Journal of Computational Physics 366 (2018) 415–447

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification

Yinhao Zhu, Nicholas Zabaras *

Center for Informatics and Computational Science, 311I Cushing Hall, University of Notre Dame, Notre Dame, IN 46556, USA

ARTICLE INFO

Article history:
Received 20 January 2018
Received in revised form 15 March 2018
Accepted 24 March 2018
Available online 11 April 2018

Keywords:
Uncertainty quantification
Bayesian neural networks
Convolutional encoder–decoder networks
Deep learning
Porous media flows

ABSTRACT

We are interested in the development of surrogate models for uncertainty quantification and propagation in problems governed by stochastic PDEs using a deep convolutional encoder–decoder network in a similar fashion to approaches considered in deep learning for image-to-image regression tasks. Since normal neural networks are data-intensive and cannot provide predictive uncertainty, we propose a Bayesian approach to convolutional neural nets. A recently introduced variational gradient descent algorithm based on Stein's method is scaled to deep convolutional networks to perform approximate Bayesian inference on millions of uncertain network parameters. This approach achieves state of the art performance in terms of predictive accuracy and uncertainty quantification in comparison to other approaches in Bayesian neural networks as well as techniques that include Gaussian processes and ensemble methods even when the training data size is relatively small. To evaluate the performance of this approach, we consider standard uncertainty quantification tasks for flow in heterogeneous media using limited training data consisting of permeability realizations and the corresponding velocity and pressure fields. The performance of the surrogate model developed is very good even though there is no underlying structure shared between the input (permeability) and output (flow/pressure) fields as is often the case in the image-to-image regression models used in computer vision problems. Studies are performed with an underlying stochastic input dimensionality up to 4225 where most other uncertainty quantification methods fail. Uncertainty propagation tasks are considered and the predictive output Bayesian statistics are compared to those obtained with Monte Carlo estimates.

1. Introduction

Uncertainty in complex systems arises from model error and model parametrization, unknown/incomplete material properties, boundary conditions or forcing terms, and other. Uncertainty propagation takes place by reformulating the problem of interest as a system of stochastic partial differential equations (SPDEs). Solution of such problems often needs to rely on the solution of the deterministic problem at a finite number of realizations of the random input using Monte Carlo sampling, or collocation methods. A dominant strategy is to train a surrogate model using limited simulation-based data, and then perform prediction and uncertainty propagation tasks using the surrogate instead of solving the actual PDEs. Unfortunately

* Corresponding author.
E-mail addresses: yzhu10@nd.edu (Y. Zhu), nzabaras@gmail.com (N. Zabaras).
URL: https://cics.nd.edu/ (N. Zabaras).

https://doi.org/10.1016/j.jcp.2018.04.018
0021-9991/

most existing surrogate models have difficulty scaling to high-dimensional problems, such as the ones based on Gaussian processes (GP) [1–3] or generalized polynomial chaos expansions (gPC [4]). High dimensionality often arises from the discretization of properties with small correlation lengths (e.g. permeability in heterogeneous media flows), random distributed sources or force input fields with multiple scales [5].

To alleviate the curse of stochastic input dimensionality, we usually assume that the given input data lie on an embedded non-linear manifold within the higher-dimensional space. This intrinsic dimensionality is captured by dimensionality reduction techniques [6], such as the Karhunen–Loève expansion (KLE), t-SNE [7], auto-encoders [8], probabilistic methods like variational auto-encoders [9], Gaussian process latent variable models (GP-LVM) [10], and many more. Most dimensionality reduction models are unsupervised learning problems that do not explicitly take the regression task into account. Thus the classical approach to uncertainty quantification is to first reduce the dimensionality of the input to obtain a low-dimensional latent representation, then to build a regression model from this latent representation to the output. This approach is certainly not efficient in particular when the map from the latent representation to the physical space of the input data is not available. In [2], KLE was used for dimensionality reduction of the permeability field and then GP was performed as independent task for Bayesian regression. In [11], this approach was taken one step further with the probabilistic mappings from input to latent space and from latent space to output being modeled by generalized linear models both trained simultaneously end-to-end using stochastic variational inference.

One of the essential upcoming approaches for handling high-dimensional data is to learn the latent input representation automatically by supervision with the output. This is the central idea of deep neural networks [12], especially convolutional neural networks (CNNs) [13] which stack (deeper) layers of linear convolutions with nonlinear activations to automatically extract the multi-scale features or concepts from high-dimensional input [14], thus alleviating the hand-craft feature engineering, such as searching for the right set of basis functions, or relying on expert knowledge.

However, the general perspective for using deep neural networks [15,16] in the context of surrogate modeling is that physical problems in uncertainty quantification (UQ) are not big data problems, thus not suitable for addressing them with deep learning approaches. However, we argue otherwise in the sense that each simulation run generates large amount of data which potentially reveal the essential characteristics about the underlying physical system, especially in dynamic systems. In addition, even for a relatively small dataset, well-designed deep neural networks show unique generalization property [17–19]. These are typically over-parameterized models (hundreds and thousands of times more parameters than training data), but they tend not to overfit, i.e. the test error does not grow as the network parameters increase.

This unique generalization behavior makes it feasible to use deep neural networks for surrogate modeling. They are capable of capturing the complex nonlinear mapping between high-dimensional input and output due to their expressiveness [20], while they only use small number of data from simulation runs. Deep learning has been explored as a competitive methodology across fields such as fluid mechanics [21], hydrology [22,23], bioinformatics [24], high energy physics [25] and other. In addition, there has been a resurgence of interest in putting deep neural networks under a formal Bayesian formalism. Bayesian deep learning [26–34] enables the network to express its uncertainty on its predictions when using a small number of training data. The Bayesian neural networks can quantify the predictive uncertainty by treating the network parameters as random variables, and perform Bayesian inference on those uncertain parameters conditioned on limited observations.

In this work, we mainly consider surrogate modeling of physical systems governed by stochastic partial differential equations with high-dimensional stochastic input such as flow in random porous media [23]. The spatially discretized stochastic input field and the corresponding output fields are high-dimensional. We adopt an end-to-end image-to-image regression approach for this challenging surrogate modeling problem. More specifically, a fully convolutional encoder–decoder network is designed to capture the complex mapping directly from the high-dimensional input field to the output fields without using any explicit intermediate dimensionality reduction method. To make the model more parameter efficient and compact, we use DenseNet to build the feature extractor within the encoder and decoder paths [35]. Intuitively, the encoder network extracts the multi-scale features from the input data which are used by the decoder network to reconstruct the output fields. In similarity with problems in computer vision, we treat the input-output map as an image-to-image map. To account for the limited training data and endow the network with uncertainty estimates, we further treat the convolutional encoder–decoder network to be Bayesian and scale a recently proposed approximate inference method called Stein Variational Gradient Descent to modern deep convolutional networks. We will show that the methodology can learn a Bayesian surrogate for a problem with an intrinsic dimensionality of 50, achieving promising results on both predictive accuracy and uncertainty estimates using as few as 32 training data. More importantly we develop a surrogate for the case of 4225 dimensionality using 512 training data. We also show these uncertainty estimates are well-calibrated using a reliability diagram. To this end, we believe that Bayesian neural networks are strong candidates for surrogate modeling and uncertainty propagation in high-dimensional problems with limited training data.

The remaining of the paper is organized as follows: In Section 2, we present the problem setup for surrogate modeling with high-dimensional input and the proposed approach in treating it as an image regression problem. We then introduce the CNNs and the encoder–decoder network used in our model. In Section 3, we present the Bayesian formulation of neural networks and a non-parametric variational method for the underlying challenging approximate inference task. In Section 4, we provide implementation details and show the experiments we conducted on a porous media flow problem. We finally conclude and discuss the various unexplored research directions in Section 5.

2. Methodology

2.1. Surrogate modeling as image-to-image regression

The physical systems considered here are modeled by stochastic PDEs (SPDEs) with solutions $\boldsymbol{y}(\boldsymbol{s}, \boldsymbol{x}(\boldsymbol{s}))$, i.e. the model response $\boldsymbol{y} \in \mathbb{R}^{d_y}$ at the spatial location $\boldsymbol{s} \in \mathcal{S} \subset \mathbb{R}^{d_s}$ ($d_s = 1, 2, 3$), with one realization $\boldsymbol{x}(\boldsymbol{s}) \in \mathbb{R}^{d_x}$ of the random field $\{\boldsymbol{x}(\boldsymbol{s}, \omega), \boldsymbol{s} \in \mathcal{S}, \omega \in \Omega\}$, where $\mathcal{S}$ is the index set and Ω is the sample space. The formulation allows for multiple input channels (i.e. $d_x > 1$) even though our interest here is on one input property represented as a scalar random field. This random field appears in the coefficients of the SPDEs, and is used to model material properties, such as the permeability or porosity fields in geological media flows. We assume the computer simulation for the physical systems is performed over a given set of spatial grid locations $\mathcal{S} = \{\boldsymbol{s}_1, \cdots, \boldsymbol{s}_{n_s}\}$ (e.g. mesh nodes in finite element methods). In this case, the random field $\boldsymbol{x}$ is discretized over the fixed grids $\mathcal{S}$, thus is equivalent to a high-dimensional random vector, denoted as $\mathbf{x}$, where $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^{d_x n_s}$. The corresponding response $\boldsymbol{y}$ is solved over $\mathcal{S}$, thus can be represented as a vector $\mathbf{y} \in \mathcal{Y} \subset \mathbb{R}^{d_y n_s}$.

With the discretization described above and assuming for simplicity fixed boundary and initial conditions and source terms, we consider the computation simulation as a black-box mapping of the form:

$$\boldsymbol{\eta} : \mathcal{X} \to \mathcal{Y}. \tag{1}$$

In order to tackle the limitations of using the deterministic computationally expensive simulator for uncertainty propagation, a *surrogate model* $\mathbf{y} = \mathbf{f}(\mathbf{x}, \boldsymbol{\theta})$ is trained using *limited* simulation data $\mathcal{D} = \{(\mathbf{x}^i, \mathbf{y}^i)\}_{i=1}^N$, to approximate the 'ground-truth' simulation-induced function $\mathbf{y} = \boldsymbol{\eta}(\mathbf{x})$, where $\boldsymbol{\theta}$ are the model parameters, and N is the number of simulation runs (number of training simulation-based data).

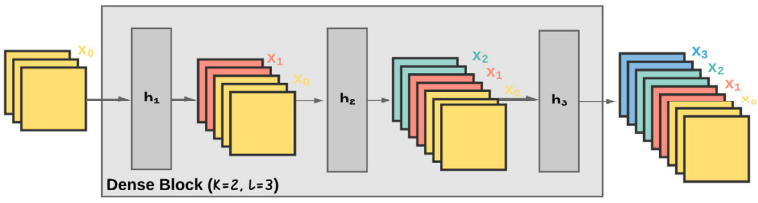
Let us consider that the PDEs governing the physical system described above are solved over 2D regular grids of $H \times W$, where H and W denote the number of grid points in the two axes of the spatial domain (height and width), and $n_s = H \times W$. It is very natural to organize the simulation data as an image dataset $\mathcal{D} = \{(\mathbf{x}^i, \mathbf{y}^i)\}_{i=1}^N$, where $\mathbf{x}^i \in \mathbb{R}^{d_x \times H \times W}$ is one input field realization, and $\mathbf{y}^i \in \mathbb{R}^{d_y \times H \times W}$ is the simulated steady-state output fields for $\mathbf{x}^i$ discretized over the same grid. Here d_x, d_y are treated herein as the number of channels in the input and output images, similar to the RGB channels in natural images. It is easy to generalize to the 3D spatial domain by adding an extra *depth* axis to the images, e.g. $\mathbf{x}^i \in \mathbb{R}^{d_x \times D \times H \times W}$, and $\mathbf{y}^i \in \mathbb{R}^{d_y \times D \times H \times W}$.

Therefore, we transform the surrogate modeling problem to an image-to-image regression problem, with the regression function as

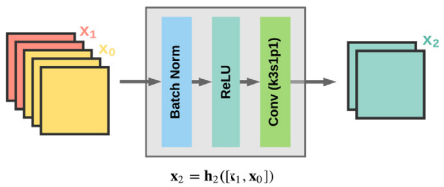
$$\boldsymbol{\eta} : \mathbb{R}^{d_x \times H \times W} \to \mathbb{R}^{d_y \times H \times W}. \tag{2}$$

In distinction from an image classification problem which requires image-wise prediction, the image regression problem is concerned with pixel-wise predictions, e.g. predicting the depth of each pixel in an image, or in our physical problem, predicting the output fields at each grid point. Such problems have been intensively studied within the computer vision community by leveraging the rapid recent progress of convolutional neural networks (CNNs), such as AlexNet [13], VGG [36], Inception [37], ResNet [38], DenseNet [35], and many more. A common model design pattern for semantic segmentation [39] or depth regression [40] is the *encoder–decoder* architecture. The intuition behind regression between two high-dimensional objects is to go through a coarse-refine process, i.e. to extract high-level coarse features from the input images using an encoder network, and then refine the coarse features to output images through a decoder network. One of the characteristics shared by those vision tasks is that the input and output images *share the underlying structure*, or they are different renderings of the same underlying structure [41]. However, for our surrogate modeling tasks, the input and output images appear to be *quite different*, due to the complex physical influences (defined by PDEs) of the random input field, forcing terms and boundary conditions on the system response. This was after all the reason of pursuing the training of a surrogate model that avoids the repeated solution of the PDEs for different input realizations. Surprisingly, as we will discuss later on in this paper, the encoder–decoder network still works very well.

Remark 1. The training data for the surrogate model of interest here include the realizations of the random input field and the corresponding multi-output obtained from simulation. Of interest is to address problems with limited training data sets considering the high-computational cost of each simulation run. A key UQ task is the ability to predict the system response and our confidence on it using input realizations (test dataset) statistically consistent with the given training data. In addition, of interest is the computation of the statistics of the response induced by the random input. Both of these tasks require the availability of a high-number of input data sets (e.g. 500 data points for testing, and 10^4 input data points for Monte Carlo calculation of the output statistics). The problem of generating more input realizations using only the training dataset is the solution of a *generative model* problem. There has been significant progress in recent years in the topic, such as the generative adversarial networks (GANs) [42] and its ever *exploding* variants, variational auto-encoders (VAEs) [9], autoregressive models like PixelCNN [43], PixelRNN [44], and other. Since in this work our focus is on the image-to-image mapping and its performance on uncertainty quantification tasks, we will assume that enough input samples are provided both for testing and output statistics calculation.



(a) Dense block.



(b) The second layer.

Fig. 1. (a) A dense block contains $L = 3$ layers h_1, h_2, h_3 with growth rate $K = 2$. (b) The second layer h_2 of the dense block, where $\mathbf{x}_2 = h_2([\mathbf{x}_1, \mathbf{x}_0])$ is its output feature map. Notice that the input to the third layer is the concatenation of the output and input features of h_2, i.e. $[\mathbf{x}_2, \mathbf{x}_1, \mathbf{x}_0]$. As is often the case, each layer is composed of Batch Normalization [46] (BatchNorm), Rectified Linear Unit [47] (ReLU) and Convolution (Conv). The convolution kernel has size $k = 3$, stride $s = 1$ and zero padding $p = 1$, which keep the size of the feature maps the same as the input.

2.2. Dense convolutional encoder–decoder networks

In this subsection, we briefly introduce a state-of-the-art CNN architecture called DenseNet [35] and fully convolutional encoder–decoder networks developed in computer vision, and then present how to utilize these advances to build our baseline network for surrogate modeling in uncertainty quantification tasks.

2.2.1. Densely connected convolutional networks

DenseNet [35] is a recently proposed CNN architecture which extends the ideas of ResNet [38] and Highway Networks [45] to create dense connections between all layers, so as to improve the information (gradient) flow through the network for better parameter efficiency.

Let $\mathbf{x}_l$ be the output of the lth layer. Traditionally CNNs pass the output of one layer only to the input of the next layer, i.e. $\mathbf{x}_l = h_l(\mathbf{x}_{l-1})$, where h_l denotes the nonlinear function of the lth hidden layer. In current CNNs, h is commonly defined as a composition of Batch Normalization [46] (BatchNorm), Rectified Linear Unit [47] (ReLU) and Convolution (Conv) or transposed convolution (ConvT) [48]. ResNets [38] create an additional identity mapping that bypasses the nonlinear layer, i.e. $\mathbf{x}_l = h_l(\mathbf{x}_{l-1}) + \mathbf{x}_{l-1}$. In this way, the nonlinear layer only needs to learn a residual function which facilitates the training of *deeper* networks.

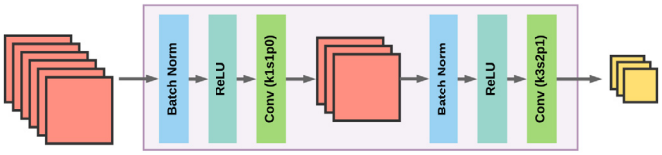
DenseNets [35] introduce connections from any layer to all subsequent layers, i.e. $\mathbf{x}_l = h_l([\mathbf{x}_{l-1}, \mathbf{x}_{l-2}, \cdots, \mathbf{x}_0])$. To put it in another way, the input features of one layer are concatenated to the output features of this layer and this serves as the input features to the next layer. Assume that the input image has K_0 channels, and each layer outputs K feature maps, then the lth layer would have input with $K_0 + (l-1) \cdot K$ feature maps, i.e. the number of feature maps in DenseNet grows linearly with the depth. K is here referred to as the *growth rate*.

For image regression based on encoder–decoder networks, down-sampling and up-sampling are required to change the size of feature maps, which makes concatenation of feature maps unfeasible. Dense blocks and transition layers are introduced to solve this problem and modularize the network design. A *dense block* contains multiple densely connected layers whose input and output feature maps are of the same size. It contains two design parameters, namely the number L of layers within and the growth rate K for each layer. An illustration of the dense block is in Fig. 1.

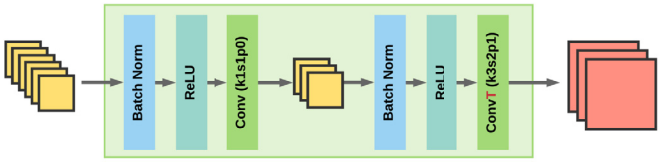
Transition layers are used to change the size of feature maps and reduce their number between dense blocks. More specifically, the encoding layer typically halfs the size of feature maps, while the decoding layer doubles the feature map size. Both of the two layers reduce the number of feature maps. This is illustrated in Fig. 2.

2.2.2. Fully convolutional encoder–decoder networks

The fully convolutional networks (FCNs) [49] are extensions of CNNs for pixel-wise prediction, e.g. semantic segmentation or depth estimation. FCNs replace the fully connected layers in CNNs with convolutional layers, add up-sampling layers in the end to recover the input spatial resolution, and introduce skip connections between the feature maps in the down-sampling and up-sampling paths to recover finer information lost in the down-sampling path. Several recent works such as Unet [39], Segnet [50] focus on improving the up-sampling path and increasing the connectivity within and between up-sampling and down-sampling paths. Fully convolutional DenseNets [51] extend DenseNets to FCNs, which are closest to our network design but with several differences. Our design addresses the explosion in the number of feature maps by reducing



(a) Encoding layer.



(b) Decoding layer.

Fig. 2. Both the (a) encoding and (b) decoding layers contain two convolutions. The first convolution reduces the number of feature maps while keeps their size the same using a kernel with parameters $k = 1, s = 1, p = 0$; the second convolution changes the size of the feature maps but not their number using a kernel $k = 3, s = 2, p = 1$. The main difference between (a) and (b) is in the type of the second convolution layer, which is Conv and ConvT, respectively, for down-sampling and up-sampling. Note that no pooling is used at transition layers for maintaining the location information. The colored feature maps used here are independent from the feature maps with the same color shown in other figures.

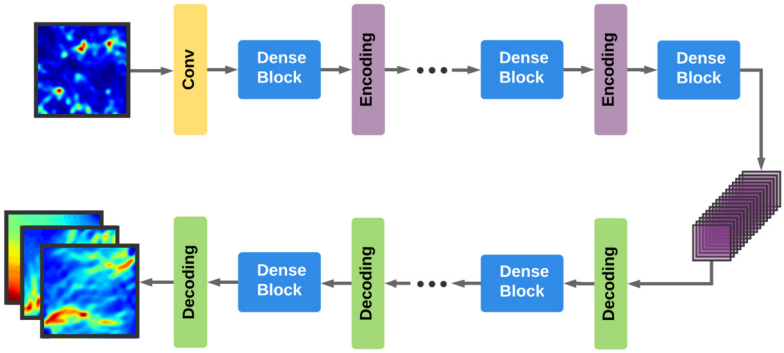


Fig. 3. Network architecture: DenseED.

the number of feature maps using the first convolution layer in the transition layer, while [51] only keeps K (growth rate) output feature maps of the last convolution layer in each dense block instead of all the feature maps from the previous layers within the block. Besides that we do not use skip connections between down-sampling and up-sampling paths because of the weak correspondence between the input and output images. We also do not use max-pooling for down-sampling in the encoding layers, instead we use convolution with stride 2.

2.3. Network architecture: DenseED

We follow the fully convolutional networks (FCNs) [49] for image segmentation without using any fully connected layers, and encoder–decoder architecture similar to U-net [39] and SegNet [50] but without the concatenation of feature maps between the encoder paths and decoder paths. Furthermore, we adapt the DenseNet [35] structure into the encoder and decoder networks. After extensive hyperparameter and architecture search, we arrived at a baseline dense convolutional encoder–decoder network, called `DenseED`, similar to the network proposed in [51] but with noticeable differences as stated above and shown in Fig. 3.

In the encoding path, the input field realizations are fed into the first convolution layer with large kernel size $k = 7$, stride $s = 2$ and zero padding $p = 2$. Then the extracted feature maps are passed through an alternative cascade of dense blocks and encoding layers as introduced in Figs. 1 and 2. The dense block after the last encoding layer outputs the high-level coarse feature maps extracted from the input, as shown in purple at the right end of the network in Fig. 3, which are subsequently fed into the decoder path. The decoding network consists of an alternation of dense blocks and decoding layers, with the last decoding layer directly leading to the prediction of the output fields.

2.4. Network architecture engineering and hyperparameter search

Network *architecture engineering* and hyperparameter search are among the main challenges and source of innovations in deep learning, mostly problem-specific and empirical. The general network architecture is introduced in Section 2.2, which is based on the recent development of neural network design for image segmentation. For our image regression problem, the main design considerations include the following:

- Down-sampling layers: convolution or pooling;
- Up-sampling layers: bilinear up-sampling or transposed convolution;
- Smallest spatial dimensions of feature maps: this is determined by the number of down-sampling layers;
- Add or not of skip connections between the encoding and decoding paths;
- Kernel of convolution layers, kernel size k, stride s, zero padding p;
- Number of layers L and growth rate K within each dense block;
- Regularizations: weight decay, batch normalization, dropout, etc.;
- Optimizer: stochastic gradient descent algorithms and their variants, such as Adam, Adagrad, RMSprop, and others;
- Training hyperparameters: batch size, learning rate and its scheduler.

The details of architecture search and hyperparameter selection for the particular problem considered are presented in Appendix A where we also report various experiments using DenseED for surrogate modeling with limited training data. No overfitting was observed in our calculations and the obtained results were quite good. This is an intriguing and active research topic in the deep learning community [17].

In our non-Bayesian calculations, we have considered $\mathcal{L}_2$ or $\mathcal{L}_1$ regularized MSE training loss function. Given an input image $\mathbf{x}$, and a target image $\mathbf{y}$, the prediction $\mathbf{f}(\mathbf{x}, \mathbf{w})$, the regularized MSE loss is

$$L(\mathbf{f}(\mathbf{x}, \mathbf{w}), \mathbf{y}) = \frac{1}{n} \sum_{i=1}^{n} \Big(\mathbf{f}_i - \mathbf{y}_i \Big)^2 + \alpha \Omega(\mathbf{w}), \tag{3}$$

where the penalty function $\Omega(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top \mathbf{w}$ for $\mathcal{L}_2$ regularization, and $\Omega(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_i |w_i|$ for $\mathcal{L}_1$ regularization, and $n = C_{out} \cdot H_{out} \cdot W_{out}$ is the number of pixels in all channels of one output image. $\mathbf{w}$ denotes all the parameters in the network. For our case, it includes the kernel weights in all the convolution and transposed convolution layers (no bias is used in convolutional kernel), the scale and shift parameters in all the batch normalization layers. See Section 4.3 for an example of the fully convolutional encoder–decoder network used for the Darcy flow problem. Note that $\mathcal{L}_2$ regularization is implemented in PyTorch optimizers by specifying *weight decay*, which is α in Eq. (3).

The network architecture selected for the non-Bayesian model is the same as that used for our Bayesian model introduced next.

Remark 2. In the encoder–decoder network, the batch normalization layer used after each convolutional layer can also be considered as an effective regularizer. It is commonly adopted nowadays in deep convolutional networks replacing dropout.

3. Bayesian neural networks

Consider a deterministic neural net $\mathbf{y} = \mathbf{f}(\mathbf{x}, \mathbf{w})$ with input $\mathbf{x}$, output $\mathbf{y}$, and all parameters $\mathbf{w}$ including the weights and biases.[1] Bayesian neural networks (BNNs) treat the parameters $\mathbf{w}$ as random variables instead of deterministic unknowns to account for epistemic uncertainty induced by lack of training data. Besides that, usually additive noise $\mathbf{n}$ is introduced to model the aleatoric uncertainty which can not be reduced by having more observations, also to make the probabilistic model have an explicit likelihood depending on the noise distribution, i.e.

$$\mathbf{y} = \mathbf{f}(\mathbf{x}, \mathbf{w}) + \mathbf{n}, \tag{4}$$

where $\mathbf{f}(\mathbf{x}, \mathbf{w})$ is the output of a neural network with the uncertain $\mathbf{w}$, and $\mathbf{n}$ is the additive noise.

3.1. Sparsity inducing prior on weights

Given the large amount of 'un-interpretable' parameters $\mathbf{w}$ in a deep neural net, there are not many choices of priors. But the demand for compression [52,34] of the neural net for lower memory and computation cost calls for sparsity promoting priors [53]. We assume a fully factorized Gaussian prior with zero mean and Gamma-distributed precision α on parameters $\mathbf{w}$

[1] http://pytorch.org/docs/master/nn.html?ht=conv2d#torch.nn.functional.conv2d.

$$p(\mathbf{w} \mid \alpha) = \mathcal{N}(\mathbf{w} \mid 0, \alpha^{-1}\mathbf{I}), \qquad p(\alpha) = \text{Gamma}(\alpha \mid a_0, b_0). \tag{5}$$

This results in a student's t-prior for $\mathbf{w}$, which has heavy tails and more mass close to zero.

Remark 3. The student's t-prior is introduced here for regularization rather than pure compression purposes. Nevertheless, the sparsity of the trained Bayesian neural net was investigated by pruning out the weights (more precisely of the realizations of the uncertain weights) under a small threshold. It was found that the test regression performance was reduced by 1% with 15% of the weights pruned out. Appendix B provides more details.

3.2. Additive noise model

Additive noise can be considered of the following form:

- Output-wise: $\mathbf{n} = \sigma\boldsymbol{\epsilon}$, same for all output pixels;
- Channel-wise: $\mathbf{n} = [\sigma_1\boldsymbol{\epsilon}_1, \cdots, \sigma_{d_y}\boldsymbol{\epsilon}_{d_y}]$, same across each of the d_y output channels/fields;
- Pixel-wise: $\mathbf{n} = \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$, distinct for each output pixel.

Here, σ, $\{\sigma_i\}_{i=1}^{d_y}$ are scalars, $\boldsymbol{\sigma}$ is a field with the same dimension as the output $\mathbf{y}$ and $\odot$ denotes the element-wise product operator. In this work, we have considered both Gaussian noise, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and Laplacian noise, $\boldsymbol{\epsilon} \sim \text{Laplace}(\mathbf{0}, \mathbf{I})$.[2] In the numerical results discussed in Section 4, we concentrate in the output-wise and channel-wise cases above. We treat the noise precision $\beta = 1/\sigma^2$ as a random variable with a conjugate prior $p(\beta) = \text{Gamma}(\beta \mid a_1, b_1)$. For the generated training data, *a priori* we assume that the noise variance (also known as nugget [54]) to be very small e.g. 10^{-6}. Thus the values $a_1 = 2, b_1 = 2 \cdot 10^{-6}$ provide a good initial guess for the prior hyperparameters.

Remark 4. We can also model the noise varying with input (pixel-wise noise model), resulting in a *heteroscedastic* noise model, i.e. $\mathbf{n}(\mathbf{x}, \mathbf{w}) = \sigma(\mathbf{x}, \mathbf{w})\boldsymbol{\epsilon}$ or $\mathbf{n}(\mathbf{x}, \mathbf{w}) = \boldsymbol{\sigma}(\mathbf{x}, \mathbf{w}) \odot \boldsymbol{\epsilon}$. Again $\boldsymbol{\epsilon}$ can be Gaussian or Laplacian. The heteroscedastic noise [55–57] can be implemented as extending the output of the neural net as:

$$[\mathbf{f}(\mathbf{x}, \mathbf{w}), \sigma^2(\mathbf{x}, \mathbf{w})] \text{ or } [\mathbf{f}(\mathbf{x}, \mathbf{w}), \boldsymbol{\sigma}^2(\mathbf{x}, \mathbf{w})]. \tag{6}$$

The output of the system becomes $\mathbf{y} = \mathbf{f}(\mathbf{x}, \mathbf{w}) + \mathbf{n}(\mathbf{x}, \mathbf{w})$. The pixel-wise case $\boldsymbol{\sigma}^2(\mathbf{x}, \mathbf{w})$ may help capture large variations for example near discontinuous regions of the output. In practice, we apply a *softplus* transformation to the second part of the output of the neural net to enforce the positive variance constraint, i.e. $\sigma^2 = \log(1 + \exp(\cdot)) + \text{eps}$, where $\text{eps} = 10^{-10}$ for numerical stability.

3.3. Stein Variational Gradient Descent (SVGD)

Approximate inference for Bayesian deep neural network is a daunting task because of the large number of uncertain parameters, e.g. tens or hundreds of millions in modern deep networks. In our surrogate problem, the task is to find a high-dimensional posterior distribution over millions of random variables using less than hundreds or thousands of training data. As reviewed in Section 1, most of variational inference methods [58] restrict the approximate posterior within certain parametric variational family, while sampling-based methods are slow and difficult to converge. Here we adopt a recently proposed non-parametric variational inference method called stochastic variational gradient descent (SVGD) [30,59] that is similar to standard gradient descent while maintaining the efficiency of particle methods.

For a *prescribed* probabilistic model with *likelihood* function $p(\mathbf{y} \mid \boldsymbol{\theta}, \mathbf{x})$ and prior $p_0(\boldsymbol{\theta})$, we are interested in Bayesian inference of the uncertain parameters $\boldsymbol{\theta}$, i.e. to find the posterior distribution $p(\boldsymbol{\theta} \mid \mathcal{D})$, where $\mathcal{D}$ denote the i.i.d. observations (training data), i.e. $\mathcal{D} = \{(\mathbf{x}^i, \mathbf{y}^i)\}_{i=1}^N$. For the BNNs with homoescedastic Gaussian noise case, $\boldsymbol{\theta} = \{\mathbf{w}, \beta\}$. Variational inference aims to approximate the target posterior distribution $p(\boldsymbol{\theta} \mid \mathcal{D})$ with a variational distribution $q^*(\boldsymbol{\theta})$ which lies in a restricted set of distributions $\mathcal{Q}$ by minimizing the KL divergence between the two, i.e.

$$q^*(\boldsymbol{\theta}) = \arg\min_{q \in \mathcal{Q}} \mathcal{KL}(q(\boldsymbol{\theta}) \parallel p(\boldsymbol{\theta} \mid \mathcal{D})) = \arg\min_{q \in \mathcal{Q}} \mathbb{E}_q[\log q(\boldsymbol{\theta}) - \log \tilde{p}(\boldsymbol{\theta} \mid \mathcal{D}) + \log Z],$$

where $\tilde{p}(\boldsymbol{\theta} \mid \mathcal{D}) = p(\mathcal{D} \mid \boldsymbol{\theta}) p_0(\boldsymbol{\theta}) = \prod_{i=1}^N p(\mathbf{y}^i \mid \boldsymbol{\theta}, \mathbf{x}^i) p_0(\boldsymbol{\theta})$ is the unnormalized posterior, and $Z = \int \tilde{p}(\boldsymbol{\theta}) d\boldsymbol{\theta}$ is the normalization constant or model evidence, which is usually computationally intractable, but can be ignored when we optimize the KL divergence.

The variational family considered here is a set of distributions obtained by smooth transforms from an initial tractable distribution (e.g. the prior) represented in terms of samples. The transforms applied to each particle take the following form:

[2] http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/VELDHUIZEN/node11.html.

$$\mathbf{T}(\boldsymbol{\theta}) = \boldsymbol{\theta} + \epsilon \boldsymbol{\phi}(\boldsymbol{\theta}), \tag{7}$$

where ϵ is the step size, $\boldsymbol{\phi}(\boldsymbol{\theta}) \in \mathcal{F}$ is the perturbation direction within a function space $\mathcal{F}$. When ϵ is small, $\mathbf{T}$ transforms the initial density $q(\boldsymbol{\theta})$ to

$$q_{[\mathbf{T}]}(\boldsymbol{\theta}) = q(\mathbf{T}^{-1}(\boldsymbol{\theta}))|\det(\nabla \mathbf{T}^{-1}(\boldsymbol{\theta}))|.$$

Instead of using parametric form for the variational posterior, a particle approximation is used, i.e. a set of samples $\{\boldsymbol{\theta}^i\}_{i=1}^S$ with empirical measure $\mu_S(d\boldsymbol{\theta}) = \frac{1}{S}\sum_{i=1}^S \boldsymbol{\delta}(\boldsymbol{\theta} - \boldsymbol{\theta}^i)d\boldsymbol{\theta}$. We would like to have μ to weakly converge to the measure of the true posterior $\nu_p(d\boldsymbol{\theta}) = p(\boldsymbol{\theta})d\boldsymbol{\theta}$. We apply the transform $\mathbf{T}$ to those samples, and denote the pushforward measure of μ as $\mathbf{T}\mu$.

The problem is to find out the direction to maximally decrease the KL divergence of the variational approximation and the target distribution, i.e. to solve the following functional optimization problem:

$$\max_{\boldsymbol{\phi}\in\mathcal{F}} \Big\{ -\frac{d}{d\epsilon}\mathcal{KL}(\mathbf{T}\mu \parallel \nu_p)|_{\epsilon=0} \Big\}. \tag{8}$$

It turns out that [30]

$$-\frac{d}{d\epsilon}\mathcal{KL}(\mathbf{T}\mu \parallel \nu_p)|_{\epsilon=0} = \mathbb{E}_\mu[\mathcal{T}_p\boldsymbol{\phi}], \tag{9}$$

where $\mathcal{T}_p$ is called the Stein operator associated to the distribution p,

$$\mathcal{T}_p\boldsymbol{\phi} = \frac{\nabla\cdot(p\boldsymbol{\phi})}{p} = \frac{(\nabla p)\cdot\boldsymbol{\phi} + p(\nabla\cdot\boldsymbol{\phi})}{p} = (\nabla \log p)\cdot\boldsymbol{\phi} + \nabla\cdot\boldsymbol{\phi}.$$

The expectation $\mathbb{E}_\mu[\mathcal{T}_p\boldsymbol{\phi}]$ evaluates the difference between p and μ, and its maximum is defined as the Stein discrepancy,

$$\mathcal{S}(\mu, p) = \max_{\boldsymbol{\phi}\in\mathcal{F}} \mathbb{E}_\mu[\mathcal{T}_p\boldsymbol{\phi}]. \tag{10}$$

It has been shown [60] that when the functional space $\mathcal{F}$ is chosen to be the unit ball in a product reproducing kernel Hilbert space $\mathcal{H}$ with the positive kernel $k(\mathbf{x}, \mathbf{x}')$, the maximal direction to perturb (or the Stein discrepancy) has a closed-form solution,

$$\boldsymbol{\phi}^*(\boldsymbol{\theta}) \propto \mathbb{E}_{\boldsymbol{\theta}'\sim\mu}[\mathcal{T}_p^{\boldsymbol{\theta}'}k(\boldsymbol{\theta}, \boldsymbol{\theta}')] = \mathbb{E}_{\boldsymbol{\theta}'\sim\mu}[\nabla_{\boldsymbol{\theta}'}\log p(\boldsymbol{\theta}')k(\boldsymbol{\theta}, \boldsymbol{\theta}') + \nabla_{\boldsymbol{\theta}'}k(\boldsymbol{\theta}, \boldsymbol{\theta}')].$$

Thus we have the following algorithm to transform an initial distribution μ_0 to the target posterior ν_p [30].

Algorithm 1: Bayesian inference by Stein Variational Gradient Descent.

Input: A set of initial samples $\{\boldsymbol{\theta}_0^i\}_{i=1}^S$, score function $\nabla \log p(\boldsymbol{\theta})$, kernel $k(\boldsymbol{\theta}, \boldsymbol{\theta}')$, step-size scheme $\{\epsilon_t\}$
Result: A set of samples $\boldsymbol{\theta}^i$ that approximate the target posterior
for *iteration* t **do**
  $\boldsymbol{\theta}_{t+1}^i \leftarrow \boldsymbol{\theta}_t^i + \epsilon_t\boldsymbol{\phi}(\boldsymbol{\theta}_t^i)$;
  $\boldsymbol{\phi}(\boldsymbol{\theta}_t^i) = \frac{1}{S}\sum_{j=1}^S \Big[k(\boldsymbol{\theta}_t^j, \boldsymbol{\theta}_t^i)\nabla_{\boldsymbol{\theta}_t^j}\log p(\boldsymbol{\theta}_t^j) + \nabla_{\boldsymbol{\theta}_t^j}k(\boldsymbol{\theta}_t^j, \boldsymbol{\theta}_t^i)\Big]$
end

This is an online algorithm, where the gradient $\boldsymbol{\phi}(\boldsymbol{\theta})$ pushes the samples towards the high posterior region by kernel smoothed gradient term $k(\cdot,\cdot)\nabla\log p$, while maintaining a degree of diversity by repulsive force term $\nabla k(\boldsymbol{\theta}, \boldsymbol{\theta}')$. When the number of samples becomes 1, then the algorithm reduces to the MAP estimate of the posterior. This algorithm is implemented in PyTorch with GPU acceleration and scaled to our deep convolutional encoder–decoder network `DenseED`. The number of samples S is subject to the memory constraint of GPUs. However it is shown later in the numerical experiments that there is only diminishing return when the number of samples increases.

Here we use one toy example to illustrate the idea of SVGD. We start with the 20 samples from the Normal distribution $\mathcal{N}(-10, 1)$, and transport the samples iteratively to the target Gaussian mixture distribution $0.8\mathcal{N}(-2, 1) + 0.2\mathcal{N}(2, 1)$ with Algorithm 1 (see also Fig. 4).

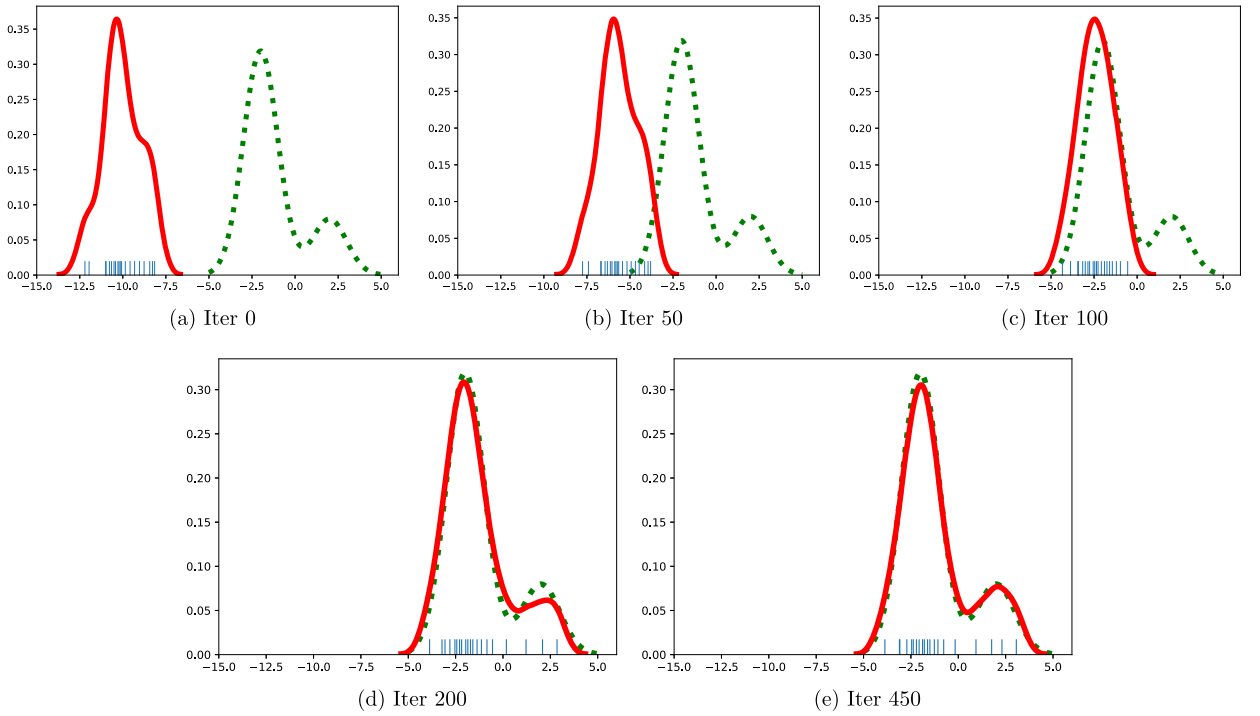


Fig. 4. Example of transporting 20 samples from $\mathcal{N}(-10, 1)$ to the Gaussian mixture $0.8\mathcal{N}(-2, 1) + 0.2\mathcal{N}(2, 1)$. The green dash line represents the target Gaussian mixture. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

3.3.1. Implementation and training

- We use S samples of $\boldsymbol{\theta}$ to approximate the empirical measure of the posterior. They are initialized and stored in 20 deterministic DenseED neural networks.
- At each step t, for each model j, the gradient of its joint likelihood (or unnormalized posterior) $\nabla_{\boldsymbol{\theta}_t^j} \log p(\boldsymbol{\theta}_t^j)$ is computed by the automatic differentiation tool in PyTorch. We first compute the joint likelihood $\log p(\boldsymbol{\theta}_t^j) = \prod_{i=1}^N p(\mathbf{y}_i \mid \boldsymbol{\theta}_t^j, \mathbf{x}_i) p(\boldsymbol{\theta}_t^j)$ by feeding forward the data $\{(\mathbf{x}^i, \mathbf{y}^i)\}_{i=1}^N$, then back propagate to compute its gradient $\nabla_{\boldsymbol{\theta}_t^j} \log p(\boldsymbol{\theta}_t^j)$. Note that this gradient is stored in PyTorch module associated to the weights $\boldsymbol{\theta}_t^j$ in each network.
- Then we can proceed to compute the kernel matrix $\left[k(\boldsymbol{\theta}_t^j, \boldsymbol{\theta}_t^i)\right]_{i,j\in\{1,\cdots,S\}}$, and its gradient $\nabla_{\boldsymbol{\theta}_t^j} k(\boldsymbol{\theta}_t^j, \boldsymbol{\theta}_t^i)$, as well as the kernel weighted gradient of the joint likelihood $k(\boldsymbol{\theta}_t^j, \boldsymbol{\theta}_t^i) \nabla_{\boldsymbol{\theta}_t^j} \log p(\boldsymbol{\theta}_t^j)$. For this to happen, we need to vectorize (extract out) the parameters $\boldsymbol{\theta}_t^j$ and the computed gradient $\nabla_{\boldsymbol{\theta}_t^j} \log p(\boldsymbol{\theta}_t^j)$ from each neural network. The optimal perturbation direction $\boldsymbol{\phi}$ is further computed by the sum of the two terms as shown in the SVGD algorithm.
- After $\{\boldsymbol{\phi}(\boldsymbol{\theta}_t^j)\}_{j=1}^S$ is computed, we send $\boldsymbol{\phi}(\boldsymbol{\theta}_t^j)$ back to each neural network the gradient w.r.t. its parameters $\boldsymbol{\theta}_t^j$ (overwriting the previously computed $\nabla_{\boldsymbol{\theta}_t^j} \log p(\boldsymbol{\theta}_t^j)$), and updating locally in each neural network using PyTorch's optimization library such as Adam or SGD to compute $\boldsymbol{\theta}_{t+1}^j$.
- With one iteration complete, the algorithm repeats the above steps until convergence in the parameters is achieved.

3.4. Uncertainty quantification

Of interest to the classical UQ problem is the computation of the posterior predictive distribution (predict the system response for a test input) as well as the computation of the output response averaged over the input probability distribution. In particular, we are interested in computing the following:

- Predictive uncertainty at $\mathbf{x}^*$: $p(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D})$, and in particular the moments $\mathbb{E}[\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D}]$, $\mathrm{Var}(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D})$.
- Propagated uncertainty to the system response by integrating over $p(\mathbf{x})$: $p(\mathbf{y} \mid \boldsymbol{\theta}), \boldsymbol{\theta} \sim p(\boldsymbol{\theta} \mid \mathcal{D})$, and in particular $\mathbb{E}[\mathbf{y} \mid \boldsymbol{\theta}]$, $\mathrm{Var}(\mathbf{y} \mid \boldsymbol{\theta})$. One can use these moments to compute the statistics of conditional output statistics, e.g. $\mathbb{E}_{\boldsymbol{\theta}}\left[\mathbb{E}[\mathbf{y} \mid \boldsymbol{\theta}]\right]$, $\mathrm{Var}_{\boldsymbol{\theta}}\left(\mathbb{E}[\mathbf{y} \mid \boldsymbol{\theta}]\right)$ and $\mathbb{E}_{\boldsymbol{\theta}}\left(\mathrm{Var}(\mathbf{y} \mid \boldsymbol{\theta})\right)$, $\mathrm{Var}_{\boldsymbol{\theta}}\left(\mathrm{Var}(\mathbf{y} \mid \boldsymbol{\theta})\right)$.

We can use Monte Carlo to approximate the moments of the predictive distribution

$$p(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D}) = \int p(\mathbf{y}^* \mid \mathbf{x}^*, \mathbf{w}, \beta) p(\mathbf{w}, \beta \mid \mathcal{D}) d\mathbf{w} d\beta, \tag{11}$$

with mean (by the law of total expectation)

$$\begin{aligned} \mathbb{E}[\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D}] &= \mathbb{E}_{p(\mathbf{w},\beta|\mathcal{D})}\Big[\mathbb{E}[\mathbf{y}^* \mid \mathbf{x}^*, \mathbf{w}, \beta]\Big] \\ &= \mathbb{E}_{p(\mathbf{w}|\mathcal{D})}[\mathbf{f}(\mathbf{x}^*, \mathbf{w})] \\ &\approx \frac{1}{S}\sum_{i=1}^{S} \mathbf{f}(\mathbf{x}^*, \mathbf{w}^i), \qquad \mathbf{w}^i \sim p(\mathbf{w} \mid \mathcal{D}). \end{aligned} \tag{12}$$

Note that the SVGD algorithm provides a sample representation of the joint posterior of all parameters $p(\mathbf{w}, \beta \mid \mathcal{D})$. To obtain the samples of the marginal posterior $p(\mathbf{w} \mid \mathcal{D})$ as needed above, one simply needs to use the samples corresponding to $\mathbf{w}$.

The predictive covariance can also be easily calculated using the law of total variance. The variance and expectation below are w.r.t. to the posterior of the parameters. We can show the following:

$$\begin{aligned} \mathrm{Cov}(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D}) &= \mathbb{E}_{\mathbf{w},\beta}\Big[\mathrm{Cov}(\mathbf{y}^* \mid \mathbf{w}, \beta, \mathbf{x}^*)\Big] + \mathrm{Cov}_{\mathbf{w},\beta}\Big(\mathbb{E}[\mathbf{y}^* \mid \mathbf{w}, \beta, \mathbf{x}^*]\Big) \\ &= \mathbb{E}_{\mathbf{w},\beta}[\beta^{-1}\mathbf{I}] + \mathrm{Cov}_{\mathbf{w},\beta}(\mathbf{f}(\mathbf{x}^*, \mathbf{w})) \\ &= \mathbb{E}_{\beta}[\beta^{-1}\mathbf{I}] + \mathbb{E}_{\mathbf{w}}[\mathbf{f}(\mathbf{x}^*, \mathbf{w})\mathbf{f}^\top(\mathbf{x}^*, \mathbf{w})] - \mathbb{E}_{\mathbf{w}}[\mathbf{f}(\mathbf{x}^*, \mathbf{w})]\,\mathbb{E}_{\mathbf{w}}^\top[\mathbf{f}(\mathbf{x}^*, \mathbf{w})] \\ &\approx \frac{1}{S}\sum_{i=1}^{S}\Big((\beta^i)^{-1}\mathbf{I} + \mathbf{f}(\mathbf{x}^*, \mathbf{w}^i)\mathbf{f}^\top(\mathbf{x}^*, \mathbf{w}^i)\Big) \\ &\quad - \Big(\frac{1}{S}\sum_{i=1}^{S}\mathbf{f}(\mathbf{x}^*, \mathbf{w}^i)\Big)\Big(\frac{1}{S}\sum_{i=1}^{S}\mathbf{f}(\mathbf{x}^*, \mathbf{w}^i)\Big)^\top, \end{aligned} \tag{13}$$

where $\beta^i \sim p(\beta \mid \mathcal{D}), \mathbf{w}^i \sim p(\mathbf{w} \mid \mathcal{D})$. The predictive variance is the diagonal of the predictive covariance:

$$\begin{aligned} \mathrm{Var}(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D}) &= \mathrm{diag}\,\mathrm{Cov}(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D}) \\ &= \frac{1}{S}\sum_{i=1}^{S}\Big((\beta^i)^{-1}\mathbf{1} + \mathbf{f}^2(\mathbf{x}^*, \mathbf{w}^i)\Big) - \Big(\frac{1}{S}\sum_{i=1}^{S}\mathbf{f}(\mathbf{x}^*, \mathbf{w}^i)\Big)^2, \end{aligned} \tag{14}$$

where $\mathbf{1}$ is a vector of ones with the same dimension as $\mathbf{f}$, and the square $(\cdot)^2$ is applied element-wise to the vectors.

The above computation is the prediction at a specific input $\mathbf{x}^*$. We would also like to compute the average prediction over the distribution of the uncertain input. We first compute the output statistics given the realizations of the uncertain parameters $\boldsymbol{\theta} = \{\mathbf{w}, \beta\}$, where $\boldsymbol{\theta} \sim p(\boldsymbol{\theta} \mid \mathcal{D})$. The conditional predictive mean is

$$\mathbb{E}[\mathbf{y} \mid \boldsymbol{\theta}] = \mathbb{E}_{\mathbf{x}}\,\mathbb{E}[\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}] = \mathbb{E}_{\mathbf{x}}[\mathbf{f}(\mathbf{x}, \mathbf{w})] \approx \frac{1}{M}\sum_{j=1}^{M}\mathbf{f}(\mathbf{x}^j, \mathbf{w}), \quad \mathbf{x}^j \sim p(\mathbf{x}), \tag{15}$$

and the conditional predictive covariance is

$$\begin{aligned} \mathrm{Cov}(\mathbf{y} \mid \boldsymbol{\theta}) &= \mathbb{E}_{\mathbf{x}}[\mathrm{Cov}(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta})] + \mathrm{Cov}_{\mathbf{x}}(\mathbb{E}[\mathbf{y} \mid \mathbf{x}, \theta]) \\ &= \mathbb{E}_{\mathbf{x}}[(\beta)^{-1}\mathbf{I}] + \mathrm{Cov}_{\mathbf{x}}(\mathbf{f}(\mathbf{x}, \mathbf{w})) \\ &\approx \beta^{-1}\mathbf{I} + \frac{1}{M}\sum_{j=1}^{M}\mathbf{f}(\mathbf{x}^j, \mathbf{w})\mathbf{f}^\top(\mathbf{x}^j, \mathbf{w}) - \Big(\frac{1}{M}\sum_{j=1}^{M}\mathbf{f}(\mathbf{x}^j, \mathbf{w})\Big)\Big(\frac{1}{M}\sum_{j=1}^{M}\mathbf{f}(\mathbf{x}^j, \mathbf{w})\Big)^\top. \end{aligned} \tag{16}$$

Also the conditional predictive variance (at each spatial location) is

$$\mathrm{Var}(\mathbf{y} \mid \boldsymbol{\theta}) = \mathrm{diag}\,\mathrm{Cov}(\mathbf{y} \mid \boldsymbol{\theta}) = \beta^{-1}\mathbf{1} + \frac{1}{M}\sum_{j=1}^{M}\mathbf{f}^2(\mathbf{x}^j, \mathbf{w}) - \Big(\frac{1}{M}\sum_{j=1}^{M}\mathbf{f}(\mathbf{x}^j, \mathbf{w})\Big)^2, \tag{17}$$

where $\mathbf{1}$ is a vector of ones with the same dimension as $\mathbf{y}$, and the square operator is here applied element-wise to the vectors. Then we can further compute the statistics of the above conditional statistics due to the uncertainty in the surrogate, i.e. $\boldsymbol{\theta}$, such as $\mathbb{E}_{\boldsymbol{\theta}}\Big[\mathbb{E}[\mathbf{y} \mid \boldsymbol{\theta}]\Big]$, $\mathrm{Var}_{\boldsymbol{\theta}}\Big(\mathbb{E}[\mathbf{y} \mid \boldsymbol{\theta}]\Big)$ and $\mathbb{E}_{\boldsymbol{\theta}}\Big(\mathrm{Var}(\mathbf{y} \mid \boldsymbol{\theta})\Big)$, $\mathrm{Var}_{\boldsymbol{\theta}}\Big(\mathrm{Var}(\mathbf{y} \mid \boldsymbol{\theta})\Big)$. These can be approximated with the sample mean and sample variance of the conditional predictive mean and variance, respectively.

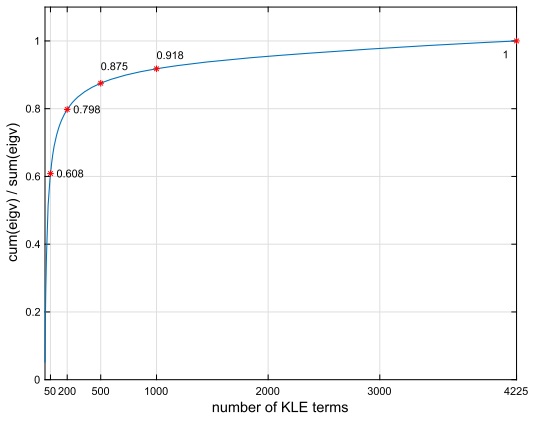


Fig. 5. KLE profile.

4. Numerical implementation and results

We study the two-dimensional, single phase, steady-state flow through a random permeability field following the case study in Section 3.2 in [2]. Consider the random permeability field K on a unit square spatial domain $\mathcal{S} = [0,1]^2$, the pressure field p and velocity field $\mathbf{u}$ of the fluid through the porous media are governed by Darcy's law:

$$\begin{aligned} \mathbf{u}(\mathbf{s}) &= -K(\mathbf{s})\nabla p(\mathbf{s}), \qquad \mathbf{s} \in \mathcal{S}, \\ \nabla \cdot \mathbf{u}(\mathbf{s}) &= f(\mathbf{s}), \qquad \mathbf{s} \in \mathcal{S}, \\ \mathbf{u}(\mathbf{s}) \cdot \hat{\mathbf{n}}(\mathbf{s}) &= 0, \qquad \mathbf{s} \in \partial\mathcal{S}, \\ \int_{\mathcal{S}} p(\mathbf{s}) d\mathbf{s} &= 0, \end{aligned} \tag{18}$$

where $\hat{\mathbf{n}}$ denotes the unit normal vector to the boundary and the source term f is used to model an injection well on the left-bottom corner of $\mathcal{S}$ and a production well on the right-top corner. We also enforce no-flux boundary condition, and an integral constraint to ensure the uniqueness of the solution as in [2]. More specifically,

$$f(\mathbf{s}) = \begin{cases} r, & \text{if } |\mathbf{s}_i - \frac{1}{2}w| \le \frac{1}{2}w, \text{for } i = 1,2, \\ -r, & \text{if } |\mathbf{s}_i - 1 + \frac{1}{2}w| \le \frac{1}{2}w, \text{for } i = 1,2, \\ 0, & \text{otherwise}, \end{cases} \tag{19}$$

where r is the rate of the wells and w is their size. The input log-permeability field is restricted in this work to be a Gaussian random field, i.e.

$$K(\mathbf{s}) = \exp(G(\mathbf{s})), \quad G(\cdot) \sim \mathcal{N}(m, k(\cdot,\cdot)), \tag{20}$$

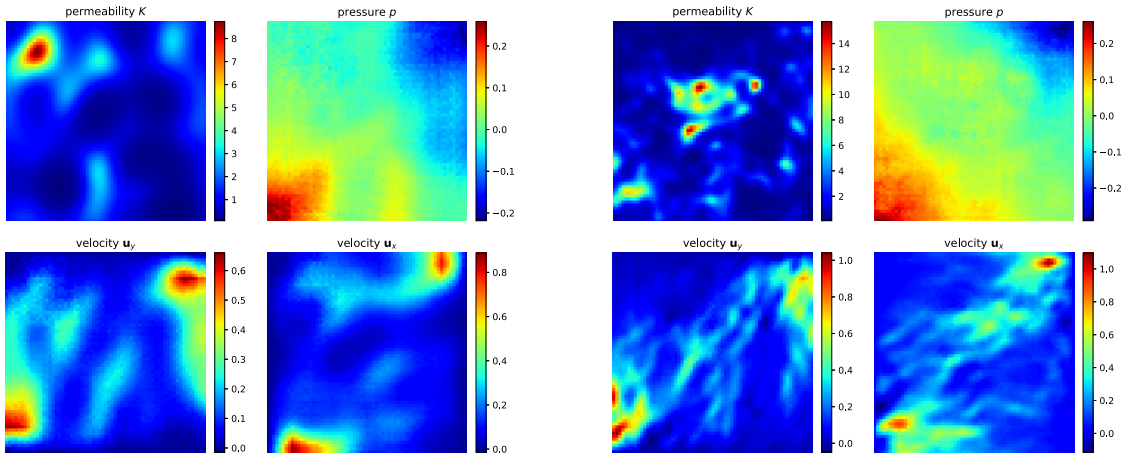
where m is the constant mean and covariance function k is specified in the following form using the L_2 norm in the exponent instead of the L_1 norm in [2], i.e.

$$k(\mathbf{s}, \mathbf{s}') = \exp(-\left\|\mathbf{s} - \mathbf{s}'\right\|_2 / l). \tag{21}$$

4.1. Datasets

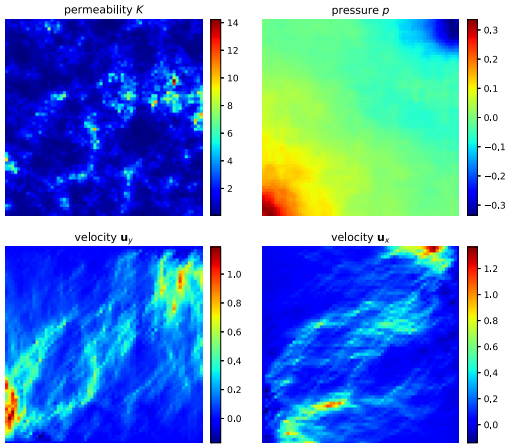
When the length scale of the Gaussian random field is small, the discretized field realizations vary highly even across adjacent grid points [61]. This high variability creates a significant challenge for data-driven models to capture. For this case, the intrinsic dimensionality of the discretized random field is the total number of pixels, e.g. 4225 for 65×65 grids (which is taken as the reference grid for our calculations). To evaluate the generality and effectiveness of the methodology, we use KLE to control the intrinsic dimensionality of the permeability dataset. We evaluated our model using datasets produced with increasing dimensionality of 50, 500, 4225 (called KLE50, KLE500, and KLE4225, respectively). Notice that for KLE4225, the permeability field is directly sampled from the Gaussian random field. The intrinsic dimensionality of the input dataset is hidden from our model that develops an end-to-end mapping from the input fields to the output fields. In fact, we will present one specific network architecture that works well for all three datasets.

We consider solving the Darcy flow Eq. (18) over a unit squared domain $\mathcal{S} = [0,1]^2$ with fixed 65×65 grid, and length scale $l = 0.1$, kernel mean $m = 0$, rate of source $r = 10$, size of source $w = 0.125$. The ratio of the cumulative sum of



(a) KLE50

(b) KLE500



(c) KLE4225

Fig. 6. Sample permeability K obtained from the (a) KLE50 dataset, (b) KLE500 dataset, and (c) KLE4225 dataset (no dimensionality reduction) and the corresponding velocity components $\mathbf{u}_x$, $\mathbf{u}_y$ and pressure p obtained from the simulator. All figures are shown using pixels (`imshow`) to reveal the high variability of the input and output fields.

eigenvalues (in decreasing order) over the total sum of them is shown in Fig. 5. The Darcy flow equation is solved using a mixed finite element formulation implemented in FEniCS [62] with third-order Raviart–Thomas elements for the velocity, and fourth-order discontinuous elements for the pressure. The sample input permeability field and computed output pressure and velocity fields for the three datasets are shown in Fig. 6.

Our current data includes four sets: the training, validation, test, and uncertainty propagation sets. The training set is sampled using Latin hypercube sampling. More specifically, the KLE for the log-permeability field is

$$G(\mathbf{s}) = m + \sum_{k=1}^{q} \sqrt{\lambda_k} z_k \phi_k(\mathbf{s}), \tag{22}$$

where λ_k and $\phi_k(\mathbf{s})$ are the eigenvalues and eigenfunctions of the exponential covariance function of the Gaussian field specified in Eqs. (20) and (21), z_k's are i.i.d. standard normal, and q is the number of KLE coefficients maintained in the expansion. The maximum number that can be used is finite and equal to the number of grid points used in the discretization of the field over the unit square. We first use Latin hypercube design to sample ξ_k from the hypercube $[0,1]^q$, then obtain the eigenvalue by $z_k = \Phi^{-1}(\xi_k)$, where Φ is the cumulative distribution function of the standard normal distribution. The KLE50 case contains 32, 64, 128, and 256 training data; KLE500 contains 64, 128, 256, and 512 training data; and KLE4225 contains 128, 256, 512, and 1024 training data.

The log-permeability fields in the other three sets are reconstructed directly from z_k which are sampled from the standard normal. The validation and test sets each contain 500 input permeability fields, and the dataset for uncertainty propagation contains 10,000 realizations. All datasets are organized in the form of images as discussed in Section 2.1.

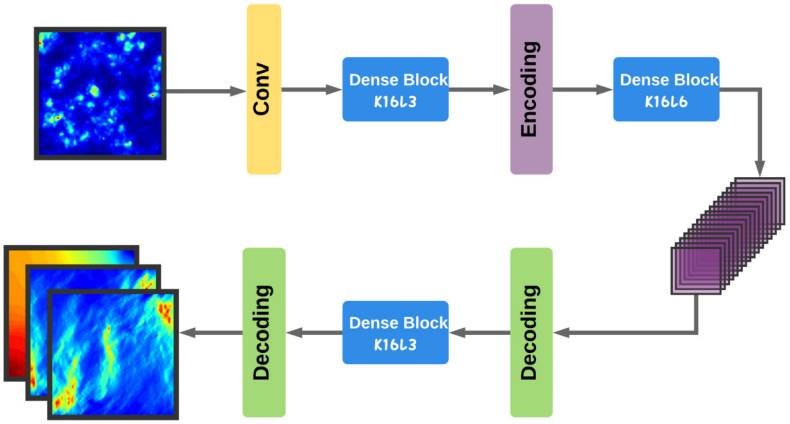


Fig. 7. `DenseED-c16` with blocks $(3, 6, 3)$, growth rate 16, and 48 initial feature maps after the first convolution layer (yellow in the figure). There are in total 19 (conv) layers and 241,164 parameters in the network. The number of network parameters is optimized based on the generalization error analysis reported in Appendix C. It contains two down-sampling layers, thus the smallest spatial dimension (code dimension) of feature maps (purple in the figure) is 16×16, hence the network is named `DenseED-c16`. The first convolution kernel is of $k7s2p2$. The last transposed conv kernel in the decoding layer is $k5s2p1$. The number of its output feature maps is 3 (corresponding to 3 output fields). For the decoding layers, the output padding is set to 1. The other conv kernels in dense blocks and encoding/decoding layers are described in Section 2.2.

Table 1
`DenseED-c16` architecture for the Darcy flow dataset.

Layers	C_f	Resolution $H_f \times W_f$	Number of parameters
Input	1	65×65	–
Convolution `k7s2p2`	48	32×32	2352
Dense Block (1) `K16L3`	96	32×32	28032
Encoding Layer	48	16×16	25632
Dense Block (2) `K16L6`	144	16×16	77088
Decoding Layer (1)	72	32×32	57456
Dense Block (3) `K16L3`	120	32×32	38544
Decoding Layer (2)	3	65×65	12060

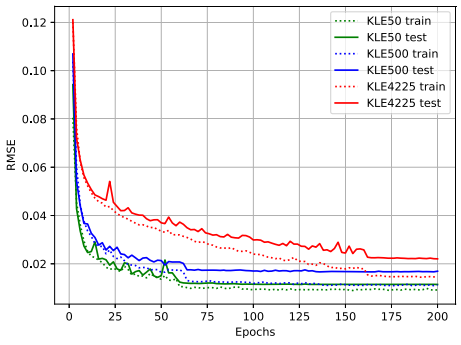


Fig. 8. Training process of `DenseED-c16` with 128 training data.

4.2. Evaluation metrics

Several metrics are used to evaluate the trained models on test data $\{\mathbf{x}^i, \mathbf{y}^i\}_{i=1}^T$. In particular, we consider the following:

Coefficient of determination (R^2-score):

$$R^2 = 1 - \frac{\sum_{i=1}^T \left\| \mathbf{y}^i - \hat{\mathbf{y}}^i \right\|_2^2}{\sum_{i=1}^T \left\| \mathbf{y}^i - \bar{\mathbf{y}} \right\|_2^2}, \tag{23}$$

where $\hat{\mathbf{y}}^i$ is the output mean of the Bayesian surrogate, i.e. $\sum_{i=1}^S \mathbf{f}(\mathbf{x}, \mathbf{w}^i)/S$ as in Eq. (12) or the output $\mathbf{f}(\mathbf{x})$ of the non-Bayesian surrogate, $\mathbf{y}^i$ is the test target, $\bar{\mathbf{y}}$ is the mean of the test target, and T is the total number of test data. This metric enables the comparison between different datasets since the error is normalized, with the score closer to 1 correspond-

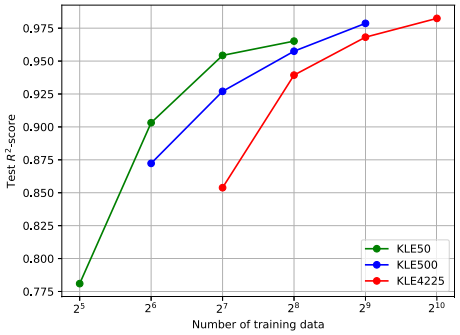
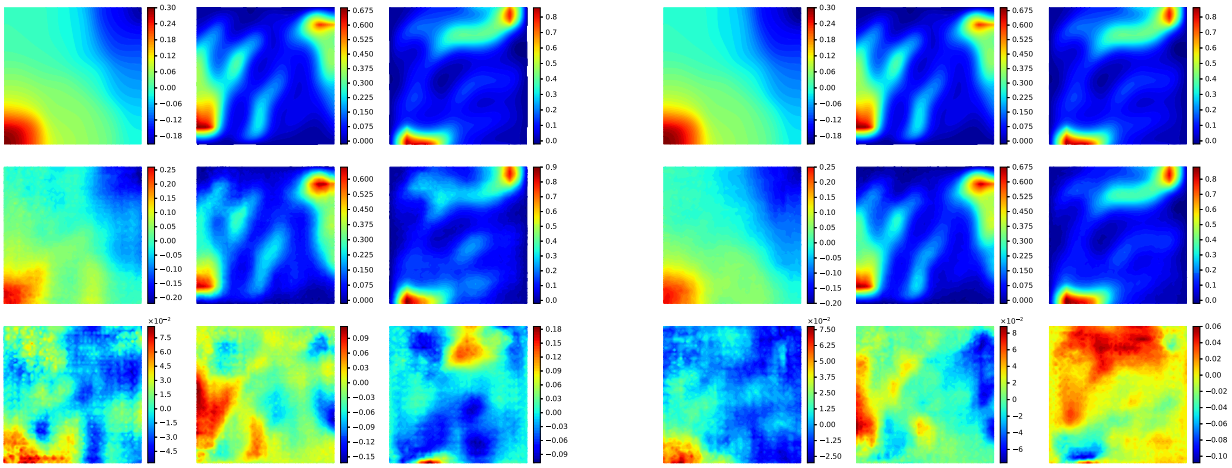
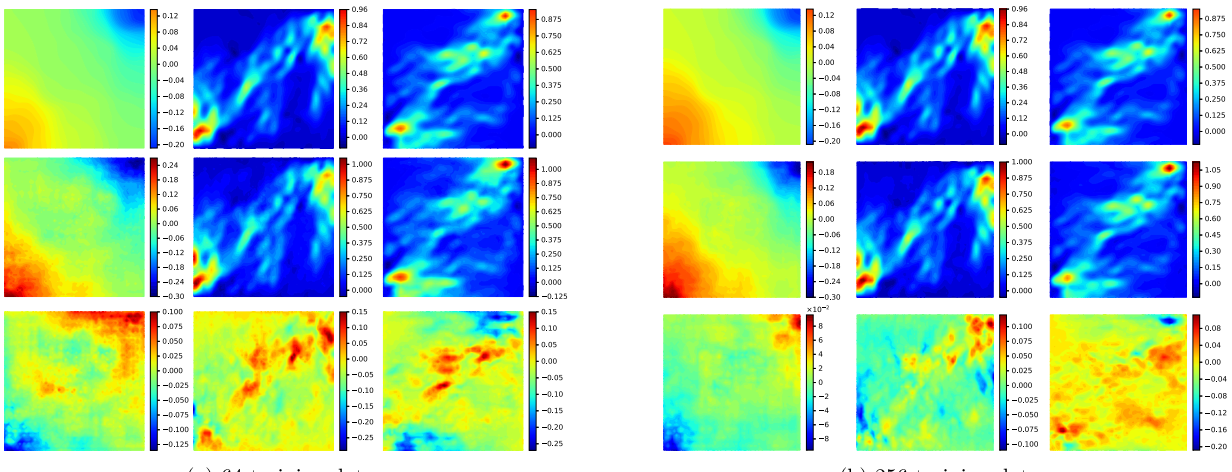


Fig. 9. Test R^2 scores for the non-Bayesian surrogate.



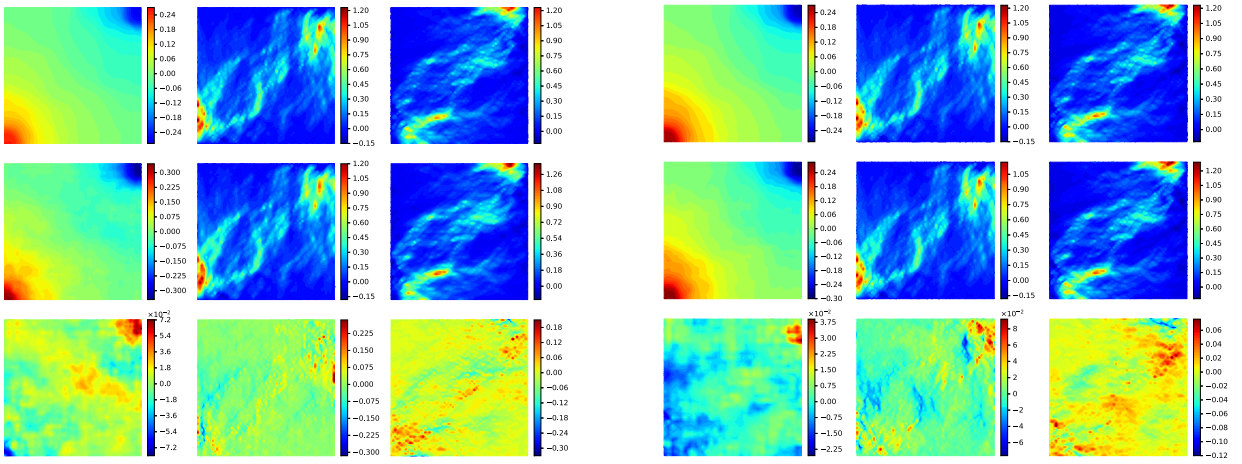
(a) 32 training data (b) 128 training data

Fig. 10. Prediction for the input realization shown in Fig. 6a from the KLE50 dataset using `DenseED-c16` which is trained with datasets of sizes (a) 32 and (b) 128, respectively. In both subfigures, the first row shows the three test target fields (simulation output), i.e. pressure p and velocity $\mathbf{u}_y$, $\mathbf{u}_x$, the second row shows the corresponding model predictions, the third row shows the error.



(a) 64 training data (b) 256 training data

Fig. 11. Prediction for the input realization as shown in Fig. 6b from KLE500 dataset using `DenseED-c16` which is trained with datasets of sizes (a) 64 and (b) 256, respectively. In both subfigures, the first row shows the three test target fields (simulation output), i.e. pressure p and velocity $\mathbf{u}_y$, $\mathbf{u}_x$, the second row shows the corresponding model predictions, the third row shows the error.



(a) 128 training data (b) 512 training data

Fig. 12. Prediction for the input realization as shown in Fig. 6c from KLE4225 dataset using `DenseED-c16` which is trained with datasets of sizes (a) 128 and (b) 512, respectively. In both subfigures, the first row shows the three test target fields (simulation output), i.e. pressure p and velocity $\mathbf{u}_y$, $\mathbf{u}_x$, the second row shows the corresponding model predictions, the third row shows the error.

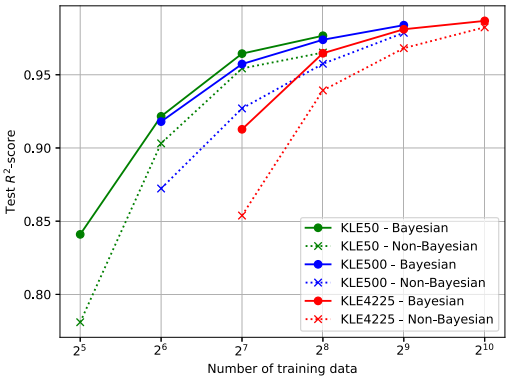


Fig. 13. R^2-scores comparison for the non-Bayesian and Bayesian models.

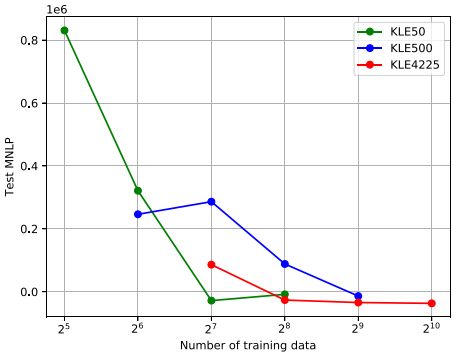
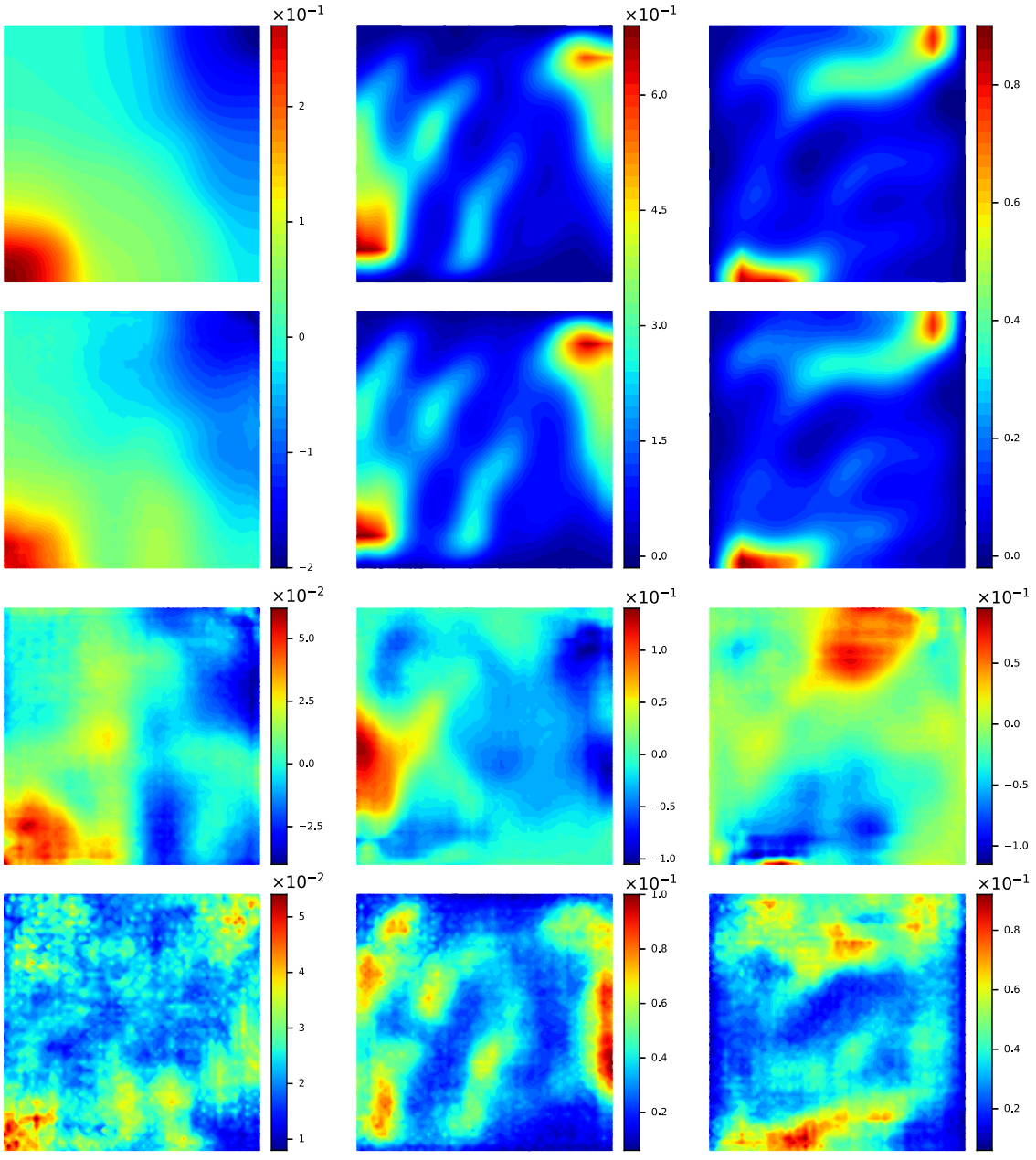


Fig. 14. MNLP of test data under the trained Bayesian surrogate.



(a) 64 training data

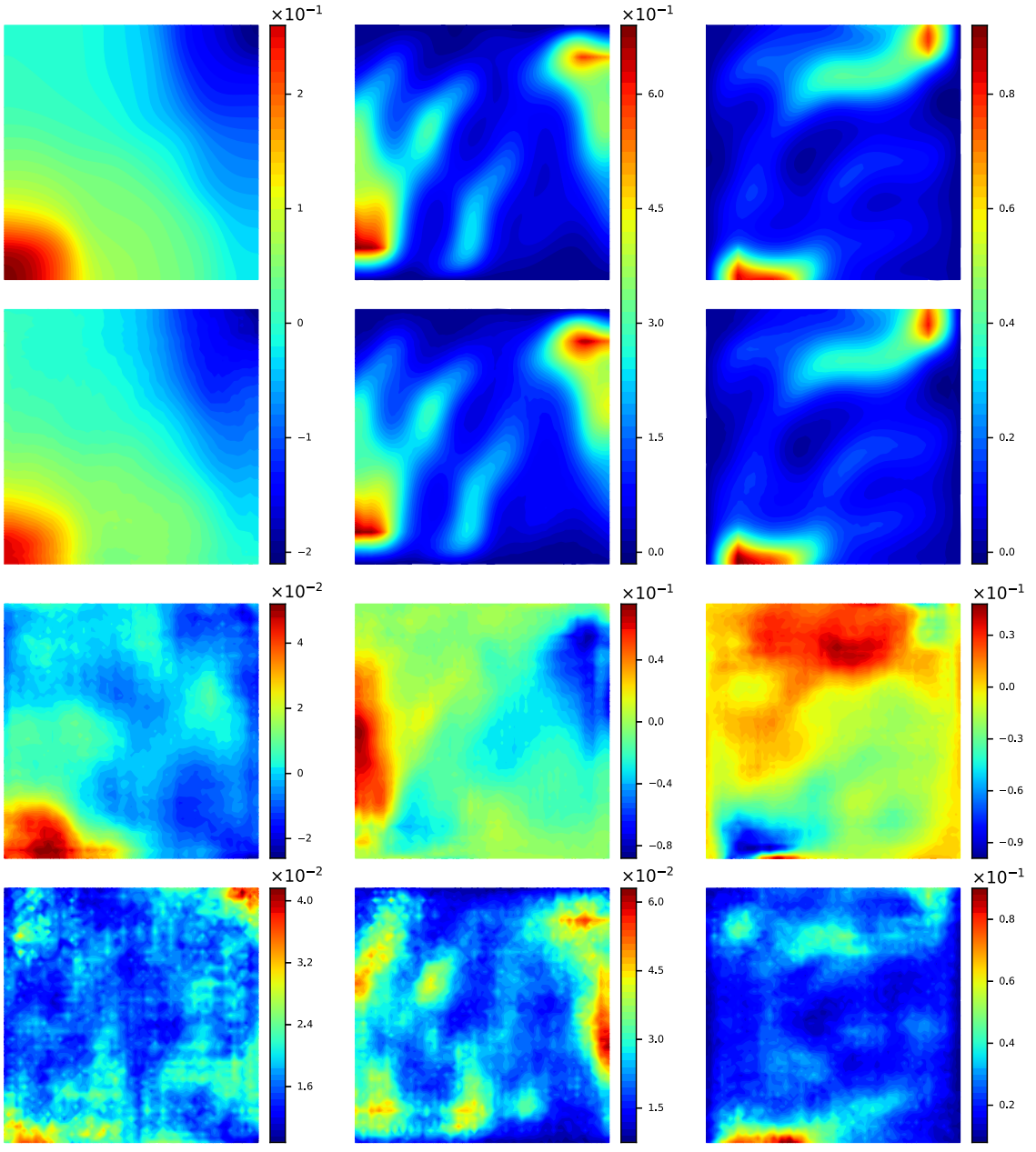
Fig. 15. Prediction for the input realization shown in Fig. 6a from the KLE50 dataset. The rows from top to bottom show the simulation output fields (ground truth), the predictive mean $\mathbb{E}[\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D}]$, the error of the above two, and two standard deviation of the predictive output distribution per pixel $\mathrm{Var}(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D})$. The three columns from left to right correspond to the pressure field p, and the two velocity fields $\mathbf{u}_y$, $\mathbf{u}_x$, respectively.

ing to better regression. For the non-Bayesian surrogate, the R^2-score is the only evaluation metric used. For the Bayesian surrogate, we introduce additional metrics given below.

Root Mean Squared Error (RMSE):

$$\sqrt{\frac{1}{T}\sum_{i=1}^{T}\left\|\hat{\mathbf{y}}^i - \mathbf{y}^i\right\|_2^2}.$$

This is a common metric for regression that is used in our experiments for monitoring the convergence of training.



(b) 256 training data

Fig. 15. (*continued*)

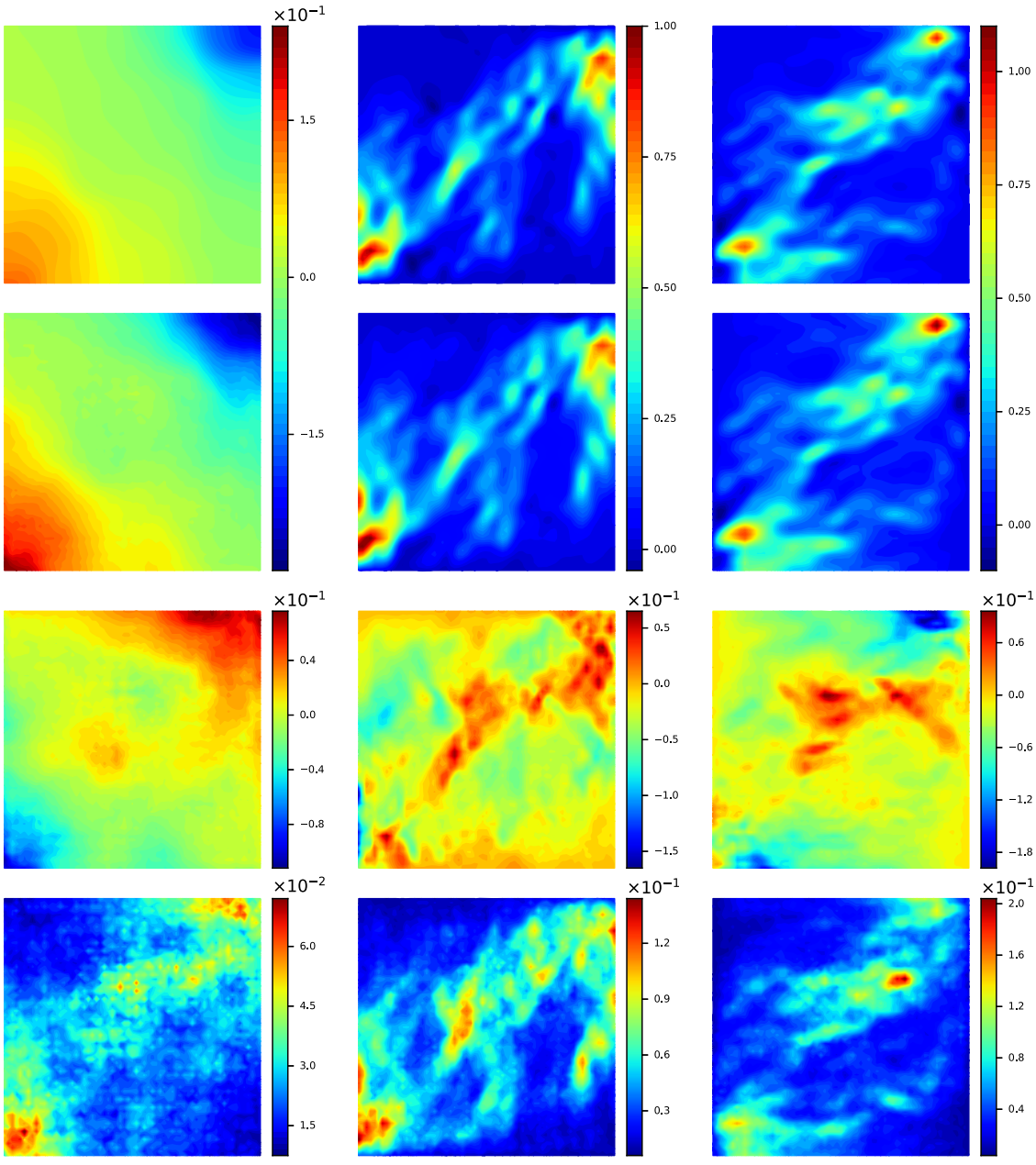
Mean Negative Log-Probability (MNLP):

$$\text{MNLP} = -\frac{1}{T}\sum_{i=1}^{T} \log p(\mathbf{y}^i \mid \mathbf{x}^i, \mathcal{D}).$$

This metric evaluates the likelihood of the observed data. It is used to assess the quality of the predictive model. To compute the MNLP, we estimate $p(\mathbf{y}|\mathbf{x}, \mathcal{D})$ by $p(\mathbf{y} \mid \mathbf{x}, \mathcal{D}) \frac{1}{S} \sum_{i=1}^{S} p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta}^i, \mathcal{D})$, where θ_i are the trained network parameters (i.e. $S = 20$ samples from the posterior). The log-sum-exp trick is used to complete this calculation.

Predictive uncertainty and Propagated Uncertainty: These metrics were introduced in Section 3.4.

Estimated Distributions: They include histograms or kernel density estimates for the output fields at certain locations of the physical domain.



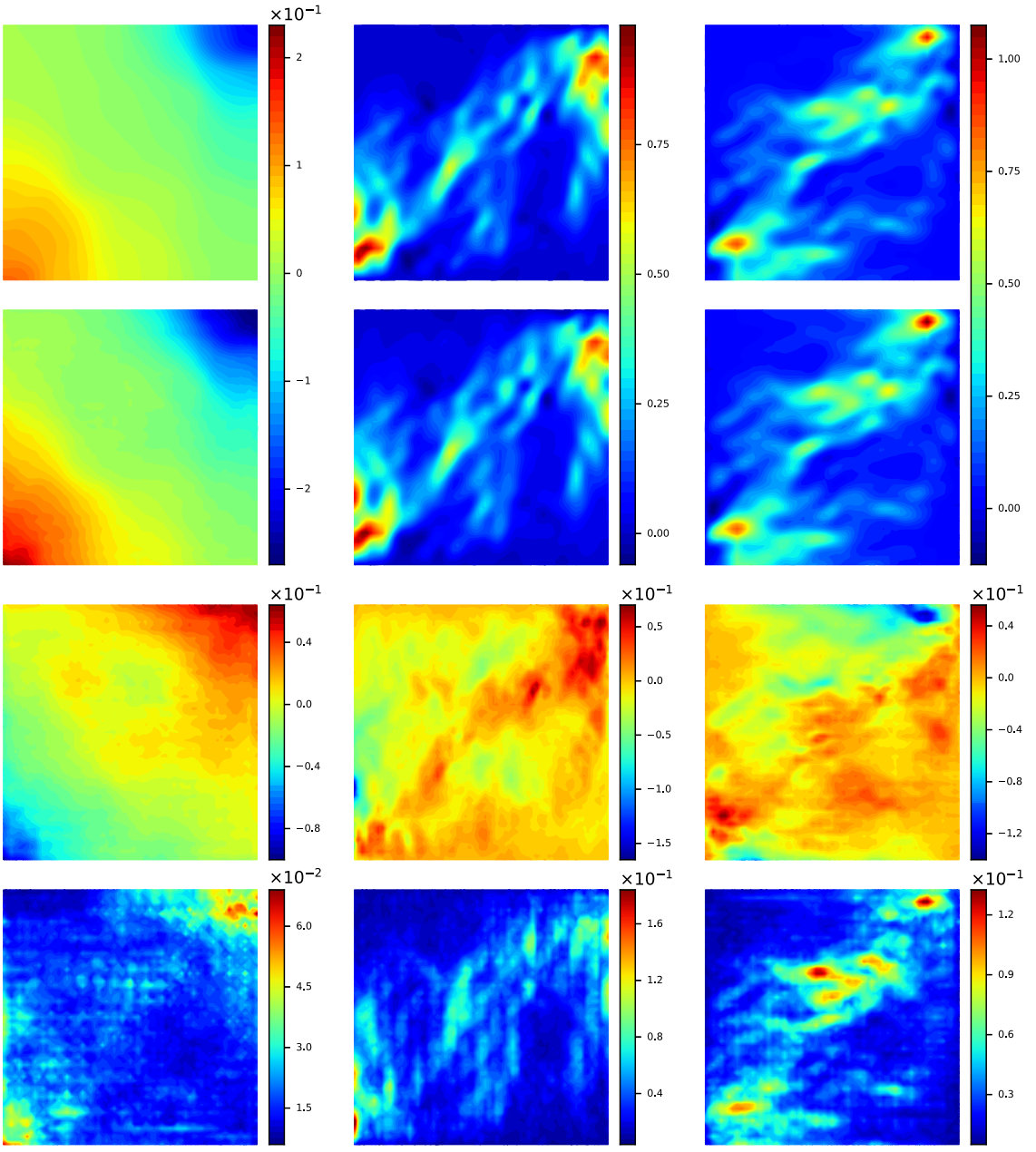
(a) 64 training data

Fig. 16. Prediction for the input realization shown in Fig. 6b from the KLE500 dataset. The rows from top to bottom show the simulation output fields (ground truth), the predictive mean $\mathbb{E}[\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D}]$, the error of the above two, and two standard deviation of the predictive output distribution per pixel $\mathrm{Var}(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D})$. The three columns from left to right correspond to the pressure field p, and the two velocity fields $\mathbf{u}_y$, $\mathbf{u}_x$, respectively.

Reliability Diagram: Given a trained Bayesian surrogate and a test data set, we can compute the $p\%$ predictive interval for each test data point based on the Gaussian quantiles using the predictive mean and variance [63]. We then compute the frequency of the test targets that fall within this predictive interval. For a well-calibrated regression model, the observed frequency should be close to $p\%$. The reliability diagram is the plot of the observed frequency with respect to p. Thus a well-calibrated model should have a reliability diagram close to the diagonal.

4.3. Non-Bayesian surrogate model

The hyperparameters to search include the parameters that determine the network architecture and the ones that specify training process, which both affect model performance. We use the Hyperband [64] algorithm to optimize those hyper-

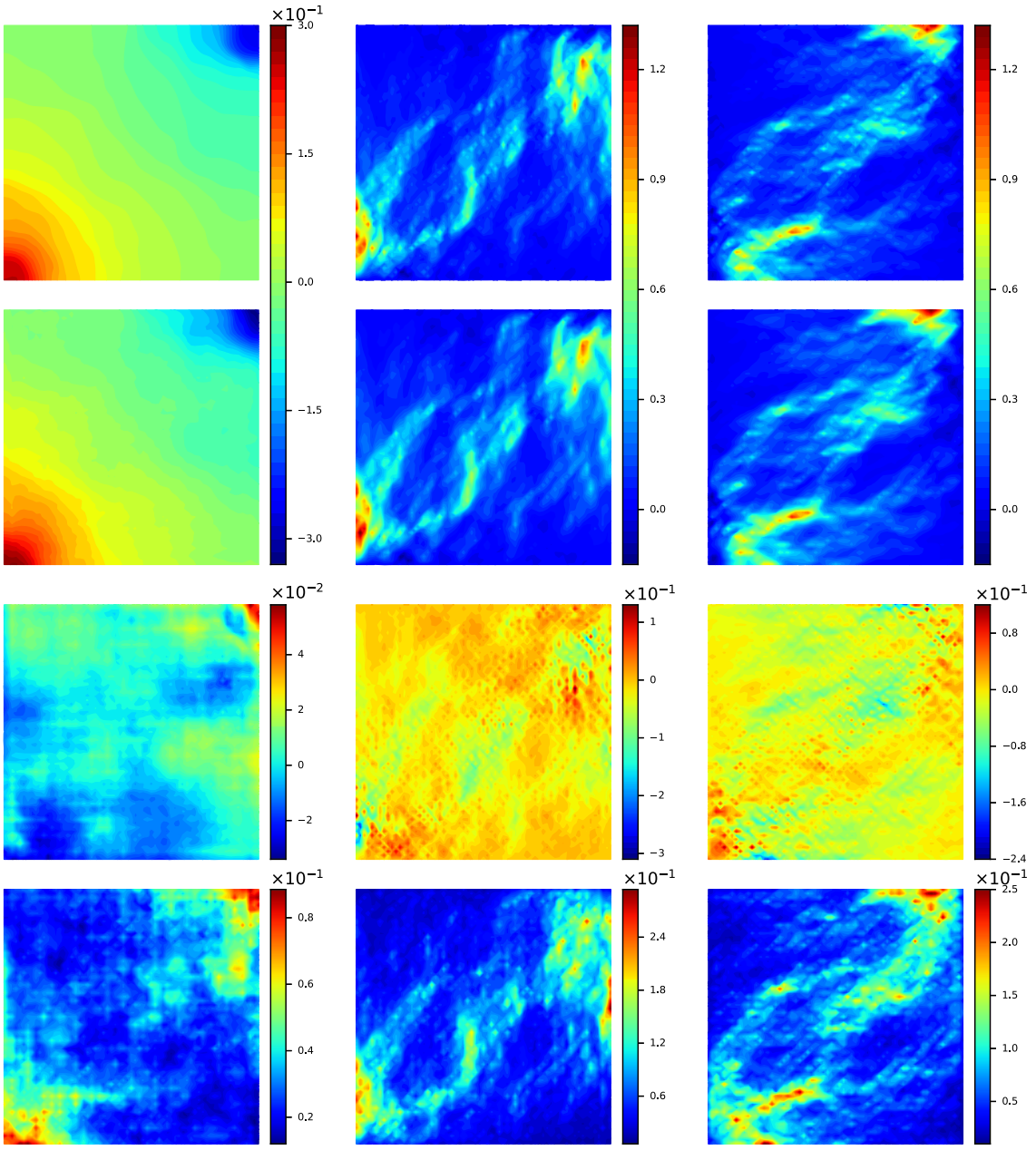


(b) 256 training data

Fig. 16. (*continued*)

parameters with a constraint that the number of model parameters being less than 0.25 million. The details of these experiments are given in Appendix A. The network configuration with the highest R^2-score that Hyperband finds is shown in Fig. 7 with more details provided in Table 1. This configuration is referred to as `DenseED-c16`. The 2nd–4th columns of Table 1 show the number C_f of output feature maps, the spatial resolution $H_f \times W_f$ of output feature maps, and the number of parameters of each layer in the network.

The network `DenseED-c16` is trained with Adam [65], a variant of stochastic gradient descent, with the loss function being L_2 regularized MSE which is implemented as weight decay in modern neural net frameworks, such as PyTorch and TensorFlow. Other loss functions may achieve better results, such as smoothed L_1 loss, or conditional GAN loss [41]. This requires further investigations to be considered in future publications. The initial learning rate is 0.015, weight decay (regularization on weights) is 0.0005, the batch size is 16. We also use a learning rate scheduler which drops 10 times on plateau of the rooted MSE. The model is trained 200 epochs. We train the model with the dataset introduced in Section 4.1.

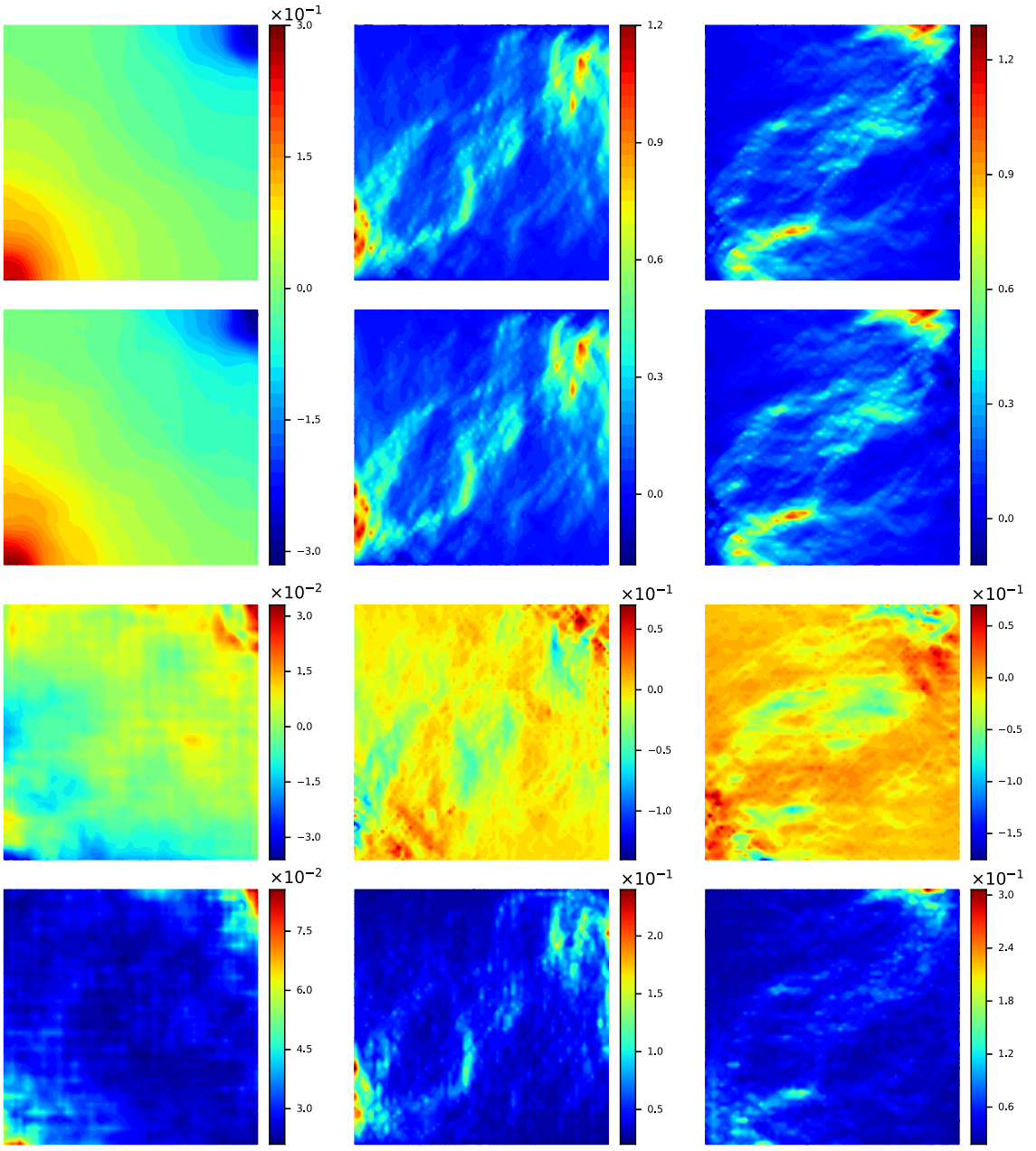


(a) 128 training data

Fig. 17. Prediction for the input realization shown in Fig. 6c from the KLE4225 dataset. The rows from top to bottom show the simulation output fields (ground truth), the predictive mean $\mathbb{E}[\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D}]$, the error of the above two, and two standard deviation of the predictive output distribution per pixel $\mathrm{Var}(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{D})$. The three columns from left to right correspond to the pressure field p, and the two velocity fields $\mathbf{u}_y$, $\mathbf{u}_x$, respectively.

Training the deterministic neural networks with L_2 regularized MSE is equivalent to finding the maximum *a posterior* of the uncertain parameters in Bayesian neural networks whose prior is independent normal. The typical training process is shown in Fig. 8.

We train each network with different number of training data of KLE50, KLE500, and KLE4225. The validation R^2-score is shown in Fig. 9, which shows that, with the same training data, the R^2-score is closer to 1 when the intrinsic dimensionality is smaller, and the R^2-score is higher with more training data of the same dimensionality. Note that the score is more than 0.9 with reasonably small size training dataset for all the three cases which have dimensionality from 50 to 4225. This shows the effectiveness of the network `DenseED-c16` for both low-dimensional and high-dimensional problems.



(b) 512 training data

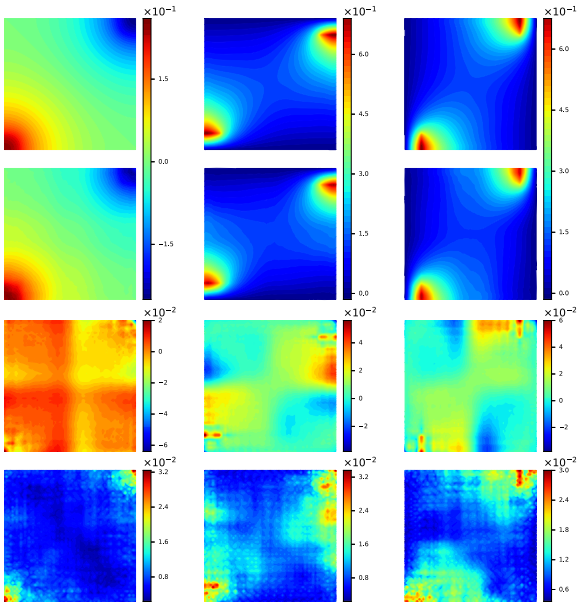
Fig. 17. (*continued*)

The prediction of the output fields can be easily obtained in the test time by feeding the test input permeability field $\mathbf{x}^*$ into the trained network, i.e. $\hat{\mathbf{y}}^* = \mathbf{f}(\mathbf{x}^*)$. We show the prediction of the test input shown in Fig. 6 using `DenseED-c16`, which is trained with three datasets (KLE50, KLE500, KLE4225) in Figs. 10, 11, and 12, respectively. The predictions are quite good even for the KLE4225 case, where both the input and output fields vary rapidly in certain regions of the domain.

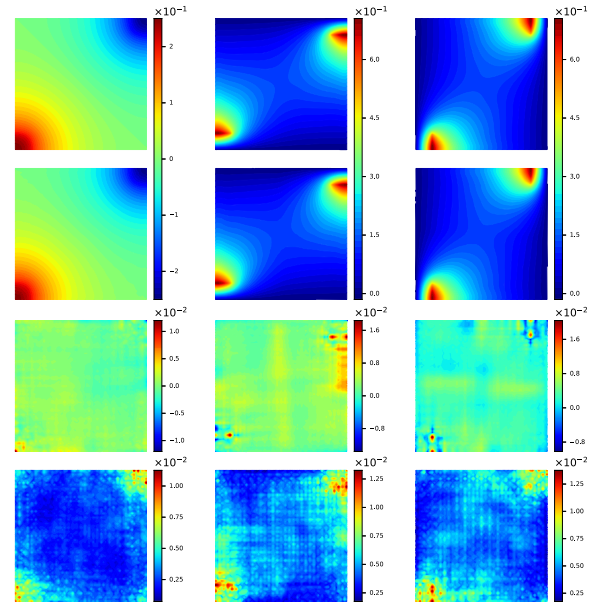
4.4. Bayesian surrogate model

For all experiments, we only consider the homoscedastic noise model for Bayesian neural networks, i.e. output-wise Gaussian noise with Gamma prior on its precision β, and Student's t-prior on $\mathbf{w}$.

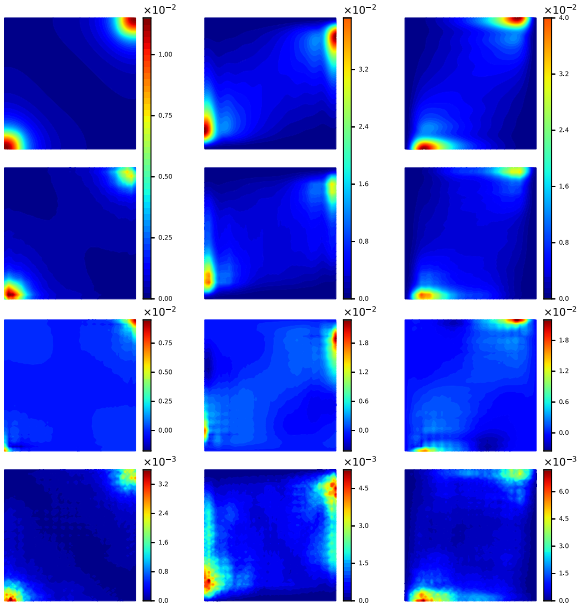
The set of all uncertain parameters is denoted as $\boldsymbol{\theta} = \{\mathbf{w}, \beta\}$. We apply SVGD to the Bayesian neural network with S samples $\{\boldsymbol{\theta}^i\}_{i=1}^S$ from the posterior $p(\boldsymbol{\theta} \mid \mathcal{D})$, i.e. S set of deterministic model parameters $\{\mathbf{w}^i\}_{i=1}^S$ of `DenseED`'s and noise



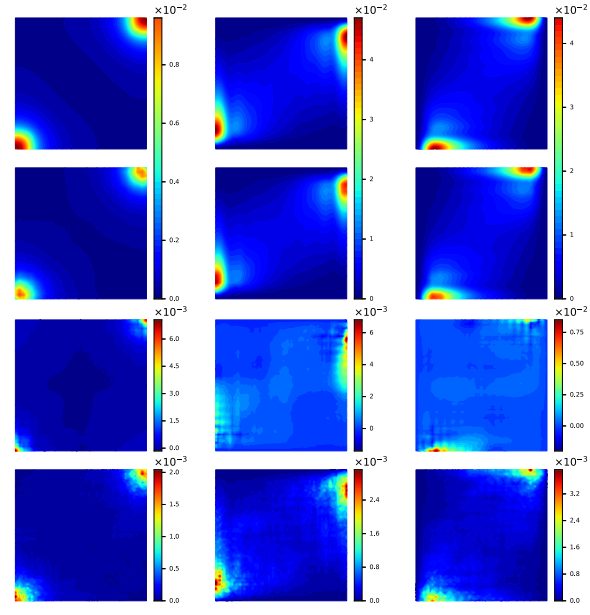
(a) 32 training data



(b) 128 training data



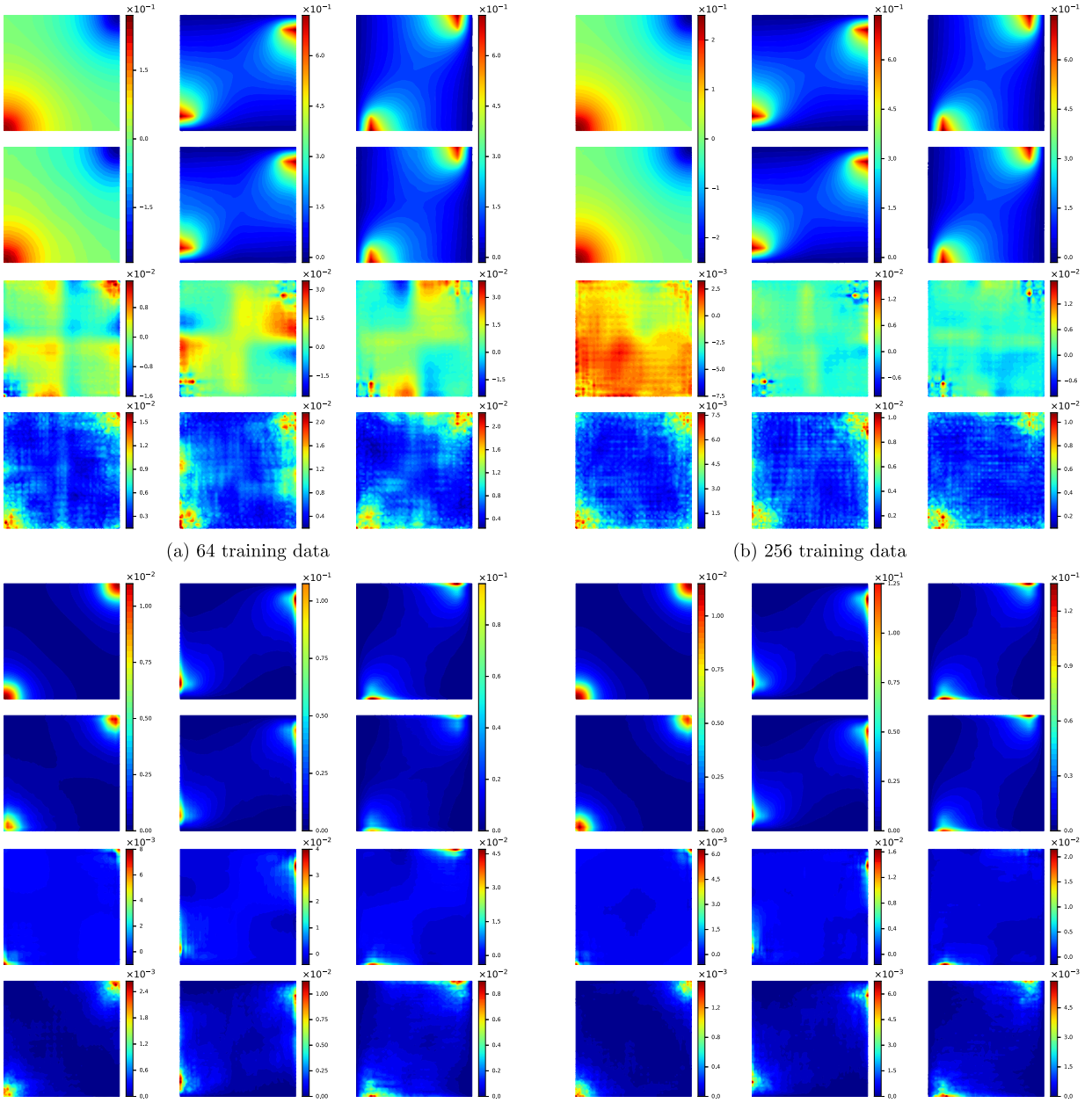
(c) 32 training data



(d) 128 training data

Fig. 18. Uncertainty propagation for KLE50: In (a) and (b) from top to bottom, we show the Monte Carlo output mean, predictive output mean $\mathbb{E}_{\boldsymbol{\theta}}[\mathbb{E}[\mathbf{y} \mid \boldsymbol{\theta}]]$, the error of the above two, and two standard deviation of the conditional predictive mean $\mathrm{Var}_{\boldsymbol{\theta}}(\mathbb{E}[\mathbf{y} \mid \boldsymbol{\theta}])$. The three columns from left to right correspond to the pressure field p, and the two velocity fields $\mathbf{u}_y$, $\mathbf{u}_x$, respectively. In (c) and (d) from top to bottom, we show the Monte Carlo output variance, predictive output variance $\mathbb{E}_{\boldsymbol{\theta}}[\mathrm{Var}(\mathbf{y} \mid \boldsymbol{\theta})]$, the error of the above two, and two standard deviation of the conditional predictive variance $\mathrm{Var}_{\boldsymbol{\theta}}(\mathrm{Var}(\mathbf{y} \mid \boldsymbol{\theta}))$. The three columns are the same as in (a) and (b).

precision $\{\beta^i\}_{i=1}^S$. In our implementation, this corresponds to S different initializations for the deterministic `DenseED` and noise precision (a scalar). We update the parameters of S `DenseED`s and the corresponding noise precision using the SVGD Algorithm 1. The kernel is chosen to be $k(\mathbf{x}, \mathbf{x}') = \exp(-\left\|\mathbf{x} - \mathbf{x}'\right\|_2^2 / h)$, with median heuristic for the choice of the kernel bandwidth $h = H^2 / \log S$, where H is the median of the pairwise distances between the current samples $\{\boldsymbol{\theta}^i\}_{i=1}^S$. We typically use $S = 20$ samples of $\boldsymbol{\theta}$ to approximate the empirical measure of the posterior. For large number of training data, the unnormalized posterior is evaluated using mini-batches of training data, i.e. $\tilde{p}(\boldsymbol{\theta} \mid \mathcal{D}) = \prod_{i=1}^N p(\mathbf{y}^i \mid \boldsymbol{\theta}, \mathbf{x}^i) p_0(\boldsymbol{\theta}) \approx N/B \prod_{i=1}^B p(\mathbf{y}^i \mid \boldsymbol{\theta}, \mathbf{x}^i) p_0(\boldsymbol{\theta})$. We observe that even for small training data such as 512, using smaller batch size (e.g. 16) helps to obtain lower training and test errors, but with more time for training. We use Adam [65] to update $\boldsymbol{\theta}$ using the



(a) 64 training data (b) 256 training data

(c) 64 training data (d) 256 training data

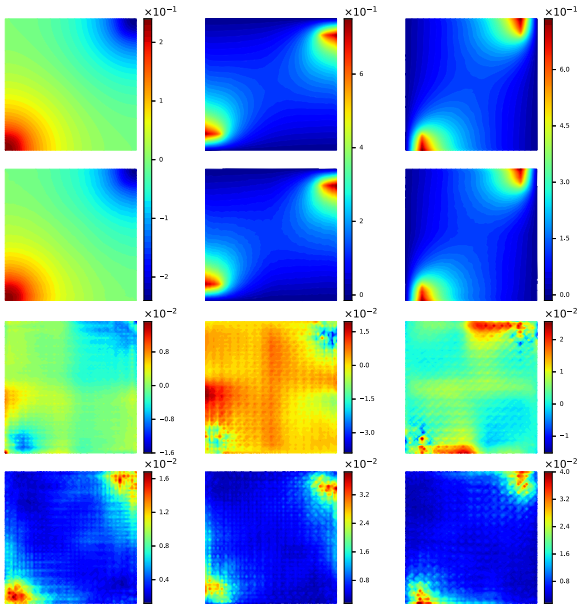
Fig. 19. Uncertainty propagation for KLE500: (a), (b), (c), and (d) refer to Fig. 18.

gradient $\boldsymbol{\phi}$, instead of the vanilla stochastic gradient descent, for 300 epochs, with learning rate 0.002 for $\mathbf{w}$ and 0.01 for β, and a learning rate scheduler that decreases by 10 times when the training RMSE is on plateau.

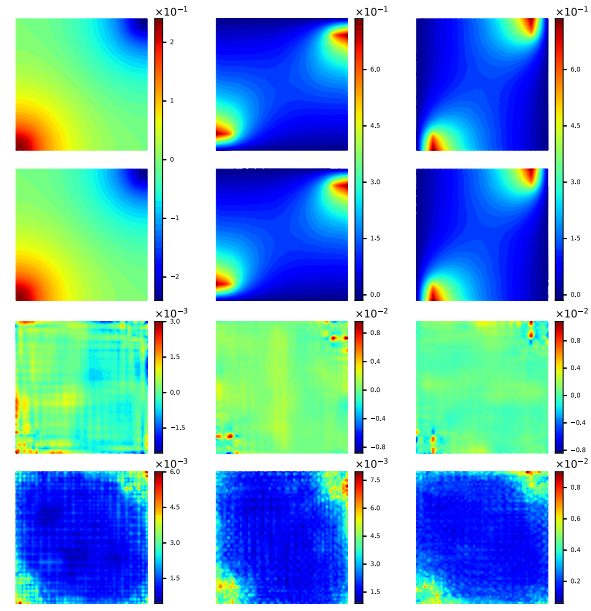
The algorithm is implemented in PyTorch and runs on a single NVIDIA GeForce GTX 1080 Ti X GPU which requires about 2000–7000 seconds for training 300 epochs, when the training data size varies from 32 to 512. The training time depends heavily on the training mini-batch size, which is 16 for all cases. Potential ways to speed up significantly the training process include increasing the mini-batch size (for a fixed number of epochs) [66], or implementing the SVGD in parallel using multi-GPUs. The python source code is available at https://github.com/cics-nd/cnn-surrogate.

We report next the R^2-score computed similar to the non-Bayesian case, except that the predicted output mean is used to compare with the test target. The scores are shown in Fig. 13. We can see that the Bayesian surrogate improves the R^2-score significantly over the non-Bayesian model.

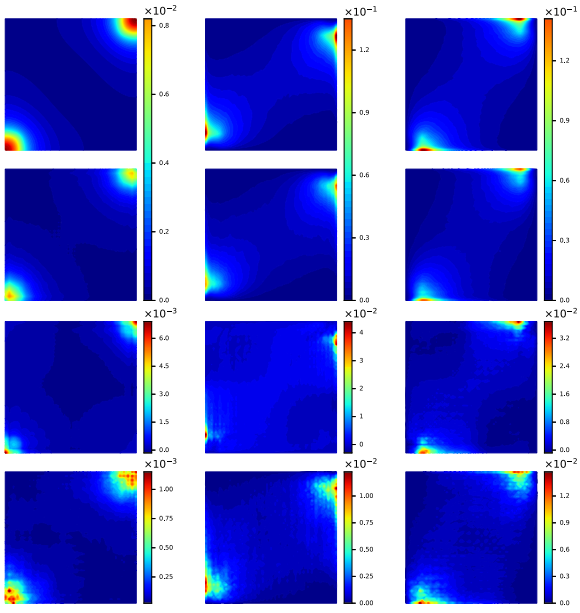
We also report the MNLP for test data in Fig. 14, which is a proper scoring rule and usually used to access the quality of the predictive uncertainty [67].



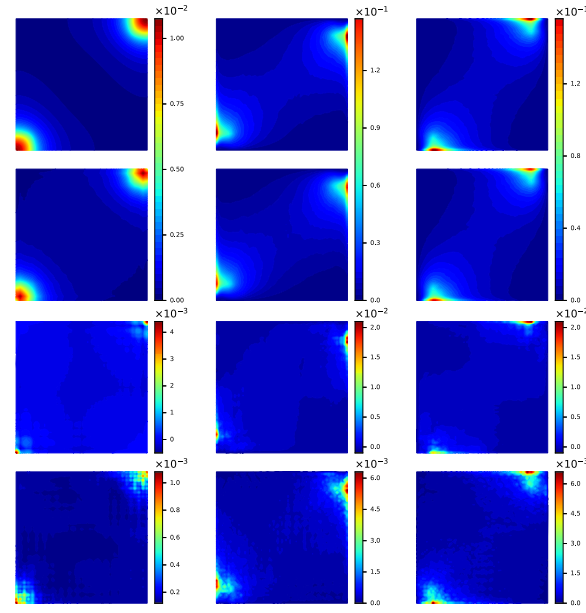
(a) 128 training data



(b) 512 training data



(c) 128 training data



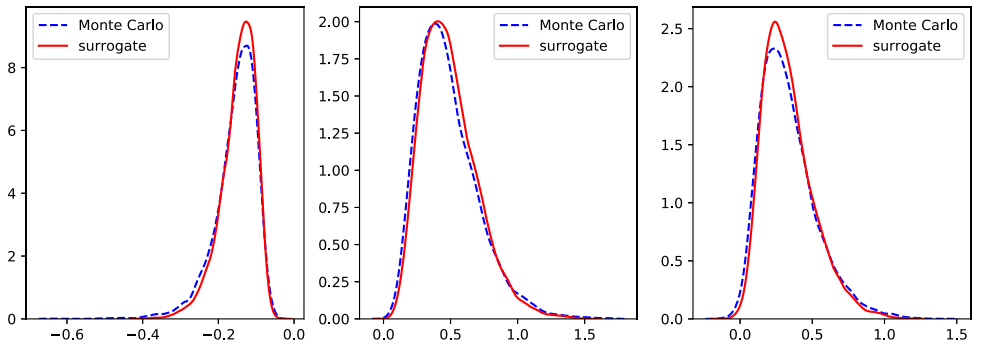
(d) 512 training data

Fig. 20. Uncertainty propagation for KLE4225: (a), (b), (c), and (d) refer to Fig. 18.

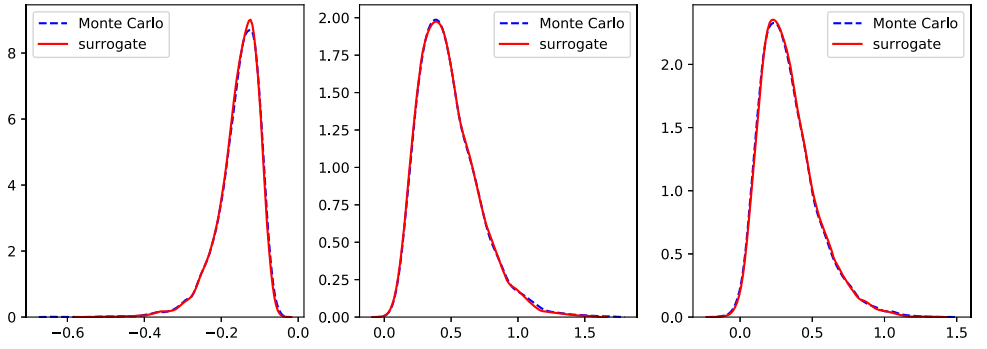
The R^2-score gives us a general estimate of how well the regression performs. For a given input permeability field, the Bayesian neural network can predict the mean of the corresponding output fields, and also gives an uncertainty estimate represented as predictive variance at each spatial location. This is information unavailable for deterministic models but desirable when the training data is small. In Figs. 15, 16, and 17, we show predictions for the test input shown in Fig. 6 with training data from KLE50, KLE500, and KLE4225, respectively. We can see that the predictive accuracy improves as the size of the training dataset increases, while the predictive uncertainty drops.

We also performed uncertainty propagation by feeding the trained Bayesian surrogate with $10,000$ input realizations sampled from the Gaussian field, and calculating the output statistics as in Section 3.4. In Figs. 18, 19 and 20, we show the uncertainty propagation results and compare with Monte Carlo using the $10,000$ uncertainty propagation data for the Bayesian surrogate trained with the datasets KLE50, KLE500, and KLE4225.

We show the estimate of pressure p, velocity components u_x, u_y at locations $(0.85, 0.88)$, and $(0.68, 0.05)$ on the unit square for 128, and 512 training data of KLE4225 in Fig. 21, and 22, respectively. The PDF is obtained by kernel density



(a) Location $(0.85, 0.88)$ 128 training data



(b) Location $(0.85, 0.88)$ 512 training data

Fig. 21. Distribution estimate for the pressure, and the two velocity components (from left to right) at location $(0.85, 0.88)$ with Bayesian surrogate trained with KLE4225 dataset.

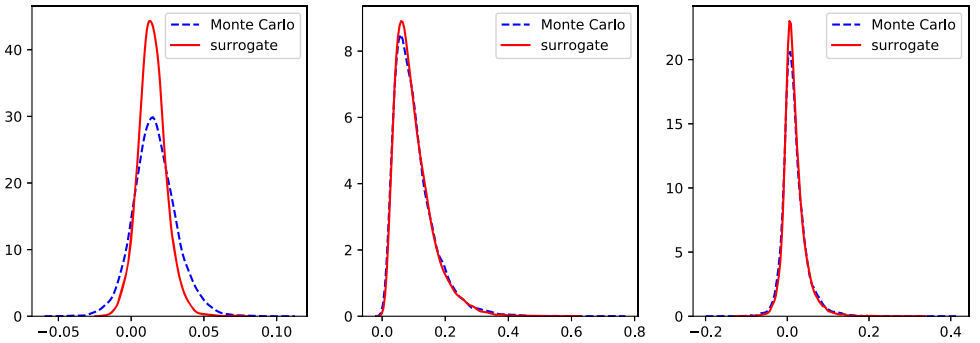
estimation using the predictive mean. We can see that the density estimate is close to the Monte Carlo result even when the training dataset is small, and becomes closer as the training dataset increases. From Fig. 22, we observe that the predictions for the velocity fields are better than the pressure field especially in locations away from the diagonal of the unit square domain, and this is in general the case for our current network architecture, where the three output fields are treated the same.

In order to access the quality of the computed uncertainty, we adopt the reliability diagram which expresses the discrepancy between the predictive probability and the frequency of falling in the predictive interval for the test data. The diagram is shown in Fig. 23. Overall our models are well-calibrated since they are quite close to the diagonal line which is the ideal case, especially the case when the training dataset size is 128 as shown in Fig. 23d. In general the model turns to be over-confident (small predictive uncertainty) when the training data is small, and gradually becomes prudent (larger predictive uncertainty) when the training data increases. The main reason for this observation is that the predictive uncertainty is dominated by the variation seen in the training data, which is smaller when there is less training data. The initial learning rates and their scheduling scheme for network parameters $\mathbf{w}$ and noise precision β may also play roles here since the uncertainty is determined by the optimum that stochastic optimization obtains.

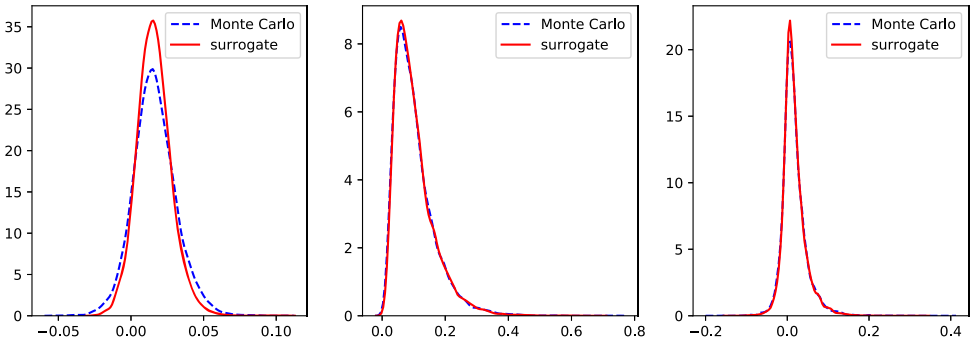
Remark 5. The reliability diagram not only can be used for assessing the quality of uncertainty but also for evaluating how well it is *calibrated*. Fig. 23c shows that in general when the number of training data increases, the model predictive uncertainty also increases. This is attributed to two reasons. Firstly, the output variance is underestimated when limited data is available. Secondly, the optimized noise level β is related to the hyperparameters, e.g. learning rate for β, which should be tuned to calibrate the uncertainty. As can be seen in Fig. 23d, the uncertainty is well calibrated for all models trained with 128 data points for the selected set of hyperparameters.

The training processes with different datasets are shown in Fig. 24. We can see that the training and test RMSE converge around $50 \sim 75$ epochs of training for KLE50, and KLE500, but the convergence for KLE4225 seems to take longer time. The training dataset size is 256 for all three sets.

To empirically show that using 20 samples of the Bayesian neural nets for the SVGD algorithm is sufficient in our problem, we show in Fig. 25 the convergence of the test and training RMSE versus the number of samples.



(a) Location $(0.68, 0.05)$ 128 training data



(b) Location $(0.68, 0.05)$ 512 training data

Fig. 22. Distribution estimate for the pressure, and the two velocity components (from left to right) at location $(0.68, 0.05)$ with Bayesian surrogate trained with KLE4225 dataset.

5. Conclusions

In this work, we explore the use of a Bayesian neural network to predict the output fields of a system governed by stochastic partial differential equations with high-dimensional stochastic input. The approach is built on the highly expressive deep convolutional encoder–decoder networks. The end-to-end image-to-image regression avoids the usual linear dimensionality reduction step, and achieves promising results in terms of predictive performance and uncertainty modeling, even with limited training data.

We show in experiments that one network (`DenseED-c16`) works well across problems with different intrinsic dimensionalities. The performance of deep neural networks with limited training data is due on their unique generalization property [17,68] which effectively says that over-parameterized deep neural nets lack over-fitting.

The Bayesian model provides uncertainty estimates for our predictions thus accounting for epistemic uncertainty when training with small datasets. We show that Bayesian inference based on SVGD works well for training the Bayesian neural network, and presented results on uncertainty propagation tasks. The uncertainty of the trained model is well-calibrated by investigating the reliability diagrams.

The presented methodology was also tested on a channelized permeability field.[3] The dataset considered contained 144 patches cropped from the original 250×250 channelized field, with patch size 64 and stride 16. Without much tuning of the network structure and hyperparameters, the trained network showed very good predictability over the three output fields even through this problem is clearly more challenging than the one with a Gaussian random field for the log-permeability.

There are two potential directions to improve the regression performance of the non-Bayesian surrogate:

- One could use a stronger loss function such as an adversarial loss [41] to enforce the model to capture the strong variations in the output. Similar situation has been explored for jet maps in high-energy particle physics [69].
- Since the prediction performance for the velocity fields is better than for the pressure field using our current network architecture `DenseED-c16`, one could use group convolution in the last decoding layer to reduce the influences of features for predicting the pressure and the velocity fields. Extra experiments could be conducted to see whether training separate models for each output field brings benefit.

[3] http://www.trainingimages.org/uploads/3/4/7/0/34703305/ti_strebelle.sgems.

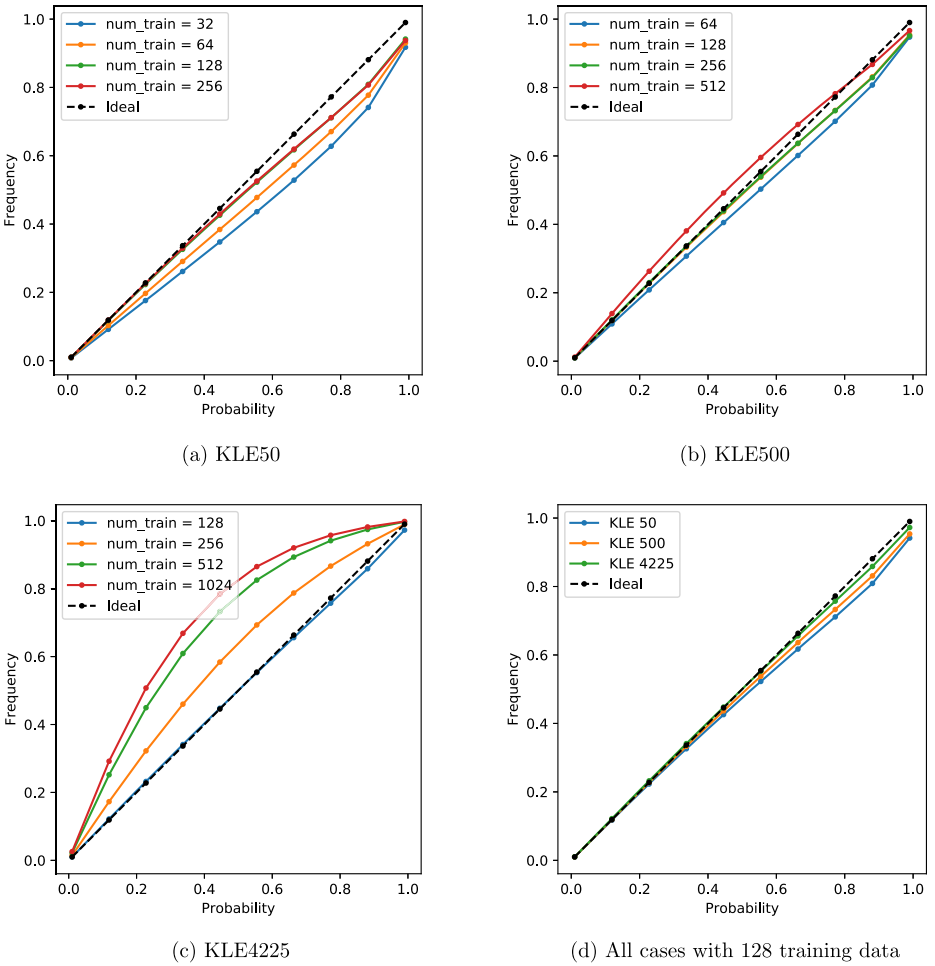


Fig. 23. Reliability diagrams. Subfigures (a), (b), (c) show the reliability diagram for KLE50, KLE500, KLE4225, respectively. (d) shows the diagrams when the training dataset size is 128 for all three datasets.

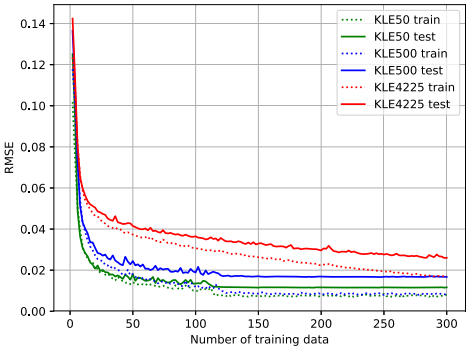


Fig. 24. Test and training RMSE for training of the SVGD using 128 training data from KLE50, KLE500, and KLE4225.

For the Bayesian neural network, the inference task is genuinely difficult since it is asked to compute the posterior of millions of random variables based only on hundreds of training data. It is thus important to emphasize the need for further investigations on the uncertainty quality of the Bayesian surrogate. Exploring other priors for the network weights, and using recent advances in variational inference for Bayesian neural networks are valuable directions to pursue.

The image-to-image regression approach can be used to handle the prediction of systems with different source terms (or boundary conditions), by *adding the source field as another channel in the input besides the material property field*. For implementation, one only needs to increase the number of input channels by 1, and leave everything else unchanged. Similarly one can address dynamical problems by adding time as an additional input channel.

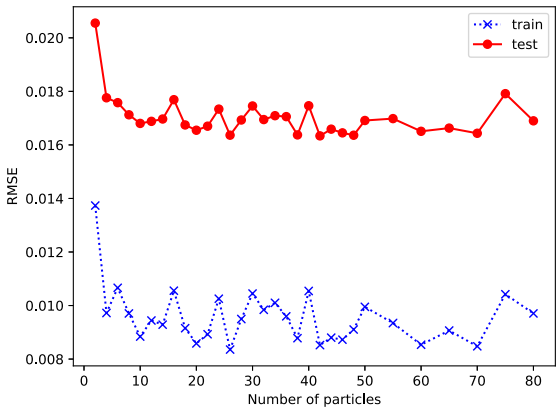


Fig. 25. Test and training errors using the SVGD to train the Bayesian NN with 128 training data of KLE500. As we increase the number of posterior samples for $\boldsymbol{\theta}$, i.e. the number of deterministic neural networks (and noise precision), both training and test errors drop and become steady. This plot empirically supports the reason we choose 20 model instances/samples for SVGD.

Our current network does not utilize any information from physics, such as the governing equations or constraints between the three output fields. Incorporating physics information into deep neural networks is a more principled ways to improve the model performance.

Acknowledgements

This work was supported from the University of Notre Dame, the Center for Research Computing (CRC) and the Center for Informatics and Computational Science (CICS). Early developments were supported by the Computer Science and Mathematics Division of ORNL under the DARPA EQUiPS program. N.Z. thanks the Technische Universität München, Institute for Advanced Study for support through a Hans Fisher Senior Fellowship, funded by the German Excellence Initiative and the European Union Seventh Framework Programme under Grant Agreement No. 291763. The authors acknowledge Dr. Steven Atkinson from the CICS for generating and providing us the Darcy flow datasets.

Appendix A. Optimization of the network hyperparameters

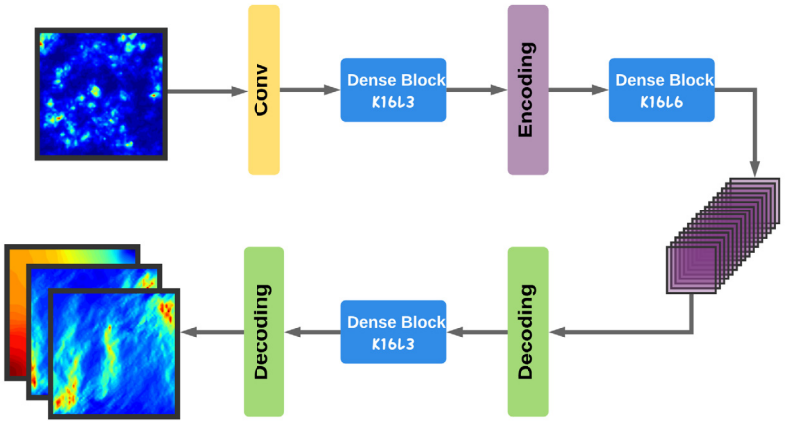
We optimize the network hyperparameters following the general architecture shown in Fig. 3. The encoding layer in Fig. 2(a) down-samples its input feature maps by 2 using convolution of stride 2 as in the first convolution layer. The decoding layer in Fig. 2(b) up-samples the feature maps by 2 using transposed convolution instead of pooling layers (as commonly seen in image classification networks) since the location information is critical for regression. The number of down-sampling and up-sampling layers are the same. The datasets are described in Section 4.1. We aim to find one general network architecture that works well across different datasets for both the Bayesian and non-Bayesian network models.

Experiment 1 – Spatial dimension of feature maps at the coarsest scale: This is determined by how many down-sampling layers are used. Shrinking the spatial dimension of feature maps can extract high-level or coarse information of the input permeability field, which is subsequently used to predict the output pressure and velocity fields. This design choice is related to the *receptive field* [70] of an unit within a certain layer, which is the region in the input image that affects this unit feedforward, or the region in the output image that affects this unit when back-propagating gradients. For dense prediction tasks such as our surrogate modeling problems, it is important for each pixel in the code feature maps to have the suitable receptive field in the input and output images. We observe that for the dataset generated with fewer KLE terms, both the input and output velocity fields are smoother, or the output has stronger correlation across pixels, such as KLE50 in Fig. 6a. But for KLE500 or KLE4225 in Figs. 6b and 6c, the fields vary more rapidly from pixel to pixel, but still there is weak long-range correlation between pixel values.

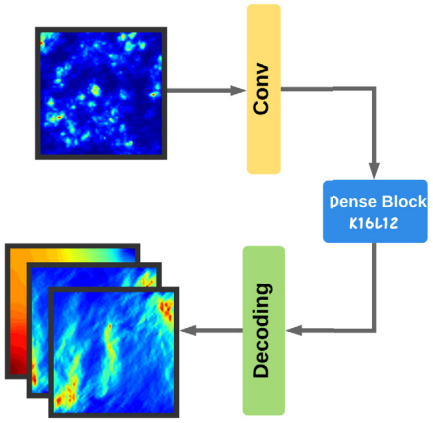
In our experimental setup we have used no skip connections, the convolution kernels were fixed to be $k7s2p2$ for the first Conv layer, $k3s1p1$ in the layers within dense block, $k1s1p0$ and $k3s2p1$ in the encoding and decoding layers, and $k5s2p2$ in the last ConvT layer. No dropout was used and the growth rate of the dense blocks was taken as 16.

The loss function is taken as the regularized MSE in Eq. (3), and the validation metric as the R^2-score in Eq. (23). Adam optimizer is used for training 200 epochs, with learning rate 0.001 and a plateau scheduler on the test RMSE. The batch size is always smaller than the number of training data (e.g. $16, 32, 64$ for datasets with $32, 64, 128$ data, respectively). The weight decay is set at 0.0005.

We vary the *dense block configurations*, i.e. a list of integers specifying the number of layers within each dense block. This list contains odd number of integers because of the symmetry between encoding and decoding paths. For example, blocks



(a) DenseED-c16 with blocks $(3, 6, 3)$.



(b) DenseED-c32 with blocks $(12,)$.

Fig. A.26. Two configurations DenseED-c16 and DenseED-c32 of DenseED.

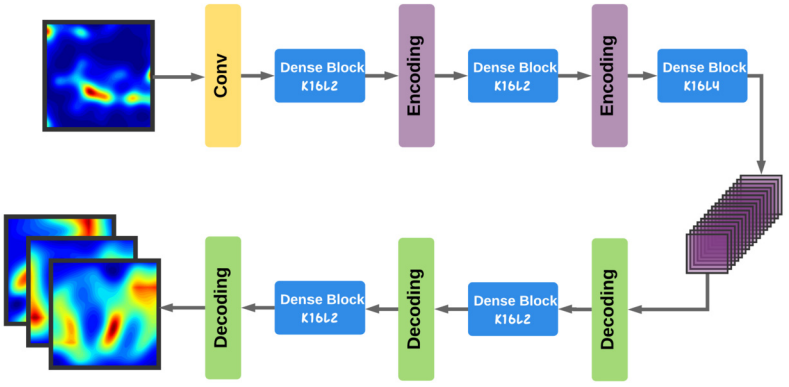


Fig. A.27. Configuration DenseED-c8 with blocks $(2, 2, 4, 2, 2)$.

$(3, 6, 3)$ specify the network architecture in Fig. A.26a. $K16L6$ in the second dense block means there are 6 layers within the dense blocks, and the growth rate for each layer is 16. In this case, the code dimension (purple feature maps) is 16×16 since there are two down-sampling layers in the encoding path.

The second block configuration is $(12,)$, i.e. only one dense block sits after the first Conv layer, as shown in Fig. A.26b or $(2, 2, 4, 2, 2)$ as shown in Fig. A.27. The numbers within parenthesis stand for the number of layers within each dense block.

Table A.2
Test R^2-score for `DenseED-c16`.

	KLE50	KLE500	KLE4225
32	0.718	0.551	0.280
64	0.883	0.817	0.662
128	0.947	0.913	0.829
256	0.970	0.954	0.927
512	–	0.976	0.963

Table A.3
Test R^2-score for `DenseED-c8`.

	KLE50	KLE500	KLE4225
32	0.661	0.344	0.052
64	0.883	0.687	0.411
128	0.931	0.842	0.607
256	0.963	0.934	0.753
512	–	0.971	0.829

Table A.4
Test R^2-score for `DenseED-c32`.

	KLE50	KLE500	KLE4225
32	0.563	0.558	0.413
64	0.811	0.807	0.704
128	0.893	0.899	0.863
256	0.931	0.937	0.909
512	–	0.954	0.939

Table A.5
Network architecture hyperparameters and associated ranges for the `DenseED-c16` network.

Hyperparameters	Type	Values
Encoding dense block layers	Q-uniform	[1, 8]
Bottom dense block layers	Q-uniform	[3, 8]
Decoding dense block layers	Q-uniform	[1, 8]
Growth rate	Categorical	16, 32, 48
Features after 1st Conv	Categorical	32, 48

The validation R^2-scores of the above three networks `DenseED-c16`, `DenseED-c8`, `DenseED-c32` on the test set (500 Monte Carlo samples from each KLE case) are summarized in Tables A.2, A.3, and A.4. From these tables, it can be seen that the *DenseED-c*16 network works well for all three datasets. The network `DenseED-c8` does not work well for the KLE4225 dataset, since its code dimension is too small. This enforces the output field to be too smooth, which is still favorable to the KLE50 dataset. We can also clearly note that the network `DenseED-c32` does not work well for KLE50, even though it performs well for the KLE4225 dataset.

Predictions and learning curve for the selected network design `DenseED-c16` are presented in Section 4.3.

Experiment 2 – Hyperparameter optimization with Hyperband: The hyperparameters to select include the ones specifying the network architecture and the training process. We separately optimize these two sets of hyperparameters. Hyperband [64] is a bandit random search algorithm which has several rounds of successive halving. There are two inputs for this algorithm, i.e. $R = 243$, the maximum iterations for each hyperparameter configuration (here each iteration corresponding to one epoch of training) and $\eta = 3$ which specifies to keep 1/3 of the best configurations after running through all the candidate configurations.

With the search space specified in Table A.5 together with the constraint that the number of parameters is less than 0.25 million, results in the network architecture presented in Table 1.

The search for the training process hyperparameters as in Table A.6 gives approximately the following hyperparameters: initial learning rate 0.002, weight decay 0.0005, batch size 16, and Adam optimizer.

Note that the hyperparameters found by Hyperband are only sub-optimal but indicate the potential range. The actual hyperparameters used for training are presented in Section 4.3.

Table A.6
Training hyperparameters and associated ranges for the `DenseED-c16` network.

Hyperparameters	Type	Values
Initial learning rate	Uniform	$[10^{-4}, 5 \times 10^{-3}]$
Weight decay	Uniform	$[5 \times 10^{-5}, 10^{-2}]$
Batch size	Categorical	16, 24
Optimizer	Categorical	Adam, RMSprop

Table A.7
Network sparsity experiments: pruned weights under a threshold of 0.0075.

Models	Percentage of pruned weights	Percentage of reduced R^2
KLE4225 – 128	16.05%	1.14%
KLE4225 – 256	15.67%	1.24%
KLE4225 – 512	15.25%	1.23%
KLE4225 – 1024	13.89%	0.72%
KLE500 – 64	17.27%	0.71%
KLE500 – 128	16.38%	0.75%
KLE500 – 256	15.40%	0.97%
KLE500 – 512	14.49%	0.68%
KLE50 – 32	17.88%	1.29%
KLE50 – 64	17.74%	0.71%
KLE50 – 128	17.00%	0.86%
KLE50 – 256	16.67%	0.76%

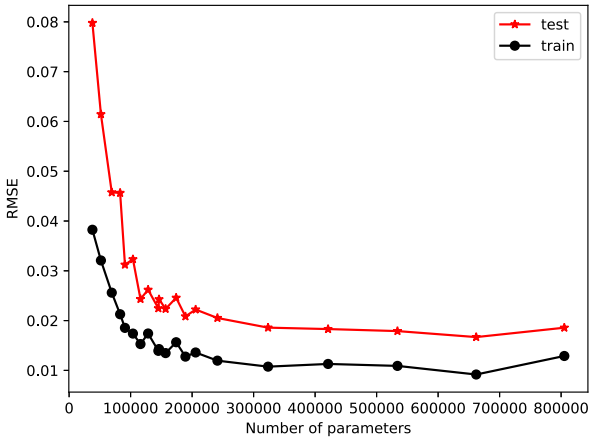


Fig. A.28. Generalization error of over-parameterized neural networks `DenseED-c16`.

Appendix B. Sparsity of Bayesian neural networks

In this section, a study is provided of the sparsity of the Bayesian neural networks obtained after training. As reported recently [19], there is a direct connection between generalization and compression of deep neural networks. For the Darcy flow model, the neural network weights (corresponding to the 20 realizations of the uncertain weights) are set to 0 when under the threshold of 0.0075. The percentage of pruned weights and corresponding percentage of reduction in the R^2-score are reported in Table A.7. It can be seen that with about 15% of the weights pruned out, the R^2-score reduces only by 1%. This is certainly not competitive with the state of the art published work in network compression.

Appendix C. Generalization behavior

We show in this section that over-parameterization does not increase the generalization error in the encoder–decoder network. We vary the number of layers in each of the dense blocks of `DenseED-c16` as in Fig. 7, while keeping all other settings unchanged (including 256 training data, regularization, epochs, learning rate, etc.). The training and test RMSE are shown in Fig. A.28. This generalization curve can also be used to select a network with an appropriate number of model parameters. In the Darcy flow studies, a network with 241, 164 parameters was selected.

References

[1] I. Bilionis, N. Zabaras, Bayesian uncertainty propagation using Gaussian processes, in: Handbook of Uncertainty Quantification, 2016, pp. 1–45.
[2] I. Bilionis, N. Zabaras, B.A. Konomi, G. Lin, Multi-output separable Gaussian process: towards an efficient, fully Bayesian paradigm for uncertainty quantification, J. Comput. Phys. 241 (2013) 212–239.
[3] I. Bilionis, N. Zabaras, Multi-output local Gaussian process regression: applications to uncertainty quantification, J. Comput. Phys. 231 (17) (2012) 5718–5746.
[4] D. Xiu, G.E. Karniadakis, The Wiener–Askey polynomial chaos for stochastic differential equations, SIAM J. Sci. Comput. 24 (2) (2002) 619–644.
[5] S. Torquato, Random Heterogeneous Materials: Microstructure and Macroscopic Properties, vol. 16, Springer Science & Business Media, 2013.
[6] L.J.P. van der Maaten, E.O. Postma, H.J. van den Herik, Dimensionality Reduction: A Comparative Review, Tech. rep., Tilburg University, 2009, http://homepage.tudelft.nl/19j49/Matlab_Toolbox_for_Dimensionality_Reduction_files/TR_Dimensiereductie.pdf.
[7] L.v.d. Maaten, G. Hinton, Visualizing data using t-sne, J. Mach. Learn. Res. 9 (Nov 2008) 2579–2605.
[8] G.E. Hinton, R.R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507.
[9] D.P. Kingma, M. Welling, Auto-encoding variational bayes, CoRR, arXiv:1312.6114, http://dblp.uni-trier.de/db/journals/corr/corr1312.html#KingmaW13.
[10] N.D. Lawrence, Gaussian process latent variable models for visualisation of high dimensional data, in: Advances in Neural Information Processing Systems, 2004, pp. 329–336.
[11] C. Grigo, P.-S. Koutsourelakis, Bayesian model and dimension reduction for uncertainty propagation: applications in random media, arXiv preprint, arXiv:1711.02475.
[12] Y. Bengio, Learning deep architectures for AI, Found. Trends Mach. Learn. 2 (1) (2009) 1–127, https://doi.org/10.1561/2200000006.
[13] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.
[14] M.D. Zeiler, R. Fergus, Visualizing and understanding convolutional networks, in: European Conference on Computer Vision, Springer, 2014, pp. 818–833.
[15] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.
[16] G.E. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets, Neural Comput. 18 (7) (2006) 1527–1554.
[17] C. Zhang, S. Bengio, M. Hardt, B. Recht, O. Vinyals, Understanding deep learning requires rethinking generalization, CoRR, arXiv:1611.03530, http://arxiv.org/abs/1611.03530.
[18] G.K. Dziugaite, D.M. Roy, Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data, arXiv preprint, arXiv:1703.11008.
[19] S. Arora, R. Ge, B. Neyshabur, Y. Zhang, Stronger generalization bounds for deep nets via a compression approach, arXiv preprint, arXiv:1802.05296.
[20] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, J. Sohl-Dickstein, On the expressive power of deep neural networks, arXiv preprint, arXiv:1606.05336.
[21] J.N. Kutz, Deep learning in fluid dynamics, J. Fluid Mech. 814 (2017) 1–4.
[22] J. Marçais, J.-R. de Dreuzy, Prospective interest of deep learning for hydrological inference, Groundwater 55 (5) (2017) 688–692.
[23] S. Chan, A.H. Elsheikh, A machine learning approach for efficient uncertainty quantification using multiscale methods, J. Comput. Phys. 354 (Supplement C) (2018) 493–511, https://doi.org/10.1016/j.jcp.2017.10.034, http://www.sciencedirect.com/science/article/pii/S0021999117307933.
[24] S. Min, B. Lee, S. Yoon, Deep learning in bioinformatics, Brief. Bioinform. 18 (5) (2017) 851–869.
[25] P. Baldi, P. Sadowski, D. Whiteson, Searching for exotic particles in high-energy physics with deep learning, Nat. Commun. 5 (2014) 4308.
[26] D.J. MacKay, A practical Bayesian framework for backpropagation networks, Neural Comput. 4 (3) (1992) 448–472.
[27] R.M. Neal, Bayesian Learning for Neural Networks, vol. 118, Springer Science & Business Media, 2012.
[28] Y. Gal, Z. Ghahramani, Dropout as a Bayesian approximation: representing model uncertainty in deep learning, in: International Conference on Machine Learning, 2016, pp. 1050–1059.
[29] C. Blundell, J. Cornebise, K. Kavukcuoglu, D. Wierstra, Weight uncertainty in neural networks, arXiv preprint, arXiv:1505.05424.
[30] Q. Liu, D. Wang, Stein variational gradient descent: a general purpose Bayesian inference algorithm, in: Advances In Neural Information Processing Systems, 2016, pp. 2378–2386.
[31] D.P. Kingma, T. Salimans, M. Welling, Variational dropout and the local reparameterization trick, in: Advances in Neural Information Processing Systems, 2015, pp. 2575–2583.
[32] J.M. Hernández-Lobato, R. Adams, Probabilistic backpropagation for scalable learning of bayesian neural networks, in: International Conference on Machine Learning, 2015, pp. 1861–1869.
[33] C. Louizos, M. Welling, Multiplicative normalizing flows for variational bayesian neural networks, arXiv preprint, arXiv:1703.01961.
[34] C. Louizos, K. Ullrich, M. Welling, Bayesian compression for deep learning, arXiv preprint, arXiv:1705.08665.
[35] G. Huang, Z. Liu, K.Q. Weinberger, L. van der Maaten, Densely connected convolutional networks, arXiv preprint, arXiv:1608.06993.
[36] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint, arXiv:1409.1556.
[37] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2818–2826.
[38] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[39] O. Ronneberger, P. Fischer, T. Brox, U-net: convolutional networks for biomedical image segmentation, in: International Conference on Medical Image Computing and Computer-Assisted Intervention, Springer, 2015, pp. 234–241.
[40] D. Eigen, C. Puhrsch, R. Fergus, Depth map prediction from a single image using a multi-scale deep network, in: Advances in Neural Information Processing Systems, 2014, pp. 2366–2374.
[41] P. Isola, J. Zhu, T. Zhou, A.A. Efros, Image-to-image translation with conditional adversarial networks, CoRR, arXiv:1611.07004, http://arxiv.org/abs/1611.07004.
[42] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, in: Advances in Neural Information Processing Systems, 2014, pp. 2672–2680.
[43] A. van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, et al., Conditional image generation with pixelcnn decoders, in: Advances in Neural Information Processing Systems, 2016, pp. 4790–4798.
[44] A.v.d. Oord, N. Kalchbrenner, K. Kavukcuoglu, Pixel recurrent neural networks, arXiv preprint, arXiv:1601.06759.
[45] R.K. Srivastava, K. Greff, J. Schmidhuber, Training very deep networks, in: Advances in Neural Information Processing Systems, 2015, pp. 2377–2385.
[46] S. Ioffe, C. Szegedy, Batch normalization: accelerating deep network training by reducing internal covariate shift, in: International Conference on Machine Learning, 2015, pp. 448–456.
[47] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, in: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, 2011, pp. 315–323.
[48] Theano Development Team, Theano: a Python framework for fast computation of mathematical expressions, http://arxiv.org/abs/1605.02688.
[49] J. Long, E. Shelhamer, T. Darrell, Fully convolutional networks for semantic segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3431–3440.

[50] V. Badrinarayanan, A. Kendall, R. Cipolla, Segnet: a deep convolutional encoder–decoder architecture for image segmentation, arXiv preprint, arXiv:1511.00561.
[51] S. Jégou, M. Drozdzal, D. Vazquez, A. Romero, Y. Bengio, The one hundred layers tiramisu: fully convolutional densenets for semantic segmentation, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPRW, IEEE, 2017, pp. 1175–1183.
[52] S. Han, H. Mao, W.J. Dally, Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding, arXiv preprint, arXiv:1510.00149.
[53] A. Lee, F. Caron, A. Doucet, C. Holmes, A hierarchical Bayesian framework for constructing sparsity-inducing priors, arXiv preprint, arXiv:1009.1914.
[54] R.B. Gramacy, H.K. Lee, Cases for the nugget in modeling computer experiments, Stat. Comput. 22 (3) (2012) 713–722.
[55] D.A. Nix, A.S. Weigend, Estimating the mean and variance of the target probability distribution, in: 1994 IEEE International Conference on Neural Networks, IEEE World Congress on Computational Intelligence, vol. 1, IEEE, 1994, pp. 55–60.
[56] Q.V. Le, A.J. Smola, S. Canu, Heteroscedastic Gaussian process regression, in: Proceedings of the 22nd International Conference on Machine Learning, ACM, 2005, pp. 489–496.
[57] A. Kendall, Y. Gal, What uncertainties do we need in Bayesian deep learning for computer vision?, arXiv preprint, arXiv:1703.04977.
[58] D.M. Blei, A. Kucukelbir, J.D. McAuliffe, Variational inference: a review for statisticians, J. Am. Stat. Assoc. 112 (518) (2017) 859–877, https://doi.org/10.1080/01621459.2017.1285773.
[59] Q. Liu, Stein variational gradient descent as gradient flow, in: Advances in Neural Information Processing Systems, 2017, pp. 3117–3125.
[60] Q. Liu, J. Lee, M. Jordan, A kernelized Stein discrepancy for goodness-of-fit tests, in: International Conference on Machine Learning, 2016, pp. 276–284.
[61] C.E. Rasmussen, C.K. Williams, Gaussian Processes for Machine Learning, vol. 1, MIT press, Cambridge, 2006.
[62] M.S. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M.E. Rognes, G.N. Wells, The FEniCS Project version 1.5, Arch. Numer. Softw. 3 (100) (2015), https://doi.org/10.11588/ans.2015.100.20553.
[63] B. Lakshminarayanan, A. Pritzel, C. Blundell, Simple and scalable predictive uncertainty estimation using deep ensembles, arXiv preprint, arXiv:1612.01474.
[64] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, A. Talwalkar, Hyperband: a novel bandit-based approach to hyperparameter optimization, arXiv preprint, arXiv:1603.06560.
[65] D. Kingma, J. Ba, Adam: a method for stochastic optimization, arXiv preprint, arXiv:1412.6980.
[66] P. Goyal, P. Dollár, R.B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, K. He, Accurate, large minibatch SGD: training ImageNet in 1 hour, CoRR, arXiv:1706.02677, http://arxiv.org/abs/1706.02677.
[67] J. Quinonero Candela, C. Rasmussen, F. Sinz, O. Bousquet, B. Schölkopf, Evaluating predictive uncertainty challenge, in: Machine Learning Challenges: Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Textual Entailment, Max-Planck-Gesellschaft, Springer, Berlin, Germany, 2006, pp. 1–27.
[68] T. Poggio, K. Kawaguchi, Q. Liao, B. Miranda, L. Rosasco, X. Boix, J. Hidary, H. Mhaskar, Theory of Deep Learning III: explaining the non-overfitting puzzle, arXiv:1801.00173.
[69] L. de Oliveira, M. Paganini, B. Nachman, Learning particle physics by example: location-aware generative adversarial networks for physics synthesis, arXiv preprint, arXiv:1701.05927.
[70] W. Luo, Y. Li, R. Urtasun, R. Zemel, Understanding the effective receptive field in deep convolutional neural networks, in: Advances in Neural Information Processing Systems, 2016, pp. 4898–4906.