Deep learning for multivariate time series : from vehicle control to gesture recognition and generation

Guillaume Devineau

PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à MINES ParisTech

Deep Learning for Multivariate Time Series: From Vehicle Control to Gesture Recognition and Generation

Apprentissage profond pour les séries temporelles multivariées : contrôle de véhicule autonome, reconnaissance de gestes et génération de mouvement

Soutenue par

Guillaume DEVINEAU

Le 2 septembre 2020

École doctorale n°621

Ingénierie des Systèmes, Matériaux, Mécanique, Énergétique

Spécialité

Informatique temps réel, Robotique et Automatique

Composition du jury :

Alexandre GRAMFORT Directeur de recherche, INRIA	*Président du jury*
Catherine ACHARD Maître de Conférences, HDR, Sorbonne Université	*Rapporteur*
Christian WOLF Maître de Conférences, HDR, INSA Lyon	*Rapporteur*
Alexander GEPPERTH Professeur, University of Applied Sciences Fulda	*Examinateur*
Jean-Philippe VANDEBORRE Professeur, IMT Lille Douai	*Examinateur*
Fabien MOUTARDE Professeur, MINES ParisTech	*Directeur de thèse*

MINES ParisTech | PSL

This thesis is dedicated to my grandfather.

Abstract

Artificial intelligence is the scientific field which studies how to create machines that are capable of intelligent behaviour. Deep learning is a family of artificial intelligence methods based on neural networks. In recent years, deep learning has lead to groundbreaking developments in the image and natural language processing fields. However, in many domains, input data consists in neither images nor text documents, but in time series that describe the temporal evolution of observed or computed quantities. In this thesis, we study and introduce different representations for time series, based on deep learning models. Firstly, in the autonomous driving domain, we show that, the analysis of a temporal window by a neural network can lead to better vehicle control results than classical approaches that do not use neural networks, especially in highly-coupled situations. Secondly, in the gesture and action recognition domain, we introduce 1D parallel convolutional neural network models. In these models, convolutions are performed over the temporal dimension, in order for the neural network to detect -and benefit from- temporal invariances. Thirdly, in the human pose motion generation domain, we introduce 2D convolutional generative adversarial neural networks where the spatial and temporal dimensions are convolved in a joint manner. Finally, we introduce an embedding where spatial representations of human poses are sorted in a latent space based on their temporal relationships.

Contents

List of Figures

List of Tables

Acknowledgment

I would like to express my sincere gratitude to my advisor Prof. Fabien MOUTARDE. During my thesis, his guidance regularly helped me, and I genuinely appreciated his advice on my research directions, his support, and our general and specific discussions about deep learning approaches and my thesis. He has always been supportive when I made decisions regarding the research directions to take and helped me very much to conciliate curiosity (ideas exploration) with coherence in the research goals followed (ideas exploitation).

I am also grateful to both of my reviewers, Catherine ACHARD and Christian WOLF, for accepting to read and evaluate the work carried out during these years. They provided insightful remarks and suggestions to make this manuscript clearer and easier to understand.

It has been a great pleasure for me to work with Alexandre THOMAS and Wang XI during my thesis, and I want to express them my sincere appreciation, both from a scientific perspective and from a human perspective.

I genuinely want to thank everyone (!) from the MINES ParisTech Centre for Robotics (CAOR) lab, including Christine, Christophe, Jacky, David, Arnaud, Alexis, Alina, Amaury, Bogdan, Brigitte, Cyril, François, Jean-Emmanuel, Olivier, Patrick, Philippe, Silvère, Simon, Sotiris, Sébastien and Thomas, for making the lab such a nice place to work. More especially I want to warmly thank all the MINES ParisTech Centre for Robotics lab's former and current Ph.D. students, including Philip[1], Grégoire, Aubrey[2], Florent, Marion, Antoine, Arthur, Brenda, Claire, Daniele, Edgar, Emmanuel, Eva, Hassan, Hugues, Jesus, Joseph, Joël, Laëtitia, Marin, Martin, Mathieu, Maximilian, Michelle, Paul, Sami, Selma, Sofiane, Xavier, Yann and Yannick. While only about half of them could help me study the crucial effect of the chocolate mousse found at the French Ministry of Research's canteen on a young Ph.D. student's afternoon intellectual fruitfulness, spending time with all of them was a very delighting experience for me: it was very fun, playful, mind opening and intellectually stimulating to spend time together. I would like to thank all of the other very inspiring and very nice people I met in MINES ParisTech[3], including Akin, Antoine, Arthur, Asma, Benoit, Chloé-Agathe, Hector, Igor, Judith, Marc, Mehdi and Peter.

Whether it was at conferences, at summer schools or at all the research meetings I attended to, I also encountered an incredible amount of very memorable people, including Edouard, Florian, Marc, Marie, Maximilian, Mickaël, Min, Nicolas, Natalia and Quentin, that I would also like to thank.

Last, but not least, I want to thank everyone I love, my dear friends and my very loving family.

[1]The best teammate I could ever find to hike on an active volcano!

[2]The best teammate I could ever find for the Valeo Innovation Challenge!

[3]Most of them belong to the MINES ParisTech Centre for Computational Biology (CBIO).

Chapter 1

Introduction

"Αἰὼν παῖς ἐστι παίζων, πεσσεύων· παιδὸς ἡ βασιληίη.
Time is a child playing at draughts, a child's kingdom."

Heraclitus

Contents

1.1 Context

Our current societies are based upon the implicit assumption that progress is, ultimately, closely tied to better standards of living for human people. In particular, the access to information and communication tools has blatantly soared in the last decades, up to a point where information technology holds a central and strategic position in the organization of our lives. Computer science plays a crucial role in telecommunications networks, in the management of vital networks (electricity grids, transportation networks, water or gas distribution networks), in office automation, for the Internet, for the *media* or in robotics, for instance. Moreover, information technology also penetrates new domains, such as health, domotics, smart wearables or autonomous vehicles. Autonomous vehicles (AV) can already detect pedestrians in order to avoid them and drive smoothly without any human intervention in simple scenarii[1]. Computers offer undeniable advantages over humans, including faster speeds, bigger memory, absence of fatigue, fewer errors and lower costs. They also raise unresolved concerns regarding a variety of topics such as ethics, environment, social implications, user's rights, data regulation, sovereignty, or equal and fair treatment, to name a few. From a strictly scientific point of view, computers also still present a major drawback: computer programs are unable to adapt themselves in order to cope with difficult tasks.

Intelligence can arguably be considered as the ability to adapt oneself and to process information in order to maximize one's chance of achieving one's objectives. Artificial Intelligence (AI) is the scientific domain that studies "intelligent" programs. More broadly speaking, AI can also be considered as an umbrella term for a wide range of concepts and technologies that allow machines to exhibit human-like capabilities.

Deep Learning (DL) is an artificial intelligence method that allows a program to learn a hierarchy of concepts from examples, without requiring a human to explicitly provide any of these concepts before. By combining several concepts, programs can discover more abstract concepts. For instance, by combining a geometrical concept of a cross with a concept of a green color, programs can discover the more abstract concept of a pharmacy cross. The composition of discovered (learned) representations in deep learning algorithms allows them to build a hierarchical representation of the data; that representation can become more and more abstract as the number of layers in the composition goes up. Deep learning techniques have proved to be extremely successful in the image domain in the last decade, as well in the natural language processing (NLP) domain -i.e. the text domain- in the very recent years. All current state-of-the-art approaches in these domains make use of deep learning techniques and models.

However, in numerous domains, the observed input data to process are neither images nor texts but time series that represent the evolution of measurements or computed values. It is the case in meteorology (e.g. temperature, pressure, wind speed), in economy (rates, index, spread), in seismology, in the industry (voltage, electric energy consumption, sensors), in medicine (EEG, ECG, temperature, blood pressure), in epidemiology (active cases in a disease or a pandemic), in speech recognition (audio sequences, mel-spectograms), in autonomous vehicle control (spatial reference trajectories) and in

[1] Current autonomous vehicles are able control the steering and the acceleration or deceleration based on their perception of the surrounding environment, but still require a human driver to monitor the driving environment. The long-term goal pursued in research on autonomous vehicles is to make vehicles fully autonomous. Research on autonomous vehicles is a very active research domain driven by actors from academia, startups and multinational corporations.

gesture recognition (positions or orientations of the human body's joints), to name a few domains.

1.2 Objectives

The initial objectives of this thesis are to explore deep learning approaches for time series, and to find out whether designing an approach based on deep learning techniques can lead to substantial gains and progress for tasks that involve time series. In order to delimit the scope of this thesis while trying to address these broad objectives, we orient our research with several choices.

First, in order to focus on the temporal aspect of time series, we choose to work on motion-related time series data: in chapter 3 the input data consists in the reference trajectory that a vehicle should follow, while in chapter 4 and chapter 5 the input data consists in sequence of human poses.

Second, for the same reason, we choose to design deep learning architectures that (essentially) act on the temporal dimension.

Third, we choose to avoid the use of Recurrent Neural Networks[2,3] (RNNs). At the time of the beginning of this thesis, the state-of-the-art approaches for sequence modeling with deep learning methods were RNN-based. While being a very elegant theoretical answer for sequence modeling with neural networks, RNNs present many issues at both applied and theoritical levels: e.g. RNNs are slow to train, very sensitive to initialization, struggle to capture very long-term relationships and are hard to regularize.

While exploring deep learning approaches for motion-related tasks, the aim of this thesis is therefore to answer the following questions:

Question 1 *Recurrent Neural Networks (RNNs) resort to a memory vector when they process a sequence. How to model sequences with a neural network without having to resort to an external memory like RNNs do, while achieving comparable or better performance?*

Question 2 *Can time-series be sufficient to serve as the only input when modeling non-trivial and non-periodic tasks, like motion-related tasks?*

1.3 Contributions

The main contributions of this thesis can be synthesized as follows.

Coupled Vehicle Control We propose a novel approach for coupled vehicle control based on a deep learning model.

Our exact contributions are discussed more precisely in a subsection in chapter 3.

To the best of our knowledge, deep neural networks have not been used before for the coupled control of wheeled vehicles.

[2]Recurrent Neural Networks (RNNs) are introduced in detail in chapter 2.
[3]May it be vanilla RNNs or their extensions. See chapter 2.

Two possible architectures are presented and tested. The neural network model learns the inverse dynamics of a vehicle, in particular the coupled longitudinal and lateral dynamics. Once trained, the neural network is used as a controller for the vehicle. Such deep learning controller is able to handle situations with strongly coupled longitudinal and lateral dynamics in a very short time.

- Guillaume Devineau et al. (2018c). "Coupled longitudinal and lateral control of a vehicle using deep learning". In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, pp. 642–649

Human Pose Motion/Sequence Recognition We propose a novel deep learning architecture for action and gesture recognition of human pose sequences, where parallel 1D convolutions over time are used to detect temporal patterns.

Our exact contributions are discussed more precisely in a subsection in chapter 4.

The proposed architecture only uses convolutions, which are easy to train and to audit, and leads to fast and relatively light models. Moreover the model architecture can be applied without adaptation work to various types of gestures, various types of sensors, and various types of pose sequences including facial landmarks sequences, hand pose sequences and full-body pose sequences.

- Guillaume Devineau et al. (2018b). "Deep learning for hand gesture recognition on skeletal data". In: *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*. IEEE, pp. 106–113
- Guillaume Devineau et al. (2018a). "Convolutional neural networks for multivariate time series classification using both inter-and intra-channel parallel convolutions". In: *2018 RFIAP (Reconnaissance des Formes, Image, Apprentissage et Perception) conference, (RFIAP 2018)*. RFIAP

Human Pose Embedding We propose a novel denoising auto-encoder neural network architecture for self-supervised learning on human poses. Our exact contributions are discussed more precisely in a subsection in chapter 5. Our approach can be summarized as follows:

The latent space of the denoising auto-encoder is constrained with a spatial loss and a temporal loss. The spatial loss ensures that the poses are correctly reconstructed spatially by the auto-encoder. The temporal loss, a time-sampled triplet loss, ensures that the internal (spatial) representations of the poses learned by the network are organized[4] in a temporally coherent manner. The resulting latent space, and the encoder and decoder networks can notably be used for human pose embedding purposes, in both offline and online settings. The latent space of the denoising auto-encoder model displays semantic meaning properties.

[4]E.g. in the latent space.

Human Pose Motion/Sequence Generation We propose a novel Generative Adversarial Network (GAN) architecture for human pose motion generation. Our exact contributions are discussed more precisely in a subsection in chapter 5. Our approach can be summarized as follows:

Our approach first converts human pose sequences into a 2D spatio-temporal image-like format, that can thus be used by deep convolutional GANs architectures.

1.4 Thesis outline

This thesis is laid out in six chapters:

Introduction In this chapter, we briefly introduced the context of this thesis, the problematics linked to this thesis and the main contributions of this thesis.

Overview of Sequence Modeling with Neural Networks In this chapter, we introduce the different families of deep learning architectures and models prominent in the research literature for sequence modeling. The models fall into four broad main categories: fully-connected neural networks, recurrent neural networks, convolutional neural networks and attention-based associative memory neural networks. Fully-connected layers virtually make no assumption about the 1D structure of sequences, and, as such, do not benefit from an inductive bias. Convolutional Neural Networks (CNNs) share a reduced amount of parameters along the time axis in order to detect temporal regularities in sequences. Recurrent Neural Networks (RNNs) take a different approach: they take advantage of an external dynamic[5] memory -i.e. a vector- to store information between time steps. Finally, attention-based associative memory neural networks propose to organize the models' internal representations based on similarities between inputs. This last category of neural networks can be seen as a first step towards neural networks trained with self-supervised learning only.

Vehicle Control with Feedforward Neural Networks Autonomous vehicles rely on three main subsystems: namely the perception, the planning, and the control systems. Perception translates raw sensor data into meaningful intelligence about a vehicle environment. Planning refers to the process of making decisions in order to achieve the vehicle goals, e.g. to bring the vehicle to a goal location while avoiding obstacles. Finally, control refers to the process of executing the planned actions given the state of the vehicle and its environment.

In this chapter, we propose a novel framework for vehicle control where the control commands are estimated by a neural network; we propose two architectures that can be used for that neural network: a vanilla fully-connected neural network architecture and a convolutional neural network (CNN) architecture. In experiments on a test track, we show that the proposed framework leads to better results than human-designed non-deep classical approaches. We observe that the CNN model is able to better capture the temporal dynamics than the fully-connected model, leading to better results in the CNN case.

[5]The processing is still based on static weights/parameters, like fully-connected and convolutional neural networks do.

Gesture Recognition with Convolutional Neural Networks over Time Gesture is one of the most natural and simplest way to interact with one's environment, including other humans and machines. It is complementary to voice and does not require a complex physical brain-machine interface. The ability to recognize human intents and actions is useful for numerous real-life situations, and even critical for the design of meaningful interactions between humans and machines. Moreover, it is known that 3D human pose, i.e. a sparse representation of the human body based on information about its joints, is sufficient to describe and understand human motion, from a human perspective.

In this chapter, we propose a novel approach to perform gesture recognition based on parallel 1D convolutions over the time dimension, on human pose sequences. The proposed network architecture, the SkelNet, only uses convolution layers, dispensing with recurrence entirely. We study the performance of the proposed SkelNet models with regards to the architecture design choices, the parameters used, and on different tasks.

Human Motion Generation with Deep Learning A generative model is a model that models the data generation process. Since generative models mostly rely on inductive biases, they tend to be good at out-of-domain generalization.

Motivations for human pose motion generation include: realistic video synthesis, professional training in technical skills and gestures, animation movies, virtual reality, augmented reality, video games, ergonomic assessments, motion analysis for sport, musical or medical purposes, exploration and creation of dance choreographies. Finally, human pose motion generation could also be used to perform data augmentation for other deep learning tasks.

In this chapter, we propose two novel approaches for human pose motion generation, with the help of neural networks.

The first approach is based on a Denoising Auto-Encoder of (spatial) human poses, where an additional temporal constraint is enforced with the help of a triplet loss. This approach allows to embed static human poses in a spatially and temporally constrained latent space. The latent space of the denoising auto-encoder model displays semantic meaning properties.

The second approach is based on a 2D Generative Adversarial Network (GAN). Human poses sequences are first represented as 2D, image-like, data. A conditional GAN with a convolutional neural network architecture operating on this representation is then proposed to generate human pose sequences.

Conclusion In this chapter, we summarize this thesis and propose possible future extensions to our work.

Chapter 2

Overview of Sequence Modeling with Neural Networks

"Pour bien savoir les choses, il en faut savoir le détail."

La Rochefoucauld

Contents

2.1 Introduction

In this chapter, we present an overview of major approaches for sequence modeling using artificial neural networks.

Sequence modeling

Let $n \in \mathbb{N}^*$. A *sequence* $\mathbf{s}$ is a finite, or infinite, sequence of data points $\mathbf{x}_t$ (where $\forall t \in \mathbb{T}$, $\mathbf{x}_t \in \mathbb{R}^n$) indexed by a totally ordered set $\mathbb{T}$ called time:

$$\mathbf{s} = (\mathbf{x}_t)_{t \in \mathbb{T}} \tag{2.1}$$

When $\mathbb{T}$ is a discrete set (e.g. $\mathbb{T} = \mathbb{N}$ or $\mathbb{T} = [\![1, 100]\!]$), the sequence is said to be discrete. When $\mathbb{T}$ is continuous (e.g. $\mathbb{T} = \mathbb{R}$), the sequence is said to be continuous.

A text sequence is a sequence of embedding vectors, where each embedding vector represents an individual word or "token". While text sequences may seem no different than other (types of) sequences, they are actually very different, by nature. Text sequences represent highly semantic, extremely sparse and very arbitrary content. In contrast, almost all other types of sequences represent content that is more regular. For instance, an audio sequence displays regularities that are directly related to the physical constraints and the dynamics of audio signals in the real, physical world.

In this thesis, we consider discrete sequences of physical quantities' observations, indexed by natural numbers (i.e. $\mathbb{T} = \mathbb{N}$), since sequences usually come from physical sensors that output values at discrete time steps[1]. Frequent tasks that involve sequences include:

prediction the goal is to predict the value of a sequence at a time index t_p based on previous values only, i.e. based on values whose temporal index t is such as $t < t_p$.

classification the goal is to assign a label to a sequence

anomaly detection the goal is to detect whether a sequence presents "abnormal" values or behavior

content-based retrieval the goal is to find, among many sequences, the sequence that is the most "similar" to another sequence called request

motive/motif discovery the goal is to determine short subsequences called motives (or *motifs*) that are repeated in a sequence

clustering the goal is to group sequences based on their similarity

transduction the goal is to transduce (or translate) a sequence into another sequence

segmentation the goal is to find a temporal partition of a sequence, so that the resulting temporal segments make sense from an application-domain perspective

[1] Sensors values are usually already temporally synchronized and resampled at a lower level, e.g. at the hardware-level.

Since a sequence can be viewed either as a whole, or as individual data evenly spaced on a 1D-grid, it is worth noting that most of these tasks can be performed at a sequence level (e.g. one classification: of the whole sequence) or at a time step level (e.g. many classifications: one for each individual value of a sequence). This also holds true for sequence transduction tasks where both sequences can be seen as a whole (one) or as a collection of values (many), leading to four possible scenarii (one-to-one, one-to-many, many-to-one, many-to-many).

Deep Learning approaches

Classic approaches for sequence modeling that do not rely on artificial neural networks are essentially model-based and not data-adaptive. As such, the only way to incorporate a domain knowledge about the time series or sequences used consists in handcrafting rules with the help of experts and devising a new model that incorporates these rules. Designing such expert rules appears to be very hard in the real world. This is a crucial issue, since, as Elman, who introduced the famous recurrent neural networks[2], noted: "*The representation of time -and memory- is highly task-dependent*". Classic approaches include AutoRegressive Integrated Moving Average (ARIMA) models (Box et al., 1970) estimated following the Box–Jenkins approach (Box et al., 1970), or Generalized AutoRegressive Conditional Heteroskedasticity (GARCH) (Bollerslev, 1986) models for instance.

Machine Learning (ML) based approaches tend to be notably more efficient for sequence modeling than classic approaches, because they are data-adaptive. However, they share the same drawback as the classic approaches: they often rely on carefully handcrafted features that require human expert knowledge. Machine learning approaches include the k-Nearest Neighbors (k-NN) model (Hastie et al., 2009; Lin et al., 2007) combined with the Dynamic Time warping (DTW) discrepancy (Bagnall et al., n.d.; Cuturi et al., 2017; Sakoe et al., 1978) for instance.

Statistical modeling approaches close to machine learning methods usually show good performances too, while sharing benefits and drawbacks of the two approach families discussed before. Statistical modeling approaches close to machine learning methods include Hidden Markov Models (HMMs) (Leong et al., 2006; Rabiner, 1989; Starner et al., 1997; Yamato et al., 1992), Conditional Random Fields (CRFs) (Lafferty et al., 2001; Sutton et al., 2006), or Singular Spectrum Analysis (SSA) (Ghil et al., 2002; Golyandina et al., 2001; Vautard et al., 1992) for instance.

Finally, the state-of-the-art approaches for sequence modeling currently are based on Deep Learning (DL) neural networks: regardless of the domain, approaches that use a neural network tend to display better results than approaches that do not use a neural network. Deep learning approaches are data-adaptive and do not require human expert knowledge contrary to the formerly mentioned approaches: neural networks learn internal representations and features by their own during training. As such, neural networks do not require expert rules to model sequence patterns. Like all the other approaches mentioned, the performance of neural networks not only results from the data but also from their sequence-focused design: the choice of a good neural network architecture is crucial.

In the next sections of this chapter, we present an overview of major deep learning architectures and approaches for sequence modeling. The proposed overview is not a complete and exhaustive list of all

[2]Recurrent neural networks are presented in section 2.3.1.

the architectures and approaches for sequence modeling, but rather a shortlist of the most commonly encountered ones in the deep learning for sequences research literature. The major approaches presented rely on the three most popular deep learning layers: fully-connected layers (TDNNs in section 2.2, and attention mechanisms in section 2.5), recurrent layers (RNNs in section 2.3) and convolutional layers (CNNs in section 2.4).

2.2 Fully-Connected Neural Networks

Multi-Layer Perceptrons (MLP) with Time Delay Neural Networks (TDNN)

The universal approximation theorem (Hornik, 1991) states that, under hypotheses on the activation function, a feedforward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbb{R}^n$. As such, one may use a trivial feedforward neural network f_θ such as a Multi-Layer Perceptron (MLP) to model sequences, like any other type of data.

When their statistics do not change over time, sequences are said to be stationary. In many real world scenarii however, sequences are not stationary; the interpretation of a feature in the data depends on earlier features and/or the time they appeared at. Sequences can also be arbitrary long. However, fully connected layers require a fixed number of inputs. Their general purpose connection pattern does not naturally benefit from regularities, like periodicity, that may exist in time series data either. Due to these shortcomings and -good, yet- poor results when compared to more complex neural network architectures, vanilla multi-layer perceptrons are hardly ever used on their own for sequence modeling when it comes to artificial neural networks.

An exception to this assessment is the so-called Time Delay Neural Network (TDNN) architecture (Waibel et al., 1989). TDNNs overcome the main obstacle of sequence processing for MLPs: the length of the sequence. Sequences are usually long and have potentially variable -or even infinite- length T of their input sequence $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$. TDNNs only perform computations on a small finite subsequence $\mathbf{x} = \left(\mathbf{x}_{\phi(t-\tau)}, \mathbf{x}_{\phi(t-(\tau-1))}, \ldots, \mathbf{x}_{\phi(t)}\right)$ of the input sequence, of length τ, at each time step t:

$$\mathbf{y}_t = f_\theta^{TDNN}(\mathbf{x}_{\phi(t-\tau)}, \mathbf{x}_{\phi(t-(\tau-1))}, \ldots, \mathbf{x}_{\phi(t)}) \tag{2.2}$$

where ϕ is a strictly increasing function $\phi : \mathbb{N} \to \mathbb{N}$ that extracts a subsequence from $\mathbf{x}$. In practice, however, that subsequence is usually based on consecutive indices or time steps:

$$\mathbf{y}_t = f_\theta^{TDNN}(\mathbf{x}_{t-\tau}, \mathbf{x}_{t-(\tau-1)}, \ldots, \mathbf{x}_t) \tag{2.3}$$

Such a TDNN processing successive inputs ($\phi = identity$) can be viewed[3] as a 1-dimensional Convolutional Neural Network (CNN) with a stride of 1 and a kernel size of $\tau + 1$. More generally, the choice of ϕ is crucial to TDNNs, as it is supposed to find regularity in the data over time.

[3]However, in practice, the value of τ used in TDNNs is often an order of magnitude greater than the value of τ used in CNNs. It is known that deep CNN models with small kernels tend to be more expressive than shallow CNN models with large kernels, for a comparable parameters count.

Finding ϕ may require an expert knowledge. Moreover, even with a large τ and/or a large sliding window, the time-context of TDNNs is limited.

An illustration of a TDNN with an exponential delay is proposed in figure 2.1. In the illustration proposed in figure 2.1, the TDNN only performs computations on a small finite subsequence of length $\tau = 5$, at each time step t. More precisely, at each time step t of the input sequence, the TDNN only considers the following input subsequence: $\left(\mathbf{x}_{\phi(t-\tau)}, \mathbf{x}_{\phi(t-(\tau-1))}, \ldots, \mathbf{x}_{\phi(t)}\right)$ where ϕ is an exponential delay function defined (at each time step t) by $\phi(t-\tau) = t$ for $\tau = 0$ and by $\phi(t-\tau) = t - 2^{\tau-1}$ for $\tau \in [\![1,4]\!]$. The exponential delay function is used for illustration purposes only: other ϕ functions can be considered as well, as mentioned earlier.

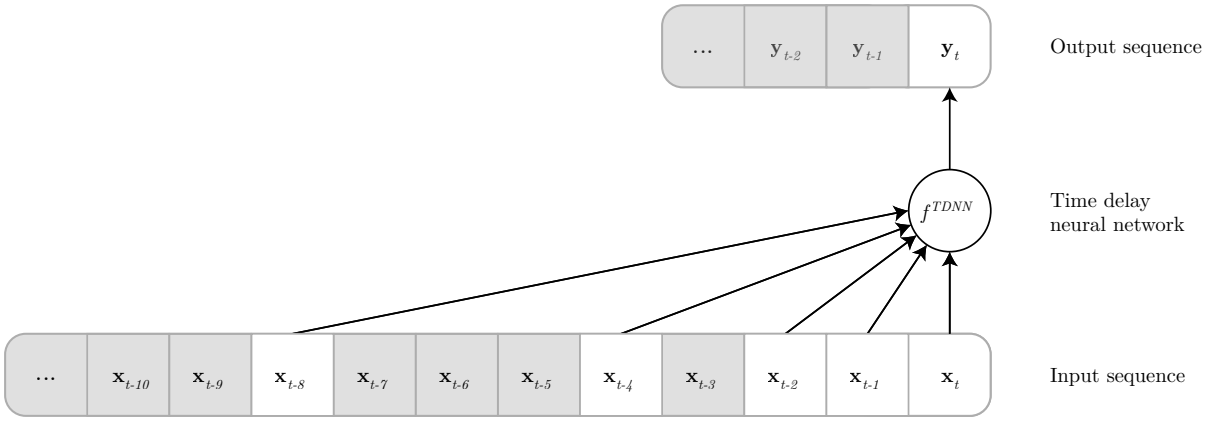


Figure 2.1 – Illustration of a Time Delay Neural Network (TDNN). The example TDNN depicted here has an exponential delay: $\mathbf{y}_t = f_\theta^{TDNN}(\mathbf{x}_t, \mathbf{x}_{t-2^0}, \mathbf{x}_{t-2^1}, \mathbf{x}_{t-2^2}, \mathbf{x}_{t-2^3})$. Input and output values not related to the the current time step t are greyed out on the illustration.

Fully-Connected Neural Networks with Attention-Mechanisms

The performance of a TDNN is closely related to the exact design of its time extraction function ϕ. However, a TDNNs' ϕ function has two major drawbacks. The first one stands in the fact that ϕ is handcrafted based on a human per-domain expert knowledge, while it could advantageously be learned during the training. The second drawback is more subtle, but closely related to the first one. The role of ϕ is to make binary decisions about whether each individual time step (value) should be included in the list of the TDNN neural network inputs, or not. Rather than performing a hard and hand-designed filtering of the input, a soft and learned filtering[4] of the input could be used to help the network better devise by itself during training which information is relevant or not in its input.

The process of selectively focusing on an aspect of the input information -while ignoring other perceivable information present in the input- is called attention.

Attention mechanisms in neural networks are not specific to fully-connected neural networks. As such, they are introduced in their own section (section 2.5). However, it is worth noting that attention-based models from the Transformer neural network family (introduced in section 2.5.2) are based on a fully-connected architecture. Such models achieve state-of-the-art performance in one domain involving

[4]Based on a (learned) mask applied to the input, for instance.

sequential data (the human written language domain), outperforming both convolutional and recurrent neural networks.

2.3 Recurrent Neural Networks

Feedforward neural networks excel at pattern recognition. However, they lack plasticity as they rely on a *fixed* architecture and on *fixed* parameters θ to process their input. Recurrent Neural Networks (RNNs) combine feedforward neural networks with hidden states -which one can view as *dynamic* memories- to overcome this shortcoming.

This section will first dive into the generic mechanism of Recurrent Neural Networks with *vanilla* RNNs and will then put the spotlight on a few typical state-of-the-art gated recurrent neural networks.

2.3.1 *Vanilla* Recurrent Neural Networks

A Recurrent Neural Network (RNN) processes a variable-length sequence of inputs $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$ sequentially, one input $\mathbf{x}_t$ at a time, using a feedforward neural network f_θ and a hidden state $\mathbf{h}_t$ whose value evolves over time. At each time step t, the hidden state value is given by

$$\mathbf{h}_t = f_\theta(\mathbf{h}_{t-1}, \mathbf{x}_t) \tag{2.4}$$

where $\mathbf{h}_0$ is an initial hidden state vector, f_θ a feedforward neural network, θ its parameters, and $(\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_T)$ is the sequence of values taken by the hidden state vector over time. For convenience, in the rest of this thesis, such a feedforward neural network f_θ will be referred to as a Recurrent Neural Network (RNN), and the associated $\mathbf{h}$ as the RNN's hidden state. Depending on the definition, the output of the RNN can either be the raw hidden state $\mathbf{h}$, or its value $\sigma_{output}(\mathbf{h})$ after an activation function σ_{output}.

If σ_h, σ_y are two non-linearity functions, f_θ a dense layer, $\mathbf{y}$ the prediction, $\mathbf{b}_h$ and $\mathbf{b}_z$ two bias vectors and $\mathbf{W}_{uv}$ a weight matrix connecting a vector $\mathbf{u}$ to a vector $\mathbf{v}$, the former recurrence relation coincides with the definition of an RNN as initially introduced in (Elman, 1990):

$$\begin{aligned} \mathbf{h}_t &= \sigma_h(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h) \\ \mathbf{y}_t &= \sigma_y(\mathbf{W}_{hy}\mathbf{h}_t + \mathbf{b}_y) \end{aligned} \tag{2.5}$$

An illustration of a RNN is proposed in figure 2.2: on the right side of the figure 2.2, the so-called "unfolded" graphical representation depicts how the same feedforward neural network f_θ is applied to different inputs $\mathbf{x}_t$ over time while conveying a memory vector $\mathbf{h}_t$; on the left side of the figure 2.2, the "folded" graphical representation that is commonly used to summarize the "unfolded" operations is presented.

A RNN can learn a probability distribution over a sequence $(\mathbf{x})_{t \in [\![1,\ldots,T]\!]}$ by being trained to predict the next symbol in that sequence: the output at each time step t is the conditional distribution $p(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{x}_{t-2}, \ldots, \mathbf{x}_1)$.

The probability of the sequence $\mathbf{x} = (\mathbf{x})_{t\in[\![1,\dots,T]\!]}$ is therefore given by:

$$p(\mathbf{x}) = p(\mathbf{x}_1)p(\mathbf{x}_2|\mathbf{x}_1)p(\mathbf{x}_3|\mathbf{x}_1,\mathbf{x}_2)\dots p(\mathbf{x}_T|\mathbf{x}_{T-1},\mathbf{x}_{t-2},\dots,\mathbf{x}_1) = \prod_{t=1}^{T} p(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{x}_{t-2},\dots,\mathbf{x}_1) \quad (2.6)$$

The learned distribution can also be used to sample new sequences iteratively, one value after another.

The RNN's model f_θ itself is invariant through time translation due to the recursive expression. This provides an inductive bias to the RNN. Conversely, the hidden state value $\mathbf{h}_t$ at time step t depends on the input subsequence $(\mathbf{x}_1,\mathbf{x}_2,\dots,\mathbf{x}_t)$ already processed by the RNN beforehand. Assuming the RNN's parameters θ are trained such as the output features $\mathbf{h}_t$ are a representation of the input subsequence $(\mathbf{x}_1,\mathbf{x}_2,\dots,\mathbf{x}_t)$ up to time step t for each time step, a RNN effectively maps a variable-length sequence $(\mathbf{x})_{t\in[\![1,\dots,T]\!]}$ to a fixed-size representation $\mathbf{h}_T$.

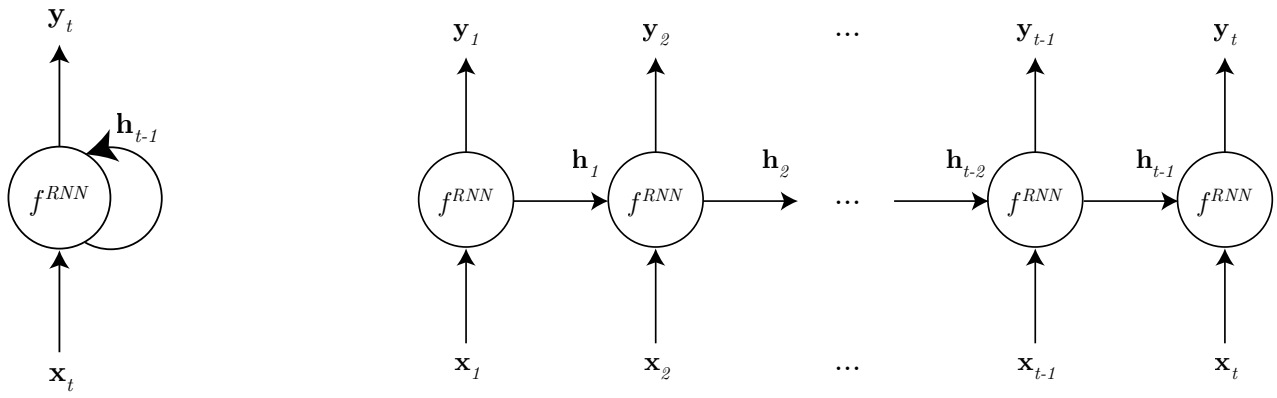


Figure 2.2 – Illustration of a Recurrent Neural Network (RNN). The *folded* graphical representation (left) is often used to summarize the *unfolded* graphical representation (right).
Figure adapted from (Le, 2015).

2.3.2 Recurrent Neural Networks Training

As with standard feedforward networks, weight initialization matters in recurrent neural networks. Some authors suggest initializing vanilla RNNs with the identity matrix to make it easier for the information to flow from one time step to another. For more complex gated RNN cells -presented in the next section- like LSTM cells, initializing the gates biases uniformly in the expected range of long-term dependencies seems to have the same effect.

Training a feedforward neural network using backpropagation requires to calculate its derivatives and to apply the derivative chain rule. However, in RNNs, gradients do not only depend on the input at a single time step, but also on the time steps before. To train RNNs, an extension of the regular backpropagation algorithm, called Backpropagation Through Time (BPTT) (Werbos, 1990) is used. BPTT unfolds the recurrent neural network in time, then uses the backpropagation algorithm, keeping in mind that the network parameters are the same at each time step. In practise, the Truncated Backpropagation Through Time (Jaeger, 2002), which truncates the history used in order to relieve the need for a complete backtrack through the whole input sequence at every step, is used, at the risk of favoring short-term dependencies.

When the gradients are being propagated back in time all the way to the initial layer, they go through numerous matrix multiplications due to the chain rule. A gradient with a small initial amplitude will tend to decay (*vanishing gradient*) during that process and have no influence on learning, while a gradient with a large initial amplitude will tend to grow too much (*exploding gradient*) and -usually- to cause numerical conditioning issues or have a disproportionate influence on learning.

A few solutions to that issue exist. One may clip the gradient amplitude (i.e. $g_{\text{clip}} = \min(g, g_{\max})$) to avoid such exploding gradients, or even normalizing it no matter its initial value. Regarding the loss function, skip connections are known to produce smoother loss functions that are easier to train, both for feedforward networks and for RNNs; as such, skipping state updates can help training RNNs. Some other tricks such as teacher forcing (Williams et al., 1989), which consists in feeding the RNN with the ground truth output $\mathbf{y}_t$ as input at time $t+1$ during training can also be used for RNN training. Layer normalization is also known to improve the performance of vanilla recurrent networks.

However, most of the formerly mentioned techniques are often not sufficient to successfully train RNNs by their own.

Besides these training initialization and training issues, regularizing RNNs reveals itself to be difficult. RNNs need specific regularization schemes, as they naturally posses a stronger inductive bias than feedforward networks due to their recurrent definition.

Most famous RNN regularization techniques include zoneout (Krueger et al., 2016), variational dropout (Gal et al., 2016), recurrent dropout (Semeniuta et al., 2016), dropconnect on hidden-to-hidden weights with an averaged stochastic gradient descent training method (Merity et al., 2017) and recurrent batch normalization (Cooijmans et al., 2016).

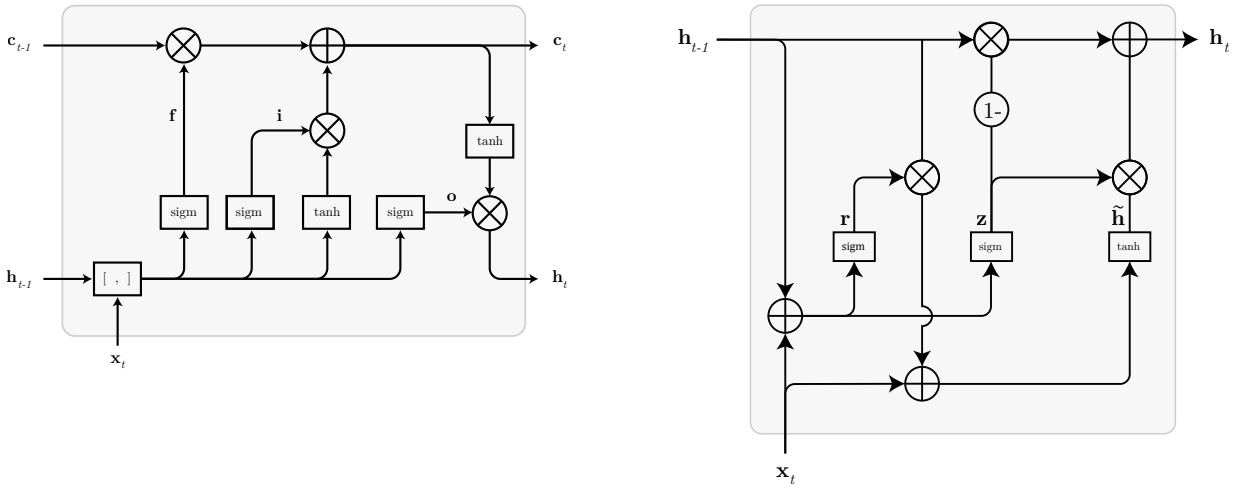
2.3.3 Recurrent Neural Networks with Gates

To succesfully train RNNs, two main approaches stand out. One is to devise a better and dedicated learning algorithm, e.g. with gradient clipping or by smoothing the gradients. The other one is to use more complex activations function than usual activation functions, in order to warp time. The second option is performed with the help of gates. Gates indicate if their input signal should be attenuated, thus allowing an information to flow more or less depending on the input. Invariance to time warping leads to gate-like mechanisms in recurrent models (Tallec et al., 2018).

RNNs with gates have proven to be much easier to train and to be able to capture longer-term dependencies than vanilla RNNs. In this section we present two main RNN models with a gating mechanism: Long Short-Term Memory Units (LSTMs) and Gated Recurrent Units (GRUs). These two models are the reference RNN models used in the literature. An illustration of these two gated RNN models is proposed in figure 2.3.

Long Short-Term Memory Units (LSTMs)

Long Short-Term Memory units (LSTMs) (Greff et al., 2016; Hochreiter et al., 1997b) are RNNs with gates and a structure called Constant Error Carousel (CEC) designed to prevent vanishing gradients. The memory is conveyed from one time step to another in a vector $\mathbf{c}_t$ whose update equation is both linear and weightless, the actual non-linear computations being performed by the other portions of the LSTM.



(a) Long Short-Term Memory (LSTM)

(b) Gated Recurrent Unit (GRU)

Figure 2.3 – Illustration of two gated RNNs: (a) LSTM and (b) GRU.
Figure reproduced from (Olah, 2015).

A LSTM is to a vanilla RNN what a sheet of paper is to chinese whispers. In vanilla RNNs, the hidden state to transfer is altered at each update, potentially a lot, as there is no mechanism designed to protect its value. In LSTMs, the hidden state is only altered deliberately to add or to remove an information, as decided by mainly two gates: the input and forget gates. These gates control how much the hidden state should stay unchanged over time. A third gate named ouput gate controls the output of the LSTM.

The mathematical formulation of the LSTM is presented in equations 2.7.

$$
\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i) \\
\mathbf{f}_t &= \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o) \\
\tilde{\mathbf{c}}_t &= \tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_n) \\
\mathbf{c}_t &= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{f}_t \odot \mathbf{c}_{t-1} \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned}
\tag{2.7}
$$

where $\mathbf{W}_i, \mathbf{W}_f, \mathbf{W}_o, \mathbf{U}_i, \mathbf{U}_f, \mathbf{U}_o$ are weight matrices, $\mathbf{x}_t$ is the input vector at time step t, $\mathbf{h}_t$ is the current exposed hidden state, $\mathbf{c}_t$ is the memory cell state, and $\odot$ is element-wise multiplication.

An illustration of a LSTM unit is proposed in figure 2.3 (a). Besides illustrating the equations 2.7, the figure 2.3 (a) also highlights the order and the role of the three gates -the forget gate ($\mathbf{f}_t$), the input gate ($\mathbf{i}_t$) and of the output gate ($\mathbf{o}_t$)- as well as the central role of the memory state $\mathbf{c}_t$ in a LSTM unit to convey information between two successive time steps.

Another commonly used variant is the LSTM with peephole connections (Gers et al., 2001). In a standard LSTM without peephole connections, gates can only observe the input units and the outputs of all cells, however there is no direct connection between them and the constant error carousel $\mathbf{c}$. This can be harmful when the output gate is closed, as none of the gates has access to the CECs they control. LSTM with peephole connections, are an extension of LSTMs where gates have access to the internal

state **c**. The formulation of a peephole LSTM, is given in equations 2.8.

$$
\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_i\mathbf{x}_t + \mathbf{U}_i\mathbf{h}_{t-1} + \mathbf{P}_i\mathbf{c}_{t-1} + \mathbf{b}_i) \\
\mathbf{f}_t &= \sigma(\mathbf{W}_f\mathbf{x}_t + \mathbf{U}_f\mathbf{h}_{t-1} + \mathbf{P}_f\mathbf{c}_{t-1} + \mathbf{b}_f) \\
\tilde{\mathbf{c}}_t &= \tanh(\mathbf{W}_c\mathbf{x}_t + \mathbf{U}_c\mathbf{h}_{t-1} + \mathbf{b}_n) \\
\mathbf{c}_t &= \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{f}_t \odot \mathbf{c}_{t-1} \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o\mathbf{x}_t + \mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{P}_0\mathbf{c}_t + \mathbf{b}_o) \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned}
\tag{2.8}
$$

where $\mathbf{W}_i, \mathbf{W}_f, \mathbf{W}_o, \mathbf{U}_i, \mathbf{U}_f, \mathbf{U}_o, \mathbf{P}_i, \mathbf{P}_f, \mathbf{P}_o$ are weight matrices, $\mathbf{x}_t$ is the vector input to the time step t, $\mathbf{h}_t$ is the current exposed hidden state, $\mathbf{c}_t$ is the memory cell state, and $\odot$ is element-wise multiplication.

LSTMs can still be trained with BPTT like vanilla RNNs but are easier to train and able to learn long(er)-term dependencies between the input.

LSTM gates do not have the same importance to remember the sequences: the forget gate is more important that the output gate (Greff et al., 2016). The input gate is of negligible importance when compared to both of theses gates: $f > o >> i$.

Gated Recurrent Units (GRUs)

Gated Recurrent Unit (GRU) (Cho et al., 2014) is a simplified variant of the LSTM architecture. A GRU does not either use peephole connections or output activation functions. In a GRU, input and forget gates are coupled into an update gate. The GRU output gate is called reset gate.

GRUs have fewer parameters than LSTMs. While LSTMs tend to slightly better perform than GRUs, their performance is comparable for many tasks. Tuning hyperparameters such as the layer size is often more important than finding the best architecture between the two in practice.

The mathematical formulation of the GRU is given in equations 2.9.

$$
\begin{aligned}
\mathbf{z} &= \sigma(\mathbf{W}_z\mathbf{x}_t + \mathbf{U}_z\mathbf{h}_{t-1} + \mathbf{b}_z) \\
\mathbf{r} &= \sigma(\mathbf{W}_r\mathbf{x}_t + \mathbf{U}_r\mathbf{h}_{t-1} + \mathbf{b}_r) \\
\widetilde{\mathbf{h}}_t &= \tanh(\mathbf{W}_h\mathbf{x}_t + \mathbf{r} \odot \mathbf{U}_h\mathbf{h}_{t-1} + \mathbf{b}_{\widetilde{h}_t}) \\
\mathbf{h}_t &= \mathbf{z} \odot \mathbf{h}_{t-1} + (\mathbf{1} - \mathbf{z}) \odot \widetilde{\mathbf{h}}_t
\end{aligned}
\tag{2.9}
$$

An illustration of a GRU unit is proposed in figure 2.3 (b). Besides illustrating the equations 2.9, the figure 2.3 (b) also highlights the central role of the hidden state $\mathbf{h}_t$ in a GRU unit to convey information between two successive time steps.

2.3.4 Seq2seq

Seq2seq is a general end-to-end approach for sequence transduction (Sutskever et al., 2014). It aims to transduce (i.e. transform) an input sequence (source) to a new one (target), where both sequences can be

of arbitrary lengths. Examples of transduction tasks include text translation between languages, audio voice translation, audio style transfer, question-answering, dialog generation or motion generation from text for instance.

The architecture of a seq2seq model is an encoder-decoder architecture. An encoder neural network first encodes the variable-length input sequence (source) into a fixed-length embedding vector. The embedding vector is then decoded by the decoder neural network which maps the fixed-length embedding vector to a new variable-length sequence (target).

An illustration of a seq2seq model is proposed in figure 2.4. The proposed illustration highlights the fact that in a seq2seq model, both the input and the output of the model are sequences. A variable-length input sequence (on top) can be encoded into a fixed-size vector (middle) before being decoded to a variable-length ouput sequence (bottom).

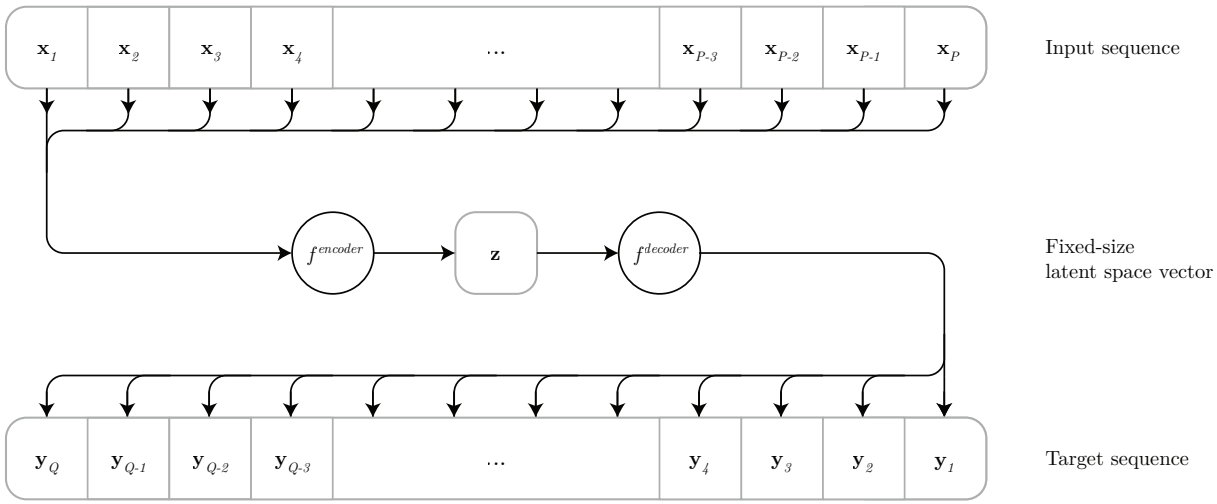


Figure 2.4 – Illustration of a seq2seq approach. An input sequence $(\mathbf{x}_i)_{i\in[\![1,P]\!]}$ of length P is encoded into a fixed-sized latent space vector $\mathbf{z}$ which is later decoded into a target sequence $(\mathbf{y}_i)_{i\in[\![1,P]\!]}$ of length Q. In the original seq2seq model, both the encoder and the decoder are LSTMs.
Figure adapted from (Sutskever et al., 2014).

While (Sutskever et al., 2014) use a (multilayered) LSTMs for the encoder and for the decoder, any architecture that maps a variable-length sequence to a fixed-length vector can be used for the encoder, and any architecture that maps a fixed-length vector to a variable-length sequence can be used for the decoder. As such, other RNN-based architectures and even CNN-based architectures can be used to form a seq2seq model.

2.3.5 Other notable RNN extensions and models

Numerous RNN extensions exist in the research literature. RNN extensions can be general RNN extensions, i.e. extensions of both vanilla RNNs and more complex RNNs like GRUs and LSTMs. RNN extensions can also be (recurrent) model-specific by introducing a new recurrent cell architecture, e.g. by introducing the GRU architecture, or by improving an existing architecture, e.g. by improving a LSTM architecture to a LSTM with peephole connections architecture.

Finally, other classes of models are sometimes directly based on a RNN architecture, e.g. this is the case with Memory Networks[5] (Weston et al., 2014). For instance, the Differentiable Neural Computer model[6] (Graves et al., 2016) is RNN-based since it is actually a RNN where the memory is partly decoupled from the controller neural network that is responsible of writing and reading from the memory, using attention-based mechanisms.

A non-exhaustive list of a few notable RNN extensions and models is proposed:

Bidirectional RNNs Bidirectional RNNs (Schuster et al., 1997) connects two hidden layers of opposite directions (i.e. time is reversed $t \leftarrow -t$ in the RNN equations of half of the neurons) to the same output. Their future input information is reachable from the current state.

Multi-Dimensional RNNs Multi-Dimensional Recurrent Neural Networks (Graves et al., 2007) and Grid Long Short-Term Memory Networks (Kalchbrenner et al., 2015) extend the one dimensional sequence RNN model to multi-dimensional RNN models.

Echo State Networks Echo State Networks (Jaeger et al., 2004) are recurrent neural networks with sparsely connected hidden layers, the connectivity and weights of hidden neurons being randomly fixed.

Clockwork RNNs Clockwork RNNs (Koutnik et al., 2014) aim at capturing different time scales by partitioning the RNN's hidden layers into modules, each module processing inputs at a different clock rate.

Fast weights Fast associative memory models with fast weights (Ba et al., 2016) extend the RNN architecture with a Hebbian connectivity to support a dynamically changing short-term memory of the units' activities. The idea of fast weights is to store short-term memories in a weight matrix, rather than in the hidden units (which are also already in charge of the non-temporary processing) in order to relieve them.

Chunkers Chunkers (Schmidhuber, 1992) are recurrent neural networks trained in order to detect temporal regularities and to learn to use them for identifying relevant points in time, leading to a more resource-efficient processing.

Adaptive computation time Finally, adaptive computation time is a way for RNNs to perform different amounts of computation at each time step (Graves, 2016).

2.4 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) is a class of feedforward neural networks where the neural network uses a convolution operation in place of a matrix multiplication in at least one of its layer. CNNs tend to exhibit good performance on data with a grid-like topology. Data with a grid-like topology

[5] Memory Networks are introduced in section 2.5.3.

[6] Differentiable Neural Computers are introduced in section 2.5.3.

include time-series and images, as they respectively can be viewed as a field of vectors taking values over an evenly spaced 1D grid (time) or 2D grid (spatial pixel grid).

Convolutional neural networks over sequences find temporal regularities. Convolutional neural networks also have fewer parameters than "equivalent" vanilla feedforward neural networks.

2.4.1 Convolution operator

Definition

A convolution between two real-valued functions x and w is another function, written $x * w$, defined by $(x*w)(t) = x(t)*w(t) = \int_{-\infty}^{\infty} x(u)w(t-u)\,du$.

Similarly, the convolution between two vector sequences $\mathbf{x} = (\mathbf{x}_t)_{t\in\mathscr{D}}$ and $\mathbf{w} = (\mathbf{w}_t)_{t\in\mathscr{D}}$ taking values over an evenly-spaced discrete time domain $\mathscr{D}$, e.g. $\mathscr{D} = \mathbb{Z}$ or $\mathscr{D} = [\![1, T]\!]$, is another sequence of vectors taking values over $\mathscr{D}$ and defined by:

$$(\mathbf{x} * \mathbf{w})_t = \sum_{\substack{u+v=t \\ (u,v)\in\mathscr{D}^2}} \mathbf{x}_u \odot \mathbf{w}_v \tag{2.10}$$

where $\odot$ is the element-wise HADAMARD product.

The convolution operator is associative, commutative and bilinear. Convolution is also closely related to the FOURIER transform, which filters a signal into circular paths.

Although the convolution operator is commutative, in convolutional network terminology, the first argument $\mathbf{x}$ to the convolution is referred to as the input and the second argument $\mathbf{w}$ is referred to as the kernel. The convolved output is referred to as the feature map.

A kernel $\mathbf{w}$ with a smaller support (i.e. duration) than the duration of a sequence $\mathbf{x}$ can still be convolved with that sequence $\mathbf{x}$, by padding the kernel with zeros so that both the kernel $\mathbf{w}$ and the sequence $\mathbf{x}$ have the same duration.

In convolutional neural networks, the kernel is usually chosen smaller than the input, rather than using a kernel with the same size as the input. Each parameter of the kernel is used at every position of the input. This choice allows both a more sparse connectivity and parameter sharing across the input locations, because of the reuse of the parameters. Parameter sharing also makes convolution equivariant to translation:

$$T(\mathbf{x} * \mathbf{w}) = T(\mathbf{x}) * \mathbf{w} \tag{2.11}$$

where T is a (temporal) translation operator.

An illustration of a 1D convolution operation is proposed in figure 2.5: at a given time step, to obtain the output of the convolution (bottom of the figure), an input sequence (top of the figure) is convolved with a small kernel (middle of the figure) of size 3 in this case. In order to compute the value of the convolution between the kernel and the input sequence at another time step, the kernel position is shifted along the time axis.

It can be proven that, given some natural constraints, a convolutional structure is actually a necessary condition for equivariance to the action of a compact group (Kondor et al., 2018).

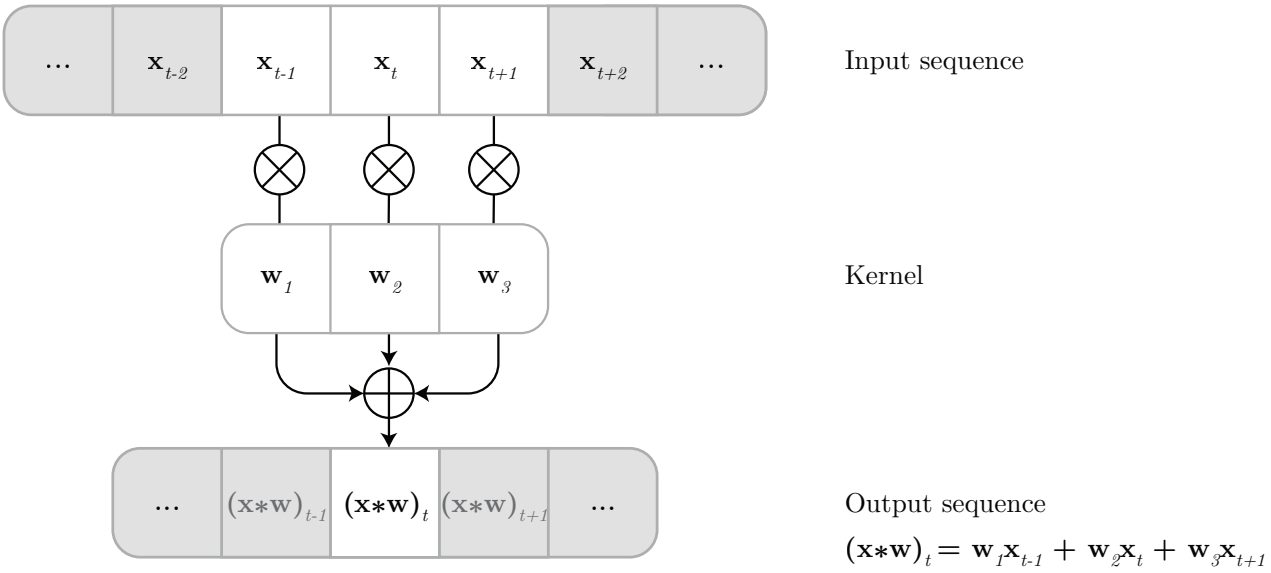


Figure 2.5 – Illustration of a 1D convolution between a sequence and a kernel $\mathbf{w}$ of length 3. Input and output values not related to the the current time step t are greyed out on the illustration.

If $\mathscr{F}$ designates the FOURIER transform, $\mathscr{F}^{-1}$ the inverse FOURIER transform, if $\mathbf{x}$ and $\mathbf{w}$ are square-summable, and if $\mathbf{x}$ and $\mathbf{w}$ have FOURIER transforms $\mathscr{F}(\mathbf{x})$ and $\mathscr{F}(\mathbf{w})$ respectively, then

$$\mathbf{x} * \mathbf{w} = \mathscr{F}^{-1}(\mathscr{F}(\mathbf{x})\mathscr{F}(\mathbf{w})) \tag{2.12}$$

Graph convolutions

The definition of the convolution operator can then be extended to graphs, using the former equation and the graph FOURIER transform $\mathscr{GF}$. On a graph, a convolution can be defined as:

$$\mathbf{x} * \mathbf{w} = \mathscr{GF}^{-1}(\mathscr{GF}(\mathbf{x}) \odot \mathscr{GF}(\mathbf{w})) \tag{2.13}$$

As the graph FOURIER transform itself is defined on values of the spectrum of the graph Laplacian, this definition of a graph convolution is spectral and therefore depends on the exact graph topology. However, one can define a graph convolution[7] using message-passing spatial approaches that do not require a spectral operator (Wu et al., 2019). The definition of the convolution operator can also be extended to other types of input data, e.g. to 2D-grid data (images) or to functions on manifolds (Kondor et al., 2018).

To summarize, one can define a convolution operator over both regular (grid-like) and irregular (graph-like) domains. This convolution operator filters the input using a kernel. Due to its properties, the convolution operator is parameter-efficient and good at extracting patterns from the data.

[7] Graph convolutional neural networks (GCNNs), i.e. CNNs that operate on graphs, can be defined using such a graph convolution operator.

Extensions of the definition

The definition of the convolution operation can be extended, and many extensions of the convolution operation (or of the convolutional layer) exist in the literature. These extensions add new domains for the convolution operation, e.g. convolutions over graphs, or propose to add additional parameters to the convolution operation. The most commonly used extensions include: strided convolutions, dilated convolutions, transposed convolutions, separable convolutions, deformable convolutions and causal (or temporal) convolutions.

Strided convolutions A stride is the step size used for the kernel when traversing a sequence: instead of using a translation stride (step) of 1, other values can be used, e.g. a stride of 2 can be used for downsampling a sequence without of having to pooling. Strided convolutions are convolutions that use a stride value different than 1.

Dilated convolutions Dilated convolutions, also known as *à trous* convolutions or atrous convolutions[8], introduce another parameter called the dilation rate. The dilatation rate is the spacing between the values in a kernel. Dilated convolutions have a wider field of view than standard convolutions at the same computational cost.

Transposed convolutions Transposed convolution, also named fractionally-strided convolution (Dumoulin et al., 2016) or deconvolution (Long et al., 2015), are convolutions that increase the input sequence duration rather than decreasing it, as standard convolutions do.

Separable convolutions There are two main types of separable convolutions: spatial separable convolutions, and depthwise separable convolutions (Chollet, 2017). Both of them aim at reducing the number of parameters in a convolution kernel, when the kernel dimension is greater than 1 (which is not the case for sequential data, by definition). A spatial separable convolution separates a kernel into two smaller kernels. A depthwise separable convolution splits a kernel into two separate kernels that perform two convolutions: a depthwise convolution and a pointwise convolution.

Deformable convolutions Convolutions are computed on a grid (a 1-dimensional temporal grid in the sequence case, or a 2-dimensional spatial grid in the image case for instance). In Deformable Convolutional Networks (Dai et al., 2017), that grid is deformed: each point that forms the grid is moved by a learnable offset; the convolution computations being performed on these moved grid points.

Causal (or temporal) convolutions A causal convolution[9] is a convolution where future values are masked and only past values are used: at each step t the sequence values indexed by t' where $t' < t$ are used for the convolution computation, the other ones indexed by t'' where $t'' \geqslant t$ being set to zero). An efficient way to compute causal convolutions consists in shifting and padding the input sequence by the kernel size and then undo the shifting. Causal convolutions

[8] While being frequently encountered, *atrous* is actually a spelling mistake, the correct form being *à trous*. In French, *à trous* means *with holes*.

[9] Causal convolutions are sometimes -and often ambiguously- referred to as "temporal convolutions".

can be used, for instance, for applications involving temporal sequences if the application requires the computation to be an online computation, i.e. a computation where an output is computed at/for each input time step. An example of causal convolution use can be found in the Temporal Convolutional Networks (TCNs[10]) (Bai et al., 2018). TCNs are made of residual[11] convolution blocks where the convolution is dilated, causal and normalized using weight normalization (Salimans et al., 2016). The idea of masking future information is also used in numerous autoregressive deep learning models like in PixelCNN (Van den Oord et al., 2016) and Wavenet (Oord et al., 2016), or in reinforcement learning models -for problems sensitive to the positional dependency- like in SNAIL (Mishra et al., 2017) for instance.

2.4.2 Pooling

A pooling function is a function that replaces a layer output at a specific location by a summary statistic of the values taken by the neighbors of that location.

For grid-like domains, the neighborhood of a location is often defined as a rectangular neighborhood centered on the location (for instance a few time steps before and after a specific time step, if the domain is time).

Pooling summarizes features over a whole neighborhood, helping the representation to become more invariant to small translations of the input. This invariance is often gained at the cost of losing the precise location (in time or in space) of information.

The two most widely used pooling functions are Maximum Pooling (maxpool) (Y.-T. Zhou et al., 1988) and Average Pooling (avgpool). For a given neighborhood of input values $\mathcal{N} = x_1, x_2, \ldots, x_p$, they are respectively defined by:

$$\text{maxpool}(\mathcal{N}) = \max_{x_i \in \mathcal{N}} x_i$$
$$\text{avgpool}(\mathcal{N}) = \frac{1}{p} \sum_{x_i \in \mathcal{N}} x_i$$

Depending on the data and the cardinality of the neighborhood, either max or average pooling may perform best of the two methods (Boureau et al., 2010). Empirical comparisons of the two methods show that max pooling often outperforms average pooling in image recognition problems, however that result may not hold for any type of data.

Since pooling tends to lose the precise location of information, architectures that require such precise location do not use pooling (Oord et al., 2016).

[10] Not to be confused with Time-Contrastive Networks (Sermanet et al., 2018) (referred to as TCNs too...) or with standard non-causal CNNs over time, also often described as Temporal Convolutional Networks (referred to as TCNs too...).

[11] Residual connections and skip-connections are introduced in section 2.4.3.

2.4.3 Convolutional Neural Networks Architecture

A convolutional neural network (CNN) is a feedforward neural networks that uses a convolution operation in place of a matrix multiplication in at least one of its layer.

General architecture of a 1D CNN

A traditional CNN model frequently involves a sequence of blocks involving a convolution layer, a (e.g. batch) normalization layer, a non-linearity layer, a regularization layer[12] and a pooling layer, as illustrated in figure 2.6. Except the convolution layer and the activation layer, any of the remaining layers, e.g. the pooling layer, can be dropped or replaced by a more appropriate layer if needed. These blocks are often followed by hidden dense layers. The convolution blocks serve as feature extractors, whereas the dense hidden layers can be seen as a classifier or a regressor, depending on the network's target(s). The final fully-connected layers not related to convolutions can be replaced with any other type of layers, the correct layer type depending on the task[13].

Such a CNN model is trainable end-to-end, as a composition of layers that are all differentiable.

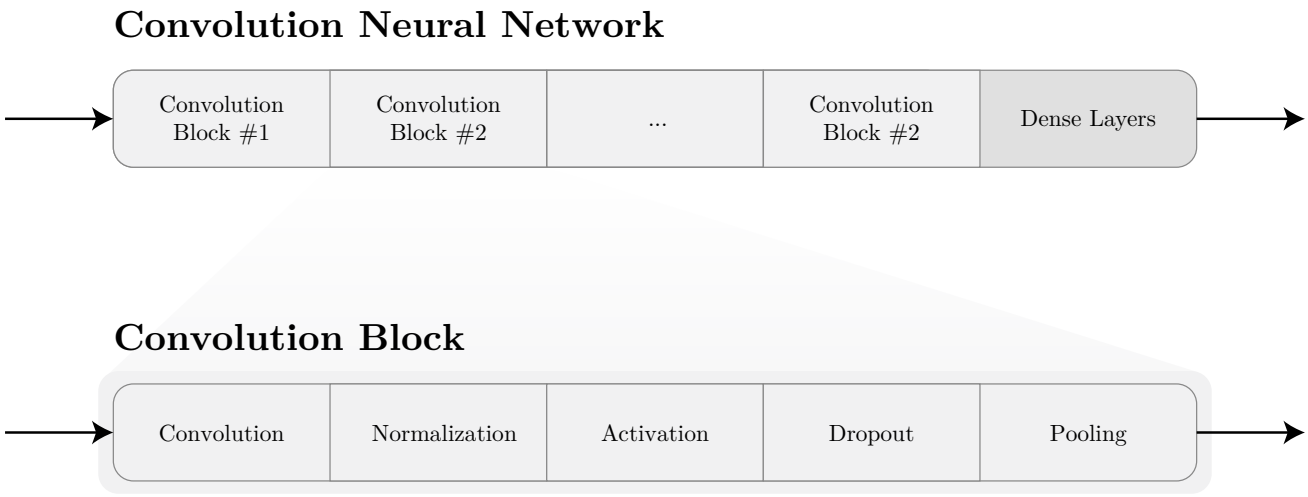


Figure 2.6 – Illustration of a typical CNN architecture (top) and a convolution block (bottom)

When their input data are time-series or temporal sequences, 1D CNNs are usually referred to as Temporal Convolutional Neural Networks (TCNs). Temporal convolutional neural networks perform their convolution and pooling operations over the time dimension. They may use either causal or non-causal convolutions, depending on the task to perform. Causal-convolutions are usually used for online tasks (Bai et al., 2018), while standard-convolutions are used for offline tasks.

[12] While regularization is often performed at the model's loss function level, it can also be performed at an architectural connectivity level, e.g. dropout (N. Srivastava et al., 2014), or at a weight level, e.g. spectral normalization (Miyato et al., 2018).

[13] E.g. both recurrent layers and convolutional layers can be used, if the CNN model is a machine translation model.

Advantages of a 1D CNN architecture

Temporal convolutional neural networks are well suited for time-series data: time can be viewed as a 1D grid and CNNs are known to have good performances on grid-like domains (Goodfellow et al., 2016).

In temporal convolutional neural networks, each neuron is not connected to all neurons of the previous layer, contrary to fully-connected layers[14]. Local connections (with only a small number of neurons) is effective in reducing parameters and speed up convergence. A group of connections can share the same weights, thus reducing parameters further.

Since the parameters of the convolution kernels are shared across the convolution domain, CNNs can process both fixed-length inputs and variable-length inputs.

The multi convolutional layer hierarchy of a temporal convolutional neural networks give them the ability to learn complex temporal hierarchies with increasing levels: first convolutional layers will extract low-level patterns of the raw input sequence over a short time horizon, while last convolutional layers extract high-level and more task-related patterns of the sequence over a wide time-horizon.

As such, temporal convolutional neural networks can advantageously be used for problems involving time. For instance, 3D convolutions could theoretically be used in tasks involving both temporal information and spatial information,. However 3D convolutions (k^3 parameters, without bias) are more computationally expensive than 1D convolutions (k kernel parameters, without bias) and 2D convolutions (k^2 kernel parameters, without bias) while spatial and temporal dynamics are not necessarily related in all tasks. As such, a better solution often consists in using separate convolutions: convolutions over time on one hand, alongside with 2D spatial convolutions on the other hand. One good architecture typically intertwines temporal convolutions and spatial convolutions, i.e. the architecture consists in a sequence of $STSTST\dots$ constitutional layers, where S denotes a spatial convolution, and T and temporal convolution, as described in (Pigou et al., 2015).

Finally, temporal convolutional neural networks inherit properties of the convolution operator (presented in 2.4.1), the most notable one being equivariance to time translation: a temporal shift in a convolutional layer input sequence produces an identical temporal shift in the output of that layer.

Skip connections and Residual connections

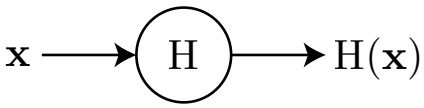
Neural networks tend to have complex loss landscapes. The deeper the network, the more chaotic the loss landscapes become. Having highly non-convex loss landscapes raises at least two issues (H. Li et al., 2018). The first one is generalization: with such landscapes, a neural network will have trouble to generalize since a small change in the input distribution might have dramatic consequences on the calculated loss distribution, due to the chaotic loss landscape. The second one, is more practical: the neural network will be harder or even impossible to train.

Skip connections or residual connections (K. He et al., 2016; G. Huang et al., 2017; R. K. Srivastava et al., 2015) allow to overcome this pitfall and substantially improve the training of very deep neural networks. Skip-connections promote flat minimizers of the loss landscape.

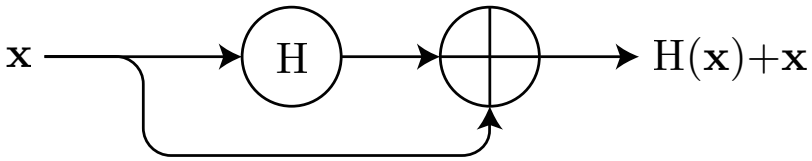
Skip connections are additional identity[15] connections between nodes from different layers; skip

[14] Fully-connected layers are often used in attention-based models for instance.

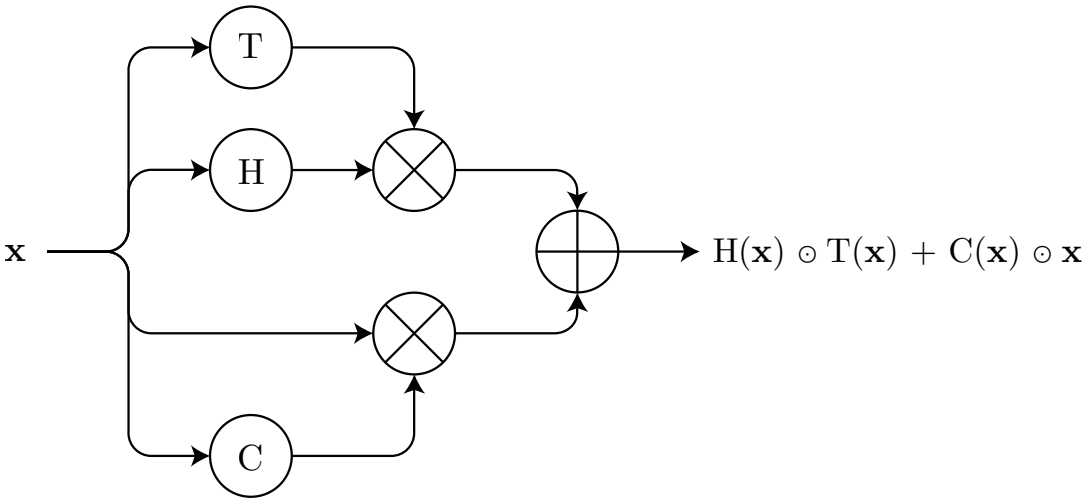
[15] Skip-connections usually use the identity operator, unless another operator is explicitly stated.



(a) Vanilla hidden layer



(b) Hidden layer with a skip-connection



(c) Hidden layer with a highway circuit

Figure 2.7 – Illustration of skip connections and highway circuits
Figure inspired by a figure from (Zilly et al., 2017).

connections skip one or more nonlinear processing layers. A residual block is a block where a layer H_θ is skipped with a residual connection. An illustration of a skip-connection is proposed in figure 2.7 (b): an identity connection between the input and the output nodes allow the incoming information to skip the neural network H_θ, whereas this was not the case in the original case depicted in figure 2.7 (a).

In a traditional neural network, a layer learns the true output $\mathbf{y} = H_\theta(\mathbf{x})$ whereas in a neural network featuring a residual block the same layer learns -hence the name- the residual

$$R_\theta(\mathbf{x}) = H_\theta(\mathbf{x}) - \mathbf{x} \tag{2.14}$$

Residual blocks and skip connections can be introduced in any neural network architecture, regardless of the computational graph topology. Besides helping the gradients to flow more easily during the backpropagation, they introduce a way to transform or bypass the signal. As such, it is worth noting that the Highway Networks architecture (R. K. Srivastava et al., 2015), which inspired the design of residual blocks (K. He et al., 2016), makes use of gating functions and is itself inspired from the design

of a LSTM cell. In a traditional neural network, a layer H_θ outputs $\mathbf{y} = \mathbf{H} = H_\theta(\mathbf{x})$ whereas in a neural network featuring a highway circuit the equations are:

$$\begin{aligned} \mathbf{H} &= H_{\theta_H}(\mathbf{x}) \\ \mathbf{T} &= T_{\theta_T}(\mathbf{x}) \\ \mathbf{C} &= C_{\theta_C}(\mathbf{x}) \\ \mathbf{y} &= \mathbf{H} \odot \mathbf{T} + \mathbf{C} \odot \mathbf{x} \end{aligned} \tag{2.15}$$

where T_θ and C_θ are two gating functions respectively named the transform gate and the carry gate. An illustration of how a highway circuit both transforms (top gate in the figure) and carries (bottom gate in the figure) the original input is proposed in figure 2.7 (c).

2.5 Attention-based Associative Memory

2.5.1 Attention Mechanisms in Neural Networks

Attention mechanisms are mechanisms that select -and focus on- a specific aspect of their input, while ignoring other perceivable present information in the input.

Attention over sequences

The first attention mechanism called as such was proposed in (Bahdanau et al., 2014) to overcome the incapability of remembering long sentences[16] in the context of machine translation with seq2seq models. This first attention mechanism proposed to allow the seq2seq decoder to access the entire *encoded* sequence -rather than only the last encoded value- and to selectively decide what values (or time steps) are important.

The most frequently encountered attention mechanism takes two sequences $\mathbf{s}^{(a)} = \left(\mathbf{x}_1^{(a)}, \mathbf{x}_2^{(a)}, \ldots, \mathbf{x}_{T^{(a)}}^{(a)}\right)$ and $\mathbf{s}^{(b)} = \left(\mathbf{x}_1^{(b)}, \mathbf{x}_2^{(b)}, \ldots, \mathbf{x}_{T^{(b)}}^{(b)}\right)$ as input, and scores all the couples $\left(\mathbf{x}_i^{(a)}, \mathbf{x}_j^{(b)}\right)$ using a scoring function γ, where $\mathbf{x}_i^{(a)}$ is a vector from the first sequence and $\mathbf{x}_j^{(b)}$ is a vector from the second sequence. The resulting scores can be stored in a matrix that is then used for task-related purposes, e.g. for sequence alignment (Bahdanau et al., 2014).

Numerous scoring functions γ can be used. For instance, the cosine: $\gamma(\mathbf{u}, \mathbf{v}) = \cos(\mathbf{u}, \mathbf{v})$ or the dot-product: $\gamma(\mathbf{u}, \mathbf{v}) = \mathbf{u}^\top \mathbf{v}$ can be used as scoring functions. Scoring functions can include parameters to learn. This is the case for the following tanh scoring function: $\gamma(\mathbf{u}, \mathbf{v}) = \theta_a^\top \tanh(\mathbf{W}_a[\mathbf{u}; \mathbf{v}])$ where θ_a and $\mathbf{W}_a$ are weight matrices to be learned in the alignment model and $[\cdot;\cdot]$ is the concatenation operation.

Attention scoring functions can also have a single input, when the two input sequences of the scoring function actually relate to the same input sequence. Attention mechanisms with such as scoring function are called self-attention mechanisms.

[16] This is likely due to the lack of truly *long-term* memory in LSTMs, despite their promises, as noted in (Bai et al., 2018).

An attention function Γ has three inputs. The first two inputs are the inputs of the scoring function γ, while the third input is directly modulated by the output of the scoring function γ. For instance the scaled dot-product attention $\Gamma_{\text{scaled dot-product}}$ has three inputs: a key $\mathbf{K}$, a value $\mathbf{V}$, and a query $\mathbf{Q}$. The output of the scaled dot-product attention is a weighted sum of the value $\mathbf{V}$, the weight assigned to each value being determined by the dot-product of the query $\mathbf{Q}$ with all the keys $\mathbf{K}$. The mathematical formulation of the scaled dot-product attention is given in equation 2.16, where n is the dimension of both $\mathbf{K}$ and $\mathbf{V}$.

$$\Gamma_{\text{scaled dot-product}}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{n}}\right)\mathbf{V} \tag{2.16}$$

Multi-head attention is a simple ensembling technique that consist in computing attention on different representation subspaces and jointly attending the computed attentions. For instance, the mathematical formulation of the multi-head scaled dot-product attention mechanism is given in equation 2.17, where p is the heads count, $[\cdot;\cdot]$ is the concatenation operation and $\mathbf{W}^O$ and $\mathbf{W}_i^Q$, $\mathbf{W}_i^Q$, $\mathbf{W}_i^Q$ ($\forall i \in [\![1,p]\!]$) are matrices of parameters to learn.

$$\begin{aligned} \mathbf{h}_i &= \Gamma_{\text{scaled dot-product}}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) \quad \forall i \in [\![1,p]\!] \\ \Gamma_{\text{multi-head scaled dot-product}}(\mathbf{Q},\mathbf{K},\mathbf{V}) &= [\mathbf{h}_1;\mathbf{h}_2;\ldots;\mathbf{h}_p]\mathbf{W}^O \end{aligned} \tag{2.17}$$

An illustration of the multi-head scaled dot-product attention is proposed in figure 2.8: the right side of the illustration highlights the ensembling aspect of the multi-head scaled dot-product attention module whereas the left side of the illustration depicts the operations performed on the key $\mathbf{K}$, the value $\mathbf{V}$ and the query $\mathbf{Q}$ by each individual scaled dot-product attention module.

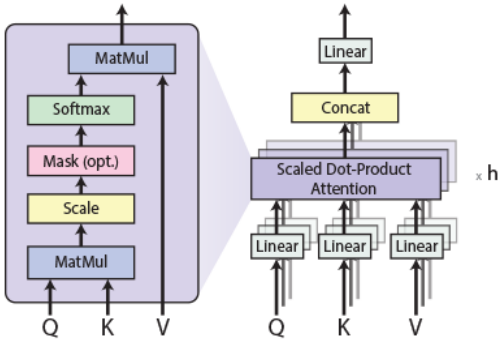


Figure 2.8 – Illustration of a multi-head scaled dot-product attention (right) and scaled dot-product attention (left).
Figure reproduced from (Vaswani et al., 2017).

One possible interpretation of a scaled dot-product attention module is that it implicitly performs selective analogies[17] on a *key-value* dictionary-like structure: it selects aspects of an input data value ($\mathbf{V}$) where the input "type" ($\mathbf{K}$) matches a desired "type" ($\mathbf{Q}$). A multi-head scaled dot-product attention module performs these analogies on different learned representations of the same input.

[17] A basic analogy being: "$\mathbf{V}$ is to $\mathbf{K}$ what the desired output is to $\mathbf{Q}$".

2.5.2 Transformer Networks

The transformer neural network architecture is an encoder-decoder architecture for sequence transduction that is based solely on attention mechanisms, dispensing with recurrence and convolutions entirely (Vaswani et al., 2017). Since the model is based on fully-connected layers, a positional encoding of the input is added to it beforehand, in order to allow the network to extract information from the sequential structure of the input. Both the encoder (Devlin et al., 2018) and the decoder (Radford et al., 2019) of a transformer are composed of repeated modules. Each of these modules essentially consists of the already introduced multi-head attention, and fully-connected layers.

Transformer-based models exhibit state-of-the-art performances in the human language and natural language processing (NLP) domains (Devlin et al., 2018).

2.5.3 Memory Networks

Memory Networks (Weston et al., 2014) are a class of models that provide an explicit memory representation, combined with inference components. Memory Networks are an attempt to externalize and isolate memory abilities of a neural network from the perception processing abilities of the network. As any memory, the external memory can be altered, written to and read from. If the memory network is differentiable, it can be trained end-to-end, allowing for *learned* write and read mechanisms. This allows complex content-addressable retrieval, e.g. with attention mechanisms to orient the input perception towards the contents of the memory.

An illustration of a Memory Network is proposed in figure 2.9. As seen on the figure 2.9, a memory network is consists in four modules I, G, O, R: the module I extracts features from the input, the module G updates the memory recursively: $\mathbf{m} = G(\mathbf{m}, I(\mathbf{x}))$, so that the module O and the module R produce the output using this updated memory. The figure 2.9 highlights the central role of the memory bank $\mathbf{m}$ in Memory Networks. Since the module G updates the memory recursively, Memory Networks could also have been introduced in the RNN section (section 2.3). Memory Networks can be considered as an early attempt at distinguishing the memory-related processing (and storage) from the rest of the processing, in a neural network.

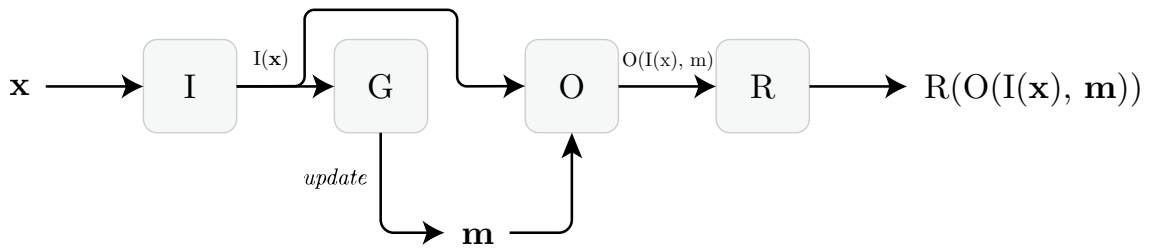


Figure 2.9 – Illustration of a Memory Network. I extracts features from the input, G updates the memory recursively: $\mathbf{m} = G(\mathbf{m}, I(\mathbf{x}))$. O and R produce the output using this updated memory.
Figure reproduced from (Weston et al., 2014).

Neural Turing Machines (NTMs) and Differentiable Neural Computers (DNCs)

Differentiable Neural Computers (DNCs) (Graves et al., 2016) are memory networks inspired from the Von-Neumann architecture; they are an extension of Neural Turing Machines (NTMs) (Graves et al., 2014).

A DNC is a neural network coupled to an external memory matrix. The neural network, called a controller, is responsible for taking input in, reading from memory, writing to memory, and producing an output. Since a DNC is differentiable end-to-end, the controller can be trained to learn how to use the memory. Finally, the DNC model is also Turing complete.

Three kinds of interactions exist between the controller and the memory: content lookup mechanisms, temporal memory linkage and dynamic memory allocation. Content can be written to the memory, as well as erased from the memory. A temporal link matrix additionally stores information about the order of writes, giving a DNC the native ability to recover sequences in the order in which it wrote them. Reading from the memory can be done either based on content lookup (i.e. searching content by similarity) or based on the order from the temporal link matrix.

In practice, a LSTM or a GRU is often used as a controller.

It is worth noting that attention mechanisms are used in all of the three possible interaction mechanisms between the controller and memory: in content lookup mechanisms, in temporal link matrix recording mechanisms and finally in memory allocation for writing mechanisms.

An illustration of a differentiable neural computer is proposed in figure 2.10. The illustration summarizes all memory access mechanisms (reading and writing heads with content lookup, temporal linkage and dynamic memory allocation) into a single block for clarity purposes. As one can see in figure 2.10, a DNC is a RNN since it conveys an external memory matrix[18] across time steps to perform computations on its input at a given time step. As such, the DNC could also have been introduced in the RNN section (section 2.3). Though, the DNC's memory-controller interactions (which are not detailed in figure 2.10 for the sake of simplicity) essentially make extensive use of attention-based mechanisms.

[18] From one time step to another, (1) a memory state, (2) contents read from the memory, and (3) a control state are conveyed.

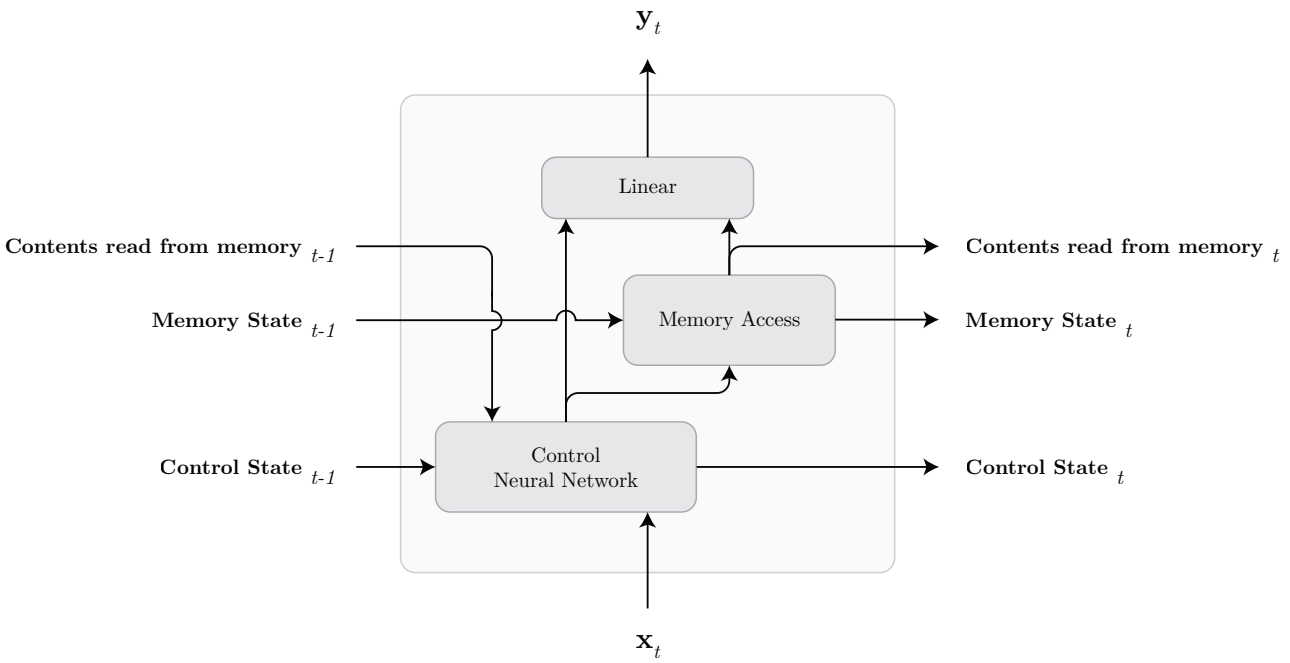


Figure 2.10 – Illustration of a Differentiable Neural Computer.
Figure reproduced from the code repository of (Graves et al., 2016).

2.6 Conclusion

In this chapter, we presented an overview of neural network module architectures used for sequence modeling. Understanding attention mechanisms, recurrent neural networks and convolutional neural networks is crucial to understand existing deep learning models that work on sequential data, and to devise new ones.

Prior to this thesis, RNN-based approaches have long -and clearly- been considered as the *de facto* standard approaches to obtain state-of-the-art performance on sequences with neural networks. However, this assumption has been drastically challenged by the recent research literature. Current state-of-the-art models for text sequences are attention-based models that do not use recurrent cells (Brown et al., 2020). In other domains involving sequential data, like the audio domain or the human motion pose domain, current state-of-the-art models frequently alternate between recurrent-only models and convolution-only models.

Fully-connected, recurrent and convolutional neural networks are all able to model sequences. However the performance of fully-connected neural networks clearly lags behind the performance of RNNs and CNNs when no additional constraint is imposed. Temporal order is lost with a basic dense-only architecture, whereas the design of both the CNN architecture and the RNN architecture allow them to find temporal regularities in the data. Attention mechanisms select and focus on a specific aspect of their input, while ignoring other perceivable present information in the input. Attention mechanisms can be used with most feedforward neural networks architectures, including fully-connected neural networks, convolutional neural networks and recurrent neural networks architectures.

After having dived into the details of each approach, it is worth noting afterwards that all these approaches are actually very closely related to each other, even if their principle and exact inner

operating mechanism may differ. When it comes to sequence modeling, a CNN can be considered as a special case of a TDNN. Reversely, a TDNN can also be considered as a special case of a CNN. A convolutional structure is known to be a necessary condition for temporal equivariance, which is usually a desired property when it comes to sequences. The major difficulty when using a TDNN is to decide -at an architectural level- where a TDNN has to attend to, temporally speaking. An attention mechanism can be viewed as a way to "attend to" only a certain aspect -e.g. a portion- of a sequence rather than the entirety of the sequence: in a sense, an attention mechanism try to filter the information that needs to be processed later on. While this filtering can partly be performed at an architectural level, e.g. based on similarities, attention mechanisms also very often learn the correct filtering with parameters during training, at the expense of additional model parameters. Finally, RNNs only consider one input sequence value at a time, rather than examining the full input at once at a sequence level, as attention mechanisms do, or rather than examining the input at a sliding window level, as TDNNs and CNNs do. In order to still be able to account for the temporal context of the input, RNNs rely on an external memory. RNNs -and memory models that are based upon them- write and read relevant information to that memory. Regardless of performance, the question of whether a deep learning model should, or should not, require an external memory is still an open question for most applications.

Chapter 3

Vehicle Control with Feedforward Neural Networks

"Quand on lui réclamait des solutions parfaites, qui écarteraient tous les risques : « C'est l'expérience qui dégagera les lois, répondait-il, la connaissance des lois ne précède jamais l'expérience. »"

Saint-Exupéry

Contents

3.1 Introduction

In this chapter, we focus on the problem of controlling a car-like vehicle in highly dynamic situations, for instance to perform evasive manoeuvrers in face of an obstacle.

We propose to use deep neural networks to implicitly model highly coupled vehicular dynamics, and perform low-level control in real-time. In order to do so, we train a deep neural network to output low-level controls (wheels torque and steering angle) corresponding to a given initial vehicle state and target trajectory. Compared to classical MPC frameworks which require integrating dynamic equations on-line, this approach allows to perform this task off-line and use only simple mathematical operations on-line, leading to much faster computations.

This chapter is organized as follows: a brief summary of our contributions in proposed in section 3.1.1. Motivations for the approach we introduce are discussed in section 3.1.2. Related works that exist in the research literature are then discussed in section 3.2. Section 3.3 presents the vehicle model used to generate the training dataset and to simulate the vehicle dynamics on a test track. Section 3.4 introduces two artificial neural networks architectures used to generate the control signals for a given target trajectory, and describes the training procedure used in this chapter. Section 3.5 compares the performance of these two networks, using simulation on a challenging test track. A comparison to conventional decoupled controllers is also provided. Finally, a conclusion is proposed in section 3.6.

3.1.1 Contributions summary

Besides the overall approach and (hyper-)parameters, our contributions more specifically are: learning the (inverse) dynamics of a wheeled vehicle with a neural network and using such a neural network as a vehicle controller[1], the creation of a public vehicle dynamics dataset for control (as described in section 3.4.1) alongside with an open-sourced vehicle dynamics simulator[2] based on the dynamics from (Polack et al., 2017), the two neural network architectures proposed in sections 3.4.2 and 3.4.3, and all the experiments performed in section 3.5. This chapter first appeared in (Devineau et al., 2018c).

3.1.2 Motivations

The recent development of deep learning has led to dramatic progress in multiple research fields, and this technique has naturally found applications in autonomous vehicles. The use of deep learning to perform perceptive tasks such as image segmentation has been widely researched in the last few years, and highly efficient neural network architectures are now available for such tasks. More recently, several teams have proposed taking deep learning a step further, by training so-called "end-to-end" algorithms to directly output vehicle controls from raw sensor data (see, in particular, the seminal work in (Bojarski et al., 2016)).

Although end-to-end driving is highly appealing, as it removes the need to design motion planning

[1] Thus effectively isolating control from perception and planning.

[2] The code for the vehicle dynamics simulator model is open source and published at `https://github.com/guillaumephd/vehicle-dynamics-model` (Vehicle dynamics simulator code by Devineau, Polack and Altché based on Polack's original work).

and control algorithms by hand, handing the safety of the car occupants to a software operating as a black box seems problematic. A possible workaround to this downside is to use "forensics" techniques that can, to a certain extent, help understand the behavior of deep neural networks (Castelvecchi, 2016).

We choose a different approach consisting in breaking down complexity by training simpler, mono-task neural networks to solve specific problems arising in autonomous driving; we argue that the reduced complexity of individual tasks allows much easier testing and validation.

3.2 Related Works

A particular challenge in highly dynamic situations, for instance to perform evasive manoeuvrers in face of an obstacle, is the important coupling between longitudinal and lateral dynamics when nearing the vehicle's handling limits, which requires highly detailed models to properly take into account (Gillespie, 1997). However, precisely modeling this coupling involves complex non-linear relations between state variables, and using the resulting model is usually too costly for real-time applications. For this reason, most references in the field of motion planning mainly focus on simpler models, such as point-mass or kinematic bicycle (single track), which are constrained to avoid highly coupled dynamics (Polack et al., 2018). Similarly, research on automotive control usually treats the longitudinal and lateral dynamics separately in order to simplify the problem (Khodayari et al., 2010).

Although these simplifications can yield good results in standard on-road driving situations, they may be problematic for vehicle safety when driving near its handling limits, for instance at high speed or on slippery roads. To handle such situations, some authors have proposed using Model Predictive Control (MPC) with a simplified, coupled dynamic model (Falcone et al., 2007) which is limited to extremely short time horizons (a few dozen milliseconds) to allow real-time computation. Other authors have proposed to model the coupling between longitudinal and lateral motions using the concept of "friction circle" (Kritayakirana et al., 2012), which allows precisely stabilizing a vehicle in circular drifts (Goh et al., 2016). However, the transition towards the stabilized drifting phase – which is critical in the ability, *e.g.*, to perform evasive manoeuvrers – remains problematic with this framework.

Several authors have already proposed a divide-and-conquer approach by using machine learning on specific sub-tasks instead of performing end-to-end computations, and in particular on the case of motion planning and control. For instance, (Drews et al., 2017) used a Convolutional Neural Network (CNN) to generate a cost function from input images, which is then used inside an MPC framework for high-speed autonomous driving; however, this approach has the same limitations as model predictive control. Other approaches, such as the one in (Se-Young Oh et al., 2000), used reinforcement learning to output steering controls for a vehicle, but were limited to low-speed applications. In (Punjani et al., 2015) the authors used a Rectified Linear Unit (ReLU) network model to identify the dynamics of a helicopter in order to predict its future accelerations, but this model has not been used for control.

Closer to our work, (Rivals et al., 1994) trained neural networks integrating a priori knowledge of the bicycle model for decoupled longitudinal and lateral control of a vehicle; in (Y. Chen et al., 2017), authors used supervised learning to generate lateral controls for truck and integrated a control barrier function to ensure the safety of the system. In (Cui et al., 2017) the authors coupled a standard control and an adaptive neural network to compensate for unknown perturbations in order to perform trajectory

tracking for autonomous underwater vehicle.

To the best of our knowledge, deep neural networks have not been used in the literature for the coupled control of wheeled vehicles.

3.3 Vehicle simulator: 9 DoF vehicle model

In this section, we present the 9 Degrees of Freedom (9 DoF) vehicle model (Polack et al., 2017) which is used both to generate the training and testing dataset, and as a simulation model to evaluate the performance of the deep-learning-based controllers. All the equations presented in this section 3.3 are introduced in (Polack et al., 2017). They are only reproduced here for the sake of clarity and completeness, and with the author's acknowledgement.

The Degrees of Freedom comprise 3 DoF for the vehicle's motion in a plane ($V_x, V_y, \dot{\psi}$), 2 DoF for the carbody's rotation ($\dot{\theta}, \dot{\phi}$) and 4 DoF for the rotational speed of each wheel ($\omega_{fl}, \omega_{fr}, \omega_{rl}, \omega_{rr}$). The model takes into account both the coupling of longitudinal and lateral slips and the load transfer between tires. The control inputs of the model are the torques T_{ω_i} applied at each wheel i and the steering angle δ of the front wheel. The low-level dynamics of the engine and brakes are not considered here. The notations are given in Table 3.1 and illustrated in Figure 3.1.

Remark: the subscript $i = 1..4$ refers respectively to the front left (fl), front right (fr), rear left (rl) and rear right (rr) wheels.

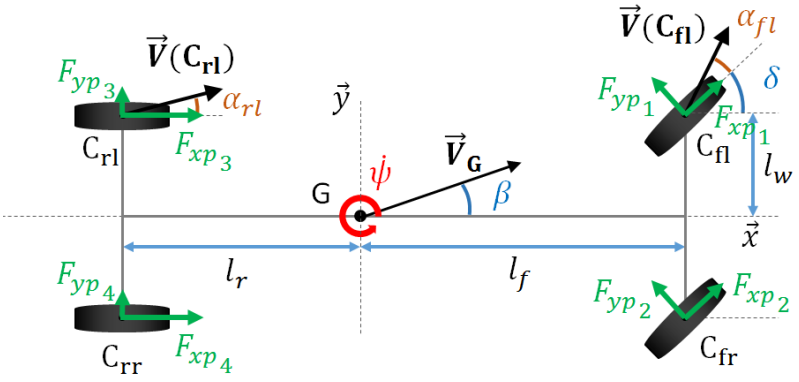


Figure 3.1 – Vehicle model and notations. Figure reproduced from (Polack et al., 2017).

Several assumptions were made for the model:

- Only the front wheels are steerable.
- The roll and pitch rotations happen around the center of gravity.
- The aerodynamic force is applied at the height of the center of gravity. Therefore, it does not involve any moment on the vehicle.
- The slope and road-bank angle of the road are not taken into account.

Table 3.1 – Notations

X, Y	Position of the vehicle in the ground frame
θ, ϕ, ψ	Roll, pitch and yaw angles of the carbody
V_x, V_y	Longitudinal and lateral speed of the vehicle in its inertial frame
M_T	Total mass of the vehicle
I_x, I_y, I_z	Inertia of the vehicle around its roll, pitch and yaw axis
I_{r_i}	Inertia of the wheel i
T_{ω_i}	Total torque applied to the wheel i
F_{xp_i}, F_{yp_i}	Longitudinal and lateral tire forces generated by the road on the wheel i expressed in the tire frame
F_{x_i}, F_{y_i}	Longitudinal and lateral tire forces generated by the road on the wheel i expressed in the vehicle frame (x,y)
F_{z_i}	Normal reaction forces on wheel i
l_f, l_r	Distance between the front (resp. rear) axle and the center of gravity
l_w	Half-track of the vehicle
h	Height of the center of gravity
r_{eff}	Effective radius of the wheel
ω_i	Angular velocity of the wheel i
V_{xp_i}	Longitudinal speed of the center of rotation of wheel i expressed in the tire frame

3.3.1 Vehicle dynamics

Equations (3.1a-3.1e) give the expression of the vehicle dynamics:

$$M_T\dot{V}_x = M_T\dot{\psi}V_y + \sum_{i=1}^{4} F_{x_i} - F_{aero} \tag{3.1a}$$

$$M_T\dot{V}_y = -M_T\dot{\psi}V_x + \sum_{i=1}^{4} F_{y_i} \tag{3.1b}$$

$$I_x\ddot{\theta} = l_w(F_{z_1} + F_{z_3} - F_{z_2} - F_{z_4}) + h\sum_{i=1}^{4} F_{y_i} \tag{3.1c}$$

$$I_y\ddot{\phi} = l_r(F_{z_3} + F_{z_4}) - l_f(F_{z_1} + F_{z_2}) - h\sum_{i=1}^{4} F_{x_i} \tag{3.1d}$$

$$\begin{aligned} I_z\ddot{\psi} &= l_f(F_{y_1} + F_{y_2}) - l_r(F_{y_3} + F_{y_4}) \\ &+ l_w(F_{x_2} + F_{x_4} - F_{x_1} - F_{x_3}) \end{aligned} \tag{3.1e}$$

F_{x_i} and F_{y_i} denote respectively the longitudinal and the lateral tire forces expressed in the vehicle frame; $F_{aero} = \frac{1}{2}\rho_{air}C_xSV_x^2$ denote the aerodynamic drag forces with ρ_{air} the mass density of air,

C_x the aerodynamic drag coefficient and S the frontal area of the vehicle; F_{z_i} denote the damped mass/spring forces depending on the suspension travel ζ_i due to the roll θ and pitch ϕ angles according to Equation (3.1f). The parameters k_s and d_s are respectively the stiffness and the damping coefficients of the suspensions.

$$F_{z_i} = -k_s\zeta_i(\theta,\phi) - d_s\dot{\zeta}_i(\theta,\phi) \tag{3.1f}$$

The position (X,Y) of the vehicle in the ground frame can then be derived using Equations (3.1g) and (3.1h).

$$\dot{X} = V_x\cos\psi - V_y\sin\psi \tag{3.1g}$$
$$\dot{Y} = V_x\sin\psi + V_y\cos\psi \tag{3.1h}$$

3.3.2 Wheel dynamics

The dynamics of each wheel $i = 1..4$ expressed in the pneumatic frame is given by Equation (3.2):

$$I_r\dot{\omega}_i = T_{\omega_i} - r_{eff}F_{xp_i} \tag{3.2}$$

3.3.3 Tire dynamics

The longitudinal force F_{xp_i} and the lateral force F_{yp_i} applied by the road on each tire i and expressed in the pneumatic frame are functions of the longitudinal slip ratio τ_{x_i}, the side-slip angle α_i, the normal reaction force F_{z_i} and the road friction coefficient μ:

$$F_{xp_i} = f_x(\tau_{x_i},\alpha_i,F_{z_i},\mu) \tag{3.3a}$$
$$F_{yp_i} = f_y(\alpha_i,\tau_{x_i},F_{z_i},\mu) \tag{3.3b}$$

The longitudinal slip ratio of the wheel i is defined as following:

$$\tau_{x_i} = \begin{cases} \frac{r_{eff}\omega_i - V_{xpi}}{r_{eff}|\omega_i|} & \text{if } r_{eff}\omega_i \geq V_{xp_i} \text{ (Traction phase)} \\ \frac{r_{eff}\omega_i - V_{xpi}}{|V_{xpi}|} & \text{if } r_{eff}\omega_i < V_{xpi} \text{ (Braking phase)} \end{cases} \tag{3.4}$$

The lateral slip-angle α_i of tire i is the angle between the direction given by the orientation of the wheel and the direction of the velocity of the wheel (see figure 3.1):

$$\alpha_f = \delta - \operatorname{atan}\left(\frac{V_y + l_f\dot{\psi}}{V_x \pm l_w\dot{\psi}}\right);\ \alpha_r = -\operatorname{atan}\left(\frac{V_y - l_r\dot{\psi}}{V_x \pm l_w\dot{\psi}}\right) \tag{3.5}$$

In order to model the functions f_x and f_y, we used the combined slip tire model presented by Pacejka in (Pacejka, 2002) (cf. Equations (4.E1) to (4.E67)) which takes into account the interaction between the longitudinal and lateral slips on the force generation. Therefore, the friction circle due to the laws of friction (see Equation (3.6)) is respected. Finally, the impact of load transfer between tires is also taken

into account through F_z.

$$||\vec{F}_{xp} + \vec{F}_{yp}|| \leq \mu ||\vec{F}_z|| \tag{3.6}$$

Lastly, the relationships between the tire forces expressed in the vehicle frame F_x and F_y and the ones expressed in the pneumatic frame F_{xp} and F_{yp} are given in Equation (3.7):

$$\begin{aligned} F_{x_i} &= (F_{xp_i}\cos\delta_i - F_{yp_i}\sin\delta_i)\cos\phi - F_{z_i}\sin\phi && \text{(3.7a)} \\ F_{y_i} &= (F_{xp_i}\cos\delta_i - F_{yp_i}\sin\delta_i)\sin\theta\sin\phi && \text{(3.7b)} \\ &+ (F_{yp_i}\cos\delta_i + F_{xp_i}\sin\delta_i)\cos\theta + F_{z_i}\sin\theta\cos\phi \end{aligned}$$

More details on vehicle dynamics can be found in (Gillespie, 1997) and (Rajamani, 2012).

3.4 Feedforward Neural Networks Models

We propose two different artificial neural network architectures to learn the inverse dynamics of a vehicle, in particular the coupled longitudinal and lateral dynamics. An artificial neural network is a network of simple functions called neurons. Each neuron computes an internal state (activation) depending on the input it receives and a set of trainable parameters, and returns an output depending on the input and the activation. Most neural networks are organized into groups of units called layers and arranged in a tree-like structure, where the output of a layer is used as input for the following one. The training of the neural network consists in finding the set of parameters (weights and biases) minimizing the error (or *loss*) between predicted and actual values on a training dataset.

We generated a vehicle dynamics dataset using the 9 DoF vehicle model simulator. The dataset is presented in section 3.4.1. In section 3.4.4 we describe how to train a neural network to learn the inverse dynamics of a vehicle, in particular the coupled longitudinal and lateral dynamics, using the ground-truth vehicle dynamics dataset we generated. The two neural network models we train based on this training procedure are presented in section 3.4.2 for the Multi-Layer Perceptron model and in section 3.4.3 for the Convolutional Neural Network model.

3.4.1 Dataset

Based on the 9DoF vehicle model simulator, we generated a ground-truth vehicle dynamics dataset.

The dataset we generated with the 9DoF vehicle model has a total of 43241 instances: it is divided into a train set of 28539 instances and a test set of 14702 instances. For validation during each respective model's training, a cross-validation approach is chosen, as described in the section 3.4.4 where the training procedure is detailed.

The following procedure was used to generate each instance:

First, a control u to apply is generated randomly, as well as an initial state $\xi^{(0)}$ of the vehicle. More precisely, the vehicle is chosen to be either in an acceleration phase or in a deceleration phase with equiprobability. In the first case, the torques at the front wheels T_{ω_1} and T_{ω_2} are set equal to each other

and drawn from a uniform distribution between 0Nm and 750Nm, while the torques at the rear wheels T_{ω_3} and T_{ω_4} are set equal to zero (the vehicle is assumed to be a front-wheel drive one). In the second case, the torques of each wheel are set equal to each other and drawn from a uniform distribution between -1250Nm and 0Nm. In both cases, the steering angle δ is drawn from a uniform distribution between -0.5 and $+0.5$rad. The initial state $\xi^{(0)}$ is composed of the initial position $(X^{(0)}, Y^{(0)})$ of the vehicle in the ground frame, the longitudinal and lateral velocities $V_x^{(0)}$ and $V_y^{(0)}$, the roll, pitch and yaw angles and their derivatives, and the rotational speed $\omega_i^{(0)}$ of the each wheels. The initial longitudinal speed $V_x^{(0)}$ is drawn from a uniform distribution between 5 and 40m.s^{-1}; the initial lateral speed $V_y^{(0)}$ is drawn from a uniform distribution whose parameters depend of $V_x^{(0)}$; the rotational speed $\omega_i^{(0)}$ is chosen such that the longitudinal slip ratio is zero. All the other initial states are set to zero.

Secondly, the 9 DoF vehicle model is run for 3s, starting from the initial state $\xi^{(0)}$ and keeping the control u constant during the whole simulation. The resulting trajectories are downsampled to 301 time steps, corresponding to a sampling time of 10ms.

Consequently, each instance of the dataset consists in the following triplet:

$\xi^{(0)}$ the initial state of the vehicle,

$u = (T_{\omega_1}, T_{\omega_2}, T_{\omega_3}, T_{\omega_4}, \delta)$ the control -kept constant over time- to apply,

$(X^{(0)}, Y^{(0)}), \ldots, (X^{(300)}, Y^{(300)})$ the resulting trajectory.

The dataset generation method is summarized in Algorithm 1.

Algorithm 1 Dataset Generation

function GENERATE INSTANCE
  `is_accelerating` $\sim \mathscr{B}(0,1)$ ▷ Coin flipping
  if `is_accelerating` $= 1$ **then**
    $u_1 \sim \mathscr{U}(0, 750)$ ▷ uniform; in N.m
    $\delta \sim \mathscr{U}(-0.5, +0.5)$ ▷ uniform; in rad
    $u \leftarrow [u_1, u_1, 0, 0, \delta]$
  else if `is_accelerating` $= 0$ **then**
    $u_1 \sim \mathscr{U}(-1250, 0)$ ▷ uniform; in N.m
    $\delta \sim \mathscr{U}(-0.5, +0.5)$ ▷ uniform; in rad
    $u \leftarrow [u_1, u_1, u_1, u_1, \delta]$
  $V_x^{(0)} \sim \mathscr{U}(5, 40)$ ▷ uniform; in m.s^{-1}
  $V_y^{(0)} \sim \mathscr{U}(a, b)$ ▷ uniform; in m.s^{-1}
  where $a = \max\left(-1, -\frac{V_x^{(0)}}{3}\right)$
  and $b = \min\left(+1, +\frac{V_x^{(0)}}{3}\right)$
  `trajectory` $\leftarrow 9DoF(\xi^{(0)}, u, T_{sim} = 3s)\big|_{(X,Y)}$
  save $(\xi^{(0)}, u,$ `trajectory`$)$
function GENERATE DATASET($n = 43241$)
  for $i \leftarrow 1 \ldots n$ **do** GENERATE INSTANCE()

3.4.2 Model 1: Multi-Layer Perceptron

A Multi-Layer Perceptron (MLP), or multi-layer feedforward neural network, is a neural network f whose equations are:

$$\mathbf{h}^{(0)} = \mathbf{x} \tag{3.8a}$$

$$\mathbf{h}^{(k)} = \sigma^{(k)}(\mathbf{W}^{(k)\top}\mathbf{h}^{(k-1)} + \mathbf{b}^{(k)}), \text{ for } k = 1..L \tag{3.8b}$$

where $\mathbf{x}$ denotes the input vector, $\mathbf{h}^{(k)}$ the output of layer $k \in [\![1, L]\!]$, $L \in \mathbb{N}^*$ the number of layers of the MLP and $\sigma^{(k)}$ denotes the k-th activation function. $\mathbf{h}^{(L)} = f(\mathbf{x})$ denotes the output vector of the neural network.

The MLP, presented in figure 3.2, is used to predict the constant control $(T_{\omega_1}, T_{\omega_2}, T_{\omega_3}, T_{\omega_4}, \delta)$ to apply given an initial state $\xi^{(0)}$ and a desired trajectory $(X^{(0)}, Y^{(0)}), \ldots, (X^{(300)}, Y^{(300)})$. It is trained on the dataset presented in subsection 3.4.1. It comprises $L = 5$ layers, respectively containing 32, 32, 128, 32 and 128 neurons. All the activations functions of the network are rectified linear units (ReLU): $\sigma(x) = \max(0, x)$. The loss function used, as well as weights initialization or regularization are discussed in the section 3.4.4, as they are common for the two neural networks proposed. We performed a grid search to choose the sizes of the layers among $3^5 = 243$ possibilities by allowing each layer to have a size of either 32, 64, or 128 neurons, training the corresponding MLP for 200 epochs and evaluating its performance on the test dataset.

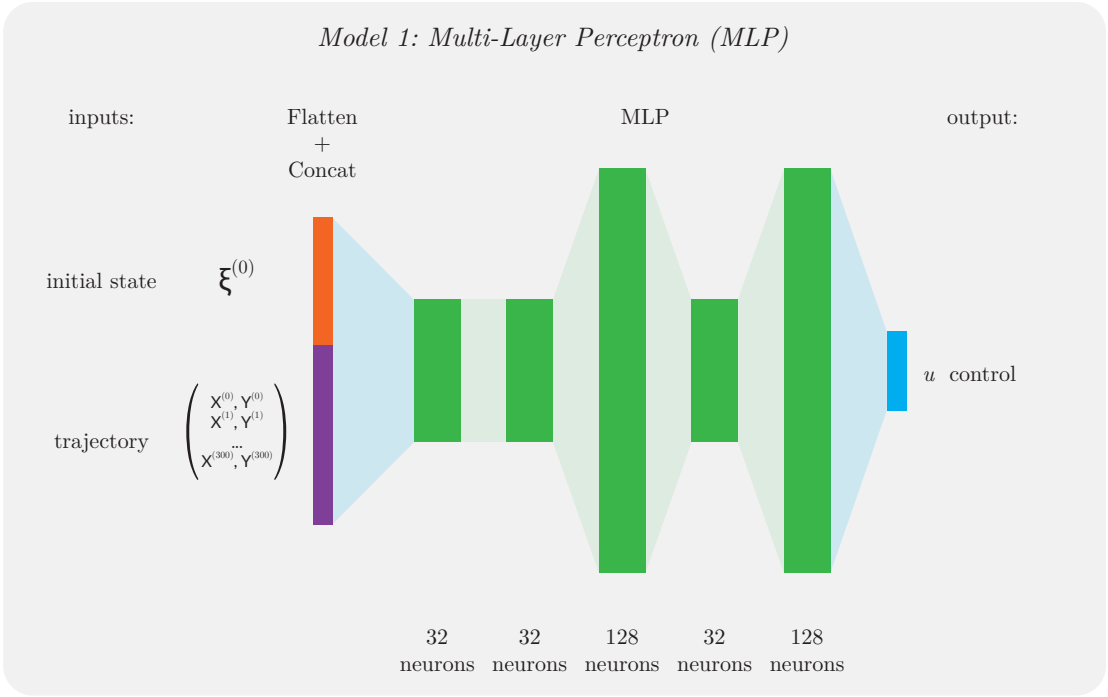


Figure 3.2 – Multi-Layer Perceptron

3.4.3 Model 2: Convolutional Neural Network

Convolutional Neural Networks (CNN) are neural networks that use convolution in place of general matrix multiplication in at least one of their layers. A traditional CNN model almost always involves

a sequence of convolution and pooling layers. CNNs have a proven history of being successful for processing data that has a known grid-like topology. For instance, numerous authors make use of CNNs for classification (Dong et al., 2015), or semantic segmentation (Badrinarayanan et al., 2017) purposes.

We propose to use convolutions to pre-process the vehicle trajectory before feeding it to the MLP, as illustrated in figure 3.3. Trajectories are time-series data, which can be thought of as a 1D grid taking samples at regular time intervals, and thus are very good inputs to process with a CNN. We decided to process the X and Y coordinates separately. For each channel $\mathbf{x}$ (either X or Y), we construct the following CNN module, which is depicted in figure 3.4:

$$\mathbf{h}^{(0)} = \mathbf{x} \tag{3.9a}$$

$$\mathbf{h}^{(k)} = \sigma^{(k)}(\pi^{(k)}(\mathbf{W}^{(k)} * \mathbf{h}^{(k-1)} + \mathbf{b}^{(k)})), \text{ for } k = 1..L' \tag{3.9b}$$

where $\mathbf{h}^{(L')}$ is the output of the CNN module, $L' \in \mathbb{N}^*$ the number of layers, $\sigma^{(k)}$ the k-th activation function and $\pi^{(k)}$ the k-th pooling function.

The parameters of the CNN module are $L' = 3$, with a convolution kernel size of 3 for all convolutions. The activation functions are all ReLU and the pooling functions are all average-pooling of size 2. The first two convolutions have 4 feature maps while the last convolution has only 1 feature map.

As the longitudinal and lateral dynamics are quite different, distinct sets of weights are used for the X and Y convolutions. After processing the X and Y 1D-trajectories by their dedicated CNN module, their output are concatenated. This new output is then fed to the former MLP whose characteristics remain the same except from the dimension of its input. The whole model shown in figure 3.3 is designated as the "CNN model" in the rest of this chapter.

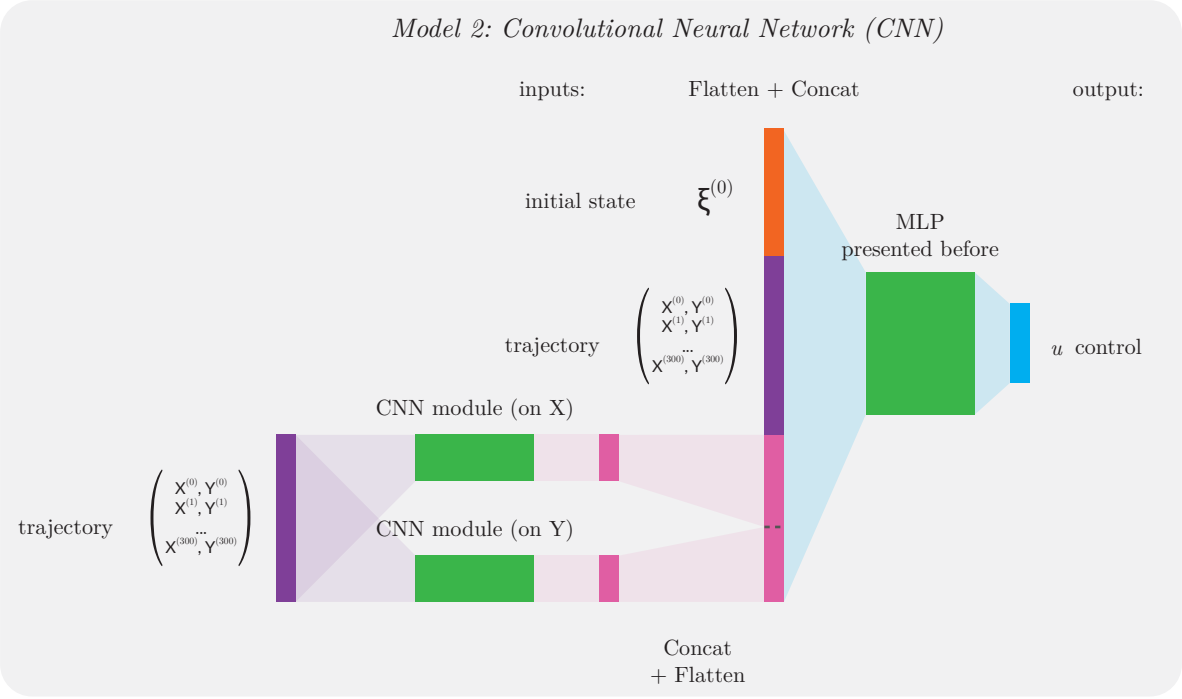


Figure 3.3 – Convolutional Neural Network

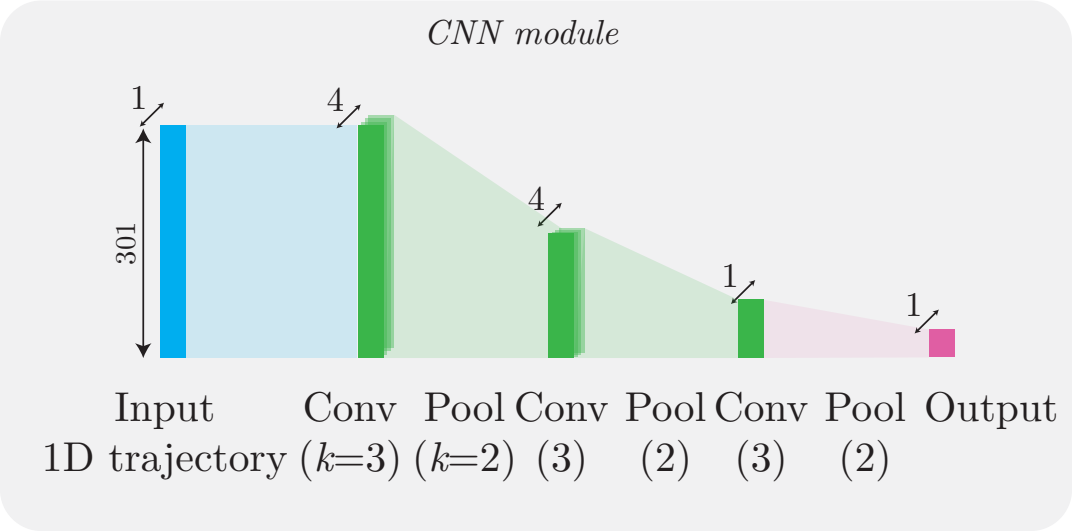


Figure 3.4 – Vehicle Control: CNN Module

3.4.4 Training procedure

The training procedure is the same for the two neural networks:

Weights Initialization & Batching

Each training batch is composed of 32 instances of the dataset. The Xavier initialization (Glorot et al., 2010) (also known as GLOROT uniform initialization) is used to set the initial random weights for all the weights of our model.

Train, Test and Validation sets

During training, a 5-fold cross-validation approach is chosen: for each one of the 5 folds of the train set, the model is trained on the remaining 4 folds of the train set, and validated on the remaining part (1 fold) of the train data. The final score measure reported by the 5-fold cross-validation is the average of the values computed in the loop. Once the model's parameters have been adjusted on the 5 different folds of the train set, a final evaluation on the test set is performed.

Loss function, Regularization & Optimizer

The objective of the training is to reduce the mean square error (MSE) between the controls predicted u^{pred} by the neural network and the ones u^{real} that were really applied to obtain the given trajectory. The neural network is trained in order to minimize the loss function L defined by Equation (3.10) on the train dataset, before evaluation on the test dataset.

$$L = \gamma L_{\delta} + (1-\gamma) L_T + L_{reg} \tag{3.10}$$

where

$$L_\delta(\delta^{real}, \delta^{pred}) = \frac{1}{0.5} MSE(\delta^{real}, \delta^{pred}) \tag{3.11a}$$

$$L_T(T_{\omega_i}^{real}, T_{\omega_i}^{pred}) = \frac{1}{4 \times 2000} \sum_{i=1}^{4} MSE(T_{\omega_i}^{real}, T_{\omega_i}^{pred}) \tag{3.11b}$$

$$L_{reg}(W) = \gamma_{reg} ||W||_2^2 \tag{3.11c}$$

The scaling factors $1/0.5$ and $1/(4 \times 2000)$ were chosen in order to normalize the steering and the torques. The parameter $\gamma = 0.99$ was chosen in order to prioritize the lateral dynamics over the longitudinal one. Equation (3.11c) is an L2 regularization of our model, where W is the vector containing all the weights of the network. We set $\gamma_{reg} = 10^{-5}$.

To train our model, we used the Adam optimization algorithm (D. Kingma et al., 2014). It calculates an exponential moving average of the gradient and the squared gradient. For the decay rates of the moving averages, we used the parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$. The values of other parameters were $\alpha = 10^{-3}$ for the learning rate, and $\varepsilon = 10^{-8}$ to avoid singular values.

3.5 Comparison between Classical approaches and Neural Networks approaches

In order to compare their ability to learn the vehicle dynamics, the two different artificial neural networks are used as “controllers"[3]. The reference track, presented in figure 3.5, comprises both long straight lines and narrow curves. The reference speed is set to $V_{ref} = 10$m/s on the whole track.

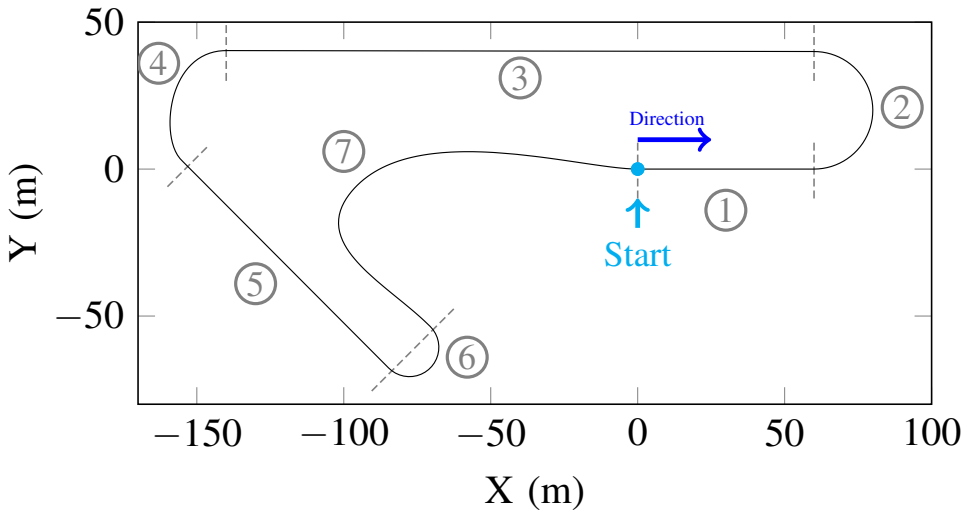


Figure 3.5 – Top view of the test track; numbers 1 to 7 refers to different road sections delimited by dashed lines in order to facilitate the matching with figures 3.9 to 3.13.

[3]Properly speaking, they are not real controllers as they do not learn how to reject disturbances and modeling errors.

3.5.1 Generating the control commands

In order to compute the control commands to be applied to the vehicle, the artificial neural network needs to know the trajectory the vehicle has to follow in the next 3s, as in the train dataset. One problem that arises is that it has only learned to follow trajectories starting from its actual position such as in figure 3.6. However, in practice, the vehicle is almost never exactly on the reference path. Therefore, a path section starting from the actual position of the vehicle and bringing it back to the reference path is generated: for that purpose, cubic Bezier curves were chosen as illustrated in figure 3.7. Thus, at each iteration, (i) a Bezier curve with length 3s is computed to link the actual position of the vehicle to the reference trajectory; (ii) a query comprising the previously computed Bezier curve is sent to the artificial neural network; (iii) the artificial neural network returns the torques at each wheel and the front steering angle to apply until the next control commands are obtained. The computation sequence is run every 300ms, even though the query takes less than 2ms.

3.5.2 Comparison of the models

The results obtained for the MLP and the CNN models are displayed respectively in blue and in red in figures 3.9 to 3.13. The resulting videos, obtained using the software PreScan (*TASS International* n.d.), are available online[4]. Clearly, it appears that the results obtained using a CNN are better than a MLP. First, we observe that the control commands are smoother in curves with the CNN. There are steep steering (see figure 3.9) and front torques (see figure 3.10) variations for the MLP around $s = 360$m in road sections n°4 and around $s = 480$m in road sections n°6. In the latter case, the steering angle reaches its saturation value $+0.5$rad and the wheel torques change suddenly from 1000Nm to -1000Nm and vice-versa, which is impossible in practice. On the contrary, the control signals of the CNN model remains always smooth and within a reasonable range of values. Secondly, both the longitudinal and lateral errors are smaller for the CNN than the MLP as shown respectively in Table 3.2 and 3.3.

Table 3.2 – Comparison of the longitudinal performances of the MLP and CNN controllers (in m/s).

model	RMS	average	std. dev.	max
MLP	0.76	-0.29	0.70	-4.94
CNN	0.60	-0.39	0.46	-2.33

Table 3.3 – Comparison of the lateral performances of the MLP and CNN controllers (in m).

model	RMS	average	std. dev.	max
MLP	0.61	0.003	0.61	3.26
CNN	0.43	0.014	0.43	1.7

However, unlike classic controllers, stability cannot be ensured for these “controllers" as they are black boxes. In particular, for the CNN, we observe a lateral static error in straight lines. This static error is caused in fact by the Bezier curves which do not converge fast enough to the reference track on

[4]https://www.youtube.com/watch?v=yyWy1uavlXs

straight lines as only the first 300ms are really followed by the CNN model (see figure 3.8). Moreover, figure 3.9 shows that the steering angle applied during straight lines is the same for MLP and CNN.

3.5.3 Coupling between longitudinal and lateral dynamics

The speed limit a kinematic bicycle model can reach in a curve of radius R is given by Equation (3.12) where $\mu = 1$ is the road friction coefficient and g the gravity constant (Polack et al., 2018). This corresponds to 9.9m/s ($R = 20$m) in road section n°2 and 7.0m/s ($R = 10$m) in road section n°6. As the reference speed is set to 10m/s throughout the track, conventional decoupled longitudinal and lateral controllers (based on a kinematic bicycle model) will not perform well in road section n°6.

$$V_{kbm_{lim}} = \sqrt{0.5\mu g R} \tag{3.12}$$

On the contrary, both models (especially the CNN) are able to pass this road section, showing the ability of artificial neural networks to handle coupled longitudinal and lateral dynamics. More precisely, we observe in figure 3.12 that the speed is reduced in section n°6 because the artificial neural networks deliberately brake (see figure 3.10 and 3.11), even though the speed of the vehicle is below the reference speed. This is due to the loss function used during training and given by Equation (3.10) that penalizes more steering angle errors than torque errors. Hence, the models prioritize the lateral over the longitudinal dynamics.

Therefore, such “controllers" are particularly interesting for highly dynamic maneuvers such as emergency situations or aggressive driving where the longitudinal and lateral dynamics are strongly coupled. However, they should be used sparingly as they are only black boxes, or should at least be supervised by model-based systems. Moreover, for normal driving situations, conventional decoupled longitudinal and lateral controller should be preferred.

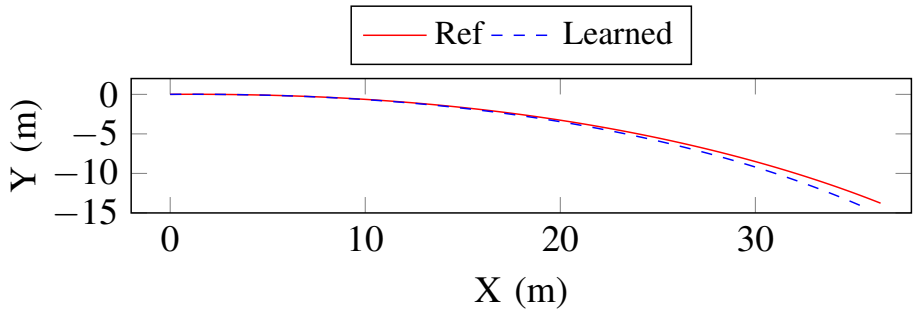


Figure 3.6 – Example of a training dataset instance: in red, the reference trajectory, in blue the one obtained from the control predicted by the CNN model.

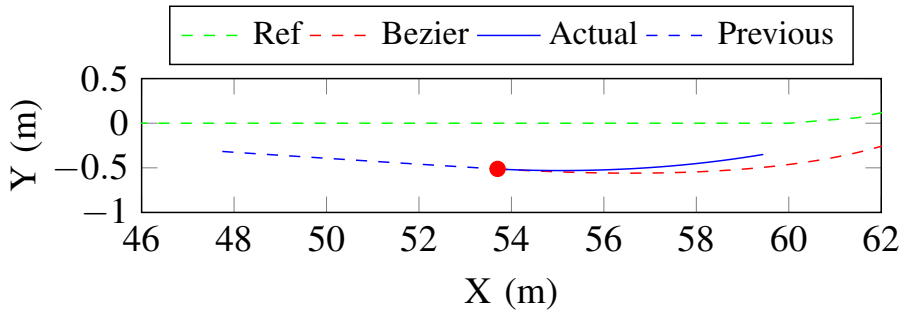


Figure 3.7 – Example of a Bezier curve (in red) joining the actual position of the vehicle (the red circle) to the reference trajectory (in green). The actual trajectory followed by the vehicle is shown in blue.

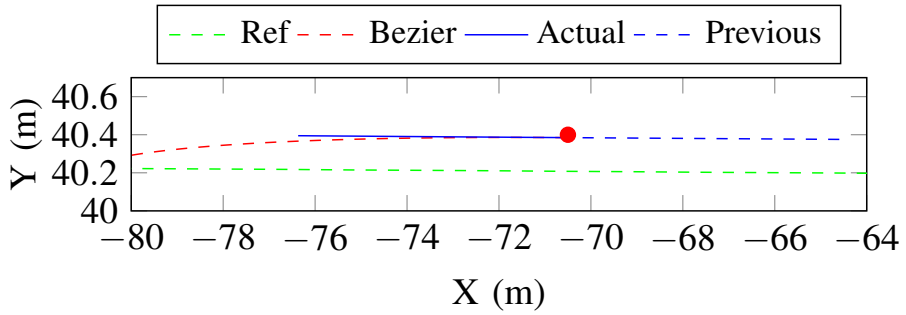


Figure 3.8 – Example of a Bezier curve on a straight line section of the reference trajectory. The lateral error is not corrected since the convergence of the Bezier curve to the reference trajectory is too slow.

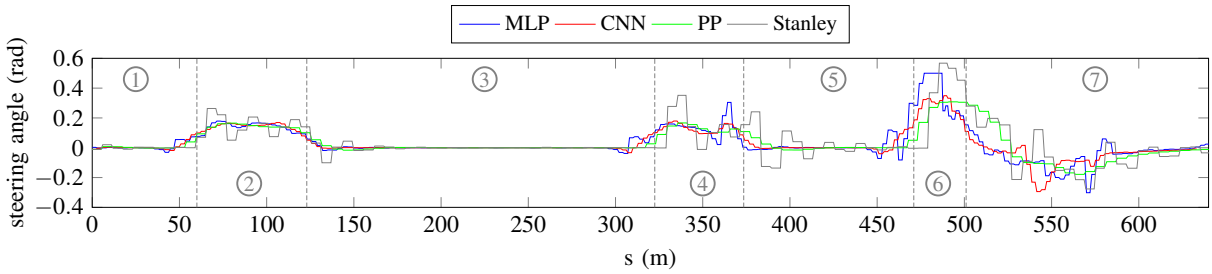


Figure 3.9 – Comparison of the steering command computed by the different controllers. The numbers 1 to 7 correspond to the different road sections presented in Figure 3.5.

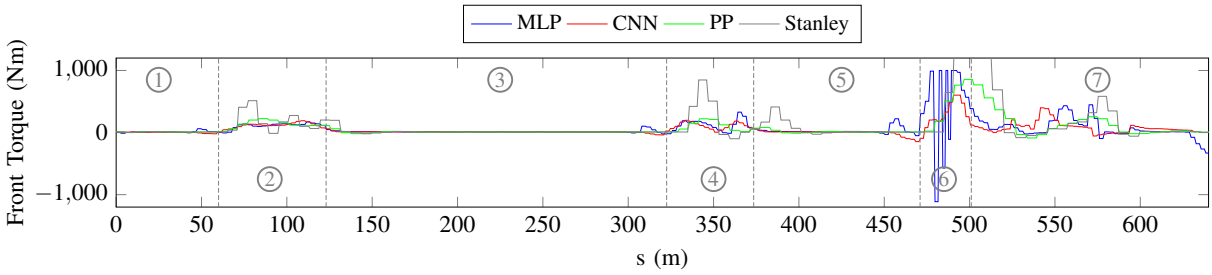


Figure 3.10 – Comparison of the torque applied at the front wheels computed by the different controllers.

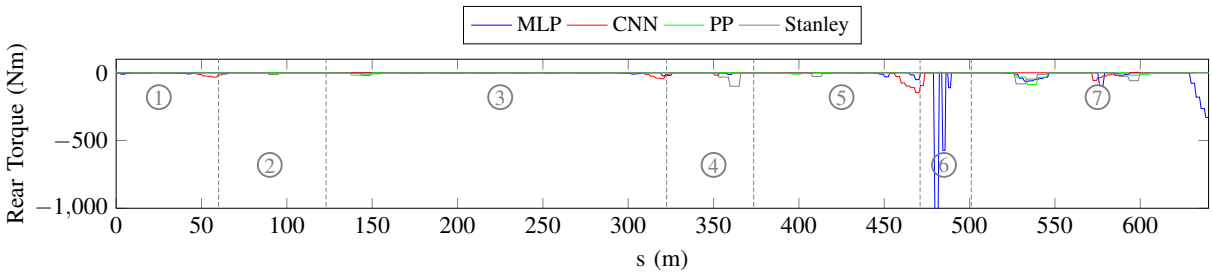


Figure 3.11 – Comparison of the torque applied at the rear wheels computed by the different controllers.

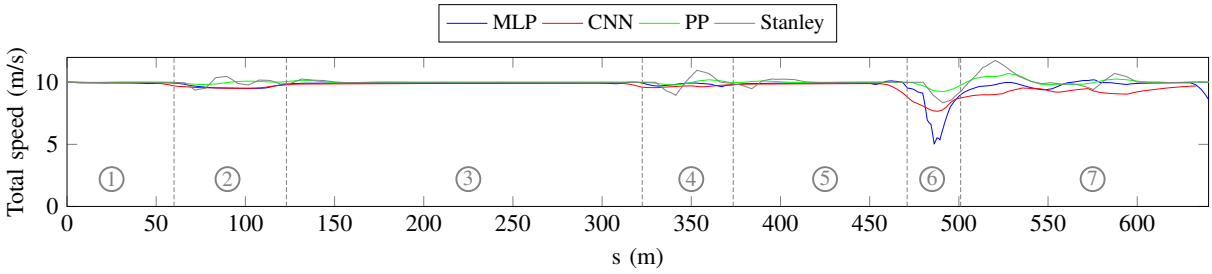


Figure 3.12 – Comparison of the total speed obtained with the different controllers.

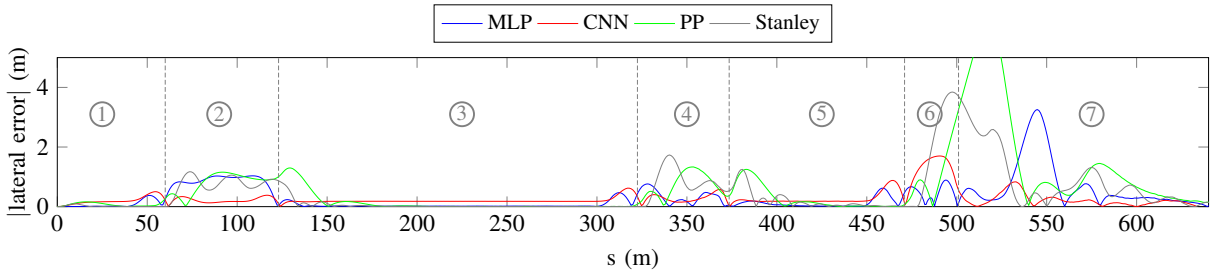


Figure 3.13 – Comparison of the absolute value of the lateral error obtained with the different controllers.

3.5.4 Comparison with decoupled controllers

Finally, the “controllers" obtained with the MLP and CNN models are compared with commonly used decoupled controllers: the lateral controller is either a pure-pursuit (PP) (Coulter, 1992) or a Stanley (Thrun et al., 2006) controller while in both cases, the longitudinal controller is ensured by a Proportional-Integral (PI) controller with gains $K_P = 600$ and $K_I = 10$. The gain for the front lateral error is 0.75 for the Stanley controller. The preview distance of the pure-pursuit controller is defined as a function of the total speed V_g at the center of gravity: $L_P = l_f + T_A V_g$ where $T_A = 1.5$s is the anticipation time. The results of the PP and the Stanley controllers are shown respectively in green and grey in figures 3.9 to 3.13. Clearly, a decrease of performance can be observed when using these decoupled controllers in the challenging part of the track. In particular, the lateral error becomes huge in both cases during the sharp turn of road section n°6 while the CNN was able to perform reasonnably well.

3.6 Conclusion

This work presented some preliminary results on deep learning applied to trajectory tracking for autonomous vehicles. Two different approaches, namely a MLP and a CNN, were trained on a high-fidelity vehicle dynamics model in order to compute simultaneously the torque to apply on each wheel and the front steering angle from a given reference trajectory. It turns out that the CNN model provides better results, both in terms of accuracy and smoothness of the control commands. Moreover, compared to most of the existing controllers, it is able to handle situations with strongly coupled longitudinal and lateral dynamics in a very short time. However, the controller obtained is a black-box and might not be used standalone.

The results proved the ability of deep learning algorithms to learn the vehicle dynamics characteristics. From a deep learning perspective, it is worth noting that (inverse) vehicle dynamics -who are highly dependent on temporal evolutions- can be learned by neural networks that make, in a sense, a relatively limited use of the temporal information by itself.

Being able to learn vehicle dynamics with neural networks opens a wide range of new possible applications of such techniques, for example for generating dynamically feasible trajectories. Future work will focus on (i) replacing the complex dynamics models by a learned off-line model in Model Predictive Control for motion planning, (ii) using Generative Adversarial Networks (GAN) to generate safe trajectories where the learned dynamics is used as constraint, and (iii) performing real-world experiments with our approach on a real car.

Chapter 4

Gesture Recognition with Convolutional Neural Networks over Time

"On ne peut oublier le temps qu'en s'en servant."

Baudelaire

Contents

4.1 Introduction

Gesture is one of the most natural and simplest way to interact with one's environment, including other humans and machines. It is complementary to voice and does not require a complex physical brain-machine interface. The ability to recognize human intents and actions is useful for numerous real-life situations, and even critical for the design of meaningful interactions between humans and machines.

While pose estimation is often performed at a frame-level only[1], gesture recognition naturally takes into account temporal information from the motion performed. Different sensors can be used to capture gestures, including mono cameras, stereo cameras, depth-aware cameras, as well as event cameras, on-body IMUs and motion capture sensors placed on suits or gloves.

Several approaches exist for gesture recognition, using either dense or sparse sensor data as input. One can make use of dense 2D image data obtained by the previously cited sensors, and extract features from the data to classify the gesture performed. One can also work on dense 3D data, which require more computational resources but have more potential, and can also be acquired with these sensors. Such dense 3D data can represent tremendous amount of data, however it has been known for a long time that skeletal pose, which is a sparse representation of the human body based on its joints, is sufficient to describe and understand human motion (Johansson, 1973). Joints positions, rotations and other relevant vector features, can be either estimated from 2D or 3D data using vision techniques, or even be directly provided by the sensor, at a very high frame rate (Lugaresi et al., 2019; Raaj et al., 2019). One can also note that the use of pose sequence reduces privacy concerns, compared to image sequences, because of the nature of the information involved in these two representations. All these reasons support and motivate the interest of performing gesture recognition using only 3D or 2D pose data sequences.

For illustration purposes, a handstand movement viewed both as an RGB image sequence and as a pose sequence is proposed in figure 4.1. As one can see in figure 4.1, human pose sequences preserve more privacy than image sequences but they still allow recognition of the action performed.

For widespread adoption, gesture recognition algorithms need to be reliable and fast enough to be computed in real-time on embedded devices like smartphones and robots. This requirement of fast and reliable computing becomes crucial in many practical use cases, e.g. for human-robot collaboration in factories.

We propose a Convolutional Neural Network (CNN) approach to perform gesture recognition by computing 1D convolutions over the time dimension to capture temporal information. Gesture can either be analyzed after each new frame or at the end of the gesture.The most important contributions of this model are a source-agnostic approach, able to work with different types of sensors, and an architecture light enough to be run on embedded devices. The model architecture only uses convolutions, which are easier to train, more constrained and easier to audit, compared to recurrent cells. Moreover the model architecture can be applied without adaptation work to various type of gestures.

The rest of this chapter is organized as follows: in section 4.1.1 we summarize our contributions for

[1] In the literature, pose estimation is sometimes improperly designated as "static" gesture recognition, whereas gesture recognition itself is coined as "dynamic" gesture recognition in order to highlight the importance of the temporal dynamics involved in gesture recognition.

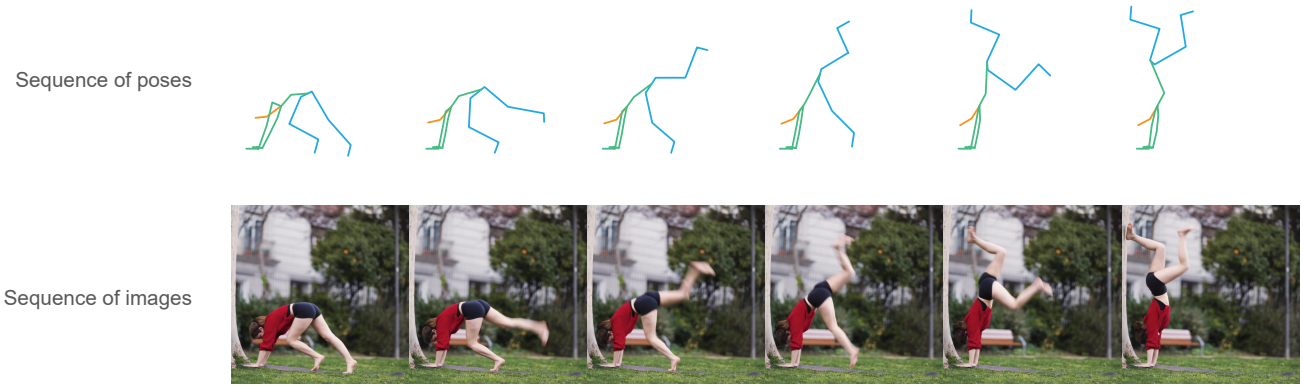


Figure 4.1 – Human pose sequences can be used as a sparse representation of movements (here, a handstand movement)

this chapter. In section 4.2 we propose a review of related works in this research area. In section 4.3 we introduce the proposed approach, including the network architecture and the evaluation protocol we use. In section 4.4 we study the influence of individual components of the model, all other things being equal. Finally, in section 4.6 we summarize our approach and highlight the main outcomes of the experiments we conduct.

4.1.1 Contributions summary

Besides the overall approach and (hyper-)parameters, our contributions are described in this section. In section 4.3.2, we introduce a deep convolutional neural network model (SkelNet model) architecture for gesture recognition based on temporal information. In section 4.4.2, we introduce a simple early recognition method for the SkelNet model. In section 4.4.4, we study the influence of varying hyperparameters for the different building blocks of the model, in order to justify the design choices behind the neural network architecture proposed. In section 4.4.5, we study whether -and how- sharing some of the weights of the model could help reduce the model's parameters' count without degrading too much the model recognition performances. In section 4.4.6, we show that the model is not specific to hand gesture recognition: the model can be used for human action recognition and for facial emotion recognition, for instance. In section 4.5, in order to -ultimately- better understand the model, we perform visualization experiments and we propose an attempt to interpret them. Finally, all the other experimental results presented in this chapter are also ours.[2]

4.2 Related Works

In this section, we review published state-of-the-art approaches for skeleton-based human activity and hand gesture recognition. These approaches begin with the extraction of spatial or temporal features from raw data. The extracted features are later provided to a machine learning algorithm which performs the classification. The features can either be handcrafted by human experts or learned directly from the data, using a deep learning approach. Deep learning approaches for skeleton gesture recognition can be

[2]When the experiments we perform make use of already existing methodologies -e.g. for Procrustes in section 4.4.2 or for TCN's causal-convolutional layers in section 4.4.4-, references and motivations are also provided.

split into three main categories: the ones that make use of recurrent cells, the ones that make use of convolutional cells, and the ones that make use of an attention mechanism. However, it is possible to combine them together, e.g. using convolutional cells followed by recurrent cells or using attention over recurrent cells.

In the following subsections, we review the handcrafted approaches and the deep learning approaches.

4.2.1 Handcrafted features

Rather than directly providing the raw hand skeletal data to a classifier, one can pre-compute relevant representations from the raw data to feed the classifier. These hand-crafted representations are entirely designed by human experts using their knowledge, e.g. regarding usual gesture speed, or physiological constraints. Hand-crafted representations usually describe geometric properties, physical constraints and statistical features about skeletons: e.g. distance between joints, orientations of the joints, curvature of the joints' trajectories. Hand-crafted representations can also involve any other human interpretable metric computed from the skeletal data.

Examples of hand-crafted representations range from histograms of 3D joints locations (HOG3D) (Klaser et al., 2008), to 3DSURF (Knopp et al., 2010), or to Laban descriptors (Truong et al., 2016), to name a few. Enumerating all the numerous published hand-crafted representations approaches is out of the scope of this chapter.

(Han et al., 2017) provide a comprehensive review of hand-crafted spatio-temporal features relevant for 3D skeletal data. (L. Wang et al., 2019) compare the effectiveness of ten of these hand-crafted features. Besides multi-modal representations, (Han et al., 2017) identify that published handcrafted representations usually fall into one of the following three main categories: displacement-based representations, orientation-based representations, and representations based on raw joint positions. The overwhelming majority of the approaches introduced below have been introduced in the context of full-body skeletal gesture recognition but can still provide insights -and be applied- to hand-skeletal gesture recognition.

Full-body skeleton

(Xia et al., 2012) propose to project HOG3D features using Linear Discriminant Analysis (LDA) and cluster them into posture visual words which represent the prototypical poses of actions. The visual words temporal evolutions are modeled by discrete hidden Markov models (HMMs). In a similar fashion, (Jin et al., 2012) propose to use quantized 3D joint angles and the (position) displacement of a central joint to describe atomic movements. (Hussein et al., 2013) propose a covariance of 3D joints (Cov3DJ) descriptor to captures the dependence of locations of different joints on one another. (Ofli et al., 2014) propose a sequence of the most informative joints (SMIJ) descriptor where they select the most informative joints at each time step with regards to interpretable metrics such as the mean or variance of joint angle trajectories. (Jiang Wang et al., 2013) propose to use the pairwise distances of joints positions as well as Fourier Temporal Pyramids (FTP). (Ohn-Bar et al., 2013) propose to use pairwise affinities of joints angles as well as descriptors extracted by an extension of the histogram

of oriented gradients (HOG) algorithm. (Zanfir et al., 2013) propose a moving pose descriptor that considers both the position of each joint as well as its speed and acceleration. The computed descriptors are classified by a modified non-parametric k-nearest neighbor (k-NN) classifier. (Chaudhry et al., 2013) encode skeletal sequences into spatiotemporal hierarchical models, and then use linear dynamical systems (LDS) to learn the dynamic structures. (Slama et al., 2015) propose to use the geometric structure of the Grassmann manifold. (Vemulapalli et al., 2014) propose a human skeletal representation within the Lie group $SE(3) \times \ldots \times SE(3)$, based on the idea that rigid body rotations and translations in 3D space are members of the Special Euclidean group $SE(3)$. Human actions are then viewed as curves in this manifold. The final classification is performed in the corresponding Lie algebra using the dynamic time warping (DTW) discrepancy, Fourier temporal pyramids (FTP) to model the temporal dynamics, and a linear support-vector machine (SVM) for the classification. (Anirudh et al., 2015) use the same representation with manifold functional principal component analysis (mfPCA) to reduce the dimensionality of the features. (Devanne et al., 2014) represent skeletal joints' sequences as trajectories in a n-dimensional space. The trajectories of the joints are interpreted in a Riemannian manifold. Similarities between the shape of trajectories in this shape space are calculated with k-nearest neighbor (k-NN) in order to achieve the sequence classification. (Amor et al., 2015) views the evolution of skeleton shapes over time as trajectories on Kendall's manifold and introduces a suite of tools derived from Riemannian geometry to study them. (Ben Tanfous et al., 2018) proposes to code a sparse skeletal shape represented as a point from Kendall's manifold on its attached tangent space. (Chrungoo et al., 2014) propose a scale-invariant and speed-invariant descriptor called histogram of direction vectors (HODV). (Evangelidis et al., 2014) propose a local skeletal representation which implies a view-invariant descriptor of joint quadruples, as well as a multi-level representation of Fisher vectors (FV) obtained with Fisher kernel representations encoding the generation of such local skeletal representations from a gaussian mixture model (GMM). (H. Zhang et al., 2015) propose a bio-inspired approach. They project 3D human skeleton trajectories onto three anatomical planes: coronal, transverse and sagittal planes. (Tao et al., 2015) propose a framework for jointly learning the feature representations and action classifiers, using classifiers called moving poselet (MP) that describe bodypart configurations undergoing certain movements. (Coppola et al., 2015) introduce qualitative descriptions of joints trajectories using 3D qualitative trajectory calculus and use hidden Markov models (HMM) to perform the final classification. (Ding et al., 2015) use a graph to avoid periodic sequences and then use a spatiotemporal feature chain (STFC) to represent the human actions by trajectories of joint positions. (G. Zhu et al., 2016) segment normalized relative orient (NRO) features to train an offline model that predicts key poses and atomic motions, which can then also be used for an online action recognition approach based on a variable-length maximum entropy markov model (MEMM). (Cippitelli et al., 2016) use the k-means clustering algorithm on normalized distances between joints to select the most important postures in a sequence and then classify the sequence using a multiclass support-vector machine (SVM) classifier with a radial basis function (RBF) kernel. (Chunyu Wang et al., 2016) propose to use key-pose-motifs for each action class, a key-pose-motif containing a set of ordered poses required to be close but not necessarily adjacent in the action sequences. Sequence classification is performed by matching it to the motifs of each class and selecting the class that maximizes the matching score. (Lillo et al., 2016) use a hierarchical model with motion poselets dictionary entries and

histograms of these motion poselets. The human body representation is split into a set of spatial regions. (Pei Wang et al., 2016) smooth joints trajectories using B-splines to generate motionlets. Undirected complete labeled graphs combining these motionlets and their spatio-temporal correlations are created to represent the gestures. Finally, a graph kernel called subgraph-pattern graph kernel (SPGK) is proposed to measure the similarity between the graphs.

Hand skeleton

(Ionescu et al., 2005) use local orientation histograms and the superposition of all hand region skeletons within a sequence. The Baddeley's distance between the resulting signature and signatures from a gesture alphabet is finally computed. (Reddy et al., 2011) use distance between joints and the superposition of hand skeletons also called dynamic signature. (Wang et al., 2014) use a superpixel earth mover's distance to measure the dissimilarity between hand gestures, where the superpixels come from hand shapes (depth) and the corresponding textures (color) provided by a Kinect camera. (De Smedt et al., 2017) use three hand-crafted descriptors: shapes of connected joints (SoCJ), histograms of hand directions (HoHD) and histograms of wrist rotations (HoWR), as well as Fisher Vectors (FV) for the final representation.

4.2.2 Deep-learning

To the best of our knowledge, most of the existing work in the literature of deep neural networks based methods for hand or body gesture recognition focus on video-based approaches. (H.-B. Zhang et al., 2019) provide a review of vision-based approaches. However there is significantly less published deep learning methods that deal with skeleton data as input. In this subsection we survey a list of these deep-learning based gesture recognition methods on skeletal data.

Fully-Connected Layers (FC)

Unconstrained vanilla fully-connected layers alone are not efficient to model temporal phenomena. (C. Li et al., 2019) propose to use path signature (PS) features both for spatial and temporal features. They also propose a differentiable temporal transformer module (TTM) that can be included in any neural network and use fully connected layers to perform sequence classification. The TTM module consists in a localization step and a temporal shift step. The TTM module transforms input gestures temporally in order to alleviate the temporal difference that inevitably arises between sequences.

Recurrent neural networks (RNN)

For the last few years, recurrent neural networks have nearly been the *de facto* state-of-the-art approaches for sequence modeling, which explains why most of the deep-learning approaches for gesture recognition use recurrent cells like the long short-term memory (LSTM) cell introduced by (Hochreiter et al., 1997a) or the gated recurrent unit (GRU) cell introduced by (Cho et al., 2014).

(Avola et al., 2018) use hand-crafted joint angles features learned by a recurrent LSTM architecture. (Lefebvre et al., 2013) propose a bi-directional LSTM to learn inertial 3D gestures from micro-electro-

mechanical (MEM) systems. (H. Wang et al., 2017) propose a two-stream spatio-temporal recurrent neural network model with LSTMs, where the joint coordinates are given as input of the two RNN branches. (Du et al., 2015) propose a hierarchical recurrent neural network model to represent the human body spatial structure and temporal dynamics of the joints. (W. Zhu et al., 2016) propose a recurrent neural network model with LSTM cells and fully connected layers for the classification. They introduce regularization both in the fully connected layers with a mixed-norm reguarization term in the loss function to drive the model to learn co-occurrence features of the joints at different layers and in the LSTM cells, where they derive an internal dropout similar to the DropConnect regularization method introduced in (Wan et al., 2013) or to one of the the regularization methods introduced in (Merity et al., 2017) for language models in the natural language processing (NLP) domain. (Veeriah et al., 2015) introduce a differential gating scheme for the LSTM neural network, which emphasizes on the change in information gain caused by salient motions between the successive frames, in a model called differential Recurrent Neural Network (dRNN). (J. Liu et al., 2016) also introduce a gating mechanism within LSTM cells to improve recognition robustness, and use a tree-structure based traversal method for better representation of human skeletons. (Shahroudy et al., 2016) propose a part-aware LSTM where the network learns the long-term context representations individually for each part of the skeleton.

Convolutional neural networks (CNN)

Recurrent cells are relatively slow and difficult to train or to use in a hardware-accelerated parallel computing setting, compared to convolution-only methods. Convolutions can be computed on 2D images with 2D convolutions, in 3D video images with 3D convolutions, but also on trajectories of joints positions or orientations with 1D convolutions. Deep learning tools for graphs like graph convolutions can also be computed by taking the skeleton graph as input of the graph convolution. (Wu et al., 2019) propose a survey on graph neural networks in which they describe spectral-based and and spatial-based possible definitions of graph convolution and graph pooling, as well as graph neural networks architectures families, including temporal recurrent and spatiotemporal ones.

(Neverova, 2016a; Neverova et al., 2015) propose an adaptive multi-modal convolutional neural network gesture recognition approach featuring a "ModDrop" training technique. ModDrop performs a gradual fusion between modalities and randomly drops separate channels during training, in order to improve the model robustness and prevent false co-adaptations between data representations.

(F. Yang et al., 2019) propose a lightweight convolution-only model which takes skeletal joints positions and the pairwise distances between the joints as input. (S. Yan et al., 2018) propose a spatial-temporal graph convolutional network (ST-GCN) model where each skeleton is viewed as a graph over which can be processed by graph convolutions. They extend spatial graph convolutions into spatio-temporal convolutions by including temporally connected joints in the neighborhood used by the convolution. (X. S. Nguyen et al., 2019; X. Nguyen et al., 2019) also model hand skeletons as graphs for hand gesture recognition. Their classification pipeline involves graph convolutions and learned symmetric positive definite (SPD) matrices which are known to lie on a Riemannian manifold, namely the Stiefel manifold. (J. Weng et al., 2018) propose a deformable pose traversal convolution network, using deformable convolutions that use a learned receptive field.

Convolutional and Recurrent layers

Recurrent neural networks and convolutional neural networks can also be combined. Most common case consists in features being first extracted with convolutional layers, before their temporal dynamic is modeled by recurrent layers.

(Neverova, 2016b) explore the use of 1D convolutions over the time dimension, with both convolution-only and convolution-and-recurrent model architectures on the Google Abacus biometric user authentication dataset. They obtain the best results using a "Conv-DCWRNN" architecture that mixes convolutions with clockwork RNNs.

(Baccouche et al., 2011) propose a to use 3D convolutions on video images sequences to extract spatio-temporal features that are fed to a RNN trained to classify the gesture sequences. (Donahue et al., 2015) propose to perform 2D convolutions on video images to get features that are modeled by two layers of LSTM cells. (I. Lee et al., 2017) propose an ensemble of temporal sliding LSTM networks to capture short-, medium-, and long-term temporal dependencies. (Nunez et al., 2018) propose to perform 1D convolutions on the joints followed by RNNs. (J. Weng et al., 2018) also propose a deformable pose traversal convolution method based on 1D convolutions and LSTMs.

Attention mechanisms

Attention mechanisms have attracted a vivid interest of the deep-learning research community in the last years. In biological human settings, attention serve multiple purpose: it can both enhance the perception of important stimuli and prioritize such stimuli for decision-making. In artificial neural networks, attention is commonly used for alignment and masking purposes. In its most general form, attention can be described as a way to quantify the interdependence between two vectors or more. For instance, one can learn attention masks to align two sequences, or to emphasize certain joints or time steps over others. Attention is usually not a deep learning architecture *per se* but rather a mechanism used to provide pertinent information to other building blocks of a neural network.

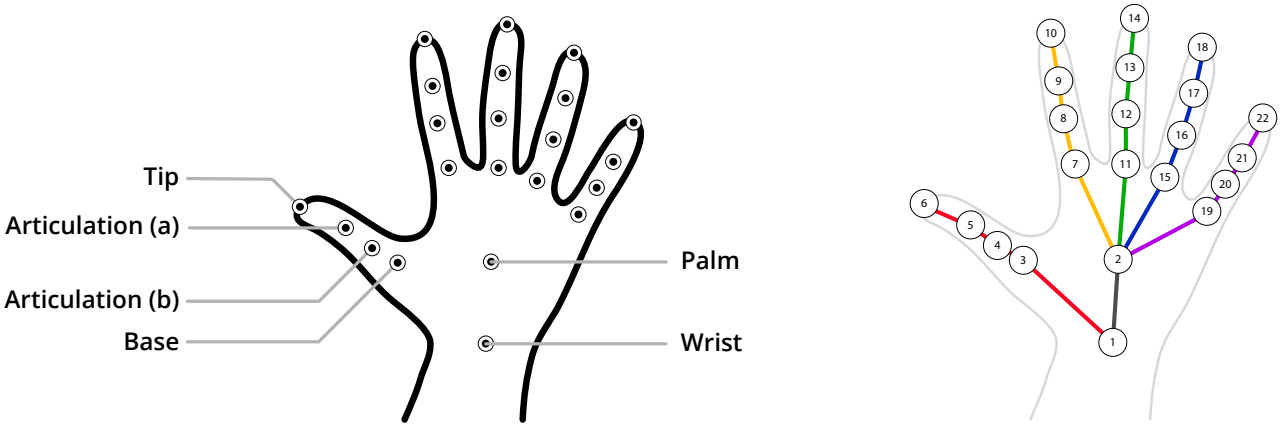
(Baradel et al., 2017a,b, 2018) propose a spatio-temporal soft-attention mechanism on RGB frames that is conditioned on pose features. The pose features are obtained with a convolutional model that takes a 3D subsequence pose data tensor as input and processes it. (Hou et al., 2018) propose a convolutional neural network with attention called spatial-temporal attention residual temporal convolutional network (STA-Res-TCN). Their network processes hand skeletons with a main convolutional branch which uses temporal convolutions (TCN) from (Lea et al., 2017). Another parallel mask branch for attention is also present, the final generation of attention-aware features being obtained with element-wise multiplications. (Maghoumi et al., 2018) propose a model based on a stack of 5 recurrent layers of gated recurrent unit (GRU) cells, over which an attention mechanism is applied before being fed to another GRU layer and two fully-connected layers. (Song et al., 2017) combine the use of an LSTM-based neural network for human action recognition from skeletal data with a spatio-temporal attention mechanism. Joint-selection and frame-selection gates are used to adaptively allocate different attentions. (J. Liu et al., 2017) propose a global context-aware attention LSTM (GCA-LSTM) model in a similar fashion to (Song et al., 2017), but with an attention mechanism which explicitly uses the full spatial and temporal context, rather than -somehow more- implicitly on each individual the LSTM cells hidden

states. (Fan et al., 2018) propose an attention mechanism for multiview fusion of skeletons preprocessed by LSTM cells.

4.3 Convolutional Neural Networks over Time approach

In this section we present our approach to gesture recognition from human pose sequences.

4.3.1 Pose Data Representation



(a) Hand is represented by 22 joints that reflect and summarize the anatomy of a human hand: 4 joints are used for each finger, 1 for the palm (roughly located at the center of the metacarpal bones) and 1 for the wrist. Each dot represents one of the 22 joints of the hand skeleton.

(b) Hand joints are ordered and indexed as illustrated above. The graph representing the hand can be viewed as a tree, whose root is the joint #1 (wrist).

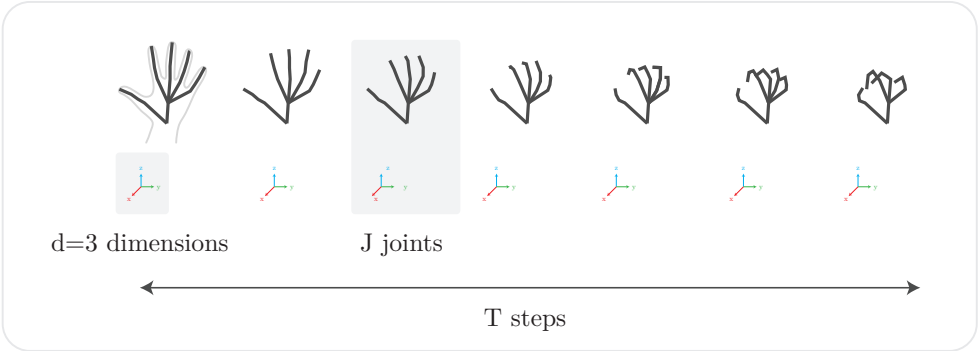
Figure 4.2 – Hand skeleton returned by the Intel RealSense camera.

Human body can be represented with a skeletal representation. Skeletal representations are sparse, robust to illumination changes and scene variations.

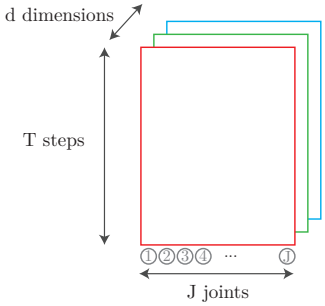
Phenomenologically, visual skeletal representations are also known to be sufficient for human people to describe and understand biological motion, including human motion (Johansson, 1973).

Skeletons can be directly provided by sensors, e.g. by highly accurate consumer-grade depth sensors, or either be estimated from 2D or 3D data using vision techniques, at a very high frame rate, even in real-time and in on-device mobile settings (Lugaresi et al., 2019; Raaj et al., 2019). For instance, the Intel RealSense camera provides in real time a full hand skeleton with the topology presented in figure 4.2. Figure 4.2 (a) presents the structure of the hand skeleton, whereas figure 4.2 (b) details more exactly its topology and the indexation order of the skeleton's joints.

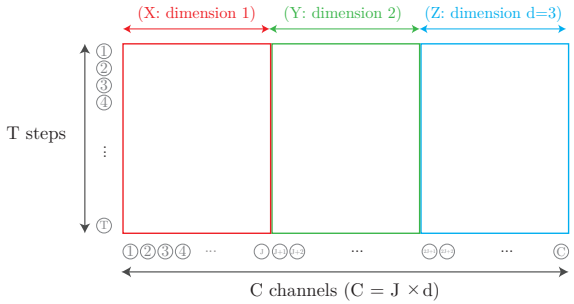
A skeleton is a graph with a fixed adjacency matrix of size $J \times J$. Its J nodes $\mathbf{p}_1, \ldots, \mathbf{p}_J$ are called joints, and its adjacency matrix describes the connections between joints, i.e. the skeletal structure. In the physical world, joints usually represent distinct and precise human body articulations, some of those joints being connected by bones as indicated by the adjacency matrix. Since the structure of human skeletons does not usually evolve over time, one can only consider a skeleton as being an ordered set of joints:



(a) **Projected view of a hand gesture.** A hand gesture is a sequence of a skeleton's joints' poses over time.



(b) **Gesture as a 3D tensor.** A gesture can be represented by a tensor $\mathbf{g}$ whose shape is (T,J,d) where T is the sequence duration, J the number of joints in the skeleton and d the dimensions ($d = 3$ for 3D (x,y,z) gestures). While this 3D tensor could be viewed as a 2D colored image, we avoid using such data visualization method in this chapter, as it visually mixes different (xyz) dimensions that we compare in the last section of this chapter.



(c) **Gesture as a 2D monochrome image (flattened tensor).** A gesture can also be represented by a tensor $\mathbf{g}$ whose shape is (T,C) where $C = J \times d$ by concatenating the xyz dimensions along the joints axis in order to reshape the 3D tensor. The resulting 2D tensor can be viewed as a 2D mono image, where each line represents a different time step and where each column represents a 1D channel. The 1D channels are in the following order: $x_1, x_2, \ldots, x_J, y_1, y_2, \ldots, y_J, z_1, z_2, \ldots, z_J$. Important note: while the image is monochrome, fake colors are used in amplitude-color-coded visualizations for better readability purposes.

Figure 4.3 – Illustration of different pose sequence representations. The 2D image representation used for visualizations in all this chapter is described in subfigure (c).

$$\mathbf{s} = (\mathbf{p}_1, \ldots, \mathbf{p}_J) \tag{4.1}$$

Joints are vectors that can convey 2D-3D information as well as any other relevant information computed or provided by sensors, e.g. velocity and acceleration information. In practice, joints components usually represent the orientation or the position or the corresponding physical articulation.

In this thesis, we assume each joint $\mathbf{p_i}$ is represented by its position in the 3D physical world:

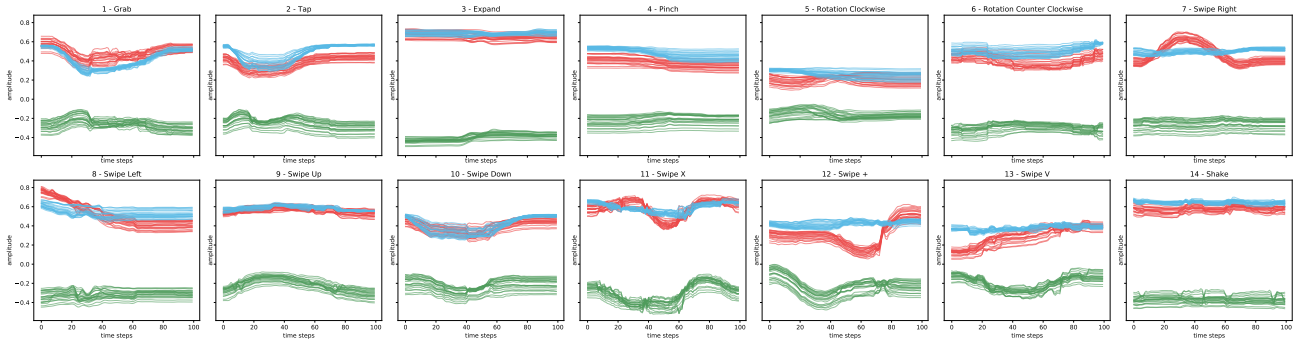


Figure 4.4 – Visualization of one random gesture (sequence) of each class (DHG 14 classes). All the x (respectively the y and z) channels are plotted in blue (respectively in green and in blue).

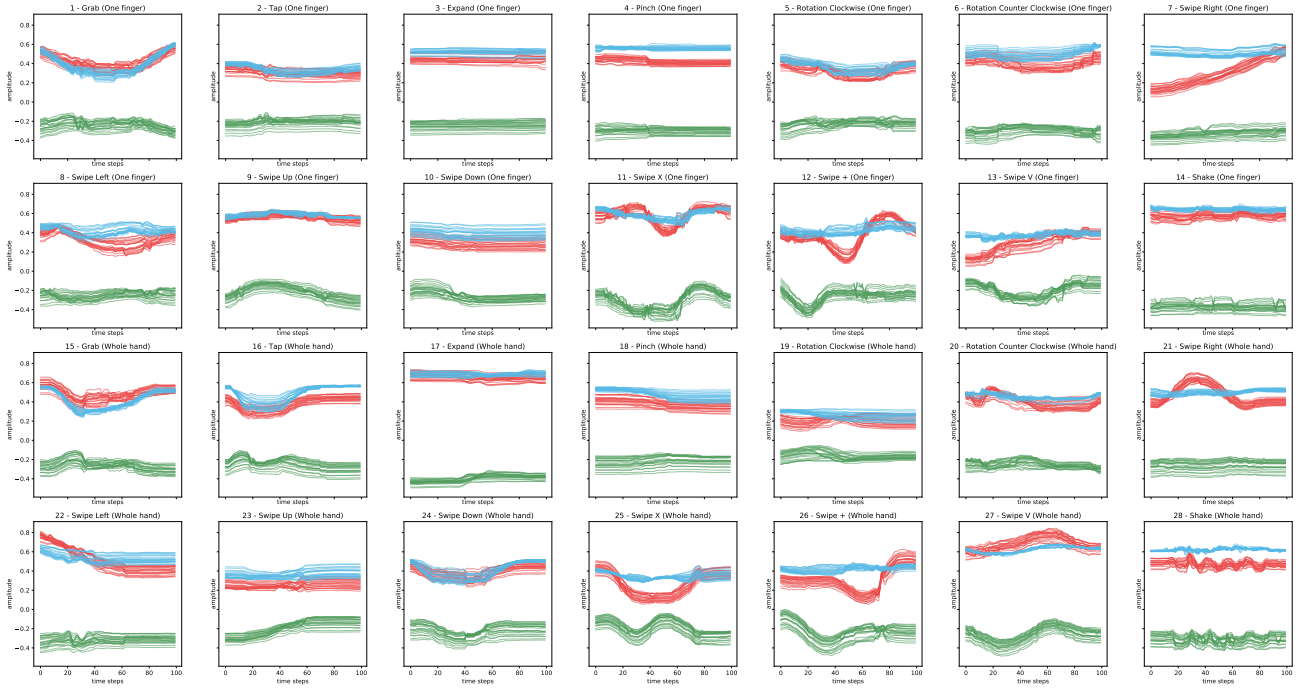


Figure 4.5 – Visualization of one random gesture (sequence) of each class (DHG 28 classes). The sequences are labelled according to 14 or 28 label classes, depending on the gesture represented and whether the number of fingers used is distinguished -with one finger or with the whole hand- or not.

$$\mathbf{p_i} = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} \tag{4.2}$$

A gesture $\mathbf{g}$ is represented as the evolution of the skeleton $\mathbf{s}$ over time:

$$\mathbf{g} = (\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(T)}) \tag{4.3}$$

where T is the duration of the sequence.

To summarize, a gesture can be viewed as a 3D tensor of shape:

$$(T, J, d)$$

where T is the sequence duration, J the number of joints and d their dimension: the first axis of the tensor is indexed by time indices, the second axis is indexed by joints, and the last axis is indexed by joint (position or rotation) components, i.e. dimensions. A projected view of a hand gesture is proposed in figure 4.3 (a) with the associated 3D tensor in figure 4.3 (b).

The 3D shape of a gesture tensor allows different visualizations methods, the most classical one being a simple time-series plot. As an illustration of a such plot, figure 4.4 illustrates the evolution of different gestures in the DHG14 case of the DHG14/28 dataset (De Smedt et al., 2017), while figure 4.5 illustrates the evolution of the same gestures in the DHG28 case of the DHG14/28 dataset (the DHG14/28 dataset (De Smedt et al., 2017) is described later on in details in section 4.3.3). In both figures 4.4 and 4.5, the horizontal axis of each plot represents time whereas the vertical axis represents the amplitude of the channels.

Another useful alternative representation of the same sequences is devised in details in figure 4.3 (c) and used for data visualizations through all the chapter. The representation consists in a 2D tensor of shape

$$(T, C)$$

where $C = J \times d$ are 1D time series channels. To obtain the C channels, the xyz dimension axis of the 3D (T, J, d)-shaped gesture tensor is concatenated along the joints axis, so that the 1D channels respectively represent the temporal evolution of $x_1, x_2, \ldots, x_J, y_1, y_2, \ldots, y_J, z_1, z_2, \ldots, z_J$, in that order. An example of such color-coded representation is proposed in figure 4.6. In figure 4.6, each one of the 28 plots represents a gesture sequence. The vertical axis represents time, whereas the horizontal axis represents all the x, y, and z channels. To represent the amplitude of the channels, a purple-blue to orange-yellow colormap is used: low amplitudes are represented with a purple-blue color whereas high amplitudes are represented with an orange-yellow color. In figure 4.6, for each gesture (i.e. for each plot), the evolution over time of the amplitudes of the channels are clearly visible.

4.3.2 SkelNet Neural Network Architecture

We propose a neural network architecture for gesture recognition.

The neural network architecture we introduce is detailed and explained below. It is also depicted in figure 4.8 for the overall architecture and in figure 4.7 for a more detailed illustration with the different temporal CNN branches but without the preprocessing module.

The architecture mostly consists of three blocks.

First, a preprocessing module is proposed to generate (learned) features that may be more relevant than the original input, e.g. by mixing information from different channels into new mixed channels. To that extent, we propose to use a linear module to apply spatial combinations (over the "channels" axis) in this chapter. However, that preprocessing module may be extended to more complex modules in future works, e.g. to modules that make use of graph convolutional layers.

In the second block of the model, n temporal-feature extractor modules are used (by default $n = J$), each of them taking a one-dimensional time series for its input and outputting a vector of features.

Third, fully-connected layers take as input the n vectors and predict the gesture category.

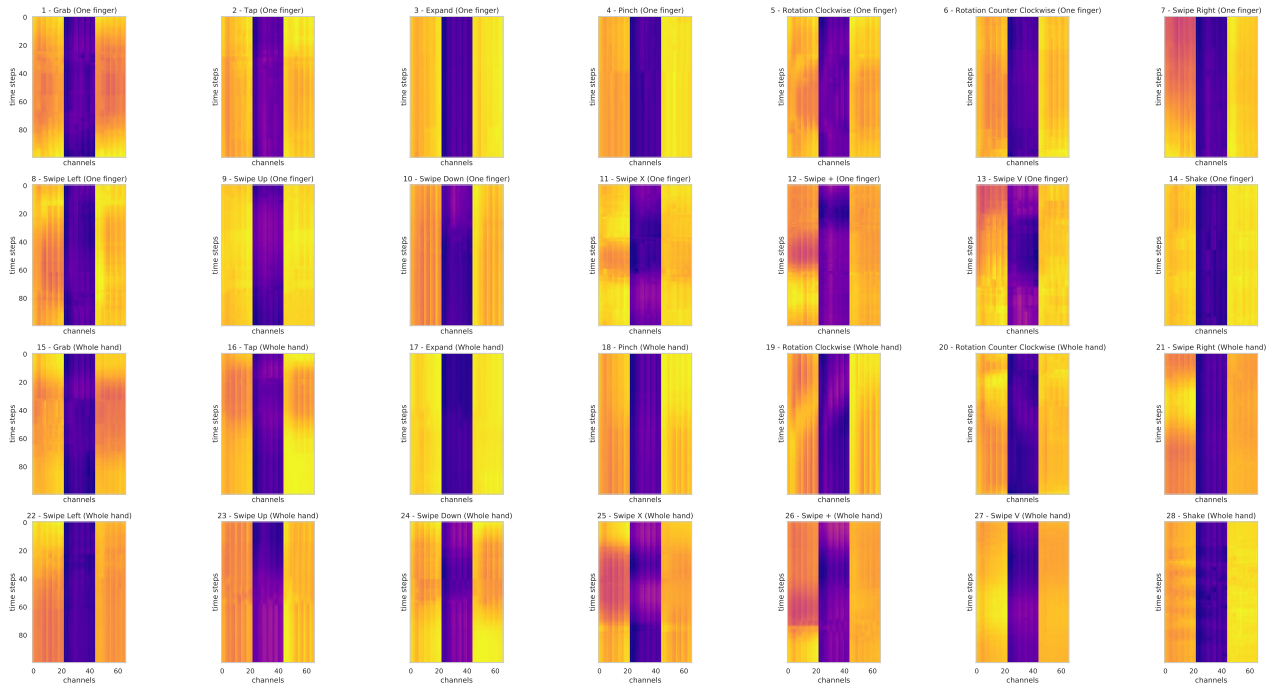


Figure 4.6 – Amplitude-color-coded visualization of one random gesture (sequence) of each class (DHG 28 classes). Examples used are the same as the ones in figure 4.5 (and figure 4.4). Abscissa: all x channels, all y channels and all z channels (in that order). Ordinate: Time steps. Color: from purple-blue (low values) to orange-yellow (high values).

The three blocks of the neural network are trained jointly on a skeletal-based or landmark-based gesture recognition task in an end-to-end manner.

The feature engineering is conveyed in the n extractors crafting the vectors of features. All these ones share the same architecture (but not necessarily the same weights), and are based on Convolutional Neural Network (CNN).

In an extractor module, every channel is fed to 3 branches, namely 2 modules for temporal feature extraction that both follow a convolutional neural network scheme and 1 pooling branch, that we call "residual" branch in this chapter. An illustration of an extractor module is proposed in figure 4.7, on the right side of the figure (with an input channel on the top of the illustration, the three processing branches vertically on the middle, and an output channel on the bottom).

The convolutional neural scheme used consists in a sequence of several convolutional layers made of a convolution followed by an average pooling of size 2, the convolution kernel sizes for the 2 branches being respectively equal to 3 and 7. The use of two different temporal convolution kernel sizes provides the network the ability to directly work at different time resolutions.

While the pooling branch that we refer to as a "residual" branch only performs a pooling operation -in order to subsample the original channel input to a much shorter duration- it can still arguably be considered as a "residual" branch. Indeed, its role is to skip the two convolutional branches (thus matching the definition of a residual branch given in chapter 2). This can been seen clearly on the right side of the figure 4.7 where an extractor module is illustrated: in the extractor, information present in the input vector (top) can flow through the residual pooling branch (middle) thus effectively skipping the two convolution branches (left and right). The difference between this residual pooling branch and a

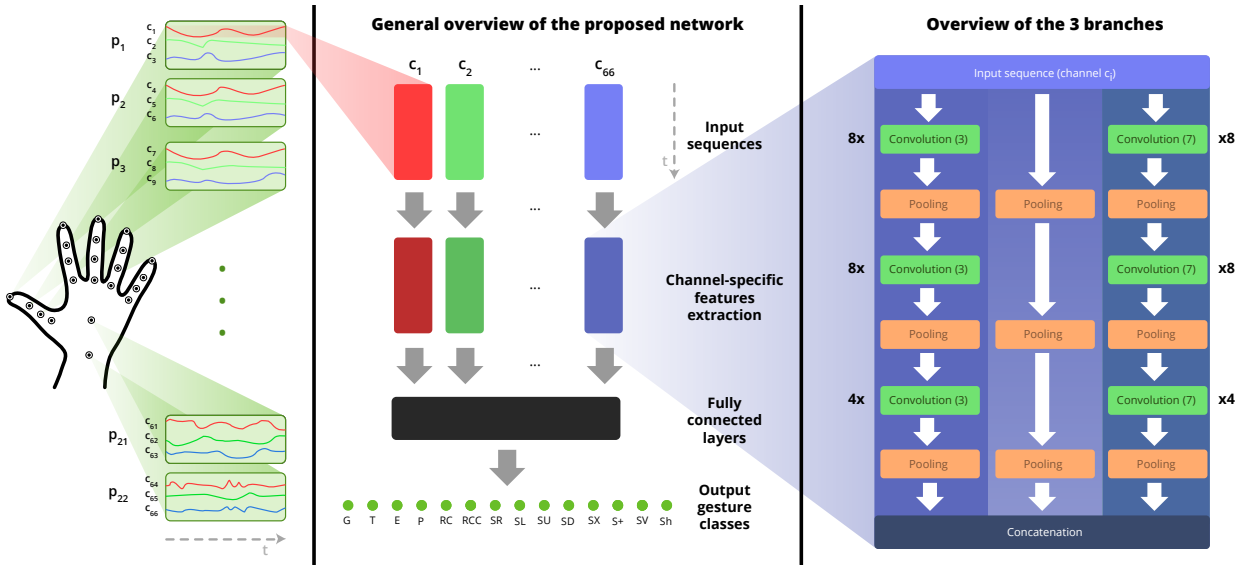


Figure 4.7 – Illustration of the SkelNet parallel convolutional neural network without the preprocessing module. Every channel is processed separately before the Multi Layer Perceptron. The parallel feature extraction module presented on the right is not shared between the 66 channels.

basic residual identity branch being that the residual pooling branch has to perform as many pooling operations as there are pooling operations in the convolutional branches, in order for the output "time" dimension of the three branches to match[3].

Finally the several vectors outputted by these extractor module's branches are concatenated into a larger feature vector, which serve as an input for the fully connected neural classifier.

For a more formal outlook of an extractor let $\mathbf{h}^{(m,\beta)}$ be the input of the m-th convolution layer of the β branch, for $m \in \mathbb{N}^*$ and $\beta \in \{1,2\}$. Let $K^{(m,\beta)}$ be the corresponding number of feature maps, let $\mathbf{W}_k^{(m,\beta)}$ be the k-th convolution kernel of the m-th layer in the branch β (for $k \in [1,\ldots,K^{(m,\beta)}]$), and let $\mathbf{b}_k^{(m,\beta)}$ represent the bias of the k-th filter map in the m-th layer in the branch β. The output $\mathbf{h}^{(m+1,\beta)}$ of the $m-th$ convolution layer is given by:

$$\mathbf{h}^{(m+1,\beta)} = \sigma(\mathbf{h}^{(m,\beta)} * \mathbf{W}_k^{(m,\beta)} + \mathbf{b}_k^{(m,\beta)}) \tag{4.4}$$

with σ being the activation function.

The outputs $\mathbf{h}^{(m+1,1)}$, $\mathbf{h}^{(m+1,2)}$ of the last layer m, and the output of the pooling branch are merged together, into a vector $\mathbf{v}$. The vectors $\mathbf{v}$ for every channel are passed as the inputs of the fully-connected layers.

All of the subsampling layers used in an extractor perform an average pooling with a temporal size of 2. An average pooling step computes the average value of a feature in a neighborhood (of 2 time steps in our case), while the other popular pooling technique, namely the max pooling, keeps the maximum value in the neighborhood. (Boureau et al., 2010) have shown empirically that the latter technique can increase the performances in the image recognition field, compared to an average pooling step. However, as shown later in section 4.4.4, results we obtained on hand gesture classification show that slightly higher performances are achieved when average pooling is used.

[3]While not strictly required, this is useful both for convenience -e.g. for concatenation- and for flexibility purposes.

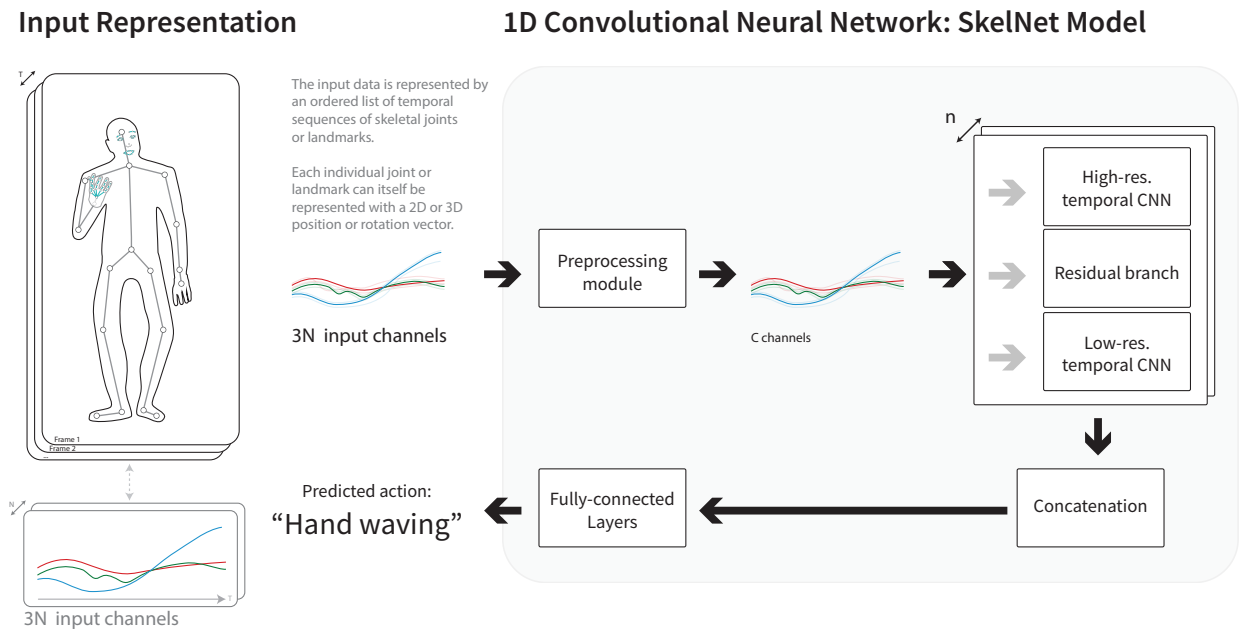


Figure 4.8 – Illustration of the proposed SkelNet neural network approach. The channels are first preprocessed before being given to the temporal feature extractor module on he right. This parallel feature extraction module on the right can be shared or not between the channels depending on the experiments we perform. The resulting features are finally given as input of the fully-connected layers which perform the final classification. The whole model is trained end-to-end on recognition tasks.

The fully-connected layers have 2 hidden layers with respectively 1024 and 128 hidden units. They use ReLU activation functions $\sigma(x) = ReLU(x) = max(0,x)$, at the only exception of the output neurons which uses a softmax activation function for classification purpose. The use of residual branches in our architecture is inspired from the original Residual Networks article (K. He et al., 2016). Skip-connections and residual branches smooth the training loss landscapes of neural networks, easing the network training optimization process, as shown in (H. Li et al., 2018).

The number of fully connected hidden layers and the size of these layers result from grid search experiments we performed on the DHG 14/28 dataset where we observed that the use of 2 hidden layers slightly outperformed single hidden layer networks in terms of final test accuracy. Moreover, the 2-hidden-layer architectures with large layer sizes (denoted fc_layer_0 and fc_layer_1) are more accurate than the smaller architectures for the DHG 14/28 dataset. Yet, the experiments did not show a clear optimum, hence the values chosen of 1024 and 128 neurons respectively.

Our model uses a preprocessing module applying a linear layer over the “channels” axis, to combine information from the various time sequences $(\mathbf{s_i}(t_k))_{k\in[0,T-1]}$ in order to capture meaningful spatial patterns to be submitted to the downstream convolutional layers (the combination could for instance capture relative distances and similar meaningful predictors).

If $\mathbf{s} = (\mathbf{s_i}(t_k))_{i,k}$ is a gesture tensor of shape $T \times C$ ($T = 100, C = 66 = 22 \times 3$), and $\mathbf{y} = (\mathbf{y_j}(t_k))_{j\in[0,C_{out},k]}$ the output of the preprocessing module, where C_{out} is the size of the output of the preprocessing layer, we learn the linear transformation (over the “channels” axis):

$$\mathbf{y} = \mathbf{W}_{LSC}\,\mathbf{s} + \mathbf{b}_{LSC} \tag{4.5}$$

This transformation is constant across time in order to keep the temporal information untouched.

4.3.3 Evaluation protocol

In this chapter, we study the design choices behind the neural network architecture proposed. We evaluate qualitatively and quantitatively the recognition performance of models belonging to this family of neural networks. Unless specified otherwise, all the experiments are performed three times and the mean and the standard deviation of the results are provided when relevant.

We train the model on one machine with a GPU (NVIDIA GeForce GTX 1080 Ti) using CUDA and CuDNN accelerated GPU operations. Still, a comprehensive hyperparameters search is not realistically affordable for the models' family on large-scale datasets with our hardware due to the duration of model trainings and the count of model variations, even when considering random search, bayesian optimization or improved methods; e.g. like hyperband from (L. Li et al., 2016) or BOHB from (Falkner et al., 2018).

Though being skeleton-type agnostic, the model was first designed for hand skeletal gesture recognition. For comprehensive analysis, all experiments are performed on a medium-sized hand gesture dataset called the DHG 14/28 dataset (De Smedt et al., 2017). The DHG 14/28 dataset is presented more thoroughly in the subsection "**Datasets**" below. To study the applicability of our approach on other datasets and to study the influence of input data representation while alleviating the training time of the numerous neural network variations, some but not all experiments are also performed on a bigger-sized body skeletal action recognition dataset called the NTU RGB+D dataset and on a small-to-medium-sized facial emotion recognition dataset called the RAVDESS dataset. The three datasets are presented below.

Datasets

The main dataset we use to study the proposed architecture is the Dynamic Hand Gesture-14/28 (DHG 14/28, or DHG) dataset created and introduced by (De Smedt et al., 2017) in the SHREC2017 - 3D Shape Retrieval Contest.

The DHG 14/28 dataset is a dataset of hand gesture sequences for supervised learning gesture classification. It contains a total of 2800 examples, the gestures being performed by 28 different participants in total. Gesture duration is variable. Each labeled example consists of the raw data sequence returned by the camera, associated with two labels representing the category of the recorded gesture. For all sequences a depth image of the scene is provided at each time step, alongside with both a 2D and a 3D skeletal representation of the hand. The camera provides information about the joints positions but not the joints rotations.

Each gesture falls into one of 14 categories as described in table 4.1.

Moreover, each gesture can be performed with either only one finger or with the whole hand. It means that gestures are classified with either 14 labels or 28 labels, depending on the number of fingers used. In our experiments, the RGB+D images are discarded and we only use the 3D hand skeletal representation of the hand.

In the standard evaluation protocol introduced in the SHREC2017 - 3D Shape Retrieval Contest, the training and testing sets of the DHG 14/28 dataset are pre-split, as described at the end of this section.

Class index	Name of the gesture	Type
1	Grab	Fine
2	Tap	Coarse
3	Expand	Fine
4	Pinch	Fine
5	Rotation Clockwise	Fine
6	Rotation Counter Clockwise	Fine
7	Swipe Right	Coarse
8	Swipe Left	Coarse
9	Swipe Up	Coarse
10	Swipe Down	Coarse
11	Swipe X	Coarse
12	Swipe +	Coarse
13	Swipe V	Coarse
14	Shake	Coarse

Table 4.1 – Gestures in the DHG 14/28 dataset.

Some of the works in the literature use another evaluation protocol for the DHG 14/28 dataset which consists in a leave-one-out cross-validation protocol, and distinguish these two protocols by calling the dataset the “DHG 14/28 dataset” when using the leave-one-out protocol and calling it the “SHREC’17 dataset” when using the train/test split protocol. We use the train/test split protocol and refer to the dataset as the “DHG 14/28 dataset”.

The dataset we use to study applicability of the approach on other datasets and the input data type representation influence is the NTU RGB+D dataset introduced in (Shahroudy et al., 2016).

The NTU RGB+D dataset is a large-scale full-body sequences dataset for supervised learning of human activity classification. It contains a total of 56880 examples, the actions being performed by 40 different participants in total. Action duration is variable.

Except for 302 sequences where skeleton data is missing or incomplete, a RGB+D image of the scene is provided at each time step, alongside with IR sequences and a 3D skeletal representation of the human body. Both the joints positions and the joints rotations are provided. In our experiments, the IR and RGB+D images are discarded and we only use the 3D body skeletal representations. Each sequence is captured by three Kinect cameras, each camera having different orientations.

An illustration of a RGB frame from a sample NTU RGB+D dataset sequence ("shaking hands" action), as well as the two associated 3D poses overlaid on top of it, is proposed in figure 4.9. One can notice that the poses estimated by the Kinect camera in figure 4.9 are noisy (especially on the feet), but still contain enough information to describe the action performed (shaking hands).

Two evaluation methods are proposed for this dataset: cross-view evaluation and cross-subject evaluation. In cross-view evaluation, the participants are split into a training and a testing group, while in cross-view evaluation, two cameras make up a training group while the last one makes up a testing group. Because we only use the skeletal information without RGB+D or IR, and because the skeletal

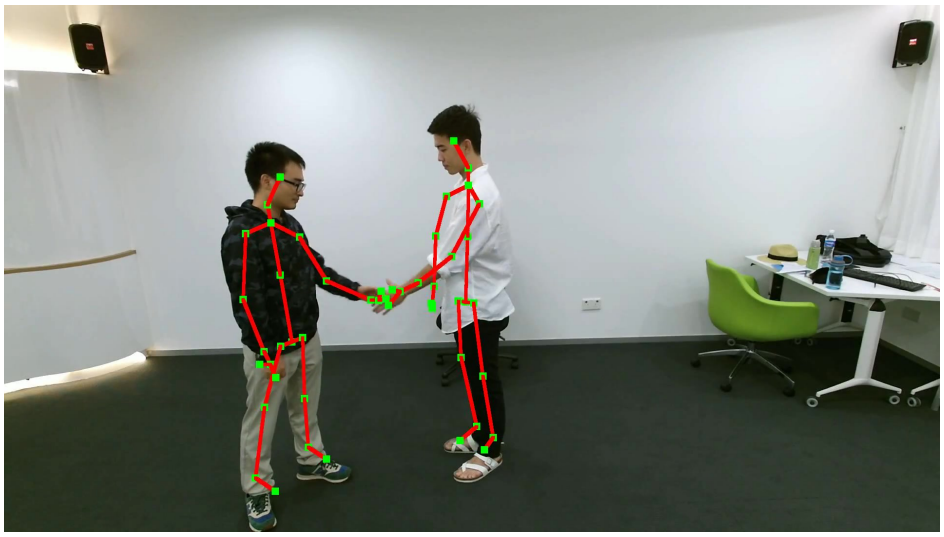


Figure 4.9 – Two actors poses (overlaid on the associated video frame) from the NTU RGB+D dataset. While the poses estimated by the Kinect camera are noisy (especially on the feet), they contain enough information to summarize most of the scene information relevant to describe the action performed (shaking hands).
Frame from (Shahroudy et al., 2016).

information is supposedly the same for the three cameras up to a rotation transformation from a camera to another, we choose to use the cross-subject evaluation method.

In the NTU RGB+D dataset each action falls into one out of 60 possible classes. The classes are divided into three major groups: 40 daily actions (`drinking`, `eating`, `reading`, etc.), 9 health-related actions (`sneezing`, `staggering`, `falling down`, etc.), and 11 mutual actions (`punching`, `kicking`, `hugging`, etc.).

The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS) dataset is an audiovisual dataset for supervised learning human emotion classification introduced in (Livingstone et al., 2018). It contains a total of 7356 examples of realistic emotional speeches and songs, recorded by 24 different professional actors in total.

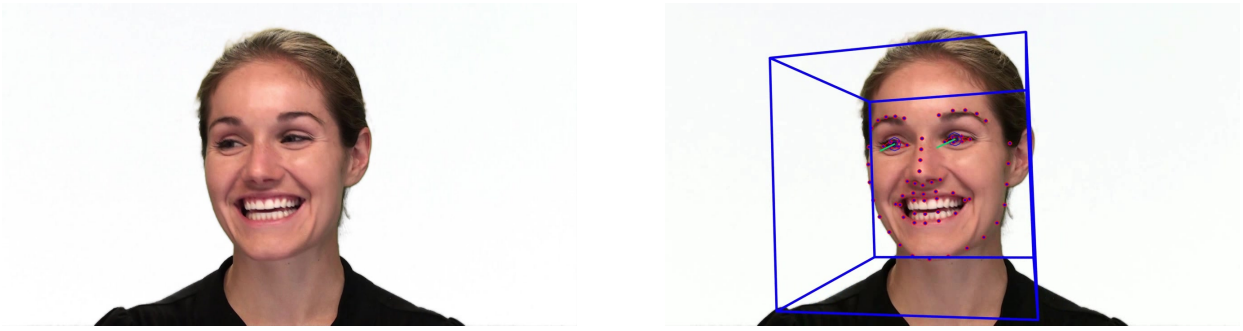
Each recording falls into one of the following possible emotions: `calm`, `happy`, `sad`, `angry`, `fearful`, `surprise`, `disgust` and `neutral`.

Speech duration and song duration are variable. Each labeled example consists of the raw video (audio and RGB frames) returned by the camera filming the actor face and an associated label that describes the emotional class.

For each sequence, we discard audio and extract facial landmarks of the actors from the video frames sequence using a state-of-the-art convolutional experts constrained local model proposed in (Baltrusaitis et al., 2018). We obtain 68 ordered facial landmarks that represent the actor face.

This extraction is illustrated in figure 4.10. Figure 4.10 represents a video frame of an actress' recording: on the left (a) the original frame, and on the right (b) the information we extracted from it, overlaid on the original frame. The extraction model from (Baltrusaitis et al., 2018) estimates facial landmarks (red-circled blue dots), global head orientation (blue box) and eye gaze directions (green lines). We only use the facial landmarks information.

While the human face physiology is different from the human body or hand physiologies, these



(a) Original video frame (b) Extracted facial landmarks

Figure 4.10 – Extraction of facial landmarks. On the left (a) the original frame from the dataset, and on the right (b) superposed over it, in red, the facial landmarks we extract.
Frame from the RAVDESS dataset (Livingstone et al., 2018).

landmarks can be seen as the nodes of a (non-rigid) face skeleton graph. Our neural network model's input is the sequence of the positions of the facial landmarks over time.

Regardless of the dataset used, the sequences exhibit variable lengths whereas our neural networks take a fixed-size input. To that end, we consider the trajectories of joints positions or joints rotations as unidimensional sequences that we temporally re-sampled. To temporally resize these sequences, a simple linear interpolation (over the "time" T axis) of the time series is used. Edge case values at the boundaries of the input are filled with a reflection. After interpolation, every sequence has a fixed length. For the DHG and NTU RGB+D datasets, the final length is 100 time steps, whereas for the RAVDESS dataset the final length is 122 time steps.

Datasets splits

The DHG 14/28 dataset is split into 1960 train sequences (70% of the dataset) and 840 test sequences (30% of the dataset).

The NTU RGB+D dataset split follows the cross-subject benchmark proposed by the dataset authors: actions performed by subjects 1, 2, 4, 5, 8, 9, 13, 14, 15, 16, 17, 18, 19, 25, 27, 28, 31, 34, 35, 38 are considered as train sequences, whereas the other actions are considered as test sequences. There are 40320 train sequences (71% of the dataset) and 16560 test sequences (29% of the dataset).

The RAVDESS dataset is split into 5150 train sequences (70% of the dataset) and 2206 test sequences (30% of the dataset).

For all the trainings and all the datasets, we perform a 5-fold cross-validation during training to avoid overfitting.

Metric, Loss, Regularization & Optimization algorithm

For each model variation, we provide the model accuracy on the test set.

To predict the class C_m of a sequence y_i, the output ξ_i of the last fully-connected layer of the model is normalized by the softmax function to give the posterior probability:

$$p_{model}(y_i \in C_m) = \frac{e^{\xi_i^m}}{\sum_{k=1}^{M} e^{\xi_i^k}} \tag{4.6}$$

During training, we minimize the cross-entropy loss $\mathscr{L}_{CE}$ between the empirical distribution defined by the training set and the probability distribution defined by the model being trained:

$$\mathscr{L}_{CE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{m=1}^{M} \mathbb{1}_{y_i \in C_m} \log(p_{model}(y_i \in C_m)) \tag{4.7}$$

where N is the total number of sequences, M is the cardinal of the set of classes $\{C_1, \ldots, C_m\}$, $\mathbb{1}_{y_i \in C_m}$ is the value of the indicator function of the i-th sequence belonging to the m-th class and $p_{model}(y_i \in C_m)$ is the probability by the model (after the softmax) that the i-th sequence belongs to the class C_m. Minimizing the cross-entropy is equivalent to maximizing the log-likelihood.

Model is regularized with dropout (N. Srivastava et al., 2014). The drop rate is set to $p = 0.4$ for all the models evaluated in this chapter. The choice of the drop rate for the dropout results from grid search experiments on drop values from 0.0 to 0.8. Changing the drop rate increases up to +2.5% the model accuracy on the DHG 14/28 dataset, however no consistent relation between accuracy evolution was observed with regards to dropout rate.

To train the models, we use the Adam optimization algorithm (D. Kingma et al., 2014) which calculates an exponential moving average of the gradient and the squared gradient. For the decay rates of the moving averages we used the parameters $\beta_1 = 0.90$, $\beta_2 = 0.99$. The value of the parameter ε is $\varepsilon = 1.00 \times 10^{-8}$.

For our experiments, we use a learning rate of $\alpha = 1.00 \times 10^{-3}$ unless specified otherwise, which is the default value proposed for Adam (1.00×10^{-3}). The optimal initial learning rate we found for the reference model presented in the next section is $\alpha = 5.00 \times 10^{-5}$.

Regarding the number of training epochs, we perform early stopping: if the validation loss does not improve during the 50 last steps by more than 0.01%, we stop the training in order to prevent the network from overfitting the training data.

Weights Initialization & Batching

Each training batch contains a set of 32 skeletal gesture sequences and their associated labels. The choice results from experiments we ran on batch sizes in the range of values : $batch\_size \in \{8, 16, 32, 64, 128, 256, 512, 1024$
Even though higher batch sizes are quicker to process with parallelization, it has been shown by (Keskar et al., 2016) that using large batches often lead to poorer generalization.

We use the Xavier initialization (also known as Glorot uniform initialization) from (Glorot et al., 2010) to set the initial random weights for all the weights of our model.

While using Kaiming initialization (also known as He uniform initialization) from (K. He et al., 2015) reportedly helps neural networks with ReLU-like activations to converge much earlier than when using Xavier initialization, no significant difference between the two initialization methods was observed. We use Xavier initialization for all the trainings.

4.4 Experiments

In this section, we study models from the SkelNet family and analyze the factors that influence their performance at gesture recognition.

We perform experiments on the input data representation, as well as experiments on the models. Regarding experiments on the models, both the architecture of the neural networks and the choice of the hyperparameters are studied, all other things being kept equal unless specified, including the original input data representation.

4.4.1 Reference models

In this section we introduce two reference models that we call the "SkelNet reference model" and the "SkelNet-Hand model". They are introduced below.

Through all the chapter, the terms "the model", "the reference model", and "the SkelNet reference model" all refer to the neural network of the SkelNet family with the following architecture hyperparameters: no preprocessing module, average pooling of length 2, PReLU activation (K. He et al., 2015), standard convolution with only 1 single convolutional layer ($C_{out} = 22, groups = 22$) and 2 hidden fully-connected layers of size ($N_{out,1} = 1400, N_{out,2} = 42$). Regarding the training hyperparameters, we fix a dropout rate equal to 0.4 and an initial learning rate of 1.00×10^{-4}.

The selection of PReLU as the activation function results from experiments where the following different activation functions were tested: rectified linear unit (ReLU), parametric rectified linear unit (PReLU) from (K. He et al., 2015), exponential linear unit (ELU) from (Clevert et al., 2015) and Swish from (Ramachandran et al., 2017). The number of fully connected hidden layers, the size of these layers and the group count all result from experiments too.

This model is very shallow[4] and lightweight[5] compared to almost every other neural network proposed in the literature. As such, the model is ready for on-device computations on embedded systems. While slightly below state-of-the-art, the model exhibits a good performance for hand gesture recognition, especially for a lightweight model, as presented in table 4.3.

In the section 4.4.5, we analyze the impact of weight sharing between temporal feature extractor modules, depending on sharing choices such as the convolution abstraction level (i.e. whether the convolution is applied to raw signals, or to already convoluted signals) for instance. As such, a SkelNet model with several convolutional layers is required for these experiments.

To that end, we consider a second reference model that we refer to as "SkelNet-Hand". The SkelNet-Hand model is a standard SkelNet model with the following configuration: no preprocessing module, average pooling of length 2, ReLU activation, standard convolution with 3 convolutional layers, and 1 hidden fully-connected layer of size ($N_{out} = 1996$). The kernel size for all the convolutions is 3, and in each branch the 3 convolutions have respectively 8, 8 and 4 feature maps. It is important to note that in the SkelNet-Hand model, no extractor module is shared by default (i.e. C extractor modules are used,

[4]Both the SkelNet-1024-128 and the SkelNet-1400-42 have only one convolutional layer.

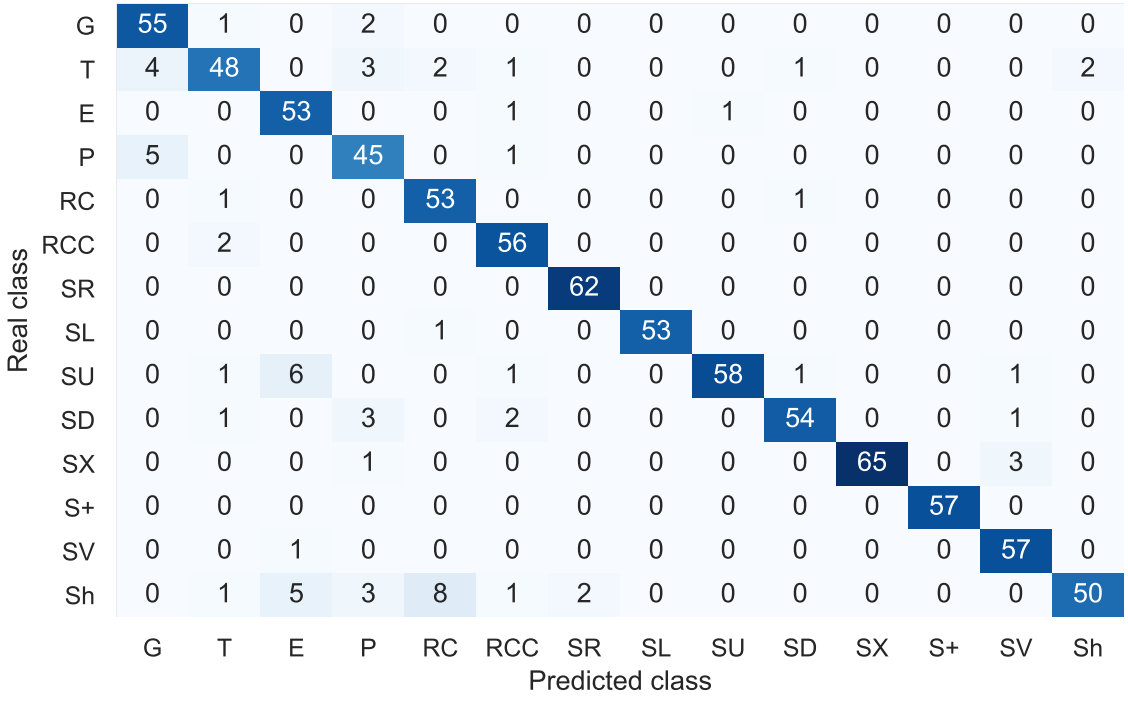
[5]The SkelNet-1024-128 has ∼2.4M parameters: 2404393 parameters for the 14 classes case and 2406199 parameters for the 28 classes case. The SkelNet-1400-42 has ∼3.1M parameters: 3144547 parameters for the 14 classes case and 3145149 parameters for the 28 classes case.

where C denotes the input channels count). For the sake of performing fair comparisons, experiments involving the SkelNet-Hand model do not make use of data augmentation techniques (the experiments involving the SkelNet reference model do not make use of data augmentation techniques either, unless explicitly stated otherwise).

The "SkelNet-Hand" model displays performances that are comparable to the "reference SkelNet" model performances, as shown in table 4.3.

The confusion matrices of the SkelNet-Hand model for the 14 actions and 28 actions are presented in figure 4.11 and in figure 4.12 respectively. The confusion matrices show that the SkelNet-Hand model is robust to each action class.

Figure 4.11 – Confusion matrix for the SkelNet-Hand model on the DHG14/28 dataset (14 classes)

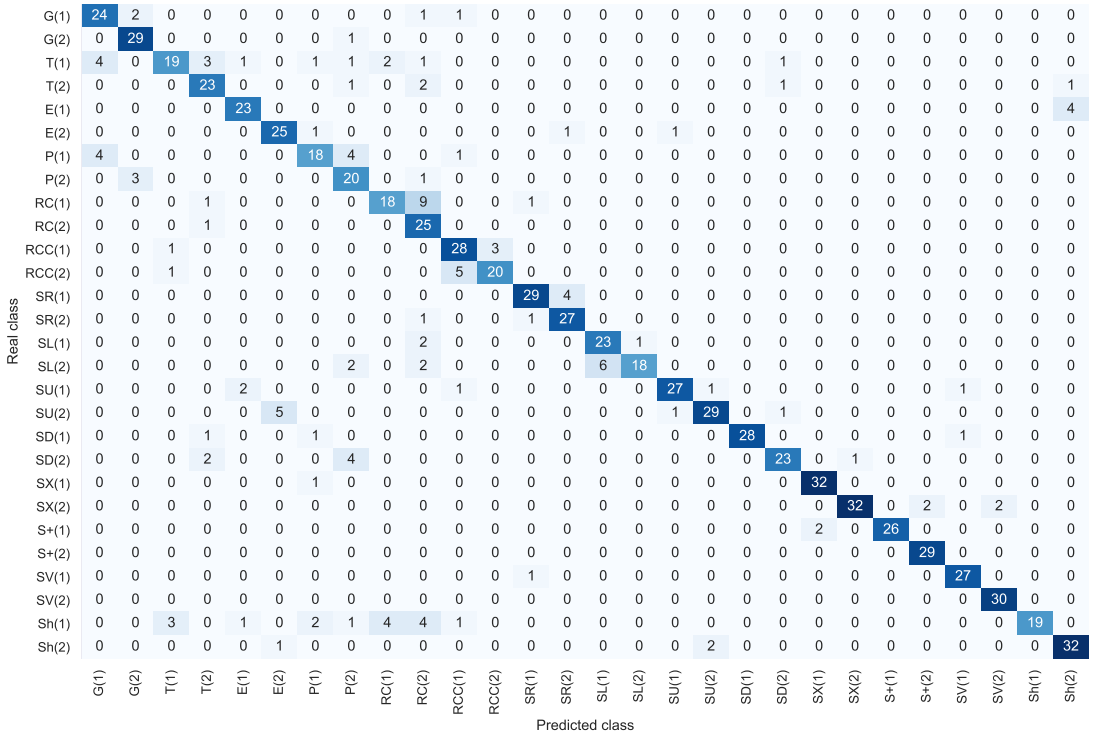


The table 4.2 details precision, recall and F_1 values for the SkelNet-Hand model, and confirms that robustness.

4.4.2 Input Representation

The neural network is designed to extract temporal features from gestures and then to merge them in order to perform the final classification. Intermediate representations of the gestures are entirely learned by the model, without any manual intervention. However, since model representations are based on the input data representation -chosen by human practitioners-, finding an appropriate input representation is crucial to leverage the full potential of the network. In this subsection, we evaluate how the original input representation influences the final model accuracy. We specifically study the effect of Procrustes

Figure 4.12 – Confusion matrix for the SkelNet-Hand model on the DHG14/28 dataset (28 classes)



standardization, data augmentation, sequence length, position and rotation representations and the time location of information inside the gesture sequences.

Procrustes

The shape of a hand skeleton can be defined as its geometry regardless of its position, orientation, and size. Procrustes Analysis (PA) (Krzanowski, 2000) and its extension Generalized Procrustes Analysis (GPA) (Gower, 1975) are two means by which shape variables can be obtained from landmark and skeletal data. In layman's terms, the aim of Procrustes Analysis is to align one shape with another, without using any symmetry. In Greek mythology, Procrustes was a bandit who attacked people and forced them to arbitrary "fit" on his bed: he would turn them and stretch out or cut off their legs in order to make sure they fit to the size of the bed. Hence, Procrustes Analysis aims, in a manner inspired by the eponymous mythological bandit, at fitting one shape to an other (arbitrary) standard one.

Procrustes Analysis is commonly used as a post-processing alignment method between two poses in the full-body human pose estimation domain and in the full-body human pose prediction domain (Bogo et al., 2016; Martinez et al., 2017b; Moreno-Noguer, 2017). Even if Procrustes Analysis has not been applied *per se* on hand skeletons for hand gesture recognition purposes before -to the best of our knowledge-, it has however already been successfully applied to very similar domains. First, in the static image hand recognition domain, PA has been used to align hand contours extracted from images A. K. Jain et al., 1999; Yörük et al., 2006. Finally, even closer to the experiment we perform in

	Ours			DE SMEDT *et al.*			Difference
Gesture	Precision	Recall	F_1-score	Precision	Recall	F_1-score	F_1-score
G	72.4%	94.8%	**82.1%**	67.5%	57.0%	61.8%	20.3%
T	71.2%	77.0%	74.0%	85.2%	87.0%	**86.1%**	-12.1%
E	84.7%	90.9%	**87.7%**	84.8%	87.0%	85.9%	1.8%
P	90.9%	78.4%	**84.2%**	52.1%	61.0%	56.2%	28.0%
RC	69.2%	98.2%	**81.2%**	80.0%	77.5%	78.8%	2.5%
RCC	97.8%	77.6%	86.5%	90.9%	85.5%	**88.1%**	-1.6%
SR	91.2%	100.0%	**95.4%**	85.1%	92.5%	88.6%	6.7%
SL	98.0%	88.9%	**93.2%**	78.4%	85.5%	81.8%	11.4%
SU	98.2%	79.4%	**87.8%**	89.3%	85.5%	87.4%	0.4%
SD	93.3%	91.8%	**92.6%**	80.8%	88.0%	84.3%	8.3%
SX	100.0%	89.9%	**94.7%**	95.8%	85.0%	90.1%	4.6%
S+	98.3%	100.0%	**99.1%**	90.2%	98.5%	94.1%	5.0%
SV	90.6%	100.0%	**95.1%**	93.2%	92.0%	92.6%	2.5%
Sh	96.2%	71.4%	82.0%	88.6%	81.0%	**84.7%**	-2.7%

Table 4.2 – Comparison of F_1 score for the SkelNet-Hand model in the 14 gesture classes case

this section, PA has also been applied in the facial expression recognition domain (S. Jain et al., 2011; Martins, 2008): PA is first used as a pre-processing method to align facial landmarks between faces, the aligned landmarks (i.e. faces) being then fed to a classifier in a second stage.

Finally, let's note that other pose standardization approaches exist, besides PA. In the human limbs (i.e. skeleton segments, or bones) lengths normalization approach proposed in Zanfir et al., 2013, average skeleton segment lengths are learned using training data. The proposed approach aims at explicitly normalizing human skeletons. To normalize a pose, the lengths of the limbs of the pose are adjusted to fit a desired length calculated from the learned lengths, while keeping body joint's angles constants during the transformation. Finally, the resulting pose is centered around the hip center. The motivation behind this skeleton normalization approach is conceptually very close to the motivation behind the use of Procrustes Analysis skeletal alignment approach: in both cases both scale and location of the skeleton are standardized. However, the PA alignment can arguably be more powerful: compared to the limbs lengths normalization approach from Zanfir et al., 2013, in PA other alignment transformations can be directly learned, including rotations of the skeleton. Moreover, PA alignment does not require the pose to have a root node, whereas the limbs lengths normalization approach proposed in Zanfir et al., 2013 does require a root node. As such, PA alignment can be used directly without adaptation to any other pose topology even if the topology does not have a root node: this is often the case for facial landmarks topologies, for instance.

Given two centred matrices A and B, Procrustes Analysis finds the linear transformation matrix Ω

Model	Accuracy (14)	Accuracy (28)
(Oreifej et al., 2013)	78.5	74
(Devanne et al., 2014)	79.6	62
(De Smedt et al., 2017)	82.9	71.9
(Ohn-Bar et al., 2013)	83.9	76.5
(J. Weng et al., 2018)	85.8	80.2
(De Smedt et al., 2016) (SoCJ + HoHD + HoWR)	88.2	81.9
(Caputo et al., 2018)	89.5	-
(Boulahia et al., 2017)	90.5	80.5
(Hou et al., 2018) (Res-TCN)	91.1	87.3
SkelNet-Hand	91.3	84.4
(X. Chen et al., 2019)	91.3	86.6
(X. S. Nguyen et al., 2019)	92.38	86.31
(Y. Li et al., 2019) (HG-GCN)	92.8	88.3
(Hou et al., 2018) (STA-Res-TCN)	93.6	90.7
(Maghoumi et al., 2018)	94.5	91.4
(F. Yang et al., 2019)	94.6	91.9
(Avola et al., 2018)	97.62	91.43
SkelNet-1400-42 ($C_{out} = 22, groups = 22$) ∼2.4M parameters	88.0	77.9
SkelNet-1024-128 ($C_{out} = 22, groups = 1$) ∼2.4M parameters	**91.5**	**84.0**
Last SkelNet model but with TCN instead of convolutional layers	91.3	**84.8**

Table 4.3 – Results on the DHG-14/28 SHREC'17 dataset using the train/test split protocol (in %)

that minimizes the following objective:

$$\arg\min_{\Omega} \|\Omega A - B\| \tag{4.8}$$

where:

$A \in \mathbb{R}^{(J,d)}$ is a centred matrix that represents a shape of J joints of dimension d,

$B \in \mathbb{R}^{(J,d)}$ is a centred matrix that represents a shape of J joints of dimension d,

with $J = 22$ joints and $d = 3$ (xyz) dimensions in the experiments.

The intuition behind Procrustes is to translate two vectors to a same origin in order to center them -if not already the case-, then to scale them down to unit-size and finally to optimally rotate them with respect to a norm minimization problem and with regards to the rigid transformations. The resulting vectors represent a standardized version of the original vectors, and lie on a curved space related to Kendall's shape space (Kendall, 1984) where gestures can be viewed as Riemannian trajectories (Ben Tanfous et al., 2018) and compared. When the matrix Ω is orthogonal, i.e. $\Omega^{\top}\Omega = I$, the orthogonal

Procrustes objective has an analytical solution given by:

$$\Omega = UV^\top \tag{4.9}$$

where U and V come from the singular value decomposition $U\Sigma V^\top$ of the matrix $BA^\top$.

We make use of Procrustes Analysis to create a Procrustes-standardized version of the gestures, and compare the performances of the model trained on these standardized gestures to the same model trained on original non-procrusted gestures. More precisely, we test two different Procrustes Analysis registration approaches: a "by-time step" registration approach, and a "full-sequence" registration approach.

Let J refer to the joints' count, and d to their dimensions, and T to the duration of the hand gesture sequence. The "by-time step" PA approach applies PA to individual skeletons (i.e. on tensors of shape (J, d)). The "full-sequence" PA approach considers a gesture as a single tensor of shape (T, J, d) that is first reshaped as to a shape of $(J \times T, d)$ by concatenating successive time steps along the the joints axis. A PA is performed on that $(J \times T, d)$ tensor.

Our motivation for a "by-time step" Procrustes Analysis can be explained as follows. The convolutional neural network models we introduced perform convolution operations on the temporal dimension in order to extract temporal patterns relevant for classification. As such, coarse gestures that mostly involve a global translation and/or a global rotation of the hand are relatively easy to classify by the models. However fine gestures are more difficult, as the inner shape of the hand evolves over time. As such, registering the current hand skeleton at each time step to a reference hand skeleton -e.g. the hand skeleton at the first time step- may arguably help the models to better compare the hand configuration between different time steps and thus better recognize fine gestures. We refer to this PA registration method as the "by-time step" separate skeleton registration. In equation 4.8, for a "by-time step" Procrustes Analysis, $B \in \mathbb{R}^{(J,d)}$ represents the first skeletal frame of a gesture sequence, while $A \in \mathbb{R}^{(J,d)}$ represents the current skeletal frame for a specific time step of that gesture sequence. For each individual time step, a PA is performed.

Our motivation for a "full-sequence" Procrustes Analysis can be explained as follows. All possible gestures are not evenly distributed in the physical space. As such, we define a reference gesture as the mean of all the gestures from the training set of the dataset. Registering a new gesture sequence to classify to this reference gesture may present an advantage as it could help the models to be more robust to slight global rotations and perturbations since the gesture is registered. Moreover, overall temporal evolutions may be better conserved with this registration than with the "by-time step" PA. However, we also would expect this registration method to have a major drawback: some gestures may appear almost identical after the registration: e.g. horizontal swipes might appear undistinguishable from vertical swipes. We refer to this PA registration method as the "full-sequence" registration. In equation 4.8, for a "full-sequence" Procrustes Analysis, $B \in \mathbb{R}^{(J\times T,d)}$ represents the reference gesture (i.e. the mean of all the gestures from the training set of the dataset), while $A \in \mathbb{R}^{(J\times T,d)}$ represents the gesture sequence to classify. Note that all time steps are concatenated to form the matrices A and B from the original tensors. Only one PA is performed.

We performed Procrustes Analysis on the full sequences considering either the sequences as a whole

("full-sequence" PA), or considering skeletons separately at each time step ("by-time step" PA), in order to compare if one of the two methods would lose more temporal information than the other.

For the 14 classes case, the SkelNet-1400-42 model achieves only a $31.3\%(\pm 2.9\%)$ accuracy when performing PA on the whole sequences and a $41.2\%(\pm 2.9\%)$ accuracy when performing PA on the sequences separately at each time step, compared to an baseline accuracy of $88.0\%(\pm 1.4\%)$. For the harder 28 classes case, the SkelNet-1400-42 model achieves only a $20.5\%(\pm 0.3\%)$ accuracy when performing PA on the whole sequences and a $42.7\%(\pm 2.1\%)$ accuracy when performing PA on the sequences separately at each time step, compared to an baseline accuracy of $77.9\%(\pm 3.8\%)$.

The orthogonal Procrustes analysis performs even worse than the Procrustes analysis, though. For the 14 classes case, the SkelNet-1400-42 model achieves only a $19.8\%(\pm 0.7\%)$ accuracy when performing orthogonal PA on the whole sequences and a $13.0\%(\pm 9.3\%)$ accuracy when performing orthogonal PA on the sequences separately at each time step, compared to an baseline accuracy of $88.0\%(\pm 1.4\%)$. For the harder 28 classes case, the SkelNet-1400-42 model achieves only a $11.6\%(\pm 3.0\%)$ accuracy when performing orthogonal PA on the whole sequences and a $4.9\%(\pm 0.2\%)$ accuracy when performing orthogonal PA on the sequences separately at each time step, compared to an baseline accuracy of $77.9\%(\pm 3.8\%)$.

Surprisingly, we observe a dramatic drop in the recognition accuracy not only for the "by-time step" PA but also for the "full-sequence" PA, compared to the reference model. The orthogonal Procrustes objective has an analytical solution which eliminates hypothetical optimization issues, but using orthogonal PA does not improve the model accuracies at all.

This suggests that in both cases the temporal information is either lost or at least not salient-enough to present patterns easily distinguishable by the convolutions over time series used by our model.

Data augmentation

Data augmentation is a commonly used technique in deep learning to leverage available data and help neural networks become more robust to noise. Modern neural networks have hundreds of thousands, millions or even billions of parameters. With so many parameters, overfitting is a serious issue for neural networks, especially when they are trained on small datasets.

We perform data augmentation by generating new sequences obtained as modifications of existing sequences, and study the performance of our model. We augment the data with the following transformations:

1. **Scale**: A random scale transformation is applied to the skeleton's channels. The scaling factor is sampled from a uniform distribution in $[0.8, 1.2]$. The scaling factor s can either be the same for all channels: $\forall i \in [\![1,66]\!]\ s_i = s$ with $s \sim \mathscr{U}(0.8, 1.2)$ or be channel-specific to each channel i: $\forall i \in [\![1,66]\!]\ s_i \sim \mathscr{U}(0.8, 1.2)$.

2. **Shift**: A random shift transformation is applied to the skeleton's channels. The shift offset is sampled from a uniform distribution in $[-0.1, +0.1]$. The shift offset o can either be the same for all channels: $\forall i \in [\![1,66]\!]\ o_i = o$ with $o \sim \mathscr{U}(-0.1, +0.1)$ or be channel-specific to each channel i: $\forall i \in [\![1,66]\!]\ o_i \sim \mathscr{U}(-0.1, +0.1)$.

3. **Temporal Interpolation**: New gestures are generated by interpolating the positions of the joints

between consecutive successive time steps. The displacement $\vec{\Delta}$ between two successive frames is calculated and a new position is generated by replacing the original displacement $\vec{\Delta}$ with a scaled displacement $r \cdot \vec{\Delta}$ where the displacement factor r is sampled from a Gaussian distribution centered at 1.0 with a standard deviation of 0.3: $r \sim \mathcal{N}(1.0, 0.3)$. The displacement factor r can either be the same for all channels: $\forall i \in [\![1,66]\!]\ r_i = r$ with $r \sim \mathcal{G}(1.0, 0.3)$ or be channel-specific to each channel i: $\forall i \in [\![1,66]\!]\ r_i \sim \mathcal{G}(1.0, 0.3)$.

4. **Temporal Masking**: A contiguous portion $p = 0.2$ of the gesture is masked. The temporal location of the masked portion is chosen randomly uniformly. To mask the channels, different filling strategies are used: filling with zeros and padding with the channel values before the mask, after the mask, or the mean of these values. Temporal masking can either be the same for all channels or be channel-specific to each channel.

5. **Noise**: A random uniform noise is added to the skeleton's channels. The noise level is sampled from a uniform distribution in $[-0.1, +0.1]$. The noise level ε can either be the same for all channels: $\forall i \in [\![1,66]\!]\ \varepsilon_i = o$ with $\varepsilon \sim \mathcal{U}(-0.1, +0.1)$ or be channel-specific to each channel i: $\forall i \in [\![1,66]\!]\ \varepsilon_i \sim \mathcal{U}(-0.1, +0.1)$.

The model accuracy obtained on the DHG 14/28 dataset (14 classes case and 28 classes case) for each one of the data augmentation strategies mentioned above is detailed in table 4.4.

Model	Original Data	Scale	Shift	Time	Mask	Noise	All
14 classes	88.0 ± 1.4	92.4 ± 1.0	91.6 ± 1.1	91.8 ± 0.7	92.4 ± 0.7	91.9 ± 0.5	92.2 ± 0.8
28 classes	77.9 ± 3.8	83.0 ± 1.4	80.2 ± 6.6	84.2 ± 1.1	81.8 ± 2.5	81.5 ± 2.4	81.5 ± 3.5

Table 4.4 – Influence of data augmentation on the model accuracy (in %)

We observe a gain in accuracy of 4.4 points for the 14 class case and 6.3 points for the 28 class case when we perform data augmentation. This gain is greater than the upper-bound of the original reference model accuracy confidence interval.

When compared to the results from the SkelNet-Hand model evaluated on the same dataset, we conclude that using a lightweight model such as our reference model with data augmentation leads to accuracies comparable to the ones obtained with the SkelNet-Hand model, but with only a fraction of its weights and computational requirements.

With data augmentation, our reference SkelNet model has only $\frac{2404393}{13869871} \approx 17\%$ of the original model weights but achieves roughly the same accuracy (+0.2%) than the original model on the 14 classes case. This result is confirmed on the 28 classes case using data augmentation: the model also represents $\frac{2406199}{13896989} \approx 17\%$ of the original model weights and achieves roughly the same accuracy (-0.3%) than the original model.

This highlights the importance of data augmentation. First, sequences of skeletal hand gesture benefit from data augmentation like more traditional data such as text documents and images. Second, no matter the method used, data augmentation always improves accuracy. Third, among all data augmentation techniques tested, the scale appears to be more useful. Adding noise and temporally interpolating the gestures can also be useful, but combining all the data augmentation techniques does not necessarily improve the final accuracy. All the results of the three techniques cited are not surprising because of the

variability between the subjects that perform the gestures: the performer's hand can move fast or slow, be more or less close to the camera and potentially be detected with some error. As such new examples generated with a method based on these observations are more likely to look like real gestures while still providing data variability to the network. Since skeletal data is sparse, using data augmentation likely improves not only the accuracy but also the robustness of the model.

Sequence Length

We compared the influence of the length of the input sequences on the model performances by training a model for each one of 70, 80, ..., 130 input time steps configuration both for 14 and 28 cases.

The model accuracy obtained on the DHG 14/28 dataset (14 classes case and 28 classes case) for each one of the input data durations mentioned above is detailed in table 4.5.

Model	$T = 70$	$T = 80$	$T = 90$	$T = 100$	$T = 110$	$T = 120$	$T = 130$
14 classes	88.8 ± 1.0	89.8 ± 0.5	88.9 ± 1.5	88.4 ± 2.0	87.7 ± 1.4	88.7 ± 0.6	84.8 ± 3.1
28 classes	74.2 ± 4.8	73.3 ± 6.1	76.0 ± 4.3	74.9 ± 4.7	75.7 ± 5.6	78.3 ± 1.5	75.7 ± 0.8

Table 4.5 – Influence of the input data length on the model accuracy (in %)

For the reference model architecture on the 14 classes case, the accuracy varies between -3.6 and +1.4 points, compared to a reference accuracy computed for the 100 time steps case.

For the reference model architecture on the 28 classes case, the accuracy varies between -1.6 and +3.4 points, compared to a reference accuracy computed for the 100 time steps case.

However, the sequence length of the best performing models are not the same depending the number of classes: $T = 80$ for the 14 classes case and $T = 120$ for the 28 classes case.

As such, we chose $T = 100$ since this sequence length appears to be a good compromise between the model performance on 14 classes, on 28 classes and on the model size which is dependent on the input size.

As a result, the model is always trained on inputs that have a 100 time steps length.

Position/Rotation representations

We compare the influence of recognizing gestures and actions based on the joints' positions or the joints' orientations. We use the NTU RGB+D dataset for the experiments, as the DHG 14/28 dataset does not provide the joints' orientations.

The NTU RGB+D dataset requires a deeper SkelNet neural network compared to the ones we use on the DHG 14/28 dataset. Although being deeper, that SkelNet model has less parameters than the ones used on DHG 14/28. Except the depth of the model used, all of these models follow the same (SkelNet) architecture. Its exact architecture is presented in the last section of this chapter.

The SkelNet model taking the joints' positions as input data achieves a $61.5\%(\pm 0.42\%)$ accuracy, while it only achieves a $51.53\%(\pm 0.59\%)$ accuracy when the input data is the joints' orientations.

We observe the best representation is the position-based one. To the best of our knowledge, there is no consensus in the literature about the best representation for gesture recognition between position

and orientation of the joints. One hypothesis we propose to explain the high difference in performance between our position-based and orientation-based models lies in the estimation method used by the Kinect cameras that were used to create this dataset: as estimating correct positions is easier than estimating rotations, the position data may be less noisy than the orientation data in the dataset.

Early recognition

One drawback of using convolutional-based approaches for gesture recognition lies in the fixed input size used by the convolutional models. As one can use sliding windows over time to perform by-window gesture classification, this drawback is mostly an issue at the beginning of gestures, when the gesture acquisition is not yet finished.

We propose an extremely simple solution to that specific issue, by stretching the portion of the gesture already performed to the fixed-input size required by the model.

Instead of classifying a full gesture, we analyze the recognition performance of our model on an ongoing gesture, depending on the portion of the gesture already performed, as depicted in figure 4.13.

As expected, the performances are low when a very small fraction of the gesture is performed. However, the performances soar as the portion of the gesture performed comes close to 1.

This results suggests that using temporal resampling alongside with a convolutional network approach on non-finished gestures is sufficient to obtain satisfactory accuracies for early gesture recognition. When 80% of the gesture has already been performed, the recognition accuracy is as high as 79.2% for the 14 classes case and 68.1% for the harder 28 classes case.

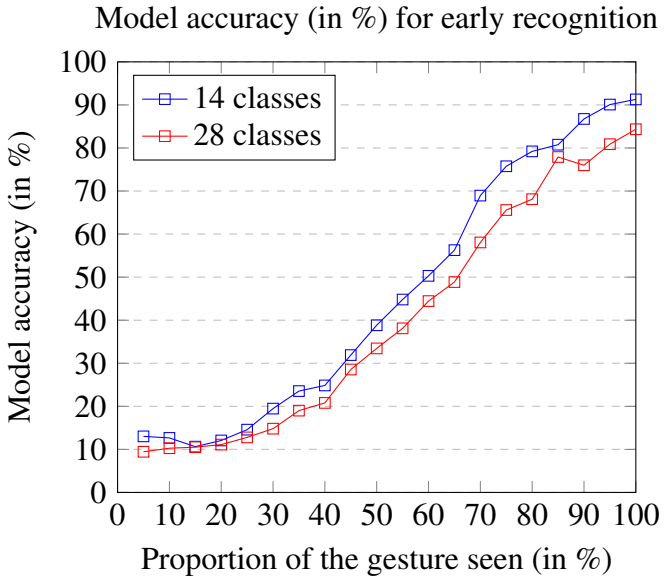


Figure 4.13 – Evolution of the recognition performances as gestures are performed

4.4.3 Module Ablation Study

In this section we analyze the incidence of each part of the neural network, by removing individual modules of the model, all other factors being equal. In this section, the temporal feature extractor modules of the reference model are not shared, as in the SkelNet-Hand model.

Model	Accuracy (14 classes)	Delta	Accuracy (28 classes)	Delta
Reference model	91.28		84.35	
Without Residual branch	90.32	-1.05	79.93	-5.24
Without High-resolution branch	90.80	-0.53	78.97	-6.38
Without Low-resolution branch	90.08	-1.31	80.17	-4.96

Table 4.6 – Influence of branch ablation on the model accuracy (in %)

We highlight the importance of using three parallel branches for the 28 gesture classes case in table 4.6 where performances significantly drop if one of the two high- or low- resolution branches are removed.

4.4.4 Neural Network Design Choices

We now focus on the influence of varying hyperparameters for the different building blocks of the model.

Pooling Method

A comparison of average pooling and max pooling shows that average pooling slightly outperforms max pooling by +0.88% in accuracy for the 14 classes classification problem. For gesture classification problems where no complex semantic information is present, average pooling likely acts as a regularizer, compressing channels in the time domain. As gestures are smoother and more regular than lots of other time series data because of the physical constraints inherent to gesture, and especially when compared to sequences of arbitrary word vectors in the natural language processing domain, average pooling could arguably result in loosing less useful information than max pooling.

Preprocessing module

The preprocessing module is trained jointly with the convolutional network. We do not apply a normalization step before this preprocessing module, as the DHG14/28 data has already nearly normalized values.

The influence of the size of the output of the preprocessing module is illustrated in figure 4.14. We conclude of this experiment that the size of the preprocessing module has no significant impact on the accuracy for our gesture classification task. The model can gain up to 1.0% in accuracy when choosing the appropriate number of output channels (e.g. $C_{out} = 132$ or $C_{out} = 44$) compared to the baseline SkelNet-1024-128 model, however no trend is easily distinguishable from the results.

The 1.0% gain in accuracy observed when $C_{out} = 44$ (while there are 22 joints) may suggest that most of the gestures may be distinguishable with a simple 2D projection from the 3D data.

As the preprocessing module has little impact on the accuracy, it is not applied by default for the hyper-parameter search, and the input sequences are directly processed by the convolutional module.

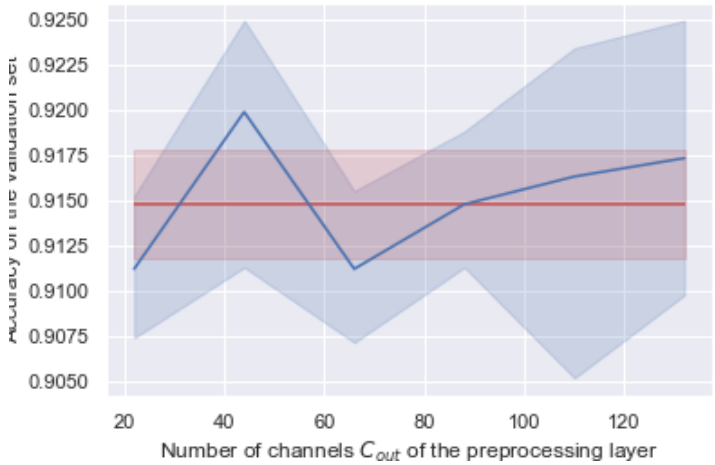


Figure 4.14 – Accuracy obtained on the 14 classes case, with various sizes of the linear (over the "channels" axis) preprocessing module (blue curve). The red line is the SkeltNet-1024-128 reference architecture, without a preprocessing module.

Convolutional module

The network's convolutional module extracts temporal features from the gesture channels. In this section we evaluate the use of either standard convolutions or temporal convolution networks (TCN) introduced by (Lea et al., 2017) for the temporal feature extraction. TCNs are 1D convolutional networks where standard convolutions are replaced by *causal* convolutions, i.e. convolutions where an output at time t is convolved only with elements from time t and earlier in the previous layer. More precisely, a TCN convolutional layer is a residual layer whose processing block is made of two dilated causal convolutions with ReLU activations, with both weight normalization (Salimans et al., 2016) and dropout (N. Srivastava et al., 2014) regularization methods.

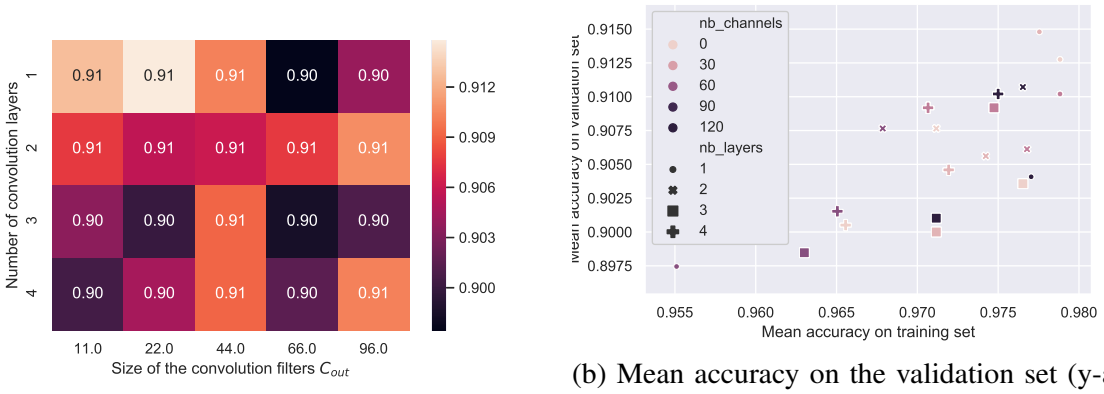
We first evaluate performances obtained with standard convolution. By using a grid search for $n \in \{1,2,3,4\}$ convolutional layers and $C_{conv} \in \{11,22,44,66,96\}$ output channels per layer, we show that shallow and small neural networks perform better (figure 4.15).

In particular, in the case of small convolutions ($C_{conv} \in \{11,22\}$) the validation accuracy decreases with the number of layers, while this number has little impact for larger convolution kernels $C_{conv} \geq 44$. The best overall score for both validation and training sets is reached for $C_{conv} = 22$ and $n = 1$. This suggests that shallow and small neural networks generalize better but also converge more easily on a medium-sized dataset like the DHG 14/28 dataset. We though observe that deeper architectures may be more adapted or even required for larger datasets that includes more classes, like the NTU RGB+D dataset for instance.

We now present the results obtained with the TCN convolutional layers.

We explore 3 essential hyper-parameters :

- The depth d of the network, i.e. the number of successive convolution layers
- The number of convolution channels per layer C_{conv}
- The filter size k of the convolutions



(a) Mean accuracy of different convolution architectures on the validation set (using 3-fold cross-validation)

(b) Mean accuracy on the validation set (y-axis) and the training set (x-axis). Models in the bottom right part of the graph are more likely to overfit on the training set.

Figure 4.15 – Influence of the number and the size of convolution layers on accuracy.

Experimental results (figure 4.16) suggest that :

- TCNs require more depth than our classical CNN to be accurate: at least 5 layers (figure 4.16a)
- Having large networks with a high $C_{conv}(\geq 200)$ leads to a good performance (figure 4.16b)
- A minimum kernel size is required to perform well, presumably for the model to have a long enough memory. We see that, with $d = 5$ layers, kernel size $k = 2$ (giving a receptive field of $k^d = 2^5 = 32$ time steps to the network) is not sufficient, whereas $k = 3$ (giving a receptive field of $k^d = 3^5 = 243$) is enough (figure 4.16c)
- The SkelNet-1024-128 with standard convolutions performs better than all the equivalent TCNs configurations tested

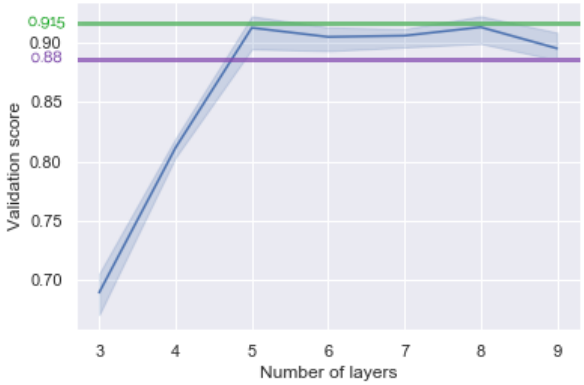
In the TCN case, the classification module's input is only the last time step output $(y^{(c)}(t = L))_{c\in[0,C_{conv}-1]}$, of size C_{conv} (where C_{conv} is the number of channels outputs of the last convolution layer), whereas the size is (C_{conv}, L_{conv}) for our regular CNN model. Even though we choose a larger C_{conv} than in the equivalent regular CNN architecture, the output of the convolution module is much smaller. Therefore we use a MLP with only 1 hidden layer (of size 128) for classification.

The SkelNet-1024-128 with standard convolutions has $\frac{4662123}{2404393} \approx 2$ times less parameters than its counterpart with temporal convolutions, while the SkelNet NTU with standard convolutions introduced in the section 4.4.6 has even $\frac{6384192}{1545984} \approx 4$ times less parameters than its counterpart with temporal convolutions.

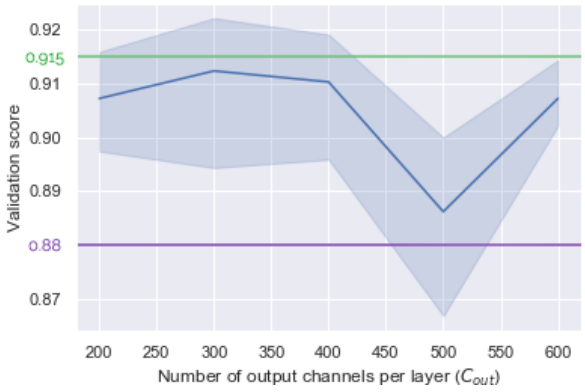
The SkelNet models that use TCN convolutional layers have considerably more parameters than the SkelNet models that use a standard convolution approach. They are also much slower to train (about 20 times slower than a small and shallow CNN architecture).

Dropout

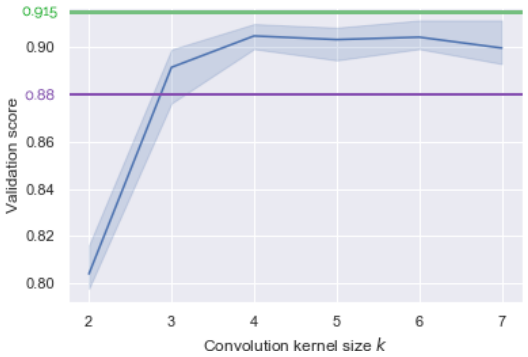
We study in figure 4.17 the impact of dropout rate on accuracy, for three variations of a SkelNet-1936-128 model.



(a) Influence of the number of layers on the model accuracy, on a model with $C_{conv} = 300$ channels per convolution layer



(b) Influence of the number of output channels per layer on the model accuracy, on a model with $d = 5$ layers



(c) Influence of the kernel size (previously set to 3) on the model accuracy, on a model with 5 layers and $C_{conv} = 300$ output channels

Figure 4.16 – Experiments on TCN architectures (14 classes case). For comparison, the accuracies of both the SkelNet-1024-128 and the SkelNet-1400-42 models **with standard convolutions** are displayed, in green and in violet respectively.

The first variation (in red in figure 4.17) consists in a model with 66 independent processing branches. The second variation (in green in figure 4.17) consists in a model with only 3 processing branches: 1 that processes all the x channels, 1 that processes all the y channels and 1 that processes all the z channels. Finally the third variation (in blue in figure 4.17) consists in a model with a single processing branch shared for all channels regardless of the joint or the joint component.

In figure 4.17 (a) the experiment is performed for the 14 classes case of the DHG 14/28 dataset, whereas in figure 4.17 (a) the experiment is performed for the 28 classes case of the DHG 14/28 dataset.

The results of the experiment show an almost constant final model accuracy with a wide confidence margin, regardless of the dropout rate: noisy random weights initialization at the beginning of the training have more impact on the final accuracy than the dropout rate *per se*.

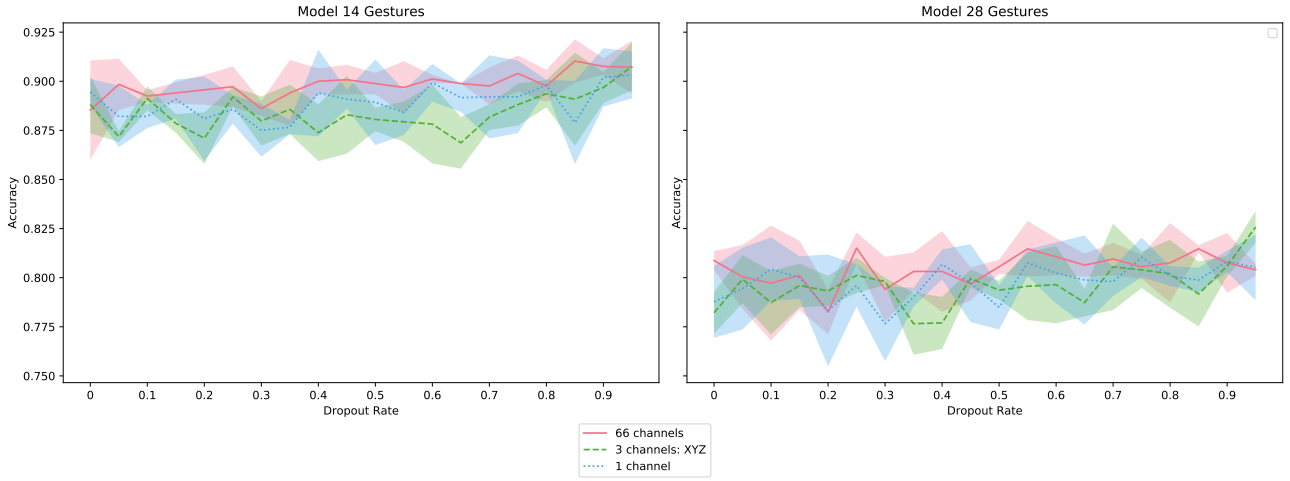


Figure 4.17 – Model accuracy for different dropout rates.

4.4.5 Weight Sharing Study

Sharing Weights at the Input Level

To reduce the total parameters count of the learned neural network and impose regularities across channels, one could think of feeding several channels to the same convolutional layers, using grouped convolutions. Channels are split into *groups* different groups of equal sizes.

All $\frac{66}{groups}$ input channels in the same group share the same set of filters. For instance, when $groups = 1$ all 66 channels are convolved with the same filters, whereas $groups = 66$ means that each input channel is processed with its own filters.

Our network working with 66 parallel extractors, using 3 convolutional layers of size $C_{out,conv} = (8,8,4)$ for 1 input sequence, is equivalent to a neural network with 3 layers

$$[(C_{out} = 66 \times 8, groups = 66), (C_{out} = 66 \times 8, groups = 66), (C_{out} = 66 \times 4, groups = 66)]$$

working on all 66 input sequences. We observe that having this many channels and this many layers is not necessary, at least on the DHG 14/28 dataset.

However, we still want to analyze how this *groups* parameter affects our model, and if processing some sequences separately improves the performance. Results are presented in figure 4.18.

In figure 4.18 (a), a SkelNet model with 1 convolutional layer operating on the 66 input channels is trained for gesture recognition on the 14 classes case of the DHG14/28 dataset. The impact of the *groups* parameter on the model accuracy is evaluated. The results are plotted in figure 4.18 (a), as well as the associated confidence intervals. Since *groups* must divide the input channels' count, only *groups* values up to 66 that divide 66 are considered: namely 1, 2, 3, 6, 11, 22, 33 and 66. We observe that the best model accuracies are obtained with $groups = 1$, $groups = 2$ or $groups = 66$. Since the model's accuracy's confidence interval is wider (and therefore the model's accuracy more uncertain) for $groups = 2$ than for $groups = 1$ and since the model with $groups = 1$ has way less parameters than the model with $groups = 66$, we finally choose to use $groups = 1$ in our baseline model.

In figure 4.18 (b), two SkelNet models with 96 output channels in both cases, are trained for gesture recognition on the same 14 classes case of the DHG14/28 dataset. The models are identical, except for the depth of the temporal features extractor modules: one model (results in plain red, with a confidence interval, at the top in figure 4.18 (b)) uses 2 convolutional layers whereas the other model (results in plain blue, with a confidence interval, at the top in figure 4.18 (b)) uses 4 convolutional layers. For each one of the models, the impact of the *groups* parameter on the model accuracy is evaluated. Since *groups* must divide both 66 and 96, only the following *groups* values are considered: 1, 2, 3 and 6. We observe that the models' accuracies are very close to each other with consideration to the accuracies' confidence interval. This conjecture from this observation that the SkelNet model architecture does not benefit from grouped convolutions for arbitrary output channel sizes: in other words, the model's architecture does not benefit from grouped convolutions when the (input channels') groups are arbitrary themselves.

Sharing Weights at the Resolution Level

In this section, the reference model is as the SkelNet-Hand model, with non-shared temporal feature extractor modules.

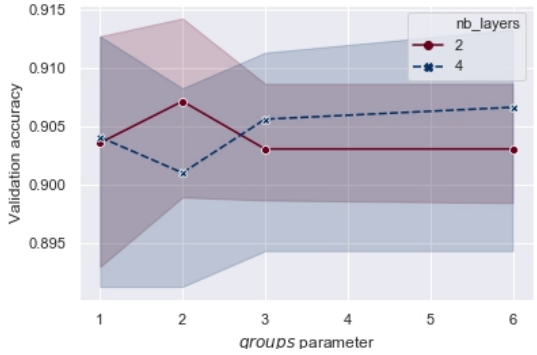
To reduce the total parameters count of the learned neural network ($\sim$13.8M parameters) and impose regularities across channels, one could think of re-using the same temporal feature extractor module entirely across all channels.

Since the temporal feature extractor module has two convolutional branches, it is also possible to share only one of the two branches across channels and still have a channel-specific one for the other.

The model accuracy obtained on the DHG 14/28 dataset (14 classes case and 28 classes case) when sharing all high-resolution branches across channels or all low-resolution branches across channels is detailed in table 4.7.

This result highlights the importance of processing the two branches adaptively, since the accuracy drops when one of the branches is shared amongst channels.

(a) Model accuracy on the 14 classes case for the 1 convolutional layer SkelNet model, depending on the *groups* parameter. The values *groups* can take are limited because it must divide 66 (since there are 66 input channels).



(b) Model accuracy on the 14 classes case for the 2 convolutional layer model (in red) and for the 4 convolutional layer model (in blue), depending on the *groups* parameter. The values *groups* can take are limited because it must divide 66 (input channels' count) and 96 (output channels' count).

Figure 4.18 – Influence of the *groups* parameter on the model accuracy. Each one of the C_{out} convolutions sees only $\frac{66}{groups}$ input sequences. $groups = 1$ is a classical convolution layer, $groups = 66$ means that input sequences are processed independently.
The average score on the 3 splits of the cross-validation is taken, and the blue bars represent the 95% confidence interval.

Model	Accuracy (14 classes)	Delta	Accuracy (28 classes)	Delta
Reference model	91.28		84.35	
Sharing all high-res. branches	89.73	-1.55	81.48	-2.87
Sharing all low-res. branches	89.8	-1.48	81.96	-2.39

Table 4.7 – Influence of sharing branches on the model accuracy (in %)

Sharing Weights at the Abstraction Level

In this section, the reference model is as the SkelNet-Hand model, with non-shared temporal feature extractor modules.

Due to their sequential nature, deep convolutional neural networks are usually believed to create hierarchical representations, where the first convolutional layers extract low-level patterns from the input signal, whereas the last convolutional layers extract more abstract patterns related to the final classification task.

Regardless of its resolution, each of the two convolutional branches of the temporal feature extractor module have three convolutional layers.

To reduce the total parameters count of the learned neural network and impose regularities across channels, one could think of re-using the same low-level convolutions in all feature extractor modules.

The model accuracy obtained on the DHG 14/28 dataset (14 classes case and 28 classes case) when re-using the same low-level convolutions in high-resolution branches only, in low-resolution branches only, or in both branches, is detailed in table 4.8.

One can also think of sharing weights both at a resolution-level (i.e. the branches) and an abstraction-

Model	Accuracy (14 classes)	Delta	Accuracy (28 classes)	Delta
Reference model	91.28		84.35	
Sharing first conv. in high-res. branch	90.08	-1.20	80.65	-3.70
Sharing first conv. in low-res. branch	88.77	-2.51	83.15	-1.20
Sharing first conv. in both branches	89.37	-1.91	82.92	-1.43

Table 4.8 – Influence of sharing first convolutional layers on the model accuracy (in %)

level (i.e. the first convolutions) across channels. The model accuracy obtained on the DHG 14/28 dataset (14 classes case and 28 classes case) when sharing weights both at a branches-resolution-level and an abstraction-level (i.e. the first convolutions) across channels is detailed in table 4.9.

Model	Accuracy (14 classes)	Delta	Accuracy (28 classes)	Delta
Reference model	91.28		84.35	
Sharing first conv. in low-res. branch and sharing all high-res. branches	89.96	-1.32	82.92	-1.43
Sharing first conv. in high-res. branch and sharing all low-res. branches	89.61	-1.67	81.84	-2.51

Table 4.9 – Influence of sharing first convolutional layers and branches on the model accuracy (in %)

While sharing the weights of both branches decreases the model accuracy more, sharing either the low-resolution branch or the high-resolution branch (and not the other one) is feasible and only reduces the accuracy of up to $\sim$1.7% for the 14 classes case and up to $\sim$2.5% for the 28 classes case.

Sharing only the first convolution of a branch does not improve the overall accuracy, which may indicate that either the network depth is not big enough to benefit from such low-level processing sharing, or that the dataset size is too small to benefit from this sharing. The results obtained on the 28 classes case suggests that some channels -likely some joints- are more useful than others to predict the class, especially at a high-resolution level.

Overall, sharing weights at the resolution or at the abstraction level slightly reduces the parameters count, but at the cost of up to $\sim$2.5% for the 14 classes case and up to $\sim$3.7% for the 28 classes case.

4.4.6 Applicability of the approach to other databases

To evaluate whether our approach is applicable to other types of skeleton-based motions, we train our model on other skeleton-based datasets.

Full body

The first database, we use is the NTU RGB+D dataset introduced in (Shahroudy et al., 2016) which is a human-body skeletal action recognition database. After performing experiments, we observe that our SkelNet model requires more convolutional layers than in the hand gesture case. For that dataset, the SkelNet model is made of 4 successive convolutional layers with $C_{out} = 25, groups = 1$, followed by 3

fully-connected layers $N_{out,1} = 1024, N_{out,2} = 1024, N_{out,3} = 128$. On the NTU RGB+D dataset, that SkelNet model yields to an accuracy of 61.5% $\pm$ 0.42 (see table 4.10).

The SkelNet model is therefore better performing than Deep LSTM models from (Shahroudy et al., 2016). This highlights that convolution-based models such as SkelNet can perform better than recurrent-based models, while being faster to train and easier to understand. However, the SkelNet model is significantly outperformed by other approaches that include domain-knowledge, attention mechanisms and/or graph-based deep learning models. Integrating these elements into the SkelNet architecture might therefore significantly improve our results. It should also be noted that our model is very lightweight, with only $\sim$1.5M parameters, paving the way for real-time gesture recognition on embedded devices like smartphones and robots.

Face Landmarks

We also apply our model on the RAVDESS dataset which is a facial emotion recognition video dataset by first extracting 68 face landmarks from the videos as described in the section 4.3.3. With all the frames, we get times series of the face landmarks positions. We use the same shallow SkelNet configuration as for the hand gesture task, the input being landmarks instead of joints. We finally train the model on the emotion classification task, and achieve a 91.2% recognition accuracy. This is significantly above the 79.74% accuracy result reported by (Z. He et al., 2019) with their CNN-based method on the same RAVDESS dataset (visual modality). Furthermore, our model is lightweight, especially compared to approaches based on vision-only, and suited to facial emotion recognition.

Model	(Cross-Subject) Accuracy
(Evangelidis et al., 2014)	38.6
(Vemulapalli et al., 2014)	50.1
(Du et al., 2015)	59.1
(Hu et al., 2015)	60.2
(Shahroudy et al., 2016) (Deep LSTM)	60.7
(Shahroudy et al., 2016) (Part-aware LSTM)	62.9
(J. Liu et al., 2016)	69.2
(H. Wang et al., 2017)	71.3
(Song et al., 2017)	73.4
(Fan et al., 2018)	73.8
(J. Liu et al., 2017) (Direct GCA-LSTM)	74.3
(I. Lee et al., 2017)	74.6
(Tas et al., 2018)	75.3
(Ke et al., 2017a)	75.9
(M. Liu et al., 2017)	76.0
(J. Liu et al., 2017) (Stepwise GCA-LSTM)	76.1
(Pichao Wang et al., 2016)	76.3
(J. Weng et al., 2018)	76.8
(P. Zhang et al., 2017)	79.4
(H. Wang et al., 2018)	79.5
(Ke et al., 2017b)	79.6
(M. Liu et al., 2017)	80.0
(S. Weng et al., 2018)	81.1
(S. Yan et al., 2018)	81.5
(Xie et al., 2018)	82.7
(Y. Tang et al., 2018)	83.5
(Si et al., 2018)	84.8
(Maghoumi et al., 2018)	84.9
SkelNet NTU ($\sim$ 1.5M parameters only)	**61.5**($\pm$**0.4**)

Table 4.10 – Results on NTU RGB+D Dataset (in %)

4.5 Model visualizations

Richard Feynman, who received the Nobel Prize in Physics in 1965, once stated in a famous sentence: "What I cannot create, I do not understand". Feynman thought that understanding something was not just about working through advanced mathematics, but to have the ability to reduce it to it's fundamental underlying concepts (first principles) and have a firm grasp on each of them.

As scientists and as humans, we want to be able to answer "why?" questions, either to understand the

world or to act on it in a way we desire. Before trusting a model and allowing a widespread adoption of that model, people want to know the model performs "well", e.g. either in terms of statistical accuracy, or in terms of their inner mechanism: their mechanism should be appropriate, fair, and open for audit and control.

While it is easy to get a mental intuition about how SkelNet models may process information through temporal filtering of the joints, one should cross-check if this intuition is likely a correct one or not, and try to understand the reasons behind a model giving one specific outcome rather than another one.

Interpreting a neural network model is still a significant challenge in machine learning today. Methods that attempt to interpret neural networks usually fall into: game theoretic approaches, probabilistic and statistical approaches, clustering approaches, visualizations-related approaches, or a combination of them. Among them, methods that attribute the prediction of a deep network to its input features receive more attention, as they mimick the way we try to understand new unknown phenomena as humans.

However, the credit assignment problem, which refers to the question of determining how much 'credit' (or 'blame') a given neuron should get for a given outcome, is still also an open and active problem in research, with direct implications in reinforcement learning (e.g. what actions should be credited for the reward, given that all actions previous to a time step affect the environment at that time step?) or in the time-series supervised learning (e.g. for efficient backpropagation through time in recurrent networks). Post-hoc interpretability is one family of methods that seek to "explain" the prediction without considering the details of black-box model's hidden mechanisms.

Attribution Methods

To study the problem of attributing the prediction of a deep SkelNet model to its input features, we choose to use two state-of-the-art post-hoc approaches attribution methods and visualize the result. These two post-hoc approaches: Integrated Gradients (Sundararajan et al., 2017) and DeepLift (Shrikumar et al., 2017).

Both of these methods extend the idea of examining the gradient of an output with respect to its input, the latter not being viable as-is with non-linear deep learning models (e.g. negative values going through a ReLU activation always lead to a zero-gradient, regardless of the input). More precisely, they are based on the study of how much an output changes from a baseline as the input changed from a baseline. Using a baseline is needed both for philosophical reasons -since understanding is closely related to the essence of objects: properties kept invariant regardless of external change or interactions- and for practical reasons, mostly due to the non-linearities of deep learning models.

Integrated Gradients calculate the integral of the gradients between a baseline tensor $\mathbf{x}_{baseline}$ and a tensor of interest $\mathbf{x}$ (equation 4.10).

$$IG(\mathbf{x}, \mathbf{x}^{baseline}) = (\mathbf{x} - \mathbf{x}^{baseline}) \int_0^1 \nabla F \left(\alpha \mathbf{x} + (1-\alpha)\mathbf{x}^{baseline} \right) \mathrm{d}\alpha \tag{4.10}$$

Rather than focusing on the gradient (how the output changes at the input point), DeepLIFT explains the difference in output from a reference' (baseline) output in terms of the difference of the input from some 'reference' (baseline) input. While the underlying equations are more complex and separate positive and negative contributions, the idea is still close to the Integrated Gradients methods, replacing

gradients of a layer by a slope (equation 4.11) called multiplier[6], leading to feature importance (equation 4.12).

$$\mathbf{m} = \frac{\mathbf{y} - \mathbf{y}^{baseline}}{\mathbf{x} - \mathbf{x}^{baseline}} = \frac{\Delta \mathbf{y}}{\Delta \mathbf{x}} \tag{4.11}$$

$$feature_i^{DL} = (\mathbf{x} - \mathbf{x}_i^{baseline}) \frac{\Delta \mathbf{Y}_i}{\Delta \mathbf{x}_i} \tag{4.12}$$

While interpretation of neural networks is fragile (Ghorbani et al., 2019), especially when it comes to adversarial perturbations, such methods still reveal themselves to be helpful, in order to get an intuition about how and why a model makes its predictions, especially when combined with a visual representation of the results.

Introduction to the visualization approach chosen

In this subsection, we explain the visualization approach chosen for all the figures of this section, namely for figures 4.19 (a) and (b), for figures 4.20 (a) and (b), for figures 4.21 and for figures 4.22 (a) and (b). All the figures are later explained in more detail in the next sections.

Each figure, e.g. figure 4.19 (a), features $14 \times 2 = 28$ plots. Each one of the 28 plots represents a sequence, using the standard 2D format introduced earlier at the beginning of the chapter (see figure 4.3 (c) for a short description of the 2D format). The vertical axis of each plot represents time (from time step 0 at the top, to time step 100 at the bottom). The horizontal axis of each plot represents the $C = 22 \times 3 = 66$ input channels (in the following order: $x_1, x_2, \ldots, x_J, y_1, y_2, \ldots, y_J, z_1, z_2, \ldots, z_J$, with $J = 22$).

The first and third rows of each figure, i.e. 14 plots in total, represent prototypes of each one of the 14 classes of the DHG14/28 classes (14 classes case). To construct the prototype of a given gesture class, all gestures of that class are averaged, leading to a mean tensor of the same shape: (T, C) where $T = 100$ and $C = 66$. The colormap used for the 14 plots is a blue-to-yellow colormap: blue indicates low values whereas yellow indicates high values.

The second and fourth rows of each figure, i.e. 14 plots in total too, represent models attributions for the prototype sequences illustrated immediately above them. The colormap used for the 14 plots is a red-to-green colormap: red indicates negative values whereas green indicates positive values.

As an example, in figure 4.19 (a), the second plot of the third row represents the prototype (i.e. mean sequence) of the class #9 ("Swipe Up"): the X and Z channels blocks (on the left and on the right, in green-yellow colors) exhibit approximately constant values, while the Y channels block (on the middle, in blue) exhibit Y values that evolve over time (from low values in dark blue at the beginning of the sequence (top) to higher values in lighter blue later (middle and bottom)).

In the same figure, the second plot of the fourth row (i.e. just below the plot of the "Swipe Up" prototype) represents the models' attributions for that "Swipe Up" prototype. The choices of the exact attribution method and baseline are discussed in the next sections. In this example, the attribution method is the DeepLift method and the baseline is a gesture prototype defined as the mean sequence

[6]Multipliers respect a chain rule like gradients, and partial-slopes can be computed in a manner that is analog to the partial-derivatives computation.

of all gestures, regardless of their class. Reading the attribution plot, we can observe how or why the DeepLift attribution method considers that a Swipe Up gesture (or, more precisely, the Swipe Up prototype plotted right above) is effectively classified as a "Swipe Up" gesture by our SkelNet model. Our model considers the evolution of Y channels between $t = 40$ and $t = 70$ to be very informative of a "Swipe Up" gesture, whereas the evolution of X and Z channels are not considered as being very informative of a "Swipe Up" gesture at any moment. These attributions seem credible with regards to the prototype plotted above and tend to support our convolutions-over-time model architecture .

Still as an example, in the same figure 4.19 (a), the attributions related to the class #8 ("Swipe Left") are also informative. While the conclusion of the observed attributions plot is similar (the model seems to consider the evolution of X channels between $t = 40$ and $t = 70$ to be very informative of a "Swipe Left" gesture, whereas the evolution of Y and Z channels are not considered as being very informative of a "Swipe Left" gesture at any moment), the attributions plot reveals that the model tends to rely not on all X channels but on a very limited set of X channels (about 3 or 4 channels).

In the next subsections, we experiment different visualizations choices and try to interpret the resulting visualizations.

Choice of the attribution method

We experimentally investigate the differences between the two attribution methods (Integrated Gradients and DeepLift) for SkelNet models in figure 4.19. More precisely, the attributions obtained with the Integrated Gradients approach are presented in figure 4.19 (a) whereas the attributions obtained with the DeepLift approach are presented in figure 4.19 (b).

On figure 4.19, the two attribution methods lead to similar visualizations: most gesture prototypes (i.e. mean gestures for a given class) have close attributions by the two methods, while for some of them (e.g. 5 - Rotation Clockwise and 6 - Rotation Counter Clockwise) one of the two methods appears to perform better than the other (e.g. Integrated Gradients arguably perform much better than DeepLift for the two aforementioned classes on the visualization).

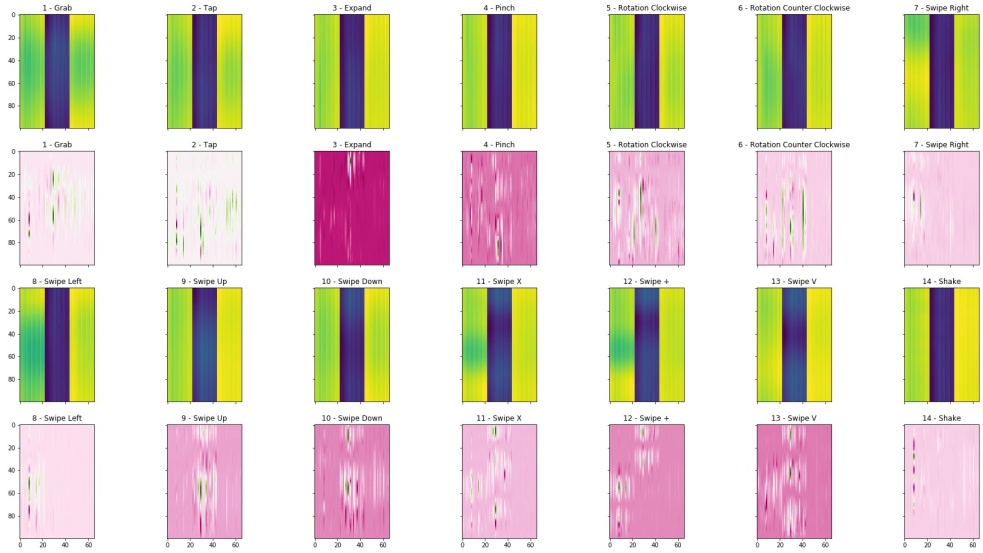
After observing differences between the two attributions methods for several other SkelNet architectures and different hyper-parameters, we qualitatively conclude that neither one nor the other is consistently more informative than the other one and choose to only use Integrated Gradients for further visualizations.

Choice of the attribution baseline

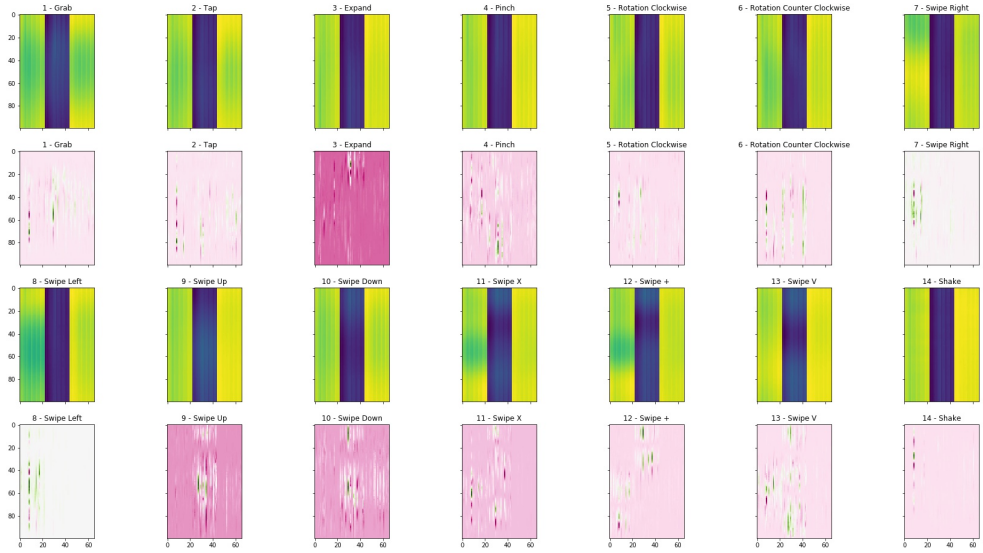
Two natural baselines sequences are the zeros sequence: $\mathbf{x}^{baseline}_{zeros} = \mathbf{0}$ and the $\mathbf{x}^{baseline}_{mean\,sequence} = \frac{1}{N}\sum_{1\in[[1,N]]}\mathbf{x}_i$ mean of all sequences in the DHG 14/28 dataset, regardless of the class of the gestures.

We compare the two baselines in figure 4.20.

In figure 4.20 (a), i.e. in the zero-baseline case, the attributions plots can almost be directly obtained by thresholding the prototypes sequences plotted right above them. In figure 4.20 (b), i.e. in the mean-baseline case, the attributions plots appear to be more related to motions that are specific to individual gesture prototypes.



(a) Visualization of by-class mean gesture of each class (DHG 14 classes) and the associated attributions to the respective class. Red indicate negative values, while green indicate positive attribution. Horizontal axis: channels. Vertical axis: time. Attribution method: Integrated Gradients.

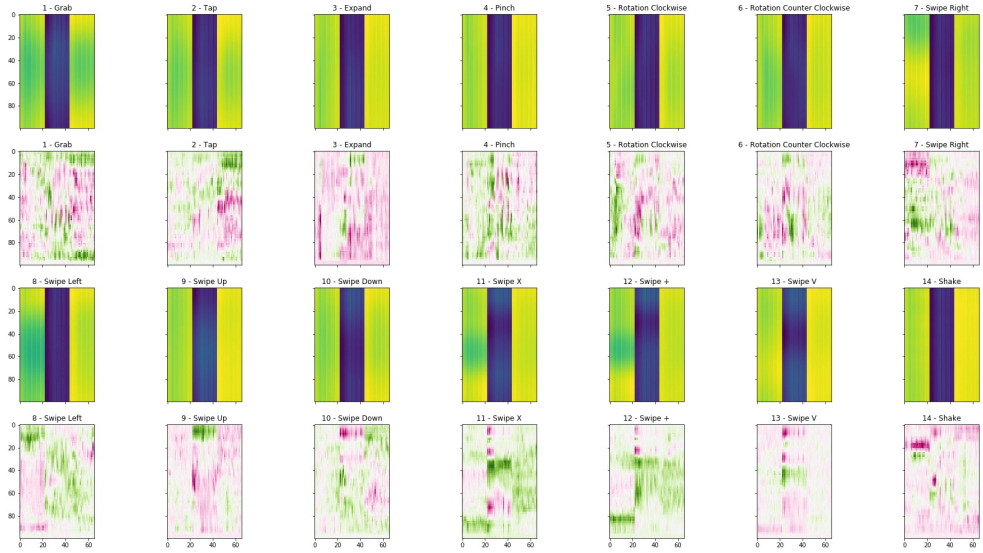


(b) Visualization of by-class mean gesture of each class (DHG 14 classes) and the associated attributions to the respective class. Red indicate negative values, while green indicate positive attribution. Horizontal axis: channels. Vertical axis: time. Attribution method: DeepLift.

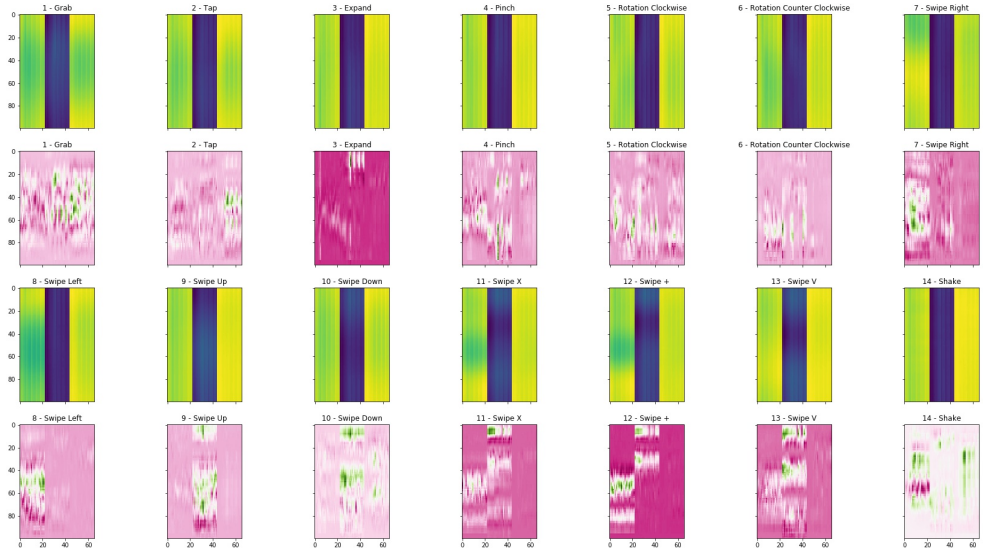
Figure 4.19 – Comparison and visualization of two attribution methods: Integrated Gradients and DeepLift.

As such, the zero-baseline attributions focuses on the movement *per se*, while the mean-baseline attributions focus on the gesture specificity compared to other gestures.

Since explaining the models classification is the goal of this section, we choose to use mean-baseline



(a) Integrated Gradients attributions for SkelNet model with a zero-baseline (no motion)



(b) Integrated Gradients attributions for SkelNet model with a mean-baseline (mean motion)

Figure 4.20 – Comparison and visualization attributions depending on the baseline.

for further visualizations.

Visualizing branch-sharing (1 / 3:XYZ / 66) in SkelNet models

We now focus on the model by itself and compare three variations of the SkelNet architecture. The variations we study are the three variations we already introduced in section 4.4.4.

The first variation consists in a model with 66 independent processing branches. The second variation consists in a model with only 3 processing branches: 1 that processes all the x channels, 1 that

processes all the y channels and 1 that processes all the z channels. Finally the third variation consists in a model with a single processing branch shared for all channels regardless of the joint or the joint component.

In figure 4.21, we present the attributions we obtain for each one of the three models. For the sake of clarity, we refer to the first variation as the $C = 66$ channels model, the second variation as the $C = 3$ (XYZ) channels model, and the third one as the $C = 1$ channels model in the legend of the figure 4.21.

In figure 4.21, the 1-shared-branch model surprisingly appears to not only correctly focus on temporal variations of the channels' amplitude, but also have comparable attributions for all channels in a single x-block (resp. y-block and z-block) of input channels. One shortcoming of that model may lie in the fact that the temporal features tend to group together for several joints or channels. This suggests that this model may be arguably more adapted for global motion of the hand than for fine finger-level motions.

The 3:XYZ-shared-branch model also appears to correctly model temporal variations of the channels' amplitude. However, this model does not seem to present any clear advantage over the 1-shared-branch model, even with the introduction of human-knowledge with specific branches for the X, Y and Z axes in order to reflect the importance of these axes in the 14/28 gesture classes.

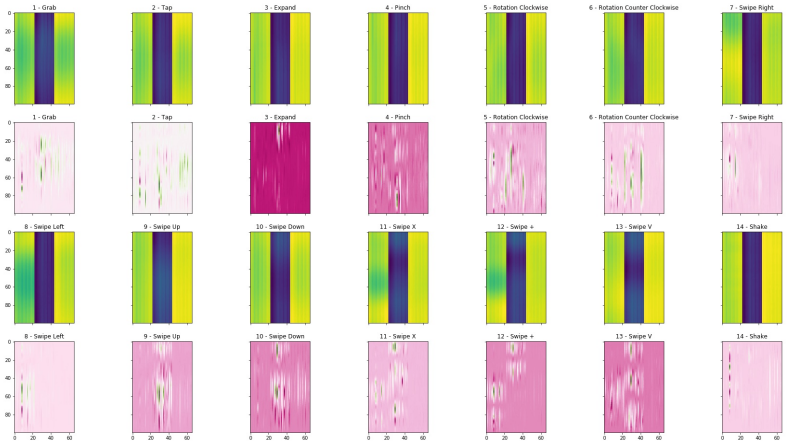
The 66 independent branches model also appears to correctly model temporal variations of the channels' amplitude. Contrary to the former two other models, the attributions are more sparse than before: temporal patterns of some (very) few channels can be responsible for a classification. Depending on the task, this high specificity can be an advantage, but also a disadvantage since one could expect the model robustness to suffer from these sparse positive attributions.

Dropout rate influence

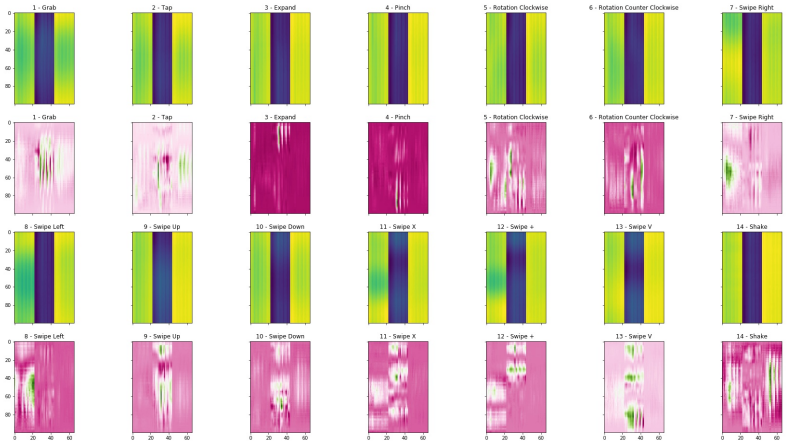
Robustness of model attributions for the last 66 independent branches model is visualized in figure 4.22.

The 66 independent branches model appears to have a comparable accuracy regardless of the dropout probability. However, since it also has more sparse attributions than the 3:XYZ-shared-branch model and the 1-shared-branch model as shown in figure 4.21, one can wonder how dropout rate affects the 66 independent branches model attributions. In figure 4.22, three models trained with different dropout rate are visualized: one without dropout, one with a $p = 0.40$ dropout rate, and one with a very high dropout rate: $p = 0.95$. For comparison, the same 66 independent branches model from previous figure 4.21 is trained with a $p = 0.55$ dropout rate.

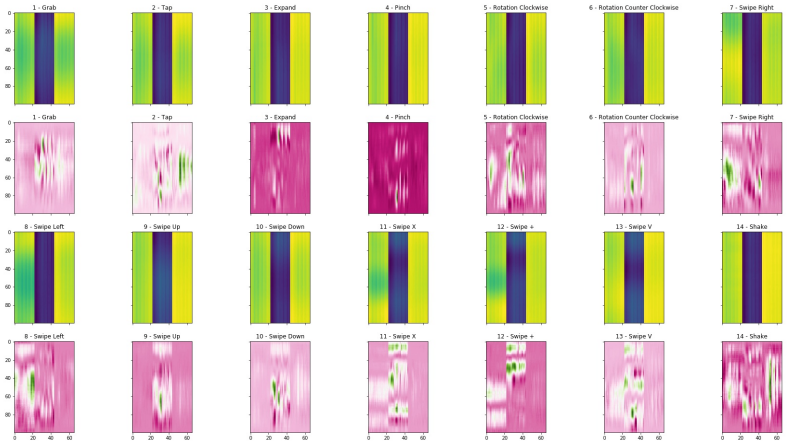
The model leads to similar attributions for a wide range of dropout rates, like $p = 0.40$. When no dropout is applied, attributions are slightly less sparse. However for extremely large dropout rates, the model does not benefit of its branches since most of them are dropped, making the model very close and comparable to the single channel model from figure 4.21. As a summary, excepted for extremely high dropout rates, the 66 independent branches model appears consistent and robust in the way it attributes a sequence to a given class, almost regardless of the dropout rate.



(a) Integrated Gradients attributions for SkelNet model with 66 independent branches.

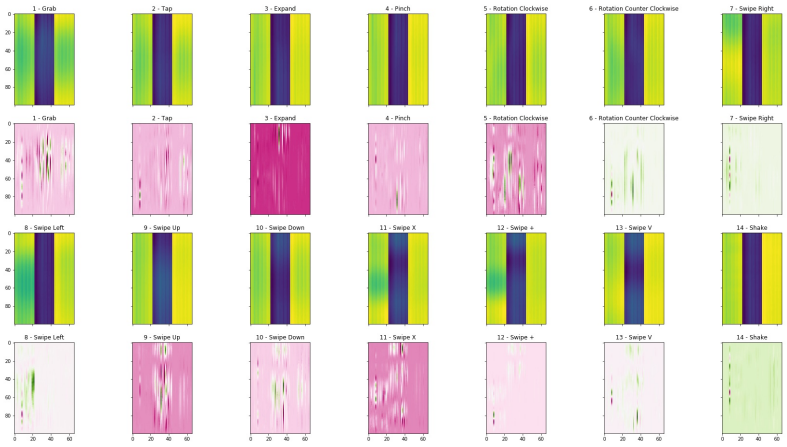


(b) Integrated Gradients attributions for SkelNet model with 3 (1 for X, 1 for Y, 1 for Z) shared branches.

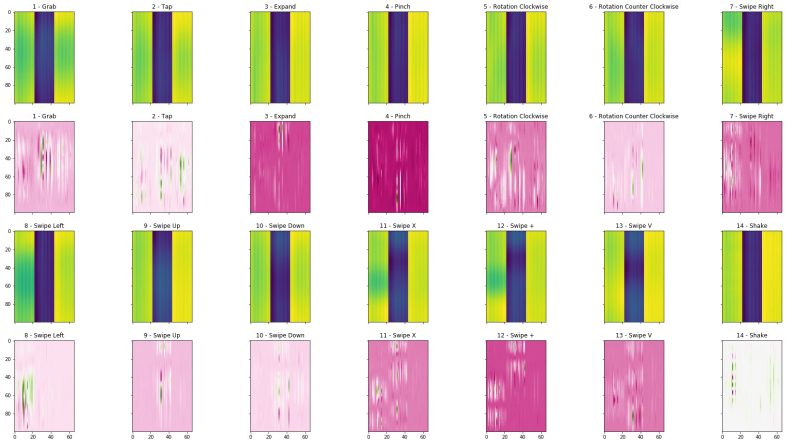


(c) Integrated Gradients attributions for SkelNet model with 1 shared branch for all channels.

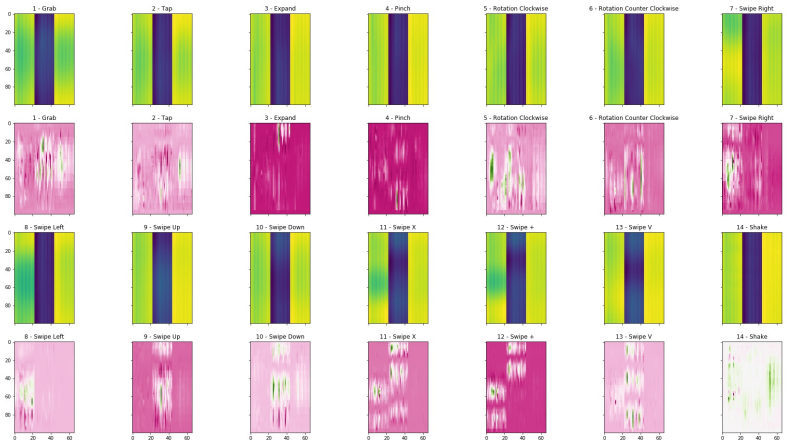
Figure 4.21 – Comparison and visualization of attributions for three models: C=66, C=3 (XYZ), C=1.



(a) Integrated Gradients attributions for a SkelNet model without dropout: $p = 0$



(b) Integrated Gradients attributions for a SkelNet model with dropout rate: $p = 0.40$



(c) Integrated Gradients attributions for a SkelNet model with dropout rate: $p = 0.95$

Figure 4.22 – Comparison and visualization of attributions for three dropout rates: $p = 0$, $p = 0.40$ and $p = 0.95$.

4.6 Conclusion

We present an approach for gesture recognition from human pose sequences, based on a family of 1D convolutional neural networks. The family of model we introduce, SkelNet, makes use of convolutions over the time dimension in order to extract the temporal patterns of the skeleton-based gesture dynamics. We study the performance of the models and observe shallow networks are sufficient for medium-sized datasets while deeper networks seem to be needed for bigger datasets. The effectiveness of our method is demonstrated by achieving high accuracy (91.5% accuracy) on a challenging hand gesture dataset (DHG 14 dataset from the SHREC17 Shape Retrieval Contest). Furthermore, the SkelNet model used is lightweight ($\sim$2.4M parameters). We also validated the applicability and the effectiveness of SkelNet models on two other type of datasets: one with sequences of full body poses and the other one on sequences of facial landmarks. As for future direction, we intend to improve our early recognition method, to integrate domain knowledge and to extend our method to support unsegmented data streams.

Chapter 5

Human Motion Generation with Deep Learning

"Every body continues in its state of rest, or of uniform motion in a right line, unless it is compelled to change that state by forces impressed upon it."

Newton

Contents

5.1 Introduction

In this chapter, we propose two novel approaches for human pose motion generation, with the help of neural networks. Motivations for human pose motion generation are first discussed in section 5.1.2. The two proposed approaches aim at generating sequences of human poses. They do not require any image information.

The first approach, introduced in section 5.3, is a time-contrastive autoencoder-based method that is trained on individual pose frames. We refer to this approach as a Temporal Triplet Pose Auto-Encoder (TTPAE) approach. The second approach, introduced in section 5.4, is a generative adversarial neural network method that works at a sequence level. We refer to this approach as a Spatio-Temporal Conditional Generative Adversarial Network (STCGAN) approach. Motivations for the two proposed approaches are discussed in sections 5.3.3 and 5.4.3 respectively.

While the two approaches can create new motions, the first one (TTPAE) almost decouples spatial and temporal information, whereas the second one (STCGAN) rather uses this coupled spatiotemporal information. In the TTPAE approach, spatial information is learned using fully-connected neural networks and temporal information is taken into account during training through a time-based triplet loss function. In the STCGAN approach, the spatial information and the temporal information are learned jointly using convolutional neural networks.

Besides adopting very different strategies from one another, the two proposed approaches also differ from each other at inference time when it comes to human pose sequence generation. Indeed, in the TTPAE approach, human pose sequences can be generated on-the-fly (i.e. one time step after another), whereas human pose sequences can only be generated all-at-once (i.e. a full sequence) in the STCGAN approach, because of the convolutional neural network architecture proposed for the STCGAN generator neural network.

5.1.1 Contributions summary

Our contributions are summarized in section 5.3.2 for the time-contrastive autoencoder-based approach, and in section 5.4.2 for the generative adversarial neural network approach.

5.1.2 Motivations

A generative model is a model that models the data generation process. Since generative models mostly rely on inductive biases, they tend to be good at out-of-domain generalization.

Motivations for pose motion generation include: realistic video synthesis, professional training in technical skills and gestures, animation movies, virtual reality, augmented reality, video games, ergonomic assessments, motion analysis for sport, musical or medical purposes, exploration and creation of dance choreographies.

From a machine learning perspective, models that generate realistic data can also be used for data augmentation. Data augmentation is a very common technique in supervised learning meant to improve the robustness of machine learning models.

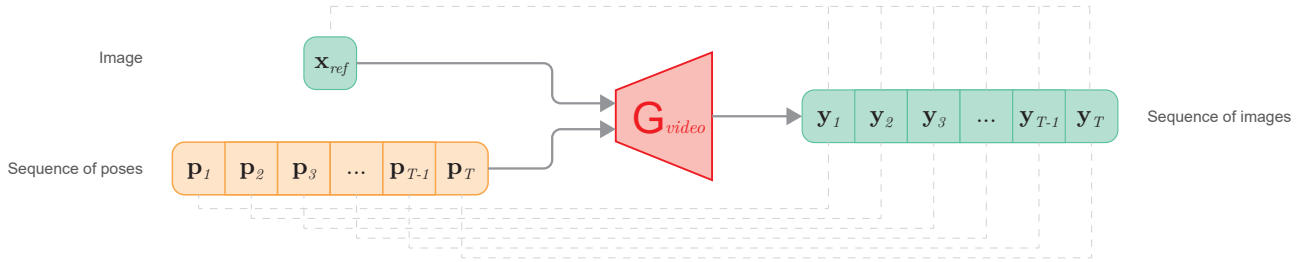


Figure 5.1 – Classical pipeline for human motion video generation: a sequence of images $\mathbf{y}$ is generated with a model conditioned on a still image $\mathbf{x}$ and a sequence of poses $\mathbf{s}$

The first motivation for motion pose generation is realistic video synthesis. Deep-learning-based models for video generation (video being considered as a sequence of images) have begun to produce realistic results in the recent years. Models with the most convincing outputs often consist of GAN-like generative neural networks, conditioned on a couple of: (1) a still image $\mathbf{x}_{ref}$ of the subject of the action; and (2) a sequence $(\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_T)$ of the subject's desired poses $\mathbf{p}_t$ over time. Based on that, they output a sequence $(\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_T)$ of images $\mathbf{y}_t$ over time, with textures and poses consistent with both $\mathbf{x}_{ref}$ and $\mathbf{p}_t$ at each time step t, as illustrated in figure 5.1. Existing generative models that are not conditioned on pose sequences lack of temporally coherent structure, highlighting the importance of pose sequences.

While the end-result performance and the inner mechanism of these models (e.g. GAN-based, VAE-based, RNN-based, ...) vary, they share a common drawback: they rely on a pre-existing known motion sequence of poses $\mathbf{s}$.

In animation movies, characters are represented with 3D meshes and joints that allow the 3D meshes to be easily deformable, and thus easy to animate using hand-chosen keyframes of the joint poses at specific time steps and time-interpolated values between these time steps. In virtual reality, augmented reality and other 3D tools, human-like avatars are usually represented in the same way.

In sport, musical and medical settings, analyzing one's motion can be very useful. A key step in such analyses lies in the motion decomposition, after the initial motion recognition step. Indeed, the motion decomposition allows for expert feedback about one specific aspect of the gesture performed, essentially on how to improve that aspect of the gesture. As such, motion decomposition and recombination can also be useful for ergonomics or technical skills professional training. Examples range from learning how to avoid musculoskeletal disorders, how to make better swings in golf, how to make better hand gestures for surgeons, to learning how to play cello better or faster. Being able to generate new motions (e.g. using a neural network for this generation) is as important as being able to distinguish motions (e.g. using a neural network to discriminate if a gesture is well performed or not).

Finally, more creative applications that use pose motion information can be considered, like new motion creation for sport or dance purposes: in these settings, new gestures, new moves and new choreographies could be created, either purely from an exploratory perspective (e.g. generating random gestures until one is artistically innovative, or discovering what would have happened if a tennis player had jumped while hitting the ball) or given a known context (e.g. creating a new dance whose style would lie in-between a waltz style and a shuffle dance style).

All of the examples mentioned above show that pose motion generation is increasingly gaining

interest from diverse communities. However human motion is hard to represent due to the complexity of human morphology. One way to apprehend human motion dynamics consists in encoding human expert knowledge into models of human body, e.g. by including weight of body parts and muscular constraints. Formalizing such expert knowledge is very difficult and not necessarily efficient. More recent approaches are based on deep learning models, capturing statistically relevant patterns by their own during training, relieving the need for human expert knowledge. Most deep learning models for human pose motion generation somehow combine the use of recurrent cells with a generative GAN or VAE framework, but still lack of realism or variety.

Once pose-only sequences are generated by a deep learning model, the motion can be given as the input of other deep learning models if required, e.g. the motion could be fed into a deep learning model conditioned on a still image and a pose sequence to generate a realistic video. End-to-end training of the two neural networks as one single neural network may likely be possible in most cases, e.g. in order to get a neural network that generates a realistic video given a still image only and no pose information.

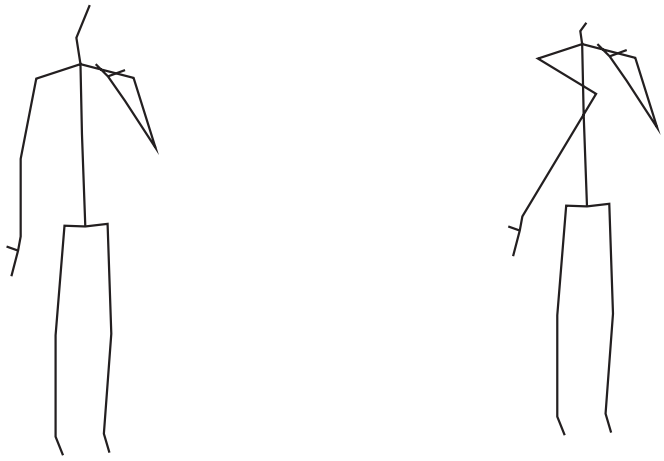
5.1.3 Spatial and temporal coherence of pose data

Pose-based recognition methods and pose-based generation methods are still active research fields. Studies on human visual perception of biological motion have shown that humans can recognize human body motion actions, using the motion of the body's (skeletal) joints positions only (Johansson, 1973). Human pose representations are high-level, sparse, representations of the human body. As such, working with pose-based models rather than image-based models may be more relevant for action recognition and generation, since these tasks are high-level tasks that involve semantics. Of course, one can also create models that use images, texts, audio signals or any other relevant information, in addition to poses.

Skeleton-based action recognition systems still face significant issues today.

First, skeletons are often noisy. This issue is especially noticeable for skeletons estimated using image-based approaches. Pose estimation methods tend to exhibit high nonlinearity in certain regions of the problem domain, and are not always robust to occlusions. Temporal smoothness may not be present if the vision-based pose estimation approach only works at a frame level. In other cases, pose estimation models can present overfitting, or not be very robust to the camera orientation. Regardless of misdetections, estimated skeletons are often noisy as illustrated in figure 5.2.

Second, generating realistic individual skeletons is difficult. For instance, one could think of training an auto-encoder (AE) to minimize the L_1 or L_2 norm reconstruction loss between a ground truth skeleton and the generated autoencoded skeleton. When generating skeletons using such a successfully trained autoencoder, the quantitative reconstruction error of the generated skeletons do not necessarily correlates with the human perception of what a skeleton is or not. This may be explained by several reasons. On the one hand, the human eye does not tolerate reconstruction errors evenly: a small error on a joint's location will be perceived as higher if the joint is close to another one for instance. On the other hand, L_1 and L_2 distances highly suffer from noise. However, ground-truth skeletons, which are considered to represent the reality, are sometimes very noisy. Handcrafting such a better reconstruction loss function is difficult, especially if taking into account the noise.



(a) Noise-free Human Pose (b) Noisy Human Pose

Figure 5.2 – Illustration of noise on a human pose. Note the elbow and the head joints' positions.

Third, the main issue left to generate pose motion is to produce spatially coherent skeletons at each time step and to ensure in the meantime that the overall movement displays a temporal consistency. This can be an issue for most methods that generate skeleton data, including methods based on variational auto-encoders (VAEs). While recurrent neural networks (RNNs) generally ensure temporal consistency of the sequences they generate, RNNs present several drawbacks. The theoretically very long time horizon of RNNs like LSTMs and GRUs is largely absent in practice. RNNs are slow to train, tend to repeat elements and to be cyclic. Moreover LSTMs and GRUs are known to have underlying dynamical systems that exhibit chaotic behavior. Finally, models that try to mix realistic individual skeleton generation methods (e.g VAEs) with temporally coherent methods (e.g. RNNs) tend to quickly become complicated while producing almost inconclusive empirical results.

5.2 Related Works

Human motion generation gains increasing attention in the research community (Längkvist et al., 2014), the recent works using deep-learning approaches for spatiotemporal modeling. In particular two main directions have been explored in recent works: recurrent-based models that essentially separate the spatial structure and the temporal dependencies, and generative adversarial networks that learns them simultaneously.

5.2.1 Recurrent Neural Networks (RNN) models

Recurrent neural networks (RNNs) are widely employed for sequential data and several recurrent-based and autoencoder-based models are proposed in literature for a variety of human motion tasks including action classification, action prediction, action anticipation and motion synthesis (Fragkiadaki et al., 2015; Ghosh et al., 2017).

In (Chung et al., 2015), authors developed a variational recurrent neural network (VRNN) with high-level latent random variables conditioning a variational auto-encoder at each time step, that has

been extended on human activity modeling in a semi-supervised fashion in (Bütepage et al., 2018). In the related field of object extraction and prediction in videos, an auto-encoder is combined with a VRNN in (Minderer et al., 2019). The autoencoder detects key points in 2D images while the VRRN learns the dynamics of these key points. Recurrent networks and auto-encoders used in Encoder-Recurrent-Decoder (ERD) fashion has been introduced in (Fragkiadaki et al., 2015) but they produce unrealistic human motion due to accumulated errors. In (Fragkiadaki et al., 2015), authors address it by gradually adding noise to the input during training, but this noise scheduling process is hard to tune in practice. The Dropout Autoencoder LSTM (DAE) introduced in (Ghosh et al., 2017) achieves better performances and robustness due to the DAE removing randomly some joints during training in order to learn human skeleton dependencies. More recently (Martinez et al., 2017a) has improved the RNN baseline by adding residual connections that enables the model to focus on short-term motion prediction. Recurrent encoder-decoders with RNN can be used to directly predict the whole sequence, using recurrent networks in on-sequence encoder and decoder. In (Harvey et al., 2018), an on-frame autoencoder is combined with an on-sequence autoencoder, showing that the use of on-frame autoencoding tends to denoise noisy skeletal data joints.

5.2.2 Generative Adversarial Networks (GAN) models

Another major approach of human motion prediction and synthesis resorts to Generative Adversarial Networks (GANs) in order to directly synthesize a whole pose sequence. A simple approach is proposed in (Kiasari et al., 2018) whose model jointly trains an AE and a GAN, conditioned on the initial state and the given class label: the AE encodes the initial state in a low-dimensional representation used to train a GAN -conditioned on action class label- to generate sequences of low-dimensional sequences. The use of GAN for motion generation has received major attention in the related field of 2D video prediction conditioned with a starting 2D image and an already known sequence of poses (Balakrishnan et al., 2018; Z. Huang et al., 2018; Ma et al., 2017; H. Tang et al., 2018), and (Siarohin et al., 2019) for the prediction of a single 2D image from one reference image and one pose. In practice, however, human motion models based on recurrent networks and generative adversarial networks lack of smoothness. The generated sequences often notably lack of realism to the human eye, even for models with apparent low spatial joints' position prediction errors. A triplet loss can be added to the generation loss in order to penalize the adjacent frames having larger distance than the non-adjacent ones, as (Y. Yan et al., 2017) do for 2D images. In the field of 3D human motion prediction (Barsoum et al., 2018) simultaneously trains a Wasserstein Generative Adversarial Network (WGAN) to predict a whole sequence of poses and a motion-quality-assessment model to learn whether a given skeleton sequence is a real human motion. The potential of a multi-source discriminator to generate more anthropometrically realistic poses has been shown in (W. Yang et al., 2018) in the related field of 3D human pose estimation from 2D images. Another solution to ensure better sequence realism has been proposed in (Pavllo et al., 2018) that uses quaternions to represent rotations and that penalizes absolute position errors rather than angle errors in order to avoid error accumulation along the kinematic chain and to ensure more realistic positions of skeletal joints. However, for 3D rotations, both quaternions and Euler angles representations are discontinuous and difficult for neural networks to learn (Y. Zhou et al., 2019). In order to generate

dance from music, (H.-Y. Lee et al., 2019) propose an approach with three main phases: decomposition, composition, and testing. To that end, individual poses are first encoded by a Variational Auto-Encoder (VAE). The encoded vectors are then used to train a Generative Adversarial Network (GAN) conditioned on the audio. Finally, the trained generator neural network is used in a Recurrent fashion to synthesize the final dance.

5.2.3 Triplet loss

A formal definition of a triplet loss is proposed in section 5.3.9.

In the deep metric learning research area (Kaya et al., 2019), a triplet loss is a loss function where a reference (anchor) input is compared to a positive input and a negative input, according to a margin parameter: as a result a triplet loss constrains the distance gap between dissimilar clusters. For instance, triplet loss is frequently used in image-based deep-learning models for (re-)identification and distinction of video frames (Y. Yan et al., 2017). Deep metric learning aims at automatically constructing task-specific distance metrics from data. Except approaches based on a triplet loss, other promising approaches in deep metric learning exist, including contrastive loss (Hadsell et al., 2006) where only two inputs are considered[1], neighbourhood components analysis (Goldberger et al., 2005) where a distance is learned based on the classification performance of an average leave-one-out approach, or angular losses (Deng et al., 2019; Jian Wang et al., 2017) where angular constraints and/or representations are favoured over euclidean distance constraints and/or representations -as in the triplet loss case-.

While newer state-of-the-art loss functions have been proposed in the research literature, it actually appears that they tend to "perform marginally better than, and sometimes on par with, classic methods" (Musgrave et al., 2020). As such, triplet loss can still be considered as a key and central tool for deep metric learning purposes.

While the triplet loss is very frequently and successfully used in the image domain, very limited results based on a triplet loss exist in the 2D/3D human pose domain. Except in this work, and to the to the best of our knowledge, the triplet loss has not been applied to pose-only data before, except in two cases: namely for few-shot learning in gesture recognition (Granger, 2019) and for human pose embedding (Sun et al., 2020). While our work was performed in parallel of the two approaches mentioned -the earliest one being (Granger, 2019)- and independently from them, we do not claim the use of a triplet loss on static human poses as being a contribution of ours *per se*.

For images, triplet loss usually helps models to display texture continuity between two successive similar frames, however no temporal continuity is usually neither enforced nor guaranteed, especially due to the information density inherent to images. Time-Contrastive networks (TCNs) from (Sermanet et al., 2018) use triplet-loss for self-supervised temporal frames matching between images from different views.

The TCN frame sampling method in a video single-view setting is a sampling approach very close to the sampling approach we propose: where the TCN samples RGB frames from a video, we propose to sample human poses of a sequence, as described in more details in section 5.3.10. Nonetheless, in the

[1] Arguably leading to an absolute similarity learning and not a relative similarity learning as in the triplet loss case, according to some authors.

TCN paper no guarantee that the sampling method is sufficient for sequence generation is provided, and only content similarity *per se* is pursued. Moreover, the data type used in the TCN paper is RGB frames and not on human poses. Human poses contain way less redundant information compared to RGB images where large portions of an image may be related and/or similar. Distinguishing between signal and noise is also much harder in the human pose domain compared to the image domain (see figure 5.2). Distinguishing between different clusters becomes more difficult when the distance inter-cluster (related to time, in our case) is comparable or even lower to the distance intra-cluster (related to static poses, in our case). Given the specificity inherent to human poses, we consider our frame sampling method (and its application to human pose sequence generation with a triplet loss) as a contribution.

In the image sequence (i.e. in the video) domain, other methods for triplet sampling have been proposed in the literature, in order to preserve a temporal coherence between images. In image sequences, adjacent frames can sometimes have totally different semantics from the next ones, e.g. to camera shaking. For a given anchor frame, (Pan et al., 2016) propose to extend the time-based triplet sampling method by ranking all frames of the sequence based on their respective Euclidean distance to the anchor frame, and restrict the positive and negative anchor frames based on the frames ranking before using a time-based sampling approach. However, it is widely known that the Euclidean distance is a poor discrepancy metric for time-series applications (Gharghabi et al., 2018). While handcrafting more appropriate discrepancy measures between poses might be worth the effort in order to shortlist acceptable candidates for the triplet, a formal exploration of all the descriptors introduced in the related works section of the chapter 4 might probably be as much worth the effort. Finally, this approach requires handcrafted features or loss functions whereas the goal of most deep metric learning approaches is essentially to fully learn the metrics based on data. Still in the image sequence (i.e. in the video) domain, (Redondo-Cabrera et al., 2019) propose to extend the learning based on a temporal coherence not only from between frames in the same video but also between frames from different videos. Finally, as detailed in (Kaya et al., 2019), numerous other works focus on learning the mining of hard triplet samples, e.g. with the help of adversarial losses. However, to the best of our knowledge, these works' approaches do not guarantee a temporal coherence (other than by performing a "smart" shortlist of triplet candidates regardless of time, the candidates later being used as inputs in a time-based triplet sampling approach).

In the approach we propose in the section 5.3, both the temporal sampling method and the sparsity of the data suggest that the vanilla triplet loss we use enforces a temporal coherence and continuity of poses. To the best of our knowledge, our triplet-loss-aided approach is the first one to use denoising auto-encoders with a triplet loss for skeleton data sequences synthesis.

5.2.4 Other approaches

Some original approaches of human motion anticipation do not resort to RNNs or GANs. In (S. Yan et al., 2019) a Convolutional Sequence Generation Network (CSGN) is proposed: latent vectors are sampled with a Gaussian Process (GP) and convolved at different time scales to generate time-structured skeletal joint sequences. The AE-ProMPs introduced in (Dermy et al., 2018) uses Probabilistic Movement Primitives computing (PRoMPs), inspired from the robotics field, to infer the continuation of a trajectory

conditioned by its initial portion. However, this dynamics-based method does not enable human motion generation from scratch.

5.3 Temporal Triplet Human Pose Auto-Encoder

In this section, we propose a self-supervised learning method to generate sequences of human poses that are temporally coherent and robust to noise, using a temporal triplet loss.

5.3.1 Approach summary

We propose a self-supervised approach for learning representations of human poses in a latent space, with the help of a denoising autoencoder neural network. The proposed approach is illustrated in figure 5.3 and explained below.

A denoising autoencoder learns to encode a pose into a latent space and to decode it back. Compared to a vanilla autoencoder whose role is to output the same pose as the input pose, the denoising autoencoder's role is to output a cleaned version of a noisy input pose, i.e. to output the input pose but with noise removed.

The denoising autoencoder is trained to minimize a loss that combines a reconstruction term, to ensure spatial coherence, and a temporal triplet term, to ensure temporal coherence.

Once the neural network is trained, we observe that both the encoder and the decoder modules of the neural network are highly continuous. Generating a new (static) pose is as easy as sampling a random vector in the latent space and decoding it with the decoder. Generating a new (dynamic) sequence of poses is deceptively simple as well. Two random latent vectors are first sampled in the latent space; an encoded sequence is created using a simple linear interpolating between the two vectors; each one of the encoded poses is decoded with the decoder yielding to the final sequence.

Generated sequences are spatially and temporally coherent, and even often look more realistic to the human eye than real, ground-truth, but noisy, sequences.

We deliberately describe the different parts of our approach separately, for the sake of clarity. First, we describe the denoising encoder-decoder as if it were standalone.

5.3.2 Contributions summary

As mentioned earlier in section 5.2.3, the use of a triplet loss in the human pose domain was first proposed in (Granger, 2019) and our temporal sampling method is inspired by the temporal sampling method from (Sermanet et al., 2018) in the image domain.

Besides the overall approach and (hyper-)parameters, our contributions more specifically are: a denoising autoencoder architecture introduced in section 5.3.4 coupled with a temporal loss, a temporal sampling method introduced in section 5.3.10, a noise sampling method introduced in section 5.3.7. Finally we propose to generate human pose sequences based on the proposed approach and the latent-code interpolation sampling method from section 5.3.11.

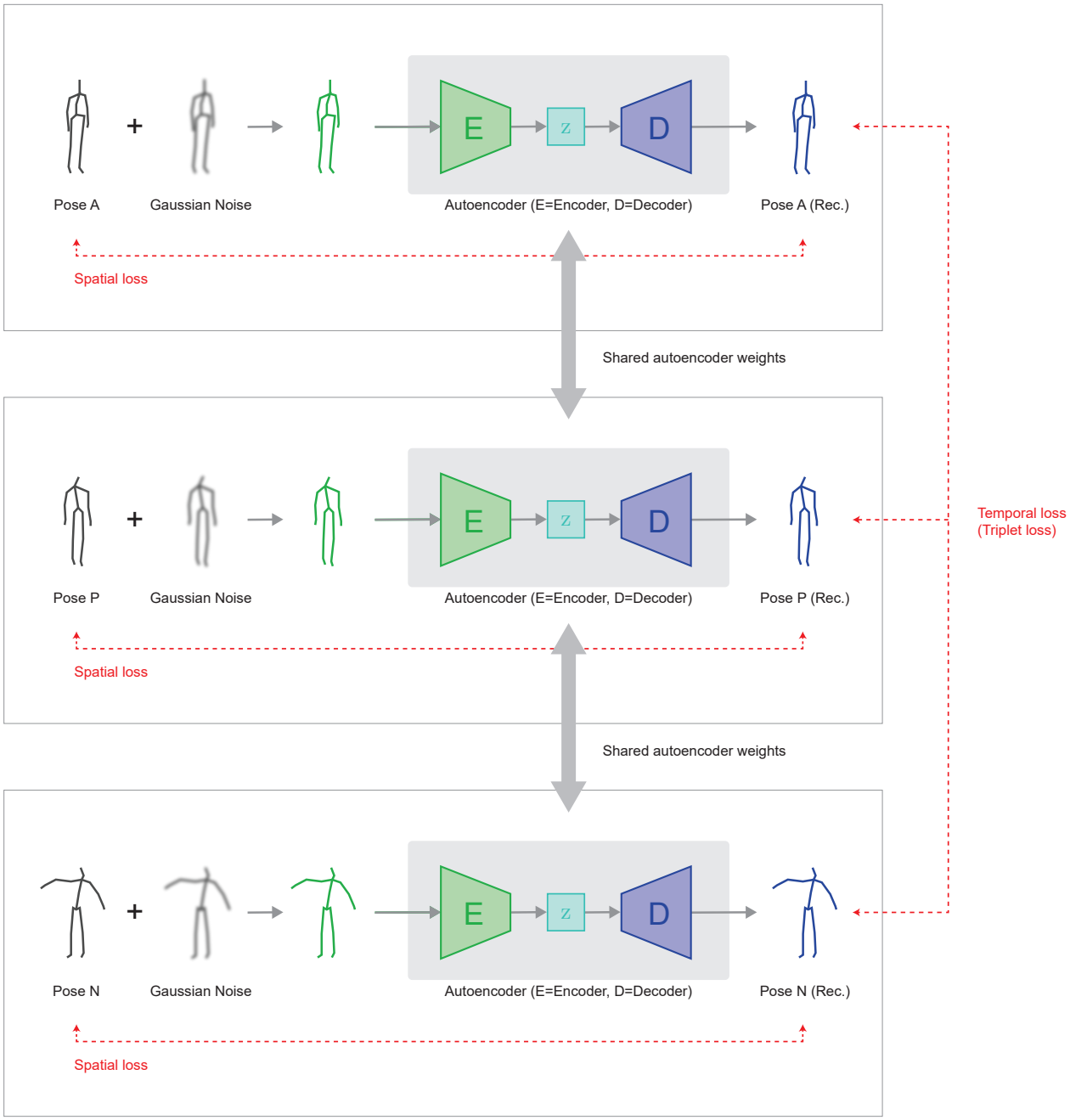


Figure 5.3 – Triplet Denoising Autoencoder for Human Poses. Three human poses: A (for anchor pose), P (for positive pose) and N (for negative pose) are sampled, as described in section 5.3.9. A and P are similar, while A and N are dissimilar. All the poses are first corrupted with a Gaussian noise perturbation, as described in section 5.3.7. They are then reconstructed -independently- by a *same* denoising-autoencoder, whose role is to learn to (spatially) reconstruct original noise-free poses from corrupted noisy poses. A triplet loss is used to constrain the autoencoder's generated poses not only spatially, but also temporally, as described in section 5.3.9. The model is differentiable end-to-end.

5.3.3 Model intuition

Neural networks whose layers size form an hourglass-like shape typically excel at compressing and decompressing information. Such neural networks are called autoencoders since they encode their input

into a smaller vector and then decode it back to an output which is equal to the input, hence the "auto-" prefix. The first layers of the network form a so-called "encoder" while the remaining last layers form a "decoder". The whole network can thus be seen as an "encoder-decoder". We will train an autoencoder to compress poses into a compressed vector in a latent space and convert that representation back to poses. The network will be optimized to ensure the reconstructed pose are spatially correct. Rather than only autoencoding poses, our network will also remove noise present in the poses.

5.3.4 Model architecture

The architecture of the model consists in a neural network with hourglass-like shaped fully-connected layers; the sizes of the layers in the encoder and in the decoder are symmetrical. No weight is shared between the encoder and the decoder. All layers use Leaky ReLU activations with a negative slope of 0.2.

For regularization, the units of the fully-connected layers are dropped with a Dropout probability of $p = 0.15$ in order to limit their co-adaptation. Batch Normalization (BatchNorm) technique is also used. BatchNorm is performed after each layer's nonlinearity and dropout. The reasons behind the effectiveness of batch normalization remain under active discussion in the research community. The use of both dropout and batchnorm altogether is also subject to discussion and debated, however no clear consensus has emerged at the time of the writing of this PhD thesis about whether both techniques could, could not, should or should not be used altogether. We use both methods to regularize the denoising autoencoder model.

The autoencoder takes a flattened pose vector of dimension $J \times d$, where J is the count of joints in the body's structure and d is the dimension (count of components) of each joint. In our experiment, $J \times d = 75$. We chose a latent space of size $z_{dim} = 32$ for the auto-encoder.

The model architecture is detailed in table 5.1 for the encoder and in table 5.2 for the decoder.

Table 5.1 – Human Motion Generation: Encoder neural network architecture

Encoder				
#	FC(input size, output size)	Activation(negative slope)	Dropout(rate)	BatchNorm
1	FC($75 = J \times d$, 2048)	LeakyReLU(0.2)	Dropout(0.15)	No
2	FC(2048, 1024)	LeakyReLU(0.2)	Dropout(0.15)	Yes
3	FC(1024, 512)	LeakyReLU(0.2)	Dropout(0.15)	Yes
4	FC(512, 256)	LeakyReLU(0.2)	Dropout(0.15)	Yes
5	FC(256, $32 = z_{dim}$)	LeakyReLU(0.2)	Dropout(0.15)	Yes

5.3.5 Model loss function, dataset, model training

The full denoising encoder-decoder model is trained end-to-end in order to optimize a spatiotemporal objective. The spatiotemporal objective consists of two different, mostly decoupled objectives: a spatial objective and a temporal objective.

The $\mathscr{L}_{\text{spatiotemporal}}$ spatiotemporal model loss function to optimize is thus given by:

Table 5.2 – Human Motion Generation: Decoder neural network architecture

Decoder				
#	FC(input size, output size)	Activation(negative slope)	Dropout(rate)	BatchNorm
1	FC($32 = z_{dim}$, 256)	LeakyReLU(0.2)	Dropout(0.15)	No
2	FC(256, 512)	LeakyReLU(0.2)	Dropout(0.15)	Yes
3	FC(512, 1024)	LeakyReLU(0.2)	Dropout(0.15)	Yes
4	FC(1024, 2048)	LeakyReLU(0.2)	Dropout(0.15)	Yes
5	FC(2048, $75 = J \times d$)	No activation (Linear)	Dropout(0.15)	No

$$\begin{aligned}\mathscr{L}_{\text{spatiotemporal}} &= \mathscr{L}_{\text{spatial}} + \mathscr{L}_{\text{temporal}} \\ &= \left(\alpha_{\text{reconstruction}}\mathscr{L}_{\text{reconstruction}} + \alpha_{\text{anatomy}}\mathscr{L}_{\text{anatomy}}\right) + \alpha_{\text{triplet}}\mathscr{L}_{\text{triplet}}\end{aligned} \tag{5.1}$$

where:

$\mathscr{L}_{\textbf{reconstruction}}, \mathscr{L}_{\textbf{anatomy}}$ are two spatial loss functions,

$\mathscr{L}_{\textbf{triplet}}$ is a temporal loss function,

$\alpha_{\textbf{reconstruction}}, \alpha_{\textbf{anatomy}}, \alpha_{\textbf{triplet}}$ are hyperparameters.

The exact formal expression of the spatial objective and its hyperparameters will be introduced in section 5.3.6 while the exact formal expression of the temporal objective and its hyperparameter will be introduced in section 5.3.9.

Backpropagation with the Adam optimizer (D. Kingma et al., 2014) is used to perform the training of the model parameters. The optimizer initial learning rate is set to 1.0×10^{-4} and the following Adam parameters: $\beta_1 = 0.5$ and $\beta_2 = 0.99$.

For experimentation, the NTU RGB+D dataset (Shahroudy et al., 2016) is used. Training is performed with the 3D (x, y, z) pose sequences. In training and validation, the dataset author's cross-subject split is used, in order to avoid model overfitting on individual subjects. Data is normalized: all sequences are max-normalized (i.e. scaled such as the whole sequence fits in a 3D bounding box of unitary width) and centered such as the joint that represents the human body's hips is at the origin. Data augmentation is performed on the original sequences by applying random -credible- rotations centered around the hips joint and by adding noise to the input pose sequences.

5.3.6 Spatial coherence loss: reconstruction and anatomy

Formally, let $\mathbf{s} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T) \in \mathbb{R}^{T \times J \times d}$ a human pose sequence, where T is the sequence duration, J is the number of body joints and d is the dimensionality of each joint.

Let D and E be two fully connected neural networks. The exact model architectures used for D and E are the ones previously detailed in section 5.3.4. We consider the encoder-decoder neural network $AE = D \circ E$ where E is the encoder network D is the decoder network.

The encoder-decoder $AE = D \circ E$ is trained to minimize a spatiotemporal coherence loss, which consists in a spatial coherence term and a temporal coherence term. The spatial coherence loss for a single pose is given by:

$$\mathscr{L}_{\text{spatial}}(\mathbf{x}, \widetilde{\mathbf{x}}) = \alpha_{\text{reconstruction}} \mathscr{L}_{\text{reconstruction}}(\mathbf{x}, \widetilde{\mathbf{x}}) + \alpha_{\text{anatomy}} \mathscr{L}_{\text{anatomy}}(\mathbf{x}, \widetilde{\mathbf{x}}) \tag{5.2}$$

with

$$\mathscr{L}_{\text{reconstruction}}(\mathbf{x}, \widetilde{\mathbf{x}}) = \|\mathbf{x} - D \circ E(\widetilde{\mathbf{x}})\| \tag{5.3}$$

$$\mathscr{L}_{\text{anatomy}}(\mathbf{x}, \widetilde{\mathbf{x}}) = \|\sigma(\mathbf{x}) - \sigma(\widetilde{\mathbf{x}})\| \tag{5.4}$$

$$\sigma(\mathbf{x}) = \sum_{(i,j) \in \mathscr{B}} \left\| \mathbf{x}^{(i)} - \mathbf{x}^{(j)} \right\| \tag{5.5}$$

where:

$\|\cdot\|$ is the L_2 norm,

$\mathbf{x}$ is a pose,

$\widetilde{\mathbf{x}}$ is a noisy pose, i.e. $\widetilde{\mathbf{x}} = \mathbf{x} + \varepsilon$ where ε is a pose noise,

$\mathbf{x}^{(i)}$ is the i-th component of the pose $\mathbf{x}$, i.e. $\mathbf{x}^{(i)}$ is the joint i of the pose $\mathbf{x}$

$\mathscr{B}$ is the set of all couples (i, j) such as $i < j$ and a bone exists between the joint i and the joint j,

$\alpha_{\text{reconstruction}}$ is an hyperparameter whose value is set to 0.9995,

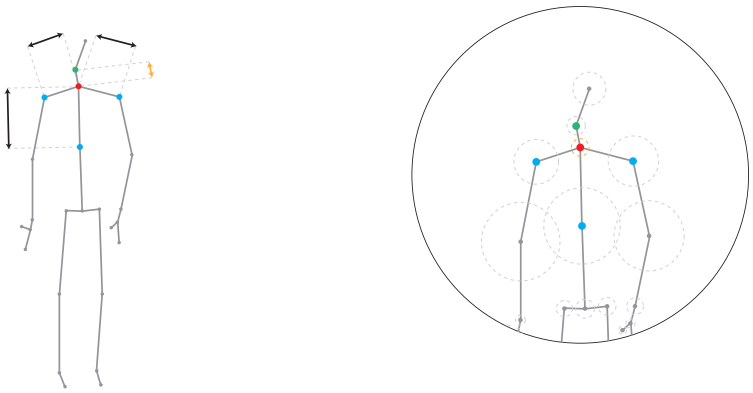
α_{anatomy} is an hyperparameter whose value is set to 0.0002.

The goal of optimizing the reconstruction loss $\mathscr{L}_{\text{reconstruction}}$ is to give the neural network $D \circ E$ the ability to reconstruct a noise-free pose $\mathbf{x}$ from a noisy pose $\widetilde{\mathbf{x}}$. Another loss, $\mathscr{L}_{\text{anatomy}}$, is added in order to ensure that the human anatomy is respected. $\mathscr{L}_{\text{anatomy}}$ compares the cumulative length of all human body's bones in the generated pose with the cumulative length of all human body's bones in the original pose. The anatomy loss is referred to as the "anthropometric regularity" constraint in 3D human pose estimation domain (Ramakrishna et al., 2012). The two losses (reconstruction and anatomy losses) are weighted with two hyperparameters, $\alpha_{\text{reconstruction}}$ and α_{anatomy}.

5.3.7 Noise sampling during training

Let B the number of bones in the body, and let $\mathbf{d}_{\text{bones}} \in \mathbf{R}^B$ represent the vector of lengths of a human body's bones. Knowing the human body's structure (or equivalently its adjacency matrix), we easily derive from $\mathbf{d}_{\text{bones}}$ the vector $\mathbf{d}_{\text{joints}} \in \mathbf{R}^J$ where each component i represents the minimum distance from the joint i to, and only to, the other joints it is connected with. That vector will be used for noise sampling during training. A visual illustration of (a) the minimum distances between a joint and its

neighbors and (b) a visual interpretation of the vector $\mathbf{d}_{\text{joints}} \in \mathbf{R}^J$ as a "safe" noise scale description is proposed in figure 5.4.



(a) Minimum distance between a joint and its neighbors

(b) All these minimal distances are used to determine the Gaussian noise sampling's standard deviation

Figure 5.4 – Noise amplitude determination

During training, a random Gaussian noise ε is sampled and added to a ground-truth pose $\mathbf{x}$ in order to create a noisy pose $\widetilde{\mathbf{x}} = \mathbf{x} + \varepsilon$. Modeling the noise as a Gaussian noise is not a strong assumption, given that poses are often estimated with vision-based methods or vision-like sensors. However, appropriately modeling the parameters of the Gaussian distribution is crucial, because pose is very sensitive to noise, due to the human anatomy. We use a centered Gaussian noise. Rather than being unit-scale, that Gaussian noise is scaled differently for each dimension. More precisely, for each joint i, the standard deviation ${\sigma_\varepsilon}^{(i)}$ component is chosen so that, $3 \times {\sigma_\varepsilon}^{(i)} = 0.35 \times \mathbf{d}^{(i)}_{\text{joints}}$. This ensures that roughly 99.7% of the sampled noises will have a magnitude less or equal to 35% of $\mathbf{d}_{\text{joints}}$. This ensures that adding the Gaussian noise to the original pose does not deform the original too much, e.g. by avoiding that a shoulder joint be "swapped" with a head joint due to noise. While the exact value of 0.35 is somehow arbitrary, a value less than 0.5 nevertheless seems experimentally guarantee that noisy poses still look like credible poses to the human eye.

5.3.8 Generating a static pose

We train the encoder-decoder neural network with the spatial loss only for now for analysis[2].

We observe that training the proposed encoder-decoder model with the spatial loss only and with data augmentation (limited random rotations and translations of the original ground-truth poses) leads to a model that is perceptually relatively continuous in term of encoder-transformation or decoder-transformation. We observe that the use of data augmentation is crucial to get more continuity in the model: almost surprisingly, data augmentation on our encoder-decoder trained with the spatial-loss only,

[2]The model optimized on the complete loss with both spatial and temporal loss terms will also be trained end-to-end all at once.

leads to results that are visually comparable to VAE-based models for poses like the one from (Pavlakos et al., 2019), but without requiring any non-deterministic variational Bayesian method.

To generate new poses, we sample an encoded vector $\mathbf{z}$ from a standard Gaussian distribution with zero mean and unit variance: $\mathbf{z} \sim \mathcal{N}(0,1)$, and then decode it with the decoder neural network D to obtain a pose $\mathbf{x}$:

$$\mathbf{x} = D(\mathbf{z}) \tag{5.6}$$

Since the decoder network is continuous, one may think of generating a sequence of poses as generating a sequence of encoded vectors and decoding them with the decoder. That is the approach we propose to generate motion pose sequences.

Ideally, a linear interpolation, defined in equation 5.7, between to encoded vectors $\mathbf{z}_A$ and $\mathbf{z}_B$, should lead to a realistic motion after decoding the interpolated vector $\mathbf{z}(t)$ at each time step t.

$$\mathbf{z}(t) = \alpha(t)\mathbf{z}_A + (1-\alpha(t))\,(\mathbf{z}_B - \mathbf{z}_A) \tag{5.7}$$

where $\alpha(t) = \frac{t}{N}$ for $t \in [\![0, N]\!]$.

However, while our AE approach is largely sufficient to produce realistic frames, no temporal consistency is guaranteed between frames[3]. To resolve that issue, we introduce another additional loss in the complete loss function we optimize: a temporal loss.

5.3.9 Temporal coherence loss: triplet

The encoder-decoder $AE = D \circ E$ is trained to minimize a spatiotemporal coherence loss, which consists in a spatial coherence term and a temporal coherence term. The temporal coherence loss for a triplet of poses is given by:

$$\mathscr{L}_{\text{temporal}}(\mathbf{x}_A, \mathbf{x}_\oplus, \mathbf{x}_\ominus, \widetilde{\mathbf{x}_A}, \widetilde{\mathbf{x}_\oplus}, \widetilde{\mathbf{x}_\ominus}) = \alpha_{\text{triplet}} \mathscr{L}_{\text{triplet}}(\mathbf{y}_A, \mathbf{y}_\oplus, \mathbf{y}_\ominus) \tag{5.8}$$

where:

$\mathbf{x}_A$ is an "anchor" pose,

$\mathbf{x}_\oplus$ is a "positive" pose (similar to $\mathbf{x}_A$),

$\mathbf{x}_\ominus$ is a "negative" pose (dissimilar to $\mathbf{x}_A$),

$\widetilde{\mathbf{x}}$ is a noisy pose, i.e. $\widetilde{\mathbf{x}} = \mathbf{x} + \varepsilon$ where ε is a pose noise:

$$\begin{aligned} \widetilde{\mathbf{x}_\mathbf{A}} &= \mathbf{x}_\mathbf{A} + \varepsilon_\mathbf{A} \\ \widetilde{\mathbf{x}_\oplus} &= \mathbf{x}_\oplus + \varepsilon_\oplus \\ \widetilde{\mathbf{x}_\ominus} &= \mathbf{x}_\ominus + \varepsilon_\ominus \end{aligned} \tag{5.9}$$

α_{triplet} is an hyperparameter whose value is set to 0.0003,

[3] Other existing models, e.g. VAE-based models, suffer from the same issue.

$\mathbf{y}$ is the reconstruction of a noisy pose $\widetilde{\mathbf{x}}$ by the denoising autoencoder $AE = D \circ E$:

$$\begin{aligned} \mathbf{y}_A &= D \circ E(\widetilde{\mathbf{x}_A}) \\ \mathbf{y}_\oplus &= D \circ E(\mathbf{x}_\oplus) \\ \mathbf{y}_\ominus &= D \circ E(\mathbf{x}_\ominus) \end{aligned} \tag{5.10}$$

and where $\mathscr{L}_{\text{temporal}}$ is the triplet loss given by:

$$\mathscr{L}_{\text{triplet}}(\mathbf{y}_A, \mathbf{y}_\oplus, \mathbf{y}_\ominus) = \max\left(\|\mathbf{y}_A - \mathbf{y}_\oplus\| - \|\mathbf{y}_A - \mathbf{y}_\ominus\| + \delta, 0\right) \tag{5.11}$$

where:

$\|\cdot\|$ is the L_2 norm,

δ is a "margin" scalar whose value is set to 1.0

Deep Metric Learning is a family of deep learning techniques whose goal is to measure the similarity among samples using a learned embedding (Duan et al., 2018; Kaya et al., 2019). Among deep metric learning techniques, a key and widely used technique consists in the use of a triplet loss (Weinberger et al., 2009). A formal definition of the triplet loss equations is given in equation 5.11, while a visual illustration of the triplet loss is proposed in figure 5.5.

The intuition behind the triplet loss is to force the neural network to have similar internal representations for poses that are known to be similar, and less similar internal representations for poses that are known to be less similar.

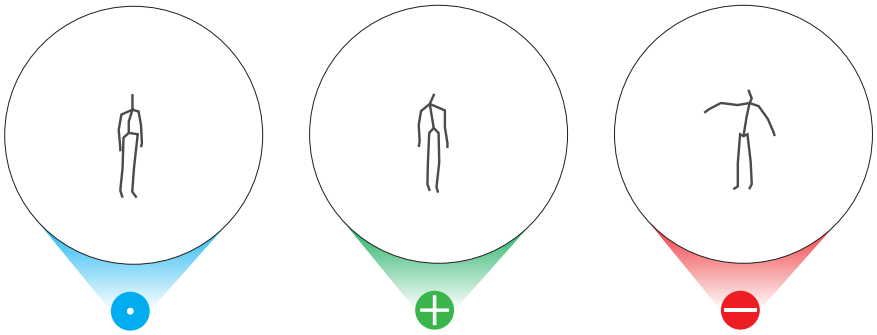
To that end, a triplet of poses are sampled. First a reference "anchor" pose $\mathbf{y}_A$. Second, another pose $\mathbf{y}_\oplus$ that is known to be very similar to the anchor pose. Third, a last pose $\mathbf{y}_\ominus$ that is known to be dissimilar to the anchor pose.

The triplet loss compares the distance $\|\mathbf{y}_A - \mathbf{y}_\oplus\|$ between the anchor pose $\mathbf{y}_A$ and the positive pose $\mathbf{y}_\oplus$ with the distance $\|\mathbf{y}_A - \mathbf{y}_\ominus\|$ between the anchor pose $\mathbf{y}_A$ and the negative pose $\mathbf{y}_\ominus$, and ensures that the first one is lesser or equal than the second one, effectively distancing the representations of the positive and the negative poses. A fixed additional margin δ is also added, to ensure the negative pose is spaced with the anchor pose, in a relatively similar fashion to the support vector machines (SVM) margins. Both ordering the poses and introducing a margin between them help to make their representations more separable.

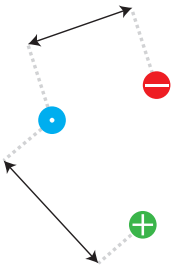
5.3.10 Pose (temporal) sampling during training

While triplet loss for poses increases robustness of the poses' representations and makes it easy to navigate between them since similar poses have representations that are grouped together, one still has to devise how to sample triplets in order to train the network. Moreover, triplet loss *per se* does not guarantee temporal continuity.

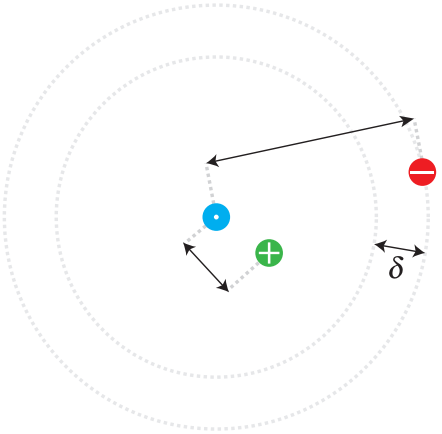
Appropriately sampling the triplets can be a difficult task in the general case, since one usually does not know the best distance to compare two samples. Sampling strategies, also known as "mining"



(a) A triplet of poses. The anchor pose is symbolized in blue, the positive pose in green, and negative pose in red.



(b) Without triplet loss, the three poses representations have close similarity, making them harder to distinguish.



(c) With triplet loss, the representation of the positive pose is closer to the representation of the anchor pose than the negative one. The negative one is at least at a distance δ of the anchor one.

Figure 5.5 – Illustration of the effect of triplet loss on human poses.
Figure inspired and adapted from (Y. Huang et al., 2019).

methods, often involve computing features (angle, loss, ...) over a random subset of candidate vectors in order to elect the most relevant vectors to be fed as input of the triplet loss. Since the training procedure largely relies on hard negative samples, hard negative poses can also be synthesized, e.g. in an adversarial fashion. However, we do not need such complex triplet mining strategies in our case.

Our main goal is to create temporally coherent motion, since poses already are spatially coherent. As such, we only need to sample pose triplets that are similar or dissimilar essentially with regards to time.

We propose a very simple sampling method to that end.

In a training sequence of poses $\mathbf{s} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$, we first extract a random anchor pose $\mathbf{x}^A = \mathbf{x}_{t_A}$ at time step $t = t_A$. The positive pose $\mathbf{x}^{\oplus} = \mathbf{x}_{t_{\oplus}}$ and negative pose $\mathbf{x}^{\ominus} = \mathbf{x}_{t_{\ominus}}$ are then chosen so that $t_{\ominus}$ has a delay with t_A about an order of magnitude greater than the delay between $t_{\oplus}$ and t_A:

$$t_A < t_{\oplus} \ll t_{\ominus} \tag{5.12}$$

In our experiment, we resample pose sequences from the NTU RGB+D dataset so that all sequences have a duration of $T = 100$ time steps. For each pose sequence $\mathbf{s} = \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T$, and each time step $0 \leq t \leq T$, we construct the following triplet:

$$(\mathbf{x}_A, \mathbf{x}_\oplus, \mathbf{x}_\ominus) = (\mathbf{x}_t, \mathbf{x}_{t\pm2}, \mathbf{x}_{t\pm10})$$

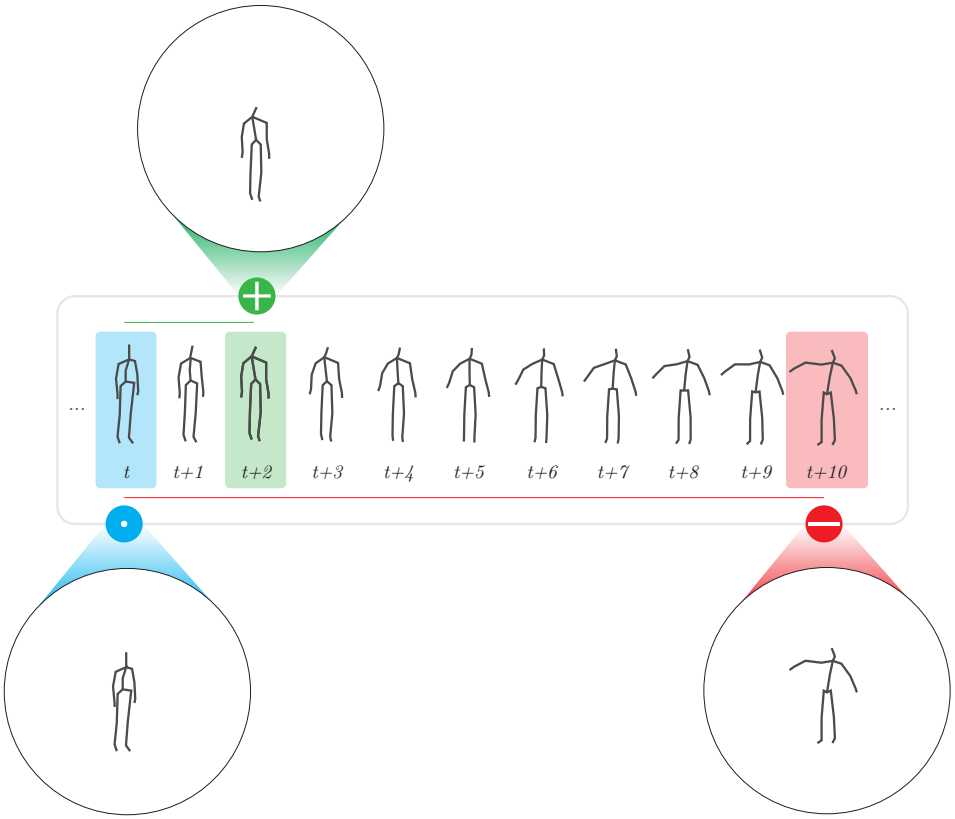


Figure 5.6 – Temporal sampling of the triplet of poses.

An illustration of that (temporal) sampling method is proposed in figure 5.6.

The triplet loss will enforce that two temporally very close (but still distinct) poses will have representations that are more similar than a representation of a later pose.

Given that the by-frame pose denoising encoder-decoder already displays spatial continuity and given that pose representation is already sparse, training the model with the triplet sampling method and the triplet loss enforces temporal continuity.

5.3.11 Generating pose motion with latent semantic space interpolation

Generation procedure

To generate a new pose sequence, two vectors $\mathbf{z}_A$ and $\mathbf{z}_B$ are chosen in the latent space. A T-step linear interpolation of vectors between these two vectors is then performed. Finally, all the latent vectors are decoded individually by the decoder in order to produce the final pose sequence $\mathbf{s} = \mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_T$:

$$\mathbf{y}_t = D\Big(\alpha(t)\mathbf{z}_A + (1 - \alpha(t))\,(\mathbf{z}_B - \mathbf{z}_A)\Big) \quad \forall t \in [\![0, T]\!] \tag{5.13}$$

where $\alpha(t) = \frac{t}{T}$ for $t \in [\![0, T]\!]$ and where D is the decoder neural network.

The two latent vectors $\mathbf{z}_A$ and $\mathbf{z}_B$ used in the procedure can either (1) be sampled, e.g. from a Gaussian distribution, or (2) be encodings of real poses by the encoder neural network E.

Semantics in the latent space

We observe continuity of both the encoder and the decoder neural networks.

Moreover, we observe the emergence of a semantic in the latent space. Individual components of a latent code present a semantic meaning, from a human point of view.

In order to illustrate the semantic meaning of the latent space, we consider a latent code $\mathbf{z}$:

$$\mathbf{z} = \begin{pmatrix} z_1 & z_2 & \cdots & z_{\mathbf{z}_{\text{dim}}} \end{pmatrix}^\top \in \mathbf{R}^{\mathbf{z}_{\text{dim}}}$$

where $\mathbf{z}_{\text{dim}} = 32$ is the dimension of the latent space, and plot the evolution of the generated decoded pose when altering a single component z_i of the vector.

Let $\Gamma(i,u) = \begin{pmatrix} \gamma_1 & \gamma_2 & \cdots & \gamma_{\mathbf{z}_{\text{dim}}} \end{pmatrix}^\top \in \mathbf{R}^{\mathbf{z}_{\text{dim}}}$ design the vector where all the components j are centered except the component $j = i$ whose component is linear with regards to u:

$$\gamma_j(i,u) = \begin{cases} u & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \qquad \forall j \in [\![1, \mathbf{z}_{\text{dim}}]\!]$$

We plot the evolution of $u \mapsto D(\Gamma(i,u))$, for several fixed values of the vector components i in figure 5.7 to explore the latent space. The model architecture used for figure 5.7 has been introduced in section 5.3.4 and features a 32-dimension latent space.

A semantic interpretation for each component is proposed below.

We first observe that when linearly interpolating between two latent codes, the appearance of decoded poses changes continuously. The semantics contained in the pose also change gradually.

The latent representation does not appear to be a fully disentangled representation, since several axes seem able to control the rotation of the body. Moreover, each individual axis appears to control all body joints, and not only one arm of one leg for instance. However, each individual axis presents a semantic meaning: a human interpretation of the evolution of the model's decoded poses from that axis involves more-or-less atomic movements (e.g. "lower an arm" or "cross one's legs") that, together, make up class-specific movements (e.g. "drinking" or "jumping" movements). It is important to note that the model never has access to categorical information (classes) about the sequence it is trained on, at any point. Since no class information can leak to the model, the emergence of such high-level semantic information in the latent space is worth noting.

Even if each individual dimension in the latent space could affect the full body at once, we observe that is often only controls a (semantically related) part of the body.

For instance, in figure 5.7 the dimension $i = 5$ appears to control the arms of the body, and more precisely how much the two arms are crossed over the chest or not. That dimension also controls how much one is sitting down or standing up.

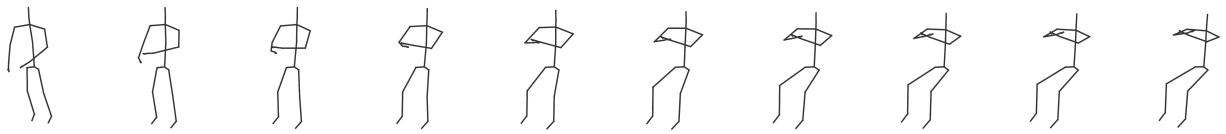
The same act of sitting down and standing up is also controlled by the dimension $i = 25$, but with a

fully different sitting pose: the sitting pose features a left arm open, pointing at something. That arm is progressively opened or closed when traveling in this dimension $i = 25$.

Other dimensions can involve more rotation-focused transformations than translation transformations. This is the case for the dimension $i = 32$ where an progressively rotates from a profile-turned pose to a front-facing pose.

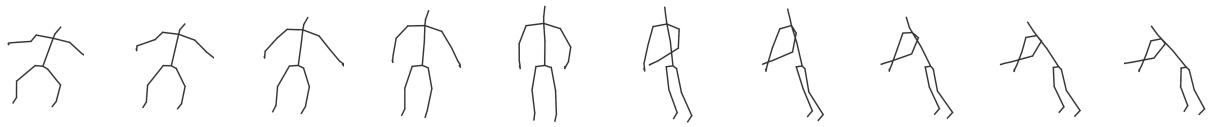
Arguably, one can even consider that higher-lever semantic meaning can appear, as illustrated with the dimension $i = 10$. That dimension appears to control whether the pose is open, active and almost aggressive or if it is closed on itself, in an almost defensive setting.

To better illustrate possible interpolations in the latent space, a visualization based on a (less powerful) model trained with a 2-dimensional latent space only is proposed in figure 5.8.



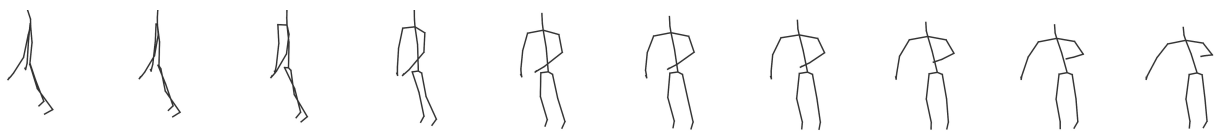
(a) Evolution of $u \mapsto D(\Gamma(i,u))$ from $u = 0$ (left) to $u = 10$ (right) on dimension $i = 5$

(b) Evolution of $u \mapsto D(\Gamma(i,u))$ from $u = -5$ (left) to $u = 10$ (right) on dimension $i = 10$



(c) Evolution of $u \mapsto D(\Gamma(i,u))$ from $u = -10$ (left) to $u = 10$ (right) on dimension $i = 23$

(d) Evolution of $u \mapsto D(\Gamma(i,u))$ from $u = -10$ (left) to $u = 0$ (right) on dimension $i = 25$



(e) Evolution of $u \mapsto D(\Gamma(i,u))$ from $u = -3$ (left) to $u = 4$ (right) on dimension $i = 32$

Figure 5.7 – Illustration of semantics emergence in the latent space

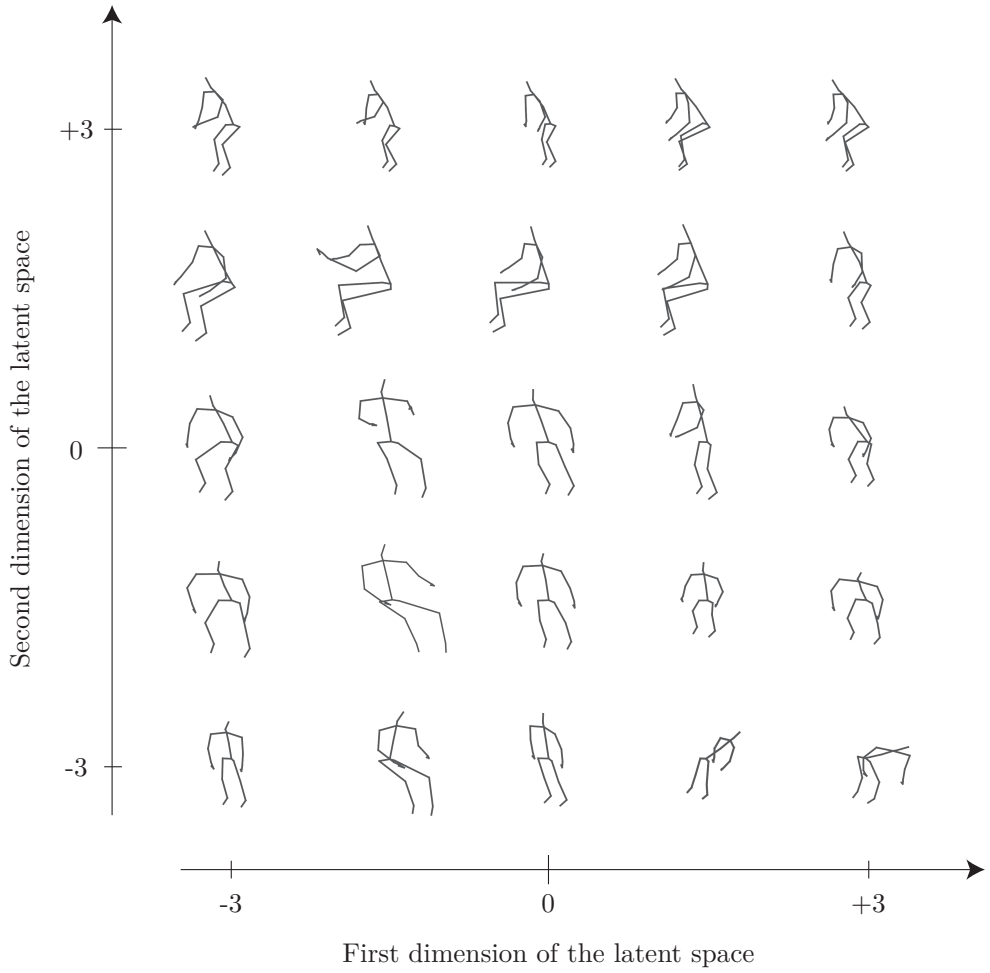


Figure 5.8 – Visualization of decoded static poses for different latent vectors for a model with a $\mathbf{z}_{\text{dim}} = 2$ only dimensional latent space. The latent space illustrated in this figure is not as continuous as our model's latent space with $\mathbf{z}_{\text{dim}} = 32$ since the two dimensional space is too small to accurately represent the complexity of both 3D poses and their temporal evolutions.

5.3.12 Role of individual components

The hyperparameters $\alpha_{\text{reconstruction}}$, α_{anatomy} and α_{triplet} from in the loss function are indicative of the relative importance of each component of our approach. As a reminder, the objective we optimize is:

$$\begin{aligned} \mathscr{L}_{\text{spatiotemporal}} &= \mathscr{L}_{\text{spatial}} + \mathscr{L}_{\text{temporal}} \\ &= \left(\alpha_{\text{reconstruction}}\mathscr{L}_{\text{reconstruction}} + \alpha_{\text{anatomy}}\mathscr{L}_{\text{anatomy}}\right) + \alpha_{\text{triplet}}\mathscr{L}_{\text{triplet}} \end{aligned} \tag{5.14}$$

Note that pose data is normalized as detailed in section 5.3.5 and that the hyperparameters sum to one: $\alpha_{\text{reconstruction}} + \alpha_{\text{anatomy}} + \alpha_{\text{triplet}} = 1$.

While the spatial and temporal aspects are coupled since the network is trained end-to-end, they still are relatively independent. The main effect of the temporal objective is to "order" the spatial representations in a temporally coherent way, as explained below, the remaining spatial objective only being to generate realistic static poses. As such, the neural network's Achilles heel that first needs to be

strengthened is the spatial aspect.

Reconstruction

The value of $\alpha_{\text{reconstruction}} = 0.9995$ highlights the drastically vital importance of pose reconstruction in our approach.

In our experiment, training leads to a reconstruction loss value (alone) of the order of $\mathscr{L}_{\text{reconstruction}} \sim 2 \times 10^{-}2$.

A common problem with pose data sequences stands in the fact that, due to the presence of noise in the training data (since pose data comes form an inherently noisy vision-based estimation method) and the high sensitivity of pose data to noise, evaluating results is hard. Indeed, the perceived visual quality of the reconstruction of a pose does not necessarily always aligns fully with the numerical reconstruction error, even with apparently low reconstruction errors. As such, a qualitative visual inspection of the results is needed besides the quantitative numerical error. This problem is frequently encountered in recurrent-based models from the literature, for instance, although often not explicitly mentioned. Devising a better metric that accounts for this issue is hard and has yet to be investigated, for instance using a noise-robust loss or an adversarial loss.

As such, the numerical value for the reconstruction loss is given for indicative purposes, the hyperparameter value being "validated" afterwards with human visual perception. A value of at least 0.98 for $\alpha_{\text{reconstruction}}$ appears to be required for to be able to reconstruct pose-like data. Conversely, a value of $\alpha_{\text{reconstruction}}$ closer to 1 is at the expense of the other objectives previously mentioned.

Anatomy

While the reconstruction error is sufficient for most frames, some rare edge-case generated poses are not realistic. To alleviate this issue without necessarily having to resort to an adversarial loss, an anatomy loss is used. The anatomy loss measures the cumulative length of the generated skeleton's bones and compares it with the cumulative length of the original skeleton's bones.

To the best of our knowledge, the anatomy loss we use has first been introduced in the 3D human pose estimation domain by (Ramakrishna et al., 2012). Other very similar anatomic constraints can also be found in (R. Li et al., 2019) or in (Chunyu Wang et al., 2014) for instance.

Using the anatomy loss resolves the immense majority of the previous edge-case issues.

It also has the advantage (and disadvantage) of not being an adversarial loss: with an adversarial loss, the adversarial discriminator network would easily spot the difference between noisy and non-noisy poses. A simple anatomy loss leads to almost comparable results, while allowing for denoising.

The value of $\alpha_{\text{anatomy}} = 0.0002$ we use is conservative. Halving it still helps limit edge-cases issues.

In our experiment, since most frames already are well reconstructed, training leads to an anatomy loss value (alone) of the order of $\mathscr{L}_{\text{anatomy}} \sim 1 \times 10^{-}1$ only (in the worst cases).

Triplet

Triplet loss is crucial for temporal continuity and coherence of the generated sequences, in our approach.

Triplet loss does not improve the reconstruction *per se*. Moreover, training the neural network without it already leads to a mostly smooth and continuous latent space.

However, the proposed sequence generation procedure (i.e. to decode a linear interpolation of vectors from the latent space to the output pose space) does not lead to temporally realistic generated sequences if no triplet loss is enforced. Locally, two close latent vectors will have close decoded poses, but on a wider temporal scale, the sequence lacks of temporal coherence when no triplet loss is used. While one could think of using recurrent neural networks or convolutional neural networks to learn a path in the latent space in order to find out which latent vectors to generate and decode, that path would be non-linear and complex. The introduction of the triplet loss allows to circumvent and even iron out that apparent issue.

One can view the triplet loss as a way to "sort" the latent space in a coherent way. It brings similar vectors closer to each other while ensuring a minimal distance between dissimilar vectors. As such, triplet loss sorts the latent space and makes each "individual" vector in the latent space more easily distinguishable. The sorting is temporal-related, since we sample the triplets in a temporally coherent strategy during sampling.

As with the reconstruction, this involves visual and partial human judgment. Due to the "sorting" effect of the triplet loss, a formal evaluation of the minimal hyperparameter value required for temporal realism would require to introduce an adversarial loss between real and generated pose sequences. In practice, however, a temporal triplet loss only -without temporal adversarial loss- reveals to be sufficient.

We observe that a value of (at least) $\alpha_{\text{triplet}} = 0.0003$ is sufficient for full temporal coherence of the generated sequences.

In our experiment, training leads to a triplet loss value (alone) of the order of $\mathscr{L}_{\text{triplet}} \sim 7 \times 10^{-}1$.

5.3.13 Conclusion

We have proposed a novel approach for human motion pose sequence generation with a self-supervised model. The approach mostly decouples the temporal and the spatial aspect of the sequence synthesis.

Our approach uses a per-frame denoising encoder-decoder neural network to deal with spatial aspects of human poses. A temporal triplet loss with a simple time-contrastive sampling method is responsible for the temporal continuity of the generated sequences.

The latent space of the encoder-decoder model displays very interesting semantic meaning properties which would be worth investigating in a future work.

To generate new pose sequences, a linear interpolation in the encoder-decoder latent is performed, each interpolated vector being decoded by the decoder module to generate a pose. At a frame level, we observe that generated denoised poses often look more realistic to the human eye than ground truth data, because of the frequently high level of noise in vision-estimated pose data. At a sequence level, we observe that sequences are both (1) temporally coherent and (2) credible. This is notable for a model that requires neither an adversarial loss nor a variational sampling.

The overall robustness and ease-of-training of the whole approach opens a wide range of new possible extensions. For instance, generating a gesture of a given class or of a given style can be done by learning a path in the denoising decoder-encoder latent space. Immediate possible extensions include

a class-conditioned model, to choose a gesture to generate, and a frame-adversarial loss that would replace the total bones' length loss. Finally, coupling the model with other multimodal deep learning models (e.g. image-based, text-based or audio-based) could remove the need for them to be conditioned on pre-existing pose.

5.4 Spatio-Temporal Generative Adversarial Networks

In this section, we propose an approach to generate sequences of human poses based on a Conditional Generative Adversarial Network.

5.4.1 Approach summary

To represent human pose sequences, a 2D-convolution-ready representation format of the sequences is first adopted. Human pose sequences are converted to a 2D image-like spatio-temporal continuous representation based on a spatial joint reordering trick (Baradel et al., 2018; J. Liu et al., 2016) and called Tree Structure Skeleton Image (TSSI) (Z. Yang et al., 2018), in order to preserve both spatial and temporal relationships. The TSSI format is a very immediate (time $\times$ repeated and re-indexed joint $\times$ dimension) tensor representation where the joints are repeated and re-indexed, the resulting tensor being viewed as an image for visualization purposes. TSSIs are described in more details in section 5.4.4.

A conditional generative adversarial network (CGAN) approach is proposed for human pose sequence generation. The CGAN is composed of a generator neural network and a discriminator neural network. The two neural networks use the TSSI format, both for their input and for their output. The architectures of the generator and the discriminator neural networks are 2D convolutional neural networks architectures with convolutions that process information from both space and time domains. The two networks are conditioned on action classes (e.g. "drinking water", "jumping" or "waving hands"). They are trained in an adversarial fashion on ground truth human pose sequences represented with the TSSI format.

After the training is complete, the generator network can generate new sequences. Since the sequences coming out of the generator are represented in the TSSI format, a last and almost immediate step consists in transforming back the generated sequences into their canonical representation: a (time $\times$ joint $\times$ dimension) tensor representation.

5.4.2 Contributions summary

The most natural and immediate way to represent a pose sequence, regardless of the nature of the keypoints -such as body joints, hand joints or facial landmarks- is to use a 3D tensor representation of shape (T, J, d) where the three axes of the tensor represent and are indexed by time (T timesteps), by keypoints (J keypoints), and by keypoints' dimensions (d keypoints' dimensions). As mentioned later in section 5.4.1, the representation we use is a close, but slightly different, representation introduced in (Z. Yang et al., 2018) and based on a spatial body joint reordering trick introduced in (J. Liu et al., 2016).

Besides the overall approach and (hyper-)parameters, our main contribution more specifically is the conditional generative neural network architecture introduced in section 5.4.5.

5.4.3 Model intuition and Motivation

Generative models can broadly be split into three main categories: Generative Adversarial Networks (GANs), Variational Auto-Encoders (VAEs) and exact likelihood models like flow-based models[4]. A flow-based generative model is constructed by composing a series of invertible transformations and maps observed data to a standard Gaussian latent variable. The model explicitly learns the data distribution. Generating new data with a flow-based generative model is as simple as sampling a vector in the latent space, and computing its inverse transformation back to the original domain. Despite their computational efficiency, flow-based models tend to have worse density modeling performance compared to state-of-the-art models, to this date (Ho et al., 2019). While neither VAEs nor GANs explicitly learn the probability density function of real data, both of these approach families have shown good results in generative tasks in numerous domains, including images, spoken and written human language and music. VAEs are directed probabilistic graphical models whose posterior is approximated by a neural network, forming an autoencoder-like architecture. Generating new data with a VAE model is as simple as sampling a vector in the VAE's latent space -the prior over the latent variables usually being set to be the centered isotropic multivariate Gaussian- and decoding it back to the original domain, in a similar fashion to the generation procedure we use in our temporal triplet auto-encoder proposed in section 5.3. However, combining temporal information with a VAE is not straightforward, and approaches found in the literature -e.g. combining a VAE and a RNN put before, "inside" or after the VAE- do not produce realistic results for human pose sequence generation tasks (Chung et al., 2015).

GANs are models where two neural networks are trained in an adversarial fashion. They will be introduced more formally in section 5.4.5. GANs are known to produce very realistic results -especially in the image domain-, compared to VAEs, and benefit from a wide interest in the research community. As such, our idea is to leverage the power of GANs to generate human pose sequences.

How to apply GANs to skeletal data? An option is to decouple temporal and spatial information, using a 1D-CNN GAN architecture. However, this tends to produce individually temporally coherent pose sequences, but with very limited spatial coherence. Another option consists in representing sequences as spatio-temporal graphs, and train a generator on these graphs using adversarial learning, like proposed in (S. Yan et al., 2019). While that option is very encouraging, the resulting generated human pose sequences seem to be noisy. However, analyzing failure cases of such graph-based models is difficult because of the relative novelty of graph-based deep learning tools: the audit and interpretation of graph-based models is more difficult than the audit and interpretation of image-based models.

We propose to use image-based GANs for human pose sequences generation.

We observe that human pose sequences in a d-dimensional physical space[5] can be viewed as images[6], since images are 3-dimensional tensors of shape $(\text{Height} \times \text{Width} \times \text{Channels})$ and sequences

[4]Flow-based generative models include models like Glow (D. P. Kingma et al., 2018), for instance.

[5]Human pose sequences usually have $d = 3$ dimensions in the real world, or $d = 2$ if captured from a single point of view.

[6]Image usually have Channels $= 3$ channels for RGB images, or Channels $= 1$ channel for gray-scale images, for

are 3-dimensional tensors too, of shape $(\text{Duration} \times \text{Joints} \times d)$. However, representing a sequence as an image presents a major drawback: the image axis that represent the joints ordering (spatial axis) has no continuity, since two successively-indexed joints can be anywhere in the body, regardless of its structure. As such, this representation cannot serve as the input of a convolutional neural network, even if convolutional neural architectures are very commonly used in GANs.

To overcome this issue, raw-image representations are transformed into redundant image representations called TSSIs (J. Liu et al., 2016; Z. Yang et al., 2018) where columns are reordered and duplicated so that data can be continuous along both the time axis and the spatial axis, rather than only along the time axis as in the original raw-image format. The two image-representations are described in section 5.4.4.

A GAN model is trained to generate human pose sequences in the TSSI format. To be able to control the generated gesture, the GAN is transformed into a Conditional GAN (CGAN) conditioned with a sequence class (e.g. "drinking", "jumping", "taking a selfie", ...).

Finally, the redundant columns in the generated human pose sequences TSSI representation are averaged, in order to obtain the desired and realistic human pose sequences.

5.4.4 Format of the input poses

Sequences can be viewed as images

Let a human pose sequence $\mathbf{s} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T) \in \mathbf{R}^{T \times J \times d}$, where T is the sequence duration, J is the count of joints in the body's structure, and d is the dimension (count of components) of each joint.

Since a sequence $\mathbf{s}$ is represented with a 3-dimensional (T, J, d)-shaped tensor, one can see that a sequence can almost immediately be seen as a multi-channel image, since a multi-channel image is also a (visual) representation of 3-dimensional tensor.

Compared to RGB images whose values always are in a fixed range (e.g. 0-255), sequences values could be unbounded. In practice however, on a fixed temporal horizon T, human motion is bounded due to spatial and muscular limits of the human body. We normalize all the sequences so that they are centered with unitary variance. While this is not mandatory, it has two benefits: the first one is that they will be ready for neural networks whose weights initialization (Kaiming initialization) expects such normalized input, and the second one is that they can be plotted like RGB images.

An analogy with the RGB image format is proposed in table 5.3 and a visual illustration of the different representations is proposed in figure 5.10.

For the sake of clarity, and more especially to distinguish them from the TSSI format that is described in the next subsection, we chose to refer to the normalized sequences representation as the "raw-image representation" or as "vanilla-image representation", since they can be plotted: each pixel in the image is indexed by three values representing the spatial and temporal locations of that "pixel" (value). At this stage, no information has been lost, and no information is redundant, compared to the original tensor. As such, the vanilla-image representation is still a (T, J, d)-shaped tensor, but centred and normalized.

instance.

Table 5.3 – Analogy between a vanilla sequence format and the RGB image format

Image	RGB image	Sequence
Axes	3	3
First axis	Height (H)	Time (T)
Second axis	Width (W)	Space: Joints (J)
Third axis	Channels ($C = 3$)	Space: Joints' dimensions (d)
"Continuity"	Along the H and W axes	Along the T axis. Not along the J axis.
Values	Bounded (0-255 range)	Unbounded

Ensuring spatial continuity of the images

However, the vanilla-image representation presents a major drawback: the spatial joint J axis that represents the joints ordering has no spatial "continuity", since two successively-indexed joints can be anywhere in the body, regardless of its structure.

For instance, on figure 5.11 which represents a human body, the joint #16 -left toe- has two neighbors in terms of indexing: the joint #15 -left heel- and the joint #17 -right side of the pelvis-, but has only one neighbors in terms of spatial structure, since the joint #17 is not connected by any bone to the joint #16, whereas joints #16 and #15 are connected by a bone.

To ensure a spatial continuity we use the "tree traversal over the spatial steps" spatial ordering described in (J. Liu et al., 2016). Using that spatial ordering, the tensor representation immediately becomes what (Z. Yang et al., 2018) refer to as the "Tree Structure Skeleton Image (TSSI) representation". We thus also refer to that tensor representation as a "Tree Structure Skeleton Image (TSSI) representation".

Even earlier than (Z. Yang et al., 2018), the authors of (Baradel et al., 2018) also propose to repeat and re-index the body's joints based on the spatial ordering described in (J. Liu et al., 2016): moreover they also make use of a similar 3D tensor representation. However their 3D tensor representation is different from the TSSI (3D tensor) representation. A comparison between the 3D tensor TSSI representation and the 3D tensor representation used in (Baradel et al., 2018) can be found below at the end of the current section.

In (Baradel et al., 2018; J. Liu et al., 2016; Z. Yang et al., 2018), the direct concatenation of joints $\#1 \rightarrow \#2 \rightarrow \#3 \rightarrow \ldots \rightarrow \#J$ is replaced by a depth-first tree traversal order of the body's skeletal graph structure, which happens to be a tree whose root is the join #2 -belly-. That traversal order has pros and cons. The immediate benefit is that two neighbors in terms of indexing become neighbors in terms of spatial relationship. A minor drawback of the method is that the information partly becomes redundant, since columns are repeated twice or more. A traversal order that respects a spatial continuity of the joints is illustrated in figure 5.11, the traversal order being indicated by a red path with arrows and dots.

Following (Baradel et al., 2018; J. Liu et al., 2016; Z. Yang et al., 2018) we replace the vanilla concatenated traversal order of the joints $\#1 \rightarrow \#2 \rightarrow \#3 \rightarrow \ldots \rightarrow \#J$ by a depth-first tree traversal order of the joints: $\#2 \rightarrow \#21 \rightarrow \#9 \rightarrow \ldots \rightarrow \#17 \rightarrow \#1 \rightarrow \#2$. Once the traversal order is established, columns from the original vanilla-image are simply reordered and duplicated when necessary so that

the traversal order is correct to form the TSSI, as illustrated in figure 5.10 (c) and in figure 5.12 (d). As such, the TSSI representation is a $(T \times RJ \times d)$-shaped tensor, where RJ is the length of the depth-first tree traversal ordering of the joints. A few (ground truth) sequences in the TSSI-format are plotted in figure 5.9 for illustration.

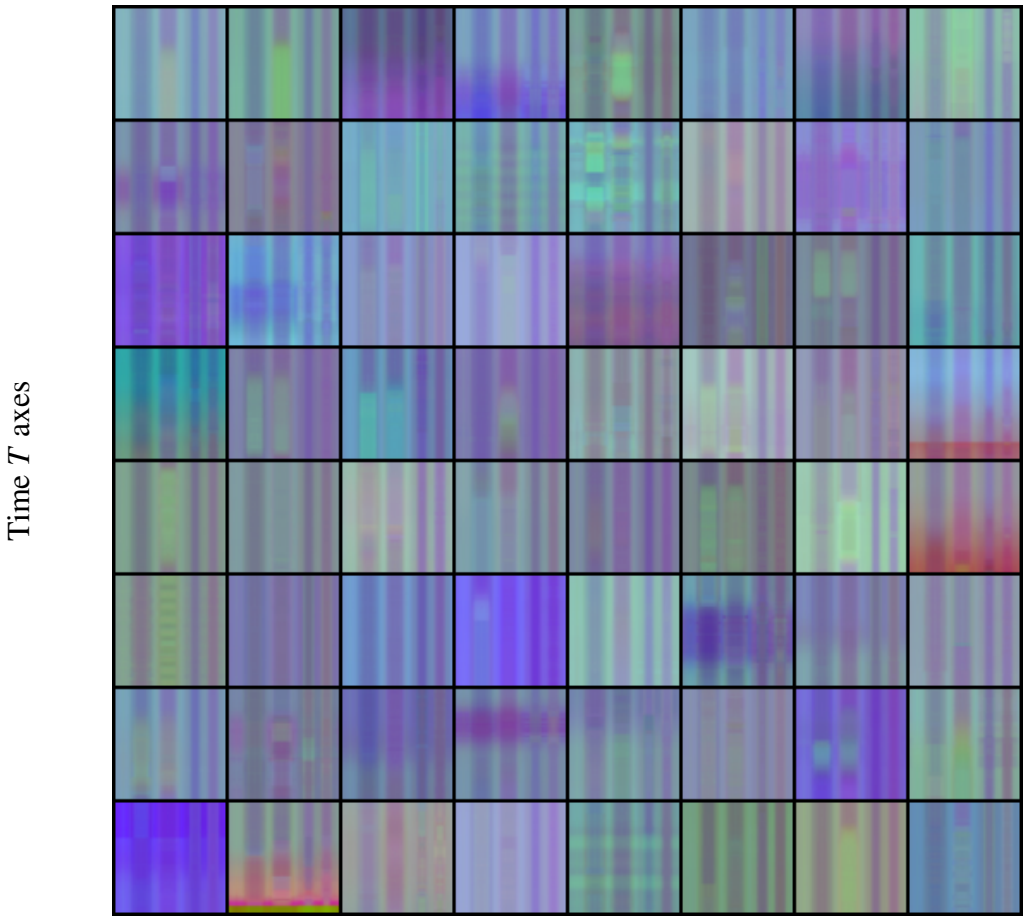


Figure 5.9 – 64 different *ground-truth* sequences represented in a TSSI format. The horizontal axis of each TSSI sequence is the joint J axis. The vertical axis of each TSSI sequence is the time T axis. The *xyz* dimensions are mapped to RGB channels for visualization.

In (Z. Yang et al., 2018), the authors use a 3D tensor representation "TSSI" of shape (T, RJ, d)-shaped tensor, where T is the duration of the sequence, d is the joints' dimension and RJ is the length of the depth-first tree traversal ordering of the joints. In (Baradel et al., 2018), however, a fourth axis is considered in order to incorporate derivates of the represented signals, namely: the raw signals, their first-order derivates (i.e. velocities) and their second-order derivates (i.e. accelerations). As such, a sequence could be represented as a 4D tensor representation of shape (T, RJ, d, Δ)-shaped tensor, where T is the duration of the sequence, d is the joints' dimension, RJ is the length of the depth-first tree traversal ordering of the joints, and $\Delta = 3$ is the derivatives count. The authors introduce a deep learning model for human activity recognition based on 3D convolutional neural networks. To that end, they value a 3D tensor representation over the 4D tensor representation previously mentioned: the dimensions (d) axis is concatenated along the joints (RJ) axis to form a channels axis. As such, they

use a 3D tensor representation, whose shape is (T, C, Δ)-shaped tensor, where T is the duration of the sequence, d is the joints' dimension, RJ is the length of the depth-first tree traversal ordering of the joints, $C = RJ \times d$ is the channels count and $\Delta = 3$ is the derivatives count. Both the TSSI representation from (Z. Yang et al., 2018) and the 3D tensor representation from (Baradel et al., 2018) have 3 axes, but -even if they are close to each other- the semantics of the two representations are different.

It is worth noting that (Baradel et al., 2018) perform experiments related to the topological joint order proposed in (J. Liu et al., 2016) and used by both (Baradel et al., 2018) and (Z. Yang et al., 2018). In the experiments by (Baradel et al., 2018), three orderings are considered: the topological order, a random joint order and topological order without double entries. On the human activity recognition task used for evaluation, their model displays a better recognition accuracy when using the topological order than when using a random order, with an in-between accuracy being obtained when the topological order does not feature duplicated joints. This result highlights the importance of the repeated entries in the joint traversal order introduced in (J. Liu et al., 2016): they ensure that the traversal order preserves neighbourhood relationships.

Converting back the images to sequences

Since columns in TSSI formatted-sequences are redundant, averaging columns that represent the same joint and sorting them (based on the joints' indices) is sufficient to go back to original vanilla-image representations. Vanilla-image representations are equivalent to the original sequence tensor represention, up to an optional (de)normalization step.

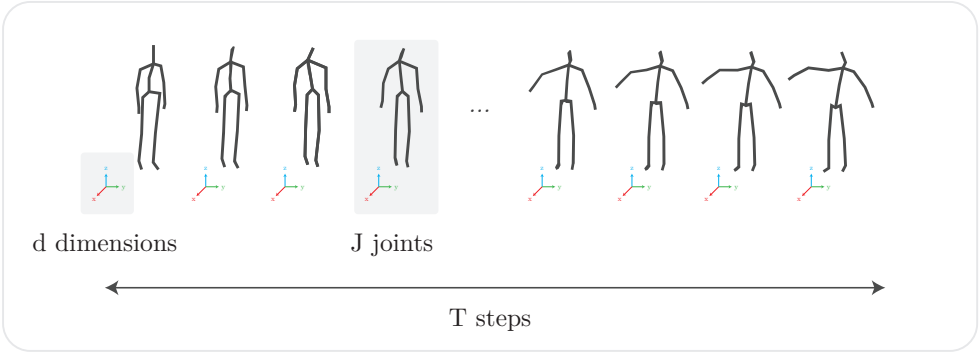
5.4.5 Spatio-Temporal Conditional Generative Adversarial Neural Network (STC-GAN)

Approach presentation

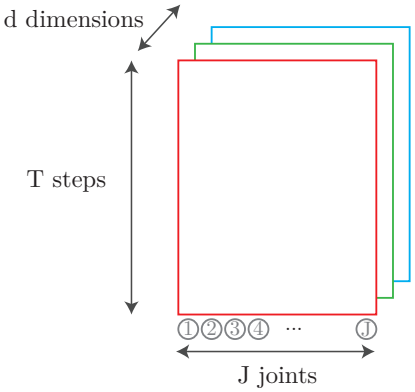
Now that sequences are represented as spatiotemporal images where all images axes are continuous, we propose a Generative Adversarial Network (GAN) approach to generate sequences. The complete process is illustrated in figure 5.12.

Generative adversarial networks (GANs) (Goodfellow et al., 2014) are a framework for the estimation of generative models via an adversarial training procedure in which two models, a discriminator D and a generator G, are trained simultaneously and contest with each other in a game theory scenario. The generator (G in all of our figures), generates candidates while the discriminator (D in all of our figures), evaluates them. GANs rely upon the idea that discriminative models can improve generative model performances: if a data generator is good, one should not be able to tell generated data apart from real ground truth data.

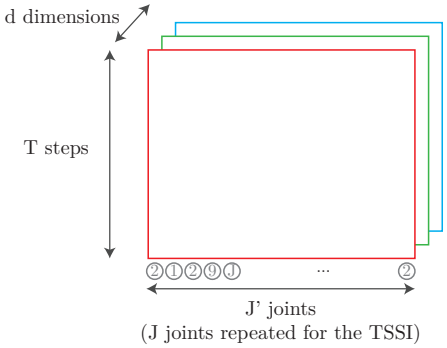
The two networks are trained simultaneously with competing objectives. The discriminator neural network is a binary classifier whose goal is to correctly distinguish if its input is sampled from the ground truth distribution of real samples, or if its input has been generated by the generator neural network. Conversely, since we want the generator to generate data so realistic that it could have been sampled from the ground truth distribution of real samples, the generator is trained to fool the discriminator.



(a) **Projected view of a sequence.** A sequence of poses can be represented by a tensor **s** whose shape is (T, J, d) where T is the sequence duration, J the number of joints in the skeleton and d the dimensions ($d = 3$ for 3D (x, y, z) gestures).



(b) **Raw sequence as an image.** The same sequence **s** can equivalently be represented as an image with d channels where the height of the image is the sequence duration T and the width of the image is the joints' count J. When $d = 3$, the three x, y and z channels can be stored in the RGB format.



(c) **Sequence as a Tree Structure Skeleton Image (TSSI). (J. Liu et al., 2016; Z. Yang et al., 2018)** To ensure that neighboring columns in the image representation are spatially related in the original skeleton structure, joints (i.e. columns) are reordered and repeated when needed. The joint arrangement comes from a depth-first tree traversal order of the skeletal (tree) structure.

Figure 5.10 – Illustration of different human pose sequence representations.

The generator learns a mapping from a prior noise distribution $p_{\mathbf{z}}$ to a data space. As such, generating new data is performed by sampling a vector from that distribution $\mathbf{z} \sim p_{\mathbf{z}}$ and feeding it as the input of the generator network, as illustrated in figure 5.12 (c). The noise distribution we choose is the standard Gaussian distribution.

Formally, let G be a neural network, D a neural network which outputs a single scalar in the $[0, 1]$ interval, $p_{\text{ground truth data}}$ be the ground truth data distribution, $p_{\mathbf{z}}$ be the prior noise distribution. The objective $\mathscr{L}_{\text{GAN}}$ of a GAN is defined in equation 5.15 and equation 5.16.

$$\mathscr{L}_{\text{GAN}}(G, D) = \mathbb{E}_{\mathbf{x} \sim p_{\text{ground truth data}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))] \tag{5.15}$$

The GAN is trained in an adversarial fashion, such as the two networks G and D compete against

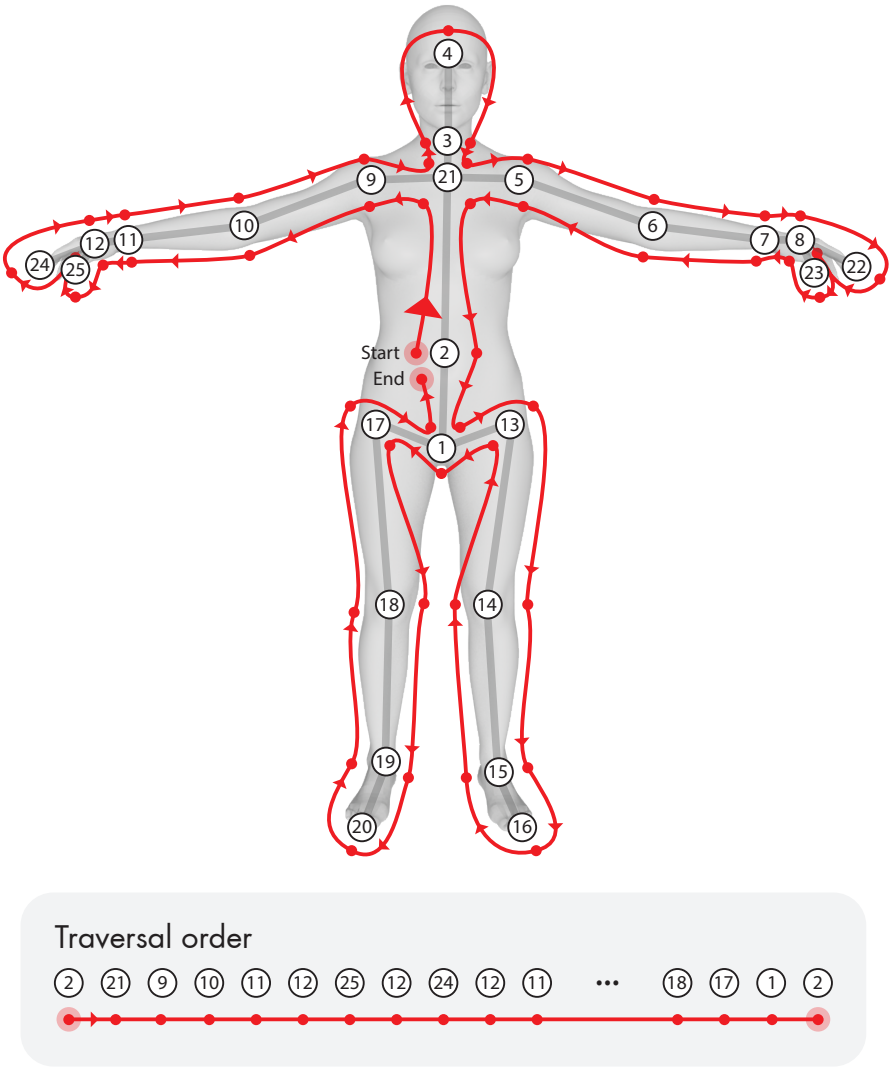


Figure 5.11 – Illustration of a traversal order that respects spatial relations.
Figure reproduced and adapted from (Baradel et al., 2018; J. Liu et al., 2016; Z. Yang et al., 2018).

each other, playing the following zero-sum min-max game:

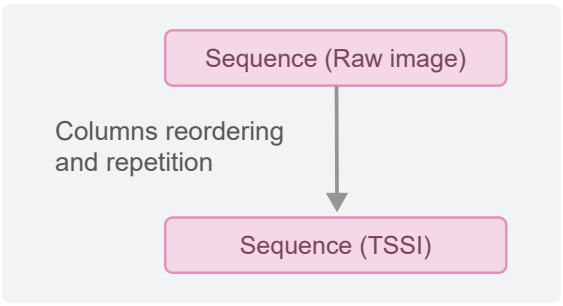
$$\min_G \max_D \mathscr{L}_{\text{GAN}}(G,D) \tag{5.16}$$

GANs can be extended to a conditional generative adversarial neural network (CGAN) model by conditioning either G, or D, or both, on a conditional information $\mathbf{c}$. When both G and D are conditioned by $\mathbf{c}$, the objective $\mathscr{L}_{\text{CGAN}}$ of the CGAN becomes:

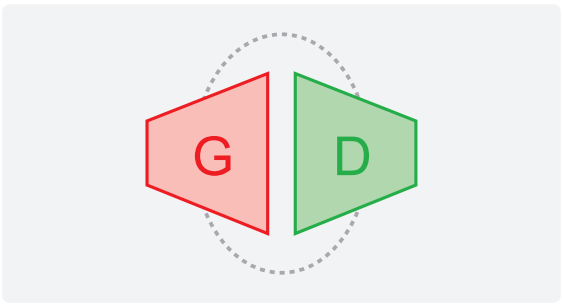
$$\mathscr{L}_{\text{CGAN}}(G,D) = \mathbb{E}_{\mathbf{x}\sim p_{\text{ground truth data}}(\mathbf{x}|\mathbf{c})}[\log D(\mathbf{x}|\mathbf{c})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z}|\mathbf{c})))] \tag{5.17}$$

where the two networks play the same zero-sum game as described earlier in equation 5.16. The training of the two networks is performed jointly.

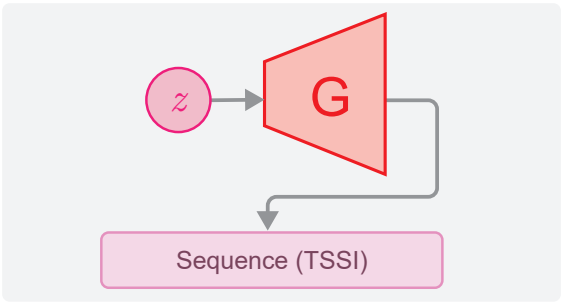
We propose a spatiotemporal conditional generative adversarial neural network (STCGAN) approach for human pose sequence generation. The STCGAN is trained on spatiotemporal TSSI-formatted human pose sequences, and conditioned on classes of actions (e.g. "drinking", "opening bottle" or "fold paper").



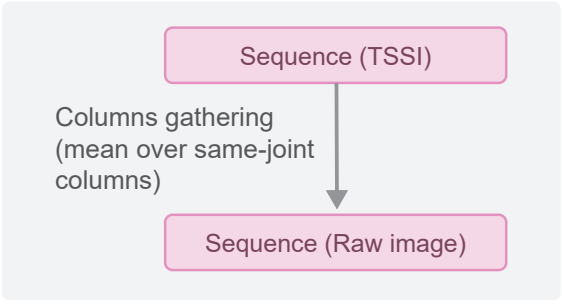
(a) Ground Truth Sequence Format Preparation



(b) GAN Training (with Ground Truth Sequences)



(c) Sequence Generation



(d) Generated Sequence Format Adaptation

Figure 5.12 – Overview of the GAN Pose Sequence Generation Pipeline

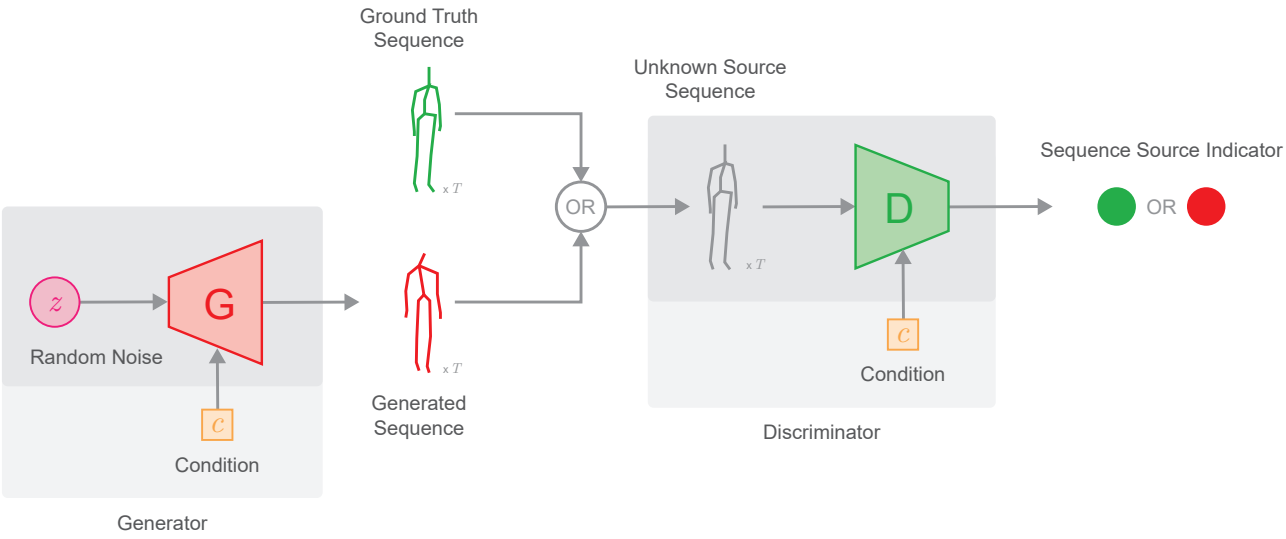


Figure 5.13 – Overview of our Generative Adversarial Neural Network for Pose Sequences. Note that all sequences (ground truth and generated) are represented using TSSIs that are not shown in the illustration. The class-conditioned vector is only used for the CGAN model.

An illustration of the principle of our STCGAN is proposed in figure 5.13.

One-hot actions

In our experiments, we use the previously mentioned NTU RGB+D dataset, described 5.3.4, but transform the sequences into the TSSI format. The dataset features 60 different types of actions.

To condition the sequences, we encode classes with one-hot encoding. The one-hot encoded vector is then repeated over a 2D grid to match the time and joint (i.e. width and height) dimensions of the input of the neural network it is given to.

Models architectures

TSSI-formatted sequences are spatiotemporal image representations that are continuous in both the temporal T dimension and the spatial J dimension. As such, we choose to use a deep convolutional generative adversarial network (DCGAN) architecture for the generator and for the discriminator of our STCGAN, as convolutional layers are known to be effective at filtering patterns in "continuous" grid-like data. Two-dimensional convolutions allow the generator (resp. discriminator) network to generate (resp. recognize) spatially and temporally coherent patterns.

The architecture of the generator is detailed in table 5.4, while the architecture of the discriminator is detailed in table 5.5.

In the generator, a first branch processes a noise vector $\mathbf{z}$ sampled from a standard Gaussian distribution, while another branch processes a class condition vector $\mathbf{c}$. The two resulting outputs are then merged by concatenation (along the channel axis, keeping the time and joint dimensions untouched). Finally, the concatenated vector is sequentially processed by the different blocks of the generator to generate a new TSSI-formatted sequence.

Except for the concatenation step, all generator blocks are made of a 2D bilinear upsampling layer, followed by a 2D convolution, batch normalization, and a ReLU activation. The last layer's activation function is tanh.

In the discriminator, the candidate vector $\mathbf{x}$ is immediately concatenated with the one-hot encoded condition vector $\mathbf{c}$ along the dimension (i.e. channels) axis. The resulting output is sequentially processed by the different blocks of the discriminator.

All discriminator blocks are made of a 2D convolution followed by a batch normalization, and a Leaky ReLU activation with a 0.2 negative slope. The last layer's activation function is a sigmoid.

5.4.6 Experimental results

We train the proposed STCGAN model on TSSI-formatted sequences and their respective associated classes, for 100 epochs. After the training, we generate new TSSI-sequences using the generator. Examples of generated TSSI-formatted sequences are plotted in figure 5.14.

Finally, the TSSI are converted back to vanilla image-representations that we plot in 2D (image-view) and 3D (projected-pose-view).

Visual comparisons between the quality and the variety of the generated sequences and the ground truth sequences is proposed in figure 5.15 for the "hair brushing" action, in figure 5.16 for the "kicking something" action, in figure 5.17 for the "clapping" action, and in figure 5.18 for the "checking time on

Table 5.4 – Generator Architecture

Generator								
# → #	2D Upsample + 2D Convolution						BatchNorm	Activation
	in. channels	out. channels	kernel size	stride	padding	bias		
$\mathbf{z} \to 1$	$\mathbf{z}_{\text{dim}} = 100$	256	4	1	0	No	Yes	ReLU
$\mathbf{c} \to 2$	$\mathbf{c}_{\text{dim}} = 60$	256	4	1	0	No	Yes	ReLU
$1,2 \to 3$	Concatenation (channels axis)							
$3 \to 4$	512=256+256	256	4	2	1	No	Yes	ReLU
$4 \to 5$	256	128	4	2	1	No	Yes	ReLU
$5 \to 6$	128	64	4	2	1	No	Yes	ReLU
$6 \to 7$	64	$d = 3$	4	2	1	No	No	Tanh

Table 5.5 – Discriminator Architecture

Discriminator								
# → #	2D Convolution						BatchNorm	Activation
	in. channels	out. channels	kernel size	stride	padding	bias		
$\mathbf{x}, \mathbf{c} \to 1$	Concatenation (channels axis)							
$1 \to 2$	$63 = d + \mathbf{c}_{\text{dim}}$	64	4	2	1	No	No	LeakyReLU(0.2)
$2 \to 3$	64	128	4	2	1	No	Yes	LeakyReLU(0.2)
$3 \to 4$	128	256	4	2	1	No	Yes	LeakyReLU(0.2)
$4 \to 5$	256	512	4	2	1	No	Yes	LeakyReLU(0.2)
$5 \to 6$	512	1	4	1	0	No	No	Sigmoid

a watch" action.

5.4.7 Conclusion

We have proposed a novel approach for human motion pose sequence generation using generative adversarial networks over images.

Sequences are represented as a 2D spatiotemporal grid with several dimensions, in an image-like fashion: to represent human motion pose sequences, we borrow a representation called TSSI (Z. Yang et al., 2018) that ensures a spatial and a temporal continuity of the data, based on a spatial joint reordering trick (Baradel et al., 2018; J. Liu et al., 2016). As such, any GAN architecture that can digest images may be used to generate sequences. We call Spatio-Temporal Conditional Generative Adversarial Network (STCGAN) the CGANs networks that use the TSSI image format. Since in a TSSI, the image axes are the time and the joint (space) axes, STCGANs directly have access to space and time in a coupled way rather than separately. A simple STCGAN architecture is proposed, and trained in an experiment. To generate human pose sequences, a random vector is sampled from a standard Gaussian noise prior distribution and mapped with the generator neural network to a TSSI-formatted sequence, which is later converted back to a human pose sequence with a vanilla format.

The approach mostly couples the temporal and the spatial aspect of the sequence synthesis.

Immediate possible extensions include: better conditioning the GAN, harnessing the power of the numerous GAN models from the very rich and extensive literature about image-based GANs -e.g. by

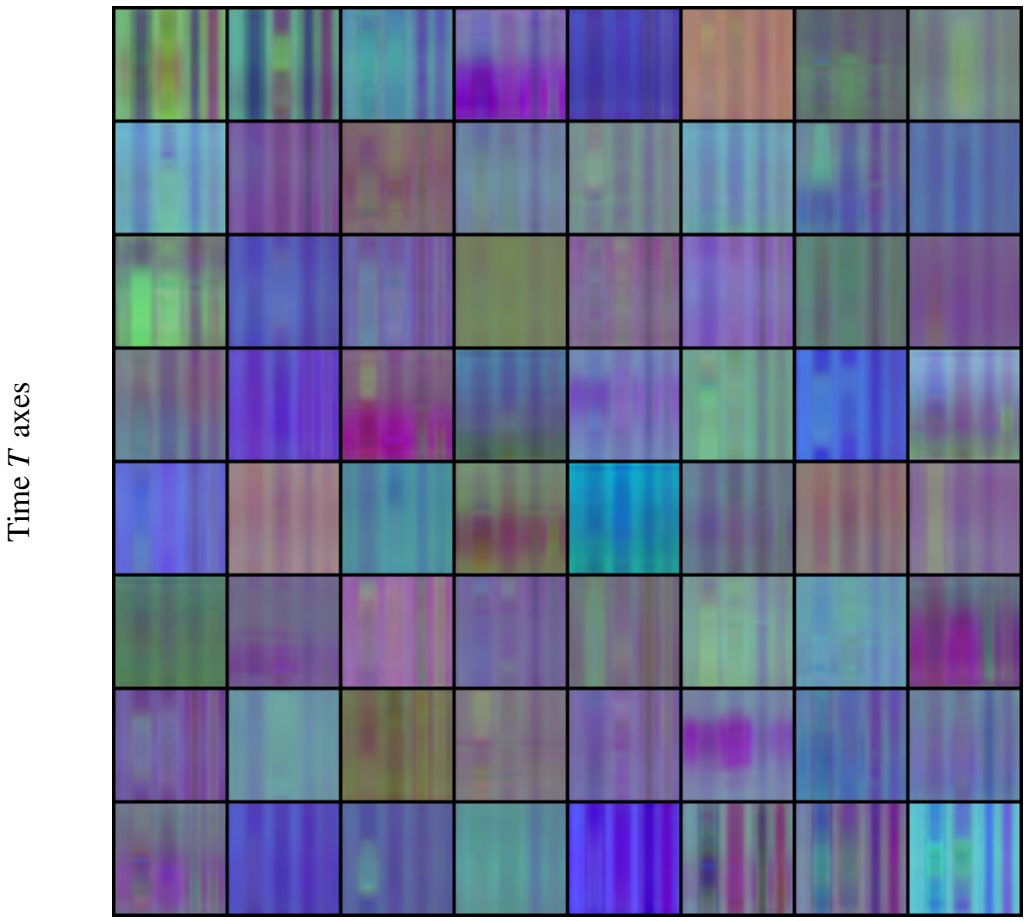
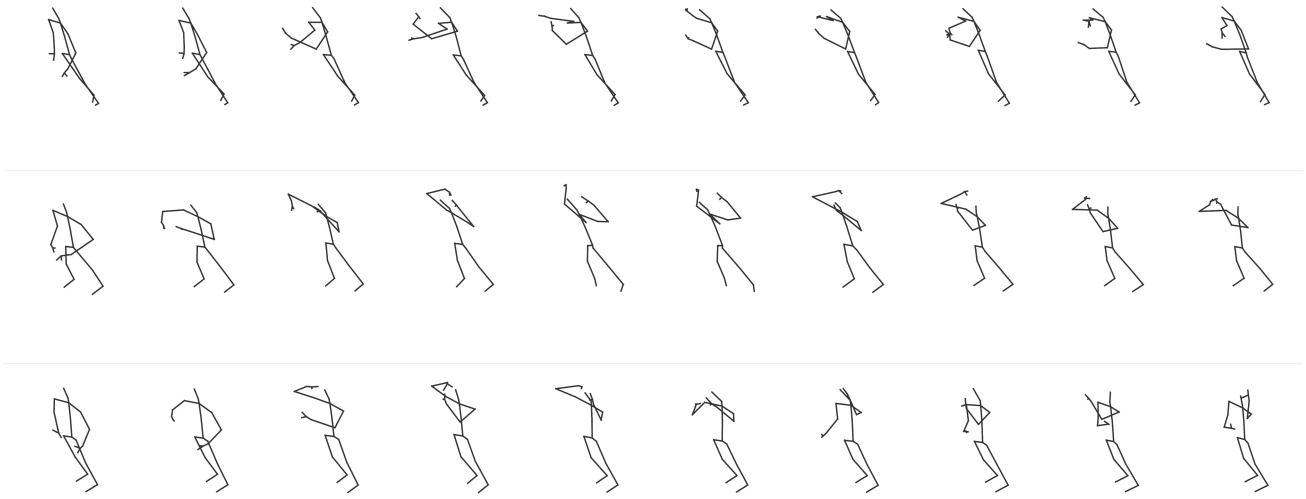


Figure 5.14 – 64 different *generated* sequences represented in a TSSI format. The horizontal axis of each TSSI sequence is the joint J axis. The vertical axis of each TSSI sequence is the time T axis. The *xyz* dimensions are mapped to RGB channels for visualization.

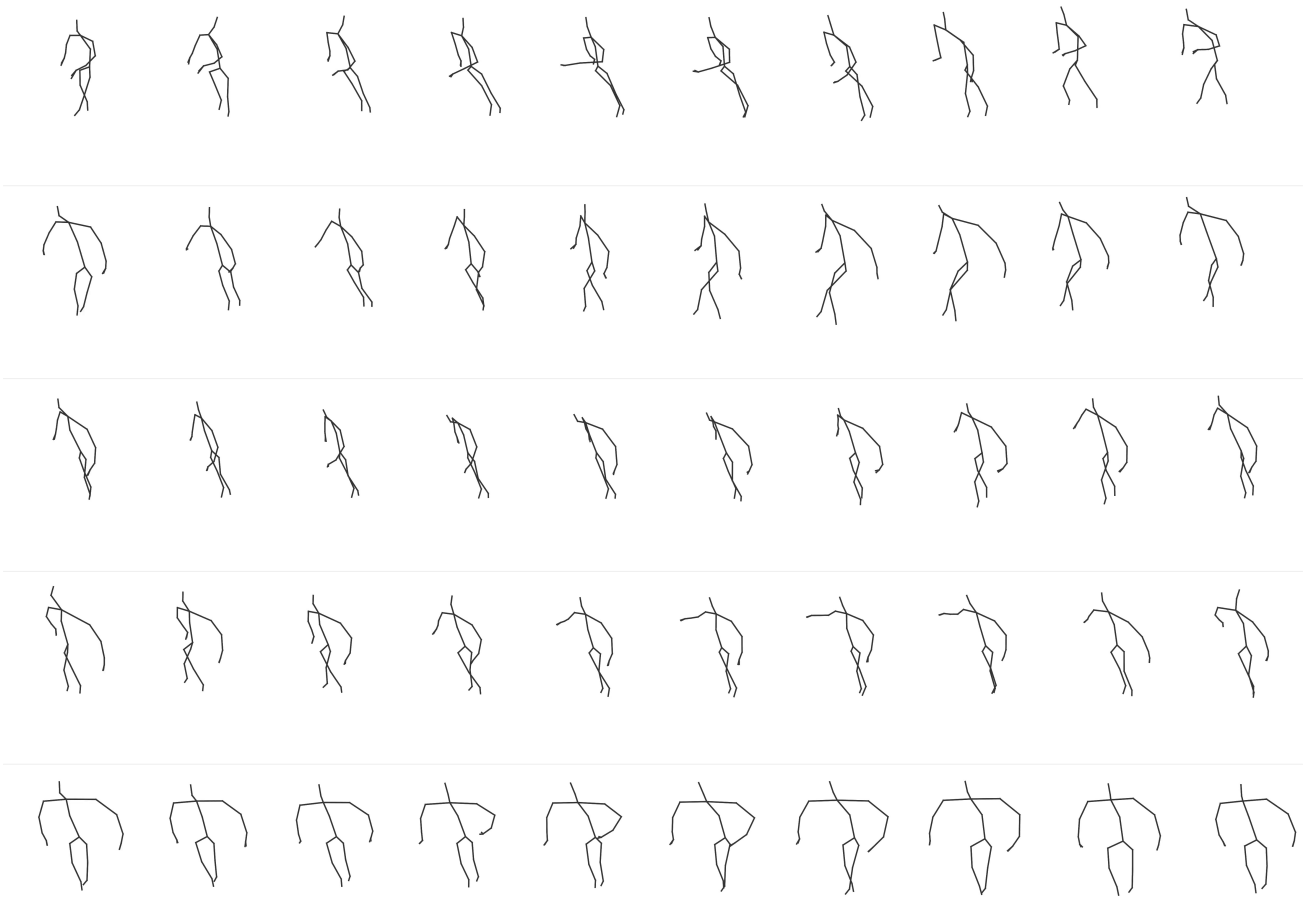
adding attention mechanisms or deformable convolutions layers-, or using progressive GANs, denoising GANs or hole-filling GANs for a more robust generation.

(a) Generated sequences from the proposed GAN model (“Hair Brushing” action)

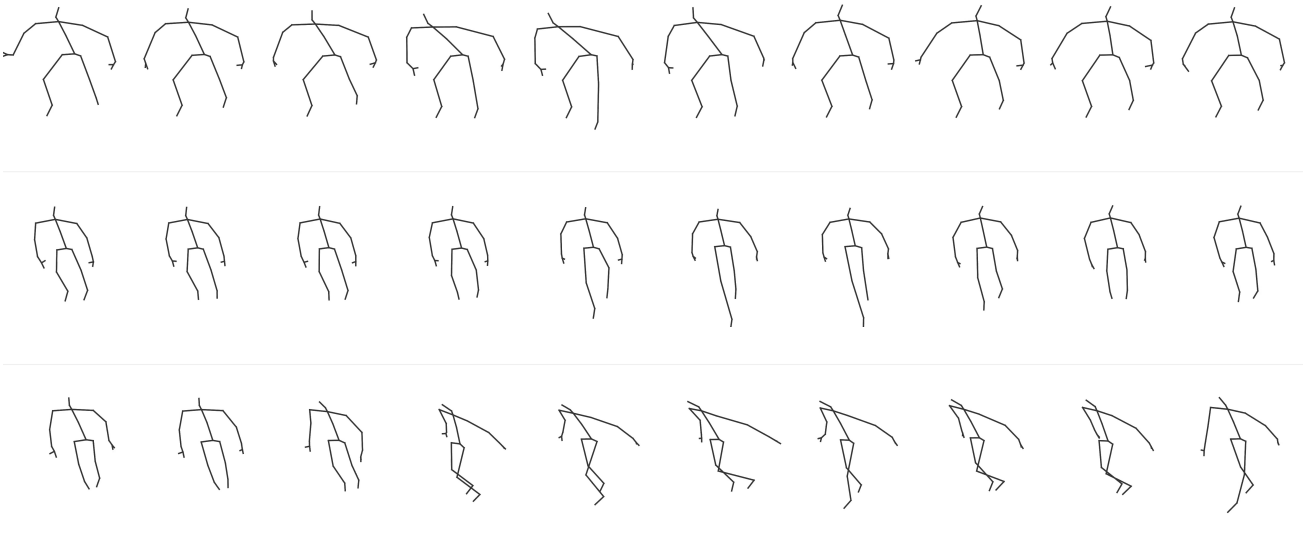


(b) Ground-truth sequences from the dataset (“Hair Brushing” action)

Figure 5.15 – Visual comparison of GAN-generated sequences and ground-truth sequences. Each line represents an individual sequence. All sequences represent the “hair brushing” action and they are all independent. Temporal order: left to right.



(a) Generated sequences from the proposed GAN model (“Kicking something” action)



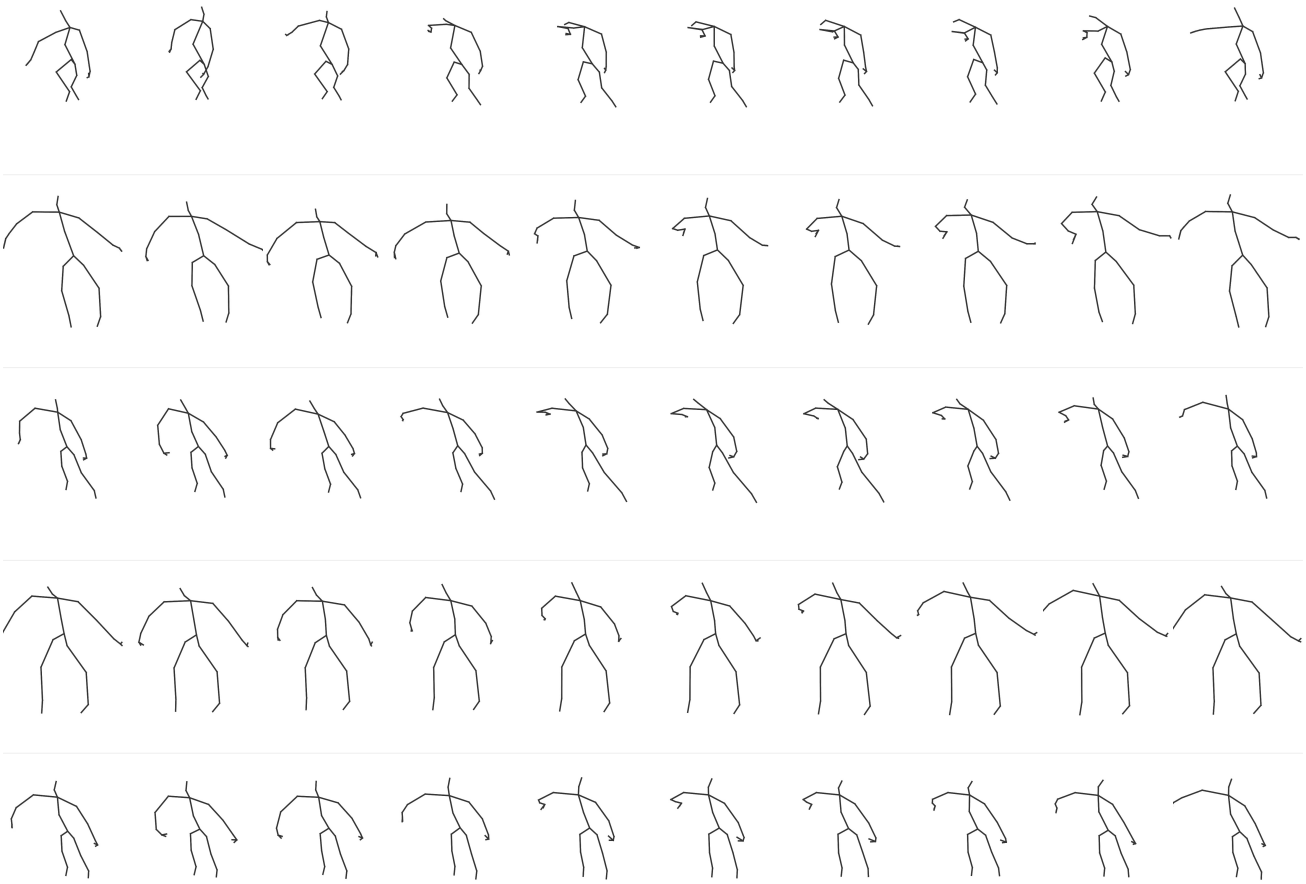
(b) Ground-truth sequences from the dataset (“Kicking something” action)

Figure 5.16 – Visual comparison of GAN-generated sequences and ground-truth sequences. Each line represents an individual sequence. All sequences represent the “kicking something” action and they are all independent. Temporal order: left to right.

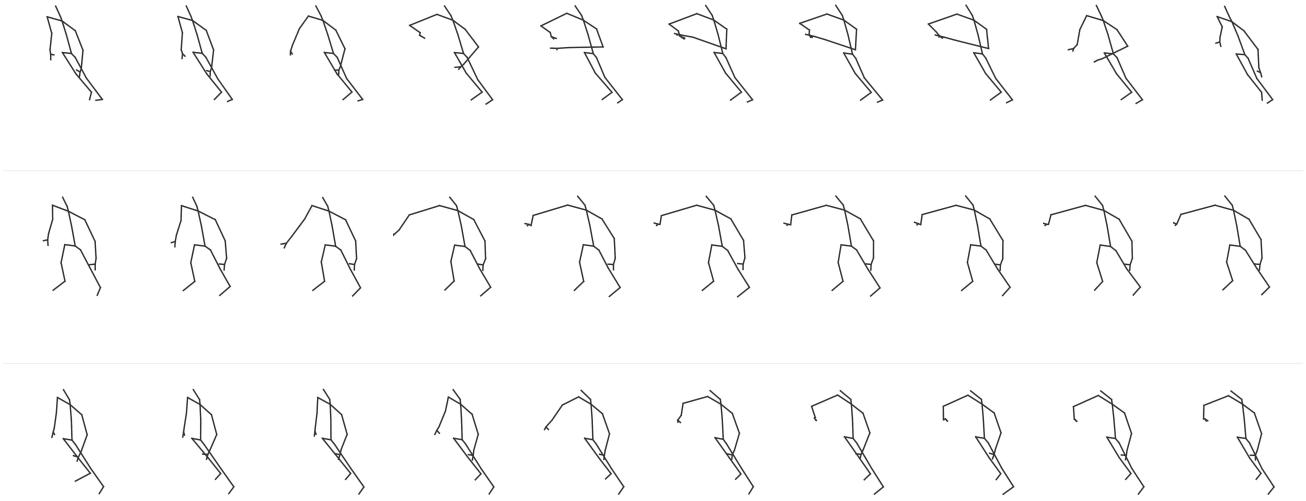
(a) Generated sequences from the proposed GAN model (“Clapping” action)

(b) Ground-truth sequences from the dataset (“Clapping” action)

Figure 5.17 – Visual comparison of GAN-generated sequences and ground-truth sequences. Each line represents an individual sequence. All sequences represent the “clapping” action and they are all independent. Temporal order: left to right.



(a) Generated sequences from the proposed GAN model (“Checking time on a watch” action)



(b) Ground-truth sequences from the dataset (“Checking time on a watch” action)

Figure 5.18 – Visual comparison of GAN-generated sequences and ground-truth sequences. Each line represents an individual sequence. All sequences represent the “checking time on a watch” action and they are all independent. Temporal order: left to right.

5.5 Conclusion

In this chapter, we proposed two different and novel approaches for human pose sequence generation. More precisely, our contributions are summarized in sections 5.3.2 and 5.4.2. The two proposed approaches are simple enough to be easily understood and reproduced. As the rare human pose sequence generation models found in the literature sometimes do not produce realistic results at all despite their apparent sophistication, the simplicity of the two approaches we propose is worth noting.

The first approach we propose, a Temporal Triplet Pose Auto-Encoder (TTPAE), mostly decouples spatial and temporal information, as the temporal objective used mainly "organizes" spatial pose representations in the latent space in a temporally-coherent manner. The second approach we propose, a Spatio-Temporal Conditional Generative Adversarial Network (STCGAN) couples spatial and temporal information.

The Temporal Triplet Pose Auto-Encoder approach mostly works at a frame-level, whereas the Spatio-Temporal Conditional Generative Adversarial Network approach essentially works at a sequence level.

Both of the proposed approaches might easily be extended. For instance, adversarial constraints at a frame-level or at a sequence-level might ensure even more realism to pose sequences generated with the Temporal Triplet Pose Auto-Encoder. Adding denoising abilities to the Spatio-Temporal Conditional Generative Adversarial Network and improving the embedding used for the class conditioning -a vanilla one-hot encoding for now- could lead to substantial improvements of the approach.

Chapter 6

Conclusion

"It takes something more than intelligence to act intelligently."

Dostoyevsky

In this thesis, we studied and introduced different representations for time series, based on deep learning models.

Summary

In recent years, deep learning has attracted increasing attention from both industrial actors and academic actors. Deep learning methods and approaches fueled significant advances in areas ranging from image processing to natural language processing. However, in many domains, input data consists in neither images nor text documents, but in time series that describe the temporal evolution of observed or computed quantities. Time series can be found in many domains including signal processing, telecommunications, weather forecasting, earthquake prediction, electroencephalography, electrocardiography, statistics, astronomy, robotics, human action recognition, handwriting recognition, speech recognition, music composition, health or finance.

A classical deep learning approach for sequence modeling consists in the use of recurrent neural networks (RNNs), where a dynamic memory is used to store information from one time step to another, alongside with a feedforward neural network that is responsible: (1) to read from the memory and to write to the memory; and (2) to model the sequence. While this approach is appealing, it also comes with its set of complex issues[1] from both a theoretical perspective and an applied perspective. These issues motivate us to look for other families of neural networks, in order to learn time series representations.

Progress made in the last decade in deep learning suggest that the context of an object[2] provides a lot of information about the object itself. As the saying goes: birds of feather flock together.

In this thesis, all the contributions we proposed implicitly embrace the idea that temporal context provides useful information that could or should be exploited by neural networks.

[1] E.g. regularization issues, vanishing-gradient issues or training duration issues, to name a few.

[2] E.g. a temporal context around a specific time step, when it comes to time-series.

Firstly, in the autonomous driving domain, we showed that, the analysis of a temporal window by a neural network can lead to better vehicle control results than classical approaches that do not use neural networks, especially in highly-coupled situations.

Secondly, in the gesture and action recognition domain, we introduced 1D parallel convolutional neural network (CNN) models. The use of a CNN architecture allows for a limited number of parameters. In our models, convolutions are performed over the temporal dimension, in order to detect and benefit from temporal invariances. Since the parameters of the convolution kernels are shared across the time dimension, these models can process both fixed-length inputs and variable-length inputs.

Thirdly, in the human pose motion generation domain, we introduced 2D convolutional generative adversarial neural networks where the spatial and temporal dimensions are convolved in a joint manner, allowing for a context that is not only temporal but also spatial.

Finally, we introduced an embedding where spatial representations of human poses are sorted in a latent space based on their temporal relationships. In layman's terms, this approach can be seen as an anamorphose of a *spatial*[3] latent space of human body poses, where the deformations are based on *temporal* distances between the poses. Compared to the other contributions of this thesis, where the temporal context is explicit, in this contribution the temporal context becomes implicit once the latent space has been learned.

Future Works

Our work can be extended in several directions. We suggest a few prospective ideas that possibly might be relevant for future research.

Vehicle control

Generative Adversarial Networks While the proposed neural network architectures show promising results, no guarantee is provided on whether the estimated control command is correct or not. A potential risk of the proposed approach is to fall outside of the domain of validity of the network, leading to unrealistic -and potentially dangerous- values. To circumvent this issue, an option could consist in getting always more training examples, and monitor the performance obtained by the neural network. This is easy in simulation, but much harder on vehicles in the real world. Another better option consists in the use of generative adversarial networks (GANs). One network, the generator, should estimate the correct control. A second network, the critic, should select progressively harder and harder couples of initial conditions and reference trajectory to follow, the goal of the critic being to find situations where the generator produces inappropriate results. The two networks would be trained in an adversarial fashion. The control generator network would thus be more robust to estimate the correct control command values, even in difficult scenarii.

Obstacle avoidance Obstacle avoidance competencies are essential and critical to guarantee human and vehicle safety, in the context of autonomous driving. While many obstacle

[3] As it relates to poses, which are grounded in the physical -spatial- space.

avoidance approaches exist, they usually occur at a path planning level in the perception-planning-control paradigm. However, obstacles may appear at a very last moment, e.g. due to earlier misdetections in the perception block, or due to an unforeseeable and abrupt change in the vehicle's environment. In such cases, the vehicle safety can not be guaranteed by an obstacle avoidance approach working at a path planning level only, since the generated emergency trajectory to follow may not feasible in the real world due to the vehicle's dynamics. One idea to overcome this critical issue could consist in training our neural network to not only generate a control command based on a reference trajectory and initial conditions, but also based on a bird-view map of the surrounding environment, including obstacles. Learning jointly both (inverse) vehicle dynamics and obstacle avoidance competencies offline during training would allow real-time and dynamics-informed obstacle avoidance. Such competencies would be extremely valuable, especially when imminent -and likely dangerous- obstacles appear on the road. This proposed approach might be trained in simulations, and then fine-tuned with transfer learning on real vehicles. It would allow more reactive and safer driving styles.

Reinforcement Learning Reinforcement Learning (RL) can be used to drive autonomous vehicles based on data coming from the sensors mounted on the vehicles: RGBD cameras, LIDARs, radars, ultrasounds, GPS devices, IMU sensors. Such approaches typically lead to bang-bang controls[4] with abrupt switches between two states. To slightly smooth the vehicle's trajectory, a fine-tuned proportional–integral–derivative (PID) controller can be added. A better approach might consist in informing the RL algorithm about the vehicle dynamics beforehand. To that end, one could either integrate our proposed neural network architecture(s) into a deep-RL approach, where our network would estimate the control to apply, leaving the room for the RL algorithm to adapt it based on other information coming from SLAM[5] data and from raw sensors data.

Human gesture and action recognition

Spatial Information While gesture and action recognition are temporal phenomena, many gestures and actions might actually be inferred based on spatial poses only. For instance, the existence of a static frame of one's touching one's shoes may be a very good predictor for a "tie one's shoelaces" action, regardless of temporal evolutions. As such, it may be worth spending time on the spatial aspect of gestures.

Spatial Attention To that end, a spatial attention head could be added to the model, to help the neural network select, or weight differently, body parts before analyzing the temporal evolutions of the joints.

Distances between joints To describe the spatial structure, a potentially useful hand-crafted feature to use to improve the performance of the neural network could be

[4]This is especially true when the RL approach involves a discrete action space, with quantized steering angle values or quantized wheels' torques.

[5]SLAM stands for "Simultaneous Localization And Mapping".

the distances between the joints. One drawback of this feature is that the number of distances grows quadratically with the number of joints. As such, it cannot be used on all structures, e.g. it cannot be used on detailed 3D meshes. To circumvent this drawback, an attention head could be introduced in the model to select a limited amount of distances (channels), the selected channels being considered as relevant for the recognition.

Pose standardization The introduction of a pre-processing module to make the pose (i.e. the landmarks or the skeletons) view-invariant could greatly improve the performances of the proposed neural networks. To that end, three heads could be integrated in the neural network. They would learn how to locate, how to rotate and how to scale the pose to a "standard" pose, which would then be used by the rest of the neural network model.

Embeddings Rather than hand-designing features and architectures, or in addition to it, the input poses could be embedded beforehand into a task-relevant representation. The denoising encoder-decoder embedding we proposed could potentially be an example of such a task-relevant embedding.

Graph-only approach Another research direction might consist in considering pose sequences as a single big spatio-temporal graph, where nodes represent a joint and edges represent either rigid-constraints (i.e. bones) or temporal continuity constraints, and exploring various approaches from the deep learning on graphs research area. Deep learning on graphs has attracted increasing attention by the research community in the very last years, enabling numerous potential research directions.

Fusion At the one hand, when it comes to gesture recognition, it appears that some information is redundant, either spatially (e.g. moving one's hand necessarily implies to move one's elbow for instance) or temporally (e.g. due to different temporal resolutions involved in an action). At the other hand, some piece of information in a gesture can shed a light on other piece of information from that gesture (e.g. when one's claps, the right and left moves of the hands are related to each other). For these reasons, one may want to perform fusion between inputs, in order to better allocate the resources of the neural network. Learning how to perform fusion of multivariate time series is difficult. The main questions to consider are: what should be merged? When? How? At which stage in the neural network architecture? Regarding the fusion between channels, late fusion appears to perform better than early fusion. However, a progressive fusion of information between channels might still improve the performance of the neural network. A fusion based on spatial relations or even on semantic relations might be relevant. To perform the fusion, self-supervised spatial, temporal or spatio-temporal embeddings could probably be exploited.

Human pose embedding

Static Pose Adversarial loss The anatomy loss may be replaced by an adversarial loss. The idea would be to resort to adversarial training in order to ensure that each individual (static) reconstructed human pose is considered as realistic by a neural network (human pose critic).

Dynamic Poses Adversarial loss To ensure that all transitions obtained via linear interpolations in the latent space trained with the time-sampled triplet loss, another adversarial loss might be used. The idea would be to resort to adversarial training in order to ensure that each individual (dynamic) reconstructed sequence of human poses -obtained via linear interpolations in the latent space- are considered as realistic by a neural network (human pose motion critic).

Sparsity loss The latent space we introduced sometimes exhibits redundant axes: two different dimensions sometimes code the same human pose semantical evolution. A simple L_1 loss could be used to encourage a sparse representation of the latent code z. An expression for that sparsity loss could be $\mathscr{L}_{sparsity}(z) = \alpha_{sparsity}\|z\|_1$ where $\alpha_{sparsity}$ would be an training hyperparameter.

Human motion generation

GAN Architectures The GAN architecture we introduced is relatively simple and could very likely be improved in order to produce even more realistic human pose motion sequences. For instance, one could consider using GAN architectures based on attention mechanisms or on deformable convolutions layers.

Classes embedding The action classes are one-hotted in the architecture we introduced. Rather than a one-hot encoding, an embedding of the classes could be used to reduce the dimensions required. Besides using traditional embedding techniques, introducing an embedding that respects the semantic relationships involved in the gestures might be worth investigating.

Poses and mask conditioning The model we introduced can generate new sequences conditioned on existing action classes. From a human perspective, it could be useful to be able to edit generated gestures. For instances, animators might want to manually fix a pose at a specific time step, and to keyframe it so that the generated pose at that time step in the generated sequence matches the pose they manually chose. To manually fix a human pose, the human pose embedding we introduced earlier could be used, to easily edit semantically the pose. However, matching that pose at a specific time step in the generated sequence requires a modification to our GAN architecture. One idea could consist in conditioning the GAN not only on a class, but also on (1) a temporal binary keyframes mask and (2) a manually-provided reference human pose motion sequence of the same length as the one to generate. The temporal binary keyframes mask would indicate whether a keyframe exists, or not, at each individual time step. At each time step, if a keyframe is set, the reference human pose would be used to condition the generator and the discriminator. Otherwise, if not keyframe is set, the generator and the discriminator would learn to discard the reference pose and not condition themselves

on it. Finally, a loss term that measures the distance between the generated poses and the associated keyframes, if any, would be added to the discriminator's losses, to encourage the generator to respect the provided reference keyframes. Such conditioning would allow human practitioners to harness the generator neural network, opening the door for AI-assisted human pose motion design tools.

Perspectives

Deep learning is a very exciting and a very dynamic research domain. From a practical perspective, deep learning approaches open the door to a myriad of very powerful applications. From a theoretical perspective, numerous discoveries related to deep learning, neural networks and high-dimensional spaces remain to be found. Some of the practical and theoretical key deep learning results only date back to very recent times, even if research on artificial neural networks has been ongoing for decades.

Forging human intuitions that are relevant in the deep learning area is hard, though, or, at least, not as immediate as one would like. One possible explanation for this observation might stand in the fact that human intuitions usually come from our direct experience with low-dimensional spaces, both at practical and at conceptual levels; however, these intuitions may not always hold true in high-dimensional spaces. All of this forces deep learning practitioners and researchers to adapt and, sometimes, to step into totally unknown territories. Current deep learning approaches can be robust to task-related noise, even though they may not be very well suited for ever changing environments and situations.

A very promising research area in the machine learning domain that might likely alleviate this issue is called self-supervised learning. Self-supervised learning is a learning technique where the training data is autonomously labeled, rather than manually labeled by humans. For instance, autonomous data labeling could be performed based on relations that exist between different pieces of information in the input data. Self-supervised learning provides more supervisory signals than supervised learning, and even more than reinforcement learning.

As such, more knowledge about the structure of the world might be learned through self-supervised learning than from both supervised learning and reinforcement learning.

Résumé en français

Chapitre 1
Introduction

Ce chapitre constitue l'introduction de la thèse. Il est subdivisé en quatre sections :

1. Contexte
2. Objectifs
3. Contributions
4. Structure de la thèse

Contexte Cette thèse s'inscrit dans le domaine de la recherche en intelligence artificielle.

Dans le langage courant, le terme "intelligence" est difficile à définir de manière précise et exhaustive car il recouvre des notions souvent floues et mal définies. Néanmoins, pour beaucoup, l'intelligence peut être considérée comme la capacité à s'adapter et à organiser des informations, le plus souvent dans le but de maximiser ses chances d'atteindre un objectif.

L'Intelligence Artificielle (IA) est le domaine scientifique qui étudie des programmes "intelligents".

Au sein des approches actuelles en IA, l'Apprentissage Profond désigne un ensemble de méthodes d'intelligence artificielle qui permettent à un programme d'apprendre une hiérarchie de concepts à partir d'exemples, sans nécessiter d'intervention humaine pour trouver ces concepts. En combinant plusieurs concepts, un programme peut découvrir des concepts plus abstraits. Par exemple, en combinant le concept géométrique de "croix" avec le concept de "couleur verte" , le programme peut découvrir le concept plus abstrait de "croix de pharmacie". La composition de représentations découvertes (apprises) par les programmes basés sur de l'apprentissage profond permet à ces programmes d'apprendre des concepts de plus en plus abstraits au fur et à mesure que l'on progresse dans les couches successives de composition.

Les méthodes basées sur des techniques d'apprentissage profond ont démontré leur excellence dans les domaines de l'imagerie et de la vision par ordinateur au cours de la dernière décennie, ainsi que dans le domaine du Traitement Automatique du Langage (TAL) au cours des toutes dernières années. Actuellement, l'état de l'art dans ces domaines repose sur des techniques et des modèles d'apprentissage profond.

Pourtant, dans de nombreux domaines, les données observées à considérer ne sont ni des images ni du texte mais des séries temporelles qui représentent l'évolution de grandeurs mesurées ou calculées au cours du temps. C'est par exemple le cas en météorologie (température, pression, vitesse du vent par ex.), en économie (taux, indices, *spread*), en sismologie, dans l'industrie (tension, consommation d'énergie électrique, capteurs), en médecine (électroencéphalogrammes, électrocardiogrammes, température, pression sanguine), en épidémiologie (nombre de cas positifs à une maladie ou à une pandémie), en reconnaissance vocale (séquences audio, mel-spectrogrammes), pour le contrôle de véhicules autonomes (trajectoires de référence) ou encore pour la reconnaissance de gestes (positions ou orientations des articulations du corps humain) pour ne citer que quelques exemples.

Objectifs Cette thèse vise à explorer des approches d'apprentissage profond pour les séries temporelles.

A cette fin, des architectures de réseaux de neurones artificiels sont étudiées.

Une architecture particulière de réseau de neurones artificiels, appelée réseau de neurones récurrents, ou *recurrent neural network* (RNN) en anglais, est réputée être adaptée au traitement de données séquentielles. Néanmoins, cette architecture présente des défauts et des limitations, autant d'un point de vue théorique que d'un point de vue pratique.

A ce titre, l'un des objectifs de cette thèse est d'étudier si des architectures de réseaux de neurones non-récurrents peuvent être -qualitativement et quantitativement- pertinentes pour traiter des données séquentielles.

Un des autres objectifs de la thèse est de découvrir si des séries temporelles contiennent "suffisamment" d'information pour pouvoir être la seule source de donnée à considérer pour des tâches complexes, par exemple pour des tâches de reconnaissance d'émotions faciales ou de contrôle de véhicules autonomes.

Contributions On peut résumer les contributions principales de cette thèse de la manière suivante :

Contrôle Couplé de Véhicule On propose une nouvelle approche, qui fait appel à des modèles d'apprentissage profond, pour le contrôle couplé de véhicule.

Sauf erreur, il s'agit de la première utilisation connue de réseaux de neurones profonds pour le contrôle couplé de véhicules à roues.

Deux exemples d'architectures de réseaux de neurones sont présentées et testées. Dans les deux cas, un modèle apprend la dynamique inverse d'un véhicule, en particulier la dynamique latérale-longitudinale couplée. Après la phase d'apprentissage, on utilise le réseau de neurones pour contrôler le véhicule. Un tel contrôleur appris par apprentissage profond est capable de gérer en temps réel des situations avec un fort couplage longitudinal et latéral.

Reconnaissance de Mouvements de Pose Humaine On propose une nouvelle architecture de réseaux convolutifs pour la reconnaissance d'actions et de gestes à partir de séquences de poses humaines.

Dans cette architecture, des convolutions parallèles unidimensionnelles opérant sur la dimension temporelle sont utilisées pour détecter des motifs temporels.

L'architecture proposée n'utilise que des convolutions, qui sont faciles à entraîner, à auditer, et permettent d'aboutir à des modèles relativement légers. De plus, cette architecture ne nécessite que des adaptations minimales pour être appliquée à des nouveaux types de gestes, de capteurs, ou de formats de séquences de poses tels que des séquences de positions d'articulations ou de points particuliers de la main, du corps humain ou du visage.

Génération de Mouvements de Pose Humaine On propose deux nouvelles architectures de réseaux de neurones pour la synthèse de séquences de poses humaines : une architecture de réseaux de neurones auto-encodeurs débruitants pour l'apprentissage auto-supervisé de poses humaines, et une architecture de réseaux antagonistes génératifs.

L'espace latent de l'auto-encodeur proposé est contraint par un objectif spatial et par un objectif temporel. Une fonction de coût spatial encourage les poses à être correctement reconstruites spatialement par l'auto-encodeur. Une fonction de coût temporel, basée sur une triplette de poses échantillonnées temporellement, encourage les représentations internes (spatiales) des poses apprises par le réseau de neurones à s'organiser d'une manière cohérente temporellement. Les réseaux encodeurs et décodeurs qui constituent l'auto-encodeur peuvent être utilisés pour obtenir des représentations de poses humaines, en particulier pour des usages en temps réel. L'espace latent de l'auto-encodeur débruitant présente des propriétés sémantiques qu'il serait intéressant d'étudier dans des travaux ultérieurs.

Les réseaux antagonistes génératifs proposés font quant à eux appel à une représentation des séquences de poses qui permet d'utiliser une architecture de neurones convolutifs.

Structure de la thèse Cette section présente le plan de la thèse.

Chapitre 2
État de l'art de la Modélisation de Séquences par des Réseaux de Neurones

Ce chapitre propose un état de l'art des principales approches de modélisation de données séquentielles avec des réseaux de neurones artificiels. Il est subdivisé en six sections :

1. Introduction
2. Réseaux de neurones à propagation avant
3. Réseaux de neurones récurrents

4. Réseaux de neurones convolutifs

5. Mémoire associative basée sur l'attention

6. Conclusion

Introduction Ce chapitre présente un état de l'art des approches de modélisation des données séquentielles avec des réseaux de neurones artificiels. Plus précisément, cette thèse s'intéresse aux données séquentielles relatives à des grandeurs physiques observées. Ces données sont représentées sous la forme de suites, finies ou infinies, de vecteurs de $\mathbb{R}^n$ indexées par le temps, où n désigne la dimension de la série temporelle considérée. Les principales méthodes d'apprentissage automatique qui permettent de modéliser des données séquentielles sans toutefois relever des réseaux de neurones sont brièvement citées. Le reste du chapitre présente les principales méthodes basées sur des réseaux de neurones et de l'apprentissage profond pour modéliser de phénomènes à partir de séries temporelles.

Réseaux de neurones à propagation avant Les données séquentielles présentent généralement des régularités : un signal peut présenter une ou plusieurs périodicités, deux signaux représentant un même phénomène peuvent ne pas être indépendants l'un de l'autre, etc. A ce titre, il peut être judicieux de faire appel à des réseaux de neurones artificiels qui présentent des biais inductifs pour modéliser des séries temporelles.

Les séries temporelles pouvant être de longueur (durée) arbitrairement grande, les réseaux de neurones artificiels ne travaillent en général que sur une sous-fenêtre temporelle de longueur finie à l'échelle du phénomène considéré. C'est le cas des réseaux de neurones à retard temporel, ou *time delay neural networks* (TDNN) en anglais, où, à chaque instant t, le réseau de neurones artificiel ne considère qu'un nombre fini de valeurs passées de la série temporelle d'entrée pour calculer la valeur de sortie à l'instant t, par exemple les valeurs de la série temporelle d'entrée à $t-1$, $t-2$, $t-4$ et $t-8$. La liste des indices temporels considérés dans un TDNN à un instant t est fixée, et ne dépend donc pas de la valeur prise par la série temporelle à l'instant t.

Déterminer la liste adéquate des indices temporels d'intérêt pour un problème donné nécessite une connaissance experte et se révèle être une tâche difficile à réaliser. Plutôt que de fixer "à la main" la liste des pas de temps de la série temporelle que le réseau de neurones va considérer comme c'est le cas pour un TDNN, il est possible pour certains réseaux de neurones d'apprendre de manière automatique quelles entrées prendre en compte (ou non), et à quel degré. Ce processus de sélection de certains aspects de l'information -au détriment d'autres aspects pourtant visibles- est appelé "attention". Les réseaux de neurones qui utilisent un mécanisme d'attention sont introduits dans la section 5. de ce chapitre.

Réseaux de neurones récurrents Les réseaux de neurones à propagation avant sont performants pour reconnaître des motifs. Néanmoins, ils manquent de plasticité dans la mesure où ces réseaux possèdent une architecture et des paramètres qui sont fixés une fois pour toutes. Un réseau de neurones récurrents, ou *recurrent neural network* (RNN) en anglais, combine un réseau de

neurones à propagation avant avec un état caché, ce dernier pouvant être considéré comme une mémoire dynamique. A chaque pas de temps t, la sortie du réseau de neurones est fonction de la valeur de la série temporelle d'entrée à l'instant t, ainsi que de la valeur de l'état caché à l'instant t.

Entraîner des réseaux de neurones récurrents se révèle être relativement difficile, la valeur du gradient ayant souvent tendance à diminuer jusqu'à devenir évanescente ou au contraire à augmenter jusqu'à exploser, et ceci pour des raisons à la fois théoriques (liées au choix de la fonction d'activation utilisée ou à des questions de normalisation, par exemple) et pratiques (liées au conditionnement des vecteurs et des matrices utilisés, par exemple).

Dans la pratique, deux types d'approches se distinguent pour l'entraînement des réseaux de neurones récurrents. La première consiste à concevoir des algorithmes d'entraînement plus robustes, par exemple en tronquant les gradients en-deça d'un certain seuil ou encore en lissant les gradients. La deuxième consiste à utiliser des fonctions d'activations plus complexes, afin d'aligner ou de déformer dynamiquement le temps en fonction des entrées. Pour ce faire, des portes, ou *gates* en anglais, sont utilisées. Les portes indiquent si, et à quel point, leur signal d'entrée doit être atténué ou non; elles permettent ainsi de réaliser un filtrage de l'information en entrée.

Les deux principaux réseaux de neurones récurrents qui utilisent un système de portes sont les *Long Short-Term Memory* (LSTM) et les *Gated Recurrent Unit* (GRU). L'architecture de ces deux réseaux de neurones récurrents permet de limiter grandement une potentielle évanescence du gradient : l'entraînement de ces réseaux de neurones par des méthodes de rétropropagation de gradient est ainsi grandement facilité. Par métonymie et abus de langage, certains auteurs désignent par le terme de "RNN" les réseaux LSTM ou GRU car ces deux réseaux sont en pratique les deux réseaux de neurones récurrents (RNN) les plus couramment utilisés à ce jour.

Il existe néanmoins de nombreux autres réseaux de neurones récurrents moins connus.

Réseaux de neurones convolutifs Un réseau de neurones convolutifs, ou *convolutional neural network* (CNN) en anglais, est un réseau de neurones à propagation avant dans lequel l'opération de multiplication matricielle est remplacée par une opération de convolution au sein d'au moins une des couches.

Les CNN ont tendance à se révéler très performants sur des données dont la topologie est décrite par une grille. C'est le cas des séries temporelles et des images par exemple, puisque ces types de données peuvent être vus comme un champ de vecteur qui prend des valeurs sur une grille espacée régulièrement, cette grille étant à une dimension (temps) dans le cas des séries temporelles et à deux dimensions (grille spatiale de pixels) dans le cas des images. Les réseaux de neurones convolutifs appliqués à des séquences détectent des régularités temporelles. Les réseaux de neurones convolutifs présentent l'avantage de nécessiter moins de paramètres que les réseaux de neurones à propagation avant classiques "équivalents" sans convolution.

Il est possible d'étendre la définition des réseaux de neurones convolutifs à des domaines irréguliers, tels que des graphes, par exemple.

Mémoire associative basée sur l'attention Les mécanismes d'attention sont des mécanismes qui sélectionnent -et ciblent- un aspect spécifique des données en entrée, tout en écartant le reste de l'information pourtant également perceptible au sein de ces données.

Dans le cas des données séquentielles, les mécanismes d'attention reviennent souvent, en pratique, à des mécanismes d'alignement entre deux séquences. Un tel alignement entre deux séquences $\mathbf{u}$ et $\mathbf{v}$ peut être représenté par une matrice $\mathbf{A}$ dont chacune des dimensions est donnée par la longueur des deux séquences considérées. Chaque coefficient $\mathbf{A}_{i,j}$ de cette matrice représente "l'attention" (c'est-à-dire le score d'appariement ou d'affinité) : $\mathbf{A}_{i,j} = \text{affinité}(\mathbf{u}_i, \mathbf{v}_j)$ entre la valeur prise par les deux séquences prises aux pas de temps respectifs i et j. De nombreuses fonctions d'affinité sont envisageables. Par exemple, la fonction cosinus ou le produit scalaire peuvent être utilisés. Il est également possible d'utiliser des fonctions avec des paramètres qui peuvent être appris au cours d'un entrainement.

Le terme de réseaux de neurones transformateurs, ou *transformer networks* en anglais, désigne une architecture de réseaux de neurones faisant appel à des mécanismes d'attention, initialement conçus pour traiter des données séquentielles textuelles dans le domaine du traitement du langage naturel. L'architecture de ces réseaux de neurones se retrouve dans tous les modèles à l'état de l'art actuel en traitement du langage naturel. A ce jour, cette architecture de réseaux de neurones demeure récente. Ainsi, malgré les très bonnes performances obtenues par ces réseaux dans le domaine de la modélisation de données séquentielles, certaines questions relatives à ces modèles restent ouvertes : on peut par exemple songer à la "nature" de ce qui est appris par ces réseaux, à l'importance des têtes utilisées dans ces réseaux, aux choix relatifs à l'encodage positionnel ou encore au lien avec l'apprentissage en quelques coups, ou *few-shot learning* en anglais, pour ne citer que quelques exemples.

Enfin, il existe de nombreuses autres architectures de réseaux de neurones qui font appel à des mécanismes d'attention, par exemple de manière à obtenir des réseaux de neurones avec des mémoires associatives où les données peuvent-être indexées et retrouvées autant par leur adresse (localisation) que par leur contenu (nature).

Conclusion Pour modéliser des données séquentielles avec des réseaux de neurones, il est possible d'utiliser des réseaux de neurones à propagation avant densément connectés simples ou avec de l'attention, des réseaux de neurones convolutifs, qui filtrent l'information de manière spectrale, ou des réseaux de neurones récurrents, qui font appel à une mémoire externe dynamique pour stocker de l'information. Au-delà de la question de la performance, la question de la nature même des réseaux de neurones à utiliser pour modéliser des données séquentielles de manière pertinente reste une question ouverte pour la plupart des applications.

Chapitre 3
Réseaux de Neurones Artificiels pour le Contrôle Couplé de Véhicule

Ce chapitre porte sur la question du contrôle couplé de véhicules autonomes avec des réseaux de neurones artificiels. Il est subdivisé en six sections :

1. Introduction
2. Revue de littérature sur le sujet
3. Simulateur de véhicule : modèle à 9 degrés de liberté
4. Modélisation par réseaux de neurones artificiels
5. Comparaison entre les approches classiques et les approches basées sur des réseaux de neurones
6. Conclusion

Introduction Le développement récent de l'apprentissage profond a permis des progrès colossaux dans de nombreux domaines de recherche, y compris dans le domaine des véhicules autonomes. L'utilisation de réseaux de neurones à des fins de segmentation d'image pour la conduite est un domaine qui a été largement exploré ces dernières années; des architectures de réseaux de neurones très performantes pour réaliser ces tâches sont désormais disponibles. Plus récemment, plusieurs équipes ont proposé d'entraîner des réseaux de neurones de bout en bout, ou *end-to-end* en anglais, pour calculer directement les commandes à appliquer à un véhicule pour le conduire à partir de données brutes telles que des images en couleurs ou des données radar et lidar. Cette idée de conduite de bout en bout est particulièrement attrayante car elle élimine le besoin de concevoir à la main des algorithmes de planification de mouvement et des algorithmes de contrôle. Néanmoins, déléguer la sécurité des occupants de la voiture à un logiciel dont les décisions ne sont pas intelligibles semble problématique. Une solution intermédiaire consiste à séparer les différentes étapes de la conduite en blocs fonctionnels auditables : par exemple sous la forme d'un bloc de perception, d'un bloc de plannification et d'un bloc de contrôle. Chaque bloc possède alors une complexité plus faible, ce qui permet d'une part de le modéliser par des réseaux de neurones mono-tâches plus simples, et d'autre part une plus grande capacité de validation et d'audit de ces réseaux par des experts. Ce chapitre étudie l'utilisation de réseaux de neurones artificiels pour le contrôle couplé de véhicules autonomes.

Revue de littérature sur le sujet Dans certaines situations hautement dynamiques, par exemple dans le cas d'une manœuvre d'évitement à haute vitesse face à un obstacle sur la route, le couplage important entre la dynamique latérale et longitudinale est difficile à modéliser lorsque l'on s'approche des limites du véhicule. Une modélisation précise de ce couplage fait intervenir des relations complexes et non-linéaires entre plusieurs variables d'état. Il n'est ainsi pas possible d'utiliser un tel modèle pour des applications en temps-réel. Pour cette raison, la plupart

des approches qui existent dans le domaine de la planification de mouvement s'intéressent à des modèles plus simples qui évitent les situations à fort couplage. De manière similaire, la recherche dans le domaine du contrôle considère généralement la dynamique latérale et longitudinale séparément pour simplifier le problème. Bien que ces simplifications mènent à de bons résultats dans des situations de conduite classiques, elles se révèlent problématiques et dangereuses lorsque le véhicule s'approche des cas limites, par exemple à haute vitesse ou sur des routes glissantes.

Simulateur de véhicule : modèle à 9 degrés de liberté Cette section présente les équations utilisées dans le modèle très réaliste qui sert à simuler un véhicule. Les dynamiques du châssis du véhicule, des roues, et des pneus sont modélisées. Le modèle possède neuf degrés de liberté : trois degrés de liberté décrivent le mouvement du véhicule dans le plan, deux degrés de liberté décrivent la rotation du châssis du véhicule, et quatre degrés de liberté décrivent la vitesse de rotation des quatre roues.

La commande à imposer correspond à l'angle de rotation de la roue avant gauche (angle du volant) et au couple de chacune des quatre roues.

Modélisation par réseaux de neurones artificiels Cette section propose de modéliser la dynamique inverse du véhicule par un réseau de neurones artificiels.

Dans un premier temps, le simulateur, introduit dans la section précédente, est utilisé pour générer un jeu de données de conduite qui sera utilisé par la suite dans le cadre d'un apprentissage supervisé. Ce jeu de données est composé de plusieurs dizaines de milliers d'exemples. Chaque exemple est décrit par un triplet : ce triplet se compose de l'état initial du véhicule, du contrôle (commande) qui est appliqué au véhicule pendant la durée de la simulation, et de la trajectoire (série temporelle des positions) réellement suivie par le véhicule au cours de la simulation. Pour chacun des exemples, le contrôle appliqué et l'état initial du véhicule sont tirés aléatoirement de manière à pouvoir couvrir de nombreuses situations.

Deux architectures de réseaux de neurones sont proposées pour apprendre de manière supervisée la dynamique inverse du véhicule.

La première architecture consiste en un perceptron multicouche, ou *multilayer perceptron* (MLP) en anglais, à cinq couches cachées, dont les entrées sont le vecteur d'état initial du véhicule et la trajectoire suivie par le véhicule présentée au réseau sous forme de vecteur colonne.

La deuxième architecture étend l'architecture de ce perceptron multicouche, en ajoutant en entrée la sortie d'un module de pré-traitement de la trajectoire par un réseau de neurones (module) convolutif. Le terme de CNN est utilisé pour désigner cette deuxième architecture.

Un entrainement de chacun des réseaux de neurones est réalisé. L'objectif de chaque réseau de neurones consiste à prédire le contrôle (a priori inconnu) à appliquer à un véhicule dans un état initial donné (connu) afin de respecter une trajectoire de référence (connue). Pour l'entraînement, une fonction de coût en erreur quadratique moyenne est utilisée avec une

régularisation L_2. De plus, grâce à un hyperparamètre γ, l'entraînement accorde plus de priorité à la dynamique latérale qu'à la dynamique longitudinale.

Comparaison entre les approches classiques et les approches basées sur des réseaux de neurones Une fois chacun des deux réseaux de neurones appris de manière supervisé sur le jeu de données, les paramètres des réseaux de neurones sont gelés.

Les deux réseaux de neurones sont ensuite utilisés comme contrôleurs pour contrôler un véhicule. Les réseaux sont évalués sur une piste de test inconnue. La trajectoire de référence à suivre à chaque instant est générée par une courbe de bézier cubique qui ramène le véhicule vers la piste à suivre.

Ainsi, à chaque itération, (i) une courbe de bézier est calculée pour relier la position réelle du véhicule à la piste, (ii) une requête composée de l'état du véhicule ainsi que de la courbe de bézier précédemment calculée est fournie en entrée du réseau de neurones, (iii) le réseau de neurones retourne l'angle de rotation de la roue avant gauche (angle du volant) et le couple de chacune des quatre roues que le véhicule va appliquer jusqu'à la prochaine itération. La durée d'une itération est de 300ms, mais la requête au réseau de neurones prend moins de 2ms.

Enfin, deux méthodes classiques de contrôle sont également considérées. Elles ne font pas appel à des réseaux de neurones. Le profil des erreurs en termes d'angle du volant, de couple de chaque roue et d'erreur latérale est étudié.

Il s'avère que les commandes de contrôle obtenues sont plus lisses pour le modèle CNN que pour le modèle MLP, en particulier dans les virages.

Pour les situations de conduite normales faiblement couplées, les approches classiques -qui présentent l'avantage de posséder des garanties théoriques- avec des contrôleurs découplés en latéral et en longitudinal peuvent sembler préférables à des approches neuronales.

Néanmoins, dans les situations hautement couplées (c'est le cas de la section de route numéro 6) l'usage de réseaux de neurones pour le contrôle de véhicule est vital. Les deux modèles MLP et CNN traversent avec succès la section de route en tenant compte du couplage de la dynamique latérale et longitudinale. Il est à noter que les deux réseaux de neurones anticipent les virages et, si nécessaire, ralentissent délibérément la vitesse du véhicule en-deçà de la vitesse de consigne afin de pouvoir réussir à suivre correctement la trajectoire de référence dans les virages.

Dans les parties les plus difficiles du parcours, les performances des modèles classiques se révèlent bien plus faibles que les performances obtenues avec les réseaux de neurones.

Conclusion Ce chapitre présente des approches et des modèles pour le suivi de trajectoire de véhicules autonomes grâce à des méthodes d'apprentissage profond. Deux architectures de réseaux de neurones sont évaluées à partir d'un modèle de dynamique de véhicule très fidèle. L'objectif des réseaux de neurones est d'estimer l'angle de rotation du volant et le couple de chacune des quatre roues à appliquer pour pouvoir suivre une trajectoire de référence étant donné l'état du véhicule. Il s'avère que les meilleures performances sont obtenues dans le cas du modèle CNN :

les commandes obtenues sont lisses, sans à-coups, et sont d'une grande précision. De plus, comparés à des méthodes classiques, les réseaux de neurones présentent l'avantage majeur de pouvoir gérer des situations à fort couplage longitudinal-latéral, et cela en temps-réel.

Chapitre 4
Reconnaissance de Gestes avec des Réseaux de Neurones Convolutifs sur le Temps

Ce chapitre porte sur la question de la reconnaissance de gestes et d'actions humaines avec des réseaux de neurones convolutifs où la dimension temporelle du mouvement est convoluée. Il est subdivisé en six sections :

1. Introduction
2. Revue de littérature sur le sujet
3. Réseaux neuronaux convolutifs relativement au temps
4. Expériences
5. Visualisations des modèles
6. Conclusion

Introduction Pouvoir reconnaitre les intentions humaines et les actions humaines est utile dans la vie courante, et peut même être crucial dans certaines situations. En particulier, la faculté de reconnaitre des gestes, des attitudes, des postures ou encore des expressions faciales se révèle souhaitable et utile dans de nombreuses situations. Le geste est un moyen simple et naturel qui nous permet d'interagir avec notre environnement.

Différents capteurs permettent de représenter numériquement des gestes humains. C'est le cas bien sûr des appareils photographiques et des caméras classiques qui fournissent des images en couleurs. C'est également le cas des caméras stéréo, des caméras de profondeur, des caméras événementielles, des centrales inertielles ou des systèmes de capture de mouvement par exemple.

Or, des expériences ont montré que la seule connaissance de la position des articulations du corps humain et de leur mouvement au cours du temps est suffisante pour permettre à un sujet de reconnaitre et de discerner des actions humaines. La position des articulations du corps humain peut être estimée en temps réel grâce à des techniques de vision par ordinateur, ou être fournie par des capteurs physiques.

Ainsi, la simple donnée d'une séquence temporelle de vecteurs décrivant les positions des articulations du corps humain au cours d'un mouvement semble a priori suffisante pour pouvoir reconnaître ce mouvement. Une telle donnée de pose humaine est plus respectueuse de la vie

privée que des données plus denses comme des vidéos (qui garantissent moins l'anonymat) et présente l'avantages d'être plus légère et donc potentiellement plus rapide à analyser. Ce critère de vitesse est important car, dans la plupart des situations, l'analyse du mouvement n'est utile que si elle est réalisée en temps-réel. C'est par exemple la cas pour des tâches de domotique ou de conduite autonome pour saisir l'intention d'un piéton de traverser, ou non, la rue.

Ce chapitre propose une architecture de réseaux de neurones convolutifs (CNN) pour réaliser des tâches de reconnaissance de gestes et d'actions en se basant sur des informations temporelles. Les mouvements peuvent être analysés soit à chaque pas de temps, soit à la fin du mouvement. Outre sa relative légèreté, le modèle proposé possède l'avantage majeur de ne faire appel qu'à des couches de convolutions qui sont plus facile à entraîner et à auditer que des couches récurrentes. Enfin, l'architecture du modèle ne nécessite qu'une adaptation minime et immédiate pour pouvoir être appliquée à différentes sources de capteurs ou à des mouvements variés : gestes de la main, mouvements du corps entier, expressions faciales, etc.

Revue de littérature sur le sujet De nombreuses approches ont été proposées dans la littérature scientifique dans le but de reconnaître des mouvements. Certaines sont basées sur des méthodes d'apprentissage automatique et font appel à des caractéristiques expertes définies manuellement, basées par exemple sur des distances géométriques entre les différentes articulations, sur des histogrammes ou encore sur des travaux de géométrie riemannienne. D'autres se basent sur des méthodes d'apprentissage profond avec des réseaux de neurones : on retrouve des approches qui font appel à toutes les principales architectures de réseaux de neurones artificiels, à savoir les réseaux de neurones à propagation avant densément connectés, les réseaux de neurones récurrents, les réseaux de neurones convolutifs, ou encore des réseaux de neurones faisant intervenir des mécanismes d'attention. Néanmoins, les approches qui font appel à des réseaux de neurones convolutifs sont encore relativement peu explorées, et ne se basent que très rarement sur des données de pose humaine.

Réseaux neuronaux convolutifs relativement au temps Cette section propose une nouvelle approche pour la reconnaissance de gestes et d'actions à partir de séquences de poses humaines.

Dans un premier temps, une définition formelle d'une séquence de poses sous la forme d'un tenseur à trois dimensions (Temps, Articulations, Dimensions) est donnée. Quelques visualisations possibles de ces tenseurs sont présentées à titre illustratif. Il est possible, sans perte d'information, de considérer ces séquences de poses comme des tenseurs à deux dimensions (Temps, Canaux).

Dans un second temps, l'architecture de la famille de réseaux de neurones proposée, intitulée SkelNet, est détaillée. Enfin dans un dernier temps, les modalités d'évaluation des performances des réseaux SkelNet sont présentées : métriques retenues, fonction de coût choisie pour l'entraînement, régularisations appliquées, algorithme d'optimisation choisi, jeu de données utilisé, méthode d'initialisation des poids, et taille des lots, ou *batch size* en anglais, durant l'entrainement.

Un modèle d'apprentissage profond SkelNet est un réseau de neurones modélisant, grâce à convolutions temporelles, des séquences de poses humaines de gestes ou d'actions, par exemple dans un but de classification (reconnaissance) de ces gestes ou de ces actions. Le modèle prend en entrée des séquences de poses.

Dans un premier temps, un module de pré-traitement apprend à générer (sans impact sur la dimension temporelle) des caractéristiques susceptibles d'être plus pertinentes que les données brutes, par exemple en mélangeant l'information de différents canaux vers de nouveaux canaux plus pertinents.

Une des forces principales du modèle réside dans l'extraction de caractéristiques et de motifs temporels, ces motifs étant ensuite utilisés pour réaliser la classification de la séquence. Dans le modèle de référence, chaque canal d'entrée est traité séparément. Il est néanmoins possible de mettre en commun ces traitements comme le montrent les expériences réalisées dans les sections suivantes. Pour plus de performance et de robustesse, la phase d'extraction de caractéristiques temporelles est séparée en trois branches parallèles, et dont les sorties sont finalement concaténées pour être réunies.

La première branche est constituée d'une succession de plusieurs couches convolutives où la convolution est unidimensionnelle et ne s'applique qu'à la dimension temporelle du tenseur. Chaque couche convolutive suit une architecture classique dans les réseaux convolutifs avec la présence d'une alternance de couches de convolutions, de mises en commun et d'échantillonnages, ou *pooling* en anglais, de régularisations, et d'activations non-linéaires. Les convolutions et les mises en commun considèrent uniquement la dimension temporelle du tenseur.

Une deuxième branche parallèle à la première est présente dans le module d'extraction de motifs temporels. Elle est identique en tous points à la première, à la différence près que la taille des noyaux de convolutions varie. Les deux branches réalisent ainsi un filtrage temporel à des résolutions temporelles différentes.

Une troisième branche parallèle aux deux précédentes, improprement appelée branche résiduelle, réalise un sous-échantillonnage par moyenne temporelle du tenseur d'entrée du module.

Enfin, pour chaque canal, les sorties des trois branches sont concaténées dans un vecteur unique.

Une fois que les caractéristiques temporelles ont été extraites de chacun des canaux, elles sont toutes fournies à un réseau de neurones densément connecté qui se charge de réaliser la classification finale.

Le modèle global, avec le module de pré-traitement, les modules d'extraction de motifs temporels, et le module final de classification, est différentiable et peut être entraîné de bout en bout.

Expériences De nombreuses expériences sont proposées et effectuées afin d'étudier l'architecture de réseaux de neurones convolutifs SkelNet proposée.

Dans un premier temps, l'influence de la représentation initialement choisie pour les données est étudiée : effet de la standardisation des données par une méthode de Procrustes, effet des différentes techniques d'augmentation de données, effet de l'échantillonnage, choix de la représentation en positions ou en rotations, et reconnaissance précoce de gestes.

Dans un deuxième temps, une justification aux différents choix qui sous-tendent l'architecture du réseau de neurones est proposée : effet de la méthode de mise en commun, ou *pooling* en anglais, effet du module de pré-traitement, effet du nombre de couches et de l'opérateur de convolution temporelle, effet de la méthode de régularisation.

Dans un troisième temps, une étude des paramètres du modèle et de leur éventuel partage est étudié : effet du partage des poids au niveau de l'entrée, effet du partage des poids selon la résolution, effet du partage des poids selon le niveau d'abstraction et de profondeur.

Enfin, dans un dernier temps, on montre que le modèle proposé peut s'appliquer à de nouveaux jeux de données, à de nouvelles sources de données et à de nouvelles tâches : reconnaissance d'action à partir de mouvements du corps entier, reconnaissance d'émotions à partir de mouvements d'expressions faciales.

Visualisations des modèles La question de l'interprétation des traitements réalisés par les réseaux de neurones artificiels est une question scientifique qui reste ouverte et débattue. Il est souhaitable de tenter de "comprendre" au moins partiellement ce que le réseau de neurones apprend pour mieux saisir pourquoi il classe des gestes ou des actions dans une catégorie plutôt qu'une autre. Pour ce faire, des visualisations, basées sur des méthodes d'attribution a posteriori, sont étudiées dans cette section.

Il apparait ainsi des différences entre les variations du modèle proposé. Des différences sont par exemple observées selon que les canaux d'entrée du modèle (i) ne sont pas partagés, (ii) sont partagés par blocs de canaux x, y et z, ou (iii) sont partagés globalement pour tous les canaux.

Conclusion Ce chapitre propose une nouvelle approche de reconnaissance de gestes et d'actions à partir de séquences de poses humaines. L'approche se base sur une famille de réseaux de neurones convolutifs 1D. Cette famille de modèles, intitulée SkelNet, fait usage de convolutions appliquées à la dimension temporelle afin d'extraire des motifs temporels du mouvement (du squelette) de la pose humaine. Une étude extensive de la performance des modèles est réalisée. Des réseaux peu profonds se révèlent suffisants pour des jeux de données de tailles moyennes tandis que des réseaux plus profonds sont plus avantageux pour des jeux de données de grande taille. Le modèle SkelNet est relativement léger ($\sim$2.4M de paramètres). L'approche proposée n'est pas spécifique aux gestes de la main et son applicabilité est confirmée pour des tâches de reconnaissances d'actions humaines ou de reconnaissance d'émotions faciales.

Chapitre 5
Génération de Mouvements Humains par Apprentissage Profond

Ce chapitre porte sur la question de la synthèse de mouvements humains avec des réseaux de neurones artificiels. Il est subdivisé en cinq sections :

1. Introduction
2. Revue de littérature sur le sujet
3. Auto-encodeur de pose humaine entraîné avec des triplettes temporelles
4. Réseaux antagonistes génératifs spatio-temporels
5. Conclusion

Introduction Dans ce chapitre, deux nouvelles approches sont proposées pour la synthèse de séquences de poses humaines, c'est-à-dire pour la synthèse de mouvements. Les deux approches sont conçues pour générer des données de poses, et ne nécessitent ni image ni information préalable, comme c'est parfois le cas dans d'autres approches.

La première approche se base sur un auto-encodeur temporellement-contrastif entraîné sur des triplettes de poses individuelles. La seconde approche se base sur des réseaux antagonistes génératifs, les réseaux étant cette fois-ci entraînés à l'échelle de séquences entières.

Revue de littérature sur le sujet Dans la littérature scientifique, deux directions principales ont été explorées pour la synthèse de mouvements de pose humaine via des réseaux de neurones : les modèles basés sur des réseaux récurrents qui ont tendance à séparer la structure spatiale et la structure temporelle, et les modèles génératifs adverses qui les apprennent de manière conjointe. Plusieurs approches tentent notamment de conjuguer des réseaux de neurones récurrents avec des réseaux de neurones auto-encodeurs variationnels, avec un succès néanmoins limité. Dans une autre voie, plusieurs approches sont basées sur des réseaux antagonistes génératifs appliqués à des images couleurs issus d'une caméra et associés ou conditionnés avec les poses humaines correspondantes. Ces approches nécessitent ainsi des données images en plus des données de poses.

Auto-encodeur de pose humaine entraîné avec des triplettes temporelles Cette section propose une nouvelle approche pour apprendre de manière auto-supervisée des représentations de poses humaines dans un espace latent, avec l'aide d'un réseau de neurones de type auto-encodeur débruitant. Un auto-encodeur débruitant apprend à encoder une pose vers un espace latent et à la décoder depuis ce même espace latent. La différence entre un auto-encodeur débruitant et un auto-encodeur simple réside dans le fait que là où auto-encodeur simple apprend à encoder une pose puis à la décoder à l'identique, un auto-encodeur débruitant apprend à décoder une version débruitée de cette pose, c'est-à-dire la même pose qu'en entrée mais sans le bruit.

L'auto-encodeur débruitant proposé est entraîné à minimiser une fonction de coût qui combine un terme de reconstruction, qui assure la cohérence spatiale des poses individuelles, et un terme de triplette temporelle, qui assure une cohérence temporelle entre les poses individuelles.

Après entraînement du réseau de neurones, l'encodeur et le décodeur se révèlent très continus. La synthèse (statique) d'une pose est réalisée en échantillonnant un vecteur aléatoire dans l'espace latent et en décodant celui-ci avec le décodeur. La synthèse (dynamique) d'une séquence de poses est elle aussi réalisée de manière très immédiate : (i) deux vecteurs aléatoires sont d'abord échantillonnés dans l'espace latent, (ii) une séquence de vecteurs latents est obtenue par interpolation linéaire entre ces deux vecteurs, et (iii) chacun de ces vecteurs latents est décodé par le décodeur de façon à aboutir à la séquences de pose souhaitée. Les séquences de poses générées sont cohérentes spatialement et temporellement. À l'œil humain, les séquences de poses générées sont même souvent perçues comme étant plus réalistes que les séquences réelles, mais bruitées, de vérité terrain.

Réseaux antagonistes génératifs spatio-temporels Cette section propose une nouvelle approche basée sur des réseaux antagonistes génératifs, ou *generative adversarial networks* (GAN) en anglais, pour la synthèse de séquences de poses humaines. Pour représenter les séquences de poses, une représentation 2D pouvant servir d'entrée à des convolutions 2D est adoptée. Cette représentation, reprise de la littérature scientifique, s'obtient en parcourant les articulations du corps selon un chemin qui assure -quitte à réordonner les articulations ou à les répéter plusieurs fois- une continuité spatiale entre deux "nouvelles" articulations successives ainsi obtenues. Il fait alors sens de calculer des convolutions sur le tenseur représentant la séquence de pose, et dont les dimensions peuvent être vues par la pensée comme les dimensions d'un tenseur représentant une image. Deux réseaux de neurones, un générateur et un discriminateur, (comme pour tous les réseaux antagonistes génératifs), sont entraînés de manière antagoniste à synthétiser et à discriminer des séquences de pose. L'architecture de chacun de ces deux réseaux est de type convolutif, chaque convolution 2D traitant ainsi de manière conjointe des informations spatiales et temporelles. Les deux réseaux sont conditionnés sur des classes d'actions, dans l'optique de pouvoir synthétiser des actions spécifiques. Une fois les deux réseaux entrainés, les paramètres des réseaux sont gelés. Le générateur est ensuite utilisé pour synthétiser des séquences dans la représentation 2D précédemment évoquée. Les séquences de poses sont ensuite retransformées en leur représentation tensorielle canonique.

Conclusion Deux nouvelles approches différentes pour la synthèse de séquences de poses humaines sont proposées dans ce chapitre.

Dans la première approche, un réseau de neurones auto-encodeur, entraîné avec des triplettes de poses échantillonnées temporellement, découple l'essentiel des informations spatiales et temporelles. La fonction de coût temporel choisie a pour rôle principal d'"organiser" et de "trier" les représentations spatiales de poses d'une manière cohérente d'un point de vue temporel.

Dans la seconde approche, des réseaux antagonistes génératifs spatio-temporels conditionnels

couplent l'information spatiale et temporelle.

De nombreuses extensions des approches proposées sont envisageables; certaines étant même relativement faciles à obtenir. Par exemple, l'inclusion d'une contrainte antagoniste permettrait au modèle auto-encodeur de poses de garantir un réalisme des poses synthétisées encore accru. L'ajout d'un mécanisme de débruitage aux réseaux antagonistes génératifs pourrait lui aussi se révéler pertinent. Enfin, une représentation des classes par un encodage plus judicieux que l'encodage 1 parmi *n*, ou *one-hot encoding* en anglais, pourrait ouvrir la voie à de nouvelles approches.

Chapitre 6
Conclusion

Ce chapitre résume les contributions de cette thèse. Quelques pistes de recherche en lien avec les résultats obtenus dans la thèse sont suggérées. Enfin, on mentionne des perspectives à plus long terme pour l'intelligence artificielle, en particulier celle de l'apprentissage auto-supervisé et celle de l'apprentissage par renforcement.

Bibliography

Amor, Boulbaba Ben, Jingyong Su, and Anuj Srivastava (2015). "Action recognition using rate-invariant analysis of skeletal shape trajectories". In: *IEEE transactions on pattern analysis and machine intelligence* 38.1, pp. 1–13.

Anirudh, Rushil, Pavan Turaga, Jingyong Su, and Anuj Srivastava (2015). "Elastic functional coding of human actions: From vector-fields to latent variables". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3147–3155.

Avola, Danilo, Marco Bernardi, Luigi Cinque, Gian Luca Foresti, and Cristiano Massaroni (2018). "Exploiting recurrent neural networks and leap motion controller for the recognition of sign language and semaphoric hand gestures". In: *IEEE Transactions on Multimedia* 21.1, pp. 234–245.

Ba, Jimmy, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu (2016). "Using fast weights to attend to the recent past". In: *Advances in Neural Information Processing Systems*, pp. 4331–4339.

Baccouche, Moez, Franck Mamalet, Christian Wolf, Christophe Garcia, and Atilla Baskurt (2011). "Sequential deep learning for human action recognition". In: *International workshop on human behavior understanding*. Springer, pp. 29–39.

Badrinarayanan, Vijay, Alex Kendall, and Roberto Cipolla (2017). "Segnet: A deep convolutional encoder-decoder architecture for image segmentation". In: *IEEE transactions on pattern analysis and machine intelligence* 39.12, pp. 2481–2495.

Bagnall, A, A Bostrom, J Large, and J Lines (n.d.). "The great time series classification bake off: an experimental evaluation of recently proposed algorithms". In: *Extended Version. CoRR, abs* 1602 ().

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2014). "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473*.

Bai, Shaojie, J Zico Kolter, and Vladlen Koltun (2018). "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling". In: *arXiv preprint arXiv:1803.01271*.

Balakrishnan, Guha, Amy Zhao, Adrian V Dalca, Fredo Durand, and John Guttag (2018). "Synthesizing images of humans in unseen poses". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8340–8348.

Baltrusaitis, Tadas, Amir Zadeh, Yao Chong Lim, and Louis-Philippe Morency (2018). "Openface 2.0: Facial behavior analysis toolkit". In: *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*. IEEE, pp. 59–66.

Baradel, Fabien, Christian Wolf, and Julien Mille (Oct. 2017a). "Human Action Recognition: Pose-Based Attention Draws Focus to Hands". In: *The IEEE International Conference on Computer Vision (ICCV)*.

Baradel, Fabien, Christian Wolf, and Julien Mille (2017b). “Pose-conditioned Spatio-Temporal Attention for Human Action Recognition”. In: *arXiv preprint arXiv:1703.10106*.

Baradel, Fabien, Christian Wolf, and Julien Mille (Sept. 2018). “Human Activity Recognition with Pose-driven Attention to RGB”. In: *BMVC 2018 - 29th British Machine Vision Conference*. Newcastle, United Kingdom, pp. 1–14. URL: https://hal.inria.fr/hal-01828083.

Barsoum, Emad, John Kender, and Zicheng Liu (2018). “HP-GAN: Probabilistic 3D human motion prediction via GAN”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1418–1427.

Ben Tanfous, Amor, Hassen Drira, and Boulbaba Ben Amor (2018). “Coding Kendall’s Shape Trajectories for 3D Action Recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2840–2849.

Bogo, Federica, Angjoo Kanazawa, Christoph Lassner, Peter Gehler, Javier Romero, and Michael J Black (2016). “Keep it SMPL: Automatic estimation of 3D human pose and shape from a single image”. In: *European Conference on Computer Vision*. Springer, pp. 561–578.

Bojarski, Mariusz, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba (Apr. 2016). “End to End Learning for Self-Driving Cars”. In: *arXiv:1604*, pp. 1–9. arXiv: 1604.07316. URL: http://arxiv.org/abs/1604.07316.

Bollerslev, Tim (1986). “Generalized autoregressive conditional heteroskedasticity”. In: *Journal of econometrics* 31.3, pp. 307–327.

Boulahia, Said Yacine, Eric Anquetil, Franck Multon, and Richard Kulpa (2017). “Dynamic hand gesture recognition based on 3D pattern assembled trajectories”. In: *2017 Seventh International Conference on Image Processing Theory, Tools and Applications (IPTA)*. IEEE, pp. 1–6.

Boureau, Y-Lan, Jean Ponce, and Yann LeCun (2010). “A theoretical analysis of feature pooling in visual recognition”. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 111–118.

Box, GEORGE EP, Gwilym M Jenkins, and G Reinsel (1970). “Time series analysis: forecasting and control Holden-day San Francisco”. In: *BoxTime Series Analysis: Forecasting and Control Holden Day1970*.

Brown, Tom B, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. (2020). “Language models are few-shot learners”. In: *arXiv preprint arXiv:2005.14165*.

Bütepage, Judith, Hedvig Kjellström, and Danica Kragic (2018). “A Probabilistic Semi-Supervised Approach to Multi-Task Human Activity Modeling”. In: *arXiv preprint arXiv:1809.08875*.

Caputo, Fabio M, Pietro Prebianca, Alessandro Carcangiu, Lucio D Spano, and Andrea Giachetti (2018). “Comparing 3D trajectories for simple mid-air gesture recognition”. In: *Computers & Graphics* 73, pp. 17–25.

Castelvecchi, Davide (2016). “Can we open the black box of AI?” In: *Nature News* 538.7623, p. 20.

Chaudhry, Rizwan, Ferda Ofli, Gregorij Kurillo, Ruzena Bajcsy, and Rene Vidal (2013). “Bio-inspired dynamic 3d discriminative skeletal features for human action recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 471–478.

Chen, Xinghao, Guijin Wang, Hengkai Guo, Cairong Zhang, Hang Wang, and Li Zhang (2019). "MFA-Net: Motion Feature Augmented Network for Dynamic Hand Gesture Recognition from Skeletal Data". In: *Sensors* 19.2, p. 239.

Chen, Yuxiao, Ayonga Hereid, Huei Peng, and Jessy Grizzle (Dec. 2017). "Synthesis of safe controller via supervised learning for truck lateral control". In: *arXiv* December, pp. 1–13. arXiv: 1712.05506. URL: http://arxiv.org/abs/1712.05506.

Cho, Kyunghyun, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014). "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078*.

Chollet, François (2017). "Xception: Deep learning with depthwise separable convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1251–1258.

Chrungoo, Addwiteey, SS Manimaran, and Balaraman Ravindran (2014). "Activity recognition for natural human robot interaction". In: *International Conference on Social Robotics*. Springer, pp. 84–94.

Chung, Junyoung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio (2015). "A recurrent latent variable model for sequential data". In: *Advances in neural information processing systems*, pp. 2980–2988.

Cippitelli, Enea, Samuele Gasparrini, Ennio Gambi, and Susanna Spinsante (2016). "A human activity recognition system using skeleton data from rgbd sensors". In: *Computational intelligence and neuroscience* 2016, p. 21.

Clevert, Djork-Arné, Thomas Unterthiner, and Sepp Hochreiter (2015). "Fast and accurate deep network learning by exponential linear units (elus)". In: *arXiv preprint arXiv:1511.07289*.

Cooijmans, Tim, Nicolas Ballas, César Laurent, Çağlar Gülçehre, and Aaron Courville (2016). "Recurrent batch normalization". In: *arXiv preprint arXiv:1603.09025*.

Coppola, Claudio, O Martinez Mozos, and Nicola Bellotto (2015). "Applying a 3d qualitative trajectory calculus to human action recognition using depth cameras". In: *IEEE/RSJ IROS workshop on assistance and service robotics in a human environment*. IEEE.

Coulter, R Craig (1992). *Implementation of the Pure Pursuit Path Tracking Algorithm*. Carnegie Mellon University.

Cui, Rongxin, Chenguang Yang, Yang Li, and Sanjay Sharma (June 2017). "Adaptive Neural Network Control of AUVs With Control Input Nonlinearities Using Reinforcement Learning". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 47.6, pp. 1019–1029. ISSN: 2168-2216. DOI: 10.1109/TSMC.2016.2645699. URL: http://ieeexplore.ieee.org/document/7812772/.

Cuturi, Marco and Mathieu Blondel (2017). "Soft-DTW: a differentiable loss function for time-series". In: *arXiv preprint arXiv:1703.01541*.

Dai, Jifeng, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei (2017). "Deformable convolutional networks". In: *Proceedings of the IEEE international conference on computer vision*, pp. 764–773.

De Smedt, Quentin, Hazem Wannous, and Jean-Philippe Vandeborre (2016). "Skeleton-based dynamic hand gesture recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1–9.

De Smedt, Quentin, Hazem Wannous, Jean-Philippe Vandeborre, Joris Guerry, Bertrand Le Saux, and David Filliat (2017). "SHREC'17 Track: 3D Hand Gesture Recognition Using a Depth and Skeletal Dataset". In: *10th Eurographics Workshop on 3D Object Retrieval*.

Deng, Jiankang, Jia Guo, Niannan Xue, and Stefanos Zafeiriou (2019). "Arcface: Additive angular margin loss for deep face recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4690–4699.

Dermy, Oriane, Maxime Chaveroche, Francis Colas, François Charpillet, and Serena Ivaldi (2018). "Prediction of Human Whole-Body Movements with AE-ProMPs". In: *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*. IEEE, pp. 572–579.

Devanne, Maxime, Hazem Wannous, Stefano Berretti, Pietro Pala, Mohamed Daoudi, and Alberto Del Bimbo (2014). "3-d human action recognition by shape analysis of motion trajectories on riemannian manifold". In: *IEEE transactions on cybernetics* 45.7, pp. 1340–1352.

Devineau, Guillaume, Fabien Moutarde, Wang Xi, and Jie Yang (2018a). "Convolutional neural networks for multivariate time series classification using both inter-and intra-channel parallel convolutions". In: *2018 RFIAP (Reconnaissance des Formes, Image, Apprentissage et Perception) conference, (RFIAP 2018)*. RFIAP.

Devineau, Guillaume, Fabien Moutarde, Wang Xi, and Jie Yang (2018b). "Deep learning for hand gesture recognition on skeletal data". In: *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*. IEEE, pp. 106–113.

Devineau, Guillaume, Philip Polack, Florent Altché, and Fabien Moutarde (2018c). "Coupled longitudinal and lateral control of a vehicle using deep learning". In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, pp. 642–649.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2018). "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805*.

Ding, Wenwen, Kai Liu, Fei Cheng, and Jin Zhang (2015). "STFC: Spatio-temporal feature chain for skeleton-based human action recognition". In: *Journal of Visual Communication and Image Representation* 26, pp. 329–337.

Donahue, Jeffrey, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell (2015). "Long-term recurrent convolutional networks for visual recognition and description". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2625–2634.

Dong, Zhen, Yuwei Wu, Mingtao Pei, and Yunde Jia (2015). "Vehicle type classification using a semisupervised convolutional neural network". In: *IEEE transactions on intelligent transportation systems* 16.4, pp. 2247–2256.

Drews, Paul, Grady Williams, Brian Goldfain, Evangelos A Theodorou, and James M Rehg (July 2017). "Aggressive Deep Driving: Model Predictive Control with a CNN Cost Model". In: *Proceedings of the 1st Annual Conference on Robot Learning* 78.CoRL, pp. 133–142. ISSN: 1938-7228. arXiv: 1707.05303.

Du, Yong, Wei Wang, and Liang Wang (2015). "Hierarchical recurrent neural network for skeleton based action recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1110–1118.

Duan, Yueqi, Wenzhao Zheng, Xudong Lin, Jiwen Lu, and Jie Zhou (2018). "Deep adversarial metric learning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2780–2789.

Dumoulin, Vincent and Francesco Visin (2016). "A guide to convolution arithmetic for deep learning". In: *arXiv preprint arXiv:1603.07285*.

Elman, Jeffrey L (1990). "Finding structure in time". In: *Cognitive science* 14.2, pp. 179–211.

Evangelidis, Georgios, Gurkirt Singh, and Radu Horaud (2014). "Skeletal quads: Human action recognition using joint quadruples". In: *2014 22nd International Conference on Pattern Recognition*. IEEE, pp. 4513–4518.

Falcone, Paolo, Francesco Borrelli, Jahan Asgari, Hongtei Eric Tseng, and Davor Hrovat (2007). "Predictive active steering control for autonomous vehicle systems". In: *IEEE Transactions on Control Systems Technology* 15.3, pp. 566–580. ISSN: 10636536.

Falkner, Stefan, Aaron Klein, and Frank Hutter (2018). "BOHB: Robust and efficient hyperparameter optimization at scale". In: *arXiv preprint arXiv:1807.01774*.

Fan, Zhaoxuan, Xu Zhao, Tianwei Lin, and Haisheng Su (2018). "Attention-Based Multiview Re-Observation Fusion Network for Skeletal Action Recognition". In: *IEEE Transactions on Multimedia* 21.2, pp. 363–374.

Fragkiadaki, Katerina, Sergey Levine, Panna Felsen, and Jitendra Malik (2015). "Recurrent network models for human dynamics". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4346–4354.

Gal, Yarin and Zoubin Ghahramani (2016). "A theoretically grounded application of dropout in recurrent neural networks". In: *Advances in neural information processing systems*, pp. 1019–1027.

Gers, Felix A and E Schmidhuber (2001). "LSTM recurrent networks learn simple context-free and context-sensitive languages". In: *IEEE Transactions on Neural Networks* 12.6, pp. 1333–1340.

Gharghabi, Shaghayegh, Shima Imani, Anthony Bagnall, Amirali Darvishzadeh, and Eamonn Keogh (2018). "Matrix Profile XII: MPdist: A novel time series distance measure to allow data mining in more challenging scenarios". In: *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, pp. 965–970.

Ghil, Michael, MR. Allen, MD. Dettinger, K. Ide, D. Kondrashov, ME. Mann, Andrew W. Robertson, A. Saunders, Y. Tian, F. Varadi, and P. Yiou (2002). "Advanced spectral methods for climatic time series". In: *Reviews of geophysics* 40.1, pp. 3–1.

Ghorbani, Amirata, Abubakar Abid, and James Zou (2019). "Interpretation of neural networks is fragile". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33, pp. 3681–3688.

Ghosh, Partha, Jie Song, Emre Aksan, and Otmar Hilliges (2017). "Learning human motion models for long-term predictions". In: *2017 International Conference on 3D Vision (3DV)*. IEEE, pp. 458–466.

Gillespie, Thomas D (1997). "Vehicle dynamics". In: *Warren dale*.

Glorot, Xavier and Yoshua Bengio (2010). “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256.

Goh, Jonathan Y and J Christian Gerdes (June 2016). “Simultaneous stabilization and tracking of basic automobile drifting trajectories”. In: *2016 IEEE Intelligent Vehicles Symposium (IV)*. Iv. IEEE, pp. 597–602. ISBN: 978-1-5090-1821-5. DOI: 10.1109/IVS.2016.7535448. URL: http://ieeexplore.ieee.org/document/7535448/.

Goldberger, Jacob, Geoffrey E Hinton, Sam T Roweis, and Russ R Salakhutdinov (2005). “Neighbourhood components analysis”. In: *Advances in neural information processing systems*, pp. 513–520.

Golyandina, Nina, Vladimir Nekrutkin, and Anatoly A Zhigljavsky (2001). *Analysis of time series structure: SSA and related techniques*. CRC press.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio (2014). “Generative adversarial nets”. In: *Advances in neural information processing systems*, pp. 2672–2680.

Gower, John C (1975). “Generalized procrustes analysis”. In: *Psychometrika* 40.1, pp. 33–51.

Granger, Nicolas (2019). “Deep-learning for high dimensional sequential observations: application to continuous gesture recognition: Modélisation par réseaux de neurones profonds pour l’apprentissage continu d’objets et de gestes par un robot”. PhD thesis. Paris Saclay.

Graves, Alex (2016). “Adaptive computation time for recurrent neural networks”. In: *arXiv preprint arXiv:1603.08983*.

Graves, Alex, Santiago Fernández, and Jürgen Schmidhuber (2007). “Multi-dimensional recurrent neural networks”. In: *International conference on artificial neural networks*. Springer, pp. 549–558.

Graves, Alex, Greg Wayne, and Ivo Danihelka (2014). “Neural turing machines”. In: *arXiv preprint arXiv:1410.5401*.

Graves, Alex, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomenech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis (2016). “Hybrid computing using a neural network with dynamic external memory”. In: *Nature* 538.7626, p. 471.

Greff, Klaus, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber (2016). “LSTM: A search space odyssey”. In: *IEEE transactions on neural networks and learning systems* 28.10, pp. 2222–2232.

Hadsell, Raia, Sumit Chopra, and Yann LeCun (2006). “Dimensionality reduction by learning an invariant mapping”. In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06)*. Vol. 2. IEEE, pp. 1735–1742.

Han, Fei, Brian Reily, William Hoff, and Hao Zhang (2017). “Space-time representation of people based on 3D skeletal data: A review”. In: *Computer Vision and Image Understanding* 158, pp. 85–105.

Harvey, Félix G, Julien Roy, David Kanaa, and Christopher Pal (2018). "Recurrent semi-supervised classification and constrained adversarial generation with motion capture data". In: *Image and Vision Computing* 78, pp. 42–52.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

He, Zhihao, Tian Jin, Amlan Basu, John Soraghan, Gaetano Di Caterina, and Lykourgos Petropoulakis (2019). "Human emotion recognition in video using subtraction pre-processing". In: *Proceedings of the 2019 11th International Conference on Machine Learning and Computing*. ACM, pp. 374–379.

Ho, Jonathan, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel (2019). "Flow++: Improving flow-based generative models with variational dequantization and architecture design". In: *arXiv preprint arXiv:1902.00275*.

Hochreiter, Sepp and Jürgen Schmidhuber (Nov. 1997a). "Long Short-Term Memory". In: *Neural Comput.* 9.8, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Hochreiter, Sepp and Jürgen Schmidhuber (1997b). "LSTM can solve hard long time lag problems". In: *Advances in neural information processing systems*, pp. 473–479.

Hornik, Kurt (1991). "Approximation capabilities of multilayer feedforward networks". In: *Neural networks* 4.2, pp. 251–257.

Hou, Jingxuan, Guijin Wang, Xinghao Chen, Jing-Hao Xue, Rui Zhu, and Huazhong Yang (2018). "Spatial-temporal attention res-TCN for skeleton-based dynamic hand gesture recognition". In: *Proceedings of the European Conference on Computer Vision (ECCV)*.

Hu, Jian-Fang, Wei-Shi Zheng, Jianhuang Lai, and Jianguo Zhang (2015). "Jointly learning heterogeneous features for RGB-D activity recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5344–5352.

Huang, Gao, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger (2017). "Densely connected convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708.

Huang, Yewen, Suian Zhang, Haifeng Hu, Dihu Chen, and Tao Su (2019). "Resetting-Label Network Based on Fast Group Loss for Person Re-Identification". In: *IEEE Access* 7, pp. 119486–119496.

Huang, Zhongyue, Jingwei Xu, and Bingbing Ni (2018). "Human Motion Generation via Cross-Space Constrained Sampling." In: *IJCAI*, pp. 757–763.

Hussein, Mohamed E, Marwan Torki, Mohammad A Gowayyed, and Motaz El-Saban (2013). "Human action recognition using a temporal hierarchy of covariance descriptors on 3d joint locations". In: *Twenty-Third International Joint Conference on Artificial Intelligence*.

Ionescu, Bogdan, Didier Coquin, Patrick Lambert, and Vasile Buzuloiu (2005). “Dynamic hand gesture recognition using the skeleton of the hand”. In: *EURASIP Journal on Advances in Signal Processing* 2005.13, p. 236190.

Jaeger, Herbert (2002). *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*. Vol. 5. GMD-Forschungszentrum Informationstechnik Bonn.

Jaeger, Herbert and Harald Haas (2004). “Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication”. In: *science* 304.5667, pp. 78–80.

Jain, Anil K and Nicolae Duta (1999). “Deformable matching of hand shapes for user verification”. In: *Proceedings 1999 International Conference on Image Processing (Cat. 99CH36348)*. Vol. 2. IEEE, pp. 857–861.

Jain, Suyog, Changbo Hu, and Jake K Aggarwal (2011). “Facial expression recognition with temporal modeling of shapes”. In: *2011 IEEE international conference on computer vision workshops (ICCV workshops)*. IEEE, pp. 1642–1649.

Jin, Sou-Young and Ho-Jin Choi (2012). “Essential body-joint and atomic action detection for human activity recognition using longest common subsequence algorithm”. In: *Asian Conference on Computer Vision*. Springer, pp. 148–159.

Johansson, Gunnar (1973). “Visual perception of biological motion and a model for its analysis”. In: *Perception & psychophysics* 14.2, pp. 201–211.

Kalchbrenner, Nal, Ivo Danihelka, and Alex Graves (2015). “Grid long short-term memory”. In: *arXiv preprint arXiv:1507.01526*.

Kaya, Mahmut and Hasan Şakir Bilge (2019). “Deep metric learning: a survey”. In: *Symmetry* 11.9, p. 1066.

Ke, Qiuhong, Senjian An, Mohammed Bennamoun, Ferdous Sohel, and Farid Boussaid (2017a). “Skeletonnet: Mining deep part features for 3-d action recognition”. In: *IEEE signal processing letters* 24.6, pp. 731–735.

Ke, Qiuhong, Mohammed Bennamoun, Senjian An, Ferdous Sohel, and Farid Boussaid (2017b). “A new representation of skeleton sequences for 3d action recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3288–3297.

Kendall, David G (1984). “Shape manifolds, procrustean metrics, and complex projective spaces”. In: *Bulletin of the London Mathematical Society* 16.2, pp. 81–121.

Keskar, Nitish Shirish, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang (2016). “On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima”. In: arXiv: 1609.04836 [cs.LG].

Khodayari, Alireza, Ali Ghaffari, Sina Ameli, and Jamal Flahatgar (2010). “A historical review on lateral and longitudinal control of autonomous vehicle motions”. In: *ICMET 2010 - 2010 International Conference on Mechanical and Electrical Technology, Proceedings* Icmet, pp. 421–429. ISSN: 9781424481019. DOI: 10.1109/ICMET.2010.5598396.

Kiasari, Mohammad Ahangar, Dennis Singh Moirangthem, and Minho Lee (2018). “Human action generation with generative adversarial networks”. In: *arXiv preprint arXiv:1805.10416*.

Kingma, Diederik and Jimmy Ba (2014). “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980*.

Kingma, Durk P and Prafulla Dhariwal (2018). “Glow: Generative flow with invertible 1x1 convolutions”. In: *Advances in Neural Information Processing Systems*, pp. 10215–10224.

Klaser, Alexander, Marcin Marszałek, and Cordelia Schmid (2008). “A spatio-temporal descriptor based on 3d-gradients”. In: *BMVC 2008-19th British Machine Vision Conference*. British Machine Vision Association, pp. 275–1.

Knopp, Jan, Mukta Prasad, Geert Willems, Radu Timofte, and Luc Van Gool (2010). “Hough transform and 3D SURF for robust three dimensional classification”. In: *Computer vision–ECCV 2010*, pp. 589–602.

Kondor, Risi and Shubhendu Trivedi (2018). “On the generalization of equivariance and convolution in neural networks to the action of compact groups”. In: *arXiv preprint arXiv:1802.03690*.

Koutnik, Jan, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber (2014). “A clockwork rnn”. In: *arXiv preprint arXiv:1402.3511*.

Kritayakirana, Krisada and J. Christian Gerdes (2012). “Autonomous vehicle control at the limits of handling”. In: *International Journal of Vehicle Autonomous Systems* 10.4, p. 271. ISSN: 1471-0226. DOI: 10.1504/IJVAS.2012.051270.

Krueger, David, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Aaron Courville, and Chris Pal (2016). “Zoneout: Regularizing rnns by randomly preserving hidden activations”. In: *arXiv preprint arXiv:1606.01305*.

Krzanowski, WJ (2000). “Principles of Multivariate Analysis, Revised Edition”. In: *Oxford Statistical Science Series* 23.

Lafferty, John, Andrew McCallum, and Fernando CN Pereira (2001). “Conditional random fields: Probabilistic models for segmenting and labeling sequence data”. In:

Längkvist, Martin, Lars Karlsson, and Amy Loutfi (2014). “A review of unsupervised feature learning and deep learning for time-series modeling”. In: *Pattern Recognition Letters* 42, pp. 11–24.

Le, Quoc V. (2015). “A tutorial on deep learning part 2: Autoencoders, convolutional neural networks and recurrent neural networks”. In: *Google Brain*, pp. 1–20.

Lea, Colin, Michael D Flynn, Rene Vidal, Austin Reiter, and Gregory D Hager (2017). “Temporal convolutional networks for action segmentation and detection”. In: *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 156–165.

Lee, Hsin-Ying, Xiaodong Yang, Ming-Yu Liu, Ting-Chun Wang, Yu-Ding Lu, Ming-Hsuan Yang, and Jan Kautz (2019). “Dancing to Music”. In: *Advances in Neural Information Processing Systems*, pp. 3581–3591.

Lee, Inwoong, Doyoung Kim, Seoungyoon Kang, and Sanghoon Lee (2017). “Ensemble deep learning for skeleton-based action recognition using temporal sliding lstm networks”. In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1012–1020.

Lefebvre, Grégoire, Samuel Berlemont, Franck Mamalet, and Christophe Garcia (2013). “BLSTM-RNN based 3D gesture classification”. In: *International conference on artificial neural networks*. Springer, pp. 381–388.

Leong, Julian, Marios Nicolaou, Louis Atallah, George Mylonas, Ara Darzi, and Guang-Zhong Yang (2006). “HMM assessment of quality of movement trajectory in laparoscopic surgery”. In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2006*, pp. 752–759.

Li, Chenyang, Xin Zhang, Lufan Liao, Lianwen Jin, and Weixin Yang (2019). "Skeleton-Based Gesture Recognition Using Several Fully Connected Layers with Path Signature Features and Temporal Transformer Module". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33, pp. 8585–8593.

Li, Hao, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein (2018). "Visualizing the loss landscape of neural nets". In: *Advances in Neural Information Processing Systems*, pp. 6389–6399.

Li, Lisha, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar (2016). "Hyperband: A novel bandit-based approach to hyperparameter optimization". In: *arXiv preprint arXiv:1603.06560*.

Li, Ruotong, Weixin Si, Michael Weinmann, and Reinhard Klein (2019). "Constraint-based optimized human skeleton extraction from single-depth camera". In: *Sensors* 19.11, p. 2604.

Li, Yong, Zihang He, Xiang Ye, Zuguo He, and Kangrong Han (2019). "Spatial temporal graph convolutional networks for skeleton-based dynamic hand gesture recognition". In: *EURASIP Journal on Image and Video Processing* 2019.1, p. 78.

Lillo, Ivan, Juan Carlos Niebles, and Alvaro Soto (2016). "A hierarchical pose-based approach to complex action understanding using dictionaries of actionlets and motion poselets". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1981–1990.

Lin, Jessica, Eamonn Keogh, Li Wei, and Stefano Lonardi (2007). "Experiencing SAX: a novel symbolic representation of time series". In: *Data Mining and knowledge discovery* 15.2, pp. 107–144.

Liu, Jun, Amir Shahroudy, Dong Xu, and Gang Wang (2016). "Spatio-temporal lstm with trust gates for 3d human action recognition". In: *European Conference on Computer Vision*. Springer, pp. 816–833.

Liu, Jun, Gang Wang, Ling-Yu Duan, Kamila Abdiyeva, and Alex C Kot (2017). "Skeleton-based human action recognition with global context-aware attention LSTM networks". In: *IEEE Transactions on Image Processing* 27.4, pp. 1586–1599.

Liu, Mengyuan, Hong Liu, and Chen Chen (2017). "Enhanced skeleton visualization for view invariant human action recognition". In: *Pattern Recognition* 68, pp. 346–362.

Livingstone, Steven R and Frank A Russo (2018). "The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS): A dynamic, multimodal set of facial and vocal expressions in North American English". In: *PloS one* 13.5, e0196391.

Long, Jonathan, Evan Shelhamer, and Trevor Darrell (2015). "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440.

Lugaresi, Camillo, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, Wan-Teh Chang, Wei Hua, Manfred Georg, and Matthias Grundmann (2019). "MediaPipe: A Framework for Perceiving and Augmenting Reality". In:

Ma, Liqian, Xu Jia, Qianru Sun, Bernt Schiele, Tinne Tuytelaars, and Luc Van Gool (2017). "Pose guided person image generation". In: *Advances in Neural Information Processing Systems*, pp. 406–416.

Maghoumi, Mehran and Joseph J LaViola Jr (2018). "DeepGRU: Deep Gesture Recognition Utility". In: *arXiv preprint arXiv:1810.12514*.

Martinez, Julieta, Michael J Black, and Javier Romero (2017a). "On human motion prediction using recurrent neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2891–2900.

Martinez, Julieta, Rayat Hossain, Javier Romero, and James J Little (2017b). "A simple yet effective baseline for 3d human pose estimation". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2640–2649.

Martins, Pedro Alexandre Dias (2008). "Active appearance models for facial expression recognition and monocular head pose estimation". MA thesis. University of Coimbra.

Merity, Stephen, Nitish Shirish Keskar, and Richard Socher (2017). "Regularizing and optimizing LSTM language models". In: *arXiv preprint arXiv:1708.02182*.

Minderer, Matthias, Chen Sun, Ruben Villegas, Forrester Cole, Kevin P Murphy, and Honglak Lee (2019). "Unsupervised learning of object structure and dynamics from videos". In: *Advances in Neural Information Processing Systems*, pp. 92–102.

Mishra, Nikhil, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel (2017). "A simple neural attentive meta-learner". In: *arXiv preprint arXiv:1707.03141*.

Miyato, Takeru, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida (2018). "Spectral normalization for generative adversarial networks". In: *arXiv preprint arXiv:1802.05957*.

Moreno-Noguer, Francesc (2017). "3d human pose estimation from a single image via distance matrix regression". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2823–2832.

Musgrave, Kevin, Serge Belongie, and Ser-Nam Lim (2020). "A metric learning reality check". In: *arXiv preprint arXiv:2003.08505*.

Neverova, Natalia (2016a). "Deep Learning for Human Motion Analysis". In: *PhD thesis*.

Neverova, Natalia (2016b). "Deep learning for human motion analysis". PhD thesis. INSA Lyon.

Neverova, Natalia, Christian Wolf, Graham Taylor, and Florian Nebout (2015). "Moddrop: adaptive multi-modal gesture recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.8, pp. 1692–1706.

Nguyen, Xuan Son, Luc Brun, Olivier Lézoray, and Sébastien Bougleux (2019). "A neural network based on SPD manifold learning for skeleton-based hand gesture recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 12036–12045.

Nguyen, Xuan, Luc Brun, Olivier Lezoray, and Sébastien Bougleux (2019). "Skeleton-Based Hand Gesture Recognition by Learning SPD Matrices with Neural Networks". In: *arXiv preprint arXiv:1905.07917*.

Nunez, Juan C, Raul Cabido, Juan J Pantrigo, Antonio S Montemayor, and Jose F Velez (2018). "Convolutional neural networks and long short-term memory for skeleton-based human activity and hand gesture recognition". In: *Pattern Recognition* 76, pp. 80–94.

Ofli, Ferda, Rizwan Chaudhry, Gregorij Kurillo, René Vidal, and Ruzena Bajcsy (2014). "Sequence of the most informative joints (smij): A new representation for human skeletal action recognition". In: *Journal of Visual Communication and Image Representation* 25.1, pp. 24–38.

Ohn-Bar, Eshed and Mohan Trivedi (2013). "Joint angles similarities and HOG2 for action recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 465–470.

Olah, Christopher (2015). *Understanding lstm networks, 2015*. https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed: 09-August-2020].

Oord, Aaron van den, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu (2016). "Wavenet: A generative model for raw audio". In: *arXiv preprint arXiv:1609.03499*.

Oreifej, Omar and Zicheng Liu (2013). "Hon4d: Histogram of oriented 4d normals for activity recognition from depth sequences". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 716–723.

Pacejka, Hans B. (2002). *Tyre and Vehicle Dynamics*. Butterworth-Heinemann.

Pan, Yingwei, Yehao Li, Ting Yao, Tao Mei, Houqiang Li, and Yong Rui (2016). "Learning Deep Intrinsic Video Representation by Exploring Temporal Coherence and Graph Structure." In: *IJCAI*, pp. 3832–3838.

Pavlakos, Georgios, Vasileios Choutas, Nima Ghorbani, Timo Bolkart, Ahmed AA Osman, Dimitrios Tzionas, and Michael J Black (2019). "Expressive body capture: 3d hands, face, and body from a single image". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 10975–10985.

Pavllo, Dario, David Grangier, and Michael Auli (2018). "Quaternet: A quaternion-based recurrent model for human motion". In: *arXiv preprint arXiv:1805.06485*.

Pigou, Lionel, Aäron van den Oord, Sander Dieleman, Mieke Van Herreweghe, and Joni Dambre (2015). "Beyond Temporal Pooling: Recurrence and Temporal Convolutions for Gesture Recognition in Video". In: *arXiv preprint arXiv:1506.01911*.

Polack, Philip, Florent Altché, Brigitte d'Andréa-Novel, and Arnaud de La Fortelle (2018). "Guaranteeing Consistency in a Motion Planning and Control Architecture Using a Kinematic Bicycle Model". In: *American Control Conference*. Milwaukee, WI, United-States. URL: https://arxiv.org/pdf/1804.08290.pdf.

Polack, Philip, Florent Altché, Brigitte d'Andréa-Novel, and Arnaud de La Fortelle (2017). "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?" In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, pp. 812–818.

Punjani, Ali and Pieter Abbeel (May 2015). "Deep learning helicopter dynamics models". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 2015-June. June. IEEE, pp. 3223–3230. ISBN: 978-1-4799-6923-4. DOI: 10.1109/ICRA.2015.7139643. URL: http://ieeexplore.ieee.org/document/7139643/.

Raaj, Yaadhav, Haroon Idrees, Gines Hidalgo, and Yaser Sheikh (2019). "Efficient Online Multi-Person 2D Pose Tracking with Recurrent Spatio-Temporal Affinity Fields". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4620–4628.

Rabiner, Lawrence R (1989). "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE* 77.2, pp. 257–286.

Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever (2019). "Language models are unsupervised multitask learners". In: *OpenAI Blog* 1.8, p. 9.

Rajamani, Rajesh (2012). *Vehicle Dynamics and Control*. Springer. ISBN: 9781461414322.

Ramachandran, Prajit, Barret Zoph, and Quoc V Le (2017). "Swish: a self-gated activation function". In: *arXiv preprint arXiv:1710.05941* 7.

Ramakrishna, Varun, Takeo Kanade, and Yaser Sheikh (2012). "Reconstructing 3d human pose from 2d image landmarks". In: *European conference on computer vision*. Springer, pp. 573–586.

Reddy, K Sivarajesh, P Swarna Latha, and M Rajasekhara Babu (2011). "Hand gesture recognition using skeleton of hand and distance based metric". In: *International Conference on Advances in Computing and Information Technology*. Springer, pp. 346–354.

Redondo-Cabrera, Carolina and Roberto Lopez-Sastre (2019). "Unsupervised learning from videos using temporal coherency deep networks". In: *Computer Vision and Image Understanding* 179, pp. 79–89.

Rivals, I., D. Canas, L. Personnaz, and G. Dreyfus (1994). "Modeling and control of mobile robots and intelligent vehicles by neural networks". In: *Proceedings of the Intelligent Vehicles '94 Symposium*. IEEE, pp. 137–142. ISBN: 0-7803-2135-9. DOI: 10.1109/IVS.1994.639489. URL: http://ieeexplore.ieee.org/document/639489/.

Sakoe, Hiroaki and Seibi Chiba (1978). "Dynamic programming algorithm optimization for spoken word recognition". In: *IEEE transactions on acoustics, speech, and signal processing* 26.1, pp. 43–49.

Salimans, Tim and Durk P Kingma (2016). "Weight normalization: A simple reparameterization to accelerate training of deep neural networks". In: *Advances in neural information processing systems*, pp. 901–909.

Schmidhuber, Jürgen (1992). "Learning complex, extended sequences using the principle of history compression". In: *Neural Computation* 4.2, pp. 234–242.

Schuster, Mike and Kuldip K Paliwal (1997). "Bidirectional recurrent neural networks". In: *IEEE transactions on Signal Processing* 45.11, pp. 2673–2681.

Semeniuta, Stanislau, Aliaksei Severyn, and Erhardt Barth (2016). "Recurrent dropout without memory loss". In: *arXiv preprint arXiv:1603.05118*.

Sermanet, Pierre, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain (2018). "Time-contrastive networks: Self-supervised learning from video". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 1134–1141.

Shahroudy, Amir, Jun Liu, Tian-Tsong Ng, and Gang Wang (2016). "Ntu rgb+ d: A large scale dataset for 3d human activity analysis". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1010–1019.

Shrikumar, Avanti, Peyton Greenside, and Anshul Kundaje (2017). "Learning important features through propagating activation differences". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 3145–3153.

Si, Chenyang, Ya Jing, Wei Wang, Liang Wang, and Tieniu Tan (2018). "Skeleton-based action recognition with spatial reasoning and temporal stack learning". In: *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 103–118.

Siarohin, Aliaksandr, Stéphane Lathuilière, Enver Sangineto, and Nicu Sebe (2019). "Appearance and Pose-Conditioned Human Image Generation using Deformable GANs". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Slama, Rim, Hazem Wannous, Mohamed Daoudi, and Anuj Srivastava (2015). "Accurate 3D action recognition using learning on the Grassmann manifold". In: *Pattern Recognition* 48.2, pp. 556–567.

Song, Sijie, Cuiling Lan, Junliang Xing, Wenjun Zeng, and Jiaying Liu (2017). "An end-to-end spatio-temporal attention model for human action recognition from skeleton data". In: *Thirty-first AAAI conference on artificial intelligence*.

Srivastava, Nitish, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). "Dropout: a simple way to prevent neural networks from overfitting." In: *Journal of machine learning research* 15.1, pp. 1929–1958.

Srivastava, Rupesh Kumar, Klaus Greff, and Jürgen Schmidhuber (2015). "Highway networks". In: *arXiv preprint arXiv:1505.00387*.

Starner, Thad and Alex Pentland (1997). "Real-time american sign language recognition from video using hidden markov models". In: *Motion-Based Recognition*. Springer, pp. 227–243.

Sun, Jennifer J, Jiaping Zhao, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, and Ting Liu (2020). "View-Invariant Probabilistic Embedding for Human Pose". In: *ECCV*.

Sundararajan, Mukund, Ankur Taly, and Qiqi Yan (2017). "Axiomatic attribution for deep networks". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 3319–3328.

Sutskever, Ilya, Oriol Vinyals, and Quoc V Le (2014). "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*, pp. 3104–3112.

Sutton, Charles and Andrew McCallum (2006). "An introduction to conditional random fields for relational learning". In: *Introduction to statistical relational learning* 2, pp. 93–128.

Tallec, Corentin and Yann Ollivier (2018). "Can recurrent neural networks warp time?" In: *arXiv preprint arXiv:1804.11188*.

Tang, Hao, Wei Wang, Dan Xu, Yan Yan, and Nicu Sebe (2018). "Gesturegan for hand gesture-to-gesture translation in the wild". In: *Proceedings of the 26th ACM international conference on Multimedia*, pp. 774–782.

Tang, Yansong, Yi Tian, Jiwen Lu, Peiyang Li, and Jie Zhou (2018). "Deep progressive reinforcement learning for skeleton-based action recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5323–5332.

Tao, Lingling and René Vidal (2015). "Moving poselets: A discriminative and interpretable skeletal motion representation for action recognition". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pp. 61–69.

Tas, Yusuf and Piotr Koniusz (2018). "Cnn-based action recognition and supervised domain adaptation on 3d body skeletons via kernel feature maps". In: *arXiv preprint arXiv:1806.09078*.

TASS International (n.d.). http://www.tassinternational.com/prescan.

Thrun, Sebastian, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe van Niekerk, Eric Jensen, Philippe Alessandrini, Gary Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara Nefian, and Pamela Mahoney

(Sept. 2006). “Stanley: The Robot That Won the DARPA Grand Challenge: Research Articles”. In: *J. Robot. Syst.* 23.9, pp. 661–692. ISSN: 0741-2223.

Truong, Arthur, Hugo Boujut, and Titus Zaharia (2016). “Laban descriptors for gesture recognition and emotional analysis”. In: *The visual computer* 32.1, pp. 83–98.

Van den Oord, Aaron, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu (2016). “Conditional image generation with pixelcnn decoders”. In: *Advances in neural information processing systems*, pp. 4790–4798.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). “Attention is all you need”. In: *Advances in Neural Information Processing Systems*, pp. 6000–6010.

Vautard, Robert, Pascal Yiou, and Michael Ghil (1992). “Singular-spectrum analysis: A toolkit for short, noisy chaotic signals”. In: *Physica D: Nonlinear Phenomena* 58.1-4, pp. 95–126.

Veeriah, Vivek, Naifan Zhuang, and Guo-Jun Qi (2015). “Differential recurrent neural networks for action recognition”. In: *Proceedings of the IEEE international conference on computer vision*, pp. 4041–4049.

Vemulapalli, Raviteja, Felipe Arrate, and Rama Chellappa (2014). “Human action recognition by representing 3d skeletons as points in a lie group”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 588–595.

Waibel, Alex, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J Lang (1989). “Phoneme recognition using time-delay neural networks”. In: *IEEE transactions on acoustics, speech, and signal processing* 37.3, pp. 328–339.

Wan, Li, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus (2013). “Regularization of neural networks using dropconnect”. In: *International conference on machine learning*, pp. 1058–1066.

Wang, C and SC Chan (2014). “A new hand gesture recognition algorithm based on joint color-depth Superpixel Earth Mover’s Distance”. In: *2014 4th International Workshop on Cognitive Information Processing (CIP)*. IEEE, pp. 1–6.

Wang, Chunyu, Yizhou Wang, Zhouchen Lin, Alan L Yuille, and Wen Gao (2014). “Robust estimation of 3d human poses from a single image”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2361–2368.

Wang, Chunyu, Yizhou Wang, and Alan L Yuille (2016). “Mining 3d key-pose-motifs for action recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2639–2647.

Wang, Hongsong and Liang Wang (2017). “Modeling temporal dynamics and spatial configurations of actions using two-stream recurrent neural networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 499–508.

Wang, Hongsong and Liang Wang (2018). “Beyond joints: Learning representations from primitive geometries for skeleton-based action recognition and detection”. In: *IEEE Transactions on Image Processing* 27.9, pp. 4382–4394.

Wang, Jian, Feng Zhou, Shilei Wen, Xiao Liu, and Yuanqing Lin (2017). “Deep metric learning with angular loss”. In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2593–2601.

Wang, Jiang, Zicheng Liu, Ying Wu, and Junsong Yuan (2013). "Learning actionlet ensemble for 3D human action recognition". In: *IEEE transactions on pattern analysis and machine intelligence* 36.5, pp. 914–927.

Wang, Lei, Du Q Huynh, and Piotr Koniusz (2019). "A Comparative Review of Recent Kinect-based Action Recognition Algorithms". In: *arXiv preprint arXiv:1906.09955*.

Wang, Pei, Chunfeng Yuan, Weiming Hu, Bing Li, and Yanning Zhang (2016). "Graph based skeleton motion representation and similarity measurement for action recognition". In: *European conference on computer vision*. Springer, pp. 370–385.

Wang, Pichao, Zhaoyang Li, Yonghong Hou, and Wanqing Li (2016). "Action recognition based on joint trajectory maps using convolutional neural networks". In: *Proceedings of the 24th ACM international conference on Multimedia*. ACM, pp. 102–106.

Weinberger, Kilian Q and Lawrence K Saul (2009). "Distance metric learning for large margin nearest neighbor classification". In: *Journal of Machine Learning Research* 10.Feb, pp. 207–244.

Weng, Junwu, Mengyuan Liu, Xudong Jiang, and Junsong Yuan (2018). "Deformable pose traversal convolution for 3d action and gesture recognition". In: *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 136–152.

Weng, Shuchen, Wenbo Li, Yi Zhang, and Siwei Lyu (2018). "Sts classification with dual-stream cnn". In: *arXiv preprint arXiv:1805.07740*.

Werbos, Paul J. (1990). "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10, pp. 1550–1560.

Weston, Jason, Sumit Chopra, and Antoine Bordes (2014). "Memory networks". In: *arXiv preprint arXiv:1410.3916*.

Williams, Ronald J and David Zipser (1989). "A learning algorithm for continually running fully recurrent neural networks". In: *Neural computation* 1.2, pp. 270–280.

Wu, Zonghan, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu (2019). "A comprehensive survey on graph neural networks". In: *arXiv preprint arXiv:1901.00596*.

Xia, Lu, Chia-Chih Chen, and Jake K Aggarwal (2012). "View invariant human action recognition using histograms of 3d joints". In: *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. IEEE, pp. 20–27.

Xie, Chunyu, Ce Li, Baochang Zhang, Chen Chen, Jungong Han, Changqing Zou, and Jianzhuang Liu (2018). "Memory attention networks for skeleton-based action recognition". In: *arXiv preprint arXiv:1804.08254*.

Yamato, Junji, Jun Ohya, and Kenichiro Ishii (1992). "Recognizing human action in time-sequential images using hidden markov model". In: *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on*. IEEE, pp. 379–385.

Yan, Sijie, Zhizhong Li, Yuanjun Xiong, Huahan Yan, and Dahua Lin (2019). "Convolutional sequence generation for skeleton-based action synthesis". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4394–4402.

Yan, Sijie, Yuanjun Xiong, and Dahua Lin (2018). "Spatial temporal graph convolutional networks for skeleton-based action recognition". In: *Thirty-Second AAAI Conference on Artificial Intelligence*.

Yan, Yichao, Jingwei Xu, Bingbing Ni, Wendong Zhang, and Xiaokang Yang (2017). "Skeleton-aided articulated motion generation". In: *Proceedings of the 25th ACM international conference on Multimedia*, pp. 199–207.

Yang, Fan, Sakriani Sakti, Yang Wu, and Satoshi Nakamura (2019). "Make Skeleton-based Action Recognition Model Smaller, Faster and Better". In: *arXiv preprint arXiv:1907.09658*.

Yang, Wei, Wanli Ouyang, Xiaolong Wang, Jimmy Ren, Hongsheng Li, and Xiaogang Wang (2018). "3d human pose estimation in the wild by adversarial learning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5255–5264.

Yang, Zhengyuan, Yuncheng Li, Jianchao Yang, and Jiebo Luo (2018). "Action recognition with spatio–temporal visual attention on skeleton image sequences". In: *IEEE Transactions on Circuits and Systems for Video Technology* 29.8, pp. 2405–2415.

Yörük, Erdem, Helin Dutağaci, and Bülent Sankur (2006). "Hand biometrics". In: *Image and vision computing* 24.5, pp. 483–497.

Se-Young Oh, Jeong-Hoon Lee, and Doo-Hyun Choi (May 2000). "A new reinforcement learning vehicle control architecture for vision-based road following". In: *IEEE Transactions on Vehicular Technology* 49.3, pp. 997–1005. ISSN: 00189545. DOI: 10.1109/25.845116. URL: http://ieeexplore.ieee.org/document/845116/.

Zanfir, Mihai, Marius Leordeanu, and Cristian Sminchisescu (2013). "The moving pose: An efficient 3d kinematics descriptor for low-latency action recognition and detection". In: *Proceedings of the IEEE international conference on computer vision*, pp. 2752–2759.

Zhang, Hao and Lynne E Parker (2015). "Bio-inspired predictive orientation decomposition of skeleton trajectories for real-time human activity prediction". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3053–3060.

Zhang, Hong-Bo, Yi-Xiang Zhang, Bineng Zhong, Qing Lei, Lijie Yang, Ji-Xiang Du, and Duan-Sheng Chen (2019). "A comprehensive survey of vision-based human action recognition methods". In: *Sensors* 19.5, p. 1005.

Zhang, Pengfei, Cuiling Lan, Junliang Xing, Wenjun Zeng, Jianru Xue, and Nanning Zheng (2017). "View adaptive recurrent neural networks for high performance human action recognition from skeleton data". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2117–2126.

Zhou, Yi-Tong and Rama Chellappa (1988). "Computation of optical flow using a neural network". In: *IEEE International Conference on Neural Networks*. Vol. 1998, pp. 71–78.

Zhou, Yi, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li (2019). "On the continuity of rotation representations in neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5745–5753.

Zhu, Guangming, Liang Zhang, Peiyi Shen, and Juan Song (2016). "An online continuous human action recognition algorithm based on the Kinect sensor". In: *Sensors* 16.2, p. 161.

Zhu, Wentao, Cuiling Lan, Junliang Xing, Wenjun Zeng, Yanghao Li, Li Shen, and Xiaohui Xie (2016). "Co-occurrence feature learning for skeleton based action recognition using regularized deep LSTM networks". In: *Thirtieth AAAI Conference on Artificial Intelligence*.

Zilly, Julian Georg, Rupesh Kumar Srivastava, Jan Koutnık, and Jürgen Schmidhuber (2017). “Recurrent highway networks”. In: *International Conference on Machine Learning*, pp. 4189–4198.

RÉSUMÉ

L'apprentissage profond est une branche du domaine de l'intelligence artificielle qui vise à doter les machines de la capacité d'apprendre par elles-mêmes à réaliser des tâches précises. L'apprentissage profond a abouti à des développements spectaculaires dans le domaine de l'image et du langage naturel au cours des dernières années. Pourtant, dans de nombreux domaines, les données d'observations ne sont ni des images ni du texte mais des séries temporelles qui représentent l'évolution de grandeurs mesurées ou calculées. Dans cette thèse, nous étudions et proposons différentes représentations de séries temporelles à partir de modèles d'apprentissage profond. Dans un premier temps, dans le domaine du contrôle de véhicules autonomes, nous montrons que l'analyse d'une fenêtre temporelle par un réseau de neurones permet d'obtenir de meilleurs résultats que les méthodes classiques qui n'utilisent pas de réseaux de neurones. Dans un second temps, en reconnaissance de gestes et d'actions, nous proposons des réseaux de neurones convolutifs 1D où la dimension temporelle seule est convoluée, afin de tirer profit des invariances temporelles. Dans un troisième temps, dans un but de génération de mouvements humains, nous proposons des réseaux de neurones génératifs convolutifs 2D où les dimensions temporelles et spatiales sont convoluées de manière jointe. Enfin, dans un dernier temps, nous proposons un plongement où des représentations spatiales de poses humaines sont (ré)organisées dans un espace latent en fonction de leurs relations temporelles.

MOTS CLÉS

Apprentissage Profond, Contrôle Couplé de Véhicule, Reconnaissance de Gestes et d'Actions, Génération de Mouvements Humains, Représentation de Pose Humaine, Séries Temporelles, Modélisation de Données Séquentielles

ABSTRACT

Artificial intelligence is the scientific field which studies how to create machines that are capable of intelligent behaviour. Deep learning is a family of artificial intelligence methods based on neural networks. In recent years, deep learning has lead to groundbreaking developments in the image and natural language processing fields. However, in many domains, input data consists in neither images nor text documents, but in time series that describe the temporal evolution of observed or computed quantities. In this thesis, we study and introduce different representations for time series, based on deep learning models. Firstly, in the autonomous driving domain, we show that, the analysis of a temporal window by a neural network can lead to better vehicle control results than classical approaches that do not use neural networks, especially in highly-coupled situations. Secondly, in the gesture and action recognition domain, we introduce 1D parallel convolutional neural network models. In these models, convolutions are performed over the temporal dimension, in order for the neural network to detect -and benefit from- temporal invariances. Thirdly, in the human pose motion generation domain, we introduce 2D convolutional generative adversarial neural networks where the spatial and temporal dimensions are convolved in a joint manner. Finally, we introduce an embedding where spatial representations of human poses are sorted in a latent space based on their temporal relationships.

KEYWORDS

Deep Learning, Coupled Vehicle Control, Gesture and Action Recognition, Human Motion Generation, Human Pose Embedding, Time Series, Sequence Modeling