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

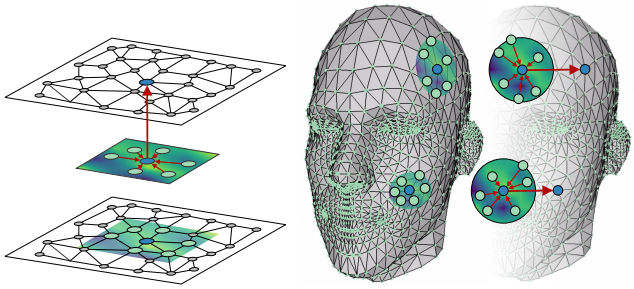
We present Spline-based Convolutional Neural Networks (SplineCNNs)*, a variant of deep neural networks for irregular structured and geometric input,* e.g.*, graphs or meshes. Our main contribution is a novel convolution operator based on B-splines, that makes the computation time independent from the 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 at 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 [13]: 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 [21]. 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*. [**?**] introduced convolution-like operators on spectral graphs, interpreting the eigenvectors of the Laplacian as Fourier basis. As an extension, Henaff *et al*. [8] 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 [**?**], 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*. [14]. 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*. [**?**] 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*. [15] 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*. [12], Masci *et al*. [16] 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.

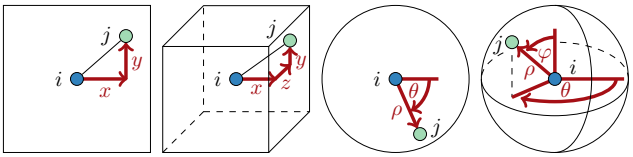
Spatial continuous convolution kernels. While the first continuous convolution kernels for graphs work in the *spectral* domain (*e.g*. [8, 6, 19]), *spatial* continuous convolution kernels for irregular structured data were introduced recently as a special case in the fields of neural message passing and self-attention mechanisms [**?**, **?**, 17]. Furthermore, Monti *et al*. [17] 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*. [17], 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.* [18]), 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 [18], 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 $\mathbf{p}$:

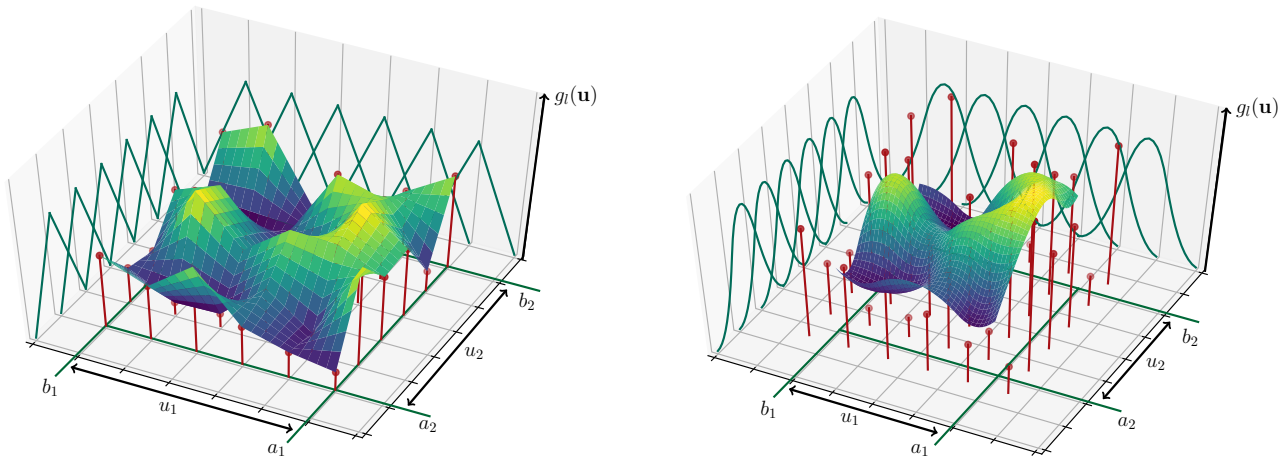
$$B_{\mathbf{p}}(\mathbf{u}) = \prod_{i=1}^{d} N^m_{i,p_i}(u_i). \tag{2}$$

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 [18]. 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

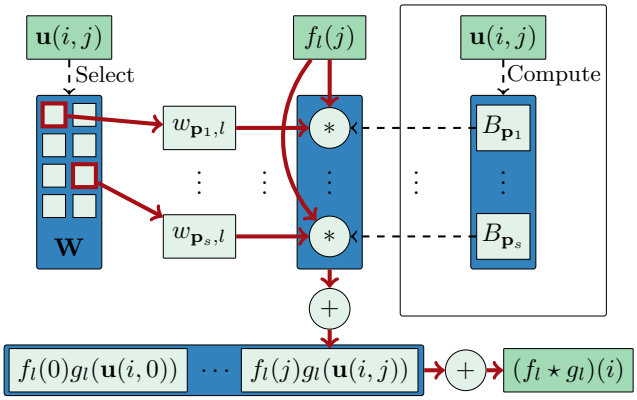


Figure 4: Forward computation scheme of the proposed convolution operation. During the backward step of the backpropagation algorithm, the gradient flows along the inverted solid arrows, reaching inputs from $\mathbf{W}$ and $f_l(i)$.

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

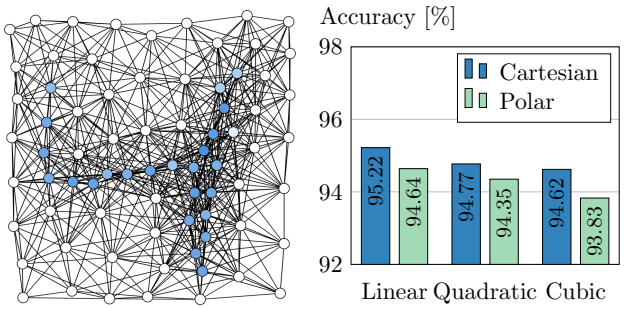
Input:
N: Number of nodes
M_{in}: Number of input features per node
M_{out}: Number of output features per node
$s = (m+1)^d$: Number of non-zero $B_{\mathbf{p}}$ for one edge
$\mathbf{W} \in \mathbb{R}^{K \times M_{\text{in}} \times M_{\text{out}}}$: Trainable weights
$\mathbf{B} \in \mathbb{R}^{E \times s}$: 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 [13] 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, 17] 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*. [17] 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 [13]	MoNet [17]	**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.

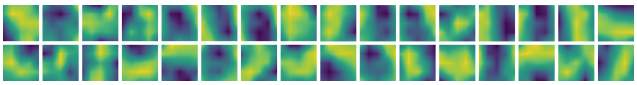
For classification on the grid data, we make use of a LeNet5-like network architecture [13]: SConv($(5, 5)$,1,32) $\rightarrow$ MaxP(4) $\rightarrow$ SConv($(5, 5)$,32,64) $\rightarrow$ MaxP(4) $\rightarrow$ FC(512) $\rightarrow$ FC(10). The initial learning rate was chosen as 10^{-3} and dropout probability 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 [10].

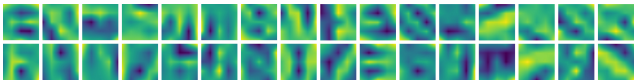
Discussion. All results of the MNIST experiments are shown in Table 1 and Figure 5b. The grid graph experiment results in approximately the same accuracy as LeNet5 and the MoNet method. For the superpixel dataset, we improve previous results by 4.11 percentage points in accuracy. Since we are using a 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 [**?**]	CayleyNet [14]	**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 [20]. 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.* [14], 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 [14, **?**, 17]: 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 [10] 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.

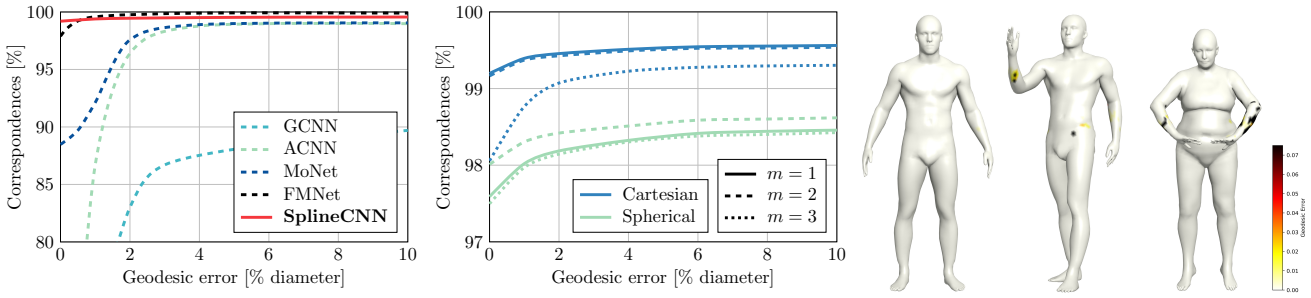
Discussion. Results of our and related methods are shown in Table 2 and report the mean classification accuracy averaged over 100 experiments. It can be seen that SplineCNNs improve the state-of-the-art in this experiment by approximately 1.58 percentage points. We contribute this improvement to the 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 [17, 2, 16, 15]. Shape correspondence refers to the task of labeling each node of a given shape to the corresponding node of a reference shape [16]. 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 [17]. 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 [9], counting the percentage of derived correspondences that lie within a geodesic radius r around the correct node.

In contrast to similar approaches, *e.g.* [17, 2, 16, 15], we go without handcrafted feature descriptors as inputs, like the local histogram of normal vectors known as SHOT descriptors [22], 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 [10] 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 [15]. 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 [15]. 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)*, 2014. 7

[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*, pages 3189–3197. 2016. 2, 3, 7

[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. 1

[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. 2

[5] F. R. K. Chung. *Spectral Graph Theory*. American Mathematical Society. 1

[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*, pages 3837–3845, 2016. 2, 6, 7

[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*. 6

[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 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1263–1272, 2017. 2

[9] M. Henaff, J. Bruna, and Y. LeCun. Deep convolutional networks on graph-structured data. *CoRR*, abs/1506.05163. 2

[10] V. G. Kim, Y. Lipman, and T. Funkhouser. Blended intrinsic maps. *ACM Transactions on Graphics*. 7

[11] D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*. 6, 7, 8

[12] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*. 2, 7

[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. 2

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, 1998. 1, 6

[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. 2, 7

[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)*, 2017. 2, 7, 8

[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. 2, 7

[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)*, 2017. 2, 6, 7

[19] L. Piegl and W. Tiller. *The NURBS Book*. Springer-Verlag New York, Inc. 3

[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*, pages 992–1002, 2017. 2

[21] P. Sen, G. Namata, M. Bilgic, and L. Getoor. Collective classification in network data. *AI Magazine*, pages 93–106, 2008. 7

[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*, pages 83–98, 2013. 1

[23] M. Simonovsky and N. Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 29–38, 2017. 2

[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. 7

[25] L. Yi, H. Su, X. Guo, and L. J. Guibas. SyncSpecCNN: Synchronized spectral CNN for 3D shape segmentation. 2017. 2