Geometric Deep Learning

SIGGRAPH ASIA 2016 COURSE NOTES

Organizers & Lecturers:

Jonathan Masci, Emanuele Rodolà, Davide Boscaini,
Michael M. Bronstein, Hao Li

SA '16 Courses, December 05-08, 2016, Macao
ACM 978-1-4503-4538-5/16/12.
http://dx.doi.org/10.1145/2988458.2988485

Contents

1 Introduction **1**
- 1.1 Course goals . . . 1
- 1.2 Motivation . . . 1
- 1.3 A brief history . . . 2
- 1.4 The challenges of geometric deep learning . . . 2

2 Basics of Deep Learning **4**
- 2.1 Random forests . . . 5
- 2.2 Neural networks . . . 6
- 2.3 Convolutional neural networks . . . 7
- 2.4 Learning techniques . . . 9

3 Extrinsic Deep Learning **11**
- 3.1 Volumetric CNNs . . . 11
  - 3.1.1 3D Shape Representation . . . 11
  - 3.1.2 Network Architecture . . . 12
  - 3.1.3 Training and Data Collection . . . 13
- 3.2 View-based CNNs . . . 13
  - 3.2.1 Classification Network . . . 14
- 3.3 Human shape correspondence . . . 15
  - 3.3.1 Correspondence Computation . . . 17
  - 3.3.2 Training Data Generation . . . 19
  - 3.3.3 Network Design and Training . . . 19
- 3.4 Performance Capture . . . 20

4 Spectral Learning Methods **24**
- 4.1 Spectral analysis on manifolds . . . 24
- 4.2 Spectral descriptors . . . 26
- 4.3 Optimal spectral descriptors . . . 27
- 4.4 Deformable shape correspondence with random forests . . . 28
  - 4.4.1 Inference . . . 28
  - 4.4.2 Learning . . . 29
  - 4.4.3 Shape matching via the label space . . . 29

5 Intrinsic Convolutional Neural Networks **31**
- 5.1 Intrinsic CNNs in the spectral domain . . . 31
  - 5.1.1 Spectral CNNs . . . 31
  - 5.1.2 Localized spectral CNNs . . . 32
- 5.2 Intrinsic CNNs in the spatial domain . . . 33

5.2.1 Geodesic CNNs 34
5.2.2 Anisotropic diffusion CNNs 35
5.3 Applications 36
5.3.1 Local descriptors 37
5.3.2 Correspondence 37

1

Introduction

1.1 Course goals

The goal of these course notes is to describe the main mathematical ideas behind geometric deep learning and to provide implementation details for several applications in shape analysis and synthesis, computer vision and computer graphics. The text in the course materials is primarily based on previously published work including [71, 56, 86, 100, 10, 59, 97, 12, 11] among several others. With these notes we gather and provide a clear picture of the key concepts and techniques that fall under the umbrella of geometric deep learning, and illustrate the applications they enable. We also aim to provide practical implementation details for the methods presented in these works, as well as suggest further readings and extensions of these ideas.

1.2 Motivation

The past decade in computer vision research has witnessed the re-emergence of "deep learning", and in particular convolutional neural network (CNN) techniques, allowing to learn powerful image feature representations from large collections of examples. CNNs achieve a breakthrough in performance in a wide range of applications such as image classification, segmentation, detection and annotation. Nevertheless, when attempting to apply the CNN paradigm to 3D shapes (feature-based description, similarity, correspondence, retrieval, etc.) one has to face fundamental differences between images and geometric objects. Shape analysis and geometry processing pose new challenges that are non-existent in image analysis, and deep learning methods have only recently started penetrating into the 3D shape community. CNNs have been applied to 3D data in recent works using standard (Euclidean) CNN architectures applied to volumetric or view-based shape representations. Intrinsic versions of CNNs have also been proposed very recently with the generalization of the CNN paradigm to non-Euclidean manifolds, allowing them to deal with shape deformations. These "generalized" CNNs can be used to learn invariant shape features and correspondence, allowing to achieve state-of-the-art performance in several shape analysis tasks, while at the same time allowing for different shape representations, e.g. meshes, point clouds, or graphs. The purpose of this short course is to overview the foundations and the current state of the art on learning techniques for 3D shape analysis and geometry processing. Special focus will be put on deep learning techniques (CNN) applied to Euclidean and non-Euclidean manifolds for tasks of shape classification, retrieval, reconstruction and correspondence. The course will present in a new light the problems of shape analysis, emphasizing the analogies and differences with

the classical 2D setting, and showing how to adapt popular learning schemes in order to deal with deformable shapes.

These course notes will assume no particular background, beyond some basic working knowledge that is a common denominator for people in the field of computer graphics. All the necessary notions and mathematical foundations will be described. The course is targeted to graduate students, practitioners, and researchers interested in shape analysis, synthesis, matching, retrieval, and big data.

1.3 A brief history

Deep learning methods have literally shaken many realms in the academia and industry in the past few years. Technology giants like Apple, Google and Facebook have been aggressively hunting for experts in the field and acquiring promising deep learning start-up companies for unprecedented amounts, all of which are indicative of the enormous bets the industry is currently placing on this technology. Nowadays, deep learning methods are already widely used in commercial applications, including Siri speech recognition in Apple iPhone, Google text translation, and Mobileye vision-based technology for autonomously driving cars.

Though "deep learning" has become somewhat of a buzzword often taken out of context and vaguely referring to artificial intelligence in general, the original term refers to learning complicated concepts by a machine, by means of building them out of simpler ones in a hierarchical or "multi-layer" manner. Artificial neural networks are a popular realization of such deep multi-layer hierarchies inspired by the signal processing done in the human brain. The first artificial neural network models are usually credited to neuroscientists Hebb and Rosenblatt (late 1940s and 50s, respectively); deep architectures and learning algorithms that resemble the modern ones can already be found in the works of Ivahnenko and coauthors in the late 1960s, though the term "deep networks" appeared much later [22, 3]. Research of artificial neural networks was thorny with cycles of popularity and near oblivion. In the past few years, the growing computational power of modern GPU-based computers, availability of large training datasets ("big data"), and efficient stochastic optimization methods allowed creating and effectively training complex network models with many layers and degrees of freedom. This allowed deep neural networks achieve a qualitative breakthrough in performance on a wide variety of tasks, from speech recognition and machine translation to image analysis and computer vision, igniting the renaissance of the field.

1.4 The challenges of geometric deep learning

Dealing with signals such as speech, images, or video on 1D-, 2D- and 3D Euclidean domains, respectively, has been the main focus of research in deep learning for the past decades. However, in the recent years, more and more fields have to deal with data residing on non-Euclidean geometric domains, which we call here "geometric data" for brevity. For instance, in social networks, the characteristics of users can be modelled as signals on the vertices of the social graph. Sensor networks are distributed interconnected sensors, whose readings are modelled as time-dependent signals on graphs. In computer graphics and vision, 3D shapes are modelled as Riemannian manifolds (surfaces) endowed with properties such as color texture or motion field (e.g. dynamic meshes). Even more complex examples include networks of operators, such as functional correspondences [66]

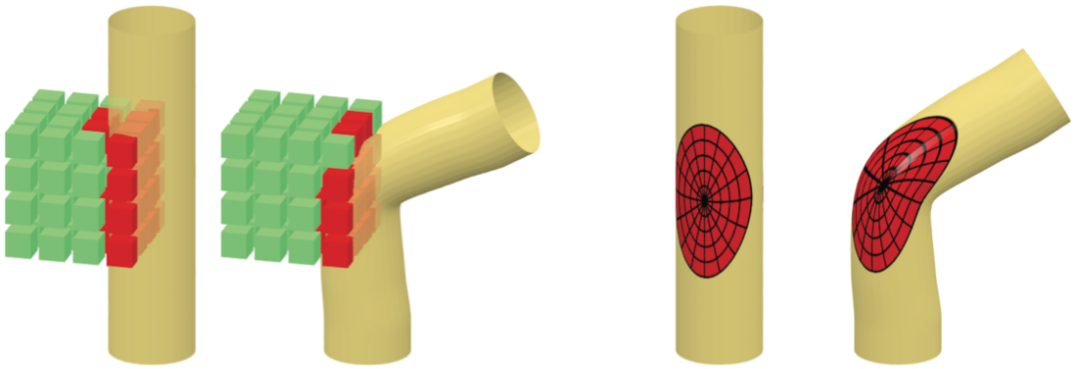


Figure 1.1: Left: extrinsic methods such as volumetric CNNs [100] treat 3D geometric data in its Euclidean representation. Such a representation is not invariant to deformations (e.g., in the shown example, the filter that responds to features on a straight cylinder would not respond to a bent one). Right: in an intrinsic representation, the filter is applied on the surface itself, thus being invariant to deformations.

or difference operators [74] in a collection of 3D shapes, or orientations of overlapping cameras in multi-view vision (structure from motion) problems. Furthermore, modelling high-dimensional data with graphs is an increasingly popular trend in general data science, where graphs are used to describe the low-dimensional intrinsic structure of the data.

On the one hand, the complexity of geometric data and the availability of large datasets (in the case of social networks, of the order of billions of examples) make it tempting and very desirable to resort to machine learning techniques. On the other hand, 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 even not well defined on non-Euclidean domains. This happens to be a major obstacle that so far has precluded the use of successful deep learning methods such as convolutional or recurrent neural networks on non-Euclidean geometric data. As a result, the quantitative and qualitative breakthrough that deep learning methods have brought into speech recognition, natural language processing, and computer vision has not yet come to fields such as computer graphics or computational sociology.

2

Basics of Deep Learning

In the recent years we have experienced a paradigm shift from axiomatic modeling to data-driven modeling due to the unprecedented performance of Deep Neural Networks in tasks such as image and speech recognition, machine translation, and image segmentation and detection.

The main reasons to this success are to be found in: (1) the growing computational power of modern GPU based computers, (2) larger annotated datasets, and (3) more efficient stochastic optimization methods and architectural changes that allow to effectively train much more complex networks with many layers and degrees of freedom.

These factors made the qualitative breakthrough possible, and the wide-spread interest from various communities ranging from pure machine learning to system design rapidly brought deep learning to be widely used in commercial applications, including Siri speech recognition, Google text translation, and the Mobileye autonomous driving technology.

Key aspect of Deep Learning (DL) systems is that they are able to learn representations directly from the raw input data without requiring any hand-crafted feature extraction stage. They allow to perform representation learning at various levels of representation, therefore in a hierarchical fashion, by composing simple non-linear building blocks usually referred to as *layers*. This composition results in very powerful models which can leverage the large quantities of annotated data available today.

This is the reason why we have experienced such an outstanding improvement in tasks where all machine learning methods rely on a feature extraction stage, namely speech signals, images, and everything related to perceptual data.

This is in sharp contrast with the now old paradigm of machine learning where features had to be provided by the user. During the neural network "dark age", when methods such as SVM defined the gold standard for classification, very little effort was devoted to the tweaking of their parameters and most of the work was instead devoted to a good feature extraction strategy. As an example, the typical computer vision recognition pipeline before DL consisted in computing SIFT, doing sparse vector quantization and pooling to finally feed everything to an SVM classifier. Of course all of these stages are completely arbitrary and sub-optimal to say the least. With DL one needs to have very few assumptions on what is good for a given task and in most cases it is enough to have the input image and its annotation; this leaves to the data-driven approach the task of finding what is the right feature and the right feature aggregation strategy.

The ability to learn representations directly from the data with very little prior has allowed many machine learning techniques to shine and to already surpass human performance in some specific domains.

According to how many processing steps separate the input to the output of a given

model we obtain *shallow* and *deep* models. SVM are shallow models as they can be mimicked by a single layer neural network (implementing the kernel function) whereas models with more than a single layer start to be called deep. However, during the years such term has had various meanings and currently it is used with architectures with at least 10 or more layers.

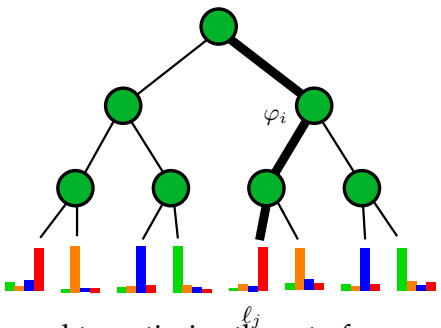
Before introducing neural networks we first review the popular random forest paradigm. Despite being composed of several decision steps, the random forest model is still considered to be shallow as the decisions at each level are taken on the original input space. In deep networks each stage operates on the output of the previous one instead, therefore performing feature learning.

2.1 Random forests

Random forests [13] are ensembles of decision trees that have become quite popular in the computer vision and pattern recognition communities to solve classification and regression problems. Applications of random forests range from object detection, tracking and action recognition [29] to 3D pose estimation [90] to name just a few. The key idea behind the random forest construction is that one can build a much stronger classifier by combining several weak ones (the individual trees), which are trained independently with a random sampling of the labelled training data [21]. This approach is particularly appealing, since each tree can be trained efficiently, can be easily parallelized, and the randomized feature selection allows to limit the correlation among trees and thus ensure good generalization.

In these notes we will focus on the application of the random forest paradigm to solve *classification* tasks, i.e., where the output of the forest is a discrete, categorical label – or in its soft formulation, a probability distribution over some label space. In Section 4.4 we will see how this model is useful to tackle shape matching problems.

As we mentioned, random forests are simple collections of decision trees, and the output of a forest is an aggregation of the outputs of its constituent trees. Decision trees are tree-structured classifiers that make a prediction by routing an input feature sample along the interior nodes until it reaches a terminal node (or *leaf*), where the actual classification takes place. Each interior node of the tree performs routing decisions based on a (binary) test or "split" function $\varphi^i_{\Theta_i}$, parametrized by Θ_i. The test function is applied to the incoming data, which is then sent to the left or right child depending on the outcome of the test. In the inset figure, the path marked in bold illustrates the routing of an input sample along a tree to reach the leaf ℓ_j, containing the final label distribution for that sample. Designing a decision tree amounts to defining two key ingredients: 1) the test functions φ_i associated to each node, and 2) the label predictors associated to each leaf (we refer the reader to Section 4.4 for examples). As is usually the case with machine learning paradigms, we distinguish between a training and a testing phase.



During *training*, a set of labelled training points is used to optimize the set of parameters $\{\Theta_i\}$ of the split functions. The training data that reaches a node is split into two subsets (one per child node), such that the information gain incurred by the split is maximized (other energy functions are possible and depend on the task at hand). A key aspect of the training process is the fact that *randomness* is injected into it: This is typically done

by using a random sample of all the training data, or by selecting the optimal parameters at each node from a pool of randomly generated values. The training process is repeated independently for each tree in the forest, leading to component trees that are different from each other. This, in turn, leads to robustness to noisy data, de-correlation between the individual tree predictions, and thus to improved generalization [21].

At *testing* time, a previously unseen data point is routed through each tree in the forest, undergoing a number of predefined tests. When the data point reaches a leaf, it is associated a class label (or a distribution over the space of labels). The final forest prediction can be obtained, for example, by simply averaging the class posteriors of each tree. Note that, differently from the training process, the testing phase is completely deterministic once the trees are fixed.

2.2 Neural networks

Neural networks are learning methods able to extract hierarchical representations from the data via several processing stages, defining a mapping from input to output which exhibits multiple levels of abstraction.

Each stage is referred to as a *layer*, or basic building block, computing a very simple function such as linear combinations and point-wise non-linearities. The concatenation of such blocks into a DAG (directed acyclic graph) constitutes a feed-forward neural network, whereas in the case of a DCG (directed cyclic graph) this results in recurrent models, of which LSTM [41] is the most prominent example.

Each layer can be thought of as a function f_i, where index i identifies the layer in the hierarchy, parametrized by the (possibly empty) set Θ_i. Deep feed-forward networks are composed of fully connected layers of the form

$$\text{fc}(x) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}), \tag{2.1}$$

where σ is a point-wise non-linearity such as ReLU, $\sigma(x) = \max(0, x)$. Other common choices are s-shaped functions such as hyperbolic tangent and logistic. The learnable parameters are the projection matrix $\mathbf{W}$ and the bias vector $\mathbf{b}$, which for simplicity are concatenated into the set $\Theta = \{\mathbf{W}, \mathbf{b}\}$.

A deep feed-forward network is therefore a complex non-linear function resulting by the composition of several such fully connected layers:

$$F_\Theta(x) = f_n(\dots f_1(x, \theta_1), \theta_n), \tag{2.2}$$

where $\Theta = \{\theta_i | i = 1..n\}$ is the set of all parameters in each of the utilized layers.

In this document we will focus only on supervised feed-forward learning methods, the most used approach in computer vision tasks, where the desired targets are known in advance. In the cases where the input-to-output mapping incurs in a long list of intermediate transformations, the models are said to be *deep*, otherwise they are referred to as *shallow*. It is important to note that the meaning of the word deep has greatly changed over the past few years given the recent advances in the field.

Unsupervised learning From a machine learning standpoint a very interesting and relatively unexplored area is the one of unsupervised learning. In this setting the model knows only the input signal and aims at modeling the generating input distribution that better explains the data: the unknown function y is generating the input x_i, and the function itself is the target of the learning process. Unsupervised models usually aim at reconstructing

the input while limiting the representational power of the model via sparsity penalties for example. Models in this family are called autoencoders. Another very popular paradigm for unsupervised learning which does not rely on the ill-posed reconstruction cost is the one of Predictability Minimization (PM) and Generative Adversarial Networks (GAN) which are able to sample very realistic natural images and whose representations are often competitive with fully supervised ones. A GAN model is composed by two networks, the generator and the discriminator. The generator is trained to produce plausible input samples from some input distribution, for example Gaussian noise, so that it makes the discriminator fail. The discriminator is trained to classify samples from the training set from samples generated by the generator. The result is a system able to learn features with no supervision which compete with the ones obtained with fully supervised techniques and quite impressive generative capabilities, in particular for scene and face generation when the generator is a deep convolutional network [31, 69].

Supervised learning In a generic supervised learning setting we are given a training dataset composed as follows

$$D = \{(x_i, t_i)\}_{i \in \mathcal{I}}, \tag{2.3}$$

where $x_i \in \mathbb{R}^n$ and the ground truth t_i is generated by an unknown function $y : \mathbb{R}^n \to \mathcal{Y}$; $t_i = y(x_i)$. $\mathcal{I}$ is an index set of samples in the dataset. In a supervised classification task the signal t_i can be thought of as a discrete quantity with values in $\mathcal{Y}$, and with cardinality equal to the number of possible classes whereas for regression $\mathcal{Y} = \mathbb{R}^m$.

More formally, given a model and a training dataset, one needs to define a way of assessing how good a solution is (by solution we mean a given Θ). This is done via the definition of a loss function $\mathcal{L}$, and the fitting of the parameters of the model is done through minimization of such function:

$$\Theta^* = \arg\min_{\Theta} \mathcal{L}_\Theta(D)\,. \tag{2.4}$$

For classification tasks, where each sample can be assigned to one of K mutually exclusive classes, the common choice is the *multinomial logistic* loss

$$\mathcal{L}_\Theta(x_i, t_i) = -\sum_{k=1}^{K} t_i \log(F_\Theta(x_i)) \tag{2.5}$$

where t_i is one-hot encoded (vector containing a 1 at the position of the ground truth label and 0 otherwise) and the output of the network $F(x_i)$ is K-dimensional with $softmax$ activation; this allows to interpret the output as posterior probability of the class given the input $p(F(x_i) = t_i | x_y, \Theta)$.

The minimization of the loss function is performed through some form of gradient descent and the gradient of the parameters of the network is computed with the backpropagation algorithm [72, 99].

2.3 Convolutional neural networks

Convolutional Networks are a particular class of neural network models whose main building blocks are *convolution* (C) and *pooling* (P).

What is nowadays known as convolutional network dates back to the Neocognitron of Fukushima [28] who first proposed to perform local operations with weight sharing. The author did not consider backpropagation, but a form of Hebbian learning instead.

LeCun and colleagues [50] later introduced gradient based learning and advanced the state-of-the-art for this class of models since then. We can say that what is now known as convolutional network is the latter variant of LeCun et al.

The convolutional layer C acts on a p-dimensional input $f(x) = (f_1(x), \ldots, f_p(x))$ and applies a filter-bank $W = (w_{l,l'} \in \mathbb{R}^{rc})$, where r and c indicate the number of rows and columns in the convolutional kernel; l and l' indicate the number of input and output maps:

$$g_l(x) = \sum_{l'=1}^{p} (f_{l'} \star w_{l,l'})(x). \tag{2.6}$$

After a convolutional layer, as with fully-connected layers, a point-wise non-linear activation function is applied; a common choice is to use ReLU (rectified linear units) of the form $\sigma(x) = max(0, x)$.

Convolutional filters have a local support, usually of very few pixels, however recently some authors [102, 107] have revised the classical dilated convolution which operates at increasing spatial resolutions and that can be efficiently learned using the well known algorithme à trous [58].

In order to introduce some invariance to small translations a pooling layer P can follow the convolution. It performs the following operation

$$g_l(x) = G(f_l(x') : x' \in \mathrm{Neigh}(x)), \tag{2.7}$$

where Neigh is a neighborhood around the point x and G is a permutation invariant aggregation function, usually L_p-norm with $p = \infty$ being the most popular and effective choice resulting in max-pooling. While pooling has been introduced to subsample the signal over non-overlapping neighborhoods – tiling the input image into regions of size 2x2 or 3x3 for example – it has become popular also to do overlapping pooling which results in a morphological dilation operation.

While initially popularized for pure classification tasks CNN thanks to the capability to reduce dimensionality through pooling operators they have immediately found application on tasks where the dimensionality does not need to be reduced, such as semantic segmentation where they excel by far the non-DL based approaches. In order to avoid losing spatial resolution deconvolution operations have been introduced, among which the simplest being the up-sampling with linear interpolation. Thanks to this many authors have started to tackle tasks where a prediction for every pixel in the input image needs to be taken, this comprehends optical flow estimation, semantic segmentation, image generation, volumetric based shape synthesis [24, 57, 105, 25].

Spatial Transformer Networks are a different class of models which aim to learn a perspective transformation of the input image, more generally a distortion field in the input space, so that the classification and detection tasks can be carried out more easily because of the suppression of unwanted signals such as the background and to the eventual rescaling and centering of the object of interest. They have been shown to deliver excellent performance in digit classification without data augmentation [44].

Attention based models instead [85, 62, 32, 101] aim at inferring an attention mask over the input or the hidden states of a network to emphasize only certain aspect of it. These models have been successfully used for tasks such as image captioning, generation, and speech recognition.

2.4 Learning techniques

Obtaining the parameters of deep learning models by minimizing a task-specific cost seems straightforward as these models are differentiable and therefore any gradient based technique could be used. This learning process is non-convex and it has always been linked with bad local minima; several techniques have been investigated to indeed mitigate such eventuality. A very interesting approach is the one called *flat minima search* [40] which aims at finding regions in the weight spaces where a small perturbation does not change the loss; in a bad local minima a tiny change of one parameter may in fact lead to huge differences in the loss. However it turns out that simpler stochastic optimization methods, in conjunction with much larger and overparameterized models, offer excellent empirical performance. Understanding the structure of the optimization problems, considering the growing interest in these methods, has of course let to a very active area of research [17, 45].

Even with the latest advances deep models are still much harder to train than shallow ones but attain much better performance when properly trained. This has opened up many interesting research directions to allow gradients to flow more easily as in Highway and Residual Networks [83, 34], better behaved activation functions as the Exponential Linear Unit [18], and understanding the training process itself from a "model of models" standpoint [84].

Having high capacity models allows to escape better local minima, and allows more complex mappings which could better capture the true important factors in the input data. However this makes possibly successful models prone to severe overfitting, meaning that the discrepancy between training performance and test performance is high and that they do not generalize well to unseen data.

Many techniques that helped breaking many of the records in pattern recognition aim indeed at reducing overfitting and are implemented as an additional weighted term to the loss function.

Here we briefly list them and refer the reader to the respective publications for details.

Pre-training Several works have shown, in the early days of deep learning, that initializing the weights by the ones obtained via unsupervised learning is beneficial and can lead improved and more robust results [27, 60, 51, 38, 93]. This process involves though a first training phase which can take lot of time and that has a diminishing return when the amount of labeled samples increases and has become less popular in practical applications. What is common in computer vision though is to take a model trained to classify objects in ImageNet [23] such as the VGG [81] or Inception [89] and to use its features or predictions for other higher level tasks.

Initialization is a critical point for deep networks and finding good initializations was one of the good arguments of pre-training. While initializing the weights from a normal distribution with zero-mean and small standard deviation is often sufficient to achieve good convergence properties, several authors have shown that ad-hoc initializations are often more powerful and can lead to better solutions [30, 35].

Weight decay is perhaps the most popular regularization technique for deep learning. It simply penalizes weights with large magnitude thus favoring smooth solutions and is defined by the regularizer $\|\Theta\|_2^2$

Dropout is a very successful technique to prevent co-adaptation of features [39]; it is among the simplest techniques to implement (a layer with no parameters) and because of this it is widely used. At training time each dimension at the input is dropped with a given probability p thus enforcing features to code for different things and to not rely on the presence of the other features. It can be seen as well as a very effective way of doing model averaging over an exponential number of networks, each one identified by the zero-one mask produced by the sampling. At test time, instead of doing an expensive averaging over an exponential number of possible masks, the input is rescaled by p; this cheap trick works reasonably well in feed forward networks and it's the classical way of applying dropout at test time. Given its popularity several follow up works have shown either improvements in terms of speed [96], applicability to recurrent networks [67], and theory [7, 47].

Batch Normalization [43] is a regularization that is implemented in form of a layer in deep networks and it is capable of reducing training times of very large models considerably, while also improving generalization. It normalizes each mini-batch during stochastic optimization to have zero mean and unit variance (estimates are build with moving averages during training), and then performs a parametric affine transformation to produce the output. After training, one can re-estimate the statistics on the test set or simply keep the training set estimates.

3

Extrinsic Deep Learning

3.1 Volumetric CNNs

Convolutional Neural Networks have demonstrated to outperform hand-crafted features and domain specific techniques for various core tasks in computer vision. These applications include classification [48], segmentation [57, 64, 75], regression [65], and synthesis tasks [25]. A wide range of applications have been recently introduced that process 3D geometric shapes, such as highly effective 3D object detection for RGB-D data [82], object classifcation of point clouds [61], 3D local feature matching [106], and 3D deformation flows [103].

A natural extention to classic CNNs that process 2D images is to process 3D data using a volumetric represencation and 3D convolutions. Wu and coworkers [100] presented 3D ShapeNets, a 3D deep learning framework for modeling shapes using a voxel discretization of 3D shapes, and demonstrated how spatial features can be learned directly in 3D. The approach consists of representing a geometric 3D shape as a probabilistic distribution of binary variables on a voxel grid. A convolutional deep belief network is used to learn the joint distribution of all 3D voxels based on massive amounts of 3D training data such as the CAD model database, ModelNet, and real-world scans from the NYU depth dataset [80]. In this work, they have demonstrated highly accurate object recognition from input depth maps and the ability to synthesize the missing parts of single-view input data.

3.1.1 3D Shape Representation

To use a volumetric convolutional neural network, a 3D shape in the form of a mesh can be represented as a probability distribution of binary values on a 3D grid. We define the inside of a mesh surface using a voxel value being 1 and the outside (or empty space) as 0. A relatively small grid resolution of $30 \times 30 \times 30$ is used in the experiments of Wu et al. [100]. A deep belief network (DBN) as introduced by Hinton et al. [37], which describes the joint probabilistic distribution over all pixels and labels in a 2D image, can be directly extended to the 3D domain by replacing pixels with voxels. Since the memory requirement increases drastically with the resolution of the 3D voxel volume, a fully connected convolutional DBN would require a massive amount of parameters to train. Wu et al. [100] therefore proposed to use a weight sharing approach to reduce the number of model parameters.

The convolutional deep learning model does not use pooling as it could yield greater

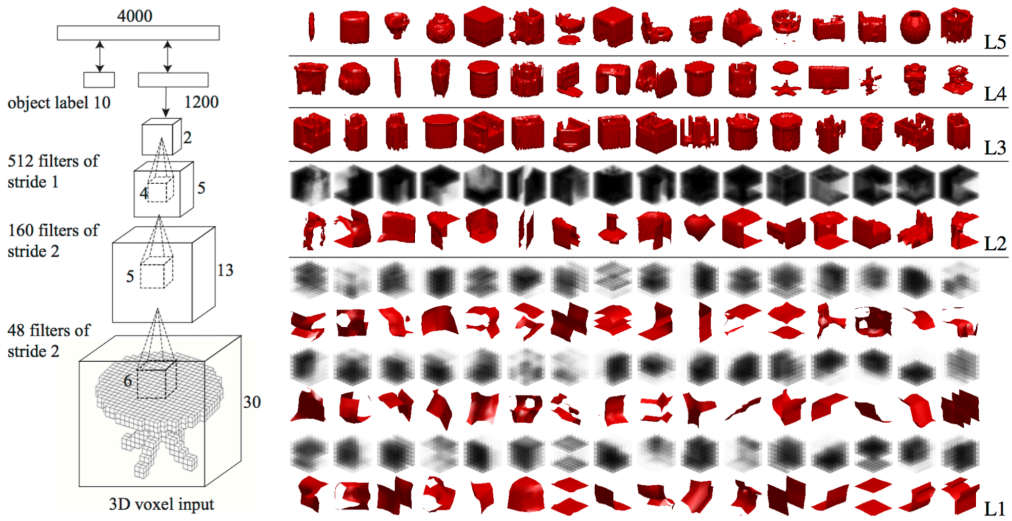


Figure 3.1: Left: network architecture of 3D ShapeNets for high-dimensional feature learning of 3D voxel input. Right: for every neuron, the 100 training examples with the highest response are averaged and their volume cropped inside the receptive field (gray). An implicit surface can be extracted via zero-crossing (red). The network captures the structures of low-level surface features (L1) to object parts (L2,L3) and entire objects (L4). Figure reproduced from [100].

uncertainty and the energy E for each convolutional layer is computed as:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{f}\sum_{j}\left(h_j^f (W^f \star v)_j + c^f h_j^f\right) - \sum_{l} b_l v_l \tag{3.1}$$

where v_l is a visibility unit, h_j^f a hidden unit in the feature channel f, W^f a convolutional filter, and $\star$ a convolution operation. To help with the reconstruction, each visibility unit v_l is associated with a unique bias term b_l and all hidden units h_j^f in the same convolution channel share the same bias term c^f. The extracted features can then be used for classification, recognition, and reconstruction tasks and similar to classic 2D CNNs, they significantly outperform existing methods as shown in [100].

3.1.2 Network Architecture

The 3D shapes in [100] have relatively low resolution and are represented by $24 \times 24 \times 24$ voxels and 3 additional cells for padding in order to reduce convolution artifacts. The network architecture of 3D ShapeNets is depicted in Figure 3.1, where only one filter is visualized for each convolutional layer for simplicity. Layer 1 consists of 48 filters of size 6 with stride 2, layer 2 has 160 filters of size 5 with stride 2, layer 3 has 512 filters of size 4, and each convolution filter is connected to all feature channels in the previous layer. Layer 4 is a fully connected Restricted Boltzmann Machine (RBM) which consists of 1200 hidden units. Layer 5 consists of 4000 hidden units which takes as input a combination of multinomial label variables and Bernouilli feature variables.

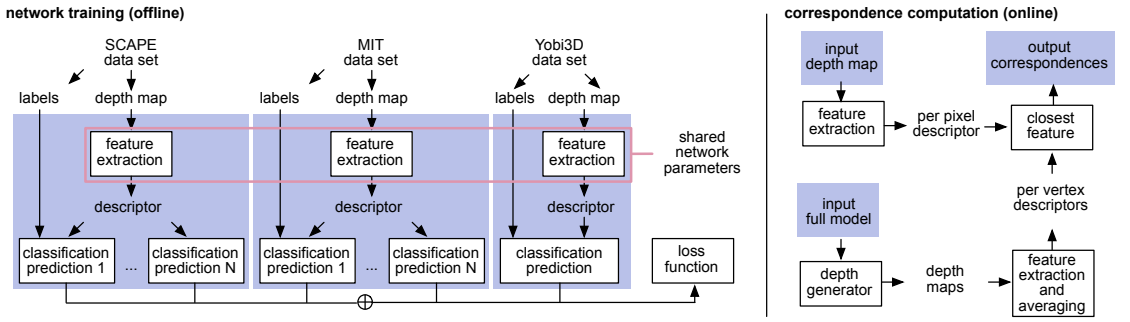


Figure 3.2: A neural network can be trained to extract a feature descriptor and predict the corresponding segmentation label on the human body surface for each point in the input depth maps. Per-vertex descriptors are generated for 3D models by averaging the feature descriptors in their rendered depth maps. The extracted features can then be used for example to compute dense correspondences.

3.1.3 Training and Data Collection

The model is first pre-trained layer-by-layer using a contrastive divergence approach [36], then fine-tuned using a wake sleep approach as detailed in [37]. In the pre-training phase of the first layer, learning signals are only collected in non-empty receptive fields. In this way we can minimize learning distraction in largely unoccupied cells for the RBM. The learned filters are shown in Figure 3.1. As with any deep neural network, a large collection of training data is required to effectively handle intra-class variations. Wu and colleagues [100] have introduced the database ModelNet which combines objects (vehicles, furniture, items, etc.) from 3D Warehouse, Yobi3D, Princeton Shape Benchmark, and the SUN database. Amazon Mechanical Turk is used to remove mis-labeled, unrealistic, duplicate, and irrelevant objects. The final database contains 151K 3D CAD models from 660 categories. During training, 40 common object categories are selected from ModelNet, each containing 100 unique CAD models. The data is augmented using a combination of rotation and translations resulting in 12 poses per object. Pre-training takes roughly 2 days on an Intel XEON E5-2690 CPU with an NVIDIA K40c GPU.

3.2 View-based CNNs

This section introduces a deep learning framework to compute high dimensional feature descriptors for classification tasks. We will show later that such classification network can be used for dense correspondence computation across full or partial human shapes. Such system is trained with depth maps of humans in arbitrary pose and with varying clothing. Categories of data other than clothed humans are also possible.

Traditional classification neural networks tend to separate the embedding of surface points lying in different but nearby classes. Thus, using such learned feature descriptors for correspondence matching between deformed surfaces often results in significant outliers at the segmentation boundaries. In order to alleviate this problem, the idea is to leverage repeated mesh segmentations to produce smoother embeddings into feature space. This technique maps shape points that are geodesically close on the surface of their corresponding 3D model to nearby points in the feature space. As a result, not only are outliers considerably reduced during deformable shape matching, but it can be also shown that the amount of training data can be drastically reduced compared to conventional

learning methods.

Given depth maps of two humans I_1, I_2, one important application is to determine which two regions $R_i \subset I_i$ of the depth maps come from corresponding parts of the body, and to find the correspondence map $\phi : R_1 \to R_2$ between them. The strategy for doing so is to formulate the correspondence problem first as a *classification* problem: first, a feature descriptor $\mathbf{f} : I \to R^d$, which maps each pixel in a *single* depth image to a feature vector, is learned. These feature descriptors are then used to establish correspondences across depth maps (see Figure 3.2).

3.2.1 Classification Network

The feature vector output of a CNN for the classification task of 3D objects should satisfy two properties:

1. $\mathbf{f}$ depends only on the pixel location on the human body, so that if two pixels are sampled from the same anatomical location on depth scans of two different humans, their feature vector should be nearly identical, irrespective of pose, clothing, body shape, and angle from which the depth image was captured;

2. $\|\mathbf{f}(p) - \mathbf{f}(q)\|$ is small when p and q represent nearby points on the human body, and large for distant points.

The literature takes two different approaches to enforcing these properties when learning descriptors using convolutional neural networks. *Direct* methods include in their loss functions terms penalizing failure of these properties (by using e.g. Siamese or triplet-loss energies). However, it is not trivial how to sample a dense set of training pairs or triplets that can all contribute to training [76]. *Indirect* methods instead optimize the network architecture to perform *classification*. The network consists of a descriptor extraction tower and a classification layer, and peeling off the classification layer after training leaves the learned descriptor network (for example, many applications use descriptors extracted from the second-to-last layer of the AlexNet). This approach works since classification networks tend to assign similar (dissimilar) descriptors to the input points belonging to the same (different) class, and thus satisfy the above properties implicitly. An indirect approach is taken, as the experiments suggest that an indirect method that uses an ensemble of classification tasks has better performance and computational efficiency.

Descriptor learning as ensemble classification. There are two challenges to learning a feature descriptor for depth images of human models using this indirect approach. First, the training data is heterogenous: between different human models, it is only possible to obtain a sparse set of key point correspondences, while for different poses of the same person, we may have dense pixel-wise correspondences (e.g., SCAPE [5]). Second, smoothness of descriptors learned through classification is not explicitly enforced. Even though some classes tend to be closer to each other than the others in reality, the network treats all classes equally.

To address both challenges, per-pixel descriptors can be learned for depth images by first training a network to solve a *group* of classification problems, using a single feature extraction tower shared by the different classification tasks. This strategy allows to combine different types of training data as well as designing classification tasks for various objectives. Formally, suppose there are M classification problems $C_i, 1 \leq i \leq M$. Denote the parameters to be learned in classification problem C_i as $(\mathbf{w}_i, \mathbf{w})$, where $\mathbf{w}_i$ and

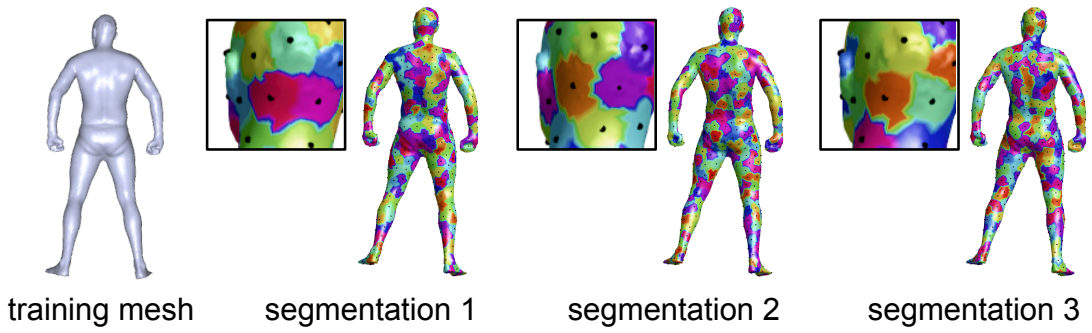


Figure 3.3: To ensure smooth descriptors, [97] defines a classification problem for multiple segmentations of the human body. Nearby points on the body are likely to be assigned the same label in at least one segmentation.

$\mathbf{w}$ are the parameters corresponding to the classification layer and descriptor extraction tower, respectively. The descriptor learning is defined as minimizing a combination of loss functions of all classification problems:

$$\{\mathbf{w}_i^\star\}, \mathbf{w}^\star = \arg\min_{\{\mathbf{w}_i\},\mathbf{w}} \sum_{i=1}^{M} \mathcal{L}(\mathbf{w}_i, \mathbf{w}). \tag{3.2}$$

After training, the optimized descriptor extraction tower becomes the output. It is easy to see that when $\mathbf{w}_i$, $\mathbf{w}$ are given by convolutional neural networks, Eq. (3.2) can be effectively optimized using stochastic gradient descent through back-propagation.

To address the challenge of heterogenous training sets, two types of classification tasks are included in this ensemble: one for classifying key points, used for iter-subject training where only sparse ground-truth correspondences are available, and one for classifying dense pixel-wise labels, e.g., by segmenting models into patches (See Figure 3.3), used for intra-subject training. Both contribute to the learning of the descriptor extraction tower.

To ensure descriptor smoothness, instead of introducing additional terms in the loss function, a simple yet effective strategy is proposed that randomizes the dense-label generation procedure. Specifically, as shown in Figure 3.3, multiple segmentations of the same person are considered, and a classification problem for each is introduced. Clearly, identical points will always be associated with the same label and far-apart points will be associated with different labels. Yet for other points, the number of times that they are associated with the same label is related to the distance between them. Consequently, the similarity of the feature descriptors are correlated to the distance between them on the human body resulting in a smooth embedding satisfying the desired properties discussed in the beginning of the section.

3.3 Human shape correspondence

The computation of correspondences between geometric shapes is a fundamental building block for many important tasks in 3D computer vision, such as reconstruction, tracking, analysis, and recognition. Temporally-coherent sequences of partial scans of an object can be aligned by first finding corresponding points in overlapping regions, then recovering the motion by tracking surface points through a sequence of 3D data; semantics can be

	0	1	2	3	4	5	6	7	8	9	10
layer	**image**	**conv**	**max**	**conv**	**max**	**2×conv**	**conv**	**max**	**2×conv**	**int**	**conv**
filter-stride	-	11-4	3-2	5-1	3-2	3-1	3-1	3-2	1-1	-	3-1
channel	1	96	96	256	256	384	256	256	4096	4096	16
activation	-	relu	lrn	relu	lrn	relu	relu	idn	relu	idn	relu
size	512	128	64	64	32	32	32	16	16	128	512
num	1	1	4	4	16	16	16	64	64	1	1

Table 3.1: The *end-to-end* network architecture generates a per-pixel feature descriptor and a classification label for all pixels in a depth map simultaneously. From top to bottom in column: The filter size and the stride, the number of filters, the type of the activation function, the size of the image after filtering and the number of copies reserved for up-sampling.

extracted by fitting a 3D template model to an unstructured input scan. With the popularization of commodity 3D scanners and recent advances in correspondence algorithms for deformable shapes, human bodies can now be easily digitized [53, 63, 26] and their performances captured using a single RGB-D sensor [52, 91].

Most techniques are based on robust non-rigid surface registration methods that can handle complex skin and cloth deformations, as well as large regions of missing data due to occlusions. Because geometric features can be ambiguous and difficult to identify and match, the success of these techniques generally relies on the deformation between source and target shapes being reasonably small, with sufficient overlap. While local shape descriptors [73] can be used to determine correspondences between surfaces that are far apart, they are typically sparse and prone to false matches, which require manual clean-up. Dense correspondences between shapes with larger deformations can be obtained reliably using statistical models of human shapes [5, 9], but the subject has to be naked [8]. For clothed bodies, the automatic computation of dense mappings [46, 54, 71, 16] has been demonstrated on full surfaces with significant shape variations, but are limited to compatible or zero-genus surface topologies.

The problem of estimating accurate dense correspondence between deformable partial shapes has been investigated recently in the works of Rodolà and colleagues [70, 20, 55], and has been the target of two recent SHREC'16 Correspondence benchmarks [19, 49]. Despite the recent interest, however, this problem has remained relatively unexplored in the literature due to the several challenges it entails.

Wei and colleagues [97] recently introduced a deep neural network structure for computing dense correspondences between shapes of clothed subjects in arbitrary complex poses. The input surfaces can be a full model, a partial scan, or a depth map, maximizing the range of possible applications. Their system is trained with a large dataset of depth maps generated from the human bodies of the SCAPE database [5], as well as from clothed subjects of the Yobi3D [2] and MIT [94] dataset. While all meshes in the SCAPE database are in full correspondence, they manually labeled the clothed 3D body models. They combine both training datasets and learned a global feature descriptor using a network structure that is well-suited for the unified treatment of different training data (bodies, clothed subjects). Similar to the unified embedding approach of FaceNet [76], the AlexNet [48] classification network can be used to learn distinctive feature vectors for different subregions of the human body. While the performance of this dense correspondence computation is comparable to state of the art techniques between two full models, they also demonstrate that learning shape priors of clothed subjects can yield highly accurate matches between partial-to-full and partial-to-partial shapes. Their examples include fully

clothed individuals in a variety of complex poses and the effectiveness of this approach has been demonstrated on a template based performance capture application that uses a single RGB-D camera as input.

3.3.1 Correspondence Computation

This trained classification network can be used to extract per-pixel feature descriptors for depth maps. For full or partial 3D scans, the authors first render depth maps from multiple viewpoints and compute a per-vertex feature descriptor by averaging the per-pixel descriptors of the depth maps. These descriptors are used to establish correspondences simply by a nearest neighbor search in the feature space (see Figure 3.2).

For applications that require deforming one surface to align with the other, one can fit the correspondences described in this chapter into any existing deformation method to generate the alignment. One possibility is to use the efficient as-rigid-as possible deformation model described in [52].

This section demonstrates the performance of deep learning-based correspondence computation evaluated on various real and synthetic datasets, naked and clothed subjects, as well as full and partial matching for challenging examples as illustrated in Figure 3.7. The real capture data examples (last column) are obtained using a Kinect One (v2) RGB-D sensor and demonstrate the effectiveness for real life scenarios. Each partial data is a single depth map frame with 512×424 pixels and the full template model is obtained using the non-rigid 3D reconstruction algorithm of [53]. All examples include complex poses (side views and bended postures), challenging garment (dresses and vests), and props (backpacks and hats).

Four different synthetic datasets are used to provide a quantitative evaluation. The 3D models from both SCAPE and MIT databases are part of the training data of the deep neural network described in the previous Sections, while the FAUST and Mixamo models [1] are not used for training. The SCAPE and FAUST data sets are exclusively naked human body models, while the MIT and Mixamo models are clothed subjects. For all synthetic examples, the partial scans are generated by rendering depth maps from a single camera viewpoint. The Adobe Fuse and Mixamo softwares [1] were used to procedurally model realistic characters and generate complex animation sequences through a motion library provided by the software.

The correspondence colorizations validate the accuracy, smoothness, and consistency of the dense matchings in extreme situations, including topological variations between source and target. Notice how the correspondences between front and back views are being correctly identified in the real capture 1 example for the full-to-partial matchings. Popular skeleton extraction methods from single-view 3D captures such as [78, 98, 92] often have difficulties resolving this ambiguity.

Comparisons. Surface matching techniques which are not restricted to naked human body shapes are currently the most suitable solutions for handling subjects with clothing. Though robust to partial input scans such as single-view RGB-D data, cutting edge non-rigid registration techniques [42, 52] often fail to converge for large scale deformations without additional manual guidance as shown in Figure 3.4. When both source and target shapes are full models, an automatic mapping between shapes with considerable deformations becomes possible as shown in [46, 54, 71, 16]. This method is compared with the recent work of Chen et al. [16] and computes correspondences between pairs of scans sampled from the same (intra-subject) and different (inter-subject) subjects. Chen et al. evaluate a rich set of methods on randomly sampled pairs from the FAUST database [9]

	intra AE	intra WE	inter AE	inter WE
Chen et al. (unsupervised)	4.49	10.96	5.95	14.18
Wei et al. (learning-based)	2.00	9.98	2.35	10.12

Table 3.2: Comparison between the method described in this Section and the recent work of Chen et al. [16] by computing correspondences for intra- and inter-subject pairs from the FAUST data set. We show the average error on all pairs (AE, in centimeters) and the average error on the worst pair for each technique (WE, in centimeters). While the learning-based technique may introduce worse WE, overall accuracies are improved in both cases.

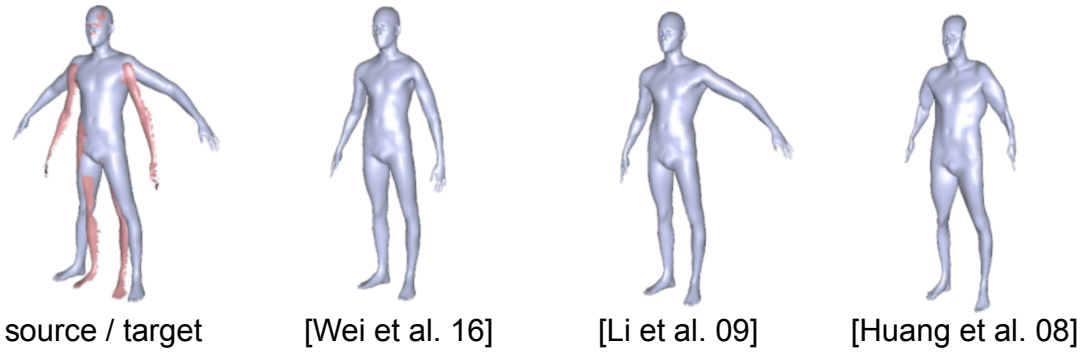


Figure 3.4: Compared to other non-rigid registration algorithms, larger deformations between a full template and a partial scan can be handled.

and report the state of the art results for their method. For a fair comparison, this method is also evaluated on the same set of pairs. As shown in Table 3.2, the learning-based method described here improves the average accuracy for both the intra- and the inter-subject pairs. Note that by using simple AlexNet structure, an average accuracy of 10 cm can be achieved. However, if multiple segmentations are not adapted to enforce smoothness, the worst average error can be up to 30 cm.

Limitations. Like any supervised learning approach, this framework cannot handle arbitrary shapes as the prior is entirely based on the class of training data. Despite superior performance compared to the state of the art, the current implementation is far from perfect. For poses and clothings that are significantly different than those from the training data set, this method still produces wrong correspondences. However, the outliers are often grouped together due to the enforced smoothness of the embedding, which could be advantageous for outlier detection. Due to the limited memory capacity of existing GPUs, this approach requires downsizing of the training input, and hence the correspondence resolutions are limited to 512×512 depth map pixels.

Performance. The shown experiments are performed on a 6-core Intel Core i7-5930K Processor with 3.9 GHz and 16GB RAM. Both offline training and online correspondence computation run on an NVIDIA GeForce TITAN X (12GB GDDR5) GPU. While the complete training of the neural network takes about 250 hours of computation, the extraction of all the feature descriptors never exceeds 1 ms for each depth map. The subsequent correspondence computation with these feature descriptors varies between 0.5 and 1 s, depending on the resolution of the input data.

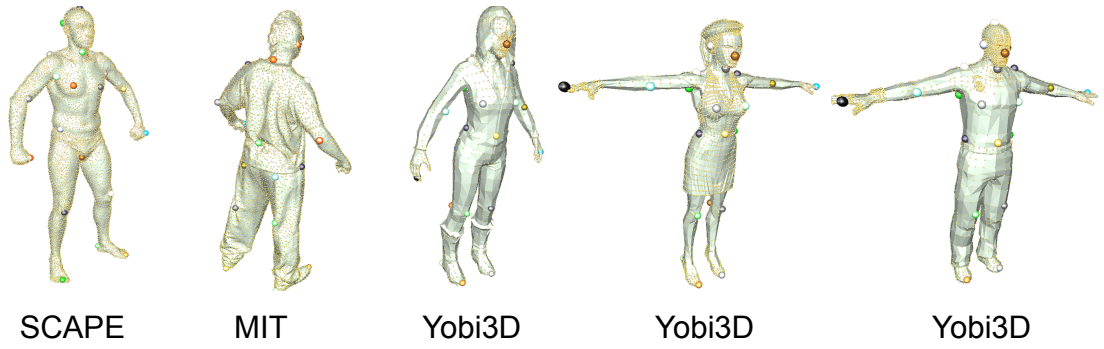


Figure 3.5: Sparse key point annotations of 33 landmarks across clothed human models of different datasets.

3.3.2 Training Data Generation

Collecting 3D Shapes. To generate the training data for the network, 3D models are collected from three major resources: the SCAPE [5], the MIT [94], and the Yobi3D [2] data sets. The SCAPE database provides 71 registered meshes of one person in different poses. The MIT dataset contains the animation sequences of three different characters. Similar to SCAPE, the models of the same person have dense ground truth correspondences. All animation sequences are used except for the *samba* and *swing* ones, which can be reserved for evaluation. Yobi3D is an online repository that contains a diverse set of 2000 digital characters with varying clothing. Note that the Yobi3D dataset covers the shape variability in local geometry, while the SCAPE and the MIT datasets cover the variability in pose.
Simulated Scans. Each model is rendered from 144 different viewpoints to generate training depth images. A depth image resolution of 512×512 pixels is used, where the rendered human character covers roughly half of the height of the depth image. This setup is comparable to those captured from commercial depth cameras; for instance, the Kinect One (v2) camera provides a depth map with resolution 512×424, where a human of height 1.7 meters standing 2.5 meters away from the camera has a height of around 288 pixels in the depth image.
Key-point annotations. Human experts are used to annotate 33 key points across the input models as shown in Figure 3.5. These key points cover a rich set of salient points that are shared by different human models (e.g. left shoulder, right shoulder, left hip, right hip etc.). Note that for shapes in the SCAPE and MIT datasets, only one rest-shape is annotated and the ground-truth correspondences are used to propagate annotations. The annotated key points are then propagated to simulated scans, providing 33 classes for training.
500-patch segmentation generation. Each distinctive model in the model collection is divided into multiple 500-patch segmentations. Each segmentation is generated by randomly picking 10 points on each model, and then adding the remaining points via furthest point-sampling. In total, 100 pre-computed segmentations are used. Each such segmentation provides 500 classes for depth scans of the same person (with different poses).

3.3.3 Network Design and Training

The neural network structure for training consists of a descriptor extraction tower and a classification module.

Extraction tower. The descriptor extraction tower takes a depth image as input and extracts for each pixel a dimension d ($d = 16$ in this paper) descriptor vector. A popular choice is to let the network extract each pixel descriptor using a neighboring patch (c.f.[33, 104]). However, such a strategy is too expensive in this setting as this has to be computed for dozens of thousands of patches per scan.

The strategy is to design a network that takes the entire depth image as input and simultaneously outputs a descriptor for each pixel. Compared with the patch-based strategy, the computation of patch descriptors are largely shared among adjacent patches, making descriptor computation fairly efficient in testing time.

Table 3.1 describes the proposed network architecture. The first 7 layers are adapted from the AlexNet architecture. Specifically, the first layer downsamples the input image by a factor of 4. This downsampling not only makes the computations faster and more memory efficient, but also removes salt-and-pepper noise which is typical in the output from depth cameras. Moreover, a similar strategy described in [77] is adapted to modify the pooling and inner product layers so that the original image resolution can be recovered through upsampling. The final layer performs upsampling by using neighborhood information in a 3-by-3 window. This upsampling implicitly performs linear smoothing between the descriptors of neighboring pixels. It is possible to further smooth the descriptors of neighboring pixels in a post-processing step, but as shown in the results, this is not necessary since the network is capable of extracting smooth and reliable descriptors.

Classification module. The classification module receives the per-pixel descriptors and predicts a class for each annotated pixel (i.e., either key points in the 33-class case or all pixels in the 500-class case). Note that one layer for each segmentation of each person is introduced in the SCAPE and the MIT datasets and one shared layer for all the key points. Similar to AlexNet, *softmax* is used when defining the loss function.

Training. The network is trained using a variant of stochastic gradient descent. Specifically, a task (i.e., key points or dense labels) is randomly picked for a random partial scan and fed into the network for training. If the task is dense labels, a segmentation is randomly selected among all possible segmentations. The network can be tuned with 200,000 iterations using a batch size of 128 key points or dense labels which may come from multiple datasets.

3.4 Performance Capture

The presented corrrespondence computation can be used for template based performance capture applications using a depth map sequence captured from a single RGB-D sensor. The complete geometry and motion is reconstructed in every sequence by deforming a given template model to match the partial scans at each incoming frame of the performance. Unlike existing methods [88, 52, 95, 91] which track a template using the previous frame, here the template model is always deformed from its canonical rest pose using the computed full-to-partial correspondences in order to avoid potential drifts. Deformation is achieved using the robust non-rigid registration algorithm presented in Li et al. [52], where the closest point correspondences are replaced with the ones obtained from the presented method. Even though the correspondences are computed independently in every frame, we observe a temporally consistent matching during smooth motions without enforcing temporal coherence as with existing performance capture techniques as shown in Figure 3.6. Since the deep learning framework does not require source and

target shapes to be close, it can effectively handle large and instantaneous motions. For the real capture data, we visualize the reconstructed template model at every frame and for the synthetic model we show the error to the ground truth.

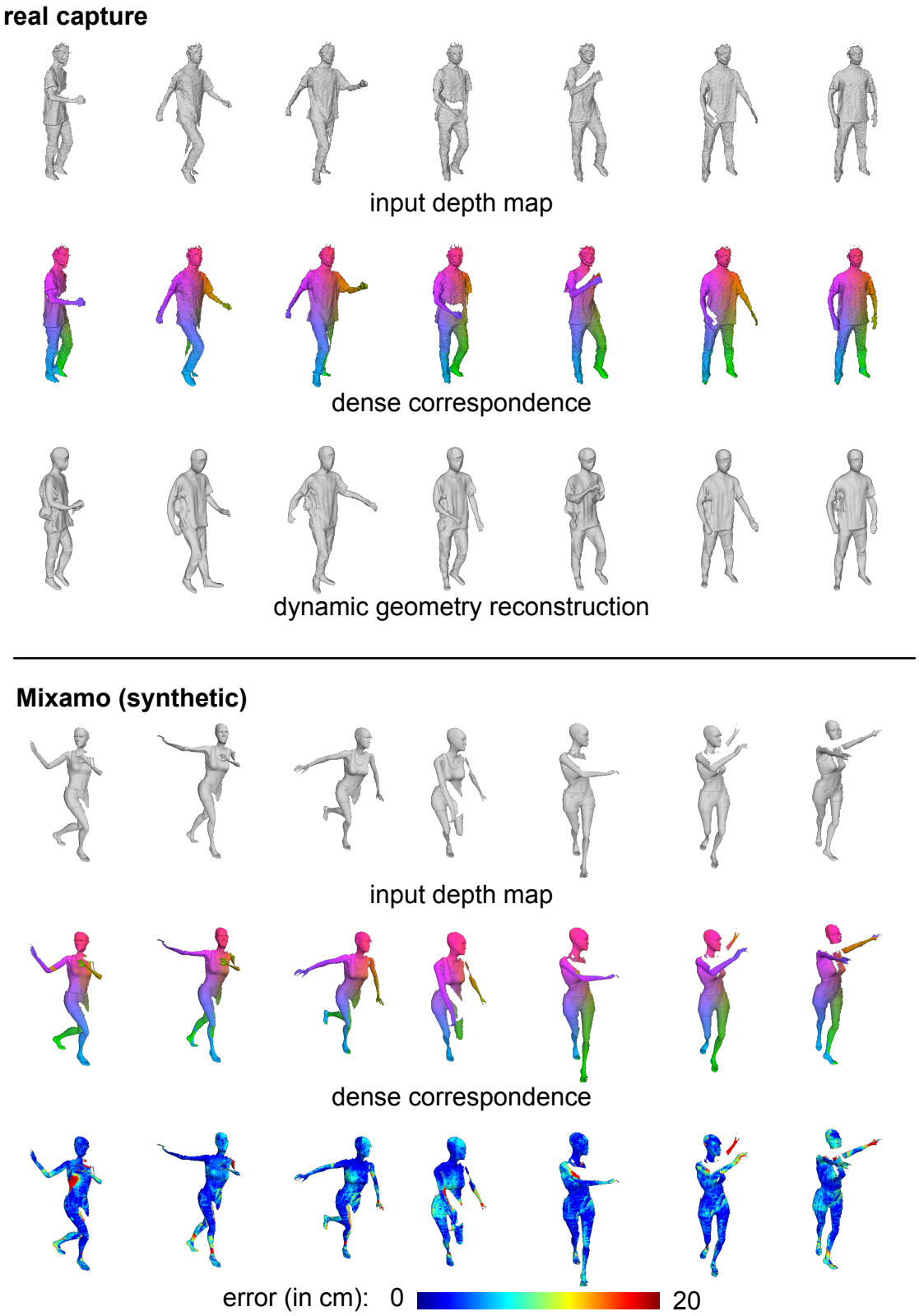


Figure 3.6: Geometry and motion reconstruction is performed by deforming a template model to captured data at each frame using the correspondences computed by the deep learning-based method. Even though temporal coherence is not enforced explicitly, we observe faithful and smooth reconstructions. Shown here are examples with both real and synthetic data.

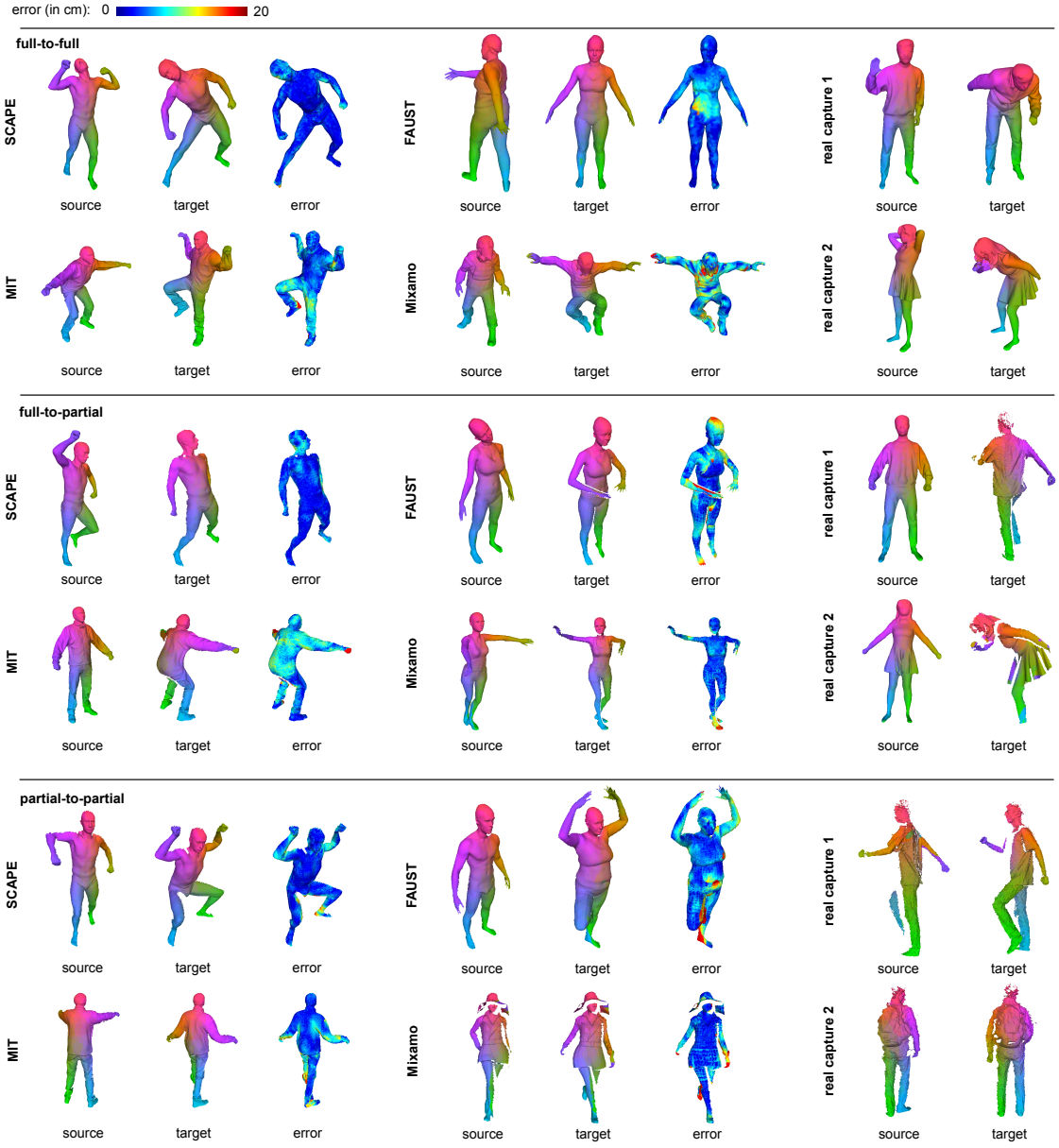


Figure 3.7: Deep learning-based correspondence computation can handle full-to-full, partial-to-full, and partial-to-partial matchings between full 3D models and partial scans generated from a single depth map. This method is evaluated on various real and synthetic datasets.

4

Spectral Learning Methods

4.1 Spectral analysis on manifolds

We model a 3D shape as a connected smooth compact two-dimensional manifold (surface) X, possibly with a boundary ∂X. Locally around each point x the manifold is homeomorphic to a two-dimensional Euclidean space referred to as the *tangent plane* and denoted by T_xX. The collection (disjoint union) of tangent spaces at all points is referred to as the *tangent bundle* and denoted by TX. A *Riemannian metric* is an inner product $\langle \cdot, \cdot \rangle_{T_xX} : T_xX \times T_xX \to \mathbb{R}$ on the tangent space depending smoothly on x.

It is important to note that the definition of a Riemannian manifold is completely abstract and does not require a geometric realization in any space. However, a Riemannian manifold can be realized as a subset of a Euclidean space (in which case it is said to be *embedded* in that space) by using the structure of the Euclidean space to induce a Riemannian metric. The *Nash Embedding Theorem* guarantees that any sufficiently smooth Riemannian manifold can be realized in a Euclidean space of sufficiently high dimension. An embedding is not necessarily unique; two different realizations of a Riemannian metric are called *isometries*.

Calculus on manifolds A *scalar field* is a smooth real function $f : X \to \mathbb{R}$ on the manifold. A *tangent vector field* $F : X \to TX$ is a mapping attaching a tangent vector $F(x) \in T_xX$ to each point x. We define the Hilbert spaces of scalar and vector fields on manifolds, denoted by $L^2(X)$ and $L^2(TX)$, respectively, with the following inner products:

$$\langle f, g \rangle_{L^2(X)} = \int_X f(x)g(x)dx; \tag{4.1}$$

$$\langle F, G \rangle_{L^2(TX)} = \int_X \langle F(x), G(x) \rangle_{T_xX} dx; \tag{4.2}$$

dx denotes here a volume element induced by the Riemannian metric.

In calculus, the notion of derivative describes how the value of a function changes with an infinitesimal change of its argument. One of the big differences distinguishing classical calculus from differential geometry is a lack of vector space structure on the manifold, prohibiting us from naïvely using expressions like $f(x + dx)$. The conceptual leap that is required to generalize such notions to manifolds is the need to work locally in the tangent space.

The *differential* of f as an operator $df : TX \to \mathbb{R}$ acting on tangent vector fields. At each point x, the differential can be identified with a linear form $df(x) = \langle \nabla_X f(x), \cdot \rangle_{T_xX}$ acting on tangent vectors $F(x) \in T_xX$, which model a small displacement around x. The change

of the function value as the result of this displacement is given by applying the form to the tangent vector, $df(x)F(x) = \langle \nabla_X f(x), F(x) \rangle_{T_x X}$, and can be thought of as an extension of the notion of the classical directional derivative. The operator $\nabla_X f : L^2(X) \to L^2(TX)$ is called the *intrinsic gradient*, and is similar to the classical notion of the gradient defining the direction of the steepest change of the function at a point. Similarly, the *intrinsic divergence* is an operator $\mathrm{div}_X F : L^2(TX) \to L^2(X)$ acting on tangent vector fields and (formal) adjoint to the gradient operator

$$\langle F, \nabla_X f \rangle_{L^2(TX)} = \langle -\mathrm{div}_X F, f \rangle_{L^2(X)}. \tag{4.3}$$

Finally, the *Laplace-Beltrami operator* (LBO) $\Delta_X : L^2(X) \to L^2(X)$ is an operator

$$\Delta_X f = -\mathrm{div}_X(\nabla_X f) \tag{4.4}$$

acting on scalar fields. From (4.3) it follows that the LBO is self-adjoint,

$$\langle \nabla_X f, \nabla_X f \rangle_{L^2(TX)} = \langle \Delta_X f, f \rangle_{L^2(X)} = \langle f, \Delta_X f \rangle_{L^2(X)}. \tag{4.5}$$

The LBO is *intrinsic*, i.e., expressible entirely in terms of the Riemannian metric. As a result, it is invariant to isometric (metric-preserving) deformations of the manifold.

Spectral analysis on manifolds The LBO of a compact manifold admits an eigendecomposition $\Delta_X \phi_k = \lambda_k \phi_k$ with a countable set of real eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \ldots$ and the corresponding eigenfunctions $\phi_1, \phi_2, \ldots$ form an orthonormal basis on $L^2(X)$. This basis is a generalization of the Fourier basis to non-Euclidean domains:[1] given a function $f \in L^2(X)$, it can be represented as the *Fourier series*

$$f(x) \quad = \quad \sum_{k \geq 1} \langle f, \phi_k \rangle_{L^2(X)} \phi_k(x), \tag{4.6}$$

where the analysis $\hat{f}_k = \langle f, \phi_k \rangle_{L^2(X)}$ can be regarded as the forward Fourier transform and the synthesis $\sum_{k \geq 1} \hat{f}_k \phi_k(x)$ is the inverse one; the eigenvalues $\{\lambda_k\}_{k \geq 1}$ play the role of frequencies.

Note that in the Euclidean case, the eigenfunctions of the 1D Laplacian takes the form $-\frac{d^2}{dx^2} e^{i\omega x} = \omega^2 e^{i\omega x}$, and the classical Fourier transform as the inner product between the signal $f(x)$ and the Laplacian eigenfunctions $e^{-i\omega x}$, i.e.

$$\hat{f}(\omega) = \langle f(x), e^{-i\omega x} \rangle_{L^2(\mathbb{R})} = \int_{-\infty}^{\infty} f(x) e^{-i\omega x} dx. \tag{4.7}$$

Heat diffusion on manifolds is governed by the *diffusion equation*,

$$\left(\Delta_X + \frac{\partial}{\partial t} \right) f(x,t) \quad = \quad 0; \tag{4.8}$$

$$f(x,0) \quad = \quad f_0(x), \tag{4.9}$$

where $f(x,t)$ denotes the amount of heat at point x at time t, $f_0(x)$ is the initial heat distribution; if the manifold has a boundary, appropriate boundary conditions must be

[1] It is easy to verify that the classical Fourier basis functions $e^{i\omega x}$ are eigenfunctions of the Euclidean Laplacian operator $-\frac{d^2}{dx^2} e^{i\omega x} = \omega^2 e^{i\omega x}$.

added. The solution of (4.8) is obtained by applying the *heat operator* $H^t = e^{-t\Delta_X}$ to the initial condition,

$$f(x,t) \quad = \quad H^t f_0(x) = \int_X f_0(x') h_t(x,x') dx', \tag{4.10}$$

Since H^t has the same eigenfunctions as Δ_X with the eigenvalues $\{e^{-t\lambda_k}\}_{k\geq 1}$, we can express the solution of (4.8) in the Fourier domain as

$$f(x,t) = \int_X f_0(x') \underbrace{\sum_{k\geq 1} e^{-t\lambda_k} \phi_k(x)\phi_k(x')}_{h_t(x,x')} dx', \tag{4.11}$$

where $h_t(x,x')$ is the *heat kernel*. Interpreting the LBO eigenvalues as 'frequencies', the coefficients $e^{-t\lambda}$ play the role of a transfer function corresponding to a low-pass filter sampled at $\{\lambda_k\}_{k\geq 1}$.

Discretization In the discrete setting, the surface X is sampled at n points $x_1, \ldots, x_n$. On these points, we construct a triangular mesh (V, E, F) with vertices $V = \{1, \ldots, n\}$, in which each interior edge $ij \in E$ is shared by exactly two triangular faces ikj and $jhi \in F$, and boundary edges belong to exactly one triangular face. The set of vertices $\{j \in V : ij \in E\}$ directly connected to i is called the *1-ring* of i. A real-valued function $f\colon X \to \mathbb{R}$ on the surface is sampled on the vertices of the mesh and can be identified with an n-dimensional vector $\mathbf{f} = (f(x_1), \ldots, f(x_n))^\top$. The discrete version of the LBO is given as an $n \times n$ matrix $\mathbf{L} = \mathbf{A}^{-1}\mathbf{W}$, where

$$w_{ij} = \begin{cases} (\cot \alpha_{ij} + \cot \beta_{ij})/2 & ij \in E; \\ -\sum_{k\neq i} w_{ik} & i = j; \\ 0 & \text{else}; \end{cases} \tag{4.12}$$

α_{ij}, β_{ij} denote the angles $\angle ikj, \angle jhi$ of the triangles sharing the edge ij, and $\mathbf{A} = \mathrm{diag}(a_1, \ldots, a_n)$ with $a_i = \frac{1}{3}\sum_{jk:ijk\in F} A_{ijk}$ being the local area element at vertex i and A_{ijk} denoting the area of triangle ijk [68].

The first $k \leq n$ eigenfunctions and eigenvalues of the LBO are computed by performing the generalized eigendecomposition $\mathbf{W\Phi} = \mathbf{A\Phi\Lambda}$, where $\mathbf{\Phi} = (\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_k)$ is an $n \times k$ matrix containing as columns the discretized eigenfunctions and $\mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_k)$ is the diagonal matrix of the corresponding eigenvalues.

4.2 Spectral descriptors

Heat Kernel Signature (HKS) Sun et al. [87] proposed a construction of intrinsic dense descriptors by considering the diagonal of the heat kernel,

$$h_t(x,x) \quad = \quad \sum_{k\geq 0} e^{-t\lambda_k} \phi_k^2(x), \tag{4.13}$$

also known as the *autodiffusivity function*. The physical interpretation of autodiffusivity is the amount of heat remaining at point x after time t. Geometrically, autodiffusivity 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. [87] defined the *heat kernel signature* (HKS) of dimension Q at point x by sampling the autodiffusivity function at some fixed times $t_1, \dots, t_Q$,

$$\mathbf{f}(x) \quad = \quad (h_{t_1}(x,x), \dots, h_{t_Q}(x,x))^\top. \tag{4.14}$$

The HKS has become a very popular approach in numerous applications due to several appealing properties. First, it is intrinsic and hence invariant to isometric deformations of the manifold by construction. Second, it is dense. Third, the spectral expression (4.13) of the heat kernel allows efficient computation of the HKS by using the first few eigenvectors and eigenvalues of the Laplace-Beltrami operator.

At the same time, 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).

Wave Kernel Signature (WKS) Aubry et al. [6] considered a different physical model of a quantum particle on the manifold, whose behavior is governed by the *Schrödinger equation*,

$$\left(i\Delta_X + \frac{\partial}{\partial t}\right)\psi(x,t) = 0, \tag{4.15}$$

where $\psi(x,t)$ is the complex wave function capturing the particle behavior. Assuming that the particle oscillates at frequency λ drawn from a probability distribution $\pi(\lambda)$, the solution of (4.15) can be expressed in the Fourier domain as

$$\psi(x,t) \quad = \quad \sum_{k\geq 1} e^{i\lambda_k t}\pi(\lambda_k)\phi_k(x). \tag{4.16}$$

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

$$p(x) \quad = \quad \lim_{T\to\infty}\int_0^T |\psi(x,t)|^2 dt = \sum_{k\geq 1}\pi^2(\lambda_k)\phi_k^2(x), \tag{4.17}$$

and depends on the initial frequency distribution $\pi(\lambda)$. Aubry et al. [6] considered a log-normal frequency distribution $\pi_\nu(\lambda) = \exp\left(\frac{\log\nu - \log\lambda}{2\sigma^2}\right)$ with mean frequency ν and standard deviation σ. They defined the Q-dimensional *wave kernel signature* (WKS)

$$\mathbf{f}(x) \quad = \quad (p_{\nu_1}(x), \dots, p_{\nu_Q}(x))^\top, \tag{4.18}$$

where $p_\nu(x)$ is the probability (4.17) corresponding to the initial log-normal frequency distribution with mean frequency ν, and $\nu_1, \dots, \nu_Q$ are some logarithmically-sampled frequencies.

While resembling the HKS in its construction and computation, WKS is based on log-normal transfer functions that act as band-pass filters and thus exhibits better spatial localization.

4.3 Optimal spectral descriptors

Litman and Bronstein [56] considered generic descriptors of the form

$$\mathbf{f}(x) = \sum_{k\geq 1}\boldsymbol{\tau}(\lambda_k)\phi_k^2(x) \approx \sum_{k=1}^{K}\boldsymbol{\tau}(\lambda_k)\phi_k^2(x) \tag{4.19}$$

where $\boldsymbol{\tau}(\lambda) = (\tau_1(\lambda), \ldots, \tau_Q(\lambda))^\top$ is a bank of transfer functions acting on LBO eigenvalues, and used parametric transfer functions

$$\tau_q(\lambda) = \sum_{m=1}^{M} a_{qm}\beta_m(\lambda) \tag{4.20}$$

in the B-spline basis $\beta_1(\lambda), \ldots, \beta_M(\lambda)$, where a_{qm} ($q = 1, \ldots, Q, m = 1, \ldots, M$) are the parametrization coefficients. Plugging (4.20) into (4.19) one can express the qth component of the spectral descriptor as

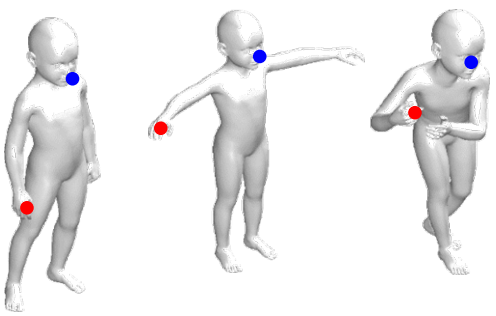
$$f_q(x) = \sum_{k\geq 1} \tau_q(\lambda_k)\phi_k^2(x) = \sum_{m=1}^{M} a_{qm} \underbrace{\sum_{k\geq 1} \beta_m(\lambda_k)\phi_k^2(x)}_{g_m(x)}, \tag{4.21}$$

where $\mathbf{g}(x) = (g_1(x), \ldots, g_M(x))^\top$ is a vector-valued function referred to as *geometry vector*, dependent only on the intrinsic geometry of the shape. Thus, (4.19) is parametrized by the $Q \times M$ matrix $\mathbf{A} = (a_{lm})$ and can be written in matrix form as $\mathbf{f}(x) = \mathbf{A}\mathbf{g}(x)$. The main idea of [56] is to *learn* the optimal parameters $\mathbf{A}$ by minimizing a task-specific loss which reduces to a Mahalanobis-type metric learning.

4.4 Deformable shape correspondence with random forests

In this Section we see how the classification forest paradigm (Section 2.1) can be employed to predict a dense correspondence between two given shapes M and N. The key idea, as introduced in [71], is to learn from examples a *canonical transformation*, i.e., a transformation from the points of an input shape (say M) to a canonical label set L. We will first show how to predict such a transformation from each individual shape M, N to the label set. A correspondence between M and N will be then obtained by a careful composition of the two mappings.

4.4.1 Inference



We start by describing the inference (or testing) step. In the context of shape matching, a decision tree routes a point $x \in M$ along the tree and to a leaf node, where a probability distribution defined on a discrete label set L is assigned to the point. Each label $\ell \in L$ identifies a tuple of corresponding points from the collection of training shapes; in the inset figure, points having the same color are associated to the same label in L. This way, the collection of probability distributions over L, which are predicted for each point $x \in M$, can be interpreted as defining a *soft map* from shape M to some reference shape from the training set. Note that such a reference is not a specific shape from the collection, but rather an abstraction that allows us to think of the label space as a physical object. We will discuss this aspect with more detail in Section 4.4.3.

According to this inference procedure, each tree $t \in \mathcal{F}$ of a forest $\mathcal{F}$ provides a posterior probability $\mathrm{P}(\ell|x, t)$ of label $\ell \in L$, given a point $x \in M$. The prediction of the whole forest $\mathcal{F}$ can be obtained by averaging the predictions of the single trees as $\mathrm{P}(\ell|x, \mathcal{F}) = \frac{1}{|\mathcal{F}|}\sum_{t\in\mathcal{F}} \mathrm{P}(\ell|x, t)$.

4.4.2 Learning

During the learning phase, the structure of the trees, the split functions and the leaf posteriors are determined from a training set. The latter consists of a collection of shapes with point-wise ground truth matches among them; for the sake of simplicity, we assume that all shapes have the same number of points, such that for each shape R_i in the training set we have a bijective mapping (or canonical transformation) $T_i : R_i \to L$. The training set of labelled data is then given by $\{(x, T_i(x)) \mid x \in R_i\}_i$. It remains to define the label predictions associated to each leaf, and the test functions associated to the interior nodes of the forest.

A straightforward way to assign a label distribution to each leaf node is to measure, for each label $\ell \in L$, the proportion of training samples (x, ℓ) among all training samples $\mathbb{S}$ that have reached the leaf:

$$\mathrm{P}(\ell|\mathbb{S}) = \frac{|\{(x,\ell) \in \mathbb{S}\}|}{|\mathbb{S}|} . \tag{4.22}$$

The probability distribution $\mathrm{P}(\cdot|\mathbb{S})$ will thus become the posterior probability during inference for every shape point reaching the leaf.

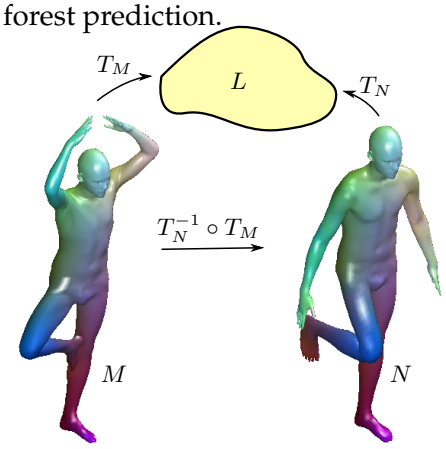
As splitting functions for the interior nodes, Rodolà et al. [71] proposed to consider binary tests of the form $f_\Theta(x) > \tau$, where function $f_\Theta(x)$ is a local shape descriptor computed at $x \in M$ and parametrized by Θ, and τ is a randomly chosen threshold. Since the parameters Θ are optimized during training, the idea is to consider an existing descriptor but let the forest automatically determine its discriminative features based on the training examples. The baseline descriptor can be chosen depending on the matching problem at hand. The WKS was considered in [71] for classical shape matching, while the HKS was used in [19, 49] due to its better resilience to missing shape parts.

4.4.3 Shape matching via the label space

Once the random forest is fixed, the inference step for a given test shape M provides us with a mapping $T : M \to \mathrm{P}(L)$. However, as briefly mentioned in Section 4.4.1, such a mapping carries no geometric meaning by itself. The transformation can be endowed with a meaningful interpretation if we consider some reference shape R_i from the training set, for which we know the bijection $T_i : R_i \to L$, and instead consider the soft map $(T_i^{-1} \circ T) : M \to \mathrm{P}(R)_i$. We can now interpret the forest prediction as a probabilistic correspondence between the input test shape and a reference shape from the training set. This construction is useful, for example, to visualize the quality of the labeling and was utilized in [71] to devise a regularization step for the forest prediction.

In a typical matching problem, we are given two shapes M and N among which we seek to establish a dense correspondence. One is then interested in $\mathrm{P}(y|x)$, the probability that a point $x \in M$ corresponds to a point $y \in N$. Using the law of total probability, we can write

$$\mathrm{P}(y|x) = \sum_{\ell \in L} \mathrm{P}(\ell|x) \cdot \mathrm{P}(y|\ell) . \tag{4.23}$$



Note that this calculation is very simple to carry out in practice. Let $\mathbf{X}_M$, $\mathbf{X}_N$ denote two matrices containing the label predictions for the two shapes, i.e., for each point $x_i \in M$ and each

label $\ell \in L$ the probability $\mathrm{P}(\ell|x_i)$ is given by $(\mathbf{X}_M)_{\ell i}$ and similarly for N. Since $\mathbf{X}_M$ and $\mathbf{X}_N$ are left-stochastic matrices, (4.23) can be written as

$$\mathrm{P}(y|x) = \sum_{\ell \in L} (\mathbf{X}_N)_{j\ell} (\mathbf{X}_M)_{\ell i} = \tilde{\mathbf{X}}_N^\top \mathbf{X}_M \,, \tag{4.24}$$

where $\tilde{\mathbf{X}}_N^\top$ denotes column normalization after transpose. In other words, it stores the probabilities of a point $y_j \in N$ being the pre-image of a label $\ell \in L$:

$$(\tilde{\mathbf{X}}_N^\top)_{j\ell} = \mathrm{P}(y|\ell) = \frac{\mathrm{P}(\ell|y)}{\sum_{y \in N} \mathrm{P}(\ell|y)} \,.$$

The resulting matrix $\tilde{\mathbf{X}}_N^\top \mathbf{X}_M$ can now be interpreted as a soft map between the two input shapes, and it can be processed in different ways depending on the specific application.

5

Intrinsic Convolutional Neural Networks

We have seen in Chapter 3 that deep learning methods, in particular, convolutional neural architectures, could be applied to geometric data that is treated as a Euclidean 3D object (volume or range image). An alternative way of treating geometric data intrinsically as manifolds, was discussed in Chapter 4. The main goal of this chapter is to generalize convolutional neural networks to this latter setting. In particular, we will focus on the intrinsic definition of the convolution operation.

5.1 Intrinsic CNNs in the spectral domain

A fundamental result of classical Euclidean signal processing, states that the Fourier transform diagonalizes the convolution operator: the convolution $f \star g$ of two functions in the spectral domain can be expressed as the element-wise product of their Fourier transforms (4.7),

$$\widehat{(f \star g)}(\omega) = \hat{f}(\omega) \cdot \hat{g}(\omega). \tag{5.1}$$

Property (5.1) can be used to define convolution on non-Euclidean domains in the spectral domain as

$$(f \star g)(x) = \sum_{k \geq 0} \langle f, \phi_k \rangle_{L^2(X)} \langle g, \phi_k \rangle_{L^2(X)} \phi_k(x). \tag{5.2}$$

Such an operation can be interpreted as a non-linear filtering of the signal f. The key difference from the classical convolution is the lack of shift-invariance, which makes the filter kernel to change depending on its position.

5.1.1 Spectral CNNs

In [15], the spectral definition of convolution (5.2) was used to generalized CNNs to graphs, by representing filters in the spectral domain. In our context, the fundamental drawbacks of this formulation is its limitation to a single given domain, since the spectral representation of the filters is *basis dependent*. It implies that 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 5.1).

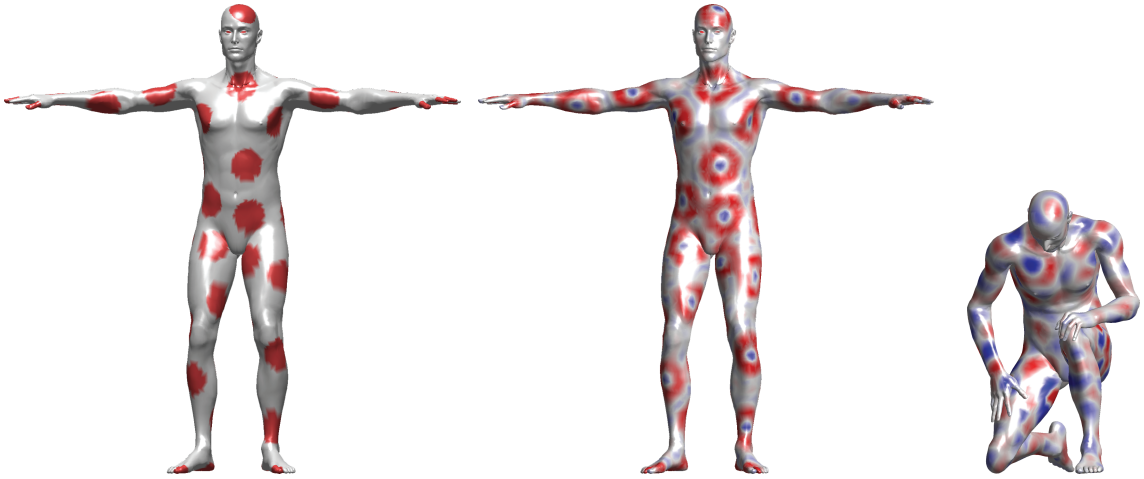


Figure 5.1: An illustration of the poor generalization of spectral filtering across non-Euclidean domains. Left: a function defined on a manifold; middle: result of the application of a filter in the frequency domain on the same manifold; right: the same filter applied on the same function but on a different (nearly-isometric) domain produces a completely different result.

5.1.2 Localized spectral CNNs

One of the key drawbacks of the Fourier transformation is the fact that it is global (in other words, the basis functions have a global support). A localized or *windowed Fourier transform* (WFT) is used in signal processing for local space-frequency analysis of signal. The main idea of the WFT is to analyze the frequency content of a signal that is localized by means of multiplication by a window. Given a function $f \in L^2(\mathbb{R})$ and some 'mother window' g localized at zero, the classical WFT is defined as

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

Alternatively, it can be defined as an inner product with a translated and modulated window,

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

where T_x and M_ω denotes the translation and modulation operator respectively. The translated and modulated window $M_\omega T_x g$ is referred to as the WFT *atom*. In the Euclidean setting the translation operator is defined simply as $(T_{x'}f)(x) = f(x-x')$, while the modulation operator is a multiplication by a Laplacian eigenfunction $(M_\omega f)(x) = e^{i\omega x}f(x)$, which amounts to a translation in the frequency domain $\widehat{(M_{\omega'}f)} = \hat{f}(\omega-\omega')$.

The notion of translation to a point x', $x-x'$, is not well defined in the non-Euclidean setting. In [79, 10] these operations were defined in the frequency domain. Translation to x' is replaced by convolution (5.2) with a delta-function centered at x', yielding

$$\begin{aligned}(T_{x'}f)(x) = (f\star\delta_{x'})(x) &= \sum_{k\geq 1}\langle f,\phi_k\rangle_{L^2(X)}\langle\delta_{x'},\phi_k\rangle_{L^2(X)}\phi_k(x)\\ &= \sum_{k\geq 1}\langle f,\phi_k\rangle_{L^2(X)}\phi_k(x')\phi_k(x).\end{aligned}$$

Note that such a translation is not shift-invariant in general, i.e., the window would change when moved around the manifold (see Figure 5.2). The modulation operator is defined as

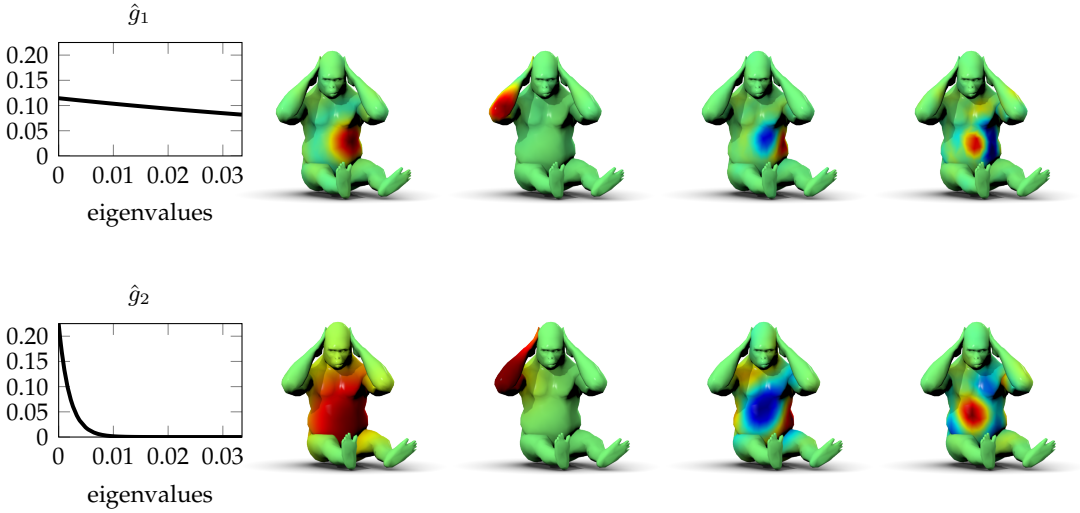


Figure 5.2: Examples of different WFT atoms $g_{x,k}$ using different windows (top and bottom rows; window Fourier coefficients are shown on the left), shown in different localizations (second and third columns) and modulations (fourth and fifth columns).

$(M_k f)(x) = \phi_k(x) f(x)$, where ϕ_k is the kth eigenfunction of the Laplace-Beltrami operator. Combining the two operators together, the WFT atom (see examples in Figure 5.2) becomes

$$g_{x',k}(x) = (M_k T_{x'} g)(x) = \phi_k(x) \sum_{i \geq 1} \hat{g}_i \, \phi_i(x) \phi_i(x').$$

Note that the 'mother window' is defined here in the frequency domain by the coefficients $\hat{g}_i$. Finally, the WFT of a signal $f \in L^2(X)$ can be defined as

$$(Sf)(x', k) = \langle f, g_{x',k} \rangle_{L^2(X)} = \sum_{i \geq 1} \hat{g}_i \, \phi_i(x') \langle f, \phi_i \phi_k \rangle_{L^2(X)}. \tag{5.3}$$

The WFT $(Sf)(x, k)$ performs a filtering of the signal f at the point x at the frequency k. By collecting its behavior over different frequencies, the content of the signal f in a local support around x is extracted, reproducing in this way the window extraction on images. The *localized spectral convolution layer* can thus be defined as

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

where $\mathbf{f}_{l'}$, $l' = 1, \ldots, p$ is the input signal, $\mathbf{W} = (w_{l,k,l'})$ is a $p \times K \times q$ tensor representing the learnable weights, and $\mathbf{g}_l$, $l = 1, \ldots, q$ is the output signal. An additional degree of freedom is the possibility to learn the window itself as well.

A drawback of this approach is its memory and computation requirements, as each window $\hat{g}_i$ in Equation (5.3) needs to be explicitly produced.

5.2 Intrinsic CNNs in the spatial domain

An alternative definition of an intrinsic equivalent of convolution is in the spatial domain. A classical convolution can be thought of as a template matching with filter, operating as a sliding window: e.g. in an image, one extracts a patch of pixels, correlates it with a

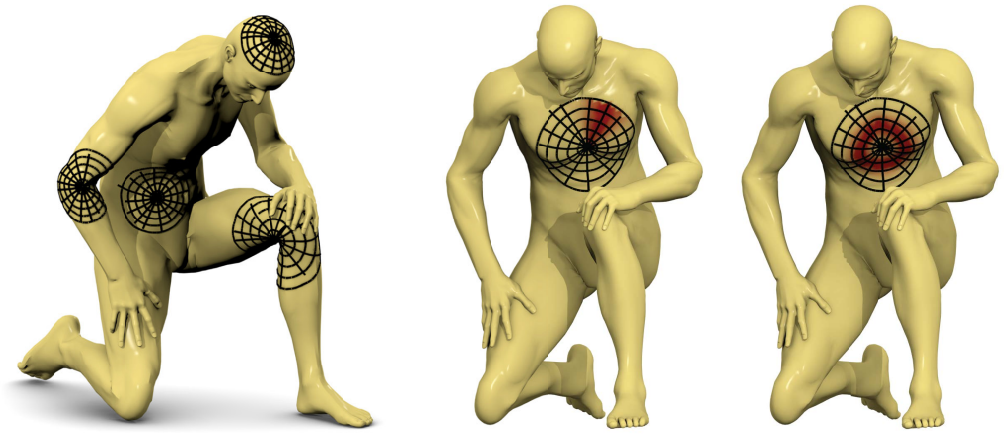


Figure 5.3: Construction of local geodesic polar coordinates on a manifold. Left: examples of local geodesic patches, center and right: example of angular and radial weights, respectively (red denotes larger weights).

template, and moves the window to the next position. In the non-Euclidean setting, the lack of shift-invariance makes the patch extraction operation position-dependent. The *patch operator* $D_j(x)$ acting on the point $x \in X$ can be defined as a re-weighting of the input signal f by means of some weighting kernels $\{w_i(x,\cdot)\}_{i=1,\dots,J}$ spatially localized around x, i.e.

$$D_j(x)f = \int_X f(x')w_j(x,x')dx', \quad j = 1,\dots,J. \tag{5.4}$$

The *intrinsic convolution* can be defined as

$$(f \star g)(x) = \sum_j g_j D_j(x) f, \tag{5.5}$$

where g_j denotes the filter coefficients applied on the patch extracted at each point. Different spatial-domain intrinsic convolutional layers amounts for a different definition of the patch operator D. In the following we will see two examples.

5.2.1 Geodesic CNNs

In [59], the authors 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 the shape X, the local 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 ρ-level sets $\{x' : d_X(x,x') = \rho\}$ of the geodesic distance function d_X for $\rho \in [0, \rho_0]$, where ρ_0 is the radius of the geodesic disc. The angular coordinate $\theta(x)$ is constructed as a set of equispaced geodesics $\Gamma(x,\theta)$ emanating from x in direction θ in a way that they are perpendicular to the geodesic distance level sets.

Once the local geodesic system of coordinates is extracted, the geodesic patch operator is defined as

$$(D(x)f)(\theta,\rho) = \int_X f(x')w_\theta(x,x')w_\rho(x,x')dx'; \tag{5.6}$$

$$w_\theta(x,x') = e^{-d_X^2(\Gamma(x,\theta),x')/2\sigma_\theta^2}; \tag{5.7}$$

$$w_\rho(x,x') = e^{-(d_X(x,x')-\rho)^2/2\sigma_\rho^2}, \tag{5.8}$$

Figure 5.4: Visualization of different heat kernels (red represent high values). Leftmost: example of an isotropic heat kernel. Remaining: examples of anisotropic heat kernels for different rotation angles θ and anisotropy coefficient α.

where w_θ, w_ρ are the angular and radial weights, respectively (see Figure 5.3 center and right). Note that the choice of the origin of the angular coordinate is arbitrary, and therefore it can vary from point to point. To overcome this problem, an *angular max pooling* was used in [59], leading to the following definition of the *geodesic convolution*

$$(f \star w)(x) = \max_{\Delta\theta \in [0,2\pi)} \int w(\theta + \Delta\theta, \rho)(D(x)f)(\theta, \rho)\, d\theta d\rho, \tag{5.9}$$

5.2.2 Anisotropic diffusion CNNs

In section 4.1, we saw how the heat propagation on a shape X is governed by the heat diffusion equation (4.8). In particular, given as initial heat distribution a delta function centered on x, the heat distribution on X after some time t is represented by the heat kernel $h_t(x, \cdot)$. Such a heat kernel is *isotropic*, i.e., diffuses heat equally in all directions (Figure 5.4, left). The diffusion strength is controlled by the diffusion time t.

A more general *anisotropic* heat diffusion is described by the anisotropic diffusion equation,

$$f_t(x,t) = -\text{div}(\mathbf{A}(x)\nabla f(x,t)), \tag{5.10}$$

where the thermal conductivity matrix $\mathbf{A}(x)$ specifies the heat conductivity properties at each point on the shape X. In particular, $\mathbf{A}(x)$ contains both positional and directional information. This more general diffusion model was considered in [4] for shape analysis tasks. In [11], the authors defined the thermal conductivity matrix as

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

where the matrix $R_\theta(x)$ performs rotation of θ w.r.t. to some reference (e.g. the maximum curvature) direction and $\alpha > 0$ is a parameter controlling the degree of anisotropy ($\alpha = 1$ corresponds to the classical isotropic case).

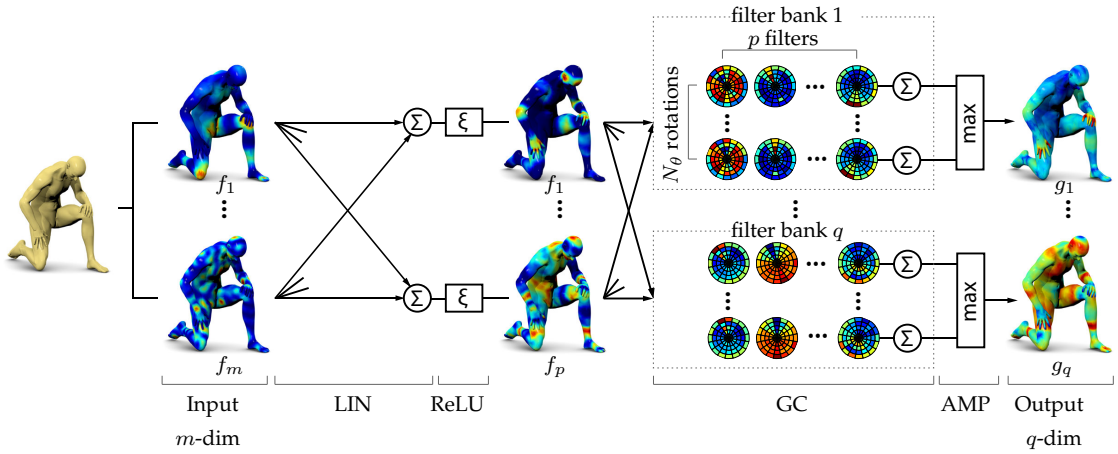


Figure 5.5: A simple example of an intrinsic convolutional neural network architecture. The network takes as input an off-the-shelf m-dimensional hand-crafted local shape descriptor for each vertex, then a linear dimensionality reduction layer is applied to reduce the input dimension to $p < m$, followed by a ReLU non-linearity. Finally, a geodesic convolution layer (2.6) 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.

Analogously to Equation (4.11), the *anisotropic heat kernel* is given by

$$h_{\alpha\theta t}(x,x') = \sum_{k\geq 0} e^{-t\lambda_{\alpha\theta,k}} \phi_{\alpha\theta,k}(x)\phi_{\alpha\theta,k}(x'),$$

where $\phi_{\alpha\theta,k}(x), \lambda_{\alpha\theta,k}$ are the eigenfunctions and eigenvalues of the anisotropic Laplacian $\Delta_{\alpha\theta} = -\mathrm{div}(\mathbf{A}_{\alpha\theta}(x)\nabla)$. The anisotropic heat kernel $h_{\alpha\theta t}$ depends on two additional parameters, the coefficient α and the rotation angle θ. Figure 5.4 shows some examples of anisotropic heat kernels computed at different rotations θ and anisotopies α. In [11], such kernels were used as the weighting functions for the construction of patch operator (5.4),

$$(D(x)f)(\theta,t) = \int_X h_{\alpha\theta t}(x,x')f(x')dx',$$

mapping the values of f around point x to a local polar-like system of coordinates (θ,t).

5.3 Applications

Similarly to the Euclidean CNNs, an *intrinsic convolutional neural network* consists of several layers that are applied subsequently, i.e. the output of the previous layer is used as the input into the subsequent one. The convolutional layer (2.6) is used with the only difference that the convolution operation is replaced by an intrinsic analogy. Figure 5.5 shows a toy example of an intrinsic CNN architecture with one intrinsic convolutional layer.

Intrinsic CNN is a non-linear hierarchical parametric map of the form

$$\Psi_\Theta = \psi^n_{\theta_n} \circ \cdots \circ \psi^2_{\theta_2} \circ \psi^1_{\theta_1},$$

where ψ_i, $i = 1,\dots,n$, represents the ith layer with parameters θ_i. The parameters of the model $\Theta = \{\theta_i : i = 1,\dots,n\}$ are the set of all the parameters of each layer. The function

is applied to point-wise input data (e.g. some simple geometric descriptors) and produces some point-wise output. In the following, we will see how intrinsic CNNs can be applied to two basic problems in computer graphics: the computation of local descriptors and correspondences.

5.3.1 Local descriptors

The result of applying an intrinsic CNN in a point-wise manner on some input feature vector $f(x)$ is a feature map $g(x)$ that can be regarded to as a dense local descriptor at point x. Ideally, a local descriptor should be as similar as possible at corresponding points (positives) across a collection of shapes, and as dissimilar as possible at non-corresponding points (negatives). Learning optimal descriptors is possible using the *siamese architecture* [14], composed of two identical copies of the same intrinsic CNN model sharing the same parameterization and fed by pairs of knowingly similar or dissimilar samples, where the training procedure tries to minimize the siamese loss

$$\mathcal{L}(\Theta) = (1-\gamma)\sum_{i=1}^{|\mathcal{T}_+|}\|\Psi_\Theta(f_i) - \Psi_\Theta(f_i^+)\|^2 + \gamma\sum_{i=1}^{|\mathcal{T}_-|}(\mu - \|\Psi_\Theta(f_i) - \Psi_\Theta(f_i^-)\|)_+^2.$$

Here, $\lambda \in [0,1]$ is a parameter trading off between the positive and negative losses, μ is a margin, $(\cdot)_+ = \max\{0,\cdot\}$ and $\mathcal{T}_\pm = \{(f_i, f_i^\pm)\}$ denotes the sets of positive and negative pairs, respectively.

In [59], the authors used the geodesic CNN architecture shown in Figure 5.5 to produce dense intrinsic pose- and subject-invariant descriptors on human shapes. A qualitative evaluation of the goodness of the learned descriptors is reported in Figure 5.6, where the Euclidean distance in the descriptor space between the descriptor at a selected point and the rest of the points on the same shape as well as its transformations is depicted. The descriptors produced by the geodesic CNN (GCNN) manifest both good localization (better than HKS) and are more discriminative (less spurious minima than WKS and OSD), as well as robustness to different kinds of noise, including isometric and non-isometric deformations, geometric and topological noise, different sampling, and missing parts.

5.3.2 Correspondence

As we discussed in Section 4.4.3 of these notes, finding the correspondence in a collection of shapes can be posed as a labelling problem, where one tries to label each vertex of a given query shape X with the index of a corresponding point on some common reference shape Y [71]. Let n and m denote the number of vertices in X and Y, respectively. For a point x on a query shape, the output of an intrinsic CNN $\Psi_\Theta(x)$ is m-dimensional and is interpreted as a probability distribution ('soft correspondence') on Y. The output of the network at all the points of the query shape can be arranged as an $n \times m$ matrix with elements of the form $\psi_\Theta(x,y)$, representing the probability of x mapped to y.

Let us denote by $y^*(x)$ the ground-truth correspondence of x on the reference shape. We assume to be provided with examples of points from shapes across the collection and their ground-truth correspondence, $\mathcal{T} = \{(x, y^*(x))\}$. The optimal parameters of the network are found by minimizing the multinomial regression loss

$$\mathcal{L}_{\text{reg}}(\Theta) = -\sum_{(x,y^*(x)\in\mathcal{T})} \log \psi_\Theta(x, y^*(x)),$$

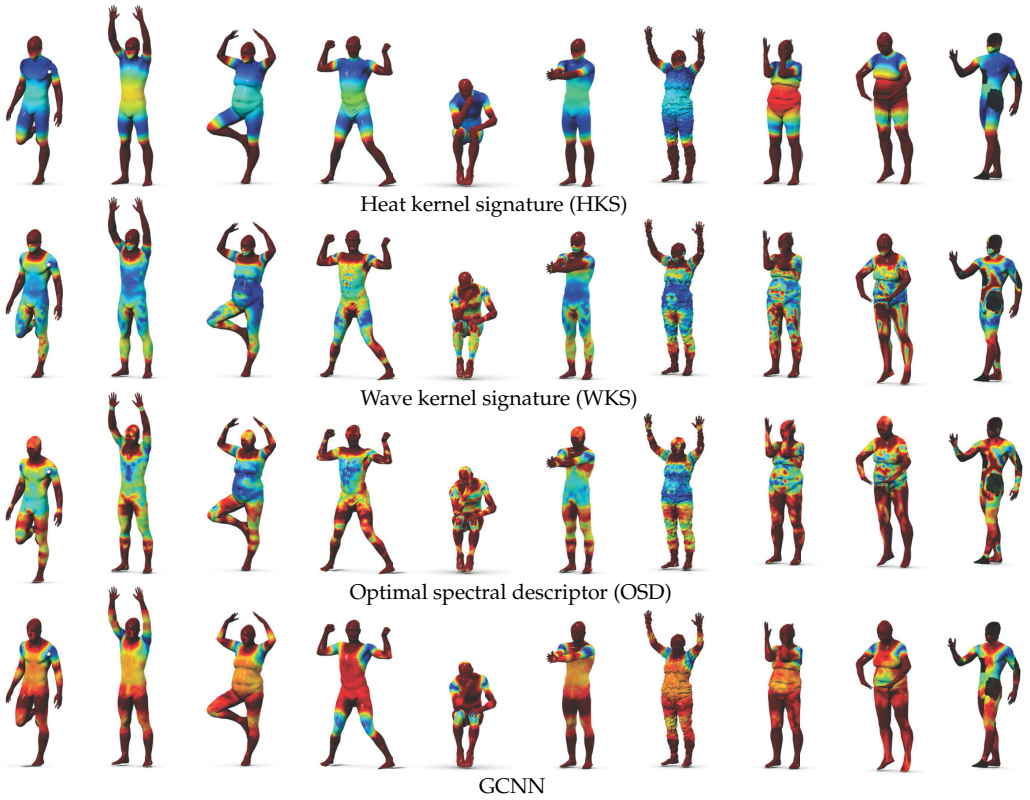


Figure 5.6: Normalized Euclidean distance between the descriptor at a reference point on the shoulder (white sphere) and the descriptors computed at the rest of the points for different transformations (shown left-to-right: near isometric deformations, non-isometric deformations, topological noise, geometric 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 corresponding point and no spurious local minima at other locations.

Figure 5.7: Examples of correspondence on the FAUST humans dataset obtained by the anisotropic diffusion CNN. Shown is the texture transferred from the leftmost reference shape to different subjects in different poses by means of our correspondence. The correspondence is nearly perfect (only very few minor artifacts are noticeable).

which represents the Kullback-Leibler divergence between the probability distribution produced by the network and the ground-truth distribution $\delta_{y^*}(x)$. Figures 5.7, 5.8 and 5.9 show a qualitative evaluation of the quality of the correspondences learned by anisotropic diffusion CNN.

Figure 5.8: Pointwise geodesic error (in % of geodesic diameter) of different correspondence methods (top to bottom: Blended intrinsic maps, geodesic CNN, anisotropic diffusion CNN) on the FAUST dataset. For visualization clarity, the error values are saturated at 10% of the geodesic diameter. Hot colors correspond to large errors. Note the different behavior of different approaches: BIM produces large distortions with very few accurate matches; geodesic CNN produces many near-perfect matches but also many matches with large distortion; anisotropic diffusion CNN produces very few matches with large distortion and many near-perfect matches.

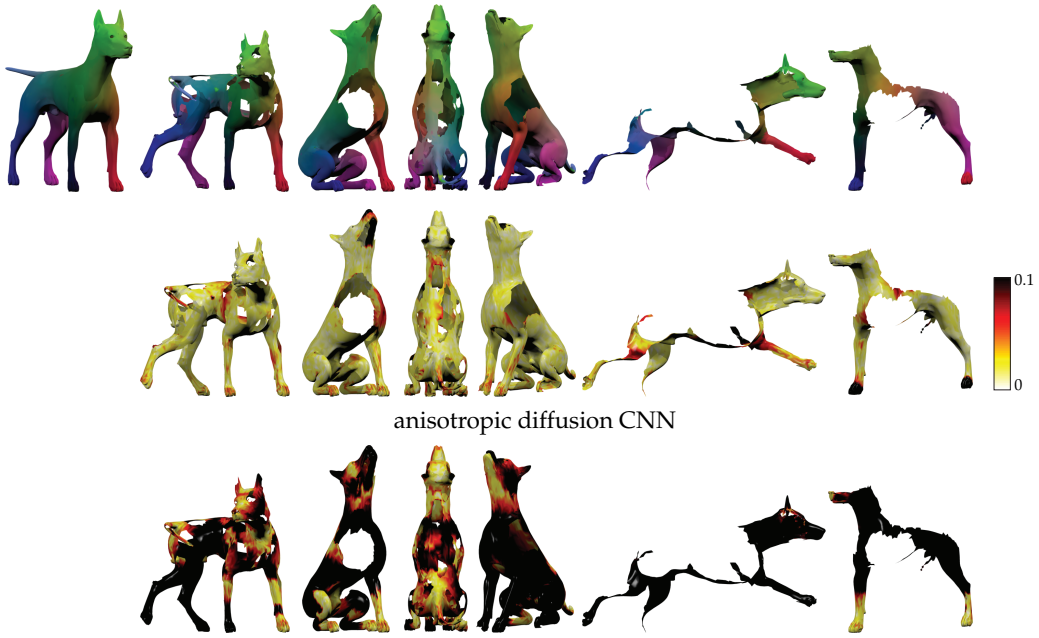


Figure 5.9: Examples of partial correspondence on the dog shape from the SHREC'16 Partial (holes) dataset. First row: correspondence produced by anisotropic diffusion CNN. Corresponding points are shown in similar color. Reference shape is shown on the left. Second and third rows: pointwise geodesic error (in % of geodesic diameter) of the anisotropic diffusion CNN and RF correspondence, respectively. For visualization clarity, the error values are saturated at 10% of the geodesic diameter. Hot colors correspond to large errors.

Bibliography

[1] 3D Animation Online Services, 3D Characters, and Character Rigging - Mixamo. https://www.mixamo.com/. Accessed: 2015-10-03.

[2] Yobi3D - free 3D model search engine. https://www.yobi3d.com. Accessed: 2015-11-03.

[3] I. Aizenberg, N. Aizenberg, and J. Vandewalle. *Multi-valued and universal binary neurons*. Springer, 2000.

[4] M. Andreux, E. Rodolà, M. Aubry, and D. Cremers. Anisotropic Laplace-Beltrami operators for shape analysis. In *Proc. NORDIA*, 2014.

[5] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, J. Rodgers, and J. Davis. SCAPE: shape completion and animation of people. *Trans. Graphics*, pages 408–416, 2005.

[6] M. Aubry, U. Schlickewei, and D. Cremers. The wave kernel signature: A quantum mechanical approach to shape analysis. In *Proc. 4DMOD*, 2011.

[7] P. Baldi and P. Sadowski. The dropout learning algorithm. *Artificial Intelligence*, 210:78–122, 2014.

[8] F. Bogo, M. J. Black, M. Loper, and J. Romero. Detailed full-body reconstructions of moving people from monocular RGB-D sequences. In *Proc. ICCV*, 2015.

[9] F. Bogo, J. Romero, M. Loper, and M. J. Black. FAUST: Dataset and evaluation for 3D mesh registration. In *Proc. CVPR*, 2014.

[10] D. Boscaini, J. Masci, S. Melzi, M. M. Bronstein, U. Castellani, and P. Vandergheynst. Learning class-specific descriptors for deformable shapes using localized spectral convolutional networks. *Computer Graphics Forum*, 34(5):13–23, 2015.

[11] D. Boscaini, J. Masci, E. Rodolà, and M. Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In *Proc. NIPS*, 2016.

[12] D. Boscaini, J. Masci, E. Rodolà, M. Bronstein, and D. Cremers. Anisotropic diffusion descriptors. *Computer Graphics Forum*, 35(2):431–441, 2016.

[13] L. Breiman. Random forests. In *Machine Learning*, volume 45, pages 5–32, 2001.

[14] J. Bromley et al. Signature verification using a "Siamese" time delay neural network. In *Proc. NIPS*. 1994.

[15] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. In *Proc. ICLR*, 2014.

[16] Q. Chen and V. Koltun. Robust nonrigid registration by convex optimization. In *IEEE ICCV*, 2015.

[17] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun. The loss surfaces of multilayer networks. In *Proc. AISTATS*, 2015.

[18] D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *Proc. ICLR*, 2015.

[19] L. Cosmo, E. Rodolà, M. M. Bronstein, et al. SHREC'16: Partial matching of deformable shapes. In *Proc. 3DOR*, 2016.

[20] L. Cosmo, E. Rodolà, J. Masci, A. Torsello, and M. Bronstein. Matching deformable objects in clutter. In *Proc. 3DV*, 2016.

[21] A. Criminisi, J. Shotton, and E. Konukoglu. Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Foundations and Trends in Computer Graphics and Vision*, 7(2-3):81–227, 2012.

[22] R. Dechter. Learning while searching in constraint-satisfaction-problems. In *Proc. AAAI*, 1986.

[23] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proc. CVPR*, 2009.

[24] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. FlowNet: Learning optical flow with convolutional networks. In *Proc. ICCV*, 2015.

[25] A. Dosovitskiy, J. T. Springenberg, and T. Brox. Learning to generate chairs with convolutional neural networks. In *Proc. CVPR*, 2015.

[26] M. Dou, J. Taylor, H. Fuchs, A. Fitzgibbon, and S. Izadi. 3d scanning deformable objects with a single rgbd sensor. In *IEEE CVPR*, 2015.

[27] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *JMLR*, 11, 2010.

[28] K. Fukushima. Neocognitron: A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.

[29] J. Gall, A. Yao, N. Razavi, L. Van Gool, and V. Lempitsky. Hough forests for object detection, tracking, and action recognition. *Trans. PAMI*, 33(11):2188–2202, 2011.

[30] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proc. Conf. Artificial Intelligence and Statistics*, 2010.

[31] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Proc. NIPS*. 2014.

[32] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra. DRAW: A recurrent neural network for image generation. In *Proc. ICML*, 2015.

[33] X. Han, T. Leung, Y. Jia, R. Sukthankar, and A. C. Berg. MatchNet: Unifying feature and metric learning for patch-based matching. In *Proc. CVPR*, 2015.

[34] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv:1512.03385*, 2015.

[35] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proc. ICCV*, 2015.

[36] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

[37] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, July 2006.

[38] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[39] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580*, 2012.

[40] S. Hochreiter and J. Schmidhuber. Simplifying neural nets by discovering flat minima. In *Proc. NIPS*, 1995.

[41] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[42] Q. Huang, B. Adams, M. Wicke, and L. J. Guibas. Non-rigid registration under isometric deformations. In *Proc. SGP*, 2008.

[43] S. I. and C. S. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167*, 2015.

[44] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. In *Proc. NIPS*, 2015.

[45] K. Kawaguchi. Deep learning without poor local minima. In *Proc. NIPS*, 2016.

[46] V. G. Kim, Y. Lipman, and T. Funkhouser. Blended Intrinsic Maps. *Trans. Graphics*, 30(4), 2011.

[47] D. P. Kingma, T. Salimans, and M. Welling. Variational dropout and the local reparameterization trick. In *Proc. NIPS*. 2015.

[48] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. NIPS*. 2012.

[49] Z. Lähner, E. Rodolà, M. M. Bronstein, et al. SHREC'16: Matching of deformable shapes with topological noise. In *Proc. 3DOR*, 2016.

[50] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

[51] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proc. ICML*, 2009.

[52] H. Li, B. Adams, L. J. Guibas, and M. Pauly. Robust single-view geometry and motion reconstruction. *Trans. Graphics*, 28(5), 2009.

[53] H. Li, E. Vouga, A. Gudym, L. Luo, J. T. Barron, and G. Gusev. 3D self-portraits. *Trans. Graphics*, 32(6), 2013.

[54] Y. Lipman and T. Funkhouser. Möbius voting for surface correspondence. *Trans. Graphics*, 28(3), 2009.

[55] O. Litany, E. Rodolà, A. M. Bronstein, M. M. Bronstein, and D. Cremers. Non-rigid puzzles. *Computer Graphics Forum*, 35(5), 2016.

[56] R. Litman and A. M. Bronstein. Learning spectral descriptors for deformable shape correspondence. *Trans. PAMI*, 36(1):170–180, 2014.

[57] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *Proc. CVPR*, 2015.

[58] S. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, 2008.

[59] J. Masci, D. Boscaini, M. M. Bronstein, and P. Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proc. 3dRR*, 2015.

[60] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *Proc. ICANN*, 2011.

[61] D. Maturana and S. Scherer. VoxNet: A 3D convolutional neural network for real-time object recognition. In *Conf. Intelligent Robots and Systems*, 2015.

[62] V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu. Recurrent models of visual attention. In *Proc. NIPS*. 2014.

[63] R. A. Newcombe, D. Fox, and S. M. Seitz. DynamicFusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proc. CVPR*, 2015.

[64] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. *arXiv:1505.04366*, 2015.

[65] K. Olszewski, J. J. Lim, S. Saito, and H. Li. High-fidelity facial and speech animation for VR HMDs. *Trans. Graphics*, 35(6), 2016.

[66] M. Ovsjanikov, M. Ben-Chen, J. Solomon, A. Butscher, and L. J. Guibas. Functional maps: a flexible representation of maps between shapes. *Trans. Graphics*, 31(4), 2012.

[67] V. Pham, C. Kermorvant, and J. Louradour. Dropout improves recurrent neural networks for handwriting recognition. *arXiv:1312.4569*, 2013.

[68] U. Pinkall and K. Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics*, 2(1):15–36, 1993.

[69] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *Proc. ICLR*. 2016.

[70] E. Rodolà, L. Cosmo, M. M. Bronstein, A. Torsello, and D. Cremers. Partial functional correspondence. *Computer Graphics Forum*, 2016.

[71] E. Rodolà, S. Rota Bulò, T. Windheuser, M. Vestner, and D. Cremers. Dense non-rigid shape correspondence using random forests. In *Proc. CVPR*, 2014.

[72] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[73] S. Rusinkiewicz, B. Brown, and M. Kazhdan. 3D scan matching and registration. In *Proc. ICCV Courses*, 2005.

[74] R. M. Rustamov, M. Ovsjanikov, O. Azencot, M. Ben-Chen, F. Chazal, and L. J. Guibas. Map-based exploration of intrinsic shape differences and variability. *Trans. Graphics*, 32(4):72, 2013.

[75] S. Saito, T. Li, and H. Li. Real-time facial segmentation and performance capture from RGB input. In *Proc. ECCV*, 2016.

[76] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. *arXiv:1503.03832*, 2015.

[77] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. OverFeat: Integrated recognition, localization and detection using convolutional networks. *arXiv:1312.6229*, 2013.

[78] J. Shotton, R. Girshick, A. Fitzgibbon, T. Sharp, M. Cook, M. Finocchio, R. Moore, P. Kohli, A. Criminisi, A. Kipman, and A. Blake. Efficient human pose estimation from single depth images. *Trans. PAMI*, 35(12):2821–2840, 2012.

[79] D. I. Shuman, B. Ricaud, and P. Vandergheynst. Vertex-frequency analysis on graphs. *arXiv:1307.5708*, 2013.

[80] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from RGBD images. In *Proc. ECCV*, 2012.

[81] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.

[82] S. Song and J. Xiao. Deep sliding shapes for amodal 3D object detection in RGB-D images. *arXiv:1511.02300*, 2015.

[83] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. In *Proc. NIPS*, 2015.

[84] R. K. Srivastava, J. Masci, F. J. Gomez, and J. Schmidhuber. Understanding locally competitive networks. In *Proc. ICLR*, 2015.

[85] M. Stollenga, J. Masci, F. Gomez, and J. Schmidhuber. Deep networks with internal selective attention through feedback connections. In *Proc. NIPS*, 2014.

[86] H. Su, S. Maji, E. Kalogerakis, and E. G. Learned-Miller. Multi-view convolutional neural networks for 3D shape recognition. In *Proc. ICCV*, 2015.

[87] J. Sun, M. Ovsjanikov, and L. J. Guibas. A concise and provably informative multi-scale signature based on heat diffusion. In *Proc. SGP*, 2009.

[88] J. Süssmuth, M. Winter, and G. Greiner. Reconstructing animated meshes from time-varying point clouds. *Computer Graphics Forum*, 27(5):1469–1476, 2008.

[89] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. Technical report, 2014.

[90] J. Taylor, J. Shotton, T. Sharp, and A. Fitzgibbon. The vitruvian manifold: Inferring dense correspondences for one-shot human pose estimation. In *Proc. CVPR*, 2012.

[91] A. Tevs, A. Berner, M. Wand, I. Ihrke, M. Bokeloh, J. Kerber, and H.-P. Seidel. Animation cartography – intrinsic reconstruction of shape and motion. *Trans. Graphics*, 31(2):12:1–12:15, 2012.

[92] J. Tompson, M. Stein, Y. LeCun, and K. Perlin. Real-time continuous pose recovery of human hands using convolutional networks. *Trans. Graphics*, 33(5), 2014.

[93] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *JMLR*, 11:3371–3408, 2010.

[94] D. Vlasic, I. Baran, W. Matusik, and J. Popović. Articulated mesh animation from multi-view silhouettes. *Trans. Graphics*, 27(3), 2008.

[95] M. Wand, B. Adams, M. Ovsjanikov, A. Berner, M. Bokeloh, P. Jenke, L. Guibas, H.-P. Seidel, and A. Schilling. Efficient reconstruction of nonrigid shape and motion from real-time 3D scanner data. *Trans. Graphics*, 28(2), 2009.

[96] S. Wang and C. Manning. Fast dropout training. In *Proc. ICML*, 2013.

[97] L. Wei, Q. Huang, D. Ceylan, E. Vouga, and H. Li. Dense human body correspondences using convolutional networks. In *Proc. CVPR*, 2016.

[98] X. Wei, P. Zhang, and J. Chai. Accurate realtime full-body motion capture using a single depth camera. *Trans. Graphics*, 31(6), 2012.

[99] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

[100] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D Shapenets: A deep representation for volumetric shapes. In *Proc. CVPR*, 2015.

[101] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proc. ICML*, 2015.

[102] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. In *Proc. ICLR*. 2016.

[103] M. E. Yumer and N. J. Mitra. Learning semantic deformation flows with 3d convolutional networks. In *Proc. ECCV*, 2016.

[104] S. Zagoruyko and N. Komodakis. Learning to compare image patches via convolutional neural networks. In *Proc. CVPR*, 2015.

[105] S. Zagoruyko, A. Lerer, T.-Y. Lin, P. O. Pinheiro, S. Gross, S. Chintala, and P. Dollár. A multipath network for object detection. In *Proc. BMVC*, 2016.

[106] A. Zeng, S. Song, M. Nießner, M. Fisher, and J. Xiao. 3DMatch: Learning the matching of local 3D geometry in range scans. *arXiv:1603.08182*, 2016.

[107] S. Zhou, J. Wu, Y. Wu, and X. Zhou. Exploiting local structures with the kronecker layer in convolutional networks. *arXiv:1512.09194*, 2015.