POLITECNICO DI MILANO

Dipartimento di Elettronica, Informazione e Bioingegneria

Doctoral Programme in Information Technology

On Iterative and Conditional Computation for Visual Representation Learning

Doctoral Dissertation of:

MARCO CICCONE

Advisor: PROF. MATTEO MATTEUCCI

Co-advisor: DR. JONATHAN MASCI

Tutor: PROF. CRISTINA SILVANO

2021 - XXXII Cycle

To my family,
Marzia, Vincenzo, Filippo and Carlotta.

ABSTRACT

Learning effective representations is crucial for scaling the performance of machine learning methods. Deep Neural Networks are flexible models that can learn powerful hierarchical representations by stacking several layers of computations. However, once learned, adapting the representation to new data or behaviours is nontrivial. In this thesis, we take a step in the direction of learning adaptive representations for visual data addressing the problem both from a practical and theoretical perspective.

First, we study *Residual Networks* from a dynamical system perspective and augment them with a mechanism to automatically adapt the number of processing steps based on the characteristics of the data.

Then, we focus on the problem of learning effective asynchronous representations for event-based data. We propose a recurrent mechanism that automatically learns how to incrementally build a two-dimensional representation from events, which can be used as input to convolutional frame-based architectures to improve their performance on optical flow prediction and image recognition tasks with respect to hand-designed features.

Finally, we focus on the challenging problem of One-Shot Video Object Segmentation, where the model is asked to segment specific objects in unseen videos after observing a single annotated frame. We tackle the problem from a Meta-Learning perspective by showing that it is possible to adapt a generic meta-representation to specific task-representations, by modulating the activations of a segmentation network conditioned on the given instance.

PUBLICATIONS

MAIN THESIS PUBLICATIONS

Cannici, Marco, **Marco Ciccone**, Andrea Romanoni, and Matteo Matteucci (2020). "A Differentiable Recurrent Surface for Asynchronous Event-Based Data." In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, pp. 1–17.

Lattari*, Francesco, **Marco Ciccone***, Matteo Matteucci, Jonathan Masci, and Francesco Visin (2018). "ReConvNet: Video Object Segmentation with Spatio-Temporal Features Modulation." In: *The 2018 DAVIS Challenge on Video Object Segmentation - CVPR Workshops*.

Ciccone, **Marco**, Marco Gallieri, Jonathan Masci, Christian Osendorfer, and Faustino Gomez (2018). "NAIS-Net: Stable Deep Networks from Non-Autonomous Differential Equations." In: *Advances in Neural Information Processing Systems (NeurIPS)*.

OTHER PUBLICATIONS

Cacciamani, Federico, Andrea Celli, **Marco Ciccone**, and Nicola Gatti (2021). "Multi-Agent Coordination in Adversarial Environments through Signal Mediated Strategies." In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Cannici, Marco, **Marco Ciccone**, Andrea Romanoni, and Matteo Matteucci (2019a). "Asynchronous Convolutional Networks for Object Detection in Neuromorphic Cameras." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW) - **Best Paper Award***.

Cannici, Marco, **Marco Ciccone**, Andrea Romanoni, and Matteo Matteucci (2019b). "Attention Mechanisms for Object Recognition with Event-Based Cameras." In: *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV)*.

Romanoni, Andrea, **Marco Ciccone**, Francesco Visin, and Matteo Matteucci (2017). "Multi-View Stereo with Single-View Semantic Mesh Refinement." In: *Proceedings of the IEEE International Conference on Computer Vision Workshops, Reconstruction Meets Semantic, (ICCVW)*, pp. 706–715.

ACKNOWLEDGMENTS

If I had to choose a single word to summarize my PhD that would be *"acceptance"*. When I started four years ago, I was at the beginning of a very painful process to find myself. My strengths and weaknesses were entangled into a mess of emotions that I couldn't process. Unfortunately, pursuing a career path in academia means finding ourselves in front of our monsters more often than expected and hiding is not an option. Facing my monster and coming to this point would not have been possible without all the amazing people that I met during these years. Hopefully, I will not forget to mention anyone.

First, I would like to thank my advisor, Matteo Matteucci, for always second my curiosity and leaving me the freedom to pursue my interests, even when they drifted from his own. I would also like to thank Sepp Hochreiter and Barbara Caputo for examining my thesis and spending very nice words of appreciation on my work. I will always be in debt with Jonathan Masci. I remember when we first met at the airport both flying to a conference in Barcelona and I awkwardly introduced myself because I was overhearing a research conversation. After a couple of days, I was sleeping on your couch after a party at your apartment. You offered me an internship after an interview at the spiciest Chinese restaurant I have ever tried. You taught me how to be a better researcher and develop critical thinking by always questioning everything. You encouraged me to explore and looking for new experiences. You showed me what leadership means and that it always goes together with respect, honesty, and politeness. I am extremely grateful for having met you along my path and I will never forget what you have done for me. Thank you to all of the people I met at NNAISENSE who created a great stimulating environment for research. Thank you to Marco Gallieri and Christian Osendorfer: without your math and ideas my first publication would not have been possible. Thank you Tino for all your draft notes and for teaching me the importance of writing and storytelling in research. I also would like to thank Jürgen Schmdhuber for the eye-opening conversations on meta-learning that had an important impact on my current research. Thanks to Iuri Frosio for mentoring me during my experience at NVIDIA and for all the hiking advice that made my experience in California even more amazing. Thanks to Rupesh and Wonmin for the many dinners that helped me adjust to the American cuisine. Thank you Luigi Malagò for letting me visit your lab in Cluj, for sharing your research experience with me, and for teaching me the importance of incremental steps in research. You are the first that clearly spelled for me that a PhD is not the end of research but just the beginning. Your advice helped me when I was doubting

myself as a researcher and I was seeing everything black. I also would like to thank Nicola Gatti and Andrea Celli for giving me the opportunity to collaborate on a new and exciting topic and challenge myself on something different from Computer Vision. Last but not least I would like to thank Francesco Visin for being my *Spirit of Christmas Future* since the first moment we have met. You have been my first mentor, and you still are. I am truly glad that we finally understood each other. Thank you for all your advice and for helping me get through the last difficult mile. You really helped me make peace with myself as a researcher. I hope there will many other beers and papers together. I have to thank all the people from AIRLab who made these last years easier to bear. Thank you Andrea for sharing your wisdom and helping me understand that my pace is the only one that matters. Thank you Marco for all the projects that we carried out together and that wouldn't be possible without your skills. Thank you, Francesco (a.k.a. Cicciolo, a.k.a. Latta) for being first a student, then a colleague, and finally, a friend. Thank you for all the laughter, and for showing me how not to deal with arrosticini and alcohol on a hot day in July. Also, thank you Pietro for your friendship, stalking, and support during these years in the form of kicks in the legs, tagliatelle al ragù and erbazzone.

Alla mia famiglia: ancora una volta questa tesi è dedicata a voi e a tutto il tempo che vi ho sottratto. Ai miei genitori Marzia e Vincenzo, per tutti i sacrifici che avete fatto per permettermi sempre di avere scelta nella vita. Grazie a mia nonna Rosa: è anche grazie a te se sono il *"Nipote d'America"*. Grazie a mio fratello Filippo e mia sorella Carlotta per i vostri incoraggiamenti. Sono fiero di voi e delle persone che siete diventati. Continuate per la vostra strada, so che arriverete dove vorrete. Non abbiate mai paura di fare le vostre scelte e non permettete a nessuno di farle per voi. I vostri errori sono vostra cosa più previosa. Vi porteranno a scoprire chi siete davvero. Ricordatevi che io ci sarò sempre. Vi voglio bene.

Infine, grazie a Lauretta per tutte le avventure che abbiamo passato insieme in questi anni. Questa tesi non sarebbe stata possibile senza il tuo premuroso supporto e incoraggiamento. Grazie per essere sempre fonte di ispirazione e per aver scelto di affrontare questo viaggio insieme a me con tanta tenacia e convinzione. Grazie perchè ogni giorno mi aiuti a scoprire una nuova parte di me che non conoscevo. Tuo, *Groviglio*.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ERM	Empirical Risk Minimization
NN	Neural Network
FNN	Feedforward Neural Network
MLP	Multilayer Perceptron
CNN	Convolutional Neural Network
TDNN	Time-Delay Neural Network
RNN	Recurrent Neural Network
VDNN	Very Deep Neural Network
LSTM	Long Short-Term Memory
BPTT	Backpropagation Through Time
SGD	Stochastic Gradient Descent
ResNet	Residual Network

1 INTRODUCTION

For years cognitive scientists studied the principles of intelligence and human behaviours, trying to unravel the secrets of our mind. The first attempts to understand how the brain processes information date back to the Ancient Greeks, when Plato and Aristotle tried to explain the nature of human knowledge. As humans, we understand the world by efficiently encoding information in powerful abstractions and rich causal models that allow us to generalize even from sparse, noisy, and ambiguous observations. Since our first months of life, we develop skills that can be combined together to solve more complex tasks. Even more fascinating is our capability to adapt and make inference in new situations, by retrieving absorbed concepts and inventing new solutions to problems never experienced before.

Artificial Intelligence (AI) has been a central part of cognitive science since the 1950s and always had the ultimate goal of providing machines with the same kind of reasoning capabilities as humans, understanding how they can be so efficient in processing information. It soon became apparent that distilling human knowledge directly into machines by writing ad hoc programs is not scalable. As the complexity of the problems increases, designing programs based on predetermined rules becomes unfeasible. Computers should learn rules by themselves, exploring the world as humans do. *Machine Learning* allows us to design machines that automatically learn algorithms from experience. By processing data autonomously, algorithms can improve themselves, learning how to extract patterns or input-output mappings from examples.

1.1 REPRESENTATION LEARNING

The performance of machine learning methods is strictly dependent on the choice of data representation. Feature engineering aims to design features that support effective learning by taking advantage of prior knowledge on the task. However, designing powerful informative features is challenging and time-consuming, often requiring the opinion of domain experts. Scaling machine learning to more complex scenarios requires making algorithms less dependent on feature engineering and automatizing this process. By learning optimal representations directly from data, rather than by hand designing features based on heuristics, *Representation Learning* provides

artificial agents with the ability to understand the world by identifying and disentangling the underlying explanatory factors hidden in raw data.

In the last decade, deep learning methods have become the primary tool for learning representations, rapidly becoming one of the most prolific research areas. By often leveraging massive annotated datasets, GPU-based parallelization, and new frameworks for automatic differentiation, *deep neural networks* are behind the most recent revolution in Artificial Intelligence, becoming the horsepower of a plethora of applications that span from Computer Vision and Natural Language Processing to Reinforcement Learning. Thanks to their compositional nature, deep neural networks are particularly suitable for representation learning. Indeed, the reasons for their success are to be found in their ability to learn *distributed hierarchical abstractions* directly from raw data through the composition of multiple nonlinear projections. Learning several layers of abstraction by jointly training them repeatedly warps the input space, making the task easier to solve for the predictor.

1.2 MOTIVATION: ADAPTIVE AND CONDITIONAL COMPUTATION IN NEURAL NETWORKS

With the increasing pervasiveness of intelligent systems in our daily life, it becomes of vital importance to design adaptive models that can modify their behaviour at inference time when required.

Think of a robot that is trained to perform a specific task, for instance, to grasp an apple from a table. One could try and train such a robot on several hours of experience until it successfully learns how to grasp it. But what happens if we ask the robot to perform the same task on a different object, for instance, something with a completely different shape and physical properties, such as a glass of water. Unfortunately, we can not expect the robot to succeed since it never experienced something like that before. In general, one could mitigate the problem by collecting more experience and training in multiple scenarios. Still, while the robot might learn to perform the tasks seen during training, it would unlikely generalize to some new behaviours by itself. This is not limited to robotics tasks, but rather the same rationale can be applied to any other domain. For instance, one would want to train a recognition system that has to recognize new persons from their face. Instead of training a model from scratch every time a new person appears in an image or video, it would be much more efficient to create a system that can be calibrated on different faces just by observing few examples. In order to achieve that, we have to explicitly design a system that can adapt to new situations with few interactions with the world without the need to learn everything from scratch.

The flexibility of machine learning models can also be analyzed from a computational perspective. Neural Networks build powerful representations by performing successive steps of computation, i.e., through layers. The optimal number of steps to solve a task can vary from input to input and is usually fixed for practical reasons. However, it might be the case that some problems are more challenging to solve than others, and thus require more processing time.

It is in fact often the case that the representation extracted from the input after only a few layers is already rich enough to solve the task at hand, and that a lot of computational power is wasted by processing this intermediate representation through several more layers.

In turn, the human brain is highly efficient, and it adapts its computational budget to the complexity of the problem. The input can also be asynchronous, requiring the model to integrate information over time to incrementally build a representation and eventually refine its prediction. Machine learning algorithms with the ability to adapt the amount of computation as required by the circumstances are crucial, especially in those scenarios where power efficiency is at the essence, such as the deployment of robots and drones.

1.3 MAIN CONTRIBUTIONS AND OUTLINE

This thesis describes advancements in learning adaptive representations for visual data addressing the problem both from a practical and theoretical perspective. We develop a new neural network model for adaptive computation that executes a varying number of computational steps based on the processed sample's characteristics (Chapter 3). We propose a recurrent model for asynchronous event-based data that incrementally builds a two-dimensional representation from a stream of sequential events (Chapter 4). Finally, we also introduce a recurrent and convolutional model for video object segmentation that can modulate its internal representation at inference time to new objects never observed before (Chapter 5). The rest of the thesis is organized as follows:

- Chapter 2 provides the reader with the necessary Machine Learning background to understand the contributions of the thesis.
- Chapter 3 focuses on the study of Residual Networks (He et al., 2016) (ResNets) one of the most successful neural architectures for computer vision. ResNets, along with Highway Networks (Srivastava et al., 2015a) gave an important contribution to the field of deep learning, enabling the training of very deep neural networks via gradient-based learning with just a simple architectural modification: *skip connections*. This chapter carries out a theoretical analysis of ResNets from a dynamical system perspective, explaining the reason why these models

perform iterative computation through successive additive non-linear projections (Greff et al., 2017; Jastrzebski et al., 2018). Starting from the observation that ResNets are unrolled time-invariant dynamical systems, we identify the issue of *system stability* and propose NAIS-Nets, a novel architecture that can be unrolled with convergence guarantees. We derive proofs and perform an extensive experimental analysis that confirms our theoretical results. We also show that the proposed stable architecture encourages conditional iterative computation, by assigning similar *processing depths* to samples with similar characteristics.

The chapter is based on Ciccone et al. (2018), published at the NeurIPS 2018 conference. The work has been carried out during an internship at NNAISENSE SA, Lugano.

- Chapter 4 focuses on the problem of learning representations for asynchronous event-based data. Event-cameras are modern sensors inspired by the functioning of the human vision system. Rather than producing full frames at regular time intervals as in conventional RGB cameras, each pixel senses brightness changes independently and outputs trains of raw *events* based on the scene dynamics. Although they provide several advantages over traditional image acquisition devices (e.g, low latency, high dynamic range, and high temporal resolution), their true potential is still to be unlocked since novel methods are required to deal with the asynchronous and sparse nature of their outputs. In this chapter, we take a first step in this direction by proposing a novel representation for event-based data that replaces hand-engineered features with a learned integration mechanism. We propose a drop-in replacement input layer, Matrix-LSTM, based on a grid of LSTM (Hochreiter and Schmidhuber, 1997b) cells that *incrementally* process events as soon as they arrive, by updating a two-dimensional time-surface that is optimal for the task at hand. We show that Matrix-LSTM produces a powerful grid representation that can be used as input to any modern computer vision pipeline, retaining most of the advantages of asynchronous data.

  The chapter in based on Cannici et al. (2020), published at the ECCV 2020 conference.

- Chapter 5 addresses the problem of learning new tasks from few examples. We focus on the challenging problem of One-Shot Video Object Segmentation, where the model is asked to segment specific objects from new videos after observing a single annotated frame. We tackle the problem from a Meta-Learning (Baxter, 1995; Schmidhuber, 1987) perspective by showing that it is possible to adapt a general meta-representation to specific task representations conditioned on the given context. We propose a meta-learning system based on two learning levels: a backbone segmentation network learns to perform

generic segmentation strategies, while secondary networks modulate its representation by generating task-specific weights.

The chapter in based on Lattari et al. (2018), published at the Workshop on Video Object Segmentation - DAVIS Challenge at CVPR 2018 Conference.

- Chapter 6 gives a unifying view of the thesis and outlines some promising future directions.

2 BACKGROUND

This chapter gives a brief and self-contained introduction of the main machine learning and deep learning tools used throughout this thesis to provide the reader with the notation and the necessary background to understand the presented concepts. Additional background topics will be introduced later where necessary. The resources in this chapter are adapted from the comprehensive Deep Learning textbook (Goodfellow et al., 2016), while we refer the readers to Bishop (2006) and Murphy (2012) for more detailed discussion on the fundamentals of machine learning, and to Shalev-Shwartz and Ben-David (2014) for statistical machine learning theory.

2.1 LEARNING FROM EXPERIENCE

Machine Learning is the study of computer algorithms that improve automatically through experience:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

This informal definition from Mitchell (1997) can result in a variety of possibilities for the experience E, the task T and the performance metric P, depending on the operative framework that we might want to use to build machine learning algorithms. Alternatively we can provide intuitive descriptions on what kind of objects can be used for each of these entities.

TASK Machine Learning helps us by tackling problems that are too difficult to be solved by designing programs with a set of predefined rules. In other words, a task is a problem that we want to solve, and the process of learning the task is our means of attaining the ability to solve it. More naively, we expect that a machine learning algorithm would develop an understanding of the task to be solved by interacting with collected data and updating its own knowledge of it.

Usually, a machine learning algorithm is described in terms of its ability to process an example as input. Examples are defined, for instance, as vectors of *features* $x \in \mathbb{R}^n$, that represent the input sample. Depending on the output mapping considered, we can list different classes of tasks:

- *Classification*: the algorithm is asked to assign a class k among a set of possible categories to a given input. This is usually achieved by learning mapping in the form of $f : \mathbb{R}^n \to \{1, \ldots, K\}$.
- *Regression*: the algorithm is asked to predict a real value, given some input. The task can be solved by learning the input-output mapping as a function $f : \mathbb{R}^n \to \mathbb{R}$.
- *Transcription/Translation*: in this setting, the output to be predicted is a sequence of symbols. In the first case, the algorithm can observe an unstructured representation that has to be transcribed, while in the latter case, the input is already a sequence to be translated into another language or space of symbols.
- *Structured output*: these types of tasks involve any function mapping where the output is a vector of predictions or any other data structure containing multiple values, sharing relationships across the elements. This might be considered a superclass of the previous tasks, but other problems fall into this category.

  In the context of computer vision, one may consider a structured classification task as pixel-segmentation. The task requires to assign a specific category to each pixel considering the surrounding context to produce a coherent segmentation. In a structured regression task as optical flow estimation, the computer program analyses two subsequent frames and outputs the vector displacement of each pixel.

Depending on the area of application, we can define many other tasks and classes of tasks. In this thesis, we are going to focus only on the types of tasks presented above.

PERFORMANCE MEASURE It is necessary to define a metric to evaluate a machine learning algorithm's ability to perform on a specific task T. The performance measure is generally defined as a numerical value computed on a subset of data by comparing the prediction output of the algorithm and its ground truth value. We are generally interested in measuring the quality of the algorithm on data never observed during the learning process to analyze its generalization capability.

The choice of the correct performance metric is far from trivial and it is has the same importance of the algorithm itself. For example, when performing pixel-segmentation, we are not usually interested in the accuracy defined as the number of correctly classified pixels over the total amount of pixels. That would be biased towards classes of objects that are much more represented than others, such as the background. Instead, we are interested in a measure such as the mean Intersection-over-Union that tells us how much the prediction overlaps with the target, penalizing both false positive and false negative.

It is important to note that it is not always possible to optimize the performance measure we are interested in directly. It is often necessary to find related proxies that are easier to compute or optimize to allow the algorithm to effectively learn the task at hand.

EXPERIENCE The experience E is defined as the data that a machine learning algorithm can process to learn how to solve the task. Machine Learning algorithms are generally categorized under three main learning paradigms based on what kind of experience they have access to during the learning process. We define a *dataset* as the entire collection of experience, and the examples that form the dataset as *data points*.

- *Supervised Learning* algorithms observe a dataset where each sample x is paired with a *label* or *target* y provided from a knowledgeable external supervisor. The learning task is often formalized as estimating $p(y|x)$ and the learning algorithms learns the mapping function from x to y.
- *Unsupervised Learning* algorithms experience a dataset that provides only examples x, then learn to extract hidden patterns in the unlabeled data by clustering together similar examples. Usually, we are interested in learning the generative model of the dataset $p(x)$, whether explicitly as in density estimation tasks or implicitly via denoising or synthesis.
- *Reinforcement Learning* algorithms aim to learn the optimal policy of an agent by interacting sequentially with an environment and observing a reward signal. The agent's goal is to maximize the total reward it receives over the long run by performing actions in each state he visits.

In this thesis, we focus solely on supervised and unsupervised learning settings in the context of computer vision tasks such as image classification Chapter 3 , optical flow prediction Chapter 4 and object segmentation Chapter 5.

2.2 SUPERVISED LEARNING

So far, we only gave an intuitive explanation of a machine learning algorithm and its components without providing any operative description. In this section, we focus on supervised learning, and we formalize it in a principled way using the Empirical Risk Minimization framework (Vapnik, 1992; 1998).

Consider an *input space* $\mathcal{X}$ and an *output* (or *target*) *space* $\mathcal{Y}$, where typically $\mathcal{X} \subseteq \mathbb{R}^d$. The form of the output space yields different kinds of tasks: regression $\mathcal{Y} \subseteq \mathbb{R}$, binary classification $\mathcal{Y} = \{-1; +1\}$, multi-class classification $\mathcal{Y} = \{1, \dots, K\}$. Let (x, y) be a pair of random variables distributed

according to the unknown joint probability distribution $p(x, y)$, defined over the *data space* $\mathcal{X} \times \mathcal{Y}$.

Let $\mathcal{D}_n = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ be a *training dataset* made of n identically and independently distributed (i.i.d.) input-target pairs, that we assume to be representative of the data distribution $p(x, y)$. Supervised Learning can be formalized as the process of finding the input-output mapping $f : \mathcal{X} \to \mathcal{Y}$ based on the training data $\mathcal{D}_n$ to predict via $f(x)$ the output corresponding to any new input $x \in \mathcal{X}$. This is usually achieved by searching for a good approximation of the true mapping over a class of candidate functions $\mathcal{H}$, called *hypothesis space*. More precisely, we can formulate an optimization problem:

$$\mathcal{R}^* = \min_{f \in \mathcal{H}} \quad \mathcal{R}(f), \tag{2.1}$$

where $\mathcal{H}$ can be any class of functions such as *linear functions*, *radial basis functions* or *deep neural networks*, and $\mathcal{R}$ is a *"risk"* functional that depends on the training data. For any fixed *loss function* $\ell : \mathcal{Y} : \mathcal{Y} \to \mathbb{R}$, the risk is defined as the expected loss over the data distribution:

$$\mathcal{R}(f) = \mathbb{E}_{p(x,y)} \left[\ell(y, f(x)) \right] = \int \int \ell(y, f(x)) p(x, y) dx dy \tag{2.2}$$

The loss function is a point-wise measure of the error $\ell(y, f(x))$ made by predicting $f(x)$ when the actual output is y, while the risk is a measure of the average performance of the machine learning algorithm on the task.

2.3 EMPIRICAL RISK MINIMIZATION

Given a fixed loss function $\ell(\cdot, \cdot)$, the *true* risk minimization is generally non-trivial because the underlying data distribution in unknown, making the expectation in Equation (2.2) non-computable and the optimization problem in Equation (2.1) intractable.

Instead of minimizing $\mathcal{R}(f)$ directly, one may replace the true data distribution $p(x, y)$ by its empirical distribution computed using the training dataset $\mathcal{D}_n$, and obtain the following minimization problem:

$$\hat{\mathcal{R}}_n = \min_{f \in \mathcal{H}} \quad \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(x_i)), \tag{2.3}$$

which is called *Empirical Risk Minimization (ERM)* (Vapnik, 1992). Because the data distribution is unknown, instead of minimizing the risk directly, ERM uses a sample-based estimate to optimize the true risk. In particular, it can be shown that, under some conditions, ERM is *statistically consistent*, meaning that, more data is collected, the empirical risk converges in probability to its true value, and ERM corresponds to minimizing the true risk (Shalev-Shwartz and Ben-David, 2014; Vapnik, 1998).

MAXIMUM LIKELIHOOD ESTIMATION The induction principle of empirical risk minimization is quite general and encompasses many popular learning methods. For instance, restricting the space of hypothesis to $\mathcal{H} = \{f(x) = \theta^T x \quad \forall \theta \in \mathbb{R}^d\}$ and taking a square loss function $\ell(y, f(x)) = (y - f(x))^2$, corresponds to the well known *least squares* estimation. *Maximum likelihood estimation (MLE)* (Bishop, 2006; Murphy, 2012) is a also a special case of ERM where the loss function is the *negative log-likelihood*:

$$\theta_{MLE} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} p(y|x, \theta), \tag{2.4}$$

2.4 MODEL COMPLEXITY: UNDERFITTING VS OVERFITTING

Although under certain conditions the empirical risk corresponds to the true risk when the sample size n grows, in practice, the number of samples is always finite. As a result, the actual predictor might not minimize the actual risk, especially when the space of hypothesis $\mathcal{H}$ is large, and the number of samples is limited. When we have a small empirical risk but still a relatively large true risk, we say that the algorithm is *overfitting* the training data. A good learning algorithm should provide a similar behaviour to the target function and perform well on new, previously unseen data — never experienced in the training set $\mathcal{D}_n$. In this case, we say that the algorithm *generalizes* well. We typically estimate the *generalization error* of a predictor by measuring the performance on a *test set* of examples that were separately collected from the training set, under the assumption of being identically distributed.

Suppose that $\hat{f}_n$ is the ERM hypothesis of a predictor, namely a function in $\mathcal{H}$ that minimizes the empirical risk, and suppose to know that the best predictor among $\mathcal{H}$ is:

$$f^* = \arg\min_{f \in \mathcal{H}} \quad \mathcal{R}(f), \tag{2.5}$$

we can define the *excess risk* of $\hat{f}_n$ and decompose the error of an ERM predictor in two components as follows:

$$\mathcal{R}(\hat{f}_n) - \mathcal{R}^* = \underbrace{(\mathcal{R}(f^*) - \mathcal{R}^*)}_{\text{approximation error}} + \underbrace{\left(\mathcal{R}(\hat{f}_n) - \mathcal{R}(f^*)\right)}_{\text{estimation error}} \tag{2.6}$$

The *approximation error* measures the risk caused by the restriction to a specific class of hypothesis $\mathcal{H}$, also called *inductive bias*. The approximation error is deterministic and does not depend on the sample size. It can be reduced by extending the hypothesis class to other functions. The *estimation error* represents how much we are losing in terms of risk by using a finite

sample approximation instead of using the true data distribution, and it gives a measure of the quality of the training set.

Similarly to the *bias-variance trade-off* for standard statistical estimation problems (MacKay, 2003), there is also a tension between the approximation error and estimation error. The approximation error term acts like a bias square term while the estimation error acts like the variance term. Choosing a very rich class of candidate functions $\mathcal{H}$ decreases the approximation error at the cost of the estimation error, but it might lead to *overfitting*. On the other hand, restricting $\mathcal{H}$ reduces the estimation error but might increase the approximation error because the predictor might not be expressive enough to represent the true mapping leading to *underfitting*.

2.5 REGULARIZATION TECHNIQUES

Overfitting can be mitigated by following the Occam's razor principle (Domingos, 1999) of *"parsimony of explanation"*, which states that if two models explain data equally one should chose the simplest one, based on some notion of complexity. Model regularization is usually achieved by adding a penalization term $\lambda R(\theta)$ to the objective function that encourages the learning of simpler models over complex ones, where λ is a coefficient that regulates the level of regularization $R(\theta)$, that is function of the model. In the context of ERM, adding regularization terms that penalize model complexity is usually referred to as *Structural Risk Minimization*. From a Bayesian perspective, adding a regularization term can be interpreted as imposing a prior over the parameters of the model. For example, a penalty in the form of $\lambda R(\theta) = \theta^T\theta = \|\theta\|$, also called *weight decay* or simply $L2$ regularization corresponds to assuming that the parameters are normally distributed with zero mean (Bishop, 2006).

2.6 STOCHASTIC GRADIENT DESCENT

Gradient descent is an iterative optimization algorithm that improves the problem's solution by taking a step in the direction of the negative of the gradient of the function to be minimized at the current point.

When the class of candidate functions and the loss function are differentiable, the risk functional is also differentiable and the minimization can be efficiently solved, albeit approximately, by applying numerical optimization techniques such as *Stochastic Gradient Descent (SGD)* (Robbins and Monro, 1951). When the exact gradient is not available, stochastic gradient descent circumvents this problem by allowing the optimization procedure to take a step along a noisy direction, as long as the expected value of the direction

is the negative of the gradient. SGD provides a method to optimize directly the risk functional since the gradient of the loss function on a randomly sampled example is an unbiased estimate of the gradient of the risk.

$$\mathbb{E}_{p(x,y)}\left[\nabla \ell(y, f(x))\right] = \nabla \mathbb{E}_{p(x,y)}\left[\ell(y, f(x))\right] = \nabla \mathcal{R}(f) \tag{2.7}$$

Given a dataset of examples, training a model via gradient descent simply consists in repeatedly sampling data points and applying the following update rule at each i-th iteration to modify the parameters of the model θ in the direction that minimizes the loss:

$$\theta^{(i+1)} = \theta^{(i)} - \eta \nabla \ell(\theta^{(i)}) \tag{2.8}$$

where $\theta^{(0)}$ is randomly initialized, η is the learning step, $\ell : \mathbb{R}^n \to \mathbb{R}$ and the vector of partial derivatives is $\nabla \ell(\theta) = \left(\frac{\partial \ell(\theta)}{\partial \theta^1}, \ldots, \frac{\partial \ell(\theta)}{\partial \theta^n}\right)$.

Combined with automatic differentiation and recent advances in stochastic optimization (Duchi et al., 2011; Hinton et al., 2012; Kingma and Ba, 2015; Nesterov, 1983a; Zeiler, 2012) this provides a turnkey approach to fitting modern differentiable machine learning models such as deep neural networks (see Section 2.7.4.1 for more details).

2.7 NEURAL NETWORKS

Neural Networks (NNs) are popular machine learning models that are able to capture input-output relationships and complex patterns in the data. Thanks to their expressivity, NNs can be employed in many machine learning problems whether in supervised, unsupervised, or reinforcement learning (Bishop, 2006; Sutton and Barto, 1998) settings.

Early Neural Networks' ancestors (e. g. Perceptrons (Rosenblatt, 1958), Neucognitron (Fukushima, 1980)), were loosely inspired by models of information processing in human and mammals brains but soon drifted from their initial biological inspiration. Since their introduction, these models have received moderate interest from the scientific community and are considered the cornerstones of modern neural networks. Nevertheless, research on neural networks stagnated for many decades before the recent deep learning revolution. One of the reasons for the initial lost of interest in neural networks was the lack of an efficient algorithm for training complex models. Before the discovery of the *backprogation algorithm*, NNs were considered intractable models. For a detailed survey on the history of neural networks, from the first *connectionist models* to the recent developments, we refer to Schmidhuber (2015). In the rest of the chapter, we introduce the basic concepts of modern deep learning architectures preparatory for understanding the thesis. An exhaustive treatise of machine learning and neural networks can be found in Bishop (1995), Bishop (2006) and Goodfellow et al. (2016).

A Neural Network can be described as a system of basic processing units, *the neurons*, that are connected by weighted directed edges. The information flows through the computational graph, and it is processed by the neurons as a function of their input — the produced output is also called *the activation*. Connections between neurons, *the weights*, are gradually adjusted during the learning process and determine the contribution to the output of the signals generated by the source neuron. The set of units in the graph are typically organized into sequences of *layers*; the number of units in a layer is referred to as its *width* and the total number of processing layers indicates the *depth* of the neural network. The set of neurons that processes the source information is usually referred to as *input layer*, while the ones that produce the prediction are called *output layer*. All the other layers are generally called *hidden layers*.

NNs with *acyclic* computational graph are named *Feedforward Neural Networks (FNNs)* because the information flows only in one direction, while those with *cyclic* graphs are called *Recurrent Neural Networks (RNNs)*. Thanks to self-connections, RNNs can process sequential inputs and retain previously processed information to be used at future time steps.

Neural Networks' flexibility make them a perfect family of candidate functions for risk minimization. FNNs are in fact know to be *universal function approximators*, i. e. a feedforward neural network with a single hidden layer can approximate any measurable function to any desired degree of accuracy on a compact set (Cybenko, 1989; Hornik, 1991; Hornik et al., 1989). Similarly, it has been shown that RNNs are Turing-complete (Siegelmann and Sontag, 1995). Although FNNs with few hidden layers can potentially represent any function, distributing the computation across multiple layers can be exponentially more efficient for some class of functions (Delalleau and Bengio, 2011; Montufar et al., 2014). Based on this observation, modern neural architectures builds hierarchical data abstractions growing models in depth, improving model efficiency at the expense of trainability — see Section 2.7.4.4 for more details on the difficulties of training neural networks.

We now present an overview of the types of layers and neural networks that are used in this thesis. In order to keep the notation uncluttered, the computational graph of each layer is described in terms of matrix multiplications rather than single units interactions.

2.7.1 Types of Layers and Networks

MULTILAYER PERCEPTRON A *Multilayer Perceptron (MLP)*, is the simplest kind of neural network and it consists of a series of dense layers where each neuron is connected to all of its input, called *fully-connected* layers. An MLP can be represented by a function $: \mathbb{R}^d \to \mathbb{R}^m$ that maps from the input to the output space via a composition of L layers. Each fully-connected layer, $\ell \in \{1, \dots L\}$, involves first an affine transformation of the input,

parametrized by a matrix of weights $\mathbf{W}_\ell$ and a bias vector $\mathbf{b}_\ell$, followed by a non-linear *activation* function $f(\cdot)$, that warps the hidden space. The input $\mathbf{x}_\ell$ of an hidden layer is the output of its predecessor $\mathbf{y}_\ell$, and each layer is applied in chain until the final output $\hat{\mathbf{y}} = \mathbf{y}_L$ is produced:

$$\mathbf{x}_\ell = \mathbf{y}_{\ell-1} \tag{2.9a}$$
$$\mathbf{z}_\ell = \mathbf{W}_\ell \mathbf{x}_\ell + \mathbf{b}_\ell \tag{2.9b}$$
$$\mathbf{y}_\ell = f(\mathbf{z}_\ell) \tag{2.9c}$$

The activation functions employed in MLPs are generally point-wise non-linearities such as *hyperbolic tangent* $f(x) = \tanh(x)$ or logistic sigmoid $f(x) = \frac{1}{1+\exp(-x)}$, but in the recent years, non-saturating functions such as *Rectified Linear Units (ReLU)* (Glorot et al., 2011; Jarrett et al., 2009; Nair and Hinton, 2010) $f(x) = \max(0, x)$ or other variants (He et al., 2015) have been preferred thanks to their better trainability properties. See Section 2.7.4.3 for more details.

CONVOLUTIONAL AND POOLING LAYERS Similarly to MLPs, *Convolutional Neural Networks (CNNs)* (LeCun et al., 1998a; LeCun, 1989) are a kind of feedforward neural network where layers form a linear computational graph. CNNs are particularly well suited for signals with spatial regularities thanks to special layers designed to exploit spatial patterns in data with a grid-like topology such as images. The main difference with fully-connected layers consists of how the units are connected and how the input is processed to preserve its spatial structure. Rather than applying a large matrix of weights to the whole input, convolutional layers take advantage of the convolutional operator to process only a portion of the input at the time and apply the same linear transformation in a sliding window fashion — this way of processing spatial data results to be beneficial for multiple reasons. Firstly, each neuron is connected only to a portion of the input, the *receptive field*, drastically reducing the number of parameters and implementing a form of *weight sharing*. Secondly, the convolution operator is equivariant to any function $g(\cdot)$ that translates the input. In other words, applying a transformation $f(\cdot)$ to a shifted version of an input x, produces the same result as applying the transformation to x and then shifting the transformed input, namely $f(g(x)) = g(f(x))$. This property is particularly useful in image recognition tasks where the same local feature should be detected regardless of its position in the image.

The weights of a convolutional layer are usually called *filters* or *kernels*, an inheritance from the signal processing community. In general each convolutional layer learns a group of filters and the output after the activation function is a volume called *feature map*.

CNNs often reduce the spatial dimensionality of their inputs by applying the convolution operator every s pixels rather than at each location of the image, i.e., with *stride* $s > 1$, or by utilizing *pooling* or *sub-sampling* layers.

Commonly, pooling layers aggregate the receptive field by taking their *mean* or *max* value. Convolutional layers can also perform the opposite operation and increase the input size via an upsampling operation (Long et al., 2015; Simonyan and Zisserman, 2015; Zeiler and Fergus, 2014). Further details on convolutional and pooling layers can be found in the detailed guide to convolution arithmetic for deep learning (Dumoulin and Visin, 2016).

RECURRENT LAYERS In principle, NNs with feedforward connections can deal with sequential data by processing multiple inputs in a fixed-length window at the same time (Lang et al., 1990). These types of NNs are called *Time-Delay Neural Networks (TDNNs)*. From a signal processing point of view, a TDNN is equivalent to preprocessing a temporal sequence via a tapped delay and feeding it to a discrete-time nonlinear filter with a finite impulse response (FIR). The coefficients of the filter are the weights of the neural network and are learned from data.

Although Time-delay NNs showed to be promising for application such as speech processing (Lang et al., 1990; Waibel et al., 1989), they can only capture temporal dependencies that happen within the fixed and predetermined window size. A more flexible solution is represented by *Recurrent Neural Networks (RNNs)* that can potentially model temporal patterns that span an indefinite number of steps. RNNs can have different topologies and characteristics, see (Elman, 1990; Lang et al., 1990; Pearlmutter, 1995; Werbos, 1988; 1990), and (Jaeger, 2001; 2002; Maass et al., 2002) for some early architectures.

In modern neural networks, a simple recurrent layer (RNN) is obtained by augmenting a feedforward layer with feedback connections, so that the output at time step t is a function $F : \mathbb{R}^d \to \mathbb{R}^n$ of the inputs and its output at the previous time step, namely $F(\mathbf{x}_{t-1}, \mathbf{u}_t; \Theta)$, where all the parameters have been collected in Θ for brevity. The recurrent layer can be expressed by the following recursive equation:

$$\mathbf{x}_t = f(\mathbf{W}_{rec}\mathbf{x}_{t-1} + \mathbf{W}_{in}\mathbf{u}_t + \mathbf{b}) \tag{2.10a}$$

where $\mathbf{W}_{in} \in \mathbb{R}^{n \times d}$ is the weight matrix which transforms the current input, $\mathbf{W}_{rec} \in \mathbb{R}^{d \times d}$ is the matrix containing the parameters of the recurrent connections weights, which transform the previous output of the layer and $\mathbf{b}$ are the neurons biases. The internal state of the layer is mapped to the output vector $\mathbf{x}_t$ via a nonlinear activation such as *tanh* or *sigmoid*. The state $\mathbf{x}_0$ is typically initialized to zero or another constant value, but it can also be filled with random noise. Alternatively, the initial state can be learned during training.

Simple RNN layers are theoretically powerful and they can in principle learn any program that can be executed by a Turing machine (Siegelmann and Sontag, 1995), i.e., they are Turing Complete, but are hard to train with gradient-based algorithms. *Long Short-Term Memory* (LSTM) (Hochreiter and

Schmidhuber, 1997b) addresses the limitations of recurrent architectures by adopting a specific design to improve gradient flow during training. We discuss the difficulties of training RNN layers with gradient-based algorithms in Section 2.7.4.4, while the LSTM layer and its advantages in Section 2.7.5.1.

2.7.2 Training Neural Networks

Consider the case of a machine learning algorithm that should learn to solve a complex decision making problem. Games have always been the preferred benchmark to evaluate artificial intelligence programs' performance (Brown and Sandholm, 2018; Campbell et al., 2002; Silver et al., 2016; 2018; 2017), so consider, for instance, a board game such as chess or go. Let assume that the program receives a single bit of information that evaluates its performance at the end of the game, 0/1 — win or lose. The game's outcome is the result of several intermediate moves, each contributing to some extent to the program's final performance. The *credit assignment problem* (Minsky, 1961) arises when the learning algorithm has to evaluate the goodness of all the intermediate decisions that it had to make, understanding which decisions should have been made differently to obtain a better result. In particular, credit assignment is essential for efficiently finding the solution to complex problems in a finite (or reasonable) amount of time, and in consequence of that, speeding up the learning.

The scenario that we described seems to fit the only specific type of reinforcement learning problems involving sequential decision making but is more general than that. The credit assignment problem is also relevant to settings involving the learning of static input-output mappings. The success of modern supervised learning models relies on learning powerful features extractors that build hierarchical representations. Rather than manually designing features that might be sub-optimal, *Representation Learning* (Bengio et al., 2013) delegates the feature extraction phase to several units learned explicitly for the task at hand. Neural Networks implement this mechanism efficiently via a composition of differentiable functions. If properly trained, a deep neural network can produce an internal representation of the data with several levels of abstraction (Rumelhart et al., 1986), far more general than hand-designed features (Dosovitskiy and Brox, 2016; Erhan et al., 2009; Mahendran and Vedaldi, 2015; Simonyan and Zisserman, 2015; Yosinski et al., 2015; Zeiler and Fergus, 2014).

The process of extracting successive intermediate representations can be thought as taking sequential steps to compute the task's internal representation. Assigning appropriate credit to each of the units participating in the computation of the representation is crucial for fast and effective learning (Rumelhart et al., 1986). In the literature, the problem of correctly evaluating the contribution of specific nodes of a system involved in the computation of a prediction is known as *structural credit assignment*. In contrast,

temporal credit assignment refers to evaluating the contribution of actions to the outcome in sequential decision-making problems. Further details on temporal credit assignment can be found in Sutton and Barto (1998).

2.7.3 Backpropagation Algorithm

The *backpropagation algorithm*, or simply *backprop* is an efficient method for computing the derivative of the output of neural networks with respect to its parameters. In particular, backprop solves the credit assignment problem in neural networks by propagating the prediction error from the top of the network through all its modules, and evaluating their contributions to the final output.

The merit of proposing the backpropagation algorithm is often assigned to Rumelhart et al. (1986), who showed that neural networks could be successfully trained to learn useful representations; however, the first application of backprop to neural networks dates back to Werbos (1974), 1981. The algorithm was soon generalized to tackle the credit assignment problem in recurrent neural networks as well (Werbos, 1988). *Backpropagation Through Time (BPTT)* allows computing the gradients of an RNN as if it was a feedforward network by applying backprop to the unroll of its computational graph. More details on the history of backpropagation and a detailed overview of structural credit assignment in neural networks, can be found in Schmidhuber (2015).

Backprop requires first to process an input $\mathbf{x}$ and compute the final output $\hat{\mathbf{y}}$. This phase is generally called *forward propagation* or *forward pass* because the information flows forward from the input layer through the network. During training, the forward pass evaluates also a scalar performance metric, called *loss* or *cost function* $\mathcal{L}(\boldsymbol{\theta})$, which has to be differentiable. Then, backpropagation computes the derivatives of the loss function with respect to the network's parameters — this step takes the name of *backward pass* because the layers of the network are visited in reversed order. In a second stage, the derivatives are then used to compute the weights adjustments, usually via gradient descent. Is is important to highlight that these two stages are distinct.

We now follow the derivation as presented in Bishop (1995) and consider the backward pass of an MLP with $\ell = \{1, \dots, L\}$ layers described by the forward pass equations in 2.9. The gradient of the loss function $\mathcal{L}$ with respect to the parameters of the layer ℓ are:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_\ell} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}_L} \frac{\partial \mathbf{y}_L}{\partial \mathbf{W}_\ell} \tag{2.11a}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}_\ell} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}_L} \frac{\partial \mathbf{y}_L}{\partial \mathbf{b}_\ell} \tag{2.11b}$$

We first introduce a useful notation δ_ℓ for the layer ℓ, to indicate the *local error* of the layer with respect to the output, or simply the *delta*:

$$\delta_\ell = \frac{\partial \mathbf{y}_L}{\partial \mathbf{z}_\ell} \tag{2.12}$$

The derivatives of the output with respect to the parameters of layer ℓ can be expressed as:

$$\frac{\partial \mathbf{y}_L}{\partial \mathbf{W}_\ell} = \frac{\partial \mathbf{y}_L}{\partial \mathbf{z}_\ell}\frac{\partial \mathbf{z}_\ell}{\partial \mathbf{W}_\ell} \tag{2.13a}$$

$$\frac{\partial \mathbf{y}_L}{\partial \mathbf{b}_\ell} = \frac{\partial \mathbf{y}_L}{\partial \mathbf{z}_\ell}\frac{\partial \mathbf{z}_\ell}{\partial \mathbf{b}_\ell} \tag{2.13b}$$

From the forward pass in Equation (2.9c), the derivative of the pre-activation variable $\mathbf{z}_\ell$ with respect to its parameters is given by the input of the layer $\mathbf{x}_\ell$, namely the activation of the previous layer:

$$\frac{\partial \mathbf{z}_\ell}{\partial \mathbf{W}_\ell} = \mathbf{x}_\ell^T \tag{2.14a}$$

$$\frac{\partial \mathbf{z}_\ell}{\partial \mathbf{b}_\ell} = \mathbb{1} \tag{2.14b}$$

By replacing Equation (2.12) in Equation (2.13) we can write:

$$\frac{\partial \mathbf{y}_L}{\partial \mathbf{W}_\ell} = \delta_\ell \mathbf{x}_\ell^T \tag{2.15a}$$

$$\frac{\partial \mathbf{y}_L}{\partial \mathbf{b}_\ell} = \delta_\ell \tag{2.15b}$$

The key feature of backpropagation is that the activations computed in the forward pass can be stored to make the computation of the backward pass much more efficient. Thus, evaluating the derivatives only requires to calculate the value of δ_ℓ for the output and hidden layers and then apply Equation (2.15):

$$\delta_L = f'(\mathbf{z}_L) \tag{2.16a}$$

$$\delta_\ell = \mathbf{W}_{\ell+1}^T \delta_{\ell+1} \cdot f'(\mathbf{z}_\ell) \tag{2.16b}$$

The backpropagation formulae tell us that the contribution of the units of each layer can be computed by propagating the δ backwards from units higher up in the network. Interestingly, backprop is a special case for scalar-valued functions of the *reverse accumulation mode* (Griewank and Walther, 2008) for automatic differentiation. For more details on automatic differentiation techniques we recommend this survey Baydin et al. (2017).

2.7.4 Training challenges of Neural Networks

Training models with a large number of parameters such as neural networks requires solving a complex optimization problem. In general training NNs is *NP-hard*, and there is no efficient algorithm for finding a solution to the optimization problem for general topologies and tasks (Blum and Rivest, 1992; Hammer and Villmann, 2003; Judd, 1990).

In practice, the training complexity of NNs is overcome by exploiting their differentiability and the efficiency of the backpropagation algorithm in computing derivatives of complex compositions of functions. Indeed, under the correct hyper-parameters choice, simple gradient methods are able to find optimal parameter configurations that are global minimizers for a given training set (Boyd et al., 2004).

Unfortunately, gradient-based learning is not free of issues, especially when it involves optimizing millions of parameters, such as in NNs. Although NNs are universal function approximators, and in principle, even the simplest MLP is capable of learning any function (Hornik, 1991; Hornik et al., 1989), the optimization algorithm could fail in minimizing the objective in terms of performance or available time budget, resulting in underfitting the training data. Specifically, the loss landscape of NNs is in general non-convex with multiple saddle-points, local minima, and plateau, causing gradient-based optimization to be highly unstable or to stagnate in poor solutions (Choromanska et al., 2015; Dauphin et al., 2014; Hochreiter and Schmidhuber, 1997a; Li et al., 2018a). These difficulties have motivated specific techniques and model architectures to improve the training efficiency of gradient-based algorithms.

2.7.4.1 Gradient-based optimization

Stochastic Gradient descent follows the negative direction of the expected gradient to minimize a certain functional in the weights space. When the learning rate decreases with the appropriate rate, this technique converges almost surely to the global minimum if the objective function is convex (Bottou, 1998; Robbins and Monro, 1951). On the contrary, neural networks generally provide non-convex optimization problems, limiting our ability to characterize their loss surface and to develop convergence results for the optimization algorithms. Indeed, the composition of several nonlinear functions results in loss landscapes presenting high dimensional *valleys*, *plateau* or *ravines* (Sutton, 1986) that requires different step sizes along different dimensions in order to be escaped. In this situation, following the actual steepest descent might not be the best strategy. In a valley, the gradient direction is almost perpendicular to the flat axis and the updates oscillate back and forth in the direction of the short axis, moving very slowly along the long axis. A widely used technique to accelerate the convergence of gradient descent is the use of a *momentum* term (Nesterov, 1983b; Polyak,

1964) such that the weight update at the current time step depends both on the current gradient and the weight update at the previous step. The momentum term helps to average out the oscillations on the short axis while adding up contributions along the long axis, leading to faster convergence. The update rule of gradient descent with the momentum term at iteration i-th:

$$\begin{aligned} V^{(i+1)} &= \beta V^{(i)} + \nabla \mathcal{L}(\theta^{(i)}) \\ \theta^{(i+1)} &= \theta^{(i)} - \eta V^{(i+1)} \end{aligned} \tag{2.17}$$

where V is the momentum buffer, η is the learning step and β the amount of momentum applied. Note that with $\beta = 0$ the original gradient descent formulation is recovered.

SECOND-ORDER METHODS Alternatively, second-order approximations such as Newton's method incorporate information about the curvature of the loss function into the optimization algorithm by rescaling the gradient with the inverse of the Hessian (Nocedal and Wright, 2006):

$$\theta^{(i+1)} = \theta^{(i)} - \mathbf{H}^{-1} \nabla \mathcal{L}(\theta^{(i)}) \tag{2.18}$$

However, in order to succeed, Newton's method requires the Hessian to be positive definite, namely invertible. When the loss surface is non-convex in high dimensions such as in NN, it contains many saddle points, and the application of Newton's method could result problematic (Dauphin et al., 2014). Near saddle points, the eigenvalues of the Hessian are not all positive, and then Newton's method can cause updates to move in the wrong direction. This situation can be avoided by regularizing the Hessian, simply summing a diagonal term. The convergence rate of gradient descent method, in fact, is heavily influenced by the *condition number*, the ratio between the largest and smallest eigenvalues of the Hessian, $\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}$ which should be close to one. If the condition number is high, the optimization problem is *ill-conditioned* and the convergence rate is slow. One important issue with second-order methods is that the size of the Hessian grows with $\Theta(n_\theta^2)$, quadratically with the number of parameters n_θ, quickly becoming problematic for NNs both in terms of space and the inverse time complexity. To solve this problem, one could rely on diagonal approximations, making the space complexity linear and the inverse computation trivial, trading off useful information on how different directions interact with each other. See (LeCun et al., 1998b) for an earlier review on the application of second-order methods to the training of NNs, while a comprehensive treatment of numerical optimization methods can be found in Nocedal and Wright (2006) and Boyd et al. (2004).

NATURAL GRADIENT Another family of gradient-based optimization techniques takes into account the probabilistic structure of the underlying manifold of the model parameters. Each parameter value θ induces an instance

of a function f_θ that belongs a family parametric models $\mathcal{F}$. Intuitively, two very different parameters values θ_1 and θ_2 (in terms of Euclidean distance) could induce very similar functions f_{θ_1} and f_{θ_2}, or conversely two very close parameters could induce very different functions. This is because the Euclidean distance in the parameters space does not necessarily reflect the actual distances between the induced functions. Rather than moving in the parameter space, *Natural gradient descent* (Amari, 1998; Pascanu and Bengio, 2014; Roux et al., 2008) moves directly in the functional manifold induced by the mapping from θ to f_θ by defining a proper distance in the function space independently by their parametrization. This can be achieved by relying on the probabilistic interpretation of neural networks, which can be seen as modeling the conditional probability $p(y|x)$. The distance between two functions f_{θ_1} and f_{θ_2} can be identified by the KL-divergence between their corresponding probabilistic interpretations p_{θ_1} and p_{θ_2}. The second-order term of Taylor's expansion of the KL-divergence results in the well known Fisher Information Matrix (FIM) (Ly et al., 2017; Martens, 2020):

$$\mathbf{F}_\theta = \mathop{\mathbb{E}}_{\substack{y\sim p_\theta(y|x) \\ x\sim p_{\text{data}}}} \left[\nabla_\theta \log p_\theta(y \mid x) \nabla_\theta \log p_\theta(y \mid x)^T\right] \tag{2.19}$$

Natural gradient descent corrects the gradient directions with the inverse of the FIM in place of the Hessian matrix, which ensures to follow the steepest descent in the Riemannian parameter manifold:

$$\theta^{(i+1)} = \theta^{(i)} - \eta \mathbf{F}_\theta^{-1} \nabla \mathcal{L}\left(\theta^{(i)}\right) \tag{2.20}$$

Similarly to the Hessian matrix, the FIM scales quadratically with n_θ which is often impractical for large networks, requiring efficient approximations that are much compact and easier to invert (Martens and Grosse, 2015). For a complete review of the field of Information Geometry we recommend the readers to check (Amari, 2016; Amari and Nagaoka, 2007).

ADVANCED FIRST-ORDER METHODS Because of the difficulties of second-order methods, a variety of successful first-order alternatives have been proposed and popular optimizers such as AdaGrad (Duchi et al., 2011), RMSProp (Hinton et al., 2012), Adadelta (Zeiler, 2012) or Adam (Kingma and Ba, 2015) are often the preferred choices for training NNs. By making use of momentum terms and automatically adapting the learning rates per dimension (Schaul et al., 2013), these methods are surprisingly effective when used in conjunction with tricks that ease the optimization process. Most of these heuristics apply normalization techniques to center and decorrelate the input of each layer (Desjardins et al., 2015; Ioffe and Szegedy, 2015; Salimans and Kingma, 2016), inspired by standard data pre-processing methods. Most of these methods can be seen as different preconditioning of the gradient that adapts the geometry of the data to improve the rate of convergence of gradient descent.

2.7.4.2 Parameters initialization strategies

The success of iterative algorithms such as gradient descent in finding optimal solutions generally depends on the initial point of the optimization process. The initial values of the parameters can determine whether the algorithm successfully converges or not, and the speed of convergence. Regularization techniques such as weight decay (Section 2.5), suggest that the weights should be small enough in order to avoid overfitting, but in general, having too small weights can severely attenuate signals as they propagate through a network — activations in the forward pass and gradients in the backward pass. Modern initialization strategies follows simple statistical intuitions and heuristics which impose uniform or gaussian prior with zero mean. Most of the popular initialization techniques are designed to make sure that all layers have the same activation variance and that the gradients variance remain constant and close to one over the layers of the network (Glorot and Bengio, 2010). For instance, Sussillo and Abbott (2014) showed that keeping constant the logarithm of the norms of the backpropagated errors is sufficient to train deep networks. The initial scale of the weights critically depends on the types of activation used in the networks (He et al., 2015).

An interesting recent line of research studies signal propagation in deep networks through the lens of statistical physics (Poole et al., 2016; Raghu et al., 2017; Schoenholz et al., 2017). Pennington et al. (2017), 2018 show that NNs train well when their input-output Jacobians exhibit *dynamical isometry*, namely the property that the distribution of singular values concentrate close to 1. This condition can be achieved for deep *linear* networks via orthogonal weight initialization (Saxe et al., 2014), making training time independent from the depth of the network. While not as fast as linear networks, sigmoidal networks with orthogonal initialization have training times growing as the square root of depth Pennington et al. (2017). Similar initializations can be obtained for other types of networks as well (Chen et al., 2018a; Xiao et al., 2018b), making the training of very deep networks possible.

2.7.4.3 Activation functions

One of the main issues with activation functions such as *sigmoid* and *tanh* is that when they operate in saturating regimes, their gradient is small, or potentially zero, being detrimental for learning. For this reason, non-saturating activation functions such as Rectified Linear Units (ReLUs) (Glorot et al., 2011; Jarrett et al., 2009; Nair and Hinton, 2010), have become very popular in many applications. This type of activation function, defined as $y = \max(0, x)$, does not saturate for positive inputs, and is less likely to contribute to the diminishing of the gradient during backpropagation, since its derivative is the identity for positive values and zero otherwise. Although

they present a saturating behaviour for negative input, if properly initialized, these units showed to be very effective for learning very large networks.

However, ReLUs are always non-negative, which causes the mean activation to be non-zero and act as a bias for the next layer. The more the units within a layer are correlated, the higher its bias shift. This phenomenon could introduce undesirable dynamics that could slow down or prevent training. For the same reason, tanh has been preferred over logistic functions before the ReLUs took over saturating activations (LeCun et al., 1998b; LeCun et al., 1991). Other piecewise activation functions have been proposed to improve traditional ReLU by replacing the saturation part, with linear behaviour at small or learnable slope (He et al., 2015; Maas et al., 2013).

A practical approach for reducing the *bias shift* is centering the activations around zero. This method has been proposed in many forms and with different perspectives on the learning problem (LeCun et al., 1998b; Raiko et al., 2012; Schraudolph, 1998). More recently, Ioffe and Szegedy (2015) propose *Batch Normalization* (BatchNorm or BN) to specifically address the *covariate shift* (Shimodaira, 2000) by centering and scaling the activations of each layer through the statistics of the batch. An affine transformation is also learned to adjust the amount of normalization that should be taken into account. BatchNorm accelerates learning reducing the sensitivity to the initialization of the parameters, and it also regularizes the model reducing the need for other regularization techniques as Dropout (Srivastava et al., 2014). However, the exact reasons behind its numerous benefits are still unclear and a major topic of scientific debate (Benz et al., 2021; Santurkar et al., 2018; Yang et al., 2019). On the same line, several other normalization techniques have been proposed to speed up training in different applications or learning scenarios (Ba et al., 2016; Desjardins et al., 2015; Ioffe and Szegedy, 2017; Ulyanov et al., 2016; Wu and He, 2018). Exponential Linear Units (ELU) (Clevert et al., 2016) explicitly addresses the bias shift issue by designing a function that pushes the mean activations at each layer closer to zero. Building upon ELUs, Self-normalizing networks (Klambauer et al., 2017) showed that it is sufficient to adopt *scaled exponential linear unit* (SELUs) for the activations propagated through the network to converge to a fixed point with zero mean and unit variance without requiring explicit normalization.

2.7.4.4 Exploding and Vanishing Gradients

We have discussed how credit assignment can be efficiently performed in neural networks via the backpropagation algorithm. We now discuss two main issues with that affect credit assignment when the depth of the networks increases, preventing training deep neural networks.

For simplicity we follow the same analysis of (Pascanu et al., 2013) and consider the following parametrization of the vanilla RNN:

$$\mathbf{x}_t = \mathbf{W}_{rec} f(\mathbf{x}_{t-1}) + \mathbf{W}_{in}\mathbf{u}_t + \mathbf{b} \tag{2.21a}$$

that is equivalent to the equation in Equation (2.10), but simpler to analyze. To further simplify the analysis, we include the bias term in $\mathbf{W}_{rec}$. The loss is the sum of the contribution each time step $t = \{1, \dots T\}$, hence the gradient with respect to general parameters θ is given by the sum of the gradients:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}_t}{\partial \theta} \tag{2.22a}$$

$$\frac{\partial \mathcal{L}_t}{\partial \theta} = \sum_{k=1}^{t} \left(\frac{\partial \mathcal{L}_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial \mathbf{x}_k}{\partial \theta} \right) \tag{2.22b}$$

$$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} = \prod_{i=k+1}^{t} \mathbf{W}_{rec}^T \operatorname{diag}(f'(\mathbf{x}_{i-1})) \tag{2.22c}$$

where the *diag* operator converts a vector into a diagonal matrix, and f' computes the derivative of an element-wise function. Any gradient component $\frac{\partial \mathcal{L}_t}{\partial \theta}$ is the sum of multiple *temporal* components, each measuring how the parameters at step k affect the cost at the successive steps $t > k$. In particular, Equation 2.22c transports the error from step t back to step k, computing the *temporal Jacobian* $\frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}}$ at each step for the specific parametrization in Equation (2.21).

Equation 2.22c takes the form of a product of $t - k$ Jacobian matrices. We can restrict ourselves to linear functions and analyse its behaviour, as the sequence length grows, by applying the *power iteration method*, but for a more general analysis we can consider nonlinear function where $f'(x)$ is bounded by $\gamma \in \mathbb{R}$ and σ_{max} be the largest singular value of $\mathbf{W}_{rec}$:

$$\left\| \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} \right\| \leq \left\| \mathbf{W}_{rec}^T \right\| \left\| \operatorname{diag}(f'(\mathbf{x}_{i-1})) \right\| \leq \gamma \sigma_{max} \tag{2.23a}$$

$$\left\| \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \right\| \leq \prod_{i=k+1}^{t} \left\| \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} \right\| \leq \| \gamma \sigma_{max} \|^{(t-k)} \tag{2.23b}$$

We can observe that, if the product $\gamma \sigma_{max}$ is less than one, $\left\| \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \right\|$ becomes exponentially smaller as the time delay $t - k$ increases. Similarly, the product of the temporal Jacobians can grow explonentially fast. Knowing that $\gamma < 1$ for tanh and $\gamma < \frac{1}{4}$ for the sigmoid function, we can recover a *necessary* condition for the gradients to explode, namely $\sigma_{max} > 1$ for tanh, and a tighter version for the sigmoid $\sigma_{max} > 4$. The *vanishing* and *exploding gradients* phenomena have been first observed by (Hochreiter, 1991) and later formalized by other groups (Bengio et al., 1993; 1994; Pascanu et al., 2013) from a dynamical systems perspective.

The same analysis can be applied to feedforward neural networks, with slight changes. From the application of the BPTT algorithm, we know that unrolling RNN for T time steps is equivalent to have an FNN with T layers, hence vanishing and exploding gradients also affect the quality of learning in deep FNNs. The main differences are that

1. FNNs have input only at the first layer, while RNNs have inputs at each time step.

2. FNNs uses different weights $\mathbf{W}_\ell$ for each layer ℓ, while RNNs share the same weight matrix $\mathbf{W}_{rec}$ at each time step, namely are time-invariant systems.

We will further stress differences and similarities of recurrent and feedforward architectures in Chapter 3 by characterizing them from a dynamical systems perspective.

2.7.5 Strategies for training Deep Neural Networks

The vanishing and exploding gradients issues have been impeding the training of RNNs capable of modeling long term dependencies of input sequences, and powerful deep FNNs requiring the development of new techniques.

2.7.5.1 LSTM

Hochreiter and Schmidhuber (1997b) propose a radically new design — the *Long Short-Term Memory (LSTM)* architecture — to alleviate the effect of vanishing gradient in RNNs allowing them to modeling long term dependencies across of hundreds of steps. The key feature of LSTM is the use of an internal *memory cell* which can be accessed through additional *gates*, computational nodes that control the flow of information into and out of the cell, called *input* and output gates. As opposed to vanilla RNNs, this specific design allows LSTMs to separate their output from the memory cell.

The original version of the memory cell was designed to maintain a constant error flow during backpropagation through a recurrent self-connection with *constant weight* equal to one — the *Constant Error Carousel (CEC)*. Later, Gers et al. (1999) propose to incorporate an extra unit, the *forget gate*, whose output is multiplied with the cell's recurrent connection. This simple modification enables the memory to reset when needed by producing forget gate outputs close to zero, which is crucial for successfully solving many sequential tasks. All components are differentiable and can be efficiently trained with BPTT, to learn how to read, write, and reset the memory directly from data (Graves and Schmidhuber, 2005).

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \mathbf{h}_{t-1} + \mathbf{W}_i \mathbf{x}_t + \mathbf{b}_i) \tag{2.24a}$$
$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{h}_{t-1} + \mathbf{W}_f \mathbf{x}_t + \mathbf{b}_f) \tag{2.24b}$$
$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c \mathbf{h}_{t-1} + \mathbf{W}_c \mathbf{x}_t + \mathbf{b}_c) \tag{2.24c}$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \tag{2.24d}$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{h}_{t-1} + \mathbf{W}_o \mathbf{x}_t + \mathbf{b}_o) \tag{2.24e}$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \tag{2.24f}$$

Since its introduction the vanilla LSTM (see Equation (2.24)) has been the most commonly used architecture for handling sequential data, but other variants have been suggested too. Gers et al. (2002) suggest one variant of LSTMs with *peephole connections* from the memory cell to the gates, arguing that in order to learn precise timings, the memory cell needs to influence the gates directly. Neil et al. (2016) introduce an additional *time gate* that allows the unit to update the cell, and the output, at irregularly sampled time steps. Cho et al. (2014) propose a simplification of the LSTM architecture called *Gated Recurrent Unit* (GRU) where the input gate and the forget gate are coupled into a single *update gate*. The memory cell and output are also merged into a single state, removing the output activation. Finally, the output gate (called *reset gate*) only gates the recurrent connections to the block input. For a thorough overview of the different LSTM variants and their comparison, we recommend the survey by Greff et al. (2016). We also recommend to check Graves (2008)'s PhD thesis for pioneering applications of LSTM on sequential tasks.

BEYOND 1-DIMENSIONAL TEMPORAL SERIES The LSTM mechanism has been also adapted to model complex dependencies in multidimensional inputs (Graves et al., 2007; Graves and Schmidhuber, 2009; Stollenga et al., 2015). Alternatively, *convLSTM* (Shi et al., 2015) replaces matrix multiplications with the convolution operator, preserving the spatial structure of the inputs to learn spatio-temporal dependencies. Extensions of multidimensional LSTM and convLSTM have being particularly successful in image and video prediction applications (Byeon et al., 2018; Su et al., 2020). Visin et al. (2015) propose *ReNet*, an alternative to conventional CNNs architectures. ReNet captures spatial correlation across pixels by sweeping images horizontally and vertically with two bidirectional LSTM cells, showing its efficacy for structured prediction tasks such as semantic segmentation (Visin et al., 2016).

In this thesis, we make abundant use of recurrent architectures to learn representations for visual data, having both temporal and spatial structures. In Chapter 4 we show how LSTM can be used to represent events-based data modeling both temporal and spatial dependencies across events. In Chapter 5 we improve the quality of video object segmentation models with convLSTM, generating more coherent and temporal consistent segmentation masks.

2.7.5.2 *Skip connections*

Skip connections modify the flow of information in feedforward networks by adding direct links from lower to higher layers. Intuitively, such connections improve credit assignment to lower layers by introducing shorter paths that bypass many layers to propagate the gradient and assist learning, reducing signals attenuation. [1]

EARLIER MODELS Skip connections are fundamental building blocks for the success of modern neural architectures, but their introduction traces back to some of the earliest work on neural networks.

Lang and Witbrock (1988) is the first presenting the use of *shortcut connections*, in a densely connected architecture that provides direct connections from each layer to all the higher layers. This work was already pointing toward using skip connections to train deeper networks, although the vanishing gradient problem had not been formally identified yet. Fahlman and Lebiere (1990) proposed the Cascade-Correlation architecture to incrementally increase the depth of a neural network by freezing existing units and adding new ones. New layers received inputs from all the existing ones, implicitly implementing skip connections from lower layers. However, being fixed during training, the lower units did not benefit from the better gradient flow provided by shortcut connections. Other work (Kalman and Kwasny, 1997; Kalman et al., 1993; Lee and Holt, 1992) used a skip connection directly from the input to the output of the network to separate the learning of linear and non-linear components and speed-up the training. In the context of bayesian neural networks, Neal (1995, Chapter 2) suggest that some pathologies in learning representations can be fixed by making each layer also depend on the original input. Later, Duvenaud et al. (2014) studied the use of *input-connected* networks to improve the training of deep Gaussian processes.

RECENT DEVELOPMENTS Inspired by LSTM, Srivastava et al. (2015a), b proposed a similar mechanism for regulating the information flow in feedforward neural networks. *Highway Networks* improve signals propagation with respect to *plain* networks, by adding a shortcut connection from the input of the ℓ^{th} layer and its output; two learned gates, the *transform gate* $\mathbf{T}(\cdot;\theta_{\mathbf{T}})$ and *carry gate* $\mathbf{C}(\cdot;\theta_{\mathbf{C}})$ control the contribution of the layer's output and the original input, respectively. These two gates are analogous to the *input* and *forget gates* in the LSTM architecture:

$$\mathbf{x}_{\ell+1} = \mathbf{C}(\mathbf{x};\theta_{\mathbf{C}}) \cdot \mathbf{x}_{\ell} + \mathbf{T}(\mathbf{x};\theta_{\mathbf{T}}) \cdot f\left(\mathbf{x}_{\ell};\theta_{\ell}\right) \tag{2.25}$$

1 This section is based on the literature review presented in Srivastava (2018).

Concurrently, He et al. (2016) introduced *Residual Networks (ResNets)*, similar architectures with *additive skip* or *residual* connections from the input to the output of each layer:

$$\mathbf{x}_{\ell+1} = \mathbf{x}_\ell + f\left(\mathbf{x}_\ell; \theta_\ell\right), \tag{2.26}$$

where $f(\cdot; \theta_\ell)$ is a nonlinear transformation of the previous layer, either convolutional or fully-connected. ResNets correspond to Highway Networks in the special case where both transform and carry gates are set to the identity. Thanks to their particular design, Highway and Residual Networks allow having paths along which information can flow across several layers without attenuation making them easier to optimize even for very large depths.

Since their introductions, ResNets showed their predominance on image recognition tasks with respect to plain networks, becoming soon the gold standard for computer vision applications, and guiding the design and the evolution of many convolutional skip connections-based architectures (Huang et al., 2017; 2016; Larsson et al., 2017; Xie et al., 2017).

LONG SKIP CONNECTIONS Long skip connections are often employed to mitigate the negative effect of the *bottleneck* in *encoder-decoder* architectures. Networks with this structure, in fact, compress and expand the dimensionality of the inputs through several downsampling and upsampling transformations, causing the degradation of low-level details. Long skip connections bypass the bottleneck, connecting features extracted from lower layers of the encoder directly to the decoder recovering details that might have been lost during the chain of computation. The combination of long and skip connections has been shown to improve performance on a variety of computer vision tasks such as (Jégou et al., 2017; Long et al., 2015; Ronneberger et al., 2015), medical image segmentation (Drozdzal et al., 2016), image restoration (Mao et al., 2016) and optical flow estimation (Dosovitskiy et al., 2015).

THEORETICAL MOTIVATION The benefits of skip connections on the training of very deep architectures have been demonstrated in several fields of machine learning. However, there are still several unanswered questions on why adding shortcut connections helps optimization and what kind of representations is learned by these architectures.

Researchers have studied skip connections from several point of views. From a biological perspective, skip connections can be motivated by the presence of similar connections between non-adjacent layers in primates' visual cortex (Maunsell and Essen, 1983). Balduzzi et al. (2017) investigate why usual remedies such as proper initializations and normalizations are not sufficient for training deep networks showing that they suffer a further problematic besides vanishing and exploding gradients. The non-smoothness of rectifier activation functions cause *gradient shattering*: gradients of deeper lay-

ers in the network become increasingly uncorrelated. In this scenario, averaging gradients over mini-batches corresponds to integrating over white-noise, making it impossible to improve the model following specific directions. The authors show the importance of skip connections to reduce the gradient shattering problem. Li et al. (2018a) explore the structure of the loss landscape of neural networks and show that skip connection-based architectures present much smoother shapes suggesting that skip-connections aid gradient-based optimization simplifying the minimization problem making it more convex. Orhan and Pitkow (2018) conjectures that the difficulty of training deep networks is partly due to the *singularities* caused by the non-identifiability of the model and that skip connections eliminate these singularities.

In the next chapter, we analyze residual networks from a different perspective by observing that their definitions correspond to the forward discretization of an ordinary differential equation (Haber et al., 2018; Weinan, 2017). Hence, we study the stability of the discretized system and build a new architecture that can be unrolled to perform iterative computation until convergence to input-dependent equilibria.

3 NON-AUTONOMOUS INPUT-OUTPUT STABLE NEURAL NETWORKS

This chapter reinterprets Residual Networks from a new dynamical system perspective, and describes the *"Non-Autonomous Input-Output Stable Network"* (NAIS-Net), a novel very deep architecture where each processing block is derived from a *time-invariant non-autonomous* system.

NAIS-Net is *stable by design* since it is built by stacking a cascade of *stable* dynamical systems that can be unrolled indefinitely until convergence, implementing a conditional form of iterative computation Specifically, we prove that the network is *globally asymptotically stable* so that for every initial condition there is exactly one *input-dependent equilibrium* assuming tanh units, and incrementally stable for ReLU. We present an efficient implementation that enforces the stability under the derived conditions for both fully-connected and convolutional layers. Experimental results show how NAIS-Net exhibits stability in practice, yielding a significant reduction in generalization gap compared to ResNets.

3.1 INTRODUCTION

Deep neural networks are now the state-of-the-art in a variety of challenging tasks, ranging from object recognition to natural language processing and graph analysis (Battenberg et al., 2017; Krizhevsky et al., 2012; Monti et al., 2017; Sutskever et al., 2014; Zilly et al., 2017). With enough layers, they can, in principle, learn arbitrarily complex abstract representations Bengio et al., 2013; Rumelhart et al., 1986 through a composition of nonlinear mappings. Each layer transforms the output from the previous one until the input is embedded in a latent space where inference can be efficiently made.

Until the advent of Highway (Srivastava et al., 2015a) and Residual (He et al., 2016) networks, training architectures beyond a certain depth with gradient descent was limited by the vanishing gradient problem (Bengio et al., 1994; Hochreiter, 1991). These Very Deep Neural Networks (VDNNs) have *skip connections* that provide shortcuts for the gradient to flow back through hundreds of layers. Building these types of architectures corresponds to imposing a specific *inductive bias* on the learned model, where layers incrementally refine features, rather than computing new representations at each step, implementing an *iterative process* (Greff et al., 2017; Jastrzebski et al., 2018).

With this in mind, one could argue that the optimal number of computational steps should depend on the specific sample processed by the network and that the optimal number of layers or *"processing depth"* should not be fixed a priori. In physics, an iterative process that terminates after an arbitrary number of computational steps is generally formulated as a dynamical system, which, mathematically, can be most simply described as an *ordinary differential equation* (ODE).

Recently, several researchers have started to view neural networks with additive skip connections from a dynamical systems perspective. Haber and Ruthotto (Haber and Ruthotto, 2017) analyzed the stability of ResNets by framing them as an Euler integration of an ODE, and (Lu et al., 2018) showed how using other numerical integration methods induces various existing network architectures such as PolyNet (Zhang et al., 2017), FractalNet (Larsson et al., 2017) and RevNet (Gomez et al., 2017).

AUTONOMOUS VS NON-AUTONOMOUS SYSTEMS A fundamental problem with the dynamical systems underlying these architectures is that they are *autonomous*: the input pattern sets the initial condition, only directly affecting the first processing stage. This means that if the system converges, there is either exactly one fixed-point or exactly one limit cycle (Strogatz, 2015). Neither case is desirable from a learning perspective because a dynamical system should have input-dependent convergence properties so that representations are useful for learning. One possible approach to achieve this is to have a *non-autonomous* system where, at each iteration, the system is forced by an external input.

TIME-VARYING VS TIME-INVARIANT SYSTEMS Moreover, because all of these networks correspond to *time-varying* dynamical systems by virtue of having a different set of weights at each processing stage (iteration), they cannot control pattern-dependent processing depth since all samples must be fed through all trained layers for proper inference. Jastrzebski et al. (2018) tried to unroll the last layers of trained residual architectures based on heuristics and aggressive normalization techniques in order to improve performance of the model on certain samples, but there are no guarantees that the model would converge to an input-dependent equilibrium point. For that, the network must behave as a *time-invariant* dynamical system where the weights are shared across process stages, so that the network can be *"unrolled"* into as many stages as needed for each pattern by repeatedly applying the same transformation.

3.1.1 Main contributions

This chapter introduces a novel network architecture, called the *"Non-Autonomous Input-Output Stable Network"* (NAIS-Net), that is derived from

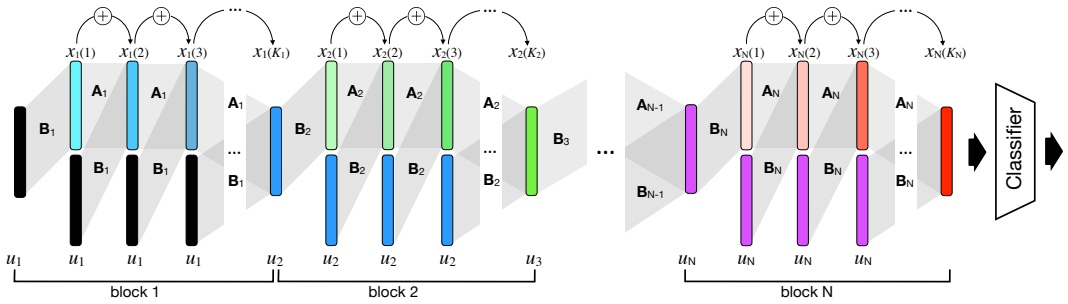


Figure 3.1: NAIS-Net architecture. Each block represents a time-invariant iterative process as the first layer in the i-th block, $\mathbf{x}_i(1)$, is unrolled into a pattern-dependent number, K_i, of processing stages, using weight matrices $\mathbf{A}_i$ and $\mathbf{B}_i$. The skip connections from the input, $\mathbf{u}_i$, to all layers in block i make the process non-autonomous. Blocks can be chained together (each block modeling a different latent space) by passing final latent representation, $\mathbf{x}_i(K_i)$, of block i as the input to block $i+1$.

a dynamical system that is both time-invariant (weights are shared) and non-autonomous.[1] NAIS-Net is a general residual architecture where a block (see Figure 3.1) is the result of the unrolling of a time-invariant system, and non-autonomy is implemented by having the external input applied to each of the unrolled processing stages in the block through skip connections. ResNets are similar to NAIS-Net except that ResNets are time-varying and only receive the external input at the first layer of the block. With this design, we can derive sufficient conditions under which the network exhibits well behaved trajectories for every initial condition. More specifically:

- In Section 3.3, we prove that with *tanh* activations, NAIS-Net has exactly one input-dependent equilibrium, while with ReLU activations lead to incrementally stable trajectories per input pattern. Moreover, the NAIS-Net architecture allows not only the internal stability of the system to be analyzed but, more importantly, the input-output stability — the difference between the representations generated by two different inputs belonging to a bounded set will also be bounded at each stage of the unrolling.[2] An analysis of the forward pass dynamics of NAIS-Net layers is carried out in Section 3.5.

- In Section 3.4, we provide an efficient implementation that enforces the stability conditions for both fully-connected and convolutional layers in the stochastic optimization setting.

- These implementations are compared experimentally with ResNets on both CIFAR-10 and CIFAR-100 datasets, in Section 3.6, showing

1 The DenseNet architecture (Huang et al., 2017; Lang and Witbrock, 1988) is non-autonomous, but time-varying.

2 In the appendix material, we also show that these results hold both for shared and unshared weights.

that NAIS-Nets achieve comparable classification accuracy with a much better *generalization gap*. NAIS-Nets can also be 10 to 20 times deeper than the original ResNet without increasing the total number of network parameters, and, by stacking several stable NAIS-Net blocks, models that implement pattern-dependent processing depth can be trained without requiring batch normalization Ioffe and Szegedy, 2015 at each step (except when there is a change in layer dimensionality, to speed up training).

The next section presents a more formal treatment of the dynamical systems perspective of neural networks, and a brief overview of work to date in this area.

3.2 BACKGROUND AND RELATED WORK

Representation learning is about finding a mapping from input patterns to encodings that disentangle the underlying variational factors of the input set. With such an encoding, a large portion of typical supervised learning tasks (e.g. classification and regression) should be solvable using just a simple model like logistic regression. A key characteristic of such a mapping is its invariance to input transformations that do not alter these factors for a given input[3].

In particular, random perturbations of the input should in general not be drastically amplified in the encoding. In the field of *control theory*, this property is central to *stability analysis*, which investigates the properties of dynamical systems under which they converge to a single steady-state without exhibiting chaos (Khalil, 2014; Sontag, 1998; Strogatz, 2015).

3.2.1 Residual Networks: a dynamical system perspective

Almost all successfully trained VDNNs (Cho et al., 2014; He et al., 2016; Hochreiter and Schmidhuber, 1997b; Srivastava et al., 2015a) share the following core building block:

$$\mathbf{x}(k+1) = \mathbf{x}(k) + f\left(\mathbf{x}(k), \theta(k)\right), 1 \leq k \leq K. \tag{3.1}$$

That is, in order to compute a vector representation at layer $k+1$ (or time $k+1$ for recurrent networks), *additively update* $\mathbf{x}(k)$ with some non-linear transformation $f(\cdot)$ of $\mathbf{x}(k)$ which depends on parameters $\theta(k)$. The reason usual given for why Eq. (3.1) allows VDNNs to be trained is that the explicit identity connections avoid the vanishing gradient problem.

3 Such invariance conditions can be very powerful inductive biases on their own: For example, requiring invariance to time transformations in the input leads to popular RNN architectures (Tallec and Ollivier, 2018).

The semantics of the forward path are however still considered unclear. A recent interpretation is that these feed-forward architectures implement *iterative inference* (Greff et al., 2017; Jastrzebski et al., 2018). This view is reinforced by observing that Eq. (3.1) is a forward Euler discretization (Ascher and Petzold, 1998) of the *Ordinary Differential Equation* (ODE):

$$\dot{x}(t) = f(\mathbf{x}(t), \Theta) \tag{3.2}$$

in the case of shared weights across layers, $\theta(k) \equiv \Theta$ for all $1 \leq k \leq K$. In fact, it is trivial to show that the forward propagation of Eq. (3.1) can be rewritten as:

$$\frac{\mathbf{x}(k+1) - \mathbf{x}(k)}{h} = f(\mathbf{x}(k), \Theta). \tag{3.3}$$

The left hand side of the above equation is the finite difference approximation to the differential operator $\partial\mathbf{x}$ with step h that corresponds to the ODE in Eq. (3.2) for $h \to 0$.

This connection between dynamical systems and feed-forward architectures was recently also observed by several other authors (Haber et al., 2018; Weinan, 2017). This point of view leads to a large family of new network architectures that are induced by various numerical integration methods (Lu et al., 2018). Moreover, stability problems in both the forward as well the backward path of VDNNs have been addressed by relying on well-known analytical approaches for continuous-time ODEs (Chang et al., 2018; Haber and Ruthotto, 2017). In the present work, we instead address the problem directly in discrete-time, meaning that our stability result is preserved by the network implementation. Additionally, viewing deep neural networks as a discretization of a dynamical system, allows the training process of such a network to be cast as an optimal control problem in continuous time (Li et al., 2018b). However, with the exception of Liao and Poggio (2016), none of this prior research considers time-invariant, non-autonomous systems.

Conceptually, our work shares similarities with approaches that build network architectures according to iterative algorithms (Gregor and LeCun, 2010; Zheng et al., 2015) and recent ideas investigating pattern-dependent processing time (Figurnov et al., 2017; Graves, 2016; Veit and Belongie, 2018). See Section 3.5.2 and Section 3.6.3 for more details.

3.2.2 Related Work on Stability of Neural Networks

In machine learning, stability has long been central to the study of recurrent neural networks (RNNs) with respect to the vanishing (Bengio et al., 1994; Hochreiter, 1991; Pascanu et al., 2013), and exploding (Baldi et al., 1996; Doya, 1992; Pascanu et al., 2013) gradient problems, leading to the development of Long Short-Term Memory (Hochreiter and Schmidhuber, 1997b) to alleviate the former. It is well known that having well-behaved forward propagation is crucial for having gradients that do not explode nor

vanish during training (Ioffe and Szegedy, 2015; Pascanu et al., 2013), hence a well-posed learning problem. More recently, general conditions for RNN stability have been presented (Kanai et al., 2017; Laurent and Brecht, 2017; Vorontsov et al., 2017; Zilly et al., 2017) based on general insights related to Matrix Norm analysis. Stability of RNNs can also be achieved through more well-known approaches such as ℓ_1 or ℓ_2-norm weights regularization (Pascanu et al., 2013), or explicitly regularizing the evolution of the hidden state trajectory (Krueger and Memisevic, 2015). Classical Input-output stability (Khalil, 2014) has also been analyzed for simple RNNs (Haschke and Steil, 2005; Knight, 2008; Singh and Barabanov, 2016; Steil, 1999).

Recently, the stability of deep feed-forward networks was more closely investigated, mostly due to adversarial attacks (Szegedy et al., 2014) on trained networks. It turns out that sensitivity to (adversarial) input perturbations in the inference process can be avoided by ensuring certain conditions on the spectral norms of the weight matrices (Cisse et al., 2017; Yoshida and Miyato, 2017). Additionally, special properties of the spectral norm of weight matrices mitigate instabilities during the training of Generative Adversarial Networks (Miyato et al., 2018).

3.3 NON-AUTONOMOUS INPUT-OUTPUT STABLE NETS (NAIS-NETS)

This section provides stability conditions for both fully-connected and convolutional NAIS-Net layers. We formally prove that NAIS-Net provides a non-trivial input-dependent output for each iteration k as well as in the asymptotic case ($k \to \infty$). The following dynamical system:

$$\mathbf{x}(k+1) = \mathbf{x}(k) + hf\left(\mathbf{x}(k), \mathbf{u}, \theta\right), \ \mathbf{x}(0) = 0, \tag{3.4}$$

is used throughout the chapter, where $\mathbf{x} \in \mathbb{R}^n$ is the latent state, $\mathbf{u} \in \mathbb{R}^m$ is the network input, and $h > 0$. For ease of notation, in the remainder of the chapter the explicit dependence on the parameters, θ, will be omitted.

3.3.1 Fully-Connected NAIS-Net Layer

Our fully-connected layer is defined by

$$\mathbf{x}(k+1) = \mathbf{x}(k) + h\sigma\Big(\mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u} + \mathbf{b}\Big), \tag{3.5}$$

where $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $\mathbf{B} \in \mathbb{R}^{n\times m}$ are the state and input transfer matrices, and $\mathbf{b} \in \mathbb{R}^n$ is a bias. The activation $\sigma \in \mathbb{R}^n$ is a vector of (element-wise) instances of an activation function, denoted as σ_i with $i \in \{1, \ldots, n\}$. In this work, we only consider the hyperbolic tangent, *tanh*, and Rectified Linear

Units (ReLU) activation functions. Note that by setting $\mathbf{B} = 0$, and the step $h = 1$ the original ResNet formulation is obtained.

3.3.2 Convolutional NAIS-Net Layer

The architecture can be easily extended to Convolutional Networks by replacing the matrix multiplications in Eq. (3.5) with a convolution operator:

$$\mathbf{X}(k+1) = \mathbf{X}(k) + h\sigma\Big(\mathbf{C} * \mathbf{X}(k) + \mathbf{D} * \mathbf{U} + \mathbf{E}\Big). \tag{3.6}$$

Consider the case of N_C channels. The convolutional layer in Eq. (3.6) can be rewritten, for each latent map $c \in \{1, 2, \ldots, N_C\}$, in the equivalent form:

$$\mathbf{X}^c(k+1) = \mathbf{X}^c(k) + h\sigma\left(\sum_i^{N_C} \mathbf{C}_i^c * \mathbf{X}^i(k) + \sum_j^{N_C} \mathbf{D}_j^c * \mathbf{U}^j + \mathbf{E}^c\right), \tag{3.7}$$

where: $\mathbf{X}^i(k) \in \mathbb{R}^{n_X \times n_X}$ is the layer state matrix for channel i, $\mathbf{U}^j \in \mathbb{R}^{n_U \times n_U}$ is the layer input data matrix for channel j (where an appropriate zero padding has been applied) at layer k, $\mathbf{C}_i^c \in \mathbb{R}^{n_C \times n_C}$ is the state convolution filter from state channel i to state channel c, $\mathbf{D}_j^c$ is its equivalent for the input, and $\mathbf{E}^c$ is a bias. The activation, σ, is still applied element-wise. The convolution for $\mathbf{X}$ has a fixed stride $s = 1$, a filter size n_C and a zero padding of $p \in \mathbb{N}$, such that $n_C = 2p + 1$. [4]

Convolutional layers can be rewritten in the same form as fully connected layers (see proof of Lemma 1 in the supplementary material). Therefore, the stability results in the next section will be formulated for the fully-connected case, but apply to both.

3.3.3 Stability analysis

Here, the stability conditions for NAIS-Nets which were instrumental to their design are laid out. We are interested in using a cascade of unrolled NAIS blocks (see Figure 3.1), where each block is described by either Eq. (3.5) or Eq. (3.6). Since we are dealing with a cascade of dynamical systems, then stability of the entire network can be enforced by having stable blocks (Khalil, 2014). The state-transfer Jacobian for layer k is defined as:

$$\begin{aligned} \mathbf{J}(\mathbf{x}(k), \mathbf{u}) &= \frac{\partial \mathbf{x}(k+1)}{\partial \mathbf{x}(k)} \\ &= \mathbf{I} + h\frac{\partial \sigma(\Delta\mathbf{x}(k))}{\partial \Delta\mathbf{x}(k)}\mathbf{A}, \quad = \mathbf{I} + h\sigma'\left(\Delta\mathbf{x}(k)\right)\mathbf{A}. \end{aligned} \tag{3.8}$$

4 If $s \geq 0$, then $\mathbf{x}$ can be extended with an appropriate number of constant zeros (not connected).

where the argument of the activation function, σ, is denoted as $\Delta\mathbf{x}(k)$. Take an arbitrarily small scalar $\underline{\sigma} > 0$ and define the set of pairs $(\mathbf{x}, \mathbf{u})$ for which the activations are not saturated as:

$$\mathcal{P} = \left\{ (\mathbf{x}, \mathbf{u}) : \frac{\partial \sigma_i(\Delta\mathbf{x}(k))}{\partial \Delta\mathbf{x}_i(k)} \geq \underline{\sigma},\ \forall i \in [1, 2, \ldots, n] \right\}. \tag{3.9}$$

Theorem 1 below proves that the non-autonomuous residual network produces a bounded output given a bounded, possibly noisy, input, and that the network state converges to a constant value as the number of layers tends to infinity, if the following stability condition holds:

Condition 1. *For any $\underline{\sigma} > 0$, the Jacobian satisfies:*

$$\overline{\rho} = \sup_{(\mathbf{x},\mathbf{u})\in\mathcal{P}} \rho(\mathbf{J}(\mathbf{x}, \mathbf{u})),\ s.t.\ \overline{\rho} < 1, \tag{3.10}$$

where $\rho(\cdot)$ is the spectral radius.

The steady-states, $\overline{\mathbf{x}}$, are determined by a *continuous* function For tanh activation, the steady states, $\overline{\mathbf{x}}$, are determined by a *continuous* function of $\mathbf{u}$. This means that a small change in $\mathbf{u}$ cannot result in a very different $\overline{\mathbf{x}}$. For *tanh* activation, $\overline{\mathbf{x}}$ depends linearly on $\mathbf{u}$, therefore the block needs to be unrolled for a finite number of iterations, K, for the mapping to be non-linear. That is not the case for ReLU, which can be unrolled indefinitely and still provide a piece-wise affine mapping which is locally Lipschitz.

In Theorem 1, the Input-Output (IO) gain function, $\gamma(\cdot)$, describes the effect of norm-bounded input perturbations on the network trajectory. This gain provides insight as to the level of robust invariance of the classification regions to changes in the input data with respect to the training set. In particular, as the gain is decreased, the perturbed solution will be *closer* to the solution obtained from the training set. This can lead to increased robustness and generalization with respect to a network that does not statisfy Condition 1. Note that the IO gain, $\gamma(\cdot)$, is linear, and hence the block IO map is *Lipschitz* even for an *infinite* unroll length. The IO gain depends directly on the norm of the state transfer Jacobian, in Eq. (3.10), as indicated by the term $\overline{\rho}$ in Theorem 1.[5]

Theorem 1. *(Asymptotic stability for shared weights)*

If Condition 1 holds, then NAIS-Net with tanh activations is Asymptotically Stable with respect to input dependent equilibrium points. More formally:

$$\mathbf{x}(k) \to \overline{\mathbf{x}} \in \mathbb{R}^n,\ \forall \mathbf{x}(0) \in \mathcal{X} \subseteq \mathbb{R}^n,\ \mathbf{u} \in \mathbb{R}^m. \tag{3.11}$$

The trajectory is described by $\|\mathbf{x}(k) - \overline{\mathbf{x}}\| \leq \overline{\rho}^k \|\mathbf{x}(0) - \overline{\mathbf{x}}\|$, where $\|\cdot\|$ is a suitable matrix norm and $\overline{\rho}$ is defined in Eq. (3.10).
In particular, with tanh activation:

5 see supplementary material for additional details and all proofs, where the untied case is also covered.

1. *the steady-state $\bar{\mathbf{x}}$ is independent of the initial state, and it is a linear function of the input, namely, $\bar{\mathbf{x}} = \mathbf{A}^{-1}\mathbf{B}\mathbf{u}$. The network is Globally Asymptotically Stable with respect to $\bar{\mathbf{x}}$.*

2. *the network is Globally Input-Output (robustly) Stable for any additive input perturbation $w \in \mathbb{R}^m$ and the trajectory of the system is described by:*

$$\begin{aligned} \|\mathbf{x}(k) - \bar{\mathbf{x}}\| \leq \bar{\rho}^k \|\mathbf{x}(0) - \bar{\mathbf{x}}\| + \gamma(\|\mathbf{w}\|), \quad \forall k \geq 0 \\ \text{with } \gamma(\|\mathbf{w}\|) = h\frac{\|\mathbf{B}\|}{(1-\bar{\rho})}\|\mathbf{w}\|. \end{aligned} \tag{3.12}$$

   where $\gamma(\cdot)$ is the input-output gain. Moreover, if $\|\mathbf{w}\| \leq \mu$ for any $\mu \geq 0$, then the following set is robustly positively invariant ($\mathbf{x}(k) \in \mathcal{X}, \forall k \geq 0$):

$$\mathcal{X} = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x} - \bar{\mathbf{x}}\| \leq \gamma(\mu)\}. \tag{3.13}$$

while with ReLU activation:

- *The network is Globally incrementally practically Stable (δ-GpS). In other words, $\forall k \geq 0$, given two initial conditions $\{x(0), \bar{\mathbf{x}}(0)\}$ and the same input u, we have:*

$$\|\mathbf{x}(k) - \bar{\mathbf{x}}(k)\| \leq \bar{\rho}^k \|\mathbf{x}(0) - \bar{\mathbf{x}}(0)\| + \zeta. \tag{3.14}$$

  The constant factor is $\zeta = \frac{\|\mathbf{x}(0) - \|}{(1-\bar{\rho})}$.

- *The network is Globally Input-Output incrementally practically Stable (δ-IOS). In other words, given $\{\mathbf{x}(0), \bar{\mathbf{x}}(0)\}$ and two respective inputs $\{\mathbf{u}, \bar{\mathbf{u}}\}$ we have:*

$$\begin{aligned} \|\mathbf{x}(k) - \bar{\mathbf{x}}(k)\| \leq \bar{\rho}^k \|\mathbf{x}(0) - \bar{\mathbf{x}}(0)\| + \gamma(\|\mathbf{u} - \bar{\mathbf{u}}\|) + \zeta, \quad \forall k \geq 0 \\ \text{with } \gamma(\|\mathbf{w}\|) = h\frac{\|\mathbf{B}\|}{(1-\bar{\rho})}\|\mathbf{w}\|. \end{aligned} \tag{3.15}$$

  where $\gamma(\cdot)$ is the input-output gain as also defined in Eq. (3.12).

  Therefore, given $\mathbf{u} = \bar{\mathbf{u}} + \mathbf{w}$ with $\|\mathbf{w}\| \leq \mu$ for any $\mu \geq 0$, and $\bar{\mathbf{u}}$ being the nominal (or training) input, then the network state delta-converges to:

$$\|x - \bar{\mathbf{x}}\| \leq \frac{\|x(0) - \bar{\mathbf{x}}(0)\|}{(1-\bar{\rho})} + \gamma(\mu). \tag{3.16}$$

3.4 STABILITY CONSTRAINTS IMPLEMENTATION

In general, an optimization problem with a spectral radius constraint as in Eq. (3.10) is hard (Kanai et al., 2017). One possible approach is to relax the

Algorithm 1 Fully-Connected layer Reprojection

Input: $\mathbf{R} \in \mathbb{R}^{\tilde{n} \times n}$, $\tilde{n} \leq n$, $\delta = 1 - 2\epsilon$ and $\epsilon \in (0, 0.5)$.
if $\|\mathbf{R}^T\mathbf{R}\|_F > \delta$ **then**
  $\tilde{\mathbf{R}} \leftarrow \sqrt{\delta}\frac{\mathbf{R}}{\sqrt{\|\mathbf{R}^T\mathbf{R}\|_F}}$
else
  $\tilde{\mathbf{R}} \leftarrow \mathbf{R}$
end if
Output: $\tilde{\mathbf{R}}$

constraint to a singular value constraint (Kanai et al., 2017) which is applicable to both fully-connected as well as convolutional layer types (Yoshida and Miyato, 2017). However, this approach is only applicable if the identity matrix in the Jacobian (Eq. (3.8)) is scaled by a factor $0 < c < 1$ (Kanai et al., 2017). In this work we instead fulfil the spectral radius constraint directly.

The basic intuition for the presented algorithms is the fact that for a simple Jacobian of the form $\mathbf{I} + \mathbf{M}$, $\mathbf{M} \in \mathbb{R}^{n \times n}$, Condition 1 is fulfilled, if $\mathbf{M}$ has eigenvalues with real part in $(-2, 0)$ and imaginary part in the unit circle. In the supplemental material we prove that the following algorithms fulfill Condition 1 following this intuition. Note that, in the following, the presented procedures are to be performed for each block of the network.

3.4.1 Fully-connected blocks.

In the fully-connected case, we restrict the matrix $\mathbf{A}$ to by symmetric and negative definite by choosing the following parameterization for them:

$$\mathbf{A} = -\mathbf{R}^T\mathbf{R} - \epsilon\mathbf{I}, \tag{3.17}$$

where $\mathbf{R} \in \mathbb{R}^{n \times n}$ is trained, and $0 < \epsilon \ll 1$ is a hyper-parameter. Then, we propose a bound on the Frobenius norm, $\|R^T R\|_F$. Algorithm 1, performed during training, implements the following. [6]:

Theorem 2. *(Fully-connected weight projection)*
Given $R \in \mathbb{R}^{n \times n}$, *the projection* $\tilde{\mathbf{R}} = \sqrt{\delta}\frac{\mathbf{R}}{\sqrt{\|\mathbf{R}^T\mathbf{R}\|_F}}$, *with* $\delta = 1 - 2\epsilon \in (0, 1)$, *ensures that* $\mathbf{A} = -\tilde{\mathbf{R}}^T\tilde{\mathbf{R}} - \epsilon\mathbf{I}$ *is such that Condition 1 is satisfied for* $h \leq 1$ *and therefore Theorem 1 holds.*

Note that $\delta = 2(1 - \epsilon) \in (0, 2)$ is also sufficient for stability, however, the δ from Theorem 2 makes the trajectory free from oscillations (critically damped), see Figure 3.6. This is further discussed in Appendix.

6 The more relaxed condition $\delta \in (0, 2)$ is sufficient for Theorem 1 to hold locally (supplementary material).

Algorithm 2 Convolutional layer Reprojection

Input: $\delta \in \mathbb{R}^{N_C}, \mathbf{C} \in \mathbb{R}^{n_X \times n_X \times N_C \times N_C}$, and $0 < \epsilon < \eta < 1$.

for each feature map c **do**

$\tilde{\delta}_c \leftarrow \max\left(\min\left(\delta_c, 1-\eta\right), -1+\eta\right)$

$\tilde{\mathbf{C}}^c_{i_{\text{centre}}} \leftarrow -1 - \tilde{\delta}_c$

if $\sum_{j \neq i_{\text{centre}}} \left|\mathbf{C}^c_j\right| > 1 - \epsilon - |\tilde{\delta}_c|$ **then**

$\tilde{\mathbf{C}}^c_j \leftarrow \left(1 - \epsilon - |\tilde{\delta}_c|\right) \dfrac{\mathbf{C}^c_j}{\sum_{j \neq i_{\text{centre}}} \left|\mathbf{C}^c_j\right|}$

end if

end for

Output: $\tilde{\delta}$, $\tilde{\mathbf{C}}$

3.4.2 Convolutional blocks

The symmetric parametrization assumed in the fully-connected case can not be used for a convolutional layer. We will instead make use of the following result:

Lemma 1. *The convolutional layer Eq. (3.6) with zero-padding $p \in \mathbb{N}$, and filter size $n_C = 2p+1$ has a Jacobian of the form Eq. (3.8) with $\mathbf{A} \in \mathbb{R}^{n_X^2 N_C \times n_X^2 N_C}$. The diagonal elements of this matrix, namely, $\mathbf{A}_{n_X^2 c+j, n_X^2 c+j}$, $0 \leq c < N_C$, $0 \leq j < n_X^2$ are the central elements of the $(c+1)$-th convolutional filter mapping $\mathbf{X}^{c+1}(k)$, into $\mathbf{X}^{c+1}(k+1)$, denoted by $\mathbf{C}^c_{i_{centre}}$. The other elements in row $n_X^2 c + j$, $0 \leq c < N_C$, $0 \leq j < n_X^2$ are the remaining filter values mapping to $\mathbf{X}^{(c+1)}(k+1)$.*

To fulfill the stability condition, the first step is to set $\mathbf{C}^c_{i_{\text{centre}}} = -1 - \delta_c$, where δ_c is trainable parameter satisfying $|\delta_c| < 1 - \eta$, and $0 < \eta \ll 1$ is a hyper-parameter. Then we will suitably bound the ∞-norm of the Jacobian by constraining the remaining filter elements. The steps are summarized in Algorithm 2 which is inspired by the Gershgorin Theorem (Horn and Johnson, 2012). The following result is obtained:

Theorem 3. *(Convolutional weight projection)*
Algorithm 2 fulfils Condition 1 for the convolutional layer, for $h \leq 1$, hence Theorem 1 holds.

Note that the algorithm complexity scales with the number of filters. A simple design choice for the layer is to set $\delta = 0$, which results in $\mathbf{C}^c_{i_{\text{centre}}}$ being fixed at -1. [7]

7 Setting $\delta = 0$ removes the need for hyper-parameter η but does not necessarily reduce conservativeness as it will further constrain the remaining element of the filter bank. This is further discussed in the supplementary.

3.5 FORWARD PROPAGATION DYNAMICS

In this section, we compare the behaviour of a fully-connected NAIS-Net block and its unstable counterpart, *non-autonomous ResNet*, by analyzing the dynamics of the forward propagation of the system induced by different parameter values.

To ease the visualization of the systems' dynamics, we focus on the *scalar* case of the parametrization in Eq. (3.5), when the activation space is 1-dimensional and A, B, b are scalars that act on scalar-valued state x and input u — note that they are not bold in this section.

We investigate different values of A that satisfy or violate the stability conditions, namely $\|\mathbf{I} + \mathbf{A}\| < 1$, which in this special case translate into $A \in (-2, 0)$. Whenever this condition is violated, we denote the resulting network as *non-autonomous ResNet* or, equivalently, *unstable NAIS-Net*. For simplicity, we set the input parameter $B = 1$, and the bias $b = 0$ for all the experiments. Note that, in this setting, varying the input is equivalent to varying the bias.

3.5.1 Fixed number of unroll steps

We consider a fixed number of unroll steps and the last point trajectories with fixed length K. In this case, a NAIS-Net block produces an input-output map that is Lipschitz for any K (even infinity) and significantly better behaved than a standard ResNet, with no need of extra regularisations or batch normalization.

Non-Autonomous ResNet (Unstable NAIS-Net block)

Figure 3.2 shows examples of pathological cases in which the *unstable non-autonomous ResNet* produces unreliable or uninformative input-output maps. Plots in the top row analyze the case of *positive unstable eigenvalues*. The *tanh* network (top left) presents *bifurcations* which can make gradients explode (Pascanu et al., 2013). The slope is nearly infinite around the origin, and finally, for large inputs, the activation collapses into a flat function equal to kA, where k is the iteration number. The ReLU network (top right) has an exponential gain increase per step k, and the gain for $k = 100$ reaches 10^{30} (see red box and pointer).

Bottom plots analyze the case of *large negative eigenvalues*. The *tanh* activation (bottom left) produces a map with a limited slope that also presents irregularities, especially around the origin, that could be problematic during training. The ReLU activation (bottom right) instead produces an uninformative map, $\max(u, 0)$, which is (locally) independent from the parameter A and the unroll length K.

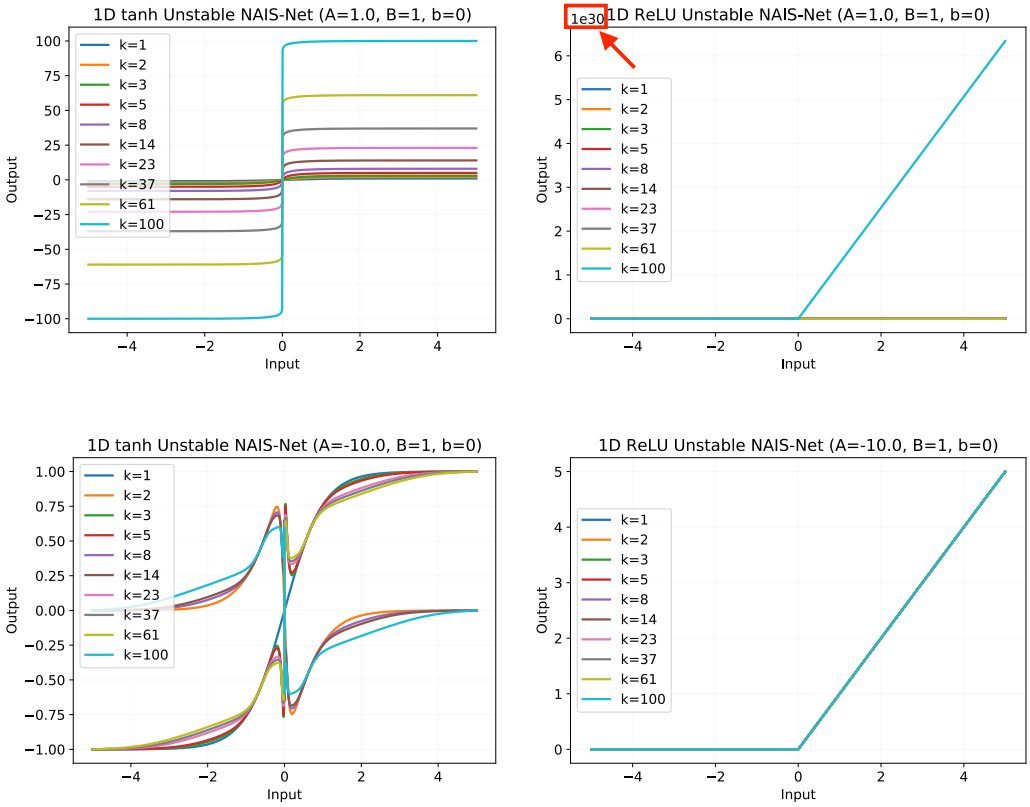


Figure 3.2: Single neuron Non-autonomous ResNet (unstable NAIS-Net block). Input-output map for different unroll length for *tanh* (Left) and ReLU (Right) activations. Pathological cases in which the unstable non-autonomous ResNet produces unreliable or uninformative input-output maps for positive unstable eigenvalues (top), and large negative eigenvalues (bottom).

NAIS-Net block

Figure 3.3 shows the input-output maps produced by stable NAIS-Net with the reprojection for fully-connected layers proposed in Algorithm 1. Top and bottom graphs present the case of positive real stable eigenvalues with different magnitude resulting in trajectories that are *critically damped* or oscillation-free.

Figures on the left confirm our theoretical result for *tanh* networks, first point of Theorem 1. As a result of the reprojection, the networks are stable and have monotonic activations (strictly increasing around the origin) that tend to a straight line as $k \to \infty$. Moreover, the map is Lipschitz, with Lipschitz constant equal to the steady-state gain presented in Theorem 1, $\|A^{-1}\| \, \|B\|$.

Figures on the right present the ReLU case where the maps remain of the same form, but they change in slope until the theoretical gain $\|A^{-1}\| \, \|B\|$ is reached. This means that one cannot have an unbounded slope or the same

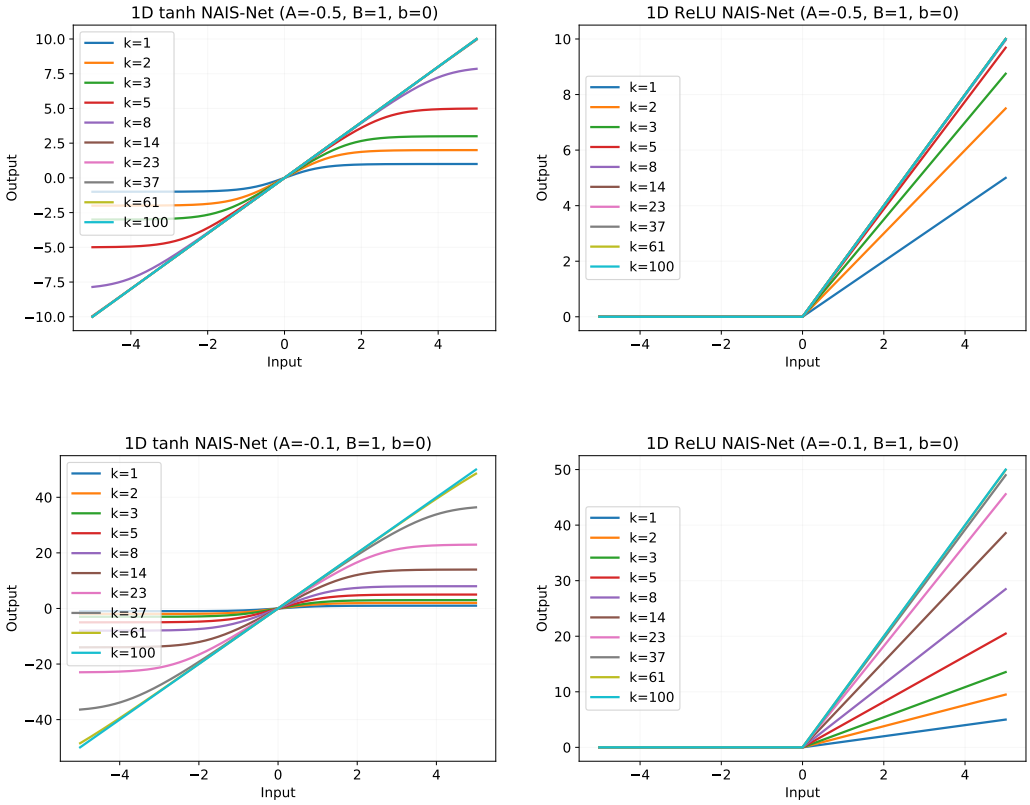


Figure 3.3: Single neuron NAIS-Net. Input-output map for different unroll length for *tanh* (Left) and ReLU (Right) activations. The input-output maps produced by stable NAIS-Net with the proposed reprojection for fully-connected architectures. Top and bottom graphs show the behaviour with positive stable eigenvalues for different magnitude.

map for different unroll length. The parameters also determine the rate of change of the map as a function of k that is always under control. Finally, one could make the Lipschitz constant unitary by multiplying the network output by $\|A\|/\|B\|$ once the recursion is terminated. This could be used as an alternative to batch normalization, and it could be investigated in future work.

Less Conservative NAIS-Net block

Figure 3.4 shows the maps produced by stable NAIS-Net with eigenvalues that are outside the region of our proposed reprojection for fully connected layers but still inside the one for convolutional layers. In particular, the top and bottom graphs present the case of *negative real stable eigenvalues* with different magnitude.

As a result, figures on the left show that *tanh* networks have monotonic activations (not strictly in this case) that still tend to a straight line as expected but present intermediate decaying oscillations as $k \to \infty$. The

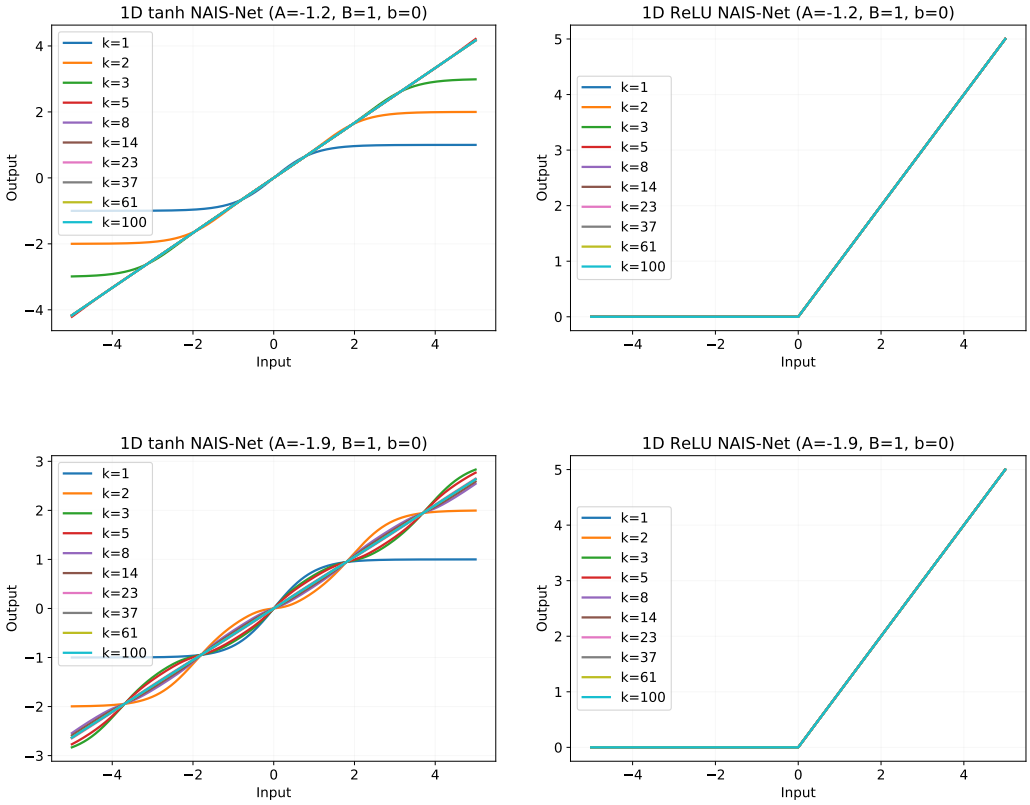


Figure 3.4: Single neuron NAIS-Net with less conservative reprojection. Input-output map for different unroll length for *tanh* (Left) and ReLU (Right) activations. Input-output maps produced by stable NAIS-Net with eigenvalues that are outside the region of our proposed reprojection for fully-connected layers but still inside the one for convolutional layers.

map is still Lipschitz, but the Lipschitz constant is not always equal to the steady-state gain because of the transient oscillations.

Figures on the right present the ReLU case where the maps remain identical independently of the parameter A, making the parameter uninformative. Note that the proposed reprojection for convolutional layers can theoretically allow for this behaviour causing training instabilities, while the fully connected version does not. However, experiments trained with Algorithm 2 have shown satisfactory results (see Section 3.6.2).

3.5.2 Pattern-dependent processing depth

We investigate the use at test time of a stopping criterion for the network unroll, $\|\Delta x(x,u)\| \leq \epsilon$, where ϵ is a hyper-parameter. This defines a *pattern-dependent processing depth* where the network results instead in a piecewise Lipschitz function for any $K > 0$ and $\epsilon > 0$. We consider again the case of

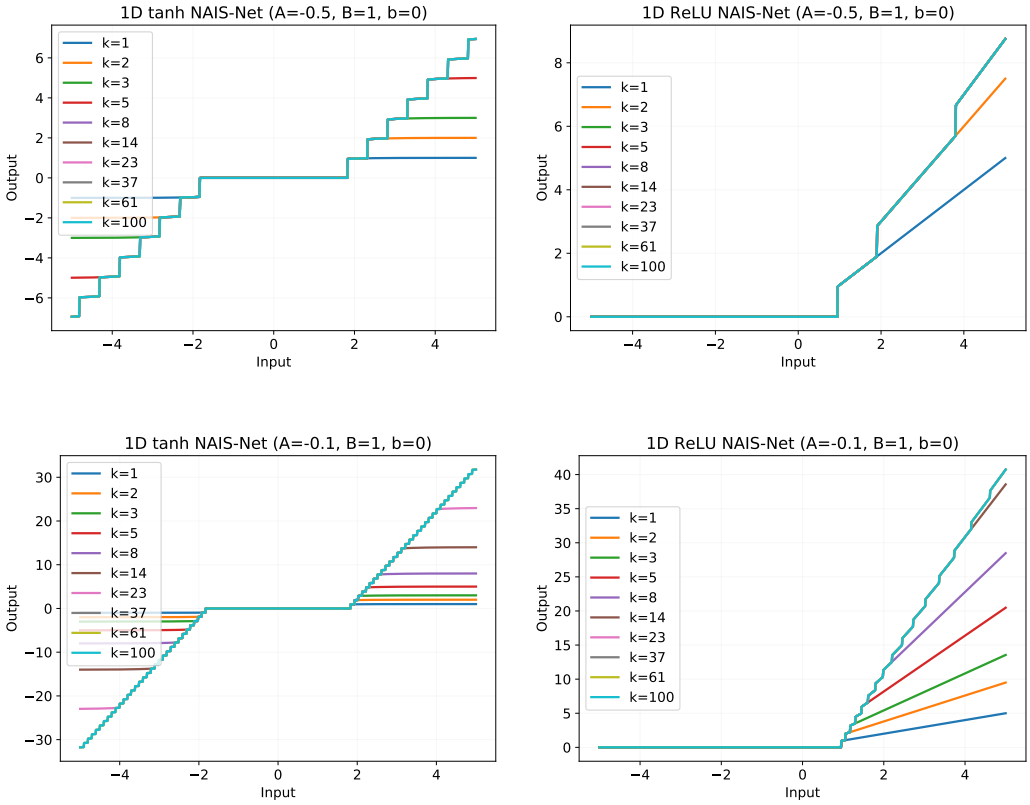


Figure 3.5: Single neuron NAIS-Net with stopping criteria for pattern-dependent processing depth. Input-output map for different unroll length for *tanh* (Left) and ReLU (Right) activations. The input-output maps produced by stable NAIS-Net with our proposed reprojection for fully-connected architectures. In particular, the top and bottom graphs present the case of positive stable eigenvalues with different magnitude.

a 1-dimensional system, where ϵ is set to 0.95 for illustrative purpose. The resulting activations are discontinuous, but locally preserve the properties illustrated in the previous section.

Figure 3.5 shows the input-output maps produced by stable NAIS-Net with our proposed reprojection for fully-connected architectures when the stopping criterion is adopted. In particular, the top and bottom graphs present the case of positive stable eigenvalues with different magnitude.

Figures on the left show that *tanh* networks are stable, with piece-wise continuous and locally strictly monotonic activations (strictly increasing around the origin) that tend to a straight line as $k \to \infty$. Moreover, the map is also piece-wise Lipschitz with Lipschitz constant less than the steady-state gain presented in Theorem 1, $\|A^{-1}\| \, \|B\|$.

Figures on the right present the ReLU case where the maps are piece-wise linear functions with slope that is upper bounded by $\|A^{-1}\| \, \|B\|$. This means that one cannot have an unbounded slope (except for the jumps) or the same

map for different unroll length. The rate of change of the map changes as a function of k is also determined by the parameters. The jump magnitude and the slopes are also dependant on the choice of threshold for the stopping criteria.

3.6 EXPERIMENTS ON IMAGE CLASSIFICATION

In this section, we present the results of the experiments that were conducted on image classification tasks comparing NAIS-Net with ResNet, and variants thereof, using both fully-connected (MNIST, Section 3.6.1) and convolutional (CIFAR-10/100, Section 3.6.2) architectures to quantitatively assess the performance advantage of having a VDNN where stability is enforced.

3.6.1 Preliminary analysis on MNIST

For the MNIST dataset (LeCun, 1998) a single-block NAIS-Net was compared with 9 different 30-layer ResNet variants each with a different combination of the following features: **SH** (shared weights i.e. time-invariant), **NA** (non-autonomous i.e. input skip connections), **BN** (with Batch Normalization), **Stable** (stability enforced by Algorithm 1). For example, ResNet-SH-NA-BN refers to a 30-layer ResNet that is time-invariant because weights are shared across all layers (SH), non-autonomous because it has skip connections from the input to all layers (NA), and uses batch normalization (BN). Since NAIS-Net is time-invariant, non-autonomous, and input/output stable (i.e. SH-NA-Stable), the chosen ResNet variants represent ablations of the these three features. For instance, ResNet-SH-NA is a NAIS-Net without I/O stability being enforced by the reprojection step described in Algorithm 1, and ResNet-NA, is a non-stable NAIS-Net that is time-variant, i.e non-shared-weights, etc. The NAIS-Net was unrolled for $K = 30$ iterations for all input patterns. All networks were trained using stochastic gradient descent with momentum 0.9 and learning rate 0.1, for 150 epochs.

Results

Test accuracy for NAIS-NET was 97.28%, while ResNet-SH-BN was second best with 96.69%, but without BatchNorm (ResNet-SH) it only achieved 95.86% (averaged over 10 runs).

After training, the behaviour of each network variant was analyzed by passing the activation, $\mathbf{x}(i)$, though the softmax classifier and measuring the cross-entropy loss. The loss at each iteration describes the trajectory of each sample in the latent space: the closer the sample to the correct steady-state the closer the loss to zero (see Figure 3.6). All variants initially refine their predictions at each iteration since the loss tends to decreases at

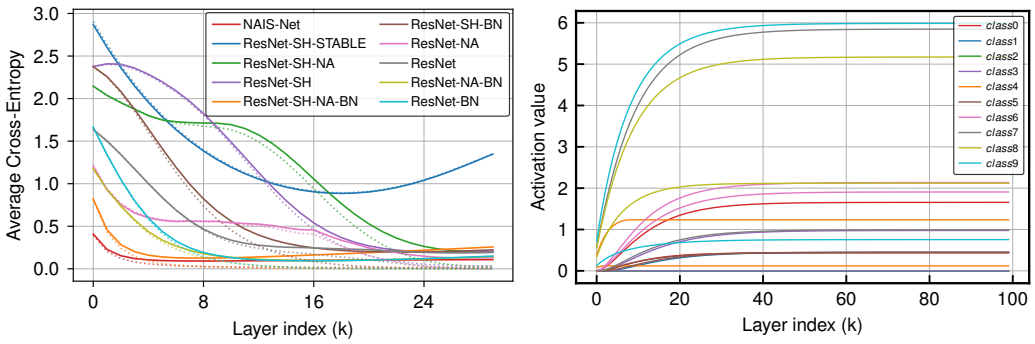


Figure 3.6: Single neuron trajectory and convergence. (Left) Average loss of NAIS-Net with different residual architectures over the unroll length. Note that both ResNet-SH-Stable and NAIS-Net satisfy the stability conditions for convergence, but only NAIS-Net is able to learn, showing the importance of non-autonomy. **Cross-entropy loss vs processing depth. (Right)** Activation of a NAIS-Net single neuron for input samples from each class on MNIST. Trajectories not only differ with respect to the actual steady-state but also with respect to the convergence time.

each layer, but at different rates. However, NAIS-Net is the only one that does so monotonically, not increasing loss as i approaches 30. Figure 3.6 shows how neuron activations in NAIS-Net converge to different steady-state activations for different input patterns instead of all converging to zero as is the case with ResNet-SH-Stable, confirming the results of (Haber and Ruthotto, 2017). Importantly, NAIS-Net is able to learn even with the stability constraint, showing that non-autonomy is key to obtaining representations that are stable *and* good for learning the task. NAIS-Net also allows training of unbounded processing depth without any feature normalization steps. Note that BN actually speeds up loss convergence, especially for ResNet-SH-NA-BN (i.e. unstable NAIS-Net). Adding BN makes the behaviour very similar to NAIS-Net because BN also implicitly normalizes the Jacobian, but it does not ensure that its eigenvalues are in the stability region.

3.6.2 Image Classification on CIFAR-10/100

Experiments on image classification were performed on standard image recognition benchmarks CIFAR-10 and CIFAR-100 (Krizhevsky and Hinton, 2009). These benchmarks are simple enough to allow for multiple runs to test for statistical significance, yet sufficiently complex to require convolutional layers.

	CIFAR-10		CIFAR-100	
	train	test	train	test
ResNet	99.86±0.03	91.72±0.38	97.42 ± 0.06	66.34 ± 0.82
NAIS-Net1	99.37±0.08	91.24±0.10	86.90 ± 1.47	65.00 ± 0.52
NAIS-Net10	99.50±0.02	91.25±0.46	86.91 ± 0.42	66.07 ± 0.24

(a)

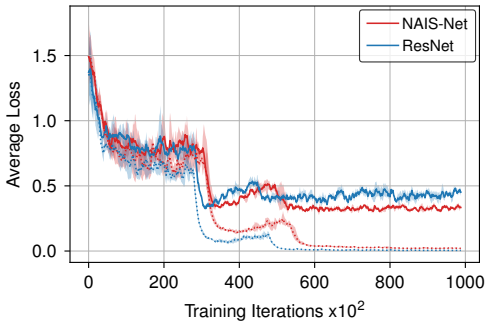


Figure 3.7: **CIFAR Results. (Left)** Classification accuracy on the CIFAR-10 and CIFAR-100 datasets averaged over 5 runs. **Generalization gap on CIFAR-10. (Right)** Dotted curves (training set) are very similar for the two networks but NAIS-Net has a considerably lower test curve (solid).

Setup

The following standard architecture was used to compare NAIS-Net with ResNet[8]: three sets of 18 residual blocks with 16, 32, and 64 filters, respectively, for a total of 54 stacked blocks. NAIS-Net was tested in two versions: NAIS-Net1 where each block is unrolled just once, for a total processing depth of 108, and NAIS-Net10 where each block is unrolled 10 times per block, for a total processing depth of 540. The initial learning rate of 0.1 was decreased by a factor of 10 at epochs 150, 250 and 350 and the experiment were run for 450 epochs. Note that each block in the ResNet of (He et al., 2015) has two convolutions (plus BatchNorm and ReLU) whereas NAIS-Net unrolls with a single convolution. Therefore, to make the comparison of the two architectures as fair as possible by using the same number of parameters, a single convolution was also used for ResNet.

Results

Table 3.7a compares the performance on the two datasets, averaged over 5 runs. For CIFAR-10, NAIS-Net and ResNet performed similarly, and unrolling NAIS-Net for more than one iteration had little affect. This was not the case for CIFAR-100 where NAIS-Net10 improves over NAIS-Net1 by 1%. Moreover, although mean accuracy is slightly lower than ResNet, the variance is considerably lower. Figure 3.7 shows that NAIS-Net is less prone to overfitting than a classic ResNet, reducing the generalization gap by 33%. This is a consequence of the stability constraint which imparts a degree of robust invariance to input perturbations (see Section 3.3). It is also

8 https://github.com/tensorflow/models/tree/master/official/resnet

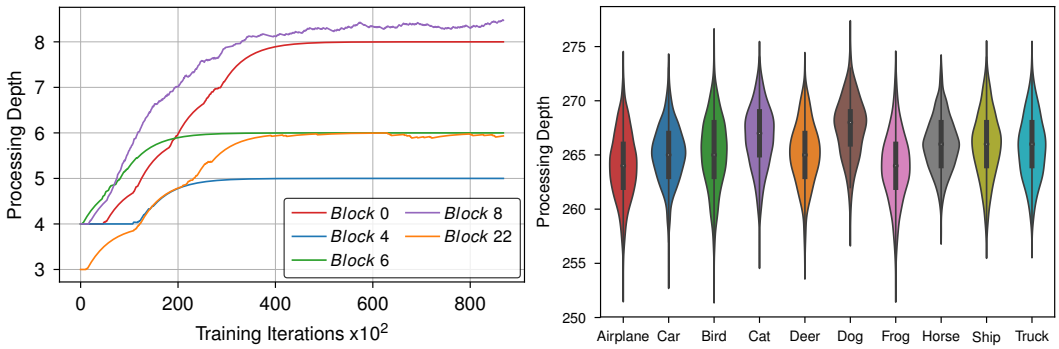


Figure 3.8: Processing depth for different NAIS-Net blocks during training (left). The figure shows the number of iterations (y-axis), of 5 different blocks taken NAIS-Net at various stages of training (x-axis) for the CIFAR-10 image classification task. **Final NAIS-Net depth distribution for CIFAR-10 classes (right).** Depth over the classes is statistically different. The Kruskal-Wallis one-way ANOVA test, under the null hypothesis of all depths coming from the same distribution, produced a p-value of zero. Pair-wise testing resulted in statistically different depths in 86.7% of cases, with a 95% confidence level.

important to note that NAIS-Net can unroll up to 540 layers, and still train without any problems.

3.6.3 Pattern-Dependent Processing Depth

For simplicity, the number of unrolling steps per block in the previous experiments was fixed. A more general and potentially more powerful setup is to have the processing depth adapt automatically. Since NAIS-Net blocks are guaranteed to converge to a pattern-dependent steady-state after an indeterminate number of iterations, processing depth can be controlled dynamically by terminating the unrolling process whenever the distance between a layer representation, $\mathbf{x}(i)$, and that of the immediately previous layer, $\mathbf{x}(i-1)$, drops below a specified threshold. With this mechanism, NAIS-Net can determine the processing depth for each input pattern. Intuitively, one could speculate that similar input patterns would require similar processing depth in order to be mapped to the same region in latent space. To explore this hypothesis, NAIS-Net was trained on CIFAR-10 with an unrolling threshold of $\epsilon = 10^{-4}$. At test time the network was unrolled using the same threshold. Figure 3.8 (left) shows the number of iterations required over the course of training to cross the depth threshold, ϵ, for five selected network blocks. As training progresses, depth increases almost monotonically at different rates per block, until becoming relatively stable. Figure 3.8 (right) shows a violin plot of the final network depth for each CIFAR-10 class. To highlight any statistical differences, the null hypothesis that the network depth being drawn

from the same distribution for all classes was tested first. One-way ANOVA test (Kruskal and Wallis, 1952) was performed over all the classes resulting in a p-value of zero, thus fully rejecting the null hypothesis. The same test was performed on each pair of classes: the 86.7% of depth distributions is statistically different with 95% confidence.

Figure 3.9 shows selected images from four different classes organized according to the final network depth used to classify them after training. The qualitative differences seen from low to high depth suggests that NAIS-Net is using processing depth as an additional degree of freedom so that, for a given training run, the network learns to use models of different complexity (depth) for different types of inputs within each class. To be clear, the hypothesis is not that depth correlates to some notion of input complexity where the same images are always classified at the same depth across runs.

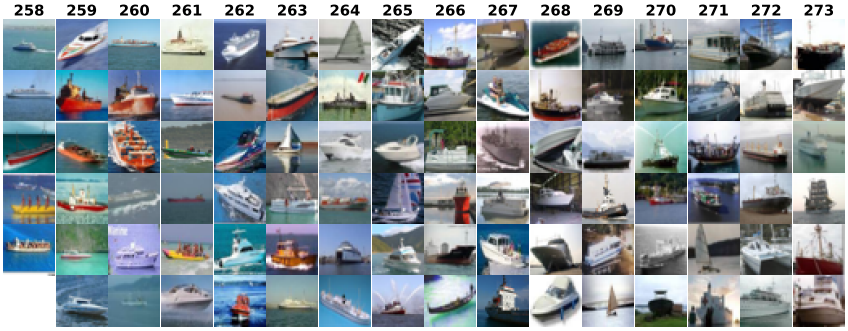
3.7 CONCLUSIONS

We presented NAIS-Net, a non-autonomous residual architecture that can be unrolled until the latent space representation converges to a stable input-dependent state. This is achieved thanks to stability and non-autonomy properties. We derived stability conditions for the model and proposed two efficient reprojection algorithms, both for fully-connected and convolutional layers, to enforce the network parameters to stay within the set of feasible solutions during training. NAIS-Net achieves asymptotic stability and, as consequence of that, input-output stability. Stability makes the model more robust and we observe a reduction of the generalization gap by quite some margin, without negatively impacting performance.

We believe that cross-breeding machine learning and control theory will open up many new interesting avenues for research, and that more robust and stable variants of commonly used neural networks, both feed-forward and recurrent, will be possible.

(a) frog

(b) bird



(c) ship

(d) airplane

Figure 3.9: CIFAR-10 image samples with corresponding NAIS-Net depth. Qualitative differences induce different final depths, indicating that NAIS-Net adapts processing systematically according characteristics of the data. For example, *"frog"* images with textured background are processed with fewer iterations than those with plain background. Similarly, *"ship"* and *"airplane"* images having a predominantly blue color are processed with lower depth than those that are grey/white, and *"bird"* images are grouped roughly according to bird size with larger species such as ostriches and turkeys being classified with greater processing depth.

4 LEARNING REPRESENTATIONS FOR ASYNCHRONOUS EVENT-BASED DATA

Dynamic Vision Sensors (DVSs), also called *Event Cameras*, are bio-inspired sensors that differ from conventional frame-based devices in the way they encode the scene. By capturing per-pixel brightness changes, event cameras output a stream of events that encode the time, location, and sign of the brightness change. Event cameras offer many advantages over traditional cameras, making them particularly suitable for challenging vision and robotics applications that require low-latency, high speed, and high dynamic range. However, to truly unlock their potential, novel machine learning methods need to be developed. For instance, sparse and asynchronous events need to be integrated into a frame or event-surface before applying well-established computer vision algorithms. This is usually attained by using ad-hoc heuristics.

In this chapter, we focus on learning incremental representations for asynchronous streams of data coming from event cameras. We propose a novel mechanism, *Matrix-LSTM*, to produce task-dependent event-surfaces through a grid of Long Short-Term Memory (LSTM) cells. We show that Matrix-LSTM can learn an integration process of sparse and asynchronous events, building a two-dimensional representation incrementally, regardless of the task and the objective function to optimize. Compared to existing reconstruction approaches, our learned event-surface shows good flexibility and expressiveness on optical flow estimation on the MVSEC benchmark Zhu et al., 2018a, and it improves the state-of-the-art of event-based object classification on the N-Cars dataset (Sironi et al., 2018).

4.1 INTRODUCTION

Event-based cameras, such as dynamic vision sensors (DVSs) (Berner et al., 2013; Lichtsteiner et al., 2008; Posch et al., 2014; Serrano Gotarredona and Linares Barranco, 2013), are bio-inspired devices that attempt to emulate the efficient data-driven communication mechanisms of the brain.

Unlike conventional frame-based Active Pixel Sensors (APS) (Fossum, 1997), which capture the scene at a predefined and constant frame-rate, event-cameras work in a fundamentally different way. These devices are *asynchronous* sensors which capture the brightness levels based on the scene dynamics. This is possible because they measure relative brightness changes,

called *"events"*, independently for every pixel, rather than measuring the absolute brightness of the entire scene based on a fixed clock. Thus, an event camera outputs a sequence of "events" at a variable rate, where each event represents the change of the log intensity of a specific pixel at a particular time instant.

In practice, each pixel actively monitors for a change of sufficient magnitude from the previous value of brightness. When the change exceeds a certain threshold, the *"smart"* pixel updates its value, and the camera sends the encoding of the event, which contains its x, y location, the time t, and a single bit of information about the polarity p of the change — 1 if brightness increases and 0 if it decreases.

4.1.1 Advantages of event cameras

The way event cameras monitor brightness changes results in many potential advantages over conventional acquisition devices.

- *High temporal resolution and low latency*: thanks to the combination of digital and analog hardware, events are detected and saved with a microsecond resolution allowing the cameras to capture very fast motions without suffering from motion blur, a typical downside of frame-based devices. Moreover, since each pixel works independently, and there is no global exposure time, event cameras have minimal latency from micro to milliseconds. As soon as the change is detected, it is transmitted.

- *High Dynamic Range (HDR)*: since the photoreceptors of the pixels operate in logarithmic scale and each pixel is independent and does not need to wait for a global shutter, event cameras have a very high dynamic range (> 120dB), surpassing by a lot the 60dB of high-quality frame-based cameras.

- *Low power*: by transmitting only brightness changes, event cameras remove redundant data and reduce power consumption. Most cameras use about 10mW, but some prototypes achieve 10μW or less.

All these characteristics are key features in challenging scenarios involving fast movements (e.g., drones or moving cars) and abrupt brightness changes (e.g., when exiting a dark tunnel in a car). They can naturally handle scenes with both very bright and very dark regions, and they also accurately describe the trajectory of moving objects, which enables fast response times, precise motion estimation, and better occlusions handling.

4.1.2 Challenges with event-based data

Event cameras pose a paradigm shift in the way visual information is acquired, requiring to specifically design novel algorithms and hardware architectures to exploit their advantages and leverage their potential in complex tasks. As an example, event-cameras only provide a timed sequence of changes that is not directly compatible with computer vision systems which typically work on frames. In particular, the stream of data produced by event cameras is *asynchronous* in time and *sparse* in space, whereas images are *synchronous* and *dense*.

Since every single event is asynchronous, and it only carries a binary information (the occurrence of a change in a specific position and time instant), mechanisms able to effectively encode the precise timing information of events are key components of effective event-based vision systems, especially in data-driven pipelines.

Driven by the great success of frame-based deep learning architectures, that learn representations directly from standard APS signals, research in event-based processing is now focusing on how to effectively aggregate event information in grid-based representations which can be directly used, for instance, by convolutional deep learning models. Nevertheless, finding the best mechanism to extract information from event streams is not trivial, being representations usually task-specific.

Multiple solutions have indeed emerged during the past few years, mostly employing hand-crafted mechanisms to accumulate events. Examples of such representations are mechanisms relying on exponential (Cohen, 2016; Lagorce et al., 2016; Sironi et al., 2018) and linear (Cannici et al., 2019a; Cohen, 2016) decays, "event-surfaces" storing the timestamp of the last received event in each pixel and extensions of such mechanism making use of memory cells (Sironi et al., 2018) and voxel-grids (Rebecq et al., 2019; Zhu et al., 2018b). Only very recently deep learning techniques have been applied to learn such surfaces in a data-driven manner (Gehrig et al., 2019a).

4.1.3 Main contribution

In this chapter, we focus on the problem of learning representations for computer vision tasks involving event-based data, and propose a mechanism to efficiently aggregate events. We apply a Long Short-Term Memory (LSTM) network (Hochreiter and Schmidhuber, 1997b) as a convolutional filter over the 2D stream of events to accumulate pixel information through time and build 2D event representations. The reconstruction mechanism is end-to-end differentiable, meaning that it can be jointly trained with state-of-the-art frame-based architectures to learn event-surfaces specifically tailored for the task at hand. Most importantly, the mechanism specifically focuses on preserving sparsity during computation without densifying the

events during the intermediate feature extraction phase that is otherwise necessary when applying standard computer vision approaches such as ConvLSTM (Shi et al., 2015).

We show that replacing hand-crafted event-surfaces with our trainable layer in state-of-the-art architectures improves their performance substantially without requiring particular effort in hyper-parameter tuning, enabling researchers to exploit event information effectively. The layer is a powerful drop-in replacement for hand-crafted features, enabling researchers to exploit event information effectively and improve the performance of existing architectures with minor modifications. The contributions of this chapter are summarized as follows:

- We propose Matrix-LSTM, a task-independent mechanism to extract grid-like event representations from asynchronous streams of events. The framework is end-to-end differentiable, it can be used as input of any existing frame-based state-of-the-art architecture and jointly trained to extract the best representation from the events.
- Replacing input representations with a Matrix-LSTM layer in existing architectures, we show that it improves the state-of-the-art on event-based object classification on N-CARS (Sironi et al., 2018) by 3.3% and performs better than hand-crafted features on N-Caltech101 (Orchard et al., 2015a). Finally, it improves optical flow estimation on the MVSEC benchmark (Zhu et al., 2018a) up to 30.76% over hand-crafted features (Zhu et al., 2018a) and up to 23.07% over end-to-end differentiable ones (Gehrig et al., 2019a).
- We developed custom CUDA kernels, both in PyTorch (Steiner et al., 2019) and TensorFlow (Abadi et al., 2016), to efficiently aggregate events by position and perform a convolution-like operation on the stream of events using an LSTM as a convolutional filter [1].

4.2 EVENT REPRESENTATIONS

One of the key features underlying modern computer vision algorithms is their ability to aggregate elementary visual information to build complex visual features. To this end, convolutional neural networks (CNNs) are by far the most widespread method in frame-based architectures for image classification (He et al., 2016; Krizhevsky et al., 2012; Szegedy et al., 2016), object detection (He et al., 2017; Liu et al., 2016; Redmon et al., 2016), semantic segmentation (Chen et al., 2017; Long et al., 2015; Yu and Koltun, 2016), and many others. Their great success resides on their end-to-end

1 Code available at `https://marcocannici.github.io/matrixlstm`

differentiable structure that allows these architectures to learn powerful visual representations for the task to be solved.

Event cameras provide outstanding advantages over ordinary devices in terms of time resolution and dynamic range. However, their potential is still unlocked, mainly due to the difficulties of building good representations from a sparser, asynchronous and much more rough source of information compared to frame-based data. In this section, we give a brief overview of related works, focusing on representations for event-based data and highlighting the differences and similarities with our work. We refer the reader to (Gallego et al., 2020) for a thorough overview.

SPIKE-BASED REPRESENTATIONS Initially, neural systems designed to perform spike-based computation, such as Spiking Neural Networks (SNNs) (Maass, 1997), have been applied to event-based processing in several tasks, e.g., edge detection (Meftah et al., 2010; Wu et al., 2007), object classification (Diehl et al., 2015; Lee et al., 2016) and hand-gestures recognition (Botzheim et al., 2012). However, their non-differentiability and the lack of a well established supervised training procedure have limited the spreading of SNNs based architectures. Recent works (Rueckauer et al., 2017) tried to overcome these limitations by first training a frame-based conventional neural network, and then convert its weights into SNNs parameters, managing to deal with complex structures such as GoogLeNet Inception-V3 (Szegedy et al., 2016). Inspired by the event integration mechanism in SNNs, simple but effective, non-parametric leaky integration procedures have been proposed in recent work (Cannici et al., 2019a; b; Cohen, 2016), where the reconstructed frame is increased of a fixed quantity whenever an event occurs in the position corresponding to the event address and linearly decayed otherwise.

HAND-CRAFTED REPRESENTATIONS Several hand-crafted event representations have been proposed over the years, ranging from biologically inspired, such as those used in Spiking Neural Networks (Maass, 1997), to more structured ones. Recently, the concept of *time-surface* was introduced (Lagorce et al., 2016; Maqueda et al., 2018), in which 2D surfaces are obtained by keeping track of the timestamp of the last event occurred in each location and by associating each event with features computed applying exponential kernels on the surface. An extension of these methods, called HATS (Sironi et al., 2018), employs memory cells that retain temporal information from past events. Instead of building the surface using just the last event, too sensitive to noise, HATS uses a fixed-length memory. Histograms are then extracted from the surface and a SVM classifier is finally used for prediction. The use of a memory to compute the event-surface closely relates HATS with the solution presented in this chapter. Crucially, the accumulation procedure employed in HATS is hand-crafted, while our

work is end-to-end trainable thanks to a grid of LSTM cells (Hochreiter and Schmidhuber, 1997b), which enable to learn a better accumulation strategy directly from data.

Several hand-crafted event representations have been proposed over the years, ranging from biologically inspired, such as those used in Spiking Neural Networks (Maass, 1997), to more structured ones. The concept of *time-surface* was recently introduced in Lagorce et al. (2016) and Maqueda et al. (2018), in which 2D surfaces are obtained by keeping track of the *timestamp* of the last event occurred in each location and by associating each event with features computed applying exponential kernels on the surface. An extension of these methods, called HATS (Sironi et al., 2018), employs memory cells that retain temporal information from past events. Instead of building the surface using just the last event, too sensitive to noise, HATS uses a fixed-length memory. Histograms are then extracted from the surface, and an SVM classifier is finally used for prediction. The use of memory to compute the event-surface closely relates HATS with the solution presented in this chapter. Crucially, the accumulation procedure employed in HATS is hand-crafted, while our work is end-to-end trainable thanks to a grid of LSTM cells (Hochreiter and Schmidhuber, 1997b), which enable to learn a better accumulation strategy directly from data. This allows the memory to potentially handle an infinite number of events, but more importantly to learn a better accumulation strategy directly from data.

In (Zhu et al., 2018b), the authors propose the EV-FlowNet network for optical flow estimation together with a new time-surface variant. Events of different polarities are kept separate to build a four-channel grid containing the number of events that occurred in each location besides temporal information. A similar representation has also been used in (Ye et al., 2018). To improve the temporal resolution of such representations, Zhu et al. (2019a) suggests discretizing time into consecutive bins and accumulating events into a voxel-grid through a linearly weighted accumulation similar to bilinear interpolation. A similar time discretization has also been used in Events-to-Video (Rebecq et al., 2019), where the event representation is used within a recurrent-convolutional architecture to produce realistic video reconstructions of event sequences. Despite being slower, the quality of reconstructed frames closely resembles actual gray-scale frames, allowing the method to take full advantage of transferring feature representations trained on natural images.

END-TO-END REPRESENTATIONS Most closely related to the current work, Gehrig et al. (2019a) learns a dense representation end-to-end directly from raw events. A multi-layer perceptron (MLP) is used to implement a trilinear filter that produces a voxel-grid of temporal features. The event time information of each event is encoded using the MLP network. The values obtained from events occurring in the same spatial location are summed

up to build the final feature. A look-up table is then used, after training, to speed-up the procedure. Events are processed independently as elements of a set, disregarding their sequentiality and preventing the network from modulating the information based on previous events. Instead, by leveraging the memory mechanism of LSTM cells, our method can integrate information conditioned on the current state and decide how much each event is relevant to perform the task and how much information to retain from past events.

A recent trend in event-based processing is studying mechanisms that do not require to construct intermediate explicit dense representations to perform the task at hand (Bi et al., 2019; Sekikawa et al., 2019; Wang et al., 2019). Among these, Neil et al. (2016) uses a variant of the LSTM network, called PhasedLSTM, to learn the precise timings of events. While it integrates the events sequentially as in our work, PhasedLSTM employs a single cell on the entire stream of events and can be used only on very simple tasks (Cannici et al., 2019b). Indeed, the model does not maintain the spatial input structure and condenses the 2D stream of events into a single feature vector, preventing the network from being used as input to standard CNNs. In contrast, in this chapter, we use the LSTM as a convolutional filter obtaining a translation-equivariant module that integrates local temporal features independently while retaining spatial structures. Finally, although it has never been adopted with event-based cameras, we also mention here the ConvLSTM (Shi et al., 2015) network, a convolutional variant of the LSTM that has previously been applied on several end-to-end prediction tasks. Despite its similarity with our method, since both implement the notion of convolution to LSTM cells, ConvLSTM is not straightforward to apply to sparse event-based streams and requires the input to be densified into frames before processing. This involves building very sparse frames of simultaneous events, mostly filled with padding, or dense frames containing uncorrelated events. Our formulation, instead, preserves sparsity during computation and does not require events to be densified, even when large receptive fields are considered.

4.3 METHOD

Event-based cameras are vision sensors composed of pixels able to work independently. Each pixel has its own exposure time and it is free to fire independently by producing an event as soon as it detects a significant change in brightness. Unlike conventional devices, no rolling shutter is used, instead, an asynchronous stream of events is generated describing what has changed in the scene. Each event e_i is a tuple $e_i = (x_i, y_i, t_i, p_i)$ specifying the time t_i, the location $(x, y)_i$ (within a $H \times W$ space) and the polarity $p_i \in \{-1, 1\}$ of the change (brightness increase or decrease). Therefore, given a time interval τ (i.e., the sample length), the set of events produced

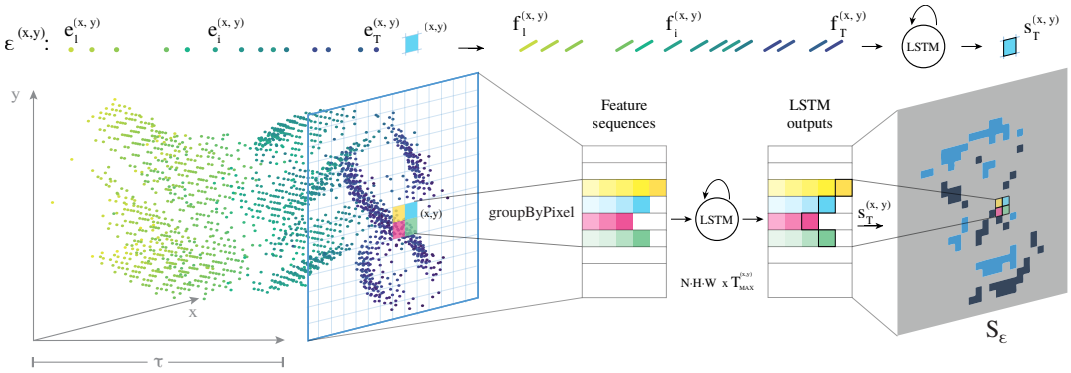


Figure 4.1: Overview of Matrix-LSTM (figure adapted from (Neil et al., 2016)). Events in each pixel are first associated to a set of features $f_i^{(x,y)}$, and then processed by the LSTM. The last output, $s_T^{(x,y)}$, is finally used to construct $\mathcal{S}_\mathcal{E}$. *GroupByPixel* is shown here on a single sample ($N = 1$) highlighting a 2×2 pixel region. Colors refer to pixel locations while intensity indicates time. For clarity, the features dimension is not shown in the figure

by the camera can be described as a sequence $\mathcal{E} = \{(x_i, y_i, t_i, p_i) \mid t_i \in \tau\}$, ordered by the event timestamp. In principle, multiple events could be generated at the same timestamp. However, the grid representation of the events at a fixed timestamp t is likely to be very sparse, hence, an integrating procedure is necessary to reconstruct a dense representation $\mathcal{S}_\mathcal{E}$ before being processed by conventional frame-based algorithms.

Note that, in this work, we do not aim to reconstruct a frame that resembles the actual scene, such as a grey-scale or RGB image (Rebecq et al., 2019; Scheerlinck et al., 2019), but instead to extract task-aware features regardless of their appearance. In the following, *"surface"*, *"reconstruction"* and *"representation"* are used with this meaning.

4.3.1 Matrix-LSTM

Analogously to Gehrig et al. (2019a), our goal is to learn end-to-end a fully parametric mapping $\mathcal{M} : \mathcal{E} \to \mathcal{S}_\mathcal{E} \in \mathbb{R}^{H \times W \times C}$, between the event sequence and the corresponding dense representation, providing the best features for the task to optimize.

In this work, we propose to implement $\mathcal{M}$ as an $H \times W$ *matrix of LSTM cells* (Hochreiter and Schmidhuber, 1997b) (see Figure 4.1). Let us define the ordered sequence of events $\mathcal{E}^{(x,y)}$ produced by the pixel (x, y) during interval τ as:

$$\mathcal{E}^{(x,y)} = \{(x_i, y_i, t_i, p_i) \mid t_i \in \tau, x_i = x, y_i = y\} \subset \mathcal{E} \tag{4.1}$$

and its length as $T^{(x,y)} = |\mathcal{E}^{(x,y)}|$, which may potentially be different for each location (x, y). A set of raw features $f_i^{(x,y)} \in \mathbb{R}^F$ is first computed for each

event occurring at location (x, y), typically the polarity and one or multiple temporal features (more details on the types of features can be found in Section 4.5.1). At each location (x, y), an $LSTM^{(x,y)}$ cell then processes these features asynchronously, keeping track of the current integration state and condensing all events into a single output vector $s^{(x,y)} \in \mathbb{R}^C$.

In particular, at each time t, the $LSTM^{(x,y)}$ cell produces an intermediate representation $s_t^{(x,y)}$. Once all the events are processed, the last output of the LSTM cell compresses the dynamics of the entire sequence $\mathcal{E}^{(x,y)}$ into a fixed-length vector $s_T^{(x,y)}$ that can be used as pixel feature (here we dropped the superscript $^{(x,y)}$ from T for readability). The final surface $\mathcal{S}_\mathcal{E}$ is finally built by collecting all LSTMs final outputs $s_T^{(x,y)}$ into a dense tensor of shape $H \times W \times C$. Since event-camera pixels produce an output only if a change is detected, a fixed all-zeros output is used where the set of events $\mathcal{E}^{(x,y)}$ is empty.

TEMPORAL BINS Taking inspiration from previous methods (Gehrig et al., 2019a; Rebecq et al., 2019; Zhu et al., 2019a) that discretize time into temporal bins, we also propose a variant of Matrix-LSTM that operates on successive time windows. Given a fixed number of bins B, the original event sequence is split into B consecutive windows $\mathcal{E}_{\tau_1}, \mathcal{E}_{\tau_2}, ..., \mathcal{E}_{\tau_B}$. Each sequence is processed independently, i.e., the output of each LSTM at the end of each interval is used to construct a surface $\mathcal{S}_{\mathcal{E}_b}$ and the LSTMs state is re-initialized before the next sub-sequence starts. This gives rise of B different reconstructions $\mathcal{S}_{\mathcal{E}_b}$ that are concatenated to form the final surface $\mathcal{S}_\mathcal{E} \in \mathbb{R}^{H \times W \times B \cdot C}$. In this formulation, the LSTM input features $f_i^{(x,y)}$ usually contain both global temporal features (i.e., with respect to the original uncut sequence) and relative features (i.e., the event position in the sub-sequence). Although LSTMs should be able to retain memory over very long periods, we found that discretizing time into intervals helps the Matrix-LSTM layer in maintaining event information, especially in tasks requiring precise time information such as optical flow estimation (see Section 4.5.2). A self-attention module (Hu et al., 2018a) is then optionally applied on the reconstructed surface to correlate intervals.

PARAMETERS SHARING Inspired by the convolution operation defined on images, we designed Matrix-LSTM to enjoy translation equivariance. This is implemented by sharing the parameters across all the LSTM cells, as in a convolutional kernel. Events in each location are processed sequentially using the LSTM memory to accumulate values and perform conditioned integration. Taking advantage of the LSTM gating mechanism, the network can indeed learn the best integration strategy given the current state, i.e., deciding each time how to encode the current event based on all the previous history (e.g., using the timing information to dynamically adapt to different

event rates). Sharing parameters not only drastically reduces the number of parameters in the network, but it also allows us to transfer a learned transformation to higher or lower resolutions as in fully-convolutional networks (Long et al., 2015).

We highlight that such an interpretation of the Matrix-LSTM functioning also fits the framework proposed in Gehrig et al. (2019a), in which popular event densification mechanisms are rephrased as kernel convolutions on the *event field*, i.e., a discretized four-dimensional manifold spanning x and y, and the time and polarity dimensions.

We finally report that this formulation is equivalent to a 1×1 ConvLSTM (Shi et al., 2015) applied *on a dense tensor* where events are stacked in pixel locations by arrival order. However, we show that our formulation is much more efficient in terms of space and time performance on sparse event sequences (see Section 4.6). Moreover, in the next section, an extension to larger receptive fields with better accuracy performance on asynchronous event data compared to ConvLSTM, is also proposed.

RECEPTIVE FIELD SIZE As in a conventional convolution operation, Matrix-LSTM can be convolved on the input space using different strides and kernel dimensions. In particular, given a receptive field of size $K_H \times K_W$, each LSTM cell processes a local neighborhood of asynchronous events:

$$\mathcal{E}^{(x,y)} = \{(x_i, y_i, t_i, p_i) \mid t_i \in \tau, |x - x_i| < K_W - 1, |y - y_i| < K_H - 1\} \quad (4.2)$$

Events features are computed as in the original formulation, however, an additional coordinate feature (p_x, p_y) specifies the relative position of each event within the receptive field. Coordinate features are range-normalized in such a way that an event occurring in the top-left pixel of the receptive field has feature $(0, 0)$, whereas one occurring in the bottom-right position has features $(1, 1)$. Events belonging to multiple receptive fields (e.g., when the LSTM is convolved with a stride 1×1 and receptive field greater then 1×1) are processed multiple times, independently.

4.4 IMPLEMENTATION

The convolution-like operation described in the previous section can be efficiently implemented by means of two carefully designed event grouping operations. Rather than replicating the LSTM unit multiple times on each spatial location, a single recurrent unit is applied over different $\mathcal{E}^{(x,y)}$ sequences in parallel. For each location, the LSTM cell has to process an ordered sequence of events, from the one having the smallest t_i to the last in the sequence. This requires a reshape operation, i.e., *groupByPixel*, that splits events based on their pixel location maintaining the events relative ordering within each sub-sequence. A similar procedure, i.e., *groupByTime*,

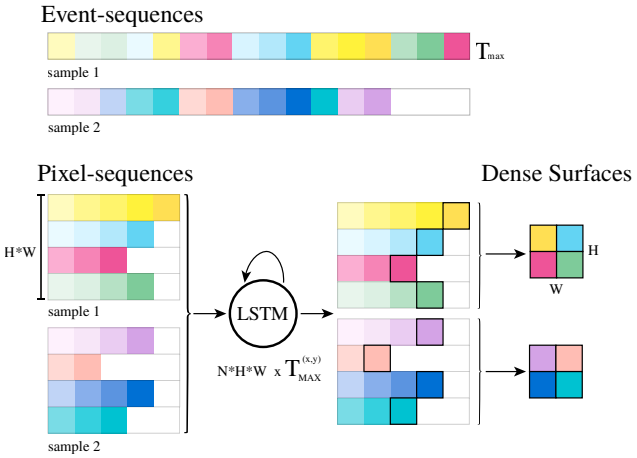


Figure 4.2: An example of the *groupByPixel* operation on a batch of $N = 2$ samples and a 2×2 pixel resolution. Different colors refer to different pixel locations while intensity indicates time. For clarity, the features dimension is not shown in the figure.

is employed to efficiently split events into consecutive temporal windows without making use of expensive masking operations. In the following we give a detailed overview of the two reshape operators. Note that these operations are not specific to Matrix-LSTM, since grouping events by pixel index is a common operation in event-based processing, and could indeed benefit other implementations making use of GPUs.

4.4.1 GroupByPixel

This operation translates from event-sequences to pixel-sequences. Let X be a tensor of shape $N \times T_{max} \times F$, representing the features $f_{n,i}^{(x,y)}$ of a batch of N samples, where T_{max} is the length of the longest sequence in the batch. We define the *groupByPixel* mapping on X as an order-aware reshape operation that rearranges the events into a tensor of pixel-sequences of shape $P \times T_{max}^{(x,y)} \times F$ where $T_{max}^{(x,y)}$ is the length of the longest pixel sequence $\mathcal{E}_n^{(x,y)}$ and P is the number of active pixels (i.e., having at least one event) in the batch, which equals $N \cdot H \cdot W$ only in the worst case. Pixel-sequences shorter than $T_{max}^{(x,y)}$ are padded with zero events to be processed in parallel.

The tensor thus obtained is then processed by the LSTM cell that treats samples in the first dimension independently, effectively implementing parameter sharing and applying the transformation in parallel over all the pixels. The LSTM output tensor, which has the same shape of the input one, is then sampled by taking the output corresponding to the last event in each pixel-sequence $\mathcal{E}_n^{(x,y)}$, ignoring values computed on padded values, and the obtained values are then used to populate the dense representation. To improve efficiency, for each pixel-sequence $\mathcal{E}_n^{(x,y)}$, *groupByPixel* keeps also

track of the original spatial position (x, y), the index of the sample inside the batch and the length of the pixel-sequence $T_n^{(x,y)}$, namely the index of the last event before padding. Given this set of indexes, the densification step can be performed as a simple slicing operation. See Figure 4.2 for visual clues. *groupByPixel* is implemented as a custom CUDA kernel that processes each sample in parallel and places each event feature in the output tensor maintaining the original temporal order.

4.4.2 GroupByTime

The Matrix-LSTM variant that operates on temporal bins performs a similar pre-processing step. Each sample in the batch is divided into a fixed set of intervals. The *groupByTime* cuda kernel pre-processes the input events generating a $N * B \times T_{max}^b \times F$ tensor where the B bins are grouped in the first dimension and taking care of properly padding intervals (T_{max}^b is the length of the longest bin in the batch). The Matrix-LSTM mechanism is then applied as usual and the resulting $N * B \times H \times W \times C$ tensor is finally reshaped into a $N \times H \times W \times B * C$ event-surface.

4.5 EVALUATION

We test the proposed mechanism on two different tasks: object classification (see Section 4.5.1) and optical flow estimation (see Section 4.5.2), where the network is required to extract useful temporal features. We evaluated the goodness of Matrix-LSTM features indirectly by taking a state-of-the-art architecture as a reference and assess the proposed method in terms of the relative gain in performance obtained by replacing the network representation with a Matrix-LSTM.

4.5.1 Object classification

We evaluated the model on the classification task using two publicly available event-based collections, namely the N-Cars (Sironi et al., 2018) and the N-Caltech101 (Orchard et al., 2015a) datasets, which represent to date the most complex benchmarks for event-based classification. N-Cars is a collection of urban scene recordings (lasting 100ms each) captured with a DVS sensor and showing two object categories: cars and urban background. The dataset is already split into 7,940 car and 7,482 background training samples, and 4,396 car and 4,211 background testing samples. The N-Caltech101 collection is an event-based conversion of the popular Caltech-101 (Li Fei-Fei et al., 2006) dataset obtained by moving an event-based camera in front of a still monitor showing one of the original RGB images. Like the original

Table 4.1: Results on N-Cars: **(a)** ResNet18-Ev2Vid, variable time encoding, and normalization; **(b)** ResNet18-EST, variable time encoding and number of bins

ResNet Norm	ts absolute	ts relative	delay relative
✓	$95.22 \pm 0.41\%$	$94.77 \pm 1.01\%$	$95.40 \pm 0.59\%$
	$\mathbf{95.75 \pm 0.27}\%$	$95.32 \pm 0.85\%$	$\mathbf{95.80 \pm 0.53}\%$

(a)

		1 bin	2 bins	9 bins
delay	glob+loc	-	$92.68 \pm 1.23\%$	$92.32 \pm 1.02\%$
	local	$92.64 \pm 1.21\%$	$92.35 \pm 0.83\%$	$92.67 \pm 0.90\%$
ts	ts glob+loc	-	$\mathbf{93.46 \pm 0.84}\%$	$\mathbf{93.21 \pm 0.49}\%$
	local	$92.65 \pm 0.78\%$	$92.75 \pm 1.38\%$	$93.12 \pm 0.68\%$

(b)

version, the dataset contains objects from 101 classes distributed amongst 8,246 samples.

Network Architectures

We used two network configurations to test Matrix-LSTM on both datasets, namely the classifier used in Events-to-Video (Rebecq et al., 2019), and the one used to evaluate the EST (Gehrig et al., 2019a) reconstruction. Both are based on ResNet (He et al., 2016) backbones and pre-trained on ImageNet (Deng et al., 2009). Events-to-Video (Rebecq et al., 2019) uses a ResNet18 configuration maintaining the first 3 channels convolution (since reconstructed images are RGB) while adding an extra fully-connected layer to account for the different number of classes in both N-Calthec101 and N-Cars (we refer to this configuration as *ResNet–Ev2Vid*). EST (Gehrig et al., 2019a) instead uses a ResNet34 backbone and replaces both the first and last layers respectively, with a convolution matching the input features and a fully-connected layer with the proper number of neurons (we refer to this configuration as *ResNet–EST*).

To perform a fair comparison, we replicated the two settings, using the same number of channels in the event representation (although we also tried different channel values) and data augmentation procedures (random horizontal flips and crops of 224×224 pixels). We perform early stopping on a validation set in all experiments, using 20% of the training on N-Cars and using the splits provided by the EST official code repository (Gehrig et al., 2019b) for N-Caltech101. ADAM (Kingma and Ba, 2015) optimizer was used for all experiments with a learning rate of 10^{-4}. Finally, we use a batch size of 64 and a constant learning rate on N-Cars in both configurations. On N-Caltech101, instead, we use a batch size of 16 while decaying the learning rate by a factor of 0.8 after each epoch when testing on *ResNet–Ev2Vid*, and a batch size of 100 with no decay with the *ResNet–EST* setup. Finally, we compute the mean and standard deviation values using five different seeds in all the experiments reported in this section to perform a robust evaluation.

Table 4.2: Results on N-Cars with ResNet18-EST: **(a)** *polarity* + *global ts* + *local ts* encoding, optional SELayer and variable number of bins; **(b)** *polarity* + *global ts* + *local ts* encoding, SELayer and variable number of channels

SE	2 bins	4 bins	9 bins	16 bins
	93.46 ± 0.84%	92.68 ± 0.62%	**93.21 ± 0,49%**	92.01 ± 0.45%
✓	**93.71 ± 0.93%**	92.90 ± 0.62%	**93.30 ± 0,47%**	92.44 ± 0.43%

(a)

	Channels		
bins	4	8	16
1	93.88 ± 0.87%	93.60 ± 0.30%	**94.37 ± 0.40%**
2	93.05 ± 0.92%	93.97 ± 0.52%	**94.09 ± 0.29%**
bins	4	7	8
9	92.42 ± 0.65%	93.56 ± 0.46%	93.49 ± 0.84%

(b)

Results

The empirical evaluation is organized as it follows for both *ResNet–Ev2Vid* and *ResNet–EST*. We always perform hyper-parameters search using ResNet18 on N-Cars, being faster to train and thus allowing us to explore larger parameter space. We then select the best configuration to train the remaining architectures, i.e., ResNet34 on N-Cars and both variants on N-Caltech101.

MATRIX-LSTM + RESNET-EV2VID We start out with the *ResNet–Ev2Vid* baseline (setting up the Matrix-LSTM to output 3 channels) by identifying the optimal time feature to provide as input to the LSTM, as reported in Table 4.1 a. We distinguish between *ts* and *delay* features and between *absolute* and *relative* scope. The first distinction refers to the type of time encoding, i.e., the timestamp of each event in the case of *ts* feature, or the delay between an event and the previous one in case of *delay*. Time features are always range-normalized between 0 and 1, with the scope distinction differentiating if the normalization takes place before splitting events into pixels (*absolute* feature) or after (*relative* feature). In the case of *ts*, *absolute* means that the first and last events in the sequence have time feature 0 and 1, respectively, regardless of their position, whereas *relative* means that the previous condition holds for each position (x, y). Note that we only consider relative delays since it is only meaningful to compute them between the same pixel events. Finally, we always add the polarity, obtaining a 2-value feature $f_i^{(x,y)}$. *Delay relative* and *ts absolute* are those providing the best results, with *ts relative* having higher variance. We select *delay relative* as the best configuration. In Table 4.1 a we also show the effect of applying the same frame normalization used while pre-training the ResNet backbone on ImageNet also to the Matrix-LSTM output. While performing normalization makes sense when training images are very similar to those used in pre-training, as in Events-to-Video (Rebecq et al., 2019), we found out that in our case, where no constraint is imposed on the appearance of reconstructions, this does not improve the performance.

MATRIX-LSTM + RESNET-EST We continue the experiments on N-Cars by considering *ResNet–EST* as a baseline, where we explore the effect of using bins, i.e., intervals, on the quality of Matrix-LSTM surfaces.

Since multiple intervals are involved, we distinguish between *global* and *local* temporal features. The first type is computed on the original sequence $\mathcal{E}$, before splitting events into intervals, whereas the latter locally, within the interval scope $\mathcal{E}_{\tau_b}$. For local features we consider the best options we identified on ResNet-Ev2Vid, namely *delay relative* and *ts absolute*, while we only consider *ts* as global feature since a global delay loses meaning after interval splitting. Results are reported in Table 4.1 b where values for single bin are missing since there is no distinction between *global* and *local* scope. Adding a global feature consistently improves performance helping the LSTM perform integration conditioned on a global timescale, thus enabling the extraction of consistent temporal features. We use the polarity feature together with *global ts + local ts* features in the next experiments since this provides better performance and reduced variance.

The next set of experiments was designed to select the optimal number of bins, searching for the best $B = 2, 4, 9, 16$ as done in EST, while using a fixed *polarity + global ts + local ts* configuration. In these experiments, we also make use of the SELayer (Hu et al., 2018a), a self-attention operation specifically designed to correlate channels. Being the number of channels limited, we always use a reduction factor of 1. Please refer to the original paper (Hu et al., 2018a) for more details. As reported in Table 4.2 a, adding the layer consistently improves performance. We explain this by noticing that surfaces computed on successive intervals are naturally correlated and, thus, explicitly modeling this behaviour helps in extracting richer features. Finally, we perform the last set of experiments to select the Matrix-LSTM hidden size (which also controls the number of output channels). Results are reported in Table 4.2 b. Note that we only consider $4, 7, 8$ channels with 9 bins to limit the total number of channels after concatenation.

Discussion

Results of the top performing configurations for both *ResNet-Ev2Vid* and *ResNet-EST* variants on both N-Cars and N-Caltech101 are reported in Table 4.3. We use *relative delay* with *ResNet-Ev2Vid* and *global ts + local ts* with *ResNet-EST*. Through an extensive evaluation, we show that using Matrix-LSTM representation as input to the baseline networks and training them jointly improves performance by a good margin. Indeed, using the ResNet34-Ev2Vid setup, our solution sets a new state-of-the-art on N-Cars, even surpassing the Events-to-Video model that was trained to extract realistic reconstructions, and that could, therefore, take full advantage of the ResNet pre-training.

The same does not happen on N-Caltech101, whose performance usually greatly depends on pre-training also on the original image-based version,

Table 4.3: Matrix-LSTM best configurations compared to state-of-the-art

Method	Classifier	Channels (bins)	N-Cars	N-Caltech101
H-First (Orchard et al., 2015b)	spike-based	-	56.1	0.54
HOTS (Lagorce et al., 2016)	histogram similarity	-	62.4	21.0
Gabor-SNN (Sironi et al., 2018)	SVM	-	78.9	19.6
HATS (Sironi et al., 2018)	SVM	-	90.2	64.2
	ResNet34-EST Gehrig et al., 2019a	-	90.9	69.1
	ResNet18-Ev2Vid Rebecq et al., 2019	-	90.4	70.0
Ev2Vid Rebecq et al., 2019	ResNet18-Ev2Vid	3	91.0	**86.6**
Matrix-LSTM (Ours)	ResNet18-Ev2Vid	3 (1)	**95.80 ± 0.53**	84.12 ± 0.84
	ResNet34-Ev2Vid	3 (1)	**95.65 ± 0.46**	85.72 ± 0.37
EST (Gehrig et al., 2019a)	ResNet34-EST	2 (9)	92.5	81.7
	ResNet34-EST	2 (16)	92.3	83.7
Matrix-LSTM (Ours)	ResNet18-EST	16 (1)	**94.37 ± 0.40**	81.24 ± 1.31
	ResNet34-EST	16 (1)	**94.31 ± 0.43**	78.98 ± 0.54
	ResNet18-EST	16 (2)	**94.09 ± 0.29**	83.42 ± 0.80
	ResNet34-EST	16 (2)	**94.31 ± 0.44**	80.45 ± 0.55
	ResNet18-EST	2 (16)	**92.58 ± 0.68**	**84.31 ± 0.59**
	ResNet34-EST	2 (16)	92.15 ± 0.73	83.50 ± 1.24

and where Events-to-Video has therefore advantage. Despite this, our model only performs 0.9% worse than the baseline. On the ResNet-EST configuration, the model performs consistently better on N-Cars, while slightly worse on N-Caltech101 on most configurations. However, we remark that the search for the best configuration was indeed performed on N-Cars, while a hyper-parameter search directly performed on N-Caltech101 would have probably lead to better results.

Additional classification experiments

We perform additional experiments on the N-MNIST dataset (Orchard et al., 2015a) and on the newly introduced ASL-DVS (Bi et al., 2019) dataset. On N-MNIST we directly compare with the Ev2Vid (Rebecq et al., 2019) reconstruction procedure, where the custom convolutional network proposed in Rebecq et al. (2019) is used as backbone, while we compare with the EST (Gehrig et al., 2019a) surface on ASL-DVS, making use of ResNet50 (He et al., 2016) as backbone. On both cases, Matrix-LSTM performs better than other event-surface mechanisms and also outperforms alternative classification architectures.

4.5.2 Optical flow prediction

For the evaluation of optical flow prediction, we used the MVSEC (Zhu et al., 2018a) suite. Fusing event-data with lidar, IMU, motion capture, and GPS sources, MVSEC is the first event-based dataset to provide a solid benchmark in real urban conditions. The dataset provides ground truth information

Table 4.4: Classification accuracy (%) on the N-MNIST (Orchard et al., 2015a) dataset.

Method	Classifier	Channels (bins)	Accuracy
H-First	spike-based	-	
HOTS (Lagorce et al., 2016)	histogram similarity	-	80.8
HATS (Sironi et al., 2018)	SVM	-	99.1
G-CNN (Bi et al., 2019)	Graph CNN	-	98.5
RG-CNN (Bi et al., 2019)	Graph CNN	-	**99.0**
Events Count (Bi et al., 2019)	ResNet50	2 (1)	98.4
Ev2Vid (Rebecq et al., 2019)	Ev2Vid custom convnet	1 (1)	98.3
Matrix-LSTM (Ours)	Ev2Vid custom convnet	1 (1)	**98.9 ± 0.21**

Table 4.5: Classification accuracy (%) on the ASL-DVS (Bi et al., 2019) dataset.

Method	Classifier	Channels (bins)	Accuracy
G-CNN (Bi et al., 2019)	Graph CNN	-	87.5
RG-CNN (Bi et al., 2019)	Graph CNN	-	90.1
Events Count (Bi et al., 2019)	ResNet50	2 (1)	88.6
EST (Gehrig et al., 2019a)	ResNet50	2 (1)	99.57
Matrix-LSTM (Ours)	ResNet50	2 (1)	**99.73 ± 0.04**

for depth and vehicle pose and was later extended in (Zhu et al., 2018b) with optical flow information extracted from depth-maps. The dataset has been recorded on a range of different vehicles and features both indoor and outdoor scenarios and different lighting conditions.

Network Architecture

We used the EV-FlowNet (Zhu et al., 2018b) architecture as a reference model. To perform a fair comparison between Matrix-LSTM and the original hand-crafted features, we built our model on top of its publicly available codebase (Zhu et al., 2019b). We first made sure to revert the baseline architecture to the original configuration, checking that could to replicate the paper results. Indeed, the public code contains minor upgrades over the paper version. We contacted the authors that provided us with the needed modifications. These consist of removing the batch normalization layers, setting to 2 the number of output channels of the layer preceding the optical flow prediction layer, and disabling random rotations during training. For completeness, we report the results we obtained by training the baseline from scratch with these fixes in Table 4.6.

The original network uses a 4-channels event-surface, collecting in pairs of separate channels based on the event polarity, the timestamp of the most recent event, and the number of events that occurred in every spatial location. We replaced this representation with a Matrix-LSTM making use of 4 output channels, as well. We trained the model on the *outdoor-day1* and *outdoor-day2* sequences for $300,000$ iterations, as in the original paper. We used the ADAM optimizer with batch size 8, and an initial learning rate of 10^{-5}, exponentially decayed every 4 epochs by a factor of 0.8. We noticed that EV-FlowNet is quite unstable at higher learning rates, while Matrix-LSTM could benefit from larger rates, so we multiply its learning rate, i.e., the Matrix-LSTM gradients, by a factor of 10 during training. Test was performed on a separate set of recordings, namely *indoor-flying1*, *indoor-flying2* and *indoor-flying3*, which are visually different from the training data. The network performance is measured in terms of average endpoint error (AEE), defined as the distance between the endpoints of the predicted and ground truth flow vectors. In addition, as proposed in the KITTI benchmark (Menze and Geiger, 2015) and as done in Zhu et al. (2018b), we report the percentage of outliers, namely points with endpoint error greater than 3 pixels and 5% of the magnitude ground truth vector. Finally, following the procedure used in (Zhu et al., 2018b), we only report the error computed in spatial locations where at least one event was generated since the remaining part of the frame is not visible from event data.

Table 4.6: Optical flow estimation on MVSEC dataset

Method		*indoor-flying1*				*indoor-flying2*				*indoor-flying3*			
		dt=1		*dt=4*		*dt=1*		*dt=4*		*dt=1*		*dt=4*	
		AEE	%Outlier	AEE	%Outlier	AEE	%Outlier	AEE	%Outlier	AEE	%Outlier	AEE	%Outlier
Two-Channel Image (Maqueda et al., 2018)	-	1.21	4.49	-	-	2.03	22.8	-	-	1.84	17.7	-	-
Ev-FlowNet Zhu et al., 2018b	-	1.03	2.2	2.25	24.7	2.12	15.1	4.05	45.3	1.53	11.9	3.45	39.7
Ev-FlowNet (ours)	-	1.015	2.736	3.432	48.685	1.606	12.089	5.957	63.226	1.548	11.937	5.247	57.662
Voxel Grid (Zhu et al., 2019a)	-	0.96	1.47	-	-	1.65	14.6	-	-	1.45	11.4	-	-
EST (Gehrig et al., 2019a)	exp. kernel	0.96	1.27	-	-	1.58	10.5	-	-	1.40	9.44	-	-
	learnt kernel	0.97	0.91	-	-	1.38	8.20	-	-	1.43	6.47	-	-
Matrix-LSTM (Ours)	1 bin	1.017	2.071	3.366	42.022	1.642	13.89	5.870	65.379	1.432	10.44	5.015	57.094
	2 bins	0.829	**0.471**	**2.269**	**23.558**	1.194	**5.341**	**3.946**	**42.450**	1.083	**4.390**	**3.172**	**31.975**
	2 bins + SE	**0.821**	0.534	2.378	25.995	**1.191**	5.590	4.333	45.396	**1.077**	4.805	3.549	36.822
	4 bins	0.969	1.781	3.023	36.085	1.505	11.63	4.870	49.077	1.507	12.97	4.652	43.267
	4 bins + SE	**0.844**	**0.634**	**2.330**	**24.777**	**1.213**	**6.057**	**4.322**	**44.769**	**1.070**	**4.625**	**3.588**	**36.442**
	8 bins	**0.881**	**0.672**	**2.290**	**24.203**	1.292	**6.594**	**3.978**	**42.230**	1.181	5.389	**3.346**	**33.951**
	8 bins + SE	0.905	0.885	2.308	24.597	**1.286**	6.761	4.046	44.366	**1.177**	**5.318**	3.391	35.452

Results

In the previous classification experiments, we observed that the type of temporal features and the number of bins play an important role in extracting effective representations. We expect *time resolution* to be a key factor in optical flow prediction; hence, we focus on measuring how different interval choices impact the flow prediction. Rather than exploring different types of time features, as done in classification experiments, we decided to use the *polarity + global ts + local ts* configuration, which worked well on N-Cars while considering different bin setups. Results are reported in Table 4.6.

As expected, varying the number of bins has a significant impact on performance. The AEE metric, indeed, dramatically reduces by only considering two intervals instead of one. Interestingly, we achieved the best performance by considering only 2 intervals, as adding more bins hurts performance. We believe this behaviour resides in the nature of optical flow prediction, where the network is implicitly asked to compare two distinct temporal instants. This configuration consistently improves the baseline up to 30.76% on *indoor-flying2*, highlighting the capability of the Matrix-LSTM to adapt to low-level vision tasks where both spatial and temporal resolutions are critical factors for performance.

Ev-FlowNet (Zhu et al., 2018b) was tested on two evaluation settings for each test sequence to test how the network adapts to different flow magnitudes. In the first setting, the input frames and corresponding events are one frame apart (denoted as *dt=1*), while in the second one, we consider the events within every four frames (denoted as *dt=4*). While we were able to closely replicate the results of the first configuration (*dt=1*), with a minor improvement in the *indoor-flying2* sequence, the performance we obtain on the *dt=4* setup is instead worse on all sequences, as reported on the second and third rows of Table 4.6. Despite this discrepancy, which prevents the Matrix-LSTM performance on *dt=4* settings to be directly compared with the results reported on the Ev-FlowNet paper, we can still evaluate the relative improvement of using our surface over the original representation on larger flow magnitudes. Using our Ev-FlowNet results as a baseline, we show that Matrix-LSTM can improve the optical flow quality even on the *dt=4* setting,

Table 4.7: Comparison between Matrix-LSTM and ConvLSTM on both Ev2Vid and EST ResNet18 configurations on the N-Cars dataset

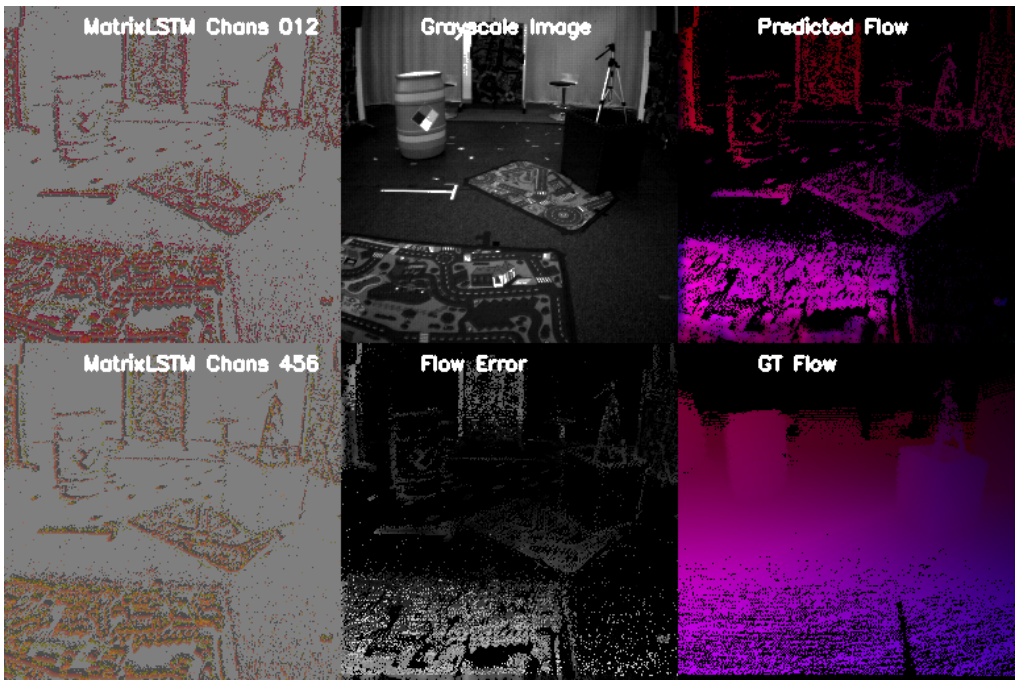
		delay relative		ts absolute	
		3×3	5×5	3×3	5×5
Ev2Vid with 3 chans, 1 bin	Matrix-LSTM (ours)	**95.05 ± 0.96%**	**93.38 ± 0.64%**	**94.92 ± 0.74%**	**94.34 ± 0.94%**
	ConvLSTM	92.33 ± 0.41%	92.65 ± 0.78%	93.97 ± 1.30%	93.61 ± 1.59%
EST with 16 chans, 1 bin	Matrix-LSTM (ours)	**93.14 ± 0.77%**	**92.18 ± 0.28%**	**92.83 ± 1.32%**	**92.15 ± 0.67%**
	ConvLSTM	90.39 ± 0.94%	90.73 ± 1.05%	92.52 ± 1.26%	92.05 ± 0.56%

highlighting the capability of the layer to adapt to different sequence lengths and movement conditions. We report an improvement of up to 39.546% on *dt=4* settings using our results as a baseline, showing the importance of having a mechanism that can model temporal dependencies across potentially distant time lags. The results we obtained show that adding an SELayer only improves performance on the 4 bins configuration for the *dt=4* benchmark, while it consistently helps reducing the *AEE* metric on the *dt=1* setting. By comparing features obtained from subsequent intervals, the SELayer adaptively recalibrates features and helps to model interdependencies between time instants, which is crucial for predicting optical flow. We believe that a similar approach can also be applied to other event aggregation mechanisms based on voxel-grids of temporal bins to improve their performance, especially those employing data-driven optimization mechanisms (Gehrig et al., 2019a).

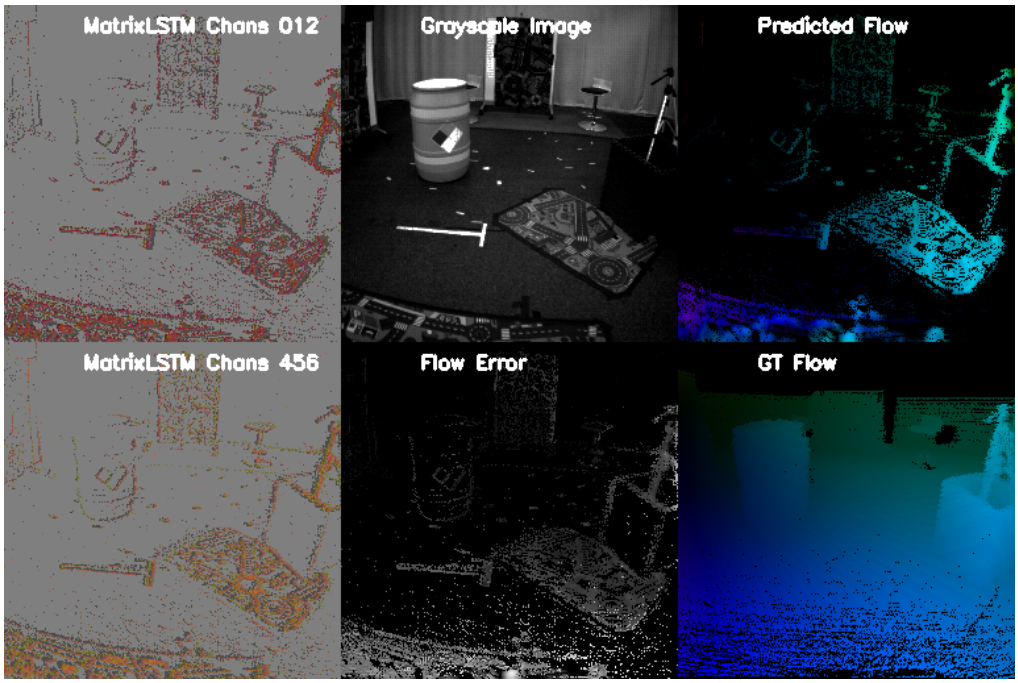
4.6 QUALITATIVE RESULTS

The event aggregation process performed by the Matrix-LSTM layer is incremental. Events in each pixel location are processed sequentially; state and output of the LSTM are updated each time. We propose to visualize the Matrix-LSTM surface as an RGB image by using the ResNet18-Ev2Vid configuration and interpreting the 3 output channels as RGB color. A video of such visualization showing the incremental frame reconstruction on N-Caltech101 samples is provided at this url: `https://marcocannici.github.io/matrixlstm`. See Figure 4.3 for some examples.

We use a similar visualization technique to show optical flow predictions for *indoor-flying* sequences. Since we use our best performing model that uses 2 temporal bins, we decide to only show the first 3 channels of each temporal interval. Moreover, instead of visualizing how the event representation builds as new events arrive, we only show the frame obtained after having processed each window of events.



(a) frame 242



(b) frame 297

Figure 4.3: Visualization of the two-dimensional representation learned by MatrixLSTM for optical flow prediction on the *indoor-flying* sequences from MVSEC (Zhu et al., 2018a). Full videos can be found at `https://marcocannici.github.io/matrixlstm`.

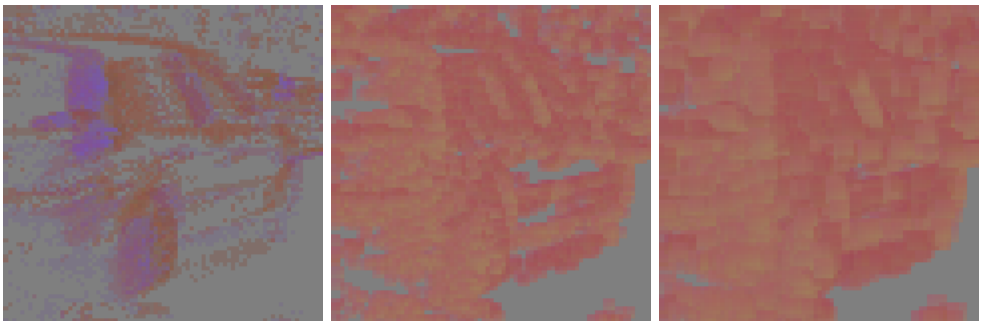


Figure 4.4: Effect of the receptive field size on Matrix-LSTM reconstructions. From left to right, 1×1, 3×3 and 5×5 kernel size. Increasing the receptive field results in blurry reconstructions

4.6.1 Matrix-LSTM vs. ConvLSTM

CLASSIFICATION ACCURACY In Table 4.6 we compare Matrix-LSTM with ConvLSTM (Shi et al., 2015) for different choices of kernel size on the N-Cars (Sironi et al., 2018) dataset using the ResNet18-Ev2Vid backbone. When using ConvLSTM, events are densified in a volume $\tilde{\mathcal{E}}_{dense}$ of shape $N \times T_{max}^{(x,y)} \times H \times W \times F$.

Matrix-LSTM performs better on all configurations, highlighting its capabilities to better handle asynchronous event-based data, compared to ConvLSTM, when larger receptive fields are considered. Indeed, since pixels at different locations most often fire at different times and with different frequencies, the $\tilde{\mathcal{E}}_{dense}[n, i, :, :, :]$ slice processed by the ConvLSTM in each iteration does not contain all simultaneous events. While delays are always consistent within each pixel sequence, they are not within the ConvLSTM kernel receptive field. By using a receptive field larger than 1×1, in fact, ConvLSTM compares a neighborhood of events that occurred at different timestamps and, therefore, not necessarily correlated. Using an absolute temporal encoding alleviates this issue on both Ev2Vid and EST architectures while still performing worst than Matrix-LSTM. Matrix-LSTM allows for greater flexibility when large receptive fields are considered. Our reconstruction layer preserves the original events arrival order within each receptive field, which allows us to achieve better performance both on *ts absolute* and *delay relative* features, without requiring events to be densified during intermediate steps. Moreover, structured *delay relative* features perform better on Matrix-LSTM than simple absolute features.

We do not compare the two LSTMs on the 1×1 configuration since, when using $\tilde{\mathcal{E}}_{dense}$ as input to ConvLSTM, the two configurations compute the same transformation, despite ConvLSTM having to process more padded values. The two settings are indeed computationally equivalent only in the worst case in which all pixels in the batch happen to receive at least one event (i.e., $P = N \cdot H \cdot W$). However, larger receptive fields achieve lower performance than the 1×1 Matrix-LSTM best configuration in Table 4.1

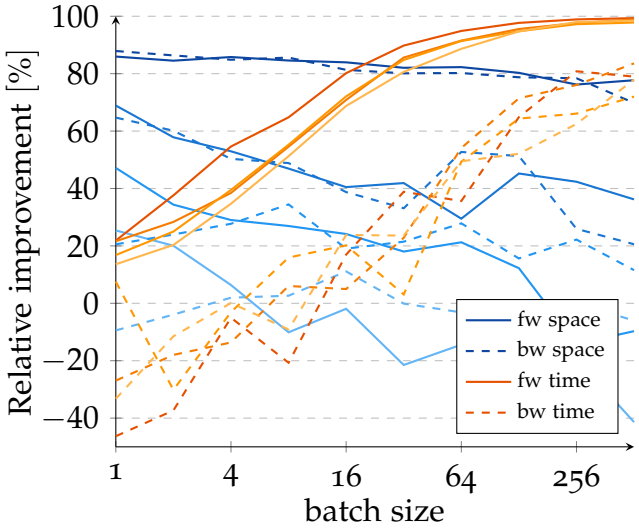
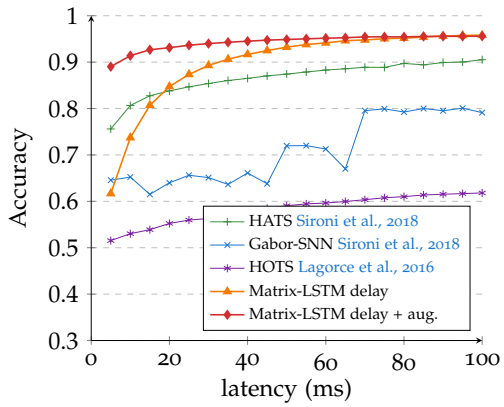


Figure 4.5: Space and time relative improvements of Matrix-LSTM over ConvLSTM as a function of input density (from 10% to 100% with 30% steps). Colors refer to different density, from low density (dark colors) to high density (light colors)

a. Event surfaces produced by the Matrix-LSTM layer are indeed more blurry with larger receptive fields (see Section 4.6), and this may prevent the subsequent ResNet backbone from extracting effective representations. Using a 1×1 kernel enables to focus on temporal information while the subsequent convolutional layers deal with spatial correlation.

TIME AND SPACE EFFICIENCY The 1×1 configurations are compared in terms of space and time efficiency in Figure 4.5. We use the two layers to extract a 224×224 frame from artificially generated events with increasing density, i.e., the ratio of pixels receiving at least one event. The reconstruction is performed using PyTorch (Steiner et al., 2019) on a 12GB Titan Xp, by varying the batch size, the LSTM hidden size, and the number of events in each active pixel (starting from 1 and increasing by a factor of 2 for the hidden size, while increasing by a factor of 10 for the number of events, until allowed by GPU memory constraints). We compute the relative improvement of Matrix-LSTM in terms of sample reconstruction time and peak processing space (i.e., excluding model and input space) during both forward and backward passes. Finally, we aggregate the results by batch size computing the mean improvement over all the trials.

Matrix-LSTM performs better than ConvLSTM on prediction time. The time efficiency improves as the batch size increases, while worse than ConvLSTM on memory efficiency in very dense surfaces ($> 70\%$ density). However, this situation is relatively uncommon in event-cameras since they only generate events when brightness changes are detected. Uniform parts of the scene that remain unchanged, despite the camera movement, do not appear in the



(a)

Method	Bins	Channels	Asynch.	Time [ms]	Speed [kEV/s]
Gabor-SNN (Sironi et al., 2018)	-	-	Yes	285.95	14.15
HOTS (Lagorce et al., 2016)	-	-	Yes	157.57	25.68
HATS (Sironi et al., 2018)	-	-	Yes	7.28	555.74
EST (Gehrig et al., 2019a)	9	2	No	**6.26**	**632.9**
Matrix-LSTM	1	3	No	10.89	385.7
(Ours)	9	2	No	8.25	468.36

(b)

	Mean reconstruction time (on GPU) [ms]		
Bins	Ev-FlowNet (Zhu et al., 2018b) surf.	EST (Gehrig et al., 2019a)	Matrix-LSTM
1	2.53 ± 2.74	3.62 ± 2.35	3.20 ± 0.97
2	2.01 ± 1.22	3.94 ± 1.47	5.18 ± 1.68
9	2.04 ± 1.20	9.09 ± 1.96	4.92 ± 1.47

(c)

Figure 4.6: **(a)** Accuracy as a function of latency (adapted from Sironi et al. (2018)). **(b)** Average sample computation time on N-Cars and number of events processed per second. **(c)** Average time to reconstruct the event surface in MVSEC test sequences.

event stream. For instance, the background sky and road in MVSEC (Zhu et al., 2018a) make *outdoor-day* sequences only have an average 10% of active pixels.

4.6.2 Time performance analysis

We compared the time performance of Matrix-LSTM with other event representations following EST (Gehrig et al., 2019a) and HATS (Sironi et al., 2018) evaluation procedure. In Table 4.6b we report the time required to compute features on a sample averaged over the whole N-Cars training dataset for both ResNet–Ev2Vid and ResNet–EST configurations. Our surface achieves similar time performance than both HATS and EST, performing only ∼2ms slower than EST on the same setting (9 bins and 2 channels). Similarly, in Table 4.6c, we compute the mean surface reconstruct time for MVSEC *indoor-flying* test sequences. While EST can exploit parallel batch computation of events within the same sample, since each event feature is processed independently, Matrix-LSTM relies on sequential computation to reconstruct the surface. However, the custom CUDA kernels we designed, enable bins and pixel sequences to be processed in parallel, drastically reducing the processing time. Please, refer to the additional materials for more details. All evaluations are performed with PyTorch on a GeForce GTX 1080Ti GPU.

In Figure 4.6a we analyze the accuracy-vs-latency trade-off on the N-Cars dataset, as proposed in Sironi et al. (2018), using the ResNet18-Ev2Vid configuration. While the performance of the model, trained on 100ms sequences, significantly drops when very few milliseconds of events are considered, the proposed method still shows good generalization, achieving better performance than the baselines when more than 20ms of events are used. However,

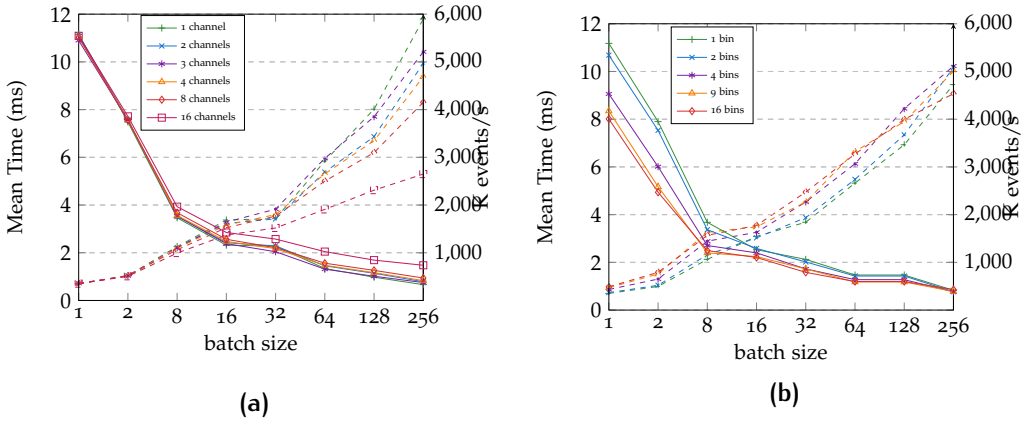


Figure 4.7: Number of processed events per second (dashed lines) and timing (solid lines) with varying number of channels **(a)**, and bins **(b)**

fixing the performance loss on small latencies is just a matter of training augmentation: by randomly cropping sequences to variable lengths (from 5ms to 100ms), our method consistently improves the baselines, dynamically adapting to sequences of different lengths.

While the performance reported in Figure 4.6b are computed on each sample independently to enable a fair comparison with the other methods, in Figure 4.7a and Figure 4.7b we study instead how the mean time required to process a sample over all the N-Cars training dataset and the corresponding events throughput change as a function of the batch size. Both performances dramatically increase when multiple samples are processed simultaneously in batch. This is crucial at training time when optimization techniques greatly benefit from the batch computation.

Furthermore, while increasing the number of output channels for the same choice of batch size increases the time required to process each sample (since the resulting Matrix-LSTM operates on a larger hidden state), increasing the number of bins has an opposite behaviour. Multiple intervals are indeed processed independently and in parallel by the Matrix-LSTM that has to process a smaller number of events in each spatial location, sequentially. In both configurations, finally, increasing the batch size reduces the mean processing time.

4.7 CONCLUSION

In this chapter, we proposed Matrix-LSTM, an effective method for learning dense event representations from event-based data. By modeling the reconstruction with a spatially shared LSTM we obtained a fully differentiable procedure that can be trained end-to-end to extract the event representation that best fits the task at hand. Focusing on efficiently handling asynchronous

data, Matrix-LSTM preserves sparsity during computation and surpasses other popular LSTM variants on space and time efficiency when processing sparse inputs. In this regard, we proposed an efficient implementation of the method that exploits parallel batch-wise computation and demonstrated the effectiveness of the Matrix-LSTM layer on multiple tasks, improving the state-of-the-art of object classification on N-Cars by 3.3% and the performance on optical flow prediction on MVSEC by up to 23.07% over previous differentiable techniques (Gehrig et al., 2019a). Although we only integrate windows of events, the proposed mechanism can be extended to process a continuous streams thanks to the LSTM memory that is able to update its representation as soon as a new event arrives.

5 VIDEO OBJECT SEGMENTATION WITH SPATIO-TEMPORAL FEATURES MODULATION

Generalization to new classes of objects that have never been observed during training is a challenging problem for supervised learning approaches, which usually need to be retrained.

In this chapter, we introduce *ReConvNet*, a recurrent convolutional architecture for one-shot video object segmentation that is able to focus on any specific object of interest by observing a single annotated example at inference time. We propose an efficient solution that learns to self-adapt spatio-temporal features via conditional affine transformations. This approach is simple, can be trained end-to-end, and does not necessarily require extra training steps at inference time. Our method shows competitive results on the DAVIS2016 dataset compared to state-of-the art approaches that use online fine-tuning and outperforms them on DAVIS2017. Furthermore, ReConvNet also shows promising results on the DAVIS-Challenge 2018, placing itself among the top-10 methods.

5.1 INTRODUCTION

Semi-Supervised Video Object Segmentation is the task of segmenting specific objects of interest in a video sequence, given their segmentation in the first frame. This poses an non-trivial challenge for standard supervised methods, as the model cannot be trained as usual to discriminate between a fixed set of classes based on semantics (Badrinarayanan et al., 2017; Long et al., 2015; Visin et al., 2016), but rather has to learn to segment unseen objects based on a single example. In this respect, classic supervised techniques typically fail to easily generalize to new objects whose traits are potentially very different from those of the training data requiring a shift of paradigm in the way learning is conceived. In the context of Video Object Segmentation this is usually solved by implementing a two stage training (Caelles et al., 2017; 2018; Voigtlaender and Leibe, 2017a): a first general training is performed on the entire dataset, then the generic segmentation model is adapted for each test sequence by fine-tuning on transformations of the first frame. Although it has been shown that fine-tuning the model on each specific object can be very effective for this kind of task, this is computationally expensive as it

requires several extra steps of back-propagation for each video sequence. Moreover this method requires to carefully design the data augmentation procedure (Khoreva et al., 2017a) in order to limit the overfitting on the first frame. This is often non-trivial, especially if the object changes across the frames significantly or is occluded by other objects in the scene. Indeed, generating high-quality segmentations from a single frame is a hard task by itself and adds an extra layer of complexity to the problem; producing convincing synthetic sequences requires, e.g., to generate realistic transformations of the foreground object and recover the background in the parts of the image that were occluded by the foreground in the original image but are not after the foreground transformation is applied.

A better way of thinking is to consider each video's segmentation as separate problems sampled from a *distribution of tasks*, where each task involves distinguishing new objects by observing a single annotated frame. This scenario is generally referred to in the literature as *few-shot learning*, where tasks challenge models to learn a new concept or behaviour with very few examples or limited experience (Fei-Fei et al., 2007; Lake et al., 2011; Li Fei-Fei et al., 2006). One approach to address this class of problems is *meta-learning* (Schmidhuber, 1987), a broad family of techniques focused on *learning how to learn* (Baxter, 1995; 1998; 2000; Hochreiter et al., 2001; Thrun and Pratt, 1998) or to quickly adapt models to new information (Hinton and Plaut, 1987).

5.1.1 Main Contributions

Following the meta-learning paradigm, we propose an approach for video object segmentation that allows a *base network* to self-adapt to the specific object of interest at inference time, without resorting to expensive fine-tuning steps. This mechanism is inspired by *fast-weights* (Schmidhuber, 1992), where a *slow network* generates the weights of a second one – namely the *fast network* – to fast adapt on the fly to a new task or to a change in the environment.

While it is often more effective to specifically fine-tune the network to distinguish the instance of interest, a self-adaptation mechanism that redirects a known and well-performing generic segmentation strategy to focus on a specific element of interest appears more biologically plausible (DiCarlo et al., 2012; Lee et al., 2015) and resonates better with those applications where an increase in inference time is non-negligible.

Following this mindset, we introduce *ReConvNet*, a recurrent convolutional architecture for one-shot video object segmentation. The proposed model is composed of a core (fast) segmentation network and two slow networks that modulate the main one via affine transformations of its activations. By means of these *modulators*, the network is able to autonomously adapt to segment unseen object(s) without resorting to extra fine-tuning steps at inference time.

The main contributions of this work are as follows:

- We reinterpret the successful OSMN (Yang et al., 2018) video segmentation architecture from a meta-learning perspective and propose an extension with convLSTM units that allow modeling intrinsic temporal correlations across frames of the sequence. Our improved network outperforms the baseline, showing the benefits of explicitly harnessing the temporal interactions between the frames.
- Our model exhibits comparable performances to methods that make use of online fine-tuning on DAVIS2016 and outperforms them on the more challenging DAVIS2017.
- We score 10-th in the DAVIS challenge 2018 without resorting to online fine-tuning or other post-processing steps.
- We show that feature modulation is orthogonal to online fine-tuning and that, indeed, combining the two results in a further performance boost.

5.1.2 General Problem definition

For simplicity, we first consider the general case of N-way K-shot classification tasks, where the goal is to assign categories from a set of labels $\{1 \dots N\}$ by observing K annotated samples. We follow the *episodic training* formulation proposed by Vinyals et al. (2016).

Each task instance τ_i is a classification problem sampled from a task distribution $p(\tau)$. As in supervised learning, samples are divided into a *meta-training* set $\mathcal{S}^{tr}$, *meta-validation* set $\mathcal{S}^{val}$ and *meta-test* set $\mathcal{S}^{test}$, with a disjoint set of target classes to evaluate performance on new tasks.

Each task $\tau_i \sim p(\tau)$ consists in two separate sets: a *training set* $\mathcal{D}_i^{train}$ (or *support set*), and a *validation set* $\mathcal{D}_i^{val}$ (or *query set*) which only contain samples from N classes randomly selected from the appropriate meta-set. In general, the support set for an N-way K-shot classification task can be defined as:

$$\mathcal{D}_i^{train} = \{(\mathbf{x}_n^k, y_n^k | k = 1 \dots K, n = 1 \dots N)\}, \tag{5.1}$$

a collection containing K samples, also called *shots*, for each n class. The query set $\mathcal{D}_i^{val}$ usually contains several other samples of the same classes and it is used to evaluate the generalization performance of the model on new instances of the task. Note the importance of understanding the difference between the meta-validation set $\mathcal{S}^{val}$, a collection of independent problems used for model selection, and the query set $\mathcal{D}_i^{val}$, which contains samples of a specific task τ_i on which the selected model is tested.

GENERATING TASKS Starting from a classification dataset annotated for supervised learning with enough C classes , it is possible to generate a dataset of episodes for few-shot classification by splitting into three sets of disjoint classes (*meta-train*, *meta-val*, *meta-test*). An N-way K-shot classification task is then obtained with *stratified sampling*, by first sampling N different classes from the available ones in the meta-set, and then sampling several instances of each class. The collected samples are then split into the support and query sets, where the first should contain K samples for each class. Since tasks are independent problems, labels are randomly assigned when sampling the tasks classes.

ONE-SHOT VIDEO OBJECT SEGMENTATION Following the definition above, semi-supervised Video Object Segmentation can be cast as a few-shot learning problem. Tasks consist of segmenting specific objects in video sequences by using a single shot annotation. In meta-learning terms, the support set consists of the first annotated frame of the sequence, and the query set consists of the rest of the video, which is used to evaluate the generalization performance of the segmentation algorithm. We distinguish between *foreground/background segmentation* tasks where the number of classes $N = 2$, the object of interest plus the background, and *multi-object segmentation* tasks where we want to distinguish between multiple objects in the same video and $N > 2$.

5.2 BACKGROUND AND RELATED WORK

Video Object Segmentation witnessed an increasing interest in the last few years, also thanks to the release of the DAVIS datasets (Perazzi et al., 2016) and to the related competitions (Caelles et al., 2018; Pont-Tuset et al., 2017). This task can be addressed either in an unsupervised or semi-supervised fashion. In the first case, no prior information on the object of interest is available, while in the second, the segmentation mask of the object is given for the first frame. In this work, we focus on the semi-supervised setting, in the specific instance of one-shot video segmentation.

5.2.1 Semi-supervised Video Object Segmentation

MASK PROPAGATION AND FINETUNING In the semi-supervised setting the Video Object Segmentation problem can be considered as tracking the mask of each given object throughout the sequence. MaskTrack (Khoreva et al., 2017b) pretrains a part of the network on static images and then builds on it to refine a rough mask estimate of the segmentation of the previous frame into the predicted mask of the current frame. A variant of the network also

explores the use of optical flow information to transform the previous mask into an estimate of the current mask to be then refined by the network.

OSVOS and its variants (Caelles et al., 2017; 2018; Voigtlaender and Leibe, 2017a) are based instead on a fully-convolutional network (Long et al., 2015) adapted from VGG-16 (Simonyan and Zisserman, 2015) that is first trained on the entire dataset and then fine-tuned on the target instance at inference time. Both OSVOS and MaskTrack have been very influential, indeed the first three approaches (Khoreva et al., 2017a; Li et al., 2017a; Nguyen et al., 2017) of the DAVIS2017 Challenge on instance video segmentation base on MaskTrack and the next six (Cheng et al., 2017; Newswanger and Xu, 2017; Shaban et al., 2017; Sharir et al., 2017; Voigtlaender and Leibe, 2017b; Zhao, 2017) on OSVOS. In order of performance, Li et al. (2017a) improves over MaskTrack by recovering the instances missed by the mask propagation module, e.g., in case of occlusions, via a re-identification module that ranks a set of bounding box candidates by similarity to the instance template. Nguyen et al. (2017) also implements a re-identification module but, as opposed to Li et al. (2017a), two different pathways are employed to detect and track human and non-human bounding boxes. Each box is then segmented using binary SVM classifiers and an heuristic to determine the relative depth of the boxes.

DATA AUGMENTATION TECHNIQUES Methods based on OSVOS and MaskTrack rely heavily on the quality of the data augmentation procedure. Indeed, applying simple random rigid transformations, as is usually done for image recognition, is not enough for video object segmentation. The shape and pose of the objects often vary significantly throughout the sequence from those of the initial frame, which makes it hard to generate samples that exhibit enough variety and complexity for generalization. Nonetheless, Lucid Dream (Khoreva et al., 2017a) proposes a data augmentation procedure that generates convincing synthetic frames from the initial segmented mask. The augmentation is performed by removing the foreground object, in-painting the background, applying a global translation and an affine non-rigid transformation to both the foreground and the background, and finally recomposing the scene via Poisson matting (Sun et al., 2004).

TEMPORAL CORRELATION Most of the works based on MaskTrack evolve one prediction into the next in a Markovian fashion, while those based on OSVOS usually perform single frame segmentation disregarding the temporal correlation between frames completely. Only a few methods exploit the full temporal correlation between the frames, usually focusing on capturing the motion displacement to use it as a guidance for segmentation. Among those, Cheng et al. (2018), Hu et al. (2018b), and Xiao et al. (2018a) either exploit the optical flow or introduce explicit tracking components in the network to perform segmentation. Tokmakov et al. (2017) instead, similarly

to our work, model the evolution of objects over time using recurrent neural networks. Here a two streams encoder decouples the motion and appearance cues, which are then recombined by a convGRU (Ballas et al., 2016) decoder.

EFFICIENCY Finally, a number of works focus on inference efficiency and propose solutions that do not resort to online finetuning. In Wug Oh et al. (2018) the authors build a Siamese encoder-decoder network for fast object video segmentation that can do inference at 10 FPS, while Chen et al. (2018b) uses a FCN to learn an embedding space with a triplet loss and then classifies each pixel according to nearest neighbor in embedding space. Interestingly, Chen et al. (2018b) also find it beneficial to provide the model with explicit spatio-temporal information in the form of indices concatenated to the input frames.

5.2.2 Meta-Learning for few-shot learning problems

Supervised approaches generally perform poorly on few-shot learning tasks, where a model has to quickly generalize from very few examples. If the data distribution drifts from the one experienced during training, the model will likely need to be retrained to adapt to the changes. This can be expensive and not always possible.

A more flexible paradigm for this kind of problem is to build a system that is able to learn how to adapt itself to the new data. Learning a general algorithm capable to tailor itself to the specific instance of the problem is know as *Learning to Learn* or *Meta-Learning* (Schmidhuber, 1987). Meta-Learning aims to learn how to extract knowledge from a collection of tasks that can be reused to solve novel, unobserved tasks sampled from the same distribution. Meta-learning is particularly tailored for few-shot learning problems where multiple related tasks share a common structure. Each task or *episode* contains only a few labeled examples, the *support set*, that are used to adapt the model and generalize to new examples, the *query set*.

BILEVEL OPTIMIZATION VIEW Meta-training is usually cast as *bilevel optimization*, namely a hierarchical optimization problem where an *outer* objective is constrained by the solution of an *inner* optimization problem (Franceschi et al., 2018). Following this notation, meta-training can be formalised as learning over a distribution of tasks $p(\tau)$ (or *episodes*) follows:

$$\begin{aligned} \omega^* &= \arg\min_{\omega} \sum_{i=1}^{M} \mathcal{L}^{meta}(\theta_i^*(\omega), \omega, \mathcal{D}_i^{val}) \\ \text{s.t.} \quad \theta_i^*(\omega) &= \arg\min_{\theta} \mathcal{L}^{task}(\theta, \omega, \mathcal{D}_i^{train}) \end{aligned} \tag{5.2}$$

where $\mathcal{L}^{meta}$ and $\mathcal{L}^{task}$ refer to the outer and inner objectives respectively, e.g., cross-entropy in the case of few-shot classification or segmentation,

and each task is defined as $\{\mathcal{D}_i^{train}, \mathcal{D}_i^{val}\} \sim p(\tau)$, by sampling from the task distribution. The inner and outer levels follow a leader-follower (Stackelberg, 1952) structure: the inner level optimization is conditional on the learning of ω defined by the outer loop, but it cannot directly modify ω during its optimization. In practice, the outer problem learns ω such that it produces a model $\theta_i^*(\omega)$ that performs well on the validation set $\mathcal{D}_i^{val}$ after the optimization on the training set $\mathcal{D}_i^{train}$. Note that ω could indicate hyper-parameters (Franceschi et al., 2018; Lorraine et al., 2020), weight initializations (Finn et al., 2017), or even parametric loss/regularization functions (Balaji et al., 2018; Bechtle et al., 2021; Li et al., 2019).

Optimization based approaches explicitly solve the inner optimization in Equation (5.2). A famous example is MAML (Finn et al., 2017), which aims to find a shared initialization of the model parameters, across all tasks of the distribution that can be quickly adapted to task-specific parameters in a few steps of gradient descent. The meta-initialization is then obtained by differentiating through the updates of the base model. The main issues with this class of algorithms is the heavy load of computation they require when meta-training complex learners because of the necessity to compute second-order derivatives. Simplified models have been proposed to scale optimization-based approaches and overcome their limitations, performing on par or better than the original MAML model (Antoniou et al., 2019; Raghu et al., 2020; Rajeswaran et al., 2019; Vuorio et al., 2019; Zintgraf et al., 2019). More elaborate alternatives also learn step sizes(Li et al., 2017b), gradient preconditioning (Flennerhag et al., 2020; Park and Oliva, 2019), or train recurrent networks to predict steps from gradients (Andrychowicz et al., 2016; Ravi and Larochelle, 2017).

MODEL-BASED VIEW A more direct way of fast-adapting to a new task is to use model-based approach rather than explicitly solve an iterative optimization problem as in Equation (5.2). This approach has been proposed in different context under many names, but it stems from the original idea of *fast-weights* (Hinton and Plaut, 1987; Schmidhuber, 1992) where a *slow network* $g(\cdot)$ synthesizes the model to solve a task $\tau_i \sim p(\tau)$ by generating context-dependent weights for a *fast network* $f(\cdot)$ conditioned on the support set:

$$\omega^* = \arg\min_{\omega} \sum_{\substack{\tau_i \sim p(\tau) \\ \{\mathcal{D}_i^{train}, \mathcal{D}_i^{val}\} \in \tau_i}} \sum_{(\mathbf{x},y) \in \mathcal{D}_i^{val}} \mathcal{L}^{task}(f(\mathbf{x}; g(\mathcal{D}_i^{train}, \theta)), y) \tag{5.3}$$

Model-based approaches are powerful meta-learner, however generating the weights on an entire neural network is not sustainable and generally ends up in rapidly overfitting the training data. Ha et al. (2017) proposed *HyperNetworks*, a modification of the original fast-weights formulation where the network infers a transformation of the weights rather than generating the weights themselves. A recurrent version of the model is also proposed,

where the weights of an RNN are dynamically generated at each timestep adapting to the changes of the input sequence. Recently *FiLM* (Perez et al., 2018) suggested to extend conditional batch-normalization with a module that produces an affine transformation to be applied to the features of each layer of the main network to impose a more explicit conditioning. The idea of modulating the features conditionally has been initially applied to improve the efficiency of the inference step in artistic style transfer (Dumoulin et al., 2017; Ghiasi et al., 2017). In the context of object segmentation, *features modulation* has been investigated on the DAVIS dataset by OSMN (Yang et al., 2018), that extends OSVOS (Caelles et al., 2017) by specializing a part of the architecture to condition the predictions on the target object.

5.3 RECONVNET: A RECURRENT CONVOLUTIONAL MODEL FOR FAST OBJECT SEGMENTATION

We continue on the same path traced by OSMN Yang et al., 2018, having a model-based meta learner that is able to self-adapt to perform segmentation on new objects without necessarily resorting to gradient update steps. Specifically, this is achieved by modulating the activations of a recurrent convolutional *Segmentation Network* (SN) via a single forward pass of a *Modulating Network* (MN) that outputs a scale parameter for each channel based on the object features, and a shift parameter for each location that acts as a spatial attention mechanism similar to Stollenga et al. (2014). The segmentation network learns to perform generic segmentation, and the modulators fast-adapts its activation to attend to the object of interest. We now give a detailed description of the main components of the proposed model.

5.3.1 Segmentation Network

The segmentation module is an encoder-decoder network that processes the frames and produces the segmentation masks. This component extends the original OSMN Segmentation Network, which is implemented as a Fully-Convolutional Network (FCN) based on the VGG-16 feature extractor (Simonyan and Zisserman, 2015).

In order to recover details at multiple scales, the encoder layers are propagated to the decoder after bilinear upsampling in a hyper-column fashion (Hariharan et al., 2015). While in OSMN each frame is processed independently, often resulting in segmentation masks that lack temporal consistency and exhibit high variance across the sequence. A natural way to incorporate temporal structure into the model is to add recurrent units. convLSTM layers, an adaptation of the original LSTM (Hochreiter and Schmidhuber, 1997b) cell that takes into account the spatial structure of

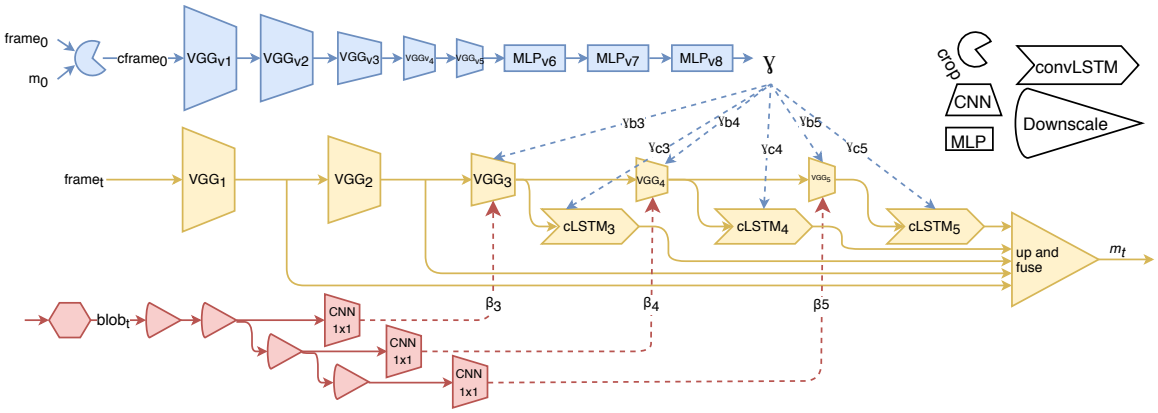


Figure 5.1: ReConvNet architecture. The Segmentation Network (yellow) is a VGG-16-based architecture enhanced with convLSTM layers. The Visual Modulator (blue) biases the SN to be selective toward the object of interest via a per-channel multiplicative interaction, while the Spatial Modulator (red) enforces a spatial prior via an additive per-pixel modulation.

its inputs as well as their temporal correlation (Xingjian et al., 2015). The convLSTM blocks are interleaved to the last three VGG-16 layers to endow the network with multi-scale spatio-temporal processing capability.

5.3.2 Visual Modulator

The *visual modulator* is in charge of biasing the activations of the segmentation network to target the object of interest. This strong conditioning is achieved by a second VGG-16 network, deprived of the last classification layer, that takes as input the first frame cropped around the target object and resized to 224×224, and produces a set Γ of vectors of *scaling coefficients* — one for each of the last three convolutional layers of the Segmentation Network. In addition to the coefficients computed for the non-recurrent components of the SN, we also compute those for the convLSTM layers. All the visual modulation coefficients are multiplied to the feature maps:

$$\tilde{f}_i = \gamma_i \odot f_i, \tag{5.4}$$

where $\odot$ indicates channel-wise multiplication. This has the effect of enhancing the maps related to the target object and suppressing the least useful, potentially distracting ones, allowing the segmentation network to quickly adapt to the object of interest without gradient-based optimization at inference time. Interestingly, the parameters produced by the visual modulator can be interpreted as a *class embedding*, as similarly done in neural processes-based architectures which generates the weights classifier directly from task or class embeddings (Garnelo et al., 2018; Gordon et al., 2019; Requeima et al., 2019).

5.3.3 Spatial Modulator

To help discriminating between multiple instances of the same object and, more generally, to provide a loose prior on the location of the target object, the network is also enriched with a spatial attention mechanism. A rough estimate of the position of the target object can be obtained by fitting a *"gaussian blob"* on the segmentation predicted at time t and fed to a *Spatial Modulator* component. This, in turn, produces a set of *shift coefficients* β, one for each of the last three VGG layers, via a 1×1 convolution applied to the blob downsampled to the layer's resolution. Note that, as opposed to the VM case, we do not generate the modulation coefficients for the convLSTM layers. The spatial coefficients are summed pixel-wise to the activations of the corresponding layers, hence shifting the focus on the parts of the image where the object is more likely located. The two modulators are then combined in a single transformation and applied to all the features:

$$\tilde{f}_i = \gamma_i \odot f_i + \beta_i(t). \tag{5.5}$$

Notably, while the VM generates a single set of weights for all the frames – being applied in the same way to all the frames of the sequence – the Spatial Modulator produces an estimate of the position of the object for each frame at instant t, given the blob at the previous time step $t-1$.

5.4 EXPERIMENTS

In this section we first describe the experimental settings used to evaluate the *ReConvNet* model, then we comment the results obtained with our best configuration on the single and multi instance segmentation tasks on DAVIS.

TWO-STAGES TRAINING In order to make use of the relative abundance of segmented static images for pre-training, we split the training process in two phases. First, we train the non-recurrent components of the model on MSCOCO (Lin et al., 2014) to learn segmentation coupled with modulation. In this phase, the backbone network learns general basic segmentation strategies, while the visual modulator learns to extract the appearance characteristics and extract a task embedding based on a cropped version of the object of interests, namely the *support set*. We generate *synthetic tasks* from static images by augmenting the inputs to the modulators with random shift, scale, and rotation transformations, simulating the test time condition where the object changes appearance during the video sequence with respect to the annotated one.

In a second phase, we train the full network on video sequences from DAVIS to account for the recurrent components (convLSTM cells) as well as to make use of the modulation to focus on the target object throughout

the sequence and capture the spatio-temporal correlation among frames. During training, multiple tasks can be obtained by the same video. Each task, in fact, consists of segmenting K consecutive frames from a randomly sampled video. If the video contains more than one object, a single one is chosen randomly as the target one. To improve data efficiency, instead of always using the first frame of the sequence as a support set, we randomly select a frame from the video outside of the ones composing the task. Again, we apply random augmentations to the inputs of the visual and spatial modulators. We also employed early-stopping to prevent overfitting.

EXPERIMENTAL SETTING To ensure a fair comparison between *ReConvNet* and the OSMN baseline we initialize the components of our architecture that are in common with the OSMN model with the pretrained weights as provided by the authors. This is done for both DAVIS2016 and DAVIS2017, to ensure that any improvement can be clearly attributed to the introduction of a recurrent architecture.

In the first stage of training all the experiments were trained for 15 epochs in total optimizing cross-entropy loss with Adam (Kingma and Ba, 2015), learning rate 10^{-5} for the first 10 epochs, decreased to 10^{-6} for the last 5 epochs, and batch size 10. For the initialization of the remaining modules, in an effort to minimize the factors of variations with respect to the baseline, the extra channels of the visual modulator are initialized as in (Yang et al., 2018) and, similarly, the input-to-hidden convolutions in the convLSTM layers use the same initialization as the convolutional layers in the baseline. Lastly, the hidden-to-hidden convolutions are initialized to be orthogonal (Saxe et al., 2014).

We further train the full architecture on DAVIS with a lower learning rate for the pre-trained non-recurrent component than for the recurrent ones, namely 10^{-6} and 10^{-5} respectively. We found it beneficial to train with the Lovasz loss (Berman et al., 2018) that directly optimizes the IoU measure. When online fine-tuning is used, the model is trained on each test sequence with random transformation of the first frame for 300 iterations and learning rate 10^{-6} for all components.

5.4.1 Single Object Segmentation

We first evaluate our model on DAVIS2016, that focuses on single objects. This is a hard task that allows us to validate the model and to compare with the OSMN baseline. As shown in Table 5.1, thanks to the combination of spatio-temporal consistency given by the convLSTM units and their features modulation, *ReConvNet* outperforms OSMN by 5.4 points on the mean IoU ($\mathcal{J}$-mean) metric and places itself right below the top three models in the

public leaderboard of the semi-supervised approaches [1] when comparing on the average between the $\mathcal{J}$ and $\mathcal{F}$ scores.

It is important to note that the top three models perform online fine-tuning on the first frame of the video sequence at inference time. Moreover, OSVOS (Caelles et al., 2017) utilizes a boundary snapping approach, onAVOS (Voigtlaender and Leibe, 2017a) makes use of a CRF post-processing step, and OSVOS-S (Maninis et al., 2018) incorporates instance-aware semantic information from a state-of-the-art instance segmentation method to improve its accuracy. While these methods require expensive computation steps at inference time that are normally not needed when resorting to features modulation, nothing prevents to pair this technique with online fine-tuning or CRF post-processing to further boost the performance. Indeed, with a few steps of fine-tuning at inference time *ReConvNet* gains 6.9 points on the $\mathcal{J}\&\mathcal{F}$-mean, only 0.5 points below onAVOS that scored 2nd in the public leaderboard.

5.4.2 Multiple Objects Segmentation

DAVIS2017 The most recent version of DAVIS introduces the challenging task of multiple objects segmentation. On this dataset *ReConvNet* has been trained by feeding the visual modulator with one randomly picked object from the scene at a time and using the segmentation of the same object in the current frame as target. Table 5.1 shows that *ReConvNet* adapts very well to the multiobject task outperforming the baseline OSMN by 10.9 points on the $\mathcal{J}\&\mathcal{F}$-mean metric. Remarkably, our method also outperforms the state-of-the-art OSVOS and onAVOS by 5.4 and 0.3 points, respectively, without the need of expensive online fine-tuning. By also performing online fine-tuning, the $\mathcal{J}\&\mathcal{F}$-mean improves by 4.5, that is 2.2 points more than OSVOS-S, the current state of the art in the public leaderboard on the DAVIS2017 validation set.

DAVIS CHALLENGE 2018 We participated to the DAVIS Challenge 2018 (Caelles et al., 2018), a public competition based on the DAVIS2017 dataset that extends it with much more challenging videos. In particular, two new sets of sequences are provided: the *test-dev* set for preliminary evaluation with unlimited submissions and the *test-challenge* for the final evaluation (limited to 5 submissions). Videos from both sets complicate the segmentation task with several occlusions and many similar instances of the same object which are hard to distinguish.

On all the experiments of the challenge, we retrained ReConvNet on both training and validation sets. Our preliminary evaluation on the test-dev set scored 52.7 and 62.9 on $\mathcal{J}\&\mathcal{F}$-mean, respectively without and with online fine-tuning, ranking 8-th in the test-dev public leaderboard.

1 https://davischallenge.org/davis2016/soa_compare.html

Table 5.1: Comparisons of our approach vs OSMN baseline (1^{st} and 2^{nd} training stages) and top-3 state-of-the-art algorithms on DAVIS2016 and DAVIS2017 validation sets. **Legend.** FT: Online fine-tuning on the first frame; M: Mean; R: Recall; D: Decay.

		DAVIS2016							DAVIS2017						
Method	FT	$\mathcal{J}\&\mathcal{F}$	$\mathcal{J}$			$\mathcal{F}$			$\mathcal{J}\&\mathcal{F}$	$\mathcal{J}$			$\mathcal{F}$		
		M ↑	M ↑	R ↑	D ↓	M ↑	R ↑	D ↓	M ↑	M ↑	R ↑	D ↓	M ↑	R ↑	D ↓
OSMN (1^{st}) (Yang et al., 2018)	✗	72.2	-	-	-	-	-	-	-	-	-	-	-	-	-
OSMN (2^{nd}) (Yang et al., 2018)	✗	-	74.0	-	-	-	-	-	54.8	52.5	60.9	21.5	57.1	66.1	24.3
ReConvNet (ours)	✗	78.1	**79.4**	89.6	7.7	76.8	86.6	7.7	**65.7**	**62.7**	**70.5**	21.6	**68.7**	**77.3**	**21.6**
OSVOS (Caelles et al., 2017)	✓	80.2	79.8	93.6	14.9	80.6	92.6	15.0	60.3	56.6	63.8	26.1	63.9	73.8	27.0
onAVOS (Voigtlaender and Leibe, 2017a)	✓	85.5	86.1	96.1	5.2	84.9	89.7	5.8	65.4	61.6	67.4	27.9	69.1	75.4	26.6
OSVOS-S (Maninis et al., 2018)	✓	86.6	85.6	96.8	5.5	87.5	95.9	8.2	68.0	64.7	74.2	**15.1**	71.3	80.7	**18.5**
OSMN (2^{nd}) (Yang et al., 2018)	✓	-	-	-	-	-	-	-	60.1	-	-	-	-	-	-
ReConvNet (ours)	✓	85.0	85.4	95.9	8.5	84.6	93.9	12.1	**70.2**	**66.6**	**75.4**	28.1	**73.7**	**83.1**	29.6

On the test-challenge set *ReConvNet* scored 54.5 $\mathcal{J}\&\mathcal{F}$-mean, and 51.8 and 57.2 $\mathcal{J}$-mean and $\mathcal{F}$-mean, respectively, ranking 10-th in the final DAVIS Challenge 2018 evaluation. This is an encouraging result considering that no online fine-tuning was employed. Indeed by adding some fine-tuning steps at inference time it is reasonable to expect a performance boost similar to the one consistently witnessed in the previous experiments. Unfortunately we could not submit the results with the fine-tuned model for lack of time.

5.5 RESULTS ANALYSIS

We conducted a suite of experiments designed to evaluate the importance of each component of ReConvNet, in parallel to the usual hyper-parameters search. For simplicity, most of the experiments have been run first on DAVIS2016, which allowed to iterate faster over the several configurations thanks to its reduced size. Once we found good candidate configurations, we applied and tested them on the more complex multiple objects task. In this analysis we focus on the second stage of training, considering only the DAVIS dataset.

LEARNING RATE We investigated the impact of the learning rate in two configurations: i) using the same value (10^{-5}) for all ReConvNet blocks, ii) using a lower (10^{-6}) learning rate for the blocks that have already been pre-trained on the COCO dataset to encourage a more impactful optimization on the recurrent layers. We found that using a lower learning rate for the non-recurrent blocks works better leading to an improvement of $\approx$ 2.8 points on the $\mathcal{J}$-mean metric that went from 75.29 to 78.08 points.

SEGMENTATION LOSS In order to enforce the network to use the temporal information coming from the earlier frames, the loss is computed considering only the prediction of the last frame. We also tested a different configurations

in which the segmentation is predicted for each frame in the sequence and we minimized the sum over the loss computed on each frame. In this case we experimented a decay of performance of ≈ 1.4 points on the $\mathcal{J}$-mean and ≈ 2 points on the $\mathcal{F}$-mean.

For our basic experiments we adopted the balanced cross-entropy loss proposed for segmentation by (Caelles et al., 2017), the same used in OSMN, to ensure a fair comparison. Notably, we report a 4.81 points of improvement on the $\mathcal{J}$-mean metric that can be attributed solely to the proposed recurrent components.

Starting from the basic experiment, a slight improvement can be achieved by training the best configuration with the Lovász loss (Berman et al., 2018), that is specifically designed to optimize the IoU metric. This allows us to obtain our best result on the $\mathcal{J}$-mean metric which scores 79.43 without fine-tuning. In addition, we found that the Lovász loss exhibit a faster convergence, reaching the best performance in only 17 training epochs as opposed to the 39 epochs needed in the cross-entropy case. The advantages of the Lovász loss can be appreciated even more on the multiple object task. On the DAVIS2017 validation set the cross-entropy trained model scores 59.1 points on the $\mathcal{J}\&\mathcal{F}$-mean metric, which is already 4.3 points better than OSMN, while the Lovász one obtains a further boost of 6.6 accuracy points, scoring a total of 65.7 points.

RECURRENT VISUAL MODULATION In order to evaluate the effect of guiding the recurrent layers via the visual modulator (i.e., in addition to the VGG layers already modulated by OSMN) we compared the performance of two networks trained with the same hyperparameters, only one of which encompassing the convLSTM's modulation. The performance is marginally worse when the visual modulator is not used, specifically 78.55 $\mathcal{J}$-mean and 76.77 $\mathcal{F}$-mean versus 78.81 and 77.26 respectively.

PRE-TRAINING Supervised and semi-supervised models that implicitly, or explicitly, make use of semantic information, need much more data than the one available in the DAVIS dataset in order to generalize well on the video object segmentation task. As it can be expected, the problem is exacerbated by models with high capacity, and even more by those that exploit a visual modulator to tackle semi-supervised segmentation. As shown in Yang et al. (2018) the set of parameters Γ, produced by the Visual Modulator, pushes the model to learn a semantic mapping in an embedding space where visually similar objects are close in ℓ_2 distance. Learning this mapping requires a large enough amount of diverse examples. MSCOCO (Lin et al., 2014) provides a wide range of classes and intra-class variations resulting in a very well suited pre-training dataset for DAVIS. Furthermore, the single frame pre-training procedure proved to be an essential proxy to bootstrap the temporal-consistent segmentation. Indeed, the DAVIS dataset contains

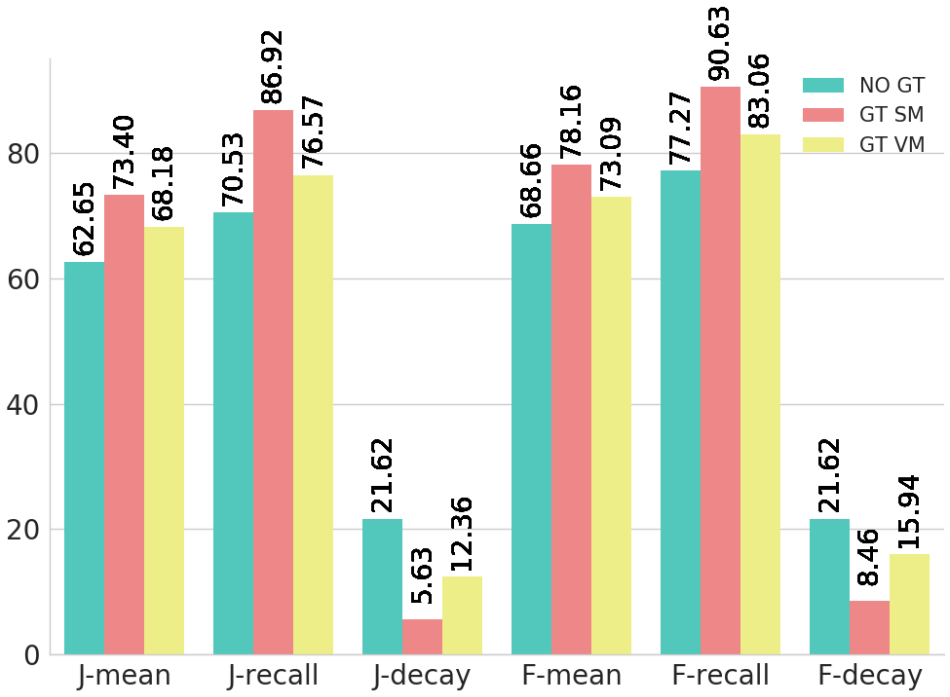


Figure 5.2: Importance of Visual and Spatial Modulators. This histogram compares the evaluation scores of a trained model (green) respectively with the same model when the visual guide ground truth is given (yellow) and when an oracle on the spatial location is available (red). The results of this study show how a precise localization is more valuable for the network than the appearance information about the object.

only a few examples for most semantic classes, making it very easy for the network to overfit, failing to generalize to unseen instances of the same classes or to completely different objects. Indeed, when training directly on DAVIS, without pre-training, the model performed poorly, a clear indication that this stage of training is essential to achieve competitive results.

IMPORTANCE OF THE LOCALIZATION Finally, we investigated the contribution of each modulator network on the quality of the segmentation by running two tests at inference time. Both the Visual and the Spatial Modulator are provided with the ground truth of the previous frame by an oracle. Note that, normally, the visual modulation is computed only once for the entire sequence, where the VM receives as input the first frame cropped around its segmentation. We studied the impact of changes in the appearance of the object of interest by recomputing the coefficients of the visual modulation at each time frame, given the oracle. Moreover, the Spatial Modulator uses the ground truth segmentation rather than the previous prediction to generate the next spatial guide, providing the exact location of the object in the previous frame. The results of this study are shown in Figure 5.2. As could be expected both the configurations improve our results

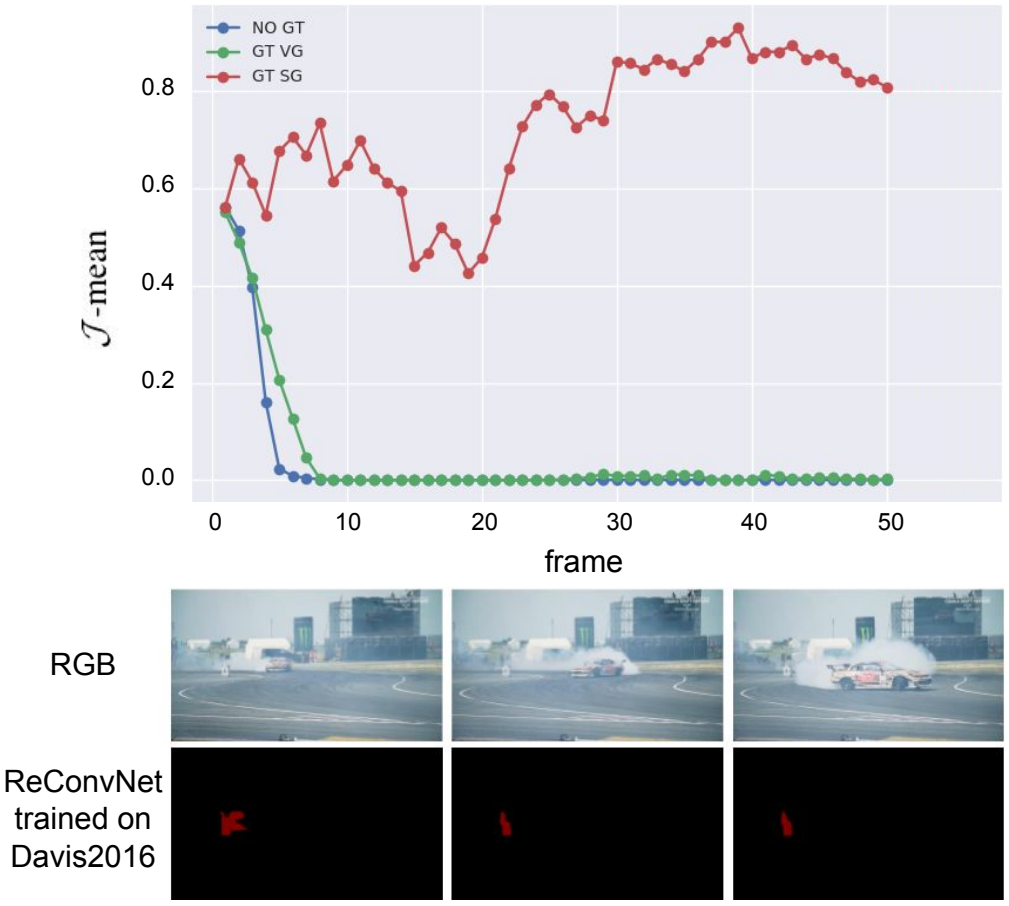


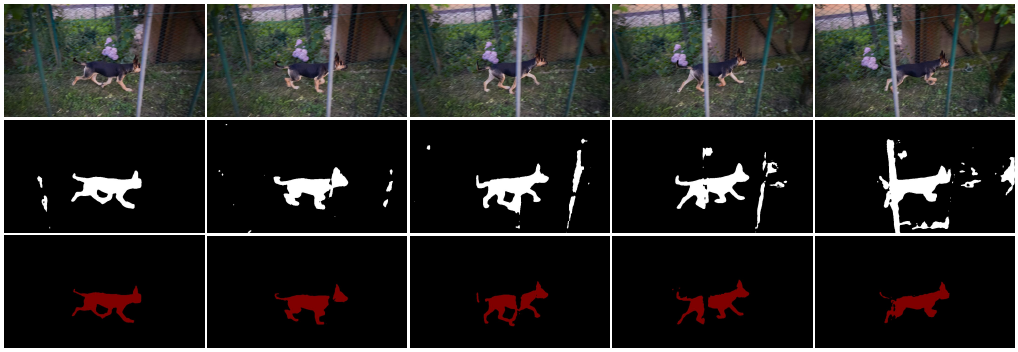
Figure 5.3: Importance of a strong spatial prior. Top: performance of ReConvNet on a hard sequence of DAVIS2016. Comparison of a trained model (blue) with the case when the VM (green) and the SM (red) receive as input from an oracle the true segmentation of the object in the previous frame. The prior on the actual position of the object proved to be more important than compensating for changes in its appearance. Bottom: Predictions and inputs for the failing case analyzed in the plot above: ReConvNet is not able to track the object.

consistently. It is interesting to notice that having a precise object localization is more effective in average than knowing the ground truth appearance of the object of interest. This applies to those videos in which the object significantly changes its appearance over time: the visual guide given in the first frame is not very informative for the segmentation network while it is more important to have a stronger spatial prior. Section 5.5 shows the results of the two inferences on a specific video of the DAVIS2016 validation set. In this case the object moves very fast between frames so that the localization has to be very precise to track the object correctly and, if this is not the case, the visual modulator is not able to guide the segmentation network even if the ground truth appearance of the object is known. This suggests that the network relies heavily on the spatial bias imposed by the SM and is not able to recover when this is not accurate.

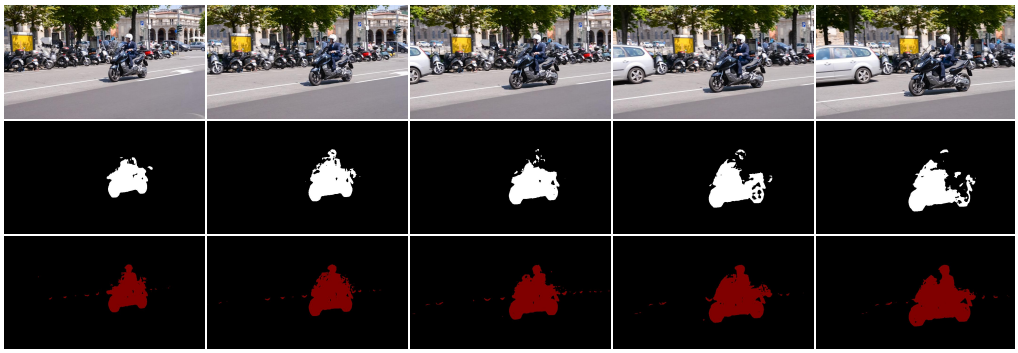
5.6 CONCLUSION

We presented *ReConvNet*, a powerful and efficient recurrent convolutional model to perform semi-supervised video object segmentation. The model is able to learn spatio-temporal features that self-adapt to focus on the object of interest without the need of extra fine-tuning at inference time. *ReConvNet* outperforms the baseline by a considerable margin, proving the effectiveness of incorporating temporal consistency into the model. Our results reinforce the conjecture that features modulation is a valid approach to semi-supervised video object segmentation. We plan to perform a more in-depth analysis of the interaction between the temporal components and the features modulation, since we believe it is crucial to better understand the potential of the proposed model.

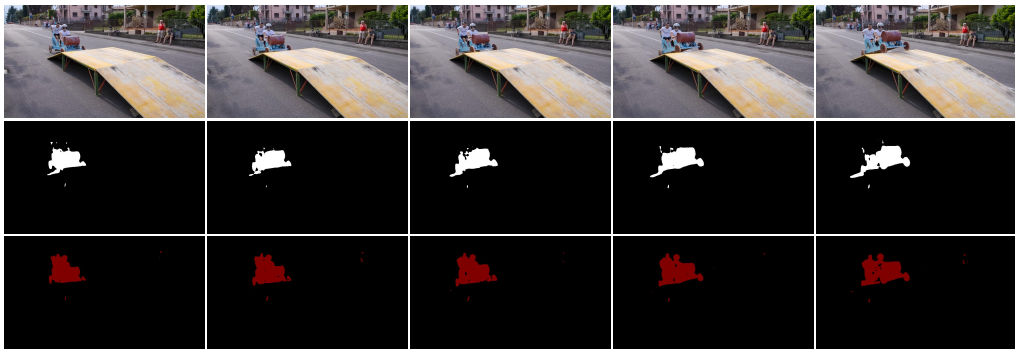
(a) drift



(b) libby



(c) scooter



(d) soapboax

Figure 5.4: Single Object. The recurrent components of our model (red) play an important role in ensuring a better temporal consistency of the segmentation masks, resulting in more temporally coherent and cleaner masks than the OSMN baseline (white).

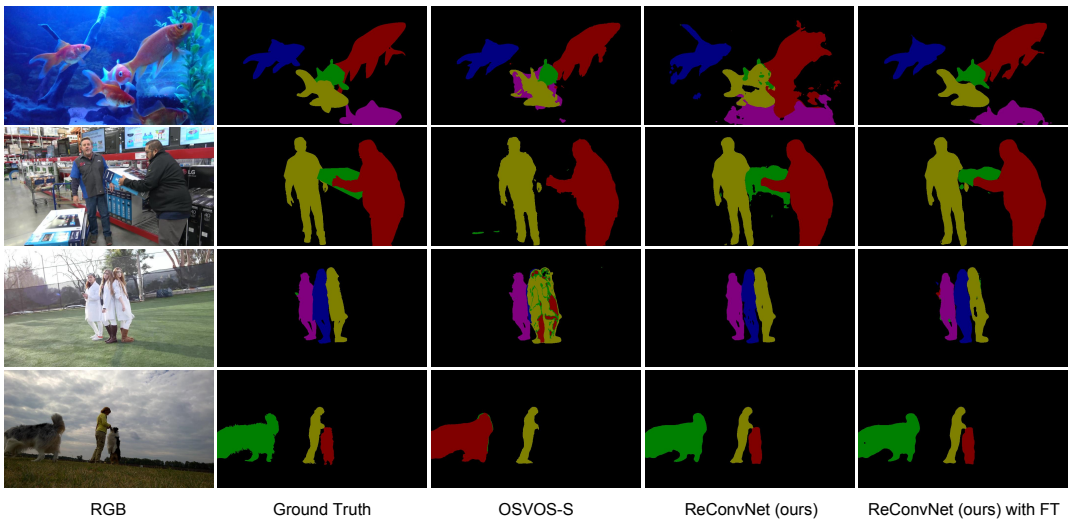


Figure 5.5: Multiple Objects. A comparison of the state of the art OSVOS-S based on fine-tuning with both the basic ReConvNet model and the fine-tuned one. It is possible to appreciate that ReConvNet shows a good level of accuracy without the need of an expensive online finetuning that, however, helps in videos with a substantial number of interacting objects. This is exemplified by the first row, it can be seen that finetuning allows to better discriminate between several instances of the same object with higher precision.

6 CONCLUSIONS AND FUTURE WORK

In this thesis, we presented advances in the field of Representation Learning for visual data with a focus on iterative and recurrent processing.

Input-Output Stable Architectures for Iterative Computation

In Chapter 3 we presented a theoretical analysis of Residual Networks from a dynamical system perspective, reinterpreting these architectures as forward Euler discretization of ODEs. We focused on the stability properties of these systems to construct NAIS-Net, a cascade of stable architectures that can be unrolled until convergence to an equilibrium point, showing that they can perform iterative inference with shared weights.

Our proposed parametrization for fully-connected and convolutional layers is an efficient way to ensure the stability of the system. At each step of gradient descent, the network's weights are projected back to the stability region if the stability constraint is not satisfied. By imposing a hard constraint, we can ensure convergence when the system is unrolled. However, a hard reprojection could hinder the representational power of the network and potentially be hard to optimize.

Anil et al. (2019) observed that it is challenging to achieve similar performance as unconstrained networks while provably enforcing a Lipschitz constraint, identifying gradient attenuation issues during training. To scale NAIS-Net to more challenging tasks such as ImageNet (Deng et al., 2009) classification could require relaxing the stability constraint introducing penalty terms (Gulrajani et al., 2017; Miyato et al., 2018; Yoshida and Miyato, 2017) to trade-off the performance on the task and the convergence guarantees of the network, or to use more versatile constrained optimization techniques such as Augmented Lagrangian (Rocha and Fernandes, 2010; Wang and Spall, 2003).

An important characteristic of NAIS-Net is that the input-output mapping of each block is guaranteed to be bounded by a constant, namely, it is *Lipschitz*. There has been much interest in training neural networks with known upper bounds on their Lipschitz constants as they can provide provable robustness against adversarial examples (Cisse et al., 2017; Tsuzuku et al., 2018) and improve generalization bounds (Sokolić et al., 2017). In Section 3.6.2 we showed that NAIS-Net has, indeed, a better generalization gap in image classification tasks on CIFAR-10/100. The degree to which NAIS-Net is robust to adversarial perturbations remains to be tested in future work.

Another exciting area of research is investigating the performance of NAIS-Net as a generative model. The iterative nature of residual architectures can be leveraged to construct reversible architectures that perform well in both discriminative and generative settings (Gomez et al., 2017; Jacobsen et al., 2018). For instance, Behrmann et al. (2019) enforce the invertibility of ResNets by introducing a normalization term during training such that the layers are contractive, namely, the Lipschitz constant is less than one. Similarly, it would be interesting to explore whether it is possible to construct an invertible version of NAIS-Net to learn flow-based (Rezende and Mohamed, 2015) generative models.

Learning Recurrent Representations for event-based data

In Chapter 4 we proposed Matrix-LSTM, a novel representation for event-based data. Learning effective representation to replace hand-designed features allowed machine learning-based methods to achieve impressive performance on many vision tasks. As event cameras are slowly but steadily gaining popularity in the context of robotics, autonomous driving, and drone navigation, it is becoming increasingly important to provide these systems with the same capabilities as their frame-based counterparts while retaining the advantages of event representations. Event cameras, in fact, offer significant advantages over conventional RGB cameras, such as a very high dynamic range, no motion blur, and latency in the order of microseconds. However, the performance of event-based systems on many vision tasks is still low compared to conventional frame-based solutions, warranting the development of new methods to process sparse and asynchronous data.

We designed a mechanism to efficiently apply a Long Short-Term Memory (LSTM) network as a convolutional filter over the 2D stream of events produced by event-based cameras. The LSTM cell learns to accumulate pixel information through time and build 2D event representations. The reconstruction mechanism is end-to-end differentiable, which allows training it jointly with state-of-the-art frame-based architectures in order to learn frame reconstructions explicitly tailored for the task at hand. We showed that the proposed mechanism can be successfully used to build static and dynamic representations for object recognition and optical flow prediction. As a future line of research, we plan to explore the use of Matrix-LSTM for more complex tasks such as gray-scale frame reconstruction (Rebecq et al., 2019), ego-motion, and depth estimation (Ye et al., 2018; Zhu et al., 2019a). We are currently testing Matrix-LSTM on object detection, using the recently released dataset by Prophesee (Perot et al., 2020). Images have been collected at high spatial resolution posing the important challenge of scaling Matrix-LSTM's accumulation mechanism to deal with thousands of events per second, balancing efficiency and performance. One interesting future direction is to investigate aggregation or pooling strategies for the events

in the same receptive field, enabling the use of larger kernels to improve efficiency.

One-Shot Video Object Segmentation with Conditional Modulations

In Chapter 5 we focused on the problem of learning how to adapt to new tasks and developed a model for One-Shot Video Object Segmentation. Semi-supervised video object segmentation models are critical components of multi-object tracking systems, especially in robotics applications. For example, it is often desirable for a robot to learn rapidly and on the fly the appearance of a specific person or object it has to follow or interact with. This can be extended to multiple people/objects tracking, where the model should be fast-adapted to a particular instance on the scene. Learning how to adapt to new appearances is also crucial for object-robot interactions, where a robot should grasp or manipulate a new object.

We cast the problem in the meta-learning setting, producing the weights of a *"fast"'* segmentation network via a secondary *"slow"* network conditioned on the object of interest. We show the importance of modeling the temporal correlations across frames by enhancing a base convolutional model with recurrent convLSTM cells.

The proposed approach showed promising results both in single-object and multi-object segmentation tasks, proving capable of adapting to new objects by observing a single annotated frame. However, when multiple instances of the same objects are present in the scene, modulation does not seem to be enough for adaptation, requiring extra steps of fine-tuning after training. More powerful adaptation capabilities can also be obtained by introducing recurrent units into the modulator networks, although recurrent modulations are usually sensitive to hyper-parameters and unstable during training (Ha et al., 2017). Another approach would be to integrate modulation networks and gradient-based meta learning (Finn et al., 2017) into the same training step to learn task-specific initializations of the weights that require only a few fine-tuning updates to yield optimal performance. Other improvements could be obtained by introducing better task embeddings. Currently, the modulation network processes the cropped annotated frame rescaled to a fixed size. This could introduce artifacts and deformations. A multi-scale task embedding that preserves the object's details and its aspect ratio would be helpful for producing accurate segmentation of small instances.

Finally, we showed the importance of having a precise localization of the object in the frame, when the objects are partially occluded or in uncommon configurations, absolutely and/or in relation to other objects. One of the future efforts will be to investigate the use of more accurate spatial priors considering explicit motion information such as optical flow (Zhu et al., 2018b), and the integration with state-of-the-art detection and tracking systems (He et al., 2017; Lin et al., 2017; Redmon and Farhadi, 2018).

Part I

APPENDIX

A DYNAMICAL SYSTEMS AND STABILITY BACKGROUND

A.1 LINEAR ALGEBRA ELEMENTS

Notation

- $\|\cdot\|$ is used to denote a *suitable* matrix norm. This norm will be characterized specifically on a case by case basis. The same norm will be used consistently throughout definitions, assumptions and proofs.
- $\mathbf{A}_{i\bullet}$ is used to denote the i-th row of a matrix $\mathbf{A}$.

Lemma 2. *(Eigenvalue shift) Consider two matrices $\mathbf{A} \in \mathbb{C}^{n\times n}$, and $\mathbf{C} = c\mathbf{I}+\mathbf{A}$ with c being a complex scalar. If λ is an eigenvalue of $\mathbf{A}$ then $c+\lambda$ is an eigenvalue of $\mathbf{C}$.*

Proof. Given any eigenvalues λ of $\mathbf{A}$ with corresponding eigenvector v we have that:

$$\begin{aligned} \lambda\mathbf{v} = \mathbf{A}\mathbf{v} \Leftrightarrow & (\lambda + c)\mathbf{v} \\ & = \lambda\mathbf{v} + c\mathbf{v} \\ & = \mathbf{A}\mathbf{v} + c\mathbf{v} = \mathbf{A}\mathbf{v} + c\mathbf{I}\mathbf{v} \\ & = (\mathbf{A} + c\mathbf{I})v = \mathbf{C}\mathbf{v} \end{aligned} \tag{A.1}$$

□

A.2 STABILITY DEFINITIONS FOR TIED WEIGHTS

This section provides a summary of definitions borrowed from control theory that are used to describe and derive our main result. The following definitions have been adapted from Gallieri (2016) and refer to the general dynamical system:

$$\mathbf{x}^+ = f(\mathbf{x}, \mathbf{u}). \tag{A.2}$$

Since then stability of a cascade of dynamical systems is stable if and only if all its blocks are stable (Khalil, 2014), we can simply focus on stability of the unroll of a single block.

Relevant Sets and Operators

Denote the slope of the activation function vector, $\sigma(\Delta\mathbf{x}(k))$, as the diagonal matrix, $\sigma^{'}(\Delta\mathbf{x}(k))$, with entries:

$$\sigma_{ii}^{'}(\Delta\mathbf{x}(k)) = \frac{\partial\sigma_i(\Delta\mathbf{x}(k))}{\partial\Delta\mathbf{x}_i(k)}. \tag{A.3}$$

where $\Delta\mathbf{x}(k)$ is the argument of the activation function $\sigma(\cdot)$.

The following definitions will be used later to obtain the stability results, where $0 < \underline{\sigma} \ll 1$:

$$\begin{aligned}
\mathcal{P}_i &= \{(\mathbf{x},\mathbf{u}) \in \mathbb{R}^n \times \mathbb{R}^m : \sigma_{ii}^{'}(\mathbf{x},\mathbf{u}) \geq \underline{\sigma}\},\\
\mathcal{P} &= \{(\mathbf{x},\mathbf{u}) \in \mathbb{R}^n \times \mathbb{R}^m : \sigma_{ii}^{'}(\mathbf{x},\mathbf{u}) \geq \underline{\sigma}, \forall i\},\\
\mathcal{N}_i &= \mathcal{P} \cup \{(\mathbf{x},\mathbf{u}) \in \mathbb{R}^n \times \mathbb{R}^m : \sigma_{ii}^{'}(\mathbf{x},\mathbf{u}) \in [0,\underline{\sigma})\},\\
\mathcal{N} &= \{(\mathbf{x},\mathbf{u}) \in \mathbb{R}^n \times \mathbb{R}^m : \sigma_{ii}^{'}(\mathbf{x},\mathbf{u}) \in [0,\underline{\sigma}), \forall i\},
\end{aligned} \tag{A.4}$$

In particular, the set $\mathcal{P}$ is such that the activation function is not saturated as its derivative has a non-zero lower bound.

Non-autonomuous Behaviour Set

The following set will be consider throughout the chapter.

Definition A.2.1. *(Non-autonomous behaviour set) The set $\mathcal{P}$ is referred to as the set of* fully non-autonomous *behaviour in the extended state-input space, and its set-projection over x, namely,*

$$\pi_{\mathbf{x}}(\mathcal{P}) = \{\mathbf{x} \in \mathbb{R}^n : \exists\mathbf{u} \in \mathbb{R}^m,\ (\mathbf{x},\mathbf{u}) \in \mathcal{P}\}, \tag{A.5}$$

is the set of fully non-autonomous behaviour in the state space. This is the only set in which every output dimension of the ResNet with input skip connection can be directly influenced by the input, given a non-zero[1] matrix $\mathbf{B}$.

Note that, for a *tanh* activation, then we simply have that $\mathcal{P} \subseteq \mathbb{R}^{n+m}$ (with $\mathcal{P} \to \mathbb{R}^{n+m}$ for $\overline{\sigma}^{'} \to 0$). For a ReLU activation, on the other hand, for each layer k we have:

$$\mathcal{P} = \mathcal{P}(k) = \{(\mathbf{x},\mathbf{u}) \in \mathbb{R}^n \times \mathbb{R}^m :\ \mathbf{A}(k)\mathbf{x} + \mathbf{B}(k)\mathbf{u} + \mathbf{b}(k) > 0\}. \tag{A.6}$$

A.2.1 Describing Functions

The following functions are instrumental to describe the desired behaviour of the network output at each layer or time step.

1 The concept of controllability is not introduced here. In the case of deep networks we just need $\mathbf{B}$ to be non-zero to provide input skip connections. For the general case of time-series identification and control, please refer to the definitions in Sontag (1998).

Definition A.2.2. *($\mathcal{K}$-function) A continuous function $\alpha : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is said to be a $\mathcal{K}$-function ($\alpha \in \mathcal{K}$) if it is strictly increasing, with $\alpha(0) = 0$.*

Definition A.2.3. *($\mathcal{K}_\infty$-function) A continuous function $\alpha : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is said to be a $\mathcal{K}_\infty$-function ($\alpha \in \mathcal{K}_\infty$) if it is a $\mathcal{K}$-function and if it is radially unbounded, that is $\alpha(r) \to \infty$ as $r \to \infty$.*

Definition A.2.4. *($\mathcal{KL}$-function) A continuous function $\beta : \mathbb{R}^2_{\geq 0} \to \mathbb{R}_{\geq 0}$ is said to be a $\mathcal{KL}$-function ($\beta \in \mathcal{KL}$) if it is a $\mathcal{K}$-function in its first argument, it is positive definite and non- increasing in the second argument, and if $\beta(r, t) \to 0$ as $t \to \infty$.*

Invariance, Stability and Robustness

The following stability definitions are given for *time-invariant systems*, that correspond to the unrolling of deep residual networks with *tied (shared) weights*. The same definitions can also be generalised to the case of *untied weights*, or *time-varying* systems by considering worst case conditions over the layer (time) index k. In this case the properties are said to hold *uniformly* for all $k \geq 0$. See Appendix B in the appendix for more details.

Definition A.2.5. *(Positively Invariant Set) A set $\mathcal{X} \subseteq \mathbb{R}^n$ is said to be positively invariant (PI) for a dynamical system under an input $\mathbf{u} \in \mathcal{U} \subseteq \mathbb{R}^n$ if*

$$f(\mathbf{x}, \mathbf{u}) \in \mathcal{X}, \ \forall \mathbf{x} \in \mathcal{X}. \tag{A.7}$$

Definition A.2.6. *(Robustly Positively Invariant Set) The set $\mathcal{X} \subseteq \mathbb{R}^n$ is said to be* robustly positively invariant *(RPI) to additive input perturbations $w \in \mathcal{W}$ if $\mathcal{X}$ is PI for any input $\tilde{\mathbf{u}} = \mathbf{u} + \mathbf{w}$, $\mathbf{u} \in \mathcal{U}, \forall \mathbf{w} \in \mathcal{W}$.*

Definition A.2.7. *(Asymptotic Stability) The system Eq. (A.2) is called Globally Asymptotically Stable around its equilibrium point $\bar{\mathbf{x}}$ if it satisfies the following two conditions:*

1. ***Stability***. Given any $\epsilon > 0$, $\exists \delta_1 > 0$ such that if $\|\mathbf{x}(t_0) - \bar{\mathbf{x}}\| < \delta_1$, then $\|\mathbf{x}(t) - \bar{\mathbf{x}}\| < \epsilon, \forall t > t_0$.
2. **Attractivity**. $\exists \delta_2 > 0$ such that if $\|\mathbf{x}(t_0) - \bar{\mathbf{x}}\| < \delta_2$, then $\mathbf{x}(t) \to \bar{\mathbf{x}}$ as $t \to \infty$.

If only the first condition is satisfied, then the system is globally stable. *If both conditions are satisfied only for some $\epsilon(\mathbf{x}(0)) > 0$ then the stability properties hold only locally and the system is said to be* locally asymptotically stable.

Local stability in a PI set $\mathcal{X}$ is equivalent to the existence of a $\mathcal{KL}$-function β and a finite constant $\delta \geq 0$ such that:

$$\|\mathbf{x}(k) - \bar{\mathbf{x}}\| \leq \beta(\|\mathbf{x}(0) - \bar{\mathbf{x}}\|, k) + \delta, \ \forall \mathbf{x}(0) \in \mathcal{X}, \ k \geq 0. \tag{A.8}$$

If $\delta = 0$ then the system is asymptotically *stable. If the positively-invariant set is $\mathcal{X} = \mathbb{R}^n$ then stability holds* globally.

Define the system output as $\mathbf{y}(k) = \psi(\mathbf{x}(k))$, where ψ is a continuous, Lipschitz function. Input-to-Output stability provides a natural extension of asymptotic stability to systems with inputs or additive uncertainty[2].

Definition A.2.8. *(Input-Output (practical) Stability) Given an RPI set $\mathcal{X}$, a constant nominal input $\overline{\mathbf{u}}$ and a nominal steady state $\overline{\mathbf{x}}(\overline{\mathbf{u}}) \in \mathcal{X}$ such that $\overline{\mathbf{y}} = \psi(\overline{\mathbf{x}})$, the system Eq. (A.2) is said to be* input-output (practically) stable *to* bounded additive input perturbations *(IOpS) in $\mathcal{X}$ if there exists a $\mathcal{KL}$-function β and a $\mathcal{K}_\infty$ function γ and a constant $\zeta > 0$:*

$$\begin{aligned} \|\mathbf{y}(k) - \overline{\mathbf{y}}\| \leq \beta(\|\mathbf{y}(0) - \overline{\mathbf{y}}\|, k) + \gamma(\|\mathbf{w}\|) + \zeta, \ \forall \mathbf{x}(0) \in \mathcal{X}, \\ \mathbf{u} = \overline{\mathbf{u}} + \mathbf{w}, \ \overline{\mathbf{u}} \in \mathcal{U}, \ \forall \mathbf{w} \in \mathcal{W}, \ \forall k \geq 0. \end{aligned} \tag{A.9}$$

Definition A.2.9. *(Input-Output (Robust) Stability) Given an RPI set $\mathcal{X}$, a constant nominal input $\overline{\mathbf{u}}$ and a nominal steady state $\overline{\mathbf{x}}(\overline{\mathbf{u}}) \in \mathcal{X}$ such that $\overline{\mathbf{y}} = \psi(\overline{\mathbf{x}})$, the system Eq. (A.2) is said to be* input-output (robustly) stable *to* bounded additive input perturbations *(IOS) in $\mathcal{X}$ if there exists a $\mathcal{KL}$-function β and a $\mathcal{K}_\infty$ function γ such that:*

$$\begin{aligned} \|\mathbf{y}(k) - \overline{\mathbf{y}}\| \leq \beta(\|\mathbf{y}(0) - \overline{\mathbf{y}}\|, k) + \gamma(\|\mathbf{w}\|), \ \forall \mathbf{x}(0) \in \mathcal{X}, \\ \mathbf{u} = \overline{\mathbf{u}} + \mathbf{w}, \ \overline{\mathbf{u}} \in \mathcal{U}, \ \forall \mathbf{w} \in \mathcal{W}, \ \forall k \geq 0. \end{aligned} \tag{A.10}$$

Definition A.2.10. *(Input-Output* incremental *Stability) Given a pair of initial conditions $\{\mathbf{x}(0), \ \overline{\mathbf{x}}(0)\}$ and constant inputs $\{\mathbf{u}, \ \overline{\mathbf{u}}\}$, with $\mathbf{y} = \mathbf{x}$, system Eq. (A.2) is said to be Globally* input-output incrementally stable *(δ-IOS) if there exists a $\mathcal{KL}$-function β and a $\mathcal{K}_\infty$ function γ such that:*

$$\begin{aligned} \|\mathbf{y}(k) - \overline{\mathbf{y}}(k)\| \leq \beta(\|\mathbf{y}(0) - \overline{\mathbf{y}}(0)\|, k) + \gamma(\|\mathbf{u} - \overline{\mathbf{u}}\|), \\ \forall \{\mathbf{x}(0), \ \overline{\mathbf{x}}(0)\} \in \mathbb{R}^{2n}, \ \{\mathbf{u}, \ \overline{\mathbf{u}}\} \in \mathcal{U}^2, \forall k \geq 0. \end{aligned} \tag{A.11}$$

2 Here we will consider only the simple case of $\mathbf{y}(k) = \mathbf{x}(k)$, therefore we can simply use notions of Input-to-State Stability (ISS).

B NAIS-NET WITH UNTIED WEIGHTS

B.1 PROPOSED NETWORK WITH UNTIED WEIGHTS

The proposed network architecture with input skip connections and the robust stability results can be extended to the *untied weight* case. In particular, a single NAIS-Net block is analysed, where the weights are not shared throughout the unroll.

B.1.1 Fully Connected Layers

Consider the following ResNet with input skip connections and untied weights:

$$\begin{aligned} \mathbf{x}(k+1) &= f(\mathbf{x}(k), \mathbf{u}, k) \\ &= \mathbf{x}(k) + h\sigma\Big(\mathbf{A}(k)\mathbf{x}(k) + \mathbf{B}(k)\mathbf{u} + \mathbf{b}(k)\Big), \end{aligned} \tag{B.1}$$

where k indicates the layer, u is the input data, $h > 0$, and f is a continuous, differentiable function with bounded slope. The activation operator σ is a vector of (element-wise) instances of a non-linear activation function. In the case of tied (shared) weights, the DNN Eq. (B.1) can be seen as a finite unroll of an RNN, where the layer index k becomes a time index and the input is passed through the RNN at each time steps. This is fundamentally a linear difference equation also known as a *discrete time dynamic system*. The same can be said for the untied case with the difference that here the weights of the RNN will not be the same at each time step, inducing a *time-varying dynamical system*.

B.1.2 Convolutional Layers

For convolutional networks, the proposed layer architecture can be extended as:

$$\begin{aligned} \mathbf{X}(k+1) &= F(\mathbf{X}(k), \mathbf{U}, k) \\ &= \mathbf{X}(k) + h\sigma\Big(\mathbf{C}(k) * \mathbf{X}(k) + \mathbf{D}(k) * \mathbf{U} + \mathbf{E}(k)\Big), \end{aligned} \tag{B.2}$$

where $X^{(i)}(k)$, is the layer state matrix for channel i, while $\mathbf{U}^{(j)}$, is the layer input data matrix for channel j (where an appropriate zero padding has been applied) at layer k. An equivalent representation to Eq. (B.2), for a given layer k, can be computed in a similar way as done for the tied weight case in

Appendix D.2. In particular, denote the matrix entries for the filter tensors $\mathbf{C}(k)$ and $\mathbf{D}(k)$ and $\mathbf{E}(k)$ as follows: $\mathbf{C}_{(i)}^{(c)}(k)$ as the state convolution filter from state channel i to state channel c, and $\mathbf{D}_{(j)}^{(c)}(k)$ is the input convolution filter from input channel j to state channel c, and $\mathbf{E}^{(c)}(k)$ is a bias matrix for the state channel c.

Once again, convolutional layers can be analysed in a similar way to fully-connected layers, by means of the following vectorised representation:

$$\mathbf{x}(k+1) = \mathbf{x}(k) + h\sigma\big(\mathbf{A}(k)\mathbf{x}(k) + \mathbf{B}(k)\mathbf{u} + \mathbf{b}(k)\big). \tag{B.3}$$

By means of the vectorised representation Eq. (B.3), the theoretical results proposed in this section will hold for both fully connected and convolutional layers.

B.2 NON-AUTONOMOUS SET

Recall that, for a *tanh* activation, for each layer k we have a different set $\mathcal{P}(k) \subseteq \mathbb{R}^{n+m}$ (with $\mathcal{P}(k) = \mathbb{R}^{n+m}$ for $\epsilon \to 0$). For ReLU activation, we have instead:

$$\mathcal{P} = \mathcal{P}(k) = \{(\mathbf{x}, \mathbf{u}) \in \mathbb{R}^n \times \mathbb{R}^m : \ \mathbf{A}(k)\mathbf{x} + \mathbf{B}(k)\mathbf{u} + \mathbf{b}(k) > 0\}. \tag{B.4}$$

B.3 STABILITY DEFINITIONS FOR UNTIED WEIGHTS

For the case of untied weights, let us consider the origin as a reference point ($\overline{\mathbf{x}} = 0$, $\overline{\mathbf{u}} = 0$) as no other steady state is possible without assuming convergence for $\mathbf{A}(k)$, $\mathbf{B}(k)$. This is true if $\mathbf{u} = 0$ and if $\mathbf{b}(k) = 0, \forall k \geq \bar{k} \geq 0$. The following definition is given for stability that is verified *uniformly* with respect to the changing weights:

Definition B.3.1. *(Uniform Stability and Uniform Robustness) Consider* $\overline{\mathbf{x}} = 0$ *and* $\overline{\mathbf{u}} = 0$. *The network origin is said to be* uniformly *asymptotically or simply uniformly stable and, respectively, uniformly practically stable (IOpS), uniformly Input-Output Stable (IOS) or uniformly incrementally stable (δ-IOS) if, respectively, Definition A.2.7, A.2.8, A.2.9 and A.2.10 hold with a unique set of describing functions,* β, γ, ζ *for all possible values of the layer specific weights,* $\mathbf{A}(k)$, $\mathbf{B}(k)$, $\mathbf{b}(k)$, $\forall k \geq 0$.

B.4 JACOBIAN CONDITION FOR STABILITY

The state transfer Jacobian for untied weights is:

$$\mathbf{J}(\mathbf{x}(k), \mathbf{u}, k) = \frac{\partial f(\mathbf{x}(k), \mathbf{u}, k)}{\partial \mathbf{x}(k)} = \mathbf{I} + h\sigma^{'}\left(\Delta \mathbf{x}(k)\right) \mathbf{A}(k). \tag{B.5}$$

The following assumption extends our results to the untied weight case:

Condition 2. *For any $\overline{\sigma}^{'} > 0$, the Jacobian satisfies:*

$$\overline{\rho} = \sup_{(\mathbf{x},\mathbf{u})\in\mathcal{P}} \sup_{k} \rho\left(\mathbf{J}(\mathbf{x}(k), \mathbf{u}, k)\right) < 1, \tag{B.6}$$

where $\rho(\cdot)$ is the spectral radius.

Condition Eq. (B.6) can be enforced during training for each layer using the procedures presented in the paper.

B.5 STABILITY RESULT FOR UNTIED WEIGHTS

Recall that we have taken the origin as the reference equilibrium point, namely, $\overline{\mathbf{x}} = 0$ is a steady state if $\overline{\mathbf{u}} = 0$ and if $\mathbf{b}(k) = 0, \forall k \geq \overline{k} \geq 0$. Without loss of generality, we will assume $\mathbf{b}(k) = 0, \forall k$ and treat u as a disturbance, $\mathbf{u} = \mathbf{w}$, for the robust case. The following result is obtained:

Theorem 4. *(Uniform stability for untied weights)*

If Condition 2 holds, then NAIS–net with untied weights and with tanh activation is Globally Uniformly Stable. In other words there is a set $\overline{\mathcal{X}}$ that is an ultimate bound, namely:

$$\mathbf{x}(k) \to \overline{\mathcal{X}} \subseteq \mathbb{R}^n, \ \forall \mathbf{x}(0) \in \mathcal{X} \subseteq \mathbb{R}^n. \tag{B.7}$$

The describing functions are:

$$\begin{aligned} &\beta(\|\mathbf{x}\|, k) = \overline{\rho}^{\,k}\|\mathbf{x}\|, \qquad \gamma(\|\mathbf{w}\|) = h\frac{\|\overline{\mathbf{B}}\|}{(1-\overline{\rho}\,)}\|\mathbf{w}\|, \\ &\overline{\mathbf{B}} = \sup_{k} \|\mathbf{B}(k)\|, \ \overline{\rho} < 1, \end{aligned} \tag{B.8}$$

where $\|\cdot\|$ is the matrix norm providing the tightest bound to the left-hand side of Eq. (B.6), where $\overline{\rho}$ is defined.

In particular, we have:

1. *If the activation is tanh then the network is Globally Uniformly Input-Output robustly Stable for any input perturbation $\mathbf{w} \in \mathcal{W} = \mathbb{R}^m$. Under no input actions, namely if $\mathbf{u} = 0$, then the origin is Globally Asymptotically Stable. If*

$\mathbf{u} = \mathbf{w} \in \mathcal{W}$ where $\mathcal{W}$ is the norm ball of radius μ, then the following set is RPI:

$$\mathcal{X} = \left\{ \mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\| \leq \bar{r} = \frac{h\|\overline{\mathbf{B}}\|}{1-\overline{\rho}}\mu \right\}. \tag{B.9}$$

2. *If the activation is ReLU then the network is Globally Uniformly Incrementally IO Stable for input perturbations in a compact $\mathcal{W} \subset \mathbb{R}^m$.*

   The describing function are given by Eq. (B.8) and the constant term is $\zeta = \frac{\bar{r}}{(1-\overline{\rho})}$, where $\bar{r}$ is the norm ball radius for the initial condition, namely,

$$\mathcal{X} = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}(0) - \bar{\mathbf{x}}(0)\| \leq \bar{r}\}. \tag{B.10}$$

   This set is positively invariant under no input action.

   If $\mathbf{u} = \mathbf{w} \in \mathcal{W}$ where $\mathcal{W}$ is the norm ball of radius μ, then the state delta-converges to the following ultimate bound:

$$\overline{\mathcal{X}} = \left\{ \mathbf{x} \in \mathbb{R}^n : \|\mathbf{x} - \bar{\mathbf{x}}\| \leq \zeta + h\frac{\|\overline{\mathbf{B}}\|\mu}{(1-\overline{\rho})} \right\}. \tag{B.11}$$

Note that, if the network consists of a combination of fully connected and convolutional layers, then a single norm inequality with corresponding β and γ can be obtained by means of matrix norm identities. For instance, since for any norm we have that $\|\cdot\|_q \leq \alpha\|\cdot\|_p$, with $\alpha > 0$, one could consider the global describing function $\beta(\cdot,\cdot) = \alpha\beta_p(\cdot,\cdot)$. Similarly for γ.

C STABILITY PROOFS

One way to assess the stability of a system is by using the so-called *Lyapunov indirect method* (Khalil, 2014; Sontag, 1998; Strogatz, 2015), that consists in the analysis of the *linearized system* around an equilibrium. If the linearized system is stable, then the original system is also stable.

This also applies for linearizations around all possible trajectories if there is a single equilibrium point, as in our case. In the following, the bias term is sometimes omitted without loss of generality (one can add it to the input). Note that the proofs are also valid for RNNs with varying input sequences $\mathbf{u}(k)$, with asymptotic results for converging input sequences, $\mathbf{u}(k) \to \overline{\mathbf{u}}$. Recall that $\mathbf{y}(k) = \mathbf{x}(k)$ by definition and let us consider the case of $\mathbf{b}(k) = 0$, $\forall k$, without loss of generality as this can be also assumed as part of the input.

C.1 STABILITY PROOF FOR UNTIED WEIGHTS

The proposed results make use of the 1-step state transfer Jacobian Eq. (B.5). Note that this is not the same as the full input-to-output Jacobian, for instance as the one defined in Duvenaud et al. (2014). The full Jacobian will also contain the input to state map, given by Eq. (C.2), which does not affect stability. The input to state map will be used later on to investigate robustness as well as the asymptotic network behaviour. First, we will focus on stability, which is determined by the 1-step state transfer map. For the sake of brevity, we denote the layer t state Jacobian from Eq. (B.5) as:

$$\mathbf{J}(t) = \mathbf{I} + h\sigma'(\Delta\mathbf{x}(t))\mathbf{A}(t), \ \forall t \geq 0.$$

Define the discrete time convolution sum as:

$$\mathbf{y}_{\mathbf{u}}(k) = \sum_{t=0}^{k-1} \mathbf{H}(k-t)\mathbf{u}(t), \quad k > 0, \ y_{\mathbf{u}}(0) = 0.$$

The above represent the *forced response* of a linear time invariant (LTI) system, namely the response to an input from a zero initial condition, where $\mathbf{H}$ is the (in the LTI case stationary) impulse response.

Conversely, the *free response* of an autonomous system from a non-zero initial condition is given by:

$$\mathbf{y}_{\mathbf{x}_0}(k) = \left(\prod_{t=0}^{k-1} \mathbf{J}(t) \right) \mathbf{x}(0).$$

The free response tends to zero for an asymptotically stable system.

Considering linearized dynamics allows us to use the *superposition principle*, in other words, the linearized system response can be analysed as the sum of the free and forced response:

$$\mathbf{y}(k) = \mathbf{y}_{\mathbf{x}_0}(k) + \mathbf{y}_{\mathbf{u}(k)}.$$

Note that this is not true for the original network, but just for its linearization.

For the considered network, the forced response of the linearized (time varying) system *to an impulse* at time t, evaluated at time k, is given by:

$$\mathbf{H}(k,t) = \begin{cases} h\sigma'(\Delta\mathbf{x}(t))\mathbf{B}(t), & \text{if } k = t+1 \\ h\sigma'(\Delta\mathbf{x}(t))\mathbf{B}(t)\left(\prod_{l=t}^{k-2}\mathbf{J}(l+1)\right), & \forall k \geq t+2 \end{cases} \tag{C.1}$$

Therefore, the forced response of the linearized system is:

$$\begin{aligned} \mathbf{y}_{\mathbf{u}}(k) &= h\sigma'(\mathbf{x}(k-1),\mathbf{u}(k-1))\mathbf{B}(k-1)\mathbf{u}(k-1) \\ &\quad + \sum_{t=0}^{k-2}\left(\prod_{l=t}^{k-2}\mathbf{J}(l+1)\right)h\sigma'(\Delta\mathbf{x}(t))\mathbf{B}(t)\mathbf{u}(t) \\ &= \sum_{t=0}^{k-1}\mathbf{H}(k,t)\mathbf{u}(t). \end{aligned} \tag{C.2}$$

Note that:

$$\|\mathbf{H}(k,t)\| \leq h \sup_{(\mathbf{x},\mathbf{u})\in\mathcal{P},\, j} \|\mathbf{J}(\mathbf{x},\mathbf{u},j)\|^{(k-t-1)}\|\mathbf{B}(j)\|,\ \forall k,t \geq 0$$

since $\|\sigma'\| \leq 1$.

To prove our results, we will use the fact that the network with *tanh* or ReLU activation is *globally Lipschitz*, and that the activation functions have Lipschitz constant $M = 1$ with respect to any norm. This follows from the fact that:

$$\sup_{\Delta\mathbf{x}\in\mathbb{R}^n} \max_i |\sigma'_{ii}(\Delta\mathbf{x})| = 1.$$

This means that the trajectory norm can be upper bounded inside $\mathcal{P} \subseteq \mathbb{R}^n \times \mathbb{R}^m$ by means of:

$$\|f(\mathbf{x},\mathbf{u},k)\| \leq \sup_{(\mathbf{x},\mathbf{u})\in\mathcal{P}} \sup_k \|\mathbf{J}(\mathbf{x},\mathbf{u},k)\|\|\mathbf{x}\| + \sup_{(\mathbf{x},\mathbf{u})\in\mathcal{P}} \sup_k \sum_{t=0}^{k-1}\|\mathbf{H}(k,t)\|\|\mathbf{u}\| \tag{C.3}$$

For a non-zero steady state $\bar{\mathbf{x}}$ or a trajectory $\bar{\mathbf{x}}(k)$ starting from a different initial condition $\bar{\mathbf{x}}(0)$, we can define the *error function*:

$$\mathbf{e}(k) = \mathbf{x}(k) - \bar{\mathbf{x}}(k), \tag{C.4}$$

From the fact that the network equation is globally Lipschitz and that the steady states satisfy $f(\overline{\mathbf{x}}, \overline{\mathbf{u}}) = \overline{\mathbf{x}}$, we can use a single statement to show *convergence* for both architectures, respectively, with respect to a steady-state for tanh or to any other trajectory for ReLU. We will show that the function that maps $\mathbf{e}(k)$ into $\mathbf{e}(k+1)$ is also Lipschitz for all k:

$$\begin{aligned}\|\mathbf{e}(k+1)\| &= \\ &= \|f(\mathbf{x}, \overline{\mathbf{u}}) - \overline{\mathbf{x}}\| = \|f(\mathbf{x}, \overline{\mathbf{u}}) - f(\overline{\mathbf{x}}, \overline{\mathbf{u}})\| \\ &= \|\mathbf{x} + h\sigma(\mathbf{A}\mathbf{x} + \mathbf{B}\overline{\mathbf{u}} + b) - \overline{\mathbf{x}} - h\sigma(\mathbf{A}\overline{\mathbf{x}} + \mathbf{B}\overline{\mathbf{u}} + b)\| \\ &\leq \sup_{(\mathbf{x},\mathbf{u})\in\mathcal{P}} \sup_{t} \left(\|\mathbf{I} + h\sigma^{'}(\Delta\mathbf{x}(t))\mathbf{A}\| \|\mathbf{x} - \overline{\mathbf{x}}\| \right) + h\|\mathbf{B}\overline{\mathbf{u}} - \mathbf{B}\overline{\mathbf{u}} + \mathbf{b} - \mathbf{b}\| \\ &= \sup_{(\mathbf{x},\mathbf{u})\in\mathcal{P}} \sup_{t} \left(\|\mathbf{I} + h\sigma^{'}(\Delta\mathbf{x}(t))\mathbf{A}\| \right) \|\mathbf{x} - \overline{\mathbf{x}}\|. \end{aligned} \tag{C.5}$$

Note that, at the next step, we also have:

$$\begin{aligned}\|\mathbf{e}(k+2)\| &= \|f(f(\mathbf{x}, \overline{\mathbf{u}}), \overline{\mathbf{u}}) - \overline{\mathbf{x}}\| = \|f(f(\mathbf{x}, \overline{\mathbf{u}}), \overline{\mathbf{u}}) - f(f(\overline{\mathbf{x}}, \overline{\mathbf{u}}), \overline{\mathbf{u}})\| \\ &\leq \sup_{(\mathbf{x},\mathbf{u})\in\mathcal{P}} \sup_{t} \left(\|\mathbf{I} + h\sigma^{'}(\Delta\mathbf{x}(t))\mathbf{A}\| \right) \|f(\mathbf{x}, \overline{\mathbf{u}}) - f(\overline{\mathbf{x}}, \overline{\mathbf{u}})\| + h\|\mathbf{B}\overline{\mathbf{u}} - B\overline{\mathbf{u}} + \mathbf{b} - \mathbf{b}\| \\ &= \sup_{(\mathbf{x},\mathbf{u})\in\mathcal{P}} \sup_{t} \left(\|\mathbf{I} + h\sigma^{'}(\Delta\mathbf{x}(t))\mathbf{A}\| \right) \|f(\mathbf{x}, \overline{\mathbf{u}}) - f(\overline{\mathbf{x}}, \overline{\mathbf{u}})\| \\ &= \sup_{(\mathbf{x},\mathbf{u})\in\mathcal{P}} \sup_{t} \left(\|\mathbf{I} + h\sigma^{'}(\Delta\mathbf{x}(t))\mathbf{A}\| \right) \|f(\mathbf{x}, \overline{\mathbf{u}}) - \overline{\mathbf{x}}\| \\ &\leq \sup_{(\mathbf{x},\mathbf{u})\in\mathcal{P}} \sup_{t} \left(\|\mathbf{I} + h\sigma^{'}(\Delta\mathbf{x}(t))\mathbf{A}\|^2 \right) \|\mathbf{x} - \overline{\mathbf{x}}\|. \end{aligned} \tag{C.6}$$

and therefore, *by induction* it also follows that the trajectory at time k satisfies:

$$\begin{aligned}\| \circ_{t=0}^{k} f(\mathbf{x}, \overline{\mathbf{u}}) - \overline{\mathbf{x}}\| &= \|f \circ f \circ \ldots f\left(f(\mathbf{x}, \overline{\mathbf{u}})\right) - \overline{\mathbf{x}}\| \\ \leq \left(\sup_{(\mathbf{x},\mathbf{u})\in\mathcal{P}} \sup_{t} \|\mathbf{I} + h\sigma^{'}(\Delta\mathbf{x}(t))\mathbf{A}\| \right)^{k} \|\mathbf{x} - \overline{\mathbf{x}}\| &= \overline{\rho}^{\,k} \|\mathbf{x} - \overline{\mathbf{x}}\| \end{aligned} \tag{C.7}$$

By definition of $\mathcal{P}$, from the above we have that, for *tanh* activation, the network trajectory with respect to an equilibrium point $\overline{\mathbf{x}}$, can be upper bounded in norm by the trajectory of the linearization while in $\mathcal{P}$. In the limit case of $\overline{\sigma}^{'} \to 0$ the bound becomes *global* as $\mathcal{P} \to \mathbb{R}^n \times \mathbb{R}^m$. On the other hand, for ReLU activation the upper bounds are valid only *locally*, in $\mathcal{P}$ iself. These considerations will be used to prove our main results.

Proof. (Theorem 4: Main result for untied weights)

In the untied weight case the network does not admit non-zero steady-states, as the matrices $\mathbf{A}(k)$ and $\mathbf{B}(k)$ are not assumed to converge with increasing k. Let us therefore consider the origin as our reference point, namely, $(\overline{\mathbf{x}} = 0, \overline{\mathbf{u}} = 0)$. Therefore, for the robust results we will consider the

input $\mathbf{u} = w$. The proof can now proceed by norm bounding the superposition of the free and forced response. Recall that $y(k) = \mathbf{x}(k)$ by definition and consider $\mathbf{b}(k) = 0,\ \forall k$, without loss of generality. Now, if $(\mathbf{x}, \mathbf{u}) \in \mathcal{P}$, for the linearized system we have the following:

$$\begin{aligned}\|\mathbf{x}(k) - \bar{\mathbf{x}}\| &= \|\mathbf{e}(k)\| \\ &\leq \sup_{(\mathbf{x},\mathbf{u})\in\mathcal{P}} \sup_j \|J(\mathbf{x},\mathbf{u},j)\|^k \|\mathbf{e}(0)\| + \sum_{t=0}^{k-1} \|\mathbf{H}(k,t)\|\,\|\mathbf{w}(t)\| \\ &\leq \overline{\rho}^{\,k}\|\mathbf{e}(0)\| + h\sum_{t=0}^{k-1} \overline{\rho}^{\,(k-t-1)} \sup_j \|\mathbf{B}(j)\|\|\mathbf{w}\| \\ &\leq \beta(\|\mathbf{e}(0)\|, k) + \gamma(\|\mathbf{w}\|).\end{aligned} \tag{C.8}$$

In the above, we have defined:

$$\begin{aligned}\beta(\|e\|, k) &= \overline{\rho}^{\,k}\|e\|, \\ \gamma(\|\mathbf{w}\|) &= h\frac{\|\overline{\mathbf{B}}\|}{(1-\overline{\rho}\,)}\|\mathbf{w}\|, \\ \overline{\mathbf{B}} &= \sup_j \|\mathbf{B}(j)\|,\ \overline{\rho}\ < 1.\end{aligned} \tag{C.9}$$

In Eq. (C.8), we have used the fact that if $\rho(J) < 1$ then there exist a suitable matrix norm $\|\cdot\|$ and a constant $\overline{\rho}\ < 1$ such that $\|\mathbf{J}\| \leq \overline{\rho}$. This stems directly from Theorem 5.6.12, page 298 of Horn and Johnson, 2012. In our case, $\overline{\rho}\ < 1$ is verified when $(\mathbf{x}, \mathbf{u}) \in \mathcal{P}$ since, from Condition 2, we have that $\sup_{(\mathbf{x},\mathbf{u})\in\mathcal{P}} \sup_j \rho\left(J(\mathbf{x},\mathbf{u},j)\right) < 1$. Outside the region $\mathcal{P}$, however, we need to consider the specific activation functions: for *tanh*, the region $\mathcal{P}$ can be taken to be any subset of the reals, therefore being outside this set as $\mathcal{P} \to \mathbb{R}^{n\times m}$ contradicts asymptotic stability. [1] For ReLU activation, being outside $\mathcal{P}$ means that (at least part of) the system is autonomous and therefore the network is simply stable as well as incrementally Input-Output practically stable.

Theorem statements are proven as follows:

1. Note that, the condition $(\mathbf{x}(t), \mathbf{u}) \notin \mathcal{P}$ is only possible for ReLU activations, since for *tanh* activation we have that $\mathcal{P} \to \mathbb{R}^{n+m}$ for $\overline{\sigma}^{'} \to 0$ and this would contradict stability result Eq. (C.8) inside the set. This means that Eq. (C.8) holds globally and therefore the considered network with tanh activations is Input-Output stable for a real-valued input $\mathbf{u}$. Same considerations apply for the robust case with an additive perturbation $w \in \mathbb{R}^m$.

1 Note that, when using *tanh*, in the limit case of $\overline{\sigma}^{'} = 0$, $\mathcal{P} = \mathbb{R}^{n+m}$ the system becomes simply stable. This is a small subtlety in our result for *tanh*, which could be defined as *almost global* or more formally *valid in every bounded subset of reals*. However, from the steady-state analysis (to follow) and the contraction result we can see that $\sigma^{'} = 0$ is highly unlikely to happen in practise.

In order to show the existence of a robust positively invariant set, consider the candidate set:

$$\mathcal{X} = \left\{ x \in \mathbb{R}^n : \|e\| = \|\mathbf{x} - \overline{\mathbf{x}}\| \leq \overline{r} = \frac{r}{1-\overline{\rho}} \right\}, \tag{C.10}$$

and the disturbance set:

$$\mathcal{W} = \{ w \in \mathbb{R}^m : \|\mathbf{w}\| \leq \mu \}. \tag{C.11}$$

Then from the IOS inequality we have that, if $\mathbf{x}(0) \in \mathcal{X}$ and $w \in \mathcal{W}$, the bound μ can be computed so that $\mathcal{X}$ is RPI. To construct the bound, it is sufficient to have the following condition to hold $\forall k \geq 0$:

$$\|\mathbf{e}(k)\| \leq \overline{\rho}^{\,k} \|\mathbf{e}(0)\| + \frac{h\|\mathbf{B}\|}{(1-\overline{\rho}\,)} \|\mathbf{w}\| \leq \overline{r},\ \forall w :\ \|\mathbf{w}\| \leq \mu. \tag{C.12}$$

The above is verified $\forall k \geq 0$ if the bound μ satisfies by the following sufficient condition when $\|\mathbf{e}(0)\| \geq \overline{r}$:

$$\begin{aligned}
\|\mathbf{e}(k)\| &\leq \overline{\rho}^{\,(k-1)} \|\mathbf{e}(0)\| + \overline{\rho}^{\,(k-1)} (\overline{\rho} - 1) \|\mathbf{e}(0)\| + \frac{h\|\overline{\mathbf{B}}\|}{(1-\overline{\rho}\,)} \|\mathbf{w}\| \\
&\leq \overline{\rho}^{\,(k-1)} \|\mathbf{e}(0)\| + (\overline{\rho} - 1) \|\mathbf{e}(0)\| \sup_j \sum_{t=0}^{j} \overline{\rho}^{\,(j-1)} + \frac{h\|\overline{\mathbf{B}}\|}{(1-\overline{\rho}\,)} \|\mathbf{w}\| \\
&= \overline{\rho}^{\,(k-1)} \|\mathbf{e}(0)\| - \frac{(1-\overline{\rho}\,)}{(1-\overline{\rho}\,)} \|\mathbf{e}(0)\| + \frac{h\|\overline{\mathbf{B}}\|}{(1-\overline{\rho}\,)} \|\mathbf{w}\| \\
&\leq \overline{\rho}^{\,(k-1)} \|\mathbf{e}(0)\| \leq \|\mathbf{e}(0)\| \\
&\leq \overline{\rho}^{\,(k-1)} \overline{r} \leq \overline{r},\ \text{if } \|\mathbf{e}(0)\| = \overline{r} \\
&\Leftarrow \frac{h\|\overline{\mathbf{B}}\|}{(1-\overline{\rho}\,)} \|\mathbf{w}\| \leq \|\mathbf{e}(0)\| \geq \overline{r} \\
&\Leftarrow \frac{r}{(1-\overline{\rho}\,)} \geq \frac{h\|\overline{\mathbf{B}}|_2}{(1-\overline{\rho}\,)} \|\mathbf{u}\| \\
&\Leftrightarrow r \geq h\|\overline{\mathbf{B}}\| \|\mathbf{w}\| \\
&\Leftarrow \|\mathbf{w}\| \leq \frac{r}{h\|\overline{\mathbf{B}}\|} = \mu.
\end{aligned} \tag{C.13}$$

This will not necessarily hold in the interior of the set but it will hold outside and on the boundary of this compact set. Therefore the set $\mathcal{X}$ is invariant under $\mathbf{u} = \overline{\mathbf{u}} + w$, with $w \in \mathcal{W}$, namely, no solution starting inside $\mathcal{X}$ can pass its boundary under any $w \in \mathcal{W}$. Note that, for *tanh* activations, this result holds globally. Conversely, given a bound μ for $\mathcal{W}$, we can compute $\overline{r}$ such that there is a set $\mathcal{X}$ that is RPI, namely:

$$\mathcal{X} = \left\{ x \in \mathbb{R}^n : \|\mathbf{x}\| \leq \overline{r} = \frac{h\|\overline{\mathbf{B}}\|}{1-\overline{\rho}} \mu \right\}. \tag{C.14}$$

Global practical Stability follows by taking $\overline{\mathbf{x}} = 0$ and $\delta = \gamma(\|\mathbf{u}\|)$.

2. For the ReLU activation, the set $\mathcal{P}(k)$ does not cover the entire $\mathbb{R}^n$. This complicates the analysis further as the output i of the network layer k becomes autonomous when $(\mathbf{x}(k), \mathbf{u}) \notin \mathcal{P}_i(k)$. In particular, if at any $k = t$ we have $(\mathbf{x}(t), \mathbf{u}) \notin \mathcal{P}_i(t)$ then, because of linearity, we also have that:

$$\begin{aligned} \mathbf{x}_i(t+1) &= \mathbf{x}_i(t) \Rightarrow \\ \|\mathbf{e}(t+1)\| &\leq \beta(\|\mathbf{e}(0)\|, t+1) + \gamma(\|\mathbf{w}\|) + \|\mathbf{e}_i(t) - (\mathbf{I}_i + \mathbf{A}_i)\mathbf{e}(t)\|, \\ &\leq \beta(\|\mathbf{e}(0)\|, t+1) + \gamma(\|\mathbf{w}\|) + \|\mathbf{e}(t) - (\mathbf{I} + \mathbf{A})\mathbf{e}(t)\|, \\ &= \beta(\|\mathbf{e}(0)\|, t+1) + \gamma(\|\mathbf{w}\|) + \| - \mathbf{A}\mathbf{e}(t)\|, \\ &\leq \beta(\|\mathbf{e}(0)\|, t+1) + \gamma(\|\mathbf{w}\|) + \zeta_t, \end{aligned} \tag{C.15}$$

where:

$$\begin{aligned} \zeta_t = \|\mathbf{A}\|\|\mathbf{e}(t)\| &\leq \|\mathbf{A}\| \sup_{\mathbf{x}(0)\in\mathcal{X}} \left(\beta(\|\mathbf{e}(0)\|, t)\right) \\ &= \overline{\rho}^{\,t}\|\mathbf{A}\| \sup_{\mathbf{x}(0)\in\mathcal{X}} \|\mathbf{e}(0)\|. \end{aligned} \tag{C.16}$$

Where $\mathcal{X} = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x} - \overline{\mathbf{x}}\| \leq \overline{r}\}$ is a bounded set of initial states. From the last two equations and the fact that $\overline{\rho}$ is the spectral radius of $(\mathbf{I} + \mathbf{A})$ we can satisfy the δ-IOpS condition, with:

$$\begin{aligned} \|\mathbf{e}(k)\| &\leq \beta(\|\mathbf{e}(0)\|, k) + \gamma(\|\mathbf{w}\|) + \sum_{t<k} \overline{\rho}^{\,t}\zeta_t, \\ \zeta_t \leq \zeta = \sup_k \sum_{j=0}^{k} \zeta_j &= \frac{\|\mathbf{A}\|}{(1-\overline{\rho}\,)}\overline{r} \leq \frac{\|\mathbf{A}\|}{\|\mathbf{I} - \mathbf{I} - \mathbf{A}\|}\overline{r} = \overline{r}. \end{aligned} \tag{C.17}$$

If $\mathbf{u} = \overline{\mathbf{u}} + \mathbf{w}$ with $\mathbf{w} \in \mathcal{W} = \{\mathbf{w} \in \mathbb{R}^m : \|\mathbf{w}\| \leq \mu\}$ then, by taking the sup over this set of the gain γ, and over k of the inequality Eq. (C.17), we have δ-IOpS condition with the ultimate bound:

$$\|\mathbf{x} - \overline{\mathbf{x}}\| \leq \frac{\|\mathbf{x}(0) - \overline{\mathbf{x}}(0)\|}{(1-\overline{\rho}\,)} + h\frac{\|\overline{\mathbf{B}}\|\mu}{(1-\overline{\rho}\,)}, \tag{C.18}$$

□

C.2 STABILITY PROOF FOR SHARED WEIGHTS

Proof. (Equation 1: Asymptotic stability for shared weights)

Tanh activation case:

1. Recall that we have assumed that, for a *tanh* activation function, the network dynamics can be globally approximated by its linearization and the steady-state condition is:

$$\bar{\mathbf{x}} = \bar{\mathbf{x}} + h\tanh(\mathbf{A}\bar{\mathbf{x}} + \mathbf{B}\mathbf{u} + \mathbf{b}) \Leftrightarrow \mathbf{A}\bar{\mathbf{x}} + \mathbf{B}\mathbf{u} + \mathbf{b} = 0. \tag{C.19}$$

   The network has a unique input-dependant steady-state $\bar{\mathbf{x}}$ with steady-state gain $G : \|\mathbf{H}(k)\| \to G$, given by, respectively[2]:

$$\bar{\mathbf{x}} = -\mathbf{A}^{-1}(\mathbf{B}\mathbf{u} + \mathbf{b}), \tag{C.20}$$

$$G = \left\| \frac{\partial \bar{\mathbf{x}}}{\partial \mathbf{u}} \right\| = \frac{\|\mathbf{B}\|}{\|\mathbf{A}\|}. \tag{C.21}$$

   From this point and the above considerations in the proof of Theorem 4 we will now prove that in the case of tied weights, the network with *tanh* activation is Globally Asymptotically Stable with equilibrium $\bar{\mathbf{x}}$ given by Eq. (C.20). In order to do this we will again use the linearized system and apply the superposition of the free response with the forced response. In particular, from the proof of Theorem 4 part 1 and from Eq. (C.7) we have that the free response of $\mathbf{x}(k) - \bar{\mathbf{x}}$, for any steady-state $\bar{\mathbf{x}}$, is norm bounded by:

$$\beta(\|\mathbf{e}(k)\|) = \bar{\rho}^{k}\|\mathbf{x}(0) - \bar{\mathbf{x}}\|, \tag{C.22}$$

   which vanishes asymptotically. Conversely, recall that the forced response (of the linearized system) to the input $\mathbf{u}$ is given by the discrete convolution Eq. (C.2). This convolution converges for any chosen linearization point as $\bar{\rho} < 1$. Therefore, by linearizing around any $\bar{\mathbf{x}}$, we have:

$$\mathbf{x}(k) \to (\mathbf{I} - \mathbf{J}(\bar{\mathbf{x}}, \mathbf{u}))^{-1} h\sigma'(\bar{\mathbf{x}}, \mathbf{u})(\mathbf{B}\mathbf{u} + \mathbf{b}) = -\mathbf{A}^{-1}(\mathbf{B}\mathbf{u} + \mathbf{b}), \tag{C.23}$$

   thus providing the desired Global Asymptotic Stability result.

2. From the fact that the network function is Lipschitz, the network is also Globally Input-Output Robustly Stable, as shown for the case of untied weights in the proof of Theorem 4, part 1. Moreover, from Eq. (C.7) we have that the IOS property holds also around any nominal equilibrium point $\bar{\mathbf{x}}$.

2 Note that the matrix $\mathbf{A}$ is invertible by construction.

ReLU activation case:

1. For ReLU activations, the network dynamics is piece-wise linear, and sub-differentiable. In particular, the network has 2^n possible 1-step transitions, where n is the number of dimensions. We will proceed by enumeration of all possible dynamics transition functions to show that one cannot determine the existence of a steady state or a single set of active neurons in the considered setting. This motivates the use of incremental stability. First of all, we have that:

$$\begin{aligned} \mathcal{P} &= \{(\mathbf{x}, \mathbf{u}) \in \mathbb{R}^{n+m} : \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{b} > 0\}, \\ \mathcal{P}_i &= \{(\mathbf{x}, \mathbf{u}) \in \mathbb{R}^{n+m} : \mathbf{A}_{i\bullet}\mathbf{x} + \mathbf{B}_{i\bullet}\mathbf{u} + \mathbf{b}_i > 0\}. \end{aligned} \tag{C.24}$$

The network state at the next step is then given by:

$$\mathbf{x}_i^+ = \begin{cases} \mathbf{x}_i + h(\mathbf{A}_{i\bullet}x + \mathbf{B}_{i\bullet}\mathbf{u} + \mathbf{b}_i) & if \ -\mathbf{A}_{i\bullet}\mathbf{x} \leq (\mathbf{B}_{i\bullet}\mathbf{u} + \mathbf{b}_i), \\ \mathbf{x}_i & if \ -\mathbf{A}_{i\bullet}x > (\mathbf{B}_{i\bullet}\mathbf{u} + \mathbf{b}_i). \end{cases} \tag{C.25}$$

The activation slope for the i-th coordinate is the set valued:

$$\sigma_{ii}^{'}(\Delta x) \in \begin{cases} \{1\}, & if \ -\mathbf{A}_{i\bullet}\mathbf{x} < (\mathbf{B}_{i\bullet}\mathbf{u} + \mathbf{b}_i) \\ \{0\}, & if \ -\mathbf{A}_{i\bullet}\mathbf{x} > (\mathbf{B}_{i\bullet}\mathbf{u} + \mathbf{b}_i) \\ [0,1], & if \ -\mathbf{A}_{i\bullet}\mathbf{x} = (\mathbf{B}_{i\bullet}\mathbf{u} + \mathbf{b}_i) \end{cases} \tag{C.26}$$

The Jacobian is again given by:

$$\mathbf{J}(\mathbf{x}, \mathbf{u}) = \mathbf{I} + \sigma^{'}(\Delta\mathbf{x})\mathbf{A}. \tag{C.27}$$

Clearly, if $(\mathbf{x}(k), \mathbf{u}) \in \mathcal{P}$, $\forall k$ then the system is linear and time invariant. In this region the system is linear and could admit a unique, input-dependant steady-state, given by:

$$\bar{\mathbf{x}} = -\mathbf{A}^{-1}(\mathbf{B}\mathbf{u} + \mathbf{b}), \tag{C.28}$$

This, however, cannot be guaranteed as discussed next.

In general, we cannot expect $(\mathbf{x}(k), \mathbf{u}) \in \mathcal{P}$, $\forall k$. We could instead look for a steady state to be either on the boundary of $\mathcal{P}$ or outside the set. In particular, if $(\mathbf{x}(k), \mathbf{u}) \notin \mathcal{P}$ for some k then $\exists\ i : \mathbf{x}_i(k+t) = \mathbf{x}_i(k)$, $\forall t \geq 0$ and therefore part of the network states will become autonomous at layer k, making the network simply stable if the activation stays saturated (which cannot be guaranteed). Consider the case in which, at some time k, we have $(\mathbf{x}(k), \mathbf{u}) \notin \mathcal{P}_i$ (for example in $\mathcal{N}_i$). In this case we have that $\mathbf{x}_i(k+t) = \mathbf{x}_i(k)$, $\forall t \geq 0$ and the network state will be free to change in its remaining dimensions with the i-th one being fixed for at least one step. For the remaining dimensions, we can take out $\mathbf{x}_i$ and consider its effect as an additional bias element. This will result in a smaller state space $\tilde{\mathbf{x}} = [\mathbf{x}_j, \ldots, \mathbf{x}_n]^T$

except for $\mathbf{x}_i$, with state transfer matrix $\mathbf{I} + h\tilde{\mathbf{A}}$, where $\tilde{\mathbf{A}}$ consists of all elements of $\mathbf{A}$, without the i-th row and column. Same for $\tilde{\mathbf{B}}$ having the elements of $\mathbf{B}$ except the i-th row. Note that $\tilde{\mathbf{A}}$ still has eigenvalues in the same region as $\mathbf{A}$ and is therefore negative definite and invertible. For this new system we have three possibilities:

a) In the first case the trajectory stays in $\mathcal{N}_i$ all the time and the steady-states for all dimensions except i are given by:

$$\bar{\tilde{\mathbf{x}}} = \tilde{\mathbf{A}}^{-1}(\tilde{\mathbf{B}}\mathbf{u} + \tilde{\mathbf{b}} + \bar{\mathbf{J}}\bar{\mathbf{x}}_i) \tag{C.29}$$

where $\bar{\mathbf{J}}$ is a diagonal matrix containing the i-th column of J except for the i-th element.

b) Another possibility is that at some point we also have that $(\mathbf{x}(k), \mathbf{u}) \notin \mathcal{P}_j$ for some other j (they also can be more than one at the time) in which case one can reduce the state space again and repeat as above.

c) The third possibility is, however, that some more, or even all of the activations become unsaturated again. Then, the system will continue to evolve in possibly all state dimensions, and its free evolution will be contracting once again but changing its direction with respect to the input. In this case, we can therefore only say that the system contracts according to $\|\mathbf{I} + \mathbf{A}\|$ as long as activations are not saturated, however, we cannot guarantee that they remain unsaturated nor that there is a steady state. Note that the network trajectory depends on the initial condition, and the fact that $(\mathbf{x}(k), \mathbf{u}) \in \mathcal{P}_i$ at time k is also dependent on $\mathbf{x}(0)$. At the same time, the contraction in $\|\mathbf{I} + \mathbf{A}\|$ and of all sub-matrices is sufficient to show convergence in the same norm space, for a zero input and for each pair of trajectories, as well as the bound discussed for the unshared weight case in Theorem 4:

$$\zeta = \frac{\|\mathbf{x}(0) - \bar{\mathbf{x}}(0)\|}{\bar{\rho}}. \tag{C.30}$$

2. To prove Input-Output incremental practical Stability, note that each combination of different vector fields that make up $f(\mathbf{x}, \mathbf{u})$ provides a vector field that is Input-Output Stable. Moreover, f is uniformly continuous. We can therefore take the worst case IOpS gain for each value of the vector field to provide suitable upper bounds for the δ-IOpS definition to be satisfied as in proof of Theorem 4, part 2.

□

Note that the gain $\gamma(\cdot)$ can be used, for instance, as a regularizer to reduce the effect of input perturbations on the output of the network.

D CONSTRAINTS IMPLEMENTATION PROOFS

In this section, the proposed implementation for fully connected and convolutional layers is shown to be sufficient to fulfil the stability constraint on the Jacobian spectral radius.

D.1 PROOF OF FULLY CONNECTED IMPLEMENTATION

Recall Algorithm 1 from Chapter 3. The following result is obtained:

Lemma 3. *The Jacobian stability condition, $\rho(\mathbf{J}(\mathbf{x},\mathbf{u})) < 1$, is satisfied $\forall(\mathbf{x},\mathbf{u}) \in \mathcal{P}$ for the fully connected layer if $h \leq 1$ and Algorithm 1 is used.*

Since Lemma 3 is equivalent to Theorem 2 from Chapter 3, the former will be proven next.

Proof. (Proof of Lemma 3)
Recall that Algorithm 1 results in the following operation being performed after each training epoch:

$$\tilde{\mathbf{R}} \leftarrow \begin{cases} \sqrt{\delta}\dfrac{\mathbf{R}}{\sqrt{\|\mathbf{R}^T\mathbf{R}\|_F}} & \text{if } \|\mathbf{R}^T\mathbf{R}\|_F > \delta \\ \mathbf{R}, & \text{Otherwise,} \end{cases} \tag{D.1}$$

with $\delta = 1 - 2\epsilon \in (0,1)$. Recall that

$$\mathbf{A} = -\mathbf{R}^T\mathbf{R} - \epsilon\mathbf{I}. \tag{D.2}$$

Then, Eq. (D.1) is equivalent to the update:

$$\tilde{\mathbf{A}} \leftarrow \begin{cases} -\tilde{\mathbf{R}}^T\tilde{\mathbf{R}} - \epsilon\mathbf{I} = -(1-2\epsilon)\dfrac{\mathbf{R}^T\mathbf{R}}{\|\mathbf{R}^T\mathbf{R}\|_F} - \epsilon\mathbf{I}, & \text{if } \|\mathbf{R}^T\mathbf{R}\|_F > 1-2\epsilon \\ \mathbf{A}, & \text{Otherwise} \end{cases} \tag{D.3}$$

Eq. (D.1) guarantees that $\|\mathbf{R}^T\mathbf{R}\|_F \leq (1-2\epsilon)$. From the fact that the Frobenius norm is an upper bound of the spectral norm and because of symmetry we have that:

$$\rho(\mathbf{R}^T\mathbf{R}) = \|\mathbf{R}^T\mathbf{R}\|_2 \leq \|\mathbf{R}^T\mathbf{R}\|_F \leq 1-2\epsilon. \tag{D.4}$$

Recall that, from Eq. (D.2), $\mathbf{A}$ is negative definite and it only has real negative real eigenvalues. Recall also Lemma 2. Therefore, by applying the definition in Eq. (D.2) we have that $R = 0 \Rightarrow \mathbf{A} = -\epsilon\mathbf{I}$, then the eigenvalues of $h\mathbf{A}$ are always located within the interval $[-h(1-\epsilon), -h\epsilon]$. This means that:

$$\rho(h\mathbf{A}) \leq h \max\{\epsilon, 1-\epsilon\}. \tag{D.5}$$

To complete the proof, recall that the network Jacobian is:

$$\mathbf{J}(\mathbf{x}, \mathbf{u}) = \mathbf{I} + h\sigma^{'}(\Delta\mathbf{x})\mathbf{A}.$$

We will now look at the specific activation functions:

1. For ReLU activation, we simply have that $\mathbf{J}(\mathbf{x}, \mathbf{u}) = \mathbf{I} + h\mathbf{A}$ in the set $\mathcal{P}$. Lemma 2 implies that $I + h\mathbf{A}$ has only positive real eigenvalues located in $[1 - h(1-\epsilon), 1 - h\epsilon]$ which, when $h \in (0, 1]$, implies that:

$$\overline{\rho} \leq \max\{1 - h(1-\epsilon), 1 - h\epsilon\} < 1. \tag{D.6}$$

2. For *tanh* activations, since the matrix

$$\overline{\mathbf{A}} = \frac{1}{2}\left(\sigma^{'}(\Delta\mathbf{x})\mathbf{A} + \mathbf{A}^T\left(\sigma^{'}(\Delta\mathbf{x})\right)^T\right) \tag{D.7}$$

is symmetric, $\mathbf{A}$ is negative definite and $\sigma^{'}(\cdot)$ is diagonal with entries $\sigma^{'}_{ii}(\cdot) \in [\underline{\sigma}, 1]$ with $0 < \underline{\sigma} \ll 1$ when $(\mathbf{x}, \mathbf{u}) \in \mathcal{P}$, then $\overline{\mathbf{A}}$ is also negative definite in this set. Therefore, in virtue of the observations at page 399-400 of Horn and Johnson, 2012, we have that the real part of the eigenvalues of $\sigma^{'}(\Delta\mathbf{x})A$ is always less than zero in $\mathcal{P}$. Namely,

$$\mathrm{Re}(\mathrm{eig}(\sigma^{'}(\Delta\mathbf{x})\mathbf{A})) < 0. \tag{D.8}$$

At the same time, by construction of $\mathbf{A}$ and again thanks to $\sigma^{'}_{ii}(\cdot) \in [\underline{\sigma}, 1]$ and $\sigma^{'}_{ij}(\cdot) = 0$ if $i \neq j$, we have that:

$$\begin{aligned}\rho(\sigma^{'}(\Delta\mathbf{x})\mathbf{A}) &\leq \|\sigma^{'}(\Delta\mathbf{x})\mathbf{A}\|_2 \\ &\leq \|\sigma^{'}(\Delta\mathbf{x})\|_2\|\mathbf{A}\|_2 \\ &\leq \|\mathbf{A}\|_2 \\ &\leq 1 - \epsilon\end{aligned} \tag{D.9}$$

From the above considerations the real part of the eigenvalues of $h\sigma^{'}(\Delta\mathbf{x})\mathbf{A}$ is in the interval $[-h(1-\epsilon), -h\underline{\sigma}\epsilon]$. Assume $h \leq 1$.

Finally, we show that the Jacobian has only positive real eigenvalues. From Theorem 2.2 in Wu, 1988, we know that $\sigma^{'}(\Delta\mathbf{x})A$ is *similar* to:

$$\begin{aligned}&\sigma^{'}(\Delta\mathbf{x})^{-1/2}\sigma^{'}(\Delta\mathbf{x})\mathbf{A}\sigma^{'}(\Delta\mathbf{x})^{1/2} \\ &= \sigma^{'}(\Delta\mathbf{x})^{1/2}\mathbf{A}\sigma^{'}(\Delta\mathbf{x})^{1/2} \\ &= \sigma^{'}(\Delta\mathbf{x})^{1/2}\mathbf{A}(\sigma^{'}(\Delta\mathbf{x})^{1/2})^T\end{aligned} \tag{D.10}$$

which is symmetric, negative definite and therefore has only negative real eigenvalues, provided that $(\mathbf{x}, \mathbf{u}) \in \mathcal{P}$. Since similarity implies eigenvalue equivalence, then $\sigma'(\Delta\mathbf{x})A$ has only negative real eigenvalues. Then, from Lemma 2, and from Eq. (D.8) and Eq. (D.9) we have that the eigenvalues of $\mathbf{J}(\mathbf{x}, \mathbf{u})$ are positive real and contained in $[1 - h(1 - \epsilon), 1 - h\underline{\sigma}\epsilon]$. Therefore we have that $\overline{\rho} \leq \max\{1 - h(1 - \epsilon), 1 - h\underline{\sigma}\epsilon\}$, which is less than 1 as $\underline{\sigma} > 0$ by definition.

□

The less restrictive bound $\delta = 2(1 - \epsilon)$ with $h \leq 1$ is also sufficient for stability but it can result in trajectories that oscillate since it it does not constrain the eigenvalues to be positive real. Practically speaking the bound $\delta = 2(1 - \epsilon)$ has has proven sufficient for our MNIST experiments to be successful, however, we believe it is important to stress the difference between this and our proposed bound. In particular, our solution for fully connected layers leads to a critically damped system, i.e., to a monotonic trajectory for the 1D case. This means that we can expect the activations to behave monotonically both in time and in space. This behaviour is demonstrated in Section 3.5. The additional regularity of the resulting function acts as a stronger regularisation on the network.

Note also that in the above proof we have shown that, in the case of ReLU, the 2-norm is suitable to prove stability. Moreover, if $h = 1$ and $\epsilon \leq 0.5$ then we have $\overline{\rho} = 1 - \epsilon$.

D.2 DERIVATION OF CONVOLUTIONAL LAYER IMPLEMENTATION

D.2.1 Mathematical derivation of the proposed algorithm

Denote the convolution operator in NAIS-Net, for each latent map c, as the following:

$$\mathbf{X}^{(c)}(k+1) = \mathbf{X}^{(c)}(k) + h\sigma\left(\Delta\mathbf{X}^{(c)}(k)\right), \tag{D.11}$$

where:

$$\Delta\mathbf{X}^{(c)}(k) = \sum_i \mathbf{C}_{(i)}^{(c)} * \mathbf{X}^{(i)}(k) + \sum_j \mathbf{D}_{(j)}^{(c)} * \mathbf{U}^{(j)} + \mathbf{E}^{(c)}, \tag{D.12}$$

and where $\mathbf{X}^{(i)}(k)$, is the layer state matrix for channel i, $\mathbf{U}^{(j)}$, is the layer input data matrix for channel j (where an appropriate zero padding has been applied) at layer k. The activation function, σ, is again applied element-wise. Recall also Algorithm 2 from Chapter 3. The following Lemma is obtained (equivalent to Theorem 3 from Chapter 3):

Lemma 4. *If Algorithm 2 is used, then the Jacobian stability condition,* $\rho(\mathbf{J}(\mathbf{x},\mathbf{u})) < 1$, *is satisfied* $\forall(\mathbf{x},\mathbf{u}) \in \mathcal{P}$ *for the convolutional layer with* $h \leq 1$.

The first step to obtain the result is to prove that the convolutional layer can be expressed as a suitable fully connected layer as in Lemma 1 from the main paper. This is shown next.

Proof. (Proof of Lemma 1 from Chapter 3)

For layer k, define $X(k)$ as a tall matrix containing all the state channel matrices $\mathbf{X}^{(c)}(k)$ stacked vertically, namely:

$$\mathbf{X} = \begin{pmatrix} \mathbf{X}^{(1)} \\ \mathbf{X}^{(2)} \\ \vdots \\ \mathbf{X}^{(N_c)} \end{pmatrix}. \tag{D.13}$$

Similar considerations apply to the input matrix $\mathbf{U}$ and the bias matrix $\mathbf{E}$. Then, convolutional layers can be analysed in a similar way to fully connected layers, by means of the following vectorised representation:

$$\mathbf{x}(k+1) = \mathbf{x}(k) + h\sigma\big(\mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u} + \mathbf{b}\big), \tag{D.14}$$

where $\mathbf{x}(k) \in \mathbb{R}^{n_{\mathbf{x}}^2 \cdot N_c}$, $u = \mathbb{R}^{n_U^2 \cdot N_c}$, where N_c is the number of channels, n_X is the size of the latent space matrix for a single channel, while n_U is the size of the input data matrix for a single channel. In Eq. (D.14), the matrices $\mathbf{A}$ and $\mathbf{B}$ are made of blocks containing the convolution filters elements in a particular structure, to be characterised next. First, in order to preserve dimensionality of $\mathbf{x}$, the convolution for $\mathbf{x}$ will have a fixed stride of 1, a filter size n_C and a zero padding of $p \in \mathbb{N}$, such that $n_C = 2p + 1$. If a greater stride is used, then the state space can be extended with an appropriate number of constant zero entries (not connected). Let's then consider, without loss of generality, a unitary stride. The matrix $\mathbf{A}$ is all we need to define in order to prove the Lemma.

In Eq. (D.14), the vector $\mathbf{x}$ is chosen to be the vectorised version of X. In particular,

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}^{(1)} \\ \mathbf{x}^{(2)} \\ \vdots \\ \mathbf{x}^{(N_c)} \end{pmatrix}, \tag{D.15}$$

where $\mathbf{x}^{(c)}$ is the vectorised version of $\mathbf{X}^{(c)}$. More specifically, these objects are defined as:

$$\mathbf{X}^{(c)} = \begin{pmatrix} \mathbf{X}_{1,1}^{(c)} & \mathbf{X}_{1,2}^{(c)} & \cdots & \mathbf{X}_{1,n}^{(c)} \\ \mathbf{X}_{2,1}^{(c)} & \mathbf{X}_{2,2}^{(c)} & \cdots & \mathbf{X}_{2,n}^{(c)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{X}_{n,1}^{(c)} & \mathbf{X}_{n,2}^{(c)} & \cdots & \mathbf{X}_{n,n}^{(c)} \end{pmatrix} \in \mathbb{R}^{n_X \times n_X}, \tag{D.16}$$

and

$$\mathbf{x}^{(c)} = \begin{pmatrix} \mathbf{X}^{(c)}_{1,1} \\ \mathbf{X}^{(c)}_{1,2} \\ \vdots \\ \mathbf{X}^{(c)}_{n,n} \end{pmatrix} \in \mathbb{R}^{n^2_{\mathbf{x}}}. \tag{D.17}$$

Similar considerations apply to u. The matrix $\mathbf{A}$ in Eq. (D.14) has the following structure:

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}^{(1)}_{(1)} & \mathbf{A}^{(1)}_{(2)} & \dots & \mathbf{A}^{(1)}_{(Nc)} \\ \mathbf{A}^{(2)}_{(1)} & \mathbf{A}^{(2)}_{(2)} & \dots & \mathbf{A}^{(2)}_{(Nc)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A}^{(Nc)}_{(1)} & \mathbf{A}^{(Nc)}_{(2)} & \dots & \mathbf{A}^{(Nc)}_{(Nc)} \end{pmatrix} \in \mathbb{R}^{(n^2 \cdot N_c) \times (n^2 \cdot N_c)}, \tag{D.18}$$

where Nc is the number of channels and $\mathbf{A}^{(c)}_{(i)}$ corresponds to the filter $\mathbf{C}^{(c)}_{(i)}$. In particular, each row of $\mathbf{A}$ contains in fact the elements of the filter $\mathbf{C}^{(c)}_{(i)}$, plus some zero elements, with the central element of the filter $\mathbf{C}^{(c)}_{(i)}$ on the diagonal. The latter point is instrumental to the proof and can be demonstrated as follows. Define the single channel filters as:

$$\mathbf{C}^{(c)}_{(i)} = \begin{pmatrix} \mathbf{C}^{(c)}_{(i)\,1,1} & \mathbf{C}^{(c)}_{(i)\,1,2} & \dots & \mathbf{C}^{(c)}_{(i)\,1,n_C} \\ \mathbf{C}^{(c)}_{(i)\,2,1} & \mathbf{C}^{(c)}_{(i)\,2,2} & \dots & \mathbf{C}^{(c)}_{(i)\,2,n_C} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{C}^{(c)}_{(i)\,n_C,1} & \mathbf{C}^{(c)}_{(i)\,n_C,2} & \dots & \mathbf{C}^{(c)}_{(i)\,n_C,n_C} \end{pmatrix}. \tag{D.19}$$

Consider now the output of the single channel convolution $\mathbf{Z}^{(c)}_{(i)} = \mathbf{C}^{(c)}_{(i)} * \mathbf{X}^{(i)}$ with the discussed padding and stride. The first element of the resulting matrix, $\mathbf{Z}^{(c)}_{(i)\,1,1}$ is determined by applying the filter to the first patch of $\mathbf{X}^{(i)}$, suitably padded. For instance, for a 3-by-3 filter ($p = 1$), we have:

$$\text{patch}_{1,1}\left(\mathbf{X}^{(i)}\right) = \begin{pmatrix} 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \mathbf{X}^{(c)}_{1,1} & \mathbf{X}^{(c)}_{1,2} & \cdots & \mathbf{X}^{(c)}_{1,n} & 0 \\ 0 & \mathbf{X}^{(c)}_{2,1} & \mathbf{X}^{(c)}_{2,2} & \cdots & \mathbf{X}^{(c)}_{2,n} & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \mathbf{X}^{(c)}_{n,1} & \mathbf{X}^{(c)}_{n,2} & \cdots & \mathbf{X}^{(c)}_{n,n} & 0 \end{pmatrix}. \tag{D.20}$$

The first element of $\mathbf{Z}^{(c)}_{(i)}$ is therefore given by:

$$\mathbf{Z}^{(c)}_{(i)\,1,1} = \mathbf{X}^{(i)}_{1,1}\,\mathbf{C}^{(c)}_{(i)\,i_{\text{centre}},i_{\text{centre}}} + \mathbf{X}^{(i)}_{1,2}\,\mathbf{C}^{(c)}_{(i)\,i_{\text{centre}},i_{\text{centre}}+1} + \dots + \mathbf{X}^{(i)}_{n_C-p,n_C-p}\,\mathbf{C}^{(c)}_{(i)\,n_C,n_C}, \tag{D.21}$$

where i_{centre} denotes the central row (and column) of the filter.

The second element of the first row can be computed by means of the following patch (again when $p = 1$, for illustration):

$$\text{patch}_{1,2}\left(\mathbf{X}^{(i)}\right) = \begin{pmatrix} 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \mathbf{X}^{(c)}_{1,1} & \mathbf{X}^{(c)}_{1,2} & \cdots & \mathbf{X}^{(c)}_{1,n} & 0 \\ 0 & \mathbf{X}^{(c)}_{2,1} & \mathbf{X}^{(c)}_{2,2} & \cdots & \mathbf{X}^{(c)}_{2,n} & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \mathbf{X}^{(c)}_{n,1} & \mathbf{X}^{(c)}_{n,2} & \cdots & \mathbf{X}^{(c)}_{n,n} & 0 \end{pmatrix}. \tag{D.22}$$

The element is therefore given by:

$$\mathbf{Z}^{(c)}_{(i)\,1,2} = \mathbf{X}^{(i)}_{1,1}\,\mathbf{C}^{(c)}_{(i)\,i_{\text{centre}},i_{\text{centre}}-1} + \mathbf{X}^{(i)}_{1,2}\,\mathbf{C}^{(c)}_{(i)\,i_{\text{centre}},i_{\text{centre}}} + \cdots + \mathbf{X}^{(i)}_{n_C-p,n_C}\,\mathbf{C}^{(c)}_{(i)\,n_C,n_C}. \tag{D.23}$$

The remaining elements of $\mathbf{Z}^{(c)}_{(i)}$ will follow a similar rule, which will involve (in the worst case) all of the elements of the filter. In particular, one can notice for Z_{ij} the corresponding element of X, X_{ij}, is always multiplied by $\mathbf{C}^{(c)}_{(i)\,i_{\text{centre}},i_{\text{centre}}}$. In order to produce the matrix $\mathbf{A}$, we can consider the Jacobian of Z. In particular, for the first two rows of $\mathbf{A}^{(c)}_{(i)}$ we have:

$$\mathbf{A}^{(c)}_{(i)\,1,\bullet} = \frac{\partial \mathbf{Z}^{(c)}_{(i)\,1,1}}{\partial \mathbf{X}^{(i)}} = \begin{bmatrix} \mathbf{C}^{(c)}_{(i)\,i_{\text{centre}},i_{\text{centre}}} & \mathbf{C}^{(c)}_{(i)\,i_{\text{centre}}+1,i_{\text{centre}}+1} & \cdots & \mathbf{C}^{(c)}_{(i)\,n_C,n_C} & 0 \end{bmatrix}, \tag{D.24}$$

where $[\cdot]$ is used to define a row vector, and where $\mathbf{A}^{(c)}_{(i)\,1,\bullet}$ has an appropriate number of zeros at the end, and

$$\mathbf{A}^{(c)}_{(i)\,2,\bullet} = \frac{\partial \mathbf{Z}^{(c)}_{(i)\,1,2}}{\partial \mathbf{X}^{(i)}} = \begin{bmatrix} \mathbf{C}^{(c)}_{(i)\,i_{\text{centre}},i_{\text{centre}}-1} & \mathbf{C}^{(c)}_{(i)\,i_{\text{centre}},i_{\text{centre}}} & \cdots & \mathbf{C}^{(c)}_{(i)\,n_C,n_C} & 0 \end{bmatrix}. \tag{D.25}$$

Note that, the vectors defined in Eq. (D.24) and Eq. (D.25), contain several zeros among the non-zero elements. By applying the filter $\mathbf{C}^{(c)}_{(i)}$ to the remaining patches of $\mathbf{X}^{(i)}$ one can inductively construct the matrix $\mathbf{A}^{(c)}_{(i)}$. It can also be can noticed that each row $\mathbf{A}^{(c)}_{(i)\,j,\bullet}$ contains *at most* all of the elements of the filter, with the central element of the filter in position j. By stacking together the obtained rows $\mathbf{A}^{(c)}_{(i)\,j,\bullet}$ we obtain a matrix, $\mathbf{A}^{(c)}_{(i)}$, which has $\mathbf{C}^{(c)}_{(i)\,i_{\text{centre}},i_{\text{centre}}}$ on the diagonal. Each row of this matrix contains, in the worst case, all of the elements of $\mathbf{C}^{(c)}_{(i)}$.

We define the vectorised version of $\mathbf{Z}_{(i)}^{(c)}$ as:

$$\mathbf{Z}_{(i)}^{(c)} = \begin{pmatrix} \mathbf{Z}_{(i)}^{(c)}{}_{1,1} \\ \mathbf{Z}_{(i)}^{(c)}{}_{1,2} \\ \vdots \\ \mathbf{Z}_{(i)}^{(c)}{}_{n,n} \end{pmatrix} \in \mathbb{R}^{n^2}, \tag{D.26}$$

which, by linearity, satisfies:

$$\mathbf{z}_{(i)}^{(c)} = \mathbf{A}_{(i)}^{(c)} x^{(i)}. \tag{D.27}$$

By summing over the index i we obtain the vectorised output of the convolution $C * X$ for the channel c:

$$\mathbf{z}^{(c)} = \sum_{i=1}^{N_c} \mathbf{A}_{(i)}^{(c)} x^{(i)} = \begin{bmatrix} \mathbf{A}_{(1)}^{(c)} & \mathbf{A}_{(2)}^{(c)} & \dots & \mathbf{A}_{(N_c)}^{(c)} \end{bmatrix} x = \mathbf{A}^{(c)} x, \tag{D.28}$$

where the matrices $\mathbf{A}_{(i)}^{(c)}$ are stacked horizontally and where we have used the definition of $\mathbf{x}$ given in Eq. (D.15). The full matrix $\mathbf{A}$ is therefore given by:

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}^{(1)} \\ \mathbf{A}^{(2)} \\ \vdots \\ \mathbf{A}^{(N_c)} \end{pmatrix} = \left(\begin{array}{cccc} \mathbf{A}_{(1)}^{(1)} & \mathbf{A}_{(2)}^{(1)} & \dots & \mathbf{A}_{(N_c)}^{(1)} \\ \hline \mathbf{A}_{(1)}^{(2)} & \mathbf{A}_{(2)}^{(2)} & \dots & \mathbf{A}_{(N_c)}^{(2)} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline \mathbf{A}_{(1)}^{(N_c)} & \mathbf{A}_{(2)}^{(N_c)} & \dots & \mathbf{A}_{(N_c)}^{(N_c)} \end{array}\right), \tag{D.29}$$

where we have conveniently separated the long blocks used to produce the single channels result, $\mathbf{z}^{(c)}$. By defining the vector

$$\mathbf{z} = \begin{pmatrix} \mathbf{z}^{(1)} \\ \mathbf{z}^{(2)} \\ \vdots \\ \mathbf{z}^{(N_c)} \end{pmatrix}, \tag{D.30}$$

and by means of Eq. (D.28) we have that $z = Ax$. This is a vectorised representation of $\mathbf{C} * \mathbf{X}$. □

We are now ready to prove Lemma 4 and consequently Theorem 3 from Chapter 3.

Proof. (Proof of Lemma 4)

Similar to what done for the fully connected layer, we will first show that Algorithm 2 places the eigenvalues of $\mathbf{I} + \mathbf{A}$ strictly inside the unit circle. Then, we will also show that $\mathbf{J}(\mathbf{x}, \mathbf{u})$ enjoys the same property in $\mathcal{P}$.

We will now derive the steps used in Algorithm 2 to enforce that $\rho(\mathbf{I}+\mathbf{A}) \leq 1-\epsilon$. Recall that, from Lemma 2, we need the eigenvalues $\mathbf{A}$ to be lying inside the circle, $\mathcal{S}$, of the complex plane centered at $(-1, 0_J)$ with radius $1-\epsilon$. More formally:

$$\mathcal{S} = \{\lambda \in \mathbb{C} : |\lambda + 1| \leq 1 - \epsilon\}. \tag{D.31}$$

The Gershgorin theorem offers a way to locate the eigenvalues of $\mathbf{A}$ inside $\mathcal{S}$. Consider the c-th long long block in Eq. (D.29), $\mathbf{A}^{(c)}$. Recall the particular structure of $\mathbf{A}^{(c)}_{(c)}$ as highlighted in Eq. (D.24) and Eq. (D.25). For each long block $\mathbf{A}^{(c)}$ we have that all associated Gershgorin disks must satisfy:

$$\left|\lambda - \mathbf{C}^{(c)}_{i_{\text{centre}}}\right| \leq \sum_{i \neq i_{\text{centre}}} \left|\mathbf{C}^{(c)}_{i}\right|, \tag{D.32}$$

where λ is any of the corresponding eigenvalues, $\mathbf{C}^{(c)}_{i_{\text{centre}}}$ is the *central element* of the filter $\mathbf{C}^{(c)}_{(c)}$, namely, $\mathbf{C}^{(c)}_{i_{\text{centre}}} = \mathbf{C}^{(c)}_{(c)_{i_{\text{centre}}, i_{\text{centre}}}}$, and the sum is performed over all of the remaining elements of $\mathbf{C}^{(c)}_{(c)}$ plus all the elements of the remaining filters used for, $\mathbf{z}^{(c)}$, namely $\mathbf{C}^{(c)}_{(j)}$, $\forall j \neq c$. Therefore, one can act directly on the kernel $\mathbf{C}$ without computing $\mathbf{A}$.

By applying Algorithm 2, to each channel c we have that:

$$\mathbf{C}^{(c)}_{i_{\text{centre}}} = -1 - \delta_c, \tag{D.33}$$

and

$$\sum_{i \neq i_{\text{centre}}} \left|\mathbf{C}^{(c)}_{i}\right| \leq 1 - \epsilon - |\delta_c| \tag{D.34}$$

where

$$|\delta_c| < 1 - \eta, \; 0 < \epsilon < \eta < 1. \tag{D.35}$$

This means that for every c, the c-th block of $\mathbf{A}$ has the largest Gershgorin region bounded by a disk, $\mathcal{S}^c$. This disk is centred at $(-1-\delta_c, 0_J)$ with radius $1 - \epsilon - |\delta_c|$. More formally, the disk is defined as:

$$\mathcal{S}^c = \{\lambda \in \mathbb{C} : |\lambda + 1 + \delta_c| \leq 1 - \epsilon - |\delta_c|\}. \tag{D.36}$$

Thanks to Eq. (D.35), this region is not empty and its center can be only strictly inside $\mathcal{S}$. Clearly, we have that, $\mathcal{S}^c \subseteq \mathcal{S}, \forall c$. This follows from convexity and from the fact that, when $|\delta_c| = 1 - \eta$, thanks to Eq. (D.35) we have

$$\mathcal{S}^c = \{\lambda \in \mathbb{C} : |\lambda + 1 \pm (1-\eta)| \leq 1 - \epsilon - (1-\eta)\} \subset \mathcal{S} \tag{D.37}$$

while on the other hand $\delta_c = 0 \Rightarrow \mathcal{S}^c = \mathcal{S}$.

We will now show that the eigenvalue condition also applies to the network state Jacobian. From Algorithm 2 we have that each row j of the matrix $\mathrm{J} = \mathbf{I} + \mathbf{A}$ satisfies (for the corresponding channel c):

$$\begin{aligned}\sum_i \left|\mathrm{J}_{ji}\right| &\leq \left|1 + \mathbf{C}^{(c)}_{i_{\text{centre}}}\right| + \sum_{i \neq i_{\text{centre}}} \left|\mathbf{C}^{(c)}_i\right| \\ &\leq |\delta_c| + 1 - \epsilon - |\delta_c| = 1 - \epsilon.\end{aligned} \tag{D.38}$$

Thus $\|\mathrm{J}\|_\infty = \|\mathbf{I} + \mathbf{A}\|_\infty \leq 1 - \epsilon < 1$. [1] Recall that, in the vectorised representation, the state Jacobian is:

$$\mathbf{J}(\mathbf{x}, \mathbf{u}) = \mathbf{I} + h\sigma^{'}(\Delta\mathbf{x})\mathbf{A}. \tag{D.39}$$

Then, if $h \leq 1$, we have that $\forall(\mathbf{x}, \mathbf{u}) \in \mathcal{P}$:

$$\begin{aligned}\|\mathbf{J}(\mathbf{x}, \mathbf{u})\|_\infty &= \max_i |1 - h\sigma^{'}_{ii}(\Delta\mathbf{x})(1 + \delta_c)| + h\sigma^{'}_{ii}(\Delta\mathbf{x}) \sum_{j \neq i} |\mathbf{A}_{ij}| \\ &\leq \max_i \{1 - h\sigma^{'}_{ii}(\Delta\mathbf{x}) + h\sigma^{'}_{ii}(\Delta\mathbf{x})|\delta_c| + h\sigma^{'}_{ii}(\Delta\mathbf{x})(1 - \epsilon) - h\sigma^{'}_{ii}(\Delta\mathbf{x})|\delta_c|\} \\ &= \max_i \{1 - h\sigma^{'}_{ii}(\Delta\mathbf{x})\epsilon\} < 1 - h\underline{\sigma}\epsilon \leq 1.\end{aligned} \tag{D.40}$$

The proof is concluded by means of the identity $\rho(\cdot) \leq \|\cdot\|$. □

Note that in the above we have also showed that in the convolutional case the infinity norm is suitable to prove stability. Moreover, for ReLU we have that $\overline{\rho} \leq 1 - h\epsilon$.

The above results can directly be extended to the untied weight case providing that the projections are applied at each stage of the unroll.

In Chapter 3 we have used $\delta_c = 0,\ \forall c$ (equivalently, $\eta = 1$). This means that one filter weight per latent channel is fixed at -1 which might seem conservative. It is however worth noting that this choice provides the biggest Gershgorin disk for the matrix $\mathbf{A}$ as defined in Eq. (D.37). Hence $\delta_c = 0,\ \forall c$ results in the least restriction for the remaining elements of the filter bank. At the same time, we have N_C less parameters to train. A tradeoff is however possible if one wants to experiment with different values of $\eta \in (\epsilon, 1)$.

D.2.2 Illustrative Example

Consider, for instance, 3 latent channels $\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(3)}$, which results in 3 blocks of 3 filters, 1 block per channel $\mathbf{X}^{(c)}(k+1)$. Each filter has the same

1 Note that this also implies that $\rho(\mathbf{I} + \mathbf{A}) \leq 1 - \epsilon$, by means of the matrix norm identity $\rho(\cdot) \leq \|\cdot\|$. This was however already verified by the Gershgorin Theorem.

size n_C, which again needs to be chosen such that the channel dimensionality is preserved. This corresponds to the following matrix:

$$\mathbf{A} = \left(\begin{array}{c|c|c} \mathbf{A}^{(1)}_{(1)} & \mathbf{A}^{(1)}_{(2)} & \mathbf{A}^{(1)}_{(3)} \\ \hline \mathbf{A}^{(2)}_{(1)} & \mathbf{A}^{(2)}_{(2)} & \mathbf{A}^{(2)}_{(3)} \\ \hline \mathbf{A}^{(3)}_{(1)} & \mathbf{A}^{(3)}_{(2)} & \mathbf{A}^{(3)}_{(3)} \end{array}\right) = \\ = \left(\begin{array}{cc|ccc|cc} \mathbf{C}^{(1)}_{i_{\text{centre}}} & \mathbf{C}^{(1)}_{j} & \mathbf{C}^{(1)}_{n_C-1} & \cdots & \cdots & \mathbf{C}^{(1)}_{3\cdot n_C-1} \\ \vdots & \vdots & \ddots & \cdots & \cdots & \vdots \\ \hline \tilde{\mathbf{C}}^{(2)}_{1} & \cdots & \mathbf{C}^{(2)}_{i_{\text{centre}}} & \cdots & \cdots & \mathbf{C}^{(2)}_{3\cdot c_C-1} \\ \vdots & \vdots & \ddots & \cdots & \cdots & \vdots \\ \hline \tilde{\mathbf{C}}^{(3)}_{1} & \cdots & \cdots & \cdots & \mathbf{C}^{(3)}_{i_{\text{centre}}} & \vdots \\ \vdots & \vdots & \ddots & \cdots & \cdots & \ddots \end{array}\right), \tag{D.41}$$

where the relevant rows include all elements of the filters (in the worst case) together with a large number of zeros, the position of which does not matter for our results, and the diagonal elements are known and non-zero. Therefore, according to Eq. (D.41), we have to repeat Algorithm 2 from Chapter 3 for each of the 3 filter banks.

BIBLIOGRAPHY

Abadi, Martin et al. (2016). "TensorFlow: A system for large-scale machine learning." In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283 (p. 55).

Amari, Shun-ichi (1998). "Natural gradient works efficiently in learning." In: *Neural Computation* (p. 21).

Amari, Shun-ichi (2016). *Information geometry and its applications*. Vol. 194. Springer (p. 21).

Amari, Shun-ichi and Hiroshi Nagaoka (2007). *Methods of information geometry*. Vol. 191. American Mathematical Soc. (p. 21).

Andrychowicz, Marcin, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas (2016). "Learning to learn by gradient descent by gradient descent." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 3981–3989 (p. 84).

Anil, Cem, James Lucas, and Roger Grosse (2019). "Sorting out lipschitz function approximation." In: *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR, pp. 291–301 (p. 97).

Antoniou, Antreas, Harrison Edwards, and Amos Storkey (2019). "How to train your MAML." In: *International Conference on Learning Representations (ICLR)* (p. 84).

Ascher, Uri M and Linda R Petzold (1998). *Computer methods for ordinary differential equations and differential-algebraic equations*. Vol. 61. Siam (p. 34).

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E Hinton (2016). "Layer normalization." In: *arXiv preprint arXiv:1607.06450* (p. 23).

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 (PAMI)* (p. 78).

Balaji, Yogesh, Swami Sankaranarayanan, and Rama Chellappa (2018). "MetaReg: Towards Domain Generalization using Meta-Regularization." In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Vol. 31. Curran Associates, Inc., pp. 998–1008 (p. 84).

Baldi, Pierre, Kurt Hornik, D Touretzky, M Mozer, and M Hasselmo (1996). "Universal approximation and learning of trajectories using oscillators." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 451–457 (p. 34).

Balduzzi, David, Marcus Frean, Lennox Leary, J. P. Lewis, Kurt Wan-Duo Ma, and Brian McWilliams (2017). "The Shattered Gradients Problem: If resnets are the answer, then what is the question?" In: *Proceedings of the International Conference on Machine Learning (ICML)*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, pp. 342–350 (p. 28).

Ballas, Nicolas, Li Yao, Chris Pal, and Aaron C. Courville (2016). "Delving Deeper into Convolutional Networks for Learning Video Representations." In: *International Conference on Learning Representations (ICLR)* (p. 83).

Battenberg, Eric, Jitong Chen, Rewon Child, Adam Coates, Yashesh Gaur Yi Li, Hairong Liu, Sanjeev Satheesh, Anuroop Sriram, and Zhenyao Zhu (2017). "Exploring neural transducers for end-to-end speech recognition." In: *IEEE Automatic Speech Recognition and Understanding Workshop*. IEEE, pp. 206–213 (p. 30).

Baxter, Jonathan (1995). "Learning internal representations." In: *Proceedings of the eighth annual conference on Computational learning theory*, pp. 311–320 (pp. 4, 79).

Baxter, Jonathan (1998). "Theoretical models of learning to learn." In: *Learning to learn*. Springer, pp. 71–94 (p. 79).

Baxter, Jonathan (2000). "A model of inductive bias learning." In: *Journal of artificial intelligence research* 12, pp. 149–198 (p. 79).

Baydin, Atılım Günes, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind (2017). "Automatic differentiation in machine learning: a survey." In: *The Journal of Machine Learning Research* 18.1, pp. 5595–5637 (p. 18).

Bechtle, Sarah, Artem Molchanov, Yevgen Chebotar, Edward Grefenstette, Ludovic Righetti, Gaurav Sukhatme, and Franziska Meier (2021). "Meta Learning via Learned Loss." In: *Proceedings of the International Conference on Pattern Recognition (ICPR)* (p. 84).

Behrmann, Jens, Will Grathwohl, Ricky TQ Chen, David Duvenaud, and Jörn-Henrik Jacobsen (2019). "Invertible residual networks." In: *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR, pp. 573–582 (p. 98).

Bengio, Yoshua, Aaron Courville, and Pascal Vincent (2013). "Representation learning: A review and new perspectives." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 35.8, pp. 1798–1828 (pp. 16, 30).

Bengio, Yoshua, Paolo Frasconi, and Patrice Simard (1993). "The problem of learning long-term dependencies in recurrent networks." In: *Proceedings of the IEEE International Conference on Neural Networks*. IEEE, pp. 1183–1188 (p. 24).

Bengio, Yoshua, Patrice Simard, and Paolo Frasconi (1994). "Learning long-term dependencies with gradient descent is difficult." In: *Neural Networks* 5.2, pp. 157–166 (pp. 24, 30, 34).

Benz, Philipp, Chaoning Zhang, and In So Kweon (2021). *Batch Normalization Increases Adversarial Vulnerability: Disentangling Usefulness and Robustness of Model Features* (p. 23).

Berman, Maxim, Amal Rannen Triki, and Matthew B Blaschko (2018). "The Lovász-Softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in Neural Networks." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 88, 91).

Berner, Raphael, Christian Brandli, Minhao Yang, Shih-Chii Liu, and Tobi Delbruck (2013). "A 240× 180 10mW 12us latency sparse-output vision sensor for mobile applications." In: *2013 Symposium on VLSI Circuits*. IEEE, pp. C186–C187 (p. 52).

Bi, Yin, Aaron Chadha, Alhabib Abbas, Eirina Bourtsoulatze, and Yiannis Andreopoulos (2019). "Graph-Based Object Classification for Neuromorphic Vision Sensing." In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 491–501 (pp. 58, 67, 68).

Bishop, Christopher M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press (pp. 12, 17).

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning*. Springer (pp. 6, 10–12).

Blum, Avrim L. and Ronald L. Rivest (1992). "Training a 3-node Neural Network is NP-complete." In: *Neural Networks* 5.1, pp. 117–127 (p. 19).

Bottou, Léon (1998). "Online learning and stochastic approximations." In: *On-line learning in Neural Networks* 17.9, p. 142 (p. 19).

Botzheim, János, Takenori Obo, and Naoyuki Kubota (2012). "Human gesture recognition for robot partners by spiking Neural Network and classification learning." In: *The 6th International Conference on Soft Computing and Intelligent Systems, and The 13th International Symposium on Advanced Intelligence Systems*. IEEE, pp. 1954–1958 (p. 56).

Boyd, Stephen, Stephen P Boyd, and Lieven Vandenberghe (2004). *Convex optimization*. Cambridge university press (pp. 19, 20).

Brown, Noam and Tuomas Sandholm (2018). "Superhuman AI for heads-up no-limit poker: Libratus beats top professionals." In: *Science* 359.6374, pp. 418–424 (p. 16).

Byeon, Wonmin, Qin Wang, Rupesh Kumar Srivastava, and Petros Koumoutsakos (2018). "Contextvp: Fully context-aware video prediction." In: *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 753–769 (p. 26).

Caelles, Sergi, Kevis-Kokitsi Maninis, Jordi Pont-Tuset, Laura Leal-Taixé, Daniel Cremers, and Luc Van Gool (2017). "One-shot video object segmentation." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 221–230 (pp. 78, 82, 85, 89–91).

Caelles, Sergi, Alberto Montes, Kevis-Kokitsi Maninis, Yuhua Chen, Luc Van Gool, Federico Perazzi, and Jordi Pont-Tuset (2018). "The 2018 DAVIS Challenge on Video Object Segmentation." In: *arXiv:1803.00557* (pp. 78, 81, 82, 89).

Campbell, Murray, A Joseph Hoane Jr, and Feng-hsiung Hsu (2002). "Deep blue." In: *Artificial intelligence* 134.1-2, pp. 57–83 (p. 16).

Cannici, Marco, Marco Ciccone, Andrea Romanoni, and Matteo Matteucci (2019a). "Asynchronous Convolutional Networks for Object Detection in Neuromorphic Cameras." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 0–0 (pp. 54, 56).

Cannici, Marco, Marco Ciccone, Andrea Romanoni, and Matteo Matteucci (2019b). "Attention Mechanisms for Object Recognition with Event-Based Cameras." In: *Proceedings of the IEEE Winter Conference on Applications of Computer Vision*. IEEE, pp. 1127–1136 (pp. 56, 58).

Cannici, Marco, Marco Ciccone, Andrea Romanoni, and Matteo Matteucci (2020). "A Differentiable Recurrent Surface for Asynchronous Event-Based Data." In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, pp. 1–17 (p. 4).

Chang, Bo, Lili Meng, Eldad Haber, Frederick Tung, and David Begert (2018). "Multi-level Residual Networks from Dynamical Systems View." In: *International Conference on Learning Representations (ICLR)* (p. 34).

Chen, Liang-Chieh, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille (2017). "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 40.4, pp. 834–848 (p. 55).

Chen, Minmin, Jeffrey Pennington, and Samuel Schoenholz (2018a). "Dynamical Isometry and a Mean Field Theory of RNNs: Gating Enables Signal Propagation in Recurrent Neural Networks." In: *Proceedings of the International Conference on Machine Learning (ICML)*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, pp. 873–882 (p. 22).

Chen, Yuhua, Jordi Pont-Tuset, Alberto Montes, and Luc Van Gool (2018b). "Blazingly fast video object segmentation with pixel-wise metric learning." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1189–1198 (p. 83).

Cheng, Jingchun, Sifei Liu, Yi-Hsuan Tsai, Wei-Chih Hung, Shalini De Mello, Jinwei Gu, Jan Kautz, Shengjin Wang, and Ming-Hsuan Yang (2017). "Learning to Segment Instances in Videos with Spatial Propagation Network." In: *The 2017 DAVIS Challenge on Video Object Segmentation - CVPR Workshops* (p. 82).

Cheng, Jingchun, Yi-Hsuan Tsai, Wei-Chih Hung, Shengjin Wang, and Ming-Hsuan Yang (2018). "Fast and Accurate Online Video Object Segmentation via Tracking Parts." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (p. 82).

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: *Empirical Methods in Natural Language Processing (EMNLP)* (pp. 26, 33).

Choromanska, Anna, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun (2015). "The loss surfaces of multilayer networks." In: *Artificial intelligence and statistics*, pp. 192–204 (p. 19).

Ciccone, Marco, Marco Gallieri, Jonathan Masci, Christian Osendorfer, and Faustino Gomez (2018). "NAIS-Net: Stable Deep Networks from Non-Autonomous Differential Equations." In: *Advances in Neural Information Processing Systems (NeurIPS)* (p. 4).

Cisse, Moustapha, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier (2017). "Parseval Networks: Improving Robustness to Adversarial Examples." In: *Proceedings of the International Conference on Machine Learning (ICML)*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Sydney, Australia: PMLR, pp. 854–863 (pp. 35, 97).

Clevert, Djork-Arné, Thomas Unterthiner, and Sepp Hochreiter (2016). "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)." In: *International Conference on Learning Representations (ICLR)* (p. 23).

Cohen, Gregory Kevin (Sept. 2016). "Event-Based Feature Detection, Recognition and Classification." Theses. Université Pierre et Marie Curie - Paris VI (pp. 54, 56).

Cybenko, George (1989). "Approximation by superpositions of a sigmoidal function." In: *Mathematics of control, signals and systems* 2.4, pp. 303–314 (p. 13).

Dauphin, Yann, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio (2014). "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2933–2941 (pp. 19, 20).

Delalleau, Olivier and Yoshua Bengio (2011). "Shallow vs. deep sum-product networks." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 666–674 (p. 13).

Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei (2009). "Imagenet: A large-scale hierarchical image database." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Ieee, pp. 248–255 (pp. 64, 97).

Desjardins, Guillaume, Karen Simonyan, Razvan Pascanu, and Koray Kavukcuoglu (2015). "Natural Neural Networks." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2071–2079 (pp. 21, 23).

DiCarlo, James J, Davide Zoccolan, and Nicole C Rust (2012). "How does the brain solve visual object recognition?" In: *Neuron* 73, pp. 415–34 (p. 79).

Diehl, Peter U, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer (2015). "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing." In: *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–8 (p. 56).

Domingos, Pedro (1999). "The role of Occam's razor in knowledge discovery." In: *Data mining and knowledge discovery* 3.4, pp. 409–425 (p. 11).

Dosovitskiy, Alexey and Thomas Brox (2016). "Inverting visual representations with convolutional networks." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4829–4837 (p. 16).

Dosovitskiy, Alexey, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox (2015). "Flownet: Learning optical flow with convolutional networks." In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 2758–2766 (p. 28).

Doya, Kenji (1992). "Bifurcations in the learning of recurrent Neural Networks." In: *Circuits and Systems, 1992. ISCAS'92. Proceedings., 1992 IEEE International Symposium on*. Vol. 6. IEEE, pp. 2777–2780 (p. 34).

Drozdzal, Michal, Eugene Vorontsov, Gabriel Chartrand, Samuel Kadoury, and Chris Pal (2016). "The importance of skip connections in biomedical image segmentation." In: *Deep Learning and Data Labeling for Medical Applications*. Springer, pp. 179–187 (p. 28).

Duchi, John, Elad Hazan, and Yoram Singer (2011). "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of Machine Learning Research* 12.7 (pp. 12, 21).

Dumoulin, Vincent, Jonathon Shlens, and Manjunath Kudlur (2017). "A learned representation for artistic style." In: *International Conference on Learning Representations (ICLR)* (p. 85).

Dumoulin, Vincent and Francesco Visin (2016). "A guide to convolution arithmetic for deep learning." In: *ArXiv e-prints* (p. 15).

Duvenaud, David, Oren Rippel, Ryan Adams, and Zoubin Ghahramani (2014). "Avoiding pathologies in very deep networks." In: *Artificial Intelligence and Statistics (AISTATS)*, pp. 202–210 (pp. 27, 109).

Elman, Jeffrey L (1990). "Finding structure in time." In: *Cognitive science* 14.2, pp. 179–211 (p. 15).

Erhan, Dumitru, Yoshua Bengio, Aaron Courville, and Pascal Vincent (June 2009). "Visualizing Higher-Layer Features of a Deep Network." In: *Workshop on Learning Feature Hierarchies at International Conference on Machine Learning (ICML)* (p. 16).

Fahlman, Scott and Christian Lebiere (1990). "The Cascade-Correlation Learning Architecture." In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by D. Touretzky. Vol. 2. Morgan-Kaufmann (p. 27).

Fei-Fei, Li, Rob Fergus, and Pietro Perona (2007). "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories." In: *Computer Vision and Image Understanding* (p. 79).

Figurnov, Michael, Artem Sobolev, and Dmitry Vetrov (2017). "Probabilistic Adaptive Computation Time." In: *arXiv preprint arXiv:1712.00386* (p. 34).

Finn, Chelsea, Pieter Abbeel, and Sergey Levine (2017). "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks." In: *Proceedings of the International Conference on Machine Learning (ICML)*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, pp. 1126–1135 (pp. 84, 99).

Flennerhag, Sebastian, Andrei A. Rusu, Razvan Pascanu, Francesco Visin, Hujun Yin, and Raia Hadsell (2020). "Meta-Learning with Warped Gradient Descent." In: *International Conference on Learning Representations (ICLR)* (p. 84).

Fossum, Eric R (1997). "CMOS image sensors: Electronic camera-on-a-chip." In: *IEEE Transactions on Electron Devices* 44.10, pp. 1689–1698 (p. 52).

Franceschi, Luca, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil (2018). "Bilevel Programming for Hyperparameter Optimization and Meta-Learning." In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, pp. 1568–1577 (pp. 83, 84).

Fukushima, Kunihiko (1980). "Neocognitron: A self-organizing Neural Network model for a mechanism of Pattern Recognition unaffected by shift in position." In: *Biological cybernetics* 36.4, pp. 193–202 (p. 12).

Gallego, Guillermo et al. (2020). "Event-based vision: A survey." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, pp. 1–1 (p. 56).

Gallieri, Marco (2016). *LASSO-MPC – Predictive Control with ℓ_1-Regularised Least Squares*. Springer-Verlag (p. 101).

Garnelo, Marta, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and S. M. Ali Eslami (2018). "Conditional Neural Processes." In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, pp. 1704–1713 (p. 86).

Gehrig, Daniel, Antonio Loquercio, Konstantinos G. Derpanis, and Davide Scaramuzza (2019a). "End-to-End Learning of Representations for Asynchronous Event-Based Data." In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (pp. 54, 55, 57, 59–61, 64, 67, 68, 70, 71, 75, 77).

Gehrig, Daniel, Antonio Loquercio, Konstantinos G. Derpanis, and Davide Scaramuzza (2019b). *End-to-End Learning of Representations for Asynchronous Event-Based Data*. `https://github.com/uzh-rpg/rpg_event_representation_learning` (p. 64).

Gers, Felix A, Jürgen Schmidhuber, and Fred Cummins (1999). "Learning to forget: continual prediction with LSTM." In: *International Conference on Artificial Neural Networks (ICANN)*. Vol. 2, 850–855 vol.2 (p. 25).

Gers, Felix A, Nicol N Schraudolph, and Jürgen Schmidhuber (2002). "Learning precise timing with LSTM recurrent networks." In: *Journal of Machine Learning Research* 3.Aug, pp. 115–143 (p. 26).

Ghiasi, Golnaz, Honglak Lee, Manjunath Kudlur, Vincent Dumoulin, and Jonathon Shlens (2017). "Exploring the structure of a real-time, arbitrary neural artistic stylization network." In: *arXiv preprint arXiv:1705.06830* (p. 85).

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward Neural Networks." In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 249–256 (p. 22).

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio (2011). "Deep sparse rectifier Neural Networks." In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)* (pp. 14, 22).

Gomez, A., M. Ren, R. Urtasun, and R. B. Grosse (2017). "The Reversible Residual Network: Backpropagation without Storing Activations." In: *Advances in Neural Information Processing Systems (NeurIPS)* (pp. 31, 98).

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press (pp. 6, 12).

Gordon, Jonathan, John Bronskill, Matthias Bauer, Sebastian Nowozin, and Richard Turner (2019). "Meta-Learning Probabilistic Inference for Prediction." In: *International Conference on Learning Representations (ICLR)* (p. 86).

Graves, A. (2016). "Adaptive Computation Time for Recurrent Neural Networks." In: *CoRR* abs/1603.08983 (p. 34).

Graves, Alex (2008). "Supervised Sequence Labelling with Recurrent Neural Networks." PhD thesis. Technische Universität München (p. 26).

Graves, Alex, Santiago Fernández, and Jürgen Schmidhuber (2007). "Multi-dimensional recurrent Neural Networks." In: *International Conference on Artificial Neural Networks (ICANN)*. Springer, pp. 549–558 (p. 26).

Graves, Alex and Jürgen Schmidhuber (2005). "Framewise phoneme classification with bidirectional LSTM and other Neural Network architectures." In: *Neural Networks* 18.5-6, pp. 602–610 (p. 25).

Graves, Alex and Jürgen Schmidhuber (2009). "Offline handwriting recognition with multidimensional recurrent Neural Networks." In: *Advances in Neural Information Processing Systems (NeurIPS)* 21 (p. 26).

Greff, Klaus, Rupesh Kumar 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 (p. 26).

Greff, Klaus, Rupesh Kumar Srivastava, and Jürgen Schmidhuber (2017). "Highway and residual networks learn unrolled iterative estimation." In: *International Conference on Learning Representations (ICLR)* (pp. 4, 30, 34).

Gregor, Karol and Yann LeCun (2010). "Learning fast approximations of sparse coding." In: *Proceedings of the International Conference on Machine Learning (ICML)* (p. 34).

Griewank, Andreas and Andrea Walther (2008). *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM (p. 18).

Gulrajani, Ishaan, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville (2017). "Improved training of wasserstein gans." In: *Advances in Neural Information Processing Systems (NeurIPS)* 30, pp. 5767–5777 (p. 97).

Ha, David, Andrew Dai, and Quoc V Le (2017). "Hypernetworks." In: *International Conference on Learning Representations (ICLR)* (pp. 84, 99).

Haber, Eldad and Lars Ruthotto (2017). "Stable architectures for deep Neural Networks." In: *Inverse Problems* 34.1, p. 014004 (pp. 31, 34, 47).

Haber, Eldad, Lars Ruthotto, and Elliot Holtham (2018). "Learning across scales-A multiscale method for Convolution Neural Networks." In: *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 29, 34).

Hammer, Barbara and Thomas Villmann (2003). "Mathematical Aspects of Neural Networks." In: *ESANN*, pp. 59–72 (p. 19).

Hariharan, Bharath, Pablo Arbeláez, Ross Girshick, and Jitendra Malik (2015). "Hypercolumns for object segmentation and fine-grained localization." In: *CVPR*, pp. 447–456 (p. 85).

Haschke, Robert and Jochen J Steil (2005). "Input space bifurcation manifolds of recurrent Neural Networks." In: *Neurocomputing* 64, pp. 25–38 (p. 35).

He, Kaiming, Georgia Gkioxari, Piotr Dollár, and Ross Girshick (2017). "Mask r-cnn." In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 2961–2969 (pp. 55, 99).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2015). "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." In: pp. 1026–1034 (pp. 14, 22, 23, 48).

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 (CVPR)*, pp. 770–778 (pp. 3, 28, 30, 33, 55, 64, 67).

Hinton, Geoffrey E and David C Plaut (1987). "Using fast weights to deblur old memories." In: *Proceedings of the ninth annual conference of the Cognitive Science Society*, pp. 177–186 (pp. 79, 84).

Hinton, Geoffrey, Nitish Srivastava, and Kevin Swersky (2012). "Neural Networks for machine learning lecture 6a overview of mini-batch gradient descent." In: *Cited on* 14.8 (pp. 12, 21).

Hochreiter, Sepp (1991). *Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis*. Advisor: Jürgen Schmidhuber (pp. 24, 30, 34).

Hochreiter, Sepp and Jürgen Schmidhuber (1997a). "Flat minima." In: *Neural Computation* 9.1, pp. 1–42 (p. 19).

Hochreiter, Sepp and Jürgen Schmidhuber (1997b). "Long short-term memory." In: *Neural Computation* 9.8, pp. 1735–1780 (pp. 4, 15, 25, 33, 34, 54, 57, 59, 85).

Hochreiter, Sepp, A Steven Younger, and Peter R Conwell (2001). "Learning to learn using gradient descent." In: *International Conference on Artificial Neural Networks (ICANN)*. Springer, pp. 87–94 (p. 79).

Horn, R. A. and C. R. Johnson (2012). *Matrix Analysis*. 2nd. New York, NY, USA: Cambridge University Press (pp. 40, 112, 119).

Hornik, Kurt (1991). "Approximation capabilities of multilayer feedforward networks." In: *Neural Networks* 4.2, pp. 251–257 (pp. 13, 19).

Hornik, Kurt, Maxwell Stinchcombe, and Halbert White (1989). "Multilayer feedforward networks are universal approximators." In: *Neural networks* 2.5, pp. 359–366 (pp. 13, 19).

Hu, Jie, Li Shen, and Gang Sun (2018a). "Squeeze-and-excitation networks." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7132–7141 (pp. 60, 66).

Hu, Ping, Gang Wang, Xiangfei Kong, Jason Kuen, and Yap-Peng Tan (2018b). "Motion-Guided Cascaded Refinement Network for Video Object Segmentation." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (p. 82).

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 (CVPR)* (pp. 28, 32).

Huang, Gao, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger (2016). "Deep networks with stochastic depth." In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, pp. 646–661 (p. 28).

Ioffe, Sergey and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." In: *Proceedings of the International Conference on Machine Learning (ICML)* (pp. 21, 23, 33, 35).

Ioffe, Sergey and Christian Szegedy (2017). "Batch renormalization: Towards reducing minibatch dependence in batch-normalized models." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1945–1953 (p. 23).

Jacobsen, Jörn-Henrik, Arnold W.M. Smeulders, and Edouard Oyallon (2018). "i-RevNet: Deep Invertible Networks." In: *International Conference on Learning Representations (ICLR)* (p. 98).

Jaeger, Herbert (2001). "The "echo state" approach to analysing and training recurrent Neural Networks-with an erratum note." In: *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report* 148.34, p. 13 (p. 15).

Jaeger, Herbert (2002). "Adaptive nonlinear system identification with echo state networks." In: *Advances in Neural Information Processing Systems (NeurIPS)* (p. 15).

Jarrett, Kevin, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun (2009). "What is the best multi-stage architecture for object recognition?" In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE, pp. 2146–2153 (pp. 14, 22).

Jastrzebski, Stanisław, Devansh Arpit, Nicolas Ballas, Vikas Verma, Tong Che, and Yoshua Bengio (2018). "Residual Connections Encourage Iterative Inference." In: *International Conference on Learning Representations (ICLR)* (pp. 4, 30, 31, 34).

Jégou, Simon, Michal Drozdzal, David Vazquez, Adriana Romero, and Yoshua Bengio (2017). "The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 11–19 (p. 28).

Judd, J Stephen (1990). *Neural Network design and the complexity of learning*. MIT Press (p. 19).

Kalman, Barry L and Stan C Kwasny (1997). "High performance training of feedforward and simple recurrent networks." In: *Neurocomputing* 14.1, pp. 63–83 (p. 27).

Kalman, Barry L, Stan C Kwasny, and Aurorita Abella (1993). "Decomposing input patterns to facilitate training." In: *Proceedings of the World Congress on Neural Networks*. Vol. 3, pp. 503–506 (p. 27).

Kanai, Sekitoshi, Yasuhiro Fujiwara, and Sotetsu Iwamura (2017). "Preventing Gradient Explosions in Gated Recurrent Units." In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., pp. 435–444 (pp. 35, 38, 39).

Khalil, Hassan (2014). *Nonlinear Systems*. 3rd. Pearson Education (pp. 33, 35, 36, 101, 109).

Khoreva, Anna, Rodrigo Benenson, Eddy Ilg, Thomas Brox, and Bernt Schiele (2017a). "Lucid Data Dreaming for Object Tracking." In: *The 2017 DAVIS Challenge on Video Object Segmentation - CVPR Workshops* (pp. 79, 82).

Khoreva, Anna, Federico Perazzi, Rodrigo Benenson, Bernt Schiele, and Alexander Sorkine-Hornung (2017b). "Learning video object segmentation from static images." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2663–2672 (p. 81).

Kingma, Diederik P. and Jimmy Ba (2015). "Adam: A method for stochastic optimization." In: *International Conference on Learning Representations (ICLR)* (pp. 12, 21, 64, 88).

Klambauer, Günter, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter (2017). "Self-Normalizing Neural Networks." In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., pp. 971–980 (p. 23).

Knight, James N (2008). *Stability analysis of recurrent Neural Networks with applications*. Colorado State University (p. 35).

Krizhevsky, Alex and Geoffrey Hinton (2009). "Learning multiple layers of features from tiny images." In: *Master's thesis, Department of Computer Science, University of Toronto* (p. 47).

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional Neural Networks." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1097–1105 (pp. 30, 55).

Krueger, David and Roland Memisevic (2015). "Regularizing RNNs by Stabilizing Activations." In: *International Conference on Learning Representations (ICLR)* (p. 35).

Kruskal, William H and W Allen Wallis (1952). "Use of ranks in one-criterion variance analysis." In: *Journal of the American statistical Association* 47.260, pp. 583–621 (p. 50).

Lagorce, Xavier, Garrick Orchard, Francesco Galluppi, Bertram E Shi, and Ryad B Benosman (2016). "HOTS: a Hierarchy of Event-Based Time-Surfaces for Pattern Recognition." In: *IEEE Transactions on Pattern Analysis*

and Machine Intelligence (PAMI) 39.7, pp. 1346–1359 (pp. 54, 56, 57, 67, 68, 75).

Lake, Brenden, Ruslan Salakhutdinov, Jason Gross, and Joshua Tenenbaum (2011). "One shot learning of simple visual concepts." In: *Proceedings of the annual meeting of the cognitive science society*. Vol. 33. 33 (p. 79).

Lang, Kevin J, Alex H Waibel, and Geoffrey E Hinton (1990). "A time-delay Neural Network architecture for isolated word recognition." In: *Neural Networks* 3.1, pp. 23–43 (p. 15).

Lang, Kevin J and Michael J Witbrock (1988). "Learning to tell two spirals apart." In: *Proceedings of the 1988 connectionist models summer school*. San Mateo, pp. 52–59 (pp. 27, 32).

Larsson, Gustav, Michael Maire, and Gregory Shakhnarovich (2017). "FractalNet: Ultra-Deep Neural Networks without Residuals." In: *International Conference on Learning Representations (ICLR)* (pp. 28, 31).

Lattari, Francesco, Marco Ciccone, Matteo Matteucci, Jonathan Masci, and Francesco Visin (2018). "ReConvNet: Video Object Segmentation with Spatio-Temporal Features Modulation." In: *The 2018 DAVIS Challenge on Video Object Segmentation - CVPR Workshops* (p. 5).

Laurent, Thomas and James von Brecht (2017). "A Recurrent Neural Network without Chaos." In: *International Conference on Learning Representations (ICLR)* (p. 35).

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998a). "Gradient-Based Learning Applied to Document Recognition." In: *Proceedings of the IEEE* 86.11, pp. 2278–2324 (p. 14).

LeCun, Yann A, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller (1998b). "Efficient backprop." In: *Neural Networks: Tricks of the trade*. Ed. by G. Orr and Muller K. Springer, pp. 9–48 (pp. 20, 23).

LeCun, Yann (1989). "Generalization and network design strategies." In: *Connectionism in perspective* 19, pp. 143–155 (p. 14).

LeCun, Yann (1998). "The MNIST database of handwritten digits." In: *http://yann.lecun.com/exdb/mnist/* (p. 46).

LeCun, Yann, Ido Kanter, and Sara A Solla (1991). "Eigenvalues of covariance matrices: Application to neural-network learning." In: *Physical Review Letters* 66.18, p. 2396 (p. 23).

Lee, Jun Haeng, Tobi Delbruck, and Michael Pfeiffer (2016). "Training Deep Spiking Neural Networks Using Backpropagation." In: *Frontiers in Neuroscience* 10, p. 508 (p. 56).

Lee, Samuel E and Bradley R Holt (1992). "Regression analysis of spectroscopic process data using a combined architecture of linear and nonlinear artificial Neural Networks." In: *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*. Vol. 4. IEEE, pp. 549–554 (p. 27).

Lee, Sang Wan, John P O'Doherty, and Shinsuke Shimojo (2015). "Neural Computations mediating one-shot learning in the human brain." In: *PLoS biology* 13.4, e1002137 (p. 79).

Li Fei-Fei, R. Fergus, and P. Perona (2006). "One-shot learning of object categories." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 28.4, pp. 594–611 (pp. 63, 79).

Li, Hao, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein (2018a). "Visualizing the loss landscape of neural nets." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 6389–6399 (pp. 19, 29).

Li, Qianxiao, Long Chen, Cheng Tai, and Weinan E (2018b). "Maximum Principle Based Algorithms for Deep Learning." In: *Journal of Machine Learning Research* 18.165, pp. 1–29 (p. 34).

Li, Xiaoxiao, Yuankai Qi, Zhe Wang, Kai Chen, Ziwei Liu, Jianping Shi, Ping Luo, Xiaoou Tang, and Chen Change Loy (2017a). "Video Object Segmentation with Re-identification." In: *The 2017 DAVIS Challenge on Video Object Segmentation - CVPR Workshops* (p. 82).

Li, Yiying, Yongxin Yang, Wei Zhou, and Timothy Hospedales (2019). "Feature-Critic Networks for Heterogeneous Domain Generalization." In: *Proceedings of the 36th International Conference on Machine Learning (ICML)*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. Long Beach, California, USA: PMLR, pp. 3915–3924 (p. 84).

Li, Zhenguo, Fengwei Zhou, Fei Chen, and Hang Li (2017b). "Meta-sgd: Learning to learn quickly for few-shot learning." In: *arXiv preprint arXiv: 1707.09835* (p. 84).

Liao, Qianli and Tomaso Poggio (2016). "Bridging the gaps between residual learning, recurrent Neural Networks and visual cortex." In: *arXiv preprint arXiv:1604.03640* (p. 34).

Lichtsteiner, Patrick, Christoph Posch, and Tobi Delbruck (2008). "A 128 × 128 120 dB 15μs Latency Asynchronous Temporal Contrast Vision Sensor." In: *IEEE journal of solid-state circuits* 43.2, pp. 566–576 (p. 52).

Lin, Tsung-Yi, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár (2017). "Focal loss for dense object detection." In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 2980–2988 (p. 99).

Lin, Tsung-Yi, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick (2014). "Microsoft COCO: Common Objects in Context." In: *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 87, 91).

Liu, Wei, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg (2016). "Ssd: Single shot multibox detector." In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, pp. 21–37 (p. 55).

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 (CVPR)*, pp. 3431–3440 (pp. 15, 28, 55, 61, 78, 82).

Lorraine, Jonathan, Paul Vicol, and David Duvenaud (2020). "Optimizing Millions of Hyperparameters by Implicit Differentiation." In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*. Ed. by Silvia Chiappa and Roberto Calandra. Vol. 108. Proceedings of Machine Learning Research. Online: PMLR, pp. 1540–1552 (p. 84).

Lu, Yiping, Aoxiao Zhong, Quanzheng Li, and Bin Dong (2018). *Beyond Finite Layer Neural Networks: Bridging Deep Architectures and Numerical Differential Equations* (pp. 31, 34).

Ly, Alexander, Maarten Marsman, Josine Verhagen, Raoul PPP Grasman, and Eric-Jan Wagenmakers (2017). "A tutorial on Fisher information." In: *Journal of Mathematical Psychology* 80, pp. 40–55 (p. 21).

Maas, Andrew L, Awni Y Hannun, and Andrew Y Ng (2013). "Rectifier nonlinearities improve Neural Network acoustic models." In: *Proceedings of the International Conference on Machine Learning (ICML)*. Vol. 30. 1. Citeseer, p. 3 (p. 23).

Maass, Wolfgang (1997). "Networks of spiking neurons: the third generation of Neural Network models." In: *Neural Networks* 10.9, pp. 1659–1671 (pp. 56, 57).

Maass, Wolfgang, Thomas Natschläger, and Henry Markram (2002). "Real-time computing without stable states: A new framework for neural computation based on perturbations." In: *Neural Computation* 14.11, pp. 2531–2560 (p. 15).

MacKay, David JC (2003). *Information theory, Inference and Learning Algorithms*. Cambridge University Press (p. 11).

Mahendran, Aravindh and Andrea Vedaldi (2015). "Understanding deep image representations by inverting them." In: *Proceedings of the IEEE*

Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5188–5196 (p. 16).

Maninis, Kevis-Kokitsi, Sergi Caelles, Yuhua Chen, Jordi Pont-Tuset, Laura Leal-Taixé, Daniel Cremers, and Luc Van Gool (2018). "Video object segmentation without temporal information." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 41.6, pp. 1515–1530 (pp. 89, 90).

Mao, Xiaojiao, Chunhua Shen, and Yu-Bin Yang (2016). "Image Restoration Using Very Deep Convolutional Encoder-Decoder Networks with Symmetric Skip Connections." In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett. Vol. 29. Curran Associates, Inc., pp. 2802–2810 (p. 28).

Maqueda, Ana I, Antonio Loquercio, Guillermo Gallego, Narciso García, and Davide Scaramuzza (2018). "Event-based vision meets deep learning on steering prediction for self-driving cars." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5419–5427 (pp. 56, 57, 70).

Martens, James (2020). "New Insights and Perspectives on the Natural Gradient Method." In: *Journal of Machine Learning Research* 21.146, pp. 1–76 (p. 21).

Martens, James and Roger Grosse (2015). "Optimizing Neural Networks with kronecker-factored approximate curvature." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 2408–2417 (p. 21).

Maunsell, JH and DAVID C van Essen (1983). "The connections of the middle temporal visual area (MT) and their relationship to a cortical hierarchy in the macaque monkey." In: *Journal of Neuroscience* 3.12, pp. 2563–2586 (p. 28).

Meftah, Boudjelal, Olivier Lezoray, and Abdelkader Benyettou (2010). "Segmentation and edge detection based on spiking Neural Network model." In: *Neural Processing Letters* 32.2, pp. 131–146 (p. 56).

Menze, Moritz and Andreas Geiger (2015). "Object Scene Flow for Autonomous Vehicles." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (p. 69).

Minsky, Marvin (1961). "Steps toward artificial intelligence." In: *Proceedings of the IRE* 49.1, pp. 8–30 (p. 16).

Mitchell, Tom (1997). *Machine Learning*. New York, USA: McGraw-Hill Higher Education (p. 6).

Miyato, Takeru, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida (2018). "Spectral Normalization for Generative Adversarial Networks." In: *International Conference on Learning Representations (ICLR)* (pp. 35, 97).

Monti, Federico, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein (2017). "Geometric deep learning on graphs and manifolds using mixture model cnns." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5115–5124 (p. 30).

Montufar, Guido F., Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio (2014). "On the Number of Linear Regions of Deep Neural Networks." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2924–2932 (p. 13).

Murphy, Kevin P (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press (pp. 6, 10).

Nair, Vinod and Geoffrey E Hinton (2010). "Rectified linear units improve restricted Boltzmann machines." In: *Proceedings of the International Conference on Machine Learning (ICML)* (pp. 14, 22).

Neal, Radford M (1995). "Bayesian learning for Neural Networks." PhD thesis (p. 27).

Neil, Daniel, Michael Pfeiffer, and Shih-Chii Liu (2016). "Phased LSTM: Accelerating recurrent network training for long or event-based sequences." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 3882–3890 (pp. 26, 58, 59).

Nesterov, Yurii (1983a). "A method for unconstrained convex minimization problem with the rate of convergence O (1/kˆ 2)." In: *Doklady an ussr*. Vol. 269, pp. 543–547 (p. 12).

Nesterov, Yurii (1983b). "A method of solving a convex programming problem with convergence rate O($1/k^2$)." In: *Soviet Mathematics Doklady* 27.2, pp. 372–376 (p. 19).

Newswanger, Amos and Chenliang Xu (2017). "One-Shot Video Object Segmentation with Iterative Online Fine-Tuning." In: *The 2017 DAVIS Challenge on Video Object Segmentation - CVPR Workshops* (p. 82).

Nguyen, Xuan-Son Trinh, Quang-Hieu Dinh, Vinh-Tiep Nguyen, Anh-Duc Duong, Akihiro Sugimoto, Tam V Nguyen, and Minh-Triet Tran (2017). "Instance Re-Identification Flow for Video Object Segmentation." In: *The 2017 DAVIS Challenge on Video Object Segmentation - CVPR Workshops* (p. 82).

Nocedal, Jorge and Stephen Wright (2006). *Numerical Optimization*. Springer verlag (p. 20).

Orchard, Garrick, Ajinkya Jayawant, Gregory K Cohen, and Nitish Thakor (2015a). "Converting static image datasets to spiking neuromorphic datasets using saccades." In: *Frontiers in neuroscience* 9, p. 437 (pp. 55, 63, 67, 68).

Orchard, Garrick, Cedric Meyer, Ralph Etienne-Cummings, Christoph Posch, Nitish Thakor, and Ryad Benosman (2015b). "HFirst: a temporal approach to object recognition." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 37.10, pp. 2028–2040 (p. 67).

Orhan, Emin and Xaq Pitkow (2018). "Skip Connections Eliminate Singularities." In: *International Conference on Learning Representations (ICLR)* (p. 29).

Park, Eunbyung and Junier B Oliva (2019). "Meta-curvature." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 3314–3324 (p. 84).

Pascanu, Razvan and Yoshua Bengio (2014). "Revisiting Natural Gradient for Deep Networks." In: *International Conference on Learning Representations (ICLR)* (p. 21).

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (2013). "On the difficulty of training recurrent Neural Networks." In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 1310–1318 (pp. 24, 34, 35, 41).

Pearlmutter, Barak A (1995). "Gradient calculations for dynamic recurrent Neural Networks: A survey." In: *IEEE Transactions on Neural Networks* 6.5, pp. 1212–1228 (p. 15).

Pennington, Jeffrey, Samuel Schoenholz, and Surya Ganguli (2017). "Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 4785–4795 (p. 22).

Pennington, Jeffrey, Samuel Schoenholz, and Surya Ganguli (2018). "The emergence of spectral universality in deep networks." In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*. Ed. by Amos Storkey and Fernando Perez-Cruz. Vol. 84. Proceedings of Machine Learning Research. Playa Blanca, Lanzarote, Canary Islands: PMLR, pp. 1924–1932 (p. 22).

Perazzi, Federico, Jordi Pont-Tuset, Brian McWilliams, Luc Van Gool, Markus Gross, and Alexander Sorkine-Hornung (2016). "A Benchmark Dataset and Evaluation Methodology for Video Object Segmentation." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 724–732 (p. 81).

Perez, Ethan, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville (2018). "FiLM: Visual Reasoning with a General Conditioning Layer." In: *Proceedings of the AAAI Conference on Artificial Intelligence* (p. 85).

Perot, Etienne, Pierre de Tournemire, Davide Nitti, Jonathan Masci, and Amos Sironi (2020). "Learning to Detect Objects with a 1 Megapixel Event

Camera." In: *Advances in Neural Information Processing Systems (NeurIPS)* 33 (p. 98).

Polyak, Boris T (1964). "Some methods of speeding up the convergence of iteration methods." In: *USSR Computational Mathematics and Mathematical Physics* 4.5, pp. 1–17 (p. 19).

Pont-Tuset, Jordi, Federico Perazzi, Sergi Caelles, Pablo Arbeláez, Alexander Sorkine-Hornung, and Luc Van Gool (2017). "The 2017 DAVIS Challenge on Video Object Segmentation." In: *arXiv:1704.00675* (p. 81).

Poole, Ben, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli (2016). "Exponential expressivity in deep Neural Networks through transient chaos." In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett. Vol. 29. Curran Associates, Inc., pp. 3360–3368 (p. 22).

Posch, Christoph, Teresa Serrano-Gotarredona, Bernabe Linares-Barranco, and Tobi Delbruck (2014). "Retinomorphic event-based vision sensors: bioinspired cameras with spiking output." In: *Proceedings of the IEEE* 102.10, pp. 1470–1484 (p. 52).

Raghu, Aniruddh, Maithra Raghu, Samy Bengio, and Oriol Vinyals (2020). "Rapid Learning or Feature Reuse? Towards Understanding the Effectiveness of MAML." In: *International Conference on Learning Representations (ICLR)* (p. 84).

Raghu, Maithra, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein (2017). "On the Expressive Power of Deep Neural Networks." In: *Proceedings of the International Conference on Machine Learning (ICML)*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, pp. 2847–2854 (p. 22).

Raiko, Tapani, Harri Valpola, and Yann LeCun (2012). "Deep learning made easier by linear transformations in perceptrons." In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 924–932 (p. 23).

Rajeswaran, Aravind, Chelsea Finn, Sham M Kakade, and Sergey Levine (2019). "Meta-learning with implicit gradients." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 113–124 (p. 84).

Ravi, Sachin and Hugo Larochelle (2017). "Optimization as a model for few-shot learning." In: *International Conference on Learning Representations (ICLR)* (p. 84).

Rebecq, Henri, René Ranftl, Vladlen Koltun, and Davide Scaramuzza (2019). "Events-to-video: Bringing modern computer vision to event cameras." In:

Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3857–3866 (pp. 54, 57, 59, 60, 64, 65, 67, 68, 98).

Redmon, Joseph, Santosh Divvala, Ross Girshick, and Ali Farhadi (2016). "You Only Look Once: Unified, Real-Time Object Detection." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788 (p. 55).

Redmon, Joseph and Ali Farhadi (2018). "Yolov3: An incremental improvement." In: *arXiv preprint arXiv:1804.02767* (p. 99).

Requeima, James, Jonathan Gordon, John Bronskill, Sebastian Nowozin, and Richard E Turner (2019). "Fast and flexible multi-task classification using conditional neural adaptive processes." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 7959–7970 (p. 86).

Rezende, Danilo Jimenez and Shakir Mohamed (2015). "Variational Inference with Normalizing Flows." In: *Proceedings of the International Conference on Machine Learning (ICML)*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, pp. 1530–1538 (p. 98).

Robbins, Herbert and Sutton Monro (1951). "A stochastic approximation method." In: *The annals of mathematical statistics*, pp. 400–407 (pp. 11, 19).

Rocha, Ana Maria AC and Edite MGP Fernandes (2010). "A Stochastic Augmented Lagrangian Equality Constrained-Based Algorithm for Global Optimization." In: *AIP Conference Proceedings*. Vol. 1281. 1. American Institute of Physics, pp. 967–970 (p. 97).

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). "U-net: Convolutional networks for biomedical image segmentation." In: *International Conference on Medical image computing and computer-assisted intervention*. Springer, pp. 234–241 (p. 28).

Rosenblatt, Frank (1958). "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, p. 386 (p. 12).

Roux, Nicolas, Pierre-antoine Manzagol, and Yoshua Bengio (2008). "Topmoumoute Online Natural Gradient Algorithm." In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by J. Platt, D. Koller, Y. Singer, and S. Roweis. Vol. 20. Curran Associates, Inc., pp. 849–856 (p. 21).

Rueckauer, Bodo, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu (2017). "Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification." In: *Frontiers in Neuroscience* 11, p. 682 (p. 56).

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). "Learning representations by back-propagating errors." In: *Nature* 323.6088, pp. 533–536 (pp. 16, 17, 30).

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 (NeurIPS)* 29, pp. 901–909 (p. 21).

Santurkar, Shibani, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry (2018). "How does batch normalization help optimization?" In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2483–2493 (p. 23).

Saxe, Andrew M., James L. McClelland, and Surya Ganguli (2014). "Exact solutions to the nonlinear dynamics of learning in deep linear Neural Networks." In: *International Conference on Learning Representations (ICLR)* (pp. 22, 88).

Schaul, Tom, Sixin Zhang, and Yann LeCun (2013). "No more pesky learning rates." In: *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR, pp. 343–351 (p. 21).

Scheerlinck, Cedric, Henri Rebecq, Timo Stoffregen, Nick Barnes, Robert Mahony, and Davide Scaramuzza (2019). "CED: Color event camera dataset." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 0–0 (p. 59).

Schmidhuber, Jürgen (1987). "Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook." PhD thesis. Technische Universität München (pp. 4, 79, 83).

Schmidhuber, Jürgen (1992). "Learning to control fast-weight memories: An alternative to dynamic recurrent networks." In: *Neural Computation* 4.1, pp. 131–139 (pp. 79, 84).

Schmidhuber, Jürgen (2015). "Deep learning in Neural Networks: An overview." In: *Neural Networks* 61, pp. 85–117 (pp. 12, 17).

Schoenholz, Samuel S, Jeffrey Pennington, and Jascha Sohl-Dickstein (2017). "A correspondence between random Neural Networks and statistical field theory." In: *arXiv preprint arXiv:1710.06570* (p. 22).

Schraudolph, Nicol N (1998). "Centering Neural Network gradient factors." In: *Neural Networks: Tricks of the Trade*. Springer, pp. 207–226 (p. 23).

Sekikawa, Yusuke, Kosuke Hara, and Hideo Saito (2019). "EventNet: Asynchronous recursive event processing." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3887–3896 (p. 58).

Serrano Gotarredona, Teresa and Bernabé Linares Barranco (2013). "A 128 × 128 1.5% Contrast Sensitivity 0.9% FPN 3μs Latency 4mW Asynchronous Frame-Free Dynamic Vision Sensor Using Transimpedance Preamplifiers." In: *IEEE Journal of Solid-State Circuits* 48.3, pp. 827–838 (p. 52).

Shaban, Amirreza, Alrik Firl, Ahmad Humayun, Jialin Yuan, Xinyao Wang, Peng Lei, Nikhil Dhanda, Byron Boots, James M Rehg, and Fuxin Li (2017). "Multiple-Instance Video Segmentation with Sequence-Specific Object Proposals." In: *The 2017 DAVIS Challenge on Video Object Segmentation - CVPR Workshops* (p. 82).

Shalev-Shwartz, Shai and Shai Ben-David (2014). *Understanding machine learning: From theory to algorithms*. Cambridge university press (pp. 6, 9).

Sharir, Gilad, Eddie Smolyansky, and Itamar Friedman (2017). "Video Object Segmentation using Tracked Object Proposals." In: *The 2017 DAVIS Challenge on Video Object Segmentation - CVPR Workshops* (p. 82).

Shi, Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun Woo (2015). "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting." In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett. Curran Associates, Inc., pp. 802–810 (pp. 26, 55, 58, 61, 73).

Shimodaira, Hidetoshi (2000). "Improving predictive inference under covariate shift by weighting the log-likelihood function." In: *Journal of statistical planning and inference* 90.2, pp. 227–244 (p. 23).

Siegelmann, Hava T and Eduardo D Sontag (1995). "On the computational power of neural nets." In: *Journal of computer and system sciences* 50.1, pp. 132–150 (pp. 13, 15).

Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. (2016). "Mastering the game of Go with deep Neural Networks and tree search." In: *nature* 529.7587, pp. 484–489 (p. 16).

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. (2018). "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." In: *Science* 362.6419, pp. 1140–1144 (p. 16).

Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. (2017). "Mastering the game of go without human knowledge." In: *nature* 550.7676, pp. 354–359 (p. 16).

Simonyan, Karen and Andrew Zisserman (2015). "Very deep convolutional networks for large-scale image recognition." In: *International Conference on Learning Representations (ICLR)* (pp. 15, 16, 82, 85).

Singh, Jayant and Nikita Barabanov (2016). "Stability of discrete time recurrent Neural Networks and nonlinear optimization problems." In: *Neural Networks* 74, pp. 58–72 (p. 35).

Sironi, Amos, Manuele Brambilla, Nicolas Bourdis, Xavier Lagorce, and Ryad Benosman (2018). "HATS: Histograms of averaged time surfaces for robust event-based object classification." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1731–1740 (pp. 52, 54–57, 63, 67, 68, 73, 75).

Sokolić, Jure, Raja Giryes, Guillermo Sapiro, and Miguel RD Rodrigues (2017). "Robust large margin deep Neural Networks." In: *IEEE Transactions on Signal Processing* 65.16, pp. 4265–4280 (p. 97).

Sontag, Eduardo D (1998). *Mathematical Control Theory: Deterministic Finite Dimensional Systems*. 2nd. Springer-Verlag (pp. 33, 102, 109).

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." In: *The journal of machine learning research* 15.1, pp. 1929–1958 (p. 23).

Srivastava, Rupesh Kumar (2018). "New architectures for very deep learning." PhD thesis. Università della Svizzera italiana (p. 27).

Srivastava, Rupesh Kumar, Klaus Greff, and Jürgen Schmidhuber (2015a). "Highway networks." In: *arXiv preprint arXiv:1505.00387* (pp. 3, 27, 30, 33).

Srivastava, Rupesh Kumar, Klaus Greff, and Jürgen Schmidhuber (2015b). "Training Very Deep Networks." In: *Advances in Neural Information Processing Systems (NeurIPS)* (p. 27).

Stackelberg, Heinrich von (1952). *Theory of the Market Economy*. William Hodge (p. 84).

Steil, Jochen J (1999). *Input Output Stability of Recurrent Neural Networks*. Cuvillier Göttingen (p. 35).

Steiner, Benoit, Zachary DeVito, Soumith Chintala, Sam Gross, Adam Paszke, Francisco Massa, Adam Lerer, Gregory Chanan, Zeming Lin, Edward Yang, et al. (2019). "PyTorch: An imperative style, high-performance deep learning library." In: *Advances in Neural Information Processing Systems (NeurIPS)* 32 (pp. 55, 74).

Stollenga, Marijn F, Wonmin Byeon, Marcus Liwicki, and Juergen Schmidhuber (2015). "Parallel multi-dimensional lstm, with application to fast

biomedical volumetric image segmentation." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2998–3006 (p. 26).

Stollenga, Marijn F, Jonathan Masci, Faustino Gomez, and Jürgen Schmidhuber (2014). "Deep Networks with Internal Selective Attention through Feedback Connections." In: *Advances in Neural Information Processing Systems (NeurIPS)* (p. 85).

Strogatz, Steven H. (2015). *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. 2nd. Westview Press (pp. 31, 33, 109).

Su, Jiahao, Wonmin Byeon, Furong Huang, Jan Kautz, and Animashree Anandkumar (2020). "Convolutional Tensor-Train LSTM for Spatio-temporal Learning." In: *Advances in Neural Information Processing Systems (NeurIPS)* (p. 26).

Sun, Jian, Jiaya Jia, Chi-Keung Tang, and Heung-Yeung Shum (2004). "Poisson Matting." In: *ACM SIGGRAPH 2004 Papers*. SIGGRAPH '04. Los Angeles, California: ACM, pp. 315–321 (p. 82).

Sussillo, David and LF Abbott (2014). "Random walk initialization for training very deep feedforward networks." In: *arXiv preprint arXiv:1412.6558* (p. 22).

Sutskever, Ilya, Oriol Vinyals, and Quoc V Le (2014). "Sequence to sequence learning with Neural Networks." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 3104–3112 (p. 30).

Sutton, Richard S and Andrew G Barto (1998). *Reinforcement learning: An introduction*. Vol. 28. MIT press (pp. 12, 17).

Sutton, Richard (1986). "Two problems with back propagation and other steepest descent learning procedures for networks." In: *Proceedings of the Eighth Annual Conference of the Cognitive Science Society, 1986*, pp. 823–832 (p. 19).

Szegedy, Christian, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna (2016). "Rethinking the inception architecture for computer vision." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826 (pp. 55, 56).

Szegedy, Christian, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus (2014). "Intriguing properties of Neural Networks." In: *International Conference on Learning Representations (ICLR)* (p. 35).

Tallec, Corentin and Yann Ollivier (2018). "Can recurrent Neural Networks warp time?" In: *International Conference on Learning Representations (ICLR)* (p. 33).

Thrun, Sebastian and Lorien Pratt (1998). "Learning to learn: Introduction and overview." In: *Learning to learn*. Springer, pp. 3–17 (p. 79).

Tokmakov, P., K. Alahari, and C. Schmid (2017). "Learning Video Object Segmentation with Visual Memory." In: *Proceedings of the IEEE International Conference on Computer Vision* (p. 82).

Tsuzuku, Yusuke, Issei Sato, and Masashi Sugiyama (2018). "Lipschitz-margin training: Scalable certification of perturbation invariance for deep Neural Networks." In: *Advances in Neural Information Processing Systems (NeurIPS)* 31, pp. 6541–6550 (p. 97).

Ulyanov, Dmitry, Andrea Vedaldi, and Victor Lempitsky (2016). "Instance normalization: The missing ingredient for fast stylization." In: *arXiv preprint arXiv:1607.08022* (p. 23).

Vapnik, Vladimir (1992). "Principles of risk minimization for learning theory." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 831–838 (pp. 8, 9).

Vapnik, Vladimir (1998). *Statistical Learning Theory*. Wiley (pp. 8, 9).

Veit, Andreas and Serge Belongie (2018). "Convolutional Networks with Adaptive Inference Graphs." In: *Proceedings of the European Conference on Computer Vision (ECCV)* (p. 34).

Vinyals, Oriol, Charles Blundell, Timothy Lillicrap, and Daan Wierstra (2016). "Matching Networks for One Shot Learning." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 3630–3638 (p. 80).

Visin, Francesco, Marco Ciccone, Adriana Romero, Kyle Kastner, Kyunghyun Cho, Yoshua Bengio, Matteo Matteucci, and Aaron Courville (2016). "Reseg: A recurrent Neural Network-based model for semantic segmentation." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 41–48 (pp. 26, 78).

Visin, Francesco, Kyle Kastner, Kyunghyun Cho, Matteo Matteucci, Aaron Courville, and Yoshua Bengio (2015). "Renet: A recurrent Neural Network based alternative to convolutional networks." In: *arXiv preprint arXiv:1505.00393* (p. 26).

Voigtlaender, Paul and Bastian Leibe (2017a). "Online Adaptation of Convolutional Neural Networks for Video Object Segmentation." In: *Proceedings of the British Machine Vision Conference (BMVC)* (pp. 78, 82, 89, 90).

Voigtlaender, Paul and Bastian Leibe (2017b). "Online Adaptation of Convolutional Neural Networks for the 2017 DAVIS Challenge on Video Object Segmentation." In: *The 2017 DAVIS Challenge on Video Object Segmentation - CVPR Workshops* (p. 82).

Vorontsov, Eugene, Chiheb Trabelsi, Samuel Kadoury, and Chris Pal (2017). "On orthogonality and learning recurrent networks with long term dependencies." In: Proceedings of Machine Learning Research 70. Ed. by Doina Precup and Yee Whye Teh, pp. 3570–3578 (p. 35).

Vuorio, Risto, Shao-Hua Sun, Hexiang Hu, and Joseph J Lim (2019). "Multimodal Model-Agnostic Meta-Learning via Task-Aware Modulation." In: *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1–12 (p. 84).

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 (p. 15).

Wang, I-Jeng and James C Spall (2003). "Stochastic optimization with inequality constraints using simultaneous perturbations and penalty functions." In: *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*. Vol. 4. IEEE, pp. 3808–3813 (p. 97).

Wang, Qinyi, Yexin Zhang, Junsong Yuan, and Yilong Lu (2019). "Space-Time Event Clouds for Gesture Recognition: From RGB Cameras to Event Cameras." In: *Proceedings of the IEEE Winter Conference on Applications of Computer Vision*. IEEE, pp. 1826–1835 (p. 58).

Weinan, E. (2017). "A proposal on machine learning via dynamical systems." In: *Communications in Mathematics and Statistics* 5.1, pp. 1–11 (pp. 29, 34).

Werbos, Paul J. (1974). "Beyond regression: New tools for prediction and analysis in the behavioral sciences." PhD thesis. Cambridge, MA, USA: Harvard University (p. 17).

Werbos, Paul J. (1981). "Applications of Advances in Nonlinear Sensitivity Analysis." In: *Proceedings of the 10th IFIP Conference, 31.8 - 4.9, NYC*, pp. 762–770 (p. 17).

Werbos, Paul J. (1988). "Generalization of backpropagation with application to a recurrent gas market model." In: *Neural Networks* 1.4, pp. 339–356 (pp. 15, 17).

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 (p. 15).

Wu, Pei Yuan (1988). "Products of Positive Semidefinite Matrices." In: *Linear Algebra and Its Applications* (p. 119).

Wu, QingXiang, Martin McGinnity, Liam Maguire, Ammar Belatreche, and Brendan Glackin (2007). "Edge detection based on spiking Neural Network model." In: *International Conference on Intelligent Computing*. Springer, pp. 26–34 (p. 56).

Wu, Yuxin and Kaiming He (2018). "Group normalization." In: *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3–19 (p. 23).

Wug Oh, Seoung, Joon-Young Lee, Kalyan Sunkavalli, and Seon Joo Kim (2018). "Fast Video Object Segmentation by Reference-Guided Mask Propagation." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (p. 83).

Xiao, Huaxin, Jiashi Feng, Guosheng Lin, Yu Liu, and Maojun Zhang (2018a). "MoNet: Deep Motion Exploitation for Video Object Segmentation." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (p. 82).

Xiao, Lechao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel Schoenholz, and Jeffrey Pennington (2018b). "Dynamical Isometry and a Mean Field Theory of CNNs: How to Train 10,000-Layer Vanilla Convolutional Neural Networks." In: *Proceedings of the International Conference on Machine Learning (ICML)*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, pp. 5393–5402 (p. 22).

Xie, Saining, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He (2017). "Aggregated residual transformations for deep Neural Networks." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1492–1500 (p. 28).

Xingjian, SHI, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo (2015). "Convolutional LSTM network: A machine learning approach for precipitation nowcasting." In: *Advances in Neural Information Processing Systems (NeurIPS)* (p. 86).

Yang, Greg, Jeffrey Pennington, Vinay Rao, Jascha Sohl-Dickstein, and Samuel S. Schoenholz (2019). "A Mean Field Theory of Batch Normalization." In: *International Conference on Learning Representations (ICLR)* (p. 23).

Yang, Linjie, Yanran Wang, Xuehan Xiong, Jianchao Yang, and Aggelos K. Katsaggelos (2018). "Efficient Video Object Segmentation via Network Modulation." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 80, 85, 88, 90, 91).

Ye, Chengxi, Anton Mitrokhin, Cornelia Fermüller, James A Yorke, and Yiannis Aloimonos (2018). "Unsupervised Learning of Dense Optical Flow, Depth and Egomotion from Sparse Event Data." In: *arXiv preprint arXiv:1809.08625* (pp. 57, 98).

Yoshida, Yuichi and Takeru Miyato (2017). "Spectral Norm Regularization for Improving the Generalizability of Deep Learning." In: *arXiv preprint arXiv:1705.10941* (pp. 35, 39, 97).

Yosinski, Jason, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson (2015). "Understanding Neural Networks through deep visualization." In: *Deep Learning Workshop at International Conference on Machine Learning (ICML)* (p. 16).

Yu, Fisher and Vladlen Koltun (2016). "Multi-Scale Context Aggregation by Dilated Convolutions." In: *International Conference on Learning Representations (ICLR)* (p. 55).

Zeiler, Matthew D (2012). "Adadelta: an adaptive learning rate method." In: *arXiv preprint arXiv:1212.5701* (pp. 12, 21).

Zeiler, Matthew D and Rob Fergus (2014). "Visualizing and understanding convolutional networks." In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, pp. 818–833 (pp. 15, 16).

Zhang, Xingcheng, Zhizhong Li, Chen Change Loy, and Dahua Lin (2017). "Polynet: A pursuit of structural diversity in very deep networks." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 718–726 (p. 31).

Zhao, Hao (2017). "Some Promising Ideas about Multi-instance Video Segmentation." In: *The 2017 DAVIS Challenge on Video Object Segmentation - CVPR Workshops* (p. 82).

Zheng, Shuai, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr (2015). "Conditional random fields as recurrent Neural Networks." In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 1529–1537 (p. 34).

Zhu, Alex Zihao, Dinesh Thakur, Tolga Özaslan, Bernd Pfrommer, Vijay Kumar, and Kostas Daniilidis (2018a). "The multivehicle stereo event camera dataset: An event camera dataset for 3D perception." In: *IEEE Robotics and Automation Letters* 3.3, pp. 2032–2039 (pp. 52, 55, 67, 72, 75).

Zhu, Alex Zihao, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis (2019a). "Unsupervised event-based learning of optical flow, depth, and egomotion." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 989–997 (pp. 57, 60, 70, 98).

Zhu, Alex, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis (2018b). "EV-FlowNet: Self-Supervised Optical Flow Estimation for Event-based Cameras." In: *Proceedings of Robotics: Science and Systems*. Pittsburgh, Pennsylvania (pp. 54, 57, 69, 70, 75, 99).

Zhu, Alex, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis (2019b). *EV-FlowNet: Self-Supervised Optical Flow Estimation for Event-based Cameras*. "https://github.com/daniilidis-group/EV-FlowNet" (p. 69).

Zilly, Julian Georg, Rupesh Kumar Srivastava, Jan Koutnık, and Jürgen Schmidhuber (2017). "Recurrent highway networks." In: *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR, pp. 4189–4198 (pp. 30, 35).

Zintgraf, Luisa, Kyriacos Shiarli, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson (2019). "Fast Context Adaptation via Meta-Learning." In: *Proceedings of the 36th International Conference on Machine Learning (ICML)*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. Long Beach, California, USA: PMLR, pp. 7693–7702 (p. 84).