SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels

Matthias Fey*, Jan Eric Lenssen*, Frank Weichert, Heinrich Müller
Department of Computer Graphics
TU Dortmund University
{matthias.fey,janeric.lenssen}@udo.edu
* Both authors contributed equally to this work.

Abstract

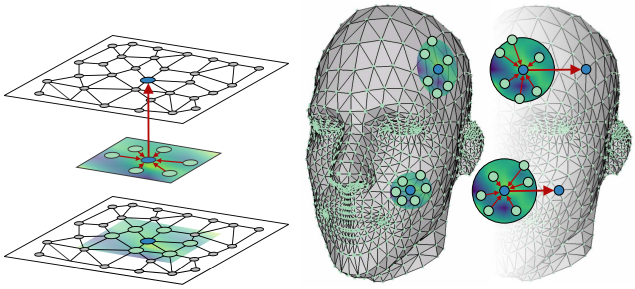
We present Spline-based Convolutional Neural Networks (SplineCNNs)*, a variant of deep neural networks for irregular structured and geometric input,* e.g.*, graphs or meshes. Our main contribution is a novel convolution operator based on B-splines, that makes the computation time independent from the kernel size due to the local support property of the B-spline basis functions. As a result, we obtain a generalization of the traditional CNN convolution operator by using continuous kernel functions parametrized by a fixed number of trainable weights. In contrast to related approaches that filter in the spectral domain, the proposed method aggregates features purely in the spatial domain. In addition, SplineCNN allows entire end-to-end training of deep architectures, using only the geometric structure as input, instead of handcrafted feature descriptors.*

For validation, we apply our method on tasks from the fields of image graph classification, shape correspondence and graph node classification, and show that it outperforms or pars state-of-the-art approaches while being significantly faster and having favorable properties like domain-independence. Our source code is available on GitHub[1].

1. Introduction

Most achievements obtained by deep learning methods over the last years heavily rely on properties of the convolution operation in convolutional neural networks [14]: local connectivity, weight sharing and shift invariance. Since those layers are defined on inputs with a grid-like structure, they are not trivially portable to non-Euclidean domains like discrete manifolds, or (embedded) graphs. However, a large amount of data in practical tasks naturally comes in the form of such irregular structures, *e.g.* graph data or meshes. Transferring the high performance of traditional convolutional neural networks to this kind of data holds the potential for large improvements in several relevant tasks.

[1] https://github.com/rusty1s/pytorch_geometric



(a) Filtering of graphs (b) Filtering of meshes

Figure 1: Examples for spatial aggregation in geometric deep learning with trainable, continuous kernel functions, showing methods for (a) image graph representations and (b) meshes.

Recently, a set of methods brought together under the term *geometric deep learning* [3] emerged, which aim to achieve this transfer by defining convolution operations for deep neural networks that can handle irregular input data. Existing work in this field can loosely be divided into two different subsets: the spectral and the spatial filtering approaches. The former is based on spectral graph theory [5], where eigenvalues of a graph's Laplacian matrix are interpreted as frequencies of node signals [22]. They are filtered in the spectral domain, analogously to Fourier domain filtering of traditional signals. The latter subset, the spatial approaches, perform convolution in local Euclidean neighborhoods w.r.t. local positional relations between points, represented for example as polar, spherical or Cartesian coordinates, as shown as examples in Figure 1.

Contribution. We present *Spline-based Convolutional Neural Networks (SplineCNNs)*, a variant of deep neural networks for irregular structured data. The main contribution is a trainable, spatial, continuous convolution kernel that leverages properties of B-spline bases to efficiently filter geometric input of arbitrary dimensionality. We show

that our method

- can be applied on different kinds of irregular structured data, *e.g*., arbitrary (embedded) graphs and meshes,
- uses spatial geometric relations of the input,
- allows for end-to-end training without using hand-crafted feature descriptors, and
- improves or pars the state-of-the-art in geometric learning tasks.

In addition, we provide an efficient GPGPU algorithm and implementation that allows for fast training and inference computation.

2. Related work

Deep learning on graphs. The history of geometric deep learning began with attempts to generalize convolutional neural networks for graph inputs. A large number of successful approaches are based on spectral graph theory. Bruna *et al*. [4] introduced convolution-like operators on spectral graphs, interpreting the eigenvectors of the Laplacian as Fourier basis. As an extension, Henaff *et al*. [9] suggest to use spline interpolation for smoothing kernels in the spectral domain. Defferrard *et al*. [6] approximates spectral filters with Chebyshev polynomials, providing a more efficient filtering algorithm, whose kernel size determines the range of aggregated local K-neighborhoods. This approach was further simplified by Kipf and Welling [12], who consider only the one-neighborhood for one filter application. A filter based on the Caley transform was proposed as an alternative for the Chebyshev approximation by Levie *et al*. [15]. Together with a trainable zooming parameter, this results in a more stable and flexible spectral filter.

It should be noted that all these spectral approaches assume that information is only encoded in the connectivity, edge weights and node features of the input. While this is true for general graphs, it does not hold for embedded graphs or meshes, where additional information is given by relative positions of nodes, which we consider with our method.

A downside of many spectral approaches is the fact that they use domain-dependent Fourier bases, which restricts generalization to inputs with identical graph connectivity. Yi *et al*. [25] tackle this problem by applying a spectral transformer network that synchronizes the spectral domains. Since our approach works directly in the spatial domain, it is not prone to this problem.

For the shape correspondence task on meshes, which we also analyze in this work, Litany *et al*. [16] present a siamese network using a soft error criterion based on geodesic distances between nodes. We compare our method against this specialized method.

Local descriptors for discrete manifolds. The issue of not representing local positional relations can be tackled by using methods that extract representations for local Euclidean neighborhoods from discrete manifolds.

Based on the intrinsic shape descriptors of Kokkinos *et al*. [13], Masci *et al*. [17] present such a method for extraction of two-dimensional Euclidean neighborhoods from meshes and propose a convolution operation locally applied on these neighborhoods. Boscaini *et al*. [2] improve this approach by introducing a patch rotation method to align extracted patches based on the local principal curvatures of the input mesh.

Our convolution operator can but does not have to receive those local representations as inputs. Therefore, our approach is orthogonal to improvements in this field.

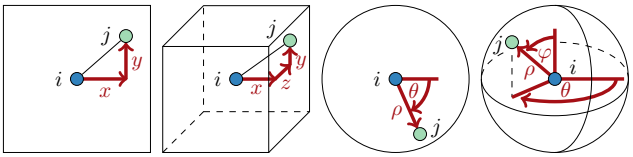
Spatial continuous convolution kernels. While the first continuous convolution kernels for graphs work in the *spectral* domain (*e.g*. [9, 6, 20]), *spatial* continuous convolution kernels for irregular structured data were introduced recently as a special case in the fields of neural message passing and self-attention mechanisms [8, 23, 18]. Furthermore, Monti *et al*. [18] presented the MoNet framework for interpreting different kind of inputs as directed graphs, on which we built upon in our work. We show that our kernels achieve the same or better accuracy as the trainable Gaussian mixture model (GMM) kernels of MoNet, while being able to be trained directly on the geometric structure.

3. SplineCNN

We define SplineCNNs as a class of deep neural networks that are built using a novel type of spline-based convolutional layer. This layer receives irregular structured data, which is mapped to a directed graph, as input. In the spatial convolutional layer, node features are aggregated using a trainable, continuous kernel function, which we define in this section.

3.1. Preliminaries

Input graphs. Similar to the work of Monti *et al*. [18], we expect the input of our convolution operator to be a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{U})$ with $\mathcal{V} = \{1, \dots, N\}$ being the set of nodes, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ the set of edges, and $\mathbf{U} \in [0,1]^{N \times N \times d}$ containing d-dimensional pseudo-coordinates $\mathbf{u}(i,j) \in [0,1]^d$ for each directed edge $(i,j) \in \mathcal{E}$. Note that $\mathbf{U}$ can be interpreted as an adjacency matrix with d-dimensional, normalized entries $\mathbf{u}(i,j) \in [0,1]^d$ if $(i,j) \in \mathcal{E}$ and $\mathbf{0}$ otherwise. Also, $\mathbf{U}$ is usually sparse with $E = |\mathcal{E}| \ll N^2$ entries. For a node $i \in \mathcal{V}$ its *neighborhood* set is denoted by $\mathcal{N}(i)$.



$\mathbf{u}(i,j) = (x,y)$ $\mathbf{u}(i,j) = (x,y,z)$ $\mathbf{u}(i,j) = (\rho,\theta)$ $\mathbf{u}(i,j) = (\rho,\theta,\varphi)$

Figure 2: Possibilities for pseudo-coordinates $\mathbf{u}$: two- and three-dimensional Cartesian, polar and spherical coordinates. Values for scaling and translation of the coordinates $\mathbf{u}$ to interval $[0,1]^d$ are omitted.

Input node features. Let $\mathbf{f} : \mathcal{V} \to \mathbb{R}^{M_{\text{in}}}$, with $\mathbf{f}(i) \in \mathbb{R}^{M_{\text{in}}}$, denote a vector of M_{in} input features for each node $i \in \mathcal{V}$. For each $1 \leq l \leq M_{\text{in}}$ we reference the set $\{f_l(i) \mid i \in \mathcal{V}\}$ as *input feature map*.

B-spline basis functions. In addition to the input graph and node features, let $((N^m_{1,i})_{1\leq i\leq k_1}, \dots, (N^m_{d,i})_{1\leq i\leq k_d})$ denote d open B-spline bases of degree m, based on uniform, *i.e.* equidistant, knot vectors (*c.f.* Piegl *et al.* [19]), with $\mathbf{k} = (k_1, \dots, k_d)$ defining our d-dimensional kernel size.

3.2. Main concept

Our convolution operator aggregates node features in local neighborhoods weighted by a trainable, continuous kernel function. The node features $\mathbf{f}(i)$ represent features on an irregular geometric structure, whose spatial relations are locally defined by the pseudo-coordinates in $\mathbf{U}$. Therefore, when locally aggregating feature values in a node's neighborhood, the content of $\mathbf{U}$ is used to determine *how* the features are aggregated and the content of $\mathbf{f}(i)$ defines *what* is aggregated. We argue that common inputs for geometric deep learning tasks can be mapped to this model while preserving relevant information:

- For **graphs**, $\mathcal{V}$ and $\mathcal{E}$ are given and $\mathbf{U}$ can contain edge weights or, for example, features like the node degree of the target nodes.
- For **discrete manifolds**, $\mathcal{V}$ contains points of the discrete manifold, $\mathcal{E}$ represents connectivity in local Euclidean neighborhoods and $\mathbf{U}$ can contain local relational information like polar, spherical or Cartesian coordinates of the target point in respect to the origin point for each edge.

We state no restriction on the values of $\mathbf{U}$, except being element of a fixed interval range. Therefore, meshes, for example, can be either interpreted as embedded three-dimensional graphs or as two-dimensional manifolds, using local Euclidean neighborhood representations like obtained by the work of Boscaini *et al.* [2]. Also, either polar/spherical coordinates or Cartesian coordinates can be used, as shown in Figure 2. Independent from the type of coordinates stored in $\mathbf{U}$, our trainable, continuous kernel function, which we define in the following section, maps each $\mathbf{u}(i,j)$ to a scalar that is used as a weight for feature aggregation.

3.3. Convolution operator

We begin with the definition of a continuous kernel function using B-spline bases, which is parametrized by a constant number of trainable control values. The local support property of B-spline basis functions [19], which states that basis functions evaluate to zero for all inputs outside of a known interval, proves to be advantageous for efficient computation and scalability.

Figure 3 visualizes the following kernel construction method for differing B-spline basis degree m. We introduce a trainable parameter $w_{\mathbf{p},l} \in \mathbf{W}$ for each element $\mathbf{p}$ from the Cartesian product $\mathcal{P} = (N^m_{1,i})_i \times \cdots \times (N^m_{d,i})_i$ of the B-spline bases and each of the M_{in} input feature maps, indexed by l. This results in $K = M_{\text{in}} \cdot \prod_{i=1}^{d} k_i$ trainable parameters.

We define our continuous convolution kernel as functions $g_l : [a_1, b_1] \times \cdots \times [a_d, b_d] \to \mathbb{R}$ with

$$g_l(\mathbf{u}) = \sum_{\mathbf{p}\in\mathcal{P}} w_{\mathbf{p},l} \cdot B_{\mathbf{p}}(\mathbf{u}), \tag{1}$$

with $B_{\mathbf{p}}$ being the product of the basis functions in $\mathbf{p}$:

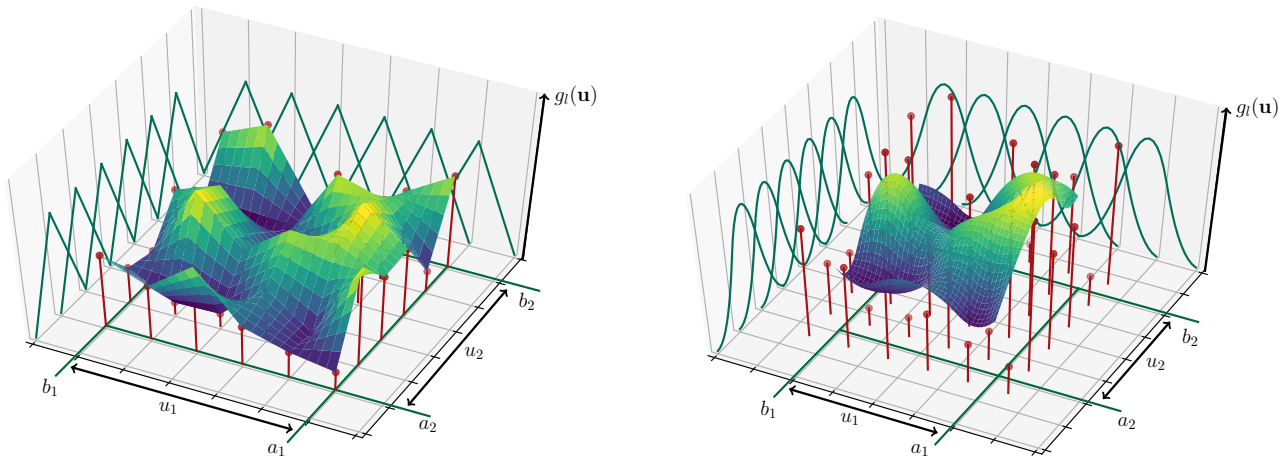
$$B_{\mathbf{p}}(\mathbf{u}) = \prod_{i=1}^{d} N^m_{i,p_i}(u_i). \tag{2}$$

One way to interpret this kernel is to see the trainable parameters $w_{\mathbf{p},l}$ as control values for the height of a $d+1$-dimensional B-spline surface, from which a weight is sampled for each neighboring point j, depending on $\mathbf{u}(i,j)$. However, in contrast to traditional $(d+1)$-dimensional B-spline approximation, we only have one-dimensional control points and approximate functions $g_l : [a_1, b_1] \times \cdots \times [a_d, b_d] \to \mathbb{R}$ instead of curves. The definition range of g_l is the interval in which the partition of unity property of the B-spline bases holds [19]. Therefore, a_i and b_i depend on B-spline degree m and kernel size $(k_1, \dots, k_d)$. We scale the spatial relation vectors $\mathbf{u}(i,j)$ to exactly match this interval, *c.f.* Figure 3.

Given our kernel functions $\mathbf{g} = (g_1, \dots, g_{M_{\text{in}}})$ and input node features $\mathbf{f}$, we define our spatial convolution operator for a node i as

$$(\mathbf{f} \star \mathbf{g})(i) = \frac{1}{|\mathcal{N}(i)|} \sum_{l=1}^{M_{\text{in}}} \sum_{j\in\mathcal{N}(i)} f_l(j) \cdot g_l(\mathbf{u}(i,j)). \tag{3}$$

Similar to traditional CNNs, the convolution operator can be used as a module in a deep neural network architecture,



(a) Linear B-spline basis functions

(b) Quadratic B-spline basis functions

Figure 3: Examples of our continuous convolution kernel for B-spline basis degrees (a) $m = 1$ and (b) $m = 2$ for kernel dimensionality $d = 2$. The heights of the red dots are the trainable parameters for a single input feature map. They are multiplied by the elements of the B-spline tensor product basis before influencing the kernel value.

which we do in our SplineCNNs. To this end, the operator is applied M_{out} times on the same input data with different trainable parameters, to obtain a convolutional layer that produces M_{out} output feature maps. It should be highlighted that, in contrast to self-attention methods, we train an individual set of weights for each combination of input and output feature map.

Local support. Due to the local support property of B-splines, $B_{\mathbf{p}} \neq 0$ only holds true for $s := (m+1)^d$ of the K different vectors $\mathbf{p} \in \mathcal{P}$. Therefore, $g_l(\mathbf{u})$ only depends on $M_{\text{in}} \cdot s$ of the $M_{\text{in}} \cdot K$ trainable parameters for each neighbor j, where s, d and m are constant and usually small. In addition, for each pair of nodes $(i, j) \in \mathcal{E}$, the vectors $\mathbf{p} \in \mathcal{P}$ with $B_{\mathbf{p}} \neq 0$, which we denote as $\mathcal{P}(\mathbf{u}(i, j))$, can be found in constant time, given constant m and d.

This allows for an alternative representation of the inner sums of our convolution operation, *c.f.* Equation 3, as

$$(f_l \star g_l)(i) = \sum_{\substack{j \in \mathcal{N}(i) \\ \mathbf{p} \in \mathcal{P}(\mathbf{u}(i,j))}} f_l(j) \cdot w_{\mathbf{p},l} \cdot B_{\mathbf{p}}(\mathbf{u}(i, j)). \quad (4)$$

and K can be replaced by s in the time complexity of the operation. Also, $B_{\mathbf{p}}(\mathbf{u}(i, j))$ does not depend on l and can therefore be computed once for all input features. Figure 4 shows a scheme of the computation. The gradient flow for the backward pass can also be derived by following the solid arrows backwards.

Closed B-splines. Depending on the type of coordinate in vectors $\mathbf{u}$, we use closed B-spline approximation in some dimensions. One frequently occurring example of such a

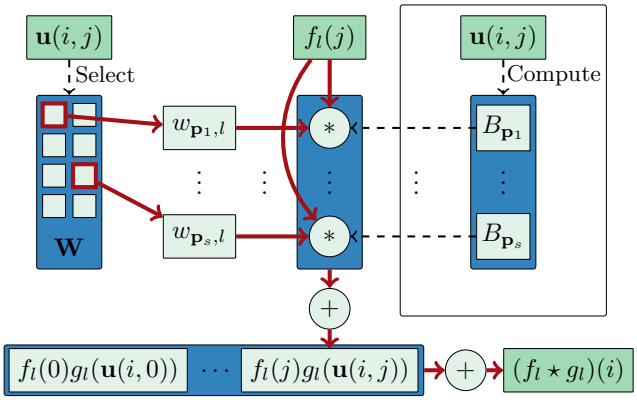


Figure 4: Forward computation scheme of the proposed convolution operation. During the backward step of the backpropagation algorithm, the gradient flows along the inverted solid arrows, reaching inputs from $\mathbf{W}$ and $f_l(i)$.

situation is when $\mathbf{u}$ contains angle attributes of polar coordinates. Using closed B-spline approximation in the angle dimension naturally enforces the angle 0 to be evaluated to the same weight as the angle 2π or, for higher m, the kernel function to be continuously differentiable at those points.

The proposed kernels can easily be modified so that they use closed approximation in an arbitrary subset of the d dimensions, by mapping different $\mathbf{p} \in \mathcal{P}$ to the same trainable control value $w_{\mathbf{p},l}$. This leads to a reduction of trainable parameters and B-spline basis functions. Referring to Figure 3, this approach can be interpreted as periodic repetition of the function surface along the corresponding axis.

Root nodes. Up to now, we did not consider the node i of neighborhood $\mathcal{N}(i)$ in our convolution operator. It is not aggregated together with all $j \in \mathcal{N}(i)$, like it would be the case in traditional CNNs. If Cartesian coordinates are used, we can simply define $\mathcal{N}(i)$ to include i. However, when using polar/spherical pseudo-coordinates, problems arise since the point with zero radius is not well defined. Therefore, we introduce an additional trainable parameter for each feature of the root node and add the product of this parameter and the corresponding feature to the result.

Relation to traditional CNNs. Except for a normalization factor, our spline-based convolution operator is a generalization of the traditional convolutional layer in CNNs with *odd* filter size in each dimension. For example, if we assume to have a two-dimensional grid-graph with diagonal, horizontal and vertical edges to be the input, B-spline degree $m = 1$, kernel size $(3, 3)$, and the vectors $\mathbf{u}$ to contain Cartesian relations between adjacent nodes, then our convolution operator is equivalent to a discrete convolution of an image with a kernel of size 3×3. This also holds for larger discrete kernels if the neighborhoods of the grid-graph are modified accordingly.

4. GPGPU algorithm

For the spline-based convolutional layer defined in the last section, we introduce a GPU algorithm which allows efficient training and inference with SplineCNNs. For simplicity, we use a tensor indexing notation with, *e.g.*, $\mathbf{A}[x, y, z]$ describing the element at position (x, y, z) of a tensor $\mathbf{A}$ with rank three. Our forward operation of our convolution operator is outlined in Algorithm 1.

We achieve parallelization over the edges $\mathcal{E}$ by first gathering edge-wise input features $\mathbf{F}_{\text{in}}^E \in \mathbb{R}^{E \times M_{\text{in}}}$ from the input matrix $\mathbf{F}_{\text{in}} \in \mathbb{R}^{N \times M_{\text{in}}}$, using the target node of each edge as index. Then, we compute edge-wise output features $\mathbf{F}_{\text{out}}^E \in \mathbb{R}^{E \times M_{\text{out}}}$, as shown in Figure 4, before scatter-adding them back to node-wise features $\mathbf{F}_{\text{out}} \in \mathbb{R}^{N \times M_{\text{out}}}$, performing the actual neighborhood aggregation. Our algorithm has a parallel time complexity of $\mathcal{O}(s \cdot M_{\text{in}})$, with small s, using $\mathcal{O}(E \cdot M_{\text{out}})$ processors, assuming that scatter-add is a parallel operation with constant time complexity.

Computing B-spline bases. We achieve independence from the number of trainable weights by computing matrices $\mathbf{P} \in \mathbb{N}^{E \times s}$ and $\mathbf{B} \in \mathbb{R}^{E \times s}$. $\mathbf{P}$ contains the indices of parameters with $B_{\mathbf{p}} \neq 0$ while $\mathbf{B}$ contains the basis products $B_{\mathbf{p}}$ for these parameters. $\mathbf{B}$ and $\mathbf{P}$ can be preprocessed for a given graph structure or can be computed directly in the kernel. For the GPU evaluation of the basis functions required for $\mathbf{B}$ we use explicit low-degree polynomial formulations of those functions for each m. For further details we refer to our PyTorch implementation, which is available on GitHub.

Algorithm 1 Geometric convolution with B-spline kernels

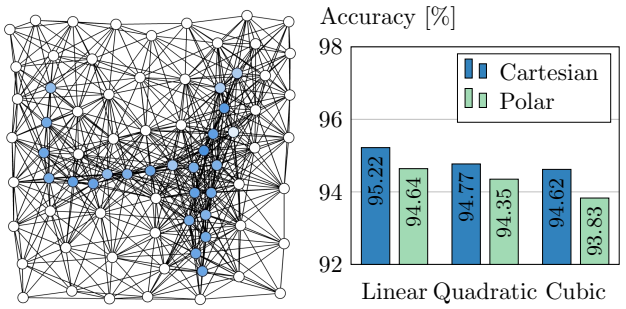
Input:
N: Number of nodes
M_{in}: Number of input features per node
M_{out}: Number of output features per node
$s = (m+1)^d$: Number of non-zero $B_{\mathbf{p}}$ for one edge
$\mathbf{W} \in \mathbb{R}^{K \times M_{\text{in}} \times M_{\text{out}}}$: Trainable weights
$\mathbf{B} \in \mathbb{R}^{E \times s}$: Basis products of s weights for each edge
$\mathbf{P} \in \mathbb{N}^{E \times s}$: Indices of s weights in $\mathbf{W}$ for each edge
$\mathbf{F}_{\text{in}} \in \mathbb{R}^{N \times M_{\text{in}}}$: Input features for each node
Output:
$\mathbf{F}_{\text{out}} \in \mathbb{R}^{N \times M_{\text{out}}}$: Output features for each node

Gather $\mathbf{F}_{\text{in}}^E$ from $\mathbf{F}_{\text{in}}$ based on target nodes of edges
Parallelize over $e \in \{1, \dots, E\}$, $o \in \{1, \dots, M_{\text{out}}\}$:
  $r \leftarrow 0$
  for each $i \in \{1, \dots, M_{\text{in}}\}$ **do**
    for each $p \in \{1, \dots, s\}$ **do**
      $w \leftarrow \mathbf{W}[\mathbf{P}[e, p], i, o]$
      $r \leftarrow r + (\mathbf{F}_{\text{in}}^E[e, i] \cdot w \cdot \mathbf{B}[e, p])$
    end for
  end for
  $\mathbf{F}_{\text{out}}^E[e, o] \leftarrow r$
Scatter-add $\mathbf{F}_{\text{out}}^E$ to $\mathbf{F}_{\text{out}}$ based on origin nodes of edges
Return $\mathbf{F}_{\text{out}}$

Mini-batch handling. For batch learning, parallelization over a mini-batch can be achieved by creating sparse block diagonal matrices out of all $\mathbf{U}$ of one batch and concatenating matrices $\mathbf{F}_{\text{in}}$ in the node dimension. For matrices $\mathbf{F}_{\text{in}}^E$, $\mathbf{B}$ and $\mathbf{P}$, this results in example-wise concatenation in the edge dimension. Note that this composition allows differing number of nodes and edges over examples in one batch without introducing redundant computational overhead.

5. Results

We perform experiments with different SplineCNN architectures on three distinct tasks from the fields of image graph classification (Section 5.1), graph node classification (Section 5.2) and shape correspondence on meshes (Section 5.3). For each of the tasks, we create a SplineCNN using the spline-based convolution operator which we denote as SConv($\mathbf{k}$, M_{in}, M_{out}) for a convolutional layer with kernel size $\mathbf{k}$, M_{in} input feature maps and M_{out} output feature maps. In addition, we denote fully connected layers as FC(o), with o as number of output neurons.



(a) MNIST superpixels example (b) Classification accuracy

Figure 5: MNIST 75 superpixels (a) example and (b) classification accuracy of SplineCNN using varying pseudo-coordinates and B-spline base degrees.

5.1. Image graph classification

For validation on two-dimensional regular and irregular structured input data, we apply our method on the widely-known MNIST dataset [14] of 60,000 training and 10,000 test images containing grayscale, handwritten digits from 10 different classes. We conduct two different experiments on MNIST. For both experiments, we strictly follow the experimental setup of Defferrard *et al*. and Monti *et al*. [6, 18] to provide comparability. For the first experiment, the MNIST images are represented as a set of *equal* grid graphs, where each node corresponds to one pixel in the original image, resulting in grids of size 28×28 with $N = 28^2 = 784$ nodes. For the second experiment, the MNIST superpixel dataset of Monti *et al*. [18] is used, where each image is represented as an embedded graph of 75 nodes defining the centroids of superpixels, *c.f*. Figure 5a, with each graph having *different* node positions and connectivities. This experiment is an ideal choice to validate the capabilities of our approach on irregular structured, image-based data.

Pooling. Our SplineCNN architectures use a pooling operator based on the Graclus method [7, 6]. The pooling operation is able to obtain a coarsened graph by deriving a clustering on the graph nodes, aggregating nodes in one cluster and computing new pseudo-coordinates for each of those new nodes. We denote a max-pooling layer using this algorithm with MaxP(c), with c being the cluster size (and approximate downscaling factor).

Architectures and parameters. For the grid graph experiments, Cartesian coordinates and a B-spline basis degree of $m = 1$ are used to reach equivalence to the traditional convolution operator in CNNs, *c.f*. Section 3.3. In contrast, we compare all configurations of m and possible pseudo-coordinates against each other on the superpixel dataset.

Dataset	LeNet5 [14]	MoNet [18]	**SplineCNN**
Grid	**99.33%**	99.19%	99.22%
Superpixels	–	91.11%	**95.22%**

Table 1: Classification accuracy on different representations of the MNIST dataset (grid and superpixel) for a classical CNN (LeNet5), MoNet and our SplineCNN approach.

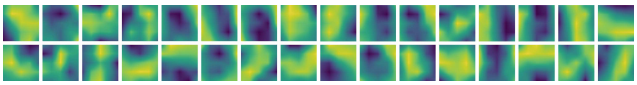
For classification on the grid data, we make use of a LeNet5-like network architecture [14]: SConv($(5, 5)$,1,32) $\rightarrow$ MaxP(4) $\rightarrow$ SConv($(5, 5)$,32,64) $\rightarrow$ MaxP(4) $\rightarrow$ FC(512) $\rightarrow$ FC(10). The initial learning rate was chosen as 10^{-3} and dropout probability as 0.5. Note that we used neighborhoods of size 5×5 from the grid graph, to mirror the LeNet5 architecture with its 5×5 filters.

The superpixel dataset is evaluated using the SplineCNN architecture SConv((k_1, k_2),1,32) $\rightarrow$ MaxP(4) $\rightarrow$ SConv((k_1, k_2),32,64) $\rightarrow$ MaxP(4) $\rightarrow$ AvgP $\rightarrow$ FC(128) $\rightarrow$ FC(10), where AvgP denotes a layer that averages features in the node dimension. We use the *Exponential Linear Unit (ELU)* as non-linearity after each SConv layer and the first FC layer. For Cartesian coordinates, we choose the kernel size to be $k_1 = k_2 = 4 + m$ and for polar coordinates $k_1 = 1 + m$ and $k_2 = 8$. Training was done for 20 epochs with a batch size of 64, initial learning rate 0.01 and dropout probability 0.5. Both networks were trained for 30 epochs using the Adam method [11].

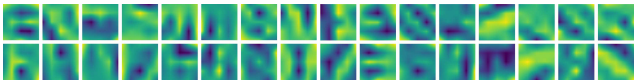
Discussion. All results of the MNIST experiments are shown in Table 1 and Figure 5b. The grid graph experiment results in approximately the same accuracy as LeNet5 and the MoNet method. For the superpixel dataset, we improve previous results by 4.11 percentage points in accuracy. Since we are using a similar architecture and the same input data as MoNet, the better results are an indication that our operator is able to capture more relevant information in the structure of the input. This can be explained by the fact that, in contrast to the MoNet kernels, our kernel function has individual trainable weights for each combination of input and output feature maps, just like the filters in traditional CNNs.

Results for different configurations are shown in Figure 5b. We only notice small differences in accuracy for varying m and pseudo-coordinates. However, lower m and using Cartesian coordinates performs slightly better than the other configurations.

In addition, we visualized the 32 learned kernels of the first SConv layers from the grid and superpixel experiments in Figure 6. It can be observed that edge detecting patterns are learned in both approaches, whether being trained on regular or irregular structured data.



(a) MNIST grid experiment



(b) MNIST superpixel experiment

Figure 6: Visualizations of the 32 kernels from the first spline-based convolutional layers, trained on the MNIST (a) grid and (b) superpixels datasets, with kernel size $(5, 5)$ and B-spline base degree $m = 1$.

ChebNet [6]	GCN [12]	CayleyNet [15]	**SplineCNN**
87.12 ± 0.60	87.17 ± 0.58	87.90 ± 0.66	$\mathbf{89.48 \pm 0.31}$

Table 2: Graph node classification on the Cora dataset for different learning methods (ChebNet, GCN, CayleyNet and SplineCNN). The presented accuracy means and standard deviations are computed over 100 experiments, where for each experiment the network was trained for 200 epochs.

5.2. Graph node classification

As second experiment, we address the problem of graph node classification using the Cora citation graph [21]. We validate that our method also performs strongly on datasets, where no Euclidean relations are given. Cora consists of 2,708 nodes and 5,429 undirected unweighted edges, representing scientific publications and citation links respectively. Each document is represented individually by a 1,433 dimensional sparse binary bag-of-words feature vector and is labeled to exactly one out of 7 classes. Similar to the experimental setup in Levi *et al*. [15], we split the dataset into 1,708 nodes for training and 500 nodes for testing, to simulate labeled and unlabeled information.

Architecture and parameters. We use a SplineCNN similar to the network architecture introduced in [15, 12, 18]: SConv((2),1433,16) $\rightarrow$ SConv((2),16,7), with ELU activation after the first SConv layer and $m = 1$. For pseudo-coordinates, we choose the globally normalized degree of the target nodes $\mathbf{u}(i, j) = (\deg(j)/\max_{v \in \mathcal{V}} \deg(v))$, leading to filtering based on the number of cites of neighboring publications. Training was done using the Adam optimization method [11] for 200 epochs with learning rate 0.01, dropout probability 0.5 and L2 regularization 0.005. As loss function, the cross entropy between the network's softmax output and a one-hot target distribution was used.

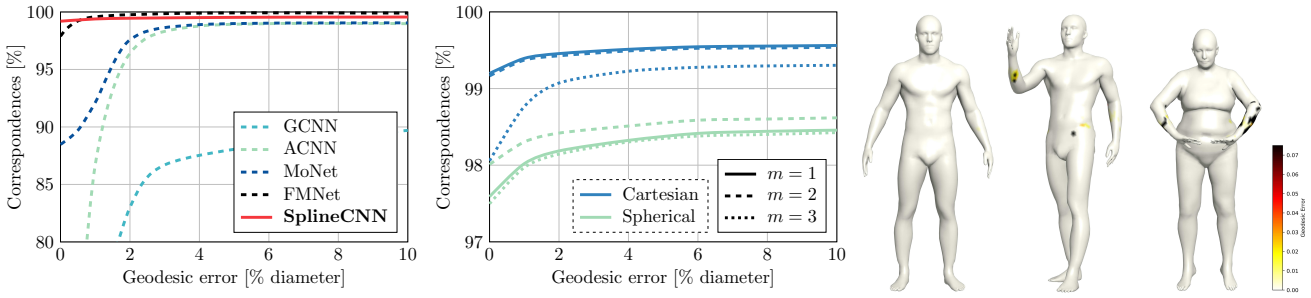
Discussion. Results of our and related methods are shown in Table 2 and report the mean classification accuracy averaged over 100 experiments. It can be seen that SplineCNNs improve the state-of-the-art in this experiment by approximately 1.58 percentage points. We contribute this improvement to the filtering based on $\mathbf{u}$, which contains node degrees as additional information to learn more complex kernel functions. This indicates that SplineCNNs can be successfully applied to irregular but non-geometric data and that they are able to improve previous results in this domain.

5.3. Shape correspondence

As our last and largest experiment, we validate our method on a collection of three-dimensional meshes solving the task of shape correspondence similar to [18, 2, 17, 16]. Shape correspondence refers to the task of labeling each node of a given shape to the corresponding node of a reference shape [17]. We use the FAUST dataset [1], containing 10 scanned human shapes in 10 different poses, resulting in a total of 100 non-watertight meshes with 6,890 nodes each. The first 80 subjects in FAUST were used for training and the remaining 20 subjects for testing, following the dataset splits introduced in [18]. Ground truth correspondence of FAUST meshes are given implicitly, where nodes are sorted in the exact same order for every example. Correspondence quality is measured according to the Princeton benchmark protocol [10], counting the percentage of derived correspondences that lie within a geodesic radius r around the correct node.

In contrast to similar approaches, *e.g*. [18, 2, 17, 16], we go without handcrafted feature descriptors as inputs, like the local histogram of normal vectors known as SHOT descriptors [24], and force the network to learn from the geometry (*i.e*. spatial relations encoded in $\mathbf{U}$) itself. Therefore, input features are trivially given by $\mathbf{1} \in \mathbb{R}^{N \times 1}$. Also, we validate our method on three-dimensional meshes as inputs instead of generating two-dimensional geodesic patches for each node. These simplifications reduce the computation time and memory consumption that are required to preprocess the data by a wide margin, making training and inference completely end-to-end and very efficient.

Architecture and parameters. We apply a SplineCNN architecture with 6 convolutional layers: SConv((k_1, k_2, k_3),1,32) $\rightarrow$ SConv((k_1, k_2, k_3),32,64) $\rightarrow$ $4\times$ SConv((k_1, k_2, k_3),64,64) $\rightarrow$ Lin(256) $\rightarrow$ Lin(6890), where Lin(o) denotes a 1×1 convolutional layer to o output features per node. As non-linear activation function, ELU is used after each SConv and the first Lin layer. For Cartesian coordinates we choose the kernel size to be $k_1 = k_2 = k_3 = 4 + m$ and for polar coordinates $k_1 = k_3 = 4 + m$ and $k_2 = 8$. We evaluate our method on multiple choices of $m = \{1, 2, 3\}$. Training was done for



(a) Results of SplineCNN and other methods (b) Results for different SplineCNNs (c) Geodesic error of test examples

Figure 7: Geodesic error plots of the shape correspondence experiments with (a) SplineCNN and related approaches and (b) different SplineCNN experiments. The x-axis displays the geodesic distance in % of diameter and the y-axis the percentage of correspondences that lie within a given geodesic radius around the correct node. Our SplineCNN achieves the highest accuracy for low geodesic error and significantly outperforms other general approaches like MoNet, GCNN and ACNN. In Figure (c), three examples of the FAUST test dataset with geodesic errors of SplineCNN predictions for each node are presented. We show the best (left), the median (middle) and worst (right) test example, sorted by average geodesic error.

100 epochs with a batch size of 1, initial learning rate 0.01 and dropout probability 0.5, using the Adam optimizer [11] and cross entropy loss.

Discussion. Obtained accuracies for different geodesic errors are plotted in Figure 7. The results for different SplineCNN parameters match the observations from before, where only small differences could be seen but using Cartesian coordinates and small B-spline degrees seemed to be slightly better. Our SplineCNN outperforms all other approaches with 99.20% of predictions on the test set having *zero* geodesic error. However, the global behavior over larger geodesic error bounds is slightly worse in comparison to FMNet [16]. In Figure 7c it can be seen that most nodes are classified correctly but that the few false classifications have a high geodesic error. We contribute this differences to the varying loss formulations. While we train against a one-hot binary vector using the cross entropy loss, FMNet trains using a specialized soft error loss, which is a more geometrically meaningful criterion that punishes geodesically far-away predictions stronger than predictions near the correct node [16]. However, it is worth highlighting that we do not use SHOT descriptors as input features, like all other approaches we compare against. Instead, we train only on the geometric structure of the meshes.

Performance We report an average forward step runtime of 0.043 seconds for a single FAUST example processed by the suggested SplineCNN architecture ($k_1 = k_2 = k_3 = 5$, $m = 1$) on a single NVIDIA GTX 1080 Ti. We train this network in approximately 40 minutes. Regarding scalability, we are able to stack up to 160 SConv($(5, 5, 5)$,64,64) layers before running out of memory on the mentioned GPU, while the runtime scales linearly with the number of layers. However, for this task we do not observe significant improvement in accuracy when using deeper networks.

6. Conclusion

We introduced SplineCNN, a spline-based convolutional neural network with a novel trainable convolution operator, which learns on irregular structured, geometric input data. Our convolution filter operates in the spatial domain and aggregates local features, applying a trainable continuous kernel function parametrized by trainable B-spline control values. We showed that SplineCNN is able to improve state-of-the-art results in several benchmark tasks, including image graph classification, graph node classification and shape correspondence on meshes, while allowing very fast training and inference computation. To conclude, SplineCNN is the first architecture that allows deep end-to-end learning directly from geometric data while providing strong results. Due to missing preprocessing, this allows for even faster processing of data.

In the future we plan to enhance SplineCNNs by concepts known from traditional CNNs, namely recurrent neurons for geometric, spatio-temporal data or dynamic graphs, and un-pooling layers to allow encoder-decoder or generative architectures.

Acknowledgments

This work has been supported by the *German Research Association (DFG)* within the Collaborative Research Center SFB 876, *Providing Information by Resource-Constrained Analysis*, projects B2 and A6. We also thank Pascal Libuschewski for proofreading and helpful advice.

References

[1] F. Bogo, J. Romero, M. Loper, and M. J. Black. FAUST: Dataset and evaluation for 3D mesh registration. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3794 –3801, 2014.

[2] D. Boscaini, J. Masci, E. Rodolà, and M. Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3189–3197, 2016.

[3] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, pages 18–42, 2017.

[4] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR)*, 2014.

[5] F. R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[6] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3837–3845, 2016.

[7] I. S. Dhillon, Y. Guan, and B. Kulis. Weighted graph cuts without eigenvectors: A multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1944–1957, 2007.

[8] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 1263–1272, 2017.

[9] M. Henaff, J. Bruna, and Y. LeCun. Deep convolutional networks on graph-structured data. *CoRR*, abs/1506.05163, 2015.

[10] V. G. Kim, Y. Lipman, and T. Funkhouser. Blended intrinsic maps. *ACM Trans. Graph.*, 30(4):79:1–79:12, July 2011.

[11] D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[12] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[13] I. Kokkinos, M. M. Bronstein, R. Litman, and A. M. Bronstein. Intrinsic shape context descriptors for deformable shapes. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 159–166, 2012.

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[15] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein. CayleyNets: Graph convolutional neural networks with complex rational spectral filters. *CoRR*, abs/1705.07664, 2017.

[16] O. Litany, T. Remez, E. Rodolà, A. M. Bronstein, and M. M. Bronstein. Deep functional maps: Structured prediction for dense shape correspondence. In *IEEE International Conference on Computer Vision (ICCV)*, pages 5660–5668, 2017.

[17] J. Masci, D. Boscaini, M. M. Bronstein, and P. Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *IEEE International Conference on Computer Vision Workshop (ICCV)*, pages 832–840, 2015.

[18] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5425–5434, 2017.

[19] L. Piegl and W. Tiller. *The NURBS Book*. Springer-Verlag New York, Inc, 1997.

[20] K. T. Schütt, P.-J. Kindermans, H. E. Sauceda, S. Chmiela, A. Tkatchenko, and K.-R. Müller. SchNet: A continuous-filter convolutional neural network for modeling quantum interactions. In *Advances in Neural Information Processing Systems (NIPS)*, pages 992–1002, 2017.

[21] P. Sen, G. M. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008.

[22] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013.

[23] M. Simonovsky and N. Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 29–38, 2017.

[24] F. Tombari, S. Salti, and L. Di Stefano. Unique signatures of histograms for local surface description. In *Proceedings of the 11th European Conference on Computer Vision (ECCV)*, pages 356–369, 2010.

[25] L. Yi, H. Su, X. Guo, and L. J. Guibas. SyncSpecCNN: Synchronized spectral CNN for 3D shape segmentation. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6584–6592, 2017.

CVPR2018 Paper Translation

姓名：宋溢涛

学号：2016303224

班号：10011605

样条卷积神经网络（SplineCNN）：使用连续 B 样条的卷积核的快速几何深度学习

Matthias Fey* , Jan Eric Lenssen* , Frank Weichert, Heinrich Muller ¨

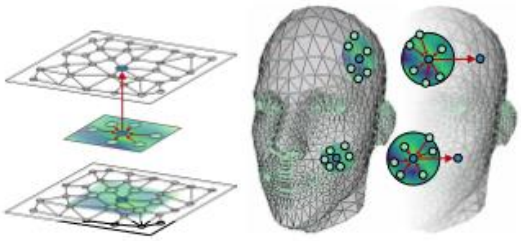
Dortmund 大学计算机图形系

摘要

我们提出了，基于样条的卷积神经网络 SplineCNN，它是一个用于不规则结构和几何输入的深度神经网络的变体。我们的主要贡献是一个基于 B 样条的新型卷积算子，由于 B 样条基函数的局部支持性质，使得其计算时间与核大小无关。因此，我们利用固定数量的训练权重的连续核函数对传统的 CNN 卷积算子进行了推广。与谱域中的相关方法相比，所提出的方法只在空间域中聚集特征。此外，SplineCNN 允许对深层体系结构进行完整的端到端训练，只使用几何结构作为输入，而不是人工提取的特征。为了验证，我们将该方法运用在了图像分类、形状对应和图的节点分类三个方面，发现它比现有方法性能优越或接近最新水平，同时显著加快、且具有如与领域无关的优点。我们的源代码在 GITHUB（https：//github.com/rustyls/pytorch_geometric）上可以找到。

1.简介

过去几年，深层学习方法所获得的大部分成果都严重依赖于卷积神经网络中卷积运算的性质[14]：局部连通性、权重共享和移位不变性。由于这些卷积层是定义在具有网格状结构的输入上的，所以对于如离散流形或嵌入图等非欧域来说，它们不能普遍地移植。然而，实际任务中的大量数据天然以不规则结构的形式出现，例如图形数据或网格。将传统卷积神经网络的高性能转换为这类数据在几个相关任务中具有巨大的改进潜力。



a.图的过滤　　b.网格的过滤

图 1.在几何深度学习中具有可训练的连续核函数的空间聚集的例子，（a）为图的方法和（b）为网格的方法。

最近，出现了一组结合在几何深度学习方面[3]下的方法，其目的是通过为能够处理不规则输入数据的深度神经网络进行去卷积操作来实现这种转换。现有工作可以大致分为两个不同的子集：谱变换方法和空间变换方法。前者基于谱图理论[5]，其中图的拉普拉斯矩阵的特征值被解释为节点信号的频率[22]。它们与传统信号的傅立叶域滤波类似。后一个子集，即空间方法，在局部欧几里得邻域中执行卷积，即点之间的局部位置关系，例如表示为极坐标、球坐标或笛卡尔坐标，如图 1 中的示例所示。

贡献：我们提出了基于样条的卷积神经网络（SplineCNN），它是用于不规则结构数据的深度神经网络的变体。其主要贡献是可训练的、空间上的、连续的卷积核，它利用 B-样条基的性质来科学地输入任意维数的几何信息。我们证明了我们的方法：

•可以应用于不同类型的不规则结构化数据，例如任意（嵌入）图形和网格；

•允许不使用手工特征描述符进行端到端训练；

•提升或追上了几何学习任务的最新进展。

此外，我们提供了一个有效的 GPGPU 算法和实现，允许快速训练和推理计算。

2.相关工作

对于图的深度学习。几何深度学习的历史开始于将卷积神经网络推广到图形输入的尝试。大量的成功方法是基于谱图理论的。Bruna 等人[4]在谱图上引入卷积算子，将拉普拉斯算子的特征向量解释为傅立叶基。作为一个扩展，Henaff 等人[9]建议在谱域中使用样条插值来平滑核。Defferard 等人[6]用切比雪夫多项式逼近滤波器，给出了一种更为有效的变换算法，其核大小决定了聚集的局部 K 邻域的范围。Kipf 和 Welling[12]进一步简化了这种方法，他们只考虑了单邻的滤波应用。Levie 等人[15]提出了一种用于切比雪夫逼近的基于 Caley 变换的滤波器。再加上一个可训练的缩放参数，这将产生一个更稳定和易被激发的滤波器。

应当注意，所有这些频谱方法都假定信息仅编码在输入的连通性、边缘权重和节点特征中。虽然这对于一般图是真实的，但对于嵌入的图或网格是不成立的，其中附加信息是由节点的相对位置给出的，这是我们用我们的方法考虑的。

许多谱方法的缺点是它们使用依赖于域的傅立叶基，这限制了对具有相同图连通性的输入的推广。Yi 等人[25]通过应用同步频谱域的频谱变压器网络来解决这个问题。由于我们的方法直接工作在空间域，它不容易出现这个问题。

for the 形网：我们的任务，我们也 litany Analyze in this work，等。[16] present a 暹罗软错误使用标准的网络节点之间的基于测地线的距离。我们比较我们的方法在这样的专业方法。

对于网格上的形状对应任务，我们也进行了分析。Litany 等人[16]提出了一种基于节点间测地线距离的软误差准则的暹罗网络。我们将我们的方法与这个特殊的方法比较。

离散流形的局部描述。不表示局部位置关系的问题可以通过使用从离散流形中提取局部欧几里德邻域的表示的方法来解决。

基于 Kokkinos 等人[13]的固有形状描述符，Masci 等人[17]提出了一种从网格中提取二维欧几里德邻域的方法，并提出了局部应用于这些邻域的卷积运算。Boscaini 等人[2]通过引入基于输入网格的局部主曲率的块旋转方法来对齐提取的块，从而改进了这种方法。

我们的卷积算子可以但不必接收这些局部表示作为输入。因此，我们的方法与此领域的改进可同时使用。

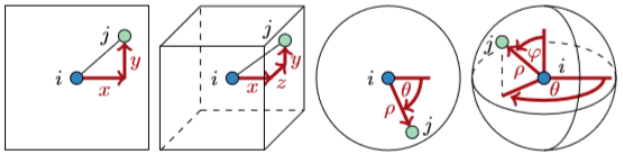
空间连续卷积核。尽管图的第一个连续卷积核在谱域中有效(如[9,6,20])，但不规则结构数据的空间连续卷积核最近作为神经消息传递和自注意机制领域的一个特例被引入[8,23,18]。此外，Monti 等人[18]提出了 MoNet 框架，用于将不同类型的输入解释为有向图，这是我们在工作中所建立的基础。结果表明，我们的核与 MoNet 的可训练高斯混合模型(GMM)核具有相同或更好的准确率，同时可以直接对几何结构进行训练。

3. SplineCNN

我们将 SplineCNNs 定义为一种使用一种新的基于样条的卷积层来构建的深度神经网络。这个层接收不规则的结构化数据，这些数据被映射到有向图，作为输入。在空间卷积层中，节点特征使用可训练的连续核函数进行聚合，我们在本节中对此进行了定义。

3.1 准备工作

输入图。与 Monti 等人[18]的工作相似，我们期望卷积算子的输入是一个有向图 G = (V,E,U)， V ={1，…N }作为节点集，E⊆V×V 作为边集，U∈$[0,1]^{N•N•d}$ 。对于每条有向边(i,j)∈E，包含 d 维的伪坐标 u(i,j)∈$[0,1]^d$ 。注意 U 可以被解释为一个 d 维的邻接矩阵，规范化后 U(i,j)∈$[0,1]^d$ 如果(i,j)∈E，否则为 0。另外，U 通常是稀疏的，E = | E |



$\mathbf{u}(i,j)=(x,y)$ $\mathbf{u}(i,j)=(x,y,z)$ $\mathbf{u}(i,j)=(\rho,\theta)$ $\mathbf{u}(i,j)=(\rho,\theta,\varphi)$

≪N^2 条目。对于节点 i∈V，它的邻集由 N(i)决定。

图 2:伪坐标 u 的可能性:二维和三维笛卡尔坐标、极坐标和球面坐标。忽略坐标 u 到区间$[0,1]^d$ 的缩放和平移的值。

输入节点的特征。对没个节点 i∈V，令 f:V→ $R^{M_{in}}$,f(i) ∈ $R^{M_{in}}$,表示一个向量在 M_{in} 的输入特征。对于 1≤l ≤ M_{in}，我们引用集{ $f_l(i)$ |i∈V }作为输入特征映射。

b 样条基函数。除了输入图和节点特性,用 $((N_{1,i}^m)_{1\le i\le k_1},\ldots,(N_{d,i}^m)_{1\le i\le k_d})$ 表示 d 开放程度的 m 度 b 样条,基于统一的,即等距、等节点向量(由 Piegl et al 提出[19]),k =(k1,…,kd)定义 d 维卷积核大小。

3.2. 主要概念

我们的卷积算子通过一个可训练的连续核函数将局部邻域的节点特征进行加权。节点特性 f(i)表示在一个不规则的几何结构的特性,其空间关系是由 U 中的伪坐标局部定义的。因此,当特性值局部聚集在一个节点的邻结点时,U 决定特征聚合的状况，而 f(i)的内容定义了聚合的对象。我们认为，几何深度学习任务的常见输入可以被映射到该模型，同时保留相关信息:

•对于**图**，V 和 E 是给定的，U 可以包含边缘权值，或者如目标节点的节点度。

•对于**离散流形**，V 包含离散流形的点，E 表示局部欧几里德邻域的连通性,U 可以包含目标点相对于每条边的原点的极坐标、球面坐标或笛卡尔坐标等局部关系信息。

我们要求 U 是一个固定区间范围的元素，除此对 U 的值没有任何限制。因此，举例来说，运用由 Boscaini 等人的工作得到的局部欧几里德邻域[2]，网格可以被解释为嵌入的三维图

形或二维流形。同样，也可以使用极坐标/球坐标或笛卡尔坐标，如图 2 所示。我们的可训练的连续核函数(我们将在下一节中定义)与存储在 U 中的坐标类型无关，将每个 U (i, j)映射到一个标量，该标量即为特性聚合的权重。

3.3. 卷积算子

我们从使用 b 样条基定义连续核函数开始，b 样条基使用一定数量的可训练控制值参数化。b 样条基函数[19]的局部支持特性表明，基函数在已知区间之外的所有输入值都为 0，这对于高效的计算和可伸缩性是有利的。

图 3 显示了不同 b 样条基度 m 下的核构造方法。我们对 b 样条基和每个输入特征映射 M_{in}，按 l 索引的笛卡儿积 $P=(N_{1,i}^{m})_i\times\ldots\times(N_{d,i}^{m})_i$ 的每一个元素 p，引入一个可训练的参量 $w_{p,l}\in W$ 。这将得到

$K=M_{in}\bullet\prod_{u=1}^{d}k_i$ 个可训练参数。

我们将我们的连续卷积核定义为

$$g_l:[a_1,b_1]\times\ldots\times[a_d,b_d]\to R,$$

$$g_l(u)=\sum_{p\in P}w_{p,l}\bullet B_p(u), \qquad (1)$$

B_p 是 p 中基函数的乘积：

$$B_p(u)=\prod_{i=1}^{d}N_{i,p_i}^{m}(u_i)。 \qquad (2)$$

将节点 i 的空间卷积算子定义为：

$$(f*g)(i)=\frac{1}{|N(i)|}\sum_{l=1}^{M_{in}}\sum_{j\in N(i)}f_l(j)\bullet g_l(u(i,j)) \qquad (3)$$

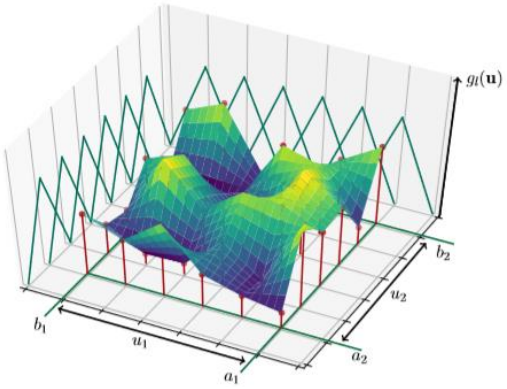
与传统的 CNN 相似，卷积算子可以作为深度神经网络结构中的一个模块，我们在我们的分支网络中就是这样做的。为此，对具有不同可训练参数的相同输入数据应用算子 M_{out} 次，得到产生 M_{out} 个输出特征映射的卷积层。应该强调的是，与自我关注方法不同，我们为输入和输出特征图的每个组合训练了一组单独的权重。

本地支持。由于 B 样条函数的本地支持属性，对于 K 个不同的向量 $p\in P$，$B_p\neq 0$ 只适用于 $s:=(m+1)^d$ 。因此，gl(u)只依赖于每个邻居 j 的 M_{in} ·K 个可训练参数中的 M_{in} ·s 个，其中 s、d 和 m 是常数，通常很小。此外，对于每一对节点(i, j)∈E,向量 p∈P 且 $B_p\neq 0$,我们表示为 P(u(i, j)),可以在常数时间,找到常数 m 和 d。

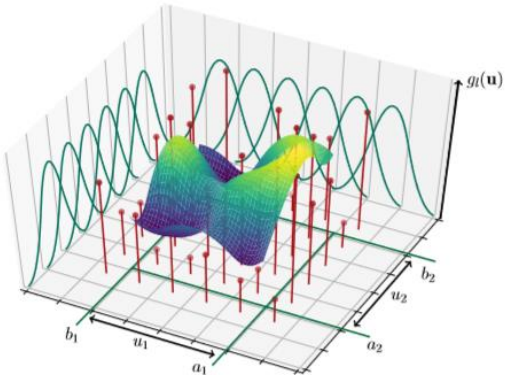
这就可以用另一种形式来表示卷积运算的内和，即公式 3：

$$(f_l*g_l)(i)=\sum_{\substack{j\in N(i)\\ p\in P(u(i,j))}}f_l(j)\bullet w_{p.l}\bullet B_p(u(i,j)) \qquad (4)$$

在处理的时间复杂度上可以用 s 代替 K。同样，Bp(u(i, j))



(a)线性 B 样条基函数



(b)二次 B 样条基函数

图 3：连续卷积核的例子。b 样条基度(a) m = 1 (b) m = 2，核维数 d = 2。红点的高度是单个输入特征图的可训练参数。在影响核值之前，它们将乘以 b 样条张量积基的元素。

不依赖于 1，因此可以通过对所有输入特征计算一次得出。图 4 显示了计算方案。向后通过的梯度流也可以通过向后按照实心箭头得到。

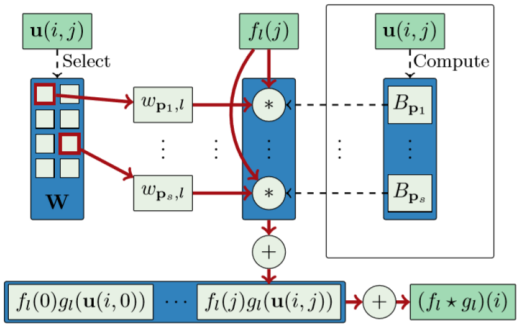


图 4：卷积运算的正演计算方法。在 BP 算法的反向传播步骤中，梯度沿着倒置的实心箭头流动，到达 W 和 fl(i)的输入。

封闭 B 样条。根据向量 u 中坐标的类型，我们在某些维上使用闭 b 样条近似。这种情况经常出现的一个例子是 u 包含极坐标的角度属性。使用封闭的 b 样条逼近角自然维度，在计算角 0 时与计算角 2π 具有相同权重，对于更高的 m，核函数只在连续可微的那些点存在。

该核可以很容易地通过将不同的 p∈P 映射到相同的可训练的控制值 $w_{p,l}$ 进行修改，以便在任意 d 维子集近似。这导致可训练参数和 b 样条基函数的减少。参考图 3，这种方法可以理解为函数曲面沿相应轴的周期性重复。

根节点。到目前为止，我们在卷积算子中没有考虑邻域 N(i)的节点 i。它并不和所有 j∈N(i)聚合在一起，就像在传统的 CNN 中。如果使用笛卡尔坐标，我们可以简单地定义 N(i)来包含 i，但是，当使用极/球面伪坐标时，由于没有很好地定义半径为零的点，会出现问题。因此，我们为根节点的每个特性引入一个额外的可训练参数，并将该参数的乘积和相应的特性添加到结果中。

与传统 CNNs 的关系。除归一化因子外，我们的基于样条的卷积算子是传统的 cnn 卷积层的推广，每个维度上都有奇滤波。例如，假设有一个二维网格图，具有对角线、水平和垂直的边的输入，b 样条的度 m = 1，内核大小(3，3)，向量 u 包含相邻节点之间的笛卡尔关系，那么我们的卷积算子相当于用大小 3×3 的一个内核对图像进行的一个离散卷积。如果网格图的邻域被相应地修改，那么对于较大的离散内核也是如此。

4. GPGPU(通用图形处理器)算法

对于上一节中定义的基于样条的卷积层，我们将介绍一种 GPU 算法，它允许使用 SplineCNN 进行有效的训练和推断。为了简单起见，我们使用一个张量索引符号，例如，A[x, y, z]来描述第 3 级张量 A 的位置在(x, y, z)的元素。在算法 1 中概述了卷积算子的前向运算。

算法 1：与 b 样条核的几何卷积

输入：

N：结点数

M_{in}：每结点输入特征数

M_{out}：每结点输出特征数

$s=(m+1)^d$：每条边非 0 的 Bp 的数量

$W \in R^{K \times M_{in} \times M_{out}}$：可训练的权重

$B \in R^{E \times s}$：每条边的 s 权重的基础乘积

$P \in N^{E \times s}$：W 中每条边的 s 权系数

$F_{in} \in R^{N \times M_{in}}$：每个结点的输入特征

输出：

$F_{out} \in R^{N \times M_{out}}$：每个结点的输出特征

Gather $\mathbf{F}_{\text{in}}^{E}$ from $\mathbf{F}_{\text{in}}$ based on target nodes of edges
Parallelize over $e \in \{1, \ldots, E\}, o \in \{1, \ldots, M_{\text{out}}\}$:
  $r \leftarrow 0$
  for each $i \in \{1, \ldots, M_{\text{in}}\}$ **do**
    for each $p \in \{1, \ldots, s\}$ **do**
      $w \leftarrow \mathbf{W}[\mathbf{P}[e, p], i, o]$
      $r \leftarrow r + (\mathbf{F}_{\text{in}}^{E}[e, i] \cdot w \cdot \mathbf{B}[e, p])$
    end for
  end for
  $\mathbf{F}_{\text{out}}^{E}[e, o] \leftarrow r$
Scatter-add $\mathbf{F}_{\text{out}}^{E}$ to $\mathbf{F}_{\text{out}}$ based on origin nodes of edges
Return $\mathbf{F}_{\text{out}}$

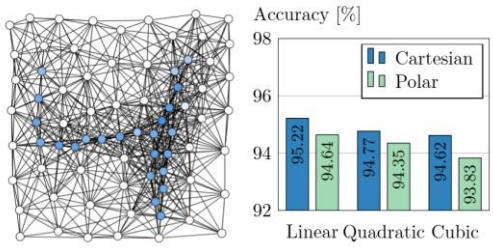
我们首先从输入矩阵 $F_{in} \in R^{N \times M_{in}}$ 收集边的输入特征 $F_{in}^{E} \in R^{E \times M_{in}}$，使用每条边的目标结点作为索引，实现在边 e 上的并行化。然后，我们计算了边的输出特征，$F_{out} \in R^{N \times M_{out}}$，如图 4 所示。在将它们添加进结点的特征 $F_{out} \in R^{N \times M_{out}}$ 之前，先执行实际的邻结点聚合。假设散点的增加是一个具有恒定时间复杂度的并行操作，我们的算法具有 O(s · M_{in})的并行时间复杂度，对于较小的 s，复杂度为 O(E · M_{out})。

计算 b 样条基。通过计算矩阵 $P \in N^{E\times s}$ 和 $B \in R^{E\times s}$，我们摆脱了可训练权重的数量。P 包含参数的指标，$B_p \neq 0$，B 包含这些参数的基乘积 B_p。对于给定的图形结构，可以对 B 和 P 进行预处理，也可以直接在内核中计算。对于 B 所需要的基函数的 GPU 评估，我们为每个 m 使用这些函数的显式低次多项式公式。要了解更多细节，可以参考 GitHub 上的 PyTorch 实现。

小批次处理。对于批处理学习，可以通过在一个批处理的全集 U 中创建稀疏块对角线矩阵，并在节点维上连接矩阵 F_{in} 来实现对一个小批处理的并行化。对于 F_{in}^{E}、B 和 P 矩阵，这将导致边界维数的级联。注意，这种组合允许在一个批处理中对示例使用不同数量的节点和边，而不会引入冗余的计算开销。

5.结果

从图像图分类（5.1 节）、图结点分类（5.2 节）和网格形状对应（5.3 节）三个方面，我们对不同的 SplineCNN 架构进行了试验。对于每个任务，我们使用基于样条的卷积运算符创建一个 SplineCNN，我们将其表示为 $SConv(k,M_{in},M_{out})$，用于核大小为 k、有个 M_{in} 输入特征映射和 M_{out} 个输出特征映射的一个卷积层。此外，我们将完全连通层表示为 FC(o)， o 表示输出神经元的数量。



(a)MNIST 超像素示例　　(b)分类准确率

图 5：MNIST 75 超像素(a)示例和(b)使用不同伪坐标和 b 样条基的分类精度。

Dataset	LeNet5 [14]	MoNet [18]	**SplineCNN**
Grid	**99.33%**	99.19%	99.22%
Superpixels	–	91.11%	**95.22%**

表 1：对于一个经典的 CNN (LeNet5)， MoNet 和我们的 SplineCNN 方法，MNIST 数据集(网格和超像素)的不同表示的分类精度。

5.1 图像图分类

为了验证二维规则和不规则的结构化输入数据，我们将我们的方法应用于众所周知的 MNIST 数据集[14]，它包含 60000 个训练和 10000 个包含 10 个不同类的灰度手写数字的测试图像。我们在 MNIST 进行了两个不同的实验。对于两个实验，我们严格遵循 Defferrard 等人和 Monti 等人[6,18]的实验设置来提供可比性。第一个实验,MNIST 图像表示为一组相同的网格图,其中每个节点对应于原始图像中的一个像素,28×28 的网格中总共有 N = 28×28 = 784 个节点。在第二个实验中,我们使用了 Monti 等人的 MNIST 超像素数据集，其中每幅图像都被表示为一个包含 75 个节点的嵌入式图形，这些结点定义了超像素数据集的矩心，如图 5a 所示，每个图形都具有不同的结点位置和连接性。这个实验是验证我们的方法在不规则的基于图像的结构化数据上的能力的理想方案。

池化。我们的 SplineCNN 架构使用了基于 Graclus 方法的池操作符[7,6]。池操作可以通过在图节点上派生集群、在一个集群中聚合节点并为每个新节点计算新的伪坐标来获得粗化的图。我们使用 MaxP(c)算法表示一个最大池化层，c 是集群大小(和近似降尺度因子)。

体系结构和参数。在网格图实验中，采用笛卡尔坐标和 m = 1 的 b 样条基度,达到了 CNNs 传统卷积算子的等价性,参见 3.3 节。相比之下，我们在超像素数据集中相互比较了 m 和可能的伪坐标的所有配置。

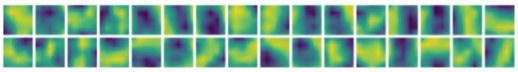
对于网格数据的分类,我们利用 LeNet5-like 网络架构[14]:SConv((5,5),1,32)→MaxP(4)→SConv((5,5),32,64)→MaxP(4)→FC(512)→FC(10)。最初的学习速率被选为 1e-3 和失活概率为 0.5。注意,我们使用大小 5×5 的网格邻域,反映 LeNet5 架构的 5×5 过滤器。超像素数据集评估使用 SplineCNN 架构 SConv((k1,k2),1,32)→MaxP(4)→SConv((k1,k2),32,64)→MaxP(4)→AvgP→FC(128)→FC(10),其中 AvgP 表示一层节点平均特征的维度。我们使用指数线性单位(ELU)作为每个 SConv 层和第一个 FC 层之后的非线性激活函数。对于笛卡尔坐标系，我们选择核大小为 k1=k2=4+m，而对于极坐标，选择 k1=1+m 和 k2=8。进行 20 周期训练，批大小 64 个，初始学习率为 0.01，失活率为 0.5。这两个网络都使用 Adam 方法[11]进行了 30 周期训练。

讨论。MNIST 实验的所有结果如表 1 和图 5b 所示。网

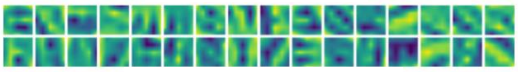
格图实验结果与 LeNet5 和 MoNet 方法的精度基本一致。对于超像素数据集，我们将之前的结果提高了 4.11 个百分点。由于我们使用与 MoNet 类似的体系结构和相同的输入数据，因此更好的结果表明我们的算子能够在输入结构中捕获更多相关信息。这可以用这样一个事实来解释:与 MoNet 内核不同，我们的内核函数对于输入和输出特征映射的每个组合都有单独的可训练权值，就像传统 CNNs 中的过滤器一样。

不同配置的结果如图 5b 所示。对于变化的 m 和伪坐标，我们只发现了精度上的微小差异。然而，在较低的 m 和使用笛卡尔坐标时的性能比其他配置略好。

此外，我们从图 6 的网格和超像素实验中可视化了第一个 SConv 层的 32 个学习内核。可以观察到，无论是在规则的还是不规则的结构化数据上进行训练，边缘检测模式都是在两种方法中习得的。



(a)MNIST 网格实验



(b)MNIST 超像素实验

图 6：在 MNIST 网格(a)和 超像素数据集(b)上训练的第一个基于样条卷积层的 32 个内核的可视化，其核大小(5,5)和 b 样条基度 m = 1。

ChebNet [6]	GCN [12]	CayleyNet [15]	SplineCNN
87.12 ± 0.60	87.17 ± 0.58	87.90 ± 0.66	**89.48 ± 0.31**

表 2:不同学习方法(ChebNet, GCN, CayleyNet 和 SplineCNN)在 Cora 数据集中的图节点分类。在 100 多次实验中计算出了所提的精度均值和标准差，每个实验的网络训练时间为 200 周期。

5.2 图结点分类

作为第二个实验，我们使用 Cora 引文图[21]来解决图节点分类问题。我们验证了我们的方法在没有给出欧几里德关系的数据集上也有很强的性能。Cora 由 2,708 个节点和 5,429 个无向非加权边组成，分别代表科学出版物和引文链接。每个文档都由一个 1433 维的稀疏二进制字袋特征向量单独表示，并被标记为 7 个类中的一个。与 Levi 等人[15]中的实验设置类似，我们将数据集分为 1708 个节点进行训练，500 个节点进行测试，来模拟标记和未标记的信息。

体系结构和参数。我们使用 SplineCNN 类似于[15、12、18]中介绍的网络体系结构:SConv((2),1433,16)→SConv((2),16,7)，第一个 SConv 层后使用 ELU 激活函数，m = 1。对于伪坐标，我们选择目标节点 $u(i,j)=(\deg(j)/\max_{v\in V}\deg(v))$ 的全局归一化程度，实现根据相邻出版物的站点数量进行过滤。采用 Adam 优化方法[11]进行 200 周期的训练，学习率 0.01，失活率 0.5,L2 正则化 0.005。损耗函数采用了网络 softmax 输出与独热目标分布的交叉熵。

讨论。我们的方法和相关方法的结果如表 2 所示，平均分类准确率超过 100 次。可以看出，SplineCNN 在实验中提高了大约 1.58 个百分点。我们将此改进归功于于 u 的过滤，它包含了节点度作为附加信息，以学习更复杂的内核函数。这表明，SplineCNN 可以成功地应用于不规则但非几何的数据中，并且可以改进这一领域以前的结果。

5.3 网格形状对应

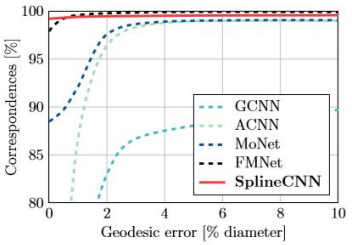
作为我们最后的也是最大的实验，我们在一个三维网格集合上验证了我们的方法，这个集合解决了类似于[18,2,17,16]中的形状的对应任务。形状对应是指将给定形状的每个节点标记为参考形状[17]的相应节点。我们使用 FAUST 数据集[1]，包含 10 个扫描人体形状的 10 种不同的姿势，导致总共 100 个非水密网格与 6890 个节点。FAUST 的前 80 个数据被用于训练，剩下的 20 个数据被用于测试，与[18]中的数据集分割相同。我们隐式给出了 FAUST 网格的地面真值对应关系，其中每个实例的节点排序顺序完全相同。对应的质量是根据 Princeton 基准协议[10]来度量的，计算在正确节点周围的测量的半径 r 范围内的派生对应的百分比。

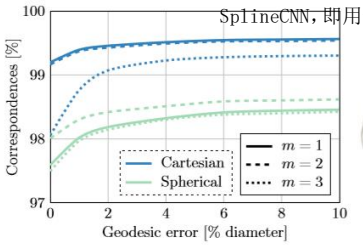
与类似的方法(如[18,2,17,16])不同的是，我们不手工制作特征描述符作为输入，比如被称为 SHOT 描述符[24]的法向量的局部直方图，并使网络从几何(即 U 编码的空间关系)本身学习。因此，输入特性平凡地给定为 $1\in R^{N\times 1}$。同时，以三维网格为输入，而不是为每个节点生成二维测量线的块，验证了该方法的有效性。这些简化减少了大量预处理数据所需的计算时间和内存消耗，使训练和推断完全端到端并且非常高效。

体系结构和参数。我们应用 SplineCNN 架构与 6 卷积层:SConv((k1,k2,k3),1,32)→SConv((k1,k2,k3),32,64)→4×SConv((k1,k2,k3),64,64)→Lin(256)→Lin(6890)，Lin(o)代表一个 1×1 卷积层 o 特性输出的每个节点。在每个 SConv 和第一 Lin 层之后使用 ELU 作为非线性激活函数。对于笛卡尔坐标系，我们选择核大小为 k1 = k2 = k3 = 4 + m，对于极坐标使用 k1 = k3 = 4 + m 和 k2 = 8。我们在 m = {1,2,3} 的多项选择上评估我们的方法。使用 Adam 优化[11]和交叉熵损失进行了 10 个周期的训练，批大小为 1，初始学习率为

0.01，失活率为0.5。

讨论。不同测量线误差得到的精度如图7所示。不同SplineCNN参数的结果与之前的观察结果相吻合，虽然只有很小的差异，但是使用笛卡尔坐标和小b样条度似乎更好一些。我们的SplineCNN在测试集中的99.20%的预测均为零测量误差，超出了其他所有方法。然而，与FMNet[16]相比，在较大的测量线误差范围内的全局行为略差。从图7c中可以看出，大多数节点分类都是正确的，但少数错误的分类具有较高的测量误差。我们将这种差异归于不同的损失公式。当我们使用交叉熵损失对独热的二进制向量进行训练时，FMNet使用一种特殊的软误差损失进行训练，这是一种更有几何意义的准则，它更强地惩罚了与正确节点[16]距离更远的测量线预测。然而，值得强调的是，我们没有使用SHOT描述符作为输入特性，就像我们比较的所有其他方法一样。相反，我们只训练网格的几何结构。





(a)SplineCNN 和其他方法的结果　(b)不同的 SplineCNN 的结果　(c)测试样例的测量误差

图 7：SplineCNN 和相关方法的形状对应实验的测量线误差图(a);不同 SplineCNN 的形状对应实验的测量线误差图(b）。x 轴以直径的百分比表示测量距离，y 轴表示在给定的测量半径范围内，在正确节点周围的对应比例。我们的 SplineCNN 在低测量线误差的情况下达到了最高的精度，并且显著地超过了其他一般的方法，如 MoNet，GCNN 和 ACNN。在图(c)中，给出了三个 FAUST 测试数据集的例子，这些数据集的测量线误差与每个节点的 SplineCNN 预测有关。我们展示了最好的(左)、中值(中)和最差的(右)测试示例，并按照平均测量线误差进行排序。

表现。我们发现在一个NVIDIA GTX 1080 Ti上由建议的SplineCNN架构(k1 = k2 = k3 = 5, m = 1)处理的FAUST示例的平均前步运行时间为0.043秒。我们在大约40分钟内训练了这个网络。关于可伸缩性，在上述GPU上耗尽内存之前，我们可以将SConv((5,5,5),64,64)层堆叠起来，而运行时随着层的数量线性扩展。然而，对于这项任务，我们没有观察到在使用更深层的网络时精度有显著的提高。

6.结论

我们介绍了SplineCNN，这是一个基于样条的卷积神经网络，带有一个新的可训练卷积算子，可以学习不规则的结构化的几何输入数据。我们的卷积滤波器在空间域内工作，并聚集局部特征，应用可训练的连续核函数，由可训练的b样条控制值参数化。结果表明，SplineCNN能够在图像图分类、图节点分类和网格形状对应等几个基准任务中提高状态结果，同时能够进行非常快速的训练和推理计算。总之，SplineCNN是第一个能够直接从几何数据进行深度端到端学习的体系结构，同时提供了强劲的结果。由于缺少预处理，这允许更快地处理数据。

在未来，我们计划通过传统CNN的概念来增强SplineCNN，即用于几何、时空数据或动态图的周期性神经元，以及用于编码器、解码器或生成架构的非池化层。

鸣谢

这项工作获得了 the Collaborative Research Center SFB 876 下的 the German Research Association (DFG)的支持 ，获得了 Resource-Constrained Analysis, projects B2 和 A6 的信息提供。我们也感谢 Pascal Libuschewski 的校对和有用的建议。

参考文献

[1] F. Bogo, J. Romero, M. Loper, and M. J. Black. FAUST: Dataset and evaluation for 3D mesh registration. In Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 3794 –3801, 2014.

[2] D. Boscaini, J. Masci, E. Rodol`a, and M. Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In Advances in Neural Information Processing Systems (NIPS), pages 3189–3197, 2016.

[3] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: Going beyond euclidean data. IEEE Signal Processing Magazine, pages 18– 42, 2017.

[4] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. In International Conference on Learning Representations (ICLR), 2014.

[5] F. R. K. Chung. Spectral Graph Theory. American Mathematical Society, 1997.

[6] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In Advances in Neural Information Processing Systems (NIPS), pages 3837–3845, 2016.

[7] I. S. Dhillon, Y. Guan, and B. Kulis. Weighted graph cuts without eigenvectors: A multilevel approach. IEEE Transactions on Pattern Analysis and Machine Intelligence, pages 1944–1957, 2007.

[8] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In Proceedings of the 34th International Conference on Machine Learning (ICML), pages 1263–1272, 2017.

[9] M. Henaff, J. Bruna, and Y. LeCun. Deep convolutional networks on graph-structured data. CoRR, abs/1506.05163, 2015.

[10] V. G. Kim, Y. Lipman, and T. Funkhouser. Blended intrinsic maps. ACM Trans. Graph., 30(4):79:1–79:12, July 2011.

[11] D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. In International Conference on Learning Representations (ICLR), 2015.

[12] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In International Conference on Learning Representations (ICLR), 2017.

[13] I. Kokkinos, M. M. Bronstein, R. Litman, and A. M. Bronstein. Intrinsic shape context descriptors for deformable shapes. In Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 159–166, 2012.

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradientbased learning applied to document recognition. In Proceedings of the IEEE, pages 2278–2324, 1998.

[15] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein. CayleyNets: Graph convolutional neural networks with complex rational spectral filters. CoRR, abs/1705.07664, 2017.

[16] O. Litany, T. Remez, E. Rodol`a, A. M. Bronstein, and M. M. Bronstein. Deep functional maps: Structured prediction for dense shape correspondence. In IEEE International Conference on Computer Vision (ICCV), pages 5660–5668, 2017.

[17] J. Masci, D. Boscaini, M. M. Bronstein, and P. Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In IEEE International Conference on Computer Vision Workshop (ICCV), pages 832–840, 2015.

[18] F. Monti, D. Boscaini, J. Masci, E. Rodol`a, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model CNNs. In Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 5425–5434, 2017.

[19] L. Piegl and W. Tiller. The NURBS Book. Springer-Verlag New York, Inc, 1997.

[20] K. T. Sch ütt, P.-J. Kindermans, H. E. Sauceda, S. Chmiela, A. Tkatchenko, and K.-R. M¨uller. SchNet: A continuousfilter convolutional neural network for modeling quantum interactions. In Advances in Neural Information Processing Systems (NIPS), pages 992–1002, 2017.

[21] P. Sen, G. M. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad. Collective classification in network data. AI Magazine, 29(3):93–106, 2008.

[22] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. IEEE Signal Processing Magazine, 30(3):83–98, 2013.

[23] M. Simonovsky and N. Komodakis. Dynamic edgeconditioned filters in convolutional neural networks on graphs. In Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 29–38, 2017.

[24] F. Tombari, S. Salti, and L. Di Stefano. Unique signatures of histograms for local surface description. In Proceedings of the 11th European Conference on Computer Vision (ECCV), pages 356–369, 2010.

[25] L. Yi, H. Su, X. Guo, and L. J. Guibas. SyncSpecCNN: Synchronized spectral CNN for 3D shape segmentation. In Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 6584–6592, 2017.