Applied Intelligence (2023) 53:22818–22833
https://doi.org/10.1007/s10489-023-04730-1

A dimensionality reduction approach for convolutional neural networks

Laura Meneghetti[1] · **Nicola Demo**[1] · **Gianluigi Rozza**[1]

Accepted: 23 May 2023 / Published online: 4 July 2023

Abstract
The focus of this work is on the application of classical Model Order Reduction techniques, such as Active Subspaces and Proper Orthogonal Decomposition, to Deep Neural Networks. We propose a generic methodology to reduce the number of layers in a pre-trained network by combining the aforementioned techniques for dimensionality reduction with input-output mappings, such as Polynomial Chaos Expansion and Feedforward Neural Networks. The motivation behind compressing the architecture of an existing Convolutional Neural Network arises from its usage in embedded systems with specific storage constraints. The conducted numerical tests demonstrate that the resulting reduced networks can achieve a level of accuracy comparable to the original Convolutional Neural Network being examined, while also saving memory allocation. Our primary emphasis lies in the field of image recognition, where we tested our methodology using VGG-16 and ResNet-110 architectures against three different datasets: CIFAR-10, CIFAR-100, and a custom dataset.

Keywords Deep neural networks · Active subspaces · Proper orthogonal decomposition · Neural network reduction

1 Introduction and motivations

Neural networks are a widespread machine learning technique, increasingly employed in various fields such as computer vision [1–3], natural language processing [4, 5], robotics [6, 7], and speech recognition [8, 9]. The accuracy of such models is strictly related to the number of layers, neurons, and inputs [10–12], therefore, to tackle more complex problems, these architectures are forced to go in depth. While on the one hand we have increasing precision, on the other hand the high number of degrees of freedom translates into a longer optimization step and, from a practical point of view, into a larger architecture to manage. The dimension of the network is rarely considered a bottleneck of this methodology, but the diffusion of neural networks in many engineering fields led to its employment also in embedded systems [13–15], which typically show limited hardware. Deep vision algorithms are indeed developed using workstations with high computational resources, posing a challenge when deploying them in industrial applications. The vision devices, in which these nets need to be integrated, are often characterized by restricted memory sizes and low CPU performance [16–18]. In these contexts the size of the architecture can thus become an additional constraint, requiring a reduction in the network's degrees of freedom.

Finding the intrinsic dimension of neural networks is a very challenging task and, to the best of the authors' knowledge, lacks rigorous theoretical proofs. Various methods have been proposed, including network pruning and sharing [19–23], low-rank matrix and tensor factorization [24–27], parameter quantization [28–30], and knowledge distillation [31–34]. In this contribution (see Fig. 1), we present an extension of the idea explored in [35], where the Active Subspace (AS) property and Polynomial Chaos Expansion (PCE) are exploited to provide a reduced and more robust version of the original network. While such work has focused on analyzing the AS capability in reducing deep architectures, we aim here to provide a generic framework for neural network

Nicola Demo and Gianluigi Rozza are authors contributed equally to this work.

✉ Gianluigi Rozza
gianluigi.rozza@sissa.it

Laura Meneghetti
laura.meneghetti@sissa.it

Nicola Demo
nicola.demo@sissa.it

[1] Mathematics Area, mathLab, SISSA,
via Bonomea 265, Trieste I-34136, Italy

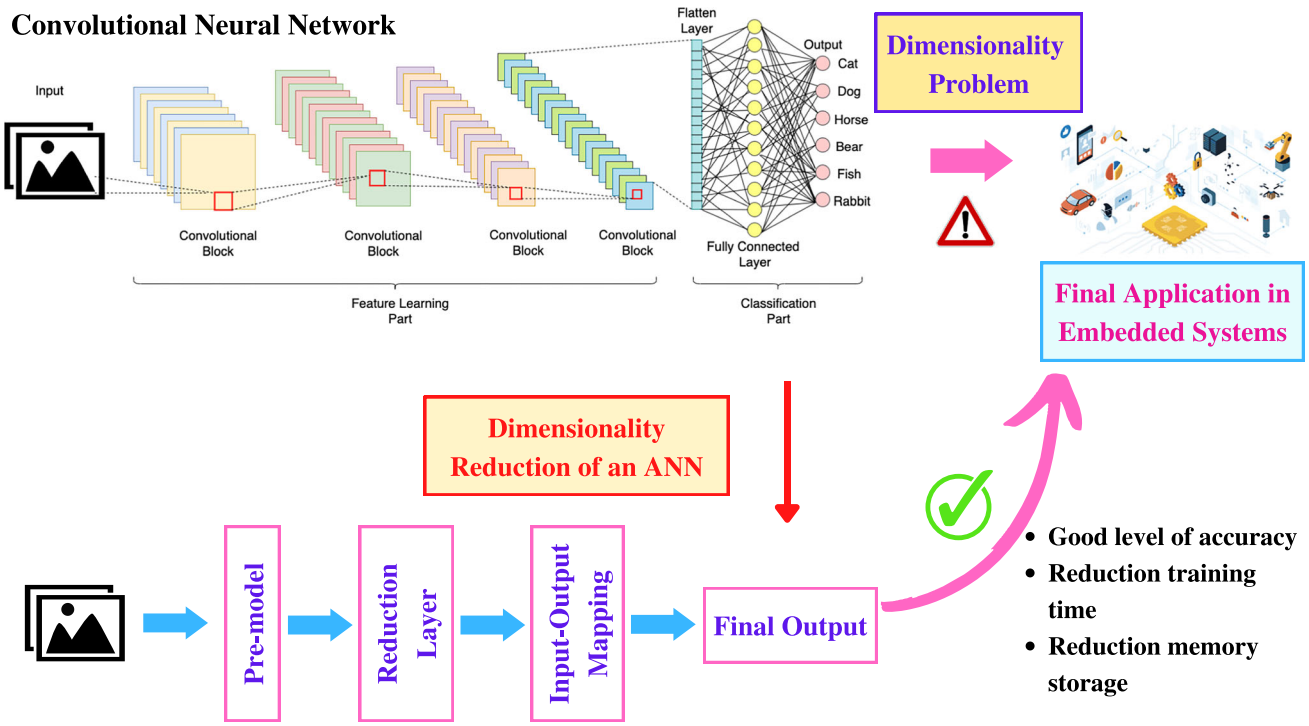

Fig. 1 Graphical representation of the problem and the proposed solution, as described in this contribution

reduction, investigating other mathematical tools rather than AS and PCE. Mimicking the procedure presented in [35], the original architecture is initially split into two cascading parts: the *pre-* and *post-model*. We assume that the second one brings a negligible contribution to the final outcome, giving us the possibility to approximate such part of the model without introducing a significant error. A response surface (or in more general terms, an input-output mapping) is indeed built to fit the data, replacing the last layers of the network. This response surface may belong to a high-dimensional space since the input dimension is equal to the dimension of the output features of the pre-model. Consequently, in order to maintain the reduction computationally affordable, we also need to reduce the dimensionality of the pre-model outputs, which, it should be noted, are also the input parameters for the response surface. By combining all these ingredients, we obtain a reduced version of the network that only includes a few of the initial layers, but achieves a level of accuracy comparable to the full model. It is important to specify that the numerical experiments we are about to present exclusively involve Convolutional Neural Networks (CNNs), but the methodology can potentially be applied to other models as well.

In this contribution, we examine various tools for the dimensional reduction and response surface analysis. In addition to AS and PCE, already tested in the aforementioned reference, we employ Proper Orthogonal Decomposition (POD) and Feedforward Neural Network (FNN). The former, similar to AS, is a well-established technique for Model Order Reduction [36–38], which compresses the data by projecting it onto a lower-dimensional space. On the other hand, FNN is employed to construct the surface response as an alternative to PCE. The advantage of FNN over PCE is twofold: *i*,) the simplified input-output mapping (thanks to the low-dimensional space) allows for a FNN with fewer layers and neurons, further reducing the already minimal space demanded for the PCE method; *ii*,) from a programming perspective, the possibility to approximate a part of the neural network with another network makes the software integration easier, especially when the hosting system is embedded.

The article is organized as follows. Section 2 provides an algorithmic overview of all the numerical methods involved in the reduction framework. This includes an analysis of AS in Section 2.1.1, POD in Section 2.1.2, PCE in Section 2.2.1, and FNN in Section 2.2.2. In Section 3, we delve into the details of the framework used to reduce the neural networks. Section 4 is dedicated to presenting the results obtained by reducing benchmark CNNs designed for image recognition with the proposed methodology. We conduct this analysis using three different datasets during the initial learning step, investigating the dependency of the results on the original problem. Finally, in Section 5 we summarize the entire procedure and propose some future perspectives to enhance the framework.

2 Numerical tools

In this section we introduce all the techniques employed for the reduction of the network, in order to facilitate the comprehension of the framework discussed in Section 3.

2.1 Dimensionality reduction techniques

This subsection provides an algorithmic overview of the reduction methods examined in this contribution: the Active Subspace (AS) property, and the Proper Orthogonal Decomposition (POD). Widely employed in the reduced order modeling community, such techniques are used here to decrease the dimensionality of the intermediate convolutive features. However, the specific details will be discussed in the next section. We just specify that, while this work concentrates on AS and POD, the framework is generic, allowing to replace these two methods with others for dimensionality reduction.

2.1.1 Active subspaces

Active Subspaces (AS) [39, 40] method is a reduction tool used to identify important directions in the parameter space by exploiting the gradients of the function of interest. Such information allows the application of a rotational transformation to the domain, in order to obtain an approximation of the original function but in a lower dimension. Let $\boldsymbol{\mu} = [\mu_1 \dots \mu_n]^T \in \mathbb{R}^n$ represent a n-dimensional variable with an associated probability density function $\rho(\boldsymbol{\mu})$, and let g be the function of interest, $g(\boldsymbol{\mu}) : \mathbb{R}^n \to \mathbb{R}$. We assume here g is scalar and continuous (for the vector-valued extension see [41, 42]). Starting from this, an uncentered covariance matrix $\mathbf{C}$ of the gradient of g can be constructed by considering the average of the outer product of the gradient with itself:

$$\mathbf{C} = \mathbb{E}[\nabla g(\boldsymbol{\mu}) \nabla g(\boldsymbol{\mu})^T] = \int (\nabla_{\boldsymbol{\mu}} g)(\nabla_{\boldsymbol{\mu}} g)^T \rho \mathrm{d}\boldsymbol{\mu}, \tag{1}$$

where the symbol $\mathbb{E}[\cdot]$ denotes the expected value, and $\nabla_{\boldsymbol{\mu}} g \equiv \nabla g(\boldsymbol{\mu})$. We assume the gradients are computed during the simulation, otherwise, if not provided, they can be approximated with different techniques such as local linear models, global models, finite difference, or Gaussian process [43–45], for example. Since $\mathbf{C}$ is symmetric, it admits the following eigenvalue decomposition:

$$\mathbf{C} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T, \quad \boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \dots, \lambda_n), \ \lambda_1 \geq \dots \geq \lambda_n \geq 0, \tag{2}$$

where $\mathbf{V}$ is the $n \times n$ orthogonal matrix whose columns $\{\mathbf{v}^1, \dots, \mathbf{v}^n\}$ are the normalized eigenvectors of $\mathbf{C}$, whereas $\boldsymbol{\Lambda}$ is a diagonal matrix containing the corresponding non-negative eigenvalues λ_i, for $i = 1, \dots, n$, arranged in descending order.

We can decompose these two matrices as:

$$\boldsymbol{\Lambda} = \begin{bmatrix} \boldsymbol{\Lambda}_1 & \\ & \boldsymbol{\Lambda}_2 \end{bmatrix},$$
$$\mathbf{V} = [\mathbf{V}_1 \ \mathbf{V}_2], \qquad \mathbf{V}_1 \in \mathbb{R}^{n \times n_{\mathrm{AS}}}, \ \mathbf{V}_2 \in \mathbb{R}^{n \times (n - n_{\mathrm{AS}})}. \tag{3}$$

The space spanned by $\mathbf{V}_1$ columns is called the *active subspace* of dimension $n_{\mathrm{AS}} < n$, whereas the *inactive subspace* is defined as the range of the remaining eigenvectors in $\mathbf{V}_2$. Once we have defined these spaces, the input $\boldsymbol{\mu} \in \mathbb{R}^n$ can be reduced to a low-dimensional vector $\tilde{\boldsymbol{\mu}}_1 \in \mathbb{R}^{n_{\mathrm{AS}}}$ using $\mathbf{V}_1$ as projection map. To be more precise, any $\boldsymbol{\mu} \in \mathbb{R}^n$ can be expressed in this way using the decomposition in Eq. 3 and the properties of $\mathbf{V}$:

$$\boldsymbol{\mu} = \mathbf{V}\mathbf{V}^T\boldsymbol{\mu} = \mathbf{V}_1\mathbf{V}_1^T\boldsymbol{\mu} + \mathbf{V}_2\mathbf{V}_2^T\boldsymbol{\mu} = \mathbf{V}_1\tilde{\boldsymbol{\mu}}_1 + \mathbf{V}_2\tilde{\boldsymbol{\mu}}_2, \tag{4}$$

where the two new variables $\tilde{\boldsymbol{\mu}}_1$ and $\tilde{\boldsymbol{\mu}}_2$ are the *active* and *inactive variable* respectively:

$$\tilde{\boldsymbol{\mu}}_1 = \mathbf{V}_1^T\boldsymbol{\mu} \in \mathbb{R}^{n_{\mathrm{AS}}}, \qquad \tilde{\boldsymbol{\mu}}_2 = \mathbf{V}_2^T\boldsymbol{\mu} \in \mathbb{R}^{n - n_{\mathrm{AS}}}. \tag{5}$$

For the actual computations of the AS, we have used the open-source Python package called ATHENA [46].

2.1.2 Proper orthogonal decomposition

In this section, we will discuss the Proper Orthogonal Decomposition (POD) approach of Reduce Order Modeling [36–38, 47] for reducing the number of degrees of freedom of a parametric system. Specifically, we will focus on the POD with interpolation (PODI) method [48–50].

Let $\mathbf{S} = [\mathbf{u}^1 \dots \mathbf{u}^{n_S}]$ be the matrix of snapshots, i.e. the full order system outputs $\mathbf{u}^i \in \mathbb{R}^N$. Once these solutions are collected, we aim to describe them as a linear combination of a few main structures, the POD modes, and thus project them onto a low dimensional space spanned by these modes. To calculate the POD modes, we need to compute the singular value decomposition (SVD) of the snapshots matrix $\mathbf{S}$:

$$\mathbf{S} = \boldsymbol{\Psi}\boldsymbol{\Sigma}\boldsymbol{\Theta}^T, \tag{6}$$

where the left-singular vectors, i.e. the columns of the unitary matrix $\boldsymbol{\Psi}$, are the POD modes, and the diagonal matrix $\boldsymbol{\Sigma}$ contains the corresponding singular values in decreasing order. Therefore, by selecting the first modes we are retaining only the most energetic ones and we can construct a reduced

space into which we project the high-fidelity solutions. As a results, we obtain:

$$\mathbf{S}^{\text{POD}} = \mathbf{\Psi}_{N_{\text{POD}}}^T \mathbf{S}, \tag{7}$$

where $\mathbf{\Psi}_{N_{\text{POD}}}$ is the matrix containing only the first N_{POD} modes, and the columns of $\mathbf{S}^{\text{POD}}$ represent the reduced snapshot $\tilde{\mathbf{u}}^i \in \mathbb{R}^{N_{\text{POD}}}$.

2.2 Input–output mapping

After reducing the dimensions of the outputs from the intermediate layer, we need to establish a correlation between these outputs and the final output of the original network. For example, in an image identification problem, this would involve determining the classes to which the image belongs. To achieve this, we construct an input–output mapping using the input dataset. The following subsections provide an algorithmic overview of the two methods that were explored to approximate this mapping: the Polynomial Chaos Expansion (PCE) [51] and the Feed–forward Neural Network (FNN) [52].

2.2.1 Polynomial chaos expansion

The Polynomial Chaos Expansion (PCE) theory was initially proposed by Wiener in [53], demonstrating that a real-valued random variable $X : \mathbb{R}^R \to \mathbb{R}$ can be decomposed in the following manner:

$$X(\boldsymbol{\xi}) = \sum_{j=0}^{\infty} c_j \boldsymbol{\phi}_j(\boldsymbol{\xi}), \tag{8}$$

i.e. as an infinite sum of orthogonal polynomials weighted by unknown deterministic coefficients c_j [54]. The vector $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_R)$ represents the multi-dimensional random vector, where each element is associated with uncertain input parameters, while $\boldsymbol{\phi}_j(\boldsymbol{\xi})$ are multivariate orthogonal polynomials, that can be decomposed into products of one-dimensional orthogonal polynomials with different variables.

We can approximate the infinite sum in Eq. 8 by truncating it at the $(P+1)$-th term, such that:

$$X(\boldsymbol{\xi}) \approx \sum_{j=0}^{P} c_j \boldsymbol{\phi}_j(\boldsymbol{\xi}). \tag{9}$$

The number of unknown coefficients in this summation is given by $P + 1 = \frac{(p+R)!}{p!R!}$ [55], where p is the degree of the polynomial we are considering in the R-dimensional space.

When the parameters $\xi_1, \ldots, \xi_R$ are independent, $\boldsymbol{\phi}_j(\boldsymbol{\xi})$ can be decomposed into products of one-dimensional functions:

$$\begin{aligned} \boldsymbol{\phi}_j(\boldsymbol{\xi}) &= \boldsymbol{\phi}_j(\xi_1, \ldots, \xi_R) \\ &= \prod_{k=1}^{R} \phi_k^{d_k}(\xi_k), \qquad j = 0, \ldots, P, \\ d_k &= 0, \ldots, p \quad \text{s.t.} \sum_{k=1}^{R} d_k \leq p. \end{aligned} \tag{10}$$

To determine the PCE, we need to find out the polynomial chaos expansion coefficients c_j for $j = 0, \ldots, P$, and the one-dimensional orthogonal polynomials $\phi_k^{d_k}$, $k = 1, \ldots, R$, of degree d_k.

Based on the work of Askey and Wilson [56], we can provide the orthogonal polynomials for different distributions. One of the possible choices is represented by the Gaussian distribution with the related Hermite polynomials.

The estimation of the coefficients of PCE can then be carried out in different ways [57, 58]. One method involves using a projection technique based on the orthogonality of the polynomials. Another method, which we will describe, is a regression-based approach.

In order to determine the coefficients c_j, we need to solve a minimization problem:

$$\mathbf{c} = \arg\min_{\mathbf{c}^* \in \mathbb{R}^P} \frac{1}{N_{\text{PCE}}} \sum_{i=1}^{N_{\text{PCE}}} \left(\hat{X} - \sum_{j=0}^{P} c_j^* \boldsymbol{\phi}_j(\boldsymbol{\xi}^i) \right), \tag{11}$$

where N_{PCE} indicates the total number of realizations of the input vector we are considering, whereas $\hat{X}$ represents the real output of the model. To solve Eq. 11 we consider the following matrix:

$$\mathbf{\Phi} = \begin{pmatrix} \boldsymbol{\phi}_0(\boldsymbol{\xi}^1) & \boldsymbol{\phi}_1(\boldsymbol{\xi}^1) & \cdots & \boldsymbol{\phi}_P(\boldsymbol{\xi}^1) \\ \boldsymbol{\phi}_0(\boldsymbol{\xi}^2) & \boldsymbol{\phi}_1(\boldsymbol{\xi}^2) & \cdots & \boldsymbol{\phi}_P(\boldsymbol{\xi}^2) \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{\phi}_0(\boldsymbol{\xi}^{N_{\text{PCE}}}) & \boldsymbol{\phi}_1(\boldsymbol{\xi}^{N_{\text{PCE}}}) & \cdots & \boldsymbol{\phi}_P(\boldsymbol{\xi}^{N_{\text{PCE}}}) \end{pmatrix}. \tag{12}$$

Thus, the solution of Eq. 11 is computed using a least-square optimization approach, as shown in Eq. 13:

$$\mathbf{c} = (\mathbf{\Phi}^T \mathbf{\Phi})^{-1} \mathbf{\Phi}^T \hat{X}. \tag{13}$$

It is important to emphasize that if the matrix $\mathbf{\Phi}^T \mathbf{\Phi}$ is ill-conditioned, as may occur, then the singular value decomposition method should be employed.

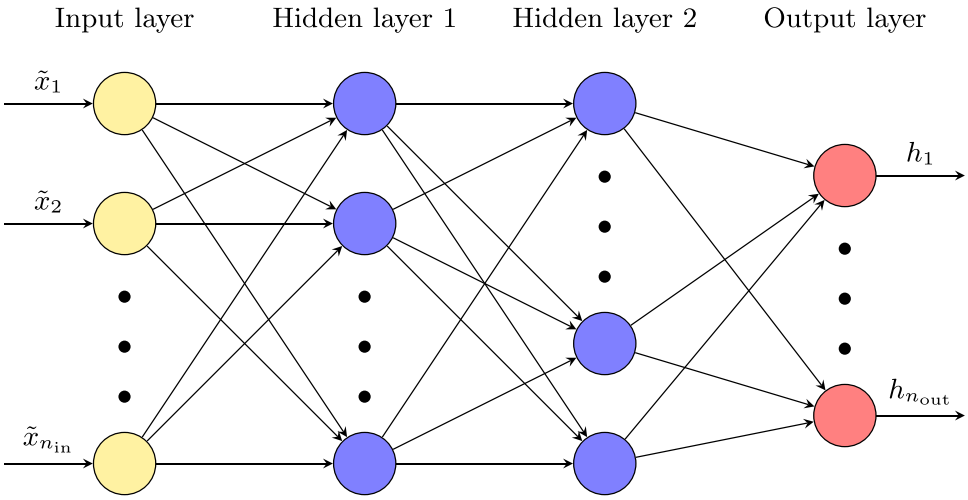

Fig. 2 Schematic structure of a Feedforward Neural Network with 2 hidden layers

2.2.2 Feedforward neural network

A Feedforward Neural Network (FNN), also known as *multilayer perceptron*, is a popular neural network model commonly used for function regression [52]. As depicted in Fig. 2, it mainly comprises an input layer, an output layer, and a certain number of hidden layers[1], where the processing units composing them are called *neurons*. Each neuron is then characterized by a weight vector that determines the strength of its connections with neurons in the subsequent layer.

From a more technical perspective, let $\tilde{\mathbf{x}} \in \mathbb{R}^{n_{\text{in}}}$ represent the input vector and M denote the total number of hidden layers in the FNN. The output vector $\mathbf{h} \in \mathbb{R}^{n_{\text{out}}}$ is obtained by applying an *activation function* to the weighted sum of all the inputs received by that neuron. The role of this activation function is to introduce non-linearity in the network. There are numerous options available [10, 60], and some common ones are represented by the ReLU function, sigmoid, logistic function, and radial activation functions.

To better understand the derivation of the general formula (15), we start by considering a FNN that comprises a single output and one hidden layer. In this scenario, the final output can be expressed as:

$$\mathbf{h} = \sigma\left(\sum_{i=1}^{n_{\text{in}}} w_i \tilde{x}_i + b_i\right), \tag{14}$$

where σ is the activation function, $W = \{w_i\}_{i=1}^{n_{\text{in}}}$ represents the weights of the net and b the bias[2]. Therefore, when considering M layers, the final output can be seen as a weighted sum of its inputs followed by the activation function, where each input can be rewritten using the same approach described in Eq. 14:

$$\begin{aligned} h_j &= \sigma\left(\sum_{i=1}^{n_M} w_{ji}^{(M+1)} \tilde{x}_i^{(M)}\right) \\ &= \sigma\left(\sum_{i=1}^{n_M} w_{ji}^{(M+1)} \left(\sigma\left(\sum_{q=1}^{n_{M-1}} w_{iq}^{(M)} \tilde{x}_q^{(M-1)}\right)\right)\right) = \\ &\ldots = \sigma\left(\sum_{i=1}^{n_M} w_{ji}^{(M+1)} \left(\sigma\left(\sum_{q=1}^{n_{M-1}} w_{iq}^{(M)} \left(\sigma\left(\ldots\left(\sigma\left(\sum_{k=1}^{n_{in}} w_{sk}^{(1)} \tilde{x}_k\right)\right)\right)\right)\right)\right)\right), \\ j &= 1, \ldots, n_{\text{out}}, \end{aligned} \tag{15}$$

where n_m, $m = 1, \ldots, M$, represents the number of neurons in layer m, whereas n_{in} and n_{out} are the neurons in the input and output layers respectively. $W^m = (w_{ki}^{(m)})_{ki}$, $k = 1, \ldots, n_m$, $i = 1, \ldots, n_{m-1}$ indicates then the weight matrix related to layer m. Note that the first number in any weight's subscript matches the index of the neuron in the next layer and the second number matches the index of the neuron in the previous layer.

Once we have constructed an FNN by choosing its architecture, we need to gain a performing model for a desired task. One of the main characteristics of an FNN is indeed its ability to learn from observational data during the so-called *training process*. In this phase, the net acquires knowledge from our dataset by minimizing the loss function[3] $\mathcal{L}$:

$$\min_W \left\{ \frac{1}{n_{\text{out}}} \sum_{i=1}^{n_{\text{out}}} \mathcal{L}(h_i, \hat{h}_i) \right\}, \tag{16}$$

[1] A priori there is not a right number of hidden layers to use: it depends on the fields of application of your net and on the problem under consideration [12, 59].

[2] For simplicity the bias is put to zero in the following discussion.

[3] There exists several types of loss functions that are used in this context, such as the Cross-Entropy Loss, the Euclidean Loss, and the Hinge Loss. The appropriate choice depends on the problem being considered [10, 61].

where $\mathbf{h} = \{h_j\}_{j=0}^{n_{\text{out}}}$ represents the expected output and $\hat{\mathbf{h}} = \hat{\mathbf{h}}(\tilde{\mathbf{x}}; W) = \{\hat{h}_j(\tilde{\mathbf{x}}; W)\}_{j=0}^{n_{\text{out}}}$ is the prediction made by our FNN. To solve this minimization problem, the *Backpropagation algorithm* [62] is commonly employed. Consequently, the model's parameters are optimized by adjusting the network's weights using the following procedure:

$$w_{ki}^{(m),t} = w_{ki}^{(m),t-1} - \epsilon \frac{d\mathcal{L}}{dw_{ki}^{(m)}}, \tag{17}$$

where ϵ is the *learning rate*, which is appropriately chosen according to the problem under consideration. The parameter t represents the training epoch, which indicates a complete repetition of the parameter update involving the complete training dataset at once. The gradients required for the weight update in Eq. 17 are then computed using the chain rule.

3 The reduced artificial neural networks

In this section, we provide the rigorous description of the proposed framework, which is summarized in Fig. 3 and Fig. 1. The primary objective of our framework is to reduce, in terms of dimensionality, a generic Artificial Neural Network (ANN). Indeed, it is important to note that the only assumption we make about the original network is that it consists of L layers.

Network splitting

In the beginning, the original network, denoted as $\mathcal{ANN} : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$ is split into two distinct parts. The first l layers constitute the *pre-model*, while the last $L - l$ layers form the so-called *post-model*. By describing the network as composition of functions $\mathcal{ANN} \equiv f_L \circ f_{L-1} \circ \cdots \circ f_1$, we can formally define the pre- and the post-model as follows:

$$\begin{aligned} \mathcal{ANN}_{\text{pre}}^{l} &= f_l \circ f_{l-1} \circ \cdots \circ f_1, \\ \mathcal{ANN}_{\text{post}}^{l} &= f_L \circ f_{L-1} \circ \cdots \circ f_{l+1}, \end{aligned} \tag{18}$$

where each function $f_j : \mathbb{R}^{n_{j-1}} \to \mathbb{R}^{n_j}$ for $j = 1, \ldots, L$, represents the different layers of the network — e.g. convolutional, fully connected, batch-normalization, ReLU, pooling layers. The original model can then be rewritten as:

$$\mathcal{ANN}(\mathbf{x}^0) = \mathcal{ANN}_{\text{post}}^{l}(\mathcal{ANN}_{\text{pre}}^{l}(\mathbf{x}^0)), \tag{19}$$

for any $1 \leq l < L$ and $\mathbf{x}^0 \in \mathbb{R}^{n_0}$.

As described in [35], the reduction of the network is achieved by approximating the post-model, which means that the pre-model is actually copied from the original network to the reduced one. Before proceeding with the algorithmic explanation of how the post-model is approximated, we specify that the index l, denoting the *cut-off layer*, is the only parameter of this initial step, and it plays an important role in the final outcome. This index indeed determines how many layers of the original network are retained in the reduced architecture, controlling, in a few words, how much information of the original network we are discarding. As described in [35], it is then chosen empirically based on considerations about the network and the dataset at hand, balancing the final accuracy and the compression ratio.

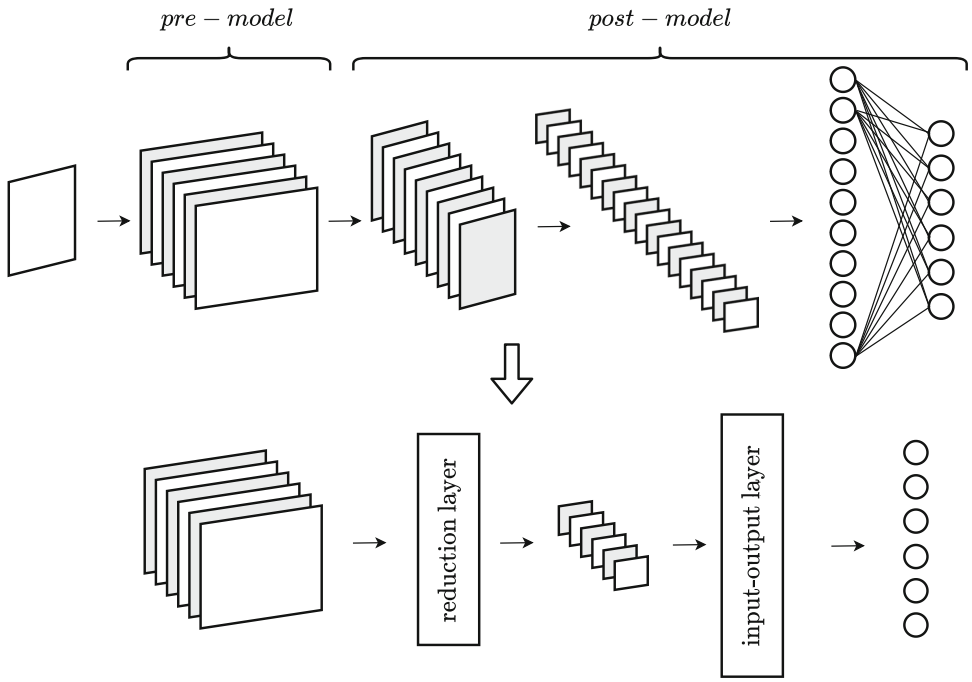

Fig. 3 Graphical representation of the reduction method proposed for a CNN

Algorithm 1 Pseudo-code for the construction of the reduced Artificial Neural Network.

Inputs:

- a dataset with N_{train} input samples $\mathcal{D}_0 = \{\mathbf{x}^{(0),j}\}_{j=1}^{N_{\text{train}}}$,
- an artificial neural network $\mathcal{ANN}$,
- $\{\mathbf{y}^j\}_{j=1}^{N_{\text{train}}}$ real output of the $\mathcal{ANN}$,
- reduced dimension r,
- index of the cut-off layer l

1: $\mathcal{ANN}^l_{\text{pre}}, \mathcal{ANN}^l_{\text{post}} = \text{splitting\_net}(\mathcal{ANN}, l)$
2: $\mathbf{x}^{(l)} = \mathcal{ANN}^l_{\text{pre}}(\mathbf{x}^0)$
3: $\mathbf{z} = \text{reduce}(\mathbf{x}^{(l)}, r)$
4: $\hat{\mathbf{y}} = \text{input\_output\_map}(\mathbf{z}, \mathbf{y})$
5: Training of the constructed reduced net

Output: Reduced Net $\mathcal{ANN}^{\text{red}}$

Dimensionality reduction

As mentioned earlier, our goal is to project the output $\mathbf{x}^{(l)}$ of the pre-model onto a lower-dimensional space using reduction techniques as:

- **Active Subspaces**: as described in Section 2.1.1 and in [35], we consider a function g_l defined by:

$$g_l(\mathbf{x}^{(l)}) = \text{loss}(\mathcal{ANN}^l_{\text{post}}(\mathbf{x}^{(l)})), \tag{20}$$

  in order to extract the most important directions and determine the projection matrix used to reduce the pre-model output.
- **Proper Orthogonal Decomposition**: as discussed in Section 2.1.2, the SVD decomposition (6) is exploited to compute the projection matrix $\mathbf{\Psi}_r$ and subsequently obtain the reduced solution

$$\mathbf{z} = \mathbf{\Psi}_r^T \mathbf{x}^{(l)}. \tag{21}$$

It is important to emphasize that in order to apply these methodologies to the pre-model output, a flattening of $\mathbf{x}^{(l)}$ should be carried out. These approaches are specifically based on flat-view matrix models, requiring the transformation of $\mathbf{x}^{(l)}$ from a tensorial structure to a two-dimensional one.

Input-Output mapping

The final part of the reduced neural network is dedicated to classifying the output generated by the reduction layer. Two different techniques have been employed for this purpose:

- the **Polynomial Chaos Expansion**, as introduced in Section 2.2.1. According to Eq. 9, the final output of the network, denoted as $\mathbf{y} = \mathcal{ANN}(\mathbf{x}^0) \in \mathbb{R}^{n_L}$, which represents the true response of the model, can be approximated as follows:

$$\hat{\mathbf{y}} \approx \sum_{|\boldsymbol{\alpha}|=0}^{p} \mathbf{c}_{\boldsymbol{\alpha}} \boldsymbol{\phi}_{\boldsymbol{\alpha}}(\mathbf{z}), \qquad |\boldsymbol{\alpha}| = \alpha_1 + \cdots + \alpha_r, \tag{22}$$

  where $\boldsymbol{\phi}_{\boldsymbol{\alpha}}(\mathbf{z})$ are the multivariate polynomial functions chosen based on the probability density function ρ associated with $\mathbf{z}$. Therefore, the estimation of coefficients $\mathbf{c}_\alpha$ is carried out by solving the minimization problem (11):

$$\min_{c_\alpha} \frac{1}{N_{\text{train}}} \sum_{j=1}^{N_{\text{train}}} \left\| \mathbf{y}^j - \sum_{|\boldsymbol{\alpha}|=0}^{p} \mathbf{c}_{\boldsymbol{\alpha}} \boldsymbol{\phi}_{\boldsymbol{\alpha}}(\mathbf{z}^j) \right\|^2. \tag{23}$$

- a **Feedforward Neural Network**, as described in Section 2.2.2. In this case, the output of the reduction layer $\mathbf{z}$ coincides with the network input. By applying Eq. 15, we can determine the final output $\hat{\mathbf{y}}$ of the reduced net[4], which is given by:

$$\begin{aligned} \hat{y}_j &= \sum_{i=1}^{n_1} w_{ji}^{(2)} z_i^{(1)} \\ &= \sum_{i=1}^{n_1} w_{ji}^{(2)} \sigma \left(\sum_{m=1}^{r} w_{im}^{(1)} z_m \right), \quad j = 1, \ldots, n_{\text{out}}, \end{aligned} \tag{24}$$

  where n_{out} corresponds to the number of categories that compose the dataset under consideration, and σ is the *Softplus* function:

$$\text{Softplus}(\mathbf{x}) = \frac{1}{\beta} \log(1 + \exp(\beta \mathbf{x})). \tag{25}$$

3.1 Training phase

Once the reduced version of the network is constructed, we need to train it. Following [35], for the training phase of the reduced ANN, we employ the technique of *knowledge distillation* [31]. A knowledge distillation framework involves a large pre-trained *teacher model*, which is our full network, and a small *student model*, in our case $\mathcal{ANN}^{\text{red}}$. Therefore, the main goal is to efficiently train the student network under the guidance of the teacher network to achieve comparable or even superior performance.

Let $\mathbf{y}$ be a vector of *logits*, which refers to the output of the last layer in a deep neural network. The probability p_i

[4] Note that in this case the number of hidden layers is set to 1 since, as discussed in Section 4, we notice that one hidden layer is enough to gain a good level of accuracy (see for example Table 1).

that the input belongs to the i-th class is determined by the softmax function:

$$p_i = \frac{exp(y_i)}{\sum_{j=0}^{n_{\text{class}}} exp(y_j)}. \tag{26}$$

As described in [31], a temperature factor T needs to be introduced in order to control the importance of each target:

$$p_i = \frac{exp(y_i/T)}{\sum_{j=0}^{n_{\text{class}}} exp(y_j/T)}, \tag{27}$$

where if $T \to \infty$ all classes have the same probability, whereas if $T \to 0$ the targets p_i become one-hot labels.

Firstly, we need then to define the *distillation loss*, which matches the logits between the teacher model and the student model, as mentioned in [35]. The knowledge transfer from the teacher to the student is accomplished by mimicking the final prediction of the full net, using *response-based knowledge*. Therefore, in this case, the distillation loss [31, 32] is given by:

$$L_D(p(\mathbf{y}_t, T), p(\mathbf{y}_s, T)) = \mathcal{L}_{\text{KL}}(p(\mathbf{y}_t, T), p(\mathbf{y}_s, T)), \tag{28}$$

where $\mathbf{y}_t$ and $\mathbf{y}_s$ indicate the logits of the teacher and student networks, respectively, while $\mathcal{L}_{\text{KL}}$ represents the Kullback-Leibler (KL) divergence loss [63]:

$$\begin{aligned}&\mathcal{L}_{\text{KL}}((p(\mathbf{y}_s, T), p(\mathbf{y}_t, T))\\ &= T^2 \sum_j p_j(y_{t,j}, T) \log \frac{p_j(y_{t,j}, T)}{p_j(y_{s,j}, T)}.\end{aligned} \tag{29}$$

The *student loss* is then defined as the cross-entropy loss between the ground truth label and the logits of the student network [32]:

$$L_S(\mathbf{y}, p(\mathbf{y}_s, T)) = \mathcal{L}_{\text{CE}}(\hat{\mathbf{y}}, p(\mathbf{y}_s, T)), \tag{30}$$

where $\hat{\mathbf{y}}$ is a ground truth vector, characterized by having only the component corresponding to the ground truth label on the training sample set to 1, while the other components are set to 0. $\mathcal{L}_{\text{CE}}$ represents instead the cross entropy loss, which is described as follows:

$$\mathcal{L}_{\text{CE}}(\hat{\mathbf{y}}, p(\mathbf{y}_s, T)) = \sum_i -\hat{y}_i \log(p_i(y_{s,i}, T)). \tag{31}$$

As can be observed, both losses, Eqs. 28 and 30, use the same logits of the student model but with different temperatures. In the distillation loss, the temperature T is set to a value greater than 1 ($T = \tau > 1$) while in the student loss, the temperature is set to 1 ($T = 1$). Finally, the final loss is calculated as a weighted sum between the distillation loss and the student loss:

$$\begin{aligned}L(\mathbf{x}^0, W) = &\lambda L_D(p(\mathbf{y}_t, T = \tau), p(\mathbf{y}_s, T = \tau))\\ &+(1 - \lambda) L_S(\hat{\mathbf{y}}, p(\mathbf{y}_s, T = 1)),\end{aligned} \tag{32}$$

where λ is the regularization parameter, $\mathbf{x}^0$ represents an input vector from the training set, and W coincides with the parameters of the student model.

4 Numerical results

In this section, we present a comparison between the results obtained using different reduction methods in terms of final accuracy, memory allocation, and procedure speed.

4.1 Neural network architectures

We used Convolutional Neural Networks (CNNs) as a test network, which is a type of ANN commonly applied to image recognition problems [64, 65]. In the past decade, several CNN architectures have been introduced [11, 61] to address this problem, such as AlexNet, ResNet, Inception, VGGNet.

As starting point for testing our methods, we have employed one of the VGG network architectures, specifically VGG-16 [66]. As shown in Fig. 4, this architecture consists of the following components:

- 13 *convolutional blocks*. Each block includes a convolutional layer followed by a non-linear layer, where ReLU is used as the activation function.
- 5 *max-pooling layers*,
- 3 *fully-connected layers*.

The ConvNet used in our study is called VGG-16, as it is composed of a total of 16 layers with tunable parameters. Out of these 16 layers, 13 are convolutional layers, and the remaining 3 are fully connected layers.

In comparison, we also tested our methodology on ResNet [67], and in particular on ResNet-110, as done in [35]. As the name suggests, ResNet-110 comprises a total of 110 layers. These layers are divided into 3 groups, each containing 18 basic residual blocks. We recall that these blocks consist of two convolutional layers, followed by batch normalization, and a skip/shortcut connection that adds the input to the output of the block.

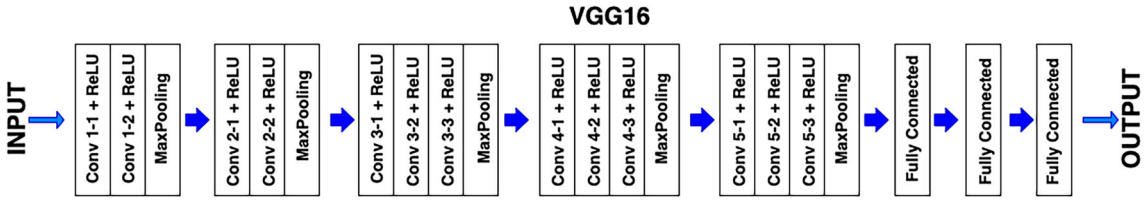

Fig. 4 Graphical representation of VGG-16 architecture

4.2 Dataset

For training and testing our net we have used[5]:

- **CIFAR-10 dataset** [68], a computer-vision dataset used for object recognition. It comprises 60000 color images of size 32×32, which are divided into 10 non-overlapping classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.
- **Custom dataset**, composed of 3448 color images of size 32×32, organized in 4 classes: 3 non-overlapping classes and a mixed one, characterized by pictures with objects of different categories present at the same time.
- **CIFAR-100 dataset** [68], another benchmark computer-vision dataset for object recognition. It consists of 60000 color images of size 32×32, divided into 100 classes, with each class containing 600 images.

4.3 Software and hardware configuration

To implement and construct the reduced version of the convolutional neural networks described in the previous sections, we utilized PyTorch [69] as our development environment. We also employed the open-source Python library SciPy [70] for scientific computing and the open-source Python package ATHENA [46] for the actual computation of the active subspaces.

Regarding the hardware configuration, we ran all experiments involving VGG-16, except for the CIFAR-100 dataset, on the CPU. All other tests were performed using an NVIDIA GPU. This decision was influenced by the availability of hardware resources during the development and testing phases for the selected architectures.

4.4 Results VGG-16

We now present the results of the reduced network constructed starting from VGG-16 and based on CIFAR-10, CIFAR-100 and our custom dataset. First of all, the original network VGG-16 has been trained[6] on each of the different datasets presented. We needed only a 60 epochs training phase for CIFAR-10 and the custom case, whereas a longer training of 300 epochs was required for CIFAR-100. From Tables 2 and 3, it can be seen that at the end of these learning processes, VGG-16 gains good accuracy: 77.98% for the CIFAR-10 and 95.65% for the custom dataset. Table 4 provides instead the accuracy achieved in the CIFAR-100 case, presenting the Top-1 and Top-5 scores, as done in [35]. It can be observed that the increase in the number of classes has resulted in a lower Top-1 value, as well as the need for longer training.

We report the results obtained with different reduced versions of VGG-16 constructed following the steps of Algorithm 1 and using several cut-off layers[7] l, as reported in [35]: 5, 6, and 7 for CIFAR-10 and the custom case, 7, 8 and 9 for the other dataset. We remark that in the case of dimensionality reduction using the Active Subspaces technique, we employed the *Frequent Direction method* [71], which was implemented within ATHENA to compute the AS. We set the parameter r, representing the dimension of the reduced space, to 50 for both AS and for POD in accordance with [35], where considerations on the structural analysis of VGG-16 can be found.

When a FNN was employed to classify our image, we trained it for 500 epochs with the dataset at hand before re-training the entire reduced net. In Table 1, we provide a summary of the results obtained by training a reduced net using various FNN architectures. This includes different numbers of hidden layers and constant numbers of hidden neurons within each hidden layer of the network. Specifically, we compare the storage requirements of the FNN with

[5] It is important to emphasize that the implementation for both VGG-16 and ResNet-110 differs from the standard approach when applied to the CIFAR dataset. Therefore, as suggested in [35], we have considered this aspect while constructing the models to maintain consistency with the original works.

[6] We have selected 60 and 300 as the number of epochs for the training phases, and for the reduced nets, we have chosen 10 and 20 epochs. This decision was made as a trade-off between achieving a high final accuracy and minimizing the required time. To ensure a fair comparison, we have maintained the same epoch values across all the different cases we are considering.

[7] In [35] and its corresponding implementation, they refer to indices 5, 6, 7, 8 and 9. These indices represent the convolutional layers in a list where only convolutional and linear layers are taken into consideration as possible cut-off layers. Thus, if we consider the entire network with all the different layers, the corresponding layers would be 11, 13, 16, 18 and 20 respectively.

Table 1 Results obtained for the reduced net POD+FNN (7) trained on CIFAR-10 with different structures for the FNN

			Hidden layers			
			1	2	3	4
Hidden neurons	10	Epoch 0	81.39%	67.92%	75.52%	81.57%
		Epoch 10	87.89%	87.59%	87.46%	87.26%
		Storage FNN (MB)	0.0024	0.0028	0.0032	0.0036
	20	Epoch 0	80.17%	80.05%	79.97%	78.28%
		Epoch 10	87.45%	87.13%	87.42%	86.68%
		Storage FNN (MB)	0.0047	0.0063	0.0079	0.0095
	30	Epoch 0	77.57%	80.36%	80.43%	76.26%
		Epoch 10	86.92%	86.25%	86.30%	85.25%
		Storage FNN (MB)	0.0070	0.0106	0.0141	0.0177
	40	Epoch 0	71.24%	70.38%	69.31%	68.15%
		Epoch 10	85.04%	84.60%	84.18%	83.64%
		Storage FNN (MB)	0.0093	0.0156	0.0219	0.0281

Table 2 Results obtained with CIFAR-10 dataset

Network	Accuracy		Storage (MB)			Time	
VGG-16	77.98%		56.15			46 h	
	Epoch 0	Epoch 10	Pre-M	AS/POD	PCE/FNN	Init	Train
AS+PCE (5)	13.52%	82.01%	2.12	3.12	0.05	43 min	4.5 h
AS+FNN (5)	33.06%	80.43%	2.12	3.12	0.0047	5 h	4.5 h
POD+FNN (5)	62.16%	80.24%	2.12	3.12	0.0047	79 min	5 h
AS+PCE (6)	14.42%	84.69%	4.37	3.12	0.05	49 min	5.5 h
AS+FNN (6)	33.76%	82.13%	4.37	3.12	0.0047	5 h	4.5 h
POD+FNN (6)	63.84%	83.93%	4.37	3.12	0.0047	83 min	5 h
AS+PCE (7)	4.25%	85.60%	6.62	0.78	0.05	35 min	5.5 h
AS+FNN (7)	75.66%	86.03%	6.62	0.78	0.0047	1.5 h	5 h
POD+FNN (7)	80.17%	87.45%	6.62	0.78	0.0047	12 min	5 h

Table 3 Results obtained with a custom dataset

Network	Accuracy		Storage (MB)			Time	
VGG-16	95.65%		56.14			22 min	
	Epoch 0	Epoch 10	Pre-M	AS/POD	PCE/FNN	Init	Train
AS+PCE (5)	29.03%	95.21%	2.12	3.12	0.02	2 min	10 min
AS+FNN (5)	94.63%	94.92%	2.12	3.12	0.0021	12.5 min	12 min
POD+FNN (5)	96.52%	96.66%	2.12	3.12	0.0021	28 sec	11.5 min
AS+PCE (6)	29.75%	95.79%	4.37	3.12	0.02	2.5 min	10 min
AS+FNN (6)	94.92%	95.36%	4.37	3.12	0.0021	12.5 min	12.5 min
POD+FNN (6)	96.23%	96.37%	4.37	3.12	0.0021	33 sec	13 min
AS+PCE (7)	28.59%	94.05%	6.62	0.78	0.02	1.5 min	11 min
AS+FNN (7)	94.34%	94.63%	6.62	0.78	0.0021	4.5 min	13 min
POD+FNN (7)	96.37%	96.52%	6.62	0.78	0.0021	33 sec	14 min

Table 4 Results obtained with CIFAR-100 dataset

Network	Accuracy		Storage (MB)			Time	
	Top-1	Top-5					
VGG-16	68.46%	89.81%	56.33			5 h	
			Pre-M	AS/POD	PCE/FNN	Init	Train
AS+PCE (7)	66.10%	90.40%	6.62	0.78	0.51	34 min	8 h
AS+FNN (7)	58.33%	80.19%	6.62	0.78	0.047	39 min	7 h
POD+FNN (7)	61.48%	85.04%	6.62	0.78	0.047	16 min	6.5 h
AS+PCE (8)	68.73%	91.37%	11.12	1.56	0.51	43 min	8.5 h
AS+FNN (8)	56.87%	78.43%	11.12	1.56	0.047	48 min	7.5 h
POD+FNN (8)	65.43%	87.10%	11.12	1.56	0.047	21 min	7 h
AS+PCE (9)	69.77%	91.08%	20.12	1.56	0.51	43 min	8.5 h
AS+FNN (9)	67.10%	86.57%	20.12	1.56	0.047	47 min	8 h
POD+FNN (9)	67.61%	87.08%	20.12	1.56	0.047	21 min	8 h

the accuracy of the considered reduced network POD-FNN under consideration at epoch 0, i.e. after its initialization, and at epoch 10, i.e. after the re-training of the whole reduced net. From the results, it can thus be observed that increasing the number of hidden layers and hidden neurons does not result in improved accuracy. Based on accuracy and memory allocation considerations (refer to Table 1 for details), we opted for the following architecture:

- **CIFAR-10**: FNN with 50 input neurons, 10 output neurons, and one hidden layer with 20 hidden neurons.
- **Custom Dataset**: FNN with 50 input neurons, 4 output neurons, and one hidden layer with 10 hidden neurons.
- **CIFAR-100**: FNN with 50 input neurons, 100 output neurons, and one hidden layer with 70 hidden neurons.

After completing these steps, the reduced neural network was re-trained using CIFAR-10 and the custom dataset for a total of 10 epochs. Additionally, it was re-trained for 20 epochs specifically on the CIFAR-100 dataset. The outcomes of this training process are summarized in Tables 2, 3, and in 4, presenting a comparison among various reduced neural networks in terms of accuracy (both before and after the final training, or using Top-1 and Top-5 scores), memory storage requirements, and the time needed for initialization and training of each reduced network. As mentioned earlier, we provide results for each reduced network, namely AS+PCE, AS+FNN, POD+FNN, using three different cut-off layers: 5,6, and 7 or 7, 8 and 9, depending on the case.

In our context, which specifically involves working with a custom dataset, understanding memory allocation is crucial. This is because we aim to include a CNN into an embedded system that has specific storage constraints. Tables 2, 3 and 4 demonstrate that the memory allocation required for the created reduced nets is decreased with respect to that of the original VGG-16. For instance, the checkpoint file[8] needed to store the full net occupies approximately 56 MB, whereas that of its reduced versions is less than 10 MB in most cases. It is then important to note that for CIFAR-100, opting for higher cut-off values results in a larger storage requirement due to the increased pre-model size. This emphasizes the significant role the cut-off index plays in the final model compression. Additionally, it is worth mentioning that replacing PCE with an FNN leads to a substantial memory space savings of two orders of magnitude: 10^{-4} as opposed to 10^{-2}.

Table 2 shows that in the case of POD+FNN, the net does not require an additional training with the entire dataset. This is because, after the initialization (epoch 0), the network's accuracy is already acceptable, and for index 7, it is already high. Additionally, we observe that all proposed reduced nets require less time to achieve well-performing models. This is reasonable since the compression in size is strictly related to the decrease in the number of CNN parameters. However, while this holds true for CIFAR-10 and the custom dataset, the increased number of classes, and thus complexity, in CIFAR-100 necessitates longer training time.

Table 5 Results obtained for POD+FNN(7) without using a pre-trained original network

Dataset	Accuracy		Storage (MB)	Time	
	Top-1	Top-5		Init	Train
CIFAR-10	86.58%	–	7.42	15.5 min	3.5 h
Custom	96.81%	–	7.42	51 sec	7.5 min
CIFAR-100	56.63%	81.61%	7.45	16 min	13.5 h

[8] Note that in all cases (CIFAR-10, CIFAR-100 and custom dataset) the checkpoint file requires 56 MB of memory. However, if you need to store additional information, such as the architecture of the network, training epochs, and loss, the required allocation increases to around 220 MB.

Table 6 Results obtained with CIFAR-10 dataset

Network	Accuracy		Storage (MB)			Time	
ResNet-110	88.77%		6.94			3 h	
	Epoch 0	Epoch 10	Pre-M	AS/POD	PCE/FNN	Init	Train
AS+PCE (31)	12.38%	80.19%	1.15	1.56	0.05	1.5 h	4.5 h
AS+FNN (31)	8.44%	78.59%	1.15	1.56	0.0047	1.5 h	4 h
POD+FNN (31)	12.01%	81.62%	1.15	1.56	0.0047	11 min	45 min
AS+PCE (33)	11.85%	81.95%	1.30	1.56	0.05	1.5 h	4.5 h
AS+FNN (33)	8.71%	78.10%	1.30	1.56	0.0047	1.5 h	4 h
POD+FNN (33)	9.76%	82.30%	1.30	1.56	0.0047	11 min	47 min
AS+PCE (35)	8.36%	82.53%	1.44	1.56	0.05	1.5 h	4.5 h
AS+FNN (35)	10.40%	80.69%	1.44	1.56	0.0047	1.5 h	4 h
POD+FNN (35)	14.21%	83.72%	1.44	1.56	0.0047	11.5 min	47.5 min

Nevertheless, an interesting aspect of this reduction methodology is the non-necessity of having a pre-trained starting model to obtain an exploitable net, as summarized in Table 5. We provide the results obtained for our proposed reduced net POD+FNN(7), constructed without starting from the pre-trained VGG-16. It can be inferred that with all datasets POD+FNN achieves a comparable level of accuracy as in the previous cases where a pre-trained VGG-16 was employed. However, for CIFAR-10 and the custom dataset, we used the same number of epochs as the pre-trained case, whereas for CIFAR-100, it required twice the number of epochs to achieve the same level of accuracy. The immediate consequence of this is the saving of the time needed to gain a performing network, which amounts to approximately 5 hours. It is evident that these considerations remain valid even when using the custom dataset under consideration. Table 3 reports also how after the initialization POD+FNN has already a greater accuracy than VGG-16 for all the choices of l.

In all cases, it can be observed that the proposed reduced CNN achieves a similar, if not higher, accuracy compared to the original VGG-16, while occupying significantly less storage. Moreover,increasing the cut-off layer index l results in improved accuracy since more original features are retained. However, this also leads to a smaller compression ratio. Consequently, as previously mentioned, determining the appropriate value for l requires striking a trade-off between the desired levels of accuracy and reduction, considering also the specific field of application.

4.5 Results ResNet-110

After obtaining interesting results with VGGNet, we proceeded to test our reduction methodology on ResNet-110, following the approach described in [35]. Initially, the network has been trained on each dataset for 60 epochs, achieving a good level of accuracy as reported in Table 6, in Tables 7, and 8. Similarly to the VGG-16 case, we provide the Top-1 and Top-5 accuracy scores for CIFAR-100.

Also in this setting, we have performed multiple experiments to determine the FNN architecture. In analogy with the approach outlined in Table 1 for reducing VGG-16, we

Table 7 Results obtained with the custom dataset

Network	Accuracy		Storage (MB)			Time	
ResNet-110	97.24%		6.94			36 min	
	Epoch 0	Epoch 10	Pre-M	AS/POD	PCE/FNN	Init	Train
AS+PCE (31)	26.27%	93.32%	1.15	1.56	0.02	4 min	10 min
AS+FNN (31)	33.53%	96.52%	1.15	1.56	0.002	4.5 min	9.5 min
POD+FNN (31)	23.95%	94.91%	1.15	1.56	0.002	48 sec	9.5 min
AS+PCE (33)	18.29%	91.58%	1.30	1.56	0.02	4.5 min	10.5 min
AS+FNN (33)	39.04%	96.08%	1.30	1.56	0.002	4.5 min	10 min
POD+FNN (33)	32.37%	95.07%	1.30	1.56	0.002	50 sec	10 min
AS+PCE (35)	19.45%	93.90%	1.44	1.56	0.02	4.5 min	10.5 min
AS+FNN (35)	37.16%	96.08%	1.44	1.56	0.002	4.5 min	10.5 min
POD+FNN (35)	31.93%	96.23%	1.44	1.56	0.002	51 sec	10 min

Table 8 Results obtained with CIFAR-100 dataset

Network	Accuracy		Storage (MB)			Time	
	Top-1	Top-5					
ResNet-110	89.49%	99.60%	6.97			15 h	
			Pre-M	AS/POD	PCE/FNN	Init	Train
AS+PCE (37)	80.93%	98.75%	1.58	0.78	0.51	1.5 h	8.5 h
AS+FNN (37)	81.18%	99.12%	1.58	0.78	0.04	1.5 h	8.5 h
POD+FNN (37)	79.33%	99.02%	1.58	0.78	0.04	16 min	8 h
AS+PCE (39)	82.78%	99.21%	2.07	0.78	0.51	1.5 h	9.5 h
AS+FNN (39)	82.39%	98.87%	2.07	0.78	0.04	1.5 h	8.5 h
POD+FNN (39)	78.80%	98.79%	2.07	0.78	0.04	16 min	8.5 h
AS+PCE (43)	82.58%	98.75%	3.20	0.78	0.51	1.5 h	9.5 h
AS+FNN (43)	83.67%	99.12%	3.20	0.78	0.04	1.5 h	9 h
POD+FNN (43)	78.89%	98.90%	3.20	0.78	0.04	16 min	9 h

used the same FNN structures described previously for VGG-16 across the different cases. Furthermore, for ResNet, the chosen reduced dimension r is also set to 50, based on the eigenvalue analysis presented in [35]. Numerous tests confirmed that this choice of r was optimal, as increasing its value did not yield improved results.

Once we finalized the compression and input–output mapping techniques, we proceeded to construct the reduced versions of our original model. Algorithm 1 describes the entire procedure, with the last step corresponding to the training phase. During this phase, we re-trained the proposed networks using the aforementioned datasets under consideration. We have thus re-train our reduced nets for 10 epochs in the case of CIFAR-10 and the custom dataset, and for 20 epochs with CIFAR-100. Tables 6, 7 and 8 provide the outcomes obtained using the described experimental setup, comparing them in terms of the achieved accuracy, memory footprint, and time required for the initialization and learning processes. Similarly to what is explained in Section 4.4, we report the results for each proposed reduced net using three different cut-off values[9]: 31, 33, 35 for CIFAR-10 and the custom dataset, and 37, 39, 43 for CIFAR-100. By combining the reduction and input–output mapping methods, we have constructed the following compressed models, AS+PCE, AS+FNN, POD+FNN, of which we are now going to analyze the performances.

In terms of memory allocation, it is worth noting that each of the aforementioned reduced nets requires less than 3 MB of space, resulting in a reduction of approximately 60% in the memory footprint. Furthermore, the introduction of an FNN in the final part of the method leads to a storage decrease of one order of magnitude.

In all cases, we can observe that the reduced networks achieved a level of accuracy comparable to the original ResNet-110. The advantage of constructing lightweight architectures is that they result in faster models in most situations. Specifically, we want to emphasize the POD+FNN net, since it consistently outperforms the other reduced networks in terms of achieved accuracy, storage requirements, initialization, and training times. Regarding the initialization process, we can observe that POD requires less time compared to AS, saving approximately one time hour. Furthermore, the training duration is similar to AS in the case of CIFAR-100 and the custom dataset, while it is faster for CIFAR-10.

In conclusion, based on the aforementioned considerations, we can deduce that the results obtained with ResNet-110 are generally in line with those previously achieved with VGG-16. The proposed reduced methodology enables the creation of lightweight versions of ResNet-110 that are equally accurate to the original model but have fewer parameters, making them more manageable to train.

5 Conclusions and perspectives

In this paper, we propose a generic framework for compressing neural networks, specifically Convolutional Neural Networks, with the objective of reducing the number of layers in the network while minimizing the error in the final prediction. This reduction is achieved by replacing a finite set of network layers with a response surface, which involves also dimensionality reduction techniques to operate on a low-dimensional space. We analyze various dimensionality

[9] The chosen cut-off indexes for ResNet-110 are determined in a similar manner as discussed for VGG-16. Specifically, they are based on the indexes used in [35]. It is important to note that these indexes refer solely to the convolutional layers. Hence, when considering the entire ResNet-110 structure, these indexes correspond to layers 61, 67, 73, 75, 81, and 87, respectively.

reduction methods, and investigate how the combination of these techniques with different input-output mappings can impact the final accuracy.

The primary goal of creating this reduced network is to compress existing deep neural network architectures to be included in embedded systems with memory and space constraints. The numerical experiments conducted on two different CNNs, namely VGG-16 and ResNet-110, demonstrate that the proposed techniques can produce a compressed version of an existing network by reducing the number of layers and parameters. This reduction in size results in memory savings while maintaining a comparable level of accuracy to the original CNN. In comparison to VGG-16, the original ResNet-110 requires less storage space, approximately 7 MB, making it already suitable for many applications in vision-embedded systems. However, the use of smaller devices or specific requirements may necessitate a compressed and faster version of the network. Additionally, the results reveal that the combination of POD with FNN generally leads to reduced training time, making the proposed framework superior to the method presented in [35].

A potential drawback of this technique is the requirement to begin with a pre-trained network in order to reduce it. However, our experiments have demonstrated the non-necessity of this starting point to reach good accuracy with the proposed reduced architecture. Despite the saved space and memory, the actual bottleneck in many problems lies in the learning procedure. In such cases, our framework could be extended to reduce the architecture dimension during training, rather than only after its completion, potentially resulting in a significant speedup in the optimization step.

In conclusion, the conducted experiments illustrate the consistency of our proposed methodology when applied to different CNNs and datasets. While we cannot claim that this reduction framework can be universally applied to all existing types of ANNs, it has proven effective in compressing CNNs for image recognition tasks.

Acknowledgements We thank Marco Tezzele for the productive discussions and comments.

Author Contributions Conceptualization: Laura Meneghetti, Nicola Demo; Methodology: Laura Meneghetti, Nicola Demo; Formal analysis and investigation: Laura Meneghetti; Writing - original draft preparation: Laura Meneghetti; Writing - review and editing: Nicola Demo, Gianluigi Rozza; Funding acquisition: Gianluigi Rozza; Supervision: Gianluigi Rozza.

Funding This work was partially supported by an industrial Ph.D. grant sponsored by Electrolux Professional, and was partially funded by European Union Funding for Research and Innovation — Horizon 2020 Program — in the framework of European Research Council Executive Agency: H2020 ERC CoG 2015 AROMA-CFD project 681447 "Advanced Reduced Order Methods with Applications in Computational Fluid Dynamics" P.I. Professor Gianluigi Rozza. Open access funding provided by Scuola Internazionale Superiore di Studi Avanzati - SISSA within the CRUI-CARE Agreement.

Data availability and access Code for the reduction of neural network is provided as part of the Smithers package. It is available at https://github.com/mathLab/Smithers. The CIFAR-10 and CIFAR-100 datasets can be downloaded from the official webpage: https://www.cs.toronto.edu/\protect\unhbox\voidb@x\penalty\@M\kriz/cifar.html. Restrictions are applied on the availability of the custom dataset, which was used under license from Electrolux Professional for the current study, and therefore is not publicly available.

Declarations

Ethical and informed consent for data used Fulfilled.

Competing interests No potential competing interest was reported by the authors.

References

1. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems 25:1097–1105. https://doi.org/10.1145/3065386
2. Elgendy M (2020) Deep Learning for Vision Systems. Simon and Schuster, New York
3. Liu L, Ouyang W, Wang X, Fieguth P, Chen J, Liu X, Pietikäinen M (2020) Deep learning for generic object detection: A survey. International journal of computer vision 128:261–318. https://doi.org/10.1007/s11263-019-01247-4
4. Young T, Hazarika D, Poria S, Cambria E (2018) Recent trends in deep learning based natural language processing. IEEE Computational intelligenCe magazine 13(3):55–75. https://doi.org/10.1109/MCI.2018.2840738
5. Khurana, D., Koli, A., Khatter, K., Singh, S.: Natural Language Processing: State of The Art, Current Trends and Challenges. Multimedia Tools and Applications 82 (2022). DOI: https://doi.org/10.1007/s11042-022-13428-4
6. Noda K, Arie H, Suga Y, Ogata T (2014) Multimodal integration learning of robot behavior using deep neural networks. Robotics and Autonomous Systems 62(6):721–736. https://doi.org/10.1016/j.robot.2014.03.003
7. Kiyokawa T, Katayama H, Tatsuta Y, Takamatsu J, Ogasawara T (2021) Robotic Waste Sorter With Agile Manipulation and Quickly Trainable Detector. IEEE Access 9:124616–124631. https://doi.org/10.1109/ACCESS.2021.3110795
8. Wali A, Alamgir Z, Karim S, Fawaz A, Ali MB, Adan M, Mujtaba M (2022) Generative adversarial networks for speech processing: A review. Computer Speech & Language 72:101308. https://doi.org/10.1016/j.csl.2021.101308

9. Yu, D., Deng, L.: Automatic Speech Recognition vol. 1. Springer, London (2016). https://doi.org/10.1007/978-1-4471-5779-3
10. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge, USA (2016). http://www.deeplearningbook.org
11. Khan A, Sohail A, Zahoora U, Qureshi AS (2020) A survey of the recent architectures of deep convolutional neural networks. Artificial Intelligence Review 53(8):5455–5516. https://doi.org/10.1007/s10462-020-09825-6
12. Trenn S (2008) Multilayer Perceptrons: Approximation Order and Necessary Number of Hidden Units. IEEE Transactions on Neural Networks 19(5):836–44. https://doi.org/10.1109/TNN.2007.912306
13. Wang, E., Davis, J.J., Zhao, R., Ng, H.-C., Niu, X., Luk, W., Cheung, P.Y.K., Constantinides, G.A.: Deep neural network approximation for custom hardware: Where we've been, where we're going. ACM Computing Surveys 52(2) (2019). https://doi.org/10.1145/3309551
14. Wuraola A, Patel N (2022) Resource efficient activation functions for neural network accelerators. Neurocomputing 482:163–185. https://doi.org/10.1016/j.neucom.2021.11.032
15. Huang J, Zhao J, Cai W (2019) Compressing convolutional neural networks using POD for the reconstruction of nonlinear tomographic absorption spectroscopy. Computer Physics Communications 241:33–39. https://doi.org/10.1016/j.cpc.2019.03.020
16. Messaoud S, Bouaafia S, Maraoui A, Ammari AC, Khriji L, Machhout M (2022) Deep convolutional neural networks-based hardware-software on-chip system for computer vision application. Computers & Electrical Engineering 98:107671. https://doi.org/10.1016/j.compeleceng.2021.107671
17. Udendhran R, Balamurugan M, Suresh A, Varatharajan R (2020) Enhancing image processing architecture using deep learning for embedded vision systems. Microprocessors and Microsystems 76:103094. https://doi.org/10.1016/j.micpro.2020.103094
18. da Silva ET, Sampaio F, da Silva LC, Medeiros DS, Correia GP (2020) A method for embedding a computer vision application into a wearable device. Microprocessors and Microsystems 76:103086. https://doi.org/10.1016/j.micpro.2020.103086
19. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: 2017 IEEE International Conference on Computer Vision (ICCV), pp. 1398–1406 (2017). https://doi.org/10.1109/ICCV.2017.155
20. Chen S, Zhao Q (2019) Shallowing deep networks: Layer-wise pruning based on feature representations. IEEE Transactions on Pattern Analysis and Machine Intelligence 41(12):3048–3056. https://doi.org/10.1109/TPAMI.2018.2874634
21. Li, Y., Adamczewski, K., Li, W., Gu, S., Timofte, R., Van Gool, L.: Revisiting random channel pruning for neural network compression. In: 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 191–201 (2022). https://doi.org/10.1109/CVPR52688.2022.00029
22. Molchanov, P., Mallya, A., Tyree, S., Frosio, I., Kautz, J.: Importance estimation for neural network pruning. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 11256–11264 (2019). https://doi.org/10.1109/CVPR.2019.01152
23. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C.: Learning efficient convolutional networks through network slimming. In: 2017 IEEE International Conference on Computer Vision (ICCV), pp. 2755–2763 (2017). https://doi.org/10.1109/ICCV.2017.298
24. Cichocki, A., Lee, N., Oseledets, I., Phan, A.-H., Zhao, Q., Mandic, D.P.: Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions. Foundations and Trends® in Machine Learning 9(4-5), 249–429 (2016). https://doi.org/10.1561/2200000059
25. Cichocki, A., Phan, A.-H., Zhao, Q., Lee, N., Oseledets, I., Sugiyama, M., Mandic, D.P.: Tensor networks for dimensionality reduction and large-scale optimization: Part 2 applications and future perspectives. Foundations and Trends® in Machine Learning 9(6), 431–673 (2017). https://doi.org/10.1561/2200000067
26. Li, Y., Gu, S., Mayer, C., Van Gool, L., Timofte, R.: Group Sparsity: The Hinge Between Filter Pruning and Decomposition for Network Compression. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 8015–8024 (2020). https://doi.org/10.1109/CVPR42600.2020.00804
27. Li, Y., Gu, S., Van Gool, L., Timofte, R.: Learning Filter Basis for Convolutional Neural Network Compression. In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pp. 5622–5631 (2019). https://doi.org/10.1109/ICCV.2019.00572
28. Yang, J., Shen, X., Xing, J., Tian, X., Li, H., Deng, B., Huang, J., Hua, X.-s.: Quantization Networks. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 7300–7308 (2019). https://doi.org/10.1109/CVPR.2019.00748
29. Hubara I, Courbariaux M, Soudry D, El-Yaniv R, Bengio Y (2017) Quantized neural networks: Training neural networks with low precision weights and activations. The Journal of Machine Learning Research 18(1):6869–6898
30. Deng L, Jiao P, Pei J, Wu Z, Li G (2018) GXNOR-Net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework. Neural Networks 100:49–58. https://doi.org/10.1016/j.neunet.2018.01.010
31. Hinton, G., Vinyals, O., Dean, J.: Distilling the Knowledge in a Neural Network. In: NIPS Deep Learning and Representation Learning Workshop (2015)
32. Gou J, Yu B, Maybank SJ, Tao D (2021) Knowledge distillation: A survey. International Journal of Computer Vision 129(6):1789–1819. https://doi.org/10.1007/s11263-021-01453-z
33. Cho, J.H., Hariharan, B.: On the Efficacy of Knowledge Distillation. In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pp. 4793–4801 (2019). https://doi.org/10.1109/ICCV.2019.00489
34. Bang D, Lee J, Shim H (2021) Distilling from professors: Enhancing the knowledge distillation of teachers. Information Sciences 576:743–755. https://doi.org/10.1016/j.ins.2021.08.020
35. Cui C, Zhang K, Daulbaev T, Gusak J, Oseledets I, Zhang Z (2020) Active subspace of neural networks: Structural analysis and universal attacks. SIAM Journal on Mathematics of Data Science 2(4):1096–1122. https://doi.org/10.1137/19M1296070
36. Benner, P., Grivet-Talocia, S., Quarteroni, A., Rozza, G., Schilders, W., Silveira, L.M.: Model Order Reduction: Volume 1: System- and Data-Driven Methods and Algorithms. De Gruyter, Berlin, Boston (2021). https://doi.org/10.1515/9783110498967
37. Benner, P., Schilders, W., Grivet-Talocia, S., Quarteroni, A., Rozza, G., Miguel Silveira, L.: Model Order Reduction: Volume 2: Snapshot-Based Methods and Algorithms. De Gruyter, Berlin, Boston (2020). https://doi.org/10.1515/9783110671490
38. Benner, P., Schilders, W., Grivet-Talocia, S., Quarteroni, A., Rozza, G., Miguel Silveira, L.: Model Order Reduction: Volume 3: Applications. De Gruyter, Berlin, Boston (2020). https://doi.org/10.1515/9783110499001
39. Constantine, P.G.: Active Subspaces: Emerging Ideas for Dimension Reduction in Parameter Studies. SIAM Spotlights, vol. 2. SIAM, U.S. (2015). https://doi.org/10.1137/1.9781611973860
40. Constantine PG, Dow E, Wang Q (2014) Active Subspace Methods in Theory and Practice: Applications to Kriging Surfaces. SIAM Journal on Scientific Computing 36(4):1500–1524. https://doi.org/10.1137/130916138
41. Romor F, Tezzele M, Lario A, Rozza G (2022) Kernel-based active subspaces with application to computational fluid dynamics parametric problems using discontinuous Galerkin method. International Journal for Numerical Methods in Engineering 123(23):6000–6027. https://doi.org/10.1002/nme.7099

42. Zahm O, Constantine PG, Prieur C, Marzouk YM (2020) Gradient-based dimension reduction of multivariate vector-valued functions. SIAM Journal on Scientific Computing 42(1):534–558. https://doi.org/10.1137/18M1221837
43. Ahnert K, Abel M (2007) Numerical differentiation of experimental data: local versus global methods. Computer Physics Communications 177:764–774. https://doi.org/10.2514/6.2003-4213
44. Williams CK, Rasmussen CE (2006) Gaussian Processes for Machine Learning, vol 2. The MIT press, Cambridge, MA, USA
45. Mohamed, S., Rosca, M., Figurnov, M., Mnih, A.: Monte Carlo Gradient Estimation in Machine Learning. Journal of Machine Learning Research 21(1) (2020). https://doi.org/10.5555/3455716.3455848
46. Romor F, Tezzele M, Rozza G (2021) ATHENA: Advanced Techniques for High dimensional parameter spaces to Enhance Numerical Analysis. Software Impacts 10:100133. https://doi.org/10.1016/j.simpa.2021.100133
47. Hesthaven, J.S., Rozza, G., Stamm, B.: Certified Reduced Basis Methods for Parametrized Partial Differential Equations, 1st edn. Springer Briefs in Mathematics, p. 135. Springer, Switzerland (2015). https://doi.org/10.1007/978-3-319-22470-1. Springer
48. Bui-Thanh T, Damodaran M, Willcox K (2003) Proper orthogonal decomposition extensions for parametric applications in compressible aerodynamics. In: 21st AIAA Applied Aerodynamics Conference, p. 4213. https://doi.org/10.2514/6.2003-4213
49. Bui-Thanh T, Damodaran M, Willcox K (2004) Aerodynamic data reconstruction and inverse design using proper orthogonal decomposition. AIAA journal 42(8):1505–1516. https://doi.org/10.2514/1.2159
50. Rozza, G., Stabile, G., Ballarin, F.: Advanced Reduced Order Methods and Applications in Computational Fluid Dynamics. Society for Industrial and Applied Mathematics, Philadelphia, PA (2022). https://doi.org/10.1137/1.9781611977257
51. Xiu D, Karniadakis GE (2002) The Wiener-Askey polynomial chaos for stochastic differential equations. SIAM journal on scientific computing 24(2):619–644. https://doi.org/10.1137/S1064827501387826
52. Fine, T.L.: Feedforward Neural Network Methodology. Information Science and Statistics. Springer, New York (1999). https://doi.org/10.1007/b97705
53. Wiener N (1938) The Homogeneous Chaos. American Journal of Mathematics 60(4):897–936. https://doi.org/10.2307/2371268
54. Janya-Anurak, C.: Framework for Analysis and Identification of Nonlinear Distributed Parameter Systems Using Bayesian Uncertainty Quantification Based on Generalized Polynomial Chaos. Karlsruher Schriften zur Anthropomatik, vol. 31. KIT Scientific Publishing, Karlsruhe, Deutschland (2017). https://doi.org/10.5445/KSP/1000066940
55. Ghanem, R.G., Spanos, P.D.: Stochastic Finite Elements: a Spectral Approach. Springer, New York (1991). https://doi.org/10.1007/978-1-4612-3094-6
56. Askey, R., Wilson, J.A.: Some basic hypergeometric orthogonal polynomials that generalize Jacobi polynomials. Memoirs of the American Mathematical Society 54(319) (1985). https://doi.org/10.1090/memo/0319
57. Sudret B (2008) Global sensitivity analysis using polynomial chaos expansions. Reliability engineering & system safety 93(7):964–979. https://doi.org/10.1016/j.ress.2007.04.002
58. Cheng K, Lu Z (2018) Adaptive sparse polynomial chaos expansions for global sensitivity analysis based on support vector regression. Computers & Structures 194:86–96. https://doi.org/10.1016/j.compstruc.2017.09.002
59. Shaham U, Cloninger A, Coifman RR (2018) Provable approximation properties for deep neuralnetworks. Applied and Computational Harmonic Analysis 44(3):537–557. https://doi.org/10.1016/j.acha.2016.04.003
60. Zaki MJ, Meira W Jr (2020) Data Mining and Machine Learning: Fundamental Concepts and Algorithms. Cambridge University Press, U.K
61. Alzubaidi L, Zhang J, Humaidi AJ, Al-Dujaili A, Duan Y, Al-Shamma O, Santamaría J, Fadhel MA, Al-Amidie M, Farhan L (2021) Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. Journal of big Data 8(1):1–74. https://doi.org/10.1186/s40537-021-00444-8
62. Rojas, R.: The Backpropagation Algorithm. In: Neural Networks, pp. 149–182. Springer, Berlin, Heidelberg (1996). https://doi.org/10.1007/978-3-642-61068-4_7
63. Borza, D.L., Ileni, T.A., Marinescu, A.I., Darabant, S.A.: Teacher or supervisor? effective online knowledge distillation via guided collaborative learning. Computer Vision and Image Understanding, 103632 (2023). https://doi.org/10.1109/CVPR.2016.90
64. LeCun Y (1989) Generalization and network design strategies. Connectionism in perspective 19(143–155):18
65. Gu J, Wang Z, Kuen J, Ma L, Shahroudy A, Shuai B, Liu T, Wang X, Wang G, Cai J et al (2018) Recent advances in convolutional neural networks. Pattern recognition 77:354–377. https://doi.org/10.1016/j.patcog.2017.10.013
66. Simonyan, K., Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015)
67. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778. https://doi.org/10.1109/CVPR.2016.90
68. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Master's thesis, Department of Computer Science, University of Toronto (2009)
69. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d' Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc., new York, United States (2019)
70. Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, İ., Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., van Mulbregt, P., SciPy 1.0 Contributors: SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods 17, 261–272 (2020). https://doi.org/10.1038/s41592-019-0686-2
71. Ghashami M, Liberty E, Phillips JM, Woodruff DP (2016) Frequent Directions: Simple and Deterministic Matrix Sketching. SIAM Journal on Computing 45:1762–1792. https://doi.org/10.1137/15M1009718