

Dimensionality Reduction Techniques for Subsurface Modeling

Misael M. Morales, Carlos Torres-Verdín, and Michael J. Pyrcz

May 7, 2025

¹ Abstract

² Dimensionality reduction is essential for subsurface modeling, where vast datasets with millions
³ of measurements pose computational challenges. Advanced hardware and algorithms have en-
⁴ abled automated rock classification, seismic interpretation, and reservoir simulation, yet high-
⁵ dimensional data remain a bottleneck. Traditional feature selection helps simplify models, but
⁶ dimensionality reduction techniques are often required for rapid predictions and enhanced inter-
⁷ pretability. Inspired by computer vision, where images are compressed into meaningful features,
⁸ subsurface modeling benefits from encoding techniques that extract latent representations while
⁹ preserving critical information. This chapter explores popular dimensionality reduction meth-
¹⁰ ods, including singular value decomposition (SVD), principal component analysis (PCA), discrete
¹¹ wavelet transform (DWT), dictionary learning (DL), and deep learning-based autoencoders (AE).
¹² These techniques optimize storage, reduce computational costs, and improve predictive accuracy
¹³ and can be applied to complex workflows such as reservoir simulation, history matching, and
¹⁴ geologic modeling. By transforming complex geospatial data into lower-dimensional space, they
¹⁵ enhance uncertainty quantification and accelerate machine learning workflows. Hence, the chap-
¹⁶ ter provides insights on different dimensionality reduction techniques for subsurface applications,
¹⁷ ensuring robust and scalable modeling for energy resource exploration and development.

¹⁸

¹⁹ **Keywords:** latent space, geologic uncertainty model, spatial data analytics, computer vision

²⁰ Introduction

²¹ During the last decades, subsurface energy resources have seen immense revolutions with the
²² accelerated developments of hardware and algorithmic technologies to support subsurface modeling
²³ [1, 2]. For example, rock classification from core measurements and well logs has been automated
²⁴ using clustering and classification algorithms, while reservoir simulation has seen benefits from

25 rapid proxy models for dynamic predictions [3–6]. In many cases, however, these datasets are
 26 composed of thousands or millions of measurements or grid blocks with tens to hundreds of
 27 features due to the complexity and uncertainty in subsurface models [7–9]. This makes subsurface
 28 modeling a perfect candidate for dimensionality reduction techniques, where hidden patterns and
 29 relationships in data can be simplified into lower-dimensional representations for more robust
 30 predictions and more interpretable models.

31 Supervised machine learning methods, both for regression and classification, can face sig-
 32 nificant challenges when dealing with large subsurface datasets [10, 11]. Several techniques for
 33 feature engineering, feature importance, and regularization have been proposed to intelligently
 34 reduce problems with a large number of variables to the key ones that have the most effect on
 35 the target variable [12, 13]. However, for cases such as reservoir modeling, seismic modeling, and
 36 well log interpretation, a single realization in an uncertainty model can have a large number of
 37 features, making dimensionality reduction crucial for rapid machine learning predictions [14–16].
 38 This is a classical problem in the field of computer vision, where a single image can have upward
 39 of thousands or millions of pixels, and a single label must be extracted. For subsurface model-
 40 ing, this translates into interpreting an uncertainty model for example, a multivariate ensemble
 41 of subsurface properties (e.g., porosity, permeability, brittleness, total organic content, acoustic
 42 impedance), to estimate the recoverable resources in place; or a large 2D or 3D geologic uncer-
 43 tainty model of heterogeneous properties (e.g., porosity, permeability) that can be compressed
 44 into a lower-dimensional representation [17, 18].

45 The main idea behind dimensionality reduction techniques is to compress a dataset, X , into a
 46 latent representation that retains the majority of the patterns and details in the data, also known
 47 as *encoding*, such that,

$$z = \mathcal{E}(X) \quad (1)$$

48 where z is the latent representation of X and $\mathcal{E}(\cdot)$ is a general encoding operator. The latent
 49 representation can then be used for a variety of tasks without the need to use the full-dimensional
 50 data given that z contains the majority of patterns and details present in X .

51 Furthermore, the latent representation, z , can be decompressed or *decoded* into the original
 52 data space using a mirroring operator of the encoder, namely the decoder, $\mathcal{D}(\cdot)$, expressed as,

$$X' = \mathcal{D}(\mathcal{E}(X)) = \mathcal{D}(z). \quad (2)$$

53 The goal is then to minimize the difference between X and X' , such that $X \approx X'$, by selecting
 54 an optimal latent dimension and encoder and decoder parameters that allow for reduced mul-
 55 ticollinearity, reduced storage, and accelerated processing while still retaining the majority of
 56 important features in the data [19, 20]. Performing computationally expensive routines, such as
 57 reservoir simulation or history matching, with the latent representation instead of the full data
 58 space now becomes significantly less computationally expensive and memory intensive, saving
 59 computational time and, with the right choice of dimensionality reduction algorithm and latent
 60 dimension, sufficiently accurate [21, 22].

61 To support the valuable adoption of dimensionality reduction in subsurface modeling, this
 62 chapter will focus on the application of several data-driven dimensionality reduction techniques for
 63 subsurface modeling, including singular value decomposition (SVD), principal component analysis
 64 (PCA), discrete wavelet transform (DWT), dictionary learning (DL), and deep learning-based
 65 AutoEncoders (AE).

66 Dataset

67 To demonstrate the application of different dimensionality reduction techniques in subsurface
 68 modeling, a synthetic geologic uncertainty model is used. The dataset consists of an ensemble
 69 of Gaussian-distributed log-permeability values simulated using Sequential Gaussian Simulation
 70 (SGSIM) in the Stanford Geostatistical Modeling Software (SGeMS) [23–25]. The values range
 71 from 0.69×10^{-3} mD to 6.77×10^3 mD. A spherical variogram is used with major and minor
 72 values of 90 and 30, respectively. A total of 500 realizations are simulated with a resolution of
 73 128×128 for a total of 16,384 parameters, such that the original data matrix can be represented
 74 as $X \in \mathbb{R}^{500 \times 128 \times 128}$. Figure 1 shows the first 15 realizations in the geologic uncertainty ensemble.

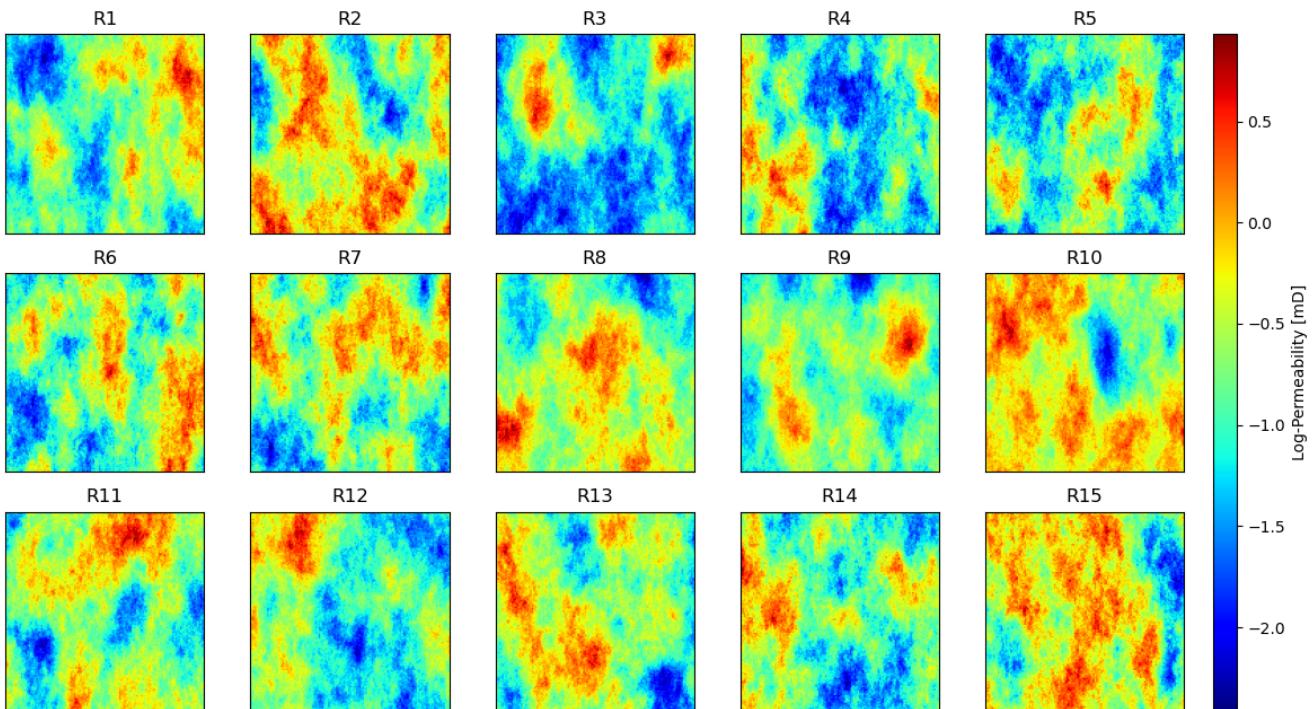


Figure 1: First 15 realizations (R1-R15) of the geologic uncertainty model depicting log-permeability values.

75 Singular Value Decomposition

76 The singular value decomposition (SVD) is among the most important matrix factorization al-
 77 gorithms of the computing and provides a numerically stable matrix decomposition useful for a
 78 wide variety of purposes and applications [26, 27]. Moreover, SVD serves as the underlying al-
 79 gorithm of principal component analysis (PCA), another significant algorithm for dimensionality
 80 reduction [28, 29]. While some dimensionality reduction algorithms provide a generic basis for
 81 latent, or hidden, low-dimensional representations, such as the fast Fourier transform (FFT) and
 82 the discrete wavelet transform (DWT), SVD provides a tailored basis. Tailored basis are extracted
 83 directly from the data matrix such that dominant patterns in the data are expressed purely from
 84 data, without the addition of expert knowledge or prior information, while generic bases use pre-
 85 determined functions, such as sines and cosines, to describe the dominant patterns in the data
 86 [30, 31]. Furthermore, SVD is proved to exist for all matrices, unlike other transformations such
 87 as the eigendecomposition, making it flexible and versatile to use with any dataset [32].

88 For subsurface applications, we will focus on image processing or computer vision applications,
 89 and where a reservoir model can be described as a matrix with each entry representing a grid block
 90 value of a subsurface property, similar to pixels in an image. Let X be our data matrix such that
 91 $X \in \mathbb{R}^{n \times m}$ is expressed as,

$$X = \begin{bmatrix} & & & | \\ | & | & \dots & | \\ x_1 & x_2 & \dots & x_m \\ | & | & & | \end{bmatrix}, \quad (3)$$

92 where each column of X represents a realization of a subsurface uncertainty model, where each
 93 realization $x_k \in \mathbb{R}^n$ has n entries representing each pixel or grid block in the model. Often, the
 94 number of entries, n , (tens of thousands or millions) is much larger than the number of realizations,
 95 m (hundreds or thousands).

96 The SVD is a matrix decomposition of X such that,

$$X = U\Sigma V^T, \quad (4)$$

97 where $U \in \mathbb{R}^{n \times n}$ and $V \in \mathbb{R}^{m \times m}$ are unitary matrices with orthonormal columns, and $\Sigma \in \mathbb{R}^{n \times m}$
 98 is a real non-negative diagonal matrix. The columns of U are called the *left-singular vectors* of X
 99 and the rows of V^T are the *right-singular vectors* of X . The diagonal entries of Σ are called the
 100 *singular values* of X and are ordered by magnitude such that the rows and columns of U and V^T
 101 represent the strength of importance of the data matrix X . Finally, the rank of X is equal to the
 102 number of nonzero singular values in Σ .

103 In the case of $m \leq n$, Σ has at most m nonzero elements in the diagonal and can be expressed
 104 as,

$$\Sigma = \begin{bmatrix} \hat{\Sigma} \\ 0 \end{bmatrix}, \quad (5)$$

105 where $\hat{\Sigma}$ is the nonzero portion of Σ . Thus, we can obtain a lower-dimensional representation of
106 X using the so-called *economy* SVD,

$$X = U\Sigma V^T = [\hat{U} \quad \hat{U}^\perp] \begin{bmatrix} \hat{\Sigma} \\ 0 \end{bmatrix} V^T, \quad (6)$$

107 where the columns of \hat{U}^\perp span a complementary and orthogonal vector space of \hat{U} .

108 In the case of a full-rank Σ , the lower-dimensional representation of X can be obtained by
109 truncation of the SVD. Here, the matrices \tilde{U} , $\tilde{\Sigma}$, and \tilde{V}^T represent the truncated versions of the
110 original decomposition matrices up to a chosen singular value such that,

$$X \approx \tilde{U}\tilde{\Sigma}\tilde{V}^T. \quad (7)$$

111 If X is not full-rank ($\text{rank}(X) = r < \min(m, n)$), then some singular values of Σ are zero.
112 Thus, if we reconstruct the original data matrix using the rank r -truncated decomposition then
113 the SVD will be exact, namely,

$$X = U_r \Sigma_r V_r^T. \quad (8)$$

114 However, for truncation values, τ , smaller than the rank of X (or the number of singular
115 values in Σ), there are several ways to optimally select τ [33]. One such way is the Scree plot,
116 or an elbow plot, of the number of singular values retained against the cumulative sum of the
117 singular values, known as *energy*. Once a truncation value τ is optimally selected, retaining only
118 the leading rows and columns of U , Σ , and V^T will provide an approximate reconstruction of X
119 and a reduced-dimensional representation of the data matrix.

120 Example

121 To perform SVD, we must vectorize the dataset by making each realization a column vector
122 instead of a matrix, namely $X \in \mathbb{R}^{500 \times 16384}$. Figure 2 shows the leading SVD bases obtained for
123 this dataset. Given that the SVD sorts the leading singular values by magnitude, we observe
124 that the first few bases retain information about the large-scale features in the dataset, while the
125 trailing bases retain fine-scale granular details.

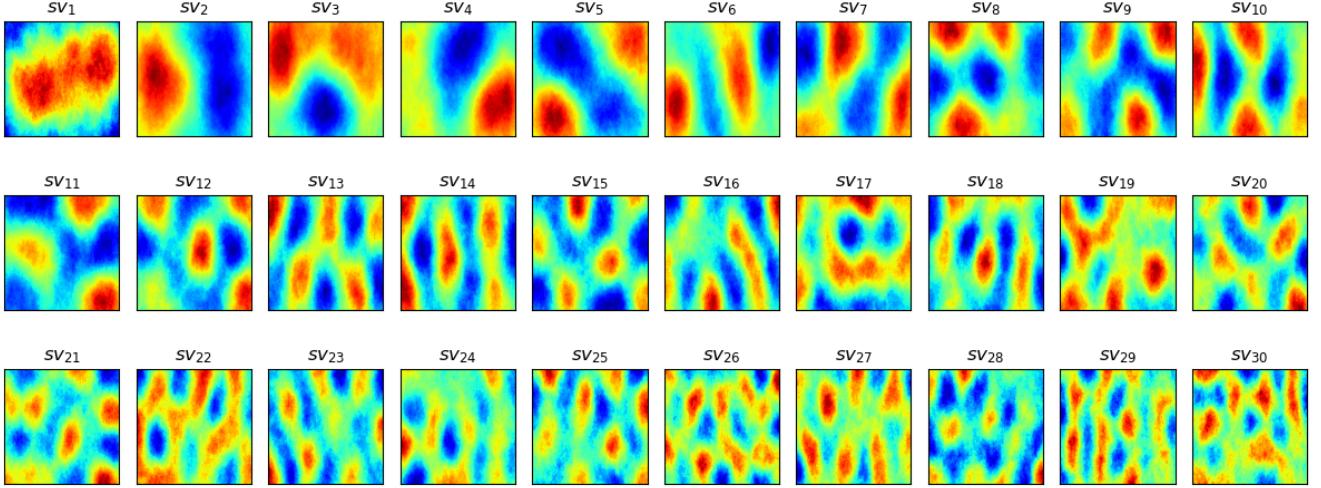


Figure 2: Leading SVD basis (SV_1 - SV_{30}) for geologic uncertainty dataset.

126 It is possible to take the SVD of the geologic uncertainty ensemble using different truncation
 127 values, τ , since $n \leq m$. The Scree plot in Figure 3 shows that by the truncation value $\tau =$
 128 288, the singular values account for approximately 80% of the image energy, and by $\tau = 441$,
 129 the singular values account for approximately 95% of the image energy. Figure 4 shows the
 130 reconstructed images using τ singular values at 20%, 50%, 80% and 95% energy, while Figure 5
 131 shows that by using only $\tau = 441$ singular values, accounting for 95% of energy retained here,
 132 we are able to accurately reconstruct the ensemble realizations with an average $R^2=99.27$ and
 133 structural similarity index measure $SSIM=98.83$. The absolute difference between the true and
 134 reconstructed images, expressed as,

$$\varepsilon = \left| \frac{X - X'}{X} \right|, \quad (9)$$

135 shows that the the majority of geologic features and important details are retained, while mostly
 136 the noise is filtered out and represented as the error or difference between the images. The code
 137 for the SVD example can be found in [Appendix A](#).

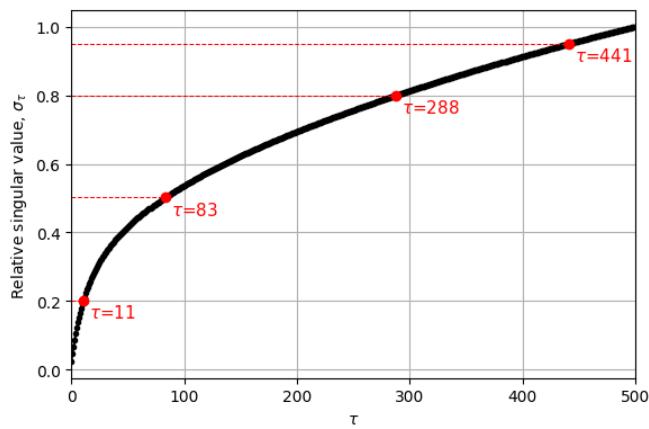


Figure 3: Cumulative energy in the first τ singular values.

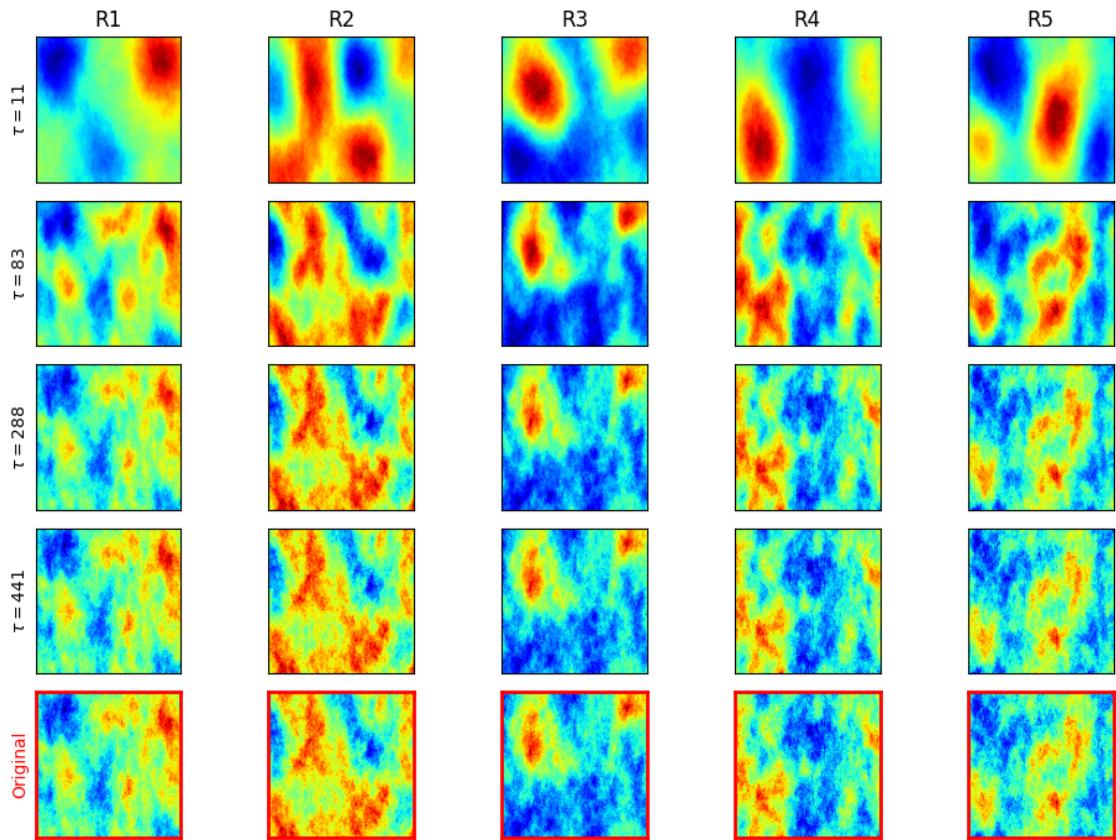


Figure 4: Reconstructed geologic models using τ singular values for the first 5 realizations (R1-R5).

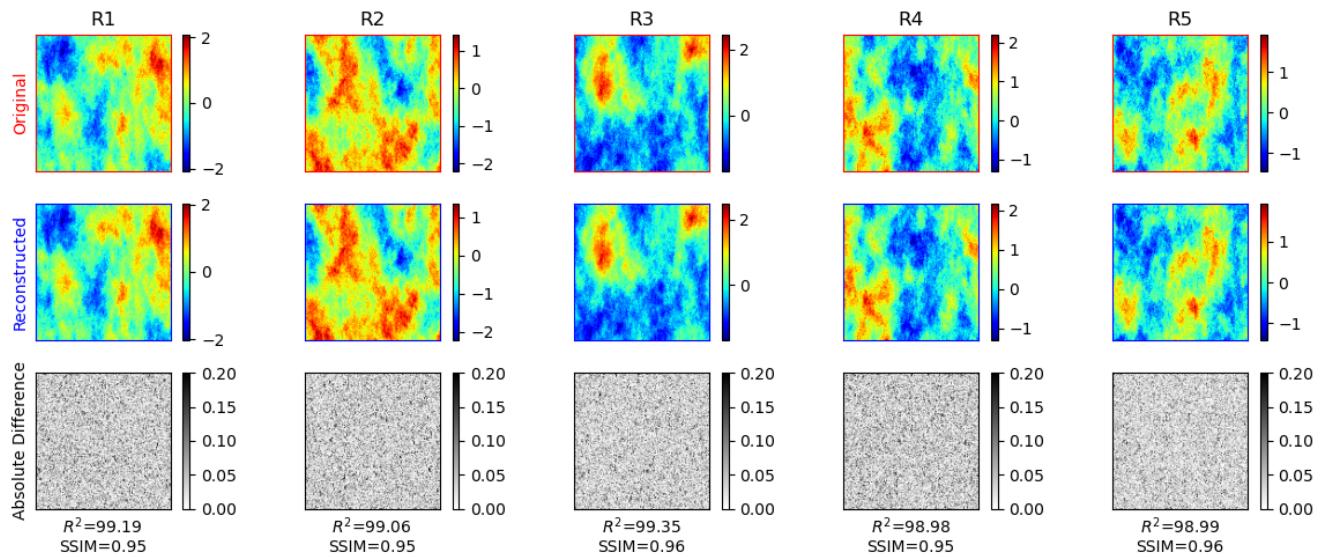


Figure 5: Reconstructed images using $\tau=441$ singular values accounting for 95% of energy retained for the first 5 realizations (R1-R5). The bottom row shows the absolute difference, calculated using Eq. 9.

¹³⁸ Principal Component Analysis

¹³⁹ Principal component analysis (PCA) is currently one of the most popular and versatile dimension-
¹⁴⁰ ality reduction techniques, and it has been widely applied to subsurface modeling [34–36]. Given
¹⁴¹ a data matrix $X \in \mathbb{R}^{n \times m}$, PCA aims to find the best linear subspace in the least-squares sense
¹⁴² using the search of rotated orthogonal bases that maximize the variance explained in the first basis
¹⁴³ vector, followed by the second orthogonal vector, and so on, as shown in Figure 6 [37, 38]. PCA
¹⁴⁴ preprocesses the data by mean subtraction and setting the variance to unity before performing
¹⁴⁵ SVD. The resulting coordinate system (principal components) are orthogonal to each other but
¹⁴⁶ have maximum correlation with respect to the data measurements.

¹⁴⁷ Let B be the mean-subtracted matrix such that,

$$B = X - \bar{X}, \quad (10)$$

¹⁴⁸ and the data covariance matrix be given by,

$$C = \frac{1}{n-1} B^T B. \quad (11)$$

¹⁴⁹ Then the first principal component is given by,

$$u_1 = \underset{\|u_1\|=1}{\operatorname{argmax}} u_1^T B^T B u_1, \quad (12)$$

¹⁵⁰ where T represents the transposition operator.

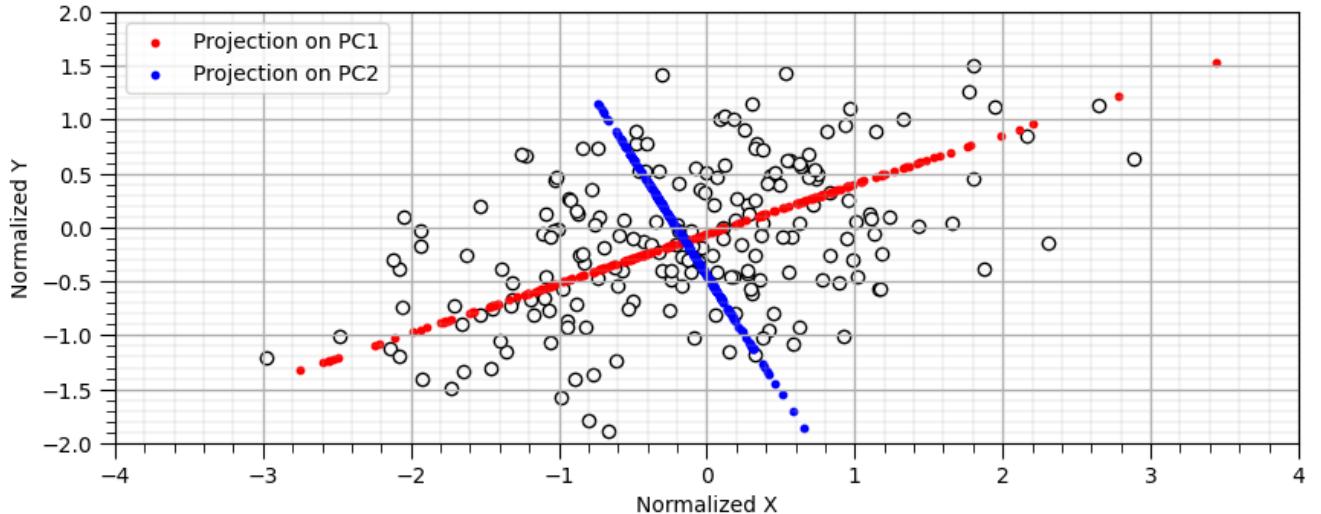


Figure 6: Random multivariate dataset and PCA projection for the first (red) and second (blue) principal components. The PCA projection is the orthogonal rotation to maximize variance explained in the first principal component and remaining variance in the second principal component.

151 Inherently, u_1 is the first eigenvector of the normalized covariance matrix $B^T B$. The maximum
152 number of principal components that can be obtained for a data matrix $X \in \mathbb{R}^{n \times m}$ is given by,

$$\min(m, n - 1), \quad (13)$$

153 and the remaining principal components can be obtained via de eigendecomposition of C , such
154 that,

$$CV = VD, \quad (14)$$

155 where V is the matrix whose columns represent the eigenvectors of the covariance matrix C , and
156 D is a diagonal matrix of eigenvalues corresponding to each eigenvector of V .

157 PCA converts the set of measurements into a set of linearly uncorrelated features, known
158 as principal components, which are a linear combination of the original features. The principal
159 components form a new orthogonal basis, and the principal component scores, or loadings, provide
160 the linear combination weights. Furthermore, PCA can also be interpreted geometrically as a
161 matrix rotation, where the original measurements are transformed into a linearly independent
162 subspace of orthogonal vectors, namely the principal components, that maximize the variance
163 explained. Similar to SVD, the principal components and their corresponding loadings are ordered
164 by magnitude, where the leading terms account for the largest possible variance explained [39].

165 To perform dimensionality reduction on a dataset, we perform truncation and only select k
166 leading principal components to perform the back-transformation. This allows us to reconstruct
167 the back-projected data matrix, X' , by maintaining the principal components with the highest
168 variance explained while removing the trailing principal components that represent the nuances
169 or noise in the data.

170 Example

171 To perform PCA we must also vectorize the geologic uncertainty dataset by making each realization
172 a column vector instead of a matrix, namely $X \in \mathbb{R}^{500 \times 16384}$. Similar to SVD, PCA sorts the
173 leading principal components by magnitude. Figure 7 shows the leading components. We observe
174 that the first few bases retain information about the large-scale features in the dataset, while the
175 trailing bases retain fine-scale granular details. Figure 8 shows that with only $k = 234$ PCs, we
176 are able to retain 95% of the variance explained in the dataset, while using $k = [2, 8, 36]$ retains
177 20%, 50%, and 36% of the variance explained in the dataset, respectively.

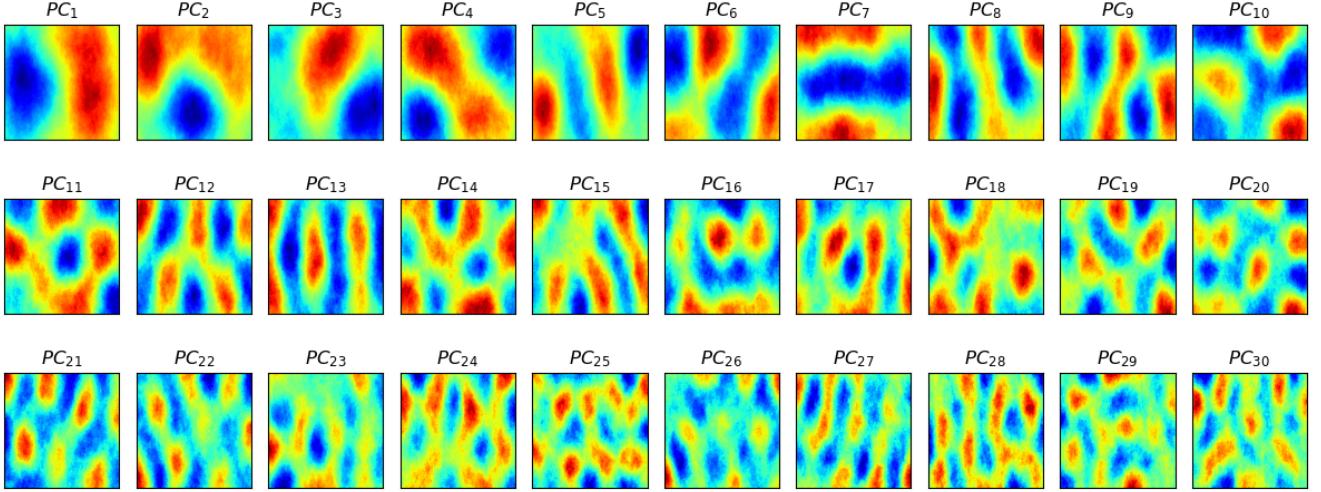


Figure 7: Leading PCA basis (PC_1 - PC_{30}) for geologic uncertainty dataset.

178 Figure 9 show the reconstructed images using k principal components at 20%, 50%, 80% and
 179 95% variance explained, while Figure 10 shows that using only $k = 234$ principal components,
 180 accounting for 95% of variance explained, we are able to accurately reconstruct the ensemble
 181 realizations with an average $R^2=95.41$ and $SSIM=92.00$. The code for the PCA example can be
 182 found in [Appendix B](#).

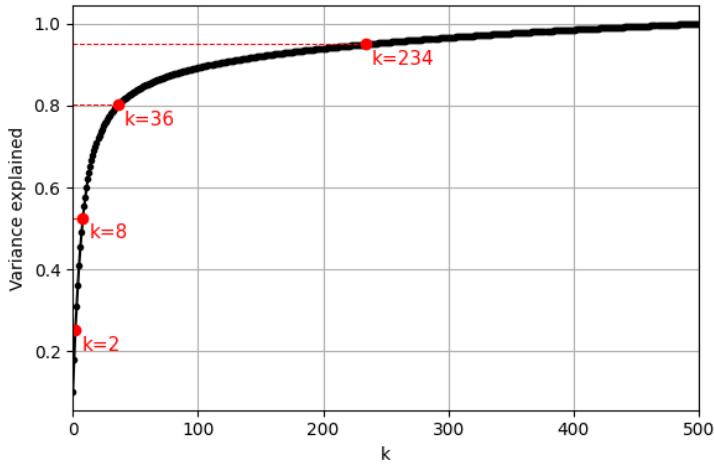


Figure 8: Variance explained against the first k principal components.

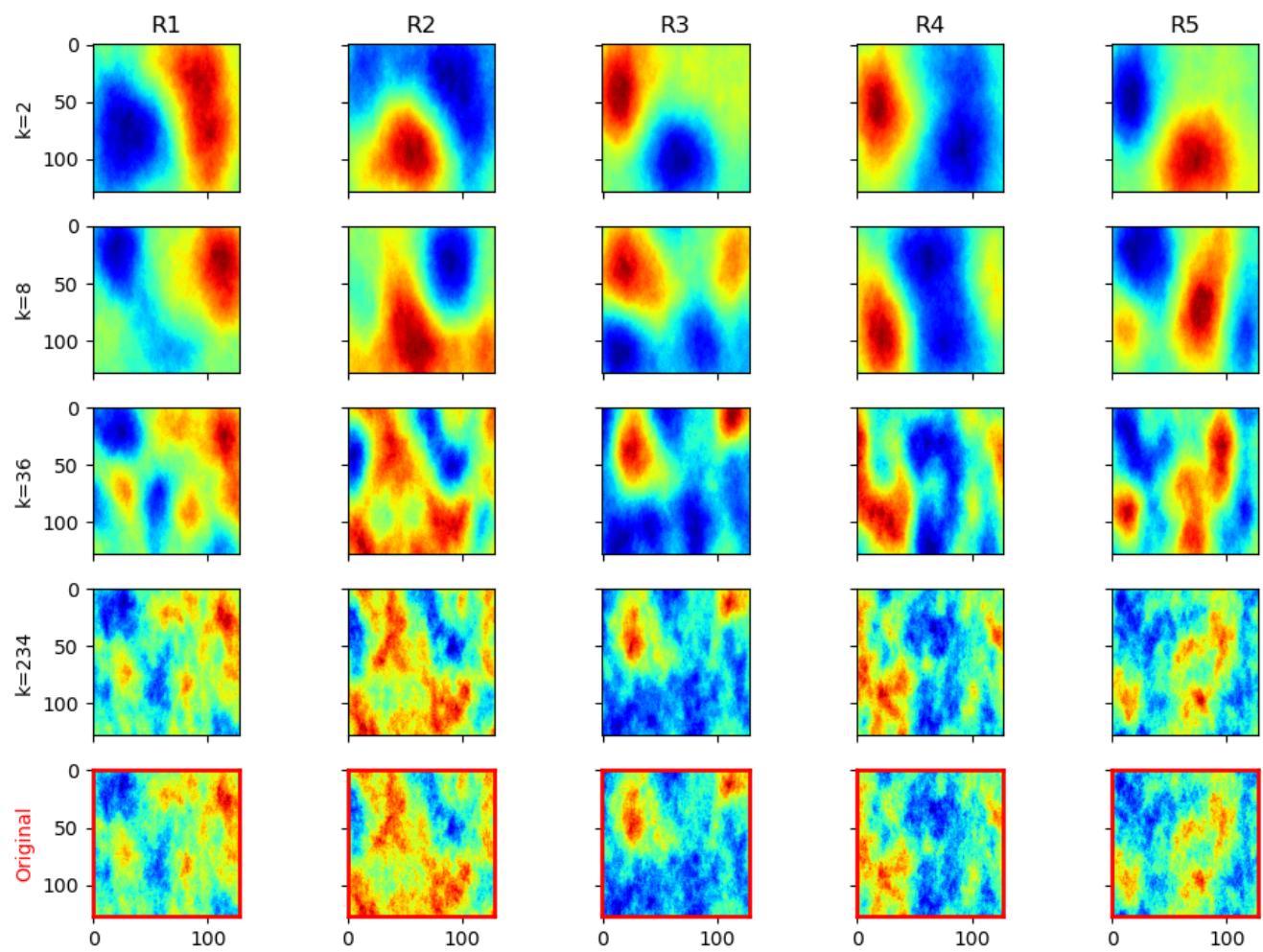


Figure 9: Reconstructed geologic models using k principal components for the first 5 realizations (R1-R5).

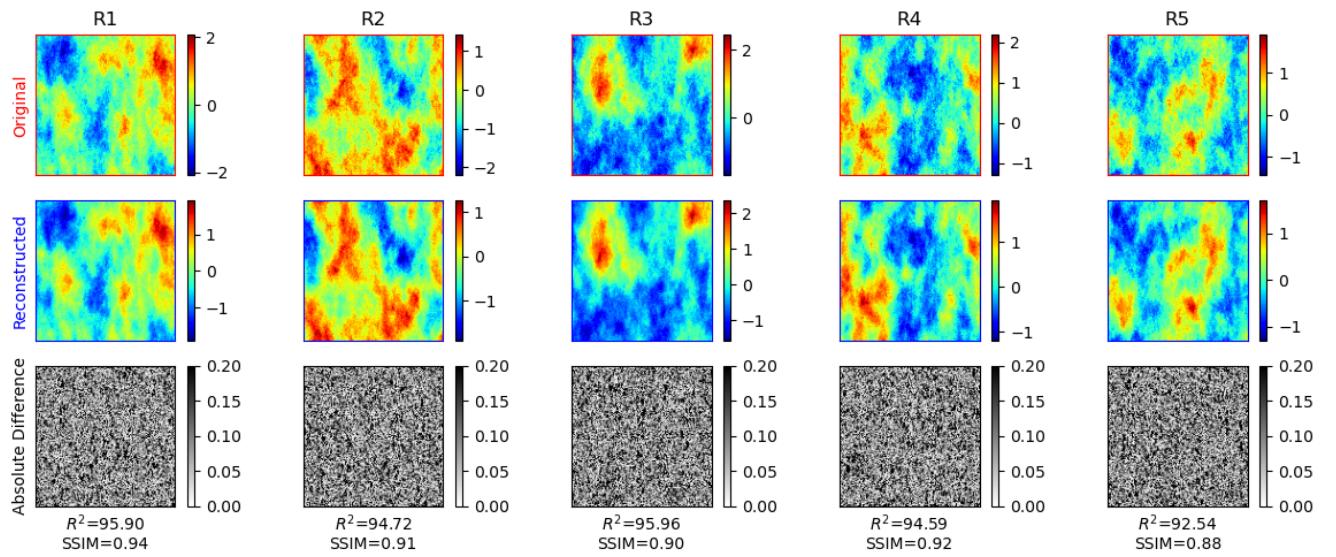


Figure 10: Reconstructed images using $k=234$ principal components accounting for 95% of variance explained for the first 5 realizations (R1-R5). The bottom row shows the absolute difference, calculated using Eq. 9.

183 Discrete Wavelet Transform

184 To explain the discrete wavelet transform, one must first take a look at the Fourier transform.
 185 The Fourier transform is a central topic in physics and engineering, involving a transformation
 186 of equations into a simple basis to simplify and decouple equations and make computations and
 187 analysis more efficient [40, 41]. The main idea behind the Fourier transform is to decompose a
 188 signal into a set of sine and cosine functions with increasing frequencies to provide an orthogonal
 189 basis for the space of solutions to an equation [42]. Unlike SVD and PCA, where the set of
 190 orthogonal bases are *tailored* to the data, the Fourier transform provides a *generic* basis (sines
 191 and cosines) to parameterize any data space into frequency and phase [30].

192 Mathematically, the Fourier transform is expressed as

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} [a_k \cos\left(\frac{k\pi x}{L}\right) + b_k \sin\left(\frac{k\pi x}{L}\right)] = \sum_{k=-\infty}^{\infty} c_k e^{\frac{ik\pi x}{L}}, \quad (15)$$

193 where a_k and b_k are the sine and cosine coefficients, respectively, L is the domain range such that
 194 $x \in [-L, L]$, and k is the frequency.

195 Computationally, the Fourier transform of a data matrix can be computed using the Fast
 196 Fourier Transform [43, 44]. This classical algorithm has become ubiquitous in all fields of science
 197 and engineering, allowing for rapid image and audio compression and other applications. However,
 198 the Fourier transform suffers from localization and loss of resolution, especially in time-frequency
 199 analysis [45]. Wavelets and the Discrete Wavelet transform (DWT) become the candidate solution
 200 for more complex problems.

201 DWT is based on the Fourier transform, but extends the transformation to a more general
 202 orthogonal basis and exploits multi-resolution decompositions by enabling different time and fre-
 203 quency scales, namely the decomposition levels [46, 47]. This is particularly useful for decomposing
 204 complex measurements, especially for image and video decomposition and compression [48]. The
 205 principal idea behind DWT is to start with a generating function, $\psi(t)$, also known as the *mother*
 206 wavelet, and generate a family of scaled and translated versions of the function, such that,

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}} \psi\left(\frac{t-b}{a}\right), \quad (16)$$

207 where a and b are learnable parameters representing the scale and translation of the function ψ ,
 208 respectively. Typically, the generating wavelet is selected to be an orthogonal function to provide
 209 a hierarchical basis. Therefore, any dataset or function $X = f(t)$ can be described as

$$X = \sum_{k=-\infty}^{\infty} a_k \phi(t) + \sum_{k=-\infty}^{\infty} \sum_{j=0}^{\infty} b_{j,k} \psi(t), \quad (17)$$

210 where ϕ is a scaling function, ψ is the general wavelet, and a and b are the scaling and translation
 211 parameters, respectively.

212 For a 2D image, a level 1 DWT decomposition divides the data matrix into two discrete
 213 components, namely the approximation and details. A filter bank, or cascading set of high-
 214 pass and low-pass filters, is applied to the data matrix such that the first set obtains a low-pass
 215 horizontal filter (L) and a high-pass horizontal filter (H). Then each sub-image is filtered vertically
 216 using the low-pass and high-pass filter to obtain LL , HL , LH , and HH , respectively, also referred
 217 to as the approximate (A), horizontal (H), vertical (V), and diagonal (D) coefficients. The level
 218 1 DWT coefficient matrix can then be represented as,

$$X = \begin{bmatrix} A & H \\ V & D \end{bmatrix}. \quad (18)$$

219 Example

220 The first step to perform DWT is to select hyperparameters, namely the mother wavelet and
 221 the number of decomposition levels. Here, we will use only 1 decomposition level and the Haar
 222 wavelet, which provides the orthogonal basis needed for the decomposition. Since DWT provides
 223 a multi-resolution orthogonal basis in two-dimensional space, there is no need to vectorize the
 224 data matrix $X \in \mathbb{R}^{500 \times 128 \times 128}$. Figure 11 shows the coefficients in the DWT decomposition for the
 225 first few realizations of our geologic uncertainty model dataset. Figure 12 shows the structural
 226 similarity index measure ($SSIM$) of the reconstructed versus true images in the ensemble as a
 227 function of the percent of DWT coefficients retained.

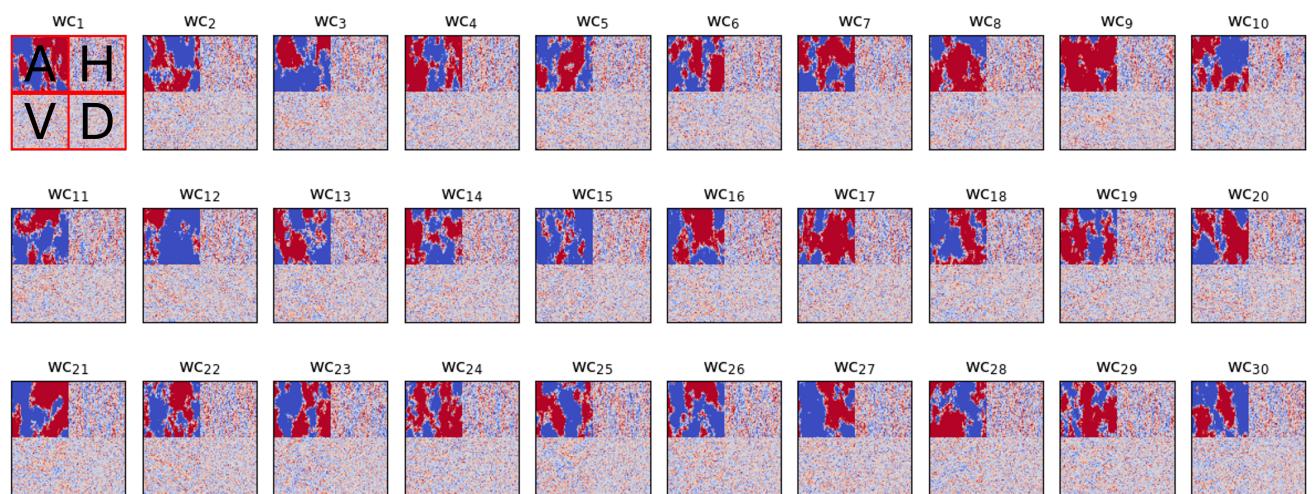


Figure 11: DWT basis, or wavelet coefficients (WC_1-WC_{30}), for geologic uncertainty dataset. The top left quadrant represents the approximate (A) coefficients, the top right quadrant represents the horizontal coefficients (H), the bottom left quadrant represents the vertical (V) coefficients, and the bottom right quadrant represents the diagonal coefficients (D), as expressed in Eq. 18.

228 To obtain a latent representation of reduced-dimensions from the DWT projection, we must
 229 truncate the trailing coefficients based on their magnitudes. We observe that by retaining only 5%
 230 of the DWT coefficients, we obtain a reconstruction with $SSIM = 46.2$, while retaining 20% of the
 231 DWT coefficients already provides a reconstruction with $SSIM = 94.7$, and retaining 50% of the
 232 DWT coefficients provides an almost lossless reconstruction with $SSIM = 99.36$. Figure 13 shows
 233 the reconstructed images from the geologic uncertainty ensemble using 5%, 20%, 50% and 80%
 234 of DWT coefficients, while Figure 14 shows that using only 50% of the DWT coefficients, we are
 235 able to obtain an accurate reconstruction with $SSIM = 99.36$, $R^2 = 99.26$, and $MSE = 0.027$.
 236 The code for the DWT example can be found in [Appendix C](#).

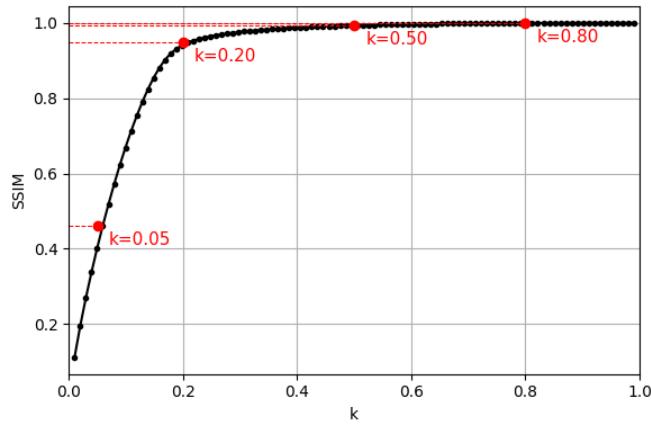


Figure 12: SSIM by number of DWT coefficients retained.

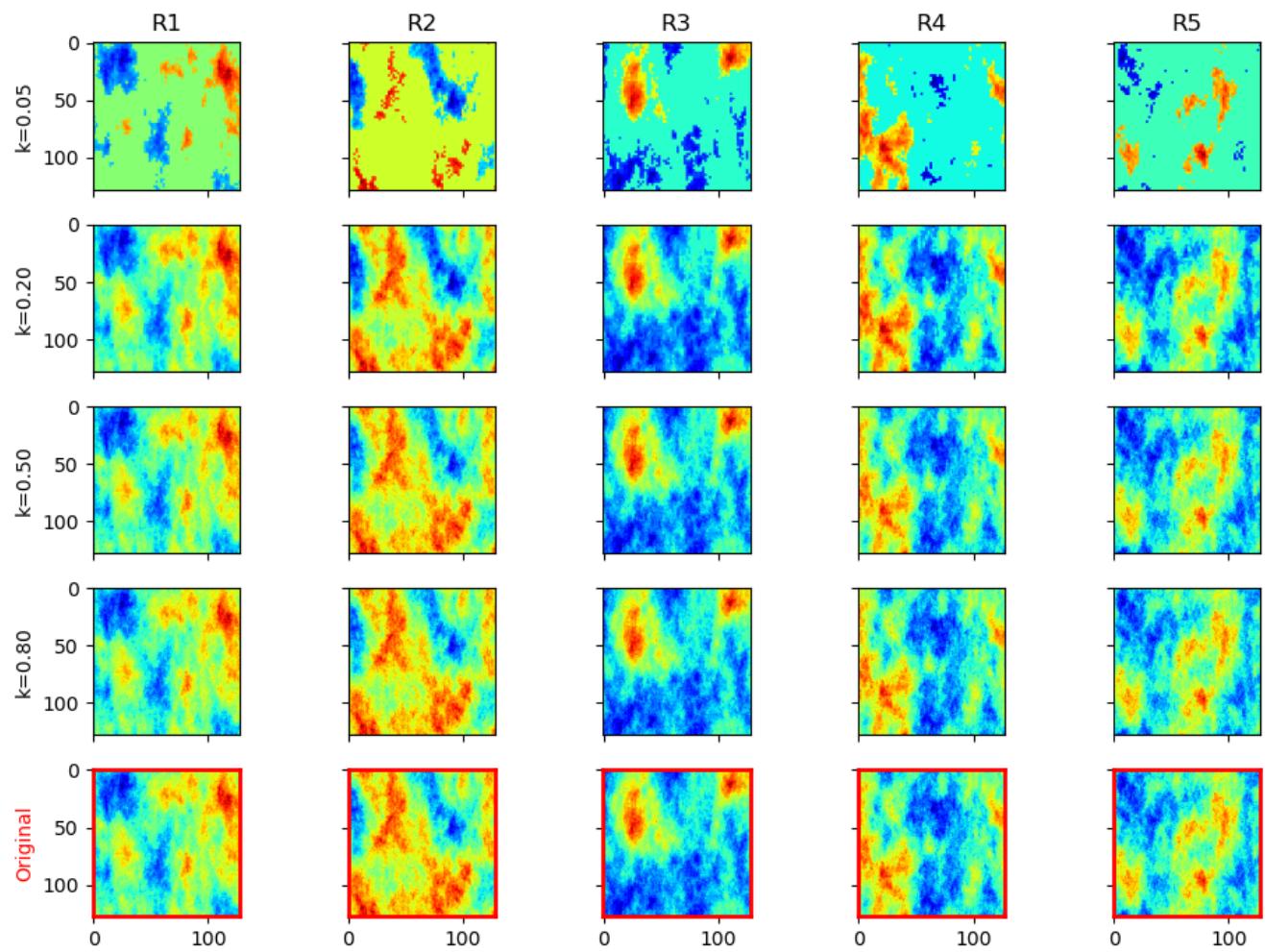


Figure 13: Reconstructed geologic models by retaining $k=\{5, 20, 50, 80\}\%$ of DWT coefficients for the first 5 realizations (R1-R5).

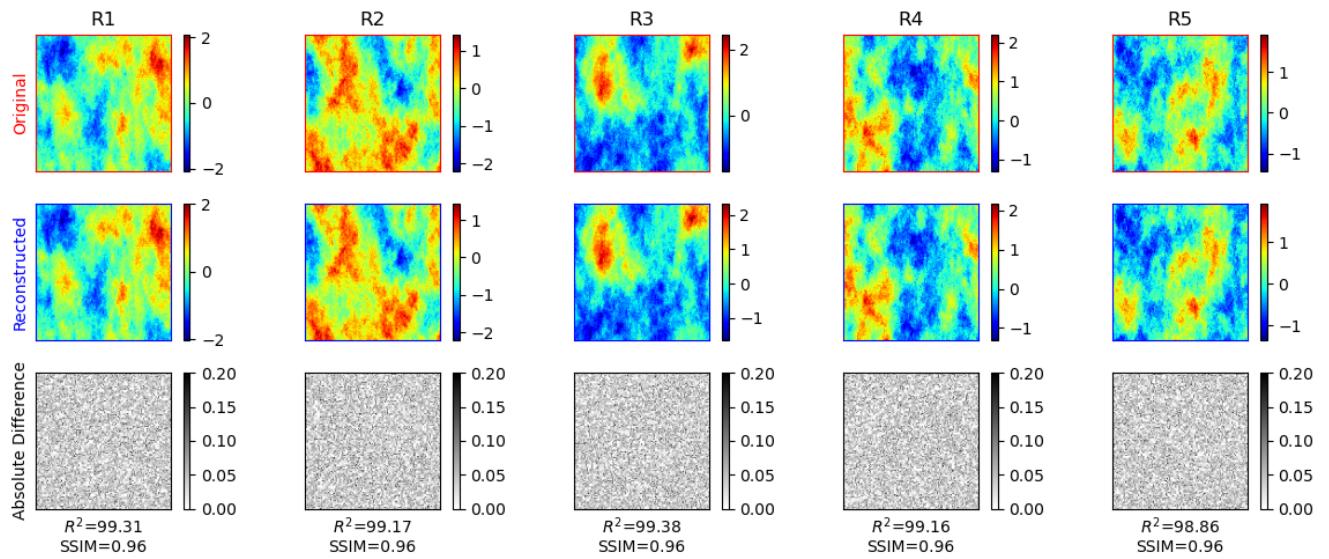


Figure 14: Reconstructed images using $k=50\%$ DWT coefficients accounting for 99% SSIM for the first 5 realizations (R1-R5). The bottom row shows the absolute difference, calculated using Eq. 9.

²³⁷ Dictionary Learning

²³⁸ Dictionary Learning (DL) was originally developed for the field of natural language processing,
²³⁹ and it can be interpreted as a dimensionality reduction method that aims to find a minimal
²⁴⁰ “dictionary” that contain the necessary information, or “words”, to reconstruct a given set of
²⁴¹ sentences. Therefore, DL is a sparsity-promoting dimensionality reduction method that aims
²⁴² to find a sparse representation of the data matrix from a linear combination of basic elements
²⁴³ [49, 50]. The basic elements are known as *atoms* or *words*, and the set of atoms is referred to as
²⁴⁴ the *dictionary*. A *coder*, or sparse pointer matrix, defines how to linearly combine the atoms in the
²⁴⁵ dictionary in order to obtain a reconstruction of the original dataset. The set of atoms need not to
²⁴⁶ be orthogonal, as the *dictionary* of atoms can be overcomplete (more atoms than realizations) or
²⁴⁷ undercomplete (less atoms than realizations). For dimensionality reduction purposes, we will focus
²⁴⁸ on undercomplete dictionaries where the data signal can be represented with a linear combination
²⁴⁹ of a limited number of atoms.

²⁵⁰ Similar to PCA and SVD, dictionary learning provides a *tailored* basis for the data, where
²⁵¹ the atoms are not generic, but rather, learned from the samples directly. DL takes advantage of
²⁵² the fact that images can often be represented as sparse signals, meaning that a set of images can
²⁵³ be represented as a linear combination of basic images, namely the atoms. The atoms can be
²⁵⁴ members of the ensemble or new realizations that combine features from the general population
²⁵⁵ but still contain relevant information for reconstruction. The data matrix, X , is approximated by
²⁵⁶ the dictionary, D , and the sparse code, S , such that,

$$X \approx DS. \quad (19)$$

²⁵⁷ Because the dictionary is not a unique parameterization, this becomes a minimization problem
²⁵⁸ such that

$$\begin{aligned} & \text{minimize} && \|X - DS\|_F^2 \\ & \text{subject to} && \|s_i\|_0 \leq K, \\ & && \|d_i\|_2^2 \leq 1 \end{aligned} \quad (20)$$

²⁵⁹ where K is the sparsity level, d_i are the atoms in the dictionary, and $\|\cdot\|_F$ is the Frobenius norm.
²⁶⁰ However, the ℓ_0 -norm is a non-convex and discontinuous function, making the problem *NP-hard*
²⁶¹ and intractable. Therefore, the solution is obtained using a relaxation term, or regularization,
²⁶² such that the minimization becomes,

$$\min_{D,S} \sum_{i=1}^K \|X - DS\|_F^2 + \lambda \|s_i\|_0. \quad (21)$$

²⁶³ This formulation still leads to a sparsity-promoting solution using a compressed sensing technique,
²⁶⁴ given that S is sufficiently sparse, D is orthonormal, and X contains sufficient measurements

[51, 52]. The compressed sensing solution of the underdetermined problem now forces D to be orthonormal under the optimal sparsity constraint. The choice of the regularization hyperparameter, λ , must also be carefully considered.

The most common approach to solve this minimization problem is the k -SVD algorithm, which is a generalization of the k -Means algorithm with iterative SVD to update the atoms of dictionary. This two-step solution aims to find the optimal sparse coding, S , followed by updating the dictionary, D , to encode each element in the data matrix by a linear combination of not more than K atoms.

Example

Dictionary Learning (DL) can be efficiently applied for dimensionality reduction of image ensembles such as our geologic uncertainty example. First, we construct a complete dictionary (500 atoms) to observe the sparse representations obtained by DL, as shown in Figure 15. However, since most images can be represented as a linear combination of sparse signals, we can construct an undercomplete dictionary to parameterize the data. The atoms in the dictionary are not necessarily ordered by magnitude or energy preserved, but are altogether a sparse representation of the data matrix. Figure 16 shows $SSIM$ of the reconstructed data matrix against the number of atoms in the dictionary used to reconstruct it. We observe that $k = 191$ atoms provide an $SSIM = 0.50$, while using $k = 426$ and $k = 491$ atoms gives $SSIM = 0.8$ and $SSIM = 0.95$, respectively.

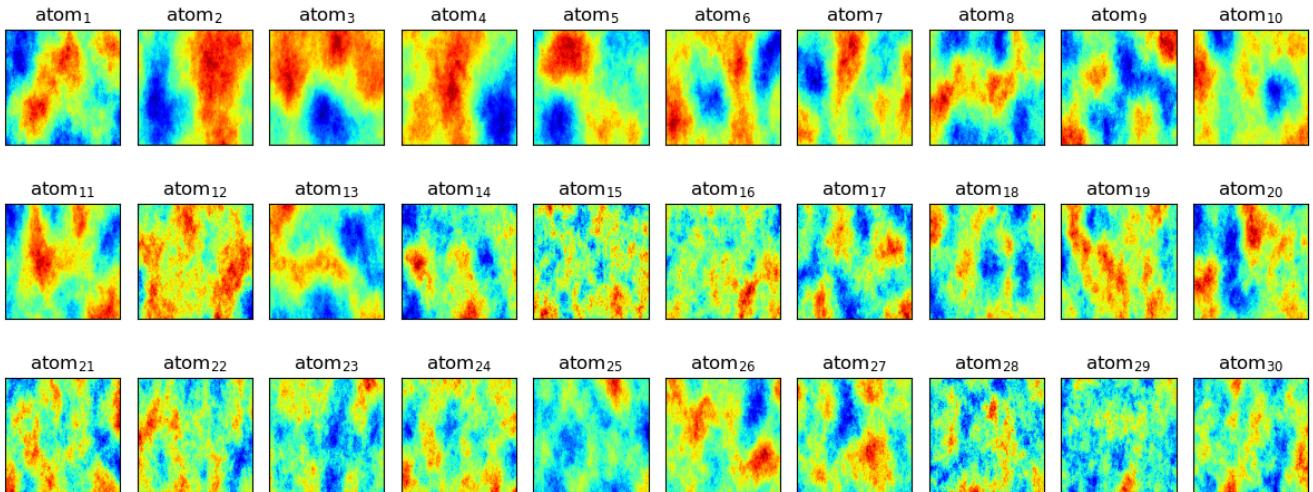


Figure 15: Dictionary atoms or basic images ($atom_1$ - $atom_{30}$) for geologic uncertainty dataset.

Figure 17 shows the reconstructed image using $k = 16$, $k = 191$, $k = 426$, and $k = 491$ atoms, representing a reconstruction $SSIM$ of 0.20, 0.50, 0.80, and 0.95, respectively. We observe

286 that using only 16 atoms, most reconstructions are not able to capture the patterns in the first
 287 few realizations of the ensemble. However, with 191 atoms, the main patterns in the images are
 288 reconstructed, and with 426 atoms we have almost lossless reconstructions. Figure 18 shows the
 289 true and reconstructed samples using $k = 426$ atoms with an average $SSIM = 0.80$ and $R^2 = 0.97$.
 290 The code for the DL example can be found in [Appendix D](#).

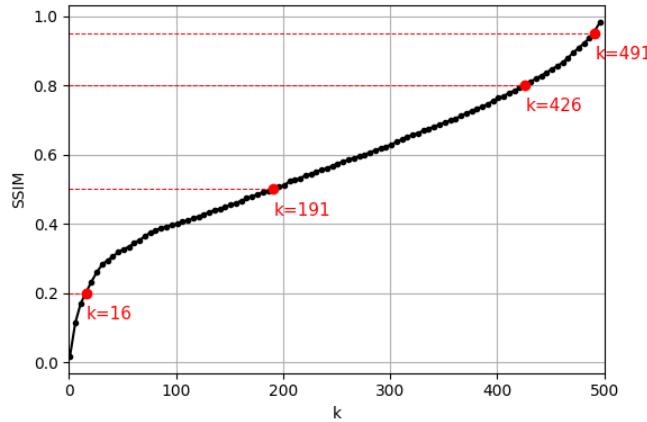


Figure 16: SSIM by number of dictionary atoms retained.

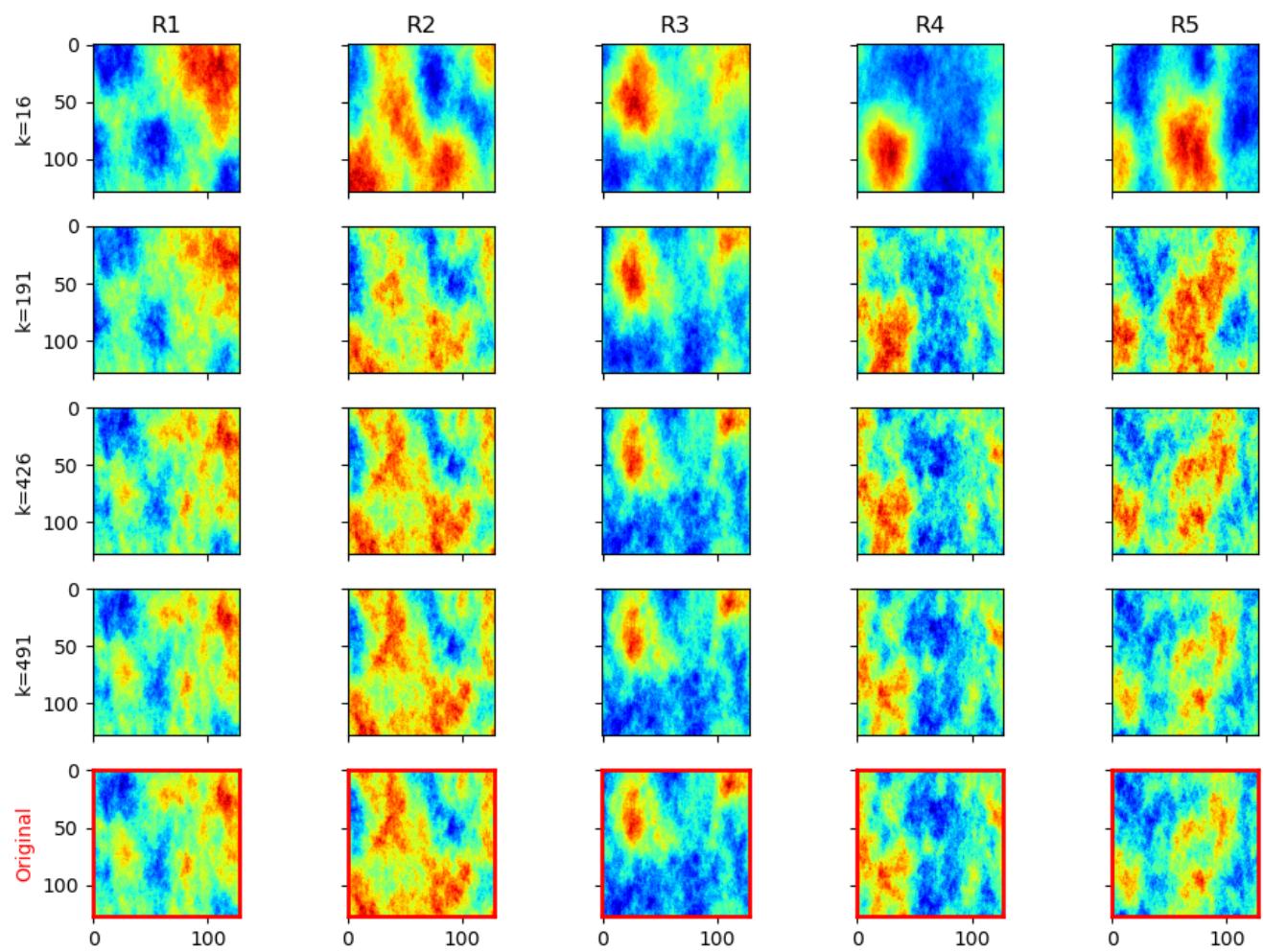


Figure 17: Reconstructed geologic models by retaining k dictionary atoms for the first 5 realizations (R1-R5).

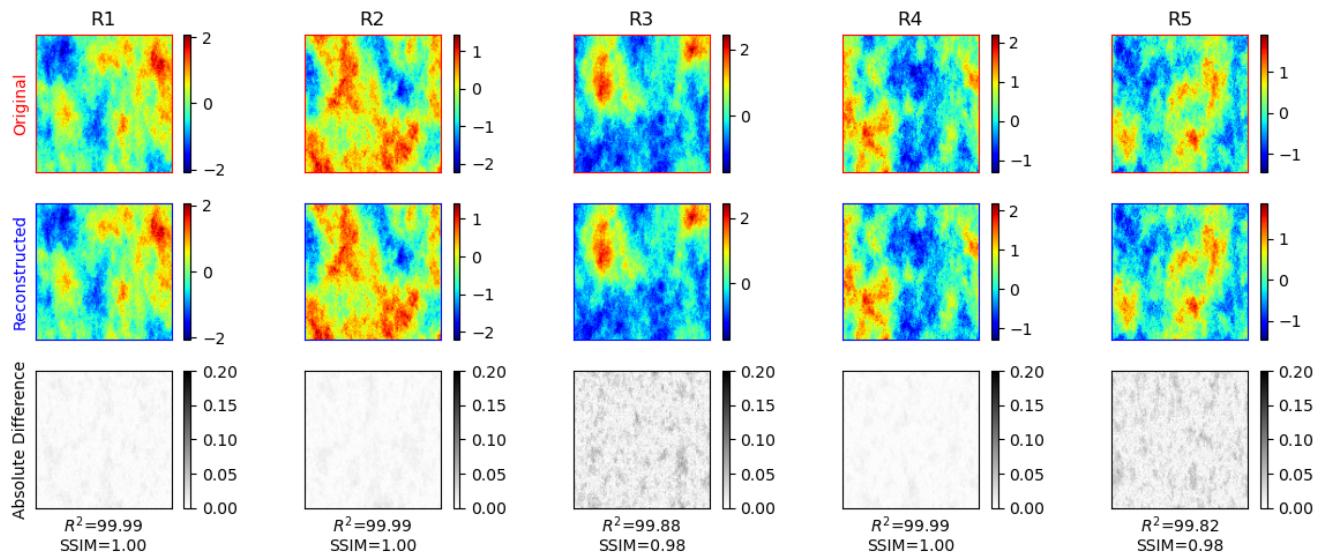


Figure 18: Reconstructed images using $k=426$ dictionary atoms accounting for 80% SSIM for the first 5 realizations (R1-R5). The bottom row shows the absolute difference, calculated using Eq. 9.

291 AutoEncoders

292 Over the last decade, neural networks have gained significant popularity in all applications of
 293 science and engineering [53, 54]. Advances in computational power and data availability make
 294 neural networks candidate methods for great many types of complex problems including computer
 295 vision. In general, neural networks can be posed as an optimization function over a functional
 296 composition such that,

$$\operatorname{argmin}_{A_j} f_M(A_m, \dots, f_2(A_2, f_1(A_1, x)) \dots) + \lambda g(A_j), \quad (22)$$

297 where A_k represents the matrix of weights between the layers k and $k + 1$ of the network, f is
 298 a nonlinear operator, and $g(\cdot)$ and λ are a regularization function and weight, respectively. The
 299 optimization problem quickly becomes a severely ill-posed problem due to the large numbers of
 300 parameters, A_k , and is typically solved by stochastic gradient descent algorithms and backpropa-
 301 gation [55].

302 Convolutional Neural Networks (CNNs) are a specialized type of neural network architecture
 303 designed for processing data that has a known grid-like topology, such as images [56]. Furthermore,
 304 they provide inherent regularization properties through the extraction of smaller pixel subsets
 305 and simpler patterns from images. The convolution operator extracts features from overlapping
 306 receptive fields over a hierarchy, such as the process of human vision [57]. This property exploits
 307 locally correlated patterns while minimizing the correlations at large distances. The convolutional
 308 filters have properties of translational equivariance and local shift invariance, where the learned
 309 weights in the filters are optimized to extract dominant patterns and structures in the data.

310 Convolutional AutoEncoders (AEs) are a special architecture of neural networks used for image
 311 compression, denoising, and translation [58, 59]. The main idea behind AEs is to compress the
 312 original data matrix, X , into a latent representation of reduced dimensions, z , through an *encoder*
 313 portion of the architecture, and then use a mirror image of the Encoder, namely the *decoder*, to
 314 reconstruct the data matrix, X' , such that,

$$X \approx X' = dec(enc(X)) = dec(z). \quad (23)$$

315 The Encoder and Decoder portions of the network are composed of multiple convolutional layers,
 316 where each layer includes a pooling (decreased dimensions) or upsampling (increased dimensions)
 317 operator for the case of the Encoder and Decoder, respectively. The latent representation, also
 318 known as latent space, z , must be able to capture the majority of the patterns and structures in
 319 the data while maximally reducing the dimensions.

320 One of the main issues when designing AEs for dimensionality reduction is the vast number
 321 of hyperparameters that must be considered [60]. The number and size of each hidden layer,
 322 convolutional filter (kernel) size, stride, padding, pooling and upsampling rates, normalization,
 323 nonlinear activation function, optimizer, learning rate, number of training epochs, loss function,
 324 and other hyperparameters must be carefully considered and tuned to obtain the best possible
 325 AE for a given dataset.

326 **Example**

327 Given the architectural complexity of designing neural networks for dimensionality reduction,
 328 we will focus on a simple convolutional AutoEncoder architecture to demonstrate its use on our
 329 geologic uncertainty model. The first step is to expand the dimensions of our dataset such that
 330 $X \in \mathbb{R}^{500 \times 1 \times 128 \times 128}$, where the new dimension represents the channel dimensions. We design an
 331 AE with three layers in the Encoder, and three mirroring layers in the Decoder, each with a
 332 convolution, batch normalization, and rectified linear unit (ReLU) activation. In the Encoder,
 333 maximum pooling is used to decrease the dimensions of the data, such that,

$$X_j \in \mathbb{R}^{b \times c_j \times \frac{n_x^j}{2} \times \frac{n_y^j}{2}} \quad (24)$$

334 where b represents the batch size and c represents the number of channels. On the other hand,
 335 the Decoder uses an upsampling operator to increase the dimensions of the data such that,

$$X'_j \in \mathbb{R}^{b \times c_j \times 2n_x^j \times 2n_y^j}. \quad (25)$$

336 The number of channels for each of the three Encoder and Decoder layers is selected as 16, 64,
 337 and 256, respectively, as it is traditional to select the number of channels in increments of 2^n .
 338 Therefore, after 3 encoding layers which decrease the dimensions of the data in half each time,
 339 the latent space is given by,

$$z \in \mathbb{R}^{b \times 256 \times 16 \times 16}. \quad (26)$$

340 We use the Adam optimizer [61] with learning rate 0.001 and MSE loss function for 200 epochs, and
 341 train the model using an NVIDIA RTX 6000 Ada GPU. The total number of trainable parameters
 342 in the model is 314,883, and the reconstruction metrics are $R^2 = 92.35$, $MSE = 0.029$, and
 343 $SSIM = 89.75$. Figure 19 shows the loss function per epoch, demonstrating significant stability
 344 and minimal overfitting between the training and validation sets. We also note from Figure 19 that
 345 the loss function stabilizes significantly after approximately 50 epochs, meaning we could decrease
 346 the number of epochs or apply an early stopping criterion to help accelerate the training time.
 347 Finally, Figure 20 shows the reconstructed images in the geologic uncertainty ensemble using our
 348 AE. We observe that the convolutional AE smooths the data, which is a common concern when
 349 using this architecture. However, all large-scale features and the majority of the fine-scale details
 350 are preserved and reconstructed using this architecture. The code for the AE example can be
 351 found in [Appendix E](#).

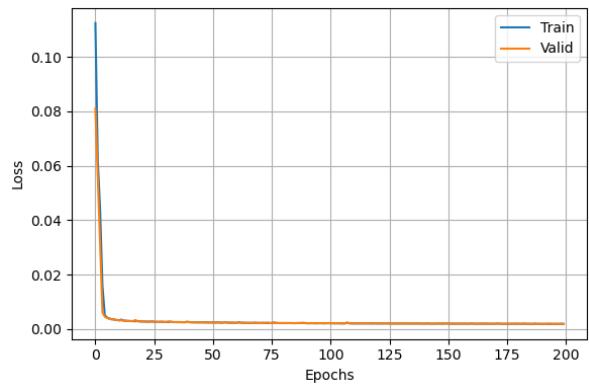


Figure 19: Loss function versus number of epochs.

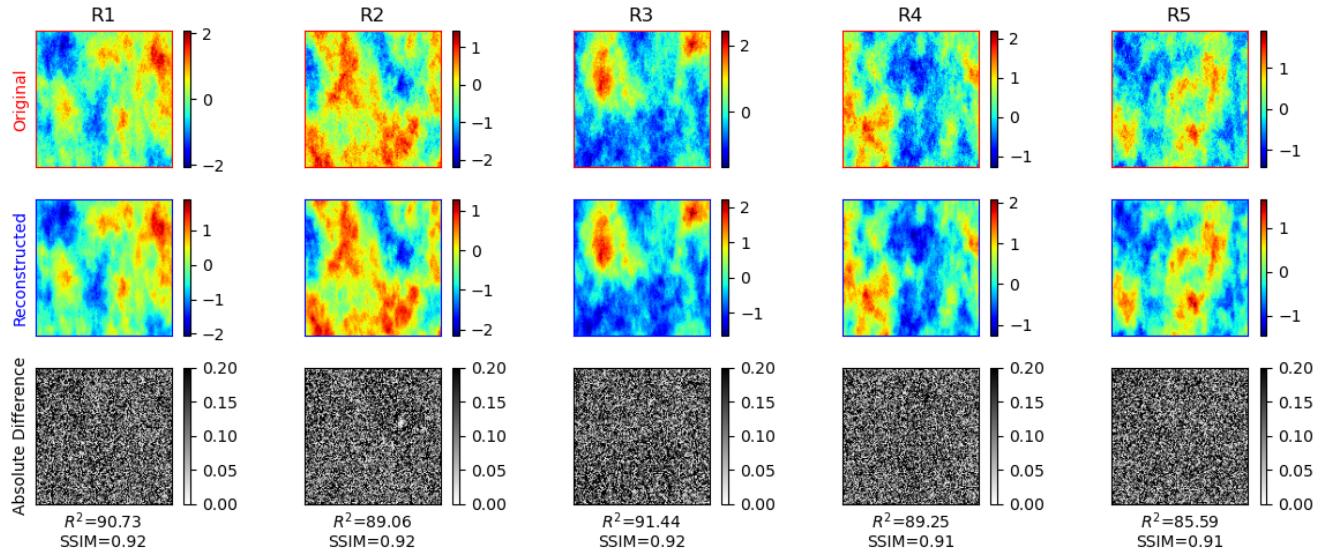


Figure 20: Reconstructed image using latent dimension 256 in the AutoEncoder for the first 5 realizations (R1-R5). The bottom row shows the absolute difference, calculated using Eq. 9.

352 Conclusions

353 Dimensionality reduction techniques serve as powerful algorithms to project high-dimensional,
354 complex datasets into lower dimensionality space while attempting to minimize the loss of infor-
355 mation by extracting relevant patterns and features in the data. The latent representation, z , is
356 designed to contain the most salient features of the data, and can be used as a proxy for complex
357 and time-consuming routines such as reservoir simulation and history matching in the case of
358 subsurface energy resource engineering.

359 Several dimensionality reduction algorithms were shown, namely singular value decomposition,
360 principal component analysis, discrete wavelet transform, dictionary learning, and deep learning
361 AutoEncoders. However, a much longer list of dimensionality reduction algorithms exists and
362 can be useful for different types of datasets and applications. Table 1 shows a comparison of the
363 different dimensionality reduction techniques applied to the geologic uncertainty model dataset in
364 terms of reconstruction accuracy and computational costs. Ultimately, the goal here is to obtain a
365 compressed representation of the data that can be used to accelerate expensive computations, and
366 the user must carefully select and design the dimensionality reduction algorithm according to the
367 data needs. The tradeoff between compression and reconstruction accuracy is typically the main
368 concern, as perfectly lossless compression is hard to obtain. Most engineering applications can
369 benefit from slightly lossy compression at the benefit of significant computational acceleration.

	20% CR	50% CR	80% CR	95% CR	Time [s]
SVD	11	83	288	441	21.1
PCA	2	8	36	234	2.6
DWT	7	30	60	125	1.3
DL	16	191	426	491	94.6

Table 1: Latent space dimension based on reconstruction accuracy. Four different dimensionality reduction techniques (SVD, PCA, DWT, DL) are compared based on different compression ratios (CR) (*coefficients retained / total number of features*) and their average CPU time required (in seconds) for compression and reconstruction.

³⁷⁰ **Code Availability**

³⁷¹ The code is publicly available on the authors GitHub repository:
³⁷² <https://github.com/misaelmmorales/Dimensionality-Reduction>

³⁷³ **Acknowledgments**

³⁷⁴ The authors thank the Formation Evaluation (FE) and Digital Reservoir Characterization Tech-
³⁷⁵ nology (DIRECT) Industry Affiliate Programs at the University of Texas at Austin for supporting
³⁷⁶ this work.

³⁷⁷ **Declaration of competing interest**

³⁷⁸ The authors declare that they have no known competing financial interests or personal relation-
³⁷⁹ ships that could have appeared to influence the work reported in this paper.

380 **Appendix A Singular Value Decomposition (SVD) Code**

```
1 import numpy as np                                # arrays operations
2 from scipy.linalg import svd                      # SVD algorithm
3 from sklearn.metrics import r2_score              # reconstruction metric
4 from skimage.metrics import structural_similarity # reconstruction metric
5
6 df_perm = np.load('data/data_500_128x128.npy')    # (500, 128, 128)
7 df_perm_f = np.reshape(df_perm, (df_perm.shape[0], -1)) # (500, 16384)
8
9 u, s, vt = svd(df_perm_f.T, full_matrices=True)   # SVD decomposition of X
10 ss = np.diag(s)                                    # diagonalize S
11
12 cutoffs = [0.2, 0.5, 0.8, 0.95]                  # SV energy cutoff values
13 energy = np.cumsum(np.diag(ss))/np.sum(np.diag(ss)) # calculate energy
14 nk = [np.argwhere(energy > cutoff)[0][0] for cutoff in cutoffs] # find truncation indices
15 dd = [u[:, :n] @ ss[:n, :n] @ vt[:n, :] for n in nk] # truncate based on energy
16 reconstructions = [np.moveaxis(np.reshape(d, (128, 128, -1)), -1, 0) for d in dd]
```

381 **Appendix B Principal Component Analysis (PCA) Code**

```
1 import numpy as np                                     # array operations
2 from sklearn.decomposition import PCA                 # PCA algorithm
3 from sklearn.metrics import r2_score                  # reconstruction metric
4 from skimage.metrics import structural_similarity     # reconstruction metric
5
6 df_perm = np.load('data/data_500_128x128.npy')        # (500, 128, 128)
7 df_perm_f = np.reshape(df_perm, (df_perm.shape[0], -1)) # (500, 16384)
8
9 pca = PCA(n_components=500, svd_solver='full')          # initialize PCA
10 pca.fit(df_perm_f.T)                                 # fit PCA to data
11 z = pca.transform(df_perm_f.T)                         # extract PCA latent space
12
13 cutoffs = [0.2, 0.5, 0.8, 0.95]                      # variance explained cutoffs
14 energy = np.cumsum(pca.explained_variance_ratio_)      # calculate energy
15 nk = [np.argwhere(energy > cutoff)[0][0] for cutoff in cutoffs] # find truncation indices
16
17 # save latent space and reconstructions for each PCA decomposition level
18 reconstructions = []
19 for i, k in enumerate(nk):
20     pca = PCA(n_components=k)                          # initialize PCA
21     pca.fit(df_perm_f.T)                            # fit PCA to data
22     z = pca.transform(df_perm_f.T)                  # extract PCA latent space
23     xhat = pca.inverse_transform(z)                # inverse PCA reconstruction
24     r = np.moveaxis(np.reshape(xhat, (128, 128, -1)), -1, 0) # reshape reconstruction
25     reconstructions.append(r)                      # save reconstructions
```

382 **Appendix C Discrete Wavelet Transform (DWT) Code**

```
1 import numpy as np                                # array operations
2 from pywt import wavedec2, waverec2               # DWT operations
3 from pywt import coeffs_to_array, array_to_coeffs # DWT coefficients-to-arrays
4 from skimage.metrics import structural_similarity # reconstruction metrics
5 from skimage.metrics import peak_signal_noise_ratio # reconstruction metrics
6 from skimage.metrics import mean_squared_error    # reconstruction metrics
7
8 df_perm = np.load('data/data_500_128x128.npy')      # (500, 128, 128)
9
10 keep_percs = np.linspace(0.01, 0.99, 100)           # energy cutoffs
11 wavelet = 'haar'                                    # generating wavelet
12 levels = 1                                         # decomposition level
13
14 reconstructions = {'xhat':[], 'coeffs':[]}          # DWT coefficients
15 for i, k in enumerate(keep_percs):                  # coefficients-to-array
16     c = wavedec2(df_perm, wavelet=wavelet, level=levels) # sort by magnitude
17     c_arr, c_slices = coeffs_to_array(c)
18     c_sort = np.sort(np.abs(c_arr.reshape(-1)))
19
20     threshold = c_sort[int(np.floor((1-k)*len(c_sort)))] # cutoff indices
21     c_arr_t = c_arr * (np.abs(c_arr) > threshold)        # truncate coefficients
22     c_t = array_to_coeffs(c_arr_t, c_slices)              # array-to-coefficients
23     recs = waverec2(c_t, wavelet=wavelet)                 # reconstructed images
24
25     reconstructions['xhat'].append(recs)                  # store reconstructions
26     reconstructions['coeffs'].append(c_arr)                # store coefficients
```

383 **Appendix D Dictionary Learning (DL) Code**

```
1 import numpy as np                                # array operations
2 from sklearn.decomposition import DictionaryLearning # dictionary learning
3 from sklearn.decomposition import SparseCoder       # sparse coding
4
5 df_perm = np.load('data/data_500_128x128.npy')      # (500, 128, 128)
6 df_perm_f = np.reshape(df_perm, (df_perm.shape[0], -1)) # (500, 16384)
7
8 n_atoms = 500                                     # atoms in dictionary
9 dl = DictionaryLearning(n_components=n_atoms)      # initialize dictionray
10 dictionary = dl.fit(df_perm_f)                   # fit to data matrix
11 atoms = dictionary.components_                  # extract atoms
12
13 sparse_code = SparseCoder(atoms).fit_transform(df_perm_f) # train sparse coder
14 sparse_recs = np.reshape(sparse_code @ atoms, df_perm_f.shape) # reconstructed images
```

384 Appendix E AutoEncoder (AE) Code

```

1 import numpy as np                                # array operations
2 import torch                                     # deep learning library
3 import torch.nn as nn                            # neural network operations
4 import torch.optim as optim                      # neural network optimizer
5 import torch.nn.functional as F                  # functional layers
6 from torch.utils.data import DataLoader, TensorDataset
7 from torch.utils.data import random_split
8 from sklearn.preprocessing import MinMaxScaler   # tensor data opeartions
9
10 df_perm = np.load('data/data_500_128x128.npy')    # random train-test split
11 df_perm_f = np.reshape(df_perm, (df_perm.shape[0], -1)) # data preprocessing
12
13 class AutoEncoder(nn.Module):                    # AutoEncoder architecture class
14     def __init__(self, layers=[4, 16, 64]):        # size of 3 hidden layers
15         super(AutoEncoder, self).__init__()
16         self.encoder = nn.Sequential(               # ENCODER
17             self.conv_block(1, layers[0]),           # first encoding layer
18             self.conv_block(layers[0], layers[1]),  # second encoding layer
19             self.conv_block(layers[1], layers[2]),  # third encoding layer
20         )
21         self.decoder = nn.Sequential(              # DECODER
22             self.deconv_block(layers[2], layers[1]), # first decoding layer
23             self.deconv_block(layers[1], layers[0]), # second decoding layer
24             self.deconv_block(layers[0], 1),        # third decoding layer
25         )
26
27     def conv_block(self, ic, oc):                # architecture of encoding layer
28         return nn.Sequential(                      # convolution
29             nn.Conv2d(ic, oc, kernel_size=3, padding=1), # batch normalization
30             nn.BatchNorm2d(out_channels),            # activation
31             nn.ReLU(),                            # pooling
32             nn.MaxPool2d(kernel_size=2, stride=2)
33         )
34
35     def deconv_block(self, ic, oc):                # architecture of decoding layer
36         return nn.Sequential(                      # upsampling
37             nn.Upsample(scale_factor=2),            # convolution
38             nn.Conv2d(ic, oc, kernel_size=3, padding=1),

```

```

39         nn.BatchNorm2d(oc),                      # batch normalization
40         nn.ReLU()                                # activation
41     )
42
43     def forward(self, x):                      # AutoEncoder
44         z = self.encoder(x)                     # Encoder
45         y = self.decoder(z)                    # Decoder
46         return y
47
48
49     scaler = MinMaxScaler()                   # initialize data scaler
50     scaler.fit(df_perm_f)                   # fit data scaler
51     XX = scaler.transform(df_perm_f).reshape(df_perm.shape)      # normalize data
52     X_dataset = torch.tensor(np.expand_dims(XX,1), dtype=torch.float32) # tensor dataset
53     X_train, X_valid = random_split(X_dataset, [450, 50])        # train-valid split
54     trainloader = DataLoader(X_train, batch_size=16, shuffle=True)    # training data loader
55     validloader = DataLoader(X_valid, batch_size=16, shuffle=False)   # validation data loader
56
57     device = torch.device("cuda" if torch.cuda.is_available() else "cpu") # use GPU for training
58     model = AutoEncoder(layers=[16, 64, 256]).to(device)                 # initialize NN
59     optimizer = optim.Adam(params=model.parameters(), lr=1e-3)          # initialize optimizer
60     criterion = nn.MSELoss().to(device)                                    # initialize loss
61
62     epochs = 200                                         # define number of training epochs
63     train_loss, valid_loss = [], []                         # initialize train and valid losses
64     for epoch in range(epochs):                            # begin training loop
65         # training
66         epoch_train_loss = []                           # reset epoch loss - train
67         for i, x in enumerate(trainloader):
68             x = x.to(device)                            # send batch to GPU
69             optimizer.zero_grad()                      # reset gradients
70             y = model(x)                            # forward pass
71             loss = criterion(y, x)                  # calculate loss
72             loss.backward()                          # backward pass
73             optimizer.step()                        # calculate gradients
74             epoch_train_loss.append(loss.item())    # append epoch loss
75     train_loss.append(np.mean(epoch_train_loss))       # appen total loss
76
77     # validation
78     model.eval()                                     # freeze model
79     epoch_valid_loss = []                            # reset epoch loss - valid

```

```
79     with torch.no_grad():
80         for i, x in enumerate(validloader):
81             x = x.to(device)                      # send batch to GPU
82             y = model(x)                        # forward pass
83             loss = criterion(y, x)            # calculate loss
84             epoch_valid_loss.append(loss.item()) # append epoch loss
85             valid_loss.append(np.mean(epoch_valid_loss)) # append total loss
86
87 # Obtain final predictions for dataset and back-normalize
88 pred = model(X_dataset.to(device)).cpu().detach().numpy().squeeze()
89 xhat = scaler.inverse_transform(pred.reshape(df_perm_f.shape)).reshape(df_perm.shape)
```

References

- [1] Mehdi Mohammadpoor and Farshid Torabi. Big data analytics in oil and gas industry: An emerging trend. *Petroleum*, 6(4):321–328, 2020.
- [2] Abdeldjalil Latrach, Mohamed L Malki, Misael Morales, Mohamed Mehana, and Minou Rabie. A critical review of physics-informed machine learning applications in subsurface energy systems. *Geoenergy Science and Engineering*, page 212938, 2024.
- [3] Oriyomi Raheem, Wen Pan, Misael M Morales, and Carlos Torres-Verdín. Best practices in automatic permeability estimation: machine-learning methods vs. conventional petrophysical models. *Petrophysics-The SPWLA Journal of Formation Evaluation and Reservoir Description*, 65(05):789–812, 2024.
- [4] Shaowen Mao, Bailian Chen, Mohamed Malki, Fangxuan Chen, Misael Morales, Zhiwei Ma, and Mohamed Mehana. Efficient prediction of hydrogen storage performance in depleted gas reservoirs using machine learning. *Applied Energy*, 361:122914, 2024.
- [5] Misael M Morales, Oriyomi Raheem, Carlos Torres-Verdín, Michael Pyrcz, Murray Christie, and Vladimir Rabinovich. Automatic rock classification from core data to well logs: Using machine learning to accelerate potential co₂ storage site characterization. In *Fourth International Meeting for Applied Geoscience & Energy*, pages 632–636. Society of Exploration Geophysicists and American Association of Petroleum . . . , 2024.
- [6] Misael M Morales, Carlos Torres-Verdín, Michael Pyrcz, Murray Christie, and Vladimir Rabinovich. Automatic well-log baseline correction via deep learning for rapid screening of potential co₂ storage sites. In *SEG International Exposition and Annual Meeting*, pages SEG–2024. SEG, 2024.
- [7] K. Rashid, W. Bailey, B. Couët, and D. Wilkinson. An efficient procedure for expensive reservoir-simulation optimization under uncertainty. *SPE Economics and Management*, 5(4): 21–33, 2013. doi: 10.2118/167261-PA.
- [8] Javier E. Santos, Bernard Chang, Alex Gigliotti, Eric Guiltinan, Mohamed Mehana, Arvind Mohan, James McClure, Qinjun Kang, Hari Viswanathan, Nicholas Lubbers, Masa Prodanovic, and Michael Pyrcz. Learning from a big dataset of digital rock simulations. In *AGU Fall Meeting Abstracts*, volume 2021, pages H25O–1207, December 2021.
- [9] Shaowen Mao, Bailian Chen, Misael Morales, Mohamed Malki, and Mohamed Mehana. Cushion gas effects on hydrogen storage in porous rocks: Insights from reservoir simulation and deep learning. *International Journal of Hydrogen Energy*, 68:1033–1047, 2024.

- [10] Zeeshan Tariq, Murtada Saleh Aljawad, Amjad Hasan, Mobeen Murtaza, Emad Mohammed, Ammar El-Husseiny, Sulaiman A Alarifi, Mohamed Mahmoud, and Abdulazeez Abdulraheem. A systematic review of data science and machine learning applications to the oil and gas industry. *Journal of Petroleum Exploration and Production Technology*, pages 1–36, 2021.
- [11] Anirbid Sircar, Kriti Yadav, Kamakshi Rayavarapu, Namrata Bist, and Hemangi Oza. Application of machine learning and artificial intelligence in oil and gas industry. *Petroleum Research*, 6(4):379–391, 2021.
- [12] Jie Cai, Jiawei Luo, Shulin Wang, and Sheng Yang. Feature selection in machine learning: A new perspective. *Neurocomputing*, 300:70–79, 2018.
- [13] Alice Zheng and Amanda Casari. *Feature engineering for machine learning: principles and techniques for data scientists.* ” O'Reilly Media, Inc.”, 2018.
- [14] Siddharth Misra, Hao Li, and Jiabo He. *Machine learning for subsurface characterization.* Gulf Professional Publishing, 2019.
- [15] Shuvajit Bhattacharya. *A primer on machine learning in subsurface geosciences*, volume 1. Springer, 2021.
- [16] Misael M Morales, Ali Eghbali, Oriyomi Raheem, Michael J Pyrcz, and Carlos Torres-Verdín. Anisotropic resistivity estimation and uncertainty quantification from borehole triaxial electromagnetic induction measurements: Gradient-based inversion and physics-informed neural network. *Computers & Geosciences*, 196:105786, 2025.
- [17] Misael M Morales, Carlos Torres-Verdín, and Michael J Pyrcz. Stochastic pix2vid: A new spatiotemporal deep learning method for image-to-video synthesis in geologic co 2 storage prediction. *Computational Geosciences*, pages 1–22, 2024.
- [18] Su Jiang and Louis J Durlofsky. Use of multifidelity training data and transfer learning for efficient construction of subsurface flow surrogate models. *Journal of Computational Physics*, 474:111800, 2023.
- [19] Ademide O Mabadeje and Michael J Pyrcz. Evaluating the stability of deep learning latent feature spaces. *arXiv preprint arXiv:2402.11404*, 2024.
- [20] Ademide O Mabadeje and Michael J Pyrcz. Rigid transformations for stabilized lower dimensional space to support subsurface uncertainty quantification and interpretation. *Computational Geosciences*, pages 1–21, 2024.
- [21] Misael M Morales, Mohamed Mehana, Carlos Torres-Verdín, Michael J Pyrcz, and Bailian Chen. Optimal monitoring design for uncertainty quantification during geologic co2 sequestration: A machine learning approach. *Geoenergy Science and Engineering*, 244:213402, 2025.

- [22] Bailian Chen, Misael M Morales, Zhiwei Ma, Qinjun Kang, and Rajesh J Pawar. Assimilation of geophysics-derived spatial data for model calibration in geologic co₂ sequestration. *SPE Journal*, pages 1–10, 2024.
- [23] Michael J Pyrcz and Clayton V Deutsch. *Geostatistical reservoir modeling*. Oxford University Press, USA, 2014.
- [24] Michael J Pyrcz. *Applied Geostatistics in Python: a Hands-on guide with Geostatspy*. 2024. URL https://geostatsguy.github.io/GeostatsPyDemos_Book.
- [25] N Remy. *Applied geostatistics with SGeMS: A user’s guide*. Cambridge University Press, 2009.
- [26] Virginia Klema and Alan Laub. The singular value decomposition: Its computation and some applications. *IEEE Transactions on automatic control*, 25(2):164–176, 1980.
- [27] Gilbert W Stewart. On the early history of the singular value decomposition. *SIAM review*, 35(4):551–566, 1993.
- [28] Michael E Wall, Andreas Rechtsteiner, and Luis M Rocha. Singular value decomposition and principal component analysis. In *A practical approach to microarray data analysis*, pages 91–109. Springer, 2003.
- [29] Kirk Baker. Singular value decomposition tutorial. *The Ohio State University*, 24:22, 2005.
- [30] Steven L Brunton and J Nathan Kutz. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2022.
- [31] Behnam Jafarpour and Dennis B McLaughlin. Reservoir characterization with the discrete cosine transform. *SPE Journal*, 14(01):182–201, 2009.
- [32] Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.
- [33] Per Christian Hansen. Truncated singular value decomposition solutions to discrete ill-posed problems with ill-determined numerical rank. *SIAM Journal on Scientific and Statistical Computing*, 11(3):503–518, 1990.
- [34] Pallav Sarma, Louis J Durlofsky, Khalid Aziz, and Wen H Chen. A new approach to automatic history matching using kernel pca. In *SPE Reservoir Simulation Conference?*, pages SPE–106176. SPE, 2007.
- [35] Hai X Vo and Louis J Durlofsky. A new differentiable parameterization based on principal component analysis for the low-dimensional representation of complex geological models. *Mathematical Geosciences*, 46:775–813, 2014.

- [36] Michael J Pyrcz. *Applied Machine Learning in Python: a Hands-on Guide with Code*. 2024. URL https://geostatsguy.github.io/MachineLearningDemos_Book.
- [37] Ian T Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical transactions of the royal society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016.
- [38] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.
- [39] KH Esbensen and Paul Geladi. 2.02-principal component analysis: Concept, geometrical interpretation, mathematical background, algorithms, history, practice. In *Comprehensive chemometrics: Chemical and biochemical data analysis*, volume 2. Elsevier, 2020.
- [40] Pierre Duhamel and Martin Vetterli. Fast fourier transforms: a tutorial review and a state of the art. *Signal processing*, 19(4):259–299, 1990.
- [41] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.
- [42] Iorio Júnior Iorio Jr and Valéria de Magalhães Iorio. *Fourier analysis and partial differential equations*. Cambridge University Press, 2001.
- [43] James W Cooley, Peter AW Lewis, and Peter D Welch. The fast fourier transform and its applications. *IEEE Transactions on Education*, 12(1):27–34, 1969.
- [44] Henri J Nussbaumer. *The fast Fourier transform*. Springer, 1982.
- [45] G Susrutha, K Mallikarjun, M Ajay Kumar, and M Ashok. Analysis on fft and dwt transformations in image processing. In *2019 International Conference on Emerging Trends in Science and Engineering (ICESE)*, volume 1, pages 1–4. IEEE, 2019.
- [46] Christopher E Heil and David F Walnut. Continuous and discrete wavelet transforms. *SIAM review*, 31(4):628–666, 1989.
- [47] Tim Edwards. Discrete wavelet transforms: Theory and implementation. *Universidad de*, 1991:28–35, 1991.
- [48] Gheyath Othman and Diyar Qader Zeebaree. The applications of discrete wavelet transform in image processing: A review. *Journal of Soft Computing and Data Mining*, 1(2):31–43, 2020.
- [49] Ivana Tošić and Pascal Frossard. Dictionary learning. *IEEE Signal Processing Magazine*, 28(2):27–38, 2011.

- [50] Julien Mairal, Jean Ponce, Guillermo Sapiro, Andrew Zisserman, and Francis Bach. Supervised dictionary learning. *Advances in neural information processing systems*, 21, 2008.
- [51] Kenneth Kreutz-Delgado, Joseph F Murray, Bhaskar D Rao, Kjersti Engan, Te-Won Lee, and Terrence J Sejnowski. Dictionary learning algorithms for sparse representation. *Neural computation*, 15(2):349–396, 2003.
- [52] Entao Liu and Behnam Jafarpour. Learning sparse geologic dictionaries from low-rank representations of facies connectivity for flow model calibration. *Water Resources Research*, 49(10):7088–7101, 2013.
- [53] Ian Goodfellow. *Deep learning*, volume 196. MIT press, 2016.
- [54] Wengang Zhang, Xin Gu, Libin Tang, Yueping Yin, Dongsheng Liu, and Yanmei Zhang. Application of machine learning, deep learning and optimization algorithms in geoengineering and geoscience: Comprehensive review and future challenge. *Gondwana Research*, 109:1–17, 2022.
- [55] Léon Bottou et al. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nimes*, 91(8):12, 1991.
- [56] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE transactions on neural networks and learning systems*, 33(12):6999–7019, 2021.
- [57] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks. *Advances in neural information processing systems*, 29, 2016.
- [58] Jinsheng Jiang, Haoran Ren, and Meng Zhang. A convolutional autoencoder method for simultaneous seismic data reconstruction and denoising. *IEEE geoscience and remote sensing letters*, 19:1–5, 2021.
- [59] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks. *Advances in neural information processing systems*, 30, 2017.
- [60] Frank Hutter, Jörg Lücke, and Lars Schmidt-Thieme. Beyond manual tuning of hyperparameters. *KI-Künstliche Intelligenz*, 29:329–337, 2015.
- [61] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.