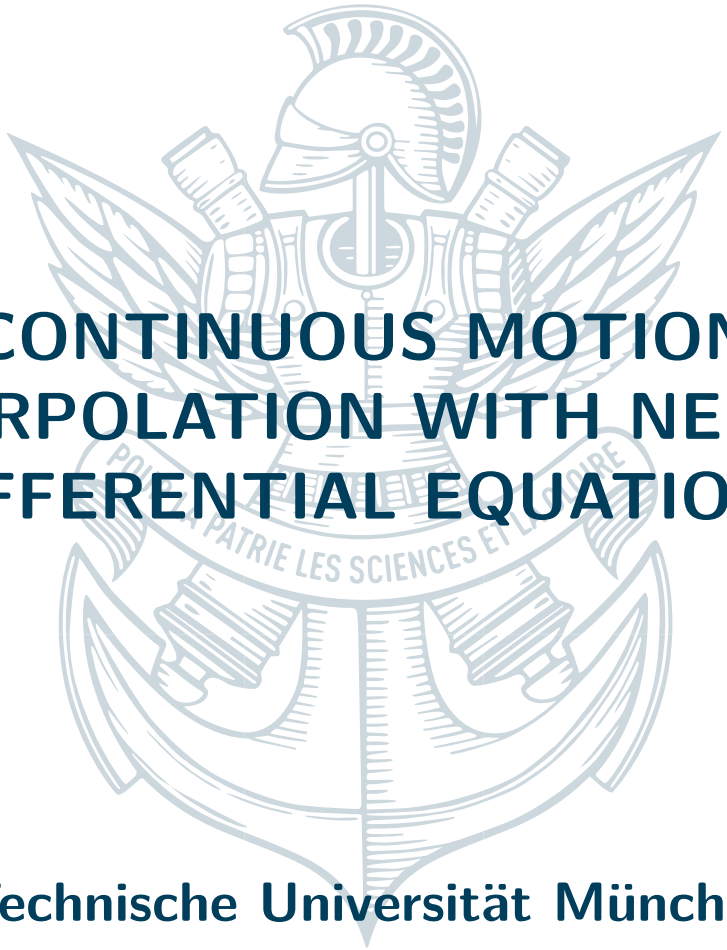

CONTINUOUS MOTION INTERPOLATION WITH NEURAL DIFFERENTIAL EQUATIONS

Technische Universität München

March 2022 - August 2022

Maxime BONNIN X2019

ÉCOLE POLYTECHNIQUE

IP PARIS

CONTENTS

1 ACKNOWLEDGEMENT

The internship opportunity I had with the **Computer Aided Medical Procedures Augmented Reality** team at the **Technische Universität München** was a great chance to learn and to be exposed to public research. Therefore, I consider myself a very fortunate individual for having had the opportunity to be a part of it. I am also grateful for having had the chance to meet such wonderful people who led me through this internship period.

With that in mind, I would like to take this opportunity to express my deepest gratitude and special thanks to **Dr. Benjamin Busam**, who took the time to hear, guide, and keep me on the correct path, allowing me to carry out my project.

It is my sincere sentiment to place on record my best regards and deepest sense of gratitude to **Mr. Konstantinos Zacharis** for his careful and valuable guidance, which helped me greatly in my studies, both theoretically and practically.

I view this opportunity as a major milestone in my career development, and I will strive to use the skills and knowledge I gained in the best possible way.

Sincerely,

Maxime Bonnin

2 SUMMARY

Learning the physics from images is an appealing way to improve unsupervised video prediction method. Knowing the underlying physics is not enough to describe images, we define a deep neural network based on convolutions and neural ordinary differential equations that disentangles the dynamics from the image content and learns the underlying dynamics. This is the first ODE based framework to work on the disentanglement and allows appearance modification without altering the dynamics. Besides, we propose a model that is continuous-time that can be also used for image interpolation.

3 INTRODUCTION

Frame generation is a computer vision task in which either you interpolate or extrapolate frames. Video interpolation consists in predicting one or multiple frames in between given frames, whereas video extrapolation is the task of forecasting the future frames of a video given the previous ones. We decided to focus on the latter for its potential use in many different contexts such as weather forecasting [26, 27], medical purpose [1, 21], action recognition [36] or autonomous driving [32]. In addition, we pay attention to long term forecasting as it can be very beneficial for these fields. Our project is located in the unsupervised video prediction because it is a more realistic setup, especially in the medical or climate fields where semantic labels doesn't exist. The goal is to design a model that will learn the underlying dynamics from raw data. State-of-the-art methods rely on deep learning methods, such as architecture using 2D or 3D convolutions [15, 5], recurrent neural networks [35, 34], or neural ordinary differential equations (ODE) [27, 38, 6]. This associated to adversarial training [15, 5, 32, 30, 27], stochastic models [16, 14], consistency losses [27, 12, 32, 11, 17], constraint predictions by using geometric knowledge [4, 28] or by disentangling factors of variation [10, 13, 25, 22, 17] help to improve the predictions.

One way to model the video dynamics is to learn the underlying physical dynamics that could be defined by partial differential equations (PDE) [7, 24] but by using images the underlying physics are mixed with the appearance parameters. One appealing method consists in disentangling [10, 13, 25, 22, 17] the appearance and the dynamics to not only improve the predictions but also increase the long term consistency of the predicted images. Once disentangled, learning the dynamics is possible thanks to neural ordinary equations (Neural ODE) [23, 9], which is the generalisation of residual network that performs well on the interpolation and extrapolation tasks. However, residual networks lack of time flexibility and because Neural ODE based models generalise them, they outperform the state-of-the-art in addition to give a lot more flexibility in terms of time steps. Besides, this is an effective model for long-term predictions and useful to predict complex phenomena. These models can be trained with regular or irregular time steps and being used with regular or irregular time steps. Most of the models use regular time steps in input and output because they focus on video data, but they're not able to modify the frame rate depending on the video content. Besides, this time flexibility is important when the data aren't regularly capture, e.g. climate, medical data or simply dataset with missing values. Thus, models without time flexibility lead to suboptimal solutions on those datasets.

In this work, a neural ODE based model with the disentanglement of the appearance and the dynamics. The dynamics are not learnable directly from the pixel space, we need to create a latent space H in which the physical laws are learnable by a Neural ODE. We work mostly on generated dataset such as the Moving MNIST MovingMNIST with one or two digits or other custom datasets, e.g the gaussian ball dataset in which we can change the underlying dynamic equation.

We summarise our contributions as follows :

- To the best of our knowledge, this is the first ODE based framework to work on the disentanglement and allows the modification of appearance without modifying the dynamics.
- We provide a continuous-time model that can deal with irregular time steps.
- Even if we focus on extrapolation, our model can be used as well for interpolation.
- We use the spatial encoding in order to have better spatial correlation and therefore better dynamics inside the latent space $\mathcal{H}$.

4 RELATED WORK

Deep neural networks is the state-of-the-art method for video forecasting. Recurrent [35, 34] or convolutional [15, 5] networks achieved great results for this task. Other works investigate the Generative Adversarial Networks (GANs) [32, 30] in order to get sharper predictions, leading the state-of-the-art to whether adopt adversarial loss [15, 5, 27] to sharpen the outputs or create additional losses [27, 12, 32, 11, 17] that produce sharper results. In fact, using only a reconstruction loss (L_1 or L_2) leads to blurry outputs [6] [29, 38]. Some works also investigate Neural ODE based framework, e.g. ODE^2VAE [38] attempted to decompose the latent representations into the position and the momentum to generate low-resolution image sequences. For the others, they replace the residual network by a Neural ODE [6, 27] and even use a Neural ODE for the encoder [27] as well, but the adversarial and additional losses are necessary to reach the state-of-the-art performances. The most common methods are based on geometric transformation [4, 28] between frames or use optical flow [33, 11, 30, 12] because generating an image from scratch is hard while dealing with complex datas. This method works well for short-term predictions but not long-term forecasting. The disentanglement of the content and the dynamics appears as a solution to foster long-term forecasting but neural ODE is not yet used in order to achieve long-term predictions of videos.

5 EXPLORATION

5.1 Neural Ordinary Differential Equations

5.1.1 • Theory

Neural Ordinary Differential Equations [23] (also called **Neural ODE** or **NODE**) is a new family of deep learning models. It can be seen as a generalization of residual networks, recurrent neural network decoders, and normalizing flows. For all of them, information is propagated by composing a sequence of complicated transformations from a hidden state to another :

$$\mathbf{h}_{t+1} = \mathbf{h}_t + f\left(\mathbf{h}_t, \theta_t\right) \tag{1}$$

where $t \in \{0 \dots T\}$ and $\mathbf{h}_t \in \mathbb{R}^D$. These iterative updates can be seen as an Euler discretization of a continuous transformation. [8, 3]

By adding more layers and taking smaller steps, we converge towards a continuous model that is able to learn the dynamics. Therefore, by taking the limit inside the equation 1, we obtain an ordinary differential equation (ODE) where the function f is a neural network parameterized by θ :

$$\frac{d\mathbf{h}(t)}{dt} = f(\mathbf{h}(t), t, \theta) \tag{2}$$

By rewriting this equation in an integral manner, we obtain for $t \in [0, T]$ an equation that allows us to compute the gradient with respect to θ, the parameters of the neural network:

$$\mathbf{h}(\mathbf{t}) = \mathbf{h}(\mathbf{0}) + \int_{u \in [0,t]} f(\mathbf{h}(u), u, \theta)\, dx \tag{3}$$

It is useful to mention that a Neural ODE always take the initial conditions and the time steps as input and it outputs the values for the considered time steps. Their solvers work by using an ODE solver for the forward pass and back-propagate thanks to mathematical formulas of the gradient. For the formula we refer to the paper in which it was introduced [23]

This model has major benefits such as :

1. **Memory efficiency**: Back-propagating is cost-less in comparison of residual network as well as the storage of the model itself.

2. **Continuous time-series models**: The time steps of the output can be irregular whereas with residual neural network they should be the same during training and testing, giving way more flexibility for its training and use.

3. **Learning potential**: Because a lot of problems can be described by an ODE or could be approximated by an ODE that is unknown, this model can learn them by itself. In the toy

examples, it can achieve almost perfect results even outside of the training window. In addition, ax n-th order ODE can be rewritten as a first order ODE simply by transforming the input and the function describing the system.

However, this model has also some major drawbacks such as :

1. **Computation cost**: While back-propagating has a constant cost, the forward pass relies on the number of time steps used by the solver. Thus, the forward pass varies according to the input and during the training.

2. **Training time**: In comparison to residual neural network, it learns slower even for simple equations because the task itself is harder. We want to find the parameters that work in a continuous setup. For the forward pass, the ODE solver uses $f(\mathbf{h}(u), u, \theta)$ as much as the number of solving time steps, which propagates the error over the time.

5.1.2 • Experimentation for an unique trajectory

Our first experimentation to see how the Neural ODE works and behaves when we consider a spiral or circle trajectory. For that we generate the coordinates of one trajectory with $\Delta t = 0.001$ and 1000 points with a angular velocity w_0 and an exponential decay λ. The data generation was done using an Euler scheme with the following equations 4 :

$$\begin{cases} x(t+\Delta t) = x(t) - \Delta t \times w_0 \times \exp^{-\lambda t} \sin(w_0 t) \\ y(t+\Delta t) = y(t) + \Delta t \times w_0 \times \exp^{-\lambda t} \cos(w_0 t) \end{cases} \tag{4}$$

We use the first 80% coordinates for training and the 20% last for testing. Therefore, we can determine whether or not our model can extrapolate the dynamics. We tried to learn the dynamics when $x(0) = x_0$, $\dot{x}(0) = v_0$ and when λ and w_0 are fixed. To do so, we tried the simplest model that is described by the following system 5:

$$\begin{cases} \textbf{Input} : X(0) = X_0 \in \mathcal{R}^2 \\ \textbf{Equation} : \frac{d\mathbf{X}}{dt} = f(\mathbf{X}(t), t, \theta), \forall t \in [0, T] \\ \textbf{Constants} : X(0) = X_0, V(0) = 0, \lambda, w_0 \end{cases} \tag{5}$$

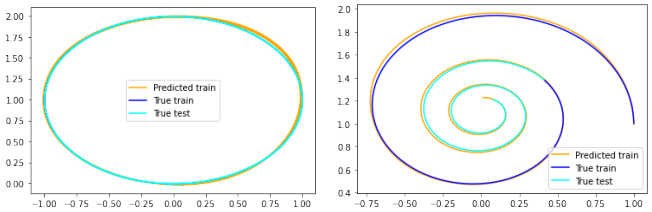

Figure 1: Results for two trajectories, one circle and one spiral, where we have in dark blue the training set ($0s \rightarrow 0.8s$), in light blue the test set ($0.8s \rightarrow 1s$) and in orange the prediction of our model ($0s \rightarrow 1s$).

Because we fixed $x(0)$ and $v(0)$, it is feasible to learn such a system with our model. On the figure 1, we can see that our model is able to learn the trajectory for the training part but

it can also extend beyond training. We also tried to do the same except that we add some gaussian noise to know the model's robustness. The following figure 2 shows us that our model can still learn a pretty accurate trajectory but it is important to mention that the output is continuous.

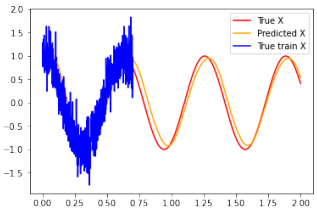

Figure 2: Result on the X axis for a circle trajectory where we have in dark blue the training set ($0s \to 0.8s$), in light blue the test set ($0.8s \to 1s$) and in orange the prediction of our model ($0s \to 1s$).

5.1.3 • Experimentation for multiple trajectories

Our final goal is to create a model that learns the dynamics of a system. In the previous section, we only learned one trajectory at a time.

Thus, at this point, we wanted to create a model that can not only predict one trajectory but different ones. However, the definition of Neural ODE is only a first order ODE, whereas the physics involved are 2nd order. The solution is to use $Z(t) = (X(t), \dot{X}(t))$ instead of X(t). Then, by fixing λ and w_0 the hyper-parameters of the dynamics, our model can predict the trajectory only by providing $Z(0) = (X_0, \dot{X}_0) \in [0, 1]^4$. The system can be rewritten as 6 :

$$\begin{cases} \textbf{Input} : Z(0) = (X(0), \dot{X}(0)) \in \mathcal{R}^4 \\ \textbf{Equation} : \dot{\mathbf{Z}} = f(\mathbf{Z}(t), t, \theta), \forall t \in [0, T] \\ \textbf{Constants} : \lambda, w_0 \end{cases} \tag{6}$$

The end goal is to learn dynamics from images and not coordinates. In the latter, we learned that we need to provide both the position and the velocity to our ODE. Thus, we can conclude it works in the case of images.

5.2 Auto-encoder

5.2.1 • Gaussian ball dataset

After using spiral and circle coordinates of trajectories, we aim to use a dataset that is its direct translation in terms of images. We generate the dataset as follow :

1. **Sampling:** $X_0, V_0 \sim \mathcal{U}(-1, 1) \times \mathcal{U}(-1, 1)$
2. **Simulating trajectory:** By using (X_0, V_0) and the hyper-parameters of the physics we gather the simulated coordinates $(X_i)_{1 \leq i \leq n}$ for all the considered time steps.

3. **Normalization:** $(\hat{X}_i)_{1\leq i\leq n} = (\frac{X_i - \min\limits_j X_j}{\max\limits_j X_j - \min\limits_j X_j})_{1\leq i\leq n} \in ([0,1]^2)^n$

4. **Image rendering:** The rendered image $I = (I_{i,j,t})_{(i,j,t)\in[1,28]^2\times[1,n]} \in [0,1]^{28\times 28}$ is equal to $I_{i,j,t} = \frac{1}{\sqrt{2\pi\sigma}} \exp^{-\frac{\|(i,j)-X_t\|_2^2}{2\sigma^2}}$ where σ is a fixed hyper-parameter

5.2.2 • Architecture

An Auto-Encoder is a model composed of two parts, one called an encoder **E**, the other a decoder **D**. The encoder is used to reduce the dimensions of the input towards the dimension of an other vector space called latent space. The decoder is used to decompress information from the latent space to obtain an image from the initial space. The goal of this model is to be able to have a good latent space to allow a compression and decompression without loosing too much information. The model can be summarized by the following equation 7 where $\|.\|$ is a distance metrics and n the batch size.

$$\begin{cases} \textbf{Input} : X \in \mathbb{R}^N \\ \textbf{Latent vector} : z = E(X) \\ \textbf{Reconstructed input} : \hat{X} = D(z) \\ \textbf{Objective} : \underset{\theta\in\Theta}{\text{argmax}} \sqrt{\frac{1}{n}\sum_{i=1}^n \|X - \hat{X}\|^2} \end{cases} \tag{7}$$

For our first exploration, we considered the MNIST dataset with its 28×28 digit images. We choose as first architecture the one proposed in [31] that is described more precisely in the figure 10 and we use the Mean Square Error (MSE) Loss. In those conditions, it was enough to learn a good representation of our dataset. However, it fails to learn our gaussian ball dataset and our model is stuck in a local minima where the output is always the black image whatever the input. Hopefully, using a pre-trained model on MNIST is able to learn our dataset afterwards but never from scratch. One solution we found was to use additional layers that give spatial information to our model. We derived this idea from the spatial encoding used in NeRF [18] and we define our additional channels as :

- **Linear channels:** For $L_x, L_y \in [0,1]^{28\times 28}$, we have $L_x = (i/28)_{i,j}$ and $L_y = (j/28)_{i,j}$
- **Additional frequencies:** For a frequency f and the l-th order, we define $S_x^{l,f} = (\sin(2^{l-1}\pi i))_{i,j}$ and $C_x^{l,f} = (\cos(2^{l-1}\pi i)_{i,j}$. Likewise, we define $S_y^{l,f} = {}^tS_x^{l,f}$ and $C_y^{l,f} = {}^tC_x^{l,f}$

For our simple dataset, the linear channels are enough but with bigger images or more complex ones, we can expect to add more additional frequencies. This solution works because the dataset had rotational symmetry and adding these channels breaks it as well as providing spatial information to better locate the ball. Thus, the ball is well reconstructed and located as shown in the figure 3. It was too hard for the model to learn it without pre-training or spatial encoding.

Even if the output images are well reconstructed, we want at the end to learn the dynamics from the images, or rather learn the dynamics inside the latent space. We were curious about

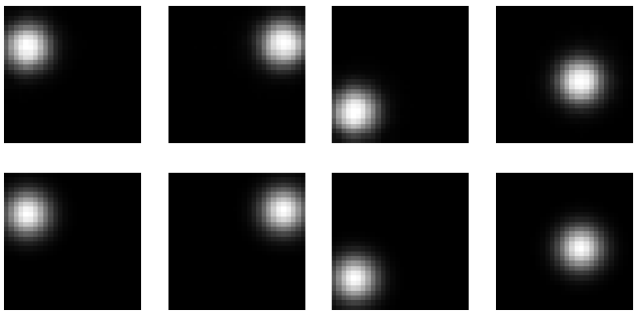

Figure 3: Comparison between predictions (on top) and ground truth (on the bottom)

the quality of the latent space without any constrains and dynamics to learn. We trained several models with different dimension for the latent space in order to compare the quality of the latent space in comparison with the trajectory itself as showed in figure 4.

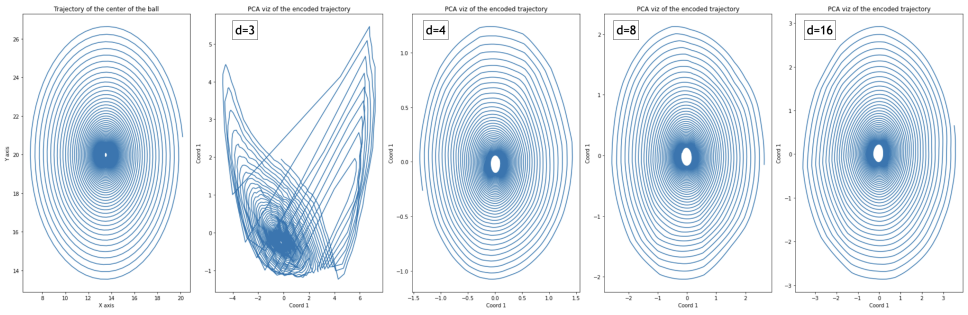

Figure 4: On the Left the true coordinates of one trajectory and the four others plot are the PCA visualisation of the two first coordinates when the latent space has respectively the dimension 3, 4, 8, 16

Surprisingly, our model couldn't fit if the latent space dimension is lower or equal than 3. In theory, the image only depends on the location of the ball's center. So, we expected to have some results with a 2 dimension latent space. In fact, adding regularization on the latent space allowed the model to fit for a 2-d latent space. In addition, adding regularization improved the quality of the latent space as showed in the figure 5.

5.3 Latent Neural Ordinary Differential Equations

5.3.1 • Architecture

From the previous steps, we learned that we need to provide information about the position and the velocity to our Neural ODE if we want to learn physical dynamics. Thus, the most natural model is to use everything that was done before.

Therefore, we define the following model (also explained in figure 6) :

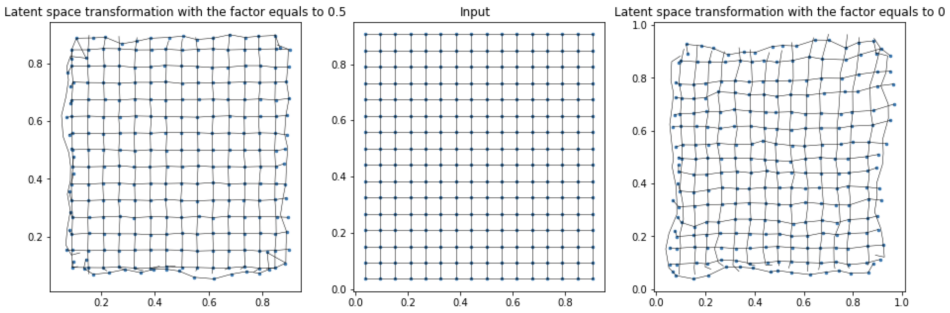

Figure 5: Comparison between the input grid (Middle) and its transformation inside the latent space with (Left) and without (Right) regularization. The input grid was used to render images and then transformed by the encoder E of our model.

- **Initial latent dynamics**: We estimate both the initial position and velocity inside the latent space by $\hat{z}_0 = E(I_0)$ and $\frac{d\hat{z}_0}{dt} = \frac{E(h_0)-E(h_0)}{dt}$). We write $\hat{Z}_0 = \begin{bmatrix} \hat{z}_0 & \frac{d\hat{z}_0}{dt} \end{bmatrix}$
- **Output latent dynamics**: Given the time steps $(t_1, ..., t_n)$ and the initial vector, the Neural ODE allows us to get the latent positions $(\hat{z_1}, ..., \hat{z_n})$ inside the latent space.
- **Reconstruction**: We reconstruct the output images $\hat{I}_i = D(\hat{z}_i)$

E contains convolutional layers followed by dense layers and D is built reversely. We estimate the vector $\hat{Z}_0$ by using the average velocity to be as closest as possible to the previous experimentation.

5.3.2 • Experimentation

We fitted our model on the gaussian ball dataset using a learning rate of 1^{-3} and a learning decay of 0.9 every 1000 epochs. We used the MSE loss between the predicted images and the original images and we added L_2 regularization on $\hat{z}_1, ..., \hat{z}_n$ multiplied by a factor 1^{-5} It took 3h to train on a GPU for 10000 epochs. The figure 7 shows how the dynamics are inside the latent space. For that, we compare the trajectory obtained by the encoding of $I_1, ..., I_n$ and the predicted position vectors $\hat{z}_1, ..., \hat{z}_n$ after a PCA transformation. We see that the ODE finds a smoother path fitting the most the overall trajectory. Our latent space needs to be the smoothest possible to be well learnt by an ODE.

In the figure 7, we see that the ODE follows pretty well the dynamics until 0.6s but comes off after. By looking at the figure 11, this shift affects both the position and appearance. However, to have the best results in long term extrapolation, we need to reduce the appearance modification to focus only on the dynamics. In our example, the appearance is simple but in other dataset, such as the Moving MNIST [20], the appearance changes. Therefore, the ODE learns not only the dynamics but also the appearance modification, leading the output to be have a deteriorated appearance. And if we want good visual quality for long term extrapolation, the appearance needs to be stable and to do not change. Thus, we followed the idea from [29] to disentangle the dynamics from the appearance.

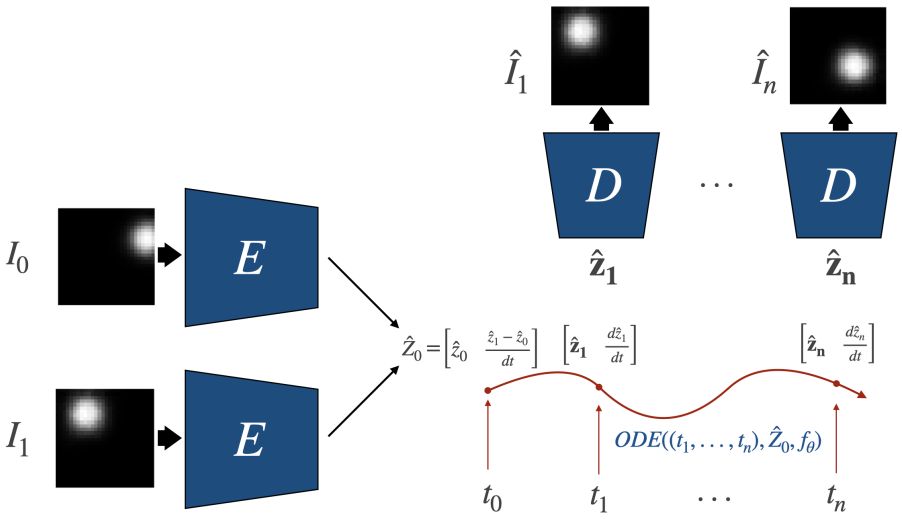

Figure 6: Summary of our model where we transform the input images I_0, I_1 thanks to E in order to build $\hat{Z}_0$ that is fed to our ODE providing use the estimations of the latent positions $(\hat{z}_1, ..., \hat{z}_n)$ for the considered time steps $(t_1, ..., t_n)$. Finally we get the predictions by decoding them with the decoder D.

6 OUR FINAL METHOD

6.1 Architecture

In order to disentangle the dynamics from the appearance, we create a model in which the architecture allows it. We propose the model in the figure 8 that generalizes the model explained in the figure 6 and adds the disentanglement. The encoder $E_{(p,v)}$ is the same as in the figure 6, we add just an appearance vector created thanks the input images by our encoder E and then concatenated to our predicted latent positions $(\hat{z}_1, ..., \hat{z}_n)$. To use the same vector for the appearance for every time steps ensures that it stores the information about the appearance.

6.2 Experimentation

To begin our experimentation, we used the method to generate the Moving MNIST [20] and we generate sequences by using one digit per sequence (instead of two for the basic dataset). We train our model by using two images as input and 18 images as output. By training the model of the figure 8 on the one digit Moving MNIST [20], our model had an interesting property. Our

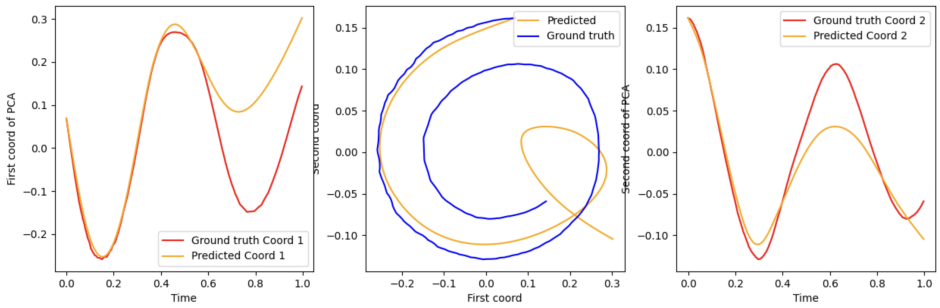

Figure 7: Comparison after a PCA transformation for a spiral trajectory between the encoded positions by E using $I_1, ..., I_n$ and the predicted latent positions $\hat{z}_1, ..., \hat{z_n}$. On the left and on the right we have respectively the first and second coordinates of the PCA over the time and in the middle we have the phase portrait.

disentanglement allowed us to change the appearance while keeping the dynamics. For that, we retrieve to dynamics $(\hat{z}_1^1, ..., \hat{z}_n^1)$ from the second input I_0^1, I_1^1 and we create the appearance vector $\hat{z}_a^2$ from the second input I_0^2, I_1^2. Then, by concatenating them like the basic model, ie creating the i-th frame equal to $\begin{bmatrix} \hat{z}_i^1 & \hat{z}_a^2 \end{bmatrix}$, we decode it with D. In the figure 12, we show the last frame. On the left, we have the ground truth image of the sequence used for the dynamics and on the right, for the appearance. In the middle, our predictions by using dynamics and appearance from different inputs. We can see that the position and the appearance are relatively accurate. The appearance isn't perfect because the disentanglement is not perfect.

6.3 Comparison

We switch to 64×64 in order to compare our results with state-of-the-art models such as PhyDNet [29], LOCI [17] and Vid-ODE [27]. We modified both the encoder and decoder to have 9 convolution layers and 9 up convolution layers. The architecture and its hyper-parameters are detailed in the appendix inside the tables 1 for the encoder and 2 for the decoder. In order to be more flexible to the number of input, we used the encoder E with all the inputs stacked to the spatial encoding forming, whereas before we used E on the two input images and then concatenating them before being encoded by the appearance encoder E_a and the dynamics encoder $E_{(p,v)}$. Thus, the number of channels in input C_in is equal to the number of input frames plus the number of spatial encoding channels.

State-of-the-art models use the 10 first images as input and extrapolate on the next 10 images. Consequently, we trained our model in this set-up for a fair comparison. We also trained PhyDNet and our model in the set-up of the 2 first images as input and the next 18 images as output. In order to compare the models, we used Peak Signal Noise Ratio (PSNR) [37] [2] and Structural Similarity index (SSIM) [37] [2]. The higher the better for SSIM and the lower the better for PSNR. We trained our model with the generation method of sequences explained in [20] and we test our model on the testing dataset provided by the University of

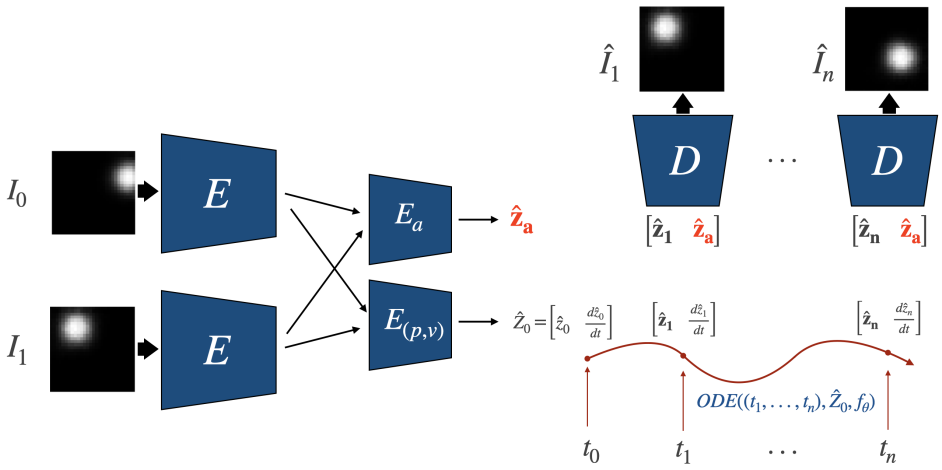

Figure 8: Summary of our model where we transform the input images I_0, I_1 thanks to E. $E_{(p,v)}$ creates the vector $\hat{Z}_0$ that is fed to our ODE providing use the estimations of the latent positions $(\hat{z}_1, ..., \hat{z}_n)$ for the considered time steps $(t_1, ..., t_n)$. Finally, we concatenate the predictions with the appearance vector $\hat{z}_a$ created by E_a and we get the predictions by decoding them with the decoder D.

Toronto [19]. This training method was also used for PhyDNet.

In the figure 9, we show the comparison of the different models on the two setups. First, we see that VidODE [27] outperforms the extrapolation task on the 10/10 setup. Our model is far away from the state-of-the-art models on this setup. Even beyond the training area our model is still far from them. However, our model succeeds better while using two and not ten images as input. We can only compare with PhyDNet but the gap is consistent, especially when we forecast after the training area. It can be explained by our encoding system that captures dynamics information inside the appearance vector. Indeed, dynamics will be added to every vector of prediction and will deteriorate the prediction quality in the short-term but above all the long term predictions. We show in the figure 14, the predictions of PhyDNet [29] and our model for a complex input. We see that PhyDNet in this setup struggles to keep decent results after the two digits being overlayed, whereas our model is less sensitive to this case. It explains therefore the gap between those two models, especially the long term predictions where necessarily the overlaying case happens.

6.4 Critics

6.4.1 • Appearance vector

Our model showed promising results with the one digit Moving MNIST [20] because it can not only well extrapolate using only the two first images but also allow us to change the appearance of another output. We admit that the disentanglement isn't perfect yet as showed in the figure 13 but the simplicity of our model and loss is promising. Using the two digits dataset,

Model	Input length	Training output length	Score PSNR	Score SSIM	Testing output length	Score PSNR	Score SSIM
PhyDNet	10	10	**22.82**	0.913	90	**16.49**	0.7253
VidODE	10	10	-	**0.934**	90	-	-
LOCI	10	10	-	0.923	90	-	**0.7715**
Our	10	10	18.13	0.853	90	11.21	0.6468
PhyDNet	2	18	16.28	0.6980	38	**15.45**	0.5712
Our	2	18	**16.44**	**0.7362**	38	15.31	**0.7107**

Figure 9: Summary of our results evaluated on the test set Moving MNIST with the PSNR (↑) and SSIM (↓) for the 10/10 and 2/18 setup but also evaluating long term prediction

we loose this property because the appearance is not fixed. Thus, our architecture struggles to create a relevant appearance vector from the input and the Neural ODE predicts both the dynamics and the appearance modifications. It shows the limitations of our architecture but there are possible solutions to foster better results with the same architecture, such as adding a loss on the appearance vector to improve its quality. Especially, we want a vector that depends only on the appearance of the moving digits and not their positions in the images nor their velocities. Thus, this assumption implies the following property (where the input sequence is of length L_{in} and S_{enc} is the used spatial encoding) :

$$\forall (I_i)_{0\leq i\leq 19}, \forall \sigma \in \mathfrak{S}$$

We define $\hat{z}_a^\sigma, \hat{z}_a$ as :

- $\hat{z}_a^\sigma = E_a(E([(I_\sigma(i))_{0\leq i\leq L_{in}}, S_{enc}]))$
- $\hat{z}_a = E_a(E([(I_i)_{0\leq i\leq L_{in}}, S_{enc}]))$

And we have the equation :

$$\hat{z}_a^\sigma = \hat{z}_a \tag{8}$$

This property 8 can be then used as a loss such as adding a L_2 loss between $\hat{z}_a$ and $\hat{z}_a^\sigma$ for a random permutation σ.

In the case of more complex appearance transformations, using a fixed vector for the appearance is pretty limited. It might work for easy dataset such as Moving MNIST [20] with one, two or more digits on each sequence as their appearances stay the same, they're just overlapped. In more complex dataset where the digits can rotate or change in scale, it might help to add a Neural ODE on top of our appearance vector $\hat{z}_a$. It could improve the final results and the quality of our disentanglement.

6.4.2 • Quality of the reconstructions

The figure 14 shows our predictions on top. By comparing our predictions with the ground truth, we see that we predict blurry outputs whereas the ground truth image is sharp. Using only the MSE loss and not any additional loss for the image quality results in blurry images. Predicting blurry images improves the performance in terms of MSE loss but not in terms of visual aspect nor the chosen evaluation metrics. One common solution of the state-of-the-art is to use losses on the visual aspect such as a perceptual loss [27, 11], adversarial loss [27, 32, 30, 15, 5] or other losses [12, 17]. For the adversarial loss, we can use one dealing with single images or the whole sequence, so that we have higher quality images and consistent sequences. A sequence adversarial loss improves its consistency over the time but also improves the quality of long-term predictions because it learns to keep visual coherence.

Also, using the MSE gives more importance to outliers and fosters blurry images. Small errors are meaningless because of the square. The Mean Absolute Error (MAE) on the contrary is less sensitive to outliers. So, small errors are more meaningful and reduce the blurry effect. However, training our model by using the MAE instead of the MSE is too hard and do not converge. The MAE can be used in addition to the MSE or we can add to the MAE other losses to help convergence.

7 CONCLUSION

We propose a Neural ODE based framework for disentangling the underlying dynamics from other image information for video forecasting. Our continuous-time model can swap the appearance while keeping the original dynamics. The short-term and long term predictions are not yet at the level of the state-of-the-art performances and the disentangling is not perfect yet. Additional losses such as sequence consistency loss, perceptual loss or other additional loss can improve our results on the SSIM and PSNR results and we can expect to improve the disentangling by modifying the architecture and adding losses.

8 APPENDIX

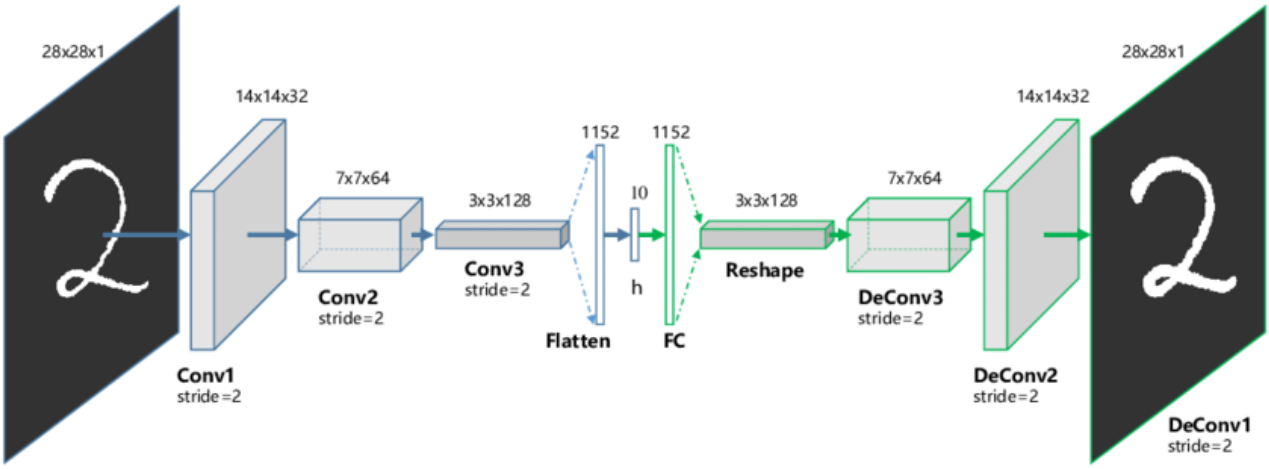

Figure 10: Architecture of the Convolutional Auto Encoder for the MNIST dataset using 3 convolutional layers and 3 up convoltional layers.

	Parameters	Encoder
Block 1	$Conv_{11}$	(C_in, 64, 3, 1, 0)
	$ReLU_{11}$	-
	$BatchNorm2D_{11}$	64
	$Conv_{12}$	(64, 64, 3, 1, 0)
	$ReLU_{12}$	-
	$BatchNorm2D_{12}$	64
	$MaxPool2D_{11}$	2, 2
Block 2	$Conv_{21}$	(64, 128, 3, 1, 0)
	$ReLU_{21}$	-
	$BatchNorm2D_{21}$	128
	$Conv_{22}$	(128, 128, 3, 1, 0)
	$ReLU_{22}$	-
	$BatchNorm2D_{21}$	128
	$MaxPool2D_{21}$	2, 2
Block 3	$Conv_{31}$	(128, 256, 3, 1, 0)
	$ReLU_{31}$	-
	$BatchNorm2D_{31}$	256
	$Conv_{32}$	(256, 256, 3, 1, 0)
	$ReLU_{32}$	-
	$BatchNorm2D_{31}$	256
	$MaxPool2D_{31}$	2, 2
Dense layers	Dynamics layer	$256{\times}4 \times 4 \rightarrow 96$
	Appearance layer	$256{\times}4 \times 4 \rightarrow 64$

Table 1: Summary of the encoder where parameters are C_in for the number of channels in input, (in channel, out channel, kernel size, stride, padding) for **convolution layers** and (kernel size, stride) for **maxpool layers**

	Parameters	Decoder
Dense layer	Dynamics and appearance layer	$160 \rightarrow 256 \times 4 \times 4$
	$relu_{01}$	-
Block 1	$Upsample_{11}$	(2, bilinear)
	$ConvTranspose_{11}$	(256, 128, 3, 1, 0)
	$ReLU_{11}$	-
	$BatchNorm2D_{11}$	128
	$ConvTranspose_{12}$	(128, 128, 3, 1, 0)
	$ReLU_{12}$	-
	$BatchNorm2D_{12}$	128
Block 2	$Upsample_{21}$	(2, bilinear)
	$ConvTranspose_{21}$	(128, 64, 3, 1, 0)
	$ReLU_{21}$	-
	$BatchNorm2D_{21}$	64
	$ConvTranspose_{22}$	(64, 64, 4, 1, 0)
	$ReLU_{22}$	-
	$BatchNorm2D_{22}$	64
Block 3	$Upsample_{31}$	(2, bilinear)
	$ConvTranspose_{31}$	(64, 32, 4, 1, 0)
	$ReLU_{31}$	-
	$BatchNorm2D_{31}$	32
	$ConvTranspose_{32}$	(32, 32, 4, 1, 0)
	$ReLU_{32}$	-
	$BatchNorm2D_{32}$	32
	$ConvTranspose_{33}$	(32, 1, 1, 1, 1)
	$Sigmoid_{31}$	-

Table 2: Summary of the decoder where parameters are (in channel, out channel, kernel size, stride, padding) for **up convolutions** and (scaling factor, interpolation method) for **upsample**

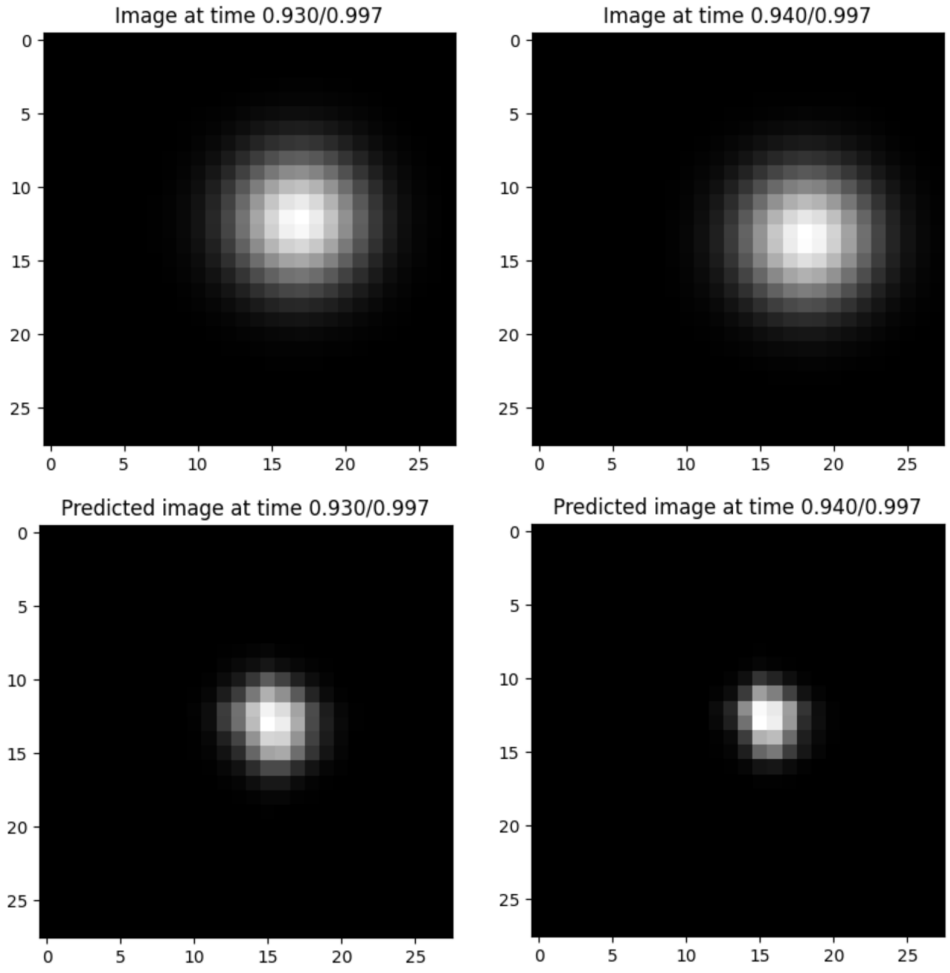

Figure 11: Comparison of the ground truth (on top) and the predictions of our model (on the bottom). We see that the appearance of the predictions is altered over the time from $0.930s \rightarrow 0.940$

Figure 12: Comparison of the last frames between the first trajectory (**left**) used for its dynamics, the second trajectory (**right**) used for its appearance, and the **accurate** prediction of our model (**middle**) using the dynamic and appearance information from different sources

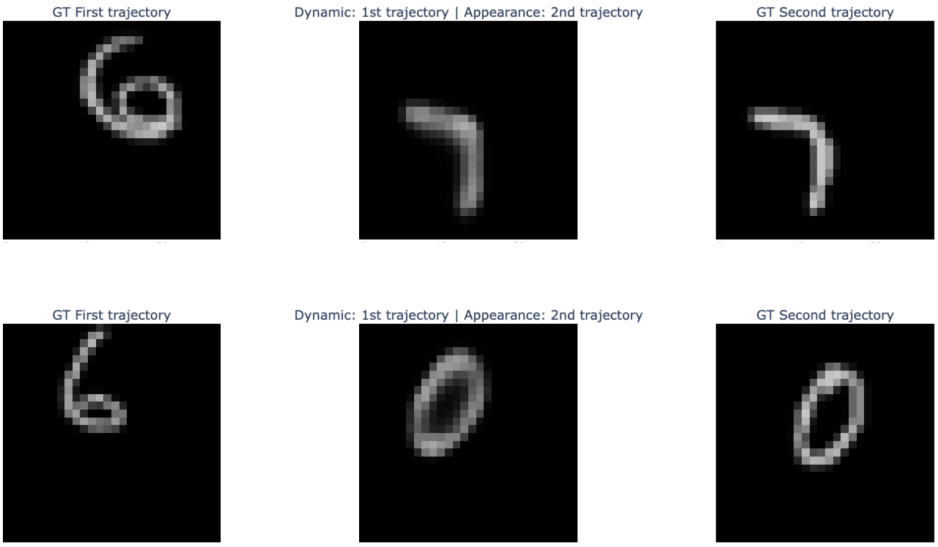

Figure 13: Comparison of the last frames between the first trajectory (**left**) used for its dynamics, the second trajectory (**right**) used for its appearance, and the **not accurate** prediction of our model (**middle**) using the dynamic and appearance information from different sources

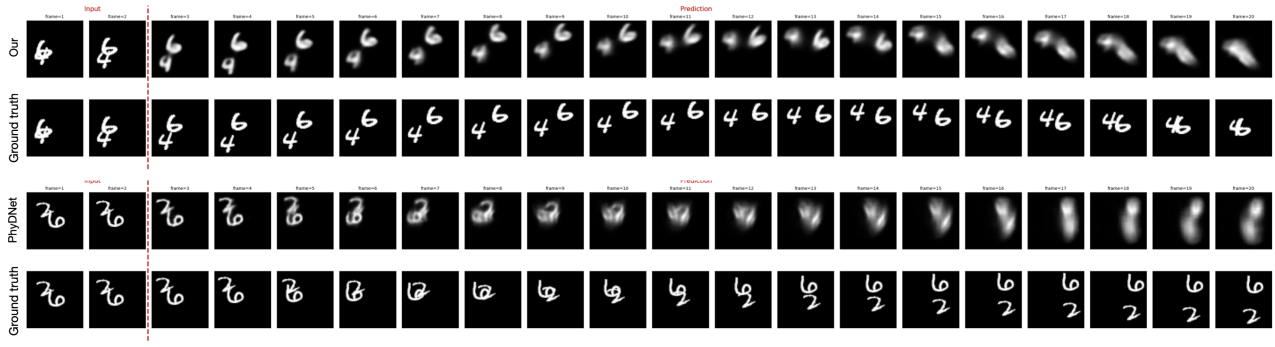

Figure 14: Comparison of PhyDNet (**bottom**) and our (**top**) model on two complex trajectories for the setup 2/18

REFERENCES

[1] Tianfu Wang Ahmed Elazab, Jamal Gardezi. Gp-gan: Brain tumor growth prediction using stacked 3d generative adversarial networks from longitudinal mr images. In *INNS*, 2020.

[2] Djemel Ziou Alain Horé. Image quality metrics: Psnr vs. ssim. 2010.

[3] Eldad Haber Lars Ruthotto David Begert Elliot Holtham Bo Chang, Lili Meng. Reversible architectures for arbitrarily deep residual neural networks. 2017.

[4] S. Levine C. Finn, I. Goodfellow. Unsupervised learning for physical interaction through video prediction. In *NeurIPS*, 2016.

[5] A. Torralba C. Vondrick, H. Pirsiavash. Generating videos with scene dynamics. In *NeurIPS*, 2016.

[6] Samira Ebrahimi Kahou Christopher Pal David Kanaa, Vikram Voleti. Simple video generation using neural odes. 2021.

[7] A. Pajot E. de Bezenac and P. Gallinari. Deep learning for physical processes: Incorporating prior scientific knowledge. In *ICLR*, 2018.

[8] Lars Ruthotto Eldad Haber. Stable architectures for deep neural networksn. 2019.

[9] Yee Whye Teh Emilien Dupont, Arnaud Doucet. Augmented neural odes. 2019.

[10] E. L. Denton et al. Unsupervised learning of disentangled representations from video. In *NeurIPS*, 2017.

[11] Aysegul Dundar Mohammad Shoeybi Guilin Liu Kevin J. Shih Andrew Tao Jan Kautz Bryan Catanzaro Fitsum A. Reda, Deqing Sun. Unsupervised video interpolation using cycle consistency. In *ICCV*, 2019.

[12] Erik Learned-Miller Jan Kautz Huaizu Jiang Deqing Sun Varun Jampani, Ming-Hsuan Yang. Super slomo: High quality estimation of multiple intermediate frames for video interpolation. 2018.

[13] D.-A. Huang L. F. Fei-Fei J. C. Niebles J.-T. Hsieh, B. Liu. Learning to decompose and disentangle representations for video prediction. In *NeurIPS*, 2018.

[14] A. Courville L. Castrejon, N. Ballas. Improved conditional vrnns for video prediction. In *ICCV*, 2019.

[15] Y. LeCun M. Mathieu, C. Couprie. Deep multi-scale video prediction beyond mean square error. In *ICLR*, 2015.

[16] R. Villegas-F. Cole K. Murphy H. Lee M. Minderer, C. Sun. Unsupervised learning of object structure and dynamics from videos. In *NeurIPS*, 2019.

[17] Tobias Menge-Matthias Karlbauer Jannik Thümmel Martin V. Butz Manuel Traub, Sebastian Otte. Learning what and where – unsupervised disentangling location and identity tracking. In *CVPR*, 2022.

[18] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

[19] Ruslan Salakhutdinov Nitish Srivastava, Elman Mansimov. Moving MNIST evaluation dataset. `https://www.cs.toronto.edu/~nitish/unsupervised_video/mnist_test_seq.npy`, 2015. [Online; accessed 10-August-2022].

[20] Ruslan Salakhutdinov Nitish Srivastava, Elman Mansimov. Unsupervised learning of video representations using lstms. 2015.

[21] Yubo Wang-Haiyong Zheng Qingyun Li, Zhibin Yu. Tumorgan: A multi-modal data augmentation framework for brain tumor segmentation. 2020.

[22] S. Hong-X. Lin R. Villegas, J. Yang and H. Lee. Decomposing motion and content for natural video sequence prediction. In *ICLR*, 2017.

[23] Jesse Bettencourt David Duvenaud Ricky T. Q. Chen, Yulia Rubanova. Neural ordinary differential equations. 2019.

[24] Y. Liu S. Seo. Differentiable physics-informed graph networks. 2019.

[25] X. Yang J. Kautz. Mocogan S. Tulyakov, M.-Y. Liu. Decomposing motion and content for video generation. In *CVPR*, 2018.

[26] H. Wang D.-Y. Yeung W.-K. Wong W.-c. Woo S. Xingjian, Z. Chen. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *NeurIPS*, 2015.

[27] Junsoo Lee Jaegul Choo Joonseok Lee-Sookyung Kim Edward Choi Sunghyun Park, Kangyeol Kim. Vid-ode: Continuous-time video generation with neural ordinary differential equation. In *CVPR*, 2020.

[28] K. Bouman B. Freeman T. Xue, J. Wu. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. In *NeurIPS*, 2016.

[29] Nicolas Thome Vincent Le Guen. Disentangling physical dynamics from unknown factors for unsupervised video prediction. In *CVPR*, 2020.

[30] W. Dai E. P. Xing X. Liang, L. Lee. Dual motion gan for future-flow embedded video prediction. In *ICCV*, 2017.

[31] En Zhu Jianping Yin Xifeng Guo, Xinwang Liu. Deep clustering with convolutional autoencoders. 2017.

[32] M.-G. Park Y.-H. Kwon. Predicting future frames using retrospective cycle gan. In *CVPR*, 2019.

[33] J. Yang Z. Wang X. Lu-M.-H. Yang Y. Li, C. Fang. Flow-grounded spatial-temporal video prediction from still images. In *ECCV*, 2018.

[34] J. Wang Z. Gao S. Y. Philip Y. Wang, M. Long. Predrnn: Recurrent neural networks for predictive learning using spatiotemporal lstms. In *NeurIPS*, 2017.

[35] M. Long J. Wang P. S. Yu Y. Wang, Z. Gao. Predrnn++: Towards a resolution of the deep-in-time dilemma in spatiotemporal predictive learning. 2018.

[36] X. Tang Y. Liu A. Agarwala. Z. Liu, R. A. Yeh. Video frame synthesis using deep voxel flow. In *ICCV*, 2017.

[37] Hamid Rahim Sheikh Zhou Wang, Alan Conrad Bovik and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. 2004.

[38] Harri Lähdesmäki Çağatay Yıldız, Markus Heinonen. Ode2vae: Deep generative second order odes with bayesian neural networks. 2019.