Exploring Methods for Enhancing Linear Prediction of Video Sequences

Master’s Thesis
Escola Tècnica d’Enginyeria de Telecomunicació de Barcelona - Universitat Politècnica de Catalunya and Northeastern University
by
Armand Comas Massague

Advisors:
Octavia Camps, Xavier Giró

July 8, 2019
Abstract

Video prediction has for a long time received attention within the field of computer vision, but it has gained importance during the last decade with the popularization of deep neural networks and their applications to computer vision.

In this thesis, the main focus will be to linearize the dynamics of time sequences by exploiting the spatial context that video offers, with the final scope of obtaining better predictions.

In the first place, we provide the theoretical base for dynamics. Following, we present several modifications for an existing deterministic predictor network called Dynamical Atoms-based Network (DYAN) [1], which models time sequences as the output of Linear Time-Invariant (LTI) systems using system identification and dynamics foundations. The solutions present different levels of success and in some cases they beat the State Of The Art (SOTA) for at least one dataset, in the metrics SSIM, MSE and MMF.

We also present two novel convolutional autoencoder architectures (LODAEs) for low order dynamics manifold embedding, strongly based on deep neural networks, with the primary aim of giving a generalized solution for mapping video sequences into a new manifold, to adapt them to the pipeline of predictors such as DYAN, based on system identification. The results for the LODAEs are promising as they seem to achieve their goal for a very simple synthetic dataset by lowering the order of the latent space sequences and providing good reconstructions and in some cases, predictions.
Acknowledgements

I would want to thank professors Octavia Camps and Mario Sznaier for greeting me in their lab and providing me with all the personal and academical support and freedom of research I needed. I also want to thank professor Xavier Giró for following my work throughout this thesis and giving me useful advice.

Also an appreciation to everyone in Robust Systems lab for their collaboration and friendship, and to all the interesting people that I have met this year in Boston, specially those who seem to have become my de facto family there.

Finally, with special emphasis I want to thank my family, for their unconditional support not only for this period of my academical path, but for every single step I’ve made until here. I couldn’t ask for more from you.
## Contents

List of Figures

List of Tables ix

List of Acronyms x

1 Introduction 1

1.1 Context .......................................................... 3

2 Background 5

2.1 Dynamics ......................................................... 5

2.1.1 Dynamics-based affine invariants ................................. 5

2.1.2 Linear Time-Invariant identification using atoms: How DYnamic Atoms-based Network (DYAN) works ................................. 6

2.2 Dynamics order minimization ...................................... 9

2.2.1 Matrix structures for order minimization ......................... 9

2.2.1.1 Hankel Matrix ................................................. 9

2.2.1.2 Gramian Matrix .............................................. 11

2.2.2 Rank minimization strategies ................................... 11

2.2.3 Re-weighted Trace Heuristic .................................... 13

2.3 DYAN’s limitations ................................................ 14

3 Models 18

3.1 Spatiotemporal DYAN ............................................. 18

3.1.1 Intuition ....................................................... 18

3.1.2 Solution ........................................................ 19

3.1.3 Implementation and Experiments ............................... 20

3.2 Group Lasso DYAN ............................................... 22

3.2.1 Intuition ....................................................... 22

3.2.2 Solution ........................................................ 22

3.2.3 Implementation and Experiments ............................... 23

3.3 Warping DYAN: Aligning optical flows ............................ 25

3.3.1 Intuition ....................................................... 25

3.3.2 Solution and architecture ...................................... 26
3.3.3 Implementation and Experiments ........................................... 28
3.4 Low Order Dynamics Autoencoders Low Order Dynamics Auto-Encoder (LODAE) ................................................................. 31
3.4.1 Re-weighted Trace Heuristic LODAE Re-weighted Trace Heuristic LODAE (RTH-LODAE) ....................................................... 32
3.4.1.1 Intuition ........................................................................ 32
3.4.1.2 Solution .................................................................... 32
3.4.1.3 Implementation and Experiments ................................. 34
3.4.2 Auto-Regressive LODAE Auto-Regressive LODAE (AR-LODAE) .. 39
3.4.2.1 Intuition .................................................................. 39
3.4.2.2 Solution .................................................................. 39
3.4.2.3 Implementation and Experiments ............................... 41

4 Conclusions and Future Work ..................................................... 46
4.1 Conclusions ........................................................................ 46
4.2 Future Work .................................................................... 47

Bibliography .............................................................................. 48
List of Figures

2.1 DYAN identifies the dynamics for each pixel, expressing them as a linear combination of a small subset of dynamics-based atoms from a dictionary (learned during training). The selected atoms and the corresponding coefficients are represented using sparse feature vectors, found by a sparsification step. These features are used by the decoder to reconstruct the input data and predict the next frame by using the same dictionary, but with an extended temporal horizon. [1] ................................. 8

2.2 Global architecture of DYAN. It takes pixel sequences as time sequences (1 : T), obtains a sparse code (with length N) with the encoder and reconstructs and predicts the next value (1 : T + 1) of the original sequences with the decoder. ....................................................... 9

2.3 Different relaxations of the rank function. Relaxation by absolute value |·|, ℓ2 norm or log det (·). .......................................................... 12

2.4 Illustration of spatial context exploitation. Portion of a video sequence of Caltech dataset (presented further in this document). The orange dots represent the observed location. The graphs in the bottom are a simplified illustration of how the sequence might look like. A task such as tracking is an example of how understanding the spatial distribution of a video sequence can produce input sequences with very simple dynamics. .... 15

3.1 Architecture of SpatioTemporal DYAN (ST-DYAN). It is a nested autoencoder. The spatial encoder provides the new inputs (coefficients c) to the temporal DYAN, which predicts. The spatial decoder maps back the coefficients to the original domain. ................................. 19

3.2 Qualitative results for ST-DYAN in the following setup: each axis has been split in 4 chunks, the weight of the sparsity term is λ = 0.2 and the dictionary is composed by 40 poles (mirrored horizontally and vertically in the unit circle) plus a pole in the unity of the real axis. ............... 21
3.3 Qualitative results given by Group Lasso DYAN (GL-DYAN) after 20 epochs. Similarly to the original DYAN architecture, we can see lagging (red/white lines) in both Optical Flows (OF)s and Red Green Blue (channels of an image) (RGB) frames and artifacts (red circle) in the warped predictions. The weight of the sparsity term is $\lambda = 0.2$ and the dictionary is composed by 40 poles (mirrored horizontally and vertically in the unit circle) plus a pole in the unity of the real axis. .......................... 23

3.4 Poles of the dictionary of GL-DYAN after 20 epochs where the most used poles have higher area and a greener color. This shows that the most used poles are in the right half of the unit circle (non-oscillating impulse response of the system). There is a second activated are close to the imaginary axis and the negative side of the real axis. Same parameters as in figure 3.3................................. 24

3.5 A set of overlapping frames. (a) Trajectories of two objects. (b) Eulerian point of view looks at changes at a fixed location. (c) Lagrangian point of view looks at changes along trajectories. (d) Recursive warping aligns all the optical flow frames by placing the optical flow of corresponding pixels at their final location [2].................................. 26

3.6 An Eulerian observer sees changes in optical flow at pixels (i.e $(x_2,y_2)$) on occluding boundaries ([2]).................................. 26

3.7 W-DYAN uses recursive backward optical flow warping to align corresponding pixels. In the figure, the first three optical flows in the left column are aligned (right most column) in order to predict the fourth optical flow of $4$. Colors indicate objects, while the numbers indicate the horizontal optical flow of the object. As the diagram shows, temporal sequences of Lagrangian optical flow are smoother than the corresponding Eulerian optical flow. In this synthetic example different cases are shown: constant velocity, accelerations, occlusions and oscillation. Note that only horizontal motion is represented, namely, the first channel of an OF. .... 27

3.8 Qualitative example predicting 5 future frames for the Caltech dataset set $10V011$. Top row shows the ground truth frames. The right column shows details from the fifth predictions. Circled in red: incorrect position of the front tire and orientation of the window frame of the car; Green arrows: sharper roof line, correct orientation of the window frame ([2]). .... 29

3.9 UCF101 qualitative example. Warping DYAN (W-DYAN) makes sharp predictions with less artifacts than DYAN’s output. The second row shows details from the predictions. Circled in red: incorrect predictions (artifacts or lagging); Circled in green: improvement with respect to DYAN’s output. (modified from [2]). .... 29
3.10 Global architecture of RTH-LODAE. It uses the encoder $E_\theta$ to map the inputs $y_{(1:T)}$ into the latent space represented by $z$ where the linearization loss is computed by generating the matrices $H_z$ and $G_z$ at each iteration of the heuristic. The buffer represents the memory required to save the $z$ values computed in the previous iteration of the heuristic. The decoder $D_\phi$ maps the latent representations back to the original domain, where the reconstruction loss is computed.

3.11 Training strategy for RTH-LODAE. We initialize $G_z^{(k-1)}$ to $I$ and compute $K$ iterations with the same instance $i$ of the inputs computing reconstruction and linearization loss at each step and backpropagating from the general loss. Finally, we switch the input to the instance $i + 1$ and reinitialize. We repeat until processing all the training data.

3.12 Sample of an input sequence for the moving MNIST dataset, and the corresponding reconstructions of a trained LODAE.

3.13 Qualitative results for RTH-LODAE. The lower row shows the reconstructions by the LODAE architecture, which seem to be good.

3.14 Qualitative results for dynamics linearization of RTH-LODAE. Top two heatmaps show the performance on linearization of the latent spaces dynamics without standardization in testing, before and after training. Bottom row shows the same results with standardization at the input of the latent space and unstandardization before decoding.

3.15 AR-LODAE training approaches. $E_\theta$ indicates the encoder, which is composed by the convolutional and fully connected modules, in a similar way as the decoder $D_\phi$, but mirrored. It is also shown how the losses are computed in both cases.

3.16 Qualitative results for the first approach of AR-LODAE. It shows 4 clusters (training/testing 20/280 epochs) of 3 ground truth frames (top of cluster) and the reconstructed (bottom left and center of cluster) and predicted (bottom right of cluster) frames. Red circle: highlight good location prediction after few epochs.

3.17 Qualitative results for the second approach of AR-LODAE. Same disposition as in figure 3.16.

3.18 Qualitative results for the first approach of AR-LODAE. Representation of the heatmaps for the normalized singular values of the Hankel matrices $H_z$ of the latent space. The rows of each heatmap represent the normalized singular values from lower to higher (top down). The $x$ axis or columns represent each one of the dimensions for all batches. Cases: training/testing and after 1/100 epochs. Gray rectangles: highlight the last singular values after several epochs (the lower the better).
3.19 Qualitative results for the first approach of AR-LODAE. Representation of the heatmaps for the normalized singular values of the Hankel matrices $H_z$ of the latent space. Same disposition as in figure 3.18. Cases: training/testing and after 1/280 epochs.
List of Tables

3.1 Quantitative results for single frame prediction on Caltech and UCF101 datasets. Except for DYAN’s scores (for which we ran the code provided by the authors), we adopted the scores from [3] and [4]. The modified DYAN networks, W-DYAN, K-DYAN, and KW-DYAN yield the best performance on most metrics, with the smallest model size ([2]). 30

3.2 Quantitative results and ablation study for single frame prediction on Moving MNIST dataset for RTH-LODAE, with fixed number of iterations and variations in sequence length $T$ and linearization weight $\lambda$. We test for the Mean Square Error (MSE) in reconstruction and Mean Last Singular Value (MLSV). We can see that reconstruction is similar in all cases, but $\lambda$ influences substantially on the dynamics linearization. 37

3.3 Quantitative results and ablation study for single frame prediction on Moving MNIST dataset for RTH-LODAE, with fixed $\lambda$ and $T$ and variations in number of iterations for the Re-weighted Trace Heuristic (RTH). We test for the MSE in reconstruction and MLSV. We can see that reconstruction is similar in all cases, but the number of iterations influences considerably on the dynamics linearization. 38

3.4 Quantitative results for the first approach of AR-LODAE. We vary the weight $\lambda$ and the sequence length $T$ for a fixed $z_{\text{dim}} = 128$. In red the best results. 43

3.5 Quantitative results for the second approach of AR-LODAE. We vary the weight $\lambda$ and the sequence length $T$ for a fixed $z_{\text{dim}} = 128$. In red the best results. 44

3.6 Quantitative results for the second approach of AR-LODAE. For a fixed $\lambda = 1e - 3$, sequence length $T = 9$ we vary $z_{\text{dim}}$ to test different. In red the best results. We can see better linearization results as we decrease $z_{\text{dim}}$ but a clear best prediction for $z_{\text{dim}} = 128$. 45

3.7 Quantitative results for the second approach of AR-LODAE. For a fixed $\lambda = 1e - 3$, sequence length $T = 9$ and $z_{\text{dim}} = 0$ we test the network when the digits in the input have velocity $v = 0$. It behaves as expected as reconstruction and prediction are equally low and MLSV is 0. 45
# Acronyms list

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECCV</td>
<td>European Conference on Computer Vision</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
<tr>
<td>RGB</td>
<td>Red Green Blue (channels of an image)</td>
</tr>
<tr>
<td>OF</td>
<td>Optical Flows</td>
</tr>
<tr>
<td>SOTA</td>
<td>State Of The Art</td>
</tr>
<tr>
<td>SSIM</td>
<td>Structural Similarity Index Measure</td>
</tr>
<tr>
<td>PSNR</td>
<td>Peak Signal-to-Noise Ratio</td>
</tr>
<tr>
<td>MSE</td>
<td>Mean Square Error</td>
</tr>
<tr>
<td>LPIPS</td>
<td>Learned Perceptual Image Patch Similarity</td>
</tr>
<tr>
<td>MMF</td>
<td>Mean of the Maximum optical Flow</td>
</tr>
<tr>
<td>MLSV</td>
<td>Mean Last Singular Value</td>
</tr>
<tr>
<td>LTI</td>
<td>Linear Time-Invariant</td>
</tr>
<tr>
<td>RTH</td>
<td>Re-weighted Trace Heuristic</td>
</tr>
<tr>
<td>SVD</td>
<td>Singular Value Decomposition</td>
</tr>
<tr>
<td>PSD</td>
<td>Positive Semi-Definite</td>
</tr>
<tr>
<td>FISTA</td>
<td>Fast Iterative Shrinkage-Thresholding Algorithm</td>
</tr>
<tr>
<td>DYAN</td>
<td>DYnamic Atoms-based Network</td>
</tr>
<tr>
<td>W-DYAN</td>
<td>Warping DYAN</td>
</tr>
<tr>
<td>K-DYAN</td>
<td>Kalman DYAN</td>
</tr>
<tr>
<td>KW-DYAN</td>
<td>Kalman-Warping DYAN</td>
</tr>
<tr>
<td>ST-DYAN</td>
<td>SpatioTemporal DYAN</td>
</tr>
<tr>
<td>GL-DYAN</td>
<td>Group Lasso DYAN</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long Short-Term Memory</td>
</tr>
<tr>
<td>GRU</td>
<td>Gated Recurrent Unit</td>
</tr>
<tr>
<td>GAN</td>
<td>Generative Adversarial Network</td>
</tr>
<tr>
<td>VAE</td>
<td>Variational Auto-Encoder</td>
</tr>
<tr>
<td>RNN</td>
<td>Recurrent Neural Network</td>
</tr>
<tr>
<td>LODAE</td>
<td>Low Order Dynamics Auto-Encoder</td>
</tr>
<tr>
<td>RTH-LODAE</td>
<td>Re-weighted Trace Heuristic LODAE</td>
</tr>
<tr>
<td>AR-LODAE</td>
<td>Auto-Regressive LODAE</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This project has been carried out in Robust Systems lab at Northeastern University. This laboratory is coordinated jointly by Octavia Camps, whose primary research is related to computer vision, and Mario Sznaier, specialized in control and optimization. Most part of the project is in collaboration with some of the Ph.D. candidates and I have used previous work that had been produced in the laboratory as a baseline for my own work. This heterogeneous environment eases the introduction of concepts from control and optimization, such as system identification techniques or certain kind of algorithms, to the solutions proposed to computer vision problems, which has been reflected in my work.

This collaboration with the laboratory and thus, this work, has been build on top of a recently published ECCV’18 paper called: DYnamical Atoms-Based Network (DYAN) [1], which presents a novel architecture for deterministic video prediction that joins concepts of deep learning and system identification. I will widely develop it further in this document. As a consequence of working with this baseline, several questions have emerged; Can time sequences of real video frames be correctly modeled by linear systems? Which are the main drawbacks of this approach and how can they be solved? Can these sequences be transformed to lay in a different manifold in which they can be correctly modeled by linear time-invariant systems? How can deep neural networks help us in these tasks? These and other questions are addressed in this document and several solutions, with different degrees of success are presented.

Consequently, this project aims to explore different approaches to video prediction with the restrictions of: basing prediction methods on system identification and using Encoder-Decoder architectures.

This work is divided in several sections. We will start by providing a context of the field, and more concretely how video prediction is approached currently further in this chapter (in section 1.1). In chapter 2, we give the fundamental theory to understand the intuition behind the presented models, and in which the mentioned DYAN will be explained. In chapter 3, the most extensive part of the document, we describe the five architectures proposed with the following structure: the intuition behind them, the de-
scription of the solution and the experiments for testing. Finally, in chapter 4 we give our conclusions regarding the achieved results and the future lines of research that arise from this work. Note that throughout the document we will describe deep learning networks whose theoretical basis is assumed known by the reader.

During the project development, there has been a continuous supervision by Octavia Camps and Mario Sznaier, who also helped in providing some of the theoretical background for this work. More specifically, there has been a checkpoint meeting every week. Xavier Giró, my supervisor in Barcelona provided me with weekly guidance, more strictly since the second half of the project.

There hasn’t been a scheduled process, as the project started by learning and participating in the development of the already existing projects. It has been after the submission of a paper in February, that this work has been independent of the laboratory’s state of the art.

Summarizing, this project has ultimately contributed in giving a new point of view on the existing line of research in Robust System Lab, and possible contributions to the field.
1.1 Context

Deep neural networks (and therefore deep learning) appeared relatively recently as a very powerful solution for many computer vision problems, and it has achieved accuracy in problem resolution that has largely surpassed any previous technique in several tasks, such as pose estimation, activity recognition, video prediction and understanding, object detection and tracking and an incredibly extensive list of other well known tasks in the field of computer vision.

However, deep neural networks have been adopted at such speed, that many issues regarding them have been overlooked by most of the community.

The first cost inevitably linked to the use of these networks is interpretability, and in different aspects. The inherent complexity of the addition of several layers composed by a large amount of units, weighted links and biases, makes extremely difficult to know or understand the shape of the problem that is being solved, and therefore define the best possible strategy to solve it or understanding the results. Also, the parameters of the networks are not always easy to interpret and they lack of apparent physical meaning.

The current availability of very large datasets, often annotated, and the usage of Graphics Processing Unit (GPU)s together with programming frameworks that ease the task of interacting with deep architectures and using them, have made possible the usage of deep neural networks for learning in a reasonable time span for several specific tasks.

However, as the complexity increases, other well known costs of deep learning appear. Large amounts of the wanted data with the correct labeling are not always available, and when they are, the great amount of parameters used and the time that is required to tune them during the optimization process is considerably large.

In many cases, deep learning also fails when it switches domain. In other words, if the training data differs substantially (has a different distribution) from the test data, it is very probable that the performance drops significantly. Of course, this always relies on the specific architecture that is used, but it is fair to say that this behavior is generally common.

A great part of the previously exposed drawbacks has been addressed in some of the works carried out at Robust Systems laboratory, in which I did this project. As mentioned, the laboratory is formed by the union of two groups specialized in different fields; computer vision and control and optimization. This heterogeneity, has resulted in interesting work pieces combining both fields. The most recently published one, is called DYAN [1], which was published in European Conference on Computer Vision (ECCV) 2018, and intends to give a new perspective on the specific task of video prediction, while mitigating (sometimes a great deal) the problems exposed above. It preserves the training scheme of deep learning but it introduces notions of dynamics and sparse coding, and without supervision.

Video prediction has recently been one of the hot topics in the field given the importance of forecasting the future for many tasks. However, it has also been used as a
proxy to induce an internal representation of the time-space dependencies in the frame evolution within a video sequence. The work of [5] shows that learning representations by predicting the next sequence of image features improves classification in some datasets. In the specific case of DYAN, prediction is used both as an end and a mean to learn a representation of the dynamics, as it will be shown further in this document.

The task of predicting future frames from a known sequence will of course require generative models with the ability to forecast from the available past data. This has been successfully done with Recurrent Neural Network (RNN) and more specifically with the Long Short-Term Memory (LSTM) layer (as recently in [6]). Gated Recurrent Unit (GRU) have also been used to model time dependencies while addressing the vanishing gradient problem (that often occurs while training LSTMs). However, these architectures have ad-hoc designs, with many components whose purpose might be difficult to interpret.

The most recent approaches are often based on generative architectures, such as Variational Auto-Encoder (VAE) or Generative Adversarial Network (GAN). They focus on a good frame generation for the prediction and a good approximation of the underlying distribution of the data, with the intuition that by learning to generate predictions indistinguishable from ground truth in training, the network will also learn the representation of time-space dependencies within video sequences [7, 8]. As they are stochastic, they approach solutions such as multiple future generation. Instead, deterministic approaches tend to average the possible futures into one [9, 10], resulting in blurry predictions. Stochastic architectures have been very successful in the last few years for generation in many tasks, but they have proven very hard to train. In the case of GANs, for instance, training requires finding the Nash equilibrium of a game which is not trivial.

For video generation, DYAN was proposed to exploit the known dynamical behavior of real images in time-sequences. Similarly to LSTMs, DYAN can capture short and long term dependencies but it uses concepts of system identification theory to ease the task, reduce drastically the amount of parameters and introduce physical meaning and interpretability. As it will be later developed, DYAN captures dynamics-based affine invariants, which are represented by the weighted combination of “atoms” (or poles) stored in a dictionary. This creates a set of very sparse features that can also be used for other tasks such as activity recognition for a video sequence or semantic segmentation.
Chapter 2

Background

To understand the architectures and methods that will be explained further in this document, it is important to firstly explain the foundations on which they are built. We will now give some fundamental theory as a support for the whole work. Moreover, we provide a list of limitations associated to DYAN and propose solutions that may help improve them, which is the main motivation of this work.

2.1 Dynamics

As introduced, DYAN is strongly based in concepts of dynamic system identification theory, and a large part of this work is at the same time based on DYAN. This section will focus on describing those theoretical fundamentals, and relate them to DYAN and to other theory that will be used further on.

2.1.1 Dynamics-based affine invariants

Dynamics-based affine invariants are strongly based on properties of LTI systems. They lay on the idea that the sequential data can be modelled as the output of a LTI system, unknown a priori, which at the same time can be modelled as the linear combination of several first order LTI systems.

This sequential data can be, for instance, the trajectory of a tracked object or the time evolution of a pixel’s value.

The LTI systems by which the sequential data is modelled has several attributes that are invariant to any affine transformation. In other words, they are invariant to a switch in our coordinates system. There are two ways of describing an LTI system: with an autoregressive model or with a state-space model:

\[
\begin{align*}
y_k &= \sum_{i=1}^{n} a_i y_{k-i} \quad \% \text{Autoregressive Representation} \\
x_{k+1} &= Ax_k; \quad y_k = Cx_k \quad \% \text{State Space Representation}
\end{align*}
\]
with
\[
x_k = \begin{bmatrix} y_{k-n} \\ \vdots \\ y_k \end{bmatrix}, \quad A = \begin{bmatrix} 0 & 1 & \ldots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \\ a_n & a_{n-1} & \ldots & a_1 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & \ldots & 0 & 1 \end{bmatrix}.
\]

where \( y_k \) is the observation at time \( k \), and \( n \) is the (unknown a priori) order of the model. As seen in Equation 3 of the original paper of DYAN, we can prove that assuming non-repeated poles, any sequence in the \( z \)-domain \( Y(z) \) can be expressed as:

\[
Y(z) = \sum_{i=1}^{n} \frac{z c_i}{z - p_i}
\]

and therefore, in the discrete time domain:

\[
y(k) = \sum_{i=1}^{n} c_i p_i^k, \quad k = 1, 2, \ldots
\]

where \( c_i \) is a set of coefficients that depend on the initial conditions and the poles (and roots of the denominator) \( p_i \) are the eigenvalues of \( A \). For a demonstration of this expressions refer to [1].

Also, if we consider now an affine transformation \( \Pi \), and using the autoregressive model described in equation 2.1, we have:

\[
y'_k = \Pi(y_k) = \Pi\left( \sum_{i=1}^{n} a_i y_{k-i} \right) = \sum_{i=1}^{n} a_i \Pi(y_{k-i})
\]

Thus, it can be seen that both the order \( n \) and the coefficients \( a_i \) are not affected by the transformation. This property allows us to describe several different time sequences with the same set of coefficients of the autoregressive model.

### 2.1.2 Linear Time-Invariant identification using atoms: How DYAN works

DYAN’s goal is to identify a system for each sequence that we input to the architecture. With this objective we should define a cost function and constraints and solve it with an available method. Note that what DYAN is learning is not how to identify a single system, but the correct finite set of variables (poles in this case) that can make this task possible for many different sequences. Thus, we should before define the problem and propose an algorithm to solve it. In this case we summarize an atoms-based algorithm to identify an LTI system from a given sequence.

We define an atom as the impulsive response of an LTI first order (or second order in

---

\(^1\)For simplicity of notation, we consider here \( y_k \) scalar, but the invariants also hold for \( y_k \in \mathbb{R}^d \).
2.1 Dynamics

case of complex conjugates) system. Thus, the expression of an atom is given by:

\[ G_p(z) = \frac{\omega z}{z - p} \quad \text{and} \quad G_p(z) = \frac{\omega z}{z - p} + \frac{\omega^* z}{z - p^*} \]  

(2.6)

where \( \omega \in \mathbb{C} \), and impulse responses given by \( g_p = \omega [1, p, p^2, p^3, \ldots] \) and \( g_p = \omega [1, p, p^2, p^3, \ldots] + \omega^* [1, p^*, p^2*, p^3*, \ldots] \). If we had an infinite pool of poles, we can approximate every transfer function arbitrarily precisely by combining linearly a set of transfer functions associated to a pole or a pole pair. Hence, dynamical models can be estimated by solving the following optimization problem:

\[
\min_{c \in \{c_i\}} \|c\|_0 \quad \text{s.t.} \quad \|y - \sum c_i g_p\|_2 \leq \eta \]  

(2.7)

where \( \|\cdot\|_0 \) denotes cardinality and in this case sparsity, and the constraint denotes fidelity to the input data with \( c_i \) as coefficients multiplying impulse responses \( g_p \).

There are two major issues in the previous equation. Firstly, minimizing cardinality is an NP-Hard problem and secondly, the reconstruction requires considering an infinite pool of poles. We therefore relax the problem by approximating cardinality with the \( \ell_1 \) norm, and define a finite set of atoms (a dictionary) that will be learned and will define our system. We redefine our problem as:

\[
c^*_l = \min_{c \in \{c_i\}} \frac{1}{2} \left\| y_{l,1:T} - D^{(T)} c \right\|^2_2 + \lambda \|c\|_1, \quad l = 1, \ldots, HW
\]  

(2.8)

where

\[
D^{(T)} = \begin{pmatrix}
p_0^0 & p_1^0 & \cdots & p_N^0 \\
p_1^1 & p_1^1 & \cdots & p_N^1 \\
p_1^2 & p_2^2 & \cdots & p_N^2 \\
\vdots & \vdots & \ddots & \vdots \\
p_1^{T-1} & p_2^{T-1} & \cdots & p_N^{T-1}
\end{pmatrix}
\]

(2.9)

is the dictionary that englobes the impulse responses of our pool of poles, the atoms. The superindices represent the discrete time instance, and the subindices in equation part 2.9, the index of every pole within the pool. When implementing it for DYAN, we divide each impulse response associated to pole \( p_n^t \) in its cosine and sine components given the module \( \rho_n^t \) and phase \( \psi_n \) in the unit circle. As seen in equation 2.10, a column of ones has been included to model values constant in time.
2.1 Dynamics

Figure 2.1: DYAN identifies the dynamics for each pixel, expressing them as a linear combination of a small subset of dynamics-based atoms from a dictionary (learned during training). The selected atoms and the corresponding coefficients are represented using sparse feature vectors, found by a sparsification step. These features are used by the decoder to reconstruct the input data and predict the next frame by using the same dictionary, but with an extended temporal horizon. [1]

The architecture of DYAN (figure 2.2) will be divided in Encoder and Decoder, very similarly to a traditional auto-encoder structure.

With the encoder, DYAN will solve the problem presented in equation 2.8 by using a sparse recovery algorithm called Fast Iterative Shrinkage-Thresholding Algorithm (FISTA) [11], which was chosen particularly because there were very efficient implementations for GPU available. The pseudo-code of this algorithm is presented in Section 4.1 of [1]. We will not focus in this algorithm solver throughout this document. During the experiments, the maximum iterations set for FISTA will be 100 unless specified differently.

As indicated in equation 2.8 every pixel location of a set of frames of height $H$ and width $W$ belonging to a video, is interpreted as a time sequence. In this case, this will be the input sequence to our optimization problem. Therefore, using FISTA algorithm, DYAN’s encoder produces a sparse vector $c$ of coefficients for each pixel time sequence, which indicate the weight for each column of the dictionary that linearly combined will estimate such sequence.

Intuitively, if we add one row to the dictionary corresponding to the time $t = T$ and decode from the latent space (the coefficients), we will obtain a prediction of each pixel sequence, and therefore, the full predicted frame of a video. To obtain such prediction, DYAN’s decoder extends the dictionary and maps the latent information back to the original domain. The prediction is given by:

$$y_{l,(1:T+1)} = D^{(T+1)} c_l$$ (2.11)

where the added row of the dictionary will be:
2.2 Dynamics order minimization

Despite the work directly related with DYAN, this project explores different ways of mapping the input sequences in a manifold in which the dynamics of the sequences can be modelled by lower order LTI systems. If this task is achieved, linear predictors such as DYAN applied to the learned low order latent space, will give better predictions than applied directly to the original domain. However, it is not trivial to understand how this task can be approached, or even how can we obtain an actual measurement of the order of a set of modelled time sequences. For this reason, we now introduce some tools that will help us perform both tasks.

2.2.1 Matrix structures for order minimization

As introduced, there are some tools that we can use to obtain the order of a linear system. At Robust Systems Lab, structures such as the Hankel Matrix (and block Hankel matrix) or the Gramian matrix have been used for several tasks regarding rank minimization [12, 13, 14, 15], and this is due to the most interesting property (for us) that they have. The rank of those matrices, when constructed from a set of measurements in time, indicates the order of the LTI system that can model those measurements.

2.2.1.1 Hankel Matrix

The Hankel matrix is a square matrix used in linear algebra, in which each ascending skew-diagonal from left to right is constant, which is similar in shape to the well-known Toeplitz matrix. It has the following shape:

\[
D^{(T:T+1)} = \begin{pmatrix}
1 & \rho_1^{(T)} \cos(T)\psi_1 & \rho_1^{(T)} \sin(T)\psi_1 & \cdots & -(\rho_N)^{(T)} \sin(T)\psi_N \\
\end{pmatrix}
\]  

(2.12)

A graphical representation of DYAN’s workflow, for a better understanding is represented in figure 2.1.

Figure 2.2: Global architecture of DYAN. It takes pixel sequences as time sequences (1 : T), obtains a sparsecode (with length N) with the encoder and reconstructs and predicts the next value (1 : T + 1) of the original sequences with the decoder.
\[ H = \begin{pmatrix} a & b & c & d & e \\ b & c & d & e & f \\ c & d & e & f & g \\ d & e & f & g & h \\ e & f & g & h & i \end{pmatrix} \] (2.13)

When we use it to describe a set of measurements in time, it is constructed in the following way:

\[ H_y = \begin{pmatrix} y_1 & y_2 & y_3 & \cdots & y_{(T+1)/2} \\ y_2 & y_3 & \cdots & \cdots & \vdots \\ y_3 & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & y_{T-2} & y_{T-1} \\ y_{(T+1)/2} & \cdots & \cdots & y_{T-2} & y_{T-1} \end{pmatrix} \] (2.14)

where \( T \) is the time length of the sequence and \( y_t \), the set of measurements. There is a variant of this matrix. The block hankel matrix, which is not square, is used to obtain the order of more than one time sequence when those should be modeled by the same system. However, throughout this document we will focus on the square hankel matrix that represents only one system. As said, the interesting property of this matrix is that the rank of it indicates the order of the linear system that would model the time sequence forming the matrix. Intuitively, for instance, if the last column of the matrix is a linear combination of the previous columns, this means that the last measurement can be expressed by a linear combination of the previous measurements, therefore there is an underlying linear system that can represent the whole sequence, and its order is \((T - 1)/2\).

Hence, it is our objective to see which is the rank of such matrix in order to estimate the order of the underlying system. This task can be achieved by doing a Singular Value Decomposition (SVD) on the Hankel matrix. After performing this decomposition, we obtain a set of singular values (together with their respective singular vectors). If any of those values is \( \sigma_i = 0 \), the matrix is rank deficient, and therefore we know that we can model the input time measurement as a linear system. If the matrix is rank complete it gives us no useful information, given that if the system underlying the time sequence was non-linear, the matrix would also with very high probability be rank complete.

Also, the number of non-zero singular values will provide us with the estimated order of the underlying system:

\[ \text{Order}(y) = \text{Rank}(H_y) = \|\sigma_H\|_0, \quad \text{for} \quad \sigma_{H,i} \neq 0 \] (2.15)

where \( \sigma \) is the vector with the non-zero singular values associated to \( H_y \), and \( \sigma_i \) are each one of the components of \( \sigma \). \( \|\|_0 \) (or \( \ell_0 \)) indicates cardinality, and thus, the number of elements within the vector.
2.2 Dynamics order minimization

2.2.1.2 Gramian Matrix

The Hankel matrix is used for a substantial variety of methods. However, the Hankel matrix is not Positive Semi-Definite (PSD) by definition. A matrix $M$ is PSD if and only if the scalar given by $x^T M x$ is positive or 0 for every non-zero column vector $x$ containing only real values. In practice, this means that the eigenvalues $\lambda$ associated with the matrix are not necessarily positive or 0, but they can also be negative. This prevents some of the rank minimization algorithms to be applicable to Hankel matrices. Instead, some approaches [12] use the Gramian (Gram) Matrix for their purposes, which is defined by:

$$G_y = H_y^T H_y$$

By definition, the Gram matrix will be PSD. Let’s take the eigendecomposition of $H$, for instance, and apply it to $G$:

$$G = H^T H = (U_H \Lambda_H U_H^{-1})^T (U_H \Lambda_H U_H^{-1}) = U_H \Lambda_H \Lambda_H U_H^{-1} = U_H \Lambda_H' U_H^{-1}$$

where $U_H$ is the square $n \times n$ matrix whose $i$th column is the eigenvector $u_i$ of $H$, and $\Lambda_H$ is the diagonal matrix whose diagonal elements are the corresponding eigenvalues, $\Lambda_{H,ii} = \lambda_i$. Moreover, $\Lambda_H' = \Lambda_H \Lambda_H$, so $\Lambda_H'$ will be the diagonal matrix containing the square of each eigenvalue of $H$. We can also see that $U_H \Lambda_H' U_H^{-1}$ is also the eigendecomposition of $G$ and, thus, $\Lambda_H' = \Lambda_G$. To sum up, this means that each element of $\Lambda_G$ will be positive or 0, which makes $G$, PSD. Also, $\lambda_{G,i} = \lambda_{H,i}^2$.

Hence, the Gram matrix of a time sequence will have the same rank as its Hankel matrix, with the difference that the Gram matrix will be PSD.

2.2.2 Rank minimization strategies

As said, further in this document some of the presented approaches will require the usage of a rank minimization strategy, in order to embed in a low order manifold our input time sequences. For this reason, we will now introduce the main issues with rank minimization and how do we face them.

Rank minimization problems are generically NP-hard. While it’s true that deep learning approaches have been able to deal with high complexity and non-convexity, which will also be explored here, traditional machine learning and optimization strategies may rely on understanding how does a problem behave and what complexity does it have. We also believe that relaxing a problem to a lower complexity will most surely also help deep learning approaches to reach a solution faster and more accurately. For this reason, we explore convex relaxations to the rank function.

Let’s put $H$ as the matrix whose rank we want to minimize. This is equivalent to saying that we want to minimize the cardinality of the vector $\lambda_H$, that contains the eigenvalues of $H$, or the vector $\sigma_H$, containing its singular values. This is:
Instead, a first relaxation would be the nuclear norm of $H$, $\|H\|_*$. Minimizing the nuclear norm is equivalent to minimizing $\|\sigma_H\|_1 = \sum_{\forall i} \sigma_{H,i}$. It is important to highlight that it is not equivalent to minimizing the sum of all eigenvalues, as they could also be negative. This may lead to computation issues given that the SVD must be computed in order to obtain the singular values, and this operation has complexity $O(N^3)$.

To avoid such complexity issues, the rank can also be approximated by the 2-norm of $H$, $\|H\|_2^2$ or the nuclear norm of the Gram matrix $G$, $\|G\|_*$. Similarly, this is equivalent to $\|\sigma_H\|_2^2 = \sum_{\forall i} \sigma_{H,i}^2 = \sum_{\forall i} \sigma_{G,i}$. Given the fact that the eigendecomposition of the Gram matrix is equivalent to the SVD (as $G$ is PSD), we have that:

$$\text{Trace}(G) = \sum_{\forall i} \lambda_{G,i} = \sum_{\forall i} \sigma_{G,i}$$

(2.17)

where $\sigma_{G,i}$ is the $i_{th}$ singular value of $G$ and $\lambda_{G,i}$, the $i_{th}$ eigenvalue. Therefore, the nuclear norm of $G$ is equivalent to its trace, which means that it doesn’t require the computation of an SVD decomposition.

A third relaxation has been regarded, which is the log det $(G + \delta I)$, where the constant $\delta$ is a small regularization constant.

However, minimizing this expression is not trivial. To do so, a heuristic has been proposed in [16], which is based on using the mentioned surrogate of the rank, which is non-convex but it is solved locally via iteratively solving a sequence of convex problems. In the next subsection (2.2.3), we define the solution.

Note that in figure 2.3, we can see how the different relaxations that have been described resemble the rank function in dimension 1.

Figure 2.3: Different relaxations of the rank function. Relaxation by absolute value $|\cdot|$, $\ell2$ norm or $\log \det(\cdot)$.
2.2.3 Re-weighted Trace Heuristic

The Re-weighted Trace Heuristic is an evolution to other heuristics that intend to minimize the mentioned nuclear norm of a matrix. In this case, to reduce the rank of the solution further, the objective is weighted at each iteration of the heuristic.

There is a general formulation of this solution, but for better understanding, we will present the particularization to our case. We want to minimize the rank of either $H_y$ or $G_y$, as they represent the order of the LTI system that models the underlying sequence. Furthermore, if we are working with PSD matrices the following is true for a matrix $M$:

\[ \|M\|_* = \text{Trace}(M) \]  \hspace{1cm} (2.18)

subject to \( M \geq 0 \)

The proposed heuristic, as introduced, will globally solve the following expression:

\[ \text{minimize} \quad \log \det(M + \delta I) \] \hspace{1cm} (2.19)

subject to \( M \geq 0 \)

As $M$ is imposed to be PSD, [16] shows that 2.19 can be solved locally by iterative linearization of the objective, so that the $k_{th}$ step of the algorithm will solve:

\[ \text{minimize} \quad \text{Trace}((M^K + \delta I)^{-1}M) \] \hspace{1cm} (2.20)

subject to \( M \geq 0 \)

to obtain $M^{K+1}$. This expression is the RTH that has been proposed. It is also recommended to initialize the heuristic with $M^0 = I$, therefore the first iteration will basically minimize $\text{Trace}(M)$.

To finalize, if we particularize to our problem, we have to choose whether we will minimize the rank of $H_y$ or $G_y$, but we choose $G_y$ as the heuristic requires $M$ to be PSD.
2.3 DYA\textquotesingle s limitations

Videos of real scenarios are generally very complex. There are different objects with different trajectories, occlusions and disocclusions, non-rigid objects, objects appearing and disappearing of the scenario, and a lot of uncertainty associated.

These behaviours have led recent approaches to model the associated uncertainty and avoiding the generation of raw pixels. Instead some of them have focused on the prediction of motion and the spatial transformation to generate future frames.

In the specific case of DYA, we have seen a number of limitations that penalize the prediction accuracy in the previously described cases, but not only in those. There are limitations that are inherent to the architecture of DYA that have also been approached.

- Lack of spatial context: DYA and one of its main variations, Kalman DYA

Kalman DYA (K-DYA) (which will be later briefly described) uses what is called an \textit{Eulerian} point of view. This means that it predicts the next value in location \((x, y)\), based on the previous values in that same location \((x, y)\), resulting in a null sense of the spatial context.

Hence, modelling time sequences associated to a specific pixel location of RGB frames as linear systems often leads to error.

The uncertainty associated to real video generates non-linearity that needs to be addressed. In real RGB images, textures within moving objects make very difficult to associate a pixel location to the output of a casual LTI system for a significant period of time.

This problems are addressed in sections 2.3, 3.2 and 3.3. in which it is described how keeping the same core architecture for DYA, we can mitigate part of the drawbacks associated to an \textit{Eulerian} point of view.

In the first place, the presented solution for DYA uses OF instead of RGB data as input. OF models the motion of the objects, therefore the new inputs are smoother, simpler and present local uniformity. Having uniformity within rigid moving objects, for instance, mitigates the error linked to the \textit{Eulerian} point of view for that object. Moreover, motion of objects in real scenarios can be assumed to have linear dynamics in a broad range of situations.

For DYA, OFs as input where obtained by pre-processing the raw images with a deterministic method.

As said, this motion-based approach for prediction using OFs was the main case explored in the original DYA paper. However, there is still a very frequent source of non-linearity that lowers the performance which is mostly related to the occlusions and dis-occlusions (either foreground-background or foreground-foreground) and the emergence of new information in the frame. The inability of the modelled LTI systems to predict the appearance of the new sources of information in their receptive field (one pixel) generates a great amount of error. Even when working with OFs.
An intuitive solution to this problem would be to take into account the spatial context, so that the surrounding of a pixel location was regarded when modelling the time sequence for that specific location. For this reason, solutions for spatial codification, spatial similarity enforcement or a switch in the point of view have been proposed.

An illustration of how intuitively can the spatial context be exploited is shown in figure 2.4, where a simple change in the point of view can linearize the given input sequence dynamics.

- Can’t generalize well when there is difference in scaling between time sequences:

When predicting with DYAN, as described in section 2.1, we must use the algorithm FISTA to solve equation 2.8 and obtain the \( c \) vectors that represent the weights that each pole will have when modelling each time sequence for a set of frames. FISTA will therefore be performed once for each pixel location (\( HW \) times, where \( H \) is height and \( W \) is width of a frame). We also observe in equation 2.8 that the sparsity factor \( \| \cdot \|_1 \) is multiplied by a weight \( \lambda \) that indicates how much importance we give to the sparsity of the vectors \( c \). \( \lambda \) is hence an hyperparameter that we should find and its used for every single computation of FISTA.
The problem is, that our sequences should have a similar norm (or energy) so that a single $\lambda$ allows us to generalize properly. Note that we have relaxed the sparsity factor from $\|\cdot\|_0$ to $\|\cdot\|_1$ and thus, to the sum of all components in vector $\mathbf{c}$. This means that if one of the coefficients of $\mathbf{c}$ is specially large, FISTA will understand that $\mathbf{c}$ is sparse.

For instance, if we have several sequences one of which ($x_{\text{biased}}$) has a very large bias (the sequence is multiplied by a large constant), the pole located in the unit of the real axes (which represents a constant) will have an associated $c \in \mathbf{c}$ with a very high value.

Therefore, if we have tuned $\lambda$ to be adapted to the rest of sequences with similar norms, FISTA will learn (erroniously) that the vector sparsity is higher than it actually is, and this is likely to repercute in the modelling accuracy.

To prevent this to happen there is a simple solution. We standardize (normalize) the time sequences independently, subtracting the mean and dividing by the standard deviation (regularized by a small constant $\epsilon$):

$$\hat{y} = \frac{y - \mu_y}{\sigma_y + \epsilon}$$

where $y$ is the original vectorized sequence, $\mu_y$ the mean and $\sigma_y$ the standard deviation of that sequence. Hence, $\hat{y}$ is the normalized sequence.

To obtain the original sequence we save the mean and variance and do the inverse process, disregarding $\epsilon$.

- **DYAN can’t hold long-term memory:**

  For each frame prediction, DYAN is identifying a system to model each of the time sequences, with a fixed time span. If a new and consecutive frame wants to be predicted, DYAN will re-model the time sequences from scratch, using the previously predicted frame but with the same time span $T$. This means that the architecture can only hold a fixed memory, which corresponds to the time length used for prediction. DYAN could also predict more than one frame in the future extending the dictionary further, but the experiments have shown that the one-at-a-time prediction has better performance, as the models are only approximations to the sequences, and tend to have worse fitting to the data as the time expands.

- **FISTA is costly**

  In extension to the previous point, let’s assume we are in a set-up in which we predict one frame (first iteration), compare it to the ground truth and then predict the next one (second iteration) with the ground truth. With the regular setting of DYAN, we would model all sequences of the first iteration independently and using FISTA. In the second iteration, we would disregard the previously fitted models and recompute new models for all sequences, also using FISTA.
2.3 DYAN’s limitations

This is clearly inefficient, given that some of the time sequences for the second iteration could still be modelled by keeping the model computed in the first iteration, but still it is recomputed.

- DYAN is slow detecting changes in dynamics

Finally, because of the previously mentioned non-linearity inherent to natural video, the appropriate LTI system associated to each sequence will suffer switches (e.g. the case in which we are modelling a time sequence corresponding to a pixel in the background and it is suddenly replaced by the occlusion of an object, that should be modelled by a different LTI system) DYAN doesn’t have a mechanism to detect those switches, thus, while such switch happens within the input time sequence, the prediction will be wrong.

The last three points, have been approached by an architecture presented (together with other improvements that we will also discuss) in [2]. which is called K-DYAN and uses the widely known Kalman filter together with the regular architecture of DYAN to not only fit a model to the given time sequences, but also detect switches and update the models with every new measurement.

Using K-DYAN instead of DYAN results in:

- Holding longer term memory. If the memory resources allow it, saving the state for each time sequence (understanding state as the system that models it) and the parameters of the Kalman filter, we virtually remember as much information as needed for the predictions, which corresponds to the current states.

- Changes in dynamics are detected by the Kalman filter when it classifies a new measurement as an outlier to the current state. Therefore, FISTA is only performed when new dynamics are detected, otherwise the states are only updated.

This document shows the work done for facing the first of the exposed limitations. Obtaining information from the spatial distribution for better prediction. K-DYAN was also developed in Robust Systems lab but will not be further detailed as it was not the scope of this project. However, some of the shown results for the approaches presented will be in combination to K-DYAN.
Chapter 3

Models

In this chapter, we present the different models or approaches that have been developed and tested throughout this work. We structure this part of the document in the following way. Each one of the models will be a separated section divided in three parts. Intuition; in which we explain the motivation or idea behind the approach, Solution; the description of the model, its architecture and modules, the training strategy and why it fulfills the intuition, and the Implementation and Experiments; giving a comprehensive description of how the model has been implemented and which experiments have been carried out, for which scenarios and what results does it produce.

In this last section, details regarding the dataset, hardware or software that will be shared by other models are explained only once, and made reference further on.

3.1 Spatiotemporal DYAN

Exploiting space dependencies for better prediction with DYAN can be also understood as using the space in order to linearize the dynamics of the inputs, as finding linear dynamics is what DYAN does best.

The first approach to use the spatial information is naive in some sense. It exploits space dependencies in defined neighborhoods of a frame while it doesn’t follow a clear intuition for the usage of space in order to linearize the input.

3.1.1 Intuition

DYAN can also be understood as an architecture for representing input sequences as the output LTI systems. We mostly use it to model time sequences, but space sequences can also be modelled by it. The first aim, thus, will be to obtain a sense of the space dependencies by encoding them using DYAN, with the hope that this encoding has better properties than the raw data in terms of dynamics.

Following, the time predictions will again be performed by the original DYAN, although in this new encoded domain.

Again, there is no clear intuition of why a spatial encoding will necessarily have more
3.1 Spatiotemporal DYAN

Figure 3.1: Architecture of ST-DYAN. It is a nested autoencoder. The spatial encoder provides the new inputs (coefficients $c$) to the temporal DYAN, which predicts. The spatial decoder maps back the coefficients to the original domain.

linear dynamics than the original data, despite of the fact that the features will now have information of the non-linearity (associated to occlusions, for instance) of the spatial chunk that has been encoded.

3.1.2 Solution

The architecture of the ST-DYAN is a nested autoencoder. Figure 3.1 shows how the spatial and temporal encoders and decoders are concatenated.

Spatially, however, the encoding has some added complexity. Each frame is decomposed in a number of chunks (the neighborhoods that we will independently encode) and each frame chunk is horizontally and vertically encoded, by rows and by columns respectively.

Following the original formulation of DYAN the new encoding is given by:

$$c_{\text{temp},(l:l+L)}^* = \min_{c_{\text{temp},(l:l+L)}} \frac{1}{2} \left\| c(1:T)_{\text{spa},(l:l+L)} - D^{(T)}_{\text{temp}} c_{\text{temp},(l:l+L)} \right\|_2^2 + \lambda \left\| c_{\text{temp},(l:l+L)} \right\|_1$$

(3.1)

$$c(t)_{\text{spa},(l:l+L)}^* = \min_{c(t)_{\text{spa},(l:l+L)}} \frac{1}{2} \left\| y(t)_{(l:l+L)} - D^{(L)}_{\text{spa}} c(t)_{\text{spa},(l:l+L)} \right\|_2^2 + \lambda \left\| c_s,(l:l+L) \right\|_1$$

(3.2)

And the decoding, both temporally and spatially is described by:

$$c(1:T+1)_{\text{spa},(l:l+L)}^* = D^{(T+1)}_{\text{temp}} c_{\text{temp},(l:l+L)}$$

(3.3)

$$y(1:T+1)_{(l:l+L)} = \frac{D^{(L)}_{\text{spa} (\text{horz})} c(1:T+1)_{\text{spa} (\text{horz}), (l:l+L)} + D^{(L)}_{\text{spa} (\text{vert})} c(1:T+1)_{\text{spa} (\text{vert}), (l:l+L)}}{2}$$

(3.4)
3.1 Spatiotemporal DYAN

where the optimal set of coefficients is $c^*$ and the subindices for the different components represent; whether it represents space spa or time temp, if it is spatial whether it is used for the horizontal encoding horz or the vertical vert and the spatial portion $(l : l + L)$ of the frame. The superindex $(L)$ or $(T)$ are used to represent which length does the spatial or temporal dictionaries (respectively) have. The notation has been slightly modified in order to simplify the expressions for better understanding. Note that to obtain the final representation of a chunk, the horizontal and vertical decodings are equally weighted in a linear combination. Also, a different dictionary is learned for time, horizontal space and vertical space.

3.1.3 Implementation and Experiments

In this subsection, we evaluate the proposed method on widely used real-world dataset:

**KITTI and Caltech Datasets:** Car-mounted camera video datasets KITTI [17], with 57 recording sessions from the City, Residential and Road categories for training and Caltech Pedestrian dataset [18] testing partition consisting of 66 video sequences for testing. In both cases the frames have been center cropped and resized to $128 \times 160$ and normalized the pixel values between 0 and 1. This setup will also be used for architectures in sections 3.2 and 3.3, and will be referred to as KITTI-Caltech experiment setup.

For these experiment, the network was trained on 1 NVIDIA TITAN XP GPU. In some cases 2 GPUs have been used, using one GPU for each of the optical flow channels. This hardware will be used for the rest of experiments throughout the project.

The input optical flows were computed using the “coarse to fine optical flow” method, which is not a deep flow learning method and therefore doesn’t have parameters to train. It has been provided by the authors of [19], and will be used to compute the OFs of each experiment that requires them.

For this architecture specifically, we have evaluated it qualitatively only in the KITTI-Caltech experiment setup. Qualitative results have been enough to assess that the usage of spatial context in this case doesn’t produce the desired results. We show the ground truth OFs together with the predictions generated by the trained architecture for a specific example in figure 3.2a, after training for 20 epochs and with $\lambda = 0.2$ for both time and space (in both equations 3.1), learning rate $LR = 1.5e^{-3}$ and 40 poles at each quadrant of the unit circle plus one in the unity of the real axis representing a constant system. We input $T = 9$ OFs, generated from $F = T + 1$ frames and predict the $10_{th}$ OF from which the performance is evaluated. The OFs are applied to the original frames for warping in testing. It is relevant to highlight that for DYNABased architectures, training for 20 epochs is usually enough, meaning that the losses stabilize after around 15 epochs. This is due to the little amount of parameters that are trainable,
in this case, the position of the 40 poles for each DYAN encoder.

Results show how the results get better throughout the training, as expected, but produce blurriness and what can be called the \textit{ghost effect} (highlighted in red). This effect happens when there is a trace of the past information, coming from the first frames of the input sequence, that are present in the prediction. This trace shows that the occlusions and disocclusion can’t be avoided by spatially codifying the input OFs using DYAN.

![Figure 3.2: Qualitative results for ST-DYAN in the following setup: each axis has been split in 4 chunks, the weight of the sparsity term is $\lambda = 0.2$ and the dictionary is composed by 40 poles (mirrored horizontally and vertically in the unit circle) plus a pole in the unity of the real axis.](image)

(a) Qualitative results given by ST-DYAN after 20 epochs. From the top down, the last of the input OFs, the ground truth expected prediction and the two predictions in the bottom (left after 1 epoch, right after 20 epochs). In red we highlight the “ghost” effect in our prediction.

(b) Poles of the dictionaries of ST-DYAN after 20 epochs. From left to right, we represent the vertical and horizontal (spatial) dictionaries in the top and in the bottom, the temporal dictionary.

Figure 3.2: Qualitative results for ST-DYAN in the following setup: each axis has been split in 4 chunks, the weight of the sparsity term is $\lambda = 0.2$ and the dictionary is composed by 40 poles (mirrored horizontally and vertically in the unit circle) plus a pole in the unity of the real axis.

In figure 3.2b, we show the dictionaries learned in the same setup as before, also after 20 epochs. The learned poles show not only that the temporal and spatial dictionaries converge to different solutions because of the nature of the inputs they get (spatial layer encodes OFs and temporal layer encodes sparse codes), but also that there is a remarkable difference between the dictionaries learned for horizontal and vertical spatial encoding. However, it is difficult to extract conclusions from the differences between dictionaries.

Although this approach doesn’t seem to accomplish successfully the given task, the results are similar to the original DYAN. This indicates that predictions over codes is possible and therefore opens the possibility to explore tasks such as prediction of compressed data.
3.2 Group Lasso DYAN

In this section we describe an approach that modifies the objective function of the problem solved by DYAN.

3.2.1 Intuition

The objective function (equation 2.8) of the problem solved by FISTA establishes a balanced objective between sparsifying the vector of coefficients $c$ and correctly fitting the measurements (input data) to an LTI system. The latest is necessary as it is the ultimate objective of DYAN, model the data by means of a system. However, the sparsity factor is used as a way to avoid overfitting; if we have high sparsity, fewer poles will be used for each sequence to be modelled and therefore, the order of these models will be as low as expected.

In this section, we explore the outcome of changing the focus and force the $c$ vectors to be similar within a spatial neighborhood, instead of sparse. The intuition behind it is simple: OFs tend to be locally smooth (or constant), as objects in the foreground move uniformly and the background stays still. In the borders of objects, the largest source of non-linearity, we expect the neighbouring pixels to be influenced by each other, and again give a notion of spatial context to the architecture.

3.2.2 Solution

To enforce neighborhoods to be modelled by similar LTI systems, we must enforce them to choose similar $c$ vectors when performing FISTA. This can be done by modifying the objective function and use the Group Lasso regression analysis method instead, adapted to our specific case.

Consequently, the analogous of equation 2.8 will now be given by:

$$c^* = \sum_{k=0}^{HW-1} \left( \min_{c} \frac{1}{2} \left\| y_{(1:T)} - \sum_{m=p_0}^{p_0+M-1} D^{(T)} c^{m,k} \right\|_2^2 + \lambda \sum_{m=p_0}^{p_0+M-1} \left\| c^{m,k} \right\|_2 \right)$$

(3.5)

where $p_0$ is the initial position of a spatial neighborhood of size $M$ to which the pixel $k$ belongs. The subindices indicate time spans and the superindices indicate space location or in the case of the dictionaries (in parentheses), their number of rows (namely, the time extension of the modelled systems). In this case, the obtained $c^*$ is a matrix with size $N$ (the number of coefficients per $c$ vector) $\times$ $HW$ (the number of sequences or pixels for each frame). The solution is therefore computed for each chosen neighborhood, although the dictionary $D$ will be the same for every neighborhood within a frame. Analogously to 2.11, the decoding and prediction is now given by:

$$y_{(1:T+1)} = D^{(T+1)} c^*$$

(3.6)
3.2 Group Lasso DYAN

By substituting the sparsity factor ($\ell_1$) by the $\ell_2$ between codes of the same neighborhood we are enforcing them to be similar to each other instead of sparse. The architecture of the solution, however, is the exact same as for original DYAN.

3.2.3 Implementation and Experiments

We evaluate the proposed method in the same setup as the previous section 3.1. KITTI-Caltech experimental setup, with the same computation for the OFs and the same hardware (GPU’s) used.

Also similarly to section 3.1, for the experiments we evaluate the architecture qualitatively and ascertain that the proposed solution doesn’t achieve its initial goal; to avoid non-linearity and errors linked to the boundaries of the moving foreground objects.

The input is again a set of $T = 9$ OFs and we predict the $10_{th}$. After prediction we use the generated OF to warp the last ground truth input frame and obtain the predicted RGB frame. In figure 3.3 we display the qualitative results for several consecutive predictions in a rolling window fashion (one prediction at a time).

We train for 20 epochs, with $\lambda = 0.2$ and the dictionary is generated from 40 poles mirrored in the four quadrants of the unit circle plus one fixed pole in the unity of the real axis, for modelling biases. Results show that the lagging presented by DYAN is still present in our predictions (red slashed line), and some artifacts appear in the warped RGB predicted frames (red doted circle). These results are very similar to the ones achieved by the original DYAN architecture (see Experiments section in [1]), and they don’t give any sign of improvement in the direction of avoiding non-linearity.

![Figure 3.3: Qualitative results given by GL-DYAN after 20 epochs. Similarly to the original DYAN architecture, we can see lagging (red/white lines) in both OFs and RGB frames and artifacts (red circle) in the warped predictions. The weight of the sparsity term is $\lambda = 0.2$ and the dictionary is composed by 40 poles (mirrored horizontally and vertically in the unit circle) plus a pole in the unity of the real axis.](image-url)
In figure 3.4 we show the learned poles after 20 epochs. In this figure particularly we highlight the most used poles for modelling the time sequences of all the testing videos in set "10V011" of the Caltech dataset, which is a difficult example given the amount of motion of the objects. We can observe that clearly the poles around the unit in the real axis are used more often. They represent the impulse responses close to the constant, with a low impulse decay or slightly unstable. There are also some activated conjugated pole pairs in either close to the imaginary axis or to the $-1$ value of the real axis, both showing an oscillating behaviour that might represent the modelling of the noise, or in case of fast decay (values closer to the origin) the abrupt changes given by system switches and non-linearity.

Once more, quantitative results are not computed given that the qualitative results show that the primary objective is not fulfilled.
3.3 Warping DYAN: Aligning optical flows

In this section we address a new approach to give DYAN a sense of the spatial context. However, in this case, we exploit properties of natural motion and estimate a modification in the input data that will naturally tend to have linear dynamics. Therefore, we have to make assumptions.

3.3.1 Intuition

For DYAN to work ideally, the time sequences input to the pipeline would have to be modelled properly by the output of a single LTI system. Assuming that currently, the architecture is unable to detect whether there’s a dynamics switch within a time sequence or not and will assume there isn’t one, it makes sense to try to approach the problem by providing sequences that can be modelled by LTI systems but with higher probability of not having a switch.

In section 2.3 we already introduce an idea that will be used here. The motion of the objects in the foreground tend to have linear dynamics in natural videos, and the background will remain still or move homogeneously. For instance, a car moving will have constant velocity or maybe will accelerate but very rarely would it have a non-linearity such as a discrete change in the motion value. This is, after all, an assumption. However, experiments will show numerically that the intuition behind it is, to some extend, valid.

DYAN is currently predicting OFs to finally obtain the RGB predictions by spatially transforming a ground truth RGB frame, assuming that the information will be the same or very similar but in different locations. However, the OFs have been computed from consecutive frames, as it is commonly done because the accuracy is higher than if it was computed between frames more temporally spaced. This results in the same nature of sources of non-linearity that have been already discussed, located in the boundaries of objects or frames, as the motion fields associated with a certain object are also displaced when the object does.

This leads back to the concept of Eulerian point of view, also introduced in section 2.3. It is a key concept, as the aim of this approach is to somehow give a new point of view.

Both, DYAN and K-DYAN use an Eulerian point of view: they predict the optical flow at a pixel location based on its past values at the same pixel location (figure 3.5(b)).

Note that an Eulerian observer will not see changes in the dynamics of the optical flow at a pixel \((x, y)\) while it remains an interior point of a rigid object moving slowly relative to its size (i.e. pixel \((x_1, y_1)\) on the truck in figure 3.6). However, if the object moves away from the pixel (i.e. pixel \((x_2, y_2)\)), then the Eulerian observer will see the dynamics change (i.e. to the dynamics of the wagon).

The main idea is to work with the Lagrangian evolution of the optical flow instead of
3.3 Warping DYAN: Aligning optical flows

Figure 3.5: A set of overlapping frames. (a) Trajectories of two objects. (b) Eulerian point of view looks at changes at a fixed location. (c) Lagrangian point of view looks at changes along trajectories. (d) Recursive warping aligns all the optical flow frames by placing the optical flow of corresponding pixels at their final location [2].

Figure 3.6: An Eulerian observer sees changes in optical flow at pixels (i.e $(x_2, y_2)$) on occluding boundaries ([2]).

its Eulerian changes. That is, we propose to predict the optical flow at each pixel location based on the past values along the temporal trajectory of its corresponding pixels in the previous frames (figure 3.5(c)).

However, instead of tracking the pixels, we will warp (understanding warping as the application of a spatial transformation defined by an OF) the optical flow frames so that the Lagrangian changes can be processed using an Eulerian point of view. That is, we propose incorporating a (parameter free) warping block at the front end of DYAN, which uses each of the input optical flow frames to recursively perform backward warping and align themselves. In this way, the optical flow of corresponding pixels across frames are moved to their final spatial coordinates (figure 3.5(d)).

3.3.2 Solution and architecture

Following the intuition in subsection 3.3.1, our aim will be to align the input OFs in the position where the objects will be located in the predicted OF. This is done by recursively warping the OF as if they were images. Hence, we spatially transform them.
3.3 Warping DYN: Aligning optical flows

The recursive warping is given by:

\[
\check{y}_t = W_T(W_{T-1}(\ldots(W_{t+1}(W_t(y_t)))))
\]  

(3.7)

where \( W_t(\cdot) \) is the operation of warping using the OF of time \( t \). \( y_t \) is the input frame in time \( t \), which in this case corresponds to the OF in time \( t \).

In figure , there is a graphical representation of the output of this method compared to the original inputs for a set of synthetic OFs, and the process of recursively warping. In this example, we only represent horizontal motion (indicated by the numbers). In red, the area that would be a cause of error for this synthetic example. Due to issues related to the implementation, warping an OF by itself is lossy and it creates error in the borders. However, the error is low compared to not using this method.

Figure 3.7: W-DYAN uses recursive backward optical flow warping to align corresponding pixels. In the figure, the first three optical flows in the left column are aligned (right most column) in order to predict the fourth optical flow of4. Colors indicate objects, while the numbers indicate the horizontal optical flow of the object. As the diagram shows, temporal sequences of Lagrangian optical flow are smoother than the corresponding Eulerian optical flow. In this synthetic example different cases are shown: constant velocity, accelerations, occlusions and oscillation. Note that only horizontal motion is represented, namely, the first channel of an OF.
3.3.3 Implementation and Experiments

In this subsection, we will reproduce the results presented in [2], which is also part of this project. The following experiments were carried out in the Robust Systems Laboratory at Northeastern University by the first author of [2], Wenqian Liu and the second, myself.

We will therefore include quantitative results which are product of combining W-DYAN with K-DYAN. (Kalman-Warping DYAN (KW-DYAN)), although architectures involving Kalman won’t be detailed as they are out of the scope of the document. For the qualitative results, we will focus on W-DYAN alone.

Firstly, we introduce the metrics by which we will compare the results to the SOTA: We adopt the commonly used numerical measurements: mean Peak Signal-to-Noise Ratio (PSNR) [10], MSE, and Structural Similarity Index Measure (SSIM) [20], to evaluate performance at the pixel level. Additionally, we report Learned Perceptual Image Patch Similarity (LPIPS) [21] distances, since it has been shown to be a good perceptual metric, and we introduce a new metric, the mean of the maximum optical flow (MMF) between ground truth and predictions, to measure prediction lagging.

Quantitatively, the higher PSNR/SSIM and the lower the MSE/LPIPS/MMF, the better the performance.

For the KITTI-Caltech experimental setup described in the previous sections, we input $T = 9$ OFs, generated from $F = T + 1$ frames and predict the $10_{th}$ OF, with $\lambda = 0.01$ in FISTA and 40 poles at each quadrant of the unit circle plus one in the unity of the real axis representing a constant system output. We input $T = 9$ OFs, generated from $F = T + 1$ frames and predict the $10_{th}$ OF from which the performance is evaluated.

Firstly, we evaluate the qualitative results (figure 3.8), for only W-DYAN in comparison to original DYAN.

There is a clear correction of the lagging that can be seen in the lower-left corner of the amplified section (the wheel of the car), although there is still some lagging with respect to the ground truth and therefore, it’s marked in red. The green arrow shows how this approach corrects small artifacts or malformations that appeared in DYAN’s predictions.

In order to fully compare to the SOTA approaches (including DYAN) we also test the new architecture in a second dataset.

**UCF101 Dataset:** We also ran experiments on natural human action videos from the UCF-101 dataset [22], which includes more complex motion than street views. UCF-101 contains 101 action categories with a total of 13,320 video sequences. Three different splits are provided by the dataset. Following existing state-of-art approaches, we chose the first split to train and test. Our models were trained with $F = 4$ input frames, $T = 3$ optical flow files. During fine tuning, FISTA was ran with $\lambda = 0.1$ and 100 maximum iterations.

For testing, we followed [10] [23] to set up the testing experiments. 378 videos provided by [10] were sampled along with their evaluation script for testing. PSNR and SSIM were measured using the motion masks provided by [23] in order to evaluate only on regions with obvious movements.
3.3 Warping DYAN: Aligning optical flows

Figure 3.8: Qualitative example predicting 5 future frames for the Caltech dataset set 10V011. Top row shows the ground truth frames. The right column shows details from the fifth predictions. Circled in red: incorrect position of the front tire and orientation of the window frame of the car; Green arrows: sharper roof line, correct orientation of the window frame ([2]).

Qualitative results in 3.9, show the same kind of improvement seen in KITTI-Caltech setup. In green we see both a correction of the lagging and artifacts of the predictions of DYAN.

Figure 3.9: UCF101 qualitative example. W-DYAN makes sharp predictions with less artifacts than DYAN’s output. The second row shows details from the predictions. Circled in red: incorrect predictions (artifacts or lagging); Circled in green: improvement with respect to DYAN’s output. (modified from [2]).

After testing the networks and assess them quantitatively, we report the results in comparison to the SOTA in table 3.1.

The modified DYAN networks, in this case K-DYAN, W-DYAN and KW-DYAN, have the best performance for Caltech, in all metrics, and for UCF in all metrics except LPIPS. If we evaluate the sole contribution of W-DYAN, it beats the SOTA (excluding the other X-DYAN architectures) only in Caltech dataset and with the exception of LPIPS. For
Table 3.1: Quantitative results for single frame prediction on Caltech and UCF101 datasets. Except for DYAN's scores (for which we ran the code provided by the authors), we adopted the scores from [3] and [4]. The modified DYAN networks, W-DYAN, K-DYAN, and KW-DYAN yield the best performance on most metrics, with the smallest model size ([2]).

UCF101, the results are not as good as expected. This could be given by the fact that UCF dataset itself has errors regarding repeated frames, which generate OFs with 0 value.

This problem affects both the system identification task shared by all DYAN architectures, and the recursive warping task of W-DYAN. This problem has been previously reported in [10].

Both, W-DYAN and K-DYAN address the problem of lagging, but in different ways. W-DYAN tries to avoid changes in dynamics, while K-DYAN seeks to detect them and forget previous inputs. Thus, it is not surprising that they have similar performance.
3.4 Low Order Dynamics Autoencoders LODAE

The previous approaches for enhancing DYAN in sections 3.1 and 3.3, had in common that they where trying to pre-process the inputs in order to make them fit to DYAN’s pipeline for a better prediction. In other words, and equivalently, we were trying to manipulate the original OF data in order to linearize its underlying dynamics and avoid system switches and non-linearity. In section 3.2, the approach had a similar objective, but was carried out by modifying the objective problem.

However, they rely in assumptions that might only be right in certain natures of video, such as the nature of motion in real scenarios or the apparent rigidness and uniformity of objects. Moreover, for better performance they need to work with OF data instead of RGB, which at the same time introduces error linked to the accuracy of such OFs’ computation.

Thus, this section presents two architectures that make use of the dynamics theory explained in subsection 2.1 and specially subsection 2.2 to provide a set of features that fit a predictor such as DYAN. These architectures intend to generalize a solution for any kind of video input (with an underlying spatiotemporal structure, of course), avoiding the usage of previously computed OFs.

Ultimately, LODAEs intends to learn a mapping from the raw input data to a manifold in which the underlying dynamics can be modelled as the output of an LTI system, and the inverse mapping to the original domain.
3.4 Low Order Dynamics Autoencoders LODAE

3.4.1 Re-weighted Trace Heuristic LODAE RTH-LODAE

In this section, we will focus in the first architecture proposed as a LODAE, which will make special use of the heuristic and principles explained in subsection 2.2.3.

3.4.1.1 Intuition

In subsections 2.2.2 and 2.2.3, we review several strategies for rank minimization and more specifically, the RTH as a heuristic to solve the rank minimization problem with a surrogate of the rank function. In this case the surrogate is $\log \det(\cdot)$ and it is solved locally by the heuristic.

We propose to fit the RTH in a deep learning framework, so that we can exploit the benefits of RTH as a rank minimization method and the well-known ability of deep learning architectures to learn very complex dependencies from data. Furthermore, the objective will be to learn the mapping from an original domain to a manifold in which the underlying dynamics are linear, and being able to generalize it for a set of data with similar distribution.

However, any heuristic is used to solve a specific problem for the given input data, without generalization of any kind. For this reason, in the following subsection it is explained how deep learning and RTH are merged to achieve a generalized solution.

3.4.1.2 Solution

In order to learn the sought mapping to a low order dynamics space, we propose an autoencoder architecture (figure 3.10), with an encoder formed by a convolutional feature extractor followed by a fully connected layer (further developed in subsection 3.4.1.3). The decoder seeks to be a mirrored version of the encoder. It has a fully connected layer followed by a stacked set of transposed convolutional, batch normalization and activation layers. It is important to highlight that we will make reference to the space in the bottleneck, named $z$, as the latent space. The latent space is, therefore, the low order dynamics manifold that we intend to achieve.

To introduce the RTH in such deep learning framework, we start by posing the objective function to solve at each iteration of the heuristic, as a loss function. In equation 2.20 of subsection 2.2.3 there is the formulation of the convex problem that is locally solved by the RTH to achieve the final objective function given in 2.19.

Following the previous expression, the loss function used for linearization purposes will therefore be:

$$L_{lin} = \lambda \text{Trace}((G_{z,(k-1)} + \delta I)^{-1}G_{z,(k)})$$

where $\lambda$ indicates the weight of the linearization loss within the general loss, $G_z = H_z^T H_z$ and subindex $k$ indicates iteration $k_{th}$ of the RTH. $z$ is the latent representation of the data, namely, the data after being transformed by the encoder. $\delta$ is a small regularization
Figure 3.10: Global architecture of RTH-LODAE. It uses the encoder $E_{\theta}$ to map the inputs $y^{(1:T)}$ into the latent space represented by $z$ where the linearization loss is computed by generating the matrices $H_z$ and $G_z$ at each iteration of the heuristic. The buffer represents the memory required to save the $z$ values computed in the previous iteration of the heuristic. The decoder $D_{\phi}$ maps the latent representations back to the original domain, where the reconstruction loss is computed.

constant. Note that as explained in the background section, we use the Gramian matrix $G_z$ instead of the Hankel matrix, because the former is PSD.

As in any generalist autoencoder structure, a reconstruction loss is computed to force the architecture to decode with fidelity to the original data, in a self-supervised way:

$$L_{rec} = \text{MSE}(y_1^i - D_{\phi}(E_{\theta}(y_1^i)))/T$$

(3.9)

Therefore, the general loss will be:

$$L = L_{rec} + L_{lin}$$

(3.10)

From that point, the strategy for training will be the following (represented in figure 3.11):

- Given an input video $y_1^i$ with frames 1 to $T$, $K$ iterations of the RTH will be performed with that same input, while updating by backpropagation the weight $\theta$ and $\phi$ of the encoder and decoder respectively at each iteration. At each iteration $k$ we use $G_z^{(k)}$ (obtained from the latent representations) to compute the linearization loss and store it in a buffer for the next iteration of the heuristic. We also compute the reconstruction loss.

- $G_z^{(k-1)}$ is initialized as the identity $I$ so that the initial expression of the linearization loss ($1_{st}$ iteration) is $L_{lin} \approx \lambda \text{Trace}((G_z^{(k)}))$

- After $K$ iterations, we switch the input to the following instance $y_1^{i+1}$ and reinitialize $G_z^{(k-1)}$ to $I$. 
3.4 Low Order Dynamics Autoencoders LODAE

- Repeat until processing all the training data.

In a certain way, this training strategy creates a small overfitting for each input instance $y_i^i$. However, this happens for every instance, hopefully achieving a generalized solution.

Figure 3.11: Training strategy for RTH-LODAE. We initialize $G_{z, (k-1)}$ to $I$ and compute $K$ iterations with the same instance $i$ of the inputs computing reconstruction and linearization loss at each step and backpropagating from the general loss. Finally, we switch the input to the instance $i + 1$ and reinitialize. We repeat until processing all the training data.

3.4.1.3 Implementation and Experiments

The implementation of both LODAE architectures presented in this document (in subsections 3.4.1 and 3.4.2) share a lot of similarities.

The network is implemented using Pytorch version 1.0. It takes $T$ RGB frames (in this case we dispense with OF) and reconstructs them at the output.

The image encoder (convolutional feature extractor) has $n_{blocks} = \lfloor \log_2 H - 1 \rfloor$ number of layer blocks (with $H=W$ height and with of the squared input frames). Each layer block is composed of a stacked 2D convolutional layer with kernel size 4, stride 2 and padding 1. Except from the first and last blocks, the rest of blocks have a Batch normalization layer and a LeakyReLu activation layer following the convolution. The image
encoder is followed by a fully connected layer with variable size (depending on the experimental case). The decoder will intend to undo the encoder transformation. It starts by a fully connected layer, followed by a convolutional image decoder. Similarly to the encoder, the decoder has the same amount of blocks, but in this case they are composed of a 2D transposed convolutional layer, a batch normalization layer and a ReLu activation layer. The last block, however, doesn’t have batch normalization and is activated by a Tanh function.

The weights of the convolutional and fully connected layers are initialized following a normal distribution with mean 0 and standard deviation 0.02. The batch normalization layers are initialized with mean 1 and standard deviation 0.02.

For the experiments we will use the same dataset also for both the LODAE architectures. Frame prediction will be performed on the Moving MNIST dataset [28].

**Moving MNIST dataset:** Is a synthetic dataset that for most experiments consists of two digits that move independently within a 64 × 64 frame along a video sequence. However, in this case we want to make use of the simplest case to be able to correctly test the architectures. Consequently, the generated frames will have one digit with size 28 × 28 moving in constant velocity through the frame. For training, each video sequence is generated on-the-fly with a randomly sampled digit with a trajectory given by a randomly sampled velocity (multiplied by a pre-defined step length) and angle. A sample image of a trajectory is given by figure 3.12. The test set is a fixed dataset downloaded from [28] consisting of 10,000 sequences.

In order to test quantitatively both LODAE architectures, we will make use of the previously presented MSE, when testing the fidelity of the generated data to the ground truth (namely, testing reconstruction and prediction accuracy). For the task of dynamics linearization, we generate a new metric: MLSV. As we are testing whether the sequences in the latent space can be modelled by LTI systems, we want to know if those sequences can be expressed in an autoregressive representation (equation 2.1) and thus, if their Hankel matrix \( H_k \) will be rank deficient. For MLSV, we compute the SVD of each \( H_k \), and check how close is the smallest (last) singular value to 0. The closest the better. Hence, to have a general metric, once we have the SVD of every Hankel matrix, we average them
through the number of dimensions of the latent space, obtaining the mean value of each singular value and express it as a percentage. MLSV will be the percentage of the mean last singular value with respect to the rest for all $H_{z_i}$ of the obtained latent representations.

For RTH-LODAE in particular, we test only reconstruction and MLSV, as the architecture doesn’t have a prediction module, and this task should be done independently by another architecture, such as DYAN. For the qualitative experiments, we set the latent space dimension to $z_{\text{dim}} = 128$, the batch size to $b_{\text{sz}} = 96$, learning rate $LR = 5e^{-4}$ and the linearization loss weight to $\lambda = 5e^{-3}$. The number of iterations of the RTH for each input instance is $\text{iter} = 5$. We start by testing it qualitatively with the results shown in figure 3.13. We can see how the reconstructions after 200 epochs are acceptable, leaving the evaluation of the model to its performance in lowering MLSV.

Figure 3.13: Qualitative results for RTH-LODAE. The lower row shows the reconstructions by the LODAE architecture, which seem to be good.

Figure 3.14 shows different aspects of the dynamics linearization task, in testing. Firstly, we can evaluate qualitatively how the architecture lowers the dynamics. The heatmaps show for the same network tuning as in 3.13, the last singular value (first row of each map) tends to decrease after several epochs. There are two cases to show the importance of standardization to avoid trivial solutions, explained in section 2.3.

In the top row we can see the performance without standardization. Visually it might suggest that the dynamics are lower, and it is true. However, without any kind of control over the norm of the sequences, the encoder may add biases that are later removed by the decoder, for instance, which would amplify the first singular value and therefore, the percentage of the last singular value would be reduced. The main problem however, is that the encoder may map many of the sequences to 0, as they are perceived of very low order, but they carry no information.

With standardization in the output of the encoder and the reversed process (unstandardization) at the input of the decoder, we avoid trivial solutions by giving the same relevance to every sequence and forcing the encoder and decoder to find a solution that actually exploits the space-time dependencies of data within a video sequence. However, as shown in the figure, the results are not as notorious but more meaningful.

Quantitatively, we have performed a brief ablation study. In case of table 3.2, with
3.4 Low Order Dynamics Autoencoders LODAE

Figure 3.14: Qualitative results for dynamics linearization of RTH-LODAE. Top two heatmaps show the performance on linearization of the latent spaces dynamics without standardization in testing, before and after training. Bottom row shows the same results with standardization at the input of the latent space and unstandardization before decoding.

RTH iterations fixed to 5 and the same learning rate and batch size as in the qualitative experiments, we can see similar reconstruction values, in all cases very low MSE. For linearization purposes values are also very similar for all \( \lambda \) values with lower MLSV the more we increase \( \lambda \) (around 4.5%). MLSV vary considerably when we change the length of the video input sequences. For every reduction of \( T = 2 \) input frames within the sequence, the generated Hankel matrixes (see equation 2.14) will have one less row and column and therefore, one less singular value. This means that the percentage of the last singular values shown as MLSV will also be affected by this variation, which makes MLSV difficultly comparable between different sequence lengths.

In table 3.3 we can see quantitative results for a fixed sequence length of \( T = 9 \), \( \lambda = 5e - 3 \) and the same parameters as before except from the number of iterations of RTH. Results show that the more iterations, the lower the MLSV, but differences are very small (around 0.3%). Reconstruction error is again very low and doesn’t show a clear pattern.

<table>
<thead>
<tr>
<th>MSE Reconstruction</th>
<th>MLSV</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda = 1e - 1 )</td>
<td>( \lambda = 1e - 3 )</td>
</tr>
<tr>
<td>T=9</td>
<td>0.0093</td>
</tr>
<tr>
<td>T=7</td>
<td>0.0042</td>
</tr>
</tbody>
</table>

Table 3.2: Quantitative results and ablation study for single frame prediction on Moving MNIST dataset for RTH-LODAE, with fixed number of iterations and variations in sequence length \( T \) and linearization weight \( \lambda \). We test for the MSE in reconstruction and MLSV. We can see that reconstruction is similar in all cases, but \( \lambda \) influences substantially on the dynamics linearization.
### Table 3.3: Quantitative results and ablation study for single frame prediction on Moving MNIST dataset for RTH-LODAE, with fixed $\lambda$ and $T$ and variations in number of iterations for the RTH. We test for the MSE in reconstruction and MLSV. We can see that reconstruction is similar in all cases, but the number of iterations influences considerably on the dynamics linearization.

<table>
<thead>
<tr>
<th>Iterations</th>
<th>MSE Reconstruction</th>
<th>MLSV</th>
</tr>
</thead>
<tbody>
<tr>
<td>iter=3</td>
<td>0.0051</td>
<td>0.0485</td>
</tr>
<tr>
<td>iter=5</td>
<td>0.0030</td>
<td>0.048</td>
</tr>
<tr>
<td>iter=10</td>
<td>0.0074</td>
<td>0.045</td>
</tr>
</tbody>
</table>
3.4.2 Auto-Regressive LODAE AR-LODAE

3.4.2.1 Intuition

The previous architecture intended (with relative success) to lower the order of the underlying dynamics of the latent space as much as possible. However, this is not exactly our primary objective. Actually, the wanted solution is a manifold in which the time sequences that conform it can be modelled by LTI systems. In other words, we want the last singular value of each Hankel matrix (or alternatively, the Gramian) to be 0 or as close as possible to 0, which means that those matrices are rank deficient.

Following the background provided in subsection 2.2, if the Hankel (or Gramian) matrix is rank deficient, this means that at least one of its columns is a linear combination of the rest of them. This is sufficient to be able to model a time sequence as the output of an LTI model.

Hence, the magnitude of the last and smaller singular value of a Hankel matrix generated from the time sequence $y_1$ can be seen as an indicator of how well can an LTI system model $y_1$.

3.4.2.2 Solution

According to the intuition, the proposed solution intends to find a mapping from the original data to a manifold in which the Hankel matrices associated to the time sequences in the latent space are rank deficient, independently of how low the rank of such matrices is.

To achieve this we use a convolutional feature extractor (same as in subsection 3.4.1) to obtain a first deep representation of the input frames, followed by two branches of sequential fully connected layers. One of the branches will provide the low order dynamics latent space (namely, the low order manifold embedding) and the second branch (with two consecutive fully connected layers) will provide a coefficients vector $a$.

If the network works as expected, such vector $a$ will learn to belong to the null space (kernel) of the Hankel matrices $H_z$ associated to the time sequences in the latent space $z_{1:T}$. Hence, if $H_z$ has a null space it is necessarily rank deficient and it has a vector $a$ such that:

$$a^T H_z^T H_z a = a^T G_z a = 0 \quad (3.11)$$

Therefore, the global scope of the problem solved here will be to find a non-trivial (avoid $0$ vector) null space vector $a$ and an invertible and non-trivial latent representation $z_{1:T}$ which satisfies equation 3.11.

If we assume that $a$ belongs to the null space of a matrix $H_z$ defined by equation 2.14,
the following expression is also true:

\[ \mathbf{z}_{((T+3)/2;(T+1))} \mathbf{a} = 0 \]  

(3.12)

where, \( \mathbf{z}_{((T+3)/2;(T+1))} = \begin{bmatrix} \mathbf{z}_{(T+3)/2}, \ldots, \mathbf{z}_{T-1}, \mathbf{z}_{T}, \mathbf{z}_{T+1} \end{bmatrix} \)

\( \mathbf{a} = [a_0, a_1, \ldots, a_{N-2}, 1]^T \) \text{ with } N = (T + 1)/2

where \( \mathbf{z}_{((T+3)/2;(T+1))} \) is also the last row of \( \mathbf{H}_x \) if we added one more measurement, \( T \) the time horizon of the input sequence and \( N \) the length of vector \( \mathbf{a} \). We avoid \( \mathbf{a} \) being trivial by fixing the last element to 1. Therefore, in case we wanted to predict \( z_{T+1} \) as a linear combination of the previous samples, we can do it by isolating it from the previous equation:

\[ \mathbf{z}_{T+1} = \mathbf{z}_{\text{pred}} = - \sum_{i=0}^{N-2} a_i \mathbf{z}_{T+i-(N-2)} \]  

(3.13)

If this is true, each time sequence in the latent space can be represented as an autoregressive system as in 2.1, and consequently we name the architecture after it.

In order to achieve our task, two different approaches are proposed, which are variations of the same architecture.

In a first approach (see figure 3.15a) similarly to RTH-LODAE, we will have a reconstruction loss \( L_{\text{rec}} \) and a linearization loss \( L_{\text{lin}} \) which will be given by:

\[ L_{\text{lin}} = \lambda \mathbf{a}^T \mathbf{H}_x^T \mathbf{H}_x \mathbf{a} \]  

(3.14)

\[ L_{\text{rec}} = \text{MSE}(\mathbf{y}_{1:T} - \mathbf{D}_\phi(\mathbf{E}_\theta(\mathbf{y}_{1:T}))) / T \]  

(3.15)

\[ L_{\text{total}} = L_{\text{rec}} + L_{\text{lin}} \]  

(3.16)

The \( \text{MSE}(\cdot) \) indicates the Mean Square Error operation or \( \ell^2 \), \( \mathbf{D}_\phi \) stands for the decoder of the latent features with weights \( \phi \) and \( \mathbf{E}_\theta \) the encoder (feature extractor with fully connected layers). The rest of elements have been previously described. With this first approach, we trust the linearization loss to help learn \( \mathbf{a} \) and \( \mathbf{z}_{(1:T)} \) such that the former belongs to the null space of \( \mathbf{H}_x \) and thus \( \mathbf{z}_{(1:T)} \) can be modeled as the output of an LTI system and \( \mathbf{a} \) can be used as a predictor.

In a second instance, we add a predictor branch to the previous architecture as in figure 3.15b. We also add a term to the objective which corresponds to the prediction loss:

\[ L_{\text{pred}} = \text{MSE}(\mathbf{y}_{T+1} - \mathbf{D}_\phi(\mathbf{E}_\theta(\mathbf{z}_{\text{pred}}))) \]  

(3.17)

\[ L_{\text{total}} = L_{\text{rec}} + L_{\text{pred}} + L_{\text{lin}} \]  

(3.18)

where \( \mathbf{z}_{\text{pred}} \) is given by equation 3.13. Note that this means that in the second approach, the term \( \lambda \) weighting the linearization loss will not have the same effect as in the first approach, because a term \( (L_{\text{pred}}) \) has been added to the general loss.
3.4 Low Order Dynamics Autoencoders LODAE

3.4.2.3 Implementation and Experiments

The setup to test AR-LODAE is very similar to the one used for RTH-LODAE. As said in section 3.4.1, the dataset (and training and testing details), hardware, software framework and the internal structure and implementation of the encode and decoder modules are the same. The singularities of the coming experiments will be specified throughout the subsection.

For approach 1, the predictor module shown in figure 3.15b is only used in testing, as for training we only use the losses $L_{\text{rec}}$ and $L_{\text{lin}}$ defined in the previous subsection and shown in figure 3.15a, and hope that these will be enough to obtain good predictions.

In both approaches, for qualitative assessment we use the following scenario: input length $T = 9$ and 1 prediction; latent space ($z$) dimension $z_{\text{dim}} = 128$; batch size of 96 for training and 1 for testing; learning rate $LR = 5e^{-4}$; and dynamics linearization loss weight $\lambda = 1e^{-3}$. The losses are computed as in equations 3.14 and 3.17 for approaches 1 and 2 respectively.

Qualitative results for the first approach in figure 3.16 show the ground truth in the top rows and reconstruction and prediction in the bottom rows of each 6 frame cluster, during training and in testing, and after 20 and 280 epochs. The last column of each cluster shows the predictions (time $t+1$), and the rest are the two last reconstructed frames. Highlighted in red we can see that after only 20 epochs of training, the LODAE hasn’t still learned to reconstruct the digits but it can predict where the information will be located, which in a way means that it has learned the dynamics associated to motion. After several more epochs, the architecture has learned approximately the distribution of the digits’ appearance, but not accurately specially for prediction in testing.

Similarly, we assess the qualitative results for the second approach in figure 3.17, after...
training with the prediction loss $L_{\text{pred}}$ by using the prediction module during training. The results show similar behaviour as the first approach, with the difference that the prediction and even reconstruction has better accuracy in this case.

Figure 3.16: Qualitative results for the first approach of AR-LODAE. It shows 4 clusters (training/testing 20/280 epochs) of 3 ground truth frames (top of cluster) and the reconstructed (bottom left and center of cluster) and predicted (bottom right of cluster) frames. Red circle: highlight good location prediction after few epochs.

Figure 3.17: Qualitative results for the second approach of AR-LODAE. Same disposition as in figure 3.16.

We also observe qualitatively how the dynamics linearization task is done for the first approach in figure 3.18. We highlight the last (smaller) singular value as the ultimate objective is to reduce it, regardless of the rest. From the figures we can assess that after several epochs the last singular value has been lowered considerably, although in some
cases, the other singular values have gained weight (as in the bottom-right case). The second approach shows a very similar behaviour in figure 3.19.

Finally, as the qualitative results seem promising, we evaluate quantitatively the architecture. For this first experiment we use the same values for the hyperparameters as in the qualitative examples, except for the weight $\lambda$ and input sequence length $T$ which we will vary in a brief ablation study. Note that, as said in the previous section, MLSV can’t be compared for different sequence lengths.

For approach 1 in table 3.4, we can see that it achieves relatively low MLSV values, specially when compared to table 3.3 (representing results for architecture RTH-LODAE). For $T = 9$ AR-LODAE MLSV is reduced for approximately an 80%, when compared to
3.4 Low Order Dynamics Autoencoders LODAE

Figure 3.19: Qualitative results for the first approach of AR-LODAE. Representation of the heatmaps for the normalized singular values of the Hankel matrices $H_z$ of the latent space. Same disposition as in figure 3.18. Cases: training/testing and after 1/280 epochs.

<table>
<thead>
<tr>
<th>MLSV</th>
<th>MSE Prediction</th>
<th>MSE Reconstruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda = 1e-3$</td>
<td>$\lambda = 5e-4$</td>
<td>$\lambda = 1e-3$</td>
</tr>
<tr>
<td>$T=9$</td>
<td>0.02</td>
<td>0.0061</td>
</tr>
<tr>
<td>$T=7$</td>
<td>0.057</td>
<td>0.0053</td>
</tr>
<tr>
<td>$T=5$</td>
<td>0.071</td>
<td>0.0052</td>
</tr>
</tbody>
</table>

Table 3.5: Quantitative results for the second approach of AR-LODAE. We vary the weight $\lambda$ and the sequence length $T$ for a fixed $z_{\text{dim}} = 128$. In red the best results.

RTH-LODAE. Regarding reconstruction and prediction, we see that the values don’t show a clear pattern with respect to $\lambda$ or $T$ but there is a clear gap of approximately 1 order of magnitude between accuracy in prediction and reconstruction. This gap led us to try training with approach 2.

When training with the prediction module in approach 2, with the same setup, we obtain the results in table 3.5. There we can see the effectiveness of this approach. For $T = 9$ MLSV is reduced to around 2% (more than with approach 1) but reconstruction and prediction give very low MSE values, and more interestingly are close to each other (gap of $\approx 0.001$). For the rest of cases the results are similarly good. However, when varying $\lambda$ we see a pronounced difference in terms of MLSV.

We also tested the difference in performance when varying the size of the latent space ($z_{\text{dim}}$), to see how the level of compression affects the performance of the architecture. Results in table 3.6, for $T = 9$ and $\lambda = 1e-3$, suggest that the higher the compression, the
Table 3.6: Quantitative results for the second approach of AR-LODAE. For a fixed $\lambda = 1e^{-3}$, sequence length $T = 9$ we vary $z_{\text{dim}}$ to test different $\lambda$. In red the best results. We can see better linearization results as we decrease $z_{\text{dim}}$ but a clear best prediction for $z_{\text{dim}} = 128$.

Table 3.7: Quantitative results for the second approach of AR-LODAE. For a fixed $\lambda = 1e^{-3}$, sequence length $T = 9$ and $z_{\text{dim}} = 0$ we test the network when the digits in the input have velocity $v = 0$. It behaves as expected as reconstruction and prediction are equally low and MLSV is 0.

lower is MLSV. However, the peak performance for prediction is given by $z_{\text{dim}} = 128$, which suggests that 128 is the optimal value for the dimension of the latent space.

Finally, to ensure that the AR-LODAE’s results are valid, we propose the simplest experiment, for which we know how should the architecture behave. In this case we set the step size to 0, meaning that the digits will not move throughout the video sequence. In this case we won’t standardize the sequences at the output of the encoder, as all of them should be constant in time. Results for approach 2 with $T = 9$, $z_{\text{dim}} = 128$ and $\lambda = 1e^{-3}$ in table 3.7, show equal performance in reconstruction and prediction (ideal prediction) and a MLSV of 0, as would be expected from a constant sequence.

We conclude that AR-LODAE performs well in a dataset as simple as one digit Moving MNIST, giving room for improvement and adaptation to more complex scenarios.
Chapter 4

Conclusions and Future Work

4.1 Conclusions

In this work we presented several new approaches for video reconstruction and prediction, with the ultimate goal of mapping input data in a linear dynamics manifold for predictors with a specific nature.

We have studied the limitations of those predictors and exploited the unused resources that video data provides us with.

In the case of the first three approaches presented in sections 3.1, 3.2 and 3.3, we intended to manipulate the inputs for the system identification learning-based predictor DYTAN, in order to overcome the limitations associated to this architecture, described in section 2.3. This has been done with different levels of success.

We have shown qualitatively that approaches in sections 3.1 and 3.2 fail relatively in the main objective of this work, but succeed in giving lines of research for new tasks, such as compressed video prediction.

In section 3.3 we present the part of the work in [2] that has been carried out during this project (W-DYTAN), which has shown success in two different experiments and beats the SOTA in the KITTI-Caltech experimental setup for several metrics: MSE, SSIM and MMF, if we exclude the other methods presented in citekwdyan (K-DYTAN, KW-DYTAN). This is achieved with a parameter free transformation at the input.

With the same main goal, but without restricting the prediction to DYTAN, we presented two novel architectures called LODAE (in subsections 3.4.1, 3.4.2) that are specifically engineered to map a general video sequence in a manifold in which time sequences have linear dynamics, using a deep learning framework.

We have tested these architectures with a very simple dataset (one digit Moving MNIST) to show their effectiveness.

RTH-LODAE in subsection 3.4.1, performs good in the reconstruction and linear dynamics manifold embedding tasks but leaves the prediction task to other architectures.

A second LODAE architecture, AR-LODAE in subsection 3.4.2, is divided in two approaches. Both have the ability to predict and perform better than RTH-LODAE
in the task of linearization and equally in the task of reconstruction. For prediction, AR-LODAE achieves the wanted results, with a fairly good prediction both qualitatively and quantitatively for different tuning of the network.

4.2 Future Work

Given the results presented throughout the document, we assess that the most promising line of research is the second LODAE architecture, AR-LODAE. We believe that it provides a new and interesting approach to linear dynamics manifold embedding, and therefore, to prediction and interpolation of video by LTI system modelling. Moreover, there are several improvements that could be a key for a competitive performance for this kind of architectures:

- Introduce multiple outcomes and stochastic modelling using VAEs instead of regular AEs. This would certainly help to cope with the inherent uncertainty linked to natural video, between other perks.

- Perform disentanglement of motion and appearance to model separately the dynamics of different objects. This is itself a challenge, but it is crucial in a task such as dynamics linearization for a video with pieces of information behaving differently.

- Adding spatial transformation modules. As deeply discussed throughout this work, modelling motion instead of general appearance is one of the ways of simplifying the dynamics of data. Therefore, giving spatial transformation tools to the architecture would likely help improving its performance.
Bibliography


