Deep Ensemble Learning for High-dimensional Subsurface Fluid Flow Modeling

Abouzar Choubineh[a,b], Jie Chen[b], David A. Wood[c], Frans Coenen[a], Fei Ma[b]

[a] *Department of Computer Science, University of Liverpool, Liverpool, United Kingdom*
[b] *Department of Applied Mathematics, Xi'an Jiaotong-Liverpool University, Suzhou, China*
[c] *DWA Energy Limited,, Lincoln, United Kingdom*

Abstract

The accuracy of Deep Learning (DL) algorithms can be improved by combining several deep learners into an ensemble. This avoids the continuous endeavor required to adjust the architecture of individual networks or the nature of the propagation. This study investigates prediction improvements possible using Deep Ensemble Learning (DEL) to determine four distinct multiscale basis functions in the mixed Generalized Multiscale Finite Element Method (GMsFEM), involving the permeability field as the only input. 376,250 samples were initially generated, filtered down to 367,811 after data pre-processing. A standard Convolutional Neural Network (CNN) named SkiplessCNN and three skip connection-based CNNs named FirstSkipCNN, MidSkipCNN, and DualSkipCNN were developed for the base learners. For each basis function, these four CNNs were combined into an ensemble model using linear regression and ridge regression, separately, as part of the stacking technique. A comparison of the coefficient of determination (R^2) and Mean Squared Error (MSE) confirms the effectiveness of all three skip connections in enhancing the performance of the standard CNN, with DualSkip being the most effective among them. Additionally, as evaluated on the testing subset, the combined models meaningfully outperform the individual models for all basis functions. The case that applies linear regression delivers R^2 ranging from 0.8456 to 0.9191 and MSE ranging from 0.0092 to 0.0369. The ridge regression case achieves marginally better predictions with R^2 ranging from 0.8539 to 0.922, and MSE ranging from 0.009 to 0.0349 because its solution involves more evenly distributed weights.

Keywords:

Deep ensemble learning, Big data, Linear/ridge regression,
Convolutional neural network, Skip connection, Subsurface fluid flow,
mixed generalized multiscale finite element method.

1. Introduction

The wide range of Machine Learning (ML) algorithms available all have to contend with reducible and irreducible errors. The latter is typically a consequence of noise within the datasets being evaluated and cannot be addressed by the ML models themselves. On the other hand, bias and variance combine to generate reducible errors, which can be effectively reduced by the algorithmic actions of the ML. Bias errors are a consequence of the differences between predicted and actual dependent variable values generated with a training subset of samples. Variance errors result from small fluctuations in the training subsets actual values. Mathematically, we may assume that there is an input vector $\mathbf{X}$ (here the permeability field) that influences an output vector $\mathbf{Y}$ (here the basis function). The function $f(\mathbf{X})$ denotes the correct relationship between the input and output, but it is accompanied by some noise that can be represented by σ_ϵ^2 that constitutes the irreducible error:

$$\mathbf{Y} = f(\mathbf{X}) + \sigma_\epsilon^2 \tag{1}$$

ML models strive to determine the best function $\hat{f}(\mathbf{X})$ that can predict the true underlying function $f(\mathbf{X})$ as precisely as possible. Given the Total Error (TE) as $TE = E[(\mathbf{Y} - \hat{f}(\mathbf{X}))^2]$:

$$TE = [E\hat{f}(\mathbf{X}) - f(\mathbf{X})]^2 + E[\hat{f}(\mathbf{X}) - E\hat{f}(\mathbf{X})]^2 + \sigma_\epsilon^2 \tag{2}$$

$$TE = bias^2 + variance + irreducible\ error \tag{3}$$

Simpler models tend to generate high bias accompanied by low variance. On the other hand, more elaborate models tend to generate lower bias accompanied by higher variance. Linear regression, for example, has a high bias since it tends to oversimplify and, therefore, cannot accurately capture the relationship between input variables and output data. In contrast, artificial neural networks involving multiple hidden layers and many nodes tend to generate substantial variance, because they tend to overfit training datasets, making it difficult for the trained models to be generalized and accurately predict unseen data. High bias is typically a consequence of models underfitting a dataset, whereas high variance is typically a consequence of models overfitting a dataset. As a modeling strategy, it makes sense, therefore, to attempt to trade off bias and variance errors to assist the trained models in being applied in a more generalized way, thereby more accurately predicting data not seen during the training/validation process.

Ensemble learning, where by a number of base learners are combined into an “ensemble” to produce a single model whose predictive or classification accuracy is better than that of the individual component base learners, is a well-established technology (Wang et al., 2011, 2014; Nguyen and Logofătu, 2018; Kanda et al., 2020; Verma and Chandra, 2023; Sahin and Demir, 2023). The majority of published research directed towards ensemble learning has been founded on traditional ML techniques; for example *Random Forests* (Ho, 1998). Such established mechanisms represent Shallow Ensemble Learning (SEL). The alternative is Deep Ensemble Learning (DEL), which combines a number of deep learners into an ensemble. Although Deep Learning (DL) algorithms tend to generate fewer prediction errors than ML methods when applied to many datasets, there is scope to further improve their accuracy. Combining several deep learners into an ensemble is one way to potentially achieve this. Moreover, in a set of deep models, the different strengths of each DL model may complement one another, and weaknesses cancel each other out. Unlike SEL, DEL has received little attention to date. Rather, attempts to improve DL accuracy have been focused on optimizing the various control-parameter values used by deep learners, for example by adjusting the architecture of a DL network or the nature of the propagation applied. This study challenges this view by investigating the performance of DEL. It is not possible to use common ML techniques such as Fully Connected (FC) models for the problem investigated in this paper, which is mapping an input of 100×9 to an output of 900×1. This is mainly because the input is a 2D tensor. Among DL techniques, recurrent neural networks are usually applied to video, sound, or text data. On the other hand, Convolutional Neural Networks (CNNs) are specifically designed for problems with 2D arrays. Therefore, base learners are selected from complex CNN models with different variants. These base learners are then combined using two regression models (linear regression and ridge regression), separately.

A key motivation for this study is to develop DEL models to assist in the prediction of fluid-flow characteristics in subsurface reservoirs. This is of interest to provide more detailed insights to the flow of fluids into producing wells penetrating oil and gas reservoirs, the seepage of fluids through soil, and land subsidence as a consequence of groundwater and oil and gas extraction. With respect to oil and gas reservoirs, the main goal is to predict the performance of reservoirs at any future point in time and to optimize petroleum fluid recovery under different operating conditions. Fluid flow in petroleum reservoirs is typically modelled using a set of non-linear Partial Differential Equations (PDEs). In general, these equations can be solved analytically (exact solutions) or numerically (approximate solutions). However, for reservoir simulation models comprised of many thousands of grid cells the datasets generated to do this involve hundreds of thousands of data points leading to long and tedious calculations. Moreover, the datasets involved are typically too large for ML models to handle. This study provides an innovative solution by developing efficient and reliable DEL models to assist with a specific time-consuming aspect of fluid-flow simulation.

The developed DEL models are applied to a large dataset associated with subsurface fluid flow modeling. This dataset can be modeled in different ways but a mixed Generalized Multiscale Finite Element Method (GMsFEM) is used to generate multiscale data formats. $376,250$ samples (data records) were generated with a 2D permeability field (in a Cartesian coordinate system) representing the input variable and the multiscale basis functions as the output. Traditionally, a substantial number of PDEs need to be solved to produce these functions, but this involves considerable time and computational effort. Deep ensemble learning made up of several deep base models offers an innovative and more effective alternative to PDE solvers, specifically with respect to determining accurate reservoir pressure distributions to improve estimates of resource recovery factors from oil/gas reservoirs.

Following on from this introduction, Section 2 considers ensemble modeling along with published research relevant to it. A review of the mixed GMsFEM is given in Section 3. The characteristics of stacking CNN ensemble models are presented in Section 4. Prediction error evaluation of applications of the developed DEL models to the subsurface fluid flow dataset, in terms of the coefficient of determination (R^2) and Mean Squared Error (MSE), is presented in Section 5. Section 6 discusses the implications of the results generated leading to the conclusions drawn (Section 7).

2. Ensemble Modeling with Its Related Research

Ensemble systems can be categorised according to the method by which the ensemble learning is achieved:

1. **Boosting**: Boosting (Schapire, 1990; Kumari and Toshniwal, 2021) is a sequential method. The different base learners are dependent on each other. The aim is to fit models in steps such that model training at each step is influenced by the models constructed in the previous steps. Each step is focused on examples in the dataset that have been poorly predicted by the previous steps.
2. **Bagging**: Bagging (standing for bootstrap aggregating) (Breiman, 1996a; Tüysüzoğlu and Birant, 2020) is a parallel approach to ensemble learning where multiple models are generated using the same ML algorithm but with different portions of the training data, which

are then merged to produce a single more robust model than the individual base models. Multiple training subsets (bootstrap samples) are randomly selected from the initial training dataset with replacement (a single row of the initial data might be chosen zero, one, two, or even more times). Each model is developed from one subset, resulting in an ensemble of several models. The final prediction is obtained by averaging (or simply ranking) all the predictions of the different learners.

3. **Stacking**: Unlike boosting and bagging, stacking (Breiman, 1996b; Yin et al., 2021) uses base learners generated by different machine learners. The voting ensemble represents a simple stacking method in which a statistical mechanism is used to combine different types of ML models, such as decision trees and support vector machines. No matter how well the individual ML models perform on the training dataset, they all contribute equally to the merged model. One can consider the simple average of the predictions from the underlying ML models. However, using a weighted average ensemble makes the results more sensitive to the prediction errors generated by each contributing ML model. A further improvement can be made through stacked generalization, which applies a ML model to learn how to best combine the predictions derived from the base learners. It does this by first developing base models using the training dataset inputs. It then feeds the underlying ML models into a meta-learner,

which attempts to make a new model using the predictions of the weak learners based on new data.

Ensemble techniques are now widely applied in various engineering and geoscience disciplines. For example, three models of Bayesian, functional, and meta-ensemble were applied to Land Subsidence Susceptibility (LSS) mapping (Oh et al., 2019). The models split the dataset 50:50 between training and testing subsets with errors measured in relation to operating-characteristic curve. The ensemble including the logit boost model delivered the most accurate (91.44%) LSS maps.

Slope stability predictions were generated using a hybrid stacking ensemble method (Kardani et al., 2021). An artificial bee colony optimizer was applied to identify the optimal combination of base classifiers (ensemble level 0). These were then used to develop an effective meta-classifier (ensemble level 1), considering eleven separate tuned ML models. Finite element analysis was employed to create a synthetic database (150 records) for training the models. The trained models were then applied to predict 107 naturally occurring slope cases to test model performances. The hybrid-stacking ensemble model generated less errors than each ML model used in isolation.

Ensemble random forest, ensemble Gradient Boosted Regression Tree (GBRT) and multiLayer perceptron neural network were applied to model the spatial extent of landslides in Norway (Liu et al., 2021). Eleven landslide-influencing factors were considered related to geomorphologic, geologic, geo-environmental, and anthropogenic effects. 3,399 positive landslide records and 6,798 non-landslide were considered. Seventy percent of the data records in each of these two categories were selected for training the models. The remaining thirty percent of the data records were used to test the trained models. Slope angle was confirmed by the models to be the most important influencing factor. The ensemble GBRT model outperformed the other ensemble models, achieving a 95% probability of detecting landslides in that region.

Support vector machine, multilayer perceptron, random forest, Adaptive Boosting (AdaBoost), and extreme gradient boosting were used to develop synthetic geochemical logs for pre-salt reservoirs in Brazil (de Oliveira and de Carvalho Carneiro, 2021). Seven petrophysical logs: natural gamma-ray, gamma-ray spectroscopy, density, photoelectric factor, neutron porosity, nuclear magnetic resonance, and sonic formed the input variables. The chemical element concentrations for Al, Ca, Fe, Mg, Na, Si, S, and Ti were the prediction objectives. In addition to showing the best results, AdaBoost was found to be the most practical tree-ensemble algorithm to apply as it involved simpler pre-processing and control variable optimization.

3. Application Focus

The objective of the ensemble models developed is to predict subsurface fluid flow characteristics. This is of interest to both engineers and scientists with respect to, for example, the flow of fluids into producing wells penetrating oil and gas reservoirs, the seepage of wastewater through soil, and land subsidence as a consequence of groundwater and oil and gas extraction. With respect to oil and gas reservoirs, the main goal is to predict the performance of reservoirs at any future point in time and to optimize the petroleum fluid recovery under different operating conditions. Fluid flow in petroleum reservoirs is typically modelled using a set of non-linear PDEs. In general, these equations can be solved analytically (exact solutions) or numerically (approximate solutions).

A mixed GMsFEM method, as a numerical method has recently been proposed to solve Darcy's flow conditions (linear pressure gradient versus velocity) considering single-phase fluids in a porous medium characterized by heterogeneties in two dimensions (i.e., matrix composition and fracture distribution) (Chen et al., 2020). The model approximates reservoir pressure in multiscale space. It does so by applying several multiscale basis functions to a single coarse grid of the reservoir volume. The fluid velocity is directly estimated across a fine grid space.

The fluid flow conditions are defined as:

$$k^{-1}u + \nabla p = 0 \quad in \quad \Omega \tag{4}$$

$$\nabla . u = f \quad in \quad \Omega \tag{5}$$

Heterogeneous boundary conditions are included:

$$u.n = g \quad on \quad \partial\Omega \tag{6}$$

in which k is permeability, u is the Darcy velocity, p is the pressure, f is the source term, g is the normal to the Darcy velocity prevailing at the reservoir 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. Assuming $V = H(div, \Omega)$ and $W = L^2(\Omega)$, the mixed finite element spaces become:

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

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

$\left\{\Psi_j\right\}$ represents a set of multiscale base functions related to the coarse element. The multiscale space relating to pressure (p) can then be expressed as the linear extent of the local basis functions. This relationship is expressed as:

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

In that form, the mixed GMsFEM is configured to find $(u_H, p_H) \in (V_h, W_H)$

constrained by:

$$\int k^{-1} u_H . v_H - \int div\,(v_H)\, p_H = 0 \quad \forall\ v_H\ \in\ V_h^0 \tag{7}$$

$$\int div(u_H) w_H = \int f w_H \quad \forall\ w_H\ \in\ W_H \tag{8}$$

in which $u_H.n = g_H$ on $\partial\Omega$ is relating to the coarse edges at the boundaries, whereas g_H is the average of function g at those coarse edges.

It is necessary to establish a multiscale space, W_H, to approximate p. This is achieved by solving local cell conditions for each coarse grid element by applying Dirichlet's boundary conditions. If $T_i \in \tau^H$ represents the coarse grid elements relating to Ω, the purpose is to find $(u_j^{(i)}, p_j^{(i)}) \in (V_h,\ W_h)|_{T_i}$ by solving the following problem on T_i:

$$k^{-1} u_j^{(i)} + \nabla p_j^{(i)} = 0 \quad in\ T_i \tag{9}$$

$$div(u_j^{(i)}) = 0 \quad in\ T_i \tag{10}$$

where $(V_h,\ W_h)|_{T_i}$ is the restriction of $(V_h,\ W_h)$ on T_i.

The coarse grid boundary element represents the junction of fine grid edges, i.e., $\partial T_i = \bigcup_{j=1}^{J_i} e_j$ in which J_i is the total number of fine grid edges at boundary T_i. $\delta_j^{(i)}$ represents a piecewise constant related to ∂T_i and the fine grid and = 1 for e_j and = 0 for the remaining fine grid edges.

Therefore, the boundary condition on the boundary of T_i is taken as the Dirichlet boundary condition:

$$p_j^{(i)} = \delta_j^{(i)} \quad on\ \partial T_i \tag{11}$$

By combining the local problem solutions a snapshot of spatial conditions is derived. Assuming $\Psi_j^{i,snap} := p_j^{(i)}$ defines the snapshot fields, then the snapshot space can be expressed:

$$W_{snap} = span\left\{\Psi_j^{i,snap} : 1 \le j \le\ J_i\ ,\ 1 \le i \le\ N_t\ \right\} \tag{12}$$

In the case of using the single-index notation:

$$W_{snap} = span\left\{\Psi_i^{snap} : 1 \le i \le\ M_{snap}\right\} \tag{13}$$

where $M_{snap} = \sum_{i=1}^{N_t} J_i$ represents the total number of snapshot fields.

The snapshot space can then be further reduced by solving local grid problems. The local problem solutions are referred to as the offline space. The snapshot space corresponding to T_i becomes:

$$W_{snap}^{(i)} = span\left\{\Psi_j^{i,snap} : 1 \le j \le\ J_i\right\}$$

In a local grid problem, the real number $\lambda \ge 0$ and the function $p \in W_{snap}^{(i)}$

need to be derived

$$a_i\,(p,w) = s_i\,(p,w) \quad \forall w \in W_{snap}^{(i)} \tag{14}$$

For each T_i:

$$a_i\,(p,w) = \sum_e k\,[p]\,[w] \quad and \quad s_i\,(p,w) = \int kpw \;\; in\; T_i \tag{15}$$

in which $[p]$ and $[w]$ are the jump of functions p and w, respectively. Also, e represents the fine edge interior of T_i.

The eigenvalues of Equation 14 are arranged in increasing order:

$$\lambda_1^{(i)} < \lambda_2^{(i)} < \cdots < \lambda_{J_i}^{(i)} \tag{16}$$

where $\lambda_k^{(i)}$ denotes the kth eigenvalue for T_i. The corresponding eigenvectors are $Z_k^{(i)} = (Z_{kj}^{(i)})_{j=1}^{J_i}$ with $Z_{kj}^{(i)}$ being the jth component of the vector $Z_k^{(i)}$. Initial l_i eigenfunctions are selected to represent the offline space. Offline basis functions are then defined as:

$$\Psi_k^{i,off} = \sum_{j=1}^{J_i} Z_{kj}^{(i)} \Psi_j^{i,snap} \quad k = 1,\ 2,\ \ldots,\ l_i.$$

Then, global offline space becomes:

$$W_{off} = span\left\{ \Psi_k^{i,off} : 1 \le k \le\ l_i\ ,\ 1 \le i \le\ N_t \right\}$$

Applying single-index notation, the global offline space can be defined as:

$$W_{off} = span\left\{\Psi_k^{off} : 1 \leq k \leq \ M_{off}\right\}$$

where $M_{off} = \sum_{i=1}^{N_t} l_i$ is the total number of offline basis functions.

Each Ψ_k^{off} can be expressed by a vector ψ_k^{off} which contains coefficients from Ψ_k^{off} relating to the fine grid basis functions. Thus:

$$R_{off} = \left[\psi_1^{off}, \ldots, \psi_{M_{off}}^{off}\right]$$

The offline space is mapped using these functions to the fine grid space. The mixed GMsFEM system (Equations 7 and 8) is expressed in matrix terms as:

$$M_{fine} U_H + B_{fine}^T R_{off} P_H = 0 \tag{17}$$

$$R_{off}^T B_{fine} U_H = R_{off}^T F_H \tag{18}$$

M_{fine} constitutes a symmetric, positive definite, and sparse matrix. U_H and P_H are the unknown fluid velocity and pressure vectors that describe grid spaces V_h and W_H, respectively. Execution of the mixed GMsFEM therefore requires two fine grid matrices to be constructed (M_{fine}, B_{fine}) accompanied by one offline matrix (R_{off}).

Fluid velocity can be solved directly from the fine grid matrix combination. Considering k as a diagonal tensor, M_{fine} is readily estimated with the diagonal matrix $\widehat{M_{fine}}$, applying the trapezoidal quadrature rule. The convergence rate of that easier-to-execute system is essentially the same as that using the unmodified matrix. M_{fine} can therefore be replaced by that diagonal matrix $\widehat{M_{fine}}$ without compromising prediction accuracy. As $\widehat{M_{fine}}$ is easier to invert, the system described by Equations 17 and 18 is solved as follows:

$$-R_{off}^T B_{fine} \widehat{M_{fine}}^{-1} B_{fine}^T R_{off} P_H = R_{off}^T F_H$$

Taking this approach, an original mixed formulation is expressed approximately by a positive-definite, sparse linear system. In that linear system, fewer pressure unknowns are involved for each coarse-grid element.

Generally, the number of PDEs requiring solutions to enable multiscale basis functions to be derived is dependent on the number of local cell and local eigenvalue problems involved. The local cell problem relating to the coarse grid relates to the original system definition but excludes the source function in Equation 5. A boundary condition (delta) relates to the coarse grid boundary; delta=1 for fine grid edges and delta=0 for coarse grid edges. Local cell problems are therefore determined by the fine grid edges impacting the coarse grid boundary. In the model configured for this study, the number of fine grid edges/coarse grid boundary is 12.

In this study, the Karhunen-Loeve expansion was used to parameterize the heterogeneous permeability field. This Gaussian random field generation method decomposes a random process into the eigenvalue and eigenfunction of its covariance kernel. The fine grid system adopted involves a uniform 30×30 mesh. On the other hand, a sparser, uniform 10×10 mesh was applied to represent the coarse grid network (Figure 1). This configuration consists of 1300 separate PDEs, made up of 1200 PDEs addressing the local cell problems (100 coarse grid mesh units by 12 fine grid edges) plus 100 local eigenvalue problems (one per each 100 coarse grid mesh units). The input to this model is comprised of a randomly-generated permeability field. For each permeability field, there are five basis functions, (numbered Basis 1 to 5). Basis 1 is a piecewise constant, with binary values of -1 and $+1$. Basis 1 is defined as part of the finite element method, it therefore requires no training for DL modeling. On the other hand, Basis 2 to 5 take values distributed across the range $(-1, +1)$, and therefore require training for DL modeling.

4. Stacking CNN Ensemble Model

A schematic of the proposed stacking CNN ensemble model is given in Figure 2. The data generation and pre-processing steps involved are described in Sub-section 4.1. The mechanism adopted for training the base learners is outlined in Sub-section 4.2. The procedure for combining the DL model results is presented in Sub-section 4.3.

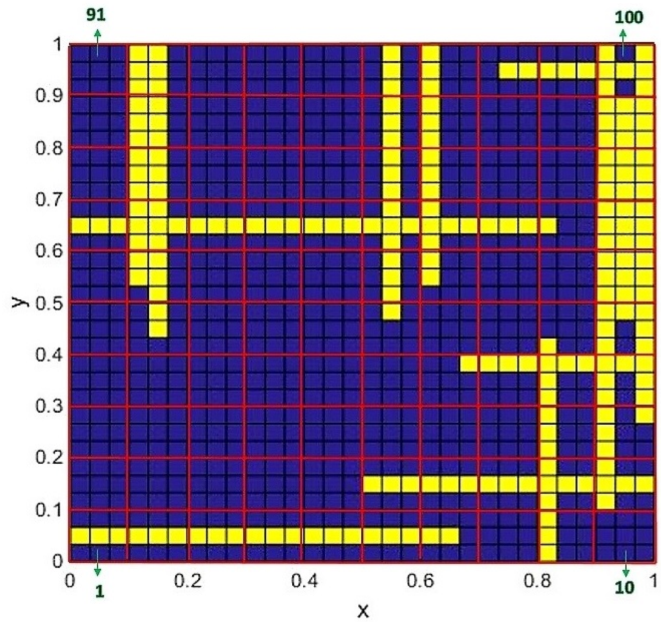


Figure 1: Schematic description of the permeability field of a simulated fractured porous reservoir formation. Matrix permeability is assumed to be 4 milliDarcies (mD). Fracture permeability is assumed to be 2000 mD. Fine grid squares represent the formation matrix (blue) in some cases and fractures (yellow) in other cases (selected randomly). The red lines define the coarse grid. Each coarse grid square contains of nine fine grid squares. There are fifteen fractures assigned to this porous medium.

4.1. Data preparation and pre-processing

The ranges of permeability values applied to the formation matrix were 1, 2, 3, 4, and 5 mD, and to the fractures were 500, 750, 1000, 1250, 1500, 1750, and 2000 mD. The number of fractures per 10×10 coarse grid was varied between 1 and 25. This ranges meant that 875 cases (5 matrix permeabilities by 7 fracture permeabilities by 25 different fracture densities) in total required evaluation by the DL model. The format defined for the permeability field was as a vector (900×1), subsequently adjusted to be

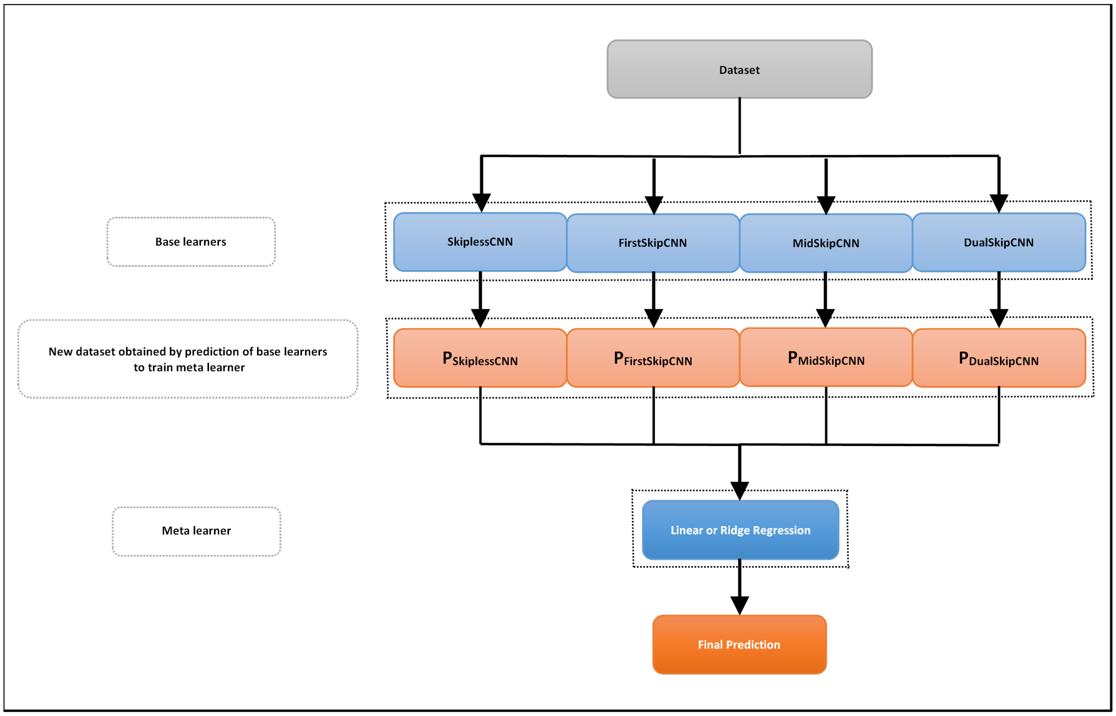


Figure 2: The workflow diagram of the stacking CNN ensemble model. Four base learners of SkiplessCNN, FirstSkipCNN, MidSkipCNN, and DualSkipCNN are developed using training/validation subsets. After being trained, they are used to make predictions on the validation data ($P_{\text{SkiplessCNN}}$, $P_{\text{FirstSkipCNN}}$, $P_{\text{MidSkipCNN}}$, and $P_{\text{DualSkipCNN}}$). Then, two meta models are separately developed using linear regression and ridge regression. Once the meta models are trained, they can be used to make predictions on the testing data (28,879 samples).

expressed as a 2D tensor (100×9), in which, coarse grid units=100 and each coarse grid contains 9 fine grids. Each row in the array therefore represents a coarse grid. Such a configuration enables the use of 2D CNN kernels. However, it was necessary to maintain the five basis functions as 900×1 vectors, so that they could be evaluated in the FC layers forming the final section of the CNN network.

Each of the 875 cases was evaluated 350 times as part of model training. Additionally, 40 validation runs, and 40 independent testing runs were executed for each case. Matlab code was used to run the mixed GMs-FEM models. These models generated 376,250 data records one for each 10×10 coarse grid configuration. 306,250 records were used for DL model training, 35,000 records for DL model validation, and 35,000 for independently testing the trained and validated models. The random generation of each permeability field (10×10 coarse grid) involves the possibility that some duplicate fields could be generated. Consequently, the generated dataset was filtered to remove any duplicate data records. This is necessary to remove the risk of introducing bias towards specific model configurations in the DL analysis. This data filtering step removed 1739 duplicate training data records , 579 duplicate validation data records, and 6121 duplicate testing data records were excluded. This pre-processing step reduced the training subset to 304,511 data records, the validation subset to 34,421 data records, and the independent testing subset to 28,879 data records.

4.2. Training the base learners

The CNN algorithm has, over recent years, become one of the most trusted DL models with respect to many application domains (Rao et al., 2017; Chen and He, 2018; Pratt et al., 2019; Hakim et al., 2022). The CNN was originally designed to solve problems with 2D arrays, particularly images, although it can also be applied to 1D arrays. It progressively and flexibly learns feature relationships, spatially in 2D models, by applying an optimizer of choice to various types of network layers. The primary CNN layer types are (i) convolutional, (ii) pooling, and (iii) FC, usually configured in that sequence. 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 (Yamashita et al., 2018; Elgendy, 2020).

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, for example, 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 (Yamashita et al., 2018; Elgendy, 2020).

When the output of the network is in the format of a vector, feature maps in the final convolutional or pooling layer 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 the 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 (Yamashita et al., 2018; Elgendy, 2020).

To develop the base learner for this study, the 304,511 training data records, together with the 34,421 validation data records were employed. Distinct CNN model configurations, involving various combinations of convolutional, pooling, FC, Batch Normalization (BN), regularization, and dropout filtering were tested separately for each basis function requiring training (Basis 2 to 5). A similar optimal CNN configuration for each of those four basis functions (Figure 3: identified as SkiplessCNN) was obtained. That initial CNN architecture consists of five convolutional layers, two FC layers but does not include any pooling layers. Convolutional layers 1 to 5 (CONV1 to CONV5) consist of 5, 10, 15, 20, and 25 kernels, respectively. To determine the size of a convolution output for an input with the size of $I_h(height) \times I_w(width)$ and a kernel with the size of $K_h \times K_w$, we can use Equation 19 if the padding is set to 'valid':

$$\begin{aligned} Output\ height = O_h = (I_h - K_h)/S_h\ + 1 \\ Output\ width = O_w = (I_w - K_w)/S_w\ + 1 \end{aligned} \tag{19}$$

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 'valid' 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.

Each convolutional layer is followed by a single BN layer of the same dimensions. Typically, neural network models converge more quickly when the input to each layer is normalized; hence the value of adding the BN layers. Each FC layer contains 2000 neurons. For a given neuron or kernel, the inputs are multiplied by weights and the resulting products summed together. A bias term is then applied to that sum. Such rigid computations mean that only linear transformations are performed on the layer inputs using the weights and biases to generate the layer outputs. Although this operation makes the neural network simpler, it is less powerful and unable to learn complex patterns in a dataset. This is where the activation function is beneficial. Mathematically, this can be represented as shown in Equation 20 where w_{i} represents the weight value, z_{i} is the input value, b is the bias, f refers to the activation function applied, and y is the dependent variable prediction output. The developed models in this study used the 'Rectified Linear Unit (ReLU)' activation function for the convolutional layers, 'sigmoid' activation function for the FC layers, and 'linear' activation function for the output.

$$y = f(\sum_{i=1}^{n}(w_{\text{i}}z_{\text{i}}) + b) \tag{20}$$

In order to better understand the standard architecture (i.e., Skipless-CNN) developed in this study, it is compared to structurally similar CNN architectures AlexNet (Krizhevsky et al., 2017) 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. 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. BN, 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.

The CNN training process seeks to find optimum values for weights and biases applied to kernels (convolutional layers) and neurons (FC layers). Such values generate the lowest collective errors for all data records evaluated between actual and predicted dependent variable values. The back-propagation algorithms are commonly applied to train many types of neural network. They calculate 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.

The different optimizers available all strive to achieve a minimum loss or cost value. Multi-layer perceptron neural networks focus on the feed-forward sequence through its layered structure on to which weights and biases are initialized. However, in training the backward pathway is used to modify the layer weights and biases in each iteration. In that way back propagation acts to improve a model's performance.

The CNN models in this study were constructed using Keras with TensorFlow as a backend on Python. The models were compiled using 'MSE' as the loss (objective) function. The learning rate is a key DL hyper-parameter. It states how quickly a model learns in each epoch that parameters are updated. When it is too small, the training process takes a long time. If too large, it results in sub-optimal CNN learning, locking into sets of weights and biases too quickly, which can lead to a less stable training process that tends to converge prematurely. Hence, setting the right value of the learning rate is crucial. Adaptive methods such as Adam can be used to automatically resolve this issue. Adam applies distinctive learning rates to each scalar variable. It progressively adapts those rates throughout the training iterations, with those adaptions being influenced by partial-derivative trends of rates applied to each variable in previous model iterations. Adam is gradient based in its calculations and benefits from a combination of its AdaGrad component to cope with sparse gradients and an RMSProp function in its application. It is suitable for DL applications to large datasets with many data records and/or multiple variables. The Adam learning rate can adjust at a finer scale as the optimum values are approached, although in some cases such fine tuning can result in overfitting. AMSGrad extends the performance of the Adam optimizer by converging in a more effective and smoother manner, avoiding step changes. By storing the highest values of second- momentum vectors generated in all previous model iterations, AMSGrad is able to normalize the moving average gradient in each iteration. To benefit from these advantages, the AMSGrad with the default values i.e., the initial global learning rate = 0.001, $beta_1 = 0.9$, $beta_2 = 0.999$, and $epsilon = 1e-7$ has been applied to the CNN models developed for this study. In addition, the models were trained with a batch size of 32 samples over 100 epochs.

In feed-forward neural networks with multiple layers, such as most DL models, back propagation works from the latter layers back through multiple layers to reach the initial layers. This extended sequence can result in the gradient being reduced rapidly, in very few model iterations, to a value close to zero. This generally unfavorable premature convergence is referred to the "vanishing gradient phenomenon". It is ineffective because it prematurely halts the training process before the early layers of the network have fully explored potentially more favorable values. A beneficial strategy that acts to reduce the risk of premature convergence is to involve "skip" connections between certain network layers, acting as short circuits for the back-propagation sequence (He et al., 2016). The introduction of skip connections enables the gradient to be directly back propagated to earlier layers of a CNN. Skip connections (shortcuts) are involved in three of the base learners used in this study (Figure 3). How and where in the CNN structure the shortcuts are located differs from scheme to scheme:

1. **FirstSkip**: a single skip connection from the first convolutional layer to the last one.
2. **MidSkip**: a single skip connection from the middle convolutional layer to the last layer.
3. **DualSkip**: two skip connections from the middle convolutional layer to the last and the second-to-last layers.

FirstSkip adds a single shortcut from the output of the first convolution layer to the last convolutional block. The input and output of this part have the same dimension of 98×7 because an identity type of shortcut is used. **MidSkip** is designed to discover how much a shortcut from the middle layer to the final layer can improve the performance of a model. Here, the input and output of this section with the shortcut have a dimension of 94×3. **DualSkip** was developed mainly to gain knowledge of the effect of involving the raw input features along with **MidSkip**. In all three cases, the main path and the shortcut meet each other before applying the activation function. For all three architectures, the FC layers remain unchanged.

Adding the skip connections increases the complexity of the base structure, in terms of the number of parameters. There are as many as 10,414,170 trainable and 150 non-trainable parameters for the base structure, without skip connections. By adding FirstSkip, the number of trainable and non-trainable parameters increases to 12,670,510 and decreases to 110, respectively. For MidSkip, there are 14,272,340 trainable and 130 non-trainable parameters. For DualSkip, the number of trainable and non-trainable parameters changes to 14,270,975 and 120, respectively.

4.3. Combination of the base learner outputs

A new training dataset for the meta learner was established by providing all data records from the validation subsets to each of the four sub-models and collecting the predictions they generated i.e., $P_{\text{SkiplessCNN}}$, $P_{\text{FirstSkipCNN}}$, $P_{\text{MidSkipCNN}}$, and $P_{\text{DualSkipCNN}}$. These resulted in four (referring to the number of base learners) arrays with the shape $[34421, 900]$, the first element referring to the number of validation data and the second to the output (basis function). Thus, a 3D array was developed with the shape $[34421, 4, 900]$, which was transformed into a $[34421, 3600]$ shaped array. This flattened input data, along with their output was used to train a meta learner.

As mentioned earlier, there are 875 cases that each need to be processed through the sequence of training, validation, and testing. Given that the input/output dimensions are so large, it did not make sense to apply boosting to focus on samples in the dataset that have been predicted incorrectly by the previous models in the sequence. Bagging is usually applied to relatively small datasets. Additionally, conducting bootstrap sampling incorporating all 875 cases was not feasible for addressing this large dataset. Therefore, stacking was selected to establish the ensemble model. A stacked generalization method was chosen, mainly because it is more flexible mathematically than voting or weighted average methods. To be more specific, the four base learners were combined into an ensemble model using linear and ridge regression, separately.

5. Evaluation

The prediction errors associated with each CNN model developed are assessed using two statistical error metrics: the R^2 and MSE. R^2 values can exist within a range -∞ and 1, with values closest to 1, representing the better prediction performance. MSE, by definition, has to be a non-negative value, and where values closer to zero represent the better performance. Table 1 presents the prediction error results for the base learners using the training and validation subsets. All the constructed Skipless-CNN models yield satisfactory results for the training samples; the best is for Basis 4 with an R^2 of 0.9156 and MSE of 0.0126. All three skip connection schemes enhance performance over the standard structure for all basis functions evaluated (i.e., Basis 2, 3, 4, and 5) by their training subsets. The defined schemes have the maximum effect on the base model for Basis 5 and the minimum effect for Basis 4. For example, R^2 increases from 0.8466 to 0.8844, 0.9026, and 0.8847 for Basis 5 by including FirstSkip, MidSkip, and DualSkip into the standard structure, respectively. MidSkip and DualSkip perform marginally better than FirstSkip.

Table 1: Prediction error analysis of the base learners applied to training and validation data subsets.

Subset	**Model**	R^2				MSE			
		Basis 2	Basis 3	Basis 4	Basis 5	Basis 2	Basis 3	Basis 4	Basis 5
Training	SkiplessCNN	0.8657	0.8952	0.9156	0.8466	0.0327	0.0220	0.0126	0.0100
	FirstSkipCNN	0.8908	0.9219	0.9247	0.8844	0.0266	0.0164	0.0112	0.0075
	MidSkipCNN	**0.9083**	0.9302	**0.9372**	**0.9026**	**0.0224**	0.0147	**0.0093**	**0.0063**
	DualSkipCNN	0.9002	**0.9327**	0.9283	0.8847	0.0243	**0.0141**	0.0107	0.0075
Validation	SkiplessCNN	0.7770	0.8237	0.8777	0.7816	0.0544	0.0371	0.0182	0.0142
	FirstSkipCNN	0.7814	0.8160	0.8798	0.7974	0.0529	0.0387	0.0181	0.0132
	MidSkipCNN	0.7867	0.8139	0.8802	**0.8160**	0.0519	0.0391	0.0179	**0.0120**
	DualSkipCNN	**0.7900**	**0.8434**	**0.8811**	0.8038	**0.0512**	**0.0329**	**0.0176**	0.0128

The prediction error performance of the SkiplessCNN models is acceptable for the validation data subsets, with an R^2 of 0.7770 to 0.8777, and MSE of 0.0142 to 0.0544. FirstSkip has a marginally positive effect on the validation subset with respect to Basis 2, 4, and 5 models. For instance, regarding Basis 5, the R^2 value increases from 0.7816 to 0.7974 and MSE decreases from 0.0142 to 0.0132. However, it has an adverse effect on the Basis 3 model. Specifically, R^2 decreases from 0.8237 to 0.8160. Compared to FirstSkip, MidSkip has a more positive effect on the Basis 2, 4, and 5 models. Furthermore, it has a negative effect on the Basis 3 model. DualSkip is beneficial in all cases related to validation samples, especially for Basis 3 and 5. For example, for Basis 5, R^2 increases from 0.7816 to 0.8038.

The results obtained for FirstSkip imply that transferring feature maps from earlier convolutional layers to final ones has a very positive effect on the training dataset. This architecture has a marginally positive impact on the validation subset for Basis 2, 4, and 5 models, but an adverse impact on the Basis 3 model. In other words, the corresponding skip connection tends to make the predictive model focus more on capturing the underlying trend of the training (seen) subset.

Compared to FirstSkip, flowing information from the middle convolutional layer to the last layer via the MidSkip skip connection has a more positive impact on all basis functions models of the training subset and the Basis 2, 4, and 5 models of the validation subset. This suggests that the feature maps of the middle convolution process contain important information.

Adding two simultaneous skip connections (DualSkip) favorably affects all basis functions with respect to the training and validation subsets. By comparing the architectures and results produced using MidSkip and DualSkip, the positive role of transferring raw feature maps is understandable. Therefore, enriching the last convolutional blocks with information hidden in the neighboring layers is more efficient than enriching them using earlier convolutional blocks near the input layer.

Figures 4 and 5 illustrate whether the combined models significantly influence the performance of the base learners based on the testing subset. The trend for the base learners over the testing subset is the same as their performance in the validation subset. The most likely reason is that the validation and testing subsets were drawn from the same data distribution. Both subsets also consisted of the same number of data records. It is apparent from Figures 4 and 5 that the ensemble models built by either linear or ridge regression perform substantially better on the testing subset than the individual models. In the case of applying linear regression, the R^2 and MSE lie in the range of 0.8456 to 0.9191, and 0.0092 to 0.0369, respectively. The results reveal that ridge regression works marginally better than linear regression with an R^2 ranging from 0.8539 to 0.9220, and ranging from MSE of 0.0090 to 0.0349.

Figure 6 presents an example of the pressure distributions for a representative permeability field, whose matrix permeability and fracture permeability are 1 and 1750 mD, respectively. The figure displays a very close match between the actual pressure distribution and the one obtained by applying the predictions derived from the developed ridge regression ensemble model in this study. The reservoir pressure distribution is important information used to determine the potential recovery factor of oil/gas reservoirs. These results confirm that this study's innovative approach to develop efficient and reliable DEL models to assist this specific aspect of fluid-flow simulation is successful and worthy of further development.

6. Discussion

The linear regression is one of the most straightforward approaches to predict output via a linear function of input features. In the context of ML, it refers to the most usual least square linear regression method that attempts to minimize the cost function. A drawback, however, is that it does not penalize high magnitude weights in its error function and it assumes independence between its features. These characteristics can lead in some cases to certain features being assigned very high weights during the training. The cost function for linear regression is typically expressed as:

$$cost\ function_{\text{linear}} = \sum_{i=1}^{m} (\mathbf{Y} - \hat{f}(\mathbf{X}))^2 \tag{21}$$

The ridge regression, as a modification of linear regression, involves a penalty (L2 regularization) to its error term, calculated as the sum of squared value of the weights. Giving a penalty in such a way results in a set of more evenly distributed weights. The cost function for ridge regression becomes:

$$cost\ function_{\text{ridge}} = \sum_{i=1}^{m} (\mathbf{Y} - \hat{f}(\mathbf{X}))^2 + \alpha \sum_{j=1}^{p} (w_{\text{j}})^2 \tag{22}$$

Here, α is included as a coefficient to penalize weights. It can take different values. A ridge model with $\alpha = 0$ is the same as a simple linear regression. As the α value nears infinity, an increasing number of coefficients of the model becomes zero until it is just a flat model with an intercept. In this study, we used the default value of ”one” for all cases.

It was expected that ridge regression would perform better than linear regression, and the results presented in Section 5 confirm this point.

Analysis of the case study, the proposed method, and the results lead to three recommendations for future research. First, the scope of the present study was restricted to 2D porous media with vertical and horizontal fractures. It is recommended to extend it to 3D porous media and incorporate inclined fractures. Moreover, it would be worthwhile to consider a wider range of permeability values for both the matrix and fractures. These adjustments would provide a more comprehensive representation of subsurface conditions. Second, the effectiveness of the stacked generalization method using linear regression and ridge regression was confirmed on the given task. However, there is still room for improvement in the testing subset. To further enhance performance, exploring more advanced ensemble techniques could be one direction to consider. Third, developing more diverse base learners could also provide valuable insights for further improving the performance of the stacking ensemble.

7. Conclusions

The substantial quantity of parameters involved in DL means that a large number of samples must be processed to provide effective results. The ReLU has emerged as a popular activation functions applied in DL, and AMSGrad, an enhanced version of the Adam optimizer, improves DL convergence. By using skip connection (shortcut) schemes during gradient-based training, such as back propagation, the vanishing gradient problem can be mitigated. These four DL performance features were applied to a case study to predict subsurface fluid flow and simplify a time-consuming component of oil/gas reservoir simulation. For this purpose, four distinct CNN learners - SkiplessCNN, FirstSkipCNN, MidSkipCNN, and DualSkipCNN - were developed for each multiscale basis function. Linear regression and ridge regression were then used separately to combine the four CNN into an ensemble model. The results confirm the effectiveness of the two tested ensemble architectures since they strike a more stabilized balance between bias and variance errors.

Acknowledgment

We appreciate the University of Liverpool for providing us with a powerful computer system to run our codes. Also, we would like to thank Ms. Roslyn Joy Irving and Dr. Trevor Mahy for providing helpful comments and suggestions.

Funding

This work is partially supported by Key Program Special Fund in XJTLU (KSF-E-50), XJTLU Research Development Funding (RDF-19-01-15), and Research Grants of School of Mathematics and Physics.

References

Breiman, L., 1996a. Bagging predictors. Machine learning 24, 123–140.

Breiman, L., 1996b. Stacked regressions. Machine learning 24, 49–64.

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

Chen, S., He, H., 2018. Stock prediction using convolutional neural network, in: IOP Conference series: materials science and engineering, IOP Publishing. p. 012026.

Elgendy, M., 2020. Deep learning for vision systems. Simon and Schuster.

Hakim, W.L., Rezaie, F., Nur, A.S., Panahi, M., Khosravi, K., Lee, C.W., Lee, S., 2022. Convolutional neural network (cnn) with metaheuristic optimization algorithms for landslide susceptibility mapping in icheon, south korea. Journal of environmental management 305, 114367.

He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778.

Ho, T.K., 1998. The random subspace method for constructing decision forests. IEEE transactions on pattern analysis and machine intelligence 20, 832–844.

Kanda, E., Epureanu, B.I., Adachi, T., Tsuruta, Y., Kikuchi, K., Kashihara, N., Abe, M., Masakane, I., Nitta, K., 2020. Application of explainable ensemble artificial intelligence model to categorization of hemodialysis-patient and treatment using nationwide-real-world data in japan. Plos one 15, e0233491.

Kardani, N., Zhou, A., Nazem, M., Shen, S.L., 2021. Improved prediction of slope stability using a hybrid stacking ensemble method based on finite element analysis and field data. Journal of Rock Mechanics and Geotechnical Engineering 13, 188–201.

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2017. Imagenet classification with deep convolutional neural networks. Communications of the ACM 60, 84–90.

Kumari, P., Toshniwal, D., 2021. Extreme gradient boosting and deep neural network based ensemble learning approach to forecast hourly solar irradiance. Journal of Cleaner Production 279, 123285.

Liu, Z., Gilbert, G., Cepeda, J.M., Lysdahl, A.O.K., Piciullo, L., Hefre, H., Lacasse, S., 2021. Modelling of shallow landslides with machine learning algorithms. Geoscience Frontiers 12, 385–393.

Nguyen, M., Logofătu, D., 2018. Applying tree ensemble to detect anomalies in real-world water composition dataset, in: International Conference on Intelligent Data Engineering and Automated Learning, Springer. pp. 429–438.

Oh, H.J., Syifa, M., Lee, C.W., Lee, S., 2019. Land subsidence susceptibility mapping using bayesian, functional, and meta-ensemble machine learning models. Applied Sciences 9, 1248.

de Oliveira, L.A.B., de Carvalho Carneiro, C., 2021. Synthetic geochemical well logs generation using ensemble machine learning techniques for the brazilian pre-salt reservoirs. Journal of Petroleum Science and Engineering 196, 108080.

Pratt, H., Williams, B., Broadbent, D., Harding, S., Coenen, F., Zheng, Y., 2019. Learning the features of diabetic retinopathy with convolutional neural networks. EUROPEAN JOURNAL OF OPHTHALMOLOGY 29, NP15–NP16.

Rao, Y., He, L., Zhu, J., 2017. A residual convolutional neural network for pan-shaprening, in: 2017 International Workshop on Remote Sensing with Intelligent Processing (RSIP), IEEE. pp. 1–4.

Sahin, E.K., Demir, S., 2023. Greedy-automl: A novel greedy-based stacking ensemble learning framework for assessing soil liquefaction potential. Engineering Applications of Artificial Intelligence 119, 105732.

Schapire, R.E., 1990. The strength of weak learnability. Machine learning 5, 197–227.

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

Tüysüzoğlu, G., Birant, D., 2020. Enhanced bagging (ebagging): A novel approach for ensemble learning. International Arab Journal of Information Technology 17.

Verma, R., Chandra, S., 2023. Repute: A soft voting ensemble learning framework for reputation-based attack detection in fog-iot milieu. Engineering Applications of Artificial Intelligence 118, 105670.

Wang, G., Hao, J., Ma, J., Jiang, H., 2011. A comparative assessment of ensemble learning for credit scoring. Expert systems with applications 38, 223–230.

Wang, G., Sun, J., Ma, J., Xu, K., Gu, J., 2014. Sentiment classification: The contribution of ensemble learning. Decision support systems 57, 77–93.

Yamashita, R., Nishio, M., Do, R.K.G., Togashi, K., 2018. Convolutional neural networks: an overview and application in radiology. Insights into imaging 9, 611–629.

Yin, X., Liu, Q., Pan, Y., Huang, X., Wu, J., Wang, X., 2021. Strength of stacking technique of ensemble learning in rockburst prediction with imbalanced data: Comparison of eight single and ensemble models. Natural Resources Research 30, 1795–1815.

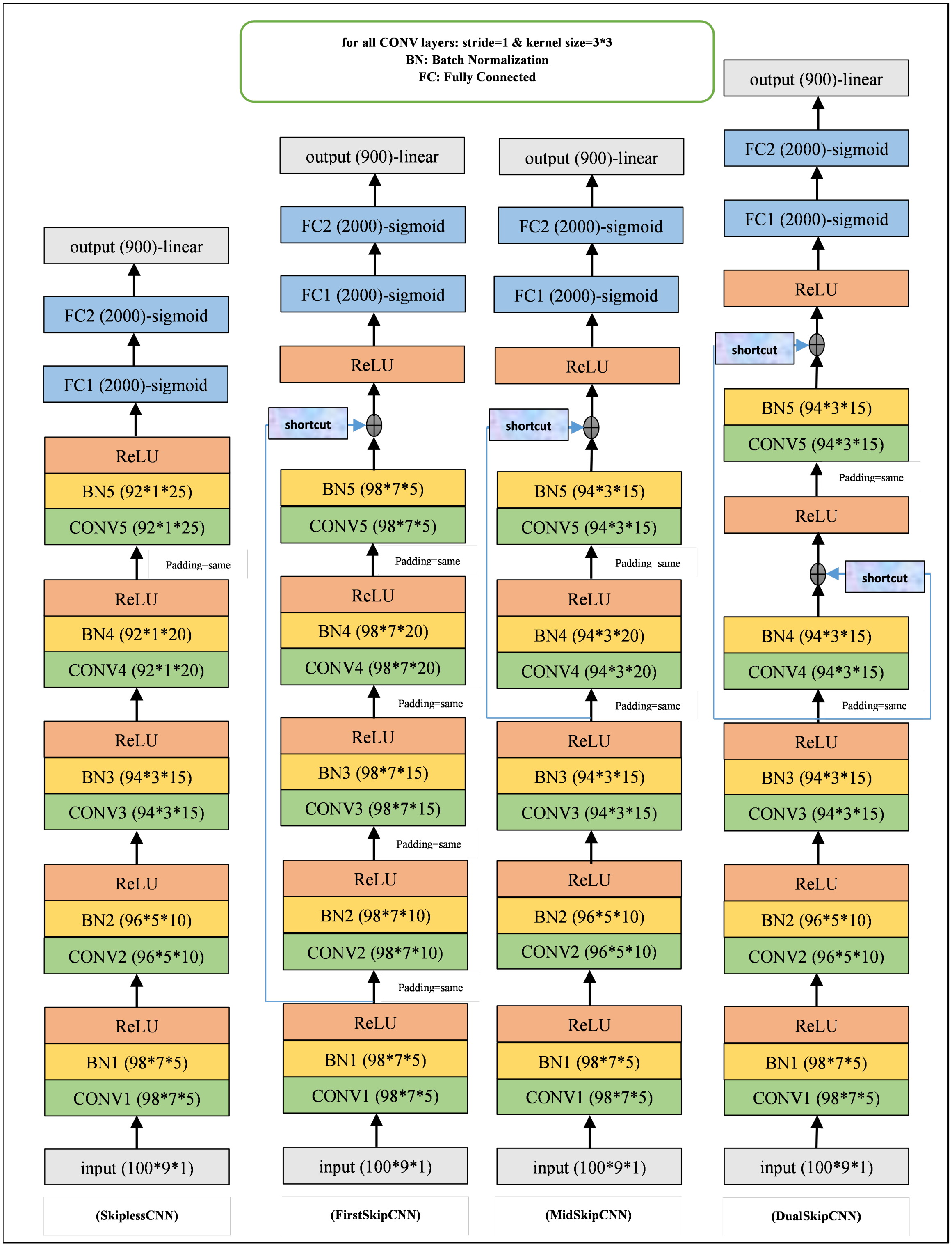
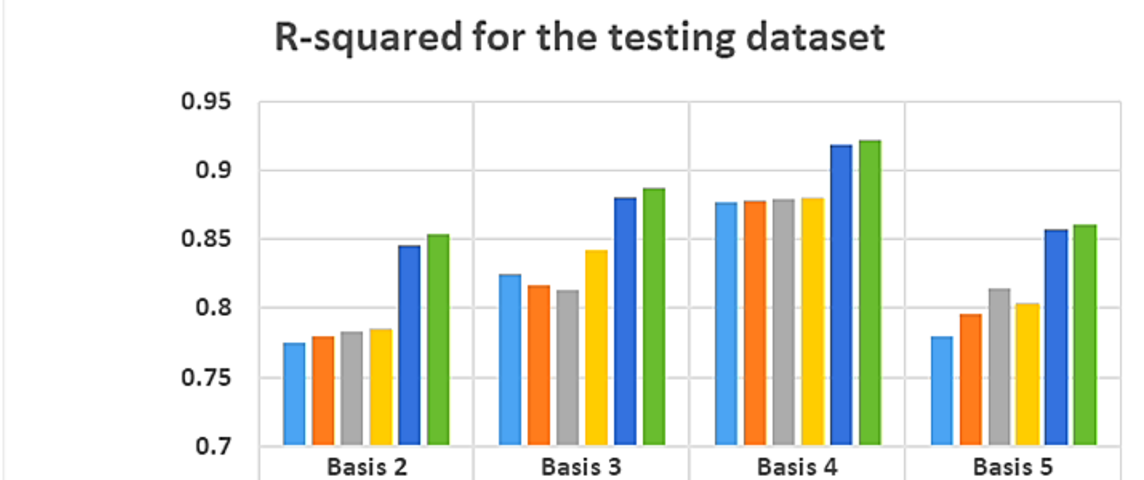
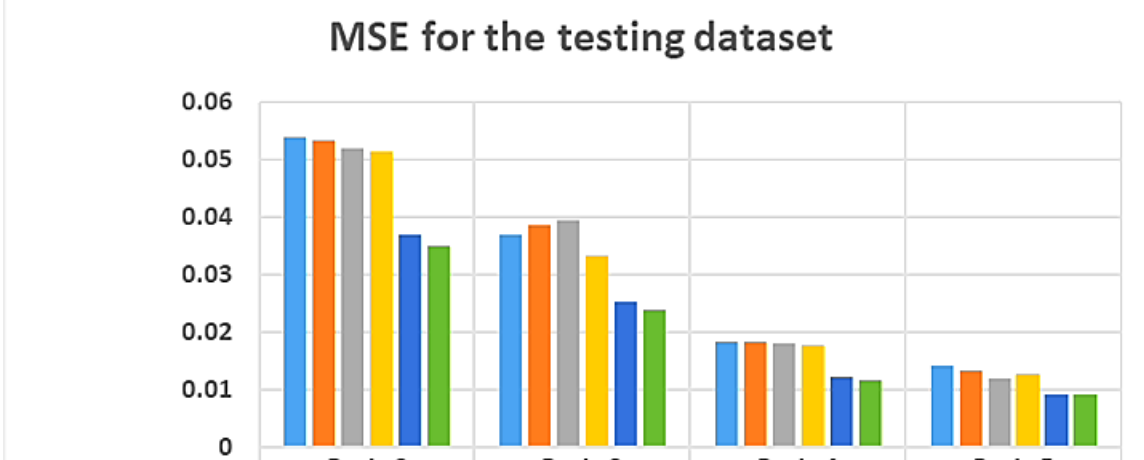


Figure 3: Structure of the CNN base learners configured in this study.



	Basis 2	Basis 3	Basis 4	Basis 5
SkiplessCNN	0.775	0.8247	0.8769	0.78
FirstSkipCNN	0.7801	0.8164	0.8778	0.7954
MidSkipCNN	0.7829	0.8127	0.8791	0.8141
DualSkipCNN	0.7857	0.8422	0.8809	0.8044
Linear	0.8456	0.8807	0.9191	0.8569
Ridge	0.8539	0.887	0.922	0.8605

Figure 4: Prediction error analysis of the DEL-based and CNN models applied to the testing data subset, expressed in terms of R^2.



	Basis 2	Basis 3	Basis 4	Basis 5
SkiplessCNN	0.0538	0.0369	0.0184	0.0142
FirstSkipCNN	0.0534	0.0387	0.0182	0.0132
MidSkipCNN	0.0519	0.0395	0.018	0.012
DualSkipCNN	0.0513	0.0332	0.0177	0.0126
Linear	0.0369	0.0251	0.0121	0.0092
Ridge	0.0349	0.0238	0.0116	0.009

Figure 5: Prediction error analysis of the DEL-based and CNN models applied to the testing data subset, expressed in terms of MSE.

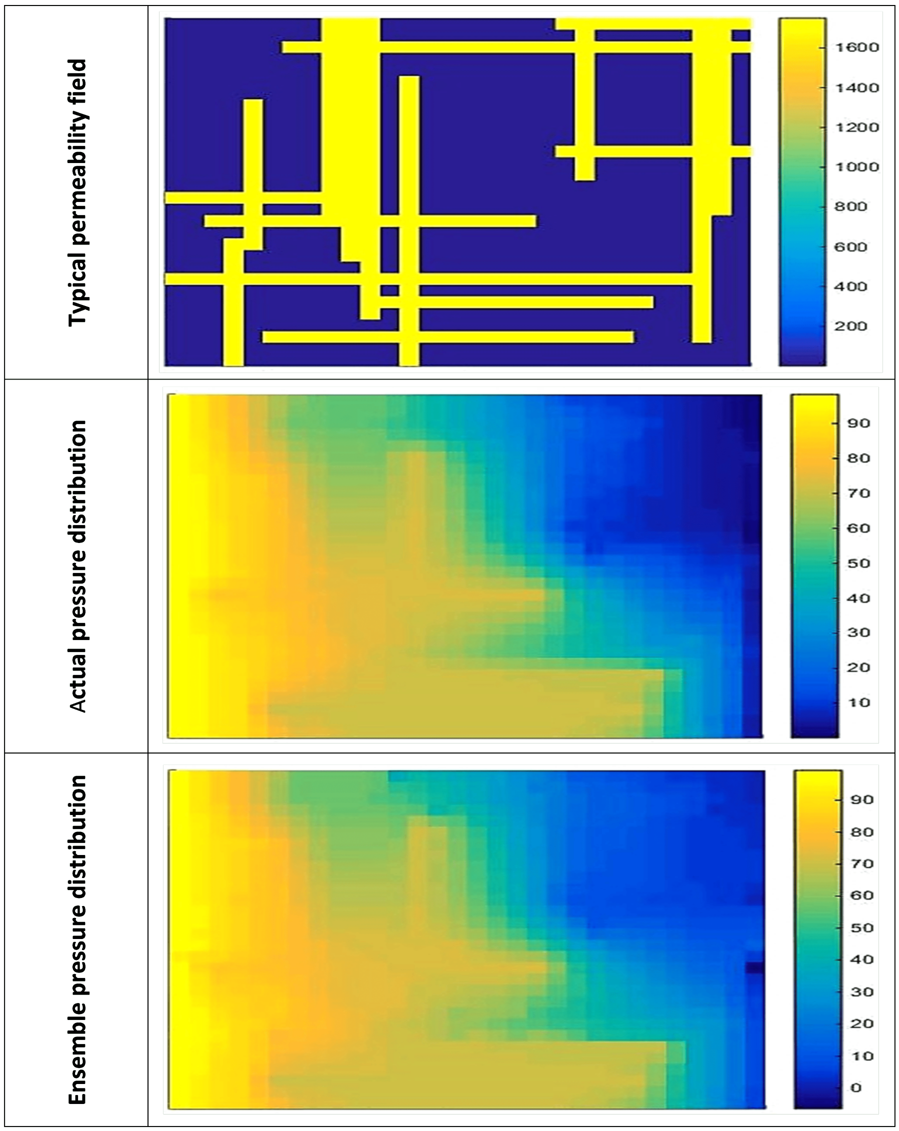


Figure 6: A comparison between the actual pressure distribution and the one obtained by ridge regression ensemble model for a representative permeability field.