HAL
open science

Deep Learning and Information Geometry for Time-Series Classification

Daniel Brooks

To cite this version:

Daniel Brooks. Deep Learning and Information Geometry for Time-Series Classification. Machine Learning [cs.LG]. Sorbonne Université, 2020. English. NNT : 2020SORUS276 . tel-03984879

HAL Id: tel-03984879

https://theses.hal.science/tel-03984879v1

Submitted on 13 Feb 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE DE DOCTORAT DE SORBONNE UNIVERSITÉ
Spécialité **Informatique**
École Doctorale Informatique, Télécommunications et Électronique (Paris)

Deep Learning and Information Geometry for Time-Series Classification

Apprentissage Profond et Géométrie de L'Information pour la Classification de Signaux Temporels

Présentée par
Daniel BROOKS

Dirigée par
Matthieu CORD et Olivier SCHWANDER

Pour obtenir le grade de
DOCTEUR de SORBONNE UNIVERSITÉ

Présentée et soutenue publiquement le 6 avril 2020

Devant le jury composé de :

Mme. Florence TUPIN — Rapportrice
Professeure, Télécom Paris – LTCI – Institut Polytechnique de Paris

M. Marco CONGEDO — Rapporteur
Chargé de Recherche (HDR) – GIPSA-lab, CNRS, Université Grenoble Alpes, Grenoble-INP

M. Yannick BERTHOUMIEU — Examinateur
Professeur, Bordeaux INP

M. Frédéric BARBARESCO — Examinateur
KTD PCC "Sensing" Segment Leader, Senior Radar expert – THALES

M. Hichem SAHBI — Examinateur
Chargé de recherche CNRS (HDR), Sorbonne Université – LIP6

M. Olivier SCHWANDER — Co-directeur de thèse
Maître de Conférence, Sorbonne Université – LIP6

M. Matthieu CORD — Directeur de thèse
Professeur, Sorbonne Université – LIP6 & Senior Scientist, Valeo.ai

Daniel Brooks: *Deep Learning and Information Geometry for Time Series Classification,*
© 2020

ABSTRACT

Machine Learning, and in particular Deep Learning, is a powerful tool to model and study the intrinsic statistical foundations of data, allowing the extraction of meaningful, human-interpretable information from otherwise unpalatable arrays of floating points. While it provides a generic solution to many problems, some particular data types exhibit strong underlying physical structure: images have spatial locality, audio has temporal sequentiality, radar has time-frequency structure. Both intuitively and formally, there can be much to gain in leveraging this structure by adapting the subsequent learning models. As convolutional architectures for images, signal properties can be encoded and harnessed within the network. Conceptually, this would allow for a more intrinsic handling of the data, potentially leading to more efficient learning models. Thus, we will aim to use known structures in the signals as model priors. Specifically, we build dedicated deep temporal architectures for time series classification, and explore the use of complex values in neural networks to further refine the analysis of structured data.

Going even further, one may wish to directly study the signal's underlying statistical process. As such, Gaussian families constitute a popular candidate. Formally, the covariance of the data fully characterizes such a distribution; developing Machine Learning algorithms on covariance matrices will thus be a central theme throughout this thesis. Statistical distributions inherently diverge from the Euclidean framework; as such, it is necessary to study them on the appropriate, curved Riemannian manifold, as opposed to a flat, Euclidean space. Specifically, we contribute to existing deep architectures by adding normalizations in the form of data-aware mappings, and a Riemannian Batch Normalization algorithm. We showcase empirical validation through a variety of different tasks, including emotion and action recognition from video and Motion Capture data, with a sharpened focus on micro-Doppler radar data for Non-Cooperative Target Recognition drone recognition. Finally, we develop a library for the Deep Learning framework PyTorch, to spur reproducibility and ease of use.

RÉSUMÉ

L'apprentissage automatique, et en particulier l'apprentissage profond, unit un arsenal d'outillages puissants pour modeler et étudier les distributions statistiques sous-jacentes aux données, permettant ainsi l'extraction d'informations sémantiquement valides et interprétables depuis des séquences tabulaires de nombres par ailleurs indigestes à l'oeil humain. Bien que l'apprentissage fournisse une solution générique à la plupart des problèmes, certains types de données présentent une riche structure issue de phénomènes physiques: les images ont la localité spatiale, les sons la séquentialité temporelle, le radar la structure temps-fréquence. Il est à la fois intuitif et démontrable qu'il serait bénéfique d'exploiter avec astucieuse ces formations fondatrices au sein même des modèles d'apprentissage. A l'instar des architectures convolutives pour les images, les propriétés du signal peuvent être encodées et utilisées dans un réseau de neurones adapté, avec pour but l'apprentissage de modèles plus efficaces, plus performants. Spécifiquement, nous oeuvrerons à intégrer dans la conception nos modèles profonds pour la classification de séries temporelles des sur leurs structures sous-jacentes, à savoir le temps, la fréquence, et leur nature proprement complexe.

En allant plus loin dans une veine similaire, l'on peut s'atteler à la tâche d'étudier non pas le signal en tant que tel, mais bel et bien la distribution statistique dont il est issu. Dans ce scenario, les familles Gaussiennes constituent un candidat de choix. Formellement, la covariance des vecteurs de données caractérisent entièrement une telle distribution, pour peu qu'on la considère, à peu de frais, centrée; le développement d'algorithmes d'apprentissage, notamment profonds, sur des matrices de covariance, sera ainsi un thème central de cette thèse. L'espace des distributions diverge de manière fondamentale des espaces Euclidiens plats; il s'agit en fait de variétés Riemanniennes courbes, desquelles il conviendra de respecter la géométrie mathématique intrinsèque. Spécifiquement, nous contribuons à des architectures existantes par la création de nouvelles couches inspirées de la géométrie de l'information, notamment une couche de projection sensible aux données, et une couche inspirée de l'algorithme classique de la Batch Normalization. La validation empirique de nos nouveaux modèles se fera dans trois domaines différents: la reconnaissance d'émotions par vidéo, d'action par squelettes, avec une attention toute particulière à la classification de drones par signal radar micro-Doppler. Enfin, nous proposerons une librairie PyTorch aidant à la reproduction des résultats et la facilité de ré-implémentation des algorithmes proposés.

REMERCIEMENTS

Cette thèse, non contente de ne s'être écrite toute seule ou un jour, est le fruit de riches et nombreuses interactions, tant professionnelles que simplement humaines, du court au long terme et par tous les moyens, avec une floppée de personnes qu'il est bon de citer.

Je tiens tout d'abord à remercier mes directeurs: Olivier et Matthieu à Jussieu, pour les longues discussions scientifiques et stylistiques; Frédéric et Jean-Yves à Limours, pour leur expertises respectives; à tous, pour leurs encouragements et bienveillance vis-à-vis d'un thésard parfois tête-en-l'air.

Je remercie également mes collègues en tous genres, colorant le quotidien d'un arc-en-ciel collectif.

Je n'oublie non plus Christelle et Gilles à l'ONERA, sans qui cette thèse n'a pas commencé.

J'adresse ma reconnaissance au groupe de travail OTAN SET245, qui a bien voulu transmettre une base de données radar riche en contenu, largement exploitée lors de ces travaux.

Enfin, je remercie mes proches, sans qui rien de tout cela n'aurait de sens. Mes parents, dont le soutien sans faille m'a toujours porté de l'avant, mes amis, collègues et Yann, certains voisins qui se reconnaîtront, Miel, la cuisine basque, les danseurs et danseuses. Tout particulièrement, je remercie de plein coeur Céline pour m'avoir supporté et accompagné avec patience et tendresse.

CONTENTS

ABSTRACT i
RÉSUMÉ iii
REMERCIEMENTS v
CONTENTS vii
LIST OF FIGURES ix
LIST OF TABLES xiii
ACRONYMS xv
1 INTRODUCTION 1
1.1 Context 1
1.2 Motivations 3
1.3 Contributions and outline 10
1.4 Related publications 13
2 THEORETICAL BACKGROUND 15
2.1 Introduction 17
2.2 Radar Signal and Simulation 18
2.3 Euclidean Machine Learning 23
2.4 Information Geometry 38
2.5 Riemannian Machine Learning 46
2.6 Conclusion 52
3 SECOND-ORDER PIPELINE FOR TEMPORAL CLASSIFICATION 55
3.1 Introduction 57
3.2 Learning on structured time series representations 59
3.3 Full pipeline for temporal classification 71
3.4 Experimental validation 75
3.5 Conclusion 91
4 ADVANCES IN SPD NEURAL NETWORKS 95
4.1 Introduction 97
4.2 Data-Aware Mapping Network 100
4.3 Batch-Normalized SPDNet 104
4.4 Riemannian manifold-constrained optimization 108
4.5 Convolution for covariance time series 116
4.6 Experimental validation 119
4.7 Conclusion 130
5 CONCLUSION AND PERSPECTIVES 133
BIBLIOGRAPHY 137

A ALORITHMIC DETAILS AND PROPERTIES OF THE PROPOSED LEARNING MODELS 157
A.1 Model sizes and speeds 157
A.2 Sample code 159
A.3 Use cases 159
A.4 Organization of the provided PyTorch library 161
A.5 Implementation details 162
B DETAILED DESCRIPTION OF THE NATO RADAR DATABASE 165
B.1 Global overview 165
B.2 Description of the data 166
B.3 Contents of the disk 166
B.4 Possible setting for learning 167

LIST OF FIGURES

Chapter 1: INTRODUCTION 1

Figure 1.1 Example of a drone's micro-Doppler signal. 3
Figure 1.2 The race to ever bigger Deep Neural Networks. 4
Figure 1.3 Spectrogram versus Capon sepctrogram 6
Figure 1.4 Raw signal and its Fourier and covariance representations 7
Figure 1.5 Example of image deformation through scale, rotation, translation and illumination. 9

Chapter 2: THEORETICAL BACKGROUND 17

Figure 2.1 The basic form of a standard signal emitted by a radar . . 18
Figure 2.2 Illutration of the Radar Cross-Section (RCS) 19
Figure 2.3 Depiction of the three drones used in simulations 20
Figure 2.4 Illustration of the simulated signal 21
Figure 2.5 Illustration of the Fourier spectrum of a radar signal 22
Figure 2.6 Mean-Squared Error (MSE) versus Logistic Loss (LL). 24
Figure 2.7 Activation functions 26
Figure 2.8 Illustration of the kernel trick 28
Figure 2.9 Representation power of Deep Neural Networks (DNNs) through hierarchical learning. 30
Figure 2.10 Illustration of the convolution as a linear transformation . 32
Figure 2.11 LeNet, the historical first Convolutional Neural Network (CNN) 32
Figure 2.13 Illustration of a convolutional block 32
Figure 2.12 AlexNet, the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winner 33
Figure 2.14 VGG neural network, second place in the 2014 ILSVRC challenge 34
Figure 2.15 The convolutionalization trick 35
Figure 2.16 Recurrent Neural Network (RNN) with one hidden layer . 36
Figure 2.17 Illustration of the fundamental representation equivalence between a 2D time-frequency image and a series of 1D spectrums. 37
Figure 2.18 Illustration of a Long Short-Term Memory network 38
Figure 2.19 The sphere: a simple manifold 39
Figure 2.20 Kullback-Leibler divergence between two univariate Gaussians 42

Figure 2.21 Manifold mappings 45
Figure 2.22 Illustration of one iteration of the Karcher flow (Karcher 1977) 49

CHAPTER 3: SECOND-ORDER PIPELINE FOR TEMPORAL CLASSIFICATION 57
Figure 3.1 Illustration of the Bilinear Mapping (BiMap) layer 60
Figure 3.2 Illustration of a generic Symmetric Positive Definite (SPD) neural network 61
Figure 3.3 Downsampling of a spectro-temporal signal 63
Figure 3.4 The proposed architecture for radar signal classification . . 64
Figure 3.5 Illustration of how strided filters of size k and stride s work on an input of size n 65
Figure 3.6 The proposed complex-valued architecture for radar signal classification 68
Figure 3.7 Illustration of the proposed Fourier-like convolutional layer 72
Figure 3.8 Illustration of the global pipeline proposed for the classification of time-frequency signals 73
Figure 3.9 Illustration of the Second-Order Fully Temporal Network (SOFTNet) 74
Figure 3.10 Structured time series representations 76
Figure 3.11 Evolution of the scattering points and normals for the Vario, Phantom2 and S1000+ drones 77
Figure 3.12 Spectrograms of noisy and uncluttered signals 81
Figure 3.13 Classification accuracies for three learning models 83
Figure 3.14 Performance of the learning models for increasingly good conditions 84
Figure 3.15 Growth of classification confidence over time in a very challenging environment ($SNR = 10dB$ and $PRF = 2kHZ$) . . 85
Figure 3.16 Growth of classification confidence of a Fully Temporal Convolutional Network (FTCN) in audio recognition . . . 86

CHAPTER 4: ADVANCES IN SPD NEURAL NETWORKS 97
Figure 4.1 Log Eigenvalues (LogEig) as a special case of logarithmic mapping 101
Figure 4.2 Illustration of the Data-Aware Mapping Network (DAMNet) architecture 103
Figure 4.3 Illustration of manifold-constrained gradient update 110
Figure 4.4 The convolutional BiMap 118
Figure 4.5 Validation accuracy and loss curves of SPD neural network (SPDNet) and Batch-Normalized SPDNet (SPDNetBN) on the North Atlantic Treaty Organization (NATO) dataset . . 122

Figure 4.6 Confusion matrix on the NATO dataset 123
Figure 4.7 Performance of all models in function of the amount of synthetic radar data 124
Figure 4.8 Separability of two classes of the radar dataset. Separation between classes is higher with a reference matrix near the barycenter. 126
Figure 4.9 Validation accuracy for the closest barycenter algorithm. Iteration 0 corresponds to the arithmetic mean, iteration 1 to the LEM barycenter. 127
Figure 4.10 Distribution of class instance number and sequence length for the Hochschule der Medien 05 (HDM05) dataset 129

Appendix A: alorithmic details and properties of the proposed learning models 157

Appendix B: detailed description of the NATO radar database 165

LIST OF TABLES

Chapter 2: theoretical background 17

Chapter 3: second-order pipeline for temporal classification 57

Table 3.1 The parameters are found or estimated from drone specifications 79
Table 3.2 Simulation parameters and their default values 80
Table 3.3 Performance comparison of real and complex deep structures on radar data on various amount of noisy data 89
Table 3.4 Performance comparison of real and complex deep structures on radar data on various amount of less noisy data . 89
Table 3.5 Performance comparison of first- and second-order models on radar data 90

Chapter 4: advances in SPD neural networks 97

Table 4.1 Performance of SPDNet, FTCN and Minimum Riemannian Distance to Riemannian Mean (MRDRM) on the NATO dataset 121
Table 4.2 Performance of SPDNet, DAMNet and SPDNetBN on the NATO dataset 121
Table 4.3 Performance of SPDNet, DAMNet and SPDNetBN on the Acted Faces Expressions in the Wild (AFEW) dataset 128
Table 4.4 Accuracy comparison of SPDNets and DAMNets on the HDM05 dataset. 129

Appendix A: alorithmic details and properties of the proposed learning models 157

Appendix B: detailed description of the NATO radar database 165

Table B.1 Duration of recorded data in the database 168

ACRONYMS

AI	Artificial Intelligence
BN	Batch Normalization
CV	Computer Vision
DL	Deep Learning
DNN	Deep Neural Network
GAN	Generative Adversarial Network
GPU	Graphics Processing Unit
MSE	Mean-Squared Error
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
SHADE	SHAnnon DEcay
SIFT	Scale-Invariant Feature Transform
VQA	Visual Question Answering
SAR	Synthetic Aperture Radar
DSP	Digital Signal Processing
BoW	Bag of Words
MLE	Maximum Likelihood
RCS	Radar Cross-Section
PRI	Pulse Repetition Intervall
PRF	Pulse Repetition Frequency
RPM	Rounds per Minute
UAV	Unmanned Aircraft Vehicle
SNR	Signal-to-Noise Ratio
NCTR	Non-Cooperative Target Recognition
ML	Machine Learning
SVM	Support Vector Machine
MLP	Multi Layer Perceptron
NN	Neural Network
CNN	Convolutional Neural Network
ILSVRC	ImageNet Large Scale Visual Recognition Challenge

MIL	Multiple Instance Learning
ASR	Automatic Speech Recognition
EAR	Environmental Audio Recognition
BCI	Brain-Computer Interface
EEG	electroencephalography
MRI	Magnetic Resonance Imagery
FCN	Fully Convolutional Network
FTCN	Fully Temporal Convolutional Network
RCNN	Region Convolutional Neural Network
GAP	Global Average Pooling
GMP	Global Max Pooling
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
LL	Logistic Loss
SL	Supervised Learning
UL	Unsupervised Learning
LN	Layer Normalization
MM	Matrix Multiplication
DARPA	Defense Advanced Research Projects Agency
FFT	Fast Fourier Transform
SCM	Sample Covariance Matrix
OT	Optimal Transport
EF	Exponential Families
μ-D	micro-Doppler
MoCap	Motion Capture
US	Unsupervised Learning
NLP	Natural Language Processing
BPTT	Backpropagation Through Time
k-NN	k-nearest neighbours
MDM	Minimum Distance to Mean
MNIST	Modified National Institute of Standards and Technology
EF	Exponential Family
KL	Kullback-Leibler

FIM	Fisher Information Matrix
SPD	Symmetric Positive Definite
HPD	Hermitian Positive Definite
HPDNet	HPD neural network
SPDNet	SPD neural network
IG	Information Geometry
LEM	Log-Euclidean Metric
DAMNet	Data-Aware Mapping Network
BarNorm	Barycentric Normalization
ParNorm	Parametric Normalization
SOFTNet	Second-Order Fully Temporal Network
BiMap	Bilinear Mapping
ReEig	Rectified Eigenvalues
LogEig	Log Eigenvalues
AIRM	Affine Invariant Riemannian Metric
TSLR	Tangent Space Linear Regression
CovPool	Covariance Pooling
MRDRM	Minimum Riemannian Distance to Riemannian Mean
SVD	Singular Value Decomposition
CGS	Classical Gram-Schmidt
RMT	Random Matrix Theory
DFT	Discrete Fourier Transform
dB	decibel
FourierNet	Fourier Neural Network
CovPool	covariance pooling
GPU	Graphical Processing Unit
$\mathbb{CR}$Net	$\mathbb{CR}$ Neural Network
SpectroSPD	Spectral SPD Neural Network
PT	Parallel Transport
BatchNorm	Batch Normalization
SPDNetBN	Batch-Normalized SPDNet
NATO	North Atlantic Treaty Organization
AFEW	Acted Faces Expressions in the Wild
HDM05	Hochschule der Medien 05

CHAPTER 1

INTRODUCTION

1.1 Context

Artificial Intelligence, and in particular Machine Learning, has witnessed a soaring worldwide interest, leading to possibilities which yet eluded the realms of the possible: from better-than human face recognition (Hern 2015) to self-driving cars ("As Self-Driving Cars Stall, Players Revive an Old Approach" 2019), from disease prediction (M. Chen et al. 2017) to virtual doctors (*Your virtual doctor will see you now* 2019), from helicopter pilot assistance (Abbeel et al. 2007) to mastering the game of Go (*AlphaGo Zero* 2019), progress has been phenomenal and its usage sprawling to an ever more diverse field of applications. Historically, many of the leading innovations in the field were and are primarily designed for Computer Vision, *i.e.* the field of image analysis. Since the first applications by US Naval Research (Rosenblatt 1958), to document zip code recognition (LeCun et al. 1989; Lecun et al. 1998), then to modern Computer Vision, whether as image classification (Durand et al. 2017), visual reasoning (Ben-younes et al. 2017), style transfer (Johnson 2019) or image generation (*NVIDIA/pix2pixHD* 2019), Machine Learning is a historic foundation of modern computing.

More specifically, most of the frenzy in this paradigm shift revolves around Deep Learning, which is a subclass of Machine Learning mostly involving Deep Neural Networks. In short, Deep Neural Networks build a latent representation of data through the learning of a hierarchy of layers; as such, Deep Learning can be seen as a generalization of certain classes of standard Machine Learning algorithms, by jointly learning feature representation space and separation within a single architecture. A major subgroup of Machine Learning is the field of Supervised Learning, where a dataset of pairs of inputs and outputs is given to allow the learning of a decision function through the optimizing of a loss function. While Deep Learning has successfully been used in the context of supervised classification, there exists a plethora of different possible tasks. These may include Visual Question Answering, where the inputs are couples of images and

text, image captioning, where the output given an image is text, object or instance segmentation, where the output is bounding boxes or a list of pixels given an input image. While each one involves task-specific engineered model and loss functions, the general principle remains the same: to learn a decision function on a dataset of inputs and outputs, based on the outputs of a loss function.

Despite the popularity of Computer Vision, the instrinsic genericity of Deep Learning makes it a suitable match for scores of other applicative fields such as health, neuroscience, automated systems and robotics, social network studies, Brain-Computer Interfaces, audio, radar, lidar... Of these, some closely follow pure Computer Vision, such as hyperspectral (Tao et al. 2015; Liang and Q. Li 2016) or Synthetic Aperture Radar imaging (Tupin et al. 1998; Tupin et al. 2018; Atto et al. 2013). However, some types of data exhibit a rich structure within their inherent construction, as for example the case of micro-Doppler radar data. Here, the stream of physical processes involved in the signal formation shape it in a visually recognizable fashion (Figure 1.1). Leveraging the resulting structures within the associated learning models thus seems a promising idea.

The usage of micro-Doppler radar data is gaining momentum in diverse branches of the industry, such as in autonomous vehicles (Inman, M. 2020) or fine-grained gesture recognition in smartphones (Lien et al. 2016; S. Wang et al. 2016). Radar roots its historical foundations in the theoretical works of Maxwell (1865), H. Hertz (1889) and N. Tesla (Page 1962), then grounds its preliminary applications with the notable patents of G. Marconi and C. Hülsmeyer (Fahie 1899; Blanchard 2019). Although radar has been key in its proven ability to detect potentially hostile airborne crafts, the recent emergence of less conventional, smaller and erratic-behaving Unmanned Aircraft Vehicles confronts the classification task of radars to an unprecedented challenge. Unrecognized drone sightings over airports ("Gatwick 'drone sighting' diverts flights" 2019) or nuclear plants (Bouchaud 2014) bring a new kind of threat to civilian security, while military operations both harness ("Top Iranian general killed by US in Iraq" 2020) and heed the imminent danger of stealthy strikes (Press 2018; "Syrian army foils drone attack on military base in northwest" 2019), to the point of redefining strategical paradigms and international law altogether (Boyle 2013; Shah 2014). In this modern perspective, new approaches with a focus on versatility, efficiency and scalability should provide key elements to tackle these issues.

This thesis focuses on Deep Learning in the Supervised Learning setting, also termed deep supervised learning. It brings particular attention to developing models adapted to the underlying structure of the data. Furthermore, experimental approaches thus far, while spanning a variety of different applicative fields, mainly focus on micro-Doppler radar data. We motivate the work thereupon below.

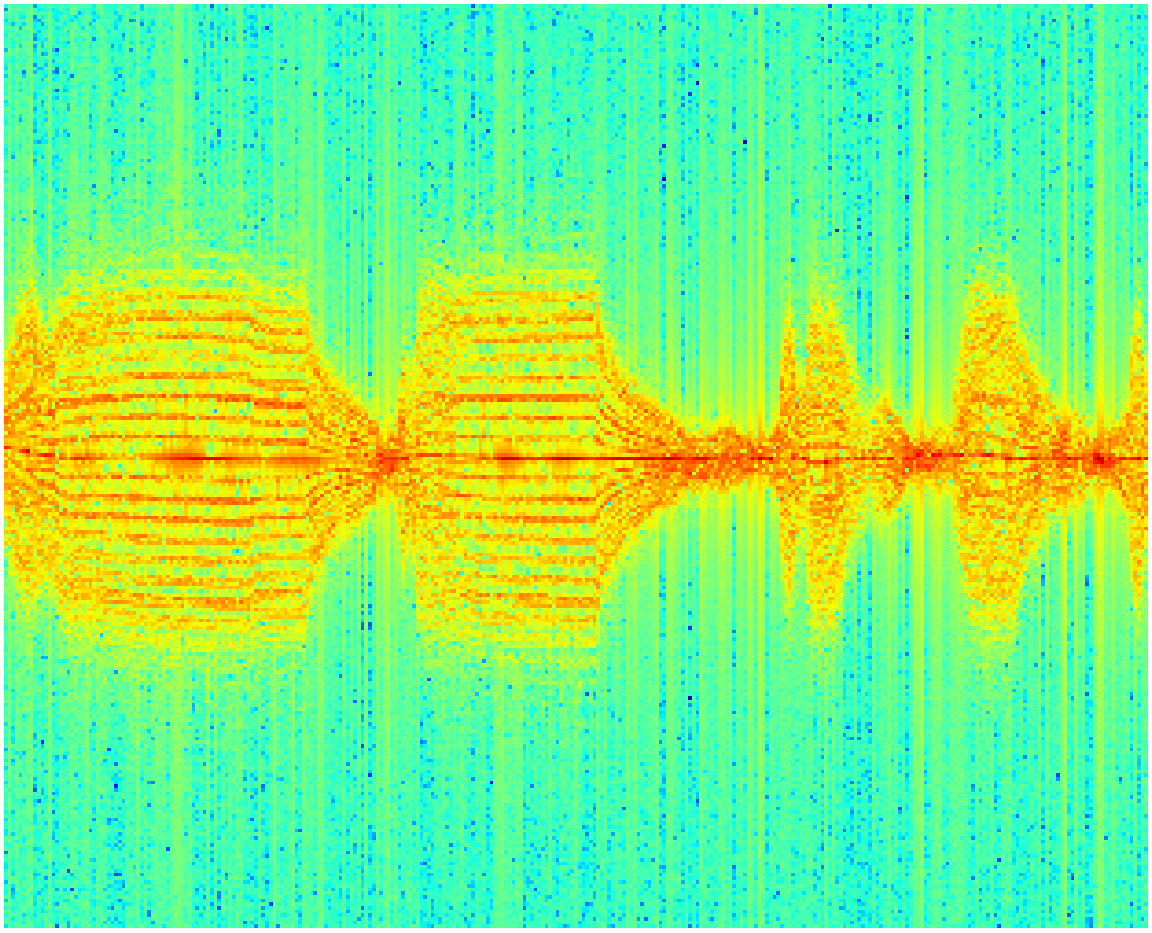

Figure 1.1.: **Example of a drone's micro-Doppler signal.** This time-frequency representation exhibits rich structure; much information about the target is inferable from Digital Signal Processing tools.

1.2 Motivations

The core research in industry is shifting towards Artificial Intelligence, for better or worse. Indeed, alongside insightful and game-changing applications of Deep Learning, some vulnerabilities, or even failures leading to real-world fiascos, have tainted the otherwise gleaming aura of the field. One can think of, for instance, Microsoft's chatbot Tay which became a genocidal racist after some hours on Twitter (Vincent 2016), Amazon's recruiting tool's strong bias to recommend Caucasian males ("Amazon scraps secret AI recruiting tool that showed bias against women" 2018), or Florida's crime recidivism software unreasonably over-biased against minority communities (Julia Angwin 2016). Even in simple Computer Vi-

sion tasks, models with outstanding performance are not necessarily as robust as we may believe: as shown in (Goodfellow et al. 2015)'s seminal work, slightly perturbing the image with no visual impact on humans can quickly cause a seasoned model to completely fail at recognizing the initial object. At the root of this major issue, one can argue that data representation plays a key role: indeed, while humans naturally infer concepts, causality and relationships from given data, all the models "see" are matrices of floating numbers; the mathematical space spawned by these data points is thus a true corner stone in building robust and trustable models (*Tackling bias in artificial intelligence (and in humans) | McKinsey* 2019). While the fact that Deep Learning's unchallenged performance is matched but by our lack of understanding of that performance has been a long-standing meme in the community (see Figure 1.2), these seemingly instrinsic problems further accent the need for a better understanding of how models interact with data.

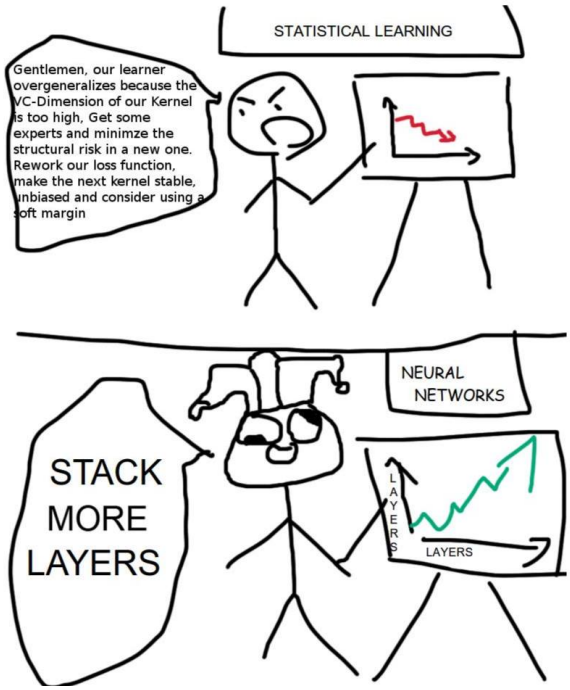

Figure 1.2.: **The race to ever bigger Deep Neural Networks.** Image source: *Deep Learning in Practice* (2019).

The shift from Deep Neural Networks to Convolutional Neural Networks in Computer Vision is a paramount example of architectures adapting to their input data: while standard Deep Neural Networks feature fully-connected, or dense layers (*i.e.* full Matrix Multiplication), Convolutional Neural Networks replace the latter with convolution banks, which can be seen as sparse, locally connected Matrix Multiplications. Contrary to a full Matrix Multiplication, a convolution harnesses the spatially local structure of images, producing much better results

with many less parameters, with an additional level of robustness to perturbation in the data: indeed, convolutions are formally equivariant to translation.

Aiming for more instrinsic representations of data and designing adapted models with a notion of robustness will be a common leitmotiv throughout this thesis. Simply put, this issue can be explored both through data pre-processing and model design. Again, micro-Doppler data will constitute a illustrative example of mult-faceted structured data. We give a brief overview of these two complementary approaches.

1.2.1 Pre-processing data for better representativity

The idea of processing raw data to a more intrinsic representation is by no means recent, as indicated in the early literature of Computer Vision. Indeed, its first and foremost champion probably is the Scale-Invariant Feature Transform (Lowe 2004), an image representation algorithm integrating into a single vector information about color, gradients, angles, such that it be robust to scale, rotation, translation and illumination. Even earlier, the idea of deformable kernels, or steerable filters was developed in the same objective of robustness of the representation to known transformations (Perona 1995). These ideas secured a rich progeny all the way to modern Computer Vision, as we will see below.

Again, this idea is by no means limited to Computer Vision. In the analysis of structured time series, such as audio, or radar data, the Fourier transform (Shannon C. E. 2013) - from here on out, we will rather refer to the fast implementation ubiquitously used, the Fast Fourier Transform - is the most used representation, rather than the raw time data. In the Digital Signal Processing literature, there exists an abundance of alternative representations, most of them with a strong theoretical link to the Fourier transform. The Capon spectrum introduced by Capon (1969) mimicks the behaviour of the Fourier transform while adding constraints to improve on the time-frequency resolution, inherently limited by Heisenberg's incertainty principle (Heisenberg and Broglie 1958). In the same principle, the Wigner-Ville spectrum, instance of the so-called Cohen class of time-frequency operators (L. Cohen 1989), allows for a more fine-grained resolution as illustrated in Figure 1.3. Other approaches build hierarchical frequencies at different time scales, circumventing the resolution problem altogether; the founding innovation in that regards is the notion of wavelets (Mallat 2008), which are now ubiquitous in signal compression.

A tightly-linked notion (Flandrin 1998), that of covariance, is also a powerful representation of the data, albeit less used. Figure 1.4 captures different possible

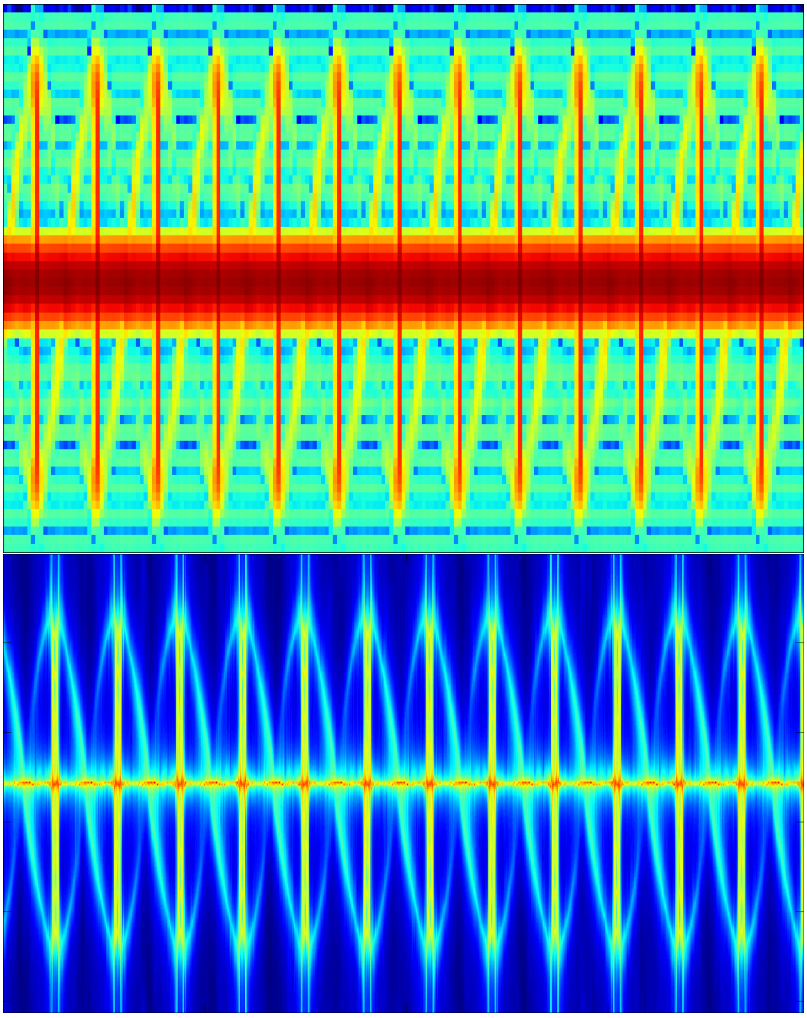

Figure 1.3.: **Spectrogram versus Capon sepctrogram**. The Capon spectrum is a variant of the Wigner-Ville spectrum, making use of parametric constraints on the spectrum to enhance its time-frequency resolution.

representations of the data, all of which are rooted in the Digital Signal Processing literature and share mathematical properties.

We now dwell a bit longer on the covariance representation of structured time series data, as it will be a central theme throughout this thesis. Conceptually, the Sample Covariance Matrix captures temporal fluctuations in a compact and meaningful way. In other words, the Sample Covariance Matrix is a powerful discriminative feature representation for structured time series, and deserves the exploration of adapted Machine Learning models. This holds even more for signals with strong physical structure; again, one can think of radar data, Brain-Computer Interface, Magnetic Resonance Imagery or audio. Figure 1.4 gives the intuition of building different learning models upon different representations of the data.

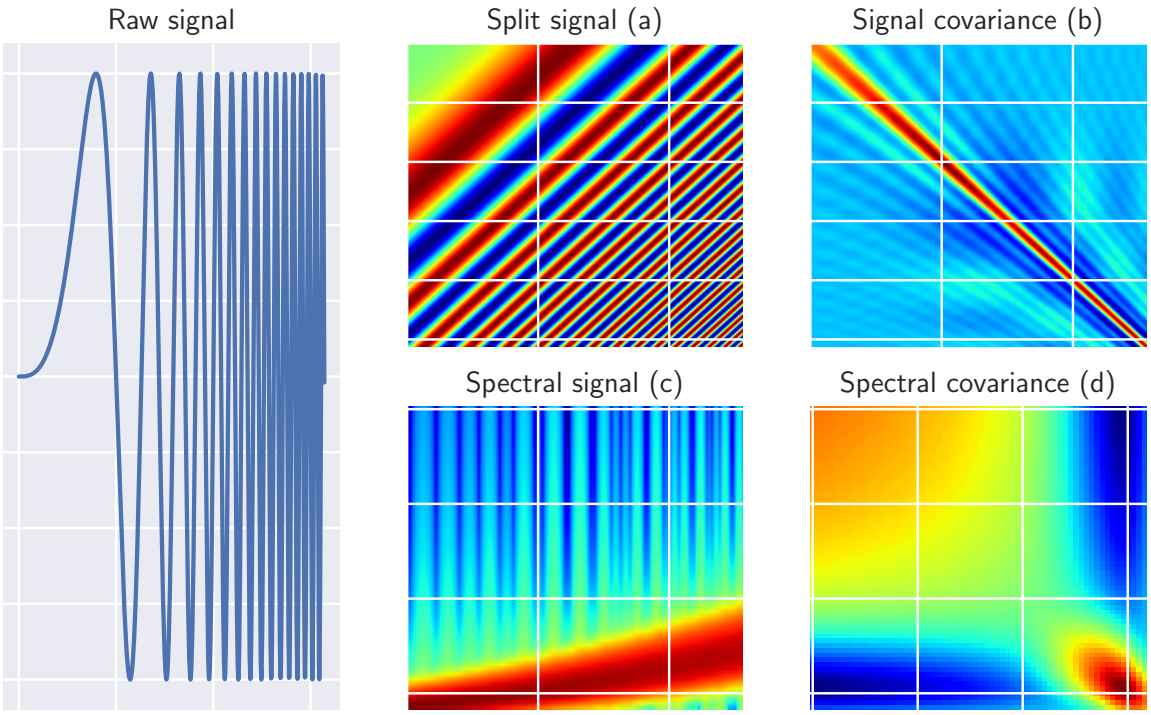

Figure 1.4.: **Raw signal and its Fourier and covariance representations** The raw signal is split through a sliding window, from which either covariance or Fourier transform can be computed.

1.2.2 Designing architectures for powerful representation learning

It is broadly accepted than one main factor for what is now known as the "Artificial Intelligence winters" is the lack of data and computing power to successfully train competitive models. The year 2012 is now seen as a defining landmark to the renewal and explosion of interest in Artificial Intelligence with the seminal work implementing a Convolutional Neural Network dubbed $AlexNet$ (Krizhevsky et al. 2012) for image classification on the ImageNet dataset within the ImageNet Large Scale Visual Recognition Challenge, outperforming for the first time, and by a large margin, traditional methods. The following years led to a trend, simplistically illustrated in Figure 1.2, which constituted the mainstream approach for several years, with the emergence of deeper and more complex architectures (notably the VGG model Simonyan and Zisserman (2015)), to the point in 2015 when models beat human performance on the ImageNet Large Scale Visual Recognition Challenge challenge (Tsang 2019). However, the models now boasted orders of magnitude more parameters than the number of examples: $60M$ for $AlexNet$, $133M$ for VGG, when the large-scale ImageNet dataset contained "only" $1.2M$ images. Alongside the practical challenges of training such large models, this

disproportion arose the concern that perhaps models were mostly overfitting on the datasets.

This concern led to a further paradigm shift: from Deep Learning pioneers blaming models to simply learn dataset biases (*Colloquium d'Informatique de Sorbonne Université* 2018) and encouraging the push towards reasoning, rather than purely learning models (*Yoshua Bengio | From System 1 Deep Learning to System 2 Deep Learning | NeurIPS* 2019), to the setup of the "xAI" (for "explainable Artificial Intelligence") Defense Advanced Research Projects Agency (Gunning 2017), focus shifted towards smaller, more efficient models. Following this trend, another 2015 famed architecture, the $ResNet101$ (He et al. 2016), exhibited a mere $1.7M$ parameters, while outperforming all previously cited models. Many followed, such as the $AllConvolutionalNet$ (Springenberg et al. 2015), or the $HybridNet$ architecture (Robert et al. 2018), which lead us to the following paragraph.

1.2.3 Enforcing constraints for intrinsic data modelling

The quest to models intrinsically adapted to their input data has thus become an ongoing pursuit in the Deep Learning community. Indeed, since the a Deep Neural Network learns its own feature representation through the hierarchy of layers, clever engineering of these layers hopefully leads to more intrinsic representations of the data.

This engineering often draws inspiration from the older data processing techniques mentioned above, transferring them in Deep Neural Networks in a differentiable fashion. This is for instance embodied in part-based image segmentation (Mordan et al. 2017) or steerable Convolutional Neural Networks (T. S. Cohen and Welling 2016). This idea of models being invariant or robust to a given class of transformations, is key to learning better representations, as illustrated in Figure 1.5: a learning model should be expected to recognize that both images feature the exact same content, although deformed through given transformations.

In a similar vein, other works design layers to stabilize models through normalization, which can be seen as explicitly enforcing invariance to scale and translation within the network layers. The seminal effort in this direction is found in the introduction of Batch Normalization in Deep Neural Networks (Ioffe and Szegedy 2015). Further works have improved upon the idea, such as in Layer Normalization (J. Xu et al. 2019) . Others have studied the generalizability of such normalizations across different domains (X. Wang et al. 2019) . The recurring motivation behind inner normalizations is to reduce the network's dependence the representation space covariate shift at each layer.

Figure 1.5.: **Example of image deformation through scale, rotation, translation and illumination.** Ideally, a learning model should be insensitive to the deformation to infer in both cases the same image class: cat.

Taking further inspiration in the historic roots of Machine Learning, recent works have investigated incorporating information (as in Shannon information) to regularize the Deep Neural Network in a meaningful data-driven fashion. For instance, a process named SHAnnon DEcay (Blot et al. 2018) improves upon Neural Network weight decay (Krogh and J. A. Hertz 1992) by leveraging Shannon information. In a similar vein, integrating entropy in generative Neural Networks seems to improve on generalization capacity (Vu et al. 2019). Furthermore, by pushing the frontier of image generation in 2017, Wassertein Generative Adversarial Networks (Arjovsky et al. 2017; Gulrajani et al. 2017) kindled a surge in integrating geometric constraints in Neural Networks - in this case, Optimal Transport.

Digging even deeper within the elder literature, modern works have made use of Information Geometry. Whether it be in graph analysis (Bronstein et al. 2017), domain adapation (Yair et al. 2019), Neural Network optimization (Marceau-Caron and Ollivier 2016), Brain-Computer Interface (Barachant et al. 2012), Magnetic Resonance Imagery (Pennec et al. 2006), and even Computer Vision (Acharya et al. 2018; Mollahosseini et al. 2016; Tuzel et al. 2006). The field of Information Geometry, initiated with the works of Rao (1992), Fréchet (1943) and Amari (2016), "*is an interdisciplinary field that applies the techniques of differential geometry to study probability theory and statistics. It studies statistical manifolds, which are Riemannian manifolds whose points correspond to probability distributions.*" (*Information geometry* 2019). The core idea of using Information Geometry in Machine Learning is to model the individual data points as the underlying statistical distribution they are sampled from, instead than simply considering a Euclidean structure to the data. Fundamentally, these distributions form a differential manifold, which then constitutes the subject study of the subsequent Machine Learning algorithm. A key example lies within Exponential Familys, more specifically the family of Gaussian distributions. Indeed, many natural phenomena can be reasonably modelled as

the realization of a Gaussian law, characterized by the first- and second-order moments, *i.e.* the mean and covariance: whenever some form of structure is involved, whether temporal, spatial or more abstract relational principles, moments can naturally found within the sample data.

To this day, learning models based on Information Geometry do not yet enjoy the popularity of Deep Learning methods. However, some particular instances seem to be known by a relatively larger audience in the general field of Machine Learning. The Fisher kernel method for instance (Sánchez et al. 2013) has know success in general Machine Learning, including Bag of Words methods for image retrieval (Jegou et al. 2012). Another method named natural gradient, introduced by Amari (1998), aims at modifying the gradient of an optimization problem by fitting it to the manifold spanned by the problem's parameters. It is nowadays not uncommon to see it used to regularize the learning in Deep Neural Networks (the term "neuromanifold" is then used), and has since then enjoyed an upsurge in interest (Pascanu and Bengio 2014; Marceau-Caron and Ollivier 2016; Marceau-Caron and Ollivier 2017). While both methods rely on Information Geometry, they considerably differ in terms of purpose and realization. Given this information, it becomes important to distinguish two broad learning settings making use of Information Geometry. In the first one, its theoretical background is summoned to refine "traditional" methods; it is the case for the Fisher kernel and the natural gradient. In the second scenario, objects with a particular geometry, *i.e.* belonging to a Riemannian manifold, formally require the tools of Information Geometry for Machine Learning methods to be built upon them. Henceforth, because this thesis will attempt to study structured time series data with an underlying physical geometry, we will focus on the second class of settings.

Machine Learning models within the Information Geometry framework understandably lag behind the complexity and diversity found within the standard Euclidean framework. As argued above, filling this gap is a pressing matter, especially for fields where much is to gain when exploiting the strong underlying structures. Most of this thesis will thus deal with developing such models; specifically, the focus is set on the manifold of Gaussian distributions, with applications to a variety of fields. Next section details the particular proposed contributions.

1.3 Contributions and outline

For the reasons above, the works presented throughout this thesis bring particular care to developing Deep Learning models adapted to the underlying form of the

data, incorporating these structural priors within. We both explore standard deep architectures, and less explored Deep Neural Networks on covariance matrices, rooted in Information Geometry. Specifically, we take inspiration from Computer Vision's intuitions and translate them to a different field of application - mainly, micro-Doppler radar. Furthermore, we expand upon existing architectures on covariance matrices to seek general improvement in a variety of tasks - in addition to radar, emotion classification from video, action recognition from Motion Capture data.

- Chapter 2: THEORETICAL BACKGROUND
  The presented works require cross-overs from a variety of fields. Thus, we first go over the core aspects of radar data formation, Machine Learning, and Information Geometry, with a focus on the concepts which will then be further utilized.

- Chapter 3: SECOND-ORDER PIPELINE FOR TEMPORAL CLASSIFICATION
  This chapter introduces various learning models on structured time series data. We show how knowledge of its physical morphology can help design adapted deep models; specifically, we present:
  1. A Fully Temporal Convolutional Network (FTCN), a Deep Learning model operating on real-valued spectrograms of temporal data;
  2. An Information Geometry-based Deep Neural Network operating on covariance matrix representations of the time series;
  3. A complex-valued version of either model above, allowing to generalize the learning to complex-valued time series.

  Finally, we show how these models can be combined into a single pipeline for structured time series classification, called Second-Order Fully Temporal Network (SOFTNet). We report empirical properties of all introduced models through experimentations on micro-Doppler radar data.

- Chapter 4: ADVANCES IN SPD NEURAL NETWORKS
  Here, we set our focus entirely on deep models for covariance data, or SPD neural networks: at a high level, we improve upon the existing art by introducing geometry- and data-aware normalizations within the network.

  We first propose the Data-Aware Mapping Network (DAMNet) architecture, which generalizes the mathematical framework leveraged by the original SPD neural network. In both theory and practice, a DAMNet allows to refine both the inference and training by using the input data's underlying geometric information, by refining the Euclidean mapping stage at network's final layer.

Then, we develop the Riemannian equivalent, for SPD neural networks, to the well-known Batch Normalization (BatchNorm) algorithm in Deep Neural Networks, leading to an architecture we dub Batch-Normalized SPDNet (SPDNetBN). Using the intuitions from the DAMNet architecture which it further generalizes, and additonal concepts of Information Geometry, we again show improvement over the previous methods. Finally, we propose an incipient 1D convolutional layer for SPD matrices.

Throughout this chapter, various applicative fields are set forth; though radar data remains the main focus, we set up experiments on video and Motion Capture data, respectively for emotion and action recognition. Finally, we present an implementation of all proposed algorithms as library within the renouned PyTorch Deep Learning framework.

1.4 Related publications

This thesis is based on the material published in the following papers:

- D. Brooks, O. Schwander, F. Barbaresco, J. Schneider, and M. Cord. "Temporal Deep Learning for Drone Micro-Doppler Classification". In: *2018 19th International Radar Symposium (IRS)*. June 2018, pp. 1–10;

- D. Brooks, O. Schwander, F. Barbaresco, J. Schneider, and M. Cord. "Exploring Complex Time-series Representations for Riemannian Machine Learning of Radar Data". In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. May 2019, pp. 3672–3676;

- D. Brooks, O. Schwander, F. Barbaresco, J.-Y. Schneider, and M. Cord. "Complex-valued neural networks for fully-temporal micro-Doppler classification". In: *2019 20th International Radar Symposium (IRS)*. ISSN: 2155-5753, 2155-5745. June 2019, pp. 1–10;

- D. Brooks, O. Schwander, F. Barbaresco, J.-Y. Schneider, and M. Cord. "Second-Order Networks in PyTorch". en. In: *Geometric Science of Information*. Ed. by F. Nielsen and F. Barbaresco. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 751–758;

- D. Brooks, O. Schwander, F. Barbaresco, J.-Y. Schneider, and M. Cord. "A Hermitian Positive Definite neural network for micro-Doppler complex covariance processing". In: *International Radar Conference*. Toulon, France, Sept. 2019;

- D. Brooks, F. Barbaresco, Y. Ziani, J.-Y. Schneider, and C. Adnet. "IA & réseaux de neurones profonds pour la reconnaissance Radar de drones sur critères Micro-Doppler et Cinématique". fr. In: Rennes, FRANCE: Computer & Electronics Security Applications Rendez-vous (C&ESAR), Nov. 2019, p. 16;

- D. Brooks, O. Schwander, F. Barbaresco, J.-Y. Schneider, and M. Cord. "Riemannian batch normalization for SPD neural networks". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 15463–15474.

CHAPTER 2

THEORETICAL BACKGROUND

Chapter abstract

This chapter details the various fields of interest for the work at hand. The primary applicative domain, radar signal formation and processing, will be covered at first. Conceptually speaking, radar signals posses a rich underlying physical structure; as such, many mathematical representations, whether equivalent or not, aptly capture the signal's many properties. The key point is, while these different representation spaces feature a common ground, their structure may differ to a point where a thorough statistical analysis would require an altogther different learning framework. Specifically, we will focus on three main ways of representing a raw radar signal: the raw received radar wave, the Fourier transform of the wave (or more generally, any time-frequency representation), and the covariance of the wave. As such, concepts of Euclidean Machine Learning (ML), along with the fundamental algorithms used on the data, will follow the radar description. Then, we propose an introduction to Information Geometry (IG), which constitutes the rigorous, Riemannian framework for the analysis of the covariance representation, and finally give an overview of existing Riemannian ML methods.

Contents

2.1 Introduction **17**
2.2 Radar Signal and Simulation **18**
2.2.1 Radar Core Concepts 18
2.2.2 Representation of radar Signals 21
2.3 Euclidean Machine Learning **23**
2.3.1 Supervised Learning 23
2.3.2 Deep Learning 29
2.4 Information Geometry **38**
2.4.1 Riemannian manifold 39
2.4.2 Fisher information 40
2.4.3 Explicit computation of the Riemannian metric on Gaussian distributions 43
2.4.4 Tangent space 44
2.5 Riemannian Machine Learning **46**
2.5.1 Nearest neighbours in Riemannian space 46
2.5.2 Karcher algorithm for nearest Riemannian barycenter 47
2.5.3 Tangent space linear regression 49
2.5.4 Natural gradient 50
2.6 Conclusion **52**

2.1 Introduction

The classification of structured time series using methods of both Euclidean and Riemannian Machine Learning (ML) involves a rather involved comprehension of several different fields.

We begin this chapter with a description of micro-Doppler (μ-D) radar data, which both acts as a glowing instanciation of structured time series, and as the main applicative focus in our works. In a nutshell, a radar signal is formed by the reflection of an emitted wave on a target. The emitted signal is itself designed in a specific manner to assuage inherent inner physical constraints, and to optimize the reflected signal's resolution and ambiguity. A key feature in the received signal lies within the Doppler effect (Doppler 1842), from which speed and frequency information is deductible. By briefly explaining the core physical formation process of the signal, we hope to convince the reader that much is to gain by leveraging its underlying physical structure in learning models; by revealing the rich variety of representations arising from such structure, we wish to inspire the reader into believing this variety can also be incorporated within.

We then turn to the fundamentals of ML and its overall contextualization; in particular we distinguish between Supervised Learning (SL) and Unsupervised Learning (UL) (is the task given labelled examples to learn upon or not), and parametric and non-parametric learning (whether a parameterized distribution models the task or not). We rapidly tunnel our focus to Deep Learning (DL) as it will be the main algorithmic class we ambition to build upon, and specifically target our attention to the convolutional architectures initially developed in the context of Computer Vision (CV), and in particular Fully Convolutional Networks (FCNs), which heavily inspire the works presneted in Chapter 3.

We then switch to an overview of the field of Information Geometry (IG). The overall goal of this section is the introduction of mathematical framework allowing the manipulation of statistical distributions. In the process of doing so, we also wish to convey the fundamental differences between Euclidean, vectorial objects, and the more delicate members of Riemannian manifolds. In essence, we showcase useful tools for the following section, dedicated to...

Riemannian ML. In this final section, we aim at providing insight into algorithmic methods adapted from common-knowledge ML to the Riemannian realm of IG, through the demonstration of several simple learning algorithms. The objective is to pave the way to both following chapters (Chapter 3 and Chapter 4) by set-

ting a reasonably comprehensive contextualization of the proposed contributions within.

2.2 Radar Signal and Simulation

2.2.1 Radar Core Concepts

A standard radar consists in both an emitter and a receiver. The former emits a signal at a given wavelength, or more generally a given waveform (typically a linearly evolving frequency as depicted in Figure 2.1), while the latter is left to interpret all incoming signals. Let us consider a simple model where the emission frequency f_e is kept constant. The base waveform is itself emitted repeatedly, at every time intervall called Pulse Repetition Intervall (PRI), the inverse of which is called Pulse Repetition Frequency (PRF). The latter can be seen as the sampling frequency. This dual system induces many problems to be dealt with such as ambiguities in distance and velocity or compromises in time versus distance resolution.

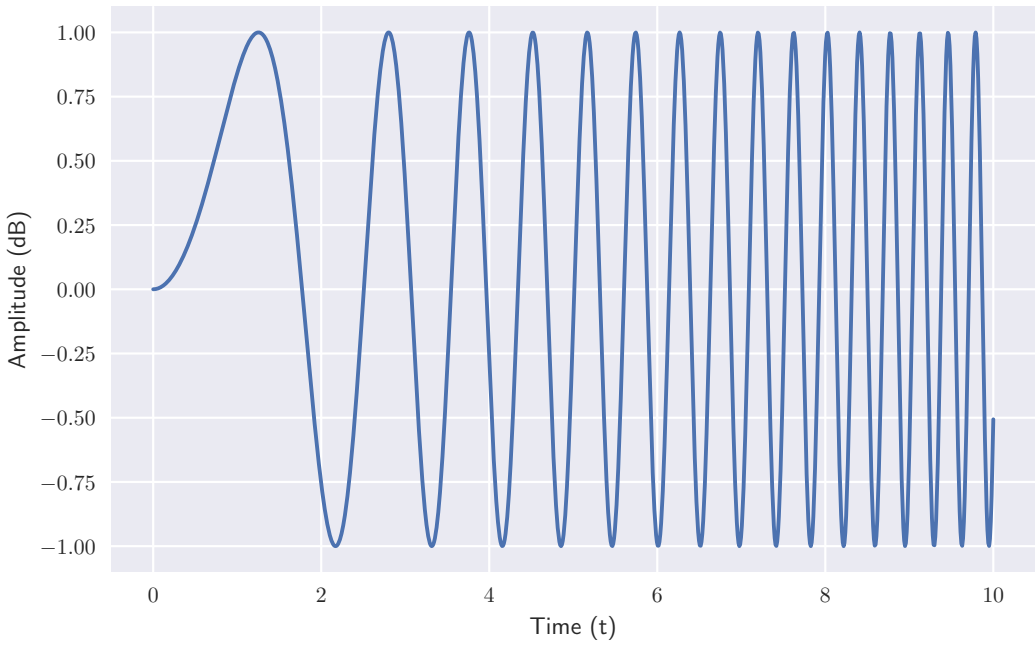

Figure 2.1.: **The basic form of a standard signal emitted by a radar**. The frequency typically grows linearly in time. Imposing a known frequency evolution helps solve ambiguities in the received signal.

Different compromises to solve different problems lead to a wide variety of radars which exist for different purposes: active radars combine emitter and receiver while passive radars only receive estranged signals, surveillance radars span a wide area of space by being poorly resolved while tracking radars sweep only a small portion of space to gain resolution on targets, antennaes are either fixed for a longer integration time or rotating for a better coverage, and the list goes on. The type of target also contributes to the choice of radar parameters; for instance, targets with smaller Radar Cross-Section (RCS) (see Figure 2.2) will require a more powerful radar, leading to further compromises. The curious reader may dig in detailed explanations in excellent references such as V. C. Chen et al. (2006).

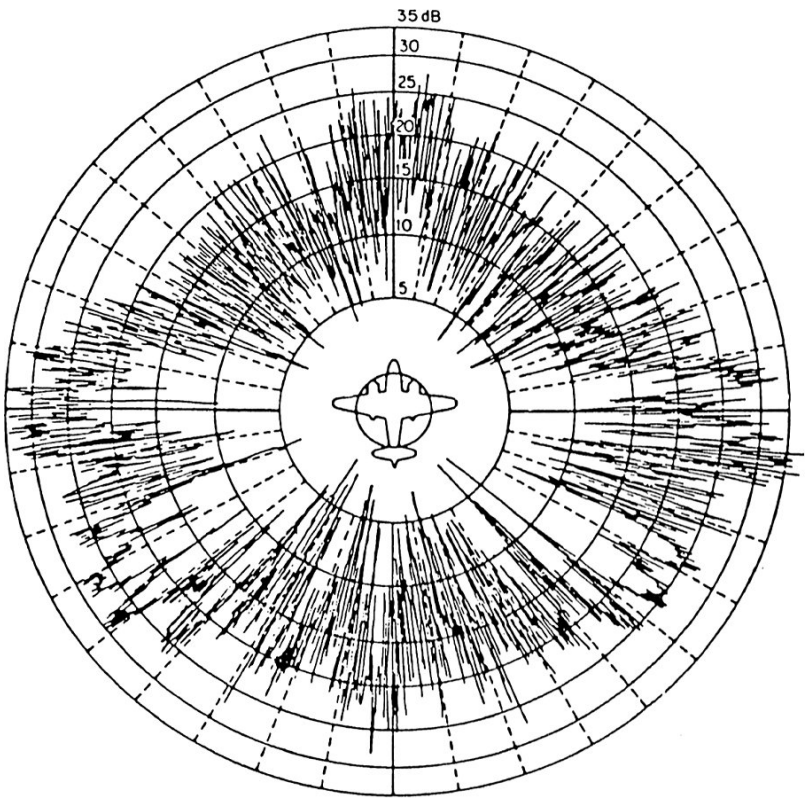

Figure 2.2.: **Illutration of the RCS**. Expressed in m^2, it represents the spatial distribution of reflectance across an object: as such, it is intrinsic to the object, and corresponds to a certain extent to its global shape size from the radar's point of view. This particular figure shows the RCS along a single rotation axis. Much research has been devoted to minimizing it to manufacture stealth jets.

While our work does not focus on a specific set of radars, it does give special attention to Unmanned Aircraft Vehicles (UAVs). These particular objects exhibit

shared characteristics, such as being blade-propelled or fix-winged (contrary to birds, for instance), small (contrary to wind turbines) and relatively symmetric RCS (contrary to airborne missiles), and potentially rapidly varying radial velocity V_R (contrary to commercial airplanes). Figure 2.3 shows three different drones studied in initial works: The Vario helicopter and DJI's Phantom2 and S1000+. Many UAVs such as those featured in the figure share specific characteristics:

- Blade rotation speed $\Omega = \frac{RPM}{60}$ expressed in $rad.s^{-1}$ (Rounds per Minutes (RPMs) being rotations per minute);
- RCS;
- Number N_b and length L_b of blades;
- Radial velocity V_r, the amplitude of the UAV's body's velocity vector projected on the axis linking it to the radar.

Figure 2.3.: **Depiction of the three drones used in simulations**. From left to right: The Vario helicopter, DJI's Phantom2, a quadrocopter, and S1000+, an octocopter. Scale is not respected.

As we can see, radar detection and classification is a task highly dependent on the parameters involved; to make a first analogy with CV, the finesse required to obtain "good" signals can compare to finetuning an ancient camera to particularly harsh photographic conditions. The catch is that no camera will ever be capture a drone hidden in a forest several kilometers away, while a properly calibrated radar will. Here we give a non-exhaustive list of intrinsic problems which are bound to limit our ability to aptly classify objects, which will hence become matter of attention in future study of radar models:

- Signal-to-Noise Ratio (SNR): classification can only be possible when the signal is actually detected; a scalar quantity which determines a detection threshold is the Signal-to-Noise Ratio, expressed in decibels, which represents how high in amplitude the signal is *w.r.t.* the background noise;

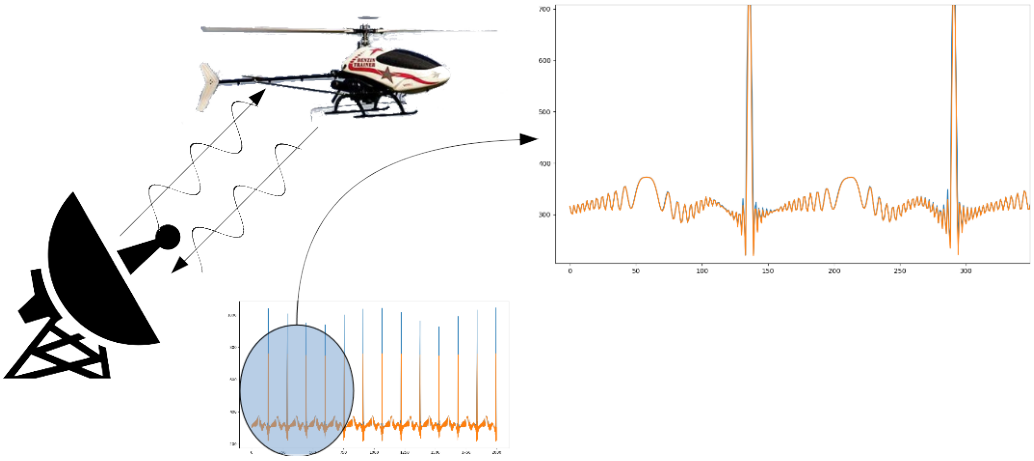

Figure 2.4.: **Illustration of the simulated signal**. Here, a PRF of $8kHz$ on a signal of $250ms$ yields a time series of 2000 complex points, jointly drawn as modulus and real part.

- PRF: the sampling frequency should by all means be high enough to avoid frequency amibiguities; indeed, according to the well-known Shannon condition (or Nyquist condition), the spectrum of a signal sampled at a lower rate than the maximum frequency of the signal is bound to fold on itself, thereby somewhat corrupting the spectrum; for instance, in classifying blade-propelled UAV, the PRF should ideally be proportionally higher than the fastest point on the UAV (typically the blade tip);

- f_e: the emission frequency range determines how finely we resolve variations in frequency between signals along with the spectral bandwidth; also considering a blade-propelled UAV, f_e should also be both proportionally higher than the fastest blade tip, but not so high as to induce spectral foldings.

2.2.2 Representation of radar Signals

As discussed in the previous section, a radar signal is the result of an emitted wave reflected off a target, sampled at a given frequency, which yields a numerical time series of complex points (amplitude and phase), as illustrated in Figure 2.4. This will serve as our base representation.

A good representation is the first step to any statistical inference, be it classification or feature analysis. In our case, the signal has an intrinsic time-frequency nature, an intermediary step being a pure frequency representation, *i.e.* a Fourier transform. A mere spectrum already allows for human interpretation and feature

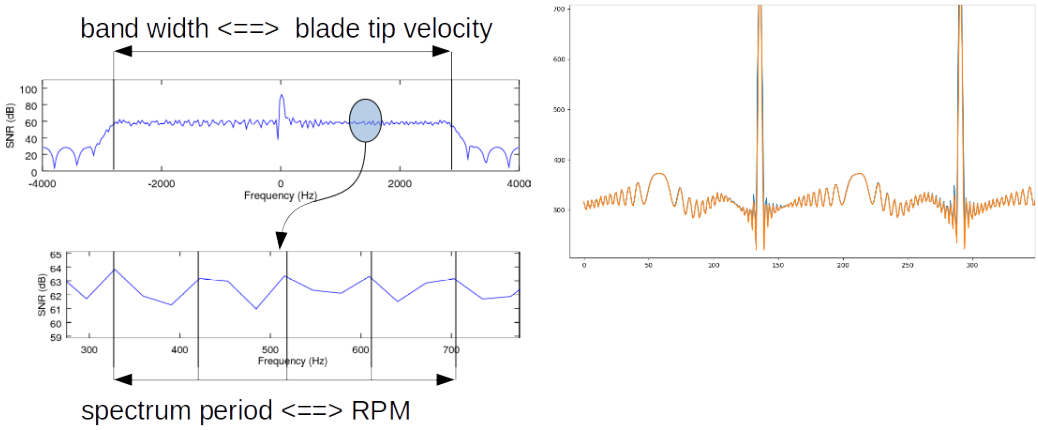

Figure 2.5.: **Illustration of the Fourier spectrum of a radar signal**. From a simple analysis of the featured spectrum, we can deduce three important characteristics of the UAV: rotation speed Ω, number N_b and length L_b of blades.

extraction such as blade rotation speed or RCS, as shown in Figure 2.5. In the figure:

- To the right: Zoom-in of the original signal;
- Top left: Complete Fourier transform ($PRF = 8kHz$, $SNR = 50dB$), where we define B as the signal's bandwidth (the plateau in the spectrum);
- Bottom left: Zoom-in of the spectrum, where we define ν the observed period of its assumed periodicity.

Then we have $\nu = N_b\Omega$ and $B = \frac{4L_b\Omega f_e}{c}$, which gives access to the three unkowns up to a hypothesis on the number of blades, clean analysis pending. Note that for the sake of clear figures, the signal representation parameters such as as the PRF and the SNR were set to wishfully accomodating values. It is also important to point out that a gain in SNR of $10\ \log(n_f ft) \approx 15dB$ is achieved in performing the Fourier transform. The given SNR values take that gain into account. However, the extraction will likely not be robust to real-world variations, thus nor will any subsequent classification algorithm.

2.3 Euclidean Machine Learning

This section gives an overview of the baseline concepts of ML developed in our classification algorithms. From a general point of view, the core of most learning algorithms is finding a representation space where the data are linearly separable, and then finding an optimal separation hyperplane in that space. In a more formal setting, ML aims at designing and building a function f on the inputs x, such that f outputs the decisions y involved in the task at hand. Although we have until now described f as a function, it can take the more general form of any algorithm yielding decisions from inputs. For instance, the form taken by f is usually split in non-parametric functions, and parametric functions. In our works we focus on the latter, where f is parameterized by a set of parameters Θ (we then note $f_{|\Theta}$, or $f(\cdot, \Theta)$, which are to be learnt through optimization of the loss function l. As briefly described in the introduction, ML can cover a variety of tasks, which are by no means limited to the usual framework of SL, which we notwithstanding focus on throughout this work. For excellent references on the matter of Machine Learning, we mention the following books, considered near biblical by many: Bishop (1995), Bishop (2006), Haykin and Haykin (2009), and LeCun et al. (2015), amongst of course many others.

2.3.1 Supervised Learning

The SL framework allows to learn f based on a dataset $\mathcal{D}$ of pairs $\{x_i, y_i\}_{i \in [1,N]}$ of inputs and corresponding decisions by minimzing a loss function l, designed to be high when the prediction $f(x)$ differs from the true decision y and vice-versa.

The most common example would be supervised image classification, where the inputs x are images and the decisions y are the image labels. For instance, f could be a linear discriminator $f : x \longmapsto w^t x$, and l could be the Mean-Squared Error (MSE):

$$l(f(x), y) = ||f(x) - y||^2, \tag{2.1}$$

or the Logistic Loss (LL):

$$l(f(x), y) = -y \log(f(x)) - (1 - y) \log(1 - f(x)), \tag{2.2}$$

which both encourage in their own way $f(x)$ to be close to y, as illustrated in Figure 2.6. The two loss functions shown share the same goal, but differ in behaviour and properties; as such, MSE in rather used in regression tasks while LL is dominant in classification.

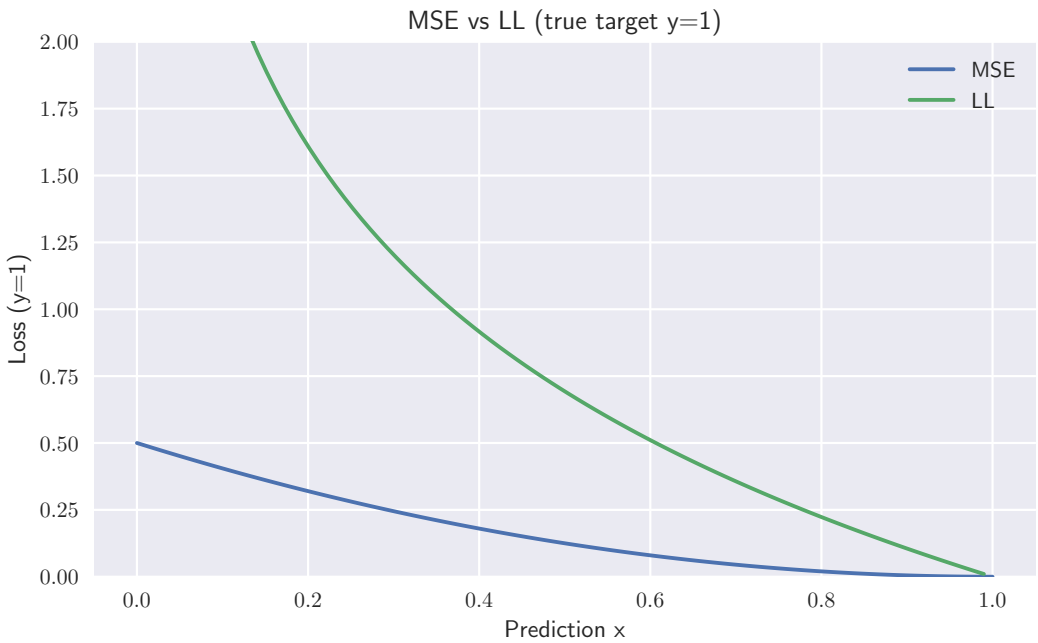

Figure 2.6.: **MSE versus LL.** Inputs to the functions are supposed normalized to $(0, 1)$; the plot shows the cost of labelling input x with true class $y = 1$.

We further describe the major classes of "traditional" ML algorithms. We first give the example of a class of non-parametric functions.

2.3.1.1 Nearest neighbours algorithms

Although parametric methods are usually favoured in SL, non-parametric functions have the conceptual edge of being straightforwardly adaptable in Unsupervised Learning (US), where clustering is a preferred task to classification. The nearest neighbours algorithms is an commonplace instance of the class of non-parametric learning algorithms.

k-nearest neighbours (k-NN) algorithm The main instanciation of nearest neighbours is the k-NN algorithm. Its mechanism is trivial to state: given a dataset of labelled inputs $\mathcal{D} = \{x_i, y_i\}_{i \in [1,N]}$, we seek to classify a new example x. To do so,

we collect the k data points $\{x_i\}_{i \leq k}$ (with a different ordering of points) which are the closest to x (given a metric d, usually the Euclidean metric). The label assigned to x is then the voted through the majority label in $\{y_i\}_{i \leq k}$.

Minimum Distance to Mean (MDM) algorithm A popular alternative to k-NN is the MDM algorithm. This time, given the dataset $\mathcal{D}$, we first compute the barycenters $\{\mu_c\}_{c \leq C}$ (again, given the metric d) of each class in the dataset, with C the total number of classes. Then, the new data point x is assigned to the closest barycenter. It is interesting to note that this algorithm is highly moldable depending on the choice of the metric d - more on that later.

k-means algorithm The k-means algorithm, first formulated in Steinhaus (1956) and popularized in MacQueen (1967), is a popular choice for unsupervised clustering, where only the number of classes C is known. Simply put, it is an iterative version of the MDM in an unsupervised setting: given a random barycenter initialization for each class $c \leq C$, points are assigned to their closest barycenter, as in the MDM, which yields C clusters. Then, barycenters are updated from the new clusters; this process in then iterated until convergence.

We now shift to parametric learning in its the simplest form, *i.e.* supervised linear classification, which performs the separation of data in the original input space.

2.3.1.2 Logistic Regression as a Baseline Classification Algorithm

Logistic regression is considered one of the most efficient (generalized) linear classification algorithms along with the Support Vector Machine (SVM) , and has known a tremendous success since its creation by Cox (1958) in many a field. It builds on logarithmically-scaled probability ratios between events: in a binary classification scenario, this model formalizes as $\ln(\frac{P_\omega(y=1|x)}{P_\omega(y=0|x)}) = \omega^T x$, with ω being the separating hyperplane, $x \in \mathbb{R}^n$ the random variable to be classified and $y \in (0, 1)$ the binary class. Put simply, if x belongs to class 0, $\omega^T x$ is negative, and positive otherwise, which naturally corresponds to modelling the distrinution $y|x$ as a Bernouilli random variable of parameter η. Furthermore, considering $P_\omega(y = 1|x) = 1 - P_\omega(y = 0|x)$, we have $P_\omega(y = 1|x) = \sigma(\omega^T x) = \eta$ with:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.3}$$

being the logistic, or sigmoid function, illustrated in Figure 2.7. Two useful properties of σ are $\sigma(-z) = 1 - \sigma(z)$ and $\sigma'(z) = \sigma(z)\sigma(-z)$.

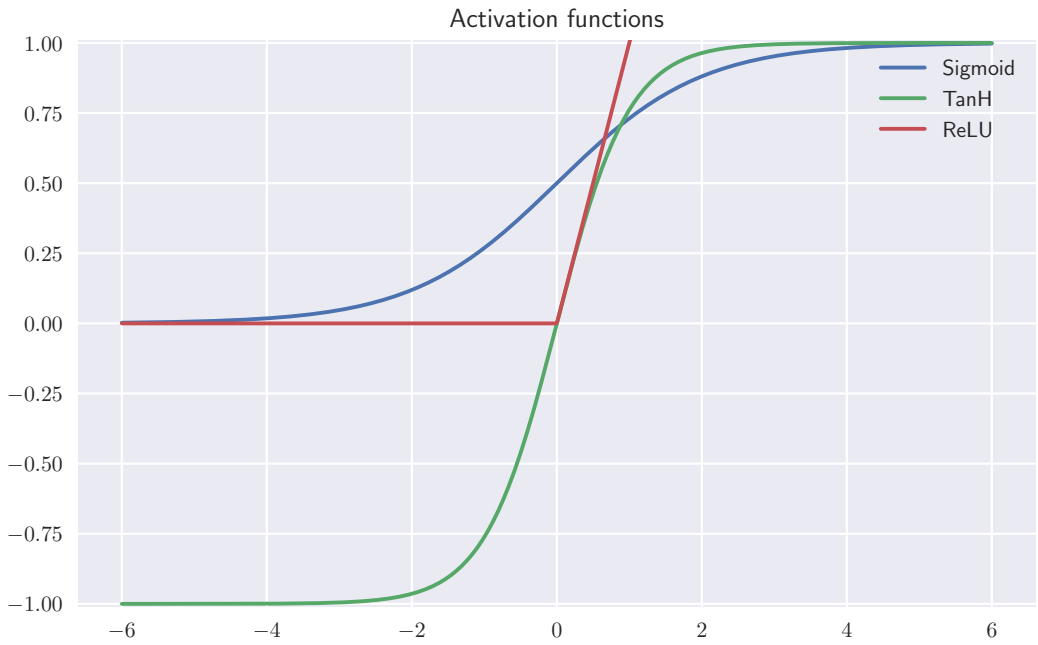

Figure 2.7.: **Activation functions** Along with the sigmoid used in logistic regression, two other popular choices of so-called "activations" are depicted: the hyperbolic tangent and the ReLU.

We will now detail how the hyperplane ω is computed. Though this computation is standard is the ML field, it will help us understand more advanced concepts in future sections. Given a dataset $\mathcal{D}$ of N pairs $\mathcal{D} = \{x_i, y_i\}_{i\in[1,N]}$, we wish to maximize the distribution log-likelihood of our Bernouilli distribution over $\mathcal{D}$:

$$\begin{aligned}
\max_{\omega\in\mathbb{R}^n} l_{\mathcal{D}}(\omega) &= \frac{1}{N} ln(\prod_{i=1}^{N} \eta_i^{y_i}(1-\eta_i)^{1-y_i}) \\
&= \frac{1}{N}\sum_{i=1}^{N} y_i ln(\eta_i) + (1-y_i)ln(1-\eta_i) \\
&= \frac{1}{N}\sum_{i=1}^{N} y_i \omega^T x_i + ln(\sigma(-\omega^T x_i))
\end{aligned} \tag{2.4}$$

To maximize $l_{\mathcal{D}}$, we minimize its opposite using Newton's method, which extends and refines gradient descent by multiplying to the left of the gradient the inverse of the Hessian:

$$\begin{aligned}\nabla\Big(-l_{\mathcal{D}}\Big)(\omega) &= \frac{1}{N}\sum_{i=1}^{N} -y_i x_i + \frac{x_i\sigma'(-\omega^T x_i)}{\sigma(-\omega^T x_i)} \\ &= \frac{1}{N}\sum_{i=1}^{N} x_i(\eta_i - y_i) \\ &= \frac{1}{N}X^T(\eta - Y)\end{aligned} \tag{2.5}$$

$$\begin{aligned}\Delta\Big(-l_{\mathcal{D}}\Big)(\omega) &= \frac{1}{N}\sum_{i=1}^{N} \eta_i(1-\eta_i)x_i x_i^T \\ &= \frac{1}{N}X^T D_\eta X,\end{aligned} \tag{2.6}$$

where $X \in \mathbb{R}^{N,n}$ is data design matrix such that $X(i,j) = x_i(j)$, similarly for η and Y, and $D_\eta = diag(\eta_i(1-\eta_i))$.

The update rule for computing the optimal hyperplane ω is thus as follows:

$$\omega^{(t+1)} = \omega^{(t)} - \alpha^{(t)}\Delta l_{\mathcal{D}}(\omega^{(t)})^{-1}\nabla l_{\mathcal{D}}(\omega^{(t)}). \tag{2.7}$$

In practice, we do not explicitly compute the inverse Hessian but rather solve for Z the linear system $\Delta l_{\mathcal{D}} Z = \nabla l_{\mathcal{D}}$. It is also important incorporate bias in the otherwise purely linear model. This can be done with no additional variable by using the bias trick: $\omega x + b = \omega^+ x^+$, with $\omega^+ = (\omega \cdots b)$ and $x^+ = (x \cdots 1)$. This trick is implicitly used throughout all algorithms.

2.3.1.3 The Perceptron as a link between Logistic Regression and Neural Networks

Also in 1958, F. Rosenblatt independently published a novel classification algorithm named the Multi Layer Perceptron (MLP) (Rosenblatt 1958), which aspires to mimick the way neurons and synapses process information within the brain. From another perspective, it aims to build a seperating hyperplane in a learnt feature space, as allustrated in Figure 2.8. We show in this section how a single layer perceptron with entropic loss is mathematically equivalent to logistic regression, thus providing a link between "black box"-ness of Deep Neural Networks (DNNs) and well-known and studied classical learning algorithms. Given the same scenario as above, nodes (which represent neurons) are connected together with weights (synapses) which in turn fire in response to an activation function σ (axon gated

channel). The goal is to adjust the weights such that a given input x, when passed through the network, outputs an estimation $\tilde{y}$ which matches the ground-truth y. Again, in the special case of binary classification, $y \in (0, 1)$, but we can now generalize to multi-class classification. There are different approaches to generalizing binary classification, such as one-versus-all or one-versus-one strategies, but the most used in DNNs and thus is in our work is the one-hot encoding, in which each label y is a C-dimensional vector with $y(c) = 1$ and 0 everywhere else, where C is the number of classes and c the particular class of y. The weights w_{ij} can be summarized in matrix W, hence the building block of MLPs:

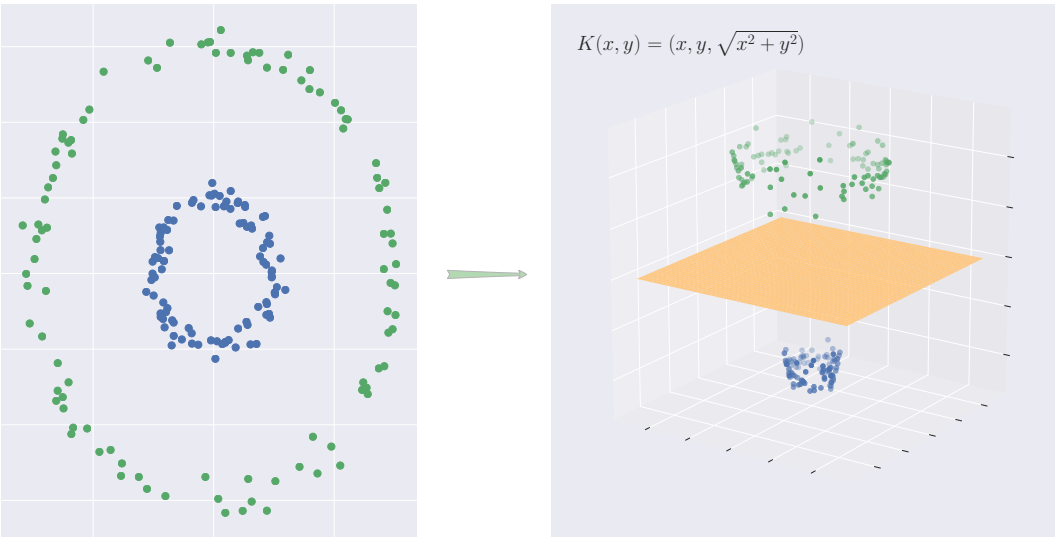

Figure 2.8.: **Illustration of the kernel trick**. Input data usually has no particular reason to be linearly separable in its raw form. A good representation space aims to that goal through the mapping K, from where we can then separate and classify the data.

$$X^{(k+1)} = f^{(k)}(X^{(k)}) := \sigma(X^{(k)} W^{(k)}) \tag{2.8}$$

In the equation above, we again use the bias trick, and X is the design matrix as defined in Equation 2.5. In a multi-class problem, the final sigmoid function is replaced by its generalized softmax formulation:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k \leq K} e^{z_k}} \tag{2.9}$$

A vanilla MLP is merely a stacking of L such layers such that $\tilde{y} = f^{(L)} \circ \cdots \circ f^{(0)}(x)$, upon which a loss is computed *w.r.t.* the ground truth y. The $L2$ loss is a widespread choice, but we will see how the cross-entropy loss, also popular, is more naturally suited to the classification problem:

$$\begin{aligned} l_{\mathcal{D}}(Y, \tilde{Y}) &= -\frac{1}{N}\sum_{i=1}^{N} y_i ln(\tilde{y}_i) + (1-y_i)ln(1-\tilde{y}_i) \\ &= -\frac{1}{N}\sum_{i=1}^{N} y_i ln(f\{x_i\}) + (1-y_i)ln(1-f\{x_i\}) \end{aligned} \tag{2.10}$$

At this point, we observe that the loss for a single layer perceptron matches exactly that of the logistic regression defined in Equation 2.4, with W acting as the separating hyperplane ω and η as $f(x)$, which yields the same weight update. This formal equivalence of these two algorithms not only allows for a interpretation of DNNs as stacked generalized linear models, it also inspires to attempt second order derivation of the loss as is done in logistic regression. However, the first step is standard gradient computation on stacked MLPs, which we describe in the next section.

2.3.2 Deep Learning

Using the notations dfined above: in DL, f is designed in a general fashion as a hierarchy of L sub-functions $\{f^{(k)}\}_{l\in[0,L)}$, commonly known as layers. Then, $f = f^{(L-1)} \circ \cdots \circ f^{(0)}$. Individual layers are made of elementary modules, such as linear layers, non-linear activations and so on, the idea being to allow the DNN to learn the representation space rather than fixing it through expert human decision. Indeed, while traditional ML models seek to learn a suitable decision function, the concept behind the stacking of layers is to learn both the feature representation space and the decision function, in a seamless end-to-end fashion (see Figure 2.9).

2.3.2.1 First-order Backpropagation

Gradients in DNNs are derived using the chain rule, leading to the gradient backpropagation algorithm, popularized by Cun (1988). Let us first introduce the useful intermediate variable $P^{(k)} = X^{(k)}W^{(k)}$ at each layer (k), such that $X^{(k+1)} = \sigma(P)$ and $\tilde{Y} = \sigma(P^{(L)})$. We first compute $\frac{\partial l_{\mathcal{D}}}{\partial P^{(L)}}$:

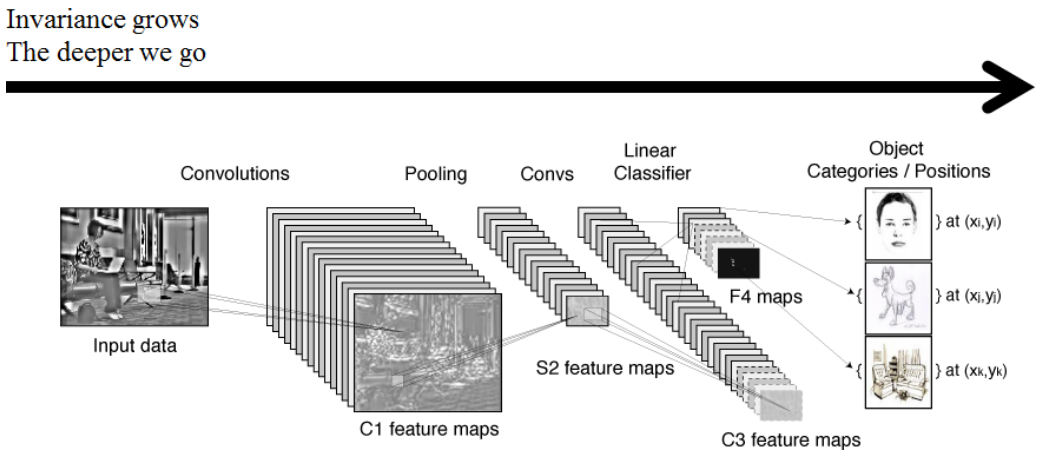

Figure 2.9.: **Representation power of DNNs through hierarchical learning.** Image source: Lawrence et al. (1997).

$$
\begin{aligned}
\frac{\partial l_{\mathcal{D}}}{\partial P^{(L)}} &= -\frac{1}{N}\frac{\partial l_{\mathcal{D}}}{\partial \tilde{Y}}\frac{\partial \tilde{Y}}{\partial P^{(L)}} \\
&= -\frac{1}{N}\left(\sum_{i=1}^{N} y_i \frac{1}{\tilde{y}_i} + (1-y_i)\frac{-1}{1-\tilde{y}_i}\right)\sigma'(P^{(L)}) \\
&= -\frac{1}{N}\sum_{i=1}^{N} y_i \left(\frac{1}{\sigma(p_i^{(L)})} + (1-y_i)\frac{-1}{1-\sigma(p_i^{(L)})}\right)\sigma(p_i^{(L)})\sigma(-p_i^{(L)}) \\
&= -\frac{1}{N}\sum_{i=1}^{N} y_i \frac{\sigma(p_i^{(L)})\sigma(-p_i^{(L)})}{\sigma(p_i^{(L)})} + (y_i-1)\frac{\sigma(p_i^{(L)})\sigma(-p_i^{(L)})}{\sigma(-p_i^{(L)})} \\
&= -\frac{1}{N}\sum_{i=1}^{N} y_i(1-\tilde{y}_i) + (y_i-1)\tilde{y}_i \\
&= \frac{1}{N}(Y-\tilde{Y})
\end{aligned}
\tag{2.11}
$$

Knowing that $P^{(k)} = \sigma(P^{(k-1)})W^{(k)}$ for any layer (k), we can then recursively derive $\frac{\partial P^{(k)}}{\partial P^{(k-1)}}$ using the chain rule:

$$
\frac{\partial P^{(k)}}{\partial P^{(k-1)}} = \left(\frac{\partial P^{(k+1)}}{\partial P^{(k)}} W^{(k)T}\right) \odot \sigma'(P^{(k)}) \text{ , where } \odot \text{ is the element-wise Hadamard product} \tag{2.12}
$$

We now have immediate recursive access to the derivatives of $l_{\mathcal{D}}$ *w.r.t.* the weights or data at every level to the network:

$$\begin{aligned}\frac{\partial P^{(k)}}{\partial W^{(k-1)}} &= X^{(k-1)}\left(\left(\frac{\partial P^{(k+1)}}{\partial P^{(k)}}W^{(k)T}\right)\odot\sigma'(P^{(k)})\right)\\ \frac{\partial P^{(k)}}{\partial X^{(k-1)}} &= \left(\left(\frac{\partial P^{(k+1)}}{\partial P^{(k)}}W^{(k)T}\right)\odot\sigma'(P^{(k)})\right)W^{(k-1)}\end{aligned} \tag{2.13}$$

The weights at every layer are then adjsuted accordingly. It is interesting to note there are "good" and "bad" ways to use the chain rule: this one initiates the recursion at the last layer; doing so from the first layer is possible but leads to computationally expensive high-rank tensor products. Backpropagation is the core of deep learning algorithms: more sophiticated architectures than MLPs require but per-layer gradients to implement. Such sophistication is shown in the next section.

2.3.2.2 Convolutional Neural Networks for Computer Vision

Further algorithmic refinements, formalizations and design branchings led the concept of perceptron to extend to that of fully-connected DNNs. It was in 1989 that LeCun et al. (1989) first proposed to replace dense layers with shared and locally connected layers (see Figure 2.10) to exploit locality in images as it was in Markov random fields (Cross and Jain 1983; S. Z. Li 1994), hereby creating the first convolutional network, or Convolutional Neural Network (CNN), popularized five to ten years later as $LeNet$ (Lecun et al. 1998), described in Figure 2.11, which beat all results over the six previous years (Bottou et al. 1994) of dense (or fully-connected) networks on the historical Modified National Institute of Standards and Technology (MNIST) handwritten digits database (Lecun et al. 1998). However, it was only in 2012 that Krizhevsky et al. (2012) popularized on a worldwide scale the usage of CNNs by winning by a large margin the tough ImageNet Large Scale Visual Recognition Challenge (ILSVRC) competition (Russakovsky et al. 2015) with their $ALexNet$ CNN depicted in Figure 2.12. Since then, CNNs have known an explosive amount of development, mostly in the field of CV, and until more recently, in other fields dealing with temporal series, such as sleep analysis (Aboalayon et al. 2016), human Brain-Computer Interface (BCI) (Lawhern et al. 2018), Automatic Speech Recognition (ASR) and Environmental Audio Recognition (EAR) (Takahashi et al. 2018; Piczak 2015) and, to a more limited extent, μ-D radar classification (Trommel et al. 2016).

The core component of a 2D CNN is the convolutional block, which inputs and outputs 3-D image volumes as shown in Figure 2.13. As in the figure, $x \in \mathbb{R}^{n\times n\times m}$ is passed through the m' filter banks f^i of filters f^{ij} to output $y \in \mathbb{R}^{n'\times n'\times m'}$, such that:

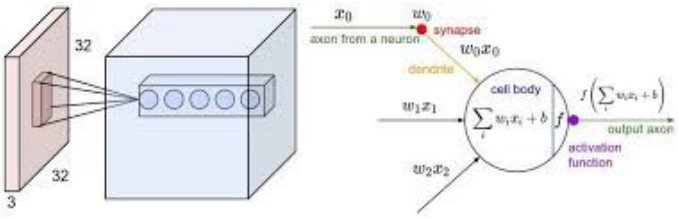

Figure 2.10.: **Illustration of the convolution as a linear transformation**. The weights in a CNN are shared and locally connected throughout the layers, contrary to the fully-connected DNN.

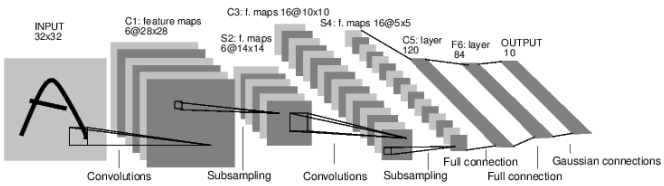

Figure 2.11.: **LeNet, the historical first CNN**.

$$\forall i \leq m', y^i = \sum_{j=1}^{m} f^{ij} * x^j \tag{2.14}$$

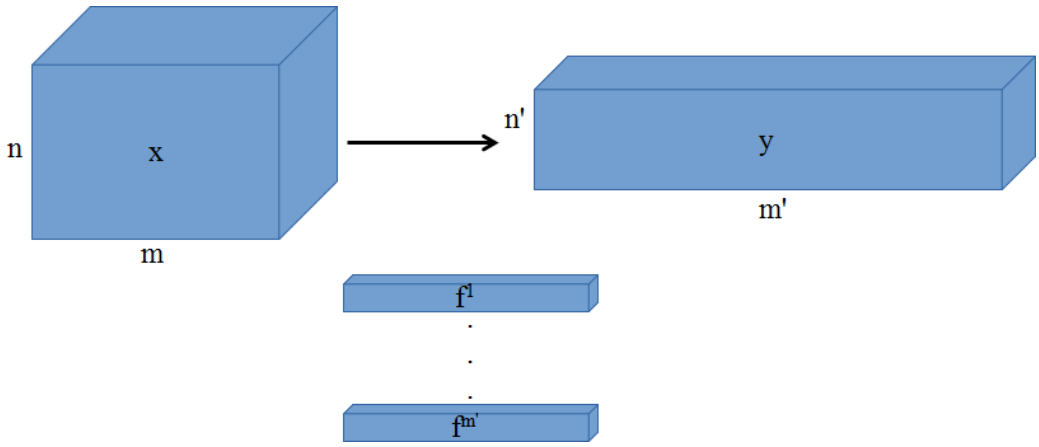

Figure 2.13.: **Illustration of a convolutional block**. A set of m' filters are applied to the input block. Each $n \times n \times m$ block can be seen as an $n \times n$ image with m artificial channels. For the sake of notational simplicity, we consider a square image, which may not be the case.

Anecdotally, Equation 2.14 can be rewritten as a fully-connected DNNs' core element defined in Equation 2.8 using a vectorized form of the image blocks

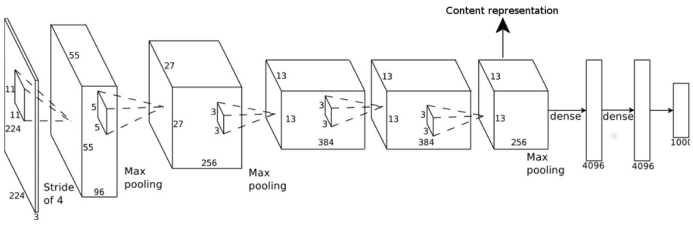

Figure 2.12.: **AlexNet, the 2012 ILSVRC winner**.

and a convolution matrix F representing the group of filterbanks f, which takes a Toeplitz form: $\tilde{y}^i = \sum_{j=1}^{m} F_{ij}\tilde{x}^j$, with F the Toeplitz matrix of convolutional coefficients.

2.3.2.3 Fully-Convolutional Networks

We now study a modification of CNNs which, although seemingly trivial, allows for semantic segmentation, *i.e.* pixel-level, or, in the case of radar classification, timestep-level classification: the FCN. A standard CNN such as VGG (Simonyan and Zisserman 2015) (see Figure 2.14) typically ends with one to three fully-connected layers, a legacy from MLPs: indeed, it seems natural to keep final representations as vectors rather than 3-D image blocks. One disadvantage is that all spatial relationships, which to this point were maintained in the previous convolutional layers, are now lost. It is in 2016 that J. Long and E. Shelhamer introduced the concept of FCNs (Long et al. 2015), where dense layers of size n are replaced with convolutional layers of size 1 and depth n. This artificial transformation is referred to in the paper as the "convolutionalization trick", illustrated in Figure 2.15. As explained in the caption, final image blocks are feature maps, whose pixels contain semantic information (labels). Upsampling the feature maps (typically 7×7 in the original paper), yields pixel-level segmentation. FCNs became an elegant state-of-the-art after the region-based family of CNNs: in 2013, Girshick et al. (2014) introduced the Region Convolutional Neural Network (RCNN) algorithm, which consists of region proposals sub-images within the image which are then passed through a pre-trained classification network; strong final activations are matched as semantically relevant. RCNN, followed by fast-RCNN (Girshick 2015) and faster-RCNN (Ren et al. 2015) in 2015, although a huge step forward in detection, lacked the unified framework and precision naturally allowed by FCNs, which we will in turn use for fine-grained signal classification.

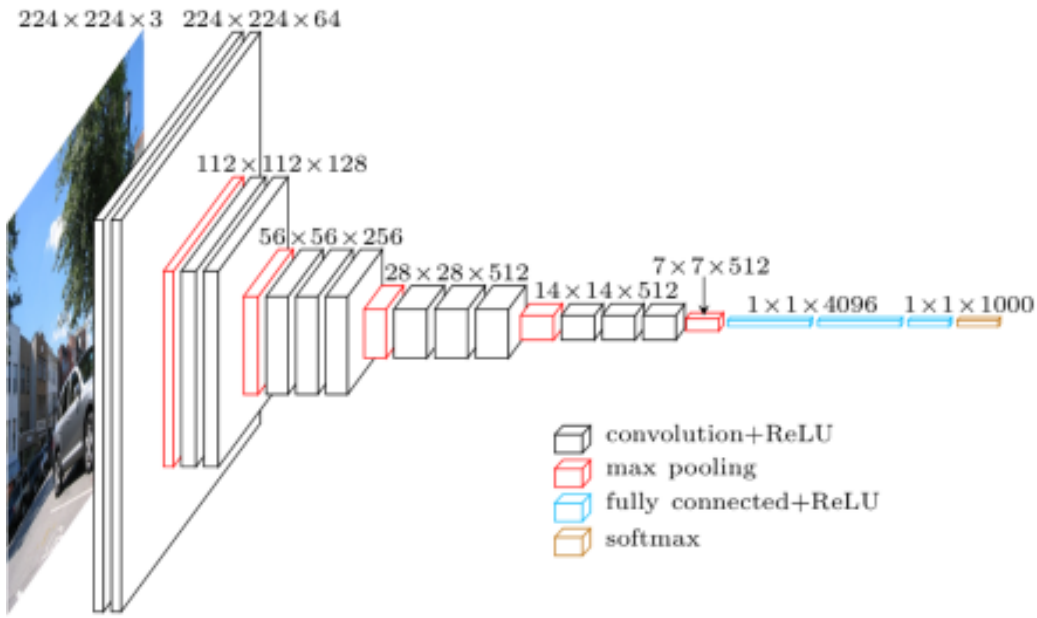

Figure 2.14.: *VGG* **neural network, second place in the 2014 ILSVRC challenge**.

The convolutionalization trick consists in replacing the fully-connected layers by 1×1 convolution filters, which actually sums up to a fully-connected layer, shared across the features. Indeed, continuing Equation 2.14, we have:

$$\begin{aligned} (f * x)_{n,m} &= \sum_{k,l} f_{k,l} x_{n-k,m-l} \\ &= f x_{n,m} \\ \Rightarrow \forall i \leq m', y^i &= \sum_{j=1}^{m} f^{ij} * x^j = \sum_{j=1}^{m} f^{ij} x^j \end{aligned} \tag{2.15}$$

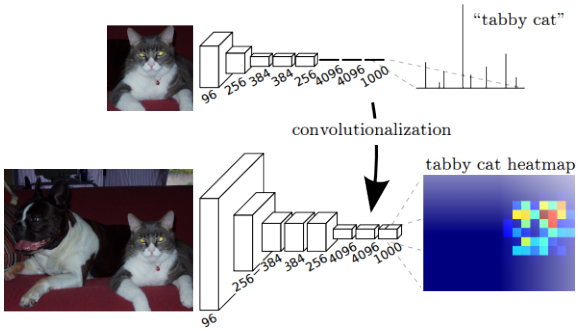

Figure 2.15.: **The convolutionalization trick**. The trick consists in replacing dense layers by convolutions of size 1, which maintains end-to-end spatial structure. The final convolutional layer can be seen as feature maps, or downsampled versions of the input image containing semantic information, *i.e.* spatial labels.

Building a FCN upon a backbone CNN is a conceptually trivial step to accomplish: a mere replacement of dense final layers, followed by a Global Average Pooling (GAP) or Global Max Pooling (GMP) layer, which sums up information from the feature maps to 1D features, necessary for classification (note that the segmentation information is taken right before the GAP).

2.3.2.4 Batch Normalization

CNNs and other convolutional architectures owe their success to careful design of, amongst many others parameters, the number of layers, the choice of activation functions, the filter sizes... However, novel layers aiming to regularize, lessen overfitting, or normalize outputs also proved to be major contributors to the phenomenal performances brought by these architectures. Batch Normalization (BatchNorm) is probably a most famed example of the latter, as first demonstrated by Ioffe and Szegedy (2015) and its (to this date) 16 078 citations.

The problem tackled by BatchNorm is the intensity imbalance between different firing neurons. While the learning of a network might favour a certain group of neurons, some with as much underlying relevance may be left aside by the vicissitudes of stochastic training. The formulation undertaken by the original authors is that of "covariate shift": the difficulty for marginally out-of-distribution, yet relevant examples to influence the weight updates within the network. BatchNorm aims to a fairer learning procedure, by imposing a normalization step after each block in the network. Concretely, given a batch of training examples, it subtracts the batch mean, then divides by the batch standard deviation. It then multiplies the result by a learnable scaling parameter, and adds a learnable parameter bias.

The consequence is, that whichever sub-distribution the current batch samples, the neurons fire to a standardized output.

The BatchNorm algorithm is now a default component in many state-of-the-art architectures. Its conception also resonates with our guiding principle of adapting learning models to underlying data distributions; as such, the concept will be summoned in further chapters, in a different context.

2.3.2.5 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) admit their foundations in the works of Bengio et al. (1994) and Hochreiter and Schmidhuber (1997) originally stem from standard perceptrons, but loop the inner states to learn on sequences of data rather than on individual, unordered points as illustrated in Figure 2.16. Let us recall the buiding block equation for MLPs Equation 2.8. The analog equation for RNNs is quite similar for it only adds the hidden state time dependency; using the notations in Figure 2.16, we have Equation 2.16:

$$\begin{aligned} s_{l+1} &= x_{l+1}U_l + s_l W_l \\ o_{l+1} &= \sigma(s_{l+1}V_l) \end{aligned} \tag{2.16}$$

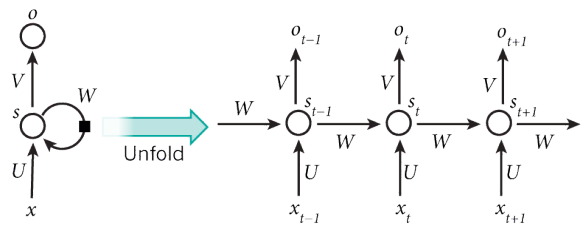

Figure 2.16.: **RNN with one hidden layer**. The series of data points x_i is passed in sequence to the network to become hidden states s_i which are function not only of x_i but also of all the s_j, $j < i$, hence of the previous data points themselves.

RNNs do not naturally handle out time-frequency images as they are vector-based. Nonetheless, by considering the spectrogam no longer as an image but rather as a series of 1D spectrums, which in essence it is, we face no more concern. Figure 2.17 shows the two different yet equivalent representations for a same signal. In the recurrent framework, the finesse coefficient $\mathfrak{k}$ is equal to its maximal value τ as we are dealing with individual spectrums. We can nonetheless relax $\mathfrak{k}$ by adding a stride in the sequential spectrums, which amounts to subsampling the original time-frequency representation.

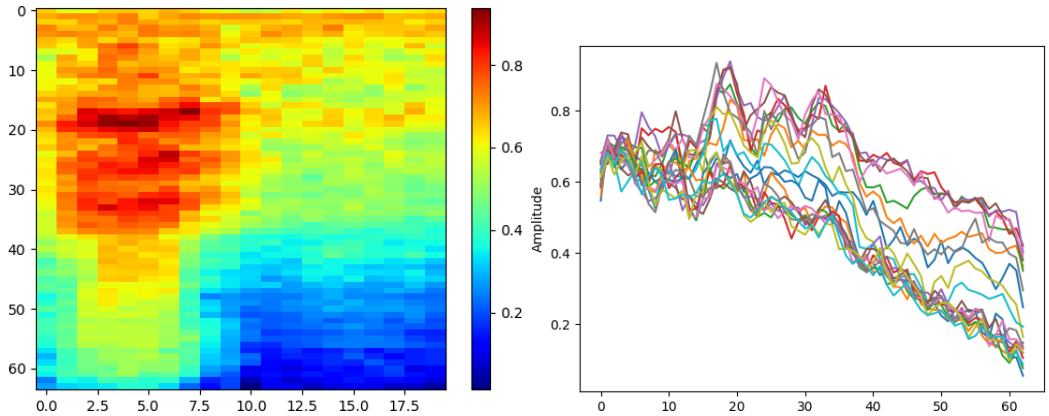

Figure 2.17.: **Illustration of the fundamental representation equivalence between a 2D time-frequency image and a series of 1D spectrums.**.

We have described here a standard, or vanilla, RNN, the one major drawback of which being what is called the problem of vanishing gradient, which we shortly describe here. RNNs accept in theory arbitrarily long sequences, and, as any neural network, are optimized using gradient backpropagation (specifically, the temporal unfolding of the network leads to an algorithm more complicated, yet identical in essence called Backpropagation Through Time (BPTT)). In a standard network, the gradient loses an order of magnitude in its computation at each layer, and by the time it reaches the first one it has decreased exponentially so, or vanished. RNNs suffer from the same drawback, not only in depth but also in time. To put it in a nutshell, a vanilla RNN has trouble learning long-term temporal relationships, as Bengio et al. (1994) shed light on, later supported by, notably, Pascanu et al. (2013). More sophisticated forms of recurrent networks have since been introduced, the most established at the time being Long Short-Term Memory (LSTM) networks (Hochreiter and Schmidhuber 1997) (see Figure 2.18 for a comparison of LSTMs with vanilla RNNs) and Gated Recurrent Unit (GRU) networks (Cho et al. 2014; B. Xu et al. 2015), along with blends with CNNs (Y. Xu et al. 2017). These architectures are widely spread in Natural Language Processing (NLP) and time series analysis and prediction (Sutskever et al. 2011). Further developments led to the now praised attention mechanisms in neural nets (Luong et al. 2015; Vaswani et al. 2017). One interesting theoretical justification of LSTM's architecture is given in Tallec and Ollivier (n.d.), where the authors build upon vanilla RNNs by adding invariance to slight time warpings, and end up with an architecture similar to the LSTM.

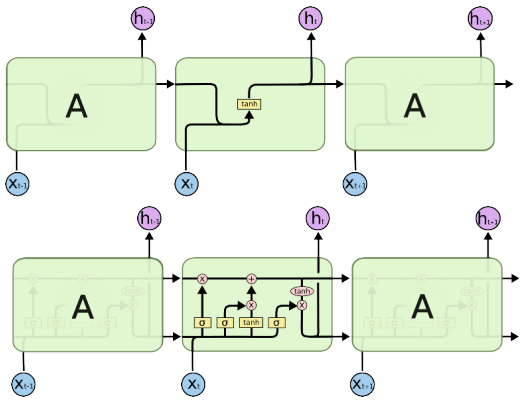

Figure 2.18.: **Illustration of a Long Short-Term Memory network**. LSTMs include a memory stream, which allows for unobtrused temporal data flow, along with a forget gate which leaves aside unnecessary information. Image credits to Colah's excellent blog article on RNNs: *Understanding LSTM Networks – colah's blog* (2017).

Until now, evoqued models handle vector-like data in a Euclidean fashion. As hinted in the introduction, much of our work focuses on models built on covariance representation of the data, *i.e.* on data lying within a Riemannian manifold. Although presented methods of DL are built to achieve high levels of representative abstraction, the acknowledgement of working on a possibly irregular grid opens up the possibility of combining the powerhouse mechanisms of DNNs and the inherent mathematical precision of the underlying data manifold. The curious reader may for example consult the works of Bronstein et al. (2017) on deep graph representation. The following section is dedicated to introducing the relevant fied of Information Geometry (IG), with the goal set in mind to fuse the concepts of Euclidean ML together with mathematical tools on Riemannian manifolds.

2.4 Information Geometry

Many signals reasonably lie within Euclidean spaces, where inner products and metrics are naturally defined and simple operations such as finding a barycenter of points are straightforwardly tractable and defined globally across the space. However, some entities intrinsically lie within a curved "sub-space", called manifold, defined by geometric, statistical or physical constraints. A simple example is illustrated in Figure 2.19. This section deals with the notion of locally defined

metrics on curved Riemannian manifolds, which generalize the concept of a flat Euclidean space. For instance, μ-D radar signals are successive series of pulses which can be represented as many realisations of a centered Gaussian process, the instantiation of which verifies certain constraints. The differential geometric study of statistical manifolds is the field of IG, for which we will introduce the important concepts in the following paragraphs.

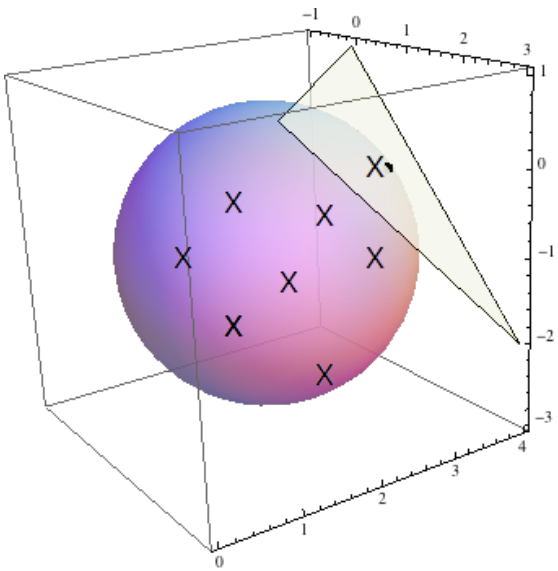

Figure 2.19.: **The sphere: a simple manifold**. In this figure, the cube is the Euclidean space $\mathbb{R}^3$. The data however intrinsically lie on the colored sphere, which is a manifold, *i.e.* can be locally approximated to its tangent plane.

2.4.1 Riemannian manifold

Formally, a n-manifold $\mathcal{M}$ is locally homeomorphic to a n-dimensional Euclidean space, *i.e.* $\forall\{x, R\} \in (\mathcal{M}, \mathbb{R}_+^*), \exists r \in \mathbb{R}_+^*$ and $\phi \in \mathcal{I}^0(\mathcal{M}, \mathbb{R}^n) \mid \phi(B_{\mathcal{M}}^{x,R}) = B_{\mathbb{R}^n}^{x,r}$, where $\mathcal{I}^0(\mathcal{M}, \mathbb{R}^n)$ contains all homeomorphisms, or continuous bijections, from $\mathcal{M}$ to $\mathbb{R}^n$ and $B_{\mathbb{R}^n}^{x,r}$ is the topological ball of $\mathbb{R}^n$ centred in x and of radius r. The manifold is also said to be Riemannian when it is equipped with a positive definite matrix $\boldsymbol{F}(\xi)$ defined $\forall \xi \in \mathcal{M}$, which is to be at least twice differentiable in ξ, such that the local squared distance ds^2 between two infinitesimallly close points ξ_0 and $\xi = \xi_0 + d\xi$ can be written as:

$$ds^2 = d\xi^T \boldsymbol{F}(\xi_0) d\xi, \tag{2.17}$$

where we note an element on the manifold $\xi = \{\xi_1, \ldots, \xi_n\}$ in a coordinate system for one homeomorphic local Euclidean space at point x in $\mathcal{M}$. We recall as a sidenote that the previous definitions also generalize in infinite dimension (the homeomorphic spaces are then pre-Hilbertian) and in addition for complex values (then they are Hilbert spaces). We will detail the results for real-valued n-manifolds.

2.4.2 Fisher information

The manifold we will put to use, is the space of multivariate centered Gaussian laws, which we will from now on note $\mathcal{S}_*^+$, or $\mathcal{S}_*^+(n)$ in a specific finite dimension n. The elements of the manifold are then entirely characterized by the $n \times n$ covariance matrix Σ_n. The parameter $\xi := \Sigma_n$ of one such distribution constitutes a coordinate system for the local Euclidean space. The fundamental definition of the covariance Σ of a random vector X of dimension n is:

$$\Sigma \;:=\; Cov(X, X) \;=\; \mathbb{E}((X - \mathbb{E}(X))^T (X - \mathbb{E}(X))). \tag{2.18}$$

Above, $\mathbb{E}(X)$ denotes the expectation of X. In practice, given a sequence of Independent and Identically Distributed samples $\{x_i\}_{i \in [1,N]}$ of X, the Sample Covariance Matrix (SCM) writes:

$$\Sigma \;=\; \frac{1}{N-1} \sum_{i=1}^{N} ((x_i - \overline{x})^T (x_i - \overline{x})). \tag{2.19}$$

Above, $\overline{x} \in \mathbb{R}^n$ denotes the sample expectation of the time series, *i.e.* the mean $\overline{x} = \sum_{i=1}^{N} x_i$. As stated before, the SCM $\Sigma \in \mathbb{R}^{n \times n}$ synthesizes the data's characteristic features in a compact and meaningful way.

Though we will later on focus on Gaussian distributions, the following computations hold for any member of an Exponential Family (EF). First of all, it is important to introduce the Kullback-Leibler (KL) divergence, which is a widespread tool to express how one distribution p_{ξ_1} "diverges" from another p_{ξ_2}:

$$D_{KL}(p_{\xi_1} \,||\, p_{\xi_2}) = \int p_{\xi_1} \ln(\frac{p_{\xi_1}}{p_{\xi_2}}). \tag{2.20}$$

The term "distance" would not be appropriate because D is not symmetric, nor does it verify the triangle inequality. It is however positive and equals zero *i.f.f.*

$p_{\xi_1} = p_{\xi_2}$, *i.e.* $\xi_1 = \xi_2$ for the same distribution family. From now on we will use the following simplified notations: $p_\xi \equiv p$ and $p_{\xi_0} \equiv p_0$, again with $\xi = \xi_0 + d\xi$, and we will use interchangeably a distribution and its parameter.

The KL divergence is derived from the even more fundamental concept of Shannon entropy (Shannon C. E. 2013):

$$H(p_\xi) = -\int p_\xi \ln(p_\xi) \tag{2.21}$$

Put simply, $D_{KL}(p_1 \,||\, p_2) = H(p_1) - H(p_1, p_2)$, which respectively represent the entropy of p_1 and the cross-entropy of p_1 and p_2. We will now see how equipping $\mathcal{M}$ with the KL divergence effectively provides it with a Riemannian structure. We define the function f_0:

$$f_0(\xi) = D_{KL}(p \,||\, p_0). \tag{2.22}$$

As we are dealing with exponential families, f_0 is indefinitely differentiable and by definition:

$$f_0(\xi_0) = 0. \tag{2.23}$$

We also have:

$$\begin{aligned}
\nabla f_0 &= \partial_\xi \int p_0(x) \ln(\frac{p_0(x)}{p(x)}) dx \\
&= 0 - \int p_0(x) \partial_\xi \ln(p(x)) dx \\
&= \int \frac{p(x)}{p_0(x)} \partial_\xi p(x) dx \\
\Rightarrow \nabla f_0(\xi_0) &= \int 1 * \Big(\partial_\xi p(x)\Big)(p_0) dx \\
&= \partial_\xi \int p(x) dx \\
&= \partial_\xi(1) \\
&= 0
\end{aligned} \tag{2.24}$$

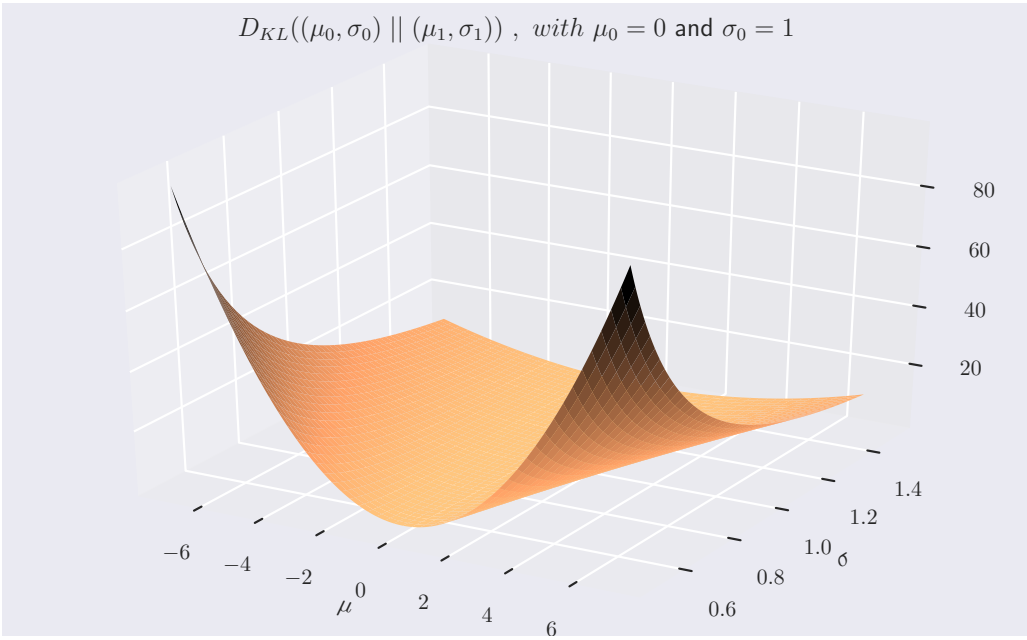

Figure 2.20.: **Kullback-Leibler divergence between two univariate Gaussians**. We fix a first distribution at (μ_0, σ_0) and plot the KL divergence from the second to the first. We see it annuls *i.f.f.* $(\mu_0, \sigma_0) = (\mu_1, \sigma_1)$.

By noting $l_x(\xi) = \ln(p(x))$ the log-likelihood of p, we have $\nabla f_0 = -\mathbb{E}_{\xi_0}(\partial_\xi l_x(\xi))$, which in p_0 yields the intuitive equality $\mathbb{E}_{\xi_0}(\partial_\xi l_x(\xi_0)) = 0$, which means the log-likelihood expectedly reaches a maximum at the optimal parameter ξ_0. Figure 2.20 plots the KL divergence between two one-dimensional Gaussian distributions, one being fixed, and exposes the to above cancellations. We now proceed to a second-order Taylor expansion in p_0, omitting the term in $o(||d\xi||^2)$. The last two equalities will annull orders 0 and 1, leaving only the second-order term:

$$\begin{aligned}
f_0(p) &= f_0(p_0) + d\xi^T \nabla f_0(p_0) + \frac{1}{2} d\xi^T \nabla^2 f_0(p_0) d\xi \\
&= 0 + 0 + \frac{1}{2} d\xi^T \nabla^2 f_0(p_0) d\xi \\
&= \frac{1}{2} d\xi^T \Big(- \nabla \mathbb{E}_{\xi_0}(\partial_\xi l_x(\xi_0)) \Big) d\xi \\
&= \frac{1}{2} d\xi^T \Big(- \mathbb{E}_{\xi_0}(\partial_\xi \partial_\xi l_x(\xi_0)) \Big) d\xi \\
\Leftrightarrow D_{KL}(p \,||\, p_0) &= \frac{1}{2} d\xi^T \boldsymbol{F}(\xi_0) d\xi.
\end{aligned} \tag{2.25}$$

$\boldsymbol{F}$ is the Hessian matrix of f_0, and specifically $\boldsymbol{F}(\xi_0) = -\mathbb{E}_{\xi_0}(\partial_\xi \partial_\xi l_x(\xi_0))$ is the Fisher Information Matrix (FIM) $F(p_0)$ of p_0. In other words, the Hessian of the entropy is the Fisher matrix of the distribution. To be exact, the formal and equivalent definition at index level of the FIM for a distribution p is:

$$\begin{aligned}
F_{ij}(p) &= \mathbb{E}_\xi(\partial_i l_x(\xi)\partial_j l_x(\xi)) \\
&= \int p\partial_i \ln(p)\partial_j \ln(p) \\
&= \int \Big(\partial_i p\Big)\Big(\partial_j \ln(p)\Big) \text{ , using } \partial\xi_i \ln(p) = \frac{\partial_i p}{p} \\
&= \int p\partial_j \ln(p) - \int p\partial_i\partial_j \ln(p) \text{ , using integration by parts} \\
\Leftrightarrow \mathbb{E}_\xi(\partial_i l_x(\xi)\partial_j l_x(\xi)) &= -\mathbb{E}_\xi(\partial_i\partial_j l_x(\xi)) \text{ , using } \mathbb{E}_\xi(\partial_i l_x(\xi)) = 0 \text{ (Equation 2.24)},
\end{aligned} \tag{2.26}$$

where ∂_i abbreviates the operator ∂_{ξ_i}. The above equations show the compliance of the KL divergence with a Riemannian structure as defined in Equation 2.17. The equations presented above show that, given a definition of entropy and divergence between distributions, we can build a notion of local metric on the distribution manifold. These remain valid for any EF; the following subsection studies the induced metric on the family of Gaussian distributions.

2.4.3 Explicit computation of the Riemannian metric on Gaussian distributions

To summarize, we have showed that the manifold of centred Gaussian laws admits a Riemannian metric which is the FIM. Typically, in $\mathcal{E} = \mathbb{R}^n$, we define $< \cdot \mid \cdot > : \mathcal{E}^2 \longrightarrow \mathbb{R}$ as $< x \mid y > = \sum_{i=1}^n x_i y_i$. The induced metric is then $|| \cdot || : \mathcal{E} \longrightarrow \mathbb{R}_+$ as $||x|| = \sqrt{< x \mid x >}$. Considering manifold $\mathcal{M}$, the metric is local to each point ξ and its tangent hyperplane $\mathcal{T}_\xi$, where $< \xi_1 \mid \xi_2 >_\xi = (\xi_2 - \xi_1)^T \boldsymbol{F}(\xi)(\xi_2 - \xi_1)$, where $\boldsymbol{F}(\xi)$ is the FIM of ξ. In particular, for $\xi_1 = \xi$, $< \xi_2 \mid \xi >_\xi = \nabla^2_{|\xi} D_{KL}(\cdot \mid \xi)$.

We now explicitly compute the FIM for Gaussian distributions by finding the Hessian of its entropy, which is known and easily derivable from the distribution's expression.

$$\nabla^2 ln\big((2\pi e)^n det(\Sigma)\big) = \nabla\Sigma^{-1}. \tag{2.27}$$

This in turn amounts to the natural local inner product of two symmetric matrices S_1 and S_2 on the tangent plane to any on covariance matrix $P \in \mathcal{S}_*^+$:

$$< S_1 \mid S_2 >_P = Tr(S_1 P^{-1} S_2 P^{-1}), \tag{2.28}$$

From there, integration along the shortest pathleads to the geodesic distance between two Symmetric Positive Definite (SPD) matrices P_1 and P_2, located anywhere on the manifold, also called Affine Invariant Riemannian Metric (AIRM), defined using the Frobenius norm $|| \cdot ||_F$:

$$\delta_{\mathfrak{R}}(P_1, P_2) \;=\; \frac{1}{2} || \log(P_1^{-\frac{1}{2}} P_2 P_1^{-\frac{1}{2}}) ||_F. \tag{2.29}$$

The alert reader may note that while the above metric is the correct one from the information geometric viewpoint, it is notoriously computation-heavy, mostly because of the involvement of eignevalue decomposition of symmetric matrices. Other metrics or divergences, either closely approximate it or provide an alternate theoretical apporach, while contributing the highly desirable property of lightweight computational complexity, especially in the modern context of machine learning. Notable examples may include the usage of the Fisher-Bures metric (Sun et al. 2019), the Bregman divergence (Boissonnat et al. 2010; Siahkamari et al. 2019; Banerjee et al. 2005), and optimal transport (Arjovsky et al. 2017).

Nonetheless, the AIRM remains a founding cornerstone for the exact study of covariance matrices on their correct manifold. Contrary to the local metric, related to the asymmetric KL divergence, it is a valid, globally-defined metric, usable as is within potential contenders for learning algorithms. This connection between entropy and differential metrics was first made in 1945 by Rao (1992) and in 1943 by Fréchet (1943), and further axiomatized in 1965 by Cencov (2000) and the 1976 confidential reportby S.T. Jensen cited in the works of Atkinson and Mitchell (1981). The keen reader may refer to Atkinson and Mitchell (1981) for the detailed derivations relative to the AIRM, and Burbea (1984) and Skovgaard (1984) for further details and proofs. We now move on to studying the link between the Riemannian metric and the manifold's tangent bundle.

2.4.4 Tangent space

The concept of tangent space allows for local Euclidean bearings, but to utilize it to its full extent we must define how to project points from the manifold to the tangent space and vice-versa, as is visualized in Figure 2.21. A Riemannian manifold defines a tangent space at each point; the set of all tangent spaces is referred to as the tangent bundle.

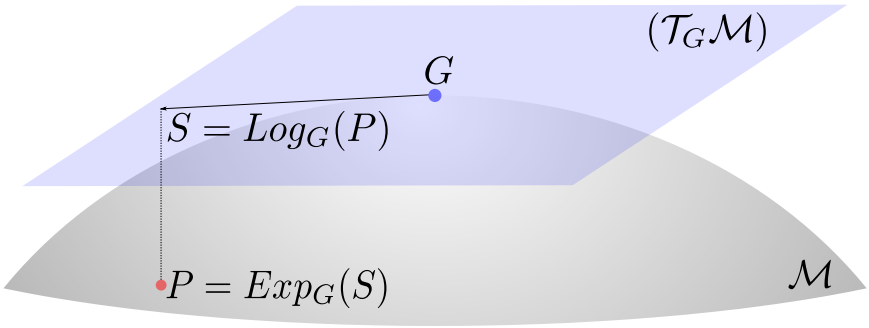

Figure 2.21.: **Manifold mappings**. This figure illustrates the mappings between the manifold $\mathcal{M}$ and the tangent space $\mathcal{T}_G$ in G. The bijection from $\mathcal{M}$ to $\mathcal{T}_G$ is called logarithmic mapping, and its reverse exponential mapping.

Moreover, at each point G, there exists a mapping operator to project from the manifold to the tangent space at this point, and its inverse. These are the exponential mapping and the logarithmic mapping, which are known in closed form on the manifold of SPD matrices $\mathcal{S}_*^+$:

$$\begin{aligned} \forall S \in \mathcal{T}_G, \mathrm{Exp}_G(S) &= G^{\frac{1}{2}} \exp(G^{-\frac{1}{2}} S G^{-\frac{1}{2}}) G^{\frac{1}{2}} \in \mathcal{M} \\ \forall P \in \mathcal{M}, \mathrm{Log}_G(P) &= G^{\frac{1}{2}} \log(G^{-\frac{1}{2}} P G^{-\frac{1}{2}}) G^{\frac{1}{2}} \in \mathcal{T}_G. \end{aligned} \tag{2.30}$$

In Equation 2.30, $\exp$ and $\log$ are the matrix exponential and logarithm functions, that is:

$$\begin{aligned} &\forall A \in \mathcal{S}_*^+ \text{ such that } A = Udiag(\lambda_i)U^T \text{ , U being an orthonormal diagonalising basis of A,} \\ &\qquad \exp(A) = Udiag(\exp(\lambda_i))U^T \\ &\qquad \log(A) = Udiag(\log(\lambda_i))U^T \end{aligned} \tag{2.31}$$

Notice non-linear operations intervene in the mappings' definitions: inverse matrix, square root; in fact, for symmetric matrices, these all reduce in complexity to eigen-decomposition, followed by the operation being applied on the vector of eigenvalues. Furthermore for SPD matrices, the Singular Value Decomposition (SVD), more stable and faster can replace the eigen-decomposition, as its eigenvalues are by definition strictly positive.

These two mappings are fundamental to the geometric understanding of the manifold, as they provide a bridge from a curved Riemannian setting to a stright

Euclidean one. For instance, taking the reference point as the identity matrix, the mappings equate to the standard matrix $\log$ and $\exp$ functions, which tells us the set of symmetric matrices is spanned by the $\log$ of SPD matrices. As a sidenote, which will eventually be developed upon in further chapters, this correspondance at the identity matrix defines a particular metric, the Log-Euclidean Metric (LEM).

The algorithmic consequence of these mappings, is they open the possibility of implementing learning methods in a Euclidean setting on the tangent space of some given reference point, which is a key feature of tremendous importance, illustrated throughout the various methods exposed in the following section on Riemannian ML.

2.5 Riemannian Machine Learning

In this section we describe the generalization of standard ML algorithms from Euclidean spaces to Riemannian manifolds. In fact, we limit ourselves to $\mathcal{S}_*^+$, the manifold of SPD matrices, in order to be able to fully derive the actual algorithms used later on. Moreover, we make an important distinction between the presented methods: on the one hand, some act directly upon manifold objects; on the other hand, some summon IG as a refinement to an otherwise Euclidean method. In this thesis, we rather use the first category, as the objects we consider are covariance matrices. We therefore describe three such methods: two non-parametric neighbour-based, and Tangent Space Linear Regression (TSLR). Finally, to illustrate the second category, we describe the better-known natural gradient.

2.5.1 Nearest neighbours in Riemannian space

Being now equipped with a Riemannian metric and manifold mappings, we will now build the very simple machine learning algorithm seen previously, the k-NN algorithm, only this time adapted to manifold-valued data, specifically here SPD matrices. Given the dataset $\mathcal{D}$ of pairs $\{P_i, y_i\}_{i\in[1,N]}$, we now classify new points according to the AIRM defined in Equation 2.29. We showcase the process in its basic form in Algorithm 2.1.

Algorithm 2.1 Riemannian k-NN algorithm on SPD matrices.

Require: Dataset $\mathcal{D} := \{P_i, y_i\}_{i\in[1,N]}$, new data point to classify P

```
L ← [0,0]^N                          // List to store distances to and labels of neighbours
for i ∈ [1,N] do
    L_i ← [δ_R(P_i, P) , y_i]
end for
sort_↓(L[:,·])                       // sort_↓ sorts a list in descending order
y ← mode(L[·,1:K])                   // mode returns the most frequent element of a list
return y                             // Label of P
```

2.5.2 Karcher algorithm for nearest Riemannian barycenter

Similar to above, we can also define the nearest barycenter algorithm in the Riemannian setting. Given a dataset $\mathcal{D}$ of N covariance matrices P_i spread in C classes, the latter takes place in two steps:

- Compute the barycenter for each class;
- Classify new points to the closest barycenter.

The first subtlety here is, while it may be possible to use the arithmetic mean $\frac{1}{N}\sum_{i\in[1,N]} P_i$, we will rather use the more geometrically appropriate Riemannian barycenter $\mathfrak{G}$, also known as the Fréchet mean (Yang et al. 2010a) or geometric average, which we also note $\mathrm{Bar}(\{P_i\}_{i\in[1,N]})$ or $\mathrm{Bar}(\mathcal{D})$. The Riemannian barycenter has shown strong theoretical and practical interest in Riemannian data analysis (Pennec et al. 2006), which justifies its usage in this context. By definition, $\mathfrak{G}$ is the point on the manifold that minimizes inertia in terms of the Riemannian metric defined in Equation 2.29:

$$\mathfrak{G} \;=\; \mathrm{Bar}(\{P_i\}_{i\in[1,N]}) \;:=\; arg\min_{G\in S_*^+} \sum_{i=1}^{N} \delta^2_{\mathfrak{R}}(G, P_i). \tag{2.32}$$

When the metric δ is Euclidean (*i.e.* based on a standard, globally-defined inner product), the minimization is tractable and leads to the arithmetic mean. However when it is Riemannian (*i.e.* based on a locally-defined inner product), it isn't (except in certain configurations, for instance in the centred Gaussian case, when the dimension is 1 or 2). The definition above is trivially extensible to a weighted Riemannian barycenter, noted $\mathrm{Bar}^w(\{P_i\}_{i\in[1,N]})$ or $\mathrm{Bar}^w(\mathcal{D})$, where the weights $w := \{w_i\}_{i\in[1,N]}$ respect the convexity constraint:

$$\mathfrak{G} \;=\; \mathrm{Bar}^{w}(\{P_i\}_{i\in[1,N]}) \;:=\; arg\min_{G\in S_*^+} \sum_{i=1}^{N} w_i\, \delta^2_{\mathfrak{R}}(G, P_i)\ \text{, with } \begin{cases} w_i \geq 0 \\ \sum_{i\in[1,N]} w_i = 1 \end{cases} \tag{2.33}$$

When $N = 2$, i.e. when $w = \{w, 1-w\}$, a closed-form solution exists, which exactly corresponds to the geodesic between two points P_1 and P_2, parameterized by $w \in [0, 1]$ (Bonnabel and Sepulchre 2010):

$$\mathrm{Bar}^{(w,1-w)}(P_1, P_2) \;=\; P_2^{\frac{1}{2}} \left(P_2^{-\frac{1}{2}} P_1 P_2^{-\frac{1}{2}}\right)^{w} P_2^{\frac{1}{2}}\ \text{, with } w \geq 0. \tag{2.34}$$

Unfortunately, when $N > 2$, the solution to the minimization problem is not known in closed-form, whether weighted or not: thus $\mathfrak{G}$ is usually computed using the so-called Karcher flow algorithm (Karcher 1977; Yang et al. 2010b), which we chiefly describe in Algorithm 2.2 and illustrate in Figure 2.22. In short, the Karcher flow is an iterative process in which data points projected using the logarithmic mapping (Equation 2.30) are averaged in tangent space and mapped back to the manifold using the exponential mappings (Equation 2.30), with a guaranteed convergence on a manifold with constant negative curvature, which is the case for $\mathcal{S}_*^+$ (Karcher 1977). The initialization of $\mathfrak{G}$ is arbitrary, but a reasonable choice is the arithmetic mean. Another point of interest is that selecting $K = 1$ (that is, only one iteration of the flow) and $\alpha = 1$ (unit step size) in the Karcher algorithm (Algorithm 2.2), corresponds exactly to the barycenter from the LEM viewpoint (Pennec et al. 2006).

Algorithm 2.2 Karcher flow Karcher 1977 to compute the Riemannian mean of N SPD matrices.

Require: data points $\{P_i\}_{i\in[1,N]}$, iterations K, step α

1: $\mathfrak{G} \leftarrow \sum_{i\in[1,N]} P_i$
2: **for** $k \leq K$ **do**
3: $\quad G \leftarrow \frac{1}{N} \sum_{i\in[1,N]} \mathrm{Log}_{\mathfrak{G}}(P_i)$
4: $\quad \mathfrak{G} \leftarrow \mathrm{Exp}_{\mathfrak{G}}(\alpha G)$
5: **end for**
6: return $\mathfrak{G}$

Once the geometric mean is known for all classes, we simply pick the one with the smallest geometric distance for a new point to classify. It is interesting to note (and easy to derive) that the geometric mean in $S_1 = \mathbb{R}_*^+$ does correspond to the definition learnt in highschool: $\mathfrak{G}_c = \sqrt{x_1 \cdots x_{N_c}}$. The nearest Riemannian barycenter, also named Minimum Riemannian Distance to Riemannian Mean (MRDRM) in works such as in Barachant et al. (2012), achieves state-of-the-art results in certain fields of highly structured time series data, notably in

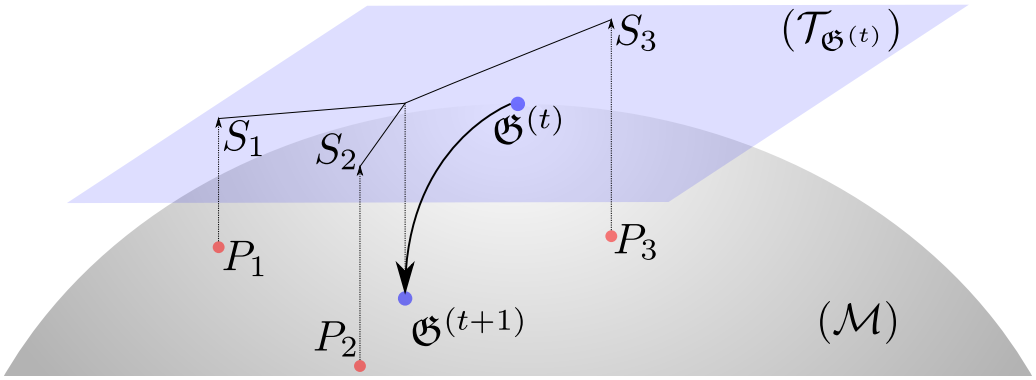

Figure 2.22.: **Illustration of one iteration of the Karcher flow (Karcher 1977).** Data points P_i are logarithmically mapped to the S_i on the tangent space at $\mathfrak{G}^{(t)}$. The S_i are then arithmetically averaged, the result of which is exponentially mapped back to the manifold, yielding $\mathfrak{G}^{(t+1)}$.

BCI applications. Also, pioneering works such as Yger and Sugiyama (2015) for electroencephalography (EEG) modelisation, or Yang et al. (2010b) for radar classification, make use of a MRDRM scheme to reach high performance with relatively lightweight machinery. Nearest baryencter schemes still remains a simple algorithm from the ML viewpoint, which strongly encourages the usage of IG in ML applications, allowing models better fit to the inout data and its inherent physical conditioning.

2.5.3 Tangent space linear regression

The generalization of nearest neighbours to manifolds was straightforward once the appropriate tools were identified and known, *i.e.* the metric, and, perhaps less trivial, the Riemannian barycenter. Nonetheless, their non-parametric nature helped bridge the gap quite effortlessly. On the other hand, parametric learning methods are by design often specifically built for Euclidean spaces: in the founding case of linear regressions, the algorithms' parameters involve projecting to seperating hyperplanes, a tricky notion in manifolds. The key idea to convey here is that one can circumvent this mathematical obstacle by instead projecting the manifold data to the tangent bundle, through the logarithmic mapping defined in Equation 2.30, and setup the linear regression in this now Euclidean space: this procedure is called TSLR, and is successfully used in the analysis of highly-structured data, such as in BCI (Barachant et al. 2012; Barachant et al. 2013). The choice of projection anchor is thus of prime relevance. One could default to the identity matrix by lack of better choice. However, chosing a point better

suited to the data could perhaps enjoy better final performances: for instance, the Riemannian barycenter of the studied data points seems a promising candidate.

In the case of covariance matrices $\{P_i\}_{i\in[1,N]}$, we now know how to build the Riemannian barycenter $\mathfrak{G}$ using the Karcher flow (Algorithm 2.2). The manifold points P_i are then mapped to vectors s_i as such:

$$s_i \leftarrow vec\left(\mathfrak{G}^{\frac{1}{2}} \log(\mathfrak{G}^{-\frac{1}{2}} P_i \mathfrak{G}^{-\frac{1}{2}}) \mathfrak{G}^{\frac{1}{2}}\right). \tag{2.35}$$

Then, any linear method, such as SVM or logistic regression can be applied to the transformed problem. The projection can even be reformulated as a kernel transform, as shown in (Barachant et al. 2013), which paves the way for further generalizations.

2.5.4 Natural gradient

We have seen how the standard, Euclidean gradient works: to rephrase it synthetically, we simply choose, in the parameter space, an optimal direction vector ν^* on an ϵ-ball around current parameter θ such that it minimizes the loss function l:

$$\nu^* = \underset{d(\theta,\nu)=\epsilon}{\arg\min}\, l(\theta + \nu). \tag{2.36}$$

The optimization yields the negative gradient $(-\nabla_\theta l)$ of the loss *w.r.t.* the parameter, thus the descent reads:

$$\theta \leftarrow \theta - \alpha\, \nabla_\theta l. \tag{2.37}$$

Above, α is the learning rate.

In gradient descent, the result is dependent on the topological shape of the ϵ-ball around θ. In this Euclidean version, which constitutes the industry standard, this ball is isotropic as it is defined by the Euclidean metric. However, we have also learned from the section on IG that the Euclidean metric is an ill-advised candidate for studying distributions: rather, the natural, Riemannian metric $\delta_{\mathcal{R}}$ inherent to the distribution's underlying manifold should be used. In this context of optimization on an ϵ-ball, the infinitesimal version of the metric suffices, *i.e.* by construction the KL divergence, or equivalently the second-order term of the Taylor expansion, which involves the FIM $G(\theta)$ of $l(\theta)$ (recall Equation 2.25). The feasible surface of the problem now follows the curvature of the distribution's manifold:

$$\nu^* = \underset{D_{KL}(\theta|\nu)=\epsilon}{\arg\min}\ l(\theta+\nu). \tag{2.38}$$

We now solve Equation 2.38, by first writing its Lagrangian form, using the equivalence between the KL divergence and the Taylor expansion and also developing the loss function to the first order:

$$\begin{aligned}
&\nu^* = \underset{D_{KL}(\theta|\nu)-\epsilon=0}{\arg\min}\ l(\theta+\nu) \\
&\Leftrightarrow \nu^* \sim \underset{\nu}{\arg\min}\ l(\theta) + (\nabla_\theta l)^T\nu + \lambda\Big(\nu^T G(\theta)\nu - \epsilon\Big) && \text{(Lagrangian)} \\
&\Leftrightarrow 0 = \nabla_\nu l + \nabla_\theta l + \lambda G(\theta)\nu && \text{(setting the gradient } w.r.t.\ \nu \text{ to zero)} \\
&\Leftrightarrow\ -\lambda G(\theta)\nu = 0 + \nabla_\theta l && (l \text{ only depends on } \theta) \\
&\Leftrightarrow \nu = -\frac{1}{\lambda}\, G^{-1}(\theta)\ \nabla_\theta l && \text{(we assume invertibility)}
\end{aligned} \tag{2.39}$$

We can now finally write the natural gradient update:

$$\theta \leftarrow \theta - \alpha G^{-1}(\theta)\nabla l(\theta). \tag{2.40}$$

The resolution above reveals the gradient update is distorted by the inverse Fisher matrix, *i.e.* the Cramer-Rao bound of the distribution family parameterized by θ. The FIM being the Hessian of a convex function, the operation can be seen as an orthonormal basis change: to complete the link with IG concepts, this amounts to rigidly transforming the canonical Euclidean space to the tangent plane of the manifold at θ. Again, this manifold represents the feasible distributions according to $l(\theta)$ - in the case of a neural network distribution, it is referred to as the neuromanifold, first introduced by Amari (1998), and perhaps made popular in CV by (Pascanu and Bengio 2014).

Notice the resemblance of the natural gradient with second-order optimization, notably Newton's method, where the FIM is replaced by the Hessian of the loss (Boyd and Vandenberghe 2004). It is interesting to note the two methods are mathematically equivalent for the logistic regression, one single-layered perceptron, presented previously, although in general they are not. They do however share the prohibitively computational cost of high-order methods: while the standard Stochastic Gradient Descent (SGD) requires $\mathcal{O}(n)$ operations (n being here the dimension of θ, in other words the number of scalar parameters), the former typically require $\mathcal{O}(n^3)$. While computational improvements along with attempts to democratize the Riemannian gradient, it remains to this day a niche method, seldom used in DL applications.

Yet again, the full potential of such Riemannian methods based on IG seems yet rather unexploited by the general community. These methods show great theoretical and practical advantages, and strongly hint to the possibility of successfully applying them to structured data, a hint we will make sure to follow in the next chapter.

2.6 Conclusion

In this chapter, we gave an overview of radar data formation and structure, Euclidean Machine Learning, Information Geometry, and finally Riemannian Machine Learning. With the presented tools at hand, we are now able to imagine an inceptive framework to perform, say, micro-Doppler radar classification.

Given a signal, which we may sensibly represent as a real-valued spectrogam, we can build a vector form out of it, and train, for instance, a Multi Layer Perceptron on the dataset. Recall that a single-layered perceptron is equivalent to logistic regression, which can serve as a baseline algorithm. Of course, doing so destroys the structure of the data from the start, which defies the guiding principle behind our works. It is however a valid approach used in the literature, which we will compare against.

Following the same example, a finer method would consist in interpreting the spectrogam as a time series of spectra. In this scenario, an individual spectrum takes a vector form, and a sequence of such can be modelled by any form of Recurrent Neural Network. Alternatively, one may consider joint temporal and frequential locality, *i.e.* interpret the spectrogam as an image, to be modelled through a Convolutional Neural Network. In both cases the structure of the signal is taken into account within the learning models.

While using convolutional architectures on micro-Doppler data constitutes a conceptual improvement over the naive purely vectorial approach, treating the time and frequency dimensions in a symetric fashion proves an infringement to their inherent physical difference. To this end, designing rectangular (as in, non square) filter banks allows to better control each dimension's behaviour within the network; using separable convolutions would push the concept further by handling time and frequency independently. Moreover, as it sometimes happens in Computer Vision, inputs may have different shapes; this is even more common in radar signals, where the acquisition pipeline, as well as the tracking of non-cooperative targets, potentially leads to a high disparity in signal durations. Here, a standard Convolutional Neural Network might strugle to handle this variability,

to the point of malevolently discarding it in the final fully-connected layers. This issue justifies the use of Fully Convolutional Networks, which we also described in the chapter. To repeat the main interest of the Fully Convolutional Network, the temporal structure of the input data is preserved throughout the whole network, allowing a fine-grained classification - in fact, we will see later on it is possible to exactly control the temporal resolution of the network.

The above developments give a first example of adapting popular notions in Machine Learning to the structure of data. In the case of radar data, this kind of approach had not yet been thoroughly explored: while Convolutional Neural Networks and Recurrent Neural Networks had indeed been used, filter bank design and temporal resolution analysis was non existent; we will argue these notions are key to implementing more efficient models, which perform better while remaining relatively lightweight. Specifically, we will demonstrate the development of a Fully Temporal Convolutional Network (FTCN) adapted to the structure of the data, based on the key elements described throughouth this chapter. Moreover, this architecture will be expanded in to the complex domain, pushing further the potential power and flexibility of these new deep architectures, named $\mathbb{CR}$ Neural Network ($\mathbb{CR}$Net) and Fourier Neural Network (FourierNet).

Going further on, we will also harness the strengths of Information Geometry to directly study the underlying distribution of the data; specifically, centered Gaussian distributions, characterized by their Sample Covariance Matrix. To this end, we will make use of Riemannian Machine Learning models presented above, along with a covariance-based Riemannian neural model, and go even further by proposing to unite in both Euclidean and Riemannian models in a single classification pipeline, thus leveraging the data's structure from multiple angles. This pipe-like model, which we call Second-Order Fully Temporal Network (SOFTNet), is expected to gather the best from both Euclidean and Riemannian worlds, *i.e.* yield competitive performance on large datasets while remaining robust to the lack of data, a key issue throughout our studies, as previously stated.

The final chapter of the thesis will focus on the deep covariance-based models, making use of the IG knowledge to propose novel layers and improve upon existing architectures. Concretely, we will first generalize to the AIRM mathematical framework presented in this chapter of a key layer to the main existing deep SPD architecture, yielding a Data-Aware Mapping Network (DAMNet) architecture. We then further generalize the DAMNet to a Riemannian BatchNorm algorithm respecting the manifold's geometry, amounting to the Batch-Normalized SPDNet (SPDNetBN) architecture, which thus posses another level of regularization compared to the original models. All improvements are based on knowledge of the distribution's underlying manifold, such as notions of manifold mappings or Parallel Transport (PT), allowing for models intrinsically better fit to the data.

A proposal of convolutional SPD architecture will also be described, in resonance with the convolutional nets described in this chapter.

The detailed presentation of the proposed Fully Temporal Convolutional Network and complex-valued counterparts, the base Riemannian model and the full SOFTNet pipeline is the subject of Chapter 3. Then, Chapter 4 proposes the theoretical and practical improvements on the covariance-based neural model.

CHAPTER 3

SECOND-ORDER PIPELINE FOR TEMPORAL CLASSIFICATION

Chapter abstract

This chapter is purposed to disentangle the possible dilemma arising from the multiplicity of valid representations and associated learning models introduced in the previous chapter in the context of temporal signals. The chapter's overall contribution is to show how we can make use of all representations in a single pipeline incorporating both Euclidean and Riemannian learning frameworks. To achieve this end, we first describe a deep Riemannian learning model, specifically a neural network operating directly on Symmetric Positive Definite (SPD) matrices. We then show how to combine the Euclidean models operating on first-order components (raw signal, time-frequency representation), and the Riemannian model operating on second-order components (covariance matrix), which we refer to as Second-Order Fully Temporal Network (SOFTNet). We also demonstrate experimentally the compound benefits of this pipeline model, compared to its elementary components.

Contents

3.1 Introduction **57**
3.2 Learning on structured time series representations **59**
3.2.1 SPD neural networks 59
3.2.2 Fully Temporal Convolutional Network 62
3.2.3 Complex Fully Temporal Convolutional Network 66
3.2.4 HPD neural network 68
3.3 Full pipeline for temporal classification **71**
3.3.1 The Fourier convolution layer 71
3.3.2 Pipeline bifurcations 72
3.3.3 Covariance pooling 74
3.4 Experimental validation **75**
3.4.1 Drone micro-Doppler radar data simulator 77
3.4.2 Experiments and results 80
3.5 Conclusion **91**

3.1 Introduction

The first part of this chapter makes use of the structured time series representations, along with the possible learning models introduced in the previous chapter. Succintly, a temporal signal such as micro-Doppler (μ-D) data can take a variety of forms, each one highlighting a different set of properties within the signal. Established references in the signal analysis community, such as Flandrin (1998) and Hlawatsch and Auger (2013). The second part shows the collaboration of these representations and associated models in a single pipeline for temporal classification, along with experimental comparisons on μ-D radar signals.

The raw signal, as a time series of phase and amplitude, doesn't reveal much information to the naked eye. It does however house a rather rich family of semantic features, potentially revealing information of key relevance about the signal. Building a hierarchy of layers to construct an even more compact and meaningful feature representation certainly would benefit to the subsequent classification task. Learning models adapted to this kind of representation include Recurrent Neural Networks (RNNs) and 1D or 2D Convolutional Neural Networks (CNNs).

The spectral representation obtained through the Fourier transform, on the other hand, yields a synthetic view of frequency information throughout the signal. Global trends in the signal's overall behaviour are easily spotted, which makes the spectral representation a more favourable candidate for representing the signal. Because it posseses the same structure as the raw signal, similar models can be used on the spectral representation.

However, a fully spectral feature space, by construction, drops all temporal information. More than the overall frequency information, it is perhaps in the fluctuation of such information that resides the most discriminative features, even more so in a context of challenging data with a wide and interlaced spectrum of behavioural activity, such as in Non-Cooperative Target Recognition (NCTR) on drone μ-D radar data. As such, a windowed Fourier transform seems to constitute the resolving compromise , by providing an effective time-frequency representation of the signal: there, local trends in frequency are seen evolving through time. See for instance Stoica and Moses (2005) and Moruzzis and Colin (1998) for a background review of learning models on μ-D data.

The two dimensions embodied by time and frequency add a layer of structure in the data: while Recurrent Neural Network (RNN) are still an option, Computer Vision (CV) literature strongly hints towards the usage of 2D CNNs. However, the locality found in the time-frequency "image" is not symmetric as the spatial

locality in actual images: time and frequency don't spawn the same space. While omitting this fact is entirely an option, taking it into account during architecture design could perhaps lead to more efficient models. Previously cited literature generally does not take this step; while CNNs and to a lesser RNNs exist for μ-D classification, the architectures are more often than not taken directly off the shelf.

As already stated in previous sections, a companion notion to the Fourier transform is that of covariance; like the Fast Fourier Transform (FFT), it allows for a compact representation of trends within the signal. By examining the underlying Gaussian process, the Sample Covariance Matrix (SCM) resulting from the signal thus constitutes a highly-structured, promising representation to make use of in learning models. As was also mentioned before, its inherent Riemannian structure calls for carefully constrained design on the manifold of centered Gaussian distributions, *i.e.* the manifold of Symmetric Positive Definite (SPD) matrices $\mathcal{S}_*^+$. While some models have been presented in the previous chapter, we will rather rely on a Deep Neural Network (DNN)-like formulation of the problem, and base our works on a so-called SPD neural network (SPDNet), introduced by Huang and Van Gool (2017). To the best of our knowledge, such an SPD architecture has not yet been adapted to time-frequency data, as we will here do.

Before concluding the chapter's introduction, we would like to remind, that while the proposed models theoretically fit any structured time series data, the experimental validations are centered around μ-D radar data classification. As asserted in the introduction and Chapter 2, radar data is plagued by the cost of its acquisition, its inherently sensitive nature, along with the variety of different physical configurations. It thus and ends up a rather rare commodity. This particular characteristic, shared with many real-life applications (medical data being a prominent example), beg for solutions, not all found within the powerful machinery of DNNs. As previously intuited, Riemannian models fitting to the underlying data manifold hopefully allows for more efficient learning. Moreover, the simulation of data is a parallel track to attempt at filling the missing gaps needed to leverage the full expressiveness of deep models. In this context, we also propose a drone μ-D radar data simulator.

To summarize our proposed contributions:

- A Fully Temporal Convolutional Network (FTCN) designed to respect the temporal structure of the data;
- Two closely resembling complex declinations of the FTCN, notably:
  - → A $\mathbb{CR}$ Neural Network ($\mathbb{CR}$Net), with a design inspired from signal analysis principles;

→ A Fourier convolution layer, allowing the training from raw complex temporal data, yielding a Fourier Neural Network (FourierNet) architecture;

- A full pipeline for structured time series classification, combining multiple representations and models;
- A μ-D drone radar simulator to allow for extensive experimental studies;
- Validation, comparison and interpretation of the various models, on both simulated and real data.

The chapter begins in Section 3.2 by a description of the proposed models on the different representations, from the SPDNet on covariance matrices to the CNN architectures adapted to the time series data. Then, Section 3.3 introduces the proposed classification pipeline, the Second-Order Fully Temporal Network (SOFTNet) model. We end the chapter in Section 3.4 with the experimental validations, along with the description of our drone simulator.

3.2 Learning on structured time series representations

As we have seen, time series data can be seen from various viewpoints, all rooted in sound theoretical background, and different learning models may operate independently on the different representations. Since the experimental model validations rely on μ-D radar data, we will also generalize the framework to complex-valued data. We begin with the SPDNet on covariance matrices. We then introduce a FTCN deep architecture on real-valued time-frequency representations, *i.e.* spectrograms. Finally, we generalize the latter to the complex domain, and by doing so exhibit a formal link between CNN architectures operating on the time series on the one hand, and on a time-frequency representation on the other hand.

3.2.1 SPD neural networks

The SPDNet architecture mimics that of classical neural networks with a first stage devoted to compute a pertinent representation of the input data points and a second stage which allows to perform the final classification. The particular structure of $\mathcal{S}_*^+$, the manifold of SPD matrices, is taken into account by layers crafted

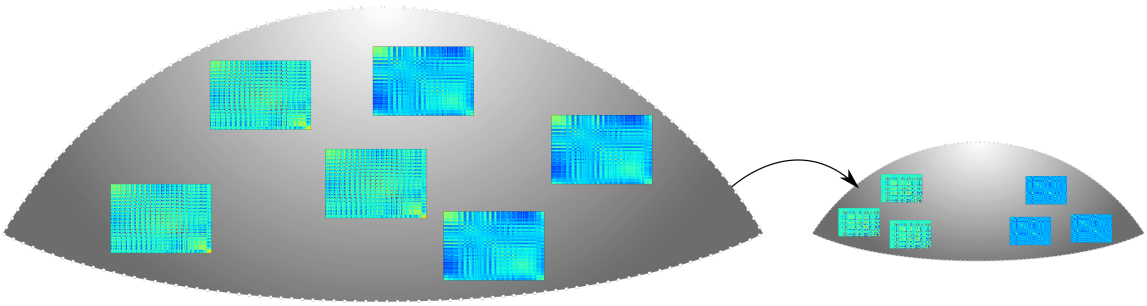

Figure 3.1.: Illustration of the BiMap layer
The input manifold is transformed to a lower-dimensional, more discriminative one.

to respect and exploit this geometry. Layers introduced in Huang and Van Gool (2017) are recalled in the following paragraphs.

Bilinear Mapping layer In analogy to the classical linear layer, the Bilinear Mapping (BiMap) layer transforms a SPD matrix P (of size n) at layer $(k-1)$ to a SPD matrix X (of size n') at layer (k), which we wish to be more compact and discriminative in terms of the final loss function, via a basis matrix W of $\mathcal{G}l(n, n')$ (the group of left-invertible matrices of size $n \times n'$) as illustrated in 3.1. Note that $n' \leq n$ is required to guarantee W's full-rank, thus the positive definiteness of the output matrix. However, $\mathcal{G}l(n, n')$ being a group of matrices unbounded in norm, optimizing on it constitutes a threat of gradient divergence. Thus, as in Huang and Van Gool (2017), the transformation matrices are constrained to the compact manifold of semi-orthogonal matrices, also known as Stiefel manifold, noted $\mathcal{O}(n, n')$. Finally, the forward pass of the BiMap layer is expressed as follows:

$$X^{(k)} = W^{(k)^T} P^{(k-1)} W^{(k)} \text{ with } W^{(k)} \text{ semi-orthogonal} \tag{3.1}$$

Rectified Eigenvalues layer Inspired from the Rectified Linear Unit (ReLU) – perhaps the most popular activation in DNNs (B. Xu et al. 2015) – Huang and Van Gool (2017) introduce the Rectified Eigenvalues (ReEig) layer, which lifts small (as in close to zero) eigenvalues to a given threshold ϵ. The function being applied solely to the matrix' eigenvalues, we first need to eigen-decompose the matrix P as $P = U \Sigma U^T$, where the ortogonal matrix U groups P's eigenvectors in columns and Σ is the diagonal matrix of n eigenvalues. Then, the forward ReEig layer is expressed as follows:

$$X^{(k)} = U^{(k)} \max(\Sigma^{(k)}, \epsilon I_n) U^{(k)^T} \text{ , with } P^{(k)} = U^{(k)} \Sigma^{(k)} U^{(k)^T} \tag{3.2}$$

While the ReEig acts a layer activation, another viewpoint would deem it a regularization of the covariance's positive definiteness. This different perspective opens the door to considering other options for the activation, possibly inspired from the covariance estimation, or more generally the Random Matrix Theory (RMT) community. While not the subject here, interesting ideas improving the notion of matrix regularization can be found in Ledoit and Wolf (2004), Ledoit and Wolf (2012), Y. Chen et al. (2011), Tiomoko et al. (2019a), Tiomoko et al. (2019b), Cao and Bouman (2009), Cao et al. (2011), and Mezzadri (2006).

Log Eigenvalues layer The first stage of a SPD neural network consists in a stack of manifold-to-manifold BiMap+ReEig layers. Once we reach a satisfying feature manifold, we aim to perform classification on the corresponding matrix P. To do so, Huang and Van Gool (2017) propose to map P to a Euclidean space, where it is possible to resort to classical Machine Learning (ML). They do so by harnessing the work of Arsigny et al. (2006), in which the authors endow the SPD manifold with a Lie group structure, for which the associated Riemannian distance computed from infinitesimal integration across the associated Lie algebra, the Log-Euclidean Metric (LEM), first introduced in Pennec et al. (2006) and Harris (2004), reduces to the Euclidean distance between the matrix logarithms. This particularity reveals a natural mapping from the manifold of SPD matrices to the space of symmetric matrices via the matrix logarithm $\log$, which is chosen as the mapping; additionally, the resulting matrix is vectorized to an $n * n$-dimensional vector, an operation we note vec:

$$X^{(k)} = vec(\ U^{(k)} \log(\Sigma^{(k)}) U^{(k)^T}\ )\ \text{, with } P^{(k)} = U^{(k)} \Sigma^{(k)} U^{(k)^T} \tag{3.3}$$

We illustrate such an SPDNet architecture in Figure 3.2.

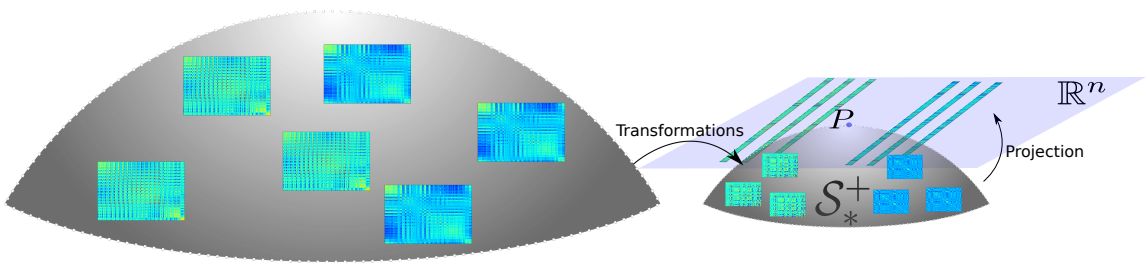

Figure 3.2.: Illustration of a generic SPD neural network
Successive bilinear layers followed by activations build a feature SPD manifold, which is then transformed to a Euclidean space to allow for classification.

These three layers sum up the inference phase of the original SPDNet. The specifics of the backpropagation will be discussed in the following chapter. The ardent reader may refer to Ionescu et al. (2015) and Edelman et al. (1998) for

the detailed mechanisms of an SPDNet; otherwise, the key concepts are reminded when needed, in Chapter 4.

3.2.2 Fully Temporal Convolutional Network

Here we present the DNN for structured temporal signals. We first justify the choice of developing a fully convolutional model, then detail the proposed architecture and comment on its properties, notably the preservation of temporal structure and temporal resolution.

3.2.2.1 All you need is convolutions

Interpreting a time-frequency representation as an image to be analyzed via a CNN constitutes a first attempt at deep modelling. One first subtlety to handle is that choosing the same representation for input signals of potentially different lengths will lead to frequency dilution, exposed in Figure 3.3. The more natural solution is to use the same projection instead of the same representation, *e.g.* a spectrogam of fixed window size and overlap. This of course allows input data of different dimensions, not easily handled by standard CNNs. A natural bypass is to consider the problem as Multiple Instance Learning (MIL). In this framework, one input data point is seen as a collection of multiple atomic instances of the same class. Concretely, we divide spectrogams in segments of equal length, which we denote as τ. The choice of τ is rather arbitrary, and is interpreted as the minimal duration one must observe the signal to accurately determine its class; its influence on results will thus be an interesting analysis. As an analogy to speech or music recognition's phonemes (Lee et al. 2009; Bartz et al. 2017; Dieleman and Schrauwen 2014; Parascandolo et al. 2016; Cakir et al. 2017; Hershey et al. 2017), we may call these atomic portions of signal "noisemes". We thereby consider that observing a noiseme-length signal portion is sufficient to classify the whole signal, but that observing multiple portions is expected to boost the classification confidence.

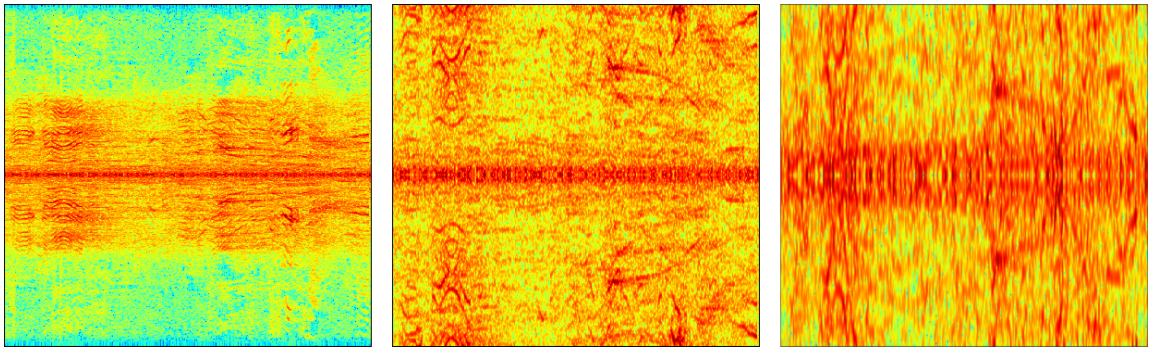

Figure 3.3.: **Downsampling of a spectro-temporal signal**. Here we choose the same input representation, *e.g.* a 512×1024 time-frequency spectrogram.

From left to right in Figure 3.3:

- the spectrogram of the original signal;
- that of the same signal downsampled four times;
- that of the signal downsampled sixteen times.

We observe how projecting to the same frequency domain signals of different intrinsic time length yields dilutions in frequency, which in turn unavoidably leads to bad classification.

Although convolutions do preserve temporal structure, the final layers of a typical CNN being fully-connected, that structure is lost, precisely when we reach the most representative feature spaces in the network. The conservation of the temporal structure requires the usage of a FTCN, described in the previous chapter.

3.2.2.2 Deep architecture for temporal classification

The desirable consequence of being fully convolutional is that, having transformed the time series to a deep feature representation through the network's layers, these learnt features can also be modelled and analysed as a final temporal representation of the signal; for instance, it now becomes possible to study their covariance matrix, which will constitute a cornerstone for our future classification pipeline.

The resulting architecture, which as mentioned above focuses on μ-D radar data, is illustrated in Figure 3.4. Such a network, which we may name FTCN, naturally allows input signals of varying length, and outputs temporal feature maps when fed with longer signals.

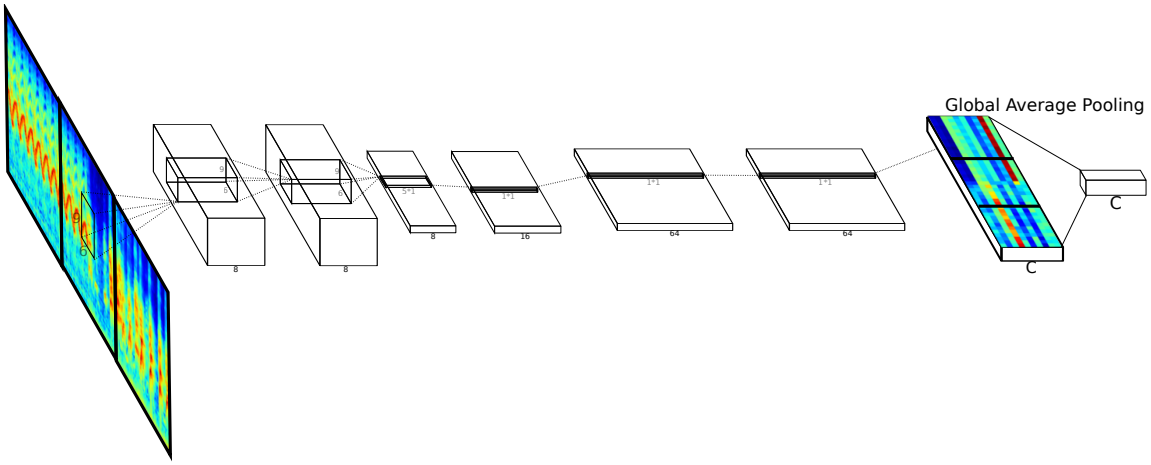

Figure 3.4.: **The proposed architecture for radar signal classification**. The boxes represent the successive filter banks. Horizontal is time, vertical is frequency. Contrary to CV, where the feature maps are spatial, our feature maps extend solely in time; the frequency domain is flattened through the convolutions.

The choice of hyperparameters for the network directly influences the inherent temporal precision achieved by the FTCN, in particular the filter and pooling sizes and associated strides. Figure 3.5 gives a reminder of how strided filters work, along with a quick intuition to the concept of receptive field. Since the FTCN is built upon the CNN classifier, the temporal feature map length l_o equals 1 when given an input of length $l_i = \tau$. We note $\mathcal{S}(l_i) := l_o$ the global sizing function of the network, composed of L layer-wise such functions $\mathcal{S}_l$ as defined in Figure 3.5. The values taken by $\mathcal{S}$ for multiples of τ define the temporal finesse of the network, obtained by successive receptive field overlaps. For instance, no receptive overlap, achieved through non-overlapping strided filters, gives $\forall k \in \mathbb{N}^*,\ \mathcal{S}(k\tau) = k$. In the case of the proposed architecture, (Figure 3.4), we have:

$$\forall k \in \mathbb{N}^*,\ \mathcal{S}(k\tau) = \mathcal{S}_L \circ \cdots \circ \mathcal{S}_1(k\tau) = \lfloor \frac{5}{2}k\tau - \frac{5}{2} \rfloor + 1 := \lfloor (k-1)\mathfrak{f}\tau \rfloor + 1 \tag{3.4}$$

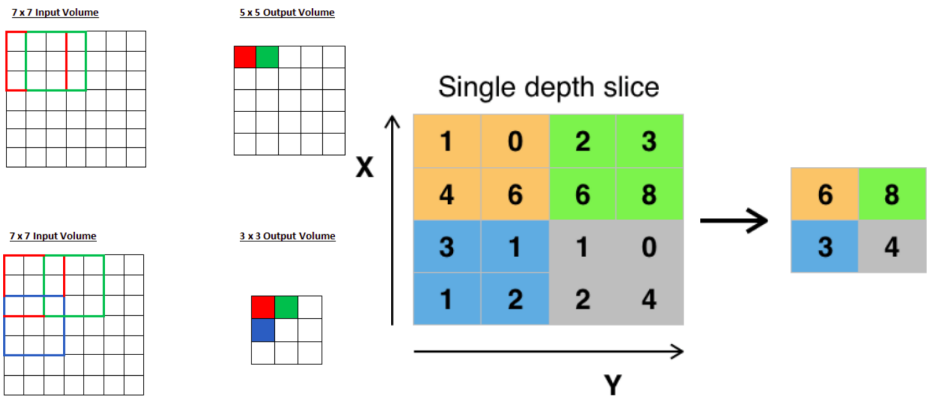

Figure 3.5.: **Illustration of how strided filters of size k and stride s work on an input of size n.** Image source: Deshpande (2017).

We introduced in the above equation the finesse coefficient $\mathfrak{f}$ which quantifies the super-resolution reached within a noiseme. By construction, $\mathfrak{f} \in [1, \tau]$, and its explicit formula is simply $\mathfrak{f} = (\prod_{l=1}^{L} s_l)^{-1}$ (s_l being the stride at layer (k)). For instance with $\tau = 20$ and $\mathfrak{f} = \frac{5}{2}$, we can classify the signal every 8 timesteps instead of 20, *i.e.* perform a segmentation of controled precision. The takeaway message here is that reducing the amount of pooling and strided filters leads to better temporal resolution. A trade-off thus arises, as strided filters are usually seen as a way for the network to better generalize.

In Figure 3.5:

- To the left:
  - Above: Standard convolution with $k = 3$ and $n = 7$ with no stride. The output is of size $n' = n - k + 1$.
  - Below: Strided convolution with $k = 3$, $s = 2$ and $n = 7$. The output is of size $n' = \frac{n-k}{s} + 1$.
- To the right: the same concept applies for pooling: in this example we have $k = 2 = s$.

In addition to stride, padding is often used to constrain the output to the same size as the input. In general, the sizing function from layer (k) to layer $(k+1)$ then reads $S_l(n) := n_{l+1} = \left\lfloor \frac{n_l - k_l + 2p_l}{s_l} \right\rfloor + 1$. The notion of receptive field arises from how many pixels in the previous image the next one "sees". In general, CNNs are designed so that the final receptive field more or less covers the input area.

We conclude this first model description by insisting that, while the usage of CNNs is not unheard of in the time-frequency classification literature, a fully-

convolutional architecture, designed to fit the structure of data as detailed above, is novel. The next paragraphs generalize the described model family to the complex domain.

3.2.3 Complex Fully Temporal Convolutional Network

We begin this generalization with a reminder of complex calculus. In the context of DNNs, the backpropagation involves defining gradients of complex-valued functions, which are non-holomorphic, *i.e.* which requires itself a generalization from the usual framework.

3.2.3.1 Non-holomorphic complex calculus

It is at first tempting to handle complex numbers in CNNs by simply considering a 2-channeled input containing the real and imaginary parts. One should however take care of respecting the inherent structure of complex numbers; both channels are neither independent nor interchangeable. In this section we describe how complex numbers can indeed be handled in a 2-channel fashion given certain constraints and interactions.

First, we establish the formal equivalence between complex numbers and 2D real vectors as developed by Wirtinger (1927) and rediscovered by Brandwood (1983), and later on by Bos (1994). This rather old framework is named Wirtinger calculus, or $\mathbb{CR}$ calculus (Kreutz-Delgado 2009). The context of these original developments was the generalization of the complex gradient to non-holomorphic complex functions, to which the proper conceptualisation of complex differentiability is usually limited to. As such, the following equations also establish the complex gradient operators for non-holomorphic functions, which in turn may be used in the backpropagation phase of subsequent neural networks.

The key idea equates the Taylor expansions of a function $f : z \in \mathbb{C} \mapsto y \in \mathbb{R}$ with its counterpart, which by abuse of notation is also noted $f : (u, v)^T \in \mathbb{R}^2 \mapsto y \in \mathbb{R}$; both functions map to the same scalar y. The Taylor expansion for both forms is written:

$$f(z) \approx f(z_0) + \nabla_z f_{z_0}(z - z0) \tag{3.5}$$

$$f(u, v) \approx f(u_0, v_0) + \begin{bmatrix} \nabla_u f & \nabla_v f \end{bmatrix}_{(u_0, v_0)} \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} \tag{3.6}$$

We now introduce the 2×2 real-to-complex matrix T:

$$T := \begin{bmatrix} 1 & j \\ 1 & -j \end{bmatrix} \text{ such that: } \begin{cases} T^H T = 2I = T T^H \\ T \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} u + jv \\ u - jv \end{bmatrix} = \begin{bmatrix} x \\ x^* \end{bmatrix} := \underline{\mathrm{x}} \end{cases} \tag{3.7}$$

In the equation above, $\cdot^*$ and $\cdot^H$ denote the complex conjugate and transconjugate. We can now rewrite the first-order moments of the vector function f as a function on complex values:

$$\begin{aligned} \begin{bmatrix} \nabla_u f & \nabla_v f \end{bmatrix}_{(u_0,v_0)} \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} &= \left(\frac{1}{2} \begin{bmatrix} \nabla_u f & \nabla_v f \end{bmatrix}_{(u_0,v_0)} T^H \right) \left(T \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} \right) \\ &:= \left(\nabla_{\underline{\mathrm{x}}} f_{\underline{\mathrm{x}}_0} \right) \left(\underline{\mathrm{x}} - \underline{\mathrm{x}}_0 \right) \end{aligned} \tag{3.8}$$

In the equation above, we defined the complex gradient:

$$\begin{aligned} \nabla_{\underline{\mathrm{x}}} f_{\underline{\mathrm{x}}_0} &= \begin{bmatrix} \nabla_x f_{x_0} & \nabla_{x^*} f_{x_0} \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{2}(\nabla_u f - j \nabla_v f)_{(u_0,v_0)} & \frac{1}{2}(\nabla_u f + j \nabla_v f)_{(u_0,v_0)} \end{bmatrix} \end{aligned} \tag{3.9}$$

The complex gradient is composed of the complex differential operator ∇_x and the complex conjugate differential operator ∇_{x^*}, which can then be inserted in the backpropagation framework such as *HIPS/autograd* (2020) for any neural network using the formal representation of complex numbers as 2D vectors. In the simple case of a convex optimization problem (Boyd and Vandenberghe 2004), we see that equating the complex vector gradient to zero is equivalent to setting either component to zero. By chosing for instance the second component, and using the equivalent 2D vector form, we arrive to a simpler optimization problem:

$$\begin{aligned} &\frac{1}{2}(\nabla_u f + j \nabla_v f)_{(u_0,v_0)} = 0 \text{ (complex domain)} \\ \Leftrightarrow &\begin{bmatrix} (\nabla_u f)_{u_0} & (\nabla_v f)_{v_0} \end{bmatrix} = 0 \text{ (real domain)} \end{aligned} \tag{3.10}$$

Similarly for gradient descent, we chose $(\nabla_u f + j \nabla_v f)$ as the gradient to propagate backwards, as done in Brandwood (1983) and Böddeker et al. (2017). Thus, in practice, by using the $\mathbb{CR}$ calculus framework, it suffices to represent complex numbers as 2-channeled inputs, design the inference layers with correct interactions between the real and complex channels, and simply compute the gradient as a 2-channeled vector to use as is during backpropagation. We refer to Trabelsi et al. (2018) for a description of the usual CNN layers translated to the complex domain, which we in turn implement in our proposed model. The latter builds upon the real-valued architecture introduced in Figure 3.4; while the $\mathbb{CR}$ framework allows

for a purely complex network, we rather implement a semi-complex network, *i.e.* the final layers stay in the real domain. The reasons for this are mainly empirical, and detailed in the dedicated Section 3.4. The transition from real to complex values done through the decibel (dB) function, as suggested from signal analysis standards:

$$\forall z \in \mathbb{C},\ dB(z) = 10\log_{10}(|z|^2) \in \mathbb{R} \tag{3.11}$$

The choice of the dB function, though guided by common practice, also empirically proves useful in experiments as shown later on. See Figure 3.6 for a description of the semi-complex model.

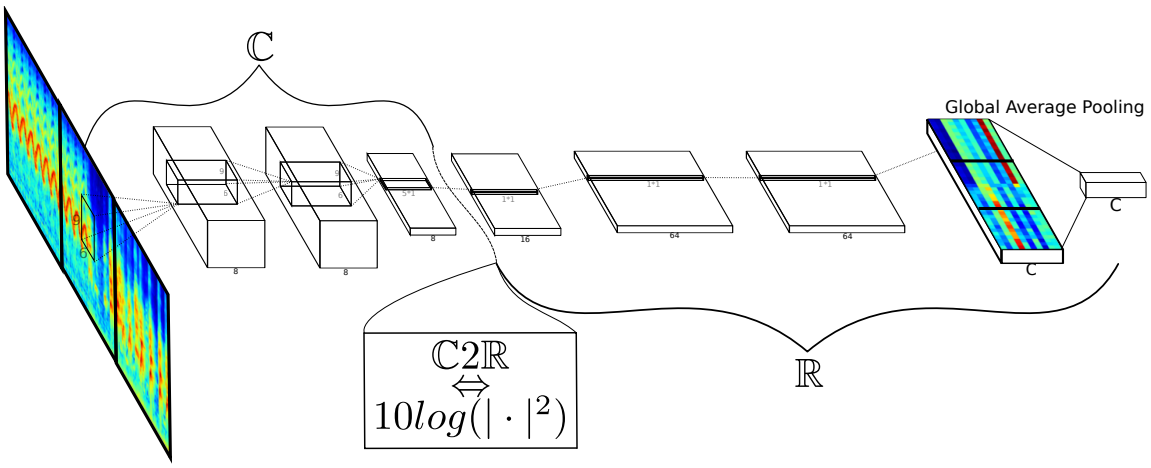

Figure 3.6.: **The proposed complex-valued architecture for radar signal classification**. All layers are complex, except for the final ones. The decibel function transitions between the two domains.

This complex architecture adapts closer yet to the μ-D data structure: by preserving phase information further on than the spectrogam, it allows a more powerful and flexible deep model for classification. While we credit Trabelsi et al. (2018) for founding elements of deep complex networks, the architectural exploration which is detailed further on, along with the improvement of the complex Batch Normalization (BatchNorm) layer, in the context of truly appropriate data types (*i.e.*, inherently complex-valued), remains a novel orientation of the matter.

3.2.4 HPD neural network

In a similar trend to the complex-valued FTCN, and following generalizations of the Minimum Riemannian Distance to Riemannian Mean (MRDRM) scheme to the

complex domain (Barbaresco 2013), we develop a HPD neural network (HPDNet), designed through the same principles of Wirtinger calculus. In this section, we present how we adapt the SPDNet layers to complex values, making use of the previously used formal two-channel representation. We note $\mathcal{H}_*^+$ the manifold of Hermitian Positive Definite (HPD) matrices.

3.2.4.1 Formal HPD representation in a real-valued computational framework

Because calculus of non-holomorphic functions is to this day not widespread in computational frameworks (for instance as of end 2018, complex tensors are not integrated in the deep learning framework PyTorch Paszke et al. 2017), it remains interesting to provide a custom integration, especially given the simplicity of the resulting implementation. From the results derived above, we can thus represent an HPD matrix $H = H_\Re + jH_\Im$ as a two-channel SPD matrix $(H_\Re, H_\Im)$. In any neural architecture, the gradient of the loss function l involving $(H_\Re, H_\Im)$ is computed as $\nabla_{H_\Re} l + j\nabla_{H_\Im} l$, which corresponds exactly to $\nabla_H l$. For this reason, it is natural to choose $\nabla_H l$ over $\nabla_{H^*} l$, as foreseen above. The backpropagation through the network can therefore be done with no additional pain, using any out-of-the-box backpropagation algorithm in a real-valued computational framework. Similar to before, the inference still needs to respect the internal structure of complex numbers. Below we show how to adapt the BiMap layer to $\mathcal{H}_*^+$.

3.2.4.2 Complex bilinear mapping

Here we show how to generalise the BiMap layer described previously to complex values, using the Wirtinger formalism. First we write the complex expresison of the bilinear mapping, then structure it to fit the $\mathbb{CR}$ framework:

$$\begin{aligned}
P &= WXW^H \\
&= (W_\Re + jW_\Im)(X_\Re + jX_\Im)(W_\Re^H + jW_\Im^H) \\
&= \left(W_\Re X_\Re W_\Re^H - W_\Im X_\Im W_\Re^H - W_\Re X_\Im W_\Im^H - W_\Im X_\Re W_\Im^H\right) \\
&\quad + j\left(W_\Re X_\Re W_\Im^H - W_\Im X_\Im W_\Im^H + W_\Re X_\Im W_\Re^H + W_\Im X_\Re W_\Re^H\right) \\
&:= \begin{bmatrix} W_\Re X_\Re W_\Re^H - W_\Im X_\Im W_\Re^H - W_\Re X_\Im W_\Im^H - W_\Im X_\Re W_\Im^H \\ W_\Re X_\Re W_\Im^H - W_\Im X_\Im W_\Im^H + W_\Re X_\Im W_\Re^H + W_\Im X_\Re W_\Re^H \end{bmatrix}
\end{aligned} \tag{3.12}$$

Note that the parameter matrix is now unitary (that is, complex orthogonal), and the transpose becomes a transconjugate.

3.2.4.3 Complex structured non-linearities

We study here the case of the non-linear structured complex functions involved in the HPDNet, the complex ReEig and complex Log Eigenvalues (LogEig). As stated previously, both take the form a non-linear function f acting on the an HPD matrix P's eigenvalues. We assume an eigen-decomposition of $P = U\Sigma U^H$. Note that although U is unitary, Σ is real because P is Hermitian. This is a very strong result, which greatly simplifies the generalisation of aforementioned functions to the complex setting: in fact, there is nothing in f to actually generalise, since the input eigenvalues are already real. However, one must take care to correctly handle the eigen-decomposition itself, separating the real and imaginary parts in order to respect the $\mathbb{CR}$ formalism:

$$\begin{aligned}
X &= f(P) \\
&= Uf(\Sigma)U^H \\
&= (U_\Re + jU_\Im)f(\Sigma)(U_\Re - jU_\Im)^T \\
&= \left(U_\Re f(\Sigma)U_\Re^T - U_\Im f(\Sigma)U_\Im^T\right) \\
&\quad + j\left(U_\Re f(\Sigma)U_\Im^T + U_\Im f(\Sigma)U_\Re^T\right) \\
&:= \left[U_\Re f(\Sigma)U_\Re^T - U_\Im f(\Sigma)U_\Im^T U_\Re f(\Sigma)U_\Im^T + U_\Im f(\Sigma)U_\Re^T\right]
\end{aligned} \tag{3.13}$$

Using the equations above, we are now able to perform inference through the complex BiMap, ReEig and LogEig layers. Posterior to the LogEig, a fully-connected layer (or several) handles the hyperplanar separation for classification. At any point in the network, it is possible to go back to $\mathbb{R}$ using the $\mathbb{C}2\mathbb{R}$ transfer function below:

$$\forall H \in \mathcal{H}_*^+,\ \frac{1}{2}(H_\Re + H_\Im) = S \in \mathcal{S}_*^+. \tag{3.14}$$

This is necessary because, again, we cannot yet perform the classification in the complex manifold. Furthermore, it is not obvious that the best performing model would maximize the use of complex numbers; it remains of interest to study the influence of the $\mathbb{C}2\mathbb{R}$'s positioning in the network's hierarchy to optimize its performance andd robustness.

3.3 Full pipeline for temporal classification

To sum up, we have presented various models adapted to different representations of the time series data, specifically the real-valued spectrogam, the complex spectrogram and the covariance matrix. The cunning reader may have noticed we have not presented a model for the raw time series data, notwithstanding the fact we had hinted in previous sections it was possible either with 1D CNNs or RNNs. Before moving on to the full pipeline, we first present a simple trick which allows to partly elude this matter, and naturally integrate the 1D raw time series within the models shown above.

3.3.1 The Fourier convolution layer

First we recall the Discrete Fourier Transform (DFT) of a discrete complex signal z of total duration N:

$$\forall k \leq N,\ DFT_z(k) = \sum_{l \leq N} z(l) e^{-2i\pi k \frac{l}{N}} \tag{3.15}$$

Thus, a windowed Fourier transform of length n can be replaced by a convolution of the signal by the n Fourier atoms, which are the n base vectors of n-th-roots of the unit, $(e^{-2i\pi k \frac{\cdot}{n}})_{k \leq n}$. Attempting to learn filter banks on 1D inputs has been used in the context of audio recognition (Lee et al. 2009; Dieleman and Schrauwen 2014); however, while they observed the learned filters did seem to converge to frequency-selective filters, the Fourier atoms were not used. Thus, instead of fixing the DFT, we propose to prepend to the FTCN presented in Figure 3.6 a 1D complex convolutional layer initialized with the Fourier atoms, and allow the weights to be learnable, *i.e.* we allow Fourier atoms to vary to best accomodate the optimization crtiterion. By doing so, we allow additional flexibility in the learning process, all the while utilizing the expert knowledge of a proper time-frequency representation. We illustrate this novel convolutional layer in Figure 3.7.

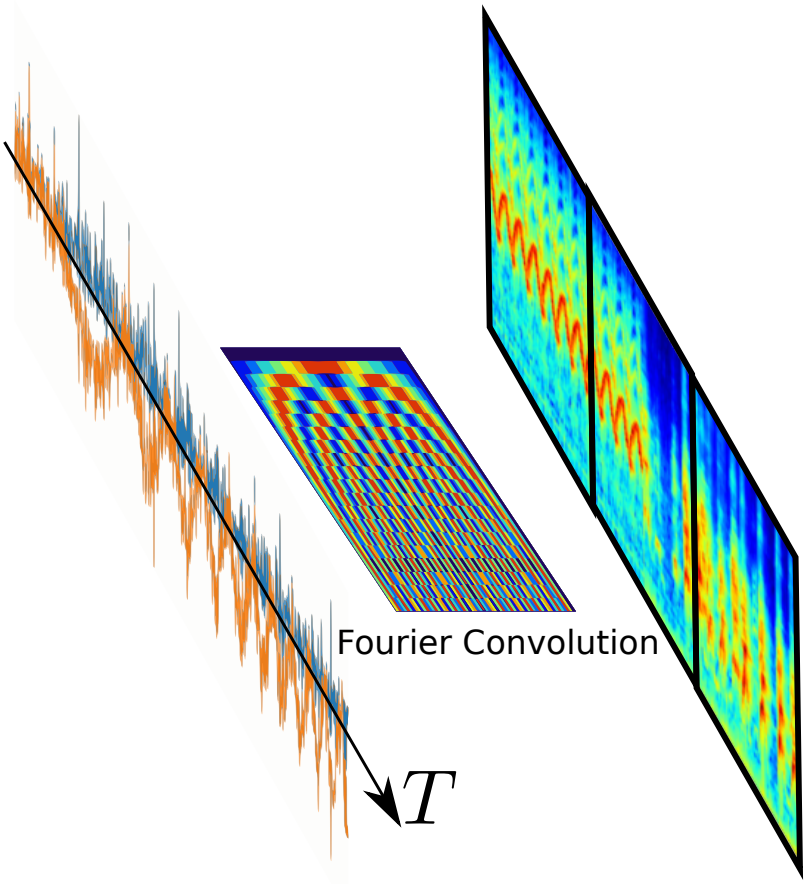

Figure 3.7.: **Illustration of the proposed Fourier-like convolutional layer**. The Fourier atoms are represented as sine waves of increasing frequency.

3.3.2 Pipeline bifurcations

We now introduce the proposed pipeline, displayed in Figure 3.8, and show how branching through the its blocks leads to different models on the signal representations. Four global models, noted from (1) to (4) in the figure, can be extracted from the pipeline, which we detail throughout the section.

As a first note, never bifurcating to covariance analysis, *i.e.* remaining on the first row in Figure 3.8 exactly amounts to the FTCN taking as input the raw signal, sequentially building a spectral and hierarchical feature representations. If the DFT is allowed to be fine-tuned, we call such an architecture a filter-bank learning network, or FourierNet; if not, we simply refer to it as a $\mathbb{CR}$Net. While the FourierNet thus consitutes a generalization of the $\mathbb{CR}$Net, both correspond to model (4) in Figure 3.8; the distinction is made to allow to see potential benefits from learning upon the Fourier atoms.

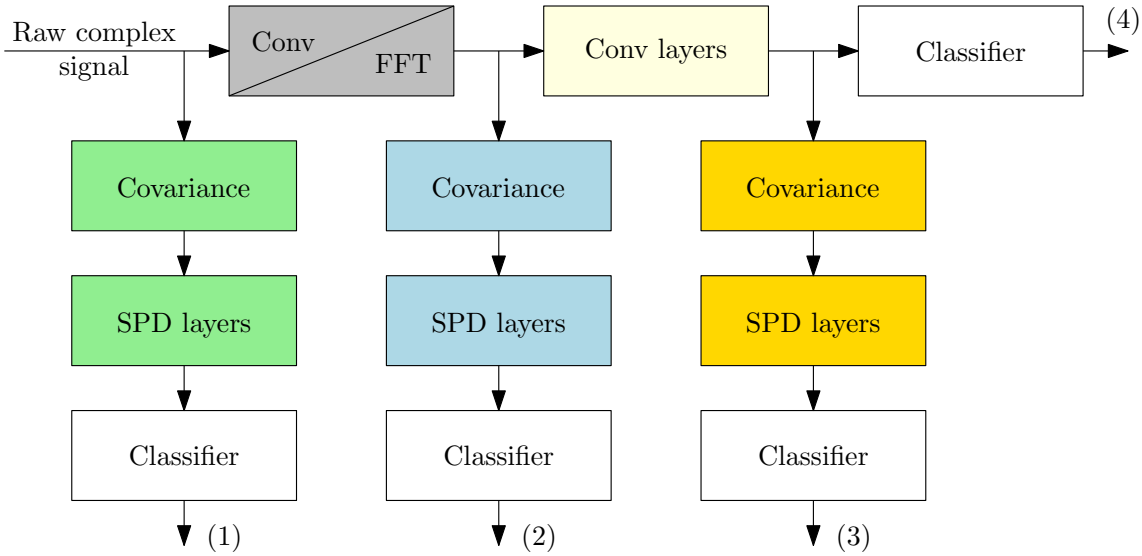

Figure 3.8.: **Illustration of the global pipeline proposed for the classification of time-frequency signals**. Possible bifurcations to covariance analysis stem either from the raw data, the spectral representation or the learnt temporal features. The first convolutional layer can be either learnt fom the data and the Fourier atoms or used as a Fourier transform.

Using a SPDNet to classify the covariance of the signal amounts to branching out downwards in the pipeline. This can be done either at the raw signal representation, or at the Fourier representation, whether it be a proper DFT or the Fourier convolution. In other words, it is possible to learn on the covariance of the raw temporal data, or on that of the spectral, Fourier trnasform of the latter. The two resulting SPDNet models correspond to (1) and (2) respectively.

The classification in the FTCN uses Global Average Pooling (GAP) to pool the feature maps' temporal evolution to a single dimension. However, as mentioned before it is possible to branch out in the pipeline at any stage, precisely thanks to the fully-temporal property of the network. It is precisely the property of conservation of temporal structure verified by the FTCN which allows to extract temporal feature maps, and thus, gives the possibility to study the covariance of these maps. Doing so hints towards a powerful representation learning model, making use both of a potentially complex-valued Euclidean FTCN operating on first-order moments of the data, and a Riemannian SPDNet operating on second-order moments of the data. We thus call this particular model the Second-Order Fully Temporal Network (SOFTNet), and illustrate it in Figure 3.9.

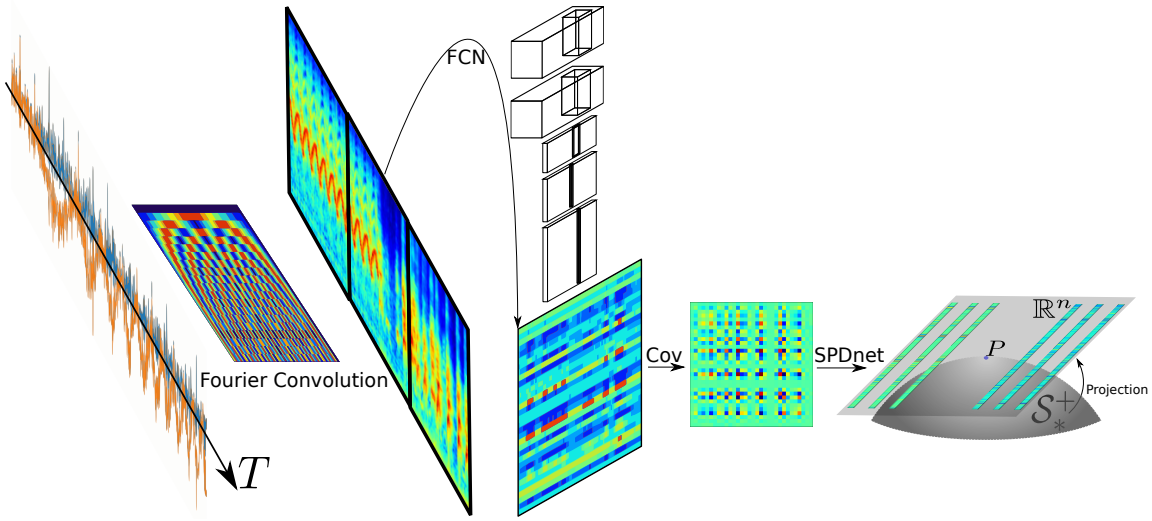

Figure 3.9.: **Illustration of the SOFTNet**. This model makes use of all cited representations for structured time series data: time, frequency and covariance.

3.3.3 Covariance pooling

A key module linking the first- and second-order models is the covariance pooling (CovPool): given an n-dimensional signal sampled during N timesteps $s = (s_i)_{i\in[1,N]}$, where $s_i \in \mathbb{R}^n$, its covariance matrix $X \in \mathcal{S}_*^+(n)$ is estimated as:

$$X = \sum_{i\in[1,N]} \bar{s}_i \bar{s}_i^T \tag{3.16}$$

In the equation above, often referred to as Maximum Likelihood (MLE), $\bar{s}_i$ is the centered version of s_i, *i.e.* $\bar{s}_i = s_i - \frac{1}{N}\sum_{i\in[1,N]} s_i$.

In our context, the raw signal is univariate as it is, by construction, a scalar complex time series. We overcome this problem by considering the raw signal as a series of possibly overlapping windowed elementary sub-signals, introducing effective multi-variability through the length τ of the window. Thus, one s_i in Equation 3.16 becomes $[x(l) \cdots x(l+\tau)]^T$. In essence, the windowing is similar to performing a windowed filtering, and can also be represented as a convolution, for which the weights are no longer the Fourier atoms but adequately placed zeros and ones. Figure 3.10 further on illustrates the signal splitting.

Another problem may arise from the MLE estimation: theoretically, for X to be regular, i.e. positive definite, it is required to have $n \geq N$, i.e. to observe more samples than their dimension. This can be problematic in estimating the covariance of the learnt features, *i.e.* when the s_i in Equation 3.16 are the FTCN's final fea-

ture maps f_i. In this scenario, the temporal sampling has been reduced through the FTCN's convolutions and possible poolings, while feature size could have been set to arbitrarily large dimensions. The intent reader may refer to *DeepKSPD* (2018), Acharya et al. (2018), and Yu and Salzmann (2017) for details on how to best extrapolate covariance from CNN features: in summary, it is possible to use robust estimators such as described in Ledoit and Wolf (2004) and Ledoit and Wolf (2012), to split the feature maps in smaller groups, or even to introduce an transitional convolutional layer to learn a dimension reduction suited for the covariance pooling. In the experimental validation section below, several combinations of the above were tried; without going into much detail, we give examples of theoretically better conditioned covariance estimators used, but not cited, in the experiments:

$$\begin{aligned} \Sigma_{\text{regul}} &= (1-\lambda)\hat{\Sigma} + \lambda \frac{Tr(\hat{\Sigma})}{n} I_d \\ \Sigma_{\text{adjust}} &= U\left(\frac{1}{\alpha}\hat{\Lambda} + (\frac{1-2\alpha}{2\alpha})^2\right)U^T + \frac{1-\alpha}{2\alpha}\mathbb{1} \end{aligned} \tag{3.17}$$

In the two equations above $\hat{\Sigma}$ is the SCM of dimension n, and $\hat{\Lambda}$ its eigenvalues; Σ_{regul} tempers the covariance eigenvalues with a constant value $\lambda = 1e^{-6}$ (typical value); Σ_{adjust} fits eigenvalues to a parametric oracle constraint imposing regularity, with a typical value $\alpha = 0.75$ and $\mathbb{1}$ a $n \times n$ matrix of ones. See the established Scikit-learn package (Pedregosa et al. 2011), and more precisely the documentation for covariance estimation (*2.6. Covariance estimation — scikit-learn 0.22.1 documentation* 2020).

3.4 Experimental validation

As stated above, we perform experiments on μ-D radar data, specifically on the task of drone recognition. We will compare the different models performing on the different kind of input representations (raw signal, spectrum, covariance), and the full pipeline making use of all the latter. Before diving head-on in tables and numbers, one must take special care to ensure fair comparisons between the models; indeed, how does comparing the performances, for example, of a FTCN on spectrograms against a SPDNet on covariance matrices? This concern is solved by considering the input data processing - signal splitting, Fourier transform or convolution, covariance pooling - as particular instances of filtering; conceptually, it suffices to chose the same elementary duration τ for all processings to make all

comparisons fair. This is illustrated in Figure 3.10: as can be seen, the formal analysis of the full signal through different representations is made possible by chosing a temporal integration framework coherent throughout all representations. This concept refines Figure 1.4 shown in the introduction, where the Fourier transform and covariance pooling were performed on the signal's global time scale.

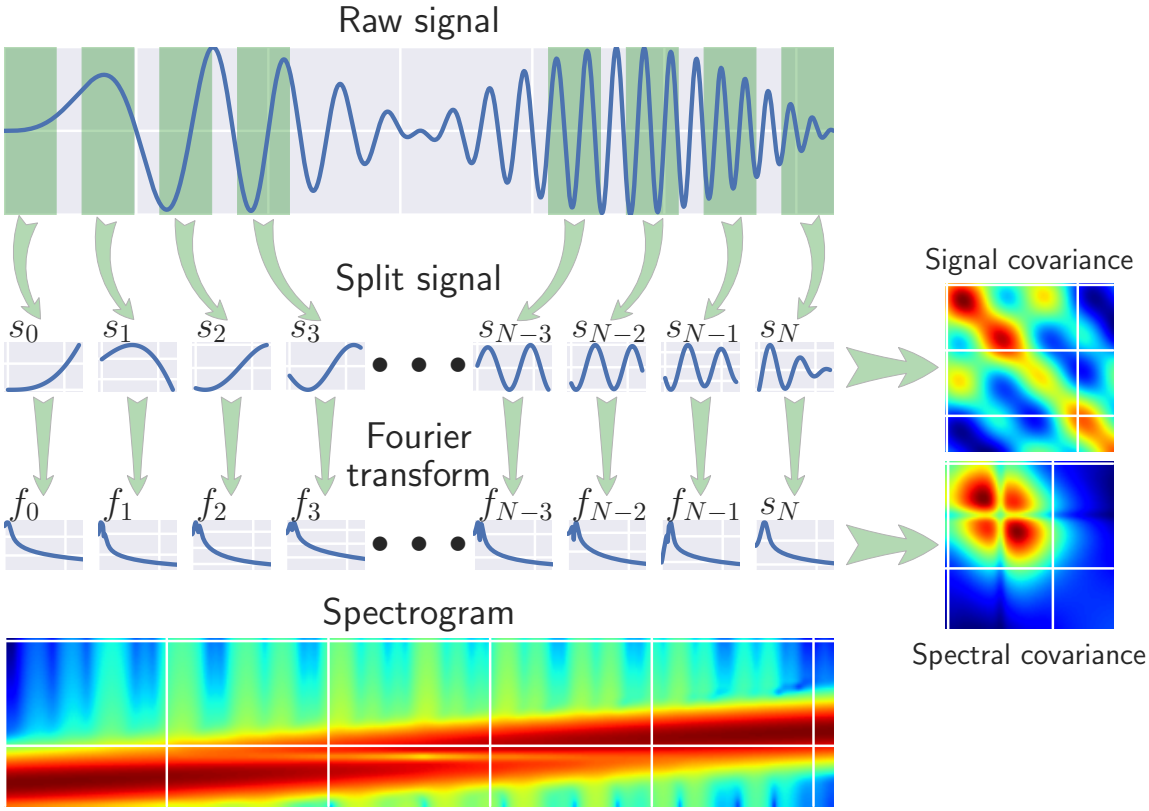

Figure 3.10.: **Structured time series representations**. The input raw signal is split in sub-signals of elementary duration τ. From there, either Fourier transform or covariance pooling then operate at the same time scale, resulting in output spectrograms and covariance matrices representing the same object, and fit to be compared against in further processings: FTCN, SPDNet, and eventually SOFTNet.

This founding issue being now addressed, we now describe in detail the bulk of the data used in the experiments. Specifically, we propose a data generator, which simulates the behaviour and physical properties of various drones, and then emulates a radar wave forming upon the latter targets. We now describe our proposed drone data simulator.

3.4.1 Drone micro-Doppler radar data simulator

Radars are a sensitive field for companies and states as they are usually involved in defense and security. For our problem, this amounts to a lack of real data and reproducibility. Moreover, the proliferation of Unmanned Aircraft Vehicle (UAV) models begs for generic modelling. Building an accurate and powerful simulator is thus key the task at hand, especially for deep learning models, which traditionnally require massive amounts of data, or rather close-to-exhaustive variety in the dataset. Here, we introduce a simple yet expressive drone simulator.

3.4.1.1 A physical drone model

Firstly, the UAV is modelled by a discrete set of N_p scattering points disseminated along its physical structure. Scaterring points model the reflectance of the target, as a discrete set instead of a continuous ensemble. Figure 3.11 shows the distribution of the scattering points, their reflection direction and Radar Cross-Section (RCS) along with their temporal evolution. Note the model is 2D as we consider the UAV to remain roughly in the same plane *w.r.t.* the radar. This assumption is fairly reasonable in many situations, greatly simplifies the simulation, and slight changes in inclination can be rather accurately modelled by variations in the RCS.

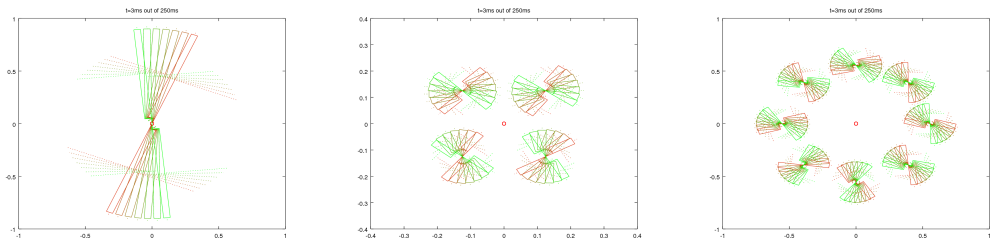

Figure 3.11.: **Evolution of the scattering points and normals for the Vario, Phantom2 and S1000+ drones**. The total simulation time is 250ms, of which we only illustrate the first 3ms (corresponding to 7 frames) for the sake of visual simplicity. The body is modelled as an isotropic reflector whereas blades are scattered with directional punctual reflectors in the direction of the normals, the length of which are proportional to the points' reflectance. Notice the multicopters' helices alternatively turn clockwise and counter-clockwise. The 2D scale is in meters; dimensions and distances are accurately scaled *w.r.t.* reality. Time goes from red to green.

The N_p scattering points moving in time yield a set of N_p series of 2D coordinates, which are then fed to wave equations described below. Equation 3.20 shows the final result of raw μ-D radar signal of the body and the helices, respectively S_0 and S.

The azimuth $\vec{Az} \in \mathbb{R}^2$ of the target *w.r.t.* the radar varies in function of time and the cuurent Rounds per Minute (RPM):

$$\vec{Az}_k(t) = -2\pi \frac{RPM_k(t)}{60} t + \vec{Az}_k(0) \tag{3.18}$$

The total angular vector $\vec{\alpha} \in \mathbb{R}^2$ modulating the final signal depends on the azimuth of the target and the orientation $\vec{n} \in \mathbb{R}^2$ (we drop the time dependency for clarity):

$$\alpha_k = \vec{Az}_k - \vec{n}_k \frac{\pi}{180} \tag{3.19}$$

Finally, the received signal S_0 from the drone's body is modelled as a trigonometric function of a spatial dot product of the azimuth and the positional vector X_k of the target's body scattering points and its RCS, and similarly for the contribution S of the drone's blades' scattering points:

$$\begin{aligned} S_0 &= \sum_{k=1}^{N_{P_0}} \sqrt{RCS_k} exp\left(4i\pi \frac{f_e}{c}(\vec{X}_k \cdot \vec{Az}_k)\right) \\ S &= \sum_{k=1}^{N_P} \sqrt{RCS_k} max(cos(\vec{X}_k \cdot \vec{\alpha}_k), 0) sinc(2\frac{f_e}{c} L_k sin(\vec{X}_k \cdot \vec{\alpha}_k)) exp\left(4i\pi \frac{f_e}{c}(\vec{X}_k \cdot \alpha_k)\right) \end{aligned} \tag{3.20}$$

Finally, the signal is immersed in a noisy and cluttered environment. The noise is unavoidable white thermic noise from the sensing mechanism, the clutter sums up the influence the outside environment on the signal of interest; Billingsley's model was used to model ground clutter (Billingsley 2002). The following section extends the simulator to account for unpredictible behavior, thereby introducing intra-class variety.

3.4.1.2 Dataset generation

The simulator described above gives a deterministic output given one configuration. Real data are however subject to multiple variations even within one single measurement, whether it be responses to an unkown controller, alterations of the environment or the data acquisition itself, all in all to NCTR. These variations

will define the intra-class disparity and thus the inter-class separability. Table 3.1 introduces the varying parameters as well as ranges of variations for each drone.

Drone parameters	Vario	Phantom	S1000+
Maximal flight curvature $\kappa_{max}(10^{-4}m^{-1})$	$[0; 2.55]$	$[0; 18.5]$	$[0; 48.5]$
Velocity V $(m.s^{-1})$	$[-10; 10]$	$[-10; 10]$	$[-10; 10]$
Blade RPM $(tr.min^{-1})$	$[1445; 1655]$	$[3000; 7000]$	$[1500; 6500]$
Body's scattering point RCS (m^2)	0.1	0.1	0.1
Blade's scattering point RCS $(10^{-3}m^2)$	$\left\{\begin{matrix}258\\230\\0.796\end{matrix}\right\}$	$\left\{\begin{matrix}3.18\\1.79\\0.199\end{matrix}\right\}$	$\left\{\begin{matrix}11.4\\8.66\\0.796\end{matrix}\right\}$

Table 3.1.: **The parameters are found or estimated from drone specifications**. Their exact values and variations still need to be heuristically estimated in order to allow for more expressive sampling.

Maximal flight curvature is computed as being proportional to mass over carrying surface, *i.e.* $\kappa_{max} = a_\kappa \frac{m}{\pi * L_b^2}$ with $a_\kappa = 5e^{-8}$. A directional scattering point's RCS is computed as being proportional to the dimensions d of the surface it represents, *i.e.* $RCS = \frac{d^2}{\pi}$. Note we have listed velocity and not radial velocity. We consider the case where a drone alternates between straight line to circular trajectories, we thus have $V_r = \pm V sin(\kappa(t))$, where $\kappa(t)$ is the curvature evolving through time.

Figure 3.12 shows intra and inter-class variations for the three drones Vario, Phantom2 and S1000+ for a given set of representation parameters. The reader may refer to Table 3.2 for an exhaustive list of default values. Intra-class variety is achieved by sampling pairs of drone parameter values in the acceptable ranges set in Table 3.1 and linearly interpolating the resulting values in time. Therefore the parameters are defined in time as in Equation 3.21:

$$
\begin{aligned}
&V(t) = V_1 + (V_2 - V_1)\frac{t}{T} \text{ with } V_1, V_2 \sim \mathcal{U}(V_{min}, V_{max}) \\
&\Omega(t) = \Omega_1 + (\Omega_2 - \Omega_1)\frac{t}{T} \text{ with } \Omega_1, \Omega_2 \sim \mathcal{U}(\Omega_{min}, \Omega_{max}) \\
&\kappa(t) = \kappa_0 \sim \mathcal{U}(0, \kappa_{max}) \\
&V_r(t) = \mathfrak{s}V(t) \odot sin(\kappa(t)) \text{ , } \odot \text{ being the element-wise Hadamard product and } \mathfrak{s} \text{ a random spin.}
\end{aligned}
\tag{3.21}
$$

f_e (GHz)	Pulse Repetition Frequency (PRF) (kHz)	T (ms)	N_{fft}	w_s	p_{ov}
3	2.5967	250	64	20	0.5

Table 3.2.: **Simulation parameters and their default values**. w_s is the Fourier window size and p_{ov} the overlap percentage. The table is split in two, to the left are radar parameters, to the right representation parameters.

Important parameters of the representation in Figure 3.12 are:

- Signal-to-Noise Ratio (SNR): 40 dB
- PRF: 8 kHz
- Simulation time T: 250 ms
- N_{fft}: 64 points
- Fourier window size $w_s = 20$ points
- Window overlap percentage $p_{ov} = 50\%$

Again, we take into account the gain in he Fourier transform. Visible temporal variations are as accurate as possible, although exaggerated, *w.r.t.* reality. They exhibit discriminative behaviours which we hope to capture in learning algorithms. The reader may observe that the S1000+, under strong variations, bypasses the allowed bandwidth set by the PRF, which leads to the well known frequency folding phenomenon (Shannon C. E. 2013), or Doppler ambiguity, which in turn can hurt the representation's credibility as a robust one. Unfortunately, though we can set the parameter arbitrarily high in simulations (again, which we do here for the sake of visual clarity), real-world radar costs constrain the PRF to possibly sub-optimal thresholds.

We are now ready to present results upon a simulated database of μ-D radar signals.

3.4.2 Experiments and results

Experimental setup All datasets contain $N = 1000$ examples for each of the $C = 3$ drone classes presented in Figure 2.3, of which we reserve 25% for validation and another 25% for testing. All models are run in a 5-fold cross-validation. Each example simulates a recording of length $250ms$, *i.e.* 1000 sample points for a standard PRF of 4kHz. This amounts to a total of ≈ 4 minutes of data per class, *i.e.* 2 minutes of training data. We choose the noiseme length $\tau \approx 10ms$, which corre-

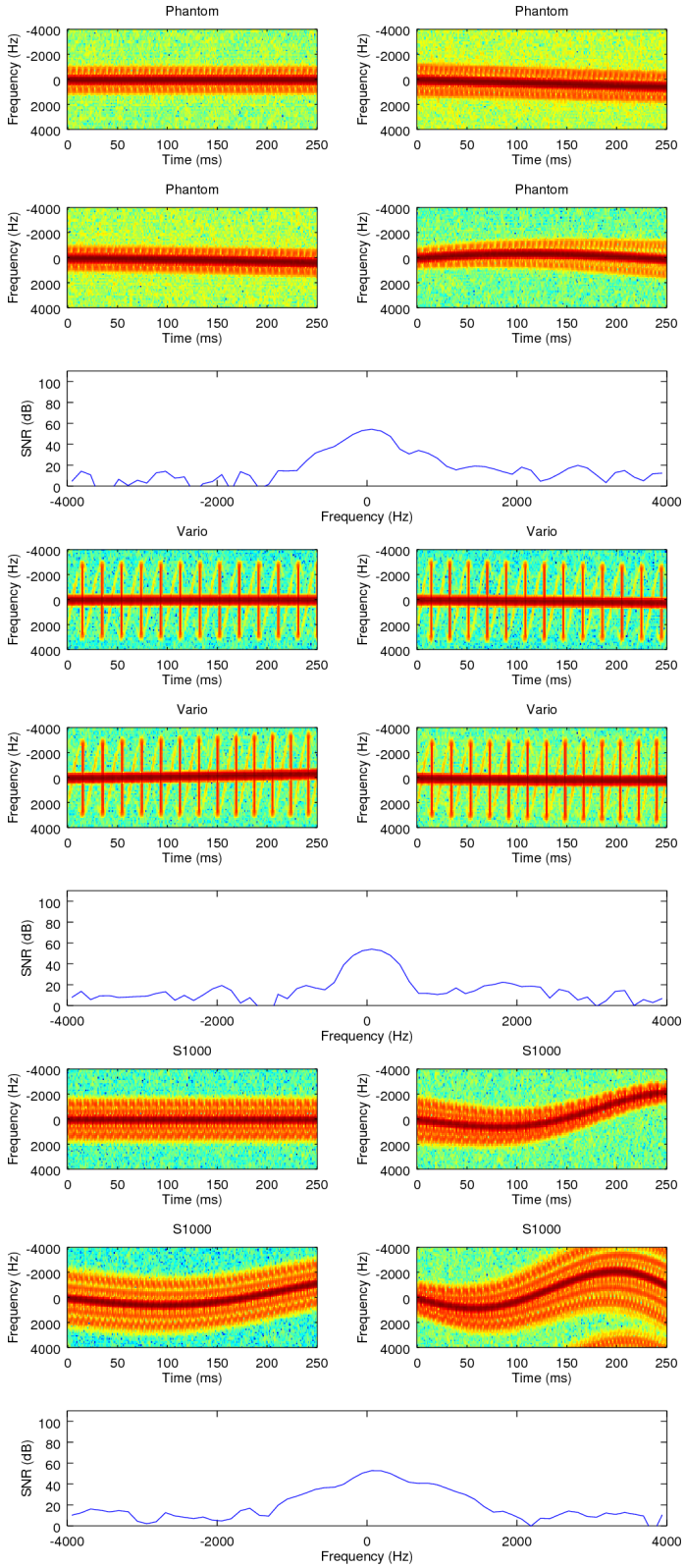

Figure 3.12.: **Spectrograms of noisy and uncluttered signals**. These are sampled from the three drones mentioned in Figure 2.3, from left to right: the Vario, the Phantom2 and the S1000+. Each group is organized as follows: four varying spectrogams of the same class are displayed; the top-left one always corresponds to a version constant in time. Below we plot an arbitrary time cut of one of the varying spectrograms.

sponds to 20 timesteps at 4Hz. The choice is guided by the largest approximate period in all signals, *i.e.* in the case of drones, approximately one blade rotation. Conceptually, this means the elementary decision resolution spans about one blade rotation.

3.4.2.1 Validation and comparison of the deep first-order models

We begin with the simplest proposed FTCN, *i.e.* operating on real-valued spectrograms obtained through a valid Fourier transform, and ratify its usage against two benchmarks: a logistic regression (*i.e.* a single-layered Multi Layer Perceptron (MLP)), and a Long Short-Term Memory (LSTM) architecture, which we will simply refer to as the RNN.

Throughout the experiments we train the three classifiers (MLP, RNN and FTCN) with the same strategy to remain consistent: Stochastic Gradient Descent (SGD) with initial learning rate $\alpha = 0.5$ for the MLP, RMSProp (Ruder 2016) with $\alpha = 2e - 3$ and decay $\beta = 1e - 6$ for the RNN, and accelerated SGD with $\alpha = 4e - 3$ and momentum $\mu = 0.9$ for the FTCN. All optimizations are initialized by Glorot uniform sampling and run for $K = 100$ epochs without early stopping nor cross-validation (except manual hyper-parameter finetuning). All learning rates are divided by 2 every 25 epochs. These initial models were implemented on both the Keras (Chollet 2015) library with Tensorflow (Abadi et al. 2016) backend and PyTorch (Paszke et al. 2017) deep learning framework, and trained on a single Nvidia GTX 1070M Graphical Processing Unit (GPU). Training time for any of the above lasts less than an hour.

General performance and robustness First we train the classifiers multiple times on a standard configuration to evaluate overall performance and robustness to initialization to evaluate the simulator's expressivity, seen in figure 3.13. Running the training 20 times for each classifier (we actually test three FTCN architectures with different finesse coefficients defined in Equation 3.4, *i.e.* different temporal resolutions), we observe consistent test accuracy results, the most consistent being the FTCN, which exhibits the smallest spread. In terms of accuracy, the MLP falls way behind the RNN and FTCNs, which are close though the FTCNs seem to generally perform better.

The FTCNs show better performance than the RNN and MLP, along with less variance over cross-validations.

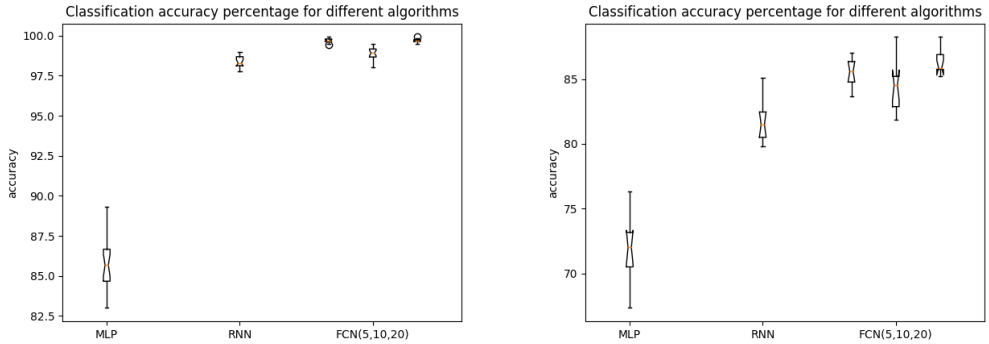

Figure 3.13.: **Classification accuracies for three learning models**. Here we chose a setup of cluttered, noisy signals with $SNR = 30dB$ on the left and $SNR = 10dB$ to the right ($PRF = 4kHz$), which is more than reasonable from a practicioner's point of view.

Impact of radar configuration parameters Here we train all models in different configurations; specifically we vary SNR and PRF from extremely challenging to wishfully accomodating. Results are found in figure 3.14. The most challenging configuration being at $SNR = 0dB$ and $PRF = 4kHz$, we nevertheless achieve $\approx 58\%$ with the FTCN, versus total confusion ($\approx 33\%$)) with the MLP. Note this is an unrealistically challenging configuration, as an SNR of $0dB$ means an original signal to noise ratio before Fourier transform of $\approx -15dB$. Performances seem to plateau when reaching high PRF, which is sufficient from a practicioner's point of view. This could potentially be explained by the high dilution of the signal when the sampling frequency PRF is much higher than the signal's bandwidth.

Again, the FTCN seems to be more robust to bad quality signals.

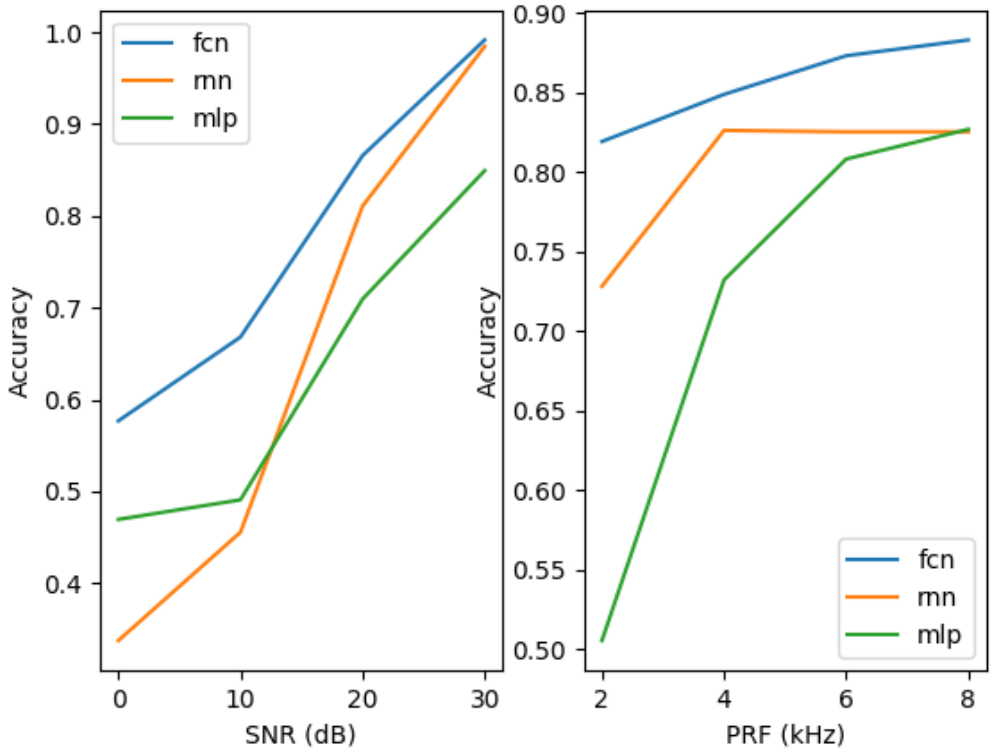

Figure 3.14.: **Performance of the learning models for increasingly good conditions**. The graphs show intuitive behavior *w.r.t.* the configurations, *i.e.* a general increase.

Super-resolved classification We now come back on the notion of temporal reosolution, or finesse, of the proposed FTCN architecture. We showed that less pooling in the newtork increased its temporal resolution, with a potential risk in loosing ability to generalize. In the following experiment, we plot, in Figure 3.15, the accuracy of the three FTCNs used above, in function of the duration of the input signal. In Figure 3.15, the input spectrograms span 40 timesteps. Using $\tau = 20$ and three different networks with finesse $\mathfrak{f} \in \{\frac{1}{4}, \frac{1}{2}, 1\}$, the input is thus seen as a series of 4 noisemes and the corresponding output feature maps' lengths are respectively 6, 11 and 21 according to equation 3.4. Confidence is plotted on the 21 timesteps of the most resolved FTCN, the other two being subsampled accordingly. To ensure a fair comparison, the three networks are identical in all except for the convolution strides: each thus has its own resolution, and will require different accumulation of data before updating its confidence. The figure shows that the gradual income of data best profits the most resolved architecture, *i.e.* here the one with no strides at all, *i.e.* with fully overlapping receptive fields.

From this experiment, we prove that controling the network's temporal resolution constitutes a thought-through architecture design choice adapted to the type of input data, which resonates with the overall philosophy of the thesis.

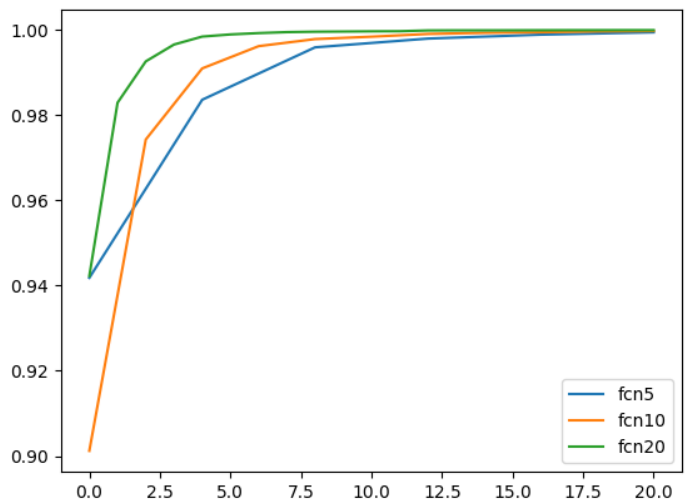

Figure 3.15.: **Growth of classification confidence over time in a very challenging environment ($SNR = 10dB$ and $PRF = 2kHZ$).** The result of interest here is how a lower finesse coefficient, *i.e.* higher temporal resolution, provides finer and faster confidence during measurements with no additional cost in global accuracy.

We also present in Figure 3.16 a similar result in a different field of application, here Environmental Audio Recognition (EAR), specifically on the UrbanSound8K dataset (Salamon et al. 2014), a commonly used dataset in this field. Again, we use a pre-trained FTCN architecture, similar in essence to the one presented above in the case of radar classification, and input signals of longer duration to observe a general increase in performance.

The key idea to retain here is that chosing a finer temporal resolution, perhaps to the detriment of spectral resolution, seems to favor better performances whether it be for the μ-D radar or EAR applications.

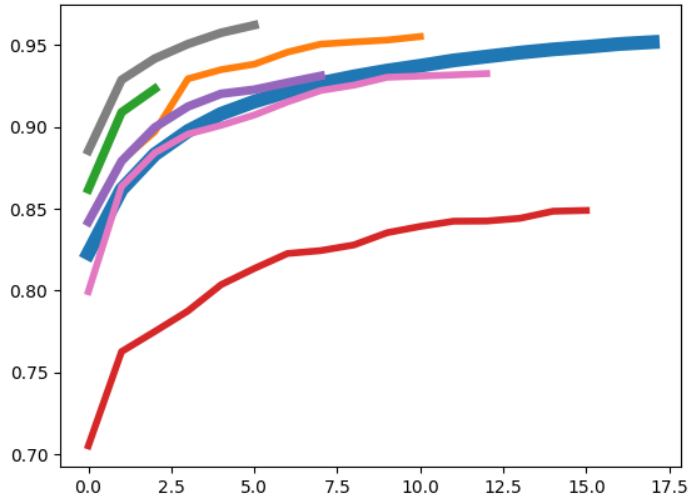

Figure 3.16.: **Growth of classification confidence of a FTCN in audio recognition**. The graph plots the confidence in function of the duration of input data. Each line groups signals of same total length; the thickness of the line represents the proportion of such signals within the dataset.

3.4.2.2 Exploring the use of complex-value data and models

Having now validated the performance, and described the advantages of the FTCN architecture, we now study how the usage of complex values within the network impacts on performance. The first step is to introduce the new design choices arising from the complex framework, with the end goal being the comparison with a real-valued conterpart.

Number of parameters Intrinsicly, a complex network will have twice as many parameters as its real counterpart; in practice, it is not obvious how this increase would affect performance. For instance, neural networks tend to better generalize in the case of a large dataset when allowed more parameters, but may also suffer from overfitting when a sufficient amount of diversified data is not met. A reasonable way to experiment on this interrogation is to allow half as many channels in the $\mathbb{CR}$Net's convolutional blocks and focusing experiments on small amounts of data. Results show that keeping the same number of channels as in the real network still performs better, which is a conclusive statement as, while the practical number of parameters has doubled, the network did not suffer even when

presented with few data. As a sanity check, doubling the number of channels in the complex network performs the worst of all cases.

The $\mathbb{CR}$Net architecture may keep the same number of parameters as its real-valued FTCN counterpart, even though it doubles in parameters and quadruples in operations.

Signal scaling and complex representation Raw radar data, along with their Fourier transforms, often exhibit major variations in scale, due to different intervening physical phenomena operating in a variety of scales. This translates to the practical habit of converting spectrograms to a logarithmic scale, most often dBs, whether it be for visualisation or further analysis. A real-valued network benefits from this rescaling from the start as the inputs are the decibel-spectrograms. In the $\mathbb{CR}$Net however, the log-scale is ambiguously defined for complex values, which allows potentially harmful variations in scale to propagate within. Proper weight initialization and BatchNorm explicitly combat this issue, but formally fail to recover a log-scale as they remain linear transformations. To this end, we propose a partially complex network for which the output complex representation is log-scaled after the passage to absolute value, and heuristically study the impact on performance of the complex-to-real ($\mathbb{C}2\mathbb{R}(x) = 10\ \log(|x|^2|)$) function's position in the layer hierarchy. The empirical conclusion is conceptually satisfying as it places the $\mathbb{C}2\mathbb{R}$ right after the final temporal representation layer, ie right before the convolutionalized fully-connected layers, as represented in Figure 3.6. This result leads to a rather natural interpretation:

While the complex spectral representation of the signal in a real-valued FTCN ends with the FFT, complex feature mapping in a $\mathbb{CR}$Net explores a hierarchy of further filter banks in addition to the Fourier-like filtering.

Fourier convolution parameters The first layer of the real-valued FTCN on spectrograms is conceptually preceded by a windowed Fourier transform, which remains a fixed pre-processing. While the previsou paragraph dealt with the scaling of the complex spectrogram within the $\mathbb{CR}$Net, the FourierNet however directly handles the raw complex data, and as such, its first layer is a 1D convolution. While conventional initialization schemes such as Xavier initialization (Glorot and Bengio 2010) can be applied, we may benefit from exploiting the spectral properties of the radar signal: indeed, experiments show a consistent improvement when using the proposed Fourier-like convolutional layer. Similarly, the window overlap percentage or hop length of the Fourier transform corresponds to the convolution stride. In the context of learning the 1D filter banks, a low stride (set to 1 in the experiments, ie maximum overlap) proved paramount to the network's performance, regardless of initialization. On the other hand, real-valued counter-

parts seemed much more robust to this hyperparameter. One interpretation of this phenomenon is that the passage from raw complex data to real-valued spectrograms averages through coherent integration any potential added information from a higher overlap, while keeping both amplitude and phase sensitizes further processing to this added information.

Using the Fourier convolution in a FourierNet may require careful hyper-parameter finetuning for it to outperform the fixed FFT in its $\mathbb{CR}$Net counterpart. For instance, a strong overlap (i.e. low stride) seems necessary to a successful training.

Quality and amount of data Throughout conducted experiments, a general trend seemed to emerge: complex networks overpowered real networks when presented with a large yet complicated dataset. Specifically, we observed improvement for SNRs on the raw data close to zero or in the negatives, positive SNRs leading to insignificant improvements. Furthermore, when the amount of training data was kept relatively small (in our scenario, less than 5 minutes), $\mathbb{CR}$Nets performed poorly to worse than their real-valued FTCN.

In short, a $\mathbb{CR}$Net seems to yield significant improvements mostly when the data are generally challenging and their amount is vast.

Numerical results We now give numerical comparisons between FTCN, $\mathbb{CR}$Net and FourierNet architectures, the latter two designed based on the previous empirical evaluations. The simulation configurations are first set to an extremely noisy case, where the raw data is $5dB$ below noise ($SNR = -5dB$). We also initially quintuple the size of the simulated dataset, yielding 20 minutes of total continuous recording per class instead of 4, our usual standard (*i.e.* again, 10 minutes of training data). For reference, a coherent integration of 20 timesteps brings the spectrum $8dB$ above noise. A PRF of $4kHz$ is used; at this frequency and with the considered drones, Doppler ambiguity, *i.e.* spectral folding, is omnipresent. As stated above, we voluntarily chose a large amount of data in a very challenging configuration. We also give performance results for the models when trained on a fraction of the data to quantify the robustness of the models to lack of data: 20% of this larger dataset corresponds to the original size, 5% to ≈ 1 minute per class (30 seconds of training data). The rest of the experimental setup remains the same. Results on the three models are presented in table 3.3.

Table 3.3.: **Performance comparison of real and complex deep structures on radar data on various amount of noisy data**. Models are compared on a decreasing amount of training data, from the full 10 minutes to 30 seconds.

Train size	100%	20%	5%
FTCN	67.2 ± 0.27	65.1 ± 0.39	$\mathbf{63.5 \pm 0.46}$
$\mathbb{CR}$Net	68.8 ± 0.17	65.1 ± 0.50	59.3 ± 1.51
FourierNet	$\mathbf{70.8 \pm 0.22}$	$\mathbf{67.8 \pm 0.40}$	62.1 ± 0.90

The first observation is the improvement of the two complex networks over the real counterpart when given all 10 minutes of training data (the 50% training split of the total 20 minutes), the FourierNet being superior to the $\mathbb{CR}$Net. Given 20% of available training data (2 minutes), the FourierNet still outperforms all models, but the $\mathbb{CR}$Net starts decreasing towards the FTCN's performance. Given only 5% of training data (30 seconds), all complex models perform worse than the FTCN. Finally, we repeat the experiments on a cleaner dataset, by changing the SNR from $-5dB$ to $5dB$ (we limit ourselves to the FTCN and FourierNet). Results shown in table 3.4 naturally exhibit better performances overall, but the FourierNet struggles to outperform the FTCN, which supports the argument of complex networks working noticeably better in challenging configurations only.

In summary, the complex models seem to improve upon a real-valued couterpart, in situations where data is plentiful and inherently challenging. They thus seem a poor choice in scarce data scenarios, or when a real-valued model already comes close to solving the task. The cost is twice as many parameters and four times as many computations, leading to slower training and inference.

Table 3.4.: **Performance comparison of real and complex deep structures on radar data on various amount of less noisy data**. Models are compared on a decreasing amount of training data, from the full 10 minutes to 30 seconds.

Train size	100%	20%	5%
FTCN	98.6 ± 0.34	94.3 ± 0.57	$\mathbf{91.6 \pm 0.98}$
FourierNet	$\mathbf{99.0 \pm 0.07}$	$\mathbf{94.4 \pm 0.14}$	88.7 ± 1.12

3.4.2.3 Comparison with second-order models and validation of the full pipeline

We now add to the experiment pool the SPDNet operating on covariance matrices, which will then allow us to proceed to evaluating the proposed pipeline. The

Table 3.5.: **Performance comparison of first- and second-order models on radar data**. The SOFTNet model combines both orders in a single pipeline; its performance expectedly matches or tops that of the individual, unit models.

Train size	100%	20%	5%
SPDNet	92.6 ± 0.54	91.5 ± 0.74	88.4 ± 3.06
HPDNet	94.4 ± 0.76	91.8 ± 0.80	87.1 ± 1.11
FTCN	98.9 ± 0.44	93.4 ± 1.21	84.3 ± 2.51
FourierNet	99.4 ± 0.17	96.2 ± 1.12	87.4 ± 1.94
SpectroSPD	95.1 ± 0.49	91.9 ± 0.82	84.6 ± 3.49
SOFTNet	99.5 ± 0.16	97.2 ± 0.90	93.9 ± 0.74

same windowing is used for covariance and spectral representations to keep comparisons fair, as per illustrated in Figure 3.10. We bifurcate the introduced pipeline at various stages in various configurations, which amounts to different learning models which we relate to in Figure 3.8, specifically the following:

1. SPDNet: a SPDNet on the raw complex data's covariance immediately made real, estimated over 99 windows of 20 samples;
2. HPDNet: a HPDNet on the raw complex data's covariance, estimated over 99 windows of 20 samples;
3. Spectral SPD Neural Network (SpectroSPD): a SPDNet on spectrum time series;
4. FTCN: a FTCN on spectrograms (mathematically equivalent to spectrum time series);
5. FourierNet: a FTCN on raw complex data as described above, where Fourier filter banks are fine-tuned;
6. SOFTNet: a Second-Order Fully Temporal Network, where a SPDNet is appended to the the final feature representation of the FTCN.

Furthermore, we repeat the experiments with decreasing amount of training data in the hope that injecting geometric information in the learning through second-order modelling would compensate for lack in data volume. Results are displayed in Table 3.5.

The first remark is that the SPDNet yields the worst accuracy when all training data is available. Intuitively, it makes sense that it score lower than SpectroSPD and SOFTNet as the covariance is then sampled from a more adapted or discriminative model, *i.e.* respectively a spectrogram and FTCN learnt features. The HPDNet performs better mostly when much data is available, losing its potential when data is scarce while being more greedy on computation time. As for the

FTCN and FourierNet, these are deep learning models with a total of about 14000 parameters or more, whereas the SPDNet is the Riemannian equivalent a rather shallow network, with only about 700 parameters.

However, as training data decreases, we see the SPD methods become better than the traditional deep models: they seem to exhibit much higher robustness to lack of data, which validates the usefulness of exploiting the geometric information of the data.

The final observation concerns the SOFTNet (Figure 3.9), which performs best across all configurations. In a sense, this model benefits both from deep temporal feature learning, and from covariance geometry modeling. In practice, affordable overhead is observed when using this larger model: one epoch lasts $\sim 7s$, compared to $\sim 1.5s$ for the FTCN trained on a Nvidia GTX 1070M GPU and $\sim 3.5s$ for the SPDNet trained on a i7-6700HQ CPU.

To conclude on the classification pipeline:

> *Riemannian SPD-based models don't compete against Euclidean deep models when data is plentiful. However, the exhibit strong robustness to the lack of data, outperforming the latter in scarcity scenarios. The full SOFTNet pipeline exhibits complimentary behaviours, yielding competitive performance throughout all situations.*

Details execution times, code snippets illustrating the models mentioned throughout this chapter, along with indications for implementation can be found in Appendix A.

3.5 Conclusion

In the order of things, we first introduced an expressive simulator for blade-propelled engines such as drones, or Unmanned Aircraft Vehicles (UAVs). The simulator sports sufficiently variational characteristics, and allows for the generation of realistic μ-D radar signals. The synthetic datasets thereby obtained were thus suited for the Machine Learning (ML) task of classification on several distinct models: Multi Layer Perceptrons (MLPs), Recurrent Neural Networks (RNNs), and Convolutional Neural Networks (CNNs). In a first set of experimentations, we assessed these models performed well even in harsh simulation conditions, from a radar operation viewpoint. We furthermore justified the assumption that the recurrent and convolutional models, because they inherently learn temporal fluctuations, are particularly appropriate to the task of radar classification.

In particular, we proposed a relatively lightweight Fully Temporal Convolutional Network (FTCN), which, being free of fully-connected layers, is able to handle signals of varying length over time. The FTCN showcases two interesting properties: its output feature maps preserve the temporal structure of the signal, and are themselves a time series, and, it is possible to control exactly the temporal resolution of this output feature time series. The properties of the FTCN, along with its experimental success on both synthetic and real datasets, make it a satisfying candidate for the main streamline of this thesis, the search for efficient models, adapted to the underlying structure of the data.

We also developed a partially complex-valued counterpart to the FTCN, in order to take into account the inherent complex structure of μ-D data, which again follows the line of research throughout this thesis. We furthermore introduced a Fourier-like convolutional layer, which harvests the advantages of both the Fourier transform and of learning filter banks on the raw data, an intuition proved to be consistently true in practice. We performed extensive experimentation on synthetic data to isolate the cases where performance benefitted from complex values. The main conclusions obtained were, that above a certain amount of observed data (a couple of minutes for our datasets), in challenging configurations (under $5dB$ of SNR in our scenarios), complex-valued networks significantly outperformed their real counterparts. These results initiate a hopeful stance on introducing complex values in deep learning-based classification methods on μ-D radar data.

Finally, we introduced a neural pipeline consisting of a first-order FTCN, onto which a second-order SPD neural network (SPDNet) can be appended at any stage of the first-order model. We find that the deeper down the first-order model, the better the performance of the global second-order model. The best-performing, and also deepest model issued was baptized Second-Order Fully Temporal Network (SOFTNet), and consists in a succession of meaningful representations of the data: raw complex form, Fourier transform, deep features and covariance. Furthermore, models exploiting covariance structure seem to be more robust to lack of data than the first-order models, even outperforming them in critically sparse scenarios (with only a few seconds of training data per class). Finally, the end-to-end second-order model outperforms all other models in any of the experimented configurations, which points to the definite possibility of getting the best from the two worlds: a set of discriminative features learnt by convolutional layers, and a Riemannian processing on the resulting temporal covariance matrices.

Opening perspectives envisioned throughout this chapter are numerous. The drone radar simulator on the one hand, could surely benefit from further developments such as its refinement to real-life physical and maneuvering subtleties, and its extension to other kinds of UAVs, as it is for now limited to rotary-blade

drones (the most common type, however). As for the ML perspective, the presented temporal architectures pave the way to more difficult and varied tasks than classification, for instance detection, labelling and segmentation, other key tasks in the field of radar operations. On the other hand, concerning the SOFTNet pipeline, many undiscussed technicalities on how to connect the first- and second-order networks give rise to interesting challenges; the briefly mentioned covariance estimation operator, in itself, constitutes a field of research by itself. Other ideas one might come up with include the smoothing of the connection: penalizing the gradients passing by the connection (either with L2 loss or an additional Stiefel constraint), using a smooth transition kernel after the first-order representation space, or after the covariance estimation, adding layers in between... On the architectural side, one may look back at Figure 3.8 and wonder whether a fusion of each of the four feature representations could be imagined; again, the semantic feature fusion promises major research material.

CHAPTER 4

ADVANCES IN SPD NEURAL NETWORKS

Chapter abstract

This chapter addresses the budding area of geometric deep architectures for covariance processing. Specifically, it is devoted to the invention of novel layers for the SPD neural network (SPDNet) presented earlier on. The driving challenge throughout the chapter will be the conformation of these layers to the particular manifold geometry of Symmetric Positive Definite (SPD) matrices. In this spirit, we first introduce a natural improvement of these architectures, based on the precepts of Information Geometry (IG). This improvement consists in a data-aware normalization of the final layer, the Euclidean mapping, which is of paramount importance to the SPDNet. We propose two distinct normalization schemes, one based on the data's barycenter, called Barycentric Normalization (BarNorm), the other on a learned parameter, called Parametric Normalization (ParNorm). We name Data-Aware Mapping Network (DAMNet) an architecture equipped with either of the normalization layers. Then, we introduce a Riemannian Batch Normalization (BatchNorm) algorithm, set within a novel Batch-Normalized SPDNet (SPDNetBN) architecture. Finally, we develop a convolutional layer for the SPD matrices. These novel layers make use of geometric operations on the manifold, notably the Riemannian barycenter, parallel transport and non-linear structured matrix transformations. We derive a new manifold-constrained gradient descent algorithm working in the space of SPD matrices, allowing to learn the BatchNorm, or BatchNorm layer. We validate our proposed approach with extensive experiments in three different contexts on diverse data types: drone recognition data from radar observations, and on emotion and action recognition datasets from video and Motion Capture (MoCap) data. Experiments show that the SPDNetBN systematically gives better classification performance compared with leading methods and a remarkable robustness to lack of data.

Contents

4.1 Introduction **97**
4.2 Data-Aware Mapping Network **100**
4.2.1 Log-Euclidean Metric 100
4.2.2 Barycentric Normalization 102
4.2.3 Parametric Normalization 102
4.2.4 DAMNet architecture 103
4.3 Batch-Normalized SPDNet **104**
4.3.1 Centering SPD matrices using Parallel Transport 104
4.3.2 Statistical distribution on SPD matrices 107
4.3.3 Riemannian BatchNorm algorithm 108
4.4 Riemannian manifold-constrained optimization **108**
4.4.1 Learning with SPD constraint 109
4.4.2 Structured matrix backpropagation 110
4.5 Convolution for covariance time series **116**
4.5.1 Single-channel convolution 116
4.5.2 Weighted average convolution 116
4.5.3 Riemannian convolution using the weighted Fréchet mean 117
4.5.4 Multi-channel convolution 118
4.6 Experimental validation **119**
4.6.1 Drones recognition 119
4.6.2 Emotion recognition 127
4.6.3 Action recognition 128
4.7 Conclusion **130**

4.1 Introduction

Covariance matrices are ubiquitous in any statistical related field but their direct usage as a representation of the data for machine learning is less common. However, it has proved its usefulness in a variety of applications: object detection in images (Tuzel et al. 2006), analysis of Magnetic Resonance Imagery (MRI) data (Pennec et al. 2006), classification of electroencephalography (EEG) time series for Brain-Computer Interfaces (BCIs) (Barachant et al. 2013). It is particularly interesting in the case of structured temporal data since a global covariance matrix is a straightforward way to capture and represent the temporal fluctuations of data points of different lengths. As previously stated, the main difficulty resides in their inherently curved nature, which calls for the usage of tools from non-Euclidean geometry. A full overview of the Riemannian nature of Symmetric Positive Definite (SPD) matrices, along with a plethora of theoretical justifications and properties on the matter, is given in the book of Bhatia (2015). For this reason most of classification methods (which implicitly make the hypothesis of a Euclidean input space) cannot be used successfully on SPD matrices. Furthermore, this mathematical difficulty comes with a much higher cost: computation often involves eigenvalues decompositions or expensive iterative algorithms. Lack of straightforward parallelizability and super-quadratic algorithmic complexity often hinders the direct usage of covariate data in learning models.

Interestingly, relatively lightweight machine learning techniques can nonetheless produce state-of-the-art results as soon as the particular Riemannian geometry is taken into account. This is the case for BCI: Barachant et al. (2013) and Barachant et al. (2012) respectively use a nearest Riemannian barycenter scheme (described in Section 2.5.2) and a tangent-space Support Vector Machine (SVM) (described in Section 2.5.3) to successfully classify covariances matrices computed on multivariate EEG signals; in the same field, Yger and Sugiyama (2015) propose kernel methods for metric learning on the SPD manifold . Another example is in MRI, where Pennec et al. (2006) and Arsigny et al. (2006) develop a Riemannian k-nearest neighbours (k-NN) as described in Section 2.5.1. Motion recognition from Motion Capture (MoCap) skeletal data also benefits from Riemannian geometry, as exposed in Cavazza et al. (2017), Huang et al. (2018), and Huang et al. (2016).

As mentioned before, a SPD neural network (SPDNet) architecture specifically adapted for these matrices has been proposed in the context of neural networks (Huang and Van Gool 2017). While the overall aspect is similar to a classical (Euclidean) network (transformations, activations and a final stage of classification), each layer

processes a point on the SPD manifold; the final Riemanian layer transforms the feature manifold to a Euclidean space for further classification. Following this seminal work, more architectures have followed, proposing alternatives to the basic building blocks: in Dong et al. (2017) and Gao et al. (2017), a more lightweight transformation layer is proposed; in T. Zhang et al. (2018) and Chakraborty et al. (2018), the authors propose alternate convolutional layers, respectively based on multi-channel SPD representations and Riemannian means; a recurrent model is further proposed in Chakraborty et al. (2018); in Mao et al. (2019) and P. Li et al. (2018), an approximate matrix square-root layer replaces the final Euclidean projection to lighten computational complexity. Transversally, works on accelerating the optimization of SPD layers are emerging, in answer to the frustratingly slow pace the optimization usually takes place at (H. Zhang et al. 2016; Liu et al. 2017; Alimisis et al. 2020). In Acharya et al. (2018) and Yu and Salzmann (2017), a SPDNet is appended to a Euclidean Convolutional Neural Network (CNN) to improve on performance. In this case, a so-called covariance pooling (CovPool) layer is devoted to compute a Sample Covariance Matrix (SCM) of the final feature maps.

All in all, most of the developments focus on improving or modifying existing blocks in an effort to converge to their most relevant form, both theoretically and practically; throughout this chapter, we will propose new building blocks, with a shared goal of normalizing the inner mechanics of the SPDNet architecture. As an additional, independent SPD building block, this novel layer is agnostic to the particular way the other layers are computed, and as such can fit into any of the above architectures; we do however focus on the original one (Huang and Van Gool 2017).

In our first proposition, we introduce a new architecture, called a Data-Aware Mapping Network (DAMNet), which focuses on the final layer of the SPDNet, the Euclidean mapping: we argue that a better projection can be done by making the layer dependent on the data. The original projection layer relies on the Log-Euclidean Metric (LEM) framework (Arsigny et al. 2006), which endows the manifold of SPD matrices with a Lie group structure. This framework is much simpler than the full Riemanian setting and allows efficient computations while keeping good theoretical properties (Pennec et al. 2006). While useful, this framework is only a particular case: we thus introduce an improved projection layer working in the broader Riemaniann setting. This new projection maps the points to the tangent space of some reference matrix and comes in two variants: a barycentric projection, called Barycentric Normalization (BarNorm), which uses the Riemannian barycenter as the reference matrix; and a parametric projection, called Parametric Normalization (ParNorm), which uses a parameter SPD matrix, learnt during training.

Our second main contribution is inspired by the well-known and well-used Batch Normalization (BatchNorm) layer, introduced in the context of (Euclidean) CNNs for Computer Vision (CV) tasks in Ioffe and Szegedy (2015). This layer makes use of batch centering and biasing, operations which in our case need to be defined on the SPD manifold. Although the overall structure of the original BatchNorm is preserved, its generalization to SPD matrices requires geometric tools on the manifold, both for the forward and backward pass. The overall architecture, which we call Batch-Normalized SPDNet (SPDNetBN), is expected to perform better than either the SPDNet or DAMNet.

In this study, we further assess the particular interest of batch-normalized SPDNets in the context of learning on scarce data with lightweight models: indeed, many fields are faced with costly, private or evasive data, which strongly motivates the exploration of architectures naturally resilient to such challenging situations. Medical imagery data is well-known to face these issues (Pennec et al. 2006), as is the field of drone radar classification (Brigant et al. 2016). Indeed, radar signal acquisition is prohibitively expensive, the acquired data is usually of confidential nature, and drone classification in particular is plagued with an ever-changing pool of targets, which we can never reasonably hope to encapsulate in comprehensive datasets. Furthermore, hardware integration limitations, such as for real-time embedded systems-on-chip in the context of radar systems, further motivate the development of lightweight models based on a powerful representation of the data.

To summarize our proposed contributions:

- A neural architecture called Data-Aware Mapping Network (DAMNet) with two new layers (which are mutually exclusive):
  - → A barycentric normalization layer using a Riemannian barycenter, called BarNorm;
  - → A parametric normalization layer using a SPD-constrained parameter, called ParNorm;
- A neural architecture called Batch-Normalized SPDNet (SPDNetBN) with one new layer:
  - → A Riemannian BatchNorm layer for SPD neural networks, respecting the manifold's geometry;
- A generalized gradient descent allowing to learn the DAMNet and SPDNetBN models;
- A convolutional layer for SPD matrices;

- Extensive experimentations on three datasets from three different fields.

The chapter is organized as follows: we begin Section 4.2 by describing the DAMNet architecture with the BarNorm or ParNorm layers, along with a brief mathematical reminder of its driving motivation. We then move on in Section 4.3 to describe our proposed Riemannian BatchNorm algorithm; again, essential concepts to the layer are debriefed. Section 4.4 ends the technical contributions with by devising the projected gradient descent algorithm for learning the aforementioned layers. Finally, we validate experimentally and assess the properties of our proposed architectures in Section 4.6.

4.2 Data-Aware Mapping Network

The DAMNet architecture improves upon the original SPDNet. The core incentive behind this improvement lies in the formal generalization of the background mathematical framework the Euclidean mapping layer us based upon. Specifically, the Log Eigenvalues (LogEig) layer described in Section 3.2.1 is explicitly defined within the realms of the LEM framework; we propose to elevate the notion to the more general Riemannian framework. We begin by briefly describing this LEM framework.

4.2.1 Log-Euclidean Metric

We recall the notations $\mathcal{S}_*^+$ the space of SPD matrices and $\mathcal{S}^+$ the space of symmetric matrices. Let us define the logarithmic product of two SPD matrices:

$$\forall P_1, P_2 \in \mathcal{S}_*^+,\ P_1 \odot P_2 = \exp(\log(P_1) + \log(P_2)) \tag{4.1}$$

We now endow $\mathcal{S}_*^+$ with the group structure defined by the logarithmic product. We claim that $(\mathcal{S}_*^+ \odot, I_d)$ is an abelian Lie group, the neutral element I_d being the identity matrix and the inversion the standard matrix inversion $(\cdot)^{-1}$. This is easy to see, as the logarithmic product commutes (yielding a group structure) and involves smooth differentiable operations in matrix space (yielding a Lie group structure). Furthermore, it reduces to the standard matrix product when the matrices commute:

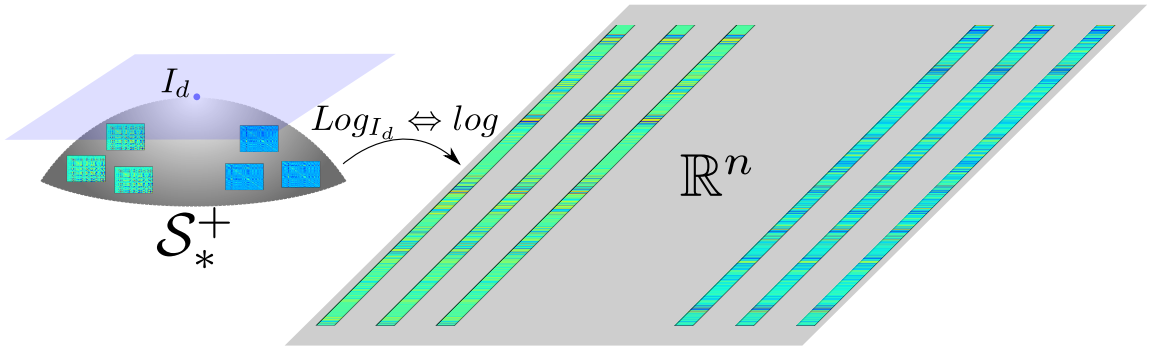

Figure 4.1.: **LogEig as a special case of logarithmic mapping**. While the LogEig is defined within the LEM framework, its generalization to the AIRM framework allows for a more flexible, powerful data-aware mapping. The main idea to retain is the passage from the manifold to a certain Euclidean space.

$$\forall P_1, P_2 \in \mathcal{S}_*^+ \mid P_1 P_2 = P_2 P_1, \;\; P_1 \odot P_2 = P_1 P_2 \tag{4.2}$$

The Lie group exponential associated to the Lie group is by definition the exponential mapping Exp of the associated manifold $\mathcal{S}_*^+$ at the neutral element; recall from its definition in Equation 2.30, that $\mathrm{Exp}_{I_d} = \exp$, the standard exponential mapping. Its inverse is thus the standard logarithm, and thus the associated Lie algebra $\mathfrak{s}$ corresponds to $\mathcal{S}^+$, as $\log \; : \; \mathcal{S}_*^+ \longrightarrow \mathcal{S}^+$ constitutes a diffeomorphism. The LEM noted δ_{Log} is then defined as the Euclidean metric on the Lie algebra:

$$\delta_{\mathrm{Log}}(P_1, P_2) = ||\log(P_1) - \log(P_2)|| \tag{4.3}$$

From the elements described above, we now see that the LogEig layer (Equation 3.3) corresponds to a special case of the logarithmic mapping on $\mathcal{S}_*^+$ (Equation 2.30), specifically with the reference G point being the identity matrix. We illustrate this important notion in the context of a SPDNet in Figure 4.1. In fact, the LEM itself is a special case of the Affine Invariant Riemannian Metric (AIRM) in the sub-group of commuting SPD matrices. Our initial main idea thus consists in generalizing the concept by considering data- and geometry-aware mappings based on the more general AIRM framework, *i.e.* in practice mapping the points at the LogEig layer other than at the identity matrix. The question then becomes, how to chose this reference point.

4.2.2 Barycentric Normalization

Our first idea is to map data points at their barycenter's tangent space. It is important here to distinguish the Riemannian (or geometric) barycenter $\mathfrak{G}$ defined on the manifold from the standard arithmetic mean $\frac{1}{N}\sum_{i\leq N} P_i$. Note that the arithmetic mean might in the general case not even belong to the manifold, although it actually does in $\mathcal{S}_*^+$. Recall Section 2.5.2 for the computation of $\mathfrak{G}$ using the Karcher flow algorithm.

As the batch barycenter changes with each batch, the tangent space will be different at each iteration. Posterior to the mapping, classification is performed by learning a separating hyperplane, *i.e.* a standard dense layer. However, this process may deal poorly with an ever-evolving projection space to learn on: indeed, each step in the optimization would yield a different vector space. Therefore, we instead center, or parallel transport, each point $P_i^{(k)}$ at the final SPD layer (k) in the batch around their geometric barycenter $\mathfrak{G}_{\{P_i^{(k)}\}_{i\leq N}}$, as done in Yger (2013) and Barachant et al. (2013), which amounts to omitting the final congruence in the logarithmic map in Equation 2.30. More details on centering with parallel transport are given in the next section. At each iteration, we compute the barycenter $\mathfrak{G}_{\{P_i^{(k)}\}_{i\leq N}}$ of the current batch of transformed data points $\{P_i\}_{i\leq N}$. All in all, this BarNorm layer becomes:

$$\begin{aligned} &\mathfrak{G} := \mathfrak{G}_{\{P_i^{(k)}\}_{i\leq N}}(\text{barycenter of the mini-batch}) \\ &X^{(k)} = \log(\mathfrak{G}^{-\frac{1}{2}} P^{(k)} \mathfrak{G}^{-\frac{1}{2}}) \end{aligned} \tag{4.4}$$

4.2.3 Parametric Normalization

In the same vein as previously, we aim to generalize the Euclidean projection from the LEM framework to the AIRM framework, for which a reference point G is required. G can be the barycenter, which takes data into account by optimizing a minimal dispersion criterion within the batch. Since this criterion has no reason to be related to the training objective, we introduce a second data-driven mapping layer, this time explicitly optimizing the SPD reference point G for the global training loss.

Contrary to the BarNorm layer described previously, G is now an additional parameter of the network to be learnt alongside the others. The learning mechanics linked to such an operation are covered further on in this chapter. The difference with the standard LogEig then reduces to a congruence by $G^{-\frac{1}{2}}$ before applying the $\log$. However, G needs to belong to $\mathcal{S}_*^+$, which we will need to enforce in

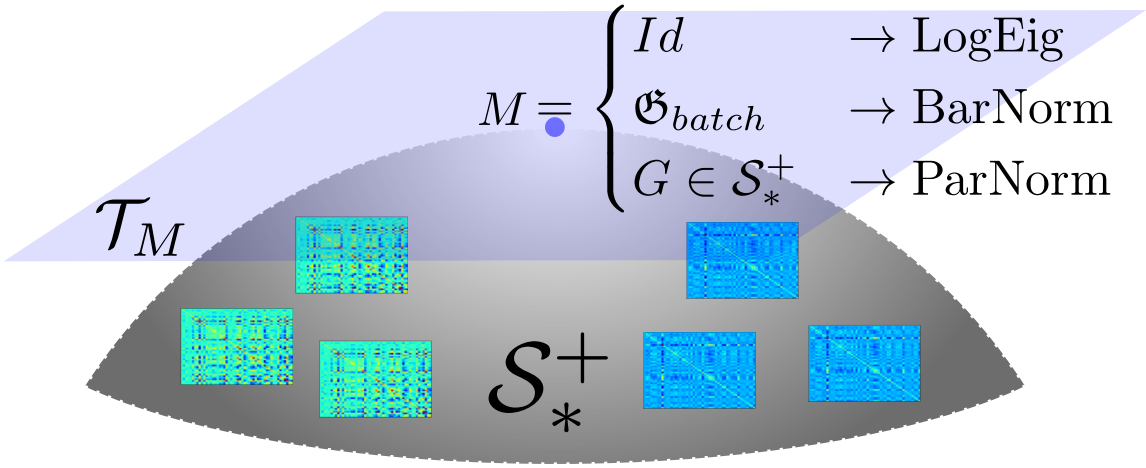

Figure 4.2.: **Illustration of the DAMNet architecture**. When the reference point of the Euclidean mapping is the identity matrix, it reduces to the SPDNet architecture.

the gradient update, as done in Yger and Sugiyama (2015). Note that a similar constraint applies to the Bilinear Mapping (BiMap) weights (Equation 3.1) of the SPDNet, only the latter is orthogonal rather than symmetric. Finally, the parametric projection for the learnable logarithm of the transformed data points $P^{(k)}$ at the last SPD layer reads:

$$X^{(k)} = \log(G^{-\frac{1}{2}} P^{(k)} G^{-\frac{1}{2}}), G \in \mathcal{S}_*^+ \tag{4.5}$$

4.2.4 DAMNet architecture

To sum up, the difference between a SPDNet and a DAMNet is the Euclidean mapping, which in the case of the former is fixed to the matrix logarithm, contrary to the latter which depends on a data-aware reference point M, defined either as an additional parameter or a barycenter; as such; SPDNet is a special case of DAMNet, as illustrated in Equation 4.6 and in Figure 4.2:

$$\begin{aligned} X^{(k)} &= log(M^{-\frac{1}{2}} P^{(k)} M^{-\frac{1}{2}}) \text{ , with} \\ M &= \begin{cases} Id & \rightarrow \text{LogEig with no regularization} \\ \mathfrak{G}_{batch} & \rightarrow \text{BarNorm (barycenter)} \\ G \in \mathcal{S}_*^+ & \rightarrow \text{ParNorm (gradient descent)} \end{cases} \end{aligned} \tag{4.6}$$

In the DAMNet architecture, the two normalization schemes BarNorm and ParNorm are mutually exclusive. However, they are not incompatibe, and can in principle be chained. Moreover, while our theoretical justification for their existence focused on the LogEig layer, it still holds at any echelon of the network. The centering of a batch within the network, followed by a parametric displacement of this batch, may remind the acute reader of the famed Batch Normalization algorithm. We thus extend in the next section the ideas underlying the DAMNet, to develop a Batch-Normalized SPDNet (SPDNetBN) architecture.

4.3 Batch-Normalized SPDNet

This section is dedicated to the Riemanian BatchNorm and its associated architecture, the SPDNetBN, a core contribution in our works. As stated above, it generalizes our DAMNet architecture, which itself generalizes the original SPDNet. The Riemannian BatchNorm shares the goal of its Euclidean counterpart, that is essentially the normalization of data conditioning through the network by the reduction of internal covariate shift (Ioffe and Szegedy 2015). Recall Section 2.3.2 for a brief overview of the standard, Euclidean BatchNorm algorithm. We begin the section by clarifying a matter of importance, sheepishly skimmed over in the previous section. While we initially introduced the normalization schemes as different logarithmic mappings, in reality they are defined from Parallel Transport (PT). Similarly, the centering and addition of bias within the proposed Riemannian BatchNorm will involve PT. We thus clarify this ambiguity in the context of the Riemannian BatchNorm.

4.3.1 Centering SPD matrices using Parallel Transport

The Euclidean BatchNorm involves centering and biasing the batch $\mathcal{B}$, which is done via subtraction and addition. However on a curved manifold, there is no such group structure in general, so these seemingly basic operations are ill-defined. To shift SPD matrices around their mean $\mathfrak{G}$ or towards a bias parameter $G \in \mathcal{S}_*^+$, we propose to rather use parallel transport on the manifold (Amari 2016).

4.3.1.1 Parallel Transport and SPD transport

In short, the operator $\Gamma_{P_1 \to P_2}(S)$ of a vector $S \in \mathcal{T}_{P_1}$ in the tangent plane at P_1, between $P_1, P_2 \in \mathcal{S}_*^+$ defines the path from P_1 to P_2 such that S remains parallel to itself in the tangent planes along the path. The geodesic $\gamma_{P_1 \to P_2}$ is itself a special case of the PT, when S is chosen to be the direction vector $\gamma'_{P_1 \to P_2}(0)$ from P_1 to P_2. The expression for PT is known on $\mathcal{S}_*^+$:

$$\forall S \in \mathcal{T}_{P_1},\ \Gamma_{P_1 \to P_2}(S) = (P_2 P_1^{-1})^{\frac{1}{2}}\ S\ (P_1^{-1} P_2)^{\frac{1}{2}} \in \mathcal{T}_{P_2} \tag{4.7}$$

The equation above defines PT for tangent vectors, while we wish to transport points on the manifold. To do so, we simply project the data points to the tangent space at the initial point P_1 using the logarithmic mapping at P_1, parallel transport the resulting vector from Equation 4.7 towards P_2 which we then map back to the manifold using exponential mapping at P_2. We show in Theorem 4.1 that the resulting operation, which we call SPD transport, turns out to be exactly the same as the formula above, which is not an obvious result in itself. Note that by abuse of notation, we also use $\Gamma_{P_1 \to P_2}$ to denote the SPD transport. We first show two useful lemmas for the proof:

Lemma 4.1. $\forall P_1, P_2 \in \mathcal{S}_*^+,\ A := (P_2 P_1^{-1})^{\frac{1}{2}} = P_1^{\frac{1}{2}} (P_1^{-\frac{1}{2}} P_2 P_1^{-\frac{1}{2}})^{\frac{1}{2}} P_1^{-\frac{1}{2}} =: B$

Proof. A and B trivially are similar matrices, as in they share the same eigenvalues. These values are by construction all strictly positive, therefore they are the unique square roots of the respective squares, and it thus suffices to prove $A^2 = B^2$. We have:

$$\begin{aligned}
B^2 &= \left(P_1^{\frac{1}{2}} (P_1^{-\frac{1}{2}} P_2 P_1^{-\frac{1}{2}})^{\frac{1}{2}} P_1^{-\frac{1}{2}}\right) \left(P_1^{\frac{1}{2}} (P_1^{-\frac{1}{2}} P_2 P_1^{-\frac{1}{2}})^{\frac{1}{2}} P_1^{-\frac{1}{2}}\right) \\
&= P_1^{\frac{1}{2}} (P_1^{-\frac{1}{2}} P_2 P_1^{-\frac{1}{2}}) P_1^{-\frac{1}{2}} \\
&= P_2 P_1^{-1} \\
&= A^2 \\
\Rightarrow B &= A
\end{aligned}$$

■

Lemma 4.2. $\forall P_1, P_2 \in \mathcal{S}_*^+,\ X := P_2^{-\frac{1}{2}} B P_1^{\frac{1}{2}}$ *is unitary, i.e.* $XX^T = X^T X$ *(using the same notations as in Lemma 4.1).*

Proof.

$$
\begin{aligned}
XX^T &= \Bigg(P_2^{-\frac{1}{2}}\underbrace{\big(P_1^{\frac{1}{2}}(P_1^{-\frac{1}{2}}P_2P_1^{-\frac{1}{2}})^{\frac{1}{2}}\overbrace{P_1^{-\frac{1}{2}}}^{I_d}\big)P_1^{\frac{1}{2}}}_{B}\Bigg)\Bigg(\overbrace{P_1^{\frac{1}{2}}\big(P_1^{-\frac{1}{2}}}^{I_d}\underbrace{(P_1^{\frac{1}{2}}P_2P_1^{-\frac{1}{2}})^{\frac{1}{2}^T}P_1^{\frac{1}{2}}}_{B^T}\big)P_2^{-\frac{1}{2}}\Bigg)\\
&= (P_2^{-\frac{1}{2}}\overbrace{P_1^{\frac{1}{2}})(P_1^{-\frac{1}{2}}}^{I_d}\underbrace{P_2\overbrace{P_1^{-\frac{1}{2}})^{\frac{2}{2}}}^{I_d}}_{\in\mathcal{S}_*^+}(P_1^{\frac{1}{2}}P_2^{-\frac{1}{2}})\\
&= P_2^{-\frac{1}{2}}P_2P_2^{-\frac{1}{2}}\\
&= I_d
\end{aligned}
$$

■

As a consequence of X being unitary, we have that $\exp(XMX^T) = X\exp(M)X^T$ for any matrix M (by definition of the matrix exponential $\exp(M) = \sum_{\mathbb{N}} \frac{M^k}{k!}$). We now show the main result, establishing the SPD transport $\Gamma_{P_1\to P_2}(S)$ of $S \in \mathcal{S}_*^+$ from P_1 to P_2.

Theorem 4.1. *SPD transport of SPD matrix S from P_1 to P_2 on $\mathcal{S}_*^+$.*

$$
\forall P_1, P_1, S \in \mathcal{S}_*^+,\ \Gamma_{P_1\to P_2}(S) = (P_2P_1^{-1})^{\frac{1}{2}}\ S\ (P_1^{-1}P_2)^{\frac{1}{2}} \in \mathcal{S}_*^+ \tag{4.8}
$$

Proof.

$$
\begin{aligned}
\Gamma_{P_1\to P_2}(S) &= \mathrm{Exp}_{P_2}\Big(\Gamma_{P_1\to P_2}\big(\mathrm{Log}_{P_1}(S)\big)\Big) && \text{(by definition)}\\
&= P_2^{\frac{1}{2}}\exp\Bigg(\underbrace{P_2^{-\frac{1}{2}}\big(\overbrace{(P_2P_1^{-1})^{\frac{1}{2}}}^{=A=B}P_1^{\frac{1}{2}}}_{X}\log(P_1^{-\frac{1}{2}}SP_1^{-\frac{1}{2}})\underbrace{P_1^{\frac{1}{2}}\overbrace{(P_1^{-1}P_2)^{\frac{1}{2}}}^{=A^T=B^T}\big)P_2^{-\frac{1}{2}}}_{X^T}\Bigg)P_2^{\frac{1}{2}} && \text{(Lemma 4.1)}\\
&= P_2^{\frac{1}{2}}X\exp\Big(\log(P_1^{-\frac{1}{2}}SP_1^{-\frac{1}{2}})\Big)X^TP_2^{\frac{1}{2}} && \text{(Lemma 4.2)}\\
&= \overbrace{P_2^{\frac{1}{2}}P_2^{-\frac{1}{2}}}^{I_d}\ B\ \overbrace{P_1^{\frac{1}{2}}P_1^{-\frac{1}{2}}}^{I_d}\ S\ \overbrace{P_1^{-\frac{1}{2}}P_1^{\frac{1}{2}}}^{I_d}\ B^T\ \overbrace{P_2^{-\frac{1}{2}}P_2^{\frac{1}{2}}}^{I_d} && \text{(Lemma 4.2)}\\
&= (P_2P_1^{-1})^{\frac{1}{2}}\ S\ (P_1^{-1}P_2)^{\frac{1}{2}} && \text{(Lemma 4.1)}
\end{aligned}
$$

■

4.3.1.2 SPD centering and biasing

The definition of the SPD transport Γ clarifies the BarNorm and ParNorm operators, and allows the formal definition of

- The batch centering of matrices $\{P_i\}_{i \leq N}$ with Riemannian barycenter $\mathfrak{G}$ as the PT from $\mathfrak{G}$ to the identity I_d;
- The biasing of the batch towards a parametric SPD matrix G as the PT from I_d to G.

Note that the transport is transitive (Yair et al. 2019), so both operations can be concatenated in a single transport. We can now fully define the batch centering and biasing:

$$\text{Centering from } \mathfrak{G} := \mathrm{Bar}(\mathcal{B}): \quad \forall i \leq N, \ \bar{P}_i = \Gamma_{\mathfrak{G} \to I_d}(P_i) = \mathfrak{G}^{-\frac{1}{2}} \ P_i \ \mathfrak{G}^{-\frac{1}{2}} \tag{4.9a}$$

$$\text{Biasing towards parameter } G\text{: } \forall i \leq N, \ \tilde{P}_i = \Gamma_{I_d \to G}(\bar{P}_i) = G^{\frac{1}{2}} \ \bar{P}_i \ G^{\frac{1}{2}} \tag{4.9b}$$

At this point, the next step in the original BatchNorm algorithm would be the standardization. We instead choose, amongst the possible definitions of normal distributions on $\mathcal{S}_*^+$, one which naturally involves no notion of standard deviation.

4.3.2 Statistical distribution on SPD matrices

In traditional Neural Networks (NNs), BatchNorm is defined as the centering and standardization of the data within one batch, followed by the multiplication and addition by parameterized variance and bias, to emulate the data sampling from a learnt Gaussian distribution. In order to generalize to batches of SPD matrices, we must first define the notion of Gausian density on $\mathcal{S}_*^+$. Although this definition has not yet been settled for good, several approaches have been proposed. In Jaquier and Calinon (2017), the authors proceed by introducing mean and variance as second- and fourth-order tensors. On the other hand, Said et al. (2017) derive a scalar variance. In another line of work synthesized in Barbaresco (2019), which we adopt in this work, the Gaussian density is derived from the definition of maximum entropy on exponential families using Information Geometry (IG) on the cone of SPD matrices. In this setting, the natural parameter of the resulting

exponential family is simply the Riemannian mean; in other words, this means the notion of variance, which appears in the Eucidean setting, takes no part in this definition of a Gaussian density on $\mathcal{S}_*^+$. Specifically, such a density p on SPD matrices P of dimension n writes:

$$p(P) \propto det(\alpha\ \mathfrak{G}^{-1})e^{-tr(\alpha\ \mathfrak{G}^{-1}P)}\ \text{, with } \alpha = \frac{n+1}{2} \tag{4.10}$$

In the equation above, $\mathfrak{G}$ is the Riemannian mean of the distribution. Again, there is no notion of variance: the main consequence is that a Riemannian BatchNorm on SPD matrices will only involve centering and biasing of the batch.

4.3.3 Riemannian BatchNorm algorithm

While the normalization is done on the current batch during training time, the statistics used in inference are computed as running estimations. For instance, the running mean over the training set, noted $\mathfrak{G}_{\mathcal{S}}$, is iteratively updated at each batch. In a Euclidean setting, this would amount to a weighted average between the batch mean and the current running mean, the weight being a momentum typically set to 0.9. The same concept holds for SPD matrices, but the running mean should be a Riemannian mean weighted by η, *i.e.* $\mathrm{Bar}^{(\eta,1-\eta)}(\mathfrak{G}_{\mathcal{S}}, \mathfrak{G}_{\mathcal{B}})$, which amounts to transporting the running mean towards the current batch mean by an amount $(1-\eta)$ along the geodesic. We can now write our full Riemannian BatchNorm in Algorithm 4.1.

In practice, Riemannian BatchNorm is appended after each BiMap layer in the network. Though we have now introduced the full inference phase of the algorithm, its backpropagation remains to be solved. The next section dives in the gory details of Riemannian manifold-constrained optimization, addressing the gradient derivations both for the DAMNet architecture and the Riemanian BatchNorm, as they share common ground.

4.4 Riemannian manifold-constrained optimization

The specificities of a the proposed DAMNet architecture and BatchNorm algorithm are the non-linear manipulation of manifold values in both inputs and parameters and the use of a Riemannian barycenter. Here we present the two results

Algorithm 4.1 Riemannian batch normalization on $\mathcal{S}_*^+$, training and testing phase

TRAINING PHASE

Require: **batch of** N **SPD matrices** $\{P_i\}_{i \leq N}$**, running mean** $\mathfrak{G}_\mathcal{S}$**, bias** G**, momentum** η

$\mathfrak{G}_\mathcal{B} \leftarrow \mathrm{Bar}(\{P_i\}_{i \leq N})$ // compute batch mean
$\mathfrak{G}_\mathcal{S} \leftarrow \mathrm{Bar}^\eta(\mathfrak{G}_\mathcal{S}, \mathfrak{G}_\mathcal{B})$ // update running mean
for $i \leq N$ **do**
  $\bar{P}_i \leftarrow \Gamma_{\mathfrak{G}_\mathcal{B} \to I_d}(P_i)$ // center batch
  $\tilde{P}_i \leftarrow \Gamma_{I_d \to G}(\bar{P}_i)$ // bias batch
end for
return normalized batch $\{\tilde{P}_i\}_{i \leq N}$

INFERENCE PHASE

Require: **batch of** N **SPD matrices** $\{P_i\}_{i \leq N}$**, final running mean** $\mathfrak{G}_\mathcal{S}$**, learnt bias** G

for $i \leq N$ **do**
  $\bar{P}_i \leftarrow \Gamma_{\mathfrak{G}_\mathcal{S} \to I_d}(P_i)$ // center batch using set statistics
  $\tilde{P}_i \leftarrow \Gamma_{I_d \to G}(\bar{P}_i)$ // bias batch using learnt parameter
end for
return normalized batch $\{\tilde{P}_i\}_{i \leq N}$

necessary to correctly fit the learning of the Riemannian BatchNorm in a standard backpropagation framework.

4.4.1 Learning with SPD constraint

The bias parameter matrix G of the Riemannian BatchNorm is by construction constrained to the SPD manifold. However, noting $\mathcal{L}$ the network's loss function, the usual Euclidean gradient $\frac{\partial \mathcal{L}}{\partial G}$, which we note ∂G_{eucl}, has no particular reason to respect this constraint. To enforce it, ∂G_{eucl} is projected to the tangent space of the manifold at G using the manifold's tangential projection operator $\Pi \mathcal{T}_G$, resulting in the tangential gradient ∂G_{riem}. The update is then obtained by computing the geodesic on the SPD manifold emanating from G in the direction ∂G_{riem}, using the exponential mapping Exp_G defined in Equation 2.30, or any retraction operation on the manifold, *i.e.* any operation mapping from the manifold to the tangent bundle. Both $\Pi \mathcal{T}_G$ and Exp_G are known in $\mathcal{S}_*^+$ (Yger 2013):

$$\forall P,\ \Pi \mathcal{T}_G(P) = G \frac{P + P^T}{2} G \in \mathcal{T}_G \subset \mathcal{S}^+ \tag{4.11}$$

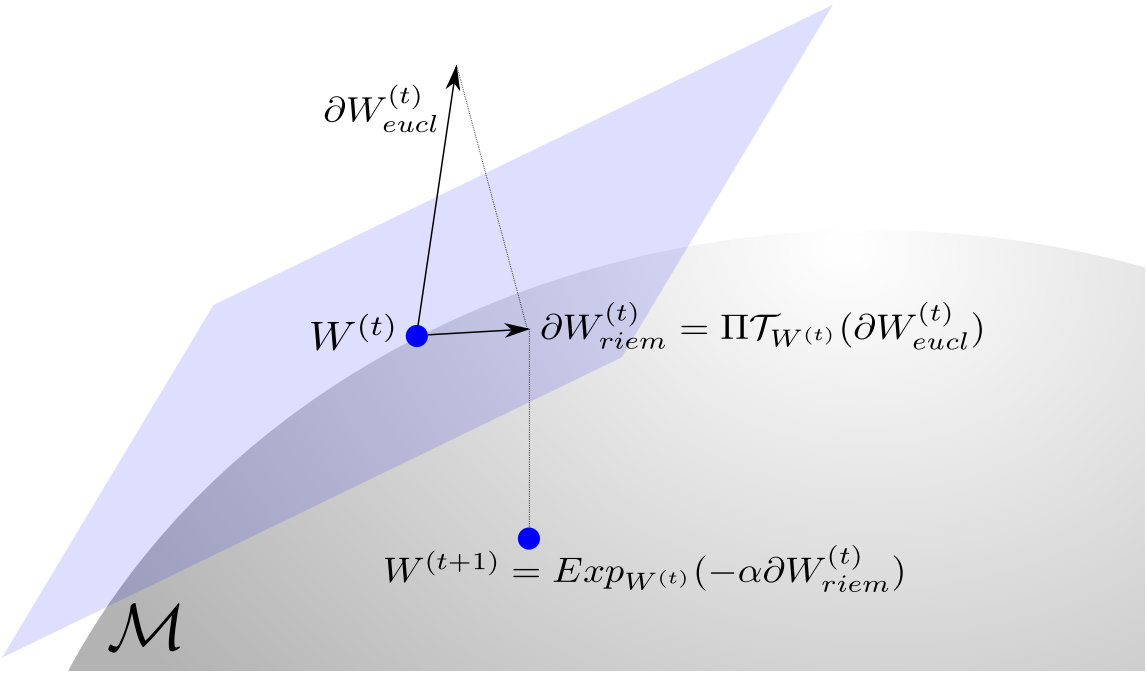

Figure 4.3.: **Illustration of manifold-constrained gradient update**. The Euclidean gradient is projected to the tangent space, then mapped to the manifold through retraction, which can be the exponential mapping.

We illustrate this two-step process in Figure 4.3, explained in detail in Edelman et al. (1998), which allows to learn the parameter in a manifold-constrained fashion. However, this is still not enough for the optimization of the layer, as the BatchNorm involves not simply G and $\mathfrak{G}$, but $G^{\frac{1}{2}}$ and $\mathfrak{G}^{-\frac{1}{2}}$, which are structured matrix functions of G, *i.e.* which act non-linearly on the matrices' eigenvalues without affecting its associated eigenspace. The next subsection deals with the backpropagation through such functions.

4.4.2 Structured matrix backpropagation

Classically, the functions involved in the chain rule are vector functions in $\mathbb{R}^n$, whereas we deal here with structured (symmetric) matrix functions in the $\mathcal{S}^+_*$, specifically the square root $(\cdot)^{\frac{1}{2}}$ for the parametric bias and the inverse square root $(\cdot)^{-\frac{1}{2}}$ for the barycenter (in Equation 4.9 and Equation 4.6).

4.4.2.1 General principles

We recall, a neural network of depth L can be seen as a hierarchical function $f = l \circ f^{(L)} \circ \ldots \circ f^{(1)}$ of input data $X^{(0)} := X$ and associated label Y composed

of L elementary blocks and a loss function l penalizing the distance of the output $X^{(L)}$ to Y. We also note $\mathcal{L}^{(k)} := l \circ f^{(L)} \circ \ldots \circ f^{(k)}$ the intermediate loss function of layer $(k-1)$. A generalization of the chain rule to $\mathcal{S}_+^*$, linking $\frac{\partial \mathcal{L}^{(k)}}{\partial P}$ to $\frac{\partial \mathcal{L}^{(k+1)}}{\partial X}$ given the forward pass $P \longmapsto X$, is thus required for the backpropagation through the Riemannian BatchNorm and the ParNorm layers to be correct. Note that a similar requirement applies to the ReEig and LogEig layers, respectively with a threshold and $\log$ function. We generically note f a monotonous non-linear function; both $(\cdot)^{\frac{1}{2}}$ and $(\cdot)^{-\frac{1}{2}}$ check out this hypothesis. To be clear, the formula for the gradients through the ReEig and LogEig layers stems for a shared, more general formula, which we then put to use in our application, *i.e.* the square root and inverse square root functions. This general formula for the gradient of f, applied on a SPD matrix' eigenvalues $(\sigma_i)_{i \leq n}$ grouped in Σ's diagonal, was independently developed by Ionescu et al. (2015) and Brodskiĭ et al. (1965).

Theorem 4.2. *Given the forward pass $f \ : \ P \longmapsto X$, the backward pass $\frac{\partial \mathcal{L}^{(k+1)}}{\partial X} \longmapsto \frac{\partial \mathcal{L}^{(k)}}{\partial P}$ is constructed as follows:*

$$\frac{\partial \mathcal{L}^{(k)}}{\partial P}(P, Y) = \mathcal{F}^*_{(k)}\left(\frac{\partial \mathcal{L}^{(k+1)}}{\partial X}(X, Y)\right) \tag{4.12}$$

In the equation, $\mathcal{F}^*_{(k)}$ is the adjoint operator to the directional derivative $(Df^{(k)})_P(dP)$ of $f^{(k)}$ at point P in direction dP. Conceptually, $\mathcal{F}_{(k)}$ is the operator mapping dP to dX.

Proof. The proof, as overviewed in Ionescu et al. (2015) and Yger and Sugiyama (2015), stems for equating variation to the loss function given a perturbation dP in P at layer (k), and the resulting perturbation dX of X at the following layer $(k+1)$.We simplify the notations by dropping the layer indices (k) and $(k+1)$ such that $\mathcal{L} := \mathcal{L}^{(k+1)}$ and using $\mathcal{L}^{(k)} = \mathcal{L} \circ f$. We retrospectively introduce $\mathcal{F} \ : \ dP \longmapsto dX$. The proof thus initiates with equating the loss variation at the two successive layers:

$$\mathcal{L} \circ f(P + dP) - \mathcal{L} \circ f(P) = \mathcal{L}(X + dX) - \mathcal{L}(X)$$

A first-order Taylor expansion then procedes; we will further omit the remainders as the method is first-order only:

$$\begin{aligned}
&\Leftrightarrow \langle \frac{\partial \mathcal{L} \circ f}{\partial P}, dP \rangle + \mathcal{O}(||dP||^2) = \langle \frac{\partial \mathcal{L}}{\partial X}, dX \rangle + \mathcal{O}(||dX||^2) && \text{(Taylor expansion)} \\
&\Leftrightarrow \langle \frac{\partial \mathcal{L} \circ f}{\partial P}, dP \rangle = \langle \frac{\partial \mathcal{L}}{\partial X}, \mathcal{F}(dP) \rangle && \text{(by definition)} \\
&\Leftrightarrow \langle \frac{\partial \mathcal{L} \circ f}{\partial P}, dP \rangle = \langle \mathcal{F}^*(\frac{\partial \mathcal{L}}{\partial X}), dP \rangle && \text{(adjoint operation)} \\
&\Leftrightarrow \frac{\partial \mathcal{L} \circ f}{\partial P} = \mathcal{F}^*(\frac{\partial \mathcal{L}}{\partial X}) && \text{(inner product valid } \forall\, dP\text{)}
\end{aligned}$$

■

In the Euclidean case of Equation 4.12, $\mathcal{F}^*_{(k)}$ would typically reduce to the standard linear form of the chain rule. We now show a practical usage of this generalized chain rule, through the demonstration of a central example: the non-linear eigenvalue modification, generically noted:

$$g \circ \text{Eig} \ : \ P \longmapsto (U, \Sigma, U^T) \longmapsto U g(\Sigma) U^T =: X \tag{4.13}$$

Equation 4.13 maps P to its eigenspace U and eigenvalues Σ, then to the non-linear (monotonous) modification of the eigenvalues. By construction, $U \in \mathcal{O}$ (where we reuse the notation $\mathcal{O}$ for orthogonal matrices), and Σ is diagonal. We decompose the derivation of $\frac{\partial \mathcal{L} \circ g \circ \text{Eig}}{\partial P}$ through $\frac{\partial \mathcal{L} \circ \circ \text{Eig}}{\partial P}$ and $\frac{\partial \mathcal{L} \circ g}{\partial (U, \Sigma)}$ We begin with Lemma 4.3.

Lemma 4.3. *The tangent plane $\mathcal{T}_U \mathcal{O}$ of $U \in \mathcal{O}$ is $U \times \mathcal{S}_\dagger$ (where $\mathcal{S}_\dagger$ is the space of skew-symmetric matrices). In other words, $dU = U\Omega$ with $\Omega^T = -\Omega$.*

Proof.

$$\begin{aligned}
U^T U = I_d \ &\Rightarrow U^T Z + Z^T U = 0 && \text{(noting } Z \coloneqq dU\text{)} \\
&\Rightarrow \Omega \in \mathcal{S}_\dagger && \text{(noting } \Omega \coloneqq U^T Z\text{)} \\
&\Rightarrow dU = U\Omega
\end{aligned}$$

■

In practice, making use of Theorem 4.2 amounts for the Eig function to finding $d\Sigma$ and dU in function of dP, given the differential $\mathcal{F}$ form of dP, which we do in Lemma 4.4:

Lemma 4.4. *The function* $\mathrm{Eig} \ : \ P \longmapsto (U, \Sigma, U^T) \mid P = U\Sigma U^T$ *admits the following differential form:*

$$\begin{cases} d\Sigma = (U^T dP\, U)_{diag} & \textit{(using the operation } (\cdot)_{diag} \textit{ which zeroes all off-diagonal elements)} \\ dU = U(K^T \odot (U^T dP\, U)) & \textit{(with } K_{ij} = \begin{cases} \frac{1}{\sigma_i - \sigma_j} & (i \neq j) \\ 0 & (i = j) \end{cases}\textit{)} \end{cases} \tag{4.14}$$

Proof.

$$\begin{aligned} P = U\Sigma U^T \ &\Rightarrow dP = dU\Sigma U^T + U d\Sigma U^T + U\Sigma dU^T \\ &\Rightarrow U^T dP\, U = (U^T dU)\Sigma + d\Sigma + \Sigma(U^T dU)^T \end{aligned}$$

In the above equation, we know from the proof of Lemma 4.3 that $U^T dU$ is skew-symmetric; thus, its diagonal is zero, and therefore its multiplication with diagonal matrix Σ yields a zero-diagonal matrix. Furthermore, $d\Sigma$ is also diagonal by construction, so in the end we have $d\Sigma = 0 + (U^T dP\, U)_{diag} + 0$. Continuing the equalities, we have:

$$\begin{aligned} &\Rightarrow X = (U^T dU)\Sigma + A_{diag} - \Sigma(U^T dU) && \text{(with } A := U^T dP\, U\text{)} \\ &\Rightarrow \tilde{A} = (U^T dU)\Sigma - \Sigma(U^T dU) && \text{(with } \tilde{A} := A - A_{diag}\text{)} \\ &\Rightarrow \begin{cases} (U^T dU)_{ij}\sigma_j - (U^T dU)_{ij}\sigma_i = \tilde{A}_{ij} & (i \neq j) \\ (U^T dU)_{ij} = 0 & (i = j) \end{cases} \\ &\Rightarrow \begin{cases} (U^T dU)_{ij} = \frac{1}{\sigma_j - \sigma_i}\tilde{A}_{ij} & (i \neq j) \\ (U^T dU)_{ij} = 0 & (i = j) \end{cases} \\ &\Rightarrow U^T dU = K^T \odot \tilde{A} \\ &\Rightarrow dU = U(K^T \odot (U^T dP\, U)) && \text{(because } K^T \odot A_{diag} = 0\text{)} \end{aligned}$$

■

Finally, we show the backward pass of the Eig function in Theorem 4.3; all we need now given Lemma 4.4 is to compute the adjoint operator:

Theorem 4.3. *The function* $\mathrm{Eig} \ : \ P \longmapsto (U, \Sigma, U^T) \mid P = U\Sigma U^T$ *admits the following backward pass:*

$$\frac{\partial \mathcal{L} \circ \mathrm{Eig}}{\partial P} = U\left((K^T \odot (U^T \frac{\partial \mathcal{L}}{\partial U})) + (\frac{\partial \mathcal{L}}{\partial \Sigma})_{diag}\right)U^T \quad \textit{(with } K_{ij} = \begin{cases} \frac{1}{\sigma_j - \sigma_i} & (i \neq j) \\ 0 & (i = j) \end{cases}\textit{)} \tag{4.15}$$

Proof. To compute the adjoint aoperator, only a few linear algebra tricks are involved, which we bunch in one single line:

$$\begin{aligned}
\langle \frac{\partial \mathcal{L}}{\partial U}, dU\rangle + \langle \frac{\partial \mathcal{L}}{\partial \Sigma}, d\Sigma\rangle &= \langle \frac{\partial \mathcal{L}}{\partial U}, U(K^T \odot (U^T dP\, U))\rangle + \langle \frac{\partial \mathcal{L}}{\partial \Sigma}, (U^T dP\, U)_{diag}\rangle \\
&= \langle U(K^T \odot (U^T \frac{\partial \mathcal{L}}{\partial U}))U^T, dP\rangle + \langle U(\frac{\partial \mathcal{L}}{\partial \Sigma})_{diag} U^T, dP\rangle \\
&= \langle U(K^T \odot (U^T \frac{\partial \mathcal{L}}{\partial U}))U^T + U(\frac{\partial \mathcal{L}}{\partial \Sigma})_{diag} U^T, dP\rangle \\
\Rightarrow \frac{\partial \mathcal{L} \circ \mathrm{Eig}}{\partial P} &= U\left((K^T \odot (U^T \frac{\partial \mathcal{L}}{\partial U})) + (\frac{\partial \mathcal{L}}{\partial \Sigma})_{diag}\right)U^T
\end{aligned}$$

■

We now derive in Theorem 4.4 the backward pass for the eigenvalue modification $g \ : \ (U, \Sigma) \longmapsto X$:

Theorem 4.4. *The function* $\mathrm{Eig} \ : \ P \longmapsto (U, \Sigma, U^T) \mid P = U\Sigma U^T$ *admits the following backward pass:*

Proof.

$$\begin{aligned}
X = Ug(\Sigma)U^T \Rightarrow dX &= dUg(\Sigma)U^T + Ug'(\Sigma)d\Sigma U^T + Ug(\Sigma)dU^T \\
\Rightarrow dX &= \overbrace{(dUg(\Sigma)U^T)_{sym}}^{M_{sym} := \frac{M+M^T}{2}} + Ug'(\Sigma)d\Sigma U^T
\end{aligned}$$

Given this newfound differential form, we now compute its adjoint; similarly to before, we use the inner product adjoint property:

$$\begin{aligned}
\langle \frac{\partial \mathcal{L}}{\partial X}, dX \rangle &= \langle \frac{\partial \mathcal{L}}{\partial X}, (dU g(\Sigma) U^T)_{sym} + U g'(\Sigma) d\Sigma U^T \rangle \\
&= \langle g'(\Sigma) U^T \frac{\partial \mathcal{L}}{\partial X} U, d\Sigma \rangle + \langle 2(\frac{\partial \mathcal{L}}{\partial X})_{sym} U g(\Sigma) \rangle \\
\Rightarrow & \begin{cases} \frac{\partial \mathcal{L} \circ g}{\partial \Sigma} & = 2(\frac{\partial \mathcal{L}}{\partial X}))_{sym} U g(\Sigma) \\ \frac{\partial \mathcal{L} \circ g}{\partial U} & = g'(\Sigma) U^T (\frac{\partial \mathcal{L}}{\partial X}))U \end{cases}
\end{aligned}$$

■

The investigative reader may refer to other established references of matrix computation and calculus for a deeper understanding of the matter, such as Michal (1947), Bers (1948), Dwyer and Macphail (1948), Papadopoulo and Lourakis (2000), Petersen et al. (2006), Giles (2008), and Magnus and Neudecker (2019)

4.4.2.2 Takeaway backpropagation formulas

We now summarize the findings above in one simple formula, obtained by chaining the backward passes of Eig and g (we omit the calculations). In short: given the function $P \longmapsto X := g(P)$ and the succeeding gradient $\frac{\partial L^{(k+1)}}{\partial X}$, the output gradient $\frac{\partial L^{(k)}}{\partial P}$ is:

$$\frac{\partial L^{(k)}}{\partial P} = U \left(L \odot (U^T (\frac{\partial L^{(k+1)}}{\partial X}) U) \right) U^T \tag{4.16}$$

The equation above, also decribed in Nielsen and Bhatia (2013), is called the Daleckĭi-Kreĭn formula and dates back to 1956, (but was translated from Russian 9 years later), predating the other formulation by 60 years. It involves the eigenspace U of the input matrix P, and the Loewner matrix L, or finite difference matrix defined by:

$$L_{ij} = \begin{cases} \frac{g(\sigma_i) - g(\sigma_j)}{\sigma_i - \sigma_j} & \text{if } \sigma_i \neq \sigma_j \\ g'(\sigma_i) & \text{otherwise} \end{cases} \tag{4.17}$$

In the case at hand, $\left((\cdot)^{-\frac{1}{2}} \right)' = -\frac{1}{2} (\cdot)^{-\frac{3}{2}}$ and $\left((\cdot)^{\frac{1}{2}} \right)' = \frac{1}{2} (\cdot)^{-\frac{1}{2}}$. We credit Engin et al. (2018) for first showing the equivalence between the two cited formulations, of which we expose the most concise.

In summary, the Riemannian barycenter (approximation via the Karcher flow for a batch of matrices, or exact formulation for two matrices), the parallel transport and its extension on the SPD manifold, the SPD-constrained gradient descent and the derivation of a non-linear SPD-valued structured function's gradient allow for training and inference of the proposed Riemannian BatchNorm algorithm.

4.5 Convolution for covariance time series

Here we introduce a novel temporal convolution layer for time series of SPD matrices, inspired from the usual Euclidean 1D convolution layer, and using the BiMap layer defined above. This layer produced no satisfactory results, so does not appear in the experimental validations.

4.5.1 Single-channel convolution

Input P is a time series of SPD matrices of dimension n_i and of duration T_i, *i.e.* an input of shape (T_i, n_i, n_i). The convolution parameter K is a kernel of length k of BiMap operators of shape (n_i, n_o), i.e. a block of shape (k, n_i, n_o). The output X of the layer, of duration T_o and overal shape (T_o, n_o, n_o), is computed as a valid temporal convolution using the bilinear mapping as base operation:

$$\begin{aligned} &X \coloneqq P * K \\ &\forall t \leq T_o,\ X_t = \frac{1}{k} \sum_{l \leq k} K_l^T\ P_{t+k-1-l}\ K_l \end{aligned} \tag{4.18}$$

Stride and padding can also be used to influence the duration T_o of the output signal.

4.5.2 Weighted average convolution

A key difference between the Euclidean 1D convolution and the one defined above is the BiMap operator preserves the matrix norm (up to normalization by the dimension), which gives the same importance to all the SPD "pixels" in the time series; essentially, the convolution is valid, but weightless. Therefore, we

introduce weights w of shape $(k,)$ to the convolution, constrained to represent a valid distribution, i.e. $\forall l \leq k,\ w_l \geq 0$ and $\sum_{l \leq k} w_l = 1$. The convolution then becomes:

$$\begin{aligned} &X := P * {}^{w} K \\ &\forall t \leq T_o,\ X_t = \sum_{l \leq k} w_l\ K_l^T\ P_{t+k-1-l}\ K_l \end{aligned} \tag{4.19}$$

4.5.3 Riemannian convolution using the weighted Fréchet mean

The convolution as defined above takes the form of an arithmetic mean; one could instead imagine using a Riemannian mean, better-suited to the SPD data:

$$\begin{aligned} &X := P * {}^{w} K \\ &\forall t \leq T_o,\ X_t = \mathrm{Bar}^{w}_{l \leq k}\ (K_l^T\ P_{t+k-1-l}\ K_l) \end{aligned} \tag{4.20}$$

In the equation above, one must take care not to confuse the weighted Fréchet mean of the operands with the unweighted Fréchet mean of the weighted operands.

Figure 4.4 illustrates the somewhat convoluted process of performing the SPD BiMap convolution.

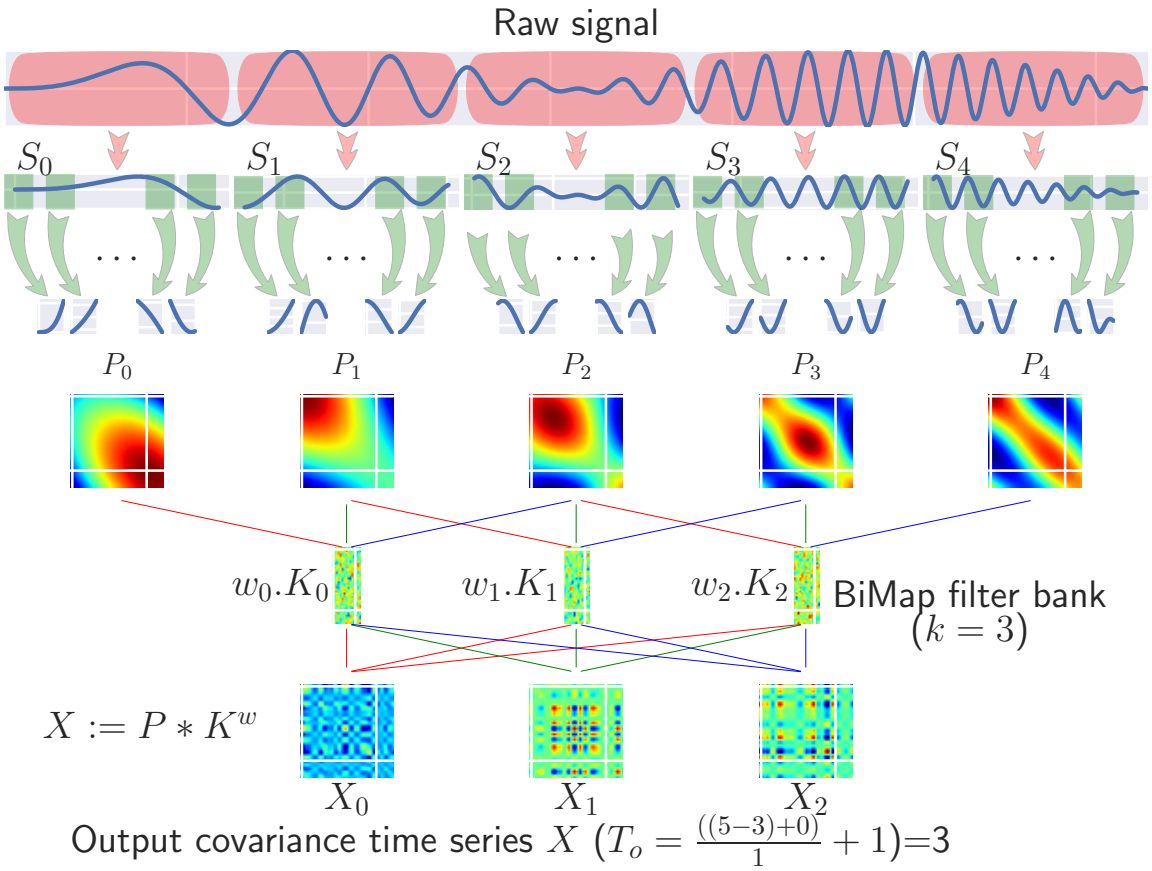

Figure 4.4.: **The convolutional BiMap.** The input raw signal is split in T_i sub-signals corresponding to a first level of temporal resolution. Each sub-signal is then split in sub-sub-signals of elementary duration τ (from Section 3.2.2). From there, a covariance matrix is sampled for each timestep $t \leq T_i$, yielding a time series of covariances, which is convolved with the temporal BiMap filter bank. The output duration T_o follows the standard rule of convolutions, here with no stride and padding.

4.5.4 Multi-channel convolution

More realistically, we wish to instigate the network's potential expressiveness by further building multi-channeled SPD representations. We do so in the usual fashion; input P is now of shape (T_i, C_i, n_i, n_i), kernel K of shape (C_o, C_i, k, n_i, n_o), weights w are still of shape $(k,)$, and output X of shape (T_o, C_o, n_o, n_o), and is computed as follows:

$$\forall c_o \leq C_o,\ X_{c_o} = \sum_{c_i \leq C_i} P_{c_i} * {}^{w}K_{c_o, c_i} \tag{4.21}$$

In all definitions above, the convolution uses the BiMap layer (and therefore induces dimension reduction), which is novel.

4.6 Experimental validation

Here we evaluate the gain in performance of the SPDNetBN against the baseline SPDNet on different tasks: micro-Doppler (μ-D) radar data classification, emotion recognition from video, and action recognition from MoCap data. We call the depth L of an SPDNet the number of BiMap layers in the network, and denote the dimensions as $\{n_0, \cdots, n_L\}$. The vectorized input to the final classification layer is thus of length n_L^2. To be precise, the symmetry of the matrix allows to gather a vector of length $\frac{n(n+1)}{2}$, but our experiments seemed to slightly favor using the whole matrix. All networks are trained for 200 epochs using Stochastic Gradient Descent (SGD) with momentum set to 0.9 with a batch size of 30 and learning rate $5e^{-3}$, $1e^{-2}$ or $5e^{-2}$. We provide the data in a pre-processed form alongside the PyTorch (Paszke et al. 2017) code for reproducibility purposes, and also show snippets to demonstrate its ease of use. We recall that the SPDNetBN appends a Riemannian BatchNorm after each BiMap layer of the associated SPDNet. Finally, we also report performances of shallow learning method on SPD data, namely the Minimum Riemannian Distance to Riemannian Mean (MRDRM) scheme presented in Section 2.5.2, in order to bring elements of comparison between shallow and deep learning on SPD data.

4.6.1 Drones recognition

Our first experimental target focuses on drone μ-D radar classification. First we validate the usage of our proposed method over a baseline SPDNet, and also compare to the previously exposed deep learning methods. Then, we study the models' robustness to lack of data, a challenge which, as stated previously, plagues the task of radar classification and also a lot of different tasks. Experiments are conducted on a confidential dataset of real recordings issued from the North Atlantic Treaty Organization (NATO). To spur reproducibility, we also experiment on synthetic, publicly available data.

Radar data description As previously described, the radar signal is the result of an emitted wave reflected on a target; as such, one data point is a time series of

N values, which can be considered as multiple realizations of a locally stationary centered Gaussian process, as done in Charon and Barbaresco (2009). The signal is split in windows of length $n = 20$, the series of which a single covariance matrix of size 20×20 is sampled from, which represents one radar data point. We refer to Figure 3.10 for a visual clarification of the splitting operation. The NATO data features 10 classes of drones, whereas the synthetic data is generated by a realistic simulator of 3 different classes of drones following the protocol previously described. We chose here to mimick the real dataset's configuration, *i.e.* we consider a couple of minutes of continuous recordings per class, which correspond to 500 data points per class, so 1500 in total.

Comparison of Riemannian and Euclidean models We test the SPD-based models in a $\{20, 16, 8\}$, 2-layer configuration for the synthetic data, and in a $\{20, 16, 14, 12, 10, 8\}$, 5-layer configuration for the NATO data, over a 5-fold cross-validation, split in a train-test of $75\% - 25\%$. We also wish to compare the Riemannian models to the common Euclidean ones, which consitute our state-of-the-art in μ-D classification. We compare two Fully Temporal Convolutional Networks (FTCNs): the first one is the same as in Section 3.2.2; for the second one, the number of parameters is set to approximately the same number as for the SPD neural nets, which amounts to an unusually small Deep Neural Network (DNN). This second "deep" architecture is chosen to ensure fairness from the viewpoint of parameter intesity. All in all, the SPD neural models and the small FTCN on the one hand, and the full-size FTCN on the other hand respectively have approximately 500 and 10000 parameters.

Table 4.1 reports the average accuracies and variances on the NATO data of the SPDNet against the Euclidean models, and the MRDRM scheme. A first obvious result, is that the MRDRM performs relatively poorly, albeit showcasing great stability. Secondly, the Euclidean method outperforms both Riemannian methods by a large margin. However, the Euclidean method crumbles when given ten times less data to learn upon, while the Riemanian method remains quite stable, to the point of reversing the tendency. This robustness to lack of data, already observed in the previous chapter, will recur in the following experiments, providing a strong argument in favour of the Riemannian methods.

Table 4.1.: **Performance of SPDNet, FTCN and MRDRM on the NATO dataset**. Accuracy is compared on a 5-fold cross-validation; experiments are repeated with only 10% of training data.

Model	SPDNet	MRDRM	FTCN	
# Parameters	~ 500	-	~ 500	~ 10000
Acc. (all data)	$72.6\% \pm 0.61$	$69.7\% \pm 1.12$	$73.4\% \pm 3.66$	$\mathbf{88.7}\% \pm 0.83$
Acc. (10% data)	$\mathbf{69.1}\% \pm 0.97$	$67.1\% \pm 2.17$	$61.1\% \pm 3.50$	$65.6\% \pm 2.74$

Table 4.2 reports the average accuracies and variances of the proposed DAMNet and SPDNetBN architectures, compared with the original SPDNet. We observe from these results a strong gain in performance on the SPDNetBN and DAMNets over the SPDNet and over the small FTCN, which validates the usage of the Riemannian BatchNorm along with the exploitation of the geometric structure underlying the data. All in all, we reach better performance with much fewer parameters, which again is a key feature for radar classification, and much others.

Table 4.2.: **Performance of SPDNet, DAMNet and SPDNetBN on the NATO dataset**. Accuracy is compared on a 5-fold cross-validation; experiments are repeated with only 10% of training data.

Model	SPDNet	DAMNet		SPDNetBN
Normalization	-	BarNorm	ParNorm	BatchNorm
Accuracy	$72.6\% \pm 0.61$	$79.9\% \pm 1.19$	$80.3\% \pm 0.55$	$\mathbf{82.3}\% \pm 0.80$
Acc. (10% data)	$69.1\% \pm 0.97$	$73.8\% \pm 0.25$	$70.2\% \pm 1.74$	$\mathbf{77.7}\% \pm 0.95$

The raw SPDNet, though not competing with the FTCN when all data is available, remains more robust to the lack data. The SPDNetBN on the other hand, exhibits a strong gain in performance to the point of competing with the deep model. Note that we iterate only once for the barycenter estimation in the BarNorm or BatchNorm ($K = 1$ in the Karcher flow). This is most likely due to the fact, that within the normalization of a batch, a noisy estimate is probably preferable as the estimation itself is restricted to a small sample size. This hypothesis was validated in experiments we do not include in the text: in the end, using the single iteration yielded better results than more iterations, and no iteration (*i.e.*, the arithmetic mean).

Finally, in the interest of convergence analysis, we also report learning curves for the model's accuracy with and without Riemannian BatchNorm in Figure 4.5.

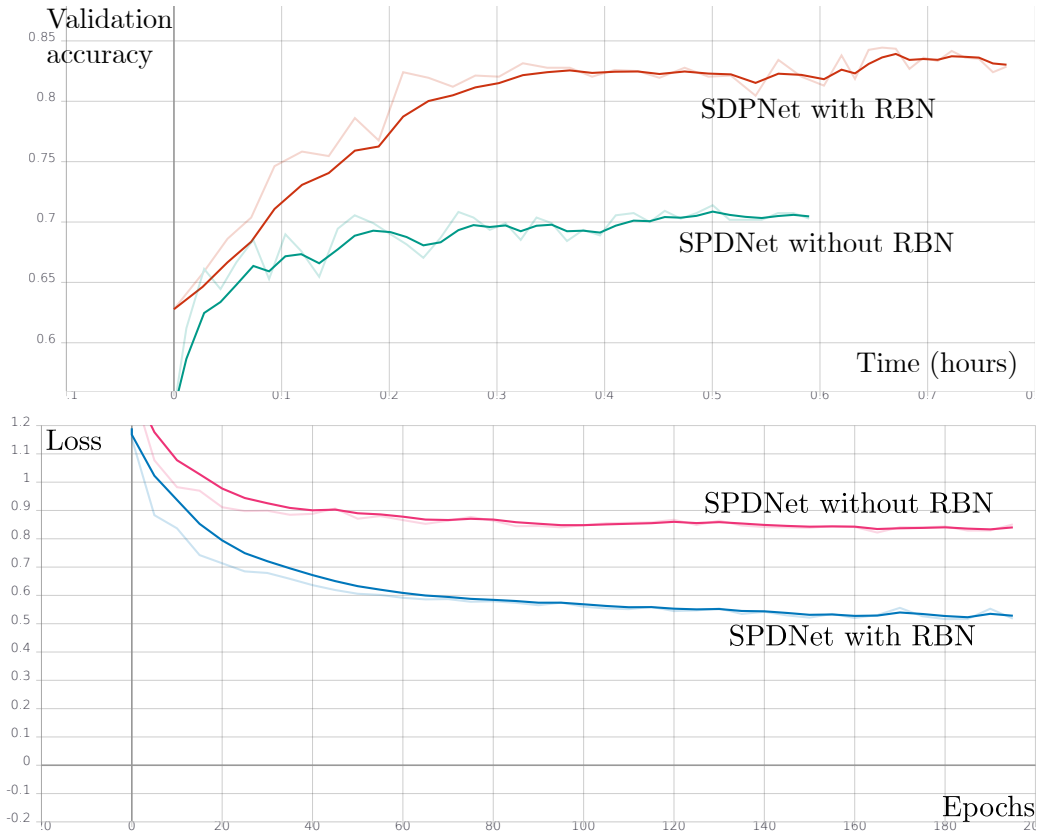

Figure 4.5.: **Validation accuracy and loss curves of SPDNet and SPDNetBN on the NATO dataset**. The x-axis measures hours instead of epochs for the accuracy, epochs for the loss.

In Figure 4.5, the SPDNetBN exhibits a steeper learning curve. For the same number of epochs, it does take more time overall, but reaches better accuracy much faster, allowing to reduce the number of epochs. A similar results is found for the BarNorm regularization, while the ParNorm yields an overall less stable curve, which is a reasonable behaviour condsidering the additional SPD parameter to be learnt, without the benefit of normalizing around the barycenter first.

For the sake of expression, we also show in Figure 4.6 a confusion matrix of the SPDNetBN model for the NATO dataset. Notice the strong imbalance throughout the classes precisions, due to the same imbalance found within the class samples. A thorough description of the NATO dataset is geiven in Appendix B.

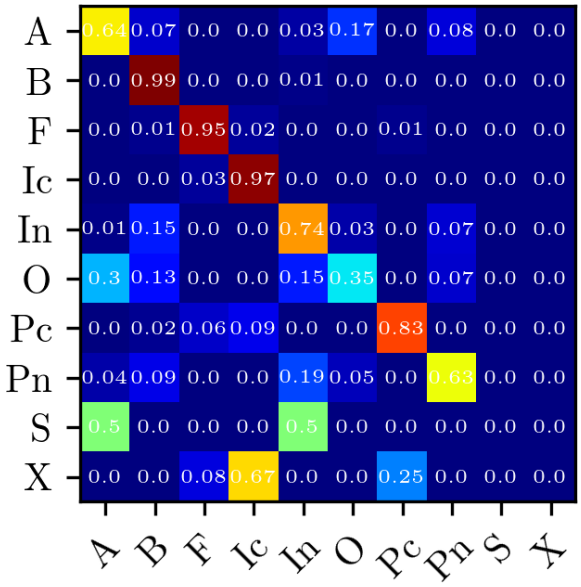

Figure 4.6.: **Confusion matrix on the NATO dataset**. Class imbalance within the dataset yields rather strongly imbalanced results.

Robustness to lack of data As stated previously, it is of great interest to consider the robustness of learning algorithms when faced with a critically low amount of data. The last line in both Table 4.1 and Table 4.2 shows that when given only 10% of available training data, the SPD-based models remain highly robust to the lack of data while the FTCNs plummet. Further, we study robustness on synthetic data, artificially varying the amount of training data while comparing performance over the same test set. As the simulator is unbounded on potential training data, we also increase the initial training set up to double its original size. Results are reported in Figure 4.7.

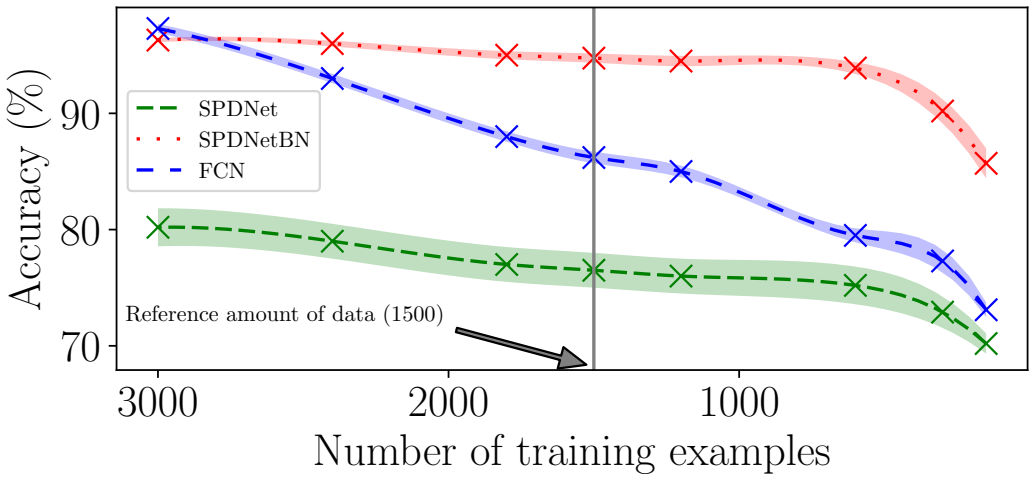

Figure 4.7.: **Performance of all models in function of the amount of synthetic radar data**. The SPDNetBN model outperforms the other ones and continues to work even with a little fraction of the train data.

We can conclude from these that the SPDNetBN both exhibits higher robustness to lack of data and performs much better than the Euclidean deep methods with much fewer parameters. When the available training data allowed skyrockets, we do observe that the FTCN comes back to par with the SPDNetBN to the point of outperforming it by a small margin in the extremal scenario; in the meantime, the SPDNet lags behind by a large margin to the SPDNetBN, which thus seems to benefit strongly from the normalization. In any case, the manifold framework seems well suited in a scarce data learning context, especially considering the introduced normalization layers, which again pinpoints the interest of taking into account the geometric structure of the data, all the while without introducing specific prior knowledge during training.

SPD Data separability To study how the reference projection point influences the separability of datapoints, we choose two arbitrary classes c_1 and c_2 in the radar dataset; we map all points from the test set belonging to c_1 and c_2 to the final SPD representation layer of a pre-trained SPDNet; we compute the barycenter $\mathfrak{G}$ of all mapped points, and the discretized geodesic (Pennec et al. 2006), parameterized by $t \in [0, 1]$ linking the identity matrix I_d to $\mathfrak{G}$; we also further extrapolate the geodesic for better visualisation; we compute the Euclidean projections of the mapped SPD points at all reference points along the geodesic; $t = 0$ corresponds to LogEig, and $t = 1$ to the BarNorm; for one given projection reference point, we evaluate class separability as the Euclidean distance between the Euclidean class means of the projected points; separability is evaluated at each reference point along the geodesic; our hope is to observe high separability at the barycenter (*i.e.* at $t = 1$).

1. We choose two arbitrary classes c_1 and c_2;
2. We map all points from the test set belonging to c_1 and c_2 to the final SPD representation layer of a pre-trained SPDNet;
3. We compute the barycenter $\mathfrak{G}$ of all mapped points, and the discretized geodesic, parameterized by $t \in [0, 1]$ linking the identity matrix I_d to $\mathfrak{G}$; we also further extrapolate the geodesic for better visualisation;
4. We compute the Euclidean projections of the mapped SPD points at all reference points along the geodesic; $t = 0$ corresponds to LogEig, and $t = 1$ to the BarNorm;
5. For one given projection reference point, we evaluate class separability as the Euclidean distance between the Euclidean class means of the projected points;
6. Separability is evaluated at each reference point along the geodesic; our hope is to observe high separability at $\mathfrak{G}$ (*i.e.* at $t = 1$).

The red curve in Figure 4.8 measures the distance between the two classes: we see it increasing along the geodesic. Note it is not obvious (though it is intuitive) that the barycenter criterion should provide this result, as class separability on the manifold does not necessarily translate to the tangent space; in the case of $\mathcal{S}_+^*$ however, negative curvature actually ensures this. This increase experimentally validates the interest of the batch centering BarNorm scheme.

For a more extensive visual study, we also shoot geodesics in random directions around the identity ($t = 0$, green curves), and around the barycenter $\mathfrak{G}$ ($t = 1$, blue curves), and also plot the corresponding class separability at each reference point along each random geodesic. We observe that the neighborhood of $\mathfrak{G}$ seems to constitute local maximum in class separability. This observation validates the practical relevance of information geometric theoretical elements involved in our normalization schemes in the development of manifold-valued neural networks.

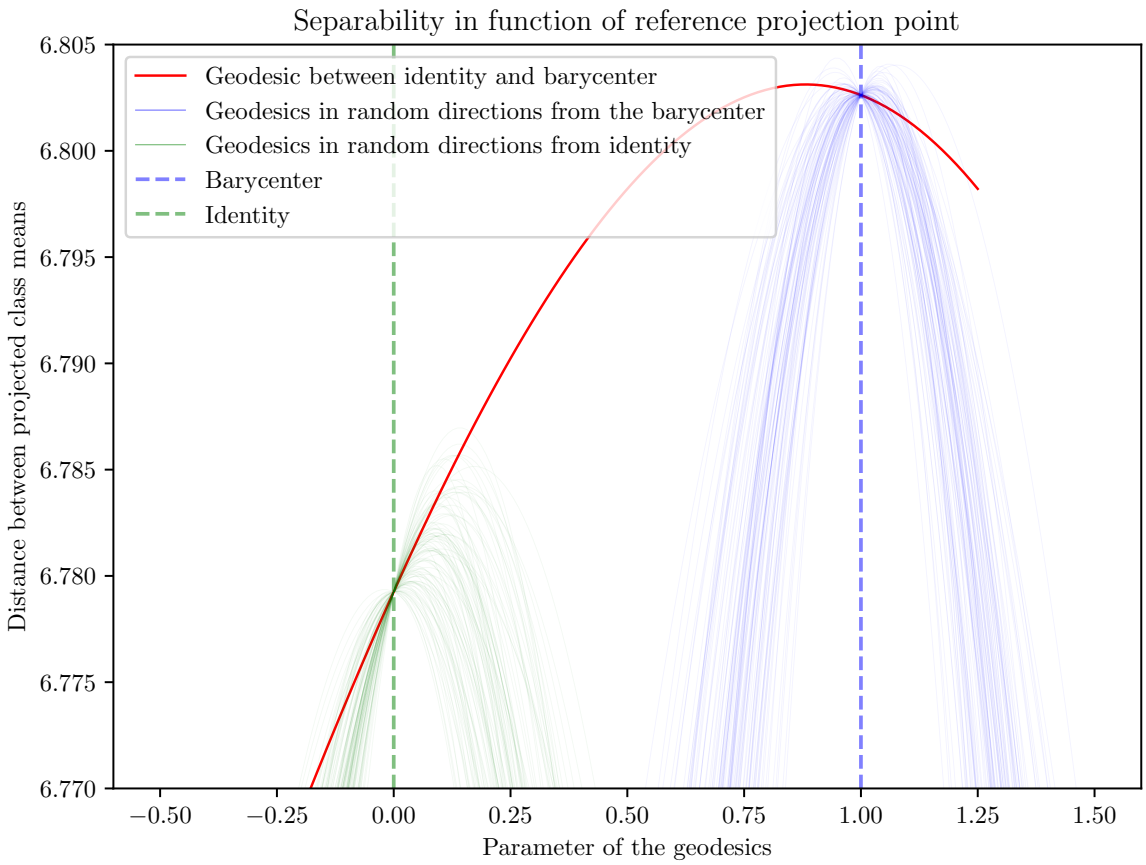

Figure 4.8.: Separability of two classes of the radar dataset. Separation between classes is higher with a reference matrix near the barycenter.

Interest of the Riemannian barycenter Within the BarNorm scheme and the Riemannian BatchNorm, the Fréchet mean is computed using the Karcher flow (Algorithm 2.2). Actually, we run one single step of the Karcher flow, which when the learning rate is set to 1 amounts to the barycenter from the LEM viewpoint (Section 4.2.1). Conceptually, step 0 of the flow, *i.e.* the initialization, amounts to the Euclidean barycenter, or arithmetic average. One could question the efficiency of using the Karcher flow, which involves eigenvalue decompositions, in terms of the network's final performance.

The first rebuttal, is that within the SPD networks, the decomposition is already performed, and the results are stored for the backpropagation step. The second counter-argument lies within the following experiment. We execute the Minimum Distance to Mean (MDM) classification scheme, successively using the three possible barycenters (arithmetic mean, LEM, Fréchet with several steps) on the synthetic radar data. Since the three candidates instantiate as iterations through the Karcher flow, we plot in Figure 4.9 a single curve of validation accuracy.

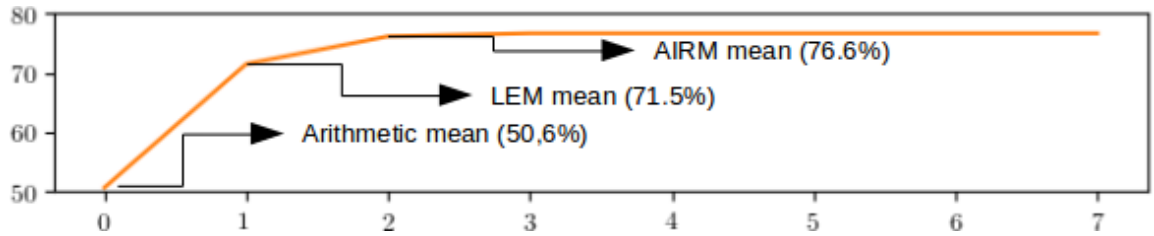

Figure 4.9.: Validation accuracy for the closest barycenter algorithm. Iteration 0 corresponds to the arithmetic mean, iteration 1 to the LEM barycenter.

Figure 4.9 shows performance increasing with the fidelity to the manifold's geometry, justifying the interest of the Fréchet mean. However, the computational overhead induced from multiple iterations may not be worth the gain in accuracy depending on the application. Furthermore, we found that adding iterations of the Karcher flow in the batch centering scheme actually deteriorates performance. This could be explained by the fact that one single batch may not adequately represent the underlying data distribution, thus a theoretically less accurate batch barycenter may reduce the variance in the reference point computation and ease training.

We now move on to validating the DAMNet and SPDNetBN on a broader set of tasks.

4.6.2 Emotion recognition

In this section we experiment on the Acted Faces Expressions in the Wild (AFEW) dataset (Dhall et al. 2011), which consists of videos depicting 7 classes of emotions; we follow the setup and protocol in Huang and Van Gool (2017). The dataset consists of 2118 SPD matrices of size 400×400 split in 1747+371 training and validation sets, each matrix in the dataset summarizing the pixel cross-correlations of a video clip scaled down to frames of size 20×20. Since Huang and Van Gool (2017) perform no cross-validation and use a fixed seed for initialization, we also initialize with the same exact weights, as we observed a rather high variance in accuracy for different random initializations. We train 1-, 2-, 3- and 4-layer networks; results are summarized in Table 4.3. In comparison, the MRDRM yields a 20.5% accuracy.

Table 4.3.: **Performance of SPDNet, DAMNet and SPDNetBN on the AFEW dataset**. As in previous works, accuracy is measured on a held-out test set.

Model architecture	SPDNet	DAMNet		SPDNetBN
Normalization	-	BarNorm	ParNorm	BatchNorm
$\{400, 50\}$	29.9%	32.1%	32.6%	**34.9**%
$\{400, 100, 50\}$	31.2%	33.2%	31.8%	**35.2**%
$\{400, 200, 100, 50\}$	34.5%	36.1%	35.0%	**36.2**%
$\{400, 300, 200, 100, 50\}$	33.7%	34.3%	36.3%	**37.1**%

We first clarify we do not necessarily seek state-of-the-art in the general sense for the following tasks, but rather in the specific case of the family of SPD-based methods. Our own implementation's performances closely match that those cited in Huang and Van Gool (2017), ensuring a fair comparison. We observe a small yet consistent improvement using barycentric centering of the data. It also seems the improvement is higher the fewer the layers. This could be attributed to the fact the network learnt a more discriminative SPD representation in deeper layers, which then benefits less from centering. We also observe a consistent improvement using the Riemanian BatchNorm, with systematically better scores, although the BarNorm rivals it closely in certain scenarios in the higher end of deepness spectrum. Again, we blame the additional parameters to be learnt, in the ParNorm or BatchNorm This dataset being our largest-scale experiment, we also report the increase in computation time using the Riemannian BatchNorm, specifically for the deepest net: one full training lasted on average $81s$ for SPDNet, and $88s$ ($+8.6\%$) for SPDNetBN. Computational complexity inherent to SPD-based methods is counterbalanced by having much fewer parameters to learn than an usual deep net.

4.6.3 Action recognition

In this section we experiment on the Hochschule der Medien 05 (HDM05) (Müller et al. 2007) dataset, which consists in MoCap data depicting various actions performed by 5 human actors. The actions are divided in 14 main action classes, which are subdivided in 130 classes. We perform the experiments in the 130-class setting. One data point is a sequence of m body states evolving through time, one body state being a frame of 3-D coordinates of the $n_{joints} = 31$ joints constituting the body. For each body state, we consider the $n = 3n_{joints} = 93$-dimensional feature vector concatenating all coordinates; the whole sequence is then described by its centered $n \times n$ joint covariance, ie the summed centered covariances of each frame feature vector. The resulting matrix can in theory be SPD *i.f.f.* $n \leq m$. The database contains 2337 sequences. Some of the 130 classes being vastly under-

represented, we select those with at least 5 sequences yielding SPD descriptors (as done in Harandi et al. (2014)), which trims the dataset down to 2086 points scattered throughout 117 classes. Figure 4.10 plots the distribution of sequence lengths across the retained classes (minimum, mean and maximum) along with the instances per class. Note that by construction the minimal sequence length possible is 93 and the smallest number of instances is 5.

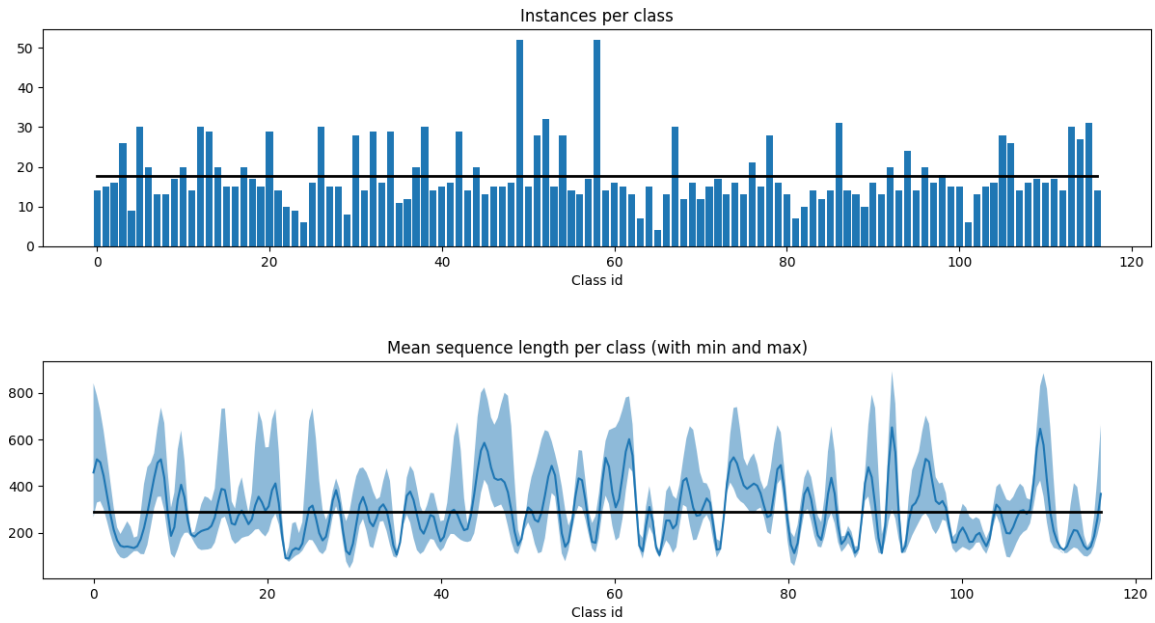

Figure 4.10.: **Distribution of class instance number and sequence length for the HDM05 dataset**. Visibly, there are less instances per class than there are classes, and some sequences are too short to yield positive definite covariance.

As in Huang and Van Gool (2017), we train a $\{93, 30\}$ SPDNet on half the data and test on the other half with 10-fold cross-validation; results are shown in Table 4.4. Note that all tested models exhibit noticeable variance depending on the weights initialization and the initial random split of the dataset; the results displayed were obtained by setting a fixed seed of 0 for both. In comparison, the MRDRM yields a $27.3\% \pm 1.06$ accuracy.

Table 4.4.: Accuracy comparison of SPDNets and DAMNets on the HDM05 dataset.

Model	SPDNet	DAMNet		SPDNetBN
Norm.	-	BarNorm	ParNorm	BatchNorm
Acc.	$61.6\% \pm 1.35$	$64.2\% \pm 1.10$	$63.5\% \pm 0.98$	$\mathbf{65.2}\% \pm 1.15$

Again, we validate a better performance using the proposed normalization schemes, the Riemanian BatchNorm outperforming the other models.

4.7 Conclusion

This chapter focused on the introduction of novel layers in the context of deep architectures for SPD matrices. Making use of the geometry of the underlying manifold, we presented two main levels of contributions.

We first proposed a Data-Aware Mapping Network (DAMNet), a new architecture with two mutually exclusive layers which improve on the performances of an equivalent SPDNet. Both rely on the generalization of the mathematical background upon which the final Euclidean projection layer is based on, from the Log-Euclidean Metric (LEM) framework to the Affine Invariant Riemannian Metric (AIRM) framework. The first one, the Barycentric Normalization (BarNorm), uses the Riemaniann barycenter to improve the logarithmic mapping, making it computationally tractable by only computing the barycenter on a batch of reduced dimension: incoming SPD matrices are centered to their barycenter, ensuring a better conditioned distribution in the arrival Eucidean space. The second one, the Parametric Normalization (ParNorm), makes the projection dependent on the classification loss. More precisely, a parametric SPD matrix acts as the reference projection point, and thus guides the mapping according to the loss.

The second proposition is a Batch Normalization (BatchNorm) algorithm for SPD NNs, mimicking the original BatchNorm in Euclidean nets. The resulting architecture is dubbed SPDNetBN. The algorithm makes use of the SPD Riemannian manifold's geometric structure, namely the Riemannian barycenter, parallel transport, and manifold-constrained backpropagation through non-linear structured functions on SPD matrices. It is interesting to note the proposed Riemanian BatchNorm is an architectural generalization of the DAMNet; indeed, it conceptually acts as the succession of a BarNorm, then a ParNorm, appended after each layer instead of simply the final one. However, experiments show the parametric normalization scheme sometimes hurts the learning, possibly due to the additional degree of freedom and the inherent complexity of learning on the SPD manifold. A final proposition, for which no satisfactory experimental results were obtained, consists in a 1D convolutional layer based on the BiMap, the idea being to allow a more fine-grained temporal splitting of the signal, and a more expressive SPD neural network.

Results were led on μ-D radar data, and also on video data for emotion recognition, and MoCap data for action recognition. They show that both new architectures improve upon the performances of the baseline SPDNet, on multiple datasets. Although the SPDNetBN usually performs best, the BarNorm presents a

more reliable, smooth training curve. Also, neither the BarNorm or the ParNorm seems to systematically prevail over the other, justifying the interest of both for the DAMNet in different application cases. A comparison to a well-used, better known Riemannian learning method, the MRDRM scheme, systematically showcases much better performances for the neural-based methods.

An additional result, obtained exclusively on μ-D radar data, is the better robustness to lack of data obtained with our two architectures, compared to the baseline SPDNet and to the Fully Temporal Convolutional Network (FTCN) proposed in the previous chapter. The overall performances and robustness of our proposed SPDNetBN makes it a suitable candidate in learning scenarios where data is structured, scarce, and where model size is a relevant issue, which is of frequent occurrence in the radar scenery, by and large in other fields as well.

Also on radar data, the practical interest of the Riemannian barycenter was shown, with two different visualisations. A first one simply illustrates the gain in performance in a simple MDM scheme: we see this gain increasing from the Euclidean mean, through the LEM mean, to the AIRM mean, or Fréchet mean. The second visualisation, richer in content, shows the performance of a frozen DAMNet architecture, given different Euclidean mapping reference points. The best plotted result is the Riemanian barycenter, with the identity matrix performing better than random.

All in all, the potential inspirations from this chapter are quite mind-boggling. Indeed, each and every of the developments of neural networks led from 1953 to today in 2020 constitute a source of inspiration for the development of SPD neural models, and in general neural nets on any Riemannian manifold. A main chunk of work in this chapter was inspired by the BatchNorm algorithm, while the same could be done with past novelties as diverse as recurrent networks, entropy regularization, more sophisticated learning schemes... Innovation in Riemannian neural models will also most logically stem from the field of Information Geometry itself, for instance to allow the classification to be done directly on the manifold through "separating hyper-geodesics" instead of requiring the projection to a Euclidean plane. Other tools in statistics, linear algebra and general mathematics can also be involved in the refinement or speedup of such models: for instance, the logarithmic mapping could be approximated or replaced (under certain conditions) by a matrix square root, which is approximable through an iterative algorithm requiring inly linear operations, making it easily parallelizable. In the end, we believe the research in Riemannian neural models, not only shows great potential in yet untested fields, but also shows evident fields of exploration for future works.

CHAPTER 5

CONCLUSION AND PERSPECTIVES

We first provide a final conclusion to this thesis, before discussing opening perspectives for future ideas.

Final conclusion

In this thesis, we worked on developing neural models inherently adapted to the data, in the context of structured time series classification, with a more thorough spotlight on micro-Doppler (μ-D) radar data.

The primary goal of this thesis was finding learning models, with the internal structure naturally suited to the input data. As a part of this goal, the understanding of the input data itself is of prime importance, and in particular the various representations used to extract meaningful sense out of it. Two main classes of representations were considered in addition to the raw time series: a time-frequency spectrogram, *i.e.* some form of Fourier transform, and a covariance matrix.

The first form of representation, along with raw time series, was mainly the concern of Chapter 3. Since the spectrogram can be seen as both a time series of spectral vectors, or an image of time and frequency dimensions, a first benchmark was established between the use of Recurrent Neural Networks (RNNs) on the one hand, and Convolutional Neural Networks (CNNs) on the other hand, or more precisely a Fully Temporal Convolutional Network (FTCN). While the convolutional architecture prevailed, it was interetsing to notice a strong upper hand of both models *w.r.t.* a standard logistic regression, justifying the incorporation of signal knowledge within the learning models. Other features of interest of the FTCN are, it preserves the temporal structure of the input data, and the resolution of the output feature time series is also fully controlable. Furthermore, a complex-valued version of the FTCN was devised; we also introduced a novel Fourier convolution layer, allowing for a flexible refining of the Fourier filter-bank; the two complex architectures proposed were thus the $\mathbb{CR}$ Neural Network ($\mathbb{CR}$Net) and the Fourier Neural Network (FourierNet), one operating on the complex spectrogram, the other on the raw complex time series. Subsequent experiments on μ-D radar data

further showed the practical interest of using the complex architectures – mainly in the context of plentiful, and challenging data.

The second form of representation, the covariance, was mainly the concern of Chapter 4. The covariance matrix posseses a particular geometry, that of being Symmetric Positive Definite (SPD). As such, it does not belong to a flat, Euclidean space but rather to a curved, Riemannian manifold; learning models must thus be suited to this underlying manifold. We mainly focused on the Riemannian version of Deep Neural Networks (DNNs), a founding instance of which we base ourselves on, the SPD neural network (SPDNet). In this context, we developed two series of contributions. A first one, the Data-Aware Mapping Network (DAMNet) architecture, dwelled upon the final Euclidean projection layer, generalizing it with two distinct layers, the Barycentric Normalization (BarNorm) making use of the Riemannian barycenter, and the Parametric Normalization (ParNorm) making use of a parametric, learnable SPD matrix. Then, these normalization layers were pushed to their natural evolution in the Batch-Normalized SPDNet (SPDNetBN), in which we introduced a Riemannian equivalent to the well-established Batch Normalization (BatchNorm) layer. Experimentations were conducted on radar data, but also on video data for emotion recognition, and Motion Capture (MoCap) data for action recognition. They showed rather systematic improvements from the SPDNet, through the DAMNet, to the SPDNetBN, although it seems the DAMNet with BarNorm regularization eased the learning in some cases.

Given these different learning models associated to different input representations, it may be tough to chose between one or the other in a given scenario. The second purpose of Chapter 3 was also to provide a congregating solution, by making use of all representations and models in a single pipeline, called Second-Order Fully Temporal Network (SOFTNet). This final new model consists in plugging a SPD neural net to a FTCN, which is made possible by the FTCN feature maps being of temporal nature. This amounts to building a first-order representation of the data through a DNN, and then appending a second-order neural model to study the covariance of the feature time series, which makes for a conceptually strong model, making use of a variety of representations and adapted learning models of the data. This pipeline makes for a fine transition to perspectives on future works.

Opening perspectives

As parting words, let us now jot down a few thoughts brought to light in the midst of this thesis.

As hinted above, the SOFTNet pipeline provides a good starting point: from what one may gather for its figurative aspect, its modularity potentially allows a combinatorial expanse of re-configured models suited to the task at hand: to our mind, the interaction of first, second and even higher statistical orders of the data within adapted Euclidean or Riemannian models shows the strongest promises for future research potential. This is already the case for second-order models, which simply plug a SPD neural net after a CNN to improve the performance of the CNN in its classification task. The independent blocks of Euclidean and Riemannian models may advance seperately in their journey towards more efficient and powerful learning abilities, but their union may prove the ultimate provider of cutting-edge performance.

These independent evolutions are also a cornerstone subject of interest, specifically in our case developments in SPD architectures, not only on the technical advances, but also in domain applications. The use of Parallel Transport (PT) for domain adaptation is starting to be explored, with potential applications as diverse as there are domains to adapt; for instance, the self-driving car problem of having to perform equally well in very different landscapes or weather conditions. The usage of local covariance descriptors for image segmentation is also a incipient possibility, paving the way to possible innovations such as pixel-covariance-RNN, or more simply an application of a convolutional SPDNet.

We end this thesis by reenacting our original guiding principle, and perhaps the most obvious perspective of our works: whenever any kind of local structure defines the data to be studied, a covariance structure may be extracted. Then, SPD-based methods may be applied, in possible collaboration with a deep Euclidean model. For it is together, not isolated one from the other, that their individual qualities may be expressed in emergence towards a dominant performance.

BIBLIOGRAPHY

[1] A. Hern. "Computers are now better than humans at recognising images". en-GB. In: *The Guardian* (May 2015) (cit. on p. 1).

[2] "As Self-Driving Cars Stall, Players Revive an Old Approach". en. In: *Wired* () (cit. on p. 1).

[3] M. Chen, Y. Hao, K. Hwang, L. Wang, and L. Wang. "Disease Prediction by Machine Learning Over Big Data From Healthcare Communities". In: *IEEE Access* 5 (2017), pp. 8869–8879 (cit. on p. 1).

[4] *Your virtual doctor will see you now: AI app as accurate as doctors in 80% of primary care diseases*. en (cit. on p. 1).

[5] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng. "An Application of Reinforcement Learning to Aerobatic Helicopter Flight". In: *Advances in Neural Information Processing Systems 19*. Ed. by B. Schölkopf, J. C. Platt, and T. Hoffman. MIT Press, 2007, pp. 1–8 (cit. on p. 1).

[6] *AlphaGo Zero: Starting from scratch*. ALL (cit. on p. 1).

[7] F. Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386 (cit. on pp. 1, 27).

[8] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (Dec. 1989), pp. 541–551 (cit. on pp. 1, 31).

[9] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324 (cit. on pp. 1, 31).

[10] T. Durand, T. Mordan, N. Thome, and M. Cord. "WILDCAT: Weakly Supervised Learning of Deep ConvNets for Image Classification, Pointwise Localization and Segmentation". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017)*. Honolulu, HI, United States: IEEE, July 2017 (cit. on p. 1).

[11] H. Ben-younes, R. Cadene, M. Cord, and N. Thome. "MUTAN: Multimodal Tucker Fusion for Visual Question Answering". en. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. Venice: IEEE, Oct. 2017, pp. 2631–2639 (cit. on p. 1).

[12] J. Johnson. *jcjohnson/neural-style*. original-date: 2015-09-01T04:55:14Z. Dec. 2019 (cit. on p. 1).

[13] *NVIDIA/pix2pixHD*. original-date: 2017-12-01T19:19:14Z. Dec. 2019 (cit. on p. 1).

[14] C. Tao, H. Pan, Y. Li, and Z. Zou. "Unsupervised Spectral–Spatial Feature Learning With Stacked Sparse Autoencoder for Hyperspectral Imagery Classification". In: *IEEE Geoscience and Remote Sensing Letters* 12.12 (Dec. 2015), pp. 2438–2442 (cit. on p. 2).

[15] H. Liang and Q. Li. "Hyperspectral Imagery Classification Using Sparse Representations of Convolutional Neural Network Features". en. In: *Remote Sensing* 8.2 (Feb. 2016), p. 99 (cit. on p. 2).

[16] F. Tupin, H. Maitre, J.-F. Mangin, J.-M. Nicolas, and E. Pechersky. "Detection of linear features in SAR images: application to road network extraction". In: *IEEE Transactions on Geoscience and Remote Sensing* 36.2 (Mar. 1998), pp. 434–453 (cit. on p. 2).

[17] F. Tupin, G. Liu, and Y. Gousseau. "A contrario comparison of local descriptors for change detection in Very High spatial Resolution (VHR) satellite images of urban areas". en. In: *IEEE Transactions on Geoscience and Remote Sensing* (2018) (cit. on p. 2).

[18] A. M. Atto, E. Trouve, Y. Berthoumieu, and G. Mercier. "Multidate Divergence Matrices for the Analysis of SAR Image Time Series". In: *IEEE Transactions on Geoscience and Remote Sensing* 51.4 (Apr. 2013), pp. 1922–1938 (cit. on p. 2).

[19] Inman, M. *6 things I learned from riding in a Google Self-Driving Car*. en (cit. on p. 2).

[20] J. Lien, N. Gillian, M. E. Karagozler, P. Amihood, C. Schwesig, E. Olson, H. Raja, and I. Poupyrev. "Soli: Ubiquitous Gesture Sensing with Millimeter Wave Radar". In: *ACM Trans. Graph.* 35.4 (July 2016), 142:1–142:19 (cit. on p. 2).

[21] S. Wang, J. Song, J. Lien, I. Poupyrev, and O. Hilliges. "Interacting with Soli: Exploring Fine-Grained Dynamic Gesture Recognition in the Radio-Frequency Spectrum". en. In: ACM Press, 2016, pp. 851–860 (cit. on p. 2).

[22] J. C. Maxwell. "VIII. A dynamical theory of the electromagnetic field". In: *Philosophical Transactions of the Royal Society of London* 155 (Jan. 1865), pp. 459–512 (cit. on p. 2).

[23] H. Hertz. "Die Kräfte electrischer Schwingungen, behandelt nach der Maxwell'schen Theorie". en. In: *Annalen der Physik* 272.1 (1889), pp. 1–22 (cit. on p. 2).

[24] R. M. Page. "The Early History of Radar". In: *Proceedings of the IRE* 50.5 (May 1962), pp. 1232–1236 (cit. on p. 2).

[25] J. J. Fahie. *A history of wireless telegraphy, 1838-1899:including some bare-wire proposals for subaqueous telegraphs /*. Edinburgh : 1899 (cit. on p. 2).

[26] Y. Blanchard. "Une histoire du radar en lien avec les mutations du système technique". In: *Revue de l Electricité et de l Electronique* 2019 (2019), pp. 35–46 (cit. on p. 2).

[27] "Gatwick 'drone sighting' diverts flights". en-GB. In: *BBC News* (Apr. 2019) (cit. on p. 2).

[28] M. Bouchaud. *Drones Have Been Spotted Flying Over French Nuclear Power Plants*. en. Oct. 2014 (cit. on p. 2).

[29] "Top Iranian general killed by US in Iraq". en-GB. In: *BBC News* (Jan. 2020) (cit. on p. 2).

[30] A. Press. "Russia implies US involvement in drone strikes on Syria military bases". en-GB. In: *The Guardian* (Jan. 2018) (cit. on p. 2).

[31] "Syrian army foils drone attack on military base in northwest". en. In: *Reuters* (Sept. 2019) (cit. on p. 2).

[32] M. J. Boyle. "The costs and consequences of drone warfare". In: *International Affairs (Royal Institute of International Affairs 1944-)* 89.1 (2013), pp. 1–29 (cit. on p. 2).

[33] S. A. Shah. *International Law and Drone Strikes in Pakistan: The Legal and Socio-political Aspects*. en. Google-Books-ID: 1hlWBQAAQBAJ. Routledge, Nov. 2014 (cit. on p. 2).

[34] J. Vincent. *Twitter taught Microsoft's friendly AI chatbot to be a racist asshole in less than a day*. en. Mar. 2016 (cit. on p. 3).

[35] "Amazon scraps secret AI recruiting tool that showed bias against women". en. In: *Reuters* (Oct. 2018) (cit. on p. 3).

[36] J. L. Julia Angwin. *Machine Bias*. en. text/html. May 2016 (cit. on p. 3).

[37] I. Goodfellow, J. Shlens, and C. Szegedy. "Explaining and Harnessing Adversarial Examples". In: *International Conference on Learning Representations*. 2015 (cit. on p. 4).

[38] *Tackling bias in artificial intelligence (and in humans) | McKinsey*. en (cit. on p. 4).

[39] *Deep Learning in Practice* (cit. on p. 4).

[40] D. G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". en. In: *International Journal of Computer Vision* 60.2 (Nov. 2004), pp. 91–110 (cit. on p. 5).

[41] P. Perona. "Deformable Kernels for Early Vision". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 17.5 (May 1995), pp. 488–499 (cit. on p. 5).

[42] Shannon C. E. "A Mathematical Theory of Communication". In: *Bell System Technical Journal* 27.3 (July 2013), pp. 379–423 (cit. on pp. 5, 41, 80).

[43] J. Capon. "High-resolution frequency-wavenumber spectrum analysis". In: *Proceedings of the IEEE* 57.8 (Aug. 1969), pp. 1408–1418 (cit. on p. 5).

[44] W. Heisenberg and L. d. Broglie. "Les Principes Physiques de la Théorie des Quanta". In: *Les Etudes Philosophiques* 13.1 (1958), pp. 75–75 (cit. on p. 5).

[45] L. Cohen. "Time-frequency distributions-a review". In: *Proceedings of the IEEE* 77.7 (July 1989), pp. 941–981 (cit. on p. 5).

[46] S. Mallat. *A Wavelet Tour of Signal Processing, Third Edition: The Sparse Way*. 3rd. USA: Academic Press, Inc., 2008 (cit. on p. 5).

[47] P. Flandrin. *Time-Frequency/Time-Scale Analysis, Volume 10*. 1st. Orlando, FL, USA: Academic Press, Inc., 1998 (cit. on pp. 5, 57).

[48] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105 (cit. on pp. 7, 31).

[49] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *International Conference on Learning Representations*. 2015 (cit. on pp. 7, 33).

[50] S.-H. Tsang. *Review: Batch Normalization (Inception-v2 / BN-Inception) —The 2nd to Surpass Human-Level...* en. Mar. 2019 (cit. on p. 7).

[51] *Colloquium d'Informatique de Sorbonne Université*. 2018 (cit. on p. 8).

[52] *Yoshua Bengio | From System 1 Deep Learning to System 2 Deep Learning | NeurIPS* (cit. on p. 8).

[53] D. Gunning. "Explainable Artificial Intelligence (XAI)". en. In: (2017), p. 36 (cit. on p. 8).

[54] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". en. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, June 2016, pp. 770–778 (cit. on pp. 8, 160).

[55] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. "Striving for Simplicity: The All Convolutional Net". In: *ICLR (workshop track)*. 2015 (cit. on p. 8).

[56] T. Robert, N. Thome, and M. Cord. "HybridNet: Classification and Reconstruction Cooperation for Semi-Supervised Learning". In: 2018, pp. 153–169 (cit. on p. 8).

[57] T. Mordan, N. Thome, G. Henaff, and M. Cord. "Deformable Part-based Fully Convolutional Network for Object Detection". en. In: *Procedings of the British Machine Vision Conference 2017*. London, UK: British Machine Vision Association, 2017, p. 88 (cit. on p. 8).

[58] T. S. Cohen and M. Welling. "Steerable CNNs". In: (Nov. 2016) (cit. on p. 8).

[59] S. Ioffe and C. Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". en. In: *International Conference on Machine Learning*. June 2015, pp. 448–456 (cit. on pp. 8, 35, 99, 104).

[60] J. Xu, X. Sun, Z. Zhang, G. Zhao, and J. Lin. "Understanding and Improving Layer Normalization". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 4383–4393 (cit. on p. 8).

[61] X. Wang, Y. Jin, M. Long, J. Wang, and M. I. Jordan. "Transferable Normalization: Towards Improving Transferability of Deep Neural Networks". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 1951–1961 (cit. on p. 8).

[62] M. Blot, T. Robert, N. Thome, and M. Cord. "Shade: Information-Based Regularization for Deep Learning". In: *2018 25th IEEE International Conference on Image Processing (ICIP)*. ISSN: 2381-8549. Oct. 2018, pp. 813–817 (cit. on p. 9).

[63] A. Krogh and J. A. Hertz. "A Simple Weight Decay Can Improve Generalization". In: *Advances in Neural Information Processing Systems 4*. Ed. by J. E. Moody, S. J. Hanson, and R. P. Lippmann. Morgan-Kaufmann, 1992, pp. 950–957 (cit. on p. 9).

[64] T.-H. Vu, H. Jain, M. Bucher, M. Cord, and P. Perez. "ADVENT: Adversarial Entropy Minimization for Domain Adaptation in Semantic Segmentation". In: 2019, pp. 2517–2526 (cit. on p. 9).

[65] M. Arjovsky, S. Chintala, and L. Bottou. "Wasserstein Generative Adversarial Networks". en. In: *International Conference on Machine Learning*. July 2017, pp. 214–223 (cit. on pp. 9, 44).

[66] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. "Improved Training of Wasserstein GANs". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 5767–5777 (cit. on p. 9).

[67] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. "Geometric Deep Learning: Going beyond Euclidean data". In: *IEEE Signal Processing Magazine* 34.4 (July 2017), pp. 18–42 (cit. on pp. 9, 38).

[68] O. Yair, M. Ben-Chen, and R. Talmon. "Parallel Transport on the Cone Manifold of SPD Matrices for Domain Adaptation". In: *IEEE Transactions on Signal Processing* 67.7 (Apr. 2019), pp. 1797–1811 (cit. on pp. 9, 107).

[69] G. Marceau-Caron and Y. Ollivier. "Practical Riemannian neural networks". In: *arXiv preprint arXiv:1602.08007* (2016) (cit. on pp. 9, 10).

[70] A. Barachant, S. Bonnet, M. Congedo, and C. Jutten. "Multiclass Brain–Computer Interface Classification by Riemannian Geometry". In: *IEEE Transactions on Biomedical Engineering* 59.4 (Apr. 2012), pp. 920–928 (cit. on pp. 9, 48, 49, 97).

[71] X. Pennec, P. Fillard, and N. Ayache. "A Riemannian Framework for Tensor Computing". en. In: *International Journal of Computer Vision* 66.1 (Jan. 2006), pp. 41–66 (cit. on pp. 9, 47, 48, 61, 97–99, 124).

[72] D. Acharya, Z. Huang, D. P. Paudel, and L. V. Gool. "Covariance Pooling for Facial Expression Recognition". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. June 2018, pp. 480–4807 (cit. on pp. 9, 75, 98).

[73] A. Mollahosseini, D. Chan, and M. H. Mahoor. "Going deeper in facial expression recognition using deep neural networks". In: *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. Mar. 2016, pp. 1–10 (cit. on p. 9).

[74] O. Tuzel, F. Porikli, and P. Meer. "Region Covariance: A Fast Descriptor for Detection and Classification". en. In: *Computer Vision – ECCV 2006*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, May 2006, pp. 589–600 (cit. on pp. 9, 97).

[75] C. R. Rao. "Information and the Accuracy Attainable in the Estimation of Statistical Parameters". en. In: *Breakthroughs in Statistics*. Springer Series in Statistics. Springer, New York, NY, 1992, pp. 235–247 (cit. on pp. 9, 44).

[76] M. Fréchet. "Sur l'extension de certaines evaluations statistiques au cas de petits echantillons". In: *Revue de l'Institut International de Statistique / Review of the International Statistical Institute* 11.3/4 (1943), pp. 182–205 (cit. on pp. 9, 44).

[77] S.-i. Amari. *Information Geometry and Its Applications*. en. Applied Mathematical Sciences. Springer Japan, 2016 (cit. on pp. 9, 104).

[78] *Information geometry*. en. Page Version ID: 898040366. May 2019 (cit. on p. 9).

[79] J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek. "Image Classification with the Fisher Vector: Theory and Practice". en. In: *International Journal of Computer Vision* 105.3 (Dec. 2013), pp. 222–245 (cit. on p. 10).

[80] H. Jegou, F. Perronnin, M. Douze, J. Sanchez, P. Perez, and C. Schmid. "Aggregating Local Image Descriptors into Compact Codes". en. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.9 (Sept. 2012), pp. 1704–1716 (cit. on p. 10).

[81] S.-i. Amari. "Natural Gradient Works Efficiently in Learning". In: *Neural Computation* 10.2 (Feb. 1998), pp. 251–276 (cit. on pp. 10, 51).

[82] R. Pascanu and Y. Bengio. "Revisiting natural gradient for deep networks". In: *In International Conference on Learning Representations*. 2014 (cit. on pp. 10, 51).

[83] G. Marceau-Caron and Y. Ollivier. "Natural Langevin Dynamics for Neural Networks". In: *Geometric Science of Information*. Ed. by F. Nielsen and F. Barbaresco. Vol. 10589. Cham: Springer International Publishing, 2017, pp. 451–459 (cit. on p. 10).

[84] D. Brooks, O. Schwander, F. Barbaresco, J. Schneider, and M. Cord. "Temporal Deep Learning for Drone Micro-Doppler Classification". In: *2018 19th International Radar Symposium (IRS)*. June 2018, pp. 1–10 (cit. on p. 13).

[85] D. Brooks, O. Schwander, F. Barbaresco, J. Schneider, and M. Cord. "Exploring Complex Time-series Representations for Riemannian Machine Learning of Radar Data". In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. May 2019, pp. 3672–3676 (cit. on p. 13).

[86] D. Brooks, O. Schwander, F. Barbaresco, J.-Y. Schneider, and M. Cord. "Complex-valued neural networks for fully-temporal micro-Doppler classification". In: *2019 20th International Radar Symposium (IRS)*. ISSN: 2155-5753, 2155-5745. June 2019, pp. 1–10 (cit. on p. 13).

[87] D. Brooks, O. Schwander, F. Barbaresco, J.-Y. Schneider, and M. Cord. "Second-Order Networks in PyTorch". en. In: *Geometric Science of Information*. Ed. by F. Nielsen and F. Barbaresco. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 751–758 (cit. on p. 13).

[88] D. Brooks, O. Schwander, F. Barbaresco, J.-Y. Schneider, and M. Cord. "A Hermitian Positive Definite neural network for micro-Doppler complex covariance processing". In: *International Radar Conference*. Toulon, France, Sept. 2019 (cit. on p. 13).

[89] D. Brooks, F. Barbaresco, Y. Ziani, J.-Y. Schneider, and C. Adnet. "IA & réseaux de neurones profonds pour la reconnaissance Radar de drones sur critères Micro-Doppler et Cinématique". fr. In: Rennes, FRANCE: Computer & Electronics Security Applications Rendez-vous (C&ESAR), Nov. 2019, p. 16 (cit. on p. 13).

[90] D. Brooks, O. Schwander, F. Barbaresco, J.-Y. Schneider, and M. Cord. "Riemannian batch normalization for SPD neural networks". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 15463–15474 (cit. on p. 13).

[91] C. Doppler. *Ueber das farbige Licht der Doppelsterne und einiger anderer Gestirne des Himmels*. de. Google-Books-ID: zl5RAAAAcAAJ. Calve, 1842 (cit. on p. 17).

[92] V. C. Chen, F. Li, S.-S. Ho, and H. Wechsler. "Micro-Doppler effect in radar: phenomenon, model, and simulation study". In: *IEEE Transactions on Aerospace and electronic systems* 42.1 (2006), pp. 2–21 (cit. on p. 19).

[93] C. Bishop. *Neural Networks for Pattern Recognition*. USA: Oxford University Press, Inc., 1995 (cit. on p. 23).

[94] C. Bishop. *Pattern Recognition and Machine Learning*. en. Information Science and Statistics. New York: Springer-Verlag, 2006 (cit. on p. 23).

[95] S. S. Haykin and S. S. Haykin. *Neural networks and learning machines*. en. 3rd ed. OCLC: ocn237325326. New York: Prentice Hall, 2009 (cit. on p. 23).

[96] Y. LeCun, Y. Bengio, and G. Hinton. "Deep learning". en. In: *Nature* 521.7553 (May 2015), pp. 436–444 (cit. on p. 23).

[97] H. Steinhaus. "Sur la division des corp materiels en parties". In: *Bull. Acad. Polon. Sci* 1.804 (1956), p. 801 (cit. on p. 25).

[98] J. MacQueen. "Some methods for classification and analysis of multivariate observations". EN. In: The Regents of the University of California, 1967 (cit. on p. 25).

[99] D. R. Cox. "The regression analysis of binary sequences". In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1958), pp. 215–242 (cit. on p. 25).

[100] S. Lawrence, C. Giles, Ah Chung Tsoi, and A. Back. "Face recognition: a convolutional neural-network approach". en. In: *IEEE Transactions on Neural Networks* 8.1 (Jan. 1997), pp. 98–113 (cit. on p. 30).

[101] Y. L. Cun. "A Theoretical Framework for Back-Propagation". In: (1988) (cit. on p. 29).

[102] G. R. Cross and A. K. Jain. "Markov Random Field Texture Models". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-5.1 (Jan. 1983), pp. 25–39 (cit. on p. 31).

[103] S. Z. Li. "Markov random field models in computer vision". en. In: *Computer Vision — ECCV '94*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, May 1994, pp. 361–370 (cit. on p. 31).

[104] L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, L. D. Jackel, Y. LeCun, U. A. Muller, E. Sackinger, P. Simard, and V. Vapnik. "Comparison of classifier methods: a case study in handwritten digit recognition". In: *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3 - Conference C: Signal Processing (Cat. No.94CH3440-5)*. Vol. 2. Oct. 1994, 77–82 vol.2 (cit. on p. 31).

[105] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision* 115.3 (Dec. 2015), pp. 211–252 (cit. on p. 31).

[106] K. A. I. Aboalayon, M. Faezipour, W. S. Almuhammadi, and S. Moslehpour. "Sleep Stage Classification Using EEG Signal Analysis: A Comprehensive Survey and New Investigation". en. In: *Entropy* 18.9 (Sept. 2016), p. 272 (cit. on p. 31).

[107] V. J. Lawhern, A. J. Solon, N. R. Waytowich, S. M. Gordon, C. P. Hung, and B. J. Lance. "EEGNet: a compact convolutional neural network for EEG-based brain–computer interfaces". en. In: *Journal of Neural Engineering* 15.5 (July 2018), p. 056013 (cit. on p. 31).

[108] N. Takahashi, M. Gygli, and L. V. Gool. "AENet: Learning Deep Audio Features for Video Analysis". In: *IEEE Transactions on Multimedia* 20.3 (Mar. 2018), pp. 513–524 (cit. on p. 31).

[109] K. J. Piczak. "Environmental sound classification with convolutional neural networks". In: *Machine Learning for Signal Processing (MLSP), 2015 IEEE 25th International Workshop on*. IEEE, 2015, pp. 1–6 (cit. on p. 31).

[110] R. P. Trommel, R. I. A. Harmanny, L. Cifola, and J. N. Driessen. "Multi-target human gait classification using deep convolutional neural networks on micro-doppler spectrograms". In: *2016 European Radar Conference (EuRAD)*. Oct. 2016, pp. 81–84 (cit. on p. 31).

[111] J. Long, E. Shelhamer, and T. Darrell. "Fully convolutional networks for semantic segmentation". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. ISSN: 1063-6919. June 2015, pp. 3431–3440 (cit. on p. 33).

[112] R. Girshick, J. Donahue, T. Darrell, and J. Malik. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. June 2014, pp. 580–587 (cit. on p. 33).

[113] R. Girshick. "Fast R-CNN". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. Dec. 2015, pp. 1440–1448 (cit. on p. 33).

[114] S. Ren, K. He, R. Girshick, and J. Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett. Curran Associates, Inc., 2015, pp. 91–99 (cit. on p. 33).

[115] Y. Bengio, P. Simard, and P. Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE Transactions on Neural Networks* 5.2 (Mar. 1994), pp. 157–166 (cit. on pp. 36, 37).

[116] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780 (cit. on pp. 36, 37).

[117] R. Pascanu, T. Mikolov, and Y. Bengio. "On the difficulty of training recurrent neural networks". In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML'13. Atlanta, GA, USA: JMLR.org, June 2013, pp. III–1310–III–1318 (cit. on p. 37).

[118] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734 (cit. on p. 37).

[119] B. Xu, N. Wang, T. Chen, and M. Li. "Empirical Evaluation of Rectified Activations in Convolutional Network". In: *arXiv:1505.00853 [cs, stat]* (May 2015). arXiv: 1505.00853 (cit. on pp. 37, 60).

[120] Y. Xu, Q. Kong, Q. Huang, W. Wang, and M. D. Plumbley. "Convolutional gated recurrent neural network incorporating spatial features for audio tagging". In: *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017*. IEEE, 2017, pp. 3461–3466 (cit. on p. 37).

[121] I. Sutskever, J. Martens, and G. Hinton. "Generating text with recurrent neural networks". In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. ICML'11. Bellevue, Washington, USA: Omnipress, June 2011, pp. 1017–1024 (cit. on p. 37).

[122] T. Luong, H. Pham, and C. D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 1412–1421 (cit. on p. 37).

[123] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. "Attention is All you Need". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 5998–6008 (cit. on p. 37).

[124] C. Tallec and Y. Ollivier. "Can recurrent neural networks warp time?" In: () (cit. on p. 37).

[125] *Understanding LSTM Networks – colah's blog* (cit. on p. 38).

[126] K. Sun, P. Koniusz, and Z. Wang. "Fisher-Bures Adversary Graph Convolutional Networks". In: *arXiv:1903.04154 [cs, stat]* (Mar. 2019). arXiv: 1903.04154 (cit. on p. 44).

[127] J.-D. Boissonnat, F. Nielsen, and R. Nock. "Bregman Voronoi diagrams". In: *Discrete and Computational Geometry* (2010), p. 200 (cit. on p. 44).

[128] A. Siahkamari, V. Saligrama, D. Castanon, and B. Kulis. "Learning Bregman Divergences". In: *arXiv:1905.11545 [cs, stat]* (May 2019). arXiv: 1905.11545 (cit. on p. 44).

[129] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh. "Clustering with Bregman Divergences". In: *Journal of Machine Learning Research* 6.Oct (2005), pp. 1705–1749 (cit. on p. 44).

[130] N. N. Cencov. *Statistical Decision Rules and Optimal Inference*. en. Google-Books-ID: 63CPCwAAQBAJ. American Mathematical Soc., Apr. 2000 (cit. on p. 44).

[131] C. Atkinson and A. F. S. Mitchell. "Rao's Distance Measure". In: *Sankhyā: The Indian Journal of Statistics, Series A (1961-2002)* 43.3 (1981), pp. 345–365 (cit. on p. 44).

[132] J. Burbea. *Informative Geometry of Probability Spaces:* en. Tech. rep. Fort Belvoir, VA: Defense Technical Information Center, Dec. 1984 (cit. on p. 44).

[133] L. T. Skovgaard. "A Riemannian Geometry of the Multivariate Normal Model". In: *Scandinavian Journal of Statistics* 11.4 (1984), pp. 211–223 (cit. on p. 44).

[134] L. Yang, M. Arnaudon, and F. Barbaresco. "Riemannian median, geometry of covariance matrices and radar target detection". In: Nov. 2010, pp. 415–418 (cit. on p. 47).

[135] S. Bonnabel and R. Sepulchre. "Riemannian Metric and Geometric Mean for Positive Semidefinite Matrices of Fixed Rank". en. In: *SIAM Journal on Matrix Analysis and Applications* 31.3 (Jan. 2010), pp. 1055–1070 (cit. on p. 48).

[136] H. Karcher. "Riemannian center of mass and mollifier smoothing". en. In: *Communications on Pure and Applied Mathematics* 30.5 (Sept. 1977), pp. 509–541 (cit. on pp. 48, 49).

[137] L. Yang, M. Arnaudon, and F. Barbaresco. "Riemannian median, geometry of covariance matrices and radar target detection". In: *The 7th European Radar Conference*. Sept. 2010, pp. 415–418 (cit. on pp. 48, 49).

[138] F. Yger and M. Sugiyama. "Supervised LogEuclidean Metric Learning for Symmetric Positive Definite Matrices". In: *arXiv:1502.03505 [cs]* (Feb. 2015). arXiv: 1502.03505 (cit. on pp. 49, 97, 103, 111).

[139] A. Barachant, S. Bonnet, M. Congedo, and C. Jutten. "Classification of covariance matrices using a Riemannian-based kernel for BCI applications". In: *Neurocomputing* 112 (July 2013), pp. 172–178 (cit. on pp. 49, 50, 97, 102).

[140] S. Boyd and L. Vandenberghe. *Convex Optimization*. en. Google-Books-ID: IUZdAAAAQBAJ. Cambridge University Press, Mar. 2004 (cit. on pp. 51, 67).

[141] F. Hlawatsch and F. Auger. *Time-Frequency Analysis*. en. Google-Books-ID: tOeeJyP95IQC. John Wiley & Sons, Mar. 2013 (cit. on p. 57).

[142] P. Stoica and R. L. Moses. *Spectral analysis of signals*. Upper Saddle River, N.J: Pearson/Prentice Hall, 2005 (cit. on p. 57).

[143] M. Moruzzis and N. Colin. "Automatic recognition of air targets for future shorad radars". In: *RTO SCI Symposium on" Non cooperative Air Target Identification Using Radar", Mannheim, Germany*. 1998 (cit. on p. 57).

[144] Z. Huang and L. Van Gool. "A Riemannian Network for SPD Matrix Learning". In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. AAAI'17. event-place: San Francisco, California, USA. AAAI Press, 2017, pp. 2036–2042 (cit. on pp. 58, 60, 61, 97, 98, 127–129).

[145] O. Ledoit and M. Wolf. "A well-conditioned estimator for large-dimensional covariance matrices". en. In: *Journal of Multivariate Analysis* 88.2 (Feb. 2004), pp. 365–411 (cit. on pp. 61, 75).

[146] O. Ledoit and M. Wolf. "Nonlinear shrinkage estimation of large-dimensional covariance matrices". In: *The Annals of Statistics* 40.2 (Apr. 2012). arXiv: 1207.5322, pp. 1024–1060 (cit. on pp. 61, 75).

[147] Y. Chen, A. Wiesel, and A. O. Hero III. "Robust Shrinkage Estimation of High-dimensional Covariance Matrices". In: *IEEE Transactions on Signal Processing* 59.9 (Sept. 2011). arXiv: 1009.5331, pp. 4097–4107 (cit. on p. 61).

[148] M. Tiomoko, R. Couillet, F. Bouchard, and G. Ginolhac. "Random Matrix Improved Covariance Estimation for a Large Class of Metrics". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. 2019, pp. 6254–6263 (cit. on p. 61).

[149] M. Tiomoko, R. Couillet, E. Moisan, and S. Zozor. "Improved Estimation of the Distance between Covariance Matrices". In: *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2019, Brighton, United Kingdom, May 12-17, 2019*. 2019, pp. 7445–7449 (cit. on p. 61).

[150] G. Cao and C. Bouman. "Covariance Estimation for High Dimensional Data Vectors Using the Sparse Matrix Transform". In: *Advances in Neural Information Processing Systems 21*. Ed. by D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou. Curran Associates, Inc., 2009, pp. 225–232 (cit. on p. 61).

[151] G. Cao, L. R. Bachega, and C. A. Bouman. "The Sparse Matrix Transform for Covariance Estimation and Analysis of High Dimensional Signals". In: *IEEE Transactions on Image Processing* 20.3 (Mar. 2011), pp. 625–640 (cit. on p. 61).

[152] F. Mezzadri. "How to generate random matrices from the classical compact groups". In: *arXiv:math-ph/0609050* (Sept. 2006). arXiv: math-ph/0609050 (cit. on p. 61).

[153] V. Arsigny, P. Fillard, X. Pennec, and N. Ayache. "Log-Euclidean metrics for fast and simple calculus on diffusion tensors". en. In: *Magnetic Resonance in Medicine* 56.2 (Aug. 2006), pp. 411–421 (cit. on pp. 61, 97, 98).

[154] W. F. Harris. "The average eye". en. In: *Ophthalmic and Physiological Optics* 24.6 (2004), pp. 580–585 (cit. on p. 61).

[155] C. Ionescu, O. Vantzos, and C. Sminchisescu. "Matrix Backpropagation for Deep Networks with Structured Layers". en. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. Santiago, Chile: IEEE, Dec. 2015, pp. 2965–2973 (cit. on pp. 61, 111).

[156] A. Edelman, T. Arias, and S. Smith. "The Geometry of Algorithms with Orthogonality Constraints". In: *SIAM Journal on Matrix Analysis and Applications* 20.2 (Jan. 1998), pp. 303–353 (cit. on pp. 61, 110).

[157] H. Lee, P. Pham, Y. Largman, and A. Y. Ng. "Unsupervised feature learning for audio classification using convolutional deep belief networks". In: *Advances in neural information processing systems*. 2009, pp. 1096–1104 (cit. on pp. 62, 71).

[158] C. Bartz, T. Herold, H. Yang, and C. Meinel. "Language Identification Using Deep Convolutional Recurrent Neural Networks". en. In: *Neural Information Processing*. Ed. by D. Liu, S. Xie, Y. Li, D. Zhao, and E.-S. M. El-Alfy. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 880–889 (cit. on p. 62).

[159] S. Dieleman and B. Schrauwen. "End-to-end learning for music audio". In: *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 6964–6968 (cit. on pp. 62, 71).

[160] G. Parascandolo, H. Huttunen, and T. Virtanen. "Recurrent neural networks for polyphonic sound event detection in real life recordings". In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. ISSN: 2379-190X. Mar. 2016, pp. 6440–6444 (cit. on p. 62).

[161] E. Cakir, G. Parascandolo, T. Heittola, H. Huttunen, T. Virtanen, E. Cakir, G. Parascandolo, T. Heittola, H. Huttunen, and T. Virtanen. "Convolutional Recurrent Neural Networks for Polyphonic Sound Event Detection". In: *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)* 25.6 (June 2017), pp. 1291–1303 (cit. on p. 62).

[162] S. Hershey, S. Chaudhuri, D. P. W. Ellis, J. F. Gemmeke, A. Jansen, C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, M. Slaney, R. Weiss, and K. Wilson. "CNN Architectures for Large-Scale Audio Classification". In: *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2017 (cit. on p. 62).

[163] A. Deshpande. *A Beginner's Guide To Understanding Convolutional Neural Networks Part 2* (cit. on p. 65).

[164] W. Wirtinger. "Zur formalen Theorie der Funktionen von mehr komplexen Veränderlichen". de. In: *Mathematische Annalen* 97.1 (Dec. 1927), pp. 357–375 (cit. on p. 66).

[165] D. H. Brandwood. "A complex gradient operator and its application in adaptive array theory". In: *IEE Proceedings H - Microwaves, Optics and Antennas* 130.1 (Feb. 1983), pp. 11–16 (cit. on pp. 66, 67).

[166] A. v. d. Bos. "Complex gradient and Hessian". In: *IEE Proceedings - Vision, Image and Signal Processing* 141.6 (Dec. 1994), pp. 380–383 (cit. on p. 66).

[167] K. Kreutz-Delgado. "The Complex Gradient Operator and the CR-Calculus". In: *arXiv:0906.4835 [math]* (June 2009). arXiv: 0906.4835 (cit. on p. 66).

[168] *HIPS/autograd*. original-date: 2014-11-24T15:50:23Z. Jan. 2020 (cit. on p. 67).

[169] C. Böddeker, P. Hanebrink, L. Drude, J. Heymann, and R. Haeb-Umbach. "On the Computation of Complex-valued Gradients with Application to Statistically Optimum Beamforming". In: *CoRR* abs/1701.00392 (2017) (cit. on p. 67).

[170] C. Trabelsi, O. Bilaniuk, Y. Zhang, D. Serdyuk, S. Subramanian, J. F. Santos, S. Mehri, N. Rostamzadeh, Y. Bengio, and C. J. Pal. "Deep Complex Networks". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. 2018 (cit. on pp. 67, 68).

[171] F. Barbaresco. "Information Geometry of Covariance Matrix: Cartan-Siegel Homogeneous Bounded Domains, Mostow/Berger Fibration and Fréchet Median". en. In: *Matrix Information Geometry*. Springer, Berlin, Heidelberg, 2013, pp. 199–255 (cit. on p. 69).

[172] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. "Automatic differentiation in PyTorch". In: (Oct. 2017) (cit. on pp. 69, 82, 119, 162).

[173] *DeepKSPD: Learning Kernel-matrix-based SPD Representation for Fine-grained Image Recognition*. en-us (cit. on p. 75).

[174] K. Yu and M. Salzmann. "Second-order Convolutional Neural Networks". In: *arXiv:1703.06817 [cs]* (Mar. 2017). arXiv: 1703.06817 (cit. on pp. 75, 98).

[175] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 75).

[176] *2.6. Covariance estimation — scikit-learn 0.22.1 documentation* (cit. on p. 75).

[177] J. B. Billingsley. *Low-angle Radar Land Clutter: Measurements and Empirical Models*. en. IET, 2002 (cit. on p. 78).

[178] S. Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv:1609.04747 [cs]* (Sept. 2016). arXiv: 1609.04747 (cit. on p. 82).

[179] F. Chollet et al. *Keras*. 2015 (cit. on p. 82).

[180] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. "TensorFlow: A System for Large-scale Machine Learning". In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. OSDI'16. Berkeley, CA, USA: USENIX Association, 2016, pp. 265–283 (cit. on p. 82).

[181] J. Salamon, C. Jacoby, and J. P. Bello. "A Dataset and Taxonomy for Urban Sound Research". In: *22nd ACM International Conference on Multimedia (ACM-MM'14)*. Orlando, FL, USA, Nov. 2014, pp. 1041–1044 (cit. on p. 85).

[182] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks". en. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Mar. 2010, pp. 249–256 (cit. on p. 87).

[183] R. Bhatia. *Positive Definite Matrices*. Princeton, NJ, USA: Princeton University Press, 2015 (cit. on p. 97).

[184] J. Cavazza, P. Morerio, and V. Murino. "When Kernel Methods Meet Feature Learning: Log-Covariance Network for Action Recognition From Skeletal Data". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. July 2017, pp. 1251–1258 (cit. on p. 97).

[185] Z. Huang, J. Wu, and L. V. Gool. "Building Deep Networks on Grassmann Manifolds". en. In: *Thirty-Second AAAI Conference on Artificial Intelligence*. Apr. 2018 (cit. on p. 97).

[186] Z. Huang, C. Wan, T. Probst, and L. Van Gool. "Deep Learning on Lie Groups for Skeleton-based Action Recognition". In: *arXiv:1612.05877 [cs]* (Dec. 2016). arXiv: 1612.05877 (cit. on p. 97).

[187] Z. Dong, S. Jia, C. Zhang, M. Pei, and Y. Wu. "Deep manifold learning of symmetric positive definite matrices with application to face recognition". In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. AAAI'17. San Francisco, California, USA: AAAI Press, Feb. 2017, pp. 4009–4015 (cit. on p. 98).

[188] Z. Gao, Y. Wu, X. Bu, and Y. Jia. "Learning a Robust Representation via a Deep Network on Symmetric Positive Definite Manifolds". In: *arXiv:1711.06540 [cs]* (Nov. 2017). arXiv: 1711.06540 (cit. on p. 98).

[189] T. Zhang, W. Zheng, Z. Cui, and C. Li. "Deep Manifold-to-Manifold Transforming Network". In: *2018 25th IEEE International Conference on Image Processing (ICIP)*. Oct. 2018, pp. 4098–4102 (cit. on p. 98).

[190] R. Chakraborty, J. Bouza, J. Manton, and B. C. Vemuri. "ManifoldNet: A Deep Network Framework for Manifold-valued Data". In: *arXiv:1809.06211 [cs]* (Sept. 2018). arXiv: 1809.06211 (cit. on p. 98).

[191] Y. Mao, R. Wang, S. Shan, and X. Chen. "COSONet: Compact Second-Order Network for Video Face Recognition". en. In: *Computer Vision – ACCV 2018*. Ed. by C. V. Jawahar, H. Li, G. Mori, and K. Schindler. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 51–67 (cit. on p. 98).

[192] P. Li, J. Xie, Q. Wang, and Z. Gao. "Towards Faster Training of Global Covariance Pooling Networks by Iterative Matrix Square Root Normalization". en. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Salt Lake City, UT: IEEE, June 2018, pp. 947–955 (cit. on p. 98).

[193] H. Zhang, S. J. Reddi, and S. Sra. "Riemannian SVRG: Fast Stochastic Optimization on Riemannian Manifolds". In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett. Curran Associates, Inc., 2016, pp. 4592–4600 (cit. on p. 98).

[194] Y. Liu, F. Shang, J. Cheng, H. Cheng, and L. Jiao. "Accelerated First-order Methods for Geodesically Convex Optimization on Riemannian Manifolds". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 4868–4877 (cit. on p. 98).

[195] F. Alimisis, A. Orvieto, G. Bécigneul, and A. Lucchi. "Practical Accelerated Optimization on Riemannian Manifolds". In: *arXiv:2002.04144 [math]* (Feb. 2020). arXiv: 2002.04144 (cit. on p. 98).

[196] A. L. Brigant, F. Barbaresco, and M. Arnaudon. "Geometric barycenters of time/Doppler spectra for the recognition of non-stationary targets". In: *2016 17th International Radar Symposium (IRS)*. May 2016, pp. 1–6 (cit. on p. 99).

[197] F. Yger. "A review of kernels on covariance matrices for BCI applications". In: *2013 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*. Sept. 2013, pp. 1–6 (cit. on pp. 102, 109).

[198] N. Jaquier and S. Calinon. "Gaussian mixture regression on symmetric positive definite matrices manifolds: Application to wrist motion estimation with sEMG". en. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vancouver, BC: IEEE, Sept. 2017, pp. 59–64 (cit. on p. 107).

[199] S. Said, L. Bombrun, Y. Berthoumieu, and J. H. Manton. "Riemannian Gaussian Distributions on the Space of Symmetric Positive Definite Matrices". In: *IEEE Transactions on Information Theory* 63.4 (Apr. 2017), pp. 2153–2170 (cit. on p. 107).

[200] F. Barbaresco. "Jean-Louis Koszul and the Elementary Structures of Information Geometry". en. In: *Geometric Structures of Information*. Ed. by F. Nielsen. Signals and Communication Technology. Cham: Springer International Publishing, 2019, pp. 333–392 (cit. on p. 107).

[201] M. Brodskiĭ, J. Daleckiĭ, O. Èĭdus, I. Iohvidov, M. Kreĭn, O. Ladyženskaja, V. Lidskiĭ, J. Ljubič, V. Macaev, A. Povzner, L. Sahnovič, J. Šmuljan, I. Suharevskiĭ, and N. Uralceva. *Thirteen Papers on Functional Analysis and Partial Differential Equations*. en-US. Vol. 47. American Mathematical Society Translations: Series 2. American Mathematical Society, Dec. 1965 (cit. on p. 111).

[202] A. D. Michal. "Matrix and tensor calculus with applications to mechanics, elasticity, and aeronautics". In: *New York* (1947) (cit. on p. 115).

[203] L. Bers. "Review: A. D. Michal, Matrix and tensor calculus with applications to mechanics, elasticity and aeronautics". EN. In: *Bulletin of the American Mathematical Society* 54 (Nov. 1948), pp. 1092–1093 (cit. on p. 115).

[204] P. S. Dwyer and M. S. Macphail. "Symbolic Matrix Derivatives". EN. In: *The Annals of Mathematical Statistics* 19.4 (Dec. 1948), pp. 517–534 (cit. on p. 115).

[205] T. Papadopoulo and M. I. A. Lourakis. "Estimating the Jacobian of the Singular Value Decomposition: Theory and Applications". en. In: *Computer Vision - ECCV 2000*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2000, pp. 554–570 (cit. on p. 115).

[206] K. B. Petersen, M. S. Pedersen, J. Larsen, K. Strimmer, L. Christiansen, K. Hansen, L. He, L. Thibaut, M. Barão, S. Hattinger, V. Sima, and W. The. *The matrix cookbook*. Tech. rep. 2006 (cit. on p. 115).

[207] M. B. Giles. "Collected Matrix Derivative Results for Forward and Reverse Mode Algorithmic Differentiation". en. In: *Advances in Automatic Differentiation*. Ed. by C. H. Bischof, H. M. Bücker, P. Hovland, U. Naumann, and J. Utke. Lecture Notes in Computational Science and Engineering. Berlin, Heidelberg: Springer, 2008, pp. 35–44 (cit. on p. 115).

[208] J. R. Magnus and H. Neudecker. *Matrix Differential Calculus with Applications in Statistics and Econometrics*. en. Google-Books-ID: 8sOKDwAAQBAJ. John Wiley & Sons, Mar. 2019 (cit. on p. 115).

[209] F. Nielsen and R. Bhatia, eds. *Matrix Information Geometry*. en. Berlin Heidelberg: Springer-Verlag, 2013 (cit. on p. 115).

[210] M. Engin, L. Wang, L. Zhou, and X. Liu. "DeepKSPD: Learning Kernel-Matrix-Based SPD Representation For Fine-Grained Image Recognition". en. In: *Computer Vision – ECCV 2018*. Ed. by V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss. Vol. 11206. Cham: Springer International Publishing, 2018, pp. 629–645 (cit. on p. 115).

[211] N. Charon and F. Barbaresco. *A new approach for target detection in radar images based on geometric properties of covariance matrices' spaces*. fr. 2009 (cit. on p. 120).

[212] A. Dhall, R. Goecke, S. Lucey, and T. Gedeon. "Static facial expression analysis in tough conditions: Data, evaluation protocol and benchmark". In: *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. Nov. 2011, pp. 2106–2112 (cit. on p. 127).

[213] M. Müller, T. Röder, M. Clausen, B. Eberhardt, B. Krüger, and A. Weber. *Documentation Mocap Database HDM05*. Tech. rep. CG-2007-2. Universität Bonn, June 2007 (cit. on p. 128).

[214] M. T. Harandi, M. Salzmann, and R. Hartley. "From Manifold to Manifold: Geometry-Aware Dimensionality Reduction for SPD Matrices". In: *arXiv:1407.1120 [cs]* (July 2014). arXiv: 1407.1120 (cit. on p. 129).

[215] A. Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". en. In: (), p. 60 (cit. on p. 160).

APPENDIX A

ALORITHMIC DETAILS AND PROPERTIES OF THE PROPOSED LEARNING MODELS

Contents

A.1 Model sizes and speeds 157
A.1.1 Model size 157
A.1.2 Model speed 158
A.2 Sample code 159
A.3 Use cases 159
A.4 Organization of the provided PyTorch library 161
A.5 Implementation details 162

Here we benchmark both model's properties and illustrate with some sample code. Benchmark are done on two different datasets: a synthetic one of the three drone classes mentioned above (see Figure 2.3), and one of real recordings, the North Atlantic Treaty Organization (NATO) dataset introduced in Section 4.6, the description of which is given in Appendix B (in the experiments, we use only about 3% of the dataset). To recall in short, the synthetic data gathers 1000 examples per class of 250 ms signals sampled at $4kHz$, so of initial size $(1000,)$, rendered to a $(99, 20)$ time-frequency spectrogam after Fourier convolution.

A.1 Model sizes and speeds

A.1.1 Model size

The number of parameters per model are as follows:

1. Fully Temporal Convolutional Network (FTCN): 13 547 parameters;
2. SPD neural network (SPDNet): 640 parameters.

As such, the Symmetric Positive Definite (SPD)-based model is much more lightweight than the Euclidean model.

A.1.2 Model speed

Speed is evaluated on the two mentioned datasets, synthetic and real, and is given in seconds per epoch. For reference, we give approximate convergent training times on a shared basis of 200 epochs, on the following hardware setup:

1. GPU: GeForce GTX 960M (4GB VRAM)
2. CPU: 16 RAM
3. Processor: Intel Core i7-6700HQ CPU @ 2.60GHz x 8

The FTCN model involves heavyweight yet parallelizable operations, while the SPDNet relies on non-parallelizable operations. For these reasons, the FTCN is trained on GPU and the SPDNet on CPU. Model training speeds (includes all computations) are as follows:

1. FTCN: 1.3s/ep so ~ 4 min for synthetic; 3.2s/ep so ~ 10 min for real (GPU);
2. SPDNet: 3.1s/ep so ~ 10 min for synthetic; 5.2s/ep so ~ 17 min for real (CPU);
3. Second-Order Fully Temporal Network (SOFTNet): 6.7s/ep so ~ 22 min for real (GPU+CPU);
4. SOFTNet with pre-trained and freezed FTCN: 4.8s/ep so ~ 16 min for real, but: the model converges must faster, in about 20ep, so you reach best performance in < 2 min.

Model inference speeds on a 20% validation split of the synthetic dataset, *i.e.* on 600 data samples, are as follows:

1. FTCN: 210 ms, *i.e.* 0.140% of sample duration for synthetic; 380 ms, *i.e.* 0.253% of sample duration for real;
2. SPDNet: 370 ms, *i.e.* 0.247% of sample duration for synthetic; 670 ms, *i.e.* 0.447% of sample duration for real.

A.2 Sample code

The library we propose seamlessly integrates manifold-constrained optimization of structured functions on $\mathcal{S}_*^+$: the code for setting up the learning of a model in PyTorch is only modified in the usage of the *MixOptimizer* class, which mixes a conventional optimizer with the Riemannian ones:

```
import torch.nn as nn
from mixoptimizer import MixOptimizer
...
model=... #define the model
...
l=nn.CrossEntropyLoss()
# define the loss function and mixed optimizer
opt=MixOptimizer(model.parameters(),lr=lr,momentum=0.9,weight_decay=5e-4)
...
l.backward()
# in the training loop, compute gradients and update weights as usually done
opt.step()
```

A.3 Use cases

Here we show how to use the library in practice. Following the PyTorch logic, elementary functions are defined in *torchspdnet.functional* and high-level modules in *torchspdnet.nn*.

A.3.0.1 Basic SPDNet model

Here we give the most basic use case scenario: given input covariance data of size 20×20, we build an SPDNet which reduces its size to 15 then 10 through two Bilinear Mappings (BiMaps) and a Rectified Eigenvalues (ReEig) activation, followed by the Log Eigenvalues (LogEig) and vectorization. Finally, a standard fully-connected layer allows for classification over the 3 classes

```
import torch.nn as nn
import torchspdnet.nn as nn_spd

model=nn.Sequential(
        nn_spd.BiMap(1,1,20,15),
        nn_spd.ReEig(),
        nn_spd.BiMap(1,1,15,10),
        nn_spd.LogEig(),
        nn_spd.Vectorize(),
        nn.Linear(10**2,3)
    )
```

Note that our implementation of the BiMap module supports an arbitrary number of channels, represented by the additional parameters all set to 1 in this example.

A.3.0.2 First-order and second-order combined

In a more complex example, an SPDNet acts upon the features maps of a convolutional network. For an image recognition task, these features may come from a pre-trained deep network but nothing keeps from training the whole network in an end-to-end fashion or to fine-tune the parameters. Here we describe the combination of a pre-trained ResNet-18 (He et al. 2016) on the CIFAR10 (Krizhevsky n.d.) challenge and of SPDNet layers. We recall we call such a model a SOFTNet.

```
import torch.nn as nn
import torchspdnet.nn as nn_spd
from resnet import ResNet18

class SOCNN(nn.Module):
    def __init__(self):
        super(__class__,self).__init__()

        # first-order model
        self.model_fo=ResNet18()
        state_dict=th.load('pretrained/ResNet18.pth')['state_dict']
        self.model_fo.load_state_dict(state_dict)

        # convolutional connection
        self.connection=nn.Conv2d(512,256,kernel_size=(1,1))

        # second-order model
        self.model_so=nn.Sequential(
```

```
            nn_spd.BiMap(1,1,256,128),
            nn_spd.ReEig(),
            nn_spd.BiMap(1,1,128,64),
        ).to(self.device_so)

        self.dense=nn.Sequential(
            nn.Linear(64**2,1024),
            nn.Linear(1024,10)
        )

    def forward(self,x):
        x_fo=self.model_fo(x)
        x_co=self.connection(x_fo)
        x_sym=nn_spd.CovPool()(x_co.view(x_co.shape[0],x_co.shape[1],-1))
        x_so=self.model_so(x_sym)
        x_vec=nn_spd.LogEig()(x_so).view(x_so.shape[0],x_so.shape[-1]**2)
        y=self.dense(x_vec)
        return y
```

A.4 Organization of the provided PyTorch library

The library is organized following the PyTorch module and functional logic:

1. *spd/*: building blocks for the SPDNetBN architecture and training:

   functional.py: core machinery for inference and backpropagation in a SPDNetBN;

   nn.py: modules used to build a SPDNetBN architecture;

2. *cplx/*: building blocks for the complex data handling (only useful for radar data);

   functional.py: core machinery for inference and backpropagation in a complex neural net;

   nn.py: modules used to build a complex net architecture;

3. *experiments/*:

   radar.py: code for lauching experiments on the synthetic radar dataset (results on the NATO dataset are not reproducible beacause of data confidentiality issues);

hdm05.py: code for lauching experiments on the Hochschule der Medien 05 (HDM05) dataset;

afew.py: code for lauching experiments on the Acted Faces Expressions in the Wild (AFEW) dataset;

radar_mrdrm.py: code for lauching minimum Riemannian distance to Riemannian mean (Minimum Riemannian Distance to Riemannian Mean (MRDRM)) experiment on the synthetic radar dataset (results on the real dataset are not reproducible beacause of data confidentiality issues);

hdm05_mrdrm.py: code for lauching MRDRM experiment on the HDM05 dataset;

afew_mrdrm.py: code for lauching MRDRM experiment on the AFEW dataset;

data/: the folder to copy the data to; the others are made available for download [1]. Radar, HDM05 and AFEW datasets respectively weigh 47Mb, 139Mb and 1.3Gb.

To launch the code in the *experiments/* directory, execute for instance *python radar.py*. The code was developed in Python3 and tested under PyTorch (Paszke et al. 2017) v0.4.1 on CUDA version 8.0.61 and run on a laptop i7-6700HQ CPU.

A.5 Implementation details

Finally, we list some details of interest to reproduce the results:

1. Training did not benefit from GPU acceleration, seemingly bottlenecked at eigenvalue operations: computation time desribed in the paper is not decreased using an Nvidia GTX 1070M;

2. Symmetric matrix vectorization: following the Euclidean mapping, it is possible to vectorize only the upper triangular part of the matrix (and normalize the outer-diagonal coefficients by $\sqrt{2}$ to conserve the norm). We conducted all experiments in both settings and observed no significant impact, except for the parametric centering, which seemed to suffer from a smaller representation dimension. The reported results use full matrix vectorization;

[1] The synthetic radar, HDM05 and AFEW datasets may be found at `https://www.dropbox.com/s/dfnlx2bnyh3kjwy/data.zip?dl=0`

3. Matrix orthonormalization: for reasons we are unsure of, performance benefitted from a home-made implementation of the Classical Gram-Schmidt (CGS) process compared to the built-in QR decomposition, especially in the AFEW experiments; again, the exception was the parametric centering which followed the opposite rule; however for fairness of evaluation, we reported the results for the same configuration, i.e. our own CGS;

4. Floating-point precision: as in previous works using structured matrix differentiation, working in double precision was paramount for smooth convergence in all cases; in practice, we observe the precision is most important in the computation of the Loewner matrix, i.e. in the backprop of structured matrix functions, and specifically while inverting differences in eigenvalues;

5. Since we deal with SPD matrices, eigen-decomposition (EIG) is equivalent to Singular Value Decomposition (SVD): the latter being more stable and cheaper to compute, it is possible to use an SVD algorithm, as we and previous works on SPD matrices do. That being said, the matrix exponentiation during the computation of the Riemannian barycenter needs switching to an EIG algorithm as negative eigenvalues can exist in the symmetric matrices.

Appendix B

DETAILED DESCRIPTION OF THE NATO RADAR DATABASE

This appendix details the North Atlantic Treaty Organization (NATO) database used in experiments, as we have found it to be of major interest both experimentally and of practical interest.

B.1 Global overview

The NATO database consists of recordings of 8 different aiborne subjects, including one class encompassing birds and 7 different drones:

- DJI Phantom 3 (carbon fiber & nylon blades) [quadcopter];
- 3DR X8 (carbon fiber blades) [quadcopter];
- 3DR Iris (carbon fiber & nylon blades) [quadcopter];
- Firefly [hybrid]
- Anaconda [fixed wing]
- Opterra [fixed wing]
- Skywalker [fixed wing]

The Phantom and the Iris come in two versions: carbon fiber or nylon blades, which can consitute child classes. The drones are categorized in three types: quadcopter, fixed wing and hybrid, which can consitute parent classes. The radar signals are accessible in their raw form of time series of complex points of amplitude and phase. The subjects were furthermore recorded in 6 frequency bands

(L,S,C,X,Ku and Ka) and with both vertical and horizontal polarization (except for the L band).

B.2 Description of the data

We call one data point the continuous recording of one subject for an arbitrary amount of time. Signals vary in lenth from $7s$ to $672s$. The Pulse Repetition Frequency (PRF) is set to $25kHz$, meaning a signal of 1 min totals $1.5e6$ complex numbers. One data point is furthermore segmented into N non-overlapping chunks of size $M = 1024$ upon which Fast Fourier Transform (FFT) is performed, thus outputting a spectrogram of size $M \times N$. Again, for a signal lasting 1 min, the corresponding spectrogram would be of size 1024×1465.

B.3 Contents of the disk

The disk is divided in 5 CDs, in which folders separate the recordings of different subjects. We compactly describe one such folder with the length in seconds of the different signals within it; we omit the multiplicity of bi-polarization and multiple frequency bands, such the description '*Phantom3 (nylon): 465 + 84*' means that the folder contains two different data points of respective time length $465s$ and $84s$ of the Pantom 3 with nylon blades, repeated across all polarizations and frequency bands. Matlab code is also provided to read and plot the data.

- CD1
  - Phantom3 (nylon): 465 + 84
  - Skywalker: 12
  - X8 (1): 23
  - X8 (2): 45
- CD2

  - Anaconda: 629 + 7
  - Firefly: 562 + 178
  - Iris (nylon): 22 + 170 + 669 + 98
- CD3
  - Iris (carbon): 9 + 672 + 136
  - Opterra: 487 + 43
  - Phantom3 (carbon): 123 + 442 +72
- CDs 4 & 5
  - birds...

B.4 Possible setting for learning

The database is complete, varied, and as such can inspire many possible settings for learning (usage of both polarizations, usage of hierarchy in classes...). We propose here one simple setting, using single polarization in the S band. We also do not consider hierarchy and retain only the 7 drones, which we do not split in subclasses of blade fabric. Furthermore, from a machine learning point-of-view, it is reasonable to trim potentially overkill information: for instance, $M = 1024$ FFT points may be more resolution than we need; we propose instead to use a $p_{ov} = 50\%$-overlapping $N_{fft} = 256$-point FFT. Also, we wish to consider a minimal elementary signal duration for which learning performance is not affected much. In previous works, it was found a sane choice to operate on a duration of the order of one complete blade rotation. We do not know these values for the drones at hand; however, given that a DJI Phantom 2 rotates at about 4800tr/min, we propose to set the minimal duration around $10ms$, which discretizes to 250 sample points, which we round up to 256. We thus propose to apply the same setting here, which converts a $T = 1$ min signal to a spectrogram of length 11720, or equivalently a set of $N = 360$ spectrograms of length $n = 32$ and frequency resolution 256. In the following Table B.1, we sum up the proposed setting by grouping the total recording time for each class along with the corresponding

Table B.1.: **Duration of recorded data in the database**. The duration is split through the classes, and gives a reference amount of number of data points as individual spectrograms.

	Phantom3	X8	Iris	Firefly	Anaconda	Opterra	Skywalker
Total duration (s)	1186	68	1776	740	636	530	12
Number of spectrograms	7239	415	10840	4516	3882	3235	73

number of 256×32 spectrograms. Note that number is obtained with the formula $N = \frac{T*PRF}{p_{ov}*N_{fft}*n}$.