Parameterization learning with convolutional neural networks for gridded data fitting

Michele De Vita[1], Carlotta Giannelli[1], Sofia Imperatore[1]**, and Angelos Mantzaflaris[2]

[1] Dipartimento di Matematica e Informatica "U. Dini", Università degli Studi di Firenze, Italy
michele.devita@stud.unifi.it,{carlotta.giannelli,sofia.imperatore}@unifi.it
[2] Centre Inria of the French Riviera University, Sophia Antipolis, France
angelos.mantzaflaris@inria.fr

Abstract. The approximation of point clouds in terms of parametric representations is a fundamental task for geometric modeling and processing applications to properly analyze the (re-)constructed model in its full complexity. A necessary and key step in this process requires the identification of a suitable data parameterization. If the data connectivity is determined by a rectilinear grid, standard closed-form methods are available for general multivariate configurations. Existing data-driven parameterization methods instead are limited to the univariate case and require pre/post-processing steps of the input data to handle point sequences without fixed length. In this paper we propose a dimension independent method based on convolutional neural networks to assign parameter values to gridded point clouds of arbitrary size, without the need for additional data processing steps. We train the proposed networks by considering polynomial least squares approximations and demonstrate, both in the univariate and bivariate settings, that the accuracy of the final model properly scales when uniform and adaptive spline refinement is considered. A selection of numerical experiments on point clouds of different sizes highlights the performance of our parameterization scheme. Noisy data sets which simulate measurement errors are also considered.

Keywords: deep learning, convolutional neural networks, parameterization, data fitting, THB-spline, tensor data

1 Introduction

Gridded data consists of (measurement) values obtained at regularly spaced points that form a rectilinear grid. This kind of geometric data with a regular structure arises in different scientific application settings, such as geographic information systems, meteorology, and medical imaging, when, for example, measurements at regular space or time intervals are acquired through different types of technological instruments. The result is a set of ordered points that

** Corresponding author

can represent curvilinear, surface, volumetric, or even higher dimension data. Since the possibility of suitably extending the measurement information to regions where no data is available is often required, efficient parameterization and fitting schemes are usually needed.

A data fitting scheme addresses the construction of curves and surfaces which properly approximate an arbitrary collection of geometric data, such as points or derivative vectors. This type of approximation problem is very common in different applications, where the generated points typically contain measurement errors or computational noise. The re-constructed geometric model should captures the shape of the data, while simultaneously avoiding artifacts and unwanted oscillations, which may easily arise when an (almost) exact match of all the data points is constructed. In the polynomial spline setting, a variety of different approximation schemes can be considered. For example, by focusing on spline approximations of a given structured point cloud, a non-linear optimization problem to minimize the fitting error could be performed with respect to the spline coefficients (e.g., the B-spline control points), the parameters, and the knot values. To avoid non-linear problems, the parameters and the knots are usually computed in advance. These two pre-processing steps are usually indicated as *parameterization* and *knot placement* problems, respectively.

In this paper we employ Convolutional Neural Networks (CNNs) for the *parameterization learning* problem to assign parameter values at an arbitrary number of points on a rectilinear grid for their subsequent use in multivariate polynomial spline approximation schemes. Convolutional neural networks are widely exploited in image processing, see e.g., [10], but they also thrive when any kind of data involving spatial-correlated patterns, as for example audio files [19] and geometric data processing [20], have to be processed. We train our model using synthetic data generated by sampling random tensor-product polynomial curves and surfaces by considering univariate and bivariate polynomial least squares approximations. In our experiments we also add noise to the test data to simulate measurement errors in the input of the proposed networks. Moreover, we use the resulting parameterization to fit point clouds by non-uniform spline constructions and show that the accuracy of the model properly scales under adaptive mesh refinement. The choice of CNNs allows us to obtain a method that is agnostic to the size of the input point cloud, and performs well with an input of a different size from the one considered in the training phase. The experiments highlight that our CNN model can also be effective when multivariate adaptive spline fitting with truncated hierarchical B-splines (THB-splines) is considered, see e.g., [6, 7, 2].

The structure of the paper is as follows. Section 2 presents a brief overview of related works on the parameterization of gridded data. The polynomial spline approximation problem for data belonging to $\mathbb{R}^N$ is briefly summarized in Section 3 together with the parameterization problem in $\mathbb{R}^D$. The proposed convolutional parameterization learning model is presented in Section 4 in its general formulation. Details about the hyperparameters selection and the training procedure are provided in Section 5, with special focus on the curve and surface cases. A

selection of numerical experiments is reported in Section 6. Finally, Section 7 contains few concluding remarks.

2 Related works

Over the last decades several approaches have been developed to handle the parameterization of gridded data. Standard methods rely either on a *uniform* distribution of parameter values over the parametric domain or on scaled configurations of the physical points on the parametric space, as for example the *chord-length* or *centripetal* parameterizations [13]. Alternatives to these standard methods were presented in the literature, with special focus on the univariate setting. In particular, variants of the scaled parameterizations were proposed in [18, 3], whereas an example of an iterative procedure of different kind can be found in [1]. In addition, the so–called universal parameterization method, closely related to the uniform parameterization, was presented in [14] in the context of univariate and bivariate spline interpolation.

The potential of exploiting machine learning techniques for B-spline curve approximation was originally outlined in [12] and [11] by considering support vector machines and multilayer perceptrons, respectively. While the focus of [12] was on identifying a suitable knot placement strategy, a scheme based on two interdependent deep neural networks to compute both the knot and parameter values was proposed in [11]. In particular, the neural network used to predict the parameters consists of a standard multilayer perceptron which takes in input a planar point sequence of fixed length equal to 100 and returns as output the corresponding parametric values. Therefore, if the given number of data points does not match 100, pre- and post-processing steps for super/sub-sampling the input data need to be performed. To overcome the computational limitations of this method, the authors in [17] proposed a method for univariate polynomial curve approximation based on a residual neural network. In this case the network takes in input a planar point sequence of length $2q + 1$, where $q \in \mathbb{N}$ is the polynomial degree. The residual building blocks of the proposed networks are characterized by fully connected layers, which constrain the architecture to deal with an input of fixed size. Consequently, also this approach requires multiple network evaluations and a post-processing step to handle point sequences of arbitrary size. It should also be noted that, to the best of our knowledge, data-driven methods to address the parameterization problem of gridded data sets in a multivariate setting were not previously proposed. To properly process input datasets with varying dimension without requiring additional computations, we here propose a deep convolutional approach.

3 The parameterization problem for data fitting schemes

Let $\boldsymbol{P} = \{\boldsymbol{p}_I \in \mathbb{R}^N : I \in \Gamma\}$ be a *gridded* point cloud, with a rectilinear grid structure along $D \leq N$ directions, defined by the set of multi-indices Γ, so that each item of the point cloud belongs to $\mathbb{R}^N$. Let $d = 1, \ldots, D$ be the index

associated to the parametric direction and let m_d be the point sequence length along each direction d. We define the multi-index set

$$\varGamma = \{I = (i_1, \dots, i_D) : i_d = 1, \dots, m_d,\ d = 1, \dots, D\}, \tag{1}$$

where each multi-index I uniquely identifies a point $\boldsymbol{p}_I$ of the point cloud $\boldsymbol{P}$. Consequently, the point cloud can be rewritten as

$$\boldsymbol{P} = \left\{\boldsymbol{p}_I \in \mathbb{R}^N : I = (i_1, \dots, i_D), i_d = 1, \dots, m_d,\ d = 1, \dots, D\right\}. \tag{2}$$

We denote the cardinality of the point cloud as $m \coloneqq \prod_{d=1}^{D} m_d$. Fig. 1 shows an example of a point cloud $\boldsymbol{P}$, characterized by $D = 2$ parametric directions and elements belonging to $\mathbb{R}^N$, with $N = 3$. The point sequences in the two directions have lengths $m_1 = m_2 = 4$. Each item of the point cloud $\boldsymbol{p}_I$ is uniquely determined by the multi-index $I = (i_1, i_2)$, with i_1 and i_2 varying between 1 and 4.

The least squares spline fitting method addresses the construction of a spline model on a certain parametric space $\varOmega \subseteq \mathbb{R}^D$, which minimizes the residual approximation error. More precisely, by choosing the polynomial multi-degree $\boldsymbol{q} \in \mathbb{N}^D$ and a domain tessellation $\boldsymbol{T}(\varOmega)$, the objective of the method consists in finding a spline $\boldsymbol{s} : \varOmega \to \mathbb{R}^N$,

$$\boldsymbol{s}(\boldsymbol{x}) = \sum_{j=1}^{n} \boldsymbol{c}_j \phi_j(\boldsymbol{x}), \quad \boldsymbol{x} \in \varOmega, \tag{3}$$

where $\phi_j : \varOmega \to \mathbb{R}$ for $i = 1, \dots, n$ are D-variate splines, which minimizes the residual

$$\min_{\boldsymbol{c}_1, \dots, \boldsymbol{c}_n} \sum_{I \in \varGamma} \|\boldsymbol{p}_I - \boldsymbol{s}(\boldsymbol{x}_I)\|_2^2 = \min_{\boldsymbol{c}} \|\boldsymbol{P} - \boldsymbol{B}\boldsymbol{c}\|_2^2 \tag{4}$$

for certain choice of the parameters $\boldsymbol{x}_I \in \varOmega$, for $I \in \varGamma$, where $\boldsymbol{P}$ are the samples of the given point cloud, $\boldsymbol{B} \coloneqq \phi_j(\boldsymbol{x}_I)$, for $j = 1, \dots, n$ and $I \in \varGamma$, is the system collocation matrix, $\boldsymbol{c} \coloneqq \left(\boldsymbol{c}_1, \dots \boldsymbol{c}_n\right)^T$ is the matrix of unknowns containing the coefficients of the reconstructed geometry. Note that in equation (4), $\boldsymbol{p}_I \in \mathbb{R}^N$, for $I \in \varGamma$, are the Cartesian coordinates of the physical points in $\mathbb{R}^N$, whereas $\boldsymbol{x}_I \in \varOmega$ for $I \in \varGamma$ are their associated parametric values.

To construct the gridded data approximation, a parameterization of the point cloud $\boldsymbol{P}$ is needed. This problem requires to identify a set of suitable parametric values

$$\boldsymbol{X} = \{\boldsymbol{x}_I \in \varOmega : I = (i_1, \dots, i_D), i_d = 1, \dots, m_d,\ d = 1, \dots, D\}, \tag{5}$$

so that any point $\boldsymbol{p}_I \in \boldsymbol{P}$ is associated to the parameter $\boldsymbol{x}_I \in \boldsymbol{X}$. Without loss of generality, we set $\varOmega = [0, 1]^D$. In the tensor-product polynomial spline setting, once the univariate parameterization along each parametric direction is computed, the multivariate parameterization can be simply obtained by averaging across all the other parametric directions [16].

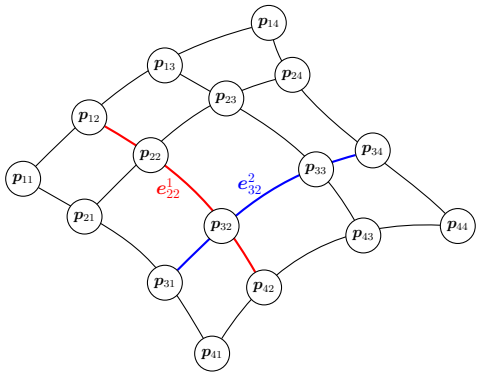


Fig. 1: Example of a gridded point cloud with items in $\mathbb{R}^N$, with $N = 3$, and $D = 2$ parametric directions. The feature extraction step derives the edge vectors $\boldsymbol{e}_{ij}$ of the input grid.

The identification of a suitable parametrization method for the definition of the set $\boldsymbol{X}$ of parameter values is a challenging open problem within the polynomial spline fitting framework. In particular, it can naturally influence the quality of the final approximation leading to sub-optimal approximation error results if not carefully handled, see e.g., [4, 5] which address the case of curve interpolation. Among other optimization approaches, in the next sections, we design a parameterization scheme based on convolutional neural network to address this point.

4 The convolutional parameterization model

The data connectivity information on proximity and distance plays a fundamental role in geometric data processing. Within the data learning framework, modeling the interactions between data relays on their weighted sums, computed via neural networks. Convolutional operators have become a building block for deep learning architectures to process data with a grid-like topology, for example when reconstructing a free-form model starting from a set of gridded observations $\boldsymbol{p}_I \in \mathbb{R}^N$, for $I \in \Gamma$. Architectures which involve convolutional operators within their layers are addressed as *convolutional neural networks* (CNNs). Thanks to the nature of convolutional operators, these networks are characterized by three unique properties: sparse connectivity, parameter sharing, and shift equivariant representations [8]. Sparse connectivity means that the networks have *local* receptive fields, which collect information jointly from spatially neighbouring inputs. Parameter sharing is achieved by requiring the receptive field to assume the same values on different instances of the input, and, consequently, the same parameters are involved to compute each output unit. Finally, shift equivariant representations follow by the definition of the convolutional operations and from

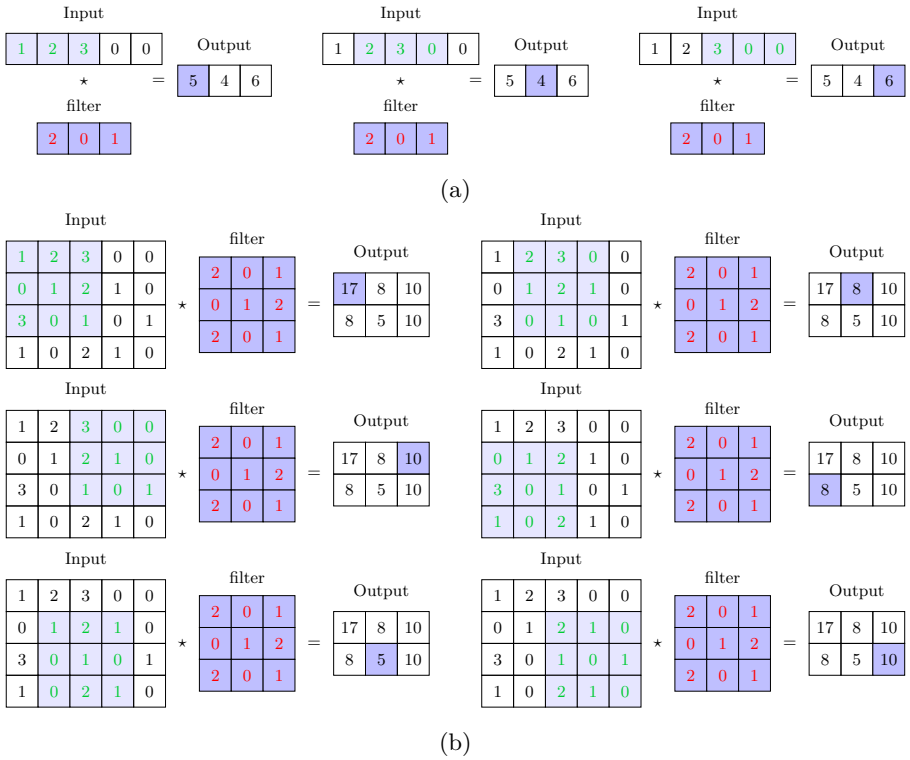


Fig. 2: Examples of convolutional operators for $D = 1$ (a) and $D = 2$ (b).

the previous two properties. Fig. 2 illustrates the locality and parameter sharing properties for the one ($D = 1$) and the two ($D = 2$) dimensional convolutional operator. In both cases, the input is processed by a filter through a matrix convolution operator to generate the output feature map, usually given in input to the successive convolutional operator. In particular, each output item is given by the sum of the element wise product between different *local* values of the input (green numbers in Fig. 2) and the corresponding values of the filter (red numbers in Fig. 2).

The architecture we designed is a *pure* CNN, hence consisting only of convolutional blocks. This choice has been made to take advantage of the locality of convolutional operators in order to be able to support variable input sizes without any additional effort and/or pre- or post-processing of the data. In particular, since the convolutional neural network is characterized by local operators, the particular size of the considered point clouds does not play a fundamental role, whereas the filter dimension does. Consequently, even if a convolutional model is trained on point cloud of fixed size, it can be easily generalized to point cloud of different dimensions. In the following we detail the proposed architecture, while

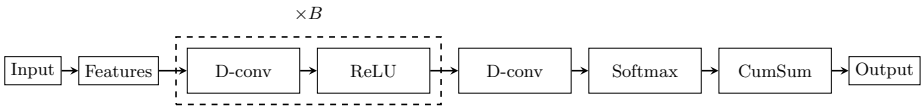


Fig. 3: The CNN architecture.

additional information on the data generation, the hyperparameters selection, and the training process are provided in the next section.

4.1 Learning model architecture

The proposed architecture, presented in Fig. 3, is a sequential CNN which consists of a feature extraction layer, B convolutional building blocks (characterized by convolutional layers coupled with ReLU activation functions), a final block of one convolutional layer, the softmax activation function, and a cumulative sum layer connected to the output. We now give a detailed description of each component of our architecture to illustrate how to obtain the output parameterization result.

Input The learning model takes in input a point cloud $\boldsymbol{P}$ of gridded data, organized in a tensor structure, as in (2).

Features The feature extraction layer produces a relational representation of the point cloud that is translation invariant. It consists in the computation of $m_d - 1$ edges in $\mathbb{R}^N$ between each pair of consecutive points along each direction $d = 1, \dots, D$. Fig. 1 illustrates the feature extraction process for a structured point cloud with $N = 3$ and $D = 2$. In particular, the edge associated to the element $\boldsymbol{p}_{22}$ along the direction $d = 1$ is $\boldsymbol{e}^1_{22} \coloneqq \boldsymbol{p}_{32} - \boldsymbol{p}_{22}$ (swown in red in Fig. 1), whereas the edge associated to the element $\boldsymbol{p}_{32}$ along the direction $d = 2$ is $\boldsymbol{e}^2_{32} \coloneqq \boldsymbol{p}_{33} - \boldsymbol{p}_{32}$ (shown in blue in Fig. 1). More precisely, for any $D \geq 1$, let $I = (i_1, \dots, i_D)$ for $i_d = 1, \dots, m_d - 1$, $d = 1, \dots, D$, and let $I + d$ be the multi-index I augmented by 1 along the d-th direction, for $d = 1, \dots, D$, namely $I + d \coloneqq (i_1, \dots, i_d + 1, \dots, i_D)$. For all $I = (i_1, \dots, i_D)$ with $i_d = 1, \dots, m_d - 1$ and $d = 1, \dots, D$, we define the edges $\boldsymbol{e}^d_I$ as $\boldsymbol{e}^d_I \coloneqq \boldsymbol{p}_{I+d} - \boldsymbol{p}_I$. Note that $\boldsymbol{e}^d_I \in \mathbb{R}^N$. By concatenating the edges $\boldsymbol{e}_I = \left(\boldsymbol{e}^1_I, \dots, \boldsymbol{e}^D_I\right)$, the extracted features are defined as the collection of edges

$$\boldsymbol{E} = \left\{ \boldsymbol{e}_I \in \mathbb{R}^{N \times D} : I = (i_1, \dots, i_D), i_d = 1, \dots, m_d - 1,\ d = 1, \dots, D \right\}. \quad (6)$$

The extracted features are then passed as input to the convolutional building blocks.

D-conv The convolutional layers perform a D-convolution over the input. Each block is characterized by discrete convolutional operators whose hyperparameters along each direction $d = 1, \dots, D$ are the number of input and output channels $c^d_{in}, c^d_{out} \in \mathbb{N} \setminus \{0\}$, the size of the convolving filter $\mathrm{ks}^d \in 2\mathbb{N} + 1$, the amount of padding added to the boundaries of the input $\mathrm{pad}^d \in \mathbb{N}$, the stride of the convolutional operator $\mathrm{s}^d \in \mathbb{N} \setminus \{0\}$, and the spacing between the filter

elements $\mathrm{dil}^d \in \mathbb{N}$. By denoting with $\boldsymbol{u}^d \in \mathbb{R}^{c_{in}^d \times L_{in}^d}$ the input instance of any convolutional layer along the direction $d = 1, \dots, D$ and with $\boldsymbol{y}^d \in \mathbb{R}^{c_{out}^d \times L_{out}^d}$ its output, the relationship between the input (L_{in}^d) and the output (L_{out}^d) sizes is

$$L_{out}^d = \left\lfloor \frac{L_{in}^d + 2 \cdot \mathrm{pad}^d - \mathrm{dil}^d \cdot \left(\mathrm{ks}^d - 1\right)}{\mathrm{s}^d} + 1 \right\rfloor. \tag{7}$$

In order to fully convert the input along any direction $d = 1, \dots, D$ through the filter of dimension ks^d and obtain an output with the same sequence length, namely $L_{in}^d = L_{out}^d$, we specify $\mathrm{s}^d = 1$ and $\mathrm{dil}^d = 0$ and suitably pad the input of each convolutional layer by adding $\left\lfloor \mathrm{ks}^d/2 \right\rfloor$ elements around the boundary of each item along any direction $d = 1, \dots, D$. In particular, we repeat the first and the last input element (reflect padding mode) $\left\lfloor \mathrm{ks}^d/2 \right\rfloor$ times at the beginning and at the end of each input item, along each direction $d = 1, \dots, D$. Moreover, we set the hyperparameter values of the architecture along each direction $d = 1, \dots, D$ as $\mathrm{ks}^d = 2k_d - 1$, where k_d is the spline order along direction d, $c_{in}^d = m$ (amount of point items) in the first convolutional layer and $c_{out}^d = D$ in the last convolutional layer. The input/output channels of the hidden layers are fixed to 100. In particular, their values depends on the user choice, with the only constrain that $c_{out}^d = c_{in}^d$ for two consecutive layers.

ReLU Activation functions are non linear transformations which have been introduced in order to obtain nonlinear learning models [8]. The rectified linear unit (ReLU) is a function which is applied element-wise to the output of all except the last convolutional hidden layer. It is defined as $\rho(t) := \max\{0, t\}$ and it is used for practical applications among most feedforward neural networks due both to its simple piecewise linear structure and its high performance [21].

Softmax As a final activation function we apply the softmax function along each direction $d = 1, \dots, D$ to generate parameter intervals that form a partition of unity along each direction d. In particular, the output of the softmax are the elements $\Delta_I^d \in [0, 1]$ associated to each edge $\boldsymbol{e}_I^d$ computed in the feature extraction layer, for $I = (i_1, \dots, i_D)$ with $i_d = 1, \dots, m_d - 1$, $d = 1, \dots, D$.

CumSum In conclusion, the parametric values are recovered by applying the cumulative sum operation along each direction $d = 1, \dots, D$. More precisely, we define the parametric values as

$$\boldsymbol{x}_{(1,i_2,\dots,i_D)}^d = 0, \quad \boldsymbol{x}_{I+d}^d = \boldsymbol{x}_I^d + \Delta_I^d,$$

where $I + d$ is the multi-index I augmented by 1 along the d-th direction, for $d = 1, \dots, D$. Thanks to the application of the softmax activation function, for each $d = 1, \dots, D$ it is also ensured that the remaining boundary points are mapped to the boundary of the parameter domain, namely $\boldsymbol{x}_{(i_1,i_2,\dots,i_{D-1},m_D)}^d = 1$.

Output The output of the learning model consists of the parametric values $\boldsymbol{X}$ as defined in (5), associated to the input point cloud $\boldsymbol{P}$.

4.2 Loss function

Once the parameterization as in (5) is computed with the convolutional parameterization model, we estimate the control points of the approximating geometry by solving the polynomial least squares minimization problem in Bernstein-Bézier form. The reconstructed model $\boldsymbol{s} : [0,1]^D \to \mathbb{R}^N$ is then used to defined the loss function of our *unsupervised* learning model.

In particular, let $\mathcal{L} : \mathbb{R}^{m\times N} \times [0,1]^{m\times D} \to \mathbb{R}$ be the mean squared error (MSE) between the data $\boldsymbol{P}$ and the corresponding values of the polynomial approximation model $\widehat{\boldsymbol{P}} = \left\{\widehat{\boldsymbol{p}}_I \in \mathbb{R}^N : I \in \mathit{\Gamma}\right\}$, so that $\boldsymbol{s}(\boldsymbol{x}_I) = \widehat{\boldsymbol{p}}_I$ for each multi-index I. Therefore, the loss function is the residual

$$\mathcal{L}\left(\boldsymbol{P}, \boldsymbol{X}\right) \coloneqq \frac{1}{m} \sum_{I\in\mathit{\Gamma}} \|\boldsymbol{p}_I - \widehat{\boldsymbol{p}}_I\|_2^2$$

of the least squared fitting problem corresponding to the parameters $\boldsymbol{X}$ in output from the network. Note that our method falls into the unsupervised learning class, since the real parametric values are not considered to guide the optimization problem and the label role is played by the input points.

5 Curve and surfaces parameterization learning

In this section, we provide the information concerning the training of our models. The method we propose can be implemented for any spatial dimension $N \geq 1$ and parametric dimension $D \leq N$, thanks to the general definition of the convolutional operator. Note that we just set up and train a different model for any distinct pair of values N and D, independently of the considered point cloud. In particular, we focus on two cases: the parameterization of point sequences, when $N = 2$ or $N = 3$ and $D = 1$, and the parameterization of spatial gridded data, when $N = 3$ and $D = 2$, to subsequently compute polynomial and spline approximations in terms of univariate curves and bivariate surfaces, respectively.

5.1 Data generation

Deep learning has its roots in data-driven artificial intelligence and the considered datasets naturally play a fundamental role for the performance of the model. In view of the lack of public datasets for the problem at hand, the points used in the training phase have been randomly generated.

We generated multiple synthetic data by sampling different kinds of parametric objects, either curves ($D = 1$, $N = 2, 3$) or surfaces ($D = 2$, $N = 3$). The proposed algorithm for data generation is the following.

1. Generate a *random* parametric object $\boldsymbol{s} : [0,1]^D \to \mathbb{R}^N$.
2. Sample $\boldsymbol{u}_d = \{u_{i_d}\}_{i_d=1}^{m_d}$ parameters according to the random uniform distribution Uniform$(0,1)$, for $d = 1, \ldots, D$.

3. Normalize the parametric values u_{i_d} in $[0,1]$, so that $u_1 = 0, u_{m_d} = 1$, i.e.

$$u_{i_d} = \frac{u_{i_d} - \min_{j=1,\dots,m_d} u_j}{\max_{j=1,\dots,m_d} u_j - \min_{j=1,\dots,m_d} u_j}.$$

4. Define a tensor-product mesh of samples $\boldsymbol{U} \coloneqq \bigotimes_{d=1}^{D} \boldsymbol{u}_d$, so that

$$\boldsymbol{U} = \left\{ \boldsymbol{u}_I \in [0,1]^D : \boldsymbol{u}_I,\ I = (i_1, \dots, i_D)\,,\ i_d = 1, \dots, m_d,\ d = 1, \dots, D \right\}.$$

5. Evaluate the curve on the parametric values $\boldsymbol{U}$, so that $\boldsymbol{p}_I = \boldsymbol{s}(\boldsymbol{u}_I)$ and collect them in $\boldsymbol{P}$.
6. Normalize $\boldsymbol{P}$ in $[0,1]^N$.

Note that the geometry $\boldsymbol{s} : [0,1]^D \to \mathbb{R}^N$ can belong to any dimension and since the input processed by the convolutional layers are the edges computed in (6), our parameterization method is affine invariant.

The type of the parametric object in step 1 naturally characterizes the generated point cloud. In particular, for $N = 2, 3$ and $D = 1$ we considered random polynomial curves of fixed degree q in Bernstein-Bézier form, where ϕ_j for $j = 1, \dots, q$ in (3) are defined in terms of Bernstein polynomials and $\boldsymbol{c}_j \in \mathbb{R}^N$ for $j = 1, \dots, q$ are the corresponding control points sampled from the standard normal distribution $\mathcal{N}(\boldsymbol{0}, \boldsymbol{1})$. For $N = 3$ and $D = 2$, we generate an initial polynomial tensor-product surface with control points $\boldsymbol{c}_j = [c_j^1, c_j^2, c_j^3]^T$, where $[c_j^1, c_j^2]^T$ are chosen as the tensor product Greville abscissae for degree q and c_j^3 are sampled randomly accordingly to the uniform distribution in $[0,1]$. The resulting surface is then also randomly rotated.

5.2 Hyperparameters selection and training process

Training our neural network consists in using our synthetic data to minimize the loss function, so that the model weights tend to be optimal for both seen and unseen input point clouds. In order to optimize the performance of the network, we set up an extensive grid search on optimizers, learning rates, filter sizes, and number of convolutional building block when applying the learning parameterization model in the context of planar polynomial curves, namely for $N = 2$ and $D = 1$. In particular, we tested three optimizers (Adam, SGD, RMSprop) with learning rates $\gamma = 1\mathrm{e} - i$, $i = 1, \dots, 6$. The filter size is set to be twice the polynomial degree plus one, as in [17], and the number of building blocks is $B = 4$. In addition, we randomly select the point sequence length of each training batch in range $m \in [2d+1, 100]$. We obtained the best results using *RMSprop* as optimizer, learning rate equals to $1\mathrm{e} - 5$, and momentum 0.9. We then used these hyperparameter configurations also to train the model designed for higher dimensions, in particular for $N = 3$ and $D = 2$ to address the surface fitting problem. In this case we increase B to 5 and we fix the point cloud size to $m = 100$. Note that we train a different model for any value of N and D without a fixed number of steps. By following the so-called early stopping criteria, the training phase continues until the validation loss stagnates, a strategy commonly used to avoid overfitting.

6 Numerical results

In this section we present a selection of experiments to analyze the performance of the introduced learning parameterization model and its generalization capabilities with respect to a variety of structured data and different spline approximation schemes. The quality of the parameterization model is defined in terms of the final geometric approximant accuracy, reported in terms of mean squared error (MSE).

We show the effectiveness of our method starting from simple planar polynomial curve approximations (for $N = 2$ and $D = 1$) to arrive at complex adaptive spline surface approximation with THB-splines (for $N = 3$ and $D = 2$). In particular, Section 6.1 presents a comparison with state-of-the-art neural networks for polynomial curve approximation in the case of polynomial data. Trigonometric and noisy data approximation is addressed in Section 6.2, in the context of adaptive spline curve approximation. The performance of the proposed CNN model on benchmark datasets for spline curve approximation is then illustrated in Section 6.3. By moving to the surface case, Section 6.4 shows the results on polynomial data, together with the model robustness to noise. Subsequently, in Section 6.5, we provide numerically evidence of the generalization capabilities of the proposed parameterization model with respect to datasets significantly different (in nature and size) from the synthetic data used during the training phase. Finally, in Section 6.6 and Section 6.7 we investigate the performance of the learning parameterization model for tensor-product B-spline and THB-spline fitting schemes, respectively.

6.1 Polynomial curve approximation

In this experiment we exploit the convolutional parameterization model on univariate polynomial data for different degrees q and variable point sequence lengths m. For each degree $q = 2, \dots, 5$, and length $m = 20, 50, 80$, we generate 100 polynomial point sequences $\boldsymbol{P} = \{\boldsymbol{p}_i \in \mathbb{R}^2 : i = 1, \dots, m\}$ as detailed in Section 5.1. Subsequently, we compute $\boldsymbol{X} = \{x_i \in [0, 1] : i = 1, \dots, m\}$ with the proposed CNN approach and with standard parameterization schemes of the form

$$x_{i+1} = x_i + \frac{\Delta \boldsymbol{p}_i}{\Delta \boldsymbol{p}} \quad \text{for} \quad i = 1, \dots, m-1,$$

with

$$x_1 = 0, \quad \Delta \boldsymbol{p}_i := ||\boldsymbol{p}_{i+1} - \boldsymbol{p}_i||^\alpha, \quad \Delta \boldsymbol{p} = \sum_{j=1}^{m-1} \Delta \boldsymbol{p}_i.$$

In particualr, the common uniform, chord-length, and centripetal parameterizations can be recovered by setting $\alpha = 0, 1, 1/2$ respectively. We then compare the averaged MSE error obtained building the Bézier fitting model at the parametric values given by the best standard technique (STD), the results presented in [16, Table 2] (RES), and the ones obtained with our convolutional model (CNN). The outcome of this analysis is reported in Table 1. The CNN outperforms STD

Table 1: MSE error for Bézier approximation of degree $q = 2, 3, 4, 5$ on polynomial data for different input lengths $m = 20, 50, 80$ in Section 6.1.

	$m = 20$			$m = 50$			$m = 80$		
	STD	RES	CNN	STD	RES	CNN	STD	RES	CNN
$q = 2$	2.97e−3	5.3e−4	**3.21e−5**	1.79e−3	7.3e−4	**2.23e−5**	1.46e−3	7.9e−4	**4.13e−5**
$q = 3$	2.20e−3	6.7e−4	**1.31e−4**	1.43e−3	8.7e−4	**8.80e−5**	1.41e−3	8.5e−4	**1.09e−4**
$q = 4$	1.66e−3	3.3e−4	**7.24e−5**	1.06e−3	4.3e−4	**5.30e−5**	9.48e−4	4.1e−4	**6.13e−5**
$q = 5$	1.35e−3	4.8e−4	**3.71e−5**	7.65e−4	5.8e−4	**2.50e−5**	5.09e−4	5.6e−4	**3.55e−5**

methods and the RES model in terms of mean squared error. In particular, the MSE obtained with CNN is reduced up to one and two orders of magnitude as regards RES and STD, respectively. The advantage of our convolutional parameterization learning model lies not only in the better performance shown in Table 1, but also in the novelty of its self-contained nature. For each input point sequence of any length, our convolutional learning method directly computes the corresponding point parameterization, without any additional computation. The approach proposed in [17] instead requires requires multiple network evaluations and a post processing step to handle arbitrary sequence lengths.

6.2 Generalization to spline curve approximation for datasets of different nature

In this example we investigate the generalization capabilities of our convolutional parameterization model to different datasets and adaptive B-spline curve fitting of various degrees ($q = 2, 3, 4, 5$), as well as its robustness to noise. By considering the algorithms presented in Section 5.1, we generate two distinct datasets of size 100 by sampling trigonometric and polynomial curves. The trigonometric curves are defined as $\boldsymbol{c}(t) = \boldsymbol{a}_0 + \sum_{j=1}^{r} a_j \cos(jt) + i \sum_{j=1}^{r} b_j \sin(jt)$, for a fixed degree r, where $\boldsymbol{a}_0 \in \mathbb{C}$ and $a_j, b_j \in \mathbb{R}$ are values sampled from the standard normal distribution $\mathcal{N}(0, 1)$, for $j = 1, \ldots, r$. In the polynomial case, we added a factor of 1e−2 random Gaussian noise to the data points.

We test the CNN parameterization by considering adaptive B-spline least squares approximation obtained by performing dyadic refinement of the knot intervals which exhibit the highest error values. The resulting MSE is shown in in Table 2 and Table 3 for trigonometric data of degree $r = 5$ and noisy polynomial data, respectively, for different input sequence lengths ($m = 20, 50, 80$) and adaptive B-spline approximations with a final number of 5 interior knots. The CNN parameterization model always obtains better results than the best standard methods (STD). In particular, in the case of trigonometric data (Table 2), it gains up to two order accuracy. For the results on noisy polynomial data (Ta-

Table 2: MSE error for adaptive B-spline approximation with 5 interior knots of degree $q = 2, 3, 4, 5$ on trigonometric data for different input lengths $m = 20, 50, 80$ in Section 6.2.

	$m = 20$		$m = 50$		$m = 80$	
	STD	CNN	STD	CNN	STD	CNN
$q = 2$	3.01e−4	**1.56e−5**	2.17e−4	**6.08e−6**	1.71e−4	**5.82e−6**
$q = 3$	1.35e−4	**1.63e−5**	1.10e−4	**1.05e−5**	3.16e−5	**7.84e−6**
$q = 4$	1.74e−4	**5.53e−6**	2.48e−5	**3.66e−6**	3.14e−5	**4.74e−6**
$q = 5$	3.09e−5	**2.54e−6**	1.17e−5	**1.88e−6**	4.03e−5	**1.92e−6**

Table 3: MSE error for adaptive B-spline approximation with 5 interior knots of degree $q = 2, 3, 4, 5$ on polynomial data with random gaussian noise for different input lengths $m = 20, 50, 80$ in Section 6.2.

	$m = 20$		$m = 50$		$m = 80$	
	STD	CNN	STD	CNN	STD	CNN
$q = 2$	1.27e−4	**7.69e−5**	1.32e−4	**9.46e−5**	1.54e−4	**1.02e−4**
$q = 3$	9.54e−5	**3.36e−5**	1.21e−4	**6.33e−5**	1.12e−4	**7.24e−5**
$q = 4$	7.37e−5	**2.68e−5**	1.66e−4	**8.89e−5**	8.41e−5	**4.04e−5**
$q = 5$	9.02e−5	**2.72e−5**	1.68e−4	**7.64e−5**	1.04e−4	**6.11e−5**

ble 3), the CNN model accuracy is improved up to one order of magnitude when compared to STD.

To better understand the generalization capabilities of the network when an increasing number of knots is considered, we present a final test with fixed input length equals to $m = 20$. We then perform a dyadic adaptive B-spline curve fitting with a varying number of interior knots, starting from the polynomial case of 0 internal knots up to 10 internal knots. The results in terms of MSE for the centripetal (CEN), chordlenght (CHL), uniform (UNI) and CNN parameterizations for $q = 3, 4, 5$ are shown in Fig. 4 (top) for trigonometric data of degree $r = 5$ and in Fig. 4 (bottom) for noisy polynomial data. We may observe that in Fig. 4 (top) for $q = 3, 5$ the MSE error gap between CNN and STD is maintained or even increased with respect to the number of interior knots inserted. In Fig. 4 (top) for $q = 4$ and in Fig. 4 (bottom) for all the degrees, the MSE error gap between CNN and STD tends to reduce with the number of interior knots inserted, but the CNN parameterization model always achieves the best results.

6.3 Spline curve approximation of benchmark datasets

In this experiment, we test our parameterization model on point sequences sampled by the benchmark presented in [15]. By considering B-spline approximation of degree 4 with at most 15 adaptive interior knots, Fig. 5 presents the results

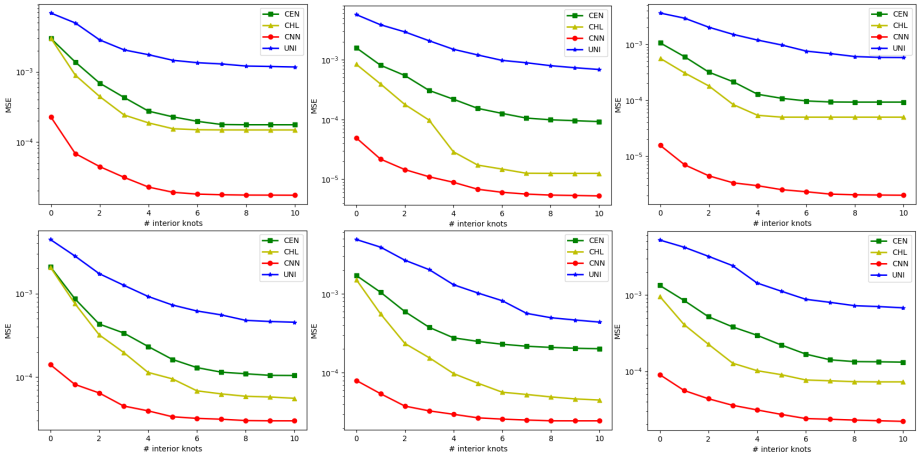


Fig. 4: MSE error for adaptive B-spline approximations with increasing number of knots tested on $m = 20$ trigonometric (top) and noisy (bottom) data for degree $q = 3$ (left), $q = 4$ (center) and $q = 5$ (right) in Section 6.2.

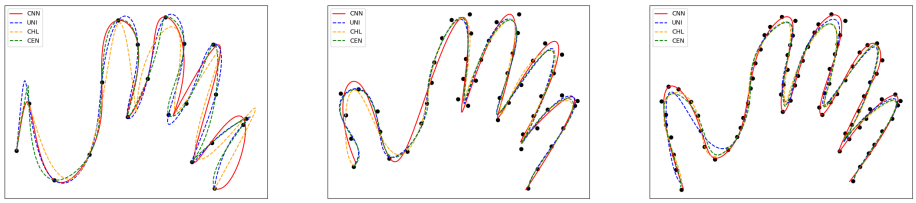


Fig. 5: B-spline approximation of degree $q = 4$ with 15 interior knots for 20 (left), 50 (center), and 80 points (right) in Section 6.3.

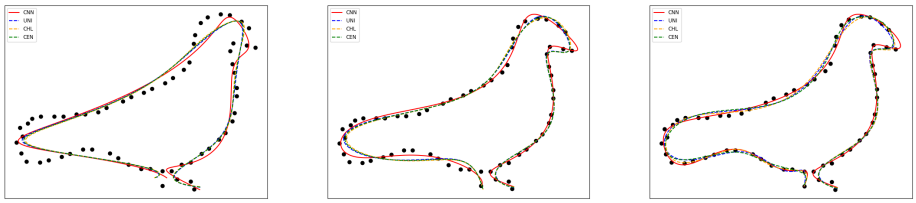


Fig. 6: B-spline approximation of 65 points with $q = 5$ and 4 (left), 7 (center), 10 (right) interior knots in Section 6.3.

obtained employing the standard (UNI, CHL, CEN) and the CNN parameterizations on an increasing number of input points, namely $m = 20, 50, 80$. An analogous comparison on a sequence of $m = 65$ points, approximated with degree $q = 5$ B-spline and adaptive knot vectors with a varying number (4, 7, and 10) of interior knots is shown in Fig. 6.

Table 4: MSE error for polynomial least squares approximation of polynomial (left) and noisy (right) data of bidegree (q, q), for $q = 2, 3, 4, 5$ in Section 6.4.

	exact		noisy	
	STD	CNN	STD	CNN
$q = 2$	1.20e−3	**5.21e−4**	1.26e−3	**5.94e−4**
$q = 3$	6.07e−4	**2.25e−4**	6.45e−4	**2.57e−4**
$q = 4$	3.91e−4	**1.46e−4**	4.62e−4	**1.94e−4**
$q = 5$	2.19e−4	**8.97e−5**	2.75e−4	**1.35e−4**

6.4 Polynomial surface approximation

In this experiment we illustrate the performance of the proposed parameterization model on both exact and noisy data sampled from polynomial patches in the bivariate case. In particular, we collect structured point clouds by sampling Bézier surfaces of bi-degree (q, q). More precisely, for each $q = 2, 3, 4, 5$, we generate 100 point clouds consisting of $m = 144$ gridded points following the algorithm described in Section 5.1. In addition we also create a noisy version of this datasets by perturbing each point cloud with Gaussian noise of factor $\epsilon = 5\text{e}{-3}$. The results of the parametric polynomial fitting of bidegree (q, q) of the aforementioned generated point clouds are reported in Table 4 in terms of MSE for the best standard parameterization technique (STD) and the proposed convolutional model (CNN). The error values give numerically evidence of the generalization capabilities of the trained model with respect to datasets which are somehow similar but at the same time independent from the ones used in the training phase. In particular, the proposed model avoids potential over-fitting problems, it is robust to noise, and it always increases the accuracy of the polynomial approximation more than 50% over STD parameterization techniques in all the considered configurations.

6.5 Generalization to point clouds sampled from geometric models

In this experiment, we analyse the generalization capabilities of the parameterization learning model for data which are different both in dimension and nature from the one used in the training phase. More precisely, we sample point clouds of dimension either $m = 1089$ or $m = 3025$ from 3 different B-spline geometries of bidegree (q, q), representing a car, a ship hull, and a wind turbine. The corresponding point clouds are illustrated in Fig. 7 for the case of 1089 (top) and 3025 (bottom) samples. Moreover, for this experiment we have also stored the true parametric values using during the sampling phase and we will refer to them as the ground truth (GT) in the comparisons. Once the data have been parametrized with the convolutional method (CNN), we performed polynomial approximation of bi-degree (q, q) and compare the MSE and MAX obtained with GT and CNN parameterizations. Note that the GT parameters come from a specific B-spline geometry, therefore if used for different spline models they not necessarily lead to zero error at the evaluation sites.

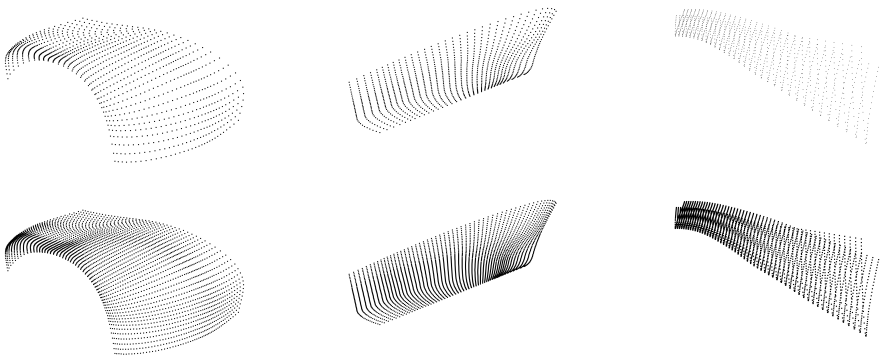


Fig. 7: Point clouds obtained from a car (left), a ship hull (center), and a wind turbine (right) model of 1089 (top) and 3025 (bottom) items.

The results are reported in Table 5. They numerically show the generalization capabilities of the proposed model with respect to the size of the input point clouds and to the nature of the data. For the car geometry, in all the considered configurations, the MSE error of the final reconstructed polynomial geometry is smaller in the case of the CNN parameterization, with a gain in accuracy up to more than 50% of for $q = 3$. When the ship hull geometry is considered, the best performances are achieved if the CNN parameterization is applied in combination with lower polynomial bidegrees, namely $q = 2, 3$, where for $q = 2$ the final reconstructed geometry of the bigger point cloud gains more than 60% of accuracy. As concerns the final wind turbine geometry, the best advantage is registered on the point cloud of size $m = 1089$ among all the different bidegrees, when the MSE is improved up to more than 60% for bidegree $(3, 3)$. The same bidegree $(3, 3)$ leads also to the best error results for the bigger wind turbine point cloud.

6.6 Generalization to tensor product B-splines approximation

In this example we show the generalization capabilities of the convolutional parameterization for tensor product B-spline models. In particular, we consider the point clouds obtained from the car shell geometry introduced in Section 6.5 for which we perform a tensor product B-spline approximation for bidegree $(3, 3)$ and 3 levels of uniform refinement, both considering the given parameterization (GT) and the convolutional parameterization (CNN).

Table 5: MSE for polynomial least squares surface approximation with different bidegrees (q, q) of the car shell, the ship hull, and the wind turbine point clouds of size $m = 1089$ and $m = 3025$ in Section 6.5.

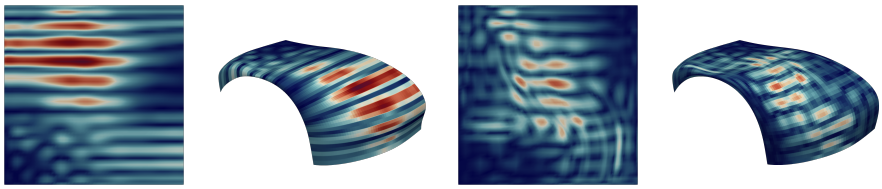
	car		ship hull		wind turbine	
	GT	CNN	GT	CNN	GT	CNN
$m = 1089$						
$q = 2$	1.38e−3	**8.28e−4**	2.53e−4	**1.11e−4**	1.95e−3	**1.30e−3**
$q = 3$	4.58e−4	**1.94e−4**	7.28e−5	**6.01e−5**	1.18e−4	**3.94e−5**
$q = 4$	1.19e−4	**7.63e−5**	2.78e−5	**1.88e−5**	4.21e−5	**3.54e−5**
$q = 5$	7.82e−5	**6.73e−5**	**5.89e−6**	1.19e−5	2.67e−5	**1.38e−5**
$m = 3025$						
$q = 2$	1.32e−3	**8.10e−4**	2.34e−4	**8.44e−5**	1.90e−3	**1.55e−4**
$q = 3$	4.42e−4	**2.03e−4**	6.76e−5	**4.95e−5**	1.14e−4	**7.63e−5**
$q = 4$	1.14e−4	**5.54e−5**	2.62e−5	**2.05e−5**	**4.14e−5**	5.55e−5
$q = 5$	7.47e−5	**5.13e−5**	**5.67e−5**	1.17e−4	2.62e−5	**2.14e−5**

Concerning the point cloud of dimension 1089, the MSE error registered using GT parameterization is 8.18e−6. If the CNN parameterization is considered, the MSE is more than halved and reduces to 3.54e−6. Analogous results are achieved when considering the point clouds of dimension 3025 for which the GT parameterization leads to MSE of 8.21e−6, whereas by using the CNN parameterization the accuracy improves more than 40% obtaining an MSE equals to 4.31e−6.

The scaled error distributions of the final approximation plotted on the parametric and physical domain, for both GT and CNN parameterizations, are illustrated in Fig. 8 for the point cloud of size 3025. More precisely, plots (a) and (b) in Fig. 8 are the error distributions for the GT parameterization, whereas the corresponding results for the CNN parameterization are shown in plots (c) and (d) of the same figure.

6.7 Hierarchical spline approximation of a ship-hull

In this experiment we exploit the performance of the convolutional parameterization model when hierarchical B-spline fitting with THB-splines is considered. In this case the final approximation is obtained with the adaptive THB-spline fitting scheme presented in [9], enriched by a step of parameter correction at each adaptive iteration. In particular, given a point cloud and a suitable parameterization, the considered adaptive fitting scheme designs a hierarchical spline model of the form (3), where the basis function are bivariate truncated hierarchical B-splines (THB-splines). The final THB-spline geometric model is obtained by the iterative alternation of two steps: the computation of the control points and an adaptive refinement of the spline space. At each iteration of the adaptive



(a) (b) (c) (d)

Fig. 8: Point-wise error on the parametric (a-c) and geometric (b-d) domain of the tensor product B-spline approximation of bi-degree $(3, 3)$ for the car shell point cloud of size 3025 using the given GT (a-b) and the CNN (c-d) parameterizations in Section 6.6.

loop, hence for a fixed spline space, the coefficients are computed by solving a penalized least squares problem. As concerns the refinement criteria, we let the refinement be guided by the point-wise approximation error, namely the element domain which registers a point wise error above a certain input tolerance are dyadically refined. The alternation of these two steps, together with the parameter correction, is performed until the point-wise errors are all within the input tolerance or a maximum number of iterations is reached. The hierarchical mesh and the THB-spline approximation obtained for the ship hull point cloud of dimension 3025 are shown on the top line of Fig. 9. In particular, the final geometry has 1349 degrees of freedom and an accuracy characterized by a MSE equals to 2.02e−11. Note that the degrees of freedom are properly distributed along the parametric domain Ω, by following the sharp feature and high curvature regions of the geometry, as shown in Fig. 9.

7 Closure

We presented a general parameterization learning method for polynomial and spline approximation of gridded data point by exploiting CNN architectures. In particular, the learning model takes in input a structured point cloud and gives in output suitable parameters to be used for the geometric modeling of the input data via spline constructions. Our network is characterized by a feature extraction layer, which yields to translation invariance of the proposed method. This layer is then followed by a sequence of CNN layers which encode and propagate proximity information to successive layers. Finally, the parametric values for the input point cloud can be recovered by applying the softmax activation function and a cumulative sum layer. For the numerical experiments, we applied the proposed method by considering its univariate and bivariate configuration. The implemented schemes compares favorably with respect to standard techniques and alternative deep parameterization methods. The results also confirm the flexibility of the convolutional approach, naturally able to approximate gridded point

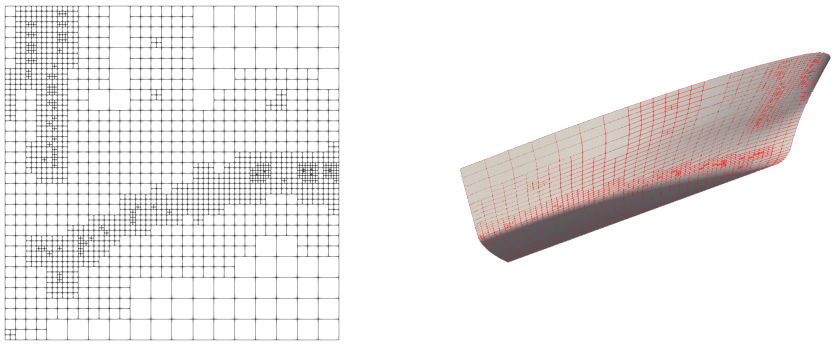


Fig. 9: Hierarchical mesh (left) and geometric spline model (right) of the ship hull point cloud of dimension 3025 via the adaptive scheme with THB-splines in Section 6.7.

clouds of variable size. In particular, we highlighted the generalization capabilities of our model for spline approximation on unseen datasets of different kind, including more general (noisy) data configurations and available datasets. We numerically demonstrate that the accuracy of the spline approximation based on the CNN parameterization properly scales with respect to (non-)uniform spline configurations, while still comparing favorably with respect to standard choices. Finally, we also show that the proposed method can be suitably combined with THB-spline approximation for geometric modeling applications. The potential of exploiting the convolutional spline learning model both for optimal knot placement and mesh refinement, as well as for control points estimation, are interesting directions for future research.

Acknowledgements CG acknowledges the contribution of the National Recovery and Resilience Plan, Mission 4 Component 2 – Investment 1.4 – CN_00000013 "CENTRO NAZIONALE HPC, BIG DATA E QUANTUM COMPUTING", spoke 6. CG and SI are members of the INdAM group GNCS. The INdAM-GNCS support is gratefully acknowledged. CG, SI, AM also acknowledge the PHC GALILEE project 47786N and AM acknowledges H2020 Marie Sklodowska-Curie grant GRAPES No. 860843.

References

1. C. Balta, S. Öztürk, M. Kuncan, and I. Kandilli. Dynamic centripetal parameterization method for B-spline curve interpolation. *IEEE Access*, 8:589–598, 2020.
2. C. Bracco, C. Giannelli, D. Großmann, S. Imperatore, D. Mokriš, A. Sestini. THB-spline approximations for turbine blade design with local B-spline approximations.

In: Barrera, D., Remogna, S., Sbibih, D. (eds) *Mathematical and Computational Methods for Modelling, Approximation and Simulation*, SEMA SIMAI Springer Series, 29:63–82, Springer, Cham. 2022.
3. J. J. Fang and C. L. Hung. An improved parameterization method for B-spline curve and surface interpolation. *Comput. Aided Design*, 45:1005–1028, 2013.
4. M. S. Floater. On the deviation of a parametric cubic spline interpolant from its data polygon. *Comput. Aided Geom. Design*, 25:148–156, 2008.
5. M. S. Floater and T. Surazhsky. Parameterization for curve interpolation, *Studies in Computational Mathematics* in *Topics in Multivariate Approximation and Interpolation*, 12:39–54, 2006.
6. C. Giannelli, B. Jüttler, H. Speleers, THB-splines: The truncated basis for hierarchical splines, *Comput. Aided Geom. Design*, 29:485–498, 2012.
7. C. Giannelli, B. Jüttler, K. Stefan, M. Angelos, S. Bernd, J. Špeh THB-splines: An effective mathematical technology for adaptive refinement in geometric design and isogeometric analysis *Comput. Methods Appl. Mech. Engrg.*, 229:337–365, 2016.
8. I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.
9. G. Kiss, C. Giannelli, U. Zore, B. Jüttler, D. Großmann, and J. Barner. Adaptive CAD model (re-)construction with THB-splines. *Graph. Models*, 76:273–288, 2014.
10. A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. *Commun. ACM*, 60:84–90, 2017.
11. P. Laube, M. O. Franz, and G. Umlauf. Deep learning parametrization for B-spline curve approximation. In *2018 International Conference on 3D Vision (3DV)*, pages 691–699. IEEE Computer Society, 2018.
12. P. Laube, M. O. Franz, and G. Umlauf. Learnt knot placement in B-spline curve approximation using support vector machines. *Comput. Aided Geom. Design*, 62:104–116, 2018.
13. E. Lee. Choosing nodes in parametric curve interpolation. *Comput. Aided Design*, 85, 1989.
14. C. G. Lim. A universal parametrization in B-spline curve and surface interpolation. *Comput. Aided Geom. Design*, 16:407–422, 1999.
15. S. Ohrhallinger, J. Peethambaran, A. D. Parakkat, T. K. Dey, and R. Muthuganapathy. 2D points curve reconstruction survey and benchmarks. *Comput. Graph. Forum*, 40:2, 2021.
16. L. Piegl and W. Tiller. *The NURBS Book*. Springer Science, 1997.
17. F. Scholz and B. Jüttler. Parameterization for polynomial curve approximation via residual deep neural networks. *Comput. Aided Geom. Design*, 85:101977, 2021.
18. S. M. H. Shamsuddin and M. A. Ahmed. A hybrid parameterization method for NURBS. In *International Conference on Computer Graphics, Imaging and Visualization (CGIV 2004)*, pages 15–20. IEEE, 2004.
19. D. Stoller, S. Ewert, and S. Dixon. Wave-u-net: A multi-scale neural network for end-to-end audio source separation. *Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018, Paris, France, September 23-27, 2018*, 334–340, 2018.
20. Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph CNN for learning on point clouds. *ACM Trans. Graphics*, 38(5):146, 2019.
21. D. Yarotsky. Error bounds for approximations with deep ReLU networks. *Neural Networks*, 94:103–114, 2017.