An innovative application of deep learning in multiscale modeling of subsurface fluid flow: Reconstructing the basis functions of the mixed GMsFEM

Abouzar Choubineh [Corresponding Author]
Department of Computer Science, University of Liverpool, Liverpool, United Kingdom
Department of Applied Mathematics, Xi'an Jiaotong-Liverpool University, Suzhou, P. R. China

Jie Chen
Department of Applied Mathematics, Xi'an Jiaotong-Liverpool University, Suzhou, P. R. China

Frans Coenen
Department of Computer Science, University of Liverpool, Liverpool, United Kingdom

Fei Ma
Department of Applied Mathematics, Xi'an Jiaotong-Liverpool University, Suzhou, P. R. China

Abstract
In multiscale modeling of subsurface fluid flow in heterogeneous porous media, standard polynomial basis functions are replaced by multiscale basis functions. For instance, to produce such functions in the mixed Generalized Multiscale Finite Element Method (mixed GMsFEM), a number of Partial Differential Equations (PDEs) must be solved, which requires a considerable overhead. Thus, it makes sense to replace PDE solvers with data-driven methods, given their great capabilities and general acceptance in the recent decades. Convolutional Neural Networks (CNNs) automatically perform feature engineering, and they also need fewer parameters via defining two-dimensional convolutional filters without reducing the quality of models. This is why four distinct CNN models were developed to predict four different multiscale basis functions for the mixed GMsFEM in the present study. These models were applied to 249,375 samples, with the permeability field as the only input. The statistical results indicate that the AMSGrad optimization algorithm with a coefficient of determination (R^2) of 0.8434 - 0.9165 and Mean Squared Error (MSE) of 0.0078 - 0.0206 performs slightly better than Adam with an R^2 of 0.8328 - 0.9049 and MSE of 0.0109 - 0.0261. Graphically, all models precisely follow the observed trend in each coarse block. This work could contribute to the distribution of pressure and velocity in the development of oil/gas fields. Looking at this work as an image (matrix)-to-image (matrix) regression problem, the constructed data-driven-based models may have applications beyond reservoir engineering, such as hydrogeology and rock mechanics.

Keywords: Subsurface fluid flow; Porous medium; Finite element method; GMsFEM; Machine learning; Convolutional neural network

1. Introduction

Modeling subsurface fluid flow in heterogeneous porous media has always been challenging, especially with real-world applications, such as the development of oil/gas fields and groundwater resources management. A heterogeneous porous medium indicates that it is not homogenous, and thus formation-related properties can have multiple scales. For example, in petroleum reservoirs, there may be numerous fractures (connected or disconnected) with different lengths, whose width is much smaller than the domain size. The permeability is defined as the ability of a rock to permit fluids to pass through it. The permeability of fractures can be lower than that of the matrix in dissolved carbonate reservoirs. There are some examples in the pre-salt carbonate reservoirs in Brazil. However, the permeability in fractures is generally much higher than that of the matrix. This is why the effect of fractures must be considered when modeling flow and transport processes. In the mesh generation stage of numerical modeling, adequately fine grids are used to resolve small scale fractures. By doing so, the discrete formulation of such problems produces a large system of equations, and consequently, the number of unknown parameters increases. The computation of the solution, therefore, becomes expensive. Model reduction techniques, upscaling (homogenization) (Ganjeh-Ghazvini, 2019; Liu et al., 2021) and multiscale (Chen and Hou, 2003; Ganis et al., 2017; Hajibeygi et al., 2008; He et al., 2021; Jenny et al., 2005; Peszynska, 2005) are necessary to decrease the degrees of freedom, and solve subsurface flow problems on coarse grids. These methods can considerably decrease resolution times without a significant loss of precision.

Upscaling is an averaging process in which the static (e.g., porosity) and dynamic (e.g., fluid saturation) properties of a fine grid model are scaled up to equivalent characteristics defined at a coarse grid level. This procedure should be performed in a way that the two models act as closely as possible to each other. For example, kernel bandwidth and wavelet transformation techniques were used to simultaneously scale up the porosity and permeability of a synthetic reservoir model in (Azad et al., 2021). Under the same circumstances, the simulation runs demonstrated that the upscaling error of the bandwidth method was much smaller than that of the wavelet method.

Multiscale techniques, such as multiscale finite element methods (Chen and Hou, 2003; He et al., 2021), multiscale finite volume methods (Hajibeygi et al., 2008; Jenny et al., 2005) and mortar multiscale methods (Ganis et al., 2017; Peszynska, 2005) solve flow problems on coarse grids through pre-calculated multiscale basis functions. These are developed locally on fine grids to capture the local multiscale information of a medium. Researchers have always been interested in enhancing the accuracy of multiscale solutions. For instance, a framework of Generalized Multiscale Finite Element Method (GMsFEM) was proposed by (Efendiev et al., 2013). This model generalizes the multiscale finite element method (Hou and Wu, 1997) by including further basis functions that can provide more local multiscale details to enrich the multiscale space.

Local mass conservation is of great importance for subsurface flow problems. The mixed multiscale finite element method is regarded as one of the most commonly used mass conservative multiscale techniques. In this technique, the multiscale process is used on two coarse elements with a common edge for the velocity, and piecewise constant basis functions are employed on a single coarse block for the pressure.

Following the basic computational procedure of GMsFEM, a framework for the mixed GMsFEM was developed by (Chen et al., 2020) to solve Darcy's flow (a linear relationship between pressure gradient and velocity) of a single-phase fluid in two-dimensional fractured porous media. Unlike previous studies, this method calculates the pressure in the multiscale space; several multiscale basis functions are obtained in a single coarse grid element, and the velocity is directly approximated in the fine grid space.

The following flow problem in the mixed formulation is considered:

$$\begin{gathered} k^{-1}u + \nabla p = 0 \quad in \quad \Omega \\ \nabla . u = f \quad in \quad \Omega \end{gathered} \tag{1}$$

with nonhomogeneous boundary condition:

$$u.n = g \quad on \quad \partial\Omega \tag{2}$$

where k is permeability, u is the Darcy velocity, p is the pressure, f is the source term, g is the given normal component of the Darcy velocity on the boundary, Ω is the computational domain and n is the outward unit norm vector on the boundary.

To illustrate the general solution framework of the mixed GMsFEM, τ^H is considered a confirming partition of Ω into finite elements with a coarse block size H, and τ^h is the fine grid partition with mesh size h. By defining $V = H(div, \Omega)$ and $W = L^2(\Omega)$, the mixed finite element spaces will be:

$$V_h = \{v_h \in V : v_h(t) = (b_t x_1 + a_t, d_t x_2 + c_t), a_t, b_t, c_t, d_t \in \mathbb{R}, t \in \tau^h\}$$

$$W_h = \{w_h \in W : w_h \; is \; a \; constant \; on \; each \; element \; in \; \tau^h\}$$

Supposing $\{\Psi_j\}$ is the set of multiscale base functions for the coarse element, the multiscale space for the pressure p is defined as the linear span of all local basis functions, which can be denoted as:

$$W_H = \oplus \{\Psi_j\} \quad in \quad \tau^H$$

The mixed GMsFEM is directed at finding $(u_H, p_H) \in (V_h, W_H)$ such that:

$$\begin{gathered} \int k^{-1} u_H . v_H - \int div(v_H) p_H = 0 \quad \forall \, v_H \in V_h^0 \\ \int div(u_H) w_H = \int f w_H \quad \forall \, w_H \in W_H \end{gathered} \tag{3}$$

where $u_H . n = g_H$ on $\partial\Omega$ for each coarse edge on the boundary and g_H is the average of function g on the corresponding coarse edge.

The following approach to construct the multiscale space W_H is adopted for approximating the pressure p in a systematic manner. A snapshot space is defined by solving a series of local cell problems on every coarse grid element with Dirichlet's boundary condition. In Dirichlet's condition, a value is first assigned to the pressure. Then the snapshot space is further decreased to find the dominant modes, where a local eigenvalue problem (one for each coarse element) needs to be solved. The linear span of these modes is termed the offline space.

The local cell problem defined in the coarse grid is the same as the original problem except that the source function is omitted. A delta function is assigned on the boundary of the coarse grid as a boundary condition. The delta function is defined as follows: a value 1 given a fine grid edge and a value of 0 for all other grid edges. As a consequence, the number of local cell problems that need to be solved equals the number of fine grid edges contained in the coarse grid boundary (here 12). The number of Partial Differential Equations (PDEs) that need to be solved to construct multiscale basis functions is equal to the number of local cell problems and local eigenvalue problems. Given the 100 coarse elements, there are 1300 PDEs, including 1200 (100×12) local cell problems and 100 (100×1) local eigenvalue problems.

Deep Learning (DL) techniques are a mathematically complex evolution of classical Machine Learning (ML) methods, which have become increasingly reliable in the last few years (Abad et al., 2022; Mardanirad et al., 2021; Wang et al., 2019b; Yu et al., 2020). Based on how an algorithm learns from the available data records, there are four types of DL algorithms: supervised, unsupervised, semi-supervised, and reinforcement. Our problem in this study is a supervised type. There are some methods in this category, such as CNNs and Recurrent Neural Networks (RNNs). RNNs are usually used while dealing with video, sound, or text data. On the other hand, CNNs are specifically designed for problems with two-dimensional arrays like our regression case, mapping an input of 100*9 to an output of 900*1. Convolutional neural networks enable us to use two-dimensional convolutional filters while developing a model. Furthermore, there is a reasonable and robust mathematical procedure behind convolutional filters. A CNN also automatically and adaptively learns the spatial hierarchies of features. Finally, it can decrease the number of parameters without reducing the quality of models. Therefore, the main contribution of this paper is to develop CNN models, instead of PDE solvers, to predict multiscale basis functions for the mixed GMsFEM. The developed models can accelerate the mixed GMsFEM. With respect to the work presented here, 249,375 samples were generated based on the permeability field covering a wide range of values for the matrix (K_m: 1, 2, 3, 4, and 5 millidarcy) and fracture (K_f: 500, 750, 1000, 1250, 1500, 1750, and 2000 millidarcy).

The paper continues with Section 2, providing relevant published research. A description of the generated data using MatLab software is given in Section 3. Section 4 describes the concepts of a typical CNN, and then presents the characteristics of the four constructed models for multiscale basis functions prediction. The statistical-graphical results and discussion are presented in Section 5. The article is concluded in Section 6, which also identifies opportunities for future studies.

2. Related research

Numerical simulation on coarse grids assisted by data-driven methods is becoming increasingly popular. Three examples are given to demonstrate the application of such techniques in the upscaling domain. (Bohne, 2018) investigated the performance of ordinary least squares and Kernel Ridge regression algorithms for the computation of upscaled permeability. For this, 100 samples were generated, with a permeability field produced using the Gaussian probability distribution and an upscaled permeability which was calculated using a finite volume method. The results demonstrated that Kernel Ridge regression performed with a lower error to capture the

upscaled behavior of the flow (i.e., the underlying physics of the problem) compared to the other technique.

(He et al., 2020) automated permeability upscaling from the detailed geological characterization of fractured reservoirs expressed by a Discrete-Fracture Model (DFM). A DL-based design was developed to find a suitable relationship between DFM images (input) and the equivalent continuum model (output), which contained the estimated equivalent permeability of every grid block. The suggested upscaling workflow had three stages: (i) data generation, (ii) model development, and (iii) validation. The 10,000 samples were generated to train a model with 18 hidden layers. The performance of the model was tested in four cases, and it was found to perform better than static and flow-based upscaling methodologies.

The Darcy Brinkman‐Stokes model was combined with decision tree multivariate regression, to upscale a microporous carbonate from the pore scale to the Darcy scale in a study by (Menke et al., 2021). Using a limited number of training images, the developed model performed as well as numerical upscaling to predict permeability. The main advantage was that the regression model was around 80 times less computationally prohibitive than the numerical methods.

These nine cases are related to assisting the data analysis in multiscale methods. (Chan and Elsheikh, 2018) concentrated on the multiscale finite volume method presented by (Jenny et al., 2003). Generally, a series of local problems over dual-grid cells have to be solved to extract coarse scale basis functions. The researchers used shallow neural networks adjusted by a series of solution samples for the computation of basis functions. The results of this data-driven method over a computational domain $[0, 1]^2$ were promising for elliptic problems.

When the permeability field is fixed, the main quantities of GMsFEM can be precomputed in an offline stage by solving local problems. Given several choices of permeability fields, however, repetitively formulating and solving such local problems might be computationally expensive. (Wang et al., 2019a) presented a DL-based approach for the rapid prediction of the GMsFEM ingredients for any online permeability fields. Deep structures were used to illustrate non-linear mapping from the fine scale permeability field coefficients to the key factors of the multiscale basis functions and the coarse scale variables. The results showed that if samples of the GMsFEM discretizations were adequate, the developed models would be able to provide exact approximations.

(Wang et al., 2020a) investigated a new deep network to a model reduction technique of non-local multi-continuum upscaling developed by (Chung et al., 2018) for multiscale simulations. The input-output maps were developed on a coarse block and trained by applying multi-layer neural network approaches. Soft thresholding operators were selected as an activation function. By relating a soft-thresholding neural network and minimization of PDE solutions, multiscale features of coarse element solutions were extracted, establishing a reduced order model for solution approximation. The numerical results confirmed the good performance of this method.

Various deep networks were used to approximate flow and transport formulae by (Wang and Lin, 2020). Implementing the sparsity structures of the underlying discrete systems, these networks had far fewer learnable parameters compared with fully connected models. Deep learning was used to

approximate the map from the source terms to the velocity solution. The networks were developed with convolutional and locally (not fully) connected layers to execute model reductions. Furthermore, a custom loss function was defined to apply the local mass conservation constraints. The achieved velocity fields were then fed into the saturation equation, and a residual network served to approximate the dynamics. Numerical results of both the single-phase and two-phase cases highlighted the huge potential of novel models to precisely forecast the underlying physical system and make computational efficiency improvements.

A multi-agent Reinforcement Learning (RL) methodology was presented to hasten the multi-level Monte Carlo Markov Chain (MCMC) sampling algorithms by (Chung et al., 2020). The authors confirmed their approach by solving an inverse, multiscale problem, which classical MCMC techniques struggle with. The first issue was computing the posterior distribution, which required a lot of time for heterogeneous media. To solve this, a GMsFEM was used as the forward solver. Moreover, finding a function able to generate meaningful sampling was not simple. For this, an RL policy was learned as a proposal generator. Experimentation revealed that this approach was capable of considerably improving the sampling process.

A DL-based method within GMsFEM was presented by (Zhang et al., 2020) to cluster (coarsen) the uncertainty space. By doing so, the number of multiscale basis functions per coarse element could decrease over the uncertainty space. Convolutional neural networks were joined with some methods in adversary neural networks. Simulation runs were carried out based on almost 240,000 local spatial fields. The numerical results confirmed that an increase in the number of clusters from 5 to 11 could decrease relative error.

A combination of DL techniques and local multiscale model reduction methods was used by (Wang et al., 2020c) to predict flow dynamics considering observed data and physics-based modeling concepts. Flow dynamics can be viewed as a multi-layer network. This means that the solution (e.g. saturation) at the time instant 'n+1' relies on the solution of 'n' as well as input parameters. Each layer is considered as a nonlinear forward map, and the number of layers relates to the internal time steps. Multi-layer neural networks find a nonlinear mapping between the time steps. Reduced-order models provide important coarse grid parameters and some other information. Three examples of numerical simulations demonstrated the efficacy of this hybrid methodology.

A Theory-guided Neural Network (TgNN) which incorporated scientific knowledge into traditional neural networks was developed for the simulation of subsurface flow by (Wang et al., 2020b). The Karhunen-Loeve expansion was used to parameterize heterogeneous porous media. The potency of the suggested framework was evaluated by multiple cases, such as predicting future responses, training from noisy data, and transfer learning. In comparison with ordinary models, TgNN produced more precise outputs. For example, the R^2 values were 0.484 for the artificial neural network and 0.996 for the TgNN to predict future responses with changed boundary conditions.

(Chung et al., 2021) proposed a multi-stage DL-based method for multiscale problems. Each stage shared a similar structure and estimated the same reduced order model of the problem with

multiscale features. The prediction of the first stage was quite imprecise. It was demonstrated numerically that performance improvements could be achieved using several reduced order models as inputs at each stage. The proposed strategy yielded good outputs on two time dependent linear and non-linear PDEs with the steady state condition based on 1600 samples.

3. Data description

In order to construct a reliable CNN-based model, it is desirable to take into account a wide range of the input/output variables. A heterogeneous parameter field can be viewed as a realization of a random field following a specific distribution with the corresponding covariance. In this study, the Karhunen-Loeve expansion (Fukunaga and Koontz, 1970) was used to parameterize the heterogeneous model. This Gaussian random field generation method decomposes a random process into the eigenvalue and eigenfunction of its covariance kernel. The computational domain was set to be $\Omega = [0, 1]^2$. A 30×30 uniform mesh was selected for the fine grid system, and a 10×10 uniform mesh was used for the coarse grid system, meaning that each coarse grid contains nine fine grids. A coarse grid may contain fractures or not. If so, it can include partial or complete fractures. Also, a fracture can pass a fracture. The K_m had five different values: 1, 2, 3, 4, and 5 millidarcy (5 cases), and the values of 500, 750, 1000, 1250, 1500, 1750, and 2000 millidarcy (7 cases) were considered for K_f. The number of fractures available in a porous medium was set to 1, 2, …, 24, and 25 (25 cases). The length of fractures was randomly distributed. For every randomly produced porous medium, there were five basis functions named bases 1, 2, 3, 4, and 5. The first one was a piecewise constant, containing only 1 and -1. Because these two values were the same for the finite element method, there was no need to train this basis. Basis functions 2, 3, 4, and 5 lie between -1 and +1 (i.e. -1 < basis functions < +1). **Fig. 1** shows a permeability field of the fractured porous media with K_m of 4 millidarcy, K_f of 2000 millidarcy, and the number of fractures is 15. Although multiscale basis functions are defined on the coarse grids, their coordinates are also important. For example, coarse grids no. 10 and no. 91 have the same permeability, but their corresponding basis functions may have been different.

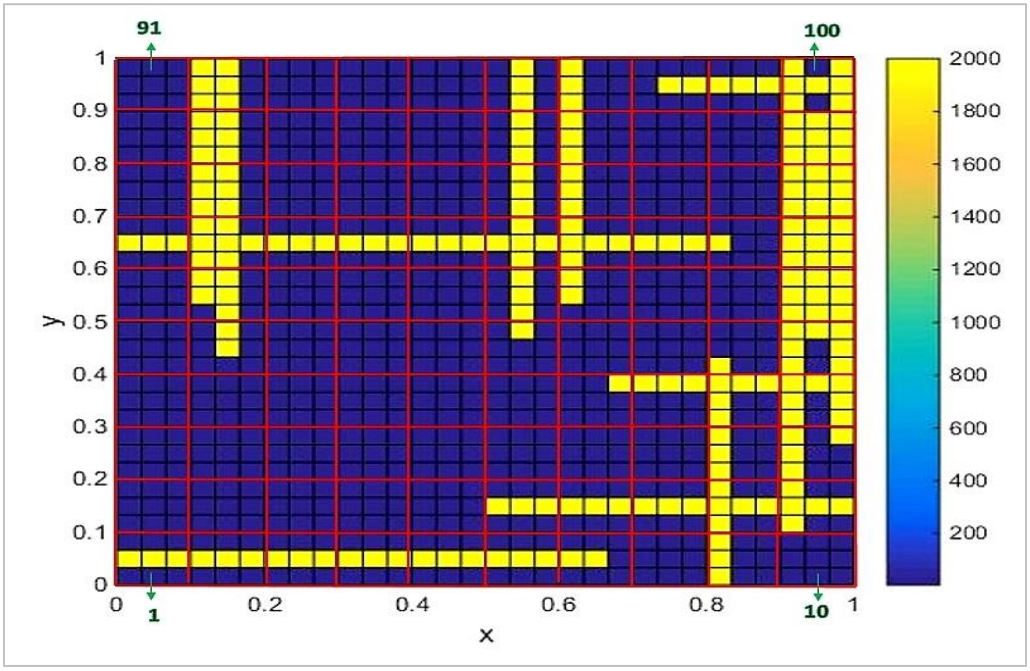

Fig. 1. A permeability field of a fractured porous medium with K_m of 4 millidarcy and K_f of 2000 millidarcy. The fine grid squares in blue refer to the matrix and those in yellow to the fracture. The red grid indicates the coarse grid. Each coarse grid square contains nine fine grids. The number of fractures is 15.

The values of permeability fields were initially in a 900×1 one-dimensional tensor (vector). This changed to a 100×9 two-dimensional tensor, where 100 refers to the number of coarse grids and 9 indicates the number of fine grids in each coarse grid. In other words, each row belongs to a coarse grid. This enabled us to use two-dimensional convolutional filters while developing a model. However, basis functions remained in the form of a 900×1 vector.

Convolutional neural networks are developed using training, validation, and testing data all drawn from the same data distribution. The training subset is utilized for training the model. To develop robust CNN models, so that they are fit for purpose in the context of real-life settings, the data used to train the model must be large. The model is evaluated during the training process using the validation data. The testing data (unseen data) is only used to appraise the model's performance once the training process has been completed.

For each of the 875 (equals to 5×7×25) cases, the MatLab code was run as many as 280 times for the training data, two times for validation, and three times for testing. So, 249,375 samples were generated with 245,000 examples for training, 1750 for validation, and 2625 for testing. Since the permeability fields were randomly generated, it could have included duplicates. These were removed to avoid giving samples an advantage or bias when running the algorithm. As a result, 6653 training, 8 validation, and 13 testing samples were excluded. This reduced the training, validation and testing samples from 245,000, 1750, and 2625 to 238,347, 1742, and 2612, respectively.

4. Methodology

In this section, it is explained how a CNN architecture is designed and what characteristics the optimum network has. Then, it is compared with two advanced CNN architectures. The third subsection is related to network optimization.

4.1 Architecture design

A typical CNN architecture may contain three types of layers: (i) convolutional, (ii) pooling, and (iii) fully connected. In mathematical sciences, convolution is a specialized linear operation on two functions that gives a third modified function. In the context of CNN, the fundamental idea is to consider an input (an array of numbers) as the first function and a convolutional filter (kernel) as the second. A kernel is a relatively small array of randomly generated numbers. The kernel moves over the whole input. The dot product of the kernel and input is calculated at each sub-region (with the same size as the kernel) of the input, obtaining an output value in the corresponding location of the convolved input. This process produces a feature map and is performed using different kernels. The outputs of the convolution process are passed through an activation (transfer) function. Such functions typically transform a linear operation into a nonlinear system (Elgendy, 2020; Yamashita et al., 2018).

The key difference between a parameter and a hyperparameter is that a model's parameters are automatically updated during the training process, whereas hyperparameters are set manually before the model begins training like the size and number of kernels. Including more convolutional layers in a CNN model increases the number of parameters. The more parameters there are in a model, the more computationally expensive the learning process is. This is where a subsampling operation can be useful. In DL, pooling layers use statistical functions (maximum and average pooling) to decrease the number of trainable parameters. This can decrease the computational complexity of mathematical operations and sometimes improve the robustness of feature maps. Pooling layers come after convolutional layers (Elgendy, 2020; Yamashita et al., 2018).

No matter how many feature maps there are in the final convolutional or pooling layer, they are first flattened to a one-dimensional array, and then connected to FC layers. In FC layers (dense layers), each neuron of a layer is connected to whole neurons in the previous layer and next layer. It is common to put a dropout layer after each FC layer (except the output layer) at the end of a CNN model. Dropout omits a percentage of neurons in the previous FC layer. This percentage, as a hyperparameter, is defined when constructing a network. During the training process, some neurons may dominate, producing errors. Dropout balances a network, checking that all neurons work equally to minimize the cost function as much as possible (Elgendy, 2020; Yamashita et al., 2018).

Various cases containing convolutional, pooling, batch normalization, FC, and dropout layers with some techniques to prevent the over-fitting issue such as regularization, were tested separately for each basis function to determine whether they met their corresponding optimal architecture. In the regularization method, an extra element is added to the loss function. This regularization term penalizes a model for using higher values than needed in the weight matrix. The same optimal architecture was coincidentally obtained for all bases 2, 3, 4 and 5 (**Fig. 2**). It has five convolutional and two FC layers. The number of kernels in each convolutional layer is 5, 10, 15, 20, and 25,

respectively. To determine the size of a convolution output for an input with the size of I_h(height)×I_w(width) and a kernel with the size of K_h×K_w, we can use Eq. (4) if the padding is set to 'valid':

$$output\ height = O_h = \frac{I_h - K_h}{S_h} + 1$$
$$output\ width = O_w = \frac{I_w - K_w}{S_w} + 1 \quad (4)$$

where S_h and S_w are the vertical and horizontal strides. When padding is set to 'same', the size does not change. The kernel size for all convolutional layers is 3×3, and $S_h = S_w$= 1. The padding was set to 'same' only for conv5. This means there was no padding for the first four convolutional layers. Therefore, conv1, conv2, conv3, conv4, and conv5 have the size of 98×7, 96×5, 94×3, 92×1, 92×1, respectively. A batch normalization layer can be added after each convolutional layer without changing its size. Neural networks use higher learning rates and converge faster by normalization of the input layer. It maintains the output average at a value close to zero and the standard deviation close to one. The two FC layers contain 2000 neurons. The models use the activation function of 'Rectified Linear Unit (ReLU)' for the convolutional layers, 'sigmoid' for the FC layers, and 'linear' for the output.

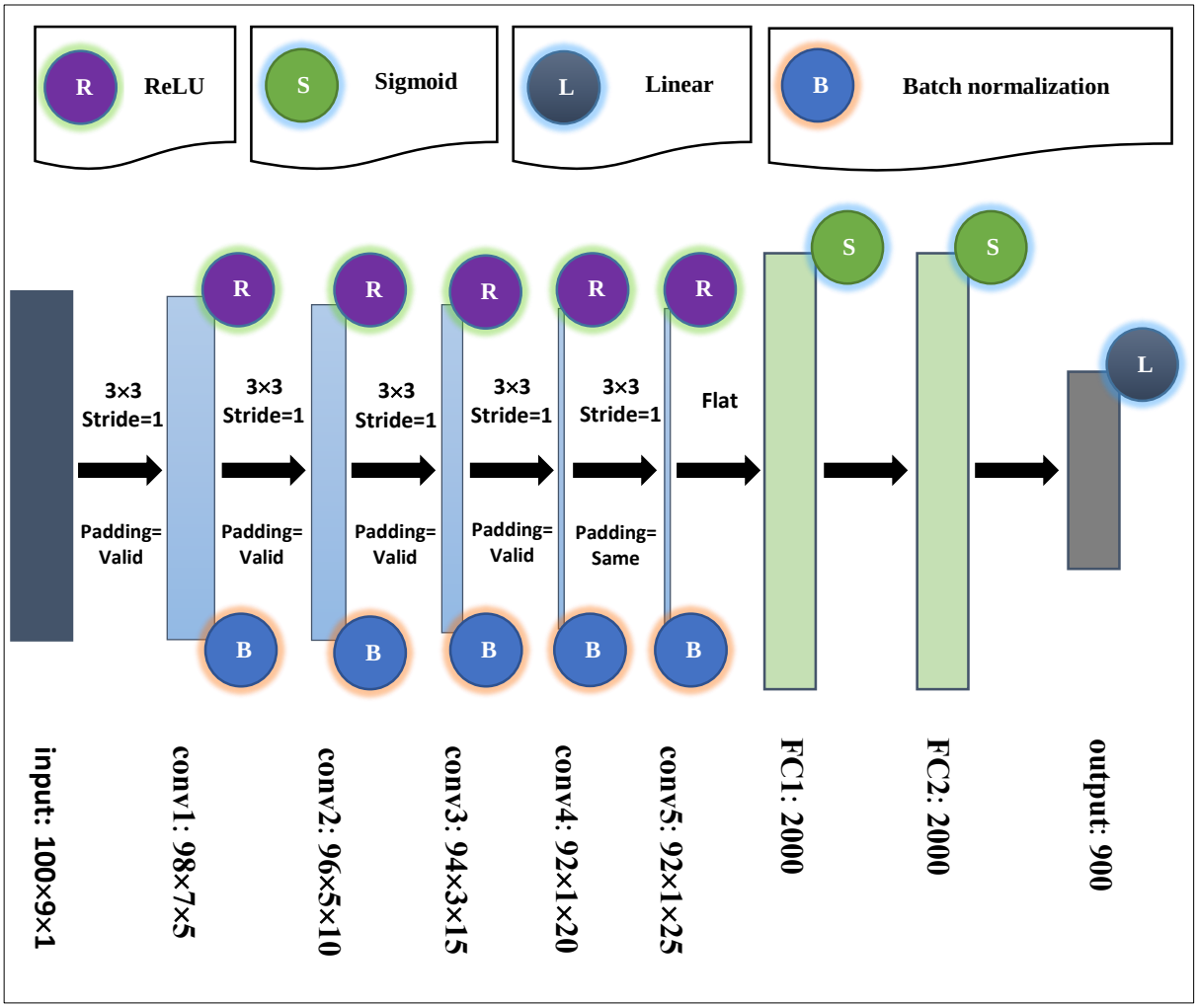

Fig. 2. CNN architecture for constructing the basis functions of the mixed GMsFEM.

4.2 Comparison of the developed architecture with AlexNet and VGGNet

In order to better understand the architecture developed in this study, it is compared to structurally similar CNN architectures AlexNet (Krizhevsky et al., 2012) and VGGNet, also known as VGG16 (Simonyan and Zisserman, 2014). AlexNet has five convolutional layers, three of which are followed by maximum pooling layers to decrease the computational cost. The number of kernels in each convolutional layer is 96, 256, 384, 384, and 256. There are two FC layers of 4096 neurons, and a 1000-neuron output layer at the end of the network. VGGNet contains thirteen convolutional layers, five maximum pooling layers, two FC layers of 4096 neurons, and an output layer with 1000 neurons. The number of kernels used in the convolutional sections is 64, 128, 256, and 512. Each image has two basic elements: depth and size (height×width). Depth of an input image refers to the color channel, where one and three are used for grayscale and color images, respectively. For the later layers, the resulting feature maps indicate how deep a convolutional layer is. Similar to common CNN architectures, going deeper through the structure of developed models, the number of feature maps increases and their size decreases. However, the number of feature maps (equals the filters number) defined in this research is significantly less than that of common CNN

models. In DL, pooling layers are primarily used to decrease the number of trainable parameters, mostly when the input shape is high, e.g., in AlexNet whose input shape is 224*224. However, the input dimension in this research is 100*9. This is why there is no pooling layer in our developed models. Batch normalization, similar to AlexNet, has helped to prevent over-fitting. As with AlexNet and VGGNet, the number of neurons (units) remained constant in FC layers, but no drop out layer was used in the proposed structure because it had a negative effect on the performance. The base structure of this work is for a regression-type problem, while AlexNet and VGGNet were essentially designed for a classification intent. Therefore, a linear activation function is used in our model for the output layer, but a softmax in AlexNet and VGGNet.

4.3 Network optimization

Training a CNN model refers to finding the weights and bias of the kernels in the convolutional layers and of neurons (nodes) in the FC layers so as to minimize the difference between the outputs of a model and actual values as much as possible. Backpropagation is one of the most extensively used methods for training neural networks. It calculates the gradient of the loss function (cost function) using the values assigned to the weights and biases. The loss function is a measure of how well an algorithm models a training dataset by evaluating the similarity between real and predicted outputs. There are various parameter optimizers to reach the minimum loss value. Neural networks generally include a feed-forward pathway which arranges various layers and initializes the parameters, as well as a backward pathway which progressively modifies parameters, thus gradually improving a model's performance. The CNN models in this study were constructed using Keras with TensorFlow as a backend on Python 3.5. The libraries of 'numpy', 'pandas', 'sklearn' and 'glob' were also used in the 'Spyder' module of Anaconda Distribution. The models were compiled using 'MSE' as the loss (objective) function and 'Adam' as the optimizer, and trained with a batch size of 32 samples. Adam uses a distinct learning rate for each scalar parameter and adapts these rates during the whole training process considering the historical values of the partial derivatives of each parameter. This gradient-based algorithm combines the ability of (i) AdaGrad to handle sparse gradients and (ii) RMSProp to function in online and non-stationary settings. Additionally, Adam is suitable for ML problems with large data/parameters. In this study, default values were used i.e., the initial global learning rate=0.001, beta_1=0.9, beta_2=0.999, and epsilon=1e-7. An advantage of Adam is that the learning rate can decrease closer to the optimum point. However, there is an increase in the learning rate in some cases which can damage the algorithm's overall performance. AMSGrad is a new version of Adam which improves convergence. AMSGrad keeps the maximum of all second momentum vectors until the present time step and uses it to normalize the running average of the gradient instead of the second momentum vector of each time step in Adam. This prevents an increasing learning rate and avoids the pitfalls of Adam.

Fig. 3 presents a flow diagram illustrating the CNN analysis methodology applied to an extensive data set of 249,375 samples. This step-by-step approach was separately done to develop an optimal model for each individual basis function. In this regard, we started with a low number of epochs to ensure the model was still improving based on the obtained MSE. Hence, the number of epochs increased up to 100 as a stopping criterion that makes an optimizer terminate the process. If a predictive model fails to correctly capture the underlying trend of the training (seen) data set, it is

considered to ‘under-fit’ the data. This is usually the result of designing an oversimplified model. To avoid under-fitting, including further layers and increasing the number of input parameters may be helpful. Once the model performs favorably on seen data, it can be applied to test (unseen) data to evaluate its generalizability. Over-fitting is a fundamental problem in ML, especially in DL due to a large number of trainable parameters. It happens when a model performs well with respect to the training data but poorly on the test data set. This could be because the model attempts to learn the noise patterns available in the unseen data.

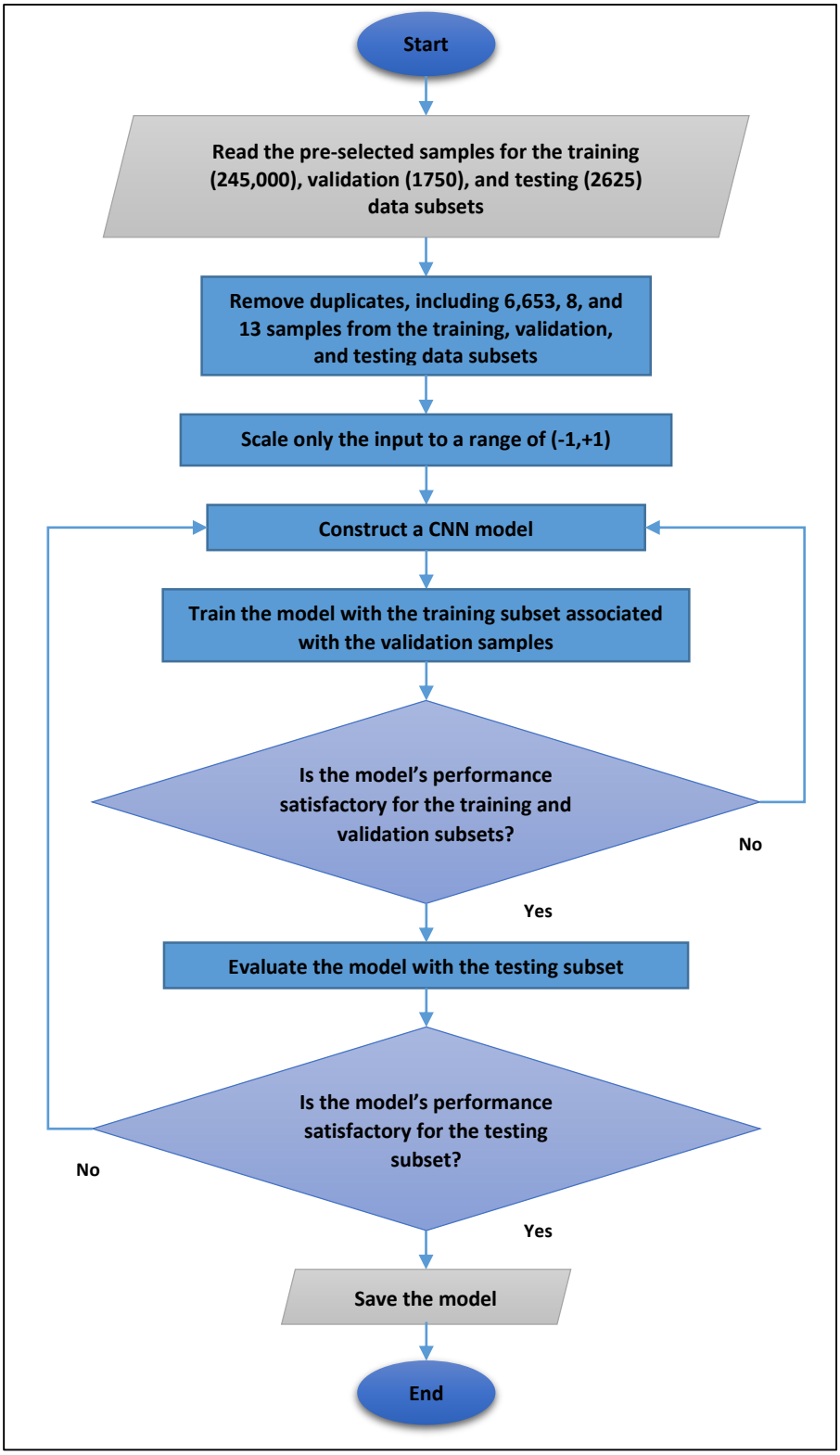

Fig. 3. Graphical representation of different steps for reconstructing basis functions using a CNN model.

5. Results and discussion

The accuracy of the developed models is investigated statistically and graphically. Analysis using widely used prediction error accuracy parameters provides great insight into the performance of the four models. The statistical error metrics considered are (Eqs. 5 and 6):

$$Coefficient\ of\ determination\ (R^2) = 1 - \frac{\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}{\sum_{i=1}^{N}(\hat{y}_i - \bar{y})^2} \quad (5)$$

$$Mean\ Squared\ Error\ (MSE) = \frac{\sum_{i=1}^{n}[\hat{y}_i - y_i]^2}{N} \quad (6)$$

where y_i, $\bar{y}$, and $\hat{y}_i$ are the actual basis function, the average of actual basis function for all samples, and the predicted basis function, respectively. As mentioned earlier, each basis function is in the form of a 900×1 one-dimensional tensor and R^2 of all outputs are averaged, weighted by the variances of each individual output. The R^2 value lies between -∞ and 1. The closer this value is to 1, the more precise predictions a model yields. Conversely, a negative or small positive R^2 value indicates that the wrong model is chosen. The MSE measures the average of squares of errors (i.e., the difference between predicted and real values). It is always non-negative and values close to zero indicate a good performance.

The performance of the constructed models by Adam in terms of the R^2 and MSE for the training, validation, and testing subsets, and the total dataset is separately shown in **Figs. 4 and 5**. According to **Fig. 4**, the models developed for basis functions 2, 3, and 4 have an R^2 of nearly 0.9; thus, they are able to predict training samples very well. The model designed for the basis function 5 has an acceptable performance, but is not as good as the others. Furthermore, it reveals the superior performance of the CNN model for basis function 4, in validation and testing subsets in terms of the R^2 error parameter. The performance of the developed models based on the total dataset is the same as the training subset. This is because a large proportion of the data generated in MatLab software was designed to train the model. **Fig. 5** also demonstrates that the MSE performance of models created for basis functions 4 and 5 are better than those for basis functions 2 and 3. In this regard, the CNN model of basis function 5 is the best with 0.0108, 0.0158, and 0.0154 for the training, validation, and testing data subsets.

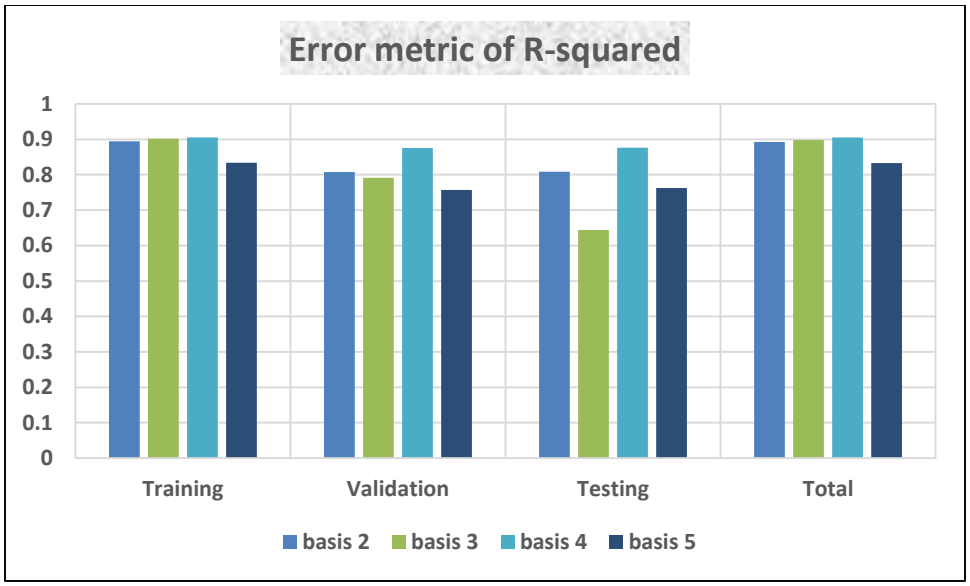

Fig. 4. Performance of the developed models by the Adam algorithm based on R^2.

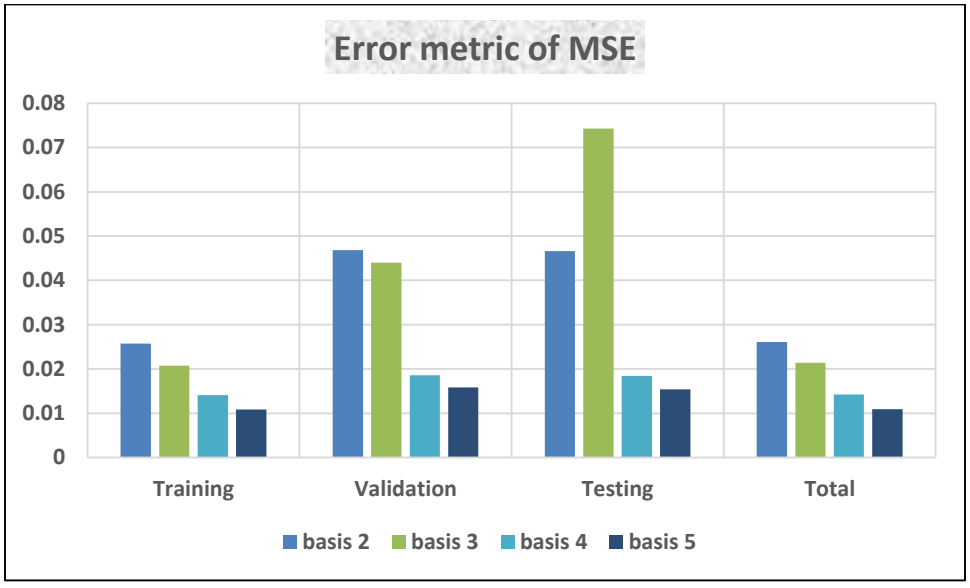

Fig. 5. Performance of the developed models by the Adam algorithm based on MSE.

Table 1 is given to investigate if replacing the Adam optimizer with AMSGrad affects the performance of the CNN architecture. The values in green rows are related to Adam, and those of orange rows are for AMSGrad. Comparing the R^2 parameter, it is clear that the performance of all models improves, especially for the training subset. For instance, the R^2 increases from 0.8945 to 0.9079 for basis 2. The MSE error parameter decreases as expected when using the AMSGrad optimizer. Across the total data, the MSE decreases from 0.0261, 0.0214, 0.0142, and 0.0109 to 0.0206, 0.0167, 0.0103, and 0.0078 for bases 2, 3, 4, and 5, respectively.

Table 1 Performance of Adam and AMSGrad on the basis functions of the mixed GMsFEM.

Error statistic	CNN model	Training	Validation	Testing	Total
R^2	basis 2 (Adam)	0.8945	0.8076	0.8083	0.8929
	basis 2 (AMSGrad)	0.9079	0.8141	0.8156	0.9062
	basis 3 (Adam)	0.9017	0.7911	0.6445	0.8981
	basis 3 (AMSGrad)	0.9142	0.7974	0.6503	0.9105
	basis 4 (Adam)	0.9054	0.8751	0.8762	0.9049
	basis 4 (AMSGrad)	0.9172	0.8777	0.8815	0.9165
	basis 5 (Adam)	0.8341	0.7569	0.7625	0.8328
	basis 5 (AMSGrad)	0.8449	0.7592	0.7671	0.8434
MSE	basis 2 (Adam)	0.0257	0.0468	0.0466	0.0261
	basis 2 (AMSGrad)	0.0202	0.0441	0.0445	0.0206
	basis 3 (Adam)	0.0207	0.044	0.0743	0.0214
	basis 3 (AMSGrad)	0.0159	0.0401	0.0697	0.0167
	basis 4 (Adam)	0.0141	0.0186	0.0184	0.0142
	basis 4 (AMSGrad)	0.0102	0.0158	0.0169	0.0103
	basis 5 (Adam)	0.0108	0.0158	0.0154	0.0109
	basis 5 (AMSGrad)	0.0077	0.0137	0.0141	0.0078

The permeability field consists of a 30×30 uniform mesh on a fine grid system. This is equivalent to a 10×10 uniform mesh on a coarse grid system, so each coarse grid contains nine fine grids. Multiscale basis functions, on the other hand, are defined in a single coarse grid element, as mentioned earlier. Consequently, the pattern available in a coarse block is tracked for the graphical investigation, with an unfractured case and a fractured case (representative samples) for each of the training (**Figs. 6 and 7**), validation (**Figs. 8 and 9**), and testing (**Figs. 10 and 11**) subsets. In the top section of the figures, fine grids in blue refer to the matrix and yellow to the fracture. **Figs. 6 and 7** demonstrate an excellent match between actual and reconstructed patterns for coarse blocks no. 71 and 24, except for basis 5 of no. 24. In the validation cases, the CNN models follow the observed trend for homogeneous (unfractured) coarse element no. 44, but behave moderately for no. 13. For the testing subset, two samples (2, and 25) were selected; the performance of the developed models is nearly identical to the validation cases, according to **Figs. 10 and 11**. In general, the patterns reconstructed by AMSGrad follow the observed trend in each coarse block slightly better than those of Adam.

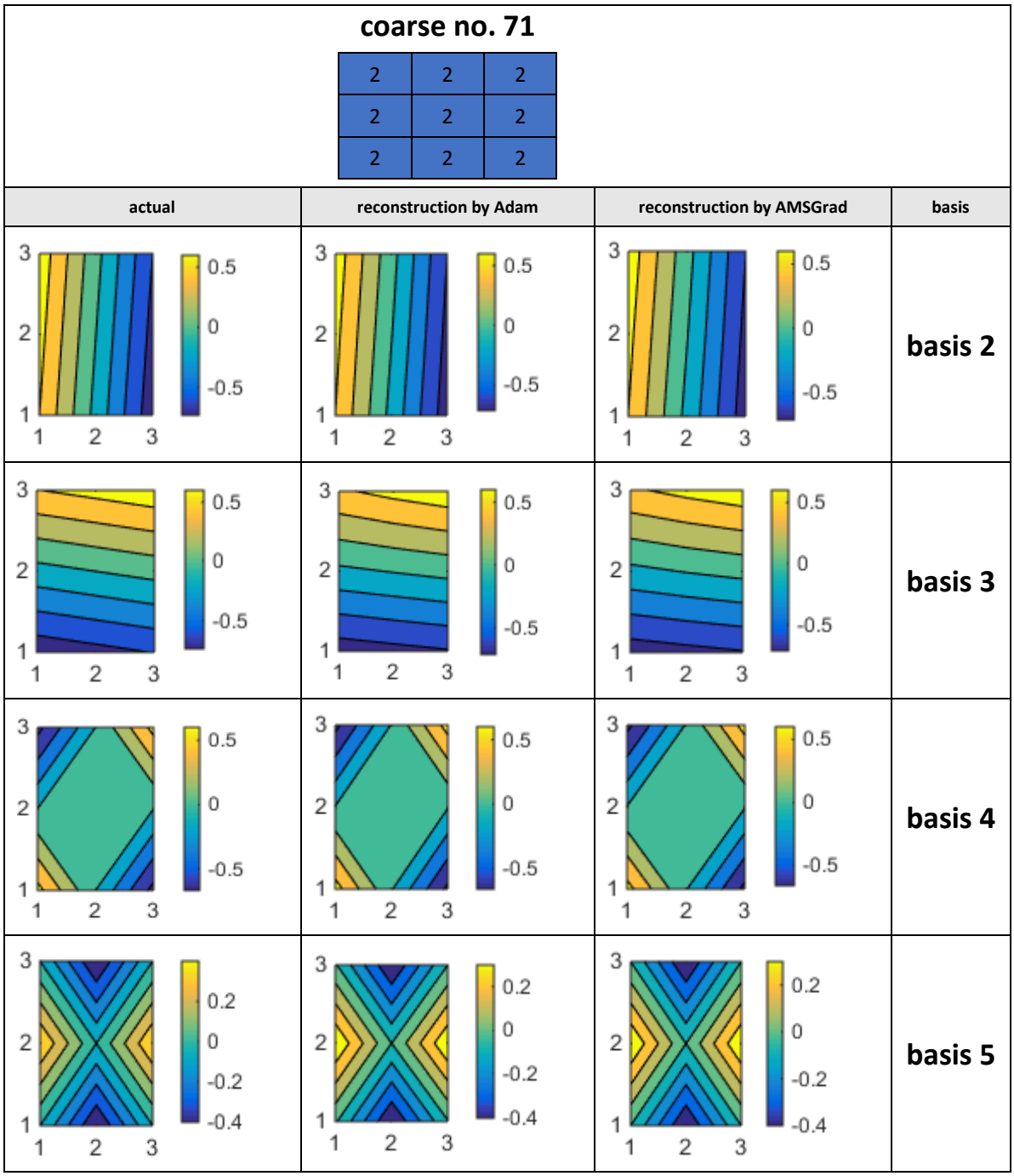

Fig. 6. A comparison between actual and reconstructed patterns for an unfractured case within the training samples.

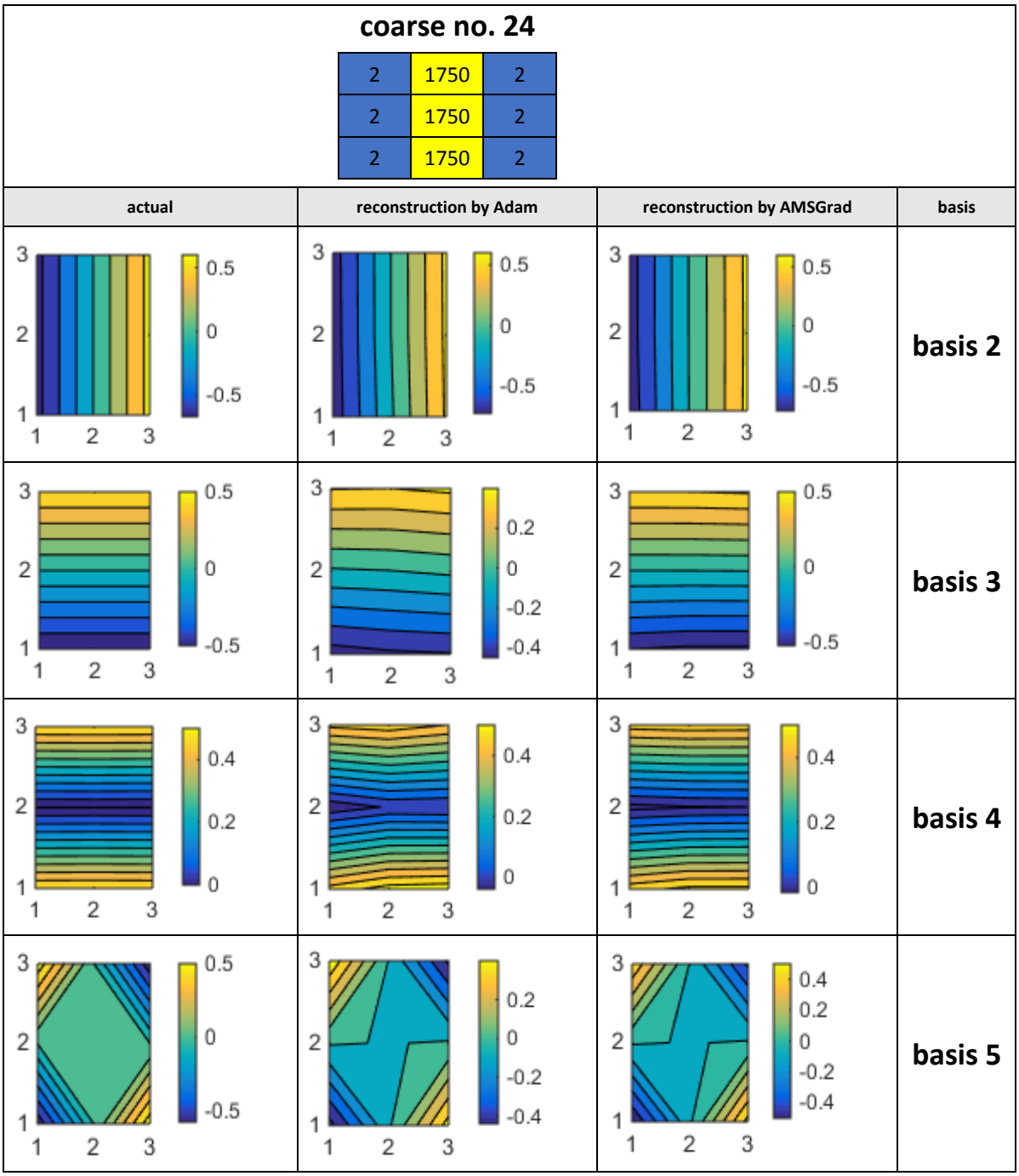

Fig. 7. A comparison between actual and reconstructed patterns for a fractured case within the training samples.

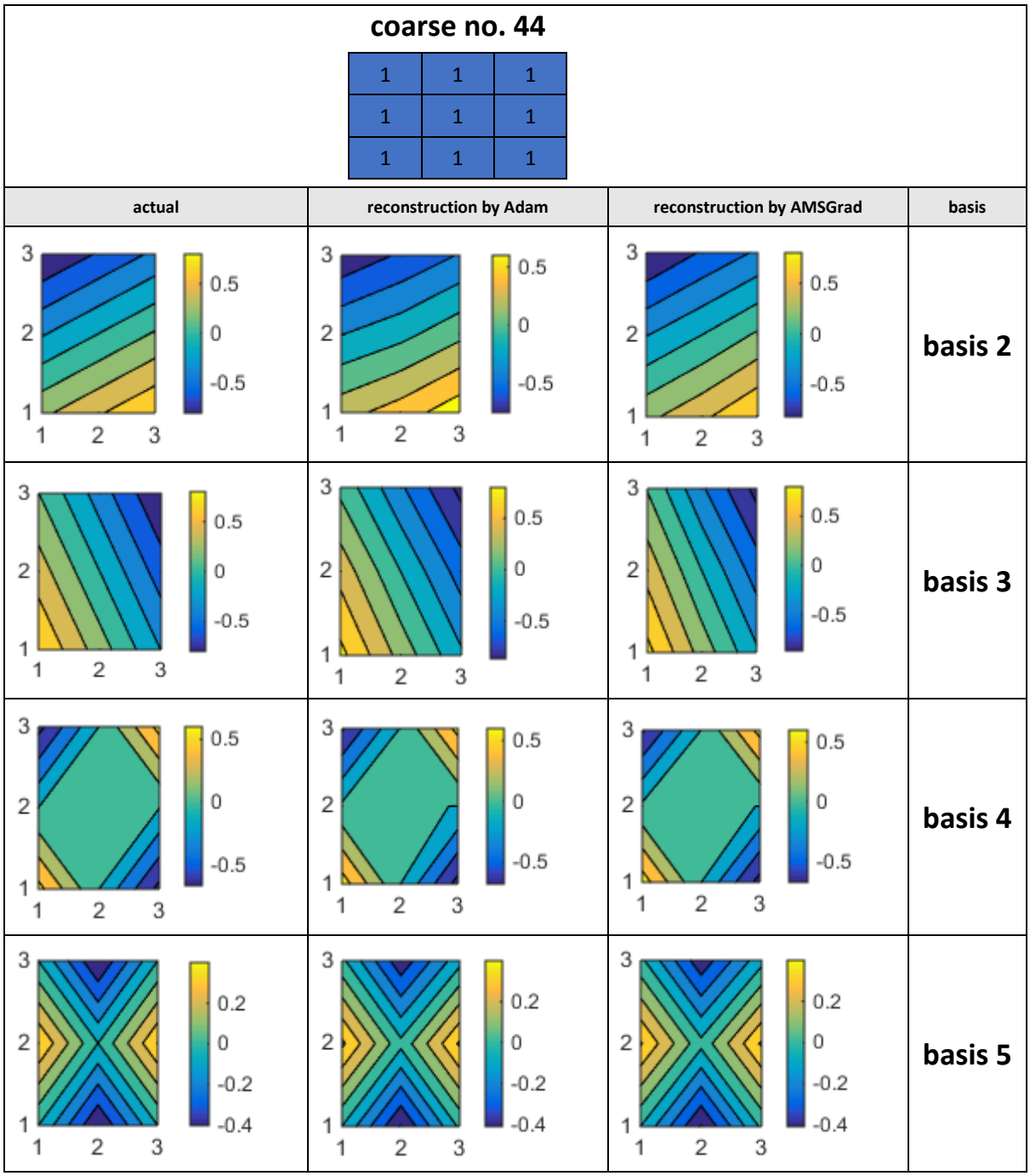

Fig. 8. A comparison between actual and reconstructed patterns for an unfractured case within the validation samples.

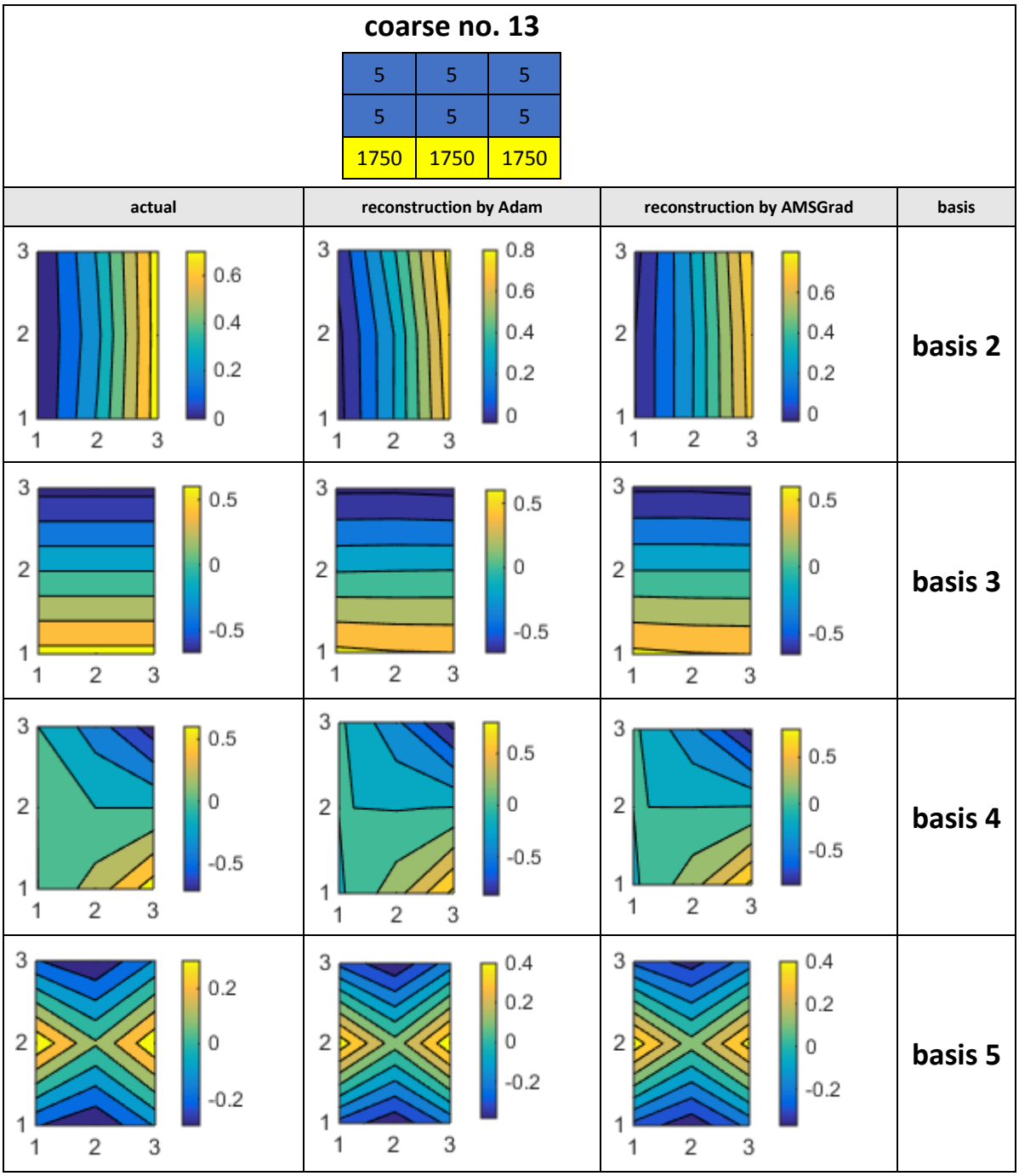

Fig. 9. A comparison between actual and reconstructed patterns for a fractured case within the validation samples.

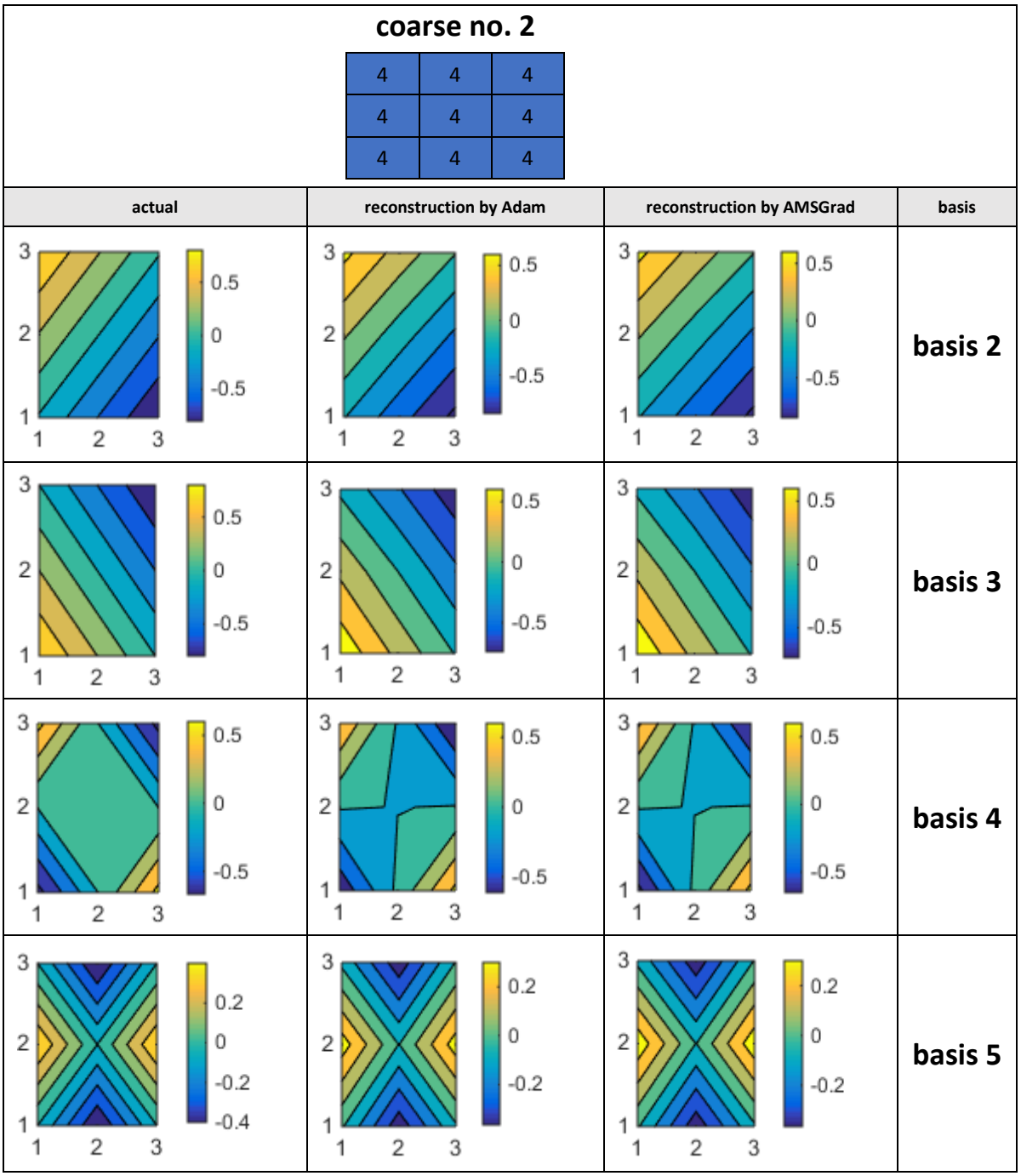

Fig. 10. A comparison between actual and reconstructed patterns for an unfractured case within the testing samples.

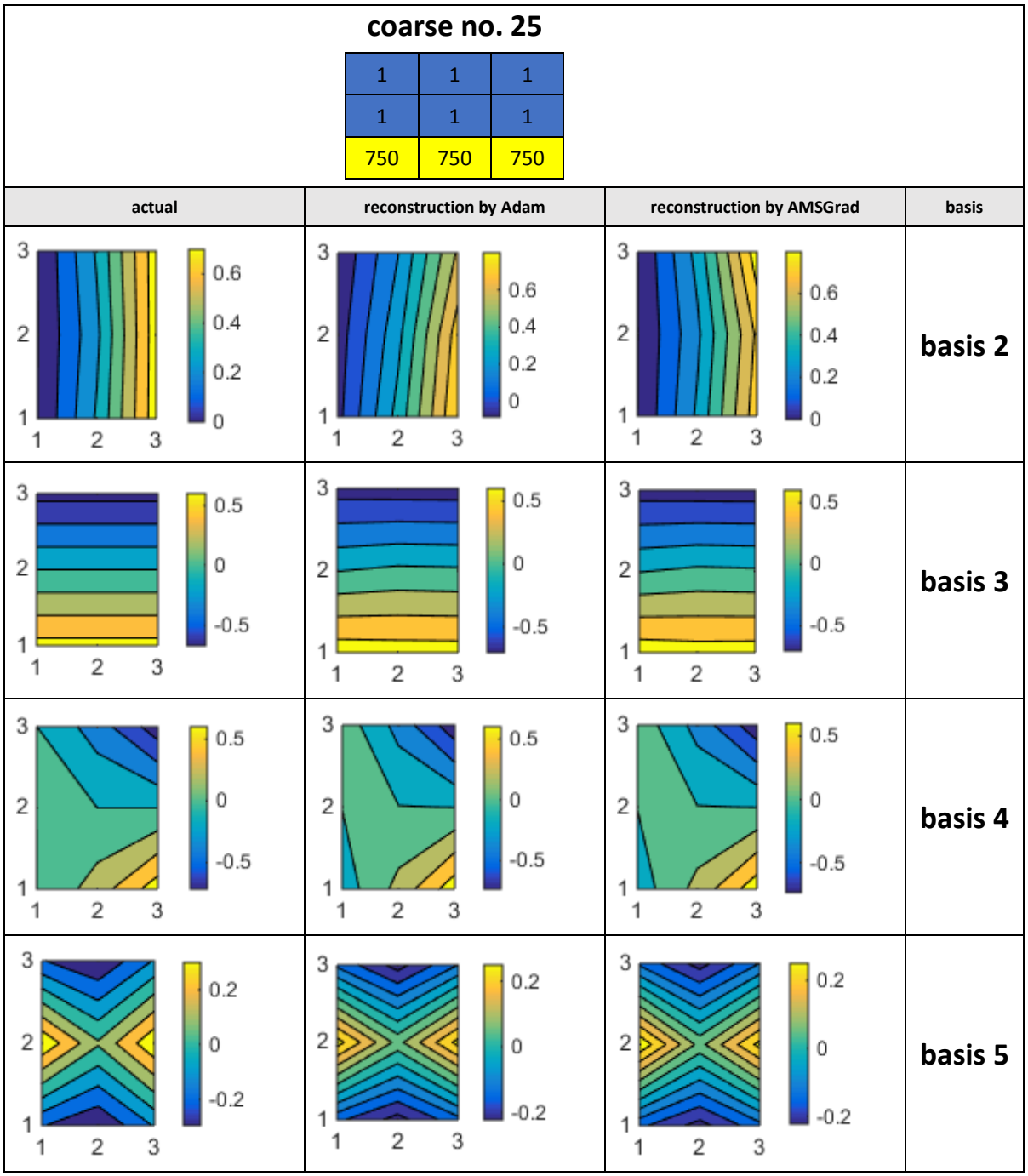

Fig. 11. A comparison between actual and reconstructed patterns for a fractured case within the testing samples.

Deep learning requires a great number of training samples because many parameters (here 10,414,170) must be trained. The process nearly always begins with unsatisfactory accuracy, so an optimizer is employed to converge the network into an optimal solution (satisfactory performance). This might be a local optimum, not the global solution. In this study, the 26,250 samples were initially generated, resulting in models with a poor performance (e.g., the R^2 was negative). Therefore, more and more data was added to the network to better tune the parameters.

Applying 249,375 samples, the results are generally promising, especially for the training samples. However, the accuracy of models over the validation and testing and data sets is not high enough. Machine learning algorithms are prone to both reducible and irreducible errors. No matter which algorithm is used, there is an irreducible error that cannot be removed, caused by the noise in the problem itself. In contrast, the bias and variance errors (together known as the reducible errors) are controllable and can be minimized. The bias refers to the difference between the predictions of a model and the actual values based on training samples. The variance is caused by the sensitivity of a model to small fluctuations in the training set. Generally speaking, simple models such as linear regression have a high bias but a low variance, while complex models have a low bias but a high variance because they might concentrate more than necessary on the training data set and fail to generalize to the unseen data. Therefore, an appropriate bias-variance trade-off can help a model generalize beyond the training subset.

6. Conclusions and further research

Instead of using PDE solvers, four DL-based models, with the same structure and number of parameters but with different parameter values, were constructed from 249,375 samples for the multiscale basis functions in the mixed GMsFEM. The optimum architecture consists of seven weight/bias layers: five convolutional and two FC layers without any pooling layer. The results obtained reveal that the four developed models by either Adam or AMSGrad yield satisfactory outputs. They also accurately follow the pattern available in each coarse block. Mathematically speaking, the mixed GMsFEM can be accelerated by the developed models. In the context of computer science, the paper demonstrates that to develop a highly-accurate model it is not necessary to develop highly complex networks like AlexNet and VGGNet. Three suggestions are provided to extend the current study: (i) training with a greater volume of data to improve the performance of the available models, (ii) developing an ensemble learning model based on the current models, and (iii) working on 3-dimensional porous media with vertical, horizontal, and inclined fractures.

Funding

This work is partially supported by Key Program Special Fund in XJTLU (KSF-E-50, KSF-E-21), XJTLU Postgraduate Research Scholarship (PGRS1912009) and XJTLU Research Development Funding (RDF-19-01-15).

Declaration

The authors confirm that they have no conflicts of interest in regards to the content of this study.

Acknowledgements

We would like to thank Ms. Roslyn Joy Irving and Dr. Trevor Mahy for providing helpful comments and suggestions.

Nomenclature	
B	batch normalization
beta_1	a float value in the Adam algorithm
beta_2	a float value in the Adam algorithm
CNN	Convolutional Neural Network
conv1	first convolutional layer
conv2	second convolutional layer
conv3	third convolutional layer
conv4	fourth convolutional layer
conv5	fifth convolutional layer
DFM	Discrete-Fracture Model
DL	Deep Learning
epsilon	a small constant in the Adam algorithm
FC	Fully Connected
FC1	first FC layer
FC2	second FC layer
GMsFEM	Generalized Multiscale Finite Element Method
I_h	input height
I_w	input width
K	number of kernels
K_f	fracture permeability
K_h	kernel height
K_m	matrix permeability
K_w	kernel width
L	linearkm
M	filters number of the previous convolutional layer
MCMC	Monte Carlo Markov Chain
ML	Machine Learning
MSE	Mean Squared Error
N	filters number of the current convolutional layer
N_c	number of neurons in the current FC layer
N_p	number of neurons in the previous FC layer
O_h	output height
O_w	output width
PDE	Partial Differential Equation
R	ReLU
R^2	coefficient of determination
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
S	sigmoid
S_h	vertical stride

S_w	horizontal stride
TgNN	Theory-guided Neural Network
$\bar{y}$	average of actual basis function for all samples
y_i	actual basis function
$\hat{y}_i$	predicted basis function
Ω	domain size

References

Abad, A.R.B. et al., 2022. Robust hybrid machine learning algorithms for gas flow rates prediction through wellhead chokes in gas condensate fields. Fuel, 308: 121872.

Azad, M.-R., Kamkar-Rouhani, A., Tokhmechi, B. and Arashi, M., 2021. Hierarchical simultaneous upscaling of porosity and permeability features using the bandwidth of kernel function and wavelet transformation in two dimensions: Application to the SPE-10 model. Oil & Gas Science and Technology–Revue d'IFP Energies nouvelles, 76: 26.

Bohne, R., 2018. Machine-learning algorithms for the computation of upscaled permeabilities, NTNU.

Chan, S. and Elsheikh, A.H., 2018. A machine learning approach for efficient uncertainty quantification using multiscale methods. Journal of Computational Physics, 354: 493-511.

Chen, J., Chung, E.T., He, Z. and Sun, S., 2020. Generalized multiscale approximation of mixed finite elements with velocity elimination for subsurface flow. Journal of Computational Physics, 404: 109133.

Chen, Z. and Hou, T., 2003. A mixed multiscale finite element method for elliptic problems with oscillating coefficients. Mathematics of Computation, 72(242): 541-576.

Chung, E., Efendiev, Y., Leung, W.T., Pun, S.-M. and Zhang, Z., 2020. Multi-agent Reinforcement Learning Accelerated MCMC on Multiscale Inversion Problem. arXiv preprint arXiv:2011.08954.

Chung, E., Leung, W.T., Pun, S.-M. and Zhang, Z., 2021. A multi-stage deep learning based algorithm for multiscale model reduction. Journal of Computational and Applied Mathematics, 394: 113506.

Chung, E.T., Efendiev, Y., Leung, W.T., Vasilyeva, M. and Wang, Y., 2018. Non-local multi-continua upscaling for flows in heterogeneous fractured media. Journal of Computational Physics, 372: 22-34.

Efendiev, Y., Galvis, J. and Hou, T.Y., 2013. Generalized multiscale finite element methods (GMsFEM). Journal of Computational Physics, 251: 116-135.

Elgendy, M., 2020. Deep Learning for Vision Systems. Manning Publications.

Fukunaga, K. and Koontz, W.L., 1970. Application of the Karhunen-Loeve expansion to feature selection and ordering. IEEE Transactions on computers, 100(4): 311-318.

Ganis, B., Vassilev, D., Wang, C. and Yotov, I., 2017. A multiscale flux basis for mortar mixed discretizations of Stokes–Darcy flows. Computer Methods in Applied Mechanics and Engineering, 313: 259-278.

Ganjeh-Ghazvini, M., 2019. The impact of viscosity contrast on the error of heterogeneity loss in upscaling of geological models. Journal of petroleum science and engineering, 173: 681-689.

Hajibeygi, H., Bonfigli, G., Hesse, M.A. and Jenny, P., 2008. Iterative multiscale finite-volume method. Journal of Computational Physics, 227(19): 8604-8621.

He, X., Santoso, R. and Hoteit, H., 2020. Application of machine-learning to construct equivalent continuum models from high-resolution discrete-fracture models, International Petroleum Technology Conference. OnePetro.

He, Z., Chen, H., Chen, J. and Chen, Z., 2021. Generalized multiscale approximation of a mixed finite element method with velocity elimination for Darcy flow in fractured porous media. Computer Methods in Applied Mechanics and Engineering, 381: 113846.
Hou, T.Y. and Wu, X.-H., 1997. A multiscale finite element method for elliptic problems in composite materials and porous media. Journal of computational physics, 134(1): 169-189.
Jenny, P., Lee, S. and Tchelepi, H.A., 2003. Multi-scale finite-volume method for elliptic problems in subsurface flow simulation. Journal of computational physics, 187(1): 47-67.
Jenny, P., Lee, S.H. and Tchelepi, H.A., 2005. Adaptive multiscale finite-volume method for multiphase flow and transport in porous media. Multiscale Modeling & Simulation, 3(1): 50-64.
Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25: 1097-1105.
Liu, X. et al., 2021. Numerical upscaling of multi-mineral digital rocks: Electrical conductivities of tight sandstones. Journal of Petroleum Science and Engineering, 201: 108530.
Mardanirad, S., Wood, D.A. and Zakeri, H., 2021. The application of deep learning algorithms to classify subsurface drilling lost circulation severity in large oil field datasets. SN Applied Sciences, 3(9): 1-22.
Menke, H.P., Maes, J. and Geiger, S., 2021. Upscaling the porosity–permeability relationship of a microporous carbonate for Darcy-scale flow with machine learning. Scientific Reports, 11(1): 1-10.
Peszynska, M., 2005. Mortar adaptivity in mixed methods for flow in porous media. Int. J. Numer. Anal. Model, 2(3): 241-282.
Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.
Wang, M. et al., 2019a. Prediction of discretization of gmsfem using deep learning. Mathematics, 7(5): 412.
Wang, M. et al., 2020a. Reduced-order deep learning for flow dynamics. The interplay between deep learning and model reduction. Journal of Computational Physics, 401: 108939.
Wang, N., Zhang, D., Chang, H. and Li, H., 2020b. Deep learning of subsurface flow via theory-guided neural network. Journal of Hydrology, 584: 124700.
Wang, S., Sobecki, N., Ding, D., Zhu, L. and Wu, Y.-S., 2019b. Accelerating and stabilizing the vapor-liquid equilibrium (VLE) calculation in compositional simulation of unconventional reservoirs using deep learning based flash calculation. Fuel, 253: 209-219.
Wang, Y., Cheung, S.W., Chung, E.T., Efendiev, Y. and Wang, M., 2020c. Deep multiscale model learning. Journal of Computational Physics, 406: 109071.
Wang, Y. and Lin, G., 2020. Efficient deep learning techniques for multiphase flow simulation in heterogeneous porousc media. Journal of Computational Physics, 401: 108968.
Yamashita, R., Nishio, M., Do, R.K.G. and Togashi, K., 2018. Convolutional neural networks: an overview and application in radiology. Insights into imaging, 9(4): 611-629.
Yu, Q. et al., 2020. Identification of rock pore structures and permeabilities using electron microscopy experiments and deep learning interpretations. Fuel, 268: 117416.
Zhang, Z., Chung, E.T., Efendiev, Y. and Leung, W.T., 2020. Learning algorithms for coarsening uncertainty space and applications to multiscale simulations. Mathematics, 8(5): 720.