UC Santa Cruz
UC Santa Cruz Previously Published Works

Title

Spatial deep convolutional neural networks

Permalink

Authors

Wang, Qi
Parker, Paul A
Lund, Robert

Publication Date

DOI

Copyright Information

Peer reviewed

Spatial deep convolutional neural networks

Qi Wang [*], Paul A. Parker, Robert Lund

Department of Statistics, University of California, Santa Cruz, United States of America

ARTICLE INFO

ABSTRACT

Spatial prediction problems often use Gaussian process models, which can be computationally burdensome in high dimensions. Specification of an appropriate covariance function for the model can be challenging when complex non-stationarities exist. Recent work has shown that pre-computed spatial basis functions and a feed-forward neural network can capture complex spatial dependence structures while remaining computationally efficient. This paper builds on this literature by tailoring spatial basis functions for use in convolutional neural networks. Through both simulated and real data, we demonstrate that this approach yields more accurate spatial predictions than existing methods. Uncertainty quantification is also considered.

1. Introduction

Gaussian processes are staple models for spatially dependent data. Gaussian processes have been applied to wind speeds (Cellura et al., 2008), groundwater quality (Hooshmand et al., 2011), climatology (Zhang et al., 2023), and air pollution concentrations (Araki et al., 2015). A common assumption with Gaussian processes is stationarity and isotropy: the mean is constant and the covariance between two observations only depends on the distance between the observations. However, stationarity and isotropy are often violated in practice, especially with multi-scale processes (Kirsner and Sansó, 2020). In short, the development of spatial methods that incorporate nonstationarity is needed.

Another issue with Gaussian processes involves the inversion of large-dimensional covariance matrices that arise in forecasting and Gaussian likelihood estimation. Covariance matrix inversion has a complexity of $O(n^3)$ for n observations. Hence, for many large spatial data sets, Gaussian process computations become infeasible. However, recent methods have been developed to reduce computations with minimal loss of precision, including nearest neighbor Gaussian processes (NNGP) methods (Finley et al., 2020), fixed-rank Kriging (Cressie and Johannesson, 2008), and INLA-SPDE methods (Rue et al., 2009; Lindgren et al., 2011).

A variety of models have been proposed to handle spatial nonstationarity. For example, Gramacy and Lee (2008) partition the spatial domain via Bayesian CART methods where stationarity reasonably holds in each subspace. Kim et al. (2005) also employ this tactic, but use Voronoi partitioning on the spatial domain. Another approach accommodates nonstationarity by warping or expanding the spatial domain. For example, Bornn et al. (2012) use dimension expansion, constructing a stationary structure on an expanded space; Schmidt and O'Hagan (2003) and Sampson and Guttorp (1992) warp the spatial domain, assuming stationarity in the warped space. Furthering this line of work, Zammit-Mangion et al. (2022) use deep neural networks to estimate the warping function. Higdon (1998) base their analysis on a discrete convolutional process having a spatially and temporally varying kernel; Lemos and Sansó (2009, 2012) and Kirsner and Sansó (2020) further this line of research. Recently, Kirsner and Sansó (2020) used a multi-scale spatial kernel convolution method that specifies local process properties at different resolutions.

As the popularity of neural networks grows, its ability to model complex nonlinear spatial functions has increased, especially in nonstationarity settings. As a baseline example, Cracknell and Reading (2014) feed spatial coordinates into neural networks to

---

* Corresponding author.
E-mail address: qwang113@ucsc.edu (Q. Wang).

predict nonstationary spatial processes. Building on this, Chen et al. (2024) introduce the so-called DeepKriging approach, which feeds a set of spatial basis functions and coordinates into a neural network. DeepKriging typically outperforms neural networks based on spatial coordinates alone (Chen et al., 2024). Nag et al. (2023b) extend DeepKriging applications to spatial–temporal models by adding temporal basis functions to the spatial basis functions. There, a quantile loss function is used on a long short-term memory (LSTM) neural network to model time dependence. Nag et al. (2023a) develop bivariate DeepKriging for spatial processes with bivariate responses. Zammit-Mangion et al. (2024) feed spatial radial basis functions (RBFs) into a Bayesian neural network (BNN) (Neal, 2012), showing that spatial BNNs can adequately describe common spatial processes. Zhan and Datta (2024) use a generalized least squares loss with weights set via the nearest neighbor Gaussian processes' precision matrix. This is accomplished by adding graph convolution layers after a multi-layer perceptron. Finally, Daw and Wikle (2023) use random Fourier expansions and apply extreme learning machine and ensembling operations to avoid training parameters and conduct uncertainty quantification. See Wikle and Zammit-Mangion (2023) for a recent summary of statistical deep learning approaches for spatial and spatio-temporal data.

Motivated by Chen et al. (2024), this paper shows how to model spatial dependence with convolutional neural networks (CNNs). Similar to Chen et al. (2024), basis functions are employed; however, we evaluate ours on a grid of knot locations. This essentially treats the basis functions as images, allowing us to work with convolutional neural networks, a powerful tool for analyzing image data (Krizhevsky et al., 2012). We use random dropout layers to quantify prediction uncertainty (Gal and Ghahramani, 2016). Our spatial deep convolutional neural network — dubbed a SDCNN — is shown to achieve superior predictive performance in non-stationarity settings. Furthermore, by using neural networks based on gradient approaches (in lieu of inverting a spatial covariance matrix), our approach is extremely computationally efficient, capable of scaling to massive datasets. Our methods also make use of GPUs to further speed computations.

The remainder of this paper proceeds as follows. In Section 2, relevant background material is introduced, including feed-forward and convolutional neural networks and their relation to the methods in Chen et al. (2024). Section 3 presents our proposed SDCNN. Section 5 offers a simulation study,showing the efficacy of the methods on synthetic data. Section 6 demonstrates method performance on temperature and soil carbon data sets. Section 7 provides concluding remarks along with some future research directions.

2. Background

Deep learning applications have exploded in recent years, partially due to their ability to make highly accurate predictions of complex nonlinear processes. In this short section, widely used neural networks such as the feed-forward and convolutional neural networks are sketched. Since the parameters of a neural network are generally estimated by optimizing a loss function, gradient based approaches are considered. A description of the DeepKriging approach of Chen et al. (2024), one of our baseline models, is included.

2.1. Feed-forward neural networks

Feed-forward neural networks (FNNs), which model non-linear structures in independent data, contain an input layer, one or more fully connected layers, and an output layer. Consider a single hidden layer FNN that models the univariate responses $\{Y_i\}_{i=1}^N$. A length-r covariate (input) vector x_i exists to help explain Y_i. The input layer first receives x_i. Then, a fully connected layer computes an affine transformation of the x_i and activates it via the non-linear activation function $f_h(\cdot)$:

$$h_i := f_h(W' x_i + c_h), \quad 1 \le i \le N, \tag{1}$$

where $f_h(\cdot)$ is the activation function and W and c_h are affine transformation parameters. Clarifying dimensions, W is an $r \times n_h$ matrix and c_h is an n_h-dimensional vector. In the deep learning literature, the components of W are called hidden layer weights and c_h is called a bias; n_h denotes the number of hidden layer nodes. The length n_h vector h_i, called a hidden unit, is the output of the hidden layer. While (1) is written in vector form for brevity, $f_h(\cdot)$ is univariate and applied element-wise to the affinely transformed covariates. This layer is fully-connected, and it can be combined with other layers depending on problem goals.

Activation functions are crucial components of neural networks as they inject non-linear features into the modeling procedure. Sharma et al. (2017) summarize frequently used activation functions. Two of the most commonly used are the rectified linear unit (ReLU) function $f(x) = \max(0, x)$ and the Sigmoid function $f(x) = [1 + \exp(-x)]^{-1}$.

The hidden layer in a single hidden-layer FNN is followed by an output layer, which utilizes another activation function $f_o(\cdot)$ to predict Y_i (hats denote predictions):

$$\hat{Y}_i := f_o(\xi' h_i + c_o), \quad 1 \le i \le N.$$

The length n_h vector ξ contains the output layer weights and c_o is a scalar intercept parameter. Here, $f_o(\cdot)$ is a univariate output layer activation function, typically chosen based on the support set of the Y_is. For example, the identity activation function $f_o(x) = x$ is often used with a continuous numeric responses, while the Sigmoid function is often employed for binary Y_i. The quantities W, c_h, ξ, and c_o above are often chosen to optimize some loss function. With continuous $\{Y_i\}$, the quadratic loss $\sum_{i=1}^N (Y_i - \hat{Y}_i)^2$ is typically minimized. As $\hat{Y}_i$ is a non-linear function of x_i, optimal parameter estimates do not usually have a closed form. Nonetheless, optimization algorithms often efficiently numerically minimize the loss via gradient descent methods. Even in complicated neural networks, back-propagation techniques can efficiently calculate the gradients involved in the numerical minimization (Rumelhart et al., 1986).

2.2. Convolutional neural networks

Some geographical or image data have spatial structures that FNNs may not efficiently model. An alternative, a convolutional neural network (CNN), is known to be a superior image analysis tool (Albawi et al., 2017). CNNs stem from LeCun et al. (1995) and more capably explore localized spatial structures. In addition to having convolutional layers, a CNN may also have fully connected and/or pooling layers, depending on needs. Convolutional layers apply a discrete convolution to the input image (regarded as matrix data) with a set of trained filters. The output of each filter is a "convolved image" that contains the Hadamard product of the input image and filter for each compatible submatrix.

Let $\mathcal{K}$ be the image's kernel function and $\mathcal{I}$ the image to be convolved, both bivariate. The discrete convolution operation $*$ obeys

$$\mathcal{K}(x, y) * \mathcal{I}(x, y) = \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} \mathcal{K}(i, j) \mathcal{I}(x - i, y - j).$$

The summation is usually truncated in practice: the indices $x - i$ and $y - j$ should be in the observation ranges.

There are many different convolutions that illuminate certain patterns or features in an image, including edges, corners, or textures (O'Shea and Nash, 2015). In the context of neural networks, these convolutions can be learned or estimated as part of the model fitting procedure, allowing the data to determine the important features for predicting a response. Outputs from convolutional layers can be vectorized and combined with a fully-connected layer to increase model flexibility, often improving standard feed-forward neural network predictions. The superior performance of convolutional neural networks for image data motivates our methods, where we generate basis functions on a regular grid, resembling an image.

2.3. DeepKriging

Spatial processes are often modeled by Gaussian dynamics. Let $\{Z(s)\}$ be a univariate Gaussian spatial process over $s \in \mathcal{R}^d$ and let $Z = \{Z(s_1), \dots, Z(s_N)\}^T$ denote the Gaussian field recorded over the N locations $s_1, \dots, s_N$. The observation $Z(s_i)$ is usually a noisy measurement taken from some latent process $\{Y(s)\}$:

$$Z(s_i) = Y(s_i) + \varepsilon(s_i),$$

where $\{\varepsilon(s)\}$ is a Gaussian error process that is independent of $\{Y(s)\}$ and independent from site to site. Here, $\{Y(s)\}$ is the latent Gaussian process of interest, which is often further decomposed into

$$Y(s_i) = x(s_i)^T \beta + v(s_i),$$

where $x(s_i)$ is a length-p vector of explanatory covariates at location s_i and β is a vector of unknown regression coefficients. Spatial dependence is incorporated by allowing $\{v(s)\}$ to be a zero mean Gaussian process with the unknown spatial covariance $\text{Cov}(v(s_i), v(s_j)) := C(s_i, s_j)$.

Methods exist that model Gaussian processes through basis function expansions. The Karhunen–Loève expansion is a common expansion that represents a random field $\{v(s)\}$ with covariance function $C(\cdot, \cdot)$ as

$$v(s) = \sum_{i=1}^{\infty} \alpha_i \phi_i(s),$$

where $\{\phi_i(\cdot)\}_{i=1}^{\infty}$ are orthogonal functions and $\{\alpha_i\}_{i=1}^{\infty}$ are uncorrelated random variables. From this, a number of basis functions, not necessarily orthogonal, can be defined. For example, one can express $\omega(s) = b(s)'\eta$, where $b(s)$ is a vector of spatial basis functions and η is a multivariate Gaussian variable having a diagonal covariance matrix. An Eigen-decomposition of $b(\cdot)$ effectively rewrites $\omega(s)$ with a Karhunen–Loève expansion.

Two applications of basis function expansions are Gaussian predictive processes (Banerjee et al., 2008) and fixed-rank Kriging (Cressie and Johannesson, 2008). Both have a discrete convolution representation (Higdon, 1998; Lemos and Sansó, 2009, 2012). RBFs are used in Chen et al. (2024) to represent the spatial process, enabling them to make predictions at unobserved locations with FNNs. There, grids of basis functions over several resolutions are generated, governed by three parameters: location, range, and kernel function. The multi-resolution model of Nychka et al. (2015), which explores different spatial structure scales, is used. The basis functions are

$$\phi(r) = \begin{cases} (1 - r)^6 (35r^2 + 18r + 3)/3, & r \in [0, 1] \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

where r is the Euclidean distance between an observation and a basis function knot location, scaled by the range parameter θ. As such, for each observation i, the J basis functions defined by

$$\psi_{i,j} := \phi\left(\frac{\|s_i - c_j\|_2}{\theta}\right), \quad 1 \leq j \leq J,$$

are introduced. Here, the Euclidean distance is used, s_i is the location of ith observation, and c_j is the location extracted from the jth basis function.

Afterward, Chen et al. (2024) input the scaled spatial coordinates and RBFs into a deep neural network. Parameters in the neural networks are estimated by minimizing a quadratic loss, and further predictions are subsequently made. This method has a smaller prediction mean squared error (compared to classical Kriging and FNNs with spatial coordinate input) and avoids covariance matrix inversion.

3. Methodology

This section introduces our spatial deep convolutional neural network (SDCNN). While RBFs are still used, they are employed in non-classical ways. For each resolution, the RBFs are stored in an image format; in this way, each observation has multiple "image covariates" (one image covariate for each resolution). For the RBFs at each resolution, convolutional layers are used to first explore image properties. Then, the image is reshaped into a long vector (flattened) and fed into fully connected layers.

Our SDCNN also employs a random dropout layer before each weight layer, avoiding overfitting and producing prediction uncertainty estimates. Gal and Ghahramani (2016) interpret dropout layers probabilistically, enabling deep neural networks to quantify prediction uncertainty. A fully connected layer is built alongside the other covariates. Finally, we concatenate the outputs from all previous layers and make predictions in the output layer. More details follow.

3.1. Basis function generation

To set up the neural network, spatial RBFs are first generated as input covariates. The R package FRK, created by Zammit-Mangion and Cressie (2023), accomplishes this and is henceforth used. In this package, the function `auto_basis` allows users to select resolution parameters and kernel function type (Gaussian, bisquare, etc.). Elaborating, each basis function has a location and scale parameter that standardizes the distance between each observation and the location of the basis function knot. Also needed is a kernel function, which is chosen as Gaussian here. The basis function evaluated at the ith observation and jth knot, denoted by $\phi(s_i, c_j)$, is

$$\phi(s_i, c_j) = \exp\left(-\frac{\|s_i - c_j\|_2}{2\theta}\right),$$

where s_i denotes the ith observation's location, c_j the jth basis function's knot location, and σ is a scale parameter. In this way, both spatial coordinates and RBFs inject information about the observation at s_i.

Our RBFs are generated under several different resolutions, which enables them to capture spatial effects occurring on different scales (this is a multi-resolution model (Nychka et al., 2015)). For the first resolution, the number of RBFs is relatively small, but the scale parameter is large, serving to explore coarse spatial structure over larger scales. For smaller scale behavior, subsequent RBF resolutions employ an increasing number of RBFs and a decreasing scale parameter, effectively allowing more localized spatial structures to be explored.

An example with three RBF resolutions for United States soil carbon data, which is analyzed later, is depicted in Fig. 1. Here, each RBF resolution has a natural grid structure. For the kth resolution in a two-dimensional spatial domain, $n_k \times m_k$ RBF factors for each observation arise; here, n_k and m_k denote the number of rows and columns, respectively, of the RBF's kth resolution. Given this structure, the RBF values are treated as an image (matrix) to fully take advantage of the power of CNNs.

3.2. Uncertainty quantification

While neural networks are powerful tools capable of accurately predicting complex processes, the gradient-based optimization methods used to fit them only provide point estimates of parameters and their corresponding predictions. To explore estimation variability, we now delve into uncertainty quantification.

Dropout layers, introduced in Srivastava et al. (2014), are layers that act to prevent overfitting while combining many potential models together. The term dropout refers to temporarily dropping a neural network unit and its subsequent influence. Our dropout layers specify a dropout probability p, and every column of the weights W in (1) have probability p of being set to zero, independently across components. This is quantified by setting

$$\tilde{W} := \mathrm{diag}(\psi)W,$$

where $\psi = (\psi_1, \psi_2, \ldots, \psi_{n_h})'$ is a length-n_h vector whose entries can only be zero or one: $\psi_i \overset{iid}{\sim}$ Bernoulli(p) (iid means independent and identically distributed). The dropout probability p is usually chosen via cross-validation. In our work, ψ is resampled at every iteration. Dropouts are typically made during training (ie, gradient descent) but not during prediction. More recently, Gal and Ghahramani (2016) mathematically show that a dropout layer used before each hidden layer probabilistically approximates the scenario with deep Gaussian processes. These authors further demonstrate that incorporating dropout layers during prediction can effectively approximate model uncertainty, while preserving computational efficiency and accuracy. Specifically, prediction uncertainty is introduced by randomly deactivating nodes in the network during the prediction phase. This allows multiple prediction samples to be generated, which can be subsequently analyzed to assess model uncertainty; see Gal and Ghahramani (2016) for additional details.
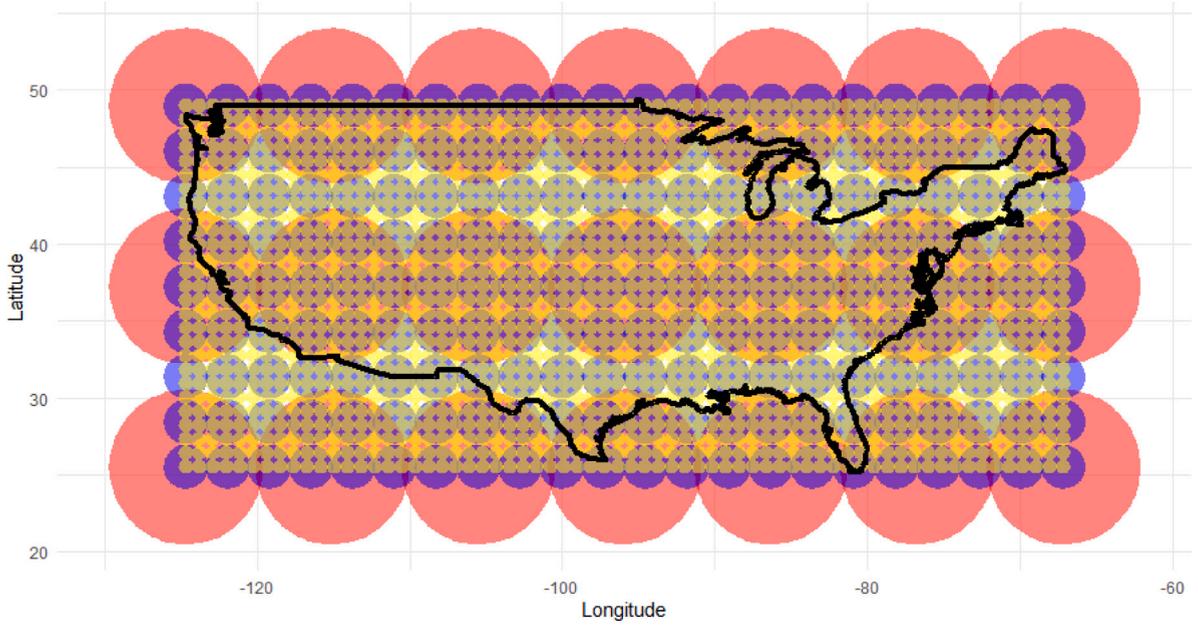
Fig. 1. Illustration of three RBF resolutions. The first resolution (red circles) has the largest scale and explores large-scale spatial effects. The second resolution (blue circles) offers a smaller more localized scale, while the third resolution (yellow circles) explores the finest localized spatial structure.

3.3. SDCNNs

After generating the RBFs, we feed each resolution's image into a convolutional layer. In our work, three different RBF resolutions are fed into three separate convolutional layers. The output from these convolutional layers are subsequently vectorized and concatenated with the output from the convolutional layer. These are then treated as input to a feed-forward layer. It is also possible to include additional input covariates within the feed-forward layer for spatial coordinates.

Taking the first resolution as an example, suppose that for each s_i, the $n_1 \times m_1$ RBF matrix $\boldsymbol{B}_{i,1}$ is calculated, where $(\boldsymbol{B}_{i,1})_{n,m}$ is the RBF value associated with location s_i for the first resolution in the nth row and mth column. Each element in $\boldsymbol{B}_{i,1}$ represents a knot, implying that $J_1 = m_1 n_1$, where J_1 denotes the total knots (basis functions) for the first resolution. We then feed $\boldsymbol{B}_{i,1}$ into a convolutional layer containing n_F filters, denoted by $\{\boldsymbol{\mathcal{K}}_\ell\}_{\ell=1}^{n_F}$, where $\{\boldsymbol{\mathcal{K}}_\ell\}_{\ell=1}^{n_F}$ is a 2×2 matrix for all ℓ. Our work takes $n_F = 128$ as is classic in the literature, but other settings can be tried. Since the terms "filter" and "kernel" describe the same things in neural networks, we use the term filter henceforth to avoid confusion with the kernel terminology common in spatial statistics. The output from the convolutional layer for the ℓth filter, denoted by $\boldsymbol{C}_{i,\ell}$, is an $(n_1 - 1) \times (m_1 - 1)$ matrix with entries

$$(\boldsymbol{C}_{i,\ell})_{p,q} = \sum_{m=1}^{2} \sum_{n=1}^{2} (\boldsymbol{B}_{i,1})_{p+m-1,q+n-1}(\boldsymbol{\mathcal{K}}_\ell)_{m,n}.$$

After computing the output from the convolutional layer, $\boldsymbol{C}_{i,\ell}$ is vectorized into a length-$(n_1 - 1)(m_1 - 1)$ vector. As there are n_F similar filters, n_F vectors with the same length are obtained after convolution and vectorization. These vectors are next stacked into one long vector, denoted by $\boldsymbol{C}_{i,1}$, of length-$n_F(n_1 - 1)(m_1 - 1)$.

Our next step feeds $\boldsymbol{C}_{i,1}$ into three feed-forward layers, with each layer having $N_h = 100$ hidden nodes (this setting can also be changed) and using the same ReLU activation function f:

$$(\boldsymbol{H}_i)_{1,1} = f(\boldsymbol{W}'_{1,1}\boldsymbol{C}_{i,1} + \boldsymbol{c}_{1,1}),$$
$$(\boldsymbol{H}_i)_{1,2} = f(\boldsymbol{W}'_{1,2}(\boldsymbol{H}_i)_{1,1} + \boldsymbol{c}_{1,2}), \tag{3}$$
$$(\boldsymbol{H}_i)_{1,3} = f(\boldsymbol{W}'_{1,3}(\boldsymbol{H}_i)_{1,2} + \boldsymbol{c}_{1,3}),$$

where $\boldsymbol{W}_{1,1}$, $\boldsymbol{W}_{1,2}$, and $\boldsymbol{W}_{1,3}$ are weight matrices of dimension $n_F(n_1 - 1)(m_1 - 1) \times N_h$ for $\boldsymbol{W}_{1,1}$ and $N_h \times N_h$ for the other two $\boldsymbol{W}$s. Here, $\boldsymbol{c}_{1,1}$, $\boldsymbol{c}_{1,2}$, and $\boldsymbol{c}_{1,3}$ are length N_h bias/intercept parameters. The vectors $(\boldsymbol{H}_i)_{1,1}$, $(\boldsymbol{H}_i)_{1,2}$, and $(\boldsymbol{H}_i)_{1,3}$ each have length N_h.

Following similar steps, transformed RBFs for the other two resolutions, denoted by $(\boldsymbol{H}_i)_{2,3}$ and $(\boldsymbol{H}_i)_{3,3}$, are computed. The effects of any covariates $\boldsymbol{x}_i$ at location s_i can be quantified by feeding these into another feed-forward layer, again with N_h nodes; this results in $(\boldsymbol{H}_i)_{\boldsymbol{x}}$. At this point, the quantities $(\boldsymbol{H}_i)_{1,3}$, $(\boldsymbol{H}_i)_{2,3}$, $(\boldsymbol{H}_i)_{3,3}$, and $(\boldsymbol{H}_i)_{\boldsymbol{x}}$ are stacked into a length $4N_h$ vector denoted by $\boldsymbol{H}_i$.

In our final step, $\boldsymbol{H}_i$ is fed into an output layer:

$$O_i = \boldsymbol{W}'_{\text{out}} \boldsymbol{H}_i + c_{\text{out}},$$

where $\boldsymbol{W}_{\text{out}}$ is a $4N_h \times 1$ weight matrix and c_{out} is a constant (the identity activation function is used here). Here, O_i serves as a prediction of Y_i given a set of neural network parameters. With gradient-based optimization approaches, one can numerically estimate all weight matrices and bias parameters by optimizing an appropriate loss function. In our case, a quadratic sum of squares loss is used with continuous responses. Random dropout layers are added before each hidden layer to quantify prediction uncertainty and avoid overfitting.

4. Model comparisons and performance evaluation methods

This section discusses how to compare our SDCNN to three other techniques. Our first comparative technique uses a Gaussian process fitted by integrated nested Laplace approximations (INLA). The second and third comparative techniques are neural networks with various inputs. Several scoring rules and performance metrics are introduced below to judge the model fits.

4.1. Competing models

We compare our SDCNN with the DeepKriging approach of Chen et al. (2024), a baseline DNN with only spatial coordinates inputted, and a Gaussian process fitted via INLA. Comparisons are done via five-folded cross-validation (out of sample prediction) for one simulated and two real data sets. Specifically, for each case, the full dataset is split first into training and test portions for cross validation. The model is only trained on the training data. For each neural network based approach, early stopping is applied: a subset of training data, called a validation set, is further split to determine when to stop model training to avoid overfitting. The model is called trained for one epoch when all training data has been used in the optimizer once. At the end of each epoch, if the model fit becomes better (a smaller loss on the validation data), the model is saved and replaces the previous optimal model. In short, only the parameters that lead to the smallest mean squared error on validation sets is saved, and further used to make out-of-sample prediction on the test portion. The optimizer Adam (Kingma and Ba, 2014) is used with a squared error loss and learning rate of 0.001 in all neural network models. Our SDCNN is trained with 1000 epochs, batch size 1000, and drop rate of 0.1. A more detailed structure of each neural network is shown in Appendix.

All code used in this paper has been uploaded to https://github.com/qwang-77/Convolutional-Kriging and the models are fitted in R. The R-INLA package in Martino and Rue (2009) is used along with the neural network package Keras (Arnold, 2017). Choices of hyperparameters, the number of hidden nodes, etc., are now discussed.

- **R-INLA** :
  The R-INLA package of Martino and Rue (2009) implements a two-dimensional stochastic partial differential equation (SPDE) approach to model spatial processes. Righetto et al. (2020) discuss hyperparameter choices in this method, such as mesh-building parameters. Two crucial parameters, which quantify the triangular densities of the mesh and are generated by `inla.mesh.2d`, are `max.edge`, which controls the largest triangle edge length, and `cutoff`, which controls the minimum distance between two points. Care is needed to define a proper value of `max.edge` as this parameter depends on `cutoff`; other parameters are set to default package values. The `max.edge` parameters are set by first choosing the smaller of the range of the longitudes and latitudes. This is then divided by the square root of the total number of observations, then multiplied by an adapting parameter $\theta > 0$, chosen by cross validation, for each data set. This procedure is quantified by a parameter ϕ^* obeying

$$\phi^* = \frac{\min\left(r(\text{Longitude}), r(\text{Latitude})\right)}{\sqrt{N}} \times \theta,$$

  where $r(\cdot)$ calculates the range of a vector of observations. Smaller θ induce denser triangles in the mesh; however, computations increase rapidly with smaller θ due to the increase in the number of knots.

- **Baseline Deep Neural Network** :
  This deep neural network trains the model using only longitude and latitude. The longitude and latitude are first scaled with min–max methods: $f_{min--max}(z) = z/(\max \boldsymbol{x} - \min \boldsymbol{x})$ before being fed into the neural network. There are three hidden layers, all using the ReLU activation function, 100 hidden units in each layer, and one output layer with a linear activation function. A dropout layer with dropout rate $p = 0.1$ is applied before each weight layer. Finally, a batch normalization layer is used after each hidden layer. In this manner, the best model among all epochs is retained. The batch size and maximum number of epochs are flexibly set for each data set after considering computational issues.

- **DeepKriging with RBFs** :
  In addition to longitude and latitude, the basis functions calculated in the FRK package (Zammit-Mangion and Cressie, 2023) are included. There may be multiple RBF resolutions; these basis functions are not scaled. Three basis function resolutions are used in our work, corresponding to the first three resolutions in the FRK package. Multiple hidden layers are used, each having 100 hidden nodes, followed by one dropout layer with a dropout rate of 0.1 and a batch normalization layer. The ReLU activation function is used. The other settings are identical to the Baseline DNN settings, but the epoch count is set to 500. Early stopping is used again.

4.2. Model performance scoring rules

Following Gal and Ghahramani (2016), after training the model in each cross-validation fold, predictive samples can be obtained at each observed location. Based on these samples, the four models are judged/scored on the following criteria:

- **Prediction MSEs** :
  For each fold, the prediction mean squared error (MSE) for the testing sample is calculated based on an out-of-sample prediction computed from 80% of the data. Thereafter, the MSE is calculated for these predictions via

  $$\text{MSE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\sum_{i=1}^{N}(\mathbf{y} - \hat{\mathbf{y}})^2}{N}.$$

  MSE measures balance bias and variance in assessing point prediction accuracy.

- **CRPS** :
  The continuous ranked probability score (CRPS) is defined as

  $$\text{CRPS}(F, x) = -\int_{-\infty}^{\infty}(F(y) - I\{y \geq x\})^2 dy.$$

  Here, F is the cumulative distribution function (CDF) of the probabilistic forecast. By Lemma 2.2 of Baringhaus and Franz (2004) (or identity (17) of Székely and Rizzo (2005)), the CRPS score has the closed form

  $$\text{CRPS}(F, x) = \frac{1}{2}E_F[|X - X'|] - E_F[|X - x|],$$

  where X and X' are independent copies of a random variable with cumulative distribution function F and finite first moment (Gneiting and Raftery, 2007). The CRPS evaluates probabilistic forecasts by assessing its entire predictive distribution; it is particularly useful for comparing models in contexts where uncertainty quantification is important. The model with the bigger CRPS is preferred. CRPS scores can be negative.

- **ICRs** :
  Level α credible intervals of the posterior predictive distribution at each location can be obtained via Monte Carlo methods. The interval coverage rate (ICR) measures the percent of observations that fall within the credible interval:

  $$\text{ICR} = \frac{\sum_{i=1}^{N} I(x_i \in (L_i(\alpha), U_i(\alpha)))}{N}.$$

  Here, x_i denotes the ith observation, α the credible interval level, $L_i(\alpha)$ is the lower $\frac{\alpha}{2}$ quantile of the prediction samples, and $U_i(\alpha)$ is the upper $1 - \frac{\alpha}{2}$ quantile of the prediction samples. The model with the closest ICR to $1 - \alpha$ is preferred. Although the coverage rate only compares the accuracy of an interval estimate and not its precision, it is a natural and common approach to evaluating uncertainty quantification.

- **ISs** :
  The interval score (IS) is another prediction interval measure, with smaller ISs being preferred. Two quantiles that control the upper and lower tails govern the width of the interval:

  $$U = Q\left(x, 1 - \frac{\alpha}{2}\right), \quad L = Q\left(x, \frac{\alpha}{2}\right),$$

  where Q is the quantile function. The IS is

  $$S_\alpha^{\text{int}}(L, U; x) = (U - L) + \frac{\alpha}{2}(L - x)I\{x < L\} + \frac{\alpha}{2}(x - U)I\{x > U\}$$

  This scoring rule, evolving from Dunsmore (1968) and Murphy and Winkler (1979), has intuitive appeal. The predictor is rewarded for an accurate and short interval, but is penalized if the observation misses the posterior interval. IS balances accuracy and precision for interval estimates (similar to the bias and variance balance of the MSE for point estimates).

5. Simulation study

This section studies the ability of our SDCNNs to predict a complex target function — the two dimensional Eggholder function

$$f(x_1, x_2) = -(x_2 + 47)\sin\left(\sqrt{\left|x_2 + \frac{x_1}{2} + 47\right|}\right) - x_1 \sin\left(\sqrt{\left|x_1 - (x_2 + 47)\right|}\right),$$

where x_1 and x_2 represent longitude and latitude. This function has multiple local maximums and minimums, as depicted in Fig. 2, making prediction difficult.

Two configurations of training data are considered. The first uses all points in a square within a five-folded cross-validation; the second eliminates a rectangular subregion of these locations. This eliminated region is graphically depicted in Figs. 5 and 6. In the latter case, the observations within the rectangular region are predicted to evaluate procedure stability and robustness. Our (x_1, x_2) locations are generated by a 300×300 grid, uniformly spaced over $(x_1, x_2) \in [-500, 500] \times [-500, 500]$. We linearly scale x_1 and x_2 so that they all lie in $[0, 1]^2$. Response variables are the $f(x_1, x_2)$ values (without noise).
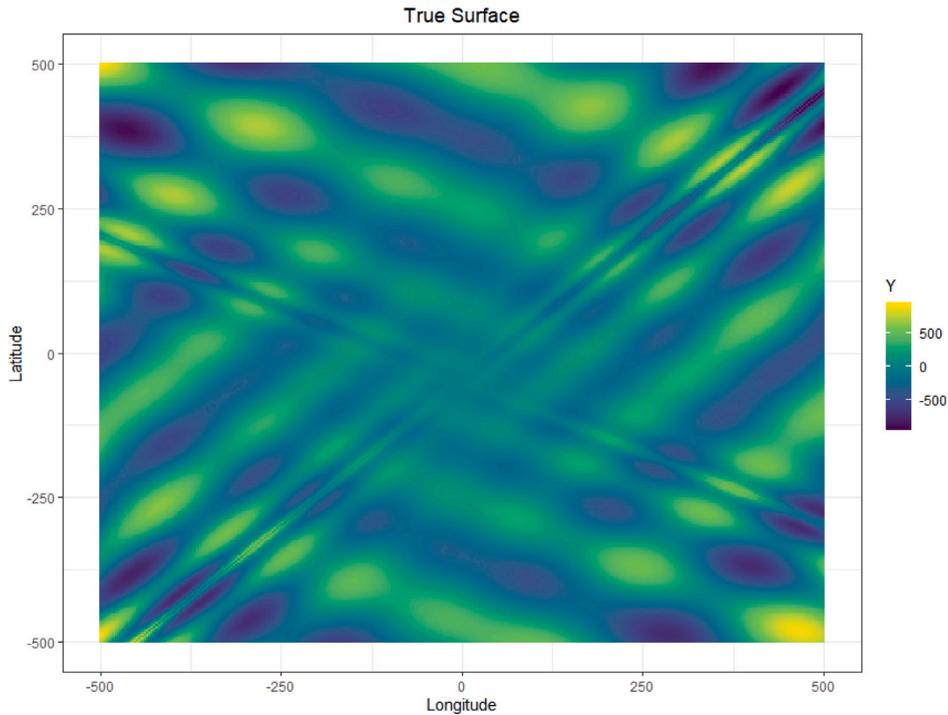
Fig. 2. True Eggholder function values.

Table 1
Scores for Full Eggholder simulation dataset.

	MSE	CRPS	ICR(0.95)	IS(0.05)
INLA-SPDE	297.57	−5.27	0.91	125.77
Baseline FNN	5584.53	−38.23	0.83	568.99
DeepKriging	171.64	−5.55	**0.96**	61.51
SDCNN	**52.68**	**−3.24**	0.99	**41.63**

5.1. Predictive eggholder function performance

5.1.1. The full data set

First, we train a model using all observations. The predicted surface for this data is shown in Fig. 3. The Baseline DNN model makes the worst predictions, ambiguously describing local Eggholder properties. Visual differences between the INLA, DeepKriging, and SDCNN methods from the plot are minor; performance scores will reveal more structure later. The uncertainty surface in Fig. 4 shows that the INLA method has the smallest prediction standard deviation of all models. The Baseline DNN has the largest standard deviation (with obvious biases), and the SDCNN model has a smaller standard deviation than the DeepKriging model.

Using dropout layers to assess prediction uncertainty, 100 prediction samples for the Baseline DNN, DeepKring, and SDCNN models were made for each point. For INLA, posterior predictive samples are accessibly obtained through the R INLA package, allowing us to calculate mean squared prediction errors, 0.95 ICRs, CRPSs, and ISs for each spatial location. The scores for these four models are shown in Table 1 and reveal that the SDCNN method has the smallest MSE, CRPSs, and ISs. This said, the interval coverage rate of 0.99 for the SDCNN seems too high (not as close to 0.95 as DeepKriging).

5.1.2. The reduced data set

This case repeats the last subsection's exercise with the reduced data set. The predictive mean surface and standard deviation plots are shown in Figs. 5 and 6. For the predictive mean surface, INLA did not reproduce the "diagonal lines" structure in the Eggholder function values. The best predictive model is again the SDCNN, whose predictions did not overly degrade with the omitted rectangle. In contrast, the standard deviation of INLA predictions increased greatly when the rectangle was omitted. The DeepKriging method also did not significantly degrade with the omitted rectangle. Here, SDCNNs are more stable than INLA techniques when local observations are rare; moreover, SDCNNs have the best performance scores when the rectangle is not omitted.
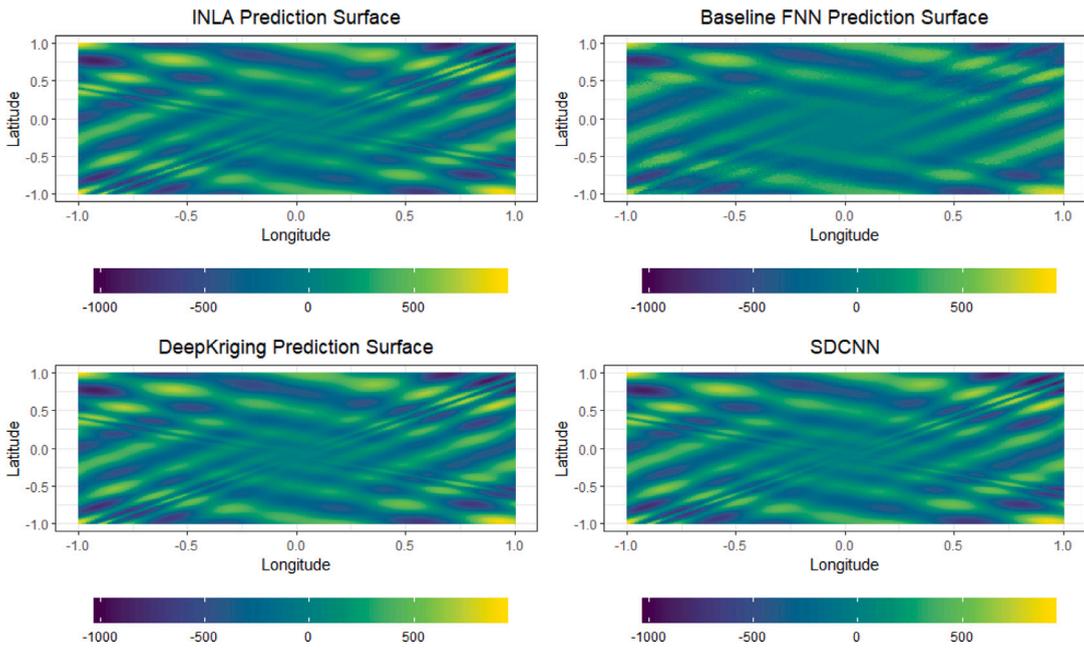
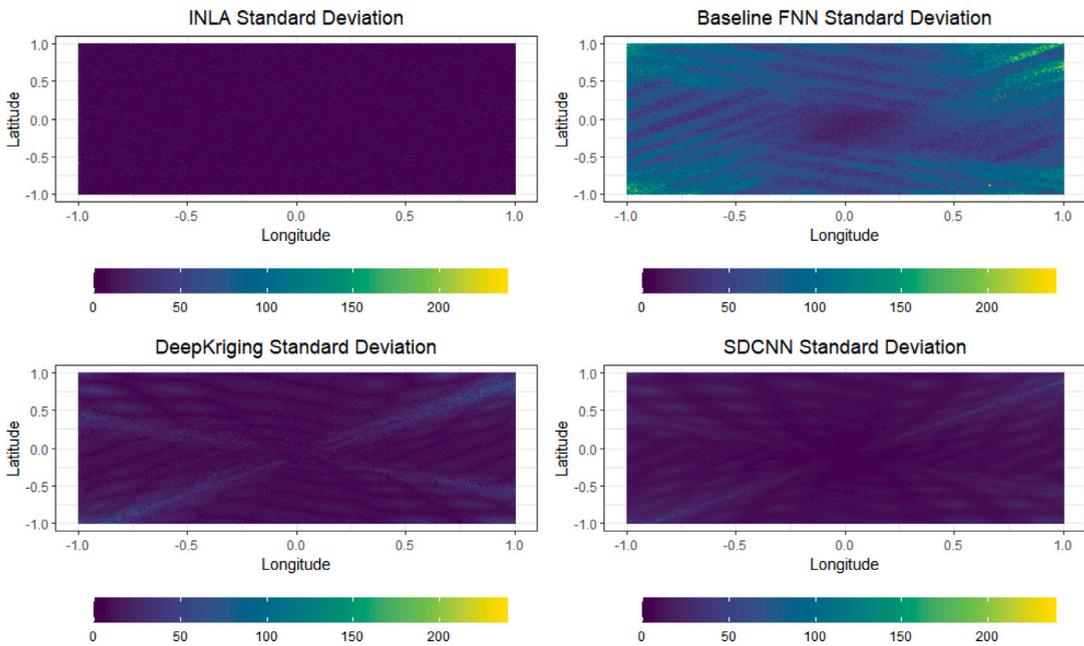**Fig. 3.** Predicted mean surface of the Eggholder function from all data.

Fig. 4. Prediction standard deviation of the Eggholder function from all data.

6. Applications

6.1. Satellite temperatures

Now we apply our SDCNN method to a data set used in Heaton et al. (2019), which contains observations on the 500×300 equally spaced grid spanning longitudes from −95.91153 degrees W to −91.28381 degrees W and latitudes from 34.29519 degrees N to 37.06811 degrees N. About 1.1% of the data are missing because of clouds. The Y variable is the daytime land surface temperature
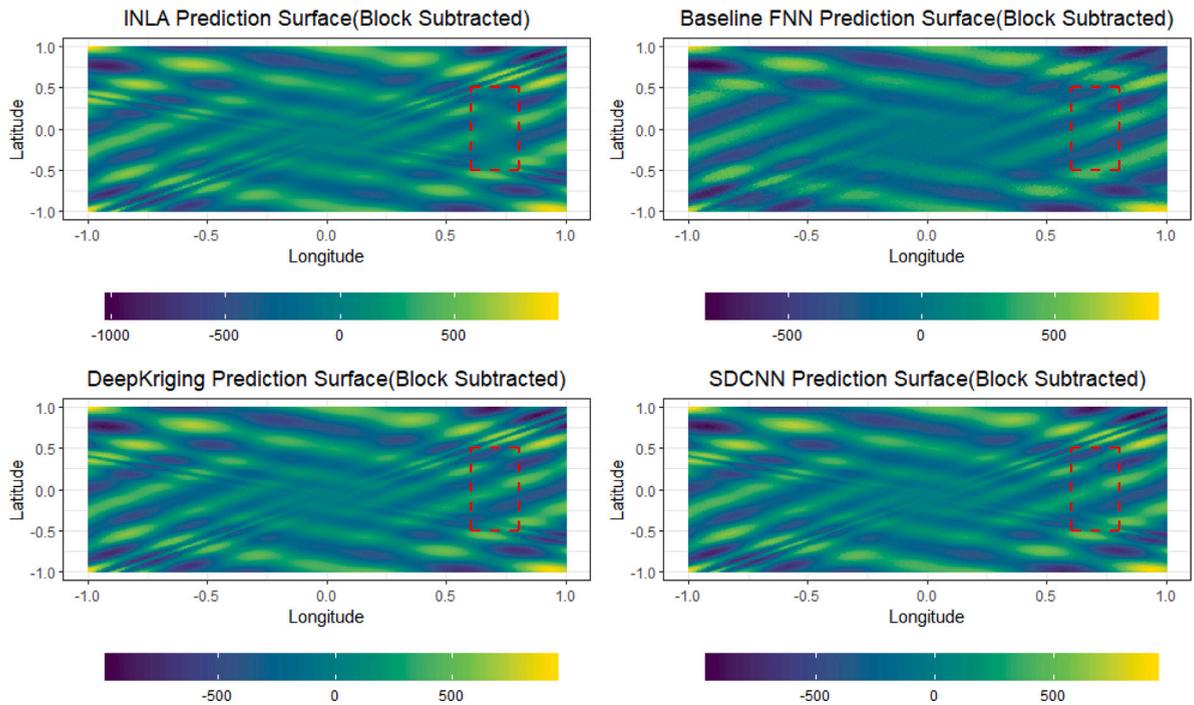
Fig. 5. Predicted Eggholder surface with a missing rectangle.
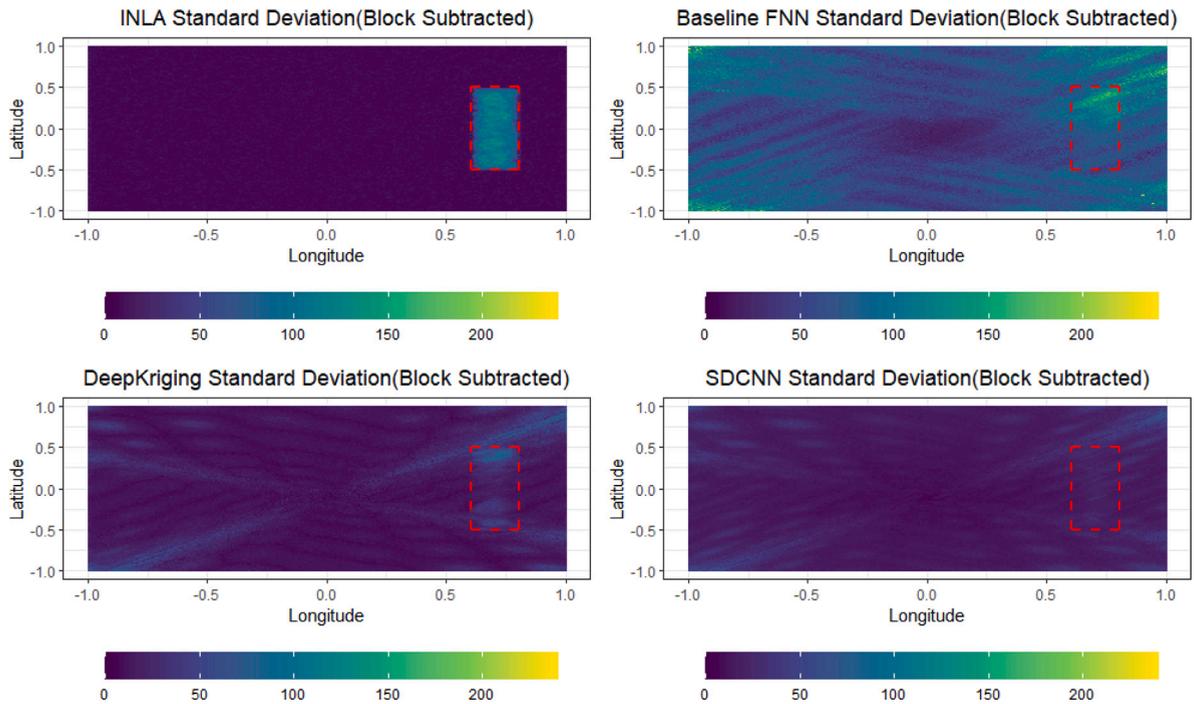
Fig. 6. Standard deviation of Eggholder surface with a missing rectangle.

in degrees Celsius recorded by the Terra instrument onboard the MODIS satellite on August 4, 2016 (Level-3 data). The data are available at https://github.com/finnlindgren/heatoncomparison, contain 148,309 observations, and are plotted in Fig. 7.

We generated a 200 × 200 grid of longitudes and latitudes over the study region and predicted the surface temperatures at all locations on the 200 × 200 grid. The predicted surface of the four models above is shown in Fig. 8 over the grid. The SDCNN

Table 2

Scores for the satellite temperatures.

	MSE	CRPS	ICR(0.95)	IS(0.05)
INLA-SPDE	1.04	−0.65	0.37	17.64
Baseline FNN	2.02	−0.89	0.44	21.48
DeepKriging	0.78	−0.54	0.59	9.99
SDCNN	**0.31**	**−0.30**	**0.70**	**4.77**
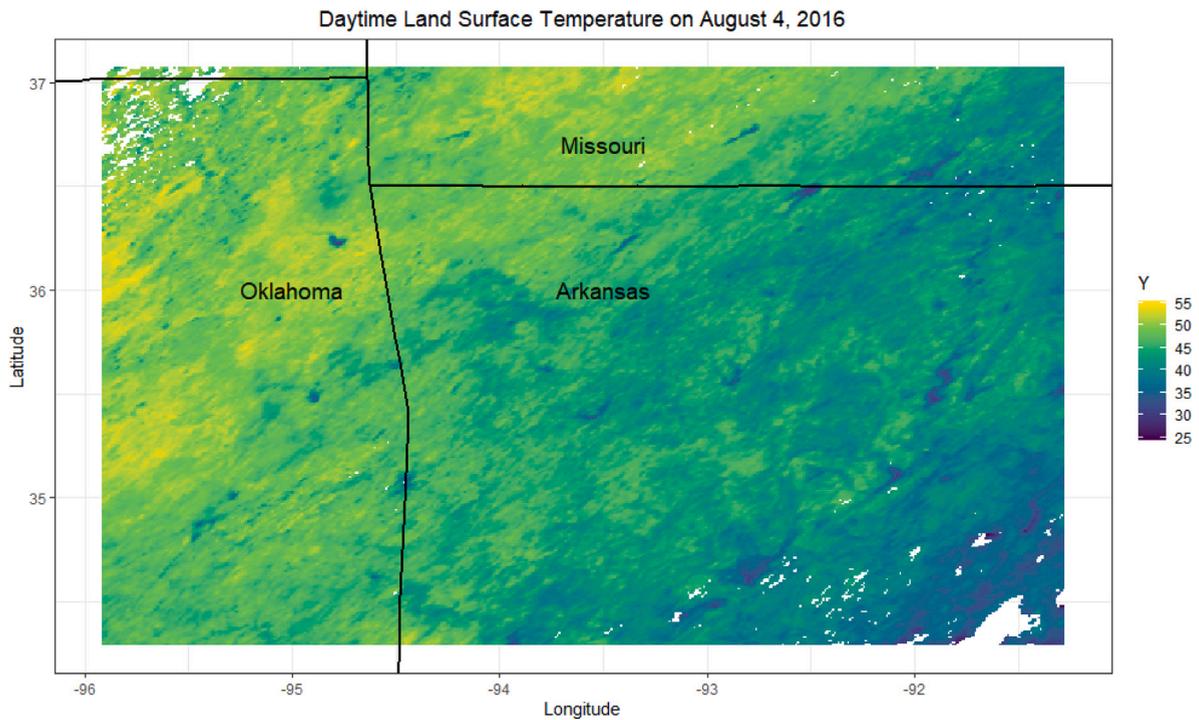
Fig. 7. Observation surface of the satellite data.

method best matches the true data, efficiently capturing local variations in the surface. INLA methods also capture some of the local structure, but the image is blurrier than the SDCNN predicted surface. The DeepKriging and Baseline DNN methods did not work well at all.

The INLA predictions in Fig. 9 have a large standard deviation in the bottom right corner locations where data are missing. This said, the INLA predictions have the smallest standard deviation when observations exist nearby.

The prediction MSEs, CRPSs, ICRs, and ISs for each fold are shown in Table 2. The SDCNN method performs best.

In the above table, the SDCNN method performs better than the other three methods, uniformly over the considered scoring metrics.

A boxplot of the CRPSs of these four models is shown in Fig. 10 and reveals further structure. The SDCNN method has the smallest quartile of all four models, indicating its superiority. Perhaps more importantly, the larger outliers of the other methods are avoided with the SDCNN approach.

Figs. 11 show boxplots of ISs. The SDCNN model again performs best.

6.2. United States soil carbon

In our previous data sets, all observations roughly lied on a lattice. Our next example involves soil carbon measurements from the conterminous United States and is more "non-lattice". The data were extracted from the in-built R package soilDB created by Beaudette et al. (2023). This package extracts soil properties from the databases of the US Department of Agriculture's Natural Resources Conservation Service (USDA-NRCS) and the National Cooperative Soil Survey (USDA-NCSS). Longitude and latitude are used as covariates; soil organic carbon (SOC) content is our response variable as measured in Mg/ha (megagrams per hectare). The code to extract the data is on our GitHub page for this paper. The dataset contains 31,491 irregularly spaced observations as shown in Fig. 12. For our prediction and uncertainty surfaces, we again generate a 200×200 grid encompassing the spatial domain.
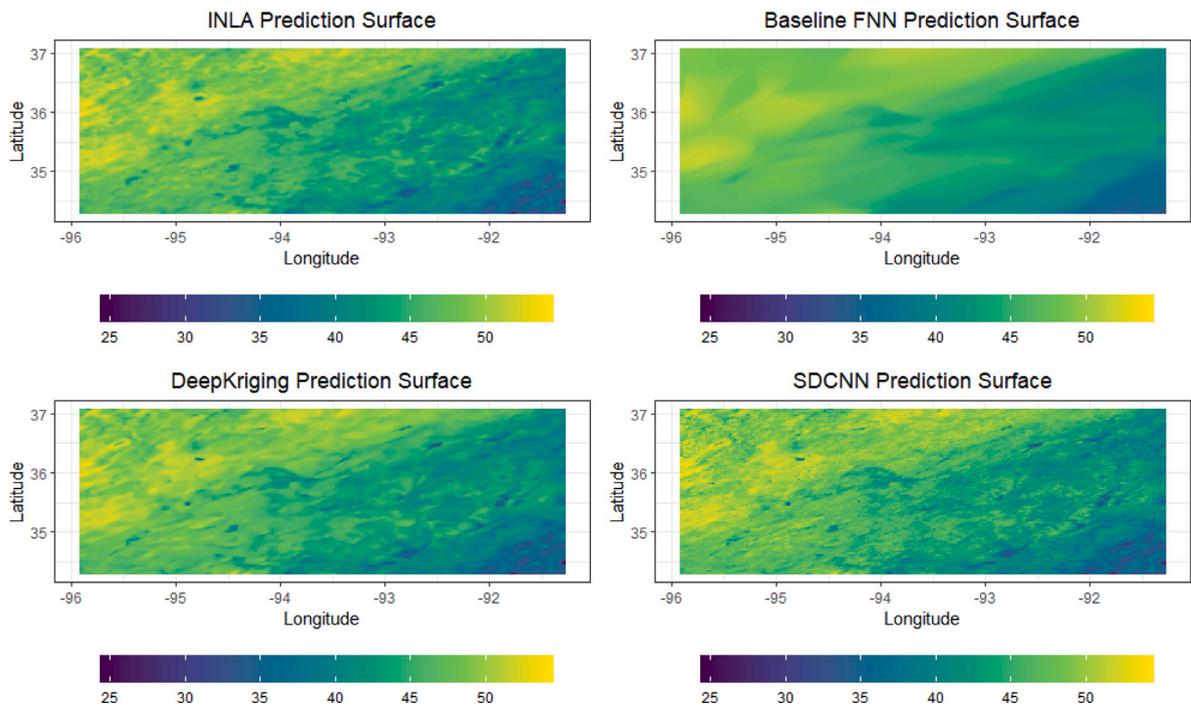
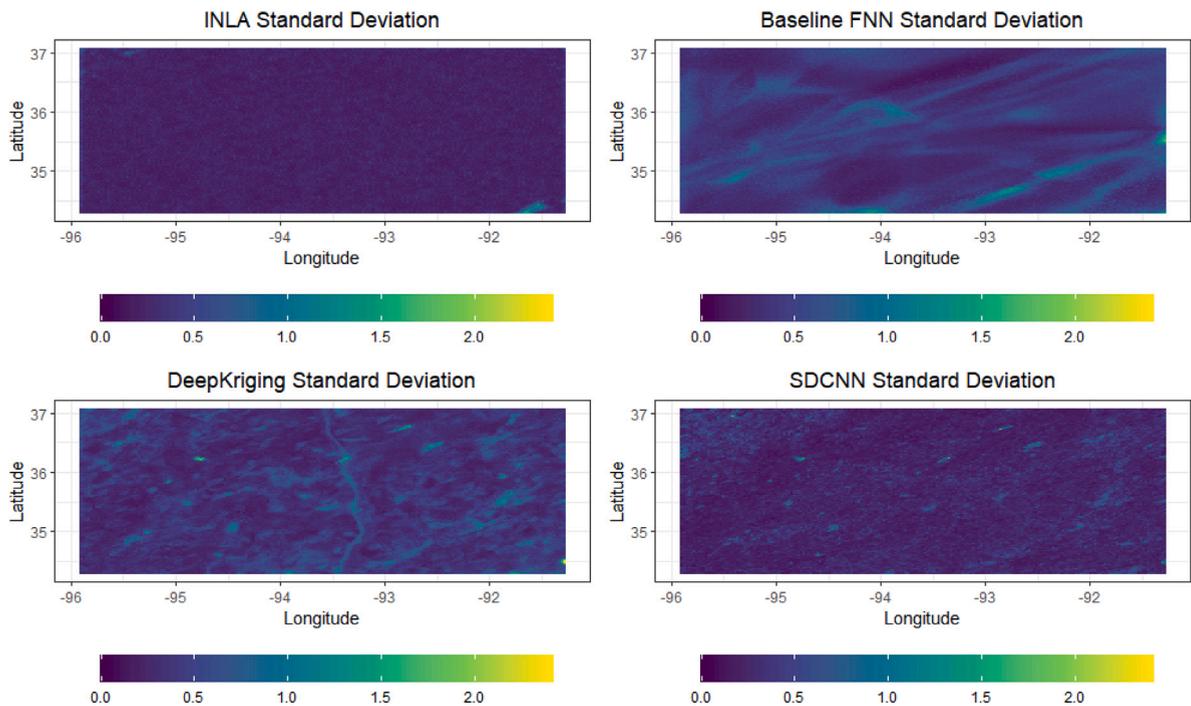**Fig. 8.** Predicted surface of the satellite temperatures.

Fig. 9. Uncertainty surface of the satellite temperatures.

Our predicted surface is shown in Fig. 13. The INLA and SDCNN surfaces better predict the observations than the other two methods, presumedly because they more accurately explore local data features. Although the DeepKriging and Baseline DNN methods both suggest that the Eastern United States has larger SOCs, they perform poorly in the Central United States.
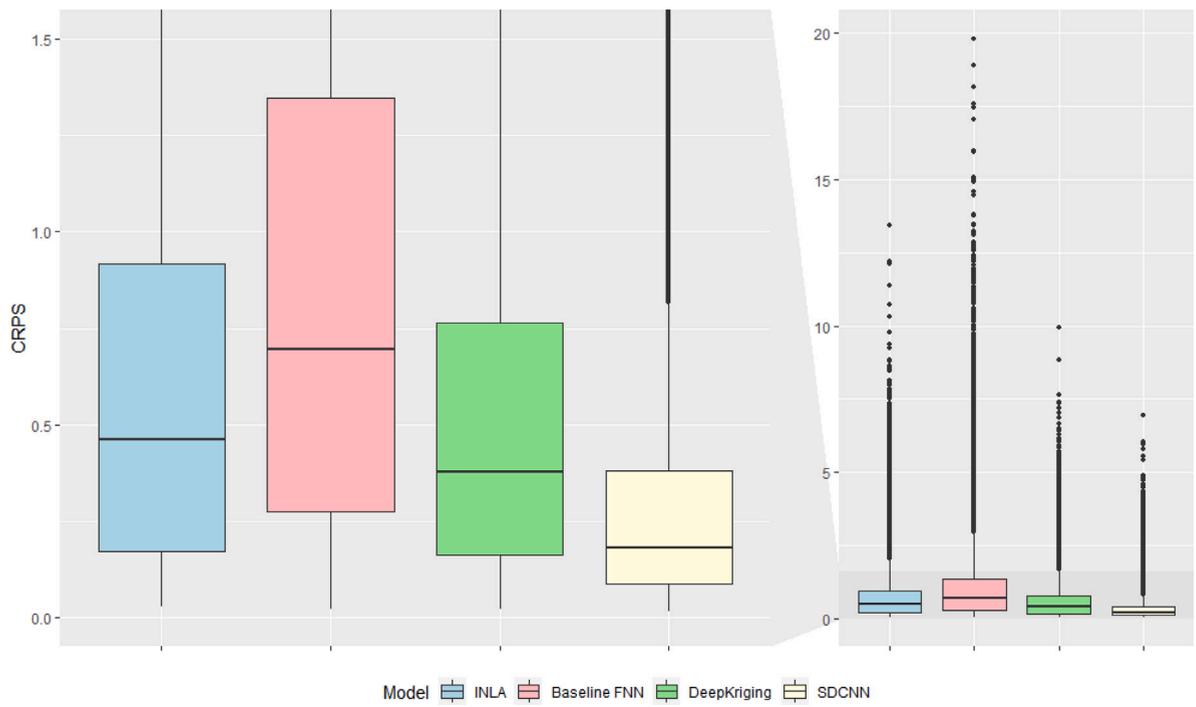
Fig. 10. Boxplot of CRPS by method for the satellite temperatures.
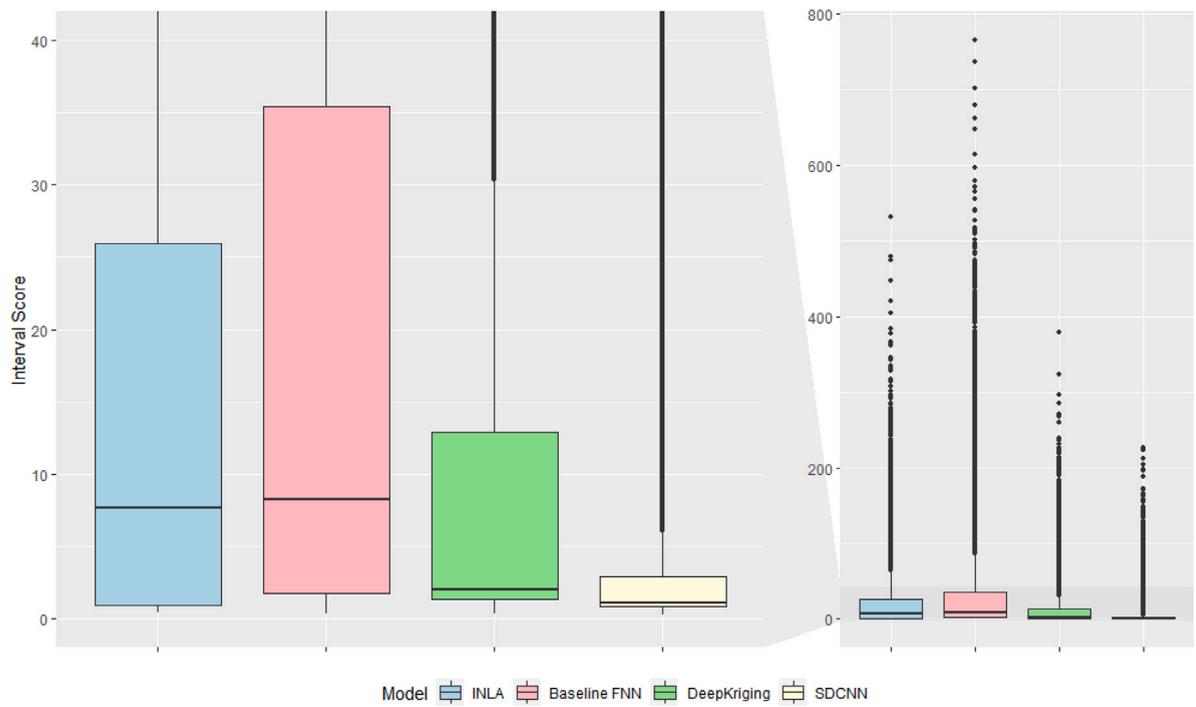
Fig. 11. Boxplot of interval scores by method for the satellite temperatures.

Observed Logarithm of Soil Organic Carbon content
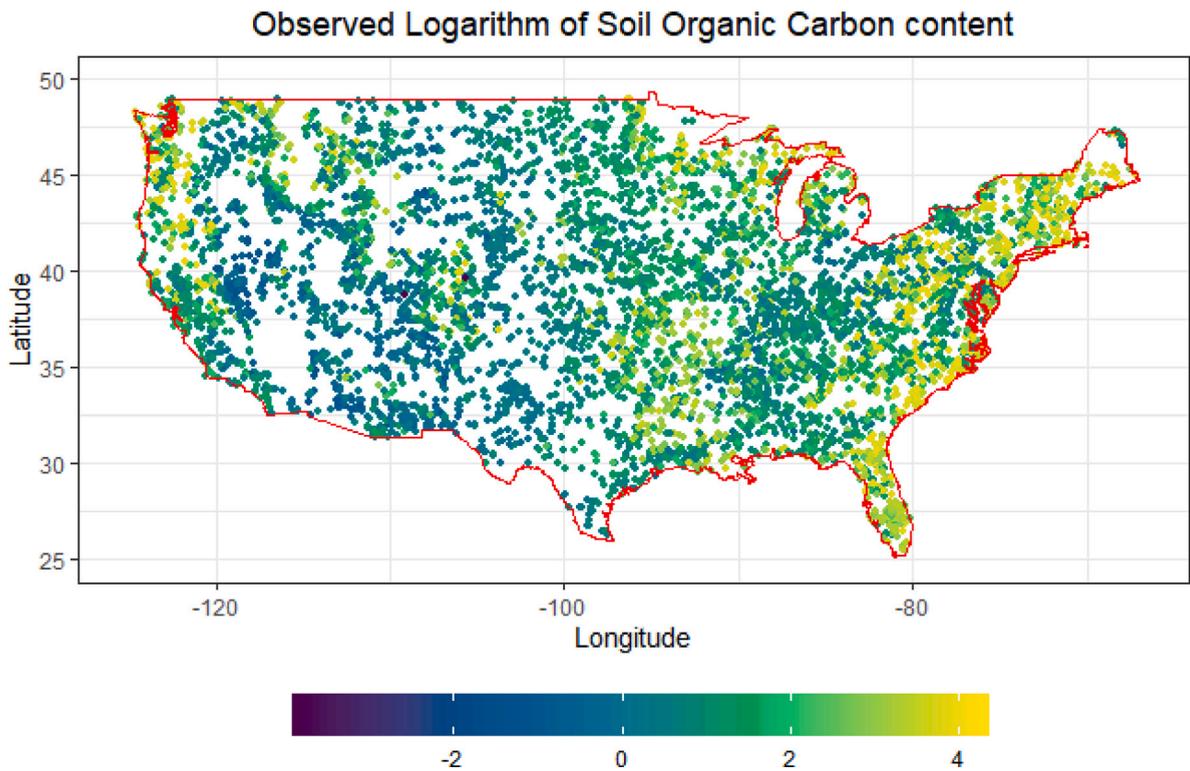
Fig. 12. Observed logarithm of the SOC data.

Table 3
Evaluation scores for the United States SOC Data.

	MSE	CRPS	ICR(0.95)	IS(0.05)
INLA-SPDE	53.32	−3.56	**0.76**	82.08
Baseline FNN	109.08	−5.95	0.32	166.64
DeepKriging	56.09	−3.49	0.56	79.97
SDCNN	**43.89**	**−2.90**	0.60	**77.81**

Fig. 14 illuminates the drawbacks of INLA, which performs more poorly here than with the satellite temperatures, presumedly because the data are more non-lattice. The DeepKriging, Baseline DNN, and SDCNN methods perform similarly, but the SDCNN predictions show a more detailed local structure.

The evaluation scores of these four models are shown in Table 3. The SDCNN model has the best scores for all criteria except the ICRs, where INLA performs better. However, the ISs for the SDCNN are better than those from INLA. CRPSs for the four models are shown in Fig. 15. The SDCNN method again has the smallest quartiles.

7. Discussion

This research proposed a new spatial prediction model and compared it to several standard models, including INLA, Baseline DNN, and DeepKriging. We first generated RBFs by using the FRK package in R and used this information as image covariates in our model. The SDCNN method was implemented in Keras in R, using image covariates. Model uncertainty was addressed via dropout layers, which generate prediction samples that provide a predictive error surface and enable model evaluation scores to be calculated.

Our SDCNN was compared to other techniques in a simulation study where the response variable was the Eggholder function surface. Two real data sets were also considered, one where locations lie primarily on a lattice and one where the observations were more irregularly spaced. The SDCNN outperformed the other techniques in most evaluation scores, although not uniformly so.

Some additional research avenues are evident. Future work includes developing convolutional recurrent neural networks (CRNNs) to analyze spatio-temporal models, where RBF generation is handled as here. Temporal aspects of the problem could also employ a CRNN, producing a spatio-temporal model that capably handles geographical time series data. Discrete data, particularly count structures, also merit study. This could be as straightforward as changing the activation and loss functions, or may demand a more detailed procedure involving the count's marginal distribution(s). How to tailor the RBF generation to the problem at hand is also worth considering.
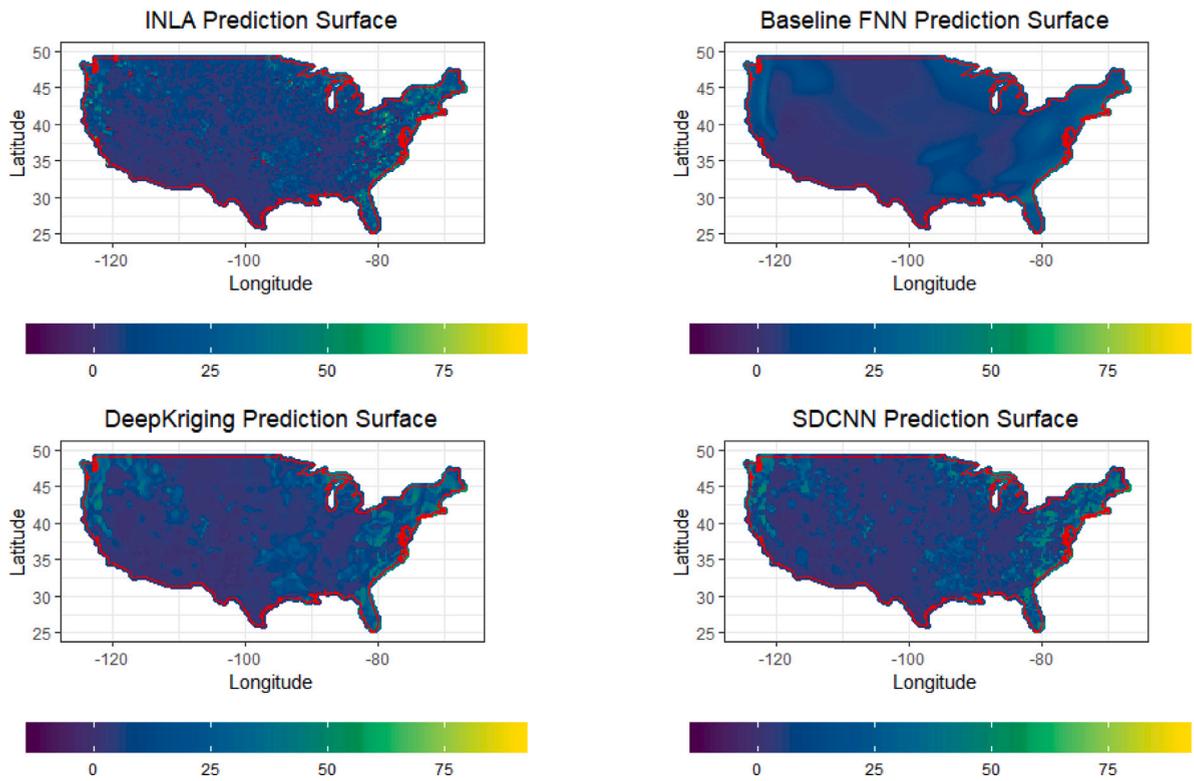
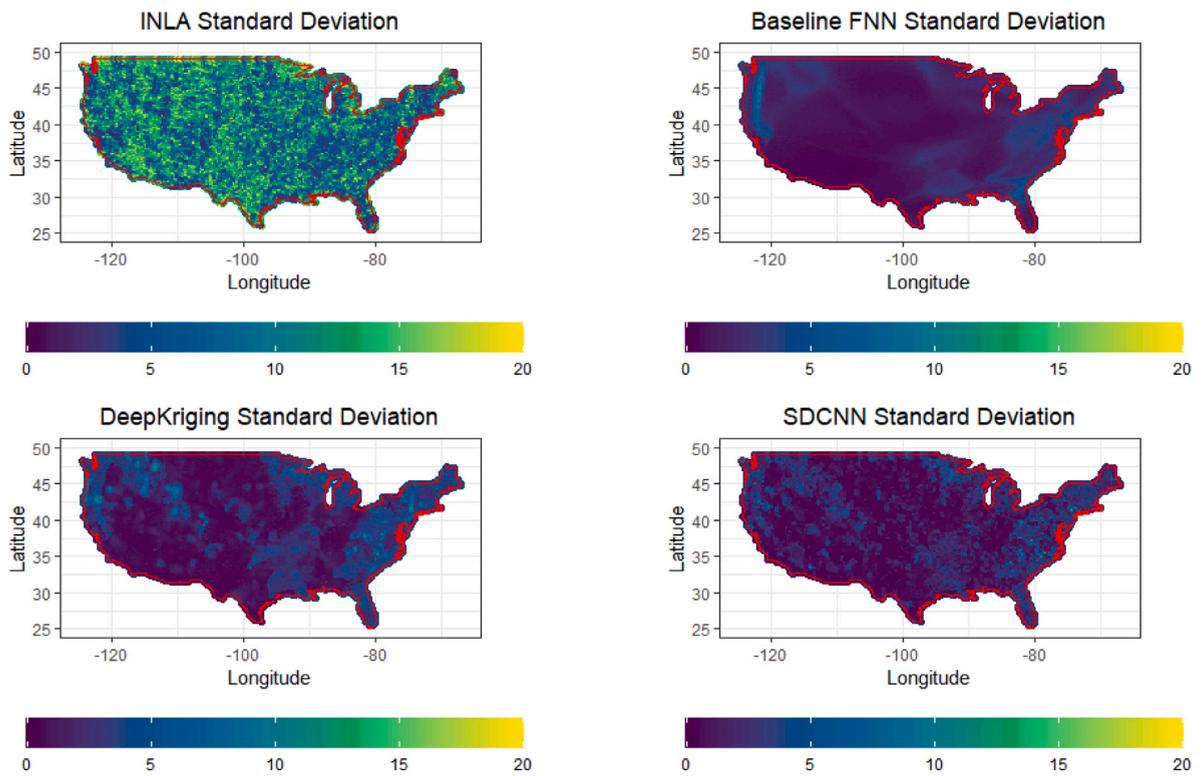**Fig. 13.** Predictions of the SOC data on a fine-scale grid.

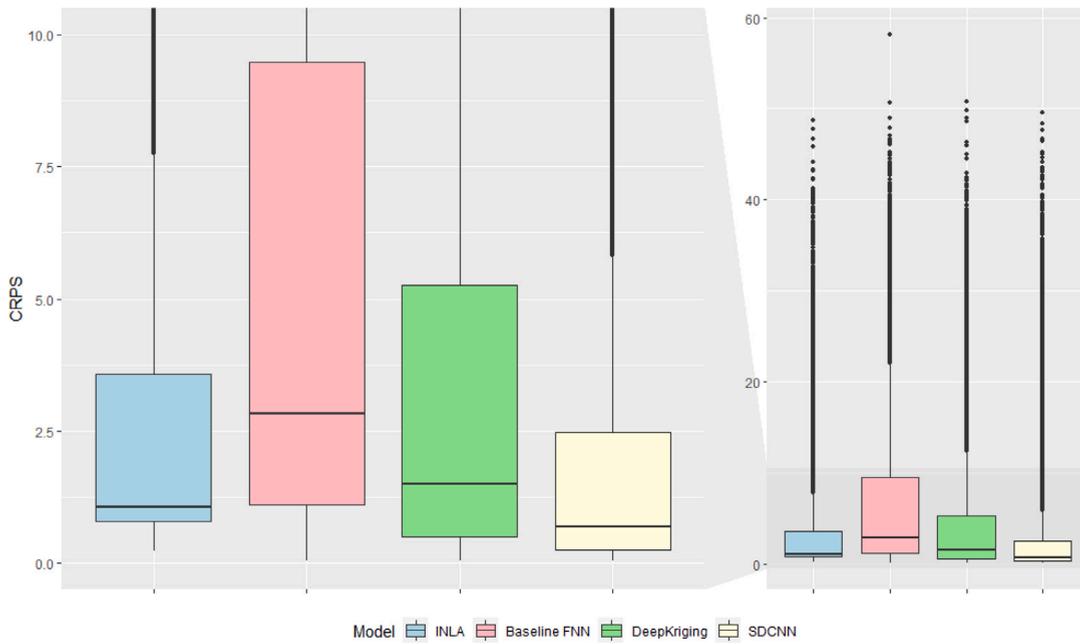**Fig. 14.** Uncertainty surface for the predicted SOC data of the last figure.

Fig. 15. CRPS for the United States SOC data.

Appendix. Model visualization

This Appendix shows how our neural networks in this paper were structured, including the Baseline DNN, DeepKriging, and SDCNN models.

A.1. Baseline DNN

This model only uses longitude and latitude as covariates. The model is depicted in the following graphic Fig. A.16.

A.2. DeepKriging

This model, depicted by the graphic below, adds three resolutions of RBFs generated by the package FRK. Bases 1–3 refer to the three RBF resolutions in the figure. No uniform basis function size exists since the number and dimension of RBFs depends on the spatial range of the data examined. Details about the neural network setting are shown in Fig. A.17.

A.3. SDCNN

This model first convolves each of the three RBF resolutions, as shown in Fig. A.18. These are subsequently concatenated via longitude and latitude. Each resolution uses a similar structure; hence, only one resolution is displayed in the figure. Here, the resolution i RBFs have m_i rows and n_i columns. The three resolutions and coordinates are calculated individually before concatenation.

After the three RBF resolutions, a model similar in structure to a baseline DNN is used for the longitudes and latitudes, concatenating these together. This procedure is graphically depicted in Fig. A.19. With all three RBF resolutions and coordinates dealt with, our next step concatenates these and feeds the result through an output layer. Fig. A.20 depicts the procedure.
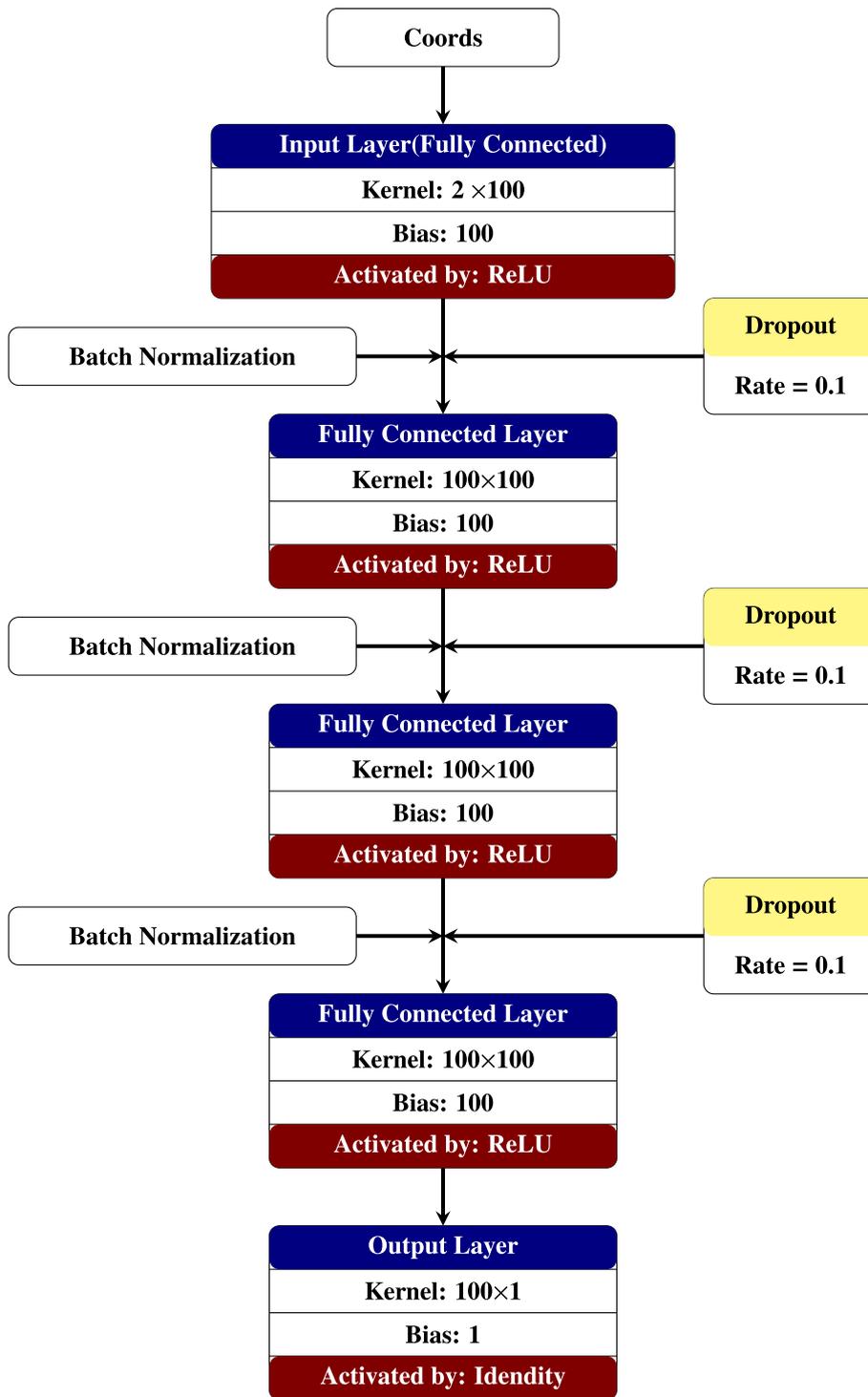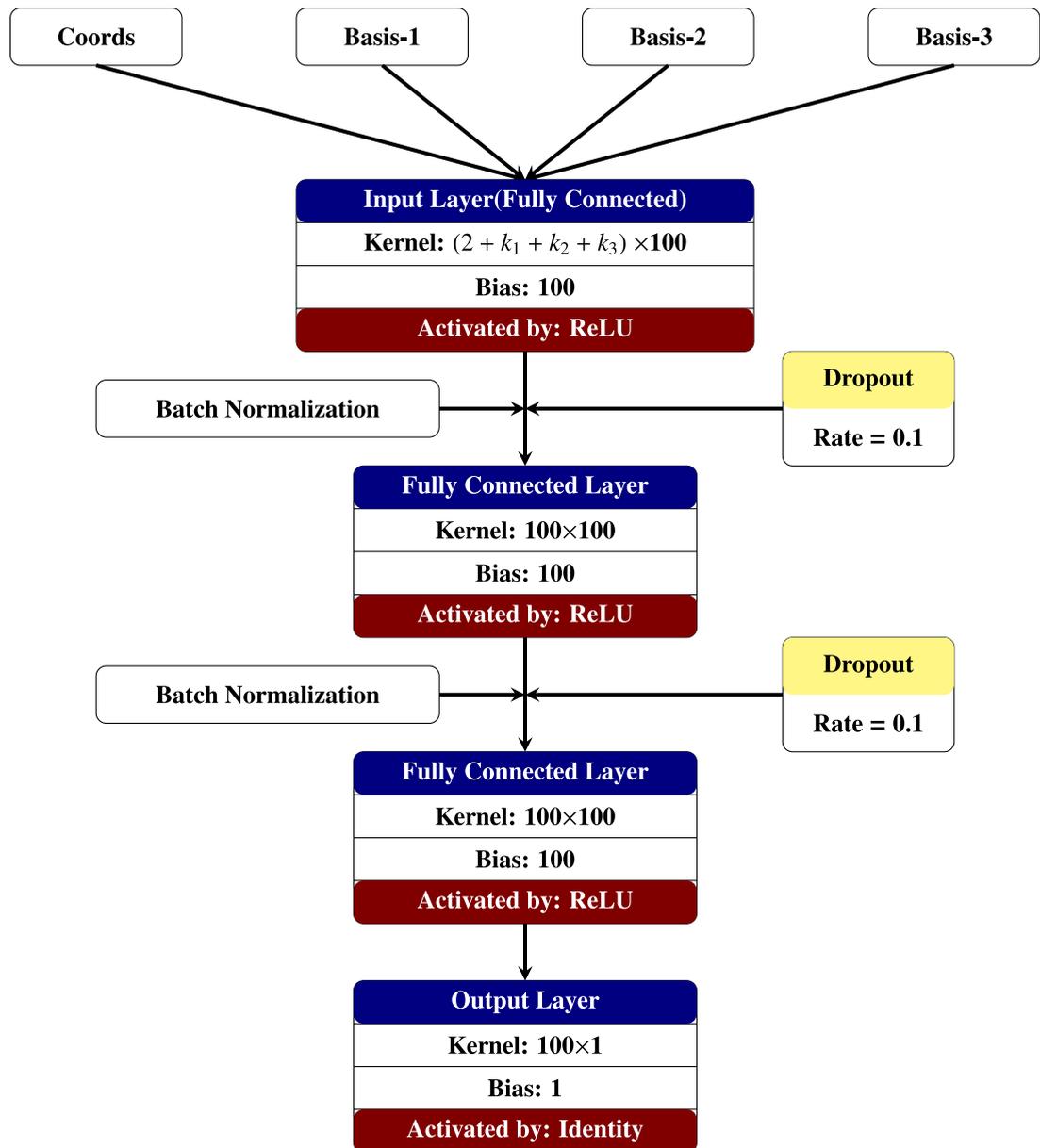
Fig. A.16. Baseline DNN model visualization.

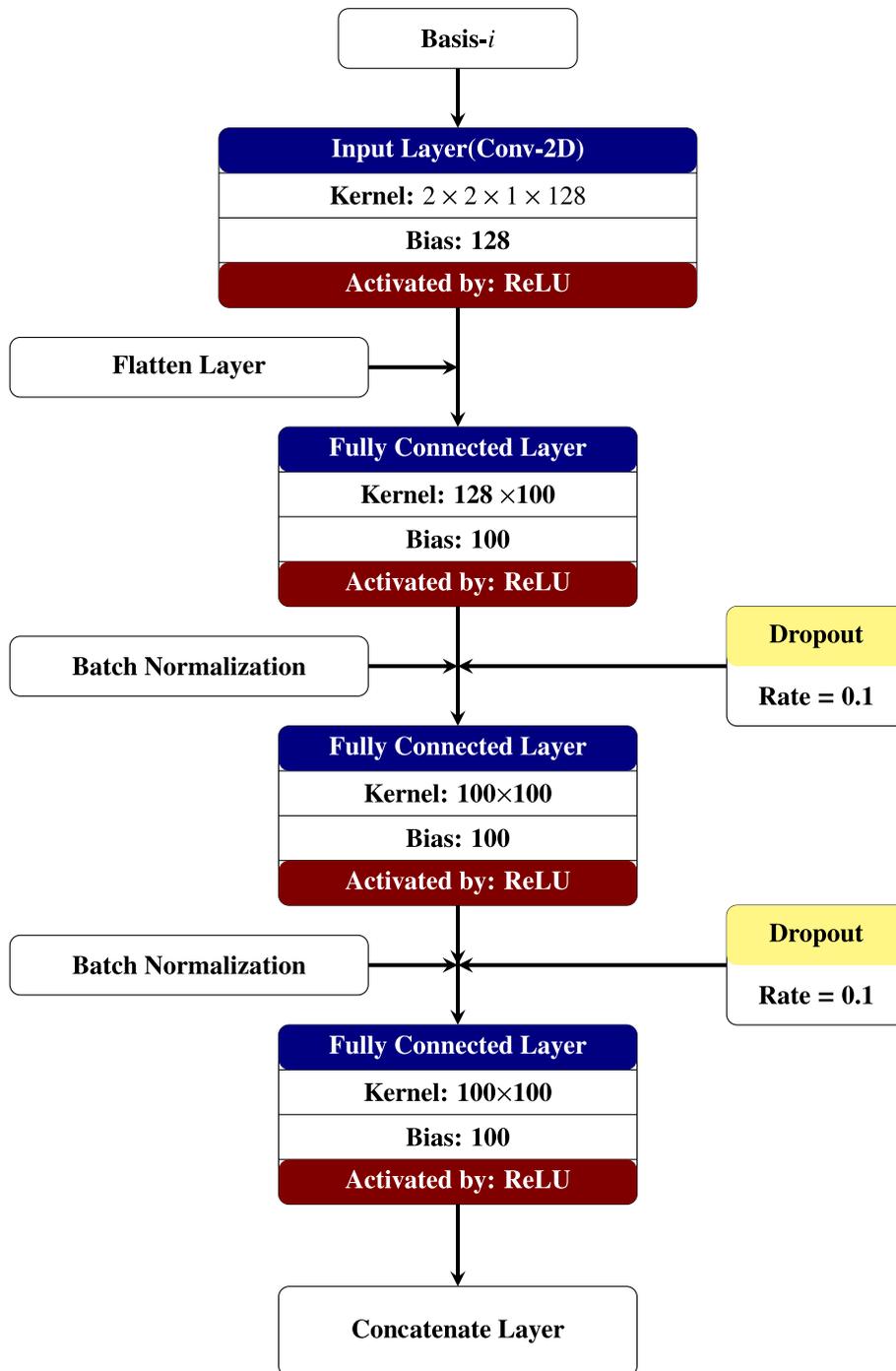**Fig. A.17.** DeepKriging model visualization.

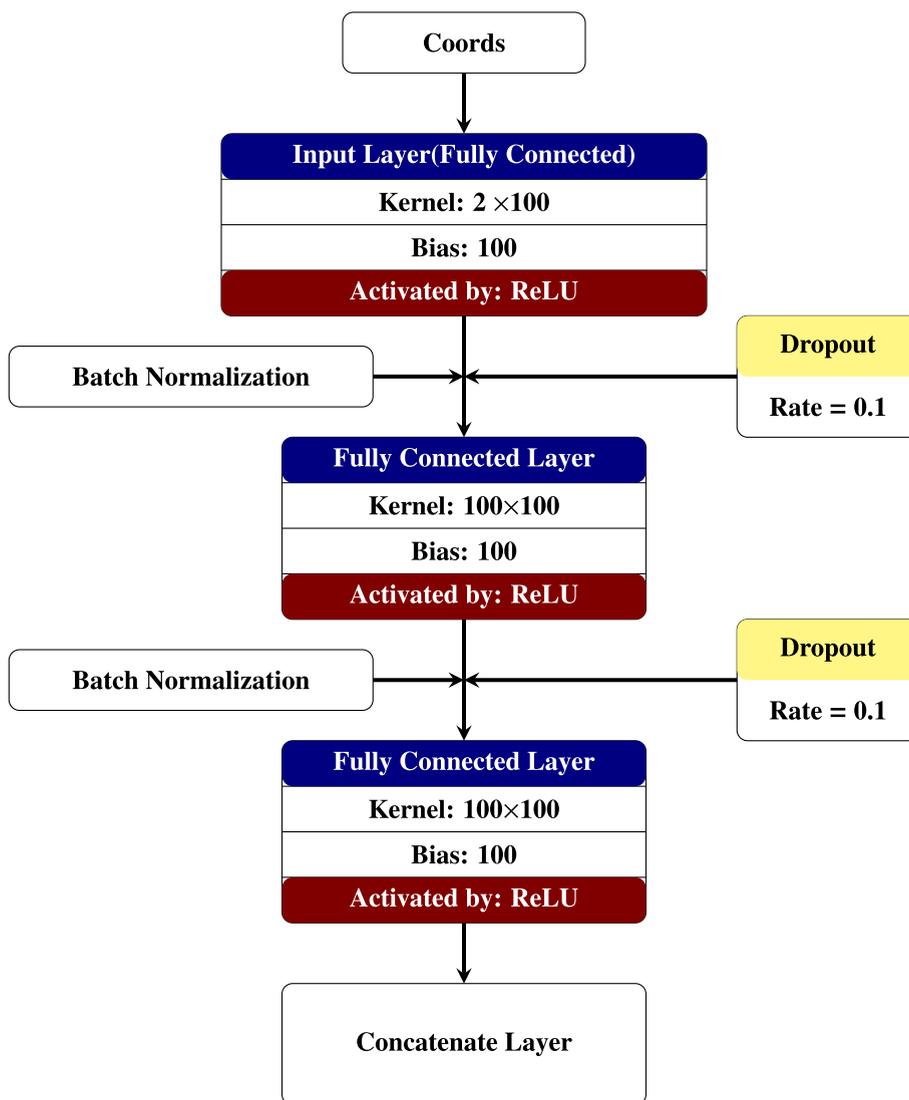**Fig. A.18.** SDCNN model visualization of convolutional part.

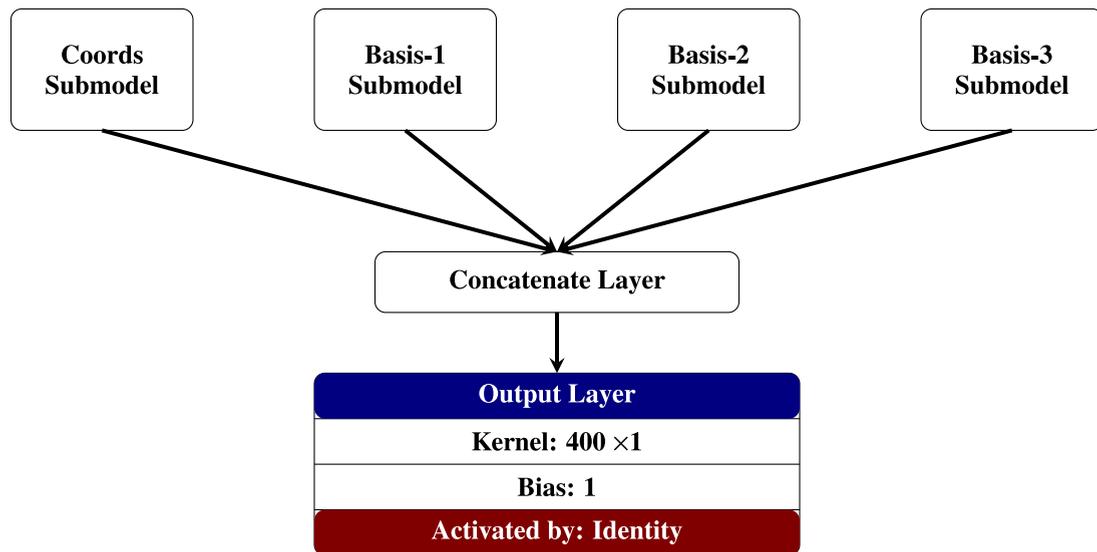**Fig. A.19.** SDCNN model visualization of covariates part.

Fig. A.20. SDCNN model visualization of concatenating and output layers.

References

Albawi, S., Mohammed, T.A., Al-Zawi, S., 2017. Understanding of a convolutional neural network. In: 2017 International Conference on Engineering and Technology. ICET, IEEE, pp. 1–6.

Araki, S., Yamamoto, K., Kondo, A., et al., 2015. Application of regression Kriging to air pollutant concentrations in Japan with high spatial resolution. Aerosol Air Qual. Res. 15 (1), 234–241.

Arnold, T.B., 2017. kerasR: R interface to the keras deep learning library. J. Open Source Softw. 2 (14), 296.

Banerjee, S., Gelfand, A.E., Finley, A.O., Sang, H., 2008. Gaussian predictive process models for large spatial data sets. J. R. Stat. Soc. Ser. B Stat. Methodol. 70 (4), 825–848.

Baringhaus, L., Franz, C., 2004. On a new multivariate two-sample test. J. Multivariate Anal. 88 (1), 190–206.

Beaudette, D., Skovlin, J., Roecker, S., Beaudette, M.D., 2023. Package 'soilDB'.

Bornn, L., Shaddick, G., Zidek, J.V., 2012. Modeling nonstationary processes through dimension expansion. J. Amer. Statist. Assoc. 107 (497), 281–289.

Cellura, M., Cirrincione, G., Marvuglia, A., Miraoui, A., 2008. Wind speed spatial estimation for energy planning in Sicily: A neural Kriging application. Renew. Energy 33 (6), 1251–1266.

Chen, W., Li, Y., Reich, B.J., Sun, Y., 2024. DeepKriging: Spatially dependent deep neural networks for spatial prediction. Statist. Sinica 34, 291–311.

Cracknell, M.J., Reading, A.M., 2014. Geological mapping using remote sensing data: A comparison of five machine learning algorithms, their response to variations in the spatial distribution of training data and the use of explicit spatial information. Comput. Geosci. 63, 22–33.

Cressie, N., Johannesson, G., 2008. Fixed rank Kriging for very large spatial data sets. J. R. Stat. Soc. Ser. B Stat. Methodol. 70 (1), 209–226.

Daw, R., Wikle, C.K., 2023. REDS: Random ensemble deep spatial prediction. Environmetrics 34 (1), e2780.

Dunsmore, I., 1968. A Bayesian approach to calibration. J. R. Stat. Soc. Ser. B Stat. Methodol. 30 (2), 396–405.

Finley, A.O., Datta, A., Banerjee, S., 2020. spNNGP R package for nearest neighbor Gaussian process models. arXiv preprint arXiv:2001.09111.

Gal, Y., Ghahramani, Z., 2016. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In: International Conference on Machine Learning. PMLR, pp. 1050–1059.

Gneiting, T., Raftery, A.E., 2007. Strictly proper scoring rules, prediction, and estimation. J. Amer. Statist. Assoc. 102 (477), 359–378.

Gramacy, R.B., Lee, H.K.H., 2008. Bayesian treed Gaussian process models with an application to computer modeling. J. Amer. Statist. Assoc. 103 (483), 1119–1130.

Heaton, M.J., Datta, A., Finley, A.O., Furrer, R., Guinness, J., Guhaniyogi, R., Gerber, F., Gramacy, R.B., Hammerling, D., Katzfuss, M., et al., 2019. A case study competition among methods for analyzing large spatial data. J. Agric. Biol. Environ. Stat. 24, 398–425.

Higdon, D., 1998. A process-convolution approach to modelling temperatures in the North Atlantic Ocean. Environ. Ecol. Stat. 5, 173–190.

Hooshmand, A., Delghandi, M., Izadi, A., Aali, K.A., et al., 2011. Application of Kriging and coKriging in spatial estimation of groundwater quality parameters. Afr. J. Agric. Res. 6 (14), 3402–3408.

Kim, H.-M., Mallick, B.K., Holmes, C.C., 2005. Analyzing nonstationary spatial data using piecewise Gaussian processes. J. Amer. Statist. Assoc. 100 (470), 653–668.

Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kirsner, D., Sansó, B., 2020. Multi-scale shotgun stochastic search for large spatial datasets. Comput. Statist. Data Anal. 146, 106931.

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. Adv. Neural Inf. Process. Syst. 25.

LeCun, Y., Bengio, Y., et al., 1995. Convolutional networks for images, speech, and time series. Handb. Brain Theory Neural Netw. 3361 (10), 1995.

Lemos, R.T., Sansó, B., 2009. A spatio-temporal model for mean, anomaly, and trend fields of North Atlantic sea surface temperature. J. Amer. Statist. Assoc. 104 (485), 5–18.

Lemos, R.T., Sansó, B., 2012. Conditionally linear models for non-homogeneous spatial random fields. Stat. Methodol. 9 (1–2), 275–284.

Lindgren, F., Rue, H., Lindström, J., 2011. An explicit link between Gaussian fields and Gaussian Markov random fields: the stochastic partial differential equation approach. J. R. Stat. Soc. Ser. B Stat. Methodol. 73 (4), 423–498.

Martino, S., Rue, H., 2009. Implementing Approximate Bayesian Inference Using Integrated Nested Laplace Approximation: A Manual for the Inla Program. Department of Mathematical Sciences, NTNU, Norway.

Murphy, A.H., Winkler, R.L., 1979. Probabilistic temperature forecasts: The case for an operational program. Bull. Am. Meteorol. Soc. 60 (1), 12–19.

Nag, P., Sun, Y., Reich, B.J., 2023a. Bivariate DeepKriging for large-scale spatial interpolation of wind fields. arXiv preprint arXiv:2307.08038.

Nag, P., Sun, Y., Reich, B.J., 2023b. Spatio-temporal DeepKriging for interpolation and probabilistic forecasting. Spat. Stat. 57, 100773.

Neal, R.M., 2012. Bayesian Learning for Neural Networks, vol. 118, Springer Science & Business Media.

Nychka, D., Bandyopadhyay, S., Hammerling, D., Lindgren, F., Sain, S., 2015. A multiresolution Gaussian process model for the analysis of large spatial datasets. J. Comput. Graph. Statist. 24 (2), 579–599.

O'Shea, K., Nash, R., 2015. An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458.

Righetto, A.J., Faes, C., Vandendijck, Y., Ribeiro, Jr., P.J., 2020. On the choice of the mesh for the analysis of geostatistical data using R-INLA. Comm. Statist. Theory Methods 49 (1), 203–220.

Rue, H., Martino, S., Chopin, N., 2009. Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. J. R. Stat. Soc. Ser. B Stat. Methodol. 71 (2), 319–392.

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

Sampson, P.D., Guttorp, P., 1992. Nonparametric estimation of nonstationary spatial covariance structure. J. Amer. Statist. Assoc. 87 (417), 108–119.

Schmidt, A.M., O'Hagan, A., 2003. Bayesian inference for non-stationary spatial covariance structure via spatial deformations. J. R. Stat. Soc. Ser. B Stat. Methodol. 65 (3), 743–758.

Sharma, S., Sharma, S., Athaiya, A., 2017. Activation functions in neural networks. Towar. Data Sci. 6 (12), 310–316.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. 15 (1), 1929–1958.

Székely, G.J., Rizzo, M.L., 2005. A new test for multivariate normality. J. Multivariate Anal. 93 (1), 58–80.

Wikle, C.K., Zammit-Mangion, A., 2023. Statistical deep learning for spatial and spatiotemporal data. Annu. Rev. Stat. Appl. 10 (1), 247–270.

Zammit-Mangion, A., Cressie, N., 2023. Introduction to fixed rank Kriging: the R package.

Zammit-Mangion, A., Kaminski, M.D., Tran, B.-H., Filippone, M., Cressie, N., 2024. Spatial Bayesian neural networks. Spat. Stat. 100825.

Zammit-Mangion, A., Ng, T.L.J., Vu, Q., Filippone, M., 2022. Deep compositional spatial models. J. Amer. Statist. Assoc. 117 (540), 1787–1808.

Zhan, W., Datta, A., 2024. Neural networks for geospatial data. J. Amer. Statist. Assoc. 1–21.

Zhang, J., Ju, Y., Mu, B., Zhong, R., Chen, T., 2023. An efficient implementation for spatial–temporal Gaussian process regression and its applications. Automatica 147, 110679.