STOCHASTIC LATENT RESIDUAL VIDEO PREDICTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Video prediction is a challenging task: models have to account for the inherent uncertainty of the future. Most works in the literature are based on stochastic image-autoregressive recurrent networks, raising several performance and applicability issues. An alternative is to use fully latent temporal models which untie frame synthesis and dynamics. However, no such model for video prediction has been proposed in the literature yet, due to design and training difficulties. In this paper, we overcome these difficulties by introducing a novel stochastic temporal model. It is based on residual updates of a latent state, motivated by discretization schemes of differential equations. This first-order principle naturally models video dynamics as it allows our simpler, lightweight, interpretable, latent model to outperform prior state-of-the-art methods on challenging datasets.

1 INTRODUCTION

Being able to predict the future of a video from a few conditioning frames in a self-supervised manner has many applications in fields such as reinforcement learning (Gregor et al., 2019) or robotics (Babaeizadeh et al., 2018). More generally, it challenges the ability of a model to capture visual and dynamic representations of the world. Video prediction has received a lot of attention from the computer vision community. However, most proposed methods are deterministic, reducing their ability to capture video dynamics, which are intrinsically stochastic (Denton & Fergus, 2018).

Stochastic video prediction is a challenging task which has been tackled by recent works. Most state-of-the-art approaches are based on image-autoregressive models (Denton & Fergus, 2018; Babaeizadeh et al., 2018), built around Recurrent Neural Networks (RNNs), where each generated frame is fed back to the model to produce the next frame. However, performances of their temporal models innately depend on the capacity of their encoder and decoder, as each generated frame has to be re-encoded in a latent space. Such autoregressive processes induce a high computational cost, and strongly tie the frame synthesis and temporal models, which may hurt the performance of the generation process and limit its applicability (Gregor et al., 2019; Rubanova et al., 2019).

An alternative approach consists in separating the dynamic of the state representations from the generated frames, which are independently decoded from the latent space. In addition to removing the aforementioned link between frame synthesis and temporal dynamics, this is computationally appealing when coupled with a low-dimensional latent space. Moreover, such models can be used to shape a complete representation of the state of a system, e.g. for reinforcement learning applications (Gregor et al., 2019), and more interpretable than autoregressive models (Rubanova et al., 2019). Yet, these State-Space Models (SSMs) are more difficult to train as they require non-trivial latent state inference schemes (Krishnan et al., 2017) and a careful design of the dynamic model (Karl et al., 2017). This leads most successful SSMs to only be evaluated on small or artificial toy tasks.

In this work, we propose a novel stochastic dynamic model for the task of video prediction which successfully leverages structural and computational advantages of SSMs that operate on low-dimensional latent spaces. The dynamic component determines the evolution through residual updates of the latent state, conditioned on learned stochastic variables. This formulation allows us to implement an efficient training strategy and process in an interpretable manner complex high-dimensional data such as videos. This residual principle can be linked to recent advances relating residual networks and Ordinary Differential Equations (ODEs) (Chen et al., 2018). This interpretation opens new perspectives such as generating videos at different frame rates, as demonstrated in our experiments. Overall, this approach outperforms current state-of-the-art models on the task of stochastic video prediction, as demonstrated by comparisons with competitive baselines on representative benchmarks.
2 Related Work

Video synthesis covers a range of different tasks, such as video-to-video translation (Wang et al., 2018), super-resolution (Caballero et al., 2017), interpolation between frames (Jiang et al., 2018), generation (Tulyakov et al., 2018), or video prediction, which is the focus of this paper.

**Deterministic models.** Inspired by prior sequence generation models using RNNs (Graves, 2013), a number of video prediction methods (Srivastava et al., 2015; Villegas et al., 2017) rely on LSTMs (Hochreiter & Schmidhuber, 1997), or, like Ranzato et al. (2014) and Jia et al. (2016), on derived networks such as ConvLSTMs (Shi et al., 2015) taking advantage of Convolutional Neural Networks (CNNs). Indeed, computer vision approaches are usually tailored to high-dimensional video sequences and propose domain-specific techniques as they often use pixel-level transformations and optical flow (Shi et al., 2015; Walker et al., 2015; Finn et al., 2016; Jia et al., 2016; Vondrick & Torralba, 2017; Liang et al., 2017; Liu et al., 2017; Lotter et al., 2017; Lu et al., 2017a; Fan et al., 2019) that help to produce high-quality predictions. Such predictions are, however, deterministic, thus hurting their performance as they fail to generate sharp long-term video frames (Babaeizadeh et al., 2018; Denton & Fergus, 2018). Following Mathieu et al. (2016), some works proposed to use an adversarial loss (Goodfellow et al., 2014) on the predictions of their model to sharpen the generated frames (Vondrick & Torralba, 2017; Liang et al., 2017; Lu et al., 2017a; Xu et al., 2018). Nonetheless, adversarial losses are notoriously hard to train, and lead to mode collapse, preventing diversity of generations.

**Stochastic and image-autoregressive models.** Some approaches rely on exact likelihood maximization, using pixel-level autoregressive generation (van den Oord et al., 2016; Kalchbrenner et al., 2017) or normalizing flows through invertible transformations between the observation space and a latent space (Kingma & Dhariwal, 2018; Kumar et al., 2019). However, they require careful design of complex temporal generation schemes manipulating high-dimensional data, thus inducing a prohibitive temporal generation cost. More efficient continuous models rely on Variational Auto-Encoders (VAEs) (Kingma & Welling, 2014; Rezende et al., 2014) for the inference of low-dimensional latent state variables. Except Xue et al. (2016) who learn a one-frame-ahead VAE, they model sequence stochasticity by incorporating a random latent variable per frame into a deterministic image-autoregressive model. Babaeizadeh et al. (2018) integrate stochastic variables into the ConvLSTM architecture of Finn et al. (2016). Denton & Fergus (2018), He et al. (2018), and Castrejon et al. (2019) in a follow-up, use a prior LSTM to feed random variables with previous frames to a predictor LSTM. Finally, Lee et al. (2018) combine the ConvLSTM architecture and this learned prior, adding an adversarial loss on the predicted videos to sharpen them at the cost of a diversity drop. Yet, all these methods are image-autoregressive as they feed their predicted frames back into the latent space, tying the frame synthesis and temporal models and increasing their computational cost. We instead propose a fully latent model acting on a small latent state, tackling these issues.

**State-space models.** Many latent state-space models have been proposed for sequence modelization (Bayer & Osendorfer, 2014; Chung et al., 2015; Fraccaro et al., 2016; 2017; Krishnan et al., 2017; Karl et al., 2017), usually trained by deep Variational Inference (VI). These methods, which use locally linear temporal transition functions, are designed for and tested on low-dimensional data, as learning such models on complex data is challenging. In contrast, our fully latent method is the first one to be successfully applied to complex high-dimensional data such as videos, thanks to a temporal model based on residual updates of its latent state. It thus falls within the scope of a recent trend linking differential equations with neural networks (Lu et al., 2017b; Long et al., 2018), leading to the integration of ODEs, that are seen as continuous residual networks, in neural network architectures (Chen et al., 2018). However, the latter work and follow-ups (Rubanova et al., 2019; Yıldız et al., 2019) are either limited to low-dimensional data, prone to overfitting or unable to handle stochasticity within a sequence. Another line of works considers stochastic differential equations (SDEs) with neural networks (Ryder et al., 2018; De Brouwer et al., 2019), but are limited to continuous Brownian noise, whereas video prediction additionally requires to model punctual stochastic events.

3 Model

We consider the task of stochastic video prediction, consisting in approaching, given a number of conditioning video frames, the distribution of possible future frames given this conditioning.
We propose to model the transition function of the latent dynamic of \( y \) where \( \mu \) is the state of the system \( \theta \) is a normal distribution with fixed diagonal variance and mean \( \sigma \) is a probability distribution parametrized by \( \theta \). In our experiments, \( \mathcal{G}(g_\theta(y_t)) \) is a normal distribution with fixed diagonal variance and mean \( g_\theta(y_t) \). Note that \( y_1 \) is assumed to have a standard Gaussian prior, and, in our VAE setting, will be inferred from conditioning frames for the prediction task, as shown in Section 3.3.

The residual update rule takes inspiration in the Euler discretization scheme of differential equations. The state of the system \( y_t \) is updated by its first-order movement, i.e., the residue \( f_\theta(y_t, z_{t+1}) \). Compared to a regular RNN, this simple principle makes our temporal model lighter and more interpretable. Equation (1), however, differs from a discretized ODE because of the introduction of the stochastic discrete-time variables \( z \). Nonetheless, we propose to allow the Euler step size \( \Delta t \) to be smaller than 1, as a way to make the temporal model closer to a continuous dynamics. The updated dynamics becomes, with \( \frac{1}{\Delta t} \in \mathbb{N} \) to synchronize the step size with the video frame rate:

\[
y_{t+\Delta t} = y_t + \Delta t \cdot f_\theta(y_t, z_{t+1+1})
\]

(2)

For this formulation, the auxiliary variable \( z_t \) is kept constant between two integer time steps. Note that a different \( \Delta t \) can be used during training or testing. This allows our model to generate videos at an arbitrary frame rate since each intermediate latent state can be decoded in the observation space, as well as observe the quality of the dynamic by testing its generalization to the continuous limit. In the following, we consider \( \Delta t \) as a hyperparameter. For the sake of clarity, we consider that \( \Delta t = 1 \) in the following; generalizing to smaller \( \Delta t \) is straightforward as Figure 1a remains unchanged.

3.2 Content Variable

Some components of video sequences can be static, such as the background or shapes of moving objects. They may not impact the dynamics; we therefore model them separately, in the same spirit...
as Denton & Birodkar (2017) and Yingzhen & Mandt (2018). We compute a content variable \( w \) that remains constant throughout the whole generation process and is fed together with \( y_t \) into the frame generator. It enables the dynamical part of the model to focus only on movement, hence being lighter and more stable. Moreover, it allows us to leverage architectural advances in neural networks, such as skip connections (Ronneberger et al., 2015), to produce more realistic frames.

This content variable is a deterministic function \( c_p \) of a fixed number \( k < T \) of frames \( x_c^{(k)} \):

\[
x_c^{(k)} = x_{i_1}, \ldots, x_{i_k}, \quad w = c_p(x_c^{(k)}) = c_p(x_{i_1}, \ldots, x_{i_k}), \quad x_t \sim \mathcal{G}(g_\theta(y_t, w)). \tag{3}
\]

During testing, \( x_c^{(k)} \) are the last \( k \) conditioning frames (usually between 2 and 5). To prevent temporal information from leaking into \( w \), we propose during training to uniformly sample these \( k \) frames within \( x_{1:T} \). Hence, \( c_p \) is designed to be a permutation-invariant function (Zaheer et al., 2017). Similarly to Santoro et al. (2017), this is done by using an MLP fed with the sum of individual frame representations. Note that this content variable can be removed from our model, yielding a more classical deep state-space model. An experiment in this setting is presented in Appendix E.

3.3 Variational Inference and Architecture

Following the generative process depicted in Figure 1a, the conditional joint probability of the full model, given a content variable \( w \), can be written as:

\[
p(x_{1:T}, z_{2:T}, y_{1:T} \mid w) = p(y_1) \prod_{t=1}^{T-1} p(z_{t+1} \mid y_t) p(y_{t+1} \mid y_t, z_{t+1}) \prod_{t=1}^{T} p(x_t \mid y_t, w), \tag{4}
\]

where \( p(y_{t+1} \mid y_t, z_{t+1}) = \delta(y_{t+1} + f_\theta(y_t, z_{t+1}) - y_t) \) and \( \delta \) is the Dirac delta function centered on 0, according to the expression of \( y_{t+1} \) in Equation (1). Thus, in order to optimize the likelihood of the observed videos \( p(x_{1:T} \mid w) \), we need to infer latent variables \( y_t \) and \( z_{2:T} \). This is done by deep Variational Inference using the inference model parameterized by \( \phi \) and shown in Figure 1b, which comes down to consider a variational distribution \( q_{Z,Y} \) defined and factorized as follows:

\[
q_{Z,Y} \triangleq q(z_{2:T}, y_{1:T} \mid x_{1:T}, w) = q(y_1 \mid x_{1:k}) \prod_{t=2}^{T} q(z_t \mid x_{1:t}) \delta(y_{t-1} + f_\theta(y_{t-1}, z_t) - y_t). \tag{5}
\]

This yields the following evidence lower bound (ELBO), whose full derivation is given in Appendix A:

\[
\log p(x_{1:T} \mid w) \geq \mathbb{E}_{(z_{2:T}, y_{1:T}) \sim q_{Z,Y}} \sum_{t=1}^{T} \log p(x_t \mid \tilde{y}_t, w) - D_{KL}(q(y_1 \mid x_{1:k}) \parallel p(y_1)) \\
- \mathbb{E}_{(z_{2:T}, y_{1:T}) \sim q_{Z,Y}} \sum_{t=2}^{T} D_{KL}(q(z_t \mid x_{1:t}) \parallel p(z_t \mid \tilde{y}_t) + f_\theta(y_{t-1}, z_t)) \triangleq \mathcal{L}(x_{1:T}; w, \theta, \phi). \tag{6}
\]

The sum of KL divergence expectations implies to consider the full past sequence of inferred states for each time step, due to the dependence on conditionally deterministic variables \( y_{2:T} \).

However, optimizing \( \mathcal{L}(x_{1:T}; w, \theta, \phi) \) with respect to model parameters \( \theta \) and variational parameters \( \phi \) can be done efficiently by sampling a single full sequence of states from \( q_{Z,Y} \) per example, and computing gradients by backpropagation (Rumelhart et al., 1988) through all inferred variables, using the reparametrization trick (Kingma & Welling, 2014; Rezende et al., 2014). We classically choose \( q(y_1 \mid x_{1:k}) \) and \( q(z_t \mid x_{1:t}) \) to be factorized Gaussian so that all KLDs can be computed analytically.

We include an \( \ell_2 \) regularization term on residues \( f_\theta \) which stabilizes the temporal dynamics of the residual network, as noted by Behrmann et al. (2019) and Rousseau et al. (2019). Given a set of videos \( \mathcal{X} \), the full optimization problem, where \( \mathcal{L} \) is defined as in Equation (6), is then given as:

\[
\arg \max_{\theta, \phi, \psi} \sum_{x_{1:T} \in \mathcal{X}} \left[ \mathbb{E}_{x_c^{(k)} \sim \mathcal{X}} \mathcal{L}
\left(\text{x}_{1:T}; c_p(x_c^{(k)}), \theta, \phi \right) - \lambda \cdot \mathbb{E}_{(z_{2:T}, y_{1:T}) \sim q_{Z,Y}} \sum_{t=2}^{T} \|f_\theta(y_{t-1}, z_t)\|_2 \right]. \tag{7}
\]

Figure 1c depicts the full architecture of our temporal model, corresponding to how the model is applied during testing. The first latent variables are inferred with the conditioning framed and are
then predicted with the dynamic model. In contrast, during training, each frame of the input sequence is considered for inference, which is done as follows. Firstly, each frame $x_t$ is independently encoded into a vector-valued representation $\tilde{x}_t$, with $\tilde{x}_t = h_\phi(x_t)$, $y_t$ is then inferred using an MLP on the first $k$ encoded frames $\tilde{x}_{1:k}$. Each $z_t$ is inferred in a feed-forward fashion with an LSTM on the encoded frames. Inferring $z$ this way experimentally performs better than, e.g., inferring them from the whole sequence $x_{1:T}$; we hypothesize that this follows from the fact that this filtering scheme is closer to the prediction setting, where the future is not available.

4 EXPERIMENTS

This section exposes the experimental results of our method on three standard stochastic video prediction datasets.\(^1\) We compare our method with state-of-the-art baselines on stochastic video prediction. Furthermore, we qualitatively study the dynamics and latent space learned by our model. Training details are described in Appendix C.

The stochastic nature and novelty of the task of stochastic video prediction make it challenging to evaluate (Lee et al., 2018): since videos and models are stochastic, comparing the ground truth and a predicted video is not adequate. We thus adopt the common approach (Denton & Fergus, 2018; Lee et al., 2018) consisting in, for each test sequence, sampling from the tested model a given number (here, 100) of possible futures and reporting the best performing sample against the true video. We report this discrepancy for three commonly used metrics: Peak Signal-to-Noise Ratio (PSNR, higher is better), Structured Similarity (SSIM, higher is better), and Learned Perceptual Image Patch Similarity (LPIPS, lower is better) (Zhang et al., 2018). PSNR tends to promote blurry predictions, as it is a pixel-level measure derived from the $\ell_2$ distance, but greatly penalizes errors in predicted positions of objects in the scenes. SSIM is a similarity metric between image patches. LPIPS is a learned distance between activations of deep CNN trained on image classification tasks, and have been shown to better correlate with human judgment on real images (Zhang et al., 2018).

We present experimental results on a simulated dataset and two real-world datasets, that we briefly present in the following and detail in Appendix B. The corresponding numerical results can be found

\(^1\)Code, video samples, and datasets are available at https://sites.google.com/view/srvp/.
Figure 4: PNSR, SSIM and LPIPS scores with respect to $t$ for all tested models on the KTH dataset.

Figure 5: Conditioning frames and corresponding ground truth, best samples from SVG, SAVP and our method, and worst and random samples from our method, for an example of the KTH dataset. Samples are chosen according to their LPIPS with respect to the ground truth. SVG fails to make a person appear unlike SAVP and our model, and the latter better predicts the pose of the subject.

in Appendix D. For the sake of concision, we only display a handful of qualitative samples in this section, and refer to Appendix H for additional samples. We compare our model against several state-of-the-art models: SV2P (Babaeizadeh et al., 2018), SVG (Denton & Fergus, 2018) and SAVP (Lee et al., 2018). All baseline results were obtained with pretrained models released by the authors. Unless specified otherwise, our model is tested with the same $\Delta t$ as in training (see Equation (2)).

**Stochastic Moving MNIST (SM-MNIST).** This dataset consists of one or two MNIST digits (LeCun et al., 1998) moving linearly and randomly bouncing on walls with new direction and velocity sampled randomly at each bounce (Denton & Fergus, 2018). As SV2P and SAVP were not tested on this dataset (in particular, with no pretrain model, code or hyperparameters), we only report scores for SVG as state-of-the-art model on SM-MNIST.

Figure 2a shows quantitative results with two digits. Our model outperforms SVG on both PSNR and SSIM; LPIPS is not reported as it is not relevant for this synthetic task. Decoupling dynamics from image synthesis allows our method to maintain temporal consistency despite high-uncertainty frames where crossing digits become indistinguishable. For instance in Figure 3, the digits shape changes after they cross in the SVG prediction, while our model predicts the correct digits. To evaluate the predictive ability on a longer horizon, we perform experiments on the classic deterministic version of the dataset (Srivastava et al., 2015). We show the results up to $t + 95$ in Figure 2b. We can see that our model better captures the dynamics of the problem compared to SVG as its performance decreases significantly less, even at a long-term horizon.

We also compare to two alternative versions of our model in Figure 2, where the residual dynamic function is replaced by an MLP or a GRU network (Cho et al., 2014). Our residual model outperforms both versions on the stochastic, and especially on the deterministic version of the dataset, showing its
intrinsic advantage at modeling dynamics. Finally, on the deterministic version of Moving MNIST, we compare to an alternative where \( z \) is entirely removed, resulting in a temporal model very close to the one presented in Chen et al. (2018). The loss of performance of this alternative model is significant, especially in SSIM, showing that our stochastic residual model offers a substantial advantage even when used in a deterministic environment.

KTH Action dataset (KTH). This dataset is composed of real-world videos of people performing a single action per video in front of different backgrounds (Schüldt et al., 2004). Uncertainty lies in the appearance of subjects, the actions they perform and how they are performed.

We outperform on this dataset every considered baseline on each metric, as depicted in Figure 6. In some videos, the subject only appears after the conditioning frames, requiring the model to sample the moment and location of the subject appearance, as well as its action. This critical case is illustrated in Figure 5. There, SVG fails to even generate a moving person; only SAVP and our model manage to do so, and our best sample is closer to the subject’s poses compared to SAVP. Moreover, the worst and a random sample of our model demonstrate that it captures the diversity of the dataset by making a person appear at different time steps and with different speeds. An additional experiment on this dataset is included in Appendix G, studying the influence of the encoder and decoder architecture on SVG and our model.

BAIR robot pushing dataset (BAIR). This dataset contains videos of a Sawyer robotic arm pushing objects on a tabletop (Ebert et al., 2017). It is highly stochastic as the arm can change its direction at any moment. We achieve similar or better results compared to state-of-the-art models, as Figure 6 shows, and second-best PSNR behind SV2P, but the latter produces very blurry samples, which can be seen in Appendix H, yielding a prohibitive LPIPS score. In contrast, we achieve the highest SSIM as well as competitive LPIPS among these models. Note that we could not add VideoFlow to our experiments due to the unavailability of pretrained models and numerical results. However, compared to results reported by Kumar et al. (2019) for BAIR (the only tested dataset in their paper), our model appears to behave better than VideoFlow, which is on par with SAVP.

Varying frame rate in testing. We challenge the ability of our model to use a different Euler step size than the one used in training (see Equation (2)). Figures 4 and 6 include corresponding results with a halved \( \Delta t \). Prediction performances remain stable while generating twice as many frames (cf. Appendix F for further discussion). Our model is thus robust to the refinement of the Euler
Figure 8: Video (bottom right) generated from the dynamic latent state $y$ inferred with a video (top) and the content variable $w$ computed with the conditioning frames of another video (bottom left). The generated video keeps the same background as the bottom left frames, while the robotic arm moves accordingly to the top frames.

Figure 9: From left to right, $x^s$, $\hat{x}^s$ (reconstruction of $x^s$ by the VAE of our model), results of the interpolation in the latent space between $x^s$ and $x^t$, $\hat{x}^t$ and $\hat{x}^t$. Each trajectory is materialized in shades of grey in the frames.

approximation, showing the quality of the learned dynamic which is close to continuous. This can be used to generate frames at a higher frame rate than the training videos without supervision. We show in Figure 7 and Appendix F frames generated at a double and quadruple frame rate on BAIR and KTH. Both figures show smooth intermediate generated frames.

**Disentangling dynamics and content.** Let us show that the proposed model actually separates content from dynamics as discussed in Section 3.2. To this end, two sequences $x^s$ and $x^t$ are drawn from the BAIR test set. While $x^s$ is used for extracting our content variable $w^s$, dynamic states $y^s$ are inferred with our model from $x^s$. New frame sequences $\hat{x}$ are finally generated from the fusion of the content vector and the dynamics. This results in a content corresponding to the first sequence $x^s$ while moving according to the dynamics of the second sequence $x^t$, as observed in Figure 8. More samples for BAIR and KTH can be seen in Appendix H.

**Interpolation of dynamics.** Our state-space structure allows us to learn semantic representations in $y_t$. To highlight this feature, we test whether two Moving MNIST trajectories can be interpolated by linearly interpolating their inferred latent initial conditions. We begin by generating two trajectories $x^s$ and $x^t$ of a single moving digit. We infer their respective latent initial conditions $y_{t1}^s$ and $y_{t1}^t$. We then use our model to generate frame sequences from latent initial conditions linearly interpolated between $y_{t1}^s$ and $y_{t1}^t$. If it learned a meaningful latent space, the resulting trajectories should also be smooth interpolations between the directions of reference trajectories $x^s$ and $x^t$, and this is what we observe in Figure 9. Additional examples can be found in Appendix H.

**5 Conclusion**

We introduce a novel dynamic latent model for stochastic video prediction which, unlike prior image-autoregressive models, decouples frame synthesis and dynamics. This temporal model is based on residual updates of a small latent state that is showed to perform better than RNN-based models. This endows our method with several desirable properties, such as temporal efficiency and latent space interpretability. We experimentally demonstrate the performance and advantages of the proposed model, which outperforms prior state-of-the-art methods for stochastic video prediction. This work is, to the best of our knowledge, the first to propose a latent dynamic model scaling for video prediction. The proposed model is also novel with respect to the recent line of work dealing with neural networks and ODEs for temporal modeling; it is the first such residual model to scale to complex stochastic data such as videos.
REFERENCES


A EVIDENCE LOWER BOUND

We develop in this section the computations of the variational lower bound for the proposed model.

Using the original variational lower bound of Kingma & Welling (2014) in Equation (8):

\[
\log p(x_{1:T} \mid w) \geq \mathbb{E}_{(\tilde{z}_{2:T}, \tilde{y}_{1:T}) \sim q_{2:T,Y}} \log p(x_{1:T} \mid \tilde{z}_{2:T}, \tilde{y}_{1:T}, w) - D_{KL}(q_{2:T,Y} \parallel p(y_{1:T}, z_{2:T} \mid w)) \tag{8}
\]

\[
= \mathbb{E}_{(\tilde{z}_{2:T}, \tilde{y}_{1:T}) \sim q_{2:T,Y}} \log p(x_{1:T} \mid \tilde{z}_{2:T}, \tilde{y}_{1:T}, w) - D_{KL}(q(y_{1:T}, z_{2:T} \mid x_{1:T}) \parallel p(y_{1:T}, z_{2:T})) \tag{9}
\]

\[
= \mathbb{E}_{(\tilde{z}_{2:T}, \tilde{y}_{1:T}) \sim q_{2:T,Y}} \sum_{i=1}^{T} \log p(x_i \mid \tilde{y}_i, w) - D_{KL}(q(y_{1:T}, z_{2:T} \mid x_{1:T}) \parallel p(y_{1:T}, z_{2:T})) \tag{10}
\]

where:

- Equation (9) is given by the forward and inference models factorizing \(p\) and \(q\) in Equations (4) and (5) and illustrated by, respectively, Figures 1a and 1b:
  - the \(z\) variables and \(y_1\) are independent from \(w\) to \(p\) and \(q\);
  - the \(y_{2:T}\) variables are deterministic functions of \(y_1\) and \(z_{2:T}\) with respect to \(p\) and \(q\);

- Equation (10) results from the factorization of \(p(x_{1:T} \mid y_{1:T}, z_{1:T}, w)\) in Equation (4).

From there, by using the integral formulation of \(D_{KL}\):

\[
\log p(x_{1:T} \mid w) \geq \mathbb{E}_{(\tilde{z}_{2:T}, \tilde{y}_{1:T}) \sim q_{2:T,Y}} \sum_{i=1}^{T} \log p(x_i \mid \tilde{y}_i, w) \tag{11}
\]

\[
+ \int \cdots \int_{y_1, z_{2:T}} q(y_{1:T}, z_{2:T} \mid x_{1:T}) \log \frac{p(y_{1:T}, z_{2:T})}{q(y_{1:T}, z_{2:T} \mid x_{1:T})} \, dz_{2:T} \, dy_1
\]
where:

- Equation (13) follows from the inference model of Equation (5), where $y_1$ only depends on $x_{1:k}$.
- Equation (14) is obtained from the factorizations of Equations (4) and (5).

By iterating Equation (15)'s step on $z_3, \ldots, z_T$ and factorizing all expectations, we obtain:

\[
\log p(x_{1:T} \mid w) \\
\geq \mathbb{E}(\tilde{z}_{1:T}, \tilde{y}_{1:T}) - q_{z,y} \sum_{t=1}^{T} \log p(x_t \mid \tilde{y}_t, w) - D_{KL}(q(y_1 \mid x_{1:k}) \parallel p(y_1)) \\
- \mathbb{E} \int \prod_{t=2}^{T} q(z_t \mid x_{1:t}) \sum_{t=2}^{T} D_{KL}(q(z_t \mid x_{1:t}) \parallel p(z_t \mid \tilde{y}_1, \tilde{z}_{1:t-1})),
\]

and we finally retrieve Equation (6) by using the factorization of Equation (5):

\[
\log p(x_{1:T} \mid w) \\
\geq \mathbb{E}(\tilde{z}_{1:T}, \tilde{y}_{1:T}) - q_{z,y} \sum_{t=1}^{T} \log p(x_t \mid \tilde{y}_t, w) - D_{KL}(q(y_1 \mid x_{1:k}) \parallel p(y_1)) \\
- \sum_{t=2}^{T} D_{KL}(q(z_t \mid x_{1:t}) \parallel p(z_t \mid \tilde{y}_t-1)).
\]
B.1 Stochastic Moving MNIST (SM-MNIST)

This dataset consists in one or two train MNIST digits (LeCun et al., 1998) of size 27 × 27 moving linearly within a 64 × 64 frame and randomly bounce against its border, sampling a new direction and velocity at each bounce (Denton & Fergus, 2018). We use the same settings as Denton & Fergus (2018), train all models on 15 timesteps and condition them at test time on 5 frames. Note that we adapted the dataset to sample more coherent bounces: the original dataset computes digit trajectories that are dependent on the chosen framerate, unlike our corrected version of the dataset. We consequently retrained SVG on this dataset, obtaining comparable results as those originally presented by Denton & Fergus (2018). Test data were produced by generating 5000 samples with a different digit for each sequence coming from the MNIST test set.

B.2 KTH Action Dataset (KTH)

This dataset is composed of real-world 64 × 64 videos of 25 people performing one of six actions (walking, jogging, running, boxing, handwaving and handclapping) in front of different backgrounds (Schüldt et al., 2004). Uncertainty lies in the appearance of subjects, the action they perform and how it is performed. The training set is formed with actions from 20 people, the remaining five being used for testing. Training is performed by sampling sub-sequences of size 20 in the train set. The test set is composed of 1000 randomly sampled sub-sequences of size 40.

B.3 BAIR Robot Pushing Dataset (BAIR)

This dataset contains 64 × 64 videos of a Sawyer robotic arm pushing objects on a table top (Ebert et al., 2017). It is highly stochastic as the arm can change its direction at any moment. Training is performed on 12 frames and testing is done with two conditioning frames on the provided test set, consisting of 256 sequences of 30 frames.

C.1 Specifications

We used Python 3.7.4 and PyTorch 1.2.0 (Paszke et al., 2017) to implement our model. Each model was trained on a Nvidia GPUs with CUDA 10 in mixed-precision training with the help of Apex.²

C.2 Architecture

**Encoder and decoder architecture.** Both $g_\theta$ and $h_\phi$ are chosen to have different architectures depending on the dataset. We choose to use a DCGAN discriminator and generator architecture (Radford et al., 2016) for Moving MNIST, and a VGG16 (Simonyan & Zisserman, 2015) architecture (mirrored for $h_\phi$) for BAIR and KTH. In both cases, the output of $h_\phi$ (i.e., $\tilde{x}$) is a vector of size 128, and $g_\theta$ and $h_\phi$ weights are initialized using a centered normal distribution with a standard deviation of 0.02.

For the Moving MNIST dataset, the content variable $w$ is obtained directly from $\tilde{x}$ and is thus a vector of size 128. For KTH and BAIR, we supplement this vectorial variable with skip connections from all layers of the encoder $g_\theta$ that are then fed to the decoder $h_\phi$ to handle complex backgrounds. For Moving MNIST, the number of frames $k$ used to compute the content variable is 5; for KTH, it is 3; for BAIR, it is 2.

**LSTM architecture.** The LSTM used for all datasets has a single layer of LSTM cells with a hidden state size of 256.

MLP architecture. All MLPs used in inference (with parameters $\phi$) have three linear layers with hidden size 256 and leaky ReLU activations. All MLPs used in the forward (with parameters $\theta$) have four linear layers with hidden size 512 and leaky ReLU activations.

Sizes of latent variables. The sizes of the latent variables in our model are the following: for Moving MNIST, $y$ and $z$ have size 20; for KTH abs BAIR, $y$ and $z$ have size 50.

Euler step size All models but those trained on KTH are trained with $\Delta t = 1$. Models on KTH are trained with $\Delta t = \frac{1}{2}$.

C.3 Optimization

Loss function. All models are trained using the Adam optimizer (Kingma & Ba, 2015) with learning rate $3 \times 10^{-4}$ and $\lambda = 1$. The batch size for Moving MNIST and BAIR is chosen to be 128, and the batch size for KTH is chosen to be 100.

Following (Higgins et al., 2017), we use $\beta = 1$ (cf. Equation (7)), except for the Moving MNIST dataset where the $\beta$ factor in front of the KL on $z$ (last term of Equation (6)) is equal to 2.

Variance of the observation. The variance $\nu$ used in the observation probability distribution $G(\theta(y)) = \mathcal{N}(\theta(y), \nu I)$ is chosen as follows:

- for Moving MNIST, $\nu = 1$;
- for KTH, $\nu = 4 \times 10^{-2}$;
- for BAIR, $\nu = \frac{1}{2}$.

Number of optimization steps. The number of optimization steps is the following for the different datasets:

- Moving MNIST (stochastic): 1 000 000 steps with additional 100 000 steps where the learning rated is linearly decreased to 0;
- Moving MNIST (deterministic): 800 000 steps with additional 100 000 steps where the learning rated is linearly decreased to 0;
- KTH: 200 000 steps, the final model being chosen among several checkpoints as the one having the best evaluation score (which differs from the test score as we extract from the train set an evaluation set);
- BAIR: 250 000 steps, the final model is chosen as for KTH.

D Additional Numerical Results

Tables 1 to 3 present, respectively, numerical results averaged over all time steps for our methods and considered baselines on the SM-MNIST, KTH and BAIR datasets, corresponding to Figures 2, 4 and 6.

E Pendulum Experiments

We test the ability of our model to model the dynamics of a common dataset used in the literature of state-space models (Karl et al., 2017; Fraccaro et al., 2017), Pendulum (Karl et al., 2017). It consists of noisy observations of a dynamic torque-controlled pendulum; it is stochastic as the information of this control is not available. We test our model, without the content variable $w$, in the same setting as DVBF (Karl et al., 2017) and KVAE (Fraccaro et al., 2017) and report the corresponding ELBO scores in Table 4. The encoders and decoders for all methods are MLPs.

Our model outperforms DVBF and is merely beaten by KVAE. This can be explained by the nature of the KVAE model, whose sequential model is not learned using a VAE but a Kalman filter allowing exact inference in the latent space. On the contrary, DVBF is learned, like our model, by a sequential
Table 1: Numerical results (mean and 95% confidence interval) for PSNR and SSIM for tested methods on the two-digits Moving MMNIST dataset. Bold scores indicate the best performing method and, where appropriate, scores whose means lie in the confidence interval of the best performing method.

<table>
<thead>
<tr>
<th>Models</th>
<th>Stochastic</th>
<th>Deterministic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PSNR</td>
<td>SSIM</td>
</tr>
<tr>
<td>SVG</td>
<td>14.45 ± 0.06</td>
<td>0.7070 ± 0.0021</td>
</tr>
<tr>
<td>Ours - GRU</td>
<td>15.81 ± 0.08</td>
<td>0.7463 ± 0.0023</td>
</tr>
<tr>
<td>Ours - MLP</td>
<td>16.55 ± 0.09</td>
<td>0.7693 ± 0.0024</td>
</tr>
<tr>
<td>Ours</td>
<td>16.90 ± 0.09</td>
<td>0.7789 ± 0.0025</td>
</tr>
</tbody>
</table>

Table 2: Numerical results (mean and 95% confidence interval) for PSNR, SSIM, and LPIPS for tested methods on the KTH dataset. Bold scores indicate the best performing method and, where appropriate, scores whose means lie in the confidence interval of the best performing method.

<table>
<thead>
<tr>
<th>Models</th>
<th>PSNR</th>
<th>SSIM</th>
<th>LPIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SV2P</td>
<td>28.18 ± 0.39</td>
<td>0.8141 ± 0.0068</td>
<td>0.2049 ± 0.0080</td>
</tr>
<tr>
<td>SAVP</td>
<td>26.51 ± 0.36</td>
<td>0.7560 ± 0.0083</td>
<td>0.1120 ± 0.0058</td>
</tr>
<tr>
<td>SVG</td>
<td>26.99 ± 0.33</td>
<td>0.8291 ± 0.0074</td>
<td>0.1083 ± 0.0058</td>
</tr>
<tr>
<td>Ours</td>
<td>29.69 ± 0.37</td>
<td>0.8697 ± 0.0057</td>
<td>0.0736 ± 0.0036</td>
</tr>
<tr>
<td>Ours - ∆t²</td>
<td>29.43 ± 0.38</td>
<td>0.8633 ± 0.0061</td>
<td>0.0790 ± 0.0042</td>
</tr>
</tbody>
</table>

Table 3: Numerical results (mean and 95% confidence interval) with respect to PSNR, SSIM, and LPIPS for tested methods on the BAIR dataset. Bold scores indicate the best performing method and, where appropriate, scores whose means lie in the confidence interval of the best performing method.

<table>
<thead>
<tr>
<th>Models</th>
<th>PSNR</th>
<th>SSIM</th>
<th>LPIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SV2P</td>
<td>20.39 ± 0.42</td>
<td>0.8169 ± 0.0110</td>
<td>0.0912 ± 0.0063</td>
</tr>
<tr>
<td>SAVP</td>
<td>18.44 ± 0.40</td>
<td>0.7886 ± 0.0117</td>
<td>0.0634 ± 0.0048</td>
</tr>
<tr>
<td>SVG</td>
<td>18.95 ± 0.41</td>
<td>0.8057 ± 0.0116</td>
<td>0.0609 ± 0.0046</td>
</tr>
<tr>
<td>Ours</td>
<td>19.64 ± 0.45</td>
<td>0.8211 ± 0.0110</td>
<td>0.0610 ± 0.0048</td>
</tr>
<tr>
<td>Ours - ∆t²</td>
<td>19.77 ± 0.44</td>
<td>0.8235 ± 0.0110</td>
<td>0.0597 ± 0.0047</td>
</tr>
</tbody>
</table>

Table 4: ELBO score for DVBF, KVAE and our model on the Pendulum dataset. The bold score indicates the best performing method.

<table>
<thead>
<tr>
<th>Score</th>
<th>DVBF</th>
<th>KVAE</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELBO</td>
<td>798.56</td>
<td><strong>807.02</strong></td>
<td>806.12</td>
</tr>
</tbody>
</table>
Table 5: Numerical results for BAIR of our model trained with $\Delta t = 1$ and tested with different values of $\Delta t$.

<table>
<thead>
<tr>
<th>Step size $\Delta t$</th>
<th>PSNR</th>
<th>SSIM</th>
<th>LPIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta t = 1$</td>
<td>19.64 ± 0.45</td>
<td>0.8210 ± 0.0110</td>
<td>0.0612 ± 0.0048</td>
</tr>
<tr>
<td>$\Delta t = \frac{1}{2}$</td>
<td>19.76 ± 0.44</td>
<td>0.8235 ± 0.0110</td>
<td>0.0597 ± 0.0047</td>
</tr>
<tr>
<td>$\Delta t = \frac{1}{3}$</td>
<td>19.82 ± 0.45</td>
<td>0.8245 ± 0.0111</td>
<td>0.0593 ± 0.0048</td>
</tr>
<tr>
<td>$\Delta t = \frac{1}{4}$</td>
<td>19.83 ± 0.46</td>
<td>0.8242 ± 0.0111</td>
<td>0.0593 ± 0.0049</td>
</tr>
<tr>
<td>$\Delta t = \frac{1}{5}$</td>
<td>19.85 ± 0.46</td>
<td>0.8243 ± 0.0111</td>
<td>0.0591 ± 0.0048</td>
</tr>
</tbody>
</table>

Table 6: Numerical results for KTH of our model trained with $\Delta t = 1$ and tested with different values of $\Delta t$.

<table>
<thead>
<tr>
<th>Step size $\Delta t$</th>
<th>PSNR</th>
<th>SSIM</th>
<th>LPIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta t = 1$</td>
<td>29.76 ± 0.38</td>
<td>0.8681 ± 0.0057</td>
<td>0.0737 ± 0.0057</td>
</tr>
<tr>
<td>$\Delta t = \frac{1}{2}$</td>
<td>29.05 ± 0.42</td>
<td>0.8539 ± 0.0066</td>
<td>0.0882 ± 0.0050</td>
</tr>
<tr>
<td>$\Delta t = \frac{1}{3}$</td>
<td>29.05 ± 0.42</td>
<td>0.8509 ± 0.0069</td>
<td>0.0924 ± 0.0055</td>
</tr>
<tr>
<td>$\Delta t = \frac{1}{4}$</td>
<td>28.98 ± 0.42</td>
<td>0.8496 ± 0.0069</td>
<td>0.0939 ± 0.0056</td>
</tr>
<tr>
<td>$\Delta t = \frac{1}{5}$</td>
<td>28.95 ± 0.42</td>
<td>0.8490 ± 0.0070</td>
<td>0.0948 ± 0.0057</td>
</tr>
</tbody>
</table>

VAE, and is thus much closer to our model than KVAE. This result then shows that the dynamic model that we chose in the context of sequential VAEs is more adapted on this dataset than the one of DVBF, and achieve results close to a method taking advantage of exact inference using adapted tools such as Kalman filters.

F INFLUENCE OF THE EULER STEP SIZE

Table 5 details the numerical results of our model trained on BAIR with $\Delta t = 1$ and tested with different values of $\Delta t$. It shows that, when refining the Euler approximation, our model can improve its performance in a setting that is unseen during training. Results stabilize when $\Delta t$ is small enough, showing that the model is close to the continuous limit.

Tables 6 and 7 details the numerical results of our model trained on KTH with, respectively, $\Delta t = 1$ and $\Delta t = \frac{1}{2}$, and tested with different values of $\Delta t$. They show that if $\Delta t$ is chosen too high when training (here, $\Delta t = 1$), the model drops in performance when refining the Euler approximation. We assume that this phenomenon arises because the Euler approximation used in training is too rough, making the model adapt to a very discretized dynamic that cannot be transferred to smaller Euler step sizes. Indeed, when training with smaller step size, (here, $\Delta t = \frac{1}{2}$), results in the training settings are equivalent while results obtained with a lower $\Delta t$ are now much closer, if not equivalent, to the nominal ones. This shows that the model learned a continuous dynamic if learned with a small enough step size.

Table 7: Numerical results for KTH of our model trained with $\Delta t = \frac{1}{2}$ and tested with different values of $\Delta t$.

<table>
<thead>
<tr>
<th>Step size $\Delta t$</th>
<th>PSNR</th>
<th>SSIM</th>
<th>LPIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta t = 1$</td>
<td>28.80 ± 0.41</td>
<td>0.8495 ± 0.0068</td>
<td>0.0994 ± 0.0057</td>
</tr>
<tr>
<td>$\Delta t = \frac{1}{2}$</td>
<td>29.69 ± 0.37</td>
<td>0.8697 ± 0.0057</td>
<td>0.0736 ± 0.0036</td>
</tr>
<tr>
<td>$\Delta t = \frac{1}{3}$</td>
<td>29.52 ± 0.38</td>
<td>0.8656 ± 0.0059</td>
<td>0.0777 ± 0.0041</td>
</tr>
<tr>
<td>$\Delta t = \frac{1}{4}$</td>
<td>29.43 ± 0.39</td>
<td>0.8633 ± 0.0061</td>
<td>0.0790 ± 0.0042</td>
</tr>
<tr>
<td>$\Delta t = \frac{1}{5}$</td>
<td>29.35 ± 0.39</td>
<td>0.8615 ± 0.0062</td>
<td>0.0810 ± 0.0045</td>
</tr>
</tbody>
</table>
Figure 10: PSNR, SSIM and LPIPS scores with respect to $t$ on the KTH dataset for SVG and our model with two choices of encoder and decoder architecture for each: DCGAN and VGG.

Note that the loss of performance using a higher $\Delta t$ in testing than in training, like in Table 7, is expected as it corresponds to loosening the Euler approximation compared to training. However, even in this adversarial setting, our model maintains state-of-the-art results, demonstrating the quality of the learned dynamic as it can be further discretized if needed at the cost of a reasonable drop in performance.

G AUTOREGRESSIVITY AND IMPACT OF ENCODER AND DECODER ARCHITECTURE

Figure 10 exposes the numerical results on KTH of our model and SVG for different choices of architectures: DCGAN and VGG.

Since DCGAN is a less powerful architecture than VGG, results of each method with VGG are expectedly better than those of the same method with DCGAN. Moreover, our model outperforms SVG for any fixed choice of encoder and decoder architecture, which is coherent with Figure 4.

We observe, however, that the difference between a method using VGG and its DCGAN counterpart differs depending on the model. Ours shows more robustness to changing of encoder and decoder architecture, as it loses much less performance than SVG when switching to a less powerful architecture. Indeed, while the difference in LPIPS is similar for both models (as expected from a score evaluating the realism of produced frames), the loss of SVG is significantly larger than our loss in terms of SSIM, and in particular PNSR. This shows that reducing the capacity of the encoders and decoders of SVG not only hurts its ability to produce realistic frames as expected but also substantially lowers its ability to learn a good dynamic. We assume that this phenomenon is caused by the autoregressive nature of SVG, which makes it reliant of the performance of its encoders and decoders. This supports our motivation to propose a non-autoregressive model for stochastic video prediction.

H ADDITIONAL SAMPLES

This section includes some additional samples corresponding to experiments described in Section 4.

H.1 STOCHASTIC MOVING MNIST

We present in Figures 11 to 14 additional samples from SVG and our model on SM-MNIST.

In particular, Figure 13 shows SVG changing a digit shape in the course of a prediction even though it does not cross another digit, whereas ours maintain the digit shape. We assume that this advantage of ours comes from the latent nature of the dynamic of our model and the use in our of a static content variable that is prevented from containing temporal information. Indeed, even when the best sample from our model is not close from the ground truth of the dataset, like in Figure 14, the shapes of the digits are still maintained by our model.
Figure 11: Conditioning frames and corresponding ground truth and best samples with respect to PSNR from SVG and our method, and worst and random samples from our method, for an example of the SM-MNIST dataset.

Figure 12: Additional samples for the SM-MNIST dataset (cf. Figure 11).

Figure 13: Additional samples for the SM-MNIST dataset (cf. Figure 11). SVG fails to maintain the shape of a digit, while ours is temporally coherent.
Figure 14: Additional samples for the SM-MNIST dataset (cf. Figure 11). This example was chosen in the worst 1% test examples of our model with respect to PSNR. Despite this adversarial criterion, our model maintains temporal consistency as digits are not deformed in the course of the video.

Figure 15: Conditioning frames and corresponding ground truth, best samples from SVG, SAVP and our method, and worst and random samples from our method, for an example of the KTH dataset. Samples are chosen according to their LPIPS with respect to the ground truth. On this specific task (clapping), all methods but SV2P (which produce blurry predictions) perform well, even though ours stays closer to the ground truth.

Figure 16: Additional samples for the KTH dataset (cf. Figure 15). In this example, the shadow of the subject is visible in the last conditioning frames, foreshadowing its appearance. This is a failure case for SVG and SAVP which only produce an indistinct shadow, whereas SAVP and our model make the subject appear. Yet, SAVP produces the wrong action and an inconsistent subject in its best sample, while ours is correct.
Figure 17: Additional samples for the KTH dataset (cf. Figure 15). This example is a failure case for each method: SV2P produces blurry frames, SVG and SA VP are not consistent (change of action or subject appearance in the video), and our model produces a ghost image at the end of the prediction on the worst sample only.

Figure 18: Additional samples for the KTH dataset (cf. Figure 15). Our model is the only one to make a subject appear in the ground truth.

Figure 19: Additional samples for the KTH dataset (cf. Figure 15). The subject in this example is boxing, which is the most challenging action in the dataset as all methods are far from the ground truth.
Figure 20: Conditioning frames and corresponding ground truth, best samples from SVG, SAVP and our method, and worst and random samples from our method, for an example of the BAIR dataset. Samples are chosen according to their LPIPS with respect to the ground truth.

Figure 21: Additional samples for the KTH dataset (cf. Figure 20).

H.2 KTH

We present in Figures 15 to 19 additional samples from SV2P, SVG, SAVP and our model on KTH, with additional insights.

H.3 BAIR

We present in Figures 20 to 22 additional samples from SV2P, SVG, SAVP and our model on BAIR, with additional insights.

H.4 OVERSAMPLING

We present in Figure 23 additional examples of video generation at a doubled frame rate by our model.

H.5 CONTENT SWAP

We present in Figures 24 to 28 additional examples of content swap as in Figure 8.
Figure 22: Additional samples for the KTH dataset (cf. Figure 20).

(a) KTH sample, centered on the subject.
(b) BAIR sample, centered on the robot arm.

Figure 23: Generation examples at doubled frame rate, using a halved $\Delta t$ compared to training. Frames including a bottom red dashed bar are intermediate frames.

Figure 24: Video (bottom right) generated from the combination of dynamic variables ($y$, $z$) inferred with a video (top) and the content variable ($w$) computed with the conditioning frames of another video (bottom left).

Figure 25: Additional example of content swap (cf. Figure 24).

Figure 26: Additional example of content swap (cf. Figure 24). In this example, the extracted content is the video background, which is successfully transferred to the target video.
Figure 27: Additional example of content swap (cf. Figure 24). In this example, the extracted content is the video background and the subject appearance, which are successfully transferred to the target video.

Figure 28: Additional example of content swap (cf. Figure 24). This example shows a failure case of content swapping.

**H.6 Interpolation in the Latent Space**

Figure 29: From left to right, $x^n$, $\hat{x}^n$ (reconstruction of $x^n$ by the VAE of our model), results of the interpolation in the latent space between $x^n$ and $x^i$, $\hat{x}^i$ and $x^i$. Each trajectory is materialized in shades of grey in the frames.

Figure 30: Additional example of interpolation in the latent space between two trajectories (cf. Figure 29).

We present in Figures 29 and 30 additional examples of interpolation in the latent space between two trajectories.