FUNCTIONAL NEURAL NETWORK SURROGATES TRAINED ON SCARCE DATA

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ENERGY RESOURCE ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Dong H. Song

September 2022

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Daniel M. Tartakovsky) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Roland N. Horne)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Anthony R. Kovscek)

Approved for the Stanford University Committee on Graduate Studies

Abstract

Neural networks (NNs) are often used as surrogates or emulators of partial differential equations (PDEs) that describe the dynamics of complex systems. A virtually negligible computational cost of such surrogates renders them an attractive tool for ensemble-based computation, which requires a large number of repeated PDE solves. Since the latter are also needed to generate sufficient data for NN training, the usefulness of NN-based surrogates hinges on the balance between the training cost and the computational gain stemming from their deployment.

To reduce this cost, we propose to use multifidelity simulations in order to increase the amount of data one can generate during the allocated computing time. High-, and low-fidelity images are generated by solving PDEs on fine and coarse meshes, respectively; these multifidelity data are then patched together in the process of training a deep convolutional NN (CNN) using transfer learning. This strategy is further generalized to incorporate three levels of multifidelity data. We use theoretical results for multilevel Monte Carlo to guide our choice of the numbers of simulations (and resultant images) of each kind. This multifidelity training is used to estimate the distribution of a quantity of interest, whose dynamics is governed by a system of nonlinear PDEs with uncertain/random coefficients (e.g., parabolic PDEs governing the dynamics of multiphase flow in heterogeneous porous media). Our numerical experiments demonstrate that a mixture of a comparatively large amount of low-fidelity data and a much smaller amount of high-fidelity data provides an optimal balance between computational speedup and prediction accuracy. When used in the context of uncertainty

quantification, our multifidelity strategy is several orders of magnitude faster than either CNN training on high-fidelity images only or Monte Carlo solution of the PDEs. This computational speedup is achieved while preserving the accuracy of the estimators of the distributions of the quantities of interest, as expressed in terms of both the Wasserstein distance and the Kullback–Leibler divergence.

To further reduce the cost of data generation, we demonstrate that one can start the CNN training for a new task (a given set of PDEs describing, e.g., reactive transport) from a CNN that was originally trained for a different task (another set of PDEs, e.g., for multiphase flow). Our numerical experiments show that this transfer learning approach yields a considerable speedup when applied to the problem of the estimation of the distribution of a quantity of interest, whose dynamics is prescribed by a system of nonlinear PDEs for advection-dispersion transport in porous media with uncertain/random conductivity field. For a given amount of training data, the method has equal or greater prediction accuracy and generalizability to unseen inputs than a CNN whose training is initialized randomly.

Acknowledgments

These last five years have undoubtedly been my favorite. I would like to thank the following people for making everything happen:

- My family Chung R. Song, Mae H. Kim-Song, and Dong K. Song for the unconditional love and continued support in everything.
- Professor Daniel M. Tartakovsky for his research guidance, help, openness, and patience. Thank you for always making time including answering random questions past 11PM. Daniel's guidance was extremely inspirational in pushing through these weird pandemic times.
- Professor Roland N. Horne for his continued interest in my research and invitations to speak at SUPRI-D meetings. Thank you for all the wonderful academic and nonacademic chats at the Friday socials.
- Professor Anthony R. Kovscek for his interest in my research. His thermodynamics class was one of my first introductions into coding; it has greatly impacted my Stanford research trajectory.
- Simona Onori for her interest in my research and graciously agreeing to be on my PhD defense committee.
- Matthias Ihme for his interest in my research and graciously agreeing to chair my PhD defense.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Machine learning techniques, especially neural networks (NNs), have affected every facet of human activity and has permeated into the field of scientific computing. In the latter setting, NNs are used to approximate highly nonlinear and irregular functions (Friedman et al., 2001), solve (ordinary and partial) differential equations (e.g., Lee and Kang, 1990; Lagaris et al., 1998; Fuks and Tchelepi, 2020, among many others), and construct computationally cheap surrogates for ensemble-based computation (e.g., Mo et al., 2019a; Raissi et al., 2019a). Examples of the latter include inverse modeling (Mo et al., 2019b; Zhou and Tartakovsky, 2021), data assimilation (Tang et al., 2020), and uncertainty quantification (UQ) (Tripathy and Bilionis, 2018; Zhu et al., 2019).

A typical ensemble-based computation involves repeated solves of (coupled, nonlinear) partial-differential equations (PDEs),

$$\mathcal{N}(\mathbf{u};\boldsymbol{\theta}) = g(\mathbf{x},t;\boldsymbol{\theta}), \qquad (\mathbf{x},t) \in D \times (0,T], \tag{1.1}$$

which describe the spatiotemporal evolution of (a set of) state variables $\mathbf{u}(\mathbf{x},t)$ in the computational domain D over the simulation time horizon $(0,T]$. Multiple solves of Equation (1.1)—for different values of the inputs $\boldsymbol{\theta}(\mathbf{x},t)$ that parameterize the differential operator $\mathcal{N}$, the source function g,

and auxiliary functions in the initial and/or boundary conditions—are required because these values are known at best in terms of their distributions, which are either inferred from data or provided by the expert. High computational cost of solving Equation (1.1) numerically often precludes one from generating enough samples to obtain meaningful statistics of $\mathbf{u}(\mathbf{x}, t)$ or the derived quantities of interest. A surrogate of Equation (1.1) carries a negligible cost, making possible ensemble-based computation with arbitrarily small sampling error.

Alternative strategies for surrogate construction include polynomial chaos expansions (Xiu, 2010), Kriging or Gaussian processes (Couckuyt et al., 2014), polynomial regression (Montgomery and Evans, 2018), tensor-product splines (Hwang and Martins, 2018) and random forests (Breiman, 2001). Current popularity of NN-based surrogates (Mo et al., 2019a; Raissi et al., 2019a) is grounded in the scalability and approximation capabilities of deep NNs (Friedman et al., 2001; Tripathy and Bilionis, 2018). Regardless of the surrogate type, the training of a surrogate requires a large number of solves of Equation (1.1) for different combinations of parameter values $\boldsymbol{\theta}$. Advanced computer architectures, e.g., CUDA-compatible graphics processing units (GPUs) and tensor processing units (TPUs), are almost a necessity to train a large NN.

The combined cost of training-data acquisition and NN training can be so large as to negate the benefits of the NN surrogate. This observation suggests that the practical utility of a NN as a surrogate model hinges on one's ability to reduce dramatically the cost of its construction. In this thesis, we introduce two strategies to achieve this overarching goal. The first relies on multifidelity simulations to reduce the cost of data generation for subsequent training of a deep convolutional NN (CNN) using transfer learning. The second is to upcycle pretrained models, a process in which the CNN training for a new task (a given set of PDEs) starts from a CNN that was originally trained for a different task (another set of PDEs). These two complementary strategies are presented in Chapters 2 and 3 and Chapter 4, respectively.

To fix the basic ideas undergirding this research, we start by providing a brief overview of NNs, their use as surrogates of PDE-based models (Section 1.1), and transfer learning techniques for NN

training (Section 1.2).

1.1 Neural Networks

NNs are universal function approximators (Cybenko, 1989; Heinecke et al., 2020; Zhou, 2020). If one thinks of a a PDE-based model like Equation (1.1) as a function that maps the input $\boldsymbol{\theta}$ onto output $\mathbf{u}$, a NN can be used to approximate this function, resulting in a surrogate for the PDE. In order to gain insight into NN architecture selection for surrogate modeling, it is important to understand the inner workings of a NN. We start with a linear input-output relation without bias,

$$\hat{\mathbf{u}} = \mathbf{w}\boldsymbol{\theta}, \tag{1.2}$$

where $\mathbf{w}$ is an $N_{\hat{u}} \times N_{\boldsymbol{\theta}}$ matrix of weights; and $N_{\hat{u}}$ and $N_{\boldsymbol{\theta}}$ are the number of elements in the output and the number of elements in the input, respectively. An optimal set of weights is obtained by minimizing the discrepancy between $\mathbf{u}$ and $\hat{\mathbf{u}}$.

Even after optimization, the linear model in (1.2) is unlikely to represent accurately the nonlinear input-output relationship encoded in Equation (1.1). To handle nonlinearity one introduces a nonlinear operator $\sigma(\cdot)$, which operates on each element of the vector $\mathbf{w}\boldsymbol{\theta}$. Popular variants of this so-called activation function include the rectified linear unit (ReLU),

$$\sigma_{\mathrm{ReLU}}(x) = \max(0, x), \tag{1.3a}$$

and the sigmoid function

$$\sigma_{\mathrm{sigmoid}}(x) = \frac{1}{1 + e^{-x}}. \tag{1.3b}$$

With the addition of the activation function, the linear function approximation (1.2) is replaced with its nonlinear counterpart,

$$\hat{\mathbf{u}} = \sigma(\mathbf{w}\boldsymbol{\theta}) \equiv (\sigma \circ \mathbf{w})(\boldsymbol{\theta}). \tag{1.4}$$

This expression is referred to as a "layer" in machine learning jargon and provides the simplest representation of a NN.

A "deep NN" is constructed by the repeated application of the activation function, resulting in multiple (N) layers. The standard convention for layer counting does not count the input while the output is counted as one of the N layers. For example, a four-layer deep NN, $\mathbf{D}_{4-\text{layer}}$, is defined as

$$\hat{u} = \mathbf{D}_{3-\text{layer}}(\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3; \boldsymbol{\theta}) = \sigma(\mathbf{w}_3\sigma(\mathbf{w}_2\sigma(\mathbf{w}_1\boldsymbol{\theta}))) \equiv (\sigma \circ \mathbf{w}_3)(\sigma \circ \mathbf{w}_2)(\sigma \circ \mathbf{w}_1)(\boldsymbol{\theta}). \tag{1.5}$$

Thus, a deep NN with N_L-layers, $\mathbf{D}(\mathbf{W}; \boldsymbol{\theta})$, approximates the model output $\mathbf{u}$ by

$$\hat{u} = \mathbf{D}(\mathbf{W}; \boldsymbol{\theta}) = (\sigma \circ \mathbf{w}_{N_L-1}) \dots (\sigma \circ \mathbf{w}_1)(\boldsymbol{\theta}), \tag{1.6}$$

where $\mathbf{W} = \{\mathbf{w}_1, \dots, \mathbf{w}_{N_L}\}$; the weight matrix $\mathbf{w}_1$ has dimensions $d_1 \times N_{\boldsymbol{\theta}}$, the weight matrix $\mathbf{w}_2$ dimensions $d_1 \times d_2$, ..., and the weight matrix $\mathbf{w}_{N_{L-1}}$ dimensions $N_{\boldsymbol{\theta}} \times d_{N_L-2}$; and d_i ($i = 1, \dots, N_L - 2$) is the number of "neurons" in the ith layer. Deep NNs are interchangeably referred to as "artificial NNs" or "multilayer perceptrons".

1.1.1 Convolutional Neural Networks

As the number of layers, N_L, and the number of neurons in each layer, d_i, increase, the number of elements within the NN $\mathbf{D}$, $N_{\mathbf{W}}$, grows multiplicatively and can become prohibitively large for realistic finite computing-memory resources. CNNs (Goodfellow et al., 2013) use a convolution operator, which maintains the spatial relationship in the input, to reduce $N_{\mathbf{W}}$. CNNs are widely used in many image-related problems ranging from image recognition (Krizhevsky et al., 2012; Simonyan and Zisserman, 2014; Szegedy et al., 2015; He et al., 2016) to image-to-image regression (Mo et al., 2019b,a; Ronneberger et al., 2015; Zhou and Tartakovsky, 2021).

The core of a CNN is the convolutional layer which enables the CNN to perform computations

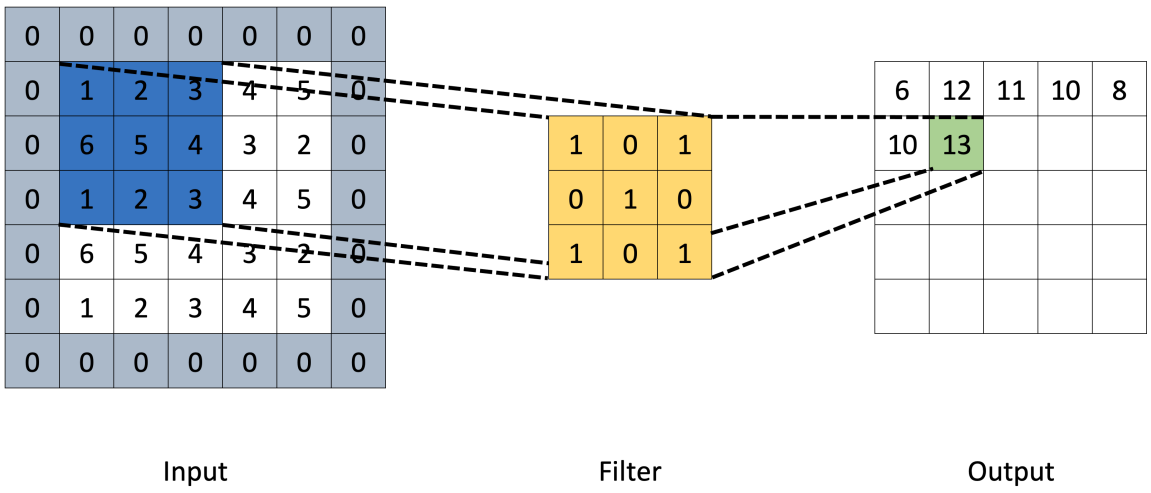

Figure 1.1: Example of a convolutional filter in mid computation. The input is a 5×5 matrix with a padding of one (indicated by the grey zeros surrounding the input). The convolutional filter is a 3×3 matrix and the resulting output would be a 5×5 matrix. The convolutional filter (yellow) scans a portion of the input data (blue) and produces parts of the output (green). The stride used in this example is one.

while maintaining spatial relationships in the input and output. The convolutional filter, also commonly referred to as a kernel, is composed of trainable weights. The dimensions of the filter are specified by "filter size". During computation, the filter slides across the height and width of the two-dimensional input while performing dot product between itself and the scanned portion of the input. The sliding of the filter between each dot product operation is specified by "slide length". The input data is often subject to a "padding" or "zero-padding", surrounding the input volume with zeros around the border, to accommodate the filter size and control the dimensions of the output. An example of this process is displayed in Figure 1.1.

Much like regular NNs, CNNs alternate between convolutional filters and nonlinear operators, such as the sigmoid function and ReLU, to perform complex tasks. It is common practice to form "blocks" or consecutive layers of the same input/output dimensions to perform tasks. As the CNN becomes deeper, and the architecture requires for more layers, we may run into the problem of vanishing gradients. One way to address this problem is to form dense blocks, within each of which the later layers form connections to early layers to facilitate more effective training.

In practice, odd filter sizes are preferred as even filter sizes cause asymmetry in how much of the padding is reflected in the output. The construction of blocks calls for the selection of an odd filter size F, which determines the padding P to be

$$P = \frac{F-1}{2}. \tag{1.7}$$

Assuming that we want a symmetric padding representation, for each dimension, we calculate the output size W_{out},

$$W_{\text{out}} = \frac{W - F - 2P}{S} + 1, \tag{1.8}$$

if we know the input size W_{in}, filter size F, padding P, and stride length S. While square data and filter dimensions are popular, they are not a requirement. Since equation (1.8) holds true for each dimension, one can have different input size W_{in}, filter size F, padding P, and stride length S in different directions.

In contrast to blocks, a CNN may benefit from reducing or increasing the size of the output volume. Two popular methods of reducing the volume of the output are pooling layers and using stride lengths greater than one. A pooling layer is a convolutional operation that examines an area of the input volume and allows only limited information to pass trough. An example of a pooling layer is the max-pool, which scans an area of the input and allows only the maximum value of the scanned area to pass through. A stride length of more than one reduces the output size by a factor of the stride length. To increase the size of the output volume, a two-dimensional transposed convolution is commonly used; a typical volume inside of a CNN operating on two-dimensional data has three characteristics: channels, length, and width. The channels are determined by the number of convolutional filters the input is subjected to. The two-dimensional transposed convolution collapses the number of output channels, compared to the number of input channels, while increasing the length and width of the output.

1.1.2 Neural Network Training Strategy

The specific training strategies associated with each method in this thesis are included in their respective chapter. However, the training of a NN via stochastic gradient descent (SGD) requires the following common steps:

1. Evaluate the loss of a model based on a forward pass
2. Calculate the derivatives of the weights with respect to the loss
3. Update weights based on the derivatives

Loss functions enable NN training to be posed as an optimization problem in which the weights are modified to minimize a loss function. The loss function often takes the following form:

$$\text{Loss} = f(\mathbf{u}, \hat{\mathbf{u}}) + R||\mathbf{W}||. \tag{1.9}$$

Popular loss terms associated with the discrepancies between the data and their NN representation, $f(\mathbf{u}, \hat{\mathbf{u}})$, include root mean square error (RMSE),

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\mathbf{u}_i - \hat{\mathbf{u}}_i)^2}, \tag{1.10a}$$

and mean absolute error (MAE),

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|\mathbf{u}_i - \hat{\mathbf{u}}_i|, \tag{1.10b}$$

where N is the number of data points used for the NN training. The loss associated with the magnitude of the weights, $R||\mathbf{W}||$, is called the regularization term. The regularization term penalizes large weights based on R and the norm taken to calculate the regularization term. The `PyTorch` implementation of the weight decay coefficient R is controlled through weight decay ψ.

Once the loss is calculated, the derivatives are computed using back propagation. Then the

weights $\mathbf{W}$ are updated according to a variant of SGD. While there are many variants of SGD, the original formulation of GD captures many of the core ideas behind NN training. The GD procedure to update the weights of a NN is

$$\mathbf{W}_{i+1} = \mathbf{W}_i - \eta \nabla_{\mathbf{W_i}} \text{Loss}(\mathbf{W}_i, \mathbf{u}_i, \hat{\mathbf{u}}_i), \tag{1.11}$$

where the index i organizes training progression and η refers to the learning rate. When index i reaches the number of training data N_train, when all of the training data has been used by the training process once, we say that the training process has gone through one epoch. The selection of learning rate η is very important: if η is too small, the training will be slow with a high risk of getting stuck in local optima; too big and the training may skip the optimum. To address this issue, one employs a training scheduler.

A scheduler changes various training parameters during training. It can enable the training procedure to start with large learning rates to quickly get near the optimum then reduce the learning rate to fine-tune the training process. The change of learning rate is prescribed by factor γ, which changes the learning rate according to

$$\eta_{j+1} = \gamma \eta_j, \tag{1.12}$$

where j is an index marking the progression of the scheduler updates. Common scheduler progression schemes include reducing learning rate based on epoch and reducing learning rate when the training error plateaus. When setting up a scheduler, setting a minimum learning rate is often a good idea as a learning rate that is too small can functionally halt the training process.

1.1.3 Neural Networks as Surrogate Models

The use of a NN as a surrogate for the PDE-based model implies the availability of domain-specific knowledge about the system under investigation; such knowledge (e.g., conservation laws) is encapsulated in the PDE formulation. Some of the NN-based surrogate modeling techniques are discussed

below; the presentation follows the original citation, while retaining our nomenclature to maintain consistency within this thesis.

Physics Informed Learning/Physics Informed Neural Network

Physics-informed learning (Raissi et al., 2021) is a popular framework to train surrogate models. It has been applied to Gaussian process regression (Raissi et al., 2017, 2018) and physics-informed neural networks (PINNs) (Raissi et al., 2019b; Cai et al., 2022). PINNs have been applied to incompressible (Jin et al., 2021; Raissi et al., 2020) and compressible (Mao et al., 2020) fluid flows. PINNs use domain knowledge (known PDEs) to formulate the minimization problem used to determine the weights $\mathbf{W}$ in Equation (1.6); it can be used with any NN architecture.

Specifically, it is used to formulate a loss function that quantifies the discrepancy between the data (PDE solution) $\mathbf{u}$ and its NN approximation $\hat{\mathbf{u}}$. Common loss functions include RMSE and MAE. In a PINN, such loss functions are augmented with additional terms:

$$\text{Loss} = \text{Loss}_{\text{data}} + \text{Loss}_{\text{constrain}} + \text{Loss}_{\text{PDE}}. \tag{1.13}$$

The term $\text{Loss}_{\text{data}}$ = RMSE or MAE accounts for the discrepancy between the data $\mathbf{u}$ and observations or the model prediction $\hat{\mathbf{u}}$; the term $\text{Loss}_{\text{constrain}}$ enforces the physical constraints, e.g., positivity of the state variables and the monotonicity of certain relationships between the state variables; and the term Loss_{PDE} accounts for the known physics encapsulated in the PDE that was used to generate the data.

By embedding the domain knowledge in the loss function formulation, PINNs might achieve higher prediction accuracy then their purely data-driven NN counterparts in Equation (1.10). However, the PINN workflow is ineffective if little is known about the data set, or if the loss function terms are difficult to calculate. The quality of a PINN-based surrogate depends crucially on the amount of data one can afford to generate by running the PDE-based simulation for different combinations of the input parameters, i.e., on N in Equation (1.10).

Deep Convolutional Encoder-Decoder Network

The deep convolutional encoder-decoder network (Zhu and Zabaras, 2018), a CNN variant, is an architecture that often yields accurate surrogates for various fluid flow problems, e.g., single- (Mo et al., 2019a) and multi-phase (Mo et al., 2019b) flows and reactive solute transport (Zhou and Tartakovsky, 2021) in porous media, and performed various ensemble calculation tasks, e.g., uncertainty quantification (Mo et al., 2019a,b) and parameter estimation (Zhou and Tartakovsky, 2021). Convolutional encoder-decoder and the dense layers are two key devices within the model architecture which assist in the surrogate model construction.

The convolutional encoder-decoder architecture's first layers operate on the input dimensions. Subsequent layers "encode" to function on a smaller dimension; this encoded state is also called the latent state and thought to be a low-dimensional representation of the original inputs. Then the encoded latent layer is decoded to function at the dimensions of the outputs. The encoder-decoder is a dimensionality-reduction technique that helps mitigate the "curse of dimensionality"; the latter is a primary reason for the failure of many surrogate methods when high-dimensional inputs are trained on limited data (Lin and Tartakovsky, 2009). The exact size, number of layers used by the CNN, is determined by two factors: initial features and growth rate. The effect of network size is investigated in Appendix A.

The CNN layers are organized in dense blocks (Huang et al., 2017). Each dense block contains connections between nonadjacent layers which enhance the flow of information during training. These dense blocks differentiate the CNN from other encoder-decoders such as the one found in UNet (Ronneberger et al., 2015).

As the CNN encodes and decodes, the intermediate and latent dimensions can be specified. Multifidelity data, e.g., data generated by solving a PDE on grids with different spatiotemporal resolutions, can be generated to match the dimensions of the intermediate and latent variables. We choose the CNN architecture because of its prescribable intermediate and latent dimensions along

with the two devices mentioned above.

1.2 Transfer Learning

Transfer learning (Donahue et al., 2014) is the process of taking a model trained on one task and applying it to another (remotely) related task. Starting with a pretrained model, the process of transfer learning proceeds as follows:

1. Lock, i.e., prevent from updating during training, most of the weights and only allow a few layers near the output to update;
2. Train the updatable weights on the new data;
3. Unlock the entire model and allow all weights to update during training;
4. Fine-tune the entire NN on the new data.

The intuition behind initially locking the weights closer to the input is to allow the pretrained weights to identify key relationships in the data and not over-train to the new data. In computer-vision applications, transfer learning is the norm; such applications include generation of image description (Karpathy and Fei-Fei, 2015), face detection (Jiang and Learned-Miller, 2017), and construction of PINNs (Haghighat et al., 2021). The detailed methodology on how to apply transfer learning is described in each Chapter.

1.3 Dissertation Overview

A CNN capable of making predictions for realizations of $\boldsymbol{\theta}(\mathbf{x})$ outside of the training data-set requires a large number N_{train} of PDE solves; e.g., $N_{\text{train}} \sim 1500$ was used by Zhou and Tartakovsky (2021) and Mo et al. (2019a) to train encoder-decoder CNNs similar to ours. For situations where each forward PDE solve is computationally expensive, the computational cost associated with large N_{train} can be so expensive as to render the CNN training unfeasible. To address this problem, we implement

transfer learning in different ways: using multiresolution/multifidelity data and starting from a model pretrained on a different application.

The idea of using multifidelity data in the context of NNs is not unique to this study. For example, Geneva and Zabaras (2020) trained surrogate models using low-fidelity simulations as a conditional input. Meng and Karniadakis (2020) built fully-connected NN surrogates by training different networks to handle the low- and high-fidelity data. The novelty of our approach is to deploy transfer learning on multifidelity data to train different parts of a single network. This thesis places a focus on minimizing the computational cost of the generation of data needed to train the model as this directly increases the utility of such surrogate models. The value of the trained surrogate model is also evaluated through a UQ task where distributions are generated using the surrogate models for a far lower data generation cost than the computational costs associated with a comparable Monte Carlo (MC) simulation.

This dissertation is organized in the following way. In Chapter 2, we present a framework to train a CNN surrogate on multifidelity data using transfer learning. The CNN is able to accurately predict multiphase flow solution on a fine mesh, while being trained on two levels of data: coarse and fine data. The accuracy and training data requirements of the CNN trained on multifidelity data are compared to those of CNNs trained on only fine or coarse data. The CNN trained on multifidelity data is also used for uncertainty quantification, and the results are compared with traditional MC simulations. This work has been published in the *Journal of Machine Learning for Modeling and Computing* (Song and Tartakovsky, 2021).

In Chapter 3, we expand the technique developed in Chapter 2 to accommodate three levels of training data. This further reduces the data generation cost. The CNN trained on multifidelity data is again compared to CNNs trained on single-resolution data. Furthermore, the performance of the CNN surrogate for uncertainty quantification is evaluated.

Chapter 4 presents a framework to effectively retrain a CNN model. We take a CNN previously trained to solve a multiphase problem (CNN from Chapter 4) and apply concepts from transfer

learning to retrain the model to solve an advection-dispersion transport problem. The data requirement and accuracy of this technique are evaluated. Also, the UQ performance of this method is benchmarked against MC Simulations.

Chapter 2

Training on Two Levels of Data

Neural networks (NNs) are often used as surrogates or emulators of partial differential equations (PDEs) that describe the dynamics of complex systems. A virtually negligible computational cost of such surrogates renders them an attractive tool for ensemble-based computation, which requires a large number of repeated PDE solves. Because the latter are also needed to generate sufficient data for NN training, the usefulness of NN-based surrogates hinges on the balance between the training cost and the computational gain stemming from their deployment. We rely on multifidelity simulations to reduce the cost of data generation for subsequent training of a deep convolutional NN (CNN) using transfer learning. Two sets of high- and low-fidelity images are generated by solving PDEs on fine and coarse meshes, respectively. We use theoretical results for multilevel Monte Carlo (MLMC) to guide our choice of the numbers of images of each kind. We demonstrate the performance of this multifidelity training strategy on the problem of estimation of the distribution of a quantity of interest, whose dynamics is governed by a system of nonlinear PDEs (parabolic PDEs of multiphase flow in heterogeneous porous media) with uncertain/random parameters. Our numerical experiments demonstrate that a mixture of a comparatively large number of low-fidelity data and smaller numbers of high- and low-fidelity data provides an optimal balance of computational

speed-up and prediction accuracy. The former is reported relative to both CNN training on high-fidelity images only and MC simulation solution of the PDEs. The latter is expressed in terms of both the Wasserstein distance and the Kullback–Leibler divergence.

2.1 Introduction

The combined cost of training-data acquisition and NN training can be so large as to negate the benefits of the NN. This observation suggests that the practical utility of a NN as a surrogate model hinges on one's ability to dramatically reduce the cost of its construction. We rely on multi-fidelity simulations to reduce the cost of data generation for subsequent training of a deep convolutional NN (CNN) using transfer learning. High- and low-fidelity images are generated by solving the PDE described by Equation (1.1) on fine and coarse meshes, respectively. A fine mesh is defined by the need to resolve the spatiotemporal variability of the model's inputs $\boldsymbol{\theta}$ and outputs $\mathbf{u}$; the resulting high-fidelity simulation carries a high computational cost. Lower-fidelity solutions of Equation (1.1), obtained on coarser meshes on which appropriately homogenized inputs $\boldsymbol{\theta}_{\text{hom}}$ are defined, are cheaper to compute but less accurate. We train a CNN on a mixture of these multifidelity data, using the theoretical results for MLMC (Heinrich, 1998, 2001; Giles, 2008; Taverniers et al., 2020) to guide our choice of the numbers of solutions $\mathbf{u}(\mathbf{x}, t)$ of each kind. Similar to MLMC (Müller et al., 2013; Peherstorfer, 2019), the varying fidelity (also referred as "levels") of predictions of $\mathbf{u}$ can be achieved not only by solving Equation (1.1) on different meshes, but also by replacing Equation (1.1) with its cheaper-to-compute counterparts. For example, the multiphase flow equations used as the computational testbed in this study can be replaced with the cheaper-to-solve Richards equation and Green-Ampt equation (Yang et al., 2020; Sinsbeck and Tartakovsky, 2015), each of which encapsulates progressively simplified physics. We leave this aspect of NN training on multifidelity data for a follow-up study.

The idea of using multifidelity data in the context of NNs is not unique to this study. For example, Geneva and Zabaras (2020) trained surrogate models using low-fidelity simulations as a

conditional input. Meng and Karniadakis (2020) built fully-connected NN surrogates by training different networks to handle the low- and-high fidelity data. The novelty of our approach is to deploy transfer learning on multifidelity data to train different parts of a single network.

Section 2.2 contains a brief description of our CNN and the workflow for its training on multifidelity of data. The performance of this algorithm is tested on a system of nonlinear parabolic PDEs governing multiphase flow in a heterogeneous porous medium with uncertain properties, which are formulated in Section 2.3. In Section 2.4, we demonstrate the accuracy and computational efficiency of the CNN-based surrogate used to quantify predictive uncertainty of Equation (1.1) in terms of the distribution of a quantity of interest. Main conclusions drawn from this study are presented in Section 2.5.

2.2 Deep Convolutional Neural Networks

The CNN-based surrogate is set up as an image-to-image regression model (Zhou and Tartakovsky, 2021). To train and test the network, we use the parameter values $\boldsymbol{\theta}(\mathbf{x}_i)$ in N_{el} elements $\{\mathbf{x}_i\}_{i=1}^{N_{\text{el}}}$ of a numerical grid as input and the discretized solution $\mathbf{u}(\mathbf{x}_i, t_k)$ of the PDE described by Equation (1.1) at N_{ts} time steps $\{t_k\}_{k=1}^{N_{\text{ts}}}$ as output. To facilitate the generalizibility of the trained CNN to unseen sets of the input $\boldsymbol{\theta}(\mathbf{x}_i)$, i.e., to ensure that the CNN is not over-fitted to a particular choice of $\boldsymbol{\theta}(\mathbf{x}_i)$, the training data comprises a large number N_{train} of the solutions $\mathbf{u}$ obtained for N_{train} realizations $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{N_{\text{train}}}\}$ of the input $\boldsymbol{\theta}$. The loss function,

$$\mathbf{L}(\mathbf{w}) = \sum_{m=1}^{N_{\text{train}}} \sum_{i=1}^{N_{\text{el}}} \sum_{k=1}^{N_{\text{ts}}} |\mathbf{u}(\mathbf{x}_i, t_k; \boldsymbol{\theta}_m) - \hat{\mathbf{u}}_{ik}(\mathbf{w}; \boldsymbol{\theta}_m)| + R \sum_{n=1}^{N_{\text{w}}} w_n^2 \tag{2.1}$$

consists of two parts. The first represents the L_1-norm discrepancy between the state variables $\mathbf{u}$ predicted by solving the PDE described by Equation (1.1), $\mathbf{u}(\mathbf{x}_i, t_k)$ and estimated by the CNN, $\hat{\mathbf{u}}_{ik}(\mathbf{w})$, with N_{w} weights $\mathbf{w} = (w_1, \ldots, w_{N_{\text{w}}})^\top$. The L_2-norm regularization prevents over-fitting by penalizing large weights $\mathbf{w}$ associated complex models; the regularization parameter R determines

how much regularization penalty is applied. The CNN training consists of finding a set of weights $\mathbf{w}^\star$ that minimizes $\mathbf{L}$.

2.2.1 Transfer Learning

Transfer learning has been implemented for face detection (Jiang and Learned-Miller, 2017), generation of image description (Karpathy and Fei-Fei, 2015), and construction of physics-informed NNs (Haghighat et al., 2021), among other applications.

Let HFS and LFS data refer to the solutions of Equation (1.1), $\mathbf{u}(\mathbf{x}_i, t_k)$, obtained on the fine ($N_{\text{el}} = N_{\text{el}}^{\text{HFS}}$) and coarse ($N_{\text{el}} = N_{\text{el}}^{\text{LFS}}$ with $N_{\text{el}}^{\text{LFS}} < N_{\text{el}}^{\text{HFS}}$) meshes, respectively. If N_{w} in Equation (2.1) denotes the number of weights in the CNN trained on the HFS data, then our implementation of transfer learning starts with the construction of a CNN composed of N_{LFS} ($N_{\text{LFS}} < N_{\text{w}}$) weights $\mathbf{w}_{\text{LFS}} = (w_1, \ldots, w_{N_{\text{LFS}}})^\top$ trained on the LFS data. Then, the HFS data are used to train the desired high-resolution CNN, i.e., to determine the remaining weights $\mathbf{w}_{\text{HFS}} = (w_{N_{\text{LFS}}+1}, \ldots, w_{N_{\text{w}}})^\top$. This transfer learning strategy is depicted in Figure 2.1 and detailed below.

2.2.2 Workflow for CNN Training on Multifidelity Data

Our strategy for CNN training on multifidelity data consists of three phases (Figure 2.1), each of which results in a CNN denoted by M_i ($i = 1, 2, 3$). During Phase 1, the CNN M_1 with the $N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}}$ output is trained on the LFS data. In Phase 2, the CNN M_2 with $N_{\text{el}}^{\text{HFS}} \times N_{\text{el}}^{\text{HFS}}$ output is constructed by adding an additional layer with the weights $\mathbf{w}_{\text{HFS}}$, which are trained on the HFS data while keeping the original weights $\mathbf{w}_{\text{LFS}}$ locked. Phase 3 consists of fine-tuning the CNN M_2 by allowing all the weights $\mathbf{w} = \{\mathbf{w}_{\text{LFS}}, \mathbf{w}_{\text{HFS}}\}$ to update during the training on the same HFS data. The numerical experiments reported in Sections 2.3 and 2.4 demonstrate that this transfer learning strategy significantly reduces the number of high-resolution PDE solves.

The workflow of our approach is provided below.

Phase 1: Train a CNN M_1, with $N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}}$ output, on the LFS data.

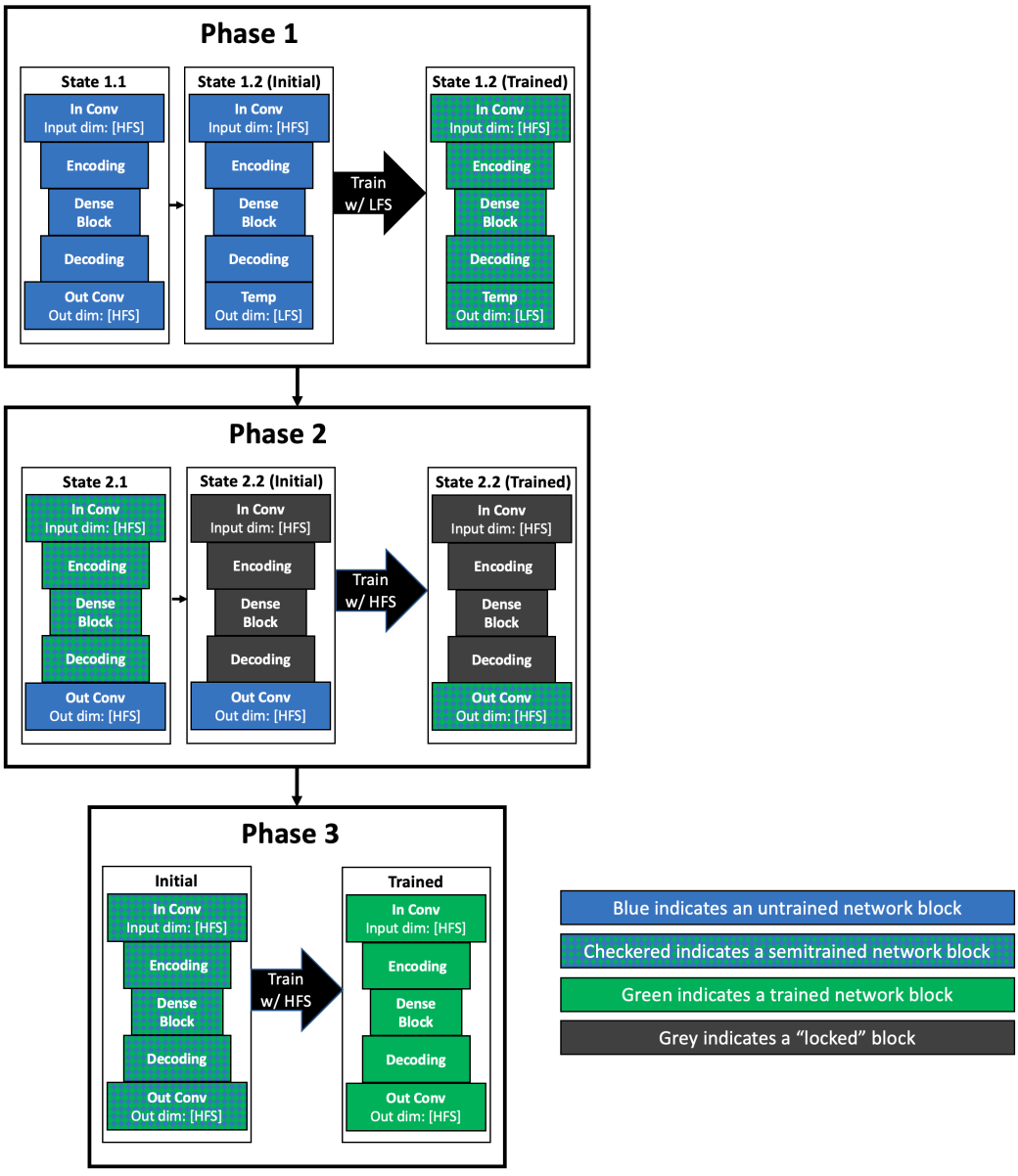

Figure 2.1: Workflow for CNN training on multifidelity data. Phase 1 returns a low-resolution CNN trained on the LFS data. Phase 2 supplements that network with an additional layer whose weights are determined from the HFS data, producing a high-resolution CNN. In Phase 3, the latter is fine-tuned by allowing all the weights to vary during the training on the same HFS data.

State 1.1: Initialize the transfer learning by employing the encoder-decoder CNN of Mo et al. (2019b), M_{init} with $N_{\text{el}}^{\text{HFS}} \times N_{\text{el}}^{\text{HFS}}$ output, whose N_{w} weights $\mathbf{w}$ are set to `PyTorch` defaults.

State 1.2: Train the CNN M_1 on the LFS data. The starting point is a CNN obtained from M_{init} by replacing its last layer L_{last}, which has N_{HFS} weights $\mathbf{w}_{\text{HFS}}$, with a temporary convolution layer L_{temp}. The latter is composed of weights $\mathbf{w}_{\text{temp}}$ and makes output of M_1 match the dimensions of the LFS data, $[N_{\text{ts}} \times N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}}]$. Then, the weights of M_1, $\mathbf{w}_{\text{phase1}} = \{\mathbf{w}_{\text{HFS}}, \mathbf{w}_{\text{temp}}\}$ are trained on the LFS data by minimizing (2.1).

Phase 2: Train a CNN M_2, with N_{w} weights $\mathbf{w} = \{\mathbf{w}_{\text{LFS}}, \mathbf{w}_{\text{HFS}}\}$ (of which N_{LFS} weights are locked) and $N_{\text{el}}^{\text{HFS}} \times N_{\text{el}}^{\text{HFS}}$ output, on the HFS data

State 2.1: Build a CNN from M_1 by replacing its layer L_{temp} with the layer L_{last} and discarding L_{temp}. The modified CNN has weights $\mathbf{w} = \{\mathbf{w}_{\text{LFS}}, \mathbf{w}_{\text{HFS}}\}$, among which weights $\mathbf{w}_{\text{LFS}}$ have been updated by data and $\mathbf{w}_{\text{HFS}}$ have not been updated by data.

State 2.2: Train the resulting CNN M_2 on the HFS data by minimizing Equation (2.1) over the weights $\mathbf{w}_{\text{HFS}}$ of layer L_{last}, while keeping the remaining weights $\mathbf{w}_{\text{LFS}}$ locked at their values in M_1.

Phase 3: Train a CNN M_3 on the HFS data by allowing all weights $\mathbf{w}$ of M_2 to vary during the minimization

Because the bulk of the CNN M_3 training is carried out on the LFS data, this procedure is more efficient than CNN training solely on HFS data. The predictive capability of the trained M_2 is only sometimes similar to that of the trained M_3; Phase 3 improves the performance if changes are needed in the layers which were locked during Phase 2.

2.3 Computational Example: Multiphase Flow

Numerical solution of problems involving multiphase flow in porous media is notoriously difficult because of the high degree of nonlinearity and stiffness of the governing PDEs. Each forward solve of these PDEs is so expensive that ensemble forward solves are uncommon, e.g., uncertainty quantification efforts in petroleum engineering have been based on as few as three model runs. This high cost and numerical complexity make the multiphase flow equations a challenging testbed for ensemble-based simulations.

We consider horizontal flow of two incompressible and immiscible fluids, with viscosities μ_1 and μ_2, in a heterogeneous, incompressible, and isotropic porous medium D. The latter is characterized by porosity ϕ and intrinsic permeability k. The viscosities of both phases are assumed to be constant $\mu_1 = \mu_2 = 1 \times 10^{-3}$ kg/(m·s). The porosity is assumed to be constant $\phi = 0.25$, and intrinsic permeability $k(\mathbf{x})$ is treated as a random variable. The time domain t is between zero and a specified terminal time T. Mass conservation of the ℓth fluid phase ($\ell = 1, 2$) implies

$$\phi \frac{\partial S_\ell}{\partial t} + \nabla \cdot \mathbf{v}_\ell + q_\ell = 0, \qquad \mathbf{x} \equiv (x_1, x_2)^\top \in D, \quad t \in [0, T], \tag{2.2a}$$

where $S_\ell(\mathbf{x}, t)$ is the phase saturation constrained by $S_1 + S_2 = 1$; q_ℓ is the source/sink term; and the macroscopic velocity $\mathbf{v}_\ell(\mathbf{x}, t)$ is described by the generalized Darcy law

$$\mathbf{v}_\ell = -k \frac{k_{r\ell}}{\mu_\ell} \nabla P_\ell. \tag{2.2b}$$

The relative permeability for the ℓth phase, $k_{r\ell}$, varies with the phase saturation, $k_{r\ell} = k_{r\ell}(S_\ell)$, in accordance with the Brooks-Corey constitutive model (Corey, 1954). The Brooks-Corey constitutive model prescribes relative permeability as:

$$k_{r\ell} = k_{r\ell,max} S_\ell^{N_\ell} \tag{2.3}$$

where $k_{r\ell,max}$ is the maximum relative permeability for the ℓth phase and N_ℓ are curve shape parameters used to define the relative permeability curves. The $k_{r\ell,max}$ and N_ℓ values used for this computational example are provided in Table 2.1.

Table 2.1: Variables used to determine relative permeability curves

Variable	Value
$k_{r1,max}$	1
$k_{r2,max}$	1
N_1	2
N_2	2.5

Following Taverniers et al. (2020) and many others, we neglect the capillary forces, i.e., assume pressure within the two phases to be equal, $P_1 = P_2 \equiv P(\mathbf{x}, t)$; that is a common assumption in applications to reservoir engineering and carbon sequestration. In this specific numerical example, the subscripts $\ell = 1$ and 2 represent water and oil respectively.

The two-dimensional computational spatial domain D is a 150 m $\times$ 150 m square (Figure 2.2) with the impermeable bottom (Γ_b or $x_2 = 0$) and top (Γ_t or $x_2 = 150$ m) boundaries; Dirichlet conditions are imposed along the left (Γ_l or $x_1 = 0$) and right (Γ_r or $x_1 = 150$ m) boundaries:

$$\frac{\partial P}{\partial x_2} = 0, \quad \mathbf{x} \in \Gamma_\mathrm{b} \cup \Gamma_\mathrm{t}; \quad P = 10.2 \;\; \& \;\; S_1 = 1.0, \quad \mathbf{x} \in \Gamma_\mathrm{l}; \quad P = 10.1, \quad \mathbf{x} \in \Gamma_\mathrm{r}; \tag{2.4a}$$

here and below, the pressure P is expressed in MPa. Initial conditions are

$$P(\mathbf{x}, 0) = 10.1, \qquad S_1(\mathbf{x}, 0) = 0, \qquad \mathbf{x} \in D. \tag{2.4b}$$

All the model parameters, except for the intrinsic permeability $k(\mathbf{x})$, are assumed to be constant and known with certainty. The uncertain permeability $k(\mathbf{x})$ is modeled as a second-order stationary random field, such that $Y(\mathbf{x}) = \ln k$ is multivariate Gaussian with mean $\langle Y \rangle = 0$, variance $\sigma_Y^2 = 2.0$, and an exponential two-point covariance $C(\mathbf{x}, \mathbf{y}) = \sigma_Y^2 \exp(-|\mathbf{x} - \mathbf{y}|/\lambda_Y)$. A correlation length of $\lambda_Y = 19$ m was used to generate one set of intrinsic permeability fields and a correlation length of

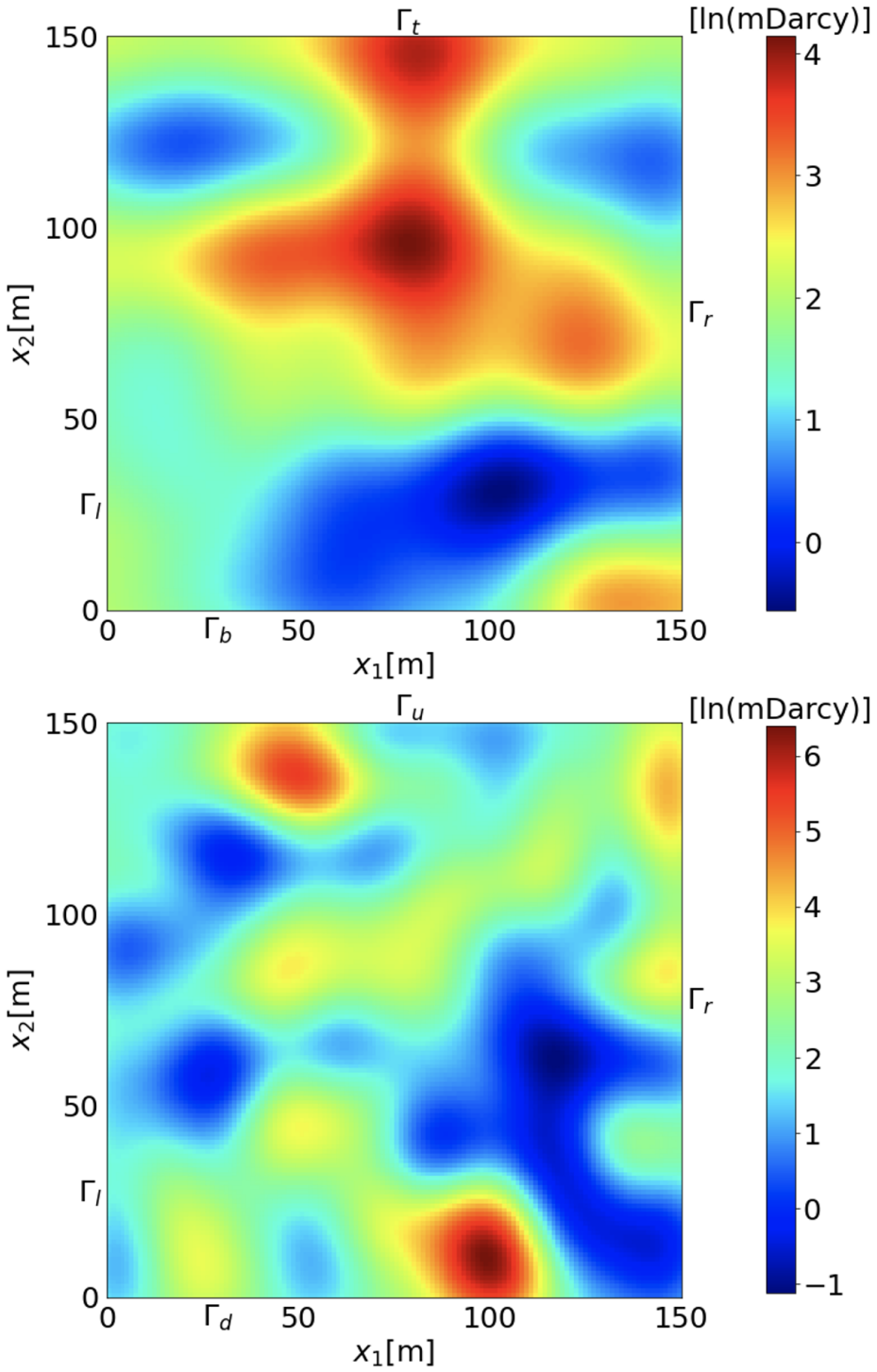

Figure 2.2: Representative realizations of log permeability fields $Y = \ln k$ on the 128×128 grid, which are used in high-fidelity simulations. A correlation length of $\lambda_Y = 19$ m was used to generate the top permeability field, and a correlation length of $\lambda_Y = 8$ m was used to generate the bottom permeability field. Permeability k is expressed in mDarcy.

$\lambda_Y = 8$ m was used to generate the other set of intrinsic permeability fields. We use a truncated Karhunen-Loéve expansion with $p = 31$ terms to represent $Y(\mathbf{x})$ (Taverniers et al., 2020). A representative realization of the resulting permeability field is shown in Figure 2.2 for the 128×128 mesh.

Equations (2.2)–(2.4) are approximated using a finite volume scheme in space and implicit Euler scheme in time, yielding a highly nonlinear algebraic system (Aziz, 1979). Adaptive time-stepping is implemented to advance the solution in time. At each time step, the nonlinear algebraic system is solved through Newton-Raphson (NR) iterations with the modified Appleyard update dampening (Appleyard et al., 1981) that improves the convergence of NR iterations by capping the maximum saturation update to a specified limit. For the νth iteration and the ith cell of volume V_i, the convergence criteria are:

$$\max_i \left| \Delta t \left(\frac{r_{\ell,i}}{\phi V_i} \right) \right| < \epsilon_1, \quad \max_i |P_i^{(\nu+1)} - P_i^{(\nu)}| < \epsilon_2, \quad \max_i |S_{\ell,i}^{(\nu+1)} - S_{\ell,i}^{(\nu)}| < \epsilon_3 \tag{2.5}$$

where $r_{\ell,i}$ is the residual of the mass balance of phase ℓ, Δt is the time step, the relative residual norm $\epsilon_1 = 10^{-6}$, the maximum pressure update $\epsilon_2 = 10^{-3}$, and the maximum saturation update $\epsilon_3 = 10^{-2}$.

2.3.1 Upscaling of Permeability

Multifidelity data are generated by solving Equation (2.2)–(2.4) on progressively coarsened grids: the 128×128 and 64×64 grids are used for HFS and LFS, respectively. When we use the correlation length of $\lambda_Y = 19$ m, the spatial discretizations of these HFS and LFS ($\Delta x = 1.17$ m and 2.34 m, respectively) are sufficient to capture the randomness in permeability fields. The latter rests on the "rule of thumb" requirement that Δx be such that $4\Delta x \leq \lambda_Y$, i.e., that a numerical mesh should have at least four elements of length Δx per correlation length λ_Y (e.g., Ye et al., 2004, and references therein). However when the correlation length of $\lambda_Y = 8$ m is used, only the HFS satisfies this requirement.

The grid coarsening must be accomplished by upscaling (coarsening) of the realizations of the random permeability $\hat{k}$ which are initially generated at the finest scale (Figure 2.2). Among alternative upscaling strategies (Paleologos et al., 1996; Tartakovsky and Neuman, 1998; Boso and Tartakovsky, 2018), we select the one proposed by Durlofsky (2005) because of its computational simplicity. This method turns a scalar permeability field defined on the fine (128×128) mesh into its upscaled tensorial (anisotropic) counterpart whose off-diagonal components are zero and the diagonal components are computed as the distance-weighted arithmetic mean perpendicular to the direction of flow and the distance-weighted harmonic mean in the direction of flow.

2.3.2 Data Acquisition

Multifidelity training data come in the form of $N_{\text{ts}} = 16$ temporal snapshots of the saturation $S_1(\mathbf{x}, t)$ computed by solving Equation (2.2)–(2.4) on the $N_{\text{el}} \times N_{\text{el}}$ grids with $N_{\text{el}} = 128 \equiv N_{\text{el}}^{\text{HFS}}$ and $64 \equiv N_{\text{el}}^{\text{LFS}}$. While two sets of data are generated for each of the correlation lengths, Figure 2.3 shows examples of such images, corresponding to the permeability field generated using the correlation length of $\lambda_Y = 19$ m in Figure 2.2. The permeability fields on the finest mesh, $[1 \times N_{\text{HFS}} \times N_{\text{HFS}}]$, are used as the input $\boldsymbol{\theta}$ for all CNNs. The size of of the CNN, $[N_{\text{ts}} \times N_{\text{el}} \times N_{\text{el}}]$, depends on the size of the training data.

The numerical solutions of Equation (2.2)–(2.4) are obtained using a `Matlab`-based multiphase flow simulator on a computer with an Intel Core i7-4790 3.6GHz processor and 64GB of RAM. The average computation time for each HFS data point is 219.13 sec and 37.13 sec for each LFS data point. These computation times were consistent for both data sets corresponding to different correlation lengths. The time needed to generate a data set is henceforth referred as "data-generation budget".

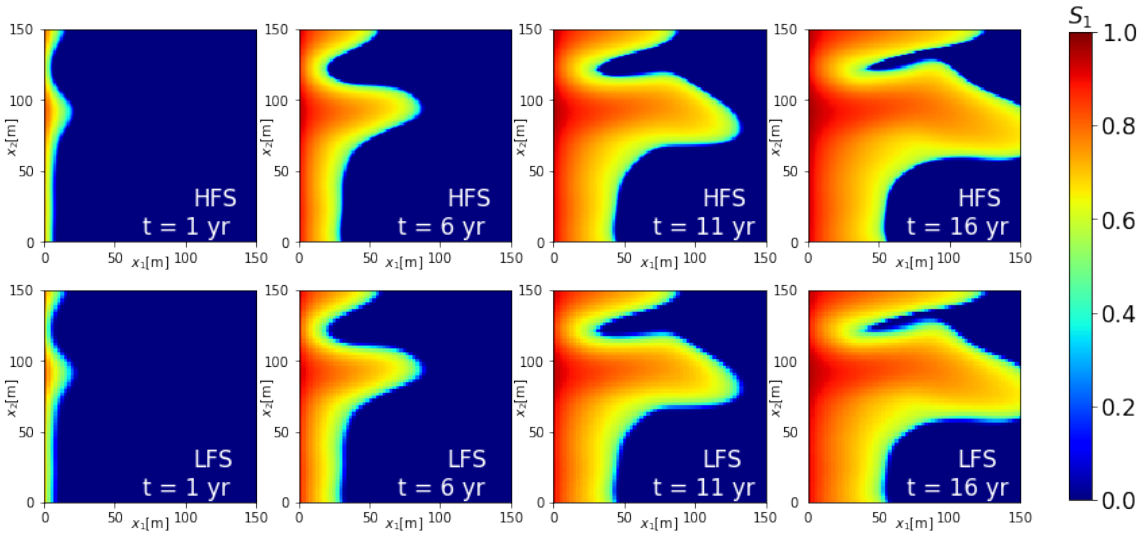

Figure 2.3: Temporal snapshots of saturation $S_1(\mathbf{x}, t)$ computed with HFS (top-row) and LFS (bottom-row) for the permeability field $k(\mathbf{x})$ generated using the correlation length of $\lambda_Y = 19$ m in Figure 2.2.

2.3.3 CNN Training

Table 2.2 describes the CNN architecture used in the implementation of our approach (see Figure 2.1). The model implementation and training is done using `PyTorch` and other open source packages. The computations were carried out on the Stanford Mazama high-performance computing cluster. The allocated computing resources include Intel Xenon Gold 6126 CPU (2.6 GHz), 60GB RAM, and Nvidia V100 GPU with 16GB vRAM. (Although available, multicores were not used for this work.)

The key hyperparameters affecting the CNN performance are the learning rate η, the weight decay ψ, the factor γ, and the minimum learning rate η_{min}. The η and ψ are parameters of the Adam optimizer (Kingma and Ba, 2014), and the γ and η_{min} are parameters of the `ReduceLROnPlateau` scheduler. The CNN training involves many more hyperparameters, but we use their default values in `PyTorch`. The regularization parameter R is specified through ψ following the implementation of Loshchilov and Hutter (2017). Further information on the hyperparameters, schedulers, and

Table 2.2: Model block description and the input and output dimensions of each model block. In our numerical experiments, the number of time steps is $N_{\text{ts}} = 16$; the number of elements in fine and coarse meshes is $N_{\text{el}}^{\text{HFS}} \times N_{\text{el}}^{\text{HFS}} = 128 \times 128$ and $N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}} = 64 \times 64$, respectively; the number of elements in the output of the dense block is $N_{\text{dense}} = 32$; and the number of channels in each of the seven layers of the CNN is $n_1 = 64$, $n_2 = 344$, $n_3 = 172$, $n_4 = 652$, $n_5 = 326$, $n_6 = 606$, and $n_7 = 303$.

Layer	Input	Output
Input: Permeability field k	$1 \times N_{\text{HFS}} \times N_{\text{HFS}}$	
Convolution 1	$n_1 \times N_{\text{el}}^{\text{HFS}} \times N_{\text{el}}^{\text{HFS}}$	$n_2 \times N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}}$
Dense Block (Encoding)	$n_2 \times N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}}$	$n_3 \times N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}}$
Convolution 2	$n_3 \times N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}}$	$n_4 \times N_{\text{dense}} \times N_{\text{dense}}$
Dense Block	$n_4 \times N_{\text{dense}} \times N_{\text{dense}}$	$n_5 \times N_{\text{dense}} \times N_{\text{dense}}$
Convolution Transpose 1	$n_5 \times N_{\text{dense}} \times N_{\text{dense}}$	$n_6 \times N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}}$
Dense Block (Decoding)	$n_6 \times N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}}$	$n_7 \times N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}}$
Convolution Transpose 2	$n_7 \times N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}}$	$N_{\text{ts}} \times N_{\text{el}}^{\text{HFS}} \times N_{\text{el}}^{\text{HFS}}$
Output: Saturation map $\hat{S}$	$N_{\text{ts}} \times N_{\text{el}}^{\text{HFS}} \times N_{\text{el}}^{\text{HFS}}$	

optimizers can be found in the `PyTorch` documentation (Paszke et al., 2019).

Table 2.3: Learning rates and epochs used at each phase.

	Learning rate	Epochs
Phase 1	5×10^{-3}	170
Phase 2	5×10^{-3}	150
Phase 3	10^{-4}	100

The hyperparameter search was conducted on the data generated using the correlation length of $\lambda_Y = 19$ m. The same hyperparameters were used for the experiments using the data generated using the correlation length of $\lambda_Y = 8$ m. The hyperparameters used by Mo et al. (2019b) in a similar CNN architecture serve as an initial guess for the hyperparameter optimization. The search iterated, in order, through variations of η, ψ, γ, and η_{min}. Once an acceptable value of a hyperparameter was found, the search moved to the next hyper parameter. A robust grid search may yield a more optimal set of hyper parameters. The search required 100 HFS, with each training pass taking about 0.65 hours to complete, when 200 epochs were used. It took 7.2 training-hours to find functional hyperparameters (12 training passes), and a considerably smaller wall-clock time because this task was parallelized across several GPU nodes. We selected the hyperparameter values yielding the smallest root mean square error (RMSE) on the HFS test data (Figure 2.4). These values are used

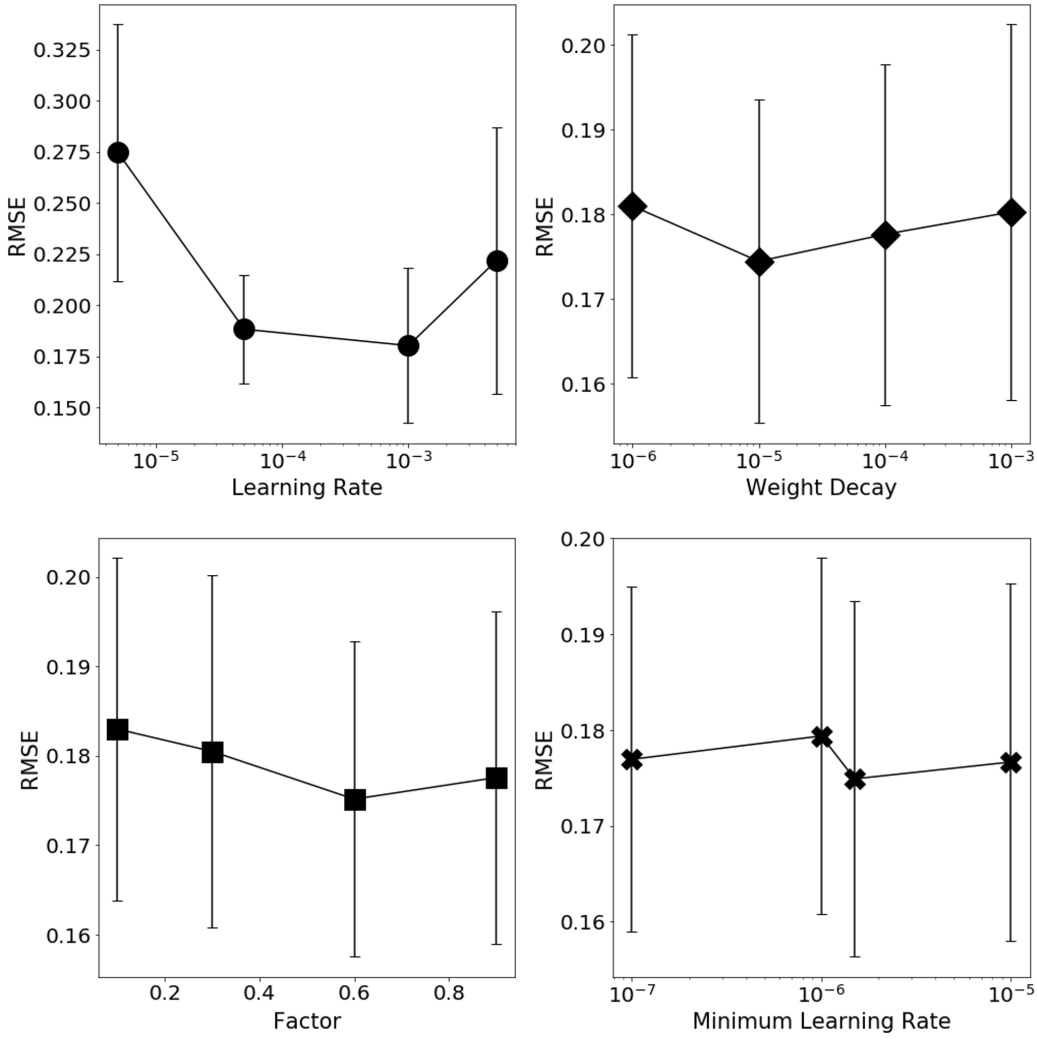

Figure 2.4: Hyperparameter performance in the neighborhood of optimum hyper parameter set in terms of the root mean square error (RMSE) for the test data. The data set used for this hyperparameter search are the simulations using permeability maps generated according to the correlation length of $\lambda_Y = 19$ m. Unless labeled as the x-axis variable, all plot correspond to $\eta = 5 \times 10^{-3}$, $\psi = 1 \times 10^{-5}$, $\gamma = 0.6$, and $\eta_{min} = 5 \times 10^{-6}$. Each data point represents the mean and standard deviation of ten training sessions.

as a starting point in the hyperparameter optimization for multifidelity transfer learning. Then, the η and epochs at each phase (Section 2.2.2) are modified to minimize the RMSE on the corresponding test data. The resulting hyperparameter values are shown in Table 2.3.

2.4 Results

CNNs were trained and the predictive accuracy and UQ performance were evaluated on data sets generated using correlation lengths of $\lambda_Y = 19$ m and $\lambda_Y = 8$ m. The data and results associated with the correlation lengths of $\lambda_Y = 19$ m and $\lambda_Y = 8$ m are referred to as "long" and "short", respectively. The long data set was used to initially explore the effectiveness of training the CNN on multilevel data using transfer learning. The short data set serves to demonstrate the robustness of the CNN architecture and selected hyper parameters.

2.4.1 Models Trained on Long Data

Once trained on multifidelity long data (in this example, on 573 LFS and 100 HFS, which took 12 hours to generate), the CNN surrogate provides an accurate approximation of the PDE solution on the fine mesh, even for such highly nonlinear problems as Equation (2.2) that exhibit sharp dynamic fronts. A forward pass of the CNN surrogate is on the order of a second, whereas the fine-mesh PDE solution takes nearly 220 seconds. This two orders of magnitude speed-up makes CNN surrogates an invaluable tool for UQ (Section 4.4.2).

Model Performance

We compare the relative performance of the CNN trained on multifidelity data and the CNNs trained on either HFS data or LFS data, in terms of both accuracy (RMSE on test data) and computational cost. We also investigate the effect of varying the amount of HFS and LFS data for a given computational budget of 12 hours.

To train the high-resolution (128×128 output) CNN solely on the LFS (64×64) data, the latter

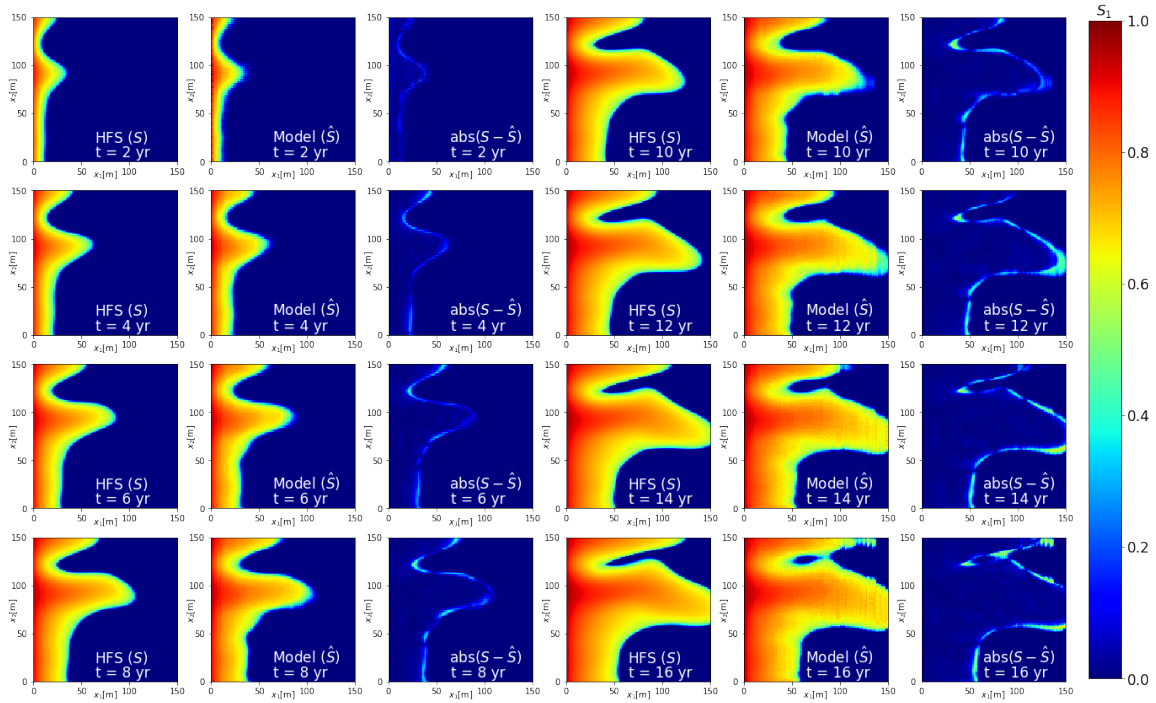

Figure 2.5: Temporal snapshots of the saturation maps $S_1(\mathbf{x}, t)$ for the permeability field $k(\mathbf{x})$ generated using the correlation length of $\lambda_Y = 19$ m in Figure 2.2. These are generated with either HFS of the PDE model described by Equation (2.2) and Equation (2.4) (labeled as S in the first and fourth columns) or the CNN surrogate (labeled as $\hat{S}$) in the second and fifth columns). The third and sixth columns display the absolute difference between the two predictions, $|S - \hat{S}|$.

have to be downscaled to match the dimensions. We do so by taking the Kronecker product of a 64×64 LFS image and a 2×2 matrix of 1s. The Kronecker product operation, commonly denoted by "$\otimes$", is defined as: given a $m \times n$ matrix $\mathbf{A}$ and a $p \times q$ matrix $\mathbf{B}$, the Kronecker product $\mathbf{A} \otimes \mathbf{B}$ results in a $mp \times nq$ matrix $\mathbf{C}$ with elements defined by

$$c_{\alpha\beta} = a_{ij} b_{kl}, \quad \alpha \equiv p(i-1) + k, \quad \beta \equiv q(j-1) + l. \tag{2.6}$$

In our application, the modified LFS data is $\mathbf{C}$, LFS data is $\mathbf{A}$, and the 2×2 matrix of ones is $\mathbf{B}$. When the definition in Equation 2.6 is applied, the transformed LFS data $\mathbf{C}$ would take a form which contains 2×2 clusters of the elements of $\mathbf{A}$:

$$\mathbf{C} = \mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix}. \tag{2.7}$$

The transformed LFS data have the desired HFS dimensions, while containing the same information as the original image. The test data are composed of HFS images (PDE solves on fine mesh) that were not used for CNN training.

Figure 2.6 exhibits the RMSEs on test data of the CNNs trained on high-, low-, and multifidelity data as function of the computational budget; each point in these graphs represents an average over ten repetitions of training and is accompanied by error bars (the standard deviation).

The left plate of Figure 2.6 reveals that, if the data-generation budget does not exceed 20 hours, the CNN trained on the LFS data outperforms its HFS-trained counterpart in terms of RMSE. That is because such budgets do not allow for generation of sufficient amounts of HFS data. As the budget increases, the error of the LFS data precludes RMSE of the CNN trained on such data from dropping below 0.125 while RMSE of the HFS-trained CNN continues to decrease. This finding is reminiscent of the cost-constrained selection between high- and low-fidelity models in the context of ensemble-based simulations (Yang et al., 2020; Sinsbeck and Tartakovsky, 2015). This figure also

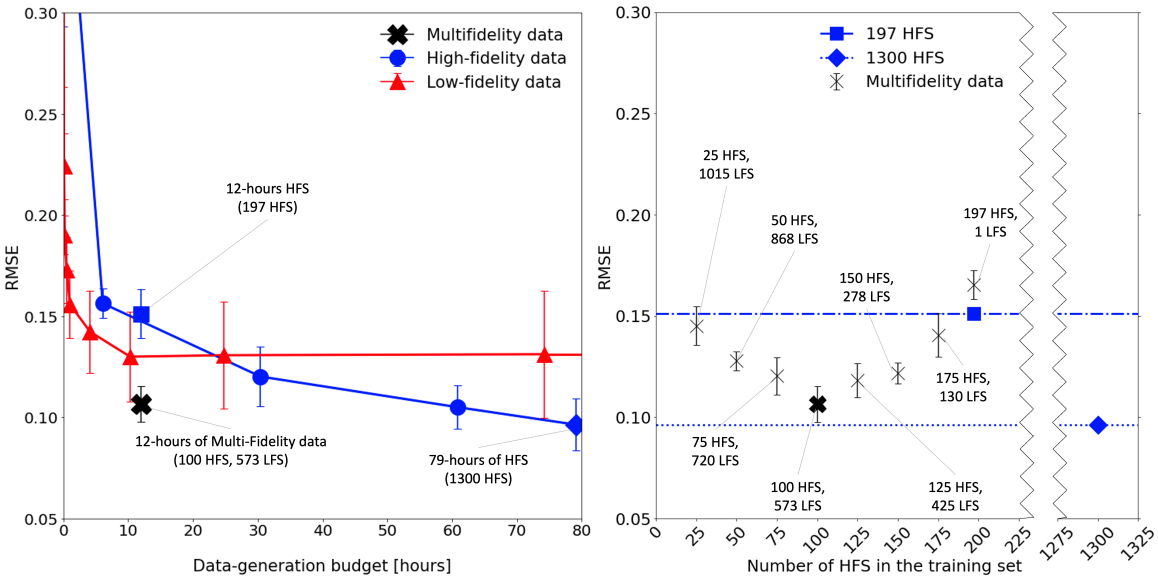

Figure 2.6: RMSE on test data for the alternative CNN training strategies. It is plotted as function of the budget allocated for data generation (left) and the number of PDE solves on the fine mesh used to generate HFS data (right). Each RMSE point in these graphs represents an average over ten iterations of training and is accompanied by error bars (the standard deviation). The left plate provides RMSE for the CNNs trained on high-fidelity (blue circles), low-fidelity (red triangles), or multifidelity (black $\times$) data. The latter corresponds to the CNN trained on an optimal (the lowest RMSE) mix of high- and low-fidelity data for a set budget of 12 hours; it is contrasted with the RMSE of the CNN trained on the HFS data generated within the same budget (blue square). The black circles in the right plate represent RMSE of the CNN trained on the multifidelity data sets, in which the number of HFS varies while the data-generation budget is fixed at 12 hours. Also shown there are RMSEs of the CNNs trained on 12 hours (dot-dashed line) and 79 hours (dotted line) of HFS.

demonstrates that, for a relatively small budget of 12 hours, the use of multifidelity data yields the CNN whose RMSE is appreciably smaller that those of the CNNs trained on either HFS data or LFS data.

An optimal mix of the HFS and LFS data is investigated in the right plate of Figure 2.6. The multifidelity training was conducted five times for each HFS/LFS ratio, with random selection of LFS/HFS from a larger pool data. At the empirically optimal mix of 573 LFS and 100 HFS, five of our experiments yield RMSE values of 0.097, 0.12, 0.098, 0.105, and 0.110. Two of the five CNN's trained on 12 hours worth of these multifidelity data achieve lower RMSEs than the RMSE of 0.099 for the CNN trained on 79 hours worth of HFS data. This LFS/HFS ratio lies near the range, $1.5 - 5.5$, suggested for MLMC (Taverniers et al., 2020). The latter theoretical results and the numerical experiments presented here, we recommend the LFS/HFS ratio of five as a suitable initial guess. For the data-generation budget of 12 hours, a mix dominated by the LFS data results in a CNN whose RMSE on test data exceeds 1.0 (beyond the scale of Figure 2.6), which indicates that the network's last Convolution Transpose 2 layer is not meaningfully trained.

CNN Surrogates for Uncertainty Quantification

We investigate the utility of our CNN surrogates for uncertainty quantification. A quantity of interest is the breakthrough time, T_{break}, at the $x_1 = 100$ m plane (Figure 2.2), with the term "breakthrough" defined as the saturation of the invading phase (S_1) exceeding 0.15. Given uncertainty in intrinsic permeability $k(\mathbf{x})$, a solution of Equation (2.2) and, hence, predictions of T_{break} are given in terms of their cumulative distribution functions (CDFs) or probability density functions (PDFs).

Figure 2.7 exhibits the CDF and PDF of T_{break} alternatively computed with HFS and LFS MC simulations and with the CNN trained on the multifidelity data. The distributions obtained via MC simulation consisting of 292 hours of HFS are treated as ground truth. The distributions obtained from 24 hours of LFS involve a sufficient number of samples for the error to be attributable solely to the low resolution, i.e., to the disretization errors in solving PDEs. The numbers of HFS samples generated during either 6 or 12 hours of simulations are insufficient for MC simulations to converge,

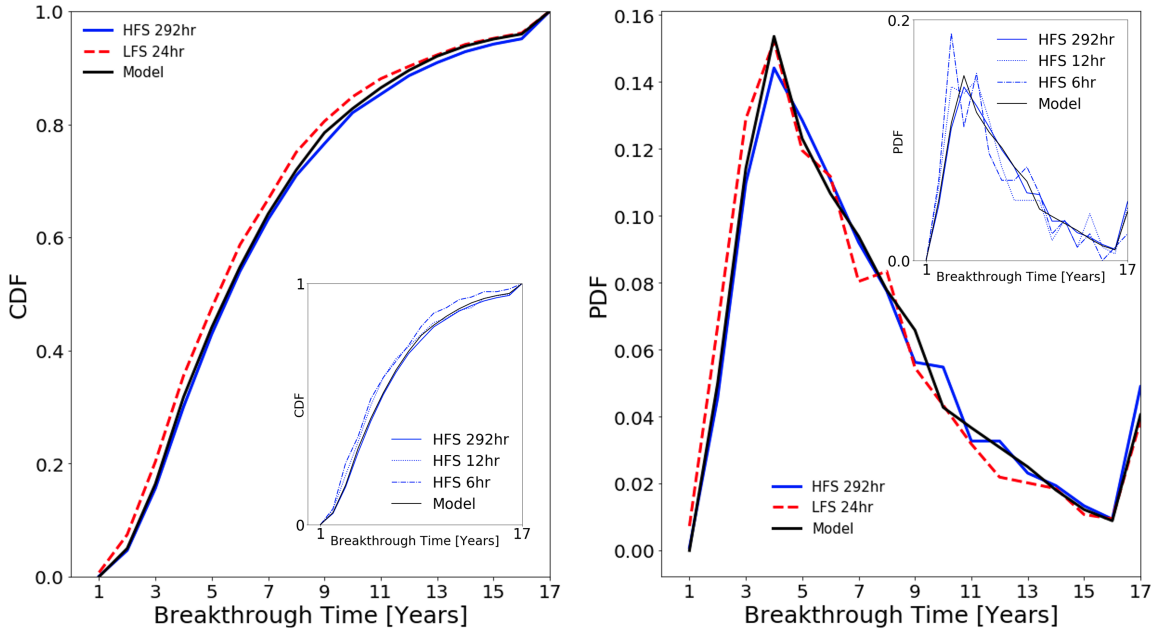

Figure 2.7: The converged CDF (left) and PDF (right) of breakthrough time is calculated using MC simulations of HFS, LFS, and the CNN surrogate model. The inner panel of each plate displays the CDF (left) and PDF (right) obtained from unconverged HFS MC simulations.

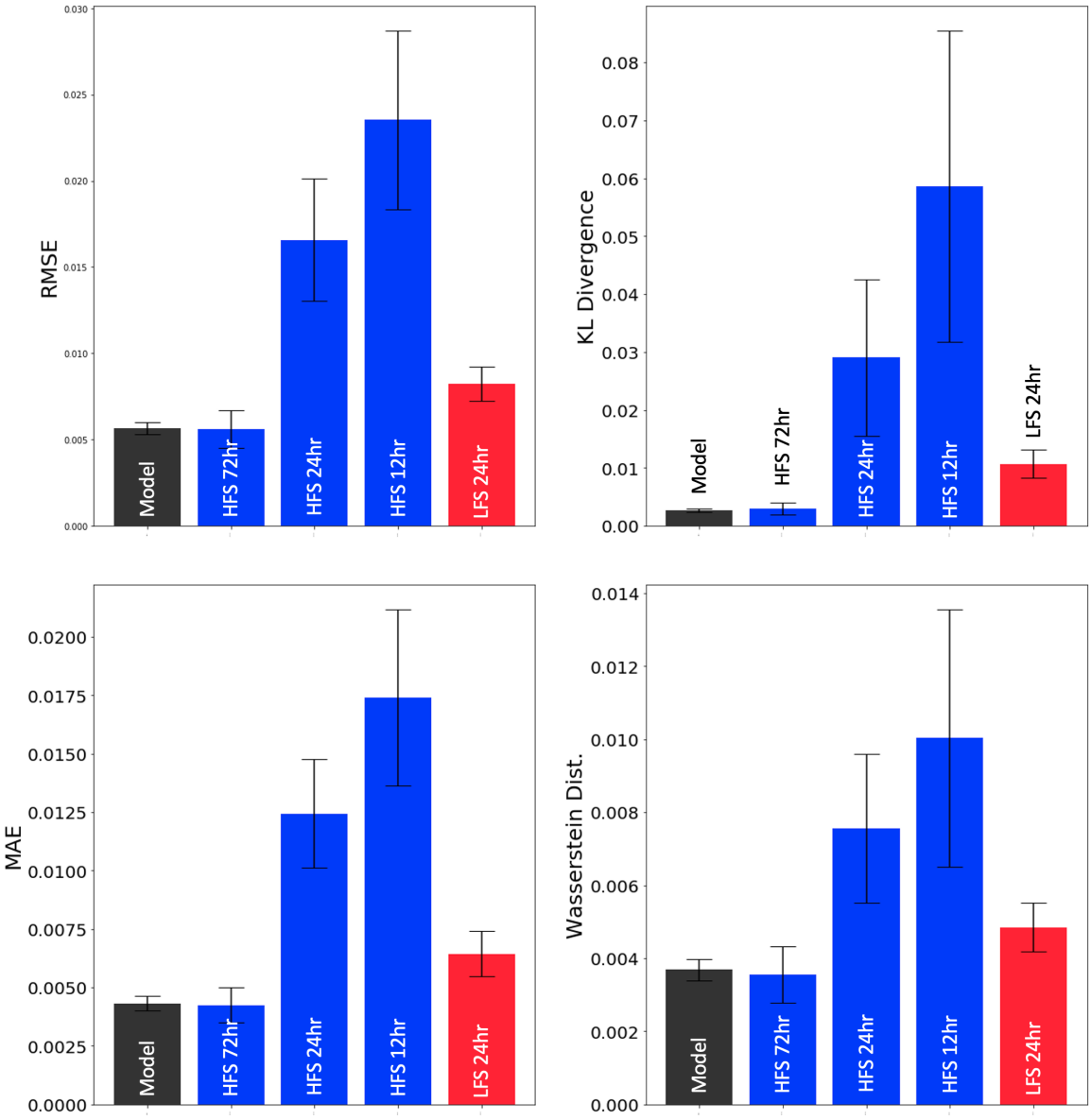

Figure 2.8: The converged CDF and PDF of breakthrough time is calculated using MC simulations of HFS, LFS , and the CNN surrogate model(Figure 2.7). The CDF and PDF calculated from varying amounts of HFS are displayed on the subplots. Bar plots: RMSE (left-top), MAE (left-bottom), KL divergence (right-top), and Wasserstein distance (right-bottom) from PDF calculated using CNN model, HFS, and LFS.

leading to the appreciable errors in estimation of PDF and CDF of T_{break}. The CNN trained on multifidelity data yields accurate estimates of these quantities, while requiring only 12 hours of data generation.

In addition to visual comparison, the alternative strategies for estimation of the distributions of T_{break} are compared in terms of RMSE, MAE, the Kullback-Leibler (KL) divergence, and the first Wasserstein distance. KL divergence D_{KL} of discrete PDFs is defined as

$$D_{\text{KL}}(P||Q) = \sum_{x \in \chi} P(x) \log \left(\frac{P(x)}{Q(x)} \right) \tag{2.8}$$

where P and Q are discrete PDFs on the same probability space χ; P represents the ground truth and Q represents the modeled PDF. The first Wasserstein distance W_1 for one-dimensional distributions, or more commonly referred to as just "Wasserstein distance", is defined as

$$W_1(u, v) = \int_{-\infty}^{\infty} |U(x) - V(x)| \mathrm{d}x \tag{2.9}$$

where u and v are probability measurements with respective CDFs U and V. The Wasserstein distance was calculated using the Python package: scipy.stats.wasserstein_distance.

The UQ task was repeated 50 times, with Figure 2.8 displaying the mean and standard deviation of these measures of discrepancy. We found 3200 forward passes of the CNN to be sufficient for the CDF/PDF estimates to converge; this UQ task took about ten minutes, whereas an equivalent HFS MC simulation takes 194 hours. By every discrepancy measure, the CNN estimates outperform the converged LFS MC simulation and are at least as accurate as the HFS MC simulation using 72 hours of simulation time. Likewise, the CNN estimates are vastly more accurate than the HFS MC simulation of a similar data-generation budget.

2.4.2 Models Trained on Short Data

CNNs were trained to solve the long problem using multifidelity data using transfer learning on particular CNN architecture and hyperparameters. The generalizability of the method, model, and hyperparameters is tested by training the same CNN architecture to solve the short problem. Once trained (using 573 LFS and 100 HFS of the short data, which took 12 hours to generate), the CNN surrogate accurately approximates the PDE solution on the fine mesh (Figure 2.9). The short data set comes, once again, from a highly nonlinear problem that exhibits sharp dynamic fronts. Compared to the long data set, the shorter correlation lengths in the short data set contains more frequent permeability field changes the CNN must handle.

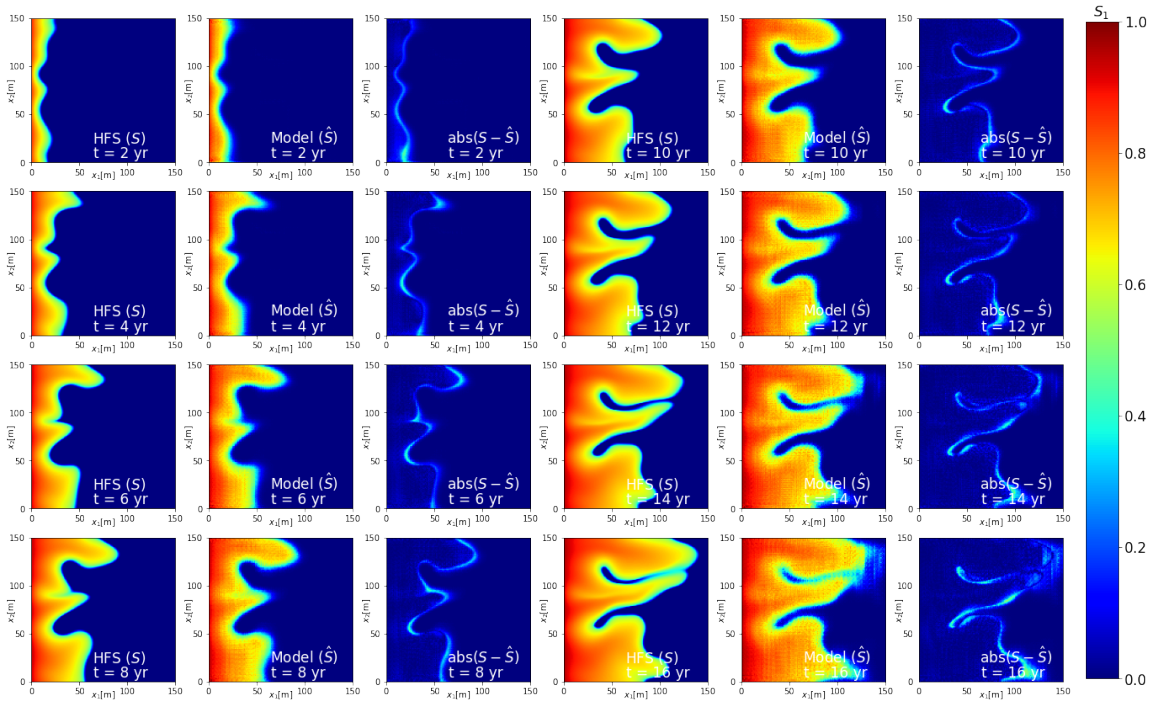

Figure 2.9: Temporal snapshots of the saturation maps $S_1(\mathbf{x}, t)$ for the permeability field $k(\mathbf{x})$ generated using the correlation length of $\lambda_Y = 8$ m in Figure 2.2. These are generated with either HFS of the PDE model described by Equation (2.2) and Equation (2.4) (labeled as S in the first and fourth columns) or the CNN surrogate (labeled as $\hat{S}$) in the second and fifth columns). The third and sixth columns display the absolute difference between the two predictions, $|S - \hat{S}|$.

Model Performance

CNNs were trained on the short data using the multifidelity data and the high-fidelity data using the same hyperparameters found using the long data. The relative performance of the CNN trained on a short multifidelity data and the CNNs trained on short HFS data is compared in terms of both accuracy (RMSE on test data) and computational cost. Furthermore, evaluation of the CNN trained on short multifidelity data was expanded to include data budgets of 6, 12, 24, and 48 hours; the ratios of the short HFS and short LFS were kept consistent for each data generation budget. The size of the HFS and LFS data sets is reported in Table 2.4. This experiment was repeated for the long data set and a comparison is made in Figure 2.10.

Table 2.4: HFS and LFS data used to train the CNNs trained on multifidelity data

Total data budget	HFS data points	LFS data points
6 hours	50	286
12 hours	100	573
24 hours	200	1146
48 hours	400	2292

Although the long and short data differ upon visual inspection, Figure 2.9 reveals that the CNN learns both problems at a similar rate when trained solely on varying amounts of high-fidelity data. As the data generation budget increases, the improvements in RMSE decreases for the CNNs trained on high-fidelity data. The CNNs trained on only high fidelity data is insensitive to whether it is training on the long or short data. The CNNs trained on long high-fidelity data were tested on long high-fidelity data and the CNNs trained on short high-fidelity data were tested on short high-fidelity data. The hyperparameters used to train the fine model are robust and functional for both the long and short data sets.

Despite the fact the hyperparameters were optimized for the long data set, the CNNs trained on multifidelity short data achieve better predictive accuracy compared to CNNs trained on multifidelity long data. For each given data budget, the CNNs trained on multifidelity data achieve more accurate RMSEs at lower data generation budgets than the CNNs trained on only high-fidelity data. As the

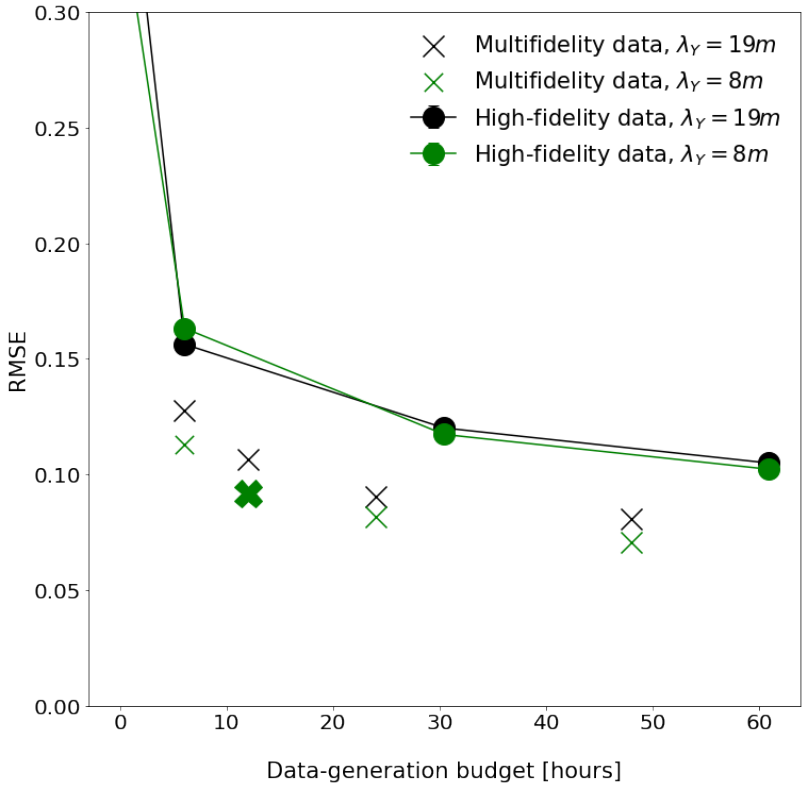

Figure 2.10: RMSE on test data for the alternative CNN training strategies. It is plotted as function of the budget allocated for data generation. Each RMSE point in these graphs represents an average over 10 iterations of training. The data points in black provide the RMSE for the CNNs trained on long ($\lambda_Y = 19$ m) data and the green data points provide the RMSE for the CNNs trained on the short ($\lambda_Y = 8$ m) data. The dots connected by the lines are provide the RMSEs of CNNs trained on high fidelity data, and the $\times$s mark the RMSE achieved by CNNs trained on various data budgets of multifidelity data.

RMSE vs. data-generation budget appears to be asymptotic and the CNNs trained on multifidelity data outperform their counterparts trained on only more than 60 hours of high-fidelity data, this gain is significant. The CNNs trained on long multifidelity data were tested using short high-fidelity data and the CNNs trained on short multifidelity data were tested using long high-fidelity data.

The CNN trained using 12 hours of short multifidelity data (573 LFS and 100 HFS short data), denoted by the bold green x, was made to perform the UQ task and the results are discussed in the next section.

CNN Surrogates for Uncertainty Quantification

Finally, we investigate the utility of our CNN surrogates trained on the short data for uncertainty quantification. Again, the quantity of interest is the breakthrough time, T_{break}, at the $x_1 = 100$ m plane (Figure 2.2), with the term "breakthrough" defined as the saturation of the invading phase (S_1) exceeding 0.15. Given uncertainty in intrinsic permeability $k(\mathbf{x})$, a solution of Equation (2.2) and, hence, predictions of T_{break} are given in terms of their cumulative distribution functions (CDFs) or probability density functions (PDFs).

Figure 2.11 presents the CDF and PDF of T_{break} alternatively computed with HFS and LFS MC simulations and with the CNN trained on the multifidelity data. The distributions obtained via MC simulations consisting of 292 hours of HFS are treated as ground truth. The distributions obtained from 24 hours of LFS involve a sufficient number of samples for the error to be attributable solely to the low resolution, i.e., to the discretization errors in solving PDEs. The numbers of HFS samples generated during either 6 or 12 hours of simulations are insufficient for MC simulations to converge, leading to the appreciable errors in estimation of PDF and CDF of T_{break}. The CNN trained on multifidelity data yields accurate estimates of these quantities, while requiring only 12 hours of data generation. It is noted that, for this short problem, the CNN's error is biased in the direction of predicting a breakthrough time that is too early as evidenced by a good match in the PDF and a slight mismatch at breakthrough times greater than 7.5 years. In contrast, the LFS MC results in a PDF which both under-predicts and over-predicts the distribution at different breakthrough times.

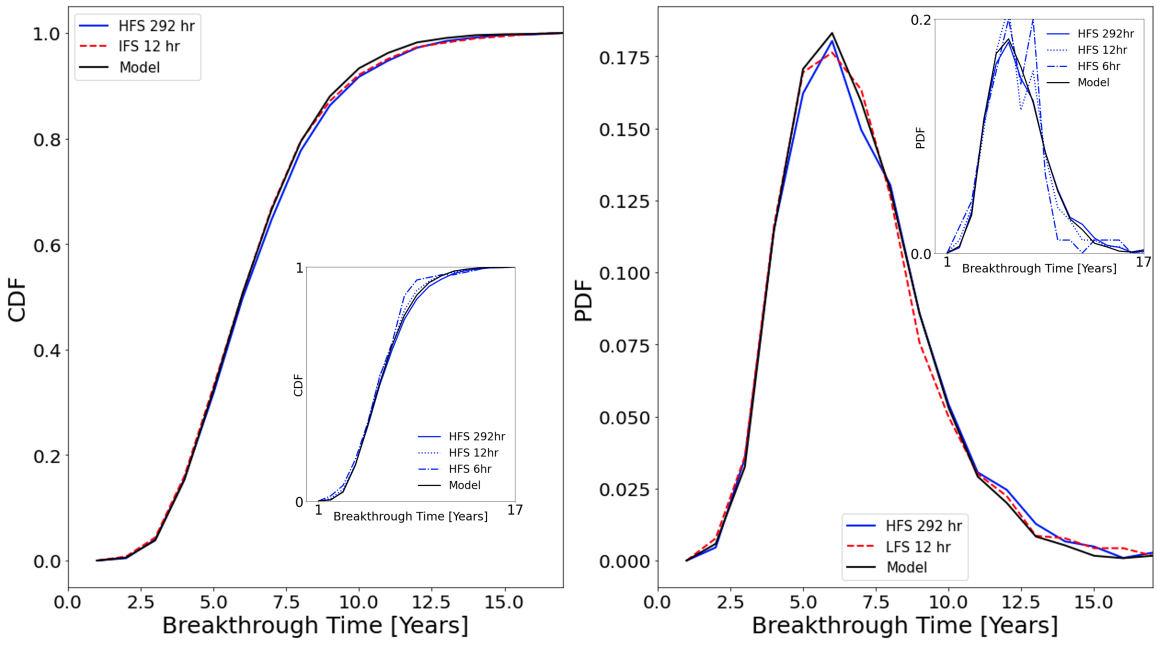

Figure 2.11: The converged CDF (left) and PDF (right) of breakthrough time is calculated using MC simulations of HFS, LFS using the short (correlation length of $\lambda_Y = 8m$) data set, and the CNN surrogate model trained on the short data set. The inner panel of each plate displays the CDF (left) and PDF (right) obtained from unconverged HFS MC simulations.

In addition to visual comparison, the alternative strategies for estimation of the distributions of T_{break} are compared in terms of RMSE, MAE, the Kullback-Leibler (KL) divergence, and the first Wasserstein distance. The UQ task was repeated 50 times, with Figure 2.12 displaying the mean and standard deviation of these measures of discrepancy. We found 3200 forward passes of the CNN to be sufficient for the CDF/PDF estimates to converge; this UQ task took about 10 minutes, whereas an equivalent HFS Monte Carlo takes 194 hours. By every discrepancy measure, the CNN estimates outperform the converged LFS Monte Carlo and are at least as accurate as the HFS Monte Carlo using 72 hours of data. Likewise, the CNN estimates are vastly more accurate than the HFS Monte Carlo of a similar data-generation budget.

2.5 Conclusions

We proposed a transfer learning-based approach to train a CNN on multifidelity (e.g., multiresolution) data. High- and low-fidelity images were generated by solving a PDE on fine and coarse meshes, respectively. The performance of our algorithm was tested on two data sets generated from a system of nonlinear parabolic PDEs governing multiphase flow in a heterogeneous porous medium with uncertain (random) permeability. The different data sets were generated from different correlation lengths defining the random permeability fields. A quantity of interest in this example is the PDF or CDF of the breakthrough time of an invading fluid. Our analysis leads to the following major conclusions.

1. CNN surrogates trained on multifidelity data, for both data sets, provide an accurate approximation of the PDE solution on the fine mesh, even for highly nonlinear problems that exhibit sharp dynamic fronts. A forward pass of the CNN surrogate is two orders of magnitude faster than a PDE solution on the fine-mesh. This speed-up makes CNN surrogates an invaluable tool for ensemble-based computation of the PDF/CDF of a quantitiy of interest.

2. CNN training on multifidelity data reduces the data-generation budget seven-fold relative to

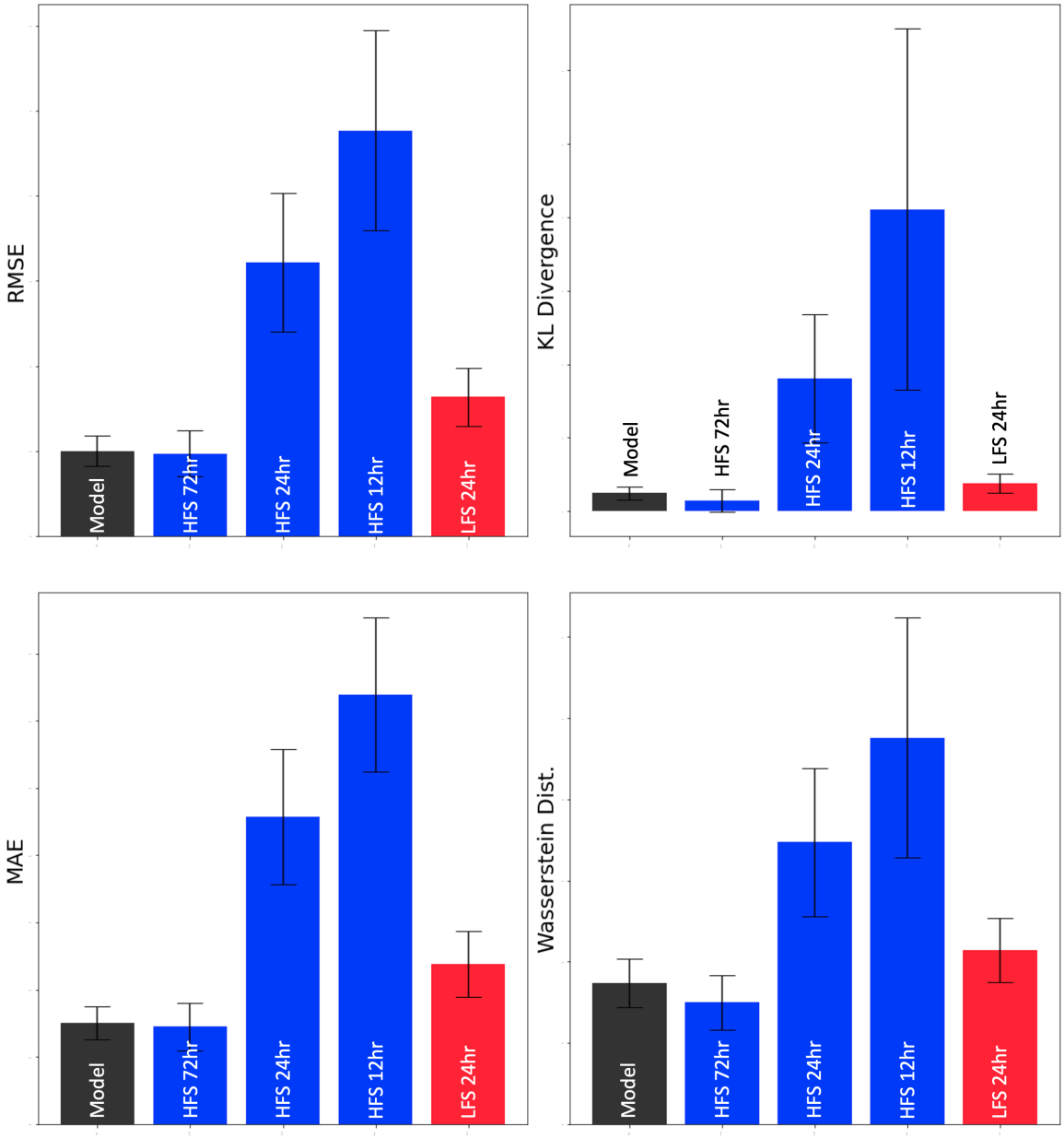

Figure 2.12: The converged CDF and PDF of breakthrough time is calculated using MC simulations of HFS, LFS , and the CNN surrogate model(Figure 2.11). The CDF and PDF calculated from varying amounts of HFS are displayed on the subplots. Bar plots: RMSE (left-top), MAE (left-bottom), KL divergence (right-top), and Wasserstein distance (right-bottom) from PDF calculated using CNN model, HFS, and LFS.

to CNN training on HFS data alone. If the budget is relatively small, the CNN trained on the LFS data is more accurate than its HFS-trained counterpart. As the budget increases, the opposite is true. This finding is reminiscent of the cost-constrained selection between high- and low-fidelity models in the context of ensemble-based simulations.

3. For a small data-generation budget (12 hours, in our example), the CNN trained on multifidelity data exhibits an appreciably smaller RMSE on test data than the CNNs trained on either HFS or LFS data. Performance of the multifidelity CNN depends on the ratio between HFS and LFS in the training set. Theoretical studies from MLMC can be used to guide the selection of an optimal mix of low- and high-fidelity data.

4. The CNNs trained on multifidelity data are largely insensitive to the discretization error of LFS. CNN-derived estimates of the PDF and CDF of the quantity of interest are close to those of converged high-fidelity MC simulations; but the former are two orders of magnitude faster to obtain than the latter.

5. The CNNs trained on large (greater than 12 hours) data budgets of multifidelity data is more accurate than the CNNs trained on HFS data. Relying on multilevel data to train one CNN via transfer learning is a promising tool for construction of accurate surrogate models.

6. The hyperparameter search for the training of a CNN surrogate can start with the hyperparameters of another surrogate model with a similar architecture. If the problem and architecture are sufficiently similar, a hyperparameter search may be unnecessary.

The computational efficiency and accuracy of CNN training on multifidelity data depend crucially on the HFS/LFS ratio. We relied on the theoretical results for MLMC as an empirical guide for the selection of this ratio; a detailed theoretical and/or experimental investigation of an optimal HFS/LFS ratio is left for the future. Another direction of subsequent studies is the use of multiple physical models of different complexity, rather than a single model solved on different numerical grids, to generate multifidelity data.

Chapter 3

Training on Three Levels of Data

NN surrogates can emulate the behavior of complex systems in the sense that they are capable of replicating PDE solutions. Once trained, a virtually negligible forward pass computational cost renders them an attractive tool for ensemble-based computation. However, as the generation of training data is itself an ensemble-based computation, reducing the computational cost of generation of the training data increases the value of the NN surrogate. In Chapter 2, we reduced the computational cost of data generation by using two levels of multifidelity data to train a CNN using transfer learning. This idea is explored further in this chapter by training a CNN on three levels of data using transfer learning. Three sets of high-, low-, and very-low-fidelity images are generated by solving PDEs on fine, coarse, and very coarse meshes, respectively. The effectiveness of this multifidelity training strategy is evaluated in terms of its predictive capability and ability to perform an uncertainty quantification (UQ) task. The predictive capability of this strategy, measured using root mean squared error of a test data set, is compared to that of the strategy in Chapter 2. The UQ task is the estimation of the distribution of a quantity of interest, whose dynamics is governed by a system of nonlinear PDEs (parabolic PDEs of multiphase flow in heterogeneous porous media) with uncertain/random parameters. The UQ task is evaluated in terms of both the Wasserstein distance and the Kullback–Leibler divergence.

3.1 Workflow for CNN Training on Three Levels of Data

Our strategy for training a CNN on three levels of multifidelity data starts by generalizing the training scheme for two degrees of fidelity: the lower-fidelity S_i and higher-fidelity S_{i+1}. The definition of S_i and S_{i+1} is intentionally minimalist. It may refer to simulation results obtained on coarse and fine meshes, as we do in this study, or to solutions of two models accounting for fewer and more physical processes, or to synthetic and experimental data. Let us define N_i and N_{i+1} as NNs that operate on S_i and S_{i+1}, respectively. Our goal is to construct a trained NN N^*_{i+1}, which operates on S_{i+1} (here and below, the * denotes a trained NN). If $N^*_i \in N^*_{i+1}$, then the difference between the models, $N^*_{\Delta i,i+1}$, is also a model and $N^*_i + N^*_{\Delta i,i_1} = N^*_{i+1}$. The transfer learning approach to obtaining N^*_{i+1} comprises the following steps.

1. Train N_i on the S_i data to obtain N^*_i.
2. Freeze the weights of N^*_i to prevent them from being updated.
3. Construct N_{i+1} by attaching $N_{\Delta i,i+1}$, whose weights are allowed to update, to N^*_i.
4. Train N_{i+1} on the S_{i+1} data to obtain N^*_{i+1}.
5. Unlock all weights of N^*_{i+1} to enable further updating.

Our results reported in Chapter 2 demonstrate that this approach can greatly reduce the data generation cost relative to the computational budget necessary for the NN training on the fine scale.

We promulgate a sequential approach to the NN training on multifidelity data, which starts with the two lowest degrees of fidelity, S_i and S_{i+1} with $i = 1$; and then repeats the above training procedure by adding the progressively refined data ($i > 1$) one level at a time. We consider simulation data $\mathbf{u}(\mathbf{x}_n, t_k)$ with three degrees of fidelity, corresponding to very-low- (VLFS), low- (LFS), and high-fidelity (HFS) solutions of Equation (1.1). These are obtained by solving Equation (1.1) on very-coarse ($N^{\mathrm{VLFS}}_{\mathrm{el}}$), coarse ($N^{\mathrm{LFS}}_{\mathrm{el}}$), and fine ($N^{\mathrm{HFS}}_{\mathrm{el}}$) meshes, respectively. The numbers of elements in these meshes satisfy the order relation $N^{\mathrm{VLFS}}_{\mathrm{el}} < N^{\mathrm{LFS}}_{\mathrm{el}} < N^{\mathrm{HFS}}_{\mathrm{el}}$; in the simulations reported in this chapter, we set $N^{\mathrm{HFS}}_{\mathrm{el}} = 4N^{\mathrm{LFS}}_{\mathrm{el}} = 16N^{\mathrm{VLFS}}_{\mathrm{el}}$.

Transfer learning allows a CNN trained on one data set to be modified and retrained to function on another data set. Thus, a CNN trained on VLFS is retrained on LFS, and this CNN is again retrained on HFS. Let $\mathbf{w}_{\mathrm{HFS}}$, $\mathbf{w}_{\mathrm{LFS}}$, $\mathbf{w}_{\mathrm{VLFS}}$ denote the weights in the CNNs functioning on HFS, LFS, and VLFS respectively. Then, a CNN architecture where $\mathbf{w}_{\mathrm{VLFS}} \in \mathbf{w}_{\mathrm{LFS}} \in \mathbf{w}_{\mathrm{HFS}}$ is ideal for transfer learning. Our implementation of transfer learning starts with the construction of a CNN with weights $\mathbf{w}_{\mathrm{VLFS}} = (w_1, \ldots, w_{N_{\mathrm{VLFS}}})^\top$ trained on the VLFS data. Then, additional weights are added to form $\mathbf{w}_{\mathrm{LFS}} = (w_1, \ldots, w_{N_{\mathrm{VLFS}}}, w_{N_{\mathrm{VLFS}}+1}, \ldots, w_{N_{\mathrm{LFS}}})^\top$; these are estimated by minimizing the discrepancy with the LFS data. Finally, more weights are added to form $\mathbf{w}_{\mathrm{HFS}} = (w_1, \ldots, w_{N_{\mathrm{LFS}}}, w_{N_{\mathrm{LFS}}+1}, \ldots, w_{N_{\mathrm{HFS}}})^\top$; these are estimated by minimizing the discrepancy with the HFS data.

The resulting strategy for CNN training on multifidelity data consists of three phases, each of which yields a CNN M_i $(i = 1, 2, 3)$.

Phase 1:
(a) The CNN M_1 with the $N_{\mathrm{el}}^{\mathrm{VLFS}} \times N_{\mathrm{el}}^{\mathrm{VLFS}}$ output is trained on the VLFS data to become M_1^*.

(b) Weights of M_1^*, $\mathbf{w}_{\mathrm{VLFS}}$, are frozen.

(c) Trainable weights are added to M_1^* to form M_2 with the $N_{\mathrm{el}}^{\mathrm{LFS}} \times N_{\mathrm{el}}^{\mathrm{LFS}}$ output. The CNN M_2 has $\mathbf{w}_{\mathrm{LFS}}$, among which the weights $\mathbf{w}_{\mathrm{VLFS}}$ are still frozen.

(d) The CNN M_2 is initially trained on the LFS data.

(e) All weights $\mathbf{w}_{\mathrm{LFS}}$ of M_2 are unlocked.

Phase: 2
(a) The CNN M_2 with the $N_{\mathrm{el}}^{\mathrm{LFS}} \times N_{\mathrm{el}}^{\mathrm{LFS}}$ output is trained on the LFS data to become M_2^*.

(b) The weights $\mathbf{w}_{\mathrm{LFS}}$ of M_2 are frozen.

(c) More trainable weights are added to M_2^* to form the CNN M_3 with the $N_{\mathrm{el}}^{\mathrm{HFS}} \times N_{\mathrm{el}}^{\mathrm{HFS}}$ output. The CNN M_3 has weights $\mathbf{w}_{\mathrm{HFS}}$, among which the weights $\mathbf{w}_{\mathrm{LFS}}$ are still frozen.

(d) The CNN M_3 is initially trained on the HFS data.

(e) All the weights $\mathbf{w}_{\text{HFS}}$ of M_3 are unlocked.

Phase:3 The CNN M_3 is trained on the HFS data to become M_3^*.

With a proper selection of the CNN architecture, this scheme estimates the bulk of the weights from the VLFS and LFS data. This procedure is most effective when the computational cost associated with the HFS data is significantly greater than that of the LFS and VLFS data. Since the bulk of the CNN M_3 training is done on the VLFS data, this procedure is more efficient than CNN training solely on the HFS data.

3.2 Data Generation

The physical phenomenon modeled in this chapter, two-phase flow in heterogeneous porous media, is the same as in Chapter 2. The key difference is that now we also generate data on the very-coarse mesh consisting of 32×32 elements.

3.2.1 Upscaling of Permeability

Multifidelity (HFS, LFS, and VLFS) data are generated by solving Equations (2.2)–(2.4) on progressively coarsened grids consisting of 128×128, 64×64, and 32×32 elements, respectively. The spatial discretizations of these HFS, LFS, and VLFS ($\Delta x = 1.17$ m, 2.34 m, and 4.38 m, respectively) are sufficient to capture the random fluctuations of a heterogeneous permeability field. Specifically, they satisfy the "rule of thumb" requirement that Δx be such that $4\Delta x \leq \lambda_Y$, i.e., that a numerical mesh should have at least four elements of length Δx per correlation length λ_Y (e.g., Ye et al., 2004, and references therein); the correlation length of our permeability field $\hat{k}$ is $\lambda_Y = 19$ m.

Grid coarsening must be accompanied by upscaling (coarsening) of the realizations of the random permeability $\hat{k}$, which are initially generated at the finest scale (Figure 2.2). Among alternative upscaling strategies (Paleologos et al., 1996; Tartakovsky and Neuman, 1998; Boso and Tartakovsky,

2018), we select the one proposed by Durlofsky (2005) because of its computational simplicity. This method turns a scalar permeability field, defined on the fine (128×128) mesh, into its upscaled tensorial (anisotropic) counterpart whose off-diagonal components are zero and the diagonal components are computed as the distance-weighted arithmetic mean perpendicular to the direction of flow and the distance-weighted harmonic mean in the direction of flow.

Multifidelity training data come in the form of $N_{\text{ts}} = 16$ temporal snapshots of the saturation $S_1(\mathbf{x}, t)$ computed by solving Equations (2.2)–(2.4) on the 128×128, 64×64, and 32×32 grids. Figure 3.1 shows examples of such images, corresponding to the permeability field in Figure 2.2. The permeability fields on the finest mesh, $[1 \times N_{\text{el}}^{\text{HFS}} \times N_{\text{el}}^{\text{HFS}}]$, are used as the input $\boldsymbol{\theta}$ for all CNNs. The size of the CNN, $[N_{\text{ts}} \times N_{\text{el}} \times N_{\text{el}}]$, depends on the size of the training data ($N_{\text{el}} = N_{\text{el}}^{\text{HFS}}$, $N_{\text{el}}^{\text{LFS}}$, or $N_{\text{el}}^{\text{VLFS}}$).

3.3 CNN Training

Table 3.1 describes the CNN architecture used to implement our approach. The model implementation and training is done using `PyTorch` and other open source packages. The computations were carried out on the Stanford Mazama high-performance computing cluster. The allocated computing resources include Intel Xenon Gold 6126 CPU (2.6 GHz), 60GB RAM, and Nvidia V100 GPU with 16GB vRAM. (Although available, multicores were not used for this work.)

The key hyperparameters affecting the CNN performance are the learning rate η, the weight decay ψ, the factor γ, and the minimum learning rate η_{min}. As our methodology prescribes five training steps, the searchable hyperparameter space is large. Since in Chapter 2 we used the same architecture and the same HFS data, the hyperparameters obtained in that chapter serve as an initial guess for the hyperparameter optimization. Then, the learning rate η and epochs at each training (Section 3.1) are modified to minimize the RMSE on the corresponding test data. The nondefault hyperparameter values are shown in Table 3.2.

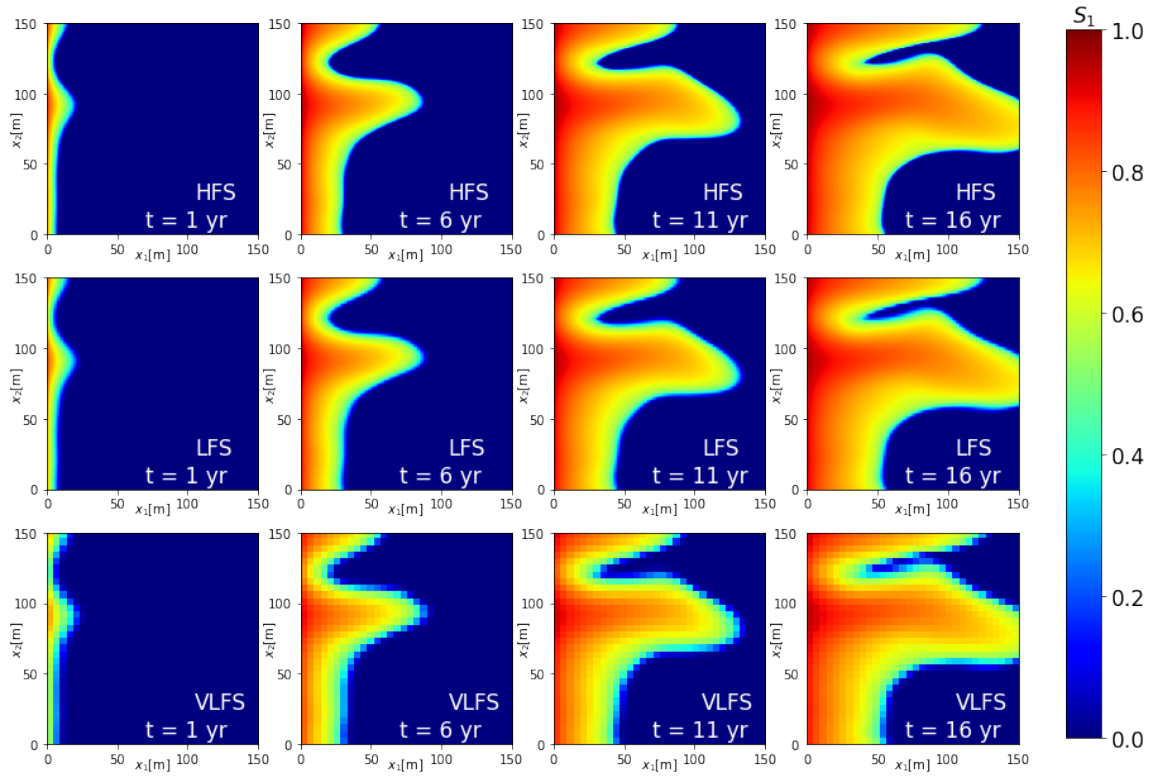

Figure 3.1: Temporal snapshots of saturation $S_1(\mathbf{x}, t)$ computed with HFS (top-row), LFS (middle-row) and VLFS (bottom-row) for the permeability field $k(\mathbf{x})$ in Figure 2.2.

Table 3.1: Model block description and the input and output dimensions of each model block. In our numerical experiments, the number of time steps is $N_{\text{ts}} = 16$; the number of elements in the fine and coarse meshes is $N_{\text{el}}^{\text{HFS}} \times N_{\text{el}}^{\text{HFS}} = 128 \times 128$, $N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{LFS}} = 64 \times 64$, and $N_{\text{el}}^{\text{LFS}} \times N_{\text{el}}^{\text{VLFS}} = 32 \times 32$ respectively; the number of elements in the output of the dense block is $N_{\text{dense}} = N_{\text{el}}^{\text{VLFS}} = 32$; and the number of channels in each of the seven layers of the CNN is $n_1 = 64$, $n_2 = 344$, $n_3 = 172$, $n_4 = 652$, $n_5 = 326$, $n_6 = 606$, and $n_7 = 303$.

Layer	Input	Output
Input: Permeability field k	$1 \times N_{\text{el}}^{\text{HFS}} \times N_{\text{el}}^{\text{HFS}}$	
Convolution 1	$n_1 \times N_{\text{el}}^{\text{HFS}} \times N_{\text{el}}^{\text{HFS}}$	$n_2 \times N_{\text{el}}^{\text{VLFS}} \times N_{\text{el}}^{\text{VLFS}}$
Dense Block (Encoding)	$n_2 \times N_{\text{el}}^{\text{VLFS}} \times N_{\text{el}}^{\text{VLFS}}$	$n_3 \times N_{\text{el}}^{\text{VLFS}} \times N_{\text{el}}^{\text{VLFS}}$
Convolution 2	$n_3 \times N_{\text{el}}^{\text{VLFS}} \times N_{\text{el}}^{\text{VLFS}}$	$n_4 \times N_{\text{dense}} \times N_{\text{dense}}$
Dense Block	$n_4 \times N_{\text{dense}} \times N_{\text{dense}}$	$n_5 \times N_{\text{dense}} \times N_{\text{dense}}$
Convolution Transpose 1	$n_5 \times N_{\text{dense}} \times N_{\text{dense}}$	$n_6 \times N_{\text{el}}^{\text{VLFS}} \times N_{\text{el}}^{\text{VLFS}}$
Dense Block (Decoding)	$n_6 \times N_{\text{el}}^{\text{VLFS}} \times N_{\text{el}}^{\text{VLFS}}$	$n_7 \times N_{\text{el}}^{\text{VLFS}} \times N_{\text{el}}^{\text{VLFS}}$
Convolution Transpose 2	$n_7 \times N_{\text{el}}^{\text{VLFS}} \times N_{\text{el}}^{\text{VLFS}}$	$N_{\text{ts}} \times N_{\text{el}}^{\text{HFS}} \times N_{\text{el}}^{\text{HFS}}$
Output: Saturation map $\hat{S}$	$N_{\text{ts}} \times N_{\text{el}}^{\text{HFS}} \times N_{\text{el}}^{\text{HFS}}$	

Table 3.2: CNN hyperparameters used in each phase: learning rate η, epoch, weight decay ψ, factor γ, and minimum learning rate η_{min}.

	Learning Rate η, ($\times 10^{-3}$)	Epoch	Weight Decay ψ, ($\times 10^{-4}$)	Factor γ	Minimum Learning Rate η_{min} ($\times 10^{-6}$)
Training 1	1.0	300	0.01	0.6	1.50
Training 2	1.0	300	0.01	0.6	1.55
Training 3	1.0	300	0.01	0.6	1.55
Training 4	5.0	300	5.0	0.6	1.50
Training 5	5.0	300	5.0	0.6	1.50

3.4 Results

Once trained on six-hours worth of data (in this example, on 65 HFS, 130 LFS, and 300 VLFS), the CNN surrogate provides an accurate approximation of the PDE solution on the fine mesh (Figure 3.2); this is a nontrivial feat since the PDE under consideration is highly nonlinear and its solution exhibits localized dynamic fronts. A typical forward pass of the CNN surrogate takes less than a second, whereas the corresponding run of the PDE solver on the fine mesh requires over 220 seconds. Ensemble-based computations, such as uncertainty quantification, make the most of this two orders of magnitude speed-up.

3.4.1 Model Performance

We compare the accuracy (RMSE on test data) of the CNN trained on the three levels of the multifidelity data and the CNNs trained on purely HFS, LFS, and VLFS data. To train the high-resolution (128×128 output) CNN solely on the LFS (64×64) or VLFS (32×32) data, the latter have to be downscaled to match the dimensions. We do so by taking the Kronecker product of a 64×64 LFS image and a 2×2 matrix of ones, or taking the Kronecker product of a 32×32 VLFS image and a 4×4 matrix of ones. The transformed VLFS and LFS data have the desired dimensions, while containing the same information as the original image. The test data are composed of HFS images (PDE solves on the fine mesh) that were not used for CNN training.

Figure 3.3 reveals that, if the data-generation budget does not exceed 20 hours, the CNN trained on the LFS data outperforms its HFS-trained counterpart in terms of RMSE. The CNN trained

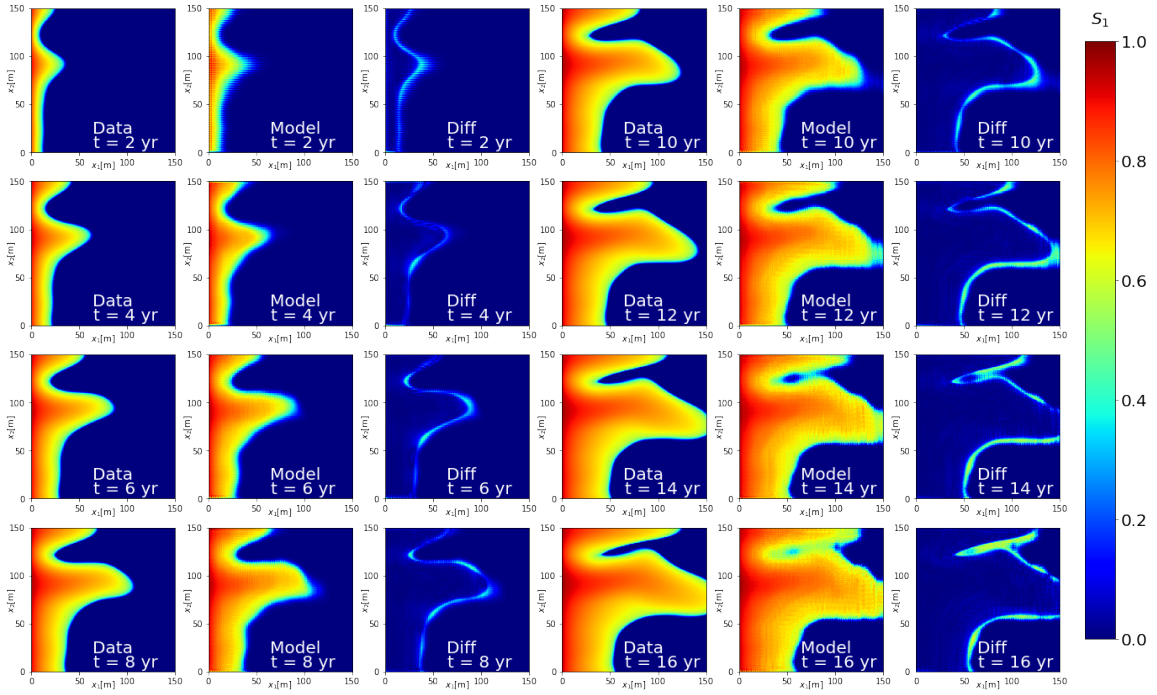

Figure 3.2: Temporal snapshots of the saturation maps $S_1(\mathbf{x}, t)$ for the permeability field $k(\mathbf{x})$ generated using the correlation length of $\lambda_Y = 19$ m in Figure 2.2. These are generated with either HFS of the PDE model Equation (2.2) and Equation (2.4) (labeled as S in the first and fourth columns) or the CNN surrogate trained on the three levels of data (labeled as $\hat{S}$) in the second and fifth columns). The third and sixth columns display absolute difference between the two predictions, $|S - \hat{S}|$.

on the VLFS data outperforms its LFS-trained counterparts when the data-generation budget does not exceed 4 hours. That is because such frugal budgets do not allow for generation of sufficient amounts of HFS and LFS data, respectively. As the budget increases, the error of the VLFS and LFS data precludes RMSE of the CNN trained on such data from dropping below 0.125 while RMSE of the HFS-trained CNN continues to decrease. This finding is reminiscent of the cost-constrained selection between high- and low-fidelity models in the context of ensemble-based simulations (Yang et al., 2020; Sinsbeck and Tartakovsky, 2015). Figure 3.3 also demonstrates that, for a relatively small budget of six hours, the use of multifidelity data yields the CNN whose RMSE is appreciably smaller than RMSEs of the CNNs trained on either HFS, LFS, or VLFS data alone.

3.4.2 CNN Surrogates for Uncertainty Quantification

We investigate the utility of our CNN surrogates for uncertainty quantification. Our quantity of interest is the breakthrough time, T_{break}, at the $x_1 = 100$ m plane (Figure 2.2), with the term "breakthrough" defined as the saturation of the invading phase (S_1) exceeding 0.15. Given uncertainty in intrinsic permeability $k(\mathbf{x})$, a solution of Equation (2.2) and, hence, predictions of T_{break} are given in terms of their cumulative distribution functions (CDFs) or probability density functions (PDFs) (Figure 3.4). The numbers of HFS samples generated during either 6 or 12 hours of simulations are insufficient for MC simulations to converge, leading to the appreciable errors in estimation of PDF and CDF of T_{break}. The CNN trained on multifidelity data yields accurate estimates of these quantities, while requiring only six hours of data generation.

Figure 3.4 exhibits the CDF and PDF of T_{break} alternatively computed with HFS, LFS, and VLFS MC simulations and with the CNN trained on the multifidelity data. The distributions obtained via MC simulation consisting of 292 hours of HFS are treated as ground truth. The distributions obtained from 24 hours of LFS or VLFS involve a sufficient number of samples for the error to be attributable solely to the low resolution, i.e., to the disretization errors in solving PDEs.

In addition to visual comparison of the distributions, the alternative strategies for estimation

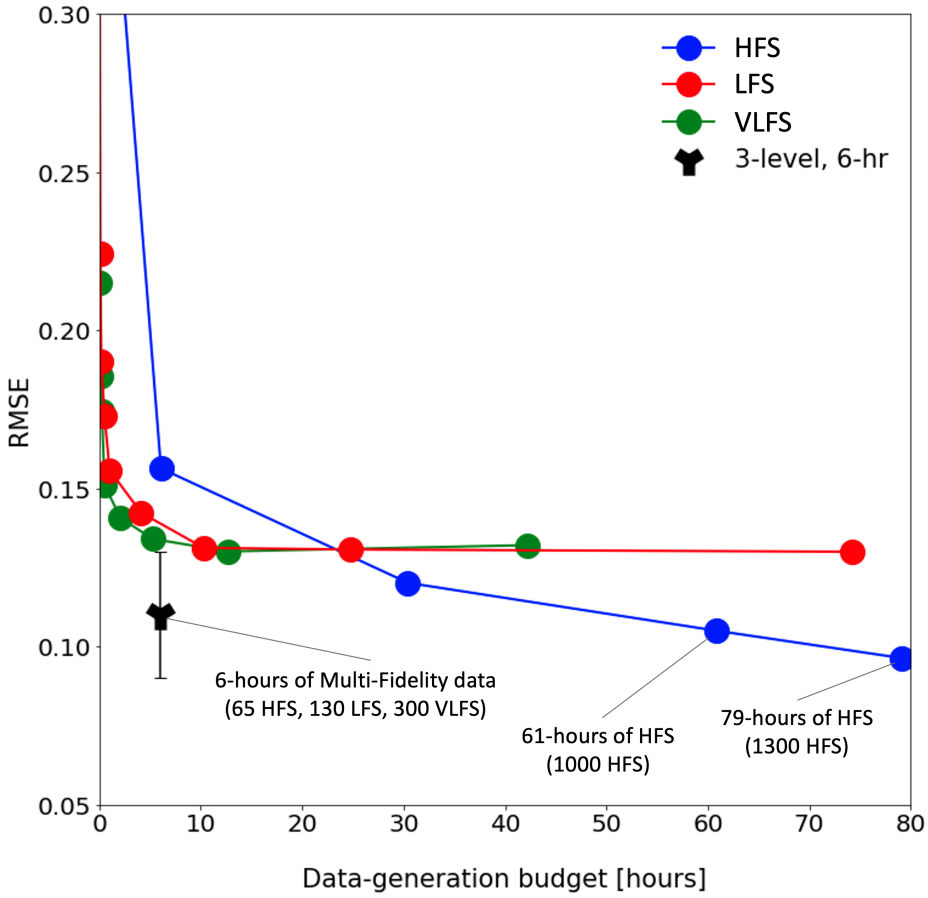

Figure 3.3: RMSE on test data for the alternative CNN training strategies plotted as function of the budget allocated for data generation. Each RMSE point in these graphs represents an average over ten iterations of training. The data point which represents the CNN trained on three levels of data also includes error bars based on 1 standard deviation. The graphic provides RMSE for the CNNs trained on strictly high-fidelity (blue circles), low-fidelity (red circles), very-low-fidelity (green circles), or multifidelity (tripoint star) data. The latter corresponds to the CNN trained on an optimal (the lowest RMSE) mix of high-, low-, and very-low-fidelity data for a set budget of six hours.

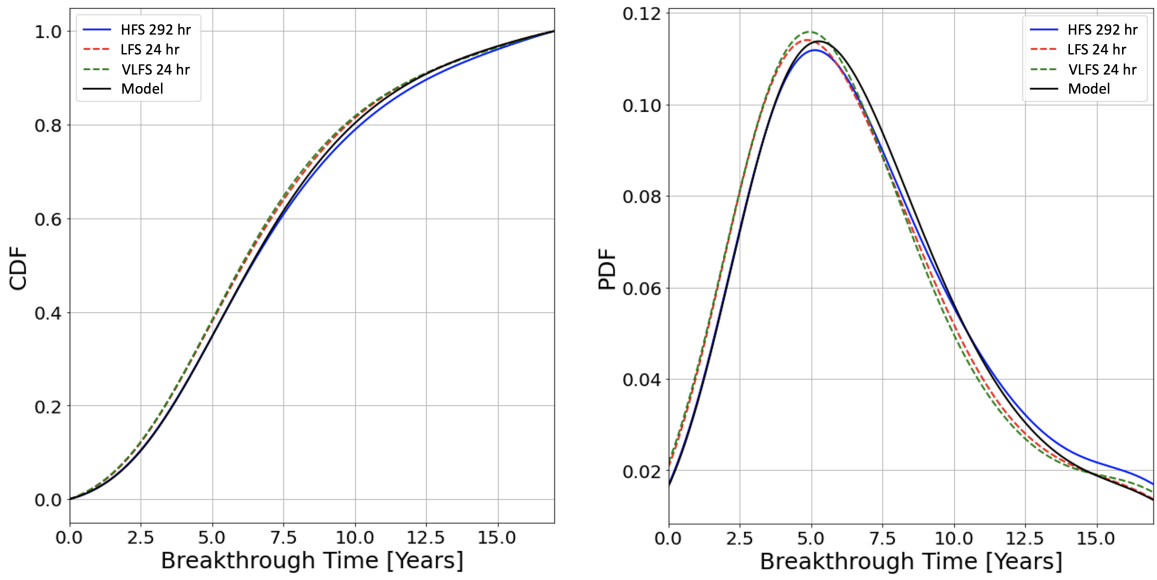

Figure 3.4: The converged CDF (left) and PDF (right) of breakthrough time is calculated using MC simulations of HFS, LFS, VLFS, and the CNN surrogate model.

of the distributions of T_{break} are compared in terms of RMSE, mean absolute error (MAE), the Kullback-Leibler (KL) divergence, and the Wasserstein distance. The UQ task was repeated 50 times, with Figure 3.5 displaying the mean and standard deviation of these measures of discrepancy. We found 3200 forward passes of the CNN to be sufficient for the CDF/PDF estimates to converge; this UQ task took seconds, whereas an equivalent HFS MC simulation takes 194 hours. In Chapter 2, the same operation took around ten minutes; we used the GPU for the forward pass to further reduce the time associated with ensemble computations. By every discrepancy measure, the CNN estimates outperform the converged LFS or VLFS MC simulations and are at least as accurate as the HFS MC simulation using 72 hours of data. Likewise, the CNN estimates are vastly more accurate than the HFS MC simulations of a similar data-generation budget.

3.4.3 CNNs Trained on Two-Levels or Three-Levels of Data

The method of training a CNN on two levels of data is promulgated in Chapter 2. In this chapter, we expand the method by training a CNN on three levels of data. To justify the added complexity

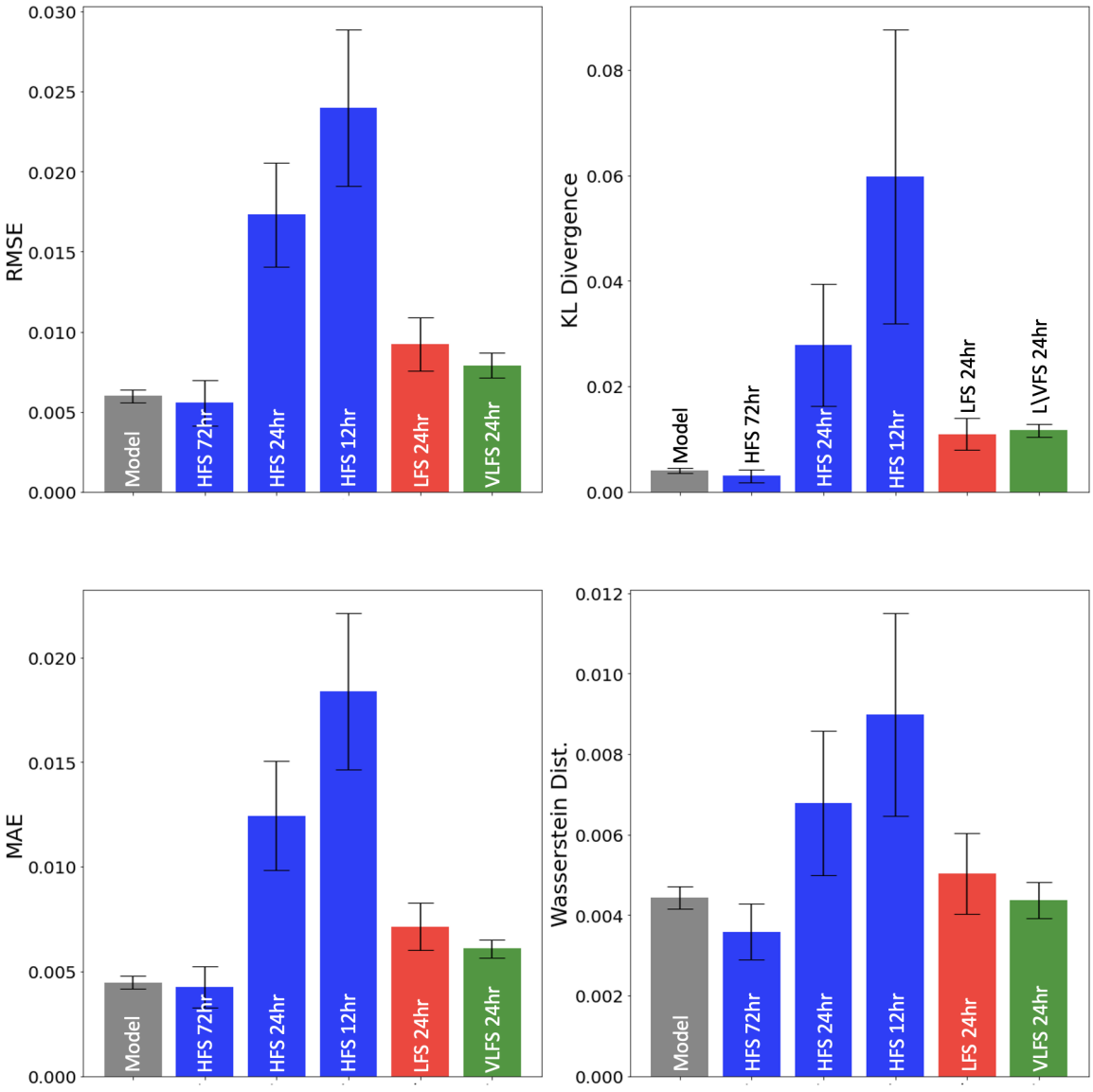

Figure 3.5: Comparison between PDFs of breakthrough times from MC simulations and the ground truth PDF (the ground truth PDF. was generated from MC using 292 hours of HFS). The compared PDFs include converged PDFs and nonconverged PDFs. The converged PDFs are generated from MC simulations of HFS, LFS, VLFS, and the CNN surrogate model (Figure 3.4). The nonconverged PDFs are generated from 12 hours and 24 hours of MC simulations using HFS. The comparisons are made using RMSE (top-left), MAE (bottom-left), KL divergence (top-right), and Wasserstein distance (bottom-right).

in the methodology, a direct comparison is made between training a CNN on three levels of data, as discussed in this chapter, and training a CNN on two levels of data, as discussed in Chapter 2. We make this comparison by evaluating the predictive accuracy of the CNNs trained on two levels of data and the CNNs trained on three levels of data at the same total data generation budgets.

Figure 3.6 compares the predictive performance, RMSE on the test data, of the following CNNs:

- the CNNs trained on two levels of data at six and twelve hours of training budget (Figure 2.10);
- the CNNs trained on three levels of data at six hours of training budget (Figure 3.3);
- the CNNs trained on three levels of data at twelve hours of training budget. In this experiment, the amounts of HFS, LFS, and VLFS in the training data are doubled their counterparts for the CNNs trained on six hours of three level data. The ratios between the HFS, LFS, and VLFS are kept constant.

Figure 3.6 shows a significant predictive performance gain by the CNNs trained on three levels data over the CNNs trained on two levels of data at six hours of data generation budget. However, at twelve hours of data generation budget, the two methods perform very similarly. The difference in RMSE at six hours of training data budget can be considered the direct gains of training on three levels of data as opposed to just two levels of data.

Figure 3.7 elucidates the significance of the gain in the predictive performance between the CNNs trained on three levels of data and CNNs trained on two levels data at six hours of data generation budget. This figure looks at the relationship between prediction accuracy and UQ performance across many experiments performed using the data set associated with the permeability fields generated using the correlation coefficient of $\lambda = 19$ m. The left panel of Figure 3.7 reveals a nonlinear relationship between the KL divergence and RMSE; this is important as an incremental gain in predictive performance of CNNs can significantly enhance the UQ capabilities of the CNNs. Consider, for example, the KL divergences in the neighborhood of RMSE values of 0.11 and 0.13, corresponding to the RMSE values of CNNs trained on three levels of data and RMSE values of CNNs trained

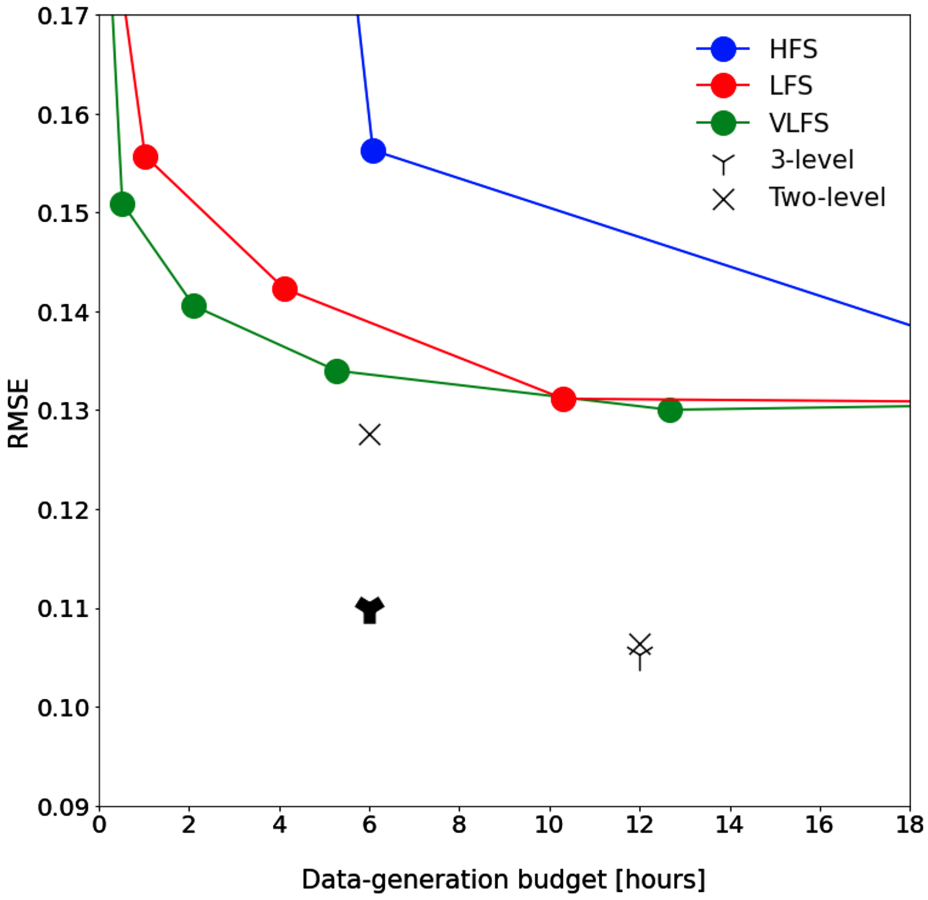

Figure 3.6: RMSE on test data for the alternative CNN training strategies plotted as function of the budget allocated for data generation. Each RMSE point in these graphs represents an average over ten iterations of training. The data points generated by CNNs trained on two levels of data are marked with ×s and the data points generated by CNNs trained on three levels of data are marked with tripoint stars. The datapoint represented by the bold tripoint star corresponds to the CNNs trained on an optimal (the lowest RMSE) mix of high-, low-, and very-low-fidelity data for a set budget of six hours. The graphic also provides RMSE for the CNNs trained on strictly high-fidelity (blue circles), low-fidelity (red circles), or very-low-fidelity (green circles).

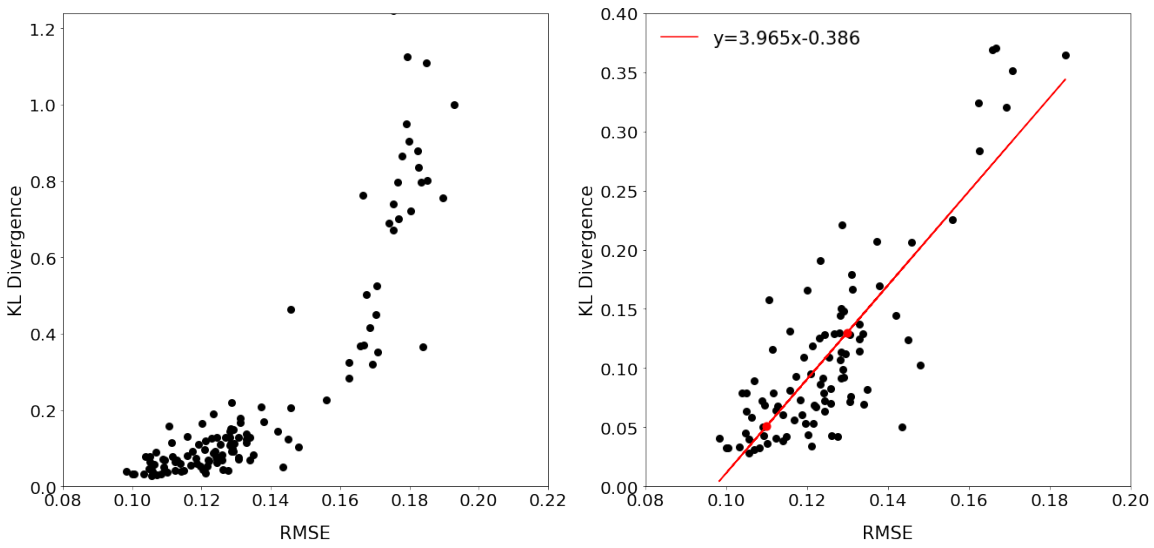

Figure 3.7: KL Divergence plotted as function of RMSE for many trained models. The left panel displays a zoomed out view, and the right panel displays a zoomed in view. The left panel displays a trend line through the data included in this panel (this data does not include the total data shown on the left panel), where y is KL divergence and x is RMSE.

on two levels of data, respectively. The trend line in Figure 3.7 yields the KL divergence values of 0.050 and 0.130 for RMSE values of 0.11 and 0.13, respectively; the KL divergence metric gain of 2.6 is contrasted with the RMSE improvement of 0.02. Although the zoomed out trend is nonlinear, a linear fit is a good tool to discern the relationships in the zoomed data.

3.5 Conclusions

We proposed an approach, based on transfer learning, to train a CNN on multifidelity (e.g., multiresolution) data. High-, low-, and very-low-fidelity images were generated by solving a PDE on fine, coarse, and very-coarse meshes, respectively. The performance of our algorithm was tested on a system of nonlinear parabolic PDEs governing multiphase flow in a heterogeneous porous medium with uncertain (random) permeability. A quantity of interest in this example is the PDF or CDF of the breakthrough time of an invading fluid. Our analysis leads to the following major conclusions.

1. When trained on three levels of multifidelity data, CNN surrogates provide an accurate approximation of the PDE solution on the fine mesh, even for highly nonlinear problems that exhibit sharp dynamic fronts. A forward pass of the CNN surrogate is two orders of magnitude faster than a PDE solution on the fine-mesh. This speed-up makes CNN surrogates an invaluable tool for ensemble-based computation of the PDF/CDF of a quantity of interest.

2. CNN training on three levels of multifidelity data reduces the data-generation budget fourteen-fold relative to CNN training on HFS data alone; two-fold reduction compared to the data-generation budget for of CNN trained on two levels of multifidelity data. If the budget is relatively small, the CNN trained on the VLFS or LFS data is more accurate than its HFS-trained counterpart. As the budget increases, the opposite is true. This finding is reminiscent of the cost-constrained selection between high- and low-fidelity models in the context of ensemble-based simulations.

3. For a small data-generation budget (six hours, in our example), the CNN trained on three levels of multifidelity data exhibits an appreciably smaller RMSE on test data than the CNNs trained on either HFS, LFS, or VLFS data. Performance of the multifidelity CNN depends on the ratio between HFS, LFS, and VLFS in the training set. Theoretical studies from MLMC can be used to guide the selection of an optimal mix of low- and high-fidelity data.

4. The CNN trained on multifidelity data is largely insensitive to the discretization error associated with LFS or VLFS. CNN-derived estimates of the PDF and CDF of the quantity of interest are close to those of converged high-fidelity MC simulations; but the former are three orders of magnitude faster to obtain than the latter (when using GPUs for the forward pass).

5. Even very small gains in the predictive accuracy of CNN surrogates can result in big improvements in the UQ performance.

Chapter 4

Upcycle Deployment of Pretrained Surrogates

Neural networks (NNs) are universal function approximators that can emulate the dynamics of complex systems. As such, surrogate models that mimic the forward solves of partial differential equations (PDEs) are powerful applications of NNs. A virtually negligible computational cost associated with the forward pass of a NN renders NN-based surrogates an attractive tool for ensemble-based computations, which otherwise require a large number of expensive numerical PDE solves. Since the latter provide the data needed to train the NNs, the utility of NN-based surrogates hinges on the balance between the training cost and the computational gain stemming from their deployment. We rely on transfer learning to train a deep CNN for a new task (solving a set of PDEs for solute transport) starting from a CNN that was originally trained for a different task (solving a different set of PDEs for multiphase flow). Our numerical experiments demonstrate that the CNN surrogate generalizes well to the inputs/outputs not seen during the training. The transfer learning scheme is effective in estimation of the distribution of a quantity of interest associated with advection-dispersion transport in a porous medium with uncertain/random conductivity fields. For a given

training-data set, the CNNs that utilize transfer learning exhibit equal or greater prediction accuracy than the CNNs whose training is initialized randomly. Furthermore, the ability of the CNN to recreate distributions was compared to MC simulations and the performance is expressed in Kullback–Leibler divergence and Wasserstein distance. The latter reveals that with only six hours of training data budget, a CNN originally trained to solve multiphase flow problems can he upcycled to accurately solve advection-dispersion problems and recreate distributions on the new problem. A randomly initialized CNN could not perform this task when trained the same data, and MC simulation could notd recreate the distributions when given a simulation computational time of six hours.

4.1 Introduction

Benefits of NNs as surrogates of PDE-based models are inextricably tied to the cost of NN training and generation of data necessary for this training. In Chapters 2 and 3, we introduced a strategy to reduce this cost by focusing on the data generation aspect of the surrogate construction. In this chapter, we tackle the other aspect, i.e., NN training, by starting the training process with a NN trained on an unrelated problem and then deploying transfer learning to build a CNN surrogate for the problem at hand. By way of example, we upcycle the CNN from Chapter 2, which was trained to solve multiphase flow in a heterogeneous porous medium, to build a surrogate for PDEs describing advection-dispersion transport in a different heterogeneous porous medium.

Section 4.2 contains a description of the CNN architecture and the workflow to train a CNN that solves PDEs for advection-dispersion transport starting from a CNN trained to solve PDEs for a multiphase flow. A set of numerical experiments aimed at testing the algorithm performance is described in Section 4.3. Results of these experiments, reported in Section 4.4, demonstrate the accuracy of the CNN-based surrogates, the computational advantages of transfer learning, and the computational efficiency of the CNN-based surrogate vis-à-vis MC simulations when used to quantify the predictive uncertainty of the transport model. The main conclusions drawn from this study are

collated in Section 4.5.

4.2 Deep Convolutional Neural Networks

The CNN-based surrogate is set up as an image-to-image regression task (Zhou and Tartakovsky, 2021). To train and test the network, we treat the inputs and outputs of Equation (1.1) as images. Specifically, the parameter values $\boldsymbol{\theta}(\mathbf{x}_i)$ in N_{el} elements $\{\mathbf{x}_i\}_{i=1}^{N_{\text{el}}}$ of a numerical grid serve as input and the discretized solution $\mathbf{u}(\mathbf{x}_i, t_k)$ of Equation (1.1) at N_{ts} time steps $\{t_k\}_{k=1}^{N_{\text{ts}}}$ is used as output. In order to train a generalized CNN sensitive to unseen sets of the input $\boldsymbol{\theta}(\mathbf{x}_i)$, i.e., to prevent over-fitting to a particular choice of $\boldsymbol{\theta}(\mathbf{x}_i)$, the training data often comprise a large number N_{train} of model runs. Generation of this training/test data-set is computationally expensive as each of its N_{train} members is a solution $\mathbf{u}$ of Equation (1.1) for different realizations $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{N_{\text{train}}}\}$ of the input $\boldsymbol{\theta}$. The performance of the CNN-based surrogate is evaluated using the loss function

$$\mathbf{L}(\mathbf{w}) = \sum_{m=1}^{N_{\text{train}}} \sum_{i=1}^{N_{\text{el}}} \sum_{k=1}^{N_{\text{ts}}} |\mathbf{u}(\mathbf{x}_i, t_k; \boldsymbol{\theta}_m) - \hat{\mathbf{u}}_{ik}(\mathbf{w}; \boldsymbol{\theta}_m)| + R \sum_{n=1}^{N_{\mathbf{w}}} w_n^2. \tag{4.1}$$

The first term in this loss function is the L_1-norm discrepancy between the state variables $\mathbf{u}(\mathbf{x}_i, t_k)$ predicted by solving Equation (1.1) and its CNN approximation, $\hat{\mathbf{u}}_{ik}(\mathbf{w})$, with $N_{\mathbf{w}}$ weights $\mathbf{w} = (w_1, \ldots, w_{N_{\mathbf{w}}})^\top$. The second term is the L_2-norm regularization, which mitigates over-fitting by penalizing large weights $\mathbf{w}$ associated with complex models; the regularization parameter R prescribes how strongly the regularization penalty is applied. The CNN training process starts with weights $\mathbf{w}$ and optimizes to $\mathbf{w}^\star$ by minimizing $\mathbf{L}$.

4.2.1 Transfer Learning

Our data are comprised of solutions of Equation (1.1), $\mathbf{u}(\mathbf{x}_i, t_k)$, obtained on a mesh that matches the dimension of the input $\boldsymbol{\theta}(\mathbf{x}_i)$. Transfer learning allows us to break apart the training process and initially train a subset of $N_{\mathbf{w},\text{train}}$ weights composed of trainable weights $\mathbf{w}_{\text{train}}$ on $\mathbf{u}$. The resulting

CNN contains $N_{\mathbf{w},\text{locked}}$ nontrainable weights $\mathbf{w}_{\text{locked}} = (w_1, \ldots, w_{N_{\mathbf{w},\text{locked}}})^\top$ and $N_{\mathbf{w},\text{train}}$ trainable weights $\mathbf{w}_{\text{train}} = (w_{N_{\mathbf{w},\text{locked}}+1}, \ldots, w_{N_\mathbf{w}})$, such that $N_{\mathbf{w},\text{train}} = N_\mathbf{w} - N_{\mathbf{w},\text{locked}}$. The network is trained and the trainable weights $\mathbf{w}_{\text{train}}$ are updated. Next, updating is enabled in all the weights such that $N_{\mathbf{w},\text{train}} = N_\mathbf{w}$. We then allow the entire set of weights $\mathbf{w}$ update on the same training data $\mathbf{u}$. The use of a CNN pretrained on different data as initialization of the training process helps prevent over-fitting by not updating all of the weights on the new data.

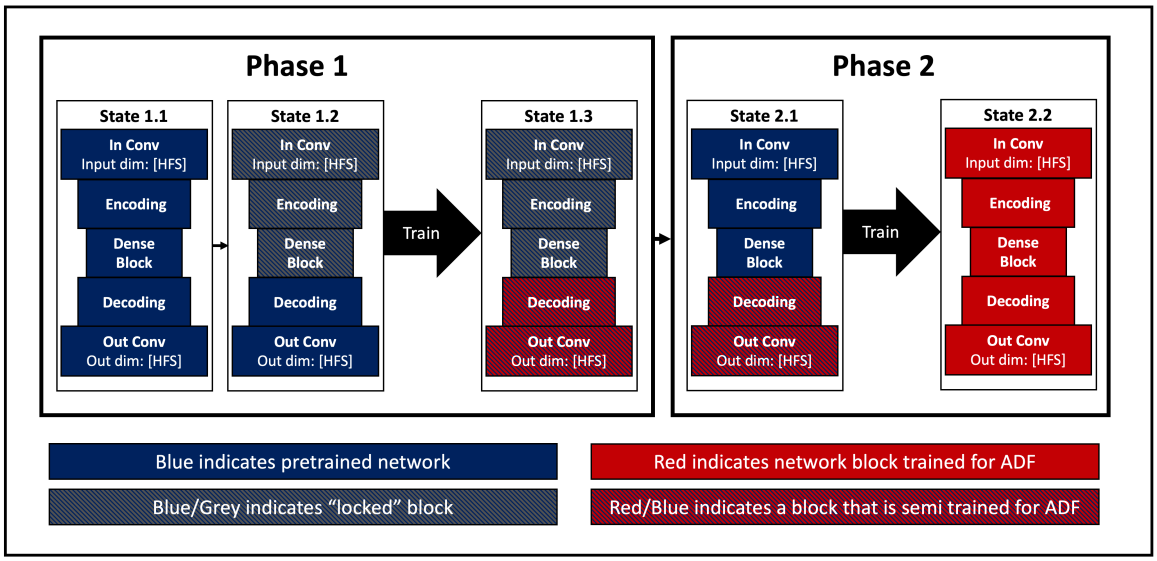

Figure 4.1: Workflow for retraining a pretrained CNN to perform a new task. Phase 1 trains the last blocks of the CNN using the new data. Phase 2 trains the entire CNN on the same data.

4.2.2 Workflow for Retraining CNN for a New Task

Our strategy for retraining a pretrained CNN to solve a new problem consists of two phases (Figure 4.1), each of which results in a CNN denoted by M_i $(i = 1, 2)$. We start the training process with a pretrained CNN denoted by M_0. In Phase 1, the CNN M_0 locks the weights in $\mathbf{w}_{\text{locked}}$ from updating while allowing $\mathbf{w}_{\text{train}}$ to train. This modified CNN is trained on $\mathbf{u}$ to form M_1. During Phase 2, all of the weights of M_2 are allowed to update and trained to form M_2. Both phases use the same training data. The numerical experiments in Sections 4.3 and 4.4 demonstrate that

this transfer learning strategy significantly lowers the number of simulation runs required to train functional CNN surrogates.

Phase 1: Train a CNN M_1, with N_w weights $\mathbf{w} = \{\mathbf{w}_\mathrm{locked}, \mathbf{w}_\mathrm{train}\}$ of which only $N_{\mathbf{w},\mathrm{train}}$ weights $\mathbf{w}_\mathrm{train}$ are updated during the training process, on the data $\mathbf{u}$

State 1.1: Initialize M_0, the model pretrained to a different task then lock $\mathbf{w}_\mathrm{locked}$ to form M_1

State 1.2: Train the CNN M_1 on the HFS data by minimizing the loss function in Equation (4.1) over the weights $\mathbf{w}_\mathrm{HFS}$ of layer L_last, while keeping the remaining weights $\mathbf{w}_\mathrm{locked}$ locked at their values in M_1.

Phase 2: Train the CNN M_2 on the HFS data by allowing all weights $\mathbf{w}$ of M_1 to vary during the minimization

4.3 Application to Advection-Dispersion Transport

Horizontal flow in an aquifer Ω with hydraulic conductivity $K(\mathbf{x})$ and porosity $\phi(\mathbf{x})$ is described by a two-dimensional steady-state groundwater flow equation,

$$\nabla \cdot (K \nabla h) = 0, \qquad \mathbf{x} \equiv (x_1, x_2)^\top \in \Omega, \tag{4.2}$$

subject to appropriate boundary conditions. A solution to this equation yields the spatial distribution of the hydraulic head $h(\mathbf{x})$; numerical differentiation of this solution gives the macroscopic flow velocity $\mathbf{v}(\mathbf{x}) = (v_1, v_2)^\top$:

$$\mathbf{v} = -\frac{K}{\phi} \nabla h. \tag{4.3}$$

The two-dimensional computational domain Ω is a 150 m $\times$ 150 m square (Figure 4.3). The top (Γ_t or $x_2 = 150$ m) and bottom (Γ_b or $x_2 = 0$) boundaries are impermeable, and flow is driven by the

difference in the hydraulic heads imposed along the left (Γ_l or $x_1 = 0$) and right (Γ_r or $x_1 = 150$ m) boundaries:

$$\frac{\partial h}{\partial x_2} = 0, \quad \mathbf{x} \in \Gamma_\mathrm{b} \cup \Gamma_\mathrm{t}; \quad h(x_1 = 0 \text{ m}, x_2) = 15 \text{ m}; \quad h(x_1 = 150 \text{ m}, x_2) = 0. \tag{4.4a}$$

Starting with initial time $t = 0$, a contaminant with volumetric concentration c_s is continuously released into the aquifer at location $\mathbf{x}_s \in \Omega$. This point source has constant intensity q_s. The spatiotemporal evolution of the contaminant concentration in groundwater, $c(\mathbf{x}, t)$, is described by an advection-dispersion equation,

$$\frac{\partial \phi c}{\partial t} = \nabla \cdot (\mathbf{D} \nabla c) - \nabla \cdot (\mathbf{v} c) + q_s c_s \delta(\mathbf{x} - \mathbf{x}_s), \tag{4.5}$$

where $\delta(\cdot)$ is the Dirac delta function. In the principle coordinate system aligned with the mean flow direction ($\mathbf{v} = (v \equiv |\mathbf{v}|, 0)^\top$), the dispersion coefficient tensor, $\mathbf{D}$, has components

$$D_{11} = \phi D_m + \alpha_L v, \quad D_{22} = \phi D_m + \alpha_T v, \quad D_{12} = D_{21} = \phi D_m, \tag{4.6}$$

where α_T and α_L are the transverse and longitudinal dispersivities, respectively, and D_m is the molecular diffusion coefficient of the contaminant in water.

The initial concentration of the contaminant is

$$c(\mathbf{x}, t = 0) = 0, \qquad \mathbf{x} \in D. \tag{4.7}$$

The concentration boundary conditions are Neumann boundary conditions where concentrations do not change across the boundaries:

$$\frac{\partial c}{\partial x_2} = 0, \quad \mathbf{x} \in \Gamma_\mathrm{b} \cup \Gamma_\mathrm{t}; \quad \frac{\partial c}{\partial x_1} + \frac{v_1}{\phi} c = 0, \quad \mathbf{x} \in \Gamma_\mathrm{l} \cup \Gamma_\mathrm{r} \tag{4.8}$$

The contamination source is located at $\mathbf{x}_s = (x_1 = 15 \text{ m}, x_2 = 75 \text{ m})^\top$. The contaminant release intensity is $q_s = 50 \text{ m}^2/\text{d}$, and the contaminant concentration at the release site is $c_s = 10 \text{ g/m}^3$.

4.3.1 Data Generation

With the exception of hydraulic conductivity $K(\mathbf{x})$, all the model parameters are assumed to be constant and known with certainty. The logarithm of the uncertain conductivity $K(\mathbf{x})$ is modeled as a stationary multivariate random field with mean $\mu = -1.91[\ln(\frac{m}{d})]$, variance $\sigma^2 = 0.2[(\ln(\frac{m}{d}))^2]$, and the exponential covariance function; the latter has correlation lengths 17.57 m (15 cell-lengths) and 2.34 m (2 cell-lengths) in the x_1 and x_2 directions, respectively (Heße et al., 2014). Realizations of this field are generated using GSTools (Müller et al., 2022), a Python-based geostatistics package, on the uniform 128×128 mesh. A representative realization of $\ln K(\mathbf{x})$ is displayed in Figure 4.2.

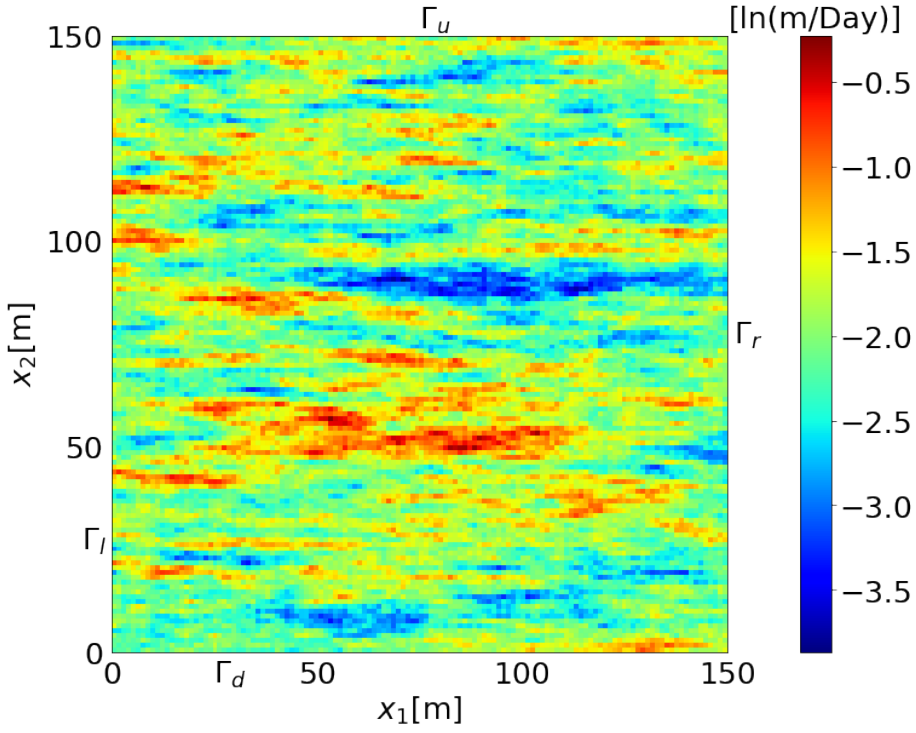

Figure 4.2: A representative realization of log conductivity field $Y = \ln K$ on the 128×128 grid. K is expressed in m/d.

The contaminant transport model, Equations (4.2)–(4.6), are solved on the same 128×128 mesh, for the parameter values provided in Table 4.1. `Flopy` (Bakker et al., 2016), a Python implementation of MODFLOW (Harbaugh, 2005) and MT3DMS (Bedekar et al., 2016), is used to solve the flow, (4.2), and transport, (4.5), equations, respectively.

Table 4.1: values of the transport parameters used for data generation.

Parameter	Value	Units
Porosity, ϕ	0.2	-
Molecular diffusion, D_m	10^{-9}	m^2/d
Longitudinal dispersivity, α_L	0.01	m
Transverse dispersivity, α_T	0.1	m

The 128×128 concentration maps $c(\mathbf{x}, t)$ from 16 time-steps of interest are saved for each realization of the 128×128 corresponding conductivity field $K(\mathbf{x})$. When multiple conductivity fields $K(\mathbf{x})$ and corresponding concentration maps $c(\mathbf{x}, t)$ are generated; they represent input and output of the training/test data, respectively.

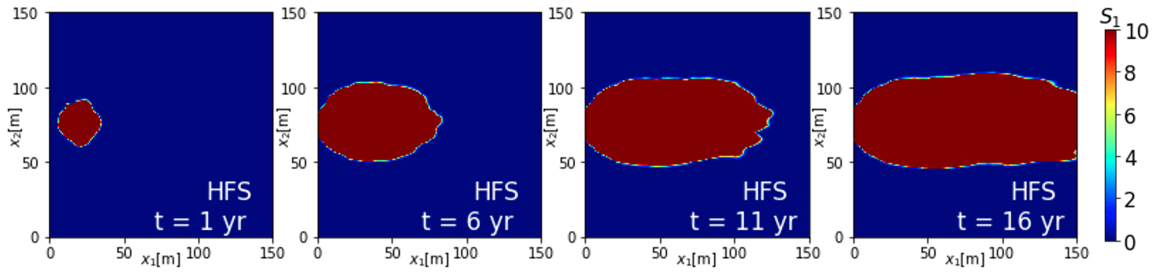

Figure 4.3: Temporal snapshots of contaminant concentration $c(\mathbf{x}, t)$ for the conductivity field $K(\mathbf{x})$ in Figure 4.2.

The forward solves were completed using a computer with an Intel Core i7-4793 3.6Ghz processor and 64GB of RAM. On average, each forward pass required 216 seconds to complete. The forward passes are saved as image files and used to train the CNN; the time needed to generate a dataset is henceforth referred to as "data-generation budget". Figure 4.3 provides snapshots of the solved forward pass for the conductivity field $K(\mathbf{x})$ in Figure 4.2.

4.3.2 CNN Architecture

Table 2.2 describes the specific input and output volumes of the CNN architecture used in the implementation of our approach. `PyTorch` is deployed for the CNN construction and training. The training/testing computations were executed on the Stanford Mazama high-performance computing cluster. The computing resources used for this study include Nvidia V100 GPU with 16GB vRAM, Intel Xenon Gold 6126 CPU (2.6 GHz), and 60GB RAM.

Table 4.2: Dimensions of the model blocks and input/output volume of each model block. The number of time steps is $N_{\text{ts}} = 16$; the number of elements the meshes are $N_{\text{data}} \times N_{\text{data}} = 128 \times 128$; the number of elements in the output of the intermediate blocks is $N_{\text{int}} = 64$; and the number of elements in the output of the dense block is $N_{\text{dense}} = 32$. The CNN uses 7x7 filters and the number of channels in each of the seven blocks of the CNN is $n_1 = 64$, $n_2 = 344$, $n_3 = 172$, $n_4 = 652$, $n_5 = 326$, $n_6 = 606$, and $n_7 = 303$.

Layer	Input	Output
Input: Conductivity field $K(\mathbf{x})$	$1 \times N_{\text{data}} \times N_{\text{data}}$	
Convolution 1	$n_1 \times N_{\text{data}} \times N_{\text{data}}$	$n_2 \times N_{\text{int}} \times N_{\text{int}}$
Dense Block (Encoding)	$n_2 \times N_{\text{int}} \times N_{\text{int}}$	$n_3 \times N_{\text{int}} \times N_{\text{int}}$
Convolution 2	$n_3 \times N_{\text{int}} \times N_{\text{int}}$	$n_4 \times N_{\text{dense}} \times N_{\text{dense}}$
Dense Block	$n_4 \times N_{\text{dense}} \times N_{\text{dense}}$	$n_5 \times N_{\text{dense}} \times N_{\text{dense}}$
Convolution Transpose 1	$n_5 \times N_{\text{dense}} \times N_{\text{dense}}$	$n_6 \times N_{\text{int}} \times N_{\text{int}}$
Dense Block (Decoding)	$n_6 \times N_{\text{int}} \times N_{\text{int}}$	$n_7 \times N_{\text{int}} \times N_{\text{int}}$
Convolution Transpose 2	$n_7 \times N_{\text{int}} \times N_{\text{int}}$	$N_{\text{ts}} \times N_{\text{data}} \times N_{\text{data}}$
Output: Saturation map $c(\mathbf{x})$	$N_{\text{ts}} \times N_{\text{data}} \times N_{\text{data}}$	

4.3.3 Training initialization

Transfer learning is common in the field of computer vision. The workflow often starts from a well-known high-performing trained NN and modifications are made to complete new tasks. In this study, the encoder-decoder CNN trained for the multiphase flow problem in Chapter 2 is used as, and henceforth referred to as, the pretrained model.

The details of the pretrained model can be found in Chapter 2 and in Song and Tartakovsky (2021); some information is replicated here for the sake of convenience. The pretrained CNN has the architecture described in Section 4.3.2. The multiphase flow data used for its training and testing are of the same dimensions as the data used in this study. Both the multiphase flow and

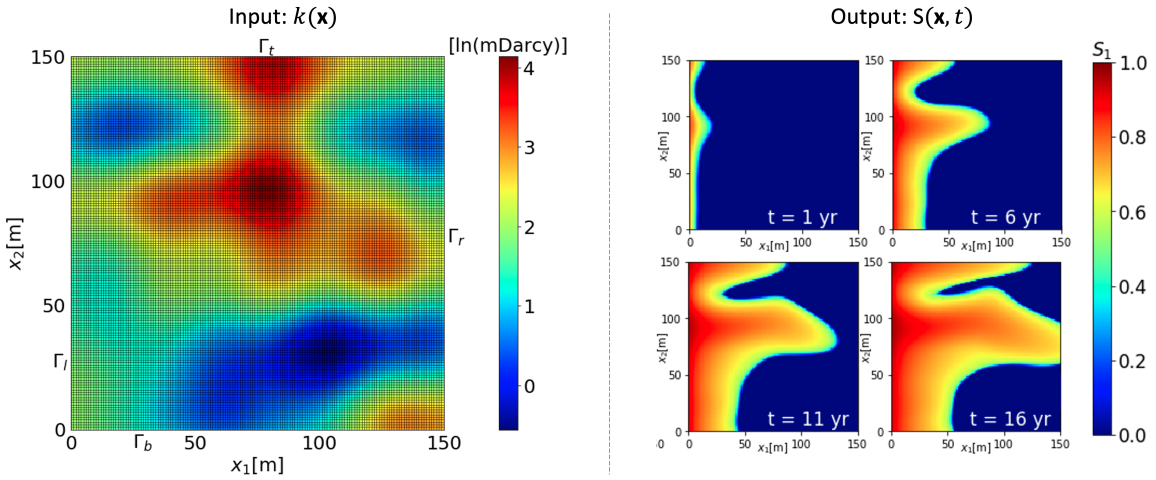

Figure 4.4: Sample data used to train/test the pretrained CNN from Song and Tartakovsky (2021). Left: 128×128 permeability field $k(\mathbf{x})$. Right: Temporal snapshots of four 128×128 saturation maps $S(\mathbf{x}, t)$; actual data sets contain 16 snapshots each.

advection-dispersion transport problems have impermeable top and bottom boundaries and a higher head/pressure on the left boundary than the right boundary resulting in a net flow from left to right. However, the physical phenomena and their mathematical descriptions that are simulated to form the data sets are fundamentally different. The multiphase flow problem consists of tightly coupled nonlinear PDEs, while the solute transport problem involves sequentially coupled linear PDEs. The differences in their solutions are manifest upon a visual inspection of the two data sets; a representative data set for the pretrained CNN is provided in Figure 4.4.

To investigate the effectiveness of our transfer learning strategy, we compare its prediction of the concentration maps $c(\mathbf{x}, t)$ with those obtained by the `PyTorch` CNNs whose training started with the default random initialization (Paszke et al., 2017). Without any training, feeding the hydraulic conductivity $K(\mathbf{x})$ from Figure 4.2 into either the pretrained CNN or a randomly initialized CNN yields predictions of $\hat{c}(\mathbf{x}, t)$ that have nothing in common with the concentration maps $c(\mathbf{x}, t)$ computed by solving PDEs (4.2)–(4.6) (Figure 4.5). This failure is to be expected since neither CNN has seen data representative of solute transport. The question we address is whether the pretrained CNN needs fewer training data than the randomply initialized CNN does.

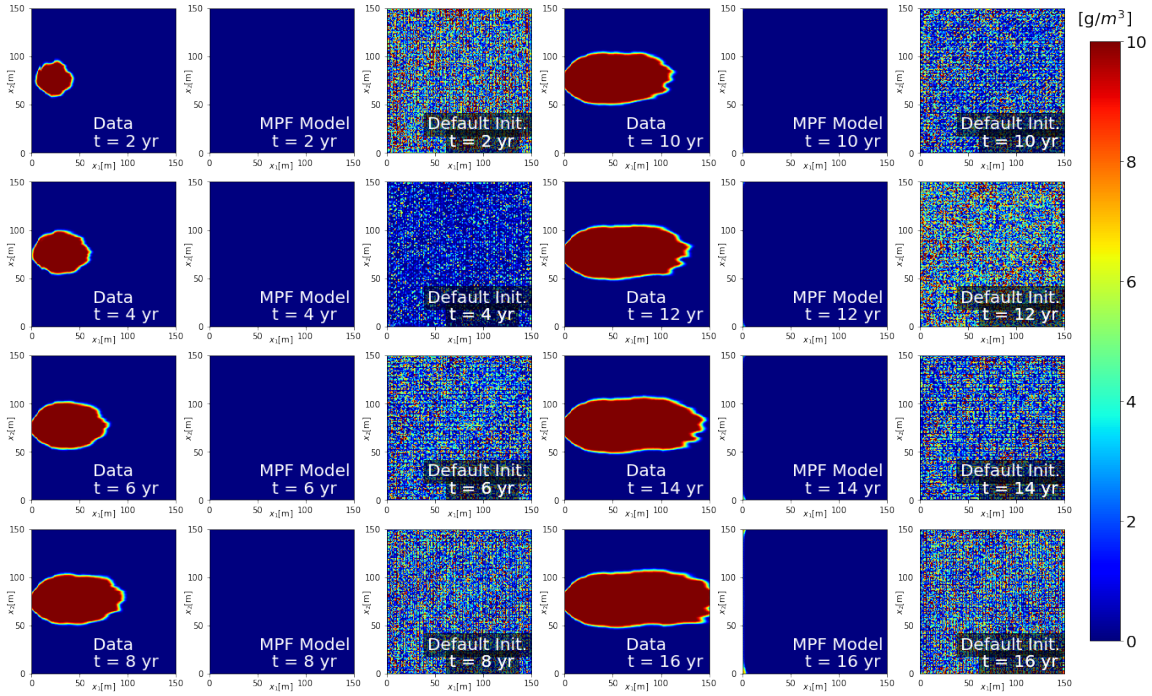

Figure 4.5: Temporal snapshots of the concentration maps alternatively predicted by solving PDEs (4.2)–(4.6) with the hydraulic conductivity field $K(\mathbf{x})$ from Figure 4.2 (the first and fourth columns) and by feeding this $K(\mathbf{x})$ into the pretrained CNN (second and fifth columns) or a randomly initialized CNN (third and sixth columns).

The following section provides some details of the training of these two CNNs on the $c(\mathbf{x}, t)$ data obtained by solving PDEs (4.2)–(4.6) for different realizations of $K(\mathbf{x})$.

4.3.4 Hyperparameter Optimization

Learning rate η, weight decay ψ, factor γ, and minimum learning rate η_{min} are hyperparameters that can impact the success of a CNN training program. The parameters η and ψ impact the Adam optimizer (Kingma and Ba, 2014), and the parameters γ and η_{min} impact the `ReduceLROnPlateau` scheduler. Many other hyperparameters, e.g., ψ that is responsible for determining the regularization parameter R (Loshchilov and Hutter, 2017), can be changed to influence the CNN training performance, but we use their default `PyTorch` values. More information on the individual hyperparameters, optimizers, and schedulers are available `PyTorch` documentation (Paszke et al., 2019).

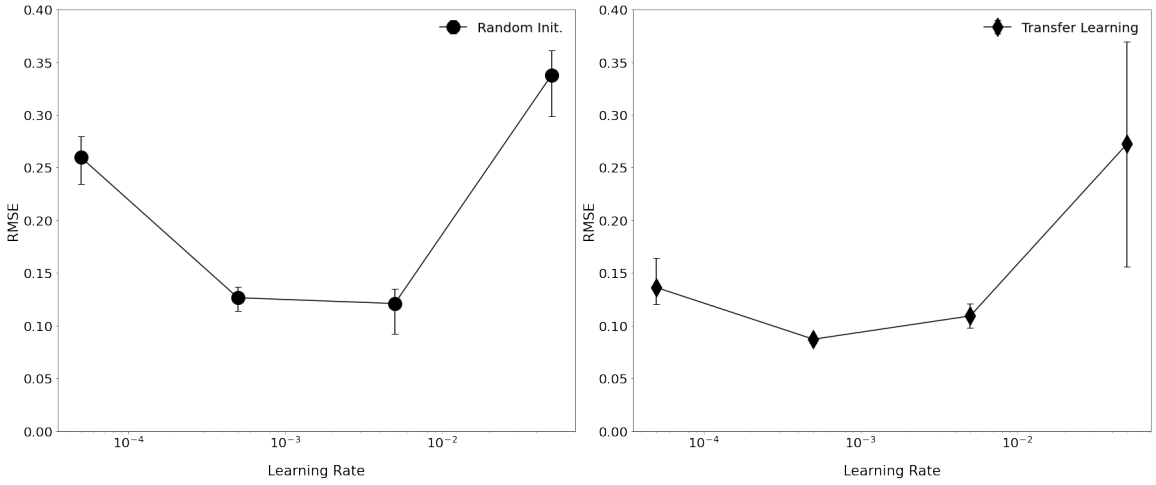

Figure 4.6: Model performance in the neighborhood of the optimum learning rates for the control/randomly initialized CNN (left) and the CNN based on our transfer-learning scheme (right). Each data point is the average over five test runs and the error bars represent a 68% confidence bound.

The optimization starts by using the hyperparameters used in Chapter 2 as initial guesses. The search iterates, in order, through variations of η, ψ, γ, and η_{min}. Once the parameter associated with the lowest RMSE has been found, the search moves to the next hyperparameter. A robust grid

Table 4.3: Non-default parameters

Parameter	Value
Learning rate η (Transfer-learning)	$5 \cdot 10^{-4}$
Learning rate η (Control)	$5 \cdot 10^{-3}$
Weight decay ψ	10^{-5}
Factor γ	0.6
Minimum learning rate η_{min}	$5 \cdot 10^{-6}$
Epochs	150

search could result in a more optimized set of hyperparameters. Our search required 100 data points; corresponding to a data budget of six hours. Each training pass involved 150 epochs, which used about 0.33 hours. The 16 training passes (four iterations per each of the four hyperparameters) took 5.3 and 10.5 training-hours to find functional hyperparameters for the randomly initialized CNN and the CNN trained with transfer-learning, respectively; the transfer-learning scheme has two training steps, as described in Section 4.1. The training-hours correspond to considerably smaller wall-clock time, since this task is parallelized across several GPU nodes. For this example, η proved to be the only hyperparameter that differs from its counterpart used in Chapter 2.

Non-default values of the hyperparameters are reported in Table 4.3, and the η space that was explored is shown in Figure 4.6. It is worthwhile noting that the value of η that yields on optimal performance of the retrained CNN is an order of magnitude smaller than the optimal η for the randomly initialized CNN. This suggests that large values of η may negate the benefits of transfer learning by training "over" the initially well-trained weights. This observation is supported by the right panel in Figure 4.6, wherein the $\eta = 5 \cdot 10^{-2}$ is associated with a high RMSE with large confidence bounds.

4.4 Results

Once trained on only six hours of the $c(\mathbf{x}, t)$ data, the updated pretrained CNN provides an accurate approximation of the PDE solution, including "bumps" at the plume edge caused by the subsurface heterogeneity (Figure 4.7). Each PDE solve takes nearly 220 seconds, while a comparable CNN

prediction takes less than a second. This more than two-orders of magnitude speed-up greatly facilitates ensemble computations, such as UQ in Section 4.4.2.

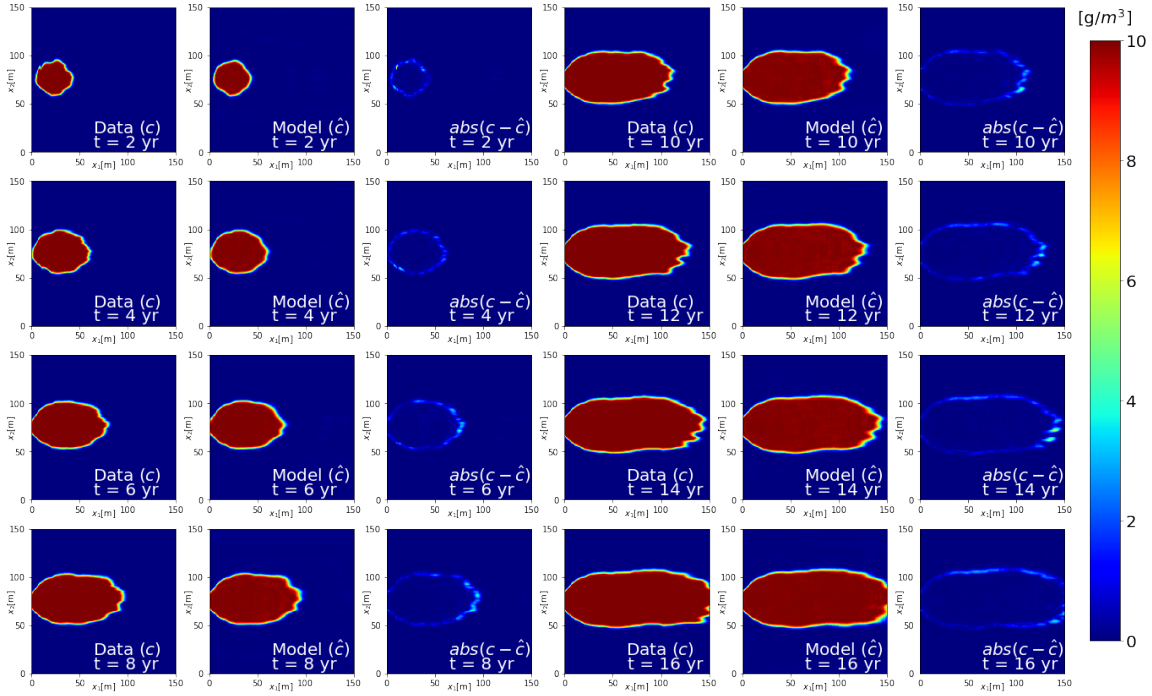

Figure 4.7: Model response after transfer learning. Snapshots of concentration maps $c(\mathbf{x}, t)$ corresponding to the conductivity field $K(\mathbf{x})$ in Figure 4.2. The PDE model defined by Equations (4.2)-(4.6) is solved to generate the concentration data $c(\mathbf{x}, t)$ in the first and fourth columns. The CNN's prediction of the concentration map $\hat{c}(\mathbf{x}, t)$ is shown in the second and fifth columns. The difference between the two, $|c - \hat{c}|$, is shown in the third and sixth columns.

4.4.1 Model Performance

The left panel of Figure 4.8 exhibits the RMSEs of the estimates of solute concentration $c(\mathbf{x}, t)$ obtained via the two CNNs trained on data with varying data-generation budgets. The pretrained network trained via transfer learning achieves more accurate results on the test data for every training-data budget. The randomly initialized CNN needs 30 hours of data-generation budget to match the RMSE of the CNN trained via transfer learning on six hours worth of training data; a factor of five reduction in the data-generation requirements. As the data-generation budget increases

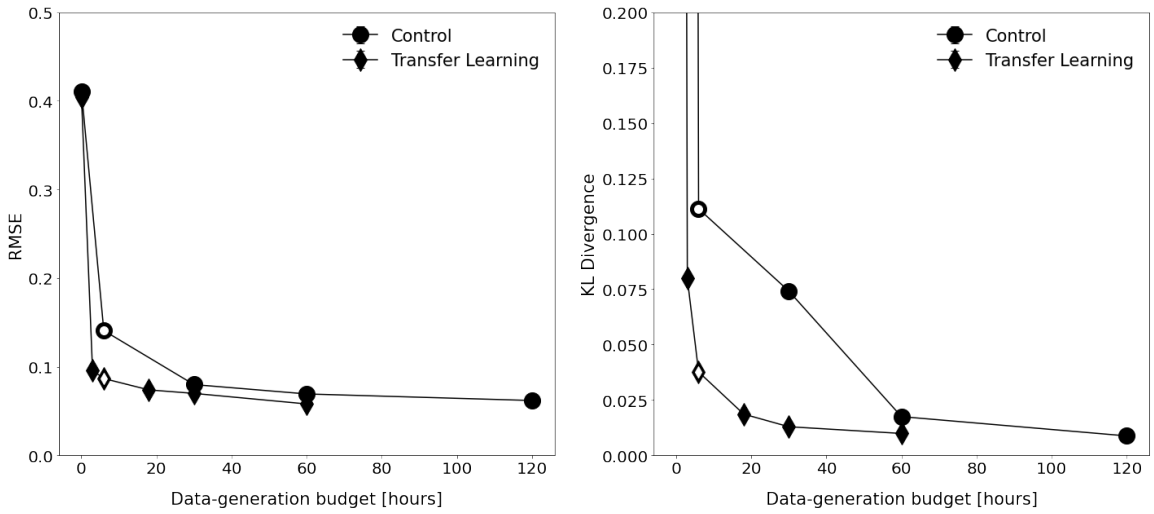

Figure 4.8: CNN performance on test RMSE (left) and on uncertainty quantification task (right) for two alternative CNN training strategies. The randomly initialized CNN (control) is marked by circles and the transfer-learning based CNN by diamonds. Both graphs are plotted as function of the training data budget. Each point on both graphs represents the average over five iterations of training. The data points corresponding to six hours of training are marked by open symbols; this represents the greatest performance difference between the two methods for any given data budget.

beyond 30 hours, the two methods of training to converge to the same RMSE of 0.08.

The right panel of Figure 4.8 shows the KL divergence for the PDF of the breakthrough time T_{break} (see Section 4.4.2) estimated via the two CNNs alternatively trained on the data with varying data-generation budgets. The CNN trained via transfer learning yields lower KL divergence, i.e., is more accurate, than the randomly initialized CNN does, for every training data budget. The latter CNN needs a bit less than 30 hours of data-generation budget to reach the same KL divergence as that of the transfer learning-based CNN trained on six hours worth of training data. In other words, our transfer learning strategy reduces the data requirements by a factor of five. As the budget increases beyond 30 hours of training data, the two methods of training converge to the KL divergence of 0.07.

Figure 4.8 demonstrates that the estimation accuracy of CNN surrogates is related to their performance on a UQ task. This relationship is nonlinear, such that a CNN with even a slightly smaller RMSE can yield much better UQ performance; an accurate CNN surrogate is critical to the

UQ performance.

4.4.2 CNN Surrogates vs Monte Carlo Simulations

We compare the performance of our CNN-based surrogate with that of Monte Carlo solution to the PDE-based model. The UQ task here is to estimate the PDF of breakthrough time T_{break}, defined as the time it takes the contaminant released at $\mathbf{x}_s$ to exceed the concentration 2 g/m^3 at any point along the control plane $x_1 = 100$ m (Figure 4.2). This task propagates uncertainty in the input, hydraulic conductivity $K(\mathbf{x})$, through the modeling process culminating in uncertainty in estimates of contaminant concentration $c(\mathbf{x}, t)$ and breakthrough time T_{break}. The resulting PDF allows one to estimate the probability of the contaminant concentration in groundwater exceeding its maximum allowable limit (2 g/m^3, in this example) over a certain time horizon $[0, T]$.

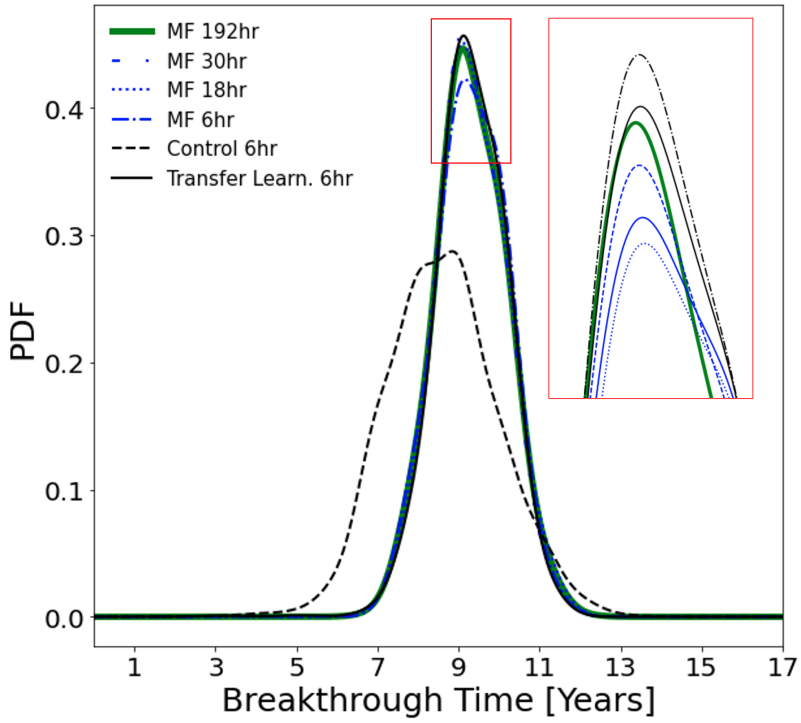

Figure 4.9: The converged CDF of breakthrough time T_{break} calculated using MC simulations, control CNN, and CNN trained via transfer learning, for various data budgets of the PDE-based simulations. The insert zooms in on the PDF peak.

In Figure 4.9, the PDF of T_{break} constructed from 192 hours of traditional MC simulations provides the ground truth. The randomly initiated CNN trained on six hours worth of data is unable to replicate accurately the breakthrough PDF. Likewise, 6 and 18 hours worth of MC simulations fail to provide an accurate estimate of this PDF. On the other hand, both 30 hours worth of traditional MC simulations and the CNN trained via transfer learning on six hours worse of data accurately reproduce the PDF of T_{break}. Hence, the MC simulations require five times more data-generation budget as our CNN.

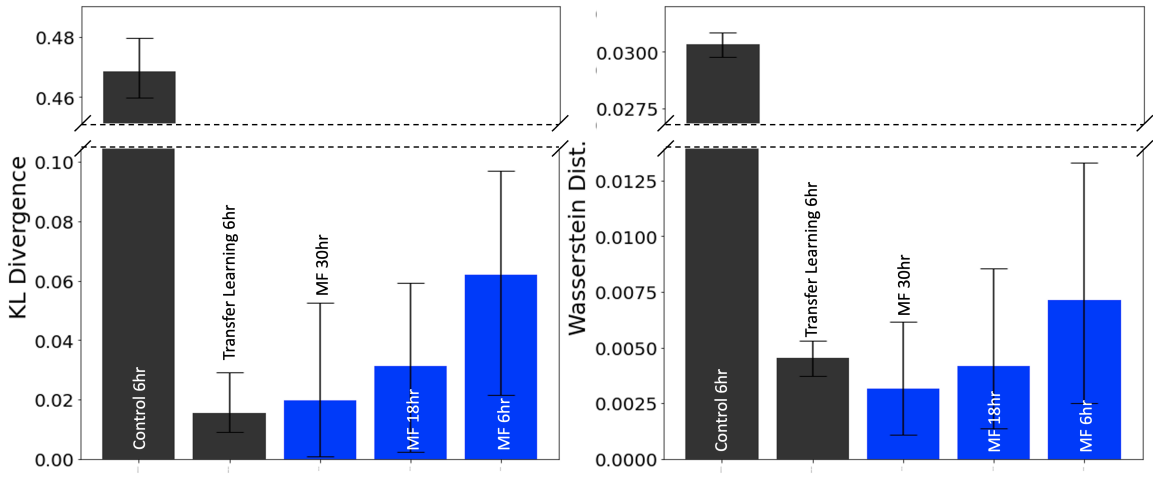

Figure 4.10: Estimation accuracy of the CDF of breakthrough time T_{break} calculated using MC simulations, control CNN, and CNN trained via transfer learning, for various data budgets of the PDE-based simulations in Figure 4.9. The accuracy is quantified in terms of the KL divergence (left panel) and the Wasserstein distance (right panel). The error bars represent a 68% confidence bound.

In additional to the visual comparison of alternative estimates of the PDF of T_{break}, Figure 4.10 exhibits their KL divergence and Wasserstein distance. These metrics make it clear that the control CNN trained on six hours worth of data and 6 and 18 hours of the traditional MC simulations fail to accurately estimate the T_{break} PDF. The CNN trained through transfer-learning on six hours worth of data has prediction accuracy of the 30 hours of MC simulations.

4.4.3 Comparison Between Transfer Learning Methods

Finally, we test i) the robustness of upcycling a pretrained CNN via transfer learning, and ii) the relative gains achieved by multifideling training and upcycling. The former strategy deploys the two-level training method from Chapter 2 to accommodate low-fidelity data gerated by solving the transport problem on the 64×64 mesh. The conducted experiments are summarized below:

Experiment 1: Training CNNs to solve the multiphase flow problem with permeability fields generated using a correlation length of $\lambda_Y = 19$ m starting from the CNN trained to solve the advection-dispersion problem.

Experiment 2: Training CNNs to solve the multiphase flow problem with permeability fields generated using a correlation length of $\lambda_Y = 8$ m starting from the CNN trained to solve the multiphase flow problem with a permeability field generated using a correlation length of $\lambda_Y = 19$ m.

Experiment 3: Training CNNs to solve the advection-dispersion problem using two levels of data. Rather than generating the coarse-resolution conductivity maps by upscaling their fine-resolution counterparts (Chapter 2), we use GSTools (Müller et al., 2022) to generate random realizations of coarse-resolution conductivity maps that have the same statistics as fine-resolution conductivity maps and the same random seeds. Representative fine and coarse log conductivity fields are displayed in Figure 4.11. We use the hyperparameters from the control CNN.

For the sake of clarity, we introduce the following plotting conventions:

- Black circles indicate results of the CNNs solving the multiphase flow problem for permeability fields whose correlation length is $\lambda_Y = 19$ m (the long problem).
- Green circles indicate results of the CNNs solving the multiphase problem for permeability fields whose correlation length is $\lambda_Y = 8$ m (the short problem).

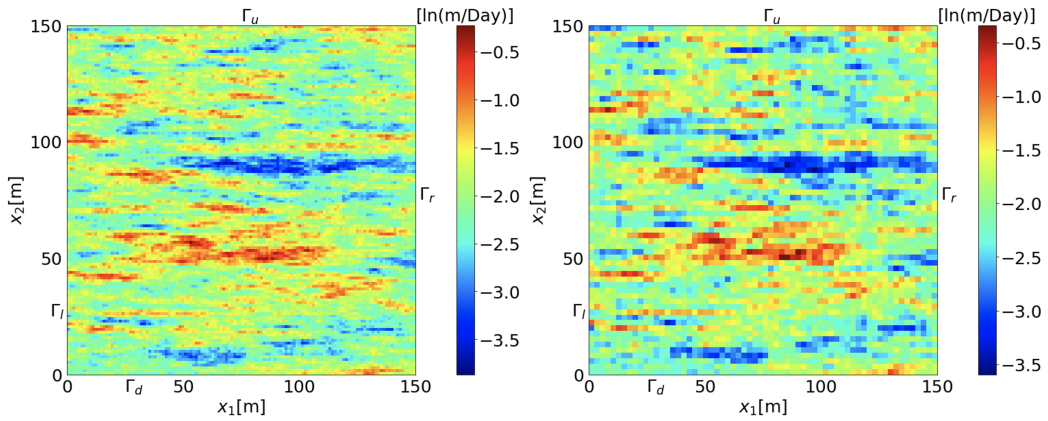

Figure 4.11: Representative realizations of log conductivity field $Y = \ln K$ on the fine (128×128) grid (left) and the coarse (64×64) grid (right). K is expressed in m/d.

- Blue circles indicate results of the CNNs solving the advection-dispersion problem.
- The CNNs starting from the default random initialization and trained only on 128×128 data are referred to as "Control".
- Crosses indicate results of the CNNs trained on multifidelity data, which are referred to as "Two-level".
- Diamonds indicate results of the upcycled CNNs, which are referred to as"Upcycled".

Across all data sets, the CNNs trained on multifidelity data and the upcycled CNNs outperform the control CNN at every data-generation budget. For the long multiphase flow problem, CNN trained on multifidelity data has the smallest RMSE among all the training strategies considered (the top left panel of Figure 4.12). For the short multiphase flow problem, the upcycled CNN outperforms the CNN trained on multifidelity data for budgets less than ten hours (the top right panel of Figure 4.12).

This figure shows that the upcycled CNN and the control CNN have similar prediction-accuracy limits, whereas training on multifidelity data may allow one to go beyond this limit. At six hours worth of data, the CNN trained on multifidelity data and the upcycled CNN trained on high-fidelity data achieve similar RMSEs. As the data generation budget increases, the multifidelity training

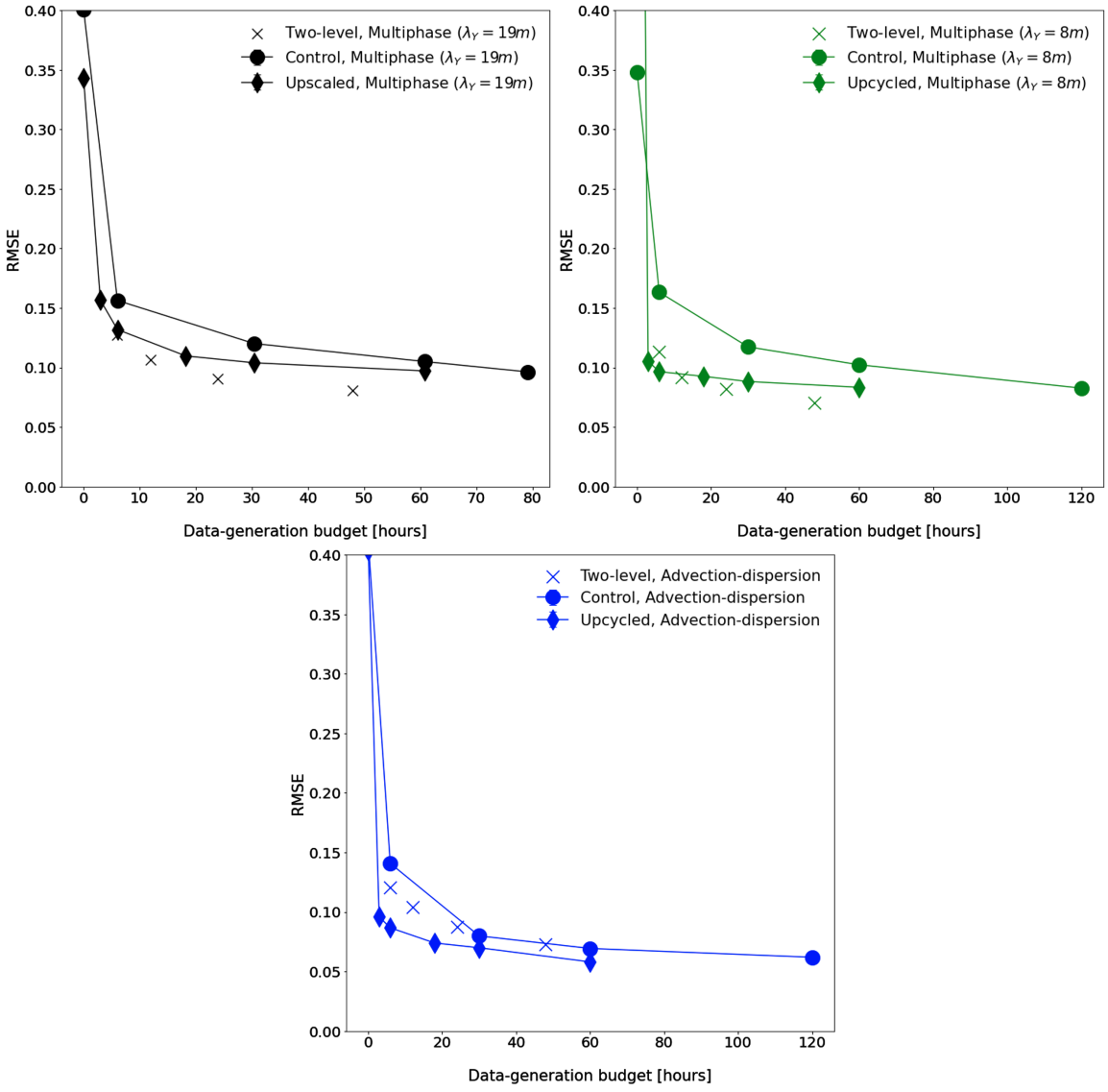

Figure 4.12: Prediction accuracy of the upcycled CNNs and the CNNs alternatively trained on the 128×128 data and the multifidelity data. The accuracy is reported in terms of the RMSE for the multiphase flow problem for the permeability field with correlation lengths $\lambda_Y = 19$ m (top left panel) and $\lambda_Y = 8$ m (top left panel), and for the advection-dispersion problem (bottom panel).

outperforms the upcycling technique, with the latter asymptoting to the RMSE of the control CNN because both are trained on the same amount and quality of data and the benefits of using the pretrained weights diminish with the data volume.

The multifidelity training also achieves a lower RMSE on the solute transport problem when the data budget exceeds ten hours (the bottom panel in Figure 4.12). The upcycled CNN has the lowest RMSE at every data-generation budget. The CNN trained on multifidelity data is less accurate than the upcycled CNNs, which we attribute to the random generation of the coarse (64×64) conductivity fields rather than the rigorous upscaling of the original fine (128×128) conductivity fields used in Chapter 2. This finding points to the importance of generating adequate low-fidelity data.

The top left panel of Figure 4.13 shows that the control CNNs for the two multiphase problems have similar RMSEs, while the control CNNs for the advection-dispersion problem has a lower RMSE. In all three problems, RMSEs of the control CNNs reach their respective asymptotic limit, i.e., fail to improve, as the data budget increases beyond a certain limit.

The top right panel of Figure 4.13 demonstrates the similar RMSE behavior of the CNNs trained on multifidelity data from the three problems. Visual comparison with Figure 4.12 suggests that the rate of the RMSE improvement with data-generation budget is greater for the CNNs trained on multifidelity data than for either the control CNNs or the upcycled CNNs.

The accuracy of the upcycled CNNs in the bottom panel of Figure 4.13 is associated with the quickest performance gains at less than six hours of data-generation budget. However, as the budget increases, their RMSEs achieve an asymptotic limit, indicating the limitation of the gains one might achieve with the upcycled strategy.

4.5 Conclusions

We proposed a transfer learning-based approach to retrain a CNN designed for one task to solve a new task. The pretrained network was originally built to solve a multiphase flow problem. The new data were generated by solving a PDE representing advection-dispersion transport in a heterogeneous

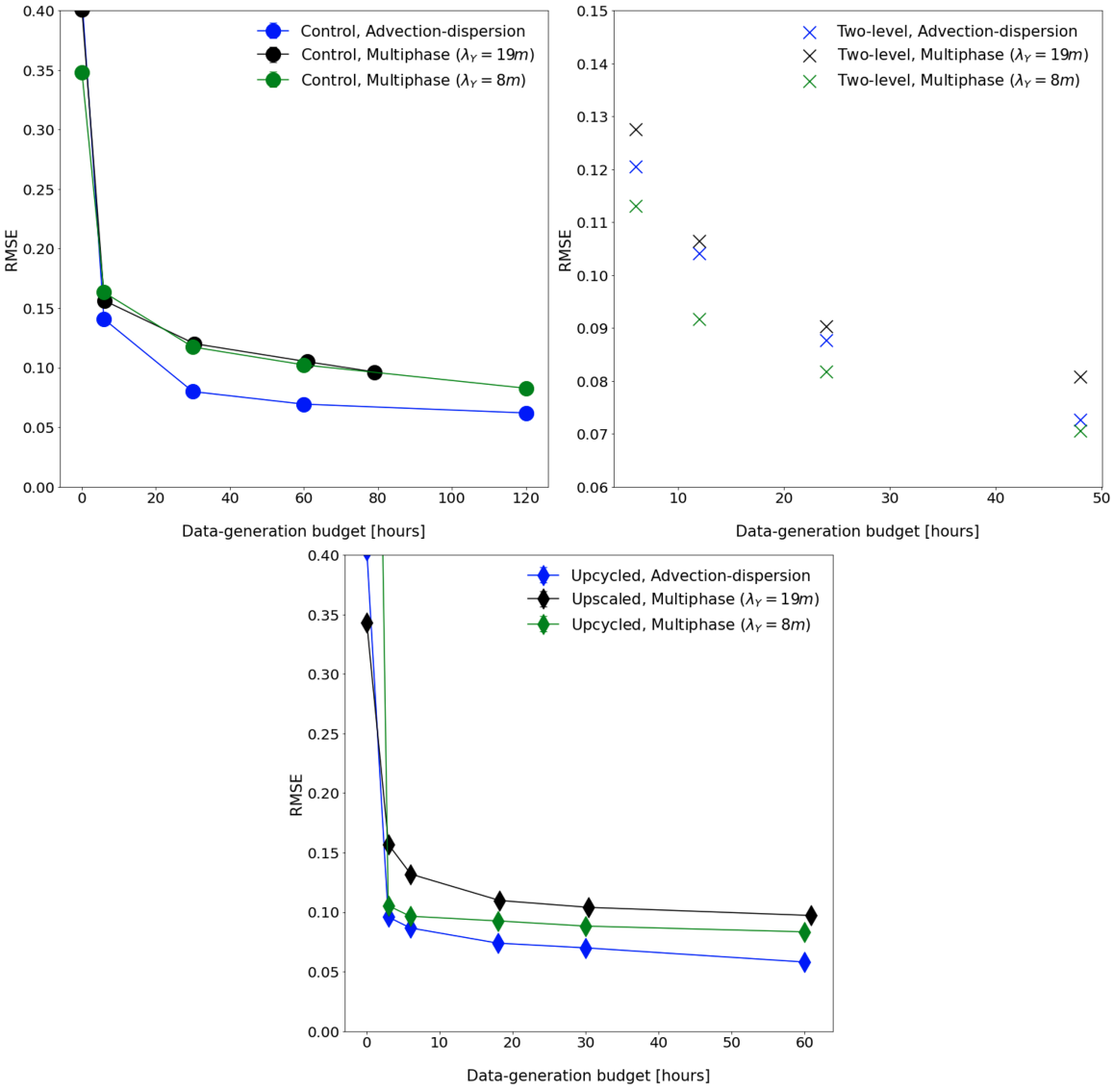

Figure 4.13: Prediction accuracy of the trained CNNs for the multiphase flow problems with permeability field whose correlation length is $\lambda_Y = 19$ m and $\lambda_Y = 8$ m, and for the advection-dispersion problem. The accuracy is reported in terms of the RMSE for the CNNs trained on multifidelity data (top left panel) and on high-fidelity data (top right panel), and for the upcycled CNNs.

porous medium with uncertain (random) hydraulic conductivity. A quantity of interest in this example is the PDF of the breakthrough time of a dissolved contaminant. Our analysis leads to the following major conclusions.

1. The retrained CNN provides an accurate surrogate of the PDE-based model of solute transport, even in the presence of sharp dynamic fronts. The CNN surrogate is three orders of magnitude faster than a numerical solution to the contaminant transport problem obtained via the industry-standard simulators.

2. The speed gains make CNN surrogates a valuable tool for ensemble-based computations of PDF of a quantity of interest.

3. Starting with a pretrained CNN reduces the data-generation budget five-fold relative to training of a randomly initialized CNN. The retrained CNN is more accurate than the CNN trained from random initializations, at every data budget. The greatest gain in performance is achieved as small budgets.

4. For every data-generation budget, the CNN retrained from a pretrained model exhibits a significantly smaller RMSE on test data than the CNNs trained using the random (`PyTorch` default) initializations.

5. The estimated PDF of the quantity of interest obtained from the upcycled CNN is close to the PDF of converged MC simulations; but the former are five orders of magnitude faster to obtain than the latter.

6. A CNN trained using a pretrained model as a starting point has the best predictive performance at data generation budgets of less than six hours. As the data budget increases, the prediction accuracy of upcycled CNNs approaches an asymptotic limit.

Chapter 5

Overall Conclusions and Future Work

In this dissertation we introduced two methods to reduce the computational costs associated with training NNs. In Chapters 2 and 3, we used transfer learning to train CNNs on multifidelity data generated by solving multiphase flow problems for different realizations of the input parameters (permeability fields). The CNN's performance was evaluated for accuracy and and was further tested on UQ tasks. In Chapter 4, we promulgated another use of transfer learning to further speed up NN training. Our strategy is to retrain a CNN originally built for one set of PDEs to act as a surrogate for another set of PDEs. The resulting surrogate was again evaluated for accuracy and was further tested on UQ tasks.

- The multifidelity data do not have to come from the coarseness/fineness of the grid. It can refer to any methods of generating data where there is a cost/accuracy trade off. Other examples of multifidelity data include a fast simulation of a process with some physics left out vs. a slow simulation that includes all relevant physics; or two-dimensional data vs. three-dimensional data. We hope that our transfer learning approaches will work on other types of multifidelity

data and allow one to inexpensively train other high-performance models.

- The output volumes of the CNN architecture are key to our implementation of transfer learning as the output volumes matched the dimensions of our multiscale data. However, the model was not originally designed with transfer learning and problems can arise. One such problem is that the model can get too big as each time we apply transfer learning, we are growing the model as well. Methods such as NN pruning (Blalock et al., 2020; Karnin, 1990) can keep model size more manageable.

- Using multifidelity data and transfer learning is effective in training accurate surrogates on a small data-generation budget. Transfer learning works best when the original task and the new task are similar. In this dissertation, the two tasks were chosen because they shared qualitative similarities. If a quantitative approach to calculating the similitude between two processes can be developed, the guesswork in selecting data set ratios for transfer learning may be optimized.

- The NN surrogates in this study successfully function on two-dimensional data. We did not work with three-dimensional data because the memory demands associated with the additional dimension tested GPU's vRAM limits. As algorithms get more efficient and training hardware become more powerful, effective training on three-dimensional data may be possible.

- In Chapter 3, we successfully trained a CNN on three levels of data. However, selecting the optimum ratio between the high-, medium-, and low-fidelity data remains a challenge as we only had the experience of training on two levels of data and the theoretical results of MLMC (Taverniers et al., 2020; Giles, 2008) to guide our data ratio selection. A more rigorous look into the selection of the training data will strengthen workflows that train models on multilevel data.

- In Chapters 2-4, the CNN surrogate functions as an image-to-image regression and is able to produce saturation and concentration maps that are evolving in time. To truly evolve a

solution in time, a recurrent neural network (RNN) training scheme may be better. RNNs are notorious for suffering from the vanishing gradient problem; a problem which is exacerbated on large inputs and outputs like the high-fidelity saturation maps used in this study. Using coarse data to train an RNN to account for the temporal evolution and using the relationship between coarse and fine data to downscale the RNN solution could result in a more general training scheme for time-dependent problems.

- Many studies of NN surrogates, including this one, deal with individual strategies in isolation. The intent in this study was to gauge the merit of each strategy by itself. However, it is plausible to think that the combination of training strategies may yield even better results than using individual strategies. A study investigating optimal ways to combine existing strategies could yield both more accurate models at a small data generation budget and high-accuracy CNN surrogates.

Appendix A

Model Size Investigation

NN architecture selection is important. A NN model that is too small, not enough layers or weights within each layer, will not be able to perform complex tasks. While a NN model, especially on a GPU, that is too big may be expensive to train; GPT-3, a well known language model developed by OpenAI, is famous for having an estimated cost of $4.6 million per training run (Dale, 2021). However there is no standard accepted method for determining the size of a NN model. For this work, the largest network that can reliably train on Nvidia V100 GPU with 16GB vRAM was selected to be the NN architecture of choice. The CNN was able to perform the tasks within this study, but there is little room for model expansion if a new application demands it. Following this line of thought, a smaller model is more valuable than a larger model if they perform the same task equally well.

The size of the CNN used in this study is determined by two variables: initial number of features and growth rate. The initial number of features determines how many convolutional filters are included in the first layer of the CNN. The growth rate determines how many layers are added from one dense block to the next. Two grid searches were performed to discern how predictive performance, test RMSE, is effected by varying initial number of features and growth rates. The first grid search was performed on the multiphase flow data set from the permeability fields generated

using the correlation length of $\lambda_Y = 19m$ and the results are displayed in Figure A.2. The second grid search was performed on the advection-dispersion data set. Both grid searches were only performed for training the CNN on HFS data, and the number of training data used in the searches was fixed at $N_train = 500$. The hyperparameters used in the grid search are the same hyperparameters used to generate the HFS data points for Chapter 2 and Chapter 4 respectively.

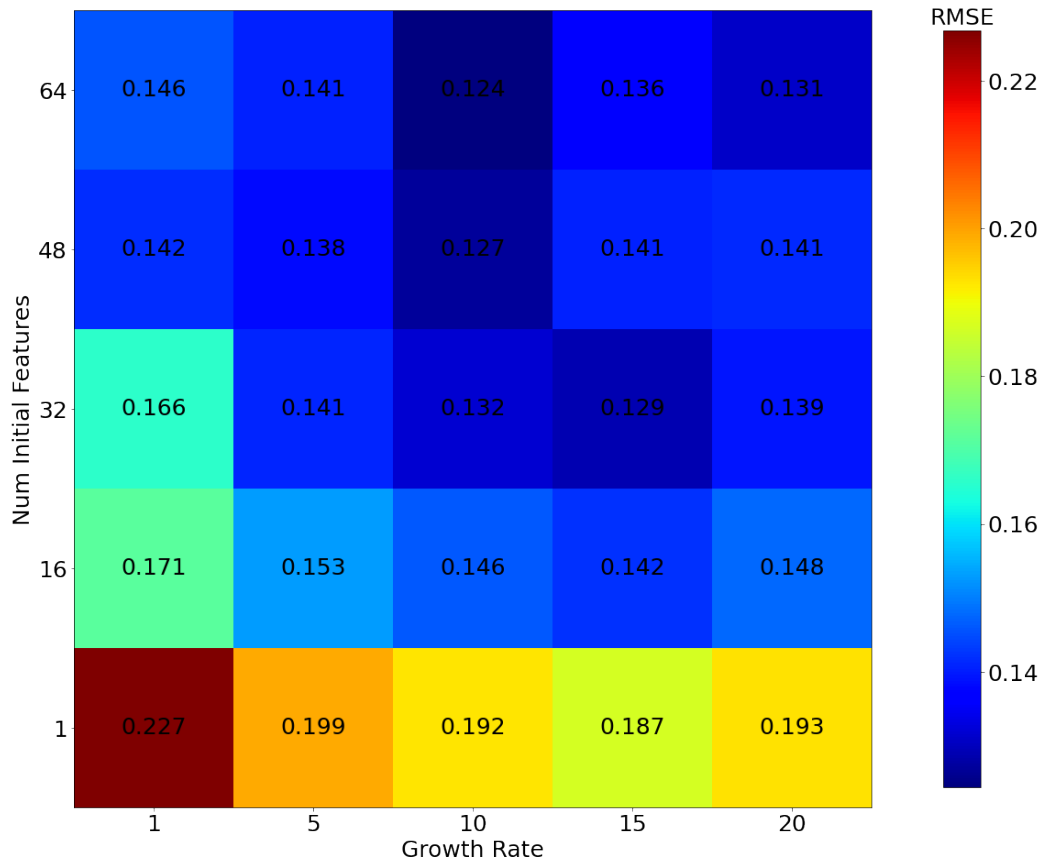

Figure A.1: The grid search reporting predictive performance, test RMSE, based on varying number of initial features and growth rate. This grid search was performed on the multiphase flow data set from the permeability fields generated using the correlation length of $\lambda_Y = 19m$.

In both grid searches, the test RMSE drops significantly when the number of initial feature increases from one to sixteen. A smaller drop in RMSE is observed when the number of initial feature increases from sixteen to thirty-two.

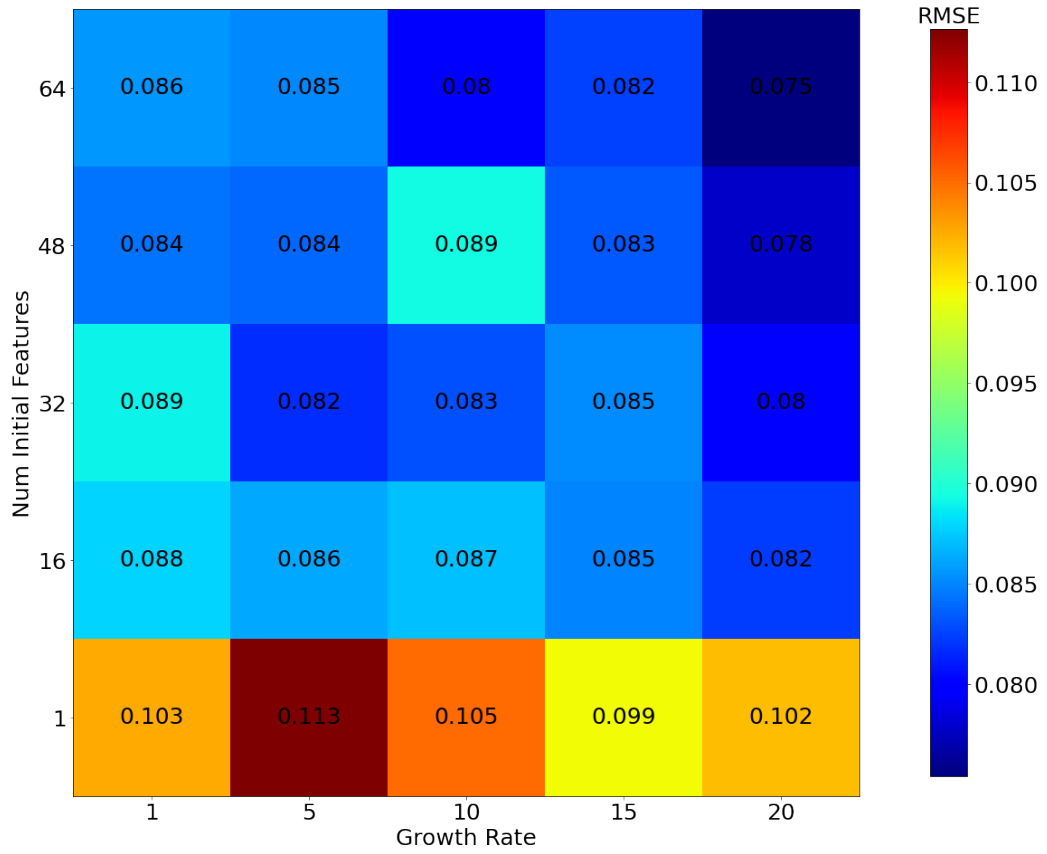

Figure A.2: The grid search reporting predictive performance, test RMSE, based on varying number of initial features and growth rate. This grid search was performed on the advection-dispersion data set.

In the multiphase flow problem, the test RMSE drops significantly by increasing the growth rate from one to five. And another drop in test RMSE is observed when increasing the growth rate from five to ten. The test RMSE drops were the greatest for CNNs with number of initial features equal or less than thirty-two.

Based on these grid searches, A CNN constructed from a number of initial features of forty-eight and a growth rate of ten would be a more optimized model; a smaller model that can solve the given problems. The CNN constructed from a number of initial features of forty-eight and a growth rate of ten would have 469033 trainable weights. The CNN used in this study had a number of initial features of sixty-four and a growth rate of twenty which resulted in 1648115 trainable weights; the optimized model would have 3.5 fold less trainable weights while performing at a similar predictive accuracy. Such a model can potentially be trained on lower cost GPU or be expanded to solve more complicated problems.

Appendix B

Supplemental Material for Chapter 2

B.1 Pseudocode

Algorithm 1: CNN training for given data

Input : Starting model (M_{in}); training data ($data_{\text{train}}$); test data ($data_{\text{test}}$); Number of phases(N_p); Epochs (eps)

Output : Best output model (M_{out})

Procedure:

for $i = 1, \ldots, N_p$ **do**

  for $j = 1, \ldots, eps$ **do**

    Train M_{in} using $data_{\text{train}}$;

    Compute $RMSE_{\text{test}}$ using $data_{\text{test}}$;

  end

  Set $RMSE_{\text{check}}$ as mean of last 10 $RMSE_{\text{test}}$;

  if $RMSE_{\text{check}} < RMSE_{\text{best}}$ **then**

    Set M_{out} as M_{in};

    Set $RMSE_{\text{best}}$ as $RMSE_{\text{check}}$;

  end

end

Algorithm 2: Phase 1: Training using LFS

Input : Original model (M_{orig}); Convolution Transpose 2 layer from M_{orig} ($L_{\text{conv,transpose2}}$); Temporary convolution layer in order to match LFS output dimensions (L_{temp}); LFS training data ($data_{\text{train,LFS}}$); LFS test data ($data_{\text{test,LFS}}$); Number of phase 1 iterations(N_{phase1}); Epoch in phase 1 (eps_{phase1})

Output : CNN trained on LFS ($M_{1,\text{ best}}$)

Procedure:

Set M_{mod1} by removing $L_{\text{conv,transpose2}}$ from M_{orig};

Set M_1 by attaching L_{temp} to the end of M_{mod1};

Train $M_{1,\text{best}}$ using Algorithm 1 (inputs: $M_{\text{in}} = M_1$, $N_{\text{phase}} = N_{\text{phase1}}$, eps = $\text{eps}_{\text{phase1}}$, $data_{\text{test}} = data_{\text{test,LFS}}$, $data_{\text{train}} = data_{\text{train,LFS}}$);

Algorithm 3: Phase 2: Initial training using HFS

Input : Model from Phase 1 ($M_{1,\text{best}}$); Convolution Transpose 2 layer from M_{orig} ($L_{\text{conv,transpose2}}$); Temporary convolution layer in order to match LFS output dimensions (L_{temp}); HFS training data ($data_{\text{train,HFS}}$); HFS test data ($data_{\text{test,HFS}}$); Number of phase 2 iterations(N_{phase2}); Epoch in phase 2 (eps_{phase2})

Output : CNN trained on LFS and HFS ($M_{2,\text{best}}$)

Procedure:

Set M_{mod2} by removing L_{temp} from $M_{1,\text{best}}$, and lock all weights;

Set M_2 by attaching $L_{\text{conv,transpose2}}$ to the end of M_{mod2} (the weights of $L_{\text{conv,transpose2}}$ remain unlocked);

Train $M_{2,\text{best}}$ using Algorithm 1 (inputs: $M_{start} = M_2$, $N_{phase} = N_{phase2}$, $eps = eps_{phase2}$, $data_{\text{test}} = data_{\text{test,HFS}}$, $data_{\text{train}} = data_{\text{train,HFS}}$);

Algorithm 4: Phase 3: Final training using HFS

Input : Model from Phase 2 ($M_{2,\text{best}}$); HFS training data ($data_{\text{train,HFS}}$); HFS test data ($data_{\text{test,HFS}}$); Number of phase 3 iterations (N_{phase3}); Epoch in phase 3 (eps_{phase3})

Output : Fine tuned CNN trained on LFS and HFS ($M_{3,\text{ best}}$)

Procedure:

Set M_3 by unlocking all weights in $M_{2,\text{best}}$;

Train $M_{3,best}$ using Algorithm 1 (inputs: $M_{start} = M_3$, $N_{phase} = N_{phase3}$, $eps = ep_{phase3}$, $data_{\text{test}} = data_{\text{test,HFS}}$, $data_{\text{train}} = data_{\text{train,HFS}}$);

Algorithm 5: Training Surrogate Model on Two Levels of Data

Input : HFS training data ($data_{\text{train,HFS}}$); HFS test data ($data_{\text{test,HFS}}$); LFS training data ($data_{\text{train,LFS}}$); LFS test data ($data_{\text{test,LFS}}$); Number of phase 1 iterations(N_{phase1}); Number of phase 2 iterations(N_{phase2}); Number of phase 3 iterations(N_{phase3}); Epoch in phase 1 (eps_{phase1}); Epoch in phase 2 (eps_{phase2}); Epoch in phase 3 (eps_{phase3})

Output : Surrogate model on the high-fidelity scale (M_3)

Procedure:

Set and initialize M_{orig} as original model as described by Table 2.2;

Set $L_{conv,transpose2}$ as "Convolution Transpose 2" layer from M_{orig};

Set L_{temp} as temporary convolution layer in order to match LFS output dimensions $(16 \times 64 \times 64)$;

Train M_1 using LFS data via Algorithm 2;

Train M_2 using HFS data via Algorithm 3;

Train M_3 using HFS data via Algorithm 4;

Appendix C

Supplemental Material for Chapter 3

C.1 Pseudocode

Algorithm 6: CNN training for given data

Input : Starting model (M_{in}); training data ($data_{\text{train}}$); test data ($data_{\text{test}}$); Number of phases(N_p); Epochs (eps)

Output : Best output model (M_{out})

Procedure:

for $i = 1, \ldots, N_p$ **do**

  for $j = 1, \ldots, eps$ **do**

    Train M_{in} using $data_{\text{train}}$;

    Compute $RMSE_{\text{test}}$ using $data_{\text{test}}$;

  end

  Set $RMSE_{\text{check}}$ as mean of last 10 $RMSE_{\text{test}}$;

  if $RMSE_{\text{check}} < RMSE_{\text{best}}$ **then**

    Set M_{out} as M_{in};

    Set $RMSE_{\text{best}}$ as $RMSE_{\text{check}}$;

  end

end

Algorithm 7: Phase 1: Training using VLFS and LFS

Input : Modified model in VLFS scale (M_1); Convolution Transpose 1 layer from M_{orig} ($L_{\text{conv,transpose1}}$); Temporary convolution layer in order to match VLFS output dimensions (L_{temp2}); VLFS training data ($data_{\text{train,VLFS}}$); VLFS test data ($data_{\text{test,VLFS}}$); LFS training data ($data_{\text{train,LFS}}$); LFS test data ($data_{\text{test,LFS}}$); Number of phase 1 iterations(N_{phase1}); Epoch in phase 1 (eps_{phase1})

Output : CNN trained on VLFS and LFS (M_2)

Procedure:

Train M_1^* using Algorithm 6 (inputs: $M_{\text{in}} = M_1$, $N_{\text{phase}} = N_{\text{phase1}}$, $\text{eps} = \text{eps}_{\text{phase1}}$, $data_{\text{test}} = data_{\text{test,VLFS}}$, $data_{\text{train}} = data_{\text{train,VLFS}}$);

Set $M_{2,\text{init},part1}$ by Lock all weights of M_1^* and attaching $L_{\text{conv,transpose1}}$ to the end of M_1^* (the weights of $L_{\text{conv,transpose1}}$ remain unlocked);

Set $M_{2,\text{init},part2}$ by attaching L_{temp2} to the end of $M_{2,\text{init},part1}$ (the weights of L_{temp2} remain unlocked);

Train $M_{2,\text{temp}}$ using Algorithm 6 (inputs: $M_{\text{in}} = M_{2,\text{init},part2}$, $N_{\text{phase}} = N_{\text{phase1}}$, $\text{eps} = \text{eps}_{\text{phase1}}$, $data_{\text{test}} = data_{\text{test,LFS}}$, $data_{\text{train}} = data_{\text{train,LFS}}$);

Set M_2 by unlocking all weights of $M_{2,\text{temp}}$

Algorithm 8: Phase 2: Training using LFS and HFS

Input : Modified model in LFS scale (M_2); Dense Block (Decoding) layer from M_{orig} ($L_{dense,decoding}$); Convolution Transpose 2 layer from M_{orig} ($L_{\text{conv,transpose2}}$); LFS training data ($data_{\text{train,VLFS}}$); LFS test data ($data_{\text{test,LFS}}$); HFS training data ($data_{\text{train,HFS}}$); HFS test data ($data_{\text{test,HFS}}$); Number of phase 2 iterations(N_{phase2}); Epoch in phase 2 (eps_{phase2})

Output : CNN trained on VLFS and HFS (M_3)

Procedure:

Train M_2^* using Algorithm 6 (inputs: $M_{\text{in}} = M_2$, $N_{\text{phase}} = N_{\text{phase2}}$, $\text{eps} = \text{eps}_{\text{phase2}}$, $data_{\text{test}} = data_{\text{test,LFS}}$, $data_{\text{train}} = data_{\text{train,LFS}}$);

Set $M_{3,\text{init},part1}$ by Lock all weights of M_2^* and attaching $L_{\text{dense,decoding}}$ to the end of M_2^* (the weights of $L_{\text{dense, decoding}}$ remain unlocked);

Set $M_{3,\text{init},part2}$ by attaching $L_{\text{conv,transpose2}}$ to the end of $M_{3,\text{init},part1}$ (the weights of $L_{\text{conv,transpose2}}$ remain unlocked);

Train $M_{3,\text{temp}}$ using Algorithm 6 (inputs: $M_{\text{in}} = M_{3,\text{init},part2}$, $N_{\text{phase}} = N_{\text{phase2}}$, $\text{eps} = \text{eps}_{\text{phase1}}$, $data_{\text{test}} = data_{\text{test,HFS}}$, $data_{\text{train}} = data_{\text{train,HFS}}$);

Set M_3 by unlocking all weights of $M_{3,\text{temp}}$

Algorithm 9: Phase 3: Training using HFS

Input : Modified model in HFS scale (M_3); HFS training data ($data_{\text{train,HFS}}$); HFS test data ($data_{\text{test,HFS}}$); Number of phase 3 iterations(N_{phase3}); Epoch in phase 3 (eps_{phase3})

Output : CNN trained on HFS (M_3^*)

Procedure:

Train M_3^* using Algorithm 6 (inputs: $M_{\text{in}} = M_3$, $N_{\text{phase}} = N_{\text{phase3}}$, $\text{eps} = \text{eps}_{\text{phase3}}$, $data_{\text{test}} = data_{\text{test,HFS}}$, $data_{\text{train}} = data_{\text{train,HFS}}$);

Algorithm 10: Training Surrogate Model on Three Levels of Data

Input : HFS training data ($data_{\text{train,HFS}}$); HFS test data ($data_{\text{test,HFS}}$); VLFS training data ($data_{\text{train,VLFS}}$); VLFS test data ($data_{\text{test,VLFS}}$); Number of phase 1 iterations(N_{phase1}); Number of phase 2 iterations(N_{phase2}); Number of phase 3 iterations(N_{phase3}); Epoch in phase 1 (eps_{phase1}); Epoch in phase 2 (eps_{phase2}); Epoch in phase 3 (eps_{phase3})

Output : Surrogate model on the high-fidelity scale (M_3)

Procedure:

Set and initialize M_{orig} as original model as described by Table 4.2;

Set $L_{conv,transpose1}$ as "Convolution Transpose 1" layer from M_{orig};

Set $L_{dense,decoding}$ as "Dense Block (decoding)" layer from M_{orig};

Set $L_{conv,transpose2}$ as "Convolution Transpose 2" layer from M_{orig};

Set L_{temp1} as temporary convolution layer in order to match VLFS output dimensions $(16 \times 32 \times 32)$;

Set L_{temp2} as temporary convolution layer in order to match LFS output dimensions $(16 \times 64 \times 64)$;

Build M_1 by removing $L_{conv,transpose1}$, $L_{dense,decoding}$, $L_{conv,transpose2}$ from M_{orig} Add L_{temp1} to M_1 Train M_1 using VLFS data via Algorithm 7;

Train M_2 using LFS data via Algorithm 8;

Train M_3 using HFS data via Algorithm 9;

Appendix D

Supplemental Material for

Chapter 4

D.1 Pseudocode

Algorithm 11: CNN training for given data

Input : Starting model (M_{in}); training data ($data_{\text{train}}$); test data ($data_{\text{test}}$); Number of phases(N_p); Epochs (eps)
Output : Best output model (M_{out})
Procedure:
for $i = 1, \ldots, N_p$ **do**
  for $j = 1, \ldots, eps$ **do**
    Train M_{in} using $data_{\text{train}}$;
    Compute $RMSE_{\text{test}}$ using $data_{\text{test}}$;
  end
  Set $RMSE_{\text{check}}$ as mean of last 10 $RMSE_{\text{test}}$;
  if $RMSE_{\text{check}} < RMSE_{\text{best}}$ **then**
    Set M_{out} as M_{in};
    Set $RMSE_{\text{best}}$ as $RMSE_{\text{check}}$;
  end
end

Algorithm 12: Phase 1: Training Last Blocks Initially

Input : Pretrained Model (M_0); Dense Block (Decoding) layer from M_0 ($L_{\text{dense,decode}}$); Convolution Transpose 2 layer from M_0 ($L_{\text{conv,transpose2}}$); Training data ($data_{\text{train}}$); Test data ($data_{\text{test}}$); Number of phase 1 iterations(N_{phase1}); Epoch in phase 1 (eps_{phase1})

Output : Intermediate CNN (M_1)

Procedure:

Lock all weights of M_0;

Set M_1 by unlocking the weights of $L_{\text{dense,decode}}$ and $L_{\text{conv,transpose2}}$;

Train M_1 using Algorithm 11 (inputs: $M_{start} = M_1$, $N_{phase} = N_{phase1}$, $eps = eps_{phase1}$, $data_{\text{test}} = data_{\text{test}}$, $data_{\text{train}} = data_{\text{train}}$);

Algorithm 13: Phase 2: Training Entire Model

Input : Model from Phase 1 (M_1); HFS training data ($data_{\text{train}}$); HFS test data ($data_{\text{test}}$); Number of phase 2 iterations (N_{phase2}); Epoch in phase 2 (eps_{phase2})

Output : Trained CNN (M_2)

Procedure:

Set M_2 by unlocking all weights in M_1;

Train M_2 using Algorithm 11 (inputs: $M_{start} = M_2$, $N_{phase} = N_{phase2}$, $eps = ep_{phase2}$, $data_{\text{test}} = data_{\text{test}}$, $data_{\text{train}} = data_{\text{train}}$);

Algorithm 14: Training Surrogate Model on Multiple Scales of Data

Input : Training data ($data_{\text{train}}$); Test data ($data_{\text{test}}$); Number of phase 1 iterations(N_{phase1}); Number of phase 2 iterations(N_{phase2}); Epoch in phase 1 (eps_{phase1}); Epoch in phase 2 (eps_{phase2})

Output : Trained CNN (M_2)

Procedure:

Import pretrained model M_0;

Set $L_{conv,transpose2}$ as "Convolution Transpose 2" layer from M_0;

Set $L_{dense,decode}$ as "Dense Block (Decoding)" layer from M_0;

Train M_1 using New HFS data via Algorithm 12;

Train M_2 using New HFS data via Algorithm 13;

Bibliography

Appleyard, J. R., Cheshire, I. M., and Pollard, R. K. (1981). Special techniques for fully implicit simulators. In *Proceedings of the European Symposium on Enhanced Oil Recovery*, pages 395–408, Bournemouth, UK.

Aziz, K. (1979). *Petroleum reservoir simulation*, volume 476. Applied Science Publishers, New York.

Bakker, M., Post, V., Langevin, C. D., Hughes, J. D., White, J. T., Starn, J., and Fienen, M. N. (2016). Scripting modflow model development using python and flopy. *Groundwater*, 54(5):733–739.

Bedekar, V., Morway, E. D., Langevin, C. D., and Tonkin, M. J. (2016). Mt3d-usgs version 1: A us geological survey release of mt3dms updated with new and expanded transport capabilities for use with modflow. Technical report, US Geological Survey.

Blalock, D., Gonzalez Ortiz, J. J., Frankle, J., and Guttag, J. (2020). What is the state of neural network pruning? *Proceedings of machine learning and systems*, 2:129–146.

Boso, F. and Tartakovsky, D. M. (2018). Information-theoretic approach to bidirectional scaling. *Water Resour. Res.*, 54(7):4916–4928.

Breiman, L. (2001). Random forests. *Mach. Learn.*, 45(1):5–32.

Cai, S., Mao, Z., Wang, Z., Yin, M., and Karniadakis, G. E. (2022). Physics-informed neural networks (pinns) for fluid mechanics: A review. *Acta Mechanica Sinica*, pages 1–12.

Corey, A. T. (1954). The interrelation between gas and oil relative permeabilities. *Producers Month.*, 19(1):38–41.

Couckuyt, I., Dhaene, T., and Demeester, P. (2014). SooDACE toolbox: A flexible object-oriented kriging implementation. *J. Mach. Learn. Res.*, 15:3183–3186.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

Dale, R. (2021). GPT-3: What's it good for? *Natural Language Engineering*, 27(1):113–118.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. (2014). Decaf: A deep convolutional activation feature for generic visual recognition. In *International Conference on Machine Learning*, pages 647–655. PMLR.

Durlofsky, L. J. (2005). Upscaling and gridding of fine scale geological models for flow simulation. In *8th International Forum on Reservoir Simulation*, volume 2024, pages 1–59, Iles Borromees, Stresa, Italy.

Friedman, J., Hastie, T., and Tibshirani, R. (2001). *The elements of statistical learning*, volume 1. Springer, New York.

Fuks, O. and Tchelepi, H. (2020). Limitations of physics informed machine learning for nonlinear two-phase transport in porous media. *J. Mach. Learn. Model. Comput.*, 1(1):19–37.

Geneva, N. and Zabaras, N. (2020). Multi-fidelity generative deep learning turbulent flows. *Found. Data Sci.*, 2(4):391–428.

Giles, M. B. (2008). Multilevel Monte Carlo path simulation. *Oper. Res.*, 56(3):607–617.

Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S., and Shet, V. (2013). Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*.

Haghighat, E., Raissi, M., Moure, A., Gomez, H., and Juanes, R. (2021). A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics. *Comput. Meth. Appl. Mech. Engrg.*, 379:113741.

Harbaugh, A. W. (2005). *MODFLOW-2005, the US Geological Survey modular ground-water model: the ground-water flow process*, volume 6. US Department of the Interior, US Geological Survey Reston, VA, USA.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Heinecke, A., Ho, J., and Hwang, W.-L. (2020). Refinement and universal approximation via sparsely connected relu convolution nets. *IEEE Signal Processing Letters*, 27:1175–1179.

Heinrich, S. (1998). Monte Carlo complexity of global solution of integral equations. *J. Complexity*, 14(2):151–175.

Heinrich, S. (2001). Multilevel Monte Carlo methods. In *International Conference on Large-Scale Scientific Computing*, pages 58–67. Springer.

Heße, F., Prykhodko, V., Schlüter, S., and Attinger, S. (2014). Generating random fields with a truncated power-law variogram: A comparison of several numerical methods. *Environmental Modelling & Software*, 55:32–48.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708.

Hwang, J. T. and Martins, J. R. R. A. (2018). A fast-prediction surrogate model for large datasets. *Aerospace Sci. Tech.*, 75:74–87.

Jiang, H. and Learned-Miller, E. (2017). Face detection with the faster R-CNN. In *2017 12th IEEE*

International Conference on Automatic Face & Gesture Recognition (FG 2017), pages 650–657. IEEE.

Jin, X., Cai, S., Li, H., and Karniadakis, G. E. (2021). Nsfnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations. *Journal of Computational Physics*, 426:109951.

Karnin, E. D. (1990). A simple procedure for pruning back-propagation trained neural networks. *IEEE transactions on neural networks*, 1(2):239–242.

Karpathy, A. and Fei-Fei, L. (2015). Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv:1412.6980*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25.

Lagaris, I. E., Likas, A., and Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans. Neural Networks*, 9(5):987–1000.

Lee, H. and Kang, I. S. (1990). Neural algorithm for solving differential equations. *J. Comput. Phys.*, 91(1):110–131.

Lin, G. and Tartakovsky, A. M. (2009). An efficient, high-order probabilistic collocation method on sparse grids for three-dimensional flow and solute transport in randomly heterogeneous porous media. *Advances in Water Resources*, 32(5):712–722.

Loshchilov, I. and Hutter, F. (2017). Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Mao, Z., Jagtap, A. D., and Karniadakis, G. E. (2020). Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789.

Meng, X. and Karniadakis, G. E. (2020). A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems. *J. Comput. Phys.*, 401:109020.

Mo, S., Zabaras, N., Shi, X., and Wu, J. (2019a). Deep autoregressive neural networks for high-dimensional inverse problems in groundwater contaminant source identification. *Water Resour. Res.*, 55(5):3856–3881.

Mo, S., Zhu, Y., Zabaras, N., Shi, X., and Wu, J. (2019b). Deep convolutional encoder-decoder networks for uncertainty quantification of dynamic multiphase flow in heterogeneous media. *Water Resour. Res.*, 55(1):703–728.

Montgomery, D. C. and Evans, D. M. (2018). Second-order response surface designs in computer simulation. *Aerospace Sci. Tech.*, 75:74–87.

Müller, F., Jenny, P., and Meyer, D. W. (2013). Multilevel Monte Carlo for two phase flow and Buckley-Leverett transport in random heterogeneous porous media. *J. Comput. Phys.*, 250:685–702.

Müller, S., Schüler, L., Zech, A., and Heße, F. (2022). Gstools v1. 3: a toolbox for geostatistical modelling in python. *Geoscientific Model Development*, 15(7):3161–3182.

Paleologos, E. K., Neuman, S., and Tartakovsky, D. M. (1996). Effective hydraulic conductivity of bounded, strongly heterogeneous porous media. *Water Resour. Res.*, 32(5):1333–1341.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A.,

dAlché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Peherstorfer, B. (2019). Multifidelity Monte Carlo estimation with adaptive low-fidelity models. *SIAM/ASA J. Uncert. Quant.*, 7(2):579–603.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2017). Machine learning of linear differential equations using gaussian processes. *Journal of Computational Physics*, 348:683–693.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2018). Numerical gaussian processes for time-dependent and nonlinear partial differential equations. *SIAM Journal on Scientific Computing*, 40(1):A172–A198.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2019a). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.*, 378:686–707.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2021). Physics informed learning machine. US Patent 10,963,540.

Raissi, M., Wang, Z., Triantafyllou, M. S., and Karniadakis, G. E. (2019b). Deep learning of vortex-induced vibrations. *Journal of Fluid Mechanics*, 861:119–137.

Raissi, M., Yazdani, A., and Karniadakis, G. E. (2020). Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Sinsbeck, M. and Tartakovsky, D. M. (2015). Impact of data assimilation on cost-accuracy tradeoff in multifidelity models. *SIAM/ASA J. Uncert. Quant.*, 3(1):954–968.

Song, D. H. and Tartakovsky, D. M. (2021). Transfer learning on multi-fidelity data. *Journal of Machine Learning for Modeling and Computing*, 2.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

Tang, M., Liu, Y., and Durlofsky, L. J. (2020). A deep-learning-based surrogate model for data assimilation in dynamic subsurface flow problems. *J. Comput. Phys.*, page 109456.

Tartakovsky, D. M. and Neuman, S. P. (1998). Transient effective hydraulic conductivities under slowly and rapidly varying mean gradients in bounded three-dimensional random media. *Water Resour. Res.*, 34(1):21–32.

Taverniers, S., Bosma, S. B., and Tartakovsky, D. M. (2020). Accelerated multilevel Monte Carlo with kernel-based smoothing and latinized stratification. *Water Resour. Res.*, 56(9):e2019WR026984.

Tripathy, R. K. and Bilionis, I. (2018). Deep uq: Learning deep neural network surrogate models for high dimensional uncertainty quantification. *J. Comput. Phys.*, 375:565–588.

Xiu, D. (2010). *Numerical Methods for Stochastic Computations: A spectral Method Approach*. Princeton University Press, Princeton, NJ.

Yang, L., Wang, P., and Tartakovsky, D. M. (2020). Resource-constrained model selection for uncertainty propagation and data assimilation. *SIAM/ASA J. Uncert. Quant.*, 8(3):1118–1138.

Ye, M., Neuman, S. P., Guadagnini, A., and Tartakovsky, D. M. (2004). Nonlocal and localized analyses of conditional mean transient flow in bounded, randomly heterogeneous porous media. *Water Resour. Res.*, 40:W05104.

Zhou, D.-X. (2020). Universality of deep convolutional neural networks. *Applied and computational harmonic analysis*, 48(2):787–794.

Zhou, Z. and Tartakovsky, D. M. (2021). Markov chain Monte Carlo with neural network surrogates: Application to contaminant source identification. *Stoch. Environ. Res. Risk Assess.*, 35(3):639–651.

Zhu, Y. and Zabaras, N. (2018). Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415–447.

Zhu, Y., Zabaras, N., Koutsourelakis, P.-S., and Perdikaris, P. (2019). Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *J. Comput. Phys.*, 394:56–81.