

---

# **Geometric Deep Learning for Shape Analysis**

**Extending deep learning techniques  
to non-Euclidean manifolds**

Doctoral Dissertation submitted to the  
Faculty of Informatics of the Università della Svizzera Italiana  
in partial fulfillment of the requirements for the degree of  
Doctor of Philosophy

presented by  
**Davide Boscaini**

under the supervision of  
**Michael M. Bronstein and Jonathan Masci**

August 2017





---

## Dissertation Committee

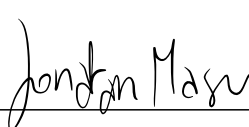
<b>Pierre Vandergheynst</b>	EPFL, Lausanne, CH
<b>Maks Ovsjanikov</b>	École Polytechnique, Paris, France
<b>Jürgen Schmidhuber</b>	Università della Svizzera italiana, Lugano, CH
<b>Kai Hormann</b>	Università della Svizzera italiana, Lugano, CH
<b>Michael M. Bronstein</b>	Università della Svizzera italiana, Lugano, CH
<b>Jonathan Masci</b>	NNAISENSE, Lugano, CH

Dissertation accepted on 16 August 2017



Research Advisor

**Michael M. Bronstein**



Co-Advisor

**Jonathan Masci**

---

PhD Program Director

**Walter Binder**

---

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Davide Boscaini

Davide Boscaini

Lugano, 16 August 2017

*To Giorgia and my family*



# Abstract

The past decade in computer vision research has witnessed the re-emergence of artificial neural networks (ANN), and in particular convolutional neural network (CNN) techniques, allowing to learn powerful feature representations from large collections of data. Nowadays these techniques are better known under the umbrella term *deep learning* and have achieved a breakthrough in performance in a wide range of image analysis applications such as image classification, segmentation, and annotation.

Nevertheless, when attempting to apply deep learning paradigms to 3D shapes one has to face fundamental differences between images and geometric objects. The main difference between images and 3D shapes is the non-Euclidean nature of the latter. This implies that basic operations, such as linear combination or convolution, that are taken for granted in the Euclidean case, are not even well defined on non-Euclidean domains. This happens to be the major obstacle that so far has precluded the successful application of deep learning methods on non-Euclidean geometric data.

The goal of this thesis is to overcome this obstacle by extending deep learning techniques (including, but not limiting to CNNs) to non-Euclidean domains. We present different approaches providing such extension and test their effectiveness in the context of shape similarity and correspondence applications. The proposed approaches are evaluated on several challenging experiments, achieving state-of-the-art results significantly outperforming other methods.

To the best of our knowledge, this thesis presents different original contributions. First, this work pioneers the generalization of CNNs to discrete manifolds. Second, it provides an alternative formulation of the spectral convolution operation in terms of the windowed Fourier transform to overcome the drawbacks of the Fourier one. Third, it introduces a spatial domain formulation of convolution operation using patch operators and several ways of their construction (geodesic, anisotropic diffusion, mixture of Gaussians). Fourth, at the moment of publication the proposed approaches achieved state-of-the-art results in different computer graphics and vision applications such as shape descriptors and correspondence.



# Contents

<b>Contents</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Related work . . . . .	3
1.3 Main contribution . . . . .	6
1.4 Publications . . . . .	8
<b>2 Background</b>	<b>11</b>
2.1 Intuition . . . . .	11
2.2 Differential geometry . . . . .	12
2.2.1 Manifolds . . . . .	12
2.2.2 Riemannian metric . . . . .	12
2.2.3 Intrinsic and extrinsic properties . . . . .	13
2.3 Calculus on manifolds . . . . .	14
2.3.1 Functional spaces on manifolds . . . . .	14
2.3.2 Intrinsic gradient and divergence operators . . . . .	14
2.3.3 Laplace-Beltrami operator . . . . .	15
2.4 Spectral analysis on manifolds . . . . .	15
2.4.1 Fourier basis and transform . . . . .	16
2.4.2 Spectral convolution . . . . .	17
2.5 Heat diffusion on manifolds . . . . .	17
2.5.1 Isotropic heat diffusion . . . . .	18
2.6 Discretization . . . . .	18
2.6.1 Discrete manifolds . . . . .	19
2.6.2 Discrete functions and operators . . . . .	19
2.7 Spectral descriptors . . . . .	23
2.7.1 Heat kernel signature . . . . .	23
2.7.2 Wave kernel signature . . . . .	24

2.7.3	Optimal spectral descriptor . . . . .	26
2.8	Shape correspondence . . . . .	28
2.8.1	Functional maps . . . . .	28
<b>3</b>	<b>Deep Learning on Euclidean data</b>	<b>33</b>
3.1	Introduction to machine learning . . . . .	33
3.1.1	Examples of machine learning tasks . . . . .	33
3.1.2	Unsupervised and supervised learning algorithm . . . . .	34
3.1.3	Training . . . . .	35
3.2	Support vector machines . . . . .	37
3.3	Challenges motivating deep learning . . . . .	39
3.4	Deep Learning . . . . .	41
3.4.1	Artificial neural networks . . . . .	41
3.4.2	Feedforward neural networks . . . . .	42
3.5	Training deep models . . . . .	45
3.5.1	Back-propagation . . . . .	46
3.5.2	Stochastic gradient descent . . . . .	47
3.6	Convolutional neural networks . . . . .	47
3.6.1	Properties of convolutional neural networks . . . . .	50
<b>4</b>	<b>Frequency domain intrinsic deep learning methods</b>	<b>55</b>
4.1	Spectral methods . . . . .	55
4.1.1	Spectral convolutional neural networks . . . . .	56
4.2	Spectrum-free methods . . . . .	60
4.2.1	Chebyshev convolutional neural networks . . . . .	60
<b>5</b>	<b>Hybrid frequency-spatial intrinsic deep learning methods</b>	<b>63</b>
5.1	Windowed Fourier transform . . . . .	63
5.2	Localized spectral convolutional neural networks . . . . .	66
<b>6</b>	<b>Spatial intrinsic deep learning methods</b>	<b>71</b>
6.1	Geodesic convolutional neural networks . . . . .	73
6.2	Anisotropic diffusion descriptors . . . . .	77
6.3	Anisotropic diffusion convolutional neural networks . . . . .	82
6.4	Mixture model convolutional neural networks . . . . .	83
<b>7</b>	<b>Learning shape descriptors with intrinsic deep learning</b>	<b>87</b>
7.1	Intrinsic convolutional neural networks . . . . .	88
7.2	Learning local shape descriptors . . . . .	90
7.2.1	Performance evaluation . . . . .	92



7.2.2	Datasets . . . . .	92
7.2.3	LSCNN experiments and results . . . . .	93
7.2.4	GCNN experiments and results . . . . .	96
7.2.5	ADD experiments and results . . . . .	99
7.3	Learning shape retrieval . . . . .	103
7.3.1	GCNN experiments and results . . . . .	104
<b>8</b>	<b>Learning shape correspondence with intrinsic deep learning</b>	<b>107</b>
8.1	Shape correspondence as a classification problem . . . . .	107
8.1.1	Correspondence refinement . . . . .	109
8.1.2	Performance evaluation . . . . .	111
8.1.3	Datasets . . . . .	111
8.2	GCNN experiments and results . . . . .	113
8.3	ADD experiments and results . . . . .	114
8.4	ACNN experiments and results . . . . .	115
8.5	MoNet experiments and results . . . . .	118
<b>9</b>	<b>Conclusions and future work</b>	<b>123</b>
9.1	Future work . . . . .	124
	<b>Bibliography</b>	<b>127</b>



# Chapter 1

## Introduction

### 1.1 Motivation

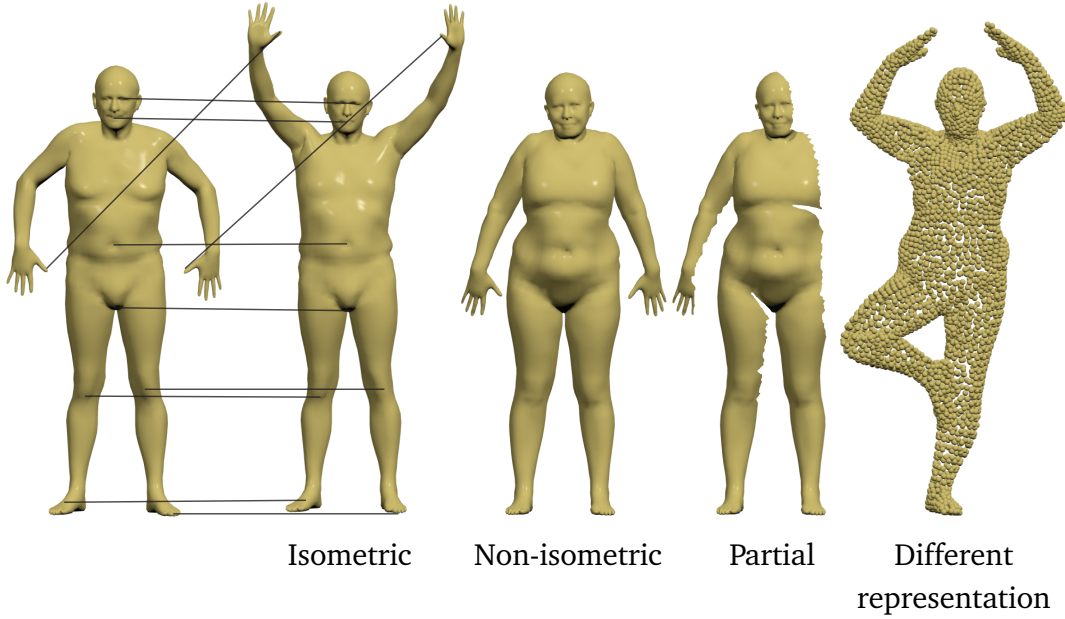
Shape analysis deals with the problem of defining automatic algorithms able to analyze, process, and classify 3D shapes. In other words, it aims to make computers able to “understand” 3D objects.

To pursue this ambitious goal, shape analysis research covers various aspects related to 3D shapes, ranging from reconstruction [NIH<sup>+</sup>11] to registration [LSP08; CK15], recognition [SZPY12], and retrieval [KLM<sup>+</sup>12], just to name a few. Interestingly, all these different applications boil down to two basic problems: defining a concept of shape *similarity*, and finding a *correspondence* between shapes.

The common way to tackle such problems is to consider *local descriptors*. A local descriptor assigns to a point on the shape a vector in some multi-dimensional feature space representing the local structure of the surface around that point. In this way, the problem of defining whether two points are similar or not boils down to a simple vector comparison.

Typically, one wishes a descriptor that is *discriminative* (highlight distinctive attributes), *robust* (invariant with respect to noise and deformations), *compact* (use a small number of dimensions), and *computationally efficient*.

Figure 1.1 shows the typical classes of noise and deformations an ideal descriptor should be robust to. The most common class of deformations is represented by *quasi-isometric* deformations, modelling the different poses a shape can assume (same shape, different poses). A more challenging class of deformations is represented by the class of *non-isometric* deformations, where shapes can be affected by elastic deformations (different shapes, different poses). Moreover, with the recent development of 3D acquisition technologies, a new class of deformations which



*Figure 1.1.* A common requirement for shape descriptors and correspondence applications is the robustness to some classes of deformations. Here we depicted the most interesting ones. *From left to right:* isometric deformations, non-isometric deformations, partial views, and different representations (in this case, a point cloud, which is a different representation w.r.t. the triangular meshes on its left).

is imperative to consider is related to acquisition artifacts such as missing parts, geometric, and topological noise. Finally, local descriptors should be independent from the representation of the shape: ideally we would like to be able to compare e.g. triangular meshes with point clouds.

Traditional approaches define local descriptors according to hand-crafted axiomatic models exploiting specific properties of the 3D shapes, such as invariant properties of surface normals [JH99; ZBVH09; STDS14] or spectral properties [SOG09; GBAL09; ASC11].

The increasing complexity and variety of the deformations considered makes the construction of axiomatic models rather elusive: to cope with these new requirements new paradigms should be considered.

The growth of publicly available 3D shape data brings the attention to machine learning approaches which allow learning invariance to complex transformations from examples (“modelling by example” paradigm). Simply put, while it is hard to model what makes a man look like a man, one can get many examples of man shapes and use a sufficiently complex generic model to learn the “man model” from the data.

Inspired by the incredible success that deep learning methods have brought to speech recognition, language translation and natural language processing [HS97; GSC99; GFGS06; GLF<sup>+</sup>09; WSC<sup>+</sup>16; BGLL17], as well as to computer vision [LKF10; CMMS11; CMS12; CGGS12; KSH12; CGGS13; FCNL13; TYRW14; SZ14; SGS15; HZRS16], it appears natural to use such techniques in shape analysis applications as well.

However, so far these techniques did not see almost any usage in the shape analysis community. The main reason resides in the fact that while speech, images, or video can be modelled as signals defined on, respectively, 1D, 2D, and 3D Euclidean domains, shapes are represented as non-Euclidean domains. The non-Euclidean nature of such data implies that there are no such familiar properties as global parametrization, common system of coordinates, vector space structure, or shift invariance. Consequently, basic operations such as linear combination or convolution that are taken for granted in the Euclidean case are not even well-defined on non-Euclidean domains.

This is the major obstacle that so far has precluded deep learning methods from producing a breakthrough in shape analysis applications as well. The goal of this thesis is to overcome this difficulty by extending deep learning techniques, and in particular convolutional neural networks, to non-Euclidean domains in the form of 3D shapes.

## 1.2 Related work

**Shape descriptors** There is a plethora of literature on hand-crafted local descriptors, and we refer the reader to a recent survey for a comprehensive overview [LGB<sup>+</sup>13]. Below we will briefly review the ones most pertinent to this thesis.

Early works on shape descriptors such as spin images [JH99], shape distributions [OFCD02], and integral volume descriptors [MCH<sup>+</sup>06] were based on *extrinsic* structures that are invariant under Euclidean transformations.

The following generation of shape descriptors used *intrinsic* structures such as geodesic distances [EK03] that are preserved by isometric deformations.

Another class of approaches proposed to extend successful image descriptors such as SIFT [Low04], HOG [DT05], MSER [MCUP04], and shape context [BMP00] to non-Euclidean domains (see e.g. [ZBVH09; DMAMS10; KBLB12], respectively).

Spectral descriptors exploit the geometric properties arising from the eigenfunctions and eigenvalues of the Laplace-Beltrami operator of the surface [BBG94; CL06; Lév06]. Popular methods include shapeDNA [RWP06], global point sig-

nature (GPS) [Rus07], heat kernel signatures (HKS) [SOG09; GBAL09], their scale-invariant version (SI-HKS) [BK10], wave kernel signatures (WKS) [ASC11], and heat kernel maps [OMMG10].

Following the recent trend in the image analysis domain, where hand-crafted descriptors are abandoned in favour of learning approaches, several machine learning frameworks have been proposed in the geometry processing community as well. In particular, Litman and Bronstein [LB14] showed that heat and wave kernel signatures [SOG09; GBAL09; ASC11] can be considered as particular parametric families of transfer functions applied to the Laplacian eigenvalues and proposed to learn an optimal transfer function leading to optimal spectral descriptors. Finally, Corman et al. [COC14] proposed to learn suitable descriptors providing optimal correspondence through the functional maps framework.

**Shape correspondence** Traditional correspondence approaches try to find a *point-wise* matching between (a subset of) the points on two or more shapes.

*Minimum-distortion methods* establish the matching by minimizing some structure distortion, which can include similarity of local features [OMMG10; ASC11; ZBVH09], geodesic distances [MS05; BBK06; CK15], diffusion distances [CLL<sup>+</sup>05], a combination thereof [TKR08], or higher-order structures [ZWW<sup>+</sup>10]. Typically, the computational complexity of such methods is high, and there have been several attempts to alleviate the computational complexity using hierarchical [SY11] or subsampling [TBW<sup>+</sup>11] methods. Several approaches formulate the correspondence problem as quadratic assignment and employ different relaxations thereof [Ume88; LH05; RBA<sup>+</sup>12; CK15; KKBL15].

*Embedding methods* try to exploit some assumption on the shapes (e.g. approximate isometry) in order to parametrize the correspondence problem with a small number of degrees of freedom. Elad and Kimmel [EK01; EK03] used multi-dimensional scaling to embed the geodesic metric of the matched shapes into a low-dimensional Euclidean space, where alignment of the resulting *canonical forms* is then performed by simple rigid matching (ICP) [CM92; BM92]. The works of [MHK<sup>+</sup>08; SK14] used the Laplacian eigenfunctions as embedding coordinates and performed matching directly in the eigenspace. Lipman et al. [LD11; KLCF10; KLF11] used conformal embeddings into disks and spheres to parametrize correspondences between homeomorphic surfaces as Möbius transformations.

As opposed to point-wise correspondence methods, *soft correspondence* approaches assign a point on one shape to more than one point on the other. Several methods formulated soft correspondence as a mass-transportation prob-

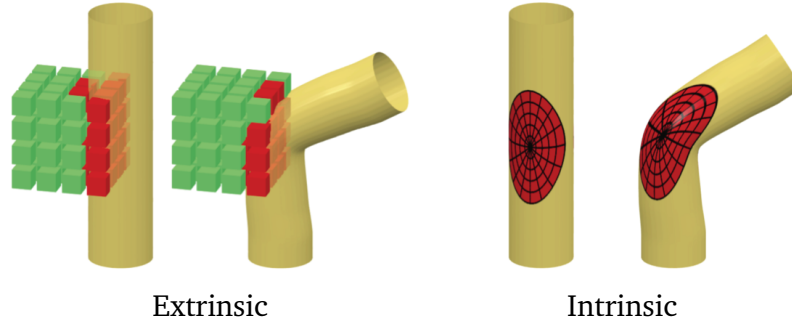


Figure 1.2. Illustration of the difference between extrinsic and intrinsic deep learning methods on geometric data. *Left*: extrinsic methods such as volumetric CNNs [WSK<sup>+</sup>15] treat 3D geometric data in its Euclidean representation. Such a representation is not invariant to deformations (e.g., in the example above, the filter that responds to features on a straight cylinder would not respond to a bent one). *Right*: in an intrinsic representation, the filter is applied to some data on the surface itself, thus being invariant to deformations.

lem [Mém11; SNB<sup>+</sup>12]. Ovsjanikov et al. [OBCS<sup>+</sup>12] introduced the *functional correspondence* framework, modeling the correspondence as a linear operator between spaces of functions on two shapes, which has an efficient representation in the Laplacian eigenbases. This approach was extended in several follow-up works [PBB<sup>+</sup>13; KBBV15; ADK16; RCB<sup>+</sup>17].

Finally, a recent trend is to use machine learning techniques, such as random forests, to learn correspondences [SSK<sup>+</sup>13; RRBW<sup>+</sup>14].

**Extrinsic deep learning** Many deep learning techniques successfully working on images were tried “as is” on 3D geometric data, represented for this purpose in some way “digestible” by standard frameworks. Su et al. [SMKLM15] used CNNs applied to range images obtained from multiple views of 3D objects for retrieval and classification tasks. Wei et al. [WHC<sup>+</sup>16] used view-based representation to find correspondence between non-rigid shapes. Wu et al. [WSK<sup>+</sup>15] used volumetric CNNs applied to rasterized volumetric representation of 3D shapes.

The main drawback of such approaches is their treatment of geometric data as Euclidean structures (see Figure 1.2). First, for complex 3D objects, Euclidean representations such as depth images or voxels may lose significant parts of the object or its fine details, or even break its topological structure (in particular, due to computational reasons, the volumetric CNNs [WSK<sup>+</sup>15] used a  $64 \times 64 \times 64$  cube, allowing only a very coarse representation of 3D geometry). Second, Euclidean representations are not intrinsic, and vary as the result of pose or deformation of

the object. Achieving invariance to shape deformations, a common requirement in many applications, is extremely hard with the aforementioned methods and requires complex models and huge training sets due to the large number of degrees of freedom involved in describing non-rigid deformations.

### 1.3 Main contribution

The main contribution of our work is a principled framework, called *intrinsic deep learning*, allowing to extend deep learning techniques, and in particular CNNs, to non-Euclidean data in the form of non-rigid 3D shapes.

In this thesis, we present different instances of the intrinsic deep learning framework. For each one of them we highlight its advantages and drawbacks and evaluate its performance in shape descriptors, retrieval, and correspondences tasks.

Chapter 5 presents an intrinsic CNN construction called *localized spectral convolutional neural network* (LSCNN) [BMM<sup>+</sup>15]. LSCNN is a generalization of the *spectral convolutional neural network* (SCNN) framework [BZSL13; HBL15], which resorts to a Fourier transform to define the convolution in the spectral domain. In LSCNN, instead, we propose an alternative formulation of the spectral convolution operation on non-Euclidean manifolds in terms of *windowed Fourier transform* [SRV16] rather than the traditional Fourier transform of SCNN. Since the definition of the Fourier transform on non-Euclidean manifolds is basis-dependent, SCNN filters learned on a shape are not generalizable to another one. One of the key advantages of LSCNN over SCNN is that the introduction of the windowed Fourier transform overcomes such drawback and allows to learn filters generalizable across shapes sharing some common geometric structure (i.e. which bases does not differ arbitrary). The spectral formulation of LSCNN allows its application to different shape representations, such as meshes and point clouds. A drawback of this approach is its memory and computation requirements, as each window needs to be explicitly produced.

Chapter 6 presents a category of intrinsic CNNs where the extension of the convolution operation on non-Euclidean manifolds occurs in the spatial domain, rather than in the spectral one. The first key contribution of this chapter consists in the interpretation of the convolution operation as correlation between a learnable filter and a local representation of the signal analogous to pixels windows on images, called *patch*. The second key contribution is the definition of the charting procedure allowing to extract such patches as an operator acting point-wise on the signal defined on the manifold, called *patch operator*. Different patch operators



lead to different methods and can deal with different manifold discretizations. In particular we present four different intrinsic deep learning methods: GCNN [MBBV15], ADD [BMR<sup>+</sup>16], ACNN [BMRB16], and MoNet [MBM<sup>+</sup>17], which we will briefly overview in the following.

*Geodesic convolutional neural network* (GCNN) [MBBV15] is characterized by a *geodesic patch operator* extracting the patches in the form of a local system of geodesic polar coordinates similar to [KBLB12]. Being a spatial approach, GCNN does not suffer from the limitations of spectral approaches (SCNN [BZSL13; HBL15], LSCNN [BMM<sup>+</sup>15]) in terms of generalization across different domains. GCNN suffers from two main drawbacks: the charting procedure is limited to meshes only and there is no guarantee that the patches extracted are always topologically meaningful.

*Anisotropic diffusion descriptors* (ADD) [BMR<sup>+</sup>16] allows to learn optimal anisotropic spectral descriptors, i.e. an extension of the optimal spectral descriptors [LB14] capturing local directional structures as well. One of the main advantages of ADD over OSD is that anisotropy allows to disambiguate intrinsic symmetries.

*Anisotropic convolutional neural network* (ACNN) [BMRB16] employs the same anisotropic diffusion kernels introduced in [BMR<sup>+</sup>16] but as spatial weighting functions, to define an *anisotropic patch operator* extracting a local intrinsic representation of functions defined on the manifold. Unlike ADD, ACNN is a convolutional neural network architecture. Compared to the geodesic patch operator, the anisotropic patch operator construction is much simpler, does not depend on the injectivity radius of the manifold, and is not limited to triangular meshes. Overall, ACNN combines all the best properties of the previous approaches without inheriting their drawbacks.

*Mixture model convolutional neural network* (MoNet) [MBM<sup>+</sup>17] is the first unified framework allowing to generalize CNN architectures to non-Euclidean domains (graphs and manifolds) and learn local, stationary, and compositional task-specific features. The main contribution of MoNet is the formulation of the patch operator as a local weighted average of the signal with a set of learnable kernels. We show that previous intrinsic CNN methods, such as GCNN and ACNN, can be casted as particular instances of MoNet corresponding to patch operators with fixed hand-crafted kernels rather than learnable ones.

In Chapter 7, we show the application of the intrinsic deep learning constructions (Chapters 4, 5, and 6) to the problem of learning discriminative and robust local shape descriptors. A key contribution of this chapter is the formulation of the problem of learning local shape descriptors in terms of the siamese framework.

Intrinsic deep learning methods achieve state-of-the-art results on a variety of challenging datasets, significantly outperforming the other approaches.

Finally, in Chapter 8 we show the application of the intrinsic deep learning constructions (Chapters 4, 5, and 6) to the problem of learning dense correspondences between deformable shapes. We rely on the definition of the problem of learning correspondences as a classification problem to an abstract label space, as shown by Rodolà et al. [RRBW<sup>+</sup>14]. Intrinsic deep learning methods achieve state-of-the-art results on some of the most difficult recent correspondence benchmarks.

Remarkably, the works presented in this thesis were among the first in the emerging field of *geometric deep learning*, dealing with techniques attempting to generalize (structured) deep neural models to non-Euclidean domains such as graphs and manifolds.

## 1.4 Publications

This thesis is mainly based on the publications listed below:

- F. Monti\*, D. Boscaini\*, J. Masci, E. Rodolà, J. Svoboda, M. M. Bronstein, *Geometric deep learning on graphs and manifolds using mixture model CNNs*, Proc. CVPR 2017 (\* indicates equal contribution) · *oral presentation*
- D. Boscaini, J. Masci, E. Rodolà, M. M. Bronstein, *Learning shape correspondence with anisotropic convolutional neural networks*, Proc. NIPS, pp. 3189–3197, 2016
- D. Boscaini, J. Masci, E. Rodolà, M. M. Bronstein, D. Cremers, *Anisotropic spectral descriptors*, Computer Graphics Forum 35(2), pp. 431–441, 2016 · *oral presentation at EG 2015*
- J. Masci\*, D. Boscaini\*, M. M. Bronstein, P. Vandergheynst, *Geodesic convolutional neural networks on Riemannian manifolds*, International IEEE Workshop on 3D Representation and Recognition (3dRR), 2015 · *oral presentation*
- D. Boscaini, J. Masci, S. Melzi, M. M. Bronstein, U. Castellani, P. Vandergheynst, *Learning class-specific descriptors for deformable shapes using localized spectral convolutional networks*, Computer Graphics Forum 34(5), pp. 13–23, 2015 · *oral presentation at SGP 2015*

The author contributed to several other papers during his PhD studies, which are not included in this dissertation for the sake of the presentation coherence:

- D. Boscaini, D. Eynard, D. Kourounis, M. M. Bronstein, *Shape-from-operator: recovering shapes from intrinsic operators*, Computer Graphics Forum 34(2), pp. 265–274, 2015 · *oral presentation at EG 2015*
- D. Boscaini, U. Castellani, *A sparse coding approach for local-to-global 3D shape description*, The Visual Computer, 2014
- D. Boscaini, R. Girdziusas, M. M. Bronstein, *Coulomb shapes: using electrostatic forces for deformation-invariant shape representation*, Eurographics Workshop on 3D Object Retrieval (3DOR), pp. 9–15, 2014



# Chapter 2

## Background

In this chapter we provide a rigorous definition and a formal derivation of the necessary mathematical tools to describe the contributions of this thesis. For a comprehensive treatise of the related topics we refer the reader to [DC76; DCF92; BBI01; Ros97].

### 2.1 Intuition

The intuitive way to think about a 3D shape is as the boundary surface of a three-dimensional object: think of such a surface as made by an infinitely thin elastic material that adapts to the geometry of the underlying 3D object and is able to “follow” its deformations.

There exist two main categories of shapes, depending on which deformations they allow: *rigid* shapes (e.g. furnitures, mechanical objects), and *non-rigid* shapes (e.g. animals, humans). As the name implies, rigid shapes allow only rigid (or Euclidean) deformations such as rotations and translations, while non-rigid shapes allow a much wider category of deformations (e.g. isometric, conformal). If we think to the 3D shape of a man, examples of non-rigid deformations include different poses, heights and musculatures. In this thesis we are interested in the more challenging category of non-rigid shapes. In the following we will refer to shape and non-rigid shape as synonyms.

Data on 3D shapes are represented as functions or vector fields defined over their surface. Examples of such data include geometric information (e.g. the surface curvature), semantic information (e.g. names of the shape parts), or color information (surface texture).

## 2.2 Differential geometry

### 2.2.1 Manifolds

A  $d$ -dimensional *manifold*  $\mathcal{X}$  is a topological space where each point  $x \in \mathcal{X}$  has a neighborhood  $U_x$  that is homeomorphic to a  $d$ -dimensional Euclidean space. Such a homeomorphism  $\varphi: U_x \rightarrow \mathbb{R}^d$  is called *chart*. If the transition between charts from overlapping neighborhoods is smooth, then  $\mathcal{X}$  is called *smooth manifold*.

Let  $x \in \mathcal{X}$  be a point on the manifold and let  $\gamma_1: (-\epsilon_1, \epsilon_1) \rightarrow \mathcal{X}$ ,  $\gamma_2: (-\epsilon_2, \epsilon_2) \rightarrow \mathcal{X}$  be two curves on the manifold passing through  $x$ , i.e.  $\gamma_1(0) = \gamma_2(0) = x$ . Such curves are said *equivalent* if and only if there exists a chart  $\varphi: U_x \rightarrow \mathbb{R}^d$  such that  $(\varphi \circ \gamma_1)'(0) = (\varphi \circ \gamma_2)'(0)$ . A *tangent vector* to the manifold  $\mathcal{X}$  at the point  $x$  is defined as the equivalence class of smooth curves on  $\mathcal{X}$  passing through  $x$ . The *tangent space* at  $\mathcal{X}$  in  $x$ , denoted by  $T_x \mathcal{X}$ , is defined as the set containing the tangent vectors at  $x \in \mathcal{X}$  and does not depend on the choice of the chart  $\varphi$ . The disjoint union of tangent spaces at all points is referred to as the *tangent bundle* and denoted by  $T\mathcal{X}$ .

A geometric realization of a manifold as a subset of an *ambient space*  $\mathbb{R}^{d'}$  is called *embedding* and the manifold is said to be *embedded* in that space. An embedded manifold  $\mathcal{X} \subset \mathbb{R}^{d'}$  can be realized by a smooth function  $\psi: U \subseteq \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$ , mapping intrinsic coordinates  $\mathbf{u} = (u_1, \dots, u_d)$  to global ones  $\mathbf{x} = (x_1, \dots, x_{d'})$ . If the function  $\psi$  is provided, tangent vectors to the embedded manifold  $\mathcal{X}$  at the point  $\mathbf{x} = \psi(\mathbf{u})$  can be defined as the derivatives  $\frac{\partial}{\partial u_i} \psi(\mathbf{u})$ ,  $i = 1, \dots, d$ . If the derivatives  $\frac{\partial}{\partial u_1} \psi(\mathbf{u}), \dots, \frac{\partial}{\partial u_d} \psi(\mathbf{u})$  are linearly independent, they form a basis for the tangent space  $T_{\psi(\mathbf{u})} \mathcal{X}$ . The vector  $\mathbf{n}(\mathbf{x})$  orthogonal to  $T_x \mathcal{X}$  is the *normal* of  $\mathcal{X}$  at  $\mathbf{x}$ .

3D shapes can be modeled as 2-dimensional smooth manifolds  $\mathcal{X}$  embedded in  $\mathbb{R}^3$ , possibly with boundary  $\partial \mathcal{X}$ .

### 2.2.2 Riemannian metric

A *Riemannian metric* is an inner product  $\langle \cdot, \cdot \rangle_{T_x \mathcal{X}}: T_x \mathcal{X} \times T_x \mathcal{X} \rightarrow \mathbb{R}$  between tangent vectors at  $x \in \mathcal{X}$  which depends smoothly on  $x$ .

On an embedded manifold  $\mathcal{X} \subseteq \mathbb{R}^{d'}$ , the Riemannian metric admits the closed form expression

$$\langle \mathbf{v}, \mathbf{w} \rangle_{T_{\psi(\mathbf{u})} \mathcal{X}} = \mathbf{v}^\top \mathbf{G} \mathbf{w},$$

where  $\mathbf{v}, \mathbf{w} \in T_{\psi(\mathbf{u})}$  and the matrix  $\mathbf{G}$  is the *first fundamental form*, i.e. a  $d \times d$

positive semi-definite matrix  $\mathbf{G} = (g_{i,j})$ , s.t.

$$g_{i,j}(u) = \left\langle \frac{\partial}{\partial u_i} \psi(\mathbf{u}), \frac{\partial}{\partial u_j} \psi(\mathbf{u}) \right\rangle.$$

The Riemannian metric allows performing local measurements of angles, distances, and volumes on the manifold  $\mathcal{X}$ . For instance, the length of a curve  $\gamma: [0, 1] \rightarrow \mathcal{X}$ , can be defined as

$$\ell(\gamma) = \int_0^1 \sqrt{\langle \gamma'(t), \gamma'(t) \rangle_{T_{\gamma(t)}\mathcal{X}}} dt,$$

while the  $d$ -dimensional volume element is given by

$$d\mu = |\mathbf{G}|^{1/2} du_1 \dots du_d.$$

A manifold equipped with a Riemannian metric is called a *Riemannian manifold*.

### 2.2.3 Intrinsic and extrinsic properties

Quantities which are expressible entirely in terms of the first fundamental form, and therefore independent on the way the surface is embedded in the ambient space, are called *intrinsic*. Intrinsic properties are particularly interesting because they are invariant to isometric (metric preserving) deformations of the manifold.

Conversely, properties related to the specific embedding of the manifold in the ambient space are called *extrinsic* and can be defined in terms of the *second fundamental form*.

For 2-dimensional embedded manifolds, the second fundamental form describes how the surface locally differs from a plane. For the point  $\mathbf{x} \in \mathcal{X}$  it can be represented as the  $2 \times 2$  matrix  $\mathbf{H} = (h_{i,j})$ ,  $i, j = 1, 2$ , s.t.

$$h_{i,j} = \left\langle \frac{\partial^2}{\partial u_i \partial u_j} \psi(\mathbf{x}), \mathbf{n}(\mathbf{u}) \right\rangle,$$

where  $\mathbf{n}(\mathbf{x})$  is the normal vector at  $\mathbf{x}$  and is defined as

$$\mathbf{n}(\mathbf{x}) = \frac{\frac{\partial}{\partial u_1} \psi(\mathbf{u}) \times \frac{\partial}{\partial u_2} \psi(\mathbf{u})}{\left\| \frac{\partial}{\partial u_1} \psi(\mathbf{u}) \times \frac{\partial}{\partial u_2} \psi(\mathbf{u}) \right\|}.$$

The eigenvalues  $K_m, K_M$  of the second fundamental form  $\mathbf{H}$  are called *principal curvatures*, while the corresponding eigenvectors  $\mathbf{v}_m, \mathbf{v}_M$  are called *principal curvature directions* and form an orthonormal basis on the tangent plane.

## 2.3 Calculus on manifolds

As we saw in the beginning of the chapter, data on non-Euclidean manifolds can be defined as functions defined thereon. The principal way to study such functions is to extend classical calculus tools to these non-Euclidean structures.

### 2.3.1 Functional spaces on manifolds

Functions on manifolds can be divided in two categories: a *scalar field* is a smooth function  $f: \mathcal{X} \rightarrow \mathbb{R}$ , while a *vector field*  $F: \mathcal{X} \rightarrow \mathbb{R}^d$  associates to each point  $x \in \mathcal{X}$  a vector  $F(x) \in \mathbb{R}^d$  in some  $d$ -dimensional Euclidean space. Among the different vector fields, a *tangent vector field*  $F: \mathcal{X} \rightarrow T\mathcal{X}$  maps points  $x \in \mathcal{X}$  on the manifold to tangent vectors  $F(x) \in T_x\mathcal{X}$ . The notion of *tangent vector field* allows to generalize the classical notion of derivation to non-Euclidean manifolds by providing a formal definition of infinitesimal displacements on the manifold (the equivalent of  $x \mapsto x + dx$  in  $\mathbb{R}$ ).

We denote by  $L^2(\mathcal{X})$  the Hilbert space of  $L^2$  scalar fields on the manifold  $\mathcal{X}$ , where the inner product between two functions  $f, g \in L^2(\mathcal{X})$  is defined as

$$\langle f, g \rangle_{L^2\mathcal{X}} = \int_{\mathcal{X}} f(x)g(x)dx.$$

Similarly, we denote by  $L^2(T\mathcal{X})$  the Hilbert space of  $L^2$  tangent vector fields on the manifold, and with

$$\langle F, G \rangle_{L^2(T\mathcal{X})} = \int_{\mathcal{X}} \langle F(x), G(x) \rangle_{T_x\mathcal{X}} dx$$

the inner product between tangent vector fields. In both cases  $dx$  denotes the volume element induced by the Riemannian metric.

### 2.3.2 Intrinsic gradient and divergence operators

The *differential* of  $f$  is an operator  $df: T\mathcal{X} \rightarrow \mathbb{R}$  acting on tangent vector fields: at each point  $x$ , the differential can be represented as a linear form  $df(x) = \langle \nabla f(x), \cdot \rangle_{T_x\mathcal{X}}$  which takes as inputs tangent vectors  $F(x) \in T_x\mathcal{X}$  and produces scalars. The input tangent vector  $F(x)$  models a small displacement around  $x$ , while the output scalar  $df(x)F(x) = \langle f(x), F(x) \rangle_{T_x\mathcal{X}}$  measures how the function  $f$  changes because of such displacement, and can be thought of as an extension of the notion of the classical directional derivative.



The operator  $\nabla f : L^2(\mathcal{X}) \rightarrow L^2(T\mathcal{X})$  is called *intrinsic gradient* and can be interpreted as an extension of the classical gradient in the sense that it provides the direction (tangent vector on  $T_x\mathcal{X}$ ) in which  $f$  changes the most at point  $x$ .

The *intrinsic divergence*  $\operatorname{div} F : L^2(T\mathcal{X}) \rightarrow L^2(\mathcal{X})$  can be defined as the adjoint operator (up to a sign) of the intrinsic gradient, in the sense that

$$\langle F, \nabla f \rangle_{L^2(T\mathcal{X})} = \langle -\operatorname{div} F, f \rangle_{L^2(\mathcal{X})}. \quad (2.1)$$

In summary, the intrinsic gradient is an operator acting on scalar fields and producing tangent vector fields, while the intrinsic divergence is an operator acting on vector fields and producing scalar fields.

### 2.3.3 Laplace-Beltrami operator

The *Laplacian* (or, more precisely, the *Laplace-Beltrami operator*)  $\Delta : L^2(\mathcal{X}) \rightarrow L^2(\mathcal{X})$  can be defined by combining the intrinsic gradient and divergence operators,  $\Delta f = -\operatorname{div}(\nabla f)$ , where  $f \in L^2(\mathcal{X})$  is a scalar field on the manifold.

The Laplacian plays a crucial role in signal processing and machine learning on non-Euclidean domains: first, its eigenfunctions generalize the classical Fourier basis to manifolds, allowing to perform spectral analysis on them; second, it models different physical phenomena on the manifold, such as heat diffusion; third, it is intrinsic, therefore it is invariant to isometric deformations of the manifold. In the following paragraph we will provide more details about the Laplacian most relevant properties for this thesis.

## 2.4 Spectral analysis on manifolds

The Laplacian is a self-adjoint positive semi-definite operator, therefore (on a compact domain) it admits an eigendecomposition

$$\Delta \phi_i = \lambda_i \phi_i, \quad i = 1, 2, \dots \quad (2.2)$$

with real eigenvalues  $\lambda_1 < \lambda_2 \leq \dots$  and orthogonal eigenfunctions  $\{\phi_i\}_{i \geq 1}$ , where the orthogonality is intended w.r.t. the standard  $L^2$  inner product, i.e.  $\langle \phi_i, \phi_j \rangle_{L^2(\mathcal{X})} = \delta_{i,j}$ , where

$$\langle \phi_i, \phi_j \rangle_{L^2(\mathcal{X})} = \int_{\mathcal{X}} \phi_i(x) \phi_j(x) dx.$$

In particular, the eigenfunctions orthogonality stems from the self-adjointness of the Laplacian, i.e.

$$\begin{aligned}
 \langle f, \Delta g \rangle_{L^2(\mathcal{X})} &= \langle f, -\operatorname{div} \nabla g \rangle_{L^2(\mathcal{X})} \\
 &\stackrel{(\dagger)}{=} \langle \nabla f, \nabla g \rangle_{L^2(T\mathcal{X})} \\
 &= \langle \nabla g, \nabla f \rangle_{L^2(T\mathcal{X})} \\
 &= \langle g, -\operatorname{div} \nabla f \rangle_{L^2(\mathcal{X})} \\
 &= \langle g, \Delta f \rangle_{L^2(\mathcal{X})},
 \end{aligned}$$

where in  $(\dagger)$  we use the fact that  $-\operatorname{div}$  is adjoint to  $\nabla$ .

The non-negativity of the eigenvalues, instead, stems from its positive semi-definiteness,

$$\begin{aligned}
 \langle f, \Delta f \rangle_{L^2(\mathcal{X})} &= \int_{\mathcal{X}} f(x) \Delta f(x) dx \\
 &= \int_{\mathcal{X}} \|\nabla f(x)\|^2 dx \\
 &\geq 0,
 \end{aligned}$$

because of the non-negativity of the integrand.

The set of the Laplacian eigenvalues is known as its *spectrum*. In general, quantities expressed in terms of the Laplacian eigenvalues and eigenfunctions are commonly referred to as *spectral*.

If the boundary  $\partial \mathcal{X}$  of the manifold  $\mathcal{X}$  is non empty, the Equation (2.2) should be endowed with additional boundary conditions, such as Dirichlet boundary conditions  $\phi_i(x) = 0$ , or Neumann boundary conditions  $\langle \nabla \phi_i(x), \mathbf{n}(x) \rangle = 0$ , where  $x \in \partial \mathcal{X}$  and  $\mathbf{n}$  denotes the normal vector to the boundary.

### 2.4.1 Fourier basis and transform

The Laplacian eigenfunctions form a basis that spans  $L^2(\mathcal{X})$ , which can be interpreted as a generalization of the standard Fourier basis to the non-Euclidean domain  $\mathcal{X}$  ([Tau95; LZ10]).

In fact, by noting that

$$-\frac{d^2}{dx^2} e^{i\omega x} = \omega^2 e^{i\omega x},$$

where  $i$  denotes the imaginary unit, it is easy to verify that the elements  $e^{i\omega x}$  of the Fourier basis on  $\mathbb{R}$  are the eigenfunctions of the Euclidean Laplacian operator  $-d^2/dx^2$ .

It is important to emphasize that the basis formed by the Laplacian eigenfunctions is intrinsic due to the intrinsic construction of the Laplacian operator.

By resorting to the interpretation of the Laplacian eigenfunctions as the Fourier basis of  $L^2(\mathcal{X})$ , we can decompose any function  $f \in L^2(\mathcal{X})$  into its *Fourier series* as

$$f(x) = \sum_{i \geq 1} \langle f, \phi_i \rangle_{L^2(\mathcal{X})} \phi_i(x), \quad (2.3)$$

where the *Fourier coefficients*  $\hat{f}_i = \langle f, \phi_i \rangle_{L^2(\mathcal{X})}$  can be thought of as the forward Fourier transform, while the inverse Fourier transform can be obtained as the summation  $\sum_{i \geq 1} \hat{f}_i \phi_i(x)$ .

### 2.4.2 Spectral convolution

The *convolution theorem* states that the convolution between two functions can be expressed as the product of their Fourier transforms, in the sense that

$$\widehat{f * g} = \hat{f} \cdot \hat{g},$$

where  $\hat{f}$  denotes the Fourier transform of  $f$ . By resorting to this interpretation as a definition rather than a property and by replacing the classical forward and inverse Fourier transforms with the ones introduced above, we can generalize the convolution operation to non-Euclidean domains as

$$(f * g)(x) = \sum_{i \geq 1} \langle f, \phi_i \rangle_{L^2(\mathcal{X})} \langle g, \phi_i \rangle_{L^2(\mathcal{X})} \phi_i(x).$$

The main drawback of this definition of the convolution operation w.r.t. the classical one is the lack of *shift-invariance*. As a result, the spectral convolution between the signal  $f$  and a filter  $g$  is position-dependent, i.e. it can vary drastically at different points.

## 2.5 Heat diffusion on manifolds

As already mentioned in Section 2.3.3, the Laplacian appears in many differential equations describing different physical phenomena. In particular, in this section we will focus on the *heat diffusion equation*, describing the heat propagation on manifolds.

### 2.5.1 Isotropic heat diffusion

If we think about the manifold  $\mathcal{X}$  as being made of an ideal material with constant heat conduction at every point, the heat diffusion on  $\mathcal{X}$  is governed by the *isotropic* heat equation

$$f_t(x, t) = -c\Delta f(x, t) = -\operatorname{div}(c\nabla f(x, t)), \quad (2.4)$$

where  $f(x, t)$  denotes the temperature at point  $x$  at time  $t$ , the proportion coefficient  $c$  is referred to as the *thermal diffusivity constant*, and appropriate boundary conditions are applied if necessary. Equation (2.4) is a mathematical description of the *Newton's cooling law*, stating that the rate of change of the temperature of an object (lhs) is proportional to the difference between its own temperature and the temperature of the surrounding (rhs).

Given some initial heat distribution  $f_0(x) = f(x, 0)$ , the solution of heat equation (2.4) at time  $t$  is obtained by applying the *heat operator*  $H^t = e^{-t\Delta}$  to the initial condition  $f_0$ , obtaining

$$f(x, t) = H^t f_0(x) = \int_{\mathcal{X}} f_0(x') h_t(x, x') dx', \quad (2.5)$$

where  $h_t(x, x')$  is called the *heat kernel*.

In the Euclidean case, the heat kernel is shift-invariant, i.e.  $h_t(x, x') = h_t(x - x')$ , therefore the solution of the heat equation (2.5) can be obtained as the convolution  $f(x, t) = (f_0 * h_t)(x)$ . In the more general non-Euclidean case, the heat kernel is not anymore shift-invariant and Equation (2.5) can be interpreted as a generalized convolution.

In the spectral domain, the heat kernel has a closed form solution and can be expressed as

$$h_t(x, x') = \sum_{i \geq 1} e^{-t\lambda_i} \phi_i(x) \phi_i(x'). \quad (2.6)$$

## 2.6 Discretization

In the continuous domain, 3D shapes are modelled as two-dimensional manifolds (Section 2.2.1), data on such 3D shapes is described in terms of functions defined thereon (Section 2.3.1), and interesting properties of such data can be processed by operators such as the Laplacian (Section 2.3.3).

In practical applications, however, we need to provide discrete approximations of such continuous quantities: 3D shapes are represented by discrete structures, while functions and operators acting on them are represented by vectors and matrices, respectively.

### 2.6.1 Discrete manifolds

There exist different discretizations and, depending on the specific task, one is preferable to another. In this thesis we will focus on three discretizations: *point clouds*, *triangular meshes*, and *raster scans*.

The starting point for any discretization procedure consists in a sampling of the continuous surface  $\mathcal{X}$  at  $n$  points  $x_1, \dots, x_n$ . What distinguishes different discretizations is the set of additional structures considered on top of such sampling.

A point cloud is the most basic manifold discretization and just consists of embedding coordinates  $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^3$  for each sample.

A triangular mesh is a topological representation that can be defined as a triplet  $(\mathcal{V}, \mathcal{E}, \mathcal{F})$ , where the set of vertices  $\mathcal{V} = \{1, \dots, n\}$  contains the indices of the sampling  $\{x_1, \dots, x_n\}$ , vertices are connected by edges  $\mathcal{E}$ , and edges are connected by triangular faces  $\mathcal{F}$  such that if  $(i, j, k) \in \mathcal{F}$ , then  $(i, j), (i, k), (k, j) \in \mathcal{E}$ .

To be a valid discretization of a manifold, a triangular mesh should satisfy the following properties: the faces incident to every vertex form a closed or an open fan, every edge must be shared by exactly two triangular faces; if the manifold has a boundary, any boundary edge must belong to exactly one triangle. A triangular mesh satisfying such properties is called *manifold triangular mesh*.

An embedding of a mesh can be provided by assigning some embedding coordinates  $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^3$  to its vertices. The geometric realization of a triangular mesh through an embedding in the three-dimensional Euclidean space can be thought as a polyhedral surface with triangular faces approximating the underlying continuous manifold  $\mathcal{X}$ .

Raster scans are a particular kind of manifold triangular mesh, where the embedding coordinates of the sampling  $x_1, \dots, x_n$  are specified as the depth values of a partial view of the 3D shape sampled over a regular grid, i.e.  $\mathbf{x}_i = (u, v, f(u, v))$ , where  $f(u, v)$  is the depth of the point  $(u, v) = (n\Delta u, m\Delta v)$ ,  $n, m \in \mathbb{N}$ . Raster scans are particularly relevant discretizations because they are the typical data captured by 3D cameras, such as Microsoft Kinect and Intel RealSense.

### 2.6.2 Discrete functions and operators

A real-valued function  $f: \mathcal{X} \rightarrow \mathbb{R}$  can be discretized by sampling its values over the vertices  $x_1, \dots, x_n$  and can be represented as the  $n$ -dimensional vector  $\mathbf{f} = (f(x_1), \dots, f(x_n))^T$ .

Discrete operators acting on  $n$ -dimensional vectors can be defined as  $n \times n$  matrices. The goal of this section is to provide a discretization of the Laplacian

using the finite elements method.

Any function  $f \in L^2(\mathcal{X})$  can be approximated with the linear combination

$$f(x) = \sum_i f_i \zeta_i(x), \quad (2.7)$$

where  $f_i = f(x_i)$ , and  $\zeta_i(x)$  are piecewise linear *hat functions*, i.e.

$$\zeta_i(x) = \begin{cases} 1 & \text{if } x = x_i, \\ 0 & \text{otherwise.} \end{cases}$$

In particular, given the function  $g(x) = \Delta f(x)$ , we have

$$g(x) = \sum_i g_i \zeta_i(x). \quad (2.8)$$

The *weak form* of Equation (2.8) can be obtained as

$$\langle g(x), \zeta_j(x) \rangle = \sum_i g_i \langle \zeta_i(x), \zeta_j(x) \rangle. \quad (2.9)$$

By Green's identity (or integration by parts) we can rewrite the left term of Equation (2.9) as

$$\langle g(x), \zeta_j(x) \rangle = \langle \Delta f, \zeta_j(x) \rangle = -\langle \nabla f, \nabla \zeta_j(x) \rangle, \quad (2.10)$$

and by plugging Equation (2.7) in Equation (2.10), we obtain

$$\langle \Delta f, \zeta_j(x) \rangle = -\langle \nabla f, \nabla \zeta_j(x) \rangle = -\sum_i f_i \underbrace{\langle \nabla \zeta_i(x), \nabla \zeta_j(x) \rangle}_{-w_{i,j}} = (\mathbf{W}f)_j.$$

The right term of Equation (2.9), instead, can be rewritten as

$$g_i \underbrace{\langle \zeta_i(x), \zeta_j(x) \rangle}_{m_{i,j}} = (\mathbf{M}g)_j.$$

By replacing the terms in Equation (2.9) with the ones just derived, we get  $\mathbf{W}f = \mathbf{M}g$ , or in other terms  $g = \mathbf{M}^{-1}\mathbf{W}f = \mathbf{L}f$ , where the matrix  $\mathbf{L}$  represents the discretization of the Laplacian operator  $\Delta$  (Section 2.3.3). The matrix  $\mathbf{W} = (w_{i,j})$  is called *stiffness* matrix, while the matrix  $\mathbf{M} = (m_{i,j})$  is called *mass* matrix.

Stiffness matrix entries are defined as  $w_{i,j} = -\langle \nabla \zeta_i(x), \nabla \zeta_j(x) \rangle$ . On the triangle  $(i, j, k)$  the linear hat function  $\zeta_i(x)$  is equal to 0 on the edge  $x_j - x_k$  and increases to 1 on the vertex  $x_i$ . Therefore  $\nabla \zeta_i(x)$  is constant and points towards

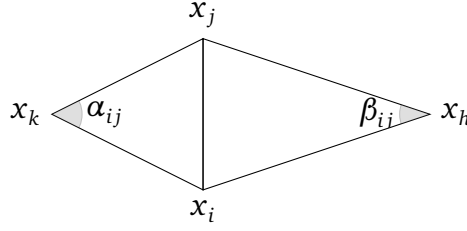


Figure 2.1. Notations: edge  $ij$  has length  $\ell_{ij}$ . Angles  $\alpha_{ij}$  and  $\beta_{ij}$  are opposite to edge  $ij$ . Triangle  $ikj$  has area  $A_{ijk}$ .

the vertex  $i$  from the opposite edge  $x_j - x_k$ . Denoted with  $h_i$  the distance from the edge  $x_j - x_k$  to the vertex  $x_i$ , we have  $\|\nabla\zeta_i(x)\| = 1/h_i$  and by using basic trigonometry arguments we can conclude that

$$\|\nabla\zeta_i(x)\| = \frac{1}{h_i} = \frac{\|x_j - x_k\|}{2A_{ijk}},$$

which implies

$$\langle \nabla\phi_i(x), \nabla\phi_j(x) \rangle = \frac{\|x_j - x_k\|}{2A_{ijk}} \frac{\|x_i - x_k\|}{2A_{ijk}} \cos \theta,$$

where  $\theta$  is the angle between  $\nabla\zeta_i(x)$  and  $\nabla\zeta_j(x)$ , and  $A_{ijk}$  is the area of the triangle  $ijk$ .

Now  $\alpha_{ij} = \pi - \theta$ , therefore  $\cos \theta = \cos(\pi - \alpha_{ij}) = -\cos \alpha_{ij}$ , therefore

$$\langle \nabla\zeta_i(x), \nabla\zeta_j(x) \rangle = -\frac{\|x_j - x_k\|}{2A_{ijk}} \frac{\|x_i - x_k\|}{2A_{ijk}} \cos \alpha_{ij}.$$

Observing that  $\|x_i - x_k\|$  can also be written as  $h_j / \sin \alpha_{ij}$ , we obtain

$$\langle \nabla\zeta_i(x), \nabla\zeta_j(x) \rangle = -\frac{\|x_j - x_k\|}{2A_{ijk}} \frac{h_j}{2A_{ijk}} \cot \alpha_{ij}.$$

Now  $\|x_j - x_k\| h_j = 2A_{ijk}$ , therefore

$$\langle \nabla\zeta_i(x), \nabla\zeta_j(x) \rangle = -\frac{\cot \alpha_{ij}}{2A_{ijk}}.$$

Similarly, on the other triangle  $(i, j, h)$ , we have

$$\langle \nabla\zeta_i(x), \nabla\zeta_j(x) \rangle = -\frac{\cot \beta_{ij}}{2A_{ijh}}.$$

Therefore, we can conclude that

$$w_{ij} = \frac{1}{2}(\cot \alpha_{ij} + \cot \beta_{ij}).$$

The integral in the mass matrix definition,

$$m_{ij} = \int_X \zeta_i(x) \zeta_j(x) dx,$$

can be approximated by a sum over the mesh triangles  $T \in \mathcal{T}$

$$m_{ij} = \sum_{T \in \mathcal{T}} \int_T \zeta_i(x) \zeta_j(x) dx. \quad (2.11)$$

Since the integrand vanishes on all the triangles that do not contain the edge  $(i, j)$ , Equation (2.11) boils down to

$$m_{ij} = \int_{T_{ijk}} \phi_i(x) \phi_j(x) dx + \int_{T_{ijh}} \phi_i(x) \phi_j(x) dx.$$

By using a *Gaussian quadrature rule*, we obtain

$$\int_{T_{ijk}} \zeta_i(x) \zeta_j(x) dx \approx \frac{A_{ijk}}{3} (\zeta_i(x_1) \zeta_j(x_1) + \zeta_i(x_2) \zeta_j(x_2) + \zeta_i(x_3) \zeta_j(x_3)),$$

where  $x_1, x_2, x_3$ , which are simply the midpoints of the edges of  $T_{ijk}$ .

Since  $\zeta_i$  are simple linear functions over each triangle, the previous quadrature rule leads to

$$m_{ij} = \sum_{T \in \mathcal{N}(i) \cap \mathcal{N}(j)} \begin{cases} \frac{A_{ijk}}{6}, & \text{if } i = j, \\ \frac{A_{ijk}}{12}, & \text{otherwise,} \end{cases}$$

where  $\mathcal{N}(i), \mathcal{N}(j)$  are the 1-ring neighbourhood of  $i, j$ , respectively.

Typically the mass matrix is lumped to obtain the diagonal matrix  $\mathbf{A} = \text{diag}(\mathbf{a}) = \text{diag}(a_1, \dots, a_n)$ , where

$$a_i = \frac{1}{3} \sum_{jk:ijk \in \mathcal{F}} A_{ijk}$$

is the local area element at vertex  $i$ .

In summary, the discrete version of the Laplacian is an  $n \times n$  matrix  $\mathbf{L} = \mathbf{A}^{-1} \mathbf{W}$ , where  $\mathbf{W}$  is the cotangent weights matrix and  $\mathbf{A}$  is the lumped mass matrix.

The eigendecomposition of the Laplacian operator  $\mathbf{L} = \mathbf{A}^{-1} \mathbf{W}$  is defined as

$$\mathbf{W} \Phi = \mathbf{A} \Phi \Lambda,$$

where  $\Phi = (\phi_1, \dots, \phi_n)$  is an  $n \times n$  matrix whose columns contains the eigenvectors of  $L$  and  $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$  is the diagonal matrix containing the corresponding eigenvalues of  $\mathbf{L}$ .



## 2.7 Spectral descriptors

Early works on shape descriptors such as spin images [JH99], shape distributions [OFCD02], and integral volume descriptors [MCH<sup>+</sup>06] were based on *extrinsic* structures that are invariant under Euclidean transformations.

Extrinsic information turns out to be quite poor when working with non-rigid shapes: a simple change of pose of a person leads to a completely different extrinsic structure.

For this reason, in the context of non-rigid shapes, the attention moved to *intrinsic* descriptors, and in particular to *spectral* descriptors.

Spectral descriptors are based on the Laplacian eigendecomposition and their popularity is owed to several nice properties. First, they are *intrinsic* by construction and thus invariant to isometric deformations. Second, they are efficiently computable. Third, they can be constructed on shapes in different representations such as meshes or point clouds, provided a valid discretization of the Laplacian for such representations is available.

Notable examples in this family include *heat kernel signature* [SOG09; GBAL09] and *wave kernel signature* [ASC11], which we will review in the following paragraphs.

### 2.7.1 Heat kernel signature

Sun et al. [SOG09] and Gebal et al. [GBAL09] proposed to construct intrinsic descriptors by considering the diagonal of the heat kernel (2.6),

$$h_t(x, x) = \sum_{i \geq 1} e^{-t\lambda_i} \phi_i^2(x)$$

also known as the *autodiffusivity function*. The physical interpretation of autodiffusivity is the amount of heat remaining at point  $x$  after time  $t$ .

Different diffusion times capture different geometric contents. For instance, for small diffusion times, the autodiffusivity  $h_t(x, x)$  is related to the Gaussian curvature  $K(x)$  by virtue of the Taylor expansion

$$h_t(x, x) = \frac{1}{4\pi t} + \frac{K(x)}{12\pi} + \mathcal{O}(t).$$

Sun et al. [SOG09] and Gebal et al. [GBAL09] defined the *heat kernel signature* (HKS) of dimension  $q$  at point  $x$  by sampling the autodiffusivity function at some fixed times  $t_1, \dots, t_q$ ,

$$\mathbf{f}(x) = \left( h_{t_1}(x, x), \dots, h_{t_q}(x, x) \right)^\top. \quad (2.12)$$

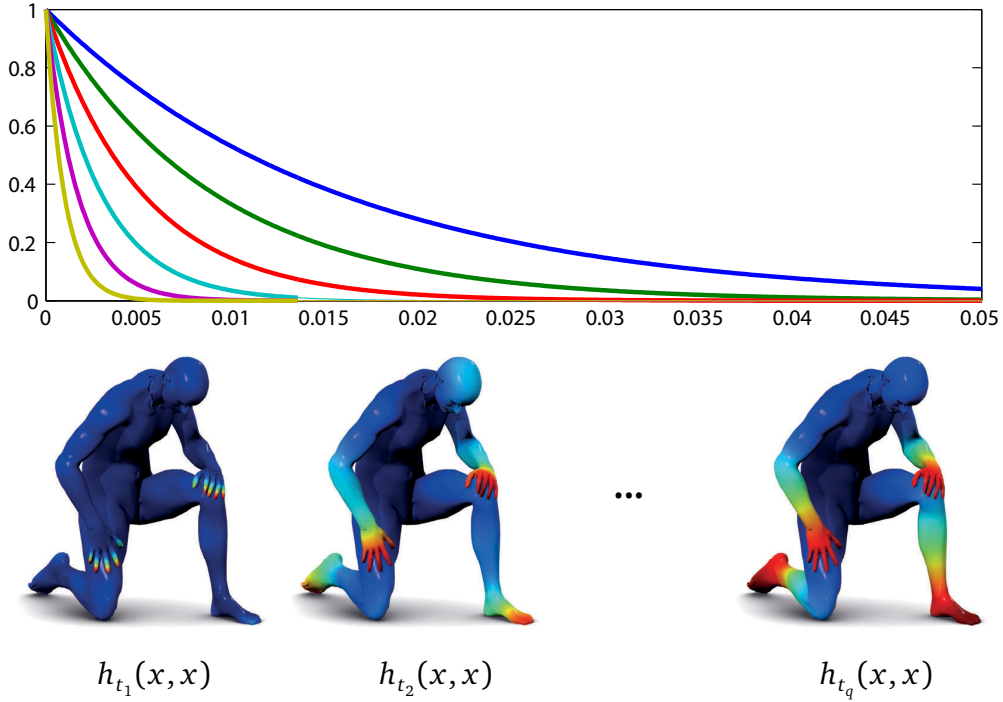


Figure 2.2. The HKS is defined as the application of low-pass filters to the Laplacian eigenfunctions. *Top*: Plot of such low-pass transfer functions  $\tau(\lambda) = e^{-t\lambda}$  for different values  $t_1, \dots, t_q$  of the diffusion time  $t$ . *Bottom*: Visualization of the HKS corresponding to the transfer functions above.

A notable drawback of HKS stemming from the use of low-pass filters is poor spatial localization (by the uncertainty principle, good localization in the Fourier domain results in a bad localization in the spatial domain).

### 2.7.2 Wave kernel signature

Aubry et al. [ASC11] considered a different physical model of a quantum particle on the manifold, whose behaviour is governed by the *Schrödinger equation*,

$$\left(i\Delta + \frac{\partial}{\partial t}\right)\psi(x, t) = 0, \quad (2.13)$$

where  $\psi(x, t)$  is the complex wave function capturing the particle behaviour.

Assuming that the particle oscillates at frequency  $\lambda$  drawn from a probability distribution  $\pi(\lambda)$ , the solution of (2.13) can be expressed in the Fourier domain

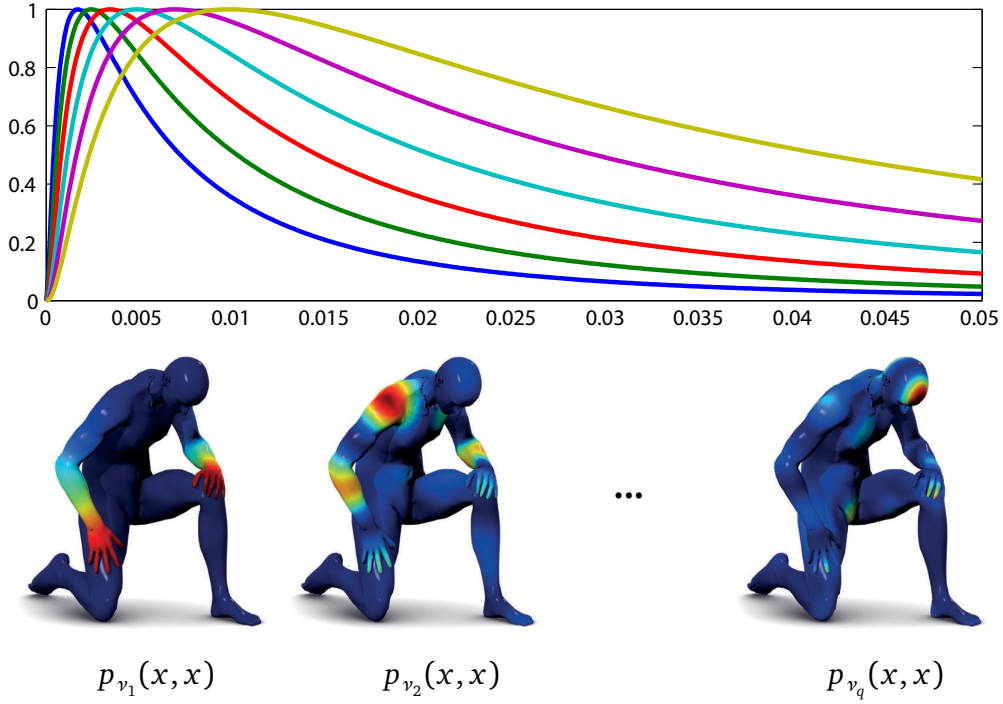


Figure 2.3. The WKS is defined as the application of band-pass filters to the Laplacian eigenfunctions. *Top*: Plot of such band-pass transfer functions  $\tau(\lambda) = \pi_\nu(\lambda)$  for different values  $\nu_1, \dots, \nu_q$  of the mean frequency  $\nu$ . *Bottom*: Visualization of the WKS corresponding to the transfer functions above.

as

$$\psi(x, t) = \sum_{i \geq 1} e^{i\lambda_i t} \pi(\lambda_i) \phi_i(x).$$

The probability of finding the particle at point  $x$  is given by

$$p(x) = \lim_{T \rightarrow \infty} \int_0^T |\psi(x, t)|^2 dt = \sum_{i \geq 1} \pi^2(\lambda_i) \phi_i^2(x), \quad (2.14)$$

and depends on the initial frequency distribution  $\pi(\lambda)$ .

Aubry et al. [ASC11] suggested to consider a log-normal frequency distribution

$$\pi_\nu(\lambda) = \exp\left(-\frac{(\log \nu - \log \lambda)^2}{2\sigma^2}\right)$$

with mean frequency  $\nu$  and standard deviation  $\sigma$ . The corresponding  $q$ -dimensional *wave kernel signature* (WKS) is defined as

$$\mathbf{f}(x) = \left(p_{\nu_1}(x), \dots, p_{\nu_q}(x)\right)^\top, \quad (2.15)$$

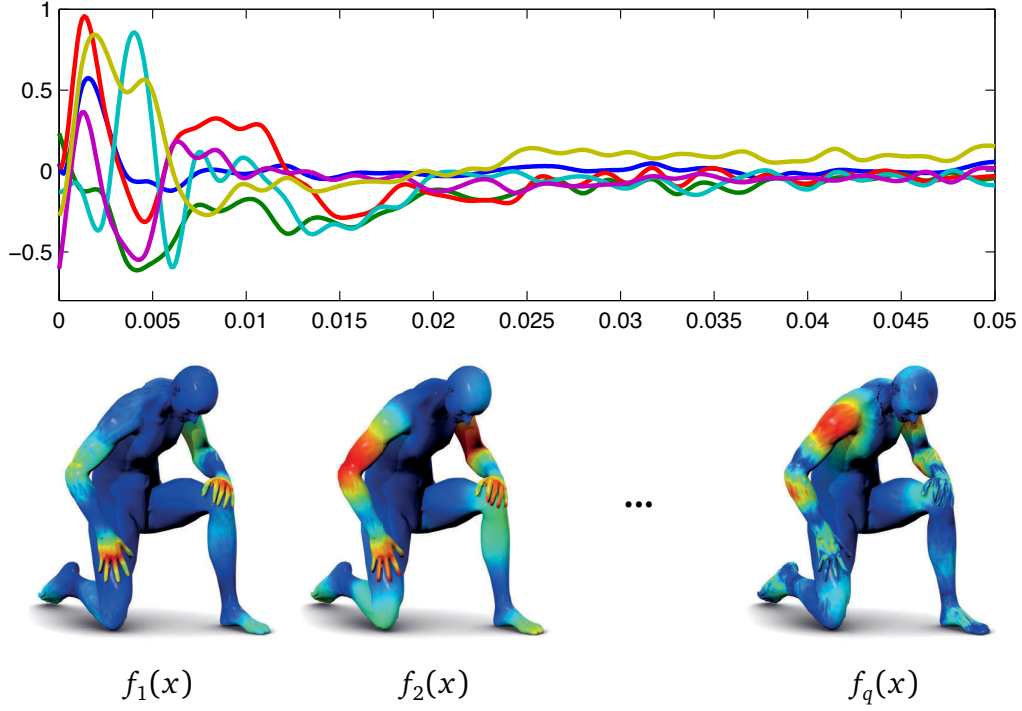


Figure 2.4. The OSD is defined as the application of parametric filters to the Laplacian eigenfunctions, whose parameters can be learned resorting to metric learning techniques. *Top*: Optimal transfer functions learned from examples. *Bottom*: Visualization of the OSD corresponding to the transfer functions above.

where  $p_\nu(x)$  is the probability (2.14) corresponding to the initial log-normal frequency distribution with mean frequency  $\nu$ , and  $\nu_1, \dots, \nu_q$  are some logarithmically-sampled frequencies.

### 2.7.3 Optimal spectral descriptor

Litman and Bronstein [LB14] noticed that a generic  $q$ -dimensional spectral descriptor can be defined as

$$\mathbf{f}(x) = \sum_{i \geq 1} \tau(\lambda_i) \phi_i^2(x) \approx \sum_{i=1}^k \tau(\lambda_i) \phi_i^2(x), \quad (2.16)$$

where  $\tau(\lambda) = (\tau_1(\lambda), \dots, \tau_q(\lambda))^\top$  is a bank of “transfer functions” acting on the Laplacian eigenvalues. Different transfer functions capture different spectral properties and lead to different spectral descriptors.

In particular, HKS [SOG09; GBAL09] and WKS [ASC11] are special cases of Equation (2.16) corresponding to the choice of low-pass filters

$$\tau_t(\lambda) = e^{-t\lambda}$$

and band-pass filters

$$\tau_\nu(\lambda) = \exp\left(-\frac{(\log \nu - \log \lambda)^2}{2\sigma^2}\right),$$

respectively.

Since the two most popular spectral descriptors, HKS [SOG09; GBAL09] and WKS [ASC11], differ only in the choice of the filter  $\tau$ , Litman and Bronstein [LB14] proposed to consider a parametric family of filters expressed as

$$\tau_l(\lambda) = \sum_{l'=1}^m a_{l,l'} \beta_{l'}(\lambda), \quad (2.17)$$

in the B-spline basis  $\beta_1(\lambda), \dots, \beta_m(\lambda)$ , where  $a_{l,l'}$ ,  $l = 1, \dots, q$ ,  $l' = 1, \dots, m$ , are the parametrization coefficients.

Plugging Equation (2.17) into Equation (2.16), one can express the  $q$ th component of the spectral descriptor as

$$f_l(x) = \sum_{l'=1}^m a_{l,l'} \underbrace{\sum_{i \geq 0} \beta_{l'}(\lambda_i) \phi_i^2(x)}_{g_{l'}(x)}, \quad (2.18)$$

where

$$\mathbf{g}(x) = (g_1(x), \dots, g_m(x))^\top, \quad (2.19)$$

is called *geometry vector* and depends only on the intrinsic geometry of the shape.

Equation (2.18) provides a parametrization of a spectral descriptor (2.16) in terms of the  $q \times m$  matrix of coefficients  $\mathbf{A} = (a_{l,l'})$  and can be written in matrix form as  $\mathbf{f}(x) = \mathbf{A}\mathbf{g}(x)$ .

Instead of resorting to hand-crafted models to drive the choice of the parameter matrix  $\mathbf{A}$ , Litman and Bronstein [LB14] proposed to learn one from data. In particular, they showed that learning such parameters can be casted as a Mahalanobis-type metric learning problem, where task-specific loss is minimized on a suitable training set. The descriptors obtained with the learned parameters are called *optimal spectral descriptors* (OSD).

## 2.8 Shape correspondence

Another basic problem in shape analysis consists in defining *shape correspondence*. A correspondence between two shapes is a mapping from the vertices of one shape to the vertices of the other one.

Traditional approaches, such as *blended intrinsic maps* [KLF11], provide the correspondence between two shapes as a point-wise map.

Recently, Ovsjanikov et al. [OBCS<sup>+</sup>12] proposed to work at a more abstract level by considering correspondences between functions defined on the manifolds rather than finding a match between the vertices (Figure 2.5). Such framework is called *functional maps* and in the following paragraph we will provide more details about it.

### 2.8.1 Functional maps

Let  $\mathcal{X}$  and  $\mathcal{Y}$  be two manifolds and let us denote by  $t: \mathcal{X} \rightarrow \mathcal{Y}$  a bijective point-wise correspondence between them. The *functional map*  $T: L^2(\mathcal{X}) \rightarrow L^2(\mathcal{Y})$  is defined as

$$T(f) = f \circ t^{-1},$$

where  $f \in L^2(\mathcal{X})$  and  $T(f) \in L^2(\mathcal{Y})$ . Figure 2.5 provides a visualization of the quantities involved in the functional map definition.

Interestingly, the standard point-wise correspondence  $t: \mathcal{X} \rightarrow \mathcal{Y}$  can be recovered from the functional map  $T$  by considering Dirac's delta functions centered at the sampling  $x_1, \dots, x_n \in \mathcal{X}$ .

Mapping functions rather than points gives a simple algebraic structure to the problem. In particular, the functional map  $T$  is linear:

$$\begin{aligned} T(\alpha f + \beta g) &= (\alpha f + \beta g) \circ t^{-1} \\ &= \alpha(f \circ t^{-1}) + \beta(g \circ t^{-1}) \\ &= \alpha T(f) + \beta T(g), \end{aligned}$$

for scalars  $\alpha, \beta \in \mathbb{R}$  and functions  $f, g \in L^2(\mathcal{X})$ .

As anticipated above, the Laplacian eigenfunctions form the Fourier basis for the  $L^2$  functional space on the manifold. Therefore, if we denote by  $\Phi = (\phi_1, \dots, \phi_n)$  and  $\Psi = (\psi_1, \dots, \psi_n)$  the Fourier bases on  $\mathcal{X}$  and  $\mathcal{Y}$ , respectively,

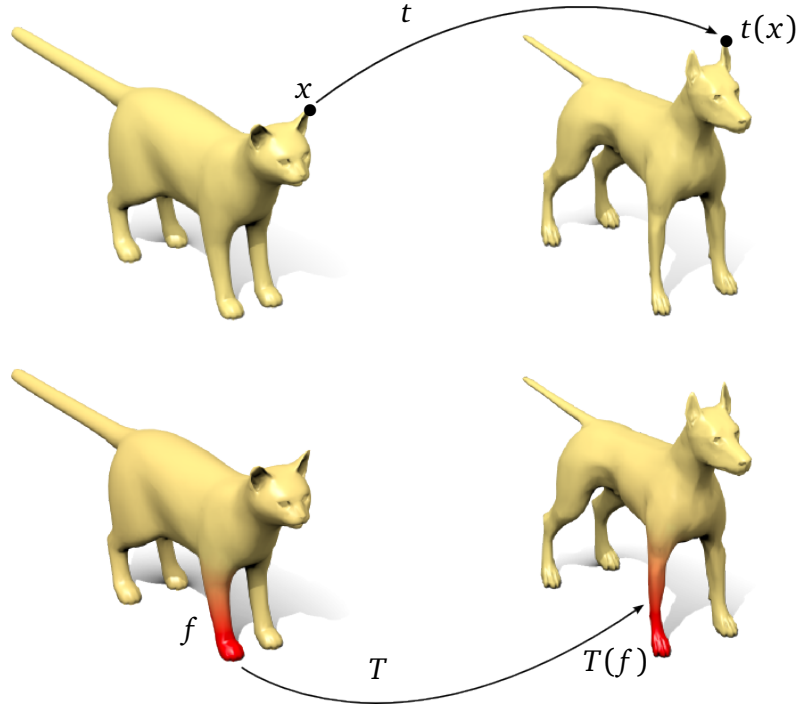


Figure 2.5. Comparison between point-wise correspondence and functional maps. *Top*: a point-wise correspondence  $t: \mathcal{X} \rightarrow \mathcal{Y}$  maps a point  $x \in \mathcal{X}$  to a point  $t(x) \in \mathcal{Y}$ . *Bottom*: a functional map  $T: L^2(\mathcal{X}) \rightarrow L^2(\mathcal{Y})$  maps a function  $f \in L^2(\mathcal{X})$  on the cat to a function  $T(f) \in L^2(\mathcal{Y})$  on the dog. Functions are depicted with red blobs.

we can express the functional map  $T$  using the Fourier expansion as

$$\begin{aligned}
 T(f) &= T\left(\sum_{i \geq 1} \langle f, \phi_i \rangle_{L^2(\mathcal{X})} \phi_i\right) \\
 &= \sum_{i \geq 1} \langle f, \phi_i \rangle_{L^2(\mathcal{X})} T(\phi_i) \\
 &= \sum_{i \geq 1} \langle f, \phi_i \rangle_{L^2(\mathcal{X})} \sum_{j \geq 1} \underbrace{\langle T(\phi_i), \psi_j \rangle_{L^2(\mathcal{Y})}}_{c_{i,j}} \psi_j \\
 &= \sum_{i,j \geq 1} \langle f, \phi_i \rangle_{L^2(\mathcal{X})} c_{i,j} \psi_j,
 \end{aligned}$$

where  $c_{i,j} = \langle T(\phi_i), \psi_j \rangle_{L^2(\mathcal{Y})}$ .

Usually the series of the previous equation is truncated by considering only

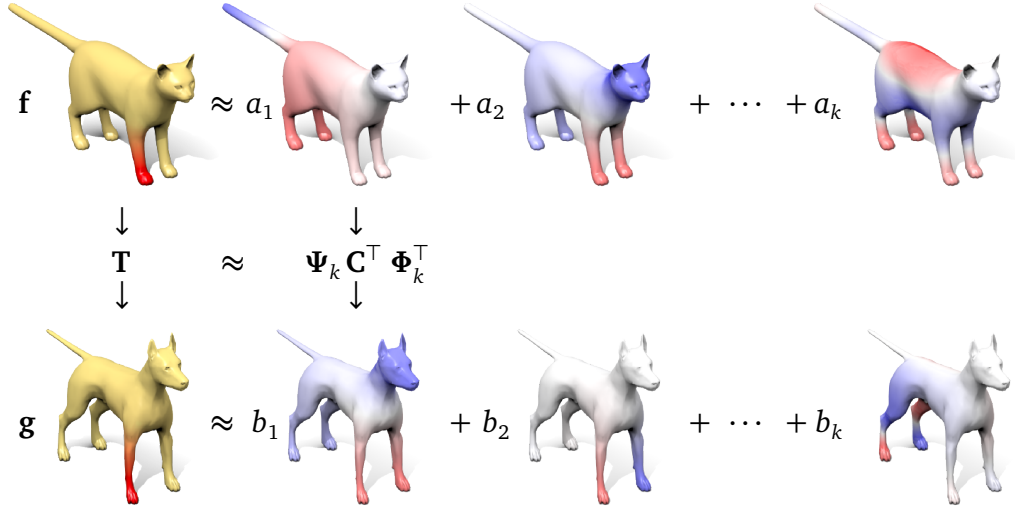


Figure 2.6. A functional map  $\mathbf{T}: L^2(\mathcal{X}) \rightarrow L^2(\mathcal{Y})$  can be defined in the spectral domain as a change of basis matrix  $\mathbf{C}$ , translating Fourier coefficients  $\mathbf{a} = (a_i)$  expressed in the basis  $\Phi$  to Fourier coefficients  $\mathbf{b} = (b_i)$  in the basis  $\Psi$ .

the first  $k$  elements,

$$T(f) \approx \sum_{i,j=1}^k \langle f, \phi_i \rangle_{L^2(\mathcal{X})} c_{i,j} \psi_j.$$

In the spectral domain, the functions  $f$  and  $T(f)$  can be represented as the vectors  $\mathbf{a} = (a_1, \dots, a_k)$  and  $\mathbf{b} = (b_1, \dots, b_k)$ , containing the respective Fourier coefficients. According to this notation, the previous equation becomes

$$b_j = \sum_i a_i c_{i,j},$$

where the  $k \times k$  matrix  $\mathbf{C} = (c_{i,j})$  is independent of  $f$  and is determined by the bases  $\Phi, \Psi$ , and the map  $T$ .

In the discrete setting, the previous equation becomes

$$\mathbf{T}\mathbf{f} = \Psi\mathbf{C}\Phi^\top \mathbf{A}_{\mathcal{X}}\mathbf{f} \approx \Psi_k\mathbf{C}\Phi_k^\top \mathbf{A}_{\mathcal{X}}\mathbf{f} = \mathbf{T}_k\mathbf{f}, \quad (2.20)$$

where  $\mathbf{f}$  is a  $n$ -dimensional vector representing the function  $f$ ,  $\mathbf{A}_{\mathcal{X}}$  is a matrix containing the local area elements of the manifold  $\mathcal{X}$ ,  $\Phi, \Psi$  are the eigenbases of the Laplacian,  $\Psi_k, \Phi_k$  are the truncated eigenbases containing only the first  $k$  eigenvectors, and  $\mathbf{T}_k$  can be thought as a rank  $k$  approximation of the matrix  $\mathbf{T}$ . Figure 2.6 shows a visualization of the quantities involved in Equation (2.20).



Now let us assume that we can provide a set of  $q$  corresponding functions, represented by  $n_{\mathcal{X}} \times q$  and  $n_{\mathcal{Y}} \times q$  matrices  $\mathbf{F} = (\mathbf{f}_1, \dots, \mathbf{f}_q)$  and  $\mathbf{G} = (\mathbf{g}_1, \dots, \mathbf{g}_q)$  satisfying  $\mathbf{T}\mathbf{F} = \mathbf{G}$ . Ovsjanikov et al. [OBCS<sup>+</sup>12] showed that, under this assumption, the functional map  $\mathbf{T}_k$  can be found by solving a linear system in the matrix  $\mathbf{C}$ ,

$$\Psi_k \mathbf{C}_k^\top \Phi_k^\top \mathbf{A}_{\mathcal{X}} \mathbf{F} = \mathbf{G},$$

or, alternatively,

$$\mathbf{G}^\top \mathbf{A}_{\mathcal{Y}} \Psi_k = \mathbf{F}^\top \mathbf{A}_{\mathcal{X}} \Phi_k \mathbf{C}.$$

Assuming  $q \geq k$ , this system is overdetermined and can be solved in the least square sense.



# Chapter 3

## Deep Learning on Euclidean data

### 3.1 Introduction to machine learning

Machine learning is the branch of computer science that deals with the problem of developing algorithms able to “learn” from data. A machine learning algorithm is said to learn from data if it is able to build a model that *explains* the data without being explicitly programmed through specific programming instructions implementing arbitrary axiomatic models. Machine learning is therefore able to overcome the limitations of the traditional axiomatic modelling by introducing a more flexible data-driven approach. This ability is extremely relevant given the large amount of publicly available data (big data) we have to deal with nowadays, making these methods even more important.

#### 3.1.1 Examples of machine learning tasks

By leveraging such data-driven modelling, machine learning is able to tackle problems which are otherwise difficult or infeasible to solve by specific hand-crafted algorithms. Axiomatic models limitations may include lack of knowledge on the task or the impossibility to model axiomatically invariance to certain noise, just to name a few.

Among the different tasks that machine learning can solve, the most common ones are:

- *Classification*, i.e. the problem of identifying to which of  $k$  different categories an observation belongs to. In more detail, a classification algorithm provides a function  $f : \mathbb{R}^n \rightarrow \{1, \dots, k\}$  assigning to a  $n$ -dimensional input vector  $\mathbf{x}$  an integer number  $y = f(\mathbf{x})$  representing one of the  $k$  categories.

If we consider a dataset containing images of birds, an example of a classification task is the identification of the bird species. In such example, the input  $\mathbf{x}$  is an image of a bird (reshaped as an  $n$ -dimensional vector by stacking the pixel brightness values of each color channel) and the output  $y$  is an integer value identifying the species.

In some cases the distinction between the different classes may be rather fuzzy. As a result, we may not be interested in a unique assignment but in a weaker classification. In such cases, the classification algorithm can be modified to output a probability distribution over the  $k$  different classes rather than a single class index, i.e.  $f : \mathbb{R}^n \rightarrow \mathbb{R}^k$ . This variant is referred to as *soft* classification.

- *Regression*, i.e. the problem of predicting continuous values given some input features. In more details, a regression algorithm provides a function  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$  assigning to an  $n$ -dimensional input vector  $\mathbf{x}$  an  $m$ -dimensional output vector  $\mathbf{y} = f(\mathbf{x})$ .

An example of a regression problem is the estimation of house prices from input features such as square footage, number of bedrooms and bathrooms, etc.

- *Synthesis*, i.e. the problem of generating new samples that resemble the input data. For instance, in the context of 3D shapes, one may be interested in automatic ways to generate novel examples, rather than asking an artist to create them manually.

### 3.1.2 Unsupervised and supervised learning algorithm

The typical ingredient in machine learning approaches is the *training set*, i.e. the set of data over which the algorithm is allowed to learn the parameters of a parametric model.

Depending on which kind of information are contained in the training set, we can distinguish between the two most broad categories of machine learning algorithms: *unsupervised* learning and *supervised* learning algorithms.

In the unsupervised learning setting, the training set is composed of samples  $\{f_i \in L^2(\mathbb{R}^d)\}_{i \in I}$  drawn from a possibly unknown data distribution. The goal of unsupervised learning algorithms is to infer useful properties of said data from its samples during the so called *training* stage.

In the supervised setting, instead, we are also given some additional information associated with each element of the training set, such as *labels*. Usually

a label constitutes the desired output value when the algorithm is given in input the sample associated with it. The training set is thus formed by the pairs  $\{(f_i \in L^2(\mathbb{R}^d), y_i \in \mathcal{Y})\}_{i \in I}$  where each sample  $f_i$  is associated with a label  $y_i$ . The typical assumption is that there exists an unknown function  $y: L^2(\mathbb{R}^d) \rightarrow \mathcal{Y}$  from which the labels  $y_i$  are sampled from, i.e. such that  $y_i = y(f_i)$ . The goal of supervised learning algorithms is to recover the unknown function  $y$  or, said differently, to predict the correct labels corresponding to unseen samples. The label space  $\mathcal{Y}$  depends on the application considered. With reference to the notation of Section 3.1.1, in the case of classification  $\mathcal{Y}$  is a discrete space with cardinality equal to the number of classes, i.e.  $|\mathcal{Y}| = k$ . For regression, instead,  $\mathcal{Y} = \mathbb{R}^m$ . In the rest of this thesis we will focus our attention mainly on supervised learning problems.

### 3.1.3 Training

Typically, the model we aim to learn can be represented as a parametric function  $U_{\Theta}$ , where  $\Theta$  represents the parameter set.

The learning process, usually referred to simply as *training*, can be casted as an optimization problem of the model parameters  $\Theta$  by minimizing a cost function  $\ell$  on the training set:

$$\min_{\Theta} \sum_{i \in I} \ell(U_{\Theta}(f_i), y_i),$$

where the choice of  $\ell$  depends on the application at hand. For instance,  $\ell(x, y) = \|x - y\|$ .

The main goal of the training process is to find optimal parameters

$$\Theta^* = \operatorname{argmin}_{\Theta} \sum_{i \in I} \ell(U_{\Theta}(f_i), y_i),$$

such that the corresponding model  $U_{\Theta^*}$  generalizes well on previously unseen data, not just on the training set.

As pointed out by [GBC16], this constitutes a major difference between machine learning and optimization: in a typical optimization problem, we just want to reduce the error on the training set, called *training error*. Here, instead, we also want the generalization error on a new sample, called *test error*, to be low.

More specifically, once the optimal parameters are estimated on the training set, the generalization ability of the machine learning algorithm is tested by measuring the test error on a disjoint dataset, called *test set*.

If the model is sufficiently complex and the training set is representative enough of the underlying data distribution, one expects the parametric function  $U_{\Theta}$  to approximate the unknown function  $y$ , i.e. to find such  $\Theta$  that  $y \approx U_{\Theta}$ .

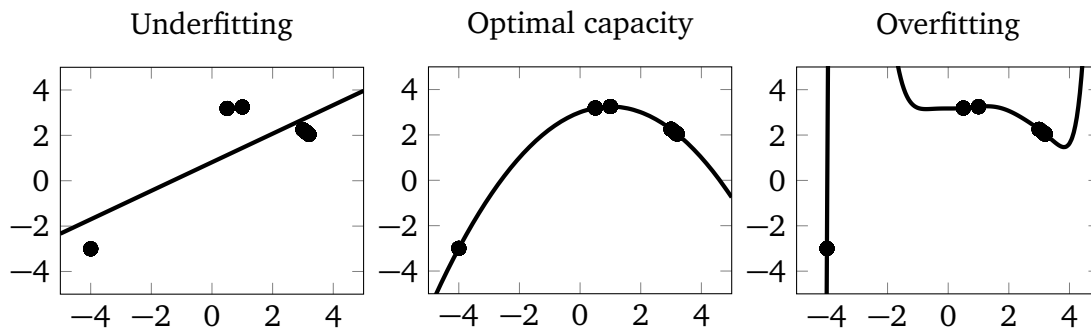


Figure 3.1. Influence of the model capacity in determining the correct polynomial fitting the training data, here represented by the black dots. In this toy example, the training data were sampled from a quadratic function. *Left:* a linear model does not even fit the training data, suffering from underfitting. *Center:* a quadratic model has the correct capacity to both fit the training data and generalizing well enough to unseen points. *Right:* a polynomial of degree 9 suffers from overfitting: the model fits correctly the training points but the underlying quadratic behaviour is not grasped. Figure reproduced from [GBC16].

However, two problems may arise:

- *underfitting*, i.e. the model is not able to capture the main factors of variation of the underlying training data, and, as a result, it cannot reduce the training error;
- *overfitting*, i.e. the model just tries to fit the training data rather than learning the underlying data distribution, resulting in poor generalization. Overfitting occurs in models with small training error but big test error.

How to avoid these two pathological cases is one of the central problems in machine learning. In particular, underfitting happens when the model is not complex enough or, said differently, when it has too few parameters with respect to the size of training data. Overfitting, instead, may happen when the model is too complex. For example, if the model has a number of parameters greater or equal to the observations in the training set, it can simply “memorize” the training samples rather than predicting the distribution from which they were sampled.

More precisely, the informal notion of model complexity used in the previous examples has a more formal counterpart in the notion of *capacity* of the model, i.e. the class of functions a model can represent. A model with low capacity is more prone to underfitting, while a model with high capacity is more likely to suffer from overfitting. By modifying the capacity of the model one can avoid

such pathological behaviours. A model performs best when its capacity is suitable for the complexity of the task it is asked to solve and the size of the training set. See Figure 3.1 for a toy example about the influence of the capacity of the model on its generalization capability.

It is important to note that the notion of capacity of a model is wider than just its number of parameters. In particular, the most common way to modify the capacity of a machine learning model is through *regularization*, i.e. by adding some constraints on the class of functions the model can span.

## 3.2 Support vector machines

One of the most effective supervised learning approaches is represented by *support vector machines* [CV95]. A support vector machine, or simply SVM, aims to provide a separating hyperplane between two categories of data.

More precisely, let us consider a training set of the form

$$\{\mathbf{x}_i \in \mathbb{R}^d, y_i \in \{-1, 1\}\}_{i \in I},$$

where the training samples  $\mathbf{x}_i$  are thought as points in a  $d$ -dimensional vector space and the labels  $y_i$  take only two values depending on which one of the two classes  $\mathbf{x}_i$  belongs to. In such settings, a hyperplane takes the form

$$\langle \mathbf{w}, \mathbf{x} \rangle + b, \quad (3.1)$$

where  $\mathbf{w} \in \mathbb{R}^d$ ,  $b \in \mathbb{R}$  represent the “slope” and the “intercept” of the hyperplane, respectively.

A hyperplane naturally separates the data domain in two semispaces and can thus be used to separate data in two classes: given a hyperplane, indeed, a decision function can be defined as  $y = \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle + b)$ , where a sample  $\mathbf{x}$  is assigned to a class ( $y = 1$ ) or the other ( $y = -1$ ) depending on which side of the hyperplane it falls into.

The basic idea behind linear SVMs is to exploit such property of hyperplanes to build the model

$$U_{\Theta}(\mathbf{x}) = \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle + b), \quad (3.2)$$

where  $\Theta = \{\mathbf{w}, b\}$ . During training, the parameters  $\mathbf{w}, b$  are learned in order to find the best separating hyperplane, i.e. the hyperplane with the maximum margin of separation between training points.

Figure 3.2 (b) shows a successful application of the SVM algorithm in the context of binary classification. The two classes are represented by white and

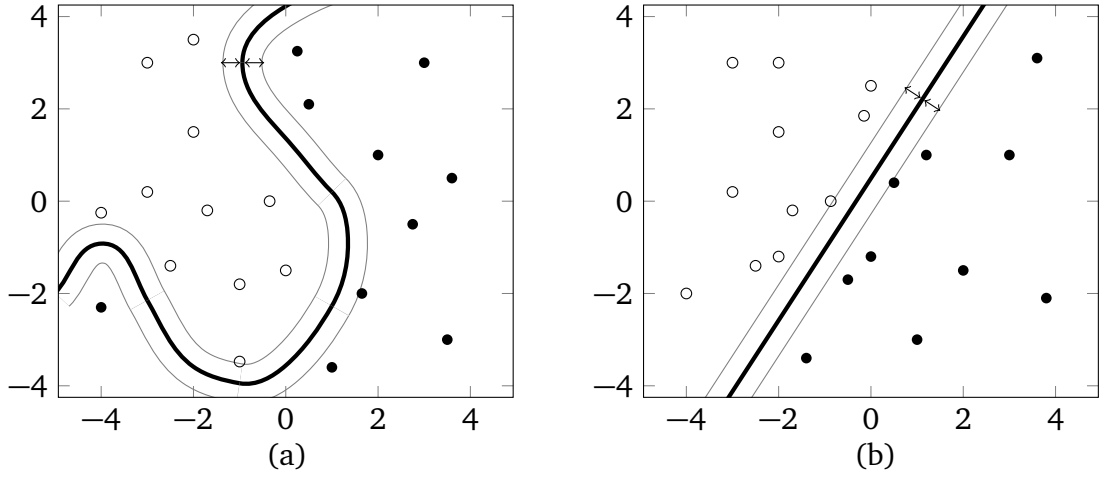


Figure 3.2. Visualization of the effect of the kernel trick on a classification problem between two classes of white and black dots. *Left*: to correctly separate the training data in two classes, a highly non-linear separation hyperplane is needed. *Right*: after the nonlinear mapping learned by the SVM, the optimal separation between the two classes can be achieved by a simple linear hyperplane.

black dots, respectively. The separating hyperplane is represented with a bold black line, while the thin gray lines represent its separation margin. It is evident that the SVM separating hyperplane does not only separate the two classes of data correctly, but, among all the separating hyperplanes, it is the one with the higher separation margin.

In most cases, however, data are not linearly separable, thus making a linear function not suitable for separating them successfully. Figure 3.2 (a) shows one of such cases in the context of a binary classification problem. In this example, the correct separation between the two classes can only be achieved by a highly nonlinear function.

To overcome this issue, Boser and colleagues [BGV92] introduced the *kernel trick*. The kernel trick consists in mapping the original finite dimensional data (Figure 3.2 (a)) in a much higher dimensional space by means of a nonlinear mapping  $\xi: \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$ ,  $d' \gg d$ , where they will be easier to separate through a linear hyperplane (Figure 3.2 (b)).

This can be accomplished by noticing that Equation (3.1) can be rewritten as

$$b + \sum_{i \in I} \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle, \quad (3.3)$$

where  $\mathbf{x}_i$  are the training samples and  $\boldsymbol{\alpha} = (\alpha_i)_{i \in I}$  is a vector of coefficients,



and by replacing the inner product  $\langle \mathbf{x}, \mathbf{x}_i \rangle$  with a more generic kernel  $k(\mathbf{x}, \mathbf{x}_i) = \langle \xi(\mathbf{x}), \xi(\mathbf{x}_i) \rangle$ , where  $\xi$  is the given nonlinear mapping.

As a result, exploiting the kernel trick, we can define the nonlinear SVM model as

$$U_{\Theta}(\mathbf{x}) = \text{sign} \left( b + \sum_{i \in I} \alpha_i k(\mathbf{x}, \mathbf{x}_i) \right), \quad (3.4)$$

where  $\Theta = \{\boldsymbol{\alpha}, b\}$ .

The nonlinear SVM model is equivalent to the application of the model presented in Equation (3.2) after a preprocessing step where the input data are mapped to a higher dimensional space through  $\xi$ . In this interpretation, the application of the nonlinear function  $\xi$  can be thought to as a feature extraction step, where  $\xi(\mathbf{x})$  represents the  $d'$ -dimensional feature associated with the  $d$ -dimensional sample  $\mathbf{x}$ . Figure 3.2 (b) shows the result of applying the nonlinear mapping  $\xi$  to the data shown on the left: in the new, higher-dimensional space the data admits a linear separation.

It is important to note that, even if  $U_{\Theta}(\mathbf{x})$  is nonlinear with respect to the original samples  $\mathbf{x}$ , it is linear with respect to the optimization parameters  $\boldsymbol{\alpha}, b$  and features  $\xi(\mathbf{x})$ , which are fixed and do not take part in the optimization. This constitutes a big advantage of SVM because it allows to exploit convex optimization techniques during learning. As a result, SVMs are guaranteed to converge to the global optimum.

SVMs, however, suffer from two major drawbacks. The first one resides in the fact that the nonlinear mapping  $\xi$  is fixed and is not optimized during learning. As a result, the quality of the SVM model (3.4) depends on the quality of the nonlinear mapping itself, which should be tailored for the specific application taken into consideration. The second drawback is the computation cost of the model (3.4), which is linear in the number of training samples since each training sample  $\mathbf{x}_i$  contributes to the cost function with the term  $\alpha_i k(\mathbf{x}, \mathbf{x}_i)$ . As a result, SVMs does not scale well with the size of the training set.

Deep learning approaches have been developed to overcome such issues. For instance, contrary to SVMs, deep learning techniques, such as neural networks, do not rely on a preprocessing step extracting features  $\xi(\mathbf{x})$  from the input data  $\mathbf{x}$ , but they allow to learn such features directly from the data.

### 3.3 Challenges motivating deep learning

In Section 3.1.3 we saw how the effectiveness of a machine learning algorithm depends on its generalization capacity. In order to achieve good generalization, a

machine learning algorithm should be able to exploit not only the information contained in the training data, but also the prior knowledge available on the current problem.

In practice, such prior knowledge is expressed in terms of properties of the output model or *priors*. A prior can be either explicit, e.g. a constraint or a regularization term in the optimization problem, or implicit, when the algorithm is biased towards some class of functions. In both cases, imposing some priors on a machine learning algorithm corresponds to restricting the class of functions from which the algorithm is allowed to learn the output model by preferring those sharing the desired properties. Implicit priors are more interesting because they allow to understand the specificity of the given algorithm, while extrinsic ones can be imposed to any algorithm.

The most common intrinsic prior is *local constancy*. The idea behind local constancy is that if the model  $U_{\theta}$  provides a good prediction for the point  $\mathbf{x}$ , we can expect such prediction to be good also in the neighbourhood of  $\mathbf{x}$ . More formally, a model  $U_{\theta}$  is said locally constant if  $U_{\theta}(\mathbf{x}) \approx U_{\theta}(\mathbf{x} + \epsilon)$  for any input  $\mathbf{x}$  and small variation  $\epsilon$ .

Most traditional machine learning algorithms rely solely on this property to achieve a good generalization. Unfortunately, this becomes a problem when the number of dimensions in the data increases.

The reason behind the insufficiency of the local consistency prior to deal with high-dimensional data depends on a phenomenon known as *curse of dimensionality*. For a comprehensive review of subject we refer the reader to Bellman's book [Bel61]. In the following, instead, we will provide a brief explanation.

Let us consider a  $d$ -dimensional input sample  $\mathbf{x} = (x_1, \dots, x_d)$ , and let us say that each of its entries  $x_i$ ,  $i = 1, \dots, d$ , may assume  $v$  possible values. Then we have  $\mathcal{O}(v^d)$  possible configurations ([GBC16]). In other words, the number of possible configurations increases exponentially when the data dimension increases. In order to achieve good generalization, we need to have at least a training sample for each possible configuration. In practice, however, training sets have fixed size and when the data dimension increases, training samples become increasingly sparse in the configuration space. As a result, if we rely only on the local consistency prior we do not have enough information to generalize well on the configuration space left uncovered by the training samples.

Interestingly, deep learning approaches do not rely only on the local constancy prior but introduce additional priors, which allow them to perform well even on tasks involving high dimensional data. In particular, the common assumption behind deep learning approaches, and in particular artificial neural networks, is the definition of the models as a composition of simpler factors, acting as a

prior. More details about deep learning methods and priors will be provided in the following.

## 3.4 Deep Learning

Most traditional machine learning algorithms, such as the SVMs presented in Section 3.2, cannot deal directly with raw data but require a preprocessing step where such raw data are mapped into proper data representations, called *features*, from which a model is built.

This is a major drawback because building a hand-crafted model extracting the most suitable feature representation for the given data or application requires advanced domain expertise and careful design.

To overcome this drawback, a category of machine learning algorithms known as *representation learning* allows to learn features directly from raw data, without requiring any hand-crafted feature extraction stage.

*Deep learning* is a particular class of representation learning, where the learnable features are modelled as a composition of many simple transformations, called *layers*. Each of these layers act as a different feature representation, therefore the output features of a deep learning algorithm exhibit multiple levels of abstraction [LBH15].

The number of layers needed to approximate the target model is known as the *depth* of the model. In order to approximate very complex models, deep learning algorithms need to compose together many different layers. This is the reason behind the term “deep learning”.

The key aspect of deep learning approaches is that the features extracted by each layer are learned from data through a general-purpose learning procedure rather than being computed by axiomatic models as in traditional machine learning algorithms.

### 3.4.1 Artificial neural networks

*Artificial neural networks* are one of the most prominent deep learning algorithms. An artificial neural network (ANN) is a parametric function  $U_{\theta}$  mapping the input data  $\mathbf{f}(x)$  to the output features  $\mathbf{g}(x)$ , i.e.  $\mathbf{g}(x) = U_{\theta}(\mathbf{f}(x))$ .

The term *network* stems from the fact that the model  $U_{\theta}$  is defined as a composition of different layers, representing differentiable functions. In the technical jargon, the input data  $\mathbf{f}(x)$  is called *input layer*, the output feature map



Figure 3.3. Graphs connecting the layers of a toy ANN with only one hidden layer. *Left*: in a feedforward neural network the connections between layers does not form loops. As a result, the information flows from the input to the output. *Right*: a recurrent neural network admits feedback connections. In this example, the output of the hidden layer is fed back into the network again.

$g(x)$  is called *output layer*, and the intermediate transformations  $H^{(i)}$  between the input and output layers are called *hidden layers*.

The model  $U_{\Theta}$  is typically associated with a graph describing how such layers are connected together. If the connections between the layers forms a directed acyclic graph, the ANN constitutes a *feedforward neural network*. If, instead, the layers are connected by a directed cyclic graph we obtain a *recurrent neural network*, of which LSTM [HS97] is the most prominent example.

Figure 3.3 shows the difference between feedforward and recurrent networks in terms of the possible connections between the layers in the case of a toy ANN composed of only one hidden layer.

In the next section, we will delve into feedforward models, while the topic of recurrent models goes beyond the purpose of this thesis.

### 3.4.2 Feedforward neural networks

A feedforward neural network (FNN) is an artificial neural network where the information is allowed to flow only in one direction: from the input  $f(x)$  to the output  $g(x)$ , passing through the intermediate transformations  $H^{(i)}$ ,  $i = 1, \dots, k$ .

More specifically, a FNN can be defined as the model

$$\begin{aligned} g(x) &= U_{\Theta}(f(x)) \\ &= (H_{\Theta^{(k)}}^{(k)} \circ \dots \circ H_{\Theta^{(2)}}^{(2)} \circ H_{\Theta^{(1)}}^{(1)})(f(x)), \end{aligned}$$

where the  $i$ th hidden layer  $H_{\Theta^{(i)}}^{(i)}$  is a parametric function with parameters  $\Theta^{(i)}$ .

The FNN parameters are defined as the collection of the parameters of the hidden layers, i.e.  $\Theta = \{\Theta^{(1)}, \dots, \Theta^{(k)}\}$ . The number  $k$  of hidden layers represents the depth of the model. If  $k$  is big enough the model is considered *deep*, otherwise it is considered *shallow*. Information such as how many hidden layers to consider and how to connect them, e.g. whether we want just to stack them or to connect them in more exotic ways, contributes to define the *architecture* of a FNN.

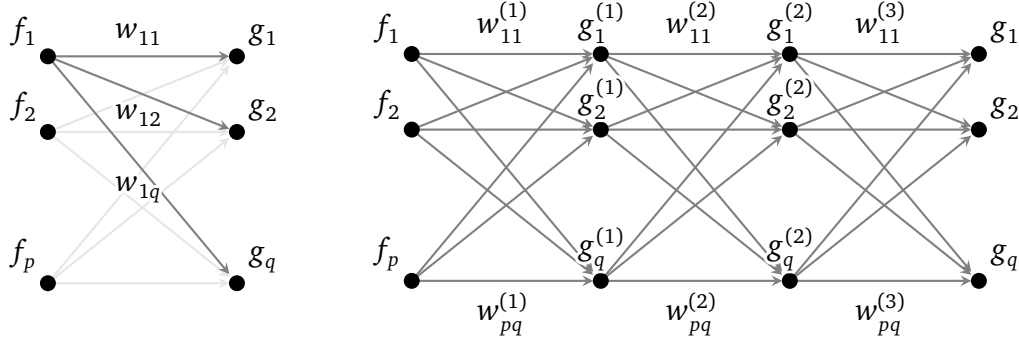


Figure 3.4. Visualization of fully connected layers. *Left*: a single fully connected layer where the weights affecting the first input unit are highlighted. *Right*: a FNN composed of three fully connected layers. The output of a fully connected layer is given in input to the following fully connected layer. As a result, the weights  $\mathbf{W}^{(2)}$ ,  $\mathbf{W}^{(3)}$  of the second and third layer, respectively, are not applied on the input data  $\mathbf{f}(x)$ , but rather on the intermediate representations  $\mathbf{g}^{(1)}$ ,  $\mathbf{g}^{(2)}$ .

Typically, hidden layers are defined as *fully connected layers*. A fully connected layer  $\mathbf{g}(x) = F_{\Theta}(\mathbf{f}(x))$  takes the form

$$\begin{aligned} \mathbf{g}(x) &= \xi(\mathbf{W}\mathbf{f}(x) + \mathbf{b}), \\ g_l(x) &= \xi\left(\sum_{l'=1}^p w_{l,l'} f_{l'}(x) + b_l\right), \end{aligned} \quad (3.5)$$

where the  $p$ -dimensional input data  $\mathbf{f}(x) = (f_1(x), \dots, f_p(x))$  is mapped to the  $q$ -dimensional output features  $\mathbf{g}(x) = (g_1(x), \dots, g_q(x))$  by means of the affine transformation  $\mathbf{W}\mathbf{f}(x) + \mathbf{b}$  followed by a pointwise non-linearity  $\xi$ , called *activation function*.

The affine transformation  $\mathbf{W}\mathbf{f}(x) + \mathbf{b}$  is composed of a linear term, represented by the matrix  $\mathbf{W} = (w_{l,l'})$ ,  $l = 1, \dots, q$ ,  $l' = 1, \dots, p$ , and a *bias* term  $\mathbf{b} = (b_l)$ ,  $l = 1, \dots, q$ . Overall, such affine transformation can be interpreted as the application of a bank of filters to the input function  $\mathbf{f}(x)$ .

Advantageously, such filters can be learned from data during training. More precisely, the learnable parameters of a fully connected layer are the weight matrix  $\mathbf{W}$  and the bias vector  $\mathbf{b}$ , which for simplicity are concatenated into the parameter set  $\Theta = \{\mathbf{W}, \mathbf{b}\}$ .

Figure 3.4 shows the typical visualization of fully connected layers showing how the weights  $w_{l,l'}$  contribute to the transformation of the input data. The entries of the input and output vectors  $\mathbf{f}(x)$ ,  $\mathbf{g}(x)$  are called *units* and are represented with black dots. The weights  $w_{l,l'}$  of the linear transformation described

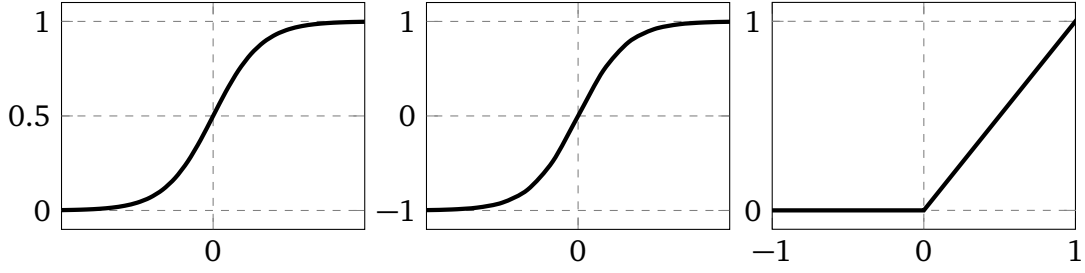


Figure 3.5. Most common activation functions. *Left*: the logistic sigmoid activation function is defined as  $\xi(x) = 1/(1 + e^{-x})$ . *Center*: the hyperbolic tangent activation function, i.e.  $\xi(x) = \tanh x$ . *Right*: the rectified linear unit (ReLU) activation function, defined as  $\xi(x) = \max(0, x)$ .

by the matrix  $\mathbf{W}$  are represented as arrows connecting the unit on which they are applied to the output unit they produce. The term “fully connected” arise from the fact that each input element is connected to each output element.

Activation functions are fixed non-linear transformations applied point-wise to the affine term  $\mathbf{W}\mathbf{f}(x) + \mathbf{b}$ . Among the most common activations functions we can find s-shaped functions such as the hyperbolic tangent (Figure 3.5, left) and the logistic sigmoid (Figure 3.5, center). Another possible choice is to consider piecewise linear functions. In the recent years, the most common choice as activations function is the *rectifying linear units* (ReLU) [NH10], i.e.  $\xi(x) = \max(0, x)$  (Figure 3.5, right).

In summary, a fully connected layer  $\mathbf{g}(x) = F_{\Theta}(\mathbf{f}(x))$  is a non-linear transformation of the input  $\mathbf{f}(x)$ . By composing together different fully connected layers, a FNN

$$\mathbf{g}(x) = (F_{\Theta^{(k)}} \circ \dots \circ F_{\Theta^{(2)}} \circ F_{\Theta^{(1)}})(\mathbf{f}(x))$$

can extract very complex features  $\mathbf{g}(x)$  from the input data  $\mathbf{f}(x)$ .

One of the most interesting properties of FNN is stated by the *universal approximation theorem* [HSW89; Cyb89]. The universal approximation theorem says that a FNN with a linear output layer and at least one hidden layer with an activation function similar to the three presented above, can approximate any continuous functions on compact subsets of  $\mathbb{R}^d$ .

In other words, a FNN with a single layer can represent a wide variety of interesting functions. However, the theorem does not state anything whether the parameters representing such functions can be learned during the training procedure or not. As noticed by [GBC16], the learning procedure may fail to find the correct parameters for two reasons:

- if the capacity of the model is not adequate, then it may suffer from overfitting or underfitting, thus approximating the wrong function;
- the nonlinearity of a FNN makes the optimization of the parameters a non-convex problem, therefore we do not have any guarantees of convergence to the desired parameters.

Another problem is related to the fact that the layer required to approximate the target function may be too large, thus impractical to implement.

Practitioners found that using deep models with smaller layers rather than shallow models with large layers help reducing the number of units required to approximate the desired function and show better generalization properties.

## 3.5 Training deep models

In Section 3.1.3 we saw how to cast the training procedure of a supervised learning algorithm as an optimization problem. More specifically, denoted by  $U_{\Theta}$  the model constructed by the supervised learning algorithm, and given a training set  $\{(f_i, y_i)\}_{i \in I}$ , the parameters  $\Theta$  can be learned by solving

$$\min_{\Theta} \sum_{i \in I} \ell(U_{\Theta}(f_i), y_i), \quad (3.6)$$

where the cost function  $\ell$  should be selected accordingly to the desired application.

Regression tasks usually consider a cost function of the form

$$\ell(U_{\Theta}(f_i), y_i) = \sum_{i \in I} \|U_{\Theta}(f_i) - y_i\|^2,$$

which promotes the feature  $U_{\Theta}(f_i)$  to be as similar as possible to the target  $y_i$ .

In case of classification, instead, the typical cost function is the *multinomial regression loss*

$$\ell(U_{\Theta}(f_i), y_i) = - \sum_{i \in I} \log U_{\Theta}(f_i),$$

where the output  $U_{\Theta}(f_i)$  can be interpreted as a probability distribution over the classes  $y_i$ . The term  $\log U_{\Theta}(f_i)$  represents the Kullback-Leibler divergence between the probability distribution produced by the model  $U_{\Theta}(f_i)$  and the ground-truth distribution  $\delta_{y_i}$ . Thus, minimizing the multinomial regression loss coincides with penalizing for the deviation of the probability distribution predicted by the model from the groundtruth.

Once the most appropriate cost function is selected, the solution of the optimization problem (3.6) can be found by gradient descent techniques.

Unfortunately, due to the non-convexity of the optimization problem associated with the training of a FNN, we cannot expect the gradient descent to converge to the global minimum. In most cases we cannot even expect the optimization to reach a good local minima. The possibility for the gradient descent algorithm to be trapped in a poor local minimum was one of the main critiques against FNNs (and deep learning in general) in the 1990s. Nowadays it is widely regarded that local minima are rarely a problem in deep models, since the quality of the corresponding model is very similar regardless of the initial condition.

The goal of this section is to provide all the necessary details about the gradient descent techniques that can solve the optimization procedure associated with the training of a FNN.

### 3.5.1 Back-propagation

The main ingredient in a gradient descent method is the computation of the gradient itself. The mathematical derivation of the gradient of a deep model composed of several nonlinear layers may be quite difficult. Fortunately, it can be easily computed with a method called *backpropagation* [Lin70; Lin76].

In a FNN, the input data  $\mathbf{f}$  is passed through different hidden layers to obtain the output features  $\mathbf{g}$ . This is called *forward propagation*. During training, the features  $\mathbf{g}$  are then used to evaluate the cost function  $\ell$ .

The backpropagation algorithm allows information to flow in the opposite direction, i.e. from the cost function to the input data through the hidden layers, to compute the gradient. More precisely, the backpropagation algorithm computes the partial derivatives of the output units w.r.t. the input units of each intermediate layer and then uses the chain rule to assemble the gradient of the cost function.

Let us consider a two-layer FNN  $g = F_{\Theta(2)}(F_{\Theta(1)}(\mathbf{f}))$  where the first hidden layer is defined as  $\mathbf{h} = F_{\Theta(1)}(\mathbf{f})$  and the second one is  $g = F_{\Theta(2)}(\mathbf{h})$ . Then, according to the chain rule we have

$$\frac{\partial g}{\partial f_i} = \sum_j \frac{\partial g}{\partial h_j} \frac{\partial h_j}{\partial f_i},$$

where the term  $\partial h_j / \partial f_i$  computes the derivative of the output units of the first layer with respect to the input units, and the term  $\partial g / \partial h_j$  computes the derivative of the output units of the second layer w.r.t. its input units (the output units of the first layer).

To compute the gradient  $\partial g / \partial f_i$ , the backpropagation algorithm starts from



the derivative of the output layer  $g$  with respect to itself, i.e.  $\partial g / \partial g = 1$ , then stores the multiplication between it and the partial derivatives for each layer of the network until reaching the input units  $f_i$ , thus obtaining the desired gradient.

### 3.5.2 Stochastic gradient descent

Cost functions used by machine learning problems, such as the ones presented at the beginning of Section 3.5, often involve a summation of terms evaluated over the training samples, i.e.

$$\ell(\Theta) = \sum_{i \in I} L(f_i, y_i, \Theta^{(i)}).$$

This means that the traditional gradient descent algorithm requires the computation of the gradient on the entire training set  $I$  at each iteration,

$$\nabla_{\Theta} \ell(\Theta) = \sum_{i \in I} \nabla_{\Theta^{(i)}} L(f_i, y_i, \Theta^{(i)}).$$

Unfortunately, this is impractical for deep learning related applications, where the typical training set varies from millions to billions of samples. The *stochastic gradient descent* algorithm is preferred, instead.

The key insight behind stochastic gradient descent is that the gradient can be thought as an expectation over some distribution of training data. Therefore it can be estimated by using only a small portion of such training data, called *minibatch*.

At each step, the stochastic gradient descent algorithm extracts a minibatch  $\{f_j\}_{j \in J}$ ,  $J \subset I$ , by a uniform sampling of the training set and then computes an estimation  $\mathbf{g}$  of the gradient  $\nabla_{\Theta} \ell(\Theta)$  as

$$\mathbf{g} = \sum_{j \in J} L(f_j, y_j, \Theta^{(j)}).$$

Once the gradient is estimated, the parameters are updated by the classical gradient descent formula,

$$\Theta = \Theta - \epsilon \mathbf{g},$$

or variations thereof.

## 3.6 Convolutional neural networks

*Convolutional neural networks* [Fuk80; LBBH98] are a particular kind of feedforward neural networks specialized in processing Euclidean data, i.e. data that has

a regular grid structure. Goodfellow et al. [GBC16] defines such data as having a known, grid-like topology, while LeCun et al. [LBH15] speak of data in the form of multiple arrays.

Examples of Euclidean data include:

- speech or audio signals, which can be described as the sequence of samples taken at regular time intervals specified by a 1D regular grid and can be stored in a 1D array;
- images, represented as the intensity values sampled on a 2D regular grid of pixels, one for each colour channel;
- volumetric images, such as medical images, represented in terms of the volumetric occupancy of a 3D regular grid of voxels.

A convolutional neural network (CNN) is a FNN containing at least a *convolution layer* as hidden layer. A convolution layer replaces the matrix multiplication operation contained in the affine term of the hidden layer with the *convolution* operation.

More precisely, a convolutional layer  $\mathbf{g}(x) = C_{\Theta}(\mathbf{f}(x))$  acts on a  $p$ -dimensional input  $\mathbf{f}(x) = (f_1(x), \dots, f_p(x))$  by applying a bank of filters  $\mathbf{W} = (w_{l,l'}), l = 1, \dots, q, l' = 1, \dots, p$  and an activation function  $\xi$ ,

$$g_l(x) = \xi \left( \sum_{l'=1}^p (f_{l'} * w_{l,l'})(x) \right), \quad (3.7)$$

producing a  $q$ -dimensional output  $\mathbf{g}(x) = (g_1(x), \dots, g_q(x))$ .

For 1D Euclidean data, the convolution between the signal  $f(x)$  and the filter  $w(x)$  is defined as the function  $s(x)$  s.t.

$$s(x) = (f * w)(x) = \int f(x - x')w(x')dx'. \quad (3.8)$$

In the discrete case, the function  $s, f, w$  are represented by the arrays  $\mathbf{s} = (s_i), \mathbf{f} = (f_i), \mathbf{w} = (w_i)$ , respectively, and the Equation (3.8) becomes

$$s_i = \sum_{h=-\infty}^{\infty} f_{i-h}w_h.$$

The discrete convolution can be extended to higher dimensional data as well. For instance, for 2D Euclidean data, it can be defined as

$$s_{i,j} = \sum_{k=-\infty}^{\infty} \sum_{h=-\infty}^{\infty} f_{i-h,j-k}w_{h,k}. \quad (3.9)$$

$a$	$b$	$c$	$d$
$e$	$f$	$g$	$h$
$i$	$j$	$k$	$l$

 $\ast$ 

$\alpha$	$\beta$
$\gamma$	$\delta$

 $=$ 

$\alpha a + \beta b$ $+ \gamma e + \delta f$	$\alpha b + \beta c$ $+ \gamma f + \delta g$	$\alpha c + \beta d$ $+ \gamma g + \delta h$
$\alpha e + \beta f$ $+ \gamma i + \delta j$	$\alpha f + \beta g$ $+ \gamma g + \delta h$	$\alpha g + \beta h$ $+ \gamma h + \delta i$

$\mathbf{f}$

$\mathbf{W}$

$\mathbf{s}$

Figure 3.6. Discrete convolution between a  $3 \times 4$  image  $\mathbf{f}$  and a  $2 \times 2$  filter  $\mathbf{W}$ . The intensities of the pixels of the image  $\mathbf{f}$  are denoted with the variables  $a, b, \dots, l$ , while the filter  $\mathbf{W}$  is specified through the learnable parameters  $\{\alpha, \beta, \gamma, \delta\}$ . The output of the discrete convolution is a  $2 \times 3$  feature map  $\mathbf{s}$  obtained as a local re-weighting of the pixels in the image on the left with the learnable filter weights.

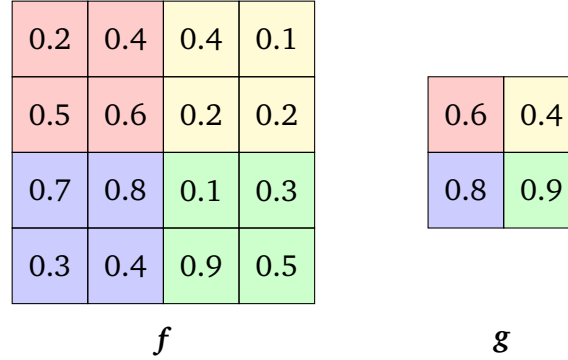


Figure 3.7. Max pooling on a  $2 \times 2$  neighbourhood of a  $4 \times 4$  image  $\mathbf{f}$  gives in output a  $2 \times 2$  feature map  $\mathbf{g}$ . Pixels of the same colour in the image  $\mathbf{f}$  are mapped into the pixel with the corresponding color in the feature map  $\mathbf{g}$ . Numbers in the pixels denote intensity values.

Figure 3.6 shows an application of Equation (3.9) to images.

As suggested by the formulas above, the convolution operation can be interpreted as a *cross-correlation* between the input data  $\mathbf{f}$  and the filter  $\mathbf{w}$ . As we will see later in Chapter 6, this is the main intuition leading to the spatial extension of the convolution layer to non-Euclidean data, which constitutes one of the main contributions of this thesis.

The other building block that characterizes CNNs is the *pooling layer*. A pooling layer  $\mathbf{g}(x) = P(\mathbf{f}(x))$  computes a subsampling of the input function  $\mathbf{f}(x)$  and can be defined as

$$g_l(x) = \|f_l(x') : x' \in \mathcal{N}(x)\|_{L^p}, \quad l = 1, \dots, q, \quad (3.10)$$

where  $\mathcal{N}(x)$  is a neighborhood around  $x$ . Depending on which  $L^p$ -norm is considered, different samplings are obtained: the case  $p = 1$  correspond to average-pooling, the case  $p = 2$  to mean-pooling and the case  $p = \infty$  to max-pooling. A pooling layer is a fixed layer, i.e. it does not contain learnable parameters. The only hyperparameter of a pooling layer is the size of the neighborhood  $\mathcal{N}(x)$ . The role of the pooling layer is to aggregate the information over a whole neighbourhood in a single feature summarizing their statistics. Figure 3.7 shows an application of max-pooling to images.

A CNN  $\mathbf{g}(x) = U_{\Theta}(\mathbf{f}(x))$  is constructed by composing together convolutional and (optionally) pooling layers, obtaining a generic hierarchical representation

$$\mathbf{g}(x) = (C_{\Theta^{(k)}} \circ P \circ \dots \circ C_{\Theta^{(2)}} \circ C_{\Theta^{(1)}})(\mathbf{f}(x)),$$

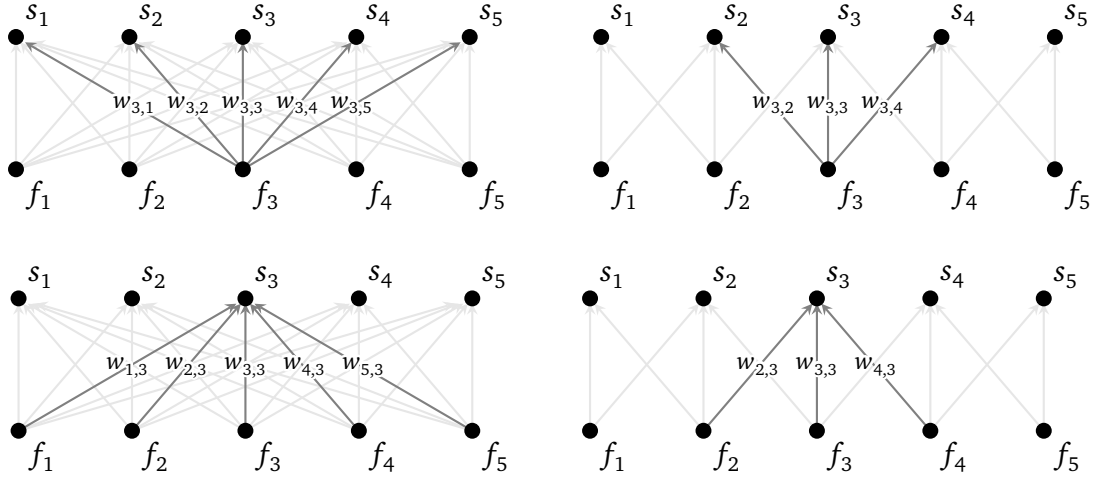
with parameters  $\Theta = \{\Theta^{(k)}, \dots, \Theta^{(1)}\}$ , where  $\Theta^{(i)}$  represents the parameters of the  $i$ th convolutional layer. By alternating between convolution and pooling layers, a CNN extracts hierarchical feature representations.

### 3.6.1 Properties of convolutional neural networks

In Section 3.2 we saw how local consistency is a typical prior shared by most traditional machine learning algorithms. Deep learning, and in particular FNNs (Section 3.4), imposes an additional prior by constructing models through composition of factors. CNNs do not only share both these priors but, advantageously, impose three additional ones: *local filters*, *parameter sharing*, and *invariant/equivariant representation*. By leveraging these priors, CNNs avoid the curse of dimensionality phenomenon presented in Section 3.3. In the following, we will go through each of these three properties.

**Local filters** As we saw in Section 3.4.2, fully connected layers are defined by the matrix multiplication  $\mathbf{W}\mathbf{f}$  where the matrix  $\mathbf{W}$  is full, which implies that every input unit is connected with every output units (Figure 3.8, left). Convolutional layers, instead, are characterized by the convolution between an input function  $\mathbf{f}$  and a filter bank  $\mathbf{W}$ . Usually the filters  $\mathbf{W}$  have a local support significantly smaller than the one of the input data. For instance, when dealing with images, the filter occupies a small window of pixels, e.g. a  $5 \times 5$  window. Figure 3.6 shows the results of the convolution between an image of size  $3 \times 4$  with a filter of size  $2 \times 2$ .

Local filters result in sparse connections between input and output units, resulting in a drastic reduction of the parameters the model has to learn: a fully connected layer with dense connections between  $p$  input units and  $q$  output units



*Figure 3.8.* Connections from input to output units in a fully connected layer (on the left) and a convolutional layer (on the right). In this toy example, we consider a 5-dimensional input  $\mathbf{f} = (f_1, \dots, f_5)$  and a convolution filter of width 3. Connections are represented by arrows. Each connection corresponds to a learnable weight  $w_{i,j}$ , where  $i, j$  are the index of the input and output units, respectively. *Top:* visualization about how input units affect the output units. In particular, we highlight the output units  $s_i$  affected by the input unit  $f_3$ . In the fully connected layer on the left  $f_3$  influences all the output units  $s_1, \dots, s_5$ , while in the convolution layer on the right  $f_3$  influences only a small portion of output units, i.e.  $s_2, s_3, s_4$ . *Bottom:* visualization about how output units are affected by the input units. In particular, we highlight the input units  $f_i$  that contribute to the output unit  $s_3$ . In the fully connected layer on the left  $s_3$  is influenced by all the input units  $f_1, \dots, f_5$ , while in the convolution layer on the right  $s_3$  is influenced only by a small portion of output units, i.e.  $f_2, f_3, f_4$ . Figure reproduced from [GBC16].

has  $p \times q$  parameters; a convolutional layer, instead, connects each output unit with a fixed number  $k \ll p$  of input units, resulting in  $k \times q$  parameters.

Figure 3.8 compares dense connections of a fully connected layer with sparse connections in a convolutional layer in terms on how the connectivity influences input and output units respectively.

In a single convolution layer, an output unit is affected only by a limited number of input units because of the locality of the filters. If, instead, we increase the depth of the model by composing together multiple convolution layers, the units of deeper layers are influenced by a larger portion of the input units, as shown in Figure 3.9.

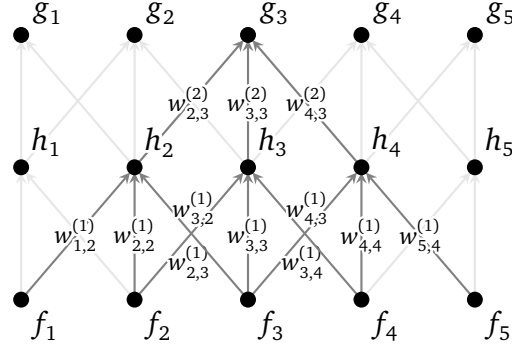


Figure 3.9. A two-layer convolutional neural network. The first convolution layer maps the input  $\mathbf{f} = (f_1, \dots, f_5)$  to an intermediate feature map  $\mathbf{h} = (h_1, \dots, h_5)$ . The second one maps the vector  $\mathbf{h} = (h_1, \dots, h_5)$  to the output feature map  $\mathbf{g} = (g_1, \dots, g_5)$ . A unit in a shallow layer is affected only by few input units, e.g.  $h_3$  is affected only by  $f_2, f_3, f_4$ . A unit in a deep layer, instead, is affected by more input units, e.g.  $g_3$  is affected by  $f_1, \dots, f_5$ . Figure reproduced from [GBC16].

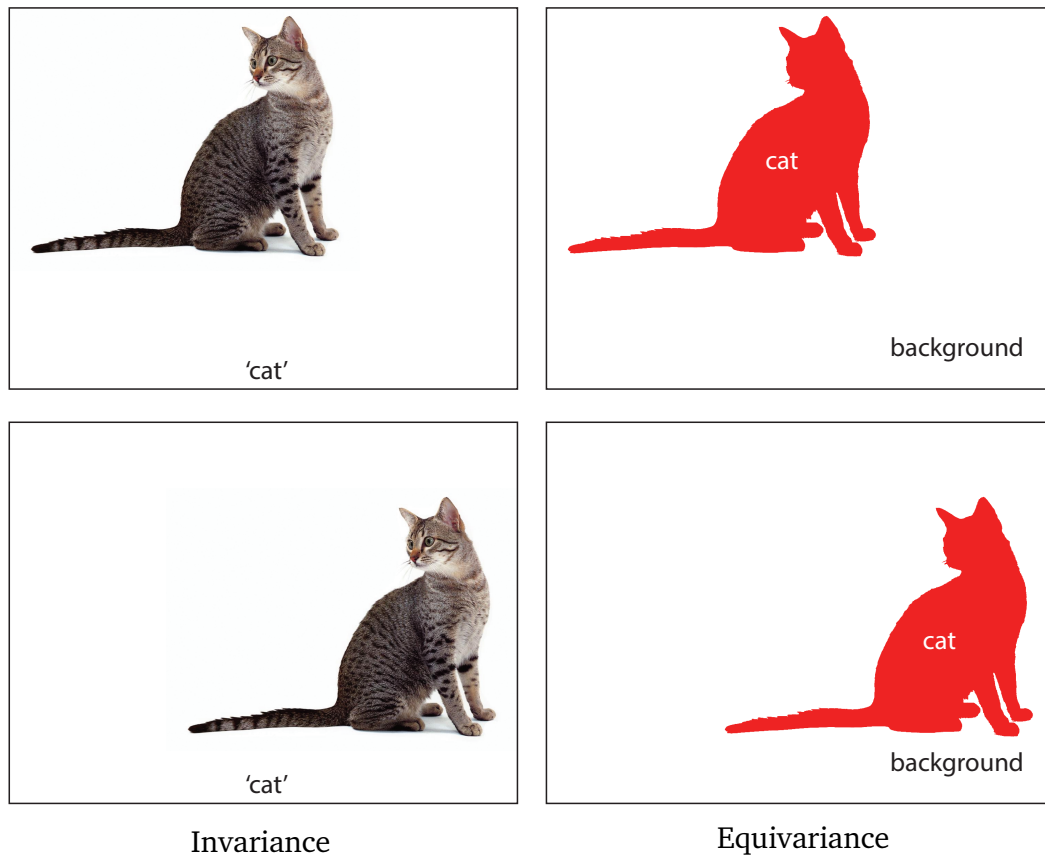
**Parameter sharing** A fully connected layer learns a parameter  $w_{i,j}$  for each input unit  $f_i$ ,  $i = 1, \dots, p$ , and output unit  $g_j$ ,  $j = 1, \dots, q$ . To the contrary, the parameters of the filters of a convolutional layers are shared, which means that the parameters of the convolutional filter applied to a certain input unit are assumed to be the same for each other input unit of the same layer.

For example, if we consider as input data an image, this means that a fully connected layer learns a different set of parameters for each pixel, while a convolutional layer learns a single set of parameters on the whole image.

This is particularly interesting when dealing with image analysis applications, where the goal is usually the understanding of the content of the given image. In an image, local features are likely to be repeated in different locations. For instance if a specific edge or corner appears in a location, we expect to find it in other locations of the same image as well. Therefore, if a local filter detects such edge or corner, we are interested in applying the same filter for each other pixel in the image, rather than learning a different filter for each location.

Thanks to parameter sharing, the number of parameters in a convolutional layer further reduces from  $k \times q$  to only  $k$ : a significant improvement w.r.t. the  $p \times q$  parameters of a traditional fully connected layer.

**Invariant/equivariant representation** Let us consider the *translation operator*  $\mathcal{T}_v f(x) = f(x - v)$ ,  $x, v \in \mathbb{R}^d$ , acting on function  $f \in L^2(\mathbb{R}^d)$ . A function  $y$  is *invariant* to  $\mathcal{T}_v$  if  $y(\mathcal{T}_v f) = y(f)$ , it is *equivariant* to  $\mathcal{T}_v$  if  $y(\mathcal{T}_v f) = \mathcal{T}_v y(f)$ .



*Figure 3.10. Left: object classification is a typical task that benefits from translation invariance. In this example, two images containing the same cat are provided, but the position of the cat differs. Despite the position of the cat in the image we would like our model to predict in both cases the correct class. Right: semantic segmentation, instead, requires translation equivariance. If we want to correctly segment the images on the left, the foreground prediction should change accordingly to the translation of the cat. Translation invariance in this case will produce a completely wrong segmentation.*

Translation invariance is required in applications where the presence of certain features is more important than their precise location, e.g. in object classification tasks, where one does not care where a certain object is localized in an image, but just if it is present or not (Figure 3.10, left).

Translation equivariance, instead, is a much stronger property because it requires the output features to ‘adapt’ to the translation of the input data. A typical task that requires translation equivariance is object localization, where

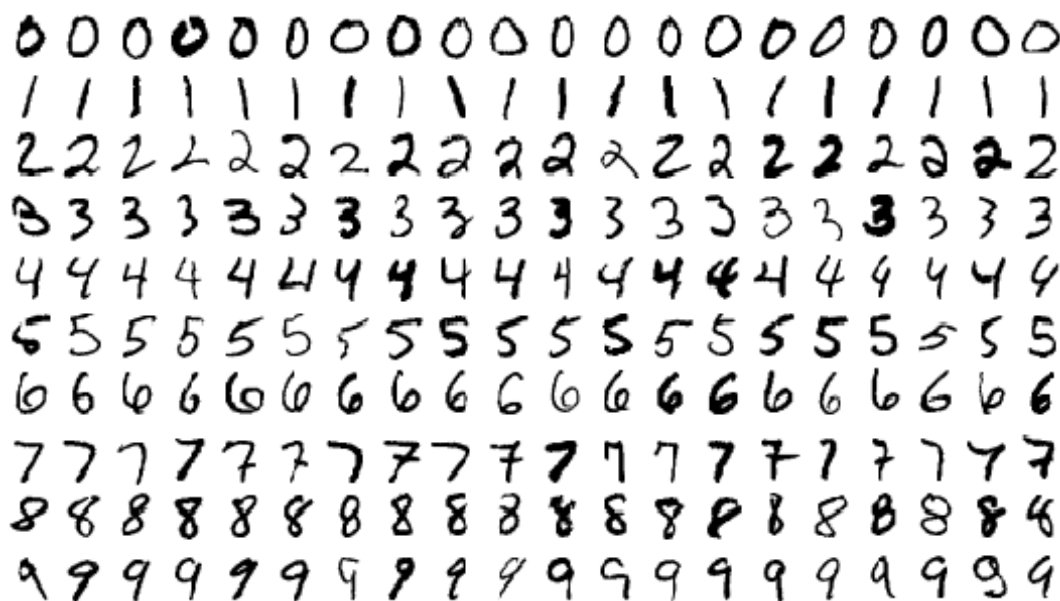


Figure 3.11. Handwritten digits from the famous MNIST dataset. Each row corresponds to different samples of the same digit. Each sample is written by a different person, with different handwriting style.

one wants to predict the position of a certain object in an image. Another typical application is semantic segmentation, where one tries to partition the image into semantically meaningful parts and classify them (Figure 3.10, right).

The convolution operation is equivariant to translation, while the pooling operation makes the model invariant to small translations. As a result, CNNs can leverage both translation invariance and equivariance, depending on how convolutional and pooling layers are combined together.

This is an extremely important property because it means that the features learned by a CNN are robust to small translations of the input: e.g. the features learned on an image  $\Omega(i, j)$  are consistent also on the image  $\Omega'(i, j) = \Omega(i - 1, j)$ , where every row is shifted by one pixel.

Advantageously, Mallat [Mal12] proved that, for specific choices of the convolutional operator and activation function, CNNs can achieve also deformation invariance. A deformation can be formally defined as the operator  $\mathcal{L}_\tau$  acting on functions  $f \in L^2(\mathbb{R}^d)$  s.t.  $\mathcal{L}_\tau f(x) = f(x - \tau(x))$ , where  $\tau: \mathbb{R}^d \rightarrow \mathbb{R}^d$  is a smooth vector field. Deformations can model local translations, rotations, and viewpoint change. Figure 3.11 shows a practical example of deformations in the case of handwritten digits from the famous MNIST dataset.



## Chapter 4

# Frequency domain intrinsic deep learning methods

In this chapter, we present two methods allowing to extend the convolution operation to non-Euclidean data in the form of graphs by resorting to the spectral definition of the convolution operation.

The first method was proposed by Bruna and colleagues [BZSL13; HBL15] and, to the best of our knowledge, was the first attempt to generalize CNNs to graphs by replacing the traditional convolutional layer with a spectral convolutional layer, implementing the spectral definition of the convolution operation. The second method, proposed by Defferrard et al. [DBV16], provides an efficient spectrum-free extension of the former approach.

These approaches have severe limitations in the context of shape analysis applications, because they do not allow to transfer the parameters learned on a domain to a different one. However, they are quite relevant for this thesis because they gave us the inspiration and motivation for developing of our contributions in this field.

### 4.1 Spectral methods

The convolution theorem states that the convolution between two functions  $f, h \in L^2(\mathbb{R})$  can be expressed in the spectral domain as the product of their Fourier transforms, in the sense that

$$(f * h)(x) = \mathcal{F}^{-1}(\mathcal{F}(f(x)) \cdot \mathcal{F}(h(x))), \quad (4.1)$$

where  $\mathcal{F}$  denotes the classical Fourier transform, and  $\mathcal{F}^{-1}$  its inverse.

Frequency-domain methods exploit this fundamental property of the convolution operation to extend it to non-Euclidean domains.

In particular, by replacing the classical Fourier transform in Equation (4.1) with the generalized one introduced in Section 2.4.2, frequency-domain methods allow to define the convolution between two functions  $f, h \in L^2(\mathcal{X})$  as

$$(f * h)(x) = \sum_{i \geq 1} \langle f, \phi_i \rangle_{L^2(\mathcal{X})} \langle h, \phi_i \rangle_{L^2(\mathcal{X})} \phi_i(x), \quad (4.2)$$

where  $\phi_i$  is the  $i$ th eigenfunction of the Laplacian on  $\mathcal{X}$ .

The terms  $\langle f, \phi_i \rangle_{L^2(\mathcal{X})}, \langle h, \phi_i \rangle_{L^2(\mathcal{X})}$  act as a forward Fourier transform on the functions  $f, h \in L^2(\mathcal{X})$  by computing their Fourier coefficients  $a_i, b_i$ , respectively. The multiplication between such coefficients creates new coefficients  $c_i = a_i b_i$  corresponding to the spectral representation of the convolution function, which is then mapped back to the spatial domain by the inverse Fourier transform  $\sum_{i \geq 1} c_i \phi_i(x)$ .

For practical applications, the series in Equation (4.2) is truncated and only the first  $k$  terms are considered, i.e.

$$(f * h)(x) \approx \sum_{i=1}^k \langle f, \phi_i \rangle_{L^2(\mathcal{X})} \langle h, \phi_i \rangle_{L^2(\mathcal{X})} \phi_i(x). \quad (4.3)$$

One of the key differences of such a construction from the classical convolution operation is the lack of shift-invariance. In terms of signal processing, Equation (4.3) can be interpreted as a position-dependent filter. While parametrized by a fixed number of coefficients in the frequency domain, the spatial representation of the filter can vary dramatically at different points.

### 4.1.1 Spectral convolutional neural networks

*Spectral convolutional neural networks* (SCNN) [BZSL13] exploit the generalized definition of convolution presented in Equation (4.3) to extend the convolutional neural network framework to graph-structured data.

In order to extend CNNs to non-Euclidean data, one has to redefine their building blocks, i.e. the convolution and pooling layers, in order to make them compatible with manifold data.

In particular, Bruna et al. [BZSL13; HBL15] proposed to replace the traditional convolutional layer of a classical Euclidean CNN (Section 3.6), with a *spectral convolutional layer* implementing the spectral convolution of Equation (4.3), and to replace the traditional pooling layer with a graph coarsening operation.

Since the graph coarsening can be computed with an off-the-shelf algorithm, it is worth to focus the attention on the spectral convolutional layer, instead.

The spectral convolutional layer is defined as

$$\mathbf{g}_l = \xi \left( \sum_{l'=1}^p \mathbf{\Phi}_k \mathbf{W}_{l,l'} \mathbf{\Phi}_k^\top \mathbf{f}_{l'} \right), \quad (4.4)$$

where the  $n \times p$  and  $n \times q$  matrices  $\mathbf{F} = (\mathbf{f}_1, \dots, \mathbf{f}_p)$  and  $\mathbf{G} = (\mathbf{g}_1, \dots, \mathbf{g}_q)$  represent the  $p$ - and  $q$ -dimensional input and output signals sampled on the vertices of the graph, the  $n \times k$  matrix  $\mathbf{\Phi}_k$  represents the Fourier eigenbasis of  $L^2(\mathcal{X})$  containing only the first  $k$  Laplacian eigenvectors,  $\mathbf{W}_{l,l'}$  is a  $k \times k$  diagonal matrix of spectral multipliers representing a filter in the frequency domain, and  $\xi$  is a nonlinear activation function applied vertex-wise.

More specifically, the  $k$ -dimensional vector  $\mathbf{\Phi}_k^\top \mathbf{f}_{l'}$  contains the Fourier coefficients  $a_i = \langle \mathbf{f}_{l'}, \phi_i \rangle_{L^2(\mathcal{X})}$ ,  $i = 1, \dots, k$ , resulting from the forward Fourier transform applied to the input signal  $\mathbf{f}_{l'}$ . The convolutional filter (the function  $h$  in Equation (4.3)) is represented in the spectral domain through its Fourier coefficients  $b_i = \langle h, \phi_i \rangle_{L^2(\mathcal{X})}$ ,  $i = 1, \dots, k$ , which are stored in the diagonal of the  $k \times k$  matrix  $\mathbf{W}_{l,l'}$ . Given the coefficients  $a_i, b_i$ , the Fourier coefficients  $c_i = a_i b_i$ ,  $i = 1, \dots, k$ , of the convolution function are computed by the term  $\mathbf{W}_{l,l'} \mathbf{\Phi}_k^\top \mathbf{f}_{l'}$ . Finally, the multiplication between such term and the Fourier eigenbasis  $\mathbf{\Phi}_k$  produces an  $n$ -dimensional vector representing the convolution function in the spatial domain.

The operations described above are repeated for each dimension  $l' = 1, \dots, p$  of the input signal. The contribution of each input dimension is averaged and passed through a point-wise non-linear activation function  $\xi$  to obtain the output  $\mathbf{g}_l$ .

Usually, a bank of  $q$  filters is considered, obtaining an  $n$ -dimensional output vector  $\mathbf{g}_l$  for each filter  $\mathbf{W}_{l,l'}$ ,  $l = 1, \dots, q$ .

In summary, a spectral convolutional layer has  $pqk$  learnable parameters: for each input dimension  $l = 1, \dots, p$ , and each filter  $l' = 1, \dots, q$ , we have  $k$  parameters (the Fourier coefficients of the filter). Assuming that  $k = \mathcal{O}(n)$  Laplacian eigenvectors are kept, a spectral convolutional layer (4.4) requires  $pqk = \mathcal{O}(n)$  parameters to train.

In order to reduce the risk of overfitting, it is important to adapt the learning complexity to reduce the number of free parameters of the model. On Euclidean domains, this is achieved by learning convolutional kernels with small spatial support, which enables the model to learn a number of parameters independent of the input size (Section 3.6).

The SCNN filters, however, are not defined in the spatial domain but only their spectral representation is provided. Interestingly, Bruna and colleagues [BZSL13;

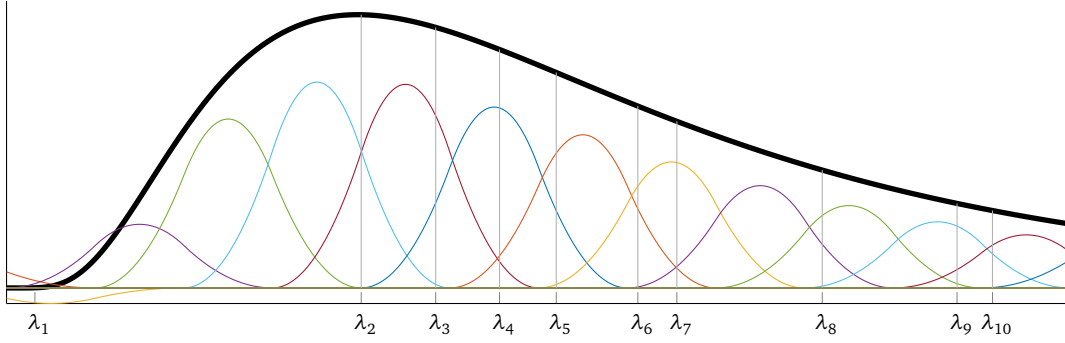


Figure 4.1. A smooth spectral filter  $\tau(\lambda)$  (black thick line) can be defined as a linear combination of the smooth kernel functions  $\beta_1(\lambda), \dots, \beta_k(\lambda)$  (colored thin lines).

HBL15] showed that the locality prior can be enforced in the SCNN framework too by restricting the class of spectral multipliers to those corresponding to localized filters. This way, the number of parameters of the convolutional layer (4.4) becomes independent upon the size of the input, which is the case of classical Euclidean CNNs.

The *Parseval's identity* provides a key intuition about how to constrain the parameter space. In particular, in the Euclidean setting, it states that

$$\int_{-\infty}^{\infty} |x|^{2k} |f(x)|^2 dx = \int_{-\infty}^{\infty} \left| \frac{\partial^k \hat{f}(\omega)}{\partial \omega^k} \right|^2 d\omega,$$

which suggests that localization in space is equivalent to smoothness in the frequency domain.

Following this intuition, Bruna et al. [BZSL13; HBL15] proposed to parametrize SCNN filters using a smooth spectral transfer function  $\tau(\lambda)$ , i.e.

$$\mathbf{W}(\lambda) = \begin{bmatrix} \tau(\lambda_1) & & \\ & \ddots & \\ & & \tau(\lambda_k) \end{bmatrix},$$

where the Laplacian eigenvalues  $\{\lambda_1, \dots, \lambda_k\}$  play the role of frequencies.

In particular  $\tau(\lambda)$  can be defined as a linear combination of smooth kernel functions  $\beta_1(\lambda), \dots, \beta_q(\lambda)$ ,

$$\tau(\lambda_i) = \sum_{j=1}^q \alpha_j \beta_j(\lambda_i) = (\mathbf{B}\boldsymbol{\alpha})_i, \quad (4.5)$$

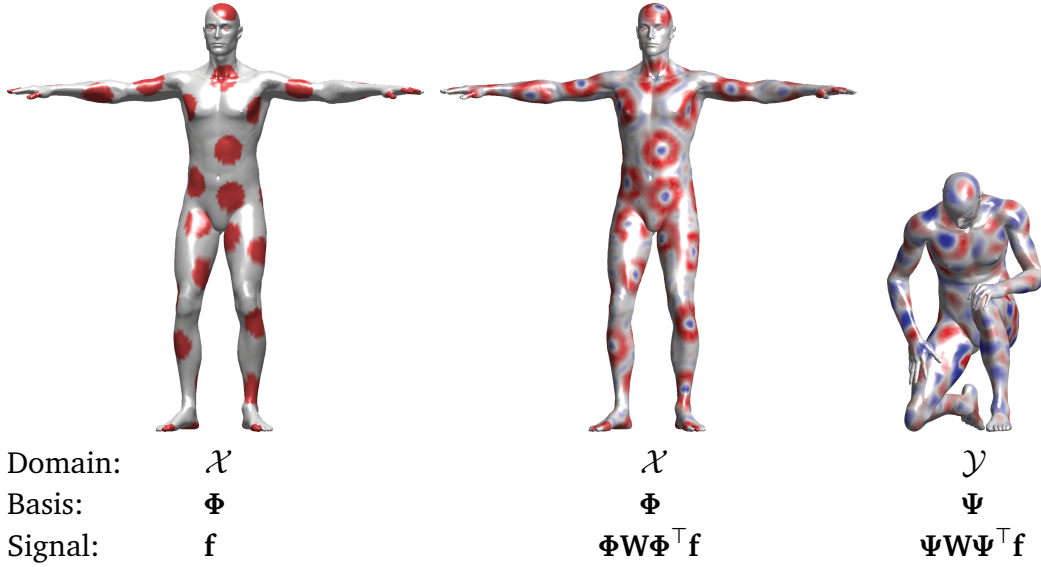


Figure 4.2. An illustration of the poor generalization of spectral filtering across non-Euclidean domains. *Left*: a function defined on a manifold. *Middle*: result of the application of a filter in the frequency domain on the same manifold. *Right*: the same filter applied on the same function but on a different (nearly-isometric) domain produces a completely different result.

where  $\mathbf{B} = (b_{i,j}) = (\beta_j(\lambda_i))$  is a  $k \times q$  fixed interpolation kernel containing the functions  $\beta_i$  as columns and  $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_q)^\top$  is the vector of  $q$  interpolation coefficients, representing the filter parameters. A typical choice for the kernel functions  $(\beta_j)$ ,  $j = 1, \dots, q$ , consists in B-splines.

As a result, the weight matrix of the spectral convolutional layer of a SCNN with smooth spectral multipliers becomes

$$\text{diag}(\mathbf{W}_{l,l'}) = \mathbf{B} \boldsymbol{\alpha}_{l,l'}, \quad (4.6)$$

where  $l, l'$  are the indexes of the input and output dimensions, respectively.

Unfortunately, SCNNs suffer from two major drawbacks:

- the computation of the forward and inverse Fourier transforms requires the multiplications with matrices  $\Phi^\top$  and  $\Phi$ , respectively, which have  $\mathcal{O}(n^2)$  complexity;
- the filters are basis-dependent, which means that they do not generalize across domains. In other words, if we learn a filter w.r.t. a basis on one domain, and then try to apply it on another domain with another basis, the result could be very different (see Figure 4.2).

## 4.2 Spectrum-free methods

Spectrum-free methods deal with the first drawback of SCNNs [BZSL13; HBL15], allowing to define the convolution operation on non-Euclidean data without explicitly requiring the cumbersome Laplacian eigendecomposition.

From the orthogonality of the Laplacian eigenfunctions, it follows that a polynomial of the Laplacian operator acts as a polynomial on its eigenvalues, i.e.

$$\tau_\alpha(\Delta) = \Phi \tau_\alpha(\Lambda) \Phi^\top. \quad (4.7)$$

The key idea behind spectrum-free methods is to leverage relation (4.7) to replace the spectral construction of the filters in Equation (4.5), with the polynomial expansion

$$\tau_\alpha(\Delta) = \sum_{j=1}^r \alpha_j \Delta^j. \quad (4.8)$$

Since the Laplacian is a local operator (acting on 1-ring neighborhoods), the action of its  $j$ th power is constrained to  $j$ -ring neighborhoods. As a consequence, the linear combination of powers of the Laplacian (4.8) is still limited to  $r$ -ring neighborhoods around each vertex. It follows that this representation automatically yields localized filters.

### 4.2.1 Chebyshev convolutional neural networks

*Chebyshev convolutional neural networks* (ChebNet) [DBV16] are a particular kind of spectrum-free methods where the polynomial  $\tau_\alpha$  is defined by the recursive family of Chebyshev polynomials

$$\begin{aligned} T_0(\lambda) &= 1, \\ T_1(\lambda) &= \lambda, \\ T_j(\lambda) &= 2\lambda T_{j-1}(\lambda) - T_{j-2}(\lambda). \end{aligned} \quad (4.9)$$

More specifically, Defferard et al. [DBV16] proposed to consider spectral filters of the form

$$g_\alpha(\Delta) = \sum_{j=1}^{r-1} \alpha_j T_j(\tilde{\Delta}) = \sum_{j=1}^{r-1} \alpha_j \Phi T_j(\tilde{\Lambda}) \Phi^\top,$$

where  $\tilde{\Delta} = 2\lambda_n^{-1}\Delta - \mathbf{I}$  is the rescaled Laplacian such that its eigenvalues  $\tilde{\Lambda} = 2\lambda_n^{-1}\Lambda - \mathbf{I}$  are in the interval  $[-1, 1]$  rather than  $[0, \lambda_n]$ . This transformation is necessary because the Chebyshev polynomials form an orthogonal basis on  $[-1, 1]$ .

The application of the spectral filter to a signal  $\mathbf{f}$  is given by

$$g_\alpha(\Delta) = \sum_{j=1}^{r-1} \alpha_j T_j(\tilde{\Delta}) \mathbf{f}.$$

If we denote with  $\hat{\mathbf{f}} = T_j(\tilde{\Delta}) \mathbf{f}$ , from the recurrence relation (4.9) it follows that

$$\begin{aligned} \hat{\mathbf{f}}^{(0)} &= \mathbf{f}, \\ \hat{\mathbf{f}}^{(1)} &= \tilde{\Delta} \mathbf{f}, \\ \hat{\mathbf{f}}^{(j)} &= 2\tilde{\Delta} \hat{\mathbf{f}}^{(j-1)} - \hat{\mathbf{f}}^{(j-2)}, \end{aligned}$$

which means that ChebNet spectral filters require  $r$  multiplications between  $n \times n$  sparse matrices, resulting in a total complexity of  $\mathcal{O}(nr)$ .

In summary, ChebNet [DBV16] has two important advantages over SCNN [BZSL13; HBL15]. First, it does not require an explicit computation of the Laplacian eigenvectors. Second, it has computational complexity  $\mathcal{O}(nr)$  instead of  $\mathcal{O}(n^2)$ .

Unfortunately, ChebNet does not solve the second drawback of SCNN: filters learned with ChebNet still do not generalize well across very different domains. This is a major issue for shape analysis tasks we aim to solve, where one has to compare filters on different shapes.

In the following, we will see how our contributions allowed to define methods able to generalize well across domains by introducing alternative extensions of the convolution operation resorting to a frequency transformation different from the Fourier one (Chapter 5) and to charting-based methods (Chapter 6).





# Chapter 5

## Hybrid frequency-spatial intrinsic deep learning methods

In the previous chapter, we saw how frequency methods don't generalize well across different domains. A possible way to overcome this difficulty is by replacing the generalized Fourier transform with a different transformation, as proposed in our paper [BMM<sup>+</sup>15]. The material presented below is mainly based on such contribution.

### 5.1 Windowed Fourier transform

One of the notable drawbacks of classical Fourier analysis is its lack of spatial localization: by virtue of the *uncertainty principle*, a signal that is localized in the spatial domain has poor frequency localization, and viceversa. Figure 5.1 shows such phenomenon in the case of 1D signals.

In classical signal processing, this problem is overcome by localizing frequency analysis in a window  $g$ , leading to the definition of the *windowed Fourier transform* (WFT, also known as short-time Fourier transform in signal processing).

Given a function  $f \in L^2(\mathbb{R})$  and some window  $g \in L^2(\mathbb{R})$  localized at zero, the classical WFT is defined as

$$(Sf)(x, \omega) = \int_{-\infty}^{\infty} f(x')g(x - x')e^{-i\omega x'} dx'.$$

The WFT is a function of two variables: the spatial location of the window  $x$  and the modulation frequency  $\omega$ . The choice of the window function  $g$  allows to control the tradeoff between spatial and frequency localization (wider windows result in better frequency resolution).

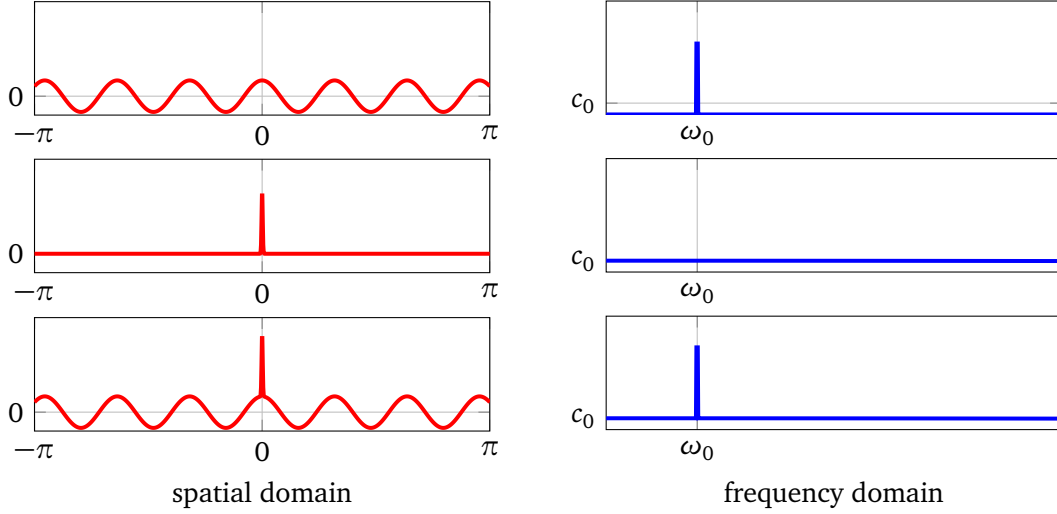


Figure 5.1. Visualization of the uncertainty principle on 1D signals. Red curves represent functions defined on the spatial domain, blue curves represent their Fourier transforms. Despite the fact that the high-frequency content of the functions in the second and third row is localized (around  $x = 0$ ), their Fourier transforms do not allow to recover such information.

Figure 5.2 shows the application of the WFT to the signals depicted in Figure 5.1 for different choices of the window  $g$ . Contrary to the Fourier transforms reported in Figure 5.1, a WFT with a sufficiently narrow window (third row) allows to localize the high frequency content of the signal.

Interestingly, the WFT can be interpreted as an inner product with a translated and modulated window,

$$(Sf)(x, \omega) = \langle f, M_\omega T_x g \rangle_{L^2(\mathbb{R})}, \quad (5.1)$$

where  $T_x$  and  $M_\omega$  denotes the translation and modulation operator, respectively. The translated and modulated window  $g_{x,\omega} = M_\omega T_x g$  is referred to as the WFT *atom*.

In the Euclidean setting, the translation operator is defined simply as  $(T_{x'}g)(x) = g(x - x')$ , while the modulation operator amounts to a multiplication by a Laplacian eigenfunction  $(M_\omega g)(x) = e^{i\omega x} g(x)$ . The action of modulation amounts to translation in the frequency domain, i.e.  $\widehat{M_\omega g} = \widehat{g}(\omega - \omega')$ .

Unfortunately, in the non-Euclidean setting, the translation to a point  $x'$  ( $x \mapsto x - x'$ ) is not even well defined.

In their seminal work, Shuman et al. [SRV16] showed how to generalize such notions to graphs, while in the paper [BMM<sup>+</sup>15] we showed how to extend them

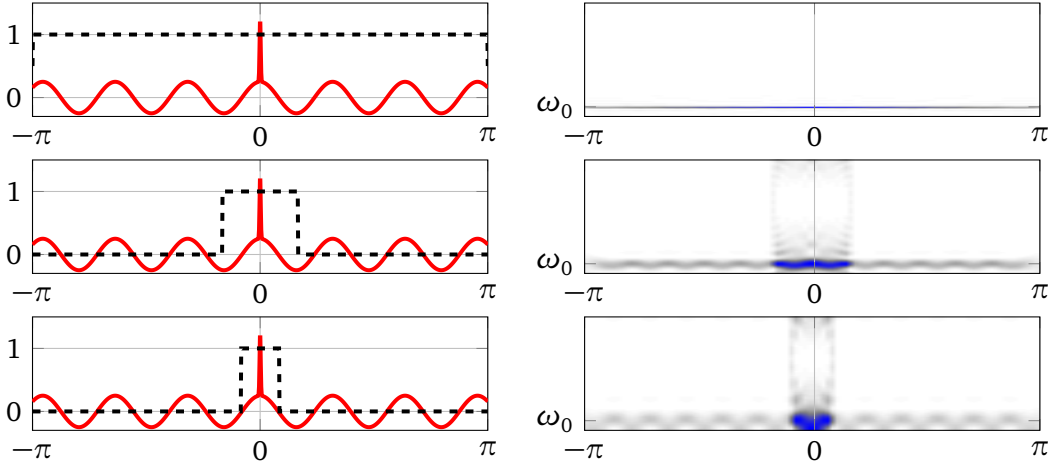


Figure 5.2. Visualization of the windowed Fourier transform of 1D signals  $f(x)$  for different choices of the window  $g(x)$ . The signals are represented by solid red curves, the windows are represented by dashed black curves and the WFT is represented on the right in the white-blue colormap. If the window is too wide (first row) the WFT boils down to the classical Fourier transform. If, instead, the window is more localized (second and third row), the WFT allows to extract more information w.r.t. the traditional Fourier transform depicted in Figure 5.1.

to 3D shapes.

Translation to a point  $x' \in \mathcal{X}$  can be replaced by convolution with a delta-function  $\delta_{x'}$  centered at  $x'$ , yielding

$$\begin{aligned}
 (T_{x'}g)(x) &= (g * \delta_{x'})(x) \\
 &= \sum_{i \geq 1} \langle g, \phi_i \rangle_{L^2(X)} \langle \delta_{x'}, \phi_i \rangle_{L^2(X)} \phi_i(x) \\
 &= \sum_{i \geq 1} \langle g, \phi_i \rangle_{L^2(X)} \phi_i(x') \phi_i(x).
 \end{aligned}$$

Note that, in general, such a translation is not shift-invariant: the window  $T_x g$  change when  $x \in \mathcal{X}$  changes (see Figure 5.3).

The modulation operator, instead, can be defined as  $(M_j f)(x) = \phi_j(x) f(x)$ , where  $\phi_j$  is the  $j$ th eigenfunction of the Laplacian.

Combining the two operators together, the WFT atom on non-Euclidean manifolds becomes

$$\begin{aligned}
 g_{x',j}(x) &= (M_j T_{x'} g)(x) \\
 &= \phi_j(x) \sum_{i \geq 1} \langle g, \phi_i \rangle_{L^2(X)} \phi_i(x) \phi_i(x'),
 \end{aligned} \tag{5.2}$$

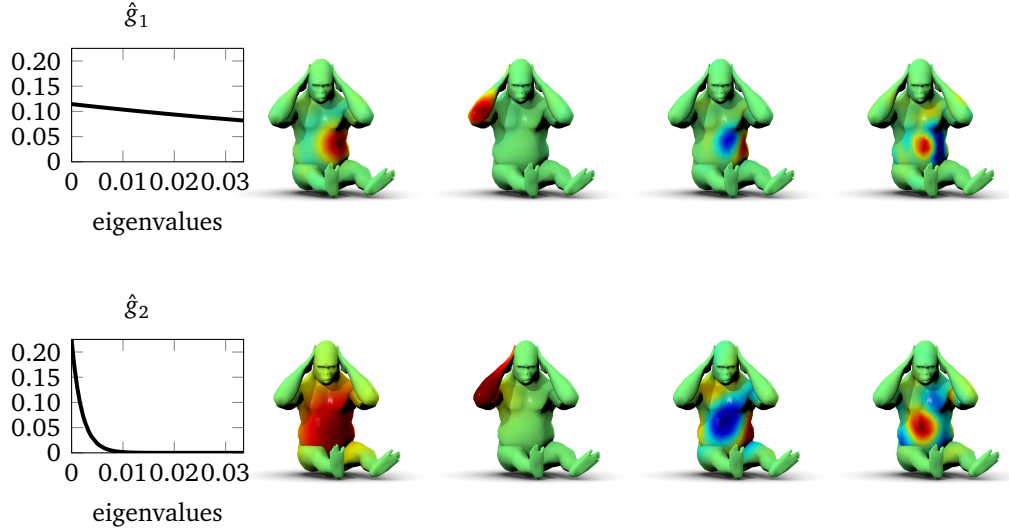


Figure 5.3. Examples of different WFT atoms  $g_{x,k}$  using different windows (top and bottom rows; window Fourier coefficients are shown on the left), shown in different localizations (second and third columns) and modulations (fourth and fifth columns).

where the window is defined in the frequency domain by its Fourier coefficients  $\langle g, \phi_i \rangle_{L^2(\mathcal{X})}$ . Figure 5.3 shows a visualization of different examples of such atoms on a gorilla shape.

Finally, by analogy with the Euclidean case (5.1), Shuman et al. [SRV16] showed that the WFT of a signal  $f \in L^2(\mathcal{X})$  can be defined as

$$\begin{aligned} (Sf)(x', j) &= \langle f, g_{x',j} \rangle_{L^2(\mathcal{X})} \\ &= \sum_{i \geq 1} \langle g, \phi_i \rangle_{L^2(\mathcal{X})} \phi_i(x') \langle f, \phi_i \phi_j \rangle_{L^2(\mathcal{X})}. \end{aligned} \quad (5.3)$$

## 5.2 Localized spectral convolutional neural networks

In our paper [BMM<sup>+</sup>15], we exploited the WFT (5.3) as an alternative to the Fourier transformation considered by the SCNN framework proposed by Bruna et al. [BZSL13; HBL15] in order to extend the convolution operation to non-Euclidean data.

The key intuition behind the LSCNN framework [BMM<sup>+</sup>15] is the following one. Given a signal  $f$ , the WFT  $(Sf)(x, j)$  performs a local filtering around the point  $x$  by extracting the response of the function  $f$  at the frequency  $j$ .

By collecting the WFTs  $(Sf)(x, j)$  for different frequencies  $j = 1, \dots, k$ , it is possible to extract a local representation of the input signal  $f$  around the point  $x$ , acting similarly to the pixels' window extracted by traditional Euclidean CNNs on images. The only difference between the pixels' window extracted from images and the WFT-based representation is that the first one is a spatial representation extracting the signal of each pixel in a neighbourhood of  $x$ , while the second one is a frequency representation of a signal that is localized by means of multiplication by a window  $g$ .

Once the local representation of the function  $f$  around the point  $x$  is extracted, the convolution operation can be defined as cross-correlation between such representation of the signal and a filter expressed in the same representation.

*Localized spectral convolutional neural networks* (LSCNN), presented in our paper [BMM<sup>+</sup>15], are an extension of traditional Euclidean CNNs where the convolutional layer  $\mathbf{g}(x) = C_{\Theta}(\mathbf{f}(x))$  is defined as

$$\mathbf{g}_l(x) = \sum_{l'=1}^p \sum_{j=1}^k w_{l,j,l'} |(S\mathbf{f}_{l'})(x, j)|, \quad (5.4)$$

where the  $n \times p$  and  $n \times q$  matrices  $\mathbf{f} = (\mathbf{f}_1, \dots, \mathbf{f}_p)$  and  $\mathbf{g} = (\mathbf{g}_1, \dots, \mathbf{g}_q)$  represent the  $p$ - and  $q$ -dimensional input and output signals on the vertices of the shape, and  $\mathbf{W} = (w_{l,j,l'})$  is a  $p \times k \times q$  tensor representing the learnable weights.

The Laplacian eigenfunctions are defined up to sign, i.e.  $\Delta(\pm\phi) = \lambda(\pm\phi)$ . Thus, even isometric domains might have different Laplacian eigenfunctions. In order to obtain convolutional filters which exhibit good generalization across different domains, the convolutional layer (5.4) applies an absolute value to the WFT in order to reduce its dependence on the eigenfunctions sign.

The WFT  $(S\mathbf{f}_{l'})(x, j)$  depends on the choice of the window  $g$ . As we saw in Equation (5.2), typically the window  $g$  is specified in the frequency domain by providing its Fourier coefficients  $\hat{g}_i = \langle g, \phi_i \rangle_{L^2(\mathcal{X})}$ ,  $i = 1, \dots, k$ . If we consider fixed windows, then the learnable parameters of the convolutional layer (5.4) are represented by the convolutional filters  $\Theta = \{\mathbf{W}\}$ .

Advantageously, LSCNN [BMM<sup>+</sup>15] allows to consider also learnable windows defined in the spectral domain as

$$\gamma_{l'}(\lambda) = \sum_{l=1}^m a_{l,l'} \beta_l(\lambda), \quad (5.5)$$

where  $\beta_1, \dots, \beta_m$  represents a fixed smooth basis in the frequency domain, such as the B-spline basis, and the  $q \times m$  matrix  $\mathbf{A} = (a_{l,l'})$  contains the learnable weights

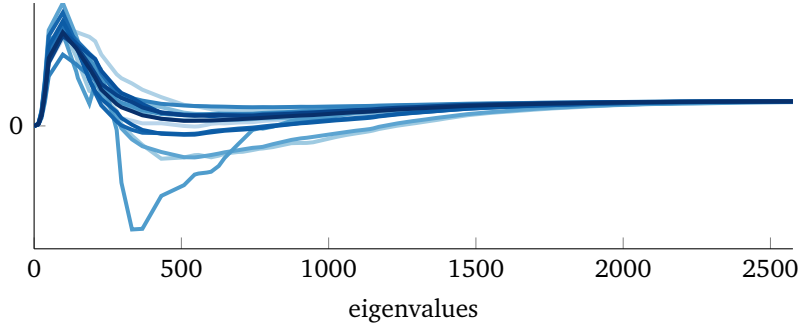


Figure 5.4. Example of a family of windows  $\hat{g}_1, \dots, \hat{g}_p$  learned by the LSCNN framework on a standard shape dataset. The numbers on the x axis denote the eigenvalue indices.

that define the windows. Figure 5.4 depicts the behaviour of the windows (5.5) once the training procedure has been completed.

By replacing the fixed window of Equation (5.3) with the learnable one in Equation (5.5), the WFT becomes

$$(Sf_{l'})(x', j) = \sum_{i \geq 1} \gamma_{l'}(\lambda_i) \phi_i(x') \langle f, \phi_i \phi_j \rangle_{L^2(X)}, \quad (5.6)$$

where a different window  $\gamma_{l'}$  is learned for each input dimension  $l' = 1, \dots, p$ .

The parameters of the LSCNN convolutional layer (5.4) with a learnable window (5.5) are  $\Theta = \{\mathbf{W}, \mathbf{A}\}$ .

It is important to notice that the WFT definition (Equation (5.3)) depends only on the Laplacian eigenfunctions.

On the one hand, this implies that the same framework can be used for any shape representation (e.g. mesh and point clouds): the specific representation of the shape influences only the Laplacian discretization.

On the other hand, however, the features extracted by the LSCNN convolutional layer (5.4) depend on the Laplacian eigenfunctions. This implies that such features are robust across shapes only if their Laplacian eigenbases, up to known ambiguities, do not differ arbitrarily. In practice, this implies that LSCNNs are effective in tasks involving shapes that share a common geometric structure (e.g. different males, animals from a common species).

We hypothesize that if one tries to deal with a class of shapes that is too broad (e.g. all mechanical objects, or all living things), the advantage of LSCNN [BMM<sup>+</sup>15] over hand-crafted descriptors such as HKS [SOG09; GBAL09] and WKS [ASC11] will diminish, and it is likely that we will learn these descriptors.

From the computational standpoint, a notable disadvantage of the WFT-based LSCNN construction is the need to explicitly produce each window, which results in high memory and computational requirements.





## Chapter 6

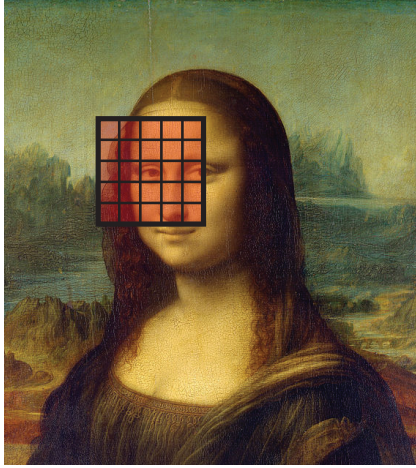
# Spatial intrinsic deep learning methods

Frequency methods (Chapter 4) provide a generalization of CNNs to non-Euclidean domains by performing the convolution operation in the spectral domain. The main drawback of these approaches is their limitation to a single domain, which is too restrictive for shape analysis tasks, where typical problems involve more than one shape, such as in the case of learning correspondences between two shapes.

Hybrid methods (Chapter 5), such as LSCNN [BMM<sup>+</sup>15], are an extension of the spectral method developed by Bruna et al. [BZSL13; HBL15] using the windowed Fourier transform (WFT) [SRV16] on manifolds. Due to the localization properties of the WFT, this method shows better generalization abilities than SCNN, however, it might have problems in the case of strongly non-isometric deformations due to the variability of the Laplacian eigenfunctions. The drawback of LSCNN approach [BMM<sup>+</sup>15] is high memory and computational requirements, since the WFT-based construction requires the explicit computation of each window.

This motivates the need to resort to an alternative generalization of the traditional convolutional layer in the spatial domain. This chapter shows our contributions to the field of spatial methods by presenting the material of our papers [MBBV15; BMR<sup>+</sup>16; BMRB16; MBM<sup>+</sup>17].

On a Euclidean domain, due to shift-invariance, the convolution can be thought of as passing a template at each point of the domain and recording the correlation of the template with the function at that point. If we consider images as our prototypical Euclidean data, this operation amounts to extracting a (typically square) patch of pixels, multiplying it element-wise with a template and summing



Euclidean



Non-Euclidean

Figure 6.1. Comparison between the patch extraction on an image and on a shape (non-Euclidean domain). *Left*: The patch on an image is represented by a square ( $5 \times 5$ ) grid of pixels. Due to the shift-invariance of Euclidean data, the operation of extracting such patch is always the same at any position on the image. *Right*: On non-Euclidean manifolds, the lack of global coordinate system implies that the patch should be represented in local coordinates, while the lack of shift-invariance implies that the patch extraction is position-dependent.

up the results, then moving to the next position in a sliding window manner (Figure 6.1, left). Shift-invariance implies that the very operation of extracting the patch at each position is always the same.

One of the major problems in applying the same paradigm to non-Euclidean domains is the lack of shift-invariance, implying that the *patch operator* extracting a local *patch* would be position-dependent. Furthermore, the lack of a meaningful global parametrization on manifolds forces to represent the patch in some local intrinsic system of coordinates (Figure 6.1, right).

Our contributions [MBBV15; BMR<sup>+</sup>16; BMRB16; MBM<sup>+</sup>17] share a common construction of the patch operator, which we briefly overview. The patch operator can be represented by defining a set of weighting functions  $v_1(x, \cdot), \dots, v_k(x, \cdot)$  localized to positions near  $x$ . Extracting a patch amounts to averaging the function  $f$  at each point by these weights,

$$D_j(x)f = \int_{\mathcal{X}} f(x')v_j(x, x')dx', \quad j = 1, \dots, k, \quad (6.1)$$

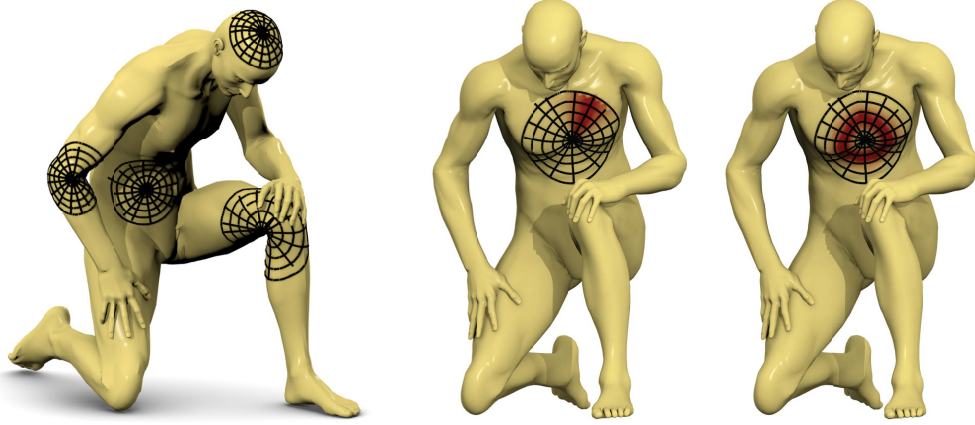


Figure 6.2. Construction of local geodesic polar coordinates on a manifold. *Left*: examples of local geodesic patches. *Middle and right*: example of angular and radial weighting functions, respectively (red denotes larger weights).

providing for a spatial definition of an intrinsic equivalent of convolution

$$(f * h)(x) = \sum_j h_j D_j(x) f, \quad (6.2)$$

where  $h_j$  denotes the filter coefficients applied on the patch extracted at each point. Overall, Equations (6.1) and (6.2) act as a kind of non-linear filtering of  $f$ , and the patch operator  $D$  is specified by defining the weighting functions  $v_1, \dots, v_k$ .

The big advantage of the spatial formulation of the intrinsic convolution (6.2) over the spectral one presented in Chapters 4 and 5 is the fact that the spatial formulation allows to learn localized filters by construction.

In the following, we will present the methods published in our papers [MBBV15; BMR<sup>+</sup>16; BMRB16; MBM<sup>+</sup>17]. Such methods essentially differ in the choice of the weighting functions  $v_1, \dots, v_k$  of the patch operator (6.1).

## 6.1 Geodesic convolutional neural networks

*Geodesic convolutional neural network* (GCNN) [MBBV15] was the first work trying to tackle the problem of extending CNNs to 3D shapes by extending the convolution operation in the spatial domain with the introduction of a patch operator (6.1).

**Intrinsic polar system of coordinates** In particular, in the paper [MBBV15], we propose to define the patch operator as a combination of Gaussian weights defined on a *local intrinsic polar system of coordinates*.

Given a point  $x$  on a shape  $\mathcal{X}$ , the local intrinsic polar system of coordinates specifies the coordinates of the surrounding points in terms of radial and angular components  $(\rho(x), \theta(x))$ .

The radial coordinate is constructed as  $\rho$ -level sets  $\{x' : d_{\mathcal{X}}(x, x') = \rho\}$  of the geodesic distance function  $d_{\mathcal{X}}$  for  $\rho \in [0, \rho_{\max}]$ , where  $\rho_{\max}$  is the radius of the geodesic disc. If the radius  $\rho_{\max}$  of the geodesic ball  $B_{\rho_{\max}}(x) = \{x' \in \mathcal{X} : d_{\mathcal{X}}(x, x') \leq \rho_{\max}\}$  is sufficiently small w.r.t. the local convexity radius of the manifold, then the resulting ball is guaranteed to be a topological disc. Empirically, we see that choosing a sufficiently small  $\rho_{\max} \approx 1\%$  of the geodesic diameter of the shape produces valid topological discs.

The angular coordinate  $\theta(x)$  is constructed as a set of equispaced geodesics  $\Gamma(x, \theta)$  emanating from  $x$  in direction  $\theta$  in a way that they are perpendicular to the geodesic distance level sets. Note that the choice of the origin of the angular coordinate is arbitrary. We denote by  $\Delta\theta$  the angle interval between the geodesics.

Figure 6.2 (left) shows the local intrinsic polar system of coordinates constructed following the previous procedure for different vertices of a shape.

**Geodesic patch operator and convolution** Once the intrinsic polar system of geodesic coordinates is extracted, the *geodesic patch operator* is defined as

$$(D(x)f)(\theta, \rho) = \int_{\mathcal{X}} f(x') v_{\rho, \theta}(x, x') dx' \quad (6.3)$$

where the weighting functions

$$v_{\rho, \theta}(x, x') = \frac{v_{\rho}(x, x') v_{\theta}(x, x')}{\int_{\mathcal{X}} v_{\rho}(x, x') v_{\theta}(x, x') dx'}$$

are obtained as the (normalized) product between the angular weighting functions

$$v_{\theta}(x, x') = e^{-d_{\mathcal{X}}^2(\Gamma(x, \theta), x') / 2\sigma_{\theta}^2},$$

and the radial weighting functions

$$v_{\rho}(x, x') = e^{-(d_{\mathcal{X}}(x, x') - \rho)^2 / 2\sigma_{\rho}^2}.$$

Figure 6.2 (center and right) shows an example of angular and radial weights  $v_{\theta}, v_{\rho}$ , respectively.

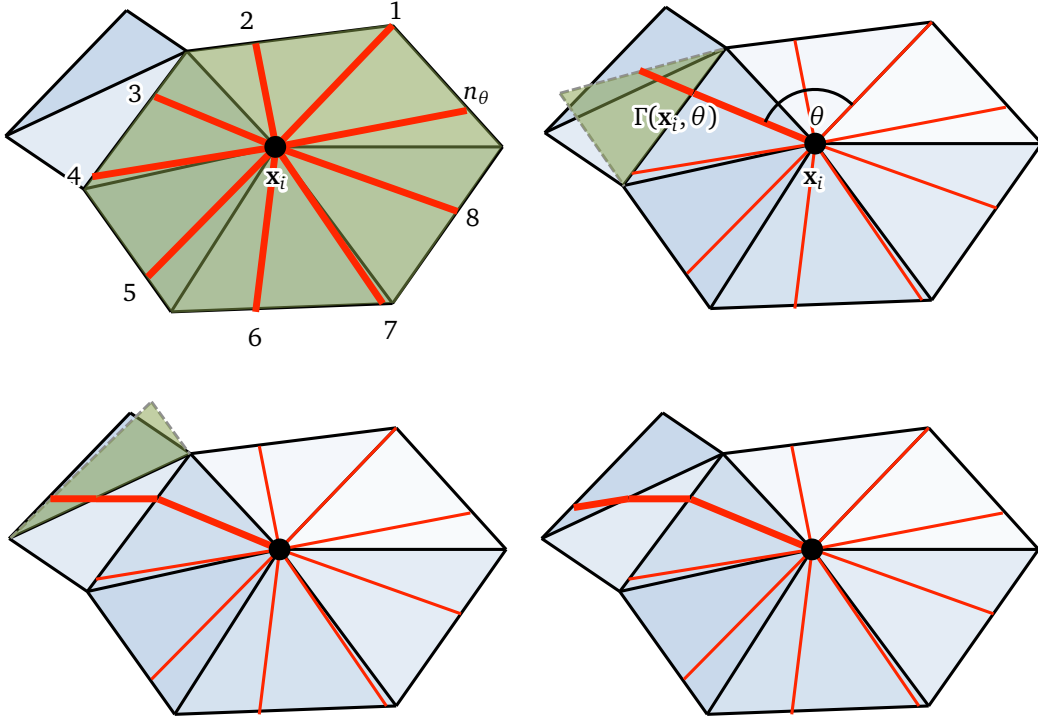


Figure 6.3. Construction of local geodesic polar coordinates on a triangular mesh. Shown clock-wise: division of 1-ring of vertex  $i$  into  $n_\theta$  equi-angular bins; propagation of a ray (bold line) by unfolding the respective triangles (marked in green).

Once the patch operator (6.3) is computed, we can regard  $D(x)f$  as a patch extracting the content of the signal  $f$  around the point  $x \in \mathcal{X}$ . On such local representation, the *geodesic convolution* is defined as

$$(f * w)(x) = \int_0^{2\pi} \int_0^{\rho_{\max}} w(\theta, \rho) (D(x)f)(\theta, \rho) d\rho d\theta, \quad (6.4)$$

where  $w(\theta, \rho)$  is a learnable filter applied on the patch.

In GCNN [MBBV15], we exploit such definition of geodesic convolution to define the *geodesic convolution layer*

$$\mathbf{g}_i(x) = \sum_{l'=1}^p (\mathbf{f}_{l'} * w_{l,l'})(x),$$

where the  $n \times p$  and  $n \times q$  matrices  $\mathbf{f} = (\mathbf{f}_1, \dots, \mathbf{f}_p)$  and  $\mathbf{g} = (\mathbf{g}_1, \dots, \mathbf{g}_q)$  represent the  $p$ - and  $q$ -dimensional input and output signals on the vertices of the shape,

$\mathbf{W} = (w_{l,l'})$  is a  $p \times q$  matrix representing the learnable weights and the convolution is understood in the sense of (6.4).

**Discrete patch operator** On triangular meshes, a discrete local system of geodesic polar coordinates has  $n_\theta$  angular and  $n_\rho$  radial bins.

Starting with a vertex  $\mathbf{x}_i$ , we first partition the 1-ring of  $\mathbf{x}_i$  by  $n_\theta$  rays into equi-angular bins, aligning the first ray with one of the edges (Figure 6.3).

Next, we propagate the rays into adjacent triangles using an unfolding procedure resembling the one used by Kimmel and Sethian [KS98], producing poly-lines that form the angular bins (see Figure 6.3). Radial bins are created as level sets of the geodesic distance function computed using fast marching [KS98].

We represent the discrete patch operator as an  $n_\theta n_\rho n \times n$  matrix  $\mathbf{D}$  applied to a vector  $n \times p$  sampling the  $p$ -dimensional input signal on the mesh vertices and producing a patch  $n_\theta n_\rho \times 1$  (which can be arranged as the matrix  $n_\theta \times n_\rho$  representing a polar histogram) for each input dimension  $l = 1, \dots, p$ . The matrix  $\mathbf{D}$  is very sparse since only the values of the function at a few nearby vertices contribute to each local system of geodesic polar coordinates.

**Angular ambiguity** One of the main drawbacks of such definition of the patch operator is the ambiguity of the angular coordinate origin: if the origin of the angular coordinate changes from point to point, the resulting patches will suffer from a misalignment along the angular axis. As a result, even patches corresponding to close points  $x, y \in \mathcal{X}$ ,  $y \in \mathcal{N}(x)$  will result in arbitrary different representations.

To overcome this problem, in the paper [MBBV15] we proposed a two-stages solution: first, a *geodesic convolution layer* computes the geodesic convolution result for all  $n_\theta$  rotations of the filters,

$$\mathbf{g}_l^{\Delta\theta}(x) = \sum_{l'=1}^p (\mathbf{f}_{l'} * w_{l,l'}^{\Delta\theta})(x), \quad (6.5)$$

where  $w_{l,l'}^{\Delta\theta}(\theta, \rho) = w_{l,l'}(\theta + \Delta\theta, \rho)$  are the coefficients of the  $p$ th filter in the  $q$ th filter bank rotated by  $\Delta\theta = 0, 2\pi/n_\theta, \dots, 2\pi(n_\theta - 1)/n_\theta$ . Second, an *angular max pooling* procedure is considered. Angular max-pooling can be implemented by a fixed layer that computes the maximum over the filter rotations,

$$\mathbf{g}_l(x) = \max_{\Delta\theta} \mathbf{g}_l^{\Delta\theta}(x), \quad (6.6)$$

where  $\mathbf{g}_l^{\Delta\theta}(x)$  is the output of the geodesic convolution layer (6.5).

The combination of the two stages described above leads to the following alternative definition of the *geodesic convolution*

$$(f * w)(x) = \max_{\Delta\theta \in [0, 2\pi)} \int_0^{2\pi} \int_0^{\rho_{\max}} w(\theta + \Delta\theta, \rho)(D(x)f)(\theta, \rho) d\rho d\theta, \quad (6.7)$$

where the maximum over  $\Delta\theta$  is necessary to remove the ambiguity of the choice of the origin of the angular coordinate.

## 6.2 Anisotropic diffusion descriptors

One of the notable drawbacks of spectral descriptors discussed in Section 2.7 is the fact that they are *isotropic*, i.e. they ignore directional information. Such directional information, however, may carry important hints about the local structure of the surface (essential for the construction of a good feature descriptor). Furthermore, intrinsic descriptors are ambiguous under intrinsic symmetries: given an intrinsic symmetry  $\eta: \mathcal{X} \rightarrow \mathcal{X}$ , an intrinsic descriptor  $\mathbf{f} \in L^2(\mathcal{X})$  is invariant to it, i.e.  $\mathbf{f} \circ \eta = \mathbf{f}$ .

*Anisotropic diffusion descriptors* (ADD) [BMR<sup>+</sup>16] are a class of direction-sensitive spectral features descriptors defined by applying some learnable filters over anisotropic diffusion kernels on meshes and point clouds, which we will present below.

**Isotropic heat diffusion** In Section 2.5, we saw how the heat propagation on a shape  $\mathcal{X}$  is governed by the heat diffusion equation (2.4). In particular, given as initial heat distribution a delta function centered on  $x \in \mathcal{X}$ , the heat distribution on  $\mathcal{X}$  after some time  $t$  is represented by the heat kernel  $h_t(x, \cdot)$ .

The heat diffusion equation (2.4) tacitly assumes that the heat conduction properties of the manifold are constant at every point. As a result, the heat kernel  $h_t(x, \cdot)$  is *isotropic*, i.e. it diffuses equally in all directions (Figure 6.4, leftmost). The diffusion time  $t$  acts as a parameter influencing the spread of the kernel.

**Anisotropic heat diffusion** If, instead, the heat conductivity properties of the shape  $\mathcal{X}$  changes from point to point, the heat equation takes the more general form

$$f_t(x, t) = -\operatorname{div}(\mathbf{A}(x)\nabla f(x, t)), \quad (6.8)$$

where  $\mathbf{A}(x)$  is the *thermal conductivity tensor* (in case of two-dimensional manifolds, a  $2 \times 2$  matrix applied to the intrinsic gradient in the tangent plane at each

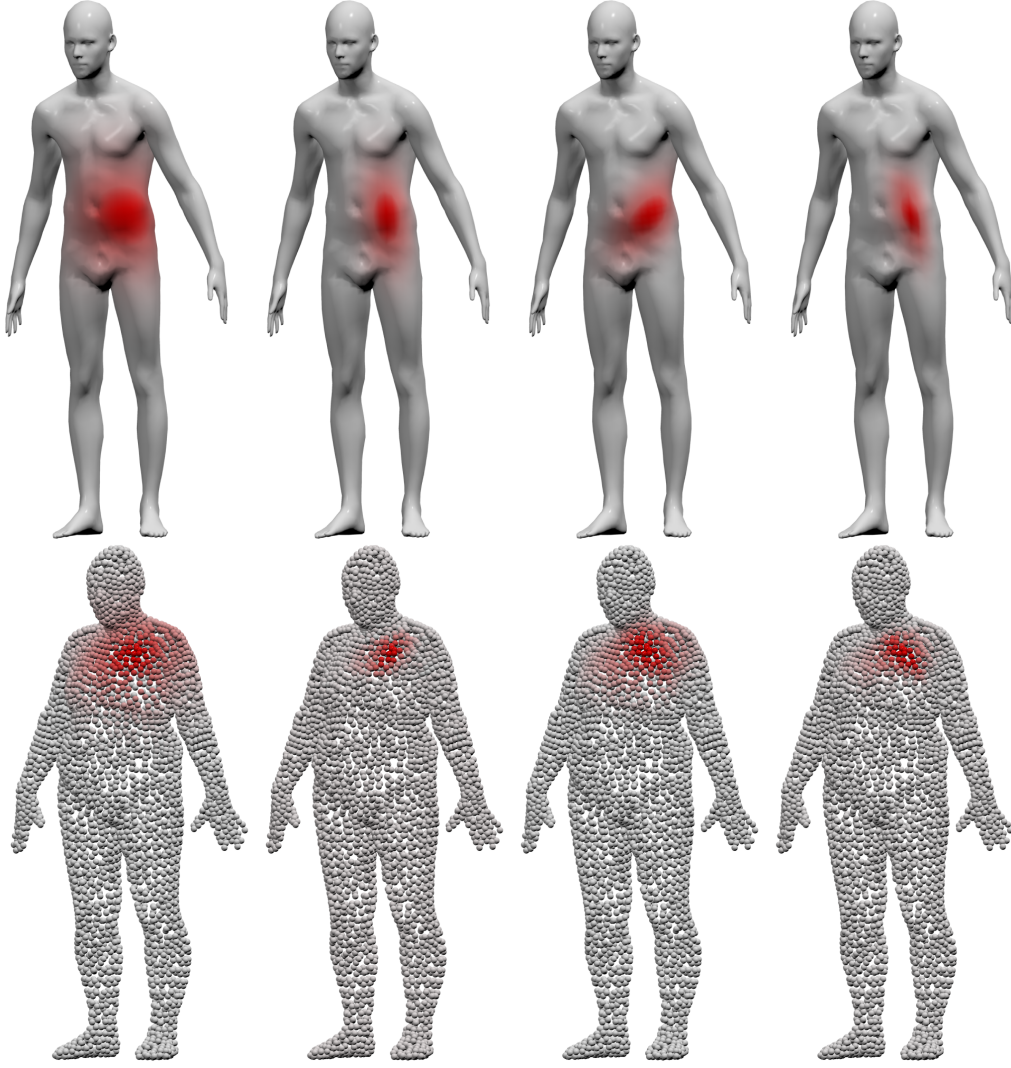


Figure 6.4. Visualization of different heat kernels on meshes (top row) and point clouds (bottom row). Red colours represent high values. *Leftmost*: example of an isotropic heat kernel. *Remaining*: examples of anisotropic heat kernels for different rotation angles  $\theta$  and anisotropy coefficient  $\alpha$ .

point). This formulation allows modeling an *anisotropic* heat flow, i.e. a position- and direction-dependent heat flow.

Andreux et al. [ARAC14] proposed to consider anisotropic diffusion on shapes driven by the surface curvature. For each point  $x \in X$ , a basis for the tangent plane  $T_x X$  is formed by considering the principal curvature directions  $\mathbf{v}_m(x), \mathbf{v}_M(x)$  (minimum and maximum curvature directions, respectively). With respect to



this basis, Andreux et al. [ARAC14] proposed to define the thermal conductivity tensor as

$$\mathbf{A}_\alpha(x) = \begin{bmatrix} \psi_\alpha(K_M(x)) & \\ & \psi_\alpha(K_m(x)) \end{bmatrix} \quad (6.9)$$

with  $\psi_\alpha(x) = 1/(1+\alpha|x|)$ . Such a thermal conductivity tensor drives the diffusion in the direction of the maximum curvature: the parameter  $\alpha$  controls the degree of anisotropy ( $\alpha = 0$  corresponds to the classical isotropic case), and the amount of diffusion depends on the extrinsic principal curvatures  $K_m(x), K_M(x)$ .

In our papers [BMR<sup>+</sup>16; BMRB16] instead, we proposed to consider a more general thermal conductivity tensor of the form

$$\mathbf{A}_{\alpha,\theta}(x) = \mathbf{R}_\theta(x) \begin{bmatrix} \alpha & \\ & 1 \end{bmatrix} \mathbf{R}_\theta^\top(x), \quad (6.10)$$

where the  $2 \times 2$  matrix  $\mathbf{R}_\theta(x)$  performs a rotation by an angle  $\theta$  w.r.t. to the maximum curvature direction  $\mathbf{v}_M(x)$ , and  $\alpha > 0$  is a parameter controlling the degree of anisotropy ( $\alpha = 1$  corresponds to the classical isotropic case).

**Anisotropic Laplacian** We refer to the operator

$$\Delta_{\alpha,\theta}f(x) = -\operatorname{div}(\mathbf{A}_{\alpha,\theta}(x)\nabla f(x)) \quad (6.11)$$

as the *anisotropic Laplacian*, and denote by  $\{\phi_{\alpha,\theta,i}, \lambda_{\alpha,\theta,i}\}_{i \geq 0}$  its eigenfunctions and eigenvalues, respectively.

Note that, strictly speaking, the anisotropic Laplacian (6.11) is not intrinsic: it depends on the principal curvature direction. If Formula (6.10) is used and we consider all the possible rotations  $\theta \in [0, 2\pi)$ , the Laplacian is intrinsic up to the choice of the origin of the angular coordinates  $\theta$ .

In the works [BMR<sup>+</sup>16; BMRB16], we showed that this ambiguity can be removed by fixing the angular coordinates origin using as reference direction the principal curvature direction  $\mathbf{v}_M$ .

**Anisotropic heat kernels** By analogy with the spectral definition of isotropic heat kernels (2.6), the *anisotropic heat kernel* can be defined as

$$h_{\alpha,\theta,t}(x, x') = \sum_{i \geq 1} e^{-t\lambda_{\alpha,\theta,i}} \phi_{\alpha,\theta,i}(x) \phi_{\alpha,\theta,i}(x'), \quad (6.12)$$

where  $\phi_{\alpha,\theta,i}$  and  $\lambda_{\alpha,\theta,i}$  are the eigenfunctions and eigenvalues of the anisotropic Laplacian  $\Delta_{\alpha,\theta}$ , respectively. The only difference with Equation (2.6) is the fact that the anisotropic heat kernel  $h_{\alpha,\theta,t}$  depends on two additional parameters, the anisotropy coefficient  $\alpha$  and the rotation angle  $\theta$ . Figure 6.4 shows some examples of anisotropic heat kernels computed at different rotations  $\theta$  and anisotropies  $\alpha$ .

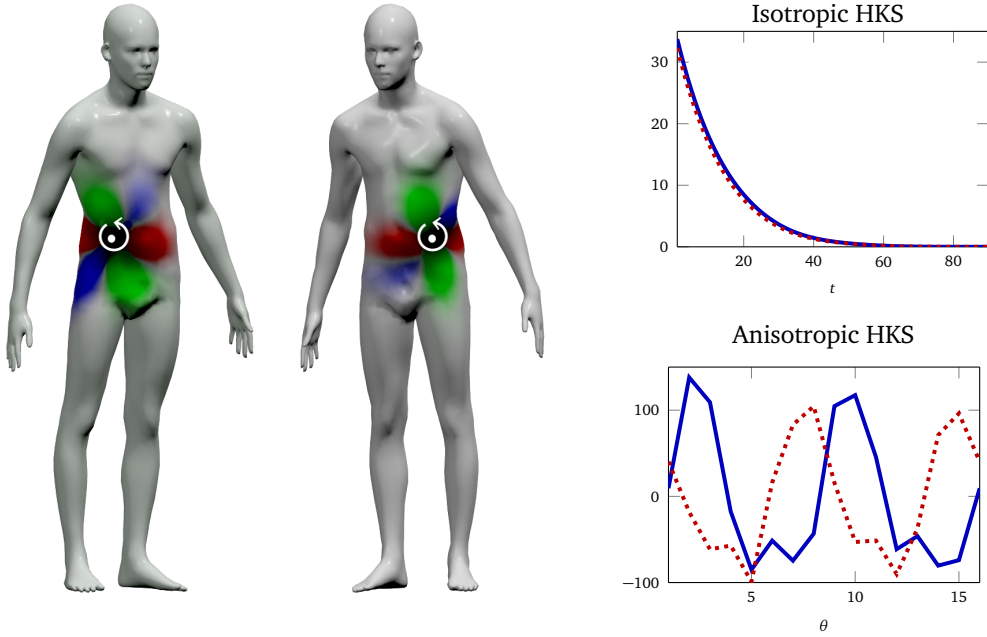


Figure 6.5. Illustration of the intrinsic symmetry ambiguity. *Left*: two symmetric points on a shape (white) and the anisotropic heat kernels for different  $\theta$  computed at these points (similar colors encode similar angles). *Right*: the values of HKS (top) and anisotropic HKS (bottom) computed at two symmetric points (solid and dotted curves). Note that HKS is fully ambiguous (both curves coincide), while anisotropic HKS allows to distinguish between symmetric points (one curve is the reflection of the other).

**Anisotropic HKS** By analogy with the HKS definition [SOG09; GBAL09] (Section 2.7.1), we define an anisotropic version of the HKS, called *anisotropic HKS*, by considering the diagonal values  $h_{\alpha,\theta,t}(x, x)$  of the anisotropic heat kernel (6.12) and sampling  $t$  and  $\theta$  at values  $t_1, \dots, t_q$  and  $\theta_1, \dots, \theta_s$ , respectively ( $\alpha$  is used as a fixed parameter).

It is important to notice that since we use the principal curvature direction  $\mathbf{v}_M$  as the origin  $\theta = 0$  of the angular coordinate, and since the curvature is an extrinsic property, our descriptor is not ambiguous under bilateral intrinsic symmetry. In fact, intrinsic symmetry  $\eta$  reflects the angular coordinate,  $\mathbf{f}_\theta \circ \eta = \mathbf{f}_{-\theta}$  (here  $\mathbf{f}_\theta(x) = h_{\alpha,\theta,t}(x, x)$ ). This phenomenon is illustrated in Figure 6.5.

**Anisotropic spectral filters** Anisotropic HKS descriptors show that anisotropic spectral kernels carry rich information about local shape structures.

The basic idea behind ADD [BMR<sup>+</sup>16] is to replace the low-pass filters rep-

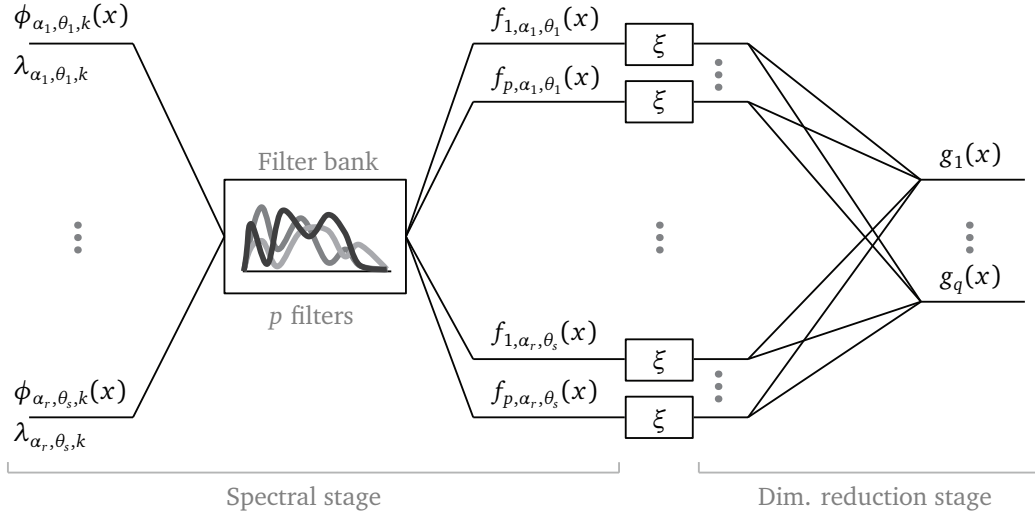


Figure 6.6. Example of a simple single-layer neural network architecture implementing the anisotropic descriptors proposed in our paper [BMR<sup>+</sup>16]. The inputs are spectral decompositions of anisotropic Laplacians with anisotropy  $\alpha$  at angle  $\theta_1, \dots, \theta_s$  w.r.t. the principal curvature direction. In the spectral stage, a learnable spectral bank of  $p$  filters (shared for all the  $s$  directions) is used to create  $ps$  directed kernels.  $\xi$  denotes the ReLU nonlinearity. The dimensionality reduction stage then reduces the descriptor to  $q$  output dimensions. Deeper architectures may contain additional layers in both stages.

representing the heat kernels (which are not necessarily best suited for some applications) with learnable filters which can be trained to fit the desired application. More specifically, in the paper [BMR<sup>+</sup>16], we propose to learn non-linear anisotropic descriptors using deep (non-convolutional) neural networks with a tailored architecture similar to the one depicted in Figure 6.6.

The key innovation of ADD [BMR<sup>+</sup>16] is the so-called *anisotropic spectral filter* (ASF) layer. An anisotropic spectral filter layer takes in input the first  $k$  eigenvalues and eigenvectors of the anisotropic Laplacians  $\Delta_{\alpha_l \theta_{l'}}$  for different anisotropies  $(\alpha_l)_{l=1, \dots, r}$  and equally-spaced rotation angles  $(\theta_{l'})_{l'=1, \dots, \theta_s}$  and performs the spectral filtering

$$f_{l, l', m}(x) = \sum_{i=1}^k \tau_m(\lambda_{\alpha_l \theta_{l'} i}) \phi_{\alpha_l \theta_{l'} i}^2, \quad (6.13)$$

for  $m = 1, \dots, p$ ,  $l = 1, \dots, r$ , and  $l' = 1, \dots, s$ , where the filter transfer functions  $\tau_m(\lambda)$  are parametrized as in Equation (4.5) and are learned.

The filters are shared across all rotations and anisotropies, resembling the shared connectivity of CNNs and allowing to reduce the number of degrees of

freedom in the model, thus reducing the chance of overfitting.

Finally, ADD [BMR<sup>+</sup>16] is a general framework which includes, as particular cases, HKS [SOG09; GBAL09], WKS [ASC11] and OSD [LB14]. HKS corresponds to a single ASF layer with fixed lowpass filters, WKS corresponds to the same fixed layer but with band-pass filters, and OSD to a single learnable ASF layer without non-linearities.

### 6.3 Anisotropic diffusion convolutional neural networks

In the follow-up work, called *anisotropic diffusion convolutional neural networks* (ACNN) [BMRB16], we considered the same anisotropic construction of ADD [BMR<sup>+</sup>16] but as an alternative definition of the patch operator, to extend the convolution operation to non-Euclidean manifolds.

The key idea of ACNN [BMRB16] is to consider the anisotropic heat kernels (6.12) as the local weighting functions for the construction of the *anisotropic patch operator*

$$(D(x)f)(\theta, t) = \frac{\int_{\mathcal{X}} h_{\alpha, \theta, t}(x, x') f(x') dx'}{\int_{\mathcal{X}} h_{\alpha, \theta, t}(x, x') dx'},$$

for some fixed anisotropy level  $\alpha > 1$ , which is a hyperparameter of the model. This way, the values of  $f$  around the point  $x$  are mapped to a local system of coordinates  $(\theta, t)$  that behave like a polar system ( $t$  denotes the scale of the heat kernel, while  $\theta$  its orientation). As a result, the *anisotropic diffusion convolution* is defined as

$$(f * w)(x) = \int_0^{2\pi} \int_0^{t_{\max}} w(\theta, t) (D(x)f)(\theta, t) dt d\theta. \quad (6.14)$$

It is important to notice that, unlike the arbitrarily oriented geodesic patches of GCNN [MBBV15], which require the additional angular max pooling operation to avoid ambiguities, in the ACNN construction [BMRB16] we can simply use the principal curvature direction  $\mathbf{v}_M$  as the reference origin of the angular coordinate  $\theta = 0$ .

The *anisotropic convolutional layer* of ACNN [BMRB16] is defined as

$$\mathbf{g}_l(x) = \sum_{l'=1}^p (\mathbf{f}_{l'} * \mathbf{W}_{l,l'})(x),$$

where the convolution is intended in the sense of Equation (6.14). Here  $\mathbf{f}_{l'}$ ,  $l' = 1, \dots, p$ , is the  $p$ -dimensional input signal,  $\mathbf{W}_{l,l'}$  are the learnable coefficients of the  $p$ th filter in the  $q$ th filter bank, and  $\mathbf{g}_l$ ,  $l = 1, \dots, q$  is the  $q$ -dimensional output feature.

Such an approach has a few major advantages compared to previous intrinsic CNN models. First, being a spectral construction, our patch operator can be applied to any shape representation (like LSCNN and unlike GCNN). Second, being defined in the spatial domain, the patches and the resulting filters have a clear geometric interpretation (unlike LSCNN). Third, our construction accounts for local directional patterns (like GCNN and unlike LSCNN). Fourth, the heat kernels are always well defined independently of the injectivity radius of the manifold (unlike GCNN).

## 6.4 Mixture model convolutional neural networks

GCNN [MBBV15] and ACNN [BMRB16] are both charting-based methods and differ in the way the local weighting functions of their patch operators are constructed. Rather than proposing yet another hand-crafted patch operator, in the follow-up paper [MBM<sup>+</sup>17], called *mixture model convolutional neural networks* (MoNet), we proposed to learn the optimal patch operator from examples, leading to the first general spatial-domain framework for deep learning on non-Euclidean domains.

The key contribution of MoNet [MBM<sup>+</sup>17] is to further parametrize the weighting functions  $v_1, \dots, v_k$  of the patch operator in Equation (6.1) in terms of local coordinates defined on the manifold  $\mathcal{X}$ . More in detail, given a point  $x \in \mathcal{X}$ , we denote by  $\mathcal{N}(x)$  the neighborhood of  $x$ . To each point  $x' \in \mathcal{N}(x)$  we associate a  $d$ -dimensional vector of *pseudo-coordinates*  $\mathbf{u}(x, x')$ . In these coordinates, we define a weighting function  $\mathbf{v}_\Theta(\mathbf{u}) = (v_1(\mathbf{u}), \dots, v_j(\mathbf{u}))$ , which is parametrized by some learnable parameters  $\Theta$ .

The patch operator can therefore be written in the following general form

$$D_j(x)f = \int_{\mathcal{X}} v_j(\mathbf{u}(x, x'))f(x'), \quad j = 1, \dots, k \quad (6.15)$$

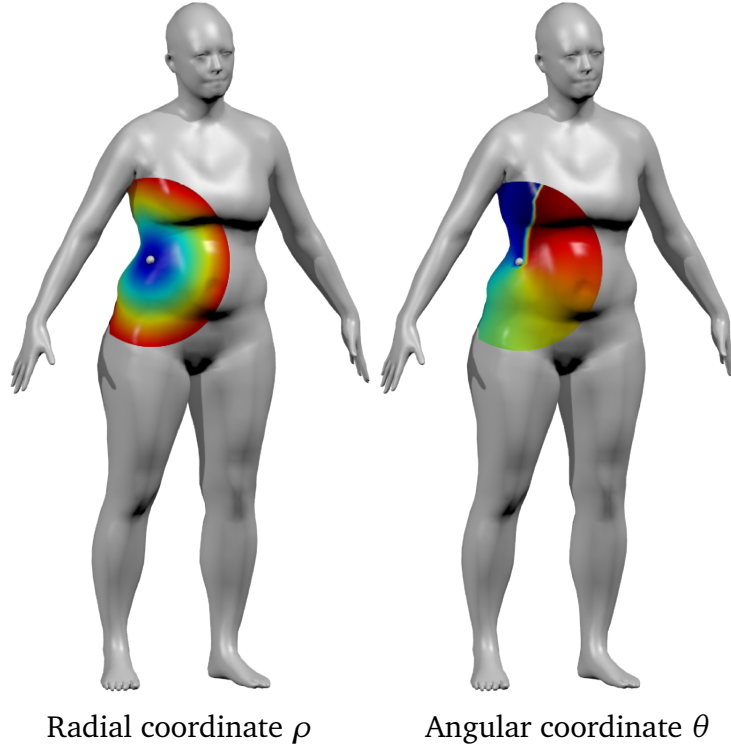


Figure 6.7. Intrinsic local polar coordinates  $\rho, \theta$  on the manifold around a point on the shape, marked in white.

where  $k$  represents the dimensionality of the extracted patch. A spatial generalization of the convolution operation to non-Euclidean domains is then given by the template-matching procedure of Equation (6.2).

The two key choices in the MoNet construction [MBM<sup>+</sup>17] are the pseudo-coordinates  $\mathbf{u}$  and the weighting functions  $v_1(\mathbf{u}), \dots, v_k(\mathbf{u})$ . In particular, as pseudo coordinates  $\mathbf{u}$  we considered a system of polar geodesic coordinates similar to the one depicted in Figure 6.8.

For what concerns weighting functions, instead, rather than using fixed hand-crafted ones, we consider parametric kernels with learnable parameters. In particular, we found that a convenient choice is

$$v_j(\mathbf{u}) = \exp\left(-\frac{1}{2}(\mathbf{u} - \boldsymbol{\mu}_j)^\top \boldsymbol{\Sigma}_j^{-1}(\mathbf{u} - \boldsymbol{\mu}_j)\right), \quad (6.16)$$

where  $\boldsymbol{\Sigma}_j$  and  $\boldsymbol{\mu}_j$  are learnable  $d \times d$  and  $d \times 1$  covariance matrix and mean vector of a Gaussian kernel, respectively. Formula (6.16) can thus be interpreted

Table 6.1. Several CNN-type geometric deep learning methods on manifolds can be obtained as a particular setting of [MBM<sup>+</sup>17] with an appropriate choice of the pseudo-coordinates and weight functions in the definition of the patch operator.  $\mathbf{x}$  denotes the reference point (center of the patch) and  $y$  a point within the patch. In the table below,  $\mathbf{x}$  denotes the Euclidean coordinates on a regular grid.  $\bar{\alpha}, \bar{\sigma}_\rho, \bar{\sigma}_\theta$  and  $\bar{\mathbf{u}}_j, \bar{\theta}_j, j = 1, \dots, k$  denote fixed parameters of the weight functions.

Method	$\mathbf{u}(x, y)$	Weight function $v_j(\mathbf{u}), j = 1, \dots, k$
CNN	$\mathbf{x}(x, y) = \mathbf{x}(y) - \mathbf{x}(x)$	$\delta(\mathbf{u} - \bar{\mathbf{u}}_j)$
GCNN	$\rho(x, y), \theta(x, y)$	$\exp\left(-\frac{1}{2}(\mathbf{u} - \bar{\mathbf{u}}_j)^\top \begin{pmatrix} \bar{\sigma}_\rho^2 & \\ & \bar{\sigma}_\theta^2 \end{pmatrix}^{-1} (\mathbf{u} - \bar{\mathbf{u}}_j)\right)$
ACNN	$\rho(x, y), \theta(x, y)$	$\exp\left(-\frac{1}{2}\mathbf{u}^\top \mathbf{R}_{\bar{\theta}_j} \begin{pmatrix} \bar{\alpha} & 1 \end{pmatrix} \mathbf{R}_{\bar{\theta}_j}^\top \mathbf{u}\right)$

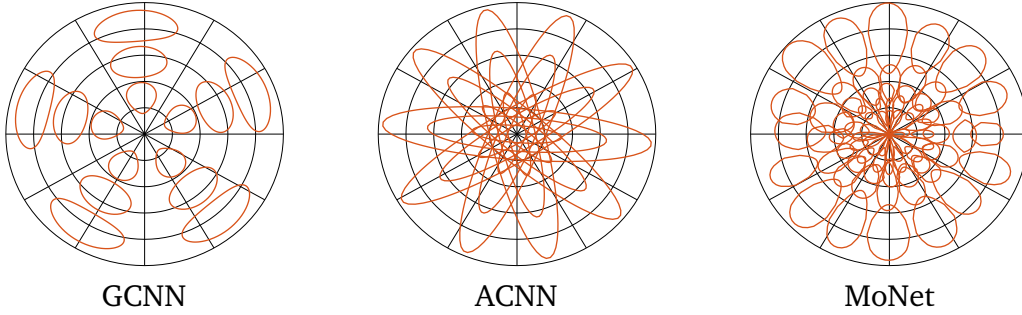


Figure 6.8. Patch operator weighting functions  $v_i(\rho, \theta)$  used in different generalizations of convolution on the manifold (hand-crafted in both GCNN and ACNN and learned in MoNet). All kernels are  $L_\infty$ -normalized; red curves represent the 0.5 level set.

as a Gaussian mixture model (GMM), which is the reason behind the name of this approach. We further restrict the covariances to have diagonal form, resulting in  $2d$  parameters per kernel, and a total of  $2kd$  parameters for the patch operator. Finally, once the weights (6.16) are computed, the patch operator can be computed as shown in Equation (6.15).

Learning not only the filters but also the patch operators provides additional degrees of freedom to the MoNet architecture, which makes it currently the state-of-the-art approach in several applications.

It is also easy to see that this approach generalizes the previous models, which can be obtained as particular setting of the MoNet framework with appropriate definition of  $\mathbf{u}$  and  $v_j(\mathbf{u}), j = 1, \dots, k$ , as shown in Table 6.1. For example, GCNN

Method	Representation	Input	Generalizable	Filters	Directional
OSD [LB14]	Mesh, point cloud	Geometry	Yes	Spectral	No
ADD [BMR <sup>+</sup> 16]	Mesh, point cloud	Geometry	Yes	Spectral	Yes
SCNN [BZSL13]	Graph	Any	No	Spectral	No
LSCNN [BMM <sup>+</sup> 15]	Mesh, point cloud	Any	Yes	Spectral	No
GCNN [MBBV15]	Mesh	Any	Spatial	Yes	Yes
ACNN [BMRB16]	Mesh, point cloud	Any	Yes	Spatial	Yes
MoNet [MBM <sup>+</sup> 17]	Any	Any	Yes	Spatial	Yes

*Table 6.2.* Comparison of different machine learning methods. The MoNet [MBM<sup>+</sup>17] method combines all the best properties of the other models. Note that OSD and ADD are local spectral descriptors operating with intrinsic geometric information of the shape and cannot be applied to arbitrary input, unlike the convolutional models.

and ACNN can be obtained as a particular case of MoNet by considering fixed Gaussian kernels as weighting functions and local polar geodesic coordinates  $(\rho, \theta)$  as local pseudo-coordinates.

Figure 6.8 shows a comparison between the hand-crafted kernels of GCNN and ACNN (left and center) and the ones learned by MoNet (right).

Finally, Table 6.2 provides a comparison between the properties of the different machine learning methods presented so far in this thesis, with a particular attention to the intrinsic deep learning we developed. The MoNet model [MBM<sup>+</sup>17] is the most generic one and combines all the best properties of the other models.



## Chapter 7

# Learning shape descriptors with intrinsic deep learning

In Chapters 4, 5, and 6, we proposed different extensions of the traditional Euclidean convolutional layer (3.7) to non-Euclidean data, called *intrinsic convolutional layers*.

Intrinsic convolutional layers was the missing piece that prevented deep learning to spread and succeed in shape analysis applications as well: Euclidean CNNs can, indeed, be extended to manifold data by replacing the standard convolutional layer with intrinsic ones. We refer to the resulting architecture as *intrinsic CNNs* to distinguish it from the traditional one.

In this chapter, we provide an experimental evaluation of the performance of intrinsic CNNs on challenging synthetic and real data in a basic problem of shape analysis: *shape similarity*. More precisely, we will test the performance of intrinsic CNNs in the problem of learning *local shape descriptors* and in the problem of learning global shape descriptors for *shape retrieval*. In particular, for the first problem we report the results from our papers [BMM<sup>+</sup>15; MBBV15; BMR<sup>+</sup>16], while for shape retrieval we show the results published in [MBBV15].

In the next chapter we will provide an experimental evaluation of the performance of intrinsic CNNs on another basic problem of shape analysis: *shape correspondence*. In particular, we will show the results achieved by our methods [MBBV15; BMR<sup>+</sup>16; BMRB16; MBM<sup>+</sup>17].

Resorting to machine learning methods allows to develop models that can adapt to the specific task with minimal manual adjustments, contrary to what happens for hand-crafted methods (Figure 7.1).

Finally, we show that the proposed deep models allow to learn state-of-the-art features with superior performances on both other (shallower) machine learning and hand-crafted models.

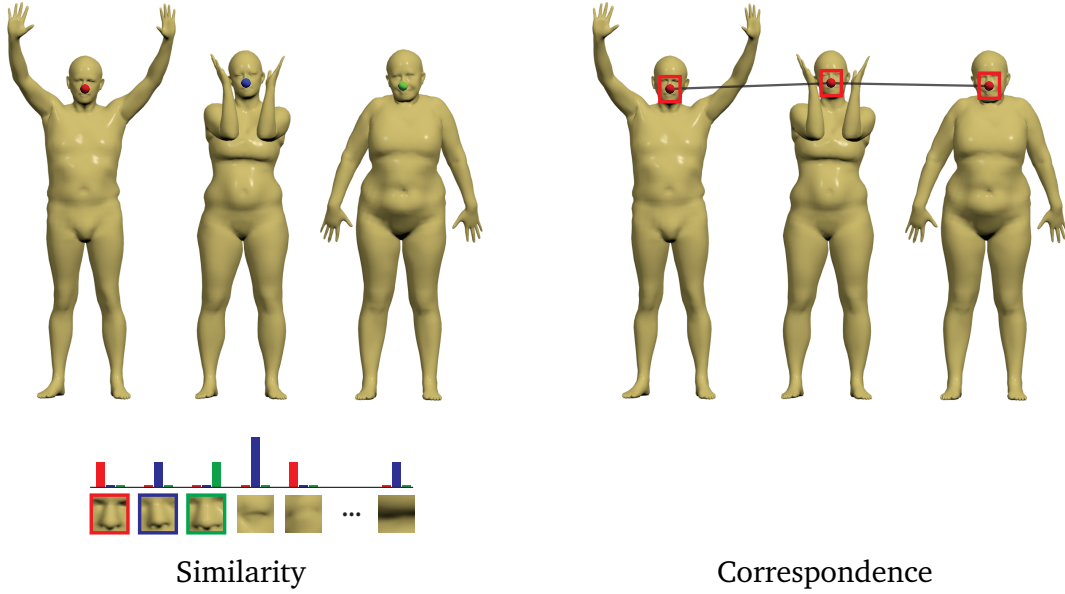


Figure 7.1. *Left*: features used for shape similarity tasks should be specific to a shape within a class, allowing to correctly distinguish between different people denoted by red, blue, and green dots, respectively. In this case we are showing the features corresponding to the noses of three people. Ideally, each nose should have distinguishable features since they belong to different people. *Right*: To the contrary, correspondence should ideally manifest invariance across the shape class: to be able to correctly identify the three noses marked in red as the corresponding part on the three shapes, their features should be as close as possible (i.e. independent on the specific person). Hand-crafting the right feature for each application is a very challenging task.

## 7.1 Intrinsic convolutional neural networks

Intrinsic CNNs are the extension of traditional Euclidean CNNs to manifold data by replacing traditional convolutional layers with intrinsic convolutional ones. As a result, an intrinsic CNN is a deep model defined as the composition of several of such layers and, possibly, some additional ones. The layers are applied subsequently, i.e. the output of the previous layer is used as input into the subsequent one (Figure 7.2).

The prototypical intrinsic CNN is thus a non-linear hierarchical parametric function of the form

$$U_{\Theta}(\mathbf{f}) = (F_{\Theta^{(k)}} \circ C_{\Theta^{(k-1)}} \circ F_{\Theta^{(k-2)}} \circ \dots \circ C_{\Theta^{(1)}})(\mathbf{f}), \quad (7.1)$$

where  $\Theta^{(i)}$ ,  $i = 1, \dots, k$ , represent the parameters of the  $i$ th layer. The parameters

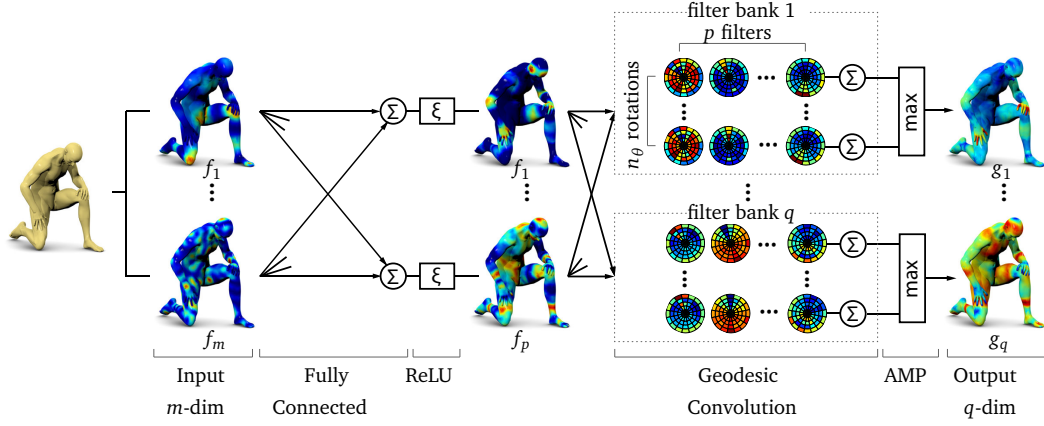


Figure 7.2. A simple example of an intrinsic convolutional neural network architecture. The network takes as input an off-the-shelf  $m$ -dimensional hand-crafted local shape descriptor for each vertex, then a linear dimensionality reduction layer is applied to reduce the input dimension to  $p < m$ , followed by a ReLU non-linearity. Finally, an intrinsic convolutional layer (e.g. the geodesic convolution layer (6.4)) with  $q$  banks of  $p$  filters is followed by an angular max pooling layer to remove the ambiguity of the choice of angular coordinate origin, which produces a  $q$ -dimensional output feature for each vertex.

of the intrinsic CNN  $U_{\Theta}$  are defined by collecting the parameters of each layer, i.e.  $\Theta = \{\Theta_i : i = 1, \dots, k\}$ .

Typically, fully connected layers  $F_{\Theta}$  (3.5) and intrinsic convolutional layers  $C_{\Theta}$  are considered, but other ad-hoc layers may be considered as well. In the following, we will provide more details about the specific architectures employed by each of the contributions [BMM<sup>+</sup>15; MBBV15; BMR<sup>+</sup>16].

Figure 7.2 shows a toy example of an intrinsic CNN architecture  $U_{\Theta}(\mathbf{f}) = C_{\Theta(2)}(P_{\Theta(1)}(\mathbf{f}))$  composed of a fully connected layer  $P_{\Theta(1)}$  and an intrinsic convolutional layer  $C_{\Theta(2)}$  (more specifically,  $C_{\Theta(2)}$  is composed of a geodesic convolutional layer (6.7) followed by a fixed angular max-pooling layer (6.6)).

The model  $U_{\Theta}$  is applied point-wise to the  $m$ -dimensional input data  $\mathbf{f}(x) = (f_1(x), \dots, f_m(x))$  and produces some point-wise  $q$ -dimensional output features  $\mathbf{g}(x) = (g_1(x), \dots, g_q(x))$  by minimizing some task-specific optimization problem.

Typically, as input data  $\mathbf{f}$  we consider some simple descriptor capturing the geometric content of the shape. Examples include *geometry vectors* (2.19) [LB14] and *signatures of histograms of orientations* (SHOT) [STDS14]. Alternatively, as input signal  $\mathbf{f}(x)$  one may consider the  $xyz$ -coordinates of the shape vertices, as well as the components of the normal vectors  $\mathbf{n}(x) = (n_1, n_2, n_3)$ . If the texture of

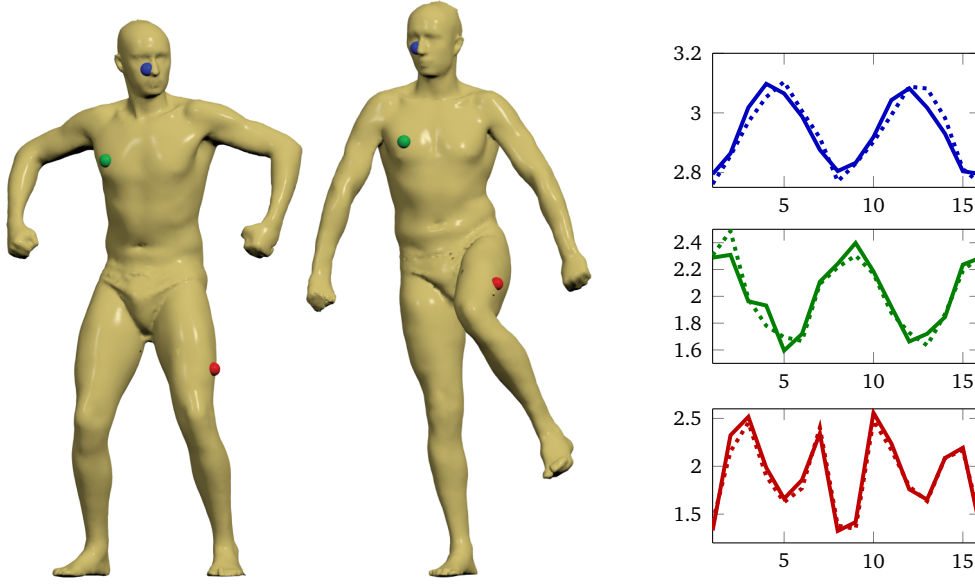


Figure 7.3. Local descriptors on two different poses of the same shape at three points marked in red, green, and blue. Similar colours encode descriptors from corresponding points. Solid lines represent the descriptors on the shape on the left, dashed lines the ones of the shape on the right.

the shape is provided, then the RGB-values corresponding to each vertex can be considered as the input signal, similarly to what happens with natural images.

In the following, we will see how intrinsic CNNs can be applied to two basic problems in shape analysis: local shape descriptors and shape retrieval.

## 7.2 Learning local shape descriptors

The point-wise application of an intrinsic CNN to some input feature vector  $\mathbf{f}(x)$  creates a feature map  $\mathbf{g}(x)$  that can be regarded to as a dense local descriptor for the points  $x \in \mathcal{X}$ .

Ideally, a local descriptor should be as similar as possible at corresponding points across a collection of shapes, and as dissimilar as possible at non-corresponding points.

Such situation is illustrated in Figure 7.3, where the descriptors of the points highlighted by red, green, and blue spheres are represented by the curves of the corresponding color on the right. Solid curves correspond to the descriptors on the shape on the left, dashed curves to descriptors of the rightmost one. Solid and dashed curves are very similar and curves of different colors are very different:

this is an ideal behaviour for local descriptors. To simulate such behaviour, we need to consider suitable training set and cost function.

**Training set** In the works [BMM<sup>+</sup>15; MBBV15; BMR<sup>+</sup>16] we assumed to be provided with examples of points  $(x, x^+)$  from different shapes that are known to be similar, as well as pairs of points  $(x, x^-)$  known to be dissimilar. More specifically, the training set is formed by the collection of pairs of similar points  $\mathcal{T}_+ = \{(x, x^+)\}$ , called *positives*, and the collection of pairs of dissimilar points  $\mathcal{T}_- = \{(x, x^-)\}$ , called *negatives*. With reference to Figure 7.3, the positive set  $\mathcal{T}_+$  would be formed by pairs of the same color, while the negative set  $\mathcal{T}_-$  would be formed by pairs of dissimilar colors. The most straightforward way to obtain such training sets is from known correspondences between some of the shapes in the collection.

**Cost function** Given such training set, our goal is to make the output  $\mathbf{g}(x) = (U_{\Theta}(\mathbf{f}))(x)$  of the intrinsic CNN (7.1) as similar as possible at positives and as dissimilar as possible at negatives across a collection of shapes.

For this purpose, we can consider a *siamese network* configuration [SP93; BGL<sup>+</sup>94; HCL06; SSTF<sup>+</sup>14] composed of two identical copies of the same intrinsic CNN model  $U_{\Theta}$  (7.1) sharing the same parametrization. During training, optimal parameters of  $U_{\Theta}$  are estimated by minimizing the *siamese loss*

$$\mathcal{L}(\Theta) = (1 - \gamma)\mathcal{L}_+(\Theta) + \gamma\mathcal{L}_-(\Theta), \quad (7.2)$$

where

$$\mathcal{L}_+(\Theta) = \frac{1}{|\mathcal{T}_+|} \sum_{i=1}^{|\mathcal{T}_+|} \|U_{\Theta}(\mathbf{f}_i) - U_{\Theta}(\mathbf{f}_i^+)\|^2, \quad (7.3)$$

and

$$\mathcal{L}_-(\Theta) = \frac{1}{|\mathcal{T}_-|} \sum_{i=1}^{|\mathcal{T}_-|} (\mu - \|U_{\Theta}(\mathbf{f}_i) - U_{\Theta}(\mathbf{f}_i^-)\|)_+^2, \quad (7.4)$$

are the positive and negative losses, respectively. Here,  $\lambda \in [0, 1]$  is a parameter trading off between the positive and negative losses, the *margin*  $\mu$  is a scalar hyperparameter of the model and  $(\cdot)_+ = \max\{0, \cdot\}$ .

The siamese loss (7.2) can be interpreted as follows: the positive loss (7.3) tries to minimize the distance between the output features corresponding to positive points, while the negative loss (7.4) tries to maximize the distance between features corresponding to negative points. To avoid a trivial solution, the negative loss accounts for a margin term  $\mu$  representing the maximum value allowed

for the distance between features corresponding to negative points. Once such distance reaches the margin  $\mu$ , the contribution of that term to the optimization vanishes.

### 7.2.1 Performance evaluation

Once the training procedure is completed and the optimal parameters  $\Theta^*$  are obtained, we can evaluate the performance of the descriptors obtained by the model  $U_{\Theta^*}$  on the test set.

We consider both a qualitative and a quantitative evaluation of the descriptor performance.

The qualitative evaluation consists in showing the *similarity map* of the learned descriptors. A similarity map depicts the Euclidean distance in the descriptor space between the descriptor at a selected point, called *reference point*, and the rest of the points on the same shape as well as on other shapes. Usually, different kind of noises are considered to better assess the descriptor robustness. The distances are represented in a red-blue colormap, where small distances are represented by cold colors, while large distances are represented by hot colors. Ideal descriptors produce a distance map with small values localized around the reference point and large values elsewhere.

The quantitative evaluation consist in measuring the descriptor performance according to three metrics: the *cumulative match characteristic* (CMC), the *receiver operator characteristic* (ROC), and the *Princeton protocol* ([KLF11]). The CMC evaluates the probability of a correct correspondence among the  $k$  nearest neighbours in the descriptor space, as a function of the parameter  $k$ . The ROC measures the percentage of positive and negative pairs falling below various thresholds of their distance in the descriptor space (*true positive* and *negative rates*, respectively). The Princeton protocol measures the quality of the correspondence obtained by matching nearest neighbours in the descriptor space. More specifically, it measures the percentage of correct matches at most  $r$ -geodesically distant from the groundtruth correspondence, for different values of the geodesic radius  $r$ . The percentage of perfect matches coincides with the value  $r = 0$ .

### 7.2.2 Datasets

To test the performances of LSCNNs we considered two public-domain datasets of scanned human shapes: SCAPE [ASK<sup>+</sup>05] and FAUST [BRLB14], and the synthetic human shapes of the public-domain dataset TOSCA [BBK08]. SCAPE [ASK<sup>+</sup>05] contains 71 different poses of the same male person, while FAUST

[BRLB14] contains 10 subjects (both male and female) in 10 different poses each for a total of 100 shapes. The latter is the most recent and particularly challenging, given a high variability of non-isometric deformations as well as significant variability between different human subjects. The meshes in SCAPE were resampled to 12.5K vertices, whereas for FAUST we used the registration meshes ( $\approx 7K$  vertices) without further pre-processing. In addition we scaled all shapes to have unit geodesic diameter. In both datasets, groundtruth point-wise correspondence between the shapes was known for all points.

Each dataset was split into disjoint training, validation, and test sets. On the FAUST dataset subjects 1 – 7 were used for training (10 poses per subject, a total of 70 shapes), subject 8 (10 shapes) for validation, and subjects 9 – 10 for testing (total of 20 shapes). On SCAPE, we used shapes 20 – 29 and 50 – 70 for training (total 31 shapes), five different shapes for validation, and the 40 remaining shapes for testing.

TOSCA contains two male subjects (Michael and David) and a female subject (Victoria) in different poses. The meshes from TOSCA were resampled to 10K vertices. In this case, the ground-truth correspondence is not known, therefore we considered TOSCA only as a test set.

The positive and negative sets of vertex pairs required for training were generated on the fly, to keep the storage requirements for the training algorithm, via uniform stochastic sampling. Each point on the first shape has only a single groundtruth match (given by the known one-to-one correspondence) and is assigned to one out of  $n - 1$  possible negatives: first, sample two shapes, then form the positive set with all corresponding points, and finally, form the negative set with first shape vertices and a random permutation of the ones of the second shape.

This strategy differs from [LB14] who considered only points on the same shape. The advantage of our sampling strategy is that it allows learning invariance also across several poses and subjects.

### 7.2.3 LSCNN experiments and results

**Architectures** In our experiments we considered as input data  $m = 150$ -dimensional geometry vectors, computed according to Equation (2.18) using B-spline bases [LB14].

We considered two different configurations of the LSCNN model. The first one, denoted with LSCNN1, consists of a fully connected layer (reducing the dimensionality of the  $m = 150$ -dimensional input descriptors to  $p = 16$  dimensions), followed by an intrinsic convolutional layer producing  $q = 16$ -dimensional

descriptors. In this configuration, the intrinsic convolutional layer employs a fixed WFT Gaussian window  $\gamma(\lambda) = e^{-\lambda^2/\sigma^2}$  with  $\sigma = 10^{-5}$ . The parameters of LSCNN1 that are learned are  $\Theta = \{(w_{qp}), (w_{qpk})\}$ , where  $w_{qp}$  are the parameters of the fully connected layer (3.5) and  $w_{qpk}$  are the parameters of the WFT layer (5.4). The second one, denoted with LSCNN2, is similar to LSCNN1 with the difference that now the WFT windows are also learned. In particular, we learn a different WFT window for each dimension for a total of 16 windows. The WFT windows are parametrized according to Equation (2.18). The learnable parameters of LSCNN2 are the same of LSCNN1 with the addition of the B-spline coefficients  $(\alpha_{pm})$  (2.18) of the learnable WFT windows, i.e.  $\Theta = \{(w_{qp}), (w_{qpk}), (\alpha_{pm})\}$ .

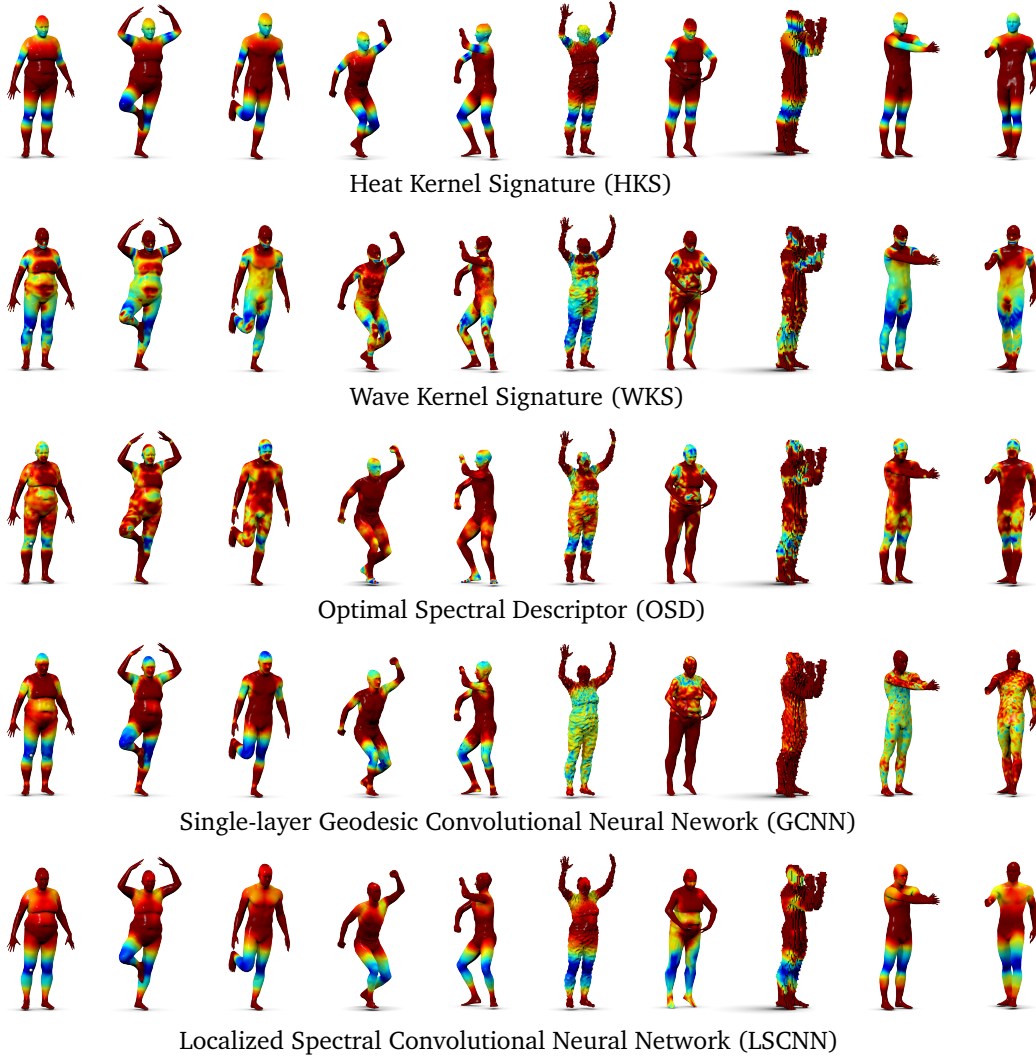
Interestingly, the hand-crafted spectral descriptors HKS [SOG09; GBAL09], WKS [ASC11], and the learnable descriptors OSD [LB14] can be thought as particular instances of the LSCNN framework corresponding to particular choices of the parameters  $\Theta$ . In particular, HKS can be implemented as a fixed fully connected layer where the weights are represented by low-pass filters  $\gamma_t(\lambda) = e^{-t\lambda}$ , WKS can be implemented as a fixed fully connected layer where the weights are represented by band-pass filters  $\gamma_{v,\sigma}(\lambda) = e^{(\log v - \log \lambda)/2\sigma^2}$ , and OSD can be implemented as a *linear* fully connected layer, i.e. a fully connected layer (3.5) without the activation function  $\xi$ . Thus, if the training set is informative enough and the training is performed correctly, descriptors learned with LSCNNs can perform only better than the above.

**Results** We compared the performance of LSCNN to HKS [SOG09; GBAL09], WKS [ASC11], OSD [LB14], and a single layer GCNN [MBBV15], denoted with GCNN1, using the settings provided by the respective authors. Additionally, we compared LSCNN also to a network FC1 consisting of a single non-linear fully connected layer: FC1 is compatible with OSD, with the addition of a ReLU non-linearity at the output. To make the comparison fair, all the descriptors were  $q = 16$ -dimensional as in [LB14].

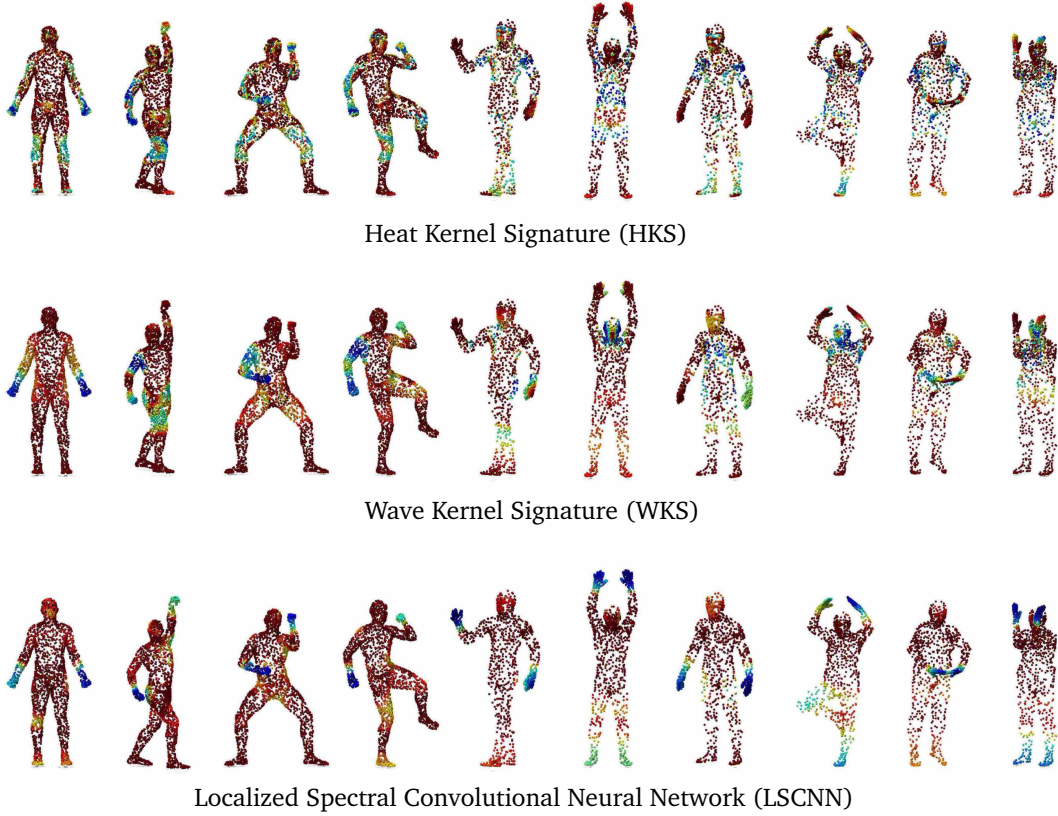
To better assess the quality of the descriptors produced by LSCNN, Figures 7.4 and 7.5 show a qualitative evaluation of the descriptor robustness in terms of the similarity map (Section 7.2.1) on meshes and point clouds, respectively. Point clouds were obtained by sampling FAUST meshes using the FPS algorithm [HS85]. The WFT on point clouds is computed using the graph Laplacian. Our approach shows a good trade-off between localization (similar to HKS) and accuracy (less spurious minima than WKS and OSD), as well as robustness to different kinds of noise.

Figures 7.6–7.9 show a quantitative evaluation of the descriptors quality according to the metrics presented in Section 7.2.1 for different configurations





*Figure 7.4.* Distance map in the descriptor space. The descriptor of a point on the knee of the leftmost shape (white dot) is compared to the descriptors of all other points on the same and on other shapes (which have undergone various transformations of increasing complexity). Shown left-to-right: reference shape from FAUST dataset, different pose of the same shape, different subject in the same dataset, two shapes from SCAPE dataset, Gaussian noise, heavy subsampling, voxelization noise, topological noise (glued fingers and missing parts). Small distances in the descriptor space correspond to cold colors, large distances to hot colors. An ideal descriptor should produce a distance map with small values localized around the knee and large values elsewhere.



*Figure 7.5.* Distance map in the descriptor space (similar to the one presented in Figure 7.4) for point clouds descriptors. In this case, the reference point is on the left hand of the leftmost shape and its descriptor is compared to all other points on the same shape and on other shapes from different datasets (the first four from SCAPE and the rest from FAUST datasets). Small distances in the descriptor space correspond to cold colors, large distances to hot colors. An ideal descriptor should produce a distance map with small values localized around the left hand and large values elsewhere.

of the training and testing sets. We observe that LSCNN1 and LSCNN2 perform comparably (slightly better) to GCNN1 and significantly outperform all other approaches. Interestingly, FC1 performs better than OSD, thanks to the non-linearity of the ReLU activation function.

## 7.2.4 GCNN experiments and results

**Architectures** As input and output settings we considered the same ones of LSCNN experiments 7.2.3.

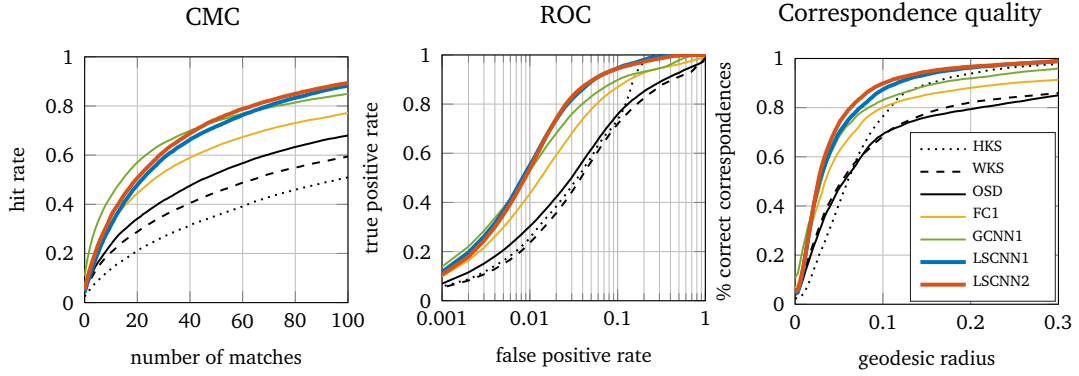


Figure 7.6. Performance of descriptors trained on a subset of FAUST dataset and tested on a disjoint subset of FAUST dataset.

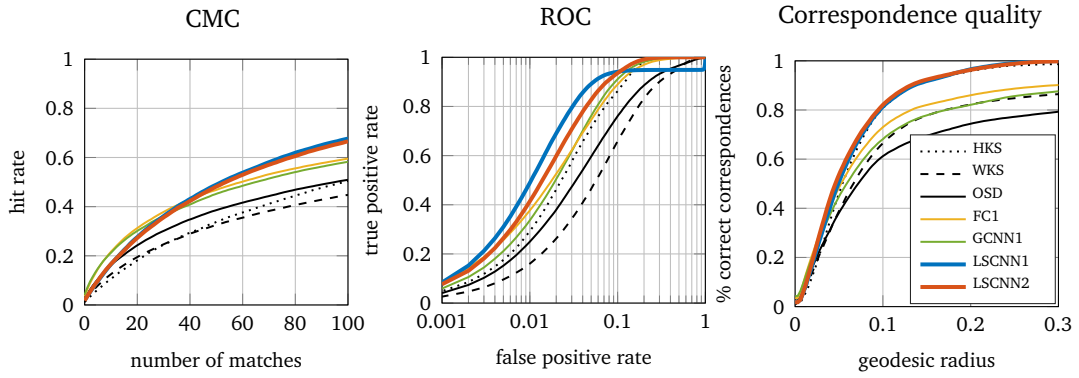


Figure 7.7. Performance of descriptors trained on a subset of FAUST dataset and tested on SCAPE dataset.

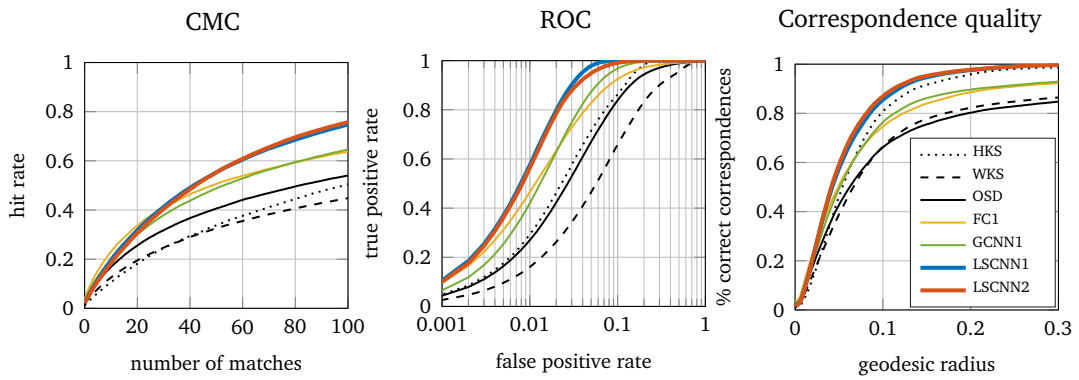


Figure 7.8. Performance of descriptors trained on a subset of SCAPE dataset and tested on a disjoint subset of SCAPE dataset.

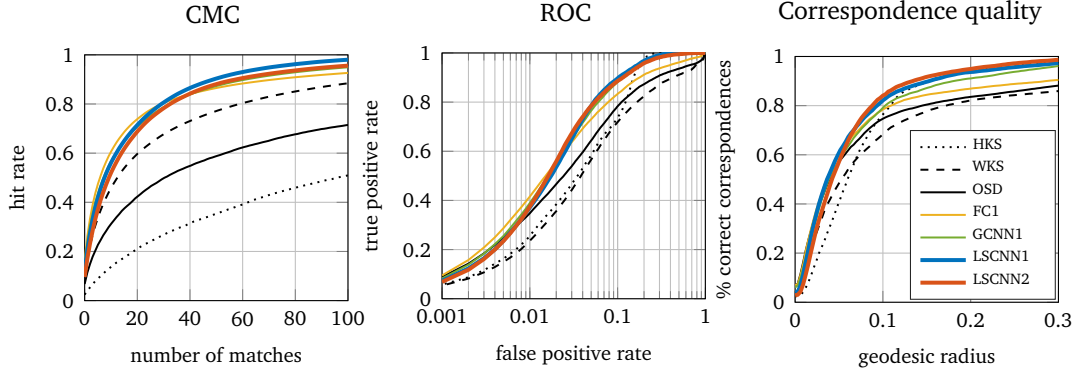


Figure 7.9. Performance of descriptors trained on a subset of SCAPE dataset and tested on FAUST dataset.

Two different configurations of the GCNN model were considered. The first one, denoted with GCNN1 (150-dim input, FC16, GC16, AMP), consists of a fully connected layer performing a dimensionality reduction of the  $m = 150$ -dimensional geometry vectors in input to obtain  $p = 16$ -dimensional features in output, followed by a geodesic convolutional layer with a bank of  $q = 16$  filters and an angular max pooling (Figure 7.2). The second one, GCNN2 (150-dim input, FC16, GC16, AMP, FC16) further applies another fully connected layer at the end of the chain.

**Results** Figures 7.10 and 7.11 show a qualitative evaluation of the descriptor robustness in terms of the similarity map (Section 7.2.1) of two different reference points. GCNN descriptors manifest both good localization (better than HKS) and are more discriminative (less spurious minima than WKS and OSD), as well as robust to different kinds of noise, including isometric and non-isometric deformations, geometric and topological noise, different sampling, and missing parts.

Figure 7.12 shows a quantitative evaluation in terms of the metrics presented in Section 7.2.1. We observe that GCNN descriptors significantly outperform other descriptors, and that the more complex model (GCNN2) further boosts performance.

In order to evaluate the generalization capability of the GCNN model, in Figure 7.13 the descriptors learned on FAUST dataset were tested on TOSCA shapes. We see that the learned model transfers well to a new dataset.

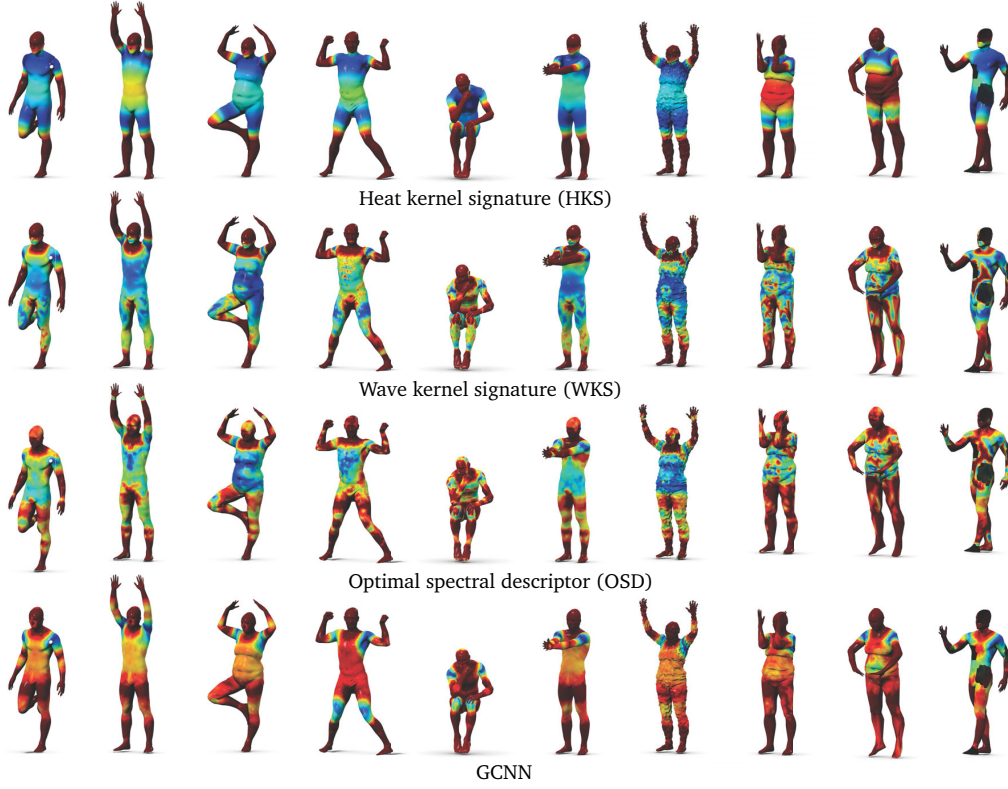


Figure 7.10. Normalized Euclidean distance between the descriptor at a reference point on the shoulder (white sphere) and the descriptors corresponding to the other points for different transformations (shown left-to-right: near isometric deformations, non-isometric deformations, topological noise, geometric noise, uniform/non-uniform subsampling, missing parts). Cold and hot colors represent small and large distances, respectively; distances are saturated at the median value. Ideal descriptors would produce a distance map with a sharp minimum at the shoulder and no spurious local minima at other locations.

### 7.2.5 ADD experiments and results

**Architectures** The ADD framework is a (non-convolutional) deep model which takes in input the anisotropic eigenvectors and eigenvalues, as described in Equation (6.13). In the case where only a single anisotropy value is considered, we use  $\alpha = 50$ , while for the case where multiple anisotropies are considered we use the values  $\alpha = 0, 25$ , and  $50$ .

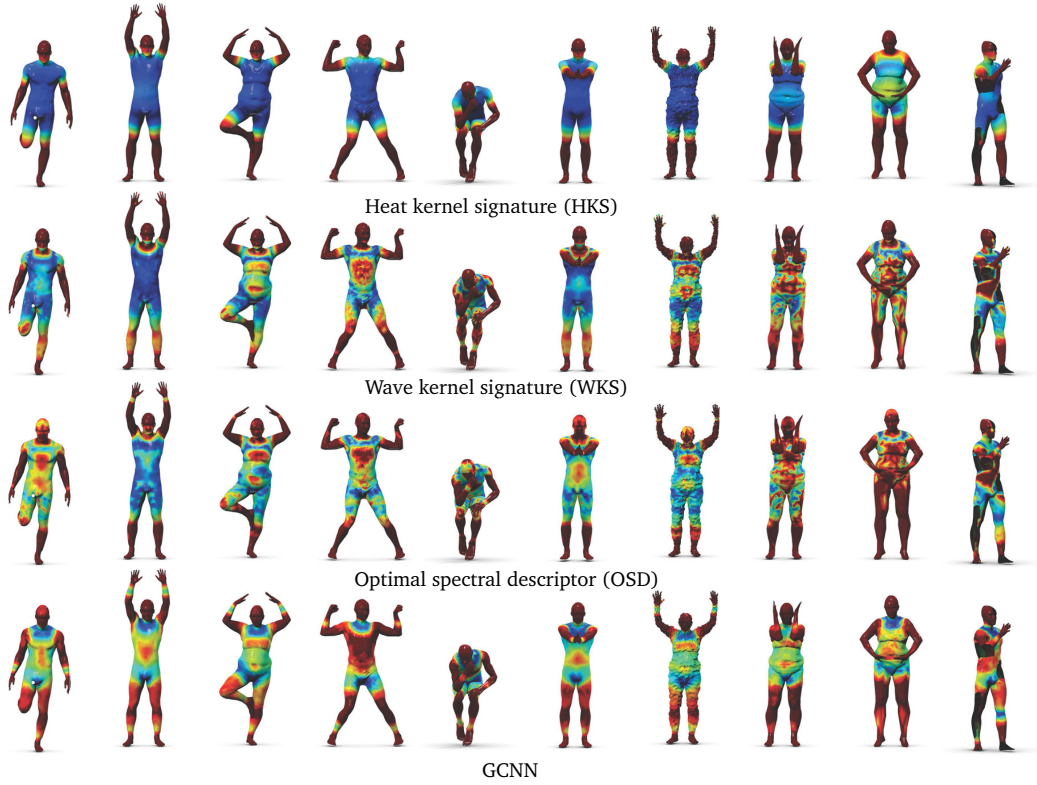


Figure 7.11. Same qualitative evaluation of Figure 7.10 for a reference point on the groin of the leftmost shape (indicated with a white sphere).

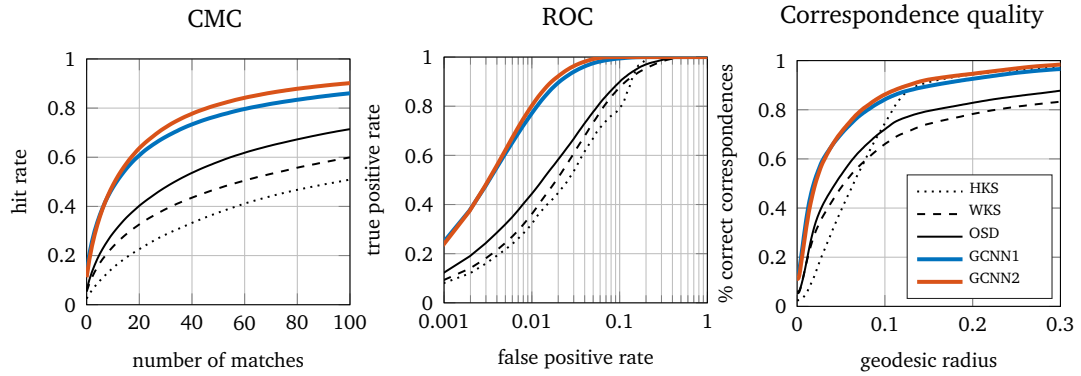


Figure 7.12. Performance of descriptors trained on a subset of FAUST dataset and tested on a disjoint subset of FAUST dataset.

For the single anisotropy case, we considered different configurations corresponding to increasingly complex architectures:

- ADD1 (SC + FC16) is a single layer model corresponding to a stack channel



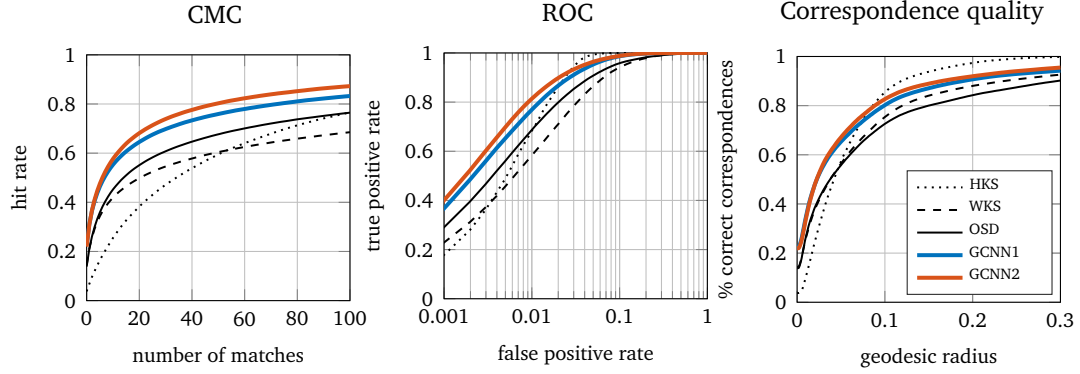


Figure 7.13. Performance of descriptors trained on FAUST dataset and tested on TOSCA dataset.

SC which just stacks the input channels ordered by angle and anisotropy values followed by a fully convolutional layer producing  $q = 18$ -dimensional features,

- ADD2 (ASF256 + ReLU + SC + FC16) is a two-layer architecture, that applies an anisotropic spectral filtering layer with a bank of 256 filters and a ReLU non-linearity before ADD1,
- ADD3 (ASF256 + ReLU + ASF256 + ReLU + SC + FC512 + ReLU + FC16) is a four-layer architecture representing a more complex model.

Architectures with multiple anisotropies in input are denoted by mADD1, mADD2 and mADD3, respectively.

**Results** Figure 7.14 shows a quantitative evaluation of the performance of the descriptors produced with the different architectures presented above in terms of the metrics presented in Section 7.2.1. The descriptors performance is measured according to two different settings: the *symmetric* setting (solid lines) considers symmetric points as correct matches, while the *asymmetric* setting (dashed lines) considers symmetric points as incorrect matches.

As expected, deep models achieve better performance than shallow ones and models with multiple anisotropies performs better than the ones with single anisotropy. Overall, the model mADD3 achieve the best performance.

Figure 7.15 shows a qualitative evaluation of the anisotropic spectral descriptors constructed with the mADD3 architecture in terms of a similarity map from a point on the right shoulder of the leftmost shape to other points of the same

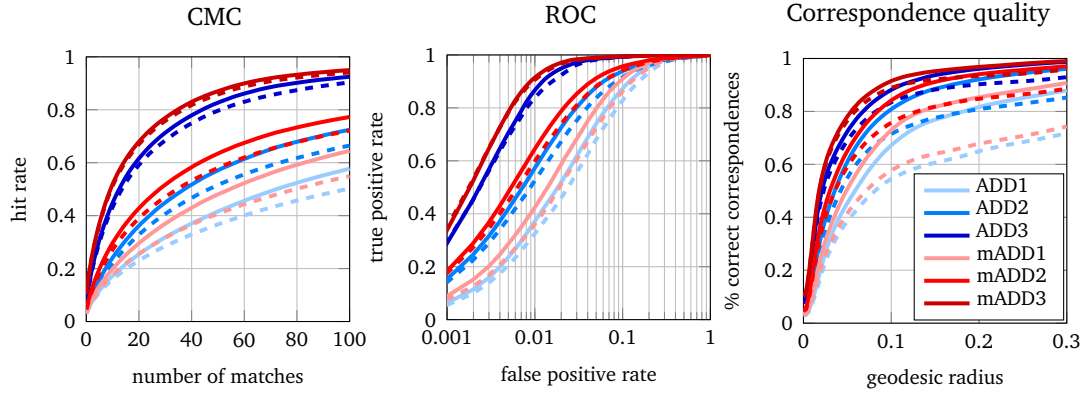


Figure 7.14. Performance of different architectures of our anisotropic descriptor according to symmetric (solid) and asymmetric (dashed) settings. Training and testing were done on disjoint sets of the FAUST dataset.

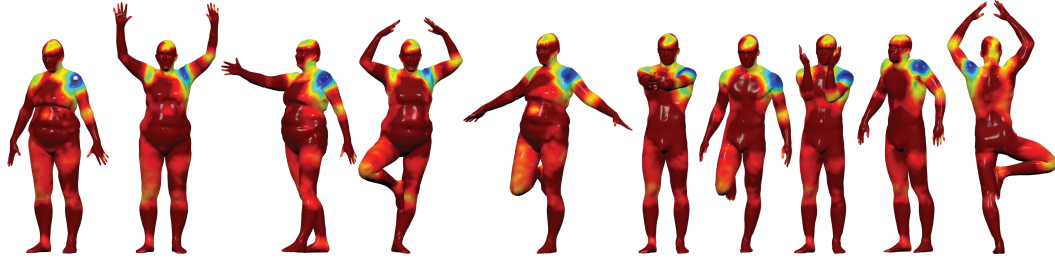


Figure 7.15. Qualitative evaluation of our mADD3 descriptors on meshes. Shown is the normalized Euclidean distance between the descriptor at a reference point on the shoulder (white point) and the descriptors corresponding to the other points for different transformations. Cold and hot colors represent small and large distances, respectively; distances are saturated at the median value. Ideal descriptors would produce a distance map with a sharp minimum at the right shoulder and no spurious local minima at other locations.

shape as well as other shapes from the FAUST dataset. Descriptors learned with ADD manifest good localization and specificity, and, remarkably, they are not ambiguous to symmetry contrary to spectral descriptors, LSCNN, and GCNN constructions (compare with Figures 7.4 and 7.10).

Finally, Figure 7.16 shows a quantitative evaluation of the descriptors produced with ADD3 and mADD3 architectures and compares them with hand-crafted spectral descriptors, such as HKS and WKS, as well as with learnable ones, such as OSD, LSCNN, and GCNN. Anisotropic constructions (ADD3 and mADD3) outperform all the other descriptors and manifest significantly smaller drop in



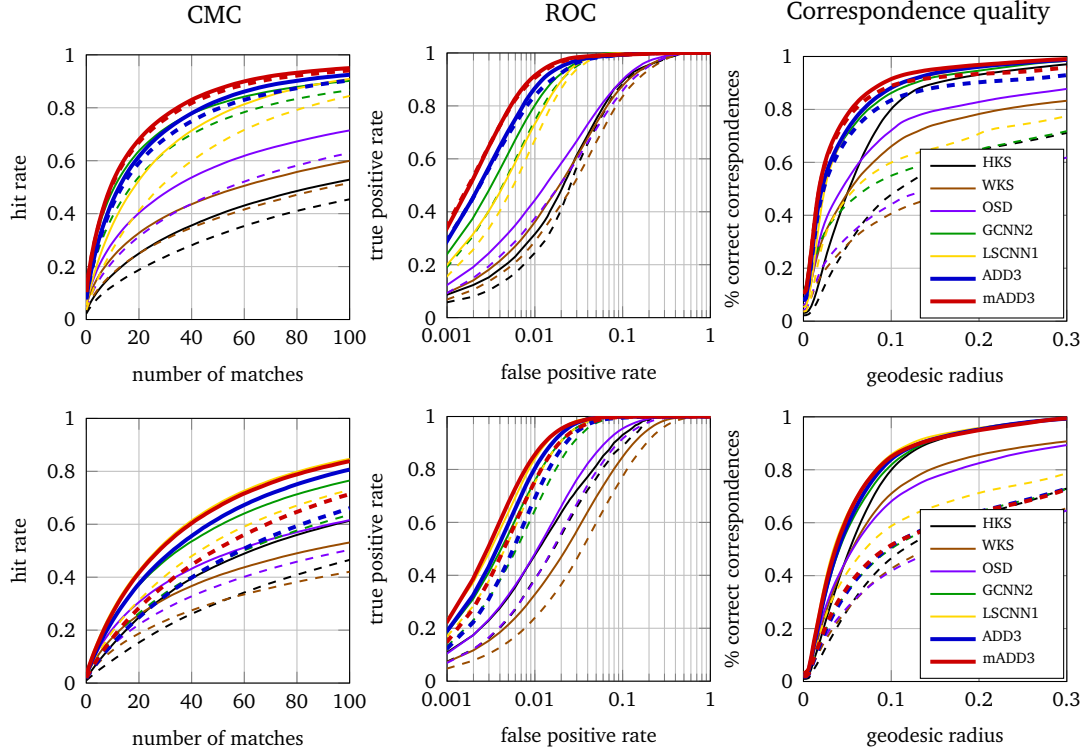


Figure 7.16. Performance of different descriptors measured using CMC (left), ROC (center), and Princeton protocol for nearest-neighbor correspondence (right); higher curves correspond to better performance. Symmetric (solid) and asymmetric (dashed) settings are shown. Learnable descriptors were trained and tested on disjoint sets of the FAUST (top) and SCAPE (bottom) datasets, respectively. All descriptors in these plots are 16-dimensional.

performance when switching from the easier symmetric evaluation (solid line) to the harder asymmetric one (dashed lines).

### 7.3 Learning shape retrieval

Given a query shape  $\mathcal{X}$ , shape retrieval deals with the problem of finding the target shape  $\mathcal{Y}$  most similar to  $\mathcal{X}$  among the samples available in the given dataset. Usually, rather than directly comparing the shapes, it is preferable to compare features encoding the global structure of the shape, called *global descriptors*.

Intrinsic convolutional networks allow to learn global descriptors simply by concatenating an additional layer computing an aggregation of the local features

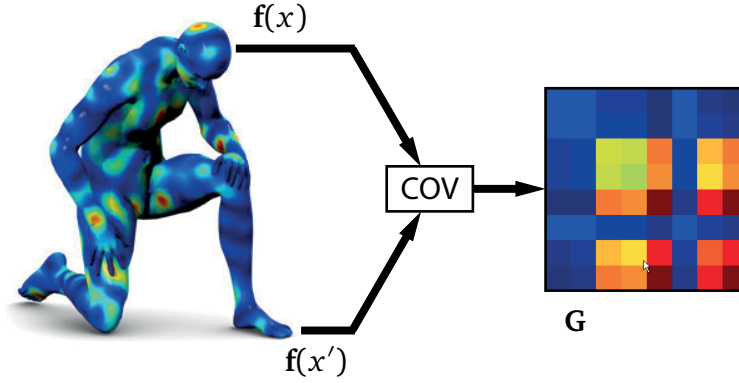


Figure 7.17. The covariance layer COV combines the  $p$ -dimensional local descriptors at each vertex  $x \in \mathcal{X}$  of the shape on the left in a  $p \times p$  matrix representing a global descriptor for the shape  $\mathcal{X}$ .

produced by the network (7.1) presented in Section 7.1.

In the paper [MBBV15], inspired by the work [TPM06], we propose to consider as feature aggregation layer the *covariance layer*, defined as

$$\mathbf{G} = \int_{\mathcal{X}} (\mathbf{f}(x) - \boldsymbol{\mu})(\mathbf{f}(x) - \boldsymbol{\mu})^{\top} dx,$$

where  $\mathbf{f}(x) = (f_1(x), \dots, f_p(x))^{\top}$  is a  $p$ -dimensional input vector,  $\boldsymbol{\mu} = \int_{\mathcal{X}} \mathbf{f}(x) dx$ , and  $\mathbf{G}$  is a  $p \times p$  matrix, which can be regarded as a global descriptor for the shape  $\mathcal{X}$  (Figure 7.17).

Training is done by minimizing the siamese loss (7.2), where positives and negatives are shapes from same and different classes, respectively.

### 7.3.1 GCNN experiments and results

As input we considered  $m = 16$ -dimensional HKS descriptors. We used a GCNN architecture composed of a fully connected layer FC8 acting as dimensionality reduction and producing  $p = 8$ -dimensional features, followed by a geodesic convolutional layer GC8 with a bank of  $q = 8$  filters and angular max pooling layer AMP (6.6). Finally a covariance layer COV produces a  $q \times q$  output matrix used as the global shape descriptor.

As dataset we considered FAUST. In particular, the training set consisted of five poses per subject (a total of 50 shapes), while testing was performed on the 50 remaining shapes in a leave-one-out fashion.

Evaluation was done in terms of precision (percentage of retrieved shapes matching the query class) and recall (percentage of shapes from the query class

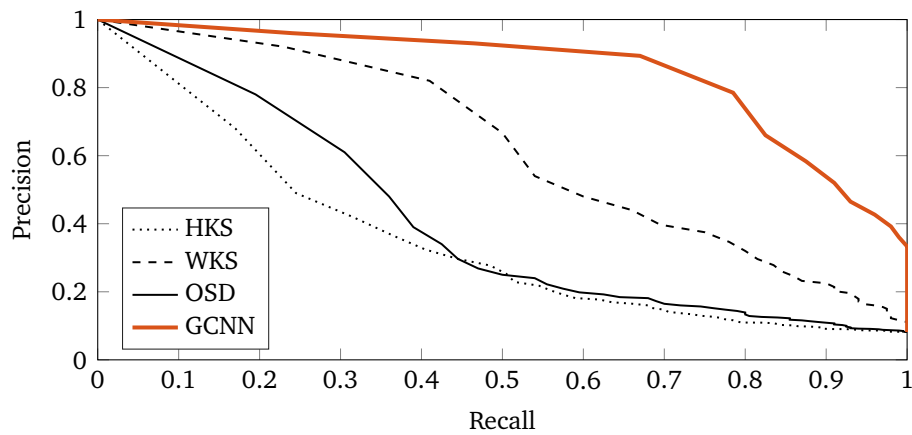


Figure 7.18. Performance (in terms of Precision-Recall) of shape retrieval on the FAUST dataset using different descriptors. Higher curve corresponds to better performance.

that is retrieved). Figure 7.18 shows the precision-recall curve. For comparison, we show the performance of other descriptors (HKS, WKS, and OSD) aggregated into a global covariance shape descriptor. GCNN outperforms significantly all other methods.



## Chapter 8

# Learning shape correspondence with intrinsic deep learning

In this chapter, we provide an experimental evaluation of the performance of intrinsic CNN methods in the problem of learning *shape correspondence* across shapes from challenging datasets. In particular, we will provide the results achieved by our methods [MBBV15; BMR<sup>+</sup>16; BMRB16; MBM<sup>+</sup>17].

### 8.1 Shape correspondence as a classification problem

Rodolà and colleagues, in their seminal work [RRBW<sup>+</sup>14], defined the problem of finding the correspondence in a collection of shapes as a classification problem, where one tries to assign each vertex of a *query* shape  $\mathcal{X}$  to one of the vertices of some common *reference* shape  $\mathcal{Y}$  (representing a *label space*).

In the papers [MBBV15; BMR<sup>+</sup>16; BMRB16; MBM<sup>+</sup>17], we showed that this can be achieved by adding a *softmax* layer at the end of the intrinsic CNN architecture presented in Equation (7.1). A softmax layer applies the softmax function

$$g_i(x) = \frac{e^{f_i(x)}}{\sum_{i=1}^n e^{f_i(x)}}$$

point-wise to the  $n$ -dimensional input  $\mathbf{f}(x) = (f_1(x), \dots, f_n(x))$  and gives as output a  $n$ -dimensional function  $\mathbf{g}(x) = (g_1(x), \dots, g_n(x))$ , which can be interpreted as a probability distribution over the  $n$  classes.

Let  $n$  and  $m$  denote the number of vertices in  $\mathcal{X}$  and  $\mathcal{Y}$ , respectively. For a point  $x$  on a query shape, the output of an intrinsic CNN  $U_{\Theta}$  is an  $m$ -dimensional vector

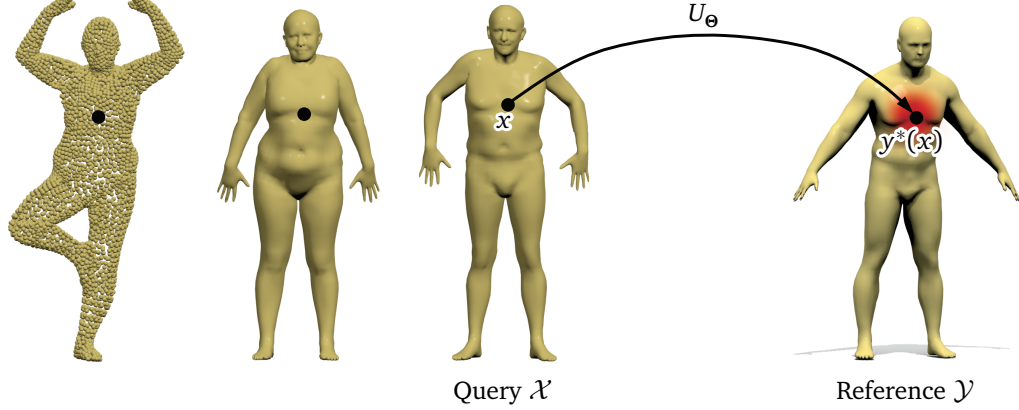


Figure 8.1. Learning shape correspondence: an intrinsic deep network  $U_{\theta}$  is applied point-wise to some input features defined at each point. The output of the network at each point  $x$  of the query shape  $\mathcal{X}$  is a probability distribution over the reference shape  $\mathcal{Y}$  that can be thought of as a soft correspondence.

and can be interpreted as a probability distribution (soft correspondence) on the vertices of the reference shape  $\mathcal{Y}$  (the label space). This situation is depicted in Figure 8.1, where red colors correspond to the values of the probability distribution over  $\mathcal{Y}$  produced by the intrinsic convolutional neural network  $U_{\theta}$  applied to the point  $x \in \mathcal{X}$ .

**Training set** Let us denote by  $y^*(x)$  the ground-truth correspondence of  $x$  on the reference shape. We assume to be provided examples of points from shapes across the collection and their ground-truth correspondence,  $\mathcal{T} = \{(x, y^*(x))\}$ . Some public-domain datasets, such as SCAPE [ASK<sup>+</sup>05] and FAUST [BRLB14], naturally provide such information.

**Cost function** The more the probability distribution associated to the point  $x \in \mathcal{X}$  is concentrated around  $y^*(x) \in \mathcal{Y}$ , the better the correspondence quality is. A perfect correspondence for the point  $x \in \mathcal{X}$  coincides with the Dirac's delta  $\delta_{y^*(x)}$ .

Such behaviour can be obtained by minimizing over the training set the *multinomial regression loss*

$$\mathcal{L}(\theta) = - \sum_{(x, y^*(x)) \in \mathcal{T}} \log U_{\theta}(f(x))(y^*(x)),$$

w.r.t. the network parameters  $\Theta$ . Here  $U_\Theta(f(x))(y)$  denotes the conditional probability  $p(y|x)$  of  $x$  being mapped to  $y$ .

The multinomial regression loss can be interpreted as the Kullback-Leibler divergence between the probability distribution produced by the network  $U_\Theta$  and the ground-truth distribution  $\delta_{y^*(x)}$ .

### 8.1.1 Correspondence refinement

The most straightforward way to convert the soft correspondence  $U_\Theta(f(x))(y)$  produced by intrinsic CNNs into a pointwise correspondence is by assigning  $x$  to

$$\hat{y}(x) = \operatorname{argmax}_{y \in \mathcal{Y}} U_\Theta(f(x))(y). \quad (8.1)$$

The value

$$c(x) = \max_{y \in \mathcal{Y}} U_\Theta(f(x))(y) \in [0, 1]$$

can be interpreted as the confidence of the prediction: the closer the distribution produced by the network is to a delta-function (in which case  $c = 1$ ), the better it is.

**Functional map refinement** By exploiting the functional map framework [OBCS<sup>+</sup>12] presented in Section (2.8.1), a slightly more elaborate scheme to refine the soft correspondences produced by the intrinsic CNN can be devised.

First, we select a subset of points  $P = \{x : c(x) > \tau\}$  at which the confidence of the predicted correspondence exceeds some threshold  $\tau$ .

Second, we use this subset of corresponding points to find a functional map between  $L^2(\mathcal{X})$  and  $L^2(\mathcal{Y})$  by solving the linear system of  $|P|k$  equations in  $k^2$  variables,

$$\Phi_P \mathbf{C} = \Psi_P, \quad (8.2)$$

where

$$\begin{aligned} \Phi_P &= (\phi_1(x), \dots, \phi_k(x)), \quad x \in P, \\ \Psi_P &= (\psi_1(\hat{y}(x)), \dots, \psi_k(\hat{y}(x))), \quad x \in P, \end{aligned}$$

are the first  $k$  Laplacian eigenfunctions of shapes  $\mathcal{X}$  and  $\mathcal{Y}$ , respectively, sampled at the subset of corresponding points (represented as  $|P|k$  matrices). The  $k \times k$  matrix  $\mathbf{C}$  represents the functional correspondence between  $L^2(\mathcal{X})$  and  $L^2(\mathcal{Y})$  in the frequency domain. The parameters  $\tau$  and  $k$  must be chosen in such a way that the system is over-determined, i.e.  $|P| > k$ .

Third, after having found  $\mathbf{C}^*$  by solving (8.2) in the least-squares sense, we produce a new point-wise correspondence by matching  $\Phi\mathbf{C}^*$  and  $\Psi$  in the  $k$ -dimensional eigenspace

$$y(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \|(\phi_1(x), \dots, \phi_k(x))\mathbf{C}^* - (\psi_1(\hat{y}(x)), \dots, \psi_k(\hat{y}(x)))\|_2.$$

**Bayesian refinement** The predicted correspondence can be refined by using the Bayesian filtering approach proposed in [VLR<sup>+</sup>17].

Since this method assumes the two shapes to be full models, when dealing with range scans we follow a different refinement process, described in the following. Let  $t: \mathcal{X} \rightarrow \mathcal{Y}$  be the map obtained by assigning to each point  $x \in \mathcal{X}$  the point  $\operatorname{argmax}_y T(x, y)$ ; in other words,  $t$  is a maximum likelihood realization of the soft correspondence  $T$ . Since the correct map is expected to be as smooth as possible, the idea is to identify the points  $x \in \mathcal{X}$  where  $t$  is discontinuous and to replace their image with better matches.

We detect the discontinuities by measuring the smoothness of the coordinate functions  $f_{\{x,y,z\}}$  of  $\mathcal{Y}$  (i.e., the xyz-coordinates of the points of  $\mathcal{Y}$  seen as three separate scalar functions on  $\mathcal{Y}$ ) when these are pulled back to  $\mathcal{X}$  via  $t$ . In particular, if  $t$  is smooth, then we expect the gradient norms  $\|\nabla_{\mathcal{X}}(f \circ t)\|$  to be approximately constant across the surface  $\mathcal{X}$  (here,  $\nabla_{\mathcal{X}}$  is the intrinsic gradient operator on  $\mathcal{X}$ ). Therefore, the points of discontinuity of  $t$  are detected as the outlier points  $\hat{x} \in \mathcal{X}$  where this norm deviates significantly.

For each detected outlier  $\hat{x}$ , the corresponding match  $t(\hat{x})$  is replaced by means of an iterative procedure alternating between two steps:

- *Non-local step.* The map  $t$  is converted to a *partial functional map* [RCB<sup>+</sup>17] in the Fourier basis, low-pass filtered to the first  $k = 20$  harmonics, and converted back to a point-wise map. This acts as a *non-local* smoothing of the entire correspondence, but only the point  $t(\hat{x})$  is actually updated.
- *Local step.* We compute a minimizer of

$$\min_{y^* \in \mathcal{Y}} \left| \sum_{y \in t(B_\epsilon(\hat{x}))} d_{\mathcal{Y}}(y, y^*) - \sum_{x \in B_\epsilon(\hat{x})} d_{\mathcal{X}}(x, \hat{x}) \right|, \quad (8.3)$$

where  $d_{\mathcal{X}}, d_{\mathcal{Y}}$  are the geodesic distance functions on  $\mathcal{X}, \mathcal{Y}$  and  $B_\epsilon(\hat{x})$  is the geodesic open ball of radius  $\epsilon$  around  $\hat{x}$ . The minimizer to the problem above is a weighted geometric median of the image, under  $t$ , of the local neighborhood of  $\hat{x}$ . This step can thus be seen as a form of median filtering of the correspondence.



The two steps are iterated until convergence to a stationary state, or until no more outliers are detected.

### 8.1.2 Performance evaluation

The correspondence performance was evaluated using the Princeton protocol [KLF11], plotting the percentage of matches that are at most  $r$ -geodesically distant from the ground-truth correspondence on the reference shape, as a function of the parameter  $r$ . Two versions of the protocol consider intrinsically symmetric matches as correct (symmetric setting) or wrong (asymmetric, more challenging setting). Some methods based on intrinsic structures (e.g. LSCNN or RF applied on WKS descriptors) are invariant under intrinsic symmetries and thus cannot distinguish between symmetric points.

### 8.1.3 Datasets

To better assess the quality of the correspondence learned by the proposed approaches we considered different datasets corresponding to different challenges: *full meshes*, *partial meshes*, and *range scans*.

Correspondences between full meshes are tested on the FAUST dataset, described in Section 7.2.2. The first 80 shapes are used for training, while the remaining 20 are used for testing.

Correspondences between partial meshes are tested on the recent very challenging SHREC'16 Partial Correspondence benchmark [CRB<sup>+</sup>16], consisting of nearly-isometrically deformed shapes from eight classes, with different parts removed. Two types of partiality in the benchmark are *cuts* (removal of a few large parts) and *holes* (removal of many small parts). In each class, the vertex-wise groundtruth correspondence between the full shape and its partial versions is given. The dataset was split into training and testing disjoint sets. For cuts, training was done on 15 shapes per class; for holes, training was done on 10 shapes per class.

Finally, range maps have been synthetically generated from FAUST meshes. For each subject and pose, we produced 10 rangemaps in  $100 \times 180$  resolution, covering shape rotations around the z-axis with increments of 36 degrees (total of 1000 range maps), keeping the groundtruth correspondence. Training and testing set splitting was done accordingly to the one on FAUST full meshes (subjects 1–8 are used for training, subjects 9–10 for testing).

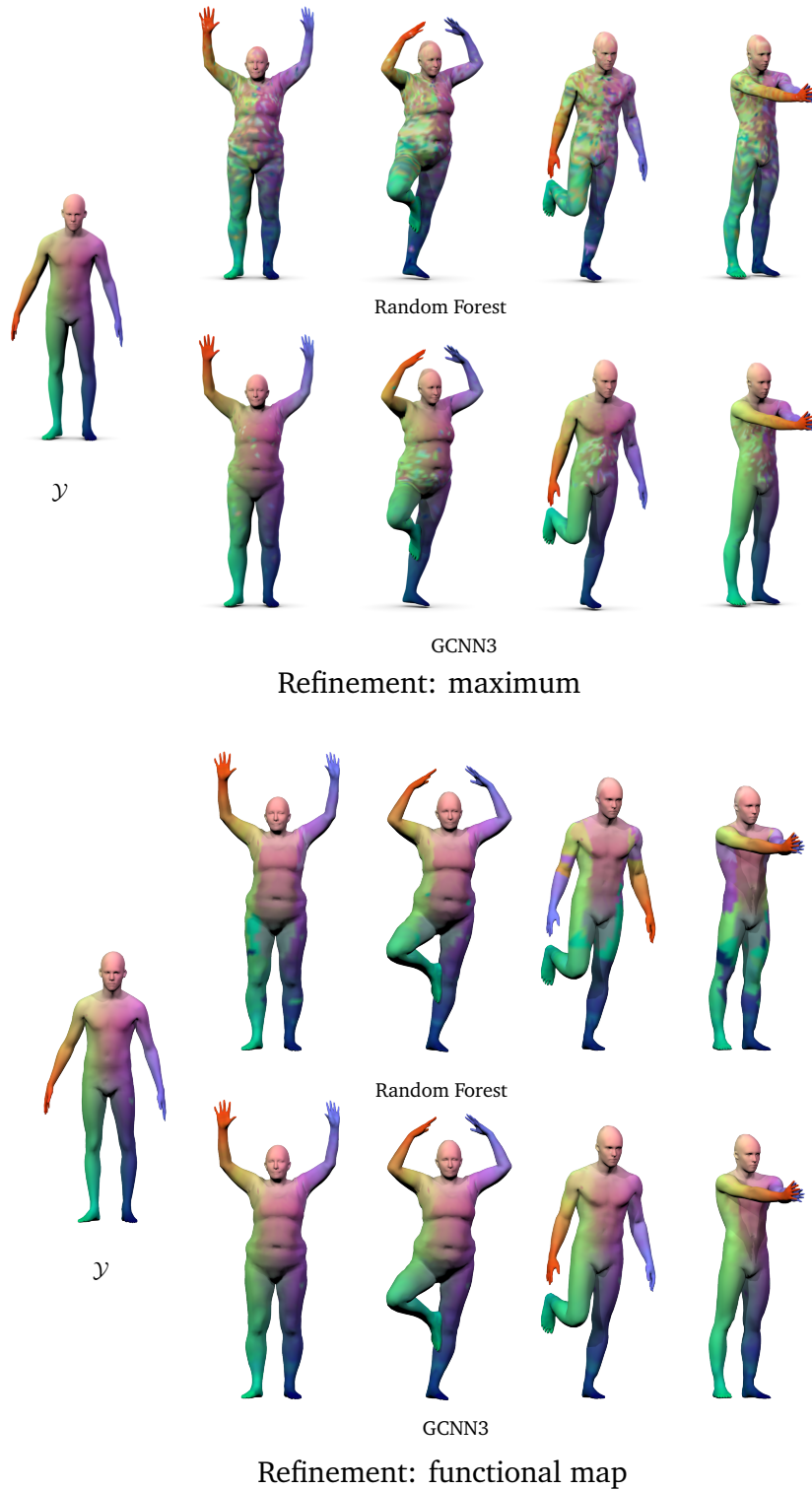


Figure 8.2. Example of correspondence obtained with GCNN [MBBV15] and Random forest [RRBW<sup>+</sup>14]. Similar colors encode corresponding points. *Top*: raw correspondences obtained by the intrinsic CNN. *Bottom*: raw correspondences are refined by the functional map refinement.

## 8.2 GCNN experiments and results

**Architecture** As input data, we considered  $m = 150$ -dimensional geometry vectors (2.19).

As GCNN model we considered an architecture GCNN3 containing three convolutional layers (FC16, GC32 + AMP, GC64 + AMP, GC128 + AMP, FC256, FC6890). The zeroth FAUST shape containing  $n = 6890$  vertices was used as reference shape  $\mathcal{V}$ .

For each point on the query shape, the output of GCNN representing the soft correspondence as a 6890-dimensional vector was converted into a point correspondence by taking the maximum or by employing the previously described functional map refinement.

**Results** Figure 8.2 shows a qualitative evaluation of the correspondence obtained with GCNN3 and the random forests model developed by Rodolà et al. [RRBW<sup>+</sup>14]. Colors are transferred using raw point-wise correspondence (top) and correspondences were refined using the functional maps algorithm (bottom). GCNN3 shows significantly better performance than random forests [RRBW<sup>+</sup>14].

Figure 8.3 shows a quantitative evaluation of the correspondences produced by GCNN3 in terms of the Princeton benchmark [KLF11] presented in Section 8.1.2. GCNN3 outperforms the hand-crafted methods Blended maps [KLF11] and Functional map [OBCS<sup>+</sup>12] and the learning method Random forests [RRBW<sup>+</sup>14].

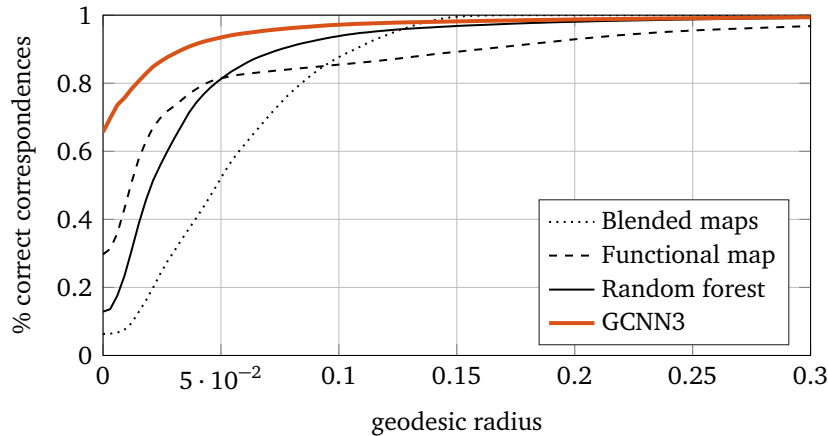


Figure 8.3. Performance of shape correspondence on the FAUST dataset evaluated using the Princeton benchmark. Higher curve corresponds to better performance.

### 8.3 ADD experiments and results

**Architecture** For the correspondence experiment, we used a 5-layer architecture (ASF512 + ReLU + ASF512 + ReLU + SC + FC2048 + ReLU + FC2048 + ReLU + FC6890 + Softmax) with multiple anisotropies, denoted with mADD4.

**Results** Figures 8.4 and 8.5 (left) quantify the quality of the correspondence learned with our method on FAUST meshes and point clouds. For comparison, we show the performance of blended intrinsic maps (BIM) [KLF11], functional maps (FM) [OBCS<sup>+</sup>12], and random forest (RF) [RRBW<sup>+</sup>14]. Note that blended maps use orientation information and thus can distinguish bilaterally symmetric points (therefore, the dashed and solid black curves in Figure 8.4 coincide). On the other hand, random forests in [RRBW<sup>+</sup>14] were learned on WKS input, which is ambiguous with respect to symmetry; this explains the significant drop when passing from the symmetric to the asymmetric evaluation setting.

Figures 8.4 and 8.5 (right) depict the quality of the obtained correspondence. To visualize the colors transferred from the reference shape to the query shapes, we use the raw point-wise correspondence produced by our method as an input to the functional maps algorithm.

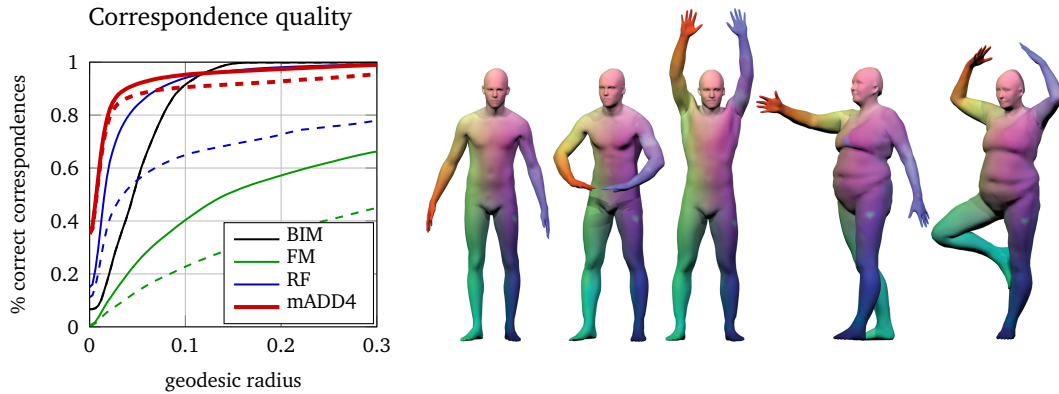


Figure 8.4. Correspondence on FAUST meshes. Left: evaluation of the correspondence using the symmetric (solid) and asymmetric (dashed) Princeton protocol. Right: example of correspondence obtained using our method (similar colors encode corresponding points, where the leftmost shape is the reference).

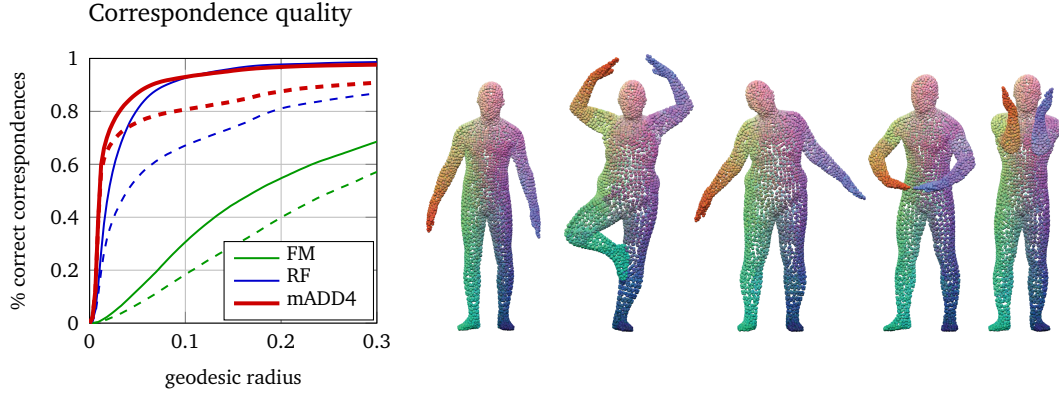


Figure 8.5. Correspondence on FAUST point clouds. *Left*: evaluation of the correspondence using the symmetric (solid) and asymmetric (dashed) Princeton protocol. *Right*: example of correspondence obtained using our method (similar colors encode corresponding points, where the leftmost shape is the reference).

## 8.4 ACNN experiments and results

**Architecture** As input data we consider  $m = 544$ -dimensional SHOT descriptors (local histogram of normal vectors) [STDS14]. For learning correspondences between full meshes, we considered the following ACNN architecture: FC64 + AC64 + AC128 + AC256 + FC1024 + FC512 + Softmax, where AC denoted the anisotropic convolutional layer. For the more challenging task of learning correspondences between partial meshes, we considered instead the architecture: AC32 + FC1024 + DO(0.5) + FC2048 + DO(0.5) + Softmax, where DO denotes a dropout layer [HSK<sup>+</sup>12].

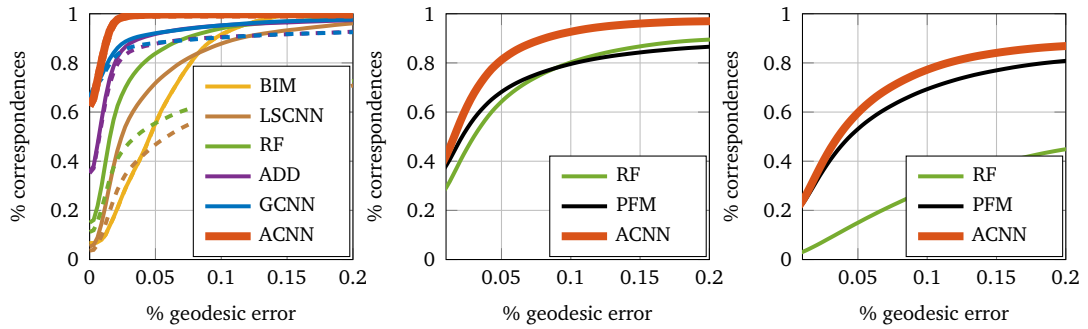
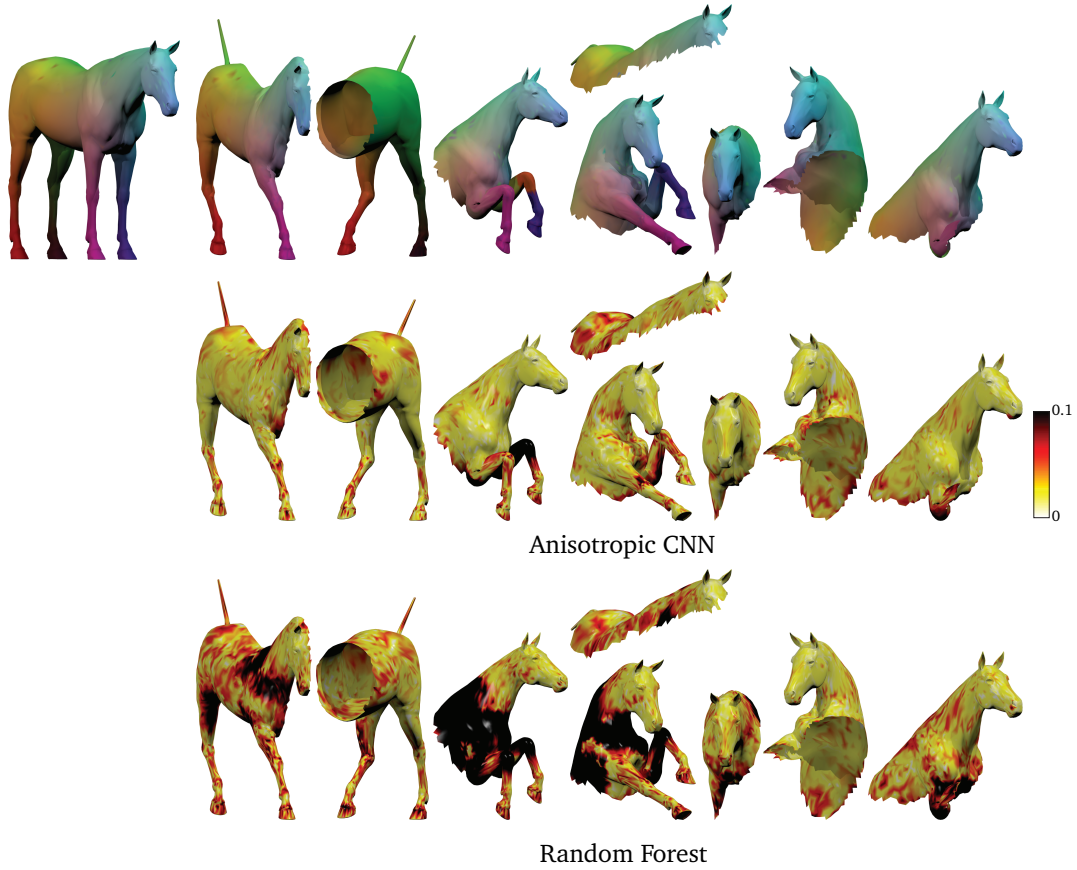


Figure 8.6. Performance of different correspondence methods on different datasets. *Left*: FAUST, *center*: SHREC'16 Partial (cuts), *right*: SHREC'16 Partial (holes). Evaluation of the correspondence was done using the symmetric (solid) and asymmetric (dashed) Princeton protocol.

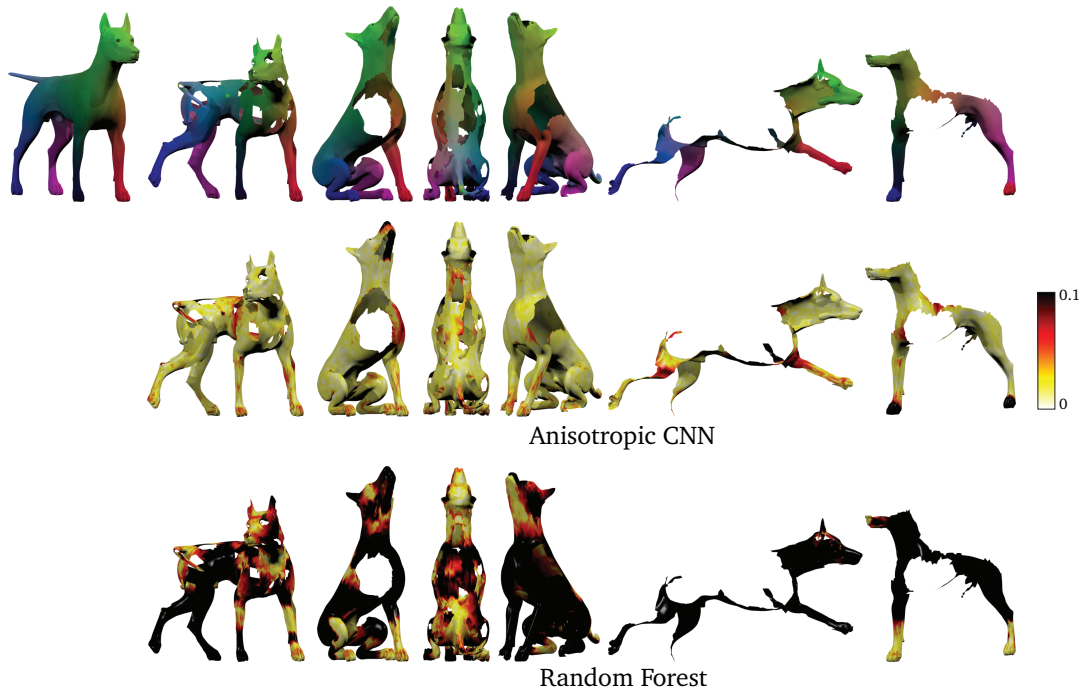
**Results** Figure 8.6 (left) shows a quantitative evaluation of the quality of the correspondence produced by the ACNN method and compares it to different other methods. The correspondence learned with the proposed ACNN clearly outperforms all the other approaches.



*Figure 8.7.* Examples of partial correspondence on the horse shape from the SHREC'16 Partial (cuts) dataset. First row: correspondence produced by ACNN. Corresponding points are shown in similar color. Reference shape is shown on the left. Second and third rows: pointwise geodesic error (in % of geodesic diameter) of the ACNN and RF correspondence, respectively. For visualization clarity, the error values are saturated at 10% of the geodesic diameter. Hot colors correspond to large errors.

Figure 8.6 (center) compares the performance of different partial matching methods on the SHREC'16 Partial (cuts) dataset. ACNN outperforms other approaches with a significant margin. Figure 8.7 (top) shows examples of partial correspondence on the horse shape as well as the pointwise geodesic error (bottom). We observe that the proposed approach produces high-quality correspondences even in such a challenging setting.

Figure 8.6 (right) compares the performance of different partial matching methods on the SHREC'16 Partial (holes) dataset. Also in this setting, ACNN outperforms other approaches with a significant margin. Figure 8.8 (top) shows examples of partial correspondence on the dog shape as well as the pointwise geodesic error (bottom).



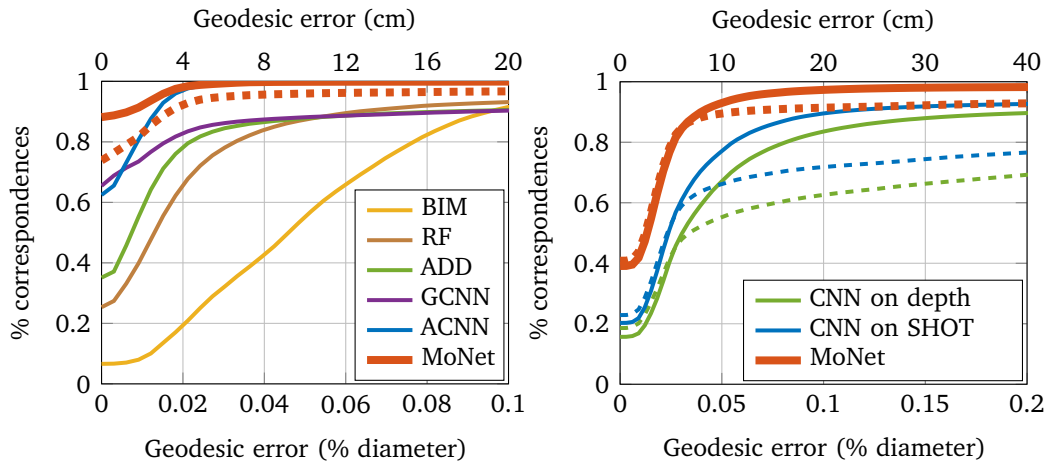
*Figure 8.8.* Examples of partial correspondence on the dog shape from the SHREC'16 Partial (holes) dataset. First row: correspondence produced by ACNN. Corresponding points are shown in similar color. Reference shape is shown on the left. Second and third rows: pointwise geodesic error (in % of geodesic diameter) of the ACNN and RF correspondence, respectively. For visualization clarity, the error values are saturated at 10% of the geodesic diameter. Hot colors correspond to large errors.



## 8.5 MoNet experiments and results

**Architecture** The quantitative and qualitative evaluation described above was repeated also for MoNet, by considering an architecture composed of 3 convolutional layers, similar to the one considered by the GCNN and ACNN methods. MoNet shows even further improvement w.r.t. ACNN and GCNN approaches and provide state-of-the-art results.

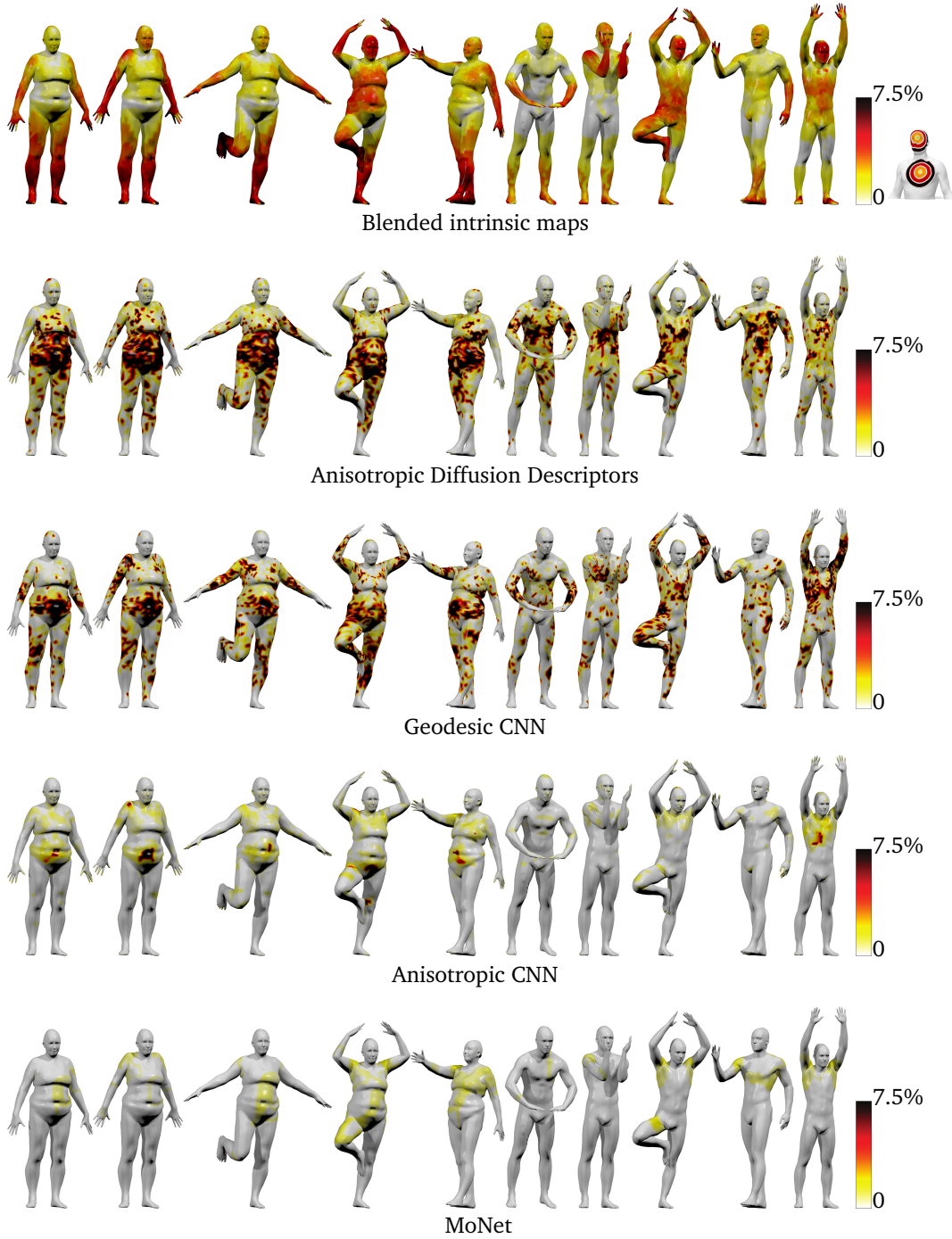
**Results** To evaluate the quality of the correspondences produced by MoNet, we did two experiments: on meshes and on range scans.



*Figure 8.9.* Shape correspondence quality obtained by different methods on the FAUST humans dataset (left) and on FAUST range scans (right). Dotted curves show raw performance, solid curves show performance after refinement. On meshes, MoNet results are compared with different other methods, including GCNN and ACNN. On range scans, we show the performance of a Euclidean CNN with a comparable 3-layer architecture.

Figure 8.9 (left) depicts the quantitative evaluation results for meshes, showing that MoNet significantly outperforms the competing approaches. In particular, close to 90% of points have *zero* error, and for 99% of the points the error is below 4cm. To get a better idea of the correspondence quality, Figure 8.10 shows the point-wise geodesic correspondence error of our method, and Figure 8.11 visualizes the obtained correspondence using texture transfer.



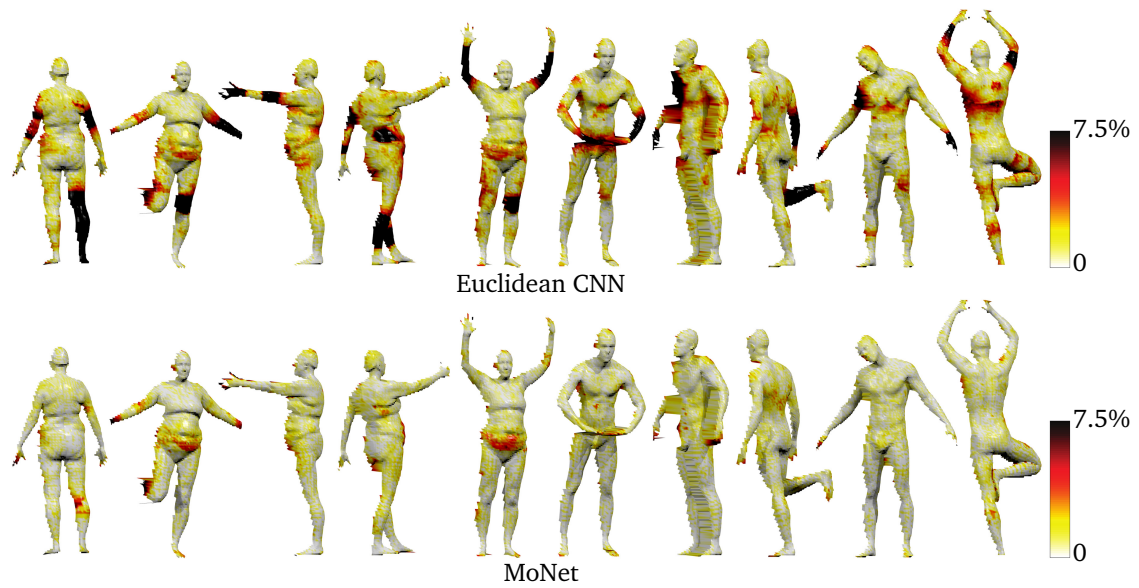


*Figure 8.10.* Pointwise error (geodesic distance from groundtruth) of different correspondence methods on the FAUST humans dataset. For visualization clarity, the error values are saturated at 7.5% of the geodesic diameter, which corresponds to approximately 15 cm. Hot colors represent large errors.



*Figure 8.11.* Examples of correspondence on the FAUST humans dataset obtained by the proposed MoNet method. What is shown is the texture transferred from the leftmost reference shape to different subjects in different poses by means of our correspondence.

Finally, we repeated the shape correspondence experiment on range scans synthetically generated from FAUST meshes. Figure 8.9 (right) shows the quality of correspondence between such range scans obtained by MoNet in terms of the Princeton protocol [KLF11]. For comparison, we show the performance of a standard Euclidean CNN with an equivalent architecture of three convolutional layers applied on raw depth values and on SHOT descriptors. Our approach clearly shows a superior performance. Figure 8.12 shows the relative point-wise geodesic correspondence error. Figure 8.13 shows a qualitative visualization of correspondence using similar color code for corresponding vertices. We also show correspondence on shapes from SCAPE [ASK<sup>+</sup>05] and TOSCA [BBK08] datasets.



*Figure 8.12.* Pointwise error (geodesic distance from groundtruth) of different methods on FAUST range maps. For visualization clarity, the error values are saturated at 7.5% of the geodesic diameter, which corresponds to approximately 15 cm. Hot colors represent large errors.



*Figure 8.13.* Visualization of correspondence on FAUST range maps as color code (corresponding points are shown in the same color). Full reference shape is shown on the left. Bottom row show examples of additional shapes from SCAPE and TOSCA datasets.



## Chapter 9

# Conclusions and future work

The incredible success of deep learning methods in computer vision and pattern recognition applications, with incredible performance on very challenging image analysis tasks, makes it very tempting for other domains and data as well.

In this thesis, we addressed the problem of providing such an extension for shape analysis applications, where data are represented as non-Euclidean domains in the form of 3D shapes.

We presented different methods allowing to provide such an extension and tested their performance on three basic problems in shape analysis: local shape descriptors, shape retrieval, and shape correspondence.

In particular, in Chapter 4 we presented SCNN [BZSL13], a spectral formulation of CNNs on non-Euclidean domains. While extremely important because it was the inspiration to our specific contribution in this field, we showed how its lack of generalizability makes it unsuitable for shape analysis tasks.

In Chapter 5, we proposed LSCNN [BMM<sup>+</sup>15], an alternative definition of the spectral construction presented in Chapter 4 by replacing the Fourier transform with a windowed Fourier transform. We showed that LSCNN allows to overcome the drawbacks of SCNN and allows to learn class-specific descriptors that are expressive, localized, and robust.

Chapter 6 is devoted to the main contribution of this thesis: a spatial domain formulation of convolution as correlation between a learnable filter and a patch, extracted by suitable patch operators. We reviewed several ways to construct such patch operator by providing details about the local charting procedure and the weighting functions defined thereon. In particular, the charting procedure of GCNN [MBBV15] extracts a local system of geodesic polar coordinates and the weighting functions are fixed Gaussians thereon. Unfortunately, this construction is limited to triangular meshes only. ACNN [BMRB16] employs fixed anisotropic

diffusion kernels as spatial weighting functions, allowing to extract patches capturing directional structures in the data. Interestingly, this construction can be used both on meshes and point clouds. The chapter ends showing that previous constructions can be casted as particular instances of MoNet [MBM<sup>+</sup>17], a more generic framework replacing hand-crafted weighting functions with learnable ones.

Spatial intrinsic CNNs are an extension of traditional Euclidean CNNs to non-Euclidean domains by replacing the traditional convolutional layers with by the ones proposed in GCNN, ACNN, or MoNet. Spatial intrinsic CNNs learn local, stationary, and compositional task-specific features.

Finally, Chapters 7 and 8 provide extensive experimental results showing that the proposed methods are successfully applicable to different shape analysis tasks and achieve state-of-the-art results.

## 9.1 Future work

There are several promising directions to extend the works presented in this thesis. In the following we highlight what we believe to be the most interesting ones:

- Traditional Euclidean CNN architectures are characterized by an alternation of convolutional and pooling layers. In the proposed approaches we focused our attention on the extension of the convolutional layer only. It appears natural to extend the pooling layer as well, for instance by constructing a hierarchy of coarser meshes.
- The lack of shift-invariance of non-Euclidean domains makes the patch operator point dependent, i.e. one has to compute a different patch operator for each vertex of the shape. This represents one of the major computational bottlenecks of the proposed approaches. A possible workaround in the context of range scans is to perform the convolution operation over the depth values in a sliding window fashion like traditional approaches. In order to make the operation intrinsic, we propose to deform the depth values inside such window using the metric tensor to adapt the filter to the underlying shape geometry.
- For the sake of presentation coherence, we only covered the application of the MoNet approach [MBM<sup>+</sup>17] to 3D shapes. Such framework, however, is not limited to shapes only, but it allows to extend CNNs to graphs as well. Extending CNNs to graphs is a very interesting research direction because a

variety of interesting data can be modelled as graphs. A possible application is related to drug design, where the molecules and proteins drugs are made of can be modeled as graphs and one is interested in analyzing their geometrical properties.

- In the proposed approaches, intrinsic CNNs are used to map the vertices of a shape to local descriptors, however one may be interested in the inverse problem of mapping features to their geometric counterpart. This is related to the problem of shape synthesis, where one wants to recover the shape geometry from some features. In computer vision, the similar problem of image generation is addressed in terms of variational autoencoders (VAE) and generative adversarial network (GAN). We would like to propose an intrinsic extension of such approaches for shape synthesis applications.





# Bibliography

- [ADK16] Yonathan Aflalo, Anastasia Dubrovina, and Ron Kimmel. Spectral generalized multi-dimensional scaling. *International Journal of Computer Vision*, 118(3):380–392, 2016.
- [ARAC14] Mathieu Andreux, Emanuele Rodola, Mathieu Aubry, and Daniel Cremers. Anisotropic Laplace-Beltrami operators for shape analysis. In *Proc. NORDIA*, 2014.
- [ASC11] Mathieu Aubry, Ulrich Schlickewei, and Daniel Cremers. The wave kernel signature: a quantum mechanical approach to shape analysis. In *Proc. ICCV*, 2011.
- [ASK<sup>+</sup>05] Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. SCAPE: Shape completion and animation of people. *TOG*, 24(3):408–416, 2005.
- [BBG94] Pierre Bérard, Gérard Besson, and Sylvain Gallot. Embedding Riemannian manifolds by their heat kernel. *Geometric & Functional Analysis*, 4(4):373–398, 1994.
- [BBI01] Dmitri Burago, Yuri Burago, and Sergei Ivanov. *A course in metric geometry*. American Mathematical Society, 2001.
- [BBK06] Alexander Bronstein, Michael Bronstein, and Ron Kimmel. Generalized multidimensional scaling: A framework for isometry-invariant partial surface matching. *PNAS*, 103(5):1168–1172, 2006.
- [BBK08] Alex Bronstein, Michael Bronstein, and Ron Kimmel. *Numerical geometry of non-rigid shapes*. Springer, 2008.
- [Bel61] Richard Bellman. *Adaptive control processes: A guided tour*. Princeton University Press, 1961.

- [BGL<sup>+</sup>94] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a “siamese” time delay neural network. In *Proc. NIPS*, 1994.
- [BGLL17] Denny Britz, Anna Goldie, Thang Luong, and Quoc Le. Massive exploration of neural machine translation architectures. Technical Report arXiv:1703.03906, 2017.
- [BGV92] Bernhard Boser, Isabelle Guyon, and Vladimir Vapnik. A training algorithm for optimal margin classifiers. In *Proc. Workshop on Computational Learning Theory*, pages 144–152, 1992.
- [BK10] Michael Bronstein and Iasonas Kokkinos. Scale-invariant heat kernel signatures for non-rigid shape recognition. In *Proc. CVPR*, 2010.
- [BM92] Paul Besl and Neil McKay. A method for registration of 3-D shapes. *Trans. PAMI*, 14(2):239–256, 1992.
- [BMM<sup>+</sup>15] Davide Boscaini, Jonathan Masci, Simone Melzi, Michael Bronstein, Umberto Castellani, and Pierre Vandergheynst. Learning class-specific descriptors for deformable shapes using localized spectral convolutional networks. *Computer Graphics Forum*, 34(5):13–23, 2015.
- [BMP00] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape context: A new descriptor for shape matching and object recognition. In *Proc. NIPS*, 2000.
- [BMR<sup>+</sup>16] Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Michael Bronstein, and Daniel Cremers. Anisotropic diffusion descriptors. *Computer Graphics Forum*, 35(2):431–441, 2016.
- [BMRB16] Davide Boscaini, Jonathan Masci, Emanuele Rodolà, and Michael Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In *Proc. NIPS*, pages 3189–3197, 2016.
- [BRLB14] Federica Bogo, Javier Romero, Matthew Loper, and Michael Black. FAUST: Dataset and evaluation for 3D mesh registration. In *Proc. CVPR*, 2014.
- [BZSL13] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. Technical Report arXiv:1312.6203, 2013.

- [CGGS12] Dan Cireşan, Alessandro Giusti, Luca Gambardella, and Jürgen Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In *Proc. NIPS*, pages 2843–2851, 2012.
- [CGGS13] Dan Cireşan, Alessandro Giusti, Luca Gambardella, and Jürgen Schmidhuber. Mitosis detection in breast cancer histology images with deep neural networks. In *Proc. MICCAI*, pages 411–418, 2013.
- [CK15] Qifeng Chen and Vladlen Koltun. Robust nonrigid registration by convex optimization. In *Proc. ICCV*, pages 2039–2047, 2015.
- [CL06] Ronald Coifman and Stéphane Lafon. Diffusion maps. *Applied and computational harmonic analysis*, 21(1):5–30, 2006.
- [CLL<sup>+</sup>05] Ronald Coifman, Stéphane Lafon, Ann Lee, Mauro Maggioni, Boaz Nadler, Frederick Warner, and Steven Zucker. Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. *PNAS*, 102(21):7426–7431, 2005.
- [CM92] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image and vision computing*, 10(3):145–155, 1992.
- [CMMS11] Dan Cireşan, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. A committee of neural networks for traffic sign classification. In *International Joint Conference on Neural Networks*, pages 1918–1921, 2011.
- [CMS12] Dan Cireşan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Proc. CVPR*, pages 3642–3649, 2012.
- [COC14] Étienne Corman, Maks Ovsjanikov, and Antonin Chambolle. Supervised descriptor learning for non-rigid shape matching. In *Proc. NORDIA*, 2014.
- [CRB<sup>+</sup>16] Luca Cosmo, Emanuele Rodolà, Michael Bronstein, Andrea Torsello, Daniel Cremers, and Yusuf Sahillioğlu. SHREC’16: Partial matching of deformable shapes. In *Proc. 3DOR*, 2016.
- [CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

- [Cyb89] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989.
- [DBV16] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proc. NIPS*, pages 3844–3852, 2016.
- [DC76] Manfredo Do Carmo. *Differential geometry of curves and surfaces*. Prentice-Hall, 1976.
- [DCF92] Manfredo Do Carmo and Francis Flaherty. *Riemannian geometry*. Birkhäuser, 1992.
- [DMAMS10] Julie Digne, Jean-Michel Morel, Nicolas Audfray, and Charyar Mehdi-Souzani. The level set tree on meshes. In *Proc. 3DPVT*, 2010.
- [DT05] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proc. CVPR*, 2005.
- [EK01] Asi Elad and Ron Kimmel. Bending invariant representations for surfaces. In *Proc. CVPR*, 2001.
- [EK03] Asi Elad and Ron Kimmel. On bending invariant signatures for surfaces. *Trans. PAMI*, 25(10):1285–1295, 2003.
- [FCNL13] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *Trans. PAMI*, 35(8):1915–1929, 2013.
- [Fuk80] Kunihiro Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.
- [GBAL09] Katarzyna Gebal, Jakob Andreas Bærentzen, Henrik Aanæs, and Rasmus Larsen. Shape analysis using the auto diffusion function. *Computer Graphics Forum*, 28(5):1405–1413, 2009.
- [GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.

- [GFGS06] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proc. ICML*, pages 369–376, 2006.
- [GLF<sup>+</sup>09] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *Trans. PAMI*, 31(5):855–868, 2009.
- [GSC99] Felix Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with LSTM. Technical report, 1999.
- [HBL15] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. Technical Report arXiv:1506.05163, 2015.
- [HCL06] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *Proc. CVPR*, pages 1735–1742, 2006.
- [HS85] Dorit Hochbaum and David Shmoys. A best possible heuristic for the k-center problem. *Mathematics of operations research*, 10(2):180–184, 1985.
- [HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
- [HSK<sup>+</sup>12] Geoffrey Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. Technical Report arXiv:1207.0580, 2012.
- [HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.
- [HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. CVPR*, pages 770–778, 2016.

- [JH99] Andrew Johnson and Martial Hebert. Using spin images for efficient object detection in cluttered 3D scenes. *Trans. PAMI*, 21:433–449, 1999.
- [KBBV15] Artiom Kovnatsky, Michael Bronstein, Xavier Bresson, and Pierre Vandergheynst. Functional correspondence by matrix completion. In *Proc. CVPR*, pages 905–914, 2015.
- [KBLB12] Iasonas Kokkinos, Michael Bronstein, Roei Litman, and Alex Bronstein. Intrinsic shape context descriptors for deformable shapes. In *Proc. CVPR*, pages 159–166, 2012.
- [KKBL15] Itay Kezurer, Shahar Kovalsky, Ronen Basri, and Yaron Lipman. Tight relaxations of quadratic matching. *Computer Graphics Forum*, 34(5), 2015.
- [KLCF10] Vladimir Kim, Yaron Lipman, Xiaobai Chen, and Thomas Funkhouser. Möbius transformations for global intrinsic symmetry analysis. *Computer Graphics Forum*, 29(5):1689–1700, 2010.
- [KLF11] Vladimir Kim, Yaron Lipman, and Thomas Funkhouser. Blended intrinsic maps. *TOG*, 30(4):79, 2011.
- [KLM<sup>+</sup>12] Vladimir Kim, Wilmot Li, Niloy Mitra, Stephen DiVerdi, and Thomas Funkhouser. Exploring collections of 3D models using fuzzy correspondences. *TOG*, 31(4):1–11, 2012.
- [KS98] Ron Kimmel and James Sethian. Computing geodesic paths on manifolds. *PNAS*, pages 8431–8435, 1998.
- [KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. ImageNet classification with deep convolutional neural networks. In *Proc. NIPS*, pages 1097–1105, 2012.
- [LB14] Roei Litman and Alexander Bronstein. Learning spectral descriptors for deformable shape correspondence. *Trans. PAMI*, 36(1):170–180, 2014.
- [LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

- [LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
- [LD11] Yaron Lipman and Ingrid Daubechies. Conformal Wasserstein distances: Comparing surfaces in polynomial time. *Advances in Mathematics*, 227(3):1047–1077, 2011.
- [Lév06] Bruno Lévy. Laplace-Beltrami eigenfunctions towards an algorithm that “understands” geometry. In *Proc. SMI*, 2006.
- [LGB<sup>+</sup>13] Zhouhui Lian, Afzal Godil, Benjamin Bustos, Mohamed Daoudi, Jeroen Hermans, Shun Kawamura, Yukinori Kurita, Guillaume Lavoué, Hien Van Nguyen, Ryutarou Ohbuchi, Yuri Ohkita, Yuya Ohishif, Fatih Porikli, Martin Reuter, Ivan Sipiran, Dirk Smeets, Paul Suetens, Hedi Tabia, and Dirk Vandermuelen. A comparison of methods for non-rigid 3D shape retrieval. *Pattern Recognition*, 46(1):449–461, 2013.
- [LH05] Marius Leordeanu and Martial Hebert. A spectral technique for correspondence problems using pairwise constraints. In *Proc. ICCV*, 2005.
- [Lin70] Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. Technical Report Master’s Thesis, 1970.
- [Lin76] Seppo Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16(2):146–160, 1976.
- [LKF10] Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. Convolutional networks and applications in vision. In *IEEE International Symposium on Circuits and Systems*, pages 253–256, 2010.
- [Low04] David Lowe. Distinctive image features from scale-invariant keypoints. *Trans. IJCV*, 60(2):91–110, 2004.
- [LSP08] Hao Li, Robert Sumner, and Mark Pauly. Global correspondence optimization for non-rigid registration of depth scans. *Computer Graphics Forum*, 27:1421–1430, 2008.
- [LZ10] Bruno Levy and Richard Hao Zhang. Spectral geometry processing. In *ACM SIGGRAPH Course Notes*, 2010.

- [Mal12] Stéphane Mallat. Group invariant scattering. *Communications on Pure and Applied Mathematics*, 65(10):1331–1398, 2012.
- [MBBV15] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Geodesic convolutional neural networks on Riemannian manifolds. In *Proc. ICCV Workshops*, pages 37–45, 2015.
- [MBM<sup>+</sup>17] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proc. CVPR*, 2017.
- [MCH<sup>+</sup>06] Siddharth Manay, Daniel Cremers, Byung-Woo Hong, Anthony Yezzi, and Stefano Soatto. Integral invariants for shape matching. *Trans. PAMI*, 28(10):1602–1618, 2006.
- [MCUP04] Jiri Matas, Ondrej Chum, Martin Urban, and Tomás Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and vision computing*, 22(10):761–767, 2004.
- [Mém11] Facundo Mémoli. Gromov-Wasserstein distances and the metric approach to object matching. *Foundations of computational mathematics*, 11(4):417–487, 2011.
- [MHK<sup>+</sup>08] Diana Mateus, Radu Horaud, David Knossow, Fabio Cuzzolin, and Edmond Boyer. Articulated shape matching using Laplacian eigenfunctions and unsupervised point registration. In *Proc. CVPR*, pages 1–8, 2008.
- [MS05] Facundo Mémoli and Guillermo Sapiro. A theoretical and computational framework for isometry invariant recognition of point cloud data. *Foundations of Computational Mathematics*, 5(3):313–347, 2005.
- [NH10] Vinod Nair and Geoffrey Hinton. Rectified linear units improve restricted Boltzmann machines. In *Proc. ICML*, pages 807–814, 2010.
- [NIH<sup>+</sup>11] Richard Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *Proc. ISMAR*, pages 127–136, 2011.



- [OBCS<sup>+</sup>12] Maks Ovsjanikov, Mirela Ben-Chen, Justin Solomon, Adrian Butscher, and Leonidas Guibas. Functional maps: A flexible representation of maps between shapes. *TOG*, 31(4):30, 2012.
- [OFGD02] Robert Osada, Thomas Funkhouser, Bernard Chazelle, and David Dobkin. Shape distributions. *TOG*, 21:807–832, 2002.
- [OMMG10] Maks Ovsjanikov, Quentin Mérigot, Facundo Mémoli, and Leonidas Guibas. One point isometric matching with the heat kernel. *Computer Graphics Forum*, 29(5):1555–1564, 2010.
- [PBB<sup>+</sup>13] Jonathan Pokrass, Alexander Bronstein, Michael Bronstein, Pablo Sprechmann, and Guillermo Sapiro. Sparse modeling of intrinsic correspondences. In *Computer Graphics Forum*, pages 459–468, 2013.
- [RBA<sup>+</sup>12] Emanuele Rodolà, Alex Bronstein, Andrea Albarelli, Filippo Bergamasco, and Andrea Torsello. A game-theoretic approach to deformable shape matching. In *Proc. CVPR*, 2012.
- [RCB<sup>+</sup>17] Emanuele Rodolà, Luca Cosmo, Michael Bronstein, Andrea Torsello, and Daniel Cremers. Partial functional correspondence. *Computer Graphics Forum*, 36(1):222–236, 2017.
- [Ros97] Steven Rosenberg. *The Laplacian on a Riemannian manifold: an introduction to analysis on manifolds*. Cambridge University Press, 1997.
- [RRBW<sup>+</sup>14] Emanuele Rodolà, Samuel Rota Buló, Thomas Windheuser, Matthias Vestner, and Daniel Cremers. Dense non-rigid shape correspondence using random forests. In *Proc. CVPR*, pages 4177–4184, 2014.
- [Rus07] Raif Rustamov. Laplace-Beltrami eigenfunctions for deformation invariant shape representation. In *Proc. SGP*, 2007.
- [RWP06] Martin Reuter, Franz-Erich Wolter, and Niklas Peinecke. Laplace-Beltrami spectra as Shape-DNA of surfaces and solids. *Computer-Aided Design*, 38(4):342–366, 2006.
- [SGS15] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. In *Proc. NIPS*, 2015.

- [SK14] Alon Shtern and Ron Kimmel. Matching the LBO eigenspace of non-rigid shapes via high order statistics. *Axioms*, 3(3):300–319, 2014.
- [SMKLM15] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3D shape recognition. In *Proc. ICCV*, pages 945–953, 2015.
- [SNB<sup>+</sup>12] Justin Solomon, Andy Nguyen, Adrian Butscher, Mirela Ben-Chen, and Leonidas Guibas. Soft maps between surfaces. *Computer Graphics Forum*, 31(5):1617–1626, 2012.
- [SOG09] Jian Sun, Maks Ovsjanikov, and Leonidas Guibas. A concise and provably informative multi-scale signature based on heat diffusion. *Computer Graphics Forum*, 28(5):1383–1392, 2009.
- [SP93] Jürgen Schmidhuber and Daniel Prelinger. Discovering predictable classifications. *Neural Computation*, 5(4):625–635, 1993.
- [SRV16] David Shuman, Benjamin Ricaud, and Pierre Vandergheynst. Vertex-frequency analysis on graphs. *Applied and Computational Harmonic Analysis*, 40(2):260–291, 2016.
- [SSK<sup>+</sup>13] Jamie Shotton, Toby Sharp, Alex Kipman, Andrew Fitzgibbon, Mark Finocchio, Andrew Blake, Mat Cook, and Richard Moore. Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, 56(1):116–124, 2013.
- [SSTF<sup>+</sup>14] Edgar Simo-Serra, Eduard Trulls, Luis Ferraz, Iasonas Kokkinos, and Francesc Moreno-Noguer. Fracking deep convolutional image descriptors. Technical Report arXiv:1412.6537, 2014.
- [STDS14] Samuele Salti, Federico Tombari, and Luigi Di Stefano. SHOT: Unique signatures of histograms for surface and texture description. *Computer Vision and Image Understanding*, 125:251–264, 2014.
- [SY11] Yusuf Sahillioğlu and Yücel Yemez. Coarse-to-fine combinatorial matching for dense isometric shape correspondence. *Computer Graphics Forum*, 30(5):1461–1470, 2011.
- [SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. Technical Report arXiv:1409.1556, 2014.

- [SZPY12] Georgia Sandbach, Stefanos Zafeiriou, Maja Pantic, and Lijun Yin. Static and dynamic 3D facial expression recognition: A comprehensive survey. *Image Vision Computing*, 30(10):683–697, 2012.
- [Tau95] Gabriel Taubin. A signal processing approach to fair surface design. In *Proc. Computer graphics and interactive techniques*, pages 351–358, 1995.
- [TBW<sup>+</sup>11] Art Tevs, Alexander Berner, Michael Wand, Ivo Ihrke, and Hans-Peter Seidel. Intrinsic shape matching by planned landmark sampling. *Computer Graphics Forum*, 30(2):543–552, 2011.
- [TKR08] Lorenzo Torresani, Vladimir Kolmogorov, and Carsten Rother. Feature correspondence via graph matching: Models and global optimization. In *Proc. ECCV*, 2008.
- [TPM06] Oncel Tuzel, Fatih Porikli, and Peter Meer. Region covariance: A fast descriptor for detection and classification. *Proc. ECCV*, pages 589–600, 2006.
- [TYRW14] Yaniv Taigman, Ming Yang, Marc’Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proc. CVPR*, pages 1701–1708, 2014.
- [Ume88] Shinji Umeyama. An eigendecomposition approach to weighted graph matching problems. *Trans. PAMI*, 10(5):695–703, 1988.
- [VLR<sup>+</sup>17] Matthias Vestner, Roei Litman, Emanuele Rodolà, Alex Bronstein, and Daniel Cremers. Product manifold filter: Non-rigid shape correspondence via kernel density estimation in the product space. Technical Report arXiv:1701.00669, 2017.
- [WHC<sup>+</sup>16] Lingyu Wei, Qixing Huang, Duygu Ceylan, Etienne Vouga, and Hao Li. Dense human body correspondences using convolutional networks. In *Proc. CVPR*, pages 1544–1553, 2016.
- [WSC<sup>+</sup>16] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol

- Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. Technical Report arXiv:1609.08144, 2016.
- [WSK<sup>+</sup>15] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *Proc. CVPR*, pages 1912–1920, 2015.
- [ZBVH09] Andrei Zaharescu, Edmond Boyer, Kiran Varanasi, and Radu Horaud. Surface feature detection and description with applications to mesh matching. In *Proc. CVPR*, pages 373–380, 2009.
- [ZWW<sup>+</sup>10] Yun Zeng, Chaohui Wang, Yang Wang, Xianfeng Gu, Dimitris Samaras, and Nikos Paragios. Dense non-rigid surface registration using high-order graph matching. In *Proc. CVPR*, 2010.