LeakyReLU, BlurPool],
[Conv2D(64,4,2), LeakyReLU]

where Conv2D(x, y, z) is the convolution operation with x, y, z being the number of output filters, the kernel size, and the stride, respectively. The first block (only) has padding of 1. LeakyReLU is the leaky rectified linear unit [6], and blur pool enables anti-aliased downsampling [10]. The output is reshaped and fed to separate two consequetive fully-connected layers [FC(256, 128), FC(128, $2\kappa_s$)] (where the two arguments are the number of input and output neurons) to yield the embeddings for $\mathbf{s}^n$ (the identity), and fed to a single fully connected layer [FC(256, $2\kappa_z$] to yield the embeddings for $\mathbf{z}_t^n$ (the time varying components). These embeddings are split into two to yield the location and scale parameters of the Normal distributions used to model the two factors.

Dataset	κ_z	κ_s	κ_d	*RNN Layers*	*RNN dim.*
MUG	30	15	50	3	512
Sprites	30	40	50	3	512
MMNIST	30	12	50	3	512
Pendula	30	8	50	3	512

Table 2. κ_z, κ_s, and κ_d are the dimensionalities of the pose, identity, and dynamics/action latent factors, respectively. Both the bi-LSTM encoder and the uni-directional LSTM decoder have the same number of layers and hidden dimensions (3 and 512, respectively).

Dynamics Layer, LSTMs and Combiner: The seq2seq encoder is a bi-LSTM, and the decoder is a uni-directional LSTM, each with settings listed in Table 2. The output of the bi-LSTM is a hidden representation with a dimensionality equal to $\text{RNN}_{dim} \times \text{RNN}_{layers} \times 2$. This hidden representation is fed into the full-connected dynamics layer with output dimensionality $2 \times \kappa_d$, and is split in half to yield the location and scale of $\mathbf{d}^n$. The RNN decoder outputs per-timepoint vectors which are fed through a fully-connected layer FC($\text{RNN}_{dim} \times 2, 2 \times \kappa_z$). The intermediary hidden size of the combiner network is 512, and otherwise the parameter shapes of the fully connected layers in the combiner are determined by the dimensionalities of the inputs and the outputs of the function (i.e., the dimensionalities of $\mathbf{z}_t^n, \mathbf{h}_t^n$, and $\mathbf{d}^n$).

Transition Network: The transition network follows the structure described in the main paper. The intermediary hidden size used in the network is 64 and otherwise, like the combiner network, has fully connected layer weight sizes determined by the input and output dimensionalities of the function (i.e., the dimensionalities of $\mathbf{z}_t^n$ and $\mathbf{d}^n$).

Mixture of Experts Decoder: The Mixture of Experts (MoE) decoder (or generator) has $N_s = \kappa_s$ number of decoders which each follow this structure:

[ConvTrans(1,0), LeakyReLU]
[ConvTrans(2,1), LeakyReLU]
[ConvTrans(2,1), LeakyReLU]
[ConvTrans(2,1), LeakyReLU]
[ConvTrans(2,1), LeakyReLU]

where ConvTrans is 2-dimensional transpose convolution operation. The weights and biases for the ConvTrans operations are blended using the $\mathbf{s}^n$ sample which is duplicated T_n times and concatenate with the pose vector for decoding.

Annealing Schedules: Although very little tuning was required, the schedules for annealing the weights on the KL terms in the objective do need to be considered. Figures 1 and 2 show the annealing schedules for pre-training and sequence training. The function describing the profile of the pretraining curves is sinusoidal, whereas for the sequential training λ_z curve is derived using quadratic interpolation.

4. Derivation of the Lower Bound

The derivation follows the same process as in [4]. We first present the factorization of the generative and inference models in Equations 1 and 2, respectively (it may be useful to reference the DAGs in the main paper). The compact representation of the ELBO objective is then shown in Equation 3, and its final form is shown in Equation 4. The first line in Equation 4 is derived straightforwardly according to the factorization of Equations 1 and 2. However, the second line s further attention, and relates to the time-dependent nature of the pose factor $\mathbf{z}_t^n$ and its dependence on the dynamics $\mathbf{d}^n$. Omitting the weighting factor λ_z, The second line can be compactly reduced to Equation 5. Note that the derivation and equations have been presented in single column format for legibility.

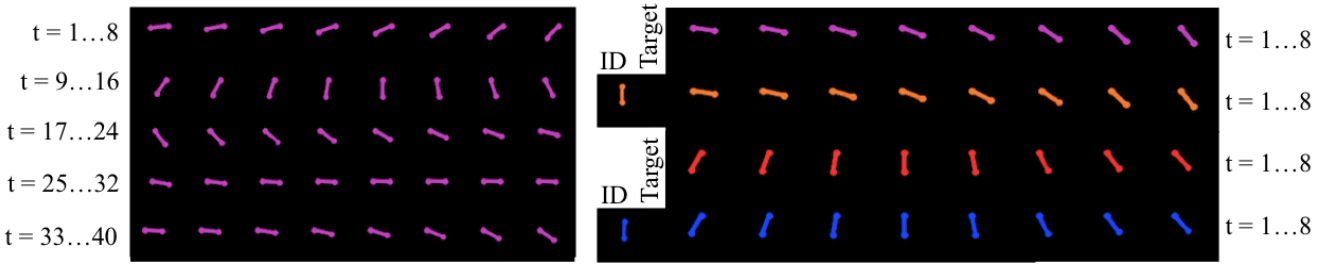

Figure 3. VDSM-generated frames for the synthetic pendulum dataset. Left figure illustrates how VDSM can be used to generate video far into the future (40 frame shown). Right figure illustrates how VDSM can transfer the action/dynamics onto a new identity. For this we simply use $\mathbf{d}^n$ from the purple or red pendulum sequences and apply it to generate sequences for the orange and blue pendula. 'Target' indicates the desired action, whilst 'ID' indicates the identity to which the action is transferred during sequence generation.

5. Additional Qualitative Results

Additional, randomly sampled qualitative results are shown in Figure 4 to 9, beginning with samples from the swinging pendulum dataset, then moving MNIST, Sprites, and finally MUG.

6. Ablation - Using a Single Decoder

Whilst the encoder used to derive a compact representation from the video frames was of comparable complexity to alternative/competing methods, the complexity of the mixture-of-experts decoder is arguably much greater. This is because it essentially comprises a bank of decoders, each which their own set of trainable weights. The network derives a mixing coefficient (which tends towards a discrete categorical latent variable) that blends or selects from the bank of decoders. Even though only one blended set of weights from the complete bank of weights is used for any one sequence, there is a significantly larger number of possible decoder configurations owing to the use of mixing.

We ran an additional experiment to explore what happens if we use only the inferred mixing coefficient as a latent variable alone, and do *not* use a bank of decoder weights (i.e. just a single decoder). We found that the reduced model resulted in a complete failure of the model to disentangle identity from pose (*i.e.* both factors were highly entangled, and identify/pose swapping was not possible). It is difficult to ascertain to what extent this failure is due to the reduction in complexity associated with the use of single set of decoder weights, and to what extent it has something to do with a difference in resulting optimization dynamics which lead to different convergence properties. One possible way to establish this would involve a full hyperparameter search over the reduced model (the one without the mixture of decoders) to understand whether it is possible to achieve convergence. We leave this to future work.

7. Hardware and Run Times

The model was trained an tested on a GPU (e.g. NVIDIA 2080Ti) driven by a 3.6GHz Intel I9-9900K CPU running Ubuntu 18.04. Using the Sprites dataset by way of example, pre-training (1st stage) took 15 seconds for each of the 300 epochs, completing in 75 minutes. Sequence training (2nd stage) took 43 seconds for each of the 200 epochs, completing in approximately 2 hours 20 minutes. It is worth noting that pretraining and sequence training was found to converge significantly faster - as few as 100 epochs and 80 epochs respectively - corresponding to a total training time of approximately 80 minutes. However, a limited hyperparameter space was explored for this work, and we leave detailed efficiency studies to future work. At inference time, it was found that a batch of 20 sequences could be generated and saved to disk in approximately 0.2 seconds.

Using the code provided here: **https://github.com/DLHacks/mocogan** we ran a $64 \times 64 \times 3$ version of the Weizmann dataset [3] to get an approximate training time comparison against MoCoGAN [9]. The default frame dimensions are $96 \times 96 \times 3$, and so it was first necessary to modify the generators, discriminators, and dataset, accordingly. Using the default training settings for this dataset resulted in a total training time of 5 hours 3 minutes (0.2 seconds per iteration, for 100,000)). Even though this is a fast and loose comparison (with smaller data), it does suggest that VDSM training time (both stages included) may considerably faster than that of MoCoGAN.

Generative model:

$$p_\theta(\mathbf{x}^n_{t=1:T_n}, \mathbf{s}^n, \mathbf{d}^n, \mathbf{z}^n_{t=1:T_n}) = p_\theta(\mathbf{s}^n)p_\theta(\mathbf{d}^n)p_\theta(\mathbf{z}^n_{t=1})\prod_{t=2}^{T_n} p_\theta(\mathbf{x}^n_t|\mathbf{s}^n, \mathbf{z}^n_t)p_\theta(\mathbf{z}^n_t|\mathbf{z}^n_{t-1}, \mathbf{d}^n) \tag{1}$$

Inference Model:

$$q_\phi(\mathbf{s}^n, \mathbf{d}^n, \mathbf{z}_{t=1:T_n}|\mathbf{x}^n_{t=1:T_n}) = q_\phi(\mathbf{s}^n|\mathbf{x}^n_{t=1:T_n})q_\phi(\mathbf{d}^n|\mathbf{x}^n_{t=1:T_n})q_\phi(\mathbf{z}^n_{t=1}, |\mathbf{x}^n_{t=1:T_n}, \mathbf{d}^n)\prod_{t=2}^{T_n} q_\phi(\mathbf{z}^n_t|\mathbf{z}^n_{t-1}, \mathbf{x}^n_{t=1:T_n}, \mathbf{d}^n) \tag{2}$$

ELBO Objective:

$$\max_{\phi,\theta} \left\langle \left\langle \frac{p_\theta(\mathbf{x}^n_{t=1:T_n}, \mathbf{s}^n, \mathbf{d}^n, \mathbf{z}^n_{t=1:T_n})}{q_\phi(\mathbf{s}^n, \mathbf{d}^n, \mathbf{z}^n_{t=1:T_n} \mid \mathbf{x}^n_{i=1:T_n}))} \right\rangle_{q_\phi} \right\rangle_{p_{\mathcal{D}(\mathbf{x}^n)}} \tag{3}$$

ELBO Objective (expanded):

$$\begin{aligned} &\mathcal{L}(\mathbf{x}^n_{1:T_n}; (\theta, \phi)) = \\ &\sum_{t=1}^{T_n} \mathbb{E}_{q_\phi(\mathbf{z}^n_t, \mathbf{s}^n|\mathbf{x}^n_{1:T_n})}[\log p_\theta(\mathbf{x}^n_t|\mathbf{z}^n_t, \mathbf{s}^n)] - \lambda_d(\mathrm{KL}(q_\phi(\mathbf{d}^n|\mathbf{x}^n_{1:T_n})||p_\theta(\mathbf{d}^n))) - \lambda_s(\mathrm{KL}(q_\phi(\mathbf{s}^n|\mathbf{x}^n_{1:T_n})||p_\theta(\mathbf{s}^n))) \\ &-\lambda_z(\mathrm{KL}(q_\phi(\mathbf{z}^n_1|\mathbf{x}^n_{1:T_n}, \mathbf{d}^n)||p_\theta(\mathbf{z}^n_1)) - \lambda_z \sum_{t=2}^{T_n} \mathbb{E}_{q_\phi(\mathbf{z}^n_{t-1}|\mathbf{x}^n_{1:T_n}, \mathbf{d}^n)}\mathrm{KL}(q_\phi(\mathbf{z}^n_t|\mathbf{z}^n_{t-1}, \mathbf{d}^n, \mathbf{x}^n_{1:T_n})||p_\theta(\mathbf{z}^n_t|\mathbf{z}^n_{t-1}, \mathbf{d}^n))) \end{aligned} \tag{4}$$

Time Varying KL Term (Compact):

$$\mathrm{KL}(q_\phi(\mathbf{z}^n_1, ..., \mathbf{z}^n_{T_n}|\mathbf{x}_{1:T_n}, \mathbf{d}^n)||p_\theta(\mathbf{z}^n_1, ..., \mathbf{z}^n_{T_n}|\mathbf{d}^n))) \tag{5}$$

Time Varying KL Derivation:

$$\begin{aligned} &\mathrm{KL}(q_\phi(\mathbf{z}^n_1, ..., \mathbf{z}^n_{T_n}|\mathbf{x}_{1:T_n}, \mathbf{d}^n)||p_\theta(\mathbf{z}^n_1, ..., \mathbf{z}^n_{T_n}|\mathbf{d}^n))) = \\ &\int_{\mathbf{z}^n_1} .. \int_{\mathbf{z}^n_{T_n}} q_\phi(\mathbf{z}^n_1|\mathbf{x}_{1:T_n}, \mathbf{d}^n)...q_\phi(\mathbf{z}^n_{T_n}|\mathbf{x}_{1:T_n}, \mathbf{d}^n, \mathbf{z}^n_{T_n-1}) \log \frac{p_\theta(\mathbf{z}^n_1, ..., \mathbf{z}^n_{T_n}|\mathbf{d}^n)}{q_\phi(\mathbf{z}^n_1|\mathbf{x}_{1:T_n}, \mathbf{d}^n)...q_\phi(\mathbf{z}^n_{T_n}|\mathbf{x}_{1:T_n}, \mathbf{d}^n, \mathbf{z}^n_{T_n-1})} = \\ &\int_{\mathbf{z}^n_1} .. \int_{\mathbf{z}^n_{T_n}} q_\phi(\mathbf{z}^n_1|\mathbf{x}_{1:T_n}, \mathbf{d}^n)...q_\phi(\mathbf{z}^n_{T_n}|\mathbf{x}_{1:T_n}, \mathbf{d}^n, \mathbf{z}^n_{T_n-1}) \log \frac{p_\theta(\mathbf{z}^n_1)p_\theta(\mathbf{z}^n_2|\mathbf{z}^n_1, \mathbf{d}^n)...p_\theta(\mathbf{z}^n_{T_n}|\mathbf{z}^n_{T_n-1}, \mathbf{d}^n)}{q_\phi(\mathbf{z}^n_1|\mathbf{x}_{1:T_n}, \mathbf{d}^n)...q_\phi(\mathbf{z}^n_{T_n}|\mathbf{x}_{1:T_n}, \mathbf{d}^n, \mathbf{z}^n_{T_n-1})} = \\ &\int_{\mathbf{z}^n_1} .. \int_{\mathbf{z}^n_{T_n}} q_\phi(\mathbf{z}^n_1|\mathbf{x}_{1:T_n}, \mathbf{d}^n)...q_\phi(\mathbf{z}^n_{T_n}|\mathbf{x}_{1:T_n}, \mathbf{d}^n, \mathbf{z}^n_{T_n-1}) \log \frac{p_\theta(\mathbf{z}^n_1)}{q_\phi(\mathbf{z}^n_1|\mathbf{x}^n_{1:T_n}, \mathbf{d}^n)} \\ &+ \sum_{t=2}^{T_n} \int_{\mathbf{z}^n_1} .. \int_{\mathbf{z}^n_{T_n}} q_\phi(\mathbf{z}^n_1|\mathbf{x}_{1:T_n}, \mathbf{d}^n)...q_\phi(\mathbf{z}^n_{T_n}|\mathbf{x}_{1:T_n}, \mathbf{d}^n, \mathbf{z}^n_{T_n-1}) \log \frac{p_\theta(\mathbf{z}^n_t|\mathbf{z}^n_{t-1}, \mathbf{d}^n)}{q_\phi(\mathbf{z}^n_t|\mathbf{z}^n_{t-1}, \mathbf{x}^n_{1:T_n}, \mathbf{d}^n)} = \\ &\int_{\mathbf{z}^n_1} q_\phi(\mathbf{z}_1|\mathbf{x}^n_{1:T_n}, \mathbf{d}^n) \log \frac{p_\theta(\mathbf{z}^n_1)}{q_\phi(\mathbf{z}^n_1|\mathbf{x}^n_{1:T_n}, \mathbf{d}^n)} + \sum_{t=2}^{T_n} \int_{\mathbf{z}^n_{t-1}} \int_{\mathbf{z}^n_t} q_\phi(\mathbf{z}^n_t|\mathbf{x}^n_{1:T_n}, \mathbf{z}^n_{t-1}\mathbf{d}^n) \log \frac{p_\theta(\mathbf{z}^n_t|\mathbf{z}^n_{t-1}, \mathbf{d}^n)}{q_\phi(\mathbf{z}^n_t|\mathbf{z}^n_{t-1}, \mathbf{x}^n_{1:T_n}, \mathbf{d}^n)} = \\ &\mathrm{KL}(q_\phi(\mathbf{z}^n_1|\mathbf{x}^n_{1:T_n}, \mathbf{d}^n)||p_\theta(\mathbf{z}^n_1)) + \sum_{t=2}^{T_n} \mathbb{E}_{q_\phi(\mathbf{z}^n_{t-1}|\mathbf{x}^n_{1:T_n}, \mathbf{d}^n)}\mathrm{KL}(q_\phi(\mathbf{z}^n_t|\mathbf{z}^n_{t-1}, \mathbf{d}^n, \mathbf{x}^n_{1:T_n})||p_\theta(\mathbf{z}^n_t|\mathbf{z}^n_{t-1}, \mathbf{d}^n))) \end{aligned} \tag{6}$$

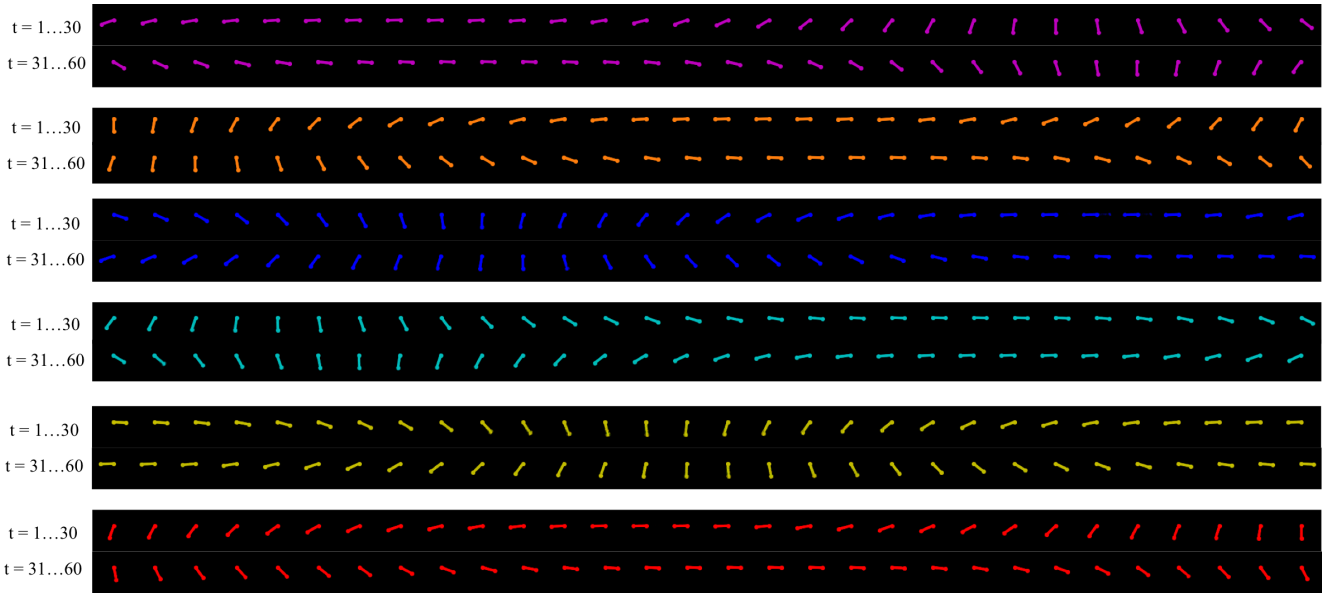

Figure 4. VDSM pendulum dataset sequence generation samples, 60 timesteps into the future.

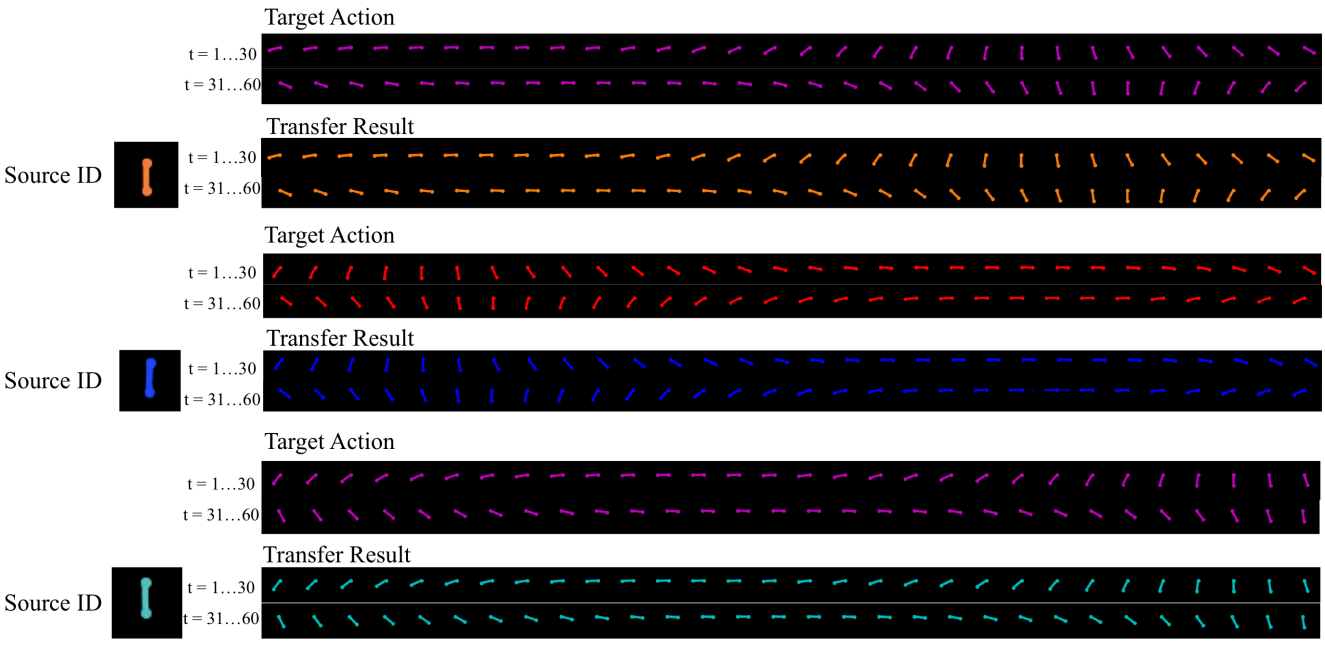

Figure 5. VDSM pendulum dataset sequence action transfer samples, 60 timesteps into the future.

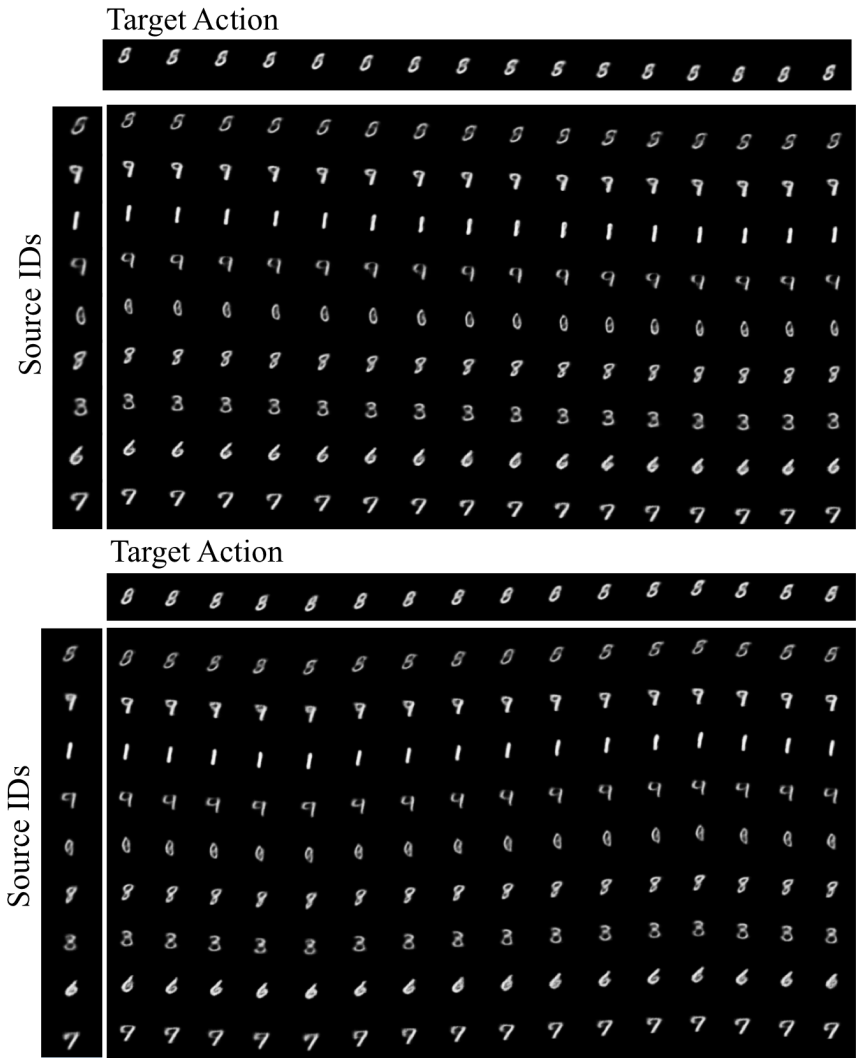

Figure 6. VDSM moving MNIST dataset action transfer, 16 timesteps into the future.

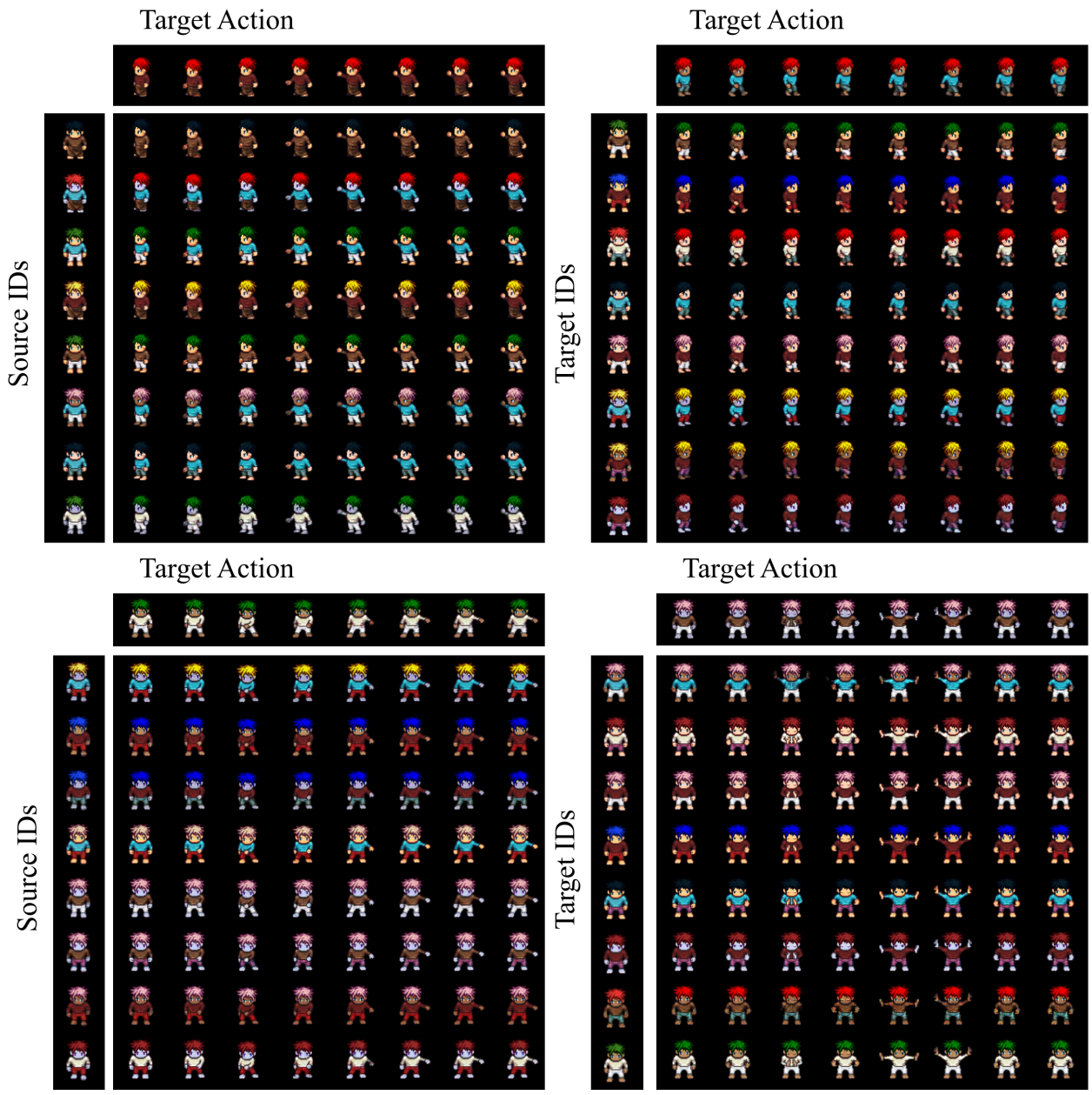

Figure 7. VDSM animated Sprites dataset action transfer, 8 timesteps into the future.

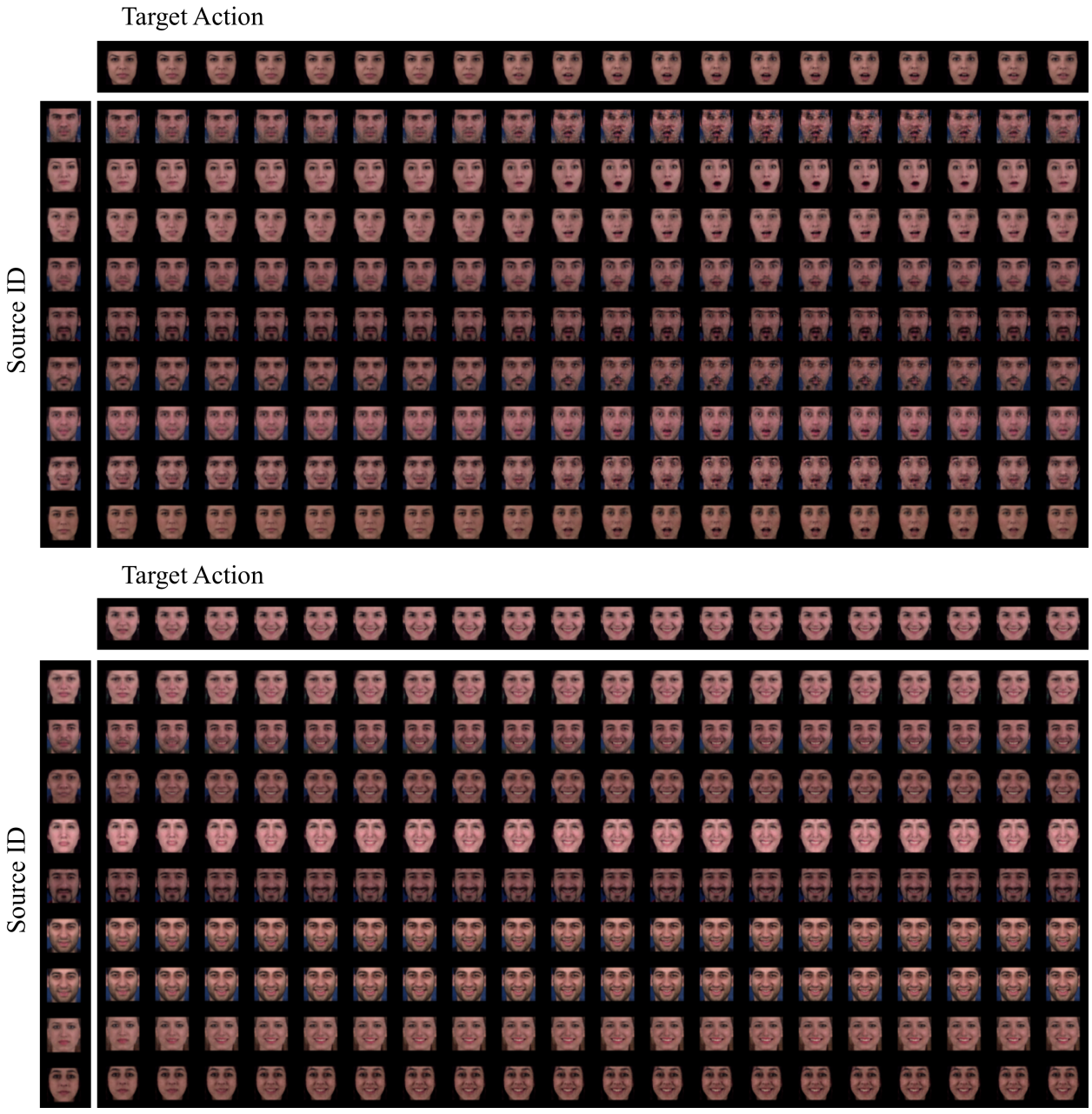

Figure 8. VDSM MUG dataset action transfer, 20 timesteps into the future.

Figure 9. VDSM MUG dataset action transfer, 20 timesteps into the future.

References

[1] N. Aifanti, C. Papachristou, and A. Delopoulos. The MUG facial expression database. *Proc. 11th Int. Workshop on Image Analysis for Multimedia Interactive Services*, 2010. 1

[2] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul A. Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep universal probabilistic programming. *J. Mach. Learn. Res.*, 20, 2019. 1

[3] L. Gorelick, E. Shechtman, M. Irani, and R. Basri. Actions as space-time shapes. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 29(12), 2007. 3

[4] R. G. Krishnan, U. Shalit, and D. Sontag. Structured inference networks for nonlinear state space models. *Association for the Advancement of Artificial Intelligence*, 2017. 2

[5] K. Li and J. Malik. Implicit maximum likelihood estimation. *arXiv:1809.09087*, 2018. 1

[6] A.L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. *ICML*, 30, 2013. 2

[7] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. *NeurIPS Workshop*, 2017. 1

[8] N. Srivastava, E. Mansimov, and R. Salakhutdinov. Unsupervised learning of video representations using LSTMs. *arXiv:1502.04681v3*, 2016. 1

[9] S. Tulyakov, M-Y. Liu, and J. Kautz. MoCoGAN: decomposing motion and content for video generation. *arXiv:1707.04993v2*, 2017. 3

[10] R. Zhang. Making convolutional networks shift-invariant again. *Proceedings of the 36th International Conference on Machine Learning*, 2019. 2