Dimensionality reduction through convolutional autoencoders for fracture patterns prediction

Krushna Shinde, Vincent Itier, José Mennesson, Dmytro Vasiukov, Modesar Shakoor

Dimensionality reduction through convolutional autoencoders for fracture patterns prediction

Krushna Shinde[1], Vincent Itier[2,3], José Mennesson[2,3], Dmytro Vasiukov[1], and Modesar Shakoor* [1]

[1]IMT Nord Europe, Institut Mines-Télécom, Univ. Lille, Centre for Materials and Processes, F-59000 Lille, France

[2]IMT Nord Europe, Institut Mines-Télécom, Centre for Digital Systems, F-59000 Lille, France

[3]Univ. Lille, CNRS, Centrale Lille, Institut Mines-Télécom, UMR 9189 CRIStAL, F-59000 Lille, France

October 26, 2022

Abstract

Dimensionality reduction methods construct a low dimensional or reduced space. Complex structural mechanics problems can be approximated on a low dimensional space with a Reduced-Order Model or surrogate model. For instance, Principal Components Analysis can compute a reduced basis from a database of full-order snapshots. Principal Components Analysis, however, does not reconstruct low dimensional spaces efficiently for highly nonlinear problems, mainly because it is a linear dimensionality reduction method. In this paper, an original approach based on an autoencoder neural network is developed to construct a nonlinear Reduced-Order Model for a highly nonlinear brittle fracture problem. We show on a set of

*Corresponding author: modesar.shakoor@imt-nord-europe.fr

simulations how the autoencoder can be efficient for dimensionality reduction or compression of highly nonlinear data. A complete deep learning framework is proposed to predict crack propagation patterns directly from the loading conditions. Finally, we validate our nonlinear Reduced-Order Model with data sets generated for two problems using proportional and non-proportional loading conditions. In the first problem, the crack always initiates at the same location but propagates in various directions, which may even vary during the simulation if loading conditions are non-proportional. In the second problem, the crack may additionally initiate at various locations. Results for the two problems evaluate the capabilities of the proposed approach.

Keyword: Dimensionality reduction, Deep learning, Autoencoder, Fracture mechanics, Crack propagation

1 Introduction

Reduced-Order Modeling (ROM) methods tackle the issue of computational cost in mechanical engineering when multiple numerical simulations should be performed under various loading conditions or model parameters. They consist in using a conventional High-Fidelity (high dimensional) Model (HFM) to generate data (snapshots) and then approximating this HFM in a low dimensional space. Then, the low-dimensional model assists in making predictions for unseen model parameters.
Principal Component Analysis (PCA) is a well-known dimensionality reduction method for finding a low dimensional space [1, 2]. This method is used in projection-based ROMs but it does not reconstruct low dimensional spaces efficiently for highly nonlinear problems [3, 4], mainly because it is a linear compression method.

More recently, deep learning Artificial Neural Networks (ANN) have found their applications for dimensionality reduction of nonlinear high dimensional data and for building data-driven ROMs to approximate HFMs [5, 6]. Among these neural networks, autoencoders are particularly interesting as they can reduce highly nonlinear data dimensionality [7, 8]. An autoencoder is a particular type of feed-forward neural network that maps original input data to a low dimensional space and reconstructs the data from the low

dimensional space to its original input dimension.

A simple autoencoder with one hidden layer and linear activation functions can produce basis functions that span the same space as PCA [9]. With an autoencoder, however, the number of hidden layers can be increased and the activation functions can be nonlinear within layers. This makes autoencoders suitable for nonlinear dimensionality reduction. Data-driven ROMs can be built based on autoencoders by coupling the trained autoencoder with an additional feed-forward neural network. In Ref. [6], time-dependent problems were solved by compressing arrays containing the solutions for all simulated time instants using an autoencoder, and then predicting the solutions for new unseen parameters using a separate ANN. A structural dynamic problem with varying material properties was solved using this approach.

In the field of fluid mechanics, autoencoders have recently been considered [10, 11]. These works have mainly focused on nonlinear dimensionality reduction of complex fluid flows. Data-driven ROMs based on autoencoders have also been developed to predict fluid flow states in transient simulations [12]. The autoencoder is trained in such a way that it takes as input the concatenated solutions for n time steps (sequence of first n fluid flow simulation states) and gives as output the solution at time step $n+1$ (future predicted time step). To predict the fluid flow state at time step $n+2$ it is required to append the result of time step $n+1$ to the first n fluid flow simulation states. In this approach, the ROM does not mimic the iterative procedure of the HFM. Fluid flow solvers, indeed, are typically based on the Navier-Stokes equations and can compute the solution at time step $n+1$ only from the solutions computed for a few previous time steps (n, $n-1$, etc.) depending on the order of the time discretization scheme.

A similar approach relying on a nonlinear POD-autoencoder based model has also been developed for convection-dominated flows [13]. This data-driven ROM does not mimic the single-step prediction procedure of the HFM either. The same remark applies to Ref. [6], where the solutions at all instants were predicted simultaneously instead of iteratively. This feature is however important in order to replace HFMs by deep learning models in practical engineering applications. In addition, these recent works on autoencoders in the field of mechanics have mostly focused on fluid mechanics, and highly nonlinear mechanical problems involving history variables such as fracture mechanics problems are still to

be investigated.

Crack initiation and propagation problems, which are fundamental in the field of fracture mechanics [14], are particularly challenging for dimensionality reduction methods because they are path-dependent phenomena. In other words, for two different loading paths leading to the same final applied load, cracks might initiate and propagate completely differently for the same initial structure.
The standard methods for solving crack initiation and propagation problems in brittle materials are the extended finite element method [15], phase-field models [16], continuum damage models [17] and cohesive zone models [18]. These methods are computationally expensive (they can be classified as HFMs) to study fracture patterns for different loading conditions.

In this work, we explore the use of an autoencoder to reduce the dimensionality of data generated by solving HFMs of two-dimensional (plane strain) brittle fracture mechanics problems. The data sets are generated by applying a large number of different random loading paths to the same structure. Each loading path is discretized over several time steps, which are solved using an HFM based on a phase-field model and the Finite Element (FE) method. This HFM computes for each time step of each loading path three variables of interest: the displacements field in the x-direction and the y-direction, and the phase-field variable. The phase-field variable takes values between 0 and 1, with 1 indicating the local presence of a crack. We restrict ourselves to rectangular domains and structured finite element meshes so that for each time step of each loading path, the data can easily be converted to three-channels pictures representing the three variables.
This format allows us to use a convolutional autoencoder for dimensionality reduction [19]. Convolutional layers in the autoencoder can efficiently capture nonlinear patterns in the data, which helps to find an optimal nonlinear low dimensional space. We also build a non-intrusive deep learning-based prediction model (predictor) from the low dimensional space generated by the autoencoder to predict fracture patterns for unseen loading paths. This is summarized in Figure 1.

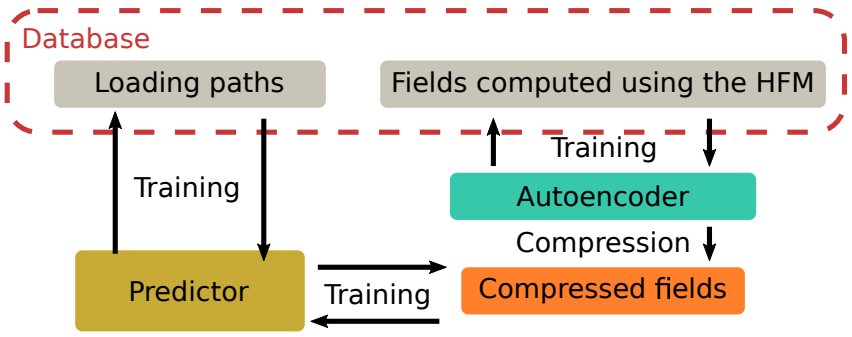


Figure 1: Deep learning-based framework proposed in this paper

This paper is organized as follows. Section 2 presents the autoencoder in details for dimensionality reduction. Section 3 describes the framework of nonlinear data-driven ROM for a highly nonlinear brittle fracture problem. In Section 4, we demonstrate the use of an autoencoder for dimensionality reduction of the data set (generated for the fracture mechanics problem) and discuss the compression accuracy. Finally, we show how the proposed nonlinear data-driven ROM can be functional in predicting fracture patterns from unseen loading paths.

2 Dimensionality reduction using an autoencoder

In this section, autoencoders are presented, and in particular convolutional autoencoders for nonlinear dimensionality reduction. A quantitative criterion is finally proposed to assess the accuracy of dimensionality reduction methods.

2.1 Autoencoders

Let $X_1, X_2, \ldots, X_M \in \mathbb{R}^N$ be M data points and X be a data matrix such that $X_1, X_2, \ldots, X_M$ are the rows of this matrix. Let us assume that it is normalized, *i.e.*, column means are equal to zero and column standard deviations are equal to one.

The simplest autoencoder we can design is a feed-forward neural network which encodes its input $X_m \in \mathbb{R}^N$ (where N is the number of input neurons) into a hidden

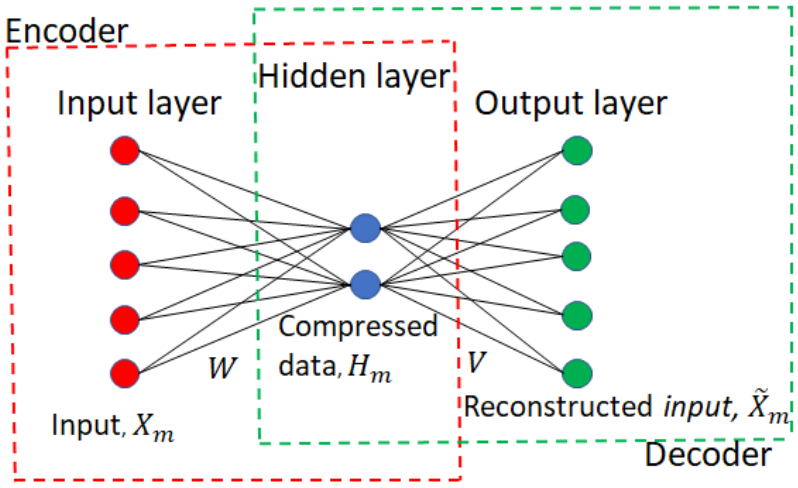


Figure 2: Example of a simple autoencoder with a single hidden layer

representation H_m (see Figure 2) as

$$H_m = g(X_m W + b), n = 1 \dots N, \tag{1}$$

where $H_m \in \mathbb{R}^K$. Here K is the number of hidden layer neurons, which in the special case of an autoencoder is called encoding dimension, $W \in \mathbb{R}^{N \times K}$ is a weight matrix, g is a so-called activation function that has to be chosen appropriately, and $b \in \mathbb{R}^K$ is a bias vector. The hidden layer represents the compressed input data (see Figure 2), often called encoded, latent, or low dimensional space. The autoencoder decodes a reconstruction $\tilde{X}_m$ from the hidden layer representation as

$$\tilde{X}_m = f(H_m V + c), \tag{2}$$

where $V \in \mathbb{R}^{K \times N}$ is another weight matrix, f is another activation function, and $c \in \mathbb{R}^N$ is another bias vector. The autoencoder model is trained to minimize a certain loss function such as the mean squared error function:

$$\underset{W,V,b,c}{\text{minimize}} \frac{1}{M} \sum_{m=1}^{M} \sum_{n=1}^{N} (\tilde{X}_{mn} - X_{mn})^2, \tag{3}$$

which ensures that $\tilde{X}_m$ is close to X_m, $m = 1 \dots M$. We can train the autoencoder just like a regular feed-forward network using back propagation. As shown in Figure 2, an autoencoder neural network can be split into two parts: the encoder, which maps the input $X_m \in \mathbb{R}^N$ to its reduced representation $H_m \in \mathbb{R}^K, K \ll N$, and a decoder, which maps the reduced state to the reconstructed output $\tilde{X}_m$.

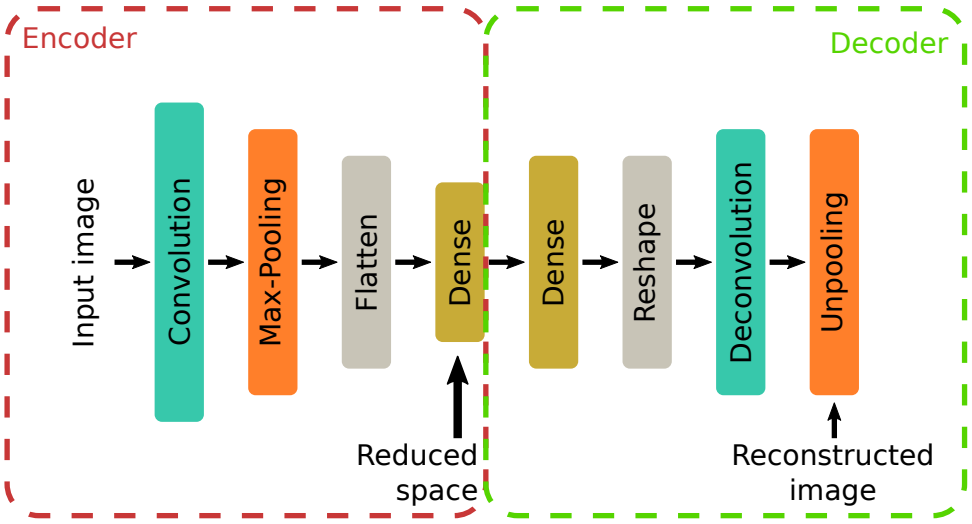


Figure 3: Example of CNN autoencoder

2.2 Deep convolutional autoencoders

In the neural network shown in Figure 2, we can see that each neuron of layer L is connected to all neurons of layer $L-1$. This kind of layer is called fully-connected or dense layer. A Convolutional Neural Network (CNN) contains layers where the connections are more efficient for input data that is structured as in time series, two-dimensional (2D) pictures, videos, etc. Fully-connected or dense layer connections can create problems in memory and computations if a neural network is large, as compared to convolutional layers.

In convolutional layers, each neuron in layer L depends on a subset of the neurons in the previous layer $L-1$. Kernels or filters are helpful for this purpose. For each neuron in layer L, the same convolutional filter is applied to the corresponding subset of neurons in layer $L-1$. Hence CNNs benefit in terms of sparse interaction between layers and parameter sharing between the neurons of hidden layers, which makes them suitable for high-dimensional data. The use of convolutional layers makes sense when we want to extract the features of spatially distributed data, mainly in the form of pictures.

In this work, we use a CNN autoencoder of the type shown in Figure 3 for dimensionality reduction. Convolutional layers in the autoencoder can efficiently capture nonlinear patterns in the data, which helps to find the nonlinear low dimensional space. Both the

encoder and decoder make use of convolutional and dense layers.

The encoder part of the CNN autoencoder takes images as input tensors with a shape: (input height) × (input width) × (input channels). Then the convolutional layer performs the operation of convolution using filters ((feature map height) × (feature map width) × (feature map channels)) along with an activation function to capture the local features of the input image. The commonly used activation functions in the convolutional layers are sigmoid, tanh, relu, and elu. The convolutional layers are followed by pooling layers (max-pooling) to reduce the features' spatial size, decreasing the required computations and weights. We then flatten the output of the max-pooling layer and use one dense layer to compress the input image to the encoding dimension.

The decoder part takes compressed input (output of the encoder) with the shape of the encoding dimension (dimension of latent space). And after that, one dense layer decompresses the encoded output to a larger size, which is then reshaped into image form. The image is passed through deconvolutional layers to reconstruct the original image back to its input shape. The deconvolution operation can be performed by a convolutional layer followed by an unpooling layer or alternatively by a single transposed convolutional layer (Conv2DTranspose).

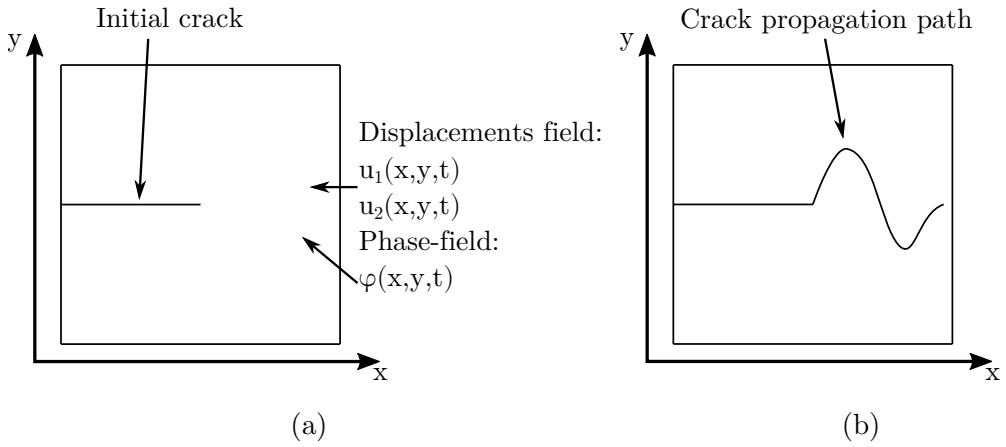
In this way, we can use the encoder and decoder part of the CNN autoencoder for image compression and decompression.

2.3 Compression accuracy

Autoencoders can reduce the dimensionality of the data and then reconstruct it with some compression error. In the present work, we wish to investigate this method's performance in terms of dimensionality reduction and reconstruction of the nonlinear data (both on training and testing data). To do this, we consider three criteria.

The first one concerns the dimension of the reduced space required by the method to produce a low dimensional or encoded space. Here, the dimension of the reduced space means the dimension of the latent or encoded space.

The second criterion concerns the reconstruction error on test data, which is new unseen data (data that is not present in the training data set used to train the autoencoder).



(a) (b)

Figure 4: An example of brittle fracture mechanics problem: the single edge notch specimen

We use the Root Mean Square Error (RMSE) for measuring this reconstruction error:

$$RMSE = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(X_{mn} - \tilde{X}_{mn})^2} \tag{4}$$

where $X_m \in \mathbb{R}^N$ is the normalized sample and $\tilde{X}_m \in \mathbb{R}^N$ is the normalized reconstructed sample. Due to the normalization, this RMSE is adimensional.

Lastly, we report the computation time required to train the models and reconstruct unseen test data.

3 Prediction of fracture patterns combining an autoencoder and an artificial neural network

This section describes the development of a so-called predictor based on deep learning for a highly nonlinear brittle fracture problem. We consider crack initiation and propagation in a 2D plate $\Omega \subset \mathbb{R}^2$ containing a heterogeneity to influence the crack path. For instance, the plate may be notched as shown in Figure 4a and the crack may propagate differently depending on the loading conditions as shown in Figure 4b.

To vary the loading conditions easily we impose the displacements u at all boundaries:

$$\begin{bmatrix} u_1(x,y,t) \\ u_2(x,y,t) \end{bmatrix} = G(t) \begin{bmatrix} x \\ y \end{bmatrix}, \ (x,y) \in \partial\Omega \tag{5}$$

where

$$G(t) = \begin{bmatrix} G_{xx}(t) & G_{xy}(t) \\ G_{yx}(t) & G_{yy}(t) \end{bmatrix} \tag{6}$$

with $t \in [0,T]$, $T > 0$. In the particular case where G has a linear relationship with respect to time, the loading path is said to be proportional. It is coined non-proportional in all other cases.

In the HFM, the loading is applied incrementally by solving balance equations at each discrete time step t_n with the displacements imposed at all boundaries. In addition, to model crack initiation and propagation, the so-called phase-field cohesive zone model is used [20]. There is hence an additional equation to solve at each time step, *i.e.*, the phase-field evolution equation. This damage model, moreover, introduces a history-dependence as the evolution of the phase-field variable at time step t_{n+1} depends on the fields at t_n. As a result, for each time step, we can compute displacements fields $u_1(x,y,t_{n+1})$, $u_2(x,y,t_{n+1})$, and phase-field variable $\varphi(x,y,t_{n+1})$ over the domain Ω. The HFM relies on the FE method. A brief overview is presented in Subsection 3.1.

The HFM is used to generate data for a set of randomly generated loading paths. For each path and for each time step t_n, the data set contains the loading matrix $G(t_n)$ as well as the fields $u_1(x,y,t_n)$, $u_2(x,y,t_n)$ and $\varphi(x,y,t_n)$ which are averaged over each element of the FE mesh. As we use only Cartesian grids, this element-wise averaging results in a three-channel 2D picture for each time step. The data set is presented in details in Subsection 3.2.

Our objective is to use the HFM only to generate a data set for a limited number of loading paths, and then rely on deep learning to make predictions for a greater number of unseen loading paths. This approach will only be relevant if the deep learning approach is less computationally expensive than the HFM when making predictions, and yields results

that compare well to those computed using the HFM. As stated earlier, our strategy consists in first reducing the dimensionality of the problem, in order to have only some unknowns to predict instead of three-channel 2D pictures. The history-dependence due to the damage model introduces an additional complexity that can also be treated using dimensionality reduction. This nonlinear data-driven ROM based on deep learning is presented in Subsection 3.3.

3.1 High-fidelity model

In statics, a mechanical model is typically composed of:

- balance equations,
- boundary conditions,
- and a constitutive model.

Boundary conditions have already been provided in Equation (5). Assuming small strains and no body force, the balance equations are given at each instant $t \in [0, T]$ by

$$\nabla.\sigma(u(x, y, t), \varphi(x, y, t)) = 0, (x, y) \in \Omega, \tag{7}$$

where σ is the Cauchy stress tensor. Note that (x, y) are always the coordinates of each point of the plate in the initial configuration, and $(x, y) + u(x, y, t)$ are its coordinates in the deformed configuration.

The constitutive model provides the relation between u and σ. It is given by Hooke's law for isotropic linear elastic material behavior which is multiplied by a degradation term originating from the phase-field model:

$$\sigma(u(x, y, t), \varphi(x, y, t)) = g(\varphi(x, y, t)) \left(2\mu(x, y)\varepsilon(u(x, y, t)) + \lambda(x, y)\mathrm{tr}(\varepsilon(u(x, y, t))I_2\right), \tag{8}$$

where g is the degradation function, μ and λ are the Lamé parameters of the material, tr is the trace operator, I_2 is the identity matrix and

$$\varepsilon(u(x, y, t)) = \frac{1}{2}\left(\nabla u(x, y, t) + \nabla^T u(x, y, t)\right), \tag{9}$$

is the small strain tensor. The idea of introducing a degradation function to model the presence of a crack can be found in both continuum damage models and phase-field models [16, 20–22]. Continuum damage models are well-known to raise a mesh-dependence issue which can be solved by introducing some regularization. Phase-field damage models borrow this idea and introduce a non-local regularization directly in the phase-field evolution law, so that regularization is built-in for phase-field models [22].
For both continuum damage models and phase-field models, the regularization removes the mesh dependence but introduces an additional parameter, the regularization length, which is often considered as a material parameter. In the so-called cohesive phase-field model used in the present work, the regularization length plays the role of a numerical parameter, and the model can be shown to converge to a unique solution for a sufficiently small regularization length [16, 20]. This model is designed to mimic a cohesive zone model with a bilinear softening law. The degradation function is

$$g(\varphi(x,y,t)) = \frac{(1-\varphi(x,y,t))^2}{(1-\varphi(x,y,t))^2 + \frac{4}{\pi}\frac{l_{ch}(x,y)}{l_c}\varphi(x,y,t)\left(1-\frac{1}{2}\varphi(x,y,t)\right)}, \tag{10}$$

with $l_{ch} = l_{ch}(x,y) = \dfrac{E(x,y)G_c(x,y)}{(\sigma_c(x,y))^2}$, E, G_c and σ_c being respectively the material's characteristic length, Young's modulus, fracture toughness and fracture strength. The characteristic length is a material parameter. The regularization length l_c, however, is a purely numerical parameter that should be chosen small enough compared to l_{ch}, and the mesh size should be small enough compared to l_c. This restrictive requirement on phase-field models is well-known to raise challenging issues regarding their computational cost.
The phase-field evolution equation is given by

$$\begin{aligned} &\frac{2G_c(x,y)}{\pi l_c}(1-\varphi(x,y,t)) \\ &+g'(\varphi(x,y,t))\mathcal{H}(\sigma(u(x,y,t))) \\ &-\nabla.\left(2\frac{G_c l_c}{\pi}\nabla\varphi(x,y,t)\right) \quad = 0, (x,y)\in\Omega, \end{aligned} \tag{11}$$

where $\mathcal{H}(\sigma(u(x,y,t))) = \dfrac{1}{2E(x,y)}\left(\max\left(\sigma_c, \max\limits_{s\leq t}\sigma_1(u(x,y,s))\right)\right)^2$ is the history variable associated to a Rankine failure criterion, σ_1 being the maximum principal stress.
To solve this phase-field evolution equation, we choose a weak coupling. In a time step

t_{n+1}, we first solve the balance equations using $g(\varphi(x, y, t_n))$ in the constitutive model to compute $u(x, y, t_{n+1})$, we then update the history variable $\mathcal{H}(\sigma(u(x, y, t_{n+1})))$ and solve the phase-field evolution equation to compute $\varphi(x, y, t_{n+1})$. This strategy typically requires a very small time step to obtain converged results.

The three variables, namely the two components of the displacements field and the phase-field variable are discretized using FE interpolation. The balance equations and the phase-field evolution equation are solved in weak form using the FE method. The implementation is standard and is not detailed here.

3.2 Data set

If we have to predict crack initiation and propagation paths (fracture patterns) for a wide range different applied loading paths, then FE computations might be too expensive. We have designed a nonlinear data-driven ROM based on deep learning to replicate the FE model and predict fracture patterns for unseen loading paths to solve this issue. To do this, we generate a data set of displacements fields and phase-field variables using the HFM composed of the FE method and the phase-field model. As opposed to previous sections, the data set is hereafter expressed as a tensor:

$$X = (u_1^m(x_i, y_j, t_n), u_2^m(x_i, y_j, t_n), \varphi^m(x_i, y_j, t_n))_{m=1...N_L, n=1...N_T, i=1...N_x, j=1...N_y}\,, \tag{12}$$

with N_L the number of loading paths, N_T the number of time steps, and $N_x \times N_y$ the number of pixels. The data set X corresponds to different loading paths which are stored in the data set

$$S = \left(G_{xx}^m(t_n), G_{xy}^m(t_n), G_{yx}^m(t_n), G_{yy}^m(t_n)\right)_{m=1...N_L, n=1...N_T}\,. \tag{13}$$

For any applied loading $S_{mn} \in \mathbb{R}^4$ at time step t_n, the displacements fields $u_1 \in \mathbb{R}^{N_x \times N_y}$, $u_2 \in \mathbb{R}^{N_x \times N_y}$, and the phase-field variable $\varphi \in \mathbb{R}^{N_x \times N_y}$ are grouped as a data point $X_{mn} \in \mathbb{R}^{N_x \times N_y \times 3}$, which is a three-channel picture. As described in Subsection 3.1, the HFM predicts X_{mn+1} from S_{mn+1} and a history variable depending on $(X_{mp})_{p \leq n}$ by solving physics-based partial differential equations. The nonlinear data-driven ROM based on deep learning presented in Subsection 3.3 is designed to make predictions following the same route as the HFM.

3.3 Data-driven ROM

To construct the nonlinear data-driven ROM, we first reduce the dimensionality of each three-channel picture $X_{mn} \in \mathbb{R}^{N_x \times N_y \times 3}$ using the encoder part of the autoencoder as per Section 2. The result is the reduced state $H_{mn} \in \mathbb{R}^K, K \ll N_x \times N_y \times 3$. To do this, we need to train the CNN autoencoder beforehand on the training samples of the data set X.

The next step is to design and train the predictor of which the inputs are the applied loading conditions $(S_{mn})_{n=2...N_T} \in \mathbb{R}^{(N_T-1)\times 4}$, and the outputs are the reduced states $(H_{mn})_{n=2...N_T} \in \mathbb{R}^{(N_T-1)\times K}$. The initial reduced state H_{m1} may also be provided as input unless it is the same for all loading paths as in the present work. In this case, there is one input/output pair per loading path.

As shown in Figure 5, the proposed deep learning model, coined predictor, works in a recurrent manner. An ANN predicts H_{m2} from a combination of H_{m1} and S_{m2}. The same ANN with exactly the same weights then predicts H_{m3} from a combination of H_{m2} and S_{m3}, and so on for each time step. Consequently, the input layer of the ANN has $K + 4$ neurons and the output layer K neurons. This ANN can hence compute reduced states for a new unseen loading path progressively, time step by time step.

The full displacements fields and phase-field variables can be reconstructed using the decoder part of the autoencoder. The combination of the ANN that uses a lower-dimensional space and the decoder is coined nonlinear data-driven ROM and surrogates the FE model described in Subsection 3.1. The data-driven ROM hence works in the same way as the HFM.

4 Results

In this section, we demonstrate the application of the CNN autoencoder as well as the predictor to fracture mechanics problems. Both neural networks have been implemented with Keras [23], which is the high-level API of TensorFlow 2 [24]. All computations were performed on a machine with an Intel(R) Xenon(R) E5-2630 V4 processor with 10 CPUs (20 cores) and an Nvidia Tesla K80 GPU (2496 cores).

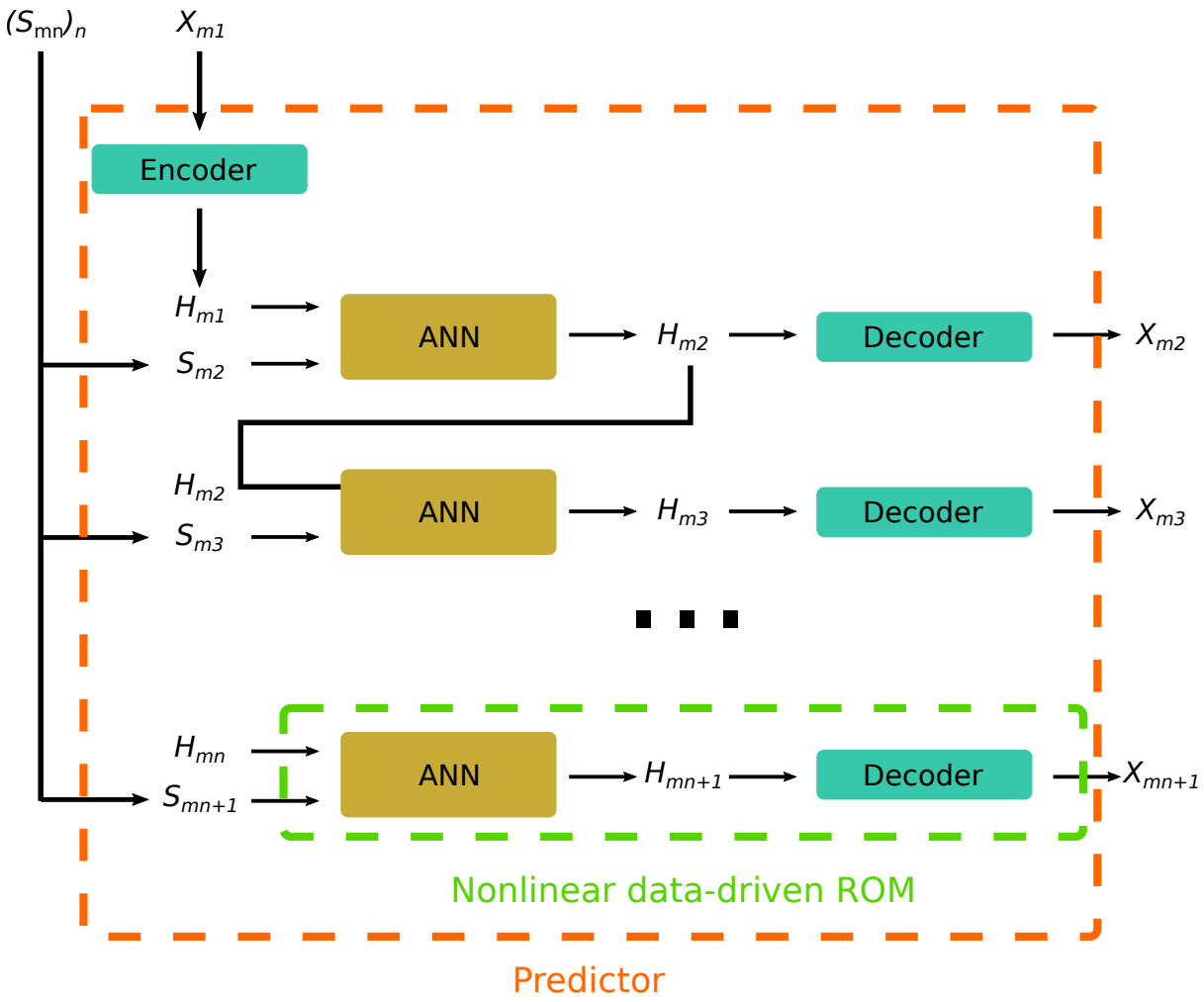


Figure 5: Predictor and nonlinear data-driven ROM for predicting crack initiation and propagation

The test problems and data sets are presented in Subsection 4.1. The CNN autoencoder architecture is detailed in Subsection 4.2 and the predictor architecture in Subsection 4.3. Results are then reported and discussed in Subsection 4.4 for a first test problem, and Subsection 4.5 for a second test problem.

4.1 Data sets

We consider different test problems for which the geometry is always a 2D square plate $\Omega = [0,1] \times [0,1]\text{mm}^2$ containing some heterogeneity. The domain is mainly composed of a brittle material of elastic properties $E = 210\,\text{GPa}$ and $\nu = 0.3$, and fracture properties $\sigma_c = 2445.42\,\text{MPa}$ and $G_c = 2.7\,\text{N}\,\text{mm}^{-1}$. The regularization length of the phase-field model is set to $l_c = 0.04\,\text{mm}$ and the element size to $\approx 0.016\,\text{mm}$ in order to obtain a structured quadrangular FE mesh of 62×62 elements. Full integration with 4 integration points per element is used.

This mesh size and regularization length are voluntarily kept large enough to obtain pictures of $N_x \times N_y = 62 \times 62$ pixels and minimize the size of the database (which, as we will see, will already be large enough with this resolution). For the same reason, the number of time steps is set to $N_T = 201$ (including the initial state), with the final time $T = 0.02\,\text{s}$. The main objective of the present section is to show that the CNN and the nonlinear ROM can compute results that compare well to the HFM. The fact that the HFM could be more accurate is not limiting regarding that objective.

For each problem, we first build a database containing the HFM results for 1024 randomly generated proportional loading paths, with $||G^m(T)||_2^2 = T, m = 1 \dots N_L$. We manually include in this database 8 particular loading paths, which correspond to

$$G^1(T) = \begin{pmatrix} T & 0 \\ 0 & 0 \end{pmatrix}, G^2(T) = \begin{pmatrix} 0 & T \\ 0 & 0 \end{pmatrix}, G^3(T) = \begin{pmatrix} 0 & 0 \\ T & 0 \end{pmatrix},$$
$$G^4(T) = \begin{pmatrix} 0 & 0 \\ 0 & T \end{pmatrix}, G^5(T) = \begin{pmatrix} -T & 0 \\ 0 & 0 \end{pmatrix}, G^6(T) = \begin{pmatrix} 0 & -T \\ 0 & 0 \end{pmatrix}, \tag{14}$$
$$G^7(T) = \begin{pmatrix} 0 & 0 \\ -T & 0 \end{pmatrix}, G^8(T) = \begin{pmatrix} 0 & 0 \\ 0 & -T \end{pmatrix}.$$

This database hence contains 205824 samples of images (corresponding to 1024 loading paths with 201 time steps) with 3 channels such that each image has a size of 62×62, *i.e.*, (1024,201,62,62,3). This database of nearly 18 GB is divided with respect to loading paths into a training and validation data set (175071 samples corresponding to 871 loading paths), and a testing data set (30753 samples corresponding to 153 loading paths). The 8 particular loading paths are always included in the training data set. All data images are normalized by subtracting the sample image mean and dividing by sample image standard deviation computed on the training and validation data set. For validation of the neural networks, a supplementary split is done to separate the validation data set (17487 samples corresponding to 87 loading paths) from the training data set (157584 samples corresponding to 784 loading paths).

We then build a second database for further testing of the models which contains 100 randomly generated non-proportional loading paths. The procedure to get a non-proportional loading path is to: generate a continuous curve of 7 line segments of equal length $\frac{T}{7}$ in the space $(G_{xx}, G_{xy}, G_{yx}, G_{yy})$, but oriented randomly ; compute a cubic B-spline going through the ends of these segments ; interpolate the $G^m(t_n), n = 1 \dots 201$ matrices. This adds 20100 samples of images to be used as supplementary test data.

As illustration, the proportional loading paths for the first test problem are shown in Figures 6a and 6b, and the non-proportional loading paths in Figures 6c and 6d.

It is clearly visible that a wide range of directions are considered for proportional loading paths, but that the magnitude is always the same. For non-proportional loading paths, the visualization is more difficult as all paths are entangled, but their length is again approximately the same.

4.2 CNN autoencoder

The autoencoder used in this section is presented in Figure 7a for the encoder and Figure 7b for the decoder.

The encoder part of the CNN architecture takes an input with the shape (62,62,3). It has two convolutional layers, each followed by a max-pooling layer. After the last max-pooling layer, a flattening layer formats the data in order to finish with one fully

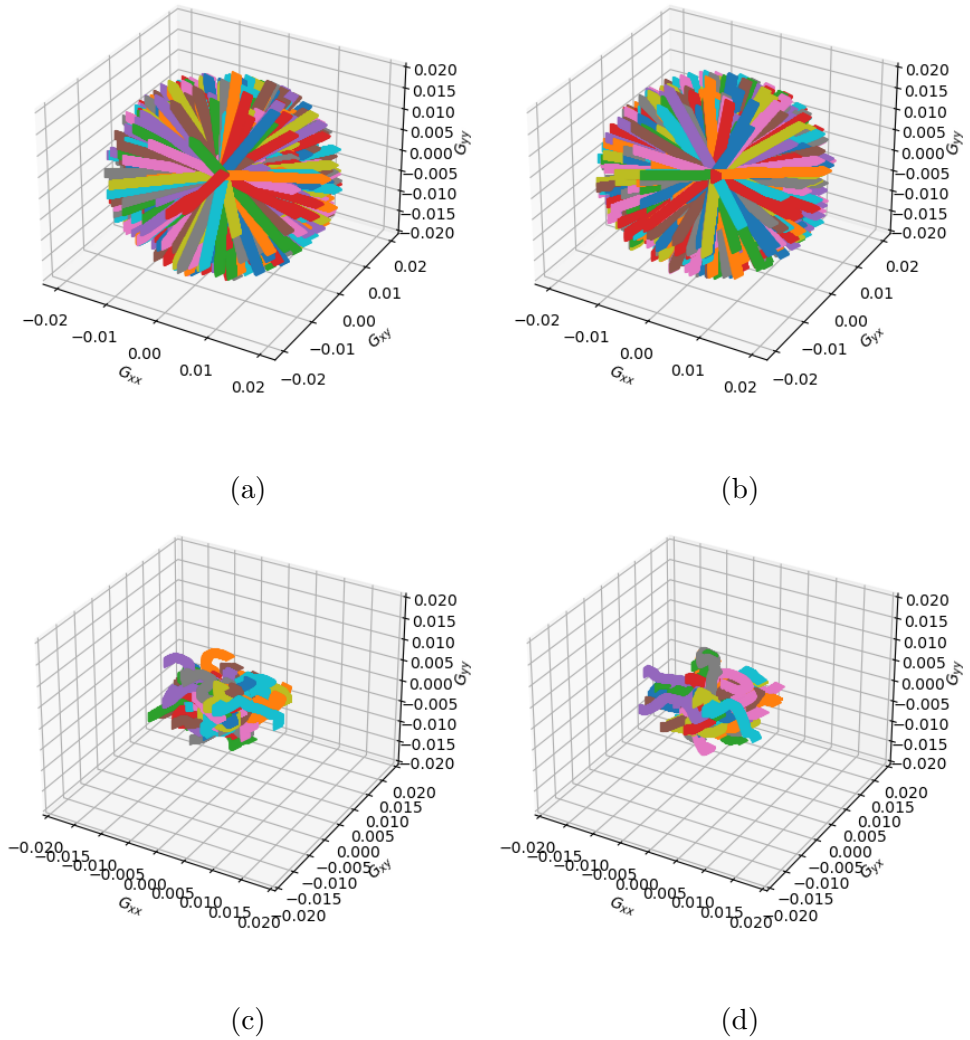


Figure 6: Loading paths generated for the first test problem: (a,b) 1024 proportional loading paths data set, (c,d) 100 non-proportional loading paths data set

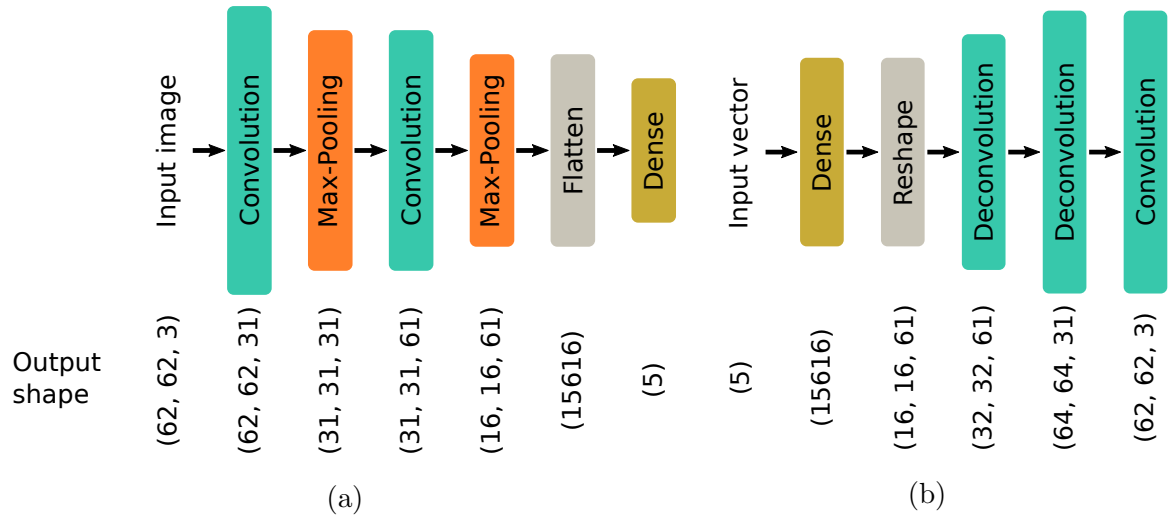


Figure 7: Architecture of the CNN autoencoder: (a) encoder part, (b) decoder part

connected layer. This encoder compresses the input image to the encoding dimension of 5.

The decoder part of the CNN architecture takes the compressed input with the shape (5). It starts with one fully connected layer and then a reshaping layer to format the data into an image. This is followed by two deconvolutional layers (Conv2DTranspose) and one convolutional layer which reconstructs the original image back to its shape (62,62,3). The activation functions used in the last dense layer of the encoder and the last convolutional layer of the decoder are linear. An ELU activation function is used for the two convolutional layers of the encoder as well as those of the decoder. We use the Mean Squared Error loss function to train our CNN autoencoder.

Depending on applications (particularly in fluid mechanics), different CNN autoencoder architectures have been proposed in the literature [12]. The architecture of the CNN autoencoder shown in Figure 7 is very similar in its structure to the one which is considered in Reference [25] for biomedical applications. We tuned our architecture according to our input image shape and the kind of data we had.

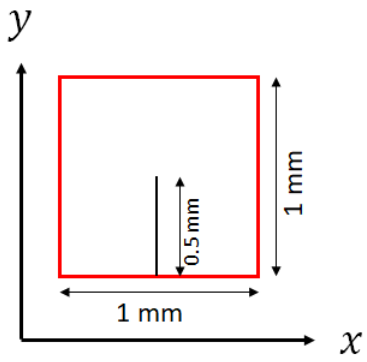


Figure 8: Geometry for the notched test problem. Edges where displacements are imposed as per Equation (5) are shown in red

4.3 Predictor

As explained in Subsection 3.3, the predictor is actually a recurrent network embedding an ANN that predicts the outputs progressively, time step by time step. The architecture of this ANN consists of dense layers. It takes an input with the shape $(K + 4)$. The input shape is a combination of the dimension of the latent space K and the number of components of the loading conditions at each time step (4 in our case). The input layer is followed by three hidden dense layers and one dense output layer. The three hidden layers contain 216, 324, and 540 neurons, respectively, and the ELU activation function is used. The output layer uses a linear activation function to compute an output with the shape (K).

Training the predictor actually means training the embedded ANN that is shown in Figure 5. We use the Mean Squared Error loss function for this training. To avoid the computational cost of repetitively decoding the reduced states during training, we choose to compute the loss directly on the reduced states, which we pre-encode for the training and validation data sets.

4.4 Notched plate

For the first test problem, a heterogeneity is introduced in the form of a notch by duplicating nodes. This only changes the number of nodes but does not affect the number of elements and hence does not raise any issue for the formatting of the data as sets of

62×62 pictures. The geometry of the notched plate is shown in Figure 8. Depending on loading conditions, the crack may propagate in any direction. For non-proportional loading paths, the propagation direction may even vary during loading.
For this test problem, we thoroughly assess the compression error of the CNN autoencoder. Then, we introduce the predictor to assess the accuracy of its predictions.

4.4.1 Reconstruction of fracture patterns for unseen loading paths

In this section, the efficiency of the CNN autoencoder for nonlinear data compression is evaluated. This comparison is done for different levels of compression depending on the encoding dimension. The reconstruction error as well as the computation time are analyzed.

We obtain a mean RMSE over the 153 unseen proportional loading paths of 0.0331 using an encoding dimension of 5, 0.0234 using 10 and 0.0119 using 50. It is reminded that the RMSE is computed over normalized data. The very low error obtained using an encoding dimension of only 5 is hence particularly promising.
The mean value may however hide large errors for some samples. The RMSE for each sample is shown in Figure 9. We can observe that there are several samples for which the RMSE is superior than 0.05, and a few samples for which it is over 0.5. We obtain a systematic decrease of the RMSE when using a larger encoding dimension. It is in fact possible to reduce the RMSE for most samples below 0.2 by using an encoding dimension of 50.

It is interesting to analyze the worst result, which is the sample leading to the highest peak in the middle of Figure 9. This sample corresponds to the last time step of some loading path. All non-normalized fields for this particular sample are shown in Figure 10. The boundary conditions for this sample correspond to shear loading, as we see the crack propagating along the diagonal in the HFM result. Due to the boundary conditions used to generate the data set, however, we also see the crack initiating and propagating along the boundaries of the plate.

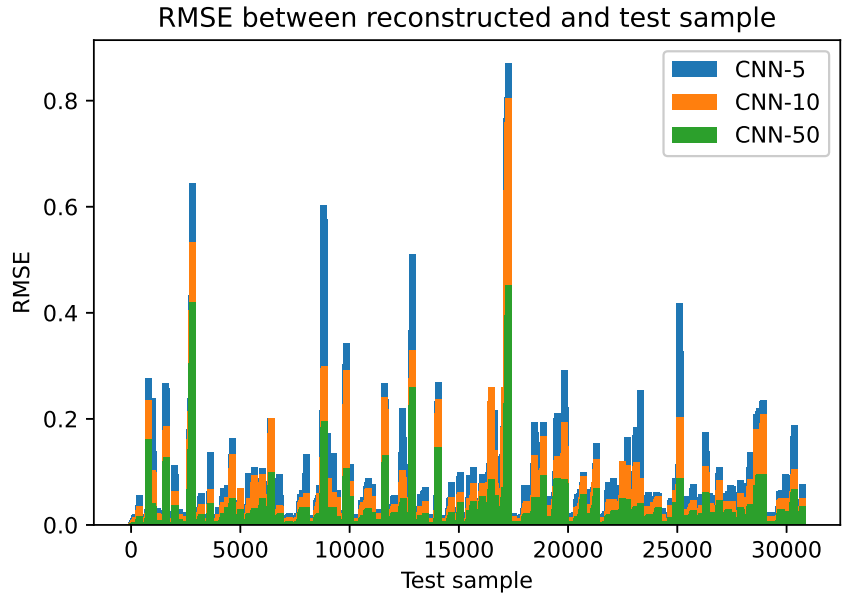


Figure 9: Notched test: RMSE between each reconstructed and original sample using different encoding dimensions for 153 unseen proportional loading paths

The CNN autoencoder clearly has a difficulty in capturing this complex failure pattern, especially for the phase-field variable. This is easier to analyze by looking at Figure 11, which shows the HFM result and the absolute error fields on non-normalized data with respect to the HFM result for the different encoding dimensions.

The difficulty comes from the phase-field variable and the propagation of the crack along the boundary of the domain. With an encoding dimension of 5, the CNN autoencoder over-estimates failure, as there are two regions with no crack along the boundary which are lost after reconstruction. These regions are recovered when using an encoding dimension of 10 or 50.

Since a crack along the boundary has no significant effect on the displacements field, the latter is accurately compressed by the CNN autoencoder, even for an encoding dimension of 5. The only errors that remain and that can be reduced using a larger encoding dimension are located near the crack and are due to the displacements jump.

Regarding computation time, it is mainly spent training the CNN autoencoder. On the one hand, the training time is indeed close to 50 h using an encoding dimension of 5, and 60 h using 50. On the other hand, computing the HFM results for the 1024 propor-

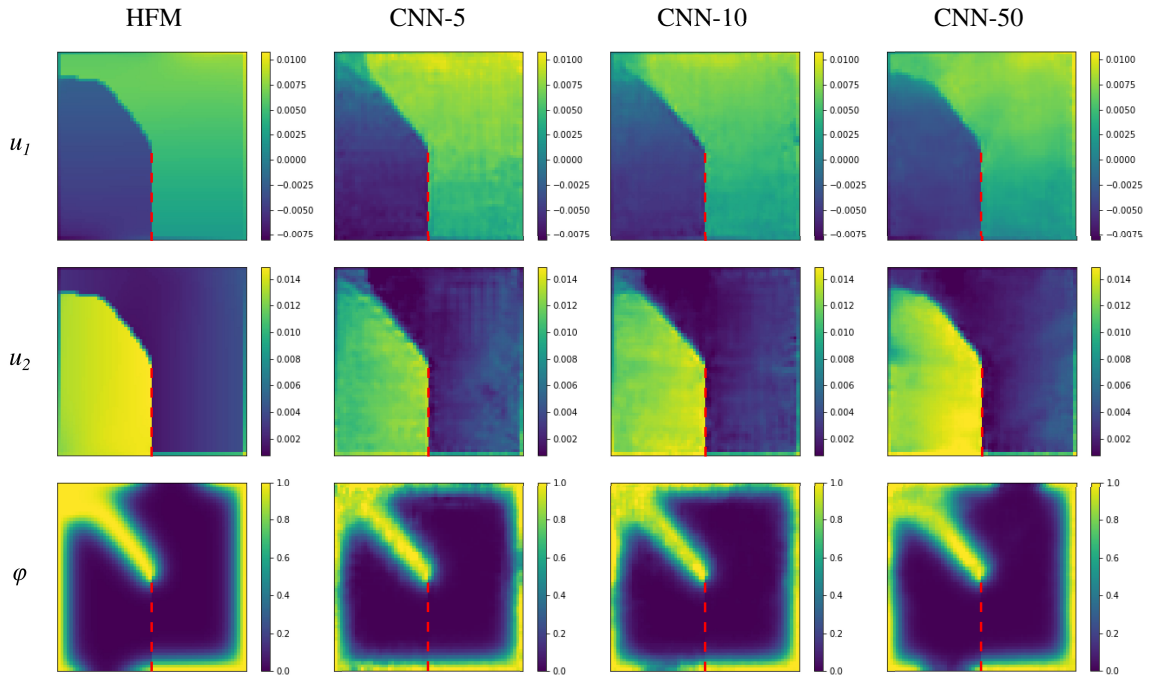


Figure 10: Notched test: Comparison of the reconstructed image with the original image for an unseen proportional loading path (see Supplementary Fig. 1) using the CNN autoencoder with different encoding dimensions. Each row is a different field and each column is a different result. Displacements are in millimeters. The notch is shown as a red dashed line

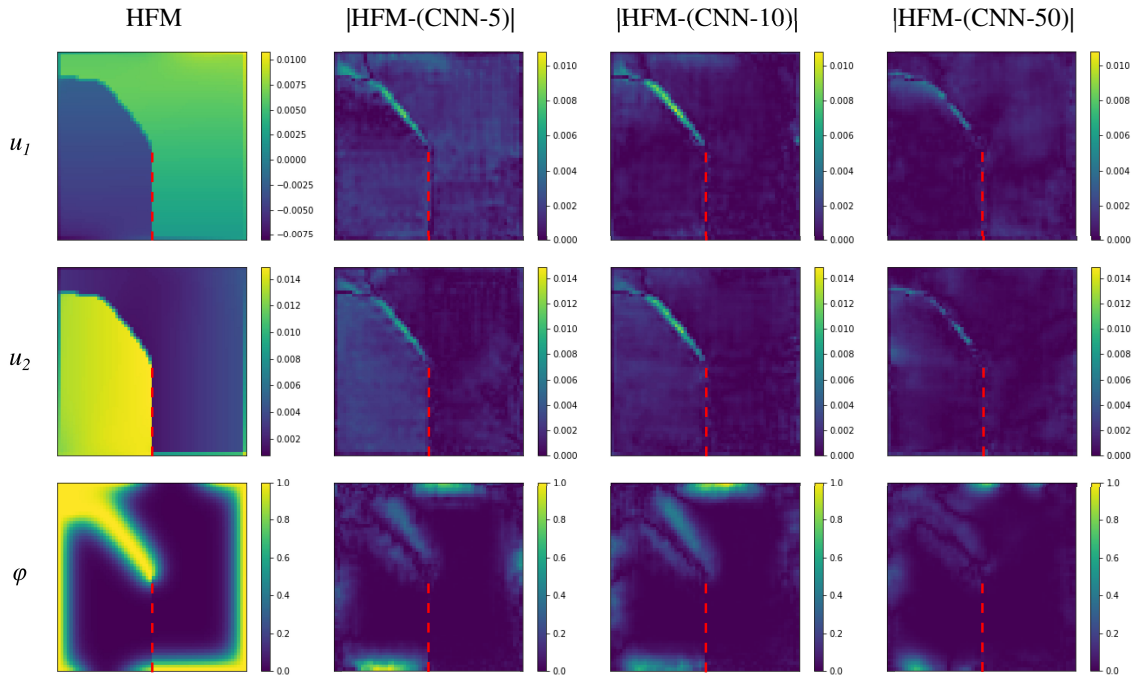


Figure 11: Notched test: Original image and absolute error between the reconstructed image and the original image for an unseen proportional loading path (see Supplementary Fig. 1) using the CNN autoencoder with different encoding dimensions. Each row is a different field and each column is a different result. Displacements are in millimeters. The notch is shown as a red dashed line

tional loading paths takes approximately 16 h (55 s per loading path) on a workstation with an Intel Xeon(R) W-2175 processor with 14 CPUs (28 cores). Generating data and training the CNN autoencoder hence takes days for this problem, with no significant increase in the training time when using a larger encoding dimension.
The CNN autoencoder is relatively fast once trained, as encoding and decoding the 201 samples for a whole loading path requires in average 1.3 s (6.5 ms per sample). This reconstruction time does not vary significantly when changing the encoding dimension.

To summarize, although the mean RMSE is already very low when using an encoding dimension of only 5, large errors are obtained for some samples. These can be reduced by using a larger encoding dimension, with no significant increase in the training time or the reconstruction time. To build our predictor, we choose an encoding dimension of 50.

4.4.2 Prediction of fracture patterns for unseen loading paths

In the following, we develop the predictor described in Figure 5 with an encoding dimension of 50 as low dimensional space, thanks to the CNN autoencoder trained with a proportional data set. To train the predictor, we use only the training data set from the proportional loading paths database. We then use this trained predictor to make predictions for unseen loading conditions of proportional as well as non-proportional loading paths.

We obtain a mean RMSE for the 153 unseen proportional loading paths of 0.0559, which is of the same order as the reconstruction error for the chosen encoding dimension (0.0119). In our work, even though the CNN autoencoder takes longer to train, its primary purpose is to produce a low dimensional space to construct the predictor. These results hence confirm that the CNN autoencoder successfully eases the task of making predictions.
The RMSE distribution over all testing samples is shown in Figure 12. We observe that the predictor almost systematically leads to larger errors as compared to the CNN autoencoder. This is no surprising as making a prediction requires going through the ANN and then the decoder.

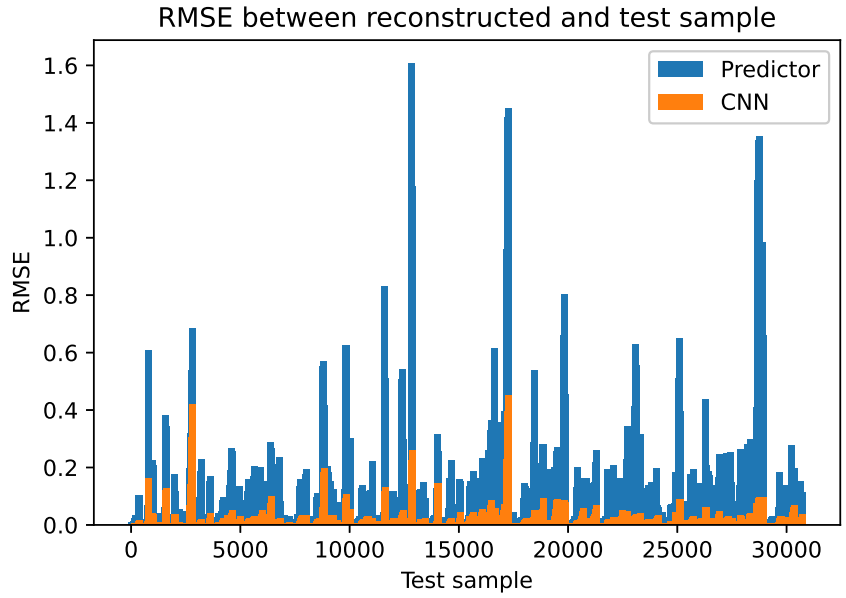


Figure 12: Notched test: RMSE between each reconstructed and original sample, as well as between each predicted and original sample for 153 unseen proportional loading paths

There are more samples for which the RMSE is larger than 0.5, and we even find some samples for which the error is larger than 1. This shows that the CNN autoencoder is a promising tool for compressing highly nonlinear mechanical data, but that more research is needed regarding the predictor.

It is interesting to look again at the same sample as previously shown in Figure 11. We indeed find that the prediction error is at least twice larger than the reconstruction error for this sample, and corresponds to the second highest peak in the middle of Figure 12. The absolute errors using the predictor for this sample are shown in Figure 13.
As opposed to what the large errors may indicate, the predictor accurately captures the propagation of the crack along the diagonal due to the shear loading. The errors are again localized along the boundary of the plate for the phase-field variable, and are relatively low for the displacements field.

The capabilities of the CNN autoencoder and the predictor to capture complex crack propagation patterns are also tested for 100 unseen non-proportional loading paths. We

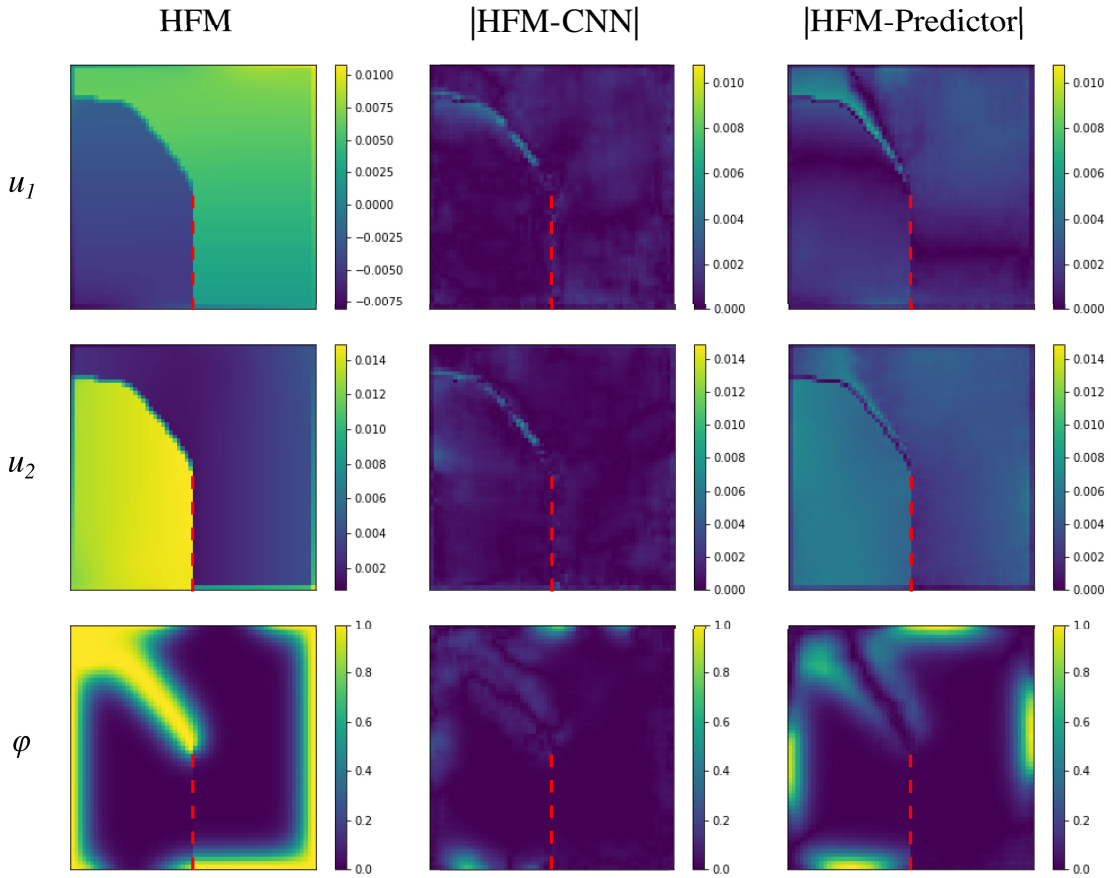


Figure 13: Notched test: Original image and absolute error between the reconstructed (using the CNN autoencoder) and predicted (using the predictor) images and the original image for an unseen proportional loading path (see Supplementary Fig. 1). Each row is a different field and each column is a different result. Displacements are in millimeters. The notch is shown as a red dashed line

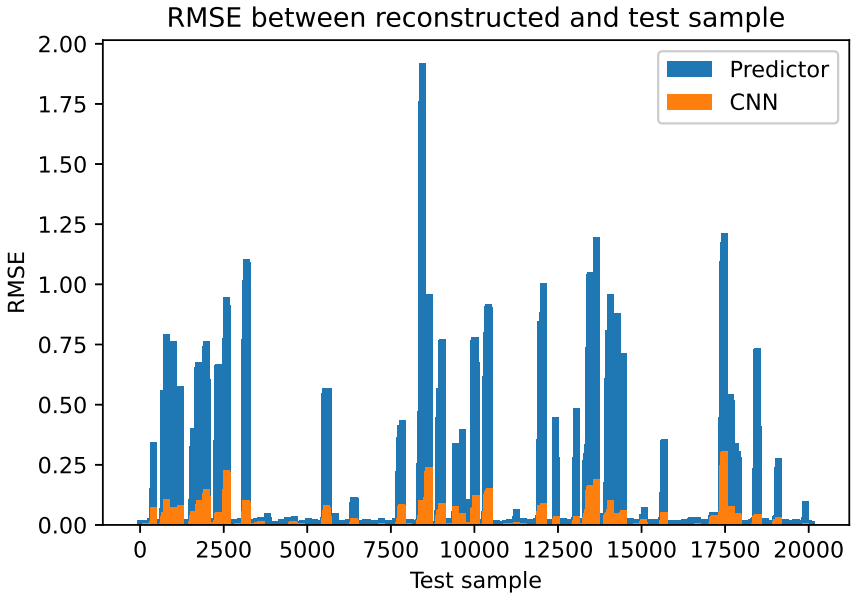


Figure 14: Notched test: RMSE between each reconstructed and original sample, as well as between each predicted and original sample for 100 unseen non-proportional loading paths

find a mean RMSE of 0.0124 for reconstruction and 0.0887 for prediction. This increase in the errors is expected as both the CNN autoencoder and the predictor were trained only on the training data set of proportional loading paths. This increase is nevertheless not significant, which demonstrates the robustness and good generalization capabilities of the proposed approach.

The error distributions are shown in Figure 14. For these non-proportional loading paths we find many cases where there is no crack initiation and which lead to errors very close to zero for both the CNN autoencoder and the predictor. For the remaining cases, we find errors that are of the same order as those found for proportional loading paths in Figure 12.

We choose again to analyze the result for the sample with largest reconstruction error. Absolute errors for this sample are shown in Figure 15. On the one hand, this example confirms that the CNN autoencoder compresses very efficiently the images, even for unseen non-proportional loading paths. It is indeed important to point out that there is no

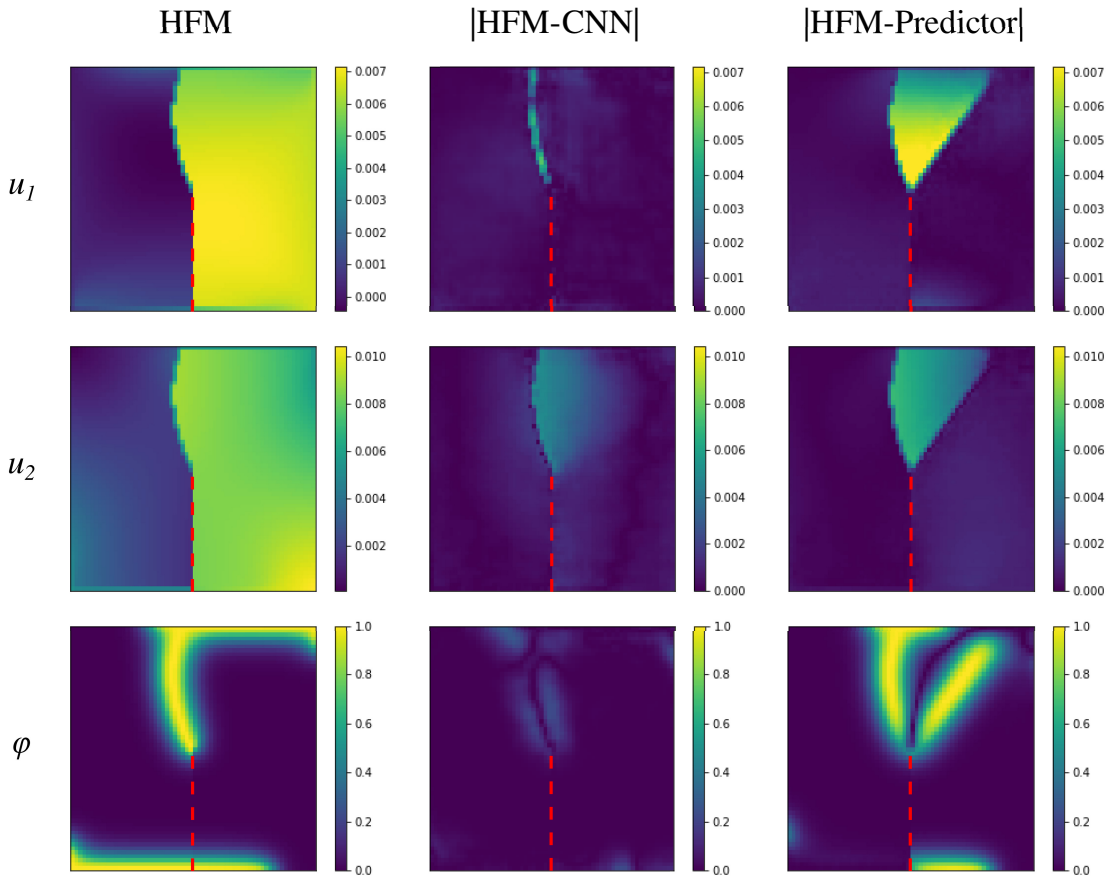


Figure 15: Notched test: Original image and absolute error between the reconstructed (using the CNN autoencoder) and predicted (using the predictor) images and the original image for an unseen non-proportional loading path (see Supplementary Fig. 2). Each row is a different field and each column is a different result. Displacements are in millimeters. The notch is shown as a red dashed line

sample with a curved crack in the training data set.

On the other hand, the predictor finds a straight crack along the wrong direction, which leads to very large errors. Improving the integration of the loading history in the predictor seems necessary to improve the results. This could be achieved by developing a CNN autoencoder for the history variable of the phase-field model and introducing it in the predictor. This will be considered in a future work.

Regarding computation times, finally, we find that the training time for the predictor is negligible, as it is of only 40 min. Once trained, the time spent predicting reduced

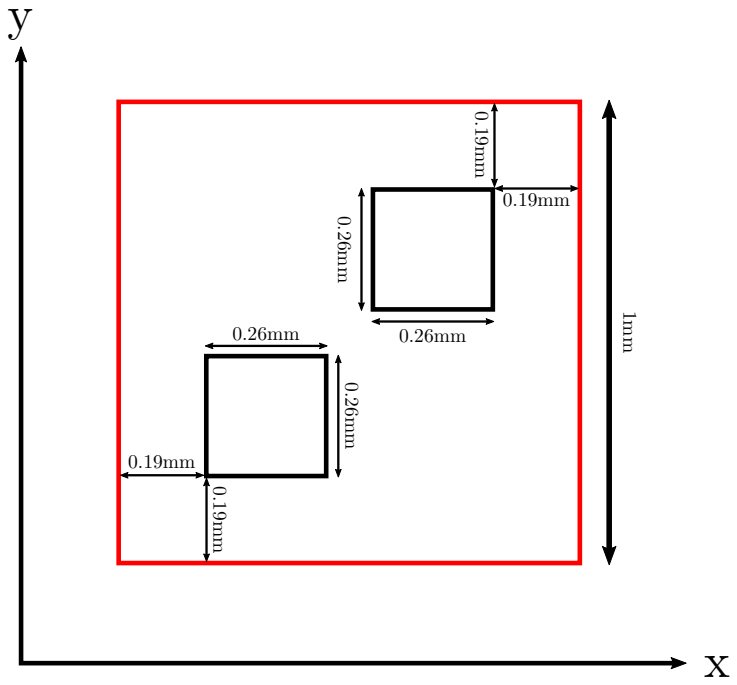


Figure 16: Geometry for the square holes test problem. Edges where displacements are imposed as per Equation (5) are shown in red

states and then decoding them is very low as it is close to 0.22 s per loading path. The proposed predictor is thus drastically cheaper than the HFM, with a reduction of the computation time by a factor of at least 250.

4.5 Plate with two square holes

In the notched test problem, the crack propagated in various directions depending on the loading path, but it always initiated at the center of the domain due to the notch or at the boundaries of the plate. In this second test problem, we change the geometry and design a plate containing two square holes as shown in Figure 16. These holes are squares and are meshed so that we can still use a structured FE mesh of 62×62 elements. This model features eight main stress concentration sites which increases the possibilities in terms of fracture patterns. The same material model and properties as previously are used for the plate, and the two square holes are modeled as a brittle material of elastic properties $E = 21\,\mathrm{MPa}$ and $\nu = 0.3$, and fracture properties $\sigma_c = 24.4542\,\mathrm{MPa}$ and $G_c = 2.7\,\mathrm{N\,mm^{-1}}$.

For this test problem, we re-train both the CNN autoencoder and the predictor on the new training data set of proportional loading paths. We first assess the performance of both the CNN autoencoder and the predictor on the testing data set of proportional loading paths, and then on the data set of non-proportional loading paths. We keep an encoding dimension of 50 for the CNN autoencoder.

4.5.1 Reconstruction and prediction of fracture patterns for unseen proportional loading paths

Similarly as done for the notched test problem, we first report the mean RMSE on 153 unseen proportional loading paths, which is of 0.0084 for the CNN autoencoder and 0.0840 for the predictor. These values are of the same order as those obtained for the notched test problem, which shows that the proposed approach can deal with complex crack patterns in termes of initiation as well as propagation.
Surprisingly, the mean RMSE for the reconstruction is slightly lower while the prediction one is slightly higher. This could be because of the presence of the holes. The RMSE for all samples is reported in Figure 17.

Even though the mean errors are of the same order as for the notched test problem, we can see that there is less variability for the CNN autoencoder. It performs very well for all loading paths, as there is no sample with an RMSE over 0.25, as opposed to the notched test problem for which we found some samples with an RMSE over 0.4 (Figure 12). This could be explained by the fact that the size of the training data set is the same for the two problems, although the complexity of the crack patterns may be of a different level. It is hence possible that the CNN autoencoder could be trained with a smaller training data set for the square holes test problem.
We cannot reach the same conclusion for the predictor, which performs similarly to the notched test problem. Figure 12 additionally shows that there is nearly no correlation between the error of the CNN autoencoder and that of the predictor, for example the sample for which the prediction error is the highest is not the same one as for the reconstruction error. We choose to look at the results for the sample corresponding to the second highest peak in Figure 12, and for which the reconstruction error is the largest.

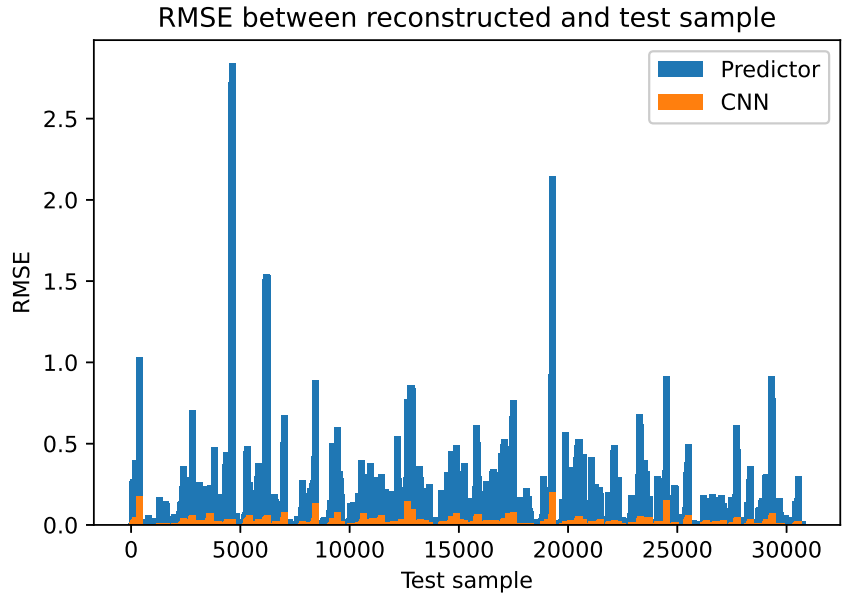


Figure 17: Square holes test: RMSE between each reconstructed and original sample, as well as between each predicted and original sample for 153 unseen proportional loading paths

The absolute errors for this sample are shown in Figure 18.

Although the reconstruction error for this sample is close to 0.25, we can observe that the CNN autoencoder accurately reconstructs both the displacements field and the phase-field. This is particularly challenging as there are multiple crack initiation sites, as opposed to the notched test problem, with small bean-shaped cracks between the square holes. Note that the loading path is proportional. Curved cracks are obtained due to the interaction between the propagating cracks and the square holes. This result clearly shows the high nonlinearity of these fracture problems.
The predictor does not capture these bean-shaped cracks accurately, which leads to large errors for both the displacements field and the phase-field in the center of the plate.

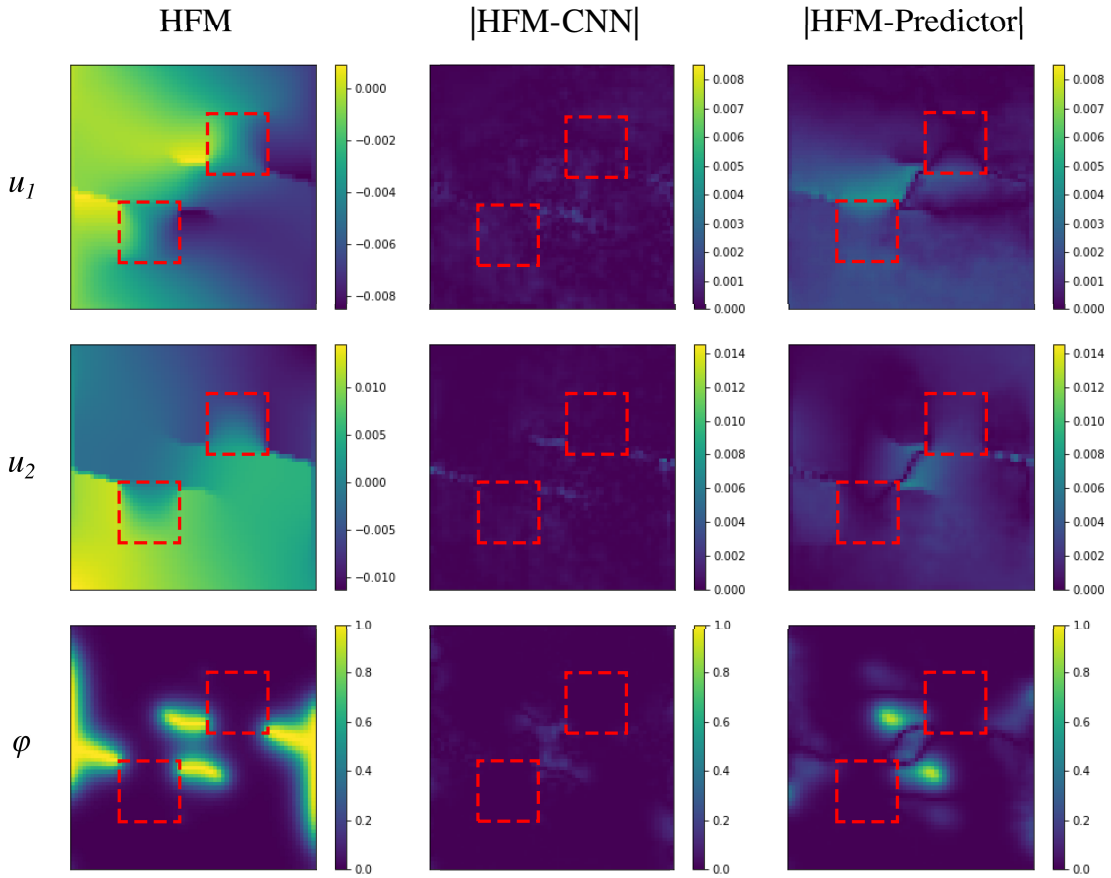


Figure 18: Square holes test: Original image and absolute error between the reconstructed (using the CNN autoencoder) and predicted (using the predictor) images and the original image for an unseen proportional loading path (see Supplementary Fig. 3). Each row is a different field and each column is a different result. Displacements are in millimeters. The squares are delimited by red dashed lines

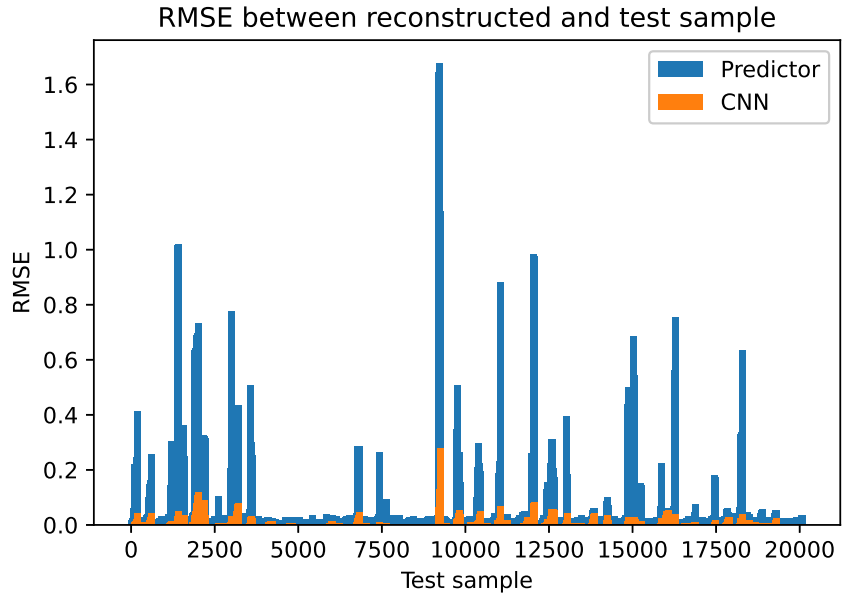


Figure 19: Square holes test: RMSE between each reconstructed and original sample, as well as between each predicted and original sample for 100 unseen non-proportional loading paths

4.5.2 Reconstruction and prediction of fracture patterns for unseen non-proportional loading paths

For non-proportional loading paths, the results in terms of mean RMSE are still of the same order as those obtained for proportional loading paths, as we find a mean RMSE of 0.0052 for the CNN autoencoder and 0.0458 for the predictor. It is quite surprising to observe that the errors on the non-proportional loading paths data set are significantly lower than on the proportional loading paths one. The models have indeed only been trained on proportional loading paths. This can be explained by the fact that for this test problem there are some loading paths where no crack initiation occurs and for which the errors are very small, as shown in Figure 19.

For these non-proportional loading paths, the sample with the largest reconstruction error is also the one with the largest prediction error. Absolute errors for this sample are shown in Figure 19. This result is completely different from the one shown in Figure 18, which demonstrates once again the high nonlinearity of fracture problems.

The CNN autoencoder performs quite well, except for the second component of the dis-

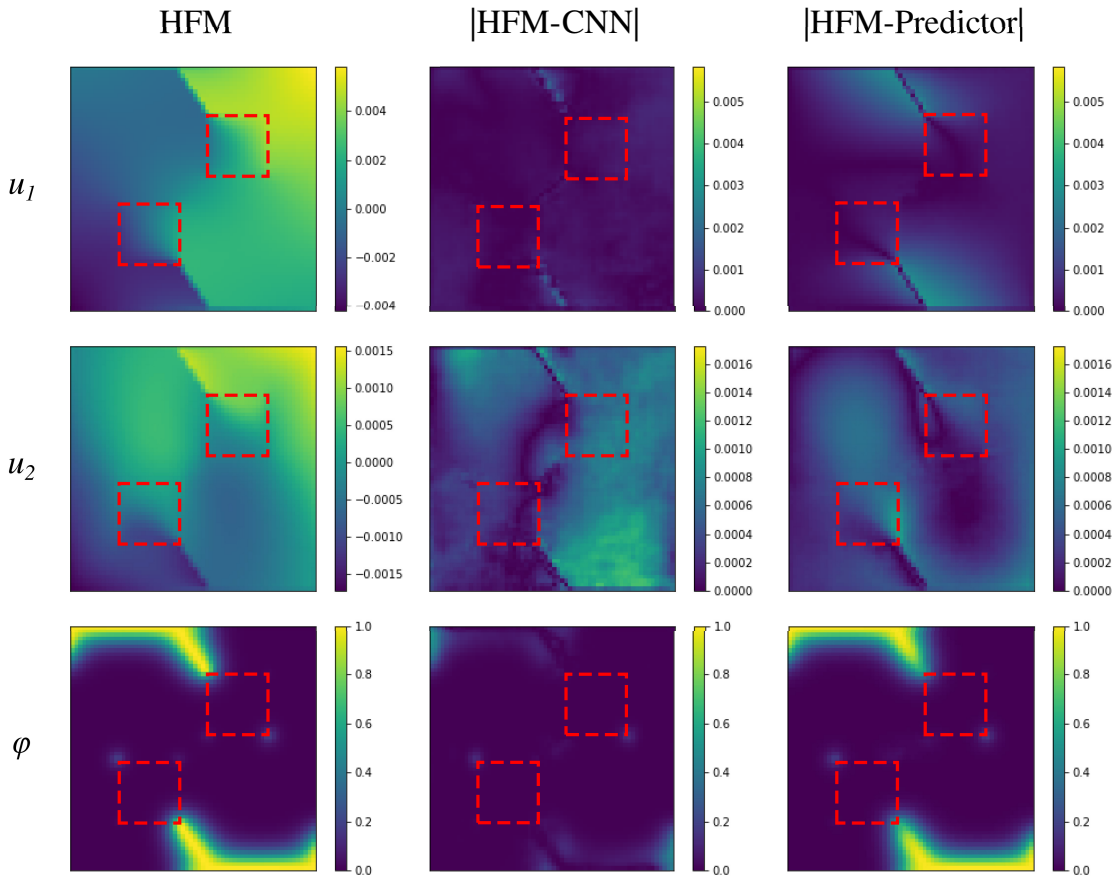


Figure 20: Square holes test: Original image and absolute error between the reconstructed (using the CNN autoencoder) and predicted (using the predictor) images and the original image for an unseen non-proportional loading path (see Supplementary Fig. 4). Each row is a different field and each column is a different result. Displacements are in millimeters. The squares are delimited by red dashed lines

placements field. The prediction is however completely wrong for the predictor, which predicts no crack. This demonstrates the capabilities of the CNN autoencoder for reconstructing highly nonlinear fracture data both for proportional and non-proportional loading paths, with a training data set computed only for proportional loading paths. The predictor requires improvement, especially regarding the integration of the loading history.

5 Conclusion

In this paper, we have designed a nonlinear data-driven ROM based on deep learning in order to model a highly nonlinear fracture mechanics problem. As a primary step of the deep learning approach, data is generated by modeling and solving the fracture mechanics problem using a phase-field model and the FE method. This is done for a wide range of randomly generated proportional loading paths. Non-proportional loading paths are also considered, but only for testing the nonlinear ROM.

Firstly, the fundamental brick of the proposed deep learning approach is dimensionality reduction. A CNN autoencoder is introduced to produce the reduced space. Autoencoders are neural networks specifically designed to compress nonlinear data.

Secondly, the decoder part of the CNN autoencoder is completed with an ANN to form the nonlinear data-driven ROM. A so-called predictor is developed where the nonlinear data-driven ROM is used recurrently to compute predictions for a whole loading path. We found that predicting crack propagation patterns directly from the loading path using the predictor is drastically cheaper in computation time as compared to the FE method, with a factor of at least 250.

Our main conclusions from these simulations are that:

- The CNN autoencoder performs well in terms of reconstruction. It is hence potentially the right choice for reducing the dimensionality of nonlinear data.
- The CNN autoencoder is however quite demanding computationally, especially in the training phase. This intensive work on dimensionality reduction is beneficial for the development of the ANN, which has a very small size and can be trained in a negligible time.
- The developed predictor, trained on proportional loading paths, is promising when predicting fracture patterns for unseen proportional loading paths and data sets related to complex fracture patterns.

These conclusions have been obtained for two different test problems leading to various and complex crack initiation and propagation patterns.

As future work, we would like to investigate the following aspects:

- In our work, the data sets generated using the FE method are based on a structured FE mesh, which allows straightforward conversion to image format, and consequently, a facilitated introduction of convolutional layers in the CNN autoencoder. Ideally, for general applications in fracture mechanics, the deep learning approach should also be compatible with unstructured FE meshes.

- The developed ROM model is purely data-driven. A hybrid approach introducing physics-based partial differential equations or at least physics-based residuals in the loss function of the CNN autoencoder could help reduce and eliminate the large compression errors that have been observed for some loading paths. Although increasing the encoding dimension has been shown to lead to the same result in this paper, a hybrid approach could be smarter and more efficient. This would be even more relevant for the predictor, in particular regarding the integration of the loading history in the recurrent algorithm. Alternative approaches relying on U-Nets could also be worth considering as promising results have recently been obtained for nonlinear material behaviors such as hyperelasticity and elasto-plasticity [26, 27].

References

[1] S. Wold, K. Esbensen, and P. Geladi. "Principal component analysis". In: *Chemometrics and Intelligent Laboratory Systems* 2.1 (1987), pp. 37–52. DOI: 10.1016/0169-7439(87)80084-9.

[2] N. Salem and S. Hussein. "Data dimensional reduction and principal components analysis". In: *Procedia Computer Science* 163 (2019). 16th Learning and Technology Conference 2019 Artificial Intelligence and Machine Learning: Embedding the Intelligence, pp. 292–299. DOI: https://doi.org/10.1016/j.procs.2019.12.111.

[3] J. Almotiri, K. Elleithy, and A. Elleithy. "Comparison of autoencoder and Principal Component Analysis followed by neural network for e-learning using handwritten recognition". In: *2017 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*. 2017, pp. 1–5. DOI: 10.1109/LISAT.2017.8001963.

[4] J. V. Aguado, D. Borzacchiello, E. Lopez, E. Abisset-Chavanne, D. González, E. Cueto, and F. Chinesta. "New trends in computational mechanics: model order reduction, manifold learning and data-driven". In: *9th Annual US-France symposium of the International Center for Applied Computational Mechanics*. Compiègne, France, June 2016.

[5] T. R. F. Phillips, C. E. Heaney, P. N. Smith, and C. C. Pain. "An autoencoder-based reduced-order model for eigenvalue problems with application to neutron diffusion". In: *International Journal for Numerical Methods in Engineering* 122.15 (2021), pp. 3780–3811. DOI: https://doi.org/10.1002/nme.6681.

[6] S. Nikolopoulos, I. Kalogeris, and V. Papadopoulos. "Non-intrusive surrogate modeling for parametrized time-dependent partial differential equations using convolutional autoencoders". In: *Engineering Applications of Artificial Intelligence* 109.November 2021 (2022), p. 104652. DOI: 10.1016/j.engappai.2021.104652.

[7] Z.-H. Hu and Y.-L. Song. "Dimensionality reduction and reconstruction of data based on autoencoder network". In: *Dianzi Yu Xinxi Xuebao/Journal of Electronics and Information Technology* 31 (2009), pp. 1189–1192.

[8] Y. Wang, H. Yao, and S. Zhao. "Auto-encoder based dimensionality reduction". In: *Neurocomputing* 184 (2016), pp. 232–242. DOI: https://doi.org/10.1016/j.neucom.2015.08.104.

[9] E. Plaut. "From Principal Subspaces to Principal Components with Linear Autoencoders". In: *ArXiv* (2018). DOI: 10.48550/ARXIV.1804.10253.

[10] K. Fukami, K. Hasegawa, T. Nakamura, M. Morimoto, and K. Fukagata. "Model Order Reduction with neural networks: application to laminar and turbulent flows". In: *SN Comput. Sci.* 2 (2021), p. 467. DOI: 10.1007/s42979-021-00867-3.

[11] T. Murata, K. Fukami, and K. Fukagata. "Nonlinear mode decomposition with convolutional neural networks for fluid dynamics." In: *Journal of Fluid Mechanics* 882 (2020), A13. DOI: 10.1017/jfm.2019.822.

[12] P. Pant, R. Doshi, P. Bahl, and A. Barati Farimani. "Deep learning for reduced order modelling and efficient temporal evolution of fluid simulations". In: *Physics of Fluids* 33.10 (2021), p. 107101. DOI: 10.1063/5.0062546.

[13] S. E. Ahmed, O. San, A. Rasheed, and T. Iliescu. "Nonlinear proper orthogonal decomposition for convection-dominated flows". In: *Physics of Fluids* 33.12 (2021), p. 121702. DOI: 10.1063/5.0074310.

[14] F. Erdogan. "Fracture mechanics". In: *International Journal of Solids and Structures* 37.1 (2000), pp. 171–183. DOI: 10.1016/S0020-7683(99)00086-4.

[15] A. Yazid, N. Abdelkader, and H. Abdelmadjid. "A state-of-the-art review of the X-FEM for computational fracture mechanics". In: *Applied Mathematical Modelling* 33.12 (2009), pp. 4269–4282. DOI: 10.1016/j.apm.2009.02.010.

[16] J.-Y. Wu and V. P. Nguyen. "A length scale insensitive phase-field damage model for brittle fracture". In: *Journal of Mechanics Physics of Solids* 119 (2018), pp. 20–42. DOI: 10.1016/j.jmps.2018.06.006.

[17] M. Seabra, P. Šuštarič, J. Cesar de Sa, and T. Rodič. "Damage driven crack initiation and propagation in ductile metals using XFEM". In: *Computational Mechanics* 52 (2013), pp. 161–179. DOI: 10.1007/s00466-012-0804-9.

[18] D. Grogan, C. Ó Brádaigh, and S. Leen. "A combined XFEM and cohesive zone model for composite laminate microcracking and permeability". In: *Composite Structures* 120 (2015), pp. 246–261. DOI: 10.1016/j.compstruct.2014.09.068.

[19] Z. Cheng, H. Sun, M. Takeuchi, and J. Katto. "Deep convolutional autoEncoder-based lossy image compression". In: *2018 Picture Coding Symposium (PCS)*. 2018, pp. 253–257. DOI: 10.1109/PCS.2018.8456308.

[20] J.-Y. Wu and Y. Huang. "Comprehensive implementations of phase-field damage models in Abaqus". In: *Theoretical and Applied Fracture Mechanics* 106.December 2019 (2020), p. 102440. DOI: 10.1016/j.tafmec.2019.102440.

[21] J. Lemaitre. "A continuous damage mechanics model for ductile fracture". In: *Journal of Engineering Materials and Technology* 107.1 (1985), pp. 83–89. DOI: 10.1115/1.3225775.

[22] C. Miehe, M. Hofacker, and F. Welschinger. "A phase field model for rate-independent crack propagation: Robust algorithmic implementation based on operator splits". In: *Computer Methods in Applied Mechanics and Engineering* 199.45-48 (2010), pp. 2765–2778. DOI: https://doi.org/10.1016/j.cma.2010.04.011.

[23] F. Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[24] M. Abadi et al. "TensorFlow: a system for large-scale machine learning". In: OSDI'16. Savannah, GA, USA: USENIX Association, 2016, pp. 265–283.

[25] T. Wen and Z. Zhang. "Deep convolution neural network and autoencoders-Based unsupervised feature learning of EEG signals". In: *IEEE Access* 6 (2018), pp. 25399–25410. DOI: 10.1109/ACCESS.2018.2833746.

[26] A. Mendizabal, P. Márquez-Neila, and S. Cotin. "Simulation of hyperelastic materials in real-time using deep learning". In: *Medical Image Analysis* 59 (2020), p. 101569. DOI: 10.1016/j.media.2019.101569.

[27] J. R. Mianroodi, N. H. Siboni, and D. Raabe. "Teaching solid mechanics to artificial intelligence—a fast solver for heterogeneous materials". In: *npj Computational Materials* 7.1 (2021), p. 99. DOI: 10.1038/s41524-021-00571-z.