SN Computer Science (2024) 5:54
https://doi.org/10.1007/s42979-023-02320-z

ORIGINAL RESEARCH

Evolutionary Optimization of Convolutional Extreme Learning Machine for Remaining Useful Life Prediction

Hyunho Mo[1,2] · **Giovanni Iacca**[1]

Received: 30 August 2022 / Accepted: 12 September 2023

Abstract

Remaining useful life (RUL) prediction is a key enabler for making optimal maintenance strategies. Data-driven approaches, especially employing neural networks (NNs) such as multi-layer perceptrons (MLPs) and convolutional neural networks (CNNs), have gained increasing attention in the field of RUL prediction. Most of the past research has mainly focused on minimizing the RUL prediction error by training NNs with back-propagation (BP), which in general requires an extensive computational effort. However, in practice, such BP-based NNs (BPNNs) may not be affordable in industrial contexts that normally seek to save cost by minimizing access to expensive computing infrastructures. Driven by this motivation, here, we propose: (1) to use a very fast learning scheme called extreme learning machine (ELM) for training two different kinds of feed-forward neural networks (FFNNs), namely a single-layer feed-forward neural network (SL-FFNN) and a Convolutional ELM (CELM); and (2) to optimize the architecture of those networks by applying evolutionary computation. More specifically, we employ a multi-objective optimization (MOO) technique to search for the best network architectures in terms of trade-off between RUL prediction error and number of trainable parameters, the latter being correlated with computational effort. In our experiments, we test our methods on a widely used benchmark dataset, the C-MAPSS, on which we search such trade-off solutions. Compared to other methods based on BPNNs, our methods outperform a MLP and show a similar level of performance to a CNN in terms of prediction error, while using a much smaller (up to two orders of magnitude) number of trainable parameters.

Keywords Evolutionary algorithm · Multi-objective optimization · Extreme learning machine · Remaining useful life · C-MAPSS

Introduction

Optimal decision-making for maintenance management is related to costs and reliability. Recently, predictions regarding the future state of industrial components have been actively used to develop efficient maintenance strategies. In particular, predicting the remaining useful life (RUL) plays a crucial role in the so-called predictive maintenance (PdM) [1]. This technology allows to conduct timely maintenance before failures occur, so that industry stakeholders can cut their losses caused by any unplanned downtime as well as improve the quality of products. A paradigmatic example of the importance of PdM is given by the airline industry, in which timely maintenance based on the RUL prediction of aircraft engines can have a large impact on aircraft operation, safety, and cost. PdM is thus an essential requirement for airlines, to avoid major disasters while minimizing the maintenance cost. As such, accurate RUL predictions can

This article is part of the topical collection "Evolutionary Computation and Applications" guest edited by Gisele Pappa, Eric Medvet, Bing Xue, Leslie Pérez Cáceres, Sebastien Verel, Juan Luis Jiménez Laredo, Iñaki Hidalgo, Kehinde O. Babaagba and Aniko Ekart.

✉ Giovanni Iacca
giovanni.iacca@unitn.it

Hyunho Mo
hyunho.mo@unitn.it; h.mo@erasmusmc.nl

[1] Department of Information Engineering and Computer Science, University of Trento, Trento, Italy

[2] Present Address: Department of Radiology & Nuclear Medicine, Erasmus MC, Rotterdam, The Netherlands

Published online: 02 December 2023

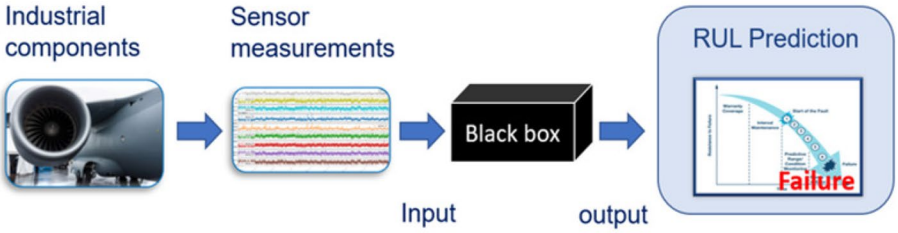

Fig. 1 Flowchart of a data-driven RUL prediction task

be a key solution to accomplish this requirement [2]. The existing works for RUL prediction can be mainly categorized into two approaches [3]: physics-based approaches and data-driven approaches. While the former require understanding the physics underlying degradation, the latter are capable of estimating the lifetime of components solely from past monitoring data. Today, the tremendous advances in machine learning (ML) and the advent of deep learning (DL) allow one to recognize complex patterns appearing in the data and to learn their relationship with the lifetime of the components. Thus, data-driven approaches, especially using ML methods, have gained increasing attention. Despite the increasing utility of ML-based approaches, physics-based approaches are still used especially when collecting monitoring or simulated data is difficult [4]. On the other hand, data-driven approaches are considered more useful when monitoring and/or simulated data are easier to acquire [5]. Figure 1 illustrates the flowchart of a data-driven RUL prediction task with a black-box ML model such as an artificial neural network (ANN). The sensors installed on a target industrial component measure physical properties related to the lifetime of the target. The collection of the sensor measurements over time can then be the health monitoring data, which typically take the form of multi-variate time-series. The following black-box model is responsible for providing the RUL prediction as its output from the current sensor measurements. The training of the model aims to minimize the training error, which is defined as the difference between the output and the actual RUL (i.e., the ground truth that is assumed to be known for a set of training samples), where historical data collected by run-to-failure operations are considered as the training data. Because the run-to-failure data are scarce and the model should perform well with limited data, developing an appropriate black-box model is a major challenge for the data-driven RUL prediction task. In this context, many ML and DL-based methods have been proposed over the past decade.

A multi-layer perceptron (MLP), discussed in [6], is one of the earliest works that use the traditional back-propagation neural networks (BPNNs) for RUL prediction. The authors also proposed employing a convolutional neural network (CNN) for estimating the RUL of aircraft engines. Later, a long short-term memory (LSTM) was used in [7] to directly recognize temporal patterns in the data instead of extracting convolutional features. Another work [8] introduced a stacked sparse autoencoder (SAE) and optimized its hyperparameters with a grid search. Focusing on the objective of minimizing the prediction error, the DL architectures for RUL prediction have become more and more complex. The authors of [9] proposed a directed acyclic graph (DAG) network that combines an LSTM with a CNN by placing them in parallel to improve the accuracy of the RUL prediction. A serial combination of an LSTM and a CNN was introduced in [10], where the LSTM is used to extract temporal features, while the CNN is used as a spatial feature extractor. The authors of [11] employed a multi-head CNN-LSTM combining a parallel branch of CNNs in series with an LSTM, and optimized its architecture based on evolutionary algorithm (EA). More recently, a combination of a convolutional autoencoder (CAE) with an LSTM has been proposed [12]. The authors of [13] presented an autoencoder-based deep belief network (AE-DBN) model to obtain accurate RUL predictions, in which, while the AE is used for extracting features, the DBN learns long-term dependencies. Most of the solutions presented so far are too focused on minimizing the RUL prediction error, which is typically achieved using deeper and more complex DL architectures. However, in practice, the recent DL models that place an overemphasis on prediction accuracy may not be affordable in industrial contexts that normally seek to save cost by minimizing access to expensive computing infrastructures. To be more specific, the recent DL architectures consist of a huge number of trainable parameters through many layers. Training a large number of parameters needs multiple iterations of computationally expensive gradient-based computations, which require having access to powerful hardware, such as a graphics processing unit (GPU). Moreover, it may happen that more monitoring data are collected after training the network. This requires continuous access to computing resources needed for retraining, which, in turn, increases costs. In this paper, we address the problem of reducing the computational cost of the model used for RUL prediction by employing the extreme learning machine (ELM) [14]. ELM is a fast learning algorithm for *single-layer feed-forward neural networks* (SL-FFNNs) for which learning consists in determining the network output weights by solving a regularized least-squares problem on top of a set of randomly initialized input weights. The

authors of [14] claim that ELMs provide the best generalization performance at an extremely fast learning speed. Moreover, studies in the existing literature have proven that the performance of ELMs is comparable, in terms of accuracy, to that of BPNNs on various classification tasks [15, 16]. Considering the advantages of ELMs, here, we develop an ELM-based model as an RUL prediction tool providing both good prediction accuracy and short learning time given by their lower number of trainable parameters, compared to BPNNs. This is explicitly beneficial when an RUL prediction model should be trained regularly on newly collected data. In this scenario, an ELM-based RUL prediction model could be quickly retrained on the up-to-date data even with rather constrained hardware resources, while a DL-based RUL prediction model would be comparably more costly and time-consuming to train, thus being more difficult to update frequently. Furthermore, in addition to the vanilla ELMs, we also consider a convolutional ELM (CELM) [17], which is obtained by combining a set of convolutional layers with random filters (whose weights are not updated during the feature extraction step) with a fully connected layer trained by an ELM. The weights on the nodes between the fully connected layer and the output layer are updated using the ELM learning scheme. Finally, we optimize the architecture of both ELMs and CELMs with respect to the two conflicting objectives of reducing the RUL prediction error while minimizing the number of trainable parameters, so as to find the best trade-off solutions in terms of prediction error vs. number of trainable parameters. To this end, we use a multi-objective optimization (MOO) technique, namely the well-known non-dominated sorting genetic algorithm II (NSGA-II) [18]. To evaluate the proposed methods, we use the commercial modular aero-propulsion system simulation (C-MAPSS) dataset provided by NASA [19], which is the *de facto* standard benchmark for RUL prediction. On each of its four sub-datasets, we search for optimal ELM and CELM trading off the RUL prediction error and the number of trainable parameters, and compare their performance to BPNNs in terms of the two objectives. To summarize, the main contributions of this work can be highlighted as follows:

- We use an MOO algorithm to achieve a successful trade-off between RUL prediction error and number of trainable parameters.
- In two out of four sub-datasets, the obtained solutions outperform an MLP and a CNN in terms of prediction error as well as number or parameters.
- To the best of our knowledge, this is the first use case of CELM applied to RUL prediction (and, in particular, on the C-MAPSS dataset).

This paper extends our previous work [20] in which, for the first time, we tackled the data-driven RUL prediction task by means of ELMs optimized by means on evolutionary computation. The additional materials w.r.t. our previous work can be outlined as follows: first, we consider the MOO of CELMs and employ the optimized CELMs for the RUL prediction task (the section " Convolutional Extreme Learning Machine); second, both the ELMs and CELMs are evaluated on the very widely used C-MAPSS dataset and, in doing so, we compare our experimental results to the numerical results reported in the literature presenting different methods (the section "Experimental Results"). The rest of the paper is organized as follows: in the section "Background", the background concepts on vanilla ELMs and CELMs are introduced. The specifications of the parameters to be optimized and the optimization algorithm are presented in Section "Methods". Then, the experimental setups considered in our work are outlined in the Section "Experimental Setup". The section "Experimental Results" presents the experimental results and their analysis. Finally, the Section "Conclusions discusses the conclusions of this work.

Background

ELM is a fast learning scheme for SL-FFNNs. While training typical BPNNs is a time-consuming process, ELM can in fact train the networks very quickly by analytically determining the output weights after randomly initializing the input weights [14]. In this section, we explain the basic principles of the fast learning algorithm for SL-FFNNs, and describe the CELM model, which is an extension of this ELM learning scheme to CNNs.

Extreme Learning Machine

When the output of SL-FFNNs can be represented as a linear system, training an ELM consists in finding the smallest norm least-squares solution of that linear system by computing the Moore–Penrose inverse. The idea behind the ELM model is that in FFNNs the input weights may not need to be adjusted at all [21, 22]. For training an SL-FFNN comprising L hidden neurons with N labeled training samples, an ELM can be formally described as follows. Each training sample is made up of a d-dimensional input vector with a corresponding c-dimensional label. Then, a given set of N training samples is written as $(\boldsymbol{x}_i, \boldsymbol{t}_i), i \in [1, N]$ with $\boldsymbol{x}_i \in \mathbb{R}^d$ and $\boldsymbol{t}_i \in \mathbb{R}^c$, where $\boldsymbol{x}_i$ and $\boldsymbol{t}_i$ denote each input vector and its corresponding label respectively. Since the RUL prediction problem is a regression task, in our case, the label is a scalar real number, i.e., $c = 1$.

Figure 2 visualizes an SL-FFNN and the notation used for describing the ELM model. As shown in the figure, the SL-FFNN used in this work consists of d input nodes, a single output node, and a single hidden layer of L neurons with

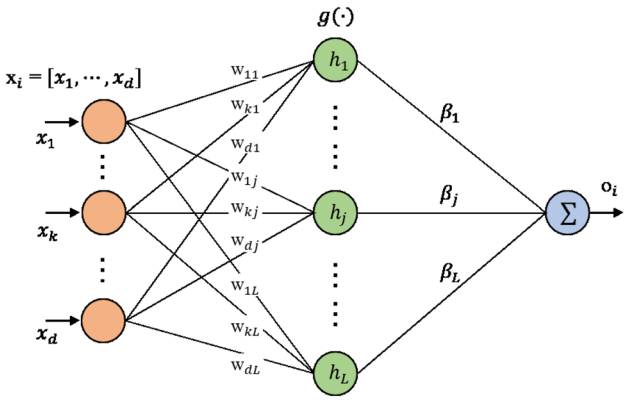

Fig. 2 Illustration of the ELM model based on an SL-FFNN

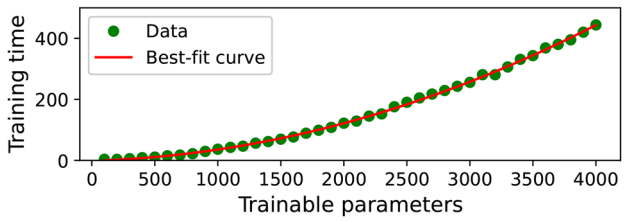

Fig. 3 Correlation between the number of trainable parameters and the training time of the ELM models. The best-fit curve shows a super-quadratic dependency between the training time and the number of trainable parameters, in line with the cubic complexity derived analytically, $\mathcal{O}(NL^2 + L^3)$

activation function $g(\cdot)$. For a given input sample $\boldsymbol{x}_i$, the output of a SL-FFNN o_i is defined by

$$o_i = \sum_{j=1}^{L} \beta_j g\left(\boldsymbol{w}_j \cdot \boldsymbol{x}_i + b_j\right), \tag{1}$$

where $\boldsymbol{w}_j = [w_{1j}, \ldots, w_{dj}]$ is a vector of weights associated to the connections between the d input neurons (which are assumed to be linear) and the jth hidden neuron, β_j is the weight on the connection between the jth hidden neuron and the output neuron, and b_j denotes the bias for the jth hidden neuron. The N equations computing the output for each training sample $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ can be written compactly as

$$\boldsymbol{H} \cdot \boldsymbol{\beta} = \begin{bmatrix} g\left(\boldsymbol{w}_1 \cdot \boldsymbol{x}_1 + b_1\right) & \cdots & g\left(\boldsymbol{w}_L \cdot \boldsymbol{x}_1 + b_L\right) \\ \vdots & \ddots & \vdots \\ g\left(\boldsymbol{w}_1 \cdot \boldsymbol{x}_N + b_1\right) & \cdots & g\left(\boldsymbol{w}_L \cdot \boldsymbol{x}_N + b_L\right) \end{bmatrix} \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_L \end{bmatrix}, \tag{2}$$

where $\boldsymbol{H}$ is the hidden layer output matrix (of size $N \times L$), and $\boldsymbol{\beta}$ (of size $L \times 1$) consists of the weights of all the connections between the hidden neurons and the output neuron. Let $\boldsymbol{T} = [\boldsymbol{t}_1, \ldots, \boldsymbol{t}_N]^\top$ be the target values, i.e., the labels for the N training samples. Then, the training of the SL-FFNN can be defined as minimizing the error between the network outputs and the target values. This is equivalent to find a least-squares solution $\hat{\boldsymbol{\beta}}$ to the linear system $\boldsymbol{H} \cdot \boldsymbol{\beta} = \boldsymbol{T}$. Thus, the training procedure can be mathematically formulated as

$$\left\| \boldsymbol{H}\hat{\boldsymbol{\beta}} - \boldsymbol{T} \right\| = \min_{\boldsymbol{\beta}} \|\boldsymbol{H}.\boldsymbol{\beta} - \boldsymbol{T}\|. \tag{3}$$

The weight vector minimizing the error $\hat{\boldsymbol{\beta}}$ is the smallest norm least-squares solution of the above equation

$$\hat{\boldsymbol{\beta}} = \boldsymbol{H}^{\dagger}\boldsymbol{T}, \tag{4}$$

where $\boldsymbol{H}^{\dagger}$ denotes the Moore–Penrose generalized inverse of the matrix $\boldsymbol{H}$. The inverse matrix is defined as $(\boldsymbol{H}^\top\boldsymbol{H})^{-1}\boldsymbol{H}^\top$. To prevent the inverse term $\boldsymbol{H}^\top\boldsymbol{H}$ from becoming singular, an L2 regularization term $\alpha\boldsymbol{I}$ can be added. As such, $\boldsymbol{H}^{\dagger}$ with the regularization term can be written as

$$\boldsymbol{H}^{\dagger} = \left(\boldsymbol{H}^\top\boldsymbol{H} + \alpha\boldsymbol{I}\right)^{-1}\boldsymbol{H}^\top, \tag{5}$$

where $\alpha \in \mathbb{R}$ is an arbitrarily small value. Overall, the ELM training algorithm for SL-FFNNs, which consists in finding a solution to Eq. (3), is defined by

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{H}^\top\boldsymbol{H} + \alpha\boldsymbol{I})^{-1}\boldsymbol{H}^\top\boldsymbol{T}. \tag{6}$$

Compared to BPNNs, the ELM training specified in Eq. (6) not only achieves significantly faster training speed, but also provides comparable performance in terms of prediction accuracy [16, 23]. Moreover, it has been shown that the generalization performance of ELMS is in general better than that provided by BP training algorithms [24, 25]. According to Eq. (6), the computational complexity of the ELM training algorithm is determined by the size of the matrix $\boldsymbol{H}$: more specifically, it is $\mathcal{O}(NL^2 + L^3)$. Namely, the complexity is cubic w.r.t. the number of trainable parameters, which is the same as L. The empiric characterization of the training time w.r.t. L is shown in Fig. 3.

When we design an ELM with an SL-FFNN, increasing the number of hidden neurons L can contribute to decreasing the prediction error, but the computational complexity (hence the training time) increases. Thus, finding the optimal value of L as well as the other parameters of an ELM is a crucial element affecting the prediction error. In this paper, we use evolutionary search to explore this parameter space, so that we obtain the solutions resulting in a trade-off between the prediction error and the number of trainable parameters.

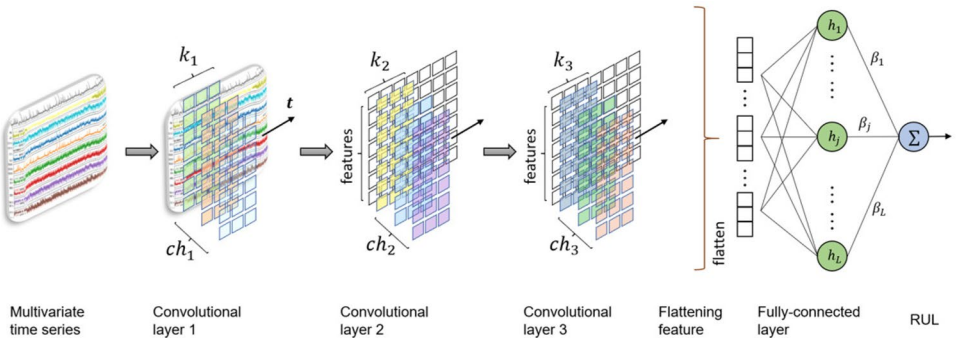

Fig. 4 Illustration of the CELM model based on CNNs

Convolutional Extreme Learning Machine

CNNs are a kind of multi-layer FFNNs that were first established by LeCun et al. [26] and applied to the classification task of two-dimensional (2D) images of handwritten digits. After years of research, CNNs have provided outstanding performances on several computer vision tasks [27, 28], and they have also achieved excellent results on regression tasks on one-dimensional (1D) time-series, based on convolutions with 1D filters [29, 30]. Figure 4 describes a CNN structure for a regression task, such as the RUL prediction, on multi-variate time-series. The network mainly comprises the feature extraction stage and the regression stage. The feature extraction stage includes a set of 1D convolutional layers aiming to extract high-level feature representations, while the following regression stage is a fully connected layer computing an output RUL value from the extracted features. BP algorithms are extensively used to train CNNs, but gradient-based learning algorithms are in general slower than required, because they tune all the parameters of the network iteratively. Considering that the slow speed of BP algorithms can be a major bottleneck in the applications of CNNs, here, we apply CELMs, which are fast training as they do not require an iterative gradient computation. Similar to the ELM model, based on an SL-FFNN, the training of CELMs consists in analytically determining the output weights on the connections between the hidden neurons of the fully connected layer and the output neuron, in conjunction with random initialization of all the remaining weights. In other words, the convolutional filters are randomly generated, and we randomly choose the input weights to the fully connected layer, as well. Thus, a major difference between the ELM and the CELM models considered in our work is the presence of the feature extraction step using the random filters.[1]

[1] We should note that, strictly speaking, "ELM" refers to the learning algorithm only. However, in the rest of this paper, both "ELM" and "CELM" will be generically used to refer to both the learning algorithm and the neural network itself.

Methods

We present now the details of the proposed methods: the section "Individual Encoding" describes the individual encoding. In the section "Individual Encoding", the evolutionary algorithm used for the MOO process is outlined.

Individual Encoding

Following what we have seen in the section "Extreme Learning Machine", both the RUL prediction error and the computational complexity (which correlates to the training time) are largely affected by the number of hidden neurons L. Furthermore, the performance of the ELMs varies with the choice of the activation function $g(\cdot)$ used in the hidden neurons. Here, we consider two widely used activation functions, namely the hyperbolic tangent *tanh* and the sigmoid *sigm*, as our $g(\cdot)$ for the ELMs. In addition, the L2 regularization parameter α described in the section "Background" is also optimized by the evolutionary algorithm. Based on the above discussion, we optimize the following integer parameters:

- n_{tanh}, number of hidden neurons with hyperbolic tangent activation;
- n_{sigm}, number of hidden neurons with sigmoid activation;
- r, L2 regularization parameter.

Hence, a genotype consists of three parameters: n_{tanh}, n_{sigm}, and r.

The lower and upper bounds (chosen empirically) for each parameter in the ELMs are set, as shown in Table 1. Regarding the parameters for the number of hidden neurons,

Table 1 Bounds of the parameters of the ELMs

Parameter	Min	Max
n_{tanh}	1	200
n_{sigm}	1	200
r	2	6

these are multiplied by a fixed value of 10 when we generate an ELM instance (i.e., the phenotype), so that we use a discretization on the number of hidden nodes to reduce the search space yet allowing ELMs of up to 2000 tan$h$ hidden neurons and 2000 sigmoid hidden neurons. In other words, both $n_{\tan}h$ and n_{sigm} range in the interval [1, 200], while the corresponding number of hidden neurons can range between 10 and 2000 with a step size of 10. These values have been chosen empirically. In particular, the maximum value for L (given by the sum of the two kinds of neurons) is 4, 000, and this upper bound is determined to limit the size of the hidden layer output matrix $\boldsymbol{H}$, which is $N \times L$, while keeping its calculation affordable during the evolutionary search. Because the whole range of integers between 1 and 2000 would be too large to explore, we divide it by 10, to decrease the number of possible combinations for those two parameters: by doing so, we reduce the number of possible combinations from 4×10^6 to 4×10^4. The remaining parameter, r, refers to the order of magnitude of the L2 regularization parameter α described in the section "Background", i.e., $\alpha = 10^{-r}$. This value should be relatively small to avoid affecting the ELM performance, but large enough to prevent the inverse term in Eq. (4) from becoming singular; we find that such a lower and an upper bound correspond to $\alpha = 10^{-2}$ and $\alpha = 10^{-6}$, respectively. The above ELM optimization has two major limitations: (1) there are only few parameters determining the network performance; and (2) we noted that the ELMs tend to converge to a certain value of L, above which the performance does not make any meaningful improvement (this latter aspect will be further discussed in the section "Experimental Setup"). On the other hand, the CELM model involves many more hyperparameters that can largely affect both the prediction error and the total number of trainable parameters in the network. In fact, the performance of the ELMs relies on the configuration of the fully connected layer, whereas the performance of the CELMs is determined not only by the fully connected layer, but also by the architecture of the preceding convolutional layers. Regarding the optimization of the CELM model shown in Fig. 4, we consider the following architecture parameters:

- ch_1, number of filters in the first convolution layer;
- k_1, length of each filter of ch_1;
- ch_2, number of filters in the second convolution layer;
- k_2, length of each filter of ch_2;
- ch_3, number of filters in the third convolution layer;
- k_3, length of each filter of ch_3;
- L, number of hidden neurons in the fully connected layer.

Considering that also, in the CELM case, the architecture parameters are all integers, the genotype consists in this case of seven integer values. The first six are reserved for constructing the three convolutional layers, while the number of hidden neurons in the following fully connected layer is determined by the remaining integer value L. Because a good performance of the CELMs with stacked convolutional layers (in particular three) has been proved in previous works [31, 32], we fix the number of the convolutional layers to three. Instead, the parameters regarding the filters in each convolutional layer are encoded, so that each individual generated during the evolutionary search extracts different convolutional features. It should be noted that the parameters regarding the number of filters are multiplied by a fixed value of 10 when we generate the phenotype, while the parameters regarding the filter lengths are used as they are. The bounds (chosen empirically) for the seven parameters of the CELMs are shown in Table 2. Since we set the maximum value of L is 4, 000 for the ELMs, the same upper bound is considered also for the CELMs to conduct a fair comparison between the two models. As shown in Table 2, the bounds of the parameter regarding the number of hidden neurons L are set to [1, 80]; however, this integer is multiplied by a fixed value of 50 when each genotype is translated into corresponding phenotype. As explained in the case of the ELM optimization, the multiplicand is used to decrease the possible number of combinations determined by the bounds of the parameters: thanks to this discretization, we reduce the number of possible combinations in the search space from 2.56×10^{14} to 5×10^9. Concerning the activation function, we use the sigmoid function for all the nodes in the fully connect layer of the CELMs. This choice follows the existing works on CELMs [31, 33, 34], that all make use of the sigmoid function in the fully connected layer, and the recent study [34], where the authors tested different widely used activation functions, including the sigmoid and the hyperbolic tangent, and verified that the sigmoid can achieve the best prediction accuracy.

Table 2 Bounds of the parameters of the CELMs

Parameter	Min	Max
ch_1	1	20
k_1	1	20
ch_2	1	20
k_2	1	20
ch_3	1	20
k_3	1	20
L	1	80

Optimization Algorithm

To optimize the parameters of the ELMs and CELMs described in the section "Individual Encoding", we consider an MOO approach looking for the best trade-off solutions in terms of RUL prediction error vs. number of trainable parameters. In the

evaluation step of our evolutionary search, the fitness of each individual is calculated by generating the phenotype, an ELM or a CELM, respectively, associated with the corresponding genotype, which is a vector containing the parameters (three or seven, respectively, for the case of ELMs and CELMs), as explained in the previous section. Given that the benchmark dataset on which we evaluate our methods consists of a training set D_{train} and a test set D_{test}, the training set D_{train} is further divided into training purpose data, E_{train}, and validation purpose data, E_{val} (i.e., $D_{\text{train}} = E_{\text{train}} \cup E_{\text{val}}$). Hence, any individual appearing during the evolutionary search is first trained on E_{train}, and then, its fitness is calculated using E_{val}. More specifically, MOO is achieved using a very well-known evolutionary algorithm, NSGA-II [18]. At the beginning of the evolutionary run, a population of n_{pop} individuals is initialized at random. In the main loop of the algorithm, following the initial generation, an offspring population of equal size n_{pop} is generated by tournament selection, crossover, and mutation. The tournament selection primarily checks the dominance level of each individual in the population. The secondary criteria, crowding distance, are then used to promote individuals that lie in less crowded areas of the Pareto front. Regarding the crossover and the mutation, we consider one-point crossover, with crossover probability p_{cx} set to 0.5, and uniform mutation, with mutation probability p_{mut} set to 0.5. The probabilities have been chosen, such that, in most cases, individuals are produced by either mutation or crossover (exclusively), so as to avoid disruptive effects due to the combination of mutation and crossover that may lead to bad individuals. The new individuals generated by the above genetic operators are then put together with the parents. The combined population of the parents and the offspring is then sorted according to non-domination. Finally, the best non-dominated sets are inserted into the new population until no more sets can be taken. For the next non-dominated set, which would make the size of the new population larger than the fixed population size n_{pop}, only the individuals that have the largest crowding distance values are inserted into the remaining slots in the new population. After that, the next generation starts. This loop is terminated after a fixed number of generations n_{gen}, and the evolutionary algorithm returns the solutions on the Pareto front. n_{pop} and n_{gen} are set both to 20. We have empirically found that these values allow enough evaluations to observe an improvement w.r.t. the hypervolume (HV) spanned by the solutions on a Pareto front. We refer to this method based on NSGA-II based for ELM as MOO-ELM. Similarly, the method for the CELM is referred to as MOO-CELM.

Experimental Setup

The details of our experimentation for evaluating the proposed methods are presented in this section: first, we describe the C-MAPSS dataset in the section "Benchmark Dataset" and the evaluation metrics in the section "Evaluation Metrics". In the section "Back-Propagation Neural Networks (BPNNs)", we describe the BPNNs that are used for the comparison with our methods. Then, the computational setup and the data preparation steps are outlined in the section "Computational Setup and Data Preparation".

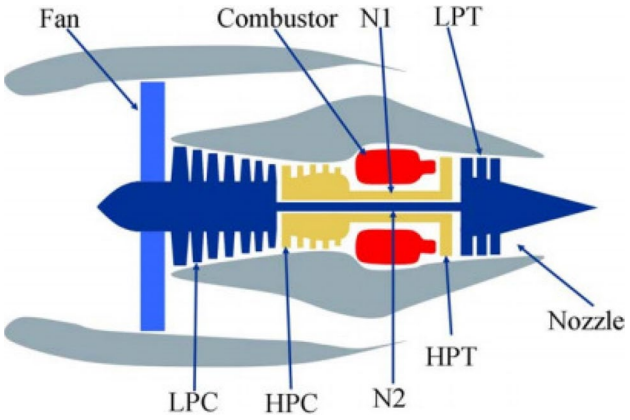

Fig. 5 Simplified diagram of the turbofan engine simulated in C-MAPSS [35]

Benchmark Dataset

As discussed in the section "Introduction", aircraft engine maintenance can have a significant impact on the total cost of operation in the aviation industry, and data-driven RUL prediction methods make possible to optimize maintenance. However, large representative run-to-failure data for developing the data-driven methods are unavailable in real operations, because failures during flight should not obviously occur in such safety-critical systems. To overcome this challenge, NASA introduced the C-MAPSS dataset, which consists of run-to-failure trajectories simulated by C-MAPSS. The C-MAPSS is a tool for simulating a realistic large commercial turbofan engine, as depicted in Fig. 5, which allows input variations of health-related parameters and recording of the resulting output sensor measurements. More specifically, the C-MAPSS simulator produces various sensor response surfaces as its outputs, based on a set of health-parameter inputs. This enables users to simulate the effects of faults in a set of rotating components which are crucial components associated with the engine's failure [19]: fan, low-pressure compressor (LPC), high-pressure compressor (HPC), high-pressure turbine (HPT), and low-pressure turbine (LPT). The outputs produced by the C-MAPSS simulation are used as input data for our experiments.

The C-MAPSS dataset contains trajectories from 21 different sensor measurements, generated under four different simulation settings. As outlined in Table 3, the dataset

Table 3 C-MAPSS dataset overview

Sub-dataset	FD001	FD002	FD003	FD004
Number of engines in training set	100	260	100	249
Number of engines in test set	100	259	100	248
Max/min cycles in training set	362/128	378/128	525/145	543/128
Max/min cycles in test set	303/31	367/21	475/38	486/19
Operating conditions	1	6	1	6
Fault modes	1	1	2	2

consists of four sub-datasets: FD001, FD002, FD003, and FD004, according to the operating states and fault mode. Each sub-dataset is split into a training set D_{train} and a test set D_{test}. The training set of each sub-dataset is made up of run-to-failure histories of different engines. In contrast, the simulation of each test engine is terminated before its failure, so that the RUL of each engine in the test set is required to be predicted for evaluating the prediction performance. One additional note is that the data of each engine consist of 21 multi-variate time-series, but 7 time-series that do not show changes over time are discarded. Thus, we use only 14 time-series as inputs. All the sensor readings and the RUL prediction are updated at the same frequency; the time unit for both the RUL prediction and the sensor measurements is referred to as *cycle*.

Evaluation Metrics

We evaluate our methods in terms of number of trainable parameters and prediction error. While, for the former, we merely count this number directly, the prediction error is quantified w.r.t. two metrics, namely the RMSE and the *s*-score [19], which are defined as follows. Let the error between the predicted and target RUL be $d_i = RUL_i^{\text{predicted}} - RUL_i^{\text{target}}$. The RMSE on D_{test} is defined as

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} d_i^2}, \tag{7}$$

where N is the total number of test samples in D_{test}. The *s*-score metric is also based on the error d_i, but it differentiates between "optimistic" and "pessimistic" predictions using an asymmetric function. Specifically, it assigns a larger value to optimistic RUL predictions w.r.t. pessimistic RUL ones. This reflects the risk of predicting an RUL value higher than the real one. As such, the *s*-score is formulated as follows:

Table 4 Compared RUL prediction methods for the C-MAPSS dataset

Method	Description
MLP [6]	1 hidden layer
CNN [6]	2 convolutional layers, 2 pooling layers, and 1 fully connected layer
LSTM [7]	2 LSTM layers and 2 fully connected layer

$$s\text{-score} = \sum_{i=1}^{N} SF_i, \quad SF = \begin{cases} e^{-\frac{d_i}{13}} - 1, & d_i < 0 \\ e^{\frac{d_i}{10}} - 1, & d_i \geq 0 \end{cases}. \tag{8}$$

It should be noted that we use the *s*-score solely for evaluating the methods on the test set; on the other hand, we perform the evolutionary optimization on the RMSE, since it provides more information from an optimization point of view w.r.t. the *s*-score. In fact, based on our previous work [36], networks optimized using the RMSE as fitness function provide better results in terms of both metrics, compared to networks optimized based on the *s*-score.

Back-Propagation Neural Networks (BPNNs)

As we discussed in the section "Introduction", data-driven methods such as traditional neural networks have been widely used in the field of RUL prediction. In particular, we consider for comparison three recent models introduced in [6] and [7], all based on BPNNs. For the evaluation purposes, these models are trained on D_{train} and their performance is evaluated by calculating the RMSE and the *s*-score on D_{test}. We specify the details of these compared methods in the following. The first two BPNNs are two FFNNs proposed in [6]: an MLP and a CNN. The architecture of the MLP comprises one hidden layer of 50 neurons. Regarding the CNN, the model consists of two pairs of convolutional layers and pooling layers, followed by a fully connected layer. The first convolutional layer has 8 filters of size 12, while the following convolutional layer contains 14 filters of size 4. Each pooling layer performs average pooling with size 1×2 to halve the feature length. The feature map is flattened at the end of the last pooling layer and passed to the fully connected layer of 50 neurons. Both the MLP and the CNN use a sigmoid as activation function. The third BPNN is an LSTM network introduced in [7]. The LSTM has four hidden layers: two stacked LSTM layers and two fully connected layers. The number of hidden units in each LSTM is 32, and the following two fully connected layers contain 8 neurons in

each layer. For all the BPNNs, the architecture is followed by a single output node. The architecture of the compared algorithms is summarized in Table 4. Regarding the training setup for the specified BPNNs, we followed the setup used in the papers introducing them. In particular, we used the *RMSprop* optimizer with a leaning rate 0.001 and set a mini-batch size to 512.

Computational Setup and Data Preparation

All the neural networks considered in our experimentation, including the proposed ELMs and CELMs, are implemented in Python. In particular, TensorFlow 2.3 is used to build the BPNNs described in the section "Back-Propagation Neural Networks (BPNNs)". To implement the ELMs, we use the high-performance toolbox for ELM (HP-ELM)[2] that supports GPU computation. The CELMs are implemented in PyTorch. All the experiments have been conducted on a workstation with an NVIDIA TITAN Xp GPU. The evolutionary optimization algorithm introduced in section "Individual Encoding" is implemented using the DEAP library.[3] Since we employ neural networks, each time-series is normalized in the range $[-1, 1]$ by min–max normalization. The methods employing an FFNN, i.e., the MLP and the ELM, take a 14-dimensional vector as an input for each timestamp. On the other hand, the networks that have a feature extraction step, i.e., the CNN, the LSTM and the CELM, require time-windowed data as input. To prepare the input for those networks, we apply a fixed-length time window with stride 1. For each sub-dataset, the length of the time window is the same as the minimum number of cycles in the test set, as described in Table 3. Finally, we split D_{train} in E_{train} and E_{val}: specifically, we randomly choose 80% of the data in D_{train} and assign them to E_{train}, used for training each individual. The remaining 20% are designated as E_{val} for the fitness evaluation.

Experimental Results

The aim of our experiments is to evaluate the optimized ELMs and CELMs discovered by the proposed methods, MOO-ELM and MOO-CELM. To perform a thorough evaluation, we compare them with the BPNNs described in the section "Back-Propagation Neural Networks (BPNNs)" not only in terms of number of trainable parameters, but also in terms of RUL prediction error, the latter being based on the two metrics defined in the section "Evaluation Metrics". For each parameter space (i.e., for the ELMs and CELMs) and each of the four C-MAPSS sub-datasets, we execute 10 independent runs of the MOO algorithm with different random seeds. The multiple runs are considered to enhance the reliability of the results obtained by the proposed methods based on NSGA-II. The convergence of each MOO run is analyzed by collecting the HV across 20 generations. After collecting the HV values, we normalize them to [0, 1] by min–max normalization, i.e., the maximum HV value is set to 1, while the minimum one is set to 0. As shown in Figs. 6 and 7, in all cases, the mean HV monotonically increases across the generations. This indicates that the used evolutionary search keeps successfully finding better non-dominated solutions across the generations. In addition to this gradual improvement, the slope of the mean HV and its standard deviation reveal convergence at the end of the generations. This implies that the algorithm explores the search space enough within 20 generations, regardless of which initial population it starts with.

When we consider the 10 independent runs for each setting, each run of the evolutionary search returns multiple solutions on the final Pareto front. Because NSGA-II tends to find "isolated" solutions considering how close the non-dominated solutions are to each other (due to the crowding distance sorting), the solutions widely spread across the two conflicting objectives. In practical applications, one should select from the Pareto front the solutions of interest based on domain knowledge or any other particular preference criteria. Here, to perform a comparative analysis with the other methods, we take a subset of the solutions by aggregating the 10 independent runs in the following way: first, we collect all the non-dominated solutions across the 10 runs. Then, we select a fraction of the solutions based on their density in the fitness space, as described in Figs. 8 and 9. When we do not have any preference for a particular objective, this strategy can be used to derive a subset of the solutions implicitly "preferred" by the MOO algorithm. As shown in the figures, we first place all the solutions from the 10 runs in the fitness space, which is discretized in equally spaced bins. The density of the solutions can then be measured by counting the number of solutions lying in each bin. As a result, we take the solutions from the bin with the highest density.

Based on this selection procedure, MOO-ELM finds 15, 31, 20, and 30 solutions on the four sub-datasets, respectively. In case of MOO-CELM, the numbers of selected solutions are 6, 4, 6, and 4. In each experiment, we calculate the test RMSE, the *s*-score, and the number of trainable parameters, for each of the available solutions. Their averages are then computed as the final results. For instance, the result of MOO-ELM on the FD001 can be derived as follows: the selected solutions are represented as 15 dots in the yellow-colored bin of Fig. 8a. For each dot, we train the corresponding ELM on D_{train} and calculate the test RMSE, the *s*-score on D_{test}, and the number of trainable parameters. After

[2] https://github.com/akusok/hpelm.

[3] https://github.com/DEAP/deap.

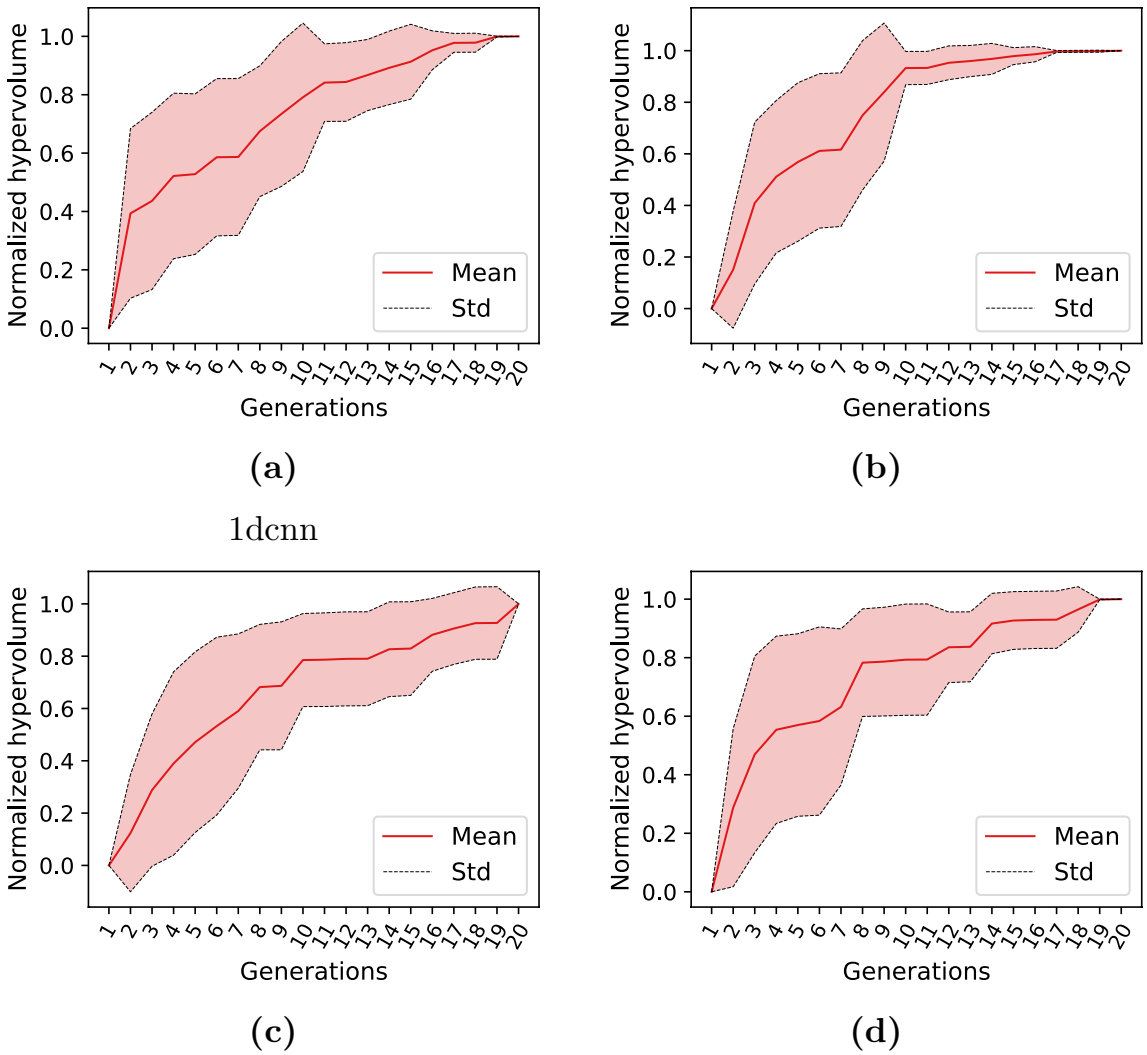

Fig. 6 Normalized hypervolume across generations (mean ± std. dev. across 10 independent runs) for the proposed MOO-ELM approach on the four sub-datasets, respectively: **a** FD001 dataset; **b** FD002 dataset; **c** FD003 dataset; **d** FD004 dataset

collecting these three values for the 15 solutions, we use the average test RMSE, the average test *s*-score, and the average number of trainable parameters to compare the proposed MOO-ELM with the other methods. The comparative results of all the considered methods are presented in Tables 5 and 6. In terms of test RMSE (Table 5), our methods are much better than the MLP, since we can obtain lower RMSE values with a smaller number of trainable parameters. Although both the proposed methods and the MLP use hundreds of parameters, our methods are considerably better in terms of computational cost and training time, because we apply an extremely fast ELM learning algorithm, while the MLP is trained by BP, which is relatively slow and expensive. The CNN has an even larger number of trainable parameters and achieves a much better test RMSE, compared to the MLP. Nevertheless, the test results of the proposed methods are still fairly comparable to those obtained by the CNN in terms of test RMSE, while our methods achieve these results using a much smaller number of trainable parameters. In particular, in terms of test RMSE, the results of MOO-ELM are slightly worse but very close to the results of the CNN for all four sub-datasets. MOO-CELM outperforms the CNN in terms of test RMSE on FD001 and FD003, but it does not provide as good results on the remaining sub-datasets. This implies that MOO-ELM can achieve a sufficient and stable performance on all the datasets including FD002 and FD004 that contain the data of six working conditions, while MOO-CELM is only advantageous for the RUL prediction task on less-complicated datasets such as FD001 and FD003 collected under only one condition. Additionally, we compare our results to the LSTM; as shown in Table 5, it is the best method in terms of test RMSE, but the number of trainable parameters in the LSTM is more than two orders of magnitude larger. In fact, the solutions given by MOO-CELM

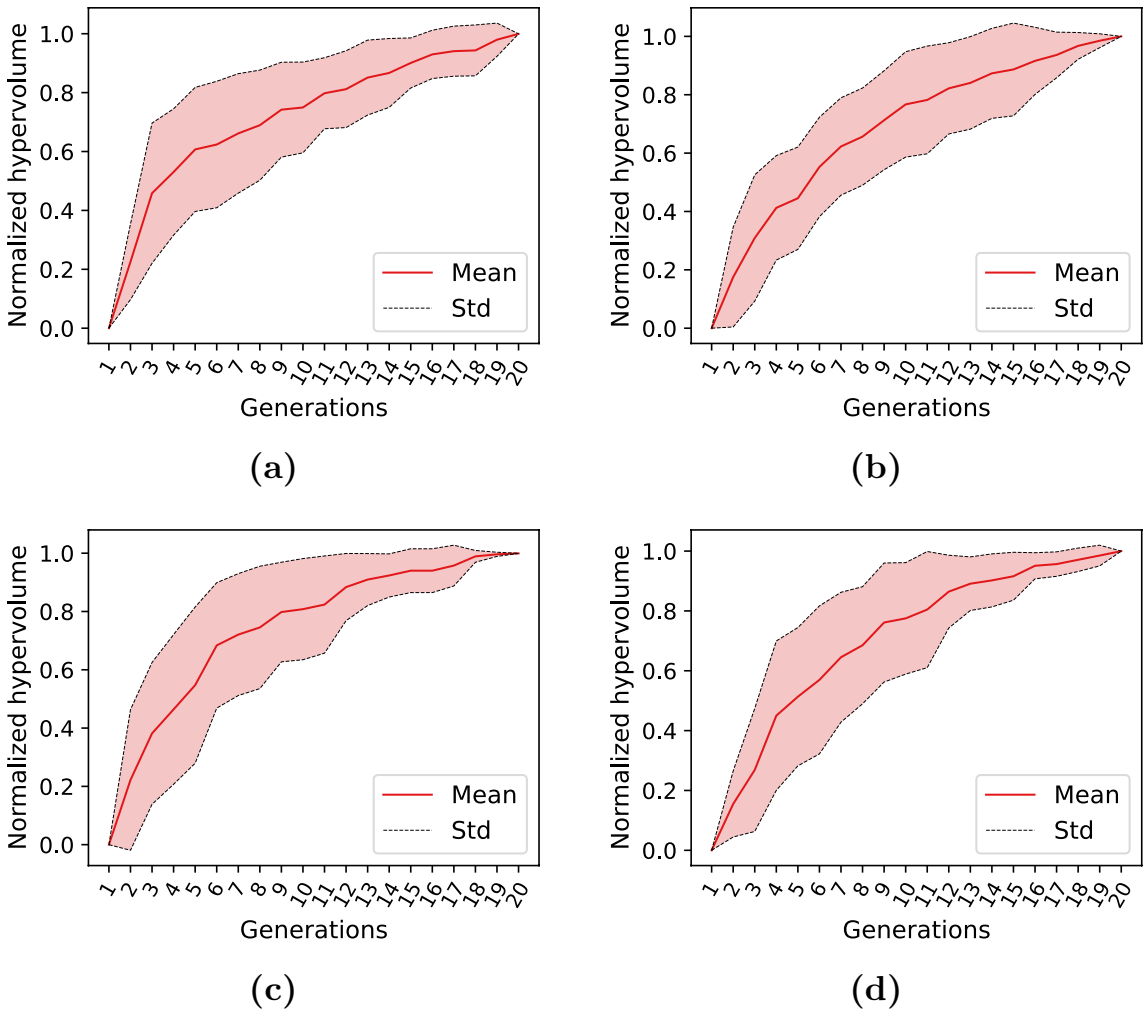

Fig. 7 Normalized hypervolume across generations (mean ± std. dev. across 10 independent runs) for the proposed MOO-CELM approach on the four sub-datasets, respectively: **a** FD001 dataset; **b** FD002 dataset; **c** FD003 dataset; **d** FD004 dataset

not only have a clear advantage in terms of number of trainable parameters but also can produce low prediction errors (comparable to those achieved by the LSTM) on the two less challenging datasets. Most of this analysis on the test RMSE is also valid for the *s*-score results summarized in Table 5. When we look at the score values in this table, those given by MOO-ELM are close to the results of the CNN. Moreover, MOO-CELM achieves the best score among the compared methods on FD003, as well as a good score (close to the score of the LSTM) on FD001.

Finally, the comparative results are visualized in Fig. 10, which easily allows to compare the performance of the different methods in terms of trade-off between the two conflicting objectives; due to the fact that the analysis of the score results is not much different from that carried out on the RMSE results, in the figure, we only illustrate the results w.r.t. the test RMSE, previously reported in Table 5. We can observe that none of the BPNNs dominates the others. Among the BPNNs, the MLP uses a lower number of trainable parameters, while the LSTM offers the best performance in terms of test RMSE. The CNN is placed between the MLP and the LSTM for both objectives. Note that the impact of the random initialization of the weights in NNs is not examined in our experiments (i.e., we assume that each architecture yields deterministic performance after its training), while we take into account 10 different random seeds for our evolutionary search. Each run of the GA with a different random seed produces indeed a different set of solutions. Therefore, having multiple runs of the proposed evolutionary algorithm allows us to verify its robustness over different random seeds. When we compare the two proposed methods with each other, adding convolutional layers with random filters to the SL-FFNN (i.e., using MOO-CELM instead of MOO-ELM) improves the RUL prediction for less-complicated data, as shown in Fig. 10a, c, although

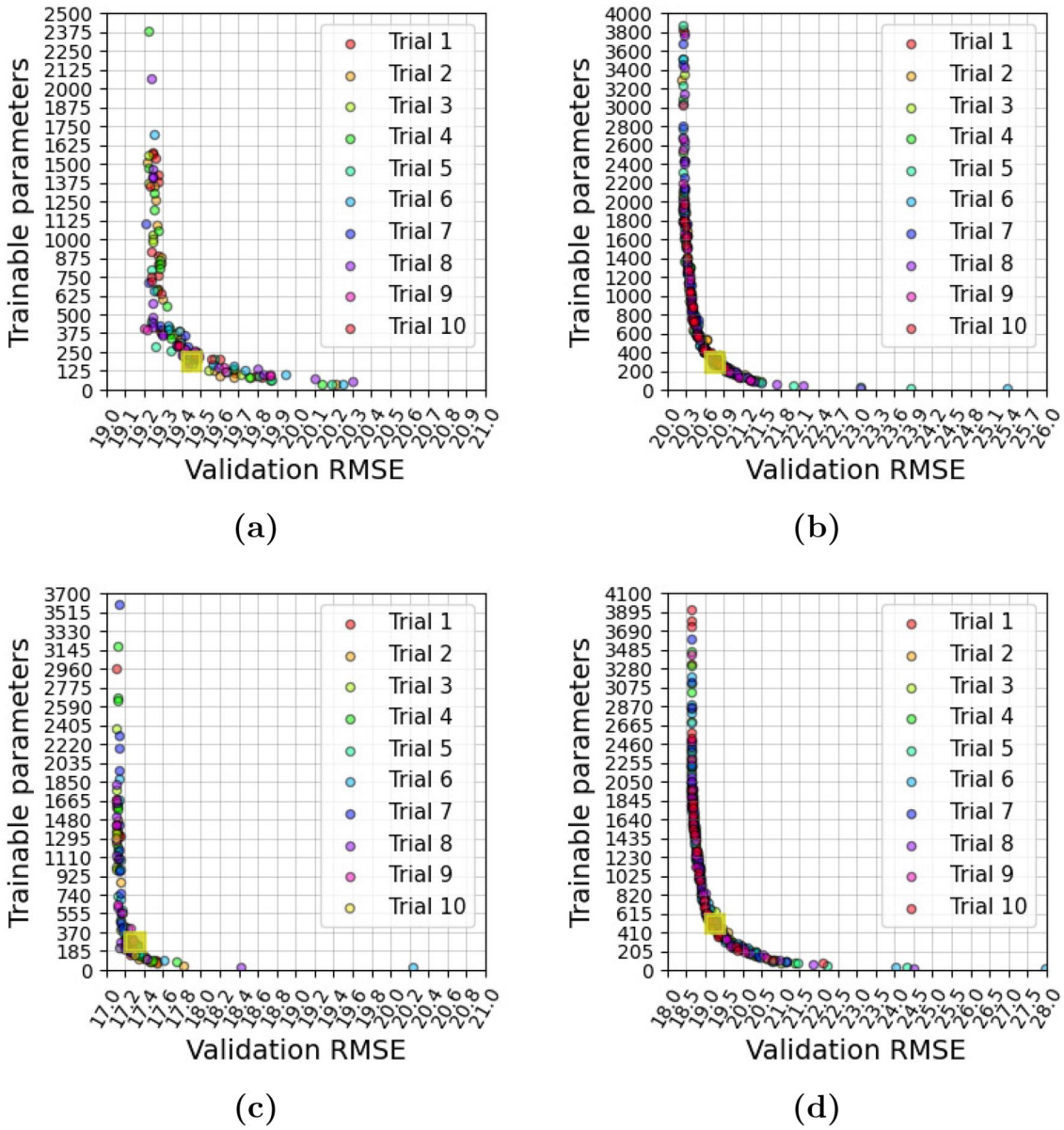

Fig. 8 Trade-off between validation RMSE and number of trainable parameters at the last generation for the 10 independent runs of the proposed MOO-ELM approach (aggregate results across runs). For further analysis, the fitness space is discretized in 20×20 bins. The bin highlighted in yellow corresponds to the one with the highest density of solutions: **a** FD001 dataset; **b** FD002 dataset; **c** FD003 dataset; **d** FD004 dataset

the number of trainable parameters slightly increases. In contrast, we find that introducing randomly generated convolutional filters can disturb the RUL prediction made by the ELMs on FD002 and FD004. The proposed methods, MOO-ELM and MOO-CELM, dominate the MLP for all the datasets except for FD004. In Fig. 10d, the number of trainable parameters of MOO-CELM is slightly larger than that of the MLP, but our method is still better in terms of computational cost, because it uses ELM, which is a much more efficient training algorithm compared to BP. In two out of four datasets (FD001 and FD003), the solutions discovered by MOO-CELM dominate the CNN. The results given by MOO-ELM show a comparable prediction performance while using a significantly lower number of trainable parameters, compared to the CNN. Although our proposed methods cannot outperform the LSTM in terms of test RMSE, they still can be good RUL prediction tools considering the much smaller number of trainable parameters and the advantages of the ELM training.

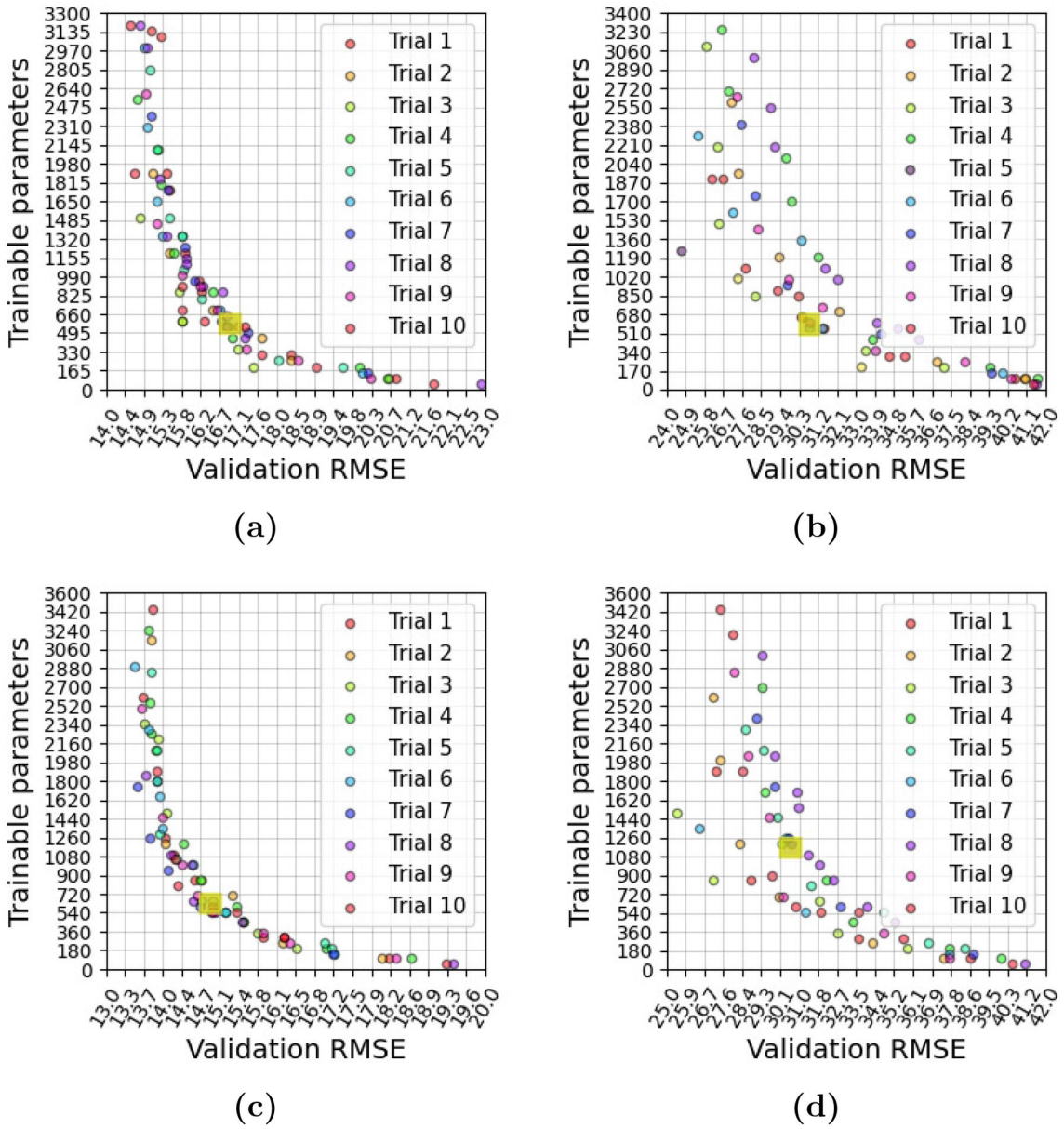

Fig. 9 Trade-off between validation RMSE and number of trainable parameters at the last generation for the 10 independent runs of the proposed MOO-CELM approach (aggregate results across runs). For further analysis, the fitness space is discretized in 20×20 bins. The bin highlighted in yellow corresponds to the one with the highest density of solutions: **a** FD001 dataset; **b** FD002 dataset; **c** FD003 dataset; **d** FD004 dataset

Conclusions

In this paper, we applied evolutionary computation to explore the parameter space of two different kinds of FFNNs trained with ELM. Our goal was to find a proper data-driven RUL prediction tool for industrial applications that implicitly require a good trade-off between prediction error and number of trainable parameters. To achieve this goal by automatically designing ELMs and CELMs, we used an MOO evolutionary algorithm, namely NSGA-II. The results obtained by the proposed methods were compared to those given by three BPNNs, namely an MLP, a CNN, and an LSTM network. The comparative evaluation was based on the C-MAPSS dataset, which is the *de facto* standard benchmark in the area of RUL prediction. We verified that the solutions discovered by our methods are clearly better than

Table 5 Summary of the comparative analysis based on the test results on C-MAPSS in terms of test RMSE and number of trainable parameters

Method	RMSE				Trainable parameters
	FD001	FD002	FD003	FD004	
MLP [6]	37.36±0.00	80.03±0.00	37.39±0.00	77.37±0.00	801
CNN [6]	18.45±0.00	30.29±0.00	19.82±0.00	29.16±0.00	6,815
LSTM [7]	**16.14±0.00**	**24.49±0.00**	**16.18±0.00**	**28.17±0.00**	14,681
MOO-ELM	18.93±0.19	30.46±0.12	20.56±0.15	31.70±0.19	**326**
MOO-CELM	16.54±0.57	39.98±0.35	17.97±0.80	42.62±0.78	751

For our methods, we report the values in terms of mean ± std. dev. across 10 independent runs. Note that the std. dev. of the trainable parameters is neglected, because it is relatively small. For the remaining methods, we report only one solution related to one single run (since their computations are deterministic)

The boldface indicates the best value per column

Table 6 Summary of the comparative analysis based on the test results on C-MAPSS in terms of *s*-score and number of trainable parameters

Method	*s*-score $\times(10^3)$				Trainable parameters
	FD001	FD002	FD003	FD004	
MLP [6]	18.00±0.00	7800.00±0.00	17.40±0.00	5620.00±0.00	801
CNN [6]	1.29±0.00	13.60±0.00	1.60±0.00	7.89±0.00	6,815
LSTM [7]	**0.34±0.00**	**4.45±0.00**	0.85±0.00	**5.55±0.00**	14,681
MOO-ELM	1.12±0.08	14.31±0.41	2.12±0.16	10.63±0.19	**326**
MOO-CELM	0.46±0.07	64.62±1.90	**0.64±0.15**	47.12±0.98	751

For our methods, we report the values in terms of mean ± std. dev. across 10 independent runs. Note that the std. dev. of the trainable parameters is neglected, because it is relatively small. For the remaining methods, we report only one solution related to one single run (since their computations are deterministic)

The boldface indicates the best value per column

the MLP. We also found that, in terms of test RMSE, MOO-ELM shows a similar performance w.r.t. the CNN for all the four sub-datasets, while MOO-CELM outperforms it on two out of four datasets. However, our solutions have smaller structures as well as can be trained quickly with ELM. Compared to the LSTM, our solutions perform slightly worse in terms of test RMSE, but the number of trainable parameters is much smaller, and this, in turn, implies a significantly shorter training time, thus a lower computational cost leveraging the advantages of ELM. One major limitation of our work is that the random filters cannot successfully extract convolutional features for ELM training on the more complicated data collected under different operating conditions (sub-datasets FD002 and FD0044). In future work, we can address this problem by employing predefined filters [37, 38] that can provide better prediction performance, instead of using random filters. Finally, in this work, we considered merely two different activation functions in neurons, but many other functions [39], such as rectified linear unit (ReLU) and leaky ReLU, can be used. Thus, one possible future direction would be to consider those additional functions when we define the search space.

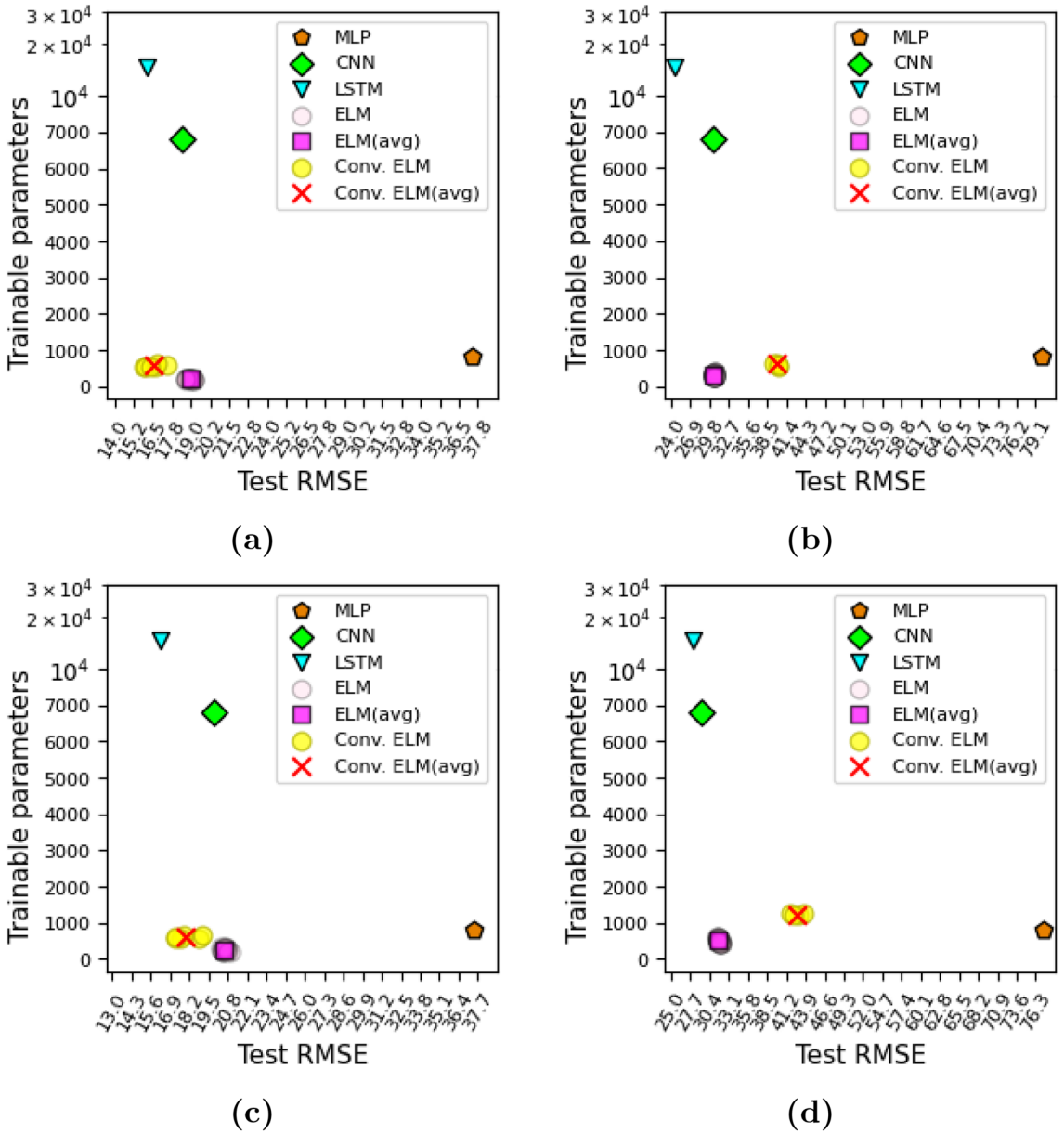

Fig. 10 Trade-off between test RMSE and number of trainable parameters for the methods considered in the experimentation. For MOO-ELM and MOO-CELM, we report the result of each of the 10 available runs, and their average. For the BPNNs, we report only one solution related to one single run (since their computations are deterministic): **a** FD001 dataset; **b** FD002 dataset; **c** FD003 dataset; **d** FD004 dataset

Author Contributions HM: conceptualization, investigation, software, data curation, visualization, and writing—original draft. GI: conceptualization, methodology, and writing–review and editing.

Funding Open access funding provided by Università degli Studi di Trento within the CRUI-CARE Agreement. Not applicable.

Availability of Data and Materials Not applicable.

Declarations

Conflicts of Interest The authors declare that they have no conflict of interest.

Code Availability The code is available at: https://github.com/mohyunho/NAS_convELM

References

1. Zhang W, Yang D, Wang H. Data-driven methods for predictive maintenance of industrial equipment: a survey. IEEE Syst J. 2019;13(3):2213–27.
2. Zheng C, Liu W, Chen B, Gao D, Cheng Y, Yang Y, Zhang X, Li S, Huang Z, Peng J. A data-driven approach for remaining useful life prediction of aircraft engines. In: International Conference on intelligent transportation systems (ITSC). 2018; pp. 184–189.
3. Atamuradov V, Medjaher K, Dersin P, Lamoureux B, Zerhouni N. Prognostics and health management for maintenance practitioners-review, implementation and tools evaluation. Int J Progn Health Manag. 2017;8(3):1–31.
4. Tinga T. Predicting critical failures using physics of failure: opportunities and challenges. In: AVT-356 Research Symposium on Physics of Failure for Military Platform Critical Subsystems, NATO Science & Technology Organization, 2021; pp. 1–13.
5. Fink O. Data-driven intelligent predictive maintenance of industrial assets. In: Women in industrial and systems engineering: key advances and perspectives on emerging topics. Cham: Springer; 2020. p. 589–605.
6. Sateesh Babu G, Zhao P, Li XL. Deep convolutional neural network based regression approach for estimation of remaining useful life. In: Database systems for advanced applications, 21st International conference, DASFAA 2016. Springer International Publishing; 2016. p. 214–28.
7. Zheng S, Ristovski K, Farahat A, Gupta C. Long short-term memory network for remaining useful life estimation. In: International Conference on prognostics and health management (ICPHM), IEEE, 2017; p. 88–95.
8. Ma J, Su H, Zhao WL, Liu B. Predicting the remaining useful life of an aircraft engine using a stacked sparse autoencoder with multilayer self-learning. Complexity. 2018;2018:1–13.
9. Li J, Li X, He D. A directed acyclic graph network combined with CNN and LSTM for remaining useful life prediction. IEEE Access. 2019;7:75464–75.
10. Remadna I, Terrissa SL, Zemouri R, Ayad S, Zerhouni N. Leveraging the power of the combination of CNN and bi-directional LSTM networks for aircraft engine RUL estimation. In: Prognostics and Health Management Conference (PHM-Besançon), IEEE, 2020; pp. 116–121.
11. Mo H, Custode LL, Iacca G. Evolutionary neural architecture search for remaining useful life prediction. Appl Soft Comput. 2021;108: 107474.
12. Ye Z, Yu J. Health condition monitoring of machines based on long short-term memory convolutional autoencoder. Appl Soft Comput. 2021;107: 107379.
13. Huthaifa AK, Nasser AR, Hasan AM, Al Mhdawi AK, Al-Raweshidy H, Humaidi AJ. Aircraft engines remaining useful life prediction based on a hybrid model of autoencoder and deep belief network. IEEE Access. 2022;10:82156–63
14. Huang GB, Zhu QY, Siew CK. Extreme learning machine: a new learning scheme of feedforward neural networks. In: International Joint Conference on Neural Networks (IJCNN). Volume 2., IEEE, 2004; pp. 985–990
15. Kim J, Shin HS, Shin K, Lee M. Robust algorithm for arrhythmia classification in ECG using extreme learning machine. Biomed Eng Online. 2009;8(1):1–12.
16. Kasun LLC, Zhou H, Huang GB, Vong CM. Representational learning with ELMs for big data. IEEE Intell Syst. 2013;28(6):31–34.
17. Dos Santos MM, da Silva Filho AG, dos Santos WP. Deep convolutional extreme learning machines: filters combination and error model validation. Neurocomputing. 2019;329:359–69.
18. Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput. 2002;6(2):182–97.
19. Saxena A, Goebel K, Simon D, Eklund N. Damage propagation modeling for aircraft engine run-to-failure simulation. In: International Conference on prognostics and health management (ICPHM), IEEE, 2008; pp. 1–9.
20. Mo H, Iacca G. Multi-objective optimization of extreme learning machine for remaining useful life prediction. In: International Conference on the applications of evolutionary computation (Part of EvoStar), Springer, 2022; pp. 191–206.
21. Huang GB, Zhu QY, Siew CK. Real-time learning capability of neural networks. IEEE Trans Neural Netw. 2006;17(4):863–78.
22. Huang GB, Chen L, Siew CK. Universal approximation using incremental feedforward networks with arbitrary input weights. IEEE Trans Neural Netw. 2006;17(4):879–92.
23. Popoola S, Misra S, Atayero PA. Outdoor path loss predictions based on extreme learning machine. Wirel Pers Commun. 2018;99:441–460.
24. Huang GB, Zhu QY, Mao K, Siew C, Saratchandran P, Sundararajan N. Can threshold networks be trained directly? IEEE Trans Circ Syst II Express Briefs. 2006;53:187–91.
25. Huang GB, Zhu QY, Siew CK. Extreme learning machine: theory and applications. Neurocomputing. 2006;70(1–3):489–501.
26. LeCun Y, Boser B, Denker J, Henderson D, Howard R, Hubbard W, Jackel L. Handwritten digit recognition with a back-propagation network. In: Advances in neural information processing systems. Vol. 2. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 1990. p. 396–404.
27. Hijazi S, Kumar R, Rowen C, et al. Using convolutional neural networks for image recognition, vol. 9. San Jose: Cadence Design Systems Inc.; 2015.
28. Chauhan R, Ghanshala KK, Joshi R. Convolutional neural network (CNN) for image detection and recognition. In: International Conference on secure cyber computing and communication (ICSCCC), IEEE, 2018; pp. 278–282.
29. Li X, Ding Q, Sun JQ. Remaining useful life estimation in prognostics using deep convolution neural networks. Reliab Eng Syst Saf. 2018;172:1–11.
30. Kiranyaz S, Ince T, Abdeljaber O, Avci O, Gabbouj M. 1-d convolutional neural networks for signal processing applications. In: International Conference on acoustics, speech and signal processing (ICASSP), IEEE, 2019; pp. 8360–8364.
31. Pang S, Yang X. Deep convolutional extreme learning machine and its application in handwritten digit classification. Comput Intell Neurosci. 2016;2016:1–10.
32. Duan M, Li K, Yang C, Li K. A hybrid deep learning CNN-ELM for age and gender classification. Neurocomputing. 2018;275:448–61.
33. Park Y, Yang HS. Convolutional neural network based on an extreme learning machine for image classification. Neurocomputing. 2019;339:66–76.

34. Cai Y, Zhang Z, Yan Q, Zhang D, Banu MJ. Densely connected convolutional extreme learning machine for hyperspectral image classification. Neurocomputing. 2021;434:21–32.
35. Frederick DK, DeCastro JA, Litt JS. User's guide for the commercial modular aero-propulsion system simulation (C-MAPSS). Technical report, NASA, TM-2007-215026. 2007.
36. Mo H, Custode LL, Iacca G. Evolutionary neural architecture search for remaining useful life prediction. Appl Soft Comput. 2021;108: 107474.
37. Li S, Jiang H, Pang W. Joint multiple fully connected convolutional neural network with extreme learning machine for hepatocellular carcinoma nuclei grading. Comput Biol Med. 2017;84:156–67.
38. Kannojia SP, Jaiswal G. Ensemble of hybrid CNN-ELM model for image classification. In: International Conference on signal processing and integrated networks (SPIN), IEEE, 2018, pp. 538–541.
39. Ratnawati DE, Marjono W, Anam S. Comparison of activation function on extreme learning machine (ELM) performance for classifying the active compound. In: AIP Conference Proceedings. Volume 2264., AIP Publishing LLC, 2020; p. 140001.